IBM WebSphere Commerce

编程指南与教程

版本 5.5

IBM WebSphere Commerce

编程指南与教程

版本 5.5

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，第一次修订（2003 年 9 月）

本版本适用于 IBM WebSphere Commerce Business Edition V5.5、IBM WebSphere Commerce - Express V5.5 和 IBM WebSphere Commerce Professional Edition V5.5（产品号 5724-A18）及所有后续发行版和修订版，直到在新版本中另有声明为止。

它也适用于所有后续发行版和修订版，直到在新版本中另有声明为止。确保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。

IBM 欢迎您提出宝贵意见。您可以使用在线 IBM WebSphere Commerce 文档反馈表单发送您的意见，该表单在以下 URL 处提供:

http://www..ibm.com/software/webservers/commerce/rcf.html

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

*《WebSphere Commerce 编程指南与教程》*提供了关于 WebSphere® Commerce 体系结构和程序设计模块的信息。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
- 修改现有控制器命令
- 扩展对象模型并修改现有的任务命令
- 扩展现有的 WebSphere Commerce 实体 bean

更改摘要

可从 WebSphere Commerce 技术资料库中获取本文档的最新版本（PDF 格式）。

本书最新版本中的更新由 Express 图像进行标识，以指示特定于 IBM WebSphere Commerce - Express 的信息。

本书的更新

还可从以下 WebSphere Commerce Web 站点的 Technical Library 部分获取本书及本书任何更新版本的 PDF 格式文件:

http://www.ibm.com/software/commerce/library/

有关其它支持信息，请访问 WebSphere Commerce 支持站点:

http://www.ibm.com/software/commerce/support/

也可从位于以下 Web 站点的 WebSphere Developer Domain 的 WebSphere Commerce Zone 获取本书的更新版本:

http://www.ibm.com/websphere/developer/zones/commerce

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。

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

AIX 指示特定于 WebSphere Commerce for AIX® 的信息。

Linux 指示特定于 WebSphere Commerce for Linux 的信息。

Solaris 指示特定于 WebSphere Commerce for Solaris Operating Environment 软件的信息。

Windows 指示特定于 WebSphere Commerce for Windows® 2000 的信息。

DB2 指示特定于 DB2 Universal Database™ 的信息。

Oracle 指示特定于 Oracle 的信息。如果您正在使用 WebSphere Commerce Business Edition 或 WebSphere Commerce Professional Edition，则可使用 Oracle 作为数据库管理系统。

Business 指示特定于 IBM WebSphere Commerce Business Edition 的信息。

Express 指示特定于 IBM WebSphere Commerce - Express 的信息。

Professional 指示特定于 IBM WebSphere Commerce Professional Edition 的信息。

Developer 指示特定于 WebSphere Commerce 开发环境的信息。对于 WebSphere Commerce Business Edition 和 WebSphere Commerce Professional Edition，开发环境为 WebSphere Commerce Studio V5.5。

对于 WebSphere Commerce - Express，开发环境为 WebSphere Commerce - Express Developer Edition V5.5

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™
- EnterpriseJavaBeans 组件体系结构
- JavaServer Pages 技术
- HTML
- 数据库技术
- Business Professional WebSphere Studio Application Developer V5

  Express WebSphere Studio Application Developer V5.1

路径变量

本指南使用以下变量来表示目录路径:

WC_installdir

这是 WebSphere Commerce 的安装目录。以下为不同操作系统上 WebSphere Commerce 的缺省安装目录:

- 400 /QIBM/ProdData/CommerceServer55
- AIX /usr/WebSphere/CommerceServer55
- Linux /opt/WebSphere/CommerceServer55
- Solaris /opt/WebSphere/CommerceServer55
- Windows C:\Program Files\WebSphere\CommerceServer55

WC_userdir

WebSphere Commerce 使用的所有数据的目录，可由用户修改或需要由用户进行配置。

- 400 /QIBM/UserData/CommerceServer55

WAS_installdir

这是 WebSphere Application Server 的安装目录。以下为不同操作系统上 WebSphere Application Server 的缺省安装目录:

- 400 /QIBM/ProdData/WebAs5/Base
- AIX /usr/WebSphere/AppServer
- Linux /opt/WebSphere/AppServer
- Solaris /opt/WebSphere/AppServer
- Windows C:\Program Files\WebSphere\AppServer

WAS_userdir

WebSphere Application Server 使用的所有数据的目录，可由用户修改或需要由用户进行配置。

- 400 QIBM/UserData/WebAS5/Base/*WAS_instancename*

  而 *WAS_instance_name* 表示与 WebSphere Commerce 实例相关联的 WebSphere Application Server 的名称。

WCDE_installdir

WebSphere Commerce 开发环境的安装目录。对于 WebSphere Commerce Business Edition 和 WebSphere Commerce Professional Edition，开发环境为 WebSphere Commerce Studio V5.5。以下是缺省安装目录: C:\WebSphere\CommerceStudio55。

对于 WebSphere Commerce - Express，开发环境为 WebSphere Commerce - Express Developer Edition V5.5。以下是缺省安装目录: C:\WebSphere\CommerceDev55

何处可以找到更多信息

关于 WebSphere Commerce 的更多信息，请参阅以下 Web 站点:

http://www.ibm.com/software/commerce/library/

目录

第 1 部分 概念和体系结构

第 1 章 概述

WebSphere Commerce 软件组件

在检查 WebSphere Commerce Server 如何工作之前，从宏观上查看与 WebSphere Commerce 相关的软件组件是很有用的。下图显示了这些软件产品的简化视图:

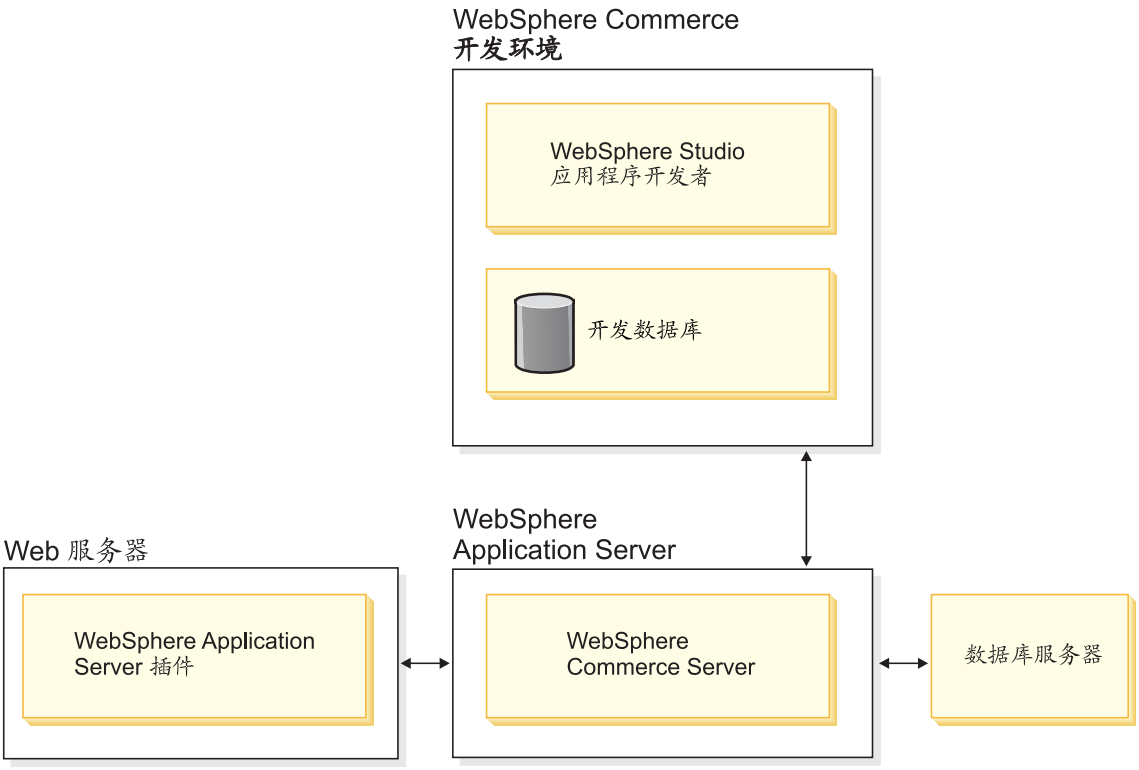

图 1.

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，使它能够利用该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 WebSphere Studio Application Developer 执行以下任务:

- 创建并定制商店前台有用资源，例如 JSP 模板和 HTML 页面
- 用 Java 创建一个新的业务逻辑
- 用 Java 修改现有的业务逻辑
- 测试代码和商店前台有用资源

WebSphere Commerce 开发环境使用一个开发数据库。开发者可以使用其偏好的数据库工具（包括 WebSphere Studio Application Developer）修改数据库。

WebSphere Commerce 应用程序体系结构

既然您已经知道了与 WebSphere Commerce 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:

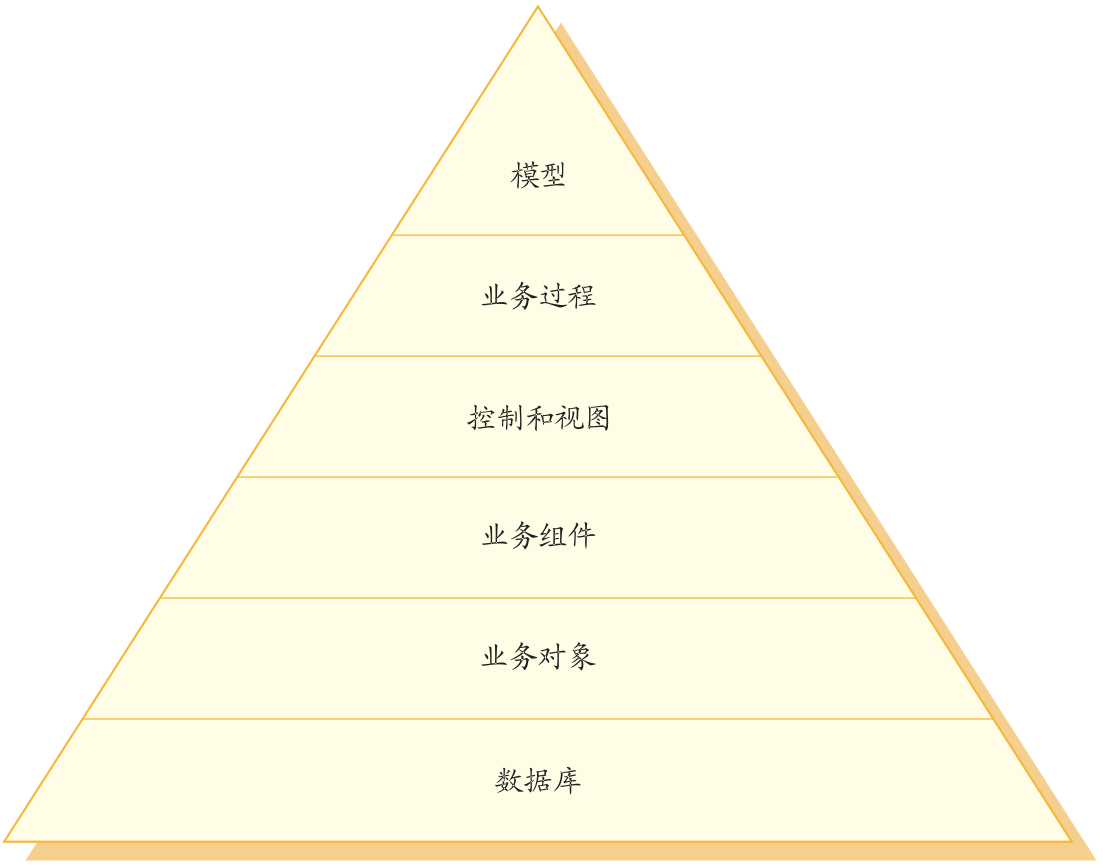

图 2.

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。

- USERS
- ORDERS
- INVENTORY

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 充当业务组件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 ProductDisplayView（返回显示购物者选定产品相关信息的产品页面）和 OrderCancelView。

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:

创建电子邮件竞销

此业务过程包含与创建电子邮件竞销的过程所涉及的所有步骤相关的业务组件和视图。

准备联机产品目录

此业务过程包含与创建联机产品目录相关的业务组件和子过程。这包含设计产品目录、装入产品目录数据、创建销售策略关联以及设置定价信息。

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“时尚潮流”样本商店中显示的客户导向模型。 Business 另一个示例是“多乐五金”样本商店中显示的 B2B 导向模型。

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:

- servlet 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 组件之间的交互作用。关于每个组件的更详细信息，请参阅后面各部分。

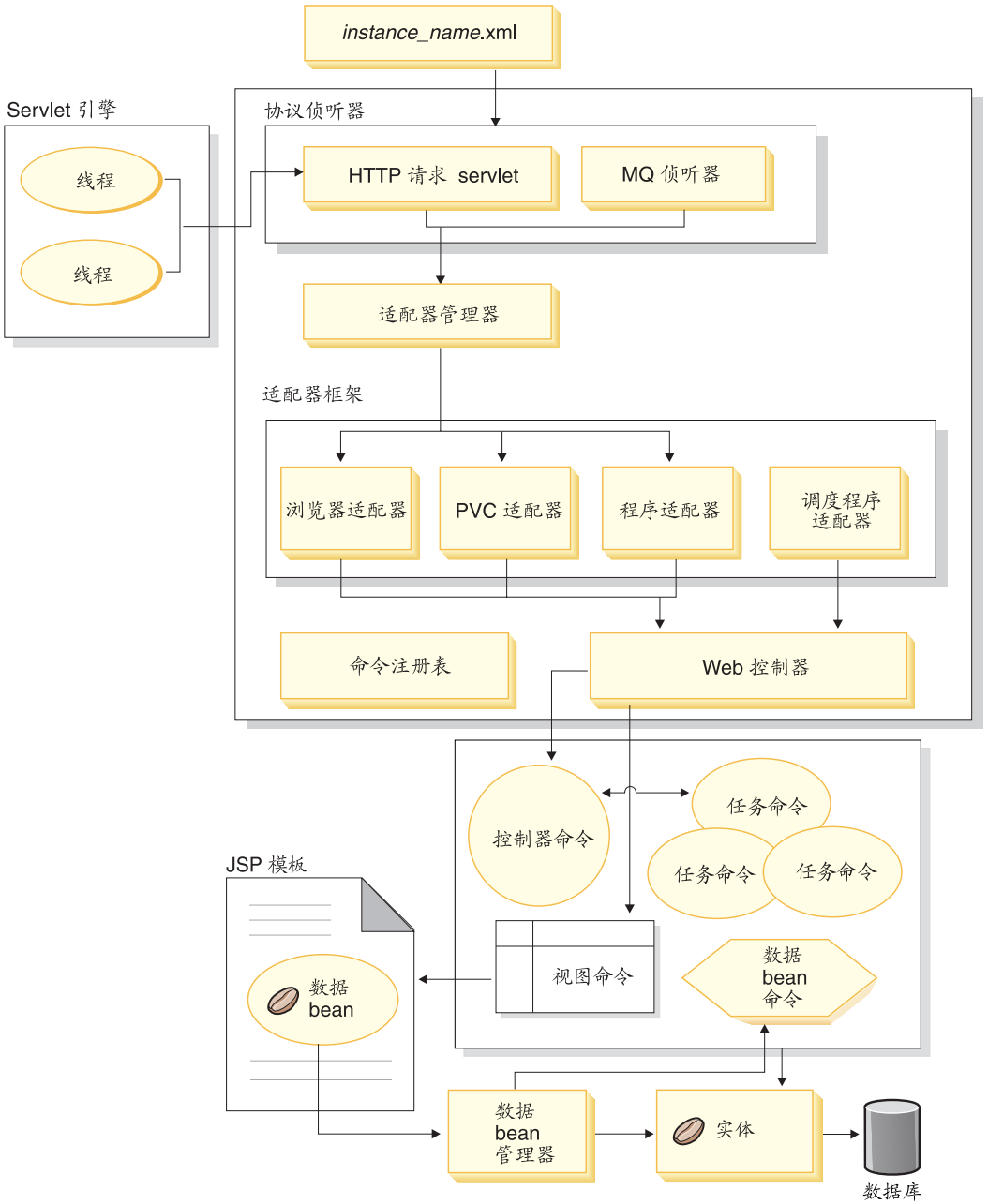
instance_name.xml
Servlet 引擎
线程
线程
协议侦听器
HTTP 请求 servlet
MQ 侦听器
适配器管理器
适配器框架
浏览器适配器
PVC 适配器
程序适配器
调度程序
适配器
命令注册表
Web 控制器
控制器命令
任务命令
任务命令
任务命令
JSP 模板
数据
bean
视图命令
数据
bean
命令
数据
bean
管理器
实体
数据库

图 3.

servlet 引擎

servlet 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。 servlet 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。

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 servlet
- MQSeries 侦听器

当请求 servlet 接收到来自 servlet 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 servlet 后，该 servlet 即读取 *instance_name*.xml 配置文件（其中 *instance_name* 是 WebSphere Commerce 实例的名称）。XML 文件中的配置块中的一个块将定义所有适配器。请求 servlet 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。

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于设备的组件。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。

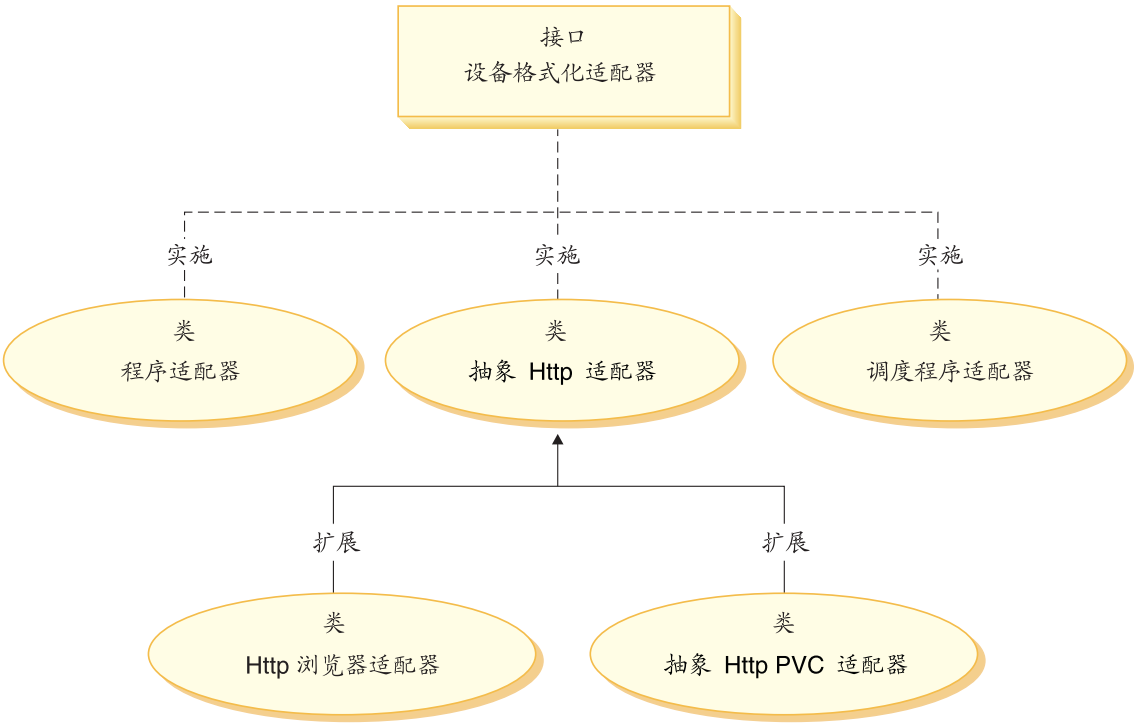

图 4.

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。控制器命令的示例包括用于订单处理的 OrderProcessCmd 命令和允许用户登录的 LogonCmd 命令。通常，控制器命令包含控制语句（例如，if、then 和 else），并调用任务命令来执行业务流程中的个别任务。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

在实例化数据 bean 时，数据 bean 命令由数据 bean 管理器调用。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 45 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

插入至 JSP 模板的 WebSphere Commerce 数据 bean 允许将动态内容包含至页面中。数据 bean 管理器激活数据 bean，以便在以下这行代码插入页面时填充其值:

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

其中 *data_bean* 是要激活的数据 bean 而 *request* 是 HTTPServletRequest 对象。

JavaServer Pages 模板

JSP 模板是通常用来显示的专用 servlet。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 servlet，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 servlet 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 servlet 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 83 页的第 4 章，『访问控制』。

instance_name.xml 配置文件

instance_name.xml 配置文件（其中 *instance_name* 是 WebSphere Commerce 实例的名称）设置 WebSphere Commerce 实例的配置信息。在初始化请求 servlet 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。

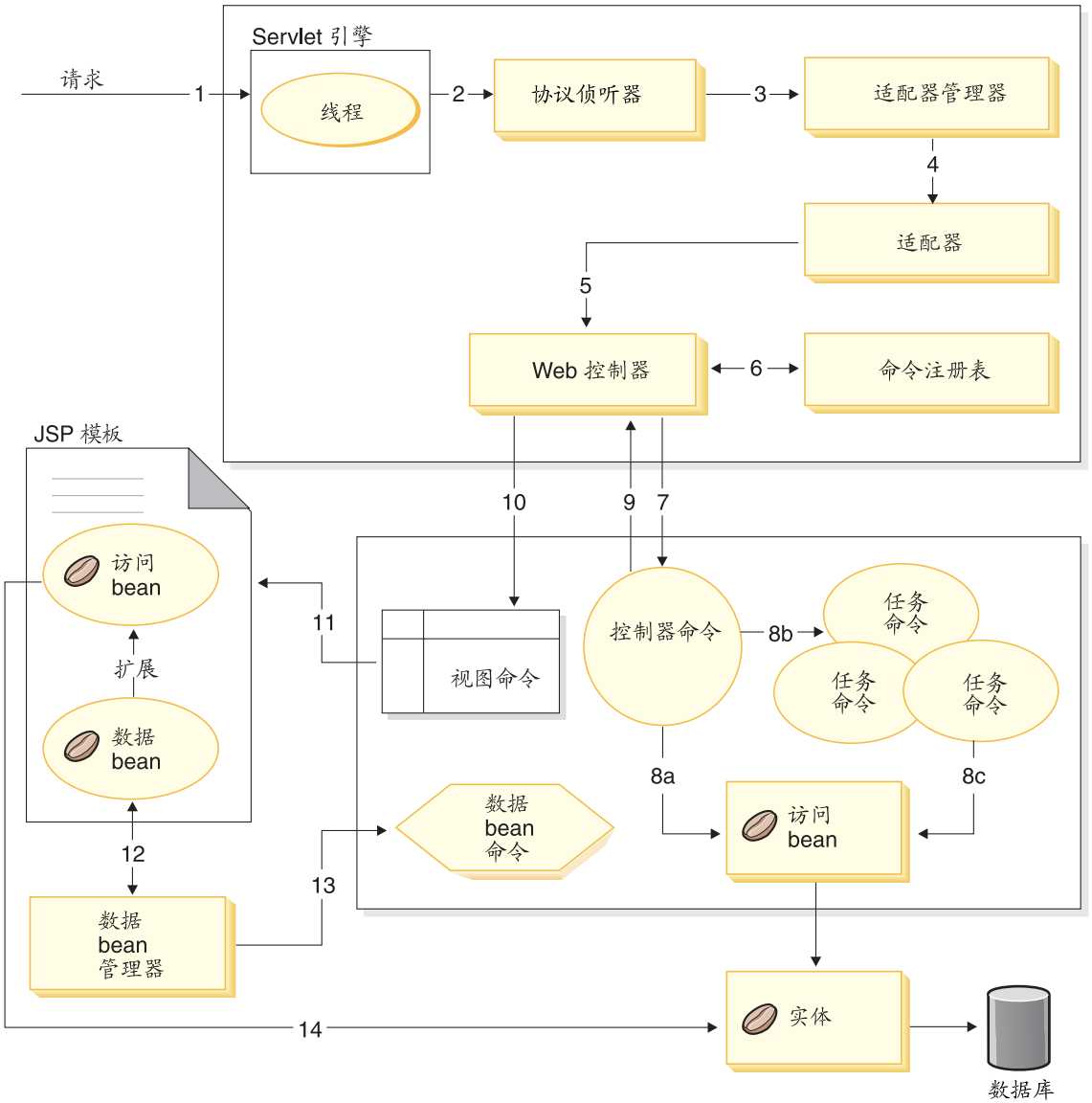

图 5.

以下信息与上图相对应。

1. 该请求通过 WebSphere Application Server 插件定向至 servlet 引擎。
2. 在它自己的线程中执行此请求。servlet 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 servlet 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，存在多个可能的路径:
   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 访问数据库（如图 5 中的 8c 所示）。
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。

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 – 控制器（MVC）设计模式指定应用程序由数据模型、表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。

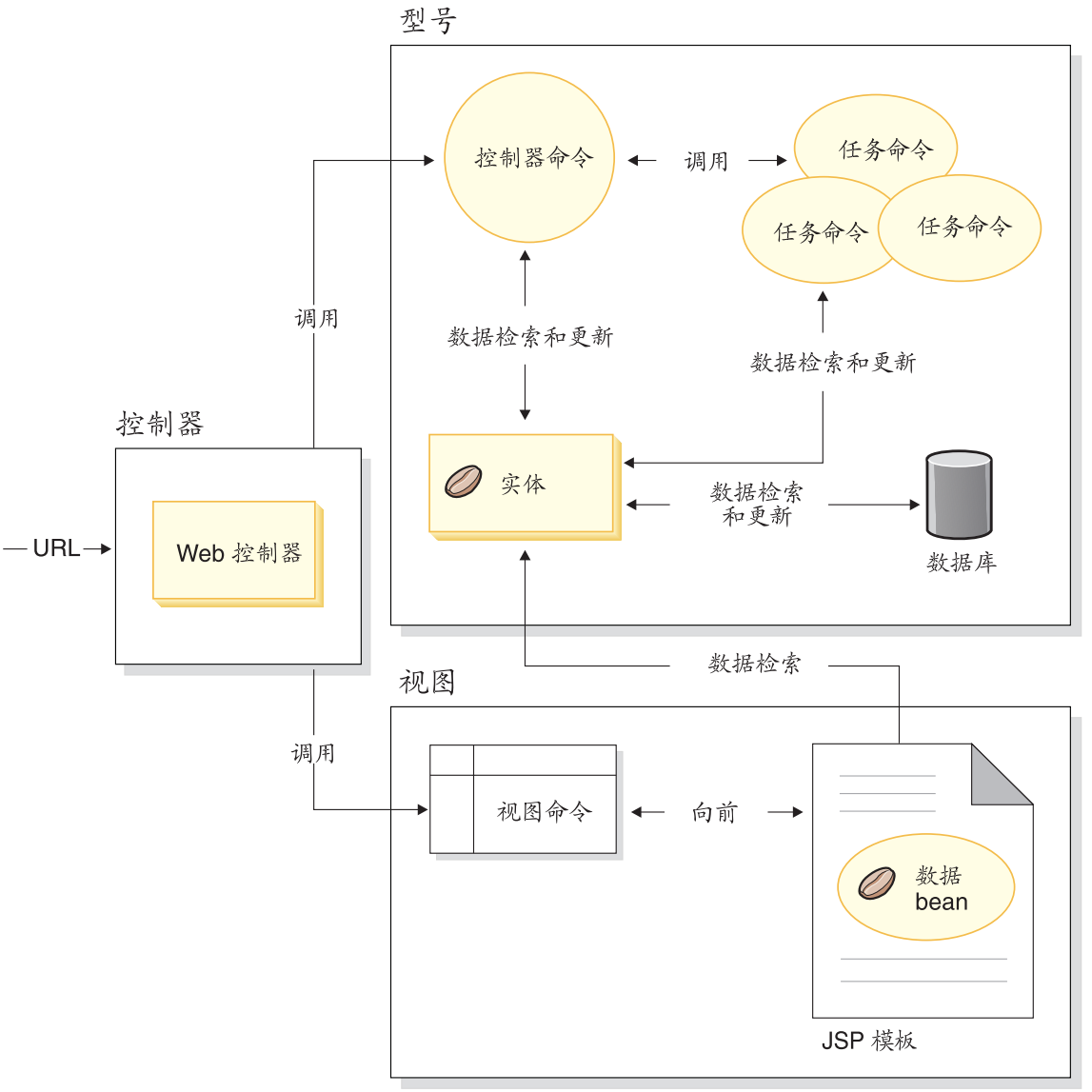
型号
控制器命令
调用
任务命令
任务命令
任务命令
调用
数据检索和更新
数据检索和更新
控制器
实体
数据检索
和更新
URL
Web 控制器
数据库
数据检索
视图
调用
视图命令
向前
数据
bean
JSP 模板

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() 方法以及检索输出属性。

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:

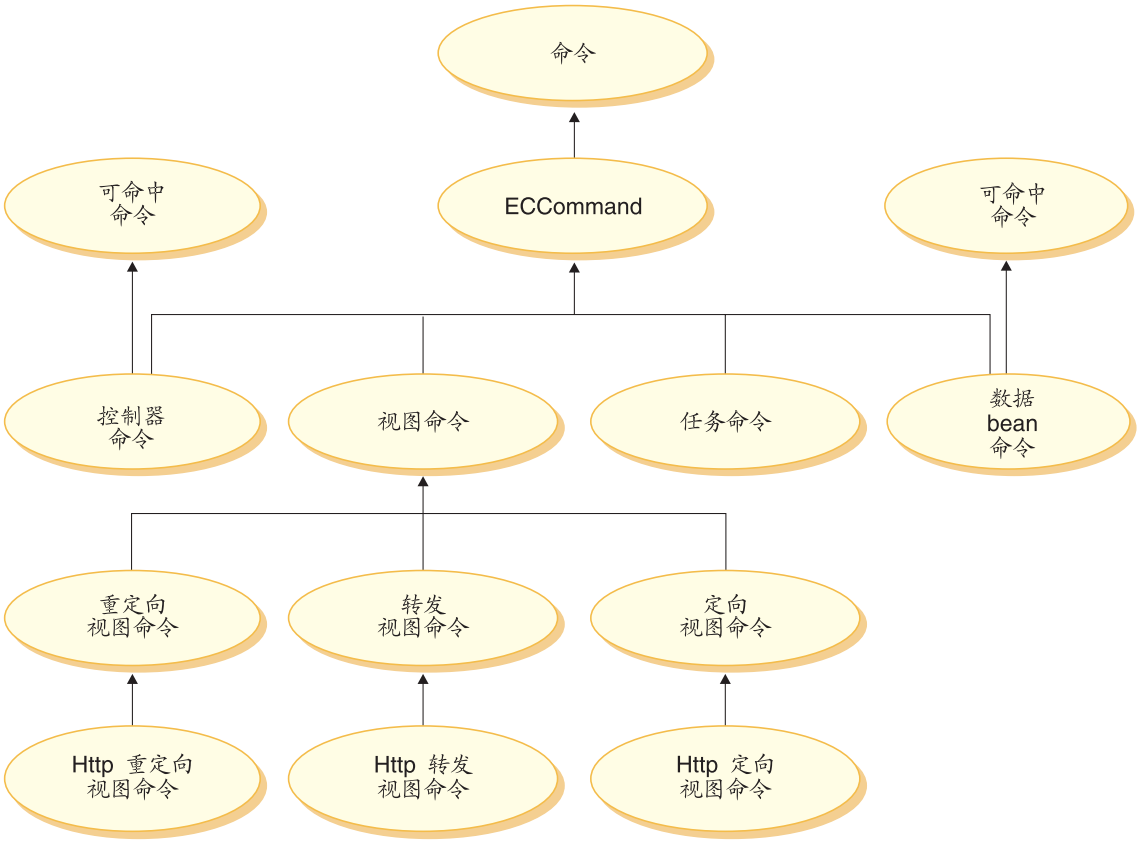

图 7.

