IBM WebSphere Commerce

Manual do Programador

Versão 54

G517-7452-00

IBM WebSphere Commerce

Manual do Programador

Versão 54

G517-7452-00

Nota:
Antes de utilizar estas informações e o produto suportado por elas, leia as informações na seção de Avisos.

Primeira Edição (Março de 2002), Release 2

Esta edição aplica-se aos seguintes produtos:

- IBM WebSphere Commerce Business Edition para Windows NT e Windows 2000, Versão 5.4 (Programa 5724-A18)
- IBM WebSphere Commerce Business Edition para AIX, Versão 5.4 (Programa 5724-A18)
- IBM WebSphere Commerce Business Edition para Software Solaris Operating Environment, Versão 5.4 (Programa 5724-A18)
- IBM WebSphere Commerce Business Edition para Linux, Versão 5.4 (Programa 5724-A18)
- IBM WebSphere Commerce Studio, Business Developer Edition para Windows NT e Windows 2000, Versão 5.4 (Programa 5724-A18)
- IBM WebSphere Commerce Professional Edition para Windows NT e Windows 2000, Versão 5.4 (Programa 5724-A18)
- IBM WebSphere Commerce Professional Edition para AIX, Versão 5.4 (Programa 5724-A18)
- IBM WebSphere Commerce Professional Edition para Sotware Solaris Operating Environment, Versão 5.4 (Programa 5724-A18)
- IBM WebSphere Commerce Professional Edition para Linux, Versão 5.4 (Programa 5724-A18)
- IBM WebSphere Commerce Studio, Professional Developer Edition para Windows NT e Windows 2000, Versão 5.4 (Programa 5724-A18)

e a todos os releases e modificações subseqüentes dos produtos listados acima, até que seja indicado de outra forma em novas edições. Certifique-se de utilizar a edição correta para o nível do produto.

Solicite publicações através de um representante autorizado IBM ou escritório da IBM de sua localidade. As publicações não são armazenadas no endereço fornecido a seguir.

A IBM agradece pelos seus comentários. Você pode enviar seus comentários sobre esta publicação através de um dos seguintes métodos:

1. Eletronicamente, para o seguinte endereço de e-mail:

   `torrcf@ca.ibm.com`

2. Por correio para o seguinte endereço:

   Centro Industrial IBM Brasil
   Centro de Traduções
   Caixa Postal 71
   Campinas, SP - Brasil
   CEP: 13001-970

Quando o Cliente envia seus comentários, concede direitos não-exclusivos à IBM para usá-los ou distribuí-los da maneira que considerar conveniente, sem que isso implique em qualquer compromisso ou obrigação para com o Cliente.

Antes de Começar

O *Manual do Programador do IBM WebSphere Commerce* fornece informações sobre o modelo de programação e arquitetura do WebSphere Commerce. Em particular, fornece detalhes sobre os seguintes tópicos:

- Interação de componentes
- Padrões de design
- Modelo de objeto persistente
- Controle de acesso
- Erro de tratamento e mensagens
- Implementação de comando
- Ferramentas de desenvolvimento
- Implementação de código personalizado

Além disso, este manual inclui os seguintes tutoriais:

Criando lógica de novos negócios
Este tutorial demonstra como criar novos comandos, beans de dados e beans corporativos seguindo o modelo de programação do WebSphere Commerce. Ele também demonstra como integrar a lógica dentro de uma loja existente e como implementar o código para um destino WebSphere Commerce Server.

Modificando e estendendo a lógica de negócios existente
Este tutorial está dividido em duas seções. A primeira seção demonstra como incluir lógica nova em um comando de controlador existente. A segunda demonstra como modificar um comando de tarefa existente e um bean de entidade do WebSphere Commerce. Ela também demonstra como integrar as modificações em uma loja existente e implementar o código em um WebSphere Commerce Server de destino.

Convenções Utilizadas neste Manual

Este manual utiliza as seguintes convenções de destaque:

Negrito indica comandos ou controles da GUI (interface gráfica com o usuário) como nomes de campos, botões ou opções de menu.

`Monoespaçado` indica exemplos de texto que você digita exatamente como mostrado, bem como caminhos de diretórios.

Itálico é utilizado para ênfase e para variáveis que você substitui com seus próprios valores.

Este ícone representa uma Dica — informações adicionais que ajudam a concluir uma tarefa.

Windows indica informações específicas aoWebSphere Commerce para Windows NT e Windows 2000.

AIX indica informações específicas aoWebSphere Commerce para AIX.

Solaris indica informações específicas ao WebSphere Commerce para o software do Ambiente Operacional Solaris™.

400 indica informações específicas ao WebSphere Commerce para o IBM @server iSeries 400 (anteriormente chamado de AS/400)

Linux indica informações específicas ao WebSphere Commerce para Linux.

DB2 indica informações específicas ao DB2 Universal Database

Oracle indica informações específicas ao Oracle®.

Professional indica informações específicas ao WebSphere Commerce Professional Edition.

Business indica informações específicas ao WebSphere Commerce Business Edition.

Conhecimentos Requeridos

Este manual deveria ser lido pelos Desenvolvedores de Armazenamento que precisam compreender como personalizar um aplicativo WebSphere Commerce. Os Desenvolvedores de Armazenamento que estão executando extensões programáticas deveriam ter conhecimentos nas seguintes áreas:

- Java
- Arquitetura do componente EJB (Enterprise JavaBeans)
- Tecnologia JavaServer Pages
- HTML
- Tecnologia de Banco de Dados

- VisualAge for Java, Enterprise Edition, Versão 3.5

Onde Localizar mais Informações

Este manual poderá ser atualizado no futuro. Verifique os seguintes sites do WebSphere Commerce na Web para atualizações:

Business

`http://www.ibm.com/software/webservers/commerce/wc_be/lit-tech-general.html`

Professional

`http://www.ibm.com/software/webservers/commerce/wc_pe/lit-tech-general.html`

As atualizações podem incluir informações novas, tutoriais adicionais ou atualizados e amostras de código relacionados aos tutoriais.

Índice

Parte 1. Conceitos e Arquitetura

Capítulo 1. Visão Geral

Componentes de Software do WebSphere Commerce

Antes de examinar como funciona o WebSphere Commerce Server, recomenda-se dar uma olhada na figura maior de componentes de software que estão relacionados ao processo de personalização. O diagrama a seguir mostra uma visão simplificada desses produtos de software:

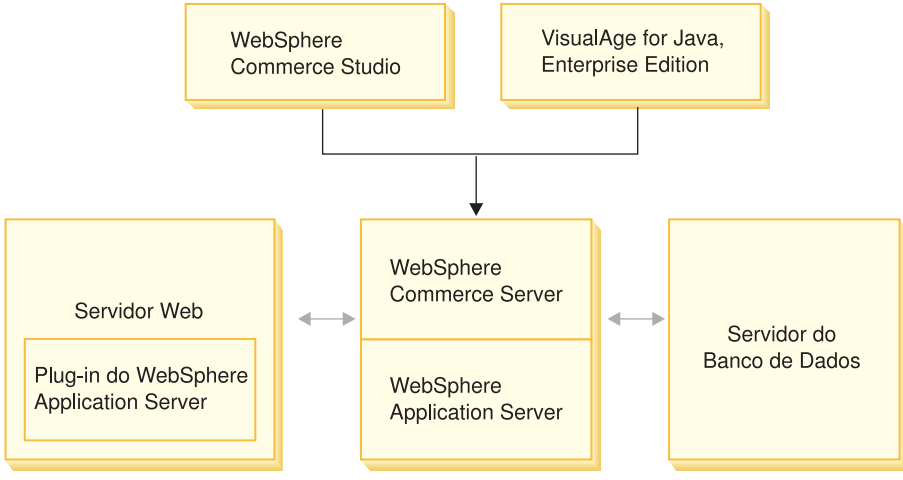


Figura 1.

O servidor Web é o primeiro ponto de contato de pedidos HTTP de entrada para o aplicativo e-commerce. Para criar uma interface eficiente com o WebSphere Application Server, ele utiliza o plug-in do WebSphere Application Server.

O WebSphere Commerce Server é executado no WebSphere Application Server, permitindo tirar proveito de muitos dos recursos do servidor de aplicativos. O servidor do banco de dados contém a maior parte dos dados do aplicativo, incluindo dados do produto e do comprador. Em geral, as extensões de seu aplicativo são feitas através da modificação ou extensão do código para o WebSphere Commerce Server. Além disso, poderá ser necessário armazenar dados que estejam fora da esfera do esquema de banco de dados do WebSphere Commerce dentro de seu banco de dados.

Os desenvolvedores utilizam duas ferramentas principais para criar lógica de negócios personalizada: o WebSphere Commerce Studio e o VisualAge for Java, Enterprise Edition. O WebSphere Commerce Studio é utilizado para criar

e gerenciar recursos de fachada da loja (por exemplo, modelos JSP). VisualAge for Java, Enterprise Edition é utilizado para criar nova lógica de negócios, em Java, que estende a funcionalidade existente ou cria funções completamente novas. Se o seu aplicativo requer extensões ao esquema do banco de dados, os desenvolvedores de banco de dados devem utilizar suas ferramentas de desenvolvimento de banco de dados preferidas para criar novas tabelas.

Arquitetura do Aplicativo WebSphere Commerce

Agora que você viu como os vários componentes de software relacionados à personalização se encaixam, é importante compreender a arquitetura do aplicativo. Isso ajudará a compreender quais partes são camadas de base e quais partes podem ser modificadas. O diagrama a seguir mostra as várias camadas que formam a arquitetura do aplicativo:

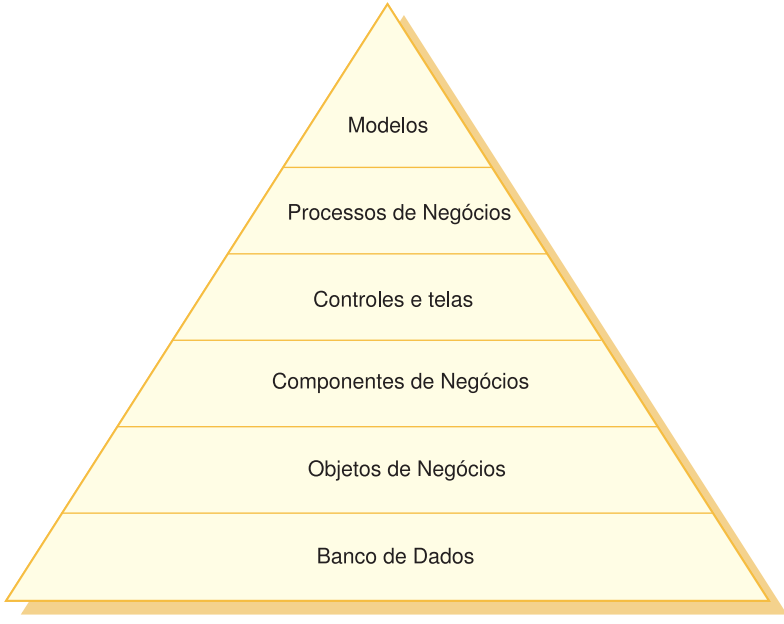


Figura 2.

Cada camada da arquitetura do aplicativo é descrita abaixo:

Banco de Dados

O WebSphere Commerce utiliza um esquema de banco de dados projetado especificamente para aplicativos de e-commerce e seus requisitos de dados. A seguir são fornecidos exemplos de tabelas nesse esquema:

- Usuário
- Pedido

- Produto

Objetos de Negócios

Os objetos de negócios representam entidades dentro do domínio de comércio e encapsulam a lógica centrada em dados necessária para extrair ou interpretar informações contidas no banco de dados. Essas entidades estão em conformidade com a especificação do Enterprise JavaBeans.

Essa entidade age como uma interface entre o aplicativo de comércio e o banco de dados. Além disso, os beans da entidade são mais fáceis de serem compreendidos do que os relacionamentos complexos entre as colunas em tabelas do banco de dados.

Componentes de Negócios

Os componentes de negócios são unidades da lógica de negócios. Eles executam procedimentos grosseiros de lógica de negócios. A lógica é implementada utilizando o modelo WebSphere Commerce dos comandos de controlador e comandos de tarefa. Um exemplo desse tipo de componente é o comando do controlador OrderProcess. Esse comando encapsula toda a lógica de negócios necessária para processar um pedido típico. O aplicativo de e-commerce chama o comando OrderProcess, que por sua vez, chama diversos comandos de tarefas para executar unidades de trabalho individuais. Por exemplo, comandos de tarefas individuais garantem que exista estoque suficiente disponível para atender aos requisitos do pedido, processar o pagamento, atualizar o status do pedido e quando o processamento for concluído, decrementar o estoque na quantidade apropriada.

Controles e Telas

Um controlador da Web determina a implementação do comando controlador apropriada e a visualização a ser utilizada. As implementações podem ser lojas específicas.
As exibições mostram os resultados dos comandos e ações do usuário. Elas são implementadas utilizando os modelos JSP. Alguns exemplos de exibições incluem ProductDisplay (retorna uma página de produtos mostrando informações relevantes sobre o produto selecionado pelo comprador) e OrderPrepare (apresenta ao comprador um formulário para que ele forneça informações apropriadas do pedido).

Processos de Negócios

Conjuntos de componentes de negócios e exibições criam processos de fluxo de trabalho e fluxo do site, conhecidos como processos de negócios. Alguns exemplos desses processos são:

Registro do usuário

Esse processo de negócios inclui os componentes de negócios

(por exemplo, o comando UserRegistrationAdd que cria um registro para um novo usuário) e exibições relacionadas a todas as etapas envolvidas no processo de registro de usuários.

Navegação no catálogo
Esse processo de negócios inclui os componentes de negócios (por exemplo, os comandos StoreCatalogDisplay e CategoryDisplay que, respectivamente, mostram os catálogos de uma loja e as categorias desse catálogo) e exibições relacionadas a todas as etapas envolvidas no processo de navegação por um catálogo.

Modelos
Quando juntas, as camadas inferiores do diagrama formam modelos comerciais do e-commerce. Um exemplo de um modelo de negócios e-commerce é o modelo de negócio-para-consumidor que é utilizado pelo exemplo de loja InFashion. Outro exemplo é o modelo business-to-business que é utilizado na loja de exemplo ToolTech.

Arquitetura do WebSphere Commerce runtime

A seção anterior introduziu a arquitetura do aplicativo, que apresenta as várias camadas do aplicativo WebSphere Commerce, a partir de um ponto de vista de aplicativo de negócios. Essa seção descreve como a arquitetura runtime é implementada.

Os componentes principais da arquitetura do WebSphere Commerce runtime são:

- Mecanismo de Servlet
- Ouvintes do protocolo
- Gerenciador de adaptadores
- Adaptadores
- Controlador da Web
- Comandos
- Beans da entidade
- Beans de dados
- Gerenciador de bean de dados
- Páginas de exibição
- Arquivos XML

As interações entre os componentes do WebSphere Commerce são mostradas no diagrama a seguir. Mais detalhes sobre cada componente podem ser encontrados nas seções subseqüentes.

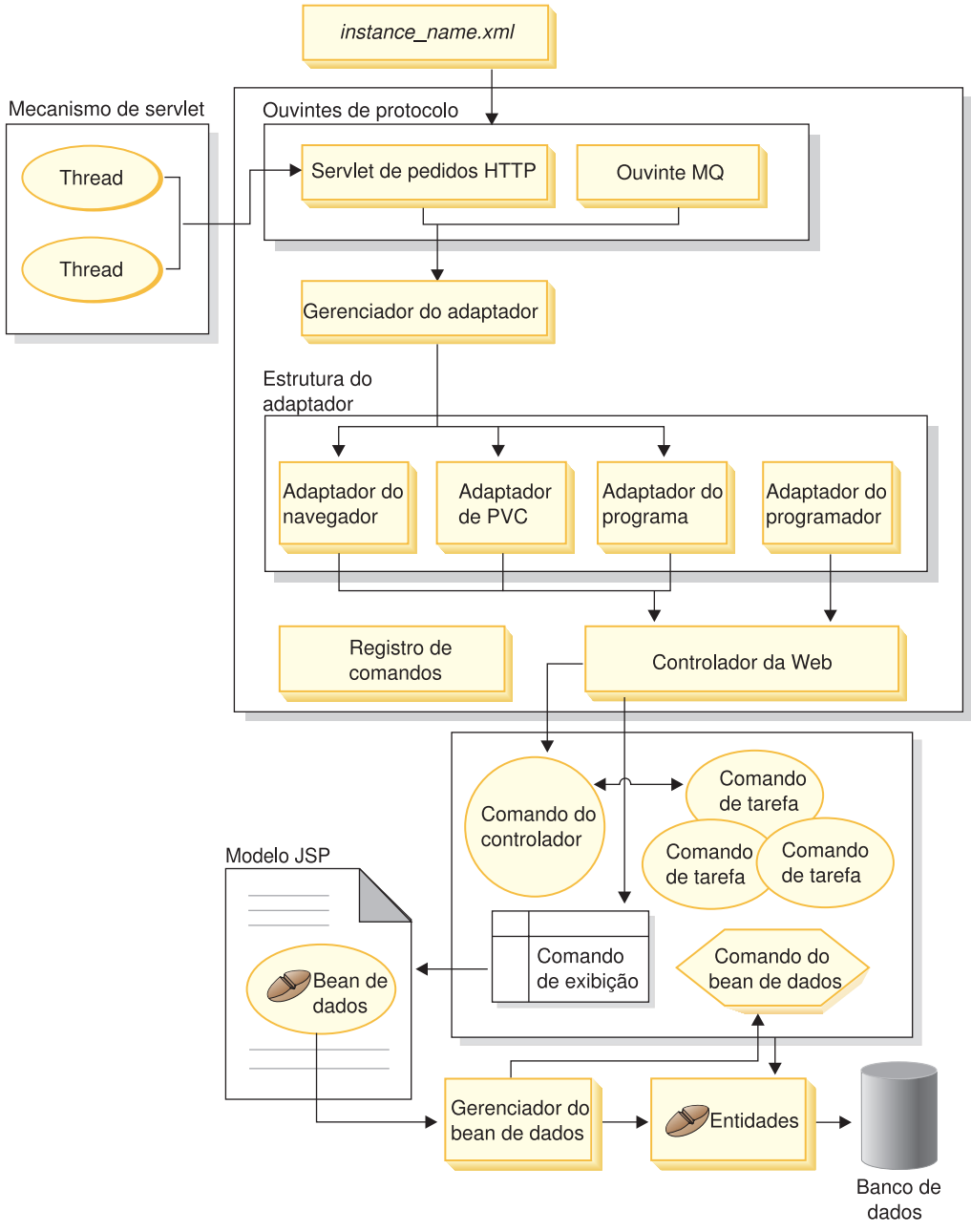

instance_name.xml
Mecanismo de servlet
Thread
Thread
Ouvintes de protocolo
Servlet de pedidos HTTP
Ouvinte MQ
Gerenciador do adaptador
Estrutura do adaptador
Adaptador do navegador
Adaptador de PVC
Adaptador do programa
Adaptador do programador
Registro de comandos
Controlador da Web
Comando do controlador
Comando de tarefa
Comando de tarefa
Comando de tarefa
Modelo JSP
Bean de dados
Comando de exibição
Comando do bean de dados
Gerenciador do bean de dados
Entidades
Banco de dados

Figura 3.

Mecanismo de Servlet

O mecanismo de servlet faz parte do ambiente WebSphere Application Server runtime que age como um dispatcher de pedidos URL de recepção. O mecanismo de servlet gerencia um conjunto de threads para manusear pedidos. Cada pedido de recepção é executado em um thread separado.

Versões anteriores do WebSphere Commerce utilizavam um servidor de aplicativos C++ que implementava seu próprio dispatcher de tarefas para pedidos de URL, mantendo um conjunto de processos do sistema pré-alocado. Esse modelo antigo, que utilizava processos de sistemas, possuía recursos mais intensivos que o novo modelo que utiliza threads Java. Explorando o mecanismo de servlet, a nova arquitetura do WebSphere Commerce runtime fornece uma solução comercial mais escalável.

Gerenciador do Adaptador

O gerenciador do adaptador determina qual adaptador é capaz de tratar da solicitação e em seguida, encaminha a solicitação a esse adaptador.

Ouvintes de Protocolo

Os comandos do WebSphere Commerce podem ser chamados a partir de vários dispositivos. Os exemplos de dispositivos que podem invocar comandos incluem:

- Navegadores típicos da Internet
- Telefones móveis que utilizam navegadores da Internet
- Aplicativos business-to-business que enviam mensagens XML utilizando o MQSeries
- Sistemas de abastecimento que enviam solicitações utilizando XML em HTTP
- O programador do WebSphere Commerce que executa um job em segundo plano

Os dispositivos podem utilizar uma variedade de protocolos de comunicação. Um ouvinte de protocolo é um componente runtime que recebe pedidos de recepção a partir de transportes e em seguida, envia os pedidos aos adaptadores apropriados, baseado no protocolo utilizado. Os ouvintes do protocolo incluem:

- Servlet de pedido
- Ouvinte do MQSeries

Quando o servlet de solicitação recebe uma solicitação de URL do mecanismo de servlet, ele passa a solicitação para o gerenciador de adaptadores. O gerenciador de adaptadores então consulta os tipos de adaptadores para determinar qual adaptador pode processar a solicitação. Após o adaptador específico ser determinado, o pedido é transmitido para ele.

Quando o servlet do pedido é inicializado, ele lê o arquivo de configuração *instance_name*.xml. Um dos blocos de configuração no arquivo XML define todos os adaptadores. O método init() do servlet de solicitação inicializa todos os adaptadores definidos.

O ouvinte do MQSeries recebe mensagens do MQSeries baseadas em XML de programas remotos e envia os pedidos para o gerenciador de adaptadores não-HTTP.

O Programador do Job não requer um ouvinte de protocolo.

Adaptadores

Os adaptadores do WebSphere Commerce são componentes específicos do dispositivo que executam funções de processamento antes de passar uma solicitação para o controlador da Web. Alguns exemplos de processamento de tarefas executadas por um adaptador incluem:

- Instrução do controlador da Web para processar o pedido de uma maneira específica para o tipo de dispositivo. Por exemplo, um adaptador de dispositivo PvC (pervasive computing, computação difusa) pode instruir o controlador da Web a ignorar a verificação do HTTPS no pedido original.
- Transformação do formato de mensagem da solicitação de recepção em um conjunto de propriedades que os comandos do WebSphere Commerce podem analisar.
- Fornecimento de persistência de sessão específica do dispositivo.

O diagrama a seguir mostra a hierarquia de classe de implementação para a estrutura do adaptador do WebSphere Commerce.

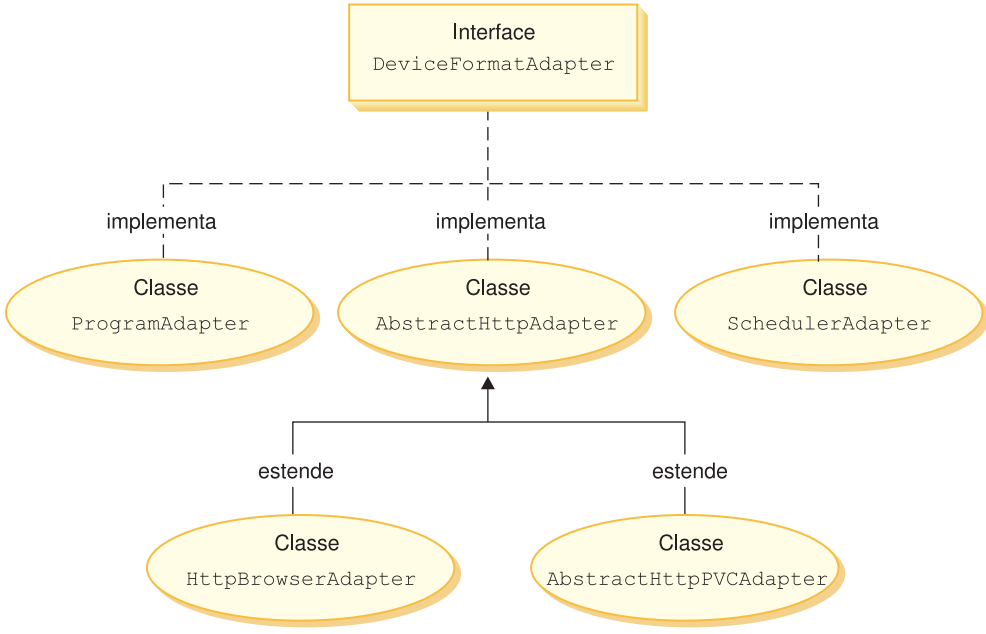


Figura 4.

Conforme exibido no diagrama anterior, todos os adaptadores implementam a interface DeviceFormatAdapter. A seguir, os adaptadores utilizados pelo ambiente WebSphere Commerce runtime:

Adaptador do programa
O adaptador do programa fornece suporte para programas remotos chamando os comandos do WebSphere Commerce. O adaptador de programa recebe solicitações e utiliza um mapeador de mensagens para converter a solicitação em um objeto CommandProperty. Depois da conversão, o adaptador de programa utiliza o objeto CommandProperty e executa a solicitação.

Adaptador do programador
O adaptador do programador fornece suporte para os comandos do WebSphere Commerce que são executados como jobs em segundo plano.

Adaptador do navegador HTTP
O adaptador do navegador HTTP fornece suporte de pedidos para invocar os comandos do WebSphere Commerce que são recebidos dos navegadores HTTP.

Adaptador de PvC HTTP
Trata-se de uma classe de adaptador abstrato que pode ser utilizado para desenvolver adaptadores de dispositivo PvC específicos. Por

exemplo, se fosse necessário desenvolver um adaptador para um determinado aplicativo de telefone celular, você estenderia a partir desse adaptador.

Se for necessário, a estrutura do adaptador pode ser estendida nas duas maneiras a seguir:

- Crie um adaptador para um dispositivo PvC específico (por exemplo, crie uma classe HttpIModePVCAdapterImpl para fornecer suporte aos dispositivos i-mode). Um adaptador desse tipo deve estender a classe AbstractHttpAdapterImpl.
- Crie um novo adaptador que se conecte a um novo ouvinte de protocolo. Esse novo adaptador deve implementar a interface DeviceFormatAdapter.

Controlador da Web

Um controlador da Web do WebSphere Commerce é um contêiner de aplicativos que segue um padrão de design parecido com o do contêiner de EJB. Esse contêiner simplifica as funções de comandos, oferecendo serviços tais como gerenciamento de sessão (baseado na persistência de sessão estabelecida pelo adaptador), controle de transação, controle de acesso e autenticação.

O controlador da Web também tem a função de reforçar o modelo de programação para o aplicativo de comércio. Por exemplo, o modelo de programação define os tipos de comandos que um aplicativo deve gravar. Cada tipo de comando tem uma finalidade específica. A lógica de negócios precisa ser implementada em comandos do controlador e a lógica de exibição precisa ser implementada em comandos de exibição. O controlador da Web espera que o comando do controlador retorne um nome de exibição. Se não for retornado um nome de exibição, será lançada uma exceção.

Para pedidos HTTP, o controlador da Web executa as seguintes tarefas:

- Começa a transação utilizando a interface UserTransaction a partir do pacote javax.transaction.
- Obtém dados de sessão do adaptador.
- Determina se o usuário deve efetuar logon antes de chamar o comando. Se necessário, direciona o navegador do usuário a uma URL de logon.
- Verifica se o HTTPS seguro é necessário para a URL. Se for necessário, mas a solicitação atual não estiver utilizando HTTPS, redireciona o navegador da Web para uma URL HTTPS.
- Invoca o comando do controlador e transmite-o ao contexto do comando e aos objetos de propriedades de entrada.
- Se ocorrer uma exceção de retrocesso da transação e o comando de controlador puder ser repetido, repete o comando de controlador.

- Um comando do controlador normalmente retorna um nome de exibição quando há um comando de exibição para ser retornado ao cliente. O controlador da Web invoca o comando de exibição para a exibição correspondente. Há várias maneiras de formar uma exibição de resposta. Por exemplo, redirecionar para uma URL diferente, encaminhar para um modelo JSP ou gravar um documento HTML no objeto de resposta.
- Salva os dados da sessão.
- Consolida os dados da sessão.
- Consolida a transação atual, caso seja bem-sucedida.
- Retorna a transação atual no caso de falha (dependendo das circunstâncias).

Comandos

Os comandos do WebSphere Commerce são beans que contêm a lógica de programação associada ao tratamento de um determinado pedido.

Há quatro tipos principais de comandos do WebSphere Commerce:

Comando do controlador

Um comando do controlador encapsula a lógica relacionada a um processo de negócios em particular. Exemplos de comandos do controlador incluem o comando *OrderProcessCmd* para processamento de pedido e o comando *UserRegistrationAddCmd* para a criação de novos usuários registrados. Em geral, um comando do controlador contém as instruções de controle (por exemplo, if, then, else) e invoca os comandos de tarefa para executar tarefas individuais no processo de negócios. Uma vez concluído, um comando do controlador retorna um nome de exibição. O controlador da Web então determina a classe de implementação apropriada para o comando de exibição e executa o comando de exibição.

Comando de tarefa

Um comando de tarefa implementa uma unidade específica de lógica de aplicativo. Em geral, um comando do controlador e um conjunto de comandos de tarefas juntos implementam a lógica do aplicativo para um pedido de URL. Um comando de tarefa é executado no mesmo contêiner que o comando do controlador.

Comando do bean de dados

Um comando de bean de dados é invocado por um modelo JSP quando o bean de dados é instanciado. A função principal de um comando do bean de dados é preencher o bean de dados com dados.

Comando de exibição

Um comando de exibição compõe uma exibição como uma resposta a um pedido de cliente. Há três tipos de comandos de exibição:

Comando de redirecionamento de exibição

Esse comando de exibição envia a exibição utilizando o

protocolo de redireção, tal como a redireção URL. Um comando do controlador deve retornar um comando de exibição nesse tipo de exibição para retornar uma exibição utilizando um protocolo de redirecionamento. Quando um protocolo de redireção é utilizado, ele altera as pilhas da URL no navegador. Quando uma tecla de recarregar é digitada, é executada a URL redirecionada, em vez da URL original.

Comando de direcionamento de exibição
Esse comando de exibição envia a exibição da resposta diretamente ao cliente.

Comando de encaminhamento de exibição
Esse comando de exibição encaminha o pedido de exibição a outro componente Web, como a um modelo JSP.

Há três maneiras em que um comando de exibição pode ser invocado:

- Um comando do controlador especifica um nome de comando de exibição após o pedido ser concluído com êxito.
- Um cliente solicita uma exibição diretamente.
- Um comando detecta um erro e uma tarefa de erro deve ser executada para processar o erro. O comando emite uma exceção com um nome de comando de exibição. Quando a exceção é propagada para o controlador da Web, executa o comando de exibição de erro e retorna a resposta ao cliente.

Beans de Entidade do WebSphere Commerce

Os beans de entidade são objetos comerciais transacionais, persistentes, fornecidos pelo WebSphere Commerce. Se você está familiarizado com o domínio de comércio, os beans de entidade representam os dados do WebSphere Commerce de uma forma intuitiva. Ou seja, em vez de ter de compreender todo o esquema do banco de dados, você pode acessar dados a partir de um bean de entidade que modelo conceitos e objetos mais precisamente no domínio de comércio. Você pode estender os beans de entidade já existentes. Além disso, para seus próprios requisitos de negócios específicos do aplicativo, é possível implementar beans de entidade completamente novos.

Os beans de entidade são implementados de acordo com o modelo do componente EJB (Enterprise JavaBeans).

Para obter mais informações sobre os beans de entidade, consulte “Implementação dos Beans de Entidade do WebSphere Commerce” na página 49.

Beans de Dados

Os beans de dados são beans Java utilizados principalmente pelos designers da Web. Mais comumente, eles fornecem acesso a uma entidade do WebSphere Commerce.Um designer da Web pode colocar esses beans em um modelo JSP, permitindo que as informações dinâmicas sejam colocadas na página no momento da exibição. O designer da Web precisa apenas saber quais dados o bean pode fornecer e quais requer como entrada. Consistente com o tema de separar exibição da lógica de negócios, não há necessidade que o designer da Web compreenda como os beans funcionam.

Gerenciador de Bean de Dados

Quando um bean de dados do WebSphere Commerce é inserido em um modelo JSP utilizando o WebSphere Studio Page Designer, é gerada uma linha de código que ocupa o bean de dados, no tempo de execução, chamando o gerenciador de bean de dados.

A seguir, um exemplo de código do Page Designer:

```
com.ibm.commerce.beans.DataBeanManager.activate(data_bean, request)
```

Modelos de JavaServer Pages

Os modelos JSP são servlets especializados utilizados normalmente para exibição. Após a conclusão de um pedido de URL, o controlador da Web invoca um comando de exibição que invoca um modelo JSP. Um cliente também pode invocar um modelo JSP diretamente do navegador sem um comando associado. Nesse caso, a URL para o modelo JSP deve incluir o servlet de solicitação em seu caminho, para que todos os beans de dados requeridos por um modelo JSP possam ser ativados em uma única transação. O servlet de solicitação pode encaminhar uma solicitação de URL para um modelo JSP e executar o modelo JSP em uma única transação.

O gerenciador de bean de dados rejeita qualquer URL para um modelo JSP que não inclua o servlet de solicitação em seu caminho. Para obter mais informações sobre a proteção de modelos JSP e outros recursos, consulte Capítulo 4, “Controle de Acesso” na página 91.

Arquivo de Configuração *Instance_name*.xml

O arquivo de configuração *Instance_name*.xml define informações de configuração para a instância. Ele é lido quando o servlet de solicitação é inicializado.

Resumo de um Pedido

Esta seção fornece um resumo do fluxo de interação entre componentes ao formar uma resposta a uma solicitação.

Uma descrição de cada uma das etapas segue o diagrama.

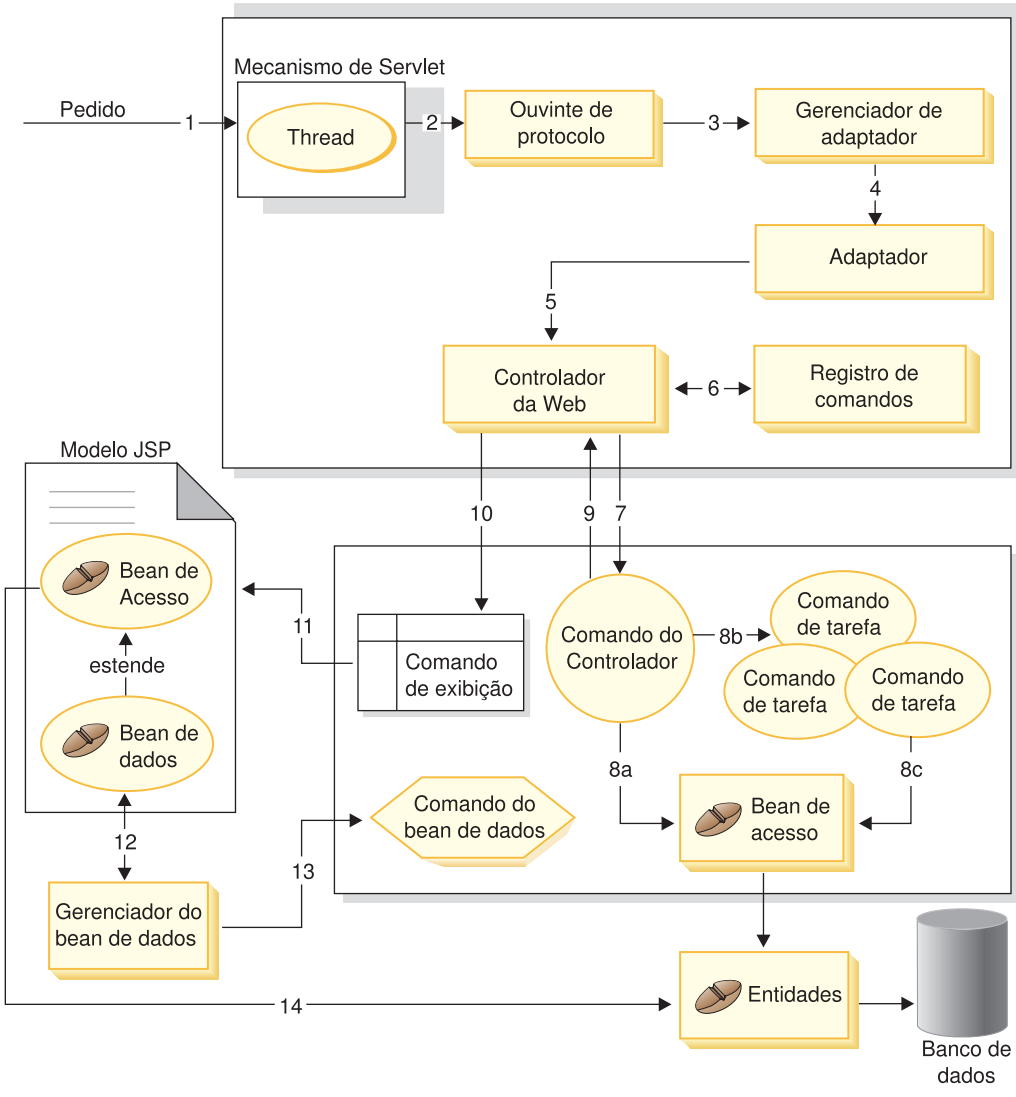


Figura 5.

As informações a seguir correspondem ao diagrama precedente.

1. O pedido é direcionado ao mecanismo de servlet pelo plug-in do WebSphere Application Server.
2. O pedido é executado em seu próprio thread. O mecanismo de servlet envia a solicitação a um ouvinte de protocolo. O ouvinte de protocolo pode ser o servlet de solicitação HTTP ou o Ouvinte MQ.

3. O ouvinte de protocolo passa a solicitação para o gerenciador de adaptadores.
4. O gerenciador de adaptador determina qual adaptador é capaz de tratar do pedido e em seguida, encaminha o pedido para o adaptador apropriado. Por exemplo, se o pedido tiver vindo de um navegador da Internet, o gerenciador de adaptadores encaminha o pedido para o adaptador de navegador da Internet.
5. O adaptador transmite o pedido para o controlador da Web.
6. O controlador da Web determina qual comando chamar, consultando o registro de comandos.
7. Assumindo que o pedido requer a utilização de um comando do controlador, o controlador da Web chama o comando do controlador apropriado.
8. Após o comando do controlador começar a execução, há poucos caminhos possíveis:
   a. O comando do controlador pode acessar o banco de dados utilizando um bean de acesso e seu bean de entidade correspondente.
   b. O comando do controlador pode invocar um ou mais comandos de tarefas. Em seguida, os comandos de tarefa podem acessar o banco de dados, utilizando os beans de acesso e seus beans de entidade correspondentes (exibido no 8(c)).
9. Uma vez concluído, o comando do controlador retorna um nome de exibição ao controlador da Web.
10. O controlador da Web procura o nome de exibição na tabela VIEWREG. Ele invoca a implementação do comando de exibição que está registrado para o tipo de dispositivo do solicitador.
11. O comando de exibição encaminha o pedido a um modelo JSP.
12. Dentro do modelo JSP, um bean de dados é exigido para recuperar as informações dinâmicas a partir do banco de dados. O gerenciador do bean de dados ativa o bean de dados.
13. O gerenciador do bean de dados invoca o comando bean de dados, se necessário.
14. O bean de acesso do qual o bean de dados é estendido acessa o banco de dados utilizando seu bean de entidade correspondente.

Diferenças Chave entre Personalização em Releases Anteriores

Iniciando com o WebSphere Commerce Suite Versão 5.1, o WebSphere Commerce Server foi gravado inteiramente em Java. Portanto, você deve utilizar Java para personalizar a funcionalidade. Isso é muito diferente do modelo que foi utilizado no WebSphere Commerce Suite, Versão 4.1 (e versões anteriores do Net.Commerce) em que as macros C++ e Net.Data foram utilizadas para personalização.

A tabela a seguir resume as principais alterações.

	Versão 4.1 (e versões anteriores) em C++	**Versão 5.1 (e versões anteriores) em Java**
Modelos de aplicativo	Comandos	Comandos do controlador
	Tarefas	Comandos de tarefas
	Funções substituíveis	Comandos de tarefas
	Tarefas de exibição	Comandos de exibição
	Tarefas de erro	Comandos de exibição
Modelos de exibição	Macros do Net.Data	Modelos JSP, beans de dados, comandos de bean de dados e comandos de exibição
Modelos de persistência	Net.Data e ODBC	Beans de entidade
Dispatcher URL	Dispatcher URL C++ do WebSphere Commerce	Mecanismo de servlet do WebSphere
Modelos de tarefas	Processos do sistema	Threads do Java
Adaptador de comando	Nenhum	Controlador e adaptadores da Web

Essa publicação não descreve o processo de migração. Para obter mais informações sobre migração, consulte o *WebSphere Commerce - Manual de Migração*.

Parte 2. Modelo de Programação

Capítulo 2. Padrões de Design

São utilizadas uma variedade de padrões de design e mecanismos para desenvolver a estrutura do WebSphere Commerce. O WebSphere Commerce fornece um padrão de design de alto nível que cada aplicativo WebSphere Commerce deve adotar. Os padrões de design a seguir são abordados nesse capítulo:

- O padrão de design model-view-controller
- O padrão de design do comando
- O padrão de design da exibição

Padrão de Design Model-View-Controller

O padrão de design MVC (model-view-controller - modelo-exibição-controlador) especifica que um aplicativo consiste em um modelo de dados, informações de apresentação e informações de controle. O padrão requer que cada um destes sejam separados em diferentes objetos.

O *modelo* (por exemplo, as informações de dados) contém apenas os dados puros do aplicativo. Ele não contém lógica que descreva como apresentar os dados a um usuário.

A *exibição* (por exemplo, as informações de apresentação) apresenta os dados do modelo para o usuário. A exibição sabe como acessar os dados do modelo, mas não sabe o que esses dados significam ou o que o usuário pode fazer para manipulá-los.

Finalmente, o *controlador* (por exemplo, as informações de controle) existe entre a exibição e o modelo. Ele atende aos eventos disparados pela exibição (ou outra origem externa) e executa a reação apropriada para estes eventos. Na maioria dos casos, a reação é chamar um método no modelo. Visto que a exibição e o modelo foram conectados por meio de um mecanismo de notificação, o resultado dessa ação será automaticamente refletido na exibição.

A maioria dos aplicativos de hoje em dia seguem este padrão, muitos com leves variações. Por exemplo, alguns aplicativos combinam a exibição e o controlador em uma classe porque já estão fortemente acopladas. Todas essas variações encorajam a separação de dados e sua apresentação. Isso não só torna a estrutura de um aplicativo mais simples, como também permite a reutilização do código.

Visto que há muitas publicações descrevendo o padrão, bem como várias amostras, esse documento não descreve o padrão com grandes detalhes.

O diagrama a seguir mostra como o padrão de design MVC se aplica ao WebSphere Commerce.

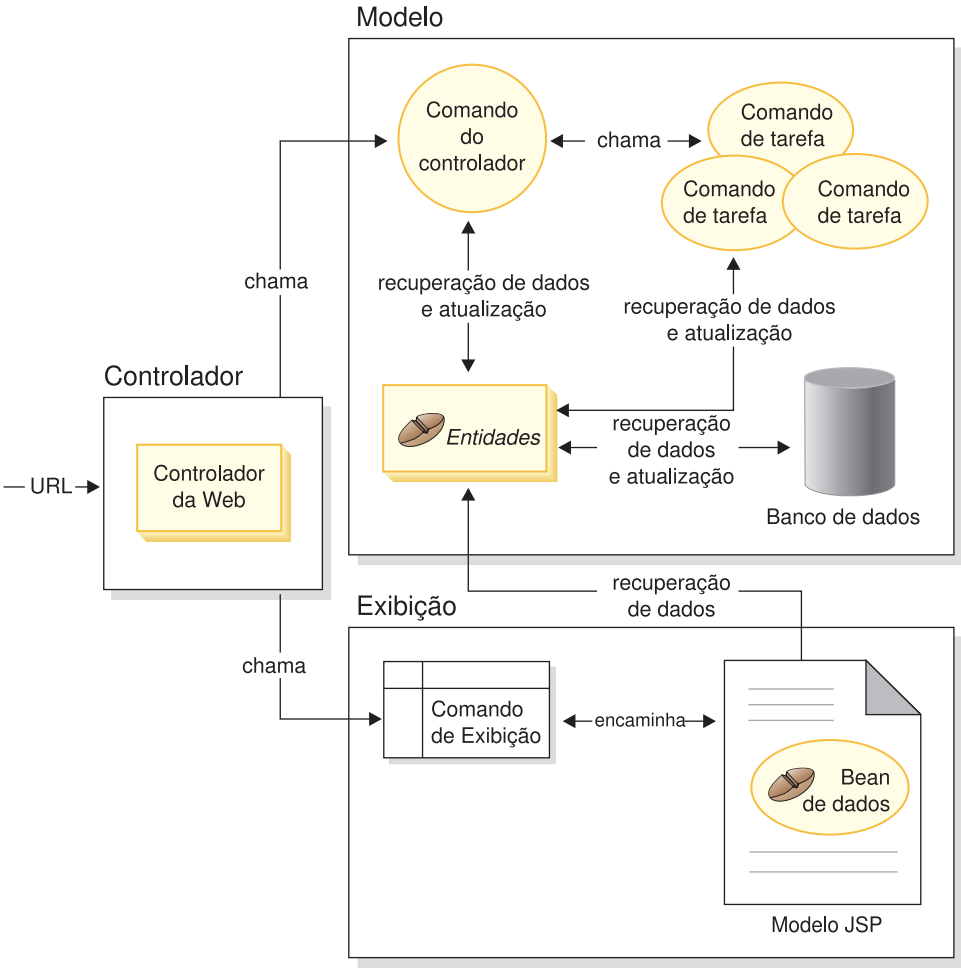


Figura 6.

Padrão de Design do Comando

O WebSphere Commerce Server aceita solicitações de aplicativos de Clientes Thin baseados em navegador, bem como de outros aplicativos remotos. Por exemplo, uma solicitação pode originar em um sistema de abastecimento remoto ou em outro servidor de comércio.

Todos os pedidos, em sua variedade de formatos, são convertidos em um formato comum pelos adaptadores que compõem a estrutura do adaptador. Quando os pedidos estão em um formato comum, eles podem ser entendidos pelos comandos do WebSphere Commerce.

Comandos são beans que executam a lógica de negócios. Eles representam lógica processual na forma de tarefas lógicas de processo de alto nível ou lógicas de negócios discretas. Um comando baseado em processo age como um controlador que estende várias entidades e outros comandos, enquanto um comando de tarefa executa uma tarefa específica e pode acessar apenas um único objeto.

Estrutura do Comando

Os beans do comando seguem um padrão de design específico. Cada comando inclui uma classe de interface (por exemplo, `CategoryDisplayCmd`) e uma classe de implementação (por exemplo, `CategoryDisplayCmdImpl`). Do ponto de vista do originador da chamada, a lógica da chamada envolve a definição das propriedades de entrada, a chamada de um método `execute()` e a recuperação das propriedades de entrada.

Sob a perspectiva do implementador de comandos, estes seguem a estrutura de comandos do WebSphere, que implementa o padrão de design de comandos permitindo um nível de procedimento indireto entre o originador da chamada e a implementação. Os mecanismos-chave permitidos dentro desde nível de procedimento indireto incluem:

1. A capacidade de chamar um gerenciador de política de controle de acesso que determine se o usuário tem permissão para chamar o comando.
2. A capacidade de executar a implementação de um comando diferente em lojas diferentes, baseado no identificador da loja.
3. A capacidade de executar uma implementação de exibição diferente baseada no tipo de dispositivo do solicitador.

O diagrama a seguir mostra uma visão geral conceitual das interfaces dos quatro tipos principais de comandos:

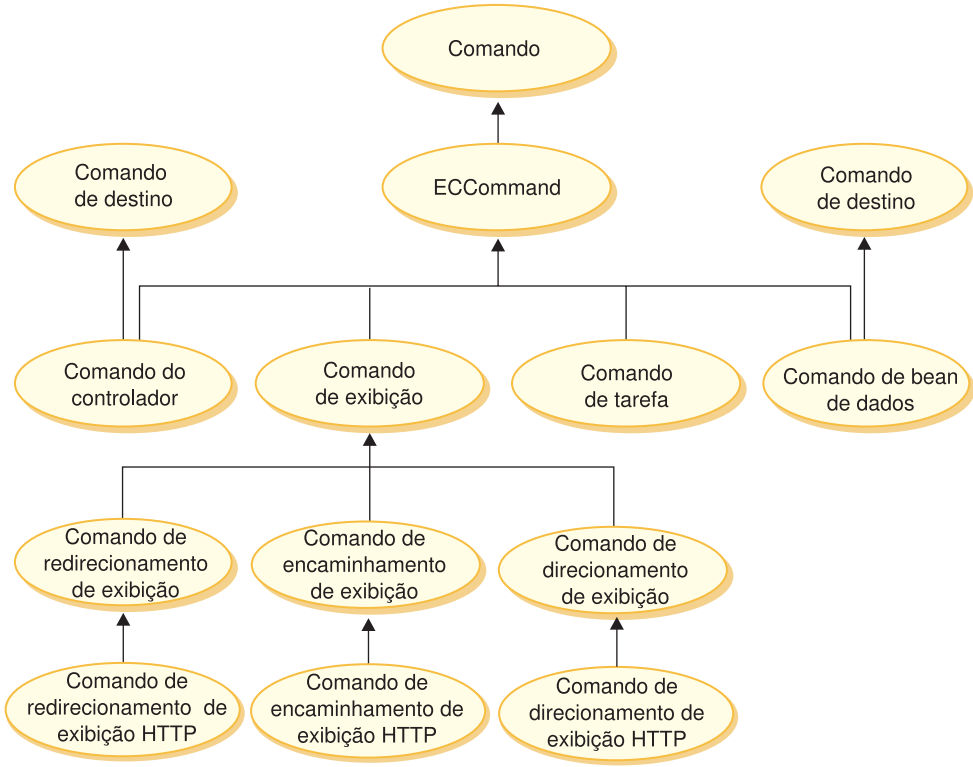


Figura 7.

Comando do controlador

Um comando do controlador encapsula a lógica relacionada a um processo de negócios em particular. Exemplos de comandos do controlador incluem o comando *OrderProcessCmd* para processamento de pedido e o comando *UserRegistrationAddCmd* para a criação de novos usuários registrados. Em geral, um comando do controlador contém as instruções de controle (por exemplo, if, then, else) e invoca os comandos de tarefa para executar tarefas individuais no processo de negócios. Uma vez concluído, um comando do controlador retorna um nome de exibição. O controlador da Web determina a classe de implementação apropriada para a tarefa de exibição e a executa.

Embora um comando do controlador seja um comando de destino, apenas o destino local é suportado.

Comando de tarefa

Um comando de tarefa implementa uma unidade específica de lógica de aplicativo. Em geral, um comando do controlador e um conjunto de comandos de tarefas juntos implementam a lógica do aplicativo

para um pedido de URL. Um comando de tarefa é executado no mesmo contêiner que o comando do controlador.

Comando de bean de dados

Um comando de bean de dados é invocado por uma página JSP quando o bean de dados é instanciado. A função principal de um comando de bean de dados é preencher os campos do bean de dados.

Embora um comando do bean de dados seja um comando de destino, apenas o destino local é suportado.

Comando de exibição

Um comando de exibição compõe uma exibição como uma resposta a um pedido de cliente. Há três maneiras em que um comando de exibição pode ser invocado:

- Um comando do controlador especifica o nome de um comando de exibição após a conclusão com êxito do pedido
- Um cliente pode solicitar diretamente uma exibição.
- Um comando do controlador ou de tarefas detecta um erro e decide que é necessário executar uma tarefa de erro para processar o erro e emite uma exceção com o nome de um comando de exibição. Quando a exceção é propagada para o controlador da Web, ele executa o comando de exibição e retorna a resposta ao cliente.

Há três tipos de comandos de exibição:

Comando de redirecionamento de exibição

Esse comando de exibição envia a exibição utilizando o protocolo de redirecionamento, tal como o redirecionamento da URL. Um comando de controlador deve retornar um comando de exibição desse tipo de exibição quando um protocolo de redirecionamento for necessário. Quando um protocolo de redirecionamento é utilizado, ele altera as pilhas da URL no navegador. Quando uma tecla de recarregar é digitada, é executada a URL redirecionada, em vez da URL original.

Comando de direcionamento de exibição

Esse comando de exibição envia a exibição da resposta diretamente ao cliente.

Comando de encaminhamento de exibição

Esse comando de exibição encaminha o pedido de exibição a outro componente Web, como a um modelo JSP.

Fábrica de Comandos

Para criar novos objetos de comando, o originador da chamada do comando utiliza a *fábrica de comandos*. A fábrica de comandos é um bean que é utilizado para instanciar comandos. Baseia-se no padrão de design da fábrica, o qual diferencia a instância de um objeto fora da classe de invocação, para a classe de fábrica que compreende qual classe de implementação instanciar.

A fábrica fornece uma maneira inteligente de instanciar novos objetos. Nesse caso, a fábrica de comandos fornece uma maneira de se determinar a classe de implementação correta ao criar um novo objeto de comando, com base na loja individual. O nome da interface do comando e o identificador de loja particular são passados em um novo objeto de comando, na instalação.

Existem duas maneiras para a classe de implementação de um comando ser especificada. Uma classe de implementação padrão pode ser especificada diretamente no código da interface do comando, utilizando a variável `defaultCommandClassName`. Por exemplo, o seguinte código existe na interface CategoryDisplayCmd:

```
String defaultCommandClassName =
     "com.ibm.commerce.catalog.commands.CategoryDisplayCmdImpl"
```

A segunda maneira de se especificar a classe de implementação é utilizar o registro de comandos do WebSphere Commerce. O registro de comandos deve sempre ser utilizado quando a classe de implementação variar de uma loja para outra. Mais informações sobre o registro de comandos podem ser encontradas na página 28.

No caso em que uma classe de implementação padrão é especificada no código para a interface e uma classe de implementação diferente é especificada no registro de comandos, o registro de comandos tem precedência.

A sintaxe para utilizar a fábrica de comandos é a seguinte:

```
cmd = CommandFactory.createCommand(interfaceName, commandContext.getStoreId())
```

em que *interfaceName* é o nome da interface para o novo bean do comando e o método getStoreId determina a loja onde o comando deve ser utilizado.

Nota: A sintaxe para utilizar a fábrica de comandos para criar comandos de política de negócios é diferente do fragmento de código acima. Para obter mais informações sobre como utilizar a fábrica de comandos para criar os comandos de política de negócios, consulte “Chamando a Nova Política de Negócios” na página 180.

Fluxo do Comando

Essa seção fornece uma visão geral do fluxo lógico entre os comandos e o banco de dados do WebSphere Commerce. O diagrama e as descrições a seguir representam esse fluxo.

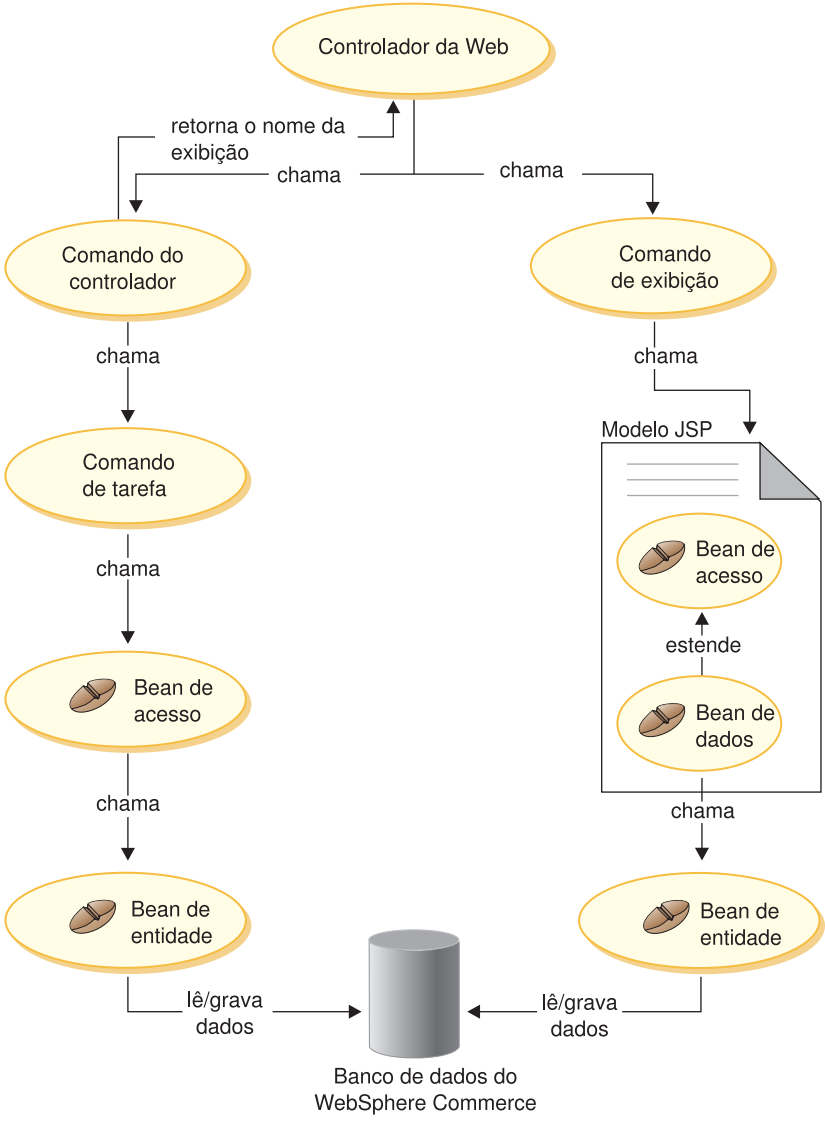


Figura 8.

Quando o controlador da Web recebe um pedido, ele determina se o pedido requer a invocação de um comando do controlador ou de um comando de

exibição. Nos dois casos, o controlador da Web determina também a classe de implementação para o comando e em seguida, o invoca.

Examine primeiro o lado esquerdo do diagrama. Já que os comandos do controlador encapsulam a lógica para um processo de negócios, eles freqüentemente invocam comandos de tarefas individuais para executar unidades específicas de trabalho no processo de negócios. Os beans de acesso são invocados quando as informações no banco de dados devem ser recuperadas ou atualizadas. Tanto um comando de tarefas quanto de controlador podem invocar os beans de acesso. Os pedidos fluem dos beans de acesso para os beans de entidade que podem ser lidos ou gravados no banco de dados do WebSphere Commerce.

Agora examine o lado direito do diagrama. Um comando de exibição é invocado pelo controlador da Web, ou quando um comando do controlador concluiu o processamento e retornou o nome de um comando de exibição a ser invocado ou quando ocorre um erro e uma exibição de erro deve ser exibida.

Os comandos de exibição geralmente invocam um modelo JSP para exibir a resposta ao cliente. Dentro do modelo JSP, os beans de dados são utilizados para preencher informações dinâmicas na página. Os beans de dados são ativados pelo gerenciador de bean de dados. O bean de dados (que se estende a partir de um bean de acesso) invoca seu bean de entidade correspondente. Quando acessado indiretamente de um modelo JSP, um bean de entidade geralmente recupera as informações do banco de dados (em vez de gravar as informações no banco de dados).

Estrutura do Registro de Comandos

Os comandos de controlador e de tarefa do WebSphere Commerce são registrados no registro de comandos. As três tabelas a seguir abrangem o registro de comandos:

- URLREG
- CMDREG
- VIEWREG

Nota: Business Esta seção não se aplica ao registro de comandos de política de negócios. Para obter informações sobre como registrar novos comandos de política de negócios, consulte “Registrando a Nova Política de Negócios e Comando de Política de Negócios” na página 164.

Tabela URLREG

A tabela URLREG mapeia os URIs (Universal Resource Indicator) para as interfaces do comando do controlador. Os URIs fornecem um mecanismo

simples e extensível para a identificação de recursos. Um URI é uma cadeia relativamente curta de caracteres utilizado para identificar um recurso físico ou abstrato. No WebSphere Commerce, o URI contém apenas informações sobre o comando. Na seguinte URL, a seção de URI é mostrada em negrito:

```
http://hostname/webapp/wcs/stores/servlet/StoreCatalogDisplay?
     storeId=store_Id&catalogId=catalog_Id&langId=-1
```

Embora haja um mapeamento um-a-um entre um URI e um nome de interface, cada loja pode especificar se o HTTPS ou AUTHENTICATION é necessário para o comando. Para cada pedido de URL de recepção, o controlador da Web procura o nome da interface para o comando do controlador e em seguida, utiliza esse nome para determinar a classe de implementação correta, conforme registrado na tabela CMDREG.

A tabela a seguir descreve as informações contidas na tabela do banco de dados URLREG.

Nome da coluna	Descrição	Comentários
URL	Nome de URI	Por exemplo `MyNewCommand` ou `ProductDisplay`
STOREENT_ID	Identificador da entidade da loja	Este pode ser definido como `0` para utilizar o comando em todas as lojas ou para um único identificador de loja a fim de indicar que o comando é utilizado apenas para uma determinada loja.
INTERFACENAME	Nome da interface do comando do controlador	Por exemplo `com.ibm.commerce.catalog.commands.GetProductDisplay.TemplateCmd`
HTTPS	HTTP seguro exigido para este pedido de URL	Utilize `1` quando HTTPS for necessário e `0` quando não for.
DESCRIPTION	Descrição do URI	Por exemplo, `Esse comando é utilizado com a finalidade de teste`.
AUTHENTICATED	O logon do usuário é requerido para esse pedido de URL	Utilize `1` quando a autenticação for necessária e `0` quando não for.
INTERNAL	Indica se o comando é ou não interno para o WebSphere Commerce	Utilize `1` quando o comando for interno e `0` quando for externo.

Quando o controlador da Web recebe um pedido de URL, ele recupera o nome da interface para o comando do controlador solicitado e o utiliza para procurar o nome da classe de implementação a partir da tabela CMDREG. Também determina se o HTTPS é exigido para o pedido de URL por meio da verificação da coluna HTTPS na tabela URLREG.

Apenas os comandos que são invocados por meio de pedidos de URL precisam ser registrados na tabela URLREG. Portanto, apenas os comandos do controlador devem ser registrados aqui, nenhum comando de exibição ou de tarefas.

A instrução SQL a seguir cria uma entrada para MyNewControllerCommand que é utilizada por uma determinada loja (cujo identificador de loja é 5):

```
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS,
DESCRIPTION, AUTHENTICATED) values ('MyNewControllerCommand',
5,'com.ibm.commerce.commands.MyNewControllerCommand',0,
'This is a test command.',null)
```

A sintaxe genérica para a instrução insert é a seguinte:

```
insert into table_name (column_name1,column_name2, ... ,column_namen)
values (column1_value,column2_value,...,columnn_value)
```

Os valores da cadeia devem estar entre aspas simples.

Tabela CMDREG

CMDREG é a tabela de registro de comandos. Essa tabela fornece um mecanismo para mapear a interface de comando para sua classe de implementação. Várias implementações de uma interface são permitidas para a personalização do comando em uma base por loja.

Apenas os comandos do controlador e os comandos de tarefas são registrados na tabela CMDREG. Os comandos de exibição são registrados na tabela VIEWREG.

A tabela a seguir descreve as informações contidas na tabela do banco de dados CMDREG.

Nome da coluna	Descrição	Comentários
STOREENT_ID	Identificador da entidade da loja	Este pode ser definido como 0 para utilizar o comando em todas as lojas ou para um único identificador de loja a fim de indicar que o comando é utilizado apenas para uma determinada loja.

Nome da coluna	Descrição	Comentários
INTERFACENAME	Nome da interface do comando	Define a interface; utilize o mesmo nome utilizado na tabela URLREG.
DESCRIPTION	Descrição desse comando	Por exemplo, `Esse comando é utilizado com a finalidade de teste.`
CLASSNAME	Nome da classe de implementação do comando	Geralmente o nome da interface com ″Impl″ anexado ao final.
PROPERTIES	Os pares nome-valor padrão definidos conforme as propriedades de entrada para o comando	O formato é o mesmo que a cadeia de consulta de URL. Por exemplo ″parm1=val1&parm2=val2″
LASTUPDATE	Última atualização nesta entrada de comando	
TARGET	Nome de destino do comando. É onde o comando é executado atualmente.	Apenas o destino local é suportado.

Em geral, quando você cria um novo controlador ou comando de tarefas, será necessário criar a entrada correspondente na tabela CMDREG. Por exemplo, a instrução SQL a seguir cria uma entrada do MyNewCommand que é utilizado por uma determinada loja (cujo identificador de loja é 5):

```
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION, CLASSNAME,
PROPERTIES, LASTUPDATE, TARGET) values
(5,'com.ibm.commerce.catalog.commands.MyNewCommand', 'This is a test
command', 'com.ibm.commerce.catalog.commands.MyNewCommandImpl',
'myDefaulParm1=myDefaultVal1', '0000-12-01', 'Local')
```

A sintaxe genérica para a instrução insert é a seguinte:

```
 insert into table_name (column_name1,column_name2, ... ,column_namen)
values (column1_value,column2_value,...,columnn_value)
```

Os valores da cadeia devem estar entre aspas simples.

Se o comando que você está escrevendo sempre utiliza a mesma classe de implementação, você não tem que necessariamente registrar o comando na tabela CMDREG. Nesse caso, poderá utilizar o atributo defaultCommandClassName na interface para especificar a classe de implementação. Por exemplo, no código da interface, você incluiria o seguinte:

```
String defaultCommandClassName =
     "com.ibm.commerce.command.MyNewCommandImpl"
```

Se a classe de implementação for especificada dessa maneira, não será possível passar as propriedades padrão para a classe de implementação e a mesma classe de implementação deverá ser utilizada em todas as lojas.

Exemplo de um comando do controlador registrado

Considere um cenário no qual seu site possui duas lojas: StoreA e StoreB. Cada loja possui requisitos de segurança diferentes para o comando do controlador MyUrl, bem como implementações diferentes do comando. Essa seção mostra como o registro de comandos é utilizado para permitir essa personalização.

A tabela a seguir mostra as entradas da StoreA e StoreB na tabela URLREG:

Nome da coluna	Entrada para StoreA	Entrada para StoreB
URL	MyUrl	MyUrl
STOREENT_ID	11	22
INTERFACENAME	com.ibm.commerce. mycommands.myUrl	com.ibm.commerce. mycommands.myUrl
HTTPS	1	1
DESCRIPTION	Exemplo de entrada na tabela URLREG.	Exemplo de entrada na tabela URLREG.
AUTHENTICATED	1	0
INTERNAL	nulo	nulo

Nota: Os espaços nos valores para INTERFACENAME são apenas com a finalidade de exibição. Cada valor é realmente uma cadeia contínua.

Com base nas entradas na tabela URLREG, o controlador da Web determina se o nome da interface para o URI MyURL é com.ibm.commerce.mycommands.MyUrl. Determina também se a StoreA requer que o comando seja executado utilizando HTTPS e autenticação, mas a StoreB requer apenas HTTPS. Os valores do HTTPS e autenticação são utilizados pelo controlador da Web, não pela interface.

O diagrama a seguir mostra este fluxo:

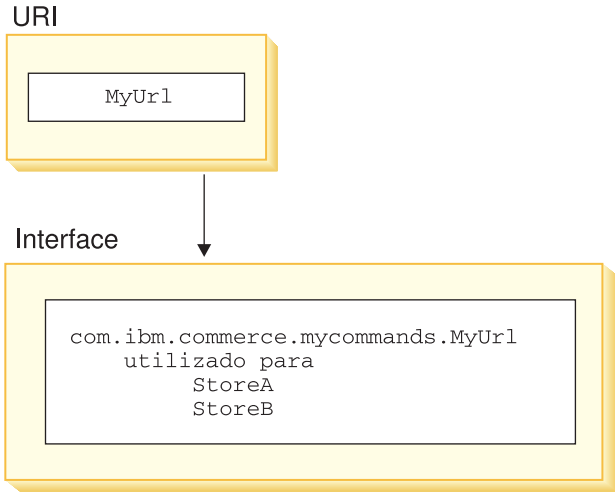


Figura 9.

A tabela a seguir mostra as entradas na tabela CMDREG. Apenas as colunas requeridas para a finalidade deste exemplo são exibidas:

Nome da coluna	Entrada para StoreA	Entrada para StoreB
STOREENT_ID	11	22
INTERFACENAME	com.ibm.commerce. mycommands.myUrl	com.ibm.commerce. mycommands.myUrl
CLASSNAME	com.ibm.commerce. mycommands. myUrlStoreAImpl	com.ibm.commerce. mycommands.myUrlStoreBImpl

Nota: Os espaços nos valores para INTERFACENAME e CLASSNAME são apenas com a finalidade de exibição. Cada valor é realmente uma cadeia contínua.

Com base nas entradas na tabela CMDREG, o controlador da Web determina se para a StoreA, a classe de implementação para a interface com.ibm.commerce.mycommands.MyUrl é com.ibm.commerce.mycommands.MyUrlStoreAImpl. Determina também para StoreB, a classe de implementação para a mesma interface é com.ibm.commerce.mycommands.MyUrlStoreBImpl. O diagrama a seguir mostra este fluxo:

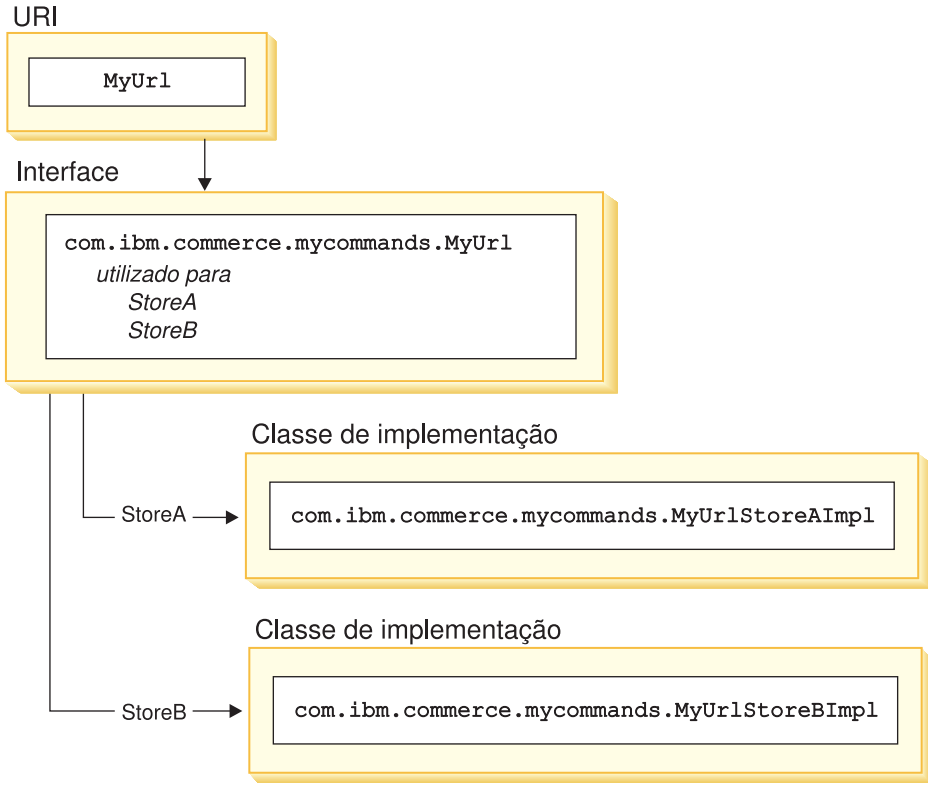


Figura 10.

Tabela VIEWREG

A tabela VIEWREG permite registro de implementações do comando de exibição específico do dispositivo. Utilizando essa tabela, várias implementações de uma exibição podem ser registradas. A estrutura do comando é, então, capaz de retornar diferentes exibições para vários clientes.

Quando um nome de comando de exibição é retornado de um comando do controlador ou especificado em uma exceção, o controlador da Web determina a classe de comando de exibição da tabela VIEWREG. Vários nomes de comando de exibição podem ser mapeados para a mesma classe de implementação.

Nome da coluna	Descrição	Comentários
VIEWNAME	Nome da exibição	Por exemplo, `AddressForm`

Nome da coluna	Descrição	Comentários
DEVICEFMT_ID	Identificador do tipo de dispositivo	As opções disponíveis incluem: • BROWSER (valor padrão) • I_MODE • E-mail • MQXML • MQNC
STOREENT_ID	Identificador da entidade da loja	Este pode ser definido como `0` para utilizar o comando em todas as lojas ou para um único identificador de loja a fim de indicar que o comando é utilizado apenas para uma determinada loja.
INTERFACENAME	Nome da interface de comando de exibição	As opções padrão são ForwardView, DirectView e RedirectView.
CLASSNAME	Nome da classe de implementação do comando de exibição	Pode utilizar a implementação padrão.
PROPERTIES	Os pares nome-valor padrão definidos conforme as propriedades de entrada para o comando	Se for exibida a mesma página, defina o nome do arquivo JSP nesta propriedade (`docname=`*`jsp_name`*`.jsp`). Se for utilizado o mesmo modelo JSP em todas as lojas, defina `storeDir=no` para impedir que um diretório específico da loja seja utilizado. Se um usuário genérico puder invocar o comando, defina `isGeneric=true`.
DESCRIPTION	Descrição desse comando	
HTTPS	HTTP seguro exigido para este pedido de URL	Utilize `1` quando HTTPS for necessário e `0` quando não for.
LASTUPDATE	Última atualização nesta entrada	
INTERNAL	Indica se o comando é ou não interno para o WebSphere Commerce	Utilize `1` quando o comando for interno e `0` quando for externo.

Quando você criar um novo comando de exibição, convém criar uma entrada correspondente na tabela VIEWREG. Se uma das seguintes condições for correspondida, o comando de exibição deverá ser registrado na tabela VIEWREG:

- O comando de exibição é executado sob o controle de acesso
- Há várias implementações do comando de exibição
- As propriedades são definidas na coluna PROPERTIES

Os comandos de exibição registrados podem ser acessados através do registro de comandos utilizando o nome da exibição ou diretamente, utilizando o nome do arquivo de exibição real. As exibições que não estão registradas na tabela VIEWREG podem ser acessadas apenas quando um cliente utilizar o nome do arquivo de exibição real.

Considere o exemplo de uma exibição nomeada *MyView*, com a entrada VIEWREG, conforme segue:

Nome da coluna	**Entrada**
VIEWNAME	`MyView`
DEVICEFMT_ID	`BROWSER`
STOREENT_ID	`0`
INTERFACENAME	`com.ibm.commerce.commands.ForwardViewCommand`
CLASSNAME	`com.ibm.commerce.commands.HTTPForwardViewCommandImp`
PROPERTIES	`docname=MyView.jsp`
DESCRIPTION	`Um exemplo de chamada de um modelo JSP utilizando ou o nome de exibição ou diretamente de uma URL.`
HTTPS	`0`
LASTUPDATE	`2000-11-30`
INTERNAL	`0`

Como o MyView é uma exibição registrada, um cliente pode acessar a exibição utilizando o nome do comando ou substituindo o nome do arquivo de exibição real pelo nome do comando. Ao utilizar o nome de exibição, uma URL de amostra será:

`http://`*`hostname`*`.com/webapp/wcs/stores/servlet/MyView`

e ao utilizar o nome do arquivo, uma URL de amostra será:

`http://`*`hostname`*`.com/webapp/wcs/stores/servlet/MyView.jsp`

Se houver a possibilidade de um cliente invocar uma exibição registrada diretamente (utilizando o nome do arquivo de exibição), será necessário

registrar o comando utilizando o mesmo nome para a exibição que o nome do arquivo de exibição real, conforme mostrado nesse exemplo (MyView e MyView.jsp).

Uma exibição que não esteja registrada na tabela somente poderá ser invocada utilizando o nome do arquivo de exibição. Portanto, se houver uma exibição não registrada que utiliza o arquivo MyUnregisteredView.jsp, a URL para acessar essa exibição será como segue:

```
http://hostname.com/webapp/wcs/stores/servlet/MyUnregisteredView.jsp
```

O exemplo a seguir de instrução SQL cria uma entrada para *MyNewViewCommand* que é utilizada por uma loja em particular:

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID,
INTERFACENAME, CLASSNAME, PROPERTIES, DESCRIPTION,HTTPS, LASTUPDATE,
INTERNAL) values ('MyNewViewCommand', 'BROWSER', 5,
'com.ibm.commerce.command.ForwardViewCommand',
'com.ibm.commmerce.command.HttpForwardViewCommandImpl',
'docname=MyNewViewCommand.jsp', 'A test view command.', 0,
'0000-12-01', 0)
```

A tabela a seguir fornece outro exemplo de tabela VIEWREG com informações-chave:

COMMAND - NAME	DEVICE - FMT_ID	INTERFACE - NAME	CLASSNAME	PROPERTIES
ProductDisplayView	BROWSER	Comando de Encaminha-mento de Exibição	HttpForwardView CommandImpl	
InterestItemAddView	BROWSER	Comando de Redirecio-namento de Exibição	HttpRedirectView CommandImpl	docname =item.jsp
InterestItemDelete View	BROWSER	Comando de Redirecio-namento de Exibição	HttpRedirectView CommandImpl	docname =item.jsp
GenericApplication Error	BROWSER	Comando de Redirecio-namento de Exibição	HttpRedirectView CommandImpl	docname =usererr.jsp

COMMAND - NAME	DEVICE - FMT_ID	INTERFACE - NAME	CLASSNAME	PROPERTIES
GenericSystemError	BROWSER	Comando de Redirecio-namento de Exibição	HttpRedirectView CommandImpl	docname =syserr.jsp
Logon	BROWSER	Comando de Encaminha-mento de Exibição	HttpForwardView CommandImpl	docname= logon.jsp & storeDir=no

Nota: Todos os espaços nos valores COMMANDNAME, INTERFACENAME, CLASSNAME e PROPERTIES são apenas devido à exibição. Cada valor é realmente uma cadeia contínua. Os hífens nos nomes das colunas também são apenas com a finalidade de exibição.

A tabela precedente ilustra os seguintes cenários:

- O nome de exibição *ProductDisplayView* é retornado ao controlador da Web por um comando do controlador (ProductDisplay, nesse caso). O controlador da Web determina a interface de comando de exibição e os nomes da classe utilizando o nome do comando de exibição ProductDisplayView e seu identificador de dispositivo. Um comando de exibição pode ter diferentes classes de implementação para diferentes lojas e identificadores de dispositivo. O nome da interface, entretanto, deve permanecer o mesmo, já que ele define o tipo de comando de exibição.
- Os comandos InterestItemAdd e InterestItemDelete retornam os nomes de exibição *InterestItemAddView* e *InterestItemDeleteView* ao controlador da Web, respectivamente. Os dois comandos requerem exibições redirecionadas, portanto, o nome da interface de comando de exibição para ambas exibições é RedirectViewCommand. Existe um modelo JSP comum registrado para ambas exibições. O controlador da Web busca as propriedades (docname=item.jsp) e as transmite ao comando de exibição (HttpRedirectViewCommandImpl).
- Se um comando do controlador ou de tarefas lança uma exceção ECApplication para um parâmetro de usuário inválido, poderá ocorrer o seguinte:
  - Se houver uma exibição especificada dentro do comando do controlador que deva ser chamada no caso de uma exceção de aplicativo, a entrada para essa exibição será recuperada da tabela VIEWREG e, dessa maneira, processada.
  - Se uma exibição não estiver especificada, o comando GenericApplicationError será chamado e o modelo JSP registrado no

banco de dados será exibido. A utilização da tabela anterior como exemplo, resultaria na exibição do modelo `usererr.jsp`.

- Se um comando do controlador ou de tarefas lança uma exceção ECSystem para uma exceção do sistema, poderá ocorrer o seguinte:
  - Se houver uma exibição especificada dentro do comando do controlador que deva ser chamada no caso de uma exceção do sistema, a entrada para essa exibição será recuperada da tabela VIEWREG e, dessa maneira, processada.
  - Se uma exibição não estiver especificada, o comando GenericSystemError será chamado e o modelo JSP registrado no banco de dados será exibido. A utilização da tabela anterior como exemplo, resultaria na exibição do modelo `syserr.jsp` .
- Os clientes do navegador podem invocar a página de logon digitando a URL de logon. Visto que a propriedade storeDir está definida como “não”, as informações específicas da loja não são incluídas no caminho para o modelo JSP. Assim, a mesma página de logon é exibida para clientes em todas as lojas.

Padrão de Design de Exibição

As páginas de exibição retornam uma resposta a um cliente. Geralmente, as páginas de exibição são implementadas como modelos JSP (o método recomendado), portanto, podem ser gravados diretamente como servlets.

Para suportar vários tipos de dispositivos, o acesso de uma URL a um comando de exibição utilizaria o nome da exibição, não o nome do arquivo JSP real.

O fundamento lógico principal atrás desse nível de procedimento indireto é que o modelo JSP representa uma exibição. A capacidade de selecionar a exibição apropriada (por exemplo, com base no locale, tipo de dispositivo ou outros dados no contexto de pedido) é altamente desejável, especialmente visto que um único pedido geralmente possui várias exibições possíveis. Considere o exemplo de dois compradores solicitado a home page de uma loja, um comprador utilizando um típico navegador da Web e o outro um telefone celular. Logicamente, a mesma home page não deverá ser exibida para cada comprador. É de responsabilidade do controlador da Web aceitar o pedido, então, com base nas informações na estrutura do registro de comandos, determinar a exibição que cada comprador receberá.

Modelos JSP e Beans de Dados

Um Bean de dados é um bean Java que é utilizado dentro de um modelo JSP para fornecer conteúdo dinâmico. Um bean de dados normalmente fornece uma representação simples do bean de entidade do WebSphere Commerce. O bean de dados encapsula propriedades que podem ser recuperadas de ou

definidas no bean de entidade. Como tal, o bean de dados simplifica a tarefa de incorporar dados dinâmicos nos modelos JSP.

O bean de dados possui uma classe *BeanInfo* que define as propriedades que podem ser utilizadas na página de exibição. A classe BeanInfo permite também a utilização dos beans de dados em sites multiculturais, fornecendo os nomes da propriedade em todos os idiomas suportados do WebSphere Commerce.

Um bean de dados é ativado pela seguinte chamada:

```
com.ibm.commerce.beans.DataBeanManager.activate(DataBean, HTTPServletRequest)
```

O WebSphere Studio Page Designer gera automaticamente a linha de códigos anterior quando um bean de dados do WebSphere Commerce é inserido em um modelo JSP.

Os desenvolvedores da loja devem considerar as propriedades da loja e as abordagens de permissão multiculturais ao desenvolver os modelos JSP. Para obter mais informações sobre a ativação multicultural, consulte a ajuda online do WebSphere Commerce.

Considerações de segurança sobre modelos JSP e beans de dados

Uma prática de codificação particular para a utilização de modelos JSP e beans de dados minimiza a possibilidade de usuários maliciosos acessarem o banco de dados de uma maneira não autorizada. A inserção, seleção, atualização e exclusão de partes de instruções SQL deve ser criada na hora do desenvolvimento. Utilize inserções de parâmetros para reunir informações de entrada de runtime.

A seguir está um exemplo da utilização de uma inserção de parâmetro para reunir informações de entrada de runtime:

```
select * from Order where owner =?
```

Em comparação, você deve evitar utilizar cadeias de entrada como uma forma de compor a instrução SQL. A seguir está um exemplo da utilização de uma cadeia de entrada:

```
select * from Order where owner = "input_string"
```

Tipos de Beans de Dados

Um bean de dados é um bean Java que é utilizado principalmente para fornecer dados dinâmicos em modelos JSP. Há dois tipos de beans de dados: beans de dados inteligentes e beans de dados do comando.

O bean de dados inteligente utiliza um método *captura atrasada* para recuperar seus próprios dados. Esse tipo de bean de dados pode fornecer um melhor desempenho em situações onde nem todos os dados do bean de acesso são

requeridos, já que recupera dados apenas conforme for necessário. Os beans de dados inteligentes que requerem acesso ao banco de dados devem estender o bean de acesso para o bean de entidade correspondente e implementarem a interface com.ibm.commerce.SmartDataBean. Por exemplo, o bean de dados ProductData estende o bean de acesso ProductAccessBean, que corresponde ao bean de entidade Produto.

Alguns beans de dados inteligentes não requerem acesso ao banco de dados. Por exemplo, o bean de dados inteligente PropertyResource recupera dados de um pacote de recursos, em vez do banco de dados. Quando o acesso ao banco de dados não é requerido, o bean de dados inteligente deve estender a classe SmartDataBeanImpl.

Um bean de dados do comando conta com um comando para recuperar seus dados e um bean de dados mais leve. O comando recupera todos os atributos para o bean de dados de uma só vez, independente se o modelo JSP os requer. Como resultado, para os modelos JSP que utilizam apenas uma seleção de atributos do bean de dados, um bean de dados do comando pode ser custoso em termos de tempo de desempenho. Para modelos JSP que requerem a maioria ou todos os atributos, o bean de dados do comando é muito conveniente.

Os beans de dados do comando também podem estender a partir de seus beans de acesso correspondente e implementarem a interface com.ibm.commerce.CommandDataBean.

Interfaces do bean de dados

Os beans de dados implementam uma ou todas as seguintes interfaces Java:

- com.ibm.commerce.SmartDataBean.
- com.ibm.commerce.CommandDataBean
- com.ibm.commerce.InputDataBean (opcional)

Cada interface Java descreve a origem de dados da qual um bean de dados é ocupado. Através da implementação de várias interfaces, o bean de dados pode acessar os dados de uma variedade de origens. Mais informações sobre cada uma das interfaces são fornecidas abaixo.

Interface SmartDataBean: Um bean de dados implementando a interface SmartDataBean pode recuperar seus próprios dados, sem um comando de bean de dados associado. Um bean de dados inteligente geralmente se estende a partir do bean de acesso de um bean de entidade correspondente. Quando um bean de dados inteligente está ativado, o gerenciador de bean de dados invoca o método de preenchimento do bean de dados. utilizando o método de preenchimento, o bean de dados pode recuperar todos os atributos, exceto os dos objetos associados. Por exemplo, se o bean de dados se estende de uma

classe de bean de dados de acesso para um bean de entidade, o bean de dados invocará o método refreshCopyHelper. Todos os atributos do bean de entidade correspondente são automaticamente preenchidos no bean de dados inteligente. Entretanto, se o bean de entidade possuir objetos associados, os atributos desses objetos não serão recuperados. As principais vantagens da utilização dos beans de dados inteligentes são:

- A implementação é simples e não há necessidade de gravar um comando do bean de dados.
- Quando novos campos são incluídos no bean de entidade, as alterações no bean de dados não são necessárias. Após o bean de entidade ter sido modificado, o bean de acesso deve ser gerado novamente (utilizando as ferramentas em VisualAge for Java). Logo que o bean de acesso for gerado novamente, todos os novos atributos estarão automaticamente disponíveis no bean de dados inteligente.
- Os beans de entidade geralmente contêm atributos que representam objetos associados. Por motivos de desempenho o bean de dados inteligente não recupera automaticamente esses atributos. Em vez disso, é preferível atrasar a recuperação desses atributos até que sejam requeridos, conforme mostrado no diagrama a seguir:

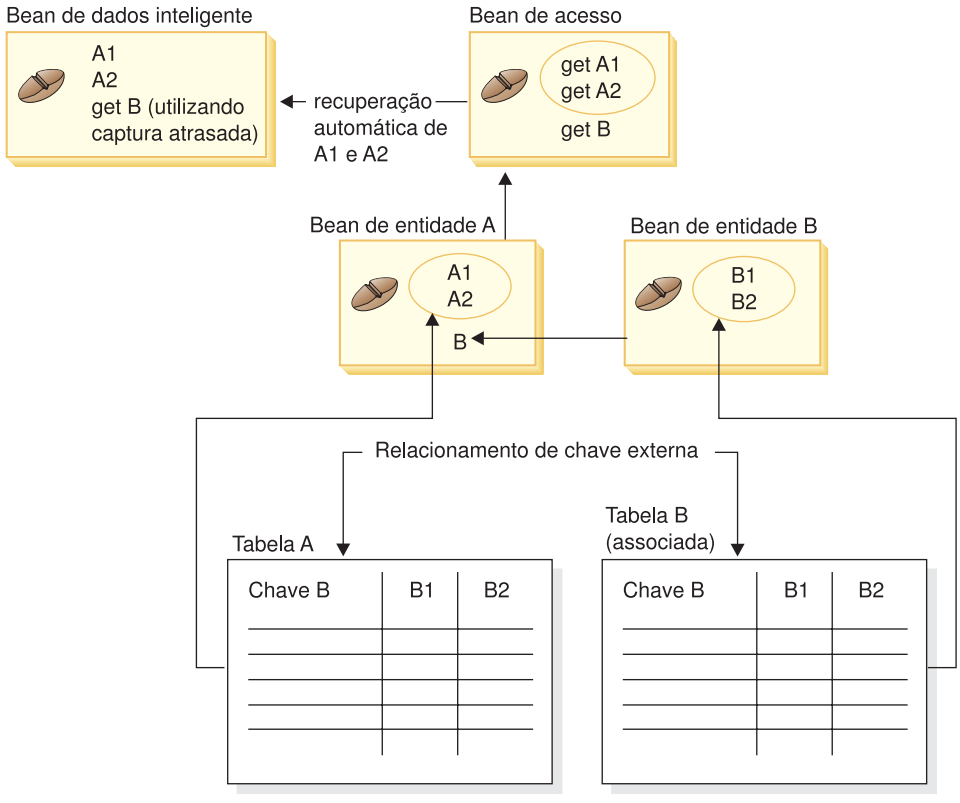


Figura 11.

Para obter mais informações sobre a implementação de uma recuperação de captura atrasada, consulte "Recuperação de Dados da Captura Atrasada" na página 45.

Interface CommandDataBean: Um bean de dados implementando a interface CommandDataBean recupera dados de um comando de bean de dados. Um bean de dados desse tipo é um objeto ligeiro; ele conta com um comando do bean de dados para preencher seus dados. O bean de dados deve implementar o método getCommandInterfaceName() (conforme definido pela interface com.ibm.commerce.CommandDataBean) que retorna o nome da interface do comando do bean de dados.

Interface InputDataBean: Um bean de dados implementando a interface InputDataBean recupera dados dos parâmetros URL ou atributos definidos pelo comando de exibição.

Os atributos definidos nessa interface podem ser utilizados como campos-chave principais para buscar dados adicionais. Quando um modelo

JSP é invocado, o código servlet JSP gerado preenche todos os atributos que correspondem aos parâmetros URL e em seguida, ativa o bean de dados passando-o ao gerenciador de bean de dados. O gerenciador de bean de dados, em seguida, invoca o método setRequestProperties() do bean de dados (conforme definido pela interface com.ibm.commerce.InputDataBean) para passar todos os atributos definidos pelo comando de exibição. Deve-se observar que o WebSphere Studio gera o seguinte código para cada bean de dados que é inserido nas páginas, utilizando o Page Designer:

```
com.ibm.commerce.beans.DataBeanManager.activate(DataBean, HTTPServletRequest);
```

Classe BeanInfo

Um bean de dados não é completo sem uma classe BeanInfo que implementa a interface java.lang.Object.BeanInfo. A classe BeanInfo é utilizada para fornecer informações explícitas sobre os métodos e as propriedades do bean de dados. Ela pode ser utilizada para ocultar métodos runtime públicos na classe de implementação do bean de dados a partir do Web designer, ou definir a cadeia de exibição apropriada para cada um dos atributos do bean de dados.

Para obter mais informações sobre a implementação de uma classe BeanInfo, consulte a especificação JavaBeans da Sun Microsystems.

Ativação do bean de dados

Os beans de dados podem ser ativados utilizando os métodos ativar ou silentActivate que são encontrados na classe com.ibm.commerce.beans.DataBeanManager. O método ativar é um método de ativação completo em que o evento de ativação é apenas bem-sucedido se todos os atributos estiverem disponíveis. Se mesmo um atributo não estiver disponível, será lançada uma exceção para todo o processo de ativação.

O método silentActivate não lança exceções quando atributos individuais não estão disponíveis.

Chamando Comandos do Controlador a Partir de um Modelo JSP

Embora a chamada de comandos do controlador de um modelo JSP não seja consistente com a separação de lógica da exibição, você pode encontrar uma situação em que isso seja obrigatório. Se for assim, o `ControllerCommandInvokerDataBean` poderá ser utilizado com essa finalidade.

Utilizando esse bean de dados, você pode especificar o nome da interface do comando a ser chamado, ou pode definir diretamente o nome do comando a ser chamado. Você também pode definir as propriedades do pedido para o comando.

Quando esse bean de dados é ativado pelo gerenciador do bean de dados, o comando do controlador é executado e as propriedades da resposta ficam disponíveis para o modelo JSP.

Depois que o comando do controlador é executado, você pode executar a exibição.

Recuperação de Dados da Captura Atrasada

Quando um bean de dados está ativado, poderá ser preenchido por um comando de bean de dados ou pelo método populate() do bean de dados. Os atributos que são recuperados vêm de um bean de dados correspondente ao bean de entidade. Um bean de entidade também pode ter objetos associados, os quais eles próprios, possuem um número de atributos.

Se, na ativação, os atributos de todos os objetos associados forem automaticamente ativados, poderá haver um problema no desempenho. O desempenho pode diminuir conforme o número de objetos associados aumenta.

Considere um bean de dados do produto que contém um grande número de vendas cruzadas, venda-expandida ou produtos de acessórios (objetos associados). É possível preencher todos os objetos associados assim que o bean de dados do produto é ativado. No entanto, o preenchimento dessa maneira pode requerer consultas em vários bancos de dados. Se nem todos os atributos forem requeridos pela página, as consultas em vários bancos de dados poderá ser ineficiente.

Em geral, nem todos os atributos são requeridos para uma página, portanto, um padrão de design melhor é executar uma captura atrasada conforme ilustrado abaixo:

```
getCrossSellProducts () {
   if (crossSellDataBeans == null)
     crossSellDataBeans= getCrossSellDataBeans();
   return crossSellDataBean;
    }
```

Definindo Atributos de JSP - Visão Geral

O modelo do programa WebSphere Commerce promove o padrão de design MVC. Como tal, a apresentação do resultado de um pedido de URL é separado dos comandos do controlador e de tarefas. Esses comandos são independentes do dispositivo. Eles implementam lógicas de negócios e produzem dados para serem retornados ao cliente, sem ter as informações sobre o cliente. De modo oposto, um comando de exibição é específico do dispositivo.

Enquanto os comandos do controlador e de tarefas não compõem diretamente a exibição, eles realmente passam informações para a exibição. É importante compreender como as informações são transmitidas para a exibição. O diagrama a seguir demonstra como as propriedades são transmitidas entre o controlador da Web, o registro de comandos, o comando do controlador e o comando de exibição:

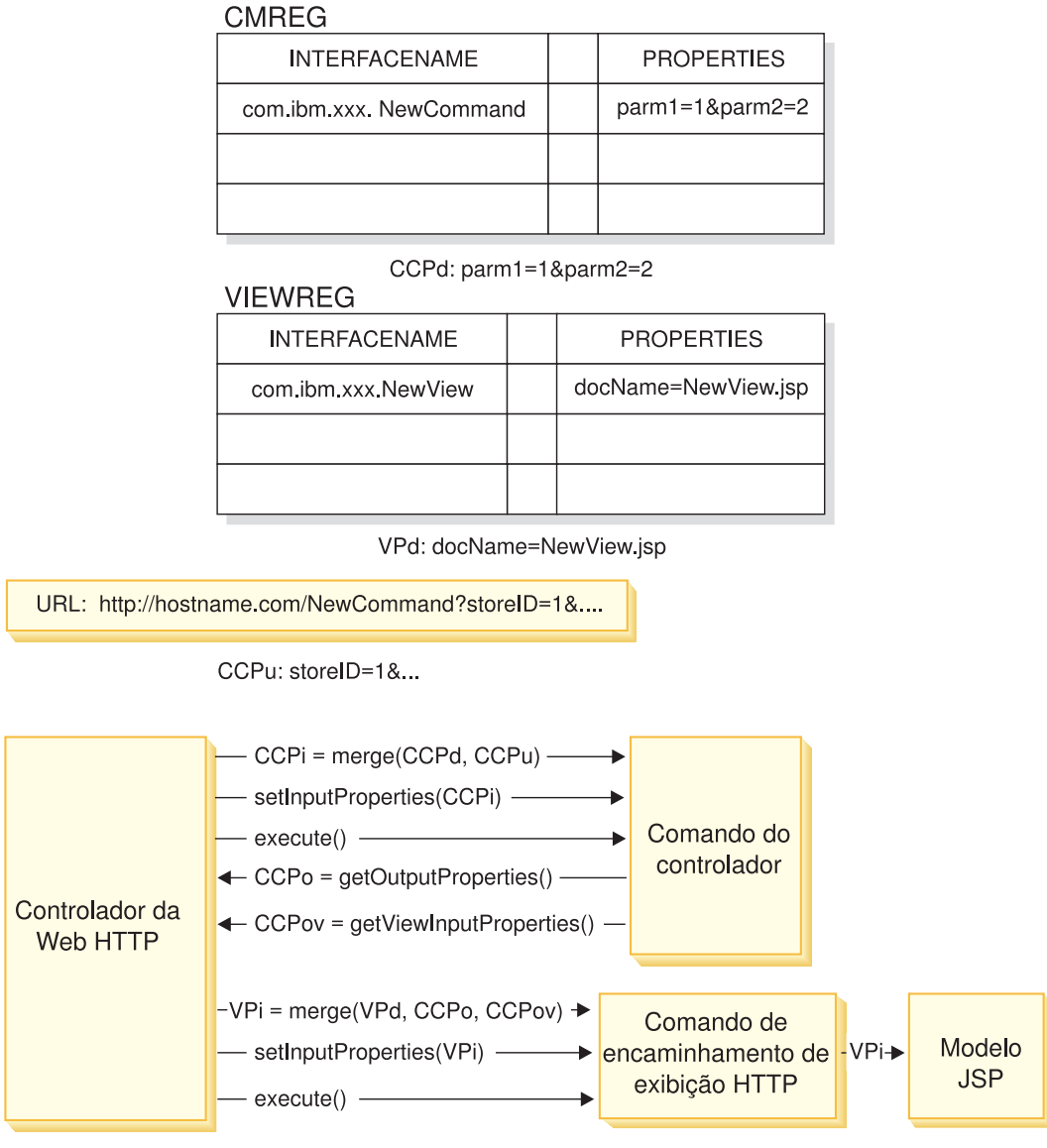


Figura 12.

O diagrama precedente mostra as seguintes interações:

- O controlador da Web combina as propriedades de entrada dos parâmetros URL (CCPu) e a entrada na tabela CMDREG para o comando do controlador (CCPd). Isto cria CCPi.
- O controlador da Web passa as propriedades mescladas (CCPi) para o comando do controlador e executa o comando do controlador.
- O comando do controlador define as propriedades de saída, como CCPo. Estas são as propriedades de saída produzidas pelo próprio comando. Uma das propriedades de saída, viewCommandName, é definida para o nome do comando de exibição desejado. Essas propriedades são recuperadas pelo controlador da Web utilizando um método get.
- O comando do controlador define outro conjunto de propriedades de saída, como CCPov. Por padrão, estes são definidos nas propriedades de entrada combinadas originais (CCPi). É possível personalizar essas propriedades. Por exemplo, pode não ser necessário passar todos os parâmetros de entrada para o comando de exibição.
- O controlador da Web combina os três conjuntos de propriedades, CCPo, CCPov e VPd (as propriedades que estão registradas na tabela VIEWREG) nas propriedades de entrada para o comando de exibição (VPi).
- O controlador da Web define as propriedades mescladas, VPi, e executa o comando de exibição.
- O comando de exibição define os atributos para o modelo JSP a partir das propriedades de entrada.

Ao gravar novos comandos, não é necessário executar explicitamente a mesclagem das propriedades. As classes de comandos abstratos incluem um método mergeProperties. Para obter mais informações sobre este método, consulte a seção "Referência" da ajuda online do WebSphere Commerce.

Definições das Propriedades Requeridas

Um comando do controlador deve definir as seguintes propriedades para cada tipo de comando de exibição. Se as propriedades não forem definidas pelo comando, deverão ser definidas na tabela VIEWREG.

- Se utilizar o comando ForwardView, defina `docname = view_file_name` em que *view_file_name* é o nome do modelo de exibição. Por exemplo, `docname = productDisplay.jsp`.
- Se utilizar o comando DirectView, proceda de uma das seguintes formas:
  - Defina `textDocument = xxx` em que *xxx* é um objeto java.io.InputStream que contém o documento na forma de texto
  - Defina `rawDocument = yyy` em que *yyy* é um objeto java.io.InputStream que contém o documento no formato binário

  Ao utilizar o comando DirectView, é opcional definir `contentType = ttt` em que *ttt* é o tipo do conteúdo do documento

- Se utilizar o comando RedirectView, defina `url = uuu` em que *uuu* é a URL redirecionada.

Capítulo 3. Modelo de Objeto Persistente

O WebSphere Commerce lida com uma grande quantidade de dados persistentes. Existem mais de 520 tabelas definidas no esquema atual do banco de dados. Mesmo com esse esquema extensivo, convém ampliar ou personalizar o esquema do banco de dados para as necessidades de seu negócio em particular.

O WebSphere Commerce utiliza beans de entidade baseados na arquitetura do componente EJBs (Enterprise JavaBeans) como a camada persistente do objeto. Esses beans de entidade representam os dados do WebSphere Commerce de uma maneira que modela os conceitos e objetos no domínio de comércio. Essa camada persistente fornece uma estrutura extensível.

VisualAge for Java, o Enterprise Edition fornece ferramentas EJB sofisticadas e um ambiente de teste de unidade que suporta desenvolvimento para esta estrutura.

Implementação dos Beans de Entidade do WebSphere Commerce

Beans de Entidade do WebSphere Commerce - Visão Geral

Conforme mencionado anteriormente, a camada persistente dentro da arquitetura do WebSphere Commerce é implementada de acordo com a arquitetura do componente EJB. A arquitetura EJB define dois tipos de beans corporativos: beans de entidade e beans de sessão. Os beans de entidade são posteriormente divididos em beans de persistência gerenciado pelo contêiner (CMP) e beans de persistência gerenciado pelo bean (BMP).

A maioria dos beans de entidade do WebSphere Commerce são beans de entidade CMP. Um número pequeno de beans de sessão sem estado são utilizados para manusear as operações intensivas do banco de dados, como executar a soma de todas as linhas em uma determinada coluna. Uma das vantagens de se utilizar os beans de entidade CMP é que os desenvolvedores podem utilizar as ferramentas EJB fornecidas em VisualAge for Java, Enterprise Edition. Essas ferramentas permitem aos desenvolvedores definir os objetos Java e seus mapeamentos da tabela do banco de dados. As ferramentas automaticamente geram persistentes requeridos para os beans de entidade. Persistentes são objetos Java que persistem os campos Java no banco de dados e ocupam os campos Java com dados do banco de dados.

O VisualAge for Java fornece duas extensões para as especificações EJB atuais: associação e herança EJB. A herança EJB permite que um bean corporativo

herde propriedades, métodos e atributos do descritor de controle de nível do método a partir de outro bean corporativo que reside no mesmo grupo. Uma associação é um relacionamento que existe entre dois beans de entidade CMP.

Alguns dos beans de entidade do WebSphere Commerce exploram o recurso de herança EJB. Os beans de entidade do WebSphere Commerce não utilizam o recurso de associações fornecido pelo VisualAge for Java. Ao desenvolver seus próprios beans de entidade, recomenda-se que não seja utilizado o recurso de associação do VisualAge for Java. Essa recomendação é feita a fim de minimizar a complexidade no modelo do objeto. Em vez de utilizar o recurso de associação fornecido pelo VisualAge for Java, o relacionamento de objeto entre os beans corporativos pode ser estabelecido incluindo métodos de obtenção explícitos nos beans corporativos.

O WebSphere Commerce fornece dois conjuntos de beans corporativos: particular e público. Os beans corporativos particulares são utilizados pelo ambiente e ferramentas do WebSphere Commerce runtime. Você *não deve* utilizar ou modificar esses beans.

Os beans corporativos públicos, de outro modo, são utilizados pelos aplicativos comerciais e podem ser utilizados e estendidos. Esses beans corporativos públicos são organizados nos seguintes grupos EJB:

- WCSActrlEJBGroup
- WCSApproval
- WCSAuction
- WCSCatalog
- WCSCommon
- WCSContract
- WCSCoupon
- WCSFulfillment
- WCSInventory
- WCSMessageExtensions
- WCSOrder
- WCSOrderManagement
- WCSOrderStatus
- WCSPayment
- WCSPVCDevices
- WCSTaxation
- WCSUserTraffic
- WCSUser
- WCSUTF

Alguns dos grupos EJB listados acima contém beans de sessão. Para, no futuro, simplificar a migração, a classe de beans de sessão não deveria ser modificada. Se for exigido, você poderá criar um novo bean de sessão em um novo grupo EJB. Para obter mais informações sobre a criação de novos beans de sessão, consulte "Escrever novos beans de sessão" na página 77.

Descritores de Implementação para os Beans de Entidade do WebSphere Commerce

Um descritor de implementação é uma classe especial que é serializada e que contém definições runtime para um bean corporativo. Os descritores de implementação dos beans corporativos do WebSphere Commerce são definidos de maneira específica e não devem ser modificados.

Ao criar novos beans corporativos (beans de entidade ou de sessão), defina os descritores de implementação do EJB Development Environment (do VisualAge for Java), clicando com o botão direito no bean e selecionando **Propriedades**. Os descritores de implementação para os novos beans corporativos devem seguir as mesmas convenções que as dos beans corporativos do WebSphere Commerce. Em especial, certifique-se de definir os atributos como segue:

Atributo	Valor
Atributo de Transação	`TX_REQUIRED`
Nível de Isolamento	`TRANSACTION_READ_COMMITTED`
Modo Executar-Como	`SYSTEM_IDENTITY` or `CLIENT_IDENTITY`
Reentrante	Assegure-se de que não esteja selecionado.

Um bean corporativo freqüentemente contém métodos que somente lêem informações do banco de dados, mas nunca atualizam bancos de dados. Esses métodos são conhecidos como *Apenas para Leitura*. Todos os métodos Apenas para Leitura devem ser explicitamente marcados como tal (clique no método com o botão direito do mouse e selecione **Atributos do Método EJB > Método Apenas para Leitura**). Se os métodos apenas para leitura não estiverem marcados dessa maneira, o contêiner EJB tentará desnecessariamente atualizar o banco de dados no final de uma transação e provocará um erro de retorno de transação na transação apenas para leitura. Isso provoca problemas no desempenho.

Níveis de Isolamento

O nível de isolamento da transação utilizado para os beans corporativos do WebSphere Commerce dentro do VisualAge for Java é `TRANSACTION_READ_COMMITTED`. Observe que há uma diferença entre a implementação desse nível de isolamento para o driver JDBC para DB2 e o

driver JDBC para Oracle. Sendo assim, o nível de isolamento utilizado ao implementar no ambiente WebSphere Application Server depende do banco de dados que está sendo utilizado.

O nível de isolamento utilizado para bancos de dados DB2 ao operar fora do WebSphere Test Environment é `TRANSACTION_REPEATABLE_READ`. O nível de isolamento utilizado para bancos de dados Oracle ao operar fora do WebSphere Test Environment é `TRANSACTION_READ_COMMITTED`.

Se você estiver implementando em um banco de dados DB2, não será necessário alterar manualmente o nível de isolamento da transação, pois ele é alterado durante a etapa em implementação quando você emite o comando `modifyIsolationLevel`.

A tabela a seguir apresenta um mapeamento dos níveis de isolamento de transação do DB2 e do Oracle para o nível de isolamento de transação do JDBC correspondente.

JDBC	**DB2**	**Oracle**
Leitura Não Consolidada	Leitura Não Consolidada	Leitura Não Consolidada
Leitura Consolidada	Estabilidade do Cursor	(Não aplicável)
Leitura que Pode Ser Repetida	Estabilidade da Leitura	Leitura Consolidada
Seriável	Leitura que Pode Ser Repetida	Seriável
Nenhum	(Não aplicável)	(Não aplicável)

Para o nível de isolamento da transação Estabilidade do Cursor no DB2, somente as linhas que foram atualizadas durante a transação determinada serão bloqueadas exclusivamente. Se nenhuma coluna de uma determinada linha for atualizada, mesmo que ela seja parte do conjunto de resultados retornado por uma instrução SQL, o bloqueio daquela linha específica será liberado quando o cursor for movido para outra linha. Em algumas situações, como na atualização do estoque, isso pode não ser um comportamento desejado. Portanto, alterando o nível de isolamento da transação para Estabilidade de Leitura no DB2, com um leve equilíbrio de concorrência, a integridade dos dados pode ser melhorada significativamente.

No caso do Oracle, em que apenas o nível de isolamento da transação Leitura Consolidada, ou o equivalente Leitura Que Pode Ser Repetida de JDBC, está disponível, a implementação real é feita de maneira que obtenha o comportamento mais semelhante ao do nível de isolamento de transação Leitura Que Pode Ser Repetida no DB2.

Estendendo o Modelo de Objeto do WebSphere Commerce

O modelo de objeto do WebSphere Commerce pode ser estendido das seguintes maneiras:

- Estender os beans corporativos públicos do WebSphere Commerce
- Escrever um novo bean de entidade
- Escrever um novo bean de sessão sem estado

Os detalhes sobre como executar estas extensões estão contidos nas seguintes seções.

Metodologias de extensão de modelo de objeto

Os requisitos do aplicativo podem fazer com que você estenda o modelo de objeto do WebSphere Commerce existente. Um exemplo de um requisito desse tipo é a inclusão de atributos adicionais em seu aplicativo. Isso pode ser realizado em uma das seguintes maneiras:

Sem modificar um bean de entidade público do WebSphere Commerce existente

Crie uma nova tabela de banco de dados e crie um novo bean de entidade para essa tabela. Inclua campos e métodos no bean de entidade para manipular o novo atributo, conforme necessário. Gere código implementado e um bean de acesso para o novo bean de entidade. Quando o aplicativo requerer o novo atributo, ele instanciará um objeto de bean de acesso e utilizará seus métodos para recuperar, definir ou manipular o atributo.

Modificando um bean de entidade público do WebSphere Commerce

Crie uma nova tabela de banco de dados e crie uma união de tabelas entre a nova tabela e a tabela existente que corresponde ao bean corporativo existente que você está modificando. Crie novos campos no bean de entidade público existente do WebSphere Commerce e mapeie os campos para suas colunas correspondentes na nova tabela, utilizando um mapa de tabela secundário. Inclua todos os métodos necessários. Gere novamente o código implementado e o bean de acesso para o bean de entidade existente. Os novos atributos estarão disponíveis quando o aplicativo instanciar o objeto de bean de acesso.

Há vantagens e desvantagens entre essas duas abordagens. Em geral, elas estão relacionadas ao desempenho e ao esforço de manutenção do código.

Exemplo de extensão: Considere um exemplo em que o aplicativo requer que você obtenha o tipo de residência que o cliente possui. Você cria uma tabela denominada USERRES que contém o ID dos clientes e o tipo de residência, em que o tipo de residência (resType) pode ser uma propriedade livre, um condomínio ou um apartamento. Esse tipo de informação é um dado pessoal e sendo assim está relacionado à tabela USERDEMO do Commerce Suite. Ao examinar o repositório de códigos do WebSphere

Commerce, você descobre que o grupo EJB WCSUser contém um bean corporativo "Demographics". Esse bean tem os getters e os setters de informações de dados pessoais armazenadas na tabela USERDEMO.

Há duas opções para executar a personalização. Você pode criar um novo bean de entidade que interaja com a tabela USERRES ou incluir um novo campo (mais os métodos apropriados de getter e setter) para o bean de dados pessoais.

Utilizando a primeira abordagem (criação de código completamente novo), você cria um novo bean Userres e mapeia seus campos para as colunas da tabela USERRES. Quando o aplicativo precisar do tipo de residência do cliente, ele deve instanciar um objeto de bean de acesso Userres e recuperar os dados. Se o aplicativo precisar de outras informações de dados pessoais ao mesmo tempo, ele também deve instanciar um objeto de bean de acesso de Dados pessoais e recuperar quaisquer outros atributos necessários. Quaisquer partes da lógica do aplicativo que tentem recuperar um conjunto completo de informações de dados pessoais de um cliente devem ser modificadas para instanciar o novo bean de acesso bem como o original. O diagrama a seguir exibe essa abordagem para estender o modelo de objeto:

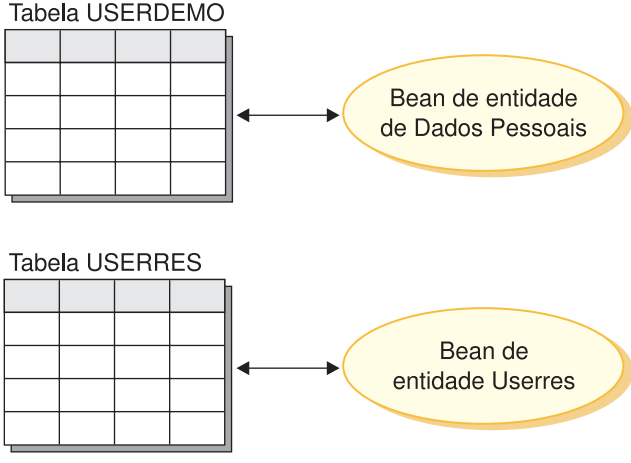


Figura 13.

De uma perspectiva de modelo de exibição, um bean de dados deve ser capaz de acessar o novo atributo, de forma que as informações estejam disponíveis para modelos JSP. Para apresentar uma visão unificada para o desenvolvedor da Web que cria os modelos JSP, você deve criar um novo bean de dados que estenda o bean de acesso para o bean de entidade original existente. O bean de dados também deve utilizar delegação para preencher os atributos a partir

do novo bean de acesso. O diagrama a seguir exibe esse cenário de implementação de bean de dados:

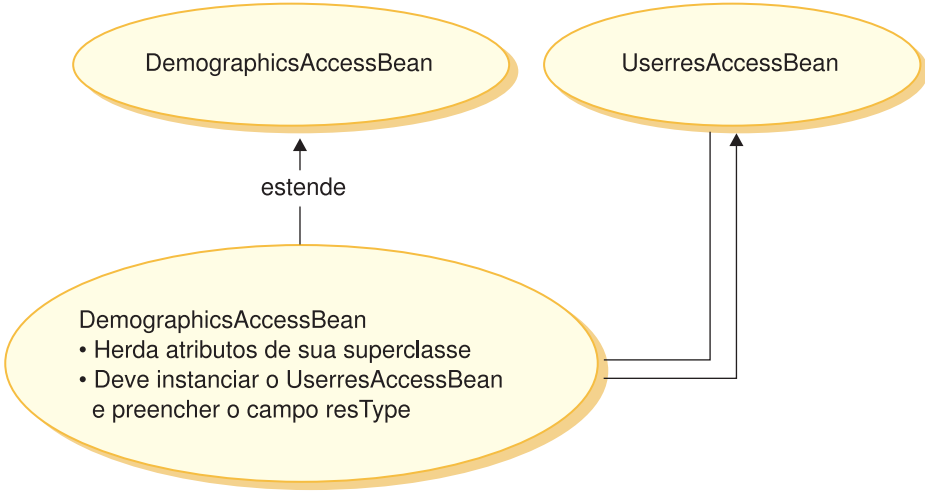


Figura 14.

Utilizando a segunda abordagem (modificação do código existente), você inclui um novo campo no bean de entidade Dados pessoais e cria um mapa secundário de tabelas entre o novo campo e a coluna apropriada na tabela USERRES. Quando o aplicativo precisar do tipo de residência do cliente, ele instanciará um objeto de bean de acesso de Dados Pessoais e recuperará o tipo de residência. Se o aplicativo precisar de quaisquer informações de dados demográficos sobre o cliente, elas estarão disponíveis na mesma chamada para o bean. O seguinte diagrama exibe essa abordagem para a modificação do bean corporativo:

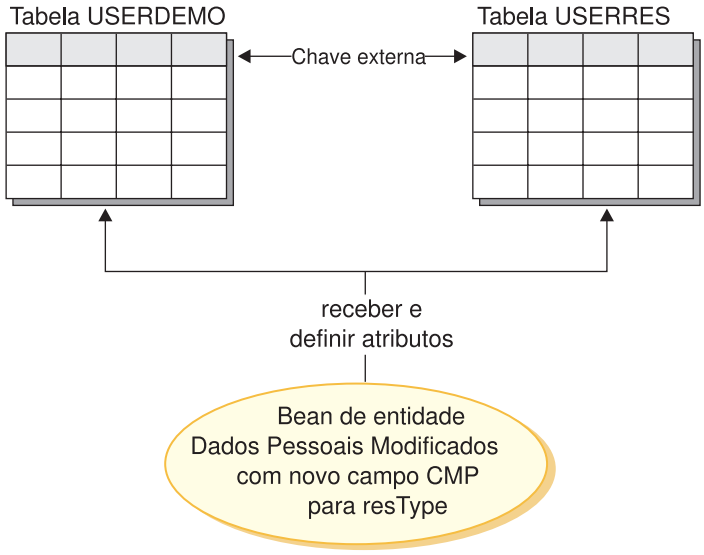


Figura 15.

De uma perspectiva do modelo de exibição, o novo atributo (resType) fica automaticamente disponível no bean de dados, assim que o DemographicsAccessBean é gerado novamente.

Observe que quando o modelo de objeto é estendido, *não* se deve incluir novas colunas nas tabelas existentes do banco de dados do WebSphere Commerce. Você deve criar uma nova tabela para o novo atributo. Se você tentar incluir novas colunas nas tabelas existentes, o novo atributo será perdido quando você migrar para futuros releases do WebSphere Commerce.

Implicações no desempenho e na manutenção do código: A segunda abordagem tem melhor desempenho de runtime. Isso acontece porque a obtenção ou a definição do novo atributo requerem apenas a instanciação de um bean de entidade único e uma única busca é utilizada para recuperar todos os atributos necessários.

Como a segunda abordagem modifica o código existente no WebSphere Commerce, surge um problema de migração quando um novo repositório de códigos WebSphere Commerce é liberado. Você deve combinar o código personalizado com o novo código, mas ao importar o novo repositório de códigos, as informações de mapeamento entre os campos incluídos no bean corporativo e a nova tabela não são preservadas. Por isso, ao migrar para um novo release de repositório de códigos do WebSphere Commerce, as seguintes etapas devem ser executadas:

1. Crie versão para o código EJB personalizado.

2. Importe a nova versão de código do WebSphere Commerce.
3. Utilize as ferramentas do VisualAge for Java, compare a versão personalizada de código com o novo release de código do WebSphere Commerce. Combine o código personalizado em sua área de trabalho.
4. Mapeie manualmente, outra vez, todos os atributos incluídos nos beans corporativos públicos do WebSphere Commerce para as colunas apropriadas no banco de dados.
5. Gere novamente o código implementado e os beans de acesso para os beans corporativos modificados na etapa 4.

Para simplificar essa migração, é importante documentar completamente suas extensões de modelos de objetos durante o desenvolvimento.

Você pode optar por utilizar uma mistura das duas abordagens ao fazer muitas extensões para o modelo de objeto. É possível utilizar a primeira abordagem para áreas do sistema menos suscetíveis à degradação no desempenho e utilizar a segunda quando o desempenho for um problema. Dessa forma, você pode minimizar o esforço de migrações futuras e, ao mesmo tempo, manter bons níveis de desempenho dos sistema.

Uso recomendado dos beans de sessão

Uma das forças do WebSphere Commerce se origina da habilidade de beneficiar-se de beans de entidade CMP (container-managed persistence). Os beans de entidade CMP são componentes Java distribuídos, persistentes, transacionais, do lado do servidor que podem ser gerados pelas ferramentas fornecidas com o VisualAge for Java. Em muitos casos, os beans de entidade CMP são uma opção extremamente boa para persistência do objeto e podem ser feitos para trabalhar de maneira tão eficiente ou mais que outras opções de mapeamento de objeto para relacional. Por esses motivos, o WebSphere Commerce implementou objetos core commerce utilizando beans de entidade CMP.

No entanto, há algumas situações em que recomenda-se utilizar um auxiliar JDBC de bean de sessão. Essas situações incluem:

- Um caso em que uma consulta retorna um conjunto grande de resultados. Isso é referenciado como o caso de *conjunto grande de resultados*.
- Um caso em que uma consulta recupera dados de várias tabelas. Isso é referenciado como o caso de *caso de entidade agregada*.
- Um caso em que uma instrução SQL executa uma operação intensiva no banco de dados. Isso é referenciado como o caso de *SQL arbitrário*.

Mais detalhes são fornecidos nas seguintes seções.

Observe que, se o bean de sessão estiver sendo utilizado como um wrapper JDBC para recuperar informações do banco de dados, será mais difícil

implementar controle de acesso em nível de recurso. Quando um bean de sessão é utilizado dessa maneira, o desenvolvedor do bean de sessão deve incluir as cláusulas "`where`" apropriadas na instrução "`select`" para prevenir que usuários não autorizados acessem recursos.

Caso de Conjunto Grande de Resultados: Há casos em que uma consulta retorna um conjunto grande de resultados e os dados recuperados são principalmente de leitura ou para exibição. Nesse caso, é melhor utilizar um bean de sessão sem estado e dentro desse bean de sessão criar um método finder que execute as mesmas funções que um método finder em um bean de entidade. Ou seja, o método finder no bean de sessão sem estado deve executar o seguinte:

- Executar uma instrução select de SQL.
- Para cada linha obtida, instanciar um bean de acesso.
- Para cada coluna recuperada, definir os atributos correspondentes no bean de acesso.

Quando o bean de acesso for retornado, o comando não saberá se o bean de acesso foi retornado por um método finder em um bean de sessão ou por um método finder em um bean de entidade. Como resultado, utilizar um método finder em um bean de sessão não causa alterações no modelo de programação. Apenas o comando que faz a chamada sabe se está chamando um método finder em um bean de sessão ou em um bean de entidade. Fica transparente para todas as outras partes do modelo de programação.

Caso de Entidade Agregada: Nesse caso, uma exibição combina partes de vários objetos e uma única página de exibição é ocupada com partes de informações oriundas de várias tabelas do banco de dados. Por exemplo, considere o conceito de "My Account". Pode consistir em informações da tabela de informações sobre o cliente (por exemplo, o nome, idade e ID do cliente) e de informações de uma tabela de endereço (por exemplo, um endereço composto de nome da rua e cidade).

É possível construir uma instrução SQL simples para recuperar todas as informações de várias tabelas executando um SQL join. Isso pode ser referenciado como a execução de uma "busca profunda". O seguinte é um exemplo de uma instrução SQL select para o exemplo "My Account", em que a tabela CUSTOMER é T1 e a tabela ADDRESS é T2:

```
select T1.NAME, T1.AGE, T2.STREET, T2.CITY
  from CUSTOMER T1, ADDRESS T2
  where (T1.ID=? and T1.ID=T2.ID)
```

A ferramenta EJB no VisualAge for Java não suporta essa concepção de busca profunda. Em vez disso, ela faz uma busca leve que resulta em um SQL select para cada objeto associado. Esse não é o método preferido para recuperar esse tipo de informações.

Para executar uma busca profunda, recomenda-se utilizar um bean de sessão. Nesse bean de sessão, crie um método finder para recuperar as informações obrigatórias. O método finder deve fazer o seguinte:

- Executar uma instrução SQL select para a busca profunda.
- Instanciar um bean de acesso para cada linha na tabela principal bem como para cada objeto associado.
- Para cada coluna e para cada objeto associado obtido, defina o atributo correspondente no bean de acesso.

Observe que um bean de acesso não coloca em cache um método getter que lance uma exceção. Nesse caso, é necessário criar uma classe wrapper simples para o bean de acesso utilizando o seguinte padrão:

```
public class CustomerAccessBeanCopy extends CustomerAccessBean {
   private AddressAccessBean address=null;

   /* The following method overrides the getAddress method in
      CustomerAccessBean.
   */
   public AddressAccessBean getAddress() {
      if (address == null)
         address = super.getAddress();
      return address;
      }

    /* The following method sets the address to the copy. */

    public void _setAddress(AddressAccessBean aBean) {
       address = aBean;
      }
}
```

Continuando o exemplo de CUSTOMER e ADDRESS, o método finder do bean de sessão instanciará um CustomerAccessBean para cada linha na tabela CUSTOMER e um AddressAccessBean para cada linha correspondente na tabela ADDRESS. Em seguida, para cada coluna na tabela ADDRESS, ele define os atributos no AddressAccessBean (rua e cidade). Para cada coluna na tabela ADDRESS, ele define os atributos no CustomerAccessBean (nome, idade e endereço). Isso é mostrado no seguinte diagrama.

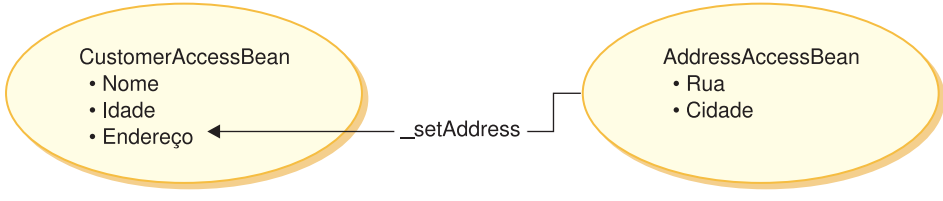


Figura 16.

Caso de SQL Arbitrário: Nesse caso, existe um conjunto de instruções SQL arbitrárias que executam operações intensivas no banco de dados. Por exemplo, a operação para somar todas as linhas em uma tabela deve ser considerada uma operação intensiva do banco de dados. É possível que nem todas as linhas selecionadas correspondam a um bean de entidade no modelo persistente.

Um exemplo que pode resultar na criação de uma instrução SQL arbitrária é quando um cliente tenta navegar por um conjunto muito grande de dados. Por exemplo, se o cliente desejar examinar todos os grampeadores em uma loja de hardware online ou todos os vestidos em uma loja de roupas online. Isso criará um conjunto de resultados muito grande. É muito provável que apenas alguns campos de cada linha sejam necessários. Ou seja, inicialmente o cliente pode receber apenas um resumo que mostre o nome, a fotografia e o preço do item.

Nesse caso, crie um método helper para o bean de sessão. Esse método helper executará uma operação de leitura ou de gravação. Ao executar uma operação de leitura, ele retornará um objeto de valor apenas para leitura utilizado para fins de exibição.

Normalmente, com a modelagem apropriada de dados, o número de casos de instruções SQL arbitrárias por ser minimizada.

Extensão dos beans de entidade públicos

Essa seção descreve o padrão de design dos beans corporativos públicos do WebSphere Commerce. Esse padrão de design permite fazer extensões de como incluir novos campos persistentes, novos métodos de negócios ou novos métodos descobridores.

O diagrama a seguir mostra as classes de implementação do bean de entidade de Catálogo.

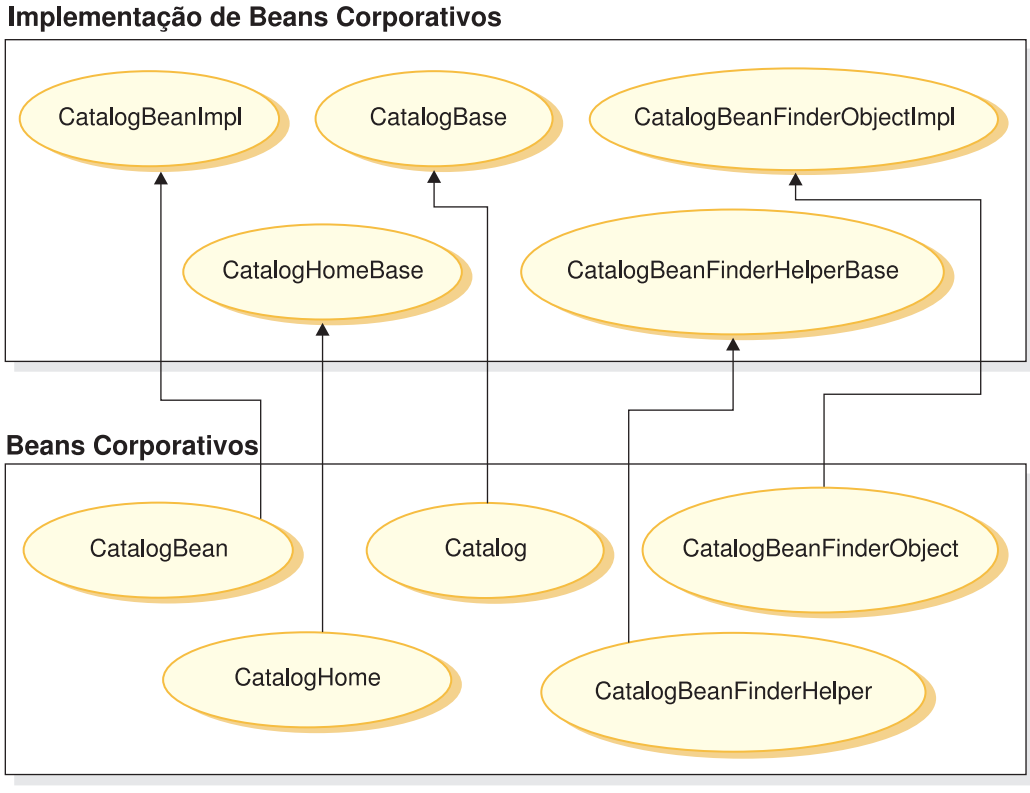


Figura 17.

O diagrama acima também se aplica a outros beans de entidade, porque eles estão estruturados de forma similar e seguem as mesmas convenções de nomenclatura. Para aplicar o diagrama a outro bean de entidade, substitua o nome do bean de entidade por "Catálogo". Por exemplo, a classe InterestItemBean estende a classe InterestItemBeanImpl e a interface InterestItem estende a interface InterestItemBase.

O diagrama mostra que a classe de implementação ou a interface dos beans corporativos públicos foi separada em duas partes, utilizando a herança Java. A superclasse ou a interface contém o código de implementação do WebSphere Commerce. Todas essas superclasses e interfaces estão definidas em pacotes e projetos separados das classes filha e interfaces.

Com a exceção das classes *bean_name*`BeanFinderHelperBase` e *bean_name*`BeanFinderObjectImpl`, o repositório de código do WebSphere Commerce contém código binário para todas essas superclasses e interfaces. O código fonte das classes *bean_name*`BeanFinderHelperBase` e

*bean_name*BeanFinderObjectImpl está incluído, para que você possa ver como a instrução SQL de cada localizador está definida. Você não deve, de forma alguma, modificar essas classes.

Para examinar o código fonte de CatalogBeanFinderHelperBase e CatalogBeanFinderObjectImpl, abra o pacote com.ibm.commerce.catalog.objsrc. Outros pacotes utilizam uma convenção de nomenclatura similar.

As modificações podem ser feitas nas classes filha e nas interfaces.

Se você incluir novos métodos de busca nos beans corporativos públicos, será necessário seguir uma convenção de nomenclatura específica para os métodos. Denomine os novos métodos findX*a_description* em que *a_description* é uma descrição de sua escolha. Alguns exemplos de nomes são findXByOwnerId e findXByOrderStatus. Utilizar esta convenção de nomenclatura evita o risco de conflito de nomes (nomes duplicados) com os métodos de busca do WebSphere Commerce.

Uma maneira de modificar um bean de entidade público existente no WebSphere Commerce é incluir campos adicionais. Nesse caso, depois de incluir os novos campos, você deve examinar cada método finder no bean. Se a parte da cláusula where dos métodos de busca tiver quaisquer aliases de banco de dados (por exemplo, T1. ou T2.), os aliases deverão ser removidos.

Criando um Novo Bean Corporativo CMP

Quando você tem um novo atributo que precisa ser incluído no modelo do objeto do WebSphere Commerce, pode criar uma nova tabela de banco de dados com uma coluna para o atributo obrigatório. Em seguida, será necessário incluir também esse atributo em um bean corporativo, de forma que os comandos do WebSphere Commerce possam acessar as informações.

Uma maneira de integrar o novo atributo ao modelo do objeto do WebSphere Commerce é criando um novo bean corporativo CMP. Nesse bean, você deverá, então, criar um campo que corresponda ao atributo na nova tabela de banco de dados.

Para criar um novo bean corporativo CMP, será necessário executar as seguintes etapas no VisualAge for Java:

1. Assegure-se de que o proprietário da área de trabalho esteja definido para Desenvolvedor do WCS.
2. Crie um novo grupo EJB para o bean.
3. Crie o novo bean corporativo CMP, utilizando a ferramenta SmartGuide Criar Bean Corporativo.

4. Para cada coluna da tabela de banco de dados correspondente, inclua um novo campo CMP no bean.
5. Se for necessário, crie um par de métodos getter e setter para cada um dos campos CMP criados.
6. Se for necessário, defina os campos FinderHelper na interface FinderHelper e inclua o novo método FinderHelper.
7. Crie um novo método ejbCreate, se for necessário, e promova o método ejbCreate para a interface inicial do bean corporativo. Essa etapa é obrigatória se o novo bean corporativo precisar criar novas entradas em sua tabela de banco de dados correspondente.
8. Mapeie os campos do bean corporativo para as colunas da tabela de banco de dados.
9. Gere o bean de acesso e o código implementado para o bean corporativo.

Mais detalhes de cada uma dessas etapas estão contidos nas seções a seguir.

Nota: Se o novo bean corporativo deverá ser protegido pelo sistema de controle de acesso do WebSphere Commerce, consulte Capítulo 4, "Controle de Acesso" na página 91 para obter mais detalhes. O controle de acesso pode ser incluído depois que você criar o bean.

Suponha que você tenha uma nova tabela chamada USERRES que especifica algumas informações sobre o tipo de residência que um usuário tem. Essa tabela contém três colunas: uma coluna USERID, uma coluna HOME, que especifica o tipo de residência, e uma coluna ROOMS, que especifica o número de quartos da residência.

Criando um Novo Grupo EJB: Ao criar novos beans de entidade, você deve criá-los em um grupo EJB que esteja separado dos grupos EJB do WebSphere Commerce. Ao trabalhar com beans corporativos no VisualAge for Java, você deve ir para a guia EJB da área de trabalho. Em seguida, no menu **EJB**, é possível selecionar **Incluir > Grupo EJB** para lançar o SmartGuide Incluir Grupo EJB.

Há dois itens importantes que devem ser especificados ao criar o grupo EJB: o projeto em que o código EJB está armazenado e o nome do grupo EJB.

O projeto EJB pode ser exibido na guia Projetos da área de trabalho. Ao criar um novo grupo EJB, é necessário especificar um projeto que seja separado dos projetos do WebSphere Commerce. Por exemplo, você pode selecionar que seu grupo EJB utilize o projeto MyCustomEJB. Esse projeto não precisa existir antes de você criar o grupo, já que o VisualAge for Java pode criá-lo automaticamente. Esse projeto deve ser utilizado somente para o código EJB; não deve ser utilizado para qualquer comando ou código de beans de dados. Essa separação de tipos de códigos é exigida com a finalidade de

implementação. A separação de seu próprio código personalizado do código do WebSphere Commerce, minimizará o impacto de migração para releases futuros.

Para os nomes do projeto e do grupo de EJB, certifique-se de seguir as convenções de nomenclatura apropriadas de seu aplicativo.

Criando o Novo Bean Corporativo CMP: Para criar seu novo bean corporativo CMP, você pode utilizar a ferramenta SmartGuide Criar Bean Corporativo. Para lançar a ferramenta, clique com o botão direito no grupo EJB no qual incluirá o novo bean e selecione **Incluir > Bean Corporativo**.

Selecione criar um novo bean corporativo e especifique um nome para o bean. A convenção de nomenclatura do WebSphere Commerce para beans corporativos é nomear o bean utilizando o mesmo nome da tabela à qual o bean corresponde. Por exemplo, se sua nova tabela de banco de dados for denominada USERRES, o bean corporativo deve ser denominado UserRes.

O campo Projeto é preenchido automaticamente com o nome de seu projeto. É necessário especificar um nome de pacote no projeto no qual o código deve ser armazenado. Um exemplo de código a ser armazenado no pacote é o código do bean de acesso que você criou para o bean corporativo. Mais uma vez, ao nomear o pacote, certifique-se de seguir as convenções de nomenclatura apropriadas para seu aplicativo. Um exemplo de nome de pacote é `com.mycompany.mycustombeans`.

Na classe bean, o VisualAge for Java cria o campo privado denominado `EntityContext`. O WebSphere Commerce fornece seu próprio campo de contexto de entidade em ECEntityBean e seu novo bean de entidade deve utilizar esse campo, em vez do campo gerado em sua própria classe. Dessa forma, você deve remover o `EntityContext` gerado de seu novo bean de entidade.

O novo bean corporativo deve conter um campo `serialVersionUID`. Se o SmartGuide for utilizado para criar o novo bean, o VisualAge for Java gerará esse campo. Caso contrário, você deverá incluir esse campo.

No campo Superclasse, é necessário especificar a classe `com.ibm.commerce.base.objects.ECEntityBean`. O exemplo de código a seguir demonstra as funções fornecidas pela superclasse:

```
public class myEJB extends com.ibm.commerce.base.objects.ECEntityBean {
   public void ejbLoad()throws java.rmi.RemoteException {
      super.ejbLoad();--the super method will add EJB trace
       --your logic --
   }
    public void ejbStore() throws java.rmi.RemoteException {
```

```
      super.ejbStore();--the super method will add EJB trace
      --your logic --
   }
}
```

Você também deve remover os métodos ejbCreate() e ejbPostCreate() que não tiverem argumentos. Deixar esses métodos no bean de entidade pode causar erros de runtime.

Para cada coluna de sua tabela de banco de dados, é necessário criar um novo campo CMP no bean (é necessário clicar em Avançar no SmartGuide para ver a opção Incluir campos CMP no bean). Para cada campo, especifique um nome de campo e o tipo de dados do campo. Para qualquer coluna com a chave principal ou parte da chave principal, ative a caixa de opção Campo de Chave. Para todas as outras colunas, ative a caixa de opção Promover os métodos getter e setter para a interface remota.

Depois de todos os campos serem preenchidos, clique em Concluir e o VisualAge for Java criará o novo bean corporativo CMP.

Criando Novos Campos FinderHelper na Interface *Bean_Name*`FinderHelper`: A interface *Bean_Name*`FinderHelper` contém cláusulas de pesquisa SQL que correspondem a todos os métodos FinderHelper diferentes do método findByPrimaryKey.

Um novo bean corporativo utiliza métodos `findXBy`*ArgName* (em que *ArgName* é o nome de um argumento) definidos na interface FinderHelper do bean juntamente com os métodos FinderHelper para compor consultas SQL. Utilize a convenção de nomenclatura “findXBy” para o nome do campo para garantir que os nomes de campos sejam sempre exclusivos em relação aos nomes de campos do WebSphere Commerce.

Para criar os novos campos do FinderHelper no bean, é necessário selecionar a interface *Bean_Name*`_FinderHelper` e modificar o código de origem para estabelecer como as instruções select devem ser formadas. Segue um exemplo:

```
public interface UserResFinderHelper {
 public static final String findXByHomeAndRoomsWhereClause = "(T1.HOME = ?
      and T1.ROOMS = ?)";
}
```

A interface inicial exigirá, então, um método denominado `findXByHomeAndRooms` que utiliza parâmetros de entrada para cada HOME e ROOMS que ocupar os valores representados pelos caracteres ? . Este tipo de construção de consulta é referido como uma *inserção de parâmetro*.

Se os parâmetros de entrada forem ″detached″ e ″3″, a instrução SQL gerada seria

```
select * from USERRES where HOME=detached and ROOMS=3
```

Por motivos de segurança, ao criar métodos FinderHelper para um novo bean de entidade, você deve utilizar inserções de parâmetro. O motivo desta recomendação é que ela evita que a consulta seja alterada pelos usuários. Uma abordagem alternativa seria utilizar uma construção semelhante à seguinte:

```
public static final String
     findXByOwnerIdWhereClause = " (T1.OWNERID = input_string) ";
```

em que *input_string* é um valor de cadeia transmitido a partir de uma URL. Isto não é desejável uma vez que um usuário mal intencionado poderia digitar um valor como "'123' OR 1=1" que altera a instrução SQL. Se um usuário puder alterar a instrução SQL, ele conseguirá acesso não autorizado aos dados. Portanto, a abordagem recomendada é utilizar inserções de parâmetros.

Se você não puder utilizar uma inserção de parâmetro e, assim, tiver que utilizar uma cadeia de entrada para compor a instrução SQL, será necessário executar a verificação de parâmetro na cadeia de entrada para garantir que o parâmetro de entrada não seja uma tentativa mal intencionada para acessar dados.

Criando Novos Métodos FinderHelper na Interface *Bean_Name*Home: Para cada campo FinderHelper especificado na interface *Bean_Name*FinderHelper, é necessário criar um método FinderHelper na interface inicial do bean. O nome dos métodos FinderHelper precisa corresponder ao nome do campo FinderHelper exatamente, exceto por "WhereClause" que é eliminado. Ou seja, com o exemplo de nome de campo findXByHomeAndRoomsWhereClause, o nome do método correspondente é findXByHomeAndRooms.

Para criar os novos métodos FinderHelper, faça o seguinte:

1. Clique com o botão direito na interface *Bean_Name***Home** e selecione **Incluir > Método**.
   O SmartGuide Criar Método abrirá.
2. Selecione **Criar um novo método** e clique em **Avançar**.
3. No campo **Nome do Método**, digite um nome para o método FinderHelper. Esse nome dos métodos FinderHelper deve corresponder ao nome do campo FinderHelper exatamente, exceto pela parte "WhereClause" que é eliminada. Por exemplo digite findXByHomeAndRooms.
4. No campo **Tipo de retorno**, digite um dos seguintes:
   - Se o método FinderHelper utiliza a chave principal para consultar o banco de dados e o método deve retornar um registro exclusivo, especifique o objeto EJB como o tipo de retorno. Por exemplo, digite UserRes.

- Se o método FinderHelper retornar um conjunto de resultados em vez de um registro exclusivo, especifique o tipo de retorno como `java.util.Enumeration`.

5. Clique no botão **Incluir** ao lado de **Quais parâmetros esse método deve ter?** para incluir os parâmetros apropriados. Por exemplo, você pode incluir `argHome` do tipo `Cadeia` para conter o tipo de residência e `argRooms` do tipo `byte` para conter o número de cômodos.
6. Quando tiver concluído a inclusão de todos os parâmetros, clique em **Avançar**.
7. Clique no botão **Incluir** ao lado de **Quais exceções esse método pode emitir?** e inclua as seguintes exceções:
   - `java.rmi.RemoteException`
   - `javax.ejb.FinderException`

   Nota: A lista de exceções anterior mostra o conjunto mínimo de exceções que o método deve lançar. Dependendo do código, pode ser necessário especificar outras exceções.
8. Clique em **Concluir**.

Criando um Novo Método ejbCreate: Quando o bean corporativo é criado, o método ejbCreate é gerado automaticamente. Esse método é, então, promovido para a interface remota, para que esteja disponível no bean de acesso. O método ejbCreate padrão contém somente parâmetros que são a chave principal ou parte da chave principal. Isso significa que somente são criadas instâncias desses valores em uma criação de instância.

Se seu bean corporativo contiver campos que não fazem parte da chave principal e são campos que não podem ser anulados, será necessário criar um novo método ejbCreate no qual você cria especificamente uma instância desses campos. Fazendo isso, toda vez que um novo registro for criado, todos os campos que não podem ser anulados serão preenchidos com os dados apropriados.

Para criar um novo método ejbCreate, faça o seguinte:

1. No painel Tipos, expanda a classe ***Bean_Name*Bean**. Por exemplo, selecione **UserResBean**.
2. Clique no método ejbCreate existente para exibir o código fonte. (Observe que pode ser ejbCreate(String), ejbCreate(String, int) ou pode utilizar alguns outros parâmetros de entrada, dependendo da chave principal de seu bean corporativo).
3. É necessário modificar o código fonte de forma que cada campo CMP seja incluído como um parâmetro de entrada do método e que seja criada uma

instância de cada campo CMP com o valor apropriado. No exemplo UserRes, em que o UserId é a chave principal, o código fonte inicialmente aparece como:

```
public void ejbCreate(int argUserId)
   throws javax.ejb.CreateException, java.rmi.RemoteExeption {
   _initLinks();
   userId = argUserId;
}
```

Mas você pode querer assegurar que o número de cômodos e o tipo de residência sejam inicializados. Nesse caso, o código seria alterado para o seguinte:

```
public void ejbCreate(int argUserId, String argHome, byte Rooms)
   throws javax.ejb.CreateException, java.rmi.RemoteExeption {
   _initLinks();
   // All CMP fields should be initialized here
   userId = argUserId;
   home = argHome;
   rooms = argRooms;
}
```

Nota: Se desejar utilizar uma chave primária gerada pelo sistema, consulte "Chaves Principais" na página 82 para obter detalhes.

4. Salve o método modificado. Ao salvar o código, o VisualAge for Java cria um novo método ejbCreate que utiliza os novos parâmetros. O método ejbCreate original permanece.
5. Exclua o método original ejbCreate (o que não tem argumentos).
6. Clique com o botão direito no novo método ejbCreate e selecione **Incluir > Interface Inicial do EJB**.
7. Crie um método ejbPostCreate correspondente. (Esse método não precisa ser incluído na interface inicial).

Mapeando a Tabela de Banco de Dados para o Novo Bean Corporativo:
Depois de criar o novo bean corporativo, será necessário criar um mapeamento entre os campos CMP do bean e as colunas da tabela de banco de dados. Esse mapeamento é chamado de esquema. O VisualAge for Java fornece ferramentas para simplificar essa tarefa.

Para criar o esquema, faça o seguinte:

1. No menu **EJB**, selecione **Abrir Em > Esquemas do Banco de Dados**. O Navegador do Esquema abrirá.
2. No menu **Esquemas**, selecione **Importar / Exportar Esquema > Importar Esquema do Banco de Dados**.
3. Digite um nome para o novo esquema e clique em **OK**.
4. Na janela Conexão do banco de dados, digite as informações da seguinte forma:

Atributo	Valor do DB2	Valor do Oracle
Tipo da Conexão	`COM.ibm.db2.jdbc.app.DB2Driver`	`Oracle.jdbc.driver.OracleDriver`
Origem de Dados	`jdbc:db2:wc_database_name`	`jdbc:oracle:thin@`*`hostname`*`:`*`port`*`:`*`Oracle_SID`*
Nome do Usuário	*`wc_db_user_name`*	*`wc_db_user_name`*
Senha	*`wc_db_password`*	*`wc_db_password`*

com valores substituídos da seguinte forma:

- DB2 *`wc_database_name`* é o nome do seu banco do dados do WebSphere Commerce.
- Oracle *`hostname`* é o nome do host do servidor Oracle.
- Oracle *`port`* é o número da porta do banco de dados Oracle.
- Oracle *`Oracle_SID`* é o ID da instância do Oracle.
- *`wc_db_user_name`* é o nome do usuário do banco de dados.
- *`wc_db_password`* é a senha do banco de dados.

5. Na lista **Qualificadores**, selecione o qualificador de seu banco de dados (pode ser o nome do usuário de seu banco de dados).
6. Clique em **Construir Lista de Tabelas**.
7. Selecione *your_new_table* (na lista gerada e clique em OK para gerar o esquema.
8. Após o esquema ser gerado, clique em *your_new_table* no painel Esquema.
9. Destaque e em seguida, dê um clique duplo em *your_new_table* no painel Tabela.
   A janela Editor de Tabelas abrirá.
10. Remova as informações do campo **Qualificador**. Isso torna seu bean corporativo portátil para outras máquinas.
11. Do menu **Esquemas**, selecione **Salvar Esquema**. Digite os nomes de projeto, pacote e classe apropriados.

Em seguida, você precisa criar o mapa do esquema. O mapa do esquema é um mapeamento entre colunas e campos no bean corporativo.

Para criar o mapa do esquema, faça o seguinte:

1. No menu **EJB**, selecione **Abrir Em > Mapas de Esquema**.
   A janela Mapa Datastore abrirá.

2. Digite o nome do mapa. Consulte "Considerações sobre Nomenclatura de Objetos de Esquema do Banco de Dados" na página 85 para obter recomendações sobre como nomear o novo mapa.
3. Selecione seu grupo EJB e esquema. Consulte "Considerações sobre Nomenclatura de Objetos de Esquema do Banco de Dados" na página 85 para obter recomendações sobre como nomear o novo esquema.
4. No painel Mapas Datastore, selecione seu mapa.
5. No painel Classes persistentes, selecione sua classe.
6. No menu **Mapas da Tabela**, selecione **Novo Mapa da Tabela > Incluir Mapa da Tabela sem Herança**.
7. Na lista suspensa **Tabela**, selecione sua tabela e clique em **OK**.
8. No painel Mapas da Tabela, destaque e em seguida, clique com o botão direito em seu mapa da tabela e selecione **Editar Mapas de Propriedade**. O Editor de Mapas de Propriedade abrirá.
9. Para cada um dos atributos de classe (os campos CMP do bean), é necessário especificar o tipo de mapa e a coluna do banco de dados para a qual deve ser mapeado. Por exemplo, você deve mapear o atributo de classe UserId utilizando um tipo de mapa ″Simples″ para a coluna USERID na tabela de banco de dados.
10. Depois de todos os campos terem sido mapeados para a coluna correspondente do banco de dados, será necessário salvar o mapa do esquema. No menu **Mapa do Banco de Dados**, selecione **Salvar Mapa Datastore**. Digite os nomes do projeto, pacote e classe apropriados para salvar o mapa. Clique em **Concluir**.

Criando o Bean de Acesso e Gerando o Código Implementado: Um bean de acesso age como um envólucro do bean corporativo que simplifica como outros componentes interagem com o bean corporativo. É necessário criar um bean de acesso para seu novo bean corporativo.

Ao gerar código implementado, as ferramentas do VisualAge for Java analisam o bean para assegurar que as regras especificadas nas especificações do EJB da Sun Microsystems EJB, assim como as regras específicas do servidor EJB, sejam atendidas.

Para criar um bean de acesso para seu novo bean corporativo, faça o seguinte:

1. No painel Beans Corporativos, clique com o botão direito em seu bean corporativo e selecione **Incluir > Bean de Acesso**. (Pode ser necessário expandir o grupo EJB que contém seu novo bean primeiro, para exibir o bean).
   O SmartGuide Criar Bean de Acesso abrirá.
2. Nos campos **Grupo EJB**, **Bean Corporativo** e **Nome do bean de acesso**, especifique o grupo EJB, o nome do bean e o nome do bean de acesso apropriados.

3. Selecione **Copiar Assistente para um Bean de Entidade para o Tipo de Bean de Acesso** e clique em **Avançar**.
4. Na lista suspensa **Selecionar método inicial para construtor de argumento zero**, selecione **findByPrimaryKey**.
5. Na lista suspensa **Conversor**, selecione **WCSStringConverter** para as propriedades iniciais e clique em **Avançar**.
6. Na janela Selecionar e Personalizar Propriedades de Beans para Copy Helper, selecione o **WCSStringConverter** para cada campo.
7. Clique em **Concluir**.

Para gerar o código implementado, faça o seguinte:

1. No painel Beans Corporativos, clique com o botão direito em seu novo bean corporativo e selecione **Gerar Código Implementado**.

Observe que o código implementado gerado com essa ferramenta é compatível com a especificação EJB 1.0 e é utilizado somente ao executar o bean corporativo no VisualAge for Java. Em um outro estágio, quando você implementar seu bean corporativo em um aplicativo do WebSphere Commerce em execução no WebSphere Application Server V4.0, será necessário que você gere um arquivo JAR contendo código implementado que seja compatível com as especificações EJB 1.1. Para obter mais informações sobre a criação deste arquivo EJB 1.1 Export JAR, consulte "Informações sobre Código EJB Implementado" na página 194 e "Gerando Código Implementado" na página 360.

Utilizando o Cliente de Teste para Testar o Bean Corporativo: O VisualAge for Java fornece um cliente de teste que pode ser utilizado para testar beans corporativos. Para utilizar o cliente de teste para testar seu novo bean, faça o seguinte:

1. Inicie o servidor EJB que contém o bean corporativo que você está testando.
2. Clique com o botão direito do mouse no bean corporativo e selecione **Executar Cliente de Teste**.
   As janelas Cliente de Teste EJB e Procura EJB abrirão.
3. No campo **Nome do JNDI**, digite o nome do JNDI do bean corporativo e clique em **Procurar**.
4. Clique com o botão direito do mouse no método **findByPrimaryKey** que tem os argumentos preenchidos e selecione **Chamar**.

Práticas de Codificação: As práticas de codificação do bean corporativo a seguir devem ser observadas:

- Não utilize o tipo de dados BLOB nem CLOB.
- Nenhum campo CMP do tipo de dados LONG (também conhecido como LONG VARCHAR) deve ser o primeiro nem o último membro da lista de

campos CMP do bean corporativo. Para verificar a lista, verifique o método `EJSJDBCPersister._hydrate()` e veja se o primeiro ou o último elemento da lista é do tipo LONG VARCHAR. Se o primeiro campo for desse tipo, faça o seguinte:

1. Remova a definição do primeiro campo. Chame-o de `fieldA`.
2. Remova a definição de outro campo que *não* tenha o tipo LONG VARCHAR. Chame-o de `fieldB`.
3. Redefina o `fieldA`.
4. Redefina o `fieldB`.
5. Abra o Navegador do Mapa do Esquema e edite o mapa da tabela para refletir essas alterações. Salve o mapa.
6. Gere novamente o código implementado.

- O código do bean corporativo não deve fazer referência a nada fora dos pacotes de beans corporativos. Por exemplo, você não deve fazer referência a comandos ou beans de dados no código do bean corporativo.
- Para ativar o controle de acesso para seu bean corporativo, inclua a interface com.ibm.commerce.security.Protectable e/ou a interface com.ibm.commerce.security.Groupable na interface remota do bean corporativo. Após incluir essas interfaces, gere novamente o código implementado e o bean de acesso do bean. Também é necessário criar um objeto da classe do assistente de acesso no pacote objsrc.

Criando um Bean de Dados Simples

Um bean de dados é um bean que é utilizado nos modelos JSP para recuperar informações do bean corporativo. Um bean de dados simples estende seu bean de acesso correspondente e implementa a interface SmartDataBean. A maior parte do código do bean de dados é gerada automaticamente pelo VisualAge for Java.

Para criar um bean de dados simples, é necessário executar as seguintes etapas:

1. Crie um projeto e um pacote para armazenar o código do bean de dados.
2. Crie um bean de dados que estenda o bean de acesso correspondente e implemente a interface do bean de dados apropriada.
3. Crie os métodos set para o bean de dados.
4. Crie os métodos get para o bean de dados.

Criando o Projeto e o Pacote para o Código do Bean de Dados: A criação de um projeto e de um pacote cria um local em que o código de beans de dados pode ser armazenado.

Para criar um novo projeto, proceda da seguinte forma:

1. Selecione a guia **Projetos**.

2. No menu **Selecionado**, selecione **Incluir > Projeto**.
   O SmartGuide Incluir Projeto é aberto.
3. Assegure-se de que **Criar um novo projeto denominado** esteja selecionado e digite um nome para o novo projeto. Por exemplo, digite `Meus Beans de Dados`.
4. Clique em **Concluir**.

Para criar um novo pacote, proceda da seguinte maneira:

1. Clique com o botão direito do mouse no projeto criado para o código de bean de dados e selecione **Incluir > Pacote**. Por exemplo, clique com o botão direito do mouse em **Meus Beans de Dados** e selecione **Incluir > Pacote**.
   O SmartGuide Incluir Pacote é aberto.
2. Verifique se **Criar um novo pacote nomeado** está selecionado e digite um nome adequado para o pacote de beans de dados. Por exemplo, digite `com.mycompany.mydatabeans`.
3. Clique em **Concluir**.

Criando um Bean de Dados: Um bean de dados é um bean Java que é utilizado dentro de um modelo JSP para fornecer conteúdo dinâmico para a página. Normalmente fornece uma representação simples (indiretamente) de um bean de entidade, estendendo um bean de acesso. O bean de dados encapsula propriedades que podem ser recuperadas de ou definidas no bean de entidade.

Para criar um bean de dados, faça o seguinte:

1. Clique com o botão direito no pacote no qual irá armazenar o bean de dados e selecione **Incluir > Classe**.
   O SmartGuide Criar Classe abrirá.
2. Os campos do nome do projeto e do pacote já estarão preenchidos.
3. Assegure-se de que **Criar uma nova classe** esteja selecionado e clique em **Avançar**.
4. No campo **Nome da Classe**, digite um nome para seu novo bean de dados. Por exemplo, para criar um bean de dados que estenda UserResAccessBean, digite `UserResDataBean`.
5. Para especificar a superclasse, clique em **Procurar**, em seguida, no campo de padrão, digite o nome do bean de acesso correspondente. Por exemplo, digite `UserResAccessBean` e clique em **OK**.
6. Clique em **Avançar**.
7. Para especificar as interfaces que o bean de dados deve implementar, clique em **Incluir**. Na janela Interface, proceda da seguinte maneira:
   a. No campo **Padrão**, digite `com.ibm.commerce.beans.SmartDataBean` e clique em **Incluir**.

b. No campo **Padrão**, digite `com.ibm.commerce.beans.InputDataBean` e clique em **Incluir**.
c. Clique em **Fechar**.

8. Clique em **Concluir**.

Incluindo Campos Obrigatórios no Bean de Dados: Esta seção descreve como incluir campos obrigatórios no novo bean de dados.

Para incluir o campo iCommandContext, proceda da seguinte maneira:

1. Clique com o botão direito do mouse no novo bean de dados (por exemplo, UserResDataBean) e selecione Incluir > Campo.
   O SmartGuide Criar Campo é aberto.
2. No campo **Nome do campo**, digite `iCommandContext`.
3. Clique em **Procurar** para incluir o tipo de campo e digite `com.ibm.commerce.command.CommandContext`. Clique em **OK**.
4. Para os modificadores de acesso, selecione **Protegido**.
5. Clique em **Concluir**.

Para incluir o campo iRequestProperties, proceda da seguinte maneira:

1. Clique com o botão direito do mouse no novo bean de dados (por exemplo, UserResDataBean) e selecione Incluir > Campo.
   O SmartGuide Criar Campo é aberto.
2. No campo **Nome do campo**, digite `iRequestProperties`.
3. Clique em **Procurar** para incluir o tipo de campo e digite `com.ibm.commerce.datatype.TypedProperty`. Clique em **OK**.
4. Para os modificadores de acesso, selecione **Protegido**.
5. Clique em **Concluir**.

Modificando os Métodos Set dos Beans de Dados: Depois de criar o bean de dados, é necessário modificar código em alguns dos métodos set gerados.

Para atualizar os métodos set, proceda da seguinte maneira:

1. Expanda o novo bean de dados para exibir seus campos e métodos.
2. Selecione o método **`setCommandContext(CommandContext)`** para exibir o código de origem.
   O painel Origem exibirá o código de origem da seguinte maneira:

```
public void setCommandContext(com.ibm.commerce.comand.CommandContext arg1)
 {
 }
```

3. Modifique o código de origem de forma que o método apareça da seguinte maneira:

```
public void setCommandContext(com.ibm.commerce.comand.CommandContext arg1)
{
   iCommandContext = arg1;
}
```

Salve o trabalho (Ctrl+S).

4. Selecione o método **setRequestProperties(TypedProperty)** para exibir seu código de origem.
   O painel Origem exibirá o código de origem da seguinte maneira:

```
public void setRequestProperties(
 com.ibm.commerce.datatype.TypedProperty arg1)
   throws Exception
   {
   }
```

5. Você pode querer modificar o código de origem para ocupar a chave primária do bean de acesso correspondente. A maneira recomendada para fazer isso é utilizar o gerenciador de beans de dados para definir esse valor de forma indireta. Esse método indireto é indicado para assegurar que um valor de chave principal obtido das propriedades da URL não substituirá a chave principal, se ele foi definido anteriormente. Para que o método setRequestProperties siga esse modelo, codifique-o de maneira semelhante ao fragmento de código a seguir. Observe que no código de exemplo a seguir, a chave principal é o ID do usuário. Isso pode ser diferente, dependendo da situação (por exemplo, o seguinte código pode não ser compilado imediatamente em seu aplicativo).

```
public void setRequestProperties(
   com.ibm.commerce.datatype.TypedProperty arg1)
   throws Exception
   {
      iRequestProperties = arg1;
     try {
          if (// check for nulls
              getDataBeanKeyUserId() == null)
             {
            super.setInitKey_UserId(aUserId);
             }
       } catch (com.ibm.commerce.exception.ParameterNotFoundException e)
           {
           }
   }
```

Há duas outras maneiras nas quais a chave principal do bean de acesso pode ser definida. Pode ser feita de maneira externa ao bean de acesso, por exemplo no modelo JSP. Nesse caso, antes de ativar o bean de dados no modelo JSP, chame o método set do bean de dados explicitamente para a chave principal. Por exemplo, o JSP pode incluir código semelhante ao seguinte (em que `db` é o objeto do bean de dados):

```
db.setInitKey_UserId(/*parâmetro de entrada*/)
db.activate();
```

Alternativamente, a chave principal pode ser definida de maneira direta. Isso é, o modelo JSP contém apenas o método db.activate e em seguida, o gerenciador de beans de dados define de maneira explícita a chave principal no bean de acesso. Por exemplo, o código do método setRequestProperties do bean de dados deve ser semelhante ao seguinte:

```
public void setRequestProperties(
   com.ibm.commerce.datatype.TypedProperty arg1)
   throws Exception
      {
      iRequestProperties = arg1;
         try
             {
            super.setInitKey_UserId(aUserId);
             }
       } catch (com.ibm.commerce.exception.ParameterNotFoundException e)
           {
           }
      }
```

Observe que o procedimento recomendado para definir a chave principal é o método indireto, mostrado na etapa 5.

Modificando os Métodos Get do Bean de Dados: Depois de criar o bean de dados, é necessário modificar código em alguns dos métodos get gerados.

Para atualizar os métodos get, proceda da seguinte maneira:

1. Expanda o novo bean de dados para exibir seus campos e métodos.
2. Selecione o método **getCommandContext()** para exibir o código de origem. O painel Origem exibirá o código de origem da seguinte maneira:

   ```
   public com.ibm.commerce.comand.CommandContext getCommandContext ()
   {
   return null;}
   ```
3. Modifique o código de origem de forma que o método apareça da seguinte maneira:

   ```
   public com.ibm.commerce.comand.CommandContext getCommandContext ()
   {
      return iCommandContext;
   }
   ```

   Salve o trabalho (Ctrl+S).
4. Selecione o método **getRequestProperties()** para exibir seu código de origem.
   O painel Origem exibirá o código de origem da seguinte maneira:

   ```
   public com.ibm.commerce.datatype.TypedProperty setRequestProperties()
   {
   return null;}
   ```

5. Modifique o código de origem de forma que ele apareça da seguinte maneira:

```
public com.ibm.commerce.datatype.TypedProperty setRequestProperties()
{
   return iRequestProperties;
}
```

   Salve o trabalho (Ctrl+S).

Modificar o Método Populate(): É necessário modificar o método populate, procedendo da seguinte maneira:

1. Expanda o novo bean de dados para exibir seus campos e métodos.
2. Selecione o método **`populate()`** para exibir o código de origem.
   O painel Origem exibirá o código de origem da seguinte maneira:

   ```
   public void populate () throws Exception {}
   ```

3. Modifique o código de origem de forma que o método apareça da seguinte maneira:

   ```
   public void populate() throws Exception {
      super.refreshCopyHelper();
   }
   ```

   Salve o trabalho (Ctrl+S).

Escrever novos beans de sessão

Ao criar novos beans de sessão, é necessário criar o bean de sessão em um grupo EJB separado dos grupos EJB do WebSphere Commerce. Armazene esse novo grupo EJB em um novo projeto que esteja separado dos projetos do WebSphere Commerce. Por exemplo, você deveria criar o grupo EJB `MyCustomBeans` dentro do pacote `com.mycompany.mycustomcode`. A separação de seu próprio código personalizado do código do WebSphere Commerce, minimizará o impacto de migração para releases futuros.

Seu novo bean de sessão deve estender a classe com.ibm.commerce.base.helpers.BaseJDBCHelper. A superclasse fornece métodos que permitem obter um objeto de conexão JDBC do objeto de origem de dados utilizado pelo servidor de comércio, dessa forma, o bean de sessão participa na mesma transação que outros beans de entidade. A seguir, um exemplo de código para demonstrar as funções fornecidas pela superclasse:

```
public class mySessionBean extends com.ibm.commerce.base.helpers.BaseJDBCHelper
  implements SessionBean {

    public Object myMethod () throws javax.naming.NamingException,
      java.rmi.RemoteException, SQLException {

       ////////////////////////////////////////////////
      //  -- your logic, such as initialization --    //
      ////////////////////////////////////////////////
```

```
    try {
            // get a connection from the WebSphere Commerce data source
            makeConnection();
            PreparedStatement stmt = getPreparedStatement(
               "your sql string");
            ////////////////////////////////////////////////////////
            // -- your logic such as set parameter into the prepared  //
            // statement --                                          //
            ////////////////////////////////////////////////////////
            ResultSet rs = executeQuery(stmt, false);

            ////////////////////////////////////////////////
            // -- your logic to process the result set -- //
           ////////////////////////////////////////////////

                }
           finally {
              // return the connection to the WebSphere Commerce data source
             closeConnection();
           }

       ////////////////////////////////////////////////////
      // -- your logic to return the result ---          //
      ////////////////////////////////////////////////////

  }

}
```

No exemplo de código anterior, o método `executeQuery` tem dois parâmetros de entrada. O primeiro é uma instrução preparada e o segundo é um sinalizador boolean relativo a uma operação de limpeza de cache. Defina esse sinalizador como `true` se necessitar que o contêiner limpe todos os objetos de entidade da transação atual da cache antes de executar a consulta. Isso seria obrigatório se você tivesse executado atualizações em alguns objetos de entidade e precisasse que a consulta pesquisasse nesses objetos atualizados. Se o sinalizador fosse definido como `false`, as atualizações desses objetos de entidade não seriam gravadas no banco de dados até o encerramento da transação.

Você deve limitar o uso dessa operação de limpeza e definir o sinalizador, como regra geral, para `false`, exceto nos casos em que seja realmente obrigatório. A operação de limpeza é uma operação que utiliza recursos intensivos.

Ciclos de Vida do Objeto

Os beans corporativos no modelo do objeto incluem objetos *independentes* e *dependentes*. Um objeto independente tem seu próprio ciclo de vida, controlado diretamente pelos pedidos de criação ou remoção da lógica de negócios invocando o objeto. Um objeto dependente possui um ciclo de vida que está

conectado a outro objeto, conhecido como *objeto do proprietário* (que pode também, sucessivamente, ser um objeto dependente, exceto se além da hierarquia de associação, existir um objeto independente). Quando o objeto do proprietário for excluído, todos os objetos dependentes também serão. As exclusões reais são controladas por especificações de exclusão em cascata dentro do banco de dados.

Por exemplo, apresentado um objeto de usuário que retorna um objeto de catálogo de endereços e uma lista de objetos de pedido, se o objeto do usuário for excluído, seu objeto de catálogo de endereços também será (já que o catálogo é de propriedade do usuário) e assim será para todos os objetos de endereço dentro do catálogo (já que os endereços são de propriedade do catálogo). No entanto, os objetos de pedido não são excluídos porque o proprietário dos pedidos é um objeto da loja e não um objeto do usuário.

Um padrão de design específico é utilizado para a criação de objetos dependentes. O método de criação de um objeto dependente deve fornecer uma referência para seu objeto de proprietário, portanto, o objeto de proprietário deve existir antes do objeto dependente ser criado.

Transações

A arquitetura Enterprise JavaBeans V1.0 especifica três opções de alternativas de tempo de consolidação com respeito ao estado da instância. Elas estão descritas como opções A, B e C no documento de especificação. Para concluir detalhes dessas opções, consulte o documento de especificação do Enterprise JavaBean's V1.0 da Sun Microystem.

Embora o WebSphere Application Server implemente as opções A e C, a opção A assume que o banco de dados não é compartilhado.

Na opção C, o contêiner do bean corporativo não armazena uma instância "pronto" entre as transações. Assim que a transação for concluída, a instância será retornada ao conjunto de instâncias disponíveis. O WebSphere Commerce utiliza as opções C pois o banco de dados é compartilhado por vários aplicativos do WebSphere Commerce. Nessa implementação, o contêiner carrega os dados persistentes para os beans de entidade no início de cada transação e os beans de entidade são colocados em cache apenas o quanto durar a transação. O contêiner ativa várias instâncias de um bean de entidade, uma para cada transação em que a entidade estiver sendo acessada. A sincronização da transação é executada pelo banco de dados.

O atributo da transação de cada bean corporativo está definido como `TX_REQUIRED`. Visto que o controlador da Web inicia uma transação antes de executar um comando que acesse um bean corporativo (através de seu bean de acesso correspondente), os métodos de negócios do bean corporativo são invocados dentro do contexto dessa transação.

Outras Considerações para Beans de Entidade

Localizar para Atualizar

Uma situação em que vários aplicativos podem acessar a mesma linha em um banco de dados para atualizar a linha é conhecida como uma *atualização simultânea*. Há situações em que as atualizações simultâneas podem ser permitidas e há outras situações em que elas definitivamente não são desejadas.

Se a atualização do banco de dados for uma sobreposição, na qual o novo valor não tem relação com o valor atual no banco de dados, as atualizações simultâneas podem ser permitidas. Se a atualização simultânea for permitida e vários aplicativos tentarem atualizar a mesma linha em um banco de dados, a última tentativa será aquela que é atualizada no banco de dados.

Se a atualização do banco de dados depender do valor atual no banco de dados, uma atualização simultânea não será desejada. Por exemplo, se um aplicativo estiver atualizando o inventário do produto, apenas um aplicativo de cada vez deverá ter permissão para atualizar o inventário.

Os fatores que afetam se uma atualização simultânea é permitida ou não incluem bloqueios do banco de dados e níveis de isolamento do bean corporativo.

Para evitar que um segundo aplicativo atualize a linha ao mesmo tempo, o primeiro aplicativo que acessa a linha deve buscar a linha utilizando a opção "localizar para atualizar (find for update)". Quando a opção "para atualizar (for update)" é utilizada, um *bloqueio de gravação* (também conhecido como um bloqueio exclusivo) é aplicado à linha. Com este bloqueio de gravação aplicado à linha, qualquer aplicativo que tentar acessar a linha utilizando a opção "localizar para atualizar (find for update)" será bloqueado.

Se o seu aplicativo permitir atualizações simultâneas, ele poderá apenas buscar os dados, sem bloquear a linha.

Considere o cenário OrderProcess em que UpdateInventory precisa encontrar todos os produtos incluídos em um pedido e atualizar o inventário. Visto que os mesmos produtos podem ser incluídos em muitos outros pedidos, *localizar para atualizar (find for update)* deveria ser utilizado o mais cedo possível dentro de um escopo de transação para reduzir a possibilidade de bloqueios. Portanto, o algoritmo UpdateInventory pode ser representado pelo seguinte pseudocódigo:

```
UpdateInventory
 find all the order items in the order
 for each order item
  fetch its inventory using "find for update"
  ...
```

No cenário Business-to-Business de longa execução, onde um pedido pode ter muitos itens, localizar para atualizar deveria ser utilizado o mais cedo possível. A lógica pode ser proveniente do seguinte:

```
find for update the inventory of all the products in an order
for each product
 if (total quantity ordered for that product < inventory)
        deduct quantity from inventory
 else
        error
```

Método remoto de alinhamento

Visto que o WebSphere Application Server não grava as alterações feitas em beans de entidade no banco de dados até o momento da consolidação da transação, o banco de dados pode temporariamente sair da sincronização com os dados em cache no contêiner do bean de entidade.

Um método remoto de alinhamento (na classe `com.ibm.commerce.base.helpers.BaseJDBCHelper`) que grava todas as alterações consolidadas em todas as transações (ou seja, leva informações da cache do bean corporativo) e atualiza o banco de dados. Esse método remoto pode ser chamado por um comando. Utilize esse método, somente quando absolutamente necessário, já que é dispendioso em termos de utilização de recursos e, portanto, tem um impacto negativo no desempenho.

Considere um comando de logon que possui a seguinte parte de código

```
UserAccessBean uab = ...;
 uab.setRegisteredTimestamp(currentTimestamp);
 uab.commitCopyHelper();
```

Antes da transação ter sido consolidada, REGISTEREDSTAMP na tabela USER não terá sido atualizado com a hora autenticada. A atualização ocorrerá somente no momento da consolidação da transação. O método de alinhamento deve ser utilizado para que toda consulta JDBC direta (na mesma transação), por exemplo, *selecionar a partir do usuário onde registeredstamp...* retorne o usuário com a hora autenticada do registro especificada.

Assegurando Beans Corporativos

Se você estiver utilizando o WebSphere Application Server para assegurar beans corporativos, será necessário atribuir os métodos de quaisquer beans corporativos novos ao grupo do método de segurança `WCSMethodGroup`, utilizando o Console do Administrador do WebSphere Application Server. Execute esta etapa ao implementar os novos beans corporativos. Além disso, se você modificar beans de entidade do WebSphere Commerce já existentes, deverá atribuir os métodos de todos os beans de entidade no grupo EJB afetado ao grupo de método de segurança `WCSMethodGroup`. Para obter uma descrição do processo de implementação para o código personalizado, consulte “Implementação de Código” na página 193.

Chaves Principais

Uma chave principal é uma chave exclusiva que faz parte da definição de uma tabela. Ela pode ser utilizada para distinguir um registro de outros. Todos os registros precisam ter uma chave principal. Ao criar um novo registro em uma tabela, pode ser necessário gerar uma chave principal exclusiva para o registro.

No modelo de programação do WebSphere Commerce, a camada persistente inclui beans de entidade que interagem com o banco de dados. Dessa forma, os registros do banco de dados podem ser criados quando é criada uma instância de um bean de entidade. Portanto, o método ejbCreate para a criação de instância de um bean de entidade pode precisar incluir lógica para gerar uma chave principal para novos registros.

Quando um aplicativo requer informações do banco de dados, ele utiliza indiretamente os beans de entidade, criando uma instância do bean de acesso correspondente do bean e em seguida, obtendo ou definindo vários campos. É criada uma instância de um bean de acesso para um registro específico de um banco de dados (por exemplo, para um perfil de usuário específico) e utiliza a chave principal para selecionar as informações corretas do banco de dados.

As seções a seguir descrevem como criar uma chave principal exclusiva e como selecionar através da chave principal.

Criando Chaves Principais: O método `ejbCreate` é utilizado para criar novas instâncias de um bean de entidade. Esse método é gerado automaticamente, mas o método gerado inclui somente lógica para inicializar chaves principais para um valor estático.

Você pode precisar assegurar que a chave principal é um valor novo e exclusivo. Nesse caso, você pode ter um método ejbCreate similar ao fragmento de código a seguir:

```
Public void ejbCreate(int argMyOtherValue)
   throws javax.ejb.CreateException,
   java.rmi.RemoteException {
      //Initialize CMP fields
      MyKeyValue = com.ibm.commerce.key.ECKeyManager.
         singleton().getNextKey("table_name");
      MyOtherValue = argMyOtherValue;
}
```

No fragmento de código acima, o método getNextKey gera um número inteiro exclusivo para a chave principal. O parâmetro de entrada *table_name* do método deve ser uma correspondência exata do valor TABLENAME definido na tabela KEYS. Certifique-se de corresponder os caracteres e maiúsculas e minúsculas exatamente.

Além de incluir o código anterior no método ejbCreate, você também deve criar uma entrada na tabela KEYS. O seguinte é um exemplo de uma instrução SQL para efetuar a entrada na tabela KEYS:

```
insert into KEYS (TABLENAME, COUNTER, KEYS_ID)
   values ("table_name", 0, 1)
```

Observe que com a instrução SQL anterior os valores padrão das outras colunas na tabela KEYS são aceitos. O valor de COUNTER indica o valor com o qual a contagem deve ser iniciada. O valor de KEYS_ID deve ser qualquer valor positivo.

Se sua chave principal for definida como um tipo de dados longo (BIGINT para DB2 ou NUMBER para Oracle), utilize o método `getNextKeyAsLong`.

Selecionando Pela Chave Principal: Em um bean de acesso, é necessário selecionar o registro de banco de dados apropriado, utilizando a chave principal. O fragmento de código a seguir demonstra como executar esta seleção. Ele também inclui lógica adicional, explicada posteriormente.

```
UserProfileAccessBean abUserProfile = new UserProfileAccessBean();
abUserProfile.setInitKey_UserId(getUserId().toString());
abUserProfile.refreshCopyHelper();
```

A primeira linha no fragmento de código acima cria uma instância de um novo UserProfileAccessBean que é chamado "abUserProfile". A segunda linha define a chave principal no bean de acesso. A convenção de nomenclatura de `setInitKey_`*xxx* (em que *xxx* é o nome do campo da chave principal) é utilizada pelo VisualAge for Java para denominar os métodos set de chaves principais. Ao instanciar um bean de acesso, você deve verificar se todos os campos definidos por um método setInitKey_xxx são inicializados antes de utilizar o método refreshCopyHelper. A ordem em que os métodos setInitKey_xxx são chamados não é importante.

Depois que todos os métodos setInitKey_xxx forem chamados, você terá inicializado todos os campos obrigatórios e poderá utilizar o método refreshCopyHelper para recuperar informações do banco de dados.

Se você atualizar valores na cache local do bean de acesso, será preciso incluir também uma chamada commitCopyHelper para atualizar o banco de dados com as informações atualizadas. Por exemplo, se após recuperar dados utilizando o método refreshCopyHelper você atualizar o nome de um cliente (definindo o valor do nome), deverá chamar, em seguida, `abUserProfile.commitCopyHelper()` para atualizar o banco de dados com as novas informações.

Utilizando Beans de Entidade

Um programa que utiliza beans corporativos deve lidar com a JNDI (Java Naming and Directory Interface) e bem como com as interfaces iniciais e remotas dos beans corporativos. Para simplificar o modelo de programação, é gerado um bean de acesso para cada bean corporativo. Quando criar seus próprios beans corporativos, utilize a ferramenta em VisualAge for Java para gerar esse bean de acesso.

Os comandos do WebSphere Commerce interagem com os beans de acesso em vez de diretamente com os beans de entidade. Conforme o diagrama ilustra, a utilização do bean de acesso fornece as seguintes vantagens:

- Uma interface de programação mais simples. O bean de acesso se comporta como um bean Java e oculta todas as interfaces de programação específicas do bean corporativo, como a JNDI e as interfaces iniciais e remotas, dos clientes.
- Em tempo de execução, o bean de acesso coloca em cache o objeto local do bean corporativo porque a procura do objeto local é expansiva, em termos de tempo e utilização de recursos.
- O bean de acesso implementa um objeto copyHelper que reduz o número de chamadas para o bean corporativo quando os comandos obtêm e definem os atributos do bean corporativo. Portanto, exige-se apenas uma única chamada para o bean corporativo, ao ler ou gravar vários atributos de bean corporativo.

O diagrama a seguir exibe a interação entre os comandos, os beans de acesso, os beans de entidade e o banco de dados.

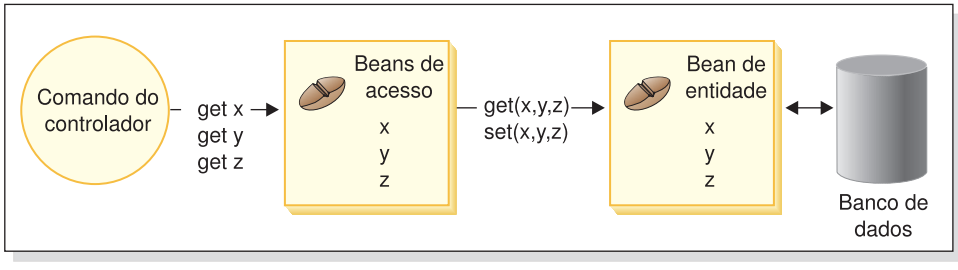


Figura 18.

Considerações sobre o Banco de Dados

Uma vez que você personaliza o aplicativo e-commerce, pode criar novas tabelas de banco de dados. Ao criar estas tabelas, recomenda-se que você siga um conjunto de convenções para que suas tabelas sejam criadas de uma maneira consistente com as tabelas do WebSphere Commerce.

Considerações sobre Nomenclatura de Objetos de Esquema do Banco de Dados

As seções subseqüentes fornecem orientação para a nomenclatura de objetos de esquema do banco de dados.

Convenções de nomenclatura para tabelas e exibições

A lista a seguir fornece orientação para a nomenclatura de novas tabelas e exibições:

- Para evitar conflito de nomes (nomes duplicados) com tabelas e exibições do WebSphere Commerce em releases futuros, o primeiro caractere no nome da tabela ou da exibição deve ser `X`. Por exemplo, `XMYTABLE`.
- O nome da tabela ou da exibição não deve ter mais de 10 caracteres de comprimento. Se o nome desejado exceder esse limite, reduza o comprimento, removendo vogais do final do nome, até somente 10 caracteres permanecerem.
- O nome da tabela ou da exibição não deve conter caracteres especiais, como "_", "+", "$", "%" ou espaços em branco.
- Não utilize palavras reservadas do banco de dados em nomes de tabelas ou de exibições.
- Nomes de exibição devem terminar com `VW`.
- Os nomes de tabelas e exibições devem ser substantivos simples.

Convenções de nomenclatura para colunas

A lista a seguir fornece orientação para a nomenclatura de colunas em tabelas novas:

- Para evitar conflito de nomes (nomes duplicados) com colunas em tabelas do WebSphere Commerce em releases futuros, o primeiro caractere no nome da coluna deve ser `X`. Por exemplo, `XMYCOLUMN`.
- O nome da coluna não deve ter mais de 18 caracteres de comprimento. Se o nome desejado exceder este limite, diminua o comprimento removendo vogais do final do nome até que haja apenas 18 caracteres.
- Nomes de colunas (diferentes de chaves externas) não devem conter caracteres especiais, como "_", "+", "$", "%" ou espaços em branco.
- Não utilize palavras reservadas do banco de dados em nomes de colunas.
- Palavras combinadas podem ser utilizadas como nomes de colunas, utilizando a combinação de voz ativa. Por exemplo, `COMBINERESULT`.
- As colunas da chave principal geradas devem ser nomeadas como *`table`*`_id`. Por exemplo, a chave principal para a tabela USERS é `USERS_ID`.
- Os nomes de coluna da chave externa gerados não devem ser alterados.
- Se reservar alguma coluna para futura personalização, ela deverá ser nomeada `field`*`x`* em que *`x`* é um dígito numérico começando do 1.

Convenções de nomenclatura para índices

A lista a seguir fornece orientação para a nomenclatura de índices em tabelas novas:

- O nome do índice não deve ter mais de 18 caracteres de comprimento.
- O nome do índice não deve conter espaços em branco.
- O nome do índice não deve conter palavras reservadas do banco de dados.
- Um índice não-exclusivo deve ser nomeado como `I_tablex` em que `table` é o nome da tabela e `x` é um número, começando em 1. Por exemplo, um índice não-exclusivo para a tabela USERS é `I_USERS1`.
- Um índice exclusivo deve ser nomeado como `UI_tablex` em que `table` é o nome da tabela e `x` é um número, começando em 1. Por exemplo, um índice exclusivo para a tabela USERS é `UI_USERS1`.
- O tamanho total do índice não deve ser maior que 254 bytes.
- O nome do índice deve ser exclusivo por todo o esquema do banco de dados.

Convenções de nomenclatura para chaves principais

A lista a seguir fornece orientação para a nomenclatura de chaves principais para tabelas novas:

- O nome da chave principal não deve ter mais de 18 caracteres de comprimento.
- O nome da chave principal não deve conter espaços em branco.
- O nome da chave principal não deve conter palavras reservadas do banco de dados.
- A chave principal deve ser nomeada como `P_table` em que `table` é o nome da tabela. Por exemplo, a chave principal para a tabela USERS é `P_USERS`.
- O nome da chave principal deve ser exclusivo por todo o esquema do banco de dados.

Convenções de nomenclatura para chaves externas

A lista a seguir fornece orientação para a nomenclatura de chaves externas para tabelas novas:

- O nome da chave externa não deve ter mais de 18 caracteres de comprimento.
- O nome da chave externa não deve conter espaços em branco.
- O nome da chave externa não deve conter palavras reservadas do banco de dados.
- A chave externa deve ser nomeada como `F_table` em que `table` é o nome da tabela. Por exemplo, a chave externa da tabela USERS é `F_USERS1`.
- O nome da chave externa precisa ser exclusiva por todo o esquema do banco de dados.

Convenções de Nomenclatura para Disparos do Banco de Dados

A lista a seguir fornece orientação para a nomenclatura de disparos do banco de dados:

- O nome do disparo do banco de dados não deve ter mais de 18 caracteres de comprimento.
- O nome do disparo do banco de dados não deve conter espaços em branco.
- O nome do disparo do banco de dados não deve conter palavras reservadas do banco de dados.
- O disparo do banco de dados deve ser nomeado como `T_`*`table`* em que *`table`* é o nome da tabela. Por exemplo, o nome do disparo do banco de dados para a tabela USERS é `T_USERS1`.
- O nome do disparo do banco de dados precisa ser exclusivo por todo o esquema do banco de dados.

Considerações sobre Tipo de Dados de Coluna do Banco de Dados

Esta seção apresenta tipos de dados de colunas que podem ser utilizados ao criar tabelas novas. As descrições dos vários tipos de dados utilizam terminologia do DB2. “Diferenças de Tipos de Dados entre Bancos de Dados” na página 89 descreve as diferenças se você estiver utilizando um banco de dados diferente.

BIGINT

Este é um número inteiro assinado de 64 que varia de -9223372036854775807 a 9223372036854775807. Compare este com INTEGER, que tem apenas a metade do tamanho de BIGINT.

INTEGER

Este é um número inteiro assinado de 32 bits que varia de -2147483647 a 2147483647. Em geral, INTEGER deve ser o tipo de dados numéricos finito padrão, em vez de BIGINT. A não ser que haja uma forte razão de negócios para utilizar BIGINT, por questões de desempenho, é melhor utilizar INTEGER como o tipo de dados numéricos. Um usuário comum do tipo de dados BIGINT é uma chave gerada pelo sistema.

A utilização dos tipos de dados SMALLINT ou SHORT é fortemente não recomendável porque esses tipos de dados são mapeados para um tipo de dados Java não-objeto e esses tipos de dados não-objeto causarão problemas em algumas instanciações do objeto bean corporativo.

TIMESTAMP

Este é um valor de sete partes (ano, mês, dia, hora, minuto, segundo e microssegundo) que designa uma data e hora, exceto se a hora incluir uma especificação fracional de microssegundos. A representação interna de uma marca de hora é uma cadeia de 10 bytes, cada um dos

quais consiste em 2 dígitos decimais empacotados. Os 4 primeiros bytes representam a data; os 3 bytes seguintes, a hora e os 3 últimos bytes, os microssegundos.

CHAR

Esta é uma cadeia de caracteres de comprimento fixo de comprimento INTEGER, que pode variar de 1 a 254 caracteres. Se a especificação do comprimento for omitida, um comprimento de 1 caractere será assumido. Como CHAR é uma coluna de banco de dados de comprimento fixo, todos os espaços de caracteres de rastreamento não utilizados são mudados para espaços brancos. A não ser por motivos de desempenho, não é recomendado utilizar o tipo de dados CHAR porque CHAR não é flexível e o comprimento não pode ser alterado posteriormente. Como regra, se a coluna da cadeia tiver menos de 64 caracteres de comprimento e for regularmente recuperada ou atualizada, utiliza CHAR para obter um melhor desempenho.

VARCHAR

Esta é uma cadeia de caracteres de comprimento variável de inteiro de comprimento máximo, que pode variar de 1 a 32.672. No entanto, diferentemente de CHAR, onde os dados da coluna são armazenados com a tabela, VARCHAR é representada internamente como um indicador de referência dentro de uma página do banco de dados. Portanto, o comprimento de uma coluna VARCHAR pode ser alterado a qualquer momento após a criação.

LONG VARCHAR

Esta é a cadeia de caracteres de comprimento variável que pode ser utilizada se VARCHAR não puder ser criada dentro da mesma página do banco de dados. LONG VARCHAR é muito semelhante a VARCHAR, exceto pelo fato de que pode abranger várias páginas do banco de dados. Restrinja a utilização do tipo de dados de LONG VARCHAR a apenas aqueles casos em que são absolutamente necessários porque os objetos de LONG VARCHAR são, geralmente, dispendiosos em termos de desempenho.

CLOB Esta é outra cadeia de caracteres de comprimento variável que pode ser utilizada se o comprimento da coluna precisar exceder o limite de 32 KB de LONG VARCHAR. O comprimento de um objeto CLOB pode atingir 1 GB sem modificar a configuração do banco de dados. Os dados de texto que estão armazenados como CLOB são convertidos apropriadamente ao mover-se entre diferentes sistemas.

BLOB Esta é uma cadeia de caracteres binários de comprimento variável que armazena dados não estruturados no banco de dados. Os objetos BLOB podem armazenar até 4 GB de dados binários. No geral, deve-se evitar utilizar BLOB como um tipo de dado de coluna, a não ser que ele seja absolutamente necessário. Em termos de desempenho,

um objeto BLOB é considerado como um dos objetos mais dispendiosos em qualquer banco de dados.

DECIMAL(20,5)
Este tipo de dados é especialmente definido para ser utilizado para a maioria dos números de ponto decimal fixos, como unidades de moeda. Para outros números decimais de ponto flutuante, FLOAT pode ser utilizado.

Diferenças de Tipos de Dados entre Bancos de Dados

A tabela a seguir lista os tipos de dados utilizados no esquema de banco de dados do WebSphere Commerce e mostra os tipos de dados correspondentes para implementações diferentes do banco de dados.

Objeto JDBC	Windows AIX Solaris Linux **DB2**	Windows AIX Solaris **Oracle®**	400 **DB2**
Hashtable	BLOB()	BLOB	BLOB()
Timestamp	TIMESTAMP	DATE	TIMESTAMP
Integer	INTEGER	INTEGER	INTEGER
BigDecimal	DECIMAL(,)	DECIMAL(,)	DECIMAL(,)
Long	BIGINT	NUMBER	BIGINT
Double	FLOAT	NUMBER	FLOAT
String	CHAR()	VARCHAR2()	GRAPHIC() CCSID 13488
byte[]	CHAR() para dados de bits	RAW()	CHAR() para dados de bits
String	VARCHAR()	VARCHAR2()	VARGRAPHIC() CCSID 13488
String	LONG VARCHAR	VARCHAR2() (Consulte nota exibida depois da tabela para obter mais detalhes).	VARGRAPHIC(4000) ALLOCATE() CCSID 13488
byte[]	LONG VARCHAR para dados de bits	LONG RAW	VARCHAR(8000) ALLOCATE() para dados de bits
String	CLOB()	CLOB()	DBCLOB() CCSID 13488

Nota:

Como resultado de taxas de êxito inconsistentes para quando o driver Oracle JDBC manipula informações do tipo de dados LONG, recomenda-se evitar o uso do tipo de dados LONG sempre que possível. O erro mais comum relatado nesta situação é “O fluxo já foi fechado”.

Se você tiver que utilizar esse tipo de dados, poderá ter apenas uma coluna por tabela de banco de dados que utilize o tipo LONG. Além disso, ao construir uma instrução select, não coloque a coluna LONG como o primeiro ou o último elemento na select. Outra solução alternativa para operações com sobrecarga é evitar o mapeamento dessa coluna específica para um campo CMP em um bean de entidade. Em vez disso, utilize um bean de sessão para executar recuperações e atualizações nessa coluna.

Capítulo 4. Controle de Acesso

Compreendendo o Controle de Acesso

O modelo de controle de acesso de um aplicativo WebSphere Commerce possui três conceitos principais: usuários, ações e recursos. Usuários são as pessoas que utilizam o sistema. Recursos são as entidades que são mantidas no aplicativo ou por ele. Por exemplo, recursos podem ser produtos, documentos ou pedidos. Os perfis de usuários que representam pessoas também são recursos. Ações são as atividades que os usuários podem executar nos recursos. O controle de acesso é o componente do aplicativo de e-commerce que determina se um determinado usuário pode executar uma determinada ação em um recurso específico.

Em um aplicativo WebSphere Commerce, há dois níveis principais de controle de acesso. O primeiro nível de controle de acesso é executado pelo WebSphere Application Server. Em relação a isso, o WebSphere Commerce utiliza o WebSphere Application Server para proteger beans corporativos e servlets. O segundo nível do controle de acesso é o sistema de controle de acesso fino do WebSphere Commerce.

A estrutura de controle de acesso do WebSphere Commerce utiliza políticas de controle de acesso para determinar se um determinado usuário tem permissão para executar uma determinada ação em um determinado recurso. Esta estrutura de controle de acesso fornece controle de acesso minucioso. Ela funciona em conjunto, mas não substitui o controle de acesso fornecido pelo WebSphere Application Server.

Visão Geral de Proteção de Recursos no WebSphere Application Server

Os recursos do WebSphere Commerce a seguir são protegidos pelo controle de acesso do WebSphere Application Server:

- Beans de entidade
  Esses beans modelam objetos em um aplicativo e-commerce. Eles são objetos distribuídos que podem ser acessados por clientes remotos.
- Modelos JSP
  O WebSphere Commerce utiliza modelos JSP para exibir páginas. Cada modelo JSP pode conter um ou mais beans de dados que recuperam dados dos beans de entidade. Os clientes podem solicitar páginas JSP através da composição de um pedido de URL.
- Comandos do controlador e de exibição
  Os clientes podem solicitar os comandos do controlador e de exibição através da composição de pedidos de URL. Além disso, uma página de

exibição pode conter um link para outra utilizando o nome do arquivo JSP ou o nome de exibição, conforme registrado na tabela VIEWREG.

O WebSphere Commerce Server geralmente é configurado para utilizar os seguintes caminhos na Web:

- `/webapp/wcs/stores/servlet/*`
  É utilizado para solicitações para o servlet de solicitação.
- `/webapp/wcs/stores/*.jsp`
  Utilizado para solicitações para o servlet JSP.

O diagrama a seguir mostra a rota que as solicitações poderiam seguir para acessar os recursos do WebSphere Commerce, para a configuração do caminho da Web acima.

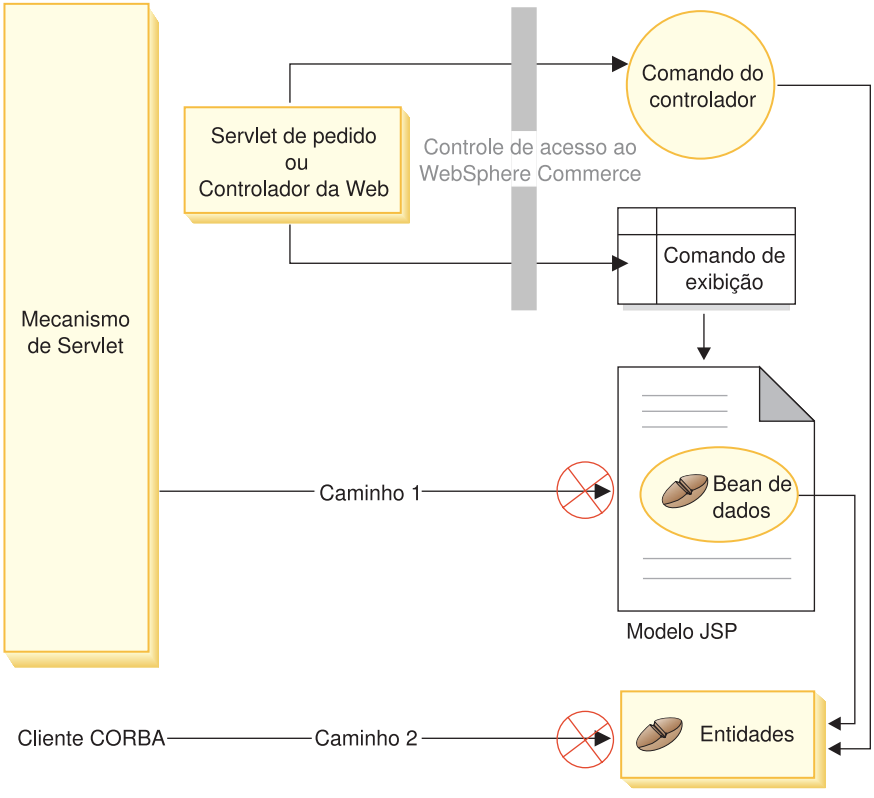


Figura 19.

Todos os pedidos legítimos devem ser direcionados para o servlet de pedido, que em seguida, os direciona para o controlador da Web. O controlador da Web implementa o controle de acesso dos comandos do controlador e das exibições. Os caminhos da Web mostrados acima fazem isso, entretanto,

possibilita que usuários maliciosos acessem diretamente os modelos JSP (caminho 1) e os beans de entidade (caminho 2). A fim de evitar que esse procedimento dos usuários seja bem-sucedido, ele deve ser rejeitado em tempo de execução.

O acesso direto aos modelos JSP e aos beans de entidade podem ser evitados utilizando uma das seguintes abordagens:

Segurança do WebSphere Application Server

O WebSphere Application Server fornece um recurso de segurança. Utilizando essa abordagem, todos os métodos de bean corporativo e modelos JSP serão configurados para que sejam invocados apenas pela Identidade do Sistema. Para acessar estes recursos do WebSphere Commerce, um pedido de URL deve ser roteado para o servlet de pedido que define a Identidade do Sistema para o thread atual, passando-o antes ao controlador da Web. O controlador da Web garante, então, que o originador da chamada possui a autorização requerida antes de passar o pedido ao comando do controlador ou exibição correspondente. Todas as tentativas para acessar diretamente os modelos JSP e os beans de entidade (ou seja, sem utilizar o controlador da Web) serão rejeitadas pelo componente de segurança do WebSphere Application Server.

Para obter informações sobre como configurar o WebSphere Application Server para assegurar recursos do WebSphere Commerce, consulte o *WebSphere Commerce Manual de Instalação*. Para obter informações sobre segurança dentro do WebSphere Application Server, consulte o tópico Administração do Sistema na documentação do WebSphere Application Server.

Para obter informações sobre como configurar a segurança do WebSphere Application Server para métodos em beans corporativos personalizados, consulte "Montando Novos Beans Corporativos em um Aplicativo Corporativo" na página 372 e "Montando Beans Corporativos Modificados em um Aplicativo Corporativo" na página 377.

Proteção de Firewall

Quando um WebSphere Commerce Server é executado atrás de um firewall, os clientes da Internet não conseguem acessar diretamente os beans de entidade. Utilizando essa abordagem, a proteção dos modelos JSP é fornecida pelo bean de dados que está incluído nesta página. O bean de dados é ativado pelo gerenciador de bean de dados. O gerenciador de bean de dados detecta se o modelo JSP foi encaminhado por um comando de exibição. Se não foi encaminhado por um comando de exibição, será lançada uma exceção e o pedido para o modelo JSP será rejeitado.

Introdução às Políticas de Controle de Acesso do WebSphere Commerce

O modelo de controle de acesso do WebSphere Commerce é baseado na aplicação de políticas de controle de acesso. As políticas de controle de acesso permitem que as regras de controle de acesso sejam exteriorizadas do código da lógica de negócios, removendo, assim, a necessidade de colocar as instruções de controle de acesso em código permanente. Por exemplo, não é necessário incluir código similar ao seguinte:

```
if (user.isAdministrator())
   then {}
```

As políticas de controle de acesso são aplicadas pelo gerenciador de políticas de controle de acesso. Em geral, quando um usuário tenta acessar um recurso protegido, o gerenciador de política de controle de acesso determina primeiro quais políticas de controle de acesso são aplicáveis para esse recurso protegido e em seguida, com base nas políticas de controle de acesso aplicáveis, determina se o usuário pode acessar os recursos solicitados.

Uma política de controle de acesso é uma política de 4 tuplas armazenada na tabela ACPOLICY. Cada política de controle de acesso tem o seguinte formato:

```
AccessControlPolicy [UserGroup, ActionGroup, ResourceGroup, Relationship]
```

Os elementos da política de controle de acesso de 4 tuplas especificam que um usuário pertencente a um grupo de usuários específico tem permissão para executar ações no grupo de ações especificado nos recursos pertencentes ao grupo de recursos especificado, desde que o usuário atenda as condições especificadas no relacionamento ou grupo de relacionamentos, em relação ao recurso em questão. Por exemplo, `[AllUsers, UpdateDoc, doc, creator]` especifica que todos os usuários poderão atualizar um documento, caso sejam os criadores do documento.

O grupo de usuários é um tipo específico de grupo de membros definido na tabela de banco de dados MBRGRP. Um grupo de usuários precisa estar associado ao tipo de grupo de membros `-2`. O valor `-2` representa um grupo de acesso e está definido na tabela MBRGRPTYPE. A associação entre o tipo do grupo de usuários e do grupo de membros está armazenada na tabela MBRGRPUSG.

A associação de um usuário a um grupo de usuários específico pode ser declarada explícita ou implicitamente. Uma especificação explícita ocorre se a tabela MBRGRPMBR afirmar que o usuário pertence a um grupo de membros específico. Uma especificação implícita ocorre se o usuário atender a uma condição (por exemplo, todos os usuários que tiverem a função de Gerente de Produto) declarada na tabela MBRGRPCOND. Pode haver também condições combinadas (por exemplo, todos os usuários que tiverem a função de Gerente de Produtos e que estejam no cargo há pelo menos 6 meses) ou exclusões explícitas.

A maioria das condições de inclusão ou exclusão de um usuário em um grupo de usuários está baseada no atendimento de uma função específica pelo usuário. Por exemplo, pode existir uma política de controle de acesso que permita que todos os usuários que preencham a função de Gerente de Produtos, executem operações de gerenciamento de catálogo. Nesse caso, qualquer usuário que tenha a função de Gerente de Produtos atribuída na tabela MBRROLE estará incluído implicitamente no grupo de usuários.

Para obter mais detalhes sobre o subsistema de grupos de membros, consulte a ajuda online do WebSphere Commerce.

O elemento ActionGroup vem da tabela ACACTGRP. Um grupo de ações refere-se a um grupo de ações especificado explicitamente. A listagem de ações está armazenada na tabela ACACTION e o relacionamento de cada ação com seu grupo (ou grupos) de ações está armazenado na tabela ACACTACTGP. Um exemplo de um grupo de ações é o ″OrderWriteCommand″. Esse grupo de ações inclui as seguintes ações que são utilizadas para atualizar pedidos:

- com.ibm.commerce.order.commands.OrderDeleteCmd
- com.ibm.commerce.order.commands.OrderCancelCmd
- com.ibm.commerce.order.commands.OrderProfileUpateCmd
- com.ibm.commerce.order.commands.OrderUnlockCmd
- com.ibm.commerce.order.commands.OrderScheduleCmd
- com.ibm.commerce.order.commands.ScheduledOrderCancelCmd
- com.ibm.commerce.order.commands.ScheduledOrderProcessCmd
- com.ibm.commerce.order.commands.OrderItemAddCmd
- com.ibm.commerce.order.commands.OrderItemDeleteCmd
- com.ibm.commerce.order.commands.OrderItemUpdateCmd
- com.ibm.commerce.order.commands.PayResetPMCmd

Um grupo de recursos é um mecanismo para agrupar tipos específicos de recursos. A associação de um recurso em um grupo de recursos pode ser especificada em uma de duas maneiras:

- Utilizando a coluna de condições da tabela ACRESGRP
- Utilizando a tabela ACRESGPRES

Na maioria dos casos, utilizar a tabela ACRESGPRES para associar recursos a grupos de recursos é suficiente. Com este método, os recursos são definidos na tabela ACRESCGRY utilizando seu nome de classe Java. Em seguida, esses recursos são associados aos grupos de recursos apropriados (tabela ACRESGRP) utilizando a tabela de associação ACRESGPRES. Em casos onde o nome de classe Java sozinho não é suficiente para definir os membros de um grupo de recursos (por exemplo, se for necessário restringir os objetos

desta classe posteriormente com base em um atributo do recurso), o grupo de recursos poderá ser definido totalmente utilizando a coluna de condições da tabela ACRESGRP. Observe que, para executar este agrupamento de recursos com base em um atributo, o recurso deverá também implementar a interface agrupável.

O diagrama a seguir mostra um exemplo de uma especificação de agrupamento de recursos. Nesse grupo de recursos de exemplo, `10023` inclui todos os recursos associados a ele na tabela ACRESGPRES. O grupo de recursos `10070` é definido utilizando-se a coluna do campo de condições da tabela ACRESGRP. Esse grupo de recursos inclui instâncias da interface remota do Pedido que também têm status = `"Z"` (especificando uma lista de requisições compartilhada).

Nota: Os detalhes sobre as informações de XML da coluna Condições da tabela ACRESGRP são encontrados no *WebSphere Commerce Access - Manual de Controle*.

ACRESGRP

AcResGrp_Id	GrpName	Condições
10023	AccountRepresentatives CmdResourceGroup	nulo
10070	SharedRequisitionList ResourceGroup	<profile> <andListCondition> <simpleCondition> <variable name="Status"/> <operator name="="/> <value data="Z"/> </simpleCondition> <simpleCondition> <variable name="classname"/> <operator name="="/> <value data="com.ibm.commerce.order.objects.Order"/> </simpleCondition> </andListCondition> </profile>

ACRESGRPES

AcResGrp_Id	AcResCgry_Id
10023	10246
10023	10247
10023	10248
10023	10249
10023	10250

ACRESCGRY

AcResCgry_Id	ResClassname
10246	com.ibm.commerce.contract.commands.ContractCreateCmd
10247	com.ibm.commerce.contract.commands.ContractCreateCmd
10248	com.ibm.commerce.contract.commands.ContractCreateCmd
10249	com.ibm.commerce.contract.commands.ContractCreateCmd
10250	com.ibm.commerce.contract.commands.ContractCreateCmd

Figura 20.

A coluna MEMBER_ID das tabelas ACACTGRP, ACRESGRP, e ACRELGRP devem ter um valor de `-2001` (Organização da Raiz).

A política de controle de acesso pode, opcionalmente, incluir um elemento Relationship ou RelationshipGroup como seu quarto elemento.

Se sua política de controle de acesso utilizar um elemento Relationship, ele virá da tabela ACRELATION. Se, por outro lado, ela incluir um elemento RelationshipGroup, ele virá da tabela ACRELGRP. Observe que não é necessário incluir nenhum dos dois, mas se um deles for incluído, o outro não poderá ser. Uma especificação RelationshipGroup da tabela ACRELGRP tem precedência sobre as informações de Relationship da tabela ACRELATION.

A tabela ACRELATION especifica os tipos de relacionamentos existentes entre usuários e recursos. Alguns exemplos de tipos de relacionamentos incluem criador, submissor e proprietário. Um exemplo do uso do elemento relationship é utilizá-lo para assegurar que o criador de um pedido sempre possa atualizar o pedido.

A tabela ACRELGRP especifica os tipos de relacionamentos que podem ser associados a recursos específicos. Um grupo de relacionamentos é um agrupamento de uma ou mais cadeias de relacionamentos. Uma cadeia de relacionamentos é uma série de um ou mais relacionamentos. Um exemplo de grupo de relacionamentos é especificar que um usuário deve ser o criador do recurso e também pertencer à entidade organizacional compradora referida no recurso.

A especificação do grupo de relacionamentos (ou relacionamento) é uma parte opcional da política de controle de acesso. Será utilizada geralmente se você tiver criado seus próprios comandos e esses comandos não estiverem restritos a determinadas funções. Nesses casos, você pode optar por aplicar um relacionamento entre o usuário e o recurso. Normalmente, se os comandos tiverem que ficar restritos a determinadas funções, isso será realizado através do elemento UserGroup da política de controle de acesso, em vez de utilizar o elemento Relationship.

Outro conceito importante relacionado a políticas de controle de acesso é o de um *proprietário* de política de controle de acesso. Um proprietário de política de controle de acesso é a entidade organizacional que possui a política de controle de acesso. Conhecer o proprietário de uma política de controle de acesso é importante porque uma política de controle de acesso pode ser aplicada apenas a recursos que são de propriedade do proprietário da política de controle de acesso.

Para cada recurso em questão, o gerenciador de política de controle de acesso aplica políticas de controle de acesso que sejam de propriedade da entidade organizacional ou de suas entidades organizacionais superiores na hierarquia de membros, até que uma política que conceda permissão seja encontrada ou até que todas as políticas tenham sido verificadas e nenhuma permissão concedida.

Considere o seguinte diagrama, que mostra uma hierarquia de membros.

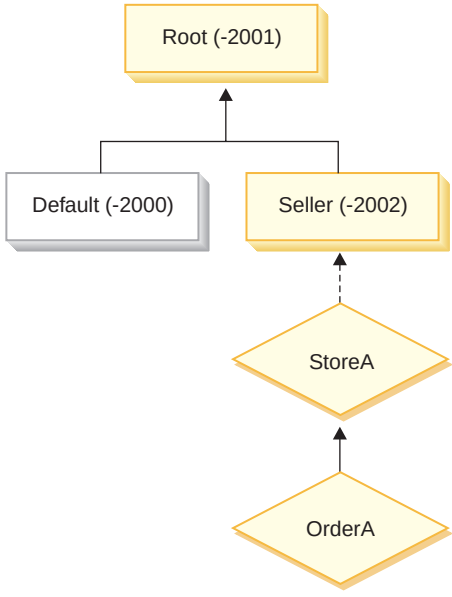


Figura 21.

Para o recurso "OrderA", qualquer política de controle de acesso que seja de propriedade da organização Seller ou Root pode ser aplicada. Se o gerenciador de política de controle de acesso encontrar uma política de propriedade dessas organizações que conceda permissão ao usuário (baseado nos quatro elementos da política de controle de acesso) ele parará imediatamente a pesquisa nas políticas de controle de acesso. No entanto, se não encontrar nenhuma política de controle de acesso pertencente a essas organizações que conceda permissão ao usuário para executar a ação nos recursos protegidos, então o acesso será negado.

Grupos de Relacionamentos

Um grupo de relacionamentos permite especificar vários relacionamentos. Um relacionamento pode ser diretamente entre um usuário e o recurso em questão, ou pode ser uma cadeia de relacionamentos relacionados indiretamente ao usuário no recurso.

Nota: Nas seções relacionadas a grupos de relacionamentos a seguir, é importante reconhecer que as únicas organizações disponíveis no WebSphere Commerce Professional Edition são a RootOrganization, a DefaultOrganization e a SellerOrganization. Exemplos referentes a outras organizações aplicam-se apenas ao WebSphere Commerce Business Edition.

Comparando Relacionamentos a Grupos de Relacionamentos: As políticas de controle de acesso podem especificar que um usuário deve preencher um determinado relacionamento relativo ao recurso sendo acessado, ou podem especificar que um usuário deve preencher as condições especificadas em um grupo de relacionamentos.

Na maioria dos casos, especificar um relacionamento deve atender os requisitos de controle de acesso de seu aplicativo. No entanto, se a política for uma que você deva especificar um relacionamento que não seja diretamente entre o usuário e o recurso, mas for realmente uma série de relacionamentos entre o usuário e o recurso, você deverá utilizar um grupo de relacionamentos.

Por exemplo, se você tiver que especificar uma associação entre um usuário e uma organização compradora em que o relacionamento requeira que o usuário esteja desempenhando uma determinada função para essa organização ou que o usuário seja um membro da organização compradora, deverá utilizar então um grupo de relacionamentos e uma cadeia de relacionamentos.

Se simplesmente precisar reforçar uma associação que seja diretamente entre o usuário e o recurso em questão, poderá utilizar um relacionamento simples. Por exemplo, esse seria o caso se você precisasse reforçar que o usuário deve ser o criador do recurso.

Se você combinar vários relacionamentos simples, por exemplo, o usuário deverá ser o criador *ou* o submissor, então isso se tornará uma cadeia de relacionamentos e você deverá utilizar um grupo de relacionamentos. Essa combinação de relacionamentos simples pode ocorrer ao utilizar o WebSphere Commerce Professional Edition ou o WebSphere Commerce Business Edition.

Informações Gerais Sobre Grupos de Relacionamentos: Uma cadeia de relacionamentos é uma série de um ou mais relacionamentos. O comprimento de uma cadeia de relacionamentos é determinada pelo número de relacionamentos contidos. Isso pode ser determinado examinando-se o número de elementos `<parameter name="aName" value="aValue" />` na representação XML da cadeia de representações.

Somente o último elemento `<parameter name="Relationship" value="aValue" />` deve ser manipulado pelo método fulfills() do recurso.Os outros são manipulados internamente pelo gerenciador de política de controle de acesso.

Quando uma cadeia de relacionamentos tem um comprimento de 2, o primeiro elemento `<parameter name="aName" value="aValue" />` está entre

um usuário e uma entidade organizacional. O último elemento `<parameter name="aName" value="aValue" />` está entre uma entidade organizacional e o recurso.

Se for necessário definir grupos de relacionamentos, isso deverá ser feito definindo-se as informações do grupo de relacionamentos em um arquivo XML. Você pode modificar o arquivo `defaultAccessControlPolicies.xml` ou criar seu próprio arquivo XML. Para obter mais informações sobre a criação dessas informações baseadas em XML, consulte o *WebSphere Commerce Access - Manual de Controle*.

As seções a seguir mostram exemplos de tipos diferentes de grupos de relacionamentos.

Grupos de Relacionamentos Compostos de uma Única Cadeia de Relacionamentos:

Business Como parte de uma política de controle de acesso, poderá ser necessário que você reforce que um usuário deva pertencer à entidade organizacional que é a BuyingOrganizationalEntity do recurso. Isso requer a criação de um grupo de relacionamentos composto por uma cadeia de relacionamentos que possua um comprimento "dois". Diz-se que a cadeia de relacionamentos possui o comprimento "dois" porque consiste em dois relacionamentos separados. O primeiro relacionamento é entre o usuário e sua entidade organizacional pai. O usuário é o "filho" nesse relacionamento. Para o segundo relacionamento, o gerenciador de política de controle de acesso verifica se a entidade organizacional pai preenche o relacionamento da BuyingOrganizationalEntity com o recurso. Em outras palavras, será retornado "true" se for a entidade organizacional compradora do recurso.

O seguinte fragmento de XML foi retirado do arquivo `defaultAccessControlPolicies.xml` e mostra como definir esse tipo de grupo de relacionamentos:

```
<RelationGroup Name="MemberOf->BuyerOrganizationalEntity"
     OwnerID="RootOrganization">
   <RelationCondition><![CDATA[
      <profile>
         <openCondition name="RELATIONSHIP_CHAIN">
            <parameter name="HIERARCHY" value="child"/>
            <parameter name="RELATIONSHIP" value="BuyingOrganizationalEntity"/>
         </openCondition>
      </profile>
   ]]></RelationCondition>
</RelationGroup>
```

Business Outro exemplo seria reforçar que o usuário deve ter a função de Representante de Conta para a entidade organizacional que é a entidade organizacional de compra do recurso em questão. Novamente, isso utiliza um grupo de relacionamentos composto de uma cadeia de relacionamentos de

comprimento ″dois″. A primeira parte da cadeia localizará todas as entidades organizacionais para as quais o usuário tenha a função de Representante de Conta. Depois, para este conjunto de entidades organizacionais, o gerenciador de política de controle de acesso verifica se pelo menos uma delas preenche o relacionamento da BuyingOrganizationalEntity com o recurso. Em outras palavras, será retornado ″true″ se uma delas for a entidade organizacional compradora do recurso.

O seguinte fragmento de XML foi retirado do arquivo `defaultAccessControlPolicies.xml` e mostra como definir esse tipo de grupo de relacionamentos:

```
<RelationGroup Name="AccountRep->BuyerOrganizationalEntity"
     OwnerID="RootOrganization">
   <RelationCondition><![CDATA[
      <profile>
         <openCondition name="RELATIONSHIP_CHAIN">
            <parameter name="ROLE" value="Account Representative"/>
            <parameter name="RELATIONSHIP" value="BuyingOrganizationalEntity"/>
         </openCondition>
      </profile>
   ]]></RelationCondition>
</RelationGroup>
```

Grupos de Relacionamentos Compostos de Várias Cadeias de Relacionamentos: É possível compor um grupo de relacionamentos para que contenha várias cadeias de relacionamentos. Ao fazer isso, é necessário especificar se o usuário deve satisfazer todas as cadeias de relacionamentos, significando que é um cenário *AND* ou se o usuário deve satisfazer pelo menos uma das cadeias de relacionamentos, significando que é um cenário *OR*.

Business Para demonstrar esse tipo de relacionamento, o seguinte fragmento de XML é utilizado para reforçar que um usuário deve ser o criador do recurso e que o usuário também deve pertencer à BuyingOrganizationalEntity especificada no recurso. A primeira cadeia, que especifica que um usuário deve ser o criador do recurso, tem comprimento ″um″. A segunda cadeia que especifica que o usuário deve pertencer à BuyingOrganizationalEntity especificada no recurso tem comprimento ″dois″.

```
<RelationGroup Name="Creator_And_MemberOf->BuyerOrganizationalEntity"
     OwnerID="RootOrganization">
 <RelationCondition><![CDATA[
  <profile>
   <andListCondition>
    <openCondition name="RELATIONSHIP_CHAIN">
     <parameter name="RELATIONSHIP" value="creator" />
    </openCondition>
    <openCondition name="RELATIONSHIP_CHAIN">
     <parameter name="HIERARCHY" value="child"/>
     <parameter name="RELATIONSHIP" value="BuyingOrganizationalEntity"/>
    </openCondition>
```

```
   </andListCondition>
  </profile>
 ]]></RelationCondition>
</RelationGroup>
```

Se, em vez do cenário *AND*, você requerer que o usuário satisfaça uma das duas cadeias de relacionamentos, a tag `<andListCondition>` deverá ser alterada para `<orListCondition>`.

Professional Business Para demonstrar um grupo de relacionamentos que possa ser utilizado no WebSphere Commerce Professional Edition (e também no WebSphere Commerce Business Edition), considere um grupo de relacionamentos utilizado para reforçar que o usuário deve ser o criador ou o submissor do recurso. Isso é mostrado no fragmento de XML a seguir.

```
<RelationGroup Name="Creator_Or_Submitter"
     OwnerID="RootOrganization">
   <RelationCondition><![CDATA [
   <profile>
      <orListCondition>
         <openCondition name="RELATIONSHIP_CHAIN">
            <parameter name="RELATIONSHIP"value="creator"/>
         </openCondition>
         <openCondition name="RELATIONSHIP_CHAIN">
            <parameter name="RELATIONSHIP"value="submitter"/>
         </openCondition>
      </orListCondition>
   </profile>
   ]]></RelationCondition>
</RelationGroup>
```

Tipos de Controle de Acesso

Há dois tipos de controle de acesso, os dois são baseados em políticas: controle de acesso no nível de comando e controle de acesso no nível de recurso.

O controle de acesso no nível de comando (também conhecido como "baseado na função") utiliza um tipo amplo de política. É possível especificar que todos os usuários de uma função específica podem executar determinados tipos de comandos. Por exemplo, você pode especificar que os usuários com função de Representante de Conta podem executar qualquer comando no grupo de recursos AccountRepresentativesCmdResourceGroup. Ou, como descrito no diagrama a seguir, outra política de exemplo especifica que todos os administradores de loja podem executar qualquer ação especificada no Grupo ExecuteCommandAction em qualquer recurso que é especificado pelo StoreAdminCmdResourceGrp.

Nota: As informações de XML da coluna Condições da tabela MBRGRPCOND são geradas quando você utiliza o Administration Console para configurar seus grupos de acesso. Para obter informações

sobre a utilização do Administration Console para configurar grupos de acesso, consulte a ajuda online do WebSphere Commerce.

ACPOLICY

PolicyName	Member_Id	MbrGrp_Id	AcActGrp_id	AcResGrp_Id	AcRelGrp_Id
StoreAdministrators ExecuteStoreAdmin CmdResourceGroup	-2001	-8	10052	10018	nulo

MBRGRP

MbrGrp_Id	MbrGrpName
-8	StoreAdministrators

MBRGRPCOND

MbrGrp_Id	Condições
-8	<profile> <simpleCondition> <variable name="role"/> <operator name="="/> <value data="Store Administrator"/> </simpleCondition> </profile>

ACACTGRP

AcActGrp_Id	GroupName
10052	ExecuteCommandActionGroup

ACRESGRP

AcResGrp_Id	GrpName
10018	StoreAdminCmdResourceGroup

Figura 22.

Uma política de controle de acesso no nível de comando sempre tem o `ExecuteCommandActionGroup` como o grupo de ações para comandos do controlador. Para as exibições, o grupo de recursos sempre é `ViewCommandResourceGroup`.

Todos os comandos do controlador devem ser protegidos pelo comando-controle de nível de acesso. Além disso, qualquer exibição que possa ser chamada diretamente ou que possa ser lançada por um redirecionador de

outro comando (em contraste a ser lançado pelo encaminhamento da visualização) pode ser protegido pelo comando-controle de nível de acesso.

O comando-controle de nível de acesso não considera o recurso sobre o qual o comando agiria. Ele meramente determina se o usuário tem permissão para executar o comando específico. Se o usuário tiver permissão para executar o comando, uma política de controle de acesso no nível do recurso subseqüente poderá ser aplicada para determinar se o usuário pode acessar o recurso em questão.

Considere quando um administrador de loja tenta executar uma tarefa administrativa. O primeiro nível de verificação de controle de acesso seria determinar se o usuário tem permissão para executar o comando de administração da loja específico. Uma vez determinado que o usuário realmente tem permissão para fazer isso (porque os administradores de loja têm permissão para executar comandos no grupo storeAdminCmds), uma política de controle de acesso no nível do recurso poderá ser chamada. Essa política pode afirmar que os administradores de loja tenham permissão apenas para executar tarefas administrativas em lojas pertencentes à organização para a qual o usuário seja um administrador de loja.

Para resumir, no controle de acesso no nível de comando, o comando é o próprio "recurso" e a "ação" deve simplesmente executar o comando (em outras palavras, instanciar o objeto de comando). O controle de acesso verifica se o usuário tem permissão para executar o comando. Por comparação, no controle de acesso no nível de recurso, o "recurso" é qualquer recurso protegido que o comando ou o bean acessam e a "ação" é o próprio comando.

Interações do Controle de Acesso

Esta seção apresenta o diagrama de interação que mostra como o controle de acesso funciona na estrutura de políticas de controle de acesso do WebSphere Commerce.

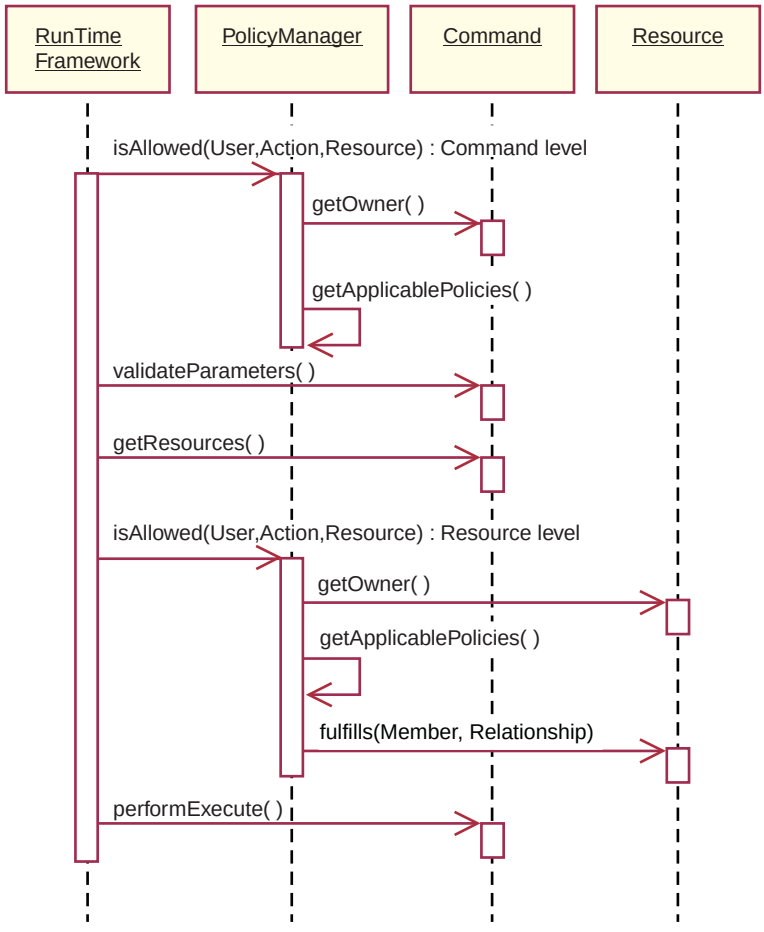


Figura 23.

O diagrama anterior mostra ações que são executadas pelo *gerenciador de política* de controle de acesso. O gerenciador de política de controle de acesso é o componente de controle de acesso que determina se o usuário atual tem permissão ou não para executar a ação especificada no recurso especificado. Ele determina isso pesquisando as políticas pertencentes ao proprietário do recurso e suas organizações ascendentes. Se pelo menos uma política conceder acesso, então a permissão será concedida.

A lista a seguir descreve as ações do diagrama de interação acima. Elas estão organizadas de cima para baixo no diagrama.

1. `isAllowed()`
   Os componentes de runtime determinam se o usuário tem acesso no nível de comando para o comando de controlador ou de exibição.

2. `getOwner()`
   O gerenciador de política de controle de acesso determina o proprietário do recurso no nível de comando. A implementação padrão retorna o identificador de membro (`memberId`) do proprietário da loja (`storeId`) que está no contexto do comando. Se não houver identificador da loja no contexto do comando, então a organização raiz (`-2001`) será retornada.
3. `getApplicablePolicies()`
   O gerenciador de política de controle de acesso localiza e processa as políticas aplicáveis, com base no usuário, ação e recurso especificados.
4. `validateParameters()`
   Verificação e resolução inicial de parâmetros.
5. `getResources()`
   Retorna um vetor de acesso que é um vetor de pares recurso-ação.

   Se nada for retornado, a verificação de controle de acesso no nível de recurso não será executada. Se houver recursos que devem ser protegidos, um vetor de acesso (que consista em pares recurso-ação) deverá ser retornado.

   Cada *recurso* é uma instância de um objeto que pode ser protegido (um objeto que implementa a interface `com.ibm.commerce.security.Protectable`. Em muitos casos, o recurso é um bean de acesso.

   Um bean de acesso pode não implementar a interface `com.ibm.commerce.security.Protectable`, no entanto, a verificação do controle de acesso ainda pode ocorrer desde que o bean corporativo correspondente esteja protegido, de acordo com as informações incluídas em "Implementando o Controle de Acesso em Beans Corporativos" na página 110.

   A *ação* é uma cadeia que representa a operação a ser executada no recurso. Na maioria dos casos, a ação é o nome da interface do comando.
6. `isAllowed()`
   Os componentes do runtime determinam se o usuário tem acesso no nível do recurso para todos os pares recurso-ação especificados por `getResources()`.
7. `getOwner()`
   O recurso retorna `memberId` de seu proprietário. Isso determina quais políticas se aplicam. Somente as políticas que pertencem ao proprietário do recurso e suas organizações ascendentes se aplicam.
8. `getApplicablePolicies()`
   O gerenciador de política de controle de acesso pesquisa as políticas aplicáveis e em seguida, as aplica. Se pelo menos uma política por par recurso-ação que concede permissão ao usuário para acessar o recurso for encontrada, então o acesso será concedido; caso contrário, o acesso será negado.

9. `fulfills()`
   Se uma política aplicável tiver um grupo de relacionamentos especificado, será feita uma verificação no recurso para ver se o membro satisfaz o relacionamento ou relacionamentos especificados, em relação ao recurso.
10. `performExecute()`
    A lógica de negócios do comando.

Interface Protegida

Um fator chave para que um recurso seja protegido pelas políticas de controle de acesso do WebSphere Commerce é que o recurso deve implementar a interface `com.ibm.commerce.security.Protectable`. Essa interface é mais comumente utilizada com beans corporativos e beans de dados, mas somente esses beans específicos que requerem proteção precisam implementar a interface.

Com a interface Protectable, um recurso deve fornecer dois métodos chave: `getOwner()` e `fulfills(Long member, String relationship)`.

As políticas de controle de acesso pertencem a organizações ou entidades organizacionais. O método `getOwner` retorna o memberId do proprietário do recurso protegível. Após o gerenciador de políticas de controle de acesso determinar o proprietário do recurso, ele também obtém o memberId de cada um dos ascendentes do proprietário na hierarquia de membros. Todas as políticas de controle de acesso que pertencem ao proprietário do pedido getOwner original, assim como todas as políticas de controle de acesso que pertencem a qualquer ascendente do proprietário serão então aplicadas.

As políticas de controle de acesso que se aplicam ao proprietário especificado, assim como as políticas de controle de acesso que se aplicam a qualquer ascendente de nível superior do proprietário na hierarquia de membros são aplicadas.

O método `fulfills` somente retornará "true" se o membro fornecido satisfizer o relacionamento requerido em relação ao recurso. Normalmente, o membro é um único usuário, entretanto, também pode ser uma organização. Seria uma organização se você estivesse utilizando um grupo de relacionamentos na política de controle de acesso.

Interface Agrupável

O aplicativo de uma política de controle de acesso é específico de um grupo de recursos. Agrupamentos de recursos podem ser feitos com base em atributos como o nome da classe, o estado de um pedido ou o valor do storeId.

Se um recurso for agrupado por um atributo diferente de seu nome de classe com a finalidade de aplicar políticas de controle de acesso, ele deverá implementar a interface `com.ibm.commerce.grouping.Groupable`.

O fragmento de código a seguir representa a interface Groupable:

```
Groupable interface {
 Object getGroupingAttributeValue (String attributeName, GroupContext context)
}
```

Por exemplo, para implementar uma política que se aplique somente a pedidos que estejam no estado pendente (`status = P (pendente)`), a interface remota do bean de entidade Order implementará a interface Groupable e o valor de attributeName será definido como `"status"`.

A utilização da interface Groupable é rara.

Procurando Mais Informações sobre Controle de Acesso

Para obter mais informações sobre o modelo de controle de acesso do WebSphere Commerce, consulte o manual *WebSphere Commerce Access - Manual de Controle*. Esse manual fornece uma visão geral detalhada do controle de acesso e descreve como utilizar o Administration Console para criar ou modificar políticas, grupos de ações e grupos de recursos.

Implementando o Controle de Acesso

Esta seção descreve como implementar o controle de acesso em código personalizado.

Identificando Recursos que Podem ser Protegidos

Em geral, beans corporativos e beans de dados são recursos que você pode querer proteger. No entanto, nem todos os beans corporativos e beans de dados devem ser protegidos. No aplicativo WebSphere Commerce existente, os recursos que requerem proteção já implementam a interface que pode ser protegida. A dúvida do que proteger aparece quando você cria novos beans corporativos e beans de dados. Decidir quais recursos proteger vai depender de seu aplicativo.

Se um comando retorna um bean corporativo no método `getResources`, então o bean corporativo precisa ser protegido, porque o gerenciador de políticas de controle de acesso chamará o método getOwner do bean corporativo. O método `fulfills` também será chamado se um relacionamento for especificado na política de controle de acesso no nível de recurso correspondente.

Se você implementasse a interface protegível (e, portanto, colocasse o recurso sob proteção) para todos os seus próprios beans corporativos e beans de

dados, seu aplicativo poderia requerer muitas políticas. À medida que o número de políticas aumenta, o desempenho pode cair e o gerenciamento de política torna-se mais desafiador.

Uma distinção teórica é feita entre os recursos principais e o recurso dependente. Um *recurso principal* pode existir por si só. Um *recurso dependente* existe somente quando existe o recurso principal relacionado a ele. Por exemplo, no código de aplicativo fornecido com o WebSphere Commerce, o bean corporativo Order é um recurso que pode ser protegido, mas o bean de entidade OrderItem não é. A razão para isso é que a existência de um OrderItem depende de um Order -- o Order é o recurso principal e o OrderItem é um recurso dependente. Se um usuário tiver acesso a um Order, ele também deve ter acesso aos itens do pedido.

De forma similar, o bean de entidade User é um recurso protegível, mas o bean de entidade Address não é. Nesse caso, a existência do endereço depende do usuário, portanto, qualquer coisa que tenha acesso ao usuário, também deve ter acesso ao endereço.

Recursos principais devem ser protegidos, mas recursos dependentes freqüentemente não requerem proteção. Se um usuário tiver permissão para acessar um recurso principal, faz sentido que, por padrão, o usuário deva ter permissão também para acessar seus recursos dependentes.

Implementando o Controle de Acesso em Beans Corporativos

Se você criar novos beans corporativos que requerem proteção por políticas de controle de acesso, será necessário fazer o seguinte:

1. Criar um novo bean corporativo, assegurando que ele estenda de `com.ibm.commerce.base.objects.ECEntityBean`.
2. Assegurar que a interface remota do bean estenda a interface `com.ibm.commerce.security.Protectable`.
3. Se os recursos com os quais o bean interage estiverem agrupados por um atributo diferente do nome de classe trademark="Java do recurso, a interface remota do bean também deverá estender a interface `com.ibm.commerce.grouping.Groupable`.
4. A classe de bean corporativo contém implementações padrão para os seguintes métodos:
   - `getOwner`
   - `fulfills`
   - `getGroupingAttributeValue`

   Substitua os métodos que precisar. No mínimo, você deverá substituir o método `getOwner`.
   As implementações padrão desses métodos são exibidas nos fragmentos de código a seguir.

```
*****************************************************************************
public Long getOwner() throws Exception, java.rmi.RemoteException
{
   return null;
}
*****************************************************************************
*****************************************************************************
public boolean fulfills(Long member, String relationship)
   throws Exception, java.rmi.RemoteException
{
      return false;
}
*****************************************************************************
*****************************************************************************
public Object getGroupingAttributeValue(String attributeName,
   GroupingContext context) throws Exception, java.rmi.RemoteException
{
      return null;
}
*****************************************************************************
```

A seguir estão exemplos de implementações desses métodos, com base no bean OrderBean:

```
*****************************************************************************
public Long getOwner() throws Exception, java.rmi.RemoteException
{
   com.ibm.commerce.common.objects.StoreEntityAccessBean storeEntAB = new
   com.ibm.commerce.common.objects.StoreEntityAccessBean();
   storeEntAB.setInitKey_storeEntityId(getStoreEntityId().toString());
   return storeEntAB.getMemberIdInEJBType();
}
*****************************************************************************
*****************************************************************************
public boolean fulfills(Long member, String relationship)
  throws Exception, java.rmi.RemoteException
   {
      if (relationship.equalsIgnoreCase("creator"))
      {
         return member.equals(getMemberId());
      }
      else if (relationship.equalsIgnoreCase (
         com.ibm.commerce.base.helpers.EJBConstants.
         SAME_ORGANIZATIONAL_ENTITY_AS_CREATOR_RELATION)) {
            com.ibm.commerce.user.objects.UserAccessBean creator = new
               com.ibm.commerce.user.objects.UserAccessBean();
            creator.setInitKey_MemberId(getMemberId().toString());
            com.ibm.commerce.user.objects.UserAccessBean ab = new
               com.ibm.commerce.user.objects.UserAccessBean();
            ab.setInitKey_MemberId(member.toString());
            if (ab.getParentMemberId().equals(creator.getParentMemberId()))
               return true;
          }
```

```
      return false;
}
*****************************************************************************
*****************************************************************************
public Object getGroupingAttributeValue(String attributeName,
   GroupingContext context) throws Exception
   {
      if (attributeName.equalsIgnoreCase("Status"))
         return getStatus();
   return null;
   }
*****************************************************************************
```

5. Criar (ou recriar) o bean de acesso e o código gerado do bean corporativo.

Implementando o Controle de Acesso em Beans de Dados

Se um bean de dados tiver de ser protegido, ele pode ser protegido direta ou indiretamente por políticas de controle de acesso. Se um bean de dados for protegido diretamente, então, existe uma política de controle de acesso que aplica-se a esse bean de dados específico. Se um bean de dados for protegido indiretamente, ele delega proteção para outro bean de dados, para o qual existe uma política de controle de acesso.

Se você criar um novo bean de dados que deve ser protegido diretamente por uma política de controle de acesso, o bean de dados deve fazer o seguinte:

1. Implementar a interface `com.ibm.commerce.security.Protectable`. Dessa forma, o bean precisa fornecer uma implementação dos métodos `getOwner()` and `fulfills(Long member, String relationship)`. Eles devem ser implementados na interface remota do bean.

   Quando um bean de dados implementa a interface Protectable, o gerenciador de bean de dados chama o método `isAllowed` para determinar se o usuário tem os privilégios de controle de acesso apropriados, de acordo com a política de controle de acesso atual. O método `isAllowed` é descrito pelo seguinte fragmento de código:

   ```
   IsAllowed(Context, "Display", protectable_databean);
   ```

2. Se os recursos com os quais o bean interage estiverem agrupados por um atributo diferente do nome da classe Java do recurso, o bean deverá implementar a interface `com.ibm.commerce.grouping.Groupable`.
3. Implementar a interface `com.ibm.commerce.security.Delegator`. Essa interface é descrita pelo seguinte fragmento de código:

   ```
   Interface Delegator {
    Protectable getDelegate();
   }
   ```

   Nota: Para estar diretamente protegido, o método `getDelegate` deve retornar o próprio bean de dados (ou seja, o bean de dados delega a si mesmo com a finalidade de controle de acesso).

A distinção entre quais beans de dados devem ser protegidos diretamente versus quais devem ser protegidos indiretamente é similar à distinção entre os recursos principal e dependente. Se o objeto bean de dados pode existir por si só, ele deve ser protegido diretamente. Se a existência do bean de dados depender da existência de outro bean de dados, então, deve delegar ao outro bean de dados a proteção.

Um exemplo de bean de dados que deve ser protegido diretamente é o bean de dados Order. Um exemplo de um bean de dados que deve ser protegido indiretamente é o bean de dados OrderItem.

Se você criar um novo bean de dados que não é protegido indiretamente por uma política de controle de acesso, o bean de dados deve fazer o seguinte:

1. Implementar a interface `com.ibm.commerce.security.Delegator`. Essa interface é descrita pelo seguinte fragmento de código:

```
Interface Delegator {
 Protectable getDelegate();
}
```

   Nota: O bean de dados retornado por `getDelegate` deve implementar a interface Protectable.

Se um bean de dados não implementar a interface Delegator, ele será preenchido sem a proteção das políticas de controle de acesso.

Implementando o Controle de Acesso em Comandos do Controlador

Ao criar um novo comando do controlador, a classe de implementação para o novo comando deverá estender a classe `com.ibm.commerce.commands.ControllerCommandImpl` e sua interface deverá estender a interface `com.ibm.commerce.command.ControllerCommand` .

Para políticas no nível do comando para comandos do controlador, o nome da interface do comando é especificado como um recurso. Para que um recurso seja protegido, deve ser implementada a interface Protectable. De acordo com o modelo de programação do WebSphere Commerce, isso é executado estendendo-se a interface do comando a partir da interface `com.ibm.commerce.command.ControllerCommand` e estendendo-se a implementação do comando a partir de `com.ibm.commerce.commands.ControllerCommandImpl` . A interface `ControllerCommand` estende `com.ibm.commerce.command.AccCommand` , que por sua vez, estende `Protectable`. A interface `AccCommand` é a interface mínima que um comando deve implementar para ser protegida pelo controle de acesso no nível de comando.

Se o comando acessar recursos que devem ser protegidos, crie uma variável de instância particular do tipo `AccessVector` para receber os recursos. Em

seguida, substitua o método getResources, uma vez que a implementação padrão desse método é retornar um valor nulo e, portanto, não ocorrerá nenhuma verificação de recurso.

No novo método getResources, você deve retornar uma matriz de recursos ou de pares recurso-ação sobre a qual o comando pode agir. Quando uma ação não é especificada explicitamente, ela é padronizada com o nome da interface do comando que está sendo executado.

Além disso, recomenda-se que o método determine se ele deve instanciar o recurso ou se ele pode utilizar a variável de instância existente que contém a referência para o recurso. Verificar se o objeto de recurso já existir pode ajudar a melhorar o desempenho do sistema. Em seguida, você pode utilizar o mesmo método getResources, se necessário, no método performExecute do novo comando do controlador.

A seguir é fornecido um exemplo do método getResources:

```
private AccessVector resources = null;

public AccessVector getResources() throws ECException {

   if (resources == null) {
      OrderAccessBean orderAB = new OrderAccessBean();
      orderAB.setInitKey_orderId(getOrderId().toString());
      resouces = new AccessVector(orderAB);
      }
   return resources;
}
```

Como um exemplo, considere o comando OrderItemUpdate. O método getResources desse comando retorna os objetos protegíveis Order e User. Como a ação não é especificada, o padrão é a interface do comando OrderItemUpdate.

Vários recursos podem ser retornados pelo método getResources. Quando isso ocorre, uma política que fornece acesso ao usuário para todos os recursos especificados precisa ser encontrada se a ação dever ser executada. Se um usuário tiver acesso a dois de três recursos, a ação pode não prosseguir (três de três seria necessário).

Se você precisar executar verificação adicional de parâmetros ou resolver parâmetros no comando de controlador, pode utilizar o método validateParameters(). Isso é opcional.

Verificação Adicional no Nível do Recurso

Nem sempre é possível determinar todos os recursos que precisam ser protegidos, no momento em que o método getResources do comando do controlador é chamado.

Se necessário, um comando de tarefa também pode implementar um método getResources para retornar uma lista de recursos, na qual o comando pode agir.

Outra maneira de a verificação no nível do recurso é fazer chamadas diretas ao gerenciador de políticas de controle de acesso, utilizando o método checkIsAllowed(Object resource, String action). Esse método está disponível para qualquer classe que se estenda da classe com.ibm.commerce.command.AbstractECTargetableCommand. Por exemplo, as seguintes classes se estendem da classe AbstractECTargetableCommand:

- com.ibm.commerce.command.ControllerCommandImpl
- com.ibm.commerce.command.DataBeanCommandImpl

O método checkIsAllowed também está disponível para as classes que estendem a classe com.ibm.commerce.command.AbstractECCommand. Por exemplo, a seguinte classe se estende a partir da classe AbstractECCommand :

- com.ibm.commerce.command.TaskCommandImpl

A seguir é mostrada a assinatura do métodocheckIsAllowed :

```
void checkIsAllowed(Object resource, String action)
   throws ECException
```

Esse método lança uma ECApplicationException se o usuário atual não tiver permissão para executar a ação especificada no recurso especificado. Se o acesso for concedido, então o método simplesmente será retornado.

Controle de Acesso para Comandos "create"

Como o método getResources é chamado antes do método performExecute em um comando, é necessária uma abordagem diferente para controle de acesso de recursos que ainda não foram criados. Por exemplo, se você tiver um WidgetAddCmd, o método getResources não poderá retornar o recurso que está prestes a ser criado. Nesse caso, o método getResources deve retornar o criador dos recursos. Por exemplo, um comando é criado por uma fábrica de comandos, um pedido é criado em uma loja e um usuário é criado em uma organização.

Implementações Padrão para Controle de Acesso no Nível de Comando

Para controle de acesso no nível de comando, a implementação padrão do método getOwner() retorna o memberId do proprietário da loja, se o storeId for especificado. Se o storeId não for especificado, o memberId da organização raiz será retornado (memberId = -2001).

A implementação padrão do método getResources() retorna null.

A implementação padrão de validateParameters() não faz nada.

Implementando Políticas de Controle de Acesso em Exibições

O controle de acesso no nível de recurso para exibições é executado pelo gerenciador de bean de dados. O gerenciador de bean de dados é chamado nos seguintes casos:

1. Quando o modelo JSP inclui a tag <useBean> e o bean de dados não está na lista de atributos.
2. Quando o modelo JSP inclui o seguinte método ativo:

```
DataBeanManager.activate(xyzDatabean, request);
```

Nota: Todo bean de dados que precise ser protegido (direta ou indiretamente) deve implementar a interface Delegator. Todo bean de dados que tiver que ser protegido diretamente será delegado a si mesmo e, assim, também deverá implementar a interface Protectable. Os beans de dados protegidos indiretamente devem ser delegados a um bean de dados que implemente a interface Protectable.

Apesar de não ser recomendável, um desvio das verificações de controle de acesso ocorre nos seguintes casos:

1. Se o modelo JSP fizer chamadas diretas a beans de acesso, em vez de utilizar beans de dados.
2. Se o modelo JSP chamar diretamente o método populate() do bean de dados.

Se os resultados de um comando de controlador precisarem ser encaminhados para uma exibição (utilizando o ForwardViewCommand), o controle de acesso no nível de comando não será executado nas exibições. Além disso, se o comando de controlador colocar os beans de dados preenchidos (utilizados na exibição) na lista de atributos da propriedade de resposta e avançar para uma exibição, o modelo JSP poderá acessar os dados sem passar pelo gerenciador de bean de dados. Isso requer que as tags <useBean> sejam utilizadas no modelo JSP. Essa pode ser uma maneira de tornar um modelo JSP mais eficiente, uma vez que ele pode ignorar qualquer verificação de controle de acesso no nível de recurso redundante em recursos (beans de dados) para os quais o usuário já obteve acesso por meio do comando de controlador.

Capítulo 5. Tratamento de Erro e de Mensagens

Tratamento de Erro de Comando

O WebSphere Commerce utiliza uma estrutura de tratamento de erro de comando bem definida que é simples de ser utilizada no código personalizado. Por projeto, a estrutura trata os erros de uma maneira que suporte lojas multiculturais. As seções a seguir descrevem os tipos de exceções que um comando pode emitir, como as exceções são tratadas, como o texto da mensagem é armazenado e utilizado, como as exceções são registradas e como utilizar a estrutura fornecida em seus próprios comandos.

Tipos de Exceções

Um comando pode emitir um dos seguintes tipos de exceções:

ECApplicationException
Essa exceção é emitida se o erro for relatado ao usuário. Por exemplo, quando um usuário digita um parâmetro inválido, um ECApplicationException é emitido. Quando essa exceção é emitida, o controlador da Web não repete o comando, mesmo se ele for especificado como um comando que pode ser repetido.

ECSystemException
Essa exceção é emitida se uma exceção runtime ou um erro de configuração do WebSphere Commerce for detectado. Exemplos desse tipo de exceção incluem exceções de ponteiro nulo e exceções de retorno de transação. Quando esse tipo de exceção é emitido, o controlador da Web repete o comando se ele for possível de ser repetido e se a exceção foi provocada por um bloqueio de banco de dados ou retorno de banco de dados.

Ambas as exceções acima listadas são classes que estendem da classe ECException, que é encontrada no pacote com.ibm.commerce.exception.

Para emitir uma dessas exceções, as informações a seguir devem ser especificadas:

- Nome da exibição de erros
  O controlador da Web procura esse nome na tabela VIEWREG.
- Objeto ECMessage
  Esse valor corresponde ao texto de mensagem contido dentro de um arquivo de propriedades.
- Parâmetros de erro
  Esses pares nome-valor são utilizados para substituir informações na mensagem de erro. Por exemplo, uma mensagem pode conter um

parâmetro para manter o nome do método que emitiu a exceção. Esse parâmetro é definido quando a exceção é emitida, então, quando a mensagem de erro é registrada, o arquivo de log conterá o nome do método real.

- Dados de erro
  Estes são atributos opcionais que podem ser disponibilizados para o modelo JSP através do bean de dados de erro.

O tratamento da exceção esta estritamente integrada com o sistema de logon. Quando uma exceção é emitida, ela é automaticamente registrada.

Arquivos de Propriedades da Mensagem de Erro

Para simplificar a manutenção das mensagens de erro e para suportar lojas multilíngües, o texto para as mensagens de erro é armazenado em arquivos de propriedades. O texto da mensagem do WebSphere Commerce é armazenado no arquivo `ecServerMessages_XX_XX.properties`, em que `_XX_XX` é o indicador de locale (por exemplo, `_en_US`).

O contexto do comando retorna um identificador para indicar o idioma utilizado pelo cliente. Quando uma mensagem é solicitada, o controlador da Web determina qual arquivo de propriedade será utilizado, com base no identificador de idioma.

Há dois tipos de mensagens definidas no arquivo `ecServerMessages_XX_XX.properties`: mensagens do usuário e mensagens do sistema. As mensagens do usuário são exibidas aos clientes em seus navegadores. As mensagens do usuário e do sistema são capturadas automaticamente no log de mensagens.

Quando um erro é emitido, um dos parâmetros requeridos é um objeto de mensagem. Para ECSystemExceptions, o objeto de mensagem deve conter duas chaves, uma para a mensagem do sistema e uma para a mensagem do usuário. Para ECApplicationExceptions, este objeto de mensagem contém a chave para a mensagem do usuário (as mensagens do sistema não são utilizadas).

Todas as mensagens do sistema são predefinidas. Não é possível criar suas próprias mensagens do sistema. Portanto, quando o código personalizado emite um ECSystemException, ele deve especificar uma chave de mensagem para uma das mensagens predefinidas do sistema. Podem ser criadas mensagens do usuário personalizadas. As novas mensagens do usuário devem ser armazenadas em um arquivo de propriedade separado.

Fluxo do Tratamento de Exceção

O diagrama a seguir mostra o fluxo de informações quando uma exceção é adquirida. Segue uma descrição de cada etapa.

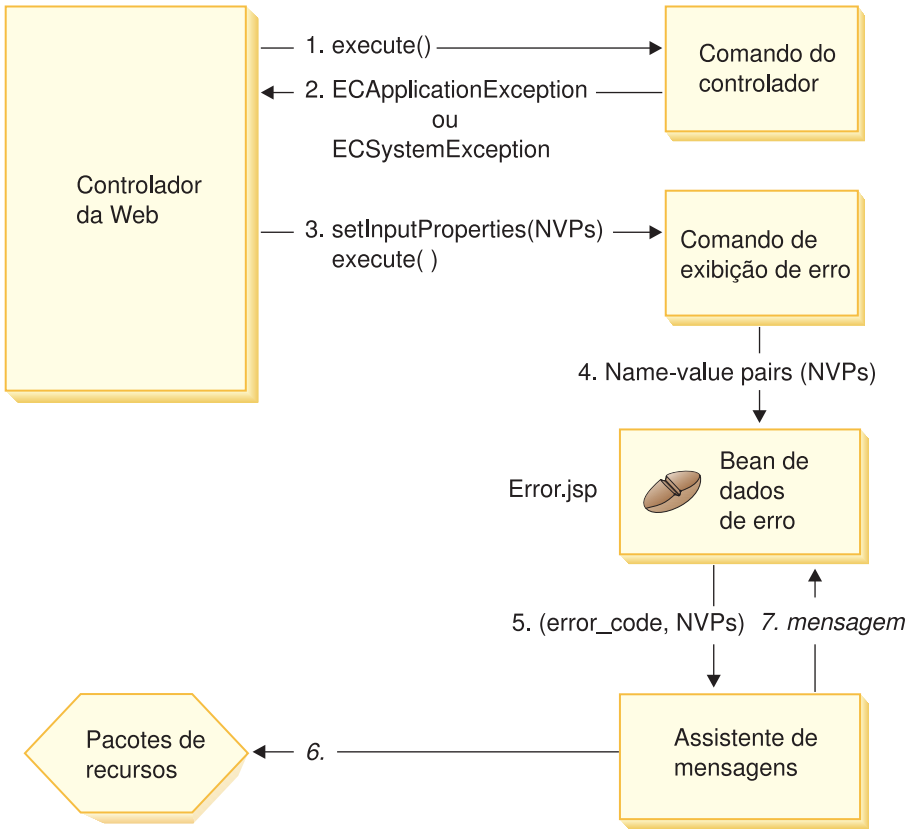


Figura 24.

1. O controlador da Web invoca um comando do controlador.
2. O comando emite uma exceção que é capturada pelo controlador da Web. Essa pode ser ECApplicationException ou ECSystemException. O objeto de exceção contém as seguintes informações:
   - Nome de exibição de erro
   - Objeto ECMessage
   - Parâmetros de erro
   - (opcional) Dados de erro
3. O controlador da Web determina o nome de exibição de erro a partir da tabela VIEWREG e chama o comando de exibição de erros especificado. Ao invocar o comando, o controlador da Web compõe um conjunto de propriedades a partir do objeto ECException e o define para o comando de exibição utilizando o método setInputProperties do comando de exibição.
4. O comando de exibição invoca um modelo JSP de erro (Error.jsp, nesse caso) e os pares nome-valor são passados para o modelo JSP.

5. O ErrorDataBean passa os parâmetros de erro para o objeto assistente de mensagem.
6. O objeto assistente de mensagem obtém a mensagem requerida (utilizando o objeto de mensagem e os parâmetros de erro) do arquivo de propriedades apropriado.
7. O bean de dados de erro retorna a mensagem ao modelo JSP.

Tratamento de Exceção no Código Personalizado

Ao criar novos comandos, é importante incluir o tratamento de exceção apropriado. Você pode tirar vantagens da estrutura de mensagens e do tratamento de erros fornecidos pelo WebSphere Commerce, especificando as informações necessárias ao capturar uma exceção.

A gravação de sua própria lógica de tratamento de exceção, envolve as seguintes etapas:

1. Capturar as exceções no comando que requer processamento especial.
2. Construir um ECApplicationException ou ECSystemException, com base no tipo de exceção capturada.
3. Se ECApplicationException utilizar uma nova mensagem, defina a mensagem em um novo arquivo de propriedades.

Captura e construção de exceções

Para ilustrar as duas primeiras etapas, o fragmento do código a seguir mostra um exemplo de captura de uma exceção de sistema dentro de um comando:

```
try {
// your business logic
}
catch(FinderException e) {
     throw new ECSystemException (ECMessage._ERR_FINDER_EXCEPTION,
          className, methodName, new Object [] {e.toString()}, e);
}
```

O objeto _ERR_FINDER_EXCEPTION ECMessage precedente está definido como segue:

```
public static final ECMessage _ERR_FINDER_EXCEPTION =
new ECMessage (ECMessageSeverity.ERROR, ECMessageType.SYSTEM,
     ECMessageKey._ERR_FINDER_EXCEPTION);
```

O texto de mensagem _ERR_FINDER_EXCEPTION está definido dentro do arquivo `ecServerMessages_xx_XX.properties` (em que *_xx_XX* é um indicador de locale, como *_en_US*), como segue:

```
_ERR_FINDER_EXCEPTION  =
     A seguinte Finder Exception ocorreu durante o processamento: "{0}".
```

Ao capturar uma exceção do sistema, existe um conjunto de mensagens predefinido que pode ser utilizado. Estão descritos na seguinte tabela:

Objeto de Mensagem	Descrição
_ERR_FINDER_EXCEPTION	Emitido quando um erro é retornado de uma chamada de método do localizador EJB.
_ERR_REMOTE_EXCEPTION	Emitido quando um erro é retornado de uma chamada de método remota EJB.
_ERR_CREATE_EXCEPTION	Emitido quando ocorre um erro ao criar uma instância EJB.
_ERR_NAMING_EXCEPTION	Emitido quando um erro é retornado do servidor de nomes.
_ERR_GENERIC	Emitido quando ocorre um erro inesperado do sistema. Por exemplo, uma exceção de ponteiro nulo.

Ao capturar uma exceção de aplicativo, você pode utilizar uma mensagem já existente especificada no arquivo `ecServerMessages_xx_xx.properties` apropriado ou criar uma nova mensagem que é armazenada em um novo arquivo de propriedades. Conforme especificado anteriormente, você *não deve* modificar nenhum dos arquivos `ecServerMessages_xx_XX.properties`.

O fragmento do código a seguir mostra um exemplo de captura de uma exceção de aplicativo dentro de um comando:

```
try {
// your business logic

}
// catch some new type of application exception
catch(//your new exception)
 {
     throw new ECApplicationException (MyMessages._ERR_CUSTOMER_INVALID,
          className, methodName, errorTaskName, someNVPs);
}
```

O objeto _ERR_CUSTOMER_INVALID ECMessage precedente está definido como segue:

```
public static final ECMessage _ERR_CUSTOMER_INVALID =
     new ECMessage (ECMessageSeverity.ERROR, ECMessageType.USER,
     MyMessagesKey._ERR_CUSTOMER_INVALID, "ecCustomerMessages");
```

Ao construir novas mensagens de usuário, você deve atribuí-las a um tipo de USER, conforme mostrado abaixo:

`ECMessageType.USER`

O texto para a mensagem _ERR_CUSTOMER_INVALID está contido no arquivo `ecCustomerMessages.properties`. Este arquivo deve residir em um diretório que esteja no caminho da classe. O texto é definido como segue:

```
_ERR_CUSTOMER_INVALID = Invalid ID "{0}"
```

Criação de Mensagens

Se o seu comando emitir um ECApplicationException que utiliza uma nova mensagem, será necessário criar essa nova mensagem. A criação de uma nova mensagem envolve as seguintes etapas:

1. Criar uma nova classe que contenha as chaves de mensagem.
2. Criar uma nova classe que contenha os objetos ECMessage.
3. Criar um pacote de recursos.
4. Testar a unidade da mensagem.

Detalhes sobre cada etapa são fornecidos nas seções a seguir.

Criação de uma classe para as chaves de mensagens

A primeira etapa na criação de novas mensagens do usuário é criar uma classe que contenha as novas chaves de mensagens. Uma chave de mensagem é um indicador exclusivo que é utilizado pelo serviço de logon para localizar o texto de mensagem correspondente em um pacote de recursos. Essa nova classe deve ser criada dentro de seu próprio pacote e armazenada em um projeto separado dos projetos do WebSphere Commerce.

Considere um exemplo, chamado `MyNewMessages`, no qual você cria uma nova classe, chamada `MyMessageKeys`, que contém as chaves de mensagem `_ERR_CUSTOMER` e `_ERR_CUSTOMER_INVALID_ID` e coloca esta classe no pacote `com.mycompany.messages`. Neste caso, a definição da classe aparece da seguinte forma:

```
public class MyMessageKeys
{
    public static String _ERR_CUSTOMER="_ERR_CUSTOMER";
    public static String _ERR_CUSTOMER_INVALID_ID="_ERR_CUSTOMER_INVALID_ID";
}
```

O fornecimento de envoltórios de Cadeia para as chaves de mensagens permite ao compilador verificar a sua validade.

Criação de uma classe para os objetos ECMessage

Dentro do mesmo pacote onde você criou a classe para suas chaves de mensagem, crie outra classe que contenha os objetos ECMessage. A classe ECMessage define a estrutura de um objeto de mensagem. Ele é utilizado para recuperar e persistir mensagens do texto sensível ao local.

O objeto de mensagem possui os seguintes atributos: gravidade, tipo, chave, pacote de recursos e pacote de recursos associados. Há vários métodos construtores para esta classe. Consulte a seção “Referência” da ajuda online do WebSphere Commerce para obter detalhes completos.

Seguindo o exemplo MyNewMessages, crie uma nova classe chamada MyMessages dentro do pacote com.mycompany.messages, como segue:

```
import com.ibm.commerce.ras.*;

public class MyMessages
    {
        static String myResourceBundle = "ecCustomerMessages";

        public static ECMessage _ERR_CUSTOMER = new ECMessage
             (ECMessageSeverity.ERROR,ECMessageType.USER,
                   MyMessageKeys._ERR_CUSTOMER, myResourceBundle);
        public static ECMessage _ERR_CUSTOMER_INVALID_ID = new ECMessage
             (ECMessageSeverity.ERROR, ECMessageType.USER,
                  MyMessageKeys._ERR_CUSTOMER_INVALID_ID,
                  myResourceBundle);

    }
```

No fragmento de código acima, a instrução de importação é requerida para a criação do objeto ECMessage. O objeto MyMessage._ERR_CUSTOMER é uma mensagem de usuário de gravidade ERROR. O MyMessageKeys._ERR_CUSTOMER é utilizado pelo serviço de logon do WebSphere Commerce para encontrar o texto da mensagem contido no arquivo de propriedades *ecCustomerMessages*.

Criando um pacote de recursos de mensagem do usuário

Você deve criar um novo pacote de recursos, onde as chaves de mensagens com texto da mensagem correspondente são armazenados. Esse pacote de recursos pode ser implementado como um objeto Java ou como um arquivo de propriedades. Recomenda-se que você utilize os arquivos de propriedades, já que eles são mais fáceis para conversão e manutenção. Os arquivos de propriedades são utilizados para as mensagens do WebSphere Commerce.

Para continuar o exemplo MyNewMessages, crie um arquivo de texto pelo nome de ecCustomerMessages.properties. Se as mensagens tiverem que ser utilizadas por um único servlet de loja simples, coloque este arquivo no seguinte diretório:

unidade:\WebSphere\AppServer\installedApps\WC_EnterpriseApp_*instanceName*.ear\
wcstores.war\WEB-INF\classes

Se as mensagens tiverem que ser utilizadas por um único servlet de ferramentas, coloque este arquivo no seguinte diretório:

unidade:\WebSphere\AppServer\installedApps\WC_EnterpriseApp_*instanceName*.ear\
wctools.war\WEB-INF\classes

Se as mensagens forem utilizadas globalmente por qualquer no aplicativo corporativo, coloque o arquivo no seguinte diretório:

```
unidade:\WebSphere\AppServer\installedApps\WC_EnterpriseApp_instanceName.ear\
   properties
```

Visto que o arquivo de propriedades contém pares de chaves de mensagens e o texto da mensagem correspondente, o arquivo ecCustomerMessages.properties conterá as seguintes linhas:

```
_ERR_CUSTOMER_MESSAGE = The customer message "{0}".
_ERR_CUSTOMER_INVALID_ID = Invalid ID "{0}".
```

Teste de unidade de uma mensagem

Depois que as classes que contêm as chaves de mensagens, as classes que contêm objetos ECMessage e o pacote de recursos tiverem sido criados, você deve testar a unidade de suas novas mensagens.

Para testar as novas mensagens descritas nas seções anteriores, proceda da seguinte forma:

1. Crie uma nova classe com a finalidade de teste. Nesse caso, crie MyTestingClass.
2. Inclua a seguinte instrução de importação na classe

   ```
   import com.ibm.commerce.ras.*;
   ```
3. Inclua um método main() na classe.
4. Examine o seguinte fragmento de código. Modifique-o de acordo com seus próprios requisitos (por exemplo, assegure-se de que o caminho do diretório indique um arquivo de configuração válido no seu sistema) e insira-o no método main()

   ```
   // the fileName String variable should point
   // to a valid WebSphere Commerce configuration file:
     String fileName = "E:\\WebSphere\\CommerceServer\\instances
         \\demo\\xml\\demo.xml";

      LogConfiguration config = LogConfiguration.getUniqueInstance ();
      config.initialize (fileName, "testClone");

      ECMessageLog.out (MyMessage._ERR_CUSTOMER,
           "MyTestingClass", "main", "Hello");
   ```

   As três primeiras linhas do código inicializam o serviço de logom do WebSphere Commerce. A última linha do código instrui o ECMessageLog para que imprima a mensagem MyMessage._ERR_CUSTOMER. MyTestingClass e main fazem parte do formato de rastreamento do log. A cadeia Hello é colocada no marcador {0} da mensagem definida no arquivo ecCustomerMessages.properties.
5. Execute a classe. No arquivo de registro, o rastreamento da mensagem aparece semelhante ao seguinte:

```
==============
 TimeStamp:      2000-11-29 16:41:42.5
 Thread ID:     <main>
 Class:          MyTestingClass
 Method:         main
 Severity:       1
 Message Text:   The customer message "Hello".
```

Você pode executar um teste semelhante para ver uma segunda mensagem.

Rastreamento do Fluxo de Execução

O WebSphere Commerce inclui a classe ECTrace que é utilizada para rastrear o fluxo de execução de componentes em execução no WebSphere Commerce Server. A classe ECTrace faz parte do pacote com.ibm.commerce.ras.

Ao criar novas lógicas de negócios, você pode inserir um rastreamento em seu código para rastrear um método com a finalidade de depuração. As informações do rastreio são capturadas no log de rastreamento. Você pode especificar uma entrada e ponto de saída para o rastreamento. Além disso, é possível especificar que dados específicos sejam rastreados entre esses dois pontos.

Para utilizar o rastreamento, ele deve estar ativado para o componente onde gostaria de executar o rastreamento. Para ativar o rastreamento para um componente específico, você pode utilizar tanto o Administration Console como o Gerenciador de Configuração.

Ao rastrear o código personalizado, você *deve* utilizar o componente EXTERN. No Gerenciador de Configuração, é chamado *External*.

Para definir o ponto de entrada de um rastreamento dentro de seu código, utilize a seguinte sintaxe:

```
ECTrace.entry (ECTraceIdentifiers.COMPONENT_EXTERN, myClassName, myMethodName);
```

em que *myClassName* é a representação da cadeia da classe que contém o método rastreado. Como esta cadeia pode ser utilizada para rastrear análise de arquivo, ela deve conter o nome de classe completo. Se o método sendo rastreado for estático, o exemplo da declaração de myClassName será

```
String myClassName = "com.mycompany.agrouping.MyTracedClass";
```

Se o método sendo rastreado não for estático, o exemplo da declaração de myClassName será

```
String myClassName = this.getClass().getName();
```

Para definir o ponto de rastreamento para rastrear dados dentro de um método, utilize a seguinte sintaxe:

```
ECTrace.trace (ECTraceIdentifiers.COMPONENT_EXTERN, myClassName,
     myMethodName, myText);
```

em que *myText* é o texto que deve aparecer no log de rastreamento.

Para definir o ponto de saída de um rastreamento dentro de seu código, utilize a seguinte sintaxe:

```
ECTrace.exit (ECTraceIdentifiers.COMPONENT_EXTERN, myClassName,
     myMethodName);
```

Se precisar rastrear o objeto retornado do método sendo rastreado, defina o ponto de saída da seguinte forma:

```
ECTrace.exit (ECTraceIdentifiers.COMPONENT_EXTERN, myClassName,
     myMethodName, returnedObject);
```

em que `returnedObject` representa o objeto Java retornado pelo método.

Considere um exemplo em que você precisa rastrear o método performExecute de um novo comando do controlador chamado MyNewControllerCmd. O fragmento de código a seguir mostra como utilizar os métodos ECTrace dentro de seu método performExecute.

```
public void performExecute() throws ECException {
     ECTrace.entry(ECTraceIdentifiers.COMPONENT_EXTERN,
               this.getClass().getName(), "performExecute");

     super.performExecute();


//////////////////////////////////////////
//   Some of your business logic        //
//////////////////////////////////////////


     ECTrace.trace(ECTraceIdentifiers.COMPONENT_EXTERN,
          this.getClass().getName(), "performExecute",
          "My code is great!");

//////////////////////////////////////////
//   Some more business logic           //
//////////////////////////////////////////


     ECTrace.exit(ECTraceIdentifiers.COMPONENT_EXTERN,
               this.getClass().getName(), "performExecute");

}
```

Quando o método performExecute anterior é chamado, o arquivo de log de rastreamento captura as seguintes informações:

```
==============
TimeStamp:     2000-12-05 17:32:00.257
Thread ID:    <P=502832:O=0:CT>
Component:     EXTERN
Class:         com.mycompany.agrouping.MyNewControllerCmd
Method:        performExecute
Trace:         ENTRY POINT
==============
TimeStamp:     2000-12-05 17:32:00.257
Thread ID:    <P=502832:O=0:CT>
Component:     EXTERN
Class:         com.mycompany.agrouping.MyNewControllerCmd
Method:        performExecute
Trace:         My code is great!
==============
TimeStamp:     2000-12-05 17:32:00.258
Thread ID:    <P=502832:O=0:CT>
Component:     EXTERN
Class:         com.mycompany.agrouping.MyNewControllerCmd
Method:        performExecute
Trace:         EXIT POINT
```

Recomenda-se que o rastreamento seja utilizado apenas em funções principais. O rastreamento não é permitido em vários idiomas, visto que destina-se a ser utilizado pelos Desenvolvedores da Loja. Isso contrasta com as mensagens, que são permitidas para vários idiomas pois as mensagens do sistema são utilizadas com finalidade administrativa e as mensagens do usuário são exibidas aos clientes.

Tratamento de Erro do Modelo JSP

O tratamento de erro para modelos JSP podem ser executados de diversas maneiras:

- Tratamento de erro de dentro da página
  Para arquivos JSP que requerem tratamento e recuperação de erros mais complexos, o arquivo pode ser escrito para tratar diretamente erros a partir do bean de dados. O arquivo JSP pode capturar exceções emitidas pelo bean de dados ou pode verificar se há códigos de erros definidos dentro de cada bean de dados, dependendo de como o bean de dados foi ativado. O arquivo JSP pode, então, tomar uma ação de recuperação apropriada com base no erro recebido. Observe que um arquivo JSP pode utilizar qualquer combinação dos seguintes escopos de tratamento de erro.
- JSP com erro no nível de página
  Um arquivo JSP também pode especificar seu próprio modelo JSP de erro padrão a partir de uma exceção que ocorre dentro de si mesmo através da tag de erro do JSP. Isso permite que um programa JSP especifique seu próprio tratamento de um erro. Um arquivo JSP que não especifique uma tag de erro JSP falhará no modelo de erro de JSP do nível do aplicativo. No

JSP de erro de nível de página, ele deve chamar a classe auxiliadora JSP (com.ibm.server.JSPHelper) para retornar a transação atual.

- JSP com erro no nível do aplicativo
  Um aplicativo no WebSphere pode especificar um modelo de JSP com erro padrão quando ocorre uma exceção de qualquer um de seus servlets ou arquivos JSP. O modelo de JSP com erro de nível de aplicativo pode ser utilizado como um tratamento de erro no nível de shopping center ou nível de loja (para um modelo de loja simples). No modelo de JSP com erro de nível de aplicativo, uma chamada deve ser feita para a classe auxiliar do servlet para retornar a transação atual. Isto porque o controlador da Web não estará no caminho de execução para retornar a transação. Sempre que possível, você deveria contar com os dois tipos anteriores de tratamento de erro de JSP. Utilize a estratégia de tratamento de erro de nível de aplicativo apenas quando for necessário.

Capítulo 6. Implementação do Comando

Esta seção fornece informações sobre como escrever novos comandos de controlador, de tarefa e de beans de dados. Ela também descreve como estender comandos de controlador, de tarefa e de beans de dados existentes.

Nota: Business Este capítulo não descreve comandos de política de negócios. Para obter informações sobre os comandos de política de negócios, consulte Capítulo 7, "Acordos Comerciais e Políticas de Negócios (Business Edition)" na página 153.

Novos Comandos - Introdução

O modelo de programação do WebSphere Commerce define quatro tipos de comandos: controlador, tarefas, exibição e comandos do beans de dados.Ao criar novas lógicas de negócios para seu aplicativo e-commerce, convém criar novo controlador, tarefa e comandos de bean de dados. Você pode não precisar criar novos comandos de exibição. Mais informações sobre os comandos de exibição poderão ser encontradas posteriormente nessa seção.

Os novos comandos devem implementar suas interfaces correspondentes (que, sucessivamente, devem estender de uma interface já existente). Para simplificar a escrita do comando, o WebSphere Commerce inclui uma classe de implementação abstrata para cada tipo de comando. Os novos comandos devem se estender destas classes.

Como uma visão geral, a tabela a seguir fornece informações sobre de qual classe de implementação um novo comando deve se estender e qual interface deve implementar:

Tipo de comando	Exemplo de nome de comando	Estender-se de	Implementa exemplo de interface
Comando do controlador	MyControllerCmdImpl	com.ibm.commerce.command.ControllerCommandImpl	MyControllerCmd
Comando de tarefas	MyTaskCmdImpl	com.ibm.commerce.command.TaskCommandImpl	MyTaskCmd
Comando do bean de dados	MyDataBeanCmdImpl	com.ibm.commerce.command.DataBeanCommandImpl	MyDataBean

Nota: Todos os espaços nos nomes das classes de implementação são apenas com a finalidade de apresentação.

O diagrama a seguir ilustra o relacionamento entre a interface e a classe de implementação de um novo comando do controlador com a classe de implementação e a interface abstrata já existente. A classe e a interface abstratas são encontradas no pacote com.ibm.commerce.command.

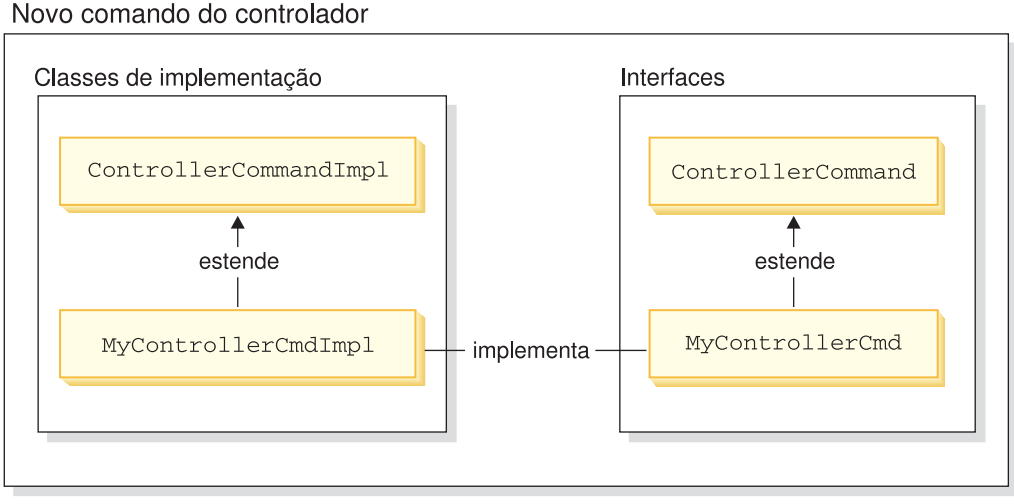


Figura 25.

O diagrama a seguir ilustra o relacionamento entre a interface e a classe de implementação de um novo comando de tarefas com a classe de implementação e a interface abstratas já existentes. A classe e a interface abstratas são encontradas no pacote com.ibm.commerce.command.

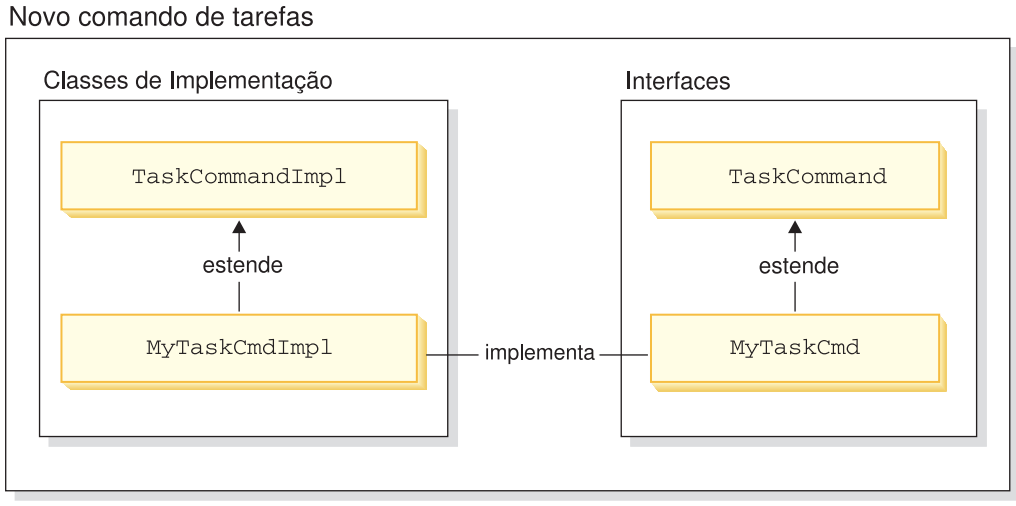


Figura 26.

O diagrama a seguir ilustra o relacionamento entre a interface e a classe de implementação de um novo comando de bean de dados com a classe de implementação e a interface abstratas já existentes. A classe e a interface abstratas são encontradas no pacote com.ibm.commerce.command.

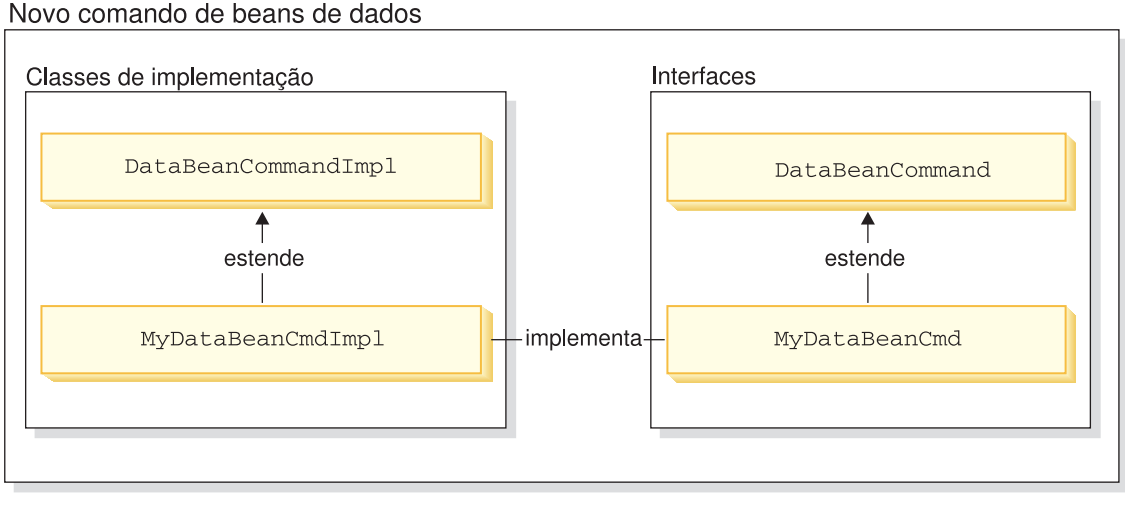


Figura 27.

Um comando de exibição possui duas funções principais: formatar uma resposta e enviar a resposta ao cliente. São fornecidos vários comandos de exibição genéricos que enviam a resposta aos clientes, utilizando protocolos diferentes. A função de formatação geralmente é tratada pelo comando de

exibição invocando um modelo JSP. Por exemplo, o comando de exibição RedirectViewCommand direciona o cliente a uma URL para obter a resposta (a resposta é, então, formatada por um modelo JSP especificado). O comando de exibição ForwardViewCommand encaminha o pedido ao modelo JSP para formatação e a página é exibida ao cliente.

Utilizando esse modelo de comando de exibição, você pode criar novas *exibições* (a resposta ao cliente) criando novos modelos JSP. O modelo JSP deve, portanto, ser invocado por um dos comandos de exibição existentes.

Empacotamento de Código Personalizado

Ao criar código personalizado, você deve seguir uma estrutura organizacional de código particular. Em geral, o código personalizado é mantido em pacotes e projetos que são separados daqueles incluídos com o WebSphere Commerce.

Ao criar novos comandos, você deve colocá-los em um nome de pacote que seja apropriado para suas necessidades de negócios. Ou seja, se os comandos se aplicam a uma determinada loja, coloque-os em um pacote que seja exclusivo para a loja. Se eles se aplicam a mais de uma loja, empacote-os concordantemente. Por exemplo, você deve ter os seguintes pacotes:

- `com.bigbusiness.storeA.commands`
- `com.bigbusiness.storeB.commands`
- `com.bigbusiness.commands`

A estrutura de empacotamento anterior permite a diferenciação entre a lógica de negócios em um nível de loja. Além disso, esses pacotes devem ser armazenados em um projeto que esteja separado dos projetos do WebSphere Commerce.Por exemplo, os pacotes precedentes podem ser colocados em um projeto chamado `BigBusinessCustomCode`.

Ao criar novos beans de dados, eles devem ser mantidos em um pacote que esteja separado da lógica do comando, portanto, esse pacote pode ser mantido dentro do projeto que armazena os pacotes de comandos. Do exemplo anterior, você colocaria o pacote `com.bigbusiness.databeans` dentro do projeto `BigBusinessCustomCode`.

Ao criar novos beans de entidade, eles devem ser armazenados em um único projeto. Portanto, você pode ter o projeto `BigBusinessCustomEntityBeans` que contém o pacote `com.bigbusiness.objects`.

Essa estratégia de empacotamento é requerida com a finalidade de implementação.

Contexto do Comando

O contexto do comando é um tratamento para o controlador da Web. Os comandos podem obter informações do controlador da Web utilizando o contexto do comando. Exemplos de informações disponíveis incluem o ID do usuário, o objeto do usuário, o identificador de idioma e o identificador de loja.

Ao escrever um comando, você tem acesso ao contexto do comando chamando o método getCommandContext() da superclasse do comando. O contexto do comando é definido para o comando do controlador quando o comando é invocado pelo controlador da Web. Um comando do controlador deveria propagar o contexto do comando para qualquer comando do controlador ou de tarefas que são invocados durante o processamento. Um comando pode obter as seguintes informações-chave do contexto do comando:

getUserId() e getUser()
: Obtém o ID do usuário ou o objeto do usuário atuais. O ID do usuário para a sessão atual é salvo em um contexto de sessão. O contexto de sessão pode ser persistido em uma das duas maneiras: utilizando o cookie do WebSphere Commerce ou um objeto de sessão persistente do WebSphere Application Server. O contexto do comando oculta a complexidade do gerenciamento de sessões a partir de um comando.

getStoreId(), getStore() e getStore(storeId)
: Obtém a loja associada ao pedido atual. O controlador da Web retorna o ID da loja na URL. Se o ID da loja não estiver especificado na URL, ele poderá ser recuperado do objeto de sessão que foi salvo a partir do pedido anterior. O ambiente do WebSphere Commerce runtime mantém um conjunto de objetos que são freqüentemente acessados.Por exemplo, ele mantém o conjunto de objetos da loja. Um comando deveria sempre obter o objeto da loja a partir do contexto do comando para tirar vantagem da cache do objeto no controlador da Web. É possível obter a loja atual chamando o método getStore() ou obter um objeto de loja específico chamando o método getStore(storeId) no contexto do comando.

getLanguageId()
: Retorna o ID do idioma que deve ser utilizado para o pedido atual. O controlador da Web implementa uma Estrutura de Globalização. O conceito dessa estrutura é determinar um idioma que seja da preferência do usuário e suportado pela loja. Se a URL contiver um ID de idioma, o controlador da Web determinará se esse idioma é suportado pela loja, em caso positivo, esse será o idioma retornado pelo método getLanguageId(). Se nenhum ID de idioma tiver sido incluído na URL, o controlador da Web percorre uma árvore de decisões para determinar se existe um ID de idioma (suportado pela

loja) no objeto da sessão atual ou nas preferências registradas do usuário ou, como última alternativa, retornará o ID do idioma padrão da loja.

getCurrency()
Retorna a moeda a ser utilizada para o pedido atual. Como a moeda é parte da Estrutura de Globalização, a lógica desse método é semelhante a do método getLanguageId().

getCurrentTradingAgreements() e getTradingAgreement(tradingAgreementId)
Retorna o conjunto de contratos de negócios utilizados para a sessão atual. Esse conjunto pode ser todos os contratos de negócios para os quais o usuário tem direitos ou pode ser um subconjunto definido pelo comando ContractSetInSession. Um comando deve sempre obter o objeto do acordo de negócios do contexto do comando para beneficiar-se da cache do objeto no controlador da Web. É possível obter o acordo de negócios atual, chamando o método getCurrentTradingAgreements() ou obter um objeto de acordo de negócios específico, chamando o método getTradingAgreement(tradingAgreementId) no contexto do comando.

O contexto do comando deve ser utilizado como um objeto apenas para leitura. Você não deve chamar os métodos instaladores. Os métodos instaladores são reservados para serem utilizados pelo ambiente do WebSphere Commerce runtime e podem ser reprovados em releases futuros.

Para obter detalhes completos sobre a API (interface de programação de aplicativo) do contexto do comando, consulte a seção “Referência” da ajuda online do WebSphere Commerce.

Novos Comandos do Controlador

Conforme mencionado acima, um novo comando do controlador deve estender-se a partir da classe de comando do controlador abstrato (com.ibm.commerce.command.ControllerCommandImpl). Ao escrever um novo comando do controlador, você deve substituir os seguintes métodos da classe abstrata:

- isGeneric()
- isRetriable()
- setRequestProperties(com.ibm.commerce.datatype.TypedProperty reqParms)
- validateParameters()
- getResources()
- performExecute()

Mais informações sobre cada um dos métodos acima são encontradas nas seções a seguir.

Método isGeneric

Na implementação padrão do WebSphere Commerce há vários tipos de usuários.Estes incluem usuários genéricos, convidados e registrados. Dentro do grupo de usuários registrados existem clientes e administradores.

O usuário genérico possui um ID do usuário comum que é utilizado em todo o sistema. Este ID de usuário comum suporta navegação geral no site de uma maneira que minimiza a utilização de recursos do sistema. É mais eficiente utilizar esse ID de usuário comum para navegação geral, já que o controlador da Web não precisa recuperar um objeto de usuário de comandos que podem ser chamados pelo usuário genérico.

O método isGeneric retorna um valor booleano que especifica se o comando pode ou não ser chamado pelo usuário genérico. O método isGeneric de uma superclasse do comando do controlador define o valor como `false` (significando que o invocador deve ser um usuário registrado ou um usuário convidado). Se o seu novo comando do controlador puder ser invocado pelos usuários genéricos, substitua esse método para que retorne `true`.

Você deve substituir esse método para que retorne `true` se o novo comando não buscar ou criar recursos associados a um usuário. Um exemplo de um comando que pode ser invocado por um usuário genérico é o comando ProductDisplay. Ele é sensível para permitir que qualquer usuário possa exibir os produtos. Um exemplo de um comando para o qual um usuário deve ser um usuário convidado ou registrado (e por isso, isGeneric retorna `false`) é o comando OrderItemAdd.

Quando isGeneric retorna um valor `true`, o controlador da Web não cria um novo objeto de usuário para a sessão atual. Como tal, os comandos que podem ser invocados pelo usuário genérico executam mais rápido, visto que o controlador da Web não precisa recuperar um objeto do usuário.

A sintaxe para utilizar este método a fim de permitir que usuários genéricos chamem um comando é mostrada abaixo:

```
public boolean isGeneric()
{
     return true;
}
```

Método isRetriable

O método isRetriable retorna um valor booleano que especifica se o comando pode ou não ser repetido em uma exceção de retorno de transação. O método isRetriable da superclasse do novo comando do controlador retorna um valor

false. Você deve substituir esse método e retornar um valor true, se o comando puder ser repetido em uma exceção de retorno de transação.

Um exemplo de um comando que não deve ser repetido no caso de uma exceção de transação é o comando OrderProcess. Esse comando invoca o processo de autorização de pagamento de terceiros. Ele não pode ser repetido, já que a autorização não pode ser revertida. Um exemplo de um comando que pode ser repetido é o comando ProductDisplay.

A sintaxe para permitir que o comando seja repetido no caso de uma exceção de retorno de transação é a seguinte:

```
public boolean isRetriable()
{
     return true;
}
```

Método setRequestProperties

O método setRequestProperties é invocado pelo controlador da Web para passar todas as propriedades de entrada para o comando do controlador. O comando do controlador deve analisar as propriedades de entrada e definir explicitamente cada propriedade individual dentro desse método. Essa definição explícita de propriedades pelo comando do controlador por si mesmo promove o conceito de tipos de propriedades seguras.

A sintaxe para utilizar este método é a seguinte:

```
public void setRequestProperties(
   com.ibm.commerce.datatype.TypedProperty reqParms)
{

     // parse the input properties and explicitly set each parameter

}
```

Método validateParameters

O método ValidateParameters é utilizado para executar a verificação do parâmetro inicial e qualquer resolução necessária de parâmetros. Por exemplo, ele pode ser utilizado para resolver orderId=*. Esse método é chamado antes dos métodos getResources e performExecute. Consulte “Interações do Controle de Acesso” na página 105 para obter mais detalhes sobre essa seqüência.

Método getResources

Esse método é utilizado para implementar controle de acesso no nível de recurso. Ele retorna um vetor de pares de recurso-ação sobre os quais o comando pretende agir. Se nada for retornado, nenhum controle de acesso no nível de recurso será executado. Para obter mais informações sobre controle de acesso, consulte Capítulo 4, “Controle de Acesso” na página 91.

Método performExecute

O método performExecute contém a lógica de negócios do seu comando. Ele deve invocar o método performExecute da superclasse do comando antes que qualquer nova lógica de negócios seja executada. No final, deve retornar um nome de exibição.

A seguir, um exemplo de sintaxe para o método performExecute em um novo comando do controlador. Nesse caso, a resposta utiliza um comando de redirecionamento de exibição, ele pode, no entanto, utilizar um comando de encaminhamento de exibição ou o comando de direcionamento de exibição:

```
public void performExecute() throws ECException
{
     super.performExecute();

//////////////////////////////////////////////
//  your business logic                     //
//////////////////////////////////////////////

// Create a new TypedProperty for response properties.
TypedProperty rspProp = new TypedProperty();

// set response properties
rspProp.put(ECConstants.EC_VIEWTASKNAME, "MyView");
//////////////////////////////////////////////////////
//  The following line is optional.  The VIEWREG   //
//  table can specify the redirect URL.            //
//////////////////////////////////////////////////////

rspProp.put(ECConstants.EC_REDIRECTURL, MyURL);

//////////////////////////////////////////////////////////////////////////
//   If you are using a forward view, you can set the                   //
//   response properties as follows:                                    //
//   TypedProperty rspProp = new TypedProperty();                       //
//   rspProp.put(ECConstants.EC_VIEWTASKNAME, "MyView");                //
//   rspProp.put(ECConstants.EC_DOCPATHNAME, "MyJSP.jsp");              //
//                                                                      //
//   Again, it is optional to explicitly set the name of the JSP template.//
//   The VIEWREG table can specify the JSP template.                    //
//////////////////////////////////////////////////////////////////////////

setResponseProperties(rspProp);
}
```

Se você especificar a URL de redirecionamento dentro do método performExecute e existir uma entrada na tabela VIEWREG, o valor especificado no código precederá o valor na tabela VIEWREG. O mesmo pedido de precedência mantém true para a especificação de um modelo JSP dentro do código.

Comandos do Controlador de Longa Execução

Se um comando do controlador levar muito tempo para ser executado, você poderá dividi-lo em dois. O primeiro comando, que é executado como o resultado de um pedido da URL, simplesmente inclui o segundo comando no Programador para que ele seja executado como um job em segundo plano. Isso está ilustrado no seguinte diagrama:

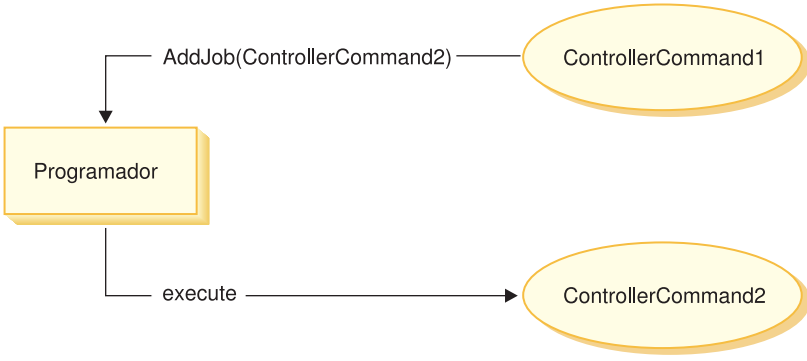


Figura 28.

O fluxo mostrado no diagrama anterior é o seguinte:

1. O ControllerCommand1 é executado como resultado de um pedido de URL
2. O ControllerCommand1 inclui um job no Programador. O job é ControllerCommand2. O ControllerCommand1 retorna uma exibição, imediatamente após incluir o job no Programador.
3. O Programador executa ControllerCommand2 como um job em segundo plano.

Nesse cenário, o cliente geralmente recebe o resultado do ControllerCommand2. O ControllerCommand2 deve gravar o estado do job no banco de dados.

Formatando as Propriedades de Entrada para Exibir Comandos

Quando um comando de controlador é concluído, ele retorna o nome de uma exibição que deve ser executada. Essa exibição pode requerer que várias propriedades de entrada sejam passadas para ela. Pode haver três fontes para esses parâmetros de entrada, conforme descrito na lista a seguir:

- Propriedades padrão armazenadas na coluna PROPERTIES da tabela CMDREG.
- Propriedades padrão da coluna PROPERTIES da tabela VIEWREG.
- Propriedades de entrada da URL.

Para obter mais informações sobre como essas propriedades são mescladas e definidas nos atributos do modelo JSP, consulte “Definindo Atributos de JSP - Visão Geral” na página 45. Essa seção descreve como as propriedades de entrada para um comando de exibição podem ser formatadas.

Para redirecionar comandos de exibição, dois tópicos são examinados:

- Condensação de uma cadeia de consulta para suportar redirecionamento da URL.
- Tratamento da extensão da URL redirecionada com um limite.

Para comandos de exibição de encaminhamento, o tópico de enumeração de parâmetros de entrada e sua definição como atributos no HttpServletRequestObject são examinados.

Condensando Parâmetros de Entrada em uma Cadeia de Consulta para HttpRedirectView

Todos os parâmetros de entrada passados para um comando de exibição de redirecionamento são condensados em uma cadeia de consulta para redirecionamento da URL. Por exemplo, suponha que a entrada do comando de exibição de redirecionamento contenha as seguintes propriedades:

```
URL = "MyView?p1=v1&p2=v2";
ip1 = "iv1";   // input to orginal controller command
ip2 = "iv2" ;  // input to original controller command
op1 = "ov1";
op2 = "ov2";
```

Com base nos parâmetros de entrada anteriores, a URL final é

```
MyView?p1=v1&p2=v2&ip1=iv1&ip2=iv2&op1=ov1&op2=ov2
```

Observe que se o comando deve utilizar SSL, os parâmetros serão criptografados e a URL final aparecerá como:

```
MyView?krypto=encrypted_value_of"p1=v1&p2=v2&ip1=iv1&ip2=iv2&op1=ov1&op2=ov2"
```

Tratando uma URL Redirecionada de Comprimento Limitado

Por padrão, todos os parâmetros de entrada para o comando de controlador são propagadas para o comando de exibição de redirecionamento. Se houver um limite para o número de caracteres na URL redirecionada, poderá ocorrer um problema. Um exemplo de quando o comprimento pode ser limitado é se o cliente estiver utilizando o navegador Internet Explorer. Nesse navegador, a URL não pode exceder 2.083 bytes. Se a URL exceder esse limite, ela será truncada. Dessa forma, é possível que se encontre um problema se houver um grande número de parâmetros de entrada ou se a criptografia estiver sendo utilizada, pois uma cadeia criptografada normalmente tem duas ou três vezes o tamanho de uma cadeia não-criptografada.

Há duas abordagens para tratar uma URL redirecionada de comprimento limitado:

1. Substituir o método getViewInputProperties no comando do controlador para retornar apenas os conjuntos de parâmetros necessários para passar para o comando de exibição de redirecionamento.
2. Utilizar um caractere especial nos parâmetros da URL para indicar quais parâmetros podem ser removidos da cadeia de parâmetros de entrada.

Para demonstrar cada uma das abordagens anteriores, considere o seguinte conjunto de parâmetros de entrada para o comando de controlador:

```
URL="MyView";
// All of the following are inputs to the original controller command.
ip1="ipv1";
ip2="ipv2";
ip3="ipv3";
iq1="iqv1";
iq2="iqv2";
ir1="ipr1";
ir2="ipr2";
is="isv";
```

Se estiver substituindo o método getViewInputProperties, o novo método pode ser gravado de forma que apenas os parâmetros a seguir sejam passados para o comando de exibição:

```
ir2="ipr2";
is="isv";
```

Utilizando a segunda abordagem, o comando de exibição pode ser chamado utilizando parâmetros especiais para indicar que determinados parâmetros de entrada devem ser removidos. Por exemplo, é possível obter o mesmo resultado especificando o seguinte como o parâmetro da URL:

```
URL="MyView?ip*=&iq*=&ir1="
```

Esse parâmetro da URL instrui o seguinte para a estrutura de runtime do WebSphere Commerce:

- A especificação `ip*=` indica que todos os parâmetros cujos nomes começam com `ip` devem ser removidos.
- A especificação `iq*=` indica que todos os parâmetros cujos nomes começam com `iq` devem ser removidos.
- A especificação `ir1=` indica que o parâmetro `ir1` deve ser removido.

Definindo Atributos no Objeto HttpServletRequest para HttpForwardView

O padrão HttpForwardViewCommandImpl enumera todos os parâmetros passados para o comando e os define como atributos no objeto HttpServletRequest.

Por exemplo, suponha que o objeto requestProperties passado para o comando de exibição de encaminhamento contenha os seguintes parâmetros:

```
p1="pv1";
p2="pv2";
p3=pv3;   // pv3 is an object
```

Em seguida, os seguintes atributos são passados para o modelo JSP que utiliza o método request.setAttribute().

