IBM WebSphere Commerce

程序员指南

版本 5.4

G152-0293-00

IBM WebSphere Commerce

程序员指南

版本 5.4

G152-0293-00

注意:

在使用本资料及其支持的产品之前，请务必阅读『声明』中的信息。

第一版（2002 年 3 月），发行版 2

本版本适用于下列产品:

- IBM® WebSphere® Commerce 商务版 Windows NT® 和 Windows® 2000 版版本 5.4（程序 5724-A18）
- IBM WebSphere Commerce 商务版 AIX® 版版本 5.4（程序 5724-A18）
- IBM WebSphere Commerce 商务版 Solaris Operating Environment Software 版版本 5.4（程序 5724-A18）
- IBM WebSphere Commerce 商务版 Linux 版，版本 5.4（程序 5724-A18）
- IBM WebSphere Commerce Studio 商务开发者版 Windows NT 和 Windows 2000 版版本 5.4（程序 5724-A18）
- IBM WebSphere Commerce 专业版 Windows NT 和 Windows 2000 版版本 5.4（程序 5724-A18）
- IBM WebSphere Commerce 专业版 AIX 版版本 5.4（程序 5724-A18）
- IBM WebSphere Commerce 专业版 Solaris Operating Environment Software 版版本 5.4（程序 5724-A18）
- IBM WebSphere Commerce 专业版 Linux 版，版本 5.4（程序 5724-A18）
- IBM WebSphere Commerce Studio 专业开发者版 Windows NT 和 Windows 2000 版版本 5.4（程序 5724-A18）

以及上述产品的所有后续发行版和修订版，直到在新版本中另有声明为止。确认您正在使用本产品级别的正确版本。

通过您当地的 IBM 代表或 IBM 分部可订购出版物。以下地址不备有出版物。

IBM 欢迎您提出宝贵意见。您可以将关于本出版物的意见通过以下一种方式寄给我们:

1. 可以通过电子的形式发送到以下电子邮件地址:

   `torrcf@ca.ibm.com`

2. 邮寄到以下地址:

   IBM Canada Ltd. Laboratory
   B3/KB7/8200/MKM
   8200 Warden Avenue
   Markham, Ontario, L6G 1C7
   Canada

当您发送信息给 IBM 后，即授予 IBM 非专有权，IBM 可以它认为合适的任何方式使用或分发此信息，而无须对您承担任何责任。

开始之前

《*IBM WebSphere Commerce 程序员指南*》提供了有关 WebSphere Commerce 体系结构和编程模型的信息。它还特别提供了关于以下主题的详细信息:

- 组件交互
- 设计模式
- 持久对象模型
- 访问控制
- 错误处理和消息
- 命令实现
- 开发工具
- 定制代码的部署

此外，本书还包括以下教程:

创建新业务逻辑

此教程演示了如何按照 WebSphere Commerce 编程模型创建新命令、数据 bean 和企业 bean。它还演示了如何将逻辑集成到现有商店中去，并将代码部署到目标 WebSphere Commerce Server 中。

修改并扩展现有的业务逻辑

此教程分成两部分。第一部分演示了如何将新逻辑添加到现有控制器命令。第二部分演示了如何修改现有任务命令和 WebSphere Commerce 实体 bean。它还演示了如何将修改集成到现有商店中去，并将代码部署到目标 WebSphere Commerce Server 中。

本书中使用的约定

本书使用以下突出显示的约定:

粗体字表示命令或图形用户界面（GUI）控件，如字段名、按钮或菜单选项。

等宽字表示完全按显示原样输入的文本示例和目录路径。

*斜体字*用于表示强调和可用自己的值替换的变量。

此图标用于标记技巧 — 可帮助完成任务的附加信息。

Windows 表示特定于 WebSphere Commerce Windows NT 和 Windows 2000 版的信息。

AIX 表示特定于 WebSphere Commerce AIX 版的信息。

Solaris 表示特定于 WebSphere Commerce Solaris™ Operating Environment software 版的信息。

400 表示特定于 WebSphere Commerce IBM @server iSeries™ 400®（以前称为 AS/400®）版的信息

Linux 表示特定于 WebSphere Commerce Linux 版的信息。

DB2 表示特定于 DB2® 通用数据库的信息

Oracle 表示特定于 Oracle® 的信息。

Professional 表示特定于 WebSphere Commerce 专业版的信息。

Business 表示特定于 WebSphere Commerce 商务版的信息。

必备知识

需要了解如何定制 WebSphere Commerce 应用程序的商店开发者应阅读本书。执行程序扩展的商店开发者应当具有以下领域的知识:

- Java™
- Enterprise JavaBeans（EJB）组件体系结构
- JavaServer Pages 技术
- HTML
- 数据库技术
- VisualAge® for Java 企业版，版本 3.5

何处可以找到更多信息

本书以后可能会更新。检查以下 WebSphere Commerce Web 站点以获得更新:

Business

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

Professional

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

更新可能包括与教程相关的新信息、附加或更新的教程和样本代码。

目录

第 1 部分 概念和体系结构

第 1 章 概述

WebSphere Commerce 软件组件

在了解如何运行 WebSphere Commerce Server 之前，从宏观上查看与定制过程相关的软件组件会有些帮助。下图显示了这些软件产品的简化视图:

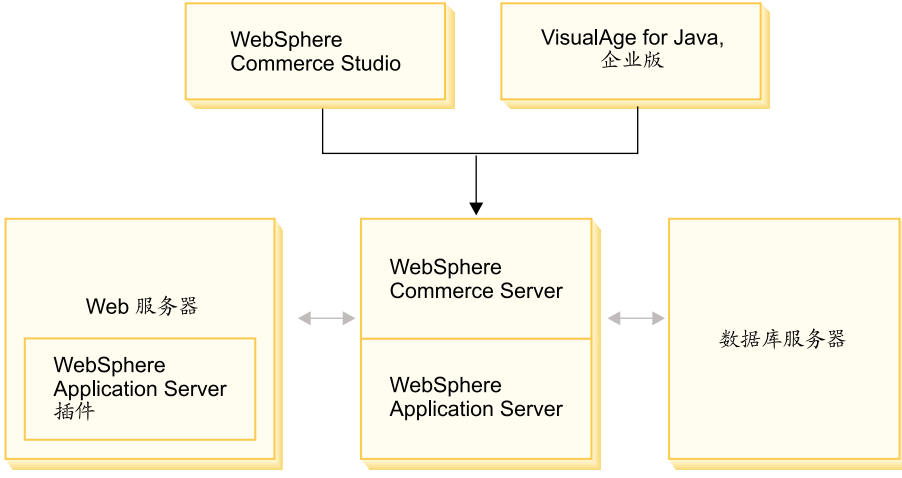


图 1.

Web 服务器是电子交易应用程序对进入 HTTP 请求的第一个联系点。为了有效地与 WebSphere Application Server 交互，它使用 WebSphere Application Server 插件。

WebSphere Commerce Server 在 WebSphere Application Server 中运行，使它能够利用应用程序服务器的许多功能。数据库服务器保留应用程序的大部分数据，包括产品数据和购物者数据。一般说来，通过修改或扩展 WebSphere Commerce Server 的代码来扩展您的应用程序。另外，您可能需要将超出 WebSphere Commerce 数据库模式域的数据存储在您的数据库中。

开发者使用两种主要工具来创建定制的业务逻辑: WebSphere Commerce Studio 和 VisualAge for Java 企业版。WebSphere Commerce Studio 用于创建和管理商店前台有用资源（例如，JSP 模板）。VisualAge for Java 企业版用于创建新的业务逻辑（Java 格式）来扩展现有的功能或创建全新功能。如果您的应用程序需要扩展数据库模式，则数据库开发者应使用他们喜欢的数据库开发工具来创建新表。

WebSphere Commerce 应用程序体系结构

既然您已经知道了与定制有关的各种软件组件的组合方式，那么了解应用程序体系结构就十分重要。它将帮助您了解哪些部分是基础层，哪些部分可以修改。下图显示了组成应用程序体系结构的各层:

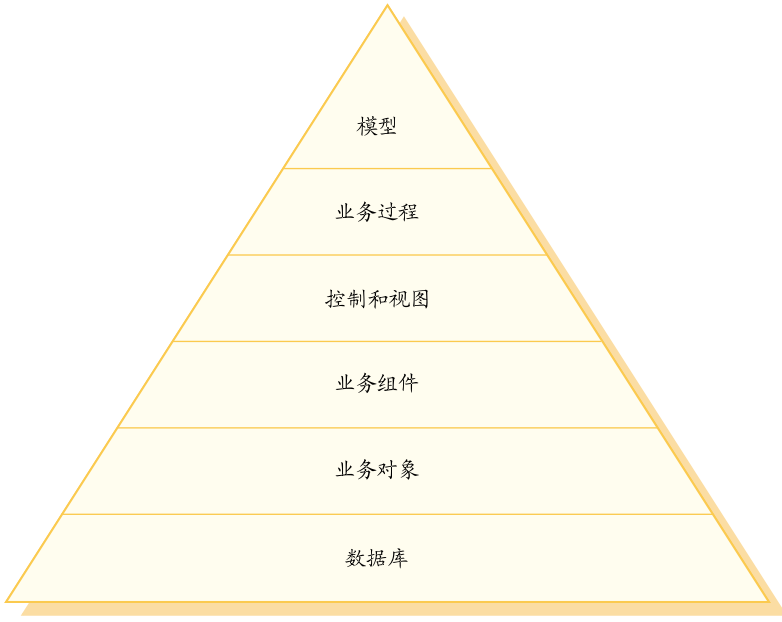


图 2.

应用程序体系结构的每一层描述如下:

数据库 WebSphere Commerce 使用专门为电子交易应用程序及其数据需求而设计的数据库模式。以下是这个模式中的表的示例。

- 用户
- 订单
- 产品

业务对象

业务对象代表商业领域中的实体，并封装以数据为中心的逻辑，此逻辑在抽取和解释数据库中包含的信息时是必需的。这些实体遵循 Enterprise JavaBeans 规范。

这些实体 bean 充当商业应用程序和数据库之间的接口。另外，与数据库表中列之间的复杂关系相比，实体 bean 更容易理解。

业务组件

业务组件是业务逻辑单元。它们执行粗粒度的过程性业务逻辑。此逻辑是使用控制器命令和任务命令的 WebSphere Commerce 模型来实现的。该类型组件的示例是 OrderProcess 控制器命令。这个特殊命令封装处理一般订单所需的所有业务逻辑。电子交易应用程序调用 OrderProcess 命令，该命令反过来调用若干任务命令来执行单独的工作单元。例如，单个的任务命令确保有足够的库存来满足订单需求、处理支付、更新订单状态以及在过程完成时适当地扣减库存数量。

控制和视图

Web 控制器可以确定要使用的适当的控制器命令实现和视图。实现可以是特定于商店的。

视图显示了命令和用户操作的结果。它们是使用 JSP 模板实现的。视图的示例包括 ProductDisplay（返回显示购物者选定产品相关信息的产品页面）和 OrderPrepare（向购物者显示表单以提交相应的订单信息）。

业务过程

业务组件和视图的集合共同创建了称为“业务过程”的工作流和站点流过程。业务过程的示例包含:

用户注册

此业务过程包含业务组件（例如，为新用户创建注册记录的 UserRegistrationAdd 命令）和与注册用户过程中所涉及的所有步骤相关的视图。

产品目录导航

此业务过程包含业务组件（例如，分别显示商店产品目录和该产品目录中类别的 StoreCatalogDisplay 和 CategoryDisplay 命令）和与浏览产品目录过程中所涉及的所有步骤有关的视图。

模型 图表的下面几层组合在一起便形成了电子交易业务模型。电子交易业务模型的一个示例是“流行时尚”样本商店中使用的“商家到消费者”模型。另一个示例是“工具技术”样本商店中使用的“商家到商家”模型。

WebSphere Commerce 运行时体系结构

前一部分介绍了应用程序体系结构，从业务应用程序观点描述了 WebSphere Commerce 应用程序中的各个层次。本部分将描述如何实现运行时体系结构。

WebSphere Commerce 运行时体系结构的主要组件有:

- 小服务程序引擎
- 协议侦听器
- 适配器管理器

- 适配器
- Web 控制器
- 命令
- 实体 bean
- 数据 bean
- 数据 bean 管理器
- 显示页面
- XML 文件

下图显示了 WebSphere Commerce 组件之间的相互关系。关于每个组件的更详细信息，请参阅后面的章节。

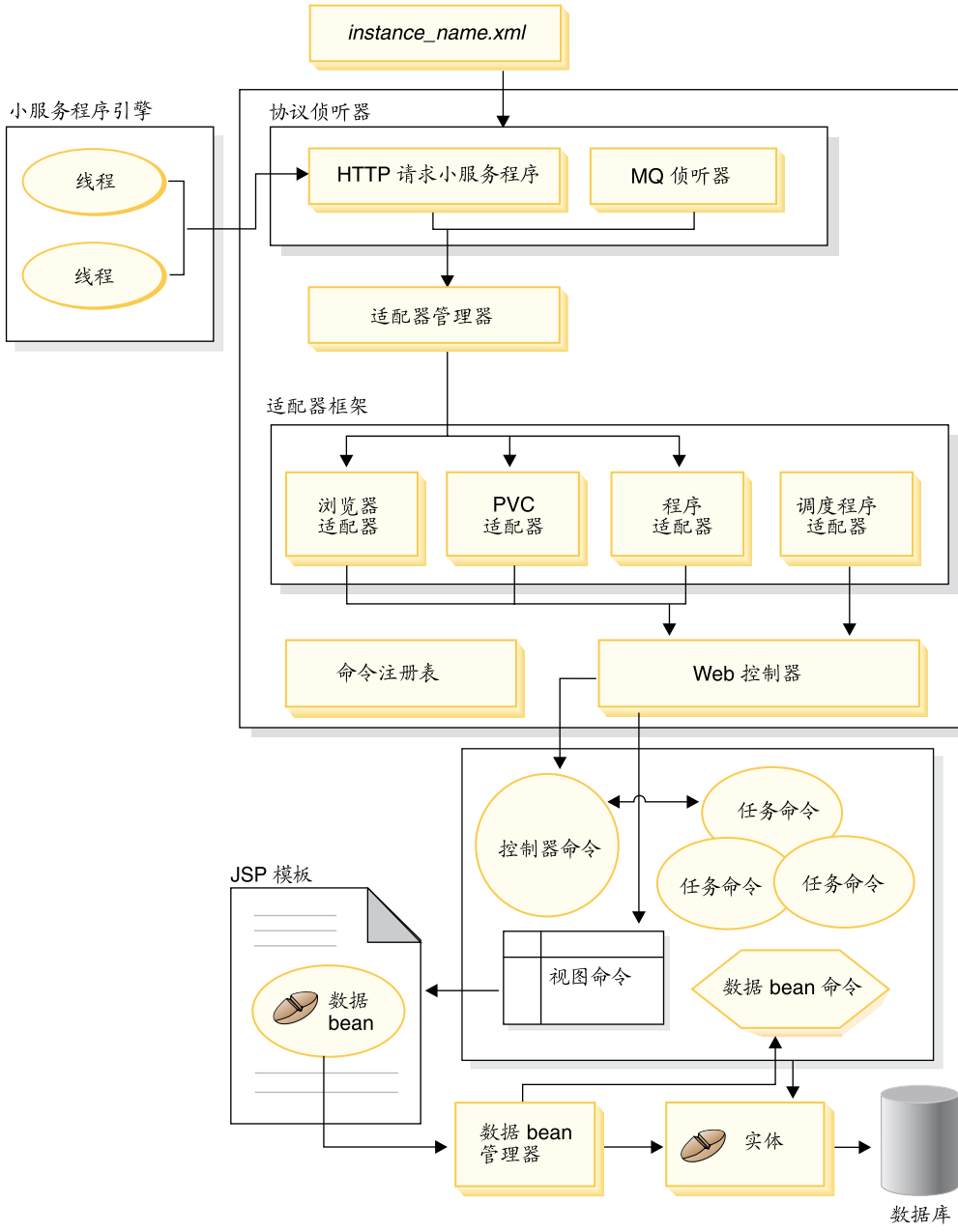

instance_name.xml
小服务程序引擎
线程
线程
协议侦听器
HTTP 请求小服务程序
MQ 侦听器
适配器管理器
适配器框架
浏览器
适配器
PVC
适配器
程序
适配器
调度程序
适配器
命令注册表
Web 控制器
控制器命令
任务命令
任务命令
任务命令
JSP 模板
数据
bean
视图命令
数据 bean 命令
数据 bean
管理器
实体
数据库

图 3.

小服务程序引擎

小服务程序引擎是 WebSphere Application Server 运行时环境的一个部件，它充当入站 URL 请求的请求分派器。小服务程序引擎管理线程池来处理请求。在独立的线程上执行各个入站请求。

WebSphere Commerce 的先前版本使用 C++ 应用程序服务器，该服务器通过维护预分配的系统进程集来实现自己对 URL 请求的任务分派器。这种使用系统进程的旧模型比使用 Java 线程的新模型对资源的使用程度更高。新的 WebSphere Commerce 运行时体系结构通过利用小服务程序引擎来提供伸缩性更好的商务解决方案。

适配器管理器

适配器管理器确定哪个适配器可以处理请求，然后将该请求转发到该适配器。

协议侦听器

WebSphere Commerce 命令可以从各种设备调用。可调用命令的设备示例包括:

- 典型的因特网浏览器
- 使用因特网浏览器的移动式电话
- 使用 MQSeries® 发送 XML 消息的商家到商家应用程序
- 使用 HTTP 上的 XML 发送请求的采购系统
- 执行后台作业的 WebSphere Commerce 调度程序

这些设备可以使用不同的通信协议。协议侦听器是运行时组件，它根据使用的协议，接收来自传送系统的入站请求，然后将这些请求发送到相应适配器。协议侦听器包括:

- 请求小服务程序
- MQSeries 侦听器

当请求小服务程序接收到来自小服务程序引擎的 URL 请求时，它便将请求传递给适配器管理器。然后，适配器管理器查询适配器类型，以确定哪个适配器可以处理此请求。一旦确定了特定适配器，请求就传递至该适配器。

初始化请求小服务程序时，它将读取 *instance_name*.xml 配置文件。XML 文件中的配置块中的一个块将定义所有适配器。请求小服务程序的 init() 方法初始化所有定义的适配器。

MQSeries 侦听器接收来自远程程序的基于 XML 的 MQSeries 消息，并将请求分派给非 HTTP 适配器管理器。

作业调度程序不需要协议侦听器。

适配器

WebSphere Commerce 适配器是在将请求传递到 Web 控制器之前执行处理功能的特定设备组件。由适配器执行的处理任务示例包含:

- 指导 Web 控制器以特定于设备类型的方式处理请求。例如，普及计算（PvC）设备适配器可指示 Web 控制器忽略初始请求中的 HTTPS 检查。
- 将入站请求的消息格式转换为 WebSphere Commerce 命令可以分析的一系列属性。
- 提供特定于设备的会话持久性。

下图显示了 WebSphere Commerce 适配器框架的实现类层次结构。

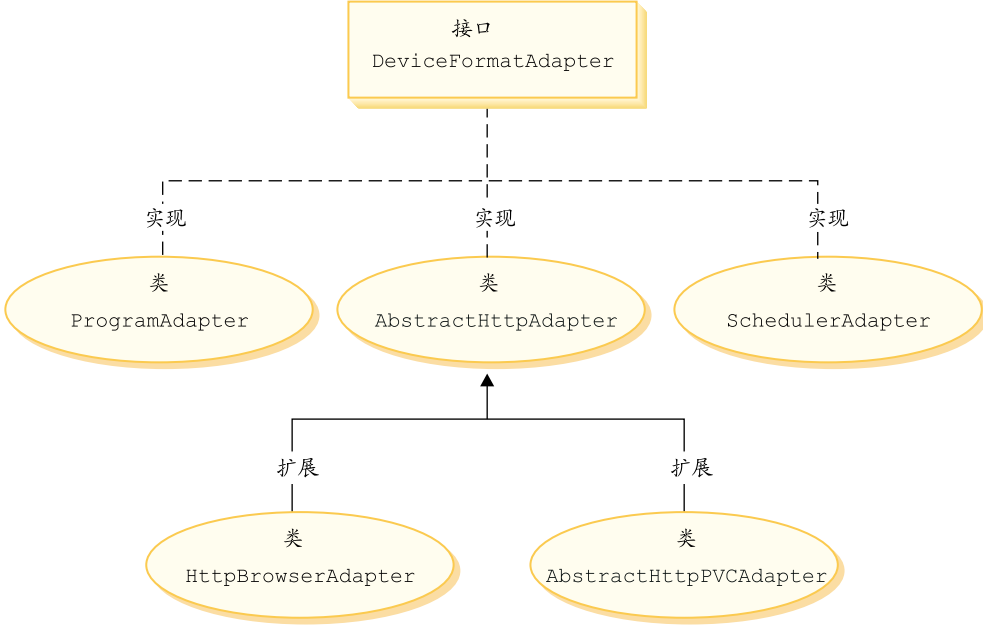


图 4.

如上图显示，所有适配器都可实现 DeviceFormatAdapter 接口。以下是 WebSphere Commerce 运行时环境所用的适配器:

程序适配器

程序适配器提供了对调用 WebSphere Commerce 命令的远程程序的支持。程序适配器接收请求，并使用消息映射器将请求转换成 CommandProperty 对象。转换后，程序适配器使用 CommandProperty 对象，并执行该请求。

调度程序适配器

调度程序适配器提供了对作为后台作业运行的 WebSphere Commerce 命令的支持。

HTTP 浏览器适配器

HTTP 浏览器适配器提供了对请求的支持，使其可以调用从 HTTP 浏览器接收的 WebSphere Commerce 命令。

HTTP PvC 适配器

这种抽象适配器类可用于开发特定 PvC 设备适配器。例如，如果您需要为一个特定的蜂窝式电话应用程序开发适配器，您可以从该适配器中进行扩展。

需要时，适配器框架可以在以下两方面扩展:

- 创建用于特定 PvC 设备的适配器（例如，创建 HttpIModePVCAdapterImpl 类来提供对 I 方式设备的支持）。此类型的适配器必须扩展 AbstractHttpAdapterImpl 类。
- 创建连接到新协议侦听器的新适配器。这个新适配器必须实现 DeviceFormatAdapter 接口。

Web 控制器

WebSphere Commerce Web 控制器是一个应用程序容器，它遵循的设计模式与 EJB 容器的设计模式相似。此容器通过提供如会话管理（根据由适配器建立的会话持久性）、事务控制、访问控制和认证等服务，简化命令的角色。

Web 控制器也起着增强商业应用程序的编程模型功能的作用。例如，编程模型定义应用程序应写入的命令类型。每种类型的命令有其特定作用。业务逻辑必须使用控制器命令实现，而视图逻辑必须使用视图命令实现。Web 控制器希望控制器命令返回一个视图名称。如果未返回视图名称，则会抛出异常。

对于 HTTP 请求，Web 控制器执行以下任务:

- 从 javax.transaction 数据包使用 UserTransaction 接口开始处理事务。
- 从适配器获取会话数据。
- 确定用户是否必须在调用命令前登录。如果需要，它将用户浏览器重定向至登录 URL。
- 检查 URL 是否需要安全 HTTPS。如果需要，但当前请求并未使用 HTTPS，则它将 Web 浏览器重定向至 HTTPS URL。
- 调用控制器命令并将命令上下文和输入属性对象传递给它。
- 如果发生事务回滚异常，且可以重试控制器命令，则它重试控制器命令。

- 当需要向客户机发送回一个视图命令时，控制器命令通常会返回视图名称。Web 控制器对相应的视图调用视图命令。有多种方式可用于形成响应视图。这些方式包含重定向到不同的 URL、转发至 JSP 模板或将 HTML 文档写入响应对象。
- 保存会话数据。
- 提交会话数据。
- 如果当前事务成功，则将其提交。
- 故障时回滚当前事务（取决于环境）。

命令

WebSphere Commerce 命令是包含与处理特定请求相关联的编程逻辑的 bean。

WebSphere Commerce 命令有四种主要类型:

控制器命令

控制器命令将与特定业务过程相关的逻辑封装起来。举例来说，控制器命令可以包括: 用于订单处理的 *OrderProcessCmd* 命令和用于创建新注册用户的 *UserRegistrationAddCmd* 命令。通常，控制器命令包含控制语句（例如，if、then 和 else），并调用任务命令以便执行业务过程中的单个任务。完成时，控制器命令将返回一个视图名称。然后，Web 控制器将确定视图命令相应的实现类，并执行该视图命令。

任务命令

任务命令实现应用程序逻辑的特定部分。通常，控制器命令和一组任务命令共同实现 URL 请求的应用程序逻辑。任务命令与控制器命令在同一容器中执行。

数据 bean 命令

数据 bean 命令由 JSP 模板在实例化数据 bean 时调用。数据 bean 命令的主要功能是用数据填充数据 bean。

视图命令

视图命令用于形成视图，并将它作为对客户机请求的响应。有三种类型的视图命令:

重定向视图命令

此视图命令使用重定向协议（例如 URL 重定向）发送视图。控制器命令应以此视图类型返回视图命令，从而使用重定向协议返回一个视图。当使用重定向协议时，它将更改浏览器中的 URL 堆栈。当输入重新装入键时，将执行重定向的 URL 而不是原始的 URL。

定向视图命令

此视图命令将响应视图直接发送到客户机。

转发视图命令

此视图命令将视图请求转发到另一个 Web 组件，例如 JSP 模板。

调用视图命令可有三种方法:

- 控制器命令在成功完成请求时指定视图命令的名称。
- 客户机直接请求视图。
- 命令检测到错误，且必须执行错误任务来处理此错误。命令抛出带有视图命令名的异常。当异常传播到 Web 控制器时，它将执行错误视图命令并将响应返回客户机。

WebSphere Commerce 实体 bean

实体 bean 是由 WebSphere Commerce 提供的持久、事务性的商务对象。如果您对商业领域比较熟悉，则实体 bean 就可以用直观的方式代表 WebSphere Commerce 数据。即: 您可以从更准确地以商业领域中概念和对象作为模型的实体 bean 来访问数据，而不必了解整个数据库模式。您可以扩展现有的实体 bean。另外，对于您自己的特定于应用程序的业务需求，还可以部署全新的实体 bean。

实体 bean 是根据 Enterprise JavaBeans（EJB）组件模型实现的。

关于实体 bean 的更多信息，请参阅第 43 页的『WebSphere Commerce 实体 bean 的实现』。

数据 bean

数据 bean 是主要由 Web 设计者使用的 Java bean。通常，它们提供对 WebSphere Commerce 实体的访问。Web 设计者可将这些 bean 放置在 JSP 模板上，这样在页面显示时也可以将动态信息填充到页面上。Web 设计者只需了解这个 bean 能够提供什么数据及它需要什么数据作为输入。同显示与业务逻辑分离的风格一致，Web 设计者亦无需了解 bean 的工作方式。

数据 bean 管理器

当使用 WebSphere Studio Page Designer 将 WebSphere Commerce 数据 bean 插入 JSP 模板时，通过在运行时调用数据 bean 管理器生成填充数据 bean 的一行代码。

以下是来自 Page Designer 的代码样本:

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

JavaServer Pages 模板

JSP 模板是通常用来显示的专用小服务程序。完成 URL 请求时，Web 控制器将调用视图命令，该命令调用 JSP 模板。客户机也可以不通过相关联的命令而直接从浏览器调用 JSP 模板。在此情况下，JSP 模板的 URL 必须在路径中包含请求小服务程序，以便 JSP 模板需要的所有数据 bean 都可以在一个事务中激活。请求小服务程序可以将一个 URL 请求转发到 JSP 模板，并在一个事务中执行该 JSP 模板。

数据 bean 管理器将拒绝路径中不包含请求小服务程序的 JSP 模板的任何 URL。关于保护 JSP 模板和其它资源的更多信息，请参阅第 79 页的第 4 章，『访问控制』。

Instance_name.xml 配置文件

Instance_name.xml 配置文件可设置实例的配置信息。在初始化请求小服务程序时读取此文件。

请求摘要

本部分提供了在形成对请求的响应时各组件间的交互流程摘要。

下图之后是对每个步骤的描述。

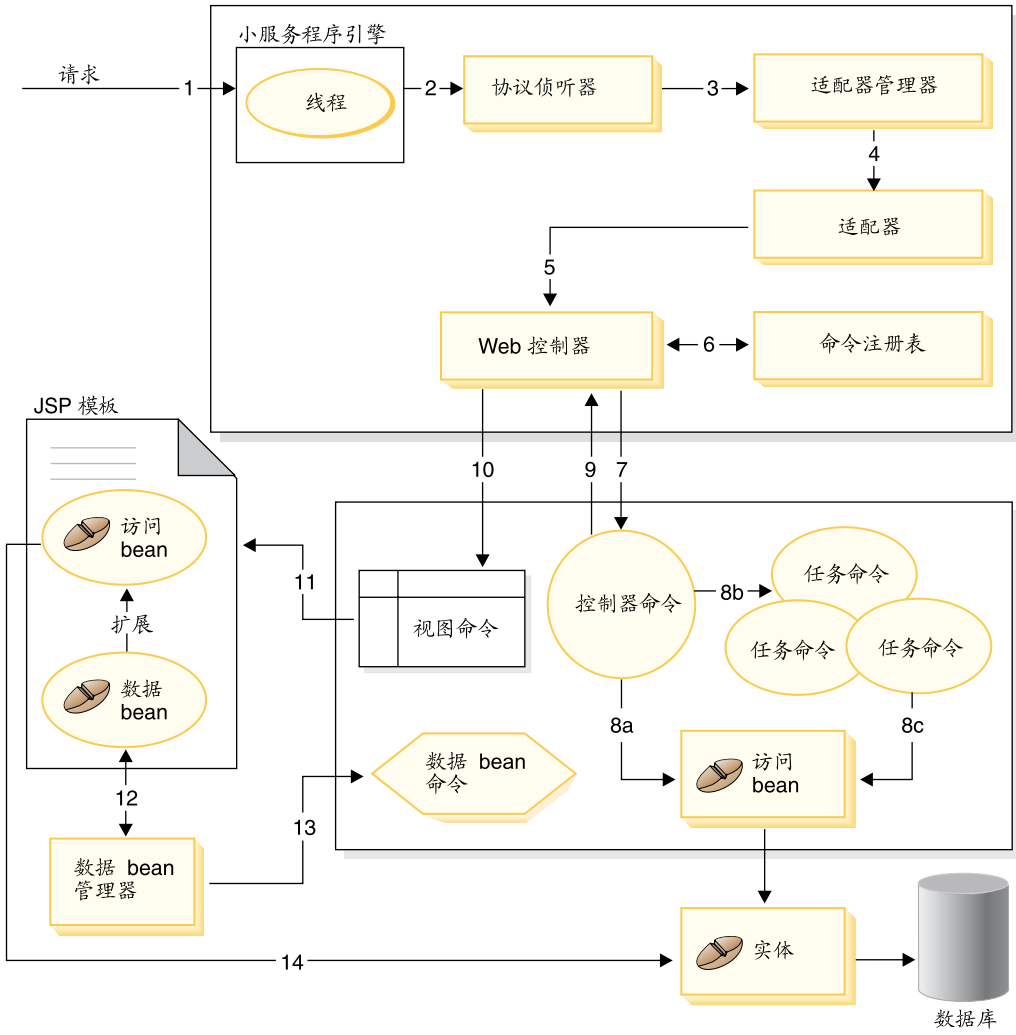


图 5.

以下信息与上图相对应。

1. 该请求通过 WebSphere Application Server 插件定向至小服务程序引擎。
2. 在它自己的线程中执行此请求。小服务程序引擎将请求分派到协议侦听器。该协议侦听器可以是 HTTP 请求小服务程序或 MQ 侦听器。
3. 协议侦听器将此请求传递到适配器管理器。
4. 适配器管理器确定哪个适配器可以处理请求，然后将该请求转发到相应的适配器。例如，如果请求来自因特网浏览器，则适配器管理器将该请求转发到 HTTP 浏览器适配器。

5. 适配器将请求传递到 Web 控制器。
6. Web 控制器通过查询命令注册表来确定要调用的命令。
7. 假设此请求需要使用控制器命令，那么 Web 控制器将调用适当的控制器命令。
8. 一旦开始执行控制器命令，存在几种可能的路径:
   a. 控制器命令可以使用访问 bean 及其相应实体 bean 来访问数据库。
   b. 控制器命令可以调用一个或多个任务命令。然后，任务命令可以使用访问 bean 及其相应的实体 bean 来访问数据库（8（c）中有叙述）。
9. 操作完成时，控制器命令将视图命令的名称返回到 Web 控制器。
10. Web 控制器在 VIEWREG 表中查找视图名称。它调用为请求者的设备类型注册的视图命令实现。
11. 视图命令将请求转发给 JSP 模板。
12. 在 JSP 模板中必须存在数据 bean 以从数据库中检索动态信息。数据 bean 管理器激活数据 bean。
13. 如果需要，数据 bean 管理器调用数据 bean 命令。
14. 从其扩展数据 bean 的访问 bean 使用它相应的实体 bean 来访问数据库。

先前各发行版中定制间的主要差别

从 WebSphere Commerce Suite 版本 5.1 开始，WebSphere Commerce Server 已全部用 Java 编写。因此，必须使用 Java 来定制功能。这与 WebSphere Commerce Suite 版本 4.1（以及 Net.Commerce™ 的早期版本）中使用的模型有很大的差别，在这些版本中使用 C++ 和 Net.Data® 宏来实现定制。

下表总结了主要的更改。

	使用 C++ 的版本 4.1（和早期版本）	**使用 Java 的版本 5.1（以及后继版本）**
应用程序模型	命令	控制器命令
	任务	任务命令
	可覆盖函数	任务命令
	视图任务	视图命令
	错误任务	视图命令
显示模型	Net.Data 宏	JSP 模板、数据 bean、数据 bean 命令和视图命令
持久性模型	Net.Data 和 ODBC	实体 bean
URL 分派器	WebSphere Commerce C++ URL 分派器	WebSphere 小服务程序引擎

	使用 C++ 的版本 4.1（和早期版本）	使用 Java 的版本 5.1（以及后继版本）
任务模型	系统进程	Java 线程
命令适配器	无	Web 控制器和适配器

本出版物中没有描述迁移过程。关于迁移的更多信息，请参阅《*WebSphere Commerce 迁移指南*》。

第 2 部分 编程模型

第 2 章 设计模式

在开发 WebSphere Commerce 框架中使用了各种不同的设计模式和机制。WebSphere Commerce 提供了每个 WebSphere Commerce 应用程序都应遵循的高级设计模式。本章中将讨论以下设计模式:

- 模型、视图和控制器设计模式
- 命令设计模式
- 显示设计模式

模型、视图和控制器设计模式

模型、视图和控制器（MVC）设计模式指定应用程序由数据模型、展示信息和控制信息构成。该模式需要它们中的每个都分入不同的对象中去。

模型（例如，数据信息）仅包含纯应用程序数据；它不包含描述如何将数据表示给用户的逻辑。

视图（例如，表示信息）将模型数据表示给用户。视图知道如何访问模型数据，但它不知道此数据表示的意义或用户可以对它进行何种操作。

最后，*控制器*（例如，控制信息）存在于视图和模型之间。它侦听视图（或其它外部资源）所触发的事件，并对这些事件执行相应的反应。大多数情况下，这种反应是对模型调用方法。由于视图和模型通过通知机制相连接，因此该操作的结果就是自动在视图中有所反映。

现在大多数应用程序都遵照此模式，其中许多有少许的变化。例如，某些应用程序将视图和控制器组合为一类，这是因为它们已经结合得非常紧密了。所有这些变体都强烈倾向于将数据及其表示分离。这不仅使应用程序的结构更加简单，同时也使代码可以得到重用。

鉴于有许多出版物都描述了此模式并提供了大量样本，本文档将不再详细描述这种模式。

下图显示了 MVC 设计模式在 WebSphere Commerce 中是如何应用的。

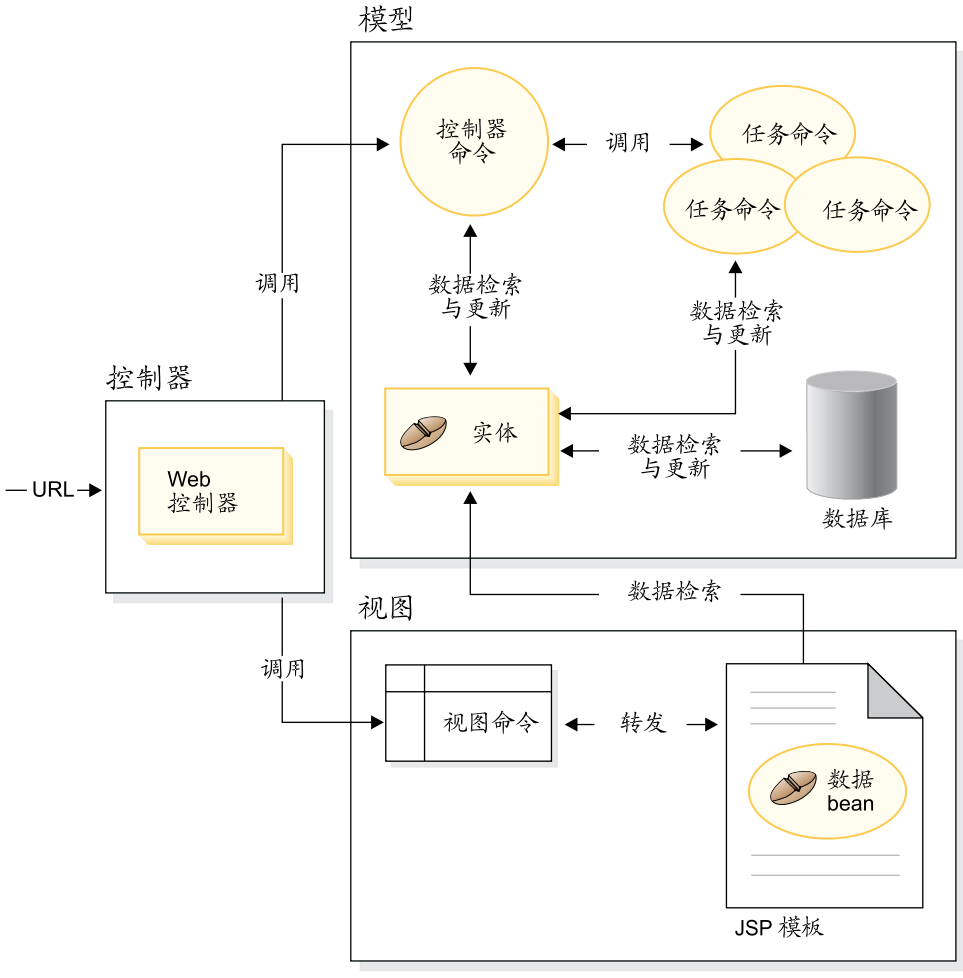


图 6.

命令设计模式

WebSphere Commerce Server 接受来自基于浏览器的瘦客户机应用程序的请求，以及来自其它远程应用程序的请求。例如，请求可来自远程采购系统，或来自另一个贸易服务器。

各种格式的请求由组成适配器框架的适配器翻译成一种公共格式。一旦请求都采用此公共格式，它们就可被 WebSphere Commerce 命令理解。

命令是执行业务逻辑的 bean。它们以高级别进程逻辑的形式或离散业务逻辑任务的形式代表程序上的逻辑。基于进程的命令作为控制器操作，它可以跨越多个实体和其它命令，而任务命令则执行特定任务，且可能仅访问单个对象。

命令框架

命令 bean 遵循特定设计模式。每个命令都包含一个接口类（例如，`CategoryDisplayCmd`）和一个实现类（例如，`CategoryDisplayCmdImpl`）。从调用函数的角度来看，调用逻辑涉及设置输入属性、调用 `execute()` 方法及检索输出属性。

从命令实现函数的角度来看，命令遵循 WebSphere 命令框架，该框架可以实现允许调用函数与实现之间间接交互级别的标准命令设计模式。此间接交互级别中支持的密钥机制包括:

1. 能够调用确定是否允许用户调用命令的访问控制策略管理器。
2. 能够对不同的商店执行不同的命令实现（根据商店标识）。
3. 能够根据请求程序的设备类型执行不同的视图实现。

下图显示了命令的 4 种主要类型接口的概念性概述:

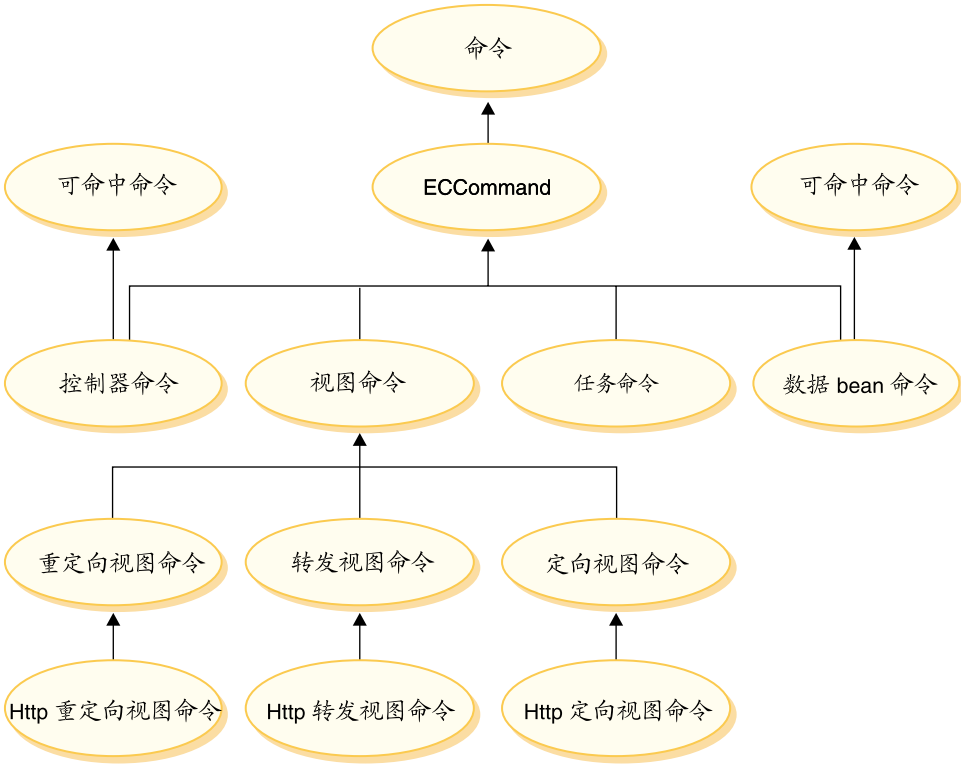


图 7.

控制器命令

控制器命令将与特定业务过程相关的逻辑封装起来。控制器命令的示例包括：用于订单处理的 *OrderProcessCmd* 命令和用于创建新注册用户的 *UserRegistrationAddCmd* 命令。通常，控制器命令包含控制语句（例如，if、then 和 else）和调用任务命令，以便在业务过程中执行单个任务。完成时，控制器命令将返回一个视图名称。然后，Web 控制器将确定视图任务相应的实现类，并执行该视图任务。

虽然控制器命令是可命中的命令，但它仅支持本地目标。

任务命令

任务命令实现应用程序逻辑的特定部分。通常，控制器命令和一组任务命令共同实现 URL 请求的应用程序逻辑。任务命令与控制器命令在同一容器中执行。

数据 bean 命令

在实例化数据 bean 时，数据 bean 命令由 JSP 页面调用。数据 bean 命令的主要功能是填充数据 bean 的字段。

虽然数据 bean 命令是可命中的命令，但它仅支持本地目标。

视图命令

视图命令用于形成视图，并将它作为对客户机请求的响应。调用视图命令可有三种方式:

- 控制器命令在成功完成请求时指定视图命令的名称。
- 客户机可以直接请求视图。
- 控制器或任务命令检测到错误，并决定必须执行错误任务来处理此错误，而且抛出带有视图命令名的异常。当异常传播到 Web 控制器时，它将执行视图命令并将响应返回客户机。

有三种类型的视图命令:

重定向视图命令

此视图命令使用重定向协议（例如 URL 重定向）发送视图。当需要重定向协议时，控制器命令应返回此视图类型的视图命令。当使用重定向协议时，它将更改浏览器中的 URL 堆栈。当输入重新装入键时，将执行重定向的 URL 而不是原始的 URL。

定向视图命令

此视图命令将响应视图直接发送到客户机。

转发视图命令

此视图命令将视图请求转发到另一个 Web 组件，例如 JSP 模板。

命令工厂

为创建新命令对象，命令调用函数可以使用*命令工厂*。命令工厂是用于将命令实例化的 bean。它以工厂设计模式为基础，该模式从调用类到工厂类（工厂类了解应实例化哪些实现类）推迟了对象的实例化。

工厂提供了将新对象实例化的智能方式。在此情况下，命令工厂提供了根据单独商店创建新命令对象时，确定正确实现类的途径。实例化时，命令接口名称和特定商店标识将传递到新命令对象中。

指定命令的实现类有两种方式。缺省实现类可以直接在命令接口的代码中使用 `defaultCommandClassName` 变量指定。例如，以下代码存在于 CategoryDisplayCmd 接口中:

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

指定实现类的第二种方式是使用 WebSphere Commerce 命令注册表。在各家商店的实现类相互间不同的情况下，应当总是使用命令注册表。关于命令注册表的更多信息可以在第 26 页上找到。

当在接口的代码中指定了缺省实现类，而在命令注册表中指定了不同的实现类时，命令注册表优先。

使用命令工厂的语法如下:

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

其中 *interfaceName* 是新命令 bean 的接口名称，getStoreId 方法用于确定要为其使用命令的商店。

注: 使用命令工厂创建业务策略命令的语法与上述代码片段不同。关于使用命令工厂创建业务策略命令的更多信息，请参阅第 160 页的『调用新业务策略』。

命令流程

本部分提供了命令与 WebSphere Commerce 数据库间逻辑流程的概述。下图和描述描写了此流程。

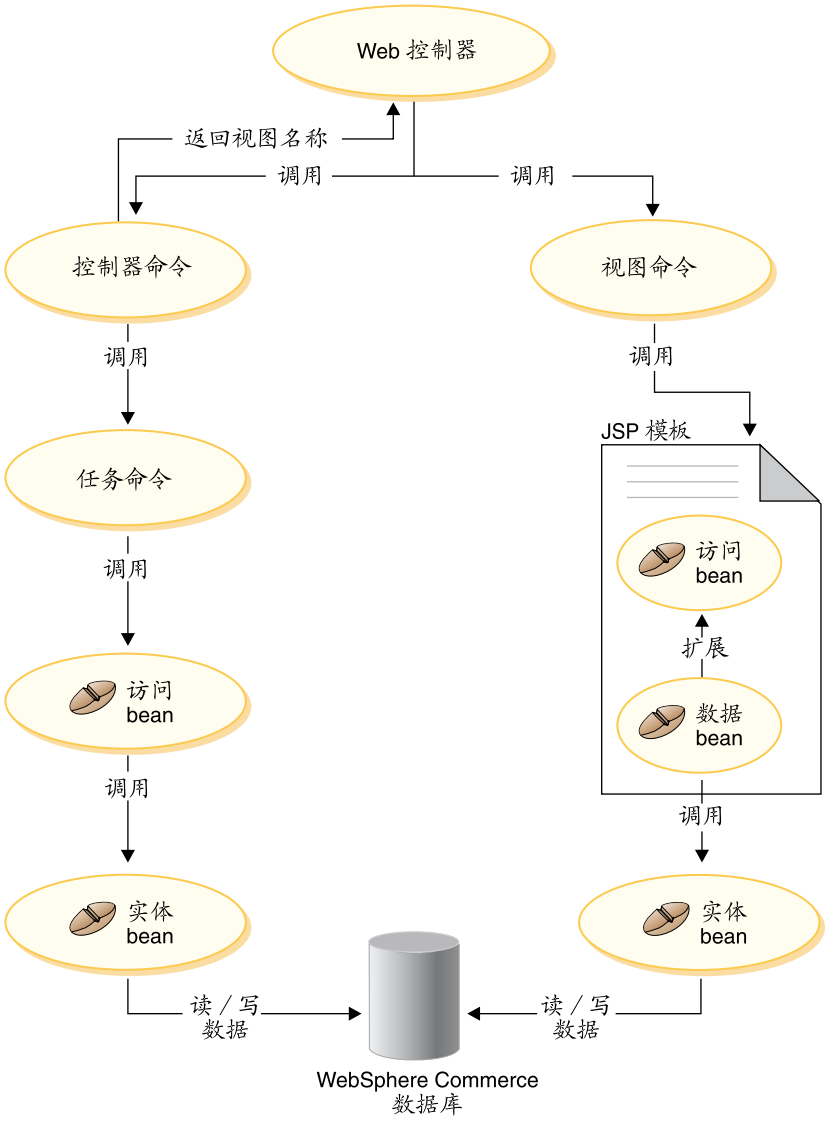


图 8.

当 Web 控制器接收到请求时，它将确定请求需要调用控制器命令还是视图命令。在这两种情况下，Web 控制器还将确定命令的实现类，然后调用它。

首先我们来检查图表的左侧。由于控制器命令封装了业务过程的逻辑，因此它们经常调用单独的任务命令来执行业务过程中的特定工作单元。当必须检索或更新

数据库中的信息时，会调用访问 bean。任务命令或控制器命令都可以调用访问 bean。然后请求从访问 bean 流向实体 bean，后者可从 WebSphere Commerce 数据库中读取，也可写入其中。

现在再来检查图表右侧。在控制器命令完成处理并返回要调用的视图命令时，或出现错误且必须显示错误视图时，Web 控制器将调用视图命令。

视图命令通常调用 JSP 模板显示对客户机的响应。在 JSP 模板内，数据 bean 用于向页面填充动态信息。数据 bean 由数据 bean 管理器激活。数据 bean（它扩展自访问 bean）调用其相应的实体 bean。当间接从 JSP 模板访问时，实体 bean 通常从数据库检索信息（而不是向数据库写入信息）。

命令注册框架

WebSphere Commerce 控制器和任务命令注册在命令注册表中。以下三个表构成了命令注册表:

- URLREG
- CMDREG
- VIEWREG

注: Business 本节不适用于业务策略命令的注册。关于注册新业务策略命令的更多信息，请参阅第 145 页的『注册新业务策略和业务策略命令』。

URLREG 表

URLREG 表将 URI（通用资源指示符）映射到控制器命令接口。URI 为资源识别提供了简单而可扩展的机制。URI 是用于标识抽象或物理资源的相对简短的字符串。在 WebSphere Commerce 中，URI 仅包含命令信息。在以下 URL 中，URI 部分以粗体显示:

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

虽然 URI 与接口名称之间是一对一的映射，但每家商店都可以指定命令需要 HTTPS 还是 AUTHENTICATION。对于每个入站 URL 请求，Web 控制器都会查找控制器命令的接口名称，然后使用该名称确定 CMDREG 表中注册的正确实现类。

下表描述了 URLREG 数据库表中包含的信息。

列名	描述	注释
URL	URI 名称	例如 MyNewCommand 或 ProductDisplay

列名	描述	注释
STOREENT_ID	商店实体标识	可以将它设置为 0，以便对所有商店使用命令，或设置为唯一商店标识来表示仅对特定商店使用命令。
INTERFACENAME	控制器命令接口名称	例如 com.ibm.commerce.catalog.commands.GetProductDisplay.TemplateCmd
HTTPS	此 URL 请求需要的安全 HTTP	需要 HTTPS 时使用 1，不需要时使用 0。
DESCRIPTION	URI 的描述	例如，此命令用于测试目的。
AUTHENTICATED	此 URL 请求需要用户登录	需要认证时使用 1，不需要时使用 0。
INTERNAL	指示命令对 WebSphere Commerce 是否是内部命令	命令是内部命令时使用 1，是外部命令时使用 0。

当 Web 控制器接收到 URL 请求时，它将检索受到请求的控制器命令的接口名称，并使用它从 CMDREG 表查找实现类名称。它还通过检查 URLREG 表中的 HTTPS 列来确定 URL 请求是否需要 HTTPS。

只有通过 URL 请求方式调用的命令才需要在 URLREG 表中注册。因此，只有控制器命令才必须在此注册，而任务命令或视图命令则不必。

以下 SQL 语句为特定商店（商店标识为 5）所使用的 MyNewControllerCommand 创建了条目:

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

insert 语句的一般语法如下:

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

字符串值应当括在单引号内。

CMDREG 表

CMDREG 是命令注册表。此表提供了将命令接口映射到其实现类的机制。一个接口存在多个实现可以允许在每家商店的基础上进行命令定制。

只有控制器命令和任务命令才注册在 CMDREG 表中。视图命令注册在 VIEWREG 表中。

下面描述了 CMDREG 数据库表中包含的信息。

列名	描述	注释
STOREENT_ID	商店实体标识	可以将它设置为 0，以便对所有商店使用命令，或设置为唯一商店标识来表示仅对特定商店使用命令。
INTERFACENAME	命令接口名称	它定义了接口；使用在 URLREG 表中的相同名称。
DESCRIPTION	此命令的描述	例如，此命令用于测试目的。
CLASSNAME	命令实现类名称	通常接口名称结尾附带 “Impl”。
PROPERTIES	缺省“名称值”对设置为命令的输入属性	格式与 URL 查询字符串相同。例如 “parm1=val1&parm2=val2”
LASTUPDATE	对此命令条目的最后更新	
TARGET	命令目标名称。这是实际执行命令的地方。	仅支持本地目标。

一般来说，创建新的控制器或任务命令时，应当在 CMDREG 表中创建相应的条目。例如，以下 SQL 语句为特定商店（商店标识为 5）使用的 MyNewCommand 创建了条目:

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

insert 语句的一般语法如下:

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

字符串值应当括在单引号内。

如果您所写的命令经常使用相同的实现类，则不必在 CMDREG 表中注册命令。在此情况下，可以在接口中使用 defaultCommandClassName 属性来指定实现类。例如，在接口的代码中，您可包含以下内容:

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

如果以这种方式指定实现类，则无法将缺省属性传递给实现类，且必须对所有商店使用相同的实现类。

已注册控制器命令示例

请考虑这样一种情况，您的站点有两家商店：StoreA 和 StoreB。每家商店对 MyUrl 控制器命令都有不同的安全性需求和不同的命令实现。本部分将描述如何使用命令注册表启用此定制。

下表显示了 URLREG 表中 StoreA 和 StoreB 的条目：

列名	**StoreA 的条目**	**StoreB 的条目**
URL	MyUrl	MyUrl
STOREENT_ID	11	22
INTERFACENAME	com.ibm.commerce.mycommands.myUrl	com.ibm.commerce.mycommands.myUrl
HTTPS	1	1
DESCRIPTION	URLREG 表中的示例条目。	URLREG 表中的示例条目。
AUTHENTICATED	1	0
INTERNAL	空	空

注：INTERFACENAME 值中的空格仅为显示目的。每个值实际上都是一个连续的字符串。

Web 控制器根据 URLREG 表中的条目确定 MyURL URI 的接口名称是 com.ibm.commerce.mycommands.MyUrl。它还确定 StoreA 需要使用 HTTPS 和认证两者来执行命令，而 StoreB 仅需要 HTTPS。HTTPS 和认证的值是由 Web 控制器使用的，而不是由接口使用。

下图显示了此流程：

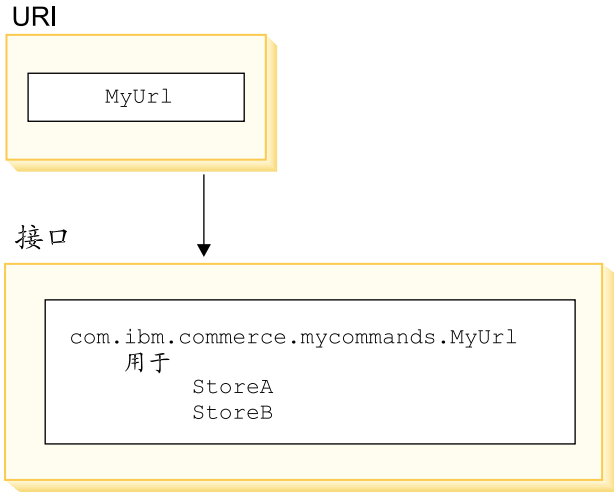


图 9.

下表显示了 CMDREG 表中的条目: 仅显示用于此示例目的所需的列:

列名	StoreA 的条目	StoreB 的条目
STOREENT_ID	11	22
INTERFACENAME	com.ibm.commerce.mycommands.myUrl	com.ibm.commerce.mycommands.myUrl
CLASSNAME	com.ibm.commerce.mycommands.myUrlStoreAImpl	com.ibm.commerce.mycommands.myUrlStoreBImpl

注: INTERFACENAME 和 CLASSNAME 值中的空格仅为显示目的。每个值实际上都是一个连续的字符串。

Web 控制器根据 CMDREG 表中的条目确定 StoreA 的 com.ibm.commerce.mycommands.MyUrl 接口的实现类是 com.ibm.commerce.mycommands.MyUrlStoreAImpl。它还确定 StoreB 相同接口的实现类是 com.ibm.commerce.mycommands.MyUrlStoreBImpl。下图显示了此流程:

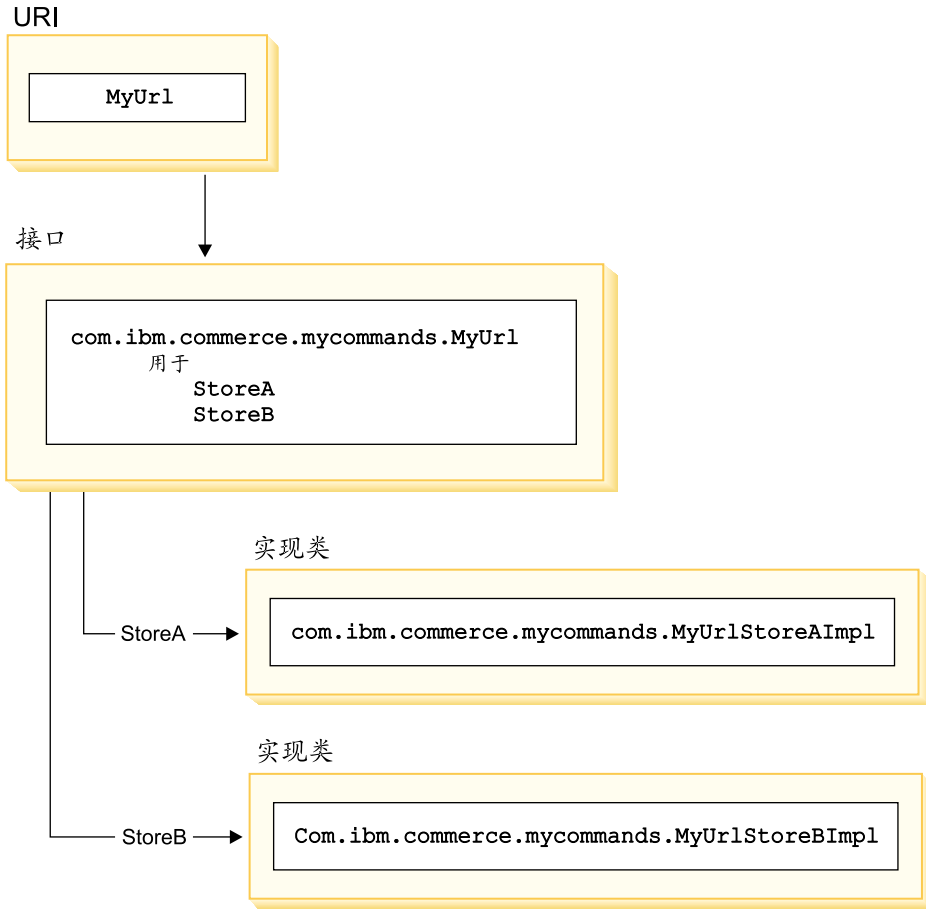


图 10.

VIEWREG 表

VIEWREG 表允许注册特定于设备的视图命令实现。使用此表可以注册视图的多个实现。然后，命令框架就能够返回不同客户的不同视图。

当从控制器命令中返回视图命令名或在异常中指定视图命令名时，Web 控制器从 VIEWREG 表中确定视图命令类。多个视图命令名可以映射到相同的实现类。

列名	描述	注释
VIEWNAME	视图名称	例如，AddressForm

列名	描述	注释
DEVICEFMT_ID	设备类型标识	可用的选项包括: • BROWSER（缺省值） • I_MODE • E-mail • MQXML • MQNC
STOREENT_ID	商店实体标识	可以将它设置为 0，以便对所有商店使用命令，或设置为唯一商店标识来表示仅对特定商店使用命令。
INTERFACENAME	视图命令接口名称	缺省选项是 ForwardView、DirectView 和 RedirectView。
CLASSNAME	视图命令实现类名称	可以使用缺省实现。
PROPERTIES	缺省“名称值”对设置为命令的输入属性	如果总是显示相同页面，则可以在此属性中设置 JSP 文件名（docname=*jsp_name*.jsp）。 如果对所有商店都使用相同 JSP 模板，应设置 storeDir=no 以防止特定于商店的目录被使用。 如果一般用户可以调用命令，则设置 isGeneric=true。
DESCRIPTION	此命令的描述	
HTTPS	此 URL 请求需要的安全 HTTP	需要 HTTPS 时使用 1，不需要时使用 0。
LASTUPDATE	对此条目的最后更新	
INTERNAL	指示命令对 WebSphere Commerce 是否是内部命令	命令是内部命令时使用 1，是外部命令时使用 0。

创建新视图命令时，可能需要在 VIEWREG 表中创建相应的条目。如果符合以下条件之一，则必须在 VIEWREG 表中注册视图命令:

- 视图命令在访问控制下执行
- 视图命令有多个实现
- PROPERTIES 列中设置了属性

已注册的视图命令可以使用视图名称通过命令注册表访问，也可以直接使用实际的显示文件名访问。要访问未在 VIEWREG 表中注册的视图，则只能在客户机使用实际显示文件名时访问。

假如有一名为 *MyView* 的视图示例，其 VIEWREG 条目如下:

列名	条目
VIEWNAME	MyView
DEVICEFMT_ID	BROWSER
STOREENT_ID	0
INTERFACENAME	com.ibm.commerce.commands.ForwardViewCommand
CLASSNAME	com.ibm.commerce.commands.HTTPForwardViewCommandImp
PROPERTIES	docname=MyView.jsp
DESCRIPTION	使用视图名称或直接从 URL 调用 JSP 模板的示例。
HTTPS	0
LASTUPDATE	2000-11-30
INTERNAL	0

由于 MyView 是已注册视图，因此客户机既可以使用命令名，也可以通过用实际显示文件名替换命令名来访问视图。使用该视图名称的样本 URL 是:

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

使用文件名的一个样本 URL 是:

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

如果客户机有可能直接（使用显示文件名）调用已注册的视图，则您必须使用与实际显示文件名相同的视图名称注册命令，如此示例中所示（MyView 和 MyView.jsp）。

未在表中注册的视图只能使用显示文件名调用。因此，如果存在使用文件 MyUnregisteredView.jsp 的未注册视图，则用于访问此视图的 URL 如下:

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

以下示例 SQL 语句为一个特定商店使用的 *MyNewViewCommand* 创建了一个条目:

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
CLASSNAME, PROPERTIES, DESCRIPTION,HTTPS, LASTUPDATE, INTERNAL) values
('MyNewViewCommand', 'BROWSER', 5,
'com.ibm.commerce.command.ForwardViewCommand',
```

