iSeries

Cenário do WebSphere Application Server e Lotus Domino

Relatório da Experiência

iSeries

Cenário do WebSphere Application Server e Lotus Domino

Índice

Capítulo 1. Descrição geral do cenário do WebSphere(R) Application Server e Lotus(R) Domino(TM)

Este relatório documenta a experiência da equipa do Teste do Sistema do iSeries(TM) que consiste em juntar o WebSphere Application Server e o Lotus Domino como parte de uma única aplicação. Nesta aplicação, utilizamos o Lotus Domino para estabelecer uma presença inicial na Web, criar uma base de dados e utilizar os serviços de directório. O WebSphere Application Server é utilizado para criar as páginas Web dinâmicas, configurar o início de sessão único e garantir a segurança. Este procedimento foi efectuado através de várias fases como parte do cenário de viagens aéreas.

O cenário de viagens aéreas simula uma empresa de transporte aéreo que está a trabalhar afincadamente para aumentar a respectiva quota de mercado. A companhia aérea determinou que, para se manter competitiva, era necessário ter um sítio da Web no qual os clientes pudessem visualizar e reservar voos. Ansiosos por obter uma presença na Web, decidiram efectuar esta transição em várias fases que incluiu:

1. Obter uma presença inicial permitindo que os clientes visualizem os voos disponíveis
2. Aumentar a assistência a clientes permitindo que os clientes reservem voos online
3. Expandir-se para criar uma relação de empresa-empresa com uma agência de viagens

Cada uma destas fases foi construída sobre a anterior e incorporou as tecnologias mais recentes disponíveis. Abaixo encontram-se mais detalhes sobre as funções e tecnologias utilizadas em cada uma destas fases.

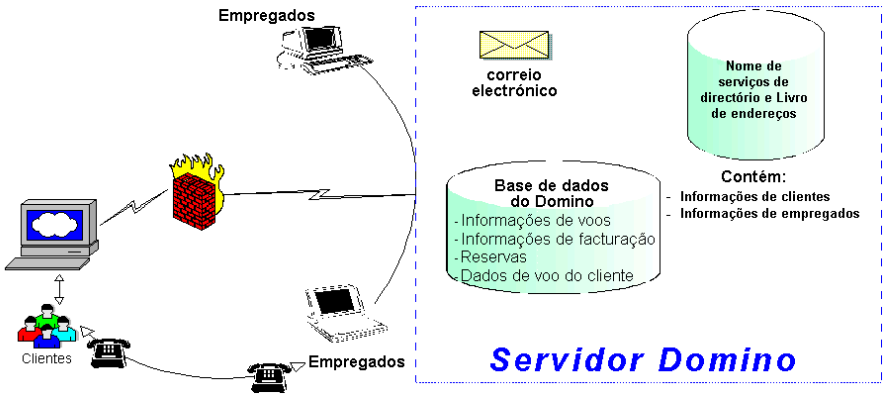
A primeira fase consistiu em criar uma presença na Web através das páginas do servidor Lotus Domino estáticas. Durante esta fase foi utilizado um servidor do Lotus Domino para uma base de dados, serviços de directório e servidor de HTTP do Lotus Domino.

A primeira fase permitiu aos clientes e empregados efectuar as seguintes funções:

- Clientes
  - Visualizar os voos disponíveis a partir do sítio da Web
  - Contactar um empregado da companhia aérea para reservar um voo
- Empregados
  - Adicionar e eliminar voos
  - Adicionar, actualizar e eliminar informações de clientes
  - Reservar voos
  - Obter informações de voo de clientes
  - Facturar aos clientes e actualizar informações de facturação

A Figura 1 ilustra a primeira fase.

Figura 1 - Primeira fase do cenário de viagens aéreas



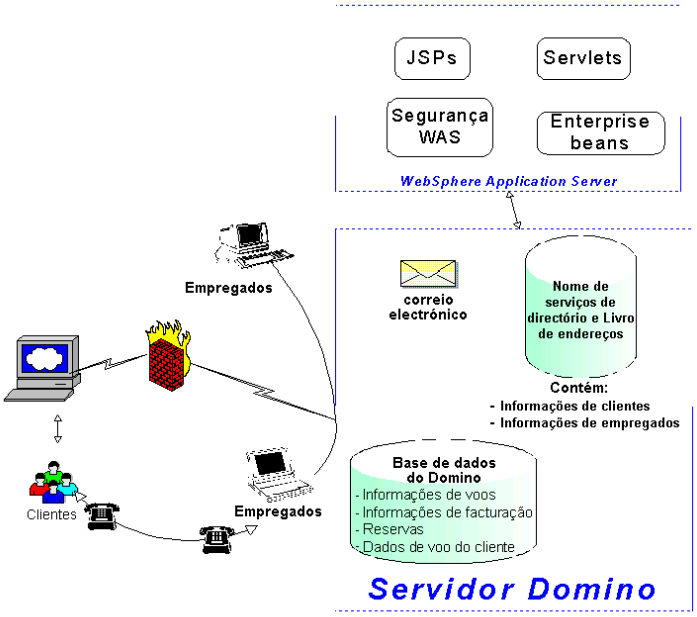
A segunda fase consistiu em utilizar o servidor do Lotus Domino, a base de dados e os serviços de directório existentes. Além disso, o WebSphere Application Server foi utilizado para fornecer sítios da Web dinâmicos que permitem aos clientes reservar voos. Os servlets, enterprise beans e JSPs foram utilizados para actualizar o cenário. O início de sessão único (SSO - single sign-on) do WebSphere Application Server e a segurança foram utilizados para concluir a aplicação.

Em resultado das alterações na segunda fase, foi possível aos clientes efectuar as funções acima listadas, bem como estas funções adicionais:

- Reservar voos disponíveis a partir do sítio da Web
- Obter informações de voo de clientes
- Actualizar informações de clientes
- Pagar contas online

A Figura 2 ilustra a segunda fase.

Figura 2 - Segunda fase do cenário de viagens aéreas



A fase três, que ainda não foi implementada, consistirá em criar uma relação de empresa-empresa com uma agência de viagens. Esta relação permitirá aos clientes reservar voos através da agência de viagens.

Para obter mais informações sobre os ambientes do Lotus Domino e do WebSphere Application Server ou a interoperacionalidade entre os dois, consulte as seguintes secções.

Capítulo 2, "Descrição geral do ambiente do Lotus(R) Domino(TM)", na página 5
Esta secção descreve o ambiente do Lotus Domino globalmente. Inclui uma descrição geral do ambiente, o fluxo de trabalho ao longo do ambiente e os nossos observações chave ao criar e utilizar este ambiente. Capítulo 3, "Descrição geral do ambiente do WebSphere(R) Application Server", na página 27
Esta secção descreve o ambiente do WebSphere Application Server globalmente. Inclui uma descrição geral do ambiente, o fluxo da aplicação através do ambiente e os nossos observações chave ao criar e utilizar este ambiente.

Capítulo 6, "Descrição geral da interoperacionalidade entre o WebSphere(R) Application Server e o Lotus(R) Domino(TM)", na página 85
Esta secção descreve a interoperacionalidade entre os ambientes do WebSphere Application Server e do Lotus Domino. Inclui uma descrição geral, detalhes sobre a configuração do início de sessão único e os nossos observações chave ao criar e utilizar o WebSphere Application Server e o Lotus Domino em conjunto.

Capítulo 2. Descrição geral do ambiente do Lotus(R) Domino(TM)

O ambiente do Lotus Domino foi configurado durante a primeira fase do trabalho no cenário de viagens aéreas do Teste do Sistema do iSeries(TM). Durante a primeira fase, o objectivo consistia em estabelecer rapidamente uma presença na Web de forma a que a companhia aérea pudesse expandir o respectivo negócio e fornecer uma solução que fosse fidedigna, disponível, escalável e pudesse integrar facilmente com outras aplicações.

Estes requisitos levaram a companhia aérea a escolher o Lotus Domino no eServer iSeries. Com o Lotus Domino, a companhia aérea conseguiu obter rapidamente uma presença na Web para os respectivos clientes e empregados fornecendo as páginas Web estáticas. Além disso, o Lotus Domino permitiu que a companhia aérea tirasse vantagem no processamento do fluxo de trabalho e fornecesse a interoperacionalidade com muitas outras plataformas.

A aplicação Lotus Domino da companhia aérea contém informações sobre clientes, voos, itinerários e facturação. A aplicação utiliza vários formulários e vistas, que criam e mantêm as informações. As informações sobre os voos consistem nos voos oferecidos incluindo as datas de regresso e partida, horários, cidades e número de lugares disponíveis por voo em primeira classe e em classe económica. Para gerir com segurança os dados de clientes e empregados, é utilizado um directório de Lightweight Directory Access Protocol (LDAP) do Lotus Domino.

A companhia aérea fornece duas interfaces da Web utilizando os conjuntos de estruturas do Lotus Domino. Existe uma interface para os clientes e uma para os empregados.

Os clientes dos voos podem executar as seguintes tarefas:

- Visualizar voos
- Contactar um empregado da companhia aérea para reservar um voo

Os empregados da companhia aérea podem executar as seguintes tarefas:

- Adicionar, apresentar e eliminar voos
- Adicionar e eliminar códigos de cidades
- Adicionar, actualizar, apresentar e eliminar informações de clientes
- Reservar voos
- Obter informações de voo de clientes
- Apresentar e actualizar informações de facturação

Durante a segunda fase, o WebSphere(R) Application Server foi utilizado para permitir mais funções de cliente incluindo a capacidade de reservar voos. Consulte o tópico ″Descrição geral do WebSphere Application Server″ para obter mais informações.

Fluxo de trabalho do ambiente do Lotus(R) Domino(TM)

Pontos de concepção da aplicação

Ao conceber a aplicação do Lotus Domino, a companhia aérea teve de escolher entre utilizar os conjuntos de estruturas ou navegadores para a respectiva interface da Web. Uma vez que a companhia aérea pretendia fornecer uma estrutura consistente em todo o sítio da Web que pudesse apresentar formulários e vistas diferentes e, já que as estruturas podem conter formulários, pastas, páginas, documentos e vistas, o conjunto de estruturas tornou-se a melhor opção.

Outra decisão com a qual se deparou companhia aérea foi encontrar e implementar a melhor forma de proteger os dados. Pretendiam um solução que fosse robusta o suficiente para fornecer a segurança necessária e tivesse flexibilidade para se integrar com outros pacotes de software. Estes requisitos levaram a companhia aérea a escolher o Lightweight Directory Access Protocol (LDAP) do Lotus Domino.

Uma vez que a companhia aérea já tinha vários empregados com competências em programação Java(TM), foi-lhes possível utilizar essas competências para escrever os agentes de segundo plano na aplicação em Java. Este facto provou ser valioso em termos de tirar o melhor partido das competências dos respectivos empregados.

Instalação da aplicação

Em todo o nosso trabalho de implementação do Lotus Domino, utilizámos o Lotus Domino Designer como ferramenta de desenvolvimento. O Lotus Domino Designer é uma aplicação cliente do Lotus Notes(R) utilizada para criar e modificar rapidamente uma aplicação do Lotus Domino. Fornece os blocos de construção da aplicação para tudo o que é necessário na base de dados, incluindo formulários, vistas e agentes. Utilizámos formulários para criar novos documentos na base de dados e apresentar os documentos actuais. As vistas fornecem uma forma flexível e intuitiva dos documentos serem organizados. Os utilizadores podem visualizar com facilidade listas de documentos, ordenar as listas de diferentes formas, abrir documentos para ler ou editar e criar novos documentos.

Formulários e vistas do Lotus Domino

Os empregados da companhia aérea criaram os formulários e vistas apresentados na Tabela 1 para a respectiva aplicação do Lotus Domino.

Tabela 1 - Formulários e vistas do Lotus Domino relativos a voos

Formulários do Lotus Domino	Vistas do Lotus Domino
Lista de voos disponíveis	Lista de voos disponíveis
Voos programados	Voos processados, Voos programados
Informações de lugares no avião	Informações de lugares no avião
Informações do bilhete	Bilhetes activos, Bilhetes inactivos, Bilhetes processados
Informações de crédito	Informações de crédito
Códigos de cidades	Códigos de cidades
Informações de facturação	Informações de facturação
Número de cliente	viewCustomerFldNumber
Número do voo	viewFlightFldNumber
Número da factura	viewInvoiceFldNumber

Número do lugar	viewSeatFldNumber
Número da instrução	viewStatementFldNumber

O formulário Lista de voos disponíveis contém as informações de voo para os voos disponíveis que são proporcionados pela companhia aérea. É utilizado para criar um novo voo ou apresentar um voo existente. Todos os dados específicos do voo estão inseridos neste formulário. As cidades onde este serviço é prestado encontram-se no formulário Códigos de cidades e o tipo de avião e os lugares disponíveis encontram-se no formulário Informações de lugares no avião.

A vista Lista de voos disponíveis apresenta os mais variados voos e é categorizada por AvailableFlightNumber (Número do voo disponível).

A Tabela 2 apresenta os campos no formulário Lista de voos disponíveis.

Tabela 2 - Formulário Lista de voos disponíveis

Nome	Nome do campo	Tipo	Descrição
Número do voo	AvailableFlightNumber	Texto	O número do voo, que pode ser utilizado em vários dias da semana mas não no mesmo dia
Hora de partida	DepartureTime	Data/hora	A hora a que o voo deixa a cidade de partida
Hora de chegada	ArrivalTime	Data/hora	A hora a que o voo deixa a cidade de destino
Duração	Duration	Número	A quantidade de horas entre a partida e chegada
Nome da transportadora aérea	AirlineName	Texto	O nome da transportadora aérea que fornece o voo
Código da cidade de partida	DisplayOnlyDeparture	Lista de diálogo com base nos códigos de cidades	O código de três letras da cidade de partida
Código da cidade de destino	DisplayOnlyArrival	Lista de diálogo com base nos códigos de cidades	O código de três letras da cidade de destino
AirplaneType	ListAirplaneType	Lista de diálogo	O tipo de avião
Refeições	Food	Selector de opção -> (pequeno-almoço, almoço, jantar, lanche, nada)	Especifica se serão fornecidos o pequeno-almoço, almoço, jantar, lanche ou nenhuma refeição (nada) durante o voo
Dias programados	ScheduledDays	Selector de confirmação -> Domingo, Segunda-feira, Terça-feira, Quarta-feira, Quinta-feira, Sexta-feira, Sábado	Os dias em que está programado efectuar o voo
Preço do bilhete de primeira classe	FirstClassPrice	Número	O preço de um bilhete de primeira classe

Preço do bilhete de classe económica	CoachPrice	Número	O preço de um bilhete de classe económica
Código da cidade de partida	DepartureCityCode	Texto	Oculto
Código da cidade de destino	ArrivalCityCode	Texto	Oculto
Apresentação do estado	DisplayStatus	Texto	Oculto

O formulário Voos programados contém as informações de voo para determinado voo numa data específica. É utilizado para reservar e controlar os lugares de um voo específico em determinada data. Todos os dados específicos do voo programado estão inseridos neste formulário.

A vista Voos programados apresenta os mais variados voos programados e é categorizada por Voo por data. A vista Voos processados apresenta todos os voos que chegaram ao destino ou que não têm lugares disponíveis para reservar.

A Tabela 3 apresenta os campos no formulário Voos programados.

Tabela 3 - Formulário Voos programados

Nome	Nome do campo	Tipo	Descrição
Voo por data	ScheduledFlightByDate	Texto	O ID a ser utilizado para controlar um voo específico em determinada data, criado pela junção do número do voo com a data do voo
Número do voo	ScheduledFlightNumber	Texto	O número do voo
AirplaneType	ScheduledAirplaneType	Lista de diálogo	O tipo de avião
Lugares de primeira classe disponíveis	FirstClassSeatsAvailable	Número	O número de lugares de primeira classe que ainda estão disponíveis no avião
Lugares de classe económica disponíveis	CoachSeatsAvailable	Número	O número de lugares de classe económica que ainda estão disponíveis no avião
Estado	Status	Selector de opção -> (Cancelado, chegada, partida, standby, novo)	O estado do voo
DepartureDate	ScheduledDepartureDate	Data	A data de partida do voo
ArrivalDate	ScheduledArrivalDate	Data	A data de chegada do voo, que pode ser diferente no caso de ser um voo efectuado durante a noite ou atravessar a linha de data internacional

O formulário Informações de lugares no avião contém as informações específicas do avião. É utilizado para listar o tipo de avião e número total de lugares disponíveis.

A vista Informações de lugares no avião apresenta os mais variados aviões e é categorizado por AirplaneType.

A Tabela 4 apresenta os campos no formulário Informações de lugares no avião.

Tabela 4 - Formulário Informações de lugares no avião

Nome	Nome do campo	Tipo	Descrição
AirplaneType	SeatAirplaneType	Texto	O tipo de avião
Lugares de primeira classe	FirstClassSeats	Número	O número de lugares de primeira classe que existem no avião
Lugares de classe económica	CoachSeats	Número	O número de lugares de classe económica que existem no avião
Todas as informações sobre o avião	AllAirplaneInfo	Texto	Oculto

O formulário Informações do bilhete contém as informações sobre um bilhete. Podem existir um ou mais bilhetes incluídos numa factura.

A vista Bilhetes activos apresenta os bilhetes que ainda não foram processados. Uma vez processadas estas reservas, são removidas desta vista e apresentadas na vista Bilhetes processados ou Bilhetes inactivos, dependendo do bilhete ter sido ou não processado com êxito.

A vista Bilhetes inactivos apresenta os bilhetes que não foram processados. Um exemplo dos bilhetes que não foi possível processar corresponde às reservas que foram colocadas quando já não havia bilhetes disponíveis para o voo ou quando tenha sido cancelado um voo.

A vista Bilhetes processados apresenta todos os bilhetes completos.

A Tabela 5 apresenta os campos no formulário Informações do bilhete.

Tabela 5 - Formulário Informações do bilhete

Nome	Nome do campo	Tipo	Descrição
Número da factura	InvoiceNumber	Texto	Número utilizado para associar os bilhetes
Número do voo	TicketFlightNumber	Texto	Associa ao documento da lista de voos disponíveis

Classe do bilhete	TicketClass	Lista de diálogo	A classe do bilhete é: primeira classe ou económica
Número do lugar	SeatNumber	Texto	O número do lugar neste voo
Preço do bilhete	TicketPrice	Número	O preço do bilhete respeitante àquele lugar
Estado do pagamento	PaidStatus	Selector de opção	O estado do pagamento
Número de cliente	TicketCustomerNumber	Texto	O número do cliente que paga o bilhete
Estado do bilhete	TicketStatus	Lista de diálogo	O estado do bilhete: activo, inactivo ou processado
Nome próprio do passageiro	PassengerFirstName	Texto	O nome do cliente que se senta no lugar atribuído. Pode ser diferente da pessoa que compra o bilhete e, por esse motivo, o número de cliente
Outros nomes do passageiro	PassengerMiddleName	Texto	Outros nomes do passageiro
Apelido do passageiro	PassengerLastName	Texto	O apelido do passageiro
Morada do passageiro	PassengerStreet	Texto	A morada do passageiro
Cidade do passageiro	PassengerCity	Texto	A cidade do passageiro
Distrito do passageiro	PassengerState	Texto	O distrito do passageiro
Código postal do passageiro	PassengerZip	Texto	O código postal da morada do passageiro
País do passageiro	PassengerCountry	Texto	O país do passageiro
Telefone do passageiro	PassengerPhone	Texto	O número de telefone do passageiro
Voo por data	FlightByDate	Texto	O número do voo por data que corresponde ao voo deste bilhete

O formulário Informações de crédito contém as informações de crédito do cliente conforme identificado pelo número de cliente.

A vista Informações de crédito apresenta as informações do cartão de crédito utilizadas pelo cliente para pagar o respectivo voo e é categorizado pelo Número de cliente.

A Tabela 6 apresenta os campos no formulário Informações de crédito.

Tabela 6 - Formulário Informações de crédito

Nome	Nome do campo	Tipo	Descrição
Tipo do cartão	CardType	Lista de diálogo (Mastercard, Visa, Diner's Club)	O tipo de cartão de crédito

Número do cartão de crédito	CreditCardNumber	Texto	O número do cartão
Data de expiração	ExpirationDate	Data / hora	A data de expiração do cartão
Número de cliente	CreditCustomerNumber	Texto	O número de cliente associado ao cartão
Número da factura	InvoiceNumber	Texto	Número utilizado para associar os bilhetes

O formulário Códigos de cidades contém a lista de cidades suportadas e os correspondentes códigos de cidades com três letras.

A vista Códigos de cidades apresenta os mais variados códigos de cidades e é categorizado pelo código.

A Tabela 7 apresenta os campos no formulário Códigos de cidades.

Tabela 7 - Formulário Códigos de cidades

Nome	Nome do campo	Tipo	Descrição
Cidade	City	Texto	A cidade
Distrito	State	Texto	O distrito ou concelho
País	Country	Texto	O país
Código	Code	Texto	O código de três letras que representa a cidade especificada
Todas as informações sobre o código da cidade	AllCityCodeInfo	Texto	Oculto
Guardar opções	SaveOptions	Número	Oculto, definido como 0 de forma a que o documento não seja guardado

O formulário Informações de facturação contém informações para a facturação dos clientes.

A vista Informações de facturação apresenta as informações do bilhete utilizadas para facturar o cliente pelo respectivo voo e é categorizado pelo Número de cliente.

A Tabela 8 apresenta os campos no formulário Informações de facturação.

Tabela 8 - Formulário Informações de facturação

Nome	Nome do campo	Tipo	Descrição
Preço na factura	BillAmount	Número	A quantia que o cliente deve

Número de cliente	BillCustomer	Texto	O número do cliente
Informações sobre o bilhete na factura	BillTicketInfo	Texto	O bilhete que está associado à factura

As vistas viewCustomerFldNumber, viewFlightFldNumber, viewInvoiceFldNumber, viewSeatFldNumber e viewStatementFldNumber estão ocultas e apresentam o número de cliente, número do voo, número da factura, número do lugar e documentos com números de instruções. Quando é criado um novo documento, o evento PostOpen utiliza o valor actual no documento para criar um número único. Uma vez criado o número, o evento PostOpen aumenta o valor e armazena-o no documento.

Para além dos formulários e vistas criados na respectiva base de dados da aplicação, a companhia aérea também adicionou o seguinte formulário e vista ao livro de nomes e endereços no servidor do Lotus Domino, apresentado na Tabela 9.