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* 命令和允许用户登录的 LogonCmd 命令。通常，控制器命令包含控制语句（例如，if、then 和 else），并调用任务命令来执行业务流程中的个别任务。完成时，控制器命令将返回一个视图名称。然后，基于视图名称、商店标识和设备类型，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。

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 27 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, storeId)
```

其中 *interfaceName* 是新命令 bean 的接口名称，而 *storeId* 是应实现该命令的商店的标识。显然的，商店标识可使用 commandContext.getStoreId() 方法检索。

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 171 页的『调用新业务策略』。

嵌套控制器命令

命令工厂最常用于创建任务命令的实例，然而，还可将命令工厂用于一个控制器命令中，来创建另一个控制器命令的实例。换言之，在某个控制器命令中调用另一个控制器命令时，将会用到它。

实例化任务命令和控制器命令的语法是相同的。即，您在这两个方案中都指定命令接口名称和商店标识。

如果您将一个控制器命令嵌套至另一个中，注意以下几点:

- 一旦您已实例化嵌套命令，调用其 setCommandContext 方法并传入当前命令上下文。注意，如果您将另一组不同的请求属性传递至嵌套命令中且这些参数将影响命令上下文，则您应在实例化嵌套命令前，克隆命令上下文。这将保留命令上下文信息供外部命令使用。
- 理想情况下，调用嵌套命令的 setRequestProperties 方法并传递一个包含输入属性的 TypedProperties 对象。否则，您可以使用命令接口上定义的个别 setter 方法来设置所需的属性。
- 设置了输入属性后，调用嵌套命令的 execute 方法。
- 虽然所有的控制器命令都必须在处理完成时返回一个视图，但是外部命令无法对嵌套命令返回的视图进行任何操作。
- 嵌套命令将在外部命令的事务作用域内执行。

请将以下代码片断视为嵌套控制器命令的示例。该示例显示了外部命令中的一个方法及如何将命令工厂用于实例化第二个控制器命令。

```
yourControllerCmd ctrlCmd = null;

public void processAndCallOtherCommand()
   throws ECException   {
      ctrlCmd = (yourControllerCmd)CommandFactory.createCommand(
         com.yourcompany.commands.yourControllerCmd, this.getStoreId());
      ctrlCmd.setCommandContext(this.getCommandContext());
      ctrlCmd.setRequestProperties(this.getRequestProperties());
      ctrlCmd.execute();
    }
```

作为另一个示例，请考虑下例，其中嵌套命令是对与第一个示例不同的商店执行的。在此例中，必须保存来自外部命令的命令上下文，让它不被内部命令覆盖。

```
// Make a clone to preserve the command context of the outer commmand
CommandContext cloneCmdCtx = (CommandContext)this.getCommandContext().clone();

//Now pass in a new set of request properties to the cloned command context
cloneCmdCtx.setRequestProperties(reqProp);

yourControllerCmd ctrlCmd = null;

public void processAndCallOtherCommandForOtherStore(int aStoreId)
   throws ECException   {
      ctrlCmd = (yourControllerCmd)CommandFactory.createCommand(
         com.yourcompany.commands.yourControllerCmd, aStoreId;
      ctrlCmd.setCommandContext(cloneCmdCtx);
      ctrlCmd.setRequestProperties(reqProp);
      ctrlCmd.execute();
    }
```

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。

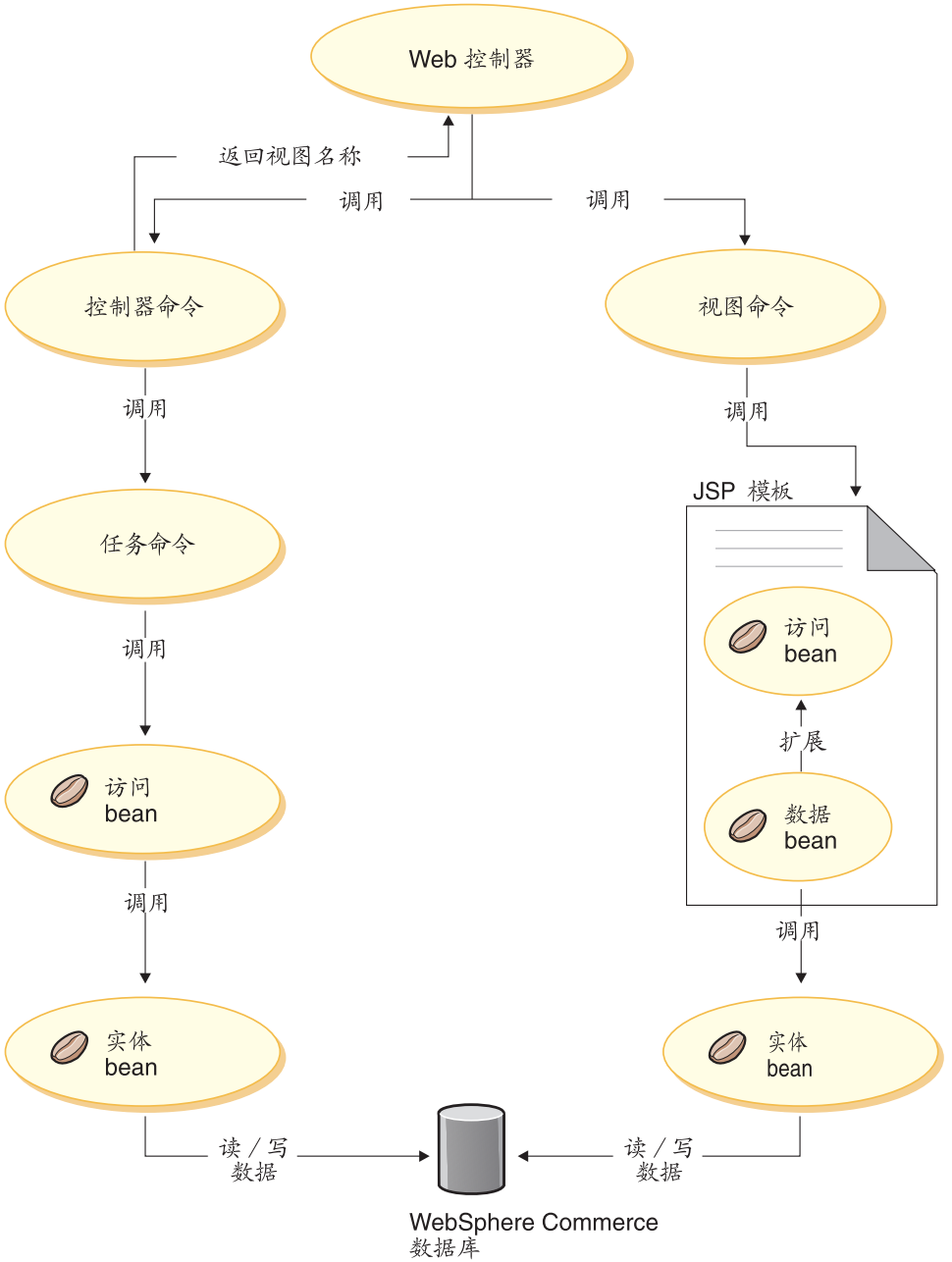

图 8.

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 本部分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 155 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&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 com.ibm.commerce.catalog.commands.ProductDisplayCmd
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.ProductDisplayCmdImpl
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.mycompany.commands.MyNewCommand', 'This is a test command',
'com.mycompany.commands.MyNewCommandImpl', 'myDefaulParm1=myDefaultVal1',
'0000-12-01', 'Local')
```

字符串值应括在单引号内。

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	null	null

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：

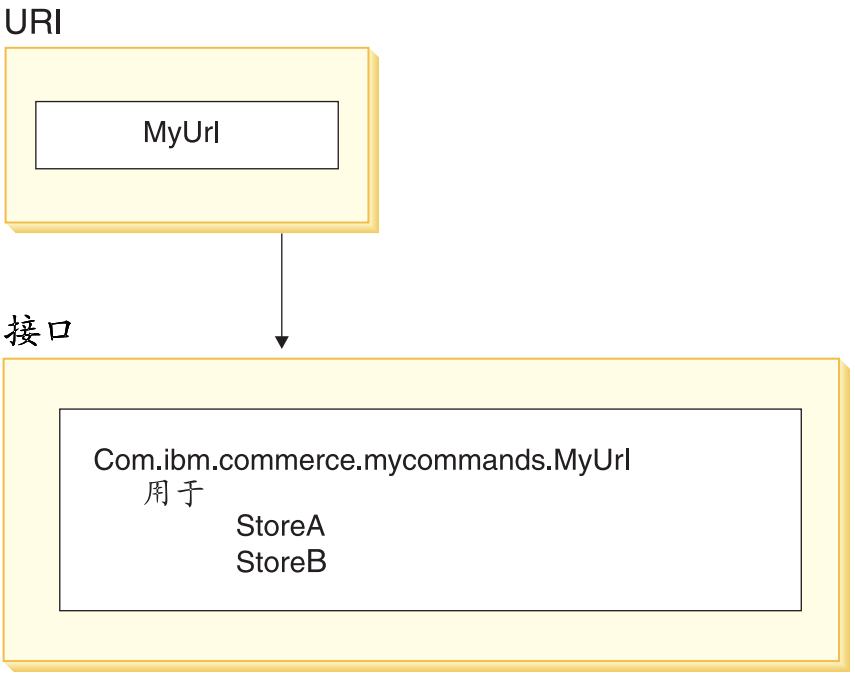

图 9.

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:

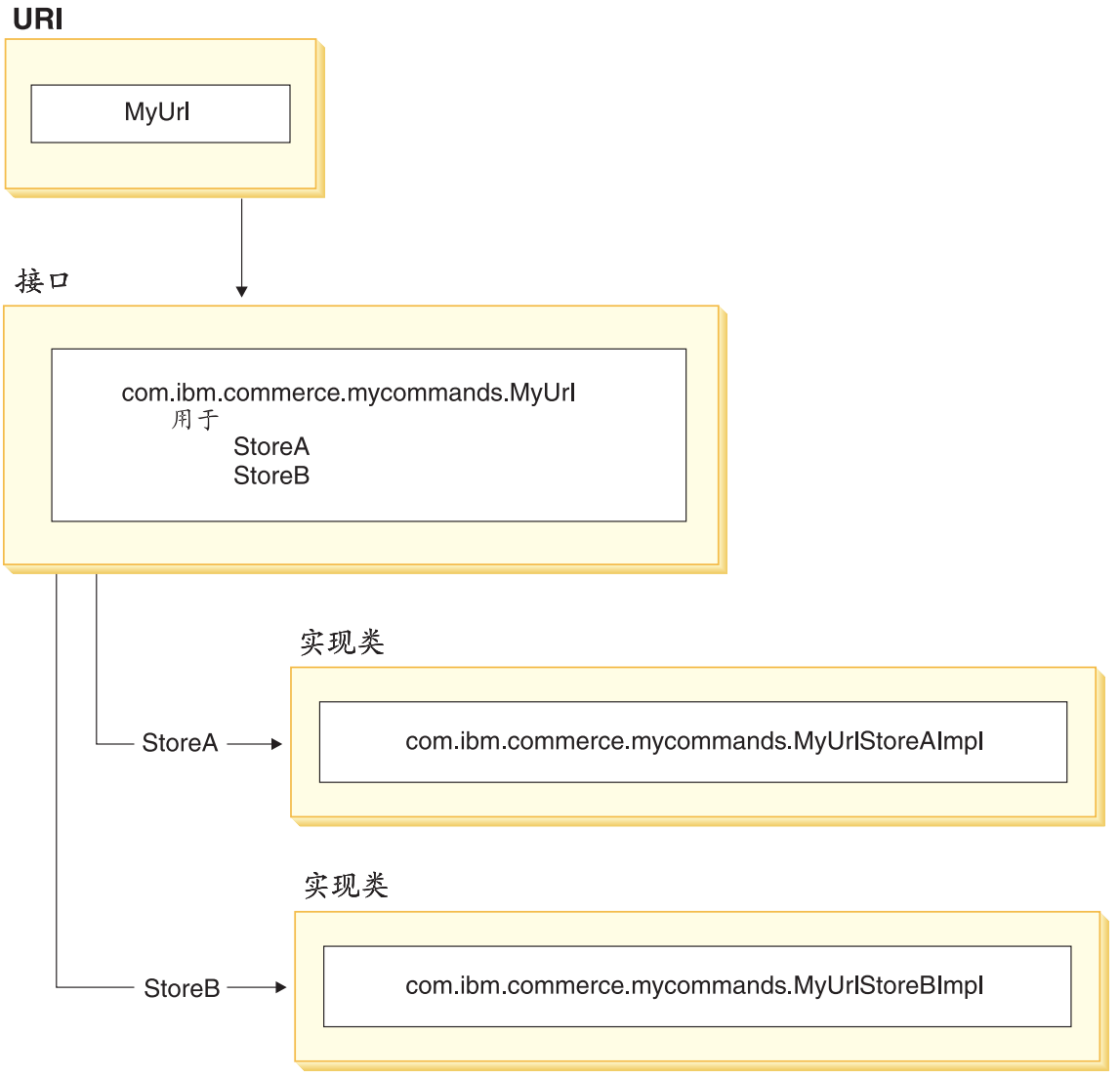

图 10.

VIEWREG 表

VIEWREG 表允许注册特定于设备以及特定于商店的视图实现。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设备的不同视图。

当从控制器命令中返回视图名或在异常中指定视图名时，Web 控制器从 VIEWREG 表中确定视图实现。多个视图名可以映射到同一实现类。

列名	描述	注释
VIEWNAME	视图名称	例如，AddressForm
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:

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

已注册的视图可以使用视图名称在视图注册表中访问，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视图名，也可以通过用实际显示文件名替换视图名来访问视图。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视图，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 *MyNewView* 创建一个条目:

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID,
INTERFACENAME, CLASSNAME, PROPERTIES, DESCRIPTION,HTTPS, LASTUPDATE,
INTERNAL) values ('MyNewView', -1, 5,
'com.ibm.commerce.command.ForwardViewCommand',
```

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

下表提供了另一个样本 VIEWREG 表及其键信息:

视图名称	INTERFACE - NAME	CLASSNAME	PROPERTIES
ProductDisplay View	转发视图命令	HttpForwardView CommandImpl	docname=UserArea/ ServiceSection/ InterestItemList Subsection/ WishListDisplay.jsp
Generic Application Error	转发视图命令	HttpForwardView CommandImpl	docname=Generic Application Error.jsp&storeDir=no
GenericSystem Error	转发视图命令	HttpForwardView CommandImpl	docname=Generic System Error.jsp &storeDir=no
LogonForm	转发视图命令	HttpForwardView CommandImpl	docname= LoginForm.jsp &generic=true &storeDir=no

注: VIEWNAME、INTERFACENAME、CLASSNAME 和 PROPERTIES 值中的空格仅是出于显示目的。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。
- 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 `GenericApplicationError.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 `GenericSystemError.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。所以对所有商店的客户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 servlet。

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 bean 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(`*`data_bean`*`, `*`request`*`)`

其中 *data_bean* 是要激活的数据 bean 而 *request* 是 HTTPServletRequest 对象。

商店开发者在开发 JSP 模板时，应当考虑商店的属性和全球化的问题。有关全球化的更多信息，请参阅*《WebSphere Commerce 商店开发指南》*。

数据 bean 安全性注意事项

使用数据 bean 的特定编码做法使恶意用户以未经授权的方式访问数据库的机会减至最小。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 WebSphere Studio Application Developer 中的工具）。一旦重新生成访问 bean，所有新属性将自动可用于智能数据 bean。
- 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:

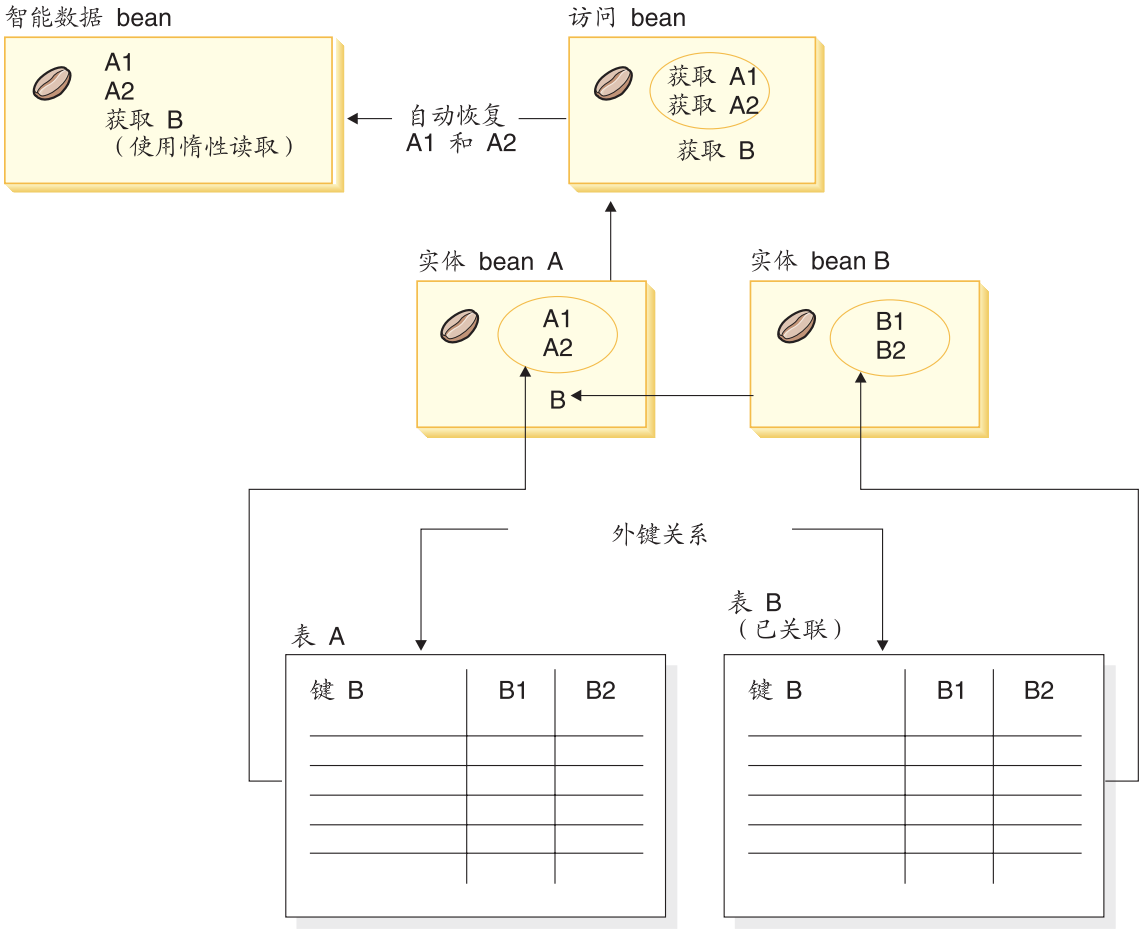

图 11.

关于实现惰性读取检索的更多信息，请参阅第 41 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 servlet 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。应注意，要激活数据 bean 需要以下代码:

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

其中 *data_bean* 是要激活的数据 bean 而 *request* 是 HTTPServletRequest 对象。

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 实现类中的公共运行时方法在 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 模板中调用控制器命令与将逻辑与显示相分离是不一致的，但您可能遇到必须这样做的情况。若为是这种情况，可为此而使用 ControllerCommandInvokerDataBean。

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。

如果在激活此数据 bean 前未使用 setRequestProperties 方法，请求对象的参数将传递至 bean 中，并因此传递至控制器命令中。然而，如果确实在激活此数据 bean 前调用了 setRequestProperties 方法，则该命令只可以使用已指定的属性（传递至 setRequestProperties 方法中的那些属性）和任何在 CMDREG 表的 PROPERTIES 列指定的缺省属性。

一旦执行了控制器命令，您可以执行视图。

您不应重新使用数据 bean 的同一实例来调用其他控制器命令，因为它将包含来自其原始使用的数据和状态信息。

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 编程模型提倡使用 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		属性
com.ibm.xxx. NewCommand		parm1=1&parm2=2

CCPd: parm1=1&parm2=2

VIEWREG

INTERFACENAME		属性
com.ibm.command.ForwardViewCommand		docname=NewView.jsp

VPd: docName=NewView.jsp

URL: http://*hostname*/webapp/wcs/stores/servlet/NewCommand?storeId=1&...

CCPu: storeID=1&...

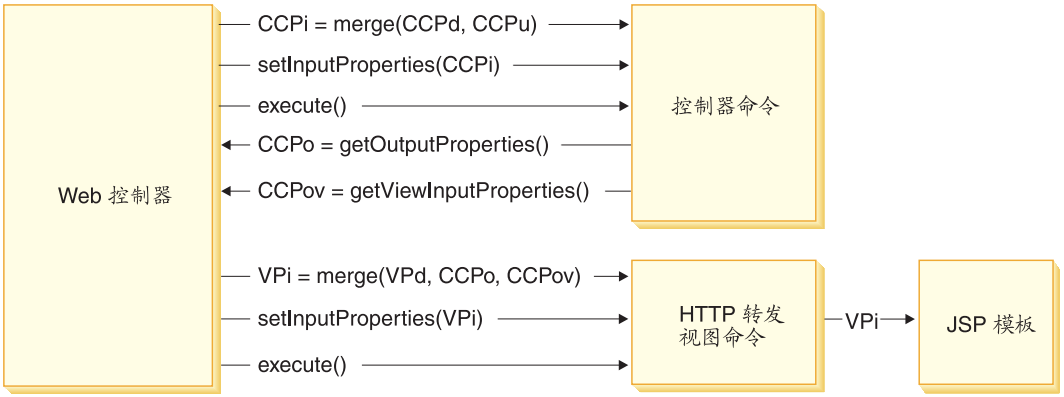

图 12.

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 生产和开发联机帮助中的“参考”主题。

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=SearchResult.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无数的表。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）V1.1 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。

WebSphere Studio Application Developer 提供了支持此框架开发的功能繁多的 EJB 工具和单元测试环境。

以下部分在 WebSphere Commerce 持续对象模型实现（遵守 EJB 1.1 规范）的实现上下文中。

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 的一个优点是开发者可以利用 WebSphere Studio Application Developer 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。

WebSphere Studio Application Developer 对 EJB 1.1 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 WebSphere Studio Application Developer 提供的关

联特征。开发自己的实体 bean 时，建议您不使用 WebSphere Studio Application Developer 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 WebSphere Studio Application Developer 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。您*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 模块:

- Catalog-ProductManagementData
- Enablement-RelationshipManagementData
- Marketing-CampaignsAndScenarioMarketingData
- Marketing-CustomerProfilingAndSegmentationData
- Member-MemberManagementData
- Merchandising-PromotionsAndDiscountsData
- Order-OrderCaptureData
- Order-OrderManagementData
- Trading-AuctionsAndRFQsData

上述列表中的某些 EJB 模块包含会话 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 WebSphereCommerceServerExtensionsData EJB 模块中创建新的会话 bean。关于创建新会话 bean 的更多信息，请参阅第 71 页的『写新会话 bean』。

WebSphere Commerce 企业 bean 的部署描述符

EJB 部署描述符包含企业 bean 的部署设置。WebSphere Studio Application Developer 提供可用于修改此部署信息的 EJB 部署描述符编辑器。

创建新的企业 bean（实体或会话 bean）时，在 WebSphere Studio Application Developer 的 J2EE 视图的 J2EE 层次结构视图中设置部署描述符信息。您可以执行以下操作来查看 WebSphere Commerce bean 的 EJB 部署描述符:

1. 打开 WebSphere Studio Application Developer 并切换至 J2EE 视图。
2. 使用 J2EE 层次结构视图，找到您想要查看其部署描述符信息的 EJB 模块。
3. 用鼠标右键单击 *EJB_moduleName* EJB 模块并选择**打开工具 > 部署描述符编辑器**。
   部署描述符编辑器打开。
4. 选择 Bean 选项卡，并注意以下:

a. 从 bean 列表中，选择 bean。该 bean 的信息填充到其它字段中。

b. bean 应显示为容器管理的实体 1.x bean。

c. “可重入”复选框应未选中。

d. 在“WebSphere 绑定”部分，使用了 JNDI 名称的缺省值。

e. 在“WebSphere 扩展”部分，未对并发控制启用“启用乐观锁定”。

f. 还请注意您可以在“查找程序”部分查看 bean 的查找程序。

5. 选择“装配描述符”选项卡，并注意以下内容:

   a. 在“方法许可权”部分，WCSecurityRole 被分配给企业 bean 中的所有方法。

   b. 在“容器事务”部分，对企业 bean 中的所有方法都指定为“必需的”。

6. 选择“访问”选项卡，并注意以下内容:

   a. 在“对实体 1.x 的访问意向”部分，定义了所有只读方法。例如，_copyFromEJB() 和手工编码的 getter 方法被指定为只读方法。当您创建您自己的实体 bean 时，确保您将相应的方法标注为只读。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。

   b. 如下设置隔离级别:

      - DB2 可重复读
      - Oracle 已提交读

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客户所拥有的房屋类型。您创建称为 USERRES 的表，它包含客户标识和居住类型，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 Member-MemberManagementData EJB 模块中包含一个 “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。当应用程序需要客户居住类型时，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客户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:

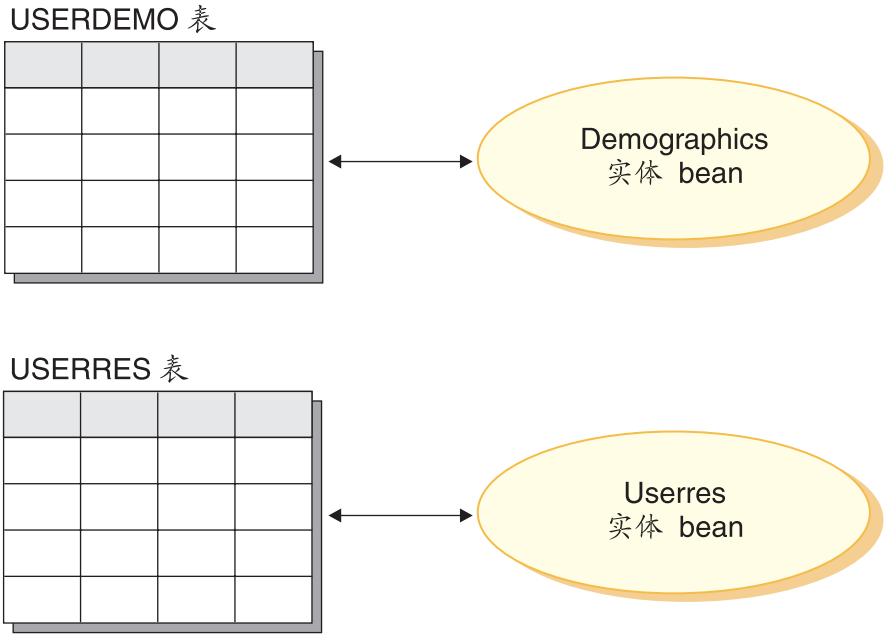

图 13.

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:

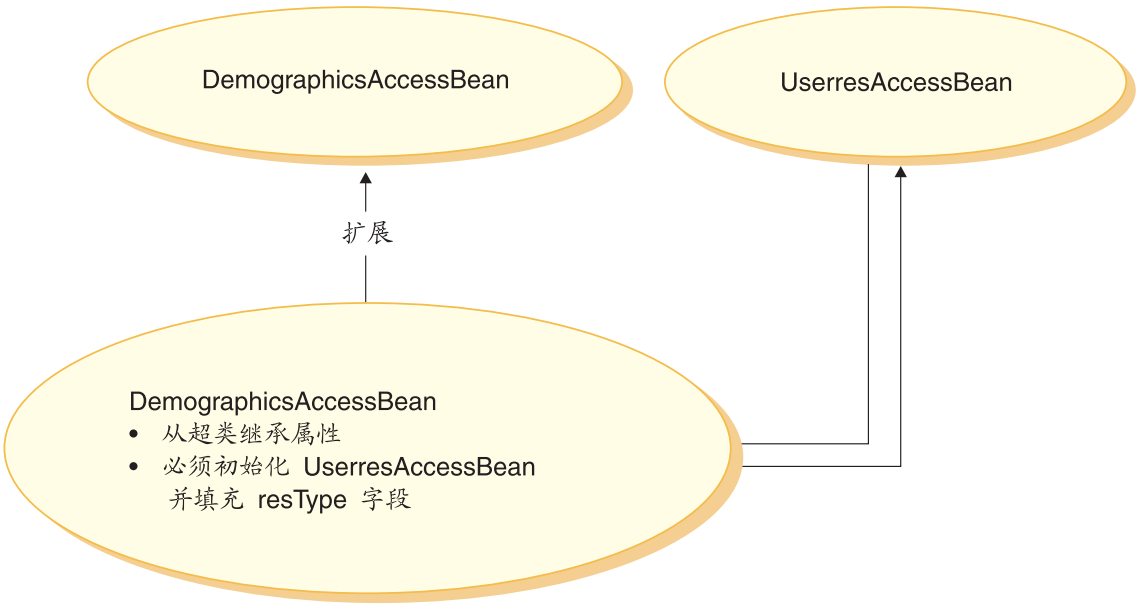

图 14.

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。当应用程序需要客户居住类型时，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客户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:

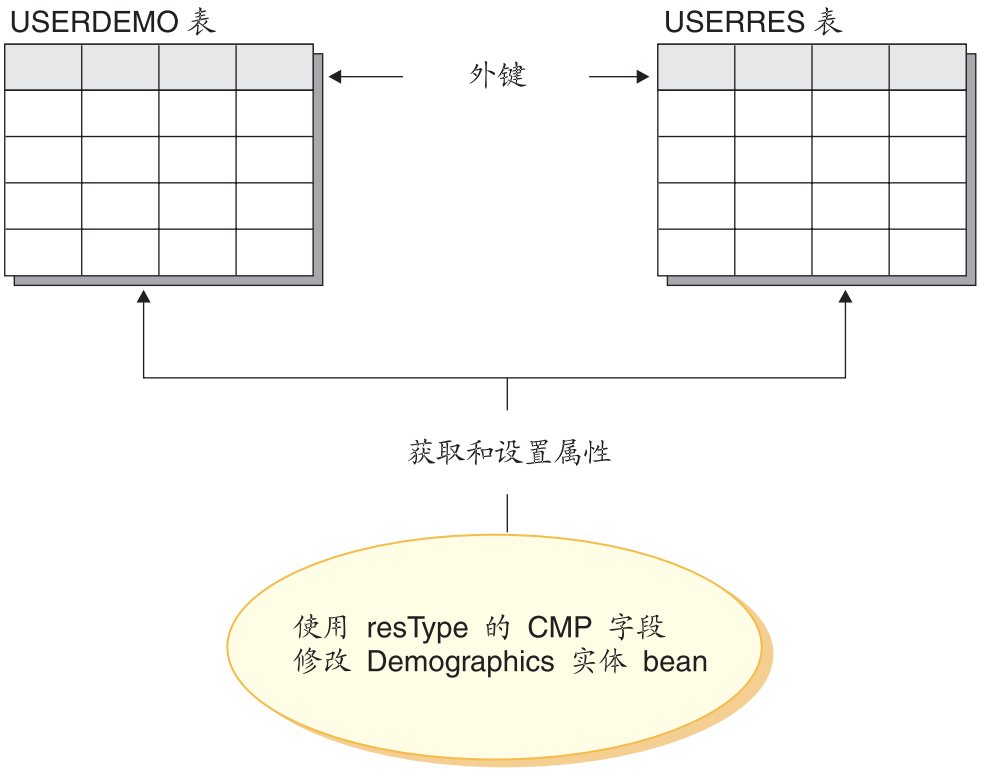

图 15.

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，因此当发布新版本的 WebSphere Commerce 时可能会引起迁移问题。您必须将定制代码与新代码合并，但在导入新 WebSphere Commerce 工作空间时，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 WebSphere Studio Application Developer 中的工具，将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、事务性、位于服务器端的 Java 组件，它们可由 WebSphere Studio Application Developer 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。这称为*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 包装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”。它由来自客户信息表的信息（例如，客户姓名、年龄和客户标识）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。

可以通过执行 SQL join 来构造简单的 SQL 语句，以便从各种表中检索所有信息。这可称为执行“深层读取”。以下是“我的帐户”示例的 SQL select 语句的一个示例，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)
```

WebSphere Studio Application Developer 中使用 EJB 1.1 规范的用于企业 bean 的工具不支持此深层读取概念。但它执行惰性读取，从而为每个关联的对象产生一个 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。

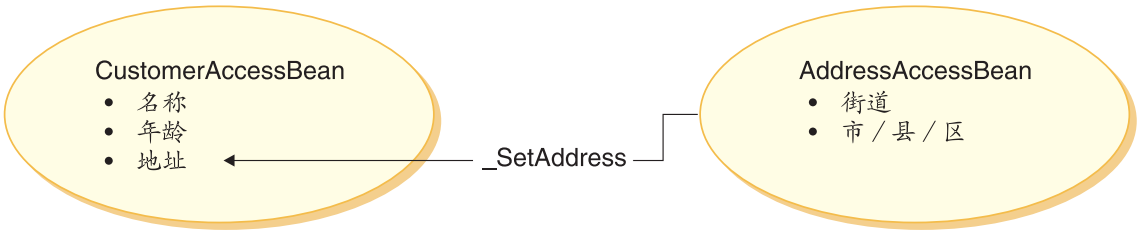

图 16.

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 语句的示例就是当客户尝试浏览整个大型数据集时。例如，如果客户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：可能只在初始时向客户显示商品名称、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。

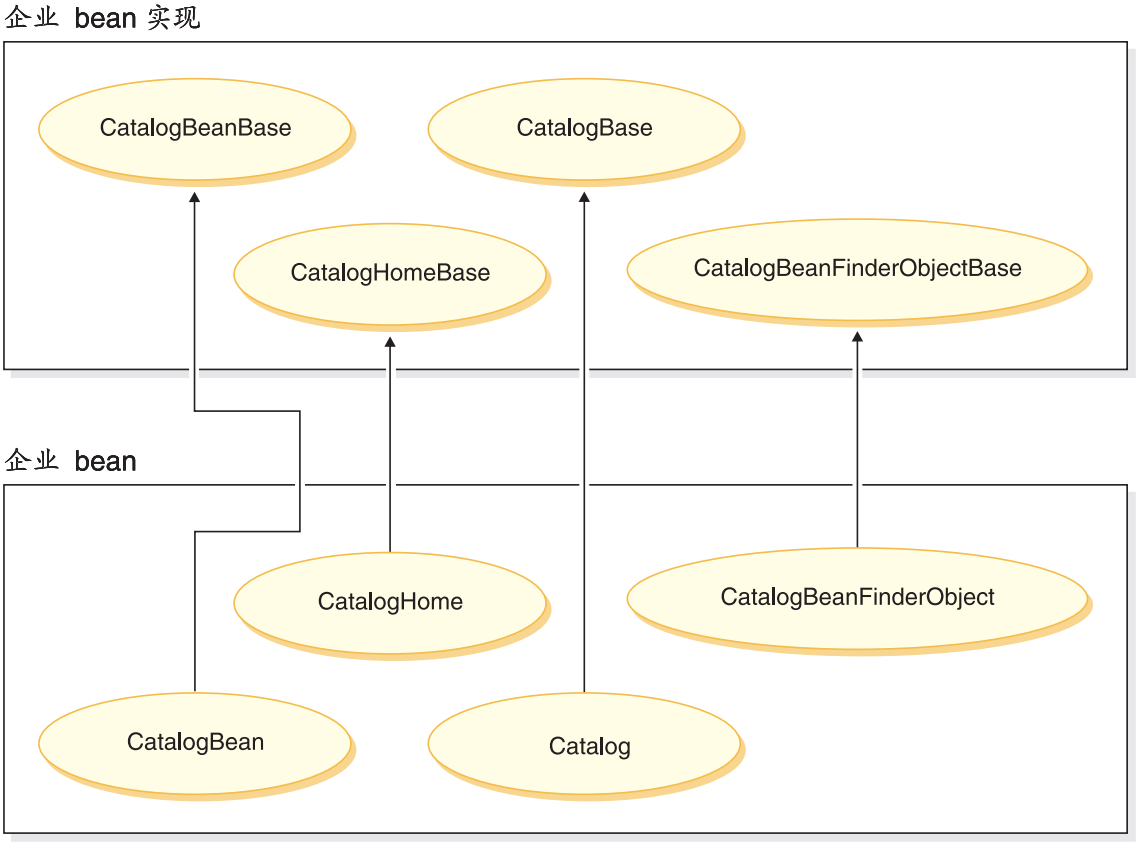

图 17.