```
'com.ibm.commmerce.command.HttpForwardViewCommandImpl',
'docname=MyNewViewCommand.jsp', 'A test view command.', 0, '0000-12-01',
0)
```

下表提供了另一个样本 VIEWREG 表及其键信息:

COMMAND - NAME	DEVICE - FMT_ID	INTERFACE - NAME	CLASSNAME	PROPERTIES
ProductDisplayView	BROWSER	转发视图命令	HttpForwardViewCommandImpl	
InterestItemAddView	BROWSER	重定向视图命令	HttpRedirectViewCommandImpl	docname =item.jsp
InterestItemDeleteView	BROWSER	重定向视图命令	HttpRedirectViewCommandImpl	docname =item.jsp
GenericApplication Error	BROWSER	重定向视图命令	HttpRedirectViewCommandImpl	docname =usererr.jsp
GenericSystemError	BROWSER	重定向视图命令	HttpRedirectViewCommandImpl	docname =syserr.jsp
Logon	BROWSER	转发视图命令	HttpForwardViewCommandImpl	docname= logon.jsp & storeDir=no

注: COMMANDNAME、INTERFACENAME、CLASSNAME 和 PROPERTIES 值中的空格仅为了显示。每个值实际上都是一个连续的字符串。列名中的连字符也是为了显示。

上表说明了以下情况:

- 控制器命令（在此例中为 ProductDisplay）将 *ProductDisplayView* 视图名称返回到 Web 控制器。Web 控制器使用 ProductDisplayView 视图命令名及其设备标识确定视图命令接口和类名。对于不同的商店和设备标识，视图命令也可以有不同的实现类。但由于接口名称定义视图命令类型，因此应当保持相同。
- InterestItemAdd 和 InterestItemDelete 命令将 *InterestItemAddView* 和 *InterestItemDeleteView* 视图名称分别返回到 Web 控制器。这两个命令都需要重定向视图，因此两个视图的视图命令接口名称都是 RedirectViewCommand。为这两个视图注册了一个公共 JSP 模板。Web 控制器读取属性（docname=item.jsp）并将它们传递到视图命令（HttpRedirectViewCommandImpl）。
- 如果控制器或任务命令因错误的用户参数而抛出 ECApplication 异常，则可能发生以下情况:
  - 如果某视图是在应当在发生应用程序异常时调用的控制器命令中指定的，则会从 VIEWREG 表中检索该视图的条目，并对它做相应的处理。

- 如果未指定视图，则调用 GenericApplicationError 命令，并显示数据库中已注册的 JSP 模板。以上表作为示例，这会导致显示 `usererr.jsp` 模板。

- 如果控制器或任务命令因某个系统异常而抛出 ECSystem 异常，则可能发生以下情况:
  - 如果某视图是在应当在发生系统异常时调用的控制器命令中指定的，则会从 VIEWREG 表中检索该视图的条目，并对它做相应的处理。
  - 如果未指定视图，则调用 GenericSystemError 命令，并显示数据库中已注册的 JSP 模板。以上表为示例，这会导致显示 `syserr.jsp` 模板。
- 浏览器客户机可以通过输入登录 URL 调用登录页面。由于 storeDir 属性设置为 “no”，因此特定于商店的信息不包含在 JSP 模板的路径中。所以对所有商店的顾客都显示相同的登录页面。

显示设计模式

显示页面将响应返回客户机。通常情况下，显示页面作为 JSP 模板实现（推荐方法），但是它们可以直接写成小服务程序。

为支持多种设备类型，对视图命令的 URL 访问应使用视图名称，而不是实际 JSP 文件的名称。

隐藏在此间接交互级别后面的主要基本原理是 JSP 模板代表一个视图。特别是因为单个请求常常有多个可能的视图，所以选择适当视图的能力（例如根据语言环境、设备类型或请求语境中的其它数据）非常尽如人意。假如有这样一个示例，两个购物者同时请求一家商店的主页，一人使用常用的 Web 浏览器，另一人使用蜂窝式电话。显然不能对每个购物者显示相同的主页。Web 控制器负责接受请求，然后根据命令注册框架中的信息确定每个购物者接收的视图。

JSP 模板和数据 bean

数据 bean 是在 JSP 模板内用于提供动态内容的 Java bean。通常，数据 bean 提供了 WebSphere Commerce 实体 bean 的简单表示。数据 bean 将可以从实体 bean 中检索或在实体 bean 中设置的属性封装起来。这样，数据 been 就简化了将动态数据结合到 JSP 模板中的任务。

数据 bean 具有 *BeanInfo* 类，这种类定义了能够在显示页面上使用的属性。BeanInfo 类还通过提供以 WebSphere Commerce 所有支持语言表示的属性名，使数据 bean 可以在多文化站点中使用。

数据 bean 由以下调用激活:

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

WebSphere Studio Page Designer 在 WebSphere Commerce 数据 bean 插入 JSP 模板时，自动生成上面的代码行。

商店开发者在开发 JSP 模板时，应当考虑商店的属性和支持多文化的问题。关于支持多文化的更多信息，请参阅 WebSphere Commerce 联机帮助。

JSP 模板和数据 bean 安全性注意事项

使用 JSP 模板和数据 bean 时编制特定代码，这样可以最大限度地减小恶意用户以未经授权的方式访问数据库的机会。SQL 语句的 insert、select、update 和 delete 部分应在开发时创建。使用参数插入来收集运行时的输入信息。

使用参数插入收集运行时输入信息的一个示例如下:

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

与之相反，您应当避免使用输入字符串作为编写 SQL 语句的方法。使用输入字符串的示例如下:

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

数据 bean 类型

数据 bean 是主要用于在 JSP 模板内提供动态内容的 Java bean。有两种类型的数据 bean: 智能数据 bean 和命令数据 bean。

智能数据 bean 使用*惰性读取*方法检索自己的数据。在并不需要来自访问 bean 的所有数据的情况下，这种类型的数据 bean 可以提供更好的性能，这是因为它仅仅按需要检索数据。需要访问数据库的智能数据 bean 应当从相应实体 bean 的访问 bean 扩展，并实现 com.ibm.commerce.SmartDataBean 接口。例如，ProductData 数据 bean 扩展了 ProductAccessBean 访问 bean，其中后者对应产品实体 bean。

有些智能数据 bean 不需要数据库访问。例如，PropertyResource 智能数据 bean 从资源绑定检索数据，而不是从数据库检索。当不需要数据库访问时，智能数据 bean 应当扩展 SmartDataBeanImpl 类。

命令数据 bean 依赖命令来检索它的数据，它是更轻量级的数据 bean。不管 JSP 模板是否需要，命令一次为数据 bean 检索所有属性。因此，对于仅使用数据 bean 某些选择属性的 JSP 模板，命令数据 bean 在性能时间上可能比较浪费。而对于需要大部分或全部属性的 JSP 模板来说，使用命令数据 bean 则是非常方便的。

命令数据 bean 也可以从它们相应的访问 bean 扩展，并实现 com.ibm.commerce.CommandDataBean 接口。

数据 bean 接口

数据 bean 实现以下一个或所有 Java 接口:

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

每个 Java 接口都描述了填充数据 bean 的数据源。通过实现多个接口，数据 bean 可以访问各种源的数据。以下提供了关于每个接口的更多信息。

SmartDataBean 接口: 实现 SmartDataBean 接口的数据 bean 可以检索它自己的数据，而不需要使用相关联的数据 bean 命令。智能数据 bean 通常从相应实体 bean 的访问 bean 扩展。当智能数据 bean 激活时，数据 bean 管理器调用数据 bean 的填充方法。数据 bean 使用填充方法可以检索所有属性（除来自相关联对象的属性外）。例如，如果数据 bean 从某个实体 bean 的访问 bean 类扩展，则数据 bean 将调用 refreshCopyHelper 方法。相应实体 bean 的所有属性都自动填充到智能数据 bean 中。然而，如果实体 bean 具有关联对象，则不会检索那些对象的属性。使用智能数据 bean 的主要优点有:

- 实现简单，无需写数据 bean 命令。
- 当新字段添加到实体 bean 中时，不需要在数据 bean 中进行更改。修改实体 bean 以后，必须重新生成访问 bean（使用 VisualAge for Java 中的工具）。一旦重新生成访问 bean，所有新属性将自动对于智能数据 bean 可用。
- 实体 bean 常常包含代表关联对象的属性。出于性能原因，智能数据 bean 并不自动检索这些属性。相反，如果在请求检索之前延迟检索这些属性则会更加可取，如下图所示:

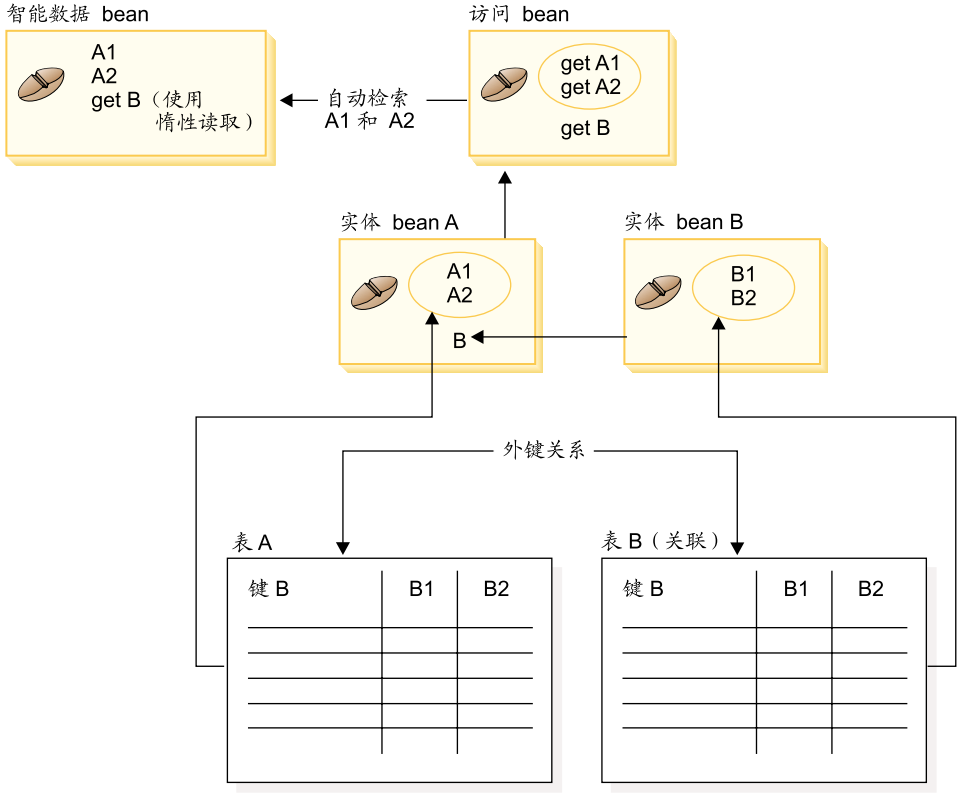


图 11.

关于实现惰性读取检索的更多信息，请参阅第 39 页的『惰性读取数据检索』。

CommandDataBean 接口: 实现 CommandDataBean 接口的数据 bean 从数据 bean 命令检索数据。此类型的数据 bean 是轻量级的对象；它依赖数据 bean 命令填充数据。数据 bean 必须实现 getCommandInterfaceName() 方法（如 com.ibm.commerce.CommandDataBean 接口所定义），该方法返回数据 bean 命令的接口名称。

InputDataBean 接口: 实现 InputDataBean 接口的数据 bean 从视图命令设置的 URL 参数或属性检索数据。

此接口中定义的属性可以用作读取附加数据的主键字段。调用 JSP 模板时，生成的 JSP 小服务程序代码将填充所有与 URL 参数匹配的属性，然后通过将数据 bean 传递至数据 bean 管理器而激活数据 bean。继而，数据 bean 管理器调用数据 bean 的 setRequestProperties() 方法（如 com.ibm.commerce.InputDataBean 接口所定义）传递视图命令所设置的所有属性。应当注意，WebSphere Studio 为使用 Page Designer 插入页面的每个数据 bean 生成以下代码:

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

BeanInfo 类

没有 BeanInfo 类的数据 bean 是不完整的，BeanInfo 类用于实现 java.lang.Object.BeanInfo 接口。BeanInfo 类用于提供关于数据 bean 方法和属性的显式信息。它可用于将数据 bean 实现类公共运行时的方法在 Web 设计器中隐藏起来，或为每个数据 bean 属性设置相应的显示字符串。

关于实现 BeanInfo 类的更多信息，请参阅 Sun Microsystems 的 JavaBeans 规范。

数据 bean 激活

数据 bean 可以用 activate 或 silentActivate 方法来激活，这些方法可以在 com.ibm.commerce.beans.DataBeanManager 类中找到。activate 方法是完全激活方法，在此方法中，激活事件仅在所有属性都可用时才能成功。即使有一个属性不可用，也会对整个激活过程抛出异常。

silentActivate 方法在单个属性不可用时不抛出异常。

从 JSP 模板调用控制器命令

虽然从 JSP 模板调用控制器命令与从显示上看是与独立逻辑不一致的，但您可能遇到需要此操作的情况。如果是这样，`ControllerCommandInvokerDataBean` 可用于此目的。

使用此数据 bean，您可以指定被调用命令的接口名称，或直接设置被调用的命令名。您也可以设置该命令的请求属性。

当此数据 bean 由数据 bean 管理器激活时，执行控制器命令并且响应属性可用于 JSP 模板。

一旦执行了控制器命令，您可以执行视图。

惰性读取数据检索

当数据 bean 激活时，它可以用数据 bean 命令或数据 bean 的 populate() 方法来填充。所检索的属性来自数据 bean 的相应实体 bean。实体 bean 也可以有相关联的对象，这些对象本身就有很多属性。

如果激活时对所有关联对象的属性都自动进行检索，则可能会遇到性能问题。随着关联对象数量的增加，性能可能会降低。

假如有一个包含大量交叉销售、优选销售或辅助产品（关联对象）的产品数据 bean。一旦激活此产品数据 bean，就可能会填充所有关联对象。但是用这种方式填充需要多个数据库查询。如果页面不需要所有属性，多数据库查询的效率可能会很低。

一般来说，页面并不需要所有属性，因此更好的设计模式是执行如下所示的惰性读取:

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

设置 JSP 属性 — 概述

WebSphere Commerce 程序模型提倡使用 MVC 设计模式。这样，URL 请求结果的表示便与控制器和任务命令相分离。这些命令是独立于设备的。它们实现业务逻辑并产生要返回到客户机的数据，而不需要关于客户机的信息。与之相反，视图命令是特定于设备的。

虽然控制器和任务命令并不直接编辑视图，但它们还是将信息传递至视图。了解信息如何传递至视图非常重要。下图演示了属性是如何在 Web 控制器、命令注册表、控制器命令和视图命令之间传递的:

CMREG

INTERFACENAME		PROPERTIES
com.ibm.xxx. NewCommand		parm1=1&parm2=2

CCPd: parm1=1&parm2=2

VIEWREG

INTERFACENAME		PROPERTIES
com.ibm.xxx.NewView		docName=NewView.jsp

VPd: docName=NewView.jsp

URL: http://hostname.com/NewCommand?storeID=1&....

CCPu: storeID=1&...

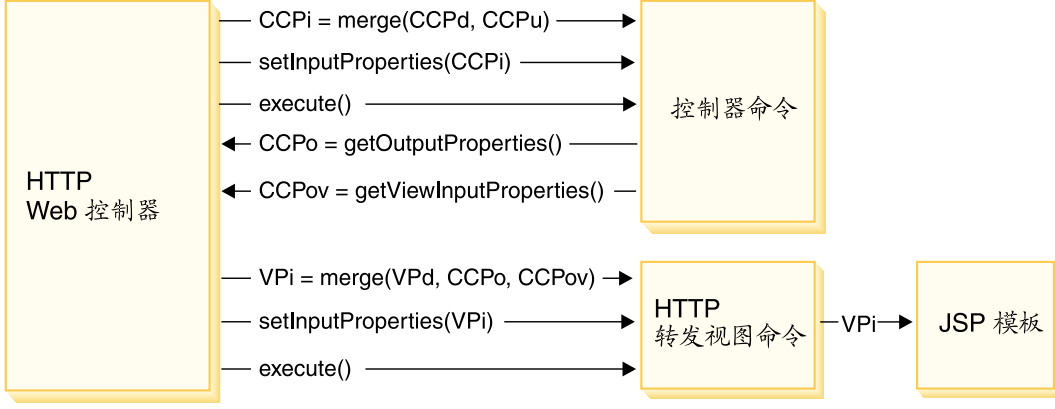


图 12.

上图显示了以下交互作用:

- Web 控制器将 URL 参数的输入属性（CCPu）与控制器命令所对应的 CMDREG 表中的条目（CCPd）合并。产生 CCPi。
- Web 控制器再将合并的属性（CCPi）传递至控制器命令，并执行控制器命令。

- 控制器命令将输出属性设置为 CCPo。它们是命令本身产生的输出属性。其中一个输出属性 viewCommandName 设置为期望的视图命令名。Web 控制器使用获取方法检索这些属性。
- 控制器命令将另一组输出属性设置为 CCPov。缺省情况下，它们设置为初始合并的输入属性（CCPi）。这些属性可能可以定制。例如，可能不必将所有输入参数都传递至视图命令。
- Web 控制器将三组属性 CCPo、CCPov 和 VPd（在 VIEWREG 表中注册的属性）合并到视图命令（VPi）的输入属性中。
- Web 控制器设置合并的属性 VPi 并执行视图命令。
- 视图命令从输入属性将属性设置到 JSP 模板中。

编新命令时，您并没有明确地执行属性合并。抽象命令类中包含了 mergeProperties 方法。关于此方法的更多信息，请参阅 WebSphere Commerce 联机帮助的 “参考” 部分。

必需的属性设置

控制器命令必须为每种类型的视图命令设置以下属性。如果命令没有设置属性，则必须在 VIEWREG 表中定义。

- 如果使用 ForwardView 命令，请设置 `docname = view_file_name` 其中 *view_file_name* 是显示模板的名称。例如，`docname = productDisplay.jsp`。
- 如果使用 DirectView 命令，请执行以下一项操作:
  - 设置 `textDocument = xxx`，其中 *xxx* 是包含文本形式的文档的 java.io.InputStream 对象
  - 设置 `rawDocument = yyy`，其中 *yyy* 是包含二进制形式的文档的 java.io.InputStream 对象

  使用 DirectView 命令时，可以选择设置 `contentType = ttt`，其中 *ttt* 是文档内容类型
- 如果使用 RedirectView 命令，应设置 `url = uuu`，其中 *uuu* 是重定向 URL。

第 3 章 持久对象模型

WebSphere Commerce 处理着大量持久数据。在当前数据库模式中定义了超过 520 个表。即便是使用如此广泛的模式，您还是可能需要扩展或定制数据库模式，以满足自己的特定业务需求。

WebSphere Commerce 使用基于 Enterprise JavaBeans（EJB）组件体系结构的实体 bean 作为持久对象层。这些实体 bean 以商业领域中的概念和对象作为模型的方式代表 WebSphere Commerce 数据。此持久层提供了可扩展的框架。

VisualAge for Java 企业版提供了支持此框架开发的复杂 EJB 工具和单元测试环境。

WebSphere Commerce 实体 bean 的实现

WebSphere Commerce 实体 bean — 概述

如前所述，WebSphere Commerce 体系结构中的持久层是根据 EJB 组件体系结构实现的。EJB 体系结构定义了两种类型的企业 bean: 实体 bean 和会话 bean。实体 bean 进一步分割为容器管理的持久性（CMP）bean 和 bean 管理的持久性（BMP）bean。

大部分 WebSphere Commerce 实体 bean 都是 CMP 实体 bean。一小部分无状态的会话 bean 用于处理密集的数据库操作，例如对特定列中的所有行执行求和。使用 CMP 实体 bean 的优点是开发者可以利用 VisualAge for Java 企业版中提供的 EJB 工具。这些工具允许开发者定义 Java 对象及其数据库表映射。它们会自动生成实体 bean 必需的持久函数。持久函数是将 Java 字段保存到数据库并用数据库的数据填充 Java 字段的 Java 对象。

VisualAge for Java 对当前 EJB 规范提供了两种扩展: EJB 继承和关联。EJB 继承允许企业 bean 从驻留在同一组中的另一个企业 bean 继承属性、方法和方法级别的控制描述符属性。关联是两个 CMP 实体 bean 之间存在的关系。

某些 WebSphere Commerce 实体 bean 利用了 EJB 继承特征。WebSphere Commerce 实体 bean 不使用 VisualAge for Java 提供的关联特征。开发自己的实体 bean 时，建议您不使用 VisualAge for Java 的关联特征。这样建议是为了将对象模型中的复杂性最小化。企业 bean 之间的对象关联可以通过在企业 bean 中添加显式获取函数方法建立，而不使用 VisualAge for Java 提供的关联特征。

WebSphere Commerce 提供了两组企业 bean: 专用和公共。专用企业 bean 由 WebSphere Commerce 运行时环境和工具使用。*不能*使用或修改这些 bean。

另一方面，公共企业 bean 由商业应用程序使用，既可以使用也可以扩展。这些公共企业 bean 组织到以下 EJB 组中:

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

上面列出的一些 EJB 组包含会话 bean。为了简化今后的迁移，请不要修改会话 bean 类。如果需要，可以在新的 EJB 组中创建新的会话 bean。关于创建新会话 bean 的更多信息，请参阅第 66 页的『写新会话 bean』。

WebSphere Commerce 企业 bean 的部署描述符

部署描述符是串行化的特殊类，它包含企业 bean 的运行时设置。WebSphere Commerce 企业 bean 的部署描述符以特定方式设置，不应修改。

创建新企业 bean（实体或会话 bean）时，请用鼠标右键单击 bean 并选择**属性**，在 VisualAge for Java 的 EJB Development Environment 中设置部署描述符。新企业 bean 的部署描述符应当遵循与 WebSphere Commerce 企业 bean 的描述符

相同的约定。特别要确保如下设置属性:

属性	值
事务属性	TX_REQUIRED
隔离级别	TRANSACTION_READ_COMMITTED
运行方式	SYSTEM_IDENTITY 或 CLIENT_IDENTITY
可重入	确保未选择此项。

企业 bean 经常包含只从数据库读取信息、但从不执行数据库更新的方法。这些方法称为*只读*方法。所有只读方法都应当如下明确标记（用鼠标右键单击方法并选择 **EJB 方法属性 > 只读方法**）。如果只读方法没有用此方式标记，EJB 容器会不必要地在事务结束时尝试更新数据库，导致在只读事务中发生事务回滚错误。这将引起性能问题。

隔离级别

在 VisualAge for Java 内部用于 WebSphere Commerce 企业 bean 的事务隔离级别是 TRANSACTION_READ_COMMITTED。注意: 在对 JDBC 驱动程序 DB2 版与 JDBC 驱动程序 Oracle 版的隔离级别的实现上存在差别。这样，在部署到 WebSphere Application Server 环境时使用的事务隔离级别就因正在使用的数据库不同而异。

在 WebSphere Test Environment 之外操作时用于 DB2 数据库的隔离级别是 TRANSACTION_REPEATABLE_READ。在 WebSphere Test Environment 之外操作时用于 Oracle 数据库的隔离级别是 TRANSACTION_READ_COMMITTED。

如果您正在部署到 DB2 数据库，则您无需手工更改交易隔离级别，在部署的步骤中，当您发出 modifyIsolationLevel 命令时将更改隔离级别。

下表显示了 DB2 和 Oracle 事务隔离级别到它们相应 JDBC 事务隔离级别的映射。

JDBC	DB2	Oracle
读不约束	不约束读	读不约束
读约束	光标稳定性	（不适用）
可重复读	读稳定性	读约束
可串行	可重复读	可串行
无	（不适用）	（不适用）

对于 DB2 中的“光标稳定性”事务隔离级别，只有已在给定事务中更新的行将被专门锁定。如果没有更新所给定行的任何列，则即使它是从 SQL 语句返回的结果集的一部分，一旦光标移开到另一行时，该特定行的行锁定也将释放。在某些情

况下，例如更新库存，这可能不是期望的行为。因此，通过在 DB2 中将事务隔离级别更改为“读稳定性”，以并行性的轻微降低为代价，数据完整性将大大提高。

在 Oracle 的情况下（仅“读约束”事务隔离级别或 JDBC“可重复读”同等级别可用），实际实现执行的方法所完成的行为与 DB2 中的“可重复读”事务隔离级别非常相似。

扩展 WebSphere Commerce 对象模型

WebSphere Commerce 对象模型可以用以下方法扩展:

- 扩展 WebSphere Commerce 公共企业 bean
- 写新的实体 bean
- 写新的无状态会话 bean

关于如何执行这些扩展的详细信息包含在以下章节中。

对象模型扩展方法论

您可能会根据应用程序的需求扩展现有 WebSphere Commerce 对象模型。此需求的一个示例就是将附加属性添加到应用程序。这可以使用以下方法之一完成:

不修改现有 WebSphere Commerce 公共实体 bean

创建新的数据库表，然后为该表创建新实体 bean。按照需要向实体 bean 添加字段和方法以处理新属性。为新实体 bean 生成部署代码和访问 bean。当应用程序需要新属性时，它将访问 bean 对象实例化，并使用其方法进行检索、设置或处理该属性。

修改现有 WebSphere Commerce 公共实体 bean

创建新的数据库表，并在新表与您正在修改的现有企业 bean 所对应的现有表之间创建表联合。在现有 WebSphere Commerce 公共实体 bean 中创建新字段，并使用二级表映射将这些字段映射到它们在新表中相应的列。添加任何需要的方法。为现有实体 bean 重新生成部署代码和访问 bean。当应用程序实例化访问 bean 对象时，新属性即可用。

这两种方法各有利弊。通常其利弊与性能和代码维护的劳动量相关。

扩展示例: 假如有这样一个示例，应用程序需要您获取顾客所拥有的房屋类型。您创建称为 USERRES 的表，它包含顾客标识和居住类型，其中居住类型（resType）可以是独立产权房、共有产权房或公寓。此类信息是情况调查信息，且这类信息与现有 Commerce Suite USERDEMO 表相关。检查 WebSphere Commerce 代码资源库，可以发现 WCSUser EJB 组中包含一个“Demographics”企业 bean。这个 bean 具有 USERDEMO 表中存储的情况调查信息的获取函数和设置函数。

要执行定制，有两种选项。既可以创建与 USERRES 表交互的新实体 bean，也可以向 Demographics bean 添加新字段（以及适当的获取函数和设置函数方法）。

使用第一种方法（创建全新代码），将创建新的 Userres 实体 bean，并将其字段映射到 USERRES 表的各列。当应用程序需要顾客居住类型时，它必须实例化 Userres 访问 bean 对象并检索数据。如果应用程序同时还需要其它情况调查信息，则它还必须实例化 Demographics 访问 bean 对象，并检索任何其它必需的属性。必须修改应用程序逻辑中任何试图检索一整套顾客情况调查信息的部分，以实例化原始访问 bean 和新访问 bean。下图显示了这种扩展对象模型的方法:

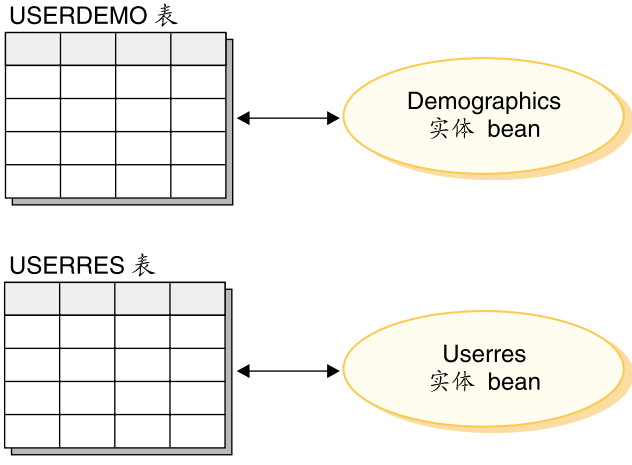


图 13.

从显示模板的角度考虑，数据 bean 必须能够访问新属性，以便信息对 JSP 模板可用。为了向创建 JSP 模板的 Web 开发者展示统一的视图，您应当创建新的数据 bean，用来为原始的现有实体 bean 扩展访问 bean。该数据 bean 还应当使用授权从新访问 bean 填充属性。下图显示了此数据 bean 实现方案:

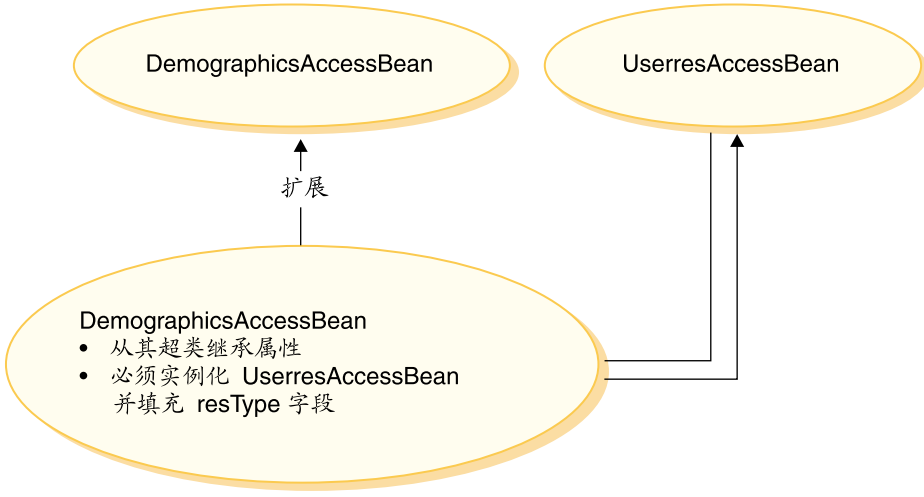


图 14.

使用第二种方法（修改现有代码），将向 Demographics 实体 bean 添加新字段，并在新字段与 USERRES 表的相应列之间创建二级表映射。当应用程序需要顾客居住类型时，它将实例化 Demographics 访问 bean 对象并检索居住类型。如果应用程序需要关于顾客的任何其它情况调查信息，则可以在对 bean 的同一次调用中获取。下图显示了修改企业 bean 的这种方法:

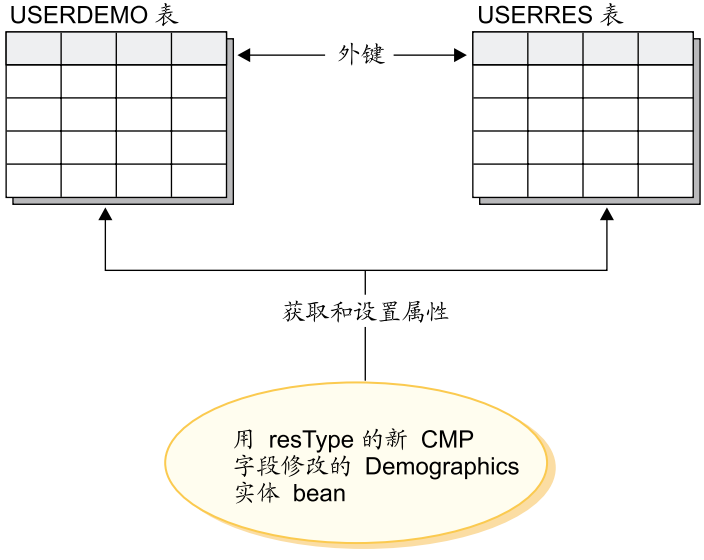


图 15.

从显示模板的角度来看，一旦 DemographicsAccessBean 重新生成，新属性（resType）在数据 bean 中便自动可用。

注意：在扩展对象模型时，*不能*向现有 WebSphere Commerce 数据库表添加新列。必须为新属性创建新表。如果确实要尝试向现有表中添加新列，则在迁移到将来的 WebSphere Commerce 发行版时，新属性将丢失。

性能和代码维护建议： 第二个方法具有更好的运行时性能。这是因为获取和设置新属性只需要实例化一个实体 bean，且它使用一次读取来检索所有必需的属性。

由于第二种方法修改了现有 WebSphere Commerce 代码，因此当发布新的 WebSphere Commerce 代码资源库时可能会引起迁移问题。您必须将已定制的代码与新代码合并，但在导入新代码资源库时，对企业 bean 添加的字段与新表之间的映射信息将不会保留。因此，在迁移到新发行版的 WebSphere Commerce 代码资源库时，必须执行以下步骤：

1. 创建已定制 EJB 代码的版本。
2. 导入新版本的 WebSphere Commerce 代码。
3. 使用 VisualAge for Java 中的工具，将已定制版本的代码与新发行版的 WebSphere Commerce 代码相比较。将已定制的代码合并回工作空间中。
4. 手工将添加到 WebSphere Commerce 公共企业 bean 的任何属性重新映射到数据库中的适当列。
5. 为您在步骤 4 中修改的企业 bean 重新生成部署代码和访问 bean。

为使迁移更加简单，在开发时用文档完整地记录对象模型扩展非常重要。

您可以在将对象模型进行多种扩展时选择使用两种方法的组合。对于对性能下降不敏感的系统区域，您可以使用第一个方法，而当性能是关键时，使用第二个方法。这样就可以在仍然维持较好的系统性能级别的同时，将以后迁移中的劳动量减小到最低程度。

会话 bean 的推荐使用

WebSphere Commerce 的强大功能之一源自于它利用容器管理持久性（CMP）实体 bean 的能力。CMP 实体 bean 是分布式、持久的、事物型、位于服务器端的 Java 组件，它们由 VisualAge for Java 中提供的工具生成。在很多情况下，CMP 实体 bean 是用于对象持久性的极佳选择；可以使 CMP 实体 bean 的工作至少与其它“对象至关系映射”选择一样高效率，或甚至比后者更高效。出于这些原因，WebSphere Commerce 使用 CMP 实体 bean，已经实现核心商务对象。

但有些情况下，建议使用会话 bean JDBC 帮助函数。这些情况包含以下情况：

- 查询返回大结果集的情况。这称为*大结果集*的情况。

- 查询从几个表中检索数据的情况。这指代为*聚集实体的情况*。
- SQL 语句执行数据库密集操作的情况。这称为 *arbitrary SQL* 的情况。

以下部分提供更多详细信息。

注意，如果正在将会话 bean 用作 JDBC 包裹程序以从数据库中检索信息，则实现资源级别访问控制变得更为困难。当以这种方式使用会话 bean 时，会话 bean 的开发者必须将适当的“where”子句添加到“select”语句中以防止非授权用户访问资源。

大结果集的情况: 有些情况中，查询返回大结果集，并且检索的数据主要用于读或显示目的。在此情况下，最好使用无状态的会话 bean，并在该会话 bean 中，创建查找函数方法，使其执行与实体 bean 中的查找函数方法相同的功能。即：该无状态的会话 bean 中的查找函数方法应执行以下操作:

- 执行 SQL select 语句
- 对于每个读取的行，实例化访问 bean
- 对于每个检索的列，设置访问 bean 中的相应属性

当返回访问 bean 时，命令不清楚访问 bean 是由会话 bean 中的查找函数方法返回，还是由实体 bean 中的查找函数方法返回。结果，在会话 bean 中使用 finder 方法不会导致对编程模型的任何更改。只有调用命令清楚它正在调用会话 bean 中的查找函数方法还是实体 bean 中的查找函数方法。它对于编程模型的所有其它部分都是透明的。

聚集实体的情况: 在此情况下，一个视图将几个对象的不同部分组合在一起，并且单个的显示页面以来自几个数据库表的许多信息填充。例如，考虑概念“我的帐户”。它由来自顾客信息表的信息（例如，顾客姓名、年龄和顾客标识）和来自地址表的信息（例如，由街道和市／县／区组成的地址）构成。

构造简单的 SQL 语句来通过执行 SQL join，从不同表中检索所有信息，这是可能的。这可称为执行“深层读取”。以下是“我的帐户”示例的 SQL select 语句的示例，其中 CUSTOMER 表是 T1，ADDRESS 表是 T2:

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

VisualAge for Java 中的 EJB 工具不支持此概念“深层读取”。相反，它执行惰性读取，从而为每个关联的对象产生 SQL select。它不是检索此类型的信息的首选方法。

要执行深层读取，建议使用会话 bean。在该会话 bean 中，创建查找函数方法来检索必需的信息。查找函数方法应执行以下操作:

- 执行深层读取的 SQL select 语句
- 为主表中的每行以及每个关联的对象实例化访问 bean。
- 为每个读取的列和每个读取的关联对象，在访问 bean 中设置相应的属性。

注意：访问 bean 不高速缓存抛出异常的获取函数方法。在此情况下，您应使用以下模式，为访问 bean 创建简单的包装类:

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

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

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

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

继续 CUSTOMER 和 ADDRESS 示例，会话 bean 查找函数方法将为 CUSTOMER 表中的每行实例化 CustomerAccessBean，并为 ADDRESS 表中的每个相应行实例化 AddressAccessBean。然后，对于 ADDRESS 表中的每列，它设置 AddressAccessBean（街道和市／县／区）中的属性。对于 ADDRESS 表中的每列，它设置 CustomerAccessBean（名称、年龄和地址）中的属性。这显示在下图中。

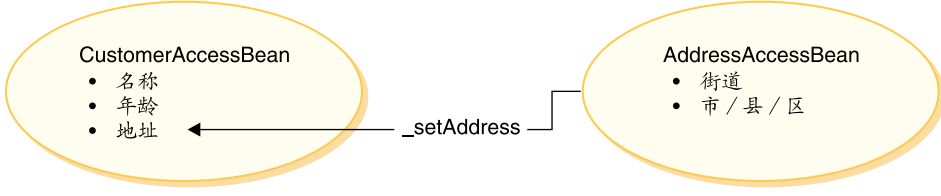


图 16.

Arbitrary SQL 的情况: 在此情况下，有一系列的执行数据库密集操作的 arbitrary SQL 语句。例如，表中的所有行的求和操作，被视为数据库密集操作。可能不是所有选定的行对应持久模型中的实体 bean。

可产生创建 arbitrary SQL 语句的示例就是当顾客尝试浏览整个大型数据集时。例如，如果顾客希望查看在线五金商店中的所有紧固件，或在线服装商店中的所有

服装。这会创建非常大的结果集，但从结果集中，很可能只需要每行中的一些字段。即：可能只在初始时向顾客显示商品名称、图片和价格的摘要。

在此情况下，创建会话 bean 帮助函数方法。此会话 bean 帮助函数方法执行读或写操作。当执行读操作时，它返回用于显示目的的只读值对象。

如果采用恰当的数据模型构建，出现 arbitrary SQL 语句的情况的次数通常可降低到最小程度。

扩展公共实体 bean

本节描述了 WebSphere Commerce 公共实体 bean 的设计模式。您可以使用这种设计模式进行扩展，例如添加新的持久字段、新的业务方法或新的查找函数方法。

下图显示了 Catalog 实体 bean 的实现类。

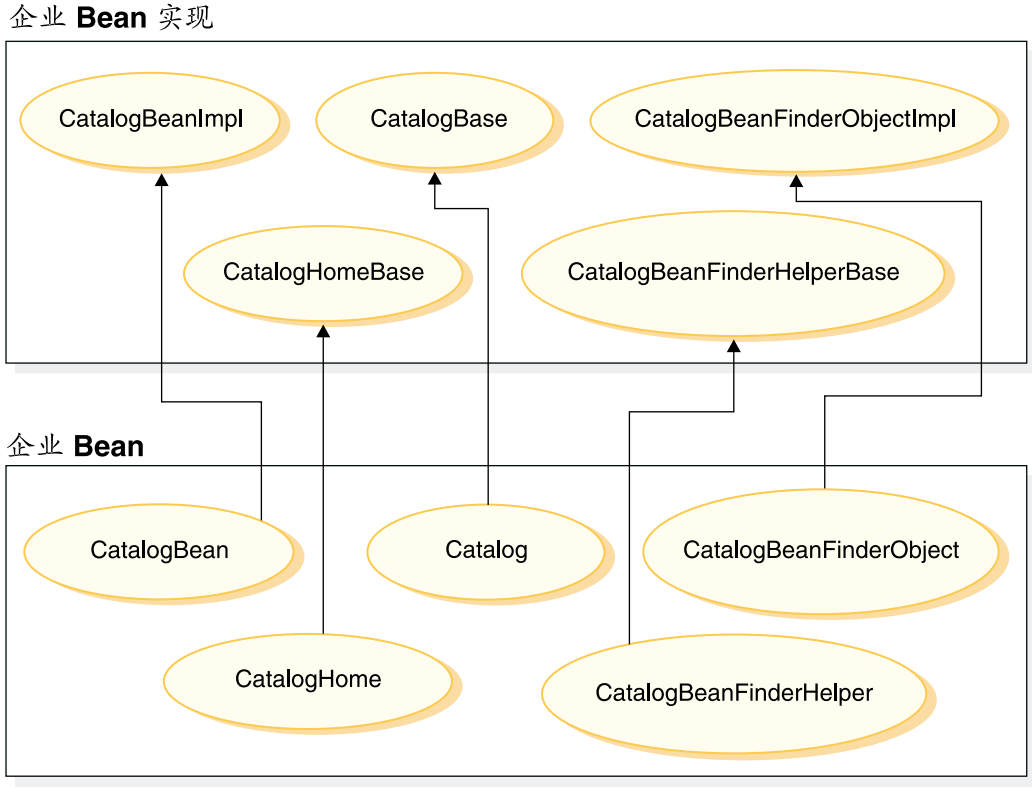


图 17.

上图也适用于其它实体 bean，这是因为它们的结构样式相似，并且它们遵循相同的命名约定。要将此图应用到另一个实体 bean，可以用实体 bean 的名称替换 “Catalog”。例如，InterestItemBean 类扩展 InterestItemBeanImpl 类，InterestItem 接口扩展 InterestItemBase 接口。

该图显示了公共企业 bean 的实现类或接口使用 Java 继承已经被分为两部分。超类或接口包含 WebSphere Commerce 实现代码。所有这些超类和接口都在子类和接口的独立数据包和项目中定义。

除 *bean_name*BeanFinderHelperBase 和 *bean_name*BeanFinderObjectImpl 类之外，WebSphere Commerce 代码资源库包含所有这些超类和接口的二进制代码。*bean_name*BeanFinderHelperBase 和 *bean_name*BeanFinderObjectImpl 类的源代码也包含在内，以便使您可以了解每个查找函数的 SQL 语句是如何定义的。您不要以任何方式修改这些类。

要检查 CatalogBeanFinderHelperBase 和 CatalogBeanFinderObjectImpl 的源代码，请打开 com.ibm.commerce.catalog.objsrc 数据包。其它数据包都使用相似的命名约定。

可以对子类和接口进行修改。

如果向公共企业 bean 添加新的查找函数方法，则必须遵循这些方法的特定命名约定。请将新方法命名为 findX*a_description*，其中 *a_description* 是您自己选择的描述。这样的名称有 findXByOwnerId 和 findXByOrderStatus。使用此命名约定可以避免与 WebSphere Commerce 查找函数方法产生名称冲突（重复名称）的风险。

一种修改现有 WebSphere Commerce 公共实体 bean 的方法是添加附加字段。在此情况下，添加新字段后，必须检查 bean 中的每种查找函数方法。如果查找函数方法的 where 子句部分包含任何数据库别名（例如，T1. 或 T2.），则必须除去别名。

创建新 CMP 企业 bean

需要将新属性添加到 WebSphere Commerce 对象模型时，可以创建一个新数据库表，其中有必需属性对应的列。然后还必须将此属性包含在企业 bean 中，以便 WebSphere Commerce 命令可以访问该信息。

一种将新属性集成到 WebSphere Commerce 对象模型中的方法是创建新的 CMP 企业 bean。然后在此 bean 中创建与新数据库表中属性相对应的字段。

要创建新 CMP 企业 bean，必须用 VisualAge for Java 执行以下步骤:

1. 确保工作空间所有者已设置为 WCS 开发者。

2. 为 bean 创建新 EJB 组。
3. 使用 “创建企业 Bean” 智能向导工具创建新 CMP 企业 bean。
4. 为相应数据库表中的每一列，将一个新的 CMP 字段添加至 bean。
5. 如有必要，为每个创建的 CMP 字段创建一对获取函数和设置函数方法。
6. 如有必要，在 FinderHelper 接口中定义 FinderHelper 字段，并添加新的 FinderHelper 方法。
7. 如有必要，创建一个新的 ejbCreate 方法，并将 ejbCreate 方法提升到企业 bean 的主接口。如果新企业 bean 必须在其相应的数据库表中创建新条目，则此步骤是必需的。
8. 将企业 bean 中的字段映射至数据库表的列。
9. 生成企业 bean 的访问 bean 和部署代码。

以下部分中包含了关于上面每个步骤的更多详细信息。

注: 如果新的企业 bean 受 WebSphere Commerce 访问控制系统保护，请参阅第 79 页的第 4 章，『访问控制』获取进一步的详细信息。当创建 bean 后可以添加访问控制。

假设一个新表的名称为 USERRES，它指定了关于用户居住类型的一些信息。此表包含三列: USERID 列、指定房屋类型的 HOME 列，以及指定住所中卧室数目的 ROOMS 列。

创建新 EJB 组: 在创建新实体 bean 时，必须在与 WebSphere Commerce EJB 组分离的 EJB 组中创建它们。在 VisualAge for Java 处理企业 bean 时，必须切换到工作空间中的 EJB 选项卡上。然后，从 **EJB** 菜单中，可以选择**添加 > EJB 组**，启动 “添加 EJB 组” 智能向导。

创建 EJB 组时，必须指定两个非常重要的项: 存储 EJB 代码的项目和 EJB 组的名称。

EJB 项目可在工作空间的 “项目” 选项卡中查看。创建新 EJB 组时，必须指定一个与 WebSphere Commerce 项目分离的项目。例如，可以选择让您的 EJB 组使用 MyCustomEJB 项目。在创建该组以前此项目不必存在，因为 VisualAge for Java 可以为您自动创建。此项目应当只用于 EJB 代码；而不应用于任何命令或数据 bean 代码。为了部署，这种代码类型的分离是必需的。通过将您自己的定制代码与 WebSphere Commerce 代码相分离，可以将迁移到以后的发行版所受到的影响最小化。

对于项目和 EJB 组的名称，请确保您遵循任何适当的应用程序命名约定。

创建新 CMP 企业 bean: 要创建新 CMP 企业 bean，可以使用“创建企业 Bean”智能向导工具。要启动此工具，请用鼠标右键单击要向其中添加新 bean 的 EJB 组，并选择**添加 > 企业 Bean**。

选择创建新企业 bean，并为此 bean 指定名称。企业 bean 的 WebSphere Commerce 命名约定是使用与 bean 对应的表相同的名称命名 bean。例如，如果新数据库表的名称是 USERRES，则企业 bean 应当命名为 UserRes。

“项目”字段是用项目名称自动填充的。您必须在应当存储代码的项目中指定数据包名称。举例来说，要存储在数据包中的代码有为企业 bean 创建的访问 bean 的代码。再次指出，命名数据包时请确保遵循所有适用于您的应用程序的命名约定。数据包名称的示例是 `com.mycompany.mycustombeans`。

在 bean 类中，VisualAge for Java 创建名为 `EntityContext` 的私有字段。WebSphere Commerce 在 ECEntityBean 中提供它自己的实体上下文字段，并且新实体 bean 应当使用该字段而不是在您自己的类中生成的字段。这样，您必须从新实体 bean 中除去生成的 `EntityContext`。

新的企业 bean 必须包含 `serialVersionUID` 字段。如果使用智能向导创建新 bean，则 VisualAge for Java 为您生成此字段。如果不使用该工具创建 bean，则您必须添加此字段。

在“超类”字段中，必须指定 `com.ibm.commerce.base.objects.ECEntityBean` 类。以下代码示例演示了超类所提供的功能:

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

您还应该除去所有不带参数的 ejbCreate() 和 ejbPostCreate() 方法。将这些方法留在实体 bean 中会造成运行时错误。

对数据库表的每一列，您都必须在 bean 中创建新 CMP 字段（必须在“智能向导”中单击“下一步”查看“添加 CMP 字段至 bean”选项）。对于每个字段，都应指定字段名称和字段的数据类型。对主键或主键一部分的任何列，启用“键字段”复选框。对于所有其它列，启用“将获取函数和设置函数方法提升到远程接口”复选框。

一旦填写了所有字段，请单击“完成”，VisualAge for Java 将创建新的 CMP 企业 bean。

在 Bean_NameFinderHelper 接口中创建新 FinderHelper 字段: *Bean_Name*FinderHelper 接口包含对应所有 FinderHelper 方法（而不是 findByPrimaryKey 方法）的 SQL 搜索子句。

新企业 bean 使用在 bean 的 FinderHelper 接口中定义的 findXBy*ArgName*（其中 *ArgName* 是参数的名称）方法，以及 FinderHelper 方法来编写 SQL 查询。对字段名称使用“findXBy”命名约定，以确保您的字段名称总是有别于 WebSphere Commerce 字段名称且唯一。

要在 bean 中创建新的 FinderHelper 字段，必须选择 *Bean_Name*_FinderHelper 接口并修改源代码来确定如何构造 select 语句。例如:

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

然后主接口将需要一个名为 findXByHomeAndRooms 的方法，它使用每个 HOME 和 ROOMS 的输入参数（这些参数填充以 ? 字符表示的值）。这种类型的查询构造称为*参数插入*。

如果输入参数是“detached”和“3”，则生成的 SQL 语句将是

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

出于安全性原因，为新实体 bean 创建 FinderHelper 方法时，应当使用参数插入。这样建议的原因是它可以保护查询不受用户更改。另一种方法是使用与以下类似的构造:

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

其中 *input_string* 是从 URL 传递来的字符串值。这方法并不尽如人意，因为恶意用户可以输入类似于“‘123’ OR 1=1”的值来更改 SQL 语句。如果用户可以更改 SQL 语句，他们就可以对数据进行未授权访问。因此，建议的方法是使用参数插入。

如果无法使用参数插入，因而不得不使用输入字符串来编写 SQL 语句，则必须对输入字符串强制进行参数检查，以确保输入参数没有恶意尝试访问数据。

在 Bean_NameHome 接口中创建新 FinderHelper 方法: 对于每个在 *Bean_Name*FinderHelper 接口中指定的 FinderHelper 字段，都必须在 bean 的主接口中创建 FinderHelper 方法。FinderHelper 方法的名称必须与 FinderHelper 字段

的名称完全匹配（除了删除“WhereClause”之外）。即：对于 findXByHomeAndRoomsWhereClause 的示例字段名称，相应的方法名称是 findXByHomeAndRooms。

要创建新 FinderHelper 方法，请执行以下操作:

1. 用鼠标右键单击 *Bean_Name***Home** 接口，并选择**添加 > 方法**。“创建方法”智能向导打开。
2. 选择**创建新方法**并单击**下一步**。
3. 在**方法名称**字段中，输入 FinderHelper 方法的名称。FinderHelper 方法的此名称必须与 FinderHelper 字段的名称完全匹配（除了删除“WhereClause”部分之外）。例如，输入 findXByHomeAndRooms。
4. 在**返回类型**字段中，输入以下内容之一:
   - 如果 FinderHelper 方法使用主键来查询数据库且方法应当返回唯一记录，请将 EJB 对象指定为返回类型。例如，输入 UserRes。
   - 如果 FinderHelper 方法返回一个结果集而不是唯一记录，则应将返回类型指定为 java.util.Enumeration。
5. 单击**此方法应有什么参数?** 旁边的**添加**按钮来添加适当的参数。例如，可以添加类型为 String 的 argHome 来保存居住类型，添加类型为 byte 的 argRooms 来保存房间的数目。
6. 完成添加所有参数后，单击**下一步**。
7. 单击**此方法会抛出什么异常?** 旁边的**添加**按钮，并添加以下异常:
   - java.rmi.RemoteException
   - javax.ejb.FinderException

   注: 上列异常显示了方法应抛出的最低限度的异常集。根据您自己的代码，您可能需要指定其它异常。
8. 单击**完成**。

创建新 ejbCreate 方法: 企业 bean 创建时会自动生成 ejbCreate 方法。然后，此方法被提升到远程接口，从而使它在访问 bean 中可用。缺省 ejbCreate 方法中仅包含以下两种参数: 主键或主键的一部分。这就是说，在实例化时只有那些值会得到实例化。

如果您的企业 bean 包含非主键部分和不得为空的字段，则必须创建新的 ejbCreate 方法，以在其中明确地将那些字段实例化。通过这样的处理，每次创建新记录时，所有不得为空的字段都会用适当的数据填充。

要创建新 ejbCreate 方法，请执行以下操作:

1. 在“类型”窗格中，展开 ***Bean_Name*Bean** 类。例如，选择 **UserResBean**。
2. 单击现有 ejbCreate 方法来查看源代码。（注意：它可以是 ejbCreate(String)、ejbCreate(String, int)，或者使用其它输入参数，这取决于企业 bean 的主键。）
3. 您必须修改源代码，使每个 CMP 字段都作为方法的输入参数包含在其中，这样每个 CMP 字段都用适当的值实例化。在 UserRes 示例中，UserId 是主键，源代码最初显示为:

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

   但是，您可能希望确保将房间数与房屋类型初始化。在此情况下，可以将代码改为:

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

   注：如果希望使用系统生成的主键，请参阅第 70 页的『主键』获取详细信息。
4. 保存修改后的方法。保存代码时，VisualAge for Java 将创建新 ejbCreate 方法来使用新参数。原始的 ejbCreate 方法被保留。
5. 删除原始的 ejbCreate 方法（不带参数的 ejbCreate 方法）。
6. 用鼠标右键单击新 ejbCreate 方法，并选择**添加 > EJB 主接口**。
7. 创建相应的 ejbPostCreate 方法。（此方法不需要添加到主接口中。）

将数据库表映射到新企业 bean: 一旦创建了新企业 bean，就必须在 bean 的 CMP 字段与数据库表的列之间创建映射。这种映射称为模式。VisualAge for Java 提供了简化此任务的工具。

要创建模式，请执行以下操作:

1. 从 **EJB** 菜单中，选择**打开到 > 数据库模式**。“模式浏览器”打开。
2. 从**模式**菜单中，选择**导入／导出模式 > 从数据库导入模式**。
3. 为新模式输入名称，并单击**确定**。

4. 在“数据库连接”窗口中，如下输入信息:

属性	DB2 值	Oracle 值
连接类型	COM.ibm.db2.jdbc.app.DB2Driver	Oracle.jdbc.driver.OracleDriver
数据源	jdbc:db2:wc_database_name	jdbc:oracle:thin@*hostname*:*port*:*Oracle_SID*
用户名	*wc_db_user_name*	*wc_db_user_name*
密码	*wc_db_password*	*wc_db_password*

如下替换其中的值:

- DB2 *wc_database_name* 是 WebSphere Commerce 数据库的名称
- Oracle *hostname* 是 Oracle 服务器主机名。
- Oracle *port* 是 Oracle 数据库的端口号。
- Oracle *Oracle_SID* 是 Oracle 实例标识。
- *wc_db_user_name* 是数据库的用户名。
- *wc_db_password* 是数据库密码。

5. 从**限定符**列表中选择数据库限定符（它可以是数据库用户名）。
6. 单击**构建表列表**。
7. 从生成的列表中选择 *your_new_table*，并单击“确定”生成模式。
8. 模式生成后，单击“模式”窗格中的 *your_new_table*。
9. 突出显示“表”窗格中的 *your_new_table*，然后双击它。
   “表编辑器”窗口打开。
10. 除去**限定符**字段中的任何信息。这将使企业 bean 可以移植到其它机器中。
11. 从**模式**菜单中选择**保存模式**。输入相应的项目、数据包和类名。

接下来必须创建模式映射。模式映射是数据库列与企业 bean 中的字段之间的映射。

要创建模式映射，请执行以下操作:

1. 从 **EJB** 菜单中选择**打开到 > 模式映射**。
   “数据仓库映射”窗口打开。
2. 输入映射的名称。请参阅第 72 页的『数据库模式对象命名注意事项』获取有关命名新映射的推荐信息。
3. 选择 EJB 组和模式。请参阅第 72 页的『数据库模式对象命名注意事项』获取有关命名新模式的推荐信息。

4. 在“数据仓库映射”窗格中选择映射。
5. 在“持久类”窗格中选择类。
6. 从**表映射**菜单中，选择**新建表映射 > 无继承添加表映射**。
7. 从**表**下拉列表中，选择表并单击**确定**。
8. 在“表映射”面板中，突出显示并用鼠标右键单击您的表映射，并选择**编辑属性映射**。
   “属性映射编辑器”打开。
9. 对于每个类属性（bean 中的 CMP 字段），您必须指定映射类型以及它应该映射到的数据库列。例如，使用映射类型“简单”将 UserId 类属性映射到数据库表的 USERID 列。
10. 在将所有字段映射到它们相应的数据库列后，必须保存模式映射。从**数据库映射**菜单中选择**保存数据仓库映射**。输入相应的项目、数据包和类名以保存映射。单击**完成**。

创建访问 bean 和生成部署代码: 访问 bean 扮演着企业 bean 包装的角色，它简化了其它组件与企业 bean 之间的相互作用。必须为新的企业 bean 创建访问 bean。

生成部署代码时，VisualAge for Java 中的工具分析 bean 以确保符合 Sun Microsystems EJB 规范中指定的规则和特定于 EJB 服务器的规则。

要为新企业 bean 创建访问 bean，请执行以下操作:

1. 在“企业 Bean”窗格中，用鼠标右键单击新企业 bean，并选择**添加 > 访问 Bean**。（您可能必须首先展开包含新 bean 的 EJB 组才能查看 bean。）
   “创建访问 Bean”智能向导打开。
2. 在 **EJB 组**、**企业 bean** 和**访问 bean 名称**字段中，指定适当的 EJB 组、bean 名称和访问 bean 名称。
3. 选择**访问 Bean 类型实体 Bean 的复制帮助函数**，并单击**下一步**。
4. 从**为零参数构造函数选择主方法**下拉列表，选择 **findByPrimaryKey**。
5. 从**转换器**下拉列表，为初始属性选择 **WCSStringConverter**，并单击**下一步**。
6. 在“为复制帮助函数选择并定制 Bean 属性”窗口中，为每个字段选择 **WCSStringConverter**。
7. 单击**完成**。

要生成部署代码，请执行以下操作:

1. 在“企业 Bean”窗格中，用鼠标右键单击新企业 bean，并选择**生成部署代码**。

注意，使用此工具生成的部署代码用 EJB 1.0 规范编译，而且仅当在 VisualAge for Java 中运行企业 bean 时才使用它。稍后当您将企业 bean 部署到在 WebSphere Application Server V4.0 中运行的 WebSphere Commerce 应用程序时，您需要生成一个 JAR 文件，它包含用 EJB 1.1 规范编译的部署代码。有关创建该 EJB 1.1 Export JAR 文件的更多信息，请参阅第 171 页的『有关 EJB 部署代码的信息』和第 322 页的『生成部署代码』。

使用测试客户机测试企业 bean: VisualAge for Java 提供可用于测试企业 bean 的测试客户机。要使用测试客户机测试您的新 bean，请执行以下操作:

1. 启动包含正在测试的企业 bean 的 EJB 服务器。
2. 用鼠标右键单击企业 bean 并选择**运行测试客户机**。
   “EJB 测试客户机”和“EJB 查找”窗口打开。
3. 在 **JNDI 名称**字段，输入企业 bean 的 JNDI 名称并单击**查找**。
4. 用鼠标右键单击填写有参数的 **findByPrimaryKey** 方法，并选择**调用**。

编码练习: 请仔细研究以下企业 bean 编码练习:

- 请不要使用 BLOB 或 CLOB 数据类型。
- LONG 数据类型（也称为 LONG VARCHAR）的 CMP 字段都不可以作为企业 bean CMP 字段列表的第一个和最后一个成员。要验证该列表，请检查 `EJSJDBCPersister._hydrate()` 方法，并查看列表中的第一和最后一个元素是否是 LONG VARCHAR 类型。如果第一个字段是此类型，请执行以下操作:
  1. 取消设置第一个字段。称其为 `fieldA`。
  2. 取消设置另一个*不是* LONG VARCHAR 类型的字段。称其为 `fieldB`。
  3. 重新设置 `fieldA`。
  4. 重新设置 `fieldB`。
  5. 打开“模式映射浏览器”并编辑表映射以反映这些更改。保存映射。
  6. 重新生成部署代码。
- 企业 bean 不应引用企业 bean 数据包外部的任何项。例如，在企业 bean 代码中不应引用命令或数据 bean
- 要为企业 bean 启用访问控制，请将 com.ibm.commerce.security.Protectable 接口和／或 com.ibm.commerce.security.Groupable 接口添加至企业 bean 的远程接口。添加这些接口后，重新生成 bean 的部署代码和访问 bean。同样必须在 objsrc 数据包中创建一个访问帮助函数类对象。

创建简单数据 bean

数据 bean 是在 JSP 模板中用于从企业 bean 检索信息的 bean。简单数据 bean 扩展其相应的访问 bean，并实现 SmartDataBean 接口。数据 bean 的大部分代码都由 VisualAge for Java 自动生成。

要创建简单数据 bean，必须执行以下步骤:

1. 创建项目和数据包来存储数据 bean 代码。
2. 创建数据 bean，它扩展相应的访问 bean，并实现适当的数据 bean 接口。
3. 为数据 bean 创建设置方法。
4. 为数据 bean 创建获取方法。

为数据 bean 代码创建项目和数据包: 创建项目和数据包可创建存储数据 bean 代码的位置。

要创建新项目，请执行以下操作:

1. 选择**项目**选项卡。
2. 从**已选项目**菜单中，选择**添加 > 项目**。
   “添加项目”智能向导打开。
3. 确保已经选择**创建新项目名为**，并为新项目输入名称。例如，输入我的数据 Bean。
4. 单击**完成**。

要创建新数据包，请执行以下操作:

1. 用鼠标右键单击已为数据 bean 代码创建的项目，并选择**添加 > 数据包**。例如，用鼠标右键单击**我的数据 Bean**，并选择**添加 > 数据包**。
   “添加数据包”智能向导打开。
2. 确保已经选择**创建新数据包名为**，并为数据 bean 数据包输入合适的名称。例如，输入 com.mycompany.mydatabeans。
3. 单击**完成**。

创建数据 bean: 数据 bean 是在 JSP 模板内用于为页面提供动态内容的 Java bean。通常，它通过扩展访问 bean 来提供实体 bean 的简单表示（间接）。数据 bean 将可以从实体 bean 中检索或在实体 bean 中设置的属性封装起来。

要创建数据 bean，请执行以下操作:

1. 用鼠标右键单击您要向其中存储数据 bean 的数据包，并选择**添加 > 类**。
   “创建类”智能向导打开。
2. 项目和数据包名称字段已经填充。

3. 确保已经选择**创建新类**，并单击**下一步**。
4. 在**类名**字段中，为新数据 bean 输入一个名称。例如，要创建扩展 UserResAccessBean 的数据 bean，请输入 UserResDataBean。
5. 要指定超类，请单击**浏览**，然后在模式字段中输入相应访问 bean 的名称。例如，输入 UserResAccessBean 并单击**确定**。
6. 单击**下一步**。
7. 要指定数据 bean 应当实现的接口，请单击**添加**。在“接口”窗口中，请执行以下操作:
   a. 在**模式**字段中，输入 com.ibm.commerce.beans.SmartDataBean，然后单击**添加**。
   b. 在**模式**字段中，输入 com.ibm.commerce.beans.InputDataBean，然后单击**添加**。
   c. 单击**关闭**。
8. 单击**完成**。

添加必需字段至数据 bean: 本部分描述了如何将必需字段添加到新数据 bean。

要添加 iCommandContext 字段，请执行以下操作:

1. 用鼠标右键单击新数据 bean（例如 UserResDataBean），并选择“添加” > “字段”。
   “创建字段”智能向导打开。
2. 在**字段名称**字段中，输入 iCommandContext。
3. 单击**浏览**添加字段类型，并输入 com.ibm.commerce.command.CommandContext。单击**确定**。
4. 对于访问修饰符，选择 **Protected**。
5. 单击**完成**。

要添加 iRequestProperties 字段，请执行以下操作:

1. 用鼠标右键单击新数据 bean（例如 UserResDataBean），并选择“添加” > “字段”。
   “创建字段”智能向导打开。
2. 在**字段名称**名称中，输入 iRequestProperties。
3. 单击**浏览**添加字段类型，并输入 com.ibm.commerce.datatype.TypedProperty。单击**确定**。
4. 对于访问修饰符，选择 **Protected**。
5. 单击**完成**。

修改数据 bean 的设置方法: 当创建了数据 bean 后，您必须在一些生成的设置方法中修改代码。

要更新设置方法，请执行以下操作:

1. 展开新数据 bean 查看其字段和方法。
2. 选择 **setCommandContext(CommandContext)** 方法查看其源代码。
   “源码”窗格显示源代码如下:

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

3. 修改源代码，使方法如下显示:

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

   保存工作（Ctrl+S）。

4. 选择 **setRequestProperties(TypedProperty)** 方法查看其源代码。
   “源码”窗格显示源代码如下:

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

5. 您可能希望修改源代码以填充相应访问 bean 的主键。对执行此操作建议的方法是使用数据 bean 管理器来间接设置该值。此间接方法设计来确保取自 URL 属性的主键值在先前已经设置主键时不会覆盖主键。要使 setRequestProperties 方法遵照此模型，请采用与以下代码片段类似的方式对它进行编码。注意: 在以下示例中，主键是用户标识。这根据情况可能会不相同（这样，以下代码可能不直接在应用程序中编译）。

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

有两种其它方法可以设置访问 bean 的主键。这可以从数据 bean 的外部完成，例如在 JSP 模板中。在此情况下，在 JSP 模板中激活数据 bean 之前，显式调用主键的数据 bean 设置方法。例如，JSP 可包含与以下类似的代码（其中 db 是数据 bean 对象）:

```
db.setInitKey_UserId(/*input parameter*/)
db.activate();
```

另一种代替方法是，可直接设置主键。即：JSP 模板仅包含 db.activate 方法，然后数据 bean 管理器显式设置访问 bean 中的主键。例如，数据 bean 的 setRequestProperties 方法的代码显示与以下类似:

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

注意：对设置主键建议的过程是步骤 5 中所示的间接方法。

修改数据 bean 的获取方法: 当创建了数据 bean 后，您必须在一些生成的获取方法中修改代码。

要更新获取方法，请执行以下操作:

1. 展开新数据 bean 查看其字段和方法。
2. 选择 **getCommandContext()** 方法查看其源代码。
   “源码”窗格显示源代码如下:

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

3. 修改源代码，使方法如下显示:

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

   保存工作（Ctrl+S）。

4. 选择 **getRequestProperties()** 方法查看其源代码。
   “源码”窗格显示源代码如下:

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

5. 修改源代码，使其如下显示:

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

   保存工作（Ctrl+S）。

修改 populate() 方法: 必须通过执行以下操作，修改填充方法:

1. 展开新数据 bean 查看其字段和方法。
2. 选择 **populate()** 方法查看其源代码。
   “源码”窗格显示源代码如下:

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

3. 修改源代码，使方法如下显示:

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

   保存工作（Ctrl+S）。

写新会话 bean

在创建新会话 bean 时，必须在与 WebSphere Commerce EJB 组分离的 EJB 组中创建会话 bean。将此新 EJB 组存储在与 WebSphere Commerce 项目分离的新项目中。例如，您可能在 `com.mycompany.mycustomcode` 数据包中创建 `MyCustomBeans` EJB 组。通过将您自己的定制代码与 WebSphere Commerce 代码相分离，可以将迁移到以后的发行版所受到的影响最小化。

新的会话 bean 应扩展 com.ibm.commerce.base.helpers.BaseJDBCHelper 类。超类提供了使您能够从贸易服务器使用的数据源获取 JDBC 连接对象的方法，从而使会话 bean 可以参与与其它实体 bean 相同的事务。以下代码示例演示了超类所提供的功能:

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

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

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

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

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

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

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

   }

}
```

在先前的代码示例中，executeQuery 方法采用两个输入参数。第一个是已准备的语句，第二个是与高速缓存刷新操作相关的布尔标志。如果在执行查询之前您需要容器从高速缓存中刷新当前交易的所有实体对象，请将此标志设置为 true。如果您已对一些实体对象执行了更新而且您需要查询搜索这些更新的对象，就需要这样做。如果标志设置为 false，则不会将这些实体对象更新写入到数据库，直至交易结束。

您应当限制该刷新操作的使用，而且通常情况下将标志设置为 false，除非真的需要这样做。刷新操作是资源密集型操作。

对象生命周期

对象模型中的企业 bean 包括*独立*和*从属*对象。独立对象有自己的生命周期，由调用对象的业务逻辑的创建或除去请求直接控制。从属对象的生命周期附加在另一个对象上，称为*所有者对象*（它因此也可能是从属对象，但深入查看关联层次结构会发现，总是存在独立对象）。当删除所有者对象后，所有从属对象也被删除。实际的删除由数据库内的级联删除规范所控制。

例如，假设一个用户对象返回通讯录对象和一列订单对象，如果该用户对象被删除，则其通讯录对象也被删除（因为通讯录是用户所有的），同样通讯录中的所有地址对象也被删除（因为地址是通讯录所有的）。但是订单对象并不会删除，因为订单的所有者是商店对象，而不是用户对象。

创建从属对象时使用特定的设计模式。从属对象的创建方法必须提供对其所有者对象的引用；因此所有者对象必须在创建从属对象前便已存在。

事务

Enterprise JavaBeans V1.0 体系结构指定了对应于实例状态的三个可相互替换的提交时间选项。在规范文档中它们被描述为选项 A、B 和 C。关于这些选项的完整详细信息，请参阅 Sun Microystem 的 Enterprise JavaBean V1.0 规范文档。

虽然 WebSphere Application Server 可以实现选项 A 和 C，但选项 A 假定数据库没有共享。

在选项 C 中，企业 bean 容器不高速缓存事务之间的 “ready” 实例。一旦事务完成，实例就会返回到可用实例的池中。WebSphere Commerce 使用选项 C 是因为数据库在多个 WebSphere Commerce 应用程序之间共享。在此实现中，容器在每个事务开始时装入实体 bean 的持久数据，且仅在事务持续时间内高速缓存实体 bean。容器激活一个实体 bean 的多个实例，每个都对应于在其中访问实体的事务。事务同步由数据库执行。

每个企业 bean 的事务属性都设置为 TX_REQUIRED。由于 Web 控制器先开始事务，然后执行访问企业 bean 的命令（通过其相应的访问 bean），因此会在此事务的上下文中调用企业 bean 的业务方法。

实体 bean 的其它注意事项

查找更新

为了更新行，多个应用程序能够访问数据库中同一行的情况称为*并行更新*。有些情况下允许进行并行更新，而其它情况下是绝对不希望的。

如果数据库更新是重写，即新值与数据库中当前的值没有任何关系，则允许进行并行更新。如果允许并行更新，且有多个应用程序试图更新数据库中的同一行，则最后一个尝试是在数据库中得到更新的尝试。

如果数据库更新依赖于数据库中的当前值，则不希望进行并行更新。例如，如果应用程序正在更新产品库存，则应当一次只允许一个应用程序更新库存。

影响是否允许并行更新的因素包括数据库锁和企业 bean 隔离级别。

为防止另一个应用程序并行更新某行，第一个访问该行的应用程序必须使用 “查找更新” 选项读取该行。使用 “用于更新” 选项时，对该行会应用*写锁*（也称为专用锁）。对行应用此写锁后，将阻拦任何应用程序试图使用 “查找更新” 访问该行。

如果应用程序允许执行并行更新，则它只能读取数据，而不锁定行。

假如有这样一个 OrderProcess 方案，UpdateInventory 需要查找订单中包括的所有产品，并相应更新库存。由于同样的产品可能包含在许多其它的订单中，因此应当使用*查找更新*，且应当在事务作用域内使用得越早越好，以减小死锁的可能性。因此，UpdateInventory 算法可以用以下伪代码表示:

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

在长期运行的“商家到商家”方案中，一份订单可能有多个商品，因此应尽早使用“查找更新”。逻辑可能如下:

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

刷新远程方法

由于在事务提交时间前，WebSphere Application Server 不将对实体 bean 所做的更改写入数据库，因此数据库可能暂时与实体 bean 容器中高速缓存的数据不同步。

我们提供了刷新远程方法（在 com.ibm.commerce.base.helpers.BaseJDBCHelper 类中），它可以写所有事务中所有提交的更改（即它从企业 bean 高速缓存中获取信息）并更新数据库。这种远程方法可以用命令调用。仅当绝对需要时才使用此方法，因为它对资源的开销非常大，从而会对性能有负面影响。

考虑一个具有以下代码片段的登录命令:

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

提交事务前，USER 表中的 REGISTEREDSTAMP 不会用当前时间戳记更新。更新只会发生在事务提交时。要使任何直接 JDBC 查询（在同一事务中）（例如，*select from user where registeredstamp ...*）返回具有指定注册时间戳记的用户，则必须使用刷新方法。

保护企业 bean

如果您在使用 WebSphere Application Server 来保护企业 bean，则必须使用 WebSphere Application Server 管理员控制台把任何新的企业 bean 的方法指定到 WCSMethodGroup 安全性方法组中。部署新企业 bean 时执行此步骤。另外，如果您修改了现有的 WebSphere Commerce 实体 bean，则您必须将受影响的 EJB 组中所有实体 bean 的方法指定到 WCSMethodGroup 安全性方法组。关于定制代码部署过程的描述，请参阅第 171 页的『代码部署』。

主键

主键是一个唯一键，它是表定义的一部分。它可以用来将一个记录与其它记录相区分。所有记录都必须有主键。在表中创建新记录时，您可能需要为记录生成唯一的主键。

在 WebSphere Commerce 编程模型中，持久层中包括与数据库相互作用的实体 bean。这样，在实例化实体 bean 时就可以创建数据库记录。因此实例化实体 bean 的 ejbCreate 方法可能需要包含为新记录生成主键的逻辑。

当应用程序需要来自数据库的信息时，它通过实例化 bean 的相应访问 bean，并获取或设置各种字段，来间接使用实体 bean。访问 bean 为数据库中的特定记录（例如，为特定用户简要表）被实例化，且它使用主键从数据库中选择正确的信息。

以下部分描述了如何创建唯一主键和如何通过主键进行选择。

创建主键: ejbCreate 方法用于实例化实体 bean 的新实例。此方法是自动生成的，但生成的方法只包含了把主键初始化为一个静态值的逻辑。

您可能需要确保主键是新的且唯一的值。在此情况下，您的 ejbCreate 方法可能与以下代码片段类似:

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

在前面的代码片段中，getNextKey 方法为主键生成了唯一整数。方法的 *table_name* 输入参数必须是对 KEYS 表中定义的 TABLENAME 值的完全匹配。字符和大小写一定要完全匹配。

除在 ejbCreate 方法中包含上述代码外，您还必须在 KEYS 表中创建条目。以下是在 KEYS 表中创建条目的示例 SQL 语句:

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

注意: 对于上述 SQL 语句，接受了 KEYS 表中其它列的缺省值。COUNTER 的值表示计数从该值开始。KEYS_ID 的值应当是任何正值。

如果您的主键定义为 long 数据类型（对于 DB2 是 BIGINT，对于 Oracle 是 NUMBER），请使用 getNextKeyAsLong 方法。

通过主键选择: 在访问 bean 内，必须使用主键选择适当的数据库记录。以下代码片段演示了如何执行此选择。它还包含附加逻辑，稍后将会解释它。

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

上述代码片段中的第一行实例化了名为“abUserProfile”的新 UserProfileAccessBean。第二行在访问 bean 中设置了主键。setInitKey_*xxx*（其中 *xxx* 是主键字段名称）命名约定由 VisualAge for Java 用于为主键的设置方法命名。实例化访问 bean 时，您应当确保 setInitKey_xxx 方法设置的所有字段都在使用 refreshCopyHelper 方法之前得到初始化。setInitKey_xxx 方法调用的顺序并不重要。

调用所有 setInitKey_xxx 方法后，便初始化了所有必需字段，且可以使用 refreshCopyHelper 方法从数据库检索信息了。

如果要更新访问 bean 的本地高速缓存中的值，则还必须将 commitCopyHelper 调用包含在其中，以便用已更新的信息来更新数据库。例如，如果在检索数据之后使用 refreshCopyHelper 方法（通过设置名称值）更新顾客名称，则您必须调用 `abUserProfile.commitCopyHelper()` 用新的属性更新数据库。

使用实体 bean

使用企业 bean 的程序必须处理“Java 命名和目录接口（JNDI）”以及企业 bean 的本地和远程接口。为简化编程模型，需要为每个企业 bean 生成访问 bean。创建您自己的企业 bean 时，请使用 VisualAge for Java 中的工具来生成此访问 bean。

WebSphere Commerce 命令与访问 bean 相互作用，而不是直接与实体 bean 相互作用。如图中所示，使用访问 bean 具有以下优点:

- 更简单的编程接口。访问 bean 就像一个 Java bean，它隐藏所有企业 bean 的特定编程接口，象 JNDI，主接口和远程接口，使它们对客户机不可见。
- 在运行时，访问 bean 高速缓存企业 bean 主对象，因为查找主对象既耗费时间也耗费资源的使用。
- 访问 bean 实现 copyHelper 对象，这样在命令获取和设置企业 bean 属性时可以减少对企业 bean 的调用次数。因此，在读写多个企业 bean 属性时只需要对企业 bean 进行一次调用。

下图显示了命令、访问 bean、实体 bean 和数据库之间的相互作用。

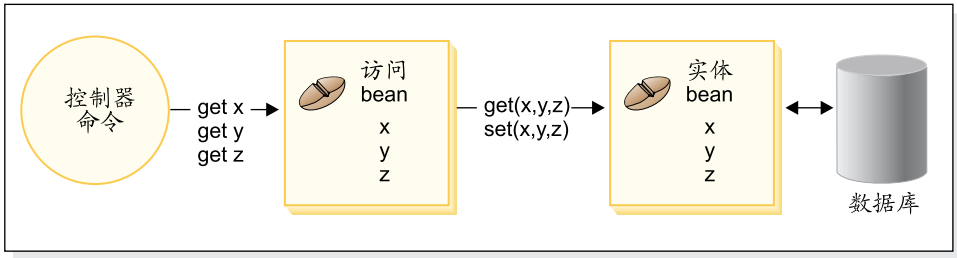


图 18.

数据库注意事项

定制电子交易应用程序时，可以创建新的数据库表。创建这些表时，建议您遵循一系列约定，以便以同 WebSphere Commerce 表一致的方式创建表。

数据库模式对象命名注意事项

下面的章节将提供对数据库模式对象的命名指导。

表和视图的命名约定

以下列表提供了对新表和新视图的命名指导:

- 为避免与将来发行版中 WebSphere Commerce 表和视图的名称冲突（重复名称），表或视图名称的第一个字符应当是 X。例如，XMYTABLE。
- 表或视图名称长度不应超过 10 个字符。如果希望的名称超过此限度，请从名称末尾除去元音来缩短长度，直到只剩下 10 个字符为止。
- 表或视图名称中不应包括任何特殊字符，例如“_”、“+”、“$”、“%”和空格。
- 不要使用数据库保留字作为表或视图名称。
- 视图名称应以 VW 结束。
- 表和视图名称应是单数名词。

列的命名约定

以下列表提供了对新表中列的命名指导:

- 为避免与将来发行版中 WebSphere Commerce 表中的列的名称冲突（重复名称），列名的第一个字符应当是 X。例如，XMYCOLUMN。
- 列名长度不应超过 18 个字符。如果希望的名称超过此限度，请从名称末尾除去元音来缩短长度，直到只剩下 18 个字符为止。

- 列名（而不是外键）不应包括任何特殊字符，例如“_”、“+”、“$”、“%”和空格。
- 不要使用数据库保留字作为列名。
- 可以将使用主动语态组合的组合词用作列名。例如，COMBINERESULT。
- 生成的主键列应命名为 *table*_id。例如，USERS 表的主键是 USERS_ID。
- 生成的外键列名不应更改。
- 如果保留任何列供以后定制，它们应当命名为 field*x*，其中 *x* 是从 1 开始的数字。

索引的命名约定

以下列表提供了对新表中索引的命名指导:

- 索引名称长度不应超过 18 个字符。
- 索引名称中不应包含空格。
- 索引名称中不应包含任何数据库保留字。
- 非唯一性索引应当命名为 I_*tablex*，其中 *table* 是表名，*x* 是从 1 开始的数字。例如，USERS 表的非唯一性索引是 I_USERS1。
- 唯一性索引应当命名为 UI_*tablex*，其中 *table* 是表名，*x* 是从 1 开始的数字。例如 USERS 表的唯一性索引是 UI_USERS1。
- 索引的总大小应不超过 254 个字节。
- 索引名称在整个数据库模式中必须是唯一的。

主键的命名约定

以下列表提供了对新表的主键的命名指导:

- 主键名称长度不应超过 18 个字符。
- 主键名称中不应包含空格。
- 主键名称中不应包含任何数据库保留字。
- 主键应当命名为 P_*table*，其中 *table* 是表名。例如，USERS 表的主键是 P_USERS。
- 主键名称在整个数据库模式中必须是唯一的。

外键的命名约定

以下列表提供了对新表的外键的命名指导:

- 外键名称长度不应超过 18 个字符。
- 外键名称中不应包含空格。
- 外键名称中不应包含任何数据库保留字。

- 外键应当命名为 F_*table*，其中 *table* 是表名。例如，USERS 表的外键是 F_USERS1。
- 外键名称在整个数据库模式中必须是唯一的。

数据库触发器命名约定

以下列表提供了对数据库触发器的命名指导:

- 数据库触发器名称长度不应超过 18 个字符。
- 数据库触发器名称中不应包含空格。
- 数据库触发器名称中不应包含任何数据库保留字。
- 数据库触发器应当命名为 T_*table*，其中 *table* 是表名。例如，USERS 表的数据库触发器名称是 T_USERS1。
- 数据库触发器名称在整个数据库模式中必须是唯一的。

数据库列数据类型注意事项

本部分介绍了可以在创建新表时使用的列数据类型。对各种不同的数据类型的描述使用 DB2 的术语。如果您正在使用不同的数据库，则第 75 页的『数据库间数据类型的差别』描述了这些差异。

BIGINT

这是 64 位的有符号的整数，范围从 -9223372036854775807 到 9223372036854775807。用它与 INTEGER 对比，INTEGER 只是 BIGINT 大小的一半。

INTEGER

这是 32 位的有符号的整数，范围从 -2147483647 到 2147483647。通常，INTEGER 应当是缺省的有限数字数据类型，而不是 BIGINT。除非有非常充分的业务理由要使用 BIGINT，出于性能原因最好使用 INTEGER 作为数字数据类型。BIGINT 数据类型的普通用户是系统生成的键。

我们强烈建议不要使用 SMALLINT 或 SHORT 数据类型，因为这些数据类型映射到非对象 Java 数据类型，而这些非对象数据类型会在一些企业 bean 对象实例化中造成问题。

TIMESTAMP

这是一个由 7 部分组成的值（年、月、日、时、分、秒和微秒），它指定日期和时间，除了时间包含微秒的小数规范外。时间戳记的内部表示法是 10 字节的字符串，每个字节由 2 个压缩的十进制数字构成。前面 4 个字节代表日期，接下来的 3 个字节代表时间，后 3 个字节代表微秒。

CHAR 这是长度为 INTEGER 的固定长度字符串，它的范围可以从 1 到 254 个字符。如果省略了长度规范，则假定长度为 1 个字符。由于 CHAR 是固

定长度的数据库列，因此任何未使用的尾随字符空格都会更改为空格。除非为了性能原因，否则建议不使用 CHAR 数据类型，因为 CHAR 不灵活，且以后长度不能更改。作为简便的规则，如果字符串列的长度小于 64 个字符，且要定期检索或更新，请使用 CHAR 以获得更好的性能。

VARCHAR

这是最长长度为 integer 的可变长度字符串，它的范围可以从 1 到 32672 个字符。但是，与 CHAR 不同（在 CHAR 中，列数据与表存储在一起），VARCHAR 是内部作为数据库页面内的引用指针表示的。因此，VARCHAR 列的长度可以在创建后随时更改。

LONG VARCHAR

这是可变长度的字符串，在同一数据库页面内无法创建 VARCHAR 时可以使用此类型。LONG VARCHAR 与 VARCHAR 非常相似，只是它可以跨越多个数据库页面。请将 LONG VARCHAR 数据类型限制为只在绝对必需的情况下才使用，因为 LONG VARCHAR 对象通常在性能上开销很大。

CLOB 这是另一个可变长度的字符串，在列的长度需要超过 LONG VARCHAR 的 32KB 限制时可以使用此类型。CLOB 对象的长度可以在不必修改数据库配置的情况下达到 1 GB。作为 CLOB 存储的文本数据在不同的系统间移动时可以进行适当的转换。

BLOB 这是可变长度的二进制字符串，它存储数据库中无结构的数据。BLOB 对象可以存储多达 4 GB 的二进制数据。通常应当避免将 BLOB 用作列数据类型，除非绝对必要。在性能方面，BLOB 对象可以认为是任何数据库中开销最大的对象之一。

DECIMAL(20,5)

此数据类型是为多数定点小数（如货币单位）专门定义的类型。对于其它浮点小数则可以使用 FLOAT。

数据库间数据类型的差别

下表列出了 WebSphere Commerce 数据库模式中使用的数据类型，并显示了不同数据库实现的相应数据类型。

JDBC 对象	Windows AIX Solaris Linux **DB2**	Windows AIX Solaris **Oracle®**	400 **DB2**
Hashtable	BLOB()	BLOB	BLOB()
Timestamp	TIMESTAMP	DATE	TIMESTAMP
Integer	INTEGER	INTEGER	INTEGER
BigDecimal	DECIMAL(,)	DECIMAL(,)	DECIMAL(,)
Long	BIGINT	NUMBER	BIGINT
Double	FLOAT	NUMBER	FLOAT
String	CHAR()	VARCHAR2()	GRAPHIC() CCSID 13488
byte[]	CHAR()（对于位数据）	RAW()	CHAR()（对于位数据）
String	VARCHAR()	VARCHAR2()	VARGRAPHIC() CCSID 13488
String	LONG VARCHAR	VARCHAR2()（请参阅表后的注解以获取更多详细信息。）	VARGRAPHIC(4000) ALLOCATE() CCSID 13488
byte[]	LONG VARCHAR（对于位数据）	LONG RAW	VARCHAR(8000) ALLOCATE()（对于位数据）
String	CLOB()	CLOB()	DBCLOB() CCSID 13488

注:

由于 Oracle JDBC 驱动程序处理 LONG 数据类型的信息时，有着不一致的成功率，因此我们建议您在可能时尽量避免使用 LONG 数据类型。此情况下，最常报告的错误是“流已关闭”错误。

如果必须使用此数据类型，则每个数据库表只有一列可以使用 LONG 类型。此外，当构造 select 语句时，不要在 select 语句中将 LONG 列作为第一个

元素也不要作为最后一个元素。在重负荷的情况下，操作时的另一个回避措施是避免将此特定列映射到实体 bean 中的 CMP 字段。而应当使用会话 bean 执行对此列的检索和更新。

第 4 章 访问控制

理解访问控制

WebSphere Commerce 应用程序的访问控制模型有三个主要的概念: 用户、操作和资源。“用户”是使用该系统的人员。“资源”是在应用程序中或由应用程序维护的实体。例如，资源可以是产品、文档或订单。代表人员的用户简要表也是资源。“操作”是用户可以在资源上执行的活动。访问控制是电子交易应用程序的组件，它确定了给定用户是否可以在给定的资源上执行给定的操作。

在 WebSphere Commerce 应用程序中，有两个主要的访问控制级别。访问控制的第一个级别由 WebSphere Application Server 执行。就此而言，WebSphere Commerce 使用 WebSphere Application Server 保护企业 bean 和小服务程序。访问控制的第二个级别是 WebSphere Commerce 的细粒度访问控制系统。

WebSphere Commerce 访问控制框架使用访问控制策略来确定是否允许给定用户在给定的资源上执行给定的操作。该访问控制框架提供了细粒度的访问控制。它与 WebSphere Application Server 提供的访问控制协同工作，但并不代替后者。

WebSphere Application Server 中资源保护的概述

以下 WebSphere Commerce 资源由 WebSphere Application Server 在访问控制下保护:

- 实体 bean
  这些 bean 模拟电子交易应用程序中的对象。它们是可以通过远程客户机访问的分布式对象。
- JSP 模板
  WebSphere Commerce 使用 JSP 模板显示页面。每一 JSP 模板可以包含一个或多个从实体 bean 中检索数据的数据 bean。通过编辑 URL 请求，客户机可以请求 JSP 页面。
- 控制器和视图命令
  通过编辑 URL 请求，客户机可以请求控制器和视图命令。此外，通过使用 JSP 文件名或视图名称（它们注册在 VIEWREG 表中），一个显示页面可以包含指向另一页面的链接。

通常配置 WebSphere Commerce Server 使用以下 Web 路径:

- `/webapp/wcs/stores/servlet/*`
  这是请求到请求小服务程序的路径。

- `/webapp/wcs/stores/*.jsp`
  这是请求到 JSP 小服务程序的路径。

下图显示了在上述 Web 路径配置下，为访问 WebSphere Commerce 资源，这些请求可能通过的路径。

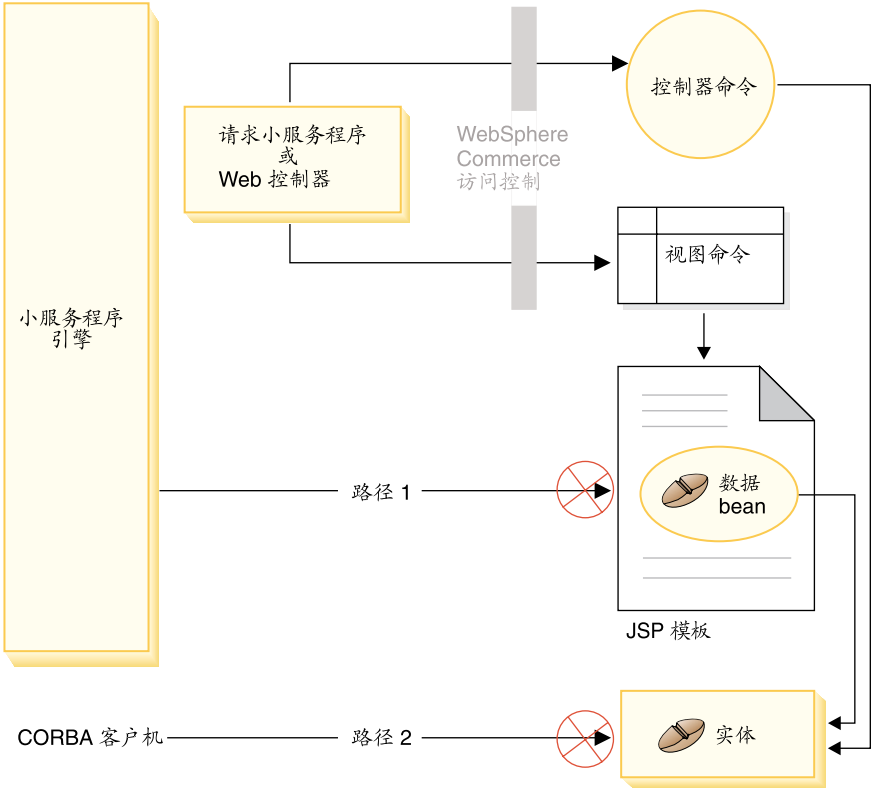


图 19.