Tabela 9 - Formulários e vistas do Lotus Domino

Formulários do Lotus Domino	Vistas do Lotus Domino
Passageiro	Clientes dos voos

O formulário Passageiro contém o nome, informações sobre o endereço e o número de cliente de cada um dos clientes dos voos. Este formulário efectuado pela companhia aérea baseou-se no já existente formulário Pessoa a partir da concepção do livro de endereços e modificou-o para melhor se adequar às suas necessidades.

A vista Clientes dos voos apresenta as informações do cliente e é categorizado pelo Número de cliente.

A Tabela 10 apresenta os campos no formulário Passageiro.

Tabela 10 - Formulário Passageiro

Nome	Nome do campo	Tipo	Descrição
Nome próprio	FirstName	Texto	O nome próprio do cliente
Iniciais dos outros nomes	MiddleInitial	Texto	As iniciais dos outros nomes do cliente
Apelido	LastName	Texto	O apelido do cliente
Endereço de Internet	InternetAddress	Texto	O endereço de Internet do cliente
Palavra-passe da Internet	HTTPPassword	Texto	A palavra-passe do cliente para acesso ao sítio da Web da companhia aérea
Morada	StreetAddress	Texto	A morada pessoal do cliente
Cidade	City	Texto	A cidade do cliente
Distrito/concelho	State	Texto	O distrito/concelho do cliente

Código postal	Zip	Texto	O código postal do cliente
País	Country	Texto	O país do cliente
Telefone da residência	PhoneNumber	Texto	O número de telefone da residência do cliente
Número de cliente	PersonalID	Número	O número do cliente

Agentes do Lotus Domino

Os agentes do Lotus Domino são elementos de concepção adicionados a uma base de dados do Lotus Domino para automatizar tarefas. Os agentes podem ser iniciados por uma acção do utilizador numa base de programação. Normalmente, os agentes são utilizados para actualizar ou criar documentos, aceder a dados a partir da base de dados do Lotus Domino ou de outras origens. Os agentes do Lotus Domino podem ser escritos em Java, LotusScript ou Formula Language.

A criação de agentes requer o Lotus Domino Designer. Quando cria um agente, especifica quando pretende que seja executado, qual o código de idioma em que será escrito e sob que documentos é executado. Após ter escrito o código e efectuado a respectiva compilação, é programado automaticamente para executar na hora previamente especificada. Quando estiver a escrever o código, pode aproveitar as capacidades de depuração incorporadas do Lotus Domino Designer para ajudá-lo a depurar o código.

O que se segue fornece informações detalhadas sobre os agentes do Lotus Domino escritos para a aplicação de viagens aéreas do Lotus Domino:

- Verificar código da cidade único
  Este agente do LotusScript recebe um código da cidade como entrada de dados a partir de um navegador da Web utilizando o formulário Códigos de cidades. Este agente utiliza essas informações juntamente com as informações dos documentos Código da cidade existentes (criados a partir do formulário Códigos de cidades) para pesquisar em todos estes documentos. Se for encontrada uma correspondência, é apresentada uma mensagem de erro, o utilizador é notificado e é-lhe permitido criar um documento Código da cidade diferente. Se não for encontrada qualquer correspondência, o documento Código da cidade será criado.
- Criar número de cliente, número de voo e número de factura aleatórios - apenas Web
  Estes agentes do LotusScript geram um número de cliente, número de voo ou número de factura únicos para novos clientes, voos ou facturas adicionados através da interface da Web do Lotus Domino.
- Botão Eliminar de códigos de cidades e lista de voos disponíveis
  Este agente Java é executado com base no utilizador fazer clique no botão de acção Eliminar nas vistas Código da cidade ou Voo. Marca o documento Código da cidade ou Voo como eliminado.
- Eliminar códigos da cidade seleccionados
  Este agente Java é executado com base numa programação. Elimina qualquer código da cidade que tenha o estado Eliminar no campo CityCodeStatus dos documentos Códigos de cidades. Também elimina os documentos Lista de voos disponíveis, Voos programados e Bilhetes que correspondam ao código da cidade.
- Eliminar voos seleccionados
  Este agente Java é executado com base numa programação. Elimina qualquer voo que tenha o estado Eliminar no campo DisplayStatus dos documentos Lista de voos disponíveis. Também elimina os documentos Voos programados e Bilhetes que correspondam ao número do voo.

- Eliminar bilhete - Actualizar lugar disponível
  Este agente Java é executado com base no utilizador fazer clique num botão de acção. Aumenta o número de lugares disponíveis com base nas selecções em FlightByDate e Classe do bilhete a partir do documento Voo programado.
- Gerar factura
  Este agente Java é executado com base no utilizador fazer clique num botão de acção. Gera informações de facturação do cliente com base no número de cliente seleccionado.
- Gerar preço e número do lugar
  Este agente Java é executado com base no utilizador fazer clique num botão de acção. Gera informações de preço e calcula o lugar disponível seguinte para determinado voo e a classe do lugar com base nas selecções em FlightByDate e Classe do bilhete.
- Preencher voos programados
  Este agente Java é executado com base numa programação. Cria os documentos Voos programados com informações reunidas a partir dos documentos Lista de voos disponíveis e Informações de lugares no avião em que a apresentação do estado é "Novo". Cada voo será preenchido durante um mês a partir da data de hoje.
- Estado do voo programado
  Este agente Java é executado com base numa programação. Altera o estado do voo de novo para partida e de partida para chegada com base na obtenção da Data e hora do voo a partir dos documentos Lista de voos disponíveis e Voos programados.
- Actualizar voos programados
  Este agente Java é executado com base numa programação. Verifica o estado dos voos e se encontrar qualquer voo que tenha partido desde a execução anterior do agente, actualiza os bilhetes associados ao voo movendo-os para a vista Bilhetes processados.
- Actualizar estado do bilhete
  Este agente Java é executado com base numa programação. Altera o estado dos bilhetes de activos para processados ou inactivos com base na obtenção da Data e hora do voo a partir dos documentos Lista de voos disponíveis e Voos programados. Um bilhete torna-se inactivo se for cancelado um voo. Os bilhetes processados são aqueles que partiram.

O código de origem dos agentes Gerar preço e número do lugar e Verificar código da cidade único é apresentado na secção "Exemplo: Código de origem dos agentes do Lotus Domino" na página 17.

Detalhes da aplicação

A partir da interface da Web do Lotus Domino, os empregados da companhia aérea podem executar várias tarefas. Estas tarefas pertencem às quatro categorias principais: cliente, voos, reservas de voos e facturação. Cada uma destas categorias contém acções que o empregado pode executar. Estas acções são as seguintes:

- Cliente
  Um emprego da companhia aérea pode adicionar um cliente pela introdução do nome do cliente, palavra-passe, endereço de Internet e morada. As informações introduzidas juntamente com o número de cliente gerado são armazenadas no directório LDAP do Lotus Domino. Assim que é criado, este documento novo pode ser actualizado, eliminado ou apresentado.
- Voos
  - Um empregado da companhia aérea pode adicionar um voo introduzindo a hora de partida e chegada, duração do voo, código da cidade de partida e destino, tipo de avião, tipo de refeição disponível, dias programados do voo e preço da classe económica e primeira classe. As informações introduzidas são armazenadas no formulário Lista de voos disponíveis. Assim que é criado, este documento novo pode ser apresentado ou eliminado.
  - Um empregado da companhia aérea também pode adicionar um código da cidade introduzindo a cidade, distrito, país e código da cidade. As informações introduzidas são armazenadas no

formulário Códigos de cidades após o código da cidade ser verificado para se certificar de que é único. Se for único, o documento é criado. Uma vez criado, o documento pode ser apresentado ou eliminado.

- Reservas de voos
  Um empregado da companhia aérea pode reservar um voo introduzindo o voo por data programado, classe do bilhete (classe económica/primeira classe), preço gerado, número do lugar gerado, estado do pagamento, número de cliente e informações do passageiro. As informações introduzidas são armazenadas no formulário Bilhete. Uma vez criado, o documento pode ser apresentado nos seguinte estados: activo, processado ou inactivo.
- Facturação
  Um empregado da companhia aérea pode gerar uma factura de um bilhete introduzindo o tipo de cartão de crédito, número, data de expiração e o número de cliente. As informações introduzidas são armazenadas no formulário Informações de crédito. Uma vez criado, o documento pode ser apresentado ou actualizado.

Observações chave do ambiente do Lotus(R) Domino(TM)

A seguir encontra-se uma lista de observações chave que se revelaram ao criar e utilizar o cenário de viagens aéreas do ambiente do Lotus Domino.

- Em vários documentos de voos, utilizámos um agente para gerar um número único. Este agente foi designado pelo evento WebQueryOpen. O agente gera o número quando um novo documento é criado, mas o número gerado não foi guardado quando o documento foi guardado. O seguinte excerto do texto da ajuda do Lotus Domino Designer explica como o evento do WebQueryOpen funciona e o motivo pelo qual este valor não foi guardado:

  "Os agentes do WebQueryOpen são executados quando o utilizador abre um formulário ou documento, mas não são executados quando o utilizador guarda um documento. O que significa que os campos computados definidos por um agente do WebQueryOpen não são guardados quando o utilizador submete um documento. Para se certificar de que os campos computados são guardados, pode calculá-los novamente no agente do WebQuerySave ou definir a propriedade 'Generate HTML for all fields' (Gerar HTML para todos os campos) de formulário."

  Após seleccionar a propriedade 'Generate HTML for all fields' de formulário, o número calculado foi guardado juntamente com todos os outros campos no documento.

- Para apresentar uma vista após submeter um documento através da Web, em vez de apresentar o texto assumido: "Form processed" (Formulário processado), efectue os seguintes procedimentos:
  - Crie um campo de texto oculto no formulário que foi computado para apresentação
  - Atribua o nome de $$Return ao campo
  - Utilize uma fórmula semelhante à seguinte no valor do campo:
    "[/"+@Subset(@DbName;-1)+"/NomeDaSuaVista?OpenView&DblClkTarget=_self]"

    Se o nome da vista tiver espaços, utilize '+' em vez de um espaço (i.e. Nome+Da+Sua+Vista).
    Se o nome da vista for categorizado com outras vistas, utilize duas barras invertidas (\\) (i.e. SuaVista\\NomeDaVista).
- Verificou-se um desempenho fraco e outros comportamentos estranhos quando tínhamos a propriedade 'Use applet in the browser' (Utilizar o applet no navegador) de vista seleccionada em For Web Access (Para acesso à Web) no separador avançado das propriedades de vista.
- No formulário Passageiro, a propriedade 'Generate HTML for all fields' (Gerar HTML para todos os campos) de formulário precisava de ser seleccionada para permitir que os dados fossem mantidos no formulário ao introduzir dados na Web. Sem a selecção desta propriedade, quando clicar no separador

seguinte no formulário de cliente, os dados nos outros separadores não são mantidos. Com esta propriedade seleccionada, os dados são mantidos à medida que cada um dos separadores é seleccionado.

- Pretendíamos apenas permitir que os códigos de cidades únicos fossem guardados ao criar um novo código da cidade único a partir da Web.

  O seguinte excerto do texto da ajuda do Lotus Domino Designer explica como o evento do WebQuerySave funciona ao verificar valores únicos:

  "A simulação de programas CGI que executam em dados fornecidos pelo utilizador através da programação de um evento do WebQuerySave e da adição, ao formulário, de um campo SaveOptions com um valor '0'. Quando o agente é executado, pode coleccionar valores de campos a partir do formulário preenchido sem gerar um novo documento do Notes(TM)."

  Ao ter o campo SaveOptions definido como 0 impedirá o documento de ser guardado. Assim que o valor for definido como 1, o documento é guardado. Não há necessidade de executar uma verificação if no campo SaveOptions no botão Submeter. O Lotus Domino processa a acção de guardar com base no valor do campo SaveOptions.

  Por exemplo, eis a solução que utilizámos:

  1. No formulário, crie um botão Submeter e também um campo oculto designado SaveOptions com um valor inicial de 0
  2. No botão Submeter, codifique a seguinte fórmula:
     @Command([FileSave]);
     @Command([FileCloseWindow])
  3. No evento WebQuerySave, introduza a fórmula que executa um agente
  4. No agente, adicione o seguinte código:
     If (foundCityCode) Then
     Print "<SCRIPT LANGUAGE=JavaScript(TM)>"
     Print "alert(""Foi localizado Código da cidade duplicado. Introduza um novo código da cidade."")"
     Print "location.href = ""../../" + ficheiro + "/City+Codes?OpenForm"""
     Print "</SCRIPT>"
     Else
     '// A seguinte linha definirá o campo SaveOptions no documento como 1, o que fará com que o Notes guarde o documento.
     Set item = doc.ReplaceItemValue("SaveOptions", 1)
     End If

Referências

- Lotus Domino Release 5.0: A Developer's Handbook, IBM Redbook(R) SG24-5331-01
- Sítio da Web IBM Lotus Domino for iSeries(TM) - OS/400(R)
  http://www.ibm.com/servers/eserver/iseries/domino/
- IBM Lotus Domino for iSeries (PartnerWorld(R) para programadores)
  http://www.as400.ibm.com/developer/domino/
- Sítio da Web Lotus
  http://www.lotus.com

Exemplo: Código de origem dos agentes do Lotus Domino

Esta secção contém o código de origem de dois dos agentes do Lotus (R) Domino (TM) da aplicação de viagens aéreas. O primeiro exemplo é o agente Gerar preço e número do lugar que é escrito em Java(TM). Este agente gera as informações sobre preços e calcula o lugar seguinte disponível. O segundo exemplo é o agente Verificar código da cidade único que é escrito em LotusScript. Este agente verifica se o código da cidade é único.

Exemplo 1: Agente Gerar preço e número do lugar

```
////////////////////////////////////////////////////////////////////
/*
* Este agente Java é executado com base no utilizador fazer clique num botão de acção.
* Gera informações de preço e calcula o lugar disponível seguinte
* com base nas selecções em FlightByDate e Classe do bilhete.
*
* Nome do cenário: Voos TFC
*
* Versão Java: JDK 1.1.8
*/
////////////////////////////////////////////////////////////////////


import lotus.domino.*;


public class JavaAgent extends AgentBase {


public void NotesMain() {


System.out.println("Iniciar: Agente Gerar preço e número do lugar.");


try
{
Session session = getSession();
AgentContext agentContext = session.getAgentContext();


generateSeat(session, agentContext);


System.out.println("Concluir: Agente Gerar preço e número do lugar.");
}
catch(Exception e) {
e.printStackTrace();
}
```

```
} // end Notes(TM) main


public void generateSeat(Session session, AgentContext agentContext)
{
// obter dados de Voo por data e classe do bilhete a partir do documento actual
boolean isFirstClass = false; //definir inicialmente este sinalizador como falso (i.e. económica)



try
{
Database db = agentContext.getCurrentDatabase();
DocumentCollection collection = agentContext.getUnprocessedDocuments();
if (collection.getCount() < 1)
{
// ocorreu um problema no acesso ao documento actual, sair do agente
System.out.println("Erro no documento actual, finalização do agente.");
return;
} // end if
else
{
Document doc = collection.getFirstDocument();
String flightByDate = doc.getItemValueString("FlightByDate");
String ticketClass = doc.getItemValueString("TicketClass");
DocumentCollection scheduledFlightDC = db.search("SELECT ((Form = \"Voos programados\")) &
((@Contains(ScheduledFlightByDate; \"" + flightByDate + "\")))");
if (scheduledFlightDC.getCount() < 1)
{
// voo por data especificado inválido - não foi encontrada correspondência com o voo por data
System.out.println("Voo por data especificado inválido - não foi encontrada correspondência com o voo
por data, finalização do agente.");
return;
} // end if
else
{
int seatsRemaining = 0;
int seatNumber = 0;
Document scheduledFlightDoc = scheduledFlightDC.getFirstDocument();



if (ticketClass.equals("Primeira classe"))
{
isFirstClass = true;



seatsRemaining = scheduledFlightDoc.getItemValueInteger("FirstClassSeatsAvailable");
```

```
//verificar se ainda existe um lugar disponível
if (seatsRemaining >= 1)
{
// obter o número seguinte do lugar disponível
seatNumber = getNextAvailableSeat(db, isFirstClass,
scheduledFlightDoc.getItemValueString("ScheduledAirplaneType"));
int nextAvailableSeat = seatNumber - seatsRemaining + 1;



String strObj = String.valueOf(nextAvailableSeat);
doc.replaceItemValue("SeatNumber", strObj);



int price = getPrice(flightByDate, session, agentContext, isFirstClass); //obter informações do preço deste
bilhete
Integer priceInt = new Integer(price);
doc.replaceItemValue("TicketPrice", priceInt);
doc.save(true, true);



// está disponível um lugar
Integer intObject = new Integer(seatsRemaining-1); //número de decréscimo dos lugares disponíveis
scheduledFlightDoc.replaceItemValue("FirstClassSeatsAvailable", intObject);
scheduledFlightDoc.save(true, true);
} //end if seats remaining >= 1



else
{ //if seats remaining = 0, confirmar se foram eliminados alguns bilhetes
String deletedSeats = scheduledFlightDoc.getItemValueString("DeletedFirstClassSeats");
if (deletedSeats != null)
{ // existe um lugar disponível que foi cancelado, reservar o lugar
int index = deletedSeats.indexOf(";");
String availSeat = deletedSeats.substring(0, index);
Integer intObject = Integer.valueOf(availSeat);
// actualizar os lugares eliminados, remover o lugar que acabou de ser reservado e deixar os outros tal
como estão
scheduledFlightDoc.replaceItemValue("DeletedFirstClassSeats", deletedSeats.substring(index + 1));
scheduledFlightDoc.save(true, true);



String strObj = String.valueOf(availSeat);
doc.replaceItemValue("SeatNumber", strObj);



int price = getPrice(flightByDate, session, agentContext, isFirstClass); //obter informações do preço deste
bilhete
Integer priceInt = new Integer(price);
doc.replaceItemValue("TicketPrice", priceInt);
doc.save(true, true);
```

```
} // end if deletedSeats != null
else
{
// o voo está completo
System.out.println (“<SCRIPT LANGUAGE=JavaScript(TM)>”);
System.out.println (“alert(\”Não existem mais lugares disponíveis neste voo. Seleccione um novo
voo.\“)”);
System.out.println (“</SCRIPT>”);
} // end else



} // end seats remaining = 0



} // end if first class
else
{ // o bilhete é de classe económica
seatsRemaining = scheduledFlightDoc.getItemValueInteger(“CoachSeatsAvailable”);



//verificar se ainda existe um lugar disponível
if (seatsRemaining >= 1)
{
// obter o número seguinte do lugar disponível
seatNumber = getNextAvailableSeat(db, isFirstClass,
scheduledFlightDoc.getItemValueString(“ScheduledAirplaneType”));
int nextAvailableSeat = seatNumber - seatsRemaining + 1;



String strObj = String.valueOf(nextAvailableSeat);
doc.replaceItemValue(“SeatNumber”, strObj);



int price = getPrice(flightByDate, session, agentContext, isFirstClass); //obter informações do preço deste
bilhete
Integer priceInt = new Integer(price);
doc.replaceItemValue(“TicketPrice”, priceInt);
doc.save(true, true);



// está disponível um lugar
Integer intObject = new Integer(seatsRemaining-1); //número de decréscimo dos lugares disponíveis
scheduledFlightDoc.replaceItemValue(“CoachSeatsAvailable”, intObject);
scheduledFlightDoc.save(true, true);
} //end if seats remaining >= 1
```

```
else
{ //if seats remaining = 0, confirmar se foram eliminados alguns bilhetes
String deletedSeats = scheduledFlightDoc.getItemValueString("DeletedCoachSeats");
if (deletedSeats != null)
{ // existe um lugar disponível que foi cancelado, reservar o lugar
int index = deletedSeats.indexOf(";");
String availSeat = deletedSeats.substring(0, index);
Integer intObject = Integer.valueOf(availSeat);
// actualizar os lugares eliminados, remover o lugar que acabou de ser reservado e deixar os outros tal
como estão
scheduledFlightDoc.replaceItemValue("DeletedCoachSeats", deletedSeats.substring(index + 1));
scheduledFlightDoc.save(true, true);



String strObj = String.valueOf(availSeat);
doc.replaceItemValue("SeatNumber", strObj);



int price = getPrice(flightByDate, session, agentContext, isFirstClass); //obter informações do preço deste
bilhete
Integer priceInt = new Integer(price);
doc.replaceItemValue("TicketPrice", priceInt);
doc.save(true, true);



} // end if deletedSeats != null
else
{
// o voo está completo
System.out.println ("<SCRIPT LANGUAGE=JavaScript>");
System.out.println ("alert(\"Não existem mais lugares disponíveis neste voo. Seleccione um novo
voo.\")");
System.out.println ("</SCRIPT>");
}
} // end seats remaining = 0
} // end else ticket is coach
}
} // end else
} // end try



catch (NotesException ne) {
ne.printStackTrace();
}
catch (Exception e) {
e.printStackTrace();
}



} // end method generateSeat
```

```
public int getNextAvailableSeat(Database db, boolean isFirstClass, String airplaneType)
{
// avançar para a vista Informações de lugares no avião e localizar o tipo de avião correspondente
int seatNum = 0;
try
{
DocumentCollection seatDC = db.search("SELECT ((Form = \"Informações de lugares no avião\")) &
((@Contains(SeatAirplaneType; \"" + airplaneType + "\")))");
if (seatDC.getCount() < 1)
{
// não foi encontrada correspondência com o tipo de avião
System.out.println("modelo do avião especificado inválido - não foi encontrada correspondência com o
registo, finalização do agente.");
return 0;
} // end if
else
{
Document seatDoc = seatDC.getFirstDocument();
if (isFirstClass)
{
seatNum = seatDoc.getItemValueInteger("FirstClassSeats");
}
else
{
seatNum = seatDoc.getItemValueInteger("CoachSeats");
}



} // end else
} // end try
catch (NotesException ne) {
ne.printStackTrace();
}
catch (Exception e) {
e.printStackTrace();
}



return seatNum;
} // end method getNextAvailableSeat



public int getPrice(String flightByDate, Session session, AgentContext agentContext, boolean isFirstClass)
{
int price = 0;
try
{
// calcular número do voo
int index = flightByDate.lastIndexOf("_");
String flightNumber = flightByDate.substring(0, index);
```

```
Database db = agentContext.getCurrentDatabase();
DocumentCollection flightDC = db.search("SELECT ((Form = \"Lista de voos disponíveis\")) &
((@Contains(AvailableFlightNumber; \"" + flightNumber + "\")))");



if (flightDC.getCount() < 1)
{
// ocorreu um problema no acesso ao documento actual, sair do agente
System.out.println("Erro no documento actual, finalização do agente.");
return 0;
} // end if
else
{
Document doc = flightDC.getFirstDocument();
if (isFirstClass)
{
price = doc.getItemValueInteger("FirstClassPrice");
}
else
{
price = doc.getItemValueInteger("CoachPrice");
}



} // end else



} // end try
catch (NotesException ne) {
ne.printStackTrace();
}
catch (Exception e) {
e.printStackTrace();
}



return price;
} // end method generateSeat
} // end class
```