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 InterestItemBeanBase 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子接口的不同 Java 数据包中定义了。

WebSphere Commerce 工作空间包含所有这些超类和接口的二进制代码。可以对子类和接口进行修改。一般可以在 com.ibm.commerce.*xxx*.objects 和 com.ibm.commerce.*xxx*.objsrc 数据包（其中 *xxx* 是组件名称）中进行修改。

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。在添加新的查找程序时要使用部署描述符编辑器。

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。

包含“findForUpdate”类型的查找程序的公共实体 bean: 如果 WebSphere Commerce 公共实体 bean 包含任何 “findForUpdate” 类型的查找程序，您不得通过为您已创建的新表创建辅助映射的方式将新的字段添加至 bean 中。这是由于如果您创建辅助映射，生成的 SQL 语句将无效，bean 将不再像预期那样运行。如果您希望扩展由这种 bean 所代表的那部分对象模型，则您必须创建新的实体 bean，并在定制代码中同时使用原始 bean 和新 bean。

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。在此 bean 中创建与新数据库表中属性相对应的字段。

由于 WebSphere Commerce 工作空间为新企业 bean 提供一个预定义的 EJB 项目，您不需要创建附加 EJB 项目。新企业 bean 应当放置在 WebSphereCommerceServerExtensionsData EJB 项目中。此后，当您部署定制 bean 时，您创建 WebSphereCommerceServerExtensionsData.jar JAR 文件并替换运行在 WebSphere Application Server 中的 WebSphere Commerce 企业应用程序的现有 JAR 文件。通过使用此封装约定，可大大简化部署。

要创建新 CMP 企业 bean，必须用 WebSphere Studio Application Developer 执行以下步骤:

1. 使用 “企业 Bean 创建” 向导创建新 CMP 企业 bean。为相应数据库表中的每一列，将一个新的 CMP 字段添加至 bean。
2. 为此新 bean 设置事务隔离级别。
3. 为此新 bean 设置安全性身份。
4. 修改实体上下文方法。

5. 如果需要，使用 EJB 部署描述符编辑器定义新的查找程序。
6. 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。
7. 如果 bean 受 WebSphere Commerce 访问控制系统的保护，则实现 bean 中所必需的访问控制方法。参阅第 83 页的第 4 章，『访问控制』以获得有关在企业 bean 中实现访问控制的更多详细信息。您也可以选择在创建访问 bean 后，实现访问控制。
8. 将企业 bean 中的字段映射至数据库表的列。
9. 为企业 bean 生成相应的访问 bean。
10. 为企业 bean 生成部署代码。
11. 使用 WebSphere Studio Application Developer 中的通用测试客户机来测试 bean。

以下部分中包含了关于上面每个步骤的更多详细信息。阅读这些部分时，假设有一个名为 XUSERRES 的新表，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 CMP 企业 bean: 要创建新 CMP 企业 bean，可以使用“企业 Bean 创建”向导，方法如下:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**。
2. 用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 模块并选择 **新建 > 企业 bean**。
   “企业 bean 创建”向导打开。
3. 从 **EJB 项目**下拉列表，选择 **WebSphereCommerceServerExtensionsData** 并单击**下一步**。
4. 请在“创建企业 bean”窗口执行以下操作:
   a. 选择**带有受容器管理的持久性（CMP）字段的实体 bean**
   b. 在 **Bean 名称**字段中，为您的 bean 输入适当的名称。通常，bean 名称与相应的数据库表的名称相匹配。例如，您将 bean 命名为 XUserRes 来对应 XUSERRES 表。
   c. 在**源文件夹**字段中，保留指定的缺省值（ejbModule）。
   d. 在**缺省数据包**字段，输入 `com.mycompany.mycomponent.objects`。
   e. 单击**下一步**。
5. 请在“企业 bean 详细信息”窗口中执行以下操作:

a. 单击**添加**来为数据库表中的列添加新的 CMP 属性。
“创建 CMP 属性”窗口打开。在此窗口中，请执行以下操作:

1) 在**名称**字段中，为该新的 CMP 字段输入适当的名称。注意如果您希望在稍后将此字段映射至数据库表中其相应的列时使用“按名称匹配”功能，请将字段命名为与列名完全一致（不区分大小写）。
2) 在**类型**字段中，为该字段输入适当的数据类型。注意您应为原语数据类型使用包装类（例如，使用 *java.lang.Long* 数据类型而不是 *long* 数据类型）
3) 如果字段是主键，选择**键字段**复选框并单击**应用**。
4) 如果字段不是主键，选择**以 getter 和 setter 方法访问**复选框。
5) 如果字段不是主键，清除**将 getter 和 setter 方法提升到远程接口**复选框。**使 getter 方法只读**复选框就会不可用，并单击**应用**。
6) 再次单击**添加**，并重复上述步骤为数据库表中每个需要 CMP 字段的列添加新字段。
7) 单击**关闭**以关闭此窗口。

b. 清除**对键类使用单一键属性类型**复选框，然后单击**下一步**。

6. 在 EJB Java 类详细信息窗口，请执行以下操作:

a. 要选择该 bean 的超类，请单击**浏览**。
“类型选择”窗口打开。

b. 在**使用以下任意方法选择一个类**字段中，输入 `ECEntityBean` 并单击**确定**。
这选择了 com.ibm.commerce.base.objects.ECEntityBean 为超类。

c. 如果新企业 bean 要受 WebSphere Commerce 访问控制框架的保护，单击**添加**来指定远程接口应该扩展的接口。“类型选择”窗口打开。

d. 在**使用以下任意方法选择一个类**字段中，输入 `Protectable` 并单击**确定**。
这选择了 com.ibm.commerce.security.Protectable。为了在访问控制下保护新资源，该接口是必需的。

e. 单击**完成**。

在 bean 类中，WebSphere Studio Application Developer 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生成的字段。这样，您必须从新实体 bean 中除去生成的 `EntityContext`。这会在后面的部分中进行描述。

指定 `com.ibm.commerce.base.objects.ECEntityBean` 作为超类时，bean 会继承特定功能。以下代码示例阐述了这些功能:

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

设置事务隔离级别: 您必须将 bean 的事务隔离级别设置为用于开发数据库类型的正确值。要设置事务隔离级别，请执行以下操作:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**。
2. 双击 **WebSphereCommerceServerExtensionsData** 项目来使用部署描述符编辑器打开它。
3. 单击**访问**选项卡。
4. 单击“隔离级别”文本框旁的**添加**。
   “添加隔离级别”窗口打开。
5. DB2 选择**可重复读**，然后单击**下一步**。
   Oracle 选择**已提交的读**，然后单击**下一步**
6. 从**已找到的 bean** 列表中，选择 *yourNewBean* bean，然后单击**下一步**。
7. 从**已找到的方法**列表中，选择 *yourNewBean* 来选中其所有方法，然后单击**完成**。
8. 保存工作（Ctrl+S），然后保持编辑器打开。

为 bean 设置安全性身份: 然后通过执行以下操作设置该 bean 的安全性身份:

1. 在部署描述符编辑器中，确保已选择“访问”选项卡。
2. 单击“安全性身份”文本框旁的**添加**。
   “添加安全性身份”窗口打开。
3. 选择**使用 EJB 服务器的身份**，然后单击**下一步**。
4. 从**已找到的 bean** 列表中，选择 *yourNewBean* bean，然后单击**下一步**。
5. 从**已找到的方法**列表中，选择 *yourNewBean* 来选中其所有方法，然后单击**完成**。
6. 保存工作（Ctrl+S）。保持编辑器打开。

为 bean 设置安全性角色: 然后通过执行以下操作为 bean 中的方法设置安全性角色:

1. 在部署描述符编辑器中，选择“Assembly 描述符”选项卡。

2. 在“方法许可权”部分，单击**添加**。
3. 选择 **WCSecurityRole** 作为安全角色并单击**下一步**。
4. 从找到的 bean 列表中，选择 *yourNewBean* 然后单击**下一步**。
5. 在“方法”元素页面，单击**应用到所有**，然后单击**完成**。
6. 保存工作（Ctrl+S）然后关闭部署描述符编辑器。

删除实体上下文字段和方法: 下一步是除去一些与 WebSphere Studio Application Developer 生成的实体上下文相关的字段和方法。需要删除这些字段的原因是 ECEntityBean 基类对于这些方法会提供其自己的实现。要删除已生成的实体上下文字段和方法，请执行以下操作:

1. 在“J2EE 层次结构”视图中，展开 **WebSphereCommerceServerExtensionsData** 项目。
2. 展开**实体 bean**、*yourNewBean* bean，然后双击 *yourNewBean***Bean** 类。
3. 在“概览”视图中，请执行以下操作:

   注: Express WebSphere Studio Application Developer 5.1 会提示您删除 getEntityContext() 和 setEntityContext(EntityContext)，这样您就不必像下面所提到的那样手工删除它们。确保选择删除这些项。

   a. 用鼠标右键单击 **myEntityCtx** 字段并选择**删除**。
   b. 用鼠标右键单击 **getEntityContext()** 方法，并选择**删除**。
   c. 用鼠标右键单击 **setEntityContext(EntityContext)** 方法，并选择**删除**。
   d. 用鼠标右键单击 **unsetEntityContext()** 方法并选择**删除**。
4. 保存工作（Ctrl+S）。

添加新的查找程序:

查找程序使用介绍: 不赞成使用查找程序帮手界面。对任何新开发工作，要求您使用 EJB 部署描述符编辑器而不是查找程序帮手界面来定义查询和方法声明。

有关使用 EJB 查询语言创建查找程序的详细信息、对于查找程序此方法与其它方法相比的优点以及关于相关工具的更多详细信息，请参阅 WebSphere Studio Application Developer 联机帮助。

将新的查找程序添加至新 bean 中: 如果您需要将新的查找程序添加至企业 bean 中，请执行以下操作:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**。
2. 双击 **WebSphereCommerceServerExtensionsData** 项目以打开 EJB 部署描述符编辑器。

3. 单击 **bean** 选项卡。
4. 在 bean 窗格中，选择 *yourNewBean* bean，然后在右窗格中向下滚动并展开 **WebSphere 扩展**。
5. 单击**查找程序**文本框旁的**添加**。
   “添加查找程序描述符”窗口打开。
6. 选择**新建**，然后在**名称**字段输入 `findByX`*yourArg*（其中 *yourArg* 是您搜索的参数的名称）。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。
7. 单击**参数**文本框旁的**添加**，然后执行以下操作:
   a. 在**名称**字段，输入 *yourArg*。
   b. 在**类型**字段，输入适当的数据类型。
   c. 单击**确定**。
8. 在**返回类型**字段，输入以下内容之一，并单击**下一步**:
   - 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`。
9. 从**查找程序类型**下拉列表中选择 **WhereClauseFinderDescriptor**。
10. 在**查找程序语句**中，输入合适的查找程序。例如，输入 `T1.MEMBERID = ?`，然后单击**完成**。
11. 保存工作，然后关闭 EJB 部署描述符编辑器。

出于安全性原因，当为新实体 bean 创建 FinderHelper 方法时，应当使用上述步骤中显示的参数插入。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:

```
T1.MEMBERID = "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。

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 null 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 null 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 “J2EE 层次结构” 视图中，双击 ***yourNewBean*Bean** 类来打开它并查看其源代码。
2. 您必须修改源代码，使每个不可空 CMP 字段都作为方法的输入参数包含进来，并且每个 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 {
    _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 {
    _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 75 页的『主键』获取详细信息。
3. 必须将此新的 ejbCreate 方法添加到人机接口。这将使方法在生成的访问 bean 中可用。要向主接口添加方法，请执行以下操作:
   a. 用鼠标右键单击 “概览” 视图中的 **ejbCreate(*yourParameters*)** 方法并选择 **企业 bean > 提升至人机接口**。

创建新的 ejbPostCreate 方法: 下一步，您必须创建和新的 ejbCreate 方法有相同输入参数的新的 ejbPostCreate 方法。要创建此新方法，请执行以下操作:

1. 双击 ***yourNewBean*Bean** 类来打开它并查看其源代码。
2. 使用新 ejbCreate 方法中所用的相同输入参数创建新 ejbPostCreate 方法。要继续使用此用户居住示例，您将以下代码引入类中:

```
public void ejbPostCreate(int argUserId,
   String argHome, byte Rooms)
   {
   }
```

保存代码更改。

向 bean 添加访问控制方法: 如果新 bean 要受访问控制的保护，则必须添加 getOwner 方法。另一用于访问控制的可选方法是 fulfills 方法。关于必需的和可选的方法的详细信息，请参阅第 99 页的『在企业 bean 中实现访问控制』。要将访问控制方法添加至新的 bean，请执行以下操作:

1. 在“J2EE 层次结构”视图中，双击 ***yourNewBean*Bean** 类来打开它并查看其源代码。
2. 在源代码中添加 getOwner 方法，此方法中包含返回此资源的所有者的逻辑。例如，要返回 UserRes bean 的所有者，则返回用户的成员标识:

   ```
   public java.lang.Long getOwner()
      throws java.lang.Exception {
         return getMemberId();
      }
   ```

3. 出于此示例的目的，添加 fulfills 方法，该方法指定允许用户对此资源进行操作前，用户必须满足什么关系。在此例中，指定仅允许此 UserRes 对象的创建者进行操作。换言之，仅允许每个用户对其自己的 UserRes 对象进行操作。此关系要求显示在以下代码片断中:

   ```
   public boolean fulfills(Long member, String relationship)
      throws java.lang.Exception {
         if (relationship.equalsIgnoreCase("creator")) {
            return member.equals(getMemberId());
         }
         return false;
      }
   ```

4. 保存工作。

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。当企业 bean 和其相应的数据库表都存在时，将使用“在中间会面”类型的映射。WebSphere Studio Application Developer 提供了简化此任务的工具。

要创建映射,请执行以下操作:

1. 在“J2EE 层次结构”视图中，用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 并选择**生成 > EJB 到 RDB 映射**。

   “EJB 到 RDB 映射”窗口打开。
2. 选择**在中间会面**并单击**下一步**。

3. 在“数据库连接”窗口中，请执行以下操作:
   a. 在**连接名称**字段，输入 `WebSphereCommerceServerExtensionsData`
   b. 在**数据库字段**，输入 *developmentDB*
   c. 在**用户标识**字段，输入 *dbuser*
   d. 在**密码**字段，输入 *dbpassword*
   e. 从数据库下拉列表，选择开发数据库的数据库供应商类型。
      - DB2 DB2 通用数据库 8.1
      - Oracle Oracle 9i
   f. Oracle 在**主机**字段，输入数据库服务器的全限定主机名。例如，输入 `dbserver.yourcompany.com`
   g. Oracle 在“类位置”字段，输入 classes12.zip 文件的位置。例如，输入 `D:\oracle\ora92\jdbc\lib\classes12.zip`
   h. 单击**下一步**。一旦连接已建立，就会显示数据库中的表的列表。您也可以稍后通过查看“数据”视图中的“数据库服务器”视图来查看连接文档。
4. 选择 ***yourNewTable*** 表并单击**下一步**。
5. 选择**按名称和类型匹配**然后单击**完成**。映射编辑器已打开。
6. Oracle 如果任何列具有“NUMBER”数据类型，则必须修改该数据类型。用鼠标右键单击 ***yourNewTable*** 表并选择**打开表编辑器**。在表编辑器中，请执行以下操作:
   a. 选择**列**选项卡。
   b. 选择需要更改其数据类型的列，并将列类型从 NUMBER 更改为更具体的类型。例如，将它更改为 INTEGER。
   c. 保存更改。
7. 在“企业 bean”窗格中，展开 ***yourNewBean*** bean。在“表”窗格，展开 ***yourNewTable*** 表。
8. 执行以下操作将 *yourNewBean* bean 中的字段映射为 *yourNewTable* 表的列:
   a. 用鼠标右键单击 ***yourNewBean*** bean 并选择**按名称匹配**。
9. 保存对 Map.mapxmi 文件所作的更改，并关闭此文件。
10. Oracle 您必须使用文本编辑器编辑表定义，方法如下:
    a. 使用文本编辑器打开 *yourNewBean*.xmi 文件。
    b. 将出现的所有 `SQLNumeric6` 替换为 `SQLNumeric3`。
    c. 保存更改，然后关闭文件。

修改模式名称: 下一步是修改模式名以使 bean 可移植到其它数据库。允许 bean 可以此方式移植的特殊值是 NULLID。要修改模式名，请执行以下操作:

1. 在 J2EE 视图中，切换至“J2EE 层次结构”视图。
2. 展开**数据库**，然后**展开 WebSphereCommerceServerExtensionsData**。
3. 用鼠标右键单击模式节点（例如，db2user）然后选择**重命名**。
4. 把值设置为 NULLID。

创建访问 bean: 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。WebSphere Studio Application Developer 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）。

要创建访问 bean，请执行以下操作:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**，然后用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 并选择**新建 > 访问 bean**。
   “添加访问 bean”窗口打开。
2. 选择**复制帮手**并单击**下一步**。
3. 选择 ***yourNewBean*** bean 并单击**下一步**。
4. 从构造函数方法下拉列表中，选择 **findByPrimaryKey(***yourPackageName***.***yourNewBean***Key)** 作为构造函数方法。
5. 选择“属性帮手”部分的所有属性。
6. 单击**完成**。
7. 保存工作。

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。它还生成通过 IIOP 进行的 RMI 访问所必需的 Java ORB、存根和 tie 类以及主接口和远程接口的存根

要生成部署代码，请执行以下操作:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**，然后用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 并选择**生成 > 部署和 RMIC 代码**。
   “部署和 RMIC 代码”窗口打开。
2. 选择 ***yourNewBean*** bean 并单击**完成**。

您可以通过切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图来查看新生成的代码。您会看到如下内容:

表 1.

代码类型	类名
容器实现生成代码	EJSCMP*yourNewBean*HomeBean.java
	EJSRemoteCMP*yourNewBean*.java
	EJSRemoteCMP*yourNewBean*Home.java
	EJSFinder*yourNewBean*Bean.java
JDBC 访问代码	EJSJDBCPersisterCMP*yourNewBean*Bean.java
RMI 约束和存根代码	_EJSRemoteCMP*yourNewBean*_Tie.java
	*_yourNewBean*_Stub.java
	_EJSRemoteCMP*yourNewBean*Home_Tie.java
	*_yourNewBean*Home_Stub.java

使用测试客户机测试企业 bean: WebSphere Studio Application Developer 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. 切换至“服务器”视图。
2. 双击 **WebSphereCommerceServer** 服务器并单击**配置**选项卡。
3. 选择**启用通用测试客户机**。保存更改。
4. 用鼠标右键单击 **WebSphereCommerceServer** 服务器并选择**启动**。
5. 用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 并选择**在服务器上运行**。
   Web 浏览器随通用测试客户机一起打开。测试企业 bean 的起点是 JNDI 资源管理器, 在其中可以找到运行在此服务器上的 EJB 容器中的 bean 的名称。
6. 用鼠标右键单击 **JNDI 资源管理器**
7. 接着, 必须浏览至 *yourNewBean*Home 接口, 方法是展开如下层次结构: **单元 > 节点 > 本地主机 > 服务器 > server1 > ejb >** *yourPackageStructure*。
8. 单击 ***yourNewBean*Home** 接口。在“引用”窗格, 选择 **EJB 引用 > *yourNewBean* > *yourNewBean*Home**。单击 create 方法。
9. 在右侧窗格与 create 方法的必需输入参数对应的字段中, 输入适当的值。
10. 单击**调用**, 结果就会显示在底端窗格内。
11. 单击**处理对象**来向“引用”窗格添加远程接口, 然后您就会在“对象引用”下看到输入的值。已在新表中创建新记录。

12. 相应地测试其它方法。
13. 关闭测试客户机并停止服务器。

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。
- 企业 bean 代码不应引用企业 bean 模块外部的任何项。例如，在企业 bean 代码中不应引用命令或数据 bean
- 前面各部分描述了在初始创建新的 bean 时如何在该 bean 中包含访问控制。它可在您创建 bean 之后通过添加 com.ibm.commerce.security.Protectable 接口来添加。如果需要，还可将 com.ibm.commerce.security.Groupable 接口添加到企业 bean 的远程接口。同时必须在 bean 中实现某些方法。添加这些接口并添加必需的方法后，重新生成 bean 的部署代码和访问 bean。关于更多信息，请参阅第 99 页的『在企业 bean 中实现访问控制』。

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 WebSphere Studio Application Developer 自动生成。

新数据 bean 存储在 WebSphereCommerceServerExtensionsLogic 项目中。

要创建简单数据 bean，必须执行以下步骤:

1. 创建数据包来存储数据 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。

在后面的部分中描述了各步骤的详细信息。

为数据 bean 代码创建数据包: 创建数据包将会创建一个可存储您的数据 bean 代码的地方。

要创建新数据包，请执行以下操作:

1. 打开 WebSphere Studio Application Developer 并切换至 Java 视图。
2. 用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 项目，选择 **新建 > 数据包**。“新建 Java 数据包”向导打开。
3. **源文件夹**的值已用 `WebSphereCommerceServerExtensionsLogic/src` 填充。保留该值。

4. 在**名称**字段，为新数据包输入适当的名称。例如，输入 com.mycompany.mydatabeans。
5. 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**新建 > 类**。“新建 Java 类”向导打开。
2. 项目和数据包名称字段已经填充。
3. 在**名称**字段，为新数据 bean 输入名称。例如，要创建扩展 UserResAccessBean 的数据 bean，请输入 UserResDataBean。
4. 在**修改量**列表中，选择**公共**。
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. 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**确定**。
7. 单击**完成**。

添加必需字段至数据 bean: 本节描述如何修改新数据 bean 中必需的字段。两个必需字段用于以下类型的信息:

- 命令上下文
- 请求属性

要修改 iCommandContext 字段，请执行以下操作:

1. 双击新数据 bean（例如 UserResDataBean）以查看其源代码。
2. 找到 getCommandContext 方法。它显示如下:

```
public CommandContext getCommandContext() {
return null;
```

将以下内容添加至源代码中:

```
private CommandContext iCommandContext = null;
 public com.ibm.commerce.command.CommandContext getCommandContext()
{
 return iCommandContext;
}
```

3. 修改 *set*CommandContext 方法。它最初显示如下:

```
public void setCommandContext(CommandContext arg0) {
 }
```

修改代码使它显示如下:

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

4. 修改 *get*CommandContext 方法。它最初显示如下

```
public CommandContext getCommandContext() {
      return null;
 }
```

修改代码使它显示如下:

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

5. 保存工作。

要修改 iRequestProperties 字段，请执行以下操作:

1. 双击新数据 bean（例如 UserResDataBean）以查看其源代码。
2. 找到 *get*RequestProperties 方法。它最初显示如下:

```
public TypedProperty getRequestProperties() {
      return null;
 }
```

修改源代码，使其如下显示:

```
private com.ibm.commerce.datatype.TypedProperty
   requestProperties;

public TypedProperty getRequestProperties() {
 return requestProperties;
}
```

3. 修改 *set*RequestProperties 方法。它最初显示如下:

```
public void setRequestProperties(TypedProperty arg0) throws Exception {
 }
```

修改源代码，使其如下显示:

```
public void setRequestProperties(com.ibm.commerce.datatype.TypedProperty
aParam)
   throws Exception {
// copy input TypedProperteis to local

  requestProperties = aParam;
}
```

4. 修改 getRequestProperties 方法。它最初显示如下:

```
public TypedProperty getRequestProperties() {
      return null;
 }
```

修改源代码，使其如下显示:

```
public TypedProperty getRequestProperties() {
 return requestProperties;
}
```

5. 保存工作。

为相应的访问 bean 填充主键: 注意: 您可能希望修改源代码来填充相应访问 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。请注意，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，用于设置主键的推荐过程是间接方法。

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. 在“概览”视图中，选择 **populate()** 方法来查看其源代码。
   它最初显示如下:

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

3. 修改源代码，使方法如下显示:

   ```
   try {
      super.refreshCopyHelper();
        } catch (javax.ejb.FinderException e) {
        throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
        "UserResDataBean", "populate");
   ```

   保存工作（Ctrl+S）。

注意，如果新数据 bean 正在扩展实例化时要求附加输入参数的访问 bean，您还必须在数据 bean 的 populate 方法中设置那些值。

写新会话 bean

创建新的会话 bean 时，在 WebSphereCommerceServerExtensionsData 项目中创建它们。

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 WebSphere Commerce Server 使用的数据源获取 JDBC 连接对象的方法，从而会话 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,
      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 V1.1 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 V1.1 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，USERS 表中的 REGISTRATIONUPDATE 不会更新为当前时间戳记。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，则必须将 `WCSecurityRole` 角色指定给任何新的企业 bean 的方法。可使用 WebSphere Application Server 应用程序组装工具来执行此任务。部署新企业 bean 时执行此步骤。另外，如果修改了现有的 WebSphere Commerce 实体 bean，您必须将 `WCSecurityRole` 角色指定给修改后的 EJB 项目的每个实体 bean 的每个方法。

关于定制代码部署过程的描述，请参阅第 187 页的第 9 章，『部署详细信息』。

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，为特定用户概要文件）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 {
 //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(38, 0)），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 WebSphere Studio Application Developer 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）更新客户名称，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 WebSphere Studio Application Developer 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。

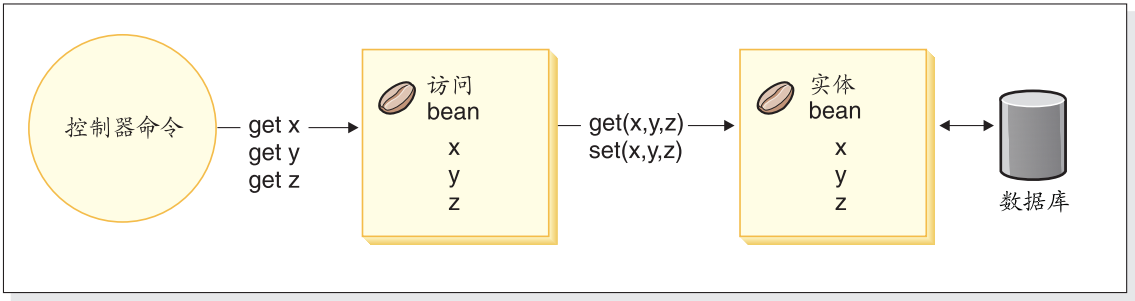

图 18.

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

通常，创建遵守上述表名约定的新表时，可对那些表中的列实施您自己的命名约定。它假定您在 SQL 语句中始终使用全限定列名。但是，如果您希望与现有 WebSphere Commerce 表合并，且不希望使用全限定列名，则必须遵循此部分中描述的列命名约定。

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 81 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	AIX Linux Solaris Windows **DB2**	AIX Solaris Windows **Oracle**	400 **DB2**
Hashtable	BLOB()	BLOB	BLOB()
Timestamp	TIMESTAMP	DATE	TIMESTAMP
Integer	INTEGER	INTEGER	INTEGER
BigDecimal	DECIMAL(,)	DECIMAL(,)	DECIMAL(,)
Long	BIGINT	NUMBER(38,0)	BIGINT
Double	FLOAT	NUMBER(38,0)	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）

JDBC 对象	AIX Linux Solaris Windows **DB2**	AIX Solaris Windows **Oracle**	400 **DB2**
String	CLOB()	CLOB()	DBCLOB() CCSID 13488

注:

1. 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。
2. 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。代表人员的用户概要文件也是资源。“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 servlet。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/*
  这用于针对请求 servlet 提出的请求。

- /webapp/wcs/stores/*.jsp
  这用于针对 JSP servlet 提出的请求。

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。

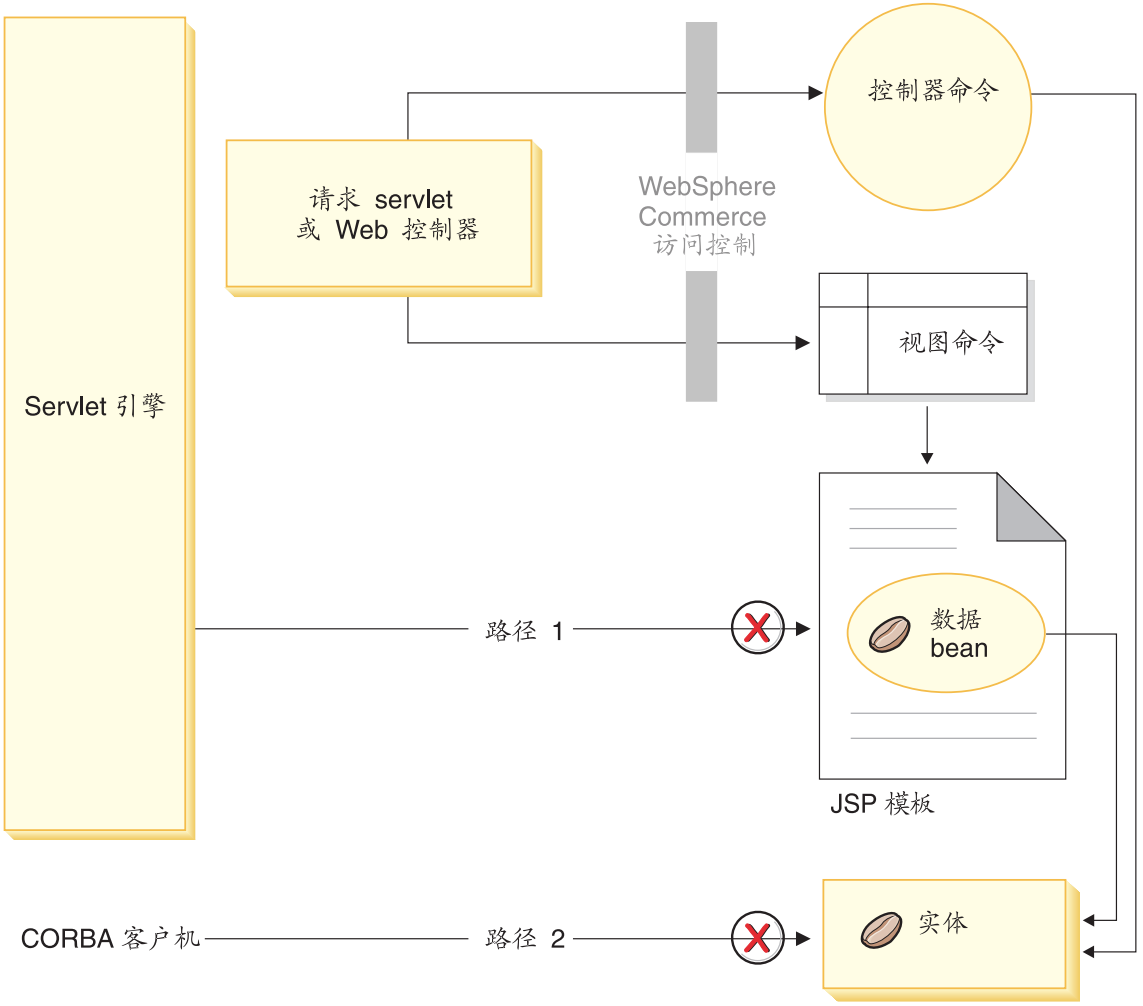

图 19.

所有合法请求应当定向到请求 servlet，然后该 servlet 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 提供了许多安全性功能。WebSphere Commerce 使用其中一个功能来确保所有企业 bean 方法和 JSP 模板都配置为仅由选定身份调用。要访问这些 WebSphere Commerce 资源，必须将 URL 请求路由到请求 servlet。请求 servlet 在把该 URL 请求传递给 Web 控制器之前会在当前线程上设置选定身份。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 安全性指南*》。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。

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。

URL 参数的安全性注意事项

URL 参数提供了传递特定于请求的信息的有用方式。为将恶意用户通过未授权的方式获得对您的数据库的访问权的可能减到最小，必须遵循特定的编码做法。SQL 语句的 insert、select、update 和 delete 部分应在开发时创建。应使用参数插入来收集运行时输入信息。

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"
```

避免使用输入字符串来撰写 select 的原因在于输入字符串易受操纵。恶意用户可以将此类输入字符串设置为某个意外的值并可能由此获得未授权的访问或对数据库执行不希望的操作。

WebSphere Commerce 访问控制策略简介

本节提供了对 WebSphere Commerce 访问控制框架的主要组件的极为简要的介绍。提供介绍的目的在于可将一些与访问控制相关的编程任务放在正确的上下文中考察。如果您需要有关在生产系统上设置访问控制的信息，或更多有关访问控制框架的详细信息，请参阅*《WebSphere Commerce 安全性指南》*。

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的操作。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 生产和开发联机帮助。

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 Conditions 列的 XML 信息的详细信息可在 *《WebSphere Commerce 安全性指南》* 中找到。

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.ContractUpdateCmd
10248	com.ibm.commerce.contract.commands.ContractDeleteCmd
10249	com.ibm.commerce.contract.commands.ContractCancelCmd
10250	com.ibm.commerce.contract.commands.ContractCloseCmd

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组的概念。WebSphere Commerce V5.5 支持各种商务模型，每个商务模型都有其自己的一套访问控制策略。为了对模型中的各套策略进行分组，需要使用策略组。将策略显式地分配至相应的策略组中，随后组织可以预订这些策略组中的一个或多个。在 WebSphere Commerce 的先前版本中，一个策略适用于该策略所有者组织的下级组织所拥有的全部资源。

在 WebSphere Commerce V5.5 中，如果组织预订了一个或多个策略组，则只有这些策略组中的策略才适用于该组织的资源。如果资源为一个未预订任何策略组的组织所有，访问控制策略管理器将向上搜索组织层次结构，直至遇到最接近的至少预订了一个策略组的上级组织；一旦找到，则它将应用属于这些策略组的策略。

请考虑下图:

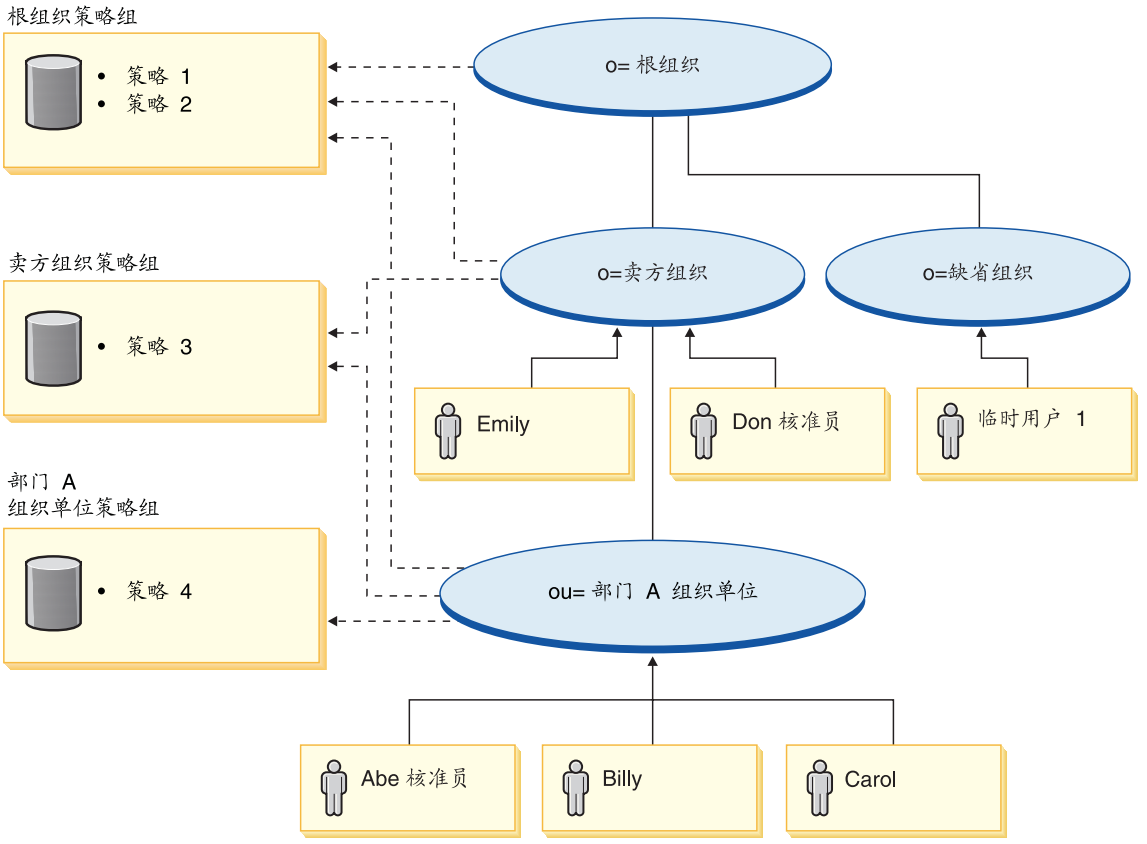

图 21.

对于卖方组织所拥有的任何资源，卖方组织策略组中的策略和根组织策略组中的那些策略是适用的。由于卖方组织显式预订了那两个策略组（带虚线的箭头表示预订），所以这些策略是适用的。共有 3 个策略适用于此组织。此示例就是拥有资源的组织显式预订策略组的一个示例。该图还显示了一个示例，其中组织未显式地预订任何策略组，因此访问控制框架必须向上进一步查看该层次结构。对于缺省组织拥有的资源，则必须沿层次结构上溯至根组织。根组织仅预订一个策略组，因此，那些是适用于缺省组织的资源的唯一策略（策略 1 和 2）。

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： Express Professional 对于与关系组有关的以下各节，认识到只有组织 RootOrganization、DefaultOrganization 和 SellerOrganization 在 WebSphere

Commerce Professional Edition 和 WebSphere Commerce - Express 中可用是很重要的。Business 有关其它组织的示例仅适用于 WebSphere Commerce Business Edition。

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 Professional Edition 或 WebSphere Commerce Business Edition 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，第一个 `<parameter name="aName" value="aValue" />` 元素介于某个用户和组织实体之间。最后一个 `<parameter name="aName" value="aValue" />` 元素介于某个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。可基于 defaultAccessControlPolicies.xml 文件创建您自己的 XML 文件。关于创建此基于 XML 的信息的更多信息，请参阅《*WebSphere Commerce 安全性指南*》。

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 生产联机帮助。

ACPOLICY

PolicyName	Member_Id	MbrGrp_Id	AcActGrp_id	AcResGrp_Id	AcRelGrp_Id
StoreAdministrators ExecuteStoreAdministrators 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	StoreAdministratorsCmdResourceGroup

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

本部分给出一个交互图，其中说明访问控制是如何在 WebSphere Commerce 访问控制策略框架中作用的。

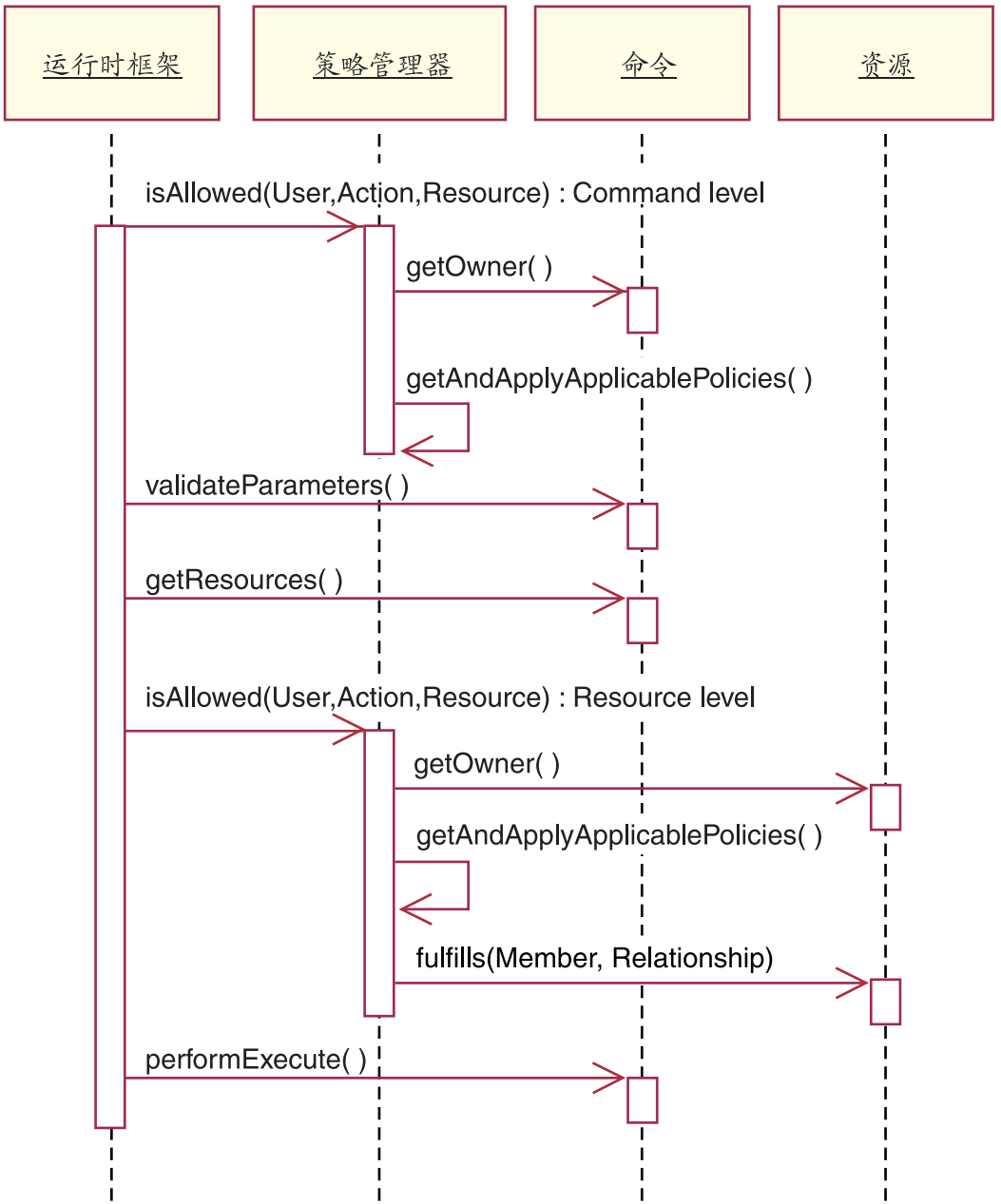

图 23.