所有合法请求应当定向到请求小服务程序，然后该小服务程序将它们定向到 Web 控制器。Web 控制器实现对控制器命令和视图的访问控制。但是，以上所示的 Web 路径使得怀有恶意的用户可以直接访问 JSP 模板（路径 1）和实体 bean（路径 2）。为了防止这些恶意攻击得逞，必须在运行时将它们拒绝。

通过使用以下方法之一，可以防止直接访问 JSP 模板和实体 bean:

WebSphere Application Server 安全性

WebSphere Application Server 提供安全性功能。使用这种方法，所有企业 bean 方法和 JSP 模板被配置为仅由“系统身份”调用。要访问这些 WebSphere Commerce 资源，在将 URL 请求传递到 Web 控制器之前，

必须将其路由到设置“系统身份”到当前线程的请求小服务程序。Web 控制器则在将请求传递到相应的控制器命令或视图之前，确保调用者具有所需权限。WebSphere Application Server 安全性组件将拒绝任何直接访问 JSP 模板和实体 bean 的企图（即：不使用 Web 控制器）。

关于配置 WebSphere Application Server 以保护 WebSphere Commerce 资源的信息，请参阅《*WebSphere Commerce 安装指南*》。关于 WebSphere Application Server 中安全性的信息，请参阅 WebSphere Application Server 文档中的“系统管理”主题。

有关为定制企业 bean 中的方法配置 WebSphere Application Server 安全性的信息，请参阅第 332 页的『将新的企业 bean 组装到企业应用程序中』和第 337 页的『将修改的企业 bean 组装到企业应用程序中』。

防火墙保护

当 WebSphere Commerce Server 在防火墙后运行时，因特网客户机不能直接访问实体 bean。包含在页面中的数据 bean 使用这种方法提供对 JSP 模板的保护。数据 bean 由数据 bean 管理器激活。数据 bean 管理器检测 JSP 模板是否由视图命令转发。如果不是由视图命令转发，则抛出异常并拒绝对 JSP 模板的请求。

WebSphere Commerce 访问控制策略简介

WebSphere Commerce 访问控制模型基于访问控制策略的执行。访问控制策略允许从业务逻辑代码将访问控制规则具体化，从而不必将访问控制语句硬编码到代码中。例如，您无需包含与以下类似的代码:

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

访问控制策略由访问控制策略管理器强制执行。一般情况下，当用户试图访问受保护的资源时，访问控制策略管理器首先确定对受保护的资源应用何种访问控制策略，然后它根据适用的访问控制策略确定是否允许此用户访问请求的资源。

访问控制策略是四元组策略，它存储在 ACPOLICY 表中。每个访问控制策略采用以下格式:

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

四元组访问控制策略中的元素指定，只要属于指定用户组的用户对于正在考虑的资源满足关系或关系组中指定的条件，则允许该用户对属于指定资源组的资源执行指定操作组中的操作。例如，`[AllUsers, UpdateDoc, doc, creator]` 指定如果用户是文档的创建者，则该用户就可更新此文档。

用户组是 MBRGRP 数据库表中定义的特定类型的成员组。用户组必须与成员组类型 -2 相关联。值 -2 代表访问组，它是在 MBRGRPTYPE 表中定义的。用户组与成员组类型之间的关联存储在 MBRGRPUSG 表中。

用户加入某一特定用户组中的成员资格可以显式声明或隐式声明。如果 MBRGRPMBR 表声明该用户属于特定成员组，则发生显式指定。如果用户满足 MBRGRPCOND 表中声明的情况（例如，所有履行“产品经理”角色的用户），则发生隐式指定。也可能有组合情况（例如，履行“产品经理”角色且担任此角色至少已经 6 个月的所有用户）或显式排除。

在用户组中包含用户的大部分情况是基于该用户履行特定角色。例如，有可能有一访问控制策略允许所有履行“产品经理”角色的用户执行产品目录管理操作。在此情况下，MBRROLE 表中指定为“产品经理”角色的任何用户都会隐式包含在该用户组中。

关于成员组子系统的更多详细信息，请参阅 WebSphere Commerce 联机帮助。

ActionGroup 元素来自 ACACTGRP 表。操作组是指显式指定的操作组。操作的列表存储在 ACACTION 表中，每个操作与其操作组（或组）的关系存储在 ACACTACTGP 表中。举例来说，操作组有“OrderWriteCommands”操作组。该操作组包括以下用于更新订单的操作:

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

资源组是将特定类型的资源组合在一起的机制。可以采用以下两种方法之一来指定资源组中资源的成员资格。

- 使用 ACRESGRP 表中的 conditions 列
- 使用 ACRESGPRES 表

在大多数情况下，要将资源关联到资源组，使用 ACRESGPRES 表就足够了。在使用此方法的情况下，将使用资源的 Java 类名在 ACRESCGRY 表中定义资源。然后，使用 ACRESGPRES 关联表将这些资源与适当的资源组（ACRESGRP 表）关联在一起。有时候仅用 Java 类名不足以定义资源组的成员（例如，您需要根据资源的属性进一步限制此类的对象），在这种情况下可以使用 ACRESGRP 表的条件列来定义整个资源组。注意，为了根据属性执行资源的分组，此资源还必须实现 Groupable 接口。

下图显示了资源分组规范的示例。在此示例中，资源组 10023 包含 ACRESGPRES 表中所有与其相关的资源。资源组 10070 是用 ACRESGRP 表中的 conditions 字段列定义的。该资源组包含订单远程接口的实例，其状态也是 "Z"（指定共享的需求列表）。

注: 有关 ACRESGRP 表 "条件" 列的 XML 信息的详细信息，请参阅《*WebSphere Commerce 访问控制指南*》。

ACRESGRP

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

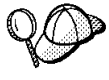
ACRESGRPES

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

ACRESCGRY

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

图 20.



ACACTGRP、ACRESGRP 和 ACRELGRP 表的 MEMBER_ID 列应具有值 -2001（根组织）。

（可选）访问控制策略可以包含 Relationship 或 RelationshipGroup 元素作为它的第四个元素。

如果您的访问控制策略使用了 Relationship 元素，则这来自于 ACRELATION 表。另一方面，如果它包含 RelationshipGroup 元素，则它来自 ACRELGRP 表。注意，并不是必须包含这两者的，但是如果您包含了其中一个，就不能包含另一个。ACRELGRP 表中的 RelationshipGroup 规范的优先级高于 ACRELATION 表中的 Relationship 信息。

ACRELATION 表指定了用户和资源之间存在的关系类型。关系类型的一些示例包含“创建者”、“提交者”和“所有者”。使用关系元素的一个示例就是使用它来确保订单的创建者始终可以更新该订单。

ACRELGRP 表指定了可以与特定资源关联的关系组的类型。关系组是一个或多个关系链的分组。关系链是一个以上关系的系列。关系组的一个示例是，用户必须是资源的创建者同时还必须属于此资源中引用的买方组织实体。

关系组（或关系）规范是访问控制策略的可选部分。如果您已创建了您自己的命令而这些命令没有限制给某个角色，则通常会使用到它。在这些情况下，您可能希望强制用户和资源之间的关系。如果要将命令限制给某些角色，则通常是通过访问控制策略的 UserGroup 元素来完成的，而不是使用 Ralationship 元素来完成的。

与访问控制策略相关的另一个重要概念是访问控制策略*所有者*。访问控制策略所有者是拥有访问控制策略的组织实体。知道访问控制策略的所有者非常重要，因为访问控制策略只可应用到访问控制策略所有者拥有的资源。

对于每个正在审议的资源，访问控制策略管理器将应用所有者组织实体或其在成员层次结构中的先辈组织实体所拥有的访问控制策略，直到发现有策略授予了许可权，或直到已检查所有策略且它们都未授予许可权。

请看下图，它显示了成员层次结构。

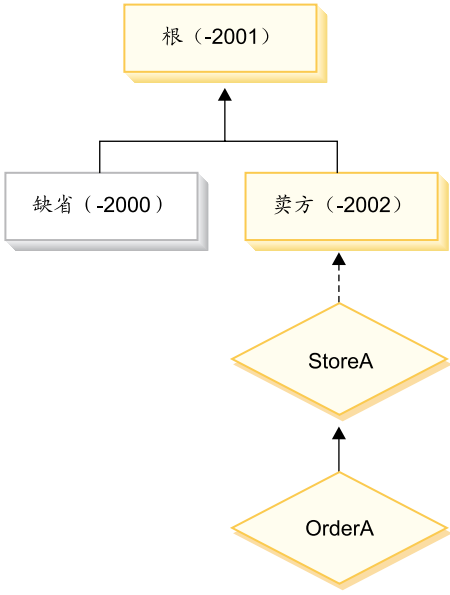


图 21.

对于资源“OrderA”，可以应用“卖方”或“根”组织所拥有的任何访问控制策略。如果访问控制策略管理器发现这些组织所有的一个策略（根据访问控制策略中的四个元素）授予用户许可权，则它立刻停止搜索访问控制策略。然而，如果它发现这些组织所拥有的任何访问控制策略都没有授予用户在受保护的资源上执行操作的权力，则访问被拒绝。

关系组

关系组允许您指定多个关系。一个关系可以是用户和正在考虑的资源之间的直接关系，也可以是关系的链（它将用户间接地与资源联系在一起）。

注： 对于与关系组有关的以下章节，认识到只有组织 RootOrganization、DefaultOrganization 和 SellerOrganization 可以在 WebSphere Commerce 专业版中使用是很重要的。有关其它组织的示例仅适用于 WebSphere Commerce 商务版。

关系和关系组的比较: 访问控制策略可以指定用户必须对正在访问的资源满足特定的关系，或者它们可以指定用户必须满足关系组中指定的条件。

在大多数情况下，指定一个关系应该满足应用程序的访问控制需求。但如果策略是您必须指定的关系不是用户与资源之间的直接关系，而实际上是用户与资源之间的一系列关系，则您必须使用关系组。

例如，如果您必须指定用户和买方组织之间的关联（此处，关系要求用户正在为该组织扮演特定角色或用户是买方组织的成员），则您必须使用关系组和关系链。

如果您仅需要加强用户和正在考虑的资源之间的直接关联，则您可以使用简单关系。例如，如果您需要强制用户必须是资源的创建者。

如果您组合了多个简单关系（例如用户必须是创建者或提交者），则这就成了关系链，且您必须使用关系组。当使用 WebSphere Commerce 专业版或 WebSphere Commerce 商务版时，可能会发生这种简单关系的组合。

关于关系组的一般信息: 关系链是一个以上关系的系列。关系链的长度由其包含的关系数量确定。这可以通过检查关系链的 XML 表示中 `<parameter name="aName" value="aValue" />` 元素的数量来确定。

只有最后的 `<parameter name="Relationship" value="aValue" />` 元素必须由资源的 fulfills() 方法来处理。其它的由访问控制策略管理器内部处理。

当关系链的长度为 2 时，则首个 `<parameter name="aName" value="aValue" />` 元素在用户和组织实体之间。最后一个 `<parameter name="aName" value="aValue" />` 元素在组织实体和资源之间。

如果您需要定义关系组，则您必须通过在 XML 文件中定义关系组信息来完成。您可以修改 `defaultAccessControlPolicies.xml` 文件，或创建自己的 XML 文件。有关创建这些基于 XML 的信息的更多信息，请参阅《*WebSphere Commerce 访问控制指南*》。

以下章节显示了不同类型的关系组的示例。

由单一关系链组成的关系组: Business 作为访问控制策略的一部分，您可能需要强制用户必须属于组织实体（该实体是资源的 BuyingOrganizationalEntity）。这就需要创建关系组，它由一个长度为 2 的关系链组成。关系链的长度之所以为“2”，是因为它包含两个独立的关系。第一个关系存在于用户与其父组织实体之间。用户是该关系中的“子”。对于第二个关系，访问控制策略管理器检查父组织实体是否实现了与资源的 BuyingOrganizationalEntity 关系。换句话说，如果它是资源的买方组织实体，则返回“true”。

以下 XML 片段采自于 `defaultAccessControlPolicies.xml` 文件，它说明了如何定义该类型的关系组:

```
<RelationGroup Name="MemberOf->BuyerOrganizationalEntity"
     OwnerID="RootOrganization">
 <RelationCondition><![CDATA[
   <profile>
```

```
    <openCondition name="RELATIONSHIP_CHAIN">
     <parameter name="HIERARCHY" value="child"/>
     <parameter name="RELATIONSHIP" value="BuyingOrganizationalEntity"/>
    </openCondition>
   </profile>
 ]]></RelationCondition>
</RelationGroup>
```

Business 另一个示例将强制用户必须具有组织实体（该实体是正在考虑的资源的买方组织实体）的“客户代表”角色。同样，这使用一个关系组，它由一个长度为 2 的关系链组成。该链的首个部分将查找所有的组织实体，以找出该用户对哪个实体具有“客户代表”角色。然后，对于这组组织实体，访问控制策略管理器检查它们中是否至少有一个组织实体实现了与资源的 BuyingOrganizationalEntity 关系。换句话说，如果它们中的一个实体是资源的买方组织实体，则返回“true”。

以下 XML 片段采自于 defaultAccessControlPolicies.xml 文件，它说明了如何定义该类型的关系组:

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

由多个关系链组成的关系组: 可能要编写一个关系组，以便它包含多个关系链。当执行此操作时，您必须指定用户是必须满足所有的关系链（这意味着它是 *AND* 方案），还是必须满足至少一个关系链（这意味着它是 *OR* 方案）。

Business 要演示此类型的关系，以下 XML 片段用于强制用户必须是资源的创建者，且用户还必须属于资源中指定的 BuyingOrganizationalEntity。第一个链（它指定用户必须是资源的创建者）的长度为 1。第二个链（它指定用户必须属于资源中指定的 BuyingOrganizationalEntity）的长度为 2。

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

```
    <parameter name="RELATIONSHIP" value="BuyingOrganizationalEntity"/>
   </openCondition>
  </andListCondition>
 </profile>
]]></RelationCondition>
</RelationGroup>
```

如果您需要用户满足两个关系链之一（而不使用 *AND* 方案），则应该将 <andListCondition> 标记更改为 <orListCondition> 标记。

Professional Business 为了演示可以在 WebSphere Commerce 专业版（以及 WebSphere Commerce 商务版）中使用的关系组，请考虑这样一个关系组，它用于强制用户必须是资源的创建者或提交者。这显示在以下 XML 片段中。

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

访问控制类型

有两种类型的访问控制，它们都是基于策略的：命令级别的访问控制和资源级别的访问控制。

命令级别（也称为“基于角色的”）访问控制使用的策略类型很广泛。您可以指定某一特定角色的所有用户都可以执行某些类型的命令。例如，您可以指定具有“客户代表”角色的用户可以在 AccountRepresentativesCmdResourceGroup 资源组中执行任何命令。或者，如下图中所描绘的，另一个示例策略是指定所有商店管理员都可以对 StoreAdminCmdResourceGrp 指定的任何资源执行在 ExecuteCommandAction 组中指定的任何操作。

注：有关 MBRGRPCOND 表“条件”列的 XML 信息是在您使用管理控制台设置访问组时生成的。有关使用管理控制台设置访问组的信息，请参阅 WebSphere Commerce 联机帮助。

ACPOLICY

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

MBRGRP

MbrGrp_Id	MbrGrpName
-8	StoreAdministrators

MBRGRPCOND

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

ACACTGRP

AcActGrp_Id	GroupName
10052	ExecuteCommandActionGroup

ACRESGRP

AcResGrp_Id	GrpName
10018	StoreAdminCmdResourceGroup

图 22.

命令级别的访问控制策略总是将 `ExecuteCommandActionGroup` 作为控制器命令的操作组。对于视图，该资源组总是 `ViewCommandResourceGroup`。

所有控制器命令都必须由命令级别的访问控制保护。此外，任何可以直接调用，或可以由其它命令通过重定向来启动的视图（而不是通过转发到视图来启动），也都必须由命令级别的访问控制保护。

命令级别的访问控制不考虑命令对其作用的资源。它只确定是否允许用户执行特定的命令。如果允许用户执行该命令，则可以应用后续的资源级别访问控制策略，以确定用户是否可以访问正在审议的资源。

试想商店管理员尝试执行管理任务时的情况。第一个级别的访问控制检查将确定是否允许此用户执行特定的商店管理命令。一旦确定了确实允许该用户执行此操作（因为允许商店管理员执行 storeAdminCmds 组中的命令），就可能会调用资源级别的访问控制。此策略可能声称仅允许商店管理员执行组织（对于该组织，该用户是商店管理员）所拥有的商店的管理任务。

总之，在命令级别的访问控制中，“资源”是命令本身，而“操作”仅仅是执行命令（换言之，即实例化命令对象）。访问控制检查确定是否允许用户执行命令。相比而言，在资源级别的访问控制中，“资源”是命令或 bean 访问的任何可保护的资源，而“操作”是命令本身。

访问控制交互

本部分给出交互图，显示访问控制如何在 WebSphere Commerce 访问控制策略框架中作用。

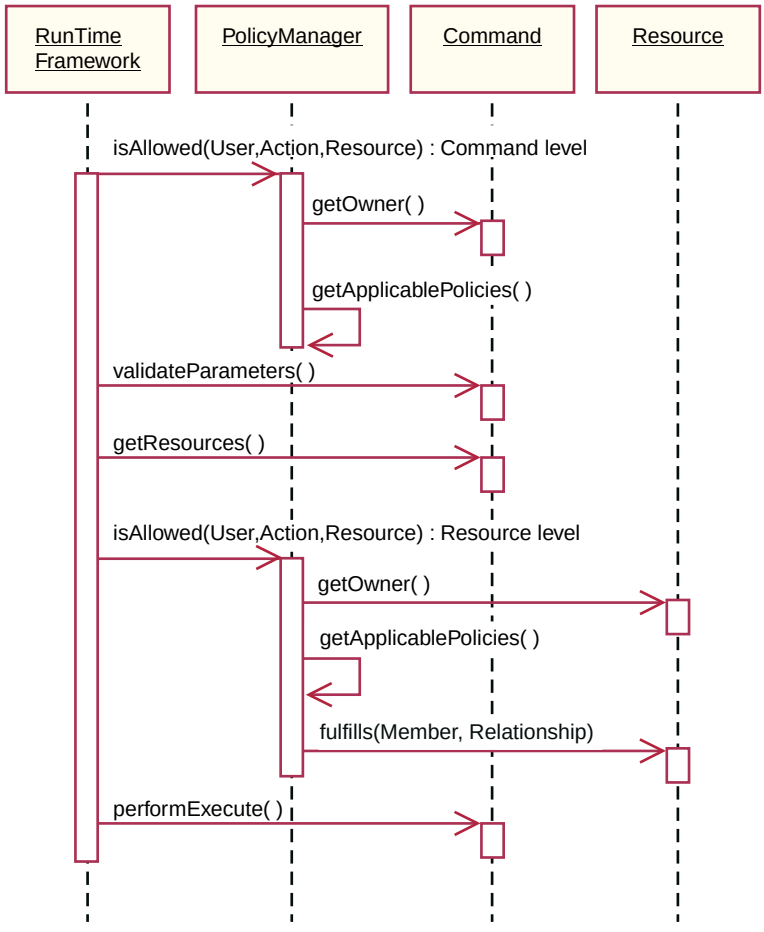


图 23.

上图显示了访问控制*策略管理器*执行的操作。访问控制策略管理器是访问控制组件，它确定是否允许当前用户对指定的资源执行指定的操作。它通过搜索资源所有者及其上级组织所拥有的策略来对此进行确定。如果至少有一个策略授予访问权限，则授予许可权。

以下列表描述了上面交互图中的操作。它们按照从图顶部至底部的顺序进行排序。

1. `isAllowed()`
   运行时组件确定用户对控制器命令或视图是否具有命令级别的访问权限。
2. `getOwner()`
   访问控制策略管理器确定命令级资源的所有者。缺省实现返回命令上下文中

的商店（storeId）所有者的成员标识（memberId）。如果在命令上下文中没有商店标识，则将返回根组织（-2001）。

3. getApplicablePolicies()
   访问控制策略管理器根据指定的用户、操作和资源查找并处理可应用的策略。
4. validateParameters()
   初始参数检查和解析。
5. getResources()
   返回一个访问向量，它是“资源 — 操作”对向量。

   如果不作任何返回，将不执行资源级别的访问控制检查。如果有应受保护的资源，则应当返回访问向量（由“资源 — 操作”对组成）。

   每个资源是一个可保护的对象的实例（实现 com.ibm.commerce.security.Protectable 接口的对象）。在很多情况下，资源就是访问 bean。

   访问 bean 可能不实现 com.ibm.commerce.security.Protectable 接口，但根据在第 95 页的『在企业 bean 中实现访问控制』中包含的信息，只要相应的企业 bean 受到保护，访问控制检查就仍可能发生。

   *操作*是表示要对资源执行的操作的字符串。在大多数情况下，操作是命令的接口名称。
6. isAllowed()
   运行时组件确定用户对 getResources() 指定的所有“资源 — 操作”对是否都具有资源级别的访问权限。
7. getOwner()
   资源返回其所有者的 memberId。它确定哪些策略适用。仅资源所有者及其上级组织拥有的策略适用。
8. getApplicablePolicies()
   访问控制策略管理器搜索可应用的策略，然后应用它们。如果每个“资源 — 操作”对至少找到一个策略授予了用户访问资源的许可权，则授予访问权限，否则访问被拒绝。
9. fulfills()
   如果可应用的策略具有指定的关系组，则将对资源执行检查以查看成员对于该资源是否满足指定的关系。
10. performExecute()
    命令的业务逻辑。

Protectable 接口

让资源受 WebSphere Commerce 访问控制策略保护的关键因素在于该资源必须实现 com.ibm.commerce.security.Protectable 接口。该接口通常与企业 bean 和数据 bean 一起使用，但是只有那些需要保护的特定 bean 才需要实现此接口。

利用 Protectable 接口，资源必须提供两个主要的方法：getOwner() 和 fulfills(Long member, String relationship)。

访问控制策略为组织或组织实体所拥有。getOwner 方法返回可保护资源所有者的 memberId。当访问控制策略管理器确定资源的所有者后，它还获取该所有者在成员层次结构中的每个上级所有者的 memberId。然后应用所有属于来自原始 getOwner 请求的所有者的访问控制策略以及所有属于该所有者的任何上级所有者的访问控制策略。

适用于指定的所有者的访问控制策略，以及适用于该所有者在成员资格层次结构中的任何更高级别上级所有者的访问控制策略，都会得到应用。

仅当给定的成员对该资源满足需要的关系时，fulfills 方法才返回 “true”。通常，成员是单一用户，然而也可以是一个组织。如果您正在使用访问控制策略中的关系组，则它将是一个组织。

Groupable 接口

访问控制策略的应用是特定于一组资源的。资源分组可以根据诸如类名、订单状态或 storeId 值等属性进行。

如果为了应用访问控制策略的目的而要根据属性（而不是类名）来将资源分组，则它必须实现 com.ibm.commerce.grouping.Groupable 接口。

以下代码片段演示了 Groupable 接口:

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

例如，要实现仅适用于处于 “未决” 状态（status = P（未决））的订单的策略，“订单” 实体 bean 的远程接口实现 Groupable 接口，而 attributeName 的值设置为 “status”。

使用 Groupable 接口的情况是很少的。

查找关于访问控制的更多信息

有关 WebSphere Commerce 访问控制模型的更多信息，请参阅《*WebSphere Commerce 访问控制指南*》。该指南提供了访问控制的详细概述并描述了如何使用管理控制台创建或修改策略、操作组和资源组。

实现访问控制

本部分描述如何用定制代码实现访问控制。

确定可保护资源

通常，企业 bean 和数据 bean 是可能希望保护的资源。然而，并不是所有企业 bean 和数据 bean 都应当保护。在现有的 WebSphere Commerce 应用程序中，需要保护的资源已经实现了可保护的接口。当创建新的企业 bean 和数据 bean 时，会出现保护什么的问题。决定保护哪些资源取决于您的应用程序。

如果命令在 `getResources` 方法中返回企业 bean，则该企业 bean 必须是受到保护，因为访问控制策略管理器将对该企业 bean 调用 getOwner 方法。如果在相应的资源级别访问控制策略中指定了某种关系，则也将调用 `fulfills` 方法。

如果要为您自己的所有企业 bean 和数据 bean 实现 Protectable 接口（从而将资源置于保护状态下），则应用程序会需要很多策略。随着策略数量的增加，性能可能会变差，而策略管理将变得更为困难。

主资源和从属资源间有理论上的差别。*主资源*可以独立存在。*从属资源*仅当其相关主资源存在时才存在。例如，在 WebSphere Commerce 应用程序代码外部，Order 实体 bean 是可保护资源，但 OrderItem 实体 bean 不是可保护资源。原因是 OrderItem 的存在取决于 Order，即 Order 主资源，OrderItem 是从属资源。如果用户应有对 Order 的访问权限，他同样应具有对订单中商品的访问权限。

同样，User 实体 bean 是可保护资源，但 Address 实体 bean 不是。在此情况下，地址的存在取决于用户，因此任何对用户具有访问权限的，也应对地址有访问权限。

主资源应受保护，但通常从属资源不需要保护。如果允许用户访问主资源，则缺省情况下，也应允许该用户访问其从属资源。

在企业 bean 中实现访问控制

如果创建需要受访问控制策略保护的新企业 bean，则必须执行以下操作:

1. 创建新的企业 bean，确保它从 `com.ibm.commerce.base.objects.ECEntityBean` 扩展。

2. 请确保 bean 的远程接口扩展了 com.ibm.commerce.security.Protectable 接口。
3. 如果与 bean 交互的资源是根据属性（而不是资源的 Java 类名）来分组的，则 bean 的远程接口还必须扩展 com.ibm.commerce.grouping.Groupable 接口。
4. 企业 bean 类包含以下方法的缺省实现:
   - getOwner
   - fulfills
   - getGroupingAttributeValue

   覆盖您需要的任何方法。至少，必须覆盖 getOwner 方法。
   这些方法的缺省实现在以下代码片段中显示。

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

   以下是这些方法基于 OrderBean bean 的样本实现:

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

```
                com.ibm.commerce.user.objects.UserAccessBean();
            creator.setInitKey_MemberId(getMemberId().toString());
            com.ibm.commerce.user.objects.UserAccessBean ab = new
                com.ibm.commerce.user.objects.UserAccessBean();
            ab.setInitKey_MemberId(member.toString());
            if (ab.getParentMemberId().equals(creator.getParentMemberId()))
     return true;
          }
      return false;
}
*****************************************************************************
*****************************************************************************
public Object getGroupingAttributeValue(String attributeName,
   GroupingContext context) throws Exception {
      if (attributeName.equalsIgnoreCase("Status"))
         return getStatus();
      return null;
   }
*****************************************************************************
```

5. 创建（或重新创建）企业 bean 的访问 bean 和生成的代码。

在数据 bean 中实现访问控制

如果要保护数据 bean，它可以由访问控制策略直接或间接保护。如果直接保护数据 bean，则存在适用于该特定数据 bean 的访问控制策略。如果间接保护数据 bean，它将保护授权给另一个存在访问控制策略的数据 bean。

如果创建由访问控制策略直接保护的新数据 bean，则该数据 bean 执行以下操作:

1. 实现 com.ibm.commerce.security.Protectable 接口。这样，bean 必须提供 getOwner() 和 fulfills(Long member, String relationship) 方法的实现。这些应该在 bean 的远程接口上实现。

   当数据 bean 实现 Protectable 接口时，数据 bean 管理器调用 isAllowed 方法确定根据当前的访问控制策略，用户是否具有适当的访问控制特权。以下代码片段描述了 isAllowed 方法:

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

2. 如果与 bean 交互的资源是按由除资源 Java 类名之外的属性分组的，则 bean 必须实现 com.ibm.commerce.grouping.Groupable 接口。
3. 实现 com.ibm.commerce.security.Delegator 接口。该接口由以下代码片段描述:

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

   注: 为了直接受到保护，getDelegate 方法应该返回数据 bean 本身（即: 为了访问控制的目的，数据 bean 授权给自己）。

哪些数据 bean 应当直接保护与哪些数据 bean 应当间接保护，这两者之间的差别与主资源与从属资源之间的差别相似。如果数据 bean 对象可以独立存在，则它可以直接保护。如果数据 bean 的存在取决于另一个数据 bean 的存在，则它应当授权另一数据 bean 进行保护。

举例来说，直接保护的数据 bean 有 Order 数据 bean。间接保护的数据 bean 有 OrderItem 数据 bean。

如果创建要由访问控制策略间接保护的新数据 bean，则该数据 bean 必须执行以下操作:

1. 实现 com.ibm.commerce.security.Delegator 接口。该接口由以下代码片段描述:

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

   注: getDelegate 返回的数据 bean 必须实现 Protectable 接口。

如果数据 bean 不实现 Delegator 接口，则它在不受访问控制策略保护的情况下填充。

在控制器命令中实现访问控制策略

创建新控制器命令时，新命令的实现类应扩展 com.ibm.commerce.commands.ControllerCommandImpl 类，并且它的接口应扩展 com.ibm.commerce.command.ControllerCommand 接口。

对于控制器命令的命令级别策略，命令的接口名称被指定为资源。为了让资源受到保护，它必须实现 Protectable 接口。根据 WebSphere Commerce 编程模型，这完成的方法是让命令接口从 com.ibm.commerce.command.ControllerCommand 接口扩展，并让命令实现从 com.ibm.commerce.commands.ControllerCommandImpl 扩展。ControllerCommand 接口扩展 com.ibm.commerce.command.AccCommand 接口，后者接着扩展 Protectable。AccCommand 接口是命令为受到命令级别访问控制的保护而应当实现的最低接口。

如果命令访问应受保护的资源，请创建了一个类型为 AccessVector 的私有实例变量以容纳资源。然后覆盖 getResources 方法，由于此方法的缺省实现返回一空值，因此不会发生资源检查。

在新的 getResources 方法中，您应当返回一个资源数组或命令可以对其作用的“资源 — 操作”对数组。当未显式指定操作，该操作缺省为正在执行的命令的接口名称。

此外，我们建议让此方法决定它是否必须实例化资源或是否可以使用现有的容纳资源引用的实例变量。检查资源对象是否已经存在，这有助于提高系统性能。然后，如果需要，您可以使用新控制器命令的 performExecute 方法中相同的 getResources 方法。

以下是 getResources 方法的示例:

```
private AccessVector resources = null;

public AccessVector getResources() throws ECException {

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

例如，考虑 OrderItemUpdate 命令。此命令的 getResources 方法返回“订单”和“用户”可保护对象。由于未指定操作，因此它缺省为 OrderItemUpdate 命令的接口。

getResources 方法可能会返回多个资源。当这种情况发生时，如果要执行操作，则必须找到让用户有权访问所有指定资源的策略。如果用户对三个资源中的两个有访问权限，该操作可能不再进行（需要具有全部三个资源的访问权限）。

如果需要在控制器命令中执行附加参数检查或参数解析，您可以使用 validateParameters() 方法。这是可选的。

附加资源级别检查

在调用控制器命令的 getResources 方法时，并不是始终可能确定所有需要保护的资源。

如果必要，任务命令还可以实现 getResources 方法以返回资源的列表（命令可以在这些资源上执行）。

调用资源级别检查的另一种方法是使用 checkIsAllowed(Object resource, String action) 方法直接调用访问控制策略管理器。此方法对于从 com.ibm.commerce.command.AbstractECTargetableCommand 类扩展的任何类都是可用的。例如，以下类从 AbstractECTargetableCommand 类扩展:

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

checkIsAllowed 方法对于扩展 com.ibm.commerce.command.AbstractECCommand 类的类同样可用。例如，以下类从 AbstractECCommand 类扩展:

- com.ibm.commerce.command.TaskCommandImpl

以下显示 checkIsAllowed 方法的特征符:

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

如果当前用户未经允许对指定的资源执行指定的操作，则此方法会抛出 ECApplicationException。如果授予了访问权限，则此方法只是简单地返回。

“create”命令的访问控制

由于在命令中 getResources 方法在 performExecute 方法之前调用，因此必须对尚未创建的资源采用不同的方法进行访问控制。例如，如果您有 WidgetAddCmd，getResources 方法将无法返回要创建的资源。在此情况下，getResources 方法应返回资源的创建者。例如，命令由命令工厂创建，订单在商店内创建，用户在组织内创建。

命令级别访问控制的缺省实现

对于命令级别的访问控制，getOwner() 方法的缺省实现是返回商店所有者（如果指定了 storeId）的 memberId。如果没有指定 storeId，则将返回根组织的 memberId（memberId = -2001）。

getResources() 方法的缺省实现返回 null。

validateParameters() 的缺省实现不执行任何操作。

在视图中实现访问控制策略

视图的资源级别访问控制由数据 bean 管理器执行。在以下情况下，调用数据 bean 管理器:

1. 当 JSP 模板包含 <useBean> 标记并且数据 bean 不在属性列表中时。
2. 当 JSP 模板包含以下激活方法时:

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

注: 任何要受到（直接或间接）保护的数据 bean 都必须实现 Delegator 接口。任何要受到直接保护的数据 bean 将授权给自己，所以还必须实现 Protectable 接口。受到间接保护的数据 bean 应该授权给实现 Protectable 接口的数据 bean。

虽然我们不推荐这样做，但在以下情况下还是会绕过访问控制检查:

1. 如果 JSP 模板直接调用访问 bean，而不是使用数据 bean。
2. 如果 JSP 模板直接调用数据 bean 的 populate() 方法。

如果要将控制器命令的结果转发给视图（使用 ForwardViewCommand），则不对视图执行命令级别的访问控制。而且如果控制器命令将填充的数据 bean（即在视图中使用的数据 bean）放在响应属性的属性列表上，然后转发给视图，则 JSP 模板可以不通过数据 bean 管理器而访问数据。这不要求在 JSP 模板中使用 `<useBean>` 标记。这可作为使 JSP 模板更有效的方法，因为它通过控制器命令，可绕过对已授予用户访问权限的资源（数据 bean）的任何冗余资源级别访问控制检查。

第 5 章 错误处理和消息

命令错误处理

WebSphere Commerce 使用良好定义的命令错误处理框架，在定制代码下使用该框架非常简单。在设计上，框架能够以支持多文化商店的方式处理错误。以下章节描述了命令可能抛出的异常的类型，如何处理异常，如何存储和使用消息文本，如何记录异常以及如何在您自己的命令中使用提供的框架。

异常类型

命令可能抛出以下异常之一:

ECApplicationException

如果错误与用户相关，则抛出此异常。例如，如果用户输入无效参数，则抛出 ECApplicationException。抛出此异常时，Web 控制器并不重试此命令，即使它指定为可重试命令。

ECSystemException

如果检测到运行时异常或 WebSphere Commerce 配置错误，则抛出此异常。此类型异常的示例包括空指针异常和事务回滚异常。当抛出此类型的异常时，如果此命令可重试并且异常是由于数据库死锁或数据库回滚造成的，则 Web 控制器将重试此命令。

以上所列的两个异常都是从 ECException 类扩展的类，ECException 类可以在 com.ibm.commerce.exception 数据包中找到。

要抛出这些异常之一，必须指定以下信息:

- 错误视图名称
  Web 控制器在 VIEWREG 表中查找此名称。
- ECMessage 对象
  该值对应于属性文件包含的消息文本。
- 错误参数
  这些“名称值”对用于将信息替代到错误消息中。例如，消息可以包含一个参数，以保留抛出异常的方法的名称。此参数在抛出异常时设置，以后记录错误消息时，该日志文件会包含实际的方法名称。
- 错误数据
  这些数据是可选属性，它们可以通过错误数据 bean 用于 JSP 模板。

异常处理与日志系统密切相关。当抛出异常时，它将自动记录下来。

错误消息属性文件

为了简化错误消息的维护以及支持多语言商店，错误消息的文本存储在属性文件中。WebSphere Commerce 消息文本存储在 ecServerMessages_*XX_XX*.properties 文件中，其中 *_XX_XX* 是语言环境指示符（例如 _en_US）。

命令上下文返回一标识，以表示客户机所使用的语言。当需要消息时，Web 控制器根据语言标识确定使用哪个属性文件。

ecServerMessages*XX_XX*.properties 文件中定义了两种类型的消息：用户消息和系统消息。在顾客的浏览器中对顾客显示用户消息。系统消息和用户消息都自动捕获到消息日志中。

当抛出一个错误时，所需参数之一是消息对象。对于 ECSystemException，消息对象必须包含两个键，一个用于系统消息，另一个用于用户消息。对于 ECApplicationException，消息对象包含用户消息的键（未使用系统消息）。

所有系统消息都是预定义的。您不能创建自己的系统消息。因此，如果定制的代码抛出 ECSystemException，其必须为预定义的系统消息之一指定一个消息键。可以创建定制的用户消息。新的用户消息必须存储在单独的属性文件中。

异常处理流程

下图显示了捕获到异常时的信息流程。随后是每一步的描述。

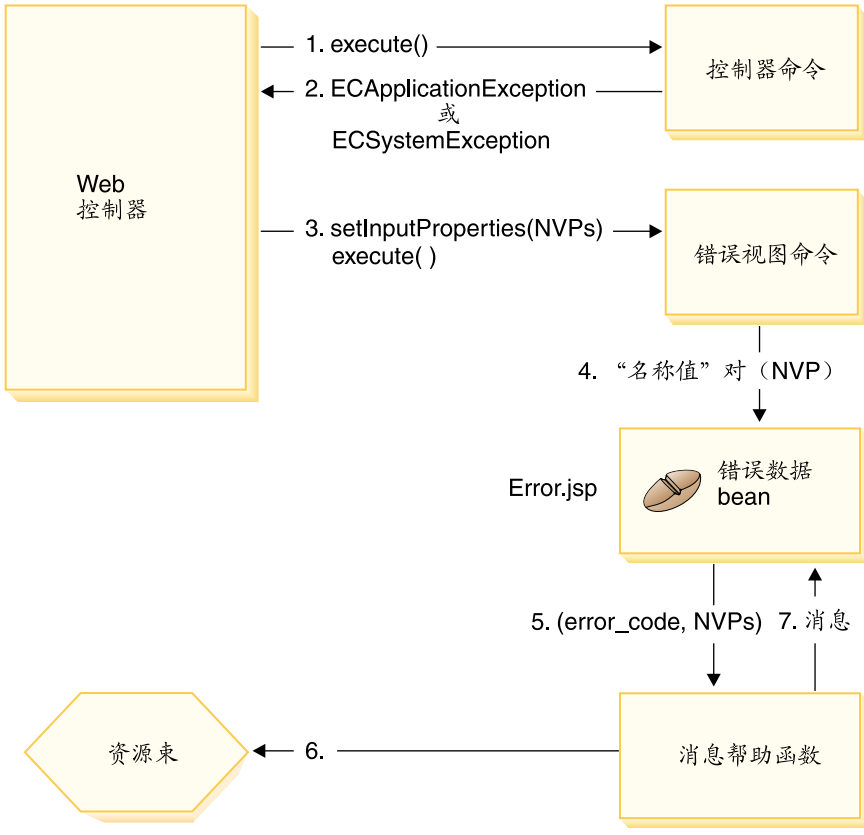


图 24.

1. Web 控制器调用控制器命令。
2. 此命令抛出 Web 控制器捕获到的异常。可以是 ECApplicationException 或 ECSystemException。异常对象包含以下信息:
   - 错误视图名称
   - ECMessage 对象
   - 错误参数
   - （可选）错误数据
3. Web 控制器从 VIEWREG 表中确定错误视图名称并调用指定的错误视图命令。调用此命令时，Web 控制器从 ECException 对象中编辑一组属性，并使用视图命令的 setInputProperties 方法将它设置为视图命令。
4. 视图命令调用错误 JSP 模板（这种情况下为 Error.jsp），并将“名称值”对传递到 JSP 模板。
5. ErrorDataBean 将错误参数传递到消息帮助函数对象。

6. 消息帮助函数对象从相应的属性文件中获取必需的消息（使用消息对象和错误参数）。
7. 错误数据 bean 将消息返回到 JSP 模板。

定制代码中的异常处理

创建新命令时，包含正确的异常处理很重要。通过指定捕获异常时所需信息，您可以利用 WebSphere Commerce 中提供的错误处理和消息传递框架。

编写自己的异常处理逻辑包括以下步骤:

1. 在命令中捕获需要特殊处理的异常。
2. 根据所捕获异常的类型，构造 ECApplicationException 或 ECSystemException。
3. 如果 ECApplicationException 使用新的消息，请在新的属性文件中定义该消息。

捕获和构造异常

要说明前两步，以下代码片段显示了在命令中捕获系统异常的示例:

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

上述的 _ERR_FINDER_EXCEPTION ECMessage 对象如下定义:

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

_ERR_FINDER_EXCEPTION 消息文本在 ecServerMessages_*xx_XX*.properties 文件中定义（其中 *_xx_XX* 是语言环境指示符，例如 *_en_US*），如下所示:

```
_ERR_FINDER_EXCEPTION  =
     The following Finder Exception occurred during processing: "{0}".
```

捕获系统异常时，可以使用一组预定义的消息。这些消息如下表描述:

消息对象	描述
_ERR_FINDER_EXCEPTION	在从 EJB 查找函数方法调用返回错误时抛出。
_ERR_REMOTE_EXCEPTION	在从 EJB 远程方法调用返回错误时抛出。
_ERR_CREATE_EXCEPTION	在创建 EJB 实例发生错误时抛出。
_ERR_NAMING_EXCEPTION	在从名称服务器返回错误时抛出。
_ERR_GENERIC	在发生意外系统错误时抛出。例如，一个空指针异常。

当捕获应用程序异常时，您可以使用在相应的 ecServerMessages_xx_xx.properties 文件中指定的现有消息，也可以创建存储在新属性文件中的新消息。如前面所指定，不得修改任何 ecServerMessages_xx_XX.properties 文件。

以下代码片段显示了在命令中捕获应用程序异常的示例:

```
try {
// your business logic

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

上述的 _ERR_CUSTOMER_INVALID ECMessage 对象定义如下:

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

构造新的用户消息时，应当将它们指定为 USER 类型，如下所示:

ECMessageType.USER

_ERR_CUSTOMER_INVALID 消息的文本包含在 ecCustomerMessages.properties 文件中。该文件必须驻留在类路径中的目录中。文本定义如下:

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

创建消息

如果命令抛出使用新消息的 ECApplicationException，则您必须创建此新消息。创建新消息包含以下步骤:

1. 创建包含此消息键的新类。
2. 创建包含 ECMessage 对象的新类。
3. 创建资源绑定。
4. 对消息进行单元测试。

有关每个步骤的详细信息可以在以下章节中找到。

为消息键创建类

创建新的用户消息的第一步是创建包含新消息键的类。消息键是个唯一指示符，记录服务使用它在资源绑定中定位相应的消息文本。这个新类必须在您自己的数据包中创建，且存储在与 WebSphere Commerce 项目分离的某个项目中。

请考虑以下名为 MyNewMessages 的示例，您在其中创建了一个新类，名为 MyMessageKeys，它包含 _ERR_CUSTOMER 和 _ERR_CUSTOMER_INVALID_ID 消息键，且您将此类放入 com.mycompany.messages 数据包。在此情况下，该类定义如下所示:

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

为消息键提供“字符串”包装允许编译器检查它们的有效性。

为 ECMessage 对象创建类

在您为消息键创建类的同一个数据包中，创建另一个包含 ECMessage 对象的类。ECMessage 类定义了消息对象的结构。它用于检索和保留对语言环境敏感的文本消息。

此消息对象具有以下属性: 严重性、类型、键、资源绑定以及相关资源绑定。此类具有若干构造函数方法。关于完整的详细信息，请参阅 WebSphere Commerce 联机帮助的“参考”部分。

依照 MyNewMessages 示例，在 com.mycompany.messages 数据包中创建名为 MyMessages 的新类，如下所示:

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

public class MyMessages
    {
        static String myResourceBundle = "ecCustomerMessages";
        public static ECMessage _ERR_CUSTOMER = new ECMessage
             (ECMessageSeverity.ERROR,ECMessageType.USER,
                   MyMessageKeys._ERR_CUSTOMER, myResourceBundle);
        public static ECMessage _ERR_CUSTOMER_INVALID_ID = new ECMessage
             (ECMessageSeverity.ERROR, ECMessageType.USER,
                  MyMessageKeys._ERR_CUSTOMER_INVALID_ID,
                  myResourceBundle);

    }
```

在上述代码片段中，import 语句对于创建 ECMessage 对象是必须的。对象 MyMessage._ERR_CUSTOMER 是严重性为 ERROR 的用户消息。WebSphere

Commerce 记录服务使用 MyMessageKeys._ERR_CUSTOMER 查找包含在 *ecCustomerMessages* 属性文件中的消息文本。

创建用户消息资源绑定

必须创建新资源绑定，以存储消息键及其相应的消息文本。该资源绑定可以作为 Java 对象或者属性文件实现。由于属性文件更易翻译和维护，建议使用属性文件。属性文件用于 WebSphere Commerce 消息。

要继续 MyNewMessages 示例，请创建名为 ecCustomerMessages.properties 的文件文件。如果单个商店小服务程序将使用该消息，请将此文件放置在以下目录中:

```
drive:\WebSphere\AppServer\installedApps\WC_EnterpriseApp_instanceName.ear\
   wcstores.war\WEB-INF\classes
```

如果单个工具小服务程序将使用该消息，请将此文件放置在以下目录中:

```
drive:\WebSphere\AppServer\installedApps\WC_EnterpriseApp_instanceName.ear\
   wctools.war\WEB-INF\classes
```

如果企业应用程序中任何小服务程序都使用该消息，请将此文件放置在以下目录中:

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

由于属性文件包含数对消息键和相应的消息文本，ecCustomerMessages.properties 文件包含以下行:

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

对消息进行单元测试

一旦创建了包含消息键的类、包含 ECMessage 对象的类和资源绑定，便应当对消息进行单元测试。

要测试以上部分描述的新消息，请执行以下步骤:

1. 创建一个用于测试目的的新类。在此情况下，创建 MyTestingClass。
2. 将以下 import 语句添加到类

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

3. 将 main() 方法添加到该类。
4. 检查以下代码片段。根据自身需要进行修改（例如，确保目录路径指向系统的有效配置文件）并将其插入 main() 方法

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

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

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

代码前三行初始化了 WebSphere Commerce 记录服务。代码的最后一行指示 ECMessageLog 打印出消息 MyMessage._ERR_CUSTOMER。MyTestingClass 和 main 是日志跟踪格式的一部分。Hello 字符串被置于 ecCustomerMessages.properties 文件中定义的消息的 {0} 占位符中。

5. 运行此类。在日志文件中，消息跟踪显示与如下所示类似:

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

您可以运行类似测试查看第二个消息。

执行流程跟踪

WebSphere Commerce 包含 ECTrace 类，此类用于跟踪运行在 WebSphere Commerce Server 上的组件的执行流程。ECTrace 类是 com.ibm.commerce.ras 数据包的一部分。

创建新的业务逻辑时，为了调试目的，可以在代码中插入跟踪以跟踪方法。来自跟踪的信息捕获在跟踪日志中。您可以为跟踪指定入口和出口点。此外，还可以指定在这两点之间跟踪特定数据。

要使用跟踪，必须对要运行跟踪的组件启用跟踪。可以使用管理控制台或配置管理器对特定组件启用跟踪。

跟踪定制的代码时，*必须*使用 EXTERN 组件。在配置管理器中，这称为 *External*。

要在您的代码中设置跟踪的入口点，请使用以下语法:

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

其中 *myClassName* 是包含该跟踪方法的类的字符串表示法。由于此字符串可以用于跟踪文件分析，它应当包含全限定类名。如果所跟踪的方法是静态的，那么 myClassName 的示例声明为

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

如果所跟踪的方法不是静态的，那么 myClassName 的示例声明为

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

要设置跟踪点以在方法内跟踪数据，请使用以下语法:

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

其中 *myText* 是出现在跟踪日志中的文本。

要在您的代码中设置跟踪的出口点，请使用以下语法:

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

如果需要跟踪从所跟踪的方法返回的对象，那么请如下设置出口点:

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

其中 returnedObject 表示由此方法返回的 Java 对象。

假如有这样一个示例，在该示例中，您需要跟踪名为 MyNewControllerCmd 的新控制器命令的 performExecute 方法。以下代码片段显示如何在 performExecute 方法中使用 ECTrace 方法。

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

      super.performExecute();


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


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

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


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

}
```

当调用上述 performExecute 方法后，跟踪日志文件捕获以下信息:

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

建议只在主要函数上使用跟踪。由于跟踪意在由商店开发者使用，因此不对多种语言启用。这与消息相反，由于消息用于管理目的并且用户消息是显示给顾客的，消息可以对多种语言启用。

JSP 模板错误处理

JSP 模板的错误处理可以用不同的方法执行:

- 页面内的错误处理
  对于需要更复杂的错误处理和恢复的 JSP 文件，可以将该文件写入数据 bean 的直接处理错误。JSP 文件可以捕获由数据 bean 抛出的异常，或者它可以检查每个数据 bean 中的错误代码集，而这取决于激活数据 bean 的方式。JSP 文件然后根据所接收的错误采取相应的恢复措施。注意: JSP 文件可以使用以下错误处理作用域的任意组合。
- 页面级别的错误 JSP
  JSP 文件也可以通过 JSP 错误标签，从它自身内发生的异常指定自己的缺省错误 JSP 模板。这使得 JSP 程序可以指定自身的错误处理。未指定 JSP 错误标记的 JSP 文件将使错误下降到应用程序级别的 JSP 错误模板。在页面级别错误 JSP，其必须调用 JSP helper 类（com.ibm.server.JSPHelper）回滚当前事务。
- 应用程序级别的错误 JSP
  WebSphere 下的应用程序可以在其任何小服务程序或 JSP 文件中发生错误时指定缺省错误 JSP 模板。应用程序级别错误 JSP 模板可以用作（单一商店模型的）购物中心级别或商店级别错误处理程序。在应用程序级别错误 JSP 模板中，

调用必须是对小服务程序 helper 类做出的，以便回滚到当前的交易。这是由于 Web 控制器不在执行路径上，不能回滚事务。只要有可能，您须依赖前面两种类型的 JSP 错误处理。仅在需要时使用应用程序级别的错误处理策略。

第 6 章 命令实现

本部分提供关于如何写新控制器命令、任务命令和数据 bean 命令的信息。它还描述了如何扩展现有的控制器命令、任务命令和数据 bean 命令。

注: Business 本章不描述业务策略命令。关于业务策略命令的更多信息，请参阅第 135 页的第 7 章，『贸易协议和业务策略（商务版）』。

新命令 — 简介

WebSphere Commerce 编程模型定义了四种类型的命令: 控制器、任务、视图和数据 bean 命令。为电子交易应用程序创建新的商业逻辑时，您可能需要创建新的控制器、任务、数据 bean 命令。您无须创建新的视图命令。关于视图命令的更多信息可稍后在本部分找到。

新命令必须实现它们相应的接口（该接口必须依次从现有接口扩展）。为了简单化写命令的过程，WebSphere Commerce 包含了每一类型命令的抽象实现类。新命令应当从这些类扩展。

作为概述，下表提供了有关新命令应当从哪个实现类扩展、以及应当实现哪个接口的信息:

命令类型	示例命令名	扩展于	实现示例接口
控制器命令	MyControllerCmdImpl	com.ibm.commerce.command.ControllerCommandImpl	MyControllerCmd
任务命令	MyTaskCmdImpl	com.ibm.commerce.command.TaskCommandImpl	MyTaskCmd
数据 bean 命令	MyDataBeanCmdImpl	com.ibm.commerce.command.DataBeanCommandImpl	MyDataBean

注: 实现类名称间的任何空格仅为显示目的。

下图说明了新控制器命令的接口和实现类与现有的抽象实现类和接口之间的关系。抽象类和接口都在 com.ibm.commerce.command 数据包中找到。

新控制器命令

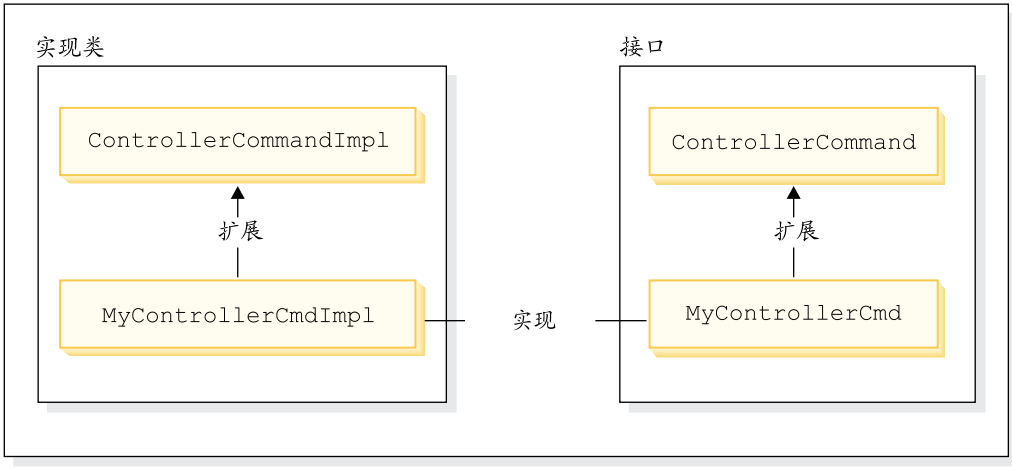


图 25.

下图说明了新任务命令的接口和实现类与现有的抽象实现类和接口之间的关系。抽象类和接口都在 com.ibm.commerce.command 数据包中找到。

新任务命令

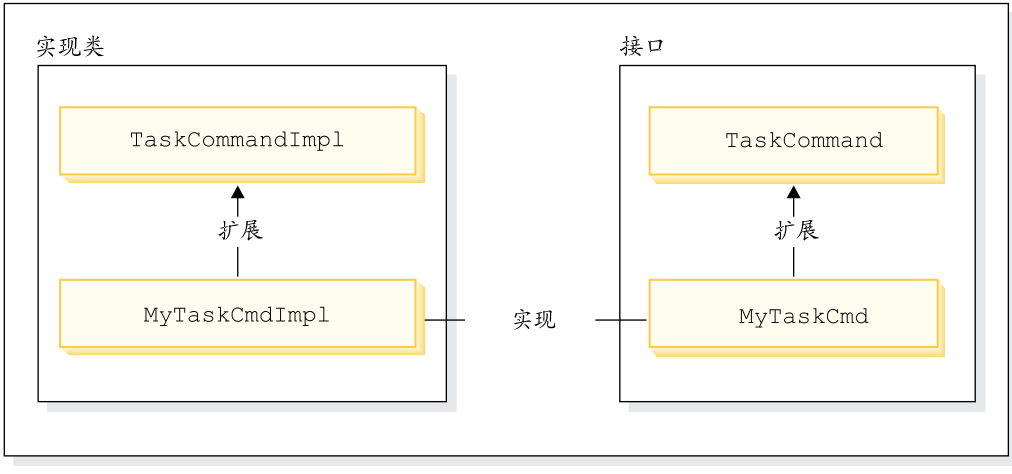


图 26.

下图说明了新数据 bean 的接口和实现类与现有的抽象实现类和接口之间的关系。抽象类和接口都在 com.ibm.commerce.command 数据包中找到。

新数据 bean 命令

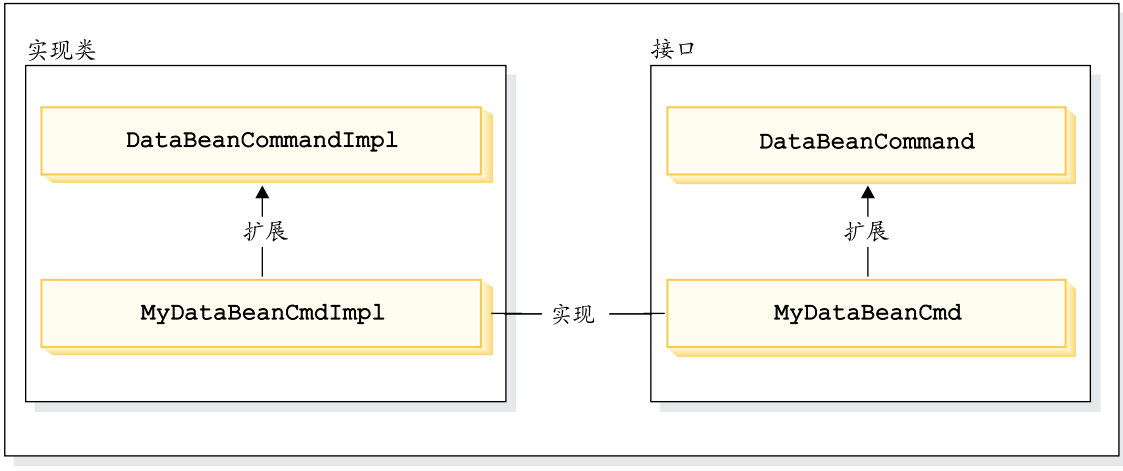


图 27.

视图命令有两个主要功能：格式化响应以及将响应发送给客户机。许多一般视图命令都已经提供，这些命令使用不同的协议将响应发送给客户机。格式化功能通常由调用 JSP 模板的视图命令处理。例如，RedirectViewCommand 视图命令将客户机定向到 URL 以获取响应（该响应然后由指定的 JSP 模板格式化）。ForwardViewCommand 视图命令将此请求转发给 JSP 模板以格式化，然后该页面显示给客户机。

您可以使用视图命令模型通过创建新 JSP 模板来创建新的*视图*（对客户机的响应）。但 JSP 模板应当由现有的视图命令之一调用。

封装定制代码

创建定制代码时，必须遵守特定的代码组织结构。通常，定制代码在与 WebSphere Commerce 包含的数据包和项目分离的数据包和项目中维护。

当创建了新的命令时，您必须将它们放置在为符合业务需求而适当命名的包中。即，如果命令适用于某一特定商店，就将这些命令封装到专用于该商店的数据包中。如果适用于多家商店，就将它们一一对应封装。例如，您可能有以下数据包：

- `com.bigbusiness.storeA.commands`
- `com.bigbusiness.storeB.commands`
- `com.bigbusiness.commands`

上述封装结构允许商店级别上的业务逻辑之间有所区别。此外，这些数据包必须存储在与 WebSphere Commerce 项目分离的项目中。例如，上述数据包可以置于名为 BigBusinessCustomCode 的项目中。

创建新的数据 bean 时，它们必须保留在与命令逻辑分离的数据包中，不过，此数据包可以保留在存储命令数据包的项目中。依据上述示例，您可能将 com.bigbusiness.databeans 数据包置于 BigBusinessCustomCode 项目中。

创建新的实体 bean 时，必须将它们存储在唯一的项目中。所以，您可能有包含 com.bigbusiness.objects 数据包的 BigBusinessCustomEntityBeans 项目。

此封装策略对于代码部署目的来说是必须的。

命令上下文

命令上下文是 Web 控制器的句柄。通过使用命令上下文，命令可以从 Web 控制器获取信息。可用信息的示例包括用户标识、用户对象、语言标识和商店标识。

写命令时，通过调用命令超类的 getCommandContext() 方法，您可以访问命令上下文。当 Web 控制器调用命令时，将命令上下文设置为控制器命令。控制器命令应当将命令上下文传播到处理期间调用的任何任务或控制器命令。命令可以从命令上下文获取以下关键信息:

getUserId() 和 getUser()

获取当前用户标识或用户对象。当前会话的用户标识保存在会话上下文中。有两种方法可以保存会话上下文: 使用 WebSphere Commerce cookie 或使用 WebSphere Application Server 持久会话对象。命令上下文隐藏了命令的会话管理的复杂性。

getStoreId()、getStore() 和 getStore(storeId)

获取与当前请求相关联的商店。Web 控制器返回 URL 中的商店标识。如果商店标识未在 URL 中指定，它可以在前一个请求保存的会话对象中检索到。WebSphere Commerce 运行时环境维护了一组经常访问的对象。例如，它维护了一组商店对象。命令应当总是从命令上下文获取商店对象，以利用 Web 控制器中的对象高速缓存。在命令上下文中，您可以通过调用 getStore() 方法获取当前商店，或通过调用 getStore(storeId) 方法获取特定的商店对象。

getLanguageId()

返回应用于当前请求的语言标识。Web 控制器实现全局化框架。此框架后的概念是确定用户首选且商店支持的语言。如果 URL 包含语言标识，Web 控制器将确定此语言是否受商店支持，如果受支持，则它是 getLanguageId() 方法返回的语言标识。如果在 URL 中没有包含语言标识，则 Web 控制

器将通过判定树，以确定在当前会话对象中或在用户注册的首选项中，是否有语言标识（即受商店支持的语言标识），或它最终将返回商店的缺省语言标识。

getCurrency()

返回用于当前请求的货币。由于货币是全局化框架的一部分，因此此命令后的逻辑与 getLanguageId() 方法类似。

getCurrentTradingAgreements() 和 getTradingAgreement(tradingAgreementId)

返回用于当前会话的贸易协议集合。此集合可以是授权给用户的所有贸易协议，或可以是由 ContractSetInSession 命令定义的子集。命令应当总是从命令上下文获取贸易协议对象，以利用 Web 控制器中的对象高速缓存。在命令上下文中，您可以通过调用 getCurrentTradingAgreements() 方法获取当前贸易协议，或通过调用 getTradingAgreement(tradingAgreementId) 方法获取特定的贸易协议对象。

命令上下文应当作为只读对象使用。您不能调用其设置函数方法。设置函数方法保留以供 WebSphere Commerce 运行时环境使用，并且它们有可能在将来的发行版中不能使用。

关于命令上下文 API（应用程序编程接口）的完整详细信息，请参阅 WebSphere Commerce 联机帮助的“参考”部分。

新控制器命令

如前所述，新的控制器命令应当从抽象控制器命令类（com.ibm.commerce.command.ControllerCommandImpl）中扩展。写新的控制器命令时，应当从抽象类重设以下方法:

- isGeneric()
- isRetriable()
- setRequestProperties(com.ibm.commerce.datatype.TypedProperty reqParms)
- validateParameters()
- getResources()
- performExecute()

有关每个先前方法的更多信息，您可以在以下章节中找到。

isGeneric 方法

在标准 WebSphere Commerce 实现中有多种类型的用户。包括一般用户、临时用户和注册用户。注册用户组中有顾客和管理员。

一般用户具有在整个系统中都会使用的公共用户标识。此公共用户标识支持以消耗系统资源最小的方式在站点上进行常规浏览。对一般浏览使用此公共用户标识更为有效，因为 Web 控制器不需要为可以由一般用户调用的命令检索用户对象。

isGeneric 方法返回一布尔值，指定命令能否由一般用户调用。控制器命令超类的 isGeneric 方法设置此值为 `false`（意思是调用函数必须为注册用户或临时用户）。如果您的新控制器命令可以由一般用户调用，请重设此方法使其返回 `true`。

如果新命令没有读取或创建与用户相关联的资源，则应当重设此方法使其返回 `true`。ProductDisplay 命令是个可以由一般用户调用的命令的示例。该命令判断是否允许任何用户可以查看产品。OrderItemAdd 命令是用户必须是临时用户或注册用户（这样，isGeneric 返回 `false`）的命令的示例。

当 isGeneric 返回值 `true` 时，Web 控制器并没有为当前会话创建新的用户对象。同样，由于 Web 控制器无须检索用户对象，一般用户可以调用的命令运行更快。

使用此方法以使得一般用户可以调用命令的语法如下所示:

```
public boolean isGeneric()
{
     return true;
}
```

isRetriable 方法

isRetriable 方法返回一布尔值，指定能否在事务回滚异常时重试命令。新控制器命令超类的 isRetriable 方法返回值 `false`。如果可以在事务回滚异常时重试命令，您应该重设该方法并返回值 `true`。

OrderProcess 命令是个在事务异常情况下不可重试的命令的示例。此命令调用第三方支付授权过程。由于不能逆向授权，所有不能重试此命令。ProductDisplay 是个可以重试的命令的示例。

使得命令在事务回滚异常时可重试的语法如下所示:

```
public boolean isRetriable()
{
     return true;
}
```

setRequestProperties 方法

setRequestProperties 方法由 Web 控制器调用，以将所有输入属性传送到控制器命令。控制器命令必须分析该输入属性并在此方法内显式设置每一单独属性。控制器命令的此属性显式设置本身便提出了类型安全属性概念。

使用此方法的语法如下所示:

```
public void setRequestProperties(
   com.ibm.commerce.datatype.TypedProperty reqParms)
{

     // parse the input properties and explicitly set each parameter

}
```

validateParameters 方法

validateParameters 方法用于执行初始参数检查以及任何必要的参数解析。例如，它可用于解析 orderId=*。此方法在 getResources 和 performExecute 方法之前调用。关于此顺序的更多详细信息，请参阅第 91 页的『访问控制交互』。

getResources 方法

此方法用于实现资源级别访问控制。它返回命令要对其操作的“资源-操作”对向量。如果不作任何返回，将不执行资源级别访问控制。关于访问控制的更多信息，请参阅第 79 页的第 4 章，『访问控制』。

performExecute 方法

performExecute 方法包含您命令的业务逻辑。在执行任何新的业务逻辑之前，它都应当调用命令超类的 performExecute 方法。最后，必须返回一个视图名称。

以下显示了新控制器命令中的 performExecute 方法的示例语法。在此情况下，响应使用重定向视图命令，不过，也可以使用转发视图命令或定向视图命令:

```
public void performExecute() throws ECException
{
     super.performExecute();

//////////////////////////////////////////////
//  your business logic                     //
//////////////////////////////////////////////

// Create a new TypedProperty for response properties.
TypedProperty rspProp = new TypedProperty();

// set response properties
rspProp.put(ECConstants.EC_VIEWTASKNAME, "MyView");
//////////////////////////////////////////////////
//  The following line is optional.  The VIEWREG   //
//  table can specify the redirect URL.           //
//////////////////////////////////////////////////

rspProp.put(ECConstants.EC_REDIRECTURL, MyURL);

//////////////////////////////////////////////////////////////////////////
//   If you are using a forward view, you can set the                    //
//   response properties as follows:                                     //
//  TypedProperty rspProp = new TypedProperty();                        //
```

```
//   rspProp.put(ECConstants.EC_VIEWTASKNAME, "MyView");                  //
//   rspProp.put(ECConstants.EC_DOCPATHNAME, "MyJSP.jsp");                //
//                                                                        //
//   Again, it is optional to explicitly set the name of the JSP template.//
//   The VIEWREG table can specify the JSP template.                      //
////////////////////////////////////////////////////////////////////////////

setResponseProperties(rspProp);
}
```

如果您在 performExecute 方法中指定重定向 URL，并且 VIEWREG 表中存在一个条目，则代码中指定的值优先于 VIEWREG 表中的值。在代码中对 JSP 模板的指定使用相同的优先顺序。

长时间运行的控制器命令

如果执行控制器命令要花费长时间，您可以将其分割为两个命令。第一个命令作为 URL 请求的结果执行，仅将第二个命令添加到调度程序，以使其作为后台作业运行。如下图所示:

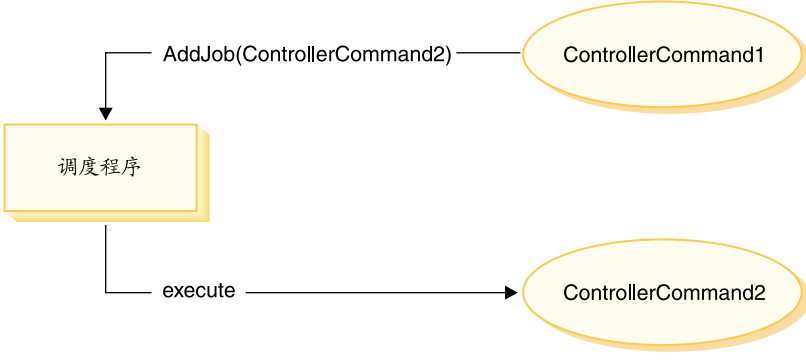


图 28.