```
request.setAttribute("p1", "pv1");
request.setAttribute("p2", "pv2");
request.setAttribute("p1", pv1);
request.setAttribute("RequestProperties", requestProperties);
request.setAttribute("CommandContext", commandContext);
```

em que `requestProperties` é o objeto TypedProperty que é passado para o comando, `commandContext` é o objeto de contexto do comando que é passado para o comando e `p1`, `p2` e `p3` são parâmetros definidos no objeto requestProperties.

Consolidações e Retrocessos para Comandos de Controlador

Durante a execução de um comando de controlador, dados são freqüentemente criados ou atualizados. Em muitos casos, o banco de dados precisa ser atualizado com as novas informações no final da transação. A transação é gerenciada pelo controlador da Web.

O controlador da Web marca o início da transação antes de chamar o comando de controlador. Quando a execução do comando de controlador estiver concluída, o comando de controlador retorna um nome de exibição ao controlador da Web. O controlador da Web é responsável por marcar o final da transação. O ponto real no qual a transação termina (antes ou depois de chamar a exibição) depende do tipo de exibição utilizado.

Há três tipos de comandos de exibição:

- Comando de encaminhamento da exibição
- Comando de redirecionamento da exibição
- Comando de direcionamento da exibição

O controlador da Web determina o comando de exibição a ser utilizado para a exibição, procurando o nome da exibição na tabela VIEWREG.

Se a entrada da tabela VIEWREG especificar a utilização de ForwardViewCommand, então, o controlador da Web encaminhará os resultados do comando de controlador para a classe de implementação ForwardViewCommand correspondente (também especificada em VIEWREG).

O comando de exibição é executado no contexto da transação atual. Nesse caso, a consolidação ou o retrocesso do banco de dados não ocorre até o comando de exibição ser concluído.

Se a entrada da tabela VIEWREG especificar a utilização de RedirectViewCommand, então, o controlador da Web encaminhará os resultados do comando de controlador para a classe de implementação RedirectViewCommand correspondente. O comando de exibição, então, opera fora do escopo da transação atual e a consolidação ou o retrocesso do banco de dados ocorre antes do comando de exibição redirecionado ser chamado.

Se a entrada da tabela VIEWREG especificar a utilização de DirectViewCommand, então, o controlador da Web encaminhará os resultados do comando de controlador para a classe de implementação DirectViewCommand correspondente. O comando de exibição é executado no contexto da transação atual. Nesse caso, a consolidação ou o retrocesso do banco de dados não ocorre até o comando de exibição ser concluído. (Observe que ForwardViewCommand e DirectViewCommand são similares. ForwardViewCommand encaminha os resultados a um modelo JSP. DirectViewCommand, por sua vez, recebe os resultados como fluxo de entrada e os transmite adiante como um fluxo de saída. Ele utiliza o método getRawDocument que trata os dados como bytes ou getTextDocument que trata os dados como texto).

Nos casos em que o comando de exibição é executado no escopo da mesma transação que o comando de controlador, um erro no comando de exibição causa um retrocesso de toda a transação. Esse pode ser ou não o resultado desejado, dependendo de sua lógica de negócios.

Exemplo de Escopo de Transação com o Comando de Controlador

Para ilustrar as diferenças no escopo da transação para um comando de controlador, dependendo do tipo do comando de exibição utilizado, considere os exemplos a seguir.

Caso 1: Executando a Exibição no Escopo da Transação do Comando de Controlador

Suponhamos que você tenha criado um novo comando de controlador chamado `YourControllerCmdA`. O método performExecute do comando, então, incluiria o seguinte:

```
.
.
// Create a new TypedProperty object for output.
TypedProperty rspProp = new TypedProperty();

//////////////////////
// Business logic   //
//////////////////////
```

```
// Return the view
rspProp.put(ECConstants.EC_VIEWTASKNAME, "YourView");
SetResponseProperties(rspProp);
```

No fragmento de código acima, o comando de controlador retorna "YourView" como a exibição. YourView está registrada na tabela VIEWREG. Segue um exemplo de instrução de inserção para registrar YourView.

```
insert into VIEWREG (ViewName, DeviceFmt_id, storeEnt_id, interfacename,
classname, properties)

values ('YourView',  -1, XX,'com.ibm.commerce.command.ForwardViewCommand',
'com.ibm.commerce.command.HttpForwardViewCommandImpl','docname=YourView.jsp');
```

em que XX é o identificador da loja. Como a exibição utiliza a classe de implementação `com.ibm.commerce.command.HttpForwardViewCommandImpl`, o controlador da Web utiliza o comando de encaminhamento de exibição genérico.

Com base no registro do comando acima, o controlador da Web lança o arquivo `YourView.jsp` no escopo da transação do comando de controlador. Se ocorrer um erro em `YourView.jsp`, a transação falha e ocorre um retrocesso do banco de dados. Como resultado, todo o comando de controlador falha.

Caso 2: Executando a Exibição Fora do Escopo da Transação do Comando de Controlador

Suponhamos que você prefira ter informações consolidadas no banco de dados, mesmo quando um erro possa ocorrer na exibição. Para que a exibição seja executada fora do escopo da transação do comando de controlador, a exibição precisa ser executada como um redirecionamento.

Para executar a exibição como um redirecionamento, o método performExecute do comando de controlador retorna a exibição da seguinte maneira:

```
.
.
// Create a new TypedProperty object for output.
TypedProperty rspProp = new TypedProperty();

//////////////////////
// Business logic   //
//////////////////////

// Return the view
rspProp.put(ECConstants.EC_VIEWTASKNAME, EC_GENERIC_REDIRECTVIEW);
rspProp.put(EC_Constants.EC_REDIRECTURL, "YourView2");
```

O exemplo a seguir da instrução SQL suporta a estratégia de redirecionamento:

```
insert into VIEWREG (ViewName, DeviceFmt_id, storeEnt_id, interfacename,
classname, properties)

values ('YourView2',  -1, XX,'com.ibm.commerce.command.ForwardViewCommand',
'com.ibm.commerce.command.HttpForwardViewCommandImpl','docname=YourView2.jsp');
```

em que XX é o identificador da loja.

Como o comando transmite o valor EC_GENERIC_REDIRECTVIEW como um parâmetro de propriedade da resposta, o controlador da Web utiliza o comando de exibição de redirecionamento genérico. A exibição de redirecionamento genérico está registrada na tabela VIEWREG com as seguintes informações:

- ViewName = RedirectView
- DeviceFmt_Id = -1
- InterfaceName = com.ibm.commerce.command.RedirectViewCommand
- ClassName = com.ibm.commerce.command.HttpRedirectViewCommandImpl

O controlador da Web chama o comando de exibição de redirecionamento genérico, que utiliza a URL de redirecionamento como uma propriedade de entrada. A resposta é redirecionada para a URL de redirecionamento. Após o redirecionamento ocorrer, YourView2 será chamada. Isso é implementado como um encaminhamento de exibição genérico.

Novos Comandos de Tarefas

Um novo comando de tarefas deve se estender da classe de comando de tarefas abstrata (com.ibm.commerce.command.TaskCommandImpl) e implementar uma interface que estenda a interface com.ibm.commerce.TaskCommand. Conforme mostrado no diagrama da página 131, o novo comando de tarefa deve ser definido da seguinte forma:

```
public class MyTaskCmdImpl extends com.ibm.commerce.command.TaskCommandImpl
     implements MyTaskCmd {

}
```

Todas as propriedades de entrada e saída para o comando de tarefas devem ser definidas na interface do comando, por exemplo MyTaskCmd. O originador da chamada programa para a interface de comando de tarefas, em vez de para a classe de implementação do comando de tarefas. Isso permite que você tenha várias implementações do comando de tarefas (uma para cada loja), sem que o originador da chamada se preocupe com qual classe de implementação chamar.

Todos os métodos definidos na interface devem ser implementados na classe de implementação. Visto que o contexto do comando deve ser definido pelo originador da chamada (um comando do controlador), o comando de tarefas não precisa definir o contexto do comando. O comando de tarefas pode, entretanto, obter informações do controlador da Web utilizando o contexto do comando.

Além de implementar os métodos definidos na interface do comando de tarefa, você deve substituir o método performExecute na classe com.ibm.commerce.command.TaskCommandImpl.

O método performExecute contém a lógica de negócios para uma determinada unidade de trabalho que o comando de tarefas executa. Ele deve chamar o método performExecute da superclasse do comando de tarefas antes de executar qualquer lógica de negócios. O fragmento de código a seguir mostra um exemplo do método performExecute para um comando de tarefas.

```
public void performExecute() throws ECException
{
     super.performExecute();

     // Include your business logic here.

     // Set output properties so the controller command
     // can retrieve the result from this task command.
}
```

A estrutura de runtime chama o método getResources do comando de controlador para determinar quais recursos que podem ser protegidos poderão ser acessados pelo comando. Pode ser o caso de que um comando de tarefa seja executado durante o escopo de um comando de controlador e tente acessar recursos que não foram retornados pelo método getResources do comando do controlador. Nesse caso, o comando de tarefa pode implementar um método getResources para assegurar que o controle de acesso seja fornecido para recursos protegidos.

Observe que, por padrão, getResources retorna `null` para um comando de tarefa e a verificação do controle de acesso no nível de recurso não é executada. Portanto, é necessário substituir isso se o comando de tarefa acessar recursos que podem ser protegidos.

Personalizando os Comandos já Existentes

Essa seção descreve as diversas maneiras na qual você pode personalizar os comandos do controlador, de tarefas e de bean de dados já existentes.

Personalizando Comandos Existentes do Controlador

Um comando do controlador encapsula a lógica de negócios para um processo de negócios. As unidades de trabalho individuais dentro do processo de negócios podem ser executadas pelos comandos de tarefas. Como tal, há várias maneiras na qual um comando do controlador pode ser personalizado, alguns dos quais envolve personalização dos comandos de tarefas.

Ao personalizar um comando de controlador, você pode fazer o seguinte:

- Incluir processamento e lógica adicionais em um comando de controlador existente. Isso pode ser incluído antes da lógica de negócios existente, depois da lógica de negócios existente ou antes e depois.
- Substitua um ou mais comandos de tarefas. Isso permite modificar como uma determinada etapa no processo de negócios será executada.
- Substituir a exibição chamada pelo comando de controlador.

As seções a seguir fornecem detalhes sobre como fazer as modificações acima.

Incluindo Nova Lógica de Negócios em um Comando de Controlador

Suponha que exista um comando de controlador do WebSphere Commerce existente chamado `ExistingControllerCmd`. Seguindo as convenções de nomenclatura do WebSphere Commerce, esse comando de controlador teria uma classe de interface denominada `ExistingControllerCmd` e uma classe de implementação denominada `ExistingControllerCmdImpl`. Agora suponha que surja uma necessidade de negócios e você precise incluir nova lógica de negócios nesse comando existente. Uma parte da lógica deve ser executada antes da lógica existente do comando e outra parte deve ser executada depois da lógica existente do comando.

A primeira etapa para incluir a nova lógica de negócios é criar uma nova classe de implementação que estenda a classe de implementação original. Nesse exemplo, você criaria uma nova classe `ModifiedControllerCmdImpl` que estenderia a classe `ExistingControllerCmdImpl`. A nova classe de implementação deveria implementar a interface original (`ExistingControllerCmd`).

Na nova classe de implementação você deve criar um novo método `performExecute` para substituir o `performExecute` do comando existente. No novo método `performExecute`, há duas maneiras de inserir a nova lógica de negócios: você pode incluir o código diretamente no comando de controlador ou criar um novo comando de tarefa para executar a nova lógica de negócios. Se um novo comando de tarefa for criado, você deve instanciar o novo objeto do comando de tarefa a partir do comando de controlador.

O fragmento de código a seguir demonstra como incluir nova lógica de negócios no início e no fim de um comando de controlador existente, incluindo a lógica diretamente no comando de controlador:

```
public class ModifiedControllerCmdImpl extends ExistingControllerCmdImpl
   implements ExistingControllerCmd
   {
      public void performExecute ()
         throws com.ibm.commerce.exception.ECException
         {

         /* Insert new business logic that must be
            executed before the original command.
            */

         // Execute the original command logic.
     super.performExecute();

         /* Insert new business logic that must be
            executed after the original command.
            */
          }
    }
```

O fragmento de código a seguir demonstra como incluir nova lógica de negócios no início de um comando de controlador, instanciando um novo comando de tarefa a partir do comando de controlador. Além disso, você também criaria a interface do novo comando de tarefa e classe de implementação, e registraria o comando de tarefa no registro de comandos.

```
// Import the package with the CommandFactory
import com.ibm.commerce.command.*;

public class ModifiedControllerCmdImpl extends ExistingControllerCmdImpl
   implements ExistingControllerCmd
   {

      public void performExecute ()
         throws com.ibm.commerce.exception.ECException
         {
     MyNewTaskCmd cmd = null;
          cmd = (MyNewTaskCmd) CommandFactory.createCommand(
               "com.mycompany.mycommands.MyNewTaskCommand",
               getStoreId());

            /*
         Set task command's input parameters, call its
         execute method and retrieve output
         parameters, as required.
            */

     super.performExecute();
         }
   }
```

Independentemente da inclusão de nova lógica de negócios no comando de controlador ou criação de um comando de tarefa para executar a lógica, é necessário também atualizar a tabela CMDREG no registro de comandos do

WebSphere Commerce para associar a nova classe de implementação do comando de controlador à interface do comando de controlador existente. A instrução SQL a seguir mostra um exemplo de atualização:

```
update CMDREG
set CLASSNAME='ModifiedControllerCmdImpl'
where INTERFACENAME='ExistingControllerCmd'
```

Substituindo Comandos de Tarefas Chamados por um Comando de Controlador

Um comando de controlador freqüentemente chama vários comandos de tarefas que executam tarefas individuais. De forma coletiva, essas tarefas formam o processo de negócios representado pelo comando de controlador. Pode ser necessário alterar a maneira como uma etapa específica do processo é executada, em vez de incluir nova lógica de negócios no início ou fim do comando de controlador. Nesse caso, é necessário substituir a criação de instância do comando de tarefa que deseja substituir pela criação de instância de um novo comando de tarefa que execute a tarefa da maneira desejada.

Como resultado do design do modelo de programação do WebSphere Commerce, você não precisa criar uma nova classe de implementação do comando de controlador para substituir o comando de tarefa. O comando de controlador cria uma instância do comando de tarefa, chamando o método `createCommand` da fábrica de comandos. A fábrica de comandos utiliza o nome da interface do comando de tarefa e em seguida, determina a classe de implementação correta com base no registro de comandos. Dessa forma, para substituir o comando de tarefa cuja instância é criada, é necessário criar uma nova classe de implementação do comando de tarefa e em seguida, atualizar o registro de comandos de forma que o nome da interface do comando de tarefa original seja associado à nova classe de implementação do comando de tarefa. Consulte “Personalizando Comandos de Tarefas Existentes” na página 150 para obter mais informações.

Substituindo a Exibição Chamada por um Comando de Controlador

Para substituir a exibição chamada por um comando de controlador, cria-se uma nova classe de implementação para o comando de controlador. Por exemplo, crie uma nova `ModifiedControllerCmdImpl` que estenda a `ExistingControllerCmdImpl` e implemente a interface `ExistingControllerCmd`.

Na classe `ModifiedControllerCmdImpl`, substitua o método `performExecute`. No novo método `performExecute`, chame `super.performExecute` para garantir que todo o processamento do comando ocorra. Depois que a lógica do comando é executada, você pode utilizar as propriedades de resposta para substituir a exibição chamada. O seguinte fragmento de código mostra como substituir a exibição quando ela é executada como um redirecionamento:

```
// Import the packages containing TypedProperty, and ECConstants.
import com.ibm.commerce.datatype.*;
import com.ibm.commerce.server.*;
```

```
public class ModifiedControllerCmdImplImpl extends ExistingControllerCmdImpl
   implements ExistingControllerCmd
   {
      public void performExecute ()
         throws com.ibm.commerce.exception.ECException
         {

         // Execute the original command logic.
     super.performExecute();

// Create a new TypedProperty for response properties.
           TypedProperty rspProp = new TypedProperty();

// set response properties
        rspProp.put(ECConstants.EC_VIEWTASKNAME, "MyView");
          ////////////////////////////////////////////////
//  The following line is optional.  The VIEWREG   //
//  table can specify the redirect URL.            //
         ////////////////////////////////////////////////

rspProp.put(ECConstants.EC_REDIRECTURL, MyURL);


 setResponseProperties(rspProp);

         }
   }
```

O seguinte fragmento de código mostra como substituir a exibição quando ela é executada como uma exibição de encaminhamento:

```
// Import the packages containing TypedProperty, and ECConstants.
import com.ibm.commerce.datatype.*;
import com.ibm.commerce.server.*;

public class ModifiedControllerCmdImplImpl extends ExistingControllerCmdImpl
   implements ExistingControllerCmd
   {
      public void performExecute ()
         throws com.ibm.commerce.exception.ECException
         {

         // Execute the original command logic.
     super.performExecute();

// Create a new TypedProperty for response properties.
           TypedProperty rspProp = new TypedProperty();

// set response properties
        rspProp.put(ECConstants.EC_VIEWTASKNAME, "MyView");

          ////////////////////////////////////////////////
         //  It is optional to explicitly set the name   //
         //  of the JSP template. The VIEWREG table can  //
```

```
            //  specify the JSP template.                   //
            ////////////////////////////////////////////////////

          rspProp.put(ECConstants.EC_DOCPATHNAME, "MyJSP.jsp");

    setResponseProperties(rspProp);

            }
        }
```

Para determinar qual exibição será utilizada por um comando de controlador existente, consulte a seção de Referência da ajuda online do WebSphere Commerce.

Personalizando Comandos de Tarefas Existentes

Há duas maneiras padrão de modificar comandos de tarefa existentes do WebSphere Commerce. Com esses métodos de modificação, você pode fazer o seguinte:

- Incluir processamento e lógica adicionais em um comando de controlador existente. Isso pode ser incluído antes da lógica de negócios existente, depois da lógica de negócios existente ou antes e depois.
- Substituir completamente a lógica de negócios existente por sua lógica de negócios.

Para realizar as modificações acima, você realmente cria uma nova classe de implementação do comando de tarefa. Mais detalhes são fornecidos nas seções a seguir.

Incluindo Nova Lógica de Negócios em um Comando de Tarefa

Suponha que exista um comando de tarefa do WebSphere Commerce existente chamado `ExistingTaskCmd`. Seguindo as convenções de nomenclatura do WebSphere Commerce, esse comando de tarefa teria uma classe de interface denominada `ExistingTaskCmd` e uma classe de implementação denominada `ExistingTaskCmdImpl`. Agora suponha que surja uma necessidade de negócios e você precise incluir nova lógica de negócios nesse comando existente. Uma parte da lógica deve ser executada antes da lógica existente do comando e outra parte deve ser executada depois da lógica existente do comando.

A primeira etapa para incluir a nova lógica de negócios é criar uma nova classe de implementação que estenda a classe de implementação original. Nesse exemplo, você criaria uma nova classe `ModifiedTaskCmdImpl` que estenderia a classe `ExistingTaskCmdImpl`. A nova classe de implementação deveria implementar a interface original (`ExistingTaskCmd`).

No novo comando, você substitui o método `performExecute` existente e inclui uma nova lógica antes e depois de chamar o método `super.performExecute`.

O pseudo-código a seguir demonstra como incluir a nova lógica de negócios em um comando de tarefa existente:

```
public class ModifiedTaskCmdImpl extends ExistingTaskCmdImpl
     implements ExistingTaskCmd {


         /* Insert new business logic that must be
            executed before the original command.
         */

         // Execute the original command logic.
     super.performExecute();

         /* Insert new business logic that must be
            executed after the original command.
         */
}
```

É necessário também atualizar a tabela CMDREG para associar a nova classe de implementação à interface existente. A instrução SQL a seguir mostra um exemplo de atualização:

```
update CMDREG
set CLASSNAME='ModifiedTaskCmdImpl'
where INTERFACENAME='ExistingTaskCmd'
```

Substituindo Lógica de Negócios de um Comando de Tarefa Existente

Para substituir a lógica de negócios de um comando de tarefa existente, você deve criar uma nova classe de implementação para o comando de tarefa. Essa nova classe de implementação deve estender-se do comando de tarefa existente, mas não deve implementar a interface existente. Adicionalmente, na nova classe de implementação, não chame o método `performExecute` da superclasse.

Embora a extensão do comando exato que você está substituindo possa parecer contra-intuitiva, a razão para executar essa abordagem está relacionada ao suporte para versões futuras do WebSphere Commerce. Essa abordagem protege seu código de alterações que possam ser feitas para interfaces de comando em versões futuras do WebSphere Commerce.

Como um exemplo, suponha que você queira substituir a lógica de negócios do comando de tarefa `OrderNotifyCmdImpl`. Nesse caso, você criará um novo comando de tarefa chamado `CustomizedOrderNotifyCmdImpl`. Esse comando estende o `OrderNotifyCmdImpl`. No novo `CustomizedOrderNotifyCmdImpl` , você cria a nova lógica de negócios, mas não chama o método `performExecute` a partir da superclasse. Se uma versão futura do WebSphere Commerce introduzir um novo método chamado`newMethod` na interface, a versão correspondente do comando `OrderNotifyCmdImpl` incluirá uma implementação padrão do método`newMethod`. Então, como o novo comando se estende a partir

de OrderNotifyCmdImpl, o compilador localizará a implementação padrão desse novo método no comando OrderNotifyCmdImpl e o novo comando será protegido da alteração da interface.

Consulte a seção Referência da ajuda online do WebSphere Commerce para assegurar que a nova classe de implementação forneça o mesmo comportamento externo que a classe existente.

Personalização do Bean de Dados

Um bean de dados normalmente estende um bean de acesso. O bean de acesso, que pode ser gerado pelo VisualAge for Java, fornece uma única maneira de acessar as informações a partir de um bean de entidade. Quando as modificações são feitas em um bean de entidade (por exemplo, incluindo um novo campo, um novo método de negócios ou um novo localizador), a atualização é refletida no bean de acesso assim que o beans de acesso for gerado novamente. Já que o bean de dados estende o bean de acesso, ele automaticamente herda os novos atributos. Como resultado desse relacionamento, nenhuma codificação é exigida para permitir que o bean de dados utilize novos atributos a partir do bean de entidade.

Se for necessário incluir novos atributos em um bean de dados que não sejam derivados de um bean de entidade, você poderá estender o bean de dados existente utilizando a herança do Java. Por exemplo, para incluir um novo campo no OrderDataBean, defina MyOrderDataBean como a seguir:

```
public class MyOrderDataBean extends OrderDataBean
{
     public String myNewField () {
     //  implement the new field here
     }

}
```

Os novos dados também devem ter uma classe BeanInfo. A seguir, uma amostra da declaração para essa classe:

```
public class MyOrderDataBeanInfo extends java.beans.SimpleBeanInfo
{

}
```

O VisualAge for Java fornece uma ferramenta que permite gerar essa classe BeanInfo.

Capítulo 7. Acordos Comerciais e Políticas de Negócios (Business Edition)

Este capítulo aplica-se apenas ao WebSphere Commerce Business Edition.

Introdução

Um dos elementos chave do comércio B2B (business-to-business) é o gerenciamento de relacionamentos. Um acordo de comercialização é utilizado para gerenciar um relacionamento de negócios entre um comprador e uma organização vendedora. O modelo de acordo de comercialização utilizado pelo WebSphere Commerce Business Edition suporta diversos tipos de acordos de comercialização, como Contrato e RFQ (request for quote).

O elemento principal de um acordo de comercialização é um conjunto de termos e condições. Cada termo e condição define uma regra específica de negócios utilizada durante a comercialização. Utilizando o WebSphere Commerce Business Edition, um conjunto de termos e condições pode ser negociado utilizando um processo de RFQ online ou negociado offline e em seguida, capturado utilizando as interfaces de gerenciamento de relacionamentos de negócios no WebSphere Commerce Accelerator.

Há várias maneiras de modelar um termo e condição:

- Um termo e condição que seleciona uma das políticas de negócios predefinidas, como uma lista de Preços e uma política de Devolução. Ou pode selecionar uma política de negócios que você tenha criado. Um objeto de termo e condição também podem referir-se a vários objetos de política de negócios.
- Um termo e condição que se aplica a um ajuste específico na política de negócios, como um ajuste nos preços padrão.
- Um termo e condição que define um conjunto de parâmetros que governam um processo do negócio. Por exemplo, poderia especificar que um centro de distribuição específico deve ser utilizado por um contrato específico.

Um contrato é composto por um conjunto de termos e condições. Isso é mostrado no seguinte diagrama.

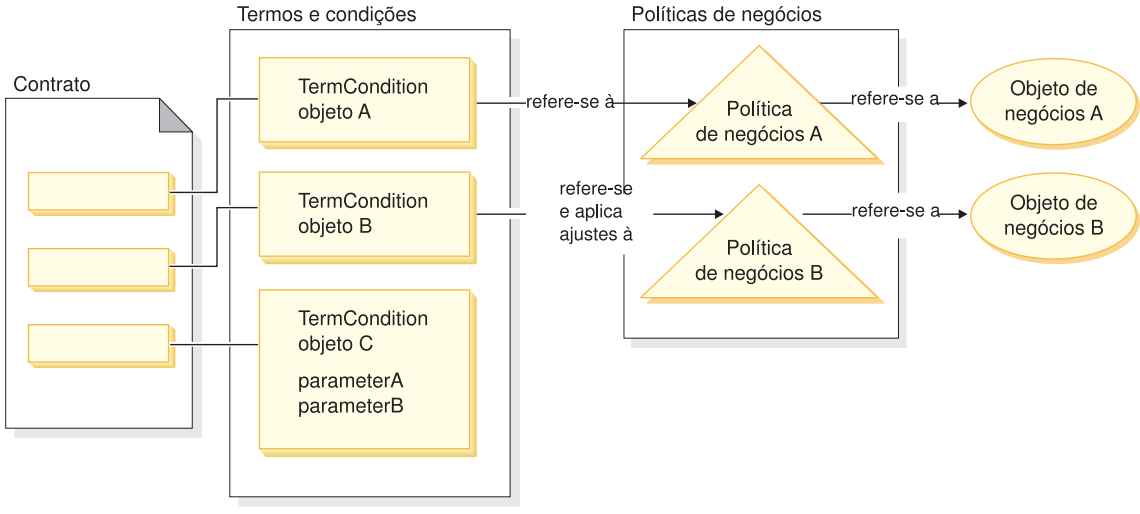


Figura 29.

A partir do diagrama anterior, anote o seguinte:

- O termo "ajuste" refere-se a uma modificação à política de negócios. Como exemplo, ele pode ser utilizado para aplicar um desconto ao resultado de uma política de negócios, como um desconto de 10% aplicado ao preço padrão. Também pode ser utilizado para influenciar a política de negócios com um conjunto de parâmetros.
- Como um exemplo, o objeto A de TermCondition pode representar um objeto de termo e condição de envio. Nesse caso, a política de negócios A pode representar uma política de negócios do modo de envio e o objeto de negócios A representa o modo de envio "A3" da transportadora XYZ.
- Um outro exemplo, o objeto B de TermCondition pode representar um objeto de termo e condição de preço que aplica um desconto de 50% do preço definido pela política de negócios B. Nesse caso, a política de negócios B é uma política de preços e o objeto de negócios B é um contêiner da posição de negócios que define a posição de negócios para a categoria mestre.

Este capítulo fornece diretrizes para programadores sobre como criar novas políticas de negócios e novos termos e condições.

A loja de exemplo ToolTech demonstra um objeto de termo e condição de envio e um objeto de termo e condição de preço em seu fluxo de negócios. Para obter mais informações sobre os dados do contrato que suportam esses exemplos, consulte "Dados do Contrato de Exemplo da ToolTech" na página 156.

Objetos e Comandos de Política de Negócios

Um objeto de política de negócios contém as seguintes informações:

- ID da política
  Essa é a chave principal para o objeto política de negócios.
- Tipo de política
  Define o tipo de política de negócios. `Price` e `ProductSet` são exemplos de tipos de política.
- Nome da política
  Cada política de negócios precisa ter um nome exclusivo.
- Entidade da loja
  A loja ou o grupo de lojas no qual a política de negócios é implementada.
- Propriedades
  Um conjunto de propriedades padrão que pode ser passado para o comando da política de negócios. Os comandos associados ao objeto política de negócios são armazenados na tabela BusinessPolicyCmd.
- Período efetivo
  O período para o qual o objeto política de negócios está efetivo.
- Comando da política de negócios
  Zero ou mais comandos da política de negócios que implementam a política de negócios.Um comando de política de negócios é normalmente chamado por um processo de negócios que pode ser um comando de tarefa ou um comando do controlador. Por exemplo, o comando `getContractPrice()` obtém o termo e a condição de preço. Esse termo e condição de preços referem-se a um preço específico e esse comando de política de preços é utilizado para calcular o preço.

Vários comandos da política de negócios podem ser associados a um único objeto de política de negócios. Cada comando da política de negócios precisa implementar a mesma interface definida pelo objeto tipo de política de negócios. A estrutura de um novo comando de política de negócios é demonstrada no diagrama a seguir:

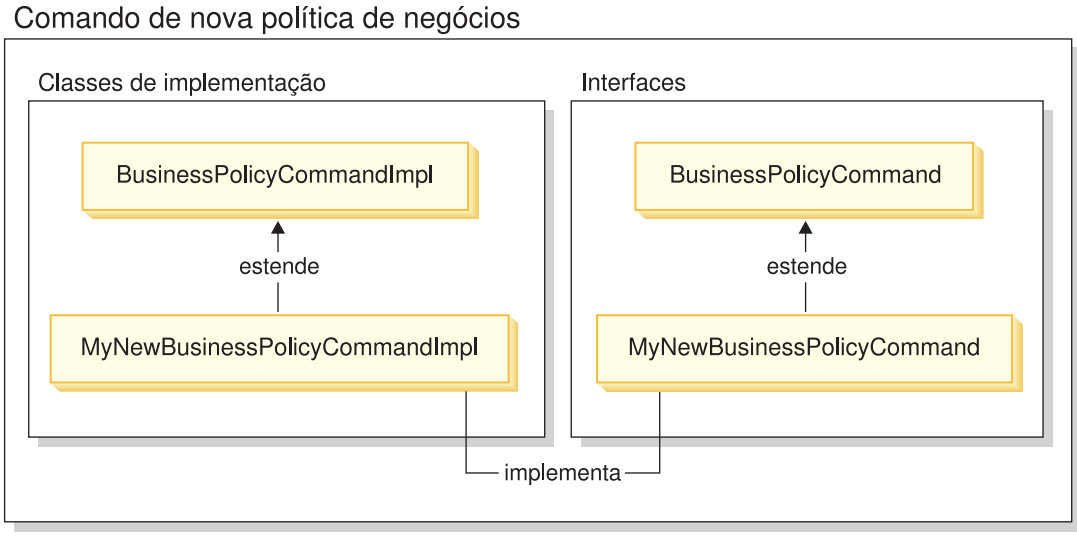


Figura 30.

Conforme mostrado no diagrama acima, para criar um novo comando de política de negócios, você cria uma nova classe de implementação que estende a classe de implementação BusinessPolicyCmdImpl do WebSphere Commerce. Você também cria uma nova interface que estende a interface BusinessPolicyCmd.

Dados do Contrato de Exemplo da ToolTech

Esta seção fornece uma introdução a alguns dados do contrato que são utilizados na loja de exemplo ToolTech.

Os dados de exemplo nas seguintes seções são organizados pela tabela de banco de dados. Somente as linhas e colunas pertinentes são exibidas. Observe também que quando a amostra é instalada, quaisquer identificadores exclusivos (como CONTRACT_ID) poderão ter valores diferentes do que os mostrados aqui.

Dados de Exemplo da Tabela CONTRACT

A seguinte tabela mostra dados de exemplo pertinentes da tabela de banco de dados CONTRACT da ToolTech. Observe que para fins de exibição, os títulos da coluna do banco de dados são mostrados na primeira coluna e a linha dos dados de exemplo da tabela é mostrada na segunda coluna.

Nome da coluna	Dados de exemplo
CONTRACT_ID	10007
MAJORVERSION	1

Nome da coluna	Dados de exemplo
MINORVERSION	0
NAME	ToolTechContractNumber 4567
MEMBER_ID	-2001
ORIGIN	0
STATE	3
USAGE	1
MARKFORDELETE	0

Dados de Exemplo da Tabela TERMCOND

A seguinte tabela mostra dados de exemplo pertinentes da tabela de banco de dados TERMCOND da ToolTech. Observe que, para fins de exibição, os cabeçalhos da coluna do banco de dados são mostrados na primeira coluna e as linhas de exemplo de dados são mostradas nas segunda e terceira colunas.

Nome da coluna	Linha de dados de exemplo 1	Linha de dados de exemplo 2
TERMCOND_ID	10025	10030
TCSUBTYPE_ID	PriceTCPriceListWith SelectiveAdjustment	ShippingTCShippingMode
TRADING_ID	10007	10007
STRINGFILED1	ProductSet2	
INTEGERFIELD2	10002	
INTEGERFIELD3	1	
BIGINTFIELD1	10051	
FLOATFIELD1	-50.0	
SEQUENCE	1	6

Dados de Exemplo da Tabela POLICYTC

A seguinte tabela mostra dados de exemplo pertinentes da tabela de banco de dados POLICYTC da ToolTech. Essa tabela estabelece o relacionamento entre uma política e um objeto de termos e condições.

	Nome da coluna	
	POLICY_ID	TERMCOND_ID
Linha de dados de exemplo 1	10053	10025

	Nome da coluna	
	POLICY_ID	**TERMCOND_ID**
Linha de dados de exemplo 2	10056	10030

Dados de Exemplo da Tabela POLICY

A seguinte tabela mostra dados de exemplo pertinentes da tabela de banco de dados POLICY da ToolTech.

Nome da coluna	**Linha de dados de exemplo 1**	**Linha de dados de exemplo 2**
POLICY_ID	10053	10056
POLICYNAME	MasterCatalogPriceList	A3
POLICYTYPE_ID	Price	ShippingMode
STOREENT_ID	10051	10051
PROPERTIES	name=ToolTech& member_id=-2001	shippingMode=A3
STARTTIME	nulo	nulo
ENDTIME	nulo	nulo

Dados de Exemplo da Tabela TRADEPOSCN

A seguinte tabela mostra dados de exemplo pertinentes da tabela de banco de dados TRADEPOSCN da ToolTech.

	Nome da coluna			
	READEPOSCN_ID	**MEMBER_ID**	**NAME**	**TYPE**
Linha de dados de exemplo	10051	-2001	ToolTech	S

Dados de Exemplo da Tabela SHIPMODE

A seguinte tabela mostra dados de exemplo pertinentes da tabela de banco de dados SHIPMODE da ToolTech.

	Nome da coluna			
	SHIPMODE_ID	**STOREENTITY_ID**	**CODE**	**CARRIER**
Linha de dados de exemplo	10053	10051	A3	Transpor-tadora XYZ

Estendendo o Modelo de Contrato Existente

Um contrato pode ser constituído de um ou mais objetos de termos e condições, cada um referindo-se a uma política. Como tal, as seções subseqüentes descrevem as etapas necessárias para criar uma nova política de negócios e integrá-la nesse fluxo de negócios.

Como uma breve visão geral, a seguir estão as etapas de alto nível para executar essa tarefa:

1. Criando uma nova política de negócios.
   As tarefas a seguir estão relacionadas para criar um comando de nova política de negócios:
   a. Crie um novo tipo de política de negócios (se necessário).
      São fornecidos vários tipos de políticas de negócios, mas se os tipos padrão não satisfizerem seus requisitos de negócios, crie um novo tipo de política de negócios.
   b. Criando um novo comando de política de negócios.
   c. Registre a nova política de negócios e o comando da política de negócios.
2. Relacione o objeto de termos e condições à nova política de negócios
   Isso pode ser feito relacionando-se um objeto de termos e condições existente à nova política de negócios ou criando-se um novo objeto de termos e condições. Se você criar um novo objeto de termos e condições, então deverá executar as seguintes etapas:
   a. Registrar o novo termo e condição no banco de dados.
   b. Registrar o novo termo e a condição no DTD do contrato (definição do tipo de documento)
   c. Criar um novo bean corporativo CMP para o termo e condição.
   d. Atualizar o WebSphere Commerce Accelerator para refletir o novo termo e condição.
3. Chamar a nova política de negócios durante o fluxo de negócios.

Criando uma Nova Política de Negócios

Criar uma nova política de negócios geralmente envolve o registro de uma política de negócios exclusiva no banco de dados, bem como criar um novo comando de política de negócios.

Criar um novo comando de política de negócios envolve as seguintes etapas de alto nível:

1. Criar um novo tipo de política de negócios (se necessário).
2. Escrever o novo comando de política de negócios.

3. Registrar a nova política de negócios e comando de política de negócios no banco de dados.

Cada uma das etapas acima é descrita mais detalhadamente nas seções a seguir.

Criando um Novo Tipo de Política de Negócios

Esta seção descreve como criar um novo tipo de política de negócios. Um tipo de política de negócios indica o domínio da transação à qual a política se aplica. Exemplos de tipos de políticas de negócios incluem:

- Price
- ProductSet
- ShippingMode
- ShippingCharge
- Payment
- ReturnCharge
- ReturnApproval
- ReturnPayment
- InvoiceFormat

Se os tipos de políticas de negócios existentes não satisfizerem suas necessidades de negócios, você deve criar um novo tipo de política de negócios. Criar o novo tipo de política de negócios consiste em definir e registrar o tipo de política de negócios.

Ao definir e registrar um novo tipo de política, é necessário atualizar as seguintes tabelas de bancos de dados:

- POLICYTYPE
- PLCYTYCMIF
- PLCYTYPDSC

A tabela POLICYTYPE especifica o tipo de política de negócios que você está criando. Ela contém uma única coluna, POLICYTYPE_ID, que é a chave principal. Um exemplo de valor é `Price`. Se você criar um novo tipo de política de negócios, certifique-se de especificar um POLICYTYPE_ID exclusivo.

A tabela PLCYTYCMIF é o tipo de política de negócios para a tabela de especificação de relacionamento de interface de comando. Ou seja, para cada tipo de política de negócios, ela especifica a interface do comando Java para o objeto da política de negócios. Embora possa haver zero ou mais comandos de política de negócios que implementam uma política de negócios, cada um dos comandos de política de negócios deve implementar a interface especificada aqui.

A tabela PLCYTYPDSC especifica uma descrição do tipo de política de negócios. Ela inclui um identificador de idioma da descrição e a descrição do tipo de política de negócios.

Para criar um novo tipo de política de negócios, crie uma entrada em cada uma dessas tabelas para o novo tipo de política de negócios. As instruções SQL a seguir fornecem um exemplo:

```
insert into POLICYTYPE (POLICYTYPE_ID) values ('MyNewPolicyType');
insert into PLCYTYCMIF (POLICYTYPE_ID, BUSINESSCMDIF)
   values ('MyNewPolicyType',
   'com.mycompany.mybusinesspolicycommands.MyNewPolicy');
insert into  PLCYTYPDSC (POLICYTYPE_ID, LANGUAGE_ID, DESCRIPTION)
   values ('MyNewPolicyType', -1,
   'My new policy type for example purposes.');
```

Como etapa final para criar o novo tipo de política de negócios, você pode codificar uma ou mais novas interfaces do tipo de política de negócios. Essas interfaces são implementadas por qualquer comando da política de negócios que recai sob o domínio deste tipo de política de negócios. Por exemplo, na loja de exemplo ToolTech, o Preço é definido como um tipo de política de negócios. Como tal, existem as interface `com.ibm.commerce.price.commands.ResolvePriceListsCmd` e `com.ibm.commerce.price.commands.RetrievePricesCmd` implementadas por todos os comandos da política de negócios relacionadas a preços.

Se você não tiver um comando de política de negócios que execute operações no novo tipo de política de negócios, então você não tem que criar uma nova interface. Isso é raro e, na maioria dos casos, ao criar um novo tipo de política de negócios, você deve criar uma nova interface do tipo de política de negócios também.

Ao criar uma interface do tipo de política de negócios, a nova interface deve estender a interface `com.ibm.commerce.command.BusinessPolicyCommand`.

Escrevendo o Novo Comando de Política de Negócios

Para criar um novo comando de política de negócios, você deve criar um novo comando que implemente a interface do tipo de política de negócios com o qual o comando se relaciona. O novo comando também deve estender a classe de implementação `com.ibm.commerce.command.BusinessPolicyCommandImpl`. Isso é muito semelhante à criação de um novo comando de controlador ou de tarefa.

Há duas abordagens diferentes pelas quais você pode passar propriedades de entrada para um comando de política de negócios. A primeira maneira é ter propriedades de entrada padrão especificadas na coluna PROPERTIES da tabela POLICY. Para obter mais informações sobre essa tabela, consulte a seção a seguir.

A segunda abordagem é criar um novo campo no comando para cada uma das propriedades de entrada. Para cada campo, crie um novo par de métodos getter e setter.

Definindo requestProperties em Comandos de Política de Negócios

Há duas maneiras de definir requestProperties em um objeto de comando de política de negócios. A primeira utiliza a coluna PROPERTIES da tabela POLICY para definir as propriedades padrão. Isso é realizado pelo método setRequestProperties. A segunda maneira de definir propriedades é fazer com que o comando (de controlador ou de tarefa) que chama o comando de política de negócios defina outras propriedades de forma explícita.

Ao criar um novo comando de política de negócios, você deve substituir o método setRequestProperties padrão para incluir a lógica para definir de forma explícita cada um dos parâmetros incluídos no objeto requestProperties.

Considere um exemplo de um novo comando de política de negócios que tenha um nome de interface MyNewBusinessPolicyCmd e nome de classe de implementação MyNewBusinessPolicyCmdImpl.

Suponha que a entrada na tabela POLICY desse novo comando de política de negócios inclua os seguintes valores na coluna PROPERTIES:

- defaultProperty1=apple
- defaultProperty2=orange
- defaultProperty3=banana

A interface desse novo comando de política de negócios é definida da seguinte maneira:

```
public interface MyNewBusinessPolicyCmd extends
   com.ibm.commerce.command.BusinessPolicyCmd {
     java.lang.String defaultCommandClassName =
         'com.mycompany.mycommands.MyNewBusinessPolicyCmdImpl';
      public void setProperty1();
      public void setProperty2();
}
```

A classe de implementação desse novo comando de política de negócios é definida da seguinte maneira:

```
public class MyNewBusinessPolicyCmdImpl extends
   com.ibm.commerce.command.BusinessPolicyCmdImpl
   implements com.mycompany.mycommands.MyNewBusinessPolicyCmd {
      // Establish default properties that are stored in the POLICY table

      private java.lang.String defaultProperty1;
      private java.lang.String defaultProperty2;
      private java.lang.String defaultProperty3;

      // Begin to establish properties that must be set
```

```
      // by the calling command.

       //  *** property1   ***
       private java.lang.String property1;
       public java.lang.String getProperty1() {
          return property1;
          }
       public void setProperty1(java.lang.String newProperty1) {
          property1 = newProperty1;
       }

       //  *** property2   ***
       private java.lang.String property2;
       public java.lang.String getProperty2() {
          return property2;
          }
       public void setProperty1(java.lang.String newProperty2) {
          property2 = newProperty2;
          }

       // End establishing properties that must be set
       // by the calling command.

       /* Upon instantiation the business policy command sets all
          default properties from the POLICY table into the
          requestProperties object.  The calling command
          is responsible for setting any other required properties.
       */

       public void setRequestProperties
          (com.ibm.commerce.datatype.TypedProperty
          requestProperties) {
          //  Get the default properties defined in the POLICY table
          setDefaultProperty1(requestProperties.get("defaultProperty1"));
          setDefaultProperty2(requestProperties.get("defaultProperty2"));
          setDefaultProperty3(requestProperties.get("defaultProperty3"));
          }

}
```

O comando que chama o novo comando de política de negócios pode ser definido de forma semelhante ao seguinte:

```
public class MyCallerCommandImpl
   extends com.ibm.commerce.command.TaskCommandImpl
   implements com.mycompany.mycommands.MyCallerCommand {

   /* Include all elements and processing required for the
      task command.
   */

   // Determine the policy ID and setPolicyId

   // Call the business policy command.
```

```
    cmd = (MyNewBusinessPolicyCmd) CommandFactory
             createPolicyCommand(policyId);

    // Set required properties

    cmd.setProperty1("Fruit salad");
    cmd.setProperty2("Favorite food");

  cmd.execute();
}
```

Registrando a Nova Política de Negócios e Comando de Política de Negócios

Após ter criado o novo comando de política de negócios, é necessário registrar a política de negócios e o comando da política de negócios no banco de dados.

As políticas de negócios são registradas na tabela POLICY. Essa tabela contém as seguintes colunas:

- POLICY_ID
  A chave principal. É o identificador da política.
- POLICYNAME
  Um nome de política exclusivo.
- POLICYTYPE_ID
  O identificador do tipo de política. É a chave externa para a tabela POLICYTYPE.
- STOREENT_ID
  A loja ou o grupo de lojas ao qual a política se aplica.
- PROPERTIES
  Propriedades padrão que podem ser definidas para o comando da política de negócios. Especificadas como pares nome-valor, por exemplo, `parm1=val1&parm2=val2`.
- STARTDATE
  A data de início (especificada como marca de hora) da política. Se NULL, a data de início é imediata.
- ENDDATE
  A data de encerramento (especificada como marca de hora) da política. Se NULL, não há data de encerramento.

Quando a nova política estiver registrada na tabela POLICY, será necessário registrar um relacionamento entre a política e o comando da política de negócios que implementa a política de negócios. A tabela POLICYCMD é utilizada com essa finalidade. A tabela POLICYCMD contém as seguintes colunas:

- POLICY_ID
  Referência da chave externa para a tabela POLICY.

- BUSINESSCMDCLASS
  O comando da política de negócios que implementa a política.
- PROPERTIES
  Propriedades padrão que podem ser definidas para o comando da política de negócios. Especificadas como pares nome-valor, por exemplo, `parm1=val1&parm2=val2`.

Relacionando um Objeto de Termos e Condições a uma Nova Política de Negócios

Na estrutura de contratos e políticas do WebSphere Commerce, os termos e condições (também referidos como *termos*) fornecem uma maneira de descrever um acordo entre um comprador e um vendedor. Os termos e condições podem ser utilizados em vários tipos de acordos de comercialização, como em um contrato e em um RFQ (request for quotation). Os objetos termos e condições normalmente se referem a políticas de negócios com um ajuste opcional. Por exemplo, um objeto de termos e condições de preço é criado por meio da escolha de um dos objetos da política de preços. No termo preço, um gerente de conta pode fazer ajustes no preço padrão da loja, como por exemplo:

- Uma porcentagem de desconto sobre a tabela de preços padrão
- Uma porcentagem de desconto em um conjunto especificado de produtos

Cada um dos ajustes é especificado como um termo e condição.

Ao criar uma nova política de negócios, deverá haver pelo menos um objeto de termos e condições que faça referência a essa política de negócios, se a política precisar ser utilizada em um contrato. Você pode relacionar um objeto de termos e condições à nova política de negócios (isso é feito capturando-se o relacionamento entre o objeto de termos e condições existente e a nova política de negócios no arquivo `B2BTrading.dtd`) ou criar um novo objeto de termos e condições relacionado à nova política de negócios.

Criando Novos Termos e Condições

Na arquitetura do WebSphere Commerce, novos objetos de termos e condições são criados pela execução das seguintes etapas:

1. Atualização do esquema do banco de dados para incluir o novo termo e condição.
2. Atualização do arquivo `B2BTrading.dtd` para refletir o novo termo e condição.
3. Criação de um novo bean corporativo para o termo e condição.
4. Atualização do WebSphere Commerce Accelerator para refletir o novo termo e condição ou utilização do comando carregamento de contrato para criar um novo contrato que utilize o novo termo e condição.

Nas seções seguintes, o exemplo MyTC é o novo objeto termo e condição.

Registrando o novo termo e condição no banco de dados

Quando um novo objeto termo e condição está sendo criado, é necessário atualizar o esquema do banco de dados para incluir esse objeto. As tabelas do banco de dados que devem ser atualizadas são a TCTYPE e a TCSUBTYPE.

A seguinte instrução SQL mostra um exemplo de como atualizar o esquema:

```
insert into TCTYPE (TCTYPE_ID) values ('MyTC');
insert into TCSUBTYPE (TCSUBTYPE_ID, TCTYPE_ID, ACCESSBEANNAME,
   DEPLOYCOMMAND)
values ('MySubTC, 'MyTC ',
   'com.ibm.commerce.contract.objects.MySubTCAccessBean',
'packagename.MySubTCDeployCmd');
```

Registrar o novo termo e condição na definição do tipo de documento de contrato

O arquivo B2BTrading.dtd é o arquivo DTD (Document Type Definition) que especifica os vários termos e condições que podem ser utilizados nas políticas de negócios. Para disponibilizar o novo termo e condição em contratos, atualize esse arquivo para incluir o novo termo e condição.

Quando você criar um novo termo e condição, é necessário incluir o novo termo e condição na definição TermCondition e criar um novo elemento que descreva o termo e a condição.

Para atualizar o arquivo B2BTrading.dtd, proceda da seguinte maneira:

1. Navegue para o seguinte diretório:
   - Windows *unidade*:\WebSphere\CommerceServer\xml\trading
   - AIX /usr/WebSphere/CommerceServer/xml/trading
   - Solaris /opt/WebSphere/CommerceServer/xml/trading
   - Linux /opt/WebSphere/CommerceServer/xml/trading
   - 400 /QIBM/ProdData/WebCommerce/xml/trading
2. Abra o arquivo B2BTrading.dtd.
3. Atualize a definição TermCondition com o novo termo e condição. Por exemplo, a atualização é mostrada em negrito na seguinte definição TermCondition:

   ```
   <!ELEMENT TermCondition (TermConditionDescription?,Participant*,
   CreateTime?,UpdateTime?,(PriceTC|ProductSetTC|ShippingTC|FulfillmentTC|
   PaymentTC|ReturnTC|InvoiceTC|RightToBuyTC|ObligationToBuyTC|
   PurchaseOrderTC|OrderApprovalTC|DisplayCustomizationTC|
   OrderTC|MyTC))>
   ```

Observe que as quebras de linha são utilizadas apenas para a finalidade de exibição.

4. Agora, inclua o novo elemento no arquivo `B2BTrading.dtd`. Por exemplo, o seguinte mostra a atualização para incluir o elemento MyTC, que se refere a uma política de negócios e tem dois atributos obrigatórios.

```
<!ELEMENT MyTC (MySubTC)>
<!ELEMENT MySubTC (PolicyReference)>
<!ATTLIST MySubTC
   attr1 CDATA #REQUIRED
   attr2 CDATA #REQUIRED
>
```

5. Salve o arquivo.

Criando um Novo Bean Corporativo CMP para o Termo e Condição

Você deve criar um novo bean corporativo CMP para o termo e objeto de condição. O bean é criado para o subtipo de condição e termo.

Observe que, geralmente, ao criar novos beans corporativos, você colocaria os beans em seu próprio grupo EJB, ao invés de incluí-los em um dos grupos EJB que contém beans corporativos do WebSphere Commerce. Neste caso, uma vez que todos os novos beans de entidade para termos e condições devem ser herdados do bean TermCondition do WebSphere Commerce, você deve colocar seus novos beans de termos e condições no grupo EJB de Contrato WCS.

Para criar o novo bean corporativo CMP para o objeto de condição e termo, faça o seguinte em VisualAge for Java:

1. Utilize um assistente para criar o novo bean corporativo fazendo o seguinte:
   a. No Workbench do VisualAge for Java, selecione a guia EJB.
   b. Destaque, em seguida, clique com o botão direito do mouse no grupo EJB de **Contrato WCS** e selecione **Incluir > Bean Corporativo com Herança**.
      O SmartGuide Criar Bean Corporativo com Herança se abre.
   c. No SmartGuide, digite as informações adequadas para o bean. Por exemplo, a tabela a seguir mostra os exemplos de valores.

Atributo	**Valor**
Nome do bean	`MySubTC`
Herdar de	`TermCondition`
Pacote	`com.ibm.commerce.contract.objects`
Classe do bean	`MySubTCBean`
Interface remota	`MySubTC`

Atributo	Valor
Interface inicial	`MySubTCHome`

d. Clique em **Incluir** para incluir os campos CMP no bean e crie novos campos para o bean, conforme necessário. Para este exemplo, dois novos campos CMP são criados utilizando as seguintes informações:

Atributo	Valor
Nome do campo	`attr1`
Tipo de campo	`String`
Acessar com os métodos getter e setter	ativar
Promover os métodos getter e setter para a interface remota	ativar

Atributo	Valor
Nome do campo	`attr2`
Tipo de campo	`Integer`
Acessar com os métodos getter e setter	ativar
Promover os métodos getter e setter para a interface remota	ativar

e. Clique em **Concluir**.

2. A próxima etapa é mapear os campos do novo bean para colunas na tabela TERMCOND. Para criar estas informações de mapeamento, faça o seguinte:
   a. Do menu **EJB**, selecione **Abrir em > Mapas de Esquema**. O Navegador do Mapa é aberto.
   b. No painel Mapas de Datastore do navegador de mapas, clique duplo **WCS Contract**.
   c. No painel Classes Persistentes, clique duplo **TermCondition** em seguida selecione **MySubTC**.
   d. No menu Mapas da Tabela, selecione **Novo Mapa de Tabela > Incluir Mapa da Tabela Sem Herança**.
   O editor Mapa da Tabela sem Herança Única é aberto.
   e. No campo **Valor Discriminador**, digite o valor TCSUBTYPE_ID. Por exemplo, neste caso digite `'MySubTC'` (inclua as aspas) e clique em **OK**.
   f. Assegure-se de que **MySubTC** ainda está selecionado no painel Classes Persistentes. No painel Mapas da Tabela, destaque e clique com o botão direito do mouse na tabela TERMCOND. Selecione Editar Mapas de

Propriedade.
O editor de Mapas de Propriedade é aberto.

g. No Editor de Mapas de Propriedade, defina os atributos conforme a seguir:

Atributos de Classe	Tipo de Mapa	Coluna da Tabela
attr1	Simples	STRINGFIELD2
attr2	Simples	INTEGERFIELD1

e clique em **OK**.

h. No menu Mapas do Datastore, selecione Salvar Mapa do Datastore. Digite as seguintes informações ao salvar o mapa:

Atributo	Valor
Projeto	IBM WCS Enterprise Beans
Pacote	WCSContract EJB Reserved
Nome da Classe	WCSContractMap

Clique em **Concluir** e em seguida, feche o Navegador do Mapa.

3. No novo bean corporativo (isto é, no MySubTCBean) crie um novo método `ejbCreate(java.lang.Long argTradingId, org.w3c.dom.Element argElement)`, da seguinte maneira:

```
public void ejbCreate(java.lang.Long argTradingId,
   org.w3c.dom.Element argElement)
   throws javax.ejb.CreateException, javax.ejb.FinderException,
   java.rmi.RemoteException, javax.naming.NamingException,
   javax.ejb.RemoveException {
    _initLinks();
    super.ejbCreate (argTradingId, argElement);
    this.attr1= null;
    this.attr2= null;
}
```

4. Crie um novo método `ejbPostCreate(java.lang.Long argTradingId, org.w3c.dom.Element argElement)`, conforme a seguir:

```
public void ejbPostCreate(java.lang.Long argTradingId,
   org.w3c.dom.Element argElement)
   throws javax.ejb.CreateException, javax.ejb.FinderException,
   java.rmi.RemoteException, javax.naming.NamingException,
   javax.ejb.RemoveException
   {
    parseXMLElement(argElement);
   }
```

5. Substitua o método `parseXMLElement(org.w3c.dom.Element argElement)` em MySubTCBean, da seguinte maneira:

```
public void parseXMLElement(org.w3c.dom.Element argElement) throws
   javax.ejb.CreateException,
   javax.ejb.FinderException,
   java.rmi.RemoteException,
   javax.naming.NamingException,
   javax.ejb.RemoveException
   {
      super.parseXMLElement(argElement);

      if (argElement == null)
     return;

      String nodeName = argElement.getNodeName();
      if (nodeName.equals("TCCopy"))
     return;

      // get element "MyTC"
      Element eMyTC = ContractUtil.getElementByTag(argElement,"MyTC");

      // get element "MySubTC" from element "MyTC"
      Element eMySubTC = ContractUtil.getElementByTag(eMyTC ,"MySubTC");
      this.attr1 = eMySubTC .getAttribute("attr1").trim();
      this.attr2 = new Integer (eMySubTC .getAttribute("attr2").trim());

      // get element "PolicyReference" from "MySubTC"
      Element ePolicyReference = ContractUtil.getElementByTag(eMySubTC,
         "PolicyReference");
      parseElementPolicyReference(ePolicyReference);

}
```

6. Substitua o método createNewVersion(Long argNewTradingId) em MySubTCBean, da seguinte maneira:

```
public Long createNewVersion(Long argNewTradingId) throws
   javax.ejb.CreateException,
   javax.ejb.FinderException,
   java.rmi.RemoteException,
   javax.naming.NamingException,
   javax.ejb.RemoveException,
   org.xml.sax.SAXException,
   java.io.IOException

   {

   // Contract a seqElement since tcSequence can not be null
   Element seqElement = ContractUtil.getSeqElementFromTCSequence(
      this.tcSequence);
   MySubTCAccessBean newTC = new MySubTCAccessBean(argNewTradingId,
     seqElement);

   Long newTCId = newTC.getReferenceNumberInEJBType();
   newTC.setInitKey_referenceNumber(newTCId.toString());
   newTC.setMandatoryFlag(this.mandatoryFlag);
   newTC.setChangeableFlag(this.changeableFlag);
   // set columns for this specific TC
```

```
   newTC.setAttr1(this.attr1);
   newTC.setAttr2(this.attr2);
   newTC.commitCopyHelper();

   return newTCId;
}
```

7. Substitua o método `getXMLString()` em MySubTCBean, da seguinte maneira:

```
public String getXMLString() throws
   javax.ejb.CreateException,
   javax.ejb.FinderException,
   java.rmi.RemoteException,
   javax.naming.NamingException
   {
      String xmlTC = "      <MyTC>"  +
                     "%XML_POLICYREFERENCE%" +
                     "         <MySubTC attr1=\"" + this.attr1 +
                     "\"  attr2=\"" + this.attr2.toString() + "\"/>"
                     "         '>" +
                     "      <MYTC></MYTC>" ;
      String xmlPolicy = getXMLStringForElementPolicyReference(
         "ProductSet") ;
      xmlTC = ContractUtil.replace( xmlTC, "%XML_POLICYREFERENCE%",
         xmlPolicy );

      return xmlTC;
}
```

8. Substitua o método `markForDelete()` em MySubTCBean, da seguinte maneira:

```
public void markForDelete() throws
   javax.ejb.CreateException,
   javax.ejb.FinderException,
   java.rmi.RemoteException,
   javax.naming.NamingException
   {
      // code: remove entries from associated tables which
      // cannot be deleted though delete cascade
}
```

9. Assegure-se de que o método `ejbCreate` tenha sido incluído à interface inicial e que todos os outros métodos modificados tenham sido incluídos à interface remota.
10. A próxima etapa é criar um bean de acesso para o bean de entidade MySubTC fazendo o seguinte:
    a. Clique com o botão direito do mouse no bean de entidade **MySubTC** e selecione **Incluir > Bean de Acesso**.
       O SmartGuide Criar Bean de Acesso é aberto.

b. Certifique-se de que as seguinte informações sejam digitadas:

Tabela 1.

Atributo	Valor
Grupo EJB	`WCSContract`
Bean Corporativo	`MySubTC`
Nome do Bean de Acesso	`MySubTCAccessBean`
Tipo do Bean de Acesso	`Copiar Assistente para um Bean de Entidade`

e clique em **Avançar**.

c. Na lista suspensa **Selecionar método inicial para construtor de argumento zero**, selecione **findByPrimaryKey(TermConditionKey)**

d. Para **initKey_referenceNumber** (na coluna Propriedades Iniciais), defina **Conversor** como `com.ibm.commerce.base.objects.WCStringConverter` e clique em **Avançar**.

e. Para todos os novos campos que você tiver incluído, assegure-se de que **CopyHelper** esteja selecionado e defina o valor do conversor para cada um dos `com.ibm.commerce.base.objects.WCStringConverter`. Clique em **Concluir**.
Após a geração do código ser concluída, é possível exibir o novo código mudando para a guia **Projetos**, expandindo o projeto **IBM WCS Enterprise Beans** e em seguida expandindo o pacote **com.ibm.commerce.contract.objects**.

11. Volte para a guia EJB, clique com o botão direito do mouse no bean corporativo **MySubTC** e selecione **Gerar Código Implementado**.

12. Também é necessário gerar novamente o código implementado para o bean pai (o bean TermCondition) e todos os beans irmãos (todos os outros beans no grupo WCS Contract EJB que contêm "TC" em seus nomes). Observe que se você tiver incluído um novo campo ou modificado a interface remota do bean TermCondition existente, então pode ser necessário gerar novamente os beans de acesso sozinhos e todos os seus beans filhos também.
Para gerar novamente o código implementado, faça o seguinte:

a. Destaque o bean TermCondition e todos os outros beans que contêm "TC" em seus nomes (por exemplo, DisplayCustomizationTC, FulfillmentTC e nvoiceTC são poucos do beans irmãos).

b. Com todos estes beans destacados, clique com o botão direito do mouse e selecione **Gerar Código Implementado**.

13. A próxima etapa é substituir métodos que estão no comando de tarefa `ValidateContractCmd`. Nesse comando, há três métodos que talvez você queira substituir para suportar o novo objeto termo e condição. São eles:

- `validateTCType()`
  Esse método verifica que tipo de termo por constar em um contrato. Por exemplo, o InvoiceTC pertence à conta e, portanto, não pode aparecer em um contrato.
- `validateTCOccurrence()`
  Esse método verifica a ocorrência dos termos. Por exemplo, na implementação padrão desse método, um contrato precisa ter pelo menos um PriceTC.
- `otherValidateCheck()`
  A implementação padrão desse método é vazia. É possível incluir qualquer validação adicional que não coincida com os dois primeiros métodos.

14. Se o termo e condição deve ser implementado, é necessário criar um novo comando de implementação e registrar esse comando no banco de dados. Se necessário, proceda da seguinte maneira:
    a. Neste exemplo, o novo comando de implementação é chamado de `MySubTCDeployedCmd` e a classe de implementação é chamada de `MySubTCDeployedCmdImpl`. Além disso, o comando é incluído no pacote `packagename`. Para registrar esse comando, emita o seguinte comando SQL:

       ```
       insert into CMDREG (STOREENT_ID, INTERFACENAME, CLASSNAME, TARGET)
       values (0, 'packagename.MySubTCDeployCmd',
       'packagename.MySubTCDeployCmdImpl', 'Local');
       ```

    b. No pacote `packagename`, crie a nova interface `MySubTCDeployedCmd`. Essa interface deve estender a interface do comando `com.ibm.commerce.contract.commands.DeployTCCmd`. O seguinte descreve a nova interface do comando:

       ```
       public interface MySubTCDeployCmd extends
          com.ibm.commerce.contract.commands.DeployTCCmd
          {
              // customized code
          }
       ```

       Há um parâmetro protegido `abTC` e um método chamado `getTargetStoreId()` em `DeployTCCmd`. O valor de `abTC` é `MySubTCAccessBean` e o método `getTargetStoreId()` retorna o identificador da loja na qual o contrato está sendo implementado.
    c. No mesmo pacote, crie a classe de implementação `MySubTCDeployCmdImpl`. Essa classe de implementação estende a `com.ibm.commerce.contract.commands.DeployTCCmdImpl`. O seguinte descreve a nova classe de implementação do comando:

```
public class MySubTCDeployCmdImpl
   extends com.ibm.commerce.contract.commands.DeployTCCmdImpl
   implements MySubTCDeployCmd
   {
       // customer code
   }
```

Atualizando o WebSphere Commerce Accelerator para Utilizar um Novo Termo e Condição

Depois de criar novos termos e condições, você pode atualizar o WebSphere Commerce Accelerator para que ele possa ser utilizado para criar novos contratos que incluem os novos termos e condições. A atualização do WebSphere Commerce Accelerator para essa finalidade inclui as seguintes etapas:

1. Criar um novo arquivo JavaScript para os novos termos e condições. Para a finalidade do exemplo nesta seção, esse arquivo é referido como `Extensions.js`.
2. Criar um novo modelo JSP que inclua uma seção HTML na qual um usuário pode digitar as informações necessárias para os novos termos e condições. Para finalidade do exemplo nesta seção, esse arquivo é referido como `ContractMyTC.jsp`.
3. Criar um novo bean de dados para os novos termos e condições. Para a finalidade do exemplo nesta seção, esse arquivo é referido como `MyTCDataBean`.
4. Registrar a nova exibição na tabela VIEWREG.
5. Atualizar o arquivo `ContractRB_`*`locale`*`.properties` para incluir os novos recursos.
6. Editar o arquivo `ContractNotebook.xml` para incluir a nova página.

Cada uma das etapas é descrita com maiores detalhes nas seções a seguir.

Criando o Novo Arquivo JavaScript: A primeira etapa para atualizar o WebSphere Commerce Accelerator para utilização de novos termos e condições é criar um novo arquivo JavaScript para eles. Como referência, você pode consultar o seguinte arquivo de exemplo:

- Windows *unidade*`:\WebSphere\CommerceServer\samples\contract\Extensions.js`
- Windows *unidade*`:\WebSphere\CommerceServerDev\samples\contract\Extensions.js`
- AIX `/usr/WebSphere/CommerceServer/samples/contract/Extensions.js`
- Solaris `/opt/WebSphere/CommerceServer/samples/contract/Extensions.js`
- Linux `/opt/WebSphere/CommerceServer/samples/contract/Extensions.js`
- 400 `/QIBM/ProdData/WebCommerce/samples/contract/Extensions.js`

Para utilizar esse arquivo de exemplo, copie-o no seguinte diretório:

- Windows *unidade*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wctools.war\javascript\tools\contract
- AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/javascript/tools/contract
- Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/javascript/tools/contract
- Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/javascript/tools/contract
- 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/javascript/tools/contract

Nesse novo arquivo, você deve criar um objeto JavaScript para armazenar os dados para o novo termo e condição. Isso é mostrado no seguinte trecho de código:

```
function ContractMyTCModel() {

   this.tcReferenceNumber = "";
   this.policyReferenceNumber = "";

   this.attr1 = "";
   this.attr2 = "";

   this.policyList = new Array();
   this.selectedPolicyIndex = "0";
}
```

Você também deve criar um novo objeto JavaScript para submeter o novo termo e condição. Isso deve ser feito de uma maneira consistente com as extensões feitas no arquivo B2BTrading.dtd. Isso é mostrado no seguinte trecho de código:

```
function submitMyTC(termsAndConditions) {

   var tcModel = get("ContractMyTCModel");

   if (tcModel != null) {

      var myTC = new Object();
      myTC.MyTC = new Object();
      myTC.MyTC.MySubTC = new Object();
      myTC.MyTC.MySubTC.attr1 = tcModel.attr1;
```

```
        myTC.MyTC.MySubTC.attr2 = tcModel.attr2;

        myTC.MyTC.PolicyReference = new Object();
        myTC.MyTC.PolicyReference.policyName =
           tcModel.policyList[tcModel.selectedPolicyIndex].policyName;
        myTC.MyTC.PolicyReference.policyType = "ProductSet";
        myTC.MyTC.PolicyReference.storeIdentity =
           tcModel.policyList[tcModel.selectedPolicyIndex].storeIdentity;
        myTC.MyTC.PolicyReference.Member =
           tcModel.policyList[tcModel.selectedPolicyIndex].member;

        if (tcModel.tcReferenceNumber != "") {
           // Change the term and condition
           myTC.action = "update";
           myTC.referenceNumber = tcModel.tcReferenceNumber;
        }
        else {
           // Create a new term and condition
           myTC.action = "new";
        }

        termsAndConditions[termsAndConditions.length] = myTC;
     }

       return true;
}
```

Criando o Novo Modelo JSP: A próxima etapa é criar um novo modelo JSP que inclua uma seção HTML na qual o usuário possa digitar as informações requeridas pelo novo termo e condição. Como referência, você pode consultar o seguinte arquivo de exemplo:

- Windows *unidade*:\WebSphere\CommerceServer\samples\contract\ContractMyTC.jsp
- Windows *unidade*:\WebSphere\CommerceServerDev\samples\contract\ContractMyTC.jsp
- AIX /usr/WebSphere/CommerceServer/samples/contract/ContractMyTC.jsp
- Solaris /opt/WebSphere/CommerceServer/samples/contract/ContractMyTC.jsp
- Linux /opt/WebSphere/CommerceServer/samples/contract/ContractMyTC.jsp
- 400 /QIBM/ProdData/WebCommerce/samples/contract/ContractMyTC.jsp

Para utilizar esse arquivo de exemplo, copie-o no seguinte diretório:

- Windows *unidade*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wctools.war\tools\contract
- AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/tools/contract
- Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/tools/contract
- Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/tools/contract
- 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/tools/contract

O seguinte trecho de código mostra uma seção HTML de exemplo de um modelo JSP que pode ser utilizada para MyTC.

```
<!--
////////////////////////////////////
// HTML SECTION
////////////////////////////////////
-->

<BODY onLoad="onLoad()" class="content">

   <H1>
   <%= contractsRB.get("MyTCHeading") %>
   </H1>

<FORM NAME="MyTCForm">

   <%= contractsRB.get("MyTCAttr1Label") %>
   <BR>
   <INPUT type=text name=Attr1 value="" size=10 maxlength=10>
   <BR>

   <%= contractsRB.get("MyTCAttr2Label") %>
   <BR>
   <INPUT type=text name=Attr2 value="" size=10 maxlength=10>
   <BR>

   <%= contractsRB.get("MyTCPolicyLabel") %>
   <BR>
   <SELECT NAME="PolicyList" SIZE="1">
   </SELECT>

</FORM>
```

Criando o Novo Bean de Dados: Nesta etapa, você criará um novo bean de dados que carrega os dados necessários do bean de acesso MySubTC. As seções relevantes de código são mostradas no seguinte trecho de código:

```
public class MyTCDataBean extends MySubTCAccessBean
   implements SmartDataBean, Delegator {
      private java.lang.Long contractId;
      private boolean hasMyTC = false;
      private CommandContext iCommandContext;


      /**
       * MyTCDataBean default constructor.
       */
      public MyTCDataBean() {
      }
      /**
       * MyTCDataBean constructor.
       */
      public MyTCDataBean(Long newContractId) {
       contractId = newContractId;
      }


      /*
       * populate the attributes from TermConditionAccessBean
       */
      public void populate() throws Exception {

            Enumeration myTCEnum = new TermConditionAccessBean().
                     findByTradingAndTCSubType(contractId, "MySubTC");
                  if (myTCEnum != null) {
                     // assume a contract only has one MyTC for this example

                        setEJBRef(((TermConditionAccessBean)
                              myTCEnum.nextElement()).getEJBRef());
                        refreshCopyHelper();
                        hasMyTC = true;
                  }

}
```

Registrando a Nova Exibição na Tabela VIEWREG: É necessário registrar a exibição recém-criada na tabela VIEWREG. A seguir está um exemplo de uma instrução SQL para registrar a nova exibição:

```
insert into VIEWREG(VIEWNAME,DEVICEFMT_ID,STOREENT_ID, INTERFACENAME,
   CLASSNAME, PROPERTIES, HTTPS, INTERNAL)
values ('ContractMyTCPanelView', -1, 0,
   'com.ibm.commerce.tools.command.ToolsForwardViewCommand',
   'com.ibm.commerce.tools.command.ToolsForwardViewCommandImpl',
   'docname=tools/contract/ContractMyTC.jsp', 1, 1)
```

Atualizando o Arquivo ContractRB_*locale*.properties: Você deve atualizar o seguinte arquivo de propriedades com informações específicas do novo termo e condição:

- Windows *unidade*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\properties\com\ibm\commerce\tools\contract\properties\ContractRB_*locale*.properties
- AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/properties/com/ibm/commerce/tools/contract/properties/ContractRB_*locale*.properties
- Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/properties/com/ibm/commerce/tools/contract/properties/ContractRB_*locale*.properties
- Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/properties/com/ibm/commerce/tools/contract/properties/ContractRB_*locale*.properties
- 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/properties/com/ibm/commerce/tools/contract/properties/ContractRB_*locale*.properties

A seguir está um exemplo das informações que devem ser incluídas no arquivo.

```
MyTCHeading=My TC
attr1Empty=Attribute One must be entered.
attr2Empty=Attribute Two must be entered.
attr1TooLong=Attribute One is too long.
attr2TooLong=Attribute Two is too long.
MyTCAttr1Label=Attribute One (required)
MyTCAttr2Label=Attribute Two (required)
MyTCPolicyLabel=Policy
```

Editando o Arquivo `ContractNotebook.xml`: A última etapa para inclusão dos novos termos e condições no WebSphere Commerce Accelerator é atualizar o arquivo a seguir para incluir a nova página.

- Windows *unidade*:\WebSphere\CommerceServer\xml\tools\contract\ContractNotebook.xml
- AIX /usr/WebSphere/CommerceServer/xml/tools/contract/ContractNotebook.xml
- Solaris /opt/WebSphere/CommerceServer/xml/tools/contract/ContractNotebook.xml

- Linux /opt/WebSphere/CommerceServer/xml/tools/contract/ContractNotebook.xml
- 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/xml/tools/contract/ContractNotebook.xml

A seguir está um exemplo de fragmento de código utilizado para incluir a nova página neste exemplo.

```
<panel name="MyTCHeading"
    url="ContractMyTCPanelView"
    parameters="contractId,accountId"
    helpKey="MC.contract.MyTCPanel.Help" />
```

Importando o Novo Contrato Utilizando o Novo Termo e Condição

Como alternativa para atualizar as ferramentas do WebSphere Commerce para utilização de um novo termo e condição, você pode utilizar o comando para importar contrato (consulte a ajuda online do WebSphere Commerce para obter informações sobre esse comando) para importar um novo contrato que inclua esse novo termo e condição. Após a importação, a seção relevante no arquivo `Contract.xml` aparece da seguinte forma:

```
<TermCondition>
   <MyTC>
      <MySubTC attr1="adc" attr2="123" />
         <PolicyReference policyName = "Product Set 1"
                          policyType = "ProductSet"
                          storeIdentity = "StoreGroup1" >
            <Member>
               <User distinguishName = "uid=wcsadmin,o=Root Organization"/>
            </Member>
         </PolicyReference>
      </MyTC>
 </TermCondition>
```

Chamando a Nova Política de Negócios

Depois de ter criado uma nova política de negócios e ela ter sido associada a pelo menos um objeto de termos e condições, você poderá atualizar a lógica do aplicativo para chamar os novos comandos da política de negócios.

Os comandos da política de negócios são chamados dos comandos de controlador e de tarefa.

A fábrica de comandos é utilizada para chamar comandos de política de negócios. Há dois métodos `create` que podem ser utilizados para chamar comandos de política de negócios. O primeiro é utilizado para chamar um comando de política de negócios quando há somente um comando de política de negócios associado à política de negócios. Isso é mostrado no fragmento de código a seguir:

```
CommandFactory createBusinessPolicyCommand(Long policyId);
```

O segundo é utilizado para chamar um comando de política de negócios quando há mais de um comando de política de negócios associado à política de negócios. Isso é mostrado no fragmento de código a seguir:

```
CommandFactory createBusinessPolicyCommand(Long policyId, String cmdIfName);
```

No exemplo acima, `cmdIfName` é utilizado para especificar o nome da interface do comando de política de negócios a ser criado.

A fábrica de comandos procura o objeto de política na tabela POLICYCMD para determinar o comando que implementa essa política. Também busca as propriedades padrão na tabela e as define como requestProperties no comando de política de negócios.

O fragmento de código a seguir mostra um exemplo de chamar uma política de reembolso:

```
RefundPolicyCmd  cmd;

   ////////////////////////////////////////////////////////
   // Get the refund policy id from the refundTC object     //
   //  and use it to create the policy command.             //
   ////////////////////////////////////////////////////////
 cmd = (RefundPolicyCmd) CommandFactory
         createPolicyCommand (refundTC.getRefundPolicy);

  cmd.execute()
```

Criando um Contrato

A próxima etapa para criar integralmente o modelo de controle no processo de negócios é criar um contrato que inclua os termos e condições referentes à nova política de negócios. Um contrato pode ser criado utilizando-se o WebSphere Commerce Accelerator ou utilizando-se um dos comandos de URL do contrato (ContractImportApprovedVersion e ContractImportDraftVersion). Para obter mais informações sobre a criação de contratos, consulte a ajuda online do WebSphere Commerce.

Cenários de Personalização de Contratos

Esta seção fornece uma visão geral das etapas envolvidas para o seguinte cenário de personalização do contrato:

- Ativando um desconto

Cenário de Desconto

Neste cenário de exemplo, é criado um desconto de taxa fixa. Uma vez que a loja de exemplo ToolTech não inclui nem um termo e condição nem um tipo

de política que corresponda ao cenário de desconto, eles deverão ser criados. Adicionalmente, uma nova política de negócios deverá ser criada, bem como uma tabela do banco de dados para armazenar os códigos de desconto.

Implementar este cenário de desconto inclui as seguintes etapas de alto nível:

1. Criando a tabela do banco de dados XREBATECODE e um bean de entidade XRebateCodeBean correspondente utilizado para acessar as informações a partir desta tabela.
2. Criando uma nova política de negócios 5DollarRebate executando as seguintes subtarefas:
   a. Crie o novo tipo de política de negócios correspondente. Isso define a interface (RebatePolicyCmd) que o novo comando da política de negócios implementará.
   b. Crie o novo comando da política de negócios CalculateRebateCmdImpl.
   c. Registre o novo comando da política de negócios e o tipo de política de negócios no banco de dados.
3. Crie um novo termo e condição (RebateTC) para o desconto executando as seguintes subtarefas:
   a. Registre o termo e a condição RebateTC no banco de dados.
   b. Atualize o arquivo `B2BTrading.dtd` para refletir o novo RebateTC.
   c. Crie um novo bean corporativo para RebateTC.
   d. Atualize o WebSphere Commerce Accelerator para refletir o novo RebateTC.
4. Crie um novo contrato que utilize o RebateTC.
5. Integrando a nova política de negócios no fluxo de compras.

Cada uma das etapas está descrita em mais detalhes nas seções subseqüentes.

Etapa 1: Criando a Nova Tabela e o Bean Corporativo

Uma vez que o esquema do banco de dados existente não inclui a especificação de um código e valor de desconto, uma nova tabela deverá ser criada. Em geral, quando uma nova tabela é criada, um novo bean de entidade também é criado, utilizado para acessar as informações contidas nesta tabela.

Para a finalidade deste exemplo, considere que a seguinte tabela do banco de dados XREBATECODE será criada.

Tabela 2. Tabela do banco de dados XREBATECODE

	Nome da coluna		
	REBATECODE_ID	AMOUNT	CURRENCY

Tabela 2. Tabela do banco de dados XREBATECODE (continuação)

Exemplo de dados	201	5	CAD
	202	10	CAD

Além disso, um novo bean de entidade CMP (XRebateCodeBean) seria criado. Para obter informações detalhadas sobre a criação deste bean, consulte "Criando um Novo Bean Corporativo CMP" na página 62.

Etapa 2: Criando a Política de Negócios "5DollarRebate"

Para criar esta nova política de negócios, você deve executar as seguintes etapas:

1. Crie a nova interface do tipo de política de negócios. Esta é a interface `RebatePolicyCmd` que `CalculateRebateCmdImpl` implementará.
2. Crie o novo comando da política de negócios `CalculateRebateCmdImpl`.
3. Registre a nova política de negócios e o comando da política de negócios no banco de dados.

Criando o Tipo de Política de Negócios "Desconto": Uma vez que não existe um tipo de política de negócios existente que corresponda aos descontos, um novo deverá ser criado. Criar um novo tipo de política de negócios envolve a definição e o registro de um tipo de política no banco de dados. As tabelas a seguir devem ser atualizadas:

- POLICYTYPE
- PLCYTYCMIF
- PLCYTYPDSC

Para este cenário, para criar o novo tipo de política REBATE, as seguintes instruções SQL seriam utilizadas:

```
insert into POLICYTYPE (POLICYTYPE_ID) values ('Rebate');
insert into PLCYTYCMIF (POLICYTYPE_ID, BUSINESSCMDIF)
   values ('Rebate',
   'com.mycompany.mybusinesspolicycommands.RebatePolicyCmd');
insert into  PLCYTYPDSC (POLICYTYPE_ID, LANGUAGE_ID, DESCRIPTION)
   values ('Rebate', -1,
   'Rebate policy type.');
```

Como resultado, a seguinte tabela mostra as colunas relevantes da tabela PLCYTYCMIF que mostra o relacionamento entre o tipo de política e o comando da política de negócios com a qual está relacionado.

Tabela 3. Atualizações feitas na tabela PLCYTYCMIF

	Nome da coluna	
	POLICYTYPE_ID	BUSINESSCMDIF

Tabela 3. Atualizações feitas na tabela PLCYTYCMIF (continuação)

Dados de exemplo	Desconto	com.mycompany.mybusinesspolicycommands.RebatePolicyCmd

Você também deve codificar a nova interface `RebatePolicyCmd`. Essa interface deve estender a interface `com.ibm.commerce.command.BusinessPolicyCommand`. Como sugerido na tabela anterior, empacote essa interface em seu próprio pacote.

Criando o Comando da Política de Negócios CalculateRebateCmdImpl: Para criar o novo comando da política de negócios, você deve criar um novo comando chamado `CalculateRebateCmdImpl` que estende a classe de implementação `com.ibm.commerce.command.BusinessPolicyCommandImpl`. Este comando deveria implementar a interface `RebatePolicyCmd` criada na etapa anterior.

Observe que, neste exemplo, o nome da interface e o nome do comando não são semelhantes. Esses nomes são escolhidos para mostrar intencionalmente que pode haver muitos comandos de política de negóciso que implementam o tipo de desconto da política de negócios. Cada implementação (ou seja, cada comando da política de negócios) implementaria o desconto de maneira exclusiva.

A lógica do comando depende de uma determinada implementação de como o cliente deve escolher as mercadorias. Adicionalmente, `CalculateRebateCmdImpl` deveria ser invocado por um controlador separado ou comando de tarefa em seu aplicativo.

Registrando a Nova Política de Negócios e Novo Comando da Política de Negócios: A nova política de negócios deve ser registrada no banco de dados. Você também deve registrar o relacionamento entre a nova política de negócios e o novo comando de política de negócios.

Para registrar essas informações, você pode utilizar o comando `com.ibm.commerce.contract.commands.PolicyAddCmd`. A seguir, um exemplo do uso do comando PolicyAdd para este cenário:

```
http://localhost:8080/webapp/wcs/stores/servlet/PolicyAdd?
   type=Rebate&name=5DollarRebate&plcyStoreId=-1
   &cmd_1=com.mycompany.mybusinesspolicycommands.CalculateRebateCmdImpl
   &startDate=2002-05-08%2000:00:00&endDate=2003-05-09%2000:00:00
   &commonProps=rebatecode_id%3D501&URL=aRedirectURL
```

Observe que os caracteres reservados da URL devem ser substituídos por seus códigos ASCII para propriedades de entrada. Como tal, o símbolo comum = (igual) é substituído por “%3D”, o & (e comercial) é substituído por “%26” e o

caractere de espaço é substituído por "%20".O formato da data utilizado no exemplo anterior é aaaa-mm-dd hh:mm:ss, com o código ASCII substituindo os caracteres reservados da URL.

As tabelas a seguir mostram as colunas relevantes das tabelas do banco de dados afetado depois de executar as atualizações.

Tabela 4. Atualizações feitas na tabela POLICY

	Nome da coluna				
	POLICY _ID	POLICY NAME	POLICYTYPE _ID	STOREENT _ID	PROPERTIES
Exemplo de dados	301	5Dollar Rebate	Rebate	-1	rebatecode_id= 201

Observe que também considera-se que os valores da data de início e da data de encerramento estão definidos como `nulos`.

Tabela 5. Atualizações feitas na tabela POLICYCMD

	Nome da coluna		
	POLICY _ID	BUSINESS CMDCLASS	PROPERTIES
Exemplo de dados	301	com.mycompany. mybusinesspolicycommands. CalculateRebateCmdImpl	nulo

Como conseqüência, agora você tem uma nova política de negócios chamada "5DollarRebate", relacioanda ao comando da política de negócios CalculateRebateCmd.

Etapa 3: Criando o Termo e a Condição "RebateTC"

Criar o termo e a condição "RebateTC" exige que as seguintes etapas sejam executadas:

1. Registre o termo e a condição RebateTC no banco de dados.
2. Atualize o arquivo B2BTrading.dtd para refletir o novo RebateTC.
3. Crie um novo bean corporativo para RebateTC.
4. Atualize o WebSphere Commerce Accelerator para refletir o novo RebateTC.

Registrando o Termo e a Condição "RebateTC" no Banco de Dados:
Quando um novo objeto termo e condição está sendo criado, é necessário atualizar o esquema do banco de dados para incluir esse objeto. As tabelas do banco de dados que devem ser atualizadas são a TCTYPE e a TCSUBTYPE.

A seguinte instrução SQL mostra um exemplo de como registrar o RebateTC no banco de dados:

```
insert into TCTYPE (TCTYPE_ID) values ('RebateTC');
insert into TCSUBTYPE (TCSUBTYPE_ID, TCTYPE_ID, ACCESSBEANNAME,
   DEPLOYCOMMAND)
values ('RebateTC, 'RebateTC ',
   'com.ibm.commerce.contract.objects.RebateTCAccessBean',
    null);
```

As tabelas a seguir mostram um resumo das colunas relevantes nas tabelas TCTYPE e TCSUBTYPE.

Tabela 6. Atualizações feitas na tabela TCTYPE

	Nome da coluna
	TCTYPE_ID
Exemplo de dados	RebateTC

Tabela 7. Atualizações feitas na tabela TCSUBTYPE

	Nome da coluna			
	TCSUBTYPE _ID	TCTYPE _ID	ACCESSBEAN NAME	DEPLOY COMMAND
Exemplo de dados	RebateTC	RebateTC	com.ibm.commerce. contract.objects. RebateTCAccessBean	nulo

Atualizando o Arquivo B2BTrading.dtd para Refletir o Novo RebateTC: Para disponibilizar o novo termo e condição em contratos, atualize o arquivo `B2BTrading.dtd` para incluir o novo termo e condição.Ao atualizar este arquivo, você deve incluir o novo termo e condição na definição TermCondition e, em seguida, criar um novo elemento que descreva o termo e a condição.

O texto em negrito mostra um exemplo de como incluir o novo RebateTC na definição TermCondition:

```
<!ELEMENT TermCondition (TermConditionDescription?,Participant*,
CreateTime?,UpdateTime?,(PriceTC|ProductSetTC|ShippingTC|FulfillmentTC|
PaymentTC|ReturnTC|InvoiceTC|RightToBuyTC|ObligationToBuyTC|
PurchaseOrderTC|OrderApprovalTC|DisplayCustomizationTC|
OrderTC|RebateTC))>
```

Observe que as quebras de linha são utilizadas apenas para a finalidade de exibição.

Você deve incluir uma nova sub-rotina que descreva este termo e condição para o arquivo. A seguir, um exemplo para RebateTC:

```
<!ELEMENT RebateTC (PolicyReference?)>
```

Criando um Novo Bean Corporativo para RebateTC: Você deve criar um novo bean corporativo para o novo RebateTC. Este novo bean deveria ser herdado do bean WebSphere Commerce TermCondition.

Um novo bean corporativo para um termo e condição geralmente chamado após o subtipo. Observe que, nesse caso, o subtipo do termo e condição é o mesmo que o tipo de termo e condição e, como tal, o nome do bean é igual ao tipo de termo e condição.

A tabela a seguir mostra algumas das informações gerais sobre o novo bean que deve ser criado. Para obter mais detalhes sobre o bean, incluindo quais métodos substituir, consulte "Criando um Novo Bean Corporativo CMP para o Termo e Condição" na página 167.

Tabela 8.

Atributo	Valor
Nome do bean	RebateTC
Herdar de	TermCondition
Pacote	com.ibm.commerce.contract.objects
Classe do bean	RebateTCBean
Interface remota	RebateTC
Interface inicial	RebateTCHome

Atualizando o WebSphere Commerce Accelerator para Incluir o RebateTC: Depois de criar novos termos e condições, você pode atualizar o WebSphere Commerce Accelerator para que ele possa ser utilizado para criar novos contratos que incluem os novos termos e condições. Para obter informações sobre como atualizar esta ferramenta, consulte "Atualizando o WebSphere Commerce Accelerator para Utilizar um Novo Termo e Condição" na página 174.

Etapa 4: Criando um Novo Contrato

Você deve criar um novo contrato que inclua o termo e a condição "RebateTC" e que se refira á política de negócios "5DollarRebate". Você pode utilizar o WebSphere Commerce Accelerator ou XML para criar um novo contrato. Cada um desses métodos para criar novos contratos são descritos na ajuda on-line WebSphere Commerce.

As tabelas a seguir mostram as atualizações para as colunas relevantes das tabelas do banco de dados TERMCOND e POLICYTC, depois que o contrato foi criado.

Tabela 9. Atualizações feitas na tabela TERMCOND

	Nome da coluna		
	TRADING _ID	TERMCOND _ID	TCSUBTYPE _ID
Exemplo de dados	25	901	RebateTC

Tabela 10. Atualizações feitas na tabela POLICYTC

	Nome da coluna	
	POLICY _ID	TERMCOND_ID
Exemplo de dados	301	901

Etapa 5: Integrando a Nova Política de Negócios no Fluxo de Compras

Neste cenário, considere que uma nova página seja incluída na loja quer permite que os clientes efetuem logon e reivindique seus descontos. Quando o cliente clicar para reivindicar o desconto, um comando que chame a nova interface `RebatePolicyCmd` deveria ser chamado. Por exemplo, poderia haver um novo comando do controlador ClaimRebateCmd que chama `RebatePolicyCmd`. A política de negócios correta é encontrada e (nesse caso) a política de negócios “5DollarRebate” é aplicada.

Parte 3. Ambiente de Desenvolvimento

Capítulo 8. Ferramentas de Desenvolvimento e Implementação

Esse capítulo introduz as principais ferramentas de desenvolvimento utilizadas para personalizar o aplicativo do WebSphere Commerce. Descreve o processo para implementar o código personalizado do VisualAge for Java para um WebSphere Commerce Server. Descreve também como implementar o código no Commerce Studio, de forma que você possa tirar vantagem do código personalizado ao desenvolver modelos JSP.

Ambiente de Desenvolvimento

O pacote de desenvolvimento recomendado para a criação de código personalizado para ser utilizado com o WebSphere Commerce Business Edition é o produto WebSphere Commerce Studio, Business Developer Edition. O pacote de desenvolvimento recomendado para a criação de código personalizado para ser utilizado com o WebSphere Commerce Professional Edition é o produto WebSphere Commerce Studio Professional Developer Edition. Esses pacotes incluem todas as ferramentas que você precisa para criar código personalizado e executar as tarefas de desenvolvimento da Web.

Tanto o WebSphere Commerce Studio Business Developer Edition quanto o WebSphere Commerce Studio Professional Developer Edition fornecem a opção de incluir uma loja de exemplo do WebSphere Commerce que é utilizada no componente WebSphere Test Environment do VisualAge for Java. Isso simplifica a configuração do ambiente de desenvolvimento, já que um desenvolvedor não é obrigado a utilizar as ferramentas do WebSphere Commerce para criar uma loja e depois mover os recursos da loja para o ambiente de desenvolvimento.

Outro componente chave para o ambiente de desenvolvimento é o repositório do código do WebSphere Commerce. Esse repositório precisa ser importado para a área de trabalho do VisualAge for Java, depois da instalação do produto ser concluída. Subseqüente à importação desse repositório, o desenvolvedor precisa executar algumas etapas de configuração relacionadas ao WebSphere Test Environment.

Após todas as tarefas de instalação e configuração serem concluídas, o desenvolvedor tem uma máquina de desenvolvimento independente, na qual o código personalizado do WebSphere Commerce pode ser criado e testado. O desenvolvedor não é obrigado a instalar o WebSphere Commerce na máquina de desenvolvimento.

WebSphere Commerce Studio

O WebSphere Commerce Studio Business Developer Edition e o WebSphere Commerce Studio Professional Developer Edition incluem o VisualAge for Java, Enterprise Edition, Versão 4.0. Essa edição do VisualAge for Java inclui recursos, como ferramentas robustas para desenvolvimento e depuração de modelos JSP avançados que auxiliam no desenvolvimento dos recursos de fachada da loja. Inclui também integração avançada com o trademark="WebSphere Studio para permitir a inclusão rápida de conteúdo nesses modelos JSP, aumentando a produtividade de programadores e desenvolvedores da Web. Para o desenvolvimento do código de aplicativo de back-office, inclui suporte para a tecnologia Enterprise JavaBeans e recursos de conectividade para suportar a integração com outros sistemas, como CICS TS, MQSeries, SAP R/3 e outros. Além disso, o WebSphere Test Environment integrado do VisualAge for Java, Enterprise Edition, permite que os desenvolvedores executem funções do WebSphere Commerce sem ter que sair do VisualAge for Java. Isso significa que testes do código personalizado do WebSphere Commerce podem ocorrer sem implementar o código em um WebSphere Commerce Server.

Nota: O componente WebSphere Test Environment do VisualAge for Java, Enterprise Edition, Versão 4.0 executa aplicativos no WebSphere Application Server V3.5.4. Observe que tanto o WebSphere Commerce Business Edition quanto o WebSphere Commerce Professional Edition utilizam o WebSphere Application Server V4.0. O resultado dessa diferença nas versões é que, antes de implementar beans corporativos novos ou modificados em uma instância do WebSphere Commerce em execução no WebSphere Application Server V4.0, é necessário gerar código implementado fora do VisualAge for Java utilizando a ferramenta EJBDeploy fornecida com o WebSphere Application Server V4.0. Na verdade, esse código implementado deve ser gerado em uma máquina que tenha o WebSphere Application Server V4.0 instalado. Consulte a seção “Informações sobre Código EJB Implementado” na página 194 para obter mais detalhes.

Para concluir os tutoriais descritos em Parte 4, “Tutoriais” na página 205, é necessário instalar o WebSphere Commerce Studio Business Developer Edition ou o WebSphere Commerce Studio Professional Developer Edition. Consulte o *WebSphere Commerce Studio Business Developer Edition - Manual de Instalação* ou o *WebSphere Commerce Studio Professional Developer Edition - Manual de Instalação* para obter mais informações sobre a instalação do Commerce Studio e a configuração do VisualAge for Java.

Características e Funções do VisualAge for Java

O objetivo deste *Manual do Programador* não é ensinar como utilizar o VisualAge for Java. Em relação a este produto, ele mostra, geralmente, como executar uma tarefa no VisualAge for Java mais como uma forma de realizar uma tarefa de programação para a criação de um aplicativo e-commerce para o WebSphere Commerce. Como tal, se você for um novo usuário do VisualAge for Java, começará a aprender como executar algumas tarefas no VisualAge for Java executando os tutoriais contidos neste manual. Entretanto, é necessário que você consulte a documentação, os tutoriais e os métodos do VisualAge for Java para tornar-se perito na utilização desta ferramenta.

Por exemplo, o tópico (Opcional) Utilizando o Debugger no VisualAge for Java do tutorial fornece uma introdução rápida sobre como você pode utilizar o Debugger para exibir o valor de variáveis em um comando de tarefa durante a execução do código. O componente Debugger do VisualAge for Java é uma ferramenta poderosa de depuração e, como tal, você deve consultar a documentação do VisualAge for Java para obter detalhes completos sobre seus recursos e como utilizá-los.

Repositório de Código do WebSphere Commerce

Para criar código personalizado de um aplicativo do WebSphere Commerce, é necessário importar um repositório de código do WebSphere Commerce para a área de trabalho do VisualAge for Java. O repositório está disponível no Disco 2 dos CDs do WebSphere Commerce Business Edition e do WebSphere Commerce Professional Edition. O repositório atual (na época da publicação) é chamado de `WC_54.dat`.

Implementação de Código

Ao personalizar o aplicativo e-commerce, você pode executar qualquer um dos itens a seguir:

- Criar novos comandos, beans de dados ou beans de entidade
- Estender beans de entidade existentes do WebSphere Commerce
- Modificar a lógica de comandos ou beans de dados já existentes

Ao desenvolver o código dentro do VisualAge for Java, você poderá testar seu código dentro do WebSphere Test Environment. Em algum momento, você deve implementar seu código em um WebSphere Commerce Server fora do ambiente de desenvolvimento.

Nas seções seguintes, *WebSphere Commerce Server de destino* refere-se ao WebSphere Commerce Server no qual você está implementando o código personalizado. Em alguns cenários de teste, você pode implementar a um WebSphere Commerce Server que esteja sendo executado na mesma máquina

que o VisualAge for Java. Em outras situações, o WebSphere Commerce Server de destino está em outra máquina e pode até estar sendo executado em uma plataforma diferente.

As seções a seguir descrevem as etapas de alto-nível para implementar os vários tipos de código personalizado. Utilize-as para entender as etapas envolvidas no processo de implementação e consulte o Apêndice B, "Detalhes de Implementação" na página 349 para obter instruções passo a passo.

Além das seções a seguir que descrevem a implementação do código personalizado, se você tiver criado novas políticas de controle de acesso no ambiente de desenvolvimento as mesmas políticas de controle de acesso deverão ser criadas no WebSphere Commerce Server de destino. Consulte "Carregando as Políticas de Controle de Acesso para os Novos Recursos" na página 281 nos tutoriais para obter um exemplo desse procedimento.

Informações sobre Código EJB Implementado

É importante reconhecer que o componente WebSphere Test Environment do VisualAge for Java V4.0 utiliza o WebSphere Application Server V3.5.4. Nesta versão do WebSphere Application Server, os beans corporativos são suportados no nível da especificação EJB (Enterprise JavaBeans) V1.0. Assim, para executar qualquer bean corporativo dentro do WebSphere Test Environment, utilize as ferramentas do VisualAge for Java para gerar código implementado para os beans corporativos que seja compatível com a especificação EJB V1.0.

Por outro lado, tanto o WebSphere Commerce Business Edition quanto o WebSphere Commerce Professional Edition utilizam o WebSphere Application Server V4.0. O WebSphere Application Server V4.0 suporta a especificação EJB V1.1. Desse modo, o código implementado para beans corporativos em execução no WebSphere Test Environment são diferentes do código implementado para beans corporativos em execução no WebSphere Application Server V4.0.

O impacto no desenvolvimento e implementação é o seguinte:

- Para testar beans corporativos dentro do WebSphere Test Environment, utilize as ferramentas do VisualAge for Java para gerar o código implementado que atenda aos requisitos do WebSphere Application Server V3.5.4 utilizado no WebSphere Test Environment. Esse código é gerado clicando com o botão direito do mouse no bean corporativo e selecionando **Gerar Código Implementado**. Observe que isto não cria um arquivo JAR.
- Para implementar beans corporativos em um WebSphere Application Server V4.0, você deve fazer o seguinte:
  1. Utilize as ferramentas do VisualAge for Java para exportar os beans corporativos para o que este documento chama de *Arquivo JAR de*

Exportação EJB 1.1. Esse arquivo JAR empacota o código em um formato utilizado pela ferramenta EJBDeploy. O arquivo é criado clicando com o botão direito do mouse no grupo EJB que contém o bean corporativo a ser implementado e selecionando **Exportar > JAR EJB 1.1**. Observe que ao criar este arquivo JAR, você deve selecionar o tipo de banco de dados apropriado da máquina na qual o código será implementado.

2. Transfira esse arquivo JAR de Exportação EJB 1.1 para um servidor de destino que execute o WebSphere Application Server V4.0.
3. No servidor de destino, execute a ferramenta EJBDeploy com o arquivo JAR de Exportação EJB 1.1 como entrada para criar um novo arquivo JAR de código implementado que seja compatível com a especificação Enterprise JavaBeans V1.1.

Existem outras etapas envolvidas na implementação de beans corporativos. Para obter mais informações, consulte "Implementação de Novos Beans de Entidade" na página 196 e "Implementação de Beans de Entidade Públicos Modificados do WebSphere Commerce" na página 199. Para obter mais informações sobre a utilização da ferramenta EJBDeploy, consulte "Gerando Código Implementado" na página 360.

Implementação de Novos Comandos e Beans de Dados

Ao criar novos comandos e beans de dados, você deverá colocá-los em um pacote que esteja nomeado apropriadamente para seu aplicativo. Por exemplo, você poderia criar um novo pacote com o nome de `com.mycompany.mycommands` no qual manteria os novos comandos. Este pacote deve ser armazenado dentro de um projeto que esteja separado dos projetos do WebSphere Commerce (IBM WC Commerce Server e IBM WC Enterprise Beans). Por exemplo, poderia criar um novo projeto chamado `My Project`. Exige-se uma organização cuidadosa do código para garantir que a implementação seja bem-sucedida.

Com todos os novos comandos e beans de dados armazenados em seu próprio projeto, a implementação consistirá nas seguintes etapas de alto nível:

1. Utilizando a máquina de desenvolvimento, crie um arquivo JAR para o projeto. A partir da página Projeto no VisualAge for Java, utilize as ferramentas para exportar o projeto para um arquivo JAR. Nomeie o arquivo JAR adequadamente para seu código, por exemplo, `CustomCommands.jar`. Além disso, você deve utilizar novamente jar para o arquivo JAR utilizando ferramentas fora do VisualAge for Java para verificar se todas as informações de nomenclatura requeridas estão incluídas para implementação no WebSphere Application Server. Consulte "Arquivos JAR para Comandos Personalizados e Beans de Dados" na página 349 para obter mais informações.
2. Copie o arquivo JAR, modelos JSP e qualquer outro recurso de loja para o diretório apropriado no WebSphere Commerce Server de destino. Consulte

"Armazenando Recursos no WebSphere Commerce Server de Destino" na página 356 para obter mais informações.

3. Registre o novo comando no registro de comandos no WebSphere Commerce Server de destino. Consulte "Estrutura do Registro de Comandos" na página 28 para obter mais informações.
4. Pare e inicie o aplicativo corporativo do WebSphere Commerce novamente, utilizando o Console do Administrador do WebSphere Application Server. Consulte o *Manual de Instalação do WebSphere Commerce* apropriado para obter mais informações sobre o início e parada deste aplicativo.

Implementação de Novos Beans de Entidade

Ao criar novos beans de entidade, você deve criá-los em um grupo de EJB que seja separado dos grupos de EJB que contêm os beans de entidade do WebSphere Commerce. Você também deve utilizar seu próprio projeto e assegurar que esse projeto não contenha o código de nenhum comando ou bean de dados. Por exemplo, você poderia criar um novo grupo EJB chamado `MyEntityBeans` e um projeto com o nome de `MyEntityBeansProject`. Se o projeto contiver código diferente do código para novos beans de entidade, a implementação poderá não ser bem-sucedida.

Uma vez satisfeito com a maneira que seu bean de entidade está funcionando dentro do WebSphere Test Environment, você deverá implementá-lo. As informações a seguir fornecem uma visão geral das etapas de implementação:

1. Utilizando a máquina de desenvolvimento, crie um novo arquivo JAR de Exportação EJB 1.1 para o novo grupo EJB. Na página EJB no VisualAge for Java, clique com o botão direito do mouse no novo grupo de EJB e selecione para exportar o código para um arquivo JAR EJB 1.1. Nomeie o arquivo adequadamente para seu código, por exemplo, `MyEntityBeans_DT.jar`. Consulte "Criando Arquivos JAR para Novos Beans de Entidade" na página 351 para obter mais informações.
2. Crie um novo arquivo JAR de implementação para o novo projeto EJB. Para criar este arquivo JAR, selecione a página Projeto e em seguida, clique com o botão direito do mouse no novo projeto EJB e selecione exportar o código para um arquivo JAR. Nomeie o arquivo adequadamente para seu código, por exemplo, `MyEntityBeansImpl.jar`. Além disso, você deve utilizar novamente jar para o arquivo JAR de implementação utilizando ferramentas fora do VisualAge for Java para verificar se todas as informações de nomenclatura requeridas estão incluídas para implementação no WebSphere Application Server. Consulte "Criando Arquivos JAR para Novos Beans de Entidade" na página 351 para obter mais informações.
3. Copie os arquivos JAR no diretório apropriado no WebSphere Commerce Server de destino. Consulte "Armazenando Recursos no WebSphere Commerce Server de Destino" na página 356 para obter mais informações.

4. No WebSphere Commerce Server de destino, utilize a Ferramenta de Implementação de EJB fornecida pelo WebSphere Application Server para gerar o código implementado para os novos beans corporativos. Essa ferramenta pega o arquivo JAR criado na etapa 1 como entrada e cria o arquivo JAR correspondente contendo o código implementado para todos os beans corporativos no grupo de EJB. Consulte "Gerando Código Implementado" na página 360 para obter mais informações.
5. No WebSphere Commerce Server de destino, modifique o nível de isolamento de transação dos beans corporativos que estão contidos no arquivo JAR de código implementado. Use o utilitário `modifyIsolationLevel` da linha de comandos fornecido pelo WebSphere Commerce para definir o nível de isolamento de transação para o nível apropriado de seu tipo de banco de dados. Consulte "Modificando o Nível de Isolamento de Transação dos Beans de Entidade" na página 363 para obter mais informações.
6. No WebSphere Commerce Server de destino, carregue as políticas de controle de acesso dos novos recursos criados. Utilize os comandos `acpload` e `acpnlsload` do WebSphere Commerce para carregar as informações sobre a política. Consulte a seção "Carregando as Políticas de Controle de Acesso para os Novos Recursos" na página 281 no tutorial Capítulo 9, "Tutorial: Criando Nova Lógica de Negócios" para obter um exemplo de como carregar políticas de controle de acesso para novos recursos.
7. No WebSphere Commerce Server de destino, exporte o aplicativo corporativo atual do WebSphere Commerce do WebSphere Application Server. Depois de concluir a exportação, um arquivo .ear é criado contendo todo o aplicativo. Para exportar o aplicativo atual, abra o WebSphere Application Server Administration Console, selecione o aplicativo corporativo do WebSphere Commerce e selecione a opção export. Na conclusão, um arquivo `WC_Enterprise_App_`*`instanceName`*`.ear` é criado. Consulte "Exportando o Aplicativo Corporativo Atual do WebSphere Commerce" na página 365 para obter mais informações.
8. No WebSphere Commerce Server de destino, exporte as informações de configuração dos beans corporativos contidos no aplicativo corporativo atual do WebSphere Commerce em execução no WebSphere Application Server. Essas informações são exportadas para um arquivo XML utilizando a opção `-export` do utilitário `XMLConfig` da linha de comandos, fornecido pelo WebSphere Application Server. O arquivo XML gerado (referido aqui como o arquivo `OutputFile.xml`) contém uma sub-rotina de informações de configuração para cada bean corporativo contido no aplicativo corporativo. Você deve em seguida, incluir uma nova sub-rotina neste arquivo para cada novo bean corporativo que estiver sendo implementado. Consulte "Exportando Informações de Configuração dos Beans Corporativos" na página 366 para obter mais informações.

9. Inclua o novo bean ou beans corporativos no aplicativo corporativo utilizando a Ferramenta de Montagem de Aplicativos fornecida pelo WebSphere Application Server. Com essa ferramenta, você pode abrir o `WC_Enterprise_App_`*`instanceName`*`.ear` do aplicativo atual e depois importar novos beans corporativos para o aplicativo. Você também define o caminho de classe do novo bean ou beans corporativos para que inclua qualquer arquivo JAR dependente e o arquivo JAR de implementação como um arquivo no aplicativo. Finalmente, você utiliza essa ferramenta para configurar a segurança do WebSphere Application Server para métodos contidos no bean ou beans corporativos. Depois de concluir essas etapas, salve o aplicativo e um novo arquivo `.ear` será criado para o aplicativo corporativo. Consulte "Montando Novos Beans Corporativos em um Aplicativo Corporativo" na página 372 para obter mais informações.
10. Importe o novo aplicativo corporativo para o WebSphere Application Server. Esta etapa é composta das seguintes três subtarefas:
    a. Utilizando o WebSphere Application Server Administration Console, pare e remova o aplicativo corporativo original do WebSphere Commerce. Consulte "Parando e Removendo um Aplicativo Corporativo" na página 381 para obter mais informações.
    b. Utilizando a opção `-import` do utilitário `XMLConfig` da linha de comandos para importar o novo aplicativo corporativo para o WebSphere Application Server. Consulte "Importando um Aplicativo Corporativo" na página 382 para obter mais informações.
    c. Utilizando o WebSphere Application Server Administration Console, atualize a exibição para mostrar o novo aplicativo corporativo e depois inicie-o. Consulte "Iniciando um Aplicativo Corporativo" na página 384 para obter mais informações.

Implementação de Extensões para Comandos e Beans de Dados Existentes

O método para estender comandos existentes depende do tipo de modificação exigida. Os métodos para extensão são explicados em "Personalizando os Comandos já Existentes" na página 145. Em geral, a modificação de lógica já existente envolve a criação de uma nova classe que herda da classe que requer personalização. Substitua os métodos da superclasse conforme solicitado para substituir ou modificar a lógica.

Ao personalizar os beans de dados, você também cria uma nova classe que estende um bean de dados existente. Na nova classe, faça as modificações necessárias.

Ao criar essas novas classes, certifique-se de que elas estejam armazenadas em um de seus próprios pacotes, que está armazenado dentro de um de seus próprios projetos.

Visto que as extensões são efetivamente manuseadas por subclasses, a implementação de extensões para comandos e beans de dados é o mesma que a implementação de novos comandos e beans de dados. Para obter mais informações, consulte “Implementação de Novos Comandos e Beans de Dados” na página 195.

Implementação de Beans de Entidade Públicos Modificados do WebSphere Commerce

Ao modificar um bean corporativo público do WebSphere Commerce, você modifica o código do WebSphere Commerce. A técnica de implementação para beans de entidade personalizados é, portanto, ligeiramente diferente da utilizada para novos beans de entidade. As informações a seguir fornecem uma visão geral das etapas de implementação:

1. Crie um arquivo JAR de Exportação EJB 1.1 para o grupo EJB do WebSphere Commerce que contém o bean de entidade modificado. Com a guia EJB no Workbench do VisualAge for Java selecionada, selecione o grupo EJB apropriado e em seguida, selecione a exportação desse grupo para um arquivo JAR de Exportação EJB 1.1. Ao nomear o arquivo JAR, você pode utilizar a convenção de nomenclatura `Cust_`*`EJBGroupName`*`-ejb_DT.jar`, em que *`EJBGroupName`* é o nome do grupo EJB modificado e o sufixo _DT é anexado ao fim do nome do arquivo JAR. Por exemplo, ao nomear o arquivo JAR para o grupo EJB WCSUser, nomeie o arquivo como `Cust_WCSUser-ejb_DT.jar`. Observe que o sufixo _DT é utilizado simplesmente como lembrete de que posteriormente esse arquivo JAR deverá ser transmitido para a Ferramenta EJBDeploy a fim de gerar o código implementado para os beans contidos no grupo EJB. Consulte “Criando Arquivos JAR para Beans de Entidade do WebSphere Commerce Personalizados” na página 353 para obter mais informações.
2. Crie um novo arquivo JAR cliente que contenha o código cliente para todos os grupos de EJB do WebSphere Commerce. Com todos os grupos de EJB do WebSphere Commerce selecionados (todos os nomes começam com `WCS`), selecione para exportar para um arquivo JAR cliente. Consulte “Criando Arquivos JAR para Beans de Entidade do WebSphere Commerce Personalizados” na página 353 para obter mais informações.
3. Copie os arquivos JAR no diretório apropriado no WebSphere Commerce Server de destino. Consulte “Armazenando Recursos no WebSphere Commerce Server de Destino” na página 356 para obter mais informações.
4. No WebSphere Commerce Server de destino, utilize a Ferramenta JBDeploy fornecida pelo WebSphere Application Server para gerar o código implementado para os beans corporativos contidos no grupo EJB que está sendo implementado. Essa ferramenta pega o arquivo JAR criado na etapa 1 como entrada e cria o arquivo JAR correspondente contendo o código implementado para todos os beans corporativos no grupo de EJB. Consulte “Gerando Código Implementado” na página 360 para obter mais informações.

5. No WebSphere Commerce Server de destino, modifique o nível de isolamento de transação dos beans corporativos que estão contidos no arquivo JAR de código implementado. Use o utilitário `modifyIsolationLevel` da linha de comandos fornecido pelo WebSphere Commerce para definir o nível de isolamento de transação para o nível apropriado de seu tipo de banco de dados. Consulte "Modificando o Nível de Isolamento de Transação dos Beans de Entidade" na página 363 para obter mais informações.
6. No WebSphere Commerce Server de destino, exporte o aplicativo corporativo atual do WebSphere Commerce do WebSphere Application Server. Depois de concluir a exportação, um arquivo .ear é criado contendo todo o aplicativo. Para exportar o aplicativo atual, abra o WebSphere Application Server Administration Console, selecione o aplicativo corporativo do WebSphere Commerce e selecione a opção export. Na conclusão, um arquivo `WC_Enterprise_App_`*`instanceName`*`.ear` é criado. Consulte "Exportando o Aplicativo Corporativo Atual do WebSphere Commerce" na página 365 para obter mais informações.
7. No WebSphere Commerce Server de destino, exporte as informações de configuração dos beans corporativos contidos no aplicativo corporativo atual do WebSphere Commerce em execução no WebSphere Application Server. Essas informações são exportadas para um arquivo XML utilizando a opção `-export` do utilitário `XMLConfig` da linha de comandos, fornecido pelo WebSphere Application Server. O arquivo XML gerado (referido aqui como o arquivo `OutputFile.xml`) contém uma sub-rotina de informações de configuração para cada bean corporativo contido no aplicativo corporativo. Consulte "Exportando Informações de Configuração dos Beans Corporativos" na página 366 para obter mais informações. Dentro desse arquivo, modifique a sub-rotina que descreve o bean corporativo modificado para que indique o novo arquivo `Cust_`*`EJBGroupName`*`-ejb.jar`.
8. No WebSphere Commerce Server de destino, utilize a ferramenta Ferramenta de Montagem de Aplicativos para integrar o grupo EJB modificado no aplicativo corporativo. Com essa ferramenta, você abre o arquivo `.ear` do aplicativo atual. Depois de aberto, é possível executar as seguintes tarefas:
   a. Anotar as informações de caminho da classe da versão original do grupo de EJB modificado. Por exemplo, se você modificou o bean de Usuário no grupo de EJB WCSUser, copie as informações do caminho da classe do grupo WCSUser para um arquivo de texto.
   b. Excluir a versão original do grupo de EJB modificado (por exemplo, excluir o grupo WCSUser).
   c. Importar a nova versão do grupo de EJB que você modificou.
   d. Aplicar as informações originais do caminho de classe no grupo de EJB recém-importado.

e. Configurar a segurança do WebSphere Application Server para os métodos contidos em todos os beans corporativos no grupo EJB modificado.

f. Salvar o aplicativo em um novo arquivo `.ear`

Consulte "Montando Beans Corporativos Modificados em um Aplicativo Corporativo" na página 377 para obter mais informações.

9. Importe o novo aplicativo corporativo para o WebSphere Application Server. Esta etapa é composta das seguintes três subtarefas:

   a. Utilizando o WebSphere Application Server Administration Console, pare e remova o aplicativo corporativo original do WebSphere Commerce. Consulte "Parando e Removendo um Aplicativo Corporativo" na página 381 para obter mais informações.

   b. Utilizando a opção `-import` do utilitário `XMLConfig` da linha de comandos para importar o novo aplicativo corporativo para o WebSphere Application Server. Consulte "Importando um Aplicativo Corporativo" na página 382 para obter mais informações.

   c. Utilizando o WebSphere Application Server Administration Console, atualize a exibição para mostrar o novo aplicativo corporativo e depois inicie-o. Consulte "Iniciando um Aplicativo Corporativo" na página 384 para obter mais informações.

Implementação de Novos Beans de Dados para Serem Utilizados no Commerce Studio

Se você estiver utilizando o Commerce Studio para desenvolver seus modelos JSP, será necessário implementar seus novos beans de dados no Commerce Studio. Em particular, você deve criar um arquivo JAR para os beans de dados.

Crie esse arquivo JAR clicando com o botão direito do mouse em seu pacote de beans de dados e selecionando para exportá-los para um arquivo JAR. Nas opções, selecione incluir o seguinte:

- **classe**
- **recurso**
- **beans**

Além disso, selecione **Selecionar tipos e recursos mencionados** para incluir os comandos e recursos que são exigidos pelos beans de dados.

Após ter exportado o arquivo JAR, será necessário atualizar o caminho da classe para que o Commerce Studio inclua esse arquivo JAR.

Quando precisar modificar um bean de dados do WebSphere Commerce já existente, você deverá criar um novo bean de dados que estenda o bean de dados que requer personalização. Portanto, você está realmente criando um novo bean de dados e a implementação será conforme descrito nesta seção.

Implementação de Beans de Entidade Públicos Personalizados para Serem Utilizados no Commerce Studio

Se você modificar qualquer um dos beans de entidade públicos e utilizar o Commerce Studio para o desenvolvimento do modelo JSP, será necessário criar um novo arquivo JAR do cliente para todos os beans de entidade públicos e modificar o caminho da classe para o Commerce Studio a fim de incluir o novo nome do arquivo JAR antes do nome do arquivo JAR original. O Commerce Studio utiliza o arquivo JAR do cliente quando os beans de dados são utilizados na ferramenta Page Designer.

Para criar esse arquivo JAR, use as ferramentas em VisualAge for Java. Da página EJB em VisualAge for Java, selecione *todos* os grupos EJB do WebSphere Commerce (não apenas aqueles modificados) e selecione para exportá-los para um arquivo JAR do cliente. Após a conclusão da exportação, certifique-se de que você especificou esse novo arquivo JAR no início do caminho da classe para o Commerce Studio.

Arquivos de Log

Durante os estágios de instalação do produto, o desenvolvimento do código e a implementação do código, são gerados arquivos de log. Esta seção lista alguns dos arquivos de log que podem ser consultados durante esses estágios.

Arquivos de log do Commerce Studio

O Commerce Studio constrói arquivos de log durante o processo de instalação. Para acessar esses logs, abra o seguinte arquivo:

```
C:\Winnt\WCStudioInstall.log
```

Logs para a configuração do Commerce Studio do WebSphere Test Environment

Uma série de etapas de configuração para o WebSphere Test Environment são executadas durante a instalação do WebSphere Commerce Studio, se você selecionar que deseja executar tarefas de desenvolvimento de back-end. Por exemplo, se você selecionar para incluir uma loja de exemplo que seja executada no componente WebSphere Test Environment do VisualAge for Java, serão criados arquivos de log relacionados ao processo de carregamento em massa e à criação do banco de dados. Para acessar esses logs, navegue para o seguinte diretório:

unidade:\WebSphere\CommerceServerDev\instances*instance_name*\logs

Arquivos de log da execução de componentes do WebSphere Commerce no WebSphere Test Environment

Ao executar uma loja, ou testar um componente individual no WebSphere Test Environment, poderão ser gerados um arquivo de rastreio e um arquivo de mensagens. A localização padrão desses arquivos está no seguinte diretório:

unidade:\WebSphere\CommerceServerDev\instances*instance_name*\logs

Logs da execução do comando modifyIsolationLevel

Ao implementar beans corporativos, utilize o comando modifyIsolationLevel para modificar o nível de isolamento de transação de cada bean corporativo no arquivo JAR. Os arquivos de log gerados são armazenados no arquivo de log especificado durante a execução do comando. Consulte "Modificando o Nível de Isolamento de Transação dos Beans de Entidade" na página 363 para obter mais informações.

Registrando no log do VisualAge for Java

Executando código no WebSphere Test Environment, a janela Console é executada como um log ativo. Além disso, você pode modificar o nível de rastreio de um servidor EJB para aumentar a quantidade de informações que podem ser encontradas na janela Console. Essas informações são especificadas como o nível de rastreio dentro das propriedades do servidor EJB.

Método de Pagamento de Teste

A loja de exemplo InFashion em execução no WebSphere Test Environment utiliza um método de pagamento de teste, por padrão. Esse método de pagamento de teste foi incluído para que você possa concluir o fluxo de compras no WebSphere Test Environment sem precisar pedir ajuda a um Payment Manager. Pedidos enviados utilizando esse método de pagamento têm o status 'M' (indicando que o pagamento foi iniciado e está aguardando processamento).

Esse método de pagamento de teste permite que você conclua apenas uma compra, ele não permite que pedidos submetidos com esse método de pagamento fiquem disponíveis para processamento posterior. Dessa forma o método de pagamento de teste deve ser utilizado apenas no WebSphere Test Environment.

As classes de implementação para comandos relacionados ao pagamento são tipos de comandos de política de negócios. Como tal, a seleção de qual classe de implementação será utilizada é orientado por uma política de negócios. Por padrão, a loja de exemplo InFashion em execução com o WebSphere Test

Environment inclui uma política de negócios (política TestPaymentMethod) que utiliza as seguintes implementações de comandos relacionados ao pagamento:

- com.ibm.commerce.payment.commands.DoPayment**Test**CmdImpl
- com.ibm.commerce.payment.commands.CheckPaymentAccept**Test**CmdImpl
- com.ibm.commerce.payment.commands.DoCancel**Test**CmdImpl
- com.ibm.commerce.payment.commands.DoDeposit**Test**CmdImpl
- com.ibm.commerce.payment.commands.DoRefund**Test**CmdImpl

Deve-se enfatizar que esses comandos são fornecidos para testar apenas o processo de registro de saída. Enquanto o conjunto de comandos anterior é fornecido com a finalidade de complementação, o DoDepositTestCmdImpl é um comando stub que emite uma exceção quando chamado. Não tente utilizar nenhuma função de gerenciamento de pedido com esses pedidos. Para que obtenha a capacidade de testar essas outras funções, configure o WebSphere Test Environment para utilizar um Payment Manager remoto.

É muito importante que ao implementar seu código para o WebSphere Commerce Server de destino você não copie a política de negócios relacionada ao método de pagamento de teste do desenvolvimento para a máquina de destino. Além disso, não registre os comandos relacionados ao método de pagamento de teste no WebSphere Commerce Server de destino.

Consulte o capítulo "Configurando o VisualAge for Java" no *Manual de Instalação do WebSphere Commerce Business Edition* ou no *Manual de Instalação do WebSphere Commerce Professional Edition* para obter informações sobre como desativar o método de pagamento de teste.

Utilizando um Payment Manager Remoto

Se seu trabalho de desenvolvimento incluir trabalhar com pedidos depois que foram feitos, por exemplo, fazer uma modificação em uma etapa no processo de gerenciamento de pedidos, configure a loja em execução no WebSphere Test Environment para utilizar um Payment Manager remoto. Consulte o *WebSphere Commerce Studio, Business Developer Edition - Manual de Instalação* para obter as etapas de configuração detalhadas. Esse documento também fornece informações sobre a remoção de pedidos feitos com o método de pagamento de teste a partir do banco de dados.

Parte 4. Tutoriais

Essa seção contém tutoriais que o ajudam a familiarizar-se com a personalização de seu aplicativo e-commerce. São fornecidos os seguintes tutoriais:

- Criando nova lógica de negócios
  Esse tutorial demonstra o processo de desenvolvimento para criar lógica de negócios. Ele inclui as seguintes sub-tarefas:
  - Preparar o exemplo do projeto
  - Escrever comandos novos
  - Criar novo bean de dados e obtenção de propriedades do pedido
  - Utilizar beans de entidade
  - Criar um novo bean corporativo
- Atualizando negócios lógicos existentes
  Este tutorial demonstra o processo de desenvolvimento de atualização de lógica de negócios do WebSphere Commerce existente. Ele inclui as seguintes sub-tarefas:
  - Estender um comando de controlador existente do WebSphere Commerce
  - Modificar um bean de entidade público existente do WebSphere Commerce
  - Criar um novo comando de tarefa que estende um comando de tarefa existente do WebSphere Commerce.

Os tutoriais contêm seções de código que precisam ser inseridas no VisualAge for Java. Recomenda-se que você obtenha uma versão PDF deste documento nos seguintes sites na Web:

Business

```
http://www.ibm.com/software/webservers/commerce/wc_be
   /lit-tech-general.html
```

Professional

```
http://www.ibm.com/software/webservers/commerce/wc_pe
   /lit-tech-general.html
```

Você pode copiar e colar os trechos de códigos da versão PDF do Manual do Programador.

Capítulo 9. Tutorial: Criando Nova Lógica de Negócios

Ambiente do Tutorial

Os tutoriais contidos neste manual requerem que você tenha o WebSphere Commerce Business Edition V5.4 ou WebSphere Commerce Professional Edition V5.4 instalado. Além disso, ao instalar o produto, você deve optar por instalar as seguintes opções:

- **Desenvolver recursos de fachada da loja utilizando o WebSphere Studio**
- **Desenvolver lógica de backend da loja utilizando o VisualAge for Java**
- **Criar banco de dados**
- **Incluir loja de exemplo**

Siga as instruções contidas no *WebSphere Commerce Studio, Business Developer Edition - Manual de Instalação* ou no *WebSphere Commerce Studio Professional Developer Edition - Manual de Instalação* para obter informações completas sobre instalação e configuração.

Antes de iniciar os tutoriais, você deve estar habilitado a executar a loja de exemplo no WebSphere Test Environment e concluir uma compra na loja.

Etapas de Implementação do Código do Tutorial

Os tutoriais contidos neste manual incluem etapas opcionais que descrevem como implementar o código personalizado para um WebSphere Commerce Server de destino. Presume-se que o WebSphere Commerce Server de destino esteja sendo executado no Windows NT ou no Windows 2000 em uma máquina que esteja separada do ambiente de desenvolvimento. Observe que, se estiver utilizando o WebSphere Commerce Studio Business Developer Edition como seu ambiente de desenvolvimento, deverá implementar no WebSphere Commerce Business Edition. Se estiver utilizando o WebSphere Commerce Studio Professional Developer Edition como ambiente de desenvolvimento, deverá implementar no WebSphere Commerce Professional Edition.

Além disso, no WebSphere Commerce Server de destino, você deve ter publicado uma loja com base na loja de exemplo InFashion. Nessa loja, deverá ser possível concluir uma compra. É possível utilizar um Payment Manager remoto ou local para o processamento do pagamento.

Se estiver implementando um WebSphere Commerce Server de destino que esteja na mesma máquina que o ambiente de desenvolvimento ou que esteja

sendo executado em um sistema operacional diferente, consulte Capítulo 8, “Ferramentas de Desenvolvimento e Implementação” na página 191 e Apêndice B, “Detalhes de Implementação” na página 349 para obter as informações apropriadas sobre implementação.

Preparando Amostra de Projetos

Nesta seção você importará o repositório `WC_SAMPLE_54.dat` que contém o ponto inicial para o tutorial “Criando nova lógica de negócios”.

Para preparar seu ambiente, proceda da seguinte forma:

1. Assegure-se de que você esteja utilizando o repositório `WC_54.dat` do código do WebSphere Commerce. Ele está disponível no Disco 2 do CD do WebSphere Commerce Business Edition V5.4 ou no Disco 2 do CD do WebSphere Commerce Professional Edition V5.4.
2. Importe a amostra do projeto para a sua área de trabalho, executando o seguinte
   a. Insira o seguinte CD na unidade de CD na máquina de desenvolvimento:
      - Business WebSphere Commerce Business Edition, V5.4 Disco 2
      - Professional WebSphere Commerce Professional Edition, V5.4 Disco 2
   b. Abra o VisualAge for Java.
   c. Do menu **Arquivo**, selecione **Importar**
      O SmartGuide Importar é aberto.
   d. Selecione importar um **Repositório** e clique em **Avançar**.
   e. Em Importar de outra janela de repositório, faça o seguinte:
      1) Selecione **Repositório local**.
      2) No campo **Nome do repositório**, digite *CD_drive*`:\repository\samples\programguide\WC_SAMPLE_54.dat` em que *CD_drive* é a unidade do CD.
      3) Selecione **Projetos** e clique em **Detalhes**. Selecione o projeto **_WCSamples**. Selecione a versão **WC Sample 5.4** e clique em **OK**.
      4) Assegure-se de que **Incluir a edição do projeto mais recente na área de trabalho** esteja selecionado.
      5) Clique em **Concluir** para iniciar a importação.
         A importação do projeto pode levar vários minutos.
3. Verifique se o proprietário da área de trabalho está definido como WCS Developer, da seguinte forma:
   a. Do menu **Área de Trabalho**, selecione **Alterar Proprietário da Área de Trabalho**.
   b. Selecione **WCS Developer** e clique em **OK**.

4. Copie os modelos JSP dos exercícios no diretório apropriado, dessa forma, eles poderão ser utilizados no WebSphere Test Environment. Para copiar estes arquivos, faça o seguinte:
   a. Localize os arquivos `Sample.jsp` e `Sample_All.jsp` no seguinte diretório
      CD_drive`:\repository\samples\programguide\`

      em que *CD_drive* é a unidade de CD para o seguinte diretório:
   b. Copie esses dois arquivos para o seguinte diretório:
      vaj_drive`:\VAJava\ide\project_resources\IBM WebSphere Test Environment \hosts\default_host\default_app\web`

      em que *vaj_drive* é a unidade na qual você instalou o VisualAge for Java.
5. Copie os arquivos da política de controle de acesso para o diretório apropriado. Esses arquivos são utilizados para carregar novas políticas de controle de acesso para os novos recursos que forem criados durante os tutoriais. Para copiar estes arquivos, faça o seguinte:
   a. Vá para o seguinte diretório:
      CD_drive`:\repository\samples\programguide\`
   b. Localize os seguintes arquivos nesse diretório:
      - `SampleCmdACPolicy.xml`
        Esse arquivo XML contém a política de controle de acesso que é utilizada quando você cria um novo comando de controlador.
      - `SampleACPolicy.xml`
        Esse arquivo XML contém a política de controle de acesso que é utilizada quando você cria um novo bean corporativo.
      - `SampleACPolicy_`*locale*`.xml`
        em que *locale* é o identificador do idioma. Esse arquivo XML contém a descrição da política de controle de acesso.
   c. Copie os arquivos anteriores para o seguinte diretório:
      unidade`:\WebSphere\CommerceServerDev\xml\policies\xml`
      em que *unidade* é a unidade em que você instalou o WebSphere Commerce Studio.
6. Teste seu ambiente para assegurar que você está pronto para iniciar os tutoriais, fazendo o seguinte:
   a. Inicie o servidor de nomes persistente, conforme descrito na página 347.
   b. Inicie o servidor EJB, conforme descrito na página 348.
   c. Inicie o mecanismo servlet, conforme descrito na página 348
   d. Abra um navegador e digite a seguinte URL:
      `http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?storeId=`*store_ID*`&catalogId=`*catalog_Id*`&langId=-1`

em que *store_ID* é o identificador para seu exemplo de loja (10001 é valor de exemplo) e *catalog_Id* é o identificador para seu catálogo de exemplo de loja (10001 é um valor de exemplo).
Quando a home page do exemplo de loja é exibida, selecione um produto e compre-o. Você precisa chegar à página "Confirmação do pedido" no fluxo de compras.

Para verificar o valor storeId de sua loja, consulte a tabela STOREENT.

e. Feche todos os navegadores e pare o WebSphere Test Environment.

Agora você está pronto para iniciar os tutoriais.

Escrevendo Comandos

Escrever um Comando do Controlador

Essa seção mostra como escrever um novo comando do controlador. Após a conclusão deste exercício, você terá um novo comando, chamado *MyNewControllerCmd*. Esse comando é utilizado por todas as lojas e cada loja utiliza a mesma implementação do comando.

Existem três etapas básicas ao criar um novo comando do controlador.

1. Registre o comando na estrutura de registro de comandos.
2. Crie a interface para o comando.
3. Crie a classe de implementação do comando.

Para obter mais informações sobre comandos, consulte "Padrão de Design do Comando" na página 22.

Antes de iniciar este tutorial
É necessário já ter concluído as etapas em "Preparando Amostra de Projetos" na página 208.

Para escrever seu novo comando, faça o seguinte:

1. A primeira etapa para escrever um comando do controlador é estabelecer um nome para seu comando. Nesse exemplo, o comando é chamado *MyNewControllerCmd*.
2. O comando deve ser registrado na estrutura de registro de comandos. O processo de registro para o novo comando de controlador envolve criar

uma entrada na tabela URLREG.

DB2 Se estiver utilizando um banco de dados DB2, faça o seguinte para registrar o comando:

a. Abra o Centro de Comandos do DB2 (**Iniciar > Programas >IBM DB2 > Centro de Comandos**).
b. No menu **Ferramentas**, selecione **Definições de Ferramentas**.
c. Selecione a caixa de opções **Utilizar caractere de finalização de instrução** e certifique-se de que o caractere especificado é um ponto e vírgula (**;**)
d. Com a guia Script selecionada, crie a entrada requerida na tabela URLREG, digitando as seguintes informações na janela de script

```
connect to your_database_name;
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS,
    DESCRIPTION, AUTHENTICATED) values ('MyNewControllerCmd',0,
    'com.ibm.commerce.sample.commands.MyNewControllerCmd',0,
    'This is a new controller command for test/education purposes.',
    null)
```

em que *your_database_name* é o nome do banco de dados, e clique no ícone Executar.
Esse comando é utilizado por todos os comerciantes (indicados pelo valor 0 para STOREENT_ID).

Oracle Se você estiver utilizando um banco de dados Oracle, faça o seguinte para registrar seu comando:

a. Abra a janela de comando Oracle SQL Plus (**Iniciar > Programas > Oracle > Desenvolvimento de Aplicativos > SQL Plus**).
b. No campo **Nome do Usuário**, digite seu nome de usuário do Oracle.
c. No campo **Senha**, digite a senha do Oracle.
d. No campo **Cadeia do Host**, digite sua cadeia de conexão.
e. Na janela SQL Plus, digite o seguinte para criar a entrada obrigatória na tabela URLREG:

```
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS,
    DESCRIPTION, AUTHENTICATED) values ('MyNewControllerCmd',0,
    'com.ibm.commerce.sample.commands.MyNewControllerCmd',0,
    'This is a new controller command for test/education purposes.',
    null);
```

e pressione Enter para executar a instrução SQL.
f. Digite o seguinte para consolidar as alterações no banco de dados:

```
commit;
```

e pressione Enter para executar a instrução SQL.

Nota: Para que esse exercício seja simples, o novo comando possui apenas uma classe de implementação, ou seja, a mesma classe de implementação é utilizada em toda as lojas. Essa classe de implementação é especificada diretamente no código para a interface. Assim, não há necessidade de registrar o mapeamento entre a interface e a classe de implementação na tabela CMDREG. Fora do ambiente de um tutorial, você deve registrar o comando do controlador na tabela CMDREG, bem como na tabela URLREG.

3. Os comandos de controlador precisam retornar uma exibição. O novo comando de controlador que você criará retorna a exibição SampleViewTask. É necessário registrar a exibição SampleViewTask na tabela VIEWREG.

   DB2 Se estiver utilizando um banco de dados DB2, faça o seguinte para registrar a exibição:

   a. Crie uma entrada na tabela VIEWREG, digitando o seguinte na janela de script (observe que você pode precisar limpar a instrução SQL anterior da primeira janela):

      ```
      insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID,
         INTERFACENAME,CLASSNAME, PROPERTIES, DESCRIPTION,
             HTTPS, LASTUPDATE)
      values ('SampleViewTask',-1, 0,
         'com.ibm.commerce.command.ForwardViewCommand',
         'com.ibm.commerce.command.HttpForwardViewCommandImpl',
         'docname=Sample.jsp','This is a sample view for the
          Bonus Point exercise', 0, null)
      ```

      e clique no ícone Executar.

   Oracle Se você estiver utilizando um banco de dados Oracle, faça o seguinte para registrar sua exibição no banco de dados:

   a. Na janela SQL Plus, digite a seguinte instrução SQL:

      ```
      insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID,
         INTERFACENAME,CLASSNAME, PROPERTIES, DESCRIPTION,
             HTTPS, LASTUPDATE)
      values ('SampleViewTask',-1, 0,
         'com.ibm.commerce.command.ForwardViewCommand',
         'com.ibm.commerce.command.HttpForwardViewCommandImpl',
         'docname=Sample.jsp','This is a sample view for the
         Bonus Point exercise', 0, null);
      ```

      e pressione Enter para executar a instrução SQL.

   b. Digite o seguinte para consolidar as alterações no banco de dados:

      ```
      commit;
      ```

      e pressione Enter para executar a instrução SQL.

4. Na janela Workbench do VisualAge for Java, expanda o projeto **_WCSamples**.
5. Expanda o pacote **com.ibm.commerce.sample.commands**, clique com o botão direito do mouse a interface **MyNewControllerCmd** e selecione **Incluir > Campo**.
   O SmartGuide Criar Campo é aberto.
6. Crie um campo que especifique a classe de implementação padrão para a interface, fazendo o seguinte:
   a. No campo **Nome do Campo**, digite `defaultCommandClassName`.
   b. Na lista suspensa **Tipo de Campo**, selecione `Cadeia`.
   c. No campo Valor Inicial, digite `"com.ibm.commerce.sample.commands.MyNewControllerCmdImpl"`. (Assegure-se de ter incluído aspas duplas).
   d. Clique em **Concluir**.
      O código para o novo campo é gerado.

Em qualquer ponto deste tutorial, quando você substituir um campo ou método que exista na superclasse, poderá ser exibido um aviso indicando que o novo campo ou método ocultará um campo ou método herdado. Se sim, clique em **Sim** para continuar.

7. Expanda a classe **MyNewControllerCmdImpl** e selecione seu método **performExecute** para exibir seu código fonte.
8. No código fonte para o método performExecute, remova o comentário da Seção 1 e da Seção 5. A Seção 1 introduz o seguinte código no método:

```
// Create a new TypedProperties for output.
  TypedProperty rspProp = new TypedProperty();
```

   A Seção 5 introduz o seguinte código no método:

```
// see how controller command call a JSP

 rspProp.put(ECConstants.EC_VIEWTASKNAME, "SampleViewTask");
 setResponseProperties(rspProp);
```

   Salve seu trabalho (**Ctrl + S**). O fragmento de código anterior define o nome de exibição a ser retornado pelo comando do controlador.
9. O controle de acesso no nível de comando deve ser especificado para esse novo comando do controlador. Nesse caso, a política de controle de acesso no nível de comando especificará que todos os usuários têm permissão para executar o comando. Essa política é especificada no arquivo `SampleCmdACPolicy.xml`. Para carregar a nova política de controle de acesso, faça o seguinte:
   a. Em um prompt de comandos, vá para o diretório: *unidade*`:\WebSphere\CommerceServerDev\bin`
   b. Você deve emitir o comando `acpload`, que tem a seguinte forma:

```
acpload db_name db_user db_password inputXMLFile
```

em que

- *db_name* é o nome do banco de dados
- *db_user* é o nome do usuário do banco de dados
- *db_password* é a senha do banco de dados
- *inputXMLFile* é o nome do arquivo XML que contém a política.

Por exemplo, você pode emitir o seguinte comando:

```
acpload VAJ_Demo user password SampleCmdACPolicy.xml
```

10. Inclua o novo projeto _WCSamples no caminho da classe do mecanismo de servlet, procedendo da seguinte maneira:
    a. No menu **Área de Trabalho**, selecione **Ferramentas > WebSphere Test Environment**.
       O WebSphere Test Environment Control Center é aberto.
    b. Clique em **Mecanismo de Servlet** e em seguida, em **Editar Caminho da Classe**
       Na janela Caminho da Classe do Mecanismo de Servlet, clique em **Selecionar Tudo** e em seguida, em **OK**.
11. Teste seu novo comando, fazendo o seguinte:
    a. Inicie o servidor de nomes persistente, conforme descrito na página 347.
    b. Inicie o servidor EJB, conforme descrito na página 348.
    c. Inicie o mecanismo servlet, conforme descrito na página 348
    d. Abra um navegador e digite a seguinte URL:

       ```
       http://localhost:8080/webapp/wcs/stores/servlet/
          MyNewControllerCmd
       ```

       Depois de alguns minutos, o navegador apresenta uma página exibindo “Sample JSP”, conforme mostrado abaixo:

Figura 31.

12. Crie uma versão do código neste estado atual. Isso permite restaurar o código para esse estado em qualquer ponto. Para criar a versão, proceda da seguinte forma:
    a. Selecione os pacotes **com.ibm.commerce.sample.commands** e **com.ibm.commerce.sample.databeans** (mantenha pressionada a tecla `Ctrl` enquanto destaca para selecionar vários pacotes), clique com o botão direito do mouse e selecione **Gerenciar > Criar Edição Aberta**.
    b. Clique com o botão direito do mouse no projeto **_WCSamples** e selecione **Gerenciar > Criar Edição Aberta**.
    c. Clique com o botão direito do mouse no projeto **_WCSamples** novamente e selecione **Gerenciar > Versão**.
    A janela Versão de Itens Selecionados é aberta.
    d. Selecione o botão de opção **Um Nome**, digite `mySample 1.2 Completed` e clique em **OK**.

Agora você tem um novo comando incluído e o testou (brevemente) no ambiente de teste integrado.

Modificar MyNewControllerCmd

Na seção anterior, você criou MyNewControllerCmd. Neste exercício, você examinará com mais cuidado o conteúdo do novo comando para desenvolver uma compreensão melhor de como escrever seu próprio comando de controlador personalizado.

Inicie examinando a estrutura do código que você criou. A interface MyNewControllerCmd estende a interface ControllerCommand. Ela define também a classe de implementação a ser utilizada como padrão. Esta classe é utilizada quando o comando não é registrado na tabela CMDREG ou quando uma classe de implementação não é especificada nessa tabela.

Você criou também a classe MyNewControllerCmdImpl. A classe de implementação é a classe que eventualmente conterá a lógica de negócios (ou a chamada de comandos de tarefas para executar tarefas de negócios individuais) que deseja implementar. Sua classe de implementação contém os seguintes métodos:

Métodos em MyNewControllerCmdImpl	**Descrição**
MyNewControllerCmdImpl()	O método construtor.
validateParameters()	Utilizado para validação de lateral do servidor dos parâmetros de entrada do comando.
isGeneric()	Determina se um usuário genérico pode ou não chamar o comando.
isRetriable()	Determina se o comando será ou não recuperado após um retorno de banco de dados.
performExecute()	Contém a lógica de negócios para seu comando.

As seções a seguir mostram mais detalhes sobre como atualizar seu novo comando do controlador.

Passar variáveis ao modelo JSP

Nesta seção, você modifica MyNewControllerCmd para transmitir variáveis ao modelo JSP. Para exibir as variáveis, será necessário utilizar um novo bean de dados, chamado DataBeanSampleBean. Para permitir a exibição de variáveis em seu modelo JSP, proceda da seguinte forma:

1. Na janela Workbench do VisualAge for Java, expanda o projeto **_WCSamples**.
2. Expanda o pacote **com.ibm.commerce.sample.commands** , em seguida, a classe **MyNewControllerCmdImpl** e selecione seu método **performExecute**.
3. No código de origem do método performExecute, retire o comentário da Seção 2. Isso introduzirá o seguinte código no método:

```
// see how controller command pass in variables to JSP

 // Add additional parameters in controller command
 // to rspProp for response
```

```
//
rspProp.put("ControllerParm1", "Hello world");
rspProp.put("ControllerParm2", "Have a nice day!");
```

O fragmento de código acima cria dois novos parâmetros que são colocados nas propriedades a serem transmitidas ao modelo JSP. Salve seu trabalho (**Ctrl + S**).

4. O bean de dados DataBeanSampleBean é utilizado pelo modelo JSP para exibir as variáveis. O bean foi criado para você, mas é necessário modificar o código fonte da seguinte forma:
   a. Expanda o pacote **com.ibm.commerce.sample.databeans**.
   b. Expanda a classe **DataBeanSampleBean** e em seguida, selecione o método **setRequestProperties**, para exibir seu código fonte.
   c. No código fonte para o método setRequestProperties, retire o comentário da Seção 1. Isso introduzirá o seguinte código no método:

      ```
      // copy input TypedProperties to local

       requestProperties = aParam;
      ```

      Salve seu trabalho. O fragmento de código anterior copia os valores de aParam (definidos na superclasse) localmente. Este é utilizado para obter as propriedades do objeto de pedido.
5. Atualize o arquivo Sample.jsp para que utilize o novo bean de dados e exiba as variáveis, procedendo da seguinte forma:
   a. Utilizando um editor de texto, abra os arquivos `Sample.jsp` e `Sample_All.jsp`. Esses arquivos estão localizados no seguinte diretório: *vaj_drive*`:\VAJava\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host \default_app\web`
   b. Copie a Seção 1 do código de Sample_All.jsp para Sample.jsp entre os marcadores `<!-- SECTION 1 -->` e `<!-- END OF SECTION 1 -->`. Isso introduzirá o seguinte código no modelo JSP:

      ```
      <!-- SECTION 1 -->
      <%
      DataBeanSampleBean testBean = new DataBeanSampleBean ();
      com.ibm.commerce.beans.DataBeanManager.activate (testBean, request);
      %>
      <!-- END OF SECTION 1 -->
      ```

      Essa seção de código instância o bean de dados.
   c. Copie a Seção 2 de Sample_All.jsp no Sample.jsp entre os marcadores `<!-- SECTION 2 -->` e `<!-- END OF SECTION 2 -->`. Isso introduzirá o seguinte código no modelo JSP:

      ```
      <!-- SECTION 2 -->
      <%
      TypedProperty prop = testBean.getRequestProperties();
      ```

```
out.print("<B>List of name value pairs in TypedProperties
           object</B><P>");

// convert from request Properties to query string
for (Enumeration pns = prop.keys(); pns.hasMoreElements();) {
    String paramName = (String) pns.nextElement();
     // do not add the url parameter to the query string
     Object val = prop.get(paramName,null);
    if (val != null) {
         if (val.getClass().isArray()) {
          // flatten the array
         String[] oarray = (String[]) val;
         int len = java.lang.reflect.Array.getLength(val);
         for (int i = 0; i < len; i++) {
         out.print(paramName + "[" + i + "]=" + oarray[i] + "<br>");
         }
    } else {
    // assume that it is a String
    out.print(paramName + "=" + val.toString() + "<br>");
    }
  }
 }
%>
<P>
<!-- END OF SECTION 2 -->
```

Salve esse arquivo.
Esta seção de código utiliza o método getRequestProperties do objeto de bean de dados testBean. Ele circula pelas propriedades e as exibe no navegador.

6. Teste as modificações para verificar se as variáveis são exibidas no modelo JSP, procedendo da seguinte forma:
   a. Certifique-se de que o WebSphere Test Environment esteja em execução.
   b. Em um navegador, digite a seguinte URL:
      `http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd`

      O arquivo Sample JSP é exibido, mostrando as propriedades do comando do controlador. A saída aparece da seguinte forma:

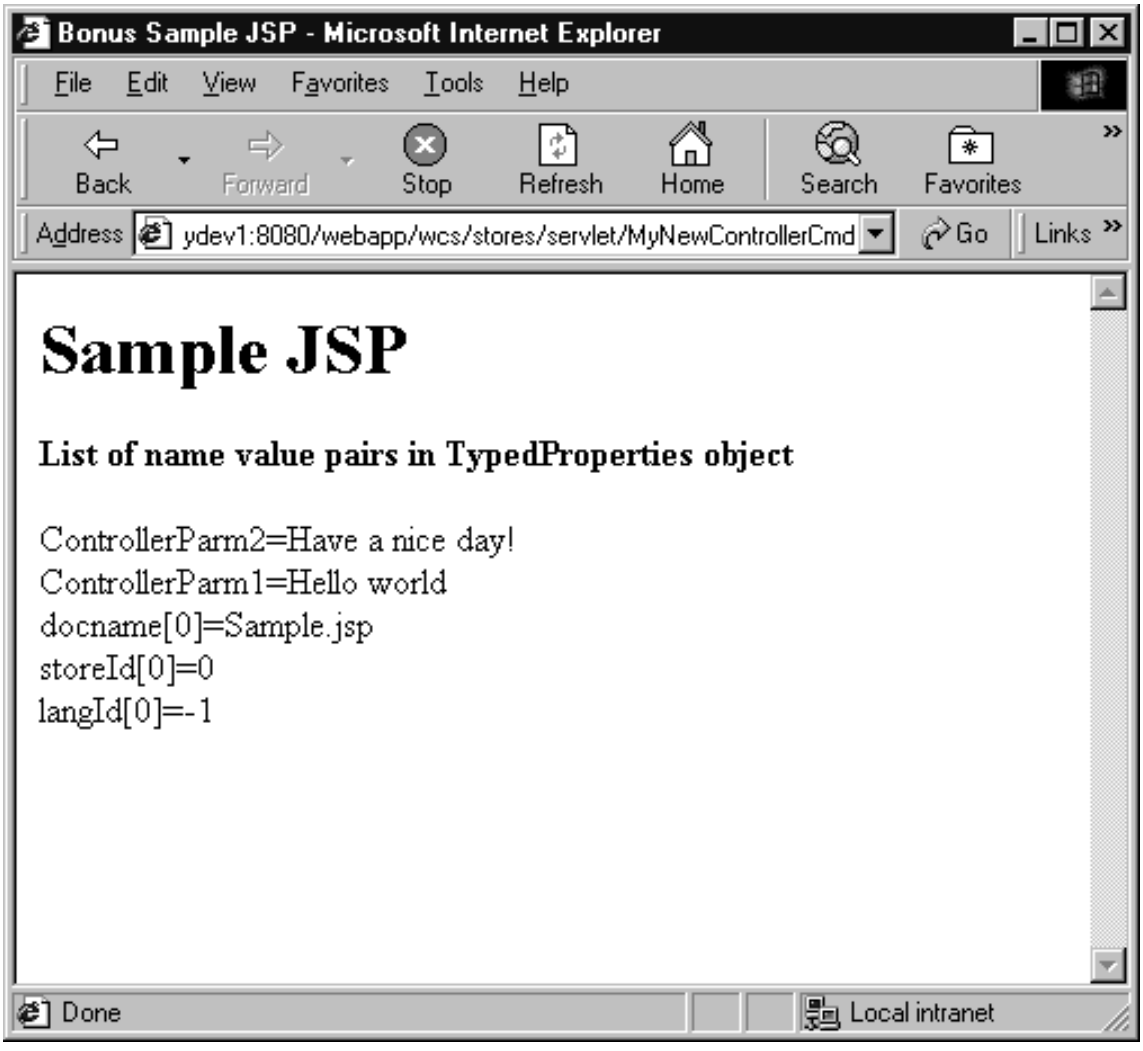


Figura 32.

7. Crie uma versão do código neste estado atual. Denomine a versão como `mySample 1.3 Completed`. Se você precisar de detalhes sobre a versão do código, consulte a etapa 12 na página 215.

Modificar o Método validateParameters

O método validateParameters que está atualmente em seu novo comando é, na verdade, apenas um stub de comando. Ele consiste no seguinte código:

```
public void validateParameters() throws ECException {
     }
```

Nesta seção, você incluirá sua própria verificação de parâmetro personalizada em seu comando e em seguida, a passará ao modelo JSP.

Ao modificar o método validateParameters, você inclui novos campos nas classes que são utilizadas pelos parâmetros.

Criação de novos campos: Ao incluir novos campos em uma interface ou classe existente, você pode utilizar o SmartGuide Criar Campos no VisualAge for Java. Esta seção descreve as etapas genéricas para a criação de um novo campo. Utilize estas informações para as seguintes seções do tutorial, quando você precisar incluir campos novos nas interfaces ou classes.

Para criar um novo campo, faça o seguinte:

1. Clique com o botão direito do mouse na interface ou classe na qual você está incluindo o novo campo e selecione **Incluir > Campo**.
   O SmartGuide Criar Campo é aberto.
2. No campo **Nome do Campo**, digite o nome do campo que deseja incluir.
3. Para especificar o tipo de campo, proceda de uma das seguintes formas:
   - Na lista suspensa **Tipo de Campo**, selecione o tipo de campo.
   - Se o tipo de campo requerido não estiver especificado na lista, clique em **Procurar**. No campo **Padrão**, digite o nome (ou nome parcial) do tipo de campo e clique em **OK**.

   Nota: Nos tutoriais, sempre que o tipo de campo `String` é especificado, isso é proveniente do pacote `java.lang`.
4. No campo de **Valor Inicial**, digite o valor inicial do campo. Para os valores iniciais que estão em cadeias, lembre-se de adicionar o valor entre aspas duplas (“ ”).
5. Para o valor **Modificadores de Acesso**, selecione o botão de opção apropriado (público, protegido, nenhum, ou particular), se necessário.
6. Selecione a caixa de opções **Acessar com métodos getter e setter** se você desejar ter os métodos ’getter’ e ’setter’ gerados para o campo. Se selecionar isto, você deve também especificar as propriedades para esses métodos, como segue:
   - Para o método ’getter’, selecione público, protegido, particular ou nenhum dos botões da caixa de opções.
   - Para o método ’setter’, selecione público, protegido, particular ou nenhum dos botões da caixa de opções.
7. Clique em **Concluir**.

Para modificar o método validateParameters, proceda da seguinte maneira:

1. Você deve criar dois novos campos na classe MyNewControllerCommandImpl. O primeiro é utilizado para cadeias de entrada, e o segundo para inteiros de entrada. Para criar estes campos, proceda da seguinte forma:
   a. Expanda o pacote **com.ibm.commerce.sample.commands**.
   b. Selecione a classe **MyNewControllerCmdImpl**.
   c. Inclua um campo na classe, utilizando os seguintes valores. Para obter etapas detalhadas de como criar um campo novo, consulte "Criação de novos campos" na página 220.

Nome do Atributo	**Valor**
Nome do Campo	`inputString`
Tipo de Campo	`String`
Valor Inicial	Deixar em branco.
Modificadores de Acesso	`private`
Outros Modificadores	Deixar todos desmarcados.
Acessar com os métodos getter e setter	`checked`
Getter	`public`
Setter	`public`

   d. Crie outro campo para input de inteiros, utilizando os seguintes valores:

Nome do Atributo	**Valor**
Nome do Campo	`inputInteger`
Tipo de Campo	`Integer` **Nota:** Clique em **Navegar** e digite `Integer`. Não selecione `int`.
Valor Inicial	Deixar em branco.
Modificadores de Acesso	`private`
Outros Modificadores	Deixar todos desmarcados.
Acessar com os métodos getter e setter	`checked`
Getter	`public`
Setter	`public`

2. Selecione o método **performExecute** na classe MyNewControllerCmdImpl.
3. No código fonte do método performExecute, retire o comentário da Seção 3 para introduzir o seguinte código no método:

```
// see how controller command pass in input variables to JSP

     rspProp.put("ControllerInput1", getInputString());
     rspProp.put("ControllerInput2", getInputInteger().toString());
```

Esse código passa as variáveis do comando do controlador para o bean de dados.Salve seu trabalho.

4. Selecione o método **validateParameters** na classe MyNewControllerCmdImpl.
5. Remova o comentário da Seção 1 no código fonte validateParameters para introduzir o seguinte código no método:

```
// uncomment to check parameters

      TypedProperty prop = getRequestProperties();

// retrieve required parameters
 //
 try {
          setInputString(prop.getString("input1"));
 } catch (ParameterNotFoundException e) {
          throw new ECApplicationException(
               ECMessage._ERR_CMD_MISSING_PARAM,
               "MyControllerCmdImpl","validateParameters",
               ECMessageHelper.generateMsgParms(e.getParamName()));
 }

 // retrieve optional Integer
 // set input2 = 0 if no input value
 //
 setInputInteger(prop.getInteger("input2", 0));
```

Salve seu trabalho.
O fragmento de código anterior verifica os dois parâmetros de entrada. O bloco de tentativas determina se o primeiro parâmetro está lá, se não estiver, será emitida uma exceção. Visto que o segundo parâmetro é opcional, esse código definirá o valor do parâmetro para `0` se o parâmetro estiver ausente ou for do tipo errado.

6. Teste o comando procedendo da seguinte maneira:
   a. Assegure-se de que o servidor de nomes persistente, o servidor EJB e o mecanismo SERVLET estejam funcionando.
   b. No navegador, digite as seguintes URLs:
      - Caso 1: Parâmetro ausente:
        Digite

        ```
        http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd
        ```

Como nenhum parâmetro é transmitido ao comando, um erro de aplicativo genérico é mostrado para indicar que está faltando um parâmetro. O resultado é exibido na seguinte captura de tela:

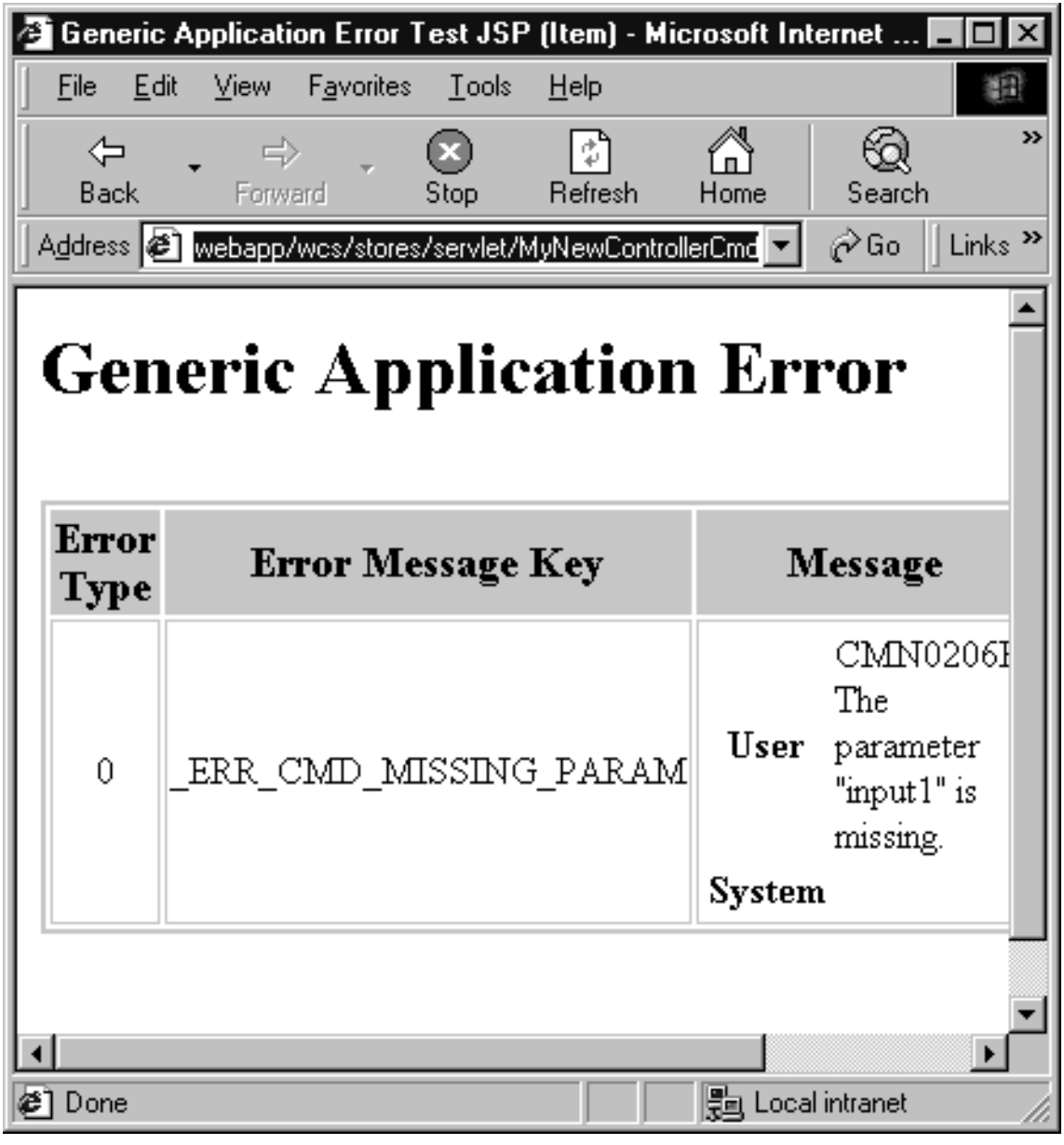


Figura 33.

Nota: Se um erro “Página não localizada” for exibido, seu mecanismo de servlet pode ter sido parado. Verifique o WebSphere Test Environment Control Center para obter detalhes. Se a página Sample JSP for exibida em vez da página Generic Application Error, poderá ser necessário parar e iniciar o mecanismo de servlet novamente, ou recarregar a página no navegador.

- Caso 2: Primeiro parâmetro válido, segundo parâmetro ausente : Digite

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
      input1=abc
```

  O resultado desse comando é que a página Sample JSP é exibida apesar do segundo parâmetro ter sido omitido. A tela a seguir exibe este resultado. Uma explicação de por que um erro não foi retornado segue a tela.

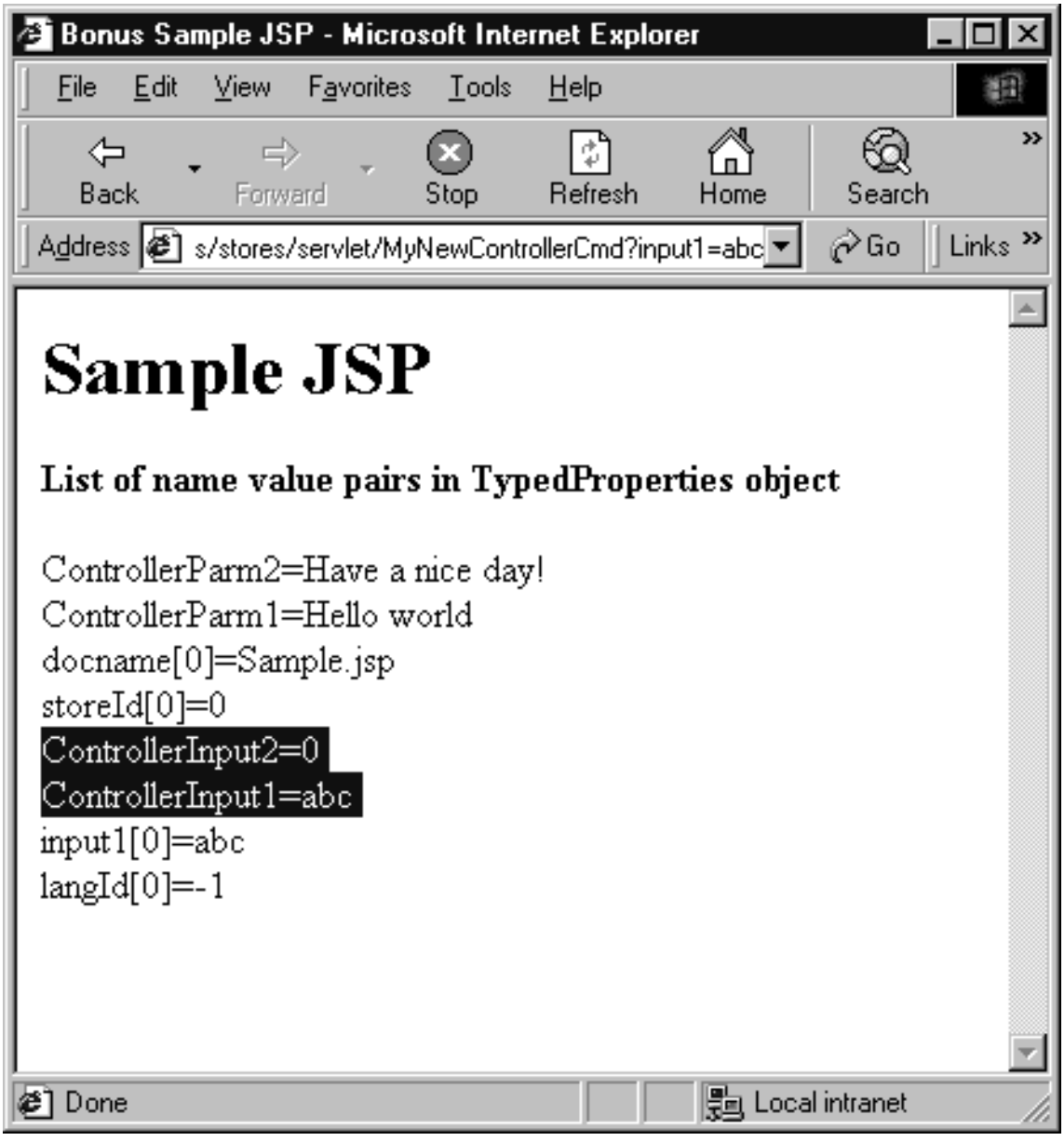


Figura 34.

Um erro não foi retornado por causa da maneira como o método `getInteger` foi utilizado. Especificamente, a linha de código `setInputInteger(prop.getInteger("input2", 0))` define um valor padrão de 0 para input2. Esse valor padrão é utilizado quando o parâmetro está ausente ou é do tipo errado. Para forçar a verificação

de tipo nesse parâmetro, altere o código para `setInputInteger(prop.getInteger("input2"))` e digite novamente a URL (certifique-se de atualizar o navegador). Deve ser mostrada uma página de erro de aplicativo genérico.

Nota: Se você testar a modificação `setInputInteger(prop.getInteger("input2"))` em seu código, retorne-o para `setInputInteger(prop.getInteger("input2", 0))` antes de continuar com a próxima etapa de teste.

- Caso 3: Primeiro parâmetro válido, segundo parâmetro inválido: Digite

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
     input1=abc&input2=abc
```

O resultado deste comando é que a página Sample JSP é exibida e o valor para o segundo parâmetro (um inteiro) é definido para o valor padrão de `0`. Este é um resultado do método `getInteger` utilizado, conforme descrito no caso de teste anterior. O resultado é exibido na seguinte captura de tela:

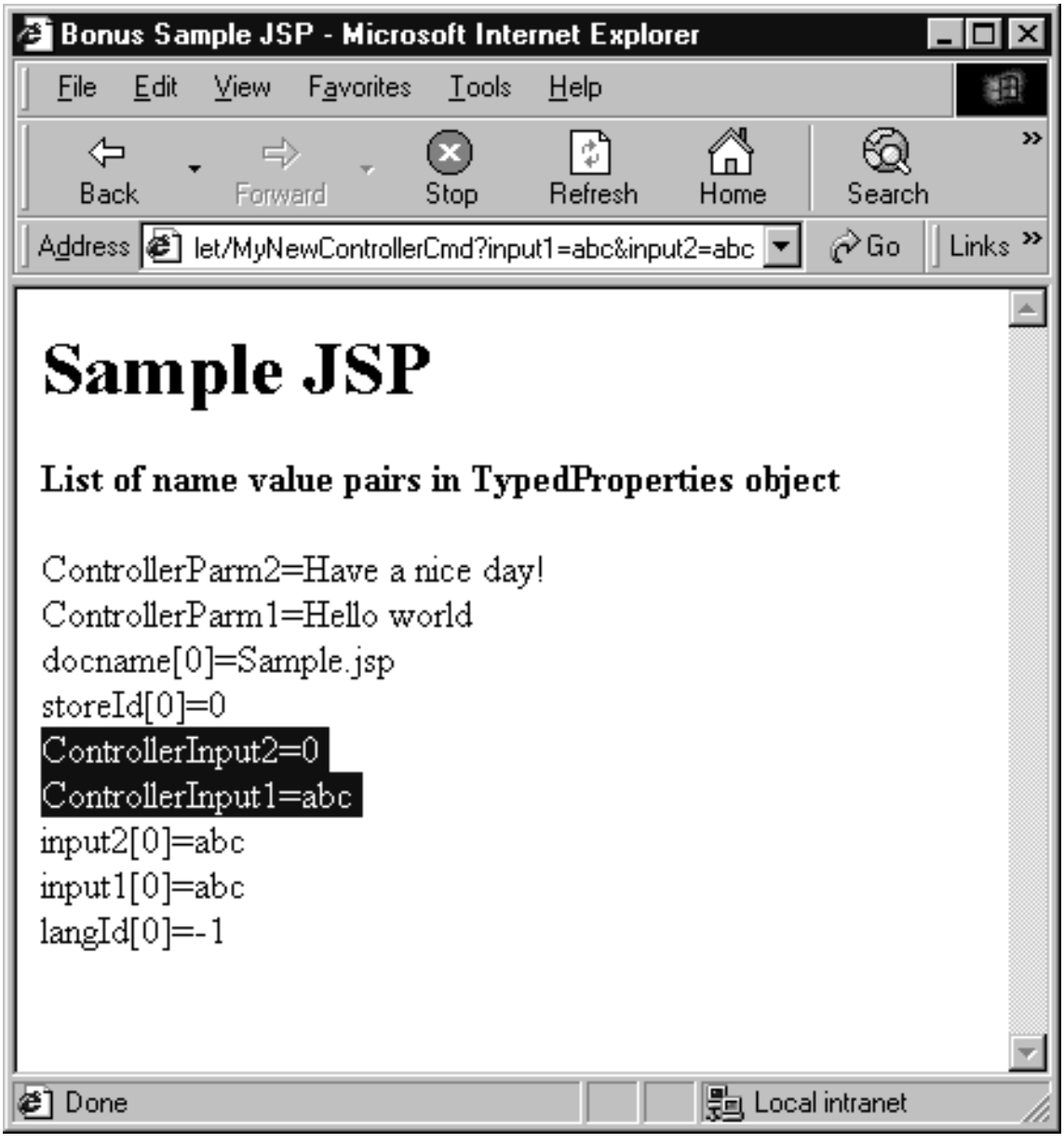


Figura 35.

- Caso 4: Dois parâmetros válidos:
  Digite

  ```
  http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
      input1=abc&input2=1000
  ```

O resultado desse comando será que a página Sample JSP será exibida e os dois valores de entrada serão exibidos conforme digitados. O resultado é exibido na seguinte captura de tela:

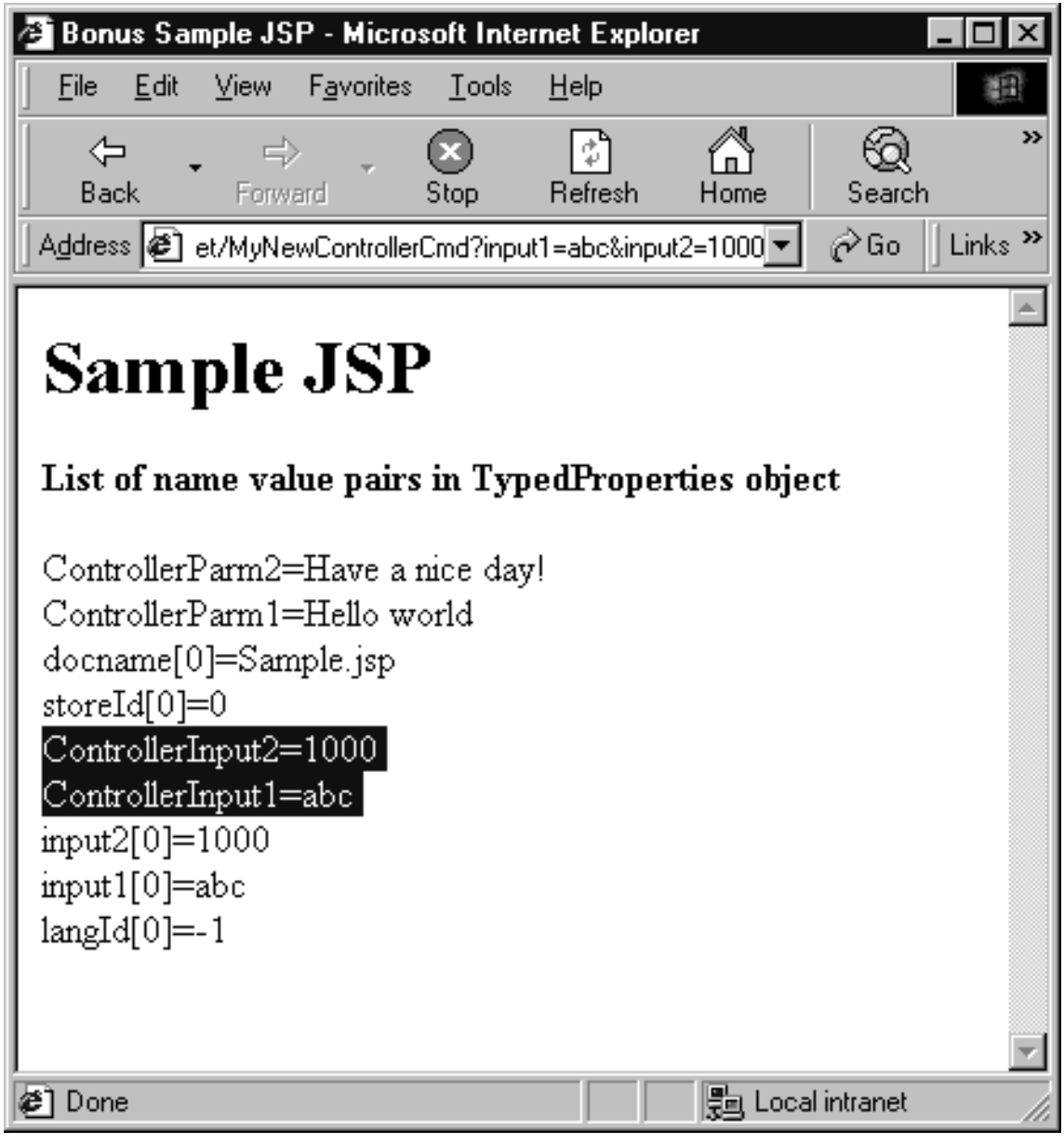


Figura 36.

7. Crie uma versão do código neste estado atual. Denomine a versão como `mySample 1.4 Completed`. Se você exigir informações detalhadas sobre como fazer a versão do código, consulte a etapa 12 na página 215.

Criar um comando de tarefas

Um comando do controlador geralmente representa um processo de negócios ou uma função complexa. Por exemplo, todas as lógicas de negócios relacionadas ao processo de pedidos são encapsuladas no comando do controlador OrderProcessCmd. Um processo de negócio pode, geralmente, ser dividido em tarefas menores e específicas. Por exemplo, dentro do comando do controlador OrderProcessCmd, há vários comandos de tarefas que foram chamados para executar unidades individuais de trabalho. Um desses comandos de tarefas chamados pelo comando do controlador OrderProcessCmd é CalculateOrderTaxTotalCmd.

MyNewControllerCmdImpl atualmente não chama nenhum comando de tarefas. Esse exercício possui duas seções. Na primeira seção, você cria o novo comando de tarefas. Na segunda seção, você modifica o método performExecute de seu comando do controlador para chamar o comando de tarefas.

O fragmento de código a seguir mostra o método performExecute atual da classe MyNewControllerCmdImpl, com todos os comentários removidos:

```
public void performExecute() throws ECException {

     super.performExecute();

     TypedProperty rspProp = new TypedProperty();
     rspProp.put("ControllerParm1", "Hello world");
     rspProp.put("ControllerParm2", "Have a nice day!");

     rspProp.put("ControllerInput1", getInputString());
     rspProp.put("ControllerInput2", getInputInteger().toString());

     rspProp.put(ECConstants.EC_VIEWTASKNAME, "SampleViewTask");
     setResponseProperties(rspProp);

}
```

Escrever o Código do Comando de Tarefas: Essa seção mostra como escrever um novo comando de tarefas. A criação de um comando de tarefas completamente novo envolve a criação de uma interface e de uma classe de implementação. Ao criar um comando de tarefas, a interface deve estender `com.ibm.commerce.commands.TaskCommand`. A classe de implementação deve estender `com.ibm.commerce.command.TaskCommandImpl`.

Ao concluir este exercício, você terá um novo comando, chamado *MyNewTaskCmd*. Esse comando é utilizado por todas as lojas e cada loja utiliza a mesma implementação do comando.

Nesta parte do tutorial, você inclui campos e métodos na interface do novo comando de tarefas. Você criou anteriormente novos campos para a classe MyNewControllerCmdImpl. Tente criar os campos para esta interface você mesmo, mas se precisar de mais detalhes, consulte "Criação de novos campos" na página 220.

Criação de métodos: Esta seção descreve as etapas genéricas para adicionar métodos às classes e interfaces existentes. Leia as instruções aqui e refira-se novamente a elas quando o tutorial exigir que você crie novos métodos.

Para criar um novo método, proceda da seguinte forma:

1. Clique com o botão direito do mouse na interface ou classe na qual vocês está incluindo o método e selecione **Incluir > Método**.
   O SmartGuide Criar Método é aberto.
2. Assegure-se de que **Criar Novo Método** esteja selecionado e clique em **Avançar**.
3. No campo **Nome do Método**, digite o nome do novo método.
4. Especifique o tipo de retorno do método procedendo de uma das seguintes maneiras:
   - Na lista suspensa **Tipo de Retorno**, selecione o tipo de retorno apropriado. Por exemplo, selecione `Cadeia`.
   - Se o tipo de retorno não estiver especificado na lista, clique em **Procurar**. Em seguida, no campo **Padrão**, digite o nome do tipo de retorno e clique em **OK**.

   Nota: Nos tutoriais, sempre que `String` é especificado como o tipo de retorno, isso é proveniente do pacote `java.lang`.
5. Se o método requer parâmetros, clique em **Incluir**. Na janela Parâmetros, especifique o nome do parâmetro e quaisquer outras informações necessárias e clique em **Incluir**. Depois que todos os parâmetros tiverem sido incluídos, clique em **Fechar**.
6. Clique em **Avançar**.
   A janela Atributos é aberta.
7. Se o método apresentar exceções, clique em **Incluir** na janela Atributos. No campo **Padrão**, digite o nome da exceção e em seguida, clique em **Incluir**. Depois que todas as exceções tiverem sido incluídas, clique em **Fechar**.
8. Clique em **Concluir**.
   O código para o método é gerado.

Para criar o comando MyNewTaskCmd, proceda da seguinte forma:

1. Expanda o pacote **com.ibm.commerce.sample.commands**.
2. Clique com o botão direito do mouse na interface **MyNewTaskCmd** e selecione **Incluir > Campo**.
3. Utilize o SmartGuide Criar Campo, crie um campo que especifique a classe de implementação padrão a ser utilizada pela interface. Utilize os valores na tabela a seguir. Se você precisar de mais detalhes ao criar um campo, consulte "Criação de novos campos" na página 220.

Nome do Atributo	Valor
Nome do Campo	`defaultCommandClassName`
Tipo de Campo	`String`
Valor Inicial	`"com.ibm.commerce.sample.commands.MyNewTaskCmdImpl"`

Quando o código do novo campo é gerado, ele aparece como segue:

```
java.lang.String defaultCommandClassName =
     "com.ibm.commerce.sample.commands.MyNewTaskCmdImpl";
```

Como a mesma classe de implementação é utilizada para todo o site e nenhuma propriedade padrão é transmitida ao comando, você pode especificar a implementação padrão direto no código. Se você tem um comando que possui várias implementações ou possui propriedades padrão (que estão armazenados na tabela CMDREG), deverá registrar o comando na tabela CMDREG para criar o mapeamento entre a interface e a classe de implementação.

4. Inclua novos métodos na interface MyNewTaskCmd, procedendo da seguinte maneira:
   a. Clique com o botão direito do mouse na interface **MyNewTaskCmd** e selecione **Incluir > Método**. Utilizando o SmartGuide Criar Método, crie novos métodos utilizando os valores especificados nas etapas a seguir. Se você precisar de informações detalhadas para a criação dos métodos, consulte "Criação de métodos" na página 230.
   b. Crie um novo método que recupere o saldo de pontos de bônus de um cliente utilizando os seguintes valores:

Nome do Atributo	Valor
Nome do Método	`getOldBonusPoint`
Tipo de Retorno	`String`
Parâmetros	nenhum
Exceções	nenhum

Nota: Um erro pode ser exibido, indicando que o método abstrato herdado que acaba de ser criado não está implementado na classe de implementação MyNewTaskCmdImpl. Isto é corrigido em uma etapa subseqüente.

c. Crie um novo método que recupere o ID do usuário de saída do comando de tarefa. Ao criar este método, utilize os seguintes valores:

Nome do Atributo	**Valor**
Nome do Método	`getTask_output_userId`
Tipo de Retorno	`String`
Parâmetros	nenhum
Exceções	nenhum

d. Crie um novo método que recupere um valor de saída do comando de tarefa. Ao criar este método, utilize os seguintes valores:

Nome do Atributo	**Valor**
Nome do Método	`getTask_output1`
Tipo de Retorno	`String`
Parâmetros	nenhum
Exceções	nenhum

e. Crie um novo método que defina o primeiro valor de entrada para o comando de tarefa. Ao criar este método, utilize os seguintes valores:

Nome do Atributo	**Valor**
Nome do Método	`setTask_input1`
Tipo de Retorno	`void`
Nome do Parâmetro	`newTask_input1`
Tipo de Referência	`String`
Exceções	nenhum

f. Crie um novo método que defina o segundo valor de entrada para o comando de tarefa. Ao criar este método, utilize os seguintes valores:

Nome do Atributo	**Valor**
Nome do Método	`setTask_input2`
Tipo de Retorno	`void`
Nome do Parâmetro	`newTask_input2`
Tipo Primitivo	`int`

Nome do Atributo	Valor
Exceções	nenhum

5. Adicione novos campos à classe MyNewTaskCmdImpl, procedendo da seguinte maneira:
   a. Clique com o botão direito do mouse na classe **MyNewTaskCmdImpl** e selecione **Incluir > Campo**. Utilizando o SmartGuide Criar Campo, crie novos campos utilizando os valores especificados nas etapas a seguir. Se você necessitar de informações adicionais sobre a criação de novos campos, consulte "Criação de novos campos" na página 220.
   b. Crie um novo campo na classe de implementação, utilizando os seguintes valores:

Nome do Atributo	Valor
Nome do Campo	`task_input1`
Tipo de Campo	`String`
Valor Inicial	Deixar em branco.
Modificadores de Acesso	`private`
Outros Modificadores	Deixar todos desmarcados.
Acessar com os métodos getter e setter	`checked`
Getter	`public`
Setter	`public`

   c. Crie um novo campo na classe de implementação, utilizando os seguintes valores:

Nome do Atributo	Valor
Nome do Campo	`task_input2`
Tipo de Campo	`int`
Valor Inicial	Deixar em branco.
Modificadores de Acesso	`private`
Outros Modificadores	Deixar todos desmarcados.
Acessar com os métodos getter e setter	`checked`
Getter	`public`
Setter	`public`

   d. Crie um novo campo na classe de implementação, utilizando os seguintes valores:

Nome do Atributo	Valor
Nome do Campo	`task_output_userId`
Tipo de Campo	`String`
Valor Inicial	Deixar em branco.
Modificadores de Acesso	`private`
Outros Modificadores	Deixar todos desmarcados.
Acessar com os métodos getter e setter	`checked`
Getter	`public`
Setter	`public`

e. Crie um novo campo na classe de implementação, utilizando os seguintes valores:

Nome do Atributo	Valor
Nome do Campo	`oldBonusPoint`
Tipo de Campo	`String`
Valor Inicial	Deixar em branco.
Modificadores de Acesso	`private`
Outros Modificadores	Deixar todos desmarcados.
Acessar com os métodos getter e setter	`checked`
Getter	`public`
Setter	`public`

f. Crie um novo campo na classe de implementação, utilizando os seguintes valores:

Nome do Atributo	Valor
Nome do Campo	`task_output1`
Tipo de Campo	`String`
Valor Inicial	Deixar em branco.
Modificadores de Acesso	`private`
Outros Modificadores	Deixar todos desmarcados.
Acessar com os métodos getter e setter	`checked`
Getter	`public`
Setter	`public`

6. Selecione o método **performExecute** da classe **MyNewTaskCmdImpl** para exibir seu código fonte.

7. No código fonte, retire o comentário da Seção 1 para introduzir o seguinte código no método:

```
// modify the task_input1 and see it in the NVP list

     setTask_output1( "Hello ! " + getTask_input1() );
```

   Salve seu trabalho.
   A seção anterior de código disponibiliza os novos atributos como saída do comando.

Chamar o Comando de Tarefas: Após ter criado seu comando de tarefas, você precisa chamar o comando de dentro do comando do controlador. As etapas a seguir descrevem como modificar o comando do controlador dessa maneira.

1. No Workbench, selecione o método **performExecute** de sua classe **MyNewControllerCmdImpl**.
2. Na área de janela, retire o comentário da Seção 4 para chamar o comando de tarefas. Isso introduzirá o seguinte código no método:

```
// see how controller command call a task command

     MyNewTaskCmd cmd = null;
     try {
          cmd = (MyNewTaskCmd) CommandFactory.createCommand(
               "com.ibm.commerce.sample.commands.MyNewTaskCmd",
               getStoreId());
          // Set input parameters to task command
          cmd.setTask_input1(getInputString());
          cmd.setTask_input2(getInputInteger().intValue());
          // This is required for all commands
          cmd.setCommandContext(getCommandContext());
          // Invoke the command's performExecute method
          cmd.execute();
          // retrieve output parameter from task command

          rspProp.put("task_output1", cmd.getTask_output1());

          if (cmd.getTask_output_userId() != null) {
               rspProp.put("task_output_userId",
               cmd.getTask_output_userId());
          }

          if (cmd.getOldBonusPoint() != null) {
               rspProp.put("task_output_oldBonusPoint",
               cmd.getOldBonusPoint());
          }

     } catch (ECException ex) {
             // throw the exception as is
             throw (ECException) ex;
     }
```

Salve seu trabalho.
O fragmento do código anterior cria um novo comando de tarefas utilizando a fábrica de comandos. Ele, então, define o contexto do comando, invoca a execução do comando de tarefas e recupera os parâmetros de saída do comando de tarefas.

3. Teste o comando digitando a URL de seu comando do controlador, da seguinte forma:

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
     input1=abc&input2=1000
```

O JSP de exemplo exibe a lista de pares nome valor no objeto do pedido, incluindo os valores de saída da tarefa. Deve aparecer da seguinte forma:

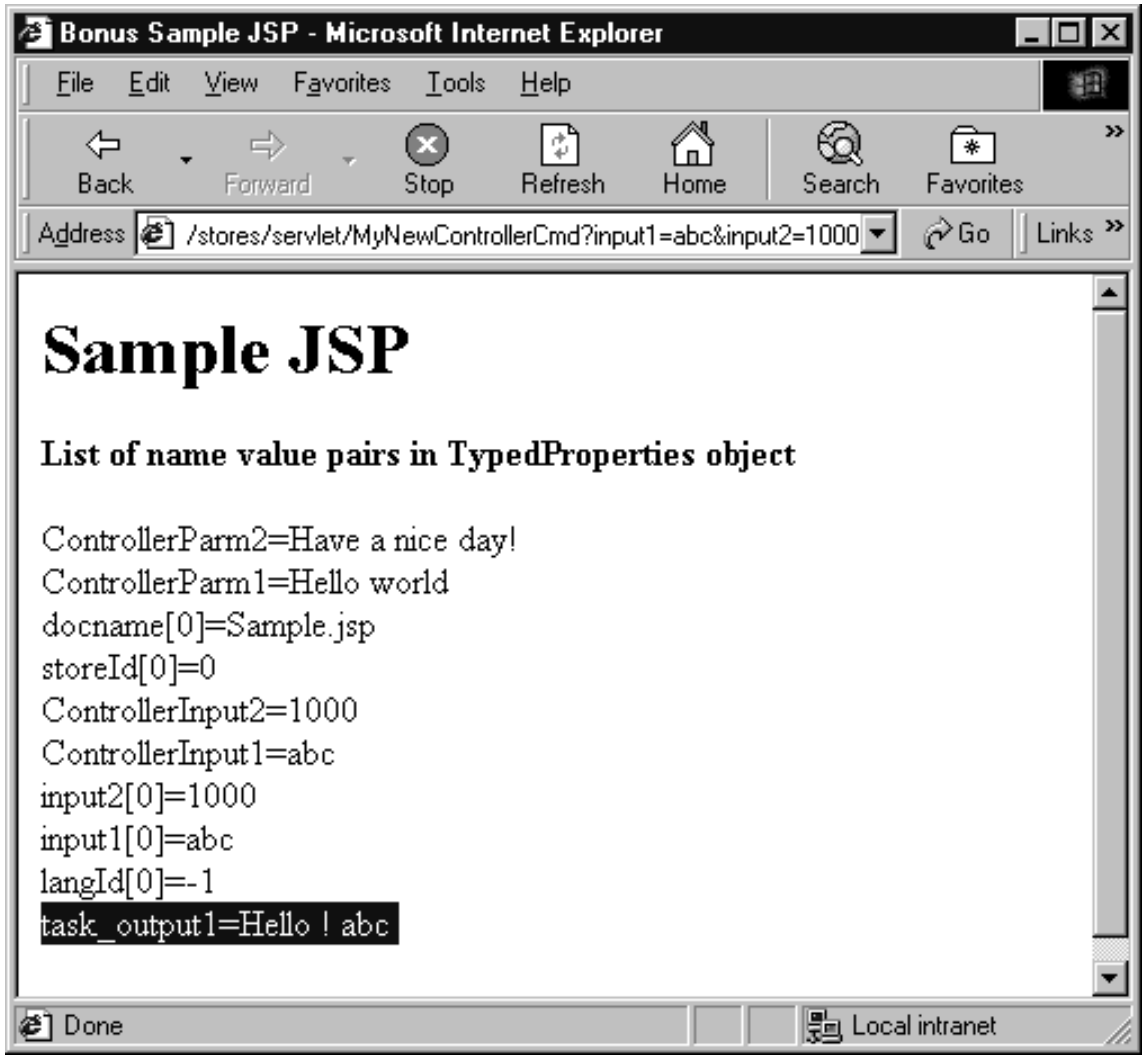


Figura 37.

4. Crie uma versão do código neste estado atual. Denomine a versão como `mySample 1.5 Completed`. Se você necessitar de mais detalhes sobre como fazer a versão do código, consulte a etapa 12 na página 215.

Validar um ID do Usuário

A próxima etapa é modificar o comando de tarefas para que utilize o UserRegistryAccessBean com a finalidade de validar se o valor digitado pelo usuário é esse ou não de um usuário registrado. Além de modificar o comando de tarefas, será necessário modificar DataBeanSampleBean.