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。它通过在资源所有

者预订的组中搜索策略对此进行确定。如果资源所有者未预订任何策略组，则它在资源所有者的最近上级所预订的组中搜索策略。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。
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. getAndApplyApplicablePolicies()
   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适用的策略。如果至少一个适用的策略授予访问权限，就能通过命令级别访问检查，策略管理器将继续至下一步以开始检查资源级别授权。相反，如果没有任何适用的策略授予命令级别的访问权限，则策略管理器将在该处返回并拒绝访问。
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 99 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。

8. `getAndApplyApplicablePolicies()`
   访问控制策略管理器搜索适用的策略，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()`
   如果适用的策略具有指定关系或关系组，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

关于 WebSphere Commerce 访问控制模型的更多信息，请参阅《*WebSphere Commerce 安全性指南*》。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。通常，当创建新的企业 bean 和数据 bean 时，会出现要保护什么的问题。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. 如果为了应用访问控制策略的目的而要根据属性（而不是它的 Java 类名）将资源分组，则 bean 的远程接口还必须扩展 com.ibm.commerce.grouping.Groupable 接口。
4. 企业 bean 类继承 com.ibm.commerce.base.objects.ECEntityBean 的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。

   如果有访问控制策略将此资源包含至其资源组中并指定了关系或关系组，则必须实现 fulfills 方法。如果一个访问控制策略中带有隐式资源组，该资源组基于特定的属性值包含此资源的特定实例（例如，如果有一个访问控制策略属于且仅属于状态为 “P” （未决）的订单），则必须实现 getGroupingAttributeValue 方法。

   注意，如果仅需 “owner” 关系，则您不需要覆盖 fulfills 方法。在此情况下，策略管理器将使用 getOwner() 方法的结果。

   这些方法的缺省实现在以下代码片段中显示。这些实现来自 ECEntityBean 类。

```
**************************************************************************
public Long getOwner() throws Exception
{
      return null;
}
**************************************************************************
**************************************************************************
public boolean fulfills(Long member, String relationship)
   throws Exception {
      return false;
}
**************************************************************************
```

```
*************************************************************************
public Object getGroupingAttributeValue(String attributeName,
   GroupingContext context) throws Exception {
      return null;
}
*************************************************************************
```

以下是这些方法的样本实现（基于 OrderBean bean 中所使用的实现）：

- 对于 getOwner 方法，所提供方法的逻辑为:

```
*************************************************************************
  com.ibm.commerce.common.objects.StoreEntityAccessBean storeEntAB =
  new com.ibm.commerce.common.objects.StoreEntityAccessBean();
  storeEntAB.setInitKey_storeEntityId(getStoreEntityId().toString());
  return storeEntAB.getMemberIdInEJBType();
*************************************************************************
```

- 对 fulfills 方法，所提供方法的逻辑为:

```
*************************************************************************
if ("creator".equalsIgnoreCase(relationship))
{
   return member.equals(bean.getMemberId());
}
else if ("BuyingOrganizationalEntity".equalsIgnoreCase(relationship))
{
   return (member.equals(bean.getOrganizationId()));
}
else if ("sameOrganizationalEntityAsCreator".
   equalsIgnoreCase(relationship))
{
   com.ibm.commerce.user.objects.UserAccessBean creator =
     new com.ibm.commerce.user.objects.UserAccessBean();
   creator.setInitKey_MemberId(bean.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;
*************************************************************************
```

- 对 getGroupingAttributeValue 方法，所提供方法的逻辑为:

```
*************************************************************************
      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。

注意，如果您查看其它 WebSphere Commerce 公共实体 bean 来理解 getOwner、fulfills 和 getGroupingAttributeValue 方法如何实现，您将注意到这些方法是在 bean 的访问助手类中实现的。由于方法是在访问助手类（而非直接在 bean

类）中实现的这一事实，因此方法签名会稍有不同。特别地，对于具有要传入访问助手中的对象自身的额外输入参数的那些方法。

您必须确保当您创建新的 bean 时，您直接在 bean 类中实现这些方法。此外，您不得在 WebSphere Commerce 公共实体 bean 的访问助手类中修改这些方法中的任何一个。

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。

要决定数据 bean 是否应受保护，请考虑以下几点:

1. 数据 bean 信息中是否有需要保护的信息？例如，是否属于专用性质的信息？若为否，则由访问控制直接保护不是必需的。若为是，则继续。
2. 数据 bean 由视图实例化。视图是否使用命令上下文中的信息（或某些其它预先确定的信息）来实例化数据 bean？若为是，访问控制已决定允许访问，则对此数据 bean 的直接保护不是必需的。若为否，则继续。
3. 视图是否使用来自某些输入参数的信息实例化数据 bean？在此情况下，由于实际上您并不确定访问控制已决定是否允许用户访问此信息，所以，您应保护新数据 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 必须执行以下操作:

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。

   当数据 bean 实现可保护接口时，数据 bean 管理器调用 `isAllowed` 方法根据现有的访问控制策略，确定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);
   ```

   其中 *protectable_databean* 是要保护的数据 bean。
2. 如果与 bean 交互的资源是按除资源 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访问控制策略，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 程序设计模块，这是通过使命令接口从 com.ibm.commerce.command.ControllerCommand 接口扩展，并让命令实现从 com.ibm.commerce.command.ControllerCommandImpl 扩展来完成的。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。

只在命令同时对同一资源类下的不同实例执行读和写操作时，才需要指定操作。例如 OrderCopy 命令，它可以读取源订单并写至目标订单。在此情况下，必须在两个操作间作出区别。可通过为源订单指定“-Read”操作和为目标订单指定“-Write”操作来做到这一点。当访问控制框架检测到这些操作时，在搜索适用的策略前，将自动以命令的接口名称分析它们。在此例中，将最终用于策略的操作为“com.ibm.commerce.order.commands.OrderCopyCmd-Read”和“com.ibm.commerce.order.commands.OrderCopyCmd-Write”操作。

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。然后如有必要，您可以在新控制器命令的 performExecute 方法中使用同一实例变量。

要决定您是否必须覆盖 getResources 方法，考虑以下几点:

- 如果您基于预定义的信息源（例如命令上下文）来派生资源，您不需要覆盖 getResources 方法。例如，WebSphere Commerce UserRegistrationUpdate 命令从命令上下文来派生用户标识。在此例中，用户已得到授权对他们自己的注册信息执行操作，所以不需要覆盖 getResources 方法。
- 如果您的命令正任意指定新资源（且此资源属于专用性质），您必须覆盖 getResources 方法。举例而言，WebSphere Commerce OrderItemUpdate 命令将订单标识作为输入参数。在本例中，当实例化订单资源后，您不了解用户是否有权对该特定资源执行操作。在此情况下，将覆盖 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 方法会在它更新现有的订单时，返回一个或多个订单对象（它们是可保护的）。由于未指定操作，操作缺省为 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，该操作可能不再继续进行（需要具有全部三个资源的访问权）。

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。此方法的使用是可选的。

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。在本例中，getResources 方法应当返回新资源的容器。例如，如果创建了订单，则是在商店资源内部完成的；同时在组织资源里创建一个新的用户。

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。

修改对现有 WebSphere Commerce 资源的访问控制

本节提供信息以引导您完成修改对现有 WebSphere Commerce 资源的访问控制。特别地，将会复查以下方案:

- 对已受访问控制保护的现有 WebSphere Commerce 实体 bean 添加新的关系。
- 对还未受访问控制保护的现有 WebSphere Commerce 实体 bean 添加访问控制保护。
- 在扩展现有控制器命令时，理解对访问控制的隐含意义。

对现有的 WebSphere Commerce 实体 bean 添加新的关系

实现 Protectable 接口的 WebSphere Commerce 实体 bean 已受访问控制的保护。访问控制要求的条款由在现成功能部件和 WebSphere Commerce 功能中使用 bean 的方式决定。您可能遇到需要将附加关系添加至对此类 bean 的访问控制的情况。例如，如果您将现有的 bean 用于您的某个定制代码中，或您修改现有的 WebSphere Commerce 公共实体 bean，则您可能需要将附加关系添加至该 bean。

以下列表提供用于对已受访问控制保护的现有 WebSphere Commerce 实体 bean 添加新关系的高级步骤。

1. 检查实体 bean 的现有 fulfills 方法。该方法位于 bean 的访问助手类中。不要修改此类，只将它用来判断您是否需要将一个或多个新关系添加至此逻辑，或判断您是否需要覆盖此方法。例如，以下 fulfills 方法出现在 com.ibm.commerce.fulfillment.objsrc.FulfillmentCenterBeanAccessHelper 类中:

```
public boolean fulfills(Object obj, Long member, String relationship)
   throws Exception {

 FulfillmentCenterBean bean = (FulfillmentCenterBean) obj;

 if ("ShippingArrangementOrganizationalEntity".
   equalsIgnoreCase(relationship))
 {
      FulfillmentJDBCHelperAccessBean ffmJDBCAB =
         new FulfillmentJDBCHelperAccessBean();
      int count = ffmJDBCAB.
         checkFulfillmentCenterByMemberIdAndFulfillmentCenterId(
         member,bean.getFulfillmentCenterId());
      if(count>0)
     return true;
 }
      return false;
}
```

2. 下一步为在 bean 类中创建新的 fulfills 方法。例如，您可以在 com.ibm.commerce.fulfillment.objects.FulfillmentBean.java 类中创建新的 fulfills 方法。对此方法的声明应当为:

```
public boolean fulfills(Long member, String relationship)
   throws Exception {
      // Place holder for relationship information
 }
```

3. 如果您将附加关系添加至现有的关系，则方法的第一行应当从超类中调用 fulfills 方法。然后，如果该方法返回 false，则按如下方式检查新关系:

```
public boolean fulfills(Long member, String relationship)
   throws Exception {
      if (super.fulfills().equals(false))
      {
      // Check if new relationship is met
     return true;
```

```
        }

        return false;
    }
```

4. 如果您完全替换原始实现中的关系，您不得调用 super.fulfills 方法，如下:

```
public boolean fulfills(Long member, String relationship)
   throws Exception {
      // Check if new relationship is met
      // If it is, then return true;

     // If the relationship is not met, return false;
 }
```

5. 保存更改。为该 bean 以及相应的访问 bean 重新生成部署和 RMIC 代码。

将访问控制添加至未受保护的现有 WebSphere Commerce 实体 bean

如果您使用现有的 WebSphere Commerce 实体 bean，且应用程序需要该 bean 受访问控制的保护，您可以添加此保护。

以下列表提供用于保护 WebSphere Commerce 访问控制系统下的现有 WebSphere Commerce 实体 bean 的高级步骤。

1. 打开 *BeanName*.java 类。这是远程接口。修改它以使它扩展 `com.ibm.commerce.security.Protectable` 接口。
2. 如果为了应用访问控制策略的目的而要根据属性（而不是它的 Java 类名）将资源分组，则 bean 的远程接口还必须扩展 `com.ibm.commerce.grouping.Groupable` 接口。
3. 将您的更改保存到远程接口。
4. 打开 *BeanName*Bean.java 类，其中 *BeanName* 是要向其添加访问控制保护的实体 bean 的名称。
5. 企业 bean 类继承 com.ibm.commerce.base.objects.ECEntityBean 的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。有关这些方法的更多信息，请参阅第 99 页的『在企业 bean 中实现访问控制』。
6. 保存更改。为该 bean 以及相应的访问 bean 重新生成部署和 RMIC 代码。

扩展控制器命令时了解访问控制的隐含意义

根据 WebSphere Commerce 编程模型，您可以创建您自己的对现有的控制器命令的实现。在本例中，您创建一个新的实现类，然后通过更新命令注册表来将该新实现类与现有的接口相关联。

执行此类扩展有三种可能的与访问控制相关的隐含意义:

1. 对 getResources 方法的影响。
2. 对命令级别访问控制策略的影响
3. 对资源级别访问控制策略的影响

后续部分对上述各点作了更详细描述。

对 getResources 方法的影响

如果您扩展一个现有的控制器命令（这意味着命令的现有逻辑将会同新定制逻辑一起执行），则新命令将作为现有命令的子类划分出来。新实现的 performExecute 方法调用超类的 performExecute 方法。如果新命令未访问任何需要保护的新资源，您并非必须覆盖 getResources 方法。然而，如果访问了新的可保护资源，新的命令应当在实现您自己的 getResource 逻辑前实现其自身的 getResources 方法并以此方法从超类调用 getResources 方法。

超类的 getResources 方法的结果应当存储在 AccessVector 类型的一个专用实例变量中。然后应当将本地 getResources 方法的结果附加到此向量的末尾。例如，新的实现类将包含与以下伪代码相似的代码:

```
private AccessVector resources = null;

public AccessVector getResources() throws ECException {
   // First, get the resources from the original implementation
   resources = super.getResources();

   // Now, append the new resources

   ////////////////////////////////////////
   //  Logic for getting new resources   //
   //  and appending to the vector.      //
  ////////////////////////////////////////

   return resources;
}
```

如果您未扩展现有控制器命令的逻辑，而是完全替换该逻辑，则新实现中将不调用 super.performExecute 方法。由此您将不需要从您自己的实现中调用 super.getResources 方法。而是仅按需要实现您自己的 getResources 方法。

对命令级别访问控制策略的影响

控制器命令的命令级别策略由作用于命令资源的“执行”操作构成。命令资源由其接口名称指定。当您扩展一个现有的命令时，命令将仍然实现其原始接口，同样，现有的命令级别策略足以维持先前的命令级别访问控制。由此，不需要更改。

对资源层次访问控制策略的影响

如果您创建了现有命令的新实现并将此实现与该现有命令的接口相关联，则不需要对资源层次的访问控制策略作更改。

如果您创建一个新的扩展现有实现的实现类，并在此类中调用 super.performExecute 方法，并且还实现新的接口，就需要对资源级别访问控制策略进行更改。

在这后一个例子中，如果新的命令实现 getResources 方法或从基本命令中继承 getResources 方法的重要实现，则需要更改资源级别策略。通常，资源级别策略由作用于商务对象资源的命令操作构成。由于操作只是简单的对命令接口的字符串表示，通常需要将新的命令添加至基本命令的操作组中。然而，如果新命令使用简单的返回空覆盖了 getResources 方法的基本实现，则将不会对此新命令执行任何访问控制检查。注意如果未小心执行，可能潜在的造成为恶意用户打开新的命令。

修改资源级别访问控制策略时，以下为高级步骤:

1. 找到 defaultAccessControlPolicies.xml 文件，该文件可在以下目录找到:
   - Developer *WCDE_installdir*\Commerce\xml\policies\xml
2. 制作此文件的副本。例如，将文件命名为 *myDefaultAccessControlPolicies.xml*。
3. 在此新文件中，您必须将新的接口定义为新的操作。例如，添加以下内容:

   ```
   <Action Name="yourNewInterface"
      CommandName="yourNewInterface">
   </Action>
   ```

   其中 *yourNewInterface* 是新接口的名称。
4. 下一步，您必须搜索整个文件来定位原始操作所属的所有操作组，并添加您的新操作。
5. 如果该新命令对先前未经基本命令指定的资源进行操作，则还必须更改相应的访问控制策略的资源组来容纳新的资源。
6. 一旦您完成了对 XML 文件的更改，您可以除去未修改的部分。确保您将文本保留在 `<Policies>` 标记前。
7. 根据《*WebSphere Commerce 安全性指南*》中包含的指示信息，将新的策略信息装入数据库。

用于开发目的的样本访问控制策略

本节提供一些可用于开发环境中的非常简单的访问控制策略，这样您可以迅速测试新资源。它们不是设计用于任何 WebSphere Commerce 生产环境中的，因为它们不提供足够的资源保护。

关于如何装入这些策略的信息，请参阅《*WebSphere Commerce 安全性指南*》。

新视图的样本访问控制策略

如果创建新视图，则可以使用以下访问控制策略，以便能够在开发环境中测试新视图（为您的环境修改该策略并使用 acpload 命令将其导入）：

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">

<Policies>
   <Action Name="YourNewView"
        CommandName="YourNewView">
   </Action>
   <ActionGroup Name="AllSiteUsersViews"
     OwnerID="RootOrganization">
      <ActionGroupAction Name="YourNewView"/>
   </ActionGroup>
</Policies>
```

其中 *YourNewView* 是新创建的视图的名称。前述的访问控制策略将新视图添加至现有的 AllSiteUsersViews 操作组。该策略允许任何用户访问该新视图。

新控制器命令的样本命令级别访问控制策略

控制器命令需要访问控制策略以满足访问控制框架的需要。如果您创建新的控制器命令，命令接口的名字指定为一个资源。可以对新命令修改以下 XML 片断并使用 acpload 命令装入:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">

<Policies>

   <Action Name="ExecuteCommand"
         CommandName="Execute">
   </Action>

   <ResourceCategory Name="com.yourcompany.yourpackage.commands.
         YourControllerCmdResourceCategory"
         ResourceBeanClass="com.yourcompany.yourpackage.commands.
         YourControllerCmd">
        <ResourceAction Name="ExecuteCommand"/>
   </ResourceCategory>

   <ResourceGroup Name="AllSiteUserCmdResourceGroup"
```

```
    OwnerID="RootOrganization">
     <ResourceGroupResource Name="com.yourcompany.yourpackage.commands.
        YourControllerCmdResourceCategory" />
  </ResourceGroup>

</Policies>
```

其中:

- *com.yourcompany.yourpackage.commands* 代表封装结构
- *YourControllerCmd* 代表新控制器命令的名称

例如，以下 XML 文件用于为在本书所包含教程中创建的新控制器命令装入访问控制策略。

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">

<Policies>
   <Action Name="ExecuteCommand"
   CommandName="Execute">
   </Action>

   <ResourceCategory Name="com.ibm.commerce.sample.commands.
          MyNewControllerCmdResourceCategory"
          ResourceBeanClass="com.ibm.commerce.sample.commands.
          MyNewControllerCmd">
         <ResourceAction Name="ExecuteCommand"/>
   </ResourceCategory>

   <ResourceGroup Name="AllSiteUserCmdResourceGroup"
     OwnerID="RootOrganization">
      <ResourceGroupResource Name="com.ibm.commerce.sample.commands.
         MyNewControllerCmdResourceCategory" />
   </ResourceGroup>

</Policies>
```

新命令和企业 bean 的样本资源级别访问控制策略

以下 XML 文件是从本指南包含的教程中摘出的。它可充当您创建新的实体 bean 时的访问控制要求的一个模板。在以下文件的例子中，新的实体 bean 称为 Bonus bean，它对应于 XBONUS 数据库表，并由 MyNewControllerCmd 控制器命令使用。在此访问控制策略中，只有 bonus bean 对象的创建者可以对该对象执行 MyNewControllerCmd 操作。

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">

<Policies>
   <Action Name="MyNewControllerCmd"
      CommandName="com.ibm.commerce.sample.commands.MyNewControllerCmd">
   </Action>
```

```
    <ResourceCategory Name="com.ibm.commerce.extension.objects.
        BonusResourceCategory"
        ResourceBeanClass="com.ibm.commerce.extension.objects.Bonus" >

          <ResourceAction Name="MyNewControllerCmd" />
    </ResourceCategory>

    <ActionGroup Name="MyNewControllerCmdActionGroup"
      OwnerID="RootOrganization">
    <ActionGroupAction Name="MyNewControllerCmd"/>
    </ActionGroup>

    <ResourceGroup Name="BonusResourceGroup"  OwnerID="RootOrganization" >
       <ResourceGroupResource Name="com.ibm.commerce.extension.objects.
            BonusResourceCategory" />
    </ResourceGroup>
    <Policy Name="AllUsersUpdateBonusResourceGroup"
       OwnerID="FashionFlowMemberId"
       UserGroup="AllUsers"
       UserGroupOwner="RootOrganization"
       ActionGroupName="MyNewControllerCmdActionGroup"
       ResourceGroupName="BonusResourceGroup"
       RelationName="creator"
       PolicyType="groupableStandard">
    </Policy>

    <PolicyGroup Name="ManagementAndAdministrationPolicyGroup"
      OwnerID="RootOrganization">
    <!-- Define policies in this policy group -->
    <PolicyGroupPolicy  Name="AllUsersUpdateBonusResourceGroup"
             PolicyOwnerID="FashionFlowMemberId" />

    </PolicyGroup>

</Policies>
```

其中 *FashionFlowMemberId* 是使用新资源的商店的成员标识。

在上述访问控制策略中，控制器命令的接口名称按操作指定，而不是使用其数据包名称进行全限定。如果您的应用程序具有多个名称相同的接口，则在访问控制策略中将它们指定为操作时必须以其数据包名称对它们进行全限定。例如，如果接口名称有不明确的地方，上述访问控制策略将要求更改，如下（请注意，仅显示更改行，且需修改处以粗体字型显示）：

```
<Action Name="com.ibm.commerce.sample.commands.MyNewControllerCmd"
      CommandName="com.ibm.commerce.sample.commands.MyNewControllerCmd">
.
.
.
<ResourceAction Name="com.ibm.commerce.sample.commands.MyNewControllerCmd" />
.
```

```
.
.
<ActionGroupAction Name="com.ibm.commerce.sample.commands.MyNewControllerCmd"/>
```

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 null 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。当抛出系统异常时，将自动记录它。

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。在客户的浏览器中对客户显示用户消息。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。

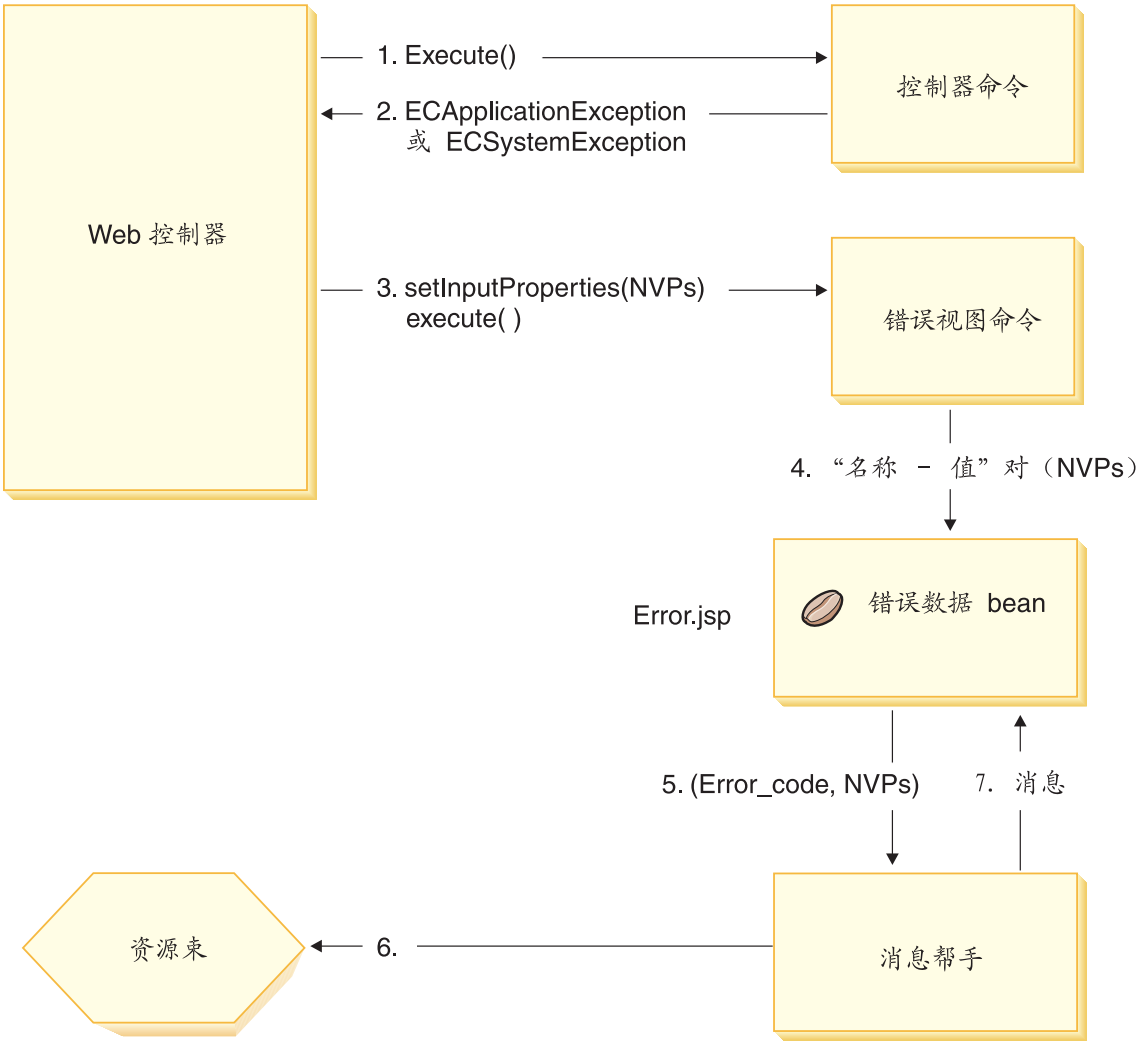

图 24.

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 null 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。

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。此新类应在您自己的数据包中创建，并存储在 WebSphereCommerceServerExtensionsLogic 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 的文本文件。如果消息将由单个商店 servlet 使用，则将该文件放在以下目录中:

Developer *workspace_dir*\Stores\Web Content\WEB-INF\classes*storeDir*

其中 *storeDir* 是您的商店的名称。

如果消息将由 WebSphere 贸易加速器使用，则将该文件放在以下目录中:

Developer *workspace_dir*\CommerceAccelerator\Web Content\WEB-INF\classes

如果消息将由管理控制台使用，则将该文件放在以下目录中:

Developer *workspace_dir*\SiteAdministration\Web Content\WEB-INF\classes

Business 如果消息将由组织管理控制台使用，则将该文件放在以下目录中:

Developer *workspace_dir*\OrganizationAdministration\Web Content\WEB-INF\classes

如果消息将由企业应用程序中的所有 servlet 全局使用，则将该文件放在以下目录中:

Developer *workspace_dir*\WebSphereCommerceServer\properties

上述目录在开发环境的上下文中指定。一旦您完成了该环境中的测试，关于部署到目标 WebSphere Commerce Server 的信息，请参阅第 187 页的第 9 章，『部署详细信息』。

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}".
```

执行流程跟踪

当您需要跟踪通过商业应用程序的执行流程时，您应当使用 WebSphere Application Server JRas 工具。这是一个可由应用程序使用的消息记录和诊断跟踪 API。

有关如何在您的定制代码中使用此工具的信息，请参阅 WebSphere Application Server InfoCenter。

关于配置开发环境中的组件跟踪的信息，请参阅第 365 页的附录 A，『配置 WebSphere Commerce 开发环境中的 WebSphere Commerce 组件跟踪』。

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 servlet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 servlet 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 145 页的第 7 章，『贸易协议和业务策略（Business Edition）』。

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

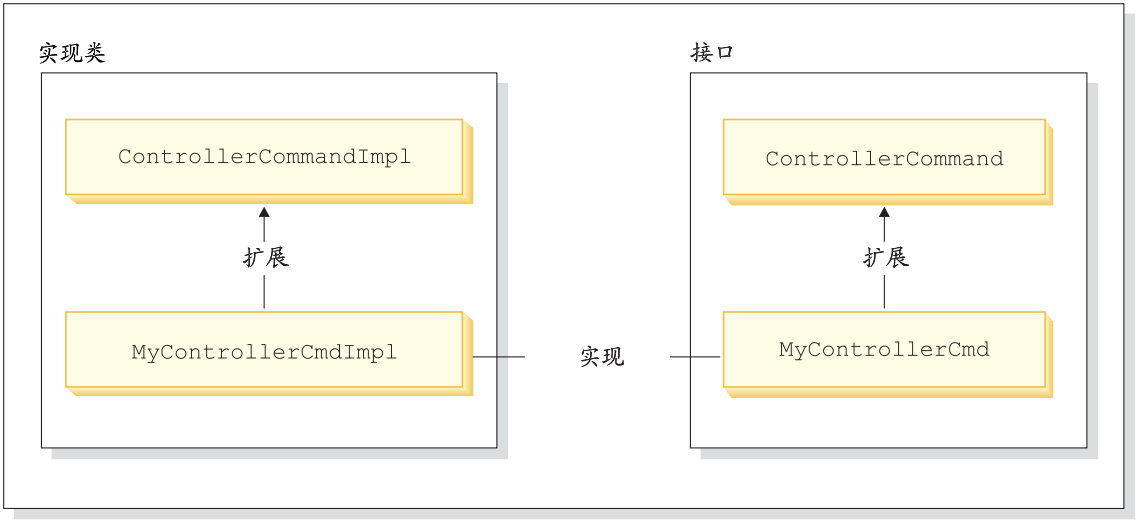

图 25.

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

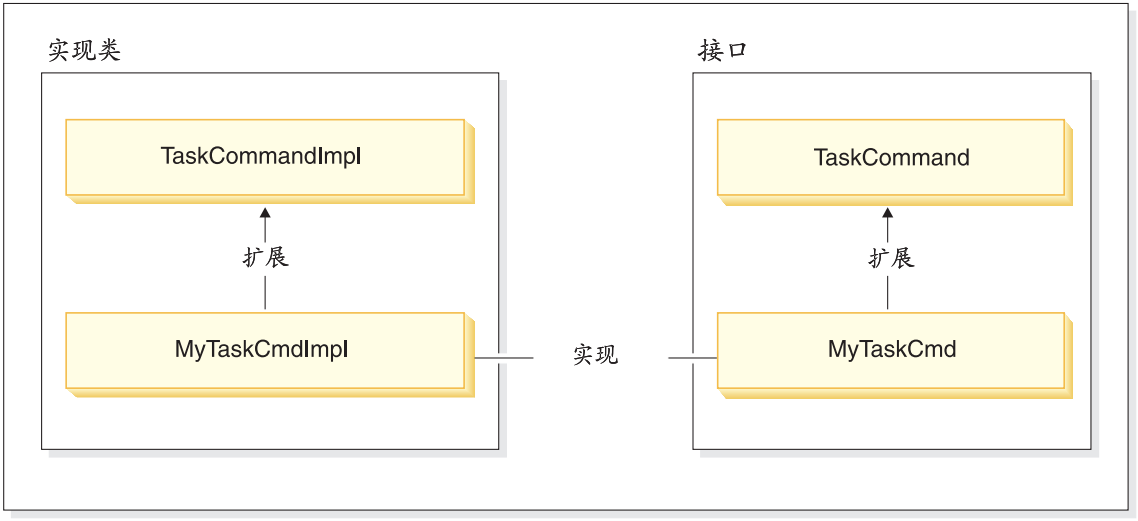

图 26.

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

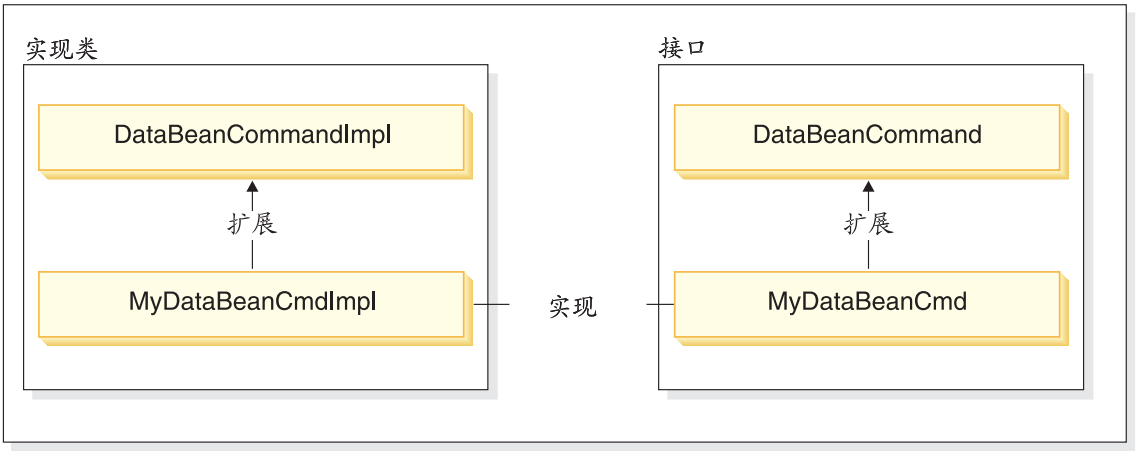

图 27.

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，在位于 WebSphere Commerce 工作空间的项目中维护定制代码，这些项目是为定制代码预定义的。提供了两个预定义的项目，WebSphereCommerceServerExtensionsLogic 项目和 WebSphereCommerceServerExtensionsData 项目。第一个项目用于命令和数据 bean 逻辑，第二个项目用于所创建的任何企业 bean。

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数据包中。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。

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，此数据包应当保留在存储命令数据包的项目（WebSphereCommerceServerExtensionsLogic）中。依据上述示例，您可将 com.bigbusiness.databeans 数据包置于 WebSphereCommerceServerExtensionsLogic 项目中。

创建新的实体 bean 时，应当将它们存储在 WebSphereCommerceServerExtensionsData 项目中。所以，您可能有包含 com.bigbusiness.objects 数据包的 WebSphereCommerceServerExtensionsData 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

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，因此此方法后的逻辑与 getLanguageId() 方法的逻辑类似。

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。

有关命令上下文 API（应用程序编程接口）的完整详细信息，请参考 WebSphere Commerce 生产和开发联机帮助中的“参考”主题。

对 URL 命令上下文信息的临时更改

在另一商店的上下文中或为另一位用户覆盖某些命令上下文信息并执行 URL 命令是可能的。URL 命令拥有以下 URL 输入参数，这些参数允许命令上下文中的此临时切换:

- forStoreId
- forUser
- forUserId

forStoreId URL 输入参数允许您指定要用于此特定 URL 请求的商店标识。作用是暂时将命令上下文中的 storeId 值更改为指定商店的 storeId 值，但此更改仅在该 URL 命令的持续时间内有效。

forUser 和 forUserId URL 输入参数都允许您指定为指定用户执行的命令，尽管当前登录的可能为不同用户。这在客户服务代表需要协助客户时极为有用。例如，客户服务代表可以代表该客户更新其客户地址信息，只需使用 URL 输入参数指定客户的用户名或用户标识。此对用户信息的更改仅在指定 URL 请求的持续时间内有效。

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。注册用户组中有客户和管理员。

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 94 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 83 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:

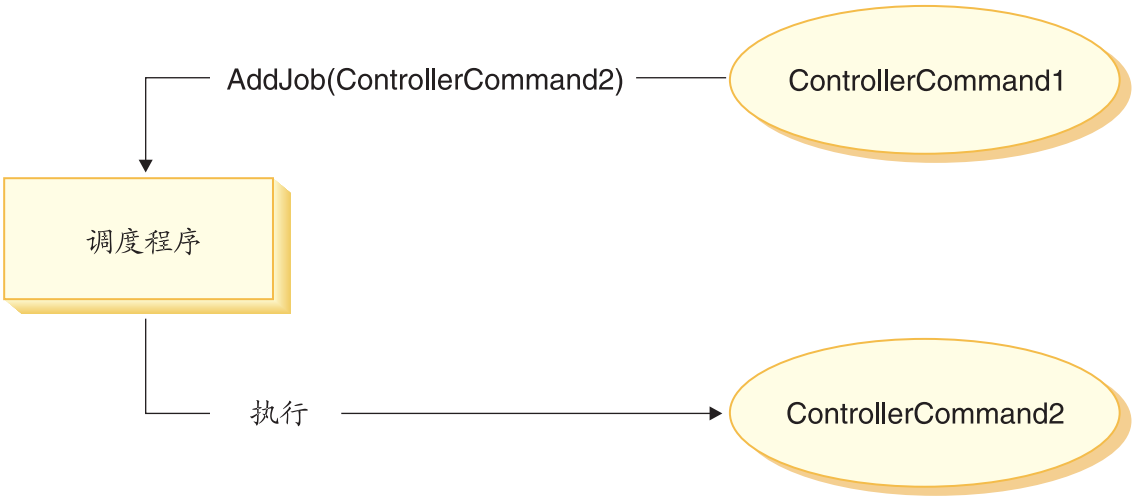

图 28.

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 41 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。然后这作为一般转发视图实现。

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.command.TaskCommand 接口的接口。如第 124 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（它以您希望的方式执行任务）的实现替换您希望覆盖的任务命令的实现。

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 141 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 生产和开发联机帮助的“参考”主题。

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。

以下代码片段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 生产和开发联机帮助的“参考”主题，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 WebSphere Studio Application Developer 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
{

}
```

WebSphere Studio Application Developer 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（Business Edition）

本章仅适用于 WebSphere Commerce Business Edition。

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 Business Edition 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 Business Edition 时，可以使用 RFQ 在线过程来协商条款和条件集，或脱机协商然后使用 WebSphere 贸易加速器中的业务关系管理界面进行捕获。

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，它可以指定特定的供货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。

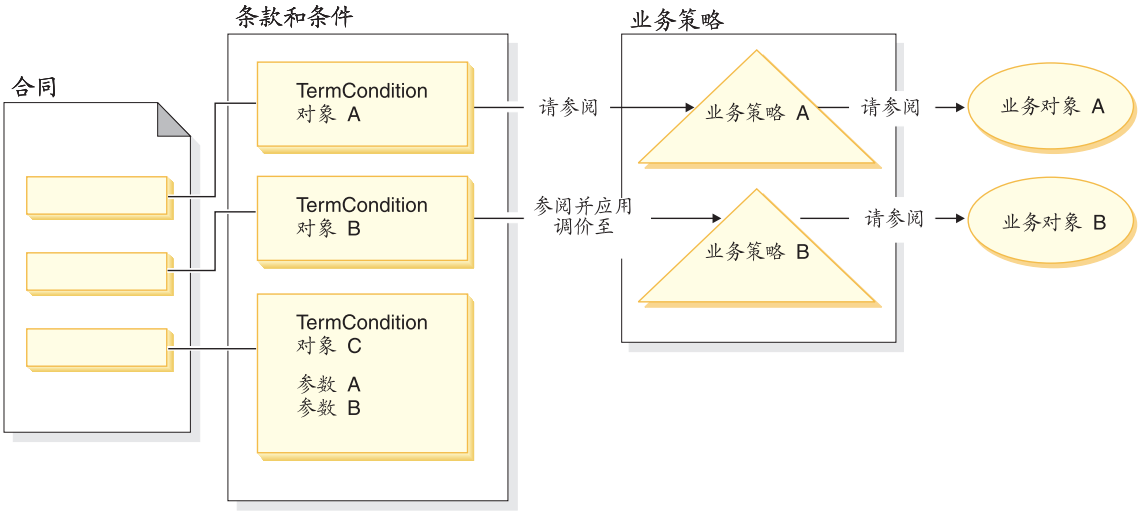

图 29.