上图显示的流程如下所示:

1. ControllerCommand1 作为 URL 请求的结果执行。
2. ControllerCommand1 将作业添加到调度程序。该作业为 ControllerCommand2。ControllerCommand1 在将该作业添加到调度程序后立即返回一个视图。
3. 调度程序将 ControllerCommand2 作为后台作业执行。

在这个情况中，客户机通常从 ControllerCommand2 轮询结果。ControllerCommand2 应当将作业状态写入数据库。

格式化视图命令的输入属性

控制器命令完成后，将返回应当执行的视图的名称。此视图可能要求将几个输入参数传递给它。如以下列表所描述，这些输入参数可以有三个源:

- 存储在 CMDREG 表 PROPERTIES 列中的缺省属性
- 来自 VIEWREG 表 PROPERTIES 列的缺省属性
- 来自 URL 的输入属性

关于这些参数在 JSP 模板的属性中是如何合并和设置的更多信息，请参阅第 40 页的『设置 JSP 属性 — 概述』。本部分描述了可如何格式化视图命令的输入属性。

对于重定向视图命令，检查两个主题:

- 平面化查询字符串以支持 URL 重定向
- 处理对重定向 URL 的长度的限制

对于转发视图命令，检查以下主题: 输入参数的枚举以及设置它们作为 HttpServletRequestObject 中的属性。

平面化输入参数至 HttpRedirectView 的查询字符串中

所有传递给重定向视图命令的输入参数都平面化到 URL 重定向的查询字符串中。例如，假定对重定向视图命令的输入包含以下属性:

```
URL = "MyView?p1=v1&p2=v2";
ip1 = "iv1";   // input to orginal controller command
ip2 = "iv2" ;  // input to original controller command
op1 = "ov1";
op2 = "ov2";
```

根据上述输入参数，最终的 URL 是

```
MyView?p1=v1&p2=v2&ip1=iv1&ip2=iv2&op1=ov1&op2=ov2
```

注意: 如果命令是使用 SSL，则将对参数加密，并且最终的 URL 显示为

```
MyView?krypto=encrypted_value_of"p1=v1&p2=v2&ip1=iv1&ip2=iv2&op1=ov1&op2=ov2"
```

处理限制长度的重定向 URL

缺省情况下，控制器命令的所有输入参数都传播到重定向视图命令。如果在重定向 URL 中对字符的数目有限制，这就可能引起问题。限制长度的示例之一就是当客户机正在使用 Internet Explorer 浏览器时。对于此浏览器，URL 不能超出 2083 个字节。如果 URL 超出此限制，则 URL 将被截短。这样，如果有大量的输入参数，或如果您正在使用加密（因为加密字符串通常比未加密字符串长两到三倍），则您可能会遇到问题。

处理限制长度的重定向 URL 有两种办法:

1. 覆盖控制器命令中的 getViewInputProperties 方法，以只返回需要传递到重定向视图命令的参数集。
2. 在 URL 参数中使用指定的特殊字符，以指出哪些参数可以从输入参数字符串中除去。

要演示每个上述方法，请考虑控制器命令的以下输入参数集:

```
URL="MyView";
// All of the following are inputs to the original controller command.
ip1="ipv1";
ip2="ipv2";
ip3="ipv3";
iq1="iqv1";
iq2="iqv2";
ir1="ipr1";
ir2="ipr2";
is="isv";
```

如果正在覆盖 getViewInputProperties 方法，则可以写新方法从而仅以下参数传递到视图命令:

```
ir2="ipr2";
is="isv";
```

使用第二个方法时，可使用特殊参数来调用视图命令，以指出应除去特定输入参数。例如，可通过指定以下作为 URL 参数，来取得同样的结果:

```
URL="MyView?ip*=&iq*=&ir1="
```

此 URL 参数对 WebSphere Commerce 运行时框架指示以下信息:

- `ip*=` 指定表示应除去所有名称以 `ip` 开始的参数。
- `iq*=` 指定表示应除去所有名称以 `iq` 开始的参数。
- `ir1=` 指定表示应除去 `ir1` 参数。

设置 HttpForwardView 的 HttpServletRequest 对象中的属性

缺省 HttpForwardViewCommandImpl 枚举所有传递到命令的参数，并将它们设置为 HttpServletRequest 对象中的属性。

例如，假定传递到转发视图命令的 requestProperties 对象包含以下属性:

```
p1="pv1";
p2="pv2";
p3=pv3;   // pv3 is an object
```

然后使用 request.setAttribute() 方法将以下属性传递到 JSP 模板。

```
request.setAttribute("p1", "pv1");
request.setAttribute("p2", "pv2");
request.setAttribute("p1", pv1);
request.setAttribute("RequestProperties", requestProperties);
request.setAttribute("CommandContext", commandContext);
```

其中 requestProperties 是传递到命令的 TypedProperty 对象，commandContext 是传递到命令的命令上下文对象，而 p1、p2 和 p3 是在 requestProperties 对象中定义的参数。

控制器命令的数据库提交和回滚

控制器命令的执行过程中，经常创建或更新数据。很多情况下，在事务结束时必须用新信息更新数据库。事务由 Web 控制器管理。

Web 控制器在调用控制器命令之前标记事务的开始。完成执行控制器命令后，控制器命令将视图名称返回给 Web 控制器。Web 控制器负责标记事务的结束。事务结束的实际点（调用视图之前或之后）取决于所使用的视图类型。

有三种类型的视图命令:

- 转发视图命令
- 重定向视图命令
- 定向视图命令

通过在 VIEWREG 表中查找视图名称，Web 控制器确定视图要使用的视图命令。

如果 VIEWREG 表中的条目指定使用 ForwardViewCommand，Web 控制器则将控制器命令的结果转发到相应的 ForwardViewCommand 实现类（也在 VIEWREG 中指定）。视图命令在当前事务的上下文中执行。在此情况下，数据库提交或回滚直到视图命令完成后才发生。

如果 VIEWREG 表中的条目指定使用 RedirectViewCommand，Web 控制器则将控制器命令的结果转发到相应的 RedirectViewCommand 实现类。然后视图命令在当前事务作用域外操作，并且数据库提交和回滚在调用重定向的视图命令之前发生。

如果 VIEWREG 表中的条目指定使用 DirectViewCommand，Web 控制器则将控制器命令的结果转发到相应的 DirectViewCommand 实现类。视图命令在当前事务的上下文中执行。在此情况下，数据库提交或回滚直到视图命令完成后才发生。（注意 ForwardViewCommand 和 DirectViewCommand 是类似的。ForwardViewCommand 将结果转发给 JSP 模板。而 DirectViewCommand 将结果作为输入流接收并作为输出流传送。它使用以字节形式处理数据的 getRawDocument 方法，或以文本形式处理数据的 getTextDocument 方法。）

如果视图命令在与控制器命令相同的事务作用域中执行，则视图命令的错误将导致整个事务的回滚。这可能是或不是所期望的结果，它取决于您的业务逻辑。

控制器命令事务作用域示例

要说明控制器命令的事务作用域的不同（取决于所使用的视图命令类型），请研究以下示例。

案例 1: 在控制器命令事务作用域内执行视图

假定您已创建一个名为 YourControllerCmdA 的新控制器命令。该命令的 performExecute 方法将包括以下内容:

```
.
.
// Create a new TypedProperty object for output.
TypedProperty rspProp = new TypedProperty();


//////////////////////
// Business logic   //
//////////////////////

// Return the view
rspProp.put(ECConstants.EC_VIEWTASKNAME, "YourView");
SetResponseProperties(rspProp);
```

在上述代码片段中，控制器命令将 “YourView” 作为视图返回。YourView 注册在 VIEWREG 表中。以下是注册 YourView 的 insert 语句的示例。

```
insert into VIEWREG (ViewName, DeviceFmt_id, storeEnt_id, interfacename,
classname, properties)

values ('YourView',  -1, XX,'com.ibm.commerce.command.ForwardViewCommand',
'com.ibm.commerce.command.HttpForwardViewCommandImpl','docname=YourView.jsp');
```

其中 XX 是商店标识。由于视图使用 com.ibm.commerce.command.HttpForwardViewCommandImpl 实现类，因此 Web 控制器使用一般转发视图命令。

根据上述命令的注册，Web 控制器在控制器命令事务作用域内启用 YourView.jsp 文件。如果 YourView.jsp 发生错误，则事务失败并发生数据库回滚。结果整个控制器命令失败。

案例 2: 在控制器命令事务作用域外执行视图

假定您更喜欢将信息提交到数据库，即使视图中有可能出现错误。要在控制器命令事务的作用域外执行视图，必须将此视图作为重定向执行。

要将视图作为重定向执行，控制器命令的 performExecute 方法将以如下方式返回视图:

```
.
.
// Create a new TypedProperty object for output.
TypedProperty rspProp = new TypedProperty();


//////////////////////
// Business logic   //
//////////////////////

// Return the view
rspProp.put(ECConstants.EC_VIEWTASKNAME, EC_GENERIC_REDIRECTVIEW);
rspProp.put(EC_Constants.EC_REDIRECTURL, "YourView2");
```

以下示例 SQL 语句支持重定向策略:

```
insert into VIEWREG (ViewName, DeviceFmt_id, storeEnt_id, interfacename,
classname, properties)

values ('YourView2',  -1, XX,'com.ibm.commerce.command.ForwardViewCommand',
'com.ibm.commerce.command.HttpForwardViewCommandImpl','docname=YourView2.jsp');
```

其中 XX 是商店标识。

由于命令将 EC_GENERIC_REDIRECTVIEW 值作为响应属性参数传递，因此 Web 控制器使用一般重定向视图命令。一般重定向视图注册在 VIEWREG 表中，其中有以下信息:

- ViewName = RedirectView
- DeviceFmt_Id = -1
- InterfaceName = com.ibm.commerce.command.RedirectViewCommand
- ClassName = com.ibm.commerce.command.HttpRedirectViewCommandImpl

Web 控制器调用将重定向 URL 作为输入属性的一般重定向视图命令。响应被重定向到该重定向 URL。重定向发生后，将调用 YourView2。这是作为一般转发视图实现的。

新任务命令

新任务命令应该从抽象任务命令类（com.ibm.commerce.command.TaskCommandImpl）扩展并实现扩展 com.ibm.commerce.TaskCommand 接口的接口。如第 116 页的图所示，新任务命令应该如下定义:

```
public class MyTaskCmdImpl extends com.ibm.commerce.command.TaskCommandImpl
     implements MyTaskCmd {

}
```

任务命令的所有输入和输出属性必须在命令接口（例如 MyTaskCmd）中定义。调用函数程序调用任务命令接口，而非该任务命令实现类。这使您能够有任务命令的多个实现（每家商店具有一个实现），而调用函数无须考虑调用哪个实现类。

接口中定义的所有方法必须在实现类中实现。由于命令上下文应当由调用函数（控制器命令）设置，因此任务命令无需设置命令上下文。但任务命令可以使用命令上下文从 Web 控制器获得信息。

除了实现任务命令接口中定义的方法外，还应当覆盖 com.ibm.commerce.command.TaskCommandImpl 类中的 performExecute 方法。

performExecute 方法包含任务命令执行的特定工作单元的业务逻辑。在执行任何业务逻辑之前，它必须调用此任务命令的超类的 performExecute 方法。以下代码片段显示了任务命令的 performExecute 方法示例。

```
public void performExecute() throws ECException
{
      super.performExecute();

     // Include your business logic here.

     // Set output properties so the controller command
     // can retrieve the result from this task command.
}
```

运行时框架调用控制器命令的 getResources 方法来确定该命令将访问哪些可保护资源。可能会出现这样一种情况，在控制器命令的作用域期间执行一个任务命令，而它试图访问控制器命令的 getResources 方法没有返回的资源。如果确实如此，任务命令本身可以实现 getResources 方法以确保已为可保护资源提供访问控制。

注意：缺省情况下，getResources 为任务命令返回 null，并且不执行资源级别的访问控制检查。因此，如果任务命令访问可保护资源，则您必须覆盖它。

现有命令的定制

本部分描述定制现有控制器、任务和数据 bean 命令的各种不同方法。

定制现有的控制器命令

控制器命令为业务过程封装业务逻辑。业务过程中的单个工作单元可以由任务命令执行。这样，定制控制器命令有若干种方法，其中某些方法与定制任务命令相关。

定制控制器命令时，您可以完成以下步骤:

- 将附加处理和逻辑添加到现有控制器命令。这可在现有业务逻辑之前、现有逻辑之后或同时在之前和之后添加。
- 替换一个或多个任务命令。这允许您修改如何执行业务过程中的某一特定步骤。
- 替换控制器命令调用的视图。

以下部分提供如何作上述修改的详细信息。

添加新业务逻辑至控制器命令

假定一个现有 WebSphere Commerce 控制器命令，名为 ExistingControllerCmd。遵循 WebSphere Commerce 命名约定，该控制器命令会有名为 ExistingControllerCmd 的接口类和名为 ExistingControllerCmdImpl 的实现类。现在假设有业务需求，因此您必须将新业务逻辑添加到此现有命令。逻辑的一部分必须在现有命令逻辑之前执行，而还有一部分必须在现有命令逻辑之后执行。

添加新的业务逻辑的第一步是创建扩展原始实现类的新实现类。在此示例中，您将创建扩展 ExistingControllerCmdImpl 类的新 ModifiedControllerCmdImpl 类。此新实现类应当实现原始接口（ExistingControllerCmd）。

在新实现类中，您必须创建新 performExecute 方法来覆盖现有命令的 performExecute。在新 performExecute 方法中，有两种方法可以插入新业务逻辑: 您可以直接在控制器命令中包含代码，或创建新的任务命令来执行新的业务逻辑。如果创建新的任务命令，则您必须从控制器命令中实例化新的任务命令对象。

以下代码片段演示了如何通过直接在控制器命令中包含逻辑，来将新的业务逻辑添加到现有控制器命令的开头和末尾;

```
public class ModifiedControllerCmdImpl extends ExistingControllerCmdImpl
   implements ExistingControllerCmd {
      public void performExecute ()
         throws com.ibm.commerce.exception.ECException
         {

         /* Insert new business logic that must be
            executed before the original command.
            */
```

```
        // Execute the original command logic.
      super.performExecute();

        /* Insert new business logic that must be
           executed after the original command.
           */
         }
    }
```

以下代码片段演示了如何通过从控制器命令中实例化新的任务命令，来将新的业务逻辑添加到现有控制器命令的开头。此外，您也将创建新的任务命令接口和实现类，并在命令注册表中注册此任务命令。

```
// Import the package with the CommandFactory
import com.ibm.commerce.command.*;

public class ModifiedControllerCmdImpl extends ExistingControllerCmdImpl
   implements ExistingControllerCmd {
      public void performExecute ()
         throws com.ibm.commerce.exception.ECException
         {
      MyNewTaskCmd cmd = null;
     cmd = (MyNewTaskCmd) CommandFactory.createCommand(
               "com.mycompany.mycommands.MyNewTaskCommand",
               getStoreId());

            /*
         Set task command's input parameters, call its
         execute method and retrieve output
         parameters, as required.
            */

      super.performExecute();
         }
   }
```

不管是在控制器命令中包含新的业务逻辑，还是创建任务命令来执行逻辑，您都必须更新 WebSphere Commerce 命令注册表中的 CMDREG 表，以将新的控制器命令实现类与现有的控制器命令接口相关联。以下 SQL 语句显示了一个更新示例:

```
update CMDREG
set CLASSNAME='ModifiedControllerCmdImpl'
where INTERFACENAME='ExistingControllerCmd'
```

替换由控制器命令调用的任务命令

控制器命令经常调用若干执行单个任务的任务命令。总体说来，这些任务组成控制器命令所代表的业务过程。您可能需要更改过程中的某一特定步骤的执行方法，而不是将新的业务逻辑添加到控制器命令的开头或末尾。在此情况下，您必须用新任务命令的实例化（此新任务命令以您希望的方式执行任务）替换您希望覆盖的任务命令的实例化。

由于 WebSphere Commerce 编程模型的设计，您无需创建新的控制器命令实现类来替换任务命令。控制器命令通过调用命令工厂的 createCommand 方法实例化此任务命令。命令工厂使用任务命令的接口名称，然后根据命令注册表确定正确的实现类。这样，要替换已实例化的任务命令，您必须创建新的任务命令实现类，然后更新命令注册表，以便使原始任务命令接口名称与新的任务命令实现类相关联。关于更多信息，请参阅第 132 页的『定制现有的任务命令』。

替换由控制器命令调用的视图

要替换由控制器命令调用的视图，您需要为控制器命令创建新的实现类。例如，创建扩展 ExistingControllerCmdImpl 并实现 ExistingControllerCmd 接口的新 ModifiedControllerCmdImpl。

在 ModifiedControllerCmdImpl 类中，覆盖 performExecute 方法。在新的 performExecute 方法中，调用 super.performExecute 以确保发生所有命令处理。当执行命令逻辑后，您可以使用响应属性来覆盖调用的视图。以下代码片段显示了如何在视图作为重定向执行时覆盖视图:

```
// Import the packages containing TypedProperty, and ECConstants.
import com.ibm.commerce.datatype.*;
import com.ibm.commerce.server.*;

public class ModifiedControllerCmdImplImpl extends ExistingControllerCmdImpl
   implements ExistingControllerCmd {
      public void performExecute ()
         throws com.ibm.commerce.exception.ECException
         {

         // Execute the original command logic.
      super.performExecute();

// Create a new TypedProperty for response properties.
    TypedProperty rspProp = new TypedProperty();

        // set response properties
        rspProp.put(ECConstants.EC_VIEWTASKNAME, "MyView");
           ////////////////////////////////////////////////////
       //  The following line is optional.  The VIEWREG   //
       //  table can specify the redirect URL.            //
          ////////////////////////////////////////////////////

rspProp.put(ECConstants.EC_REDIRECTURL, MyURL);


  setResponseProperties(rspProp);

         }
   }
```

以下代码片段显示了如何在视图作为转发视图执行时覆盖视图:

```
// Import the packages containing TypedProperty, and ECConstants.
import com.ibm.commerce.datatype.*;
import com.ibm.commerce.server.*;

public class ModifiedControllerCmdImplImpl extends ExistingControllerCmdImpl
   implements ExistingControllerCmd {
      public void performExecute ()
         throws com.ibm.commerce.exception.ECException
         {

         // Execute the original command logic.
      super.performExecute();

// Create a new TypedProperty for response properties.
    TypedProperty rspProp = new TypedProperty();

        // set response properties
        rspProp.put(ECConstants.EC_VIEWTASKNAME, "MyView");

           ////////////////////////////////////////////////
          //  It is optional to explicitly set the name   //
          //  of the JSP template. The VIEWREG table can  //
          //  specify the JSP template.                   //
          ////////////////////////////////////////////////

       rspProp.put(ECConstants.EC_DOCPATHNAME, "MyJSP.jsp");

  setResponseProperties(rspProp);

         }
    }
```

要确定现有控制器命令使用哪个视图，请参阅 WebSphere Commerce 联机帮助的“参考”部分。

定制现有的任务命令

有两种标准方式修改现有的 WebSphere Commerce 任务命令。使用这些修改方法，您可以完成以下任务:

- 将附加处理和逻辑添加到现有任务命令。这可在现有业务逻辑之前、现有逻辑之后或同时在之前和之后添加。
- 用您自己的业务逻辑完全替换现有的业务逻辑。

要完成以上修改，您实际上要创建一个新的任务命令实现类。以下部分提供更多详细信息。

添加新业务逻辑至任务命令

假定一个现有 WebSphere Commerce 任务命令，名为 ExistingTaskCmd。遵循 WebSphere Commerce 命名约定，该任务命令会有名为 ExistingTaskCmd 的接口类和名为 ExistingTaskCmdImpl 的实现类。现在假设有业务需求，因此您必须将

新业务逻辑添加到此现有命令。逻辑的一部分必须在现有命令逻辑之前执行，而还有一部分必须在现有命令逻辑之后执行。

添加新的业务逻辑的第一步是创建扩展原始实现类的新实现类。在此示例中，您将创建扩展 ExistingTaskCmdImpl 类的新 ModifiedTaskCmdImpl 类。此新实现类应当实现原始接口（ExistingTaskCmd）。

在新命令中，您覆盖现有 performExecute 方法，并在调用 super.performExecute 方法之前和之后包含新逻辑。

以下伪代码演示了如何将新的业务逻辑添加到现有任务命令:

```
public class ModifiedTaskCmdImpl extends ExistingTaskCmdImpl
     implements ExistingTaskCmd {


         /* Insert new business logic that must be
            executed before the original command.
         */

         // Execute the original command logic.
      super.performExecute();

         /* Insert new business logic that must be
            executed after the original command.
         */
}
```

您还必须更新 CMDREG 表，以将新的实现类与现有接口相关联。以下 SQL 语句显示了一个更新示例:

```
update CMDREG
set CLASSNAME='ModifiedTaskCmdImpl'
where INTERFACENAME='ExistingTaskCmd'
```

替换现有任务命令的业务逻辑

要替换现有任务命令的业务逻辑，您必须为任务命令创建新的实现类。此新的实现类必须从现有的任务命令扩展，但它不应当实现现有的接口。另外，在新的实现类中，不要调用超类的 performExecute 方法。

当从您正在替换的这个命令扩展可能看起来违反直觉时，采用此途径的原因是出于对 WebSphere Commerce 将来版本的支持。此途径保护您的代码不会遭到 WebSphere Commerce 将来版本中对命令接口的可能更改。

例如，假定您前面希望替换 OrderNotifyCmdImpl 任务命令的业务逻辑。在此情况下，您将创建名为 CustomizedOrderNotifyCmdImpl 的新任务命令。此命令扩展 OrderNotifyCmdImpl。在新的 CustomizedOrderNotifyCmdImpl 中，您创建新的业务逻辑，但不从超类调用 performExecute 方法。而如果 WebSphere Commerce 将

来版本在接口中引入名为 newMethod 的新方法，则相应版本的 OrderNotifyCmdImpl 命令将包含 newMethod 方法的缺省实现。那么由于您的新命令从 OrderNotifyCmdImpl 扩展，因此编译程序将在 OrderNotifyCmdImpl 命令中找到此新方法的缺省实现，并且您的新命令受到保护防止接口更改。

请参阅 WebSphere Commerce 联机帮助的“参考”部分，以确保新的实现类提供与现有类相同的外部特性。

数据 bean 定制

通常数据 bean 扩展访问 bean。访问 bean（它可由 VisualAge for Java 生成）提供了访问实体 bean 信息的简单方法。对实体 bean 做出修改后（例如，添加新字段、新业务方法或新的查找函数），一旦访问 bean 重新生成，更新就会在访问 bean 中反映出来。由于数据 bean 扩展访问 bean，因此它自动地继承新的属性。由于这种关系，使数据 bean 使用实体 bean 的新属性时无需编码。

如果需要将新属性添加到不是从实体 bean 派生出来的数据 bean 中，您可以使用 Java 继承来扩展现有的数据 bean。例如，如果想要向 OrderDataBean 添加新字段，请如下定义 MyOrderDataBean:

```
public class MyOrderDataBean extends OrderDataBean
{
     public String myNewField () {
     //  implement the new field here
     }

}
```

新数据 bean 也必须具有 BeanInfo 类。以下是此类的声明的样本:

```
public class MyOrderDataBeanInfo extends java.beans.SimpleBeanInfo
{

}
```

VisualAge for Java 提供让您能够生成此 BeanInfo 类的工具。

第 7 章 贸易协议和业务策略（商务版）

本章仅适用于 WebSphere Commerce 商务版。

简介

B2B（商家到商家）商业的关键元素之一是关系管理。贸易协议用于管理买方与卖方组织之间的业务关系。WebSphere Commerce 商务版使用的贸易协议模型支持不同类型的贸易协议，例如合同和 RFQ（报价请求）。

贸易协议的主要元素是一系列的条款和条件。每个条款和条件定义在贸易中使用的特定业务规则。在使用 WebSphere Commerce 商务版时，可以使用 RFQ 在线处理或脱机协商来协商条款和条件集，然后使用 WebSphere 贸易加速器中的业务关系管理接口。

有几种方法可以定制条款和条件模型:

- 选择预定义的业务策略之一（例如标价和退货策略）的条款和条件。或它可以选择您已创建的业务策略。条款和条件对象也可以引用多个业务策略对象。
- 将特定调价应用于业务策略的条款和条件，例如对标准定价的调价。
- 条款和条件，它用于定义管理业务过程的参数集。例如，它可以指定特定的实现中心由指定的合同使用。

合同由一系列的条款和条件组成。这显示在下图中。

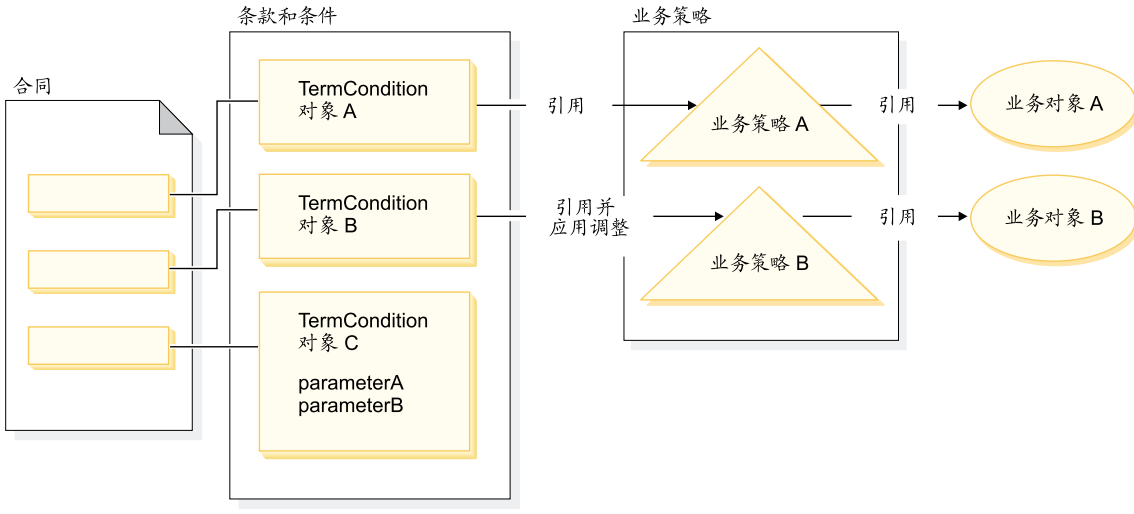


图 29.

在上图中，注意以下方面:

- 术语“调整”指的是对业务策略的修改。例如，它可用于向业务策略的结果应用折扣，这样向标准价格应用 10% 的折扣。它还可用于利用一组参数影响业务策略。
- 例如，TermCondition 对象 A 可以代表装运条款和条件对象。在此情况下，业务策略 A 可以代表装运方式业务策略，而业务对象 A 代表装运递送者 XYZ 的装运方式“A3”。
- 作为另一个示例，TermCondition 对象 B 可以代表价格条款和条件对象，它应用业务策略 B 所定义的价格的 50% 折扣。在此情况下，业务策略 B 是价格策略，而业务对象 B 是为主产品目录定义贸易状态的贸易状态容器。

本章为程序员提供了关于如何创建新业务策略以及新条款和条件的指导方针。

“多乐五金店”样本商店演示在它业务流程中的装运条款和条件对象以及价格条款和条件对象。关于支持这些示例的合同数据的更多信息，请参阅第 138 页的『“多乐五金店”样本合同数据』。

业务策略对象和命令

业务策略对象包含以下信息:

- 策略标识
  业务策略对象的主键。

- 策略类型
  定义了业务策略类型。价格和产品集都是策略类型的示例。
- 策略名称
  每个业务策略必须具有唯一的名称。
- 商店实体
  在其中部署业务策略的商店或商店组。
- 属性
  一组可以传递到业务策略命令的缺省属性。与业务策略对象关联的命令存储在 BusinessPolicyCmd 表中。
- 有效期
  业务策略对象有效的时间段。
- 业务策略命令
  实现业务策略的零个或多个业务策略命令。业务策略命令通常由业务过程调用，并且可以是任务命令或控制器命令二者之一。例如，`getContractPrice()` 命令获取价格条款和条件。此价格条款和条件引用一个特定的价格策略命令，而此价格策略命令用于计算价格。

多个业务策略命令可以与单个业务策略对象关联。每个业务策略命令必须实现业务策略类型对象定义的同一个接口。新业务策略命令的结构如下图所描绘:

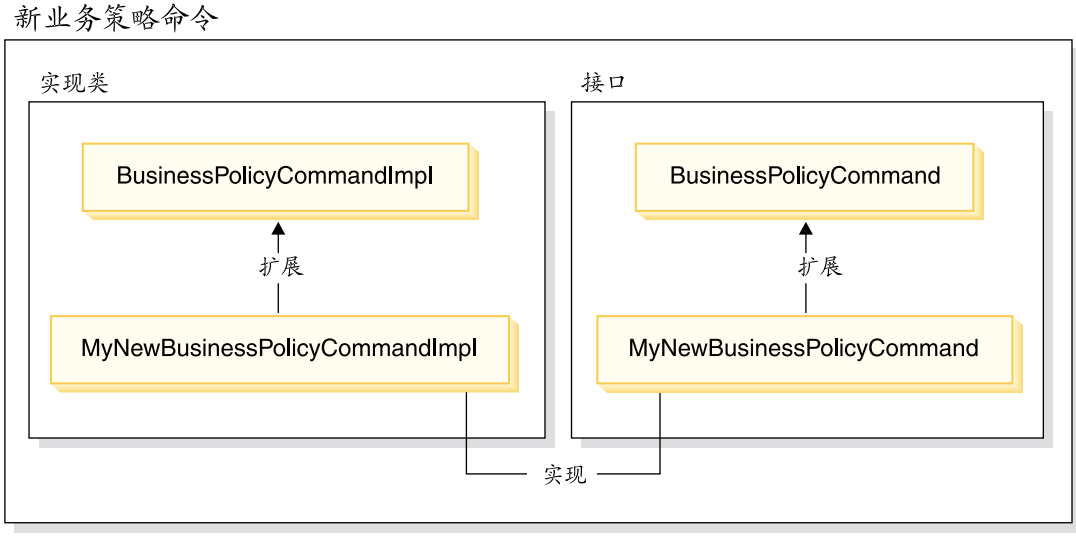


图 30.

如上图中所示，为了创建新的业务策略命令，需要创建新的实现类，它扩展 WebSphere Commerce BusinessPolicyCmdImpl 实现类。还要创建新的接口，它扩展 BusinessPolicyCmd 接口。

“多乐五金店”样本合同数据

本部分提供关于“多乐五金店”样本商店中所用的一些合同数据的介绍。

以下部分中的样本数据按数据库表组织。仅显示相关的行和列。还要注意，当安装了样本时，任何唯一标识（例如 CONTRACT_ID）与此处所显示的相比较可能具有不同的值。

CONTRACT 表样本数据

下表显示“多乐五金店”CONTRACT 数据库表中的相关样本数据。注意: 出于显示目的，数据库列标题显示在第一列中，而该数据库表中的样本数据行显示在第二列中。

列名	样本数据
CONTRACT_ID	10007
MAJORVERSION	1
MINORVERSION	0
NAME	ToolTechContractNumber 4567
MEMBER_ID	-2001
ORIGIN	0
STATE	3
USAGE	1
MARKFORDELETE	0

TERMCOND 表样本数据

下表显示“多乐五金店”TERMCOND 数据库表中的相关样本数据。注意: 出于显示目的，数据库列标题显示在第一列中，而该表中的样本数据行显示在第二和第三列中。

列名	样本数据行 1	样本数据行 2
TERMCOND_ID	10025	10030
TCSUBTYPE_ID	PriceTCPriceListWithSelectiveAdjustment	ShippingTCShippingMode
TRADING_ID	10007	10007

列名	样本数据行 1	样本数据行 2
STRINGFILED1	ProductSet2	
INTEGERFIELD2	10002	
INTEGERFIELD3	1	
BIGINTFIELD1	10051	
FLOATFIELD1	-50.0	
SEQUENCE	1	6

POLICYTC 表样本数据

下表显示“多乐五金店”POLICYTC 数据库表中的相关样本数据。本表建立策略与条款和条件对象之间的关系。

	列名	
	POLICY_ID	TERMCOND_ID
样本数据行 1	10053	10025
样本数据行 2	10056	10030

POLICY 表样本数据

下表显示“多乐五金店”POLICY 数据库表中的相关样本数据。

列名	样本数据行 1	样本数据行 2
POLICY_ID	10053	10056
POLICYNAME	MasterCatalogPriceList	A3
POLICYTYPE_ID	Price	ShippingMode
STOREENT_ID	10051	10051
PROPERTIES	n a m e = T o o l T e c h & member_id=-2001	shippingMode=A3
STARTTIME	空	空
ENDTIME	空	空

TRADEPOSCN 表样本数据

下表显示“多乐五金店”TRADEPOSCN 数据库表中的相关样本数据。

	列名			
	READEPOSCN_ID	**MEMBER_ID**	**NAME**	**TYPE**
样本数据行	10051	-2001	ToolTech	S

SHIPMODE 表样本数据

下表显示“多乐五金店”SHIPMODE 数据库表中的相关样本数据。

	列名			
	SHIPMODE_ID	**STOREENTITY_ID**	**CODE**	**CARRIER**
样本数据行	10053	10051	A3	XYZ 递送者

扩展现有的合同模型

合同可以由一个或多个条款和条件对象组成，其中每个这样的对象都引用一个策略。这样，接下来的各部分描述了创建新业务策略并将它集成到业务流程中的必要步骤。

作为简要的概述，以下是执行此任务的高级步骤:

1. 创建新的业务策略。
   以下任务与创建新的业务策略命令相关:
   a. 创建新的业务策略类型（如果要求）。
      若干业务策略类型已经提供，但如果标准的类型不适合您的业务需求，则请创建新的业务策略类型。
   b. 创建新的业务策略命令。
   c. 注册新的业务策略和业务策略命令。
2. 将条款和条件对象与新的业务策略建立关系。
   这可通过将现有的条款和条件对象与新的业务策略建立关系来完成，或通过创建新的条款和条件对象来完成。如果创建新的条款和条件对象，则您必须执行以下步骤:
   a. 在数据库中注册新的条款和条件
   b. 在合同 DTD（文档类型定义）中注册新的条款和条件
   c. 为该条款和条件创建新的 CMP 企业 bean

d. 更新 WebSphere 贸易加速器以反映新的条款和条件

3. 在业务流程中调用新的业务策略。

创建新的业务策略

创建新的业务策略通常包括在数据库中注册唯一的业务策略，以及创建新的业务策略命令。

创建新业务策略命令涉及到以下高级别步骤:

1. 创建新的业务策略类型（如果要求）。
2. 写新业务策略命令。
3. 在数据库中注册新业务策略和业务策略命令。

上面每个步骤的详细信息在后面的各部分中描述。

创建新业务策略类型

本部分描述如何创建新业务策略类型。业务策略类型指示应用策略的交易域。业务策略类型的示例包括:

- Price
- ProductSet
- ShippingMode
- ShippingCharge
- Payment
- ReturnCharge
- ReturnApproval
- ReturnPayment
- InvoiceFormat

如果现有的业务策略类型不能满足您的业务需求，则应当创建新的业务策略类型。创建新业务策略类型由定义和注册业务策略类型组成。

定义和注册新策略类型时，必须更新以下数据库表:

- POLICYTYPE
- PLCYTYCMIF
- PLCYTYPDSC

POLICYTYPE 表指定正在创建的业务策略类型。它包含一个单独的列，POLICYTYPE_ID，它是主键。它的值例如 Price。如果创建新业务策略类型，请确保指定了唯一的 POLICYTYPE_ID。

PLCYTYCMIF 表是业务策略类型到命令接口关系指定表。也就是说，对于每个业务策略类型，它为该业务策略对象指定 Java 命令接口。虽然可以有零个或多个业务策略命令来实现业务策略，但每个业务策略命令都必须实现这里指定的接口。

PLCYTYPDSC 表指定业务策略类型的描述。它包括描述的语言标识以及业务策略类型的描述。

要创建新业务策略类型，请为该新业务策略类型在每个这些表中创建一条目。以下 SQL 语句提供了一个示例:

```
insert into POLICYTYPE (POLICYTYPE_ID) values ('MyNewPolicyType');
insert into PLCYTYCMIF (POLICYTYPE_ID, BUSINESSCMDIF)
   values ('MyNewPolicyType',
   'com.mycompany.mybusinesspolicycommands.MyNewPolicy');
insert into  PLCYTYPDSC (POLICYTYPE_ID, LANGUAGE_ID, DESCRIPTION)
   values ('MyNewPolicyType', -1,
   'My new policy type for example purposes.');
```

作为创建新的业务策略类型的最终步骤，可以编码一个或多个新的业务策略类型接口。然后，任何归入此业务策略类型范围的业务策略命令实现这些接口。例如，在“多乐五金店”样本商店中，“价格”定义为业务策略类型。同样，有 com.ibm.commerce.price.commands.ResolvePriceListsCmd 和 com.ibm.commerce.price.commands.RetrievePricesCmd 接口，这些接口由所有与价格相关的业务策略命令实现。

如果将没有在新的业务策略类型上执行操作的业务策略命令，则不必创建新的接口。这很少见，而且在大多数情况下当创建新的业务策略类型时，也必须创建新的业务策略类型接口。

当创建业务策略类型接口时，该新接口必须扩展 com.ibm.commerce.command.BusinessPolicyCommand 接口。

写新业务策略命令

要创建新的业务策略命令，必须创建新的命令。该新命令实现与之相关的业务策略类型的接口。该新命令也必须扩展 com.ibm.commerce.command.BusinessPolicyCommandImpl 实现类。这与创建新控制器或任务命令非常类似。

有两种不同的方法可用将输入属性传递到业务策略命令。第一种方法是在 POLICY 表的 PROPERTIES 列中指定缺省输入属性。关于此表的更多信息，请参阅以下部分。

第二种方法是为每个输入属性在命令中创建新的字段。为每个字段创建新的获取函数和设置函数方法对。

在业务策略命令中设置 requestProperties

有两种方法可在业务策略命令对象中设置 requestProperties。第一种方法使用 POLICY 表的 PROPERTIES 列来设置缺省属性。这是由 setRequestProperties 方法完成的。设置属性的第二种方法是让调用业务策略命令的命令（控制器或任务命令）显式设置其它必需属性。

当创建新业务策略命令时，您应重设缺省的 setRequestProperties 方法，以包含适当的逻辑来显式设置 requestProperties 对象中包含的每个参数。

考虑具有接口名称 MyNewBusinessPolicyCmd 和实现类名 MyNewBusinessPolicyCmdImpl 的新业务策略命令这一示例。

假设 POLICY 表中用于此新业务策略命令的条目在 PROPERTIES 列中包含以下值:

- defaultProperty1=apple
- defaultProperty2=orange
- defaultProperty3=banana

此新业务策略命令的接口定义如下:

```
public interface MyNewBusinessPolicyCmd extends
   com.ibm.commerce.command.BusinessPolicyCmd {
     java.lang.String defaultCommandClassName =
         'com.mycompany.mycommands.MyNewBusinessPolicyCmdImpl';
      public void setProperty1();
      public void setProperty2();
}
```

此新业务策略命令的实现类定义如下:

```
public class MyNewBusinessPolicyCmdImpl extends
   com.ibm.commerce.command.BusinessPolicyCmdImpl
   implements com.mycompany.mycommands.MyNewBusinessPolicyCmd {
      // Establish default properties that are stored in the POLICY table

      private java.lang.String defaultProperty1;
      private java.lang.String defaultProperty2;
      private java.lang.String defaultProperty3;
      // Begin to establish properties that must be set
     // by the calling command.
```

```
      //  *** property1   ***
      private java.lang.String property1;
      public java.lang.String getProperty1() {
         return property1;
         }
      public void setProperty1(java.lang.String newProperty1) {
         property1 = newProperty1;
      }

      //  *** property2   ***
      private java.lang.String property2;
      public java.lang.String getProperty2() {
         return property2;
         }
      public void setProperty1(java.lang.String newProperty2) {
         property2 = newProperty2;
         }

      // End establishing properties that must be set
      // by the calling command.
      /* Upon instantiation the business policy command sets all
         default properties from the POLICY table into the
         requestProperties object.  The calling command
         is responsible for setting any other required properties.
      */

      public void setRequestProperties(com.ibm.commerce.datatype.TypedProperty
         requestProperties) {
         //  Get the default properties defined in the POLICY table
         setDefaultProperty1(requestProperties.get("defaultProperty1"));
         setDefaultProperty2(requestProperties.get("defaultProperty2"));
         setDefaultProperty3(requestProperties.get("defaultProperty3"));
         }
}
```

调用新业务策略命令的命令可以与以下类似的方式定义:

```
public class MyCallerCommandImpl
   extends com.ibm.commerce.command.TaskCommandImpl
   implements com.mycompany.mycommands.MyCallerCommand {

   /* Include all elements and processing required for the
      task command.
   */

   // Determine the policy ID and setPolicyId

   // Call the business policy command.

   cmd = (MyNewBusinessPolicyCmd) CommandFactory
            createPolicyCommand(policyId);

   // Set required properties
```

```
    cmd.setProperty1("Fruit salad");
    cmd.setProperty2("Favorite food");

cmd.execute();
}
```

注册新业务策略和业务策略命令

创建新业务策略命令后，业务策略和业务策略命令都必须在数据库中注册。

业务策略注册在 POLICY 表中。此表包含以下列:

- POLICY_ID
  主键。是策略的标识。
- POLICYNAME
  唯一的策略名称。
- POLICYTYPE_ID
  策略类型标识。是 POLICYTYPE 表的外键。
- STOREENT_ID
  应用策略的商店或商店组。
- PROPERTIES
  可以设置为业务策略命令的缺省属性。指定为“名称值”对，例如，`parm1=val1&parm2=val2`。
- STARTDATE
  策略的开始日期（指定为时间戳记）。如果是空，则开始日期为即日。
- ENDDATE
  策略的结束日期（指定为时间戳记）。如果是空，则没有结束日期。

一旦新策略在 POLICY 表中注册后，必须注册策略和实现业务策略的业务策略命令间的关系。POLICYCMD 表用于此目的。POLICYCMD 表包含以下列:

- POLICY_ID
  对 POLICY 表的外键引用。
- BUSINESSCMDCLASS
  实现策略的业务策略命令。
- PROPERTIES
  可以设置为业务策略命令的缺省属性。指定为“名称值”对，例如，`parm1=val1&parm2=val2`。

将条款和条件对象与新的业务策略建立关系

在 WebSphere Commerce 合同和策略框架中，条款和条件（也称作条款）提供了一种描述买方和卖方之间协议的方法。条款和条件可用于各种类型的贸易协议，例如合同和 RFQ（引用请求）。条款和条件对象通常进行可选的调整后引用业务策略。例如，价格条款是通过选择其中一个价格策略对象创建的。在价格条款中，财务经理可以对商店标准价格进行调整，例如:

- 在标准报价上的折扣百分比
- 在一组特定产品上的折扣百分比

每个调价都特定于条款和条件。

当创建新的业务策略时，如果此策略要用于合同中，则必须至少存在一个引用此业务策略的条款和条件对象。您可以将现有的条款和条件对象与新的业务策略建立关系（这是通过在 B2BTrading.dtd 文件中捕获现有条款和条件对象与新业务策略之间的关系来完成的），或者可以创建引用该新业务策略的新的条款和条件对象。

创建新的条款和条件

在 WebSphere Commerce 体系结构中，新的条款和条件对象是通过执行以下步骤来创建的:

1. 更新数据库模式以包含新的条款和条件。
2. 更新 B2BTrading.dtd 文件以反映新的条款和条件。
3. 为条款和条件创建新的企业 bean。
4. 更新 WebSphere 贸易加速器以反映新的条款和条件，或使用合同装入命令来创建使用新条款和条件的新合同。

在以下部分中，MyTC 的示例是新的条款和条件对象。

在数据库中注册新的条款和条件

当正在创建新的条款和条件对象时，您必须更新数据库模式以包含此对象。必须更新的数据库表是 TCTYPE 和 TCSUBTYPE。

以下 SQL 语句显示了如何更新模式的示例:

```
insert into TCTYPE (TCTYPE_ID) values ('MyTC');
insert into TCSUBTYPE (TCSUBTYPE_ID, TCTYPE_ID, ACCESSBEANNAME, DEPLOYCOMMAND)
values ('MySubTC, 'MyTC ', 'com.ibm.commerce.contract.objects.MySubTCAccessBean',
'packagename.MySubTCDeployCmd');
```

在合同文档类型定义中注册新的条款和条件

B2BTrading.dtd 是指定了可在业务策略中使用的不同条款和条件的文档类型定义（DTD）文件。要让新的条款和条件在合同中可用，您必须更新此文件以包含新的条款和条件。

当创建了新的条款和条件时，您必须将新的条款和条件添加到 TermCondition 定义并创建描述该条款和条件的新元素。

要更新 B2BTrading.dtd 文件，请执行以下操作:

1. 导航到以下目录:
   - Windows *drive*:\WebSphere\CommerceServer\xml\trading
   - AIX /usr/WebSphere/CommerceServer/xml/trading
   - Solaris /opt/WebSphere/CommerceServer/xml/trading
   - Linux /opt/WebSphere/CommerceServer/xml/trading
   - 400 /QIBM/ProdData/WebCommerce/xml/trading
2. 打开 B2BTrading.dtd 文件。
3. 使用新的条款和条件更新 TermCondition 定义。例如，该更新以粗体显示在以下 TermCondition 定义中:

   ```
   <!ELEMENT TermCondition (TermConditionDescription?,Participant*,
   CreateTime?,UpdateTime?,(PriceTC|ProductSetTC|ShippingTC|FulfillmentTC|
   PaymentTC|ReturnTC|InvoiceTC|RightToBuyTC|ObligationToBuyTC|
   PurchaseOrderTC|OrderApprovalTC|DisplayCustomizationTC|
   OrderTC|MyTC))>
   ```

   注意，断行仅为了显示目的。
4. 现在将新元素添加到 B2BTrading.dtd 文件。例如，以下显示了添加 MyTC 元素的更新，该元素引用业务策略并有两个必需的属性。

   ```
   <!ELEMENT MyTC (MySubTC)>
   <!ELEMENT MySubTC (PolicyReference)>
   <!ATTLIST MySubTC
      attr1 CDATA #REQUIRED
      attr2 CDATA #REQUIRED
   >
   ```
5. 保存该文件。

为条款和条件创建新的 CMP 企业 bean

必须为条款和条件对象创建新的 CMP 企业 Bean。该 bean 是为条款和条件子类型创建的。

请注意，具有代表性地，当创建新的企业 Bean 时，将 bean 放置到您自己的 EJB 组，而不是将它们包括在包含 WebSphere Commerce 实体 bean 的 EJB 组中的一个。然而在此案例中，因为所有条款和条件的新实体 bean 必须继承 WebSphere Commerce TermCondition bean，所以必须将新条款和条件 bean 放置到 “WCS 合同” EJB 组。

要为条款和条件对象创建新的 CMP 企业 Bean，请遵循以下 VisualAge for Java 中的操作:

1. 执行以下操作，使用向导创建新企业 Bean:
   a. 在 VisualAge for Java 工作台中，选择 EJB 选项卡。
   b. 突出显示，然后用鼠标右键单击 **WCS 合同** EJB 组并选择**添加 > 带继承的企业 Bean**。
      “带继承创建企业 Bean” 智能向导打开。
   c. 在智能向导中，输入适合 bean 的信息。例如，下表显示示例值。

属性	值
Bean 名称	MySubTC
继承自	TermCondition
数据包	com.ibm.commerce.contract.objects
Bean 类	MySubTCBean
远程接口	MySubTC
主接口	MySubTCHome

   d. 单击**添加**将 CMP 字段添加到 bean，并按要求为 bean 创建新字段。对于此示例，使用以下信息创建两个新 CMP 字段:

属性	值
字段名	attr1
字段类型	String
使用获取函数和设置函数方法进行访问	启用
将获取函数和设置函数方法提升到远程接口	启用

属性	值
字段名	attr2
字段类型	Integer
使用获取函数和设置函数方法进行访问	启用
将获取函数和设置函数方法提升到远程接口	启用

e. 单击**完成**。

2. 下一步是将字段从新 bean 映射到 TERMCOND 表中的列。要创建此映射信息，请执行以下操作:

   a. 从 **EJB** 菜单中选择**打开到 > 模式映射**。
   “映射浏览器”打开。

   b. 在映射浏览器的“数据仓库映射”面板中，双击 **WCS 合同**。

   c. 在“持久类”面板中，双击 **TermCondition**，然后选择 **MySubTC**。

   d. 在“表映射”菜单中，选择**新建表映射 > 添加单一继承表映射**。
   “单一继承表映射编辑器”打开。

   e. 在**鉴别器值**字段中，输入 TCSUBTYPE_ID 值。例如，在此情况下，输入 'MySubTC'（包含引号）并单击**确定**。

   f. 请确保在“持久类”面板中仍然选择 **MySubTC**。在“表映射”面板中，突出显示并用鼠标右键单击 TERMCOND 表。选择“编辑属性映射”。
   “属性映射编辑器”打开。

   g. 在“属性映射编辑器”中，如下设置属性:

类属性	映射类型	表列
attr1	简单	STRINGFIELD2
attr2	简单	INTEGERFIELD1

   并单击**确定**。

   h. 从“数据仓库映射”菜单中，选择“保存数据仓库映射”。当保存映射时，输入以下信息:

属性	值
项目	IBM WCS Enterprise Beans
数据包	WCSContract EJB Reserved
类名	WCSContractMap

   单击**完成**然后关闭“映射浏览器”。

3. 在新企业 bean（即在 MySubTCBean 中），如下创建新 `ejbCreate(java.lang.Long argTradingId, org.w3c.dom.Element argElement)` 方法:

```
public void ejbCreate(java.lang.Long argTradingId,
   org.w3c.dom.Element argElement)
   throws javax.ejb.CreateException, javax.ejb.FinderException,
   java.rmi.RemoteException, javax.naming.NamingException,
   javax.ejb.RemoveException {
```

```
    _initLinks();
    super.ejbCreate (argTradingId, argElement);
    this.attr1= null;
    this.attr2= null;
}
```

4. 如下创建新 ejbPostCreate(java.lang.Long argTradingId, org.w3c.dom.Element argElement) 方法:

```
public void ejbPostCreate(java.lang.Long argTradingId,
   org.w3c.dom.Element argElement)
   throws javax.ejb.CreateException, javax.ejb.FinderException,
   java.rmi.RemoteException, javax.naming.NamingException,
   javax.ejb.RemoveException
   {
   parseXMLElement(argElement);
   }
```

5. 如下重设 MySubTCBean 中的 parseXMLElement(org.w3c.dom.Element argElement) 方法:

```
public void parseXMLElement(org.w3c.dom.Element argElement) throws
   javax.ejb.CreateException,
   javax.ejb.FinderException,
   java.rmi.RemoteException,
   javax.naming.NamingException,
   javax.ejb.RemoveException
   {
      super.parseXMLElement(argElement);
      if (argElement == null)
     return;

      String nodeName = argElement.getNodeName();
      if (nodeName.equals("TCCopy"))
     return;
      // get element "MyTC"
      Element eMyTC = ContractUtil.getElementByTag(argElement,"MyTC");

      // get element "MySubTC" from element "MyTC"
      Element eMySubTC = ContractUtil.getElementByTag(eMyTC ,"MySubTC");
      this.attr1 = eMySubTC .getAttribute("attr1").trim();
      this.attr2 = new Integer (eMySubTC .getAttribute("attr2").trim());

      // get element "PolicyReference" from "MySubTC"
      Element ePolicyReference = ContractUtil.getElementByTag(eMySubTC,
         "PolicyReference");
      parseElementPolicyReference(ePolicyReference);

}
```

6. 如下重设 MySubTCBean 中的 createNewVersion(Long argNewTradingId) 方法:

```
public Long createNewVersion(Long argNewTradingId) throws
   javax.ejb.CreateException,
   javax.ejb.FinderException,
```

```
   java.rmi.RemoteException,
   javax.naming.NamingException,
   javax.ejb.RemoveException,
   org.xml.sax.SAXException,
   java.io.IOException

   {

   // Contract a seqElement since tcSequence can not be null
   Element seqElement = ContractUtil.getSeqElementFromTCSequence(
      this.tcSequence);
   MySubTCAccessBean newTC = new MySubTCAccessBean(argNewTradingId,
     seqElement);
   Long newTCId = newTC.getReferenceNumberInEJBType();
   newTC.setInitKey_referenceNumber(newTCId.toString());
   newTC.setMandatoryFlag(this.mandatoryFlag);
   newTC.setChangeableFlag(this.changeableFlag);
   // set columns for this specific TC
   newTC.setAttr1(this.attr1);
   newTC.setAttr2(this.attr2);
   newTC.commitCopyHelper();
   return newTCId;
}
```

7. 如下重设 MySubTCBean 中的 getXMLString() 方法:

```
public String getXMLString() throws
   javax.ejb.CreateException,
   javax.ejb.FinderException,
   java.rmi.RemoteException,
   javax.naming.NamingException
   {
      String xmlTC = "      <MyTC>"  +
                     "%XML_POLICYREFERENCE%" +
                     "         <MySubTC attr1=\"" + this.attr1 +
                     "\"  attr2=\"" + this.attr2.toString() + "\"/>"
                     "         '>" +
                     "      <MYTC></MYTC>" ;
      String xmlPolicy = getXMLStringForElementPolicyReference(
         "ProductSet") ;
      xmlTC = ContractUtil.replace( xmlTC, "%XML_POLICYREFERENCE%",
         xmlPolicy );

      return xmlTC;
}
```

8. 如下重设 MySubTCBean 中的 markForDelete() 方法:

```
public void markForDelete() throws
   javax.ejb.CreateException,
   javax.ejb.FinderException,
   java.rmi.RemoteException,
   javax.naming.NamingException
   {
      // code: remove entries from associated tables which
      // cannot be deleted though delete cascade
}
```

9. 请确保 ejbCreate 方法已经添加到主接口，并且所有其它修改过的方法已经添加到远程接口。
10. 下一步是执行以下操作为 MySubTC 实体 bean 创建访问 bean:
    a. 用鼠标右键单击 **MySubTC** 实体 bean 并选择**添加 > 访问 Bean**。“创建访问 Bean”智能向导打开。
    b. 确保输入以下信息:

表 1.

属性	值
EJB 组	WCSContract
企业 Bean	MySubTC
访问 Bean 名称	MySubTCAccessBean
访问 Bean 类型	实体 Bean 的复制帮助函数

    并单击**下一步**。
    c. 从**为零参数构造函数选择主方法**下拉列表中，选择 **findByPrimaryKey(TermConditionKey)**
    d. 对于 **initKey_referenceNumber**（在“初始属性”列中），将**转换器**设置为 com.ibm.commerce.base.objects.WCStringConverter 并单击**下一步**。
    e. 对于所有已添加的新字段，请确保已经选择了 **CopyHelper** 并将每个字段的转换器值设置为 com.ibm.commerce.base.objects.WCStringConverter。单击**完成**。
    完成代码生成之后，可以通过切换至**项目**选项卡，展开 **IBM WCS 企业 Bean** 项目然后展开 **com.ibm.commerce.contract.objects** 数据包，来查看新代码。
11. 回到 EJB 选项卡上，用鼠标右键单击 **MySubTC** 企业 bean 并选择**生成部署代码**。
12. 您还必须为父 bean（TermCondition bean）和所有兄弟 bean（ “WCS 合同” EJB 组中名称包含 “TC” 的所有其它 bean）重新生成部署代码。请注意，如果已经添加了新字段或修改了现有 TermCondition bean 的远程接口，则您必须为它本身和其所有子 bean 重新生成访问 bean。
    要重新生成部署代码，请执行以下操作:
    a. 突出显示 TermCondition bean 和所有其它名称中包含 “TC” 的 bean（例如 DisplayCustomizationTC、FulfillmentTC 和 InvoiceTC 就是一些兄弟 bean）。
    b. 突出显示所有这些 bean 后，用鼠标右键单击并选择**生成部署代码**。

13. 下一步是重设 ValidateContractCmd 任务命令中的方法。在此方法中，有三个方法您可能希望重设，以支持新的条款和条件对象。它们是:

    - validateTCType()
      此方法检查什么类型的条款可以在合同中。例如，InvoiceTC 属于帐户，因此它不能在出现在合同中。
    - validateTCOccurrence()
      此方法检查条款的出现。例如，在此方法的缺省实现中，合同必须至少有一个 PriceTC。
    - otherValidateCheck()
      此方法的缺省实现为空。可以添加不属于前面两个方法的附加验证。

14. 如果必须部署条款和条件，您必须创建新的部署命令，并在数据库中注册此命令。如果需要，请执行以下操作:

    a. 在此示例中，新的部署命令接口名为 MySubTCDeployedCmd，而实现类名为 MySubTCDeployedCmdImpl。另外，命令封装在 packagename 数据包中。要注册此命令，请发出以下 SQL 命令:

    ```
    insert into CMDREG (STOREENT_ID, INTERFACENAME, CLASSNAME, TARGET)
    values (0, 'packagename.MySubTCDeployCmd',
    'packagename.MySubTCDeployCmdImpl', 'Local');
    ```

    b. 在 packagename 数据包中，创建新的 MySubTCDeployedCmd 接口。此接口必须扩展 com.ibm.commerce.contract.commands.DeployTCCmd 命令接口。以下描述了新的命令接口:

    ```
    public interface MySubTCDeployCmd extends
       com.ibm.commerce.contract.commands.DeployTCCmd
       {
           // customized code
       }
    ```

    在 DeployTCCmd 中有保护参数 abTC 和名为 getTargetStoreId() 的方法。abTC 的值是 MySubTCAccessBean，getTargetStoreId() 方法将商店的标识符返回到部署的合同。

    c. 在同一个数据包中，创建 MySubTCDeployCmdImpl 实现类。此实现类必须扩展 com.ibm.commerce.contract.commands.DeployTCCmdImpl。以下描述了新的命令实现类:

    ```
    public class MySubTCDeployCmdImpl
       extends com.ibm.commerce.contract.commands.DeployTCCmdImpl
       implements MySubTCDeployCmd
       {
         // customer code
       }
    ```

更新 WebSphere 贸易加速器以使用新的条款和条件

一旦创建了新的条款和条件，您可以更新 WebSphere 贸易加速器，这样它就可以用于创建包含这些新条款和条件的新合同。为此目的更新 WebSphere 贸易加速器包括以下步骤:

1. 为新条款和条件创建新的 JavaScript 文件。为了本部分中该示例的目的，该文件被称为 Extensions.js。
2. 创建新的 JSP 模板，它包含 HTML 部分，用户可以在此部分输入新条款和条件必须的信息。为了本部分中该示例的目的，该文件被称为 ContractMyTC.jsp。
3. 为新条款和条件创建新的数据 bean。为了本部分中该示例的目的，该文件称为 MyTCDataBean。
4. 在 VIEWREG 表中注册新视图。
5. 更新 ContractRB_*locale*.properties 文件，使其包含新的资源。
6. 编辑 ContractNotebook.xml 文件，使其包含新页面。

每个这些步骤在以下部分有更详细描述。

创建新的 JavaScript 文件: 第一步，更新 WebSphere 贸易加速器以使用新条款和条件是为了为这些新条款和条件创建新的 JavaScript 文件。可以参阅以下样本文件作为参考:

-
- Windows *drive*:\WebSphere\CommerceServer\samples\contract\Extensions.js
- Windows *drive*:\WebSphere\CommerceServerDev\samples\contract\Extensions.js
- AIX /usr/WebSphere/CommerceServer/samples/contract/Extensions.js
- Solaris /opt/WebSphere/CommerceServer/samples/contract/Extensions.js
- Linux /opt/WebSphere/CommerceServer/samples/contract/Extensions.js
- 400 /QIBM/ProdData/WebCommerce/samples/contract/Extensions.js

为了使用此样本文件，请将其复制到以下目录:

- Windows *drive*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wctools.war\ javascript\tools\contract
- AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/ javascript/tools/contract

- Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/ javascript/tools/contract
- Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/ javascript/tools/contract
- 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/javascript/tools/contract

在此新文件中，必须创建一 JavaScript 对象存储新条款和条件的数据。这在以下代码片段中演示:

```
function ContractMyTCModel() {

   this.tcReferenceNumber = "";
   this.policyReferenceNumber = "";

   this.attr1 = "";
   this.attr2 = "";

   this.policyList = new Array();
   this.selectedPolicyIndex = "0";
}
```

同样还应当创建一新的 JavaScript 对象，提交新条款和条件。这必须按与对 B2BTrading.dtd 文件扩展所使用的相同的方式完成。这在以下代码片段中演示:

```
function submitMyTC(termsAndConditions) {

   var tcModel = get("ContractMyTCModel");

   if (tcModel != null) {

      var myTC = new Object();
      myTC.MyTC = new Object();
      myTC.MyTC.MySubTC = new Object();
      myTC.MyTC.MySubTC.attr1 = tcModel.attr1;
      myTC.MyTC.MySubTC.attr2 = tcModel.attr2;

      myTC.MyTC.PolicyReference = new Object();
      myTC.MyTC.PolicyReference.policyName =
         tcModel.policyList[tcModel.selectedPolicyIndex].policyName;
      myTC.MyTC.PolicyReference.policyType = "ProductSet";
      myTC.MyTC.PolicyReference.storeIdentity =
         tcModel.policyList[tcModel.selectedPolicyIndex].storeIdentity;
      myTC.MyTC.PolicyReference.Member =
         tcModel.policyList[tcModel.selectedPolicyIndex].member;

      if (tcModel.tcReferenceNumber != "") {
         // Change the term and condition
         myTC.action = "update";
```

```
            myTC.referenceNumber = tcModel.tcReferenceNumber;
        }
        else {
            // Create a new term and condition
            myTC.action = "new";
        }

        termsAndConditions[termsAndConditions.length] = myTC;
    }

    return true;
}
```

创建新的 JSP 模板: 下一步是创建新的 JSP 模板，它包含 HTML 部分，用户可以在此部分输入新条款和条件必须的信息。可以参阅以下样本文件作为参考:

- Windows *drive*:\WebSphere\CommerceServer\samples\contract\ContractMyTC.jsp
- Windows *drive*:\WebSphere\CommerceServerDev\samples\contract\ContractMyTC.jsp
- AIX /usr/WebSphere/CommerceServer/samples/contract/ ContractMyTC.jsp
- Solaris /opt/WebSphere/CommerceServer/samples/contract/ ContractMyTC.jsp
- Linux /opt/WebSphere/CommerceServer/samples/contract/ ContractMyTC.jsp
- 400 /QIBM/ProdData/WebCommerce/samples/contract/ ContractMyTC.jsp

为了使用此样本文件，请将其复制到以下目录:

- Windows *drive*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wctools.war\tools\contract
- AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/tools/contract
- Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/tools/contract
- Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/tools/contract
- 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/tools/contract

以下代码片段演示了可用于 MyTC 的 JSP 模板的 HTML 部分。

```
<!--
//////////////////////////////////////
// HTML SECTION
//////////////////////////////////////
-->

<BODY onLoad="onLoad()" class="content">

   <H1>
   <%= contractsRB.get("MyTCHeading") %>
   </H1>

<FORM NAME="MyTCForm">

   <%= contractsRB.get("MyTCAttr1Label") %>
   <BR>
   <INPUT type=text name=Attr1 value="" size=10 maxlength=10>
   <BR>

   <%= contractsRB.get("MyTCAttr2Label") %>
   <BR>
   <INPUT type=text name=Attr2 value="" size=10 maxlength=10>
   <BR>

   <%= contractsRB.get("MyTCPolicyLabel") %>
   <BR>
   <SELECT NAME="PolicyList" SIZE="1">
   </SELECT>

</FORM>
```

创建新的数据 bean: 在此步骤中，创建新的数据 bean，它从 MySubTC 访问 bean 装入必要的数据。代码相应部分在以下代码片段中演示:

```
public class MyTCDataBean extends MySubTCAccessBean
   implements SmartDataBean, Delegator {
      private java.lang.Long contractId;
      private boolean hasMyTC = false;
      private CommandContext iCommandContext;
      /**
       * MyTCDataBean default constructor.
       */
      public MyTCDataBean() {
      }
      /**
       * MyTCDataBean constructor.
       */
      public MyTCDataBean(Long newContractId) {
       contractId = newContractId;
      }


      /*
       * populate the attributes from TermConditionAccessBean
       */
```

```
public void populate() throws Exception {

                Enumeration myTCEnum = new TermConditionAccessBean().
                     findByTradingAndTCSubType(contractId, "MySubTC");
                if (myTCEnum != null) {
                        // assume a contract only has one MyTC for this example

                        setEJBRef(((TermConditionAccessBean)
                              myTCEnum.nextElement()).getEJBRef());
                        refreshCopyHelper();
                      hasMyTC = true;
                  }

}
```

在 VIEWREG 表中注册新视图: 您必须在 VIEWREG 表中注册最新创建的视图。以下是注册新视图的 SQL 语句示例。

```
insert into VIEWREG(VIEWNAME,DEVICEFMT_ID,STOREENT_ID, INTERFACENAME,
   CLASSNAME, PROPERTIES, HTTPS, INTERNAL)
values ('ContractMyTCPanelView', -1, 0,
   'com.ibm.commerce.tools.command.ToolsForwardViewCommand',
   'com.ibm.commerce.tools.command.ToolsForwardViewCommandImpl',
   'docname=tools/contract/ContractMyTC.jsp', 1, 1)
```

更新ContractRB_locale.properties 文件: 必须用特定于新条款和条件的信息更新以下属性文件:

- Windows *drive*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\properties\com\ibm\commerce\tools\contract\properties\ ContractRB_*locale*.properties
- AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/properties/com/ibm/commerce/tools/contract/properties/ ContractRB_*locale*.properties
- Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/properties/com/ibm/commerce/tools/contract/properties/ ContractRB_*locale*.properties
- Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/properties/com/ibm/commerce/tools/contract/properties/ ContractRB_*locale*.properties
- 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/properties/com/ibm/commerce/tools/contract/properties/ ContractRB_*locale*.properties

以下是要添加到该文件的信息示例。

```
MyTCHeading=My TC
attr1Empty=Attribute One must be entered.
attr2Empty=Attribute Two must be entered.
attr1TooLong=Attribute One is too long.
attr2TooLong=Attribute Two is too long.
MyTCAttr1Label=Attribute One (required)
MyTCAttr2Label=Attribute Two (required)
MyTCPolicyLabel=Policy
```

编辑 *ContractNotebook.xml* 文件: 在 WebSphere 贸易加速器中包含新条款和条件的最后一步是更新以下文件，使其包含新页面。

- Windows *drive*:\WebSphere\CommerceServer\xml\tools\contract\ContractNotebook.xml
- AIX /usr/WebSphere/CommerceServer/xml/tools/contract/ContractNotebook.xml
- Solaris /opt/WebSphere/CommerceServer/xml/tools/contract/ContractNotebook.xml
- Linux /opt/WebSphere/CommerceServer/xml/tools/contract/ContractNotebook.xml
- 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/xml/tools/contract/ContractNotebook.xml

以下是此示例中用于包含新页面的代码片段示例。

```
<panel name="MyTCHeading"
    url="ContractMyTCPanelView"
    parameters="contractId,accountId"
    helpKey="MC.contract.MyTCPanel.Help" />
```

导入使用新条款和条件的新合同

更新 WebSphere Commerce 工具以使用新条款和条件的另一个方法是，使用合同导入命令（请参阅 WebSphere Commerce 联机帮助获取此命令的信息）导入包含此新条款和条件的新合同。导入之后，Contract.xml 文件中的相关部分如下显示:

```
<TermCondition>
   <MyTC>
      <MySubTC attr1="adc" attr2="123" />
         <PolicyReference policyName = "Product Set 1"
                          policyType = "ProductSet"
                          storeIdentity = "StoreGroup1" >
            <Member>
               <User distinguishName = "uid=wcsadmin,o=Root Organization"/>
```

```
            </Member>
          </PolicyReference>
        </MyTC>
 </TermCondition>
```

调用新业务策略

一旦创建了新的业务策略，并且此业务策略已经与至少一个条款和条件对象关联，则您必须更新应用程序逻辑以调用新的业务策略命令。

业务策略命令是从控制器和任务命令内调用的。

命令工厂用于调用业务策略命令。有两种创建方法，可以用于调用业务策略命令。第一种方法用于在只有一个业务策略命令与业务策略相关时，调用业务策略命令。这显示在以下代码片段中:

```
CommandFactory createBusinessPolicyCommand(Long policyId);
```

第二种方法用于在有多个业务策略命令与业务策略相关时，调用业务策略命令。这显示在以下代码片段中:

```
CommandFactory createBusinessPolicyCommand(Long policyId, String cmdIfName);
```

在上述示例中，cmdIfName 用于指定要创建的业务策略命令接口名称。

命令工厂在 POLICYCMD 表中查找策略对象，确定实现该策略的命令。它也从表中读取所有缺省属性，并将它们在业务策略命令中设置为 requestProperties。

以下代码片段显示了调用退款策略的示例。

```
RefundPolicyCmd  cmd;

   ////////////////////////////////////////////////////////
   // Get the refund policy id from the refundTC object     //
   //  and use it to create the policy command.             //
   ////////////////////////////////////////////////////////
cmd = (RefundPolicyCmd) CommandFactory
         createPolicyCommand (refundTC.getRefundPolicy);

  cmd.execute()
```

创建合同

将合同模型扩展完全集成到业务过程中的下一步是创建合同来包含引用新业务策略的条款和条件。合同可以使用 WebSphere 贸易加速器创建或通过使用合同 URL 命令之一（ContractImportApprovedVersion 和 ContractImportDraftVersion）创建。关于创建合同的更多信息，请参阅 WebSphere Commerce 联机帮助。

合同定制方案

本部分提供对以下合同定制方案所涉及的步骤的概述:

- 启用折扣

折扣方案

在此示例方案中，创建统一收费率折扣。因为 ToolTech 样本商店既不包含条款和条件也不包含策略类型，所以就必须创建这些。此外，必须创建新业务策略，以及用于存储折扣代码的数据库。

实现此折扣方案包含以下高级步骤:

1. 创建 XREBATECODE 数据库表以及相关 XRebateCodeBean 实体 bean，该实体 bean 用于从此表中访问信息。
2. 通过执行以下子任务创建新 5DollarRebate 业务策略:
   a. 创建相关新业务策略类型。此类性定义新业务策略命令将实现的接口（RebatePolicyCmd）。
   b. 创建新 CalculateRebateCmdImpl 业务策略命令。
   c. 在数据库中注册新业务策略命令和业务策略类型。
3. 通过执行以下高级任务为折扣创建新的条款和条件（RebateTC）:
   a. 在数据库中注册 RebateTC 条款和条件。
   b. 更新 `B2BTrading.dtd` 文件，反映新的 RebateTC。
   c. 为 RebateTC 创建新的企业 Bean。
   d. 更新 WebSphere 贸易加速器，反映新的 RebateTC。
4. 创建使用 RebateTC 的新合同。
5. 将新业务策略集成到购物流程中。

在后面的部分中有上述各步骤更详细的描述。

第一步: 创建新表和企业 Bean

因为现有数据库模式不包含折扣金额和代码的规范，所以必须创建新表。通常，当创建新表时也创建了新实体 bean，该实体 bean 用于访问表中包含的信息。

因此示例起见，假定创建了以下 XREBATECODE 数据库表。

表 2. XREBATECODE 数据库表

	列名		
	REBATECODE_ID	AMOUNT	CURRENCY

表 2. XREBATECODE 数据库表 (续)

样本数据	201	5	CAD
	202	10	CAD

此外，将创建新 CMP 实体 bean（XRebateCodeBean）。请参阅第 53 页的『创建新 CMP 企业 bean』，获取关于创建此 bean 的详细信息。

第二步：创建“5DollarRebate”业务策略

要创建这个新业务策略，必须执行以下步骤:

1. 创建新业务策略类型接口。这是 `CalculateRebateCmdImpl` 将实现的 `RebatePolicyCmd` 接口。
2. 创建新 `CalculateRebateCmdImpl` 业务策略命令。
3. 在数据库中注册新业务策略和业务策略命令。

创建“折扣”业务策略类型: 因为没有现有的业务策略类型与折扣相关，必须创建新的业务策略类型。创建新的业务策略类型包括在数据库中定义和注册策略类型。必须更新以下表:

- POLICYTYPE
- PLCYTYCMIF
- PLCYTYPDSC

对于此方案，要创建新“折扣”类型，会使用以下 SQL 语句:

```
insert into POLICYTYPE (POLICYTYPE_ID) values ('Rebate');
insert into PLCYTYCMIF (POLICYTYPE_ID, BUSINESSCMDIF)
   values ('Rebate',
   'com.mycompany.mybusinesspolicycommands.RebatePolicyCmd');
insert into  PLCYTYPDSC (POLICYTYPE_ID, LANGUAGE_ID, DESCRIPTION)
   values ('Rebate', -1,
   'Rebate policy type.');
```

结果，以下表显示 PLCYTYCMIF 表的相关列，该 PLCYTYCMIF 表显示策略类型和相关业务策略命令之间的关系。

表 3. 更新 PLCYTYCMIF 表

	列名	
	POLICYTYPE_ID	BUSINESSCMDIF
样本数据	Rebate	com.mycompany.mybusinesspolicycommands.RebatePolicyCmd

还必须编码新的 RebatePolicyCmd 接口。此接口必须扩展 com.ibm.commerce.command.BusinessPolicyCommand 接口。按前表建议，将此接口封装到您自己的软件包中。

创建 CalculateRebateCmdImpl 业务策略命令: 要创建新业务策略命令，必须创建名为 CalculateRebateCmdImpl 的新命令，该命令扩展 com.ibm.commerce.command.BusinessPolicyCommandImpl 实现类。此命令应该实现在先前步骤中创建的 RebatePolicyCmd 接口。

请注意在此示例中，接口名称和命令名称是不同的。选择这些名称目的是显示可能有许多实现业务策略折扣类型的业务策略命令。然后，每个实现（即，每个业务策略命令）将以独特的方式实现折扣。

命令逻辑依赖顾客选取货物方式的特殊实现。此外，此 CalculateRebateCmdImpl 应该由应用程序中独立的控制器命令或任务命令调用。

注册新业务策略和新业务策略命令: 必须在数据库中注册新业务策略。还必须注册新业务策略和新业务策略命令之间的关系。

要注册此信息，可以使用 com.ibm.commerce.contract.commands.PolicyAddCmd 命令。以下显示此方案的 PolicyAdd 命令的使用示例:

```
http://localhost:8080/webapp/wcs/stores/servlet/PolicyAdd?
   type=Rebate&name=5DollarRebate&plcyStoreId=-1
   &cmd_1=com.mycompany.mybusinesspolicycommands.CalculateRebateCmdImpl
   &startDate=2002-05-08%2000:00:00&endDate=2003-05-09%2000:00:00
   &commonProps=rebatecode_id%3D501&URL=aRedirectURL
```

请注意，对于输入属性必须用 URL 保留字符的 ASCII 代码替换它们。这样，典型的 =（等于）符号由 “%3D” 替换，&（和号）由 “%26” 替换，空格字符由 “%20” 代替。前述示例中使用的日期格式是 yyyy-mm-dd hh:mm:ss，其中用 ASCII 代码替换 URL 保留字符。

在执行更新之后，下表显示受影响的数据库表的相关列。

表 4. 对 POLICY 表的更新

	列名				
	POLICY_ID	POLICY NAME	POLICYTYPE_ID	STOREENT_ID	PROPERTIES
样本数据	301	5 Dollar Rebate	Rebate	-1	rebatecode_id= 201

请注意，还假设开始日期和结束日期值设置为空。

表 5. 更新 POLICYCMD 表

	列名		
	POLICY _ID	BUSINESS CMDCLASS	PROPERTIES
样本数据	301	com.mycompany. mybusinesspolicycommands. CalculateRebateCmdImpl	空

结果，现在有新的称为“5DollarRebate”的业务策略，该业务策略与 CalculateRebateCmd 业务策略命令相关。

第三步: 创建“RebateTC”条款和条件

创建“RebateTC”条款和条件需要执行以下步骤:

1. 在数据库中注册 RebateTC 条款和条件。
2. 更新 B2BTrading.dtd 文件，反映新的 RebateTC。
3. 为 RebateTC 创建新的企业 bean。
4. 更新 WebSphere 贸易加速器，反映新的 RebateTC。

在数据库中注册“RebateTC”条款和条件: 当正在创建新的条款和条件对象时，您必须更新数据库模式以包含此对象。必须更新的数据库表是 TCTYPE 和 TCSUBTYPE。

以下 SQL 语句显示如何在数据库中注册 RebateTC 的示例:

```
insert into TCTYPE (TCTYPE_ID) values ('RebateTC');
insert into TCSUBTYPE (TCSUBTYPE_ID, TCTYPE_ID, ACCESSBEANNAME, DEPLOYCOMMAND)
values ('RebateTC, 'RebateTC ',
   'com.ibm.commerce.contract.objects.RebateTCAccessBean',
    null);
```

下表显示 TCTYPE 和 TCSUBTYPE 表中相关列的抽取。

表 6. 对 TCTYPE 表的更新

	列名
	TCTYPE_ID
样本数据	RebateTC

表 7. 对 TCSUBTYPE 表的更新

	列名			
	TCSUBTYPE _ID	TCTYPE _ID	ACCESSBEAN NAME	DEPLOY COMMAND
样本数据	RebateTC	RebateTC	com.ibm.commerce. contract.objects. RebateTCAccessBean	空

更新 B2BTrading.dtd 文件，反映新的 RebateTC: 要使新条款和条件在合同中可用，必须更新 B2BTrading.dtd 文件来包含该新条款和条件。当更新此文件时，必须将新条款和条件添加到 TermCondition 定义，然后创建描述该条款和条件的新元素。

粗体文本显示如何将新的 RebateTC 添加到 TermCondition 定义的示例:

```
<!ELEMENT TermCondition (TermConditionDescription?,Participant*,
CreateTime?,UpdateTime?,(PriceTC|ProductSetTC|ShippingTC|FulfillmentTC|
PaymentTC|ReturnTC|InvoiceTC|RightToBuyTC|ObligationToBuyTC|
PurchaseOrderTC|OrderApprovalTC|DisplayCustomizationTC|
OrderTC|RebateTC))>
```

请注意，断行仅用于显示的目的。

然后必须添加将描述此条款和条件的新的节添加到该文件。以下显示关于此 RebateTC 的示例:

```
<!ELEMENT RebateTC (PolicyReference?)>
```

为 RebateTC 创建新的企业 bean: 必须为此新的 RebateTC 创建新的企业 bean。这个新的 bean 应该继承自 WebSphere Commerce TermCondition bean。

条款和条件的新企业 Bean 通常按子类型命名。请注意在此情况下，条款和条件子类型与条款和条件类型相同，这样，bean 的名称与该条款和条件类型相同。

下表显示关于必须创建的新 bean 的一些一般信息。请参阅第 147 页的『为条款和条件创建新的 CMP 企业 bean』，获取更多关于 bean（包括要覆盖的方法）的详细信息。

表 8.