Exemplo 2: Agente Verificar código da cidade único

```
Sub Initialize
```

```
'//====================================== '// Este agente do Domino recebe um Código da
cidade como entrada de dados a partir de um navegador da Web utilizando o formulário Códigos de
cidades.
'// Este agente utiliza essas informações juntamente com as informações dos documentos Código da
cidade existentes (criados a partir do formulário Códigos de cidades).
'// Este agente utiliza a vista 'Códigos de cidades' para pesquisar através de todos os documentos
Código da cidade. Se for encontrada uma correspondência, o utilizador é
'// notificado e é-lhe permitido criar um novo documento Código da cidade. Se não for encontrada
qualquer correspondência, o documento Código da cidade será criado.
'//======================================


'// Definir variante de foundCityCode
Dim foundCityCode As Variant
foundCityCode = False


'// Criar uma sessão de notas
Dim session As New NotesSession


Messagebox "Agente em execução " & session.CurrentAgent.Name & " na base de dados " &
session.CurrentDatabase.Title & " as " & session.CommonUserName


'// Abrir esta base de dados
Dim db As NotesDatabase
Set db = session.CurrentDatabase
If Not ( db.IsOpen) Then
Messagebox "Base de dados " & session.CurrentDatabase.Title & " não foi aberta. Finalização do agente."
Exit Sub
End If


'// Definir documento igual aos valores transitórios da sessão actual (i.e. variáveis de código do
documento Código da cidade)
Dim doc As NotesDocument
Set doc = session.DocumentContext
Dim unid As String
Dim file As String
file = db.FileName


'// Definir a vista como Códigos de cidades, para localizar um documento Códigos de cidades
Dim CCview As NotesView
Dim CCdoc As NotesDocument
Set CCview = db.GetView("Códigos de cidades")
Set CCdoc = CCview.GetFirstDocument
```

```
'// Pesquisar através dos documentos Código da cidade até que seja encontrada uma correspondência ou não
While ((Not (CCdoc Is Nothing)) And (Not foundCityCode))
If (CCdoc.Code(0) = doc.Code(0)) Then
unid = CCdoc.UniversalID
foundCityCode = True
Else
Set CCdoc = CCview.GetNextDocument (CCdoc)
End If
Wend


'// Se for encontrada correspondência, apresentar uma mensagem e permitir ao utilizador criar um novo documento Código da cidade
'// Se for encontrada correspondência, permitir ao utilizador criar um novo documento Código da cidade
If (foundCityCode) Then
Print "<SCRIPT LANGUAGE=JavaScript>"
Print "alert(""Foi localizado Código da cidade duplicado. Introduza um novo código da cidade."")"
Print "location.href = ""../../" + ficheiro + "/City+Codes?OpenForm"""
Print "</SCRIPT>"
Else
On Error Goto Errhandle
'// Call doc.Save(True, True)
'// Print "<SCRIPT LANGUAGE=JavaScript>"
'// Print "location.href = ""../../" + ficheiro + "/City+Codes/doc?SaveDocument"""
'// Print "alert(""A guardar."")"
'// Print "location.href = ""../../" + ficheiro + "/City+Codes?OpenView"""
'// Print "</SCRIPT>"
Set item = doc.ReplaceItemValue("SaveOptions", 1)
End If


Errhandle:
' Utilizar a função Err para devolver o número do erro e
' a função Error$ para devolver a mensagem de erro.
Messagebox "Error" & Str(Err) & ": " & Error$
Exit Sub


End Sub
```

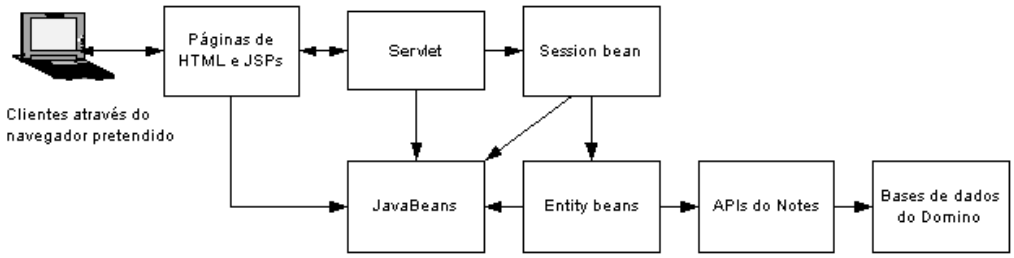
Capítulo 3. Descrição geral do ambiente do WebSphere(R) Application Server

O ambiente do WebSphere Application Server foi configurado durante a segunda fase de trabalho no cenário de viagens aéreas de Teste do Sistema do iSeries(TM). Durante a segunda fase, o objectivo da companhia aérea consistia em permitir que os cliente reservassem voos a partir do respectivo sítio da Web. Os clientes decidiram criar esta interface utilizando páginas HTML, JSPs (Java(TM) Server Pages), JavaBeans(TM), enterprise beans e servlets. Esta interface permitiu aos clientes visualizarem e reservarem voos online, obter informações sobre o voo, actualizar as informações de cliente e efectuar pagamentos online.

Modelos da aplicação

A aplicação de viagens aéreas utiliza o modelo de aplicação representado na Figura 1. Neste modelo, um navegador acede a JSPs (Java Server Pages) indirectamente através de um servlet que interage com a lógica comercial utilizando enterprise beans. Os enterprise beans extraem as informações de que necessitam da base de dados do Lotus(R) Domino(TM). Após receber o pedido do cliente, o servlet executa todos os cálculos necessários e cria JavaBeans. O JSP é invocado utilizando os JavaBeans adequados. O JSP extrai as informações de que necessita a partir dos JavaBeans e intercala as mesmas com a página HTML. Em seguida, o navegador interpreta e apresenta o HTML.

Figura 1 Modelo de aplicação de viagens aéreas



Capítulo 4. Fluxo do processo da aplicação

Existem várias páginas HTML e JSP utilizadas na aplicação de viagens aéreas. A Tabela 1 contém a lista de páginas HTML e a Tabela 2 contém a lista de JSPs. O fluxo da aplicação é apresentado na Figura 2.

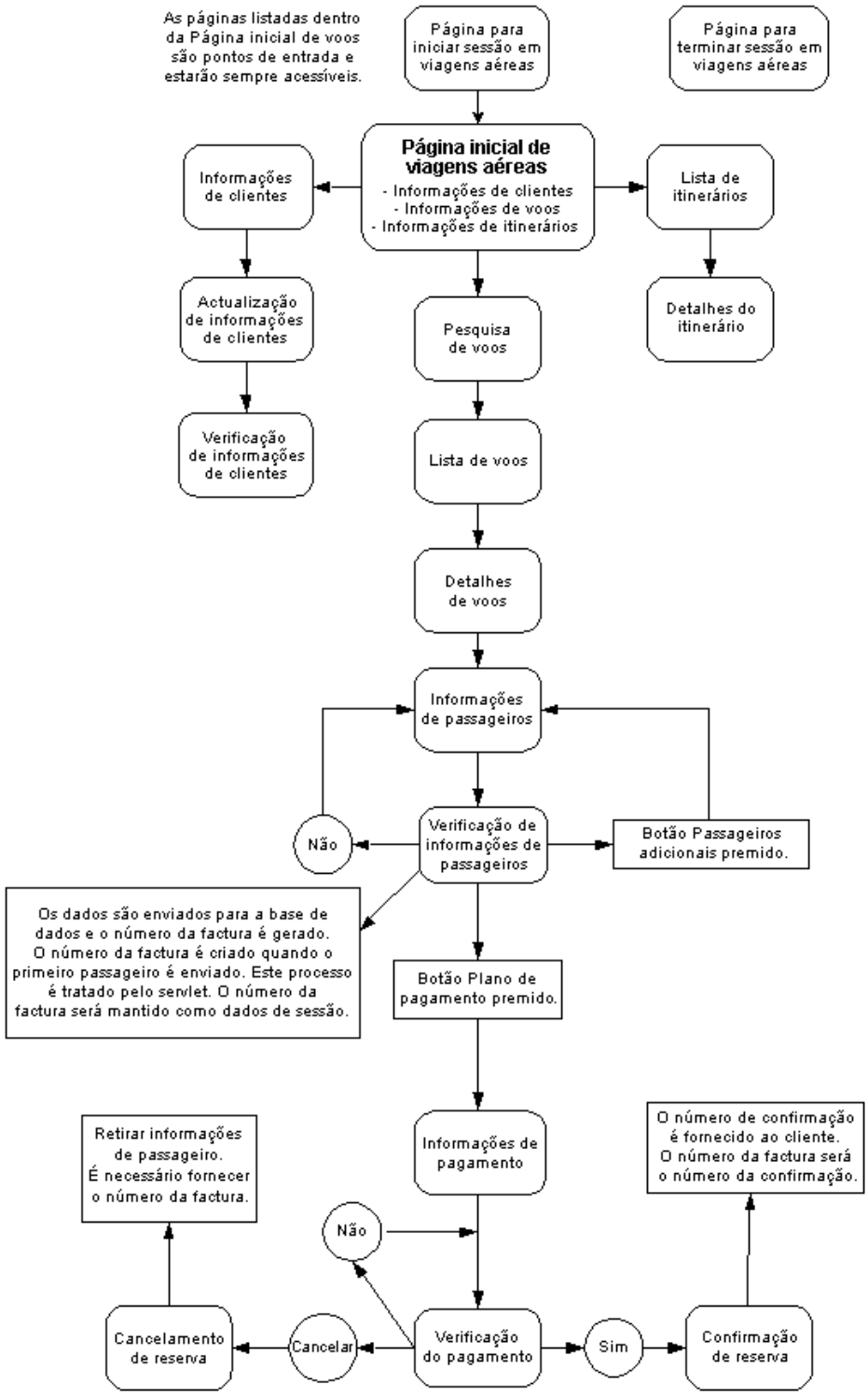
Tabela 1 Páginas HTML da aplicação de viagens aéreas

Página HTML	Funções
homePage	Permite ao cliente optar entre a apresentação das seguintes informações: • voo • cliente • itinerário
index	Apresenta as estruturas de menu, parte superior e página inicial
logOn	Permite ao cliente iniciar sessão
menu	Fornece opções de menu no lado esquerdo
top	Disponibiliza a parte superior da página que contém o logótipo

Tabela 2 JSPs da aplicação de viagens aéreas

JSP	Funções
customerInformation	Apresenta as informações sobre o cliente
logOff	Permite ao cliente terminar sessão
customerInformationUpdate	Permite ao cliente actualizar as respectivas informações
customerInformationConfirmation	Permite ao cliente verificar as respectivas informações actualizadas
flightsSearch	Fornece uma lista de critérios para efectuar a pesquisa de voos
flightsList	Fornece uma lista de voos que correspondem aos critérios de pesquisa
flightsDetails	Fornece informações de voo detalhadas sobre um voo específico
passengerInformation	Permite ao cliente introduzir as informações sobre os passageiros
passengerInformationVerification	Permite ao cliente verificar as informações sobre os passageiros
paymentInformation	Permite ao cliente introduzir as informações sobre o pagamento
paymentInformationVerification	Permite ao cliente verificar as informações sobre o pagamento
reservationCancel	Permite ao cliente cancelar a reserva
bookConfirmation	Confirma a reserva
itineraryList	Fornece a lista de itinerários ao cliente
itineraryDetails	Fornece informações detalhadas sobre o itinerário relativamente a um itinerário específico

Figura 2 Fluxo da aplicação de viagens aéreas



Ambiente de desenvolvimento

Foram utilizados os seguintes produtos durante o desenvolvimento desta aplicação:

- WebSphere Studio
- Visual Age for Java Enterprise Edition
- WebSphere Studio Application Developer

Os JSPs foram inicialmente desenvolvidos utilizando o WebSphere Studio e os enterprise beans foram desenvolvidos utilizando o Visual Age for Java. Na fase intermédia do desenvolvimento, foi efectuada a migração para o WebSphere Application Server versão 4.0. Esta alteração implicou mover as acções de desenvolvimento dos JSPs e dos enterprise beans para o WebSphere Studio Application Developer.

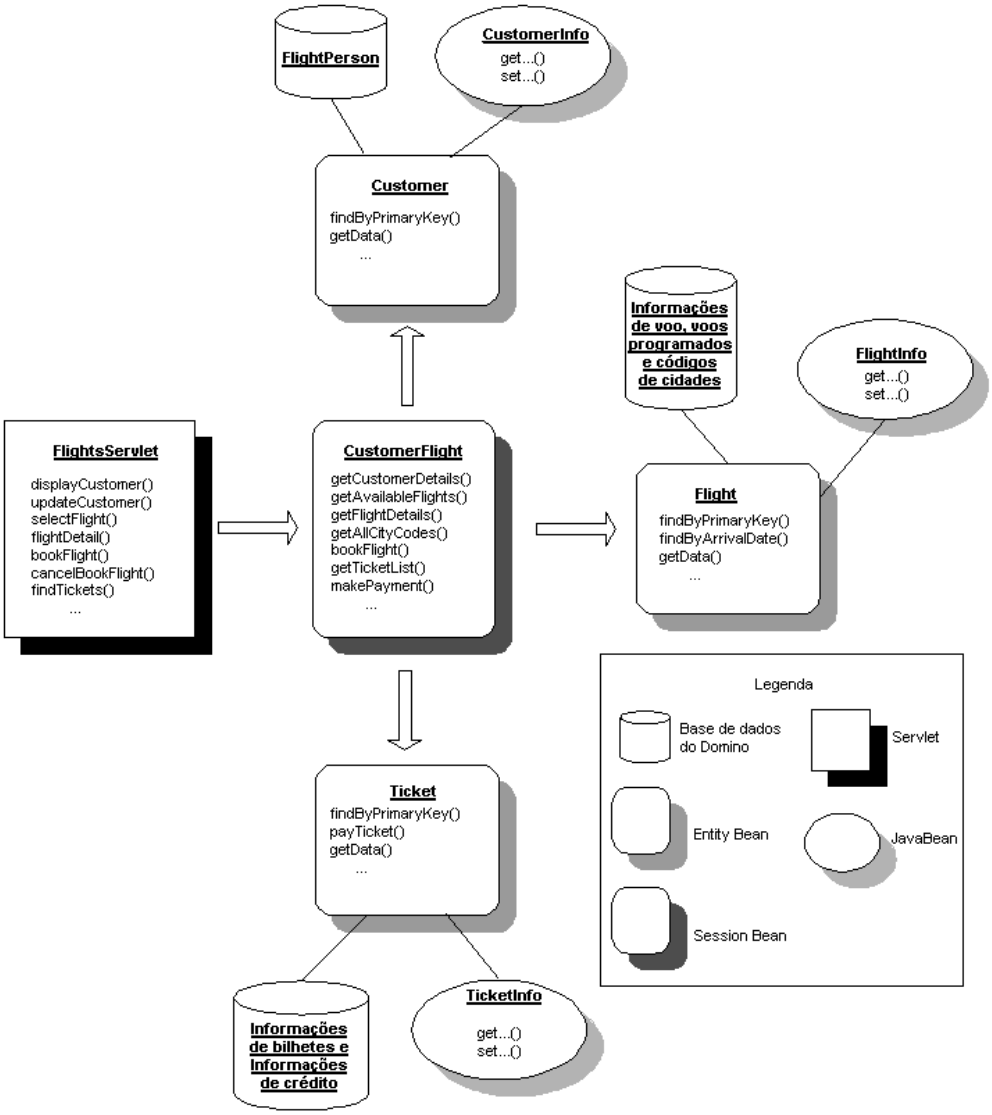
Fluxo de aplicações de ambiente do WebSphere(R) Application Server

Detalhes da aplicação

Esta secção fornece detalhes do modelo da aplicação de viagens aéreas, incluindo a utilização de servlets, JavaBeans(TM), enterprise beans e JSPs. Existe um servlet (FlightsServlet) que actua como controlador principal. Todos os pedidos são enviados para este servlet, que, em seguida, executa a tarefa pedida. O servlet acede a um session bean (CustomerFlight) para executar a maior parte do respectivo trabalho. O session bean CustomerFlight utiliza entity beans (Customer, Flight e Ticket) para realizar a tarefa pretendida.

A figura 1 ilustra a relação entre o servlet e os enterprise beans.

Figura 1 Relação da aplicação



Considerações de concepção

Inicialmente pretendia-se utilizar um Controlador JDBC do Lotus(R) Domino(TM) nos enterprise beans para aceder às bases de dados do Lotus Domino. No entanto, no WebSphere Application Server, não é possível criar uma origem dos dados que utilize o controlador JDBC do Lotus Domino no iSeries(TM). Uma vez que o iSeries ainda não transferiu o controlador JDBC do Lotus Domino, também não foi possível implementar um agrupamento de ligações próprio. A disponibilidade do controlador JDBC do Lotus Domino está agendada para uma edição futura. Devido às limitações listadas abaixo, as APIs do Lotus são utilizadas para aceder à base de dados do Lotus Domino nos entity beans.

Durante a designação da aplicação, ficou decidido utilizar entity beans com persistência gerida por bean. É necessário utilizar entity beans com persistência gerida por bean, porque, caso contrário, os entity beans utilizariam as APIs do Lotus Domino para aceder às bases de dados do Lotus Domino.

Servlet de viagens aéreas

A aplicação de viagens aéreas utiliza o FlightsServlet para controlar o respectivo fluxo. O FlightsServlet é utilizado para fornecer aos clientes de Viagens Aéreas a capacidade para visualizar e actualizar dados do cliente, visualizar dados do voo programado e reservar voos utilizando um navegador. As páginas JSP chamadas por este servlet são utilizadas para a apresentação de dados. Executam uma quantidade mínima de trabalho de processamento. A maior parte do trabalho de processamento é efectuada pelo FlightsServlet através dos seguintes métodos:

- doPost
  - Cria uma sessão entre o servidor da Web e o navegador que efectua o pedido.
  - Examina todos os pedidos e encaminha-os para o método apropriado no servlet com base no valor de jspRequest.
- doGet
  - Cria uma sessão entre o servidor da Web e o navegador que efectua o pedido.
  - Define o valor de jspRequest conformemente e chama o método doPost().
- errorHandle
  - Trata de quaisquer excepções encontradas pelo servlet ou enterprise beans.
  - Fornece uma mensagem ao utilizador sobre o erro encontrado.
- init
  - É chamado pelo servidor imediatamente depois de o mesmo criar a instância do servlet.
  - Cria e consulta o objecto da página inicial do CustomerFlight.
- bookFlight
  - Utiliza o enterprise bean CustomerFlight para reservar um voo para o cliente especificado.
  - Apresenta passengerInformation.jsp.
- cancelBookFlight
  - Utiliza o enterprise bean CustomerFlight para cancelar um voo reservado para um cliente específico.
  - Apresenta reservationCancel.jsp.
- confirmFlight
  - Utiliza o enterprise bean CustomerFlight para confirmar e efectuar o pagamento de um voo reservado.
  - Apresenta bookConfirmation.jsp.
- displayCustomer
  - Utiliza o enterprise bean CustomerFlight para obter as informações do cliente especificado.
  - Apresenta customerInformation.jsp.
- displayJSP
  - Chama uma página JSP. Todos os dados que o JSP necessita estarão presentes na sessão.
- findTickets
  - Utiliza o enterprise bean CustomerFlight para obter uma lista de bilhetes.
  - Apresenta itineraryList.jsp.
- flightDetail
  - Utiliza o enterprise bean CustomerFlight para obter as informações detalhadas do voo.
  - Apresenta flightsDetails.jsp.
- flightList
  - Utiliza o enterprise bean CustomerFlight para obter uma lista de todos os códigos de cidades.
  - Apresenta flightSearch.jsp.
- getCustomerFlight
  - Cria um objecto CustomerFlightHome consultando a classe CustomerFlightHome.

- getInitialContext
  - Obtém um contexto inicial para o URL especificado.
- itineraryDetails
  - Utiliza o enterprise bean CustomerFlight para obter os detalhes do itinerário e do voo com base no número do itinerário e voo por data.
  - Apresenta itineraryDetails.jsp.
- selectFlight
  - Utiliza o enterprise bean CustomerFlight para obter uma lista de todos os voos com base nos critérios da procura fornecidos.
  - Apresenta flightsList.jsp.
- updateCustomer
  - Utiliza o enterprise bean CustomerFlight para actualizar os dados do cliente.
  - Apresenta customerInfoConfirmation.jsp.

Enterprise beans

Na aplicação de viagens aéreas, o FlightsServlet actua como o controlador principal. Todos os pedidos são enviados para este servlet que, por sua vez, executa a tarefa pedida. O servlet acede a um session bean (CustomerFlight) para executar a maior parte do respectivo trabalho. O session bean CustomerFlight utiliza entity beans (Customer, Flight e Ticket) para realizar a tarefa pretendida.

Enterprise bean CustomerFlight

O enterprise bean CustomerFlight é um session bean sem registo. Fornece métodos para realizar tarefas que seriam executadas por um cliente, como por exemplo:

- Visualizar ou actualizar informações do cliente
- Obter uma lista de voos disponíveis
- Visualizar detalhes do itinerário
- Reservar um voo
- Efectuar um pagamento de um voo reservado

Os métodos no session bean CustomerFlight são descritos aqui:

- O método bookFlight() devolve um objecto TicketInfo que contém as informações do bilhete que foi reservado. São executadas as seguintes tarefas:
  1. Verifica se foi especificado um número da factura. Se não tiver sido especificado, criará um novo bilhete com um novo número da factura. Se o número da factura for especificado, criará um novo bilhete utilizando o número da factura especificado. Os bilhetes são criados utilizando o enterprise bean Ticket.
  2. O método getData() é invocado no enterprise bean Ticket e os dados são armazenados num objecto TicketInfo.
  3. O objecto TicketInfo é devolvido.
- O método cancelBookFlight() cancela os bilhetes para o número da factura especificado. São executadas as seguintes tarefas:
  1. Utiliza o método findByInvoiceNumber() no enterprise bean Ticket para obter uma lista de bilhetes com o número da factura especificado.
  2. A lista de bilhetes é então processada e o método remove() é chamado em cada Bilhete para remover os documentos da base de dados do Lotus Domino.
- O método getAllCityCodes() obtém uma lista de todos os códigos de cidades disponíveis no documento Códigos de cidades. São executadas as seguintes tarefas:

1. Selecciona todos os códigos de cidades a partir do formulário Códigos de cidades e as informações são armazenadas num objecto FlightInfo.
2. O objecto FlightInfo é devolvido.