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，在图中，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”。
- 再例如，在图中，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 148 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()` 命令获取条款和条件的价格。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:

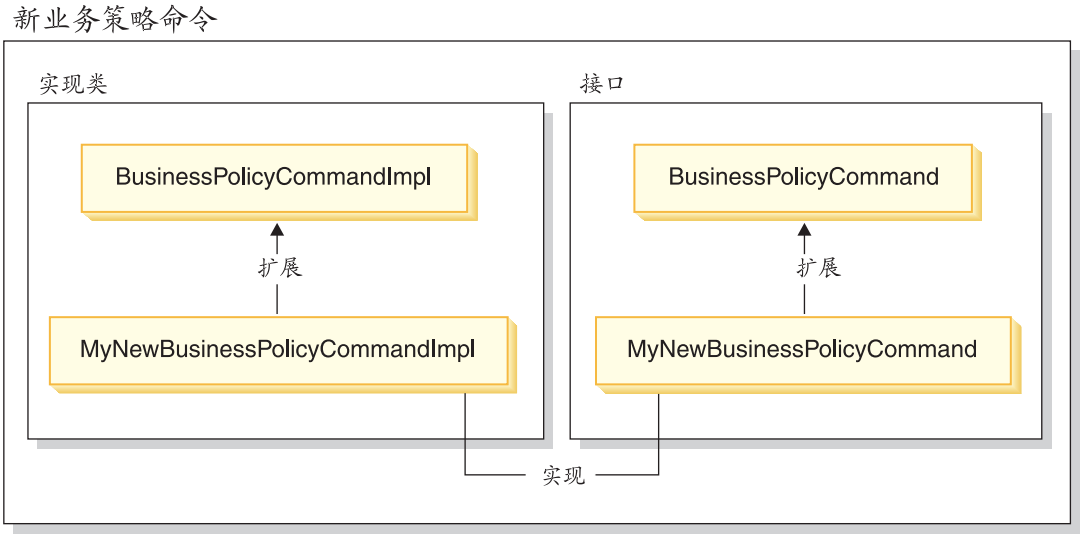

图 30.

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 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 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
STRINGFILED1	ProductSet2	

列名	样本数据行 1	样本数据行 2
INTEGERFIELD2	10002	
INTEGERFIELD3	1	
BIGINTFIELD1	10051	
FLOATFIELD1	-50.0	
SEQUENCE	1	6

POLICYTC 表样本数据

下表显示 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 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	null	null
ENDTIME	null	null

TRADEPOSCN 表样本数据

下表显示 TRADEPOSCN 数据库表中的相关样本数据。

	列名			
	TRADEEPOSCN_ID	MEMBER_ID	NAME	TYPE
样本数据行	10051	-2001	ToolTech	S

SHIPMODE 表样本数据

下表显示 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 XSD（XML 模式定义）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。

WebSphere Commerce V5.5 引入了新的合同类型。关于可用的合同类型的信息，请参阅 WebSphere Commerce 生产联机帮助中“概念”主题的“商业帐户”子主题。

以下部分使用 BuyerContract 作为扩展示例中的合同类型。类似的扩展方法用于其它合同类型。

创建新业务策略

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.');
```

作为创建新业务策略类型的最终步骤，可以编码一个或多个新业务策略类型接口。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业务策略类型时，也必须创建新业务策略类型接口。

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）。若为是 NULL，则开始日期为即日。
- ENDDATE
  策略的结束日期（指定为时间戳记）。若为是 NULL，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相关（这是通过在 XSD（XML 模式定义）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），或者可以创建与新业务策略相关的新条款和条件对象。

创建新条款和条件

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 XSD 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 语句显示如何在数据库中注册新的条款和条件的示例:

```
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 XSD 中注册新的条款和条件

要让新的条款和条件在合同中可用，您必须创建新的 XSD 文件来定义新的条款和条件。您还必须更新 Package.xsd 文件来包含该新 XSD 文件。

要创建新的 XSD 文件，请执行以下操作:

1. 导航到以下目录:
   - 400 *WC_userdir*/instances/*instanceName*/xml/trading/xsd
   - AIX Linux Solaris *WC_installdir*/xml/trading/xsd
   - Windows *WC_installdir*\xml\trading\xsd

   其中 *instanceName* 是 WebSphere Commerce 实例的名称。
2. 在此目录下创建新 XSD 文件。以下显示了用于示例 MyTC 的 XSD。文件为 CustomizedBuyerContract.xsd:

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.ibm.com/WebSphereCommerce"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:wc="http://www.ibm.com/WebSphereCommerce"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">

  <!-- include basic trading agreement xsd -->

  <include schemaLocation="BuyerContract.xsd" />
  <complexType name="MyTCType">
            <complexContent>
            <extension base="wc:TermConditionType"/>
            </complexContent>
        </complexType>
    <element name="MySubTC" substitutionGroup="wc:AbstractCustomizedTC">
        <complexType>
            <complexContent>
                <extension base="wc:MyTCType">
                    <sequence>
                        <element ref="wc:ProductSetPolicyRef"/>
                    </sequence>
                    <attribute name="attr1" type="normalizedString"
                         use="required"/>
                    <attribute name="attr2" type="int" use="required"/>
                </extension>
            </complexContent>
        </complexType>
    </element>
</schema>
```

3. 保存该新文件。

接下来，您必须更新 Package.xsd 文件以除去 BuyerContract.xsd 文件并将 CustomizedBuyerContract.xsd 文件包括进来而代之，如下:

1. 导航到以下目录:
   - Windows *WC_installdir*\xml\trading\xsd
   - AIX Solaris Linux *WC_installdir*/xml/trading/xsd
   - 400 *WC_userdir*/instances/*instanceName*/xml/trading/xsd

   其中 *instanceName* 是 WebSphere Commerce 实例的名称。
2. 在文本编辑器中打开 Package.xsd 文件。
3. 找到有关 BuyerContract.xsd 的部分并按以下方式修改:

```
<!--include schemaLocation="BuyerContract.xsd"/-->
<include schemaLocation="CustomizedBuyerContract.xsd"/>
```

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，通常，当创建新的企业 Bean 时，将 bean 放置到 WebSphereCommerceServerExtensionsData 项目中，而不是将它们包括在包含 WebSphere Commerce 实体 bean 的一个 EJB 组中。然而在此例中，因为条款和条件的所有新实体 bean 必须继承 WebSphere Commerce TermCondition bean，所以必须将新条款和条件 bean 放置到 Enablement-RelationshipManagementData 项目中。以下部分描述如何使用 WebSphere Studio Application Developer 中的工具创建新的企业 bean。

创建新的企业 bean: 要为新的条款和条件创建新的 CMP 企业 Bean，请执行以下操作:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**。
2. 用鼠标右键单击 **Enablement-RelationshipManagementData** 模块并选择**新建 > 企业 bean**。
   “企业 bean 创建”向导打开。
3. **Enablement-RelationshipManagementData** 已在 **EJB 项目**下拉列表中选定。单击**下一步**。
4. 请在“创建企业 bean”窗口执行以下操作:
   a. 选择**带有受容器管理的持久性（CMP）字段的实体 bean**
   b. 在 **Bean 名称**字段中，为您的 bean 输入适当的名称。对于此示例，输入 `MySubTC`

c. 在**源文件夹**字段中，保留指定的缺省值（ejbModule）。

d. 在**缺省数据包**字段，输入 `com.ibm.commerce.contract.objects`。

e. 单击**下一步**。

5. 请在“企业 bean 详细信息”窗口中执行以下操作:

a. 从 **Bean 超类**下拉列表，选择 **TermCondition**。

b. 单击**添加**来添加新的 CMP 属性。

“创建 CMP 属性”窗口打开。在此窗口中，请执行以下操作:

1) 在**名称**字段中，为该新的 CMP 字段输入适当的名称。对于此示例，输入 `attr1`。

2) 在**类型**字段中，为该字段输入适当的数据类型。对于此示例，输入 `String`。

3) 选择**以 getter 和 setter 方法访问**复选框。

4) 选择**将 getter 和 setter 方法提升到远程接口**复选框。

5) 清除**使 getter 只读**复选框。

6) 单击**应用**。

7) 创建另一个属性。在**名称**字段中，输入 `attr2`。

8) 在**类型**字段中，为该字段输入适当的数据类型。对于此示例，输入 `Integer`。

9) 清除**使 getter 只读**复选框。

10) 单击**应用**。

11) 单击**关闭**以关闭此窗口。

6. 单击**完成**。

将字段从新 bean 映射到 TERMCOND 表中: 下一步是将字段从新 bean 映射到 TERMCOND 表中的列。要创建此映射，请执行以下操作:

1. 切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
2. 展开以下文件夹: **Enablement-RelationshipManagementData > ejbModule >META-INF**。
3. 双击 **Map.mapxmi** 文件。
4. 在“企业 bean”窗格中，展开 **TermCondition** bean，然后展开 **MySubTC** bean，这样就可查看其属性。
5. 在“表”窗格，展开 **TERMCOND** 表，这样就可查看其列。
6. 将 **attr1** 字段从 MySubTC 拖至 TERMCOND 表中的 **STRINGFIELD3** 列。
7. 将 **attr2** 字段从 MySubTC 拖至 TERMCOND 表中的 **INTEGERFIELD3** 列。

8. 保存更改。

添加新 ejbCreate 方法: 下一步，通过执行以下操作将新的 ejbCreate 方法添加至 MySubTC bean:

1. 在 “J2EE 层次结构” 视图中，双击 **MySubTCBean** 类以打开它并查看其源代码。
2. 通过向类添加以下代码来创建新 ejbCreate(Long, Element) 方法:

```
public com.ibm.commerce.contract.objects.TermConditionKey
   ejbCreate(java.lang.Long argTradingId,
   org.w3c.dom.Element argElement)
   throws javax.ejb.CreateException,
     javax.ejb.FinderException,
     javax.naming.NamingException,
     javax.ejb.RemoveException {
          _initLinks();
    super.ejbCreate (argTradingId, argElement);
    this.attr1= null;
           this.attr2 = null;
      return null;
}
```

   保存代码更改。

3. 必须向人机接口添加新 ejbCreate(Long, Element) 方法。这将使方法在生成的访问 bean 中可用。要向主接口添加方法，请执行以下操作:
   a. 用鼠标右键单击 “概览” 视图中的 **ejbCreate(Long, Element)** 方法并选择 **企业 bean > 提升至主接口**。

添加新 ejbPostCreate 方法: 下一步通过执行以下操作创建新 ejbPostCreate(Long, Element) 方法从而使它拥有与 ejbCreate(Long, Element) 方法相同的参数:

1. 双击 **MySubTCBean** 类以打开它并查看其源代码。
2. 通过向类添加以下代码来创建新的 ejbPostCreate(Long, Element) 方法:

```
public void ejbPostCreate(java.lang.Long argTradingId,
   org.w3c.dom.Element argElement)
   throws javax.ejb.CreateException,
     javax.ejb.FinderException,
     javax.naming.NamingException,
     javax.ejb.RemoveException
        {
    parseXMLElement(argElement);
        }
```

   保存代码更改。

添加 parseXMLElement 方法: 在此步中，您必须按如下所述在 MySubTCBean 中创建 parseXMLElement 方法:

1. 双击 **MySubTCBean** 类以打开它并查看其源代码。
2. 如下更新方法:

```
public void parseXMLElement(org.w3c.dom.Element argElement) throws
    javax.ejb.CreateException,
    javax.ejb.FinderException,
    javax.naming.NamingException,
    javax.ejb.RemoveException
{
      super.parseXMLElement(argElement);
      if (argElement == null)
  return;
      String nodeName = argElement.getNodeName();
      if (nodeName.equals("TCCopy"))
  return;

 this.attr1 = argElement .getAttribute("attr1").trim();
 this.attr2 = new Integer (argElement.
     getAttribute("attr2").trim());
 // get element "ProductSetPolicyRef" from "MySubTC"
 Element ePolicyReference = null;
 ePolicyReference = ContractUtil.getElementByTag(
      argElement, "ProductSetPolicyRef");

      parseElementPolicyReference(ePolicyReference);

}
```

3. 保存工作。

添加 createNewVersion 方法: 在此步中，您必须按如下所述在 MySubTCBean 中创建新的 createNewVersion 方法:

1. 双击 **MySubTCBean** 类以打开它并查看其源代码。
2. 如下更新方法:

```
public Long createNewVersion(Long argNewTradingId) throws
  javax.ejb.CreateException,
  javax.ejb.FinderException,
  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);
```

```
    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;
}
```

3. 保存工作。

添加 getXMLString 方法: 在此步中，您必须按如下所述在 MySubTCBean 中创建新的 getXMLString 方法:

1. 双击 **MySubTCBean** 类以打开它并查看其源代码。
2. 如下覆盖方法:

```
public String getXMLString() throws javax.ejb.CreateException,
   javax.ejb.FinderException,
   javax.naming.NamingException
{
 return getXMLString(false);
}


public String getXMLString(boolean tcdata) throws
 javax.ejb.CreateException,
 javax.ejb.FinderException,
 javax.naming.NamingException
{
 String xmlTC = " <MySubTC %TC_DATA% " +
  "  attr1=\"" + this.attr1 +
  "\" attr2=\"" + this.attr2.toString() + "\">" +
  "%TC_DESC%" +
  "%PARTICIPANT%" +
                        "%XML_POLICYREFERENCE%" +
  " </MySubTC>";
 xmlTC = ContractUtil.replace( xmlTC, "%TC_DATA%",
      getXMLStringForTCData(tcdata));
      String xmlPolicy = getXMLStringForElementPolicyReference(
         "ProductSet") ;
      xmlTC = ContractUtil.replace( xmlTC, "%XML_POLICYREFERENCE%",
         xmlPolicy );
 xmlTC = ContractUtil.replaceAll( xmlTC, "%POLICY_REF_TYPE%",
     "ProductSetPolicyRef");
      return xmlTC;
}
```

3. 保存工作。

添加 markForDelete 方法: 在此步中，您必须按如下所述在 MySubTCBean 中创建新的 markForDelete 方法:

1. 双击 **MySubTCBean** 类以打开它并查看其源代码。

2. 如下覆盖方法:

```
public void markForDelete() throws
 javax.ejb.CreateException,
 javax.ejb.FinderException,
 javax.naming.NamingException
{
      // code: remove entries from associated tables which
      // cannot be deleted though delete cascade
}
```

3. 保存工作。

更新远程接口: 您必须确保已将以下方法按如下方式添加到远程接口:

1. 切换到 Business Professional "J2EE 导航器"视图 Express "项目导航器"视图。
2. 展开 **Enablement-RelationshipManagementData** 项目。
3. 展开 **com.ibm.commerce.contract.objects** 数据包。
4. 双击 **MySubTCBean** bean。
5. 在"概览"视图中，用鼠标右键单击 **getXMLString()** 方法并选择**企业 Bean > 提升至远程接口**
6. 在"概览"视图中，用鼠标右键单击 **getXMLString(boolean tcdata)** 方法并选择**企业 Bean > 提升至远程接口**
7. 在"概览"视图中，用鼠标右键单击 **parseXMLElement(org.w3c.dom.Element argElement)** 方法并选择**企业 Bean > 提升至远程接口**
8. 在"概览"视图中，用鼠标右键单击 **createNewVersion (Long argNewTradingId)** 方法并选择**企业 Bean > 提升至远程接口**
9. 在"概览"视图中，用鼠标右键单击 **markForDelete()** 方法并选择**企业 Bean > 提升至远程接口**
10. 保存更改。

为 MySubTC 创建访问 bean: 要创建访问 bean，请执行以下操作:

1. 在"J2EE 层次结构"视图中，展开 **EJB 模块**，然后用鼠标右键单击 **Enablement-RelationshipManagementData** 并选择**新建 > 访问 Bean**。"添加访问 Bean"窗口打开。
2. 选择**复制帮手**并单击**下一步**。
3. 选择 ***MySubTC*** bean 并单击**下一步**。

4. 从构造函数方法下拉列表中，选择 **findByPrimaryKey(com.ibm.commerce.contract.objects.*MySubTC*Key)** 作为构造函数方法。
5. 选择“属性帮手”部分的所有属性。
6. 单击**完成**。
7. 保存工作。

添加新的字符串转换器: 您必须添加新的字符串转换器，如下:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块 > Enablement-RelationshipManagementData > ejbModule > META-INF**。
2. 双击 **ibm-ejb-access-bean.xmi** 文件。
3. 找到关于 MySubTC 的部分。
4. 对每个 copyHelperProperties 元素，添加以下属性:

   ```
   converterClassName="com.ibm.commerce.base.objects.WCSStringConverter"
   ```

5. 保存工作。
6. 下一步，您必须以如下方式为 MySubTC 重新生成访问 bean:
   a. 用鼠标右键单击 **Enablement-RelationshipManagementData** 并选择**访问 bean > 重新生成访问 Bean**。
   b. 选择 **MySubTC** 并单击**完成**。

生成部署代码: 您必须为 MySubTC bean 和 TermCondition bean 生成部署代码，方法如下:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**。
2. 用鼠标右键单击 **Enablement-RelationshipManagementData** 并选择**生成 > 部署和 RMIC 代码**。
3. 选择 **MySubTC** 并单击**完成**。
4. 用鼠标右键单击 **Enablement-RelationshipManagementData** 并选择**生成 > 部署和 RMIC 代码**。
5. 单击**全部选中**然后单击**完成**。

注: 技术上来讲，您必须为父 bean（TermCondition bean）和所有兄弟 bean（RelationshipManagementData 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。出于简单，上述指示信息选定了此项目中的所有 bean。

覆盖 validateContract 任务命令中的方法: 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。

有关如何作此修改的详细信息，请参阅第 141 页的『定制现有的任务命令』。

创建新部署命令: 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:

1. 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');
   ```

2. 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。

3. 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。出于此目的对 WebSphere 贸易加速器进行更新包括以下步骤:

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需的信息。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:

- 400 AIX Linux Solaris *WC_installdir*/samples/contract/Extensions.js
- Windows *WC_installdir*\samples\contract\Extensions.js
- Developer *WCDE_installdir*\samples\contract\Extensions.js

为了使用此样本文件，请将其复制到以下目录:

- 400 *WAS_userdir*/installedApps/*cell_name*/WC_*instanceName*.ear/CommerceAccelerator.war/tools/contract
- AIX Linux Solaris *WAS_installdir*/installedApps/*cell_name*/WC_*instanceName*.ear/CommerceAccelerator.war/tools/contract
- Windows *WAS_installdir*\installedApps*cell_name*\WC_*instanceName*.ear\CommerceAccelerator.war\tools\contract

- Developer *workspace_dir*\CommerceAccelerator\Web Content\tools\contract

其中 *instanceName* 是 WebSphere Commerce 实例的名称，*cell_name* 是 WebSphere Application Server 单元名称。

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 XSD 文件所作扩展相一致的方式完成。这在以下代码片段中演示:

```
function submitMyTC(contract) {

   var tcModel = get("ContractMyTCModel");

   if (tcModel != null) {

      var myTC = new Object();
 myTC.attr1 = tcModel.attr1;
 myTC.attr2 = tcModel.attr2;

 myTC.ProductSetPolicyRef = new Object();
 myTC.ProductSetPolicyRef.policyName = tcModel.policyList[
      tcModel.selectedPolicyIndex].policyName;
 myTC.ProductSetPolicyRef.StoreRef = new Object();
 myTC.ProductSetPolicyRef.StoreRef.name = tcModel.policyList[
      tcModel.selectedPolicyIndex].storeIdentity;
 myTC.ProductSetPolicyRef.StoreRef.Owner = new Object();
 myTC.ProductSetPolicyRef.StoreRef.Owner = 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";
 }
```

```
 contract.MySubTC = 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需的信息。可以参阅以下样本文件作为参考:

- 400 AIX Linux Solaris *WC_installdir*/samples/contract/ContractMyTC.jsp
- Windows *WC_installdir*\samples\contract\ContractMyTC.jsp
- Developer *WCDE_installdir*\samples\contract\ContractMyTC.jsp

为了使用此样本文件，请将其复制到以下目录:

- 400 *WAS_userdir*/installedApps/*cell_name*/WC_*instanceName*.ear/CommerceAccelerator.war/javascript/tools/contract
- AIX Linux Solaris *WAS_installdir*/installedApps/*cell_name*/WC_*instanceName*.ear/CommerceAccelerator.war/ javascript/tools/contract
- Windows *WAS_installdir*\installedApps*cell_name*\WC_*instanceName*.ear\CommerceAccelerator.war\ javascript\tools\contract
- Developer *workspace_dir*\CommerceAccelerator\Web Content\tools\contract

其中 *instanceName* 是 WebSphere Commerce 实例的名称，*cell_name* 是 WebSphere Application Server 单元的名称。

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:

- 400 *WAS_userdir*/installedApps/*cell_name*/WC_*instanceName*.ear/properties/com/ibm/commerce/tools/contract/properties/ ContractRB_*locale*.properties
- AIX Linux Solaris *WAS_installdir*/installedApps/*cell_name*/WC_*instanceName*.ear/properties/ com/ibm/commerce/tools/contract/properties/ ContractRB_*locale*.properties
- Windows *WAS_installdir*\installedApps*cell_name*\ WC_*instanceName*.ear\properties\com\ibm\commerce\tools\contract\properties\ ContractRB_*locale*.properties
- Developer *workspace_dir*\WebSphereCommerceServer\properties\com\ibm\commerce\tools\contract\properties\ContractRB_*locale*.properties

其中 *instanceName* 是 WebSphere Commerce 实例的名称，*cell_name* 是 WebSphere Application Server 单元的名称。

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。

- 400 *WC_installdir*/xml/tools/contract/ContractNotebook.xml

- AIX Linux Solaris *WC_installdir*/xml/tools/contract/ContractNotebook.xml
- Windows *WC_installdir*\xml\tools\contract\ContractNotebook.xml
- Developer *WCDE_installdir*\Commerce\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 文件中的相关部分显示如下:

```
<MySubTC attr1="abc" attr2="123">
       <ProductSetPolicyRef policyName = "Product Set 1">
           <StoreRef name = "StoreGroup1">
               <Owner>
                   <OrganizationRef distinguishName = "o=Root Organization"/>
               </Owner>
           </StoreRef>
       </ProductSetPolicyRef>
   </MySubTC>
```

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包含引用新业务策略的条款和条件的合同。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 生产和开发联机帮助。

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 XSD 文件以反映新的 RebateTC。
c. 为 RebateTC 创建新的企业 Bean。
d. 更新 WebSphere 贸易加速器以反映新的 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
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 56 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，必须创建新业务策略类型。创建新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数据包中。

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。

命令逻辑依赖于客户如何选取货物的特定实现。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该新业务策略。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。

以下各表显示执行更新之后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	5Dollar 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 null。

表 5. 更新 POLICYCMD 表

	列名		
	POLICY _ID	BUSINESS CMDCLASS	PROPERTIES
样本数据	301	com.mycompany.mybusinesspolicycommands.CalculateRebateCmdImpl	null

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 XSD 文件以反映新的 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	null

在合同文档类型定义中注册折扣条款和条件: 要让折扣条款和条件在合同中可用，您必须创建新的 XSD 文件，该文件定义新的条款和条件。您还必须更新 Package.xsd 文件来包含该新 XSD 文件。

要创建新的 XSD 文件，请执行以下操作:

1. 导航到以下目录:
   - Windows *WC_installdir*\xml\trading\xsd
   - AIX Solaris Linux *WC_installdir*/xml/trading/xsd
   - 400 *WC_userdir*/instances/*instanceName*/xml/trading/xsd

   其中 *instanceName* 是 WebSphere Commerce 实例的名称。
2. 在此目录下创建新 XSD 文件。以下显示用于示例 RebateTC 的 XSD。文件为 RebateCustomizedBuyerContract.xsd:

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.ibm.com/WebSphereCommerce"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:wc="http://www.ibm.com/WebSphereCommerce"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">
```

```
<!-- include basic trading agreement xsd -->

<include schemaLocation="BuyerContract.xsd" />
<complexType name="RebateTCType">
          <complexContent>
          <extension base="wc:TermConditionType"/>
          </complexContent>
      </complexType>
  <element name="RebateTC" substitutionGroup="wc:AbstractCustomizedTC">
      <complexType>
          <complexContent>
              <extension base="wc:RebateTCType">
                  <sequence>
                      <element ref="wc:RebatePolicyRef"/>
                  </sequence>
              </extension>
          </complexContent>
      </complexType>
  </element>

  <element name="RebatePolicyRef" type="wc:BusinessPolicyRef" />

</schema>
```

3. 保存该新文件。

下一步，您必须更新 Package.xsd 来除去 BuyerContract.xsd 并包含 RebateCustomizedBuyerContract.xsd 来取代前者，如下所示:

1. 导航到以下目录:
   - Windows *WC_installdir*\xml\trading\xsd
   - AIX Solaris Linux *WC_installdir*/xml/trading/xsd
   - 400 *WC_userdir*/instances/*instanceName*/xml/trading/xsd

   其中 *instanceName* 是 WebSphere Commerce 实例的名称。
2. 在文本编辑器中打开 Package.xsd 文件。
3. 找到有关 BuyerContract.xsd 的部分并按以下方式修改:

```
<!--include schemaLocation="BuyerContract.xsd"/-->
<include schemaLocation="RebateCustomizedBuyerContract.xsd"/>
```

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 158 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
EJB 项目	Enablement-RelationshipManagementData
Bean 类型	带有受容器管理的持久性字段的实体 bean
Bean 名称	RebateTC
数据包	com.ibm.commerce.contract.objects
Bean 超类	TermCondition
Bean 类	RebateTCBean

在此新 bean 中，创建三个用于以下值的 CMP 字段:

- 折扣代码标识
- 金额
- 货币

更新 WebSphere 贸易加速器，以包含 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 166 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 生产和开发联机帮助中描述了每种创建新合同的方法。

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

表 10. 对 POLICYTC 表的更新 (续)

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 开发环境

本章介绍用于定制 WebSphere Commerce 应用程序的主要开发工具。

典型开发环境

Business 对于创建要用于 WebSphere Commerce Business Edition 的定制代码，推荐的开发软件包是 WebSphere Commerce Studio, Business Developer Edition 产品。Professional 对于创建要用于 WebSphere Commerce Professional Edition 的定制代码，推荐的开发软件包是 WebSphere Commerce Studio, Professional Developer Edition 产品。Express 对于创建要用于 WebSphere Commerce - Express 的定制代码，推荐的开发软件包是 WebSphere Commerce - Express Developer Edition 产品。所有这些软件包都包含您创建定制代码和执行 Web 开发任务所需要的工具。总体而言，本书将这些产品统称为 WebSphere Commerce 开发环境。

WebSphere Commerce 开发环境共有四个主要组件:

1. 用于 WebSphere Studio Application Developer 内的 WebSphere Commerce 工作空间
2. 开发数据库
3. 文件系统有用资源
4. WebSphere Studio Application Developer 的 WebSphere Commerce 插件

使用此开发环境，您可以创建定制代码并在 WebSphere Test Environment 上下文中测试定制代码。

要在开发环境中创建商店，您只需启动 WebSphere Studio Application Developer 中本地定义的 WebSphereCommerceServer 测试服务器上运行的管理控制台，并使用该工具来发布基于样本商店之一的商店。或者，您也可创建您自己的商店。

在提供的 WebSphere Commerce 工作空间外，WebSphere Commerce 开发环境提供额外的工具和插件。提供以下插件:

- 配置管理器插件，协助管理 WebSphere Commerce（和 WebSphere Commerce Payments）实例。

- WebSphere Commerce 联机帮助插件允许您从 WebSphere Studio Application Developer 内访问 WebSphere Commerce 联机帮助。此外，使用此插件，在运行 WebSphere Commerce 工具（例如管理控制台）时，按下 F1 可启动 WebSphere Commerce 上下文敏感联机帮助。
- WebSphere Commerce API 参考信息插件允许您从 WebSphere Studio Application Developer 内访问 WebSphere Commerce API 参考信息。

还提供了 WebSphere Commerce 企业 bean 转换工具。使用此工具，您可以使用不同于您的目标生产数据库的开发数据库来开发企业 bean。例如，此工具允许您将本地 DB2 数据库用于开发，而生产数据库（甚或是 Oracle 数据库）可以在 iSeries 平台上。

WebSphere Studio Application Developer

WebSphere Commerce 开发环境软件包包含 WebSphere Studio Application Developer，它是 IBM 的核心开发环境。它通过在其类中提供最佳做法、模板、代码生成和最全面综合的开发环境来协助您优化并简化 Java2 Enterprise Edition（J2EE）和 Web 服务开发。此功能繁多的集成开发环境（IDE）将对 Java 组件、企业 bean、servlet、JSP 文件、HTML、XML 和 Web 服务的集成支持全部包含在一个开发环境中。

在许多其它激动人心的功能部件中，还包含允许您迅速生成测试客户机的本地测试工具。它还包含一个完整的 WebSphere Application Server 测试环境，允许您测试在本地环境中对您的代码进行端到端的测试。

iSeries 开发环境

400 简而言之，为 iSeries 创建定制代码不需要特殊的开发环境。开发工作站是根据 Business Professional *《WebSphere Commerce Studio 安装指南》*或 Express *《WebSphere Commerce - Express Developer Edition 安装指南》*中概括的相同过程设置的。使用的本地开发数据库应当为 DB2。使用此配置，可以使用本地 DB2 数据库和本地 WebSphereCommerceServer 测试服务器来创建并测试定制代码。

当测试已确定定制代码在测试服务器的上下文中可以令人满意地工作之后，您必须将其部署到运行在 iSeries 平台上的目标 WebSphere Commerce Server 上。考虑到 Windows 和 iSeries 平台上数据库之间的差异，还提供了 WebSphere Commerce 企业 bean 转换工具。第 185 页的『WebSphere Commerce 企业 bean 转换工具概述』中提供关于此工具的更多信息。

当生产环境使用 Oracle 数据库时将本地 DB2 数据库用于开发

开发者可以在他们的开发机器上使用本地 DB2 数据库，甚至生产环境的数据库是 Oracle 数据库。在此情况下，WebSphere Commerce 企业 bean 转换工具用于将 bean 的元数据从 DB2 格式转换为 Oracle 格式。『WebSphere Commerce 企业 bean 转换工具概述』中提供关于此工具的更多信息。

WebSphere Commerce 企业 bean 转换工具概述

常规而言，在两种情况中需要用到企业 bean 转换工具:

- 如果目标生产环境运行在 iSeries 平台上。
- 如果目标生产数据库是 Oracle 数据库而开发机器使用本地 DB2 数据库。

该特定与 WebSphere Commerce 的工具允许您针对一种数据库类型进行开发，然后将其部署到另一数据库类型中。使用此工具，企业 bean 的元数据转换为适用于目标数据库的格式和信息，也使用该新的元数据生成部署代码。请参阅第 190 页的『使用转换创建 EJB JAR 文件』，获取有关如何使用此工具的分步式详细信息。

开发环境中的支付选项

在先前版本的 WebSphere Commerce 开发环境中，提供了测试支付方法，允许开发者在测试环境商店中完成购买而不需要调用外部的远程支付提供程序。现在，在 WebSphere Commerce 开发环境 V5.5中，WebSphere Commerce Payments 组件可以运行在测试环境中。

这意味着您现在可以选择使用本地 WebSphere Commerce Payments 实例或可配置您的 WebSphere Commerce 开发实例来使用远程 WebSphere Commerce Payments 实例。

缺省情况下，将在您安装 WebSphere Commerce 开发环境时创建本地 WebSphere Commerce Payments 实例。此外，如果发布样本商店时此实例正在运行，商店将自动配置为使用该本地 WebSphere Commerce Payments 实例。

有关配置支付选项的信息在《*WebSphere Commerce 开发环境安装指南*》中提供。

第 9 章 部署详细信息

当您在 WebSphere Commerce 开发环境中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 之外运行的目标 WebSphere Commerce Server。此目标 WebSphere Commerce Server 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。

本章描述将定制代码部署到运行在 WebSphere Test Environment 之外的目标 WebSphere Commerce Server 所必需的步骤。

本章分为几部分，它们描述了如何部署定制应用程序可能包含的各种类型的代码。描述了以下任务:

- 部署企业 bean
- 部署命令和数据 bean
- 部署商店有用资源
- 更新目标数据库

后面各部分将更详细地描述上面的每个任务。

对部署步骤的用户许可权要求

AIX Linux Solaris 您应当使用在为 WebSphere Commerce 安装过程做准备时创建的非 root 用户标识，在目标 WebSphere Commerce Server 上执行所有部署步骤（访问控制更新除外）。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。

关于执行访问控制更新所需的用户许可权的信息，请参阅《*WebSphere Commerce 安全性指南*》中的“将您的 XML 更改装入数据库”主题。

增量部署

本章中描述的部署的类型为*增量*部署。在一个增量部署中，您必须已经在目标 WebSphere Commerce 服务器上安装了 WebSphere Commerce 企业应用程序。然后在部署过程中，您只将以下有用资源（命令、数据 bean、企业 bean 及其它）部署到现有的企业应用程序中。

要在您的目标 WebSphere Commerce Server 上创建初始企业应用程序，您必须安装 WebSphere Commerce 并使用配置管理器来创建您的 WebSphere Commerce 实例（并由此创建 WebSphere Commerce 企业应用程序）。

部署企业 bean

本部分描述如何部署企业 bean。这些 bean 可能是为电子交易应用程序创建的新企业 bean，或者可能是您已修改的 WebSphere Commerce 实体 bean。无论哪种情况，部署步骤基本上是相同的。

理解 WebSphere Commerce 应用程序的部署过程的重要因素之一是理解用于定制的 WebSphere Commerce 代码的封装方案。特别地，您不需要在 WebSphere Commerce 工作空间中创建新的 EJB 项目。把新的企业 bean 放入 WebSphereCommerceServerExtensionsData 项目中，且修改后的 WebSphere Commerce 实体 bean 的定制代码保留在原始 WebSphere Commerce EJB 项目中。

在这个部署过程中有两个主要步骤:

- 创建 EJB JAR 文件
- 更新目标 WebSphere Commerce Server 上的 EJB JAR 文件

创建 EJB JAR 文件

有两个不同方法用于创建 EJB JAR 文件，这取决于部署方案，如下所示:

- 如果您在创建一个 EJB JAR 文件，该文件将要部署至使用与您的开发环境相同类型的数据库的目标 WebSphere Commerce Server，请遵照第 188 页的『不使用转换创建 EJB JAR 文件』中包含的指示信息。
- 如果您在创建一个 EJB JAR 文件，该文件将要部署至使用与您的开发环境不同类型的数据库的目标 WebSphere Commerce Server，请遵照第 190 页的『使用转换创建 EJB JAR 文件』中包含的指示信息。

不使用转换创建 EJB JAR 文件

要创建 EJB JAR 文件，请执行以下操作:

1. 打开 WebSphere Commerce 开发环境（**开始 > 程序 > IBM WebSphere Commerce 开发环境 > WebSphere Commerce 开发环境**）并切换至 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
2. 展开包含您正在部署的 bean 的 EJB 项目，如下:
   - 如果已创建新的企业 bean，请展开 **WebSphereCommerceServerExtensionsData** EJB 项目。

- 如果您已修改 WebSphere Commerce 实体 bean，请展开包含已修改 bean 的项目。例如，如果您已修改用户 bean，那么请展开 **Member-MemberManagementData** EJB 项目。

3. 双击 **EJB 部署描述符**。
4. 在选择“概述”选项卡的情况下，滚动至窗格底部以找到 **WebSphere 绑定**部分。
5. 在**数据源 JNDI** 名称字段中，输入目标 WebSphere Commerce Server 的数据源 JNDI 名称。以下是值的示例:

   DB2 `jdbc/WebSphere Commerce DB2 DataSource demo`
   其中目标 WebSphere Commerce Server 使用 DB2 数据库，并且 WebSphere Commerce 实例名称是“demo”

   Oracle `jdbc/WebSphere Commerce Oracle DataSource demo`
   其中目标 WebSphere Commerce Server 使用 Oracle 数据库，WebSphere Commerce 实例名称是“demo”。

数据源 JNDI 名称的值是通过将“jdbc/”添加到目标 WebSphere Commerce Server 的数据源名称来创建的。可通过打开目标 WebSphere Commerce Server 上的 *instanceName*.xml 文件并在该文件中搜索 `DatasourceName=` 来验证数据源名称。

6. 保存部署描述符更改（Ctrl+S）。
7. 在 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图中，用鼠标右键单击 EJB 项目（WebSphereCommerceServerExtensionsData 或者包含已修改的 WebSphere Commerce 实体 bean 的项目），并选择**导出**。
   “导出”向导打开。
8. 在“导出”向导中，请执行以下操作:
   a. 选择 **EJB JAR 文件**并单击**下一步**。
   b. Business Professional **您想要导出什么资源?** 的值是以 EJB 项目的名称预填充的。

      Express EJB 项目名称是预填充的。
      保留该值不变。

   c. Business Professional 在**您要将资源导出到哪里?** 字段中，输入要使用的全限定 JAR 文件名。

      Express 对于目的地，输入要使用的全限定 JAR 文件名。
      例如，输入 `C:\ExportTemp\`*`JarFileName`*`.jar`，其中 *JarFileName* 是 JAR 文件的名称。如果已创建新的企业 bean，则应输入

yourDir\WebSphereCommerceServerExtensionsData.jar。如果您已修改现有的 WebSphere Commerce 公共实体 bean，您必须为这一 EJB 组使用预定义的 JAR 文件名。例如，如果您在 Member-MemberManagementData EJB 模块中进行了修改，则输入 *yourDir*\Member-MemberManagementData.jar。

d. 确保未选**导出源文件**项。

e. 单击**完成**。

9. 创建 JAR 文件后，打开 EJB 部署描述符并恢复步骤 5 中进行的修改，以便返回至本地测试服务器所需的设置。保存更改。

使用转换创建 EJB JAR 文件

要转换元数据并创建 EJB JAR 文件，执行以下操作:

1. 打开 WebSphere Commerce 开发环境（**开始 > 程序 > IBM WebSphere Commerce 开发环境 > WebSphere Commerce 开发环境**）并切换至

   Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。

2. 展开包含您正在部署的 bean 的 EJB 项目，如下:

   - 如果已创建新的企业 bean，请展开 **WebSphereCommerceServerExtensionsData** 项目。
   - 如果您已修改 WebSphere Commerce 实体 bean，请展开包含已修改 bean 的项目。例如，如果您已修改用户 bean，那么请展开 **Member-MemberManagementData** EJB 项目。

3. 双击 **EJB 部署描述符**。

4. 在选择“概述”选项卡的情况下，滚动至窗格底部以找到 **WebSphere 绑定**部分。

5. 在**数据源 JNDI** 名称字段中，输入目标 WebSphere Commerce Server 的数据源 JNDI 名称。以下是值的示例:

   DB2 `jdbc/WebSphere Commerce DB2 DataSource demo`

   其中目标 WebSphere Commerce Server 使用 DB2 数据库，WebSphere Commerce 实例名称是“demo”

   Oracle `jdbc/WebSphere Commerce Oracle DataSource demo`

   其中目标 WebSphere Commerce Server 使用 Oracle 数据库，WebSphere Commerce 实例名称是“demo”。

数据源 JNDI 名称的值是通过将“jdbc/”添加到目标 WebSphere Commerce Server 的数据源名称来创建的。可通过打开目标 WebSphere Commerce Server 上的 *instanceName*.xml 文件并在该文件中搜索 DatasourceName= 来验证数据源名称。

6. 保存部署描述符更改（Ctrl+S）。
7. 关闭 WebSphere Studio Application Developer。
8. 在命令提示符下，导航到以下目录:

   WCStudio_installdir\Commerce\bin
9. 输入以下命令:

   ejbDeploy.bat *projName outputJarName mapFile workspace_dir eclipseDir ejbDeployXmlFile WCStudio_commercedir*

   其中:

   - *projName* 是要转换的 EJB 项目的名称
   - *outputJarName* 是输出 JAR 文件的全限定名称。注意，您应当使用已存在于 WebSphere Commerce 企业应用程序中的该 JAR 文件名称。以下是一个示例:

     C:\ExportTemp\WebSphereCommerceServerExtensionsData.jar
   - *mapFile* 是映射文件的名称。对此的示例包含以下文件:

     WCDE_installdir\Commerce\properties\com\ibm\commerce\metadata\conversion\oracle.mapping
     WCDE_installdir\Commerce\properties\com\ibm\commerce\metadata\conversion\as400.mapping
   - *workspace_dir* 是您当前开发工作空间的目录。
   - *eclipseDir* 是 eclipse 目录的路径。缺省情况下，它是

     WCDE_installdir\Studio5
   - *ejbDeployXmlFile* 是全限定 ejbDeploy.xml 文件。缺省情况下，它是

     WCDE_installdir\Commerce\xml\ejbDeploy.xml

     注: 运行此命令时，您可能看到一些错误，它们指示无法展开一些文件。这没有什么关系。
   - *WCStudio_commercedir* 是在安装 WebSphere Commerce 开发环境时创建的 Commerce 目录的路径。缺省情况下，它是

     WCDE_installdir\Commerce

     或更明确的为

     C:\WebSphere\CommerceDev55

   以下是此命令与所有指定值的用法示例:

   ```
   ejbDeploy.bat WebSphereCommerceServerExtensionsData
   WebSphereCommerceServerExtensionsData.jar
   C:\WebSphere\CommerceStudio55\Commerce\properties\com\ibm\
      commerce\metadata\conversion\oracle.mapping
   ```

```
C:\WebSphere\workspace_db2 C:\WebSphere\Studio5
C:\WebSphere\CommerceStudio55\Commerce\xml\ejbDeploy.xml
C:\WebSphere\CommerceStudio55\Commerce
```

注意，断行只是出于显示目的。

10. 打开 WebSphere Commerce 开发环境，并打开 EJB 部署描述符，且恢复步骤 5 中进行的修改，使之返回到本地测试服务器所需的设置。保存更改。
11. 如果您正在收集所有资源以部署到单一目录中（例如: C:\ExportTemp），请复制新创建的 EJB JAR 文件到该目录。

更新目标 WebSphere Commerce Server 上的 EJB JAR 文件

下一步是把新创建的 EJB JAR 文件复制到目标 WebSphere Commerce Server 的相应位置。

要在目标 WebSphere Commerce Server 上更新 EJB JAR 文件，请执行以下操作:

1. 停止 WebSphere Application Server 中运行着的 WebSphere Commerce 实例。关于如何停止此实例的详细信息，请参阅对应于您的平台和数据库的《*WebSphere Commerce 安装指南*》。
2. 在 WebSphere Commerce 实例中找到原始 EJB JAR 文件。例如，找到以下文件:
   - 400 *WAS_userdir*/installedApps/*WAS_node_name*/WC_*instance_name*.ear/*JarFileName*.jar
   - AIX Linux Solaris *WAS_installdir*/installedApps/*cellName*/WC_*instance_name*.ear/*JarFileName*.jar
   - Windows *WAS_installdir*\installedApps*cellName*\WC_*instance_name*.ear*JarFileName*.jar

   其中
   - *instance_name* 是 WebSphere Commerce 实例的名称。
   - 400 *WAS_node_name* 代表安装了 WebSphere Application Server 产品的 iSeries 系统。
   - *JarFileName* 是包含定制代码的 JAR 文件的名称
3. 对原始 EJB JAR 文件制作副本。
4. 将新的 EJB JAR 文件从开发机器复制到步骤 2 中的位置。
5. 如果您已修改 EJB 部署描述符，请执行以下操作:
   a. 定位该 WebSphere Application Server 单元的部署资源库（META-INF 目录）。该目录通常采用如下格式:

- 400 *WAS_userdir*/config/cells/*WAS_node_name*/applications/WC_*instance_name*.ear/deployments/WC_*instance_name*/*EJBModuleName*.jar/META-INF
- AIX Linux Solaris *WAS_installdir*/config/cells/*cellName*/applications/WC_*instance_name*.ear/deployments/WC_*instance_name*/*EJBModuleName*.jar/META-INF
- Windows *WAS_installdir*\config\cells*cellName*\applications\WC_*instance_name*.ear\deployments\WC_*instance_name**EJBModuleName*.jar\META-INF

其中

- *instance_name* 是 WebSphere Commerce 实例的名称。
- 400 *WAS_node_name* 代表安装了 WebSphere Application Server 产品的 iSeries 系统。
- *EJBModuleName* 是已修改的 EJB 模块的名称

以下为 META-INF 目录的特定于平台的示例:

- 400 *WAS_userdir*/config/cells/myNode/applications/WC_demo.ear/deployments/WC_demo/Member-MemberManagementData.jar/META-INF
- AIX Linux Solaris *WAS_installdir*/config/cells/myCell/applications/WC_demo.ear/deployments/WC_demo/Member-MemberManagementData.jar/META-INF
- Windows *WAS_installdir*\config\cells\myCell\applications\WC_demo.ear\deployments\WC_demo\Member-MemberManagementData.jar\META-INF

其中

- myCell 是 WebSphere Application Server 单元的名称
- demo 是 WebSphere Commerce 实例的名称
- Member-MemberManagementData 是已修改的 EJB 模块的名称
- 400 myNode 是 WebSphere Application Server 节点的名称

b. 备份前述目录中的所有文件。

c. 使用工具打开 *JarFileName*.jar 文件并查看其内容。

d. 把 META-INF 目录的内容从 *JarFileName*.jar 文件解压缩到步骤 5a 中的目录。

6. 如对应于您的平台和数据库的《*WebSphere Commerce 安装指南*》中所述，重新启动 WebSphere Commerce 实例。

部署命令和数据 bean

在执行以下任何一个任务时，定制代码应封装到 WebSphereCommerceServerExtensionsLogic 项目中:

- 创建新的命令
- 创建新的数据 bean
- 修改现有的 WebSphere Commerce 命令
- 修改现有的 WebSphere Commerce 数据 bean

不允许直接修改命令或数据 bean 的现有类（或包含现有类的项目）。

部署定制命令和数据 bean 涉及以下步骤:

- 创建 JAR 文件
- 更新目标 WebSphere Commerce Server 上的 JAR 文件

后面各部分将更详细地描述上述步骤。

如果您的定制代码需要对命令注册表或新的访问控制策略进行更新或进行其它数据库更新，请务必参考第 197 页的『更新目标数据库』。

创建 JAR 文件

要创建 JAR 文件，请执行以下操作:

1. 打开 WebSphere Commerce 开发环境（**开始 > 程序 > IBM WebSphere Commerce 开发环境 > WebSphere Commerce 开发环境**）并切换至 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
2. 用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 项目并选择**导出**。
   “导出”向导打开。
3. 在“导出”向导中，请执行以下操作:
   a. 选择 **JAR 文件**并单击**下一步**。
   b. **选择要导出的资源?** 左下窗格是以 EJB 项目的名称预先填充的。保留此值不变。
   c. **选择要导出的资源?** 右下窗格确保仅选择了以下资源:

- .classpath
- .project
- .serverPreference

d. 确保已选择**导出已生成的类文件和资源**。

e. 不要选择**导出 Java 源文件和资源**。

f. 在**选择导出目的地**字段中，输入要使用的全限定 JAR 文件名。例如，输入 `C:\ExportTemp\WebSphereCommerceServerExtensionsLogic.jar`。
请注意，JAR 文件名必须为 WebSphereCommerceServerExtensionsLogic.jar。

g. 单击**完成**。

一旦 JAR 文件已成功创建，下一步就是把文件传输到目标 WebSphere Commerce Server 上的合适位置。

更新目标 WebSphere Commerce Server 上的 JAR 文件

下一步是把新创建的 JAR 文件复制到目标 WebSphere Commerce Server 上的相应位置。

要更新 JAR 文件，请执行以下操作:

1. 停止 WebSphere Application Server 中运行着的 WebSphere Commerce 实例。关于如何停止此实例的详细信息，请参阅对应于您的平台和数据库的*《WebSphere Commerce 安装指南》*。
2. 在 WebSphere Commerce 实例中找到原始 EJB JAR 文件。例如，找到以下文件:
   - 400 *WAS_userdir*/installedApps/*WAS_node_name*/WC_*instance_name*.ear/WebSphereCommerceServerExtensionsLogic.jar
   - AIX Linux Solaris *WAS_installdir*/installedApps/*cellName*/WC_*instance_name*.ear/WebSphereCommerceServerExtensionsLogic.jar
   - Windows *WAS_installdir*\installedApps*cellName*\WC_*instance_name*.ear\WebSphereCommerceServerExtensionsLogic.jar

   其中
   - *instance_name* 是 WebSphere Commerce 实例的名称。
   - 400 *WAS_node_name* 代表安装了 WebSphere Application Server 产品的 iSeries 系统。
3. 将原始 JAR 文件复制到一个备份位置。
4. 把 JAR 文件从开发机复制到步骤 2 的位置。

5. 如对应于您的平台和数据库的*《WebSphere Commerce 安装指南》*中所述，重新启动 WebSphere Commerce 实例。

部署商店有用资源

商店有用资源包括如下有用资源:

- JSP 模板
- HTML 文件
- 图像文件
- XML 文件
- 属性文件和资源束

这些有用资源必须从开发环境部署到目标 WebSphere Commerce Server。这包括以下步骤:

- 从 WebSphere Studio Application Developer 导出商店有用资源。
- 把有用资源传送到目标 WebSphere Commerce Server

后面各部分将更详细地描述上述步骤。

导出商店有用资源

要从开发环境导出商店有用资源，请执行以下操作:

1. 打开 WebSphere Commerce 开发环境（**开始 > 程序 > IBM WebSphere Commerce 开发环境 > WebSphere Commerce 开发环境**）并切换至 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
2. 展开 **Stores** 文件夹。
3. 用鼠标右键单击 **Web Content** 文件夹并选择**导出**。
   “导出”向导打开。
4. 在“导出”向导中，请执行以下操作:
   a. 选择**文件系统**并单击**下一步**。
   b. 选择所有您要部署的资源。即选择所有 JSP 模板、HTML 文件、图像、属性文件和其他需要部署的商店有用资源。
   c. 选择**为选定文件创建目录结构**。
   d. 在**目录**字段，输入将放置这些资源的临时目录。例如，输入 `C:\ExportTemp\StoreAssets`
   e. 单击**完成**。

下一步是把这些资源复制到目标 WebSphere Commerce Server 上的相应位置。

传送商店有用资源

要把商店有用资源从开发机传送到目标 WebSphere Commerce Server，请执行以下操作:

1. 根据您要部署的资源类型和您的特定配置的详细信息，您可能需要停止在 WebSphere Application Server 中运行的 WebSphere Commerce 实例。如果不能确定您的特殊部署方案是否需要重新启动，您应该停止该实例。关于如何停止此实例的详细信息，请参阅对应于您的平台和数据库的*《WebSphere Commerce 安装指南》*。
2. 在目标机器上找到 Stores.war 目录。以下是此目录的示例:
   - 400 *WAS_userdir*/installedApps/*WAS_node_name*/WC_*instance_name*.ear/Stores.war
   - AIX Linux Solaris *WAS_installdir*/installedApps/*cellName*/WC_*instance_name*.ear/Stores.war
   - Windows *WAS_installdir*\installedApps*cellName*\ WC_*instance_name*.ear\Stores.war

   其中
   - *instance_name* 是 WebSphere Commerce 实例的名称。
   - 400 *WAS_node_name* 代表安装了 WebSphere Application Server 产品的 iSeries 系统。
3. 把在“导出商店有用资源”中导出的文件复制到 Stores.war 目录。
4. 如果您先前停止了 WebSphere Commerce 实例，请启动该实例。关于如何启动此实例的详细信息，请参阅对应于您的平台和数据库的*《WebSphere Commerce 安装指南》*。

更新目标数据库

由于您的目标 WebSphere Commerce Server 使用的数据库不同于开发机使用的数据库时，您必须在目标 WebSphere Commerce Server 使用的数据库上执行所有对开发数据库执行的更新。这包含所有更新，这些更新包含: 新的或已修改的命令或视图的注册，已创建的其它表以及为任何已创建的新资源所创建的访问控制策略。

400 您负责让某个实用程序运行 SQL 语句。有一种方法可获得该种程序: 使用 IBM iSeries Access for Windows。要打开此实用程序，请执行以下操作:

1. 打开 **iSeries 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 实例用户概要文件和密码。这确保 WebSphere Commerce 实例用户概要文件拥有所有创建的新表。
3. 在左侧的面板中，展开 iSeries 系统，然后展开 **DATABASES**。用鼠标右键单击“关系数据库”并选择从下拉列表列表中选择**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**选项下的 **JDBC 设置**选项来设置缺省模式。

访问控制更新

当访问控制信息包含在数据库中时，这是一种特殊类型的信息，它不一定是直接从开发环境复制到目标环境。特别地，在开发环境中，您可决定使用非常自由的访问控制策略，这一策略不适合于生产（或下一级别的测试）环境。例如，在开发环境范围内可以设置新命令的策略，这样所有的用户都可以执行这一命令，但在其它地方这可能是不合适的。

因此，在把访问控制信息从开发环境复制到目标环境前，您必须考虑新环境中的访问控制需求，并相应地调整策略。

关于装入访问控制策略的信息（包括适用于各种平台的命令语法和目录许可权需求），请参阅《*WebSphere Commerce 安全性指南*》。

第 4 部分 教程

以下教程设计用于介绍与为 WebSphere Commerce 应用程序创建定制代码相关的各种任务。与开发相关的步骤将在 WebSphere Commerce Business Edition、WebSphere Commerce - Express 开发环境或 WebSphere Commerce Studio, Professional Developer Edition 中执行。部署步骤将在 Windows 2000 上运行的 WebSphere Commerce（无论是 Business Edition 还是 Professiona Edition, 这与您的开发环境相对应）上执行。

第 10 章 教程: 创建新的业务逻辑

本教程的设计是为了向您说明创建新的业务逻辑所涉及的步骤。创建的有用资源类型包括新视图、新控制器命令、新任务命令、新数据 bean 以及新实体 bean。本教程使用的方案是开发小型接口以使用户能修改奖金点数的余额。它仅用于演示目的，且不反映构建忠实服务程序所需的逻辑。但从本教程中，您将了解创建每种以前列出的代码有用资源所通用的开发步骤。

此教程分为以下子任务:

1. 准备工作区
2. 创建新视图
3. 创建新控制器命令
4. 将信息从控制器命令传送到视图
5. 分析并验证控制器命令中的 URL 参数
6. 创建新的任务命令
7. 修改新的任务命令
8. 创建新的企业 bean:
   a. 创建新表
   b. 创建新 CMP 企业 bean
   c. 将新 bean 映射到表中并创建模式
   d. 创建相关联的访问 bean
   e. 生成部署代码
   f. 用通用测试客户机测试 bean
9. 将 Bonus bean 集成到 MyNewControllerCmd 中
   a. 修改 MyNewTaskCmdImpl 类的 performExecute 方法，以计算新的奖金点数并将这些点数保存到 XBONUS 表中。
   b. 将 getResources 方法添加到 MyNewControllerCmdImpl 类，以返回命令使用的一组资源。此方法附带包含，以用于访问控制目的。
   c. 创建 BonusDataBean 以使奖金点数可以方便地显示在 JSP 模板中。
   d. 对新资源创建新的访问控制策略。
   e. 修改 MyNewJSPTemplate.jsp 文件，以允许用户输入奖金点数并显示结果。
   f. 测试集成代码。

10. 将上面所有的代码、访问控制策略、JSP 模板、映象和资源束部署到运行在 WebSphere Application Server 上的目标 WebSphere Commerce Server。

后面各部分描述了前面每个步骤的分步式详细信息。

找到样本代码

在开始这一教程前，请下载 WC_SAMPLE_55.zip 软件包，它包含了这些编程教程的起点。把此文件保存到开发机。例如，您可以将文件保存至 *WCDE_installdir* 目录:

该软件包与《*WebSphere Commerce 编程指南与教程*》同处于以下 Web 站点:

http://www.ibm.com/software/commerce/library/

准备工作空间

在这一步中，您需要将样本代码导入 WebSphere Commerce 工作区。这一样本代码是教程的起点。

要准备工作区，请执行以下操作:

1. 确保您已安装 WebSphere Commerce 开发环境 V5.5 并且已完成了开发环境的配置。您还应已在开发环境内发布了一个基于“时尚潮流”样本商店（它是消费者直销模型的一个示例）的商店。关于如何发布商店的指示信息可在 WebSphere Commerce 生产和开发联机帮助中找到。
2. 通过执行以下操作将样本代码导入工作区:
   a. 启动 WebSphere Commerce 开发环境如下:
      - Business Professional **开始 > 程序 > IBM WebSphere Commerce Studio > WebSphere Commerce 开发环境**
      - Express Express **开始 > 程序 > IBM WebSphere - Express Developer Edition > WebSphere Commerce 开发环境**

   b. 切换到“J2EE 透视图”（**窗口 > 打开透视图 > J2EE**），然后选择 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
   c. 展开 **WebSphereCommerceServerExtensionsLogic** 项目。
   d. 用鼠标右键单击 **src** 文件夹并选择**导入**。“导入”向导打开。
   e. 从**选择导入源**列表中，选择 **Zip 文件**并单击**下一步**。

f. 单击**浏览**（在 **Zip 文件**字段旁）并浏览至样本代码。此文件的位置如下:
yourDirectory\WC_SAMPLE_55.zip
其中 *yourDirectory* 是放置下载数据包的目的地目录。

g. 单击**全部不选**，然后展开目录并选择导入以下文件:

- com\ibm\commerce\sample\commands\ MyNewControllerCmd.java
- com\ibm\commerce\sample\commands\ MyNewControllerCmdImpl.java
- com\ibm\commerce\sample\commands\MyNewTaskCmd.java
- com\ibm\commerce\sample\commands\MyNewTaskCmdImpl.java
- com\ibm\commerce\sample\databeans\MyNewDataBean.java

h. 在**文件夹**字段中，`WebSphereCommerceServerExtensionsLogic/src` 文件夹已经指定。保留该值。

i. 单击**完成**。

3. 用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 项目并选择**重构项目**。

4. 通过执行以下操作把教程中的 JSP 模板导入到相应的目录:

a. 在 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图中，展开 **Stores** 项目。然后展开 **Web Content > *FashionFlow_name***，其中 *FashionFlow_name* 是基于“时尚潮流”样本商店的商店名称。

如果您刚发布您的商店，则可能未显示 *FashionFlow_name* 目录。若为是这种情况，请用鼠标右键单击 Stores 项目并选择**刷新**。现在 *FashionFlow_name* 目录将出现在工作区。

b. 用鼠标右键单击 ***FashionFlow_name*** 目录并选择**导入**。
将打开“导入”向导。

c. 从**选择导入源**列表中，选择 **Zip 文件**并单击**下一步**。

d. 单击**浏览**（在 **Zip 文件**字段旁）并浏览至样本代码。此文件的位置如下:
yourDirectory\WC_SAMPLE_55.zip
其中 *yourDirectory* 是放置下载数据包的目的地目录。

e. 单击**全部不选**，然后选择 MyNewJSPTemplate_All.jsp 文件。

f. 在**文件夹**字段中，`Stores/Web Content/`*`FashionFlow_name`* 文件夹已指定。保留该值。

g. 单击**完成**。

5. 通过执行以下操作把教程的属性文件导入到相应的目录:

a. 在 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图中，展开以下目录:
Stores > Web Content> WEB-INF> classes > *FashionFlow_name* 目录。

b. 用鼠标右键单击 ***FashionFlow_name*** 目录并选择**导入**。
将打开“导入”向导。

c. 从**选择导入源**列表中，选择 **Zip 文件**并单击**下一步**。

d. 单击**浏览**（在 **Zip 文件**字段旁）并浏览至样本代码。此文件的位置如下:
yourDirectory\WC_SAMPLE_55.zip
其中 *yourDirectory* 是您将数据包下载至的目录。

e. 单击**全部不选**，然后选择 Tutorial_All_en_US.properties 文件。

f. 在**文件夹**字段中，`Stores/Web Content/WEB-INF/classes/`*`FashionFlow_name`*
文件夹已指定。保留该值。

g. 单击**完成**。

6. 有四个文件用于为在本教程中创建的新资源装入访问控制策略。它们是:
   - MyNewViewACPolicy.xml – 这个 XML 文件包含创建新视图时使用的访问控制策略。
   - MyNewControllerCmdACPolicy.xml – 这个 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_template.xml – 这个 XML 文件包含创建新的企业 bean 时使用的访问控制策略。
   - SampleACPolicy_template_en_US.xml – 这个 XML 文件包含创建新企业 bean 时的访问控制策略描述。

   通过执行以下操作把前面的文件复制到相应的目录:

   a. 在文件系统中，浏览至以下目录:
   yourDirectory\WC_SAMPLE_55.zip。

   b. 展开该 ZIP 文件并将前面的四个文件抽取到以下目录中:
   WCDE_installdir\Commerce\xml\policies\xml

7. 通过执行以下操作，测试您的环境以确保已准备好开始教程:

   a. 在 WebSphere Studio Application Developer 中，切换至服务器视图。

   b. 启动 Payment Server。如果正在运行本地 Payment Server，用鼠标右键单击 **WebSphereCommercePaymentsServer** 并选择**启动**（或**重新启动**）。

   c. 用鼠标右键单击 **WebSphereCommerceServer** 并选择**启动**（或**重新启动**）。

d. 监视控制台以查看 WebSphereCommerceServer 服务器何时完成其启动过程。服务器启动时，您会看到与以下类似的信息:

```
[4/2/03 12:56:06:286 EST] 66adf8d1 ApplicationMg A WSVR0221I:
应用程序已启动: WebSphereCommerceServer
[4/2/03 12:56:06:777 EST] 66adf8d1 HttpTransport A SRVE0171I:
传输 http 正在侦听端口 9,080。
[4/2/03 12:56:10:742 EST] 66adf8d1 HttpTransport A SRVE0171I:
传输 https 正在侦听端口 9,443。
[4/2/03 12:56:10:762 EST] 66adf8d1 HttpTransport A SRVE0171I:
传输 http 正在侦听端口 8,080。
[4/2/03 12:56:10:762 EST] 66adf8d1 HttpTransport A SRVE0171I:
传输 http 正在侦听端口 80。
[4/2/03 12:56:11:123 EST] 66adf8d1 HttpTransport A SRVE0171I:
传输 https 正在侦听端口 443。
[4/2/03 12:56:11:183 EST] 66adf8d1 HttpTransport A SRVE0171I:
传输 https 正在侦听端口 9,043。
[4/2/03 12:56:11:653 EST] 66adf8d1 RMIConnectorC A ADMC0026I:
RMI 连接器在端口 2809 上可用
[4/2/03 12:56:12:575 EST] 66adf8d1 WsServer
   A WSVR0001I: 服务器 server1 已为电子商务打开
```

e. 在 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图中，展开 **Stores** 项目。然后展开 **Web Content > *FashionFlow_name***。

f. 用鼠标右键单击 **index.jsp** 文件并选择**在服务器上运行**。样本商店打开。

g. 选择一个产品并确保您可以购买它。

现已准备好继续这些教程。

创建新视图

本教程的第一步是创建新视图。此新视图名为 MyNewView 并有一个相应的 JSP 模板 MyNewJSPTemplate.jsp。

在教程的本部分中，您将学习如下内容:

- 在哪里放置适用于特定商店的 JSP 模板和图形文件。
- 如何用 WebSphere Studio Application Developer 来创建 JSP 模板。
- 如何创建包含 JSP 模板文本的属性文件。
- 如何用新的“MyNewView”更新视图注册表（VIEWREG 表）
- 如何为新的视图设置访问控制。
- 如何用 WebSphere Test Environment（WTE）来测试新的视图。

通常，创建新视图包括以下步骤:

1. 命名视图并将它注册到视图注册表。

2. 创建新的属性文件，在其中存储 JSP 模板的可转换文本。
3. 创建新视图的新 JSP 模板。
4. 创建并装入视图的访问控制策略。

注册 MyNewView

在此方案，创建的视图称为 MyNewView。这一视图必须在 VIEWREG 表中注册，这个表是命令注册表的一部分。要注册新视图，您只需要使用简单的 SQL 语句在 VIEWREG 表中创建一个新条目。

在继续这一步骤前，必须知道商店的唯一标识。这可以通过对开发数据库运行以下 SQL 查询来确定:

```
select STOREENT_ID from STOREENT where IDENTIFIER = 'FashionFlow_name'
```

其中 *FashionFlow_name* 是商店名。记下此处的值，因为下一节中将需要它:

DB2 如果正在使用 DB2 数据库，请执行以下操作注册 MyNewView:

1. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 命令行工具 > 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. 关闭工具设置。
5. 在选中“脚本”选项卡的情况下，通过在脚本窗口中输入以下信息，在 VIEWREG 表中创建必需的条目:

```
connect to developmentDB user dbuser using dbpassword;
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
   CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE)
values ('MyNewView',-1, FF_storeent_ID,
   'com.ibm.commerce.command.ForwardViewCommand',
   'com.ibm.commerce.command.HttpForwardViewCommandImpl',
'docname=MyNewJSPTemplate.jsp', 'This is my new view for tutorial one',
0, null)
```

其中

- *developmentDB* 是开发数据库的名称
- *dbuser* 是数据库用户
- *dbpassword* 是数据库用户的密码
- *FF_storeent_ID* 是基于“时尚潮流”样本商店的商店唯一标识。

单击**执行**图标。

您应该看到一条消息，指示 SQL 命令已成功完成。

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:

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:

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
   CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE)
values ('MyNewView',-1, FF_storeent_ID,
   'com.ibm.commerce.command.ForwardViewCommand',
   'com.ibm.commerce.command.HttpForwardViewCommandImpl',
   'docname=MyNewJSPTemplate.jsp','This is my new view for tutorial 1',
   0, null);
```

   其中

   - *FF_storeent_ID* 是基于“时尚潮流”样本商店的商店唯一标识。

   按 Enter 键运行 SQL 语句。
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。

MyNewView 已注册。

为教程创建一个属性文件

在这一步中，您要创建一个新的属性文件用以保存在 JSP 模板中使用的所有可转换文本。把可转换文本和 JSP 模板分开，这一做法本身就使得转换任务简单了许多，而且它对拥有全球化 Web 站点也是很重要的。

要创建属性文件,请在 WebSphere Studio Application Developer 中执行以下操作:

1. 打开 Web 视图（**视窗 > 打开视图 > Web**）。
2. 在 **Stores** Web 项目中，展开 **Web Content > WEB-INF > classes > *FashionFlow_name*** 文件夹。您会发现 Tutorial_All_en_US.properties 文件。
3. 用鼠标右键单击 **Tutorial_All_en_US.properties** 并选择**打开工具 > 属性文件编辑器**。
4. 用鼠标右键单击 ***FashionFlow_name*** 文件夹并选择**新建 > 其它 > 简单 > 文件 > 下一个**来创建一个新的属性文件。
   将打开“新建文件”窗口。

5. 在**文件名**字段，输入 TutorialNLS_en_US.properties，然后单击**完成**。新的空文件打开。
6. 把 Tutorial_All_en_US.properties 文件的第一部分复制到新的 TutorialNLS_en_US.properties 文件。这向 TutorialNLS_en_US.properties 文件引入了如下“名称 – 值”对:

```
# -- SECTION 1 -- #

ProgrammerGuide=Programmer's Guide
Tutorial=Tutorial: Creating new business logic
ParametersFromCmd= List of parameter-value pairs sent from
                   the controller command
CalledByControllerCmd=MyNewView was called by a controller command
CalledByWhichControllerCmd=MyNewView was called by the controller
                           command which is -
ControllerParm1=ControllerParm1=
ControllerParm2=ControllerParm2=
Example=This is an example of using the <if> tag from JSP
        Standard Tag Library (JSTL)
UserName=UserName=
Points=Points=
Greeting=Greeting=
UserId=UserId=
FirstInput=Your first input parameter
RegisteredUser=is a registered user
ReferenceNumber=The member refernce number of this user is
NotRegisteredUser=is not a registered user
BonusAdmin=Bonus Administration
PointBeforeUpdate=The bonus point before update is
PointAfterUpdate=The bonus point after update is
EnterPoint=Please enter the points, then submit it to the controller
   command

# -- END OF SECTION 1 -- #
```

   注意值中的断行只出于显示目的。
7. 保存 TutorialNLS_en_US.properties 文件（Ctrl+S）。

新的 TutorialNLS_en_US.properties 文件保存在 Stores\Web Content\WEB-INF\classes*FashionFlow_name* 目录下并且新的 JSP 模板会将它用作资源束。

注: 当属性文件的内容已更改，在这些更改能够在测试中查看到之前必须重新启动服务器。

创建 MyNewJSPTemplate

在这一步中，您需要使用 WebSphere Studio 中的 Page Designer 工具创建新的 JSP 模板。特别地，需要创建与 MyNewView 一起使用的 MyNewJSPTemplate.jsp。创

建此模板时，请创建一个新的空白 JSP 模板，然后从另一 JSP 模板（MyNewJSPTemplate_All.jsp）添加相应的部分。

要创建新的 JSP 模板，请执行以下操作:

1. 在 Web 透视图中，切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
2. 用鼠标右键单击 **Stores** Web 项目并选择**属性**。
3. Business Professional 在左窗格中选择 **Web**，然后从“可用的 Web 项目功能”列表中选择**包含 JSP 标准标记库**。

   Express 在左窗格中选择 **Web 项目功能**，然后从“可用的 Web 项目功能”列表中选择 **JSP 标准标记库**。

   单击**应用**。更新完成后，单击**确定**以关闭属性编辑器。
4. 展开 **Web Content*FashionFlow_name*** 目录。
5. 用鼠标右键单击 **MyNewJSPTemplate_All.jsp** 文件并选择**打开工具 > Page Designer**。
6. 用鼠标右键单击 ***FashionFlow_name*** 文件夹并选择**新建> JSP 文件**，以便在该文件夹中创建新的 JSP 模板。
   将打开“新建 JSP 文件”窗口。
7. 为新文件指定值，如下:
   a. 在**文件名**字段中，输入 `MyNewJSPTemplate.jsp`。
   b. 从**标记语言**下拉列表，选择 **XHTML** 并单击**下一步**。
   c. Business Professional 单击**添加标记库**。
      “选择标记库”窗口打开。

      Express 选择**配置高级选项** > 单击**下一步** > 单击**添加**（在“标记库”表旁）。
   d. 选择以下标记库:
      - http://java.sun.com/jstl/core
      - http://java.sun.com/jstl/fmt

      单击**确定**，然后单击**下一步**。
   e. 单击**下一步**。
   f. 清除（**使用工作台缺省值**）**工作台编码**复选框。
   g. 从**编码**下拉列表选择 **ISO Latin -1**。
   h. 从**文档类型**下拉列表选择 **XHTML 1.0 Transitional**。

i. 单击**完成**。

MyNewJSPTemplate.jsp 文件打开。单击“设计”、“源”和“预览”选项卡查看文件的不同视图。

8. 在选中“设计”选项卡的情况下，单击**在此放置 MyNewJSPTemplate.jsp 的内容**文本。用 `Hello world!` 替换这一文本。
9. 切换至“源”选项卡，然后切换至“预览”选项卡。注意文本已更改。
10. 现在必须把准备部分从 MyNewJSPTemplate_All.jsp 文件复制到新的 MyNewJSPTemplate.jsp 文件中。这一部分设置了文件更新所需的占位符。把 `<%--PREPARATION SECTION 和 END OF PREPARATION SECTION --%>` 标记之间的文本复制到新的 JSP 模板。在将此文本复制到 JSP 模板时，请覆盖以下文本:

```
<title> MyNewJSPTemplate.jsp </title>
</head>
<body>
<p> Hello World! </p>
</body>
</html>
```

注: 不要把 `<%--PREPARATION SECTION 和 END OF PREPARATION SECTION --%>` 标记复制到新的 JSP 模板。只复制包含在这两个标记之间的文本。

11. 从 MyNewJSPTemplate_All.jsp 文件将 1A 和 2 部分复制到新的 MyNewJSPTemplate.jsp 文件中。把新的文本置于 `<!-- SECTION 1A -->`、`<!-- END OF SECTION 1A -->`、`<!-- SECTION 2 -->` 和 `<!-- END OF SECTION 2 -->` 标记之间。这将把以下文本引入到 MyNewJSPTemplate.jsp 中:

```
<!-- SECTION 1A -->

    <%@ include file="include/EnvironmentSetup.jsp"%>

<!-- END OF SECTION 1A -->

<!-- SECTION 2 -->

<fmt:setLocale value="${CommandContext.locale}" />
<fmt:setBundle basename="${sdb.directory}/TutorialNLS" var="tutorial" />

<!-- END OF SECTION 2 -->
```

第一部分包括用于设置环境变量的 EnvironmentSetup.jsp 文件。第二部分用于创建从属性文件检索信息所用的资源束对象以及设置语言环境。

12. 现在将图形和文本添加到 JSP 模板。同样地，此步骤也是通过将文本从 MyNewJSPTemplate_All.jsp 文件复制到 MyNewJSPTemplate.jsp 文件来完成的。这次，把第 3 部分从 MyNewJSPTemplate_All.jsp 文件复制到 MyNewJSPTemplate.jsp 文件。这将以下文本引入到 JSP 模板中:

```
<!-- SECTION 3 -->

<table cellpadding="0" cellspacing="0" border="0">
 <tr>
  <td bgcolor="#ff2d2d" >
   <img src="<c:out value="${sdb.jspStoreDirFilePath}
                           images/male_blueshirt.gif" />" border="0"/>
  </td>
 </tr>
</table>

<h1><fmt:message key="ProgrammerGuide" bundle="${tutorial}" /> </h1>

<h2><fmt:message key="Tutorial" bundle="${tutorial}" /> </h2>

<!-- END OF SECTION 3 -->
```

SECTION 3 引入了一个特定于商店的图像子文件夹（Stores\Web Content*FashionFlow_name*\images）中的图像。它也会从属性文件检索文本。

13. 保存对 MyNewJSPTemplate.jsp 文件所作的更改（Ctrl+S）。

创建并装入 MyNewView 的访问控制策略

必须对新视图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指定允许所有用户执行此视图。注意这种类型的访问控制策略对开发环境是可接受的，但是可能并不适用于其它环境。关于更高级的访问控制要求，请参阅*《WebSphere Commerce 安全性指南》*。

访问控制策略由 MyNewViewACPolicy.xml 文件定义，您可将该文件放在以下目录中，作为准备步骤的一部分:
WCDE_installdir\Commerce\xml\policies\xml

要装入新策略，请执行以下操作:

1. 在命令提示符下，浏览到以下目录:
   WCDE_installdir\Commerce\bin
2. 您必须发出 acpload 命令，此命令具有以下格式:

   ```
   acpload db_name db_user db_password inputXMLFile
   ```

   其中

   - *db_name* 是开发数据库的名称。
   - *db_user* 是数据库用户名。
   - *db_password* 是数据库用户密码。
   - *inputXMLFile* 是包含访问控制策略规范的 XML 文件。在此例中，指定 MyNewViewACPolicy.xml。

以下是带有指定变量的命令的示例:

```
acpload Demo_Dev db2user db2user MyNewViewACPolicy.xml
```

测试 MyNewView

创建新视图的最后一步是在 WebSphere Test Environment 中测试新视图。注意测试新视图时（以及以后测试新命令时）必须首先启动商店的主页。需要启动商店的原因是新视图是特别向您的商店注册的。因此，在尝试访问新视图之前，必须先启动商店主页以便在命令上下文中设置商店标识值。类似地，以后创建的控制器命令也是特别要向您的商店注册的，并需要来自命令上下文的商店标识。

要测试新视图，请执行以下操作:

1. 在 WebSphere Studio Application Developer 中，打开“服务器”视图（**视窗 > 打开视图 > 服务器**）。
2. 用鼠标右键单击 **WebSphereCommerceServer** 服务器并选择**启动**（或**重新启动**）。
3. 用鼠标右键单击 Stores\Web Content*FashionFlow_name* 目录下的 **index.jsp** 并选择**在服务器上运行**。
   商店主页显示在 Web 浏览器上。
4. 在 Web 浏览器中输入以下 URL:

   ```
   http://localhost/webapp/wcs/stores/servlet/MyNewView
   ```

   几秒种后，会显示新的 JSP 模板，如以下屏幕快照所示:


aa.jsp - Microsoft Internet Explorer provided by Sympatico
File Edit View Favorites Tools Help
Back Search Favorites Media
Address https://localhost/webapp/wcs/stores/servlet/MyNewView
Go Links
世界，您好！
程序员指南
教程：创建新的业务逻辑
Discussions Subscribe... Discussions not availa
Done
Local intranet

图 31.

创建新的控制器命令

在此步骤中，您将创建新的控制器命令，名为 MyNewControllerCmd。起初，此命令仅返回 MyNewView 视图。

在教程的本部分中，您将学习如下内容:

- 控制器命令中所包含代码的最低要求
- 如何创建新的控制器命令接口和实现类
- 如何设置控制器命令来返回视图
- 如何在命令注册表中注册控制器命令
- 如何为控制器命令设置访问控制

通常，创建新的控制器命令包括以下步骤:

1. 在命令注册表中注册新命令。
2. 为命令创建接口。
3. 为命令创建实现类。
4. 创建并装入命令的访问控制策略。
5. 测试命令。

注册 MyNewControllerCmd

在教程的此部分中，您将创建新的控制器命令，名为 MyNewControllerCmd。必须在命令注册表中注册此命令。特别地，接口必须在 URLREG 表中注册，而接口及其实现类之间的关联要在 CMDREG 表中注册。

DB2 如果您使用的是 DB2 数据库，请执行以下操作注册 MyNewControllerCmd:

1. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 命令行工具 > 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 URLREG 表中创建必需的条目:

```
connect to developmentDB user dbuser using dbpassword;
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS, DESCRIPTION,
   AUTHENTICATED) values ('MyNewControllerCmd',FF_storeent_ID,
   'com.ibm.commerce.sample.commands.MyNewControllerCmd',
   0, 'This is a new controller command for tutorial one.',null);
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
   CLASSNAME, TARGET)
values (FF_storeent_ID,
```

```
'com.ibm.commerce.sample.commands.MyNewControllerCmd',
'This is a new controller command for tutorial one.',
'com.ibm.commerce.sample.commands.MyNewControllerCmdImpl',
'local');
```

其中

- *developmentDB* 是开发数据库的名称
- *dbuser* 是数据库用户
- *dbpassword* 是数据库用户的密码
- *FF_storeent_ID* 是基于“时尚潮流”样本商店的商店唯一标识。

单击**执行**图标。

您应该看到一条消息，指示 SQL 命令已成功完成。

Oracle 如果正在使用 Oracle 数据库，请执行以下操作注册 MyNewControllerCmd:

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:

```
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS, DESCRIPTION,
   AUTHENTICATED) values ('MyNewControllerCmd',FF_storeent_ID,
   'com.ibm.commerce.sample.commands.MyNewControllerCmd',
   0, 'This is a new controller command for tutorial one.',null);
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
   CLASSNAME, TARGET)
values (FF_storeent_ID,
   'com.ibm.commerce.sample.commands.MyNewControllerCmd',
   'This is a new controller command for tutorial one.',
   'com.ibm.commerce.sample.commands.MyNewControllerCmdImpl','local');
```

其中

- *FF_storeent_ID* 是基于“时尚潮流”样本商店的商店唯一标识。

按 Enter 键运行 SQL 语句。

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。

注意第二个对 CMDREG 表的插入语句不是绝对必需的。在此方案中，接口会使用缺省实现，因此实际上无需在命令注册表中指定接口和实现类之间的该关联。此处包含它是出于完整性目的。

创建 MyNewControllerCmd 接口

根据 WebSphere Commerce 编程模型，所有新的控制器命令必须有一个接口以及一个实现类。在本教程中，我们在样本代码中提供了接口库。它被分割成许多不同的部分，这些部分当前以注释形式包含在代码中。在浏览教程时，您可以取消对代码各部分的注解。

要创建 MyNewControllerCmd 接口，请执行以下操作:

1. 在 WebSphere Studio Application Developer 中，打开 Java 视图（**视窗 > 打开视图 > Java**）。
2. 展开 **WebSphereCommerceServerExtensionsLogic** 项目。
3. 浏览至 **src** 目录，然后展开 **com.ibm.commerce.sample.commands** 数据包。
4. 双击 **MyNewControllerCmd.java** 接口以打开文件。
5. 在源代码中，取消对第 1 部分的注解（删除该部分之前和之后的“/*”）。这将以下代码引入到接口中:

   ```
   /// Section 1 ////////////////////////////////////////////

   // set default command implement class

    static final String defaultCommandClassName =
        "com.ibm.commerce.sample.commands.MyNewControllerCmdImpl";


   /// End of section 1/////////////////////////////////////////
   ```

   这部分代码指定缺省情况下接口应该使用 MyNewControllerCmdImpl 实现类。
6. 保存对接口的更改（Ctrl+S）。

创建 MyNewControllerCmdImpl 实现类

接口一旦创建，下一步就是创建命令的实现类。在本教程中，我们在样本代码中提供了实现类库。它被分割成许多不同的部分，这些部分当前以注释形式包含在代码中。在浏览教程时，您可以取消对代码各部分的注解。

要创建 MyNewControllerCmdImpl 实现类，请执行以下操作:

1. 双击 **MyNewControllerCmdImpl.java** 类以打开它。
2. 在“概览”视图中，选择 **performExecute** 方法以查看其源代码。

3. 在 performExecute 方法的源代码中，取消对第 1 部分的注解。这将以下代码引入到方法中:

```
/// Section 1 ////////////////////////////////////////////////

    /// create a new TypedProperties for output purpose.

 TypedProperty rspProp = new TypedProperty();

/// End of section 1 ////////////////////////////////////////
```

这创建了一个新的 TypedProperty 对象，这一对象用于保存命令的响应属性。

4. 在 performExecute 方法的源代码中，取消对第 5 部分的注解。这将以下代码引入到方法中:

```
/// Section 5 ////////////////////////////////////////////////

    /// see how controller command call a JSP

 rspProp.put(ECConstants.EC_VIEWTASKNAME, "MyNewView");
 setResponseProperties(rspProp);

/// End of section 5/////////////////////////////////////////
```

这部分代码实现了两个主要任务。首先，WebSphere Commerce 编程模型要求所有的控制器命令返回一个视图。在此部分中，它指定要返回的视图是先前创建的 MyNewView。此外，它将命令的响应属性设置为新的 rspProp 对象。

5. 保存更改（Ctrl+S）。
6. 要编译已对代码所作的更改，用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 项目并选择**构建项目**。

创建并装入命令的访问控制策略

必须对新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指定允许所有用户执行此命令。注意这种类型的访问控制策略对开发环境是可接受的，但是可能并不适用于其它环境。关于更高级的访问控制要求，请参阅*《WebSphere Commerce 安全性指南》*。

访问控制策略由 MyNewControllerCmdACPolicy.xml 文件定义，您可将该文件放在以下目录中，作为准备步骤的一部分:
WCDE_installdir\Commerce\xml\policies\xml

要装入新策略，请执行以下操作:

1. 在命令提示符下，浏览到以下目录:
   WCDE_installdir\Commerce\bin
2. 您必须发出 acpload 命令，此命令具有以下格式:

```
acpload db_name db_user db_password inputXMLFile
```

其中

- *db_name* 是开发数据库的名称。
- *db_user* 是数据库用户名。
- *db_password* 是数据库用户密码。
- *inputXMLFile* 是包含访问控制策略规范的 XML 文件。在此例中，指定 `MyNewControllerCmdACPolicy.xml`。

以下是带有指定变量的命令的示例:

```
acpload Demo_Dev db2user db2user MyNewControllerCmdACPolicy.xml
```

测试 MyNewControllerCmd

既然接口、实现类、命令注册和访问控制信息都已创建，您可测试新控制器命令。

如果 Java 代码已修改，在更改生效前必须重新启动测试服务器。

要测试新代码，请执行以下操作:

1. 切换至“服务器”视图（**视窗 > 打开视图 > 服务器**）。
2. 用鼠标右键单击 **WebSphereCommerceServer** 服务器并选择**启动**（或**重新启动**）。
3. 用鼠标右键单击 Stores\Web Content*FashionFlow_name* 目录下的 **index.jsp** 并选择**在服务器上运行**。
   商店主页显示在 Web 浏览器上。
4. 在 Web 浏览器中输入以下 URL:

   ```
   http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd
   ```

   几秒种后，会显示新的 JSP 模板，如以下屏幕快照所示:

图 32.

将信息从 MyNewControllerCmd 传递至 MyNewView

在这一步中，您需要修改 MyNewControllerCmd 以使它把信息传递到 MyNewView。显示了两种向视图传递信息的不同方法。首先，您将学习如何使用响应属性的 TypedProperties 对象以及如何从这一对象抽取信息到 JSP 模板。其次，您将学习如何创建用于传递信息到 JSP 模板的新数据 bean。

使用 TypedProperties 对象传递信息

在本部分中，您将修改 MyNewControllerCmdImpl 以将信息传送到 JSP 模板。特别地，您将修改命令来把附加的“名称 - 值”对添加到现有的 rspProp TypedProperties 对象，这一对象用于来自命令的响应属性。在 JSP 模板中，您需要使用 JSTL 表达式语言从响应属性中抽取信息。

在教程的本部分中，您将学习如下内容:

- 如何修改控制器命令以把附加的响应属性包含到 TypedProperty
- 如何修改 JSP 模板以使用 JSTL 表达式语言从响应属性中检索信息

要启用 JSP 模板中 TypedProperties 对象信息的显示，请执行以下操作:

1. 第一步是修改 MyNewControllerCmdImpl 类，如下:
   a. 切换至 Java 视图。
   b. 展开这些目录: **WebSphereCommerceServerExtensionsLogic > src > com.ibm.commerce.sample.commands**。
   c. 双击 **MyNewControllerCmdImpl.java** 并在“概览”视图中选择其 **performExecute** 方法。
   d. 在 performExecute 方法的源代码中，取消对第 2 部分的注释。这将以下代码引入到方法中:

```
/// Section 2 ////////////////////////////////////////////////

    /// see how the controller command pass in variables to JSP

 /// add additional parameters in controller command to rspProp
  ///  for response
    String message1 = "Hello from IBM!";

 rspProp.put("controllerParm1", message1);
 rspProp.put("controllerParm2", "Have a nice day!");

/// End of section 2/////////////////////////////////////////
```

   以上代码片段创建了两个新参数，这些参数放在响应属性对象中。此对象最终将传递到视图。
   e. 保存更改。
   f. 通过用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 项目并选择**构建项目**来编译命令。
2. 下一步必须通过执行以下操作更新 MyNewJSPTemplate.jsp 文件:
   a. 如果 JSP 模板文件还未打开，请执行以下操作:

1) 在 Web 透视图中，切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图，并展开 **Stores** Web 项目。
2) 浏览至 **Web Content*FashionFlow_name*** 目录。
3) 用鼠标右键单击 **MyNewJSPTemplate_All.jsp** 并选择**打开工具 > Page Designer**。
4) 用鼠标右键单击 **MyNewJSPTemplate.jsp** 并选择**打开工具 > Page Designer**。

b. 把第 4 部分从 MyNewJSPTemplate_All.jsp 文件复制到新的 MyNewJSPTemplate.jsp 文件。把新的文本置于 `<!-- SECTION 4 -->` 和 `<!-- END OF SECTION 4 -->` 标记之间。
这将把以下文本引入到 MyNewJSPTemplate.jsp 中:

```
<!-- SECTION 4 -->

<h3><fmt:message key="ParametersFromCmd" bundle="${tutorial}" /> </h3>

<fmt:message key="ControllerParm1" bundle="${tutorial}" />
<c:out value="${controllerParm1}"/> <br />

<fmt:message key="ControllerParm2" bundle="${tutorial}" />
<c:out value="${controllerParm2}"/> <br /> <br />

<!-- END OF SECTION 4 -->
```

这一部分使用 JSTL 表达式语言获取并显示从控制器命令传递来的值。

c. 保存更改。

3. 下一步是通过执行以下操作测试对控制器命令和 JSP 模板的修改:
   a. 切换至“服务器”视图（**视窗 > 打开视图 > 服务器**）。
   b. 用鼠标右键单击 **WebSphereCommerceServer** 服务器并选择**启动**（或**重新启动**）。
   c. 用鼠标右键单击 Stores\Web Content*FashionFlow_name* 目录下的 **index.jsp** 并选择**在服务器上运行**。
   商店主页显示在 Web 浏览器上。
   d. 在 Web 浏览器中输入以下 URL:

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd
```

几秒种后，会显示新的 JSP 模板，如以下屏幕快照所示:

图 33.

使用数据 bean 传递信息

此部分中，添加代码以确定是直接调用视图，还是由控制器命令调用它。在稍后的例子中，JSP 模板也应显示调用它的命令的名称。

为了将这一信息传递到视图，要创建一个称作 MyNewDataBean 的新数据 bean。还修改了 MyNewJSPTemplate，这样它可显示新的信息。

MyNewDataBean 严格用于使来自控制器命令的信息可用于 JSP 模板。请将此与使来自数据库的信息可用作对比。在教程稍后部分，您将学习如何创建新数据 bean，它的目的是使来自数据库的信息可用于 JSP 模板。

您也许会奇怪为什么本教程使用数据 bean 的目的仅仅是使来自控制器命令的信息可用。创建此 bean 有两个原因：首先，它是一个很好的编程实践，它实现了对属性的逻辑分组；其次，它使 Web 页面开发者能更简单地通过使用数据 bean 向 Web 页面添加信息，而不是通过使用 TypedProperties 响应属性对象来执行此操作。

在教程的本部分中，您将学习如下内容:

- 如何创建新数据 bean
- 如何修改控制器命令以实例化数据 bean
- 如何使用控制器命令设置数据 bean 的属性
- 如何把实例化数据 bean 传递到 JSP 模板
- 如何修改 JSP 模板以便从数据 bean 检索信息
- 请参阅在 JSP 模板中使用 <if> 标记的示例

创建 MyNewDataBean

MyNewDataBean 用于把信息传递到 MyNewJSPTemplate.jsp 页面。在本教程其它部分，我们在样本代码中提供了数据 bean 库。它被分割成许多不同的部分，这些部分当前以注释形式包含在代码中。在浏览教程时，您可以取消对代码各部分的注解。

要创建 MyNewDataBean，请执行以下操作:

1. 打开 Java 视图并使用“数据包资源管理器”视图。
2. 展开这些目录: **WebSphereCommerceServerExtensionsLogic > src > com.ibm.commerce.sample.databeans**。
3. 双击 **MyNewDataBean.java** 查看其源代码。
4. 在主类的源代码中，取消对第 1 部分的注解。这将以下代码引入到类中:

```
 /// Section 1 ////////////////////////////////////////////
/// create fields and accessors (setter/getter methods)

private java.lang.String callingCommandName = null;
private boolean calledByControllerCmd = false;


public java.lang.String getCallingCommandName()
{
  return callingCommandName;
 }
```

```
public void setCallingCommandName(java.lang.String newCallingCommandName)
{
  callingCommandName = newCallingCommandName;
 }

public boolean getCalledByControllerCmd()
{
  return calledByControllerCmd;
 }

public void setCalledByControllerCmd(boolean newCalledByControllerCmd)
{
  calledByControllerCmd = newCalledByControllerCmd;
 }

/// End of Section 1 ////////////////////////////////////
```

上述代码引入了两个变量，它们用于在视图由控制器命令返回而非直接受视图的 URL 调用时显示信息。

5. 保存更改。

使用 MyNewControllerCmd 实例化 MyNewDataBean 并设置其属性

此步骤中，修改 MyNewControllerCmdImpl 以实例化 MyNewDataBean，并设置该 bean 的属性。

要修改 MyNewControllerCmdImpl，请执行以下操作:

1. 在 Java 视图中，展开以下这些目录:
   WebSphereCommerceServerExtensionsLogic > src > com.ibm.commerce.sample.commands。
2. 双击 **MyNewControllerCmdImpl.java** 来查看其源代码。
3. 在主类的代码中，取消导入部分 1 的注解从而使新数据 bean 在该类中可用。这将以下代码引入到类中:

```
/// Import Section 1 ////////////////////////////////
import com.ibm.commerce.sample.databeans.*;
/// End of Import Section 1 ////////////////////////
```

4. 在“概览”视图中，选择它的 **performExecute** 方法。
5. 在 performExecute 方法的源代码中，取消对 3A 和 3B 部分的注解。这将以下代码引入到方法中:

```
/// Section 3A/////////////////////////////////////////

 /// instantiate the MyNewDataBean databean and set the properties,
  ///  then add the instance to resProp for response

 MyNewDataBean mndb = new MyNewDataBean();
 mndb.setCallingCommandName(this.getClass().getName());
 mndb.setCalledByControllerCmd(true);
```

```
/// end of section 3A/////////////////////////////////////

/// Section 3B/////////////////////////////////////
 rspProp.put("mndbInstance", mndb);

/// end of section 3B/////////////////////////////////////
```

以上代码片段实例化 MyNewDataBean 对象，在对象中设置两个参数（指示它是由某个控制器命令调用的以及是哪个命令调用了它），然后它把数据 bean 对象置于响应属性中以使它对于 JSP 模板可用。

6. 保存更改。
7. 通过用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 项目并选择**构建项目**来编译代码更改。

在 MyNewJSPTemplate 中使用 MyNewDataBean

在本部分中，修改 MyNewJSPTemplate 以指示视图是否由控制器命令返回。如果它是由控制器命令返回的，那么它还应显示该控制器命令的名称。根据来自 MyNewDataBean 的值，JSP 模板使用表示条件逻辑的 JSTL 标记来确定这一点。

要修改 JSP 模板，请执行以下操作:

1. 如果 JSP 模板文件还未打开，请执行以下操作:
   a. 在 Web 透视图中，切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图，并展开 **Stores** Web 项目。
   b. 浏览至 **Web Content*FashionFlow_name*** 目录。
   c. 同时突出显示 **MyNewJSPTemplate_All.jsp** 和 **MyNewJSPTemplate.jsp** 文件，然后用鼠标右键单击并选择**打开工具 > Page Designer**。
2. 把第 5 部分从 MyNewJSPTemplate_All.jsp 文件复制到 MyNewJSPTemplate.jsp 文件。这将以下文本引入到 JSP 模板中:

```
<!-- SECTION 5 -->

<c:if test="${mndbInstance.calledByControllerCmd}">
   <fmt:message key="Example" bundle="${tutorial}" /> <br />
   <fmt:message key="CalledByControllerCmd" bundle="${tutorial}" />
      <br />
   <fmt:message key="CalledByWhichControllerCmd" bundle="${tutorial}" />
   <b><c:out value="${mndbInstance.callingCommandName}" /></b> <br />
      <br />
</c:if>

<!-- END OF SECTION 5 -->
```

这部分代码使用 JSTL <if> 标记来确定是否显示关于调用方控制器命令的信息。它也从教程资源束检索可转换文本。

3. 保存更改。

测试已修改的 JSP 模板

要测试已修改的 JSP 模板，请执行以下操作:

1. 切换至“服务器”视图（**视窗 > 打开视图 > 服务器**）。
2. 用鼠标右键单击 **WebSphereCommerceServer** 服务器并选择**启动**（或**重新启动**）。
3. 用鼠标右键单击 Stores\Web Content*FashionFlow_name* 目录下的 **index.jsp** 并选择**在服务器上运行**。
   商店主页显示在 Web 浏览器上。
4. 在 Web 浏览器中输入以下 URL:

   ```
   http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd
   ```

   几秒种后，会显示 JSP 模板，如以下屏幕快照所示:

图 34.

5. 下一步，输入 URL 以直接调用视图，并注意在显示信息中的不同之处:

   ```
   http://localhost/webapp/wcs/stores/servlet/MyNewView
   ```

分析并验证 MyNewControllerCmd 中的 URL 参数

在本步骤中，您需要修改控制器命令来使用通过调用控制器命令的 URL 传入的参数。验证逻辑也包含在命令中，以确保已包含了必需的参数，同时确保已对这些参数使用了适当的值。

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 ECApplicationException {}
```

现在必须向命令添加定制参数检查，并把 URL 参数传递到 JSP 模板。修改 validateParameters 方法时，请添加与 URL 参数相对应的新字段。

在教程的本部分中，您将学习如下内容:

- 如何向命令添加与 URL 参数相对应的新字段
- 如何使用 getRequestProperties 方法来以 URL 输入参数实现对这些字段的填充
- 如何捕获缺少参数异常
- 向视图传递 URL 参数的适当方法
- 请参阅各种缺少或错误参数值测试案例的示例

添加新字段至 MyNewControllerCmd

在此步骤中，您将为 URL 参数创建两个新字段。它们既添加到命令接口又添加到实现类。一个 URL 参数是一个字符串值，它保存用户名，另一个是一个整数，用来接受奖金点数的输入值。

要添加这些新字段，请执行以下操作:

1. 切换至 Java 视图。
2. 展开这些目录: **WebSphereCommerceServerExtensionsLogic > src > com.ibm.commerce.sample.commands**。
3. 双击 **MyNewControllerCmd.java** 接口以查看其源代码。
4. 取消对第 2 部分的注解，从而向接口添加新字段和相应的 getter 方法。这将以下代码引入到类中:

```
/// Section 2 ////////////////////////////////////////////////

// set interface methods


 public java.lang.Integer getPoints() ;

 public java.lang.String getUserName() ;
```

```
public void setPoints(java.lang.Integer newPoints) ;

public void setUserName(java.lang.String newUserName)  ;


/// End of section 2/////////////////////////////////////////
```

5. 保存更改。
6. 双击 **MyNewControllerCmdImpl.java** 类以查看其源代码。
7. 取消对主类中第 1 部分的注解，以向类添加新字段及其相应的 getter 和 setter 方法。这将以下代码引入到类中:

```
/// Section 1 ////////////////////////////////////////////

/// create and implement controller command's fields and accessors
/// (setter/getter methods)

private java.lang.String userName = null;
private java.lang.Integer points;


public java.lang.Integer getPoints() {
 return points;
}

public java.lang.String getUserName() {
 return userName;
}

public void setPoints(java.lang.Integer newPoints) {
 points = newPoints;
}

public void setUserName(java.lang.String newUserName) {
 userName = newUserName;
}

/// End of Section 1 /////////////////////////////////////////
```

8. 保存更改。

将 URL 参数传递到视图

在本步骤中，您需要包含代码以把输入参数传递到 JSP 模板。这是通过用输入参数的值设置数据 bean 中的字段完成的。

要传递 URL 参数，请执行以下操作:

1. 双击 **MyNewControllerCmdImpl.java**。
2. 在“概览”视图中，选择 **performExecute** 方法。
3. 在 performExecute 方法的源代码中，取消对第 3C 部分的注解。这将以下代码引入到方法中:

```
/// Section 3C/////////////////////////////////////

 // pass the input information to the databean
 mndb.setUserName(this.getUserName());
 mndb.setPoints(this.getPoints());

/// end of section 3C/////////////////////////////////////
```

该代码设置了数据 bean 对象中的值，从而使它们可用于 JSP 模板。

4. 保存更改。

捕获缺少的参数并验证值

在本步骤中，您将修改 validateParameters 方法以引入错误检查和参数验证逻辑。一旦修改后，代码会检查以下内容:

- 如果没有提供第一个输入参数，就会出现“未找到参数”的异常。这是由于第一个输入参数是一个必需参数。在本例中，会向客户显示一般错误页面。
- 第二个输入参数是可选的。这样，如果不提供第二个输入参数，并不会抛出“找不到参数”异常。而第二个输入参数的值缺省为零，并且客户不会受该错误影响。处理继续。

要添加本错误检查，请执行以下操作:

1. 双击 **MyNewControllerCmdImpl.java**。
2. 在“概览”视图中，选择它的 **validateParameters** 方法。
3. 在 validateParameters 方法的源代码中，取消对第 1 部分的注解。这将以下代码引入到方法中:

```
/// Section 1 ////////////////////////////////////////////

/// uncomment to check parameters

    final String strMethodName = "validateParameters";

 TypedProperty prop = getRequestProperties();

 /// retrieve required parameters
 try {
  setUserName(prop.getString("input1"));

 } catch (ParameterNotFoundException e) {
  /// the next exception uses _ERR_CMD_MISSING_PARAM ECMessage object
     /// defined in ECMessage class
  throw new ECApplicationException(ECMessage._ERR_CMD_MISSING_PARAM,
        this.getClass().getName(), strMethodName,

 ECMessageHelper.generateMsgParms(e.getParamName()));
 }
```

```
/// retrieve optional Integer
// set input2 = 0 if no input value
setPoints(prop.getInteger("input2",0));

/// End of section 1/////////////////////////////////////////
```

以上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，则会发生异常。由于第二个参数是可选的，因此如果该参数缺少或是值错误时此代码将把该值设置为 0。

4. 保存更改。

添加新字段至 MyNewDataBean

在本步骤中，您将把新字段及其关联的 getter 方法添加至 MyNewDataBean 数据 bean，从而使 URL 参数可用于 JSP 模板。

要修改 MyNewDataBean，请执行以下操作:

1. 双击 **MyNewDataBean.java** 查看其源代码。
2. 取消对第 2 部分的注解，从而将如下代码引入到类中:

```
/// Section 2 /////////////////////////////////////////

private java.lang.String userName = null;
private java.lang.Integer points;


public String getUserName() {
 return userName;
}

public void setUserName(java.lang.String newUserName) {
 userName = newUserName;
}


public Integer getPoints() {
 return points;
}

public void setPoints(java.lang.Integer newPoints) {
 points = newPoints;
}


/// End of Section 2 ////////////////////////////////////
```

3. 保存更改。
4. 通过用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 并选择 **构建项目**来编译代码更改。

修改 MyNewJSPTemplate 以显示 URL 参数

在本步骤中，您将执行以下操作来修改 MyNewJSPTemplate.jsp 文件以便添加一个显示 URL 输入参数的新部分:

1. 如果 JSP 模板文件还未打开，请执行以下操作:
   a. 在 Web 透视图中，切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图，并展开 **Stores** Web 项目。
   b. 浏览到 Web Content*FashionFlow_name* 子文件夹。
   c. 同时突出显示 **MyNewJSPTemplate_All.jsp** 和 **MyNewJSPTemplate.jsp** 文件，然后用鼠标右键单击并选择**打开工具 > Page Designer**。
2. 把第 6 部分从 MyNewJSPTemplate_All.jsp 文件复制到 MyNewJSPTemplate.jsp 文件。这将以下文本引入到 JSP 模板中:

```
<!-- SECTION 6 -->

<fmt:message key="UserName" bundle="${tutorial}" />
<c:out value="${mndbInstance.userName}"/> <br>

<fmt:message key="Points" bundle="${tutorial}" />
<c:out value="${mndbInstance.points}"/> <br>

<!-- END OF SECTION 6 -->
```

3. 保存更改。

测试 URL 参数值

下一步是要通过执行以下操作来测试新的错误检查是否正常工作:

1. 切换至“服务器”视图（**视窗 > 打开视图 > 服务器**）。
2. 用鼠标右键单击 **WebSphereCommerceServer** 服务器并选择**启动**（或**重新启动**）。
3. 用鼠标右键单击 Stores\Web Content*FashionFlow_name* 目录下的 **index.jsp** 并选择**在服务器上运行**。
   商店主页显示在 Web 浏览器上。
4. 用例 1: 第 1 个测试用例将是从 URL 中排除这两个参数。商店主页显示后，输入以下 URL:

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd
```

   由于没有参数传送到命令，因此显示一般应用程序错误。

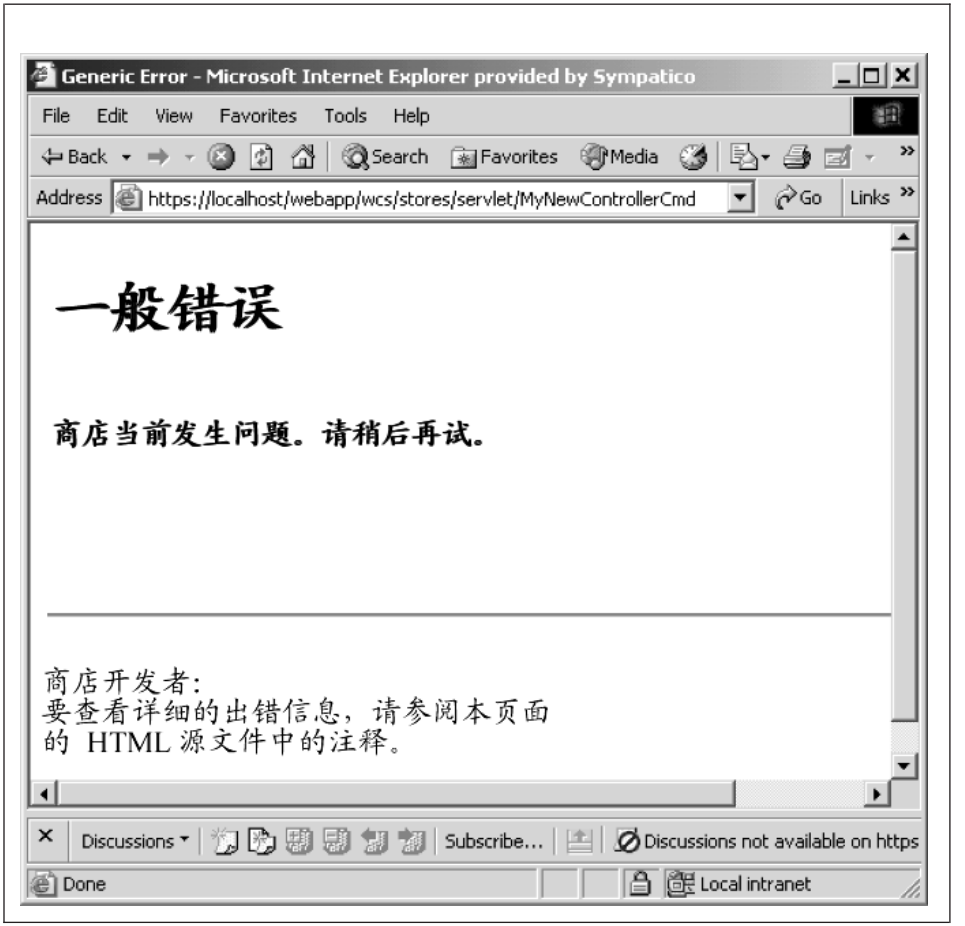

图 35.

WebSphere Studio Application Developer 中的控制台会显示类似以下内容的信息:

timeStamp 6730e546 CommerceSrvr E
com.ibm.commerce.sample.commands.MyNewControllerCmdImpl
validateParameters CMN0206E 请检查所有字段。“input1”是一个必需字段。

5. 用例 2: 下一个测试用例是使用有效的第一个参数，而忽略第二个参数。在此用例中，您可以预期应当不会检测出错误，因为缺省情况下缺少的第二个参数会使用零值。输入以下 URL:

   ```
   http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd?input1=abc
   ```

   此命令的结果是显示 MyNewJSPTemplate 页面并且 input2 显示零值。

图 36.

6. 用例 3: 在本测试用例中对第一个输入参数提供有效参数，而对第二个参数提供无效参数（使用字符串而不是整数）。与上一用例类似，您不会看到错误，因为错误处理会将第二个输入参数更改为零。输入以下 URL:

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd?
     input1=abc&input2=abc
```

此命令的结果是显示 MyNewJSPTemplate 页面并且 input2 显示零值。

图 37.

7. 用例 4: 在本用例中这两个 URL 输入参数都使用有效参数。输入以下 URL:

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd?
     input1=abc&input2=1000
```

此命令的结果是显示 MyNewJSPTemplate 页面并且向用户显示 1000 点。


aa.jsp - Microsoft Internet Explorer provided by Sympatico
File Edit View Favorites Tools Help
Back Search Favorites Media
Address ost/webapp/wcs/stores/servlet/MyNewControllerCmd?input1=abc&input2=1000
Go Links
世界，您好！
程序员指南
教程：创建新的业务逻辑
从控制器命令发出的“参数 – 值”对列表
ControllerParm1=来自 IBM 的问候！
ControllerParm2=祝您愉快！
这是一个使用 JSP 标准标记库（JSTL）中的标记的例子。
MyNewView 由控制命令调用
MyNewView 由控制命令调用 —
com.ibm.commerce.sample.commands.MyNewControllerCmdImpl
UserName= abc
Points= 1000
Discussions
Subscribe...
Discussions not available on https://localhos
Done
Local intranet

图 38.

创建新任务命令

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、更多的特定任务。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。

MyNewControllerCmdImpl 当前不调用任何任务命令。此部分分成两个步骤。在第一个步骤中，创建新的任务命令。在第二个步骤中，修改控制器命令的 performExecute 方法来调用此新的任务命令。

在此步骤中，您将创建一个新任务命令接口及其相关联的实现类。起初，新任务命令除了处理视图参数之外做得很少。它只有用于 defaultCommandClassName、URL 参数和当前奖金点数值的字段。它有一个用于获取当前奖金点数值的方法。

在教程的此步骤中，您将学习如下内容:

- 如何创建一个新任务命令接口及其关联的实现类
- 任务命令中所必需的最少代码量
- 如何向任务命令添加字段和方法

创建 MyNewTaskCmd

此步骤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，您将有一个新任务命令，称为 MyNewTaskCmd。此命令用于您的基于“时尚潮流”样本的商店。

要创建 MyNewTaskCmd，请执行以下操作:

1. 切换至 Java 视图并选择 **WebSphereCommerceServerExtensionsLogic** 项目。
2. 展开 **src** 目录，然后展开 **com.ibm.commerce.sample.commands** 目录。
3. 双击 **MyNewTaskCmd.java** 接口以查看其源代码。
4. 在本接口的源代码中，取消对第 1 部分的注解以创建一个字段，该字段指定接口使用的缺省实现类。这将以下代码引入到接口中:

```
/// Section 1 ////////////////////////////////////////////////

// set default command implement class
```

```
 static final String defaultCommandClassName=
         "com.ibm.commerce.sample.commands.MyNewTaskCmdImpl";

/// End of section 1/////////////////////////////////////////
```

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。

5. 下一步，取消对第 2 部分的注解以创建将用于 MyNewTaskCmdImpl 实现类的 getter 和 setter 方法。这些方法用于与以下类型的信息相对应的字段:
   - 客户的用户标识。
   - 奖金点数值
   - 问候消息

   通过取消对第 2 部分的注解，将以下代码引入接口:

```
/// Section 2 //////////////////////////////////////////////
// set interface methods

 public void setInputUserName(java.lang.String inputUserName);
 public void setInputPoints(Integer inputPoints);
 public void setGreetings(java.lang.String greeting);

 public java.lang.String getInputUserName();
 public java.lang.Integer getInputPoints();
 public java.lang.String getGreetings();


/// End of section 2/////////////////////////////////////////
```

6. 保存更改。
7. 双击 **MyNewTaskCmdImpl.java** 实现类以查看其源代码。
8. 取消对第 1A、1B 部分的注解以创建字段及其在实现类中相应的 getter 和 setter 方法。这将以下代码引入到类中:

```
//// Section 1A ////////////////////////////////////////////

 private java.lang.String inputUserName;
 private java.lang.String greetings;
 private java.lang.Integer inputPoints;


////End of Section 1A ////////////////////////////////////////


//// Section 1B ////////////////////////////////////////////

 public void setInputUserName(java.lang.String newInputUserName) {
  inputUserName = newInputUserName;
```

```
}

public void setInputPoints(Integer newInputPoints) {
 inputPoints = newInputPoints;
}

public void setGreetings(java.lang.String newGreetings) {
 greetings = newGreetings;
}

public java.lang.String getInputUserName() {
 return inputUserName;
}

public Integer getInputPoints() {
 return inputPoints;
}

public java.lang.String getGreetings() {
 return greetings;
}


////End of Section 1B ////////////////////////////////////////
```

保存工作。

9. 在“概览”视图中，选择 MyNewTaskCmdImpl 类的 **performExecute** 方法。
10. 在此 performExecute 方法的源代码中，取消对第 1 部分的注解，从而将如下代码引入到方法中:

```
/// Section 1 ////////////////////////////////////////////////

/// modify the greetings and see it in the NVP list

 setGreetings( "Hello ! " + getInputUserName() );

/// End of section 1 /////////////////////////////////////////
```

这更新了问候值。然后通过 getGreetings() 方法，问候值可用于其它对象。会将它添加到“名称 – 值”对（NVP）列表。

11. 保存工作。

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。以下步骤显示如何按此方式修改控制器命令:

1. 在 Java 视图中，双击 **MyNewControllerCmdImpl.java** 类
2. 在“概览”视图中，选择它的 **performExecute** 方法。
3. 在 performExecute 方法的源代码中，浏览到区域 4。

4. 取消对第 4A、4B 和 4C 部分的注解，从而将如下代码引入到方法中:

```
/// Section 4A
    /// see how the controller command call a task command

      MyNewTaskCmd cmd = null;

 try {

     cmd = (MyNewTaskCmd) CommandFactory.createCommand(
         "com.ibm.commerce.sample.commands.MyNewTaskCmd", getStoreId());

   // this is required for all commands
          cmd.setCommandContext(getCommandContext());

  /// set input parameters to task command
  cmd.setInputUserName(getUserName());
  cmd.setInputPoints(getPoints()); // change to Integer

/// End Section 4A ////////////////////////////////////

/// Section 4B ////////////////////////////////////

  /// invoke the command's performExecute method
   cmd.execute();

  /// retrieve output parameter from task command, then put it to
     /// response properties
  rspProp.put("taskOutputGreetings", cmd.getGreetings());

/// End Section 4B ////////////////////////////////////

/// Start Section 4C ////////////////////////////////////
 } catch (ECException ex) {
  /// throw the exception as is
          throw (ECException) ex;
 }

/// End Section 4C ////////////////////////////////////
```

第 4A 部分使用命令工厂创建新任务命令对象。它然后设置命令上下文，并设置命令任务的输入参数。在调用任务命令的 execute 方法前，4B 部分将调用用于访问控制目的的 validateParameters 方法。然后它检索任务命令的问候值。4C 部分是一个简单的异常捕获块。

5. 保存更改。
6. 通过用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 项目并选择**构建项目**来编译代码更改。

修改 MyNewJSPTemplate 来添加问候消息

在本步骤中，您将执行以下操作修改 MyNewJSPTemplate.jsp 文件以添加一个显示问候消息的新部分:

1. 如果 JSP 模板文件还未打开，请执行以下操作:
   a. 在 Web 透视图中，切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图，并展开 **Stores** Web 项目。
   b. 浏览到 Web Content*FashionFlow_name* 子文件夹。
   c. 同时突出显示 **MyNewJSPTemplate_All.jsp** 和 **MyNewJSPTemplate.jsp** 文件，然后用鼠标右键单击并选择**打开工具 > Page Designer**。
2. 把第 7 部分从 MyNewJSPTemplate_All.jsp 文件复制到 MyNewJSPTemplate.jsp 文件。这将以下文本引入到 JSP 模板中:

```
<!-- SECTION 7 -->

<fmt:message key="Greeting" bundle="${tutorial}" />
<c:out value="${taskOutputGreetings}"/> <br /> <br />

<!-- END OF SECTION 7 -->
```

3. 保存更改。

测试 MyNewTaskCmd

下一步是通过执行以下操作测试新的任务命令是否正常工作:

1. 切换至“服务器”视图（**视窗 > 打开视图 > 服务器**）。
2. 用鼠标右键单击 **WebSphereCommerceServer** 服务器并选择**启动**（或**重新启动**）。
3. 用鼠标右键单击 Stores\Web Content*FashionFlow_name* 目录下的 **index.jsp** 并选择**在服务器上运行**。
   商店主页显示在 Web 浏览器上。
4. 下一步，输入以下 URL:

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd?
     input1=abc&input2=1000
```

   将显示 MyNewJSPTemplate。它包含任务命令创建的问候消息。


aa.jsp - Microsoft Internet Explorer provided by Sympatico
File Edit View Favorites Tools Help
Back Search Favorites Media
Address st/webapp/wcs/stores/servlet/MyNewControllerCmd?input1=abc&input2=1000
Go Links
世界，您好！
程序员指南
教程：创建新的业务逻辑
从控制器命令发出的“参数 – 值”对列表
ControllerParm1= 来自 IBM 的问候！
ControllerParm2= 祝您愉快！
这是一个使用 JSP 标准标记库（JSTL）中的标记的例子。
MyNewView 由控制命令调用
MyNewView 由控制命令调用 —
com.ibm.commerce.sample.commands.MyNewControllerCmdImpl
UserName= abc
Points= 1000
Greeting= abc，您好！
Discussions
Subscribe...
Discussions not available on https://localhc
Done
Local intranet

图 39.

修改 MyNewTaskCmd

在教程的本步骤中，您将修改 MyNewTaskCmd 以确定包含在 URL 中的用户名是否为注册用户的用户名。还要修改 MyNewDataBean 以处理来自任务命令的字段（例如，用户名）。相应地，要修改 MyNewJSPTemplate 来显示用户是否已注册。

在教程的本部分中，您将学习如下内容:

- 如何使用 URL 参数并从您自己定制的代码访问现有的 WebSphere Commerce 信息

修改 MyNewControllerCmdImpl 以便为任务命令创建一个对象

为提高对象的有效使用，控制器命令将创建 UserRegistryAccessBean 对象实例变量。因此，此对象还可用于任务命令。通过执行此方法，任务命令不需要创建对象的独立实例。该 UserRegistryAccessBean 对象在稍后（在任务命令中）用于确定购物者是不是注册用户。

要修改 MyNewControllerCmdImpl，请执行以下操作:

1. 在 Java 视图中，双击 **MyNewControllerCmdImpl.java** 类以查看其源代码。
2. 在该类的主体中取消对第 2 部分的注解，从而将以下代码引入到类中:

```
/// Section 2 ////////////////////////////////////////////////
/// create a user registry accessbean resource instance variable

    private UserRegistryAccessBean rrb = null;

/// End of Section 2 /////////////////////////////////////////
```

3. 在“概览”视图中，选择 **performExecute** 方法。
4. 在 performExecute 方法的源代码中取消对 4D 和 4F 部分的注解，从而向任务命令传递实例变量然后使返回的用户标识在响应属性中可用。这将以下代码引入到方法中:

```
// Section 4D ///////////////////////////////////
/// pass rrb instance variable to the task command

  cmd.setUserRegistryAccessBean(rrb);

// End of section 4D ////////////////////////////


// Section 4F ///////////////////////////////////////
///using access bean to get information from database
  if (cmd.getFoundUserId() != null) {
   rspProp.put("taskOutputUserId", cmd.getFoundUserId());
  }
// End of section 4F ////////////////////////////
```

您将接收到一个错误，指出一些方法尚未定义，但这会在下一步修改任务命令时得到解决。

5. 保存工作。

修改新任务命令以供用户名验证

下一步您必须修改新任务命令以验证是否 URL 中的用户名是注册用户，如下:

1. 双击 **MyNewTaskCmd.java** 接口以查看其源代码。
2. 取消对第 3 部分的注解，从而将如下代码引入到接口中:

```
/// Section 3 ////////////////////////////////////////////////

 public void setFoundUserId(java.lang.String inputUserId);
 public java.lang.String getFoundUserId();

 public void setUserRegistryAccessBean(UserRegistryAccessBean rrb);

/// End of section 3/////////////////////////////////////////
```

3. 保存工作。
4. 双击 **MyNewTaskCmdImpl.java** 类以查看其源代码。取消对导入部分 1 的注解，从而把以下两个导入语句引入代码:

```
/// Import section 1 ////////////////////////////////////
import com.ibm.commerce.user.objects.*;
import com.ibm.commerce.sample.databeans.*;
///  End of Import section 1 ////////////////////////////////
```

5. 取消对第 2A、2B 部分的注解，从而创建新字段和与添加到接口的方法相对应的 getter 和 setter 方法。这将以下代码引入到类中:

```
//// Section 2A ////////////////////////////////////////////////

 private java.lang.String foundUserId = null;

 private UserRegistryAccessBean rrb = null;

////End of Section 2A ////////////////////////////////////////

//// Section 2B ////////////////////////////////////////////////

 public void setUserRegistryAccessBean(UserRegistryAccessBean newRRB) {
  rrb = newRRB;
 }

 public void setFoundUserId(java.lang.String newFoundUserId) {
  foundUserId = newFoundUserId;
 }

 public java.lang.String getFoundUserId() {
  return foundUserId;
 }
```

```
/// End of section 2B ////////////////////////////////////////
```

6. 在“概览”视图中，选择 **validateParameters** 方法并检查其源代码。取消对第 1 部分的注解，从而将如下代码引入到方法:

```
// section 1 ////////////////////////////////////////////////

// use UserRegistryAccessBean to check user Id

 try {

   if (rrb!=null){
     setFoundUserId(rrb.getUserId());
     } else {
     rrb =new UserRegistryAccessBean();
     rrb=rrb.findByUserLogonId(getInputUserName());
     setFoundUserId(rrb.getUserId());
   }

     } catch (javax.ejb.FinderException e) {
    return;

} catch (java.rmi.RemoteException e) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
    this.getClass().getName(), "validateParameters");
} catch (javax.naming.NamingException e) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
    this.getClass().getName(), "validateParameters");
  } catch (javax.ejb.CreateException e) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
    this.getClass().getName(), "validateParameters");
  }

// end of section 1 ////////////////////////////////////////////
```

7. 保存工作。
8. 通过用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 项目并选择**构建项目**来编译代码更改。

修改 MyNewJSPTemplate 以供用户名验证

必须修改当前的 JSP 模板以显示用户名验证信息。要修改此文件，请执行以下操作:

1. 切换至 Web 视图。
2. 同时打开 **MyNewJSPTemplate_All.jsp** 和 **MyNewJSPTemplate.jsp** 文件。
3. 把第 8 部分从 MyNewJSPTemplate_All.jsp 文件复制到 MyNewJSPTemplate.jsp 文件。这将以下文本引入到 JSP 模板中:

```
<!-- SECTION 8 -->

<c:if test="${!empty taskOutputUserId}">
   <fmt:message key="UserId" bundle="${tutorial}" />
   <c:out value="${taskOutputUserId}"/> <br />
   <fmt:message key="FirstInput" bundle="${tutorial}" />
   <b><c:out value="${userName}"/></b>
   <fmt:message key="RegisteredUser" bundle="${tutorial}" /> <br />
   <fmt:message key="ReferenceNumber" bundle="${tutorial}" />
   <b><c:out value="${taskOutputUserId}"/></b> <br /> <br />
</c:if>

<c:if test="${empty taskOutputUserId}">
   <fmt:message key="FirstInput" bundle="${tutorial}" />
   <b><c:out value="${userName}"/></b>
   <fmt:message key="NotRegisteredUser" bundle="${tutorial}" /> <br />
</c:if>


<!-- END OF SECTION 8 -->
```

4. 保存对 MyNewJSPTemplate.jsp 文件所作的更改。

测试用户名验证

要测试用户名验证逻辑，请执行以下操作:

1. 切换至“服务器”视图（**视窗 > 打开视图 > 服务器**）。
2. 用鼠标右键单击 **WebSphereCommerceServer** 服务器并选择**启动**（或**重新启动**）。
3. 用鼠标右键单击 Stores\Web Content*FashionFlow_name* 目录下的 **index.jsp** 并选择**在服务器上运行**。
   商店主页显示在 Web 浏览器上。
4. 通过执行以下操作，创建新的注册用户:
   a. 单击**注册**。
   b. 再次单击**注册**来创建新客户。
   c. 在注册表单中，在所有必填字段中输入相应的值。例如，在电子邮件字段中，输入 tester@mycompany。记下电子邮件地址的值:

      _____。

   d. 一旦输入了值，请单击**提交**。
5. 下一步，输入有效的用户名作为 input1 的值。输入以下 URL:

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd?
   input1=new_e-mail&input2=1000
```

其中 *new_e-mail* 是在步骤 4 中创建的用户的电子邮件地址。将显示 MyNewJSPTemplate。它现在应该指示 input1 的值是有效用户名。

图 40.

6. 下一步，输入以下 URL，其中的用户名值无效:

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd?
     input1=abc&input2=1000
```

将显示一般异常。如果您查看页面源，会发现发生了 _ERR_FINDER_EXCEPTION。这是由于提供的用户名未与注册用户相对应。

创建新的实体 bean

本部分描述如何创建新的实体 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。根据 WebSphere Commerce 程序设计模块，一旦创建了数据库表，您必须创建实体 bean 来访问数据。

创建 XBONUS 表

在创建实体 bean 的准备中，必须首先创建新的数据库表。要创建的表名为 XBONUS。

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 > 命令行工具 > 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 developmentDB user dbuser using dbpassword;
create table XBONUS (MEMBERID BIGINT NOT NULL,
    BONUSPOINT INTEGER NOT NULL,
           constraint p_xbonus primary key (MEMBERID),
    constraint f_xbonus foreign key (MEMBERID)
    references users (users_id) on delete cascade)
```

其中

- *developmentDB* 是开发数据库的名称
- *dbuser* 是数据库用户
- *dbpassword* 是数据库用户的密码

单击“执行”图标。
您应该看到一条消息，指示 SQL 语句已成功完成。

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 XBONUS (MEMBERID NUMBER NOT NULL,
    BONUSPOINT INTEGER NOT NULL,
           constraint p_xbonus primary key (MEMBERID),
    constraint f_xbonus foreign key (MEMBERID)
 references users (users_id) on delete cascade);
```

   并按 Enter 键运行 SQL 语句。XBONUS 表已创建。
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

一旦已经创建表，就已准备好开始创建新的实体 bean。下一步使用 WebSphere Studio Application Developer 创建该 bean。

下一步通过执行以下操作创建新 Bonus bean:

1. 在 WebSphere Studio Application Developer 中，切换至 J2EE 视图。
2. 在“J2EE 层次结构”视图中，展开 **EJB 模块**。
3. 用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 模块并选择 **新建 > 企业 bean**。
   “企业 bean 创建”向导打开。
4. 从 **EJB 项目**下拉列表，选择 **WebSphereCommerceServerExtensionsData** 并单击**下一步**。
5. 请在“创建企业 bean”窗口执行以下操作:
   a. 选择**带有受容器管理的持久性（CMP）字段的实体 bean**
   b. 在 **bean 名称**字段中输入 Bonus。
   c. 在**源文件夹**字段中，保留指定的缺省值（ejbModule）。
   d. 在**缺省数据包**字段中，输入 com.ibm.commerce.extension.objects。
   e. 单击**下一步**。
6. 请在“企业 bean 详细信息”窗口中执行以下操作:
   a. 单击**添加**以便在 BONUS 表的 MEMBERID 和 BONUSPOINT 列中添加新的 CMP 属性。
      “创建 CMP 属性”窗口打开。在此窗口中，请执行以下操作:
      1) 在**名称**字段，输入 memberId。
      2) 在**类型**字段，输入 java.lang.Long。

注: 您必须使用 *java.lang.Long* 数据类型，而不是 *long* 数据类型。

3) 选择**键字段**复选框。
4) 单击**应用**。
5) 在**名称**字段，输入 `bonusPoint`。
6) 在**类型**字段，输入 `java.lang.Integer`。

注: 您必须使用 *java.lang.Integer* 数据类型，而不是 *integer* 数据类型。

7) 选择**以 getter 和 setter 方法访问**复选框。
8) 清除**将 getter 和 setter 方法升级到远程界面**复选框。**使 getter 方法只读**复选框就会不可用。
9) 单击**应用**。
10) 单击**关闭**以关闭此窗口。

b. 清除**对键类使用单一键属性类型**复选框，然后单击**下一步**。

7. 在“EJB Java 类详细信息”窗口，请执行以下操作:
   a. 要选择该 bean 的超类，请单击**浏览**。
   “类型选择”窗口打开。
   b. 在**使用以下任意方法选择一个类**字段中，输入 `ECEntityBean` 并单击**确定**。
   这选择了 com.ibm.commerce.base.objects.ECEntityBean 为超类。
   c. 单击**添加**来指定远程接口应该扩展的接口。“类型选择”窗口打开。
   d. 在**使用以下任意方法选择一个类**字段中，输入 `Protectable` 并单击**确定**。
   这选择了 com.ibm.commerce.security.Protectable。为了在访问控制下保护新资源，该接口是必需的。
   e. 单击**完成**。

通过执行以下操作设置新 bean 的隔离级别:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**。
2. 双击 **WebSphereCommerceServerExtensionsData** 项目来使用部署描述符编辑器打开它。作为在该编辑器中打开该文件的一种备用方法，可执行以下操作:
   a. 在 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图中，展开 **WebSphereCommerceServerExtensionsData > ejbModule > META-INF**。
   b. 用鼠标右键单击 **ejb-jar.xml** 并选择**打开工具 > 部署描述符编辑器**）
3. 单击**访问**选项卡。

4. 单击“隔离级别”文本框旁的**添加**。
   “添加隔离级别”窗口打开。
5. DB2 选择**可重复读**，然后单击**下一步**。
   Oracle 选择**已提交的读**，然后单击**下一步**
6. 从**已找到的 bean** 列表中，选择 **Bonus** bean，然后单击**下一步**。
7. 从**已找到的方法**列表中，选择 **Bonus** 来选中其所有方法，然后单击**完成**。
8. 保存工作（Ctrl+S），然后保持编辑器打开。

然后通过执行以下操作设置该 bean 的安全性身份:

1. 在部署描述符编辑器中，确保已选择“访问”选项卡。
2. 单击“安全性身份”文本框旁的**添加**。
   “添加安全性身份”窗口打开。
3. 选择**使用 EJB 服务器的身份**，然后单击**下一步**。
4. 从**已找到的 bean** 列表中，选择 **Bonus** bean，然后单击**下一步**。
5. 从**已找到的方法**列表中，选择 **Bonus** 来选中其所有方法，然后单击**完成**。
6. 保存您的工作（Ctrl+S）并保持编辑器打开。

然后通过执行以下操作为 bean 中的方法设置安全性角色:

1. 在部署描述符编辑器中，选择“Assembly 描述符”选项卡。
2. 在“方法许可权”部分，单击**添加**。
3. 选择 **WCSecurityRole** 作为安全角色并单击**下一步**。
4. 从找到的 bean 列表中，选择 **Bonus** 并单击**下一步**。
5. 在“方法”元素页面，单击**应用到所有**，然后单击**完成**。
6. 保存工作（Ctrl+S）然后关闭部署描述符编辑器。

下一步是除去一些与 WebSphere Studio Application Developer 生成的实体上下文相关的字段和方法。需要删除这些字段的原因是 ECEntityBean 基类对于这些方法会提供其自己的实现。要删除已生成的实体上下文字段和方法，请执行以下操作:

1. 在“J2EE 层次结构”视图中，展开 **WebSphereCommerceServerExtensionsData** 项目。
2. 展开 **Bonus** bean，然后双击 **BonusBean** 类。
3. 在“概览”视图中，请执行以下操作:
   a. 用鼠标右键单击 **myEntityCtx** 字段并选择**删除**。
   b. 用鼠标右键单击 **getEntityContext()** 方法，并选择**删除**。

c. 用鼠标右键单击 **setEntityContext(EntityContext)** 方法，并选择**删除**。

d. 用鼠标右键单击 **unsetEntityContext()** 方法并选择**删除**。

4. 保存工作（Ctrl+S）。保持 BonusBean 类打开。

然后，通过执行以下操作，将新的 getMemberId 方法添加到企业 bean:

1. 查看 **BonusBean** 类的源代码。
2. 将以下代码添加到该类的结束处（仍在该类内）:

```
public java.lang.Long getMemberId() {
   return memberId;
}
```

3. 必须通过执行以下操作将新的方法添加到远程接口:

   a. 在“概览”视图中，用鼠标右键单击 **getMemberId** 方法并选择**企业 Bean > 提升至远程接口**。一旦该步骤完成，在方法旁就会显示一个小的 R 图标，指示已升级到远程接口。

4. 保存工作。
5. 关闭 BonusBean 编辑器。

下一步，通过执行以下操作，将新的 FinderHelper 方法添加至 BonusHome 接口:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**。
2. 双击 **WebSphereCommerceServerExtensionsData** 项目以打开 EJB 部署描述符编辑器。
3. 单击 **bean** 选项卡。
4. 在 bean 窗格中，选择 **Bonus** bean，然后在右窗格中向下滚动并展开 **WebSphere 扩展**。
5. 单击**查找程序**文本框旁的**添加**。
   “添加查找程序描述符”窗口打开。
6. 选择**新建**，然后在**名称**字段中，输入 `findByMemberId`。
7. 单击**参数**文本框旁的**添加**，然后执行以下操作:

   a. 在**名称**字段，输入 `memberId`。

   b. 在**类型**字段输入 `java.lang.Long`。

   c. 单击**确定**。

8. 从**返回类型**下拉列表中选择 **com.ibm.commerce.extension.objects.Bonus**，然后单击**下一步**。
9. 从**查找程序类型**下拉列表中选择 **WhereClauseFinderDescriptor**。
10. 在**查找程序语句**字段中输入 `T1.MEMBERID = ?`，然后单击**完成**。
11. 保存工作，然后关闭 EJB 部署描述符编辑器。

下一步，通过执行以下操作向 Bonus bean 添加新 ejbCreate 方法:

1. 在“J2EE 层次结构”视图中，双击 **BonusBean** 类以打开它并查看其源代码。
2. 创建新 ejbCreate(Long, Integer) 方法，方法是向类添加以下代码:

```
public com.ibm.commerce.extension.objects.BonusKey ejbCreate(
   java.lang.Long memberId,java.lang.Integer bonusPoint)
   throws javax.ejb.CreateException {
 _initLinks();
      this.memberId=memberId;
      this.bonusPoint=bonusPoint;
      return null;
}
```

3. 保存代码更改。
4. 必须向主接口添加新 ejbCreate(Long, Integer) 方法。这将使方法在生成的访问 bean 中可用。要向主接口添加方法，请执行以下操作:
   a. 在“概览”视图中，用鼠标右键单击 **ejbCreate(Long, Integer)** 方法并选择**企业 Bean > 提升至人机接口**。

下一步通过执行以下操作创建新 ejbPostCreate(Long, Integer) 方法从而使它拥有一些与 ejbCreate(Long, Integer) 方法相同的参数:

1. 双击 **BonusBean** 类以打开它并查看其源代码。
2. 创建新 ejbPostCreate(Long, Integer) 方法，方法是向类添加以下代码:

```
public void ejbPostCreate(java.lang.Long memberId,
   java.lang.Integer bonusPoint)
   throws javax.ejb.CreateException
   {
   }
```

3. 保存代码更改。

接下来的步骤将向 Bonus bean 添加新的方法，从而使它可以受 WebSphere Commerce 访问控制系统的保护。通过执行以下操作，getOwner 和 fulfills 方法将添加到 bean:

1. 双击 **BonusBean** 类以打开它并查看其源代码。
2. 通过把以下代码添加到类结束处来向 BonusBean 类添加新的 getOwner 方法:

```
public java.lang.Long getOwner()
   throws java.lang.Exception {
      return getMemberId();
   }
```

3. 保存工作。
4. 通过把以下代码添加到类结束处来添加新的 fulfills 方法:

```
public boolean fulfills(Long member, String relationship)
   throws java.lang.Exception {
      if (relationship.equalsIgnoreCase("creator")) {
```

```
            return member.equals(getMemberId());
        }
        return false;
    }
```

5. 保存工作并关闭 bonus bean 编辑器。

下一步是把 XBONUS 表映射至 BonusBean 实体 bean。将使用“在中间会面”（Meet-in-the-middle）映射。要创建映射,请执行以下操作: Business Professional

1. 在“J2EE 层次结构”视图中，用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 并选择**生成 > EJB 到 RDB 映射**。
   “EJB 到 RDB 映射”窗口打开。
2. 选择**在中间会面**并单击**下一步**。
3. 在“数据库连接”窗口中，请执行以下操作:
   a. 在**连接名称**字段，输入 WebSphereCommerceServerExtensionsData
   b. 在**数据库字段**，输入开发数据库的名称。
   c. 在**用户标识**字段，输入数据库用户标识。
   d. 在**密码**字段，输入数据库用户密码。
   e. 从**数据库供应商类型**下拉列表，选择开发数据库的数据库供应商类型。
      - DB2 DB2 通用数据库 8.1
      - Oracle Oracle 9i
   f. Oracle 在**主机**字段中输入数据库服务器的全限定主机名。例如，输入 dbserver.yourcompany.com
   g. Oracle 在“类位置”字段，输入 classes12.zip 文件的位置。例如，输入 D:\oracle\ora92\jdbc\lib\classes12.zip
   h. 单击**下一步**。一旦连接已建立，就会显示数据库中的表的列表。您也可以稍后通过查看“数据”视图中的“数据库服务器”视图来查看连接文档。
4. 选择 **XBONUS** 表，然后单击**下一步**。
5. 选择**按名称和类型匹配**然后单击**完成**。映射编辑器已打开。
6. Oracle 用鼠标右键单击 XBONUS 表并选择**打开表编辑器**。在表编辑器中，请执行以下操作:
   a. 选择**列**选项卡。
   b. 选择 BONUSPOINT 列，并将列类型从 NUMBER 改为 INTEGER
   c. 保存更改。

7. 在“企业 bean”窗格中，展开 **Bonus** bean。在“表”窗格，展开 **XBONUS** 表。
8. 把 Bonus bean 中的字段映射为 XBONUS 表中的列，方法是执行以下操作:
   a. 用鼠标右键单击 Bonus bean 并选择**按名称匹配**。
9. 保存 Map.mapxmi 文件（Ctrl+S）并关闭该文件。
10. Oracle 您必须使用文本编辑器编辑表定义，方法如下:
    a. 使用文本编辑器打开 XBONUS.xmi 文件。
    b. 将出现的所有 `SQLNumeric6` 替换为 `SQLNumeric3`。
    c. 保存更改。

Express

1. 打开“数据”视图并切换至“数据定义”视图。
2. 浏览至以下目录:
   WebSphereCommerceServerExtensionsData > **ejbModule** > **META-INF**。
3. 右键单击 **META-INF** 并选择**新建数据库定义**。“新建数据库定义”向导打开。
4. 在**数据库名**字段，输入开发数据库的名称。例如，输入 `Demo_Dev`。
5. 从**数据库供应商类型**下拉列表中选择 **DB2 通用数据库精简版版本 8.1** 并单击**完成**。新数据库定义已创建。
6. 浏览至以下目录:
   WebSphereCommerceServerExtensionsData > **ejbModule** > **META-INF** > **Schema >** *DevelopmentDB*。
7. 右键单击 **DevelopmentDB** 并选择**新建 > 新建模式定义**。显示“新建模式定义”窗口。
8. 在**模式名称**字段中输入 `NULLID` 并单击**完成**。
9. 浏览至以下目录:
   WebSphereCommerceServerExtensionsData > **ejbModule** > **META-INF** > **Schema >** *DevelopmentDB* **> NULLID** 并选择**新建 > 新建数据库定义**。“新建表定义”向导打开。
10. 在**表名**字段中输入 `XBONUS` 并单击**下一步**。
11. 向表定义添加键列，如下:
    a. 单击**添加另一个**以添加 MEMBERID 列。
    b. 在**列名**字段中输入 `MEMBERID`。
    c. 选择**键列**。

d. 从**列类型**下拉列表中选择以下内容:

BIGINT

NUMBER

12. 单击**添加另一个**以添加 BONUSPOINT 列。
13. 在**列名**字段中输入 BONUSPOINT。
14. 从**列类型**下拉列表中选择 **INTEGER** 并单击**完成**。
15. 切换到“J2EE 透视图”，并在“J2EE 层次结构”视图中选择 **EJB 模块** > **WebSphereCommerceServerExtensionsData**。
16. 右键单击 **WebSphereCommerceServerExtensionsData** 并选择**新建** > **访问 bean**。显示“添加访问 bean”窗口。
17. 选择**复制助手**并单击**下一步**。
18. 选择 **Bonus** bean 并单击**下一步**。
19. 从**构造方法**选择框中选择 **findByPrimaryKey(com.ibm.commerce.extension.objects.Bonus)** 并单击**完成**。
20. 在“J2EE 层次结构”视图中选择 **EJB 模块** > **WebSphereCommerceServerExtensionsData**。
21. 右键单击 **WebSphereCommerceServerExtensionsData** 并选择**打开方式** > **部署描述符编辑器**。
22. 向下滚动至“JNDI - 