Modificar MyNewTaskCmdImpl para validação do ID do usuário: Você deve modificar o método performExecute na classe MyNewTaskCmdImpl para que valide o ID do usuário, utilizando o UserRegistryAccessBean. Para incluir essa nova funcionalidade ao método performExecute, proceda da seguinte forma:

1. No Workbench, selecione o método **performExecute** de sua classe **MyNewTaskCmdImpl**.
2. Na área de janela de origem, retire o comentário da Seção 2 do método performExecute. Isso introduzirá o seguinte código no método:

```
// use UserRegistryAccessBean to check member reference number

     String refNum;

     UserRegistryAccessBean rrb = new UserRegistryAccessBean();

     try {
          rrb = rrb.findByUserLogonId(getTask_input1());
          refNum = rrb.getUserId();

     } catch (javax.ejb.FinderException e) {

     return;

     } catch (javax.naming.NamingException e) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
               this.getClass().getName(), "performExecute");
     } catch (java.rmi.RemoteException e) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
               this.getClass().getName(), "performExecute");
     } catch (javax.ejb.CreateException e) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
               this.getClass().getName(), "performExecute");
     }

      setTask_output_userId(refNum);
```

   Salve seu trabalho.

Modificar o DataBeanSampleBean: Você deve modificar o DataBeanSampleBean para utilizar os parâmetros de entrada predefinidos. Para modificar esse bean, proceda da seguinte forma:

1. No Workbench, expanda o pacote **com.ibm.commerce.sample.databeans**.
2. Inclua novos campos no bean de dados, procedendo da seguinte maneira:
   a. Clique com o botão direito do mouse na classe **DataBeanSampleBean** e selecione **Incluir > Campo**. Utilizando o SmartGuide Criar Campo, crie novos campos utilizando os valores especificados nas etapas a seguir. Se você necessitar de informações adicionais sobre a criação de campos, consulte “Criação de novos campos” na página 220.

b. Inclua um novo campo no bean de dados, utilizando os seguintes valores:

Nome do Atributo	**Valor**
Nome do Campo	`input1`
Tipo de Campo	`String`
Valor Inicial	Deixar em branco.
Modificadores de Acesso	`private`
Outros Modificadores	Deixar todos desmarcados.
Acessar com os métodos getter e setter	`checked`
Getter	`public`
Setter	`public`

c. Inclua um novo campo no bean de dados, utilizando os seguintes valores:

Nome do Atributo	**Valor**
Nome do Campo	`input2`
Tipo de Campo	`int`
Valor Inicial	Deixar em branco.
Modificadores de Acesso	`private`
Outros Modificadores	Deixar todos desmarcados.
Acessar com os métodos getter e setter	`checked`
Getter	`public`
Setter	`public`

d. Inclua um novo campo no bean de dados, utilizando os seguintes valores:

Nome do Atributo	**Valor**
Nome do Campo	`task_output_userId`
Tipo de Campo	`String`
Valor Inicial	Deixar em branco.
Modificadores de Acesso	`private`
Outros Modificadores	Deixar todos desmarcados.
Acessar com os métodos getter e setter	`checked`
Getter	`public`
Setter	`public`

3. Selecione o método **populate** da classe DataBeanSampleBean para exibir o código fonte.
4. No código fonte, retire o comentário da Seção 1A e da Seção 1B. Isso introduzirá o seguinte código no método:

```
//// Section 1A //////////
// set additional data fields

try {
   setInput1( getRequestProperties().getString("input1"));
   setInput2( getRequestProperties().getIntValue("input2", 0)  );
   setTask_output_userId(getRequestProperties().getString
     ("task_output_userId"));

//// End of Section 1A /////

   //// Section 2 ////////////
   /*
   // instantiate databean to BonusAccessBean
   setTask_output_oldBonusPoint(getRequestProperties().getString(
      "task_output_oldBonusPoint"));
   */
   //// End of Section 2 ////

//// Section 1A ///////////
// instantiate databean to BonusAccessBean and
// set additional data fields

   }
catch (ParameterNotFoundException e){}

//// End of Section 1B ////
```

   Salve seu trabalho.
   O fragmento do código anterior obtém valores do campo de dados do comando do controlador, utilizando o método getRequestProperties.

Modificar o Sample.jsp para validação do ID do usuário: O modelo JSP atual deve ser modificado a fim de executar a validação ao ID do usuário. Para modificar o arquivo Sample.jsp, proceda da seguinte forma:

1. Abra os arquivos `Sample.jsp` e `Sample_All.jsp` em um editor de texto.
2. Copie a Seção 3 do código incluída no Sample_All.jsp para o Sample.jsp entre os marcadores `<!-- SECTION 3 -->` e `<!-- END OF SECTION 3 -->`. O código a seguir é introduzido no modelo JSP

```
<!-- SECTION 3 -->

<B>Your first input is &lt; <%=testBean.getInput1()%> &gt;</B>

<%
String userId = testBean.getTask_output_userId();
```

```
if (userId == null) {

%>

<UL>
     <LI> This is not a registered user id.
</UL>

<%

} else {

%>

<B>
<UL>
     <LI> 'lt; <%=testBean.getInput1()%> &gt; is a registed user id.
     <LI> The member reference number of this user is <%=userId%>.
</UL>
</B>

<%

}

%>

<B>Your second input is < <%=testBean.getInput2()%> ></B> <P>

<!-- END OF SECTION 3 -->
```

Salve o arquivo Sample.jsp.

3. Abra um navegador e digite a seguinte URL:

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
     input1=abc&input2=1000
```

O navegador deve mostrar o arquivo Sample JSP indicando que o valor para input1 não é um ID do usuário válido, conforme mostrado na seguinte captura de tela:

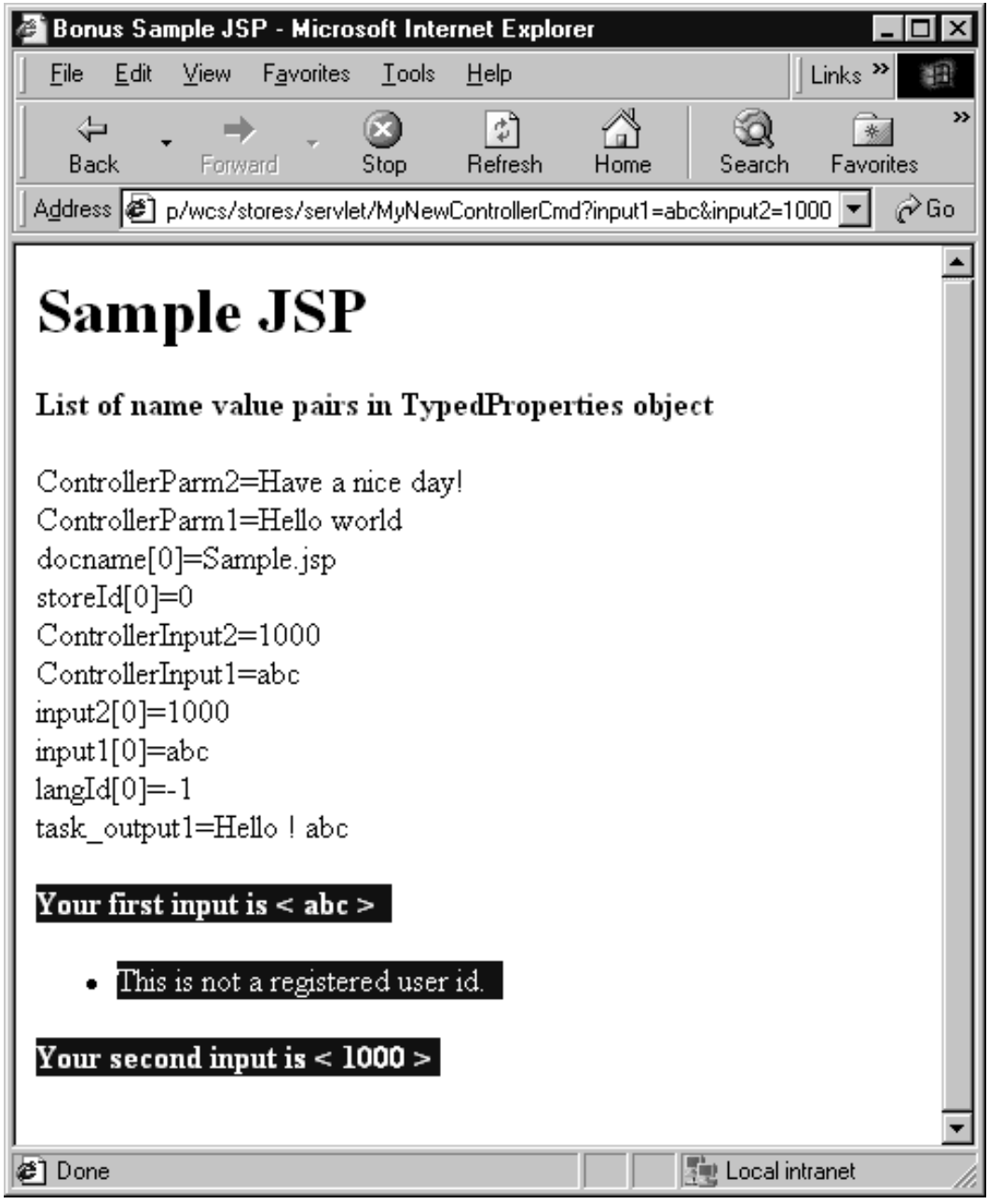


Figura 38.

4. Para ver o resultado quando o valor para input1 for um ID do usuário válido, digite a seguinte URL:

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
     input1=wcsadmin&input2=1000
```

Este é mostrado na seguinte captura de tela:

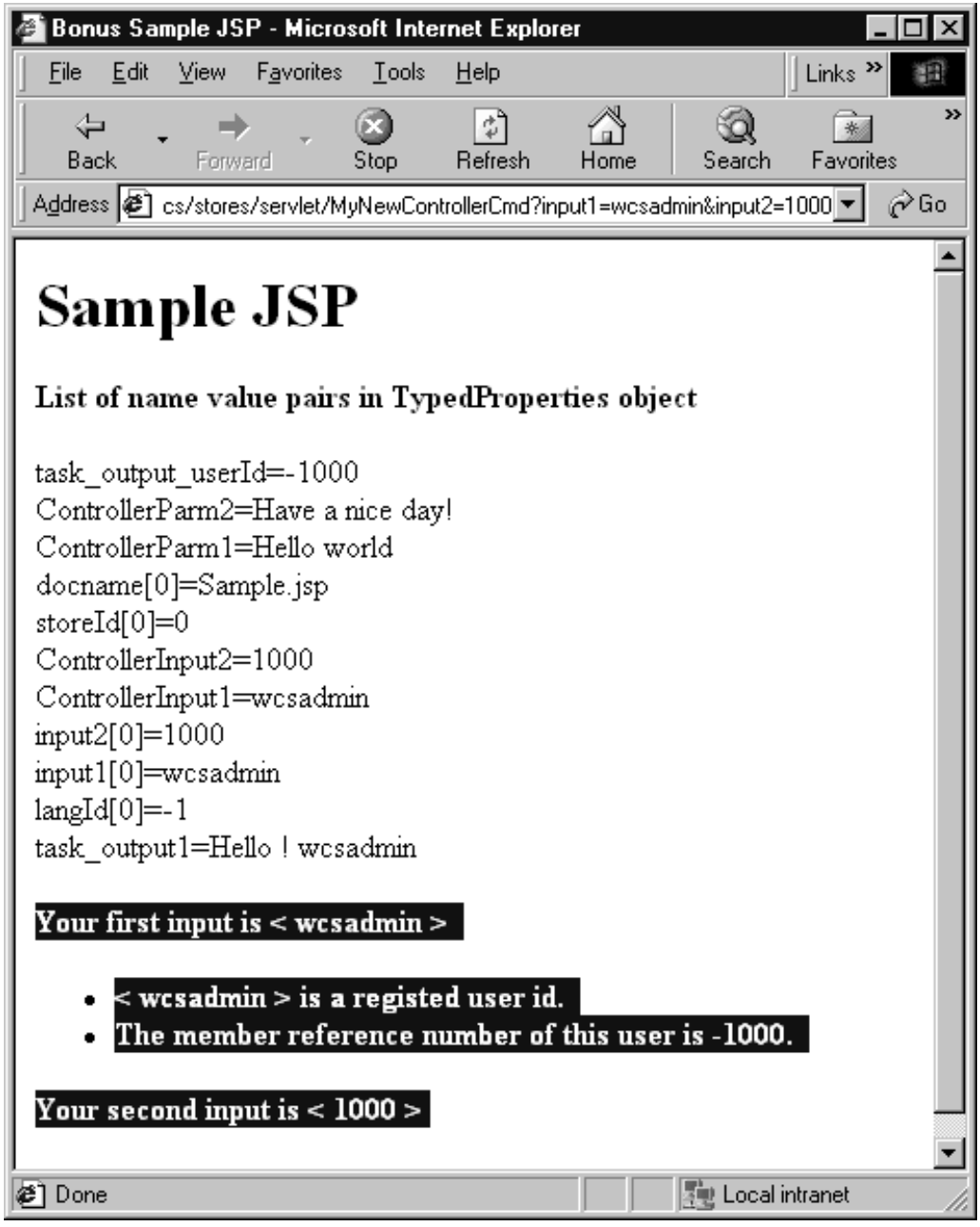


Figura 39.

5. Crie uma versão do código neste estado atual. Denomine a versão como `mySample 1.6 Completed`. Se você necessitar de informações detalhadas sobre como fazer a versão do código, consulte a etapa 12 na página 215.

Criando um Novo Bean de Entidade

Essa seção descreve como utilizar o VisualAge for Java para criar um novo bean de entidade. Nesse cenário de exemplo, você tem um requisito de negócio para incluir um registro de operações de pontos de bônus para cada usuário no aplicativo de comércio. O esquema do banco de dados do WebSphere Commerce não contém essas informações, dessa forma, será necessário criar uma nova tabela de banco de dados para que contenha essas informações. De acordo com o modelo de programação do WebSphere Commerce, depois que a tabela de banco de dados é criada, você deve criar um bean de entidade (que é um bean corporativo) para acessar os dados.

Neste exemplo, você utilizará um SmartGuide do VisualAge for Java para criar este bean de entidade.

Criando a Nova Tabela de Banco de Dados

Na preparação para a criação do bean de entidade, você deve primeiro criar a nova tabela do banco de dados. A tabela a ser criada é chamada Bonus.

DB2 Se estiver utilizando um banco de dados DB2, faça o seguinte para criar a tabela:

1. Abra o Centro de Comandos do DB2 (**Iniciar > Programas > IBM DB2 > Centro de Comandos**) e clique na guia **Script**.
2. Na janela Script, digite o seguinte:

```
connect to your_database_name;
CREATE TABLE Bonus (MEMBERID BIGINT NOT NULL,
    BONUSPOINT INTEGER NOT NULL,
    constraint p_memberid primary key (MEMBERID),
    constraint f_memberid foreign key (MEMBERID)
    references users (users_id) on delete cascade)
```

   em que *your_database_name* é o nome do seu banco de dados. Clique no ícone Executar.
   A tabela Bonus está criada agora.

   Nota: Você deve emitir o seguinte comando antes de criar a tabela Bonus, caso ninguém tenha executado anteriormente esse exemplo utilizando esse banco de dados:

```
drop table Bonus
```

Oracle Se você estiver utilizando um banco de dados Oracle, faça o seguinte para criar a tabela:

1. Abra a janela de comando Oracle SQL Plus (**Iniciar > Programas > Oracle > Desenvolvimento de Aplicativos > SQL Plus**).
2. No campo **Nome do Usuário**, digite seu nome de usuário do Oracle.

3. No campo **Senha**, digite a senha do Oracle.
4. No campo **Cadeia do Host**, digite sua cadeia de conexão.
5. Na janela SQL Plus, digite a seguinte instrução SQL:

```
CREATE TABLE Bonus (MEMBERID NUMBER NOT NULL,
    BONUSPOINT INTEGER NOT NULL,
    constraint p_memberid primary key (MEMBERID),
    constraint f_memberid foreign key (MEMBERID)
    references users (users_id) on delete cascade);
```

   e pressione Enter para executar a instrução SQL. A tabela BONUS está criada.

   Nota: Você deve emitir o seguinte comando antes de criar a tabela Bonus, caso ninguém tenha executado anteriormente esse exemplo utilizando esse banco de dados:

```
drop table Bonus;
```

6. Digite o seguinte para consolidar as alterações no banco de dados:

```
commit;
```

   e pressione Enter para executar a instrução SQL.

Criando o Bean de Entidade BonusBean

Após o banco de dados ter sido criado, você está pronto para começar a criar o novo bean de entidade. As etapas a seguir utilizam o VisualAge for Java. Para criar o novo bean de entidade, proceda da seguinte forma:

1. Crie um grupo EJB. Um grupo EJB é um grupo lógico que permite organizar seus beans corporativos. Você pode executar operações globais em um grupo EJB que se repete em todos os beans corporativos que residem nesse grupo. Por exemplo, se você selecionar um grupo EJB para exportar para um arquivo EJB JAR, todos os beans corporativos no grupo serão exportados.
   Neste tutorial, você cria um grupo EJB para organizar todos os beans corporativos relacionados à personalização da tabela Bonus.
   Para criar seu grupo EJB, proceda da seguinte forma:
   a. No Workbench, clique na guia **EJB**.
   b. Do menu **EJB**, selecione **Incluir> Grupo EJB**.
      Incluir SamrtGuide do Grupo EJB é aberto.
   c. No campo **Projeto**, digite `_WCSamplesEntityBeansProject`.

      Nota: O código do bean de entidade deve ser armazenado no seu próprio projeto, para objetivos de implementação.

   d. No campo **Criar um novo grupo EJB**, digite `WCSSamplesEntityBeans` e clique em **Concluir**.

2. Crie seu novo bean de entidade.
   Para criar seu novo bean de entidade, proceda da seguinte forma:
   a. Na área de janela Beans Corporativos, Clique com o botão direito do mouse no grupo EJB **WCSSamplesEntityBeans** e selecione **Incluir > Bean Corporativo**.
      O SmartGuide Criar Bean Corporativo é aberto.
   b. Digite as seguintes informações

Atributo	Valor
Nome do Bean	Bonus **Nota:** A convenção de nomenclatura é chamar o bean de entidade pelo mesmo nome da tabela que ele acessa.
Tipo do Bean	O bean de entidade com campos persistência gerenciada pelo contêiner (CMP)
Criar uma nova classe de bean	enable
Projeto	_WCSamplesEntityBeansProject
Pacote	com.ibm.commerce.sample.objects
Nome da Classe	BonusBean
Superclasse	com.ibm.commerce.base.objects.ECEntityBean

   e clique em **Avançar**.
   c. Clique no botão **Incluir** ao lado da caixa de texto **Incluir campos CMP no bean** para incluir um campo da coluna MEMBERID na tabela BONUS.
      O SmartGuide Criar Campos CMP é aberto.
   d. Digite as seguinte informações:

Atributo	Valor
Nome do Campo	memberId
Tipo de Campo	Long **Nota:** Você deve utilizar o tipo de dados *Long*, não *long*.
Campo-Chave	ativar

   e clique em **Concluir**.
   e. Clique em **Incluir** novamente para incluir um campo da coluna BONUSPOINT na tabela BONUS.
   f. Crie outro campo com as seguintes informações:

Atributo	Valor
Nome do Campo	bonusPoint

Atributo	Valor
Tipo de Campo	Integer **Nota:** Você deve utilizar o tipo de dados *Integer*, não *int*.
Acessar com os métodos getter e setter	ativar
Promover os métodos getter e setter para a interface remota	ativar

em seguida clique em **Concluir** na janela Criar Campo CMP.

g. Proteja esse bean corporativo utilizando controle de acesso, procedendo da seguinte maneira:
   1) Clique em **Incluir** ao lado de **Quais interfaces a interface remota deve estender?**.
   2) Digite `com.ibm.commerce.security.Protectable` no campo **Padrão** e clique em **Incluir**. Clique em **Fechar** para fechar a janela.

h. Clique em **Concluir** novamente.

A entidade Bonus é criada como um bean corporativo.

3. Defina o nível de isolamento do bean de entidade procedendo da seguinte maneira:
   a. Clique com o botão direito do mouse no bean **Bonus** e selecione **Propriedades**.
   b. Na lista suspensa **Nível de Isolamento**, selecione **TRANSACTION_READ_COMMITTED** e clique em **OK**.
4. Quando você cria um novo bean corporativo, o VisualAge for Java gera um campo EntityContext, bem como os métodos getEntityContext() e setEntityContext(EntityContext) correspondentes no bean. Seguindo o modelo de programação do WebSphere Commerce, o novo bean estende a classe com.ibm.commerce.base.objects.ECEntityBean e o ECEntityBean fornece sua própria implementação desse campo e desses métodos. Como EntityContext, getEntityContext e setEntityContext não devem ser substituídos, você deve agora excluir o campo e os métodos gerados do bean.
   Para excluir o campo EntityContext e seus métodos getter e setter, proceda da seguinte maneira:
   a. No painel Tipos, selecione a classe **BonusBean**.
      O painel Membros exibe os campos e métodos dessa classe.

      Nota: Se a área de janela Tipos não estiver visível, clique no ícone C/I (classe/interface) na área de janela Propriedades. A área de janela Tipos é aberta.

   b. No painel Membros, proceda da seguinte maneira:

1) Clique com o botão direito do mouse no campo **entityContext** e selecione **Excluir**.
2) Clique com o botão direito do mouse no campo **getEntityContext()** e selecione **Excluir**.
3) Clique com o botão direito do mouse no método **setEntityContext(EntityContext)** e selecione **Excluir**.

c. Salve o trabalho (Ctrl+S).

5. Inclua um novo método getMemberId no bean corporativo, procedendo da seguinte maneira:
   a. Clique com o botão direito do mouse na classe **BonusBean** e selecione **Incluir > Método**.
      O SmartGuide Criar Método é aberto.
   b. Crie o novo método utilizando os valores especificados na seguinte tabela. Se precisar de informações mais detalhadas sobre a criação de um novo método, consulte "Criação de métodos" na página 230.

Tabela 11.

Nome do Atributo	**Valor**
Nome do Método	`getMemberId`
Tipo de Retorno	`Long`
Parâmetros	nenhum
Exceções	nenhum

   c. Quando o novo método for gerado, exiba o código fonte.
   d. Por padrão, o método contém o seguinte código:

```
return null;
```

      Altere esse código para

```
return memberId;
```

   e. Inclua o novo método na interface remota clicando com o botão direito do mouse no método **getMemberId** e selecionando **Incluir em > Interface Remota EJB**.

6. Inclua novos campos FinderHelper no BonusBeanFinderHelper.
   Essa interface contém uma cláusula de pesquisa que corresponde a um método FinderHelper que será criado na próxima etapa. Para incluir os campos FinderHelper, proceda da seguinte forma:
   a. Na área de janela Tipos, clique na interface **BonusBeanFinderHelper**.
   b. Modifique o código na área de janela Origem para que tenha a seguinte aparência:

```
public interface BonusBeanFinderHelper {
    public static final String
                    findByMemberIdWhereClause = " (MEMBERID = ?) ";
        }
```

Nota: A sintaxe de "WhereClause" é muito importante; ela deve corresponder ao nome do método utilizado para o método FinderHelper. Nesse caso, "findByMemberId" em `findByMemberIdWhereClause` corresponde exatamente ao nome do método que você criará na etapa a seguir `findByMemberId`).

7. Inclua os novos métodos FinderHelper na interface BonusHome.
   Para incluir os novos métodos FinderHelper, proceda da seguinte forma:
   a. Na área de janela Tipos, clique com o botão direito do mouse na interface **BonusHome** e selecione **Incluir > Método**.
      O SmartGuide Criar Método é aberto.
   b. Selecione **Criar um novo método** e clique em **Avançar**.
   c. No campo **Nome do Método**, digite `findByMemberId`.
   d. No campo **ReturnType**, digite `Bonus`.
   e. Clique em **Incluir** próximo a **Quais parâmetros esse método deve ter?**
      A janela Parâmetros é aberta.
   f. No campo **Nome**, digite `argMemberId`.
   g. Selecione **Tipos de Referência** e digite `Long`. Clique em **Incluir**, em seguida, **Fechar**.
   h. Clique em **Avançar**.
   i. Clique em **Incluir** próximo ao campo **Quais exceções este método pode emitir?**, digite `RemoteException` no campo **Padrão** e clique em **Incluir**. Isto inclui a exceção `java.rmi.RemoteException` na janela Atributos. (A janela Atributos pode ser posicionada atrás da janela Exceções).
   j. No campo **Padrão** da janela Exceções, digite `FinderException`, clique em **Incluir** e em seguida, clique em **Fechar**. A exceção `javax.ejb.FinderException` é listada na janela Atributos.
   k. Clique em **Concluir**.
8. Inclua um novo método `ejbCreate` no EJB. Este método é promovido para a interface inicial, deste modo ele está disponível em um bean de acesso gerado. Para criar esse método, proceda da seguinte forma:
   a. Selecione a classe **BonusBean** na área de janela Tipos.
   b. Clique em **ejbCreate(Long)** no painel Membros.
   c. Modifique o código para que corresponda ao seguinte:

```
public void ejbCreate(java.lang.Long argMemberId,
  Integer argBonusPoint)
  throws javax.ejb.CreateException, java.rmi.RemoteException {
  _initLinks();
```

```
// All CMP fields should be initialized here.
 memberId=argMemberId;
        bonusPoint=argBonusPoint;
}
```

d. Salve o código e o VisualAge for Java criará um novo método, chamado **ejbCreate(Long, Integer)** na área de janela Membros.

e. Clique com o botão direito do mouse no método **ejbCreate(Long)** original e selecione **Excluir**.

9. Inclua o novo método na interface inicial. Isso disponibilizará o método na classe de bean de acesso. Para fazer isso, proceda da seguinte forma:

   a. Clique com o botão direito do mouse no método **ejbCreate(Long, Integer)** na classe BonusBean e selecione **Incluir > Interface Inicial EJB**.

10. Atualize o método getOwner, fazendo o seguinte:

    a. Selecione a classe **BonusBean** na área de janela Tipos.

    b. Clique no método **getOwner()** no painel Membros.

       Nota: Se você selecionou para exibir os métodos herdados, você verá dois métodos getOwner(). Um foi herdado da classe ECEntityBean. Este não é aquele que deve ser selecionado nesta etapa. Certifique-se de selecionar o método getOwner específico para a classe BonusBean.

    c. O código fonte do método getOwner aparece como a seguir:

```
public Long getOwner()
  throws Exception, java.rmi.RemoteException
   {
return null;   }
```

       Você deve alterar o valor retornado pelo método. A parte de código que deve ser alterada é mostrada em negrito no seguinte:

```
public Long getOwner()
  throws Exception, java.rmi.RemoteException
   {
      return getMemberId();
   }
```

       Salve seu trabalho.

    d. Clique no método **fulfills(Long, String)** no painel Membros.

       Nota: Se tiver selecionado para exibir os métodos herdados, você verá dois métodos fulfills(Long, String). Um foi herdado da classe ECEntityBean. Este não é aquele que deve ser selecionado nesta etapa. Certifique-se de selecionar o método fulfills(Long, String) específico para a classe BonusBean.

e. O código fonte do método fulfills(Long, String) aparece como a seguir:

```
public boolean fulfills(Long member, String relationship)
  throws Exception, java.rmi.RemoteException
  {
      return false;
  }
```

Você deve especificar o relacionamento que o usuário deve preencher. Para isso, você deverá alterar a parte do código mostrada em negrito no seguinte:

```
public boolean fulfills(Long member, String relationship)
  throws Exception, java.rmi.RemoteException
  {

      if (relationship.equalsIgnoreCase("creator"))
      {
         return member.equals(getMemberId());
      }      return false;
  }
```

Salve seu trabalho.

11. Mapeie a tabela do banco de dados BONUS para BonusBean. A primeira etapa no mapeamento do esquema do banco de dados para a entidade BonusBean envolve a utilização de ferramentas no VisualAge for Java para criar o esquema do banco de dados. Faça o seguinte para criar o esquema:
    a. Na janela Área de Trabalho, do menu **EJB**, selecione **Abrir em > Esquemas do Banco de Dados**.
    b. Do menu **Esquemas**, selecione **Importar/Exportar Esquema > Importar Esquema do Banco de Dados**.
    A janela Informações necessárias é aberta.
    c. No campo **Nome do Esquema**, digite `WCSSamples`, e clique em **OK**.
    A janela Informações da Conexão do Banco de Dados é aberta.
    d. Preencha as seguintes informações:

Atributo	DB2 **Valor do DB2**	Oracle **Valor do Oracle**
Tipo da Conexão	`COM.ibm.db2.jdbc.app.DB2Driver`	`Oracle.jdbc.driver.OracleDriver`
Origem de Dados	`jdbc:db2:`*`wcs_db_name`*	`jdbc:oracle:thin:@`*`hostname`*`:`*`port`*`:`*`SID`*
Nome do Usuário	*`wcs_db_user_name`*	*`wcs_db_user_name`*
Senha	*`wcs_db_password`*	*`wcs_db_password`*

com valores substituídos da seguinte forma:

- DB2 *wcs_database_name* é o nome de seu banco de dados do WebSphere Commerce
- Oracle *hostname* é o nome do host do Oracle
- Oracle *port* é o número da porta do banco de dados do Oracle (por exemplo, 1521).
- *wcs_db_user_name* é o nome do usuário para o banco de dados.
- *wcs_db_password* é a senha do banco de dados.

Clique em **OK**.
A janela Tabelas Selecionadas é aberta.

e. Da lista **Qualificadores**, selecione seu qualificador de banco de dados (pode ser o nome do usuário do banco de dados ou o nome da máquina) e clique em **Construir Lista de Tabelas**. Uma lista das tabelas de banco de dados disponíveis é carregada.

f. Selecione **Bonus** do painel Tabelas e clique em **OK**. Espere alguns momentos.

g. No Navegador do Esquema, clique na tabela incluída recentemente para ver se as colunas da tabela aparecem.

h. Clique com o botão direito do mouse na tabela **Bonus** e selecione **Editar Tabela**.
O Editor de Tabelas é aberto.

i. Remova quaisquer entradas do campo **Qualificador**. É recomendável remover as informações do qualificador para que o código possa ser implementado em outras máquinas utilizando um banco de dados diferente.

j. Oracle Modifique os tipos de dados da coluna, da seguinte maneira:

1) Selecione a coluna **MEMBERID**, em seguida, clique em **Editar**. Na lista suspensa Tipo, selecione **BIGINT** e clique em **OK**.

2) Selecione a coluna **BONUSPOINT**, em seguida, clique em **Editar**. Na lista suspensa Tipo, selecione **INTEGER** e clique em **OK**.

k. Clique em **OK** para sair do Editor de Tabela.

l. Do menu **Esquemas**, selecione **Salvar Esquema**.
A janela Salvar Esquema é aberta.

m. Digite as seguintes informações:

Atributo	Valor
Projeto	`_WCSamplesEntityBeansProject`
Pacote	`com.ibm.commerce.sample.objects`
Nome da Classe	`WCSSamplesSchema`

em seguida, clique em **Concluir** e feche o Navegador do Esquema.

12. Após o esquema ter sido criado, você pode criar o mapa do esquema. Para criar esse mapa entre a tabela BONUS e a entidade BonusBean, proceda da seguinte forma:
    a. Do menu **EJB**, selecione **Abrir em > Mapas de Esquema**. O Navegador do Mapa é aberto.
    b. No Navegador do Mapa, do menu **Mapas da Memória de Dados**, selecione **Novo Mapa do Grupo EJB**.
    A janela Novo Mapa da Memória de Dados é aberta.
    c. Preencha as seguintes informações:

Atributo	**Valor**
Nome	`Amostras de WCS`
Grupo EJB	`WCSSamplesEntityBeans`
Esquema	`WCSSamples`

    e clique em **OK**.
    d. No painel Mapas da Memória de Dados, clique em **Amostras de WCS**.
    e. No painel Classes Persistentes, clique em **Bonus**.
    f. No menu **Mapas da Tabela**, selecione **Novo Mapa da Tabela > Incluir Mapa da Tabela Sem Herança**.
    g. Na lista suspensa **Tabela**, selecione **Bonus** e clique em **OK**.
    h. No painel Mapas da Tabela, selecione **Bonus**, em seguida, clique com o botão direito do mouse nele e selecione **Editar Mapas da Propriedade**.
    O Editor de Mapa de Propriedade é aberto.
    i. Defina os atributos da seguinte forma:

Atributos de Classe	**Tipo de Mapa**	**Coluna da Tabela**
memberId	Simples	MEMBERID
bonusPoint	Simples	BONUSPOINT

    e clique em **OK**.
    j. Do menu **Mapas da Memória de Dados**, selecione **Salvar Mapa da Memória de Dados**.
    Salvar Mapa da Memória de Dados é aberto.
    k. Digite as seguintes informações:

Atributo	**Valor**
Projeto	`_WCSamplesEntityBeansProject`
Pacote	`com.ibm.commerce.sample.objects`

Atributo	Valor
Nome da Classe	`WCSSamplesMap`

em seguida, clique em **Concluir** e feche o Navegador do Mapa.

13. Após a entidade BonusBean ter sido criada e o esquema estar corretamente mapeado, será necessário criar um bean de acesso para o bean de entidade. Esse bean de acesso simplifica para os aplicativos o acesso às informações contidas no bean de entidade Bonus. As ferramentas no VisualAge for Java são utilizadas para gerar esse bean de acesso, com base na entidade que já foi criada (em particular, apenas os métodos que foram criados para a interface remota serão utilizados pelo bean de acesso). Para criar o bean de acesso para seu bean de entidade Bonus, proceda da seguinte forma:
    a. No Workbench, com a guia EJB selecionado, clique com o botão direito do mouse no bean corporativo **Bonus** e selecione **Incluir > Bean de Acesso**.
    A janela de SmartGuide Criar Bean de Acesso é aberta (isso pode levar algum tempo).
    b. Certifique-se de que as seguinte informações sejam digitadas:

Atributo	Valor
Grupo EJB	`WCSSamplesEntityBeans`
Bean Corporativo	`Bonus`
Nome do Bean de Acesso	`BonusAccessBean`
Tipo do Bean de Acesso	`Copiar Assistente para um Bean de Entidade`

e clique em **Avançar**.

    c. Na lista suspensa **Selecionar método de início para construtor de argumento zero**, selecione **findByMemberId(Long)**.
    d. Para **init_argMemberId**, (na coluna Propriedades Iniciais), defina **Conversor** como `com.ibm.commerce.base.objects.WCSStringConverter` e clique em **Avançar**.
    e. Para bonusPoint, assegure-se de que **CopyHelper** esteja selecionado, defina o valor de **Conversor** como `com.ibm.commerce.base.objects.WCSStringConverter` e clique em **Concluir**.

    Nota: Não será necessário fazer modificações no campo memberId.

    f. Clique em **OK** quando a mensagem "Geração de Código Concluída" for exibida.

Você pode exibir o código recém-gerado, mudando para a guia **Projetos**, expandindo o projeto **_WCSamplesEntityBeansProject** e expandindo **com.ibm.commerce.sample.objects**. A nova classe chamada `BonusAccessBean` é exibida dentro do pacote.

14. A próxima etapa é gerar o código implementado.
    O utilitário de geração de código analisa os beans para garantir que as especificações EJB da Sun Microsystems sejam correspondidas e assegura que as regras específicas do servidor EJB sejam seguidas. Além disso, para cada bean corporativo selecionado, a ferramenta de geração de código gera as implementações EJBObject (remota) e de início e as classes de implementação para as interfaces remotas e de início, bem como as classes de persistidor e localizador JDBC para os beans CMP. Gera também o Java ORB, stubs e as classes tie obrigatórias para acesso RMI sobre IIOP, bem como os stubs das interfaces inicial e remota.
    Se você selecionou um grupo EJB contendo um bean corporativo CMP, ou se selecionou um bean corporativo CMP individual, os itens a seguir também serão gerados:
    - Uma cadeia de tabela de criação que é gerada na classe do persistidor.
    - Uma implementação do persistidor que mapeia para, e a partir da, tabela

    Para gerar o código implementado, proceda da seguinte forma:
    a. Com a guia EJB selecionada, no painel Beans Corporativo, clique com o botão direito do mouse no bean corporativo **Bonus** e selecione **Gerar Código Implementado**. A geração do código leva alguns minutos.
15. Antes de testar o bean corporativo Bonus, você deve criar um novo servidor EJB que contenha todos os grupos EJB do WebSphere Commerce e também o novo grupo EJB WCSSamplesEntityBeans. Estes grupos devem ser executados no mesmo servidor para que o escopo de transação seja mantido. Uma vez criado o novo servidor, você deve iniciá-lo.
    Para criar e iniciar o novo servidor EJB, proceda da seguinte forma:
    a. Assegure-se de que todos os grupos EJB estejam com reduzidos.
    b. Na área de janela Beans Corporativos, selecione todos os grupos EJB do WebSphere Commerce e seu novo grupo EJB (ou seja, selecione todos os grupos EJB iniciando com `WCS`).
    c. Com esses grupos EJB selecionados, clique com o botão direito do mouse e selecione **Incluir em > Configuração do Servidor**.
    A janela Configuração do Servidor EJB é aberta (isso pode levar algum tempo).
    d. Clique com o botão direito do mouse no Servidor EJB criado recentemente {por exemplo **Servidor EJB (server2)**}, selecione **Propriedades** e preencha o seguinte:

Atributo	Valor do DB2 DB2	Oracle Valor do Oracle
Origem de Dados	WebSphere Commerce DB2 DataSource *instance_name*	WebSphere Commerce Oracle DataSource *instance_name*
Tipo da Conexão	<DataSource>	<DataSource>
Nome do Usuário	*wcs_db_user_name*	*wcs_db_user_name*
Senha	*wcs_db_password*	*wcs_db_password*
Tempo limite da transação	1200	1200
Tempo limite de inatividade da transação	600000	600000

e clique em **OK**.

Nota: O valor da Origem de Dados deve corresponder ao valor da origem de dados especificada no arquivo *instance_name*.xml.

Dependendo do hardware de sua máquina de desenvolvimento (por exemplo, velocidade do processador) pode ser necessário que você aumente os valores para as propriedades do servidor EJB **Tempo Limite da Transação** e **Inatividade da Transação**.

e. Se o WebSphere Test Environment estiver em execução, pare-o, bem como o servidor de nomes persistente e outro servidor EJB, conforme descrito no Apêndice A, "Iniciando e Parando o WebSphere Test Environment" na página 347.

f. Inicie o servidor de nomes persistente, conforme descrito em "Iniciando e Parando o Servidor de Nomes Persistente" na página 347.

g. Inicie o novo servidor EJB, conforme descrito em "Iniciando e Parando o Servidor EJB" na página 348.

16. Após o servidor EJB ser iniciado, você poderá iniciar o cliente de teste. Utilizando o cliente de teste, você cria um novo registro no banco de dados. Para iniciar o cliente de teste e criar esse registro, faça o seguinte:

a. Com a guia EJB selecionado, clique com o botão direito do mouse no bean corporativo **Bonus** e selecione **Executar cliente de teste**.

b. Na janela Procura de EJB, clique em **Procurar**.
A janela Bonus é aberta.

c. Clique em **create(Long, Integer)**. No painel Detalhes, preencha o seguinte:

Atributo	Valor
Long	–1000
Integer	100

e clique no ícone Invocar na janela Cliente de Teste do EJB.

Nota: O primeiro atributo deve corresponder ao memberID para todo usuário registrado. Você pode determinar os memberIds consultando a tabela USERS.

17. Verifique se o registro do banco de dados foi criado corretamente, consultando diretamente o banco de dados.

    DB2 Se estiver utilizando um banco de dados DB2, faça o seguinte:

    a. Abra o Centro de Comandos do DB2 (**Iniciar > Programas > IBM DB2 > Centro de Comandos**)
    b. Selecione a guia **Interativo**.
    c. Digite `connect to` *`wcs_database_name`* e clique no ícone Executar.
    d. Digite `SELECT * FROM Bonus` e clique no ícone Executar.
       Deverão ser retornados os seguintes valores

Coluna	Valor
MEMBERID	-1000
BONUSPOINT	100

O registro mostrado acima foi criado pelo cliente de teste EJB.

Oracle Se você estiver utilizando um banco de dados Oracle, faça o seguinte:

a. Abra a janela de comando Oracle SQL Plus (**Iniciar > Programas > Oracle - OraHome81> Desenvolvimento de Aplicativos > SQL Plus**).
b. No campo **Nome do Usuário**, digite seu nome de usuário do Oracle.
c. No campo **Senha**, digite sua senha do Oracle.
d. No campo **Cadeia do Host**, digite sua cadeia de conexão.
e. Na janela SQL Plus, digite o seguinte:

```
select * from BONUS;
```

e pressione Enter para executar a instrução SQL.
Os seguintes valores deverão ser retornados

Coluna	Valor
MEMBERID	-1000
BONUSPOINT	100

18. Utilize o cliente de teste para verificar se o bean corporativo Bonus pode acessar com êxito o registro do banco de dados, fazendo o seguinte:

a. Na janela Bonus, selecione a guia **Início**.
b. No painel Métodos, clique em **findByMemberId(Long)**.
c. No campo **Long**, digite `–1000` e clique no ícone Chamar.
d. Com a guia **Remoto** selecionada, expanda **Métodos**, clique em **getBonusPoint** e clique no ícone Invocar.
   O painel Detalhes mostra um resultado de número inteiro `100`.
e. Feche as janelas Bonus e Cliente de Teste EJB.

Integrar o Bean de Entidade Bonus com MyNewControllerCmd

Na seção anterior, você testou o novo bean de entidade Bonus, utilizando o cliente de teste que foi gerado com o VisualAge for Java. Nessa seção, você integra o bean de entidade Bonus com a lógica MyNewControllerCmd. Após o código Java ser atualizado, o modelo Sample.jsp é atualizado para criar uma interface que permite atualizações no saldo de pontos de bônus de um comprador.

A integração do bean de entidade Bonus envolve as seguintes etapas de alto nível:

1. Modifique o método performExecute da classe MyNewTaskCmdImpl para calcular os novos pontos de bônus e salvar os pontos na tabela BONUS.
2. Inclua um método getResources na classe MyNewControllerCmdImpl para retornar uma lista de recursos utilizados pelo comando. Esse método é incluído por motivos de controle de acesso.
3. Crie uma nova política de controle de acesso para os novos recursos.
4. Modifique o DataBeanSampleBean para expandir do bean de acesso para o bean de entidade Bonus. Tendo o bean de dados expandido do bean de acesso, todos os atributos do bean de acesso serão herdados pelo bean de dados.
5. Modifique os métodos no DataBeanSampleBean.
6. Modifique o caminho da classe do mecanismo de servlet no WebSphere Test Environment para incluir o novo _WCSamplesEntityBeansProject.
7. Modifique o modelo Sample.jsp para permitir que os usuários digitem os pontos de bônus e exibam os resultados.

Modificar MyNewTaskCmdImpl para cálculo de ponto de bônus:
MyNewTaskCmdImpl é utilizado como o ponto de integração do bean de entidade Bonus e MyNewControllerCmd (desde que MyNewControllerCmd invoque MyNewTaskCmd).

Para modificar MyNewTaskCmdImpl para que execute o cálculo de ponto de bônus, proceda da seguinte forma:

1. Na janela Workbench do VisualAge for Java, expanda o projeto **_WCSamples**.

2. Expanda o pacote **com.ibm.commerce.sample.commands** e em seguida, selecione a classe **MyNewTaskCmdImpl** para exibir seu código fonte.
3. Retire o comentário da seguinte instrução de importação:

```
import com.ibm.commerce.sample.objects.*;
```

   Salve seu trabalho (**Ctrl + S**).
4. Selecione o método **performExecute** da classe **MyNewTaskCmdImpl**.
5. No código fonte do método performExecute, retire o comentário da Seção 3. Isso introduzirá o seguinte código no método:

```
// use BonusAccessBean to update new bonus point

String newBonusPoint = null;
BonusAccessBean bb = new BonusAccessBean();
try {
     if (refNum != null) {
          bb.setInit_argMemberId(refNum);
          bb.refreshCopyHelper();
          oldBonusPoint = bb.getBonusPoint();
     }
} catch (javax.ejb.FinderException e) {
     try {
          bb = new BonusAccessBean(new Long(refNum),new Integer(0));
          oldBonusPoint = "0";
     } catch (javax.ejb.CreateException ec) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
               this.getClass().getName(), "performExecute");
     } catch (javax.naming.NamingException ec) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
               this.getClass().getName(), "performExecute");
     } catch (java.rmi.RemoteException ec) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
               this.getClass().getName(), "performExecute");
     }
} catch (javax.naming.NamingException e) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
               this.getClass().getName(), "performExecute");
} catch (java.rmi.RemoteException e) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
               this.getClass().getName(), "performExecute");
} catch (javax.ejb.CreateException e) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
               this.getClass().getName(), "performExecute");
}


try {
     if (oldBonusPoint != null) {
          int newBP =  Integer.parseInt(oldBonusPoint) + getTask_input2();
          newBonusPoint = Integer.toString( newBP );
          bb.setBonusPoint( newBonusPoint )   ;
          newBonusPoint=bb.getBonusPoint();
```

```
            bb.commitCopyHelper();
        }
    } catch (javax.ejb.FinderException e) {
        throw new ECSystemException(ECMessage._ERR_FINDER_EXCEPTION,
                  this.getClass().getName(), "performExecute");
    } catch (javax.naming.NamingException e) {
        throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
                  this.getClass().getName(), "performExecute");
    } catch (java.rmi.RemoteException e) {
        throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
                  this.getClass().getName(), "performExecute");
    } catch (javax.ejb.CreateException e) {
        throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
                  this.getClass().getName(), "performExecute");
    }
```

Salve seu trabalho.

Incluir Método getResources na Classe MyNewControllerCmdImpl: Nesta seção, você inclui um novo método getResources na classe MyNewControllerCmdImpl. Esse método retorna uma lista de recursos que o comando utiliza durante o processamento. Esse método é obrigatório para controle de acesso no nível de recurso.

Para incluir o método getResources, proceda da seguinte maneira:

1. Com a guia **Projetos** selecionada, expanda o projeto **_WCSamples**.
2. Expanda o pacote **com.ibm.commerce.sample.commands**.
3. Selecione a classe **MyNewControllerCmdImpl** para exibir seu código fonte.
4. No código fonte, remova o comentário da seção de controle de acesso. Esta seção aparece como mostrada no seguinte fragmento de código:

```
public AccessVector getResources() throws ECException {

// use UserRegistryAccessBean to check member reference number

   String refNum;
   String methodName="getResources";

   com.ibm.commerce.user.objects.UserRegistryAccessBean rrb =
      new com.ibm.commerce.user.objects.UserRegistryAccessBean();

     try {
      rrb = rrb.findByUserLogonId(getInputString());
          refNum = rrb.getUserId();

   }
   catch (javax.ejb.FinderException e) {

      throw new ECSystemException(ECMessage._ERR_BAD_USER_NAME,
         this.getClass().getName(),methodName);
```

```
    }
    catch (javax.naming.NamingException e) {
      throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
       this.getClass().getName(), methodName);
    }
    catch (java.rmi.RemoteException e) {
      throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
       this.getClass().getName(), methodName);
    }
    catch (javax.ejb.CreateException e) {
      throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
       this.getClass().getName(), methodName);
    }


    //find the Bonus bean for this user
    String newBonusPoint = null;
    com.ibm.commerce.sample.objects.BonusAccessBean bb =
       new com.ibm.commerce.sample.objects.BonusAccessBean();
      try {
      if (refNum != null) {
           bb.setInit_argMemberId(refNum);
           bb.refreshCopyHelper();
       }
    }
  catch (javax.ejb.FinderException e) {

    // The user doesn't have a Bonus object so return the container that
    // will hold the bonus object when it's created

    return new AccessVector(rrb);

    }
    catch (javax.naming.NamingException e) {
      throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
       this.getClass().getName(), methodName);
    }

    catch (java.rmi.RemoteException e) {
      throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
       this.getClass().getName(), methodName);
    }

    catch (javax.ejb.CreateException e) {
      throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
       this.getClass().getName(), methodName);
    }

    return new AccessVector(bb);


  }
```

Salve seu trabalho. (Ctrl+S).

Após ter salvo a seção de código anterior, o VisualAge for Java separa os campos e os acessórios fora dessa exibição particular. Observe que agora eles aparecem sob a classe MyNewControllerCmdImpl e o método está marcado com um M.

Nota: Para simplificar neste tutorial, os objetos de recurso são criados nesse método getResources. Em um aplicativo real, é recomendável criar os objetos de recurso no método validateParameters e salvá-los como variáveis de instância. Desse modo, os objetos poderiam ser reutilizados pelos métodos getResources e performExecute.

Configurando a Política de Controle de Acesso para o Novo Recurso: Uma política de controle de acesso de exemplo é fornecida. Essa política cria os seguintes objetos de controle de acesso:

Uma ação
A ação que é criada é `com.ibm.commerce.sample.commands.MyNewControllerCmd`

Um grupo de ações
O grupo de ações criado é `MyNewControllerCmdActionGroup`. Esse grupo de ações contém apenas uma ação; `com.ibm.commerce.sample.commands.MyNewControllerCmd`

Uma categoria de recurso
A categoria de recurso criada é `com.ibm.commerce.sample.objects.BonusResourceCategory.` Essa categoria de recurso é para o bean de entidade Bonus.

Um grupo de recursos
O grupo de recursos criado é `BonusResourceGroup`. Esse grupo de recursos contém apenas a categoria de recurso anterior.

Uma política
A política criada é `AllUsersUpdateBonusResourceGroup`. Essa política permite que os usuários executem a ação MyNewControllerCmd no bean Bonus apenas se o usuário for o "proprietário" do objeto bonus. Por exemplo, se o usuário tiver efetuado logon como usuário `wcsadmin`, o usuário só poderá modificar os pontos de bônus para `wcsadmin`.

A configuração da política `AllUsersUpdateBonusResourceGroup` envolve as seguintes etapas:

1. Carregar o arquivo `SampleACPolicy.xml` utilizando o comando `acpload`.
2. Carregar a descrição do `SampleACPolicy_`***`locale`***`.xml` utilizando o comando `acpnlsload`.

3. Atualizar o registro da política. Observe que esta etapa será necessária apenas se o mecanismo de servlet estiver sendo executado na hora em que a política de controle de acesso for carregada.

Para configurar a política `AllUsersUpdateBonusResourceGroup`, proceda da seguinte maneira:

1. Em um prompt de comandos, vá para o diretório:
   unidade`:\WebSphere\CommerceServerDev\bin`
2. Para carregar o arquivo `SampleACPolicy.xml`, você deve emitir o comando `acpload`, que possui a seguinte forma:
   ```
   acpload db_name db_user db_password inputXMLFile
   ```
   em que
   - *db_name* é o nome do banco de dados
   - *db_user* é o nome do usuário do banco de dados
   - *db_password* é a senha do banco de dados
   - *inputXMLFile* é o nome do arquivo XML que contém a política

   Por exemplo, você pode emitir o seguinte comando:
   ```
   acpload VAJ_Demo user password SampleACPolicy.xml
   ```
3. Para carregar a descrição da política, você deve emitir o comando `acpnlsload`, que possui o seguinte formato:
   ```
   acpnlsload db_name db_user db_password inputXMLFile
   ```
   Por exemplo, você pode emitir o seguinte comando:
   ```
   acpnlsload VAJ_Demo user password SampleACPolicy_en_US.xml
   ```
4. Se o mecanismo de servlet para o WebSphere Test Environment estiver em execução no momento, pare e em seguida, reinicie-o para atualizar o registro da política.

Modificar DataBeanSampleBean para pontos de bônus: Nesta seção, você modifica o DataBeanSampleBean para estender o BonusAccessBean, procedendo da seguinte maneira:

1. No Workbench, expanda o pacote **com.ibm.commerce.sample.databeans**.
2. Inclua um novo campo no bean de dados, procedendo da seguinte maneira:
   a. Clique com o botão direito do mouse na classe **DataBeanSampleBean** e selecione **Incluir > Campo**. Utilizando o SmartGuide Criar Campo, crie novos campos conforme descrito nas seguintes etapas. Se você necessitar de informações adicionais sobre a criação de campos, consulte “Criação de novos campos” na página 220.
   b. Inclua um novo campo no bean de dados, utilizando os seguintes valores:

Nome do Atributo	Valor
Nome do Campo	`task_output_oldBonusPoint`
Tipo de Campo	`String`
Valor Inicial	Deixar em branco.
Modificadores de Acesso	`private`
Outros Modificadores	Deixar todos desmarcados.
Acessar com os métodos getter e setter	`checked`
Getter	`public`
Setter	`public`

Clique em **Concluir**.

3. Clique na classe **DataBeanSampleBean** para exibir seu código fonte. No código fonte, retire o comentário da seguinte instrução de importação:

   ```
   import com.ibm.commerce.sample.objects.*;
   ```

   Salve seu trabalho.

4. Ainda dentro do código fonte da classe DataBeanSampleBean, remova o comentário da Seção 1 e comente a Seção 2. Isso alterará o bean para que se estenda do BonusAccessBean. Após executar essas modificações, o código aparecerá da seguinte forma:

   ```
   /// Section 1 ////////////////////////////////////////////////

   // Extend the databean to BonusAccessBean

   public class DataBeanSampleBean
      extends com.ibm.commerce.sample.objects.BonusAccessBean
      implements SmartDataBean {


   ////////////////////////////////////////////////////////////////

   /// Section 2 ////////////////////////////////////////////////
   /*
   // Extend the databean to BonusAccessBean

        public class DataBeanSampleBean implements SmartDataBean {
   */

   //
   ////////////////////////////////////////////////////////////////
   ```

   Salve seu trabalho.

5. Selecione o método **setTask_output_userId(String)** para exibir seu código fonte. Localize a seguinte linha de código:

```
public void setTask_output_userId(java.lang.String
      newTask_output_userId) {
      task_output_userId = newTask_output_userId;
```

Após a linha precedente, digite o seguinte código para instanciar o novo BonusAccessBean:

```
//////////////////////////////////////
// Section A : instantiate BonusAccessBean

if (task_output_userId != null)
     this.setInit_argMemberId(newTask_output_userId);

//////////////////////////////////////
```

Salve seu trabalho.

6. Selecione o método **populate()** para exibir seu código fonte. Retire o comentário da Seção 2 para instanciar o BonusAccessBean. Isso introduzirá o seguinte código no método:

```
setTask_output_oldBonusPoint(getRequestProperties().getString(
    "task_output_oldBonusPoint"));
```

Modificar o caminho da classe: Antes de poder testar o comando modificado no WebSphere Test Environment, deverá modificar o caminho da classe para incluir o novo projeto que foi criado para o novo bean de entidade Bonus. Para modificar esse caminho da classe, proceda da seguinte forma:

1. Do menu **Área de Trabalho** em VisualAge for Java, selecione **Ferramentas > WebSphere Test Environment**.
   O WebSphere Test Environment Control Center é aberto.
2. Clique em **Mecanismo de Servlet**.
3. Se o Mecanismo do Servlet estiver em execução, clique em **Parar Mecanismo de Servlet** e em **Editar Caminho da Classe**.
4. Clique em **Selecionar Tudo** e em seguida, clique em **OK**.

Modificar o modelo Sample.jsp para pontos de bônus: Para modificar o modelo de exibição, proceda da seguinte forma:

1. Abra os arquivos `Sample.jsp` e `Sample_All.jsp` em um editor de texto.
2. Copie a Seção 4 do código a partir do Sample_All.jsp no Sample.jsp entre os marcadores `<!-- SECTION 4 -->` e `<!-- END OF SECTION 4 -->`. O código a seguir é introduzido no modelo JSP

```
<!-- SECTION 4 -->

<h1>
Bonus Administration
</h1>

<%
```

```
if (userId != null) {
%>

<B>
<UL>
   <LI> The bonus point before update is
      <%=testBean.getTask_output_oldBonusPoint()%>
   <LI> The bonus point after update is
      <%=testBean.getBonusPoint()%>
</UL>
</B>

<%
}
%>

<br>
<B>Input to command:</B><P>

<FORM NAME=Bonus ACTION="MyNewControllerCmd">
<TABLE>

                <TD>
            <B>Logon ID </B>
         </TD>
        <TD>
             <input type=text name=input1
                 value='<%=testBean.getInput1()%>'>
        </TD>
   </TR>
   <TR>
         <TD>
             <B>Bonus Point</B>
         </TD>
         <TD>
              <input type=text name=input2>
         </TD>
     </TR>
     <TR>
          <TD COLSPAN=2>
               <input type=submit>
          </TD>
     </TR>
</TABLE>
</FORM>

<!-- END OF SECTION 4 -->
```

Salve o arquivo `Sample.jsp`.

3. Como o novo bean Bonus fica protegido sob o controle de acesso e os usuários só podem executar a ação MyNewControllerCmd em um bean que eles possuem, o usuário deve efetuar login. Desse modo, você utilizará o recurso de login na loja de exemplo para permitir que o usuário efetue

login. Isso requer que você copie o arquivo Sample.jsp para a estrutura de diretórios da loja, pois quando você efetua logon na loja, o controlador da Web procura o arquivo Sample.jsp no diretório da loja. Copie o arquivo Sample.jsp de:
vaj_drive:\VAJava\Ide\project_resources\IBM WebSphere Test Environment \hosts\default_host\default_app\web
para
vaj_drive:\VAJava\Ide\project_resources\IBM WebSphere Test Environment \hosts\default_host\default_app\web*store_directory*.

Nota: Para a loja de exemplo, o valor para *store_directory* é InFashion.

4. Certifique-se de que o WebSphere Test Environment esteja em execução (consulte o Apêndice A, "Iniciando e Parando o WebSphere Test Environment" na página 347).
5. Efetue login como o usuário wcsadmin, fazendo o seguinte:
   - Digite a seguinte URL em um navegador:
     ```
     http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
         storeId=store_Id&catalogId=catalog_Id&langId=-1
     ```
   - Clique no link **Registro**.
     A página Registro ou Login é exibida.
   - No campo **Endereço de e-mail**, digite wcsadmin.
   - No campo **Senha**, digite a senha para o usuário wcsadmin e em seguida, clique em **Login**.

     Nota: A senha original era wcsadmin, mas isso foi alterado quando o comando contractPublish foi executado como parte do processo de instalação. Esta etapa é descrita no seguinte documento:
     - Business *WebSphere Commerce Studio Business Developer Edition - Manual de Instalação*
     - Professional *WebSphere Commerce Studio Professional Developer Edition - Manual de Instalação*
6. Depois de concluir o login, digite a seguinte URL no mesmo navegador:
   ```
   http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
       input1=wcsadmin&input2=1000
   ```

   É apresentada uma página que contém todos os parâmetros de saída anteriores e também um novo formulário que permite atualizar o balanço do ponto do bonus para um usuário. A página exibida é semelhante à seguinte captura de tela:

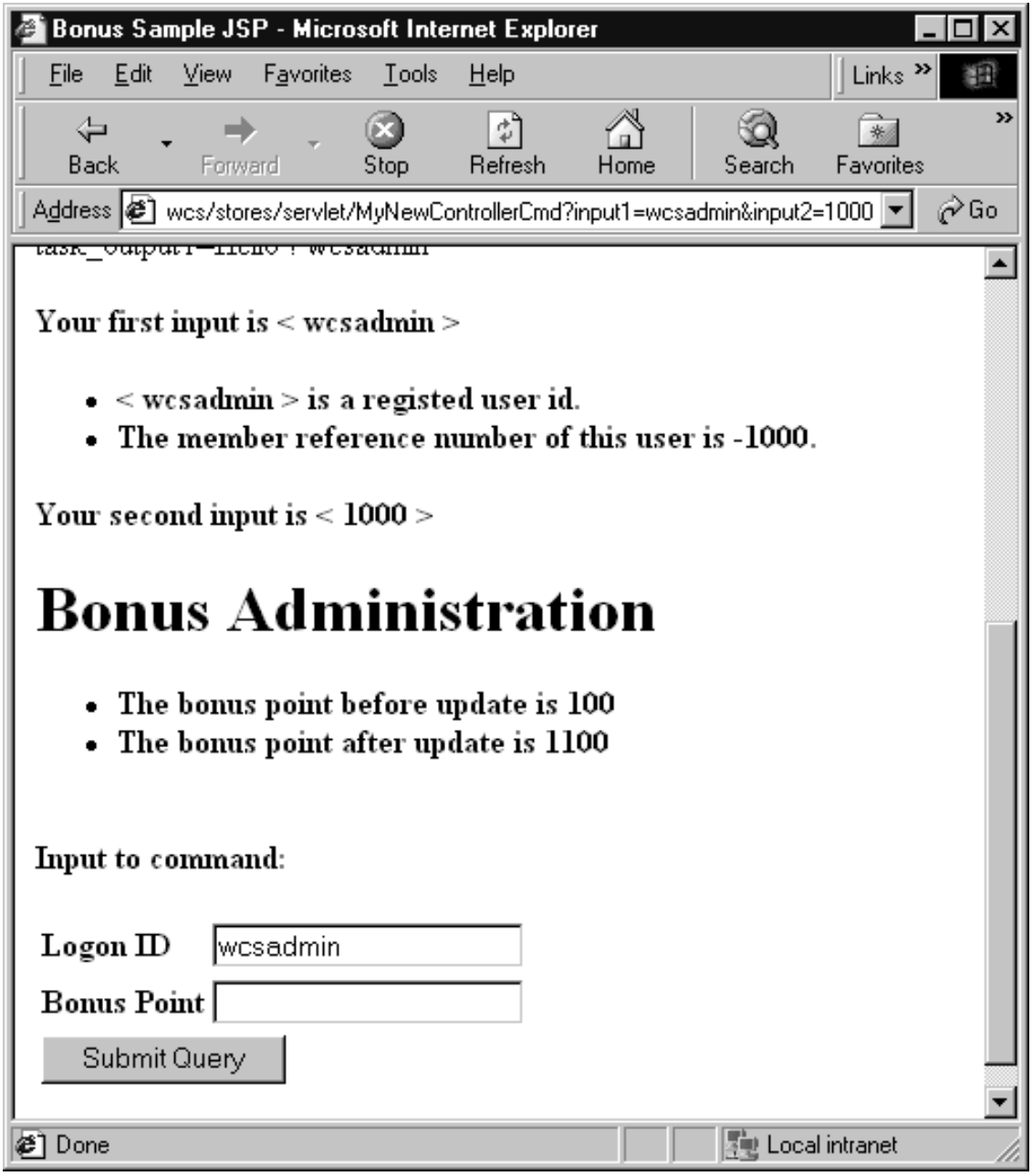


Figura 40.

7. Crie uma versão do código neste estado atual. Denomine a versão como `mySample 1.7 Completed`. Ao fazer a versão do código, certifique-se de

selecionar os dois projetos. Se você necessitar de informações detalhadas sobre como fazer a versão do código, consulte a etapa 12 na página 215.

(Opcional) Utilizando o Debugger no VisualAge for Java

Esta seção mostra como incluir um ponto de interrupção no código e lançar o componente Debugger do VisualAge for Java. Isso foi incluído para introduzir o Debugger. Para obter detalhes sobre este poderoso recurso, consulte a ajuda online do VisualAge for Java.

Esta seção é opcional, pois não introduz nenhum código novo que seja necessário para o tutorial. Em vez disso, você cria um ponto de interrupção, verifica os valores de algumas variáveis nele e em seguida, o remove.

Incluindo o Ponto de Interrupção em Seu Código

Para incluir o ponto de interrupção em seu código, faça o seguinte:

1. Verifique se o uso de pontos de interrupção está ativado em sua área de trabalho, fazendo o seguinte:
   a. No menu **Janela**, selecione **Depurar**. Verifique se **Ativação Global de Pontos de Interrupção** está selecionado.
2. Selecione a guia **Projetos**.
3. Expanda o projeto **_WCSamples**.
4. Expanda o pacote **com.ibm.commerce.sample.commands**.
5. Expanda a classe **MyNewTaskCmdImpl**.
6. Selecione o método **performExecute** para exibir seu código fonte.
7. No painel Origem, coloque o cursor (e clique) no início da seguinte linha de código:

   ```
   setTask_output1( "Hello ! " + getTask_input1() );
   ```

   Deixe o cursor nesta posição.
8. No menu **Editar**, selecione **Ponto de Interrupção**.
9. Na janela Configurando: Ponto de Interrupção #1, selecione **No Thread Selecionado** e clique em **OK**.
10. Certifique-se de que o WebSphere Test Environment esteja em execução (consulte o Apêndice A, “Iniciando e Parando o WebSphere Test Environment” na página 347).
11. Efetue login como o usuário `wcsadmin`, fazendo o seguinte:
    - Digite a seguinte URL em um navegador:

      ```
      http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
          storeId=store_Id&catalogId=catalog_Id&langId=-1
      ```

    - Clique no link **Registro**.
      A página Registro ou Login é exibida.

- No campo **Endereço de e-mail**, digite `wcsadmin`.
- No campo **Senha**, digite a senha para o usuário wcsadmin e em seguida, clique em **Login**.

12. Depois de concluir o processo de login, digite a seguinte URL:

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
    input1=wcsadmin&input2=1000
```

13. A janela do Debugger é aberta quando o ponto de interrupção no código é alcançado.

Verificando os Valores de Variáveis

Esta seção mostra como verificar os valores das variáveis `task_input1` e `task_output1` em vários pontos durante a execução do comando.

Quando o Debugger é aberto por causa do ponto de interrupção no método performExecute de MyNewTaskCmdImpl, vá para para esta janela e faça o seguinte:

1. No painel Variável, expanda **this**.
2. Clique em **task_input1**.
   No painel Valor, o valor dessa variável neste ponto durante a execução é exibido. Ela deve exibir `wcsadmin`.

Para assegurar que o valor da variável `task_output1` seja definido conforme esperado, o Debugger deverá continuar a executar o método performExecute para alcançar o ponto no código onde essa variável será definida. Isso pode ser feito da seguinte forma:

1. Utilize a função Avançar sobre para mover-se pelo código etapa por etapa. Isso pode ser feito em uma das seguintes maneiras:
   - No menu **Selecionado**, selecione **Avançar Sobre**.
   - Clique em F6.
   - Clique no ícone Avançar Sobre.

   Para este exemplo, clique em F6 quatro vezes. Isso executa o código que define a variável task_output1.
2. No painel Variável, clique em task_output1.
   No painel Valor, o valor dessa variável neste ponto durante a execução é exibido. Ele deve exibir `Hello ! wcsadmin`.
3. Você pode concluir a execução do comando clicando no ícone Continuar.

Removendo o Ponto de Interrupção

Para remover o ponto de interrupção de seu código, faça o seguinte:

1. Volte para a janela Workbench.
2. Verifique se pode exibir o código fonte do método `performExecute` da classe `MyNewTaskCmdImpl`.

3. No painel Origem, navegue para a seguinte linha de código:

```
setTask_output1( "Hello ! " + getTask_input1() );
```

   Observe que na margem esquerda do painel há um ponto azul para marcar o ponto de interrupção.
4. Dê um clique duplo em no ponto azul para remover o ponto de interrupção.

O ponto de interrupção foi removido de seu código.

Integrando MyNewControllerCmd com a Loja de Exemplo no WebSphere Test Environment

Nesta seção, você inclui um link na home page para a loja de exemplo que chama MyNewControllerCmd. Para executar essa etapa de integração, proceda da seguinte forma:

1. Em um editor de texto, abra o arquivo `sidebar.jsp`. Este arquivo está localizado no seguinte diretório:
   vaj_drive`:\VAJava\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web\`*store_directory*`\include`
   em que *vaj_drive* é a unidade em que você instalou o VisualAge for Java e *store_directory* é o nome do diretório da loja de exemplo.
2. Inclua outra linha com um link para MyNewControllerCmd na tabela inserindo o seguinte código antes da tag `</table>` final:

```
<tr>
<td>
<a href = "MyNewControllerCmd?input1=wcsadmin&input2=1000">
     MyNewControllerCmd</a>
</td>
</tr>
```

   Salve seu trabalho.
3. Certifique-se de que o WebSphere Test Environment esteja em execução.
4. Teste a integração digitando a seguinte URL em um navegador:

```
http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
     storeId=store_Id&catalogId=catalog_Id&langId=-1
```

   Clique no link **Registrar**.
   A página Registrar ou Login é exibida.
5. No campo **Endereço de e-mail**, digite `wcsadmin`.
6. No campo **Senha**, digite a senha para o usuário wcsadmin e em seguida, clique em **Login**.
7. Depois que o usuário efetuou logon, clique no link **MyNewControllerCmd** no painel de navegação lateral. A página Sample JSP é exibida.

Nota: Se a área de janela de navegação lateral não exibir o novo link, o sidebar.jsp pode estar em cache. Remova-o da cache e recarregue a página. Consulte também Apêndice C, "Dicas para o VisualAge for Java" na página 385 para obter informações sobre a exclusão de arquivos JSP compilados. Pode ser necessário que você reinicie seu mecanismo de servlet após a exclusão dos arquivos JSP compilados.

(Opcional) Implementando Nova Lógica de Negócios em um WebSphere Commerce Server Remoto

Esta seção descreve como implementar sua nova lógica de negócios em uma loja que está sendo executada em um WebSphere Commerce Server remoto. Você deve ter criado uma loja (baseada na loja de exemplo InFashion) no WebSphere Commerce Server remoto antes de iniciar essas etapas de implementação.

O processo de implementação inclui etapas que são executadas na máquina de desenvolvimento, assim como as etapas que são executadas no WebSphere Commerce Server de destino.

Criar o Arquivo JAR para a Lógica do Comando Novo

Você deve criar um arquivo JAR que contenha o comando novo e a lógica do bean de dados. Para criar esse arquivo JAR.jsp, proceda da seguinte forma:

1. Pare o WebSphere Test Environment, conforme descrito em Apêndice A, "Iniciando e Parando o WebSphere Test Environment" na página 347.
2. Com a guia Projetos selecionada, selecione o projeto **_WCSamples**.
3. Com o projeto destacado, clique com o botão direito do mouse e selecione **Exportar**.
   O SmartGuide Exportar é aberto.
4. Selecione **arquivo Jar** e clique em **Avançar.**
5. No campo Arquivo Jar, digite o seguinte:
   unidade:\WebSphere\CommerceServerDev\mytemp\wcssamples_1.jar
   em que *unidade* é a unidade na qual o Commerce Studio está instalado.
6. Selecione os atributos da seguinte forma:

Atributo	Valor
classe	Verificado
java	Não verificado
recurso	Verificado
beans	Verificado
Incluir atributos de depuração no arquivo .class	Verificado

Aceite os valores padrão para outros atributos.

7. Clique em **Concluir**.
8. Se solicitado, confirme a criação do novo diretório.

Como o arquivo JAR criado não contém informações completas sobre nomeação de pacotes, você deve usar outro utilitário de empacotamento (fora do VisualAge for Java) para empacotar novamente o arquivo JAR. Para empacotar novamente este arquivo, proceda da seguinte maneira:

1. Em uma janela de comando, navegue para o seguinte diretório:
   unidade`:\WebSphere\CommerceServerDev\mytemp`
2. Digite `mkdir temp1`.
3. Digite `cd temp1`.
4. Defina o caminho da seguinte forma:
   `set PATH=%PATH%;`*unidade*`:\WebSphere\WebSphereStudio4\bin;`
   em que *unidade* é a unidade na qual o WebSphere Studio está instalado.
5. Digite `jar xvf ../wcssamples_1.jar`
6. Digite `jar cvf ../wcssamples.jar *` (observe que o _1 é removido do nome).

Criando o Arquivo JAR para o Novo Grupo EJB

Será necessário criar um arquivo JAR para seu novo grupo EJB. Para criar esse arquivo, proceda da seguinte forma:

1. Com a guia EJB selecionada, clique com o botão direito do mouse no grupo de EJB **WCSSamplesEntityBeans** e selecione **Exportar > JAR EJB 1.1**.
   O SmartGuide Exportar para um Arquivo JAR EJB 1.1 é aberto.
2. No campo **Arquivo JAR**, digite
   unidade`:\WebSphere\CommerceServerDev\mytemp\sampleEntityBeans_DT.jar`
3. Selecione os atributos da seguinte forma:

Atributo	**Valor**
classe	Verificado
java	Não verificado
recurso	Verificado
Banco de dados de destino	DB2 Se estiver implementando para um banco de dados DB2, selecione **DB2 para NT, V7.1**. Oracle Se estiver implementando para um banco de dados Oracle, selecione **Oracle, V8**.
Incluir atributos de depuração no arquivo .class	Verificado

Aceite os valores padrão para outros atributos.

4. Clique em **Concluir**.

O arquivo JAR é criado.

O arquivo JAR foi denominado com o sufixo "_DT" como lembrete de que será necessário executar este arquivo JAR através da Ferramenta de Implementação de EJB fornecida pelo WebSphere Application Server antes de implementá-la em seu aplicativo WebSphere Commerce.

Criando o Arquivo JAR de Implementação para o Novo Bean Corporativo

Você deve criar um arquivo JAR que contenha o código de implementação para o bean corporativo Bonus. Para criar esse arquivo, proceda da seguinte forma:

1. Com a guia Projeto selecionada, clique com o botão direito do mouse no **_WCSSamplesEntityBeansProject** e selecione **Exportar**.
   O SmartGuide Exportar é aberto.
2. Selecione **arquivo Jar** e clique em **Avançar.**
3. No campo **Arquivo JAR**, digite
   unidade:\WebSphere\CommerceServerDev\mytemp\sampleImpl_1.jar
4. Selecione os atributos da seguinte forma:

Atributo	Valor
classe	Verificado
java	Não verificado
recurso	Verificado
beans	Verificado
Incluir atributos de depuração no arquivo .class	Verificado

   Aceite os valores padrão para outros atributos.
5. Clique em **Concluir**.

O arquivo JAR é criado. Feche o VisualAge for Java.

Como o arquivo JAR criado não contém informações completas sobre nomeação de pacotes, você deve usar outro utilitário de empacotamento (fora do VisualAge for Java) para empacotar novamente o arquivo JAR. Para empacotar novamente este arquivo, proceda da seguinte maneira:

1. Em uma janela de comando, navegue para o seguinte diretório:
   unidade:\WebSphere\CommerceServerDev\mytemp
2. Digite `mkdir temp2`.
3. Digite `cd temp2`.

4. Defina o caminho da seguinte forma:
   `set PATH=%PATH%;`*`unidade`*`:\WebSphere\WebSphereStudio4\bin;`
   em que *unidade* é a unidade na qual o WebSphere Studio está instalado.
5. Digite `jar xvf ../sampleImpl_1.jar`
6. Digite `jar cvf ../sampleImpl.jar *` (observe que o `_1` é removido do nome).

Copiar os Arquivos JSP para o WebSphere Commerce Server de Destino

Você deve copiar o `Sample.jsp` e os arquivos `sidebar.jsp` atualizados para os diretórios corretos da loja para qual o código está sendo implementado.

Antes de copiar os gabaritos JSP atualizados no WebSphere Commerce Server de destino, você pode querer fazer cópias de backup dos gabaritos JSP originais naquela máquina. Isto é, renomeie o arquivo existente para `sidebar.jsp.bak`.

Para copiar estes arquivos, faça o seguinte:

1. Na máquina de desenvolvimento, navegue para o seguinte diretório:
   `vaj_drive``:\VAJava\ Ide\project_resources\IBM WebSphere Test Environment \hosts\default_host\default_app\web\`*`store_directory`*, em que *vaj_drive* é a unidade na qual você instalou o VisualAge for Java e *store_directory* é o nome do diretório da loja.
   Copie `Sample.jsp`.
2. Cole `Sample.jsp` no seguinte diretório no WebSphere Commerce Server de destino:

   ```
   unidade:\WebSphere\AppServer\installedApps\
      WC_Enterprise_App_instance_name.ear\
      wcstores.war\store_directory
   ```

   em que *unidade* é a unidade na qual o WebSphere Commerce está instalado,*store_directory* é o nome do diretório da loja, e *instance_name* é o nome da sua instância do WebSphere Commerce.
3. Na máquina de desenvolvimento, navegue para o seguinte diretório:
   `vaj_drive``:\VAJava\ Ide\project_resources\IBM WebSphere Test Environment`
   `\hosts\default_host\default_app\web\`*`store_directory`*`\include`
   Copie `sidebar.jsp`
4. Cole `sidebar.jsp` no seguinte diretório no WebSphere Commerce Server de destino:

   ```
   unidade:\WebSphere\AppServer\installedApps\
      WC_Enterprise_App_instance_name.ear\
      wcstores.war\store_directory\include
   ```

Copiar os Arquivos JAR para o WebSphere Commerce Server de Destino

É necessário copiar os arquivos JAR da máquina de desenvolvimento para o diretório apropriado no WebSphere Commerce Server de destino.

Além disso, existem duas etapas de processamento posteriores que devem ser executadas no arquivo JAR contendo o novo grupo EJB. Em primeiro lugar, você deve executar a Ferramenta de Implementação de EJB do WebSphere Application Server no arquivo. Em seguida, será necessário executar o comando modifyIsolationLevel no arquivo. Como resultado, nesta etapa de cópia dos arquivos JAR no WebSphere Commerce Server de destino, em especial este arquivo JAR será armazenado em um diretório temporário para aguardar essas etapas posteriores.

Para copiar estes arquivos, faça o seguinte:

1. Na máquina de desenvolvimento, navegue até o seguinte diretório: *unidade*:\WebSphere\CommerceServerDev\mytemp e localize os seguintes arquivos:
   - wcssamples.jar
   - sampleImpl.jar
   - sampleEntityBeans_DT.jar

   em que *unidade* é o nome da unidade em que você instalou o WebSphere Commerce Studio, Business Developer Edition.

   Cada um dos arquivos anteriores devem ser copiados em um diretório específico no WebSphere Commerce Server de destino. Leia as seguintes etapas com cuidado para assegurar que cada arquivo seja armazenado no local correto.
2. Copie o arquivo wcssamples.jar no seguinte diretório no WebSphere Commerce Server de destino:

   unidade:\WebSphere\AppServer\InstalledApps\
   WC_Enterprise_App_*instance_name*.ear\wcstores.war\WEB-INF\lib

   em que *unidade* é a unidade em que você instalou o WebSphere Commerce Business Edition e *instance_name* e o nome de sua instância (por exemplo, demo).
3. Crie um diretório de biblioteca temporário no qual o arquivo JAR será colocado. Isto é, crie o seguinte diretório:

   unidade:\WebSphere\CommerceServer\temp\lib
4. Copie o arquivo sampleImpl.jar no seguinte diretório do WebSphere Commerce Server de destino:

   unidade:\WebSphere\CommerceServer\temp\lib
5. Copie o arquivo sampleEntityBeans_DT.jar no seguinte diretório do WebSphere Commerce Server de destino:

unidade:\WebSphere\CommerceServer\temp

Executando a Ferramenta de Implementação do EJB

Antes de executar seus beans corporativos com êxito em um servidor de teste ou produção, será necessário gerar um código de implementação para os beans corporativos. A Ferramenta de Implementação do Enterprise JavaBeans (também referenciada como Ferramenta de Implementação EJB) fornecida com o WebSphere Application Server, oferece uma interface de linha de comandos que pode ser utilizada para gerar códigos de implementação de bean corporativo.

Para executar esta ferramenta, faça o seguinte:

1. Em um prompt de comandos, navegue para o seguinte diretório:
   unidade:\WebSphere\CommerceServer\temp
2. Inclua temporariamente a ferramenta no caminho do sistema digitando o seguinte comando:
   PATH=*unidade*:\WebSphere\AppServer\deploytool;%PATH%
3. Digite o comando ejbdeploy da seguinte maneira:
   ejbdeploy *EJBGroupJARFile WorkingDir OutputJARFile* -nowarn -keep -35 -cp *ClassPathOfDepJARFiles*

   em que:
   - *EJBGroupJARFile* é o nome completo do arquivo JAR dos beans corporativos para o qual deseja gerar o código implementado. Neste caso, este é *unidade*:\WebSphere\CommerceServer\temp\sampleEntityBeans_DT.jar.
   - *WorkingDir* é o nome do diretório em que são armazenados os arquivos temporários que são obrigatórios para geração de códigos.
   - *OutputJARFile* é o nome completo do arquivo JAR de saída. Neste caso, digite *unidade*:\WebSphere\CommerceServer\temp\sampleEntityBeans.jar.
   - -nowarn é um parâmetro opcional para anular mensagens de aviso e de informações.
   - -keep é um parâmetro opcional para manter o diretório de trabalho depois que o comando ejbdeploy for executado.
   - -35 é um parâmetro obrigatório que utiliza as mesmas regras de mapeamento top-down para beans de entidade CMP utilizados na Ferramenta de Implementação de EJB fornecida com o WebSphere Application Server, Versão 3.5.
   - -cp *ClassPathOfDepJARFiles* é o caminho de classe de quaisquer arquivos JAR dependentes. Neste caso, digite

     "*unidade*:\WebSphere\CommerceServer\temp\lib\sampleImpl.jar;
     unidade:\WebSphere\AppServer\installedApps\
     WC_Enterprise_App_*instanceName*.ear\lib\wcsejbimpl.jar"

Nota: O valor do caminho da classe deve ser colocado entre aspas (“”).

Modificar o Nível de Isolamento de Transação para o Bean Bonus

Nesta etapa, você irá utilizar o comando `modifyIsolationLevel` para modificar o nível de isolamento da transação do bean Bonus. Esta ferramenta também define o nível de isolamento do bean para o nível requerido para seu tipo específico de banco de dados.

Para executar o comando `modifyIsolationLevel`, faça o seguinte:

1. No WebSphere Commerce Server de destino, abra uma janela de comando.
2. Vá para o seguinte diretório:
   unidade`:\WebSphere\CommerceServer\bin`
3. Será necessário emitir o comando `modifyIsolationLevel`, que possui a seguinte sintaxe geral:

   ```
   modifyIsolationLevel -jarFile jar_file_name.jar
      -logFile log_file_name -dbType db_type
   ```

   - *jar_file_name.jar* é o nome do arquivo JAR que contém o código personalizado
   - *log_file_name* é o nome de arquivo completo no qual as informações devem ser registradas
   - *db_type* é o tipo de banco de dados que está sendo utilizado. Digite DB2 ou ORACLE

   O exemplo a seguir é um exemplo do comando `modifyIsolationLevel` com todos os valores especificados:

   ```
   modifyIsolationLevel -jarFile
      D:\WebSphere\CommerceServer\temp\sampleEntityBeans.jar
      -logFile D:\WebSphere\CommerceServer\instances\demo\logs\output.log
      -dbType DB2
   ```

O comando terá sido executado com êxito se nenhuma exceção for exibida na janela de comando. Depois da conclusão, observe que a marca de hora em seu arquivo JAR foi alterada.

Nota: Os nomes dos parâmetros fazem distinção entre maiúsculas e minúsculas. Ou seja, jar**F**ile não é o mesmo que jar**f**ile. Verifique se os nomes dos parâmetros estão digitados corretamente.

Atualizando o Banco de Dados de Destino

Já que você está implementando a nova lógica de negócios para um WebSphere Commerce Server de destino que utiliza um banco de dados diferente do utilizado pelo WebSphere Test Environment, você deve atualizar o banco de dados de destino para refletir as alterações que foram feitas para o registro de comando e para criar a tabela BONUS.

DB2 Se estiver utilizando um banco de dados DB2, faça o seguinte para atualizar o banco de dados de destino:

1. Abra o Centro de Comandos do DB2 (**Iniciar > Programas >IBM DB2 > Centro de Comandos**).
2. No menu **Ferramentas**, selecione **Definições de Ferramentas**.
3. Selecione a caixa de opções **Utilizar caractere de finalização de instrução** e certifique-se de que o caractere especificado é um ponto e vírgula (**;**)
4. Com a guia Script selecionada, crie a entrada requerida na tabela URLREG, digitando as seguintes informações na janela de script

   ```
   connect to your_database_name;
   insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS,
       DESCRIPTION, AUTHENTICATED) values ('MyNewControllerCmd',0,
       'com.ibm.commerce.sample.commands.MyNewControllerCmd',0,
       'This is a new controller command for test/education purposes.',
       null)
   ```

   em que *your_database_name* é o nome do banco de dados, e clique no ícone Executar.
   Esse comando é utilizado por todos os comerciantes (indicados pelo valor 0 para STOREENT_ID).
5. Crie uma entrada na tabela VIEWREG, digitando o seguinte na janela de script:

   ```
   insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID,
      INTERFACENAME, CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE)
      values ('SampleViewTask',-1, 0,
      'com.ibm.commerce.command.ForwardViewCommand',
      'com.ibm.commerce.command.HttpForwardViewCommandImpl',
      'docname=Sample.jsp','This is a sample view for the Bonus Point
      exercise', 0, null)
   ```

   e clique no ícone Executar.
6. Crie a tabela BONUS, digitando o seguinte na janela de script:

   ```
   CREATE TABLE Bonus (MEMBERID BIGINT NOT NULL,
       BONUSPOINT INTEGER NOT NULL,
       constraint p_memberid primary key (MEMBERID),
       constraint f_memberid foreign key (MEMBERID)
       references users (users_id) on delete cascade)
   ```

   Clique no ícone Executar.

As etapas acima registram MyNewControllerCmd e SampleViewTask no registro de comandos, assim como, criam a tabela BONUS.

Oracle Se você estiver utilizando um banco de dados Oracle, faça o seguinte para atualizar o banco de dados de destino:

1. Abra a janela de comando Oracle SQL Plus (**Iniciar > Programas > Oracle > Desenvolvimento de Aplicativos > SQL Plus**).
2. No campo **Nome do Usuário**, digite seu nome de usuário do Oracle.
3. No campo **Senha**, digite sua senha do Oracle.
4. No campo **Cadeia do Host**, digite sua cadeia de conexão.
5. Crie a entrada obrigatória na tabela URLREG, digitando as seguintes informações na janela SQL Plus:

   ```
   insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS,
       DESCRIPTION, AUTHENTICATED) values ('MyNewControllerCmd',0,
       'com.ibm.commerce.sample.commands.MyNewControllerCmd',0,
       'This is a new controller command for test/education purposes.',
       null);
   ```

   e pressione Enter para executar a instrução SQL.
6. Crie uma entrada na tabela VIEWREG, digitando o seguinte na janela SQL Plus:

   ```
   insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID,
      INTERFACENAME, CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE)
      values ('SampleViewTask',-1, 0,
      'com.ibm.commerce.command.ForwardViewCommand',
      'com.ibm.commerce.command.HttpForwardViewCommandImpl',
      'docname=Sample.jsp','This is a sample view for the Bonus Point
      exercise', 0, null);
   ```

   e pressione Enter para executar a instrução SQL.
7. Crie a tabela BONUS, digitando o seguinte na janela SQL Plus:

   ```
   CREATE TABLE Bonus (MEMBERID NUMBER NOT NULL,
       BONUSPOINT INTEGER NOT NULL,
       constraint p_memberid primary key (MEMBERID),
       constraint f_memberid foreign key (MEMBERID)
       references users (users_id) on delete cascade);
   ```

   e pressione Enter para executar a instrução SQL.
8. Digite o seguinte para consolidar as alterações no banco de dados:

   ```
   commit;
   ```

   e pressione Enter para executar a instrução SQL.

Carregando as Políticas de Controle de Acesso para os Novos Recursos

No tutorial, você criou um novo bean corporativo (o bean Bonus) que é um recurso que pode ser protegido. Como tal, há uma política de controle de acesso relacionada a este recurso. Você também criou um novo comando de controlador que pode ser executado por todos os usuários. Ao trabalhar na máquina de desenvolvimento, você carregou as informações da política de

controle de acesso nessa máquina. Agora, será necessário carregar as mesmas informações da política de controle de acesso para o WebSphere Commerce Server de destino.

Para configurar as políticas de controle de acesso, faça o seguinte:

1. Insira o seguinte CD na unidade de CD:
   - Disco 2 do CD Business WebSphere Commerce Business Edition, V5.4
   - Disco 2 do CD Professional WebSphere Commerce Professional Edition, V5.4
2. Vá para o seguinte diretório:
   CD_drive:\repository\samples\programguide\
3. Localize os seguintes arquivos nesse diretório:
   - SampleCmdACPolicy.xml
     Esse arquivo XML contém a política de controle de acesso que é utilizada pelo novo comando de controlador.
   - SampleACPolicy.xml
     Esse arquivo XML contém a política de controle de acesso que é utilizada quando você cria um novo bean corporativo.
   - SampleACPolicy_*locale*.xml
     em que *locale* é o identificador do idioma. Este arquivo XML contém a descrição da política de controle de acesso.
4. Copie os três arquivos anteriores no seguinte diretório:
   unidade:\WebSphere\CommerceServer\xml\policies\xml
5. Para carregar o arquivo SampleCmdACPolicy.xml, utilize um prompt de comandos para ir para o seguinte diretório:
   unidade:\WebSphere\CommerceServer\bin
   Será necessário emitir o comando acpload, que possui a seguinte forma:
   ```
   acpload db_name db_user db_password inputXMLFile
   ```
   em que
   - *db_name* é o nome do banco de dados
   - *db_user* é o nome do usuário do banco de dados
   - *db_password* é a senha do banco de dados
   - *inputXMLFile* é o nome do arquivo XML que contém a política.

   Por exemplo, você pode emitir o seguinte comando:
   ```
   acpload mall user password SampleCmdACPolicy.xml
   ```
6. Carregue o arquivo SampleACPolicy.xml, emitindo o comando acpload que especifica o arquivo SampleACPolicy.xml como o arquivo de entrada. Por exemplo, você pode emitir o seguinte comando:
   ```
   acpload mall user password SampleACPolicy.xml
   ```

7. Para carregar a descrição da política, você deve emitir o comando `acpnlsload`, que possui o seguinte formato:

   ```
   acpnlsload db_name db_user db_password inputXMLFile
   ```

   Por exemplo, você pode emitir o seguinte comando:

   ```
   acpnlsload mall user password SampleACPolicy_en_US.xml
   ```

Observe que, normalmente, seria necessário uma atualização de registro para que a política fosse efetivada. Nesse caso, não é necessário que você execute esta etapa, uma vez que encerrará e reiniciará o aplicativo WebSphere Commerce Server no WebSphere Application Server como parte das etapas de implementação do bean corporativo. Caso contrário, você poderia utilizar o Administration Console no WebSphere Commerce para atualizar o registro. Para obter mais informações sobre o Administration Console, consulte a ajuda online do WebSphere Commerce.

400 Se você estiver implementando para uma instância do WebSphere Commerce em execução em uma máquina do iSeries, utilize comandos diferentes para carregar as informações de política de controle de acesso. Utilize o comando LODWCSAC em vez do comando acpload e o comando LODWCSACD em vez do comando acpnlsload.

A sintaxe do comando LODWCSAC é:

```
LODWCSAC DATABASE(dbName) SCHEMA(schemaName)
PASSWD(instancePassword) INSTROOT('instanceRoot')
INFILE('inputFile'). Em que
```

- *dbName* é o nome do banco de dados relacional conforme definido no comando WRKRDBDIRE.
- *schemaName* é o nome do esquema do banco de dados da instância (tem o mesmo nome que a instância).
- *instancePassword* é a senha da instância.
- *instanceRoot* é a raiz da instância. Um exemplo de raiz da instância é

  /QIBM/UserData/WebCommerce/instances/*instanceName*
- *inputFile* é o nome completo do arquivo XML de entrada que tem as políticas de acesso

A sintaxe do comando LODWCSACD é:

```
LODWCSACD DATABASE(dbName) SCHEMA(schemaName)
PASSWD(instancePassword) INSTROOT('instanceRoot')
INFILE('inputFile'). Em que
```

Você pode armazenar os arquivos XML para suas políticas de controle de acesso no seguinte diretório:

/QIBM/UserData/WebCommerce/instances/*instanceName*

Além disso, nos arquivos XML para suas políticas de controle de acesso, você deve utilizar o caminho completo para o DTD de controle de acesso. Os DTDs para políticas de controle de acesso estão armazenados no diretório /QIBM/ProdData/WebCommerce/xml/policies/dtd.

Como um exemplo, se você implementar as políticas de controle de acesso para os tutoriais em uma instância do Websphere Commerce em execução em uma máquina iSeries, deverá modificar a especificação do DTD nos arquivos XML para as políticas de controle de acesso dos tutoriais de:

```
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">
```

para

```
<!DOCTYPE Policies SYSTEM "/QIBM/ProdData/WebCommerce/
xml/policies/dtd/accesscontrolpolicies.dtd">
```

Para obter mais informações sobre o modelo de controle de acesso do WebSphere Commerce, consulte o manual *WebSphere Commerce Access - Manual de Controle*.

Exportando o Aplicativo Corporativo Atual do WebSphere Application Server

Nesta etapa, você exportará o aplicativo corporativo atual do WebSphere Application Server para que você possa abri-lo na Ferramenta de Montagem de Aplicativos.

Para exportar o aplicativo corporativo atual, faça o seguinte:

1. Crie um diretório para o qual o aplicativo corporativo atual será exportado. Observe que esse diretório não pode ser chamado do diretório "temp" para que não exista perigo do arquivo ser excluído durante a manutenção de rotina do sistema, até que você esteja satisfeito com a forma que o código personalizado se comporta quando ele for implementado. Para criar este diretório, faça o seguinte no prompt de comandos:
   a. Navegue para o seguinte diretório:
      unidade:\WebSphere\CommerceServer\
   b. Digite o seguinte comando:
      ```
      mkdir working
      ```
      Isto irá criar o diretório *unidade*:\WebSphere\CommerceServer\working.
2. Abra o WebSphere Application Server Administration Console.
3. Expanda o **Domínio Administrativo do WebSphere**.
4. Expanda **Aplicativos Corporativos**.
5. Clique com o botão direito do mouse em seu aplicativo WebSphere Commerce. Por exemplo, clique com o botão direito do mouse no aplicativo **demo** e selecione **Exportar Aplicativo**.
6. No campo **Diretório de exportação**, digite *unidade*:\WebSphere\CommerceServer\working.
   Isto irá exportar todo o aplicativo, incluindo todos os recursos, para o arquivo WC_Enterprise_App_*instanceName*.ear (em que *instanceName* é o nome da sua instância do WebSphere Commerce).
7. Clique em **OK**. A exportação do aplicativo pode levar vários minutos.

Exportando Informações sobre Configuração XML para o Aplicativo Corporativo

Também será necessário exportar as informações sobre configuração XML para o Aplicativo Corporativo. Para exportar estas informações, você irá utilizar o utilitário de linha de comandos do XMLConfig fornecido pelo WebSphere Application Server.

Para exportar estas informações de configuração, faça o seguinte:

1. Copie o arquivo `was.export.app.xml` a partir do seguinte diretório:

   unidade`:\WebSphere\CommerceServer\xml\config`

   para o seguinte diretório:

   unidade`:\WebSphere\CommerceServer\working`

2. Abra o arquivo `was.export.app.xml` em um editor de texto. Neste arquivo, substitua todas as ocorrências de `$Enterprise_Application_Name$` com

   `WebSphere Commerce Enterprise Application -` *instanceName*

   em que *instanceName* é o nome da sua instância do WebSphere Commerce (por exemplo, demo). Salve esse arquivo.

   Nota: O valor que está sendo inserido deve corresponder às informações de sua instância exibidas no WebSphere Advanced Administration Console.

3. Em um prompt de comandos, navegue para o seguinte diretório:

   unidade`:\WebSphere\CommerceServer\working`

4. Chame a ferramenta XMLConfig para executar uma exportação parcial digitando o seguinte comando:

   ```
   xmlConfig -export OutputFile.xml -partial was.export.app.xml
      -adminNodeName wasHostName
   ```

   em que *wasHostName* é o nome do nó no WebSphere Application Server que contém o aplicativo corporativo atual. Além disso, `OutputFile.xml` é o nome do arquivo criado como resultado da execução desse comando e `was.export.app.xml` é o arquivo modificado na etapa 2.

Depois de exportar as informações sobre os beans corporativos que estão no aplicativo corporativo atual, inclua uma nova sub-rotina no arquivo XML que descreva o bean Bonus.

Para incluir a nova sub-rotina que descreva o bean Bonus, proceda da seguinte forma:

1. Navegue para o seguinte diretório:

   unidade`:\WebSphere\CommerceServer\working`

2. Abra o arquivo `OutputFile.xml` em um editor de texto.
3. Localize a tag `<ear-file-name>` e substitua o valor pelo seguinte:

   unidade`:\WebSphere\CommerceServer\working\`
   `WC_Enterprise_App_`*instanceName*`.ear`

4. Também será necessário incluir uma nova sub-rotina para o bean Bonus, conforme exibido no seguinte exemplo:

```
<ejb-module name="WCSSamplesEntityBeans">
   <jar-file>sampleEntityBeans.jar</jar-file>
   <module-install-info>
      <application-server-full-name>/NodeHome:$hostName$/EJBServerHome:
         WebSphere Commerce Server - demo/</application-server-full-name>
   </module-install-info>
   <ejb-module-binding>
      <data-source>
         <jndi-name>jdbc/WebSphere Commerce DB2 DataSource demo</jndi-name>
         <default-user>user</default-user>
         <default-password>password</default-password>
      </data-source>
      <enterprise-bean-binding name="Bonus_Binding">
         <jndi-name>democom/ibm/commerce/sample/objects/Bonus</jndi-name>
      </enterprise-bean-binding>
   </ejb-module-binding>
</ejb-module>
```

em que

- *user* é o nome do usuário do banco de dados.
- *password* é a senha do usuário do banco de dados.

Notas:

a. As quebras de linha noexemplo anterior destinam-se somente para fins de exibição.

b. Assegure-se de que o valor **$hostName$** corresponda ao nome atual do servidor do nó de administração. Além disso, assegure-se de que não haja nenhum caractere de retorno de carro nesta linha.

c. A especificação `<application-server-full-name>` não pode ocupar mais de uma linha.

d. Se estiver utilizando um banco de dados Oracle, deverá modificar as informações de origem de dados. Altere a seguinte linha retirada do fragmento de código anterior:

```
<jndi-name>jdbc/WebSphere Commerce DB2 DataSource demo</jndi-name>
```

para o seguinte:

```
<jndi-name>jdbc/WebSphere Commerce Oracle DataSource demo</jndi-name>
```

e. Quando você estiver desenvolvendo seus próprios aplicativos (por exemplo, fora deste tutorial, certifique-se de que o nome JNDI dos beans corporativos especificado no arquivo XML corresponda ao nome JNDI utilizado no VisualAge for Java, mas com o nome de instância do WebSphere Commerce incluído).

5. Salve o arquivo `OutputFile.xml`.

Montando o Novo Grupo EJB no Aplicativo Corporativo

Nesta etapa, você irá abrir seu aplicativo corporativo na ferramenta de montagem de aplicativo. Assim que ele estiver aberto dentro da ferramenta, você poderá fazer o seguinte para incluir o novo bean Bonus no aplicativo corporativo:

1. Importe o novo bean Bonus. O arquivo JAR para o novo grupo EJB será armazenado dentro da seção do Módulo EJB do aplicativo corporativo.
2. Defina o caminho de classe para o bean Bonus para incluir o arquivo JAR de implementação.
3. Inclua o arquivo JAR de implementação no aplicativo. Este arquivo JAR será armazenado dentro da seção Arquivos do aplicativo corporativo.
4. Defina a segurança do WebSphere Application Server para os métodos contidos no bean Bonus.

Para montar o novo grupo EJB no aplicativo corporativo, faça o seguinte:

1. Faça um backup do aplicativo corporativo atual, da seguinte maneira:
   a. Em um prompt de comandos, navegue para o seguinte diretório:
      unidade:\WebSphere\CommerceServer\working
   b. Digite o seguinte comando:
      ```
      copy WC_Enterprise_App_instanceName.ear
         WC_Enterprise_App_instanceName.ear.bak
      ```
2. Abra o administrative console do WebSphere Application Server.
3. No menu **Arquivo**, selecione **Ferramentas > Ferramenta de Montagem de Aplicativos**.
4. Se uma janela Bem-vindo for aberta, selecione **Cancelar** para fechá-la.
5. Abra o aplicativo corporativo com o qual irá trabalhar da seguinte maneira:
   a. A partir do menu **Arquivo**, selecione **Abrir**.
   b. No campo **Nome do arquivo**, digite:
      ```
      unidade:\WebSphere\CommerceServer\working\
         WC_Enterprise_App_instanceName.ear
      ```
      e clique em **Abrir**. Aguarde até que o aplicativo seja aberto antes de prosseguir com as próximas etapas. Isto pode levar alguns minutos.
6. Clique com o botão direito do mouse nos **Módulos EJB** e selecione **Importar**.
7. No campo **Nome do arquivo**, digite
   ```
   unidade:\WebSphere\CommerceServer\temp\sampleEntityBeans.jar
   ```
   e clique em **Abrir**. Na janela Confirmar Valores, clique em **OK**.

8. Assim que o arquivo `sampleEntityBeans.jar` tiver sido importado, role para o grupo de EJB **WCSSamplesEntityBeans** e selecione-o. Informações sobre este grupo são mostradas no painel à direita.
9. No campo do classpath para o novo bean corporativo, digite o nome de quaisquer arquivo JAR dependentes. Neste caso digite,
   `lib/sampleImpl.jar lib\wcsejbimpl.jar`
10. Clique em **Aplicar**.
11. Inclua o arquivo `sampleImpl.jar` no aplicativo, executando as seguintes ações:
    a. Clique com o botão direito do mouse no nó **Arquivos** para o aplicativo corporativo e selecione **Incluir Arquivos**. O nó **Arquivos** para o aplicativo corporativo está localizado próximo à parte inferior da árvore hierárquica. Observe que existem outros nós de Arquivos para componentes dentro do aplicativo corporativo, mas você deverá selecionar o nó de Arquivos para todo o aplicativo.)
    b. Na janela Incluir Arquivos, clique em **Procurar**.
    c. Navegue para o *unidade*`:\WebSphere\CommerceServer\temp`.
    d. Com este diretório em destaque, clique em **Selecionar**.
    e. Retorne à janela Incluir Arquivos. Observe que o conteúdo do diretório *unidade*`:\WebSphere\CommerceServer\temp` está exibido. Destaque o diretório `lib`.
       O conteúdo do diretório `lib` será exibido no painel à direita.
    f. No painel à direita, selecione o arquivo `sampleImpl.jar` e clique em **Incluir**. O arquivo então será exibido no painel Arquivos Selecionados.
    g. Clique em **OK**.
12. Configure a segurança para o bean Bonus, executando as seguintes ações:
    a. Com o nó Módulos de EJB expandido, localize e expanda o nó **WCSSamplesEntityBeans**.
    b. Expanda **Beans de Entidade**.
    c. Expanda **Bonus**
    d. Clique em **Extensões do Método** e em seguida, no painel à direita, faça o seguinte:
       1) Clique na guia **Avançado**.
       2) Assegure-se de que **Identidade de segurança** esteja selecionado.
       3) Para cada método, assegure-se de que **Utilizar identidade do servidor EJB** esteja selecionado.
       4) Clique em **Aplicar** (se tiver feito alguma modificação).
    e. No painel de navegação à esquerda, clique com o botão direito do mouse em **Funções de Segurança** no grupo de EJB `WCSamplesEntityBeans` e selecione **Novo**; depois faça o seguinte:

1) No campo **Nome**, digite `WCSecurityRole` e clique em **Aplicar**. Observe, se esta função já existir, não será necessário executar esta etapa.

f. No painel de navegação à esquerda, clique com o botão direito do mouse em **Permissões do Método** no grupo de EJB `WCSamplesEntityBeans` e selecione **Novo**; depois faça o seguinte:

1) No campo **Nome de Permissão do Método**, digite `WCMethodPermission`
2) Na área de seleção **Métodos**, clique em **Incluir**. A janela Incluir Métodos é aberta.
3) Expanda **sampleEntityBeans.jar**, **Bonus** e depois expanda cada uma das listas de métodos **Inicial** e **Remota**.
4) Mantenha pressionada a tecla Shift e selecione todos os métodos iniciais e clique em **OK**.
5) Repita o processo de seleção do método para incluir também os métodos remotos (se houver algum método remoto).
6) Na área de seleção Funções, clique em **Incluir**, selecione a `WCSecurityRole` e clique em **OK**.
7) Clique em **Aplicar** para cada atualização.

13. A partir do menu **Arquivo**, selecione **Salvar**.
14. Feche o Application Assembler Tool.

Depois de concluir desta etapa, você terá criado um novo aplicativo corporativo que contém toda a lógica anterior, bem como sua nova lógica de negócios. Isto tudo está contido no arquivo recém-modificado `WC_Enterprise_App_`*`instanceName`*`.ear`.

Importando o Novo Aplicativo Corporativo no WebSphere Application Server

Os itens a seguir são etapas de alto nível envolvidas na importação de um novo aplicativo corporativo no WebSphere Application Server:

1. Parando o aplicativo corporativo que está atualmente em execução no WebSphere Application Server e em seguida, removendo-o. Essas etapas serão realizadas no Console do Administrador do WebSphere Application Server.
2. Importando o novo aplicativo, utilizando o utilitário de linha de comandos XMLConfig.
3. Atualizando o Console do Administrador do WebSphere Application Server e em seguida, inicializando o novo aplicativo corporativo.

Cada uma das etapas é descrita com maiores detalhes nas seções a seguir.

Parando e Removendo o Aplicativo Corporativo Atual

Para parar e remover seu aplicativo corporativo atual do WebSphere Application Server, faça o seguinte:

1. Abra o Administration Console do WebSphere Application Server.
2. Expanda **Domínio Administrativo do WebSphere**.
3. Expanda os **Nós**.
4. Expanda *nodeName* (em que *nodeName* é o nome do nó).
5. Expanda **Servidores de Aplicativos**.
6. Clique com o botão direito do mouse em seu aplicativo WebSphere Commerce. Por exemplo, clique com o botão direito do mouse no **WebSphere Commerce Server -** *instanceName* e selecione **Parar**.
7. Expanda **Aplicativos Corporativos**.
8. Clique com o botão direito do mouse em seu aplicativo WebSphere Commerce. Por exemplo, clique com o botão direito do mouse no aplicativo **WebSphere Commerce Enterprise Application - demo** e selecione **Parar**.
9. Clique com o botão direito do mouse em seu aplicativo WebSphere Commerce. Por exemplo, clique com o botão direito do mouse no aplicativo **WebSphere Commerce Enterprise Application - demo** e selecione **Remover**.
10. Quando solicitado se o aplicativo deve ser exportado, selecione **Não**.

Importando o Novo Aplicativo Corporativo Utilizando XMLConfig

Para importar novos aplicativos corporativos utilizando o utilitário de linha de comandos XMLConfig, proceda da seguinte forma:

1. Navegue para o seguinte diretório:

   ```
   unidade:\WebSphere\CommerceServer\working
   ```

2. No prompt de comandos, digite o seguinte comando para importar o aplicativo corporativo no WebSphere Application Server:

   ```
   xmlConfig -import OutputFile.xml -adminNodeName was_hostname
   ```

   em que *was_hostname* é o nome do nó do WebSphere Application Server que contém o aplicativo atual.

Nota: 400 Se estivesse implementando em uma instância do WebSphere Commerce em execução no iSeries, seria necessário executar uma etapa adicional para modificar as permissões de diretório depois de importar o aplicativo. Consulte “Importando um Aplicativo Corporativo” na página 382 para obter detalhes sobre como modificar essas permissões.

Iniciando o Novo Aplicativo Corporativo

Após ter importado um novo aplicativo corporativo utilizando o utilitário de linha de comandos XMLConfig, você poderá utilizar o Console do

Administrador do WebSphere Application Server para executar uma atualização e em seguida, iniciar o novo aplicativo.

Para atualizar o console e iniciar o novo aplicativo, faça o seguinte:

1. Abra o Administration Console do WebSphere Application Server.
2. Expanda **Domínio Administrativo do WebSphere**
3. Destaque **Nós**.
4. Clique no ícone **Atualizar subárvore selecionada**.
5. Inicie o aplicativo WebSphere Commerce da seguinte forma:
   - Expanda **Servidores de Aplicativos**.
   - Clique com o botão direito do mouse em seu aplicativo WebSphere Commerce. Por exemplo, clique com o botão direito do mouse em **WebSphere Commerce Server -** *instanceName* e selecione **Iniciar**.

Testar MyNewControllerCmd

A próxima etapa é testar a nova lógica em sua loja que está sendo executada no ambiente do WebSphere Application Server. Para testar MyNewControllerCmd, faça o seguinte:

1. Teste a integração digitando a seguinte URL em um navegador:

   ```
   http://hostname/webapp/wcs/stores/servlet/StoreCatalogDisplay?
        storeId=store_Id&catalogId=catalog_Id&langId=-1
   ```

   em que *store_Id* é o identificador de sua loja e *catalog_Id* é o identificador do catálogo de sua loja.
2. Clique no link **Registro**.
   A página Registro ou Login é exibida.
3. No campo **Endereço de e-mail**, digite `wcsadmin`.
4. No campo **Senha**, digite a senha do ID de wcsadmin utilizado por este site e clique em **Login**.
5. Depois que o usuário efetuou logon, clique no link **MyNewControllerCmd** no painel de navegação lateral. A página Sample JSP é exibida.

Se o arquivo `sidebar.jsp` atualizado não aparecer, proceda da seguinte maneira para limpar sua cache:

1. Exclua os arquivos armazenados em cache no seguinte diretório:
   unidade`:\WebSphere\CommerceServer\instances\`*instanceName*`\cache`
2. Exclua quaisquer arquivos armazenados em cache nos seguintes diretórios:

   ```
   unidade:\WebSphere\AppServer\temp\hostName\
   WebSphere_Commerce_Server_instanceName\
   WebSphere_Commerce_Enterprise_Application_-_instanceName\
   wcstores.war\storeName
   ```

```
unidade:\WebSphere\AppServer\temp\hostName\
WebSphere_Commerce_Server_instanceName\
WebSphere_Commerce_Enterprise_Application_-_instanceName\
wcstores.war
```

3. Limpe a cache do seu navegador.

Se a compilação da página JSP levar muito tempo, ela poderá não ser exibida. Nesse caso, recarregue a página.

Capítulo 10. Modificando e Estendendo Lógica de Negócios Existente

Os tutoriais a seguir mostram como estender ou modificar lógica de negócios do WebSphere Commerce existente.

Estendendo um Comando de Controlador Existente

Nesta seção, você estende o comando de controlador OrderProcess existente, de forma que o total de pontos de bônus acumulado na compra é exibido na página de confirmação do pedido.

Nota: A meta desse tutorial é mostrar o processo de modificação de um comando de controlador existente. Ele não foi desenvolvido para mostrar a melhor maneira de modificar a etapa do processo de pedido no fluxo de compras. Na verdade, o WebSphere Commerce fornece o comando de tarefa `ExtOrderProcess` que pode ser utilizado para modificar a etapa do processo de pedido no fluxo de compras.

Antes de iniciar este tutorial

É necessário já ter concluído as etapas do Capítulo 9, "Tutorial: Criando Nova Lógica de Negócios" na página 207.

A lista a seguir resume as etapas envolvidas na extensão do comando OrderProcess:

1. Criar um novo pacote no qual o código personalizado é armazenado. Lembrar que todo código personalizado (para comandos e beans de dados) precisa ser armazenado em projetos e pacotes separados do código do WebSphere Commerce.
2. Criar uma nova classe `OrderProcessCmdBonusImpl` que estende o comando `OrderProcessCmdImpl` existente.
3. Incluir campos e métodos na classe `OrderProcessCmdBonusImpl`.
4. Modificar o registro de comandos para utilizar a classe `OrderProcessCmdBonusImpl`.
5. Modificar o modelo confirmation.jsp para exibir a nova lógica de negócios.
6. Testar a nova lógica de negócios no WebSphere Test Environment.
7. (Opcional) Implementar a nova lógica de negócios em uma loja em um WebSphere Commerce Server remoto.

Criando o Novo Pacote para OrderProcessCmdBonusImpl

Para criar o novo pacote no qual o comando OrderProcessCmdBonusImpl é armazenado, faça o seguinte:

1. Na janela Workbench do VisualAge for Java, assegure-se de que você tenha a guia Projetos selecionada.
2. Clique com o botão direito do mouse no projeto **_WCSamples** e selecione **Incluir > Pacote**.
   O SmartGuide Incluir Pacote abrirá.
3. Assegure-se de que o botão de opção **Criar um novo pacote denominado** esteja ativado e digite com.ibm.commerce.sample.order.
4. Clique em **Concluir**.

Criando a Classe OrderProcessCmdBonusImpl

Para criar a nova classe OrderProcessCmdBonusImpl, proceda da seguinte maneira:

1. Clique com o botão direito do mouse no pacote com.ibm.commerce.sample.order e selecione **Incluir > Classe**.
   O SmartGuide Criar Classe abrirá.
2. Assegure-se de que o botão de opção **Criar uma nova classe** esteja selecionado.
3. No campo **Nome da Classe**, digite OrderProcessCmdBonusImpl.
4. Para especificar a superclasse, clique em **Procurar**, em seguida, no campo **Padrão**, digite com.ibm.commerce.order.commands.OrderProcessCmdImpl e clique em **OK**.
5. Clique em **Avançar**.
6. Para especificar os pacotes que deverão ser importados, clique em **Incluir Pacote**. No campo **Padrão**, digite os seguintes pacotes:
   - com.ibm.commerce.datatype e clique em **Incluir**
   - com.ibm.commerce.exception e clique em **Incluir**
   - com.ibm.commerce.order.commands e clique em **Incluir**
   - com.ibm.commerce.order.objects e clique em **Incluir**
   - com.ibm.commerce.ras e clique em **Incluir**
   - com.ibm.commerce.server e clique em **Incluir**
   - com.ibm.commerce.sample.objects e clique em **Incluir**
   - javax.ejb e clique em **Incluir**
   - java.io e clique em **Incluir**
   - java.math, clique em **Incluir** e em seguida, em **Fechar**.
7. Para especificar as interfaces que a classe deverá implementar, clique em **Incluir**. No campo Padrão, digite a seguinte interface:
   - OrderProcessCmd e clique em **Incluir**, em seguida, **Fechar**.

8. Clique em **Concluir**.

Incluindo Campos e Métodos em `OrderProcessCmdBonusImpl`

É necessário incluir dois campos e um método performExecute na classe.

Para incluir o campo `theOrder` na classe `OrderProcessCmdBonusImpl`, faça o seguinte:

1. Clique com o botão direito do mouse na classe `OrderProcessCmdBonusImpl` e selecione **Incluir > Campo**.
   O SmartGuide Criar Campo é aberto.
2. Inclua um campo na classe, utilizando os seguintes atributos. Para obter etapas detalhadas sobre como criar um novo campo, consulte "Criação de novos campos" na página 220

Nome do Atributo	**Valor**
Nome do Campo	`theOrder`
Tipo de Campo	`OrderAccessBean`
Valor Inicial	Deixar em branco.
Modificadores de Acesso	`private`
Outros Modificadores	Deixar todos desmarcados.
Acessar com os métodos getter e setter	`checked`
Getter	`public`
Setter	`public`

   e clique em **Concluir**.

Para incluir o campo `bonusPercentAmount` na classe `OrderProcessCmdBonusImpl`, faça o seguinte:

1. Clique com o botão direito do mouse na classe `OrderProcessCmdBonusImpl` e selecione **Incluir > Campo** novamente.
2. Inclua um campo na classe, utilizando os seguintes atributos. Para obter etapas detalhadas sobre como criar um novo campo, consulte "Criação de novos campos" na página 220

Nome do Atributo	**Valor**
Nome do Campo	`bonusPercentAmount`
Tipo de Campo	`double`
Valor Inicial	`1000`
Modificadores de Acesso	`private`
Outros Modificadores	Deixar todos desmarcados.
Acessar com os métodos getter e setter	`checked`

Nome do Atributo	Valor
Getter	public
Setter	public

e clique em **Concluir**.

Para incluir o método performExecute na classe OrderProcessCmdBonusImpl, faça o seguinte:

1. Clique com o botão direito do mouse na classe **OrderProcessCmdBonusImpl** e selecione **Incluir > Método**.
   O SmartGuide Criar Método é aberto.
2. Assegure-se de que o botão de opção **Criar um novo método** esteja selecionado e clique em **Avançar**.
3. No campo **Nome do Método**, digite performExecute.
4. Na lista suspensa **Tipo de Retorno**, selecione nulo. Clique em **Avançar**.
5. Para especificar a exceção que o método pode emitir, clique em **Incluir**. No campo **Padrão**, digite ECException, clique em **Incluir** e em **Fechar**.
6. Clique em **Concluir**.
   O método é gerado e o código fonte do método é exibido.
7. É necessário modificar o código fonte. Localize a seguinte linha no código fonte do método performExecute:

   ```
   public void performExecute() throws
      com.ibm.commerce.exception.ECException {
   ```

   Após a linha anterior, digite o seguinte código:

Você pode recortar e colar esse código da versão PDF do Manual do Programador. É recomendável que você copie inicialmente o código na janela Scrapbook no VisualAge for Java (consulte a ajuda online do VisualAge for Java para obter mais informações) e examine o código para assegurar que nenhum caractere tenha sido perdido ou modifica do durante a operação recortar e colar. Então, após validar o código, copie-o para o local de destino. Observe que copiar o texto para um outro editor pode causar a modificação de alguns caracteres.

```
final String methodName = "performExecute";
ECTrace.entry(ECTraceIdentifiers.COMPONENT_ORDER,
this.getClass().toString(), methodName);

// do all order processing as normal
  super.performExecute();

// *** updating Bonus Point Information ***

// fetch order info
```

```
theOrder = new OrderAccessBean();
theOrder.setInitKey_orderId(getOrderRn().toString());

  int bonusPt;  // bonus points for this order
  int bonusTotal;  // total bonus points
  BigDecimal subtotal;  // subtotal
  BigDecimal bonusdeter;  // bonus determinant
  BigDecimal ans;

  // determine bonus points = subtotal * bonus determinant
    try {
      subtotal = theOrder.getTotalProductPriceInEJBType();
      bonusdeter = new BigDecimal(bonusPercentAmount);
      ans = subtotal.multiply(bonusdeter);
      bonusPt = Math.round(ans.floatValue());

       System.out.println("subtotal is: " + subtotal +
              " bonus deter is: " + bonusdeter + " ans is: " + ans);
       System.out.println("Bonus Percent amount = " +
             bonusPercentAmount);
       System.out.println("Bonus calculated is: "+ bonusPt);
      }


     // Various Exceptions
catch (Exception ex) {
     throw new ECSystemException(ECMessage._ERR_GENERIC,
        this.getClass().toString(),methodName,
        ECMessageHelper.generateMsgParms(ex.getMessage()), ex);
     }

  // *** Updating bonus points in BONUS table using bean
  //     created in previous example ***

BonusAccessBean bonusBean = new BonusAccessBean();
  bonusBean.setInit_argMemberId(
     getCommandContext().getUserId().toString());
    try {
      //new bonus value = this order bonus points + Old Bonus Points
      bonusTotal = bonusPt + Integer.parseInt(bonusBean.getBonusPoint());
      bonusBean.setBonusPoint(String.valueOf(bonusTotal));
      bonusBean.commitCopyHelper();

      System.out.println("In try, BonusTotal calculated is: "+
            bonusTotal);

      }

     // Various Exceptions
  catch (FinderException e) // user does not have points setup yet
  {
     // create a row in table bonus
     bonusTotal = bonusPt;

    try {
```

```
            BonusAccessBean bonusBeanNew = new
               BonusAccessBean(getCommandContext().getUserId(),
                  new Integer(bonusTotal));

               System.out.println("In catch, BonusTotal calculated is: "+
               bonusTotal);

      }
     catch (Exception ex) {
          throw new ECSystemException(ECMessage._ERR_GENERIC,
             this.getClass().toString(), methodName,
             ECMessageHelper.generateMsgParms(ex.getMessage()), ex);
          }
    }
     catch (Exception ex) {
          throw new ECSystemException(ECMessage._ERR_GENERIC,
             this.getClass().toString(), methodName,
             ECMessageHelper.generateMsgParms(ex.getMessage()), ex);
      }

      // *** setting view details ***

      // Fetch setResponse properties and add bonus parameters
     //  needed by the JSP page
     TypedProperty resp = getResponseProperties();
     resp.put("bonus", new Integer(bonusPt).toString());
      setResponseProperties(resp);
      ECTrace.exit(ECTraceIdentifiers.COMPONENT_ORDER,
      this.getClass().toString(), methodName);
```

8. Salve seu trabalho.

Modificando o Registro de Comandos para Utilizar `OrderProcessCmdBonusImpl`

Nesse exemplo, você deve utilizar a nova classe de implementação do processamento de pedido sempre que o processamento de pedido for obrigatório. Para que isso aconteça, é necessário atualizar o registro de comandos para associar a interface OrderProcess original à nova classe de implementação `OrderProcessCmdBonusImpl`.

DB2 Se estiver utilizando um banco de dados DB2, faça o seguinte para atualizar o registro de comandos:

1. Abra o Centro de Comandos do DB2 (**Iniciar > Programas >IBM DB2 > Centro de Comandos**).
2. Com a guia Script selecionada, crie a entrada obrigatória na tabela CMDREG, digitando as seguintes informações na janela de script:

```
connect to your_database_name;
update CMDREG
set CLASSNAME='com.ibm.commerce.sample.order.OrderProcessCmdBonusImpl'
WHERE INTERFACENAME='com.ibm.commerce.order.commands.OrderProcessCmd'
and storeent_Id=0;
```

em que *your_database_name* é o nome de seu banco de dados e clique no ícone Executar
Esse comando é utilizado por todos os comerciantes (indicados pelo valor 0 para STOREENT_ID).

Oracle Se você estiver utilizando um banco de dados Oracle, faça o seguinte para atualizar o registro de comandos:

1. Abra a janela de comando Oracle SQL Plus (**Iniciar > Programas > Oracle > Desenvolvimento de Aplicativos > SQL Plus**).
2. No campo **Nome do Usuário**, digite seu nome de usuário do Oracle.
3. No campo **Senha**, digite a senha do Oracle.
4. No campo **Cadeia do Host**, digite sua cadeia de conexão.
5. Crie a entrada obrigatória na tabela URLREG, digitando as seguintes informações na janela SQL Plus:

   ```
   update CMDREG
   set CLASSNAME='com.ibm.commerce.sample.order.OrderProcessCmdBonusImpl'
   WHERE INTERFACENAME='com.ibm.commerce.order.commands.OrderProcessCmd'
   and storeent_Id=0;
   ```

   Pressione Enter para executar a instrução SQL.
   Esse comando é utilizado por todos os comerciantes (indicados pelo valor 0 para STOREENT_ID)
6. Digite o seguinte para consolidar as alterações no banco de dados:

   ```
   commit;
   ```

   e pressione Enter para executar a instrução SQL.

Modificando o Modelo`confirmation.jsp`

É necessário modificar o modelo `confirmation.jsp` para exibir a nova lógica de negócios que você incluiu no processo de negócios do processo de pedido. Para modificar o modelo de exibição, proceda da seguinte forma:

1. Navegue para o seguinte diretório:
   `vaj_drive:\VAJava\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web\`*`store_directory`*.
2. Faça uma cópia do `confirmation.jsp` e chame-o de `confirmation.jsp.bak`
3. Abra o `confirmation.jsp` em um editor de textos.
4. Após as informações de importação existentes, inclua a seguinte informação:

   ```
   <%@ page import="com.ibm.commerce.datatype.*"   %>
   ```
5. Imediatamente após a linha a seguir do modelo JSP:

   ```
   String orderRn = jhelper.getParameter("orderId");
   ```

   inclua o seguinte:

```
String bonus = ((TypedProperty)request.getAttribute(
   ECConstants.EC_REQUESTPROPERTIES)).getString("bonus");
```

6. Localize a seção a seguir no modelo JSP:

```
<tr>
<td align="left" valign="middle">
<font class="product"><%=infashiontext.getString("GRAND_TOTAL")%>
   </font></td>
<td align="right" valign="middle">
<font class="strongprice"><%=orderBean.getGrandTotal()  %></font></td>
```

   e em seguida, inclua o seguinte:

```
</tr>
<tr>
<td align="left" valign="middle">
<font class="text">Bonus Points</font></td>
<td align="right" valign="middle">
<font class="strongtext"><%=bonus  %></font></td>
```

7. Salve seu trabalho.

Testando `OrderProcessCmdBonusImpl` no WebSphere Test Environment

Você pode utilizar o WebSphere Test Environment para testar sua nova lógica de negócios. Para testar o comando `OrderProcessCmdBonusImpl`, faça o seguinte:

1. Inicie o WebSphere Test Environment conforme descrito no Apêndice A, "Iniciando e Parando o WebSphere Test Environment" na página 347.
2. Abra um navegador e digite a URL para sua loja. Por exemplo, digite a seguinte URL:

```
http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
     storeId=10001&catalogId=10001&langId=-1
```

3. Selecione e compre um produto.
4. Depois de ter comprado um produto, a confirmação do pedido exibe o número de pontos de bônus que foram obtidos no pedido.

(Opcional) Implementando a Lógica de Negócios Personalizada em um WebSphere Commerce Server Remoto

Após ter terminado de testar sua lógica de negócios no WebSphere Test Environment e estar satisfeito com seu código, você pode implementar o código em uma loja em um WebSphere Commerce Server remoto. Neste tutorial, as personalizações incluem o seguinte:

- A nova classe OrderProcessCmdBonusImpl.
- O arquivo modelo confirmation.jsp atualizado.
- O registro de comando atualizado.

Dessa forma, a implementação do código inclui as etapas a seguir:

1. Criar um arquivo JAR para a lógica de comando, utilizando as ferramentas do VisualAge for Java.

2. Copiar o arquivo JAR e modelo JSP para os diretórios apropriados no WebSphere Commerce Server de destino.
3. Atualizar o registro de comandos no WebSphere Commerce Server de destino.

Notas Sobre o Método de Pagamento de Teste

Por padrão, a loja de exemplo em execução no WebSphere Test Environment utiliza um método de pagamento de teste. Esse método de pagamento de teste é utilizado de forma que você possa concluir o fluxo de compras no WebSphere Test Environment, sem precisar pedir ajuda a um Payment Manager. Esse método de pagamento de teste permite que você conclua apenas uma compra, ele não permite que pedidos submetidos com esse método de pagamento fiquem disponíveis para processamento posterior. Dessa forma o método de pagamento de teste deve ser utilizado apenas no WebSphere Test Environment.

Certifique-se de que possa concluir uma compra na loja em que está sendo implementado este código personalizado. O processamento de pagamento pode ser feito utilizando um Payment Manager local ou remoto.

Para obter mais informações sobre o método de pagamento de teste, consulte "Método de Pagamento de Teste" na página 203.

Criando o Arquivo JAR para a Lógica do Comando

É necessário empacotar a lógica do comando em um arquivo JAR para que seja implementada para o WebSphere Commerce Server de destino. Como o `OrderProcessCmdBonusImpl` está armazenado no projeto _WCSamples, você cria um arquivo JAR para esse projeto.

Para criar o arquivo JAR, faça o seguinte:

1. Pare o WebSphere Test Environment, conforme descrito no Apêndice A, "Iniciando e Parando o WebSphere Test Environment" na página 347.
2. Clique com o botão direito do mouse no projeto **_WCSamples** e selecione **Exportar**.
   O SmartGuide Exportar é aberto.
3. Selecione **arquivo Jar** e clique em **Avançar.**
4. No campo Arquivo Jar, digite o seguinte:
   unidade`:\WebSphere\CommerceServerDev\mytemp_b\wcssamplesb_1.jar`
   em que *unidade* é a unidade na qual o Commerce Studio está instalado.
5. Selecione os atributos da seguinte forma:

Atributo	Valor
classe	Verificado
java	Não verificado

Atributo	Valor
recurso	Verificado
beans	Verificado
Incluir atributos de depuração no arquivo .class	Verificado

Aceite os valores padrão para outros atributos.

6. Clique em **Concluir**.

Como o arquivo JAR criado não contém informações completas sobre nomeação de pacotes, você deve usar outro utilitário de empacotamento (fora do VisualAge for Java) para empacotar novamente o arquivo JAR. Para empacotar novamente este arquivo, proceda da seguinte maneira:

1. Em uma janela de comando, navegue até o seguinte diretório:
   unidade:\WebSphere\CommerceServerDev\mytemp_b
2. Digite `mkdir temp1`.
3. Digite `cd temp1`.
4. Defina o caminho da seguinte forma:
   `set PATH=%PATH%;`*unidade*`:\WebSphere\WebSphereStudio4\bin;`
   em que *unidade* é a unidade na qual o WebSphere Studio está instalado.
5. Digite `jar xvf ../wcssamplesb_1.jar`
6. Digite `jar cvf ../wcssamplesb.jar *`

Armazenando recursos no WebSphere Commerce Server de destino

O arquivo JAR para a lógica de comando e o modelo `confirmation.jsp` modificado devem ser colocados nos diretórios apropriados no WebSphere Commerce Server de destino.

Para armazenar o arquivo JAR no diretório apropriado no WebSphere Commerce Server remoto, faça o seguinte:

1. Na máquina de desenvolvimento, abra uma janela de comando, navegue para o seguinte diretório:
   unidade:\WebSphere\CommerceServerDev\mytemp_b
   e localize o arquivo `wcssamplesb.jar`.
2. Copie este arquivo no seguinte diretório do WebSphere Commerce Server de destino:
   unidade:\WebSphere\AppServer\installedApps\
   WC_Enterprise_App_*instanceName*.ear\wcstores.war\WEB-INF\lib

Para armazenar o modelo `confirmation.jsp` no diretório apropriado no WebSphere Commerce Server de destino, faça o seguinte:

1. Na máquina de desenvolvimento, navegue para o seguinte diretório: `vaj_drive:\VAJava\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web\`*`store_directory`*
2. Copie o modelo `confirmation.jsp` para o seguinte diretório, no WebSphere Commerce Server de destino:

   `unidade``:\WebSphere\AppServer\installedApps\`
   `WC_Enterprise_App_`*`instanceName`*`.ear\wcstores.war\`*`storeDir`*

Atualizando o Registro de Comandos

Se você estiver implementando o comando OrderProcessCmdBonusImpl em um WebSphere Commerce Server de destino que utiliza um banco de dados diferente do WebSphere Test Environment, é necessário atualizar o banco de dados de destino para refletir as alterações feitas no registro de comandos.

DB2 Se estiver utilizando um banco de dados DB2, faça o seguinte para atualizar o banco de dados do WebSphere Commerce Server de destino:

1. Abra o Centro de Comandos do DB2 (**Iniciar > Programas >IBM DB2 > Centro de Comandos**).
2. Com a guia Script selecionada, crie a entrada obrigatória na tabela CMDREG, digitando as seguintes informações na janela de script:

   ```
   connect to your_target_database_name;
   update CMDREG
   set CLASSNAME='com.ibm.commerce.sample.order.OrderProcessCmdBonusImpl'
   WHERE INTERFACENAME='com.ibm.commerce.order.commands.OrderProcessCmd'
   and storeent_Id=0
   ```

   em que *`your_target_database_name`* é o nome de seu banco de dados e clique no ícone Executar

Oracle Se você estiver utilizando um banco de dados Oracle, faça o seguinte para atualizar o registro de comandos:

1. Abra a janela de comando Oracle SQL Plus (**Iniciar > Programas > Oracle > Desenvolvimento de Aplicativos > SQL Plus**).
2. No campo **Nome do Usuário**, digite seu nome de usuário do Oracle.
3. No campo **Senha**, digite a senha do Oracle.
4. No campo **Cadeia do Host**, digite sua cadeia de conexão.
5. Crie a entrada obrigatória na tabela CMDREG, digitando as seguintes informações na janela SQL Plus:

   ```
   update CMDREG
   set CLASSNAME='com.ibm.commerce.sample.order.OrderProcessCmdBonusImpl'
   WHERE INTERFACENAME='com.ibm.commerce.order.commands.OrderProcessCmd'
   and storeent_Id=0;
   ```

Pressione Enter para executar a instrução SQL.
Esse comando é utilizado por todos os comerciantes (indicados pelo valor 0 para STOREENT_ID)

6. Digite o seguinte para consolidar as alterações no banco de dados:

   ```
   commit;
   ```

   e pressione Enter para executar a instrução SQL.

Reiniciando seu Aplicativo Corporativo no WebSphere Application Server

Após ter incluído a lógica de comandos ao seu aplicativo corporativo, colocando os recursos de arquivo nos diretório apropriados e atualizando o registro de comandos, será necessário parar e reiniciar seu aplicativo corporativo para que a alteração seja efetivada.

Para parar e reiniciar seu aplicativo corporativo, faça o seguinte:

1. Abra o WebSphere Application Server Administration Console.
2. Expanda o **Domínio Administrativo do WebSphere**.
3. Expanda os **Nós**.
4. Expanda ***nodeName*** (em que ***nodeName*** é o nome de seu nó).
5. Expanda **Servidores de Aplicativos**.
6. Clique com o botão direito do mouse em seu aplicativo WebSphere Commerce. Por exemplo, clique com o botão direito do mouse no aplicativo **WebSphere Commerce Server - demo** e selecione **Parar**.
7. Clique com o botão direito do mouse em seu aplicativo WebSphere Commerce. Por exemplo, clique com o botão direito do mouse no aplicativo **WebSphere Commerce Server - demo** e selecione **Iniciar**.

Testando sua Nova Lógica na InFashion em Execução no WebSphere Application Server

Agora você pode verificar sua nova lógica de negócios na loja InFashion em execução no WebSphere Application Server.

Para executar essa verificação final, faça o seguinte:

1. Após a instância do WebSphere Commerce Server ser iniciada, abra um navegador e digite a URL para o home page de sua loja. Por exemplo, digite a seguinte URL:

   ```
   http://hostname/webapp/wcs/stores/servlet/StoreCatalogDisplay?
        storeId=store_Id&catalogId=catalog_Id&langId=-1
   ```

   em que *store_Id* é o identificador de sua loja e *catalog_Id* é o identificador do catálogo de sua loja.
2. Selecione e compre um produto.
3. Depois de ter comprado um produto, a confirmação do pedido exibe o número de pontos de bônus que foram obtidos no pedido.

Modificando um Bean de Entidade Existente e Extensão de um Comando de Tarefa Existente

Nota: O objetivo deste tutorial é mostrar o processo utilizado para modificar beans de entidade existentes e estender comandos de tarefa existentes. Ele não foi desenvolvido para mostrar a melhor maneira de modificar os preços de produtos. Para obter informações sobre preços com descontos, consulte o *WebSphere Commerce - Manual de Estrutura de Cálculo*.

No Capítulo 9, "Tutorial: Criando Nova Lógica de Negócios", todo um conjunto novo de lógica de negócios foi criado. Isto incluiu a criação de um novo comando de controlador, um novo modelo JSP, uma nova tabela de banco de dados, um novo bean corporativo para acessar a tabela, um bean de acesso correspondente, assim como um bean de acesso de dados. Toda esta lógica é reunida para criar um aplicativo simplificado de pontos de bônus, no qual um saldo de pontos de bônus de um usuário pode ser atualizado utilizando um modelo JSP que é iniciado pelo novo comando de controlador.

A maneira como a tabela BONUS é utilizada no Capítulo 9, "Tutorial: Criando Nova Lógica de Negócios" é descrita no seguinte diagrama:

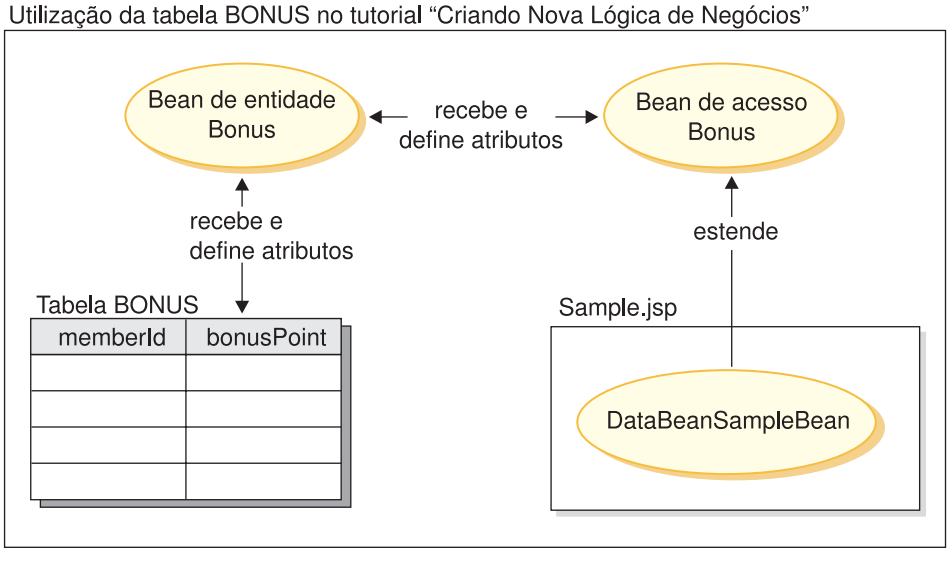


Figura 41.

No próximo tutorial, a tabela BONUS é utilizada de uma maneira diferente. De forma específica, o bean de entidade User é personalizado de tal forma que pareça para os aplicativos que a coluna BONUSPOINT da tabela BONUS

é realmente uma coluna na tabela USERS. Quando um novo registro é criado na tabela USERS, um registro correspondente é automaticamente inserido na tabela BONUS.

Para criar esta *junção de tabelas*, um novo campo CMP deve ser incluído no bean de entidade de Usuário. Este campo CMP mapeia para a coluna BONUSPOINT na tabela BONUS utilizando o recurso de Mapa de Tabelas Secundárias no Navegador de Mapas do VisualAge for Java.

Para integrar o bean de entidade User modificado no fluxo de compras, um novo preço de produtos é criado. Este novo preço leva em consideração o saldo de pontos de bônus atual do comprador. Você cria o novo preço estendendo o comando de tarefa GetProductContractUnitPriceCmd. Ao estender esse comando, você cria uma nova interface que estende a interface GetProductContractUnitPriceCmd. A nova interface inclui um atributo adicional (para o preço de bônus). Também é criada uma nova classe de implementação que estende a classe de implementação GetContractUnitPriceCmdImpl. Essa nova classe de implementação é denominada "GetNewContractUnitPriceCmdImpl". A nova classe de implementação chama o método performExecute de sua super classe e então inclui a lógica de negócios para determinar o novo preço de bônus.

O diagrama a seguir mostra como a tabela BONUS é utilizada no próximo tutorial.

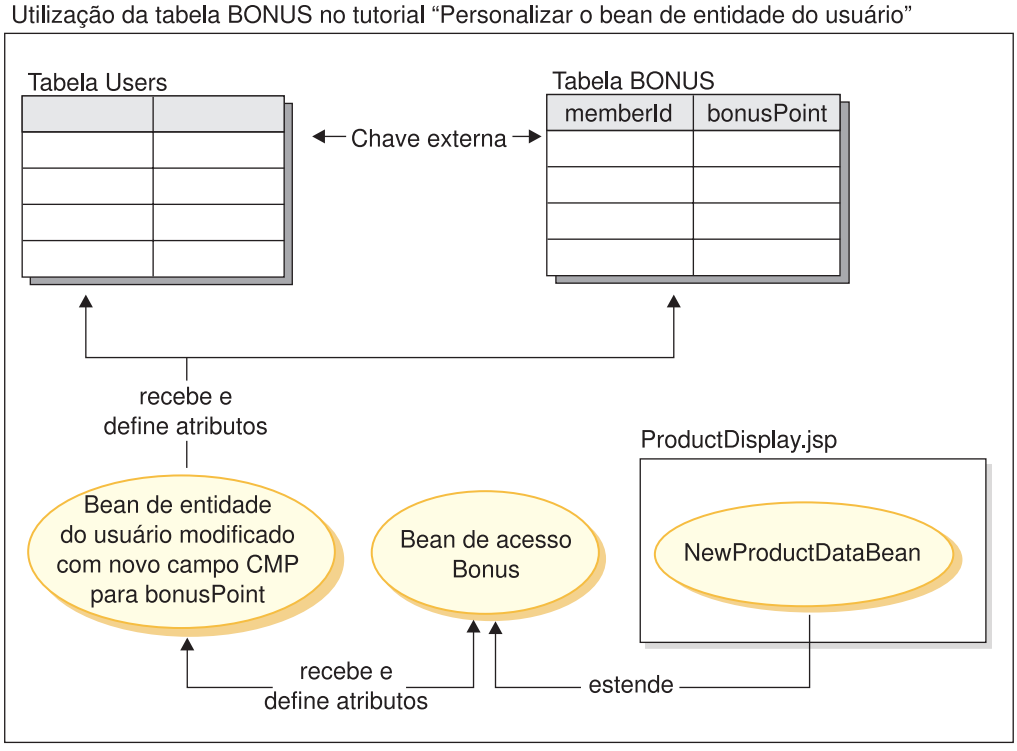


Figura 42.

Este tutorial inclui as seguintes etapas:

1. Incluir um novo campo CMP no bean de entidade do Usuário.
2. Criar e preencher a tabela BONUS.
3. Atualizar o esquema de banco de dados WCSUser e mapeamento de tabela para incluir a nova tabela BONUS.
4. Criar o relacionamento de chave externa entre as tabelas BONUS e USER.
5. Criar o mapa da tabela BONUS.
6. Gerar um código implementado e bean de acesso para o bean de entidade do Usuário.
7. Utilizar o cliente de teste para teste preliminar do bean de entidade atualizado.
8. Criar uma nova interface do comando de tarefa e classe de implementação. O novo comando de tarefa estende `GetBaseUnitProductPriceCmd`. O novo recurso mais importante nesse comando é a lógica no método performExecute() da classe de implementação. Esse método calcula um novo *preço de bônus* para o produto. Este preço de bônus é criado de modo que o preço seja reduzido pelo saldo de pontos de bônus do usuário, até uma redução

máxima de 20 por cento do preço calculado. A tabela a seguir mostra exemplos desta fórmula de redução de preço em uso:

Preço calculado	Saldo de pontos de bônus	Dedução máxima (20% do preço calculado)	Novo preço de bônus
R$1000	100	R$200	R$900 Como o saldo de pontos de bônus é menor que a dedução máxima, o preço da lista é reduzido pelos pontos de bônus.
R$1000	300	R$200	R$800 Como o saldo de pontos de bônus é maior que a dedução máxima, o preço da lista é reduzido pela dedução máxima de R$200.

9. Crie o `NewProductDataBean` que estende o `ProductDataBean`. Inclua um novo método para esse bean, de modo que o novo preço de bônus possa ser facilmente utilizado em uma página de exibição de produto.
10. Atualize a tabela JSP de exibição de produto para a loja InFashion para exibir o preço do bônus.
11. Teste a lógica de negócios na loja InFashion executando o WebSphere Test Environment.
12. (Opcional) Implemente a lógica de negócios atualizada em um WebSphere Commerce Server remoto e teste-a fora do WebSphere Test Environment.

Antes de iniciar este tutorial

Se você não concluiu o Capítulo 9, “Tutorial: Criando Nova Lógica de Negócios” na página 207, é necessário executar as etapas descritas em “Preparando Amostra de Projetos” na página 208 para iniciar este tutorial.

Incluindo um Novo Campo bonusPoint no Bean de Entidade User

Nesta seção, você utiliza as ferramentas EJB do VisualAge for Java para incluir o novo campo CMP no bean de entidade. O novo campo é denominado *bonusPoint* e, eventualmente, é mapeado para a coluna BONUSPOINT da tabela BONUS.

Para incluir o novo campo CMP no bean de entidade User, proceda da seguinte forma:

1. Se estiverem em execução, pare o mecanismo de servlet, o servidor EJB e o servidor de nome persistentes, conforme descrito em Apêndice A, "Iniciando e Parando o WebSphere Test Environment" na página 347.
2. No Workbench, clique na guia **EJB**.
3. Expanda o grupo EJB **WCSUser**.
4. Clique com o botão direito do mouse no bean **User** e selecione **Incluir > Campo CMP**.
   O SmartGuide Criar Campo CMP é aberto.
5. Crie um novo campo CMP com as seguintes propriedades:

Propriedade	Valor
Nome do Campo	`bonusPoint`
Tipo de Campo	`int`
Valor Inicial	`0`
Acessar com os métodos getter e setter	`enable`
Promover os métodos getter e setter para a interface remota	`enable`
Getter	`public`
Setter	`public`

   e clique em **Concluir**.

O campo bonusPoint é exibido no painel Propriedades. Alguns avisos podem ser exibidos, mas isso é corrigido quando o código de bean da entidade é gerado novamente.

Criando e Preenchendo a Tabela BONUS

Uma tabela de banco de dados que registra os pontos de bônus de um usuário é utilizada neste tutorial.

Se você concluiu o Capítulo 9, "Tutorial: Criando Nova Lógica de Negócios" na página 207, já está familiarizado com esta tabela. Nesse caso, é necessário

atualizar a tabela para incluir uma linha para cada usuário na tabela USERS. Se você não concluiu o Capítulo 9, "Tutorial: Criando Nova Lógica de Negócios" na página 207, é necessário criar e preencher a tabela. Instruções para cada um desses cenários são fornecidas.

DB2 Se estiver utilizando um banco de dados DB2 e precisar *atualizar* a tabela BONUS, faça o seguinte:

1. Abra o Centro de Comandos do DB2 (**Iniciar > Programas > IBM DB2 > Centro de Comandos**) e clique na guia **Script**.
2. No campo **Comando** digite

   ```
   connect to your_database_name
   ```

   em que *your_database_name* é o nome do seu banco de dados e clique no ícone Executar.
3. No campo **Comando**, digite o seguinte e depois clique no ícone Executar:

   ```
   INSERT INTO BONUS
   (SELECT USERS_ID, 0
   FROM USERS
   WHERE USERS_ID NOT IN (SELECT MEMBERID FROM BONUS))
   ```

   A tabela BONUS foi atualizada.

DB2 Se estiver utilizando um banco de dados DB2 e precisar *criar e preencher* a tabela BONUS, faça o seguinte:

1. Abra o Centro de Comandos do DB2 (**Iniciar > Programas > IBM DB2 > Centro de Comandos**) e clique na guia **Script**.
2. No campo **Comando** digite

   ```
   connect to your_database_name
   ```

   em que *your_database_name* é o nome do seu banco de dados e clique no ícone Executar.
3. No campo **Comando**, digite o seguinte e depois clique no ícone Executar:

   ```
   CREATE TABLE BONUS (MEMBERID BIGINT NOT NULL,
       BONUSPOINT INTEGER NOT NULL,
       constraint p_memberid primary key (MEMBERID),
       constraint f_memberid foreign key (MEMBERID)
       references users (users_id) on delete cascade)
   ```

   A tabela BONUS está criada.
4. Para preencher a tabela, digite o seguinte no campo **Comando** e clique no ícone Executar:

   ```
   insert into BONUS (select USERS_ID, 0 from USERS)
   ```
5. Feche o Centro de Comandos do DB2.

Oracle Se você estiver utilizando um banco de dados Oracle e precisar *atualizar* a tabela BONUS, faça o seguinte:

1. Abra a janela de comando Oracle SQL Plus (**Iniciar > Programas > Oracle > Desenvolvimento de Aplicativos > SQL Plus**).
2. No campo **Nome do Usuário**, digite seu nome de usuário do Oracle.
3. No campo **Senha**, digite a senha do Oracle.
4. No campo **Cadeia do Host**, digite sua cadeia de conexão.
5. Atualize a tabela BONUS, digitando as seguintes informações na janela SQL Plus:

   ```
   INSERT INTO BONUS
   (SELECT USERS_ID, 0
   FROM USERS
   WHERE USERS_ID NOT IN (SELECT MEMBERID FROM BONUS));
   ```

   Pressione Enter para executar a instrução SQL.
   A tabela BONUS agora foi atualizada.
6. Digite o seguinte para consolidar as alterações no banco de dados:

   ```
   commit;
   ```

   e pressione Enter para executar a instrução SQL.

Oracle Se você estiver utilizando um banco de dados Oracle e precisar *criar e preencher* a tabela BONUS, faça o seguinte:

1. Abra a janela de comando Oracle SQL Plus (**Iniciar > Programas > Oracle > Desenvolvimento de Aplicativos > SQL Plus**).
2. No campo **Nome do Usuário**, digite seu nome de usuário do Oracle.
3. No campo **Senha**, digite a senha do Oracle.
4. No campo **Cadeia do Host**, digite sua cadeia de conexão.
5. Crie a tabela BONUS, digitando as seguintes informações na janela SQL Plus:

   ```
   CREATE TABLE Bonus (MEMBERID NUMBER NOT NULL,
       BONUSPOINT INTEGER NOT NULL,
       constraint p_memberid primary key (MEMBERID),
       constraint f_memberid foreign key (MEMBERID)
       references users (users_id) on delete cascade);
   ```

   Pressione Enter para executar a instrução SQL.
   A tabela BONUS agora foi criada.
6. Preencha a tabela BONUS, digitando o seguinte:

   ```
   insert into BONUS (select USERS_ID, 0 from USERS);
   ```
7. Digite o seguinte para consolidar as alterações no banco de dados:

   ```
   commit;
   ```

e pressione Enter para executar a instrução SQL.

Atualizando o Esquema e o Mapeamento da Tabela

Nas seções seguintes, você atualiza o esquema WCSUser com a nova tabela BONUS, cria o relacionamento de chave externa para a nova tabela e cria um mapa de tabela entre os campos do bean de entidade do Usuário e as colunas da tabela BONUS.

Criando o Esquema da Tabela BONUS

Para criar o esquema de tabela, proceda da seguinte forma:

1. Clique com o botão direito do mouse no bean de entidade **User** e selecione **Abrir Para > Esquema do Banco de Dados**.
   A janela Navegador de Esquema é aberta.
2. Na lista **Esquemas**, selecione **Usuário do WCS**.
3. No menu **Tabelas**, selecione **Nova Tabela**.
   O Editor de Tabela é aberto.
4. No campo **Nome**, digite `BONUS`.
5. Deixe os campos **Qualificador** e **Nome Físico** em branco.
6. Na seção Colunas de Tabela, clique em **Nova**. O Editor de Colunas é aberto. Crie uma coluna, como segue:

Atributo	Valor
Nome	`memberId`
Nome físico	Deixe este campo em branco.
Tipo	`BIGINT`
Detalhes do Tipo	`VapConverter`
Permitir nulos	não marcado

   e clique em **OK**.
7. Selecione **memberId** na lista de colunas de Tabela e clique em **>>** para utilizar esta coluna como chave principal.
8. Crie outra coluna (clique em **Nova** novamente), como segue:

Atributo	Valor
Nome	`bonusPoint`
Nome físico	Deixe este campo em branco.
Tipo	`INTEGER`
Detalhes do Tipo	`VapConverter`
Permitir nulos	marcado

   e clique em **OK**.

9. Clique em **OK** no Editor de Tabelas.
   Depois de alguns instantes, a tabela **BONUS** é listada na lista **Tabelas** da janela Navegador de Esquemas.
10. Para verificação, clique em **BONUS** na lista **Tabelas** e verifique se as suas duas colunas estão exibidas na lista **Colunas**.

Criação de relacionamento de chave externa

Nesta seção, você estabelece um relacionamento de chave externa entre as tabelas BONUS e USERS.

Para criar o relacionamento de chave externa, proceda da seguinte forma:

1. Na janela Navegador de Esquema, clique no esquema **Usuário do WCS**. Os relacionamentos de tabelas e chave externa para este esquema são exibidos.
2. No menu **Chaves Externas**, selecione **Novo Relacionamento de Chave Externa**.
   O Editor de Relacionamento de Chave Externa é aberto.
3. No campo **Nome**, `F_User_Bonus`.
4. Assegure-se de que a caixa de opções **Limitação existe no Banco de Dados** esteja marcada.
5. Na lista suspensa **Tabela de Chave Principal**, selecione `USERS`
6. Na lista suspensa **Tabela de chave externa**, selecione `BONUS`
7. Clique no campo vazio abaixo do título **Chave Externa** para que uma lista suspensa seja exibida.Nessa lista, selecione **ID do membro** e clique em **OK**.
   `F_User_Bonus` é exibido na lista de relacionamentos de chaves externas.
8. No menu **Esquemas**, selecione **Salvar Esquema** e em seguida, clique em **Concluir**.
9. Feche o Navegador de Esquema.

Criação do mapa da tabela BONUS

Nesta seção, você cria o mapeamento entre a coluna `BONUSPOINT` na tabela BONUS e o campo `bonusPoint` no bean de entidade User.

Para criar o mapa da tabela BONUS, proceda da seguinte maneira:

1. Assegure que o Workbenck esteja aberto com a guia EJB selecionada.
2. No menu **EJB**, selecione **Abrir Para > Mapas de Esquema**.
   O Navegador de Mapas é aberto.
3. Na Lista de mapas Datastore, selecione **Usuário do WCS**.
4. Na lista **Classes Persistentes**, proceda da seguinte maneira:
   a. Dê um clique duplo em **Membro**.

b. Selecione **Usuário** (Observe que **Usuário** é exibido abaixo de **Membro** apenas depois que Membro é expandido na etapa 4a).

5. No menu **Mapas da Tabela**, selecione **Novo Mapa de Tabela > Incluir Mapa de Tabela Secundária**.
   A janela Mapa de Tabela Secundária é aberta.
6. Na lista suspensa **Tabela**, selecione **BONUS**.
7. Na lista suspensa Relacionamento de Chave Externa, selecione **F_User_Bonus** e clique em **OK**.
   Após alguns momentos, o mapa `BONUS (secundário)` é exibido na lista de Mapas de Tabelas.
8. Destaque e clique com o botão direito do mouse no mapa de tabela **BONUS (secundário)** e selecione **Editar Mapas de Propriedades**.
   O Editor de Mapa de Propriedades é aberto.
9. Role até a fileira que exibe o atributo de classe do bonusPoint. Selecione **bonusPoint**.
10. Clique na entrada correspondente na coluna **Tipo de Mapa** e selecione **Simples** na lista suspensa.
11. Clique na entrada correspondente na coluna **Coluna da Tabela** e selecione **bonusPoint** na lista suspensa.
12. Deixe todos os outros campos inalterados e clique em **OK**.
    O novo mapa de propriedade é gerado e depois de alguns instantes

    ```
    (a) bonusPoint (bonusPoint)
    ```

    é exibido na coluna **Mapas de Propriedades** do Navegador de Mapas.
13. Clique com o botão direito do mouse no mapa datastore **Usuário do WCS**, selecione **Salvar Mapa Datastore** e em seguida, clique em **Concluir**.
14. Feche o Navegador de Mapa.

Neste ponto, o bean do Usuário contém erros que indicam que algumas classes abstratas não estão implementadas. Estes erros são corrigidos quando o código implementado é gerado.

Gerando o Código e Bean de Acesso Implementados

Como você modificou o código para o bean de Usuário, deve regenerar seu código implementado, assim como seu bean de acesso. As ferramentas em VisualAge for Java tornam esta etapa de geração de código muito simples.

Para executar essa etapa, proceda da seguinte forma:

1. Assegure-se de que o Workbenck esteja aberto com a guia EJB selecionada.
2. Expanda o grupo EJB **WCSUser**.
3. Clique com o botão direito do mouse no bean **Usuário** e selecione **Gerar Código Implementado**.

Se for exibida uma janela de mensagem perguntando se deseja criar uma edição do pacote, clique em **Sim**.
A geração do código demora alguns minutos.

4. Uma vez gerado o código implementado, clique com o botão direito do mouse no bean **Usuário** novamente e selecione **Incluir > Bean de Acesso**. O SmartGuide Criar Bean de Acesso é aberto.
5. Aceite os valores padrão na página Selecionar Propriedades do Bean de Acesso e clique em **Avançar**.
6. Aceite os valores padrão na página Definir Construtor de Argumento Zero e clique em **Avançar**.
7. Na página Selecione e Personalize Propriedades do Bean para Copy Helper, role até **bonusPoint** na coluna Bean Corporativo. Marque **Copy Helper** para o bean e defina o conversor para `com.ibm.commerce.base.objects.WCSStringConverter`.
8. Clique em **Concluir**.
9. Clique em **OK** quando a Mensagem "Geração de código concluída" for exibida.

Testando a Modificação Utilizando o Cliente de Teste

Um cliente de teste pode ser utilizado para testar o bean de identidade Usuário recém personalizado. Para iniciar o cliente de teste e testar o bean de entidade, proceda da seguinte maneira:

1. Inicie o servidor de nomes persistente, conforme descrito em "Iniciando e Parando o Servidor de Nomes Persistente" na página 347.
2. Inicie o servidor EJB que tenha o grupo EJB WCSUser, conforme descrito em "Iniciando e Parando o Servidor EJB" na página 348.
3. No painel Beans Corporativos, expanda o grupo de EJBs **WCSUser**. Clique com o botão direito do mouse no bean de entidade **Usuário** e selecione **Executar Cliente de Teste**.
4. Na janela Procura de EJB, clique em **Procurar**.
5. Da lista de métodos, selecione **findByPrimaryKey(MemberKey)**.
6. No painel Detalhes, clique em **<nulo>**, digite `-1000` na caixa de argumentos e clique no ícone Chamar na janela Cliente de Teste EJB.

   Note que o valor digitado aqui deve equiparar-se a um valor na coluna USERS_ID da tabela do USUÁRIO.

   Quando a execução deste método estiver concluída, as informações para o usuário com ID -1000 serão exibidas em árvore no painel Métodos. Nessa exibição, há um nó denominado *métodos*.
7. Expanda o nó **Métodos**.
8. Clique em **getBonusPoint()** e no ícone Chamar.
   Este método recupera o saldo de pontos de bônus do cliente. O saldo atual dos pontos de bônus é retornado.

9. Clique em **setBonusPoint(int)**, insira `100` no painel Detalhes e clique no ícone Chamar.
10. Clique em **getBonusPoint()** e no ícone Chamar.
    Um valor 100 será retornado. Isto mostra que o banco de dados foi atualizado com êxito pelo bean corporativo do Usuário.
11. Feche as janelas de Usuário e Teste de Cliente EJB.

Criando a Interface `GetNewProductContractUnitPriceCmd`

Como parte deste exercício de personalização, você cria novas lógicas de negócios para calcular um preço com desconto, baseado no saldo de pontos de bônus do cliente. Este cálculo é executado por um novo comando de tarefa. Nesta seção, você cria a interface para este novo comando de tarefas.

Para criar a interface para o novo comando de tarefas, proceda da seguinte forma:

1. No Workbench, com a guia Projetos selecionada, expanda o projeto **_WCSamples**.
2. Clique com o botão direito do mouse no pacote **com.ibm.commerce.sample.commands** e selecione **Incluir > Interface**.
3. Verifique se a opção **Criar uma nova interface** está selecionada e, no campo **Nome da interface**, digite `GetNewProductContractUnitPriceCmd`.
4. Selecione a interface que deverá ser estendida clicando em **Incluir**, em seguida, no campo **Padrão**, digite `com.ibm.commerce.price.commands.GetProductContractUnitPriceCmd`, clique em **Incluir** e em seguida, em **Fechar**.
5. Clique em **Avançar**.
6. Para incluir as instruções de importação apropriadas, clique em **Incluir Pacote** e proceda da seguinte maneira:
   a. No campo **Padrão**, digite `com.ibm.commerce.price.utils` e clique em **Incluir**.
   b. No campo **Padrão**, digite `com.ibm.commerce.exception` e clique em **Incluir**.
   c. Clique em **Fechar**.
7. Assegure-se de que **Fazer a interface pública** esteja selecionado.
8. Clique em **Concluir**.
   O código fonte da nova interface é exibida no painel Fonte.
9. Crie uma assinatura de método para o método `getBonusPrice()`, procedendo da seguinte maneira:
   a. Clique com o botão direito do mouse na interface **GetNewProductContractUnitPriceCmd** e selecione **Incluir > Método**.
      O SmartGuide Criar Método é aberto.

b. Certifique-se de que **Criar um novo método** esteja selecionado e clique em **Avançar**.
c. No campo Nome do Método, digite `getBonusPrice`.
d. Para selecionar o tipo de retorno do método, clique em **Navegar**. No campo **Padrão**, digite `MonetaryAmount` , clique em **OK**, e então clique em **Avançar**.
e. Para selecionar a exceção que o método pode emitir, clique em **Incluir**. No campo **Padrão**, digite `ECSystemException`, clique em **Incluir** e em seguida, em **Fechar**.
f. Clique em **Concluir**.

10. Inclua um novo campo na interface que especifique a classe padrão de implementação para o comando, procedendo da seguinte maneira:
    a. Clique com o botão direito do mouse na interface **GetNewProductContractUnitPriceCmd** e selecione **Incluir > Campo**. O SmartGuide Criar Campo é aberto.
    b. No campo **Nome do Campo**, digite `defaultCommandClassName`.
    c. Na lista suspensa **Tipo de Campo**, selecione **Cadeia**.
    d. No campo **Valor Inicial**, digite

    ```
    "com.ibm.commerce.sample.commands.
       GetNewContractUnitPriceCmdImpl"
    ```

    e clique em **Concluir**.

    Notas:

    1) Você deve incluir aspas ao digitar esse valor.
    2) Se uma mensagem de aviso indicando que um campo na superclasse ficará oculto pela criação desse novo campo, clique em **Sim** para continuar.

11. Inclua um novo campo na interface que especifique o nome do comando, procedendo da seguinte maneira:
    a. Clique com o botão direito do mouse na interface **GetNewProductContractUnitPriceCmd** e selecione **Incluir > Campo**. O SmartGuide Criar Campo é aberto.
    b. No campo **Nome do Campo**, digite `NAME`.
    c. Na lista suspensa **Tipo de Campo**, selecione **Cadeia**.
    d. No campo **Valor Inicial**, digite

    ```
    "com.ibm.commerce.sample.commands.
       GetNewProductContractUnitPriceCmd"
    ```

    e clique em **Concluir**.

Criando a Classe de Implementação GetNewContractUnitPriceCmdImpl

Você deve criar a classe de implementação para o novo comando de tarefa. Esta classe de implementação implementa sua interface recém criada e contém a lógica de negócios para o comando de tarefa.

Para criar a classe de implementação GetNewContractUnitPriceCmdImpl, proceda da seguinte maneira:

1. No Workbench, com a guia Projetos selecionada, expanda o projeto **_WCSamples**.
2. Clique com o botão direito do mouse no pacote **com.ibm.commerce.sample.commands** e selecione **Incluir > Classe**. O SmartGuide Criar Classe é aberto.
3. Certifique-se de que **Criar uma nova classe** esteja selecionado e crie a classe como segue:
   a. No campo **Nome da classe**, digite GetNewContractUnitPriceCmdImpl.
   b. Para especificar a superclasse, clique em **Procurar**, em seguida, digite com.ibm.commerce.price.commands.GetContractUnitPriceCmdImpl no campo **Padrão** e clique em **OK**.
   c. Clique em **Avançar**.
   d. Para especificar os pacotes que deverão ser importados, clique em **Incluir Pacote**. No campo **Padrão**, digite os seguintes pacotes:
      - com.ibm.commerce.command e clique em **Incluir**
      - com.ibm.commerce.exception e clique em **Incluir**
      - com.ibm.commerce.price.commands e clique em **Incluir**
      - com.ibm.commerce.price.utils e clique em **Incluir**
      - com.ibm.commerce.ras e clique em **Incluir**
      - com.ibm.commerce.server e clique em **Incluir**
      - java.math, clique em **Incluir** e em seguida, em **Fechar**.
   e. Para especificar as interfaces que a classe deverá implementar, clique em **Incluir**. No campo Padrão, digite as seguintes interfaces:
      - GetContractSpecialPriceCmd e clique em **Incluir**.
      - GetContractUnitPriceCmd e clique em **Incluir**.
      - GetProductContractUnitPriceCmd e clique em **Incluir**.
      - GetNewProductContractUnitPriceCmd e clique em **Incluir** e em seguida, em **Fechar**.
   f. Clique em **Concluir**.
4. Clique com o botão direito do mouse na classe **GetNewContractUnitPriceCmdImpl** e selecione **Incluir > Campo**. O SmartGuide Criar Campo é aberto. Digite as informações como seguem:
   a. No campo **Nome do Campo**, digite bonusPrice .

b. Para selecionar o tipo de campo, clique em **Procurar**. No campo Padrão, digite `MonetaryAmount` e clique em **OK**.

c. Clique em **Concluir**.

5. Inclua um novo método `performExecute()` na classe. Este método procede como a seguir:
   - chama o método `performExecute()` da superclasse (GetContractUnitPriceCmdImpl)
   - define os valores de `thisClass` e `methodName` a serem utilizados pelo mecanismo de tratamento de exceções
   - instância um StoreAccessBean
   - instância um UserAccessBean
   - obtém o preço original do produto
   - calcula o desconto máximo aplicável
   - calcula o novo preço de bônus (este preço é do tipo *double*)
   - obtém o tipo de moeda para a loja
   - arrendondar o preço do bônus e armazená-lo como um valor monetário na moeda correta

   Para incluir esse método, clique com o botão direito do mouse na classe **GetNewContractUnitPriceCmdImpl** e selecione **Incluir > Método**. O SmartGuide Criar Método é aberto. Digite as informações como seguem:

   a. Certifique-se de que **Criar um novo método** esteja selecionado e clique em **Avançar**.

   b. No campo **Nome do Método**, digite `performExecute`.

   c. Na lista suspensa **Tipo de Retorno**, selecione **void** e clique em **Avançar**.

   d. Para selecionar a exceção que o método pode emitir, clique em **Incluir**. No campo **Padrão**, digite `ECException`, clique em **Incluir** e em seguida, em **Fechar**.

   e. Clique em **Concluir**.

   f. Após esta linha no código fonte:

```
public void performExecute()
   throws com.ibm.commerce.exception.ECException
{
```

inclua o seguinte código:

Você pode recortar e colar esse código da versão PDF do Manual do Programador. É recomendado que você inicialmente copie o código na janela Scrapbook no VisualAge for Java (consulte a ajuda online do VisualAge for Java para obter mais informações) e inspecione o código para assegurar-se que nenhum caractere foi perdido durante a operação de recortar e colar. Então, após validar o código, copie-o para o local de destino. Observe que copiar o texto para um outro editor pode causar a modificação de alguns caracteres.

```
     super.performExecute();

// Get and set this class name and method
// for use when exceptions occur.
final String thisClass =
   GetContractUnitPriceCmdImpl.class.getName();
final String methodName = "performExecute";

//get the store access bean
Integer storeId = getStoreId();
com.ibm.commerce.common.objects.StoreAccessBean storeAB =
     getCommandContext().getStore(storeId);

//get the user access bean
com.ibm.commerce.user.objects.UserAccessBean bonusAB =
     new com.ibm.commerce.user.objects.UserAccessBean();

// get the calculated price from the GetContractUnitPriceCmdImpl
MonetaryAmount priceOrg = super.getPrice();
double dblPriceOrg = priceOrg.getValue().doubleValue();

//calculate the maximum bonus that can apply to this product
double dblBonusPrice; // = dblPriceOrg;
double dblMaxBonusPoint = 0;

try {
bonusAB.setInitKey_MemberId(super.getUserId().toString());
bonusAB.refreshCopyHelper();
double dblMaxDed = dblPriceOrg * 0.2;
dblMaxBonusPoint =
     (new java.math.BigDecimal(
        bonusAB.getBonusPoint())).doubleValue());
if (dblMaxBonusPoint > dblMaxDed) dblMaxBonusPoint = dblMaxDed;
} catch (javax.ejb.CreateException ex) {
throw new ECSystemException(
     ECMessage._ERR_CREATE_EXCEPTION, thisClass, methodName, ex);
} catch (javax.ejb.FinderException ex) {

} catch (javax.naming.NamingException ex) {
throw new ECSystemException(
     ECMessage._ERR_GENERIC, thisClass, methodName, ex);
} catch (java.rmi.RemoteException ex) {
throw new ECSystemException(
```

```
            ECMessage._ERR_REMOTE_EXCEPTION, thisClass, methodName, ex);
        }


        //apply the maximum applicable bonus to this product price
        dblBonusPrice = dblPriceOrg - dblMaxBonusPoint ;

        //get the currency of this store
        CommandContext context = getCommandContext();
        String requestedCurrency = Helper.getCurrency( context, storeAB );

        //round off and return the bonus price in MonetoryAmount type
        bonusPrice = new MonetaryAmount(
             new BigDecimal(dblBonusPrice), requestedCurrency);
        CurrencyManager.getInstance().roundCustomized(bonusPrice, storeAB);
```

Salve seu trabalho.

6. Selecione o método **getBonusPrice()** na classe **GetNewContractUnitPriceCmdImpl** para exibir seu código fonte. No código fonte, altere

   ```
   return null;
   ```

   para

   ```
   return bonusPrice;
   ```

   Salve seu trabalho.

Criando o Bean de Dados `NewProductDataBean`

O método `getCalculatedBonusPrice()` deve ser incluído no ProductDataBean do WebSphere Commerce. Como você não deve realmente modificar o código do ProductDataBean, é necessário criar um novo bean de dados que estenda o ProductDataBean e incluir o método no novo bean de dados.

Para criar o novo bean de dados, proceda da seguinte maneira:

1. No Workbench, com a guia Projetos selecionada, expanda o projeto **_WCSamples**.
2. Clique com o botão direito do mouse no pacote **com.ibm.commerce.sample.databeans** e selecione **Incluir > Classe**. O SmartGuide Criar Classe é aberto. Digite as informações como seguem:
   a. No campo **Nome da Classe**, digite `NewProductDataBean` .
   b. Para especificar a superclasse, clique em **Procurar** e, no campo **Padrão**, digite `com.ibm.commerce.catalog.beans.ProductDataBean`. Clique em **OK** e em seguida, em **Avançar**.
   c. Inclua as instruções de importação apropriadas na classe clicando em **Incluir Pacote** e proceda da seguinte maneira:
      - Digite `com.ibm.commerce.beans` e clique em **Incluir**.

- Digite com.ibm.commerce.catalog.objects e clique em **Incluir**.
- Digite com.ibm.commerce.command e clique em **Incluir**.
- Digite com.ibm.commerce.datatype e clique em **Incluir**.
- Digite com.ibm.commerce.exception e clique em **Incluir**.
- Digite com.ibm.commerce.price.beans e clique em **Incluir**.
- Digite com.ibm.commerce.ras e clique em **Incluir**.
- Digite com.ibm.commerce.sample.commands e clique em **Incluir**.
- Digite com.ibm.commerce.server e clique em **Incluir**.
- Digite java.util, clique em **Incluir**, e em seguida, em **Fechar**.

d. Clique em **Concluir**.

3. Clique com o botão direito do mouse na classe **NewProductDataBean** e selecione **Incluir > Método**.
   O SmartGuide Incluir Método é aberto.
4. Assegure-se de que **Criar novo método** esteja selecionado e clique em **Avançar**.
5. No campo **Nome do Método**, digite getCalculatedBonusPrice.
6. Para selecionar o tipo de retorno do método, clique em **Procurar**. No campo **Padrão**, digite PriceDataBean , clique em **OK** e em seguida, clique em **Avançar**.
7. Para selecionar a exceção que o método pode emitir, clique em **Incluir**. No campo **Padrão**, digite ECSystemException, clique em **Incluir** e em seguida, em **Fechar**.
8. Clique em **Concluir**.
9. No código fonte, substitua return null; pelo seguinte:

```
PriceDataBean ibnPrice = null;
try {
     GetNewProductContractUnitPriceCmd comm =
        (GetNewProductContractUnitPriceCmd) CommandFactory.createCommand
        (GetNewProductContractUnitPriceCmd.NAME,
             getCommandContext().getStoreId());

     ECTrace.trace(ECTraceIdentifiers.COMPONENT_CATALOG,
     this.getClass().getName(), "getCalculatedBonusPrice",
     "Getting Price for CatalogEntry: " + getProductID());

     comm.setCatEntryId(new Long(getProductID()));
     comm.setCommandContext(getCommandContext());
     comm.execute();
     ibnPrice = new PriceDataBean(comm.getBonusPrice(),
          getCommandContext().getStore(),
          getCommandContext().getLanguageId());

} catch (Exception e) {
     throw new ECSystemException(ECMessage._ERR_RETRIEVE_PRICE,
```

```
            this.getClass().getName(), "getCalculatedBonusPrice",e);
}

return ibnPrice;
```

Salve seu trabalho.

Incluindo o Novo Preço de Bônus no Modelo de Exibição do Produto

A próxima etapa é incluir o novo preço de bônus no modelo de exibição de produto, para que os compradores possam ver o preço personalizado. Quando você tiver atualizado o modelo de exibição, o novo preço com desconto será exibido.

A loja de exemplo utiliza o modelo `ProductDisplay.jsp` para exibir produtos. Portanto, você deve atualizar este modelo com informações para exibir o novo preço.

Para atualizar o modelo de exibição, proceda da seguinte maneira:

1. Navegue para o seguinte diretório:
   vaj_drive`:\VAJava\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web\`*store_name*
2. Faça uma cópia do arquivo `ProductDisplay.jsp` e dê a ele o nome de `ProductDisplay.jsp.bak`.
3. Abra o `ProductDisplay.jsp` em um editor de texto.
4. Depois de `<%@ page import="com.ibm.commerce.common.beans.*" %>` inclua a seguinte instrução de importação:
   ```
   <%@ page import="com.ibm.commerce.sample.commands.*" %>
   <%@ page import="com.ibm.commerce.sample.databeans.*" %>
   ```
5. Substitua todas as ocorrências de `ProductDataBean` por `NewProductDataBean`.
6. Localize a seguinte linha:
   ```
   <font class="price"><%=product.getCalculatedContractPrice()%></font>
      <br><br>
   ```

   Após esta linha, insira a seguinte linha para recuperar e exibir o preço de bônus para o produto
   ```
   <font class="price"><%=product.getCalculatedBonusPrice()%>
         Bonus Price </font>
   <br><br>
   ```
7. Salve esse arquivo.

Nota: Se você recortar e colar seções no modelo de exibição da versão PDF do *WebSphere Commerce - Manual do Programador*, assegure-se de que nenhum caractere seja modificado durante esse processo.

Testando a Extensão do Bean Corporativo

Nesta seção, você pode testar a extensão feita para o bean corporativo, exibindo um produto na loja de exemplo InFashion. O novo preço de bônus é exibido.

Note, para a simplicidade desta amostra, o preço do bônus é visível a todos compradores (registrados ou compradores convidados). Para compradores que não tenham nenhum ponto de bônus, o preço do bônus será igual ao preço regular.

Para testar a extensão do bean corporativo e ver o preço de bônus exibido, proceda da seguinte maneira:

1. Verifique se o projeto `_WCSamples` está incluído no caminho do Mecanismo do Servlet, procedendo da seguinte maneira:
   a. No menu **Área de trabalho** do VisualAge for Java, selecione **Ferramentas > WebSphere Test Environment**.
      O WebSphere Test Environment Control Center é aberto.
   b. Clique em **Mecanismo de Servlet**.
   c. Se o Mecanismo de Servlet estiver em execução, clique em **Parar Mecanismo de Servlet** e em **Editar Caminho da Classe**.
   d. Se **_WCSamples** ainda não estiver selecionado, selecione-o agora e clique em **OK**.
2. Inicie o WebSphere Test Environment conforme descrito no Apêndice A, "Iniciando e Parando o WebSphere Test Environment" na página 347. O servidor de nomes persistente e o servidor EJB podem já estar em execução, e neste caso é necessário iniciar apenas o mecanismo de servlet.
3. Abra um navegador e digite a seguinte URL:

   ```
   http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
       storeId=store_Id&catalogId=catalog_Id&langId=-1
   ```

4. Clique no link **Registrar** sob o título Serviços e clique em **Registrar** sob o título Novo Cliente.
   Registre um novo cliente utilizando o endereço de e-mail `wctester@wc` e a senha `wctester1`. Preencha outros campos com valores de teste e clique em **Enviar**. Deixe o navegador aberto.
5. DB2 Abra o Centro de Comandos do DB2 e faça o seguinte:
   a. Clique na guia **Interativo**
   b. No campo **Comando**, proceda da seguinte maneira:
      1) Digite

         ```
         connect to your_database_name
         ```

         em que *your_database_name* é o nome do seu banco de dados do WebSphere Commerce e clique no ícone Executar.

2) Digite `select users_id from userreg where logonid = 'wctester@wc'` e clique no ícone Executar.

c. A guia Resultados da Consulta exibe a entrada para o cliente que você registrou na etapa 4. Anote o valor de USERS_ID do cliente aqui: _____

d. Atualize o saldo de pontos de bônus do cliente recém registro. Clique na guia Interativo e no campo **Comando**, digite o seguinte:

```
update BONUS set BONUSPOINT = 1000 where MEMBERID = users_id
```

em que *users_id* é o valor da etapa 5c. Clique no ícone Executar.

6. Oracle Atualize o saldo de pontos de bônus do usuário de teste, fazendo o seguinte:

a. Abra a janela de comando Oracle SQL Plus (**Iniciar > Programas > Oracle > Desenvolvimento de Aplicativos > SQL Plus**).

b. No campo **Nome do Usuário**, digite seu nome de usuário do Oracle.

c. No campo **Senha**, digite a senha do Oracle.

d. No campo **Cadeia do Host**, digite sua cadeia de conexão.

e. Digite `select users_id from userreg where logonid = 'wctester@wc';`

f. A entrada do cliente registrado na etapa 4 será exibida. Anote o valor de USERS_ID do cliente aqui: _____

g. Atualize o saldo de pontos de bônus do cliente recentemente registrado, digitando o seguinte:

```
update BONUS set BONUSPOINT = 1000 where MEMBERID = users_id;
```

em que *users_id* é o valor da etapa 6f.

h. Digite o seguinte para consolidar as alterações no banco de dados:

```
commit;
```

e pressione Enter para executar a instrução SQL.

7. No navegador, clique no link **Masculino** para exibir a seção de moda masculina da loja.

8. Clique no link do produto destacado para exibir a página do produto. A página exibe o preço normal e o preço com desconto com base no saldo de pontos de bônus do cliente.

Nota: Se for exibido um rastreio de pilha em vez do preço do bônus, pode ser necessário desativar a cache. Isso pode ser feito definindo o valor do componente CacheDaemon para false no arquivo *instance_name*`.xml`, conforme mostrado a seguir:

```
<component compClassName=
      "com.ibm.commerce.cache.daemon.CacheDaemonComponent"
               enable="false"
               name="CacheDaemon" />
```

Depois de alterar o valor no arquivo *instance_name*.xml, você deve parar e reiniciar o mecanismo de servlet no Ambiente de Teste do WebSphere.

(Opcional) Implementando a Lógica de Negócios Personalizada em um WebSphere Commerce Server Remoto

Esta seção descreve como implementar o bean de entidade modificado e o novo comando de tarefa para a loja que está sendo executada fora do WebSphere Test Environment.

A implementação envolve a criação de arquivos JAR para os beans corporativos públicos do WebSphere Commerce e a lógica de comando e do bean de dados, colocação dos arquivos JAR nos diretórios apropriados no servidor de destino, parada da instância do WebSphere Commerce, modificação dos caminhos da classe, implementação dos beans corporativos utilizando o utilitário XMLConfig e reinício da instância.

Criando o Arquivo JAR para o Novo Comando de Preço

É necessário criar um arquivo JAR para o projeto **_WCSamples**, para que o novo comando de tarefa seja implementado. Para criar esse arquivo JAR, faça o seguinte na máquina de desenvolvimento:

1. Pare o WebSphere Test Environment, conforme descrito no Apêndice A, "Iniciando e Parando o WebSphere Test Environment" na página 347.
2. Com a guia Projetos selecionada, selecione o projeto **_WCSamples**.
3. Com o projeto destacado, clique com o botão direito do mouse e selecione **Exportar**.
   O SmartGuide Exportar é aberto.
4. Selecione **arquivo Jar** e clique em **Avançar.**
5. No campo **Arquivo Jar**, digite o seguinte:
   unidade:\WebSphere\CommerceServerDev\mytemp_c\wcssamplesc_1.jar
   em que *unidade* é a unidade na qual o WebSphere Commerce está instalado.
6. Selecione os atributos da seguinte forma:

Atributo	Valor
classe	Verificado
java	Não verificado
recurso	Verificado
beans	Verificado

Atributo	Valor
Incluir atributos de depuração no arquivo .class	Verificado

Aceite os valores padrão para outros atributos.

7. Clique em **Concluir**.

Como o arquivo JAR criado não contém informações completas sobre nomeação de pacotes, você deve usar outro utilitário de empacotamento (fora do VisualAge for Java) para empacotar novamente o arquivo JAR. Para empacotar novamente este arquivo, proceda da seguinte maneira:

1. Em uma janela de comando, navegue até o seguinte diretório: *unidade*:\WebSphere\CommerceServerDev\mytemp_c
2. Digite `mkdir temp3`.
3. Digite `cd temp3`.
4. Defina o caminho da seguinte forma: `set PATH=%PATH%;`*unidade*`:\WebSphere\WebSphereStudio4\bin;` em que *unidade* é a unidade na qual o WebSphere Studio está instalado.
5. Digite `jar xvf ../wcssamplesc_1.jar`.
6. Digite `jar cvf ../wcssamplesc.jar *` (observe que _1 é removido do nome).

Criando um Arquivo JAR para o Grupo EJB WCSUser

Será necessário criar um arquivo JAR de Exportação EJB 1.1 contendo o grupo EJB que contenha o bean corporativo modificado. Dessa maneira, será necessário selecionar o seguinte grupo ao criar o arquivo JAR:

- WCSUser

Para criar o arquivo JAR para o grupo EJB WCSUser, faça o seguinte:

1. Com a guia EJB selecionada, destaque o grupo EJB **WCSUser**.
2. Clique com o botão direito do mouse no grupo EJB **WCSUser** e selecione **Exportar > JAR EJB 1.1**.
   O SmartGuide Exportar para um Arquivo JAR EJB 1.1 será aberto.
3. No campo **Arquivo JAR**, digite *unidade*:\WebSphere\CommerceServerDev\mytemp_c\CustomizedWCSUserDeployed_DT.jar
4. Selecione os atributos da seguinte forma:

Atributo	Valor
classe	Verificado
java	Não verificado
recurso	Verificado

Atributo	Valor
Banco de dados de destino	DB2 Se estiver implementando um banco de dados DB2, selecione **DB2 para NT, V7.1**. Oracle Se estiver implementando para um banco de dados Oracle, selecione **Oracle, V8**.
Incluir atributos de depuração no arquivo .class	Verificado

Aceite os valores padrão para outros atributos.

5. Clique em **Concluir**.

O arquivo JAR é criado.

O arquivo JAR foi denominado com o sufixo "_DT" como lembrete de que será necessário executar este arquivo JAR através da Ferramenta de Implementação de EJB fornecida pelo WebSphere Application Server antes de implementá-la em seu aplicativo WebSphere Commerce.

Criando o Arquivo `wcsejsclient.jar`

Para criar o arquivo JAR cliente, faça o seguinte:

1. Com a guia EJB selecionada, destaque todos os grupos EJB do WebSphere Commerce (o nome começa com `WCS`). Com todos esses grupos selecionados, clique com o botão direito do mouse e selecione **Exportar > JAR Cliente**.
   O SmartGuide de Exportação será aberto.
2. No campo **arquivo JAR**, digite
   unidade`:\WebSphere\CommerceServerDev\mytemp_c\wcsejsclient.jar`
3. Selecione os atributos da seguinte forma:

Atributo	Valor
beans	Verificado
classe	Verificado
java	Não verificado
recurso	Verificado
Incluir atributos de depuração no arquivo .class	Verificado

Aceite os valores padrão para outros atributos.

4. Clique em **Concluir**.

O arquivo JAR é criado.

Copiando o Modelo JSP Atualizado para o Diretório de Armazenamento de Destino

Em “Incluindo o Novo Preço de Bônus no Modelo de Exibição do Produto” na página 325, você atualizou o modelo de exibição para refletir o preço recém criado. Nesta etapa você copia o modelo JSP da estrutura de diretórios do WebSphere Test Environment para o diretório utilizado pela loja quando estiver em execução fora do WebSphere Test Environment.

1. Na máquina de desenvolvimento, navegue até o seguinte diretório:
   vaj_drive:\VAJava\Ide\project_resources\IBM WebSphere Test Environment \hosts\default_host\default_app\web*store_directory*
   em que *vaj_drive* é a unidade na qual você instalou o VisualAge for Java e *store_directory* é o nome do diretório da loja de exemplo.
   Copie o arquivo ProductDisplay.jsp.
2. Cole o arquivo ProductDisplay.jsp no seguinte diretório:

   ```
   unidade:\WebSphere\AppServer\installedApps\
      WC_Enterprise_App_instance_name.ear\
      wcstores.war\store_directory
   ```

   em que *unidade* é a unidade na qual o WebSphere Commerce está instalado,*store_directory* é o nome do diretório da loja, e *instance_name* é o nome da sua instância do WebSphere Commerce.

Copiar os Arquivos JAR para o WebSphere Commerce Server de Destino

É necessário copiar os arquivos JAR da máquina de desenvolvimento para o diretório apropriado no WebSphere Commerce Server de destino. Para copiar estes arquivos, faça o seguinte:

1. Na máquina de desenvolvimento, navegue para o seguinte diretório:
   unidade:\WebSphere\CommerceServerDev\mytemp_c
   e localize os seguintes arquivos:
   - wcssamplesc.jar
   - CustomizedWCSUserDeployed_DT.jar
   - wcsejsclient.jar

   em que *unidade* é o nome da unidade em que você instalou o WebSphere Commerce Studio, Business Developer Edition.

   Cada um dos arquivos anteriores devem ser copiados em um diretório específico no WebSphere Commerce Server de destino. Leia as seguintes etapas com cuidado para assegurar que cada arquivo seja armazenado no local correto.
2. Copie o arquivo wcssamplesc.jar no seguinte diretório do WebSphere Commerce Server de destino:

   ```
   unidade:\WebSphere\AppServer\installedApps\
      WC_Enterprise_App_instance_name.ear\
      wcstores.war\WEB-INF\lib
   ```

em que *unidade* é a unidade onde o WebSphere Commerce Business Edition foi instalado e *instance_name* é o nome da sua instância (por exemplo, demo).

3. Copie o arquivo wcsejsclient.jar no seguinte diretório do WebSphere Commerce Server de destino:

   unidade:\WebSphere\CommerceServer\temp\lib

4. Copie o arquivo CustomizedWCSUserDeployed_DT.jar no seguinte diretório do WebSphere Commerce Server de destino:

   unidade:\WebSphere\CommerceServer\temp

Executando a Ferramenta de Implementação do EJB

Será necessário executar uma ferramenta de implementação EJB no arquivo JAR contendo o novo grupo EJB. Esta ferramenta está incluída com o WebSphere Application Server.

Para executar esta ferramenta, faça o seguinte:

1. No prompt de comandos, navegue para o seguinte diretório:

   unidade:\WebSphere\CommerceServer\temp

2. Inclua temporariamente a ferramenta no caminho do sistema digitando o seguinte comando:

   PATH=*unidade*:\WebSphere\AppServer\deploytool;%PATH%

3. Digite o comando ejbdeploy da seguinte maneira:

   ejbdeploy *EJBGroupJARFile WorkingDir OutputJARFile* -nowarn -keep -35 -cp *ClassPathOfDepJARFiles*

   em que:

   - *EJBGroupJARFile* é o nome do arquivo JAR para seu grupo EJB. Neste caso o arquivo é CustomizedWCSUserDeployed_DT.jar.
   - *WorkingDir* é o diretório de trabalho.
   - *OutputJARFile* é o nome do arquivo JAR de saída. Neste caso, digite CustomizedWCSUserDeployed.jar.
   - -nowarn é um parâmetro opcional para anular mensagens de aviso e de informações.
   - -keep é um parâmetro opcional para manter o diretório de trabalho depois que o comando ejbdeploy foi executado.
   - -35 é um parâmetro obrigatório que utiliza as mesmas regras de mapeamento top-down para beans de entidade CMP utilizados na Ferramenta de Implementação de EJB fornecida com o WebSphere Application Server, Versão 3.5.
   - -cp *ClassPathOfDepJARFiles* é o caminho de classe de quaisquer arquivos JAR dependentes. Quando tiver modificado um bean corporativo existente do WebSphere Commerce, será necessário incluir

os arquivos wcsejsclient.jar, wcsejbimpl.jar e xml4j.jar no caminho de classe de arquivos JAR dependentes.Então, digite o seguinte:

```
"unidade:\WebSphere\CommerceServer\temp\lib\wcsejsclient.jar;
unidade:\WebSphere\AppServer\InstalledApps\
WC_Enterprise_App_instanceName.ear\lib\wcsejbimpl.jar;
unidade:\WebSphere\AppServer\InstalledApps\
WC_Enterprise_App_instanceName.ear\lib\xml4j.jar;"
```

Modificar o Nível de Isolamento da Transação para os Beans de Entidade

Nesta etapa, você utilizará o comando modifyIsolationLevel para modificar o nível de isolamento de transação dos beans de entidade para o nível requerido por seu tipo específico de banco de dados.

Para executar o comando modifyIsolationLevel, faça o seguinte:

1. No WebSphere Commerce Server de destino, utilize um prompt de comandos para navegar para o seguinte diretório:

   ```
   unidade:\WebSphere\CommerceServer\bin
   ```

2. Será necessário emitir o comando modifyIsolationLevel, que possui a seguinte sintaxe geral:

   ```
   modifyIsolationLevel -jarFile jar_file_name.jar
      -logFile log_file_name -dbType db_type
   ```

   em que

   - *jar_file_name.jar* é o nome do arquivo JAR que contém o código personalizado
   - *log_file_name* é o nome de arquivo completo no qual as informações devem ser registradas
   - *db_type* é o tipo de banco de dados que está sendo utilizado. Digite DB2 ou ORACLE

   O exemplo a seguir é um comando modifyIsolationLevel com todos os valores especificados:

   ```
   modifyIsolationLevel -jarFile
      D:\WebSphere\CommerceServer\temp\CustomizedWCSUserDeployed.jar
      -logFile D:\WebSphere\CommerceServer\instances\demo\logs\output.log
      -dbType DB2
   ```

Nota: Os nomes dos parâmetros fazem distinção entre maiúsculas e minúsculas. Ou seja, jar**F**ile não é o mesmo que jar**f**ile. Verifique se os nomes dos parâmetros estão digitados corretamente.

O comando terá sido executado com êxito se nenhuma exceção for exibida na janela de comando. Depois da conclusão, observe que a marca de hora em seu arquivo JAR foi alterada.

Atualizando o Banco de Dados de Destino

Se você estiver implementando em um WebSphere Commerce Server de destino que utiliza um banco de dados diferente daquele utilizado pelo WebSphere Test Environment, será necessário atualizar o banco de dados de destino da seguinte maneira:

- Se você tiver concluído o Capítulo 9, "Tutorial: Criando Nova Lógica de Negócios" na página 207, deverá atualizar a tabela para incluir uma linha para cada usuário na tabela USERS.
- Se você não concluiu o Capítulo 9, "Tutorial: Criando Nova Lógica de Negócios" na página 207, é necessário criar e preencher a tabela.

Se você não concluiu o Capítulo 9, "Tutorial: Criando Nova Lógica de Negócios" na página 207, é necessário criar e preencher a tabela. Instruções para cada um desses cenários são fornecidas.

DB2 Se estiver utilizando um banco de dados DB2 e precisar *atualizar* a tabela BONUS, faça o seguinte:

1. Abra o Centro de Comandos do DB2 (**Iniciar > Programas > IBM DB2 > Centro de Comandos**) e clique na guia **Script**.
2. No campo **Script**, digite

   ```
   connect to your_database_name
   ```

   em que *your_database_name* é o nome do seu banco de dados e clique no ícone Executar.
3. No campo **Script**, digite o seguinte e clique no ícone Executar:

   ```
   INSERT INTO BONUS
   (SELECT USERS_ID, 0
   FROM USERS
   WHERE USERS_ID NOT IN (SELECT MEMBERID FROM BONUS))
   ```

   A tabela BONUS foi atualizada.

DB2 Se estiver utilizando um banco de dados DB2 e precisar *criar e preencher* a tabela BONUS, faça o seguinte:

1. Abra o Centro de Comandos do DB2 (**Iniciar > Programas > IBM DB2 > Centro de Comandos**) e clique na guia **Script**.
2. No campo **Script**, digite

   ```
   connect to your_database_name
   ```

   em que *your_database_name* é o nome do seu banco de dados e clique no ícone Executar.
3. No campo **Script**, digite o seguinte e clique no ícone Executar:

```
CREATE TABLE BONUS (MEMBERID BIGINT NOT NULL,
    BONUSPOINT INTEGER NOT NULL,
    constraint p_memberid primary key (MEMBERID),
    constraint f_memberid foreign key (MEMBERID)
    references users (users_id) on delete cascade)
```

A tabela BONUS está criada.

4. Para ocupar a tabela, digite o seguinte no campo **Script** e clique no ícone Executar:

   ```
   insert into BONUS (select USERS_ID, 0 from USERS)
   ```

5. Feche o Centro de Comandos do DB2.

Oracle Se você estiver utilizando um banco de dados Oracle e precisar *atualizar* a tabela BONUS, faça o seguinte:

1. Abra a janela de comando Oracle SQL Plus (**Iniciar > Programas > Oracle > Desenvolvimento de Aplicativos > SQL Plus**).
2. No campo **Nome do Usuário**, digite seu nome de usuário do Oracle.
3. No campo **Senha**, digite a senha do Oracle.
4. No campo **Cadeia do Host**, digite sua cadeia de conexão.
5. Atualize a tabela BONUS, digitando as seguintes informações na janela SQL Plus:

   ```
   INSERT INTO BONUS
   (SELECT USERS_ID, 0
   FROM USERS
   WHERE USERS_ID NOT IN (SELECT MEMBERID FROM BONUS));
   ```

   Pressione Enter para executar a instrução SQL.
   A tabela BONUS agora foi atualizada.

6. Digite o seguinte para consolidar as alterações no banco de dados:

   ```
   commit;
   ```

   e pressione Enter para executar a instrução SQL.

Oracle Se você estiver utilizando um banco de dados Oracle e precisar *criar e preencher* a tabela BONUS, faça o seguinte:

1. Abra a janela de comando Oracle SQL Plus (**Iniciar > Programas > Oracle > Desenvolvimento de Aplicativos > SQL Plus**).
2. No campo **Nome do Usuário**, digite seu nome de usuário do Oracle.
3. No campo **Senha**, digite a senha do Oracle.
4. No campo **Cadeia do Host**, digite sua cadeia de conexão.
5. Crie a tabela BONUS, digitando as seguintes informações na janela SQL Plus:

```
CREATE TABLE Bonus (MEMBERID NUMBER NOT NULL,
    BONUSPOINT INTEGER NOT NULL,
    constraint p_memberid primary key (MEMBERID),
    constraint f_memberid foreign key (MEMBERID)
    references users (users_id) on delete cascade);
```

Pressione Enter para executar a instrução SQL.
A tabela BONUS agora foi criada.

6. Preencha a tabela BONUS, digitando o seguinte:
   ```
   insert into BONUS (select USERS_ID, 0 from USERS);
   ```
7. Digite o seguinte para consolidar as alterações no banco de dados:
   ```
   commit;
   ```

   e pressione Enter para executar a instrução SQL.

Exportando o Aplicativo Corporativo Atual do WebSphere Application Server

Nesta etapa, você exportará o aplicativo corporativo atual do WebSphere Application Server para que seja possível abri-lo posteriormente na Ferramenta de Montagem de Aplicativos.

Para exportar o aplicativo corporativo atual do WebSphere Application Server, faça o seguinte:

1. Abra o WebSphere Application Server Administration Console.
2. Expanda o **Domínio Administrativo do WebSphere**.
3. Expanda o **Aplicativo Corporativo**.
4. Clique com o botão direito do mouse em seu aplicativo WebSphere Commerce. Por exemplo, expanda o aplicativo **demo** e selecione **Exportar Aplicativo**.
5. No campo **Diretório de exportação**, digite *unidade*`:\WebSphere\CommerceServer\ working`.
   Isto irá exportar todo o aplicativo, incluindo todos os recursos, para o arquivo `WC_Enterprise_App_`*instanceName*`.ear` (em que *instanceName* é o nome da sua instância do WebSphere Commerce).

   Nota: Se já existir um arquivo `WC_Enterprise_App_`*instanceName*`.ear`, você poderá renomear o arquivo antigo ou substitui-lo.

Exportando Informações sobre Configuração XML para o Aplicativo Corporativo

Também será necessário exportar as informações sobre configuração XML para o aplicativo corporativo. Para exportar estas informações, você irá utilizar o utilitário de linha de comandos `XMLConfig` fornecido pelo WebSphere Application Server.

Para exportar estas informações de configuração, faça o seguinte:

1. No prompt de comandos, navegue para o seguinte diretório:
   unidade:\WebSphere\CommerceServer\working
2. Chame a ferramenta XMLConfig para executar uma exportação parcial digitando o seguinte comando:
   ```
   xmlConfig -export OutputFileB.xml -partial was.export.app.xml
      -adminNodeName wasHostName
   ```
   em que *wasHostName* é o nome do nó no WebSphere Application Server que contém o aplicativo corporativo atual. Além disso, OutputFileB.xml é o nome do arquivo criado como resultado da execução deste comando.

Depois de exportar as informações sobre os beans corporativos do aplicativo corporativo atual, atualize o arquivo OutputFileB.xml para que indique o arquivo JAR que contém o código do bean User modificado.

Para atualizar o arquivo OutputFileB.xml, proceda da seguinte forma:

1. Navegue para o seguinte diretório:
   unidade:\WebSphere\CommerceServer\working
2. Abra o arquivo OutputFileB.xml em um editor de texto.
3. Localize a tag <ear-file-name> e substitua o valor pelo seguinte:
   ```
   unidade:\WebSphere\CommerceServer\working\
      WC_Enterprise_App_instanceName.ear
   ```
4. Localize a sub-rotina <ejb-module> para o grupo do EJB WCSUser e altere o valor contido nas tags <jar-file> para CustomizedWCSUserDeployed.jar
5. Salve o arquivo OutputFileB.xml.

Montando o grupo EJB modificado no aplicativo corporativo

Nesta etapa, você irá abrir seu aplicativo corporativo na ferramenta de montagem de aplicativo. Assim que ele estiver aberto nessa ferramenta, você poderá fazer o seguinte para incluir o bean de Usuário modificado no aplicativo corporativo:

1. Faça uma cópia do caminho de classe para a versão existente do grupo EJB WCSUser.
2. Remova a versão existente do grupo WCSUser EJB.
3. Importe a nova versão do grupo EJB WCSUser. O arquivo JAR para o novo grupo EJB será armazenado dentro da seção de Módulo EJB do aplicativo corporativo.
4. Defina o caminho de classe para o grupo EJB WCSUser.

Para montar o novo grupo EJB no aplicativo corporativo, faça o seguinte:

1. Faça um backup do aplicativo corporativo atual, da seguinte maneira:
   a. No prompt de comandos, navegue para o seguinte diretório:

unidade:\WebSphere\CommerceServer\working

b. Digite o seguinte comando:

```
copy WC_Enterprise_App_instanceName.ear
   WC_Enterprise_App_instanceName.ear.bak2
```

2. Abra o WebSphere Application Server Administration Console.
3. No menu **Ferramentas**, selecione **Ferramenta de Montagem de Aplicativo**. (Se for aberta uma janela Bem-vindo, selecione **Cancelar** para acessar o console).
4. Abra o aplicativo corporativo com o qual irá trabalhar, da seguinte forma:
   a. A partir do menu **Arquivo**, selecione **Abrir**.
   b. No campo **Nome do arquivo**, digite:

      ```
      unidade:\WebSphere\CommerceServer\working\
         WC_Enterprise_App_instanceName.ear
      ```

      e clique em **Abrir**. Aguarde até que o aplicativo seja aberto antes de prosseguir para as próximas etapas. Isto pode levar alguns minutos.
5. Clique em **Módulos EJB**. O painel à direita exibe os módulos EJB no aplicativo corporativo.
6. Clique no módulo EJB **WCSUser**.
7. Clique na guia Geral para exibir as informações de caminho de classe para o módulo EJB WCSUser existente. Copie estas informações de caminho de classe existentes em um arquivo texto (por exemplo, `WCSUser_path.txt`).
8. Clique com o botão direito do mouse no módulo EJB **WCSUser** e selecione **Excluir**.
9. Clique com o botão direito do mouse nos **Módulos EJB** e selecione **Importar**.
10. No campo **Nome do arquivo**, digite

    ```
    unidade:\WebSphere\CommerceServer\temp\CustomizedWCSUserDeployed.jar
    ```

    e clique em **Abrir**. Na janela Confirmar Valores, clique em **OK**.
11. Assim que o arquivo `CustomizedWCSUserDeployed.jar` for importado, vá para o grupo EJB **WCSUser** e selecione este grupo.
    As informações sobre este grupo serão exibidas no painel à direita.
12. Abra o arquivo texto contendo as informações de caminho de classe para a versão anterior do grupo EJB WCSUser. Selecione e copie o caminho de classe.
13. No campo do caminho de classe para o novo grupo EJB WCSUser, cole estas informações no caminho de classe.
14. Clique em **Aplicar**.
15. No menu **Arquivo**, selecione **Fechar**.

16. Aguarde o fechamento do arquivo e em seguida, no menu **Arquivo**, selecione **Abrir** e reabra o arquivo *unidade*:\WebSphere\CommerceServer\working\WC_Enterprise_App_*instanceName*.ear.
17. Configure a segurança para o bean de Usuário, fazendo o seguinte:
    a. Com o nó Módulos EJB expandido, localize e expanda o nó **WCSUser**.
    b. Expanda **Beans de Entidade**.
    c. Expanda **Usuário**
    d. Clique em **Extensões do Método** e em seguida, no painel à direita faça o seguinte:
        1) Clique na guia **Avançado**.
        2) Assegure-se de que **Identidade de segurança** esteja selecionado.
        3) Para cada método, assegure-se de que **Utilizar identidade do servidor EJB** esteja selecionado.
        4) Clique em **Aplicar** (se tiver feito alguma modificação).
    e. No painel de navegação à esquerda, clique com o botão direito do mouse em **Funções de Segurança** no grupo de EJB WCSUser e selecione **Novo**; depois faça o seguinte:
        1) No campo **Nome**, digite WCSecurityRole e clique em **Aplicar**. Observe, se esta função já existir, não será necessário executar esta etapa.
    f. No painel de navegação esquerdo, clique com o botão direito do mouse em **Permissões do Método** no grupo EJB WCSUser, selecione **Novo** e proceda da seguinte forma:
        1) No campo **Nome de Permissão do Método**, digite WCMethodPermission
        2) Na área de seleção **Métodos**, clique em **Incluir**. A janela Incluir Métodos é aberta.
        3) Expanda **CustomizedWCSUserDeployed.jar** e selecione todos os beans corporativos (mantenha a tecla Shift pressionada durante a seleção). Clique em **OK**. Todos os beans corporativos serão exibidos na coluna Bean corporativo e **Todos os métodos** será exibido na coluna Tipos.
        4) Na área de seleção Funções, clique em **Incluir**, selecione a WCSecurityRole e clique em **OK**.
        5) Clique em **Aplicar** e em **OK**.
18. A partir do menu **Arquivo**, selecione **Salvar**.
19. Feche Application Assembler Tool.

Após a conclusão desta etapa, você terá criado um novo aplicativo corporativo que contém todas as lógicas anteriores, bem como sua nova lógica de negócios. Isto está contido no arquivo recém-modificado `WC_Enterprise_App_instanceName.ear`.

Importando o Novo Aplicativo Corporativo no WebSphere Application Server

Os itens a seguir são etapas de alto nível envolvidas na importação de um novo aplicativo corporativo no WebSphere Application Server:

1. Parando o aplicativo corporativo que está atualmente em execução no WebSphere Application Server e em seguida, removendo-o. Essas etapas são executadas no WebSphere Application Server Administration Console.
2. Importando o novo aplicativo, utilizando o utilitário de linha de comandos XMLConfig.
3. Atualizando o Console do Administrador do WebSphere Application Server e em seguida, inicializando o novo aplicativo corporativo.

Cada uma das etapas é descrita com maiores detalhes nas seções a seguir.

Parando e Removendo o Aplicativo Corporativo Atual: Para parar e remover seu aplicativo corporativo atual do WebSphere Application Server, faça o seguinte:

1. Abra o Administration Console do WebSphere Application Server.
2. Expanda **Domínio Administrativo do WebSphere**.
3. Expanda os **Nós**.
4. Expanda *nodeName* (em que *nodeName* é o nome do nó).
5. Expanda **Servidores de Aplicativos**.
6. Clique com o botão direito do mouse em seu aplicativo WebSphere Commerce. Por exemplo, clique com o botão direito do mouse no **WebSphere Commerce Server -** *instanceName* e selecione **Parar**.
7. Expanda **Aplicativos Corporativos**.
8. Clique com o botão direito do mouse em seu aplicativo WebSphere Commerce. Por exemplo, clique com o botão direito do mouse no aplicativo **WebSphere Commerce Enterprise Server - demo** e selecione **Parar**.
9. Clique com o botão direito do mouse em seu aplicativo WebSphere Commerce. Por exemplo, clique com o botão direito do mouse no aplicativo **WebSphere Commerce Enterprise Server - demo** e selecione **Remover**.
10. Quando solicitado para que o aplicativo indicado seja exportado, selecione **Não**.

Importando o Novo Aplicativo Corporativo Utilizando XMLConfig: Para importar o novo aplicativo corporativo utilizando o utilitário de linha de comandos XMLConfig, proceda da seguinte forma:

1. Navegue para o seguinte diretório:

   unidade:\WebSphere\CommerceServer\working

2. No prompt de comandos, digite o seguinte comando para importar o aplicativo corporativo no WebSphere Application Server:

   ```
   xmlConfig -import OutputFileB.xml -adminNodeName was_hostname
   ```

   em que *was_hostname* é o nome do nó do WebSphere Application Server que contém o aplicativo atual.

Nota: 400 Se estivesse implementando em uma instância do WebSphere Commerce em execução no iSeries, seria necessário executar uma etapa adicional para modificar as permissões de diretório depois de importar o aplicativo. Consulte "Importando um Aplicativo Corporativo" na página 382 para obter detalhes sobre como modificar essas permissões.

Iniciando o Novo Aplicativo Corporativo: Após ter importado um novo aplicativo corporativo utilizando o utilitário de linha de comandos XMLConfig, você poderá utilizar o Console do Administrador do WebSphere Application Server para executar uma atualização e em seguida, iniciar o novo aplicativo.

Para atualizar o console e iniciar o novo aplicativo, faça o seguinte:

1. Abra o Administration Console do WebSphere Application Server.
2. Expanda **Domínio Administrativo do WebSphere**
3. Destaque **Nós**.
4. Clique no ícone **Atualizar subárvore selecionada**.
5. Inicie o aplicativo WebSphere Commerce da seguinte forma:
   - Expanda **Servidores de Aplicativos**.
   - Clique com o botão direito do mouse em seu aplicativo WebSphere Commerce. Por exemplo, clique com o botão direito do mouse em **WebSphere Commerce Server -** *instanceName* e selecione **Iniciar**.

Testando o Novo Código na Loja de Destino

Para testar o bean de entidade modificado e o novo comando de tarefa na loja sendo executada no WebSphere Application Server, faça o seguinte:

1. Abra um navegador e digite a seguinte URL:

   ```
   http://hostname/webapp/wcs/stores/servlet/StoreCatalogDisplay?
        storeId=store_Id&catalogId=catalog_Id&langId=-1
   ```

em que *store_Id* é o identificador de sua loja e *catalog_Id* é o identificador do catálogo de sua loja.

Nota: Se ocorrer o Erro 500, pode ser necessário reiniciar o WebSphere Application Server para atualizar o aplicativo corporativo.

2. Clique no link **Registrar** sob o título Serviços e em seguida, clique em **Registrar** sob o título Novo Cliente.
Registre um novo cliente utilizando o endereço de e-mail `wctester@wc` e a senha `wctester1`. Preencha outros campos com valores de teste e clique em **Enviar**. Deixe o navegador aberto.

3. DB2 Abra o Centro de Comandos do DB2 e faça o seguinte:
   a. Clique na guia **Interativo**
   b. No campo **Comando**, proceda da seguinte maneira:
      1) Digite
      ```
      connect to your_database_name
      ```
      em que *your_database_name* é o nome do seu banco de dados do WebSphere Commerce e clique no ícone Executar.
      2) Digite `select users_id from USERREG where LOGONID = 'wctester@wc'` e clique no ícone Executar.
   c. A guia Resultados da Consulta exibe a entrada para o cliente que você registrou na etapa 2. Anote o valor de USERS_ID do cliente aqui:
      _____
   d. Atualize o saldo de pontos de bônus do cliente recém registro. Clique na guia Interativo e no campo **Comando**, digite o seguinte:
      ```
      update BONUS set BONUSPOINT = 1000 where MEMBERID = users_id
      ```
      em que *users_id* é o valor da etapa 3c. Clique no ícone Executar.

4. Oracle Atualize o saldo de pontos de bônus do usuário de teste, fazendo o seguinte:
   a. Abra a janela de comando Oracle SQL Plus (**Iniciar > Programas > Oracle > Desenvolvimento de Aplicativos > SQL Plus**).
   b. No campo **Nome do Usuário**, digite seu nome de usuário do Oracle.
   c. No campo **Senha**, digite a senha do Oracle.
   d. No campo **Cadeia do Host**, digite sua cadeia de conexão.
   e. Digite `select users_id from USERREG where LOGONID = 'wctester@wc';` para determinar o ID do usuário.
   f. A entrada do cliente registrado na etapa 2 será exibida. Anote o valor de USERS_ID do cliente aqui: _____
   g. Atualize o saldo de pontos de bônus do cliente recentemente registrado, digitando o seguinte:

```
update BONUS set BONUSPOINT = 1000 where MEMBERID = users_id;
```

em que *users_id* é o valor da etapa 4f.

h. Digite o seguinte para consolidar as alterações no banco de dados:

```
commit;
```

e pressione Enter para executar a instrução SQL.

5. No navegador, clique no link **Moda Masculina** para exibir a seção de moda masculina da loja.
6. Clique no link do produto destacado para exibir a página do produto. A página exibe o preço normal e o preço com desconto com base no saldo de pontos de bônus do cliente.

Parte 5. Apêndices

Apêndice A. Iniciando e Parando o WebSphere Test Environment

Essa seção descreve as etapas envolvidas para iniciar o WebSphere Test Environment dentro do VisualAge for Java. Em geral, a inicialização desse ambiente requer a execução das seguintes etapas:

1. Iniciar o servidor de nomes persistente.
2. Iniciar o servidor EJB.
3. Iniciar o mecanismo de Servlet.

Cada uma dessas etapas está descrita nas seções subseqüentes.

Para parar o ambiente de teste, inverta a ordem das etapas.

Nota: Para utilizar todos os recursos do WebSphere Test Environment, você deve garantir que o WebSphere Application Server não esteja em execução, antes de iniciar o WebSphere Test Environment. Para obter informações sobre como parar o WebSphere Application Server, consulte o *WebSphere Commerce, Manual de Instalação*.

Iniciando e Parando o Servidor de Nomes Persistente

O servidor de nomes persistente recebe os pedidos de procura de clientes do beans corporativos. Todos os beans corporativos são registrados com o servidor de nomes persistente.

Para iniciar ou parar o servidor de nomes persistente, proceda da seguinte forma:

1. Do menu **Área de Trabalho** em VisualAge for Java, selecione **Ferramentas > WebSphere Test Environment**.
   O WebSphere Test Environment Control Center é aberto.
2. Clique em **Servidor de Nomes Persistente** e em seguida, clique em uma das seguintes opções:
   - **Iniciar Servidor de Nomes**.
     Aguarde pela mensagem “Servidor aberto para negócios” na janela Console. A inicialização desse servidor pode levar vários minutos.
   - **Parar o Servidor de Nomes**.

Iniciando e Parando o Servidor EJB

O servidor EJB é um processo ou aplicativo de alto nível que fornece um ambiente de runtime para suportar a execução dos aplicativos do servidor que utilizam os beans corporativos.

Para iniciar ou parar o servidor ELB, proceda da seguinte forma:

1. Abra a janela de configuração do servidor EJB (clique na guia EJB e em seguida, do menu **EJB**, selecione **Abrir em > Configuração do Servidor**).
2. Clique com o botão direito do mouse no servidor EJB que você deseja iniciar ou encerrar (por exemplo, **Servidor EJB {Server 1}**) e proceda de uma das seguintes formas:
   - Selecione **Iniciar Servidor**
     Aguarde pela mensagem "Servidor aberto para negócios" na janela Console (pode ser necessário selecionar **Servidor EJB** no Console para exibir o status desse servidor). A inicialização pode levar 10 ou 15 minutos, dependendo do computador.
   - Selecione **Parar o Servidor**

O Console é o dispositivo de entrada e saída padrão para programas Java no IDE. Para ver a saída de um determinado programa, selecione primeiro o programa na área de janela Todos os Programas e em seguida, sua saída será exibida na área de janela Saída.

Iniciando e Parando o Mecanismo de Servlet

O mecanismo de servlet integra o servidor Web e a funcionalidade do WebSphere Application Server para criar um ambiente, com a finalidade de teste.

Para iniciar ou parar o mecanismo de servlet, proceda da seguinte forma:

1. Do menu **Área de Trabalho** em VisualAge for Java, selecione **Ferramentas > WebSphere Test Environment**.
   O WebSphere Test Environment Control Center é aberto.
2. Clique em **Mecanismo de Servlet** e em seguida, em um dos seguintes:
   - **Iniciar Mecanismo de Servlet**.
     Quando o Mecanismo de Servlet é iniciado, o console exibe a mensagem `***Mecanismo de Servlet iniciado***`.
   - **Parar o Mecanismo de Servlet**.

Apêndice B. Detalhes de Implementação

Depois de você ter criado código personalizado no VisualAge for Java e tê-lo testado no WebSphere Test Environment, você deverá implementá-lo em uma instância do WebSphere Commerce em execução fora do WebSphere Test Environment. Esta instância do WebSphere Commerce pode ser executada localmente em sua máquina de desenvolvimento, ou uma outra máquina (utilizando o mesmo sistema operacional ou um diferente).

Este apêndice descreve as etapas necessárias para implementação de código personalizado em uma instância do WebSphere Commerce em execução fora do WebSphere Test Environment. Você deve consultar "Implementação de Código" na página 193 para conhecer as etapas de alto nível do processo de implementação e em seguida, consultar este apêndice para obter os detalhes.

Mapeando para o Sistema de Arquivos Integrado (iSeries)

400 Esta seção é válida somente se o WebSphere Commerce Server de destino estiver executando na plataforma iSeries.

Se o WebSphere Commerce Server de destino estiver executando na plataforma iSeries, mapeie uma unidade local na máquina de desenvolvimento para o IFS (Integrated File System) no servidor iSeries. Nas próximas seções deste documento, *iSeries_drive* será utilizado para fazer referência a essa unidade local mapeada para o IFS. Além disso, *unidade* será utilizado para uma unidade local na máquina de desenvolvimento (que não tenha sido mapeada para o IFS).

Arquivos JAR para Comandos Personalizados e Beans de Dados

O código de comando personalizado e de bean de dados deve ser armazenado em um projeto separado do código do WebSphere Commerce. Quando tiver concluído o teste com o WebSphere Test Environment, você deverá criar um arquivo JAR para o projeto contendo o comando personalizado e o código de bean de dados e em seguida, colocar o arquivo JAR no diretório apropriado no WebSphere Commerce Server de destino.

Para criar um arquivo JAR para comandos personalizados e beans de dados, proceda da seguinte maneira:

1. No Workbench no VisualAge for Java, selecione a guia **Projeto**.

2. Clique com o botão direito do mouse no projeto que contém o comando personalizado e código de beans de dados e selecione **Exportar**.
   O SmartGuide Exportar é aberto.
3. Selecione **Arquivo Jar** e clique em **Avançar.**
4. No campo **Arquivo Jar**, digite o seguinte:
   unidade:\WebSphere\CommerceServerDev*temp_directory*\
   *jar_file_name*_1.jar
   em que *unidade*:\WebSphere\CommerceServerDev*temp_directory*\ é um diretório temporário com espaço livre suficiente para o arquivo JAR e *jar_file_name*_1.jar é o nome do arquivo JAR com _1 anexado.
5. Selecione os atributos para o arquivo JAR, da seguinte maneira:

Atributo	Valor
classe	Verificado
java	Não verificado
recurso	Verificado
beans	Verificado
Incluir atributos de depuração no arquivo .class	Verificado

   Aceite os valores padrão para outros atributos.
6. Clique em **Concluir**.

Como o arquivo JAR de implementação criado não contém informações completas sobre nomeação de pacotes, você deve usar outro utilitário de empacotamento (fora do VisualAge for Java) para empacotar novamente o arquivo JAR. Para empacotar novamente este arquivo, proceda da seguinte maneira:

1. Em uma janela de comando, navegue para o diretório em que você armazenou *jar_file_name*_1.jar nas etapas anteriores.
2. Digite `mkdir temp1`.
3. Digite `cd temp1`.
4. Defina o caminho da seguinte forma:
   `set PATH=%PATH%;`*unidade*`:\WebSphere\WebSphereStudio4\bin;`
   em que *unidade* é a unidade na qual o WebSphere Studio está instalado.
5. Digite `jar xvf ../`*jar_file_name*`_1.jar`.
6. Digite `jar cvf ../`*jar_file_name*`.jar *` (observe que o _1 é removido do nome).
7. Vá para o diretório *unidade*:\WebSphere\CommerceDev*temp_directory*.
8. Se estiver implementando em uma instância local do WebSphere Commerce, copie *jar_file_name*.jar para o seguinte diretório:

unidade:\WebSphere\AppServer\installedApps\
WC_Enterprise_App_*instanceName*.ear\wcstores.war\WEB-INF\lib

:NONE. Caso contrário, deixe o arquivo JAR no diretório temporário até que seja necessário transferir todos os bens ativos do arquivo para o WebSphere Commerce Server de destino.

Criando Arquivos JAR para Novos Beans de Entidade

Quando criar novos beans de entidade, você deverá armazenar o código em um projeto que seja separado de todo o código do WebSphere Commerce. Além disso, ele deverá ser separado de qualquer um dos comandos personalizados e códigos de bean de dados. Um novo bean de entidade deverá ser criado em um grupo de EJB que estiver separado dos grupos de EJB que contêm os beans de entidade públicos do WebSphere Commerce.

Implementar novos beans de entidade envolve criar dois arquivos JAR. O primeiro arquivo JAR é o JAR de Exportação EJB 1.1, criado utilizando ferramentas disponíveis quando a guia EJB está selecionada. O segundo arquivo JAR contém o código de implementação para o bean de entidade e é criado a partir do projeto que contém o código do bean de entidade. Este segundo arquivo JAR é criado com as ferramentas que estão disponíveis quando a guia Projetos está selecionada no Workbench do VisualAge for Java.

Criando o Arquivo JAR de Exportação EJB 1.1

Para criar o arquivo JAR de Exportação EJB 1.1 para novos beans de entidade, proceda da seguinte forma:

1. No Workbench no VisualAge for Java, selecione a guia **Projeto**.
2. Clique com o botão direito do mouse no grupo de EJB que contém seu código de bean de entidade personalizado e selecione **Exportar > JAR EJB 1.1**.
   O SmartGuide Exportar para um Arquivo JAR EJB 1.1 é aberto.
3. No campo **Arquivo Jar**, digite o seguinte:
   unidade:\WebSphere\CommerceServerDev\temp_directory\
   *jarFileName*_DT.jar
   em que *unidade*:\WebSphere\CommerceServerDev\temp_directory é um diretório temporário que tem espaço livre suficiente para seu arquivo JAR e *jarFileName*_DT.jar é o nome do arquivo JAR com o sufixo _DT incluído.

O arquivo JAR foi denominado com o sufixo "_DT" como lembrete de que será necessário executar este arquivo JAR através da Ferramenta de Implementação de EJB fornecida pelo WebSphere Application Server antes de implementá-la em seu aplicativo WebSphere Commerce.

4. Selecione os atributos para o arquivo JAR, da seguinte maneira:

Atributo	Valor
classe	Verificado
java	Não verificado
recurso	Verificado
Banco de dados de destino	DB2 Se você estiver implementando para um banco de dados DB2, selecione uma das seguintes opções: • Windows AIX Solaris Linux **DB2 para NT, V7.1** • 400 **DB2 para AS/400, V4** Oracle Se estiver implementando um banco de dados Oracle, selecione **Oracle, V8**
Incluir atributos de depuração no arquivo .class	Verificado

   Aceite os valores padrão para outros atributos.
5. Clique em **Concluir**.

O arquivo JAR é criado.

Criando o Arquivo JAR de Implementação

Para criar o arquivo JAR de implementação para novos beans de entidade, proceda da seguinte maneira:

1. No Workbench no VisualAge for Java, selecione a guia **Projeto**.
2. Clique com o botão direito do mouse no projeto que contém seu código personalizado de bean de entidade e e selecione **Exportar**.
   O SmartGuide Exportar é aberto.
3. Selecione **Arquivo Jar** e clique em **Avançar.**
4. No campo **Arquivo Jar**, digite o seguinte:
   unidade:\WebSphere\CommerceServerDev*temp_directory**jarFileName*_1.jar
   em que *unidade*:\WebSphere\CommerceServerDev*temp_directory* é um diretório temporário que tem espaço livre suficiente para seu arquivo JAR e *jarFileName*_1.jar é o nome do arquivo JAR com _1 anexado.
5. Selecione os atributos para o arquivo JAR, da seguinte maneira:

Atributo	Valor
classe	Verificado

Atributo	Valor
java	Não verificado
recurso	Verificado
beans	Verificado
Incluir atributos de depuração no arquivo .class	Verificado

Aceite os valores padrão para outros atributos.

6. Clique em **Concluir**.

Como o arquivo JAR de implementação criado não contém informações completas sobre nomeação de pacotes, você deve usar outro utilitário de empacotamento (fora do VisualAge for Java) para empacotar novamente o arquivo JAR. Para empacotar novamente este arquivo, proceda da seguinte maneira:

1. Em uma janela de comando, navegue para o diretório em que você armazenou *jarFileName*_1.jar nas etapas anteriores.
2. Digite `mkdir temp1`.
3. Digite `cd temp1`.
4. Defina o caminho da seguinte forma:
   `set PATH=%PATH%;`*`unidade`*`:\WebSphere\WebSphereStudio4\bin;`
   em que *unidade* é a unidade na qual o WebSphere Studio está instalado.
5. Digite `jar xvf ../`*`jarFileName`*`_1.jar`.
6. Digite `jar cvf ../`*`jarFileName`*`.jar *` (observe que o _1 é removido do nome).
7. Vá para o diretório
   unidade`:\WebSphere\CommerceServerDev\`*`temp_directory`*.
8. Se estiver implementando em uma instância local do WebSphere Commerce, copie *jarFileName*_1.jar para o seguinte diretório:
   unidade`:\WebSphere\CommerceServer\temp\lib`
   Caso contrário, deixe o arquivo JAR no diretório temporário até que seja necessário transferir todos os bens ativos do arquivo para o WebSphere Commerce Server de destino.

Criando Arquivos JAR para Beans de Entidade do WebSphere Commerce Personalizados

Você pode estender qualquer um dos beans de entidade públicos do WebSphere Commerce. Eles são encontrados nos seguintes grupos de EJB:

- WCSActrlEJBGroup
- WCSApproval

- WCSAuction
- WCSCatalog
- WCSCommon
- WCSContract
- WCSCoupon
- WCSFulfillment
- WCSInventory
- WCSMessageExtensions
- WCSOrder
- WCSOrderManagement
- WCSOrderStatus
- WCSPayment
- WCSPVCDevices
- WCSTaxation
- WCSUserTraffic
- WCSUser
- WCSUTF

Se você estender um dos beans de entidade públicos do WebSphere Commerce, crie um arquivo JAR de Exportação EJB 1.1 para o grupo EJB que contém o bean de entidade público modificado WebSphere Commerce.

Criando o Arquivo JAR de Exportação EJB 1.1

Para criar o arquivo JAR de Exportação EJB 1.1 para o grupo EJB que contém o bean de entidade modificado, proceda da seguinte forma:

1. No Workbench no VisualAge for Java, selecione a guia **Projeto**.
2. Clique com o botão direito do mouse no grupo de EJB que contém seu código de bean de entidade personalizado e selecione **Exportar > JAR EJB 1.1**.
   O SmartGuide Exportar para um Arquivo JAR EJB 1.1 é aberto.
3. No campo **Arquivo jar**, digite o seguinte:
   unidade:\WebSphere\CommerceServerDev*temp_directory*\Cust_*EJBGroupName*-ejb_DT.jar
   em que *unidade*:\WebSphere\CommerceServerDev*temp_directory* é um diretório temporário com espaço livre suficiente para o arquivo JAR e Cust_*EJBGroupName*-ejb_DT.jar é o nome do arquivo JAR com o sufixo _DT incluído.

O arquivo JAR foi denominado com o sufixo "_DT" como lembrete de que será necessário executar este arquivo JAR através da Ferramenta de Implementação de EJB fornecida pelo WebSphere Application Server antes de implementá-la em seu aplicativo WebSphere Commerce.

4. Selecione os atributos para o arquivo JAR, da seguinte maneira:

Atributo	Valor
classe	Verificado
java	Não verificado
recurso	Verificado
Banco de dados de destino	DB2 Se você estiver implementando para um banco de dados DB2, selecione uma das seguintes opções: • Windows AIX Solaris Linux **DB2 para NT, V7.1** • 400 **DB2 para AS/400, V4** Oracle Se estiver implementando um banco de dados Oracle, selecione **Oracle, V8**
Incluir atributos de depuração no arquivo .class	Verificado

Aceite os valores padrão para outros atributos.

5. Clique em **Concluir**.

Criando o Arquivo JAR Cliente

Para criar o arquivo JAR cliente, faça o seguinte:

1. Com a guia EJB selecionada, destaque todos os grupos EJB do WebSphere Commerce (o nome começa com WCS). Com todos esses grupos selecionados, clique com o botão direito do mouse e selecione **Exportar > JAR Cliente**.
   O SmartGuide de Exportação será aberto.
2. No campo **Arquivo JAR**, digite o seguinte:
   unidade:\WebSphere\CommerceServerDev*temp_directory*\wcsejsclient.jar
3. Selecione os atributos da seguinte forma:

Atributo	Valor
beans	Verificado

Atributo	Valor
classe	Verificado
java	Não verificado
recurso	Verificado
Incluir atributos de depuração no arquivo .class	Verificado

Aceite os valores padrão para outros atributos.

4. Clique em **Concluir**.

O arquivo JAR é criado.

Armazenando Recursos no WebSphere Commerce Server de Destino

Recursos relacionados com seu código personalizado de devem ser copiados para o WebSphere Commerce Server de destino. Esses recursos incluem arquivos JAR para comandos personalizados, beans de dados e beans de entidade. Você pode também ter novos modelos de JSP e gráficos que suportam a personalização.

AIX Solaris Linux Você deve executar todas as etapas de implementação no WebSphere Commerce Server de destino utilizando o usuário que foi criado quando as etapas na seção *"Executando o script postinstall"* do *WebSphere Commerce - Manual de Instalação* foram executadas. Por padrão, é o `wasuser`. Além disso, certifique-se de que os recursos do arquivo (por exemplo, arquivos JAR) e diretórios nos quais estes recursos são colocados tenham permissões de leitura, gravação e execução de arquivos concedidas a este usuário.

400 Certifique-se de que as autoridades para os diretórios `/QIBM/UserData/WebCommerce/instances/`*`instanceName`* e `/QIBM/UserData/WebCommerce/instances/`*`instanceName`*`/working` incluam o usuário `QEJB`. Inclua este usuário nos dois diretórios, definindo Autoridade de Dados como `*RWX`.

A tabela a seguir resume os diretórios padrão em que os bens ativos são armazenados em um WebSphere Commerce Server de destino que está executando o Windows NT ou Windows 2000.

Tabela 12.

Tipo de Recurso	Localização do diretório no WebSphere Commerce Server de destino
Arquivos JAR para comandos e lógica de bean de dados	*unidade*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wcstores.war\WEB-INF\lib
JAR de Exportação EJB 1.1 criado utilizando VisualAge for Java	*unidade*:\WebSphere\CommerceServer\temp **Nota:** Em seguida, esse arquivo é transmitido como entrada para a ferramenta EJBDeploy a fim de gerar código implementado que poderá ser utilizado no WebSphere Application Server V4.0.
Arquivo JAR do código EJB do cliente	*unidade*:\WebSphere\CommerceServer\temp\lib **Nota:** Este arquivo é utilizado somente no caminho da classe para a ferramenta EJBDeploy.
Arquivo JAR do código EJB de implementação	*unidade*:\WebSphere\CommerceServer\temp\lib **Nota:** Esse arquivo é incluído em seguida no aplicativo corporativo como um recurso do arquivo.
Modelos JSP	*unidade*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wcstores.war*storeDir*
Imagens	*unidade*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wcstores.war*storeDir*\images

Para armazenar bens ativos no WebSphere Commerce Server de destino, proceda da seguinte maneira:

1. Localize os arquivos JAR de seu comando e código de bean de dados. Eles devem estar em um dos seguintes diretórios na máquina de desenvolvimento:
   - Windows *unidade*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wcstores.war\WEB-INF\lib
   - Windows *unidade*:\WebSphere\CommerceServerDev*temp_directory*
2. Copie os arquivos JAR de seu comando e código de bean de dados para um dos seguintes diretórios no WebSphere Commerce Server de destino:
   - Windows *unidade*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wcstores.war\WEB-INF\lib
   - AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/WEB-INF/lib
   - Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/WEB-INF/lib

- Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/WEB-INF/lib
- 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/WEB-INF/lib

3. Localize os arquivos JAR de Exportação EJB 1.1 para todos os novos grupos EJB e também para os beans de entidade modificados do WebSphere Commerce no seguinte diretório na máquina de desenvolvimento:
   Windows *unidade*:\WebSphere\CommerceServerDev*temp_directory*
4. Copie os arquivos JAR de Exportação EJB 1.1 para um dos seguintes diretórios no WebSphere Commerce Server de destino:
   - Windows *unidade*:\WebSphere\CommerceServer\temp
   - AIX /usr/WebSphere/CommerceServer/temp
   - Solaris /opt/WebSphere/CommerceServer/temp
   - Linux /opt/WebSphere/CommerceServer/temp
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/temp
5. Se você criou novos beans corporativos, localize o arquivo JAR de implementação desses beans no seguinte diretório na máquina de desenvolvimento:
   Windows *unidade*:\WebSphere\CommerceServerDev*temp_directory*
6. Copie o arquivo JAR de implementação de seus novos beans corporativos para o seguinte diretório no WebSphere Commerce Server de destino:
   - Windows *unidade*:\WebSphere\CommerceServer\temp\lib
   - AIX /usr/WebSphere/CommerceServer/temp/lib
   - Solaris /opt/WebSphere/CommerceServer/temp/lib
   - Linux /opt/WebSphere/CommerceServer/temp/lib
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/temp/lib
7. Se você tiver modificado beans de entidade existentes do WebSphere Commerce, localize o arquivo JAR do cliente no seguinte diretório na máquina de implementação:
   Windows *unidade*:\WebSphere\CommerceServerDev*temp_directory*
8. Copie o arquivo JAR cliente para um dos seguintes diretórios no WebSphere Commerce Server de destino:
   - Windows *unidade*:\WebSphere\CommerceServer\temp\lib

- AIX /usr/WebSphere/CommerceServer/temp/lib
- Solaris /opt/WebSphere/CommerceServer/temp/lib
- Linux /opt/WebSphere/CommerceServer/temp/lib
- 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/temp/lib

9. Copie todos os modelos de JSP para um dos seguintes diretórios no WebSphere Commerce Server de destino:
   - Windows *unidade*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wcstores.war*store_directory*
   - AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*
   - Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*
   - Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*
   - 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*

   em que *store_directory* é o diretório da loja.
10. Copie quaisquer imagens para um dos seguintes diretórios no WebSphere Commerce Server de destino:
    - Windows *unidade*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wcstores.war*store_directory*\images
    - AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*/images
    - Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*/images
    - Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*/images
    - 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*/images

em que *store_directory* é o diretório da loja.

Atualizando o Banco de Dados de Destino

Quando o WebSphere Commerce Server de destino utiliza um banco de dados diferente da máquina de desenvolvimento, execute todas as atualizações efetuadas no banco de dados de desenvolvimento no banco de dados utilizado pelo WebSphere Commerce Server de destino. Isto inclui atualizações no registro de comandos novos ou modificados, tabelas adicionais criadas e a criação de políticas de controle de acesso para todos os novos recursos criados.

Para obter informações sobre como carregar políticas de controle de acesso (incluindo sintaxe de comando para diversas plataformas e requisitos de permissão de diretório), consulte o *WebSphere Commerce Access Control Guide*.

400 Você é responsável por possuir um utilitário para executar instruções SQL. Uma forma de fazer isto é utilizar o Client Access Express V5R1 (instalação completa). Para abrir esse utilitário, proceda da seguinte forma:

1. Abra o **Navegador de Operações**.
2. Quando o operations navigator for aberto, é necessário efetuar sign-on em um sistema específico. Certifique-se de selecionar a máquina iSeries de destino e utilizar o perfil do usuário e a senha da instância do WebSphere Commerce. Isto assegura que o perfil do usuário da instância do WebSphere Commerce possui todas as novas tabelas que foram criadas.
3. Clique em seu sistema no painel à esquerda e em seguida, clique com o botão direito do mouse em **DATABASE** e selecione **Executar Scripts SQL** no lista suspensa.
   A janela Executar Scripts SQL será aberta. Utilizando essa janela, você pode recortar e colar instruções SQL ou abrir um script SQL. Defina o esquema padrão utilizando as opções de configuração Conexão/JDBC.

Gerando Código Implementado

Esta seção descreve como utilizar a Ferramenta de Implementação de EJB para gerar o código implementado dos beans corporativos contidos no arquivo JAR de Exportação do EJB 1.1. Essa ferramenta é fornecida pelo WebSphere Application Server.

Para gerar o código implementado, faça o seguinte:

1. Em um prompt de comandos no WebSphere Commerce Server de destino, navegue para o seguinte diretório:
   - Windows *unidade*:\WebSphere\CommerceServer\temp

- AIX `/usr/WebSphere/CommerceServer/temp`
- Solaris `/opt/WebSphere/CommerceServer/temp`
- Linux `/opt/WebSphere/CommerceServer/temp`

2. 400 Em um prompt de comandos no WebSphere Commerce Server de destino, digite os seguintes comandos:

   ```
   STRQSH
   cd /QIBM/UserData/WebCommerce/instances/instanceName/temp
   ```

3. Inclua temporariamente a ferramenta no caminho do sistema digitando o seguinte comando:

   - Windows `PATH=unidade:\WebSphere\AppServer\deploytool;%PATH%`
   - AIX `PATH=/usr/WebSphere/AppServer/deploytool:$PATH`
   - Solaris `PATH=/opt/WebSphere/AppServer/deploytool:$PATH`
   - Linux `PATH=/opt/WebSphere/AppServer/deploytool:$PATH`
   - 400 `PATH=/QIBM/ProdData/WebASAdv4/bin:$PATH`

     `CP=ClassPathOfDepJARFiles`

     em que *ClassPathOfDepJARFiles* é o caminho da classe de quaisquer arquivos JAR dependentes. Observe que, se você modificou um bean corporativo existente do WebSphere Commerce, deverá incluir os arquivos `wcsejsclient.jar`, `wcsejbimpl.jar` e `xml4j.jar` no caminho da classe dos arquivos JAR dependentes. Assim, a seguir está um caminho de classe de exemplo para um caso em que um bean corporativo existente do WebSphere Commerce foi modificado:

     ```
     CP=/QIBM/UserData/WebCommerce/instances/instanceName/temp/lib/
     wcsejsclient.jar:
     /QIBM/UserData/WebASAdv4/WAS_AdminInstanceName/installedApps/
     WC_Enterprise_App_instanceName.ear/lib/wcsejbimpl.jar:
     /QIBM/UserData/WebASAdv4/WAS_AdminInstanceName/installedApps/
     WC_Enterprise_App_instanceName.ear/lib/xml4j.jar
     ```

     Nota: As quebras de linha no exemplo de caminho da classe anterior são somente para exibição. As informações sobre o caminho da classe devem ser digitadas em uma única linha.

4. Solaris Linux Para evitar que o limite de caracteres seja excedido na interface de linha de comandos, é utilizado um script para chamar o comando `ejbdeploy`. Faça o seguinte para criar este script e chamar o comando:

   a. Navegue para o seguinte diretório:

      `/opt/WebSphere/AppServer/bin`

b. Crie um novo arquivo e nomeie-o apropriadamente para o script. Por exemplo, nomeie o arquivo como `myejbd.sh`.

c. Inclua as seguintes informações no arquivo `myejbd.sh`:

```
#!/bin/sh
./ejbdeploy.sh EJBGroupJARFile WorkingDir OutputJARFile
-nowarn -keep -35 -cp ClassPathOfDepJARFiles
```

em que as variáveis têm as mesmas definições que as da etapa 5 (observe que você não executa essa etapa). A seguir está um exemplo do que deve ser incluído no arquivo de script, com valores para as variáveis incluídas:

```
#!/bin/sh
./ejbdeploy.sh
/opt/WebSphere/CommerceServer/temp/CustomizedWCSUserDeployed_DT.jar
. /opt/WebSphere/CommerceServer/temp/CustomizedWCSUserDeployed.jar
-nowarn -keep -35 -cp "/opt/WebSphere/AppServer/installedApps/
-nowarn -keep -35 -cp "/opt/WebSphere/AppServer/installedApps/
WC_Enterprise_App_demo.ear/lib/wcsejbimpl.jar:/opt/WebSphere/
AppServer/installedApps/WC_Enterprise_App_demo.ear/lib/xml4j.jar"
```

Nota: Todas as quebras de linha após o comando `./ejbdeploy.sh` destinam-se somente para fins de exibição. Você não deve ter quebras após o comando `./ejbdeploy.sh` em seu arquivo.

Salve o script e torne-o executável.

d. Chame o script digitando o seguinte:

```
./myejbd.sh
```

5. Windows AIX 400 Digite o comando `ejbdeploy` desta forma:

Windows 400

```
ejbdeploy EJBGroupJARFile_DT WorkingDir OutputJARFile -nowarn -keep -35 -cp
   ClassPathOfDepJARFiles
```

AIX

```
/usr/WebSphere/AppServer/bin/ejbdeploy.sh EJBGroupJARFile_DT WorkingDir
OutputJARFile -nowarn -keep -35
-cp ClassPathOfDepJARFiles
```

em que:

- *EJBGroupJARFile*_DT é o nome do arquivo JAR de seu grupo de EJB. Por exemplo, se você tiver modificado um bean no grupo EJB WCSUser, um exemplo de utilização em uma plataforma baseada em Windows seria

  ```
  unidade:\WebSphere\CommerceServer\temp\CustomizedWCSUser_DT.jar
  ```

  400 Um exemplo de utilização na plataforma iSeries é

  ```
  /QIBM/UserData/WebCommerce/instances/instanceName/temp/
  CustomizedWCSUser_DT.jar
  ```

- *WorkingDir* o nome do diretório no qual estão armazenados os arquivos temporários necessários para a geração de código.
- *OutputJARFile* é o nome completo do arquivo JAR de saída. Por exemplo, você pode digitar CustomizedWCSUserDeployed.jar.
- -nowarn é um parâmetro opcional para anular mensagens de aviso e de informações.
- -keep é um parâmetro opcional para manter o diretório de trabalho depois que o comando ejbdeploy for executado.
- -35 é um parâmetro obrigatório que utiliza as mesmas regras de mapeamento top-down para beans de entidade CMP utilizados na Ferramenta de Implementação de EJB fornecida com o WebSphere Application Server, Versão 3.5.
- -cp *ClassPathOfDepJARFiles* é o caminho de classe de quaisquer arquivos JAR dependentes. Quando tiver modificado um bean corporativo existente do WebSphere Commerce, será necessário incluir os arquivos wcsejsclient.jar, wcsejbimpl.jar e xml4j.jar no caminho de classe de arquivos JAR dependentes.Assim, a seguir está um caminho de classe de exemplo para um caso em que um bean corporativo existente do WebSphere Commerce foi modificado:

```
"unidade:\WebSphere\CommerceServer\temp\lib\wcsejsclient.jar;
unidade:\WebSphere\AppServer\installedApps\
WC_Enterprise_App_instanceName.ear \lib\wcsejbimpl.jar;
unidade:\WebSphere\AppServer\installedApps\
WC_Enterprise_App_instanceName.ear\lib\xml4j.jar;"
```

Nota: 400 Para obter os valores de caminho da classe, digite -cp $CP em que CP já foi definido com as entradas de caminho da classe apropriadas. As aspas não são necessárias.

Modificando o Nível de Isolamento de Transação dos Beans de Entidade

Esta seção descreve como usar o utilitário modifyIsolationLevel da linha de comandos para definir o nível de isolamento de transação de seus beans de entidade para o nível apropriado de seu tipo de banco de dados.

Para executar o comando modifyIsolationLevel, faça o seguinte:

1. No WebSphere Commerce Server de destino, utilize um prompt de comandos para navegar para o seguinte diretório:
   - Windows *unidade*:\WebSphere\CommerceServer\bin
   - AIX /usr/WebSphere/CommerceServer/bin
   - Solaris /opt/WebSphere/CommerceServer/bin
   - Linux /opt/WebSphere/CommerceServer/bin

2. 400 Digite os seguintes comandos:

```
STRQSH
cd /QIBM/ProdData/WebCommerce/bin
```

3. Você deve emitir o comando `modifyIsolationLevel`, que tem a seguinte sintaxe geral:

Windows AIX 400

```
modifyIsolationLevel -jarFile jar_file_name.jar
   -logFile log_file_name -dbType db_type
```

Solaris Linux

```
./modifyIsolationLevel.sh -jarFile jar_file_name.jar
   -logFile log_file_name -dbType db_type
```

onde

- *jar_file_name.jar* é o nome do arquivo JAR que contém o código personalizado
- *log_file_name* é o nome de arquivo completo no qual as informações devem ser registradas
- *db_type* é o tipo de banco de dados que está sendo utilizado. Digite DB2 ou ORACLE

A seguir é fornecido um exemplo do comando `modifyIsolationLevel` com todos os valores especificados para uso em uma plataforma Windows:

```
modifyIsolationLevel -jarFile
   D:\WebSphere\CommerceServer\temp\CustomizedWCSUserDeployed.jar
   -logFile D:\WebSphere\CommerceServer\instances\demo\logs\output.log
   -dbType DB2
```

400 A seguir está um exemplo do comando `modifyIsolationLevel` com todos os valores especificados para uso no iSeries:

```
modifyIsolationLevel -jarFile
   /QIBM/UserData/WebCommerce/instances/instanceName/temp/
   CustomizedWCSUserDeployed.jar -logFile /QIBM/UserData/WebCommerce/
   instances/instanceName/logs/output.log -dbType DB2
```

Observe que nos exemplos anteriores, as quebras de linha são somente para exibição.

Nota: Os nomes dos parâmetros fazem distinção entre maiúsculas e minúsculas. Ou seja, `jarFile` não é o mesmo que `jarfile`. Verifique se os nomes dos parâmetros estão digitados corretamente.

O comando terá sido executado com êxito se nenhuma exceção for exibida na janela de comando. Depois da conclusão, observe que a marca de hora em seu arquivo JAR foi alterada.

Exportando o Aplicativo Corporativo Atual do WebSphere Commerce

Esta seção descreve como utilizar o WebSphere Application Server Administration Console para exportar o aplicativo corporativo atual do WebSphere Commerce.

Para exportar o aplicativo corporativo atual do WebSphere Application Server, faça o seguinte:

1. Certifique-se de que o seguinte diretório esteja no WebSphere Commerce Server de destino:
   - Windows *unidade*:\WebSphere\CommerceServer\working
   - AIX /usr/WebSphere/CommerceServer/working
   - Solaris /opt/WebSphere/CommerceServer/working
   - Linux /opt/WebSphere/CommerceServer/working
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/working

   Se ainda não tiver este diretório, crie-o agora.

   Nota: AIX Solaris Linux Certifique-se de que a permissão para o diretório /working esteja definida como o usuário criado quando as etapas na seção *"Executando o script postinstall"* do *WebSphere Commerce - Manual de Instalação* foram executadas.

   400 Certifique-se de que as autoridades para os diretórios /QIBM/UserData/WebCommerce/instances/*instanceName* e /QIBM/UserData/WebCommerce/instances/*instanceName*/working incluam o usuário QEJB. Inclua este usuários nos dois diretórios, definindo a Autoridade de Dados como *RWX.
2. Abra o WebSphere Application Server Administration Console.
3. Expanda o **Domínio Administrativo do WebSphere**.
4. Expanda **Aplicativo Corporativo**.
5. Clique com o botão direito do mouse em seu aplicativo do WebSphere Commerce. Por exemplo, expanda o aplicativo **demo** e selecione **Exportar Aplicativo**.
6. No campo **Exportar diretório**, digite o seguinte:
   - Windows *unidade*:\WebSphere\CommerceServer\working
   - AIX /usr/WebSphere/CommerceServer/working
   - Solaris /opt/WebSphere/CommerceServer/working
   - Linux /opt/WebSphere/CommerceServer/working

- 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/working

Isso exporta todo o aplicativo, incluindo todos os recursos, para o arquivo WC_Enterprise_App_*instanceName*.ear (em que *instanceName* e o nome de sua instância do WebSphere Commerce).

Exportando Informações de Configuração dos Beans Corporativos

Esta seção descreve como utilizar a opção -export do utilitário XMLConfig da linha de comandos para exportar as informações de configuração dos beans corporativos contidos no aplicativo corporativo existente.

Depois que as informações dos beans do aplicativo existente forem exportadas, as informações para novos beans que estiverem sendo incluídos no aplicativo deverão ser incluídas manualmente.

Para exportar estas informações de configuração, faça o seguinte:

1. Copie o arquivo was.export.app.xml do seguinte diretório no WebSphere Commerce Server de destino:
   - Windows *unidade*:\WebSphere\CommerceServer\xml\config
   - AIX /usr/WebSphere/CommerceServer/xml/config
   - Solaris /opt/WebSphere/CommerceServer/xml/config
   - Linux /opt/WebSphere/CommerceServer/xml/config
   - 400 /QIBM/ProdData/WebCommerce/xml/config

   para o seguinte diretório no WebSphere Commerce Server de destino:
   - Windows *unidade*:\WebSphere\CommerceServer\working
   - AIX /usr/WebSphere/CommerceServer/working
   - Solaris /opt/WebSphere/CommerceServer/working
   - Linux /opt/WebSphere/CommerceServer/working
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/working

   em que
   – *instanceName* é o nome da instância do WebSphere Commerce.
2. Windows AIX Solaris Linux Abra o arquivo was.export.app.xml em um editor de texto. Neste arquivo, substitua todas as ocorrências de $Enterprise_Application_Name$ com

   WebSphere Commerce Enterprise Application - *instanceName*

em que *instanceName* é o nome da instância do WebSphere Commerce (por exemplo, demo). Salve esse arquivo.

Nota: O valor que está sendo inserido deve corresponder às informações de sua instância exibidas no WebSphere Application Server Administration Console.

3. 400 Abra o arquivo `was.export.app.xml` em um editor de texto. Neste arquivo, substitua todas as ocorrências de `$Enterprise_Application_Name$` com

   instanceName - WebSphere Commerce Enterprise Application

   em que *instanceName* é o nome de sua instância do WebSphere Commerce (por exemplo, demo). Salve esse arquivo.

   Nota: O valor que está sendo inserido deve corresponder às informações de sua instância exibidas no WebSphere Application Server Administration Console.

4. Windows AIX Solaris Linux Em um prompt de comandos, navegue para o seguinte diretório:
   - Windows *unidade*:\WebSphere\CommerceServer\working
   - AIX /usr/WebSphere/CommerceServer/working
   - Solaris /opt/WebSphere/CommerceServer/working
   - Linux /opt/WebSphere/CommerceServer/working

5. 400 Em um prompt de comandos, digite o seguinte:

   ```
   STRQSH
   PATH=/QIBM/ProdData/WebASAdv4/bin:$PATH
   cd /QIBM/UserData/WebCommerce/instances/instanceName/working
   ```

6. Windows AIX Solaris Linux Chame a ferramenta XMLConfig para executar uma exportação parcial digitando o seguinte comando:

   Windows

   ```
   xmlConfig -export OutputFile.xml -partial was.export.app.xml
      -adminNodeName wasHostName
   ```

   AIX

   ```
   /usr/WebSphere/AppServer/bin/XMLConfig.sh  -export OutputFile.xml
      -partial was.export.app.xml -adminNodeName wasHostName
    -nameServiceHost wasHostName -nameServicePort wasAdminPort
   ```

   Solaris Linux

```
/opt/WebSphere/AppServer/bin/XMLConfig.sh  -export OutputFile.xml
   -partial was.export.app.xml -adminNodeName wasHostName
 -nameServiceHost wasHostName -nameServicePort wasAdminPort
```

em que

- *wasHostName* é o nome do nó no WebSphere Application Server que contém o aplicativo corporativo atual.
- `OutputFile.xml` é o nome do arquivo criado como resultado da execução deste comando e `was.export.app.xml` é o arquivo modificado na etapa 2.
- *wasAdminPort* é a porta de administração do WebSphere Application Server.

7. 400 Chame a ferramenta XMLConfig para executar uma exportação parcial digitando o seguinte comando:

```
xmlConfig -export OutputFile.xml -partial was.export.app.xml
-adminNodeName wasHostName -nameServiceHost wasHostName
-nameServicePort wasAdminPort -instance wasInstanceName
```

em que

- *wasHostName* é o nome do nó no WebSphere Application Server que contém o aplicativo corporativo atual.

  Nota: O valor de *wasHostName* faz distinção entre maiúsculas e minúsculas e deve corresponder ao valor que está na configuração TCP/IP (utilize um processador da linha de comandos para acessar CFGTCP, opção 12 e verificar o nome do host).
- `OutputFile.xml` é o nome do arquivo criado como resultado da execução deste comando e `was.export.app.xml` é o arquivo modificado na etapa 3.
- *wasAdminPort* é a porta de administração do WebSphere Application Server.
- *wasInstanceName* é o nome de instância do WebSphere Application Server.

Depois de ter exportado as informações de configuração de cada um dos beans contidos no aplicativo corporativo atual, será necessário incluir uma nova sub-rotina que descreva cada bean corporativo que será incluído em seu aplicativo. Por exemplo, se você tiver um novo bean de entidade chamado “Bonus”, deverá incluir uma sub-rotina que descreva este bean Bonus. Além disso, deverá substituir uma variável no arquivo de configuração para especificar o nome exato do arquivo .ear de seu aplicativo corporativo.

Para fazer essas atualizações no arquivo `OutputFile.xml`, faça o seguinte:

1. Navegue para o seguinte diretório no WebSphere Commerce Server de destino:
   - Windows *unidade*:\WebSphere\CommerceServer\working
   - AIX /usr/WebSphere/CommerceServer/working
   - Solaris /opt/WebSphere/CommerceServer/working
   - Linux /opt/WebSphere/CommerceServer/working
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/working
2. Abra o arquivo OutputFile.xml em um editor de texto.
3. Localize a tag <ear-file-name> e substitua o valor pelo seguinte:
   - Windows *unidade*:\WebSphere\CommerceServer\working\WC_Enterprise_App_*instanceName*.ear
   - AIX /usr/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
   - Solaris /opt/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
   - Linux /opt/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/working/WC_Enterprise_App_*instanceName*.ear
4. Também será necessário incluir uma nova sub-rotina para cada novo bean que estiver sendo incluído no aplicativo corporativo. O exemplo a seguir mostra como incluir um novo bean, chamado "Bonus".

   Windows AIX Solaris Linux

```
<ejb-module name="yourEJBGroup">
   <jar-file>yourDeployedJarFile.jar</jar-file>
   <module-install-info>
      <application-server-full-name>/NodeHome:$hostName$/EJBServerHome:
         WebSphere Commerce Server - instanceName/
      </application-server-full-name>
   </module-install-info>
   <ejb-module-binding>
      <data-source>
         <jndi-name>jdbc/WebSphere Commerce DB2 DataSource instanceName
            </jndi-name>
         <default-user>user</default-user>
         <default-password>password</default-password>
      </data-source>
      <enterprise-bean-binding name="BeanBindingName">
```

```
        <jndi-name>instanceNameJNDINameOfBean</jndi-name>
      </enterprise-bean-binding>
   </ejb-module-binding>
</ejb-module>
```

400

```
<ejb-module name="yourEJBGroup">
   <jar-file>yourDeployedJarFile.jar</jar-file>
   <module-install-info>
      <application-server-full-name>/NodeHome:$hostName$/EJBServerHome:
         instanceName - WebSphere Commerce Server/
      </application-server-full-name>
   </module-install-info>
   <ejb-module-binding>
      <data-source>
         <jndi-name>jdbc/instanceName WebSphere Commerce DB2 DataSource
            </jndi-name>
         <default-user>user</default-user>
         <default-password>password</default-password>
      </data-source>
      <enterprise-bean-binding name="BeanBindingName">
         <jndi-name>instanceNameJNDINameOfBean</jndi-name>
      </enterprise-bean-binding>
   </ejb-module-binding>
</ejb-module>
```

em que

- *yourEJBGroup* é o nome do grupo EJB que contém o bean que está sendo incluído no aplicativo corporativo.
- *yourDeployedJarFile* é o nome do arquivo JAR que contém o código implementado do grupo de EJB.
- *instanceName* é o nome da instância do WebSphere Commerce na qual você está implementando o código.
- *user* é o nome do usuário do banco de dados.
- *password* é a senha do usuário do banco de dados.
- *BeanBindingName* é o nome de ligação do bean corporativo. Por exemplo, para o bean Bonus, é `Bonus_Binding`.
- *instanceNameJNDINameOfBean* é o nome JNDI do bean corporativo com a instância do WebSphere Commerce incluída. Esse nome JNDI deve corresponder exatamente ao do bean corporativo. Um valor de exemplo é `democom/ibm/commerce/sample/objects/Bonus`. Neste exemplo, "`demo`" é o nome da instância e "`com/ibm/commerce/sample/objects/Bonus`" é o nome JNDI do bean corporativo. Este valor deve ser digitado em uma única linha.
  Você pode verificar o nome JNDI utilizando VisualAge for Java ou a

Ferramenta de Montagem de Aplicativos.
Para verificar o nome JNDI utilizando o VisualAge for Java, faça o seguinte:

a. Clique com o botão direito do mouse no bean e selecione **Propriedades**.
   O nome JNDI é exibido.

O nome JNDI pode ser verificado utilizando a Ferramenta de Montagem de Aplicativos, porém, isto requer que você já tenha montado o novo bean corporativo no aplicativo corporativo. A Ferramenta de Montagem de Aplicativos é acessada a partir do menu Ferramentas no WebSphere Application Server Administration Console.

Para verificar o nome JNDI utilizando a Ferramenta de Montagem de Aplicativos, faça o seguinte:

a. Abra o arquivo .ear que contém o bean.
b. Expanda o módulo de EJB que contém o bean.
c. Selecione o bean.
d. Clique na guia **Ligações**.
   O nome JNDI é exibido.

Observe que utilizar um desses métodos mostra o nome JNDI e você ainda deverá anexar o nome da instância do WebSphere Commerce ao incluir as informações no arquivo XML.

Notas:

a. As quebras de linha nas sub-rotinas anteriores destinam-se somente para fins de exibição.
b. Assegure-se de que o valor **$hostName$** corresponda ao nome atual do servidor do nó de administração. Além disso, assegure-se de que não haja nenhum caractere de retorno do carro nesta linha.
c. A especificação `<application-server-full-name>` não pode ocupar mais de uma linha.
d. Se estiver utilizando um banco de dados Oracle, deverá modificar as informações de origem de dados. Altere a seguinte linha retirada do fragmento de código anterior:

```
<jndi-name>jdbc/WebSphere Commerce DB2 DataSource instanceName
   </jndi-name>
```

   para o seguinte:

```
<jndi-name>jdbc/WebSphere Commerce Oracle DataSource instanceName
   </jndi-name>
```

5. Se você estiver implementando beans de entidade modificados do WebSphere Commerce, atualize as sub-rotinas desses beans para refletir os nomes dos arquivos JAR que contêm o código implementado para os beans modificados.
6. Salve o arquivo `OutputFile.xml`.

Montando Novos Beans Corporativos em um Aplicativo Corporativo

Esta seção descreve como utilizar a Ferramenta de Montagem de Aplicativos para montar novos beans corporativos em um aplicativo corporativo existente.

400 Como a Ferramenta de Montagem de Aplicativos executa na plataforma Windows, faça referência à unidade mapeada para o IFS do iSeries quando forem solicitados nomes de caminho completos. Isto é chamado de *iSeries_drive*.

AIX Solaris Linux É recomendável aumentar o valor do tamanho de heap da memória no arquivo `assembly.sh` para evitar falta de memória ao salvar arquivos `.ear` novos ou modificados.

Este arquivo está localizado no seguinte diretório:

- AIX `/usr/WebSphere/AppServer/bin`
- Solaris `/opt/WebSphere/AppServer/bin`
- Linux `/opt/WebSphere/AppServer/bin`

Para aumentar o tamanho de heap da memória, modifique a seguinte linha:

```
$JAVA_HOME/jre/bin/java
```

para que tenha o seguinte aspecto:

```
$JAVA_HOME/jre/bin/java -mx512M
```

Nesta etapa, você abre o arquivo `.ear` do aplicativo corporativo criado na seção Exportando o Aplicativo Corporativo Atual do WebSphere Commerce na ferramenta de montagem de aplicativos. Assim que estiver aberto nessa ferramenta, execute as seguintes tarefas para incluir o novo bean de entidade no aplicativo corporativo:

1. Importe o grupo de EJB que contém o novo bean de entidade. O arquivo JAR do novo grupo de EJB será armazenado na seção Módulo de EJB do aplicativo corporativo.
2. Defina o caminho de classe do novo bean de entidade para que inclua o arquivo JAR de implementação.

3. Inclua o arquivo JAR de implementação no aplicativo. Este arquivo JAR será armazenado na seção Arquivos do aplicativo corporativo.
4. Configure a segurança do WebSphere Application Server para os métodos contidos no novo bean de entidade.

Para montar o novo grupo EJB no aplicativo corporativo, faça o seguinte:

1. Windows AIX Solaris Linux Faça backup do aplicativo corporativo atual, fazendo o seguinte no WebSphere Commerce Server de destino:
   a. No prompt de comandos, navegue para o seguinte diretório:
      - Windows *unidade*:\WebSphere\CommerceServer\working
      - AIX /usr/WebSphere/CommerceServer/working
      - Solaris /opt/WebSphere/CommerceServer/working
      - Linux /opt/WebSphere/CommerceServer/working
   b. Faça uma cópia do arquivo WC_Enterprise_App_*instanceName*.ear existente e nomeie-o como WC_Enterprise_App_*instanceName*.ear.bak.
2. 400 Faça um backup do aplicativo corporativo atual, da seguinte maneira:
   a. No prompt de comandos, digite o seguinte:

      ```
      STRQSH
      cd /QIBM/UserData/WebCommerce/instances/instanceName/working
      cp WC_Enterprise_App_instanceName.ear
      WC_Enterprise_App_instanceName.ear.bak
      ```

   b. Para evitar uma longa espera desnecessária provocada pela transferência de dados entre a máquina cliente local e a máquina do iSeries que executa o WebSphere Application Server, crie o seguinte diretório na máquina cliente local e copie o arquivo WC_Enterprise_App_*instanceName*.ear para esse diretório:
      unidade:\WebSphere\CommerceServer\working
      em que *unidade* é uma unidade local.

      Nota: Se o diretório *unidade*:\WebSphere\CommerceServer\working não existir na máquina local, crie-o agora.
3. Abra o WebSphere Application Server Administration Console.
4. No menu **Ferramentas**, selecione **Ferramenta de Montagem de Aplicativo**.
5. Se uma janela Bem-vindo for aberta, selecione **Cancelar** para fechá-la.
6. Abra o aplicativo corporativo com o qual irá trabalhar, da seguinte forma:
   a. A partir do menu **Arquivo**, selecione **Abrir**.

b. No campo **Nome do arquivo**, digite:

- Windows *unidade*:\WebSphere\CommerceServer\working\WC_Enterprise_App_*instanceName*.ear
- AIX /usr/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
- Solaris /opt/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
- Linux /opt/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
- 400 *unidade*:\WebSphere\CommerceServer\working\WC_Enterprise_App_*instanceName*.ear

e clique em **Abrir**. Aguarde até que o aplicativo seja aberto antes de prosseguir com as próximas etapas. Isto pode levar alguns minutos.

7. Clique com o botão direito do mouse nos **Módulos EJB** e selecione **Importar**.
8. No campo **Nome do arquivo**, digite o seguinte:
   - Windows *unidade*:\WebSphere\CommerceServer\temp*yourDeployedJarFile*.jar
   - AIX /usr/WebSphere/CommerceServer/temp/*yourDeployedJarFile*.jar
   - Solaris /opt/WebSphere/CommerceServer/temp/*yourDeployedJarFile*.jar
   - Linux /opt/WebSphere/CommerceServer/temp/*yourDeployedJarFile*.jar
   - 400 *iSeries_drive*:\QIBM\UserData\WebCommerce\instances*instanceName*\temp*yourDeployedJarFile*.jar

   onde
   - *yourDeployedJarFile* é o nome do arquivo JAR que contém o código implementado do grupo EJB
   - *iSeries_drive* é a unidade local mapeada para o IFS do iSeries.

   Clique em **Abrir** e em seguida, na janela Confirmar Valores e em **OK**.
9. Depois que o arquivo *yourDeployedJarFile*.jar for importado, desloque para o grupo de EJB *yourEJBGroup* (em que *yourEJBGroup* é o nome de seu grupo de EJB) e selecione este grupo.
   Informações sobre este grupo são exibidas no painel à direita.
10. No campo do classpath para o novo bean corporativo, digite o nome de quaisquer arquivo JAR dependentes. Por exemplo, você pode inserir o

arquivo JAR de implementação correspondente e o arquivo JAR de implementação para os beans de entidade do WebSphere Commerce, conforme mostrado a seguir:

`lib/yourImplJarFile.jar lib/wcsejbimpl.jar`

11. Clique em **Aplicar**.
12. Inclua o arquivo JAR de implementação do grupo de EJB no aplicativo, fazendo o seguinte:
    a. Clique com o botão direito do mouse no nó **Arquivos** do aplicativo corporativo e selecione **Incluir Arquivos**. (O nó **Arquivos** do aplicativo corporativo está localizado próximo à parte inferior da árvore hierárquica. Observe que existem outros nós de Arquivos para componentes dentro do aplicativo corporativo, mas você deverá selecionar o nó de Arquivos para todo o aplicativo.)
    b. Na janela Incluir Arquivos, clique em **Procurar**.
    c. Navegue para o seguinte diretório:
       - Windows *unidade*:\WebSphere\CommerceServer\temp
       - AIX /usr/WebSphere/CommerceServer/temp
       - Solaris /opt/WebSphere/CommerceServer/temp
       - Linux /opt/WebSphere/CommerceServer/temp
       - 400 *iSeries_drive*:\QIBM\UserData\WebCommerce\instances*instanceName*\temp
    d. Com este diretório em destaque, clique em **Selecionar**.
    e. Retorne à janela Incluir Arquivos. Observe que o conteúdo do diretório temporário é exibido. Destaque o diretório `lib`. O conteúdo do diretório `lib` será exibido no painel à direita.
    f. No painel à direita, selecione o arquivo *yourImplJarFile* e clique em **Incluir**. O arquivo então será exibido no painel Arquivos Selecionados.
    g. Clique em **OK**.
13. Configure a segurança do bean de entidade, fazendo o seguinte:
    a. Com o nó Módulos de EJB expandido, localize e expanda o nó *YourEJBGroup*.
    b. Expanda **Beans de Entidade**.
    c. Expanda *yourEntityBean* em que *yourEntityBean* é o nome do bean de entidade.
    d. Clique em **Extensões do Método** e em seguida, no painel à direita, faça o seguinte:
       1) Clique na guia **Avançado**.
       2) Assegure-se de que **Identidade de segurança** esteja selecionado.

3) Para cada método, assegure-se de que **Utilizar identidade do servidor EJB** esteja selecionado.
4) Clique em **Aplicar** (se tiver feito alguma modificação).

e. No painel de navegação à esquerda, clique com o botão direito do mouse em **Funções de Segurança** no grupo de EJB *YourEJBGroup* e selecione **Novo**; depois faça o seguinte:
1) No campo **Nome**, digite `WCSecurityRole` e clique em **Aplicar**. Observe que, se esta função já existir, não será necessário executar esta etapa.

f. No painel de navegação à esquerda, clique com o botão direito do mouse em **Permissões do Método** no grupo de EJB *YourEJBGroup* e selecione **Novo**; depois faça o seguinte:
1) No campo **Nome de Permissão do Método**, digite `WCMethodPermission`
2) Na área de seleção **Métodos**, clique em **Incluir**. A janela Incluir Métodos é aberta.
3) Expanda *yourDeployedJarFile***.jar**, **Bonus**, e depois expanda cada uma das listas de métodos **Inicial** e **Remota**.
4) Mantenha pressionada a tecla Shift e selecione todos os métodos iniciais e clique em **OK**.
5) Repita o processo de seleção do método para incluir também os métodos remotos (se existirem métodos remotos).
6) Na área de seleção Funções, clique em **Incluir**, selecione a `WCSecurityRole` e clique em **OK**.
7) Clique em **Aplicar**.

14. A partir do menu **Arquivo**, selecione **Salvar**.
15. Feche a Ferramenta de Montagem de Aplicativos.
16. 400 Copie o arquivo `WC_Enterprise_App_`*`instanceName`*`.ear` modificado recentemente da máquina local para a máquina do iSeries que executa o WebSphere Application Server. Isto é, copie o arquivo do seguinte diretório:

    `unidade``:\WebSphere\CommerceServer\working`

    para o seguinte diretório:

    `iSeries_drive``:\QIBM\UserData\WebCommerce\instances\`*`instanceName`*`\working`

    em que *`iSeries_drive`* é a letra da unidade mapeada para o IFS do iSeries.

Depois de concluir desta etapa, você terá criado um novo aplicativo corporativo que contém toda a lógica anterior, bem como sua nova lógica de negócios. Isto tudo está contido no arquivo recém-modificado `WC_Enterprise_App_instanceName.ear`.

Montando Beans Corporativos Modificados em um Aplicativo Corporativo

Esta seção descreve como utilizar a Ferramenta de Montagem de Aplicativos para montar beans corporativos modificados do WebSphere Commerce em um aplicativo corporativo.

Nesta etapa, você irá abrir seu aplicativo corporativo na ferramenta de montagem de aplicativo. Assim que estiver aberto nessa ferramenta, você poderá fazer o seguinte para incluir um bean corporativo modificado do WebSphere Commerce no aplicativo corporativo:

1. Faça uma cópia do caminho de classe da versão existente do grupo de EJB modificado.
2. Remova a versão existente do grupo de EJB que você modificou.
3. Importe a nova versão do grupo de EJB que você modificou. O arquivo JAR para o novo grupo EJB será armazenado dentro da seção do Módulo EJB do aplicativo corporativo.
4. Defina o caminho de classe do grupo de EJB modificado.
5. Configure a segurança do WebSphere Application Server para os métodos contidos no bean de entidade modificado.

AIX Solaris Linux É recomendável aumentar o valor do tamanho de heap da memória no arquivo `assembly.sh` para evitar falta de memória ao salvar arquivos `.ear` novos ou modificados.

Este arquivo está localizado no seguinte diretório:

- AIX `/usr/WebSphere/AppServer/bin`
- Solaris `/opt/WebSphere/AppServer/bin`
- Linux `/opt/WebSphere/AppServer/bin`

Para aumentar o tamanho de heap da memória, modifique a seguinte linha:

```
$JAVA_HOME/jre/bin/java
```

para que tenha o seguinte aspecto:

```
$JAVA_HOME/jre/bin/java -mx512M
```

Para montar o grupo de EJB modificado no aplicativo corporativo, faça o seguinte:

1. Windows AIX Solaris Linux Faça backup do aplicativo corporativo atual, fazendo o seguinte no WebSphere Commerce Server de destino:
   a. No prompt de comandos, navegue para o seguinte diretório:
      - Windows `unidade:\WebSphere\CommerceServer\working`
      - AIX `/usr/WebSphere/CommerceServer/working`
      - Solaris `/opt/WebSphere/CommerceServer/working`
      - Linux `/opt/WebSphere/CommerceServer/working`
   b. Faça uma cópia do arquivo `WC_Enterprise_App_instanceName.ear` existente e nomeie-o como `WC_Enterprise_App_instanceName.ear.bak`.
2. 400 Faça um backup do aplicativo corporativo atual, da seguinte maneira:
   a. No prompt de comandos, digite o seguinte:
      ```
      STRQSH
      cd /QIBM/UserData/WebCommerce/instances/instanceName/working
      cp WC_Enterprise_App_instanceName.ear
      WC_Enterprise_App_instanceName.ear.bak
      ```
   b. Para evitar uma longa espera desnecessária provocada pela transferência de dados entre a máquina cliente local e a máquina do iSeries que executa o WebSphere Application Server, crie o seguinte diretório na máquina cliente local e copie o arquivo `WC_Enterprise_App_instanceName.ear` para esse diretório:
      `unidade:\WebSphere\CommerceServer\working`
      em que *unidade* é uma unidade local.

      Nota: Se o diretório `unidade:\WebSphere\CommerceServer\working` não existir na máquina local, crie-o agora.
3. Abra o WebSphere Application Server Administration Console.
4. No menu **Ferramentas**, selecione **Ferramenta de Montagem de Aplicativo**.
5. Abra o aplicativo corporativo com o qual irá trabalhar da seguinte maneira:
   a. A partir do menu **Arquivo**, selecione **Abrir**.
   b. No campo **Nome do arquivo**, digite:
      - Windows `unidade:\WebSphere\CommerceServer\working\WC_Enterprise_App_instanceName.ear`
      - AIX `/usr/WebSphere/CommerceServer/working/WC_Enterprise_App_instanceName.ear`

- Solaris `/opt/WebSphere/CommerceServer/working/WC_Enterprise_App_instanceName.ear`
- Linux `/opt/WebSphere/CommerceServer/working/WC_Enterprise_App_instanceName.ear`
- 400 `unidade:\WebSphere\CommerceServer\working\WC_Enterprise_App_instanceName.ear`

e clique em **Abrir**. Aguarde até que o aplicativo seja aberto antes de prosseguir com as próximas etapas. Isto pode levar alguns minutos.

6. Clique em **Módulos EJB**. O painel à direita exibe os módulos EJB no aplicativo corporativo.
7. Clique no módulo de EJB do grupo de EJB que você modificou. Por exemplo, se você modificou um bean no grupo de EJB WCSUser, clique em **WCSUser**.
8. Clique na guia Geral para exibir as informações do caminho de classe. Copie estas informações de caminho de classe existentes em um arquivo texto (por exemplo, `WCSUser_path.txt`).
9. Clique com o botão direito do mouse no módulo de EJB e selecione **Excluir**.
10. Clique com o botão direito do mouse nos **Módulos EJB** e selecione **Importar**.
11. No campo **Nome do arquivo**, digite o seguinte:
    - Windows `unidade:\WebSphere\CommerceServer\temp\Cust_EJBGroupName-ejb.jar`
    - AIX `/usr/WebSphere/CommerceServer/temp/Cust_EJBGroupName-ejb.jar`
    - Solaris `/opt/WebSphere/CommerceServer/temp/Cust_EJBGroupName-ejb.jar`
    - Linux `/opt/WebSphere/CommerceServer/temp/Cust_EJBGroupName-ejb.jar`
    - 400 `iSeries_drive:\QIBM\UserData\WebCommerce\instances\instanceName\temp\Cust_EJBGroupName-ejb.jar`

    e clique em **Abrir**. Na janela Confirmar Valores, clique em **OK**.
12. Assim que o arquivo `EJBGroupJARFile.jar` tiver sido importado, role para o grupo de EJB modificado e selecione este grupo.
    As informações sobre este grupo serão exibidas no painel à direita.
13. Abra o arquivo de texto contendo as informações de caminho de classe da versão anterior do grupo de EJB. Selecione e copie o caminho de classe.

14. No campo **Classpath** do grupo EJB modificado, cole estas informações do caminho da classe.
15. Clique em **Aplicar**.
16. No menu **Arquivo**, selecione **Fechar**.
17. Aguarde o fechamento do arquivo e em seguida, no menu **Arquivo**, selecione **Abrir** e reabra o arquivo `WC_Enterprise_App_instanceName.ear`.
18. Configure a segurança do bean modificado, fazendo o seguinte:
    a. Com o nó Módulos de EJB expandido, localize e expanda o nó do grupo de EJB modificado.
    b. Expanda **Beans de Entidade**.
    c. Expanda o grupo de EJB modificado.
    d. Clique em **Extensões do Método** e em seguida, no painel à direita, faça o seguinte:
        1) Clique na guia **Avançado**.
        2) Assegure-se de que **Identidade de segurança** esteja selecionado.
        3) Para cada método, assegure-se de que **Utilizar identidade do servidor EJB** esteja selecionado.
        4) Clique em **Aplicar** (se tiver feito alguma modificação).
    e. No painel de navegação à esquerda, clique com o botão direito do mouse em **Funções de Segurança** no grupo EJB modificado, selecione **Novo** e proceda da seguinte forma:
        1) No campo **Nome**, digite `WCSecurityRole` e clique em **Aplicar**. Observe que, se esta função já existir, não será necessário executar esta etapa.
    f. No painel de navegação esquerdo, clique com o botão direito do mouse em **Permissões do Método** no grupo EJB modificado, selecione **Novo** e proceda da seguinte forma:
        1) No campo **Nome de Permissão do Método**, digite `WCMethodPermission`
        2) Na área de seleção **Métodos**, clique em **Incluir**. A janela Incluir Métodos é aberta.
        3) Expanda o *modifiedEJBGroup* e selecione todos os beans corporativos (mantenha a tecla Shift pressionada durante a seleção). Clique em **OK**. Todos os beans corporativos serão exibidos na coluna Bean corporativo e todos os métodos serão mostrados na coluna Tipos.
        4) Na área de seleção Funções, clique em **Incluir**, selecione a `WCSecurityRole` e clique em **OK**.
        5) Clique em **Aplicar**.
19. A partir do menu **Arquivo**, selecione **Salvar**.

20. Feche a Ferramenta de Montagem de Aplicativos.
21. 400 Copie o arquivo `WC_Enterprise_App_instanceName.ear` modificado recentemente da máquina local para a máquina do iSeries que executa o WebSphere Application Server. Isto é, copie o arquivo do seguinte diretório:

    `unidade:\WebSphere\CommerceServer\working`

    para o seguinte diretório:

    `iSeries_drive:\QIBM\UserData\WebCommerce\instances\instanceName\working`

    em que *iSeries_drive* é a letra da unidade mapeada para o IFS do iSeries.

Depois de concluir desta etapa, você terá criado um novo aplicativo corporativo que contém toda a lógica anterior, bem como sua nova lógica de negócios. Isto tudo está contido no arquivo recém-modificado `WC_Enterprise_App_instanceName.ear`.

Parando e Removendo um Aplicativo Corporativo

Esta seção descreve como utilizar o WebSphere Application Server Administration Console para parar um aplicativo corporativo em execução atualmente e depois removê-lo.

Para parar e depois remover seu aplicativo corporativo, faça o seguinte:

1. Abra o Administration Console do WebSphere Application Server.
2. Expanda **Domínio Administrativo do WebSphere**.
3. Expanda os **Nós**.
4. Expanda ***nodeName*** (em que ***nodeName*** é o nome de seu nó).
5. Expanda **Servidores de Aplicativos**.
6. Windows AIX Solaris Linux Clique com o botão direito do mouse no servidor de aplicativos do WebSphere Commerce. Por exemplo, clique com o botão direito do mouse no **WebSphere Commerce Server - *nodeName*** e selecione **Parar**.
7. 400 Clique com o botão direito do mouse em seu servidor de aplicativos WebSphere Commerce. Por exemplo, clique com o botão direito do mouse no ***instanceName* - WebSphere Commerce Server** e selecione **Parar**.
8. Expanda **Aplicativos Corporativos**.
9. Windows AIX Solaris Linux Clique com o botão direito do mouse no aplicativo WebSphere Commerce. Por exemplo, clique com o

botão direito do mouse no aplicativo **WebSphere Commerce Enterprise Application - *instanceName*** e selecione **Parar**.

10. 400 Clique com o botão direito do mouse em seu aplicativo WebSphere Commerce. Por exemplo, clique com o botão direito do mouse no aplicativo ***instanceName* - WebSphere Commerce Enterprise Application** e selecione **Parar**.
11. Clique com o botão direito do mouse em seu aplicativo do WebSphere Commerce. Por exemplo, clique com o botão direito do mouse no aplicativo **WebSphere Commerce Enterprise Application -*instanceName*** (ou no aplicativo ***instanceName* - WebSphere Commerce Enterprise Application** para iSeries) e selecione **Remover**.
12. Quando solicitado se o aplicativo deve ser exportado, selecione **Não**.
13. Quando for perguntado se o aplicativo deve ser removido, selecione **Sim**.

Importando um Aplicativo Corporativo

Esta seção descreve como usar o utilitário XMLConfig da linha de comandos para importar um aplicativo corporativo.

Para importar o novo aplicativo corporativo, faça o seguinte:

1. Windows AIX Solaris Linux Chame a ferramenta XMLConfig para importar o aplicativo corporativo para o WebSphere Application Server digitando o seguinte comando:

   Windows

   ```
   xmlConfig -import OutputFile.xml -adminNodeName wasHostName
   ```

   AIX

   ```
   /usr/WebSphere/AppServer/bin/XMLConfig.sh -import OutputFile.xml
    -adminNodeName wasHostName
    -nameServiceHost wasHostName -nameServicePort wasAdminPort
   ```

   Solaris Linux

   ```
   /opt/WebSphere/AppServer/bin/XMLConfig.sh -import OutputFile.xml
    -adminNodeName wasHostName
    -nameServiceHost wasHostName -nameServicePort wasAdminPort
   ```

   onde

   - *wasHostName* é o nome do nó no WebSphere Application Server que contém o aplicativo corporativo atual.
   - *OutputFile.xml* é o arquivo XML que descreve todos os beans corporativos.

- *wasAdminPort* é a porta de administração do WebSphere Application Server.

2. 400 Chame a ferramenta XMLConfig para importar o aplicativo corporativo para o WebSphere Application Server digitando o seguinte comando:

```
STRQSH
PATH=/QIBM/ProdData/WebASAdv4/bin:$PATH
xmlConfig -import
/QIBM/UserData/WebCommerce/instances/instanceName/working/OutputFile.xml
-adminNodeName wasHostName
 -nameServiceHost wasHostName -nameServicePort wasAdminPort
-instance wasInstanceName
```

onde

- *OutputFile.xml* is the fully-qualfied name of the XML file that describes all of your enterprise beans.
- *wasHostName* é o nome do nó no WebSphere Application Server que contém o aplicativo corporativo atual.

  Nota: 400 O valor de *wasHostName* faz distinção entre maiúsculas e minúsculas e deve corresponder ao valor que está na configuração TCP/IP. (utilize um processador da linha de comandos para acessar CFGTCP, opção 12 e verificar o nome do host).
- *wasAdminPort* é a porta de administração do WebSphere Application Server.
- *wasInstanceName* é o nome de instância do WebSphere Application Server.

400 Ao tentar executar o comando `XMLConfig -import` , você pode receber a seguinte mensagem de erro: "Não é possível expandir o arquivo ear sob `/QIBM/UserData/WebAsAdv4/wasInstanceName/installedApps/WC_Enterprise_App_instanceName.ear`". Se você receber esta mensagem, remova ou renomeie o diretório anterior e execute o comando novamente.

3. 400 Depois de importar o aplicativo corporativo, você deve executar um script para modificar as permissões de diretório. Para executar esse script, proceda da seguinte forma:
   - No prompt de comandos, digite o seguinte:

```
STRSQH
cd /QIBM/ProdData/WebCommerce/bin
changeAuthority wasAdminInstanceName instanceName
```

em que

– *wasAdminInstanceName* é o nome da instância administrativa do WebSphere Application Server.

– *instanceName* é o nome da instância do WebSphere Commerce.

Iniciando um Aplicativo Corporativo

Esta seção descreve como utilizar o WebSphere Application Server Administration Console para atualizar a exibição e depois iniciar um aplicativo corporativo.

1. Abra o Administration Console do WebSphere Application Server.
2. Expanda o **Domínio Administrativo do WebSphere**, e os **Nós** e em seguida, o *nodeName*
3. Destaque o nó *nodeName*.
4. Clique no ícone **Atualizar subárvore selecionada**.
5. Inicie o aplicativo WebSphere Commerce da seguinte forma:
   - Expanda **Servidores de Aplicativos**.
   - Windows AIX Solaris Linux Clique com o botão direito do mouse no aplicativo WebSphere Commerce. Por exemplo, clique com o botão direito do mouse em **WebSphere Commerce Server - *instanceName*** e selecione **Iniciar**.
   - 400 Clique com o botão direito do mouse em seu aplicativo WebSphere Commerce. Por exemplo, clique com o botão direito do mouse no aplicativo *instanceName* **- WebSphere Commerce Server** e selecione **Iniciar**.

Apêndice C. Dicas para o VisualAge for Java

Essa seção descreve algumas dicas relacionadas à resolução de problemas, melhoras no desempenho e simplificação dentro do ambiente de desenvolvimento.

Alterando as Propriedades para o Mecanismo de Servlet no WebSphere Test Environment

As propriedades do mecanismo de servlet no WebSphere Test Environment são controladas pelo arquivo de propriedades `default.servlet_engine`. Dentro desse arquivo, você pode modificar a raiz do documento do servidor Web e pode alterar a porta utilizada pelo WebSphere Test Environment.

Convém alterar a porta, no caso do WebSphere Test Environment ter suspendido a função e a reinicialização não for uma opção. Para alterar a porta, proceda da seguinte forma:

1. Abra o arquivo *VAJ_install_path*`\IDE\ProjectResources\IBM WebSphere Test Environment\properties\default.servlet_engine` em um editor de texto.
2. Na sub-rotina `<transport>`, altere o valor `8080` na linha a seguir para uma porta disponível.

   ```
   <arg name="port" value="8080"/>
   ```

3. Altere o valor `8080` nas linhas a seguir para a mesma porta especificada acima.

   ```
   <hostname-binding hostname="localhost:8080" servlethost="default_host"/>
   <hostname-binding hostname="127.0.0.1:8080" servlethost="default_host"/>
   ```

4. Salve o arquivo.
5. Abra o WebSphere Test Environment Control Center e inicie o Mecanismo de Servlet.

Por padrão, a raiz do documento para o WebSphere Test Environment é *VAJ_install_path*`\IDE\ProjectResources\IBM WebSphere Test Environment\hosts\default_host\default_app\web`. Para mudar isto para outro diretório, proceda da seguinte maneira:

1. Abra o WebSphere Test Environment Control Center e pare o Mecanismo de Servlet.
2. Abra o arquivo *VAJ_install_path*`\IDE\ProjectResources\IBM WebSphere Test Environment\properties\default.servlet_engine` em um editor de texto.

3. Na sub-rotina <websphere-webgroup name="default_app">, altere <document-root>$approot$/web</document-root> para o seguinte:

   ```
   <document-root>your_document_root</document-root>
   ```

   em que *your_document_root* é a raiz de documento desejada.
4. Altere o valor 8080 nas linhas a seguir para a mesma porta especificada acima.

   ```
   <hostname-binding hostname="localhost:8080" servlethost="default_host"/>
   <hostname-binding hostname="127.0.0.1:8080" servlethost="default_host"/>
   ```
5. Salve o arquivo.
6. Abra o WebSphere Test Environment Control Center e inicie o Mecanismo de Servlet.

Resolvendo Problemas no Servidor de Nomes Persistente

Se você teve problemas com seu servidor de nomes persistente, convém eliminar e recriar o banco de dados do servidor de nomes persistente. Consulte o *Manual de Instalação do Commerce Studio* para obter detalhes sobre a criação desse banco de dados.

Alternativamente, se uma mensagem indicando que a porta está em uso atualmente, certifique-se de que o WebSphere Application Server não esteja em execução.

Excluindo os Arquivos JSP Compilados

O VisualAge for Java mantém uma pasta de projeto em que os arquivos JSP compilados são armazenados. Pode ocorrer momentos em que seja necessário excluir os arquivos JSP compilados. Por exemplo, se você remover o bean de dados de um arquivo JSP, poderá ocorrer erros na próxima vez que você chamar o arquivo JSP. Nesse caso, você pode excluir o arquivo JSP compilado.

Para excluir os arquivos JSP compilados, proceda da seguinte forma.

1. No VisualAge for Java, selecione a guia Projeto.
2. Desloque-se para o projeto **Gerar Código de Compilação da Página JSP**.
3. Selecione o projeto inteiro (se você excluir muitos arquivos JSP compilados) ou expanda o projeto e exclua apenas aqueles que devem ser recompilados.

Avisos

Estas informações foram desenvolvidas para produtos e serviços oferecidos nos Estados Unidos. É possível que a IBM não ofereça os produtos, serviços ou recursos discutidos neste documento em outros países. Consulte um representante IBM local para obter informações sobre produtos e serviços disponíveis atualmente em sua área. Qualquer referência a produtos, programas ou serviços IBM não significa que apenas produtos, programas ou serviços IBM possam ser utilizados. Qualquer produto, programa ou serviço funcionalmente equivalente, que não infrinja nenhum direito de propriedade intelectual da IBM ou outros direitos legalmente protegidos, poderá ser utilizado em substituição a este produto, programa ou serviço. Entretanto, a avaliação e verificação da operação de qualquer produto, programa ou serviço não-IBM são de responsabilidade do Cliente.

A IBM pode ter patentes ou solicitações de patentes pendentes relativas a assuntos tratados nesta publicação. O fornecimento desta publicação não garante ao Cliente nenhum direito sobre tais patentes. Pedidos de licença devem ser enviados, por escrito, para:

Gerência de Relações Comerciais e Industriais da IBM Brasil
Av. Pasteur, 138-146
Botafogo
Rio de Janeiro, RJ
CEP: 22290-240

Para pedidos de licença relacionados a informações de byte duplo (DBCS), entre em contato com o Departamento de Propriedade Intelectual da IBM em seu país ou envie pedidos de licença, por escrito, para:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

O parágrafo a seguir não se aplica a nenhum país em que tais disposições não estejam de acordo com a legislação local:

A INTERNATIONAL BUSINESS MACHINES CORPORATION FORNECE ESTA PUBLICAÇÃO ″NO ESTADO EM QUE SE ENCONTRA″, SEM GARANTIA DE NENHUM TIPO, SEJA EXPRESSA OU IMPLÍCITA, INCLUINDO, MAS NÃO SE LIMITANDO ÀS GARANTIAS IMPLÍCITAS DE NÃO-VIOLAÇÃO, MERCADO OU ADEQUAÇÃO A UM DETERMINADO

PROPÓSITO. Alguns países não permitem a exclusão de garantias expressas ou implícitas em certas transações; portanto, esta disposição pode não se aplicar ao Cliente.

Estas informações podem conter imprecisões técnicas ou erros tipográficos. Periodicamente, são feitas alterações nas informações aqui contidas; tais alterações serão incorporadas em futuras edições desta publicação. A IBM pode, a qualquer momento, aperfeiçoar e/ou alterar os produtos e/ou programas descritos nesta publicação, sem aviso prévio.

Referências nestas informações a sites não-IBM na Web são fornecidas apenas por conveniência e não representam de forma alguma um endosso a estes sites na Web. Os materiais contidos nestes sites da Web não fazem parte dos materiais deste produto IBM e a utilização destes sites da Web é de inteira responsabilidade do Cliente.

A IBM pode utilizar ou distribuir as informações fornecidas da forma que julgar apropriada sem incorrer em qualquer obrigação para com o Cliente.

Licenciados deste programa que desejam obter informações sobre este assunto com objetivo de permitir: (i) a troca de informações entre programas criados independentemente e outros programas (incluindo este) e (ii) a utilização mútua das informações trocadas, devem entrar em contato com:

Gerência de Relações Comerciais e Industriais da IBM Brasil
Av. Pasteur, 138-146
Botafogo
Rio de Janeiro, RJ
CEP: 22290-240

Tais informações podem estar disponíveis, sujeitas a termos e condições apropriados, incluindo em alguns casos, o pagamento de uma taxa.

O programa licenciado descrito neste documento e todo o material licenciado disponível são fornecidos pela IBM sob os termos do Contrato com o Cliente IBM, do Contrato de Licença do Programa Internacional IBM ou de qualquer outro contrato equivalente.

Todos os dados de desempenho aqui contidos foram determinados em um ambiente controlado. Portanto, os resultados obtidos em outros ambientes operacionais podem variar significativamente. Algumas medidas podem ter sido tomadas em sistemas de nível de desenvolvimento e não há garantia de que estas medidas serão iguais em sistemas geralmente disponíveis. Além disso, algumas medidas podem ter sido estimadas através de extrapolação. Os resultados reais podem variar. Os usuários deste documento devem verificar os dados aplicáveis para seu ambiente específico.

As informações relativas a produtos não-IBM foram obtidas junto aos fornecedores dos respectivos produtos, de seus anúncios publicados ou de outras fontes disponíveis publicamente. A IBM não testou estes produtos e não pode confirmar a precisão de seu desempenho, compatibilidade nem qualquer outra reivindicação relacionada a produtos não-IBM. Dúvidas sobre os recursos de produtos não-IBM devem ser encaminhadas diretamente a seus fornecedores.

Todas as declarações relacionadas aos objetivos e intenções futuras da IBM estão sujeitas a alterações ou cancelamento sem aviso prévio, e representam apenas metas e objetivos.

Todos os preços IBM mostrados são preços de varejo sugeridos pela IBM, são atuais e estão sujeitos a alteração sem aviso prévio. Os preços do revendedor podem variar.

Estas informações foram projetadas apenas com o propósito de planejamento. As informações aqui contidas estão sujeitas a alterações antes que os produtos descritos fiquem disponíveis.

Estas informações contêm exemplos de dados e relatórios utilizados nas operações diárias de negócios. Para ilustrá-los da forma mais completa possível, os exemplos podem incluir nomes de indivíduos, empresas, marcas e produtos. Todos estes nomes são fictícios e qualquer semelhança com nomes e endereços utilizados por uma empresa real é mera coincidência.

LICENÇA DE COPYRIGHT:

Estas informações contêm exemplos de programas aplicativos no idioma de origem, ilustrando as técnicas de programação em diversas plataformas operacionais. Você pode copiar, modificar e distribuir estes exemplos de programas sem a necessidade de pagar a IBM, com objetivos de desenvolvimento, utilização, marketing ou distribuição de programas aplicativos em conformidade com a interface de programação do aplicativo para a plataforma operacional para a qual os programas de exemplo são criados. Estes exemplos não foram testados completamente em todas as condições. Portanto, a IBM não pode garantir ou implicar a confiabilidade, manutenção ou função destes programas. Você pode copiar, modificar e distribuir estes exemplos de programas de qualquer maneira sem pagamento à IBM, com objetivos de desenvolvimento, utilização, marketing ou distribuição de programas aplicativos em conformidade com interfaces de programação de aplicativos da IBM.

Cada cópia ou parte deste exemplo de programas ou qualquer trabalho derivado, deve incluir um aviso de copyright com os dizeres:

Se você estiver exibindo estas informações em cópia eletrônica, as fotografias e ilustrações coloridas podem não aparecer.

Marcas e Marcas de Serviço

Os termos a seguir são marcas ou marcas registradas da International Business Machines Corporation nos Estados Unidos e/ou em outros países:

400	iSeries
AIX	MQSeries
AS/400	Net.Commerce
CICS	Net.Data
DB2	VisualAge
IBM	WebSphere

Windows e Windows NT são marcas ou marcas registradas da Microsoft Corporation nos Estados Unidos e/ou em outros países.

Oracle é uma marca registrada da Oracle Corporation nos Estados Unidos e/ou em outros países.

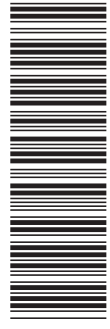
Solaris, Java e todas as marcas e logotipos baseados em Java são marcas ou marcas registradas da Sun Microsystems Inc. nos Estados Unidos e/ou em outros países.

Outros nomes de empresas, produtos e serviços podem ser marcas ou marcas de serviço de terceiros.

Índice Remissivo

Número da Peça: CT024BP

(1P) P/N: CT024BP



G517-7452-00

Spine information:

IBM WebSphere Commerce Manual do Programador Versão 5.4