属性	值
Bean 名称	RebateTC
继承自	TermCondition
数据包	com.ibm.commerce.contract.objects

表 8. (续)

属性	值
Bean 类	RebateTCBean
远程接口	RebateTC
主接口	RebateTCHome

更新 WebSphere 贸易加速器，包含 RebateTC: 一旦创建了新的条款和条件，您可以更新 WebSphere 贸易加速器，这样它就可以用于创建包含这些新条款和条件的新合同。请参阅第 154 页的『更新 WebSphere 贸易加速器以使用新的条款和条件』，获取关于如何更新此工具的信息。

第四步: 创建新合同

必须创建包含 “RebateTC” 条款和条件以及引用 “5DollarRebate” 业务策略的新合同。您既可以使用 WebSphere 贸易加速器也可以使用 XML 创建新合同。WebSphere Commerce 联机帮助中描述了每个创建新合同的方法。

在已经创建合同后，下表显示 TERMCOND 和 POLICYTC 数据库表的相关列的更新。

表 9. 对 TERMCOND 表的更新

	列名		
	TRADING _ID	TERMCOND _ID	TCSUBTYPE _ID
样本数据	25	901	RebateTC

表 10. 对 POLICYTC 表的更新

	列名	
	POLICY _ID	TERMCOND_ID
样本数据	301	901

第五步: 将新业务策略集成到购物流程中

在此方案中，假设将新页面添加到商店，该商店允许客户登录以及要求折扣。当客户单击要求折扣时，应调用一个命令，该命令调用新的 RebatePolicyCmd 接口。例如，可能有新的 ClaimRebateCmd 控制器命令调用 RebatePolicyCmd。然后查找到正确的业务策略，并且（在此情况下）应用 “5DollarRebate” 业务策略。

第 3 部分 开发环境

第 8 章 开发工具和部署

本章介绍了用于定制 WebSphere Commerce 应用程序的主要开发工具。它描述了将定制代码从 VisualAge for Java 部署到 WebSphere Commerce Server 的过程。它还描述了如何将代码部署到 Commerce Studio，以便在开发 JSP 模板时可利用已定制的代码。

开发环境

对于创建要与 WebSphere Commerce 商务版一起使用的定制代码，推荐的开发软件包是 WebSphere Commerce Studio，商务开发者版产品。对于创建要与 WebSphere Commerce 专业版一起使用的定制代码，推荐的开发软件包是 WebSphere Commerce Studio 专业开发者版产品。这两个软件包都包含创建定制代码和执行 Web 开发任务所需要的所有工具。

WebSphere Commerce Studio 商务开发者版和 WebSphere Commerce Studio 专业开发者版都提供一个选项来包含在 VisualAge for Java 的 WebSphere Test Environment 组件中使用的样本 WebSphere Commerce 商店。由于不要求开发者使用 WebSphere Commerce 的工具创建商店并然后将商店有用资源移动到开发环境，因此它简化了对开发环境的配置。

开发环境的另一个关键组件是 WebSphere Commerce 代码资源库。本产品安装完成后，必须将该资源库导入到 VisualAge for Java 的工作空间中。导入资源库之后，开发者必须执行一些与 WebSphere Test Environment 相关的配置步骤。

所有安装和配置任务完成后，开发者就有一台独立的开发机器，可以在此机器上创建并测试定制的 WebSphere Commerce 代码。开发者不需要在开发机器上安装 WebSphere Commerce。

WebSphere Commerce Studio

WebSphere Commerce Studio 商务开发者版和 WebSphere Commerce Studio 专业开发者版都包含 VisualAge for Java 企业版版本 4.0。该版本的 VisualAge for Java 包含协助开发商店前台有用资源的功能部件，例如用于开发和调试高级 JSP 模板的强大工具。它还包含改进了的与 WebSphere Studio 的集成性，这允许向这些 JSP 模板快速添加内容，从而增加了程序员和 Web 开发者的多产性。对于后台办公应用程序代码的开发，它包含对 Enterprise JavaBeans 技术的支持，它还包含连接功能以支持对其它系统的集成，例如 CICS® TS、MQSeries、SAP R/3 以及更多系

统。另外，集成的 VisualAge for Java 企业版的 WebSphere Test Environment 允许开发者运行 WebSphere Commerce 功能，而不需要退出 VisualAge for Java。这意味着不用将定制的 WebSphere Commerce 代码部署到 WebSphere Commerce Server 就可以对它进行测试。

注: VisualAge for Java 企业版版本 4.0 的 WebSphere Test Environment 组件在 WebSphere Application Server V3.5.4 中运行应用程序。注意，WebSphere Commerce 商务版和 WebSphere Commerce 专业版都使用 WebSphere Application Server V4.0。版本之间存在差异的结果是，在向运行在 WebSphere Application Server V4.0 中的 WebSphere Commerce 实例部署新的或修改的企业 bean 之前，您必须使用 EJBDeploy 工具（与 WebSphere Application Server V4.0 一起提供）在 VisualAge for Java 之外生成部署代码。实际上，该部署代码必须在安装了 WebSphere Application Server V4.0 的机器上生成。有关更多详细信息，请参阅第 171 页的『有关 EJB 部署代码的信息』。

为了完成第 181 页的第 4 部分，『教程』中描述的教程，您必须安装 WebSphere Commerce Studio 商务开发者版或 WebSphere Commerce Studio 专业开发者版。有关安装 Commerce Studio 和配置 VisualAge for Java 的更多信息，请参阅《*WebSphere Commerce Studio 商务开发者版安装指南*》或《*WebSphere Commerce Studio 专业开发者版安装指南*》。

VisualAge for Java 的功能部件和功能

此《*程序员指南*》并不是用来教您如何使用 VisualAge for Java 的。对于 VisualAge for Java，本书大致描述了如何在该产品中执行任务，而不仅仅将其作为完成创建 WebSphere Commerce 的电子交易应用程序的编程任务的方法。这样，如果您是 VisualAge for Java 的新用户，那么您需要通过执行本书中的教程学会如何在 VisualAge for Java 执行某些任务。然而，要想成为使用此工具的专家，您还必须参阅 VisualAge for Java 文档、教程和课程。

例如，（可选）使用 VisualAge for Java 中的调试器教程主题提供了一个快速介绍以说明在代码执行期间您可以如何使用调试器来查看任务命令中变量的值。VisualAge for Java 的调试器组件是一个强大的工具，因此应当参阅 VisualAge for Java 文档，获取其功能部件和如何使用它们的详细信息。

WebSphere Commerce 代码资源库

为了创建 WebSphere Commerce 应用程序的定制代码，您必须将 WebSphere Commerce 代码的资源库导入到 VisualAge for Java 工作空间。此资源库可以在 WebSphere Commerce 商务版磁盘 2 CD 和 WebSphere Commerce 专业版磁盘 2 CD 上得到。当前的资源库（发布时的当前资源库）名为 `WC_54.dat`。

代码部署

当定制电子交易应用程序时，可以执行以下任意的操作:

- 创建新命令、数据 bean 或实体 bean
- 扩展现有 WebSphere Commerce 实体 bean
- 修改现有命令或数据 bean 的逻辑

在 VisualAge for Java 中开发代码时，可在 WebSphere Test Environment 中测试代码。有时，必须将代码部署到开发环境之外的 WebSphere Commerce Server。

在以下部分中，*目标 WebSphere Commerce Server* 是指您要将定制代码部署到的 WebSphere Commerce Server。在某些测试方案中，可以部署到正运行在与 VisualAge for Java 同一台机器上的 WebSphere Commerce Server。在其它情况下，目标 WebSphere Commerce Server 在另一台机器上，甚至可以运行在不同的平台上。

以下部分描述的是部署各种类型定制代码的高级步骤。通过它们来理解部署过程中的步骤，并且请参阅第 311 页的附录 B, 『部署详细信息』获取逐步的指导。

除以下描述定制代码部署的部分外，如果在您的开发环境中创建了新的访问控制策略，则必须在目标 WebSphere Commerce Server 上创建相同的访问控制策略。关于此过程的示例，请参阅教程中的第 251 页的『装入新资源的访问控制策略』。

有关 EJB 部署代码的信息

认识到 VisualAge for Java V4.0 的 WebSphere Test Environment 组件使用 WebSphere Application Server V3.5.4，这是很重要的。在此版本的 WebSphere Application Server 中，企业 bean 在 Enterprise JavaBeans（EJB）V1.0 规范的级别上受到支持。所以，为了在 WebSphere Test Environment 中运行任何企业 bean，请使用 VisualAge for Java 的工具生成企业 bean 的部署代码（该代码用 EJB V1.0 规范来编译）。

相反，WebSphere Commerce 商务版和 WebSphere Commerce 专业版都使用 WebSphere Application Server V4.0。WebSphere Application Server V4.0 支持 EJB V1.1 规范。所以，在 WebSphere Test Environment 中运行的企业 bean 部署代码不同于在 WebSphere Application Server V4.0 中运行的企业 bean 部署代码。

对开发和部署的影响如下:

- 要在 WebSphere Test Environment 中测试企业 bean，请使用 VisualAge for Java 的工具生成符合 WebSphere Test Environment 中使用的 WebSphere Application Server V3.5.4 需要的部署代码。通过用鼠标右键单击企业 bean 并选择**生成部署代码**来生成此代码。注意，这不会创建 JAR 文件。

- 要将企业 bean 部署到 WebSphere Application Server V4.0，您必须执行以下操作:
  1. 使用 VisualAge for Java 工具将企业 bean 导出到在此文档中被称为 *EJB 1.1 Export JAR 文件*的文件中。此 JAR 文件将代码包裹进由 EJBDeploy 工具使用的格式中。用鼠标右键单击包含要部署的企业 bean 的 EJB 组，并选择 **导出 > EJB 1.1 JAR** 来创建此文件。注意，在创建此 JAR 文件时，您必须为要对其部署代码的机器选择适当的数据库类型。
  2. 将此 EJB 1.1 Export JAR 文件传送到正在运行 WebSphere Application Server V4.0 的目标服务器。
  3. 在目标服务器上，将 EJB 1.1 Export JAR 文件作为输入来运行 EJBDeploy 工具，以创建部署代码（此代码按 Enterprise JavaBeans V1.1 规范编译）的新 JAR 文件。

在企业 bean 的部署过程中还包含其它一些步骤。有关更多信息，请参阅第 173 页的『新实体 bean 的部署』和第 175 页的『已修改的 WebSphere Commerce 公共实体 bean 的部署』。有关使用 EJBDeploy 工具的更多信息，请参阅第 322 页的『生成部署代码』。

新命令和数据 bean 的部署

当创建新命令和数据 bean 时，应将它们放置在根据您的应用程序适当地命名的数据包中。例如，可以创建名为 `com.mycompany.mycommands` 的新数据包，以在其中保存新命令。此软件包必须存储在独立于 WebSphere Commerce 项目（IBM WC Commerce Server 和 IBM WC 企业 Bean）的项目中。例如，可创建名为 `My Project` 的新项目。需要小心组织代码，以确保部署成功。

当所有新命令和数据 bean 存储在您自己的项目中后，部署由以下高级步骤组成:

1. 使用开发机器为项目创建 JAR 文件。从 VisualAge for Java 中的 “项目” 页面使用工具将项目导出至 JAR 文件。相应于您的代码，命名 JAR 文件，例如 `CustomCommands.jar`。此外，必须使用 VisualAge for Java 外部工具重新编制 JAR 文件，以确保包含了所有必需的用于部署至 WebSphere Application Server 的命名信息。关于更多信息，请参阅第 311 页的『用于定制命令和数据 bean 的 JAR 文件』。
2. 将 JAR 文件、JSP 模板和任何其它的商店有用资源复制到目标 WebSphere Commerce Server 上的适当目录中。关于更多信息，请参阅第 318 页的『在目标 WebSphere Commerce Server 上存储有用资源』。
3. 在目标 WebSphere Commerce Server 上的命令注册表中注册新命令。关于更多信息，请参阅第 26 页的『命令注册框架』。

4. 使用 WebSphere Application Server 管理员控制台，停止并重新启动 WebSphere Commerce 企业应用程序。有关启动和停止此应用程序的更多信息，请参阅相应的《*WebSphere Commerce 安装指南*》。

新实体 bean 的部署

当创建新的实体 bean 时，您必须在这样的 EJB 组中创建它们，该 EJB 组独立于包含 WebSphere Commerce 实体 bean 的 EJB 组。您还必须使用自己的项目并确保此项目不包含任何命令或数据 bean 的代码。例如，可创建一个名为 `MyEntityBeans` 的新 EJB 组和名为 `MyEntityBeansProject` 的项目。如果项目中包含了与新实体 bean 的代码不同的代码，部署可能无法成功。

一旦对实体 bean 在 WebSphere Test Environment 中的运行方式感到满意，就必须部署它。以下信息提供了部署步骤的概述:

1. 使用开发机器，为新的 EJB 组创建新的 EJB 1.1 Export JAR 文件。从 VisualAge for Java 中的 EJB 页面，用鼠标右键单击新的 EJB 组并选择导出代码到 EJB 1.1 JAR 文件。相应于您的代码，命名文件，例如 `MyEntityBeans_DT.jar`。关于更多信息，请参阅第 313 页的『为新实体 bean 创建 JAR 文件』。
2. 为新 EJB 项目创建新的实现 JAR 文件。要创建此 JAR 文件，请选择“项目”页面，然后用鼠标右键单击新 EJB 项目并选择将代码导出至 JAR 文件。相应于您的代码，命名文件，例如 `MyEntityBeansImpl.jar`。此外，必须使用 VisualAge for Java 外部工具重新编制实现 JAR 文件，以确保包含了所有必需的用于部署至 WebSphere Application Server 的命名信息。关于更多信息，请参阅第 313 页的『为新实体 bean 创建 JAR 文件』。
3. 将 JAR 文件复制到目标 WebSphere Commerce Server 上的适当目录中。关于更多信息，请参阅第 318 页的『在目标 WebSphere Commerce Server 上存储有用资源』。
4. 在目标 WebSphere Commerce Server 上，使用 WebSphere Application Server 提供的“EJB 部署工具”为新的企业 bean 生成部署代码。此工具将步骤 1 中创建的 JAR 文件作为输入，而且它创建相应的 JAR 文件来包含 EJB 组中所有企业 bean 的部署代码。关于更多信息，请参阅第 322 页的『生成部署代码』。
5. 在目标 WebSphere Commerce Server 上，修改包含在部署代码 JAR 文件中的企业 bean 的交易隔离级别。使用 WebSphere Commerce 提供的 `modifyIsolationLevel` 命令行实用程序将交易隔离级别设置为数据库类型的适当级别。关于更多信息，请参阅第 325 页的『修改实体 bean 的交易隔离级别』。

6. 在目标 WebSphere Commerce Server 上，为您已创建的所有新资源装入访问控制策略。使用 WebSphere Commerce `acpload` 和 `acpnlsload` 命令装入策略信息。有关为新资源装入访问控制策略的示例，请参阅第 9 章，『教程: 创建新的业务逻辑』教程中的第 251 页的『装入新资源的访问控制策略』部分。
7. 在目标 WebSphere Commerce Server 上，从 WebSphere Application Server 中导出当前 WebSphere Commerce 企业应用程序。在导出完成以后，创建一个 .ear 文件，它包含整个应用程序。要导出当前应用程序，请打开 WebSphere Application Server 管理控制台，选择 WebSphere Commerce 企业应用程序，然后选择导出选项。完成时，将创建一个 `WC_Enterprise_App_`*`instanceName`*`.ear` 文件。关于更多信息，请参阅第 326 页的『导出当前 WebSphere Commerce 企业应用程序』。
8. 在目标 WebSphere Commerce Server 上，导出企业 bean 的配置信息，这些企业 bean 包含在运行在 WebSphere Application Server 中的当前 WebSphere Commerce 企业应用程序中。通过 WebSphere Application Server 提供的 `XMLConfig` 命令行实用程序的 `-export` 选项，将此信息导出到 XML 文件中。生成的 XML 文件（此处引用为 `OutputFile.xml` 文件）对包含在企业应用程序中的每个企业 bean 都包含一节配置信息。然后，您必须为正在部署的每个新企业 bean 向此文件中添加一个新节。关于更多信息，请参阅第 327 页的『导出企业 bean 的配置信息』。
9. 使用 WebSphere Application Server 提供的应用程序组装工具将新的企业 bean 添加到企业应用程序中。使用此工具，您可以为当前的应用程序打开 `WC_Enterprise_App_`*`instanceName`*`.ear`，然后将任何新的企业 bean 导入到应用程序中。您还要设置新企业 bean 的类路径以包含任何依赖的 JAR 文件，并将实现 JAR 文件作为一个文件添加到应用程序。最后，您使用此工具为包含在企业 bean 中的方法配置 WebSphere Application Server 安全性。在完成这些步骤之后，保存应用程序，并为您的企业应用程序创建一个新的 `.ear` 文件。关于更多信息，请参阅第 332 页的『将新的企业 bean 组装到企业应用程序中』。
10. 将新的企业应用程序导入到 WebSphere Application Server。此步骤包括以下三个子任务:
    a. 使用 WebSphere Application Server 管理控制台，停止并除去原始的 WebSphere Commerce 企业应用程序。关于更多信息，请参阅第 341 页的『停止并除去企业应用程序』。
    b. 使用 `XMLConfig` 命令行实用程序的 `-import` 选项将新的企业应用程序导入到 WebSphere Application Server。关于更多信息，请参阅第 342 页的『导入企业应用程序』。
    c. 使用 WebSphere Application Server 管理控制台，刷新视图以显示新的企业应用程序，然后启动新的应用程序。关于更多信息，请参阅第 344 页的『启动企业应用程序』。

现有命令和数据 bean 扩展的部署

扩展现有命令的方法取决于所需的修改类型。第 128 页的『现有命令的定制』中解释了扩展的方法。通常，现有逻辑的修改涉及到创建从需要定制的类继承的新类。按照需要覆盖超类的方法，以替换或修改逻辑。

当定制数据 bean 时，还要创建扩展现有数据 bean 的新类。在新类中进行必需的修改。

当创建这些新类时，请确保将它们存储在您自己的一个数据包中，而这个数据包存储在您自己的一个项目中。

由于子类有效处理了扩展，因此对命令和数据 bean 扩展的部署与对新命令和数据 bean 的部署相同。关于更多信息，请参阅第 172 页的『新命令和数据 bean 的部署』。

已修改的 WebSphere Commerce 公共实体 bean 的部署

当修改 WebSphere Commerce 公共企业 bean 时即修改了 WebSphere Commerce 代码。对已定制的实体 bean 的部署技术与用于新实体 bean 的稍有不同。以下内容提供了部署步骤的概述:

1. 为包含已修改实体 bean 的 WebSphere Commerce EJB 组创建一个 EJB 1.1 Export JAR 文件。在选择 VisualAge for Java Workbench 中的 EJB 选项卡的情况下，选择适当的 EJB 组，然后选择将该组导出到 EJB 1.1 Export JAR 文件。当命名 JAR 文件时，您可以使用 `Cust_`*`EJBGroupName`*`-ejb_DT.jar` 命名约定，其中 *`EJBGroupName`* 是已修改的 EJB 组的名称，且 _DT 后缀附加在 JAR 文件名称的结尾。例如，当命名 WCSUser EJB 组的 JAR 文件时，您可以将文件命名为 `Cust_WCSUser-ejb_DT.jar`。注意，使用 _DT 后缀仅是为了提醒您稍后必须将此 JAR 文件传递给 EJBDeploy 工具来为包含在 EJB 组中的 bean 生成部署代码。关于更多信息，请参阅第 315 页的『为定制 WebSphere Commerce 实体 bean 创建 JAR 文件』。
2. 创建新的客户 JAR 文件，它包含所有 WebSphere Commerce EJB 组的客户代码。在选择了所有 WebSphere Commerce EJB 组（名称以 `WCS` 开头的所有组）的情况下，选择导出到客户 JAR 文件。关于更多信息，请参阅第 315 页的『为定制 WebSphere Commerce 实体 bean 创建 JAR 文件』。
3. 将 JAR 文件复制到目标 WebSphere Commerce Server 上的适当目录中。关于更多信息，请参阅第 318 页的『在目标 WebSphere Commerce Server 上存储有用资源』。
4. 在目标 WebSphere Commerce Server 上，使用 WebSphere Application Server 提供的 EJBDeploy 工具为包含在正在部署的 EJB 组中的企业 bean 生成部署

代码。此工具将步骤 1 中创建的 JAR 文件作为输入，而且它创建相应的 JAR 文件来包含 EJB 组中所有企业 bean 的部署代码。关于更多信息，请参阅第 322 页的『生成部署代码』。

5. 在目标 WebSphere Commerce Server 上，修改包含在部署代码 JAR 文件中的企业 bean 的交易隔离级别。使用 WebSphere Commerce 提供的 `modifyIsolationLevel` 命令行实用程序将交易隔离级别设置为数据库类型的适当级别。关于更多信息，请参阅第 325 页的『修改实体 bean 的交易隔离级别』。
6. 在目标 WebSphere Commerce Server 上，从 WebSphere Application Server 中导出当前 WebSphere Commerce 企业应用程序。在导出完成以后，创建一个 .ear 文件，它包含整个应用程序。要导出当前应用程序，请打开 WebSphere Application Server 管理控制台，选择 WebSphere Commerce 企业应用程序，然后选择导出选项。完成时，将创建一个 `WC_Enterprise_App_`*`instanceName`*`.ear` 文件。关于更多信息，请参阅第 326 页的『导出当前 WebSphere Commerce 企业应用程序』。
7. 在目标 WebSphere Commerce Server 上，导出企业 bean 的配置信息，这些企业 bean 包含在运行在 WebSphere Application Server 中的当前 WebSphere Commerce 企业应用程序中。通过 WebSphere Application Server 提供的 `XMLConfig` 命令行实用程序的 `-export` 选项，将此信息导出到 XML 文件中。生成的 XML 文件（此处引用为 `OutputFile.xml` 文件）对包含在企业应用程序中的每个企业 bean 都包含一节配置信息。关于更多信息，请参阅第 327 页的『导出企业 bean 的配置信息』。在此文件中，您必须修改描述企业 bean 的节（您已修改了此企业 bean 以指向新的 `Cust_`*`EJBGroupName`*`-ejb.jar` 文件。
8. 在目标 WebSphere Commerce Server 上，使用应用程序组装工具将修改的 EJB 组集成到企业应用程序中。使用该工具，打开当前应用程序的 `.ear` 文件。一旦打开它，您就能执行以下任务:
   a. 记录您已修改的原始版本 EJB 组的类路径信息。例如，如果您修改了 WCSUser EJB 组中的 User bean，则您将 WCSUser 组的类路径信息复制到一个文本文件中。
   b. 删除已修改的原始版本的 EJB 组（例如，删除 WCSUser 组）。
   c. 导入已修改的新版本的 EJB 组。
   d. 将原始的类路径信息应用到刚导入的 EJB 组。
   e. 为已修改的 EJB 组中所有的企业 bean 中包含的方法配置 WebSphere Application Server 安全性。
   f. 将应用程序保存到新的 `.ear`

   关于更多信息，请参阅第 337 页的『将修改的企业 bean 组装到企业应用程序中』。

9. 将新的企业应用程序导入到 WebSphere Application Server。此步骤包括以下三个子任务:
   a. 使用 WebSphere Application Server 管理控制台，停止并除去原始的 WebSphere Commerce 企业应用程序。关于更多信息，请参阅第 341 页的『停止并除去企业应用程序』。
   b. 使用 XMLConfig 命令行实用程序的 -import 选项将新的企业应用程序导入到 WebSphere Application Server。关于更多信息，请参阅第 342 页的『导入企业应用程序』。
   c. 使用 WebSphere Application Server 管理控制台，刷新视图以显示新的企业应用程序，然后启动新的应用程序。关于更多信息，请参阅第 344 页的『启动企业应用程序』。

部署新数据 bean 以用于 Commerce Studio

如果正在使用 Commerce Studio 来开发 JSP 模板，则必须将新数据 bean 部署到 Commerce Studio。特别的是，您必须为数据 bean 创建 JAR 文件。

通过用鼠标右键单击数据 bean 的数据包并选择将它们导出至 JAR 文件，创建此 JAR 文件。在选项中，选择包含以下内容:

- **类**
- **资源**
- **bean**

此外，选择**选择引用类型和资源**以包含数据 bean 所需的命令和资源。

一旦已经导出 JAR 文件，您必须更新 Commerce Studio 的类路径以包含此 JAR 文件。

当需要修改现有的 WebSphere Commerce 数据 bean 时，您必须创建新的数据 bean，它扩展需要定制的数据 bean。因此，实际上您正在创建新的数据 bean，其部署如本部分中所述。

部署在 Commerce Studio 中使用的定制公共实体 bean

如果修改任何公共实体 bean，并将 Commerce Studio 用于 JSP 模板开发，则必须为所有的公共实体 bean 创建新的客户机 JAR 文件，并修改 Commerce Studio 的类路径以在原始 JAR 文件名之前包含新 JAR 文件名。当在 Page Designer 工具中使用数据 bean 时，Commerce Studio 将使用该客户机 JAR 文件。

要创建此 JAR 文件，请使用 VisualAge for Java 中的工具。从 VisualAge for Java 的 EJB 页面中选择全*部* WebSphere Commerce EJB 组（而不仅仅是修改过的 EJB

组），并选择将它们导出至客户机 JAR 文件。导出完成后，请确保在 Commerce Studio 类路径的开头指定此新的 JAR 文件。

日志文件

在整个产品安装、代码开发和代码部署过程中，都会生成日志文件。本部分列出了一部分在整个过程中您可能需要参考的日志文件。

Commerce Studio 日志文件

Commerce Studio 在安装过程中构建日志文件。要访问这些日志，请打开以下文件:

```
C:\Winnt\WCStudioInstall.log
```

WebSphere Test Environment 的 Commerce Studio 配置日志

如果选择希望执行后期部署任务，则在 WebSphere Commerce Studio 的安装过程期间会执行一定的 WebSphere Test Environment 配置步骤。例如，如果选择包含在 VisualAge for Java 的 WebSphere Test Environment 组件中运行的样本商店，则创建与批量装入过程和数据库创建相关的日志文件。要访问这些日志，请导航到以下文件:

drive:\WebSphere\CommerceServerDev\instances*instance_name*\logs

来自 WebSphere Test Environment 中正在运行的 WebSphere Commerce 组件的日志文件

当在 WebSphere Test Environment 中运行商店或测试单个组件时，可能会生成跟踪文件和消息文件。这些文件的缺省位置是以下目录:

drive:\WebSphere\CommerceServerDev\instances*instance_name*\logs

运行 modifyIsolationLevel 命令的日志

在部署企业 bean 时，您使用 modifyIsolationLevel 命令修改 JAR 文件中每个企业 bean 的交易隔离级别。生成的日志文件存储在运行该命令时指定的日志文件中。关于更多信息，请参阅第 325 页的『修改实体 bean 的交易隔离级别』。

在 VisualAge for Java 中记录

当在 WebSphere Test Environment 中运行代码时，“控制台”窗口作为活动的日志运行。此外，您可以修改 EJB 服务器的跟踪级别，以增加能在“控制台”窗口中找到的信息量。此信息在 EJB 服务器的属性中指定为跟踪级别。

测试支付方式

在 WebSphere Test Environment 中运行的“流行时尚”样本商店缺省情况下使用测试支付方式。该测试支付方法已经附带包含，从而可以在 WebSphere Test Environment 中完成购物流程而不需要调用远程 Payment Manager。使用此支付方式所下的订单具有状态“M”（指示该支付已初始化且正在等待处理）。

此测试支付方式只让您完成购物，它不会使得用此支付方式提交的订单可用于进一步处理。这样，测试支付方式应当只在 WebSphere Test Environment 中使用。

支付相关命令的实现类是业务策略命令类型。因此，选择使用哪个实现类是由业务策略控制的。缺省情况下，WebSphere Test Environment 中运行的“流行时尚”样本商店包含使用支付相关命令的以下实现的业务策略（TestPaymentMethod 策略）：

- com.ibm.commerce.payment.commands.DoPayment**Test**CmdImpl
- com.ibm.commerce.payment.commands.CheckPaymentAccept**Test**CmdImpl
- com.ibm.commerce.payment.commands.DoCancel**Test**CmdImpl
- com.ibm.commerce.payment.commands.DoDeposit**Test**CmdImpl
- com.ibm.commerce.payment.commands.DoRefund**Test**CmdImpl

必须强调的是这些命令是仅为测试结帐过程提供的。提供以上一套命令是为了完整性目的，而 DoDepositTestCmdImpl 是一个命令根，如果调用它，则它抛出一个异常。请勿对这些订单尝试使用任何订单管理功能。如果需要能够测试其它这些功能，您可以配置 WebSphere Test Environment 以使用远程 Payment Manager。

当您将代码部署到目标 WebSphere Commerce Server 时，不应当将与测试支付方式相关的业务策略从开发机器上复制到目标机器上，这一点很重要。另外，您不应当在目标 WebSphere Commerce Server 上注册与测试支付方式相关的命令。

有关禁用测试支付方式的信息，请参阅《*WebSphere Commerce 商务版安装指南*》或《*WebSphere Commerce 专业版安装指南*》中的『配置 VisualAge for Java』章节。

使用远程 Payment Manager

如果您的开发工作包含一旦下了订单，就处理这些订单（例如，对订单管理过程中的步骤做出修改），则您必须配置在 WebSphere Test Environment 中运行的商店以使用远程 Payment Manager。关于详细的配置步骤，请参阅《*WebSphere Commerce Studio 商务开发者版安装指南*》。该文档还提供了关于从数据库除去使用测试支付方式所下的订单的信息。

第 4 部分 教程

本部分包含帮助您熟悉定制电子交易应用程序的教程。提供了以下教程:

- 创建新业务逻辑
  本教程演示了创建新业务逻辑的开发过程。它包括以下子任务:
  - 准备样本项目
  - 写新命令
  - 创建新的数据 bean，并从请求中获取属性
  - 使用实体 bean
  - 创建新的企业 bean
- 更新现有的业务逻辑
  本教程显示了关于更新现有 WebSphere Commerce 业务逻辑的开发过程。它包括以下子任务:
  - 扩展现有的 WebSphere Commerce 控制器命令
  - 修改现有的 WebSphere Commerce 公共实体 bean
  - 创建扩展现有的 WebSphere Commerce 任务命令的新任务命令。

教程包含需要输入 VisualAge for Java 的代码段。建议您从以下 Web 站点获得本文档的 PDF 版本:

Business

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

Professional

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

您可从 PDF 版本的《程序员指南》中剪切并粘贴代码片段。

第 9 章 教程: 创建新的业务逻辑

教程环境

本书中包含的教程需要您安装有 “WebSphere Commerce 商务版, V5.4” 或 “WebSphere Commerce 专业版, V5.4”。此外, 在安装产品时, 必须选择安装以下选项:

- **使用 WebSphere Studio 开发商店前台有用资源**
- **使用 VisualAge for Java 开发商店后端逻辑**
- **创建数据库**
- **包含样本商店**

有关完整的安装和配置信息, 请遵循《*WebSphere Commerce Studio, 商务开发者版安装指南*》或《*WebSphere Commerce Studio 专业开发者版安装指南*》中包含的指导。

开始本教程之前, 必须能够在 WebSphere Test Environment 中运行样本商店, 并在商店中完成购物。

教程代码部署步骤

本书中包含的教程包含一些可选步骤, 它们描述了如何将定制代码部署到目标 WebSphere Commerce Server。已假定目标 WebSphere Commerce Server 正运行在与开发环境相分离的 Windows NT 或 Windows 2000 机器上。注意, 如果您正在使用 WebSphere Commerce Studio 商务开发者版作为您的开发环境, 则您应当部署到 WebSphere Commerce 商务版。如果您正在使用 WebSphere Commerce Studio 专业开发者版作为您的开发环境, 则您应当部署到 WebSphere Commerce 专业版。

另外, 在目标 WebSphere Commerce Server 上, 您必须已经发布了基于 “流行时尚” 样本商店的商店。在该商店中, 您必须能够完成购物。对于支付处理, 您可以使用远程或本地 Payment Manager。

如果正在部署到与开发环境相同机器上或运行在不同操作系统上的目标 WebSphere Commerce Server, 请参阅第 169 页的第 8 章, 『开发工具和部署』和第 311 页的附录 B, 『部署详细信息』, 获取适当的部署信息。

准备样本项目

在本节中，您导入 WC_SAMPLE_54.dat 资源库，它包含“创建新的业务逻辑”教程的起点。

要准备环境，请执行以下操作:

1. 请确保您正在使用 WebSphere Commerce 代码的 WC_54.dat 资源库。您可以在 WebSphere Commerce 商务版 V5.4 磁盘 2 CD 或 WebSphere Commerce 专业版 V5.4 磁盘 2 CD 上得到它。
2. 执行以下操作将样本项目导入到工作空间中:
   a. 将以下 CD 插入到您的开发机器上的光盘驱动器中。
      - Business WebSphere Commerce 商务版，V5.4 Disk 2
      - Professional WebSphere Commerce 专业版，V5.4 Disk 2
   b. 打开 VisualAge for Java。
   c. 从**文件**菜单中，选择**导入**
      “导入”智能向导打开。
   d. 选择导入**资源库**并单击**下一步**。
   e. 在“从另一个资源库导入”窗口中，执行以下操作:
      1) 选择**本地资源库**。
      2) 在**资源库名称**字段中，输入
         CD_drive:\repository\samples\programguide\WC_SAMPLE_54.dat
         其中 *CD_drive* 是光盘驱动器。
      3) 选择**项目**并单击**详细信息**。选择 **_WCSamples** 项目。选择 **WC 样本 5.4** 版本，并单击**确定**。
      4) 确保已选择**添加最新的项目修订版至工作空间**。
      5) 单击**完成**开始导入。
         导入该项目可能需要花费几分钟时间。
3. 请确保通过执行以下操作，将工作空间所有者设置为“WCS 开发者”:
   a. 从**工作空间**菜单中选择**更改工作空间**所有者。
   b. 选择 **WCS 开发者**并单击**确定**。
4. 将用于练习的 JSP 模板复制到相应的目录中，从而使它们可在 WebSphere Test Environment 中使用。要复制这些文件，请执行以下操作:
   a. 在以下目录中定位 Sample.jsp 和 Sample_All.jsp 文件

      CD_drive:\repository\samples\programguide\

      其中 *CD_drive* 是以下目录中的 CD 驱动器:

b. 将这两个文件复制到以下目录:

```
vaj_drive:\VAJava\ide\project_resources\IBM WebSphere Test Environment
    \hosts\default_host\default_app\web
```

其中 *vaj_drive* 是安装了 VisualAge for Java 的驱动器。

5. 将访问控制策略文件复制到相应的目录中。这些文件用于为您在学习本教程期间创建的新资源装入新的访问控制策略。要复制这些文件，请执行以下操作:
   a. 切换到以下目录:

      ```
      CD_drive:\repository\samples\programguide\
      ```

   b. 在该目录中定位以下文件:
      - SampleCmdACPolicy.xml
        此 XML 文件包含创建新的控制器命令时使用的访问控制策略。
      - SampleACPolicy.xml
        此 XML 文件包含创建新的企业 bean 时使用的访问控制策略。
      - SampleACPolicy_*locale*.xml
        其中 *locale* 是语言标识。此 XML 文件包含访问控制策略描述。

   c. 将先前的文件复制到以下目录:
      drive:\WebSphere\CommerceServerDev\xml\policies\xml
      其中 *drive* 是您安装了 WebSphere Commerce Studio 的驱动器。
6. 执行以下操作，测试您的环境，确保准备开始教程:
   a. 如第 309 页上所述，启动持久名称服务器。
   b. 如第 310 页上所述，启动 EJB 服务器。
   c. 如第 310 页上所述，启动小服务程序引擎
   d. 打开浏览器并输入以下 URL:

      ```
      http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
         storeId=store_ID&catalogId=catalog_Id&langId=-1
      ```

      其中 *store_ID* 是样本商店标识符（10001 是一个样本值），*catalog_Id* 是样本商店产品目录标识符（10001 是一个样本值）。
      当显示样本商店主页时，选择产品并购买它。您必须能够在购物流程中到达“订单确认”页面。

要验证商店的 storeId 值，您可参阅 STOREENT 表。

   e. 关闭所有浏览器，并停止 WebSphere Test Environment。

现已准备好开始本教程。

写控制器命令

本部分显示了如何写新的控制器命令。完成此练习后，您将有一个新命令，名为 *MyNewControllerCmd*。该命令供所有商店使用，并且每家商店使用该命令的相同实现。

创建新的控制器命令有三个基本步骤:

1. 在命令注册表框架中注册命令。
2. 为命令创建接口。
3. 为命令创建实现类。

关于命令的更多信息，请参阅第 20 页的『命令设计模式』。

开始本教程之前

您应已经完成第 184 页的『准备样本项目』中的步骤。

要写新命令，请执行以下操作:

1. 写控制器命令的第一步是建立命令的名称。在本示例中，命令名为 *MyNewControllerCmd*。
2. 必须在命令注册表框架中注册命令。新控制器命令的注册过程需要在 URLREG 表中创建条目。

   DB2 如果正在使用 DB2 数据库，请执行以下操作注册命令:

   a. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 控制中心**）。
   b. 从**工具**菜单，选择**工具设置**。
   c. 选择**使用语句终止字符**复选框，并确保指定的字符是分号（**;**）
   d. 先选择“脚本”选项卡，然后通过在脚本窗口中输入以下信息，在 URLREG 表中创建必需的条目:

```
connect to your_database_name;
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS,
    DESCRIPTION, AUTHENTICATED) values ('MyNewControllerCmd',0,
    'com.ibm.commerce.sample.commands.MyNewControllerCmd',0,
    'This is a new controller command for test/education purposes.',
    null)
```

   其中 *your_database_name* 是数据库名，并单击“执行”图标。
   此命令供所有商家使用（由 STOREENT_ID 的值为 0 指示）。

Oracle 如果正在使用 Oracle 数据库，请执行以下操作注册命令:

a. 打开 Oracle SQL Plus 命令窗口（**开始 > 程序 > Oracle > 应用程序开发 > SQL Plus**）。

b. 在**用户名**字段，输入 Oracle 用户名。

c. 在**密码**字段中，输入 Oracle 密码。

d. 在**主机字符串**字段，输入连接字符串。

e. 在 SQL Plus 窗口中，输入以下内容，在 URLREG 表中创建必需条目:

```
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS,
    DESCRIPTION, AUTHENTICATED) values ('MyNewControllerCmd',0,
    'com.ibm.commerce.sample.commands.MyNewControllerCmd',0,
    'This is a new controller command for test/education purposes.',
    null);
```

并按 Enter 键运行 SQL 语句。

f. 输入以下命令提交数据库的更改:

```
commit;
```

并按 Enter 键运行 SQL 语句。

注: 为使本练习简单化，新命令仅有一个实现类，因此所有商店使用相同的实现类。此实现类直接在接口代码中指定。从而不需要在 CMDREG 表中注册接口与实现类之间的映射。当处于教程环境外部时，您应在 CMDREG 表以及 URLREG 表中注册控制器命令。

3. 控制器命令必须返回视图。将创建的新控制器命令返回 SampleViewTask 视图。您必须在 VIEWREG 表中注册 SampleViewTask 视图。

DB2 如果正在使用 DB2 数据库，请执行以下操作注册视图:

a. 通过在脚本窗口输入以下内容，在 VIEWREG 表中创建条目（注意: 可能需要先从窗口清除上一条 SQL 语句）:

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
   CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE)
values ('SampleViewTask',-1, 0,
   'com.ibm.commerce.command.ForwardViewCommand',
   'com.ibm.commerce.command.HttpForwardViewCommandImpl',
   'docname=Sample.jsp','This is a sample view for the
    Bonus Point exercise', 0, null)
```

并单击“执行”图标。

Oracle 如果正在使用 Oracle 数据库，请执行以下操作在数据库中注册视图:

a. 在 SQL Plus 窗口中，输入以下 SQL 语句:

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
   CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE)
values ('SampleViewTask',-1, 0,
   'com.ibm.commerce.command.ForwardViewCommand',
   'com.ibm.commerce.command.HttpForwardViewCommandImpl',
   'docname=Sample.jsp','This is a sample view for the
   Bonus Point exercise', 0, null);
```

并按 Enter 键运行 SQL 语句。

b. 输入以下命令提交数据库的更改:

```
commit;
```

并按 Enter 键运行 SQL 语句。

4. 在 VisualAge for Java 工作台窗口中，展开 **_WCSamples** 项目。
5. 展开 **com.ibm.commerce.sample.commands** 数据包，用鼠标右键单击 **MyNewControllerCmd** 接口，并选择**添加 > 字段**。
   “创建字段”智能向导打开。
6. 通过执行以下操作，创建为接口指定缺省实现类的字段:
   a. 在**字段名称**字段中，输入 defaultCommandClassName。
   b. 从**字段类型**下拉列表中选择 String。
   c. 在“初始值”字段中，输入 "com.ibm.commerce.sample.commands.MyNewControllerCmdImpl"。（确保包含了双引号。）
   d. 单击**完成**。
      生成新字段的代码。

在本教程的任何地方，当您覆盖已存在于超类中的字段或方法时，将可能显示警告，指出新的字段或方法将隐藏继承的字段或方法。如果是这样，单击**是**继续。

7. 展开 **MyNewControllerCmdImpl** 类，并选择 **performExecute** 方法查看其源代码。
8. 在 performExecute 方法的源代码中，取消对第 1 段和第 5 段的注释。第 1 段将以下代码引入到方法中:

```
// Create a new TypedProperties for output.
  TypedProperty rspProp = new TypedProperty();
```

第 5 段将以下代码引入到方法中:

```
// see how controller command call a JSP

rspProp.put(ECConstants.EC_VIEWTASKNAME, "SampleViewTask");
setResponseProperties(rspProp);
```

保存工作（**Ctrl + S**）。上述代码片段将视图名称设置成由控制器命令返回。

9. 必须对此新控制器命令指定命令级别的访问控制。在此情况下，命令级别的访问控制策略将指定允许所有用户执行此命令。在 SampleCmdACPolicy.xml 文件中指定此策略。要装入新的访问控制策略，请执行以下操作:
   a. 在命令提示符下，切换到以下目录:
      drive:\WebSphere\CommerceServerDev\ bin
   b. 您必须发出 acpload 命令，此命令具有以下形式:

      ```
      acpload db_name db_user db_password inputXMLFile
      ```

      其中
      - *db_name* 是数据库的名称
      - *db_user* 是数据库用户名
      - *db_password* 是数据库密码
      - *inputXMLFile* 是包含策略的 XML 文件名称。

      例如，可以发出以下命令:

      ```
      acpload VAJ_Demo user password SampleCmdACPolicy.xml
      ```

10. 通过执行以下操作，将新的 _WCSamples 项目添加到小服务程序引擎的类路径中:
    a. 从**工作空间**菜单中选择**工具 > WebSphere Test Environment**。
       “WebSphere Test Environment 控制中心”将打开。
    b. 单击**小服务程序引擎**，然后单击**编辑类路径**
       在“小服务程序引擎类路径”窗口中，单击**全部选中**，然后单击**确定**。
11. 通过执行以下操作，测试新命令:
    a. 如第 309 页上所述，启动持久名称服务器。
    b. 如第 310 页上所述，启动 EJB 服务器。
    c. 如第 310 页上所述，启动小服务程序引擎
    d. 打开浏览器并输入以下 URL:

       ```
       http://localhost:8080/webapp/wcs/stores/servlet/
          MyNewControllerCmd
       ```

       几分钟之后，浏览器显示一个页面，它如下显示“样本 JSP”：

图 31.

12. 以代码当前状态创建代码版本。这使您可以在任何时候将代码恢复到此状态。要创建版本，请执行以下操作:

    a. 选择 **com.ibm.commerce.sample.commands** 和 **com.ibm.commerce.sample.databeans** 数据包（按住 Ctrl 键的同时突出显示来选择多个数据包），用鼠标右键单击并选择**管理 > 创建开放版本**。

    b. 用鼠标右键单击 **_WCSamples** 项目并选择**管理 > 创建开放版本**。

    c. 再次用鼠标右键单击 **_WCSamples** 项目并选择**管理 > 创建版本**。“为选定项创建版本”窗口打开。

    d. 选择**一个名称**单选按钮，输入 mySample 1.2 Completed 并单击**确定**。

现在您已添加新命令，并已在集成的测试环境中对其进行了（简单）测试。

修改 MyNewControllerCmd

在前一部分中，您创建了 MyNewControllerCmd。在本练习中，将进一步地检查新命令的内容，以更好地理解如何写自己定制的控制器命令。

从检查已创建的代码的结构开始。MyNewControllerCmd 接口扩展 ControllerCommand 接口。它还定义用作缺省值的实现类。当在 CMDREG 表中没有注册该命令或指定实现类时，将使用此类。

您还创建了 MyNewControllerCmdImpl 类。该实现类是最终将包含您所希望实现的业务逻辑（或调出任务命令以执行单个的业务任务）的类。您的实现类包含以下方法:

MyNewControllerCmdImpl 中的方法	描述
MyNewControllerCmdImpl()	构造函数方法。
validateParameters()	用于对命令输入参数的服务器端验证。
isGeneric()	确定一般用户是否可调用命令。
isRetriable()	确定数据库回滚后是否将重试命令。
performExecute()	包含命令的业务逻辑。

以下部分显示了有关如何更新新控制器命令的更多详细信息。

传递变量到 JSP 模板

在本部分中，您将修改 MyNewControllerCmd 以将变量传递到 JSP 模板。为显示变量，您必须使用新的数据 bean，名为 DataBeanSampleBean。要启用在 JSP 模板中显示变量，请执行以下操作:

1. 在 VisualAge for Java 工作台窗口中，展开 **_WCSamples** 项目。
2. 展开 **com.ibm.commerce.sample.commands** 数据包，然后展开 **MyNewControllerCmdImpl** 类，并选择 **performExecute** 方法。
3. 在 performExecute 方法的源代码中，取消对第 2 段的注释。这将以下代码引入到方法中:

   ```
   // see how controller command pass in variables to JSP

   // Add additional parameters in controller command
   // to rspProp for response
   //
   rspProp.put("ControllerParm1", "Hello world");
   rspProp.put("ControllerParm2", "Have a nice day!");
   ```

   以上代码片段创建两个新参数，这些参数放在传递到 JSP 模板的属性中。保存工作（**Ctrl + S**）。
4. DataBeanSampleBean 数据 bean 由 JSP 模板用于显示变量。已为您创建该 bean，但您需要如下修改源代码:
   a. 展开 **com.ibm.commerce.sample.databeans** 数据包。
   b. 展开 **DataBeanSampleBean** 类，然后选择 **setRequestProperties** 方法查看其源代码。
   c. 在 setRequestProperties 方法的源代码中，取消对第 1 段的注释。这将以下代码引入到方法中:

      ```
      // copy input TypedProperties to local

      requestProperties = aParam;
      ```

保存工作。上述代码片段将 aParam 的值复制到本地（aParam 在超类中定义）。这用于从请求对象获取属性。

5. 通过执行以下操作，更新 Sample.jsp 文件以使用新的数据 bean 并显示变量:

   a. 使用文本编辑器，打开 Sample.jsp 和 Sample_All.jsp 文件。这些文件位于以下目录:
   vaj_drive:\VAJava\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host \default_app\web

   b. 将 <!-- SECTION 1 --> 与 <!-- END OF SECTION 1 --> 标记符之间的代码第 1 段从 Sample_All.jsp 复制到 Sample.jsp 中。这将以下代码引入到 JSP 模板:

```
<!-- SECTION 1 -->
<%
DataBeanSampleBean testBean = new DataBeanSampleBean ();
com.ibm.commerce.beans.DataBeanManager.activate (testBean, request);
%>
<!-- END OF SECTION 1 -->
```

   此代码段实例化数据 bean。

   c. 将 <!-- SECTION 2 --> 与 <!-- END OF SECTION 2 --> 标记符之间的第 2 段从 Sample_All.jsp 复制到 Sample.jsp 中。这将以下代码引入到 JSP 模板中:

```
<!-- SECTION 2 -->
<%
TypedProperty prop = testBean.getRequestProperties();
out.print("<B>List of name value pairs in TypedProperties object</B><P>");
// convert from request Properties to query string
for (Enumeration pns = prop.keys(); pns.hasMoreElements();) {
     String paramName = (String) pns.nextElement();
     // do not add the url parameter to the query string
     Object val = prop.get(paramName,null);
     if (val != null) {
          if (val.getClass().isArray()) {
          // flatten the array
         String[] oarray = (String[]) val;
          int len = java.lang.reflect.Array.getLength(val);
          for (int i = 0; i < len; i++) {
          out.print(paramName + "[" + i + "]=" + oarray[i] + "<br>");
          }
     } else {
     // assume that it is a String
     out.print(paramName + "=" + val.toString() + "<br>");
     }
   }
}
%>
<P>
<!-- END OF SECTION 2 -->
```

保存此文件。

此代码段使用 testBean 数据 bean 对象的 getRequestProperties 方法。它在属性中循环，并将它们显示在浏览器中。

6. 通过执行以下操作，测试修改以验证变量是否显示在 JSP 模板中:
   a. 确保 WebSphere Test Environment 正在运行。
   b. 在浏览器中输入以下 URL:

      ```
      http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd
      ```

      显示样本 JSP 文件，其中显示了控制器命令的属性。输出如下显示:

图 32.

7. 以代码当前状态创建代码版本。将版本命名为 mySample 1.3 Completed。如果需要关于为代码创建版本的详细信息，请参阅第 190 页的12。

修改 validateParameters 方法

当前在新命令中的 validateParameters 方法实际上只是命令存根。它由以下代码组成:

```
public void validateParameters() throws ECException {
     }
```

在本部分中，您将把自己定制的参数检查添加到命令中，然后将参数传递到 JSP 模板。

修改 validateParameters 方法时，您将新字段添加到用于参数的类中。

创建新字段: 当将新字段添加到现有的接口或类时，您可使用 VisualAge for Java 中的“创建字段”智能向导。本部分描述了创建新字段的一般步骤。在教程以下部分中，当必须将新字段添加到接口或类时，可使用本信息。

要创建新字段，请执行以下操作:

1. 用鼠标右键单击向其添加新字段的接口或类，并选择**添加 > 字段**。“创建字段”智能向导打开。
2. 在**字段名称**字段中，输入新字段的名称。
3. 要指定字段类型，请执行以下一项操作:
   - 从**字段类型**下拉列表中选择字段类型。
   - 如果需要的字段类型在列表中没有指定，请单击**浏览**。在**模式**字段中，输入字段类型的名称（或部分名）并单击**确定**。

   注: 在教程中，每当指定 String 字段类型时，它来源于 java.lang 软件包。
4. 在**初始值**字段中，输入字段的初始值。对于是字符串的初始值，请确保使用双引号（“ ”）将值括起来。
5. 对于**访问修饰符**值，如果要求，请选择合适的单选按钮（public、protected、“无”或 private）。
6. 如果希望为该字段生成获取函数和设置函数方法，请选择**使用获取函数和设置函数方法进行访问**复选框。如果选择此复选框，则还必须如下指定获取函数和设置函数方法的属性:
   - 对于获取函数方法，选择 public、protected、private 或“无”其中一个单选按钮。
   - 对于设置函数方法，选择 public、protected、private 或“无”其中一个单选按钮。

7. 单击**完成**。

要修改 validateParameters 方法，请执行以下操作:

1. 您必须在 MyNewControllerCommandImpl 类中创建两个新字段。第一个用于输入字符串，而第二个用于输入整数。要创建这些字段，请执行以下操作:
   a. 展开 **com.ibm.commerce.sample.commands** 数据包。
   b. 选择 **MyNewControllerCmdImpl** 类。
   c. 使用以下值将字段添加到该类。关于如何创建新字段的详细步骤，请参阅第 194 页的『创建新字段』。

属性名称	值
字段名称	inputString
字段类型	String
初始值	留空。
访问修饰符	private
其它修饰符	全部保留不选中。
使用获取函数和设置函数方法进行访问	选中
获取函数	public
设置函数	public

   d. 使用以下值创建用于输入整数的另一个字段:

属性名称	值
字段名称	inputInteger
字段类型	Integer **注**: 单击**浏览**并输入 Integer。不要选择 int。
初始值	留空。
访问修饰符	private
其它修饰符	全部保留不选中。
使用获取函数和设置函数方法进行访问	选中
获取函数	public
设置函数	public

2. 选择 MyNewControllerCmdImpl 类中的 **performExecute** 方法。
3. 在 performExecute 方法的源代码中，取消对第 3 段的注释，以将以下代码引入到方法中:

```
// see how controller command pass in input variables to JSP
     rspProp.put("ControllerInput1", getInputString());
     rspProp.put("ControllerInput2", getInputInteger().toString());
```

此代码将变量从控制器命令传递到数据 bean。保存工作。

4. 选择 MyNewControllerCmdImpl 类中的 **validateParameters** 方法。
5. 取消对 validateParameters 源代码第 1 段的注释，以将以下代码引入到方法中:

```
// uncomment to check parameters

TypedProperty prop = getRequestProperties();

// retrieve required parameters
//
try {
          setInputString(prop.getString("input1"));
} catch (ParameterNotFoundException e) {
          throw new ECApplicationException(
               ECMessage._ERR_CMD_MISSING_PARAM,
               "MyControllerCmdImpl","validateParameters",
               ECMessageHelper.generateMsgParms(e.getParamName()));
}

// retrieve optional Integer
// set input2 = 0 if no input value
//
setInputInteger(prop.getInteger("input2", 0));
```

保存工作。

上述代码片段检查两个输入参数。尝试块确定第一个参数是否存在，如果不存在，它将抛出异常。由于第二个参数是可选的，因此当该参数遗漏或类型错误时，此代码将把该参数的值设置为 0。

6. 通过执行以下操作，测试该命令:
   a. 确保持久名称服务器、EJB 服务器和小服务程序引擎正在运行。
   b. 在浏览器中，通过输入以下 URL:
      - 案例 1: 遗漏参数:
        输入

        ```
        http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd
        ```

        由于没有参数传递到命令，因此显示一般应用程序错误，指出参数遗漏。结果显示在以下屏幕快照中:

图 33.

注: 如果显示“未找到页面”错误，则您的小服务程序引擎可能停止了。检查 WebSphere Test Environment 控制中心获取详细信息。如果显示“样本 JSP”页面，而不是“一般应用程序错误”页面，则您可能需要停止并重新启动小服务程序引擎，或将页面重新装入浏览器。

- 案例 2: 第一个参数有效，第二个参数遗漏:
  输入

  ```
  http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
        input1=abc
  ```

  此命令的结果是虽然第二个参数省略但仍然显示“样本 JSP”页面。以下屏幕快照显示了此结果。为什么没有返回错误的说明在屏幕快照之后。

图 34.

不返回错误是因为 `getInteger` 方法的使用方式。具体地说，`setInputInteger(prop.getInteger("input2", 0))` 代码行为 input2 设置了缺省值 0。此缺省值在参数遗漏或参数类型错误时使用。为对此参数强制执行类型检查，请将代码更改为 `setInputInteger(prop.getInteger("input2"))` 并再次输入 URL（确保刷新浏览器）。应显示一般应用程序错误页面。

注: 如果测试代码中的 setInputInteger(prop.getInteger("input2")) 修改，请在继续下一个测试案例之前将它切换回 setInputInteger(prop.getInteger("input2", 0))。

- 案例 3: 第一个参数有效，第二个参数无效:
  输入

  ```
  http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
       input1=abc&input2=abc
  ```

  此命令的结果是显示“样本 JSP”页面并且第二个参数的值（整数）设置为缺省值 0。如上述测试案例所述，这是所用 getInteger 方法的结果。结果显示在以下屏幕快照中:

图 35.

- 案例 4: 两个有效参数:
  输入

  ```
  http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
       input1=abc&input2=1000
  ```

此命令的结果是显示“样本 JSP”页面并且两个输入值都根据输入的显示。结果显示在以下屏幕快照中:

图 36.

7. 以代码当前状态创建代码版本。将版本命名为 mySample 1.4 Completed。如果需要关于如何为代码创建版本的详细信息，请参阅第 190 页的12。

创建任务命令

控制器命令通常代表业务过程或复杂的功能。例如，所有与处理订单相关的业务逻辑都封装在 OrderProcessCmd 控制器命令中。业务过程通常可分成较小的、特定的任务。例如，OrderProcessCmd 控制器命令中，有若干个任务命令可调用来执行单项的工作。其中一个由 OrderProcessCmd 控制器命令调用的任务命令是 CalculateOrderTaxTotalCmd。

MyNewControllerCmdImpl 当前不调用任何任务命令。本练习有两个部分。在第一部分中创建新的任务命令。在第二部分中修改控制器命令的 performExecute 方法来调用任务命令。

以下代码片段显示了 MyNewControllerCmdImpl 类中的当前 performExecute 方法，其中所有注释已除去:

```
public void performExecute() throws ECException {

      super.performExecute();

    TypedProperty rspProp = new TypedProperty();
     rspProp.put("ControllerParm1", "Hello world");
     rspProp.put("ControllerParm2", "Have a nice day!");

     rspProp.put("ControllerInput1", getInputString());
     rspProp.put("ControllerInput2", getInputInteger().toString());

     rspProp.put(ECConstants.EC_VIEWTASKNAME, "SampleViewTask");
  setResponseProperties(rspProp);
}
```

写任务命令代码: 本部分显示了如何写新的任务命令。创建全新的任务命令需要创建接口和实现类。创建任务命令时，接口应扩展 `com.ibm.commerce.commands.TaskCommand`。实现类应扩展 `com.ibm.commerce.command.TaskCommandImpl`。

完成此练习后，您将有一个新命令，名为 *MyNewTaskCmd*。该命令供所有商店使用，并且每家商店使用该命令的相同实现。

在教程的本部分中，您将字段和方法添加到新任务命令的接口。您先前已为 MyNewControllerCmdImpl 类创建了新字段。尝试自己为此接口创建字段，但如果需要更多详细信息，请参阅第 194 页的『创建新字段』。

创建方法: 本部分描述了将方法添加到现有类和接口的一般步骤。请阅读此处的指导，并在教程要求创建新方法时返回参阅指导。

要创建新方法，请执行下列操作:

1. 用鼠标右键单击向其添加方法的类或接口，并选择**添加 > 方法**。
   “创建方法”智能向导打开。
2. 确保已经选择**创建新方法**，并单击**下一步**。
3. 在**方法名称**字段中，输入新方法的名称。
4. 通过执行以下一项操作，指定方法的返回类型:
   - 从**返回类型**下拉列表中选择适当的返回类型。例如，选择 String。
   - 如果返回类型不在列表中，请单击**浏览**。然后在**模式**字段中，输入返回类型并单击**确定**。

   注: 在教程中，每当指定 String 为返回类型时，它来源于 java.lang 软件包。
5. 如果该方法带有参数，请单击**添加**。在“参数”窗口中，指定参数名称和任何其它必需信息，然后单击**添加**。当已经添加所有参数后，单击**关闭**。
6. 单击**下一步**。
   “属性”窗口打开。
7. 如果方法抛出异常，请在“属性”窗口中单击**添加**。在**模式**字段中，输入异常的名称，然后单击**添加**。当已经添加所有异常后，单击**关闭**。
8. 单击**完成**。
   生成方法的代码。

要创建 MyNewTaskCmd 命令，请执行以下操作:

1. 展开 **com.ibm.commerce.sample.commands** 数据包。
2. 用鼠标右键单击 **MyNewTaskCmd** 接口，并选择**添加 > 字段**。
3. 使用“创建字段”智能向导创建一个字段，它指定由接口使用的缺省实现类。使用下表中的值。如果需要关于创建字段的进一步详细信息，请返回参阅第 194 页的『创建新字段』。

属性名称	值
字段名称	defaultCommandClassName
字段类型	String
初始值	"com.ibm.commerce.sample.commands.MyNewTaskCmdImpl"

当生成该字段的代码时，它如下显示:

```
java.lang.String defaultCommandClassName =
     "com.ibm.commerce.sample.commands.MyNewTaskCmdImpl";
```

由于整个站点使用相同的实现类，并且没有缺省属性传递到命令，因此可以直接在代码中指定缺省实现。如果您的命令有多个实现或有多个缺省属性（它们存储在 CMDREG 表中），则必须在 CMDREG 表中注册该命令，以创建接口与实现类之间的映射。

4. 通过执行以下操作，将新方法添加到 MyNewTaskCmd 接口:
   a. 用鼠标右键单击 **MyNewTaskCmd** 接口，并选择**添加 > 方法**。使用“创建方法”智能向导，用在以下步骤中指定的值创建新方法。如果需要关于创建方法的详细信息，请参阅第 203 页的『创建方法』。
   b. 使用以下值，创建检索顾客奖金点数余额的新方法:

属性名称	值
方法名称	getOldBonusPoint
返回类型	String
参数	无
异常	无

注: 可能显示错误，指出在 MyNewTaskCmdImpl 实现类中没有实现刚才创建的被继承的抽象方法。这在后续步骤中进行更正。

   c. 创建从任务命令中检索用户标识输出的新方法。创建此方法时使用以下值:

属性名称	值
方法名称	getTask_output_userId
返回类型	String
参数	无
异常	无

   d. 创建从任务命令中检索输出值的新方法。创建此方法时使用以下值:

属性名称	值
方法名称	getTask_output1
返回类型	String
参数	无
异常	无

   e. 创建将第一个输入值设置到任务命令中的新方法。创建此方法时使用以下值:

属性名称	值
方法名称	setTask_input1
返回类型	void
参数名称	newTask_input1
引用类型	String
异常	无

f. 创建将第二个输入值设置到任务命令中的新方法。创建此方法时使用以下值:

属性名称	值
方法名称	setTask_input2
返回类型	void
参数名称	newTask_input2
原语类型	int
异常	无

5. 通过执行以下操作，将新字段添加到 MyNewTaskCmdImpl 类:

a. 用鼠标右键单击 **MyNewTaskCmdImpl** 类，并选择**添加 > 字段**。使用“创建字段”智能向导，用在以下步骤中指定的值创建新字段。如果需要关于创建新字段的附加信息，请参阅第 194 页的『创建新字段』。

b. 使用以下值在实现类中创建新字段:

属性名称	值
字段名称	task_input1
字段类型	String
初始值	留空。
访问修饰符	private
其它修饰符	全部保留不选中。
使用获取函数和设置函数方法进行访问	选中
获取函数	public
设置函数	public

c. 使用以下值在实现类中创建新字段:

属性名称	值
字段名称	task_input2

属性名称	值
字段类型	int
初始值	留空。
访问修饰符	private
其它修饰符	全部保留不选中。
使用获取函数和设置函数方法进行访问	选中
获取函数	public
设置函数	public

d. 使用以下值在实现类中创建新字段:

属性名称	值
字段名称	task_output_userId
字段类型	String
初始值	留空。
访问修饰符	private
其它修饰符	全部保留不选中。
使用获取函数和设置函数方法进行访问	选中
获取函数	public
设置函数	public

e. 使用以下值在实现类中创建新字段:

属性名称	值
字段名称	oldBonusPoint
字段类型	String
初始值	留空。
访问修饰符	private
其它修饰符	全部保留不选中。
使用获取函数和设置函数方法进行访问	选中
获取函数	public
设置函数	public

f. 使用以下值在实现类中创建新字段:

属性名称	值
字段名称	task_output1

属性名称	值
字段类型	String
初始值	留空。
访问修饰符	private
其它修饰符	全部保留不选中。
使用获取函数和设置函数方法进行访问	选中
获取函数	public
设置函数	public

6. 选择 **MyNewTaskCmdImpl** 类的 **performExecute** 方法查看其源代码。
7. 在源代码中，取消对第 1 段的注释以将以下代码引入到方法中:

```
// modify the task_input1 and see it in the NVP list

     setTask_output1( "Hello ! " + getTask_input1() );
```

保存工作。
上述代码段使新属性可通过该命令的输出来获取。

调用任务命令: 一旦已经创建任务命令，您需要从控制器命令中调用该命令。以下步骤描述了如何按此方式修改控制器命令。

1. 在工作台中，选择 **MyNewControllerCmdImpl** 类的 **performExecute** 方法。
2. 在“源码”窗格中，取消对第 4 段的注释以调用任务命令。这将以下代码引入到方法中:

```
// see how controller command call a task command

      MyNewTaskCmd cmd = null;
     try {
     cmd = (MyNewTaskCmd) CommandFactory.createCommand(
               "com.ibm.commerce.sample.commands.MyNewTaskCmd",
               getStoreId());
          // Set input parameters to task command
          cmd.setTask_input1(getInputString());
          cmd.setTask_input2(getInputInteger().intValue());
          // This is required for all commands
          cmd.setCommandContext(getCommandContext());
          // Invoke the command's performExecute method
          cmd.execute();
          // retrieve output parameter from task command

          rspProp.put("task_output1", cmd.getTask_output1());
          if (cmd.getTask_output_userId() != null) {
               rspProp.put("task_output_userId",
               cmd.getTask_output_userId());
          }
          if (cmd.getOldBonusPoint() != null) {
```

```
                rspProp.put("task_output_oldBonusPoint",
                cmd.getOldBonusPoint());
            }

    } catch (ECException ex) {
              // throw the exception as is
              throw (ECException) ex;
    }
```

保存工作。

上述代码片段使用命令工厂创建新任务命令。它然后设置命令上下文，调用任务命令的 performExecute，并从任务命令中检索输出参数。

3. 如下通过输入控制器命令的 URL 对命令进行测试:

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
     input1=abc&input2=1000
```

样本 JSP 显示了请求对象中的“名称值”对的列表，包括任务输出值。它应如下显示:

图 37.

4. 以代码当前状态创建代码版本。将版本命令为 `mySample 1.5 Completed`。如果需要关于如何为代码创建版本的更多详细信息，请参阅第 190 页的12。

验证用户标识

下一步是修改任务命令，以使用 UserRegistryAccessBean 来验证用户输入的值是否与注册用户的值相同。除修改任务命令外，您还必须修改 DataBeanSampleBean。

修改 MyNewTaskCmdImpl 以用于用户标识验证: 您必须修改 MyNewTaskCmdImpl 类中的 performExecute 方法，从而使它用 UserRegistryAccessBean 验证用户标识。要将此新功能添加到 performExecute 方法，请执行以下操作:

1. 在工作台中，选择 **MyNewTaskCmdImpl** 类的 **performExecute** 方法。
2. 在“源码”窗格中，取消对 performExecute 方法第 2 段的注释。这将以下代码引入到方法中:

```
// use UserRegistryAccessBean to check member reference number

     String refNum;
     UserRegistryAccessBean rrb = new UserRegistryAccessBean();

     try {
          rrb = rrb.findByUserLogonId(getTask_input1());
          refNum = rrb.getUserId();
     } catch (javax.ejb.FinderException e) {

     return;

     } catch (javax.naming.NamingException e) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
               this.getClass().getName(), "performExecute");
     } catch (java.rmi.RemoteException e) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
               this.getClass().getName(), "performExecute");
     } catch (javax.ejb.CreateException e) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
               this.getClass().getName(), "performExecute");
     }

      setTask_output_userId(refNum);
```

保存工作。

修改 DataBeanSampleBean: 您必须修改 DataBeanSampleBean 以使用预定义的输入参数。要修改此 bean，请执行以下操作:

1. 在工作台中，展开 **com.ibm.commerce.sample.databeans** 数据包。
2. 通过执行以下操作，将新字段添加到数据 bean:
   a. 用鼠标右键单击 **DataBeanSampleBean** 类，并选择**添加 > 字段**。使用“创建字段”智能向导，用在以下步骤中指定的值创建新字段。如果需要关于创建字段的其它信息，请参阅第 194 页的『创建新字段』。
   b. 使用以下值将新字段添加到数据 bean:

属性名称	值
字段名称	input1
字段类型	String
初始值	留空。
访问修饰符	private
其它修饰符	全部保留不选中。

属性名称	值
使用获取函数和设置函数方法进行访问	选中
获取函数	public
设置函数	public

c. 使用以下值将新字段添加到数据 bean:

属性名称	值
字段名称	input2
字段类型	int
初始值	留空。
访问修饰符	private
其它修饰符	全部保留不选中。
使用获取函数和设置函数方法进行访问	选中
获取函数	public
设置函数	public

d. 使用以下值将新字段添加到数据 bean:

属性名称	值
字段名称	task_output_userId
字段类型	String
初始值	留空。
访问修饰符	private
其它修饰符	全部保留不选中。
使用获取函数和设置函数方法进行访问	选中
获取函数	public
设置函数	public

3. 选择 DataBeanSampleBean 类的 **populate** 方法查看其源代码。
4. 在源代码中，取消对第 1A 段和第 1B 段的注释。这将以下代码引入到方法中:

```
//// Section 1A //////////
// set additional data fields

try {
   setInput1( getRequestProperties().getString("input1"));
   setInput2( getRequestProperties().getIntValue("input2", 0)  );
   setTask_output_userId(getRequestProperties().getString
     ("task_output_userId"));
```

```
//// End of Section 1A /////

   //// Section 2 ////////////
   /*
   // instantiate databean to BonusAccessBean
   setTask_output_oldBonusPoint(getRequestProperties().getString(
      "task_output_oldBonusPoint"));
   */
   //// End of Section 2 ////

//// Section 1A ///////////
// instantiate databean to BonusAccessBean and
// set additional data fields

   }
catch (ParameterNotFoundException e){}

//// End of Section 1B ////
```

保存工作。

上述代码片段使用 getRequestProperties 方法从控制器命令中获取数据字段值。

修改 Sample.jsp 以用于用户标识验证: 必须修改当前的 JSP 模板以执行用户标识验证。要修改 Sample.jsp 文件，请执行以下操作:

1. 在文本编辑器中打开 Sample.jsp 和 Sample_All.jsp 文件。
2. 将 <!-- SECTION 3 --> 与 <!-- END OF SECTION 3 --> 标记符之间的代码第 3 段从 Sample_All.jsp 复制到 Sample.jsp 中。以下代码引入到 JSP 模板中:

```
<!-- SECTION 3 -->

<B>Your first input is &lt; <%=testBean.getInput1()%> &gt;</B>

<%
String userId = testBean.getTask_output_userId();

if (userId == null) {

%>

<UL>
     <LI> This is not a registered user id.
</UL>

<%

} else {

%>

<B>
<UL>
     <LI> 'lt; <%=testBean.getInput1()%> &gt; is a registed user id.
```

```
     <LI> The member reference number of this user is <%=userId%>.
</UL>
</B>

<%

}

%>

<B>Your second input is < <%=testBean.getInput2()%> ></B> <P>

<!-- END OF SECTION 3 -->
```

保存 Sample.jsp 文件。

3. 打开浏览器并输入以下 URL:

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
     input1=abc&input2=1000
```

浏览器应该显示样本 JSP 文件指示 input1 的值不是有效的用户标识，如以下屏幕快照所示:

图 38.