- O método getAvailableFlights() devolve um objecto FlightInfo que contém uma lista de todos os voos que correspondem aos critérios da procura especificados. São executadas as seguintes tarefas:
  1. Os valores de parâmetro são verificados e é executada um localização com base nos critérios da procura utilizando o enterprise bean Flight.
  2. O método getData() no enterprise bean Flight é invocado e os dados são armazenados num objecto FlightInfo.
  3. O objecto FlightInfo é devolvido.
- O método getConnection() devolve uma ligação à base de dados do Lotus Domino utilizando as APIs do Lotus Domino. São executadas as seguintes tarefas:
  1. As variáveis de ambiente são utilizadas para criar a ligação.
  2. É criada uma nova sessão para base de dados do Lotus Domino.
  3. O objecto da base de dados é criado e devolvido.
- O método getCustomerDetails() devolve um objecto CustomerInfo que contém as informações do cliente para o número de cliente especificado. São executadas as seguintes tarefas:
  1. É executada uma localização por chave primária com base no número de cliente utilizando o enterprise bean Customer.
  2. O método getData() é invocado no enterprise bean Customer e os dados são devolvidos num objecto CustomerInfo.
- O método getFlightDetails() devolve um objecto FlightInfo que contém as informações do voo para o voo especificado por data. São executadas as seguintes tarefas:
  1. É executada uma localização por chave primária com base no voo por data utilizando o enterprise bean Flight.
  2. O método getData() é invocado no enterprise bean Flight e os dados são devolvidos num objecto FlightInfo.
- O método getInitialContext() devolve o contexto inicial para criar os entity beans.
- O método getTicketList() devolve um objecto TicketInfo que contém uma lista de todos os bilhetes com base num número de cliente ou num número da factura. São executadas as seguintes tarefas:
  1. É executada uma localização por número de cliente ou por número da factura com base no parâmetro introduzido utilizando o enterprise bean Ticket.
  2. O método getData() é invocado no enterprise bean Ticket e os dados são armazenados num vector.
  3. O vector é armazenado num objecto TicketInfo.
  4. O objecto TicketInfo é devolvido.
- O método makePayment() devolve um objecto TicketInfo. Este método é utilizado para efectuar um pagamento de um voo reservado. Actualizará todos os bilhetes com o mesmo número da factura. São executadas as seguintes tarefas:
  1. É executada uma localização por número da factura com base no número da factura utilizando o enterprise bean Ticket.
  2. O método payTicket() é invocado no enterprise bean Ticket.
  3. O método getData() é invocado no enterprise bean Ticket e os dados são devolvidos num objecto TicketInfo.
- O método setCustomerHome() cria um objecto CustomerHome consultando a classe CustomerHome.
- O método setFlightHome() cria um objecto FlightHome consultando a classe FlightHome.
- O método setTicketHome() cria um objecto TicketHome consultando a classe TicketHome.
- O método updateCustomer() actualiza as informações pessoais de um cliente. São executadas as seguintes tarefas:

1. É executada uma localização por chave primária com base no número de cliente utilizando o enterprise bean Customer.
2. São invocados os seguintes métodos no enterprise bean Customer:
   - Método setCustomerFirstName()
   - Método setCustomerMiddleInitial()
   - Método setCustomerLastName()
   - Método setCustomerAddress()
   - Método setCustomerCity()
   - Método setCustomerState()
   - Método setCustomerZipCode()
   - Método setCustomerCountry()
   - Método setCustomerPhoneNumber()
   - Método setCustomerInernetAddress()
   - Método getData() que armazena os dados num objecto CustomerInfo
3. O objecto CustomerInfo é devolvido.

Bean Customer

O enterprise bean Customer é um entity bean utilizado para representar um cliente de viagens aéreas. Fornece métodos que realizam tarefas executadas em dados do cliente, como por exemplo:

- Visualização das informações do cliente
- Actualização das informações do cliente

O enterprise bean Customer utiliza persistência gerida por bean e está mapeado ao formulário FlightPerson na base de dados names.nsf do Lotus Domino. O esquema do formulário FlightPerson é mostrado na Tabela 10 na secção "Fluxo de trabalho do ambiente do Lotus(R) Domino(TM)" na página 6. O enterprise bean Customer contém um método ejbFindBy. Este método de localizador é definido na interface da página inicial. A Tabela 1 contém o método de localizador.

Tabela 1 Métodos de localizador do enterprise bean Customer

Nome do método	Instrução Select	Descrição
ejbFindByPrimaryKey()	"SELECT (Form = \"FlightPerson\") & (PersonalID = " + customerNumber.trim() +")"	Localiza clientes com base no número de cliente

Os métodos getter e setter, bem como os valores que devolvem ou definem, estão listados na Tabela 2.

Tabela 2 Métodos Getter e Setter do cliente

Getter	Setter	Nome do campo no formulário	Tipo de dados
getCustomerAddress()	setCustomerAddress()	FlightPerson: StreetAddress	Cadeia
getCustomerCity()	setCustomerCity()	FlightPerson: City	Cadeia
getCustomerCountry()	setCustomerCountry()	FlightPerson: Country	Cadeia
getCustomerFirstName()	setCustomerFirstName()	FlightPerson: FirstName	Cadeia
getCustomerInternetAddress()	setCustomerInternetAddress()	FlightPerson: InternetAddress	Cadeia
getCustomerLastName()	setCustomerLastName()	FlightPerson: LastName	Cadeia
getCustomerMiddleInitial()	setCustomerMiddleInitial()	FlightPerson: MiddleInitial	Cadeia

getCustomerNumber()	setCustomerNumber()	FlightPerson: PersonalID	Cadeia
getCustomerPhoneNumber()	setCustomerPhoneNumber()	FlightPerson: PhoneNumber	Cadeia
getCustomerState()	setCustomerState()	FlightPerson: State	Cadeia
getCustomerZipCode()	setCustomerZipCode()	FlightPerson: Zip	Cadeia
getData()		JavaBean CustomerInfo com os valores no enterprise bean	CustomerInfo

O método ejbLoad() é utilizado para obter os dados de um documento FlightPerson específico e colocá-los nas propriedades do bean.

O método ejbStore() é utilizado para actualizar dados num documento FlightPerson específico a partir das propriedades do bean.

CustomerInfo

O CustomerInfo é um JavaBean utilizado para armazenar as informações do cliente. É introduzido no JSP apropriado que, por sua vez, o utiliza para obter os dados específicos do cliente.

Os métodos getter e setter, bem como os valores que devolvem ou definem, estão listados na Tabela 3.

Tabela 3 Métodos Getter e Setter de CustomerInfo

Getter	Setter	Valor	Tipo de dados
getCustomerAddress()	setCustomerAddress()	Endereço do cliente	Cadeia
getCustomerCity()	setCustomerCity()	Cidade do cliente	Cadeia
getCustomerCountry()	setCustomerCountry()	País do cliente	Cadeia
getCustomerFirstName()	setCustomerFirstName()	Nome próprio do cliente	Cadeia
getCustomerInternetAddress()	setCustomerInternetAddress()	Endereço de Internet do cliente	Cadeia
getCustomerLastName()	setCustomerLastName()	Apelido do cliente	Cadeia
getCustomerMiddleInitial()	setCustomerMiddleInitial()	Outras iniciais do cliente	Cadeia
getCustomerPhoneNumber()	setCustomerPhoneNumber()	Número de telefone do cliente	Cadeia
getCustomerState()	setCustomerState()	Distrito do cliente	Cadeia
getCustomerZipCode()	setCustomerZipCode()	Código postal do cliente	Cadeia

O código de origem utilizado nos beans Customer e CustomerInfo é apresentado na secção “Exemplo: Bean Customer” na página 63.

Bean Flight

O enterprise bean Flight é um entity bean utilizado para representar um voo. Fornece métodos que realizam tarefas executadas em dados de voos, como por exemplo:

- Obter uma lista de voos disponíveis com base em critérios da procura específicos
- Obter informações específicas de voos com base no valor de voo por data

O enterprise bean Flight utiliza persistência gerida por bean e está mapeado aos formulários Lista de voos disponíveis e Voos programados na base de dados flights.nsf do Lotus Domino. O esquema do

formulário Lista de voos disponíveis é apresentado na Tabela 2 e o formulárioVoos programados é apresentado na Tabela 3 da secção "Fluxo de trabalho do ambiente do Lotus(R) Domino(TM)" na página 6.

O enterprise bean Flight contém vários métodos ejbFindBy. Estes métodos de localizador são definidos na interface da página inicial. A Tabela 4 contém os métodos de localizador, em que o nome do método é precedido por ejbFindBy.

Tabela 4 Métodos de localizador de Flight

Nome do método	Instrução Select	Descrição
ArrivalCity()	"SELECT (Form = \"Lista de voos disponíveis\") & (ArrivalCityCode = \"" + aArrivalCity + "\")" "SELECT (Form = \"Voos programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (Status = \"New\") & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))"	Localiza voos com base na cidade de destino
ArrivalCityArrivalDate()	"SELECT (Form = \"Lista de voos disponíveis\") & (ArrivalCityCode = \"" + aArrivalCity + "\")" "SELECT (Form = \"Voss programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))"	Localiza voos com base na cidade de destino e na data de regresso
ArrivalCityArrivalDate DepartureCity()	"SELECT (Form = \"Lista de voos disponíveis\") & (ArrivalCityCode = \"" + aArrivalCity + "\") & (DepartureCityCode = \"" + aDepartureCity + "\")" "SELECT (Form = \"Voos programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))"	Localiza voos com base na cidade de destino, na data de regresso e na cidade de partida

ArrivalCityArrivalDate DepartureCityDepartureDate()	"SELECT (Form = \"Lista de voos disponíveis\") & (ArrivalCityCode = \"" + aArrivalCity + "\") & (DepartureCityCode = \"" + aDepartureCity + "\")" "SELECT (Form = \"Voos programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))"	Localiza voos com base na cidade de destino, na data de regresso, na cidade de partida e na data de partida
ArrivalCityArrivalDate DepartureCitySeatType()	"SELECT (Form = \"Lista de voos disponíveis\") & (ArrivalCityCode = \"" + aArrivalCity + "\") & (DepartureCityCode = \"" + aDepartureCity + "\")" • Classe económica "SELECT (Form = \"Voos programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")" • Primeira classe "SELECT (Form = \"Voos programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")"	Localiza voos com base na cidade de destino, na data de regresso, na cidade de partida, na data de partida e no tipo de lugar
ArrivalCityArrivalDate DepartureDate()	"SELECT (Form = \"Lista de voos disponíveis\") & (ArrivalCityCode = \"" + aArrivalCity + "\")" "SELECT (Form = \"Voos programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))"	Localiza voos com base na cidade de destino, na data de regresso e na data de partida

ArrivalCityArrivalDate DepartureDateSeatType()	"SELECT (Form = \"Lista de voos disponíveis\") & (ArrivalCityCode = \"" + aArrivalCity + "\")" • Classe económica "SELECT (Form = \"Voos programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")" • Primeira classe "SELECT (Form = \"Voos programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")"	Localiza voos com base na cidade de destino, na data de regresso, na data de partida e no tipo de lugar
ArrivalCityArrivalDate SeatType()	"SELECT (Form = \"Lista de voos disponíveis\") & (ArrivalCityCode = \"" + aArrivalCity + "\")" • Classe económica "SELECT (Form = \"Voos programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")" • Primeira classe "SELECT (Form = \"Voos programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")"	Localiza voos com base na cidade de destino, na data de regresso e no tipo de lugar

ArrivalCityDepartureCity()	“SELECT (Form = \”Lista de voos disponíveis\“) & (ArrivalCityCode = \”“ + aArrivalCity + ”\“) & (DepartureCityCode = \”“ + aDepartureCity + ”\“)” “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (Status = \”New\“) & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))”	Localiza voos com base na cidade de destino e na cidade de partida
ArrivalCityDepartureCity DepartureDate()	“SELECT (Form = \”Lista de voos disponíveis\“) & (ArrivalCityCode = \”“ + aArrivalCity + ”\“) & (DepartureCityCode = \”“ + aDepartureCity + ”\“)” “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (Status = \”New\“) & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))”	Localiza voos com base na cidade de destino, na cidade de partida e na data de partida
ArrivalCityDepartureCity DepartureDateSeatType()	“SELECT (Form = \”Lista de voos disponíveis\“) & (ArrivalCityCode = \”“ + aArrivalCity + ”\“) & (DepartureCityCode = \”“ + aDepartureCity + ”\“)” • Classe económica “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (CoachSeatsAvailable > 0) & (Status = \”New\“)” • Primeira classe “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = \”New\“)”	Localiza voos com base na cidade de destino, na cidade de partida, na data de partida e no tipo de lugar

ArrivalCityDepartureCity SeatType()	“SELECT (Form = \”Lista de voos disponíveis\“) & (ArrivalCityCode = \”“ + aArrivalCity + ”\“) & (DepartureCityCode = \”“ + aDepartureCity + ”\“)” • Classe económica “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (CoachSeatsAvailable > 0) & (Status = \”New\“)” • Primeira classe “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (FirstClassSeatsAvailable > 0) & (Status = \”New\“)”	Localiza voos com base na cidade de destino, na cidade de partida e no tipo de lugar
ArrivalCityDepartureDate()	“SELECT (Form = \”Lista de voos disponíveis\“) & (ArrivalCityCode = \”“ + aArrivalCity + ”\“)” “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (Status = \”New\“) & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))”	Localiza voos com base na cidade de destino e na data de partida
ArrivalCityDepartureDate SeatType()	“SELECT (Form = \”Lista de voos disponíveis\“) & (ArrivalCityCode = \”“ + aArrivalCity + ”\“)” • Classe económica “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (CoachSeatsAvailable > 0) & (Status = \”New\“)” • Primeira classe “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = \”New\“)”	Localiza voos com base na cidade de destino, na data de partida e no tipo de lugar

ArrivalCitySeatType()	“SELECT (Form = \”Lista de voos disponíveis\“) & (ArrivalCityCode = \”“ + aArrivalCity + ”\“)” • Classe económica “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (CoachSeatsAvailable > 0) & (Status = \”New\“)” • Primeira classe “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (FirstClassSeatsAvailable > 0) & (Status = \”New\“)”	Localiza voos com base na cidade de destino e no tipo de lugar
ArrivalDate()	“SELECT (Form = \”Voos programados\“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (Status = \”New\“) & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))”	Localiza voos com base na data de regresso
ArrivalDateDepartureCity()	“SELECT (Form = \”Lista de voos disponíveis\“) & (DepartureCityCode = \”“ + aDepartureCity + ”\“)” “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (Status = \”New\“) & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))”	Localiza voos com base na data de regresso e na cidade de partida
ArrivalDateDepartureCity DepartureDate()	“SELECT (Form = \”Lista de voos disponíveis\“) & (DepartureCityCode = \”“ + aDepartureCity + ”\“)” “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (Status = \”New\“) & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))”	Localiza voos com base na data de regresso, na cidade de partida e na data de partida

ArrivalDateDepartureCity DepartureDateSeatType()	"SELECT (Form = \"Lista de voos disponíveis\") & (DepartureCityCode = \"" + aDepartureCity + "\")" • Classe económica "SELECT (Form = \"Voos programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")" • Primeira classe "SELECT (Form = \"Voos programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")"	Localiza voos com base na data de regresso, na cidade de partida, na data de partida e no tipo de lugar
ArrivalDateDepartureCity SeatType()	"SELECT (Form = \"Lista de voos disponíveis\") & (DepartureCityCode = \"" + aDepartureCity + "\")" • Classe económica "SELECT (Form = \"Voos programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (CoachSeatsAvailable > 0) & (Status = \"New\")" • Primeira classe "SELECT (Form = \"Voos programados\") & (ScheduledFlightNumber = \""+ flightNum + "\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (FirstClassSeatsAvailable > 0) & (Status = \"New\")"	Localiza voos com base na data de regresso, na cidade de partida e no tipo de lugar
ArrivalDateDepartureDate()	"SELECT (Form = \"Voos programados\") & (ScheduledArrivalDate = [" + aArrivalDate + "]) & (ScheduledDepartureDate = [" + aDepartureDate + "]) & (Status = \"New\") & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))"	Localiza voos com base na data de regresso e na data de partida

ArrivalDateDepartureDate SeatType()	“SELECT (Form = \”Lista de voos disponíveis\“)” • Classe económica “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \””+ flightNum + ”\“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (CoachSeatsAvailable > 0) & (Status = \”New\“)” • Primeira classe “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \””+ flightNum + ”\“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = \”New\“)”	Localiza voos com base na data de regresso, na data de partida e no tipo de lugar
ArrivalDateSeatType()	• Classe económica “SELECT (Form = \”Voos programados\“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (CoachSeatsAvailable > 0) & (Status = \”New\“)” • Primeira classe “SELECT (Form = \”Voos programados\“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = \”New\“)”	Localiza voos com base na data de regresso e no tipo de lugar
DepartureCity()	“SELECT (Form = \”Lista de voos disponíveis\“) & (DepartureCityCode = \”“ + aDepartureCity + ”\“)” “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \””+ flightNum +”\“) & (Status = \”New\“) & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))”	Localiza voos com base na cidade de partida

DepartureCityDepartureDate()	“SELECT (Form = \”Lista de voos disponíveis\“) & (DepartureCityCode = \”“ + aDepartureCity + ”\“)” “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (Status = \”New\“) & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))”	Localiza voos com base na cidade de partida e na data de partida
DepartureCityDepartureDate SeatType()	“SELECT (Form = \”Lista de voos disponíveis\“) & (DepartureCityCode = \”“ + aDepartureCity + ”\“)” • Classe económica “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (CoachSeatsAvailable > 0) & (Status = \”New\“)” • Primeira classe “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = \”New\“)”	Localiza voos com base na cidade de partida, na data de partida e no tipo de lugar
DepartureCitySeatType()	“SELECT (Form = \”Lista de voos disponíveis\“) & (DepartureCityCode = \”“ + aDepartureCity + ”\“)” • Classe económica “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (CoachSeatsAvailable > 0) & (Status = \”New\“)” • Primeira classe “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (FirstClassSeatsAvailable > 0) & (Status = \”New\“)”	Localiza voos com base na cidade de partida e no tipo de lugar

DepartureDate()	“SELECT (Form = \”Voos programados\“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (Status = \”New\“) & ((CoachSeatsAvailable > 0) | (FirstClassSeatsAvailable>0))”	Localiza voos com base na data de partida
DepartureDateSeatType()	• Classe económica “SELECT (Form = \”Voos programados\“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (CoachSeatsAvailable > 0) & (Status = \”New\“)” • Primeira classe “SELECT (Form = \”Voos programados\“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = \”New\“)”	Localiza voos com base na data de partida e no tipo de lugar
PrimaryKey()	“SELECT (Form = \”Voos programados\“) & (ScheduledFlightByDate = \””+ tempScheduledFlightByDate +”\“)” “SELECT ((Form = \”Lista de voos disponíveis\“)) & (AvailableFlightNumber = \”“ + flightNum + ”\“)”	Localiza voos com base no voo programado por data

SearchCriteria()	“SELECT (Form = \”Lista de voos disponíveis\“) & (ArrivalCityCode = \”“ + aArrivalCity + ”\“) & (DepartureCityCode = \”“ + aDepartureCity + ”\“)” • Classe económica “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (CoachSeatsAvailable > 0) & (Status = \”New\“)” • Primeira classe “SELECT (Form = \”Voos programados\“) & (ScheduledFlightNumber = \”“+ flightNum + ”\“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = \”New\“)”	Localiza voos com base em todos os critérios da procura
SeatType()	• Classe económica “SELECT (Form = \”Voos programados\“) & (CoachSeatsAvailable > 0) & (Status = \”New\“)” • Primeira classe “SELECT (Form = \”Voos programados\“) & (FirstClassSeatsAvailable > 0) & (Status = \”New\“)”	Localiza voos com base no tipo de lugar

O método ejbLoad() é utilizado para obter os dados com base voo programado por data a partir de um documento Voos programados e do documento correspondente Lista de voos disponíveis e coloca-os nas propriedades do bean.

Os métodos getter e setter, bem como os valores que devolvem ou definem, estão listados na Tabela 5.

Tabela 5 Métodos Getter e Setter de Flight

Getter	Setter	Nome do campo no formulário	Tipo de dados
getArrivalCityCode()	setArrivalCityCode()	Lista de voos disponíveis: DisplayOnlyArrival	Cadeia
getArrivalDate()	setArrivalDate()	Voos programados: ScheduledArrivalDate	Cadeia

getArrivalTime()	setArrivalTime()	Lista de voos disponíveis: ArrivalTime	Cadeia
getCoachPrice()	setCoachPrice()	Lista de voos disponíveis: CoachPrice	Cadeia
getCoachSeatsAvailable()	setCoachSeatsAvailable()	Voos disponíveis: CoachSeatsAvailable	Cadeia
getData()		JavaBean FlightInfo com os valores no enterprise bean	FlightInfo
getDepartureCityCode()	setDepartureCityCode()	Lista de voos disponíveis: DisplayOnlyDeparture	Cadeia
getDepartureDate()	setDepartureDate()	Voos programados: ScheduledDepartureDate	Cadeia
getDepartureTime()	setDepartureTime()	Lista de voos disponíveis: DepartureTime	Cadeia
getFirstClassPrice()	setFirstClassPrice()	Lista de voos disponíveis: FirstClassPrice	Cadeia
getFirstClassSeatsAvailable()	setFirstClassSeatsAvailable()	Voos programados: FirstClassSeatsAvailable	Cadeia
getFood()	setFood()	Lista de voos disponíveis: Food	Cadeia
getListAirplaneType()	setListAirplaneType()	Lista de voos disponíveis: ListAirplaneType	Cadeia
getScheduledFlightByDate()	setScheduledFlightByDate()	Voos programados: ScheduledFlightByDate	Cadeia

FlightInfo

O FlightInfo é um JavaBean utilizado para armazenar as informações do voo. É introduzido no JSP apropriado que, por sua vez, o utiliza para obter os dados específicos do voo.

Os métodos getter e setter, bem como os valores que devolvem ou definem, estão listados na Tabela 6.

Tabela 6 Métodos Getter e Setter de FlightInfo

Getter	Setter	Valor	Tipo de dados
getArrivalCityCode()	setArrivalCityCode()	o ódigo da cidade de destino do voo	Cadeia
getArrivalDate()	setArrivalDate()	Data de regresso do voo	Cadeia
getArrivalTime()	setArrivalTime()	Hora de chegada do voo	Cadeia
getCityCode()	setCityCode()	Código da cidade do voo	Cadeia
getCoachPrice()	setCoachPrice()	Preço do bilhete de classe económica do voo	Cadeia
getCoachSeatsAvailable()	setCoachSeatsAvailable()	Lugares de classe económica do voo disponíveis	Cadeia
getDepartureCityCode()	setDepartureCityCode()	Código da cidade de partida do voo	Cadeia
getDepartureDate()	setDepartureDate()	Data de partida do voo	Cadeia
getDepartureTime()	setDepartureTime()	Hora de partida do voo	Cadeia

getFirstClassPrice()	setFirstClassPrice()	Preço do bilhete de primeira classe do voo	Cadeia
getFirstClassSeatsAvailable()	setFirstClassSeatsAvailable()	Lugares de primeira classe do voo disponíveis	Cadeia
getFlightList()	setFlightList()	Lista de voos	Vector
getFood()	setFood()	Refeições durante o voo	Cadeia
getListAirplaneType()	setListAirplaneType()	Tipo de avião do voo	Cadeia
getScheduledFlightByDate()	setScheduledFlightByDate()	Voo programado por data	Cadeia

Bean Ticket

O bean Ticket é um entity bean utilizado para representar um bilhete. Fornece métodos que realizam tarefas executadas em dados do bilhete, como por exemplo:

- Criar um novo bilhete para um voo que foi reservado por um cliente
- Obter uma lista de bilhetes para um cliente ou número de itinerário especificado
- Sinalizar o pagamento do bilhete e guardar as informações do cartão de crédito

O enterprise bean Ticket utiliza persistência gerida por bean e está mapeado aos formulários Informações do bilhete e Informações de crédito na base de dados flights.nsf do Lotus Domino. O esquema do formulário Informações do bilhete é apresentado na Tabela 5 e o formulário Informações de crédito é apresentado na Tabela 6 na secção "Fluxo de trabalho do ambiente do Lotus(R) Domino(TM)" na página 6.

O enterprise bean Ticket contém vários métodos ejbFindBy. Estes métodos de localizador são definidos na interface da página inicial. A Tabela 7 contém os métodos de localizador.

Tabela 7 Métodos de localizador de Ticket

Nome do método	Instrução Select	Descrição
ejbFindByCustomerNumber()	"SELECT ((Form=\"Informações do bilhete\")) & (TicketCustomerNumber = \"" + aCustomerNumber + "\")"	Localiza bilhetes com base no número de cliente
ejbFindByInvoiceNumber()	"SELECT ((Form=\"Informações do bilhete\")) & (InvoiceNumber = \"" + aInvoiceNumber + "\")"	Localiza bilhetes com base no número da factura
ejbFindByPrimaryKey()	"SELECT ((Form=\"Informações do bilhete\")) & (FlightByDate = \"" + tk.flightByDate + "\") & (SeatNumber = \"" + tk.seatNumber +"\")" "SELECT ((Form=\"Informações de crédito\")) & (InvoiceNumber = \"" + invoiceNumber + "\")"	Localiza bilhetes com base no voo por data e no número do lugar

O método ejbLoad() é utilizado para obter os dados com base no voo por data e no número do lugar a partir de um documento Informações do bilhete e do documento correspondente Informações de crédito e coloca-os nas propriedades do bean.

O método ejbStore() é utilizado para actualizar os dados num documento Informações do bilhete específico e do documento correspondente Informações de crédito a partir das propriedades do bean.

Os métodos getter e setter, bem como os valores que devolvem ou definem, estão listados na Tabela 8.

Tabela 8 Métodos Getter e Setter de Ticket

Getter	Setter	Nome do campo no formulário	Tipo de dados
getCreditCardExpirationDate()	setCreditCardExpirationDate()	Informações de crédito: ExpirationDate	Cadeia
getCreditCardNumber()	setCreditCardNumber()	Informações de crédito: CreditCardNumber	Cadeia
getCreditCardType()	setCreditCardType()	Informações de crédito: CardType	Cadeia
getCustomerNumber()	setCustomerNumber()	Informações do bilhete: TicketCustomerNumber	Cadeia
getData()		JavaBean TicketInfo com os valores no enterprise bean	Cadeia
getFlightByDate()	setFlightByDate()	Informações do bilhete: FlightByDate	Cadeia
getInvoiceNumber()	setInvoiceNumber()	Informações do bilhete: InvoiceNumber	Cadeia
getPaidStatus()	setPaidStatus	Informações do bilhete: PaidStatus	Cadeia
getPassengerAddress()	setPassengerAddress()	Informações do bilhete: PassengerStreet	Cadeia
getPassengerCity()	setPassengerCity()	Informações do bilhete: PassengerCity	Cadeia
getPassengerCountry()	setPassengerCountry()	Informações do bilhete: PassengerCountry	Cadeia
getPassengerFirstName()	setPassengerFirstName()	Informações do bilhete: PassengerFirstName	Cadeia
getPassengerLastName()	setPassengerLastName()	Informações do bilhete: PassengerLastName	Cadeia
getPassengerMiddleInitial()	setPassengerMiddleInitial()	Informações do bilhete: PassengerMiddleInitial	Cadeia
getPassengerPhoneNumber()	setPassengerPhoneNumber()	Informações do bilhete: PassengerPhone	Cadeia
getPassengerState()	setPassengerState()	Informações do bilhete: PassengerState	Cadeia
getPassengerZipCode()	setPassengerZipCode()	Informações do bilhete: PassengerZip	Cadeia
getSeatNumber()	setSeatNumber()	Informações do bilhete: SeatNumber	Cadeia
getTicketClass()	setTicketClass()	Informações do bilhete: TicketClass	Cadeia
getTicketPrice()	setTicketPrice()	Informações do bilhete: TicketPrice	BigDecimal

getTicketStatus()	setTicketStatus()	Informações do bilhete: TicketStatus	Cadeia
payTicket()		Informações do bilhete: PaidStatus e define Credit Information (Informações de crédito)	Cadeia

TicketInfo

O TicketInfo é um JavaBean utilizado para armazenar as informações do bilhete. É introduzido no JSP apropriado que, por sua vez, o utiliza para obter os dados específicos do bilhete.

Os métodos getter e setter, bem como os valores que devolvem ou definem, estão listado na Tabela 9.

Tabela 9 Métodos Getter e Setter de TicketInfo

Getter	Setter	Valor	Tipo de dados
getCreditCardExpirationDate()	setCreditCardExpirationDate()	Data de expiração do cartão de crédito	Cadeia
getCreditCardNumber()	setCreditCardNumber()	Número do cartão de crédito	Cadeia
getCreditCardType()	setCreditCardType()	Tipo de cartão de crédito	Cadeia
getCustomerNumber()	setCustomerNumber()	Número de cliente	Cadeia
getFlightByDate()	setFlightByDate()	Voo por data	Cadeia
getInvoiceNumber()	setInvoiceNumber()	Número da factura	Cadeia
getPaidStatus()	setPaidStatus()	Estado de Pago	Cadeia
getPassengerAddress()	setPassengerAddress()	Endereço do passageiro	Cadeia
getPassengerCity()	setPassengerCity()	Cidade do passageiro	Cadeia
getPassengerCountry()	setPassengerCountry()	País do passageiro	Cadeia
getPassengerFirstName()	setPassengerFirstName()	Nome próprio do passageiro	Cadeia
getPassengerLastName()	setPassengerLastName()	Apelido do passageiro	Cadeia
getPassengerMiddleInitial()	setPassengerMiddleInitial()	Outras iniciais do passageiro	Cadeia
getPassengerPhoneNumber()	setPassengerPhoneNumber()	Número de telefone do passageiro	Cadeia
getPassengerState()	setPassengerState()	Distrito do passageiro	Cadeia
getPassengerZipCode()	setPassengerZipCode()	Código postal do passageiro	Cadeia
getSeatNumber()	setSeatNumber()	Número do lugar	Cadeia
getTicketClass()	setTicketClass()	Classe do bilhete	Cadeia
getTicketList()	setTicketList()	Lista de bilhetes	Vector
getTicketPrice()	setTicketPrice()	Preço do bilhete	Cadeia
getTicketStatus()	setTicketStatus()	Estado do bilhete	Cadeia
getTotalCost()		Custo total de todos os bilhetes no vector da lista de bilhetes	Cadeia
getUniqueInvoiceNumbers()		Números de facturas únicos de todos os bilhetes no vector da lista de bilhetes	Vector

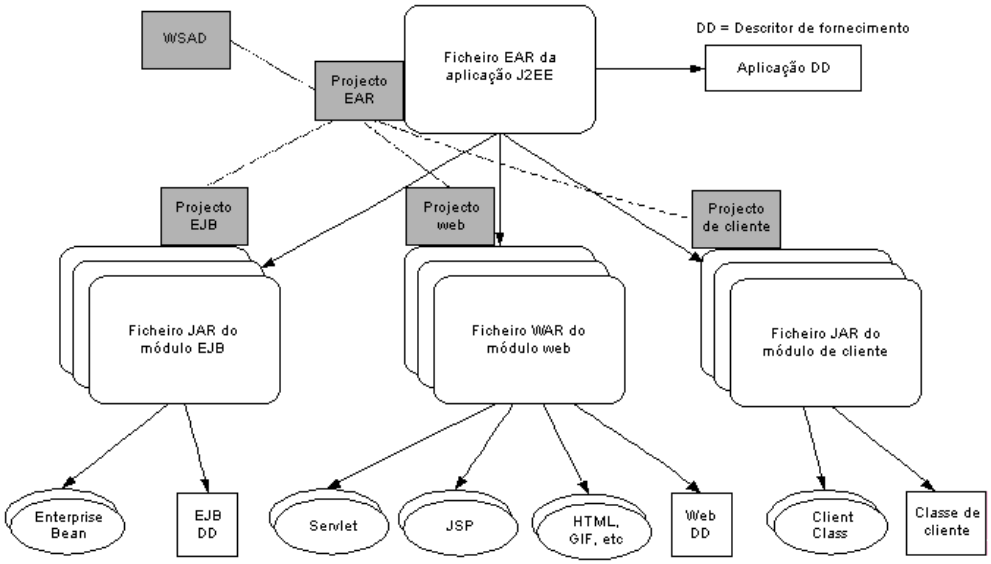
toString()		Representação em cadeia do objecto	Cadeia

Instalação da Aplicação Empresarial

O WebSphere Studio Application Developer combina a funcionalidade encontrada no Visual Age for Java(TM) e no WebSphere Studio. No entanto, foram adicionadas muitos componentes novos. Suporta perspectivas, que lhe permitem trabalhar com as aplicações a partir de vistas diferentes. Por exemplo, a perspectiva J2EE permite-lhe trabalhar num ambiente personalizado para a criação de aplicações em conformidade com J2EE.

O Application Developer permite-lhe exportar as aplicações directamente para formatos em conformidade com J2EE, tais como ficheiros EAR (enterprise archive file) e ficheiros WAR (Web archive file). É possível instalar estes ficheiros como aplicações empresariais no WebSphere Application Server 4.0 sem utilizar a Ferramenta do conjunto de aplicações (AAT - Application Assembly Tool) do WebSphere. A figura 2 apresenta a hierarquia de J2EE e o suporte correspondente no WebSphere Studio Application Developer.

Figura 2 Arquitectura de J2EE



A aplicação J2EE é armazenada num ficheiro EAR (Enterprise Archive) que contém os módulos dos enterprise beans (armazenados num ficheiro JAR do enterprise bean), módulos da Web (armazenados em ficheiros WAR (Web Archives)) e módulos de clientes (armazenados num ficheiro JAR). Um ficheiro WAR contém todos os componentes de uma aplicação Web: servlets, JSPs, ficheiros HTML, imagens, etc. Cada módulo contém um descritor de fornecimento. Por exemplo, um ficheiro WAR contém um ficheiro web.xml.

A hierarquia de J2EE é correspondida por projectos em WSAD. Um projecto EAR contém referências aos projectos Cliente, Web e enterprise bean.

Nesta aplicação, existe um ficheiro EAR FlightsEAR que contém os módulos da Web FlightsEJBModule e FlightsWebModule. O módulo FlightsEJBModule contém todos os enterprise beans (CustomerFlight, Customer, Flight e Ticket). O módulo FlightWebModule contém todos os componentes da Web(FlightsServlet, HTML, JSP e ficheiros de imagens).

Para gerar o ficheiro da Aplicação Empresarial de viagens aéreas:

1. Seleccione File -> Export from the Application Developer main screen (Ficheiro -> Exportar a partir do ecrã principal do Application Developer)
2. É apresenta a janela Export (Exportação) e o utilizador escolhe o ficheiro EAR
3. Na janela seguinte, seleccione o recurso FlightsEAR a partir da lista pendente do campo "What resource do you want to export?" ("Qual o recurso que pretende exportar?").
4. Introduza a localização SystemName:\QIBM\UserData\WebASAdv4\instanceName\installableApps\flight.ear para o campo "Where do you want to export resources to?" ("Para onde pretende exportar os recursos?"). Faça clique em Finish (Terminar).
5. Após a exportação da Aplicação Empresarial de viagens aéreas é possível ver o ficheiro na pasta.

Para instalar a Aplicação Empresarial de viagens aéreas:

1. Abra uma linha de comandos para iniciar a Consola Administrativa. Aguarde até ser apresentada a mensagem Console Ready (Consola pronta).
2. Na Consola, seleccione o ícone do assistente e faça clique em Install Enterprise Application (Instalar Aplicação Empresarial). É apresentada a janela Specifying the Application Module (Especificar o módulo da aplicação). Certifique-se de que o selector de opção Install Application (Instalar aplicação) está seleccionado. Faça clique no botão Browse (Procurar), junto a Path (Caminho), para localizar o flight.ear no directório \QIBM\UserData\WebASAdv4\instanceName\installableApps.
3. Depois de fazer clique em Next (Seguinte), é apresentada a seguinte mensagem: "This application contains method permissions. Do you wish to deny all unprotected methods?" ("Esta aplicação contém permissões de métodos. Pretende recusar todos os métodos não protegidos?") Resposta Yes (Sim).
4. Na janela Mapping User Roles (Mapear funções de utilizadores), prima o botão Select (Seleccionar), marque "All Authenticated Users" (Todos os utilizadores autenticados) e, em seguida, prima OK.
5. Continue a fazer clique em Next (Seguinte) até ser apresentada a janela "Selecting Application Servers" (Seleccionar servidores da aplicação). Nesta janela, seleccione todos os módulos na caixa Module (Módulo) e prima o botão Select Server (Seleccionar servidor). Escolha o servidor assumido e prima OK.
6. Faça clique em Next (Seguinte) e, em seguida, em Finish (Terminar) para instalar a aplicação, quando for apresentada a caixa de diálogo Regenerate the application (Gerar novamente a aplicação), faça clique em No (Não).

Agora que a Aplicação Empresarial de viagens aéreas está instalada no WebSphere Application Server Version 4.0 Advanced Edition, é necessário parar o servidor. Antes de reiniciar o servidor, é necessário copiar o NCSOW.jar para o directório QIBM/UserData/WebASAdv4/InstanceName/lib/ext, de modo a que seja efectuado um levantamento de fundos do mesmo no caminho da classe quando o servidor for reiniciado. Reinicie o servidor de modo a deixar a nova aplicação pronta.

A Aplicação Empresarial de viagens aéreas está pronta a ser utilizada. Para utilizar a aplicação, abra um navegador e introduza o seguinte URL: http://systemName:port/webapp/Flights/index.html

Observações chave sobre o ambiente do WebSphere(R) Application Server

Segue-se uma lista de observações chave não abordadas na criação nem na utilização do cenário de viagens aéreas do ambiente do WebSphere Application Server.

- Para definir o valor Session Timeout (Tempo de espera da sessão esgotado) num session bean com registo para um valor superior, execute os seguintes passos na Ferramenta do conjunto de aplicações (AAT - Application Assembly Tool) do WebSphere Application Server:
  1. Abra o enterprise bean JAR ou EAR no AAT.
  2. Pesquise detalhadamente o enterprise bean de tipo Session (Sessão) e faça clique no mesmo.
  3. Faça clique no separador IBM(R) Extensions (Extensões da IBM) na área da janela à direita.
  4. Defina a propriedade Timeout (Tempo de espera esgotado).
  5. Guarde o JAR ou EAR.
  6. Saia de AAT.
- A Ferramenta do conjunto de aplicações (AAT) do WebSphere Application Server pode ser utilizada para gerar o código de uma aplicação da empresa. Se ocorrerem problemas ao gerar o código de uma aplicação da empresa durante a instalação, a consola de administração do WebSphere Application Server não fornece muitas informações sobre o motivo da falha da instalação. Neste caso, pode utilizar a AAT para gerar o código implementado para o mesmo ficheiro EAR, sendo fornecidas informações adicionais sobre o motivo da falha. A consola de administração do WebSphere Application Server e a AAT utilizam o mesmo código subjacente para gerar o código. Existe também uma opção Verify (Verificar) que pode ser executada no ficheiro EAR que fornecerá informações adicionais.

- Para satisfazer as necessidades dos JSPs, foi necessário adicionar alguns campos aos documentos do Lotus(R) Domino(TM). No ficheiro flightSearch.jsp, foi necessário adicionar os seguintes campos ao formulário Scheduled Flights (Voos programados): Departure Date (Data de partida) e Arrival Date (Data de regresso). No ficheiro passengerInformation.jsp, foi necessário adicionar os campos de informações sobre os passageiros ao formulário (Informações do bilhete).

- Quando utilizar o controlador JDBC do Lotus Domino, nas instruções de SQL, os nomes das colunas e documentos do Lotus Domino são sensíveis a maiúsculas e minúsculas.

- Na aplicação de viagens aéreas, as APIs do Lotus Domino Java(TM) são utilizadas para estabelecer ligação a uma base de dados do Lotus Domino. Inicialmente pretendia-se utilizar o controlador JDBC do Lotus Domino; no entanto, no WebSphere Application Server, não pode criar uma origem de dados que utilize o controlador JDBC do Lotus Domino no iSeries(TM). Uma vez que o iSeries ainda não efectuou a transferência do controlador JDBC do Lotus Domino, também não foi possível implementar o conjunto de ligação. Está agendada a disponibilização de um controlador JDBC do Lotus Domino para o iSeries numa próxima edição.

- As seguintes sugestões poderão ser úteis caso tenham ocorrido problemas ao estabelecer ligação remotamente a um servidor Lotus Domino a partir de uma aplicação Java do WebSphere:

  – Certifique-se de que DIIOP está configurado no servidor Lotus Domino. Se DIIOP não estiver configurado, recebe uma mensagem indicando que o sistema central remoto recusou a ligação.

  – Certifique-se de que especifica o número da porta do servidor de HTTP do Lotus Domino ao tentar obter uma sessão do NotesFactory. Por exemplo:
  Session session = NotesFactory.createSession(“nomeSistema.nomeDomínio:númeroPorta”, “ID do utilizador”, “palavra-passe”).

Deste modo, o pedido CORBA para obter o servidor correcto é autorizado. Se o número da porta não for especificado, recebe uma excepção NotesException.

– Poderá receber uma excepção NotesException 4377: o servidor tem de estar no mesmo sistema central da sessão ao executar a instrução getDatabase a partir da aplicação Java utilizando as seguintes linhas de código:
Session session = NotesFactory.createSession(notesServer, notesUser, password);
ndbContent = session.getDatabase(notesServer, notesDatabase);

Existem duas soluções:

1. Na instrução getDatabase, envie apenas o nome do servidor e não o nome do servidor juntamente com o número da porta.
2. Utilize "" como notesServer na chamada de getDatabase, forçando a utilização da sessão acabada de criar:
ndbContent = session.getDatabase("", notesDatabase);

– Certifique-se de que CLASSPATH contém o ficheiro NCSOW.jar obtido a partir da versão do Lotus Domino que estiver a utilizar. Trata-se da versão do WebSphere do ficheiro NCSO.jar. O ficheiro NCSO.jar não está em condições, uma vez que os níveis de IIOP entram em conflito e não poderá criar uma sessão.

– Para utilizar as APIs Java do Domino para estabelecer ligação a partir de uma aplicação Java do WebSphere a uma base de dados do Domino através de IIOP é necessário que a aplicação importe o pacote lotus.domino.*.

– Para obter informações adicionais sobre NotesExceptions, adicione System.out.println, conforme exemplificado no código abaixo. Deste modo, pode imprimir as variáveis estáticas que explicam melhor o erro:

```
catch(lotus.domino.NotesException ne)
{
System.out.println(ne.text + " " + ne.id);
ne.printStackTrace();
}
```

- Numa aplicação Java que utilize APIs Java do Lotus Domino, ao escrever uma instrução Select para procurar uma data, tem de especificar a data que procura como uma constante (ou seja, colocar [ ] em redor do valor). Por exemplo: SELECT (Form = "Voos programados") & (ScheduledDepartureDate = [10/31/2001]).

- Ao codificar métodos do localizador em entity beans com persistência gerida por bean, o localizador é implementado pelo método ejbFindBy*xxxx*(). Os métodos do localizador na interface inicial terão o nome findBy*xxxx*(). O contentor implementa findBy*xxxx*() essencialmente como um conector em redor de ejbFindBy*xxxx*() para que o código do localizador real tenha de ser gravado.

- Para evitar receber uma excepção CSITransactionRolledbackException ao chamar um entity bean com persistência gerida por bean a partir de um session bean, é necessário que o session bean seja fornecido com uma definição de transacção TX_SUPPORTS.

  Quando o código do cliente está a tentar utilizar um objecto que o servidor do Lotus Domino já não reconhece ocorre uma excepção org.omg.CORBA.INV_OBJREF. Esta situação pode ocorrer se a sessão do cliente permanecer inactiva durante um período superior ao permitido pelo parâmetro de tempo de

espera esgotado da sessão no registo do servidor ou se um utilizador forçar uma ligação de todas as sessões a partir da consola do servidor. Permitir que a sessão fique inactiva durante um longo período de tempo não é recomendável, pelo que DIIOP é interrompido e termina estas sessões. O cliente é responsável pelo modo como resolver a situação.

Se o cliente não resolver o erro, o tempo de espera da sessão do Lotus Domino esgota-se e o objecto da sessão torna-se obsoleto, sendo apresentadas as seguintes mensagens de erro.

Segue-se a mensagem de erro do WebSphere Application Server:

```
org.omg.CORBA.INV_OBJREF: minor code: 1229062208 completed: No
java/lang/Throwable.<init>(Ljava/lang/String;)V+4 (Throwable.java:94)
org/omg/CORBA/INV_OBJREF.<init>(Ljava/lang/String;ILorg/omg/CORBA
/CompletionStatus;)V+1
(INV_OBJREF.java:72)
org/omg/CORBA/INV_OBJREF.<init>(Ljava/lang/String;)V+6 (INV_OBJREF.java:48)
com/ibm/CORBA/iiop/ReplyMessage.getSystemException()Lorg/omg/CORBA/
SystemException;+119
(ReplyMessage.java:181)
com/ibm/rmi/iiop/ClientResponseImpl.getSystemException()Lorg/omg/CORBA/
SystemException;+11
(ClientResponseImpl.java:89)
com/ibm/CORBA/iiop/ClientDelegate.invoke(Lorg/omg/CORBA/Object;Lorg/omg/
CORBA/portable/OutputStream;)Lorg/omg/CORBA/portable/InputStream;+235
(ClientDelegate.java:439)
org/omg/CORBA/portable/ObjectImpl._invoke(Lorg/omg/CORBA/portable/
OutputStream;)Lorg/omg/CORBA/portable/InputStream;+4
(ObjectImpl.java:251)
lotus/domino/corba/_IDatabaseStub.search(Ljava/lang/String;Llotus/domino/
corba/IDateTime;I)Llotus/domino/corba/DCData;+0
(_IDatabaseStub.java:0)
lotus/domino/cso/Database.search(Ljava/lang/String;Llotus/domino/DateTime;I)
Llotus/domino/DocumentCollection;+0
(Database.java:1478)
lotus/domino/cso/Database.search(Ljava/lang/String;)Llotus/domino/
DocumentCollection;+0
(Database.java:1452)
com/flights/ejb/session/CustomerFlightBean.getAllCityCodes()Lcom/flights/
FlightInfo;+0
(CustomerFlightBean.java:110)
com/flights/ejb/session/EJSRemoteCustomerFlight.getAllCityCodes()Lcom/
flights/FlightInfo;+0
(EJSRemoteCustomerFlight.java:31)
com/flights/ejb/session/_EJSRemoteCustomerFlight_Tie._invoke(Ljava/lang/
String;Lorg/omg/CORBA/portable/InputStream;Lorg/omg/CORBA/portable/
ResponseHandler;)Lorg/omg/CORBA/portable/OutputStream;+0
(_EJSRemoteCustomerFlight_Tie.java:82)
com/ibm/CORBA/iiop/ExtendedServerDelegate.dispatch(Lcom/ibm/rmi/
ServerRequest;)Lcom/ibm/rmi/ServerResponse;+224
(ExtendedServerDelegate.java:506)
com/ibm/CORBA/iiop/ORB.process(Lcom/ibm/rmi/ServerRequest;)Lcom/ibm/
rmi/ServerResponse;+20
(ORB.java:2282)
com/ibm/CORBA/iiop/WorkerThread.run()V+89 (WorkerThread.java:195)
com/ibm/ejs/oa/pool/ThreadPool$PooledThread.run()V+67 (ThreadPool.java:641)
```

Segue-se a mensagem de erro do servidor Domino:

DIIOP SYSTEM EXCEPTION: INV_OBJREF, minor
code 49420040, SOMDERROR_BadObjref [somd_refdata_to_obj
(CORBA::ReferenceData*):1091]

Pode tentar definir o tempo de espera esgotado de IIOP para um valor superior ou pode tentar detectar o erro e restabelecer a sessão.

Seguem-se algumas sugestões utilizadas para tentar determinar se um objecto de sessão do Lotus Domino está aberto a partir de um cliente Java:

– Detecte a excepção a partir do enterprise bean e processe a excepção nesse local. Esta sugestão não funciona devido:

  Na especificação de Enterprise JavaBean 1.1, secção 12.3.4 ″Excepções e transacções″:

  “Se uma instância-objecto gerar uma excepção não verificada ao executar um contexto de transacção de cliente, o contentor tem de marcar a transacção para remover alterações e gerar uma excepção javax.jts.TransactionRolledbackException para o cliente.”

  “Se o contentor decidir, por um determinado motivo, marcar uma transacção para efectuar a remoção de alterações, deverá gerar a excepção javax.jts.TransactionRolledbackException para o cliente. A excepção javax.jts.TransactionRolledbackException constitui uma subclasse da excepção java.rmi.RemoteException e informa o cliente de que qualquer tentativa de recuperação no âmbito da transacção seria desnecessária, uma vez que a transacção não pode ser consolidada.”

  Em resumo, uma excepção não verificada que ocorra num enterprise bean faz sempre com que sejam removidas as alterações da transacção efectuada durante o voo, caso esteja a processar a mesma explicitamente. Basicamente, uma excepção não verificada trata-se de uma excepção que não derive de uma excepção java.lang.Exception. Uma excepção CORBA.INV_OBJREF não deriva de uma excepção java.lang.Exception;, deste modo, pode ser classificada como uma excepção não verificada. Por isso, o contentor gera sempre uma excepção TransactionRolledbackException quando esta excepção é gerada.

– Utilize Session.isValid() a adicionar a uma versão posterior das APIs Java do Lotus Domino. Este procedimento constitui a melhor forma de determinar o estado de uma sessão, mas ainda não se encontra disponível.

  Componente: Session.isValid()
  Trata-se de parte do componente do conjunto de ligações DIIOP.

  Finalidade:
  A finalidade deste método apenas em Java consiste em determinar se um objecto Session criado continua a ser válido. Relativamente à API remota, determina se a tarefa do servidor DIIOP continua a considerar válida esta sessão e, assim, este método pode efectuar uma operação de rede. Por este motivo, este método não deverá ser utilizado num ciclo fixo.
  Relativamente à API local, também determina se uma sessão continua a ser válida.

  Assinatura:
  boolean isValid();

NOTA: Este método não gera quaisquer Excepções.

Utilização:

Segue-se um exemplo sobre como utilizar o método num tipo de executor de servlet de módulo.

```
class WorkerThread extends Thread {
...
public void run()
{
while (WaitForWork()) {
if ( ! session.isValid() ) {
// é necessário criar nova sessão
}
// efectuar o trabalho
}
}
}
```

  - Modifique o código getConnection para obter sempre uma nova sessão no Lotus Domino. Deste modo, evita-se a excepção CORBA.INV_OBJREF na globalidade. Esta sugestão funciona, ainda que seja fornecido tempo de sistema adicional à aplicação Java.

- No documento do servidor no separador IIOP, o número máximo de módulos que podem ser especificados pelo administrador não é restrito. Numa edição posterior do Lotus Domino, esta definição deixará de ser utilizada pelo DIIOP. Será disponibilizada no documento do servidor apenas para suporte de versões anteriores. O valor mínimo de tempo de espera esgotado de uma sessão no documento do servidor do Lotus Domino é de cinco minutos.

- Para especificar a informação do caminho da classe para objectos que residam em QIBM sem utilizar a AAT, no WebSphere Studio Application Developer, recomenda-se que crie um ficheiro MANIFEST.MF nas seguintes localizações:

  Num módulo de enterprise bean, recomenda-se que crie ou utilize o ficheiro de guia de diagnóstico existente na seguinte localização: ejbModule -> META-INF -> MANIFEST.MF

  Num módulo da Web, recomenda-se que crie ou utilize o ficheiro de guia de diagnóstico existente na seguinte localização: webApplication -> META-INF -> MANIFEST.MF

  Segue-se um exemplo do aspecto do caminho de classe num ficheiro Manifest.mf:

  ```
  Versão de Manifest: 1.0
  Caminho da classe:
  /qibm/userdata/webasadv4/flight4/installedapps/flightsear.ear/flightsejbmodule.jar
  ```

- Após importar as versões publicadas dos JSPs das viagens aéreas para o WebSphere Studio Application Developer, cada uma foi editada utilizando o Page Designer. Após a edição, o EAR foi exportado para o directório instalável e, em seguida, cada JSP foi exportado para uma localização de equipa central. Os JSPs aos quais foram aplicadas actualizações não foram exportados para a localização da equipa. Também se detectou que estes JSPs não podiam ser copiados nem eliminados no WebSphere Studio Application Developer. O erro recebido indicava que o recurso não estava em sincronização com o sistema de ficheiros. Os JSPs que foram abertos, mas aos quais não foram efectuadas alterações,

encontravam-se funcionais. Em seguida, o módulo da Web foi actualizado a partir do local. Este procedimento permitiu que os JSPs fossem copiados, eliminados e exportados.

- A tabela 1 lista os atalhos de teclado que pode utilizar no editor Java do WebSphere Studio Application Developer:

Tabela 1 Atalhos de teclado

Descrição	Sequência de teclas
Importar	Ctrl+Shift+M
Avançar para o número da linha	Ctrl+L
Indentar o texto evidenciado	Ctrl+I
Pesquisar/substituir	Ctrl+F
Copiar	Ctrl+C
Cortar	Ctrl+X
Desfazer	Ctrl+Z
Seleccionar tudo	Ctrl+A
Avançar para o erro seguinte	Ctrl+E
Apresenta a pesquisa Java incluindo o item seleccionado na tabela de pesquisa	Ctrl+H
Apresenta o assistente de codificação/conteúdo. Após efectuar a selecção, é apresentado o Javadoc na ajuda sensível ao rato	Ctrl+Barra de espaços
Executa uma criação progressiva de um projecto na vista de navegação	Ctrl+B
Mantenha premida a tecla Ctrl e utilize o recurso arrastar e largar para copiar o recurso entre diferentes janelas do Workbench	Ctrl+Arrastar e largar

Capítulo 5. Referências

- Developing iSeries J2EE Applications for WebSphere 4.0, IBM Redbook SG24-6559-00
- WebSphere Studio Application Developer Programming Guide, IBM Redbook SG24-6585-00
- Tips for Working with Lotus Domino Objects
  http://www.advisor.com/Articles.nsf/aidp/BALAB03
- WebSphere Studio Application Developer Migration Guide
  http://www7b.boulder.ibm.com/wsdd/library/techarticles/0110_wsad_mig/migration_ga.html

Exemplo: Bean Customer

O exemplo que se segue ilustra a codificação do entity bean Customer. O bean Customer utiliza persistência gerida por bean e está mapeado ao formulário FlightPerson na base de dados names.nsf do Lotus(R) Domino(TM). Utiliza as APIs do Lotus Domino para aceder à base de dados do Lotus Domino database.

O código de origem para CustomerKey é apresentado no Exemplo 1.

Exemplo 1: Código de origem de CustomerKey

```
package com.flights.ejb.bmp;


/**
* This is a Primary Key Class for the Entity Bean
**/
public class CustomerKey implements java.io.Serializable {
public String primaryKey;
final static long serialVersionUID = 3206093459760846163L;


/**
* CustomerKey() constructor
*/
public CustomerKey() {
}
/**
* CustomerKey(String key) constructor
*/
public CustomerKey(String key) {
primaryKey = key;
}
/**
* equals method
* - user must provide a proper implementation for the equal method. The generated
* method assumes the key is a String object.
*/
```

```
public boolean equals (Object o) {
if (o instanceof CustomerKey)
return primaryKey.equals(((CustomerKey)o).primaryKey);
else
return false;
}
/**
* hashode method
* - user must provide a proper implementation for the hashCode method. The generated
* method assumes the key is a String object.
*/
public int hashCode () {
return primaryKey.hashCode();
}
}
```

O código de origem para a interface de CustomerHome é apresentado no Exemplo 2.

Exemplo 2: Código de origem de CustomerHome

```
package com.flights.ejb.bmp;


/**
* This is a Home interface for the Entity Bean
*/
public interface CustomerHome extends javax.ejb.EJBHome {


/**
* create method for a BMP entity bean
* @return com.flights.ejb.bmp.Customer
* @param primaryKey com.flights.ejb.bmp.CustomerKey
* @exception javax.ejb.CreateException The exception description.
* @exception java.rmi.RemoteException The exception description.
*/
com.flights.ejb.bmp.Customer create(com.flights.ejb.bmp.CustomerKey primaryKey) throws
javax.ejb.CreateException, java.rmi.RemoteException;
/**
* findByPrimaryKey method comment
* @return com.flights.ejb.bmp.Customer
* @param key com.flights.ejb.bmp.CustomerKey
* @exception java.rmi.RemoteException The exception description.
* @exception javax.ejb.FinderException The exception description.
*/
com.flights.ejb.bmp.Customer findByPrimaryKey(com.flights.ejb.bmp.CustomerKey key) throws
java.rmi.RemoteException, javax.ejb.FinderException;
}
```

O código de origem para a interface remota de Customer é apresentada no Exemplo 3.

Exemplo 3: Código de origem da interface remota de Customer

```
package com.flights.ejb.bmp;


/**
* This is the enterprise bean Remote Interface for the Customer bean
*/
public interface Customer extends javax.ejb.EJBObject {


/**
* Returns the address for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerAddress() throws java.rmi.RemoteException;
/**
* Returns the city for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerCity() throws java.rmi.RemoteException;
/**
* Returns the country for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerCountry() throws java.rmi.RemoteException;
/**
* Returns the first name of the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerFirstName() throws java.rmi.RemoteException;
/**
* Returns the internet address for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerInternetAddress() throws java.rmi.RemoteException;
/**
* Returns the last name of the customer.
* @return java.lang.String
* @exception String The exception description.
*/
```

```
java.lang.String getCustomerLastName() throws java.rmi.RemoteException;
/**
* Returns the middle initial of the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerMiddleInitial() throws java.rmi.RemoteException;
/**
* Returns the customer number.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerNumber() throws java.rmi.RemoteException;
/**
* Returns the customer phone number.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerPhoneNumber() throws java.rmi.RemoteException;
/**
* Returns the state for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerState() throws java.rmi.RemoteException;
/**
* Returns the zip code for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerZipCode() throws java.rmi.RemoteException;
/**
* Returns the values within the bean as data stored within a CustomerInfo JavaBean.
* @return com.flights.CustomerInfo
* @exception String The exception description.
*/
com.flights.CustomerInfo getData() throws java.rmi.RemoteException;
/**
* Sets the address for the customer.
* @return void
* @param newCustomerAddress java.lang.String
* @exception String The exception description.
*/
void setCustomerAddress(java.lang.String newCustomerAddress) throws java.rmi.RemoteException;
/**
* Sets the city for the customer.
* @return void
* @param newCustomerCity java.lang.String
* @exception String The exception description.
*/
void setCustomerCity(java.lang.String newCustomerCity) throws java.rmi.RemoteException;
/**
* Sets the country for the customer.
* @return void
* @param newCustomerCountry java.lang.String
* @exception String The exception description.
```

```
*/
void setCustomerCountry(java.lang.String newCustomerCountry) throws java.rmi.RemoteException;
/**
* Sets the first name of the customer.
* @return void
* @param newCustomerFirstName java.lang.String
* @exception String The exception description.
*/
void setCustomerFirstName(java.lang.String newCustomerFirstName) throws java.rmi.RemoteException;
/**
* Sets the internet address for the customer.
* @return void
* @param newCustomerInternetAddress java.lang.String
* @exception String The exception description.
*/
void setCustomerInternetAddress(java.lang.String newCustomerInternetAddress) throws
java.rmi.RemoteException;
/**
* Sets the last name of the customer.
* @return void
* @param newCustomerLastName java.lang.String
* @exception String The exception description.
*/
void setCustomerLastName(java.lang.String newCustomerLastName) throws java.rmi.RemoteException;
/**
* Sets the middle initial of the customer.
* @return void
* @param newCustomerMiddleInitial java.lang.String
* @exception String The exception description.
*/
void setCustomerMiddleInitial(java.lang.String newCustomerMiddleInitial) throws
java.rmi.RemoteException;
/**
* Sets the customer number.
* @return void
* @param newCustomerNumber java.lang.String
* @exception String The exception description.
*/
void setCustomerNumber(java.lang.String newCustomerNumber) throws java.rmi.RemoteException;
/**
* Sets the phone number for the customer.
* @return void
* @param newCustomerPhoneNumber java.lang.String
* @exception String The exception description.
*/
void setCustomerPhoneNumber(java.lang.String newCustomerPhoneNumber) throws
java.rmi.RemoteException;
/**
* Sets the state for the customer.
* @return void
* @param newCustomerState java.lang.String
* @exception String The exception description.
*/
void setCustomerState(java.lang.String newCustomerState) throws java.rmi.RemoteException;
/**
* Sets the zip code for the customer.
```

```
* @return void
* @param newCustomerZipCode java.lang.String
* @exception String The exception description.
*/
void setCustomerZipCode(java.lang.String newCustomerZipCode) throws java.rmi.RemoteException;
}
```

P código de origem para o enterprise bean Customer é apresentado no Exemplo 4.

Exemplo 4: Código de origem do enterprise bean Customer

```
package com.flights.ejb.bmp;


import java.rmi.RemoteException;
import java.security.Identity;
import java.util.Properties;
import javax.ejb.*;
import lotus.domino.*;
import javax.naming.*;
/**
* This is an Entity Bean class with BMP fields
*/
public class CustomerBean implements EntityBean {
private javax.ejb.EntityContext entityContext = null;
private final static long serialVersionUID = 3206093459760846163L;


private java.lang.String customerAddress;
private java.lang.String customerCity;
private java.lang.String customerCountry;
private java.lang.String customerFirstName;
private java.lang.String customerInternetAddress;
private java.lang.String customerLastName;
private java.lang.String customerMiddleInitial;
private java.lang.String customerPhoneNumber;
private java.lang.String customerState;
private java.lang.String customerZipCode;
private java.lang.String customerNumber;
private transient Database ndbContent = null;
/**
* ejbActivate method comment
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbActivate() throws java.rmi.RemoteException {}
/**
* ejbCreate method for a BMP entity bean
* @return com.flights.ejb.bmp.CustomerKey
```

```
* @exception javax.ejb.CreateException The exception description.
* @exception java.rmi.RemoteException The exception description.
*/
public com.flights.ejb.bmp.CustomerKey ejbCreate() throws javax.ejb.CreateException,
java.rmi.RemoteException {
return null;
}
/**
* ejbCreate method for a BMP entity bean
* @return com.flights.ejb.bmp.CustomerKey
* @param key com.flights.ejb.bmp.CustomerKey
* @exception javax.ejb.CreateException The exception description.
* @exception java.rmi.RemoteException The exception description.
*/
public com.flights.ejb.bmp.CustomerKey ejbCreate(com.flights.ejb.bmp.CustomerKey key) throws
javax.ejb.CreateException, java.rmi.RemoteException {



return null;
}
/**
* ejbFindByPrimaryKey method comment
* @return com.flights.ejb.bmp.CustomerKey
* @param primaryKey com.flights.ejb.bmp.CustomerKey
* @exception java.rmi.RemoteException The exception description.
* @exception javax.ejb.FinderException The exception description.
*/
public com.flights.ejb.bmp.CustomerKey ejbFindByPrimaryKey(com.flights.ejb.bmp.CustomerKey
primaryKey) throws java.rmi.RemoteException, javax.ejb.FinderException {
refresh(primaryKey);
return primaryKey;
}
/**
* Used to refresh the enterprise bean from the persistent storage.
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbLoad() throws java.rmi.RemoteException {



System.out.println("Customer.ejbLoad()");
try
{
refresh((CustomerKey) entityContext.getPrimaryKey());
}
catch (FinderException fe)
{
throw new RemoteException(fe.getMessage());
}
}
/**
* ejbPassivate method comment
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbPassivate() throws java.rmi.RemoteException {}
```

```
/**
* ejbPostCreate method for a BMP entity bean
* @param key com.flights.ejb.bmp.CustomerKey
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbPostCreate(com.flights.ejb.bmp.CustomerKey key) throws java.rmi.RemoteException {}
/**
* ejbRemove method comment — currently not implemented
* @exception java.rmi.RemoteException The exception description.
* @exception javax.ejb.RemoveException The exception description.
*/
public void ejbRemove() throws java.rmi.RemoteException, javax.ejb.RemoveException {}
/**
* ejbStore method comment
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbStore() throws java.rmi.RemoteException {



System.out.println("Customer.ejbStore() ");



DocumentCollection dclResult = null;
Document docResult = null;



try
{
ndbContent = getConnection();



// Search for document with specified key values
System.out.println("Searching for document: " + customerNumber);
dclResult = ndbContent.search("SELECT (Form = \"FlightPerson\") & (PersonalID = "+
customerNumber.trim() +")");
docResult = dclResult.getFirstDocument();



if (docResult != null)
{
// Update document to contain new values
System.out.println("Should be updating customer info");



docResult.replaceItemValue("FirstName", customerFirstName);
docResult.replaceItemValue("MiddleInitial", customerMiddleInitial);
docResult.replaceItemValue("LastName", customerLastName);
docResult.replaceItemValue("StreetAddress", customerAddress);
docResult.replaceItemValue("City", customerCity);
docResult.replaceItemValue("State", customerState);
```

```
docResult.replaceItemValue(“Zip”, customerZipCode);
docResult.replaceItemValue(“Country”, customerCountry);
docResult.replaceItemValue(“PhoneNumber”, customerPhoneNumber);
docResult.replaceItemValue(“InternetAddress”, customerInternetAddress);



docResult.save();
}
else
throw new FinderException(“Customer EJB Store: CustomerBean (” + customerNumber + “) not found”);



System.out.println(“After update”);



}
catch(lotus.domino.NotesException ne)
{
System.out.println(ne.text + “ ” + ne.id);
ne.printStackTrace();
throw new RemoteException(ne.toString());
}
catch(Exception e)
{
e.printStackTrace();
throw new RemoteException(e.toString());
}



finally
{
try
{
System.out.println(“Recycle objects”);
if (dclResult != null)
dclResult.recycle();
if (docResult != null)
docResult.recycle();
}
catch(Exception ex)
{
throw new RemoteException(ex.toString());
}
}



}
/**
* Used to return a connection to the Lotus Domino database using Lotus Domino APIs .
* Creation date: (10/19/2001 3:14:11 PM)
* @return javax.sql.DataSource
```

```
* @exception java.sql.SQLException The exception description.
*/
private Database getConnection() throws java.rmi.RemoteException, java.sql.SQLException {



if (ndbContent == null) {
Properties properties = getEntityContext().getEnvironment();
String providerURL = properties.getProperty("provider_url");
String notesServer = properties.getProperty("notesServer");
String notesUser = properties.getProperty("notesUser");
String password = properties.getProperty("password");
String notesDatabase = properties.getProperty("notesDB");



InitialContext ctx = null;
Properties prop = new Properties();



try {
prop.put(Context.PROVIDER_URL, providerURL);
prop.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.ejs.ns.jndi.CNInitialContextFactory");
ctx = new InitialContext(prop);



System.out.println("creating notes session using current creds");
Session session = NotesFactory.createSession(notesServer, null);
System.out.println("username: " + session.getUserName());



System.out.println("got session - getting DB");
ndbContent = session.getDatabase("", notesDatabase);



System.out.println("got database");



if (!ndbContent.isOpen()) ndbContent.open();
}
catch(lotus.domino.NotesException ne){
System.out.println(ne.text + " " + ne.id);
ne.printStackTrace();
throw new RemoteException(ne.toString());
}
catch (Exception e) {
System.out.println("an error occurred");
e.printStackTrace();
throw new RemoteException(e.toString());
}
}
```

```
return ndbContent;
}
/**
* Returns address for customer.
* Creation date: (04/02/02 2:37:06 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerAddress() {
return customerAddress;
}
/**
* Returns city for customer.
* Creation date: (04/02/02 2:37:28 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerCity() {
return customerCity;
}
/**
* Returns country for customer.
* Creation date: (04/02/02 2:37:46 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerCountry() {
return customerCountry;
}
/**
* Returns first name of customer.
* Creation date: (04/02/02 2:38:05 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerFirstName() {
return customerFirstName;
}
/**
* Returns internet address of customer.
* Creation date: (04/02/02 2:38:24 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerInternetAddress() {
return customerInternetAddress;
}
/**
* Returns last name of customer.
* Creation date: (04/02/02 2:38:39 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerLastName() {
return customerLastName;
}
/**
* Returns middle initial of customer.
* Creation date: (04/02/02 2:38:55 PM)
* @return java.lang.String
```

```
*/
public java.lang.String getCustomerMiddleInitial() {
return customerMiddleInitial;
}
/**
* Returns customer number.
* Creation date: (04/02/02 2:40:12 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerNumber() {
return customerNumber;
}
/**
* Returns phone number for customer.
* Creation date: (04/02/02 2:39:10 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerPhoneNumber() {
return customerPhoneNumber;
}
/**
* Returns state for customer.
* Creation date: (04/02/02 2:39:22 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerState() {
return customerState;
}
/**
* Returns zip code for customer.
* Creation date: (04/02/02 2:39:40 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerZipCode() {
return customerZipCode;
}
/**
* Returns the values within the enterprise bean as data stored within a CustomerInfo JavaBean.
* Creation date: (03/26/02 7:20:12 AM)
*/
public com.flights.CustomerInfo getData() {


com.flights.CustomerInfo customerInfo = new com.flights.CustomerInfo();


customerInfo.setCustomerFirstName(customerFirstName);


System.out.println("getData(): Here's the customer first name: " + customerFirstName);
```

```
customerInfo.setCustomerMiddleInitial(customerMiddleInitial);



System.out.println("getData(): Here's the customer middle initial: " + customerMiddleInitial);
customerInfo.setCustomerLastName(customerLastName);
customerInfo.setCustomerAddress(customerAddress);
customerInfo.setCustomerInternetAddress(customerInternetAddress);



System.out.println("getData(): Here's the customer internet address: " + customerInternetAddress);
customerInfo.setCustomerCity(customerCity);
customerInfo.setCustomerState(customerState);
customerInfo.setCustomerZipCode(customerZipCode);
customerInfo.setCustomerCountry(customerCountry);
customerInfo.setCustomerPhoneNumber(customerPhoneNumber);



return customerInfo;
}
/**
* getEntityContext method comment
* @return javax.ejb.EntityContext
*/
public javax.ejb.EntityContext getEntityContext() {
return entityContext;
}



/**
* Refreshes the enterprise bean corresponding to the primary key from the persistent
* storage.
* Creation date: (11/14/2001 2:06:39 PM)
* @param aPrimaryKey com.flights.ejb.bmp.FlightKey
* @exception java.rmi.RemoteException The exception description.
* @exception javax.ejb.FinderException The exception description.
*/
private void refresh(CustomerKey primaryKey) throws java.rmi.RemoteException,
javax.ejb.FinderException {



DocumentCollection customerDC = null;
Document customerDoc = null;



System.out.println("starting refresh");



if (primaryKey == null)
throw new RemoteException("Primary key cannot be null");
```

```
customerNumber = primaryKey.primaryKey;
System.out.println("Customer Number in refresh: " + customerNumber);



try {



ndbContent = getConnection();



// find the customer number to obtain customer info
customerDC = ndbContent.search("SELECT (Form = \"FlightPerson\") & (PersonalID = "+
customerNumber.trim() +")");
System.out.println("number of elements in collection: " + customerDC.getCount());
customerDoc = customerDC.getFirstDocument();



if (customerDoc != null) {
customerFirstName = customerDoc.getItemValueString("FirstName");
customerMiddleInitial = customerDoc.getItemValueString("MiddleInitial");
customerLastName = customerDoc.getItemValueString("LastName");
customerAddress = customerDoc.getItemValueString("StreetAddress");
customerCity = customerDoc.getItemValueString("City");
customerState = customerDoc.getItemValueString("State");
customerZipCode = customerDoc.getItemValueString("Zip");
customerCountry = customerDoc.getItemValueString("Country");
customerPhoneNumber = customerDoc.getItemValueString("PhoneNumber");
customerInternetAddress = customerDoc.getItemValueString("InternetAddress");



System.out.println("FN: " + customerFirstName + " LN: " + customerLastName + " MI: " +
customerMiddleInitial + " Internet Addr: " + customerInternetAddress);



} // end if
else
{
throw new FinderException("Customer information not found for customer number " +
customerNumber);



}



} // end try
catch(lotus.domino.NotesException ne){
```

```
System.out.println(ne.text + " " + ne.id);
ne.printStackTrace();
throw new RemoteException(ne.toString());
}
catch(Exception e) {
e.printStackTrace();
throw new RemoteException(e.toString());
}
finally
{
try
{
System.out.println("Recycle objects in CustomerBean refresh");
if (customerDC != null)
customerDC.recycle();
if (customerDoc != null)
customerDoc.recycle();



}
catch(Exception ex)
{
throw new RemoteException(ex.toString());
}
}
}
/**
* Sets the address for the customer.
* Creation date: (04/02/02 2:37:06 PM)
* @param newCustomerAddress java.lang.String
*/
public void setCustomerAddress(java.lang.String newCustomerAddress) {
customerAddress = newCustomerAddress;
}
/**
* Sets the city for the customer.
* Creation date: (04/02/02 2:37:28 PM)
* @param newCustomerCity java.lang.String
*/
public void setCustomerCity(java.lang.String newCustomerCity) {
customerCity = newCustomerCity;
}
/**
* Sets the country for the customer.
* Creation date: (04/02/02 2:37:46 PM)
* @param newCustomerCountry java.lang.String
*/
public void setCustomerCountry(java.lang.String newCustomerCountry) {
customerCountry = newCustomerCountry;
}
/**
* Sets the first name of the customer.
* Creation date: (04/02/02 2:38:05 PM)
* @param newCustomerFirstName java.lang.String
*/
```

```
public void setCustomerFirstName(java.lang.String newCustomerFirstName) {
customerFirstName = newCustomerFirstName;
}
/**
* Sets the internet address for the customer.
* Creation date: (04/02/02 2:38:24 PM)
* @param newCustomerInternetAddress java.lang.String
*/
public void setCustomerInternetAddress(java.lang.String newCustomerInternetAddress) {
customerInternetAddress = newCustomerInternetAddress;
}
/**
* Sets the last name of the customer.
* Creation date: (04/02/02 2:38:39 PM)
* @param newCustomerLastName java.lang.String
*/
public void setCustomerLastName(java.lang.String newCustomerLastName) {
customerLastName = newCustomerLastName;
}
/**
* Sets the middle initial of the customer.
* Creation date: (04/02/02 2:38:55 PM)
* @param newCustomerMiddleInitial java.lang.String
*/
public void setCustomerMiddleInitial(java.lang.String newCustomerMiddleInitial) {
customerMiddleInitial = newCustomerMiddleInitial;
}
/**
* Sets the customer number.
* Creation date: (04/02/02 2:40:12 PM)
* @param newCustomerNumber java.lang.String
*/
public void setCustomerNumber(java.lang.String newCustomerNumber) {
customerNumber = newCustomerNumber;
}
/**
* Sets the phone number for the customer.
* Creation date: (04/02/02 2:39:10 PM)
* @param newCustomerPhoneNumber java.lang.String
*/
public void setCustomerPhoneNumber(java.lang.String newCustomerPhoneNumber) {
customerPhoneNumber = newCustomerPhoneNumber;
}
/**
* Sets the state for the customer.
* Creation date: (04/02/02 2:39:22 PM)
* @param newCustomerState java.lang.String
*/
public void setCustomerState(java.lang.String newCustomerState) {
customerState = newCustomerState;
}
/**
* Sets the zip code for the customer.
* Creation date: (04/02/02 2:39:40 PM)
* @param newCustomerZipCode java.lang.String
*/
```

```
public void setCustomerZipCode(java.lang.String newCustomerZipCode) {
customerZipCode = newCustomerZipCode;
}
/**
* setEntityContext method comment
* @param ctx javax.ejb.EntityContext
* @exception java.rmi.RemoteException The exception description.
*/
public void setEntityContext(javax.ejb.EntityContext ctx) throws java.rmi.RemoteException {
entityContext = ctx;
}
/**
* unsetEntityContext method comment
* @exception java.rmi.RemoteException The exception description.
*/
public void unsetEntityContext() throws java.rmi.RemoteException {
entityContext = null;
}
}
```

O código de origem para CustomerInfoBean é apresentado no Exemplo 5.

Exemplo 5: Código de origem de CustomerInfoBean

```
package com.flights;
import java.util.*;


/**
* CustomerInfo is a JavaBean used to store the customer information for
* a specific customer/flight. It is passed to the appropriate
* JSP which will use it to retrieve the specific customer data.
*/
public class CustomerInfo implements java.io.Serializable {


private java.lang.String customerFirstName;
private java.lang.String customerMiddleInitial;
private java.lang.String customerLastName;
private java.lang.String customerAddress;
private java.lang.String customerCity;
private java.lang.String customerState;
private java.lang.String customerZipCode;
private java.lang.String customerCountry;
private java.lang.String customerPhoneNumber;
private java.lang.String customerInternetAddress;
private Vector customerListVector;
/**
```

```
* CustomerInfo constructor.
*/
public CustomerInfo() {
super();
}
/**
* Returns address for customer.
* Creation date: (02/11/02 10:08:49 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerAddress() {
return customerAddress;
}
/**
* Returns city for customer.
* Creation date: (02/11/02 10:10:19 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerCity() {
return customerCity;
}
/**
* Returns country for customer.
* Creation date: (02/11/02 10:11:38 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerCountry() {
return customerCountry;
}
/**
* Returns first name of customer.
* Creation date: (02/11/02 10:28:15 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerFirstName() {
return customerFirstName;
}
/**
* Returns internet address for customer.
* Creation date: (04/02/02 12:31:05 PM)
* @return java.lang.String
*/
public String getCustomerInternetAddress() {
return customerInternetAddress;
}
/**
* Returns last name of customer.
* Creation date: (02/11/02 10:29:22 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerLastName() {
return customerLastName;
}
/**
* Returns list of customers.
* Creation date: (02/11/02 10:30:00 AM)
```

```
* @return java.lang.String
*/
public Vector getCustomerListVector() {
return customerListVector;
}
/**
* Returns middlet initial of customer.
* Creation date: (02/11/02 10:30:00 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerMiddleInitial() {
return customerMiddleInitial;
}
/**
* Returns phone number for customer.
* Creation date: (02/11/02 10:30:28 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerPhoneNumber() {
return customerPhoneNumber;
}
/**
* Returns state for customer.
* Creation date: (02/11/02 10:31:01 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerState() {
return customerState;
}
/**
* Returns zip code for customer.
* Creation date: (02/11/02 10:31:32 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerZipCode() {
return customerZipCode;
}
/**
* Sets address for customer.
* Creation date: (02/11/02 10:42:16 AM)
*/
public void setCustomerAddress(java.lang.String newCustomerAddress) {
customerAddress = newCustomerAddress;
}
/**
* Set city for customer.
* Creation date: (02/11/02 10:53:31 AM)
*/
public void setCustomerCity(java.lang.String newCustomerCity) {
customerCity = newCustomerCity;
}
/**
* Sets country for customer.
* Creation date: (02/11/02 10:57:06 AM)
*/
public void setCustomerCountry(java.lang.String newCustomerCountry) {
```

```
customerCountry = newCustomerCountry;
}
/**
* Sets first name of customer.
* Creation date: (02/11/02 10:57:31 AM)
*/
public void setCustomerFirstName(java.lang.String newCustomerFirstName) {
customerFirstName = newCustomerFirstName;
}
/**
* Sets internet address for customer.
* Creation date: (04/02/02 12:32:18 PM)
* @param newCustomerInternetAddress java.lang.String
*/
public void setCustomerInternetAddress(String newCustomerInternetAddress) {
customerInternetAddress = newCustomerInternetAddress;
}
/**
* Sets last name of customer.
* Creation date: (02/11/02 10:57:57 AM)
*/
public void setCustomerLastName(java.lang.String newCustomerLastName) {
customerLastName = newCustomerLastName;
}
/**
* Sets list of customers.
* Creation date: (02/11/02 10:58:12 AM)
*/
public void setCustomerListVector(Vector newCustomerListVector) {
customerListVector = newCustomerListVector;
}
/**
* Sets middle initial of customer.
* Creation date: (02/11/02 10:58:39 AM)
*/
public void setCustomerMiddleInitial(java.lang.String newCustomerMiddleInitial) {
customerMiddleInitial = newCustomerMiddleInitial;
}
/**
* Sets phone number for customer.
* Creation date: (02/11/02 10:58:39 AM)
*/
public void setCustomerPhoneNumber(java.lang.String newCustomerPhoneNumber) {
customerPhoneNumber = newCustomerPhoneNumber;
}
/**
* Sets state for customer.
* Creation date: (02/11/02 10:58:52 AM)
*/
public void setCustomerState(java.lang.String newCustomerState) {
customerState = newCustomerState;
}
/**
* Sets zip code for customer.
* Creation date: (02/11/02 10:59:08 AM)
*/
```

```
public void setCustomerZipCode(java.lang.String newCustomerZipCode) {
customerZipCode = newCustomerZipCode;
}
}
```

Capítulo 6. Descrição geral da interoperacionalidade entre o WebSphere(R) Application Server e o Lotus(R) Domino(TM)

O cenário de viagens aéreas do Teste do Sistema do iSeries(TM) consistia num terminal do WebSphere Application Server com as bases de dados do Lotus Domino no programa emissor. Como resultado, necessitávamos garantir que o terminal do WebSphere Application Server pudesse trabalhar com o programa emissor do Lotus Domino. Este procedimento incluía a capacidade de comunicar entre os dois ambientes, partilhar um início de sessão único (SSO - single sign-on) através dos ambientes e garantir a segurança no âmbito de cada um dos ambientes.

Para comunicar entre os dois ambientes, a aplicação de viagens aéreas utilizou os programas do WebSphere Application Server escritos em Java(TM) que utilizaram as APIs de Java do Lotus Domino. O fluxo de trabalho do Lotus Domino geriu o preenchimento dos voos. Os serviços de transacção do WebSphere Application Server foram utilizados para implementar o sítio da Web dos voos de clientes.

A aplicação de viagens aéreas utiliza o directório de LDAP (Lightweight Directory Access Protocol) do Lotus Domino. O directório de LDAP é necessário para implementar a segurança do WebSphere Application Server. O LDAP permite aos clientes dos voos efectuarem a autenticação utilizando os números e palavras-passe de cliente. A aplicação de viagens aéreas valida a identidade do utilizador para permitir o acesso às informações de cliente e pagamento.

A aplicação de viagens aéreas utiliza a função de início de sessão que amplia o WebSphere Application Server e o Lotus Domino. Este procedimento permite que a aplicação de viagens aéreas aceda a qualquer dos programas requerendo que o utilizador inicie sessão apenas uma vez. O que significa que quando uma transacção é iniciada a partir do WebSphere Application Server para a base de dados do Lotus Domino, são utilizadas as mesmas credenciais de início de sessão. O contrário também é possível, mas a aplicação de viagens aéreas não tinha necessidade de que o Lotus Domino acedesse ao WebSphere Application Server. O objectivo de um início de sessão único consiste em fornecer um fluxo contínuo de informações através dos produtos.

WebSphere(R) Application Server e Lotus(R) Domino(TM) início de sessão único de interoperacionalidade

Para a aplicação de viagens aéreas a segurança é uma preocupação. O WebSphere Application server e o Lotus Domino fornecem suporte para protecção de acesso e de dados. Ambos os produtos implementam mecanismos de segurança que envolvem a determinação e a verificação da identidade do utilizador (autenticação), bem como a permissão de acesso de utilizadores designados (autorização) a recursos protegidos.

Ao utilizar o início de sessão único (SSO - single sign-on), os utilizadores da Web de viagens aéreas podem efectuar uma vez a autenticação no WebSphere Application Server e, em seguida, aceder ao Lotus Domino sem ter de iniciar sessão novamente. Isto é possível através da configuração do Lotus Domino e do WebSphere Application Servers para partilharem informações de autenticação reunidas a partir do único servidor LDAP do Lotus Domino.

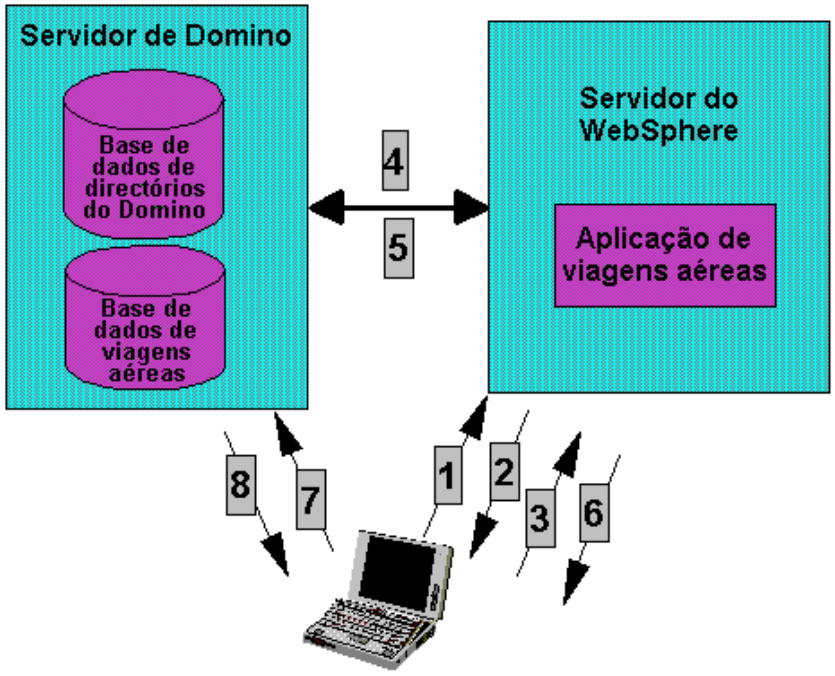
Para activar o SSO entre servidores, é utilizado o mecanismo LTPA (Lightweight Third Party Authentication). Este mecanismo utiliza um LTPAToken que contém as informações de autenticação do

utilizador, o domínio de rede no qual o SSO é válido e a data de expiração. O LTPAToken é codificado através da utilização das chaves do LTPA que devem ser partilhadas para todos os servidores que participam no SSO.

O testemunho é enviado ao utilizador da Web num cookie. Este cookie encontra-se na memória do navegador, não é armazenado no computador do utilizador e expira quando o utilizador fecha o navegador.

A activação do SSO no WebSphere requer a configuração do ficheiro EAR (Enterprise Application Resource) para segurança e configuração das definições globais de segurança no WebSphere Application Server. A figura 1 mostra o processo de início de sessão na aplicação de viagens aéreas entre o Lotus Domino e o WebSphere Application Server.

Figura 1 SSO do Lotus Domino e WebSphere



1. Um utilizador da Web apresenta um pedido ao servidor da Web (servidor de HTTP) de um recurso protegido, para obter a página inicial.
2. O servidor da Web solicita as informações de autenticação ao utilizador.
3. O utilizador responde fornecendo as informações (palavra-passe e número de cliente).
4. Em seguida, o servidor da Web entra em contacto com o servidor de LTPA (WebSphere Application Server), que por sua vez estabelece ligação ao Directório do Lotus Domino para verificar as informações de autenticação.
5. Se as informações fornecidas ao utilizador estiverem correctas, o Lotus Domino responde ao servidor (WebSphere Application Server) com as informações validadas.
6. O servidor utiliza os valores de retorno para verificar se o utilizador tem acesso ao recurso pedido e envia um testemunho de LTPA ao utilizador. O servidor da Web envia o testemunho ao utilizador como um cookie de HTTP, que é armazenado no navegador do utilizador, e está ao serviço do recurso pedido (index.html).

7. Após a autenticação do utilizador e da disponibilização do cookie, é possível pedir outro recurso protegido a partir do Lotus Domino ou do WebSphere Application Server.
8. O Lotus Domino e o WebSphere Application Server validam o testemunho fornecido ao utilizador e pedem ao navegador que envie o recurso pedido para o servidor, desde que o utilizador tenha acesso suficiente a esse recurso, sem pedir novamente as informações de desafio (challenge).

Configurar o protocolo IIOP no Lotus Domino

Os objectos remotos do Lotus Domino utilizam o CORBA (Common Object Request Broker Architecture) para implementar o acesso ao Lotus Domino. No CORBA, a comunicação entre objectos ocorre através de ORBs (Object Request Brokers) que utilizam o protocolo IIOP (Internet Inter-ORB Protocol) para enviar mensagens entre si. No Lotus Domino, o serviço do protocolo DIIOP (Lotus Domino Internet Inter-ORB Protocol) é utilizado para a comunicação CORBA. A vantagem da utilização de objectos remotos é que o servidor da aplicação da Web pode ser executado numa máquina separada a partir do servidor Lotus Domino. O código importa o pacote lotus.domino.*. O método NotesFactory() requer um endereço de IP ou nome do servidor Lotus Domino e um ID do utilizador e perfil válidos que tenham acesso ao IIOP e ao servidor Lotus Domino. O servidor remoto Lotus Domino necessita de ter o DIIOP configurado e o ID do utilizador deve ser autorizado para utilizar o servidor Lotus Domino e para executar os agentes de IIOP.

A aplicação de viagens aéreas utilizou objectos remotos do Lotus Domino quando utilizou as APIs Lotus Domino Java(TM) pelos seguintes motivos:

- Os requisitos do tempo de execução e de configuração são mais simples
- Não existem restrições relativamente à utilização em ambientes com execução de vários threads

Para configurar o servidor Lotus Domino para CORBA, execute os seguintes passos:

1. Edite o ficheiro notes.ini e adicione as tarefas às listas de tarefas especificadas no parâmetro ServerTasks. Por exemplo:
   ServerTasks=<outras tarefas>, HTTP, DIIOP
2. Abra o documento do servidor Lotus Domino para editar e escolher o separador Ports -> Internet Ports -> IIOP (Portas -> Portas Internet -> IIOP). Defina o número da porta e active-a. O número da porta assumido é 63149 para uma porta de IIOP que utiliza TCP/IP e 63148 para a porta de IIOP que utiliza SSL. Além disso, é possível especificar se permite o nome e a palavra-passe, bem como o acesso anónimo.
3. Configure o número de threads e o valor do tempo de espera esgotado. O valor do tempo de espera esgotado representa o número de minutos que uma ligação pode estar inactiva antes de ser desactivada pelo servidor. Para definir o valor do tempo de espera esgotado, escolha o separador Internet Protocols -> IIOP (Protocolos Internet -> IIOP) do documento do servidor Lotus Domino.
4. Defina a segurança de acesso ao servidor Lotus Domino e de execução de programas Java no servidor. Na secção Server access (Acesso ao servidor), especifique quem pode aceder ao servidor. Se deixar o campo em branco, todos os utilizadores poderão aceder ao servidor. Se o campo tiver nomes listados, apenas os utilizadores ou os grupos especificamente listados poderão aceder ao servidor. Se for necessário que os utilizadores iniciem sessão, estas informações serão verificadas pelo servidor após a respectiva autenticação.
5. Na secção Java/COM Restrictions (Restrições de Java/COM) do documento do servidor Lotus Domino, especifique os utilizadores que têm permissão de acesso aos objectos do Lotus Domino utilizando o CORBA. Se deixar este campo em branco, ninguém poderá aceder aos objectos do Lotus Domino utilizando o CORBA. O campo aceita caracteres globais. Um asterisco (*) neste campo permite todos os utilizadores.
6. Se o servidor Lotus Domino estiver protegido por firewall, edite o Notes(TM).ini do servidor Lotus Domino e adicione a linha DIIOP_IOR_HOST=ipAddress. Aqui, ipAddress é o endereço de IP do servidor Lotus conforme é conhecido fora da firewall.

Configurar a segurança da aplicação de viagens aéreas

Antes de configurar o WebSphere Application Server ou o Lotus Domino, necessita de configurar a segurança na aplicação e instalá-la. Para obter informações sobre a configuração da segurança de uma aplicação, consulte o Redbook da IBM(R), IBM WebSphere V4.0 Advanced Edition Security. Os capítulos em Securing Web Components e Securing Enterprise Bean Components foram acompanhados para proteger a aplicação de viagens aéreas no WebSphere Studio Application Developer.
Na aplicação de viagens aéreas, foi utilizada a autenticação baseada em formulário. Este mecanismo de autenticação permitiu que o sítio de viagens aéreas especificasse uma página de início de sessão HTML específica do sítio. Ao utilizar a autenticação baseada em formulário, a palavra-passe não é codificada e o servidor de destino não é autenticado, o que dá origem a um risco de segurança. Para evitar este risco de segurança, é possível utilizar o transporte protegido (SSL).

Activar o início de sessão único para o WebSphere Application Server

A configuração das Definições de Segurança Global para SSO no WebSphere envolve:

1. Iniciar a Consola do Administrador do WebSphere.
2. Seleccionar Console -> Security Center (Consola -> Centro de segurança). Esta acção apresentará as definições de segurança global para o WebSphere. Seleccione Enable Security (Activar segurança) no separador General (Geral).
3. Faça clique no separador Authentication (Autenticação) e escolha LTPA (Lightweight Third Party Authentication) como o tipo de mecanismo de autenticação.
4. Especificar as seguintes definições de LTPA:
   a. O número de minutos que pode decorrer antes que um cliente, utilizando um testemunho de LTPA, deva autenticar novamente o campo Token Expiration (Expiração do testemunho).
   b. Seleccione Enabled single sign-on (SSO) [Início de sessão único (SSO) activado]. O campo Domain (Domínio) será activado.
   c. Introduza um nome do domínio de DNS no campo Domain (Domínio). Este nome do domínio é utilizado quando o cookie de HTTP é criado para o SSO e determina o âmbito ao qual o SSO se aplica.

Importante: Todos os servidores que participam no SSO devem estar no mesmo domínio de DNS.

1. Seleccionar o selector de opção de LDAP e introduzir as definições do servidor de LDAP.

Nota: Certifique-se de que configura o WebSphere Application Server para utilizar o Lotus Domino 5.0 como o tipo de Directório no Centro de segurança. Certifique-se também de que o servidor Lotus Domino está em execução e de que a tarefa de LDAP é iniciada, uma vez que serão verificadas a palavra-passe e o ID do servidor de Segurança.

1. Fazer clique no botão Generate Keys (Gerar chaves) para criar as chaves de LTPA para codificar o testemunho de LTPA. Ser-lhe-á pedida uma palavra-passe de LTPA para proteger o conjunto de chaves de codificação. Estas chaves de LTPA devem ser partilhadas para todos os servidores que utilizam o SSO.

Importante: Lembre-se que a geração de chaves de LTPA deve ser efectuada quando as definições do servidor de LDAP do Lotus Domino são configuradas. Esta acção garante a presença do sistema central e da porta de LDAP no ficheiro exportado. O Lotus Domino necessita destas informações durante o processo de criação do documento de configuração do SSO na Web.

1. Após a geração das chaves de LTPA, fazer clique no botão Export Key (Exportar chave) para exportar as chaves de LTPA para um ficheiro. Este ficheiro é utilizado para importar as chaves para o Lotus Domino.
2. Fazer clique em Apply (Aplicar) e, em seguida, em OK.
3. Quando o processo estiver concluído será apresentada uma mensagem de aviso a indicar ″Changes will not take effect until the admin server is restarted″ (As alterações só entrarão em vigor após a reinício do servidor admin). Fazer clique em OK.

4. Reiniciar o servidor de administração seleccionando o nó incluído na pasta de nós localizada na vista em árvore no lado esquerdo da consola e, em seguida, fazer clique com o botão direito do rato na mesma e seleccionar Restart (Reiniciar) no menu contextual resultante.

Activar o início de sessão único para o Lotus Domino Server

1. Crie um novo documento de configuração do SSO na Web na base de dados do directório do Lotus Domino.
   a. Seleccione Server -> Servers (Servidor -> Servidores) para apresentar a vista. Faça clique no botão Web e seleccione Create Web SSO Document (Criar documento do SSO na Web) no menu contextual resultante.
   b. Será apresentado um novo documento com o seguinte campo Token Name (Nome do testemunho) (LTPAToken).
   c. Inclua o domínio de DNS no campo Token Domain (Domínio do testemunho). Este valor deve coincidir com o valor especificado no campo Domain (Domínio) no WebSphere Application Server. Este nome de domínio é utilizado quando o cookie de HTTP é criado para um início de sessão único e determina o âmbito ao qual se aplica o início de sessão único.
   d. Escolha os servidores Lotus Domino que participarão no cenário de SSO.

   Nota: Deve especificar um nome do servidor Lotus Domino totalmente qualificado (por exemplo, MeuServidorDomino/MeuOu). O nome do servidor Lotus Domino que especificar deve corresponder ao nome da página inicial/servidor de correio actualmente no documento Localização activo no cliente do Lotus Notes(R). Se a localização não corresponder, terá de criar uma que corresponda.

   a. Introduza o número máximo de minutos de validade do testemunho enviado nos ficheiros Expiration (minutes) [Expiração (minutos)]. Defina de modo a corresponder à data definida no WebSphere Application Server.
   b. Faça clique na seta pendente Keys (Chaves) e seleccione Import WebSphere LTPA keys (Importar chaves de LTPA do WebSphere).
   c. Especifique o caminho e o nome do ficheiro para o ficheiro de chaves de LTPA do WebSphere Application Server LTPA exportado anteriormente.
   d. Faça clique em OK. Será apresentada uma nova caixa de diálogo pedindo ao utilizador a palavra-passe de LTPA especificada durante a geração das chaves.
   e. Faça clique em OK. Quando o processo estiver concluído, será apresentada uma mensagem de confirmação.
   f. Será apresentada uma nova secção WebSphere Information (Informações do WebSphere) no documento. O nicho e sistema central de LDAP são lidos a partir do ficheiro WebSphere Application Server Import (Importação do WebSphere Application Server). Se tiver especificado uma porta na definição de configuração de LDAP do WebSphere, certifique-se de que adiciona uma barra invertida (\) antes dos dois pontos (:) no campo LDAP Realm (Nicho de LDAP).
   g. Faça clique no botão Save and Close (Guardar e fechar). O documento será guardado. Para verificar se o documento se encontra no directório do Lotus Domino, seleccione Server -> Web Configurations View (Servidor -> Vista de configurações da Web) e expanda a secção *-All Servers - (*-Todos os servidores -). O novo documento deverá ser apresentado como a configuração do SSO na Web para LtpaToken.
2. Active o estado da porta de TCP/IP no separador Ports -> Internet Ports -> Web (Portas -> Portas Internet -> Web) e não permita ligações anónimas através de TCP/IP modificando o documento do Lotus Domino Server.
3. Seleccione a sessão Multi-Server (Vários servidores) no separador Internet Protocols -> Lotus Domino Web Engine (Protocolos Internet -> Mecanismo da Web do Lotus Domino) no documento do servidor.
4. Seleccione More name variations with lower security (Mais variações de nomes com menor segurança) na secção de autenticação do servidor da Web do separador de segurança. Isto permite que os

utilizadores introduzam os seguintes formatos de nomes na caixa de diálogo do nome e palavra-passe: apelido, nome próprio, outros nomes, nome hierárquico completo, nome abreviado e pseudónimo.

A Tabela 1 mostra como o ACL da base de dados foi modificado para implementar segurança na base de dados da aplicação de viagens aéreas.

Tabela 1 SSO do Lotus Domino e do WebSphere

Pessoas, servidores e grupo	Tipo de utilizador	Nível de acesso	Autorização
Empregados da companhia aérea	Grupo de pessoas	Editor	Eliminar documentos, criar agente Script/Java da Lotus
Anónimo	Não especificado	Leitor	Escrever documentos públicos

O servidor Lotus Domino apresentará uma página de início de sessão assumida quando um utilizador tentar aceder a uma página da Web a partir da base de dados. O utilizador da Web necessitará de introduzir o nome do utilizador e a palavra-passe e fazer clique no botão Login (Iniciar sessão).

O servidor Lotus Domino verifica se o utilizador está registado na base de dados do Directório do Lotus Domino e se os valores das credenciais estão correctos. Verifica também se o utilizador tem acesso à base de dados. Após a autenticação do utilizador, o Lotus Domino cria um novo LTPAToken e envia-o ao utilizador como um cookie de HTTP e abre o documento do Lotus Domino.

Observações chave da interoperacionalidade entre o WebSphere(R) Application Server e o Lotus(R) Domino(TM)

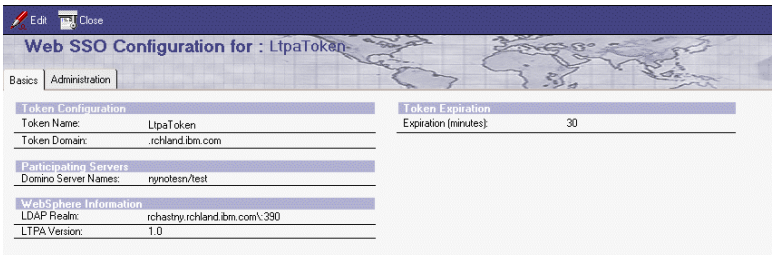
A seguir encontra-se uma lista de observações chave que foram reveladas na implementação do cenário de viagens aéreas ao lidar com a interoperacionalidade entre o WebSphere Application Server e o Lotus Domino.

- Ao criar o documento de configuração do início de sessão único (SSO - single sign-on) na Web do Lotus Domino, é necessário certificar-se de que o documento de localização do cliente Notes(TM) aponta para o servidor do Lotus Domino onde pretende permitir o SSO. Este procedimento é necessário para que uma chave pública possa ser utilizada no servidor. Se uma mensagem aparecer quando guarda o documento Web SSO Configuration (Configuração do SSO na Web) a indicar que não consegue localizar o servidor, este procedimento deve resolver o problema.

- No centro de segurança do WebSphere Application Server, o nicho (nome do domínio) tem de ser especificado em letras minúsculas.

- Uma vez que o WebSphere Application Server trata o nome de DNS como sensível a maiúsculas e minúsculas, certifique-se de que o valor do domínio de DNS é especificado como sendo exactamente o mesmo, incluindo maiúsculas e minúsculas conforme a Figura 1, sempre que utilizar este valor no Lotus Domino.

Figura 1 - SSO do Lotus Domino e WebSphere



- Certifique-se de que qualifica totalmente o URL com o nome de domínio (i.e. http://*nomeSistema.nomeDomínio:númeroPorta/uri*) ao aceder a um endereço da Web do Lotus Domino ou do WebSphere quando a segurança e o SSO estiverem configurados. Se não qualificar totalmente o URL (i.e. http://*nomeSistema:númeroPorta/uri*), ser-lhe-á devolvido o formulário de início de sessão e nunca obtém a página que está a tentar aceder.

- Quando a segurança estiver configurada no WebSphere Application Server, coloque o NCSOW.jar no directório qibm/userdata/webasadv4/*nomeInstância*/lib/ext. Uma vez que estávamos a utilizar o ficheiro NCSOW.jar para criar as nossas sessões do Lotus Domino, colocámos o ficheiro jar nas propriedades de JVM do servidor assumido. O que causou um problema quando a segurança foi activada. Com o ficheiro jar nas definições de JVM, recebíamos constantemente uma indicação de NoClassDefFound para com.ibm.ejs.oa.EJSORB. Isto acontecia devido ao NCSOW se encontrar no caminho da classe JVM. Para corrigir este problema, movemos o NCSOW.jar para o directório qibm/userdata/webasadv4/*nomeInstância*/lib/ext e removemos a entrada do caminho da classe JVM.

- Ao utilizar segurança, pode verificar se está a executar com a identidade correcta utilizando o método getCalledIdentity() no contexto de enterprise beans. Este método imprime a identidade do utilizador sob a qual está a ser executado o método dentro do enterprise bean.

- Os seguintes elementos da classe Java/CORBA (pacote lotus.domino) suportam o início de sessão nos servidores do Lotus Domino e do WebSphere num domínio de início de sessão único.
  - Propriedade SessionToken
    Só de leitura. Obtém um testemunho de sessão para activar o início de sessão nos servidores do Lotus Domino e WebSphere num domínio que suporte o início de sessão único.
    NOTA: Esta propriedade é nova com a versão R5.0.5.
    Definido em: lotus.domino.Session
    Tipo de dados: String (Cadeia)
    Sintaxe: public String getSessionToken() throws NotesException
    Utilização: O testemunho é único para cada utilizador e é válido para o tempo especificado no Lotus Domino Directory. O formato do testemunho é consistente com o cookie LtpaToken utilizado pelo WebSphere.

    Também pode obter o testemunho a partir dos cabeçalhos de HTTP num servlet com HttpServletRequest.getCookies().
    Esta propriedade é válida apenas num servidor configurado para o início de sessão único. Consulte a classe NotesFactory para obter informações de utilização e exemplos.
  - Classe NotesFactory
    NOTA: Para efectuar chamadas remotas (IIOP) para os objectos do Lotus Domino num ambiente do WebSphere, o NCSOW.jar tem de se encontrar no caminho da classe. Esta classe é nova com a versão R5.0.4.
    A descrição da classe NotesFactory é expandida conforme se segue.
    NOTA: Estas extensões são novas com a versão R5.0.5.

    Para aceder a um servidor utilizando o início de sessão único, crie um objecto Session (sessão) da

seguinte forma. Nas chamadas remotas (IIOP), o primeiro parâmetro é o nome de Internet do sistema central. Nas chamadas locais, o primeiro parâmetro é null.

- createSession(hostString, String token) - O acesso é concedido com base no token (testemunho). Este método funciona num ambiente do Lotus Domino. O testemunho tem de ser válido para o início de sessão único obtido a partir do cookie LtpaToken ou Session.getSessionToken utilizado pelo WebSphere.
- createSession(hostString, org.omg.SecurityLevel2.Credentials) - O acesso baseia-se no objecto Credentials (Credenciais). Este método funciona num ambiente do WebSphere onde o objecto Credentials é criado utilizando o loginHelper.
- createSession(hostString, null) - O acesso é concedido com base no objecto Credentials actual no ambiente do WebSphere. Este método funciona a partir de uma aplicação de enterprise beans no WebSphere.

A especificação do NotesFactory expande-se com os seguintes métodos:

- static public Session createSession(String host, String token) throws NotesException
- static public Session createSession(String host, org.omg.SecurityLevel2.Credentials) throws NotesException

– Exemplos

- **Exemplo 1:** Este agente do Lotus Domino obtém um testemunho para o início de sessão único e cria uma sessão remota (IIOP) para outro servidor com base no testemunho.

```
import lotus.domino.*;
public class JavaAgent extends AgentBase {

public void NotesMain() {
try {
Session session = getSession();
AgentContext agentContext = session.getAgentContext();
Session s2 = NotesFactory.createSession("test5.iris.com",
session.getSessionToken());
System.out.println("nome da sessão remota = " + s2.getUserName());
} catch(Exception e) {
e.printStackTrace();
}
}
}
```

- **Exemplo 2:** Este servlet obtém um testemunho para o início de sessão único a partir do cookie LTPAToken através de HttpServletRequest e cria uma sessão com base no testemunho.

```
import java.lang.*;
import java.lang.reflect.*;
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;

public class Cookies extends HttpServlet
{

private void respond(HttpServletResponse response, String entity) throws IOException
{
response.setContentType("text/plain");
```

```
if (entity == null)
{ response.setContentLength(0);}
else {
response.setContentLength(entity.length() + 1);
ServletOutputStream out = response.getOutputStream();
out.println(entity);
}
}

public void doGet (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
String s1 = "";
Cookie[] cookies = null;
String sessionToken = null;

try {
cookies = request.getCookies();
}
catch (Exception e) {
respond(response,"Excepção de request.getCookies(): " + e.toString());
return;
}

if (cookies == null) {
s1 = "Não foram recebidos cookies";
}
else {
for (int i = 0; i < cookies.length; i++) {
if (cookies[i].getName().equals("LtpaToken")) {
sessionToken = cookies[i].getValue();
}
}
}

if (sessionToken != null) {
try {
NotesThread.sinitThread();
Session session = NotesFactory.createSession(null, sessionToken);
s1 += "\n" + "Servidor: " + session.getServerName();
s1 += "\n" + "EstáNoServidor: " + session.isOnServer();
s1 += "\n" + "NomeUtilizadorComum: " + session.getCommonUserName();
s1 += "\n" + "NomeUtilizador: " + session.getUserName();
s1 += "\n" + "VersãoNotes: " + session.getNotesVersion();
s1 += "\n" + "Plataforma: " + session.getPlatform();
NotesThread.stermThread();
}
catch (NotesException e) {
s1 += "\n" + e.id + e.text;
e.printStackTrace();
}
}
```

```
respond(response,s1);
}
}
```

- **Exemplo 3:** Este extracto da aplicação cria uma sessão com base num objecto de credenciais obtido a partir do WebSphere.

```
com.ibm.CORBA.iiop.ORB orb = com.ibm.ejs.oa.EJSORB.getORBInstance();

if (orb != null) {
org.omg.SecurityLevel2.Current(R) securityCurrent =
(org.omg.SecurityLevel2.Current)orb.resolve_initial_references("SecurityCurrent");
org.omg.SecurityLevel2.Credentials invCred =
securityCurrent.get_credentials(org.omg.Security.CredentialType.SecInvocationCredentials);
System.out.println("criar sessão do Notes utilizando credenciais actuais");
session = NotesFactory.createSession(notesServer, invCred);
}
```

- **Exemplo 4:** Esta aplicação de enterprise beans do WebSphere cria uma sessão com base no objecto de credenciais actual no ambiente do WebSphere.

```
import lotus.domino.*;

public class HelloBean extends Object implements SessionBean {

... /* Consulte o HelloBean.java a partir do WebSphere para obter o código de classe completo */

/**
Devolve a saudação. Mas foi modificado para criar uma sessão remota no
servidor do Lotus Domino.
@return A saudação.
@exception RemoteException Iniciada se falhar a chamada de método remoto.
*/
public String getMessage () throws RemoteException
{
String result = "hello bean ";

try {
Session s = NotesFactory.createSession("test5.iris.com", null);
result = result + " — Obteve sessão para " + s.getUserName();
}
catch (NotesException ne) {
result = result + "— " + ne.text;
result = result + "— falha na obtenção de sessão para o utilizador";
}

return (String) result + " — concluído";
}
}
```

Capítulo 7. Referências

Os seguintes recursos fornecem informações que lhe poderão ser úteis.

- IBM(R) WebSphere V4.0 Advanced Edition Security, IBM Redbook SG24-6520-00
- Lotus Domino and WebSphere Integration on the IBM eServer iSeries(TM) Server, IBM Redbook SG24-6223
- Lotus Domino and WebSphere Together Second Edition, IBM Redbook SG24-5955-01
- Security Guide
  http://www.ibm.com/software/webservers/appserv/doc/v40/aes/infocenter/was/pdf/nav_Securityguide.pdf
- WebSphere Interoperability
  http://www-106.ibm.com/developerworks/webservices/library/ws-bpinter/