4. 要查看当 input1 值是有效用户标识时的结果，请输入以下 URL:

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
    input1=wcsadmin&input2=1000
```

结果显示在以下屏幕快照中:

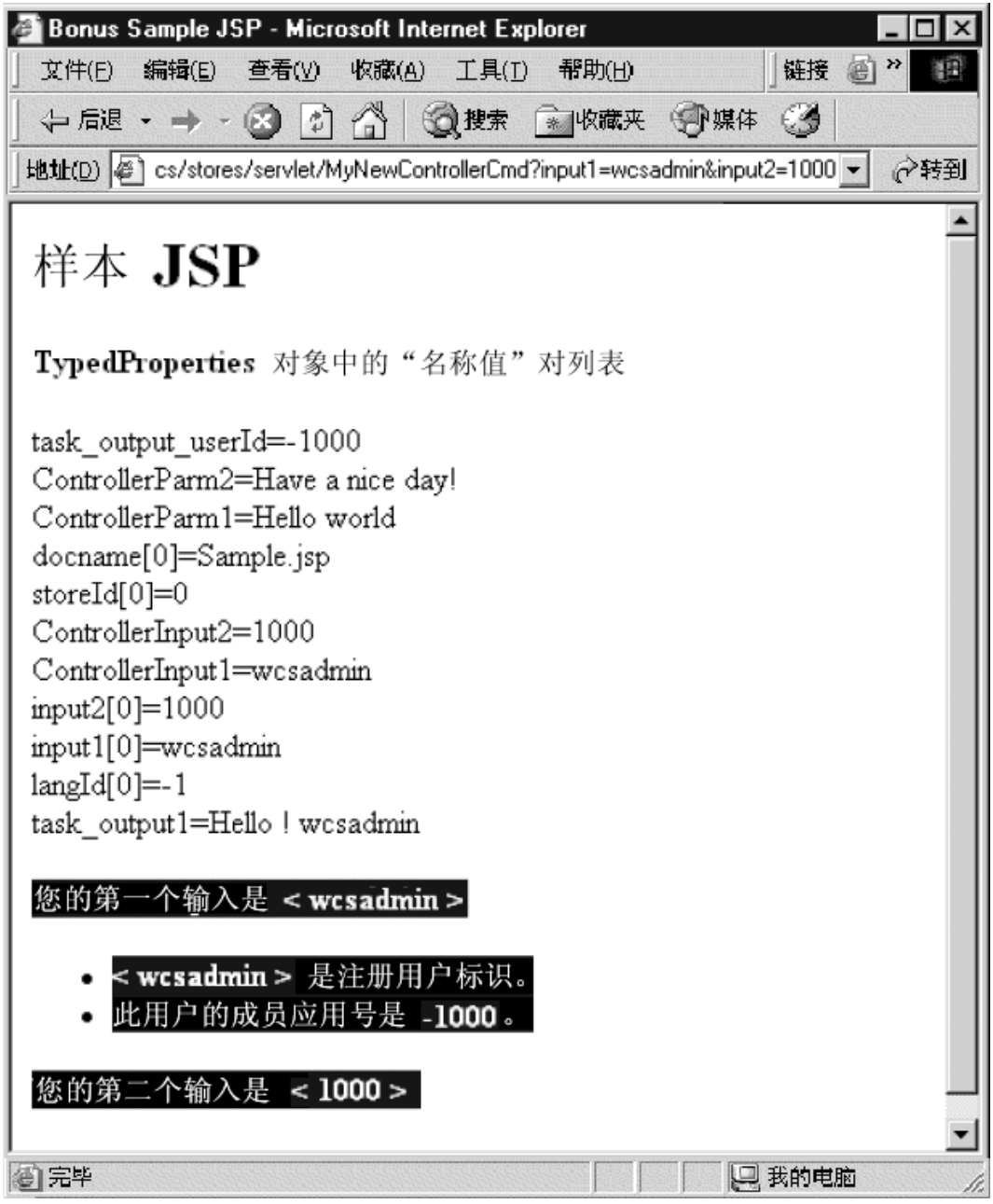


图 39.

5. 以代码当前状态创建代码版本。将版本命名为 mySample 1.6 Completed。如果需要关于如何为代码创建版本的详细信息，请参阅第 190 页的12。

创建新的实体 bean

本部分描述了如何使用 VisualAge for Java 来创建新的实体 bean。在本示例情景中，您有业务需求需要为商业应用程序中的每个用户包含其奖金点数记录。WebSphere Commerce 数据库模式不包含此信息，因此需要创建新的数据库表来包含此信息。依照 WebSphere Commerce 编程模型，一旦创建了数据库，您就必须创建实体 bean（它是企业 bean）以访问数据。

在本示例中，您将使用 VisualAge for Java 智能向导创建此实体 bean。

创建新的数据库表

在创建实体 bean 的准备中，必须首先创建新的数据库表。要创建的表称为 Bonus。

DB2 如果正在使用 DB2 数据库，请执行以下操作创建表:

1. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 控制中心**）并单击**脚本编制**选项卡。
2. 在“脚本”窗口中，输入以下内容:

```
connect to your_database_name;
CREATE TABLE Bonus (MEMBERID BIGINT NOT NULL,
BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
constraint f_memberid foreign key (MEMBERID)
references users (users_id) on delete cascade)
```

其中 *your_database_name* 是数据库的名称。单击“执行”图标。Bonus 表现已创建。

注: 如果任何人先前已使用此数据库运行本示例，则在创建 Bonus 表之前必须发出以下命令:

```
drop table Bonus
```

Oracle 如果正在使用 Oracle 数据库，请执行以下操作创建表:

1. 打开 Oracle SQL Plus 命令窗口（**开始 > 程序 > Oracle > 应用程序开发 > SQL Plus**）。
2. 在**用户名**字段，输入 Oracle 用户名。
3. 在**密码**字段中，输入 Oracle 密码。
4. 在**主机字符串**字段，输入连接字符串。

5. 在 SQL Plus 窗口中，输入以下 SQL 语句:

```
CREATE TABLE Bonus (MEMBERID NUMBER NOT NULL,
BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
constraint f_memberid foreign key (MEMBERID)
references users (users_id) on delete cascade);
```

并按 Enter 键运行 SQL 语句。BONUS 表现已创建。

注: 如果任何人先前已使用此数据库运行本示例，则在创建 Bonus 表之前必须发出以下命令:

```
drop table Bonus;
```

6. 输入以下命令提交数据库的更改:

```
commit;
```

并按 Enter 键运行 SQL 语句。

创建 BonusBean 实体 bean

一旦已经创建数据库，就已准备好开始创建新的实体 bean。下一步使用 VisualAge for Java。要创建新的实体 bean，请执行以下操作:

1. 创建 EJB 组。EJB 组是使您可以组织企业 bean 的逻辑组。您可对 EJB 组执行全局操作，EJB 组对驻留在该组中的所有企业 bean 进行迭代。例如，如果选择 EJB 组导出至 EJB JAR 文件，则将导出该组中的所有企业 bean。
   在本教程中，您将创建 EJB 组来组织与 Bonus 表定制相关的所有企业 bean。
   要创建 EJB 组，请执行以下操作:
   a. 在工作台中，单击 **EJB** 选项卡。
   b. 从 **EJB** 菜单中选择**添加 > EJB 组**。
      “添加 EJB 组”智能向导打开。
   c. 在**项目**字段中，输入 _WCSamplesEntityBeansProject。

      注: 出于部署目的，实体 bean 代码必须存储在其自己的项目中。
   d. 在**创建新 EJB 组名为**字段中，输入 WCSSamplesEntityBeans 并单击**完成**。
2. 创建新的实体 bean。
   要创建新的实体 bean，请执行以下操作:
   a. 在“企业 Bean”窗格中，用鼠标右键单击 **WCSSamplesEntityBeans** EJB 组并选择**添加 > 企业 Bean**。
      “创建企业 Bean”智能向导打开。

b. 输入以下信息:

属性	值
Bean 名称	Bonus **注**: 命名约定是以与实体 bean 所访问的表相同的名称来调用实体 bean。
Bean 类型	带有受容器管理的持久性（CMP）字段的实体 bean
创建新的 bean 类	启用
项目	_WCSamplesEntityBeansProject
数据包	com.ibm.commerce.sample.objects
类名	BonusBean
超类	com.ibm.commerce.base.objects.ECEntityBean

并单击**下一步**。

c. 单击**添加 CMP 字段到 bean** 文本框旁的**添加**按钮，为 BONUS 表中的 MEMBERID 列添加一个字段。
“创建 CMP 字段”智能向导打开。

d. 输入以下信息:

属性	值
字段名称	memberId
字段类型	Long **注**: 您必须使用 *Long* 数据类型，而不是 *long*。
关键字段	启用

并单击**完成**。

e. 再次单击**添加**为 BONUS 表中的 BONUSPOINT 列添加一个字段。

f. 使用以下信息创建另一个字段:

属性	值
字段名称	bonusPoint
字段类型	Integer **注**: 您必须使用 *Integer* 数据类型，而不是 *int*。
使用获取函数和设置函数方法进行访问	启用
提升获取函数和设置函数方法到远程接口	启用

然后在“创建 CMP 字段”窗口中单击**完成**。

g. 通过执行以下操作，使用访问控制保护此企业 bean:

1) 单击**远程接口应扩展哪些接口?** 旁边的**添加**。

2) 在**模式**字段中输入 com.ibm.commerce.security.Protectable，并单击**添加**。单击**关闭**关闭窗口。

h. 再次单击**完成**。

Bonus 实体将作为企业 bean 创建。

3. 通过执行以下操作，设置实体 bean 的隔离级别:

a. 用鼠标右键单击 **Bonus** bean，并选择**属性**。

b. 从**隔离级别**下拉列表中，选择 **TRANSACTION_READ_COMMITTED** 并单击**确定**。

4. 当您创建了新的企业 bean 时，VisualAge for Java 将在此 bean 中生成 EntityContext 字段以及相应的 getEntityContext() 和 setEntityContext(EntityContext) 方法。遵循 WebSphere Commerce 编程模型，新 bean 将扩展 com.ibm.commerce.base.objects.ECEntityBean 类，而 ECEntityBean 提供其自己对此字段和这些方法的实现。由于不应当覆盖 EntityContext、getEntityContext 和 setEntityContext，因此现在您必须从 bean 中删除生成的字段和方法。
要删除生成的 EntityContext 字段和它的获取函数和设置函数方法，请执行以下操作:

a. 在“类型”窗格中，选择 **BonusBean** 类。
“成员”窗格显示此类的字段和方法。

注: 如果“类型”窗格不可见，请在“属性”窗格中单击 C/I 图标（类／接口）。然后“类型”窗格打开。

b. 在“成员”窗格中，请执行以下操作:

1) 用鼠标右键单击 **entityContext** 字段并选择**删除**。

2) 用鼠标右键单击 **getEntityContext()** 方法，并选择**删除**。

3) 用鼠标右键单击 **setEntityContext(EntityContext)** 方法，并选择**删除**。

c. 保存工作（Ctrl+S）。

5. 通过执行以下操作，将新的 getMemberId 方法添加到企业 bean:

a. 用鼠标右键单击 **BonusBean** 类并选择**添加 > 方法**。
“创建方法”智能向导打开。

b. 使用下表中指定的值创建新方法。如果需要关于创建新方法的更多详细信息，请参阅第 203 页的『创建方法』。

表 11.

属性名称	值
方法名称	getMemberId
返回类型	Long
参数	无
异常	无

c. 当生成新方法时，请查看源代码。

d. 缺省情况下，该方法包含以下代码:

```
return null;
```

将此代码更改为

```
return memberId;
```

e. 通过用鼠标右键单击 **getMemberId** 方法并选择**添加至 > EJB 远程接口**，将新方法添加到远程接口。

6. 在 BonusBeanFinderHelper 中添加新的 FinderHelper 字段。
此接口包含与在下一步中所创建的 FinderHelper 方法相对应的搜索子句。要添加 FinderHelper 字段，请执行以下操作:

a. 在“类型”窗格中，单击 **BonusBeanFinderHelper** 接口。

b. 在“源码”窗格中修改代码，使其如下显示:

```
public interface BonusBeanFinderHelper {
public static final String
                    findByMemberIdWhereClause = " (MEMBERID = ?) ";
}
```

注: “WhereClause”的语法非常重要；它必须与用于 FinderHelper 方法的方法名称相匹配。在此情况下，findByMemberIdWhereClause 中的“findByMemberId”与在下一步中创建的方法名称（findByMemberId）完全匹配。

7. 在 BonusHome 接口中添加新的 FinderHelper 方法。
要添加新的 FinderHelper 方法，请执行以下操作:

a. 在“类型”窗格中，用鼠标右键单击 **BonusHome** 接口，并选择**添加 > 方法**。
“创建方法”智能向导打开。

b. 选择**创建新方法**并单击**下一步**。

c. 在**方法名称**字段中，输入 findByMemberId。

d. 在**返回类型**字段中，输入 Bonus。

e. 单击**此方法应有什么参数?** 旁的**添加**。“参数”窗口打开。

f. 在**名称**字段中，输入 argMemberId。

g. 选择**引用类型**并输入 Long。单击**添加**然后单击**关闭**。

h. 单击**下一步**。

i. 单击**此方法可能抛出什么异常?** 字段旁的**添加**，在**模式**字段输入 RemoteException，并单击**添加**。这在“属性”窗口中添加 java.rmi.RemoteException 异常。（“属性”窗口可能位于“异常”窗口的后面。）

j. 在“异常”窗口的**模式**字段中，输入 FinderException，单击**添加**，然后单击**关闭**。javax.ejb.FinderException 异常列出在“属性”窗口中。

k. 单击**完成**。

8. 将新的 ejbCreate 方法添加到 EJB。将此方法提升到主接口，从而使它在生成的访问 bean 中可用。要创建此方法，请执行以下操作:

   a. 在“类型”窗格中选择 **BonusBean** 类。

   b. 单击“成员”窗格中的 **ejbCreate(Long)**。

   c. 修改代码使其与以下内容相匹配:

```
public void ejbCreate(java.lang.Long argMemberId,
   Integer argBonusPoint)
throws javax.ejb.CreateException, java.rmi.RemoteException {
_initLinks();
// All CMP fields should be initialized here.
memberId=argMemberId;
            bonusPoint=argBonusPoint;
}
```

   d. 保存代码，VisualAge for Java 将在“成员”窗格中创建名为 **ejbCreate(Long, Integer)** 的新方法。

   e. 用鼠标右键单击原始的 **ejbCreate(Long)** 方法并选择**删除**。

9. 将新方法添加至主接口。这将使方法在访问 bean 类中可用。要实现此目的，请执行以下操作:

   a. 用鼠标右键单击 BonusBean 类中的 **ejbCreate(Long, Integer)** 方法并选择**添加到 > EJB 主接口**。

10. 通过执行以下操作，更新 getOwner 方法:

    a. 在“类型”窗格中选择 **BonusBean** 类。

    b. 在“成员”窗格中单击 **getOwner()** 方法。

注: 如果选择了查看继承方法，您会看到两个 getOwner() 方法。一个是从 ECEntityBean 类继承的。它不是在这步中应该选择的方法。请确保选择特定于 BonusBean 类的 getOwner 方法。

c. getOwner 方法的源代码如下显示:

```
public Long getOwner()
   throws Exception, java.rmi.RemoteException {
      return null;
   }
```

必须更改该方法返回的值。应该更改的代码部分以下用粗体显示:

```
public Long getOwner()
   throws Exception, java.rmi.RemoteException {
      return getMemberId();
   }
```

保存工作。

d. 单击“成员”窗格中的 **fulfills(Long, String)** 方法。

注: 如果您选择查看继承的方法，则您将看见两个 fulfills(Long, String) 方法。一个是从 ECEntityBean 类继承的。它不是在这步中应该选择的方法。请确保您选择了特定于 BonusBean 类的 fulfills(Long, String) 方法。

e. fulfills(Long, String) 方法的源代码如下显示:

```
public boolean fulfills(Long member, String relationship)
   throws Exception, java.rmi.RemoteException {
      return false;
  }
```

必须指定用户必须满足的关系。要实现此目的，必须更改以下用粗体显示的代码部分:

```
public boolean fulfills(Long member, String relationship)
  throws Exception, java.rmi.RemoteException
  {

      if (relationship.equalsIgnoreCase("creator"))
      {
         return member.equals(getMemberId());
      }      return false;
  }
```

保存工作。

11. 将 BONUS 数据库表映射到 BonusBean。将数据库模式映射到 BonusBean 实体的第一步需要使用 VisualAge for Java 中的工具来创建数据库模式。请执行以下操作来创建模式:

a. 在“工作空间”窗口中，从 **EJB** 菜单中选择**打开到 > 数据库模式**。

b. 从**模式**菜单中选择**导入 / 导出模式 > 从数据库导入模式**。
“必需的信息”窗口打开。

c. 在**模式名称**字段中，输入 WCSSamples，并单击**确定**。
“数据库连接信息”窗口打开。

d. 填写以下信息:

属性	DB2 **DB2 值**	Oracle **Oracle 值**
连接类型	COM.ibm.db2.jdbc.app.DB2Driver	Oracle.jdbc.driver.OracleDriver
数据源	jdbc:db2:*wcs_db_name*	jdbc:oracle:thin:@*hostname*:*port*:*SID*
用户名	*wcs_db_user_name*	*wcs_db_user_name*
密码	*wcs_db_password*	*wcs_db_password*

用以下值替换:

- DB2 *wcs_database_name* 是您的 WebSphere Commerce 数据库名称
- Oracle *hostname* 是 Oracle 主机名称
- Oracle *port* 是 Oracle 数据库的端口号（例如，1521）。
- *wcs_db_user_name* 是数据库的用户名。
- *wcs_db_password* 是数据库的密码。

单击**确定**。
“选择表”窗口打开。

e. 从**限定符**列表中选择数据库限定符（它可以是数据库用户名或机器名称），并单击**构建表列表**。将装入一列可用的数据库表。

f. 从“表”窗格中选择 **Bonus**，并单击**确定**。等待几分钟。

g. 在“模式浏览器”中，单击新添加的表，以使表中的两列都显示出来。

h. 用鼠标右键单击 **Bonus** 表，并选择**编辑表**。
“表编辑器”打开。

i. 从**限定符**字段中除去任何条目。除去限定符信息是一种好的做法，因为这样可以将代码部署到使用不同数据库的其它机器上。

j. Oracle 如下修改列数据类型:

1) 选择 **MEMBERID** 列，然后单击**编辑**。从“类型”下拉列表中，选择 **BIGINT** 并单击**确定**。

2) 选择 **BONUSPOINT** 列，然后单击**编辑**。从“类型”下拉列表中，选择 **INTEGER** 并单击**确定**。

k. 单击**确定**退出“表编辑器”。

l. 从**模式**菜单中选择**保存模式**。
“保存模式”窗口打开。

m. 输入以下信息:

属性	值
项目	_WCSamplesEntityBeansProject
数据包	com.ibm.commerce.sample.objects
类名	WCSSamplesSchema

然后单击**完成**，并关闭“模式浏览器”。

12. 一旦已经创建模式，就可以创建模式映射了。要创建 BONUS 表与 BonusBean 实体之间的映射，请执行以下操作:

a. 从 **EJB** 菜单中选择**打开到 > 模式映射**。“映射浏览器”打开。

b. 在“映射浏览器”中，从**数据仓库映射**菜单中选择**新建 EJB 组映射**。
“新建数据仓库映射”窗口打开。

c. 填写以下信息:

属性	值
名称	WCS 样本
EJB 组	WCSSamplesEntityBeans
模式	WCSSamples

并单击**确定**。

d. 在“数据仓库映射”面板中，单击 **WCS 样本**。

e. 在“持久类”面板中，单击 **Bonus**。

f. 从**表映射**菜单中选择**新建表映射 > 添加无继承表映射**。

g. 从**表**下拉列表中选择 **Bonus**，并单击**确定**。

h. 在“表映射”面板中，选择 **Bonus**，然后用鼠标右键单击它，并选择**编辑属性映射**。
“属性映射编辑器”打开。

i. 如下设置属性:

类属性	映射类型	表列
memberId	简单	MEMBERID
bonusPoint	简单	BONUSPOINT

并单击**确定**。

j. 从**数据仓库映射**菜单中选择**保存数据仓库映射**。
“保存数据仓库映射”打开。

k. 输入以下信息:

属性	值
项目	_WCSamplesEntityBeansProject
数据包	com.ibm.commerce.sample.objects
类名	WCSSamplesMap

然后单击**完成**，并关闭“映射浏览器”。

13. 一旦已经创建 BonusBean 实体，并已正确映射模式，则必须创建实体 bean 的访问 bean。此访问 bean 可使应用程序访问 Bonus 实体 bean 中包含的信息变得更为简化。VisualAge for Java 中的工具用于生成此访问 bean，这取决于已经创建的实体（特别的是，访问 bean 将仅使用已经提升到远程接口的方法）。要创建 Bonus 实体 bean 的访问 bean，请执行以下操作:

a. 在工作台中，先选择 EJB 选项卡，然后用鼠标右键单击 **Bonus** 企业 bean，并选择**添加 > 访问 bean**。
“创建访问 Bean”智能向导窗口打开（这可能会花费一定时间）。

b. 确保输入以下信息:

属性	值
EJB 组	WCSSamplesEntityBeans
企业 Bean	Bonus
访问 Bean 名称	BonusAccessBean
访问 Bean 类型	实体 Bean 的复制帮助函数

并单击**下一步**。

c. 从**为零参数构造函数选择主方法**下拉列表中选择 **findByMemberId(Long)**。

d. 对于 **init_argMemberId**，（在“初始属性”列中），将**转换器**设置为 com.ibm.commerce.base.objects.WCSStringConverter，并单击**下一步**。

e. 对于 bonusPoint，请确保选择 **CopyHelper**，将 **Converter** 值设置为 com.ibm.commerce.base.objects.WCSStringConverter，并单击**完成**。

注: 您无需对 memberId 字段做出任何修改。

f. 当显示“代码生成完成”消息时，请单击**确定**。

您可以通过切换到**项目**选项卡，展开 **_WCSamplesEntityBeansProject** 项目然后展开 **com.ibm.commerce.sample.objects**，来查看最新生成的代码。在数据包中将显示名为 BonusAccessBean 的新类。

14. 下一步是生成部署代码。
代码生成实用程序对 bean 进行分析，以确保满足 Sun Microsystems 的 EJB 规范，并确保遵循了特定于 EJB 服务器的规则。此外，对于每个选定的企业 bean，代码生成工具为主接口和远程接口生成主实现、EJBObject（远程）实现和实现类，并为 CMP bean 生成 JDBC 持久函数类和查找函数类。它还为通过 IIOP 进行的 RMI 访问生成所需的 Java ORB、存根和 tie 类，并为主接口和远程接口生成存根。
如果选择了包含 CMP 企业 bean 的 EJB 组，或选择了单个的 CMP 企业 bean，则还生成以下项:
    - 生成到持久函数类中的创建表字符串。
    - 映射到表或从表中映射出的持久函数实现

    要生成部署代码，请执行以下操作:

    a. 先选择 EJB 选项卡，然后在“企业 Bean”面板中，用鼠标右键单击 **Bonus** 企业 bean，并选择**生成部署代码**。代码的生成需要花费几分钟。

15. 在测试 Bonus 企业 bean 之前，您必须创建包含所有 WebSphere Commerce EJB 组以及新 WCSSamplesEntityBeans EJB 组的新 EJB 服务器。这些组必须运行在同一台服务器上以维持事务作用域。一旦已经创建新服务器，就必须启动它。
要创建并启动新的 EJB 服务器，请执行以下操作:

    a. 确保已经折叠所有的 EJB 组。

    b. 在“企业 Bean”面板中，选择所有的 WebSphere Commerce EJB 组和新 EJB 组（即选择所有以 WCS 开头的 EJB 组）。

    c. 先选择这些 EJB 组，然后用鼠标右键单击并选择**添加到 > 服务器配置**。“EJB 服务器配置”窗口打开（这可能需要花费一定时间）。

    d. 用鼠标右键单击新创建的 EJB 服务器（例如 **EJB 服务器（server2）**），选择**属性**并填写以下内容:

属性	DB2 **DB2 值**	Oracle **Oracle 值**
数据源	WebSphere Commerce DB2 DataSource *instance_name*	WebSphere Commerce Oracle DataSource *instance_name*
连接类型	<DataSource>	<DataSource>
用户名	*wcs_db_user_name*	*wcs_db_user_name*
密码	*wcs_db_password*	*wcs_db_password*
交易超时	1200	1200
交易非活动超时	600000	600000

并单击**确定**。

注: “数据源”的值必须与 *instance_name*.xml 文件中指定的数据源值相匹配。

根据开发机器的硬件（例如，处理器速度），您可能需要增加**事务超时**和**事务不活动** EJB 服务器属性的值。

e. 如果正在运行 WebSphere Test Environment，请如第 309 页的附录 A，『启动和停止 WebSphere Test Environment』中所述，停止它和持久名称服务器以及其它 EJB 服务器。

f. 如第 309 页的『启动和停止持久名称服务器』中所述，启动持久名称服务器。

g. 如第 310 页的『启动和停止 EJB 服务器』中所述，启动新的 EJB 服务器。

16. 一旦已经启动 EJB 服务器，就可以启动测试客户机了。使用测试客户机在数据库中创建新记录。要启动测试客户机并创建此记录，请执行以下操作:

a. 先选择 EJB 选项卡，然后用鼠标右键单击 **Bonus** 企业 bean，并选择**运行测试客户机**。

b. 在“EJB 查找”窗口中，单击**查找**。
Bonus 窗口打开。

c. 单击 **create(Long, Integer)**。在“详细信息”窗格中，填写以下内容:

属性	值
Long	–1000
Integer	100

并在“EJB 测试客户机”窗口中单击“调用”图标。

注: 第一个属性必须与任何注册用户的成员标识相匹配。您可通过查看 USERS 表确定用户标识。

17. 通过直接查询数据库，验证数据库记录创建是否正确。

    DB2 如果正在使用 DB2 数据库，请执行以下操作:

    a. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 控制中心**）
    b. 选择**交互式**选项卡。
    c. 输入 `connect to wcs_database_name` 并单击“执行”图标。
    d. 输入 `SELECT * FROM Bonus` 并单击“执行”图标。
    应返回以下值

列	值
MEMBERID	-1000
BONUSPOINT	100

    以上显示的记录是由 EJB 测试客户机创建的。

    Oracle 如果正在使用 Oracle 数据库，请执行以下操作:

    a. 打开 Oracle SQL Plus 命令窗口（**开始 > 程序 > Oracle - OraHome81> 应用程序开发 > SQL Plus**）。
    b. 在**用户名**字段，输入 Oracle 用户名。
    c. 在**密码**字段，输入 Oracle 密码。
    d. 在**主机字符串**字段，输入连接字符串。
    e. 在 SQL Plus 窗口，输入以下内容:

    ```
    select * from BONUS;
    ```

    并单击 Enter 键运行 SQL 语句。
    应当返回以下值

列	值
MEMBERID	-1000
BONUSPOINT	100

18. 通过执行以下操作，使用测试客户机验证 Bonus 企业 bean 是否可成功访问数据库记录:

    a. 在 Bonus 窗口中，选择**主**选项卡。
    b. 在“方法”面板中，单击 **findByMemberId(Long)**。
    c. 在 **Long** 字段中，输入 –1000 并单击“调用”图标。

d. 先选择**远程**选项卡，然后展开**方法**，单击 **getBonusPoint**，再单击“调用”图标。

“详细信息”面板显示整数结果 100。

e. 关闭 Bonus 和“EJB 测试客户机”窗口。

将 Bonus 实体 bean 与 MyNewControllerCmd 集成在一起

在前一部分中，您使用测试客户机测试了在 VisualAge for Java 中生成的新 Bonus 实体 bean。在本部分中，您将把 Bonus 实体 bean 与 MyNewControllerCmd 逻辑集成在一起。一旦更新 Java 代码，将更新 Sample.jsp 模板来创建一个允许对购物者奖金点数余额进行更新的接口。

集成 Bonus 实体 bean 需要执行以下高级步骤:

1. 修改 MyNewTaskCmdImpl 类的 performExecute 方法，以计算新的奖金点数并将点数保存到 BONUS 表中。
2. 将 getResources 方法添加到 MyNewControllerCmdImpl 类，以返回命令使用的资源列表。此方法附带包含，以用于访问控制目的。
3. 对新资源创建新的访问控制策略。
4. 修改 DataBeanSampleBean，以从 Bonus 实体 bean 的访问 bean 进行扩展。通过从访问 bean 扩展数据 bean，访问 bean 的所有属性都继承给数据 bean。
5. 修改 DataBeanSampleBean 中的方法。
6. 修改 WebSphere Test Environment 中小服务程序引擎的类路径，使其包含新的 _WCSamplesEntityBeansProject。
7. 修改 Sample.jsp 模板，以使用户可以输入奖金点数并显示结果。

修改 MyNewTaskCmdImpl 以用于奖金点数计算: MyNewTaskCmdImpl 用作 Bonus 实体 bean 与 MyNewControllerCmd 的集成点（因为 MyNewControllerCmd 调用 MyNewTaskCmd）。

要修改 MyNewTaskCmdImpl 以执行奖金点数计算，请执行以下操作:

1. 在 VisualAge for Java 工作台窗口中，展开 **_WCSamples** 项目。
2. 展开 **com.ibm.commerce.sample.commands** 数据包，然后选择 **MyNewTaskCmdImpl** 类查看其源代码。
3. 取消对以下 import 语句的注释:

   ```
   import com.ibm.commerce.sample.objects.*;
   ```

   保存工作（**Ctrl + S**）。
4. 选择 **MyNewTaskCmdImpl** 类的 **performExecute** 方法。

5. 在 performExecute 方法的源代码中，取消对第 3 段的注释。这将以下代码引入到方法中:

```
// use BonusAccessBean to update new bonus point

String newBonusPoint = null;
BonusAccessBean bb = new BonusAccessBean();
try {
     if (refNum != null) {
          bb.setInit_argMemberId(refNum);
          bb.refreshCopyHelper();
          oldBonusPoint = bb.getBonusPoint();
     }
} catch (javax.ejb.FinderException e) {
     try {
          bb = new BonusAccessBean(new Long(refNum),new Integer(0));
          oldBonusPoint = "0";
     } catch (javax.ejb.CreateException ec) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
               this.getClass().getName(), "performExecute");
     } catch (javax.naming.NamingException ec) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
               this.getClass().getName(), "performExecute");
     } catch (java.rmi.RemoteException ec) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
               this.getClass().getName(), "performExecute");
     }
} catch (javax.naming.NamingException e) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
               this.getClass().getName(), "performExecute");
} catch (java.rmi.RemoteException e) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
               this.getClass().getName(), "performExecute");
} catch (javax.ejb.CreateException e) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
               this.getClass().getName(), "performExecute");
}


try {
     if (oldBonusPoint != null) {
          int newBP =  Integer.parseInt(oldBonusPoint) + getTask_input2();
          newBonusPoint = Integer.toString( newBP );
          bb.setBonusPoint( newBonusPoint )   ;
          newBonusPoint=bb.getBonusPoint();
          bb.commitCopyHelper();
     }
} catch (javax.ejb.FinderException e) {
     throw new ECSystemException(ECMessage._ERR_FINDER_EXCEPTION,
               this.getClass().getName(), "performExecute");
} catch (javax.naming.NamingException e) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
               this.getClass().getName(), "performExecute");
} catch (java.rmi.RemoteException e) {
```

```
        throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
                  this.getClass().getName(), "performExecute");
    } catch (javax.ejb.CreateException e) {
        throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
                  this.getClass().getName(), "performExecute");
    }
```

保存工作。

添加 getResources 方法至 MyNewControllerCmdImpl 类: 在本部分中，您将新的 getResources 方法添加到 MyNewControllerCmdImpl 类。此方法返回命令在处理期间使用的资源列表。此方法是必需的，以用于资源级别的访问控制。

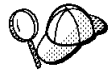
要添加 getResources 方法，请执行以下操作:

1. 先选择**项目**选项卡，然后展开 **_WCSamples** 项目。
2. 展开 **com.ibm.commerce.sample.commands** 数据包。
3. 选择 **MyNewControllerCmdImpl** 类查看其源代码。
4. 在源代码中，取消对访问控制段的注释。此段显示如以下代码片段中所示:

```
public AccessVector getResources() throws ECException {

// use UserRegistryAccessBean to check member reference number

   String refNum;
   String methodName="getResources";

   com.ibm.commerce.user.objects.UserRegistryAccessBean rrb =
      new com.ibm.commerce.user.objects.UserRegistryAccessBean();

     try {
      rrb = rrb.findByUserLogonId(getInputString());
          refNum = rrb.getUserId();
   }
   catch (javax.ejb.FinderException e) {

      throw new ECSystemException(ECMessage._ERR_BAD_USER_NAME,
         this.getClass().getName(),methodName);

   }
   catch (javax.naming.NamingException e) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
      this.getClass().getName(), methodName);
   }
   catch (java.rmi.RemoteException e) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
      this.getClass().getName(), methodName);
   }
   catch (javax.ejb.CreateException e) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
      this.getClass().getName(), methodName);
   }
```

```
   //find the Bonus bean for this user
String newBonusPoint = null;
   com.ibm.commerce.sample.objects.BonusAccessBean bb =
      new com.ibm.commerce.sample.objects.BonusAccessBean();
     try {
     if (refNum != null) {
          bb.setInit_argMemberId(refNum);
          bb.refreshCopyHelper();
      }
   }
catch (javax.ejb.FinderException e) {

   // The user doesn't have a Bonus object so return the container that
   // will hold the bonus object when it's created

   return new AccessVector(rrb);
   }
   catch (javax.naming.NamingException e) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
      this.getClass().getName(), methodName);
   }

   catch (java.rmi.RemoteException e) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
      this.getClass().getName(), methodName);
   }

   catch (javax.ejb.CreateException e) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
      this.getClass().getName(), methodName);
   }

   return new AccessVector(bb);

}
```

保存工作。（Ctrl+S）。



一旦保存上述代码段，VisualAge for Java 将把字段和访问函数从此特定视图中分离出来。注意：现在它们显示在 MyNewControllerCmdImpl 类的下方，并且方法标记有 M。

注：本教程出于简单性考虑，用此 getResources 方法创建资源对象。在实际应用中，首选用 validateParameters 方法创建资源对象，并将它们保存为实例变量。这样，可由 getResources 和 performExecute 方法重新使用这些对象。

设置新资源的访问控制策略: 附带提供了样本访问控制策略。此策略创建以下访问控制对象:

操作 创建的操作是 com.ibm.commerce.sample.commands.MyNewControllerCmd

操作组 创建的操作组是 MyNewControllerCmdActionGroup。此操作组仅包含一个操作: com.ibm.commerce.sample.commands.MyNewControllerCmd

资源类别
创建的资源类别是 com.ibm.commerce.sample.objects.BonusResourceCategory。此资源类别用于 Bonus 实体 bean。

资源组 创建的资源组是 BonusResourceGroup。此资源组仅包含上述资源类别。

策略 创建的策略是 AllUsersUpdateBonusResourceGroup。此策略仅在用户是 bonus 对象的“所有者”时，才允许用户对 Bonus bean 执行 MyNewControllerCmd 操作。例如，如果用户作为 wcsadmin 用户登录，则该用户仅可修改 wcsadmin 的奖金点数。

设置 AllUsersUpdateBonusResourceGroup 策略包含以下步骤:

1. 使用 acpload 命令装入 SampleACPolicy.xml 文件。
2. 使用 acpnlsload 命令装入 SampleACPolicy_***locale***.xml 描述。
3. 刷新策略注册表。注意: 仅当小服务程序引擎在装入访问控制策略时正在运行，才需要这一步。

要设置 AllUsersUpdateBonusResourceGroup 策略，请执行以下操作:

1. 在命令提示符下，切换到以下目录:
   drive:\WebSphere\CommerceServerDev\ bin
2. 要装入 SampleACPolicy.xml 文件，您必须发出 acpload 命令，此命令具有以下形式:

   ```
   acpload db_name db_user db_password inputXMLFile
   ```

   其中
   - *db_name* 是数据库的名称
   - *db_user* 是数据库用户名
   - *db_password* 是数据库密码
   - *inputXMLFile* 是包含了策略的 XML 文件名

   例如，可以发出以下命令:

   ```
   acpload VAJ_Demo user password SampleACPolicy.xml
   ```
3. 要装入策略描述，必须发出 acpnlsload 命令，此命令具有以下形式:

   ```
   acpnlsload db_name db_user db_password inputXMLFile
   ```

例如，可以发出以下命令:

```
acpnlsload VAJ_Demo user password SampleACPolicy_en_US.xml
```

4. 如果 WebSphere Test Environment 的小服务程序引擎当前正在运行，请停止它然后重新启动，以刷新策略注册表。

修改 DataBeanSampleBean 以用于奖金点数: 在本部分中，您将通过执行以下操作，修改 DataBeanSampleBean 以扩展 BonusAccessBean:

1. 在工作台中，展开 **com.ibm.commerce.sample.databeans** 数据包。
2. 通过执行以下操作，将新字段添加到数据 bean:
   a. 用鼠标右键单击 **DataBeanSampleBean** 类，并选择**添加 > 字段**。使用“创建字段”智能向导，按以下步骤中所述，创建新字段。如果需要关于创建字段的其它信息，请参阅第 194 页的『创建新字段』。
   b. 使用以下值将新字段添加到数据 bean:

属性名称	值
字段名称	task_output_oldBonusPoint
字段类型	String
初始值	留空。
访问修饰符	private
其它修饰符	全部保留不选中。
使用获取函数和设置函数方法进行访问	选中
获取函数	public
设置函数	public

单击**完成**。

3. 单击 **DataBeanSampleBean** 类查看其源代码。在源代码中，取消对以下 import 语句的注释:

```
import com.ibm.commerce.sample.objects.*;
```

保存工作。

4. 仍在 DataBeanSampleBean 类的源代码中，取消对第 1 段的注释，并注释掉第 2 段。这将 bean 更改为从 BonusAccessBean 扩展。作完这些修改后，代码如下显示:

```
/// Section 1 ////////////////////////////////////////////////

// Extend the databean to BonusAccessBean

public class DataBeanSampleBean
   extends com.ibm.commerce.sample.objects.BonusAccessBean
```

```
   implements SmartDataBean {


////////////////////////////////////////////////////////////

/// Section 2 ////////////////////////////////////////////////
/*
// Extend the databean to BonusAccessBean

     public class DataBeanSampleBean implements SmartDataBean {
*/

//
////////////////////////////////////////////////////////////
```

保存工作。

5. 选择 **setTask_output_userId(String)** 方法查看其源代码。对以下代码行定位:

```
public void setTask_output_userId(java.lang.String newTask_output_userId) {
task_output_userId = newTask_output_userId;
```

在上述行之后，输入以下代码以实例化新的 BonusAccessBean:

```
//////////////////////////////////////
// Section A : instantiate BonusAccessBean

if (task_output_userId != null)
     this.setInit_argMemberId(newTask_output_userId);

//////////////////////////////////////
```

保存工作。

6. 选择 **populate()** 方法查看其源代码。取消对第 2 段的注释以实例化 BonusAccessBean。这将以下代码引入到方法中:

```
setTask_output_oldBonusPoint(getRequestProperties().getString(
    "task_output_oldBonusPoint"));
```

修改类路径: 在可以在 WebSphere Test Environment 中测试修改过的命令之前，必须修改类路径以包含为新 Bonus 实体 bean 创建的新项目。要修改此类路径，请执行以下操作:

1. 从 VisualAge for Java 的**工作空间**菜单中选择**工具 > WebSphere Test Environment**。
   “WebSphere Test Environment 控制中心”将打开。
2. 单击**小服务程序引擎**。
3. 如果小服务程序引擎正在运行，请单击**停止小服务程序引擎**，然后单击**编辑类路径**。
4. 单击**全部选中**，然后单击**确定**。

修改 Sample.jsp 模板来更改奖金点数: 要修改显示模板，请执行以下操作:

1. 在文本编辑器中打开 Sample.jsp 和 Sample_All.jsp 文件。
2. 将 <!-- SECTION 4 --> 与 <!-- END OF SECTION 4 --> 标记符之间的代码第 4 段从 Sample_All.jsp 复制到 Sample.jsp 中。以下代码引入到 JSP 模板中:

```
<!-- SECTION 4 -->

<h1>
Bonus Administration
</h1>

<%
if (userId != null) {
%>

<B>
<UL>
   <LI> The bonus point before update is
      <%=testBean.getTask_output_oldBonusPoint()%>
   <LI> The bonus point after update is
      <%=testBean.getBonusPoint()%>
</UL>
</B>

<%
}
%>

<br>
<B>Input to command:</B><P>

<FORM NAME=Bonus ACTION="MyNewControllerCmd">
<TABLE>
   <TR>
         <TD>
             <B>Logon ID </B>
          </TD>
         <TD>
             <input type=text name=input1 value='<%=testBean.getInput1()%>'>
          </TD>
     </TR>
   <TR>
         <TD>
             <B>Bonus Point</B>
          </TD>
         <TD>
              <input type=text name=input2>
          </TD>
     </TR>
     <TR>
          <TD COLSPAN=2>
               <input type=submit>
          </TD>
```

```
    </TR>
</TABLE>
</FORM>

<!-- END OF SECTION 4 -->
```

保存 Sample.jsp 文件。

3. 由于新 Bonus bean 在访问控制下得到保护，且用户仅能对他们拥有的 bean 执行 MyNewControllerCmd 操作，所以用户必须登录。这样，您将在样本商店中使用登录功能部件以允许用户登录。这需要将 Sample.jsp 文件复制到商店的目录结构中，因为一旦您登录到商店，Web 控制器就将在商店的目录中搜索 Sample.jsp 文件。将 Sample.jsp 文件从:
   vaj_drive:\VAJava\Ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web
   复制到
   vaj_drive:\VAJava\Ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web*store_directory*。

   注: 对于样本商店，*store_directory* 的值为 InFashion。

4. 确保 WebSphere Test Environment 正在运行（请参阅第 309 页的附录 A，『启动和停止 WebSphere Test Environment』）。
5. 通过执行以下操作，作为 wcsadmin 用户登录:
   - 在浏览器中输入以下 URL:

     ```
     http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
         storeId=store_Id&catalogId=catalog_Id&langId=-1
     ```

   - 单击**注册**链接。
     显示“注册”或“登录”页面。
   - 在**电子邮件地址**字段中，输入 wcsadmin。
   - 在**密码**字段中，输入 wcsadmin 用户的密码，然后单击**登录**。

     注: 原始密码为 wcsadmin，但当作为安装过程的一部分执行 contractPublish 命令时更改了它。此步骤在以下文档中有描述:

     – Business 《*WebSphere Commerce Studio 商务开发者版安装指南*》

     – Professional 《*WebSphere Commerce Studio 专业开发者版安装指南*》

6. 完成登录后，在同一个浏览器中输入以下 URL:

   ```
   http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
       input1=wcsadmin&input2=1000
   ```

将向您显示一个页面，其中包含了先前的输出参数以及一份新表单（它使您可以更新用户的奖金点数余额）。显示的页面与以下屏幕快照类似:

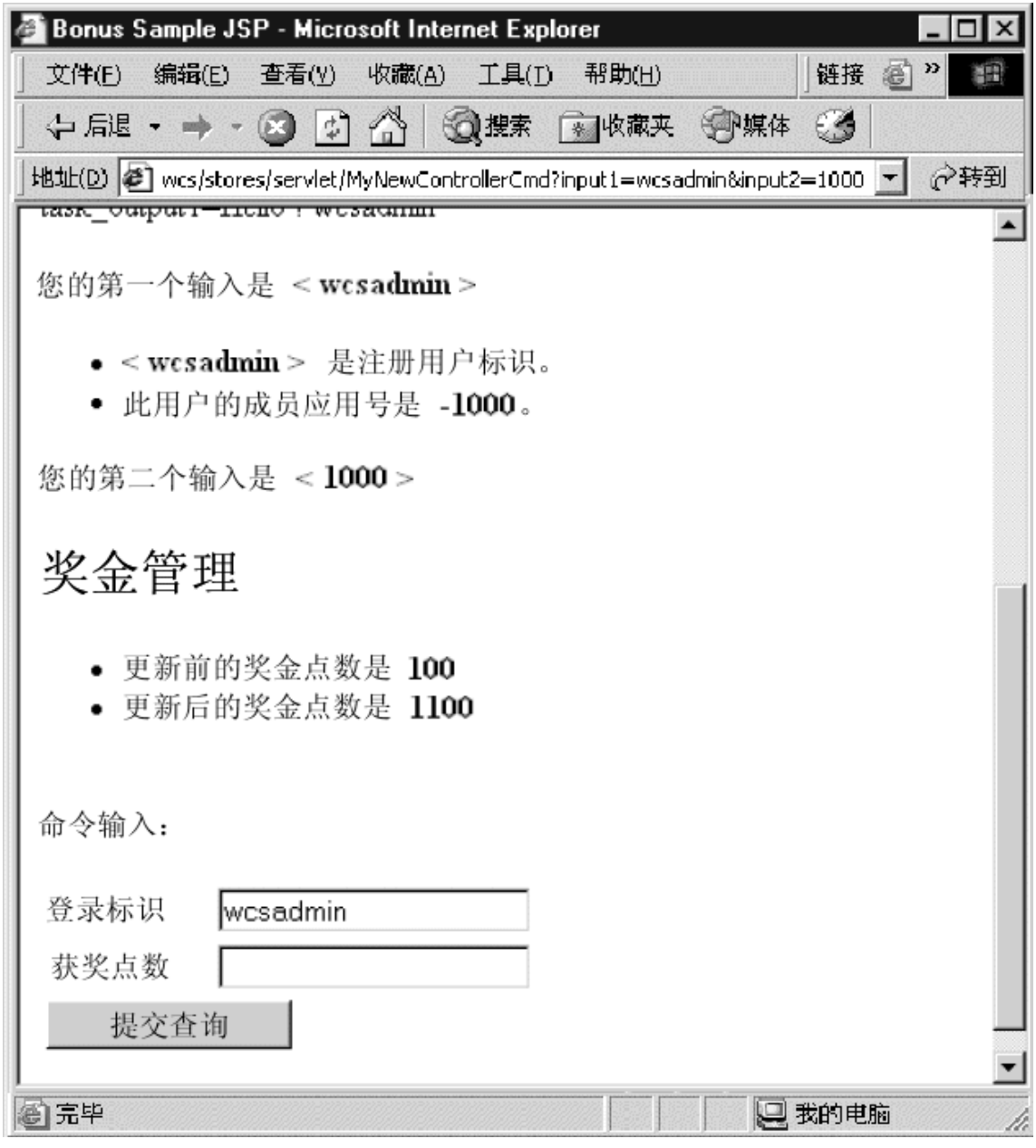


图 40.

7. 以代码当前状态创建代码版本。将版本命名为 mySample 1.7 Completed。当为代码创建版本时，请确保同时选择两个项目。如果需要关于为代码创建版本的详细信息，请参阅第 190 页的12。

（可选）使用 VisualAge for Java 中的调试器

本部分显示如何将断点添加到代码中并启动 VisualAge for Java 的调试器组件。这附带包含在其中以介绍调试器。关于此强有力功能的详细信息，请参阅 VisualAge for Java 的联机帮助。

本部分是可选的，因为它并未介绍本教程所需的任何新代码。相反，您可以创建断点，验证断点处某些变量的值然后除去断点。

添加断点至代码

要添加断点至代码，请执行以下操作:

1. 通过执行以下操作，确保已在工作空间中启用断点的使用:
   a. 从**窗口**菜单中，选择**调试**。确保已经选择**全局启用断点**。
2. 选择**项目**选项卡。
3. 展开 **_WCSamples** 项目。
4. 展开 **com.ibm.commerce.sample.commands** 数据包。
5. 展开 **MyNewTaskCmdImpl** 类。
6. 选择 **performExecute** 方法，查看其源代码。
7. 在“源码”窗格中，将光标放置（并单击）在以下代码行的开头:

   ```
   setTask_output1( "Hello ! " + getTask_input1() );
   ```

   将光标保留在此位置。
8. 从**编辑**菜单中，选择**断点**。
9. 在“配置: 断点 #1”窗口，选择**在选择的线程中**并单击**确定**。
10. 确保 WebSphere Test Environment 正在运行（请参阅第 309 页的附录 A, 『启动和停止 WebSphere Test Environment』）。
11. 通过执行以下操作，作为 wcsadmin 用户登录:
    - 在浏览器中输入以下 URL:

      ```
      http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
          storeId=store_Id&catalogId=catalog_Id&langId=-1
      ```
    - 单击**注册**链接。
      显示“注册”或“登录”页面。
    - 在**电子邮件地址**字段中，输入 wcsadmin。

- 在**密码**字段中，输入 wcsadmin 用户的密码，然后单击**登录**。

12. 登录过程完成后，输入以下 URL:

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
    input1=wcsadmin&input2=1000
```

13. 当到达代码中的断点时，“调试器”窗口打开。

验证变量的值

本节显示了在命令执行期间如何在不同的点验证 task_input1 和 task_output1 变量的值。

当由于 MyNewTaskCmdImpl 的 performExecute 方法中的断点，“调试器”打开时，切换到该窗口并请执行以下操作:

1. 在“变量”窗格中，展开**此**。
2. 单击 **task_input1**。
   在“值”窗格中，显示执行期间该变量在此点处的值。它应当显示 wcsadmin。

要验证 task_output1 变量的值是否按希望得到设置，您必须使调试器继续执行 performExecute 方法，到达在代码中设置变量处的点。这可以如下完成:

1. 使用步过功能，逐步执行代码。这可以用以下方法之一完成:
   - 从**选择**菜单中，选择**步过**。
   - 单击 F6。
   - 单击“步过”图标。

   为了达到此示例的目的，单击 F6 四次。这执行设置 task_output1 变量的代码。
2. 在“变量”窗格中，单击 task_output1。
   在“值”窗格中，显示执行期间该变量在此点处的值。它应当显示您好！wcsadmin。
3. 可以通过单击“继续”图标，完成执行此命令。

除去断点

要从代码中除去断点，请执行以下操作:

1. 切换回“工作台”窗口。
2. 确保可以查看 MyNewTaskCmdImpl 类 performExecute 方法的源代码。
3. 在“源码”窗格中，导航到以下代码行:

```
setTask_output1( "Hello ! " + getTask_input1() );
```

   注意: 在窗格的左边空白处，有一个蓝色点标记断点。
4. 双击蓝色点除去该断点。

断点现已从代码中除去。

在 WebSphere Test Environment 中将 MyNewControllerCmd 与样本商店相集成

本部分中，您将把调用 MyNewControllerCmd 的链接添加到样本商店的主页中。要执行此集成步骤，请执行以下操作:

1. 在文本编辑器中，打开 sidebar.jsp 文件。此文件位于以下目录:
   vaj_drive:\VAJava\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web*store_directory*\include
   其中 *vaj_drive* 是安装了 VisualAge for Java 的驱动器，*store_directory* 是样本商店目录的名称。
2. 通过在最后一个 </table> 标记前插入以下代码，将带有指向 MyNewControllerCmd 的链接的另一行添加到表中:

   ```
   <tr>
   <td>
   <a href = "MyNewControllerCmd?input1=wcsadmin&input2=1000">
        MyNewControllerCmd</a>
   </td>
   </tr>
   ```

   保存工作。
3. 确保 WebSphere Test Environment 正在运行。
4. 通过在浏览器中输入以下 URL 测试该集成:

   ```
   http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
        storeId=store_Id&catalogId=catalog_Id&langId=-1
   ```

   单击**注册**链接。
   显示“注册”或“登录”页面。
5. 在**电子邮件地址**字段中，输入 wcsadmin。
6. 在**密码**字段中，输入 wcsadmin 用户的密码，然后单击**登录**。
7. 用户登录后，单击边上导航窗格中的 **MyNewControllerCmd** 链接。显示样本 JSP。

   注: 如果旁边的导航窗格不显示该新链接，则可能已高速缓存 sidebar.jsp。从高速缓存中除去它，并重新装入该页面。关于删除已编译的 JSP 文件的信息，另见第 345 页的附录 C, 『VisualAge for Java 的技巧』。您可能需要在删除已编译的 JSP 文件后重新启动小服务程序引擎。

（可选）部署新的业务逻辑至远程 WebSphere Commerce Server

本节描述如何将新的业务逻辑部署到在远程 WebSphere Commerce Server 上运行的商店中。在开始这些部署步骤前，必须已经在远程 WebSphere Commerce Server 上创建了一个商店（基于“流行时尚”样本商店）。

部署过程包括在开发机器上执行的步骤，以及在目标 WebSphere Commerce Server 上执行的步骤。

为新命令逻辑创建 JAR 文件

您必须创建包含新命令和数据 bean 逻辑的 JAR 文件。要创建此 JAR 文件，请执行以下操作:

1. 如第 309 页的附录 A，『启动和停止 WebSphere Test Environment』中所述，停止 WebSphere Test Environment。
2. 先选择“项目”选项卡，然后选择 **_WCSamples** 项目。
3. 突出显示该项目，然后用鼠标右键单击并选择**导出**。
   “导出”智能向导打开。
4. 选择 **Jar 文件**，并单击**下一步**。
5. 在 Jar 文件字段中，输入以下内容:
   drive:\WebSphere\CommerceServerDev\mytemp\wcssamples_1.jar
   其中 *drive* 是安装了 Commerce Studio 的驱动器。
6. 如下选择属性:

属性	值
类	选中
java	不选中
资源	选中
bean	选中
.class 文件中包含调试属性	选中

   对于其它属性接受缺省值。
7. 单击**完成**。
8. 如果有提示消息，请确认新目录创建。

由于所创建的 JAR 文件不包含完整的数据包命名信息，因此您必须使用另一个封装实用程序（位于 VisualAge for Java 外部）来重新封装 JAR 文件。要重新封装此文件，请执行以下操作:

1. 在命令窗口中，导航到以下目录:
   drive:\WebSphere\CommerceServerDev\mytemp
2. 输入 mkdir temp1。
3. 输入 cd temp1。
4. 如下设置路径:
   set PATH=%PATH%;*drive*:\WebSphere\WebSphereStudio4\bin;
   其中 *drive* 是安装了 WebSphere Studio 的驱动器。
5. 输入 jar xvf ../wcssamples_1.jar
6. 输入 jar cvf ../wcssamples.jar * （注意: 已从名称中除去 _1）。

为新的 EJB 组创建 JAR 文件

您必须为新的 EJB 组创建 JAR 文件。要创建此文件，请执行以下操作:

1. 先选择 EJB 选项卡，然后用鼠标右键单击 **WCSSamplesEntityBeans** EJB 组，并选择**导出 > EJB 1.1 JAR**。
   “导出到 EJB 1.1 JAR 文件”智能向导打开。
2. 在 **JAR 文件**字段中，输入
   drive:\WebSphere\CommerceServerDev\mytemp\sampleEntityBeans_DT.jar
3. 如下选择属性:

属性	值
类	选中
java	不选中
资源	选中
目标数据库	DB2 如果您正在部署到 DB2 数据库，请选择 **DB2 for NT, V7.1**。 Oracle 如果您正在部署到 Oracle 数据库，请选择 **Oracle, V8**。
.class 文件中包含调试属性	选中

   对于其它属性接受缺省值。
4. 单击**完成**。

创建 JAR 文件。

JAR 文件命名时已经使用 “_DT” 后缀作为提醒符号，以提醒在将此 JAR 文件部署到 WebSphere Commerce 应用程序中之前，必须使用 WebSphere Application Server 提供的 “EJB 部署工具” 运行它。

为新的企业 bean 创建实现 JAR 文件

您必须为 Bonus 企业 bean 创建包含实现代码的 JAR 文件。要创建此文件，请执行以下操作:

1. 先选择“项目”选项卡，然后用鼠标右键单击 **_WCSSamplesEntityBeansProject** 并选择**导出**。
   “导出”智能向导打开。
2. 选择 **Jar 文件**，并单击**下一步**。
3. 在 **JAR 文件**字段中，输入
   drive:\WebSphere\CommerceServerDev\mytemp\sampleImpl_1.jar
4. 如下选择属性:

属性	值
类	选中
java	不选中
资源	选中
bean	选中
.class 文件中包含调试属性	选中

   对于其它属性接受缺省值。
5. 单击**完成**。

创建 JAR 文件。关闭 VisualAge for Java。

由于所创建的 JAR 文件不包含完整的数据包命名信息，因此您必须使用另一个封装实用程序（位于 VisualAge for Java 外部）来重新封装 JAR 文件。要重新封装此文件，请执行以下操作:

1. 在命令窗口中，导航到以下目录:
   drive:\WebSphere\CommerceServerDev\mytemp
2. 输入 mkdir temp2。
3. 输入 cd temp2。
4. 如下设置路径:
   set PATH=%PATH%;*drive*:\WebSphere\WebSphereStudio4\bin;
   其中 *drive* 是安装了 WebSphere Studio 的驱动器。
5. 输入 jar xvf ../sampleImpl_1.jar
6. 输入 jar cvf ../sampleImpl.jar * （注意: 已从名称中除去 _1）。

复制 JSP 文件至目标 WebSphere Commerce Server

必须将 Sample.jsp 和已更新的 sidebar.jsp 文件复制到部署代码的商店的正确目录中。

在将已更新的 JSP 模板复制到目标 WebSphere Commerce Server 之前，您可能希望在那台机器上制作原始 JSP 模板的备份副本。即：将现有文件重命名为 sidebar.jsp.bak。

要复制这些文件，请执行以下操作:

1. 在开发机器上，导航到以下目录:
   vaj_drive:\VAJava\Ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web*store_directory* 其中 *vaj_drive* 是安装了 VisualAge for Java 的驱动器，*store_directory* 是商店的目录名称。复制 Sample.jsp。
2. 将 Sample.jsp 粘贴到目标 WebSphere Commerce Server 上的以下目录中:

   ```
   drive:\WebSphere\AppServer\installedApps\
      WC_Enterprise_App_instance_name.ear\
      wcstores.war\store_directory
   ```

   其中 *drive* 是安装了 WebSphere Commerce 的驱动器，*store_directory* 是商店的目录名称，*instance_name* 是 WebSphere Commerce 实例的名称。
3. 在开发机器上，导航到以下目录:
   vaj_drive:\VAJava\Ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web*store_directory*\include
   复制 sidebar.jsp
4. 将 sidebar.jsp 粘贴到目标 WebSphere Commerce Server 上的以下目录中:

   ```
   drive:\WebSphere\AppServer\installedApps\
      WC_Enterprise_App_instance_name.ear\
      wcstores.war\store_directory\include
   ```

复制 JAR 文件至目标 WebSphere Commerce Server

您必须将 JAR 文件从开发机器复制到目标 WebSphere Commerce Server 上的适当目录中。

另外，有两个进一步的处理步骤必须对包含新 EJB 组的 JAR 文件执行。首先，必须对该文件运行来自 WebSphere Application Server 的 “EJB 部署工具”。然后，必须对该文件运行 modifyIsolationLevel 命令。因此在本 “复制 JAR 文件至目标 WebSphere Commerce Server 上” 步骤中，此特定 JAR 文件被存储在临时目录中以等待那些进一步的步骤。

要复制这些文件，请执行以下操作:

1. 在开发机器上，导航到以下目录:
   drive:\WebSphere\CommerceServerDev\ mytemp 并定位以下文件:
   - wcssamples.jar
   - sampleImpl.jar
   - sampleEntityBeans_DT.jar

   其中 *drive* 是安装了 WebSphere Commerce Studio，商务开发者版的驱动器。

   上述每个文件都必须复制到目标 WebSphere Commerce Server 上的特定目录中。仔细阅读以下步骤，以确保每个文件都存储在正确的位置中。
2. 将 wcssamples.jar 文件复制到目标 WebSphere Commerce Server 上的以下目录中:

   drive:\WebSphere\AppServer\InstalledApps\
   WC_Enterprise_App_*instance_name*.ear\wcstores.war\WEB-INF\lib

   其中 *drive* 是您安装了 WebSphere Commerce 商务版的驱动器，*instance_name* 是您的实例的名称（例如，demo）。
3. 创建临时库目录，您将在其中放置 JAR 文件。即: 创建以下目录:

   drive:\WebSphere\CommerceServer\temp\lib
4. 将 sampleImpl.jar 文件复制到目标 WebSphere Commerce Server 上的以下目录中:

   drive:\WebSphere\CommerceServer\temp\lib
5. 将 sampleEntityBeans_DT.jar 文件复制到目标 WebSphere Commerce Server 上的以下目录中:

   drive:\WebSphere\CommerceServer\temp

运行 EJB 部署工具

在测试服务器或生产服务器上可以成功运行企业 bean 之前，您需要为企业 bean 生成部署代码。WebSphere Application Server 提供的用于 Enterprise JavaBeans 的“部署工具”（另称为“EJB 部署工具”）提供可用于生成企业 bean 部署代码的命令行界面。

要运行此工具，请执行以下操作:

1. 在命令提示符下，导航到以下目录:

   drive:\WebSphere\CommerceServer\temp
2. 通过输入以下命令，临时性将该工具添加到系统路径中:

   PATH=*drive*:\WebSphere\AppServer\deploytool;%PATH%

3. 如下输入 ejbdeploy 命令:

   ```
   ejbdeploy EJBGroupJARFile WorkingDir OutputJARFile -nowarn -keep -35 -cp
   ClassPathOfDepJARFiles
   ```

   其中:

   - *EJBGroupJARFile* 是您希望为其生成部署代码的企业 bean 的 JAR 文件的全限定名称。在此情况下，它是 *drive*:\WebSphere\CommerceServer\temp\sampleEntityBeans_DT.jar。
   - *WorkingDir* 是代码生成所需临时文件的存储目录的名称。
   - *OutputJARFile* 是输出 JAR 文件的全限定名称。在此情况下，输入 *drive*:\WebSphere\CommerceServer\temp\sampleEntityBeans.jar。
   - -nowarn 是用于禁止警告消息和信息性消息的可选参数。
   - -keep 是用于在已经运行 ejbdeploy 命令后保留工作目录的可选参数。
   - -35 是一个强制性参数，它将对 CMP 实体 bean 使用与 “EJB 部署工具”（EJB 部署工具随 WebSphere Application Server 版本 3.5 一起提供）中所用相同的自顶向下的映射规则。
   - -cp *ClassPathOfDepJARFiles* 是任何从属 JAR 文件的类路径。在此情况下，输入

     ```
     "drive:\WebSphere\CommerceServer\temp\lib\sampleImpl.jar;
     drive:\WebSphere\AppServer\installedApps\
     WC_Enterprise_App_instanceName.ear\lib\wcsejbimpl.jar"
     ```

     注: 您必须用引号（“”）括住类路径值。

修改 Bonus bean 的事务隔离级别

在本步骤中，您使用 modifyIsolationLevel 命令修改 Bonus bean 的事务隔离级别。此工具还将 bean 的隔离级别设置为您特定类型的数据库的必需级别。

要运行 modifyIsolationLevel 命令，请执行以下操作:

1. 在目标 WebSphere Commerce Server 上，打开命令窗口。
2. 切换到以下目录:
   drive:\WebSphere\CommerceServer\bin
3. 您必须发出 modifyIsolationLevel 命令，此命令的一般语法如下:

   ```
   modifyIsolationLevel -jarFile jar_file_name.jar
      -logFile log_file_name -dbType  db_type
   ```

   其中

   - *jar_file_name.jar* 是包含定制代码的 JAR 文件名称
   - *log_file_name* 是应当作为记录信息的位置的全限定文件名

- *db_type* 是正在使用的数据库类型。输入 DB2 或 ORACLE

以下是指定了所有值的 modifyIsolationLevel 命令示例:

```
modifyIsolationLevel -jarFile
   D:\WebSphere\CommerceServer\temp\sampleEntityBeans.jar
   -logFile D:\WebSphere\CommerceServer\instances\demo\logs\output.log
   -dbType DB2
```

如果没有异常显示在命令窗口中，则命令已经成功运行。完成后，请注意在部署 JAR 文件上的时间戳记已经更改。

注: 参数名称区分大小写。也就是说，jar**F**ile 与 jar**f**ile 不一样。确保正确输入参数名称。

更新目标数据库

由于您正在部署新的业务逻辑至与 WebSphere Test Environment 使用不同数据库的目标 WebSphere Commerce Server，因此必须更新目标数据库，以反映对命令注册表所作的更改并创建 BONUS 表。

DB2 如果正在使用 DB2 数据库，请执行以下操作更新目标数据库:

1. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 控制中心**）。
2. 从**工具**菜单中，选择**工具设置**。
3. 选择**使用语句终止字符**复选框，并确保指定的字符是分号（**;**）
4. 先选择“脚本”选项卡，然后通过在脚本窗口中输入以下信息，在 URLREG 表中创建必需的条目:

   ```
   connect to your_database_name;
   insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS,
       DESCRIPTION, AUTHENTICATED) values ('MyNewControllerCmd',0,
       'com.ibm.commerce.sample.commands.MyNewControllerCmd',0,
       'This is a new controller command for test/education purposes.',
       null)
   ```

   其中 *your_database_name* 是数据库的名称，并单击“执行”图标。
   此命令供所有商家使用（由 STOREENT_ID 的值为 0 指示）。
5. 通过在脚本窗口中输入以下内容，在 VIEWREG 表中创建条目:

   ```
   insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
      CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE) values
      ('SampleViewTask',-1, 0, 'com.ibm.commerce.command.ForwardViewCommand',
      'com.ibm.commerce.command.HttpForwardViewCommandImpl',
      'docname=Sample.jsp','This is a sample view for the Bonus Point
      exercise', 0, null)
   ```

   并单击“执行”图标。

6. 通过在脚本窗口中输入以下内容，创建 BONUS 表:

```
CREATE TABLE Bonus (MEMBERID BIGINT NOT NULL,
BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
constraint f_memberid foreign key (MEMBERID)
references users (users_id) on delete cascade)
```

   单击“执行”图标。

上述步骤在命令注册表中注册了 MyNewControllerCmd 和 SampleViewTask，并创建了 BONUS 表。

Oracle 如果正在使用 Oracle 数据库，请执行以下操作更新目标数据库:

1. 打开 Oracle SQL Plus 命令窗口（**开始 > 程序 > Oracle > 应用程序开发 > SQL Plus**）。
2. 在**用户名**字段，输入 Oracle 用户名。
3. 在**密码**字段，输入 Oracle 密码。
4. 在**主机字符串**字段，输入连接字符串。
5. 通过在 SQL Plus 窗口中输入以下信息，在 URLREG 表中创建必需的条目:

```
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS,
    DESCRIPTION, AUTHENTICATED) values ('MyNewControllerCmd',0,
    'com.ibm.commerce.sample.commands.MyNewControllerCmd',0,
    'This is a new controller command for test/education purposes.',
    null);
```

   并按 Enter 键运行 SQL 语句。

6. 通过在 SQL Plus 窗口中输入以下内容，在 VIEWREG 表中创建条目:

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
   CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE) values
   ('SampleViewTask',-1, 0, 'com.ibm.commerce.command.ForwardViewCommand',
   'com.ibm.commerce.command.HttpForwardViewCommandImpl',
   'docname=Sample.jsp','This is a sample view for the Bonus Point
   exercise', 0, null);
```

   并按 Enter 键运行 SQL 语句。

7. 通过在 SQL Plus 窗口中输入以下内容，创建 BONUS 表:

```
CREATE TABLE Bonus (MEMBERID NUMBER NOT NULL,
BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
constraint f_memberid foreign key (MEMBERID)
references users (users_id) on delete cascade);
```

   并按 Enter 键运行 SQL 语句。

8. 输入以下命令提交数据库的更改:

```
commit;
```

并按 Enter 键运行 SQL 语句。

装入新资源的访问控制策略

本教程中，已创建一个新企业 bean（Bonus bean），它是可保护资源。这样，就有一个与该资源相关的访问控制策略。您也创建了可由所有用户执行的新控制器命令。当在开发机器上工作时，您已将访问控制策略信息装入到该机器上。现在也必须将相同的访问控制策略信息装入到目标 WebSphere Commerce Server 上。

要设置访问控制策略，请执行以下操作:

1. 将以下 CD 插入您的光盘驱动器:
   - Business WebSphere Commerce 商务版，V5.4 Disk 2 CD
   - Professional WebSphere Commerce 专业版，V5.4 Disk 2 CD
2. 切换到以下目录:

   CD_drive:\repository\samples\programguide\
3. 在该目录中定位以下文件:
   - SampleCmdACPolicy.xml
     此 XML 文件包含由新控制器命令使用的访问控制策略。
   - SampleACPolicy.xml
     此 XML 文件包含创建新的企业 bean 时使用的访问控制策略。
   - SampleACPolicy_*locale*.xml
     其中 *locale* 是语言标识。此 XML 文件包含访问控制策略描述。
4. 将上述三个文件复制到以下目录中:
   drive:\WebSphere\CommerceServer\xml\policies\xml
5. 要装入 SampleCmdACPolicy.xml 文件，请使用命令提示符切换到以下目录:
   drive:\WebSphere\CommerceServer\bin
   您必须发出 acpload 命令，此命令具有以下形式:

   acpload *db_name db_user db_password inputXMLFile*

   其中
   - *db_name* 是数据库的名称
   - *db_user* 是数据库用户名
   - *db_password* 是数据库密码
   - *inputXMLFile* 是包含策略的 XML 文件名称。

   例如，可以发出以下命令:

   acpload mall user password SampleCmdACPolicy.xml

6. 通过发出 acpload 命令将 SampleACPolicy.xml 文件指定为输入文件，来装入 SampleACPolicy.xml 文件。例如，可以发出以下命令:

   ```
   acpload mall user password SampleACPolicy.xml
   ```

7. 要装入策略描述，必须发出 acpnlsload 命令，此命令具有以下形式:

   ```
   acpnlsload db_name db_user db_password inputXMLFile
   ```

   例如，可以发出以下命令:

   ```
   acpnlsload mall user password SampleACPolicy_en_US.xml
   ```

注意: 通常为了使策略生效会需要刷新注册表。在此情况下，由于要停止并重新启动 WebSphere Application Server 中的 WebSphere Commerce Server 应用程序作为企业 bean 的部分部署步骤，因此不需要执行此步骤。如果不是这种情况，您可以使用 WebSphere Commerce 中的管理控制台，更新注册表。关于管理控制台的更多信息，请参阅 WebSphere Commerce 的联机帮助。

400 如果正在部署到在 iSeries 机器上运行的 WebSphere Commerce 实例，则使用不同的命令来装入访问控制策略信息。使用 LODWCSAC 命令（而不是 acpload 命令），使用 LODWCSACD 命令（而不是 acpnlsload 命令）。

LODWCSAC 命令的语法是:

```
LODWCSAC DATABASE(dbName) SCHEMA(schemaName)
PASSWD(instancePassword) INSTROOT('instanceRoot')
INFILE('inputFile')
```

其中

- *dbName* 是在 WRKRDBDIRE 命令中定义的其关系数据库的名称。
- *schemaName* 是此实例的数据库模式的名称（这与实例名称相同）。
- *instancePassword* 是实例密码。
- *instanceRoot* 是实例根。例如，实例根是

  /QIBM/UserData/WebCommerce/instances/*instanceName*
- *inputFile* 是具有访问策略的输入 XML 文件的全限定名称。

LODWCSACD 命令的语法是:

```
LODWCSACD DATABASE(dbName) SCHEMA(schemaName)
PASSWD(instancePassword) INSTROOT('instanceRoot')
INFILE('inputFile')
```

可以将访问控制策略的 XML 文件存储在以下目录中:

/QIBM/UserData/WebCommerce/instances/*instanceName*

此外，在访问控制策略的 XML 文件中，必须使用到访问控制 DTD 的全路径。访问控制策略的 DTD 存储在 /QIBM/ProdData/WebCommerce/xml/policies/dtd 目录中。

例如，如果将教程的访问控制策略部署到在 iSeries 机器上运行的 WebSphere Commerce 实例，那么必须在 XML 文件中将教程的访问控制策略的 DTD 指定从:

```
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">
```

修改为

```
<!DOCTYPE Policies SYSTEM "/QIBM/ProdData/WebCommerce/
xml/policies/dtd/accesscontrolpolicies.dtd">
```

有关 WebSphere Commerce 访问控制模型的更多信息，请参阅《*WebSphere Commerce 访问控制指南*》。

从 WebSphere Application Server 中导出当前的企业应用程序

在此步骤中，您从 WebSphere Application Server 中导出当前的企业应用程序，这样您就可以在应用程序组装工具中打开它了。

要导出当前的企业应用程序，请执行以下操作:

1. 创建一个目录，当前的企业应用程序将导出到其中。注意: 不要将它命名为“temp”目录（因为不应冒着在日常系统维护中文件被删的风险），直至确信您对定制代码一旦已经部署即起作用的方式感到满意。要创建此目录，请在命令提示符下执行以下操作:
   a. 导航到以下目录:

      drive:\WebSphere\CommerceServer\

   b. 输入以下命令:

      ```
      mkdir working
      ```

      这创建 *drive*:\WebSphere\CommerceServer\working 目录。
2. 打开 WebSphere Application Server 管理控制台。
3. 展开 **WebSphere 管理域**。
4. 展开**企业应用程序**。
5. 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 **demo** 应用程序并选择**导出应用程序**。
6. 在**导出目录**字段中，输入 *drive*:\WebSphere\CommerceServer\working。这将整个应用程序（包括所有资源）导出到 WC_Enterprise_App_*instanceName*.ear 文件中（其中 *instanceName* 是 WebSphere Commerce 实例的名称）。
7. 单击**确定**。导出应用程序可能会花费几分钟。

导出企业应用程序的 XML 配置信息

您必须导出企业应用程序的 XML 配置信息。要导出此信息，您将使用 WebSphere Application Server 提供的 XMLConfig 命令行实用程序。

要导出此配置信息，请执行以下操作:

1. 将 was.export.app.xml 文件从以下目录:

   drive:\WebSphere\CommerceServer\xml\config

   复制到以下目录中:

   drive:\WebSphere\CommerceServer\working

2. 在文本编辑器中打开 was.export.app.xml 文件。在此文件中，将 $Enterprise_Application_Name$ 所有出现的地方替换为

   WebSphere Commerce Enterprise Application - *instanceName*

   其中 *instanceName* 是 WebSphere Commerce 实例的名称（例如 demo）。保存此文件。

   注: 您正在插入的值必须与显示在 WebSphere 高级管理控制台中的实例信息匹配。

3. 在命令提示符下，导航到以下目录:

   drive:\WebSphere\CommerceServer\working

4. 通过输入以下命令，调用 XMLConfig 工具来执行部分的导出:

   ```
   xmlConfig -export OutputFile.xml -partial was.export.app.xml
      -adminNodeName wasHostName
   ```

   其中 *wasHostName* 是 WebSphere Application Server 中包含当前企业应用程序的节点的名称。另外，OutputFile.xml 是作为运行此命令的结果而创建的文件的名称，was.export.app.xml 是您在步骤 2 中修改的文件。

在导出有关当前企业应用程序中的企业 bean 的信息之后，必须向 XML 文件添加描述 Bonus bean 的新节。

要添加描述 Bonus bean 的新节，请执行以下操作:

1. 导航到以下目录:

   drive:\WebSphere\CommerceServer\working

2. 在文本编辑器中打开 OutputFile.xml 文件。
3. 定位 <ear-file-name> 标记，并将该值替换为以下值:

   ```
   drive:\WebSphere\CommerceServer\working\
      WC_Enterprise_App_instanceName.ear
   ```

4. 您还必须为 Bonus bean 添加入新的节，如以下样本所示:

   ```
   <ejb-module name="WCSSamplesEntityBeans">
      <jar-file>sampleEntityBeans.jar</jar-file>
      <module-install-info>
         <application-server-full-name>/NodeHome:$hostName$/EJBServerHome:
            WebSphere Commerce Server - demo/</application-server-full-name>
      </module-install-info>
      <ejb-module-binding>
         <data-source>
            <jndi-name>jdbc/WebSphere Commerce DB2 DataSource demo</jndi-name>
            <default-user>user</default-user>
            <default-password>password</default-password>
         </data-source>
         <enterprise-bean-binding name="Bonus_Binding">
   ```

```
            <jndi-name>democom/ibm/commerce/sample/objects/Bonus</jndi-name>
        </enterprise-bean-binding>
    </ejb-module-binding>
</ejb-module>
```

其中

- *user* 是您的数据库用户名。
- *password* 是数据库用户的密码。

注:

a. 以上示例中的断行仅为了显示。

b. 请确保 **$hostName$** 值与当前管理节点服务器名称匹配。另外，请确保在此行中没有回车符。

c. `<application-server-full-name>` 规范不能跨多行。

d. 如果您正在使用 Oracle 数据库，您必须修改数据源信息。更改取自先前代码片段的以下行:

```
<jndi-name>jdbc/WebSphere Commerce DB2 DataSource demo</jndi-name>
```

为以下行:

```
<jndi-name>jdbc/WebSphere Commerce Oracle DataSource demo</jndi-name>
```

e. 当部署您自己的应用程序（例如在此教程之外）时，请确保 XML 文件中指定的企业 bean 的 JNDI 名称与 VisualAge for Java 中使用的 JNDI 名称匹配，但是前面附有 WebSphere Commerce 实例名称。

5. 保存 `OutputFile.xml` 文件。

组装新的 EJB 组到企业应用程序中

在本步骤中，您在应用程序组装器工具中打开企业应用程序。一旦企业应用程序在该工具中打开，您就可以执行以下操作，以将新的 Bonus bean 添加到企业应用程序中:

1. 导入新的 Bonus bean。新 EJB 组的 JAR 文件存储在企业应用程序的 “EJB 模块” 部分中。
2. 设置 Bonus bean 的类路径，以包含该实现 JAR 文件。
3. 将该实现 JAR 文件添加到应用程序中。此 JAR 文件存储在企业应用程序的 “文件” 部分中。
4. 为 Bonus bean 中包含的方法设置 WebSphere Application Server 安全性。

要将新的 EJB 组组装到企业应用程序中，请执行以下操作:

1. 通过执行以下操作，备份当前的企业应用程序:

a. 在命令提示符下，导航到以下目录:

   drive:\WebSphere\CommerceServer\working

b. 输入以下命令:

   ```
   copy WC_Enterprise_App_instanceName.ear
      WC_Enterprise_App_instanceName.ear.bak
   ```

2. 打开 WebSphere Application Server 管理控制台。
3. 从**文件**菜单中，选择**工具 > 应用程序组装工具**。
4. 如果打开了“欢迎”窗口，选择**取消**关闭该窗口。
5. 通过执行以下操作，打开要处理的企业应用程序:
   a. 从**文件**菜单中，选择**打开**。
   b. 在**文件名**字段中，输入:

      ```
      drive:\WebSphere\CommerceServer\working\
         WC_Enterprise_App_instanceName.ear
      ```

      并单击**打开**。等待应用程序打开后，继续到后续步骤。这需要花费几分钟。
6. 用鼠标右键单击 **EJB 模块**，并选择**导入**。
7. 在**文件名**字段中，输入

   ```
   drive:\WebSphere\CommerceServer\temp\sampleEntityBeans.jar
   ```

   并单击**打开**。在“确认值”窗口中，单击**确定**。
8. 一旦导入了 sampleEntityBeans.jar 文件，请滚动到 **WCSSamplesEntityBeans** EJB 组，并选择此组。
   有关该组的信息显示在右侧的窗格中。
9. 在新企业 bean 的类路径字段中，输入任何从属 JAR 文件。在此情况下，输入

   ```
   lib/sampleImpl.jar lib\wcsejbimpl.jar
   ```

10. 单击**应用**。
11. 通过执行以下操作，将 sampleImpl.jar 文件添加到应用程序中:
    a. 用鼠标右键单击企业应用程序的**文件**节点，并选择**添加文件**。企业应用程序的**文件**节点位于层次结构树的底部附近。注意: 对于企业应用程序中的组件有其它的“文件”节点，但您必须选择整个应用程序的“文件”节点。）
    b. 在“添加文件”窗口中，单击**浏览**。
    c. 浏览到 *drive*:\WebSphere\CommerceServer\temp。
    d. 先突出显示此目录，然后单击**选择**。

e. 返回到“添加文件”窗口。注意，显示出 *drive*:\WebSphere\CommerceServer\temp 目录的内容。突出显示 lib 目录。
lib 目录的内容显示在右边的窗格中。

f. 在右边的窗格中，选择 sampleImpl.jar 文件并单击**添加**。然后该文件显示在“选定的文件”窗格中。

g. 单击**确定**。

12. 通过执行以下操作，为 Bonus bean 配置安全性:

a. 先展开“EJB 模块”节点，然后找到并展开 **WCSSamplesEntityBeans** 节点。

b. 展开**实体 Bean**。

c. 展开 **Bonus**

d. 单击**方法扩展**，然后在右边的窗格中执行以下操作:

1) 单击**高级**选项卡。

2) 确保已经选择**安全性身份**。

3) 对于每种方法，确保已经选择**使用 EJB 服务器的身份**。

4) 单击**应用**（如果您做出了任何修改）。

e. 在左侧导航窗格中，用鼠标右键单击 WCSamplesEntityBeans EJB 组下的**安全性角色**，选择**新建**，然后执行以下操作:

1) 在**名称**字段中，输入 WCSecurityRole，并单击**应用**。注意，如果已经存在此角色，您就无需执行此步骤了。

f. 在左侧导航窗格中，用鼠标右键单击 WCSamplesEntityBeans EJB 组下的**方法许可**，选择**新建**，然后执行以下操作:

1) 在**方法许可名称**字段中，输入 WCMethodPermission

2) 在**方法**选择区域中，单击**添加**。
“添加方法”窗口打开。

3) 展开 **sampleEntityBeans.jar**，再展开 **Bonus**，然后展开每个方法**本地**和**远程**列表。

4) 按住 Shift 键保持不动，并选择所有的本地方法，单击**确定**。

5) 重复方法选择过程同样地添加远程方法（如果存在任何远程方法）。

6) 在“角色”选择区域中，单击**添加**，选择 WCSecurityRole 并单击**确定**。

7) 单击**应用**进行每次更新。

13. 从**文件**菜单中，选择**保存**。

14. 关闭应用程序组装器工具。

完成本步骤后，您就已经创建包含所有先前逻辑以及新业务逻辑的新企业应用程序。这些逻辑正是新修改的 WC_Enterprise_App_*instanceName*.ear 文件中包含的所有内容。

导入新的企业应用程序到 WebSphere Application Server 中

以下是将新企业应用程序导入到 WebSphere Application Server 中所涉及到的高级步骤:

1. 停止当前在 WebSphere Application Server 中运行的企业应用程序，然后除去它。这些步骤在 WebSphere Application Server 管理员控制台中执行。
2. 使用 XMLConfig 命令行实用程序导入新的应用程序。
3. 刷新 WebSphere Application Server 管理员控制台，然后启动新的企业应用程序。

每个这些步骤在以下部分有更详细描述。

停止并除去当前的企业应用程序

要从 WebSphere Application Server 中停止并除去当前的企业应用程序，请执行以下操作:

1. 打开 WebSphere Application Server 管理控制台。
2. 展开 **WebSphere 管理域**。
3. 展开**节点**。
4. 展开 *nodeName*（其中 *nodeName* 是节点的名称）。
5. 展开**应用程序服务器**。
6. 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 **WebSphere Commerce Server -** *instanceName* 并选择**停止**。
7. 展开**企业应用程序**。
8. 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 **WebSphere Commerce 企业应用程序 - demo** 应用程序并选择**停止**。
9. 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 **WebSphere Commerce 企业应用程序 - demo** 应用程序并选择**除去**。
10. 当提示指出应用程序是否应导出时，选择**否**。

使用 XMLConfig 导入新的企业应用程序

要使用 XMLConfig 命令行实用程序导入新的企业应用程序，请执行以下操作:

1. 导航到以下目录:

   drive:\WebSphere\CommerceServer\working

2. 在命令提示符下，输入以下命令以将企业应用程序导入到 WebSphere Application Server 中:

   ```
   xmlConfig -import OutputFile.xml -adminNodeName was_hostname
   ```

   其中 *was_hostname* 是 WebSphere Application Server 中包含当前应用程序的节点的名称。

注: 400 如果正在部署到在 iSeries 上运行的 WebSphere Commerce 实例，那么在导入应用程序之后，您将必须执行附加的步骤以修改目录许可权。有关如何修改这些许可权的详细信息，请参阅第 342 页的『导入企业应用程序』。

启动新的企业应用程序

当使用 XMLConfig 命令行实用程序导入了新的企业应用程序后，您可以使用 WebSphere Application Server 管理员控制台来执行刷新并然后启动新的应用程序。

要刷新控制台并启动新的应用程序，请执行以下操作:

1. 打开 WebSphere Application Server 管理控制台。
2. 展开 **WebSphere 管理域**
3. 突出显示**节点**。
4. 单击**刷新选定的子树**图标。
5. 通过执行以下操作，启动您的 WebSphere Commerce 应用程序:
   - 展开**应用程序服务器**。
   - 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 **WebSphere Commerce Server -** *instanceName*，并选择**启动**。

测试 MyNewControllerCmd

下一步是测试在 WebSphere Application Server 环境中运行的商店中的新逻辑。要测试 MyNewControllerCmd，请执行以下操作:

1. 通过在浏览器中输入以下 URL 测试该集成:

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

   其中 *store_Id* 是商店的标识，*catalog_Id* 是商店产品目录的标识。
2. 单击**注册**链接。
   显示“注册”或“登录”页面。
3. 在**电子邮件地址**字段中，输入 wcsadmin。
4. 在**密码**字段中，输入此站点的 wcsadmin 标识的密码然后单击**登录**。

5. 用户登录后，单击边上导航窗格中的 **MyNewControllerCmd** 链接。显示样本 JSP。

如果已更新的 sidebar.jsp 文件未显示，请执行以下操作清除高速缓存:

1. 从以下目录中删除高速缓存文件:
   drive:\WebSphere\CommerceServer\instances*instanceName*\cache
2. 从以下目录中删除任何相关的高速缓存文件:

   drive:\WebSphere\AppServer\temp*hostName*\
   WebSphere_Commerce_Server_*instanceName*\
   WebSphere_Commerce_Enterprise_Application_-_*instanceName*\
   wcstores.war*storeName*

   drive:\WebSphere\AppServer\temp*hostName*\
   WebSphere_Commerce_Server_*instanceName*\
   WebSphere_Commerce_Enterprise_Application_-_*instanceName*\
   wcstores.war
3. 清除浏览器高速缓存。

如果 JSP 页面编译过长，则该页面可能不显示。在此情况下，重新装入该页面。

第 10 章 修改并扩展现有的业务逻辑

以下教程显示如何扩展或修改现有的 WebSphere Commerce 业务逻辑。

扩展现有的控制器命令

在本部分中，您将扩展现有的 OrderProcess 控制器命令，从而使购买已经积累的总奖金点数显示在订单确认页面上。

注: 本教程的目的是显示修改现有控制器命令的过程。它并不是为显示修改购物流程中订单处理步骤的最好方法。事实上，WebSphere Commerce 提供可用于修改购物流程中订单处理步骤的 `ExtOrderProcess` 任务命令。

开始本教程之前

您应已经完成第 183 页的第 9 章，『教程: 创建新的业务逻辑』中的步骤。

以下列表概述了扩展 OrderProcess 命令所需的步骤:

1. 创建存储定制代码的新数据包。请记住（命令和数据 bean 的）所有定制代码必须存储在与 WebSphere Commerce 代码分离的项目和数据包中。
2. 创建扩展现有 `OrderProcessCmdImpl` 命令的新 `OrderProcessCmdBonusImpl` 类。
3. 将字段和方法添加到 `OrderProcessCmdBonusImpl` 类。
4. 修改命令注册表以使用 `OrderProcessCmdBonusImpl` 类
5. 修改 confirmation.jsp 模板以显示新的业务逻辑。
6. 在 WebSphere Test Environment 中测试新的业务逻辑。
7. （可选）部署新业务逻辑至远程 WebSphere Commerce Server 上的商店。

为 `OrderProcessCmdBonusImpl` 创建新数据包

要创建存储 `OrderProcessCmdBonusImpl` 命令的新数据包，请执行以下操作:

1. 在 VisualAge for Java Workbench 窗口中，请确保您选择了“项目”选项卡。
2. 用鼠标右键单击 **_WCSamples** 项目并选择**添加 > 数据包**。
   “添加数据包”智能向导打开。
3. 确保已经启用**创建新数据包名为**单选按钮，并输入 `com.ibm.commerce.sample.order`。

4. 单击**完成**。

创建 OrderProcessCmdBonusImpl 类

要创建新的 OrderProcessCmdBonusImpl 类，请执行以下操作:

1. 用鼠标右键单击 com.ibm.commerce.sample.order 数据包，并选择**添加 > 类**。“创建类”智能向导打开。
2. 请确保选择了**创建新类**单选按钮。
3. 在**类名**字段中，输入 OrderProcessCmdBonusImpl。
4. 要指定超类，请单击**浏览**，然后在**模式**字段中，输入 com.ibm.commerce.order.commands.OrderProcessCmdImpl，并单击**确定**。
5. 单击**下一步**。
6. 要指定应导入的数据包，请单击**添加数据包**。在**模式**字段中，输入以下数据包:
   - com.ibm.commerce.datatype 并单击**添加**
   - com.ibm.commerce.exception 并单击**添加**
   - com.ibm.commerce.order.commands 并单击**添加**
   - com.ibm.commerce.order.objects 并单击**添加**
   - com.ibm.commerce.ras 并单击**添加**
   - com.ibm.commerce.server 并单击**添加**
   - com.ibm.commerce.sample.objects 并单击**添加**
   - javax.ejb 并单击**添加**
   - java.io 并单击**添加**
   - java.math 并单击**添加**，然后单击**关闭**。
7. 要指定该类应实现的接口，请单击**添加**。在“模式”字段中，输入以下接口:
   - OrderProcessCmd 并单击**添加**，然后单击**关闭**。
8. 单击**完成**。

将字段和方法添加到 OrderProcessCmdBonusImpl

您必须将两个字段以及 performExecute 方法添加到该类。

要将 theOrder 字段添加到 OrderProcessCmdBonusImpl 类，请执行以下操作:

1. 用鼠标右键单击 OrderProcessCmdBonusImpl 类，并选择**添加 > 字段**。“创建字段”智能向导打开。
2. 使用以下属性，将字段添加到该类。关于如何创建新字段的详细步骤，请参阅第 194 页的『创建新字段』

属性名称	值
字段名称	theOrder
字段类型	OrderAccessBean
初始值	留空。
访问修改量	private
其它修改量	全部保留不选中。
使用获取函数和设置函数方法进行访问	选中
获取函数	public
设置函数	public

并单击**完成**。

要将 bonusPercentAmount 字段添加到 OrderProcessCmdBonusImpl 类，请执行以下操作:

1. 再次用鼠标右键单击 OrderProcessCmdBonusImpl 类，并选择**添加 > 字段**。
2. 使用以下属性，将字段添加到该类。关于如何创建新字段的详细步骤，请参阅第 194 页的『创建新字段』

属性名称	值
字段名称	bonusPercentAmount
字段类型	double
初始值	1000
访问修改量	private
其它修改量	全部保留不选中。
使用获取函数和设置函数方法进行访问	选中
获取函数	public
设置函数	public

并单击**完成**。

要将 performExecute 方法添加到 OrderProcessCmdBonusImpl 类，请执行以下操作:

1. 用鼠标右键单击 **OrderProcessCmdBonusImpl** 类，并选择**添加 > 方法**。“创建方法”智能向导打开。
2. 确保已经选择**创建新方法**单选按钮，并单击**下一步**。
3. 在**方法名称**字段中，输入 performExecute。
4. 从**返回类型**下拉列表中选择 void。单击**下一步**。

5. 要指定该方法将抛出的异常，请单击**添加**。在**模式**字段中，输入 ECException，单击**添加**，然后单击**关闭**。
6. 单击**完成**。
   生成该方法，并显示该方法的源代码。
7. 您必须修改该源代码。在 performExecute 方法的源代码中定位以下行:

```
public void performExecute() throws
   com.ibm.commerce.exception.ECException {
```

在上述行之后，输入以下代码:

您可从 PDF 版本的《程序员指南》中剪切并粘贴此代码。建议首先将代码复制在 VisualAge for Java 剪贴板窗口中（关于更多信息，请参阅 VisualAge for Java 联机帮助），并检查该代码以确保剪切和粘贴操作期间没有字符丢失或被修改。然后，在验证完代码后将其复制到目标位置。注意将文本复制到另一编辑器可能导致要修改一些字符。

```
   final String methodName = "performExecute";
ECTrace.entry(ECTraceIdentifiers.COMPONENT_ORDER,
   this.getClass().toString(), methodName);

   // do all order processing as normal
      super.performExecute();

   // *** updating Bonus Point Information ***

// fetch order info
   theOrder = new OrderAccessBean();
   theOrder.setInitKey_orderId(getOrderRn().toString());

   int bonusPt;  // bonus points for this order
   int bonusTotal;  // total bonus points
   BigDecimal subtotal;  // subtotal
   BigDecimal bonusdeter;  // bonus determinant
   BigDecimal ans;

   // determine bonus points = subtotal * bonus determinant
     try {
       subtotal = theOrder.getTotalProductPriceInEJBType();
       bonusdeter = new BigDecimal(bonusPercentAmount);
       ans = subtotal.multiply(bonusdeter);
       bonusPt = Math.round(ans.floatValue());

        System.out.println("subtotal is: " + subtotal +
               " bonus deter is: " + bonusdeter + " ans is: " + ans);
        System.out.println("Bonus Percent amount = " +
           bonusPercentAmount);
        System.out.println("Bonus calculated is: "+ bonusPt);
       }
```

```
// Various Exceptions
     catch (Exception ex) {
          throw new ECSystemException(ECMessage._ERR_GENERIC,
               this.getClass().toString(), methodName,
               ECMessageHelper.generateMsgParms(ex.getMessage()), ex);
      }

   // *** Updating bonus points in BONUS table using bean
   //     created in previous example ***

   BonusAccessBean bonusBean = new BonusAccessBean();
bonusBean.setInit_argMemberId(getCommandContext().getUserId().toString());
     try {
     // new bonus value = this order bonus points + Old Bonus Points
     bonusTotal = bonusPt + Integer.parseInt(bonusBean.getBonusPoint());
     bonusBean.setBonusPoint(String.valueOf(bonusTotal));
     bonusBean.commitCopyHelper();

       System.out.println("In try, BonusTotal calculated is: "+
                bonusTotal);

       }
// Various exceptions
   catch (FinderException e) // user does not have points setup yet
   {
      // create a row in table bonus
     bonusTotal = bonusPt;
     try {
          BonusAccessBean bonusBeanNew = new
             BonusAccessBean(getCommandContext().getUserId(),
                new Integer(bonusTotal));

             System.out.println("In catch, BonusTotal calculated is: "+
                bonusTotal);

   }
     catch (Exception ex) {
          throw new ECSystemException(ECMessage._ERR_GENERIC,
               this.getClass().toString(), methodName,
               ECMessageHelper.generateMsgParms(ex.getMessage()), ex);
      }
}
     catch (Exception ex) {
          throw new ECSystemException(ECMessage._ERR_GENERIC,
               this.getClass().toString(), methodName,
               ECMessageHelper.generateMsgParms(ex.getMessage()), ex);
   }
   // *** setting view details ***

   // Fetch setResponse properties and add bonus parameters
  //  needed by the JSP page
TypedProperty resp = getResponseProperties();
resp.put("bonus", new Integer(bonusPt).toString());
```

```
        setResponseProperties(resp);
    ECTrace.exit(ECTraceIdentifiers.COMPONENT_ORDER,
        this.getClass().toString(), methodName);
```

8. 保存工作。

修改命令注册表以使用 OrderProcessCmdBonusImpl

在本示例中，您希望在需要订单处理的任何时候，使用新实现类来进行订单处理。要实现此目的，您必须更新命令注册表以建立原始 OrderProcess 接口与新 OrderProcessCmdBonusImpl 实现类之间的关联。

DB2 如果正在使用 DB2 数据库，请执行以下操作更新命令注册表:

1. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 控制中心**）。
2. 先选择“脚本”选项卡，然后通过在脚本窗口中输入以下信息，在 CMDREG 表中创建必需的条目:

```
connect to your_database_name;
update CMDREG
set CLASSNAME='com.ibm.commerce.sample.order.OrderProcessCmdBonusImpl'
WHERE INTERFACENAME='com.ibm.commerce.order.commands.OrderProcessCmd'
and storeent_Id=0;
```

其中 *your_database_name* 是数据库名，并单击“执行”图标
此命令供所有商家使用（由 STOREENT_ID 的值为 0 指示）。

Oracle 如果正在使用 Oracle 数据库，请执行以下操作更新命令注册表:

1. 打开 Oracle SQL Plus 命令窗口（**开始 > 程序 > Oracle > 应用程序开发 > SQL Plus**）。
2. 在**用户名**字段，输入 Oracle 用户名。
3. 在**密码**字段中，输入 Oracle 密码。
4. 在**主机字符串**字段，输入连接字符串。
5. 通过在 SQL Plus 窗口中输入以下信息，在 URLREG 表中创建必需的条目:

```
update CMDREG
set CLASSNAME='com.ibm.commerce.sample.order.OrderProcessCmdBonusImpl'
WHERE INTERFACENAME='com.ibm.commerce.order.commands.OrderProcessCmd'
and storeent_Id=0;
```

按 Enter 键运行 SQL 语句。
此命令供所有商家使用（由 STOREENT_ID 的值为 0 指示）。

6. 输入以下命令提交数据库的更改:

```
commit;
```

并按 Enter 键运行 SQL 语句。

修改 confirmation.jsp 模板

您必须修改 confirmation.jsp 模板以显示已经添加到订单处理业务过程的新业务逻辑。要修改显示模板，请执行以下操作:

1. 浏览至以下目录:
   vaj_drive:\VAJava\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web*store_directory*。
2. 制作 Confirmation.jsp 的副本，并将副本命名为 Confirmation.jsp.bak
3. 在文本编辑器中打开 confirmation.jsp。
4. 在现有的 import 语句之后，添加以下代码:

   ```
   <%@ page import="com.ibm.commerce.datatype.*" %>
   ```
5. 紧接着 JSP 模板中以下行:

   ```
   String orderRn = jhelper.getParameter("orderId");
   ```

   添加以下代码:

   ```
   String bonus = ((TypedProperty)request.getAttribute(
      ECConstants.EC_REQUESTPROPERTIES)).getString("bonus");
   ```
6. 在 JSP 模板中定位以下部分:

   ```
   <tr>
   <td align="left" valign="middle">
   <font class="product"><%=infashiontext.getString("GRAND_TOTAL")%>
      </font></td>
   <td align="right" valign="middle">
   <font class="strongprice"><%=orderBean.getGrandTotal()  %></font></td>
   ```

   然后添加以下代码:

   ```
   </tr>
   <tr>
   <td align="left" valign="middle">
   <font class="text">Bonus Points</font></td>
   <td align="right" valign="middle">
   <font class="strongtext"><%=bonus  %></font></td>
   ```
7. 保存工作。

在 WebSphere Test Environment 中测试 OrderProcessCmdBonusImpl

您现在可以使用 WebSphere Test Environment 来测试新的业务逻辑。要测试 OrderProcessCmdBonusImpl 命令，请执行以下操作:

1. 如第 309 页的附录 A, 『启动和停止 WebSphere Test Environment』中所述，启动 WebSphere Test Environment。
2. 打开浏览器并输入商店的 URL。例如，输入以下 URL:

   ```
   http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
        storeId=10001&catalogId=10001&langId=-1
   ```

3. 选择并购买产品。
4. 当已经购买产品后，订单确认将显示该订单获得的奖金点数数目。

（可选）部署定制业务逻辑至远程 WebSphere Commerce Server

当已经完成在 WebSphere Test Environment 中测试业务逻辑，并对代码感到满意后，您可以将该代码部署到远程 WebSphere Commerce Server 上的商店。在本教程中，该定制包含以下内容:

- 新的 OrderProcessCmdBonusImpl 类。
- 已更新的 confirmation.jsp 模板文件。
- 已更新的命令注册表。

这样，代码部署包含以下步骤:

1. 使用 VisualAge for Java 中的工具为命令逻辑创建 JAR 文件。
2. 将 JAR 文件和 JSP 模板复制到目标 WebSphere Commerce Server 上的适当目录中。
3. 更新目标 WebSphere Commerce Server 上的命令注册表。

关于测试支付方式的注释

在 WebSphere Test Environment 中运行的样本商店缺省情况下使用测试支付方式。使用此测试支付方式，这样您可以在 WebSphere Test Environment 中完成购物流程，而不需要发出对 Payment Manager 的调用。此测试支付方式只让您完成购物，它不会使得用此支付方式提交的订单可用于进一步处理。这样，测试支付方式应当只在 WebSphere Test Environment 中使用。

确保在将此定制代码部署到的商店中可以完成购买。支付处理可以使用本地的或远程的 Payment Manager 完成。

关于测试支付方式的更多信息，请参阅第 179 页的『测试支付方式』。

为命令逻辑创建 JAR 文件

您必须将命令逻辑封装到 JAR 文件中，以将其部署到目标 WebSphere Commerce Server。由于 `OrderProcessCmdBonusImpl` 存储在 _WCSamples 项目中，因此需要为该项目创建一个 JAR 文件。

要创建 JAR 文件，请执行以下操作:

1. 如第 309 页的附录 A, 『启动和停止 WebSphere Test Environment』中所述，停止 WebSphere Test Environment。
2. 用鼠标右键单击 **_WCSamples** 项目并选择**导出**。
   “导出”智能向导打开。

3. 选择 **Jar 文件**，并单击**下一步**。
4. 在 Jar 文件字段中，输入以下内容:
   drive:\WebSphere\CommerceServerDev\mytemp_b\wcssamplesb_1.jar
   其中 *drive* 是安装了 Commerce Studio 的驱动器。
5. 如下选择属性:

属性	值
类	选中
java	不选中
资源	选中
bean	选中
.class 文件中包含调试属性	选中

   对于其它属性接受缺省值。
6. 单击**完成**。

由于所创建的 JAR 文件不包含完整的数据包命名信息，因此您必须使用另一个封装实用程序（位于 VisualAge for Java 外部）来重新封装 JAR 文件。要重新封装此文件，请执行以下操作:

1. 在命令窗口中，导航到以下目录:
   drive:\WebSphere\CommerceServerDev\mytemp_b
2. 输入 mkdir temp1。
3. 输入 cd temp1。
4. 如下设置路径:
   set PATH=%PATH%; *drive*:\WebSphere\WebSphereStudio4\bin;
   其中 *drive* 是安装了 WebSphere Studio 的驱动器。
5. 输入 jar xvf ../wcssamplesb_1.jar
6. 输入 jar cvf ../wcssamplesb.jar *

将有用资源存储在目标 WebSphere Commerce Server 上

命令逻辑的 JAR 文件和修改过的 confirmation.jsp 模板必须放置在目标 WebSphere Commerce Server 上的适当目录中。

要将 JAR 文件存储在远程 WebSphere Commerce Server 上适当的目录中，请执行以下操作:

1. 在开发机器上，打开命令窗口并导航到以下目录:
   drive:\WebSphere\CommerceServerDev\mytemp_b
   并定位 wcssamplesb.jar 文件。

2. 将此文件复制到目标 WebSphere Commerce Server 上的以下目录中:

```
drive:\WebSphere\AppServer\installedApps\
   WC_Enterprise_App_instanceName.ear\wcstores.war\WEB-INF\lib
```

要将 confirmation.jsp 模板存储在目标 WebSphere Commerce Server 上适当的目录中，请执行以下操作:

1. 在开发机器上，浏览到以下目录:
   vaj_drive:\VAJava\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web*store_directory*
2. 将 confirmation.jsp 模板复制到目标 WebSphere Commerce Server 上的以下目录中:

```
drive:\WebSphere\AppServer\installedApps\
   WC_Enterprise_App_instanceName.ear\wcstores.war\storeDir
```

更新命令注册表

如果您正在将 OrderProcessCmdBonusImpl 命令部署到目标 WebSphere Commerce Server，并且其使用的数据库与 WebSphere Test Environment 不同，则必须更新目标数据库，以反映对命令注册表的更改。

DB2 如果正在使用 DB2 数据库，请执行以下操作更新目标 WebSphere Commerce Server 的数据库:

1. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 控制中心**）。
2. 先选择“脚本”选项卡，然后通过在脚本窗口中输入以下信息，在 CMDREG 表中创建必需的条目:

```
connect to your_target_database_name;
update CMDREG
set CLASSNAME='com.ibm.commerce.sample.order.OrderProcessCmdBonusImpl'
WHERE INTERFACENAME='com.ibm.commerce.order.commands.OrderProcessCmd'
and storeent_Id=0
```

   其中 *your_target_database_name* 是数据库的名称，并单击“执行”图标

Oracle 如果正在使用 Oracle 数据库，请执行以下操作更新命令注册表:

1. 打开 Oracle SQL Plus 命令窗口（**开始 > 程序 > Oracle > 应用程序开发 > SQL Plus**）。
2. 在**用户名**字段，输入 Oracle 用户名。
3. 在**密码**字段中，输入 Oracle 密码。
4. 在**主机字符串**字段，输入连接字符串。
5. 通过在 SQL Plus 窗口中输入以下信息，在 CMDREG 表中创建必需的条目:

```
update CMDREG
set CLASSNAME='com.ibm.commerce.sample.order.OrderProcessCmdBonusImpl'
WHERE INTERFACENAME='com.ibm.commerce.order.commands.OrderProcessCmd'
and storeent_Id=0;
```

单击 Enter 键运行 SQL 语句。
此命令供所有商家使用（通过把 STOREENT_ID 的值指定为 0 来表示）。

6. 输入以下命令提交数据库的更改:

```
commit;
```

并按 Enter 键运行 SQL 语句。

在 WebSphere Application Server 中重新启动企业应用程序

当通过把文件有用资源放置到适当的目录中并更新命令注册表，从而已经将命令逻辑添加到企业应用程序中后，您必须停止并重新启动企业应用程序，以使更改生效。

要停止并重新启动企业应用程序，请执行以下操作:

1. 打开 WebSphere Application Server 管理控制台。
2. 展开 **WebSphere 管理域**。
3. 展开**节点**。
4. 展开 ***nodeName***（其中 ***nodeName*** 是节点的名称）。
5. 展开**应用程序服务器**。
6. 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 **WebSphere Commerce Server - demo** 应用程序并选择**停止**。
7. 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 **WebSphere Commerce Server - demo** 应用程序并选择**启动**。

在运行在 WebSphere Application Server 中的“流行时尚”中测试新逻辑

您现在可以在运行在 WebSphere Application Server 中的“流行时尚”商店中验证新的业务逻辑。

要执行此最终验证，请执行以下操作:

1. 当已经启动 WebSphere Commerce Server 实例后，打开浏览器并输入商店主页的 URL: 例如，输入以下 URL:

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

其中 *store_Id* 是商店的标识，*catalog_Id* 是商店产品目录的标识。

2. 选择并购买产品。

3. 当已经购买产品后，订单确认将显示该订单获得的奖金点数数目。

修改现有实体 bean 和扩展现有任务命令

注: 本教程的目标是显示用于修改现有实体 bean 和扩展现有任务命令的过程。它并不是为了显示修改产品价格的最好方法。关于打折价格的信息，请参阅 *WebSphere Commerce Calculation Framework Guide*。

在第 9 章，『教程: 创建新的业务逻辑』中创建了一系列全新的业务逻辑。这包括创建新控制器命令、新 JSP 模板、新数据库表、用于访问表的新企业 bean、相应的访问 bean，以及数据访问 bean。所有这些逻辑组合一起创建简化的奖金点数应用程序，在该程序中可使用由新控制器启动的 JSP 模板更新用户的奖金点数余额。

第 9 章，『教程: 创建新的业务逻辑』中使用 BONUS 表的方法在下图中描绘:

BONUS 表在“创建新的业务逻辑”教程中的使用

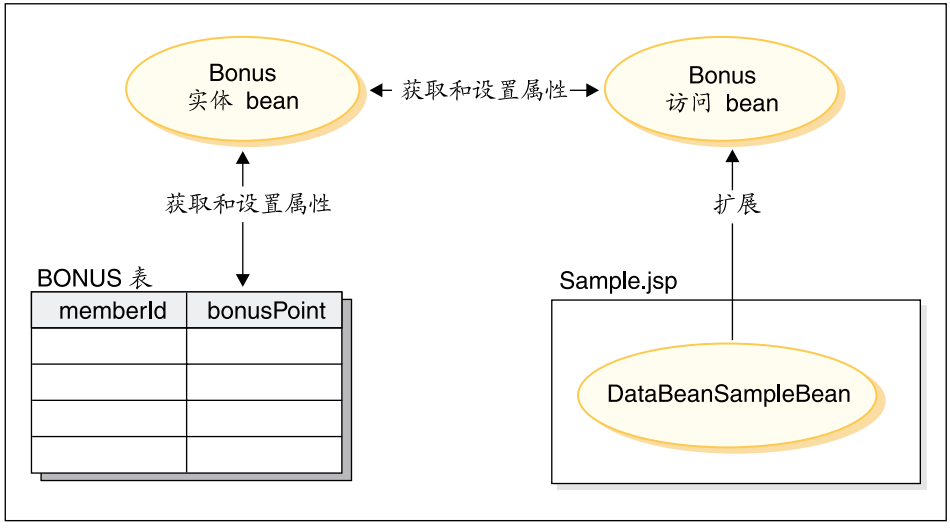


图 41.

在下一教程中，BONUS 表将以不同的方式使用。具体的说是，User 实体 bean 的定制方式使它对于应用程序而言，BONUS 表的 BONUSPOINT 列似乎实际上是 USERS 表中的一列。当在 USERS 表中创建新记录时，相应的记录将自动插入到 BONUS 表中。

为创建此*表联合*，必须将新的 CMP 字段添加到 User 实体 bean。此 CMP 字段使用 VisualAge for Java Map Browser 中的二级表映射功能，映射到 BONUS 表中的 BONUSPOINT 列。

要将修改过的 User 实体 bean 集成到购物流程中，需要创建产品的新价格。此新价格考虑到了购物者的当前奖金点数余额。您通过扩展 GetProductContractUnitPriceCmd 任务命令来创建新价格。当扩展此命令时，您将创建扩展 GetProductContractUnitPriceCmd 接口的新接口。新接口添加一个附加属性（用于奖金价格）。您还将创建扩展 GetContractUnitPriceCmdImpl 实现类的新实现类。此新实现类名为 “GetNewContractUnitPriceCmdImpl”。该新实现类调用它超类的 performExecute 方法，然后添加业务逻辑以确定新的奖金价格。

下图显示了 BONUS 表在下一教程中是如何使用的。

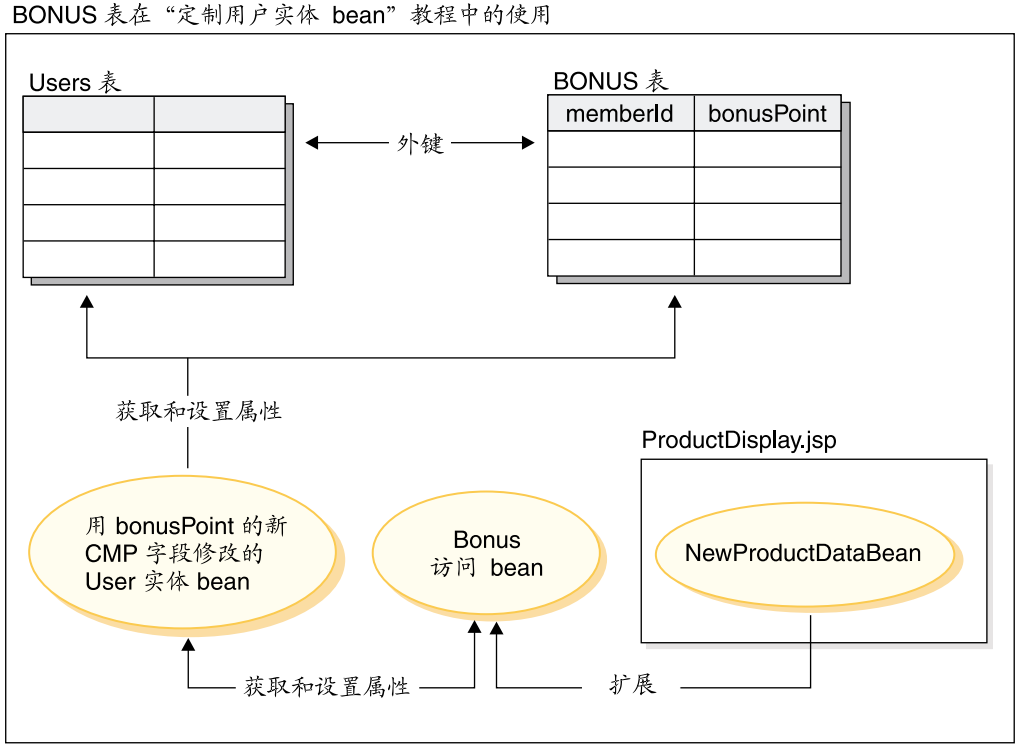


图 42.

此教程包含以下步骤:

1. 将新的 CMP 字段添加到 User 实体 bean。
2. 创建并填充 BONUS 表。
3. 更新 WCSUser 数据库模式和表映射，以包含新的 BONUS 表。
4. 创建 BONUS 表和 USER 表之间的外键关系。
5. 创建 BONUS 表映射。

6. 生成 User 实体 bean 的部署代码和访问 bean。
7. 使用测试客户机对更新过的实体 bean 进行初步测试。
8. 创建新的任务命令接口和实现类。新的任务命令扩展 GetBaseUnitProductPriceCmd。此命令中最重要的新功能是该实现类的 performExecute() 方法中的逻辑。此方法计算产品的新*奖金价格*。该奖金价格创建的方式是价格扣减购物者的奖金点数余额，最多可达所计算出价格的 20%。下表显示了所使用的价格扣减公式的示例:

计算出的价格	奖金点数余额	最大扣除（所计算出价格的 20%）	新奖金价格
$1000	100	$200	$900 由于奖金点数余额低于最大扣除，因此标价扣减奖金点数。
$1000	300	$200	$800 由于奖金点数余额超出最大扣除，因此标价扣减最大扣除 $200。

9. 创建扩展 ProductDataBean 的 NewProductDataBean。向此 bean 添加新方法，从而可在产品显示页便捷地使用新奖金价格。
10. 更新“流行时尚”商店的产品显示 JSP 模板以显示奖金价格。
11. 在运行在 WebSphere Test Environment 中的“流行时尚”商店中测试业务逻辑。
12. （可选）将已更新的业务逻辑部署到远程 WebSphere Commerce Server 并在 WebSphere Test Environment 外部对其进行测试。

开始本教程之前

如果尚未完成第 183 页的第 9 章，『教程: 创建新的业务逻辑』，则在开始本教程之前必须执行第 184 页的『准备样本项目』中所述的步骤。

将新的 bonusPoint 字段添加到 User 实体 bean

在本部分中，您将使用 VisualAge for Java 的 EJB 工具来将新的 CMP 字段添加到实体 bean。新字段名为 *bonusPoint*，并最终映射到 BONUS 表的 BONUSPOINT 列。

要将新的 CMP 字段添加到 User 实体 bean，请执行以下操作:

1. 如果小服务程序引擎、EJB 服务器和持久名称服务器正在运行，则如第 309 页的附录 A，『启动和停止 WebSphere Test Environment』中所述将它们停止运行。
2. 在工作台中，单击 **EJB** 选项卡。
3. 展开 **WCSUser** EJB 组。
4. 用鼠标右键单击 **User** bean，并选择**添加 > CMP 字段**。
   “创建 CMP 字段”智能向导打开。
5. 使用以下属性创建新的 CMP 字段:

属性	值
字段名称	bonusPoint
字段类型	int
初始值	0
使用获取函数和设置函数方法进行访问	启用
提升获取函数和设置函数方法到远程接口	启用
获取函数	public
设置函数	public

并单击**完成**。

“属性”窗格中显示 bonusPoint 字段。可能显示某些警告，但它们在重新生成该实体 bean 的代码时会得到修正。

创建并填充 BONUS 表

本教程中使用了一个记录用户奖金点数的数据库表。

如果完成了第 183 页的第 9 章，『教程: 创建新的业务逻辑』，则您对此表已经比较熟悉了。在此情况下，您必须更新该表，以为 USERS 表中的每个用户添加一行。如果没有完成第 183 页的第 9 章，『教程: 创建新的业务逻辑』，则必须创建并填充该表。每一个这种情景都提供了相应的指导。

DB2 如果正在使用 DB2 数据库而且需要*更新* BONUS 表，请执行以下操作:

1. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 控制中心**）并单击**脚本**选项卡。
2. 在**命令**字段中，输入

```
connect to your_database_name
```

（其中 *your_database_name* 是您的数据库的名称），并单击“执行”图标。

3. 在**命令**字段中，输入以下内容，然后单击“执行”图标:

```
INSERT INTO BONUS
(SELECT USERS_ID, 0
FROM USERS
WHERE USERS_ID NOT IN (SELECT MEMBERID FROM BONUS))
```

现在已经更新 BONUS 表。

DB2 如果正在使用 DB2 数据库而且需要*创建并填充* BONUS 表，请执行以下操作:

1. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 控制中心**）并单击**脚本**选项卡。
2. 在**命令**字段中，输入

```
connect to your_database_name
```

（其中 *your_database_name* 是您的数据库的名称），并单击“执行”图标。

3. 在**命令**字段中，输入以下内容，然后单击“执行”图标:

```
CREATE TABLE BONUS (MEMBERID BIGINT NOT NULL,
BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
constraint f_memberid foreign key (MEMBERID)
references users (users_id) on delete cascade)
```

现在已经创建 BONUS 表。

4. 要填充该表，请在**命令**字段中输入以下内容，然后单击“执行”图标:

```
insert into BONUS (select USERS_ID, 0 from USERS)
```

5. 关闭 DB2 控制中心。

Oracle 如果正在使用 Oracle 数据库，而且需要*更新* BONUS 表，请执行以下操作:

1. 打开 Oracle SQL Plus 命令窗口（**开始 > 程序 > Oracle > 应用程序开发 > SQL Plus**）。
2. 在**用户名**字段，输入 Oracle 用户名。
3. 在**密码**字段中，输入 Oracle 密码。
4. 在**主机字符串**字段，输入连接字符串。
5. 通过在 SQL Plus 窗口中输入以下信息，更新 BONUS 表:

```
INSERT INTO BONUS
(SELECT USERS_ID, 0
FROM USERS
WHERE USERS_ID NOT IN (SELECT MEMBERID FROM BONUS));
```

单击 Enter 键运行 SQL 语句。
现在已经更新 BONUS 表。

6. 输入以下命令提交数据库的更改:

```
commit;
```

并按 Enter 键运行 SQL 语句。

Oracle 如果正在使用 Oracle 数据库，而且需要*创建并填充* BONUS 表，请执行以下操作:

1. 打开 Oracle SQL Plus 命令窗口（**开始 > 程序 > Oracle > 应用程序开发 > SQL Plus**）。
2. 在**用户名**字段，输入 Oracle 用户名。
3. 在**密码**字段中，输入 Oracle 密码。
4. 在**主机字符串**字段，输入连接字符串。
5. 通过在 SQL Plus 窗口中输入以下信息，创建 BONUS 表:

```
CREATE TABLE Bonus (MEMBERID NUMBER NOT NULL,
BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
constraint f_memberid foreign key (MEMBERID)
references users (users_id) on delete cascade);
```

单击 Enter 键运行 SQL 语句。
现在已经创建 BONUS 表。

6. 通过输入以下内容，填充 BONUS 表:

```
insert into BONUS (select USERS_ID, 0 from USERS);
```

7. 输入以下命令提交数据库的更改:

```
commit;
```

并按 Enter 键运行 SQL 语句。

更新模式和表映射

在以下部分中，您将使用新的 BONUS 表更新 WCSUser 模式，创建新表的外键关系，并创建 User 实体 bean 字段与 BONUS 表列之间的表映射。

创建 BONUS 表模式

要创建表模式，请执行以下操作:

1. 用鼠标右键单击 **User** 实体 bean，并选择**打开到 > 数据库模式**。
   “模式浏览器”窗口打开。
2. 从**模式**列表中选择 **WCS 用户**。
3. 从**表**菜单中选择**新建表**。
   “表编辑器”打开。
4. 在**名称**字段中，输入 BONUS。
5. 将**限定符**和**物理名称**字段留空。
6. 在“表列”部分中，单击**新建**。 “列编辑器”打开。如下创建列:

属性	值
名称	memberId
物理名称	将此字段留空。
类型	BIGINT
类型详细信息	VapConverter
允许为空	不选中

   并单击**确定**。
7. 在“表列”列表中选择 **memberId**，并单击 **>>** 以使用此列作为主键。
8. 如下创建另一列（再次单击**新建**）:

属性	值
名称	bonusPoint
物理名称	将此字段留空。
类型	INTEGER
类型详细信息	VapConverter
允许为空	选中

   并单击**确定**。
9. 在“表编辑器”中单击**确定**。
   过一定时间后，**BONUS** 将列出在“模式浏览器”窗口的**表**列表中。
10. 为进行验证，请在**表**列表中单击 **BONUS**，并确保它的两列显示在**列**列表中。

创建外键关系

在本部分中，您将建立 BONUS 和 USERS 表之间的外键关系。

要创建外键关系，请执行以下操作:

1. 在“模式浏览器”窗口中，单击 **WCS 用户**模式。
   显示此模式的表和外键关系。

2. 从**外键**菜单中选择**新建外键关系**。
   “外键关系编辑器”打开。
3. 在**名称**字段中，输入 F_User_Bonus。
4. 确保已经选中**数据库中存在约束**复选框。
5. 从**主键表**下拉列表中选择 USERS
6. 从**外键表**下拉列表中选择 BONUS
7. 在**外键**标题下的空白字段中单击，显示一个下拉列表。从此列表中选择 **memberId**，并单击**确定**。
   F_User_Bonus 显示在外键关系的列表中。
8. 从**模式**菜单中，选择**保存模式**，然后单击**完成**。
9. 关闭“模式浏览器”。

创建 BONUS 表映射

在本部分中，您将创建 BONUS 表中 BONUSPOINT 列与 User 实体 bean 中 bonusPoint 字段之间的映射。

要创建 BONUS 表映射，请执行以下操作:

1. 确保已经打开“工作台”，并已选择 EJB 选项卡。
2. 从 **EJB** 菜单中选择**打开到 > 模式映射**。
   “映射浏览器”打开。
3. 从“数据仓库映射”列表中选择 **WCS 用户**。
4. 在**持久类**列表中，执行以下操作:
   a. 双击**成员**。
   b. 选择**用户**（请注意**用户**仅当在步骤 4a 中展开了“成员”后才显示在**成员**的下面。）
5. 从**表映射**菜单中选择**新建表映射 > 添加二级表映射**。
   “二级表映射”窗口打开。
6. 从**表**下拉列表中选择 **BONUS**。
7. 从“外键关系”下拉列表中选择 **F_User_Bonus**，并单击**确定**。
   过一定时间后，BONUS（二级）映射显示在“表映射”列表中。
8. 突出显示然后用鼠标右键单击 **BONUS（二级）**表映射，并选择**编辑属性映射**。
   “属性映射编辑器”打开。
9. 滚动到显示 bonusPoint 类属性的一行。选择 **bonusPoint**。
10. 在**映射类型**列的相应条目中单击，并从下拉列表中选择**简单**。
11. 在**表列**列的相应条目中单击，并从下拉列表中选择 **bonusPoint**。

12. 保留不更改所有其它字段，并单击**确定**。
    生成新属性映射，并过一定时间后，

    ```
    bonusPoint（bonusPoint）
    ```

    将显示在“映射浏览器”的**属性映射**列中。
13. 用鼠标右键单击 **WCS 用户**数据仓库映射，选择**保存数据仓库映射**，然后单击**完成**。
14. 关闭“映射浏览器”。

这时，User bean 包含指出某些抽象类未实现的错误。当生成部署代码时，这些错误将得到修正。

生成部署代码和访问 bean

由于已经修改 User 实体 bean 的代码，因此必须重新生成它的部署代码以及访问 bean。VisualAge for Java 中的工具使此代码生成步骤变得非常简单。

要执行此步骤，请执行以下操作:

1. 确保已经打开“工作台”，并已选择 EJB 选项卡。
2. 展开 **WCSUser** EJB 组。
3. 用鼠标右键单击 **User** bean，并选择**生成部署代码**。
   如果出现消息窗口询问是否以数据包版本创建，请单击**是**。
   代码生成需要花费几分钟时间。
4. 一旦已经生成部署代码，再次用鼠标右键单击 **User** bean，并选择**添加 > 访问 Bean**。“创建访问 Bean”智能向导打开。
5. 接受“选择访问 Bean 属性”页面上的缺省值，并单击**下一步**。
6. 接受“定义零参数构造函数”页面上的缺省值，并单击**下一步**。
7. 在“为复制帮助函数选择并定制 Bean 属性”页面上，向下滚动到“企业 Bean”列中的 **bonusPoint**。选中该 bean 的**复制帮助函数**，并将转换器设置为 `com.ibm.commerce.base.objects.WCSStringConverter`。
8. 单击**完成**。
9. 当显示“代码生成完成”消息时单击**确定**。

使用测试客户机测试修改

可使用测试客户机来测试新定制的 User 实体 bean。要启动测试客户机并测试该实体 bean，请执行以下操作:

1. 如第 309 页的『启动和停止持久名称服务器』中所述，启动持久名称服务器。

2. 如第 310 页的『启动和停止 EJB 服务器』中所述，启动 WCSUser EJB 组所在的 EJB 服务器。
3. 在“企业 Bean”窗格中，展开 **WCSUser** EJB 组。用鼠标右键单击 **User** 实体 bean，并选择**运行测试客户机**。
4. 在“EJB 查找”窗口中，单击**查找**。
5. 从方法列表中选择 **findByPrimaryKey(MemberKey)**。
6. 在“详细信息”窗格中，单击 **<null>**，然后在参数框中输入 -1000 ，并在“EJB 测试客户机”窗口中单击“调用”图标。

   请注意此处输入的值必须与 USERS 表 USERS_ID 列中的值相匹配。

   当此方法的执行完成时，关于标识为 -1000 的用户的信息显示在“方法”窗格的树形视图中。在此视图中有一个称为*方法*的节点。
7. 展开**方法**节点。
8. 单击 **getBonusPoint()**，然后单击“调用”图标。
   此方法检索顾客的奖金点数余额。返回当前奖金点数的余额。
9. 单击 **setBonusPoint(int)**，在“详细信息”窗格中输入 100 ，然后单击“调用”图标。
10. 单击 **getBonusPoint()**，然后单击“调用”图标。
    返回值 100。这显示 User 企业 bean 已经成功更新数据库。
11. 关闭 User 和“EJB 测试客户机”窗口。

创建 GetNewProductContractUnitPriceCmd 接口

作为此定制练习的一部分，您将创建新的业务逻辑，来根据顾客的奖金点数余额计算折扣价格。该计算由新的任务命令执行。在本部分中，您将创建此新任务命令的接口。

要创建新任务命令的接口，请执行以下操作:

1. 在工作台中，先选择“项目”选项卡，然后展开 **_WCSamples** 项目。
2. 用鼠标右键单击 **com.ibm.commerce.sample.commands** 数据包，并选择**添加 > 接口**。
3. 确保已经选择**创建新接口**，并在**接口名称**字段中，输入 GetNewProductContractUnitPriceCmd。
4. 通过单击**添加**选择应扩展的接口，然后在**模式**字段中，输入 com.ibm.commerce.price.commands.GetProductContractUnitPriceCmd，单击**添加**，然后单击**关闭**。
5. 单击**下一步**。
6. 要添加适当的 import 语句，请单击**添加数据包**，然后执行以下操作:

a. 在**模式**字段中，输入 com.ibm.commerce.price.utils，并单击**添加**。
b. 在**模式**字段中，输入 com.ibm.commerce.exception，并单击**添加**。
c. 单击**关闭**。

7. 确保已经选择**使接口公共化**。
8. 单击**完成**。
   新接口的源代码显示在“源码”窗格中。
9. 通过执行以下操作，为 getBonusPrice() 方法创建方法签名:
   a. 用鼠标右键单击 **GetNewProductContractUnitPriceCmd** 接口，并选择**添加 > 方法**。
      “创建方法”智能向导打开。
   b. 确保已经选择**创建新方法**，并单击**下一步**。
   c. 在“方法名称”字段中，输入 getBonusPrice。
   d. 要选择方法的返回类型，请单击**浏览**。在**模式**字段中，输入 MonetaryAmount，并单击**确定**，然后单击**下一步**。
   e. 要选择该方法可抛出的异常，请单击**添加**。在**模式**字段中，输入 ECSystemException，并单击**添加**，然后单击**关闭**。
   f. 单击**完成**。
10. 通过执行以下操作，向接口添加为命令指定缺省实现类的新字段:
    a. 用鼠标右键单击 **GetNewProductContractUnitPriceCmd** 接口，并选择**添加 > 字段**。
       “创建字段”智能向导打开。
    b. 在**字段名称**字段中，输入 defaultCommandClassName。
    c. 从**字段类型**下拉列表中选择 **String**。
    d. 在**初始值**字段中，输入

       ```
       "com.ibm.commerce.sample.commands.
          GetNewContractUnitPriceCmdImpl"
       ```

       并单击**完成**。

       注:

       1) 在输入此值时，您必须包含双引号。
       2) 如果警告消息指示将通过创建这个新字段来隐藏超类中的字段，请单击**是**继续。
11. 通过执行以下操作，向接口添加用于指定命令名的新字段:

a. 用鼠标右键单击 **GetNewProductContractUnitPriceCmd** 接口，并选择**添加 > 字段**。
“创建字段”智能向导打开。

b. 在**字段名称**字段中，输入 NAME。

c. 从**字段类型**下拉列表中选择 **String**。

d. 在**初始值**字段中，输入

```
"com.ibm.commerce.sample.commands.
   GetNewProductContractUnitPriceCmd"
```

并单击**完成**。

创建 GetNewContractUnitPriceCmdImpl 实现类

您必须为新任务命令创建实现类。此实现类实现新创建的接口，它并且包含任务命令的业务逻辑。

要创建 GetNewContractUnitPriceCmdImpl 实现类，请执行以下操作:

1. 在工作台中，先选择“项目”选项卡，然后展开 **_WCSamples** 项目。
2. 用鼠标右键单击 **com.ibm.commerce.sample.commands** 数据包，并选择**添加 > 类**。
“创建类”智能向导打开。
3. 确保已经选择**创建新类**，并如下创建该类:
   a. 在**类名**字段中，输入 GetNewContractUnitPriceCmdImpl。
   b. 要指定超类，请单击**浏览**，然后在**模式**字段中，输入 com.ibm.commerce.price.commands.GetContractUnitPriceCmdImpl，并单击**确定**。
   c. 单击**下一步**。
   d. 要指定应导入的数据包，请单击**添加数据包**。在**模式**字段中，输入以下数据包:
      - com.ibm.commerce.command 并单击**添加**
      - com.ibm.commerce.exception 并单击**添加**
      - com.ibm.commerce.price.commands 并单击**添加**
      - com.ibm.commerce.price.utils 并单击**添加**
      - com.ibm.commerce.ras 并单击**添加**
      - com.ibm.commerce.server 并单击**添加**
      - java.math 并单击**添加**，然后单击**关闭**。

e. 要指定该类应实现的接口，请单击**添加**。在“模式”字段中，输入以下接口:

- GetContractSpecialPriceCmd 并单击**添加**。
- GetContractUnitPriceCmd 并单击**添加**。
- GetProductContractUnitPriceCmd 并单击**添加**。
- GetNewProductContractUnitPriceCmd 并单击**添加**，然后单击**关闭**。

f. 单击**完成**。

4. 用鼠标右键单击 **GetNewContractUnitPriceCmdImpl** 类，并选择**添加 > 字段**。“创建字段”智能向导打开。如下输入信息:
   a. 在**字段名称**字段中，输入 bonusPrice。
   b. 要选择字段类型，请单击**浏览**。在“模式”字段中，输入 MonetaryAmount，并单击**确定**。
   c. 单击**完成**。
5. 将新的 performExecute() 方法添加到该类。此方法执行以下功能:
   - 调用超类（GetContractUnitPriceCmdImpl）的 performExecute() 方法
   - 设置 thisClass 和 methodName 值供异常处理机制使用
   - 实例化 StoreAccessBean
   - 实例化 UserAccessBean
   - 获取原始产品价格
   - 计算最大适用折扣
   - 计算新的奖金价格（此价格类型为 *double*）
   - 获取商店的货币类型
   - 舍入奖金价格，并将其作为货币金额以正确的货币进行存储

   要添加此方法，请用鼠标右键单击 **GetNewContractUnitPriceCmdImpl** 类，并选择**添加 > 方法**。“创建方法”智能向导打开。如下输入信息:
   a. 确保已经选择**创建新方法**，并单击**下一步**。
   b. 在**方法名称**字段中，输入 performExecute。
   c. 从**返回类型**下拉列表中选择 **void**，并单击**下一步**。
   d. 要选择该方法可抛出的异常，请单击**添加**。在**模式**字段中，输入 ECException，并单击**添加**，然后单击**关闭**。
   e. 单击**完成**。
   f. 在源代码中的此行之后:

```
public void performExecute()
   throws com.ibm.commerce.exception.ECException
{
```

添加以下代码:

您可从 PDF 版本的《程序员指南》中剪切并粘贴此代码。建议首先将代码复制在 VisualAge for Java 剪贴板窗口中（关于更多信息，请参阅 VisualAge for Java 联机帮助），并检查该代码以确保剪切和粘贴操作期间没有字符丢失。然后，在验证完代码后将其复制到目标位置。注意: 将文本复制到另一编辑器可能导致一些字符被修改。

```
super.performExecute();

// Get and set this class name and method
// for use when exceptions occur.
final String thisClass = GetContractUnitPriceCmdImpl.class.getName();
final String methodName = "performExecute";

//get the store access bean
Integer storeId = getStoreId();
com.ibm.commerce.common.objects.StoreAccessBean storeAB =
     getCommandContext().getStore(storeId);

//get the user access bean
com.ibm.commerce.user.objects.UserAccessBean bonusAB =
     new com.ibm.commerce.user.objects.UserAccessBean();

// get the calculated price from the GetContractUnitPriceCmdImpl
MonetaryAmount priceOrg = super.getPrice();
double dblPriceOrg = priceOrg.getValue().doubleValue();

//calculate the maximum bonus that can apply to this product
double dblBonusPrice; // = dblPriceOrg;
double dblMaxBonusPoint = 0;

try {
bonusAB.setInitKey_MemberId(super.getUserId().toString());
bonusAB.refreshCopyHelper();
double dblMaxDed = dblPriceOrg * 0.2;
dblMaxBonusPoint =
     (new java.math.BigDecimal(bonusAB.getBonusPoint()).doubleValue());
if (dblMaxBonusPoint > dblMaxDed) dblMaxBonusPoint = dblMaxDed;
} catch (javax.ejb.CreateException ex) {
throw new ECSystemException(
     ECMessage._ERR_CREATE_EXCEPTION, thisClass, methodName, ex);
} catch (javax.ejb.FinderException ex) {

} catch (javax.naming.NamingException ex) {
throw new ECSystemException(
     ECMessage._ERR_GENERIC, thisClass, methodName, ex);
} catch (java.rmi.RemoteException ex) {
throw new ECSystemException(
```

```
            ECMessage._ERR_REMOTE_EXCEPTION, thisClass, methodName, ex);
        }


        //apply the maximum applicable bonus to this product price
        dblBonusPrice = dblPriceOrg - dblMaxBonusPoint ;

        //get the currency of this store
        CommandContext context = getCommandContext();
        String requestedCurrency = Helper.getCurrency( context, storeAB );

        //round off and return the bonus price in MonetoryAmount type
        bonusPrice = new MonetaryAmount(
             new BigDecimal(dblBonusPrice), requestedCurrency);
        CurrencyManager.getInstance().roundCustomized(bonusPrice, storeAB);
```

保存工作。

6. 选择 **GetNewContractUnitPriceCmdImpl** 类中的 **getBonusPrice()** 方法查看其源代码。在源代码中，更改

   ```
   return null;
   ```

   为

   ```
   return bonusPrice;
   ```

   保存工作。

创建 NewProductDataBean 数据 bean

必须将 getCalculatedBonusPrice() 方法添加到现有的 WebSphere Commerce ProductDataBean。由于不必实际修改 ProductDataBean 的代码，因此必须创建扩展 ProductDataBean 的新数据 bean，并然后将此方法添加到新数据 bean。

要创建新的数据 bean，请执行以下操作:

1. 在工作台中，先选择“项目”选项卡，然后展开 **_WCSamples** 项目。
2. 用鼠标右键单击 **com.ibm.commerce.sample.databeans** 数据包，并选择**添加 > 类**。“创建类”智能向导打开。如下输入信息:
   a. 在**类名**字段中，输入 NewProductDataBean。
   b. 要指定超类，请单击**浏览**，然后在**模式**字段中，输入 com.ibm.commerce.catalog.beans.ProductDataBean。单击**确定**，然后单击**下一步**。
   c. 通过单击**添加数据包**将适当的 import 语句添加到该类，然后执行以下操作:
      - 输入 com.ibm.commerce.beans 并单击**添加**。
      - 输入 com.ibm.commerce.catalog.objects 并单击**添加**。

- 输入 com.ibm.commerce.command 并单击**添加**。
- 输入 com.ibm.commerce.datatype 并单击**添加**。
- 输入 com.ibm.commerce.exception 并单击**添加**。
- 输入 com.ibm.commerce.price.beans 并单击**添加**。
- 输入 com.ibm.commerce.ras 并单击**添加**。
- 输入 com.ibm.commerce.sample.commands 并单击**添加**。
- 输入 com.ibm.commerce.server 并单击**添加**。
- 输入 java.util 并单击**添加**，然后单击**关闭**。

d. 单击**完成**。

3. 用鼠标右键单击 **NewProductDataBean** 类，并选择**添加 > 方法**。
“添加方法”智能向导打开。
4. 确保已经选择**创建新方法**，并单击**下一步**。
5. 在**方法名称**字段中，输入 getCalculatedBonusPrice。
6. 要选择返回类型，请单击**浏览**。在**模式**字段中，输入 PriceDataBean，单击**确定**，然后单击**下一步**。
7. 要选择该方法可抛出的异常，请单击**添加**。在**模式**字段中，输入 ECSystemException，并单击**添加**，然后单击**关闭**。
8. 单击**完成**。
9. 在源代码中，替换 return null; 为以下内容:

```
PriceDataBean ibnPrice = null;
try {
     GetNewProductContractUnitPriceCmd comm =
        (GetNewProductContractUnitPriceCmd) CommandFactory.createCommand
        (GetNewProductContractUnitPriceCmd.NAME,
             getCommandContext().getStoreId());

     ECTrace.trace(ECTraceIdentifiers.COMPONENT_CATALOG,
     this.getClass().getName(), "getCalculatedBonusPrice",
     "Getting Price for CatalogEntry: " + getProductID());

     comm.setCatEntryId(new Long(getProductID()));
     comm.setCommandContext(getCommandContext());
     comm.execute();
     ibnPrice = new PriceDataBean(comm.getBonusPrice(),
          getCommandContext().getStore(),
          getCommandContext().getLanguageId());

} catch (Exception e) {
     throw new ECSystemException(ECMessage._ERR_RETRIEVE_PRICE,
          this.getClass().getName(), "getCalculatedBonusPrice",e);
}

return ibnPrice;
```

保存工作。

将新的奖金价格添加到产品显示模板

下一步是将新的奖金价格添加到产品显示模板，从而使购物者可以看到定制的价格。一旦已经更新显示模板，就会显示新的折扣价格。

样本商店使用 ProductDisplay.jsp 模板来显示产品。因此必须使用相关信息更新此模板以显示新价格。

要更新显示模板，请执行以下操作:

1. 导航到以下目录:
   vaj_drive:\VAJava\ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web*store_name*
2. 制作 ProductDisplay.jsp 文件的副本，并将它命名为 ProductDisplay.jsp.bak。
3. 在文本编辑器中打开 ProductDisplay.jsp。
4. 在 <%@ page import="com.ibm.commerce.common.beans.*" %> 之后，添加以下 import 语句:

   ```
   <%@ page import="com.ibm.commerce.sample.commands.*" %>
   <%@ page import="com.ibm.commerce.sample.databeans.*" %>
   ```
5. 用 NewProductDataBean 替换 ProductDataBean 所有出现的地方。
6. 定位以下行:

   ```
   <font class="price"><%=product.getCalculatedContractPrice()%></font>
      <br><br>
   ```

   在此行之后，插入以下行以检索并显示产品的奖金价格

   ```
   <font class="price"><%=product.getCalculatedBonusPrice()%> Bonus Price </font>
   <br><br>
   ```
7. 保存此文件。

注: 如果您从 PDF 版本的《*WebSphere Commerce 程序员指南*》中将一些部分剪切并粘贴到显示模板中，请确保在此过程中没有修改任何字符。

测试企业 bean 扩展

在本部分中，您可通过在“流行时尚”样本商店中查看产品，来测试对企业 bean 所作的扩展。显示新的奖金价格。

请注意为使此样本简单化，该奖金价格可供所有购物者查看（注册购物者或临时购物者）。对于没有奖金点数的购物者，奖金价格与一般价格相同。

要测试企业 bean 扩展并看到显示的奖金价格，请执行以下操作:

1. 通过执行以下操作，验证 _WCSamples 项目已包含在小服务程序引擎的路径中:
   a. 从 VisualAge for Java 的**工作空间**菜单中选择**工具 > WebSphere Test Environment**。
      “WebSphere Test Environment 控制中心”打开。
   b. 单击**小服务程序引擎**。
   c. 如果小服务程序引擎正在运行，请单击**停止小服务程序引擎**，然后单击**编辑类路径**。
   d. 如果未选择 **_WCSamples**，请现在选择它，并单击**确定**。
2. 如第 309 页的附录 A，『启动和停止 WebSphere Test Environment』中所述，启动 WebSphere Test Environment。持久名称服务器和 EJB 服务器可能已在运行，在这种情况下，您仅需要启动小服务程序引擎。
3. 打开浏览器并输入以下 URL

   ```
   http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
       storeId=store_Id&catalogId=catalog_Id&langId=-1
   ```

4. 单击“服务”标题下的**注册**链接，然后单击“新建顾客”标题下的**注册**。
   使用 wctester@wc 电子邮件地址和 wctester1 密码，注册新顾客。以测试值填写其它字段，并单击**提交**。保留浏览器打开。
5. DB2 打开 DB2 控制中心，并执行以下操作:
   a. 单击**交互式**选项卡
   b. 在**命令**字段中，执行以下操作:
      1) 输入

         ```
         connect to your_database_name
         ```

         （其中 *your_database_name* 是 WebSphere Commerce 数据库的名称）并单击“执行”图标。
      2) 输入 `select users_id from userreg where logonid = 'wctester@wc'` 并单击“执行”图标。
   c. “查询结果”选项卡显示在步骤 4 中所注册的顾客的条目。在此处记下顾客的 USERS_ID 值: _____
   d. 更新新注册的顾客的奖金点数余额。单击“交互式”选项卡，并在**命令**字段中，输入以下内容:

      ```
      update BONUS set BONUSPOINT = 1000 where MEMBERID = users_id
      ```

      其中 *users_id* 是步骤 5c 中的值。单击“执行”图标。
6. Oracle 通过执行以下操作，更新测试用户的奖金点数余额:

a. 打开 Oracle SQL Plus 命令窗口（**开始 > 程序 > Oracle > 应用程序开发 > SQL Plus**）。
b. 在**用户名**字段，输入 Oracle 用户名。
c. 在**密码**字段中，输入 Oracle 密码。
d. 在**主机字符串**字段，输入连接字符串。
e. 输入 `select users_id from userreg where logonid = 'wctester@wc';`
f. 显示步骤 4 中注册的顾客条目。在此处记下顾客的 USERS_ID 值:

_____

g. 通过输入以下内容，更新新注册顾客的奖金点数余额:

```
update BONUS set BONUSPOINT = 1000 where MEMBERID = users_id;
```

其中 *users_id* 是步骤 6f 中的值。

h. 输入以下命令提交数据库的更改:

```
commit;
```

并按 Enter 键运行 SQL 语句。

7. 在浏览器中，单击**男士**链接查看商店的男士时装部分。
8. 单击此特色特价商品的链接查看产品页面。该页面显示一般价格以及基于顾客奖金点数余额的折扣价格。

注: 如果显示堆栈跟踪而不是奖金价格，则您可能需要禁用高速缓存。如下所示，可通过在 *instance_name*.xml 文件中将 CacheDaemon 组件值设置为 false 来完成此操作:

```
<component compClassName=
"com.ibm.commerce.cache.daemon.CacheDaemonComponent"
                enable="false"
                name="CacheDaemon" />
```

更改了 *instance_name*.xml 文件中的值后，您必须停止并重新启动 WebSphere Test Environment 中的小服务程序引擎。

（可选）部署定制业务逻辑至远程 WebSphere Commerce Server

本部分中描述了如何向在 WebSphere Test Environment 外部运行的商店部署修改过的实体 bean 和新的任务命令。

部署需要为 WebSphere Commerce 公共企业 bean 以及命令和数据 bean 逻辑创建 JAR 文件，将 JAR 文件放置在目标服务器上的适当目录中，停止 WebSphere Commerce 实例，修改类路径，使用 XMLConfig 实用程序部署企业 bean，并重新启动实例。

为新的价格命令创建 JAR 文件

您必须为 **_WCSamples** 项目创建一个 JAR 文件，这样新的任务命令就可以得到部署。要创建此 JAR 文件，请在开发机器上执行以下操作:

1. 如第 309 页的附录 A，『启动和停止 WebSphere Test Environment』中所述，停止 WebSphere Test Environment。
2. 先选择“项目”选项卡，然后选择 **_WCSamples** 项目。
3. 突出显示该项目，然后用鼠标右键单击并选择**导出**。
   “导出”智能向导打开。
4. 选择 **Jar 文件**，并单击**下一步**。
5. 在 **Jar 文件**字段，输入以下内容:
   drive:\WebSphere\CommerceServerDev\mytemp_c\wcssamplesc_1.jar
   其中 *drive* 是安装了 WebSphere Commerce 的驱动器。
6. 如下选择属性:

属性	值
类	选中
java	不选中
资源	选中
bean	选中
.class 文件中包含调试属性	选中

   对于其它属性接受缺省值。
7. 单击**完成**。

由于所创建的 JAR 文件不包含完整的数据包命名信息，因此您必须使用另一个封装实用程序（位于 VisualAge for Java 外部）来重新封装 JAR 文件。要重新封装此文件，请执行以下操作:

1. 在命令窗口，导航到以下目录:
   drive:\WebSphere\CommerceServerDev\mytemp_c
2. 输入 mkdir temp3。
3. 输入 cd temp3。
4. 如下设置路径:
   set PATH=%PATH%; *drive*:\WebSphere\WebSphereStudio4\bin;
   其中 *drive* 是安装了 WebSphere Studio 的驱动器。
5. 输入 jar xvf ../wcssamplesc_1.jar。
6. 输入 jar cvf ../wcssamplesc.jar * （注意: 已从名称中除去 _1）。

为 WCSUser EJB 组创建 JAR 文件

您必须为包含已修改企业 bean 的 EJB 组创建 “EJB 1.1 导出 JAR” 文件。因此，在创建 JAR 文件时，您必须选择以下组:

- WCSUser

要为 WCSUser EJB 组创建 JAR 文件，请执行以下操作:

1. 先选择 EJB 选项卡，然后突出显示 **WCSUser** EJB 组。
2. 用鼠标右键单击 **WCSUser** EJB 组并选择**导出 > EJB 1.1 JAR**。
   “导出到 EJB 1.1 JAR 文件” 智能向导打开。
3. 在 **JAR 文件** 字段中，输入 *drive*:\WebSphere\CommerceServerDev\mytemp_c\CustomizedWCSUserDeployed_DT.jar
4. 如下选择属性:

属性	值
类	选中
java	不选中
资源	选中
目标数据库	DB2 如果您正在部署到 DB2 数据库，请选择 **DB2 for NT, V7.1**。 Oracle 如果您正在部署到 Oracle 数据库，请选择 **Oracle**，**V8**。
.class 文件中包含调试属性	选中

   对于其它属性接受缺省值。
5. 单击**完成**。

创建 JAR 文件。

JAR 文件命名时已经使用 “_DT” 后缀作为提醒符号，以提醒在将此 JAR 文件部署到 WebSphere Commerce 应用程序中之前，必须使用 WebSphere Application Server 提供的 “EJB 部署工具” 运行它。

创建 wcsejsclient.jar 文件

要创建客户机 JAR 文件，请执行以下操作:

1. 先选择 EJB 选项卡，然后突出显示所有的 WebSphere Commerce EJB 组（它们的名称以 WCS 开头）。突出显示所有的这些组后，用鼠标右键单击并选择**导出 > 客户机 JAR**。
   “导出” 智能向导打开。

2. 在 **JAR 文件**字段中，输入
   drive:\WebSphere\CommerceServerDev\mytemp_c\wcsejsclient.jar
3. 如下选择属性:

属性	值
bean	选中
类	选中
java	不选中
资源	选中
.class 文件中包含调试属性	选中

   对于其它属性接受缺省值。
4. 单击**完成**。

创建 JAR 文件。

复制已更新的 JSP 模板至目标商店目录

在第 290 页的『将新的奖金价格添加到产品显示模板』中，您更新了显示模板以反映新创建的价格。在本步骤中，您将把更新过的 JSP 模板从 WebSphere Test Environment 目录结构复制到商店当运行在 WebSphere Test Environment 外部时所使用的目录中。

1. 在开发机器上，导航到以下目录:
   vaj_drive:\VAJava\Ide\project_resources\IBM WebSphere Test Environment\hosts\default_host\default_app\web*store_directory*
   其中 *vaj_drive* 是安装了 VisualAge for Java 的驱动器，*store_directory* 是样本商店的目录名称。
   复制 ProductDisplay.jsp 文件。
2. 将 ProductDisplay.jsp 文件粘贴到以下目录中:

   ```
   drive:\WebSphere\AppServer\installedApps\
      WC_Enterprise_App_instance_name.ear\
      wcstores.war\store_directory
   ```

   其中 *drive* 是安装了 WebSphere Commerce 的驱动器，*store_directory* 是商店的目录名称，*instance_name* 是 WebSphere Commerce 实例的名称。

复制 JAR 文件至目标 WebSphere Commerce Server

您必须将 JAR 文件从开发机器复制到目标 WebSphere Commerce Server 上的适当目录中。要复制这些文件，请执行以下操作:

1. 在开发机器上，导航到以下目录:
   drive:\WebSphere\CommerceServerDev\ mytemp_c
   并定位以下文件:
   - wcssamplesc.jar
   - CustomizedWCSUserDeployed_DT.jar
   - wcsejsclient.jar

   其中 *drive* 是安装了 WebSphere Commerce Studio，商务开发者版的驱动器。

   上述每个文件都必须复制到目标 WebSphere Commerce Server 上的特定目录中。仔细阅读以下步骤，以确保每个文件都存储在正确的位置中。
2. 将 wcssamplesc.jar 文件复制到目标 WebSphere Commerce Server 上的以下目录中:

   ```
   drive:\WebSphere\AppServer\installedApps\
      WC_Enterprise_App_instance_name.ear\
      wcstores.war\WEB-INF\lib
   ```

   其中 *drive* 是安装了 WebSphere Commerce 商务版的驱动器，*instance_name* 是实例的名称（例如 demo）。
3. 将 wcsejsclient.jar 文件复制到目标 WebSphere Commerce Server 上的以下目录中:

   ```
   drive:\WebSphere\CommerceServer\temp\lib
   ```
4. 将 CustomizedWCSUserDeployed_DT.jar 文件复制到目标 WebSphere Commerce Server 上的以下目录中:

   ```
   drive:\WebSphere\CommerceServer\temp
   ```

运行 EJB 部署工具

您必须为包含新 EJB 组的 JAR 文件运行 EJB 部署工具。此工具随 WebSphere Application Server 附带。

要运行此工具，请执行以下操作:

1. 在命令提示符下，导航到以下目录:

   ```
   drive:\WebSphere\CommerceServer\temp
   ```
2. 通过输入以下命令，临时性将该工具添加到系统路径中:

   ```
   PATH=drive:\WebSphere\AppServer\deploytool;%PATH%
   ```
3. 如下输入 ejbdeploy 命令:

   ```
   ejbdeploy EJBGroupJARFile WorkingDir OutputJARFile -nowarn -keep -35 -cp
   ClassPathOfDepJARFiles
   ```

   其中:

- *EJBGroupJARFile* 是用于 EJB 组的 JAR 文件的名称。在此情况下，它是 CustomizedWCSUserDeployed_DT.jar。
- *WorkingDir* 是工作目录。
- *OutputJARFile* 是输出 JAR 文件的名称。在此情况下，输入 CustomizedWCSUserDeployed.jar。
- -nowarn 是用于禁止警告消息和信息性消息的可选参数。
- -keep 是用于在已经运行 ejbdeploy 命令后保留工作目录的可选参数。
- -35 是一个强制性参数，它将对 CMP 实体 bean 使用与 “EJB 部署工具”（EJB 部署工具随 WebSphere Application Server 版本 3.5 一起提供）中所用相同的自顶向下的映射规则。
- -cp *ClassPathOfDepJARFiles* 是任何从属 JAR 文件的类路径。当修改了现有的 WebSphere Commerce 企业 bean 后，必须在从属 JAR 文件的类路径中包含 wcsejsclient.jar、wcsejbimpl.jar 和 xml4j.jar 文件。这样，请输入以下内容:

```
"drive:\WebSphere\CommerceServer\temp\lib\wcsejsclient.jar;
drive:\WebSphere\AppServer\InstalledApps\
WC_Enterprise_App_instanceName.ear\lib\wcsejbimpl.jar;
drive:\WebSphere\AppServer\InstalledApps\
WC_Enterprise_App_instanceName.ear\lib\xml4j.jar;"
```

修改实体 bean 的事务隔离级别

在此步骤中，您使用 modifyIsolationLevel 命令将实体 bean 的交易隔离级别修改为特定类型数据库所需的级别。

要运行 modifyIsolationLevel 命令，请执行以下操作:

1. 在目标 WebSphere Commerce Server 上，使用命令提示符导航到以下目录:

   drive:\WebSphere\CommerceServer\bin

2. 您必须发出 modifyIsolationLevel 命令，此命令的一般语法如下:

   ```
   modifyIsolationLevel -jarFile jar_file_name.jar
      -logFile log_file_name -dbType db_type
   ```

   where

   - *jar_file_name.jar* 是包含定制代码的 JAR 文件名称
   - *log_file_name* 是应当作为记录信息的位置的全限定文件名
   - *db_type* 是正在使用的数据库类型。输入 DB2 或 ORACLE

   以下是指定了所有值的 modifyIsolationLevel 命令示例:

```
modifyIsolationLevel -jarFile
   D:\WebSphere\CommerceServer\temp\CustomizedWCSUserDeployed.jar
   -logFile D:\WebSphere\CommerceServer\instances\demo\logs\output.log
   -dbType DB2
```

注: 参数名称区分大小写。也就是说，**jarFile** 与 **jarfile** 不一样。确保正确输入参数名称。

如果没有异常显示在命令窗口中，则命令已经成功运行。完成后，请注意在部署 JAR 文件上的时间戳记已经更改。

更新目标数据库

如果正在部署到目标 WebSphere Commerce Server，并且其使用的数据库与 WebSphere Test Environment 所使用的不同，则您必须如下更新目标数据库:

- 如果已完成第 183 页的第 9 章，『教程: 创建新的业务逻辑』，则必须更新该表，以便为 USERS 表中的每个用户添加行。
- 如果没有完成第 183 页的第 9 章，『教程: 创建新的业务逻辑』，则必须创建并填充该表。

如果没有完成第 183 页的第 9 章，『教程: 创建新的业务逻辑』，则必须创建并填充该表。每一个这种情景都提供了相应的指导。

DB2 如果正在使用 DB2 数据库而且需要*更新* BONUS 表，请执行以下操作:

1. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 控制中心**）并单击**脚本**选项卡。
2. 在**脚本**字段中，输入

   ```
   connect to your_database_name
   ```

   （其中 *your_database_name* 是您的数据库的名称），并单击“执行”图标。
3. 在**脚本**字段中，输入以下信息，然后单击“执行”图标:

   ```
   INSERT INTO BONUS
   (SELECT USERS_ID, 0
   FROM USERS
   WHERE USERS_ID NOT IN (SELECT MEMBERID FROM BONUS))
   ```

   现在已经更新 BONUS 表。

DB2 如果正在使用 DB2 数据库而且需要*创建并填充* BONUS 表，请执行以下操作:

1. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 控制中心**）并单击**脚本**选项卡。
2. 在**脚本**字段中，输入

```
connect to your_database_name
```

（其中 *your_database_name* 是您的数据库的名称），并单击“执行”图标。

3. 在**脚本**字段中，输入以下信息，然后单击“执行”图标:

```
CREATE TABLE BONUS (MEMBERID BIGINT NOT NULL,
BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
constraint f_memberid foreign key (MEMBERID)
references users (users_id) on delete cascade)
```

现在已经创建 BONUS 表。

4. 要填充此表，请在**脚本**字段中输入以下信息，然后单击“执行”图标:

```
insert into BONUS (select USERS_ID, 0 from USERS)
```

5. 关闭 DB2 控制中心。

Oracle 如果正在使用 Oracle 数据库，而且需要*更新* BONUS 表，请执行以下操作:

1. 打开 Oracle SQL Plus 命令窗口（**开始 > 程序 > Oracle > 应用程序开发 > SQL Plus**）。
2. 在**用户名**字段，输入 Oracle 用户名。
3. 在**密码**字段中，输入 Oracle 密码。
4. 在**主机字符串**字段，输入连接字符串。
5. 通过在 SQL Plus 窗口中输入以下信息，更新 BONUS 表:

```
INSERT INTO BONUS
(SELECT USERS_ID, 0
FROM USERS
WHERE USERS_ID NOT IN (SELECT MEMBERID FROM BONUS));
```

单击 Enter 键运行 SQL 语句。
现在已经更新 BONUS 表。

6. 输入以下命令提交数据库的更改:

```
commit;
```

并按 Enter 键运行 SQL 语句。

Oracle 如果正在使用 Oracle 数据库，而且需要*创建并填充* BONUS 表，请执行以下操作:

1. 打开 Oracle SQL Plus 命令窗口（**开始 > 程序 > Oracle > 应用程序开发 > SQL Plus**）。
2. 在**用户名**字段，输入 Oracle 用户名。

3. 在**密码**字段中，输入 Oracle 密码。
4. 在**主机字符串**字段，输入连接字符串。
5. 通过在 SQL Plus 窗口中输入以下信息，创建 BONUS 表:

   ```
   CREATE TABLE Bonus (MEMBERID NUMBER NOT NULL,
   BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
   constraint f_memberid foreign key (MEMBERID)
   references users (users_id) on delete cascade);
   ```

   单击 Enter 键运行 SQL 语句。
   现在已经创建 BONUS 表。
6. 通过输入以下内容，填充 BONUS 表:

   ```
   insert into BONUS (select USERS_ID, 0 from USERS);
   ```
7. 输入以下命令提交数据库的更改:

   ```
   commit;
   ```

   并按 Enter 键运行 SQL 语句。

从 WebSphere Application Server 中导出当前的企业应用程序

在此步骤中，您从 WebSphere Application Server 中导出当前的企业应用程序，这样您就可以稍后在应用程序组装工具中打开它了。

要从 WebSphere Application Server 中导出当前企业应用程序，请执行以下操作:

1. 打开 WebSphere Application Server 管理控制台。
2. 展开 **WebSphere 管理域**。
3. 展开**企业应用程序**。
4. 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，展开 **demo** 应用程序，并选择**导出应用程序**。
5. 在**导出目录**字段中，输入 *drive*:\WebSphere\CommerceServer\working。
   这将整个应用程序（包括所有资源）导出到 WC_Enterprise_App_*instanceName*.ear 文件中（其中 *instanceName* 是 WebSphere Commerce 实例的名称）。

   注: 如果您已具有现有的 WC_Enterprise_App_*instanceName*.ear 文件，则您可以重命名旧的文件或覆盖它。

导出企业应用程序的 XML 配置信息

您必须导出企业应用程序的 XML 配置信息。要导出此信息，您将使用 WebSphere Application Server 提供的 XMLConfig 命令行实用程序。

要导出此配置信息，请执行以下操作:

1. 在命令提示符下，导航到以下目录:

   drive:\WebSphere\CommerceServer\working

2. 通过输入以下命令，调用 XMLConfig 工具来执行部分的导出:

   ```
   xmlConfig -export OutputFileB.xml -partial was.export.app.xml
      -adminNodeName wasHostName
   ```

   其中 *wasHostName* 是 WebSphere Application Server 中包含当前企业应用程序的节点的名称。另外，OutputFileB.xml 是作为运行此命令的结果创建的文件的名称。

在您导出有关当前企业应用程序中的企业 bean 的信息之后，您必须更新 OutputFileB.xml 文件，以便它指向包含已修改的 User bean 代码的 JAR 文件。

要更新 OutputFileB.xml 文件，请执行以下操作:

1. 导航到以下目录:

   drive:\WebSphere\CommerceServer\working

2. 在文本编辑器中打开 OutputFileB.xml 文件。
3. 定位 <ear-file-name> 标记，并将该值替换为以下值:

   ```
   drive:\WebSphere\CommerceServer\working\
      WC_Enterprise_App_instanceName.ear
   ```

4. 定位 WCSUser EJB 组的 <ejb-module> 节并将 <jar-file> 标记中包含的值更改为 CustomizedWCSUserDeployed.jar
5. 保存 OutputFileB.xml 文件。

将已修改的 EJB 组组装到企业应用程序中

在本步骤中，您在应用程序组装器工具中打开企业应用程序。一旦企业应用程序在该工具中打开，您就可以执行以下操作，以将已修改的 User bean 包含到企业应用程序中:

1. 为现有版本的 WCSUser EJB 组的类路径制作副本。
2. 除去现有版本的 WCSUser EJB 组。
3. 导入新版本的 WCSUser EJB 组。新 EJB 组的 JAR 文件存储在企业应用程序的 “EJB 模块” 部分中。
4. 设置 WCSUser EJB 组的类路径。

要将新的 EJB 组组装到企业应用程序中，请执行以下操作:

1. 通过执行以下操作，备份当前的企业应用程序:
   a. 在命令提示符下，导航到以下目录:

      drive:\WebSphere\CommerceServer\working

b. 输入以下命令:

```
copy WC_Enterprise_App_instanceName.ear
   WC_Enterprise_App_instanceName.ear.bak2
```

2. 打开 WebSphere Application Server 管理控制台。
3. 从**工具**菜单中，选择**应用程序组装工具**。（如果“欢迎”窗口打开，请选择**取消**以访问控制台。）
4. 通过执行以下操作，打开您将在其上工作的企业应用程序:
   a. 从**文件**菜单中，选择**打开**。
   b. 在**文件名**字段中，输入:

   ```
   drive:\WebSphere\CommerceServer\working\
      WC_Enterprise_App_instanceName.ear
   ```

   并单击**打开**。等待应用程序打开后，继续到后续步骤。这需要花费几分钟。
5. 单击 **EJB 模块**。右边的窗格显示企业应用程序中的 EJB 模块。
6. 单击 **WCSUser** EJB 模块。
7. 单击“常规”选项卡以查看现有 WCSUser EJB 模块的类路径信息。将此现有类路径信息复制到文本文件（例如 WCSUser_path.txt）中。
8. 用鼠标右键单击 **WCSUser** EJB 模块，并选择**删除**。
9. 用鼠标右键单击 **EJB 模块**，并选择**导入**。
10. 在**文件名**字段中，输入

    ```
    drive:\WebSphere\CommerceServer\temp\CustomizedWCSUserDeployed.jar
    ```

    并单击**打开**。在“确认值”窗口中，单击**确定**。
11. 一旦导入了 CustomizedWCSUserDeployed.jar 文件，请滚动到 **WCSUser** EJB 组，并选择此组。
    关于此组的信息显示在右边的窗格中。
12. 打开包含先前版本的 WCSUser EJB 组的类路径信息的文本文件。选择并复制该类路径。
13. 在新 WCSUser EJB 组的类路径字段中，粘贴此类路径信息。
14. 单击**应用**。
15. 从**文件**菜单中，选择**关闭**。
16. 等待文件关闭，然后从**文件**菜单中，选择**打开**，重新打开 *drive*:\WebSphere\CommerceServer\working\WC_Enterprise_App_*instanceName*.ear 文件。
17. 通过执行以下操作，配置 User bean 的安全性:

a. 先展开“EJB 模块”节点，然后定位并展开 **WCSUser** 节点。

b. 展开**实体 Bean**。

c. 展开**用户**

d. 单击**方法扩展**，然后在右边的窗格中执行以下操作:

   1) 单击**高级**选项卡。

   2) 确保已经选择**安全性身份**。

   3) 对于每种方法，确保已经选择**使用 EJB 服务器的身份**。

   4) 单击**应用**（如果您做出了任何修改）。

e. 在左侧导航窗格中，用鼠标右键单击 WCSUser EJB 组下的**安全性角色**，选择**新建**，然后执行以下操作:

   1) 在**名称**字段中，输入 WCSecurityRole，并单击**应用**。注意，如果已经存在此角色，您就无需执行此步骤了。

f. 在左侧导航窗格中，用鼠标右键单击 WCSUser EJB 组下的**方法许可**，选择**新建**，然后执行以下操作:

   1) 在**方法许可名称**字段中，输入 WCMethodPermission

   2) 在**方法**选择区域中，单击**添加**。
   “添加方法”窗口打开。

   3) 展开 **CustomizedWCSUserDeployed.jar** 并选择所有的企业 bean（在选择时按住 Shift 键）。单击**确定**。然后，所有的企业 bean 都显示在“企业 bean”列下，而**所有的方法**都显示在“类型”列下。

   4) 在“角色”选择区域中，单击**添加**，选择 WCSecurityRole 并单击**确定**。

   5) 单击**应用**，然后单击 **确定**。

18. 从**文件**菜单中，选择**保存**。

19. 关闭应用程序组装器工具。

完成本步骤后，您就已经创建包含所有先前逻辑以及新业务逻辑的新企业应用程序。这些逻辑正是新修改的 WC_Enterprise_App_*instanceName*.ear 文件中包含的所有内容。

导入新的企业应用程序到 WebSphere Application Server 中

以下是将新企业应用程序导入到 WebSphere Application Server 中所涉及到的高级步骤:

1. 停止当前在 WebSphere Application Server 中运行的企业应用程序，然后除去它。在 WebSphere Application Server 管理控制台中执行这些步骤。

2. 使用 XMLConfig 命令行实用程序导入新的应用程序。

3. 刷新 WebSphere Application Server 管理员控制台，然后启动新的企业应用程序。

每个这些步骤在以下部分有更详细描述。

停止并除去当前的企业应用程序: 要从 WebSphere Application Server 中停止并除去当前的企业应用程序，请执行以下操作:

1. 打开 WebSphere Application Server 管理控制台。
2. 展开 **WebSphere 管理域**。
3. 展开**节点**。
4. 展开 *nodeName*（其中 *nodeName* 是节点的名称）。
5. 展开**应用程序服务器**。
6. 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 **WebSphere Commerce Server -** *instanceName* 并选择**停止**。
7. 展开**企业应用程序**。
8. 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 **WebSphere Commerce 企业服务器 - demo** 应用程序并选择**停止**。
9. 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 **WebSphere Commerce 企业服务器 - demo** 应用程序并选择**除去**。
10. 当提示指出应用程序是否应导出时，选择**否**。

使用 XMLConfig 导入新的企业应用程序: 要使用 XMLConfig 命令行实用程序导入新的企业应用程序，请执行以下操作:

1. 导航到以下目录:

   ```
   drive:\WebSphere\CommerceServer\working
   ```

2. 在命令提示符下，输入以下命令以将企业应用程序导入到 WebSphere Application Server 中:

   ```
   xmlConfig -import OutputFileB.xml -adminNodeName was_hostname
   ```

   其中 *was_hostname* 是 WebSphere Application Server 中包含当前应用程序的节点的名称。

注: 400 如果正在部署到在 iSeries 上运行的 WebSphere Commerce 实例，那么在导入应用程序之后，您将必须执行附加的步骤以修改目录许可权。有关如何修改这些许可权的详细信息，请参阅第 342 页的『导入企业应用程序』。

启动新的企业应用程序: 当使用 XMLConfig 命令行实用程序导入了新的企业应用程序后，您可以使用 WebSphere Application Server 管理员控制台来执行刷新并然后启动新的应用程序。

要刷新控制台并启动新的应用程序，请执行以下操作:

1. 打开 WebSphere Application Server 管理控制台。
2. 展开 **WebSphere 管理域**
3. 突出显示**节点**。
4. 单击**刷新选定的子树**图标。
5. 通过执行以下操作，启动您的 WebSphere Commerce 应用程序:
   - 展开**应用程序服务器**。
   - 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 **WebSphere Commerce Server -** *instanceName*，并选择**启动**。

测试目标商店中的新代码

要在运行在 WebSphere Application Server 中的商店中测试修改过的实体 bean 和新的任务命令，请执行以下操作:

1. 打开浏览器并输入以下 URL:

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

   其中 *store_Id* 是商店的标识，*catalog_Id* 是商店产品目录的标识。

   注: 如果您接收到 Error 500，则您可能需要重新启动 WebSphere Application Server 以刷新您的企业应用程序。

2. 单击“服务”标题下的**注册**链接，然后单击“新建顾客”标题下的**注册**。使用 `wctester@wc` 电子邮件地址和 `wctester1` 密码，注册新顾客。以测试值填写其它字段，并单击**提交**。保留浏览器打开。

3. DB2 打开 DB2 控制中心，并执行以下操作:
   a. 单击**交互式**选项卡
   b. 在**命令**字段中，执行以下操作:
      1) 输入

         ```
         connect to your_database_name
         ```

         （其中 *your_database_name* 是 WebSphere Commerce 数据库的名称）并单击“执行”图标。
      2) 输入 `select users_id from USERREG where LOGONID = 'wctester@wc'` 并单击“执行”图标。

c. “查询结果”选项卡显示在步骤 2 中所注册的顾客的相应条目。在此处记下顾客的 USERS_ID 值: _____

d. 更新新注册的顾客的奖金点数余额。单击“交互式”选项卡，并在**命令**字段中，输入以下内容:

```
update BONUS set BONUSPOINT = 1000 where MEMBERID = users_id
```

其中 *users_id* 是步骤 3c 中的值。单击“执行”图标。

4. Oracle 通过执行以下操作，更新测试用户的奖金点数余额:

a. 打开 Oracle SQL Plus 命令窗口（**开始 > 程序 > Oracle > 应用程序开发 > SQL Plus**）。

b. 在**用户名**字段，输入 Oracle 用户名。

c. 在**密码**字段中，输入 Oracle 密码。

d. 在**主机字符串**字段，输入连接字符串。

e. 输入 `select users_id from USERREG where LOGONID = 'wctester@wc';` 以确定用户的标识。

f. 显示步骤 2 中注册的顾客条目。在此处记下顾客的 USERS_ID 值: _____

g. 通过输入以下内容，更新新注册顾客的奖金点数余额:

```
update BONUS set BONUSPOINT = 1000 where MEMBERID = users_id;
```

其中 *users_id* 是步骤 4f 中的值。

h. 输入以下命令提交数据库的更改:

```
commit;
```

并按 Enter 键运行 SQL 语句。

5. 在浏览器中，单击**男式服装**链接查看商店的“男式服装”部分。

6. 单击此特色特价商品的链接查看产品页面。该页面显示一般价格以及基于顾客奖金点数余额的折扣价格。

第 5 部分 附录

附录 A. 启动和停止 WebSphere Test Environment

本部分描述在 VisualAge for Java 中启动 WebSphere Test Environment 的步骤。通常，启动此环境需要执行以下步骤:

1. 启动持久名称服务器。
2. 启动 EJB 服务器。
3. 启动小服务程序引擎。

每个步骤将在后面的章节中加以描述。

要停止此测试环境，请按相反步骤操作。

注: 要想使用 WebSphere Test Environment 的所有功能，在启动 WebSphere Test Environment 前应确保 WebSphere Application Server 不在运行。关于停止 WebSphere Application Server 的信息，请参阅《*WebSphere Commerce 安装指南*》。

启动和停止持久名称服务器

持久名称服务器从客户机接收企业 bean 的查找请求。所有企业 bean 都向持久名称服务器注册。

要启动或停止持久名称服务器，请执行以下操作:

1. 从 VisualAge for Java 的**工作空间**菜单中选择**工具 > WebSphere Test Environment**。
   “WebSphere Test Environment 控制中心”将打开。
2. 单击**持久名称服务器**，然后单击以下选项之一:
   - **启动名称服务器**。
     请等待“服务器开始工作”消息出现在“控制台”窗口中。启动此服务器可能需要花费几分钟。
   - **停止名称服务器**。

启动和停止 EJB 服务器

EJB 服务器是高级别进程或应用程序，它提供了运行时环境以来支持使用企业 bean 的服务器应用程序的执行。

要启动或停止 EJB 服务器，请执行以下操作:

1. 打开 EJB 服务器配置窗口（单击 EJB 选项卡，然后从 **EJB** 菜单中选择**打开至 > 服务器配置**）。
2. 用鼠标右键单击要启动或停止的 EJB 服务器（例如，**EJB 服务器｛服务器 1｝**）并且执行以下操作之一:
   - 选择**启动服务器**
     请等待“服务器开始工作”消息出现在“控制台”窗口中（您可能需要在控制台中选择 **EJB 服务器**，查看此服务器的状态）。启动此服务器可能要花费 10 或 15 分钟，这取决于您的计算机。
   - 选择**停止服务器**

对于 IDE 中的 Java 程序，控制台是标准的输入和输出设备。要查看特定程序的输出，首先请在“所有程序”窗格中选择此程序，然后“输出”窗格将显示此程序的输出。

启动和停止小服务程序引擎

小服务程序引擎集成 Web 服务器和 WebSphere Application Server 的功能，为测试目的创建环境。

要启动或停止小服务程序引擎，请执行以下操作:

1. 从 VisualAge for Java 的**工作空间**菜单中选择**工具 > WebSphere Test Environment**。
   “WebSphere Test Environment 控制中心”将打开。
2. 单击**小服务程序引擎**，然后单击以下选项之一:
   - **启动小服务程序引擎**。
     当小服务程序引擎已启动，控制台显示 ***小服务程序引擎已启动*** 消息。
   - **停止小服务程序引擎**。

附录 B. 部署详细信息

在您用 VisualAge for Java 创建了定制代码并在 WebSphere Test Environment 中测试了它之后，您必须将其部署到在 WebSphere Test Environment 之外运行的 WebSphere Commerce 实例。此 WebSphere Commerce 实例能在您的开发机器上本地运行，或者在另一台机器（使用相同或者不同的操作系统）上运行。

此附录描述了在部署定制代码到在 WebSphere Test Environment 之外运行的 WebSphere Commerce 实例时所需的步骤。您应当参阅第 171 页的『代码部署』以理解部署过程的高级步骤，然后参阅此附录以获得详细信息。

映射到集成的文件系统（iSeries）

400 仅当您的目标 WebSphere Commerce Server 在 iSeries 平台上运行时，本节才适用。

如果您的目标 WebSphere Commerce Server 正在 iSeries 平台上运行，则必须将开发机器上的本地驱动器映射到 iSeries 服务器上的集成文件系统（IFS）。在此文档的后续章节中，*iSeries_drive* 用于指代这个映射到 IFS 的本地驱动器。另外，*drive* 用于开发机器上的本地驱动器（没有映射到 IFS 的驱动器）。

用于定制命令和数据 bean 的 JAR 文件

定制的命令和数据 bean 必须存储在独立于 WebSphere Commerce 代码的项目中。完成在 WebSphere Test Environment 中的测试后，必须为包含定制命令和数据 bean 代码的项目创建 JAR 文件，然后将该 JAR 文件放置在目标 WebSphere Commerce Server 上的适当目录中。

要为定制命令和数据 bean 创建 JAR 文件，请执行以下操作:

1. 在 VisualAge for Java 的工作台中选择**项目**选项卡。
2. 用鼠标右键单击包含定制命令和数据 bean 代码的项目，然后选择**导出**。“导出”智能向导将打开。
3. 选择 **Jar 文件**并单击**下一步**。
4. 在 **Jar 文件**字段，请输入以下内容:
   drive:\WebSphere\CommerceServerDev*temp_directory*\
   *jar_file_name*_1.jar

其中 *drive*:\WebSphere\CommerceServerDev*temp_directory*\ 是有足够可用空间存放 JAR 文件的临时目录，*jar_file_name*_1.jar 是 JAR 文件的名称再附加上 _1。

5. 选择 JAR 文件的属性，如下:

属性	值
类	选中
java	不选中
资源	选中
bean	选中
.class 文件中包含调试属性	选中

对于其它属性接受缺省值。

6. 单击**完成**。

由于所创建的实现 JAR 文件不包含完整的数据包命名信息，因此您必须使用另一个封装实用程序（VisualAge for Java 之外）来重新封装 JAR 文件。要重新封装此文件，请执行以下操作:

1. 在命令窗口中，浏览到在先前步骤中存储了 *jar_file_name*_1.jar 的目录。
2. 输入 mkdir temp1。
3. 输入 cd temp1。
4. 如下设置路径:
   set PATH=%PATH%; *drive*:\WebSphere\WebSphereStudio4\bin;
   其中 *drive* 是安装了 WebSphere Studio 的驱动器。
5. 输入 jar xvf ../*jar_file_name*_1.jar。
6. 输入 jar cvf ../*jar_file_name*.jar * （注意，从名称中除去了 _1）。
7. 切换至 *drive*:\WebSphere\CommerceDev*temp_directory* 目录。
8. 如果您正在部署到本地 WebSphere Commerce 实例，请复制 *jar_file_name*.jar 到以下目录:

   drive:\WebSphere\AppServer\installedApps\
   WC_Enterprise_App_*instanceName*.ear\wcstores.war\WEB-INF\lib

   目录。否则，请将 JAR 文件留在临时目录中，直到需要将所有文件有用资源传送到目标 WebSphere Commerce Server。

为新实体 bean 创建 JAR 文件

在创建新实体 bean 时，必须将代码存储于一个与所有 WebSphere Commerce 代码相分离的项目中。另外，它还必须与所有定制命令和数据 bean 代码相分离。必须在这样的 EJB 组中创建新的实体 bean，它独立于包含 WebSphere Commerce 公共实体 bean 的 EJB 组。

部署新实体 bean 需要创建两个 JAR 文件。第一个 JAR 文件是 EJB 1.1 EXPORT JAR，它是使用选择 EJB 选项卡时可用的工具创建的。第二个 JAR 文件包含实体 bean 的实现代码，该文件是从包含此实体 bean 代码的项目中创建的。第二个 JAR 文件是使用在 VisualAge for Java 工作台中选择“项目”选项卡时可用的工具创建的。

创建 EJB 1.1 Export JAR 文件

要为新的实体 bean 创建 EJB 1.1 Export JAR 文件，请执行以下操作:

1. 在 VisualAge for Java 的工作台中选择 **EJB** 选项卡。
2. 用鼠标右键单击包含定制实体 bean 代码的 EJB 组，并选择**导出 > EJB 1.1 JAR**。
   “导出到 EJB 1.1 JAR 文件”智能向导打开。
3. 在 **Jar 文件**字段中，输入以下内容:
   drive:\WebSphere\CommerceServerDev\temp_directory*jarFileName*_DT.jar
   其中 *drive*:\WebSphere\CommerceServerDev\temp_directory 是有足够可用空间存放 JAR 文件的临时目录，*jarFileName*_DT.jar 是 JAR 文件的名称再加上后缀 _DT。

命名 JAR 文件时加上“_DT”后缀是为了提醒您在将其部署到 WebSphere Commerce 应用程序之前必须通过 WebSphere Application Server 提供的“EJB 部署工具”来运行该 JAR 文件。

4. 选择 JAR 文件的属性，如下:

属性	值
类	选中
java	不选中
资源	选中

属性	值
目标数据库	DB2 如果您正在部署到 DB2 数据库，请选择以下一项: • Windows AIX Solaris Linux **DB2 NT 版，V7.1** • 400 **DB2 AS/400 版，V4** Oracle 如果您正在部署到 Oracle 数据库，请选择 **Oracle，V8**
.class 文件中包含调试属性	选中

对于其它属性接受缺省值。

5. 单击**完成**。

创建 JAR 文件。

创建实现 JAR 文件

要为新实体 bean 创建实现 JAR 文件，请执行以下操作:

1. 在 VisualAge for Java 的工作台中选择**项目**选项卡。
2. 用鼠标右键单击包含定制实体 bean 代码的项目，然后选择**导出**。
   “导出”智能向导将打开。
3. 选择 **Jar 文件**并单击**下一步**。
4. 在 **Jar 文件**字段中，输入以下内容:
   drive:\WebSphere\CommerceServerDev*temp_directory**jarFileName*_1.jar
   其中 *drive*:\WebSphere\CommerceServerDev*temp_directory* 是有足够可用空间存放 JAR 文件的临时目录，*jarFileName*_1.jar 是 JAR 文件的名称再附加上 _1。
5. 选择 JAR 文件的属性，如下:

属性	值
类	选中
java	不选中
资源	选中
bean	选中
.class 文件中包含调试属性	选中

对于其它属性接受缺省值。

6. 单击**完成**。

由于所创建的实现 JAR 文件不包含完整的数据包命名信息，因此您必须使用另一个封装实用程序（VisualAge for Java 之外）来重新封装 JAR 文件。要重新封装此文件，请执行以下操作:

1. 在命令窗口中，浏览到在先前步骤中存储了 *jarFileName*_1.jar 的目录。
2. 输入 mkdir temp1。
3. 输入 cd temp1。
4. 如下设置路径:
   set PATH=%PATH%; *drive*:\WebSphere\WebSphereStudio4\bin;
   其中 *drive* 是安装了 WebSphere Studio 的驱动器。
5. 输入 jar xvf ../*jarFileName*_1.jar。
6. 输入 jar cvf ../*jarFileName*.jar * （注意，从名称中除去了 _1）。
7. 切换至 *drive*:\WebSphere\CommerceServerDev*temp_directory* 目录。
8. 如果您正在部署到本地 WebSphere Commerce 实例，请将 *jarFileName*_1.jar 复制到以下目录:
   drive:\WebSphere\CommerceServer\temp\lib
   否则，将 JAR 文件留在临时目录中，直至您需要将所有文件有用资源传送到目标 WebSphere Commerce Server。

为定制 WebSphere Commerce 实体 bean 创建 JAR 文件

允许您扩展任何公共 WebSphere Commerce 实体 bean。可以在以下 EJB 组中找到这些 bean:

- WCSActrlEJBGroup
- WCSApproval
- WCSAuction
- WCSCatalog
- WCSCommon
- WCSContract
- WCSCoupon
- WCSFulfillment
- WCSInventory
- WCSMessageExtensions
- WCSOrder

- WCSOrderManagement
- WCSOrderStatus
- WCSPayment
- WCSPVCDevices
- WCSTaxation
- WCSUserTraffic
- WCSUser
- WCSUTF

如果您扩展了任何公共 WebSphere Commerce 实体 bean，则您必须为包含已修改的 WebSphere Commerce 公共实体 bean 的 EJB 组创建 EJB EXPORT 1.1 JAR 文件。

创建 EJB 1.1 Export JAR 文件

要为包含已修改的实体 bean 的 EJB 组创建 EJB 1.1 EXPORT JAR 文件，请执行以下操作:

1. 在 VisualAge for Java 的工作台中选择 **EJB** 选项卡。
2. 用鼠标右键单击包含定制实体 bean 代码的 EJB 组，并选择**导出 > EJB 1.1 JAR**。
   “导出到 EJB 1.1 JAR 文件”智能向导打开。
3. 在 **Jar 文件**字段，请输入以下内容:
   drive:\WebSphere\CommerceServerDev*temp_directory*\Cust_*EJBGroupName*-ejb_DT.jar
   其中 *drive*:\WebSphere\CommerceServerDev*temp_directory* 是有足够可用空间存放 JAR 文件的临时目录，Cust_*EJBGroupName*-ejb_DT.jar 是 JAR 文件的名称再加上 _DT 后缀。

命名 JAR 文件时加上“_DT”后缀是为了提醒您在将其部署到 WebSphere Commerce 应用程序之前必须通过 WebSphere Application Server 提供的“EJB 部署工具”来运行该 JAR 文件。

4. 选择 JAR 文件的属性，如下:

属性	值
类	选中
java	不选中
资源	选中

属性	值
目标数据库	DB2 如果您正在部署到 DB2 数据库，请选择以下一项: • Windows AIX Solaris Linux **DB2 NT 版，V7.1** • 400 **DB2 AS/400 版，V4** Oracle 如果您正在部署到 Oracle 数据库，请选择 **Oracle，V8**
.class 文件中包含调试属性	选中

对于其它属性接受缺省值。

5. 单击**完成**。

创建客户机 JAR 文件

要创建客户机 JAR 文件，请执行以下操作:

1. 先选择 EJB 选项卡，然后突出显示所有的 WebSphere Commerce EJB 组（它们的名称以 WCS 开头）。突出显示所有的这些组后，用鼠标右键单击并选择**导出 > 客户机 JAR**。
   “导出”智能向导打开。
2. 在 **JAR 文件**字段中，输入以下内容:
   drive:\WebSphere\CommerceServerDev*temp_directory*\wcsejsclient.jar
3. 如下选择属性:

属性	值
bean	选中
类	选中
java	不选中
资源	选中
.class 文件中包含调试属性	选中

对于其它属性接受缺省值。

4. 单击**完成**。

创建 JAR 文件。

在目标 WebSphere Commerce Server 上存储有用资源

必须将与定制代码相关的有用资源复制到目标 WebSphere Commerce Server 中。这些有用资源包含定制命令、数据 bean 和实体 bean 的 JAR 文件。同样还有支持定制的新 JSP 模板和图形。

AIX Solaris Linux 您应当使用在执行《*WebSphere Commerce 安装指南*》『*运行安装后的脚本*』部分中的步骤时创建的用户，在目标 WebSphere Commerce Server 上执行所有的部署步骤。缺省情况下，用户名是 wasuser。此外，确保您的文件有用资源（例如 JAR 文件）以及这些有用资源放置到的目录给此用户授予了读、写和执行文件的许可权。

400 确保 /QIBM/UserData/WebCommerce/instances/*instanceName* 和 /QIBM/UserData/WebCommerce/instances/*instanceName*/working 目录的权限包含用户 QEJB。通过把“数据权限”设置为 *RWX，将此用户添加这两个目录。

下表总结了正在运行 Windows NT 或 Windows 2000 的目标 WebSphere Commerce Server 上存放有用资源的标准目录。

表 12.

有用资源类型	目标 WebSphere Commerce Server 上的目录位置
命令和数据 bean 逻辑的 JAR 文件	*drive*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wcstores.war\WEB-INF\lib
使用 VisualAge for Java 创建的 EJB 1.1 Export JAR	*drive*:\WebSphere\CommerceServer\temp **注**: 然后此文件作为输入传递给 EJBDeploy 工具以生成可以在 WebSphere Application Server V4.0 中使用的部署代码。
EJB 客户机代码的 JAR 文件	*drive*:\WebSphere\CommerceServer\temp\lib **注**: 此文件仅在 EJBDeploy 工具的类路径中使用。
EJB 实现代码的 JAR 文件	*drive*:\WebSphere\CommerceServer\temp\lib **注**: 然后，此文件作为文件有用资源被添加到企业应用程序中。
JSP 模板	*drive*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wcstores.war*storeDir*
图像	*drive*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wcstores.war*storeDir*\images

要在目标 WebSphere Commerce Server 上存储有用资源，请执行以下操作:

1. 定位命令和数据 bean 代码的 JAR 文件。这些文件应当在开发机器上的以下一个目录中:

- Windows *drive*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wcstores.war\WEB-INF\lib
- Windows *drive*:\WebSphere\CommerceServerDev*temp_directory*

2. 将命令和数据 bean 代码的 JAR 文件复制到目标 WebSphere Commerce Server 上的以下一个目录中:
   - Windows *drive*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wcstores.war\WEB-INF\lib
   - AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/WEB-INF/lib
   - Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/WEB-INF/lib
   - Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/WEB-INF/lib
   - 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/WEB-INF/lib
3. 在开发机器的以下目录中定位任何新的 EJB 组以及已修改的 WebSphere Commerce 实体 bean 的 EJB 1.1 EXPORT JAR 文件:

   Windows *drive*:\WebSphere\CommerceServerDev*temp_directory*
4. 将 EJB 1.1 EXPORT JAR 文件复制到目标 WebSphere Commerce Server 上的以下一个目录中:
   - Windows *drive*:\WebSphere\CommerceServer\temp
   - AIX /usr/WebSphere/CommerceServer/temp
   - Solaris /opt/WebSphere/CommerceServer/temp
   - Linux /opt/WebSphere/CommerceServer/temp
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/temp
5. 如果您创建了新的企业 bean, 请在开发机器上的以下目录中定位这些 bean 的实现 JAR 文件:

   Windows *drive*:\WebSphere\CommerceServerDev*temp_directory*
6. 将新企业 bean 的实现 JAR 文件复制到目标 WebSphere Commerce Server 上的以下目录中:

- Windows *drive*:\WebSphere\CommerceServer\temp\lib
- AIX /usr/WebSphere/CommerceServer/temp/lib
- Solaris /opt/WebSphere/CommerceServer/temp/lib
- Linux /opt/WebSphere/CommerceServer/temp/lib
- 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/ temp/lib

7. 如果您已修改了现有的 WebSphere Commerce 实体 bean，请在开发机器上的以下目录中定位客户机 JAR 文件:

   Windows *drive*:\WebSphere\CommerceServerDev*temp_directory*

8. 将客户机 JAR 文件复制到目标 WebSphere Commerce Server 上的以下一个目录中:
   - Windows *drive*:\WebSphere\CommerceServer\temp\lib
   - AIX /usr/WebSphere/CommerceServer/temp/lib
   - Solaris /opt/WebSphere/CommerceServer/temp/lib
   - Linux /opt/WebSphere/CommerceServer/temp/lib
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/ temp/lib
9. 将所有 JSP 模板复制到目标 WebSphere Commerce Server 上的以下目录中:
   - Windows *drive*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wcstores.war*store_directory*
   - AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*
   - Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*
   - Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*
   - 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*

   其中 *store_directory* 是商店的目录。
10. 将所有图像文件复制到目标 WebSphere Commerce Server 上的以下目录中:

- Windows *drive*:\WebSphere\AppServer\installedApps\WC_Enterprise_App_*instanceName*.ear\wcstores.war*store_directory*\images
- AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*/images
- Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*/images
- Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*/images
- 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*/images

其中 *store_directory* 是商店的目录。

更新目标数据库

当您的目标 WebSphere Commerce Server 使用的数据库不同于开发机器使用的数据库时，您必须在目标 WebSphere Commerce Server 使用的数据库上执行所有针对开发数据库的更新。这包含针对新目录或已修改目录的注册的任何更新，针对已创建的附加表的任何更新以及针对任何已创建的新资源所创建访问控制策略的任何更新。

关于装入访问控制策略的信息（包括适用于不同平台的命令语法和目录许可需求），请参阅《*WebSphere Commerce 访问控制指南*》。

400 您负责用一个实用程序来运行 SQL 语句。实现此操作的一种方式是使用 Client Access Express V5R1（完全安装）。要打开此实用程序，请执行以下操作:

1. 打开**操作导航器**。
2. 当操作导航器打开后，要求您注册到特定系统。确保选择目标 iSeries 机器并使用 WebSphere Commerce 实例用户简要表和密码。这确保 WebSphere Commerce 实例用户简要表拥有所有创建的新表。
3. 在左侧面板中单击您的系统，然后用鼠标右键单击**数据库**并从下拉列表中选择**运行 SQL 脚本**。

“运行 SQL 脚本”窗口打开。您可以使用该窗口剪切和粘贴 SQL 语句或打开 SQL 脚本。您可以通过使用“连接 / JDBC”设置选项来设置您的缺省模式。

生成部署代码

本节描述如何使用“EJB 部署工具”生成 EJB 1.1 Export JAR 文件中包含的企业 bean 的部署代码。此工具由 WebSphere Application Server 提供。

要生成部署代码，请执行以下操作:

1. 在目标 WebSphere Commerce Server 上的命令提示符下，浏览至以下目录:
   - Windows *drive*:\WebSphere\CommerceServer\temp
   - AIX /usr/WebSphere/CommerceServer/temp
   - Solaris /opt/WebSphere/CommerceServer/temp
   - Linux /opt/WebSphere/CommerceServer/temp
2. 400 在目标 WebSphere Commerce Server 上的命令提示符下，输入以下命令:

   ```
   STRQSH
   cd /QIBM/UserData/WebCommerce/instances/instanceName/temp
   ```
3. 通过输入以下命令，临时性将该工具添加到系统路径中:
   - Windows PATH=*drive*:\WebSphere\AppServer\deploytool;%PATH%
   - AIX PATH=/usr/WebSphere/AppServer/deploytool:$PATH
   - Solaris PATH=/opt/WebSphere/AppServer/deploytool:$PATH
   - Linux PATH=/opt/WebSphere/AppServer/deploytool:$PATH
   - 400 PATH=/QIBM/ProdData/WebASAdv4/bin:$PATH

     CP=*ClassPathOfDepJARFiles*

     其中 *ClassPathOfDepJARFiles* 是任何从属 JAR 文件的类路径。注意: 如果您修改了现有的 WebSphere Commerce 企业 bean，那么必须在从属 JAR 文件的类路径中包含 wcsejsclient.jar、wcsejbimpl.jar 和 xml4j.jar 文件。这样，下面提供了在已修改现有 WebSphere Commerce 企业 bean 的情况下的示例类路径:

     ```
     CP=/QIBM/UserData/WebCommerce/instances/instanceName/temp/lib/
     wcsejsclient.jar:
     /QIBM/UserData/WebASAdv4/WAS_AdminInstanceName/installedApps/
     ```

```
WC_Enterprise_App_instanceName.ear/lib/wcsejbimpl.jar:
/QIBM/UserData/WebASAdv4/WAS_AdminInstanceName/installedApps/
WC_Enterprise_App_instanceName.ear/lib/xml4j.jar
```

注: 以上类路径示例中的断行仅是为了显示的目的。您应当在一行中输入类路径信息。

4. Solaris Linux 为避免超过命令行界面中的字符限制，将使用脚本调用 ejbdeploy 命令。请执行以下操作来创建此脚本并调用该命令:

   a. 导航到以下目录:

   ```
   /opt/WebSphere/AppServer/bin
   ```

   b. 为脚本创建新文件并将它适当地命名。例如，将文件命名为 myejbd.sh。

   c. 在 myejbd.sh 文件中包含以下信息:

   ```
   #!/bin/sh
   ./ejbdeploy.sh EJBGroupJARFile WorkingDir OutputJARFile
   -nowarn -keep -35 -cp
   ClassPathOfDepJARFiles
   ```

   其中，这些变量的定义与步骤 5 中的那些相同（注意: 不要执行那个步骤）。以下是要在脚本文件中包含的内容的示例，其中带有所包含变量的值:

   ```
   #!/bin/sh
   ./ejbdeploy.sh
   /opt/WebSphere/CommerceServer/temp/CustomizedWCSUserDeployed_DT.jar
   . /opt/WebSphere/CommerceServer/temp/CustomizedWCSUserDeployed.jar
   -nowarn -keep -35 -cp "/opt/WebSphere/CommerceServer/temp/lib/
   wcsejsclient.jar:/opt/WebSphere/AppServer/installedApps/
   WC_Enterprise_App_demo.ear/lib/wcsejbimpl.jar:/opt/WebSphere/
   AppServer/installedApps/WC_Enterprise_App_demo.ear/lib/xml4j.jar"
   ```

   注: ./ejbdeploy.sh 命令后的所有断行仅为了显示。在您的文件中不应在 ./ejbdeploy.sh 命令后断开。

   保存该脚本并使它可执行。

   d. 通过输入以下内容，调用该脚本:

   ```
   ./myejbd.sh
   ```

5. Windows AIX 400 如下输入 ejbdeploy 命令:

   Windows 400

   ```
   ejbdeploy EJBGroupJARFile_DT WorkingDir OutputJARFile -nowarn -keep -35 -cp
   ClassPathOfDepJARFiles
   ```

   AIX

```
/usr/WebSphere/AppServer/bin/ejbdeploy.sh EJBGroupJARFile_DT WorkingDir
OutputJARFile -nowarn -keep -35
-cp
ClassPathOfDepJARFiles
```

其中:

- *EJBGroupJARFile*_DT 是 EJB 组的 JAR 文件的名称。例如，如果您修改 WCSUser EJB 组中的 bean，则在基于 Windows 的平台上的示例用法是

  ```
  drive:\WebSphere\CommerceServer\temp\CustomizedWCSUser_DT.jar
  ```

  400 iSeries 平台的示例用法是

  ```
  /QIBM/UserData/WebCommerce/instances/instanceName/temp/
  CustomizedWCSUser_DT.jar
  ```

- *WorkingDir* 是代码生成所需临时文件的存储目录的名称。
- *OutputJARFile* 是输出 JAR 文件的全限定名称。例如，您可以输入 CustomizedWCSUserDeployed.jar。
- -nowarn 是用于禁止警告消息和信息性消息的可选参数。
- -keep 是用于在已经运行 ejbdeploy 命令后保留工作目录的可选参数。
- -35 是一个强制性参数，它将对 CMP 实体 bean 使用与 “EJB 部署工具”（EJB 部署工具随 WebSphere Application Server 版本 3.5 一起提供）中所用相同的自顶向下的映射规则。
- -cp *ClassPathOfDepJARFiles* 是任何从属 JAR 文件的类路径。当修改了现有的 WebSphere Commerce 企业 bean 后，必须在从属 JAR 文件的类路径中包含 wcsejsclient.jar、wcsejbimpl.jar 和 xml4j.jar 文件。这样，下面提供了在已修改现有 WebSphere Commerce 企业 bean 的情况下的示例类路径:

  ```
  "drive:\WebSphere\CommerceServer\temp\lib\wcsejsclient.jar;
  drive:\WebSphere\AppServer\installedApps\
  WC_Enterprise_App_instanceName.ear \lib\wcsejbimpl.jar;
  drive:\WebSphere\AppServer\installedApps\
  WC_Enterprise_App_instanceName.ear\lib\xml4j.jar;"
  ```

  注: 400 对于类路径值，请输入 -cp $CP 其中 CP 先前已使用适当的类路径条目作了定义。另外，不需要引号。

修改实体 bean 的交易隔离级别

本节描述了如何使用 modifyIsolationLevel 命令行实用程序将实体 bean 的交易隔离级别设置为数据库类型的适当级别。

要运行 modifyIsolationLevel 命令，请执行以下操作:

1. 在目标 WebSphere Commerce Server 上，使用命令提示符导航到以下目录:
   - Windows *drive*:\WebSphere\CommerceServer\bin
   - AIX /usr/WebSphere/CommerceServer/bin
   - Solaris /opt/WebSphere/CommerceServer/bin
   - Linux /opt/WebSphere/CommerceServer/bin
2. 400 输入以下命令:

   ```
   STRQSH
   cd /QIBM/ProdData/WebCommerce/bin
   ```

3. 您必须发出 modifyIsolationLevel 命令，它具有以下一般句法:

   Windows AIX 400

   ```
   modifyIsolationLevel -jarFile jar_file_name.jar
      -logFile log_file_name -dbType  db_type
   ```

   Solaris Linux

   ```
   ./modifyIsolationLevel.sh -jarFile jar_file_name.jar
      -logFile log_file_name -dbType  db_type
   ```

   其中

   - *jar_file_name.jar* 是包含定制代码的 JAR 文件名称
   - *log_file_name* 是应当作为记录信息的位置的全限定文件名
   - *db_type* 是正在使用的数据库类型。输入 DB2 或 ORACLE

   以下是 modifyIsolationLevel 命令的示例，其中所有的值是为 Windows 平台上的使用而指定的:

   ```
   modifyIsolationLevel -jarFile
      D:\WebSphere\CommerceServer\temp\CustomizedWCSUserDeployed.jar
      -logFile D:\WebSphere\CommerceServer\instances\demo\logs\output.log
      -dbType DB2
   ```

   400 以下是 modifyIsolationLevel 命令的示例，其中所有的值是为在 iSeries 上使用而指定的:

```
modifyIsolationLevel -jarFile
   /QIBM/UserData/WebCommerce/instances/instanceName/temp/
   CustomizedWCSUserDeployed.jar -logFile /QIBM/UserData/WebCommerce/
   instances/instanceName/logs/output.log -dbType DB2
```

注意，在以上示例中，断行仅是为了显示的目的。

注: 参数名称区分大小写。也就是说，jarFile 与 jarfile 不一样。确保正确输入参数名称。

如果没有异常显示在命令窗口中，则命令已经成功运行。完成后，请注意在部署的 JAR 文件上的时间戳记已经更改。

导出当前 WebSphere Commerce 企业应用程序

此部分描述了如何使用 WebSphere Application Server 管理控制台导出当前 WebSphere Commerce 企业应用程序。

要从 WebSphere Application Server 中导出当前企业应用程序，请执行以下操作:

1. 请确保您在目标 WebSphere Commerce Server 上具有以下目录:
   - Windows *drive*:\WebSphere\CommerceServer\working
   - AIX /usr/WebSphere/CommerceServer/working
   - Solaris /opt/WebSphere/CommerceServer/working
   - Linux /opt/WebSphere/CommerceServer/working
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/working

   如果您还没有此目录，请现在就创建它。

   注: AIX Solaris Linux 确保 /working 目录的许可权设置给了在执行《*WebSphere Commerce 安装指南*》『*运行安装后的脚本*』部分中的步骤时创建的用户。

   400 确保 /QIBM/UserData/WebCommerce/instances/*instanceName* 和 /QIBM/UserData/WebCommerce/instances/ *instanceName*/working 目录的权限包含用户 QEJB。通过把“数据权限”设置为 *RWX，将此用户添加这两个目录。
2. 打开 WebSphere Application Server 管理控制台。
3. 展开 **WebSphere 管理域**。
4. 展开**企业应用程序**。

5. 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，展开 **demo** 应用程序，并选择**导出应用程序**。
6. 在**导出目录**字段中，输入以下内容:
   - Windows *drive*:\WebSphere\CommerceServer\working
   - AIX /usr/WebSphere/CommerceServer/working
   - Solaris /opt/WebSphere/CommerceServer/working
   - Linux /opt/WebSphere/CommerceServer/working
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/working

   这将导出整个应用程序，并将所有资源包含到 WC_Enterprise_App_*instanceName*.ear 文件中（其中 *instanceName* 是您的 WebSphere Commerce 实例的名称）。

导出企业 bean 的配置信息

本节描述了如何使用 XMLConfig 命令行实用程序的 -export 选项导出包含在现有企业应用程序中的企业 bean 配置信息。

在导出现有应用程序中的 bean 信息之后，您必须手工添加任何添加到应用程序中的新 bean 的信息。

要导出此配置信息，请执行以下操作:

1. 将 was.export.app.xml 文件从目标 WebSphere Commerce Server 上的以下目录中:
   - Windows *drive*:\WebSphere\CommerceServer\xml\config
   - AIX /usr/WebSphere/CommerceServer/xml/config
   - Solaris /opt/WebSphere/CommerceServer/xml/config
   - Linux /opt/WebSphere/CommerceServer/xml/config
   - 400 /QIBM/ProdData/WebCommerce/xml/config

   复制到目标 WebSphere Commerce Server 的以下目录中:
   - Windows *drive*:\WebSphere\CommerceServer\working
   - AIX /usr/WebSphere/CommerceServer/working
   - Solaris /opt/WebSphere/CommerceServer/working

- Linux /opt/WebSphere/CommerceServer/working
- 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/working

  其中

  – *instanceName* 是您的 WebSphere Commerce 实例的名称。

2. Windows AIX Solaris Linux 在文本编辑器中打开 was.export.app.xml 文件。在此文件中，将 $Enterprise_Application_Name$ 所有出现的地方替换为

   WebSphere Commerce Enterprise Application - *instanceName*

   其中 *instanceName* 是您的 WebSphere Commerce 实例的名称（例如 demo）。保存此文件。

   注: 您正在插入的值必须与显示在 WebSphere Application Server 管理控制台中的实例信息匹配。

3. 400 在文本编辑器中打开 was.export.app.xml 文件。在此文件中，将 $Enterprise_Application_Name$ 所有出现的地方替换为

   instanceName - WebSphere Commerce 企业应用程序

   其中 *instanceName* 是 WebSphere Commerce 实例的名称（例如，demo）。保存此文件。

   注: 您正在插入的值必须与显示在 WebSphere Application Server 管理控制台中的实例信息匹配。

4. Windows AIX Solaris Linux 在命令提示符下，浏览至以下目录:
   - Windows *drive*:\WebSphere\CommerceServer\working
   - AIX /usr/WebSphere/CommerceServer/working
   - Solaris /opt/WebSphere/CommerceServer/working
   - Linux /opt/WebSphere/CommerceServer/working
5. 400 在命令提示符下，输入以下命令:

   ```
   STRQSH
   PATH=/QIBM/ProdData/WebASAdv4/bin:$PATH
   cd /QIBM/UserData/WebCommerce/instances/instanceName/working
   ```

6. Windows AIX Solaris Linux 通过输入以下命令，调用 XMLConfig 工具来执行部分导出:

   Windows

```
xmlConfig -export OutputFile.xml -partial was.export.app.xml
   -adminNodeName wasHostName
```

AIX

```
/usr/WebSphere/AppServer/bin/XMLConfig.sh  -export OutputFile.xml
   -partial was.export.app.xml -adminNodeName wasHostName
 -nameServiceHost wasHostName -nameServicePort wasAdminPort
```

Solaris Linux

```
/opt/WebSphere/AppServer/bin/XMLConfig.sh  -export OutputFile.xml
   -partial was.export.app.xml -adminNodeName wasHostName
   -nameServiceHost wasHostName -nameServicePort wasAdminPort
```

其中

- *wasHostName* 是 WebSphere Application Server 中包含当前企业应用程序的节点的名称。
- `OutputFile.xml` 是作为运行此命令的结果而创建的文件的名称，`was.export.app.xml` 是您在步骤 2 中修改的文件。
- *wasAdminPort* 是 WebSphere Application Server 管理端口。

7. 400 通过输入以下命令，调用 XMLConfig 工具执行部分导出:

```
xmlConfig -export OutputFile.xml -partial was.export.app.xml
-adminNodeName wasHostName -nameServiceHost wasHostName
-nameServicePort wasAdminPort -instance wasInstanceName
```

其中

- *wasHostName* 是 WebSphere Application Server 中包含当前企业应用程序的节点的名称。

  注: *wasHostName* 的值是区分大小写的，且必须与 TCP/IP 配置中的值匹配。（使用命令行处理器来访问 “CFGTCP，选项 12” 来验证主机名。）
- `OutputFile.xml` 是作为运行此命令的结果而创建的文件的名称，`was.export.app.xml` 是您在步骤 3 中修改的文件。
- *wasAdminPort* 是 WebSphere Application Server 管理端口。
- *wasInstanceName* 是 WebSphere Application Server 实例名称。

在您导出了当前企业应用程序中包含的每个 bean 的配置信息之后，您必须添加新节来描述将要添加到应用程序的每个企业 bean。例如，如果您有一个称为 “Bonus” 的新实体 bean，您必须添加一个节来描述该 Bonus bean。另外，您必须替换配置文件中的一个变量来指定企业应用程序的 .ear 文件的确切名称。

要对 OutputFile.xml 文件做出这些更新，请执行以下操作:

1. 浏览至目标 WebSphere Commerce Server 上的以下目录:
   - Windows *drive*:\WebSphere\CommerceServer\working
   - AIX /usr/WebSphere/CommerceServer/working
   - Solaris /opt/WebSphere/CommerceServer/working
   - Linux /opt/WebSphere/CommerceServer/working
   - 400 /QIBM/UserData/WebCommerce/instances/ *instanceName*/working
2. 在文本编辑器中打开 OutputFile.xml 文件。
3. 定位 <ear-file-name> 标记，并将该值替换为以下值:
   - Windows *drive*:\WebSphere\CommerceServer\working\WC_Enterprise_App_*instanceName*.ear
   - AIX /usr/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
   - Solaris /opt/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
   - Linux /opt/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/working/WC_Enterprise_App_*instanceName*.ear
4. 您还必须为每个添加到企业应用程序的新 bean 添加新节。以下示例显示如何添加新 bean（称为 “Bonus” ）。

   Windows AIX Solaris Linux

```
<ejb-module name="yourEJBGroup">
   <jar-file>yourDeployedJarFile.jar</jar-file>
   <module-install-info>
      <application-server-full-name>/NodeHome:$hostName$/EJBServerHome:
         WebSphere Commerce Server - instanceName/
      </application-server-full-name>
   </module-install-info>
   <ejb-module-binding>
      <data-source>
         <jndi-name>jdbc/WebSphere Commerce DB2 DataSource instanceName
            </jndi-name>
         <default-user>user</default-user>
         <default-password>password</default-password>
      </data-source>
      <enterprise-bean-binding name="BeanBindingName">
```

```
         <jndi-name>instanceNameJNDINameOfBean</jndi-name>
      </enterprise-bean-binding>
   </ejb-module-binding>
</ejb-module>
```

400

```
<ejb-module name="yourEJBGroup">
   <jar-file>yourDeployedJarFile.jar</jar-file>
   <module-install-info>
      <application-server-full-name>/NodeHome:$hostName$/EJBServerHome:
         instanceName - WebSphere Commerce Server/
      </application-server-full-name>
   </module-install-info>
   <ejb-module-binding>
      <data-source>
         <jndi-name>jdbc/instanceName WebSphere Commerce DB2 DataSource
            </jndi-name>
         <default-user>user</default-user>
         <default-password>password</default-password>
      </data-source>
      <enterprise-bean-binding name="BeanBindingName">
         <jndi-name>instanceNameJNDINameOfBean</jndi-name>
      </enterprise-bean-binding>
   </ejb-module-binding>
</ejb-module>
```

其中

- *yourEJBGroup* 是包含正添加到企业应用程序的 bean 的 EJB 组名称。
- *yourDeployedJarFile* 是包含 EJB 组部署代码的 JAR 文件的名称。
- *instanceName* 是您要对其部署代码的 WebSphere Commerce 实例的名称。
- *user* 是您的数据库用户名。
- *password* 是数据库用户的密码。
- *BeanBindingName* 是企业 bean 的绑定名称。例如，对于名为 Bonus 的 bean，这就是 Bonus_Binding。
- *instanceNameJNDINameOfBean* 是前面附加有 WebSphere Commerce 实例的企业 bean 的 JNDI 名称。此 JNDI 名称必须与企业 bean 的名称精确匹配。示例值为 democom/ibm/commerce/sample/objects/Bonus。在本示例中，“demo”是实例名称，“com/ibm/commerce/sample/objects/Bonus”是企业 bean 的 JNDI 名称。此值必须在一行中输入。
  您可以使用 VisualAge for Java 或应用程序组装工具来验证 JNDI 名称。要使用 VisualAge for Java 验证 JNDI 名称，请执行以下操作:
  a. 用鼠标右键单击 bean，选择**属性**。
     显示 JNDI 名称。

可以使用应用程序组装工具验证 JNDI 名称，但这要求您已经将新的企业 bean 组装到企业应用程序中。应用程序组装工具是在 WebSphere Application Server 管理控制台中从“工具”菜单进行访问的。

要使用应用程序组装工具验证 JNDI 名称，请执行以下操作:

a. 打开包含 bean 的 .ear 文件。

b. 展开包含 bean 的 EJB 模块。

c. 选择 bean。

d. 单击**绑定**选项卡。
显示 JNDI 名称。

注意，使用这些方法之一显示 JNDI 名称，您仍必须在向 XML 文件添加信息时预先附加 WebSphere Commerce 实例的名称。

注:

a. 以上节中的断行仅为了显示。

b. 请确保 **$hostName$** 值与当前管理节点服务器名称匹配。另外，请确保在此行中没有回车符。

c. `<application-server-full-name>` 规范不能跨多行。

d. 如果您正在使用 Oracle 数据库，您必须修改数据源信息。更改取自先前代码片段的以下行:

```
<jndi-name>jdbc/WebSphere Commerce DB2 DataSource instanceName
   </jndi-name>
```

为以下行:

```
<jndi-name>jdbc/WebSphere Commerce Oracle DataSource instanceName
   </jndi-name>
```

5. 如果您正在部署已修改的 WebSphere Commerce 实体 bean，您必须更新那些 bean 的节以反映包含已修改 bean 部署代码的 JAR 文件的名称。

6. 保存 `OutputFile.xml` 文件。

将新的企业 bean 组装到企业应用程序中

本节描述了如何使用应用程序组装工具将新的企业 bean 组装到现有的企业应用程序中。

400 因为应用程序组装工具在 Windows 平台上运行，所以在提示输入全限定路径名时，您必须引用已映射到 iSeries IFS 的驱动器。这称为 *iSeries_drive*。

AIX Solaris Linux 建议增加 assembly.sh 文件中的内存堆大小值来避免在保存新的或已修改的 .ear 文件时内存不足。

此文件位于以下目录中:

- AIX /usr/WebSphere/AppServer/bin
- Solaris /opt/WebSphere/AppServer/ bin
- Linux /opt/WebSphere/AppServer/bin

要增加内存堆大小，请修改以下行:

```
$JAVA_HOME/jre/bin/java
```

从而使它如下显示:

```
$JAVA_HOME/jre/bin/java -mx512M
```

在此步骤中，在应用程序组装器工具中打开在章节导出当前 WebSphere Commerce 企业应用程序中创建的企业应用程序的 .ear 文件。一旦在该工具中打开了它，请执行以下任务将新的实体 bean 添加到企业应用程序:

1. 导入包含新实体 bean 的 EJB 组。新 EJB 组的 JAR 文件将存储在企业应用程序的“EJB 模块”部分中。
2. 设置新实体 bean 的类路径以包含实现 JAR 文件。
3. 将该实现 JAR 文件添加到应用程序中。此 JAR 文件将存储在企业应用程序的“文件”部分中。
4. 为新实体 bean 中包含的方法设置 WebSphere Application Server 安全性。

要将新的 EJB 组组装到企业应用程序中，请执行以下操作:

1. Windows AIX Solaris Linux 通过在目标 WebSphere Commerce Server 上执行以下操作，备份当前的企业应用程序:
   a. 在命令提示符下，导航到以下目录:
      - Windows *drive*:\WebSphere\CommerceServer\working
      - AIX /usr/WebSphere/CommerceServer/working
      - Solaris /opt/WebSphere/CommerceServer/working
      - Linux /opt/WebSphere/CommerceServer/working
   b. 制作现有 WC_Enterprise_App_*instanceName*.ear 文件的副本并将其命名为 WC_Enterprise_App_*instanceName*.ear.bak。

2. 400 通过执行以下操作，备份当前的企业应用程序:
   a. 在命令提示符下，输入以下命令:

      ```
      STRQSH
      cd /QIBM/UserData/WebCommerce/instances/instanceName/working
      cp WC_Enterprise_App_instanceName.ear
      WC_Enterprise_App_instanceName.ear.bak
      ```

   b. 要避免因本地客户机和运行 WebSphere Application Server 的 iSeries 机器之间的数据传送而导致的不必要的长时间等待，请在本地客户机上创建以下目录，然后将 WC_Enterprise_App_*instanceName*.ear 文件复制到该目录:
      drive:\WebSphere\CommerceServer\working
      其中 *drive* 是本地驱动器。

      注: 如果在您的本地机器上不存在 *drive*:\WebSphere\CommerceServer\working 目录，请现在就创建它。
3. 打开 WebSphere Application Server 管理控制台。
4. 从**工具**菜单中，选择**应用程序组装工具**。
5. 如果打开了“欢迎”窗口，选择**取消**关闭该窗口。
6. 通过执行以下操作，打开您将在其上工作的企业应用程序:
   a. 从**文件**菜单中，选择**打开**。
   b. 在**文件名**字段中，输入:
      - Windows *drive*:\WebSphere\CommerceServer\working\WC_Enterprise_App_*instanceName*.ear
      - AIX /usr/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
      - Solaris /opt/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
      - Linux /opt/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
      - 400 *drive*:\WebSphere\CommerceServer\working\WC_Enterprise_App_*instanceName*.ear

      并单击**打开**。等待应用程序打开后，继续到后续步骤。这需要花费几分钟。
7. 用鼠标右键单击 **EJB 模块**，并选择**导入**。
8. 在**文件名称**字段中，输入以下内容:

- Windows *drive*:\WebSphere\CommerceServer\temp\ *yourDeployedJarFile*.jar
- AIX /usr/WebSphere/CommerceServer/temp/ *yourDeployedJarFile*.jar
- Solaris /opt/WebSphere/CommerceServer/temp/ *yourDeployedJarFile*.jar
- Linux /opt/WebSphere/CommerceServer/temp/ *yourDeployedJarFile*.jar
- 400 *iSeries_drive*:\QIBM\UserData\WebCommerce\instances\ *instanceName*\temp*yourDeployedJarFile*.jar

其中

- *yourDeployedJarFile* 是包含 EJB 组部署代码的 JAR 文件的名称。
- *iSeries_drive* 是映射到 iSeries IFS 的本地驱动器。

单击**打开**，然后在“确认值”窗口中，单击**确定**。

9. 一旦导入了 *yourDeployedJarFile*.jar 文件，请滚动到 *yourEJBGroup* EJB 组（其中 *yourEJBGroup* 是您的 EJB 组的名称），并选择此组。
有关该组的信息显示在右侧的窗格中。
10. 在新企业 bean 的类路径字段中，输入任何从属 JAR 文件。例如，您可以输入相应的实现 JAR 文件和 WebSphere Commerce 实体 bean 的实现 JAR 文件，如下所示：：

    lib/*yourImplJarFile*.jar lib/wcsejbimpl.jar
11. 单击**应用**。
12. 通过执行以下操作，将您的 EJB 组的实现 JAR 文件添加到应用程序:
    a. 用鼠标右键单击企业应用程序的**文件**节点，选择**添加文件**。（企业应用程序的**文件**节点的位置接近分层树的底部。注意：对于企业应用程序中的组件有其它的“文件”节点，但您必须选择整个应用程序的“文件”节点。）
    b. 在“添加文件”窗口中，单击**浏览**。
    c. 导航到以下目录:
       - Windows *drive*:\WebSphere\CommerceServer\temp
       - AIX /usr/WebSphere/CommerceServer/temp
       - Solaris /opt/WebSphere/CommerceServer/temp
       - Linux /opt/WebSphere/CommerceServer/temp
       - 400 *iSeries_drive*:\QIBM\UserData\WebCommerce\instances\ *instanceName*\temp

d. 先突出显示此目录，然后单击**选择**。

e. 返回到“添加文件”窗口。注意，显示临时目录的内容。突出显示 `lib` 目录。

`lib` 目录的内容显示在右边的窗格中。

f. 在右侧的窗格中，选择 *`yourImplJarFile`* 文件并单击**添加**。然后该文件显示在“选定的文件”窗格中。

g. 单击**确定**。

13. 通过执行以下操作,配置实体 bean 的安全性:

a. 先展开“EJB 模块”节点，然后找到并展开 *YourEJBGroup* 节点。

b. 展开**实体 Bean**。

c. 展开 *yourEntityBean*，其中 *`yourEntityBean`* 是您的实体 bean 的名称。

d. 单击**方法扩展**，然后在右边的窗格中执行以下操作:

1) 单击**高级**选项卡。

2) 确保已经选择**安全性身份**。

3) 对于每种方法，确保已经选择**使用 EJB 服务器的身份**。

4) 单击**应用**（如果您做出了任何修改）。

e. 在左侧导航窗格中，用鼠标右键单击 *YourEJBGroup* EJB 组下的**安全性角色**，并选择**新建**，然后执行以下操作:

1) 在**名称**字段中，输入 `WCSecurityRole`，并单击**应用**。注意，如果已经存在此角色，您就无需执行此步骤了。

f. 在左侧导航窗格中，用鼠标右键单击 *YourEJBGroup* EJB 组下的**方法许可**，选择**新建**，然后执行以下操作:

1) 在**方法许可名称**字段中，输入 `WCMethodPermission`

2) 在**方法**选择区域中，单击**添加**。
“添加方法”窗口打开。

3) 展开 *yourDeployedJarFile*.**jar**，再展开 **Bonus**，然后展开每个方法**本地**和**远程**列表。

4) 按住 Shift 键保持不动，并选择所有的本地方法，单击**确定**。

5) 重复方法选择过程同样地添加远程方法（如果远程方法存在）。

6) 在“角色”选择区域中，单击**添加**，选择 `WCSecurityRole` 并单击**确定**。

7) 单击**应用**。

14. 从**文件**菜单中，选择**保存**。

15. 关闭应用程序组装工具。

16. 400 将新修改的 WC_Enterprise_App_*instanceName*.ear 文件从本地机器中复制到运行 WebSphere Application Server 的 iSeries 机器上。即: 将文件从以下目录中:

    drive:\WebSphere\CommerceServer\working

    复制到以下目录中:

    iSeries_drive:\QIBM\UserData\WebCommerce\instances*instanceName*\working

    其中 *iSeries_drive* 是您已映射到 iSeries IFS 的盘符。

完成本步骤后，您就已经创建包含所有先前逻辑以及新业务逻辑的新企业应用程序。这些逻辑正是新修改的 WC_Enterprise_App_*instanceName*.ear 文件中包含的所有内容。

将修改的企业 bean 组装到企业应用程序中

本节描述了如何使用应用程序组装工具将已修改的 WebSphere Commerce 企业 bean 组装到企业应用程序中。

在本步骤中，您在应用程序组装器工具中打开企业应用程序。一旦在该工具中打开了它，您可以执行以下任务将已修改的 WebSphere Commerce企业 bean 包含到企业应用程序:

1. 为已修改的现有版本 EJB 组制作类路径的副本。
2. 除去您已修改的现有版本的 EJB 组。
3. 导入已修改的 EJB 组的新版本。新 EJB 组的 JAR 文件存储在企业应用程序的 “EJB 模块” 部分中。
4. 设置已修改的 EJB 组的类路径。
5. 为已修改的实体 bean 中包含的方法设置 WebSphere Application Server 安全性。

AIX Solaris Linux 建议增加 assembly.sh 文件中的内存堆大小值来避免在保存新的或已修改的 .ear 文件时内存不足。

此文件位于以下目录中:

- AIX /usr/WebSphere/AppServer/bin
- Solaris /opt/WebSphere/AppServer/ bin
- Linux /opt/WebSphere/AppServer/bin

要增加内存堆大小，请修改以下行:

```
$JAVA_HOME/jre/bin/java
```

从而使它如下显示:

```
$JAVA_HOME/jre/bin/java -mx512M
```

要将修改的 EJB 组组装到企业应用程序中，请执行以下操作:

1. Windows AIX Solaris Linux 通过在目标 WebSphere Commerce Server 上执行以下操作，备份当前的企业应用程序:
   a. 在命令提示符下，导航到以下目录:
      - Windows *drive*:\WebSphere\CommerceServer\working
      - AIX /usr/WebSphere/CommerceServer/working
      - Solaris /opt/WebSphere/CommerceServer/working
      - Linux /opt/WebSphere/CommerceServer/working
   b. 制作现有 WC_Enterprise_App_*instanceName*.ear 文件的副本并将其命名为 WC_Enterprise_App_*instanceName*.ear.bak。
2. 400 通过执行以下操作，备份当前的企业应用程序:
   a. 在命令提示符下，输入以下命令:

      ```
      STRQSH
      cd /QIBM/UserData/WebCommerce/instances/instanceName/working
      cp WC_Enterprise_App_instanceName.ear
      WC_Enterprise_App_instanceName.ear.bak
      ```

   b. 要避免因本地客户机和运行 WebSphere Application Server 的 iSeries 机器之间的数据传送而导致的不必要的长时间等待，请在本地客户机上创建以下目录，然后将 WC_Enterprise_App_*instanceName*.ear 文件复制到该目

录:
drive:\WebSphere\CommerceServer\working
其中 *drive* 是本地驱动器。

注: 如果在您的本地机器上不存在 *drive*:\WebSphere\CommerceServer\working 目录，请现在就创建它。

3. 打开 WebSphere Application Server 管理控制台。
4. 从**工具**菜单中，选择**应用程序组装工具**。
5. 通过执行以下操作，打开要处理的企业应用程序:
   a. 从**文件**菜单中，选择**打开**。
   b. 在**文件名**字段中，输入:
      - Windows *drive*:\WebSphere\CommerceServer\working\WC_Enterprise_App_*instanceName*.ear
      - AIX /usr/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
      - Solaris /opt/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
      - Linux /opt/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
      - 400 *drive*:\WebSphere\CommerceServer\working\WC_Enterprise_App_*instanceName*.ear

      并单击**打开**。等待应用程序打开后，继续到后续步骤。这需要花费几分钟。
6. 单击 **EJB 模块**。右边的窗格显示企业应用程序中的 EJB 模块。
7. 单击您已修改的 EJB 组的 EJB 模块。例如，如果您已修改了 WCSUser EJB 组中的 bean，您应该单击 **WCSUser**。
8. 单击“常规”选项卡查看类路径信息。将这个现有的类路径信息复制到一个文本文件中（例如，WCSUser_path.txt）。
9. 用鼠标右键单击 EJB 模块并选择**删除**。
10. 用鼠标右键单击 **EJB 模块**，并选择**导入**。
11. 在**文件名称**字段中，输入以下内容:
    - Windows *drive*:\WebSphere\CommerceServer\temp\Cust_*EJBGroupName*-ejb.jar

-
- AIX /usr/WebSphere/CommerceServer/temp/Cust_*EJBGroupName*-ejb.jar
- Solaris /opt/WebSphere/CommerceServer/temp/Cust_*EJBGroupName*-ejb.jar
- Linux /opt/WebSphere/CommerceServer/temp/Cust_*EJBGroupName*-ejb.jar
- 400 *iSeries_drive*:\QIBM\UserData\WebCommerce\instances*instanceName*\temp\Cust_*EJBGroupName*-ejb.jar

并单击**打开**。在“确认值”窗口中，单击**确定**。

12. 一旦导入了 *EJBGroupJARFile*.jar 文件，请滚动到已修改的 EJB 组，并选择此组。
    有关该组的信息显示在右侧的窗格中。
13. 打开包含先前版本 EJB 组的类路径信息的文本文件。选择并复制该类路径。
14. 在已修改的 EJB 组的 **Classpath** 字段中，粘贴这个类路径信息。
15. 单击**应用**。
16. 从**文件**菜单中，选择**关闭**。
17. 等待文件关闭，然后从**文件**菜单中，选择**打开**，重新打开 WC_Enterprise_App_*instanceName*.ear 文件。
18. 通过执行以下操作,配置已修改的 bean 的安全性:
    a. 先展开“EJB 模块”节点，然后找到并展开已修改的 EJB 组的节点。
    b. 展开**实体 Bean**。
    c. 展开已修改的 EJB 组。
    d. 单击**方法扩展**，然后在右边的窗格中执行以下操作:
       1) 单击**高级**选项卡。
       2) 确保已经选择**安全性身份**。
       3) 对于每种方法，确保已经选择**使用 EJB 服务器的身份**。
       4) 单击**应用**（如果您做出了任何修改）。
    e. 在左侧导航窗格中，用鼠标右键单击已修改的 EJB 组下的**安全性角色**，选择**新建**，然后执行以下操作:
       1) 在**名称**字段中，输入 WCSecurityRole，并单击**应用**。注意，如果已经存在此角色，您就无需执行此步骤了。
    f. 在左侧导航窗格中，用鼠标右键单击已修改的 EJB 组下的**方法许可**，选择**新建**，然后执行以下操作:

1) 在**方法许可名称**字段中，输入 WCMethodPermission
2) 在**方法**选择区域中，单击**添加**。
“添加方法”窗口打开。
3) 展开 *modifiedEJBGroup*，选择所有的企业 bean（在选择时按住 Shift 键）。单击**确定**。然后，所有的企业 bean 都显示在“企业 bean”列下，而所有的方法都显示在“类型”列下。
4) 在“角色”选择区域中，单击**添加**，选择 WCSecurityRole 并单击**确定**。
5) 单击**应用**。

19. 从**文件**菜单中，选择**保存**。
20. 关闭应用程序组装工具。
21. 400 将新修改的 WC_Enterprise_App_*instanceName*.ear 文件从本地机器中复制到运行 WebSphere Application Server 的 iSeries 机器上。即：将文件从以下目录中：

    drive:\WebSphere\CommerceServer\working

    复制到以下目录中：

    iSeries_drive:\QIBM\UserData\WebCommerce\instances*instanceName*\working

    其中 *iSeries_drive* 是您已映射到 iSeries IFS 的盘符。

完成本步骤后，您就已经创建包含所有先前逻辑以及新业务逻辑的新企业应用程序。这些逻辑正是新修改的 WC_Enterprise_App_*instanceName*.ear 文件中包含的所有内容。

停止并除去企业应用程序

本节描述了如何使用WebSphere Application Server 管理控制台停止当前正在运行的企业应用程序，然后除去它。

要停止并除去您的企业应用程序，请执行以下操作：

1. 打开 WebSphere Application Server 管理控制台。
2. 展开 **WebSphere 管理域**。
3. 展开**节点**。
4. 展开 ***nodeName***（其中 ***nodeName*** 是节点的名称）。
5. 展开**应用程序服务器**。
6. Windows AIX Solaris Linux 用鼠标右键单击您的 WebSphere Commerce 应用程序服务器。例如，用鼠标右键单击 **WebSphere Commerce Server - *nodeName*** 并选择**停止**。

7. 400 用鼠标右键单击您的 WebSphere Commerce 应用程序服务器。例如，用鼠标右键单击 ***instanceName*** **- WebSphere Commerce Server** 并选择**停止**。
8. 展开**企业应用程序**。
9. Windows AIX Solaris Linux 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 **WebSphere Commerce 企业应用程序 - *instanceName*** 应用程序并选择**停止**。
10. 400 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 ***instanceName*** **- WebSphere Commerce 企业应用程序**应用程序并选择**停止**。
11. 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 **WebSphere Commerce 企业应用程序 - *instanceName***（或者 ***instanceName*** **- WebSphere Commerce 企业应用程序** iSeries 版）应用程序并选择**除去**。
12. 当提示指出应用程序是否应导出时，选择**否**。
13. 当提示指出是否应除去应用程序时，请选择**是**。

导入企业应用程序

本节描述了如何使用 XMLConfig 命令行实用程序来导入企业应用程序。

要导入新的企业应用程序，请执行以下操作:

1. Windows AIX Solaris Linux 通过输入以下命令，调用 XMLConfig 工具来将企业应用程序导入到 WebSphere Application Server 中:

   Windows

   ```
   xmlConfig -import OutputFile.xml -adminNodeName wasHostName
   ```

   AIX

   ```
   /usr/WebSphere/AppServer/bin/XMLConfig.sh -import OutputFile.xml
      -adminNodeName wasHostName
    -nameServiceHost wasHostName -nameServicePort wasAdminPort
   ```

   Solaris Linux

   ```
   /opt/WebSphere/AppServer/bin/XMLConfig.sh -import OutputFile.xml
      -adminNodeName wasHostName
    -nameServiceHost wasHostName -nameServicePort wasAdminPort
   ```

   其中

- *wasHostName* 是 WebSphere Application Server 中包含当前企业应用程序的节点的名称。
- *OutputFile.xml* 是描述您所有的企业 bean 的 XML 文件。
- *wasAdminPort* 是 WebSphere Application Server 管理端口。

2. 400 通过输入以下命令，调用 XMLConfig 工具将企业应用程序导入到 WebSphere Application Server 中:

```
STRQSH
PATH=/QIBM/ProdData/WebASAdv4/bin:$PATH
xmlConfig -import
/QIBM/UserData/WebCommerce/instances/instanceName/working/OutputFile.xml
-adminNodeName wasHostName
 -nameServiceHost wasHostName -nameServicePort wasAdminPort
-instance wasInstanceName
```

其中

- *OutputFile.xml* 是描述所有企业 bean 的 XML 文件的全限定名称。
- *wasHostName* 是 WebSphere Application Server 中包含当前企业应用程序的节点的名称。

  注: 400 *wasHostName* 的值是区分大小写的，且必须与 TCP/IP 配置中的值匹配。（使用命令行处理器来访问 “CFGTCP，选项 12” 来验证主机名。）
- *wasAdminPort* 是 WebSphere Application Server 管理端口。
- *wasInstanceName* 是 WebSphere Application Server 实例名称。

400 当试图运行 XMLConfig -import 命令时，您可能接收到以下错误消息: “无法展开 /QIBM/UserData/WebAsAdv4/*wasInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear 下的 ear 文件”。如果接收到此消息，请除去或重命名上述目录并再次运行该命令。

---

3. 400 当导入企业应用程序后，您必须运行一个脚本来修改目录许可权。要运行此脚本，请执行以下操作:
   - 在命令提示符下，输入以下命令:

     ```
     STRSQH
     cd /QIBM/ProdData/WebCommerce/bin
     changeAuthority wasAdminInstanceName instanceName
     ```

     其中

– *wasAdminInstanceName* 是 WebSphere Application Server 管理实例的名称。
– *instanceName* 是您的 WebSphere Commerce 实例的名称。

启动企业应用程序

本节描述了如何使用WebSphere Application Server 管理控制台刷新视图，然后启动企业应用程序。

1. 打开 WebSphere Application Server 管理控制台。
2. 展开 **WebSphere 管理域**，然后展开**节点**，再展开 *nodeName*
3. 突出显示 *nodeName* 节点。
4. 单击**刷新选定的子树**图标。
5. 通过执行以下操作，启动您的 WebSphere Commerce 应用程序:
   - 展开**应用程序服务器**。
   - Windows AIX Solaris Linux 用鼠标右键单击 WebSphere Commerce 应用程序。例如，用鼠标右键单击 **WebSphere Commerce Server -** *instanceName*，并选择**启动**。
   - 400 用鼠标右键单击您的 WebSphere Commerce 应用程序。例如，用鼠标右键单击 *instanceName* **- WebSphere Commerce Server** 应用程序，并选择**启动**。

附录 C. VisualAge for Java 的技巧

本部分描述了在开发环境中与疑难解答、性能改进和简化相关的技巧。

更改在 WebSphere Test Environment 中的小服务程序引擎的属性

WebSphere Test Environment 中小服务程序引擎的属性由 default.servlet_engine 属性文件控制。在此文件中，您可以修改 Web 服务器的文档根路径，并且可以更改 WebSphere Test Environment 使用的端口。

在 WebSphere Test Environment 已停止运行，且重新启动不是选项的情况下，您可能希望更改端口。要更改端口，请执行以下操作:

1. 在文本编辑器中打开 *VAJ_install_path*\IDE\ProjectResources\IBM WebSphere Test Environment\properties\default.servlet_engine 文件。
2. 在 <transport> 节中，将以下行中的 8080 值更改为可用的端口。

   ```
   <arg name="port" value="8080"/>
   ```

3. 将以下行中的 8080 值更改为以上指定同一个端口。

   ```
   <hostname-binding hostname="localhost:8080" servlethost="default_host"/>
      <hostname-binding hostname="127.0.0.1:8080" servlethost="default_host"/>
   ```

4. 保存该文件。
5. 打开 WebSphere Test Environment 控制中心并启动小服务程序引擎。

缺省情况下，WebSphere Test Environment 的文档根路径为 *VAJ_install_path*\IDE\ProjectResources\IBM WebSphere Test Environment\hosts\default_host\default_app\web。要将它切换到其它目录，请执行以下操作:

1. 打开 WebSphere Test Environment 控制中心并停止小服务程序引擎。
2. 在文本编辑器中打开 *VAJ_install_path*\IDE\ProjectResources\IBM WebSphere Test Environment\properties\default.servlet_engine 文件。
3. 在 <websphere-webgroup name="default_app"> 节中，将 <document-root>$approot$/web</document-root> 设置为以下值:

   ```
   <document-root>your_document_root</document-root>
   ```

   其中 *your_document_root* 为所希望的文档根路径。

4. 将以下行中的 8080 值更改为以上指定同一个端口。

```
<hostname-binding hostname="localhost:8080" servlethost="default_host"/>
   <hostname-binding hostname="127.0.0.1:8080" servlethost="default_host"/>
```

5. 保存该文件。
6. 打开 WebSphere Test Environment 控制中心并启动小服务程序引擎。

解决持久名称服务器问题

如果您的持久名称服务器遇到问题，可能需要删除并重新创建持久名称服务器数据库。关于创建此数据库的详细信息，请参阅《*Commerce Studio 安装指南*》。

作为选择，如果有消息显示端口当前正在使用中，请确保您没有运行 WebSphere Application Server。

删除已编译的 JSP 文件

由于性能原因，VisualAge for Java 包含了一个存储已编译的 JSP 文件的项目文件夹。可能需要多次删除已编译的 JSP 文件。例如，如果从 JSP 文件中除去数据 bean，当您下次调用 JSP 文件时可能会出现错误。这种情况下，您可以删除已编译的 JSP 文件。

要删除已编译的 JSP 文件，请执行以下操作:

1. 在 VisualAge for Java 中，选择“项目”选项卡。
2. 向下滚动至 **JSP 页面编译生成代码**项目。
3. 选择整个项目（如果要删除的编译 JSP 文件很多）或者扩展此项目，然后仅删除需要重新编译的文件。

声明

本信息是为美国提供的产品和服务而编写的。IBM 可能在其它国家或地区不提供本文档中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代理咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可证。您可以用书面方式将许可证查询寄往:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

有关双字节（DBCS）信息的许可证查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

本条款不适用联合王国或任何这样的条款与当地法律不一致的国家或地区:

国际商业机器公司以“按现状”的基础提供本出版物，不附有任何形式的（无论是明示的，还是默示的）保证，包括（但不限于）对非侵权性、适销性和适用于某特定用途的默示保证。某些国家或地区在某些交易中不允许免除明示或默示的保证。因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本出版物的新版本中。IBM 可以随时对本出版物中描述的产品和／或程序进行改进和／或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。该 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的: （i）允许在独立创建的程序和其它程序（包括本程序）之间进行信息交换，以及（ii）允许对已经交换的信息进行相互使用，请与下列地址联系:

IBM Canada Ltd.
Office of the Lab Director
8200 Warden Avenue, Markham, Ontario L6G 1C7
Canada

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际程序许可证协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其它操作环境中获得的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其它可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其它关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

所有 IBM 的价格均是 IBM 当前的建议零售价，可随时更改而不另行通知。经销商的价格可与此不同。

本信息仅为了规划目的。其中的信息在描述的产品可以使用之前会得到更改。

本信息包含日常事务运作中使用的数据和报告的示例。为了尽可能完全地说明它们，示例中包含个体、公司、品牌和产品。所有这些名称都是虚构的，任何与实际商务企业使用的名称和地址的雷同纯属巧合。

版权许可证:

此信息包含用源语言表示的样本应用程序，说明了在各种操作平台上的编程技术。您可以按任何形式复制、修改和分发这些样本程序，用于开发、使用、买卖或分发符合操作平台（已为此平台写样本程序）的应用程序编程接口的应用程序，而无须向 IBM 支付任何费用。这些示例尚未在所有条件下经过全面测试。因此，IBM 不能保证或暗示这些程序的可靠性、适用性和功能。您可以按任何形式复制、修改和分发这些样本程序，用于开发、使用、买卖或分发符合 IBM 应用程序编程接口的应用程序，而无须向 IBM 支付任何费用。

这些样本程序的每份副本或任何部分或任何派生产品必须包含如下版权声明:

如果正在查看本信息的软拷贝，照片和彩色图例可能不会显示。

商标和服务标记

以下术语是国际商业机器公司在美国和／或其它国家或地区的商标或注册商标:

400	iSeries
AIX	MQSeries
AS/400	Net.Commerce
CICS	Net.Data
DB2	VisualAge
IBM	WebSphere

Windows 和 Windows NT 是 Microsoft Corporation 在美国和／或其它国家或地区的商标或注册商标。

Oracle 是 Oracle Corporation 在美国和／或其它国家或地区的注册商标。

Solaris、Java 和所有基于 Java 的商标和徽标是 Sun Microsystems, Inc. 在美国和／或其它国家或地区的商标或注册商标。

其它公司、产品和服务名称可能是其它公司的商标或服务标记。

索引

[T]

[X]

[Y]

[特别字符]

C

E

F

J

M

U

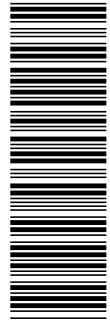
V

W

部件号: CT024SC

中国印刷

(1P) P/N: CT024SC



G152-0293-00

Spine information:

IBM WebSphere Commerce

程序员指南

版本 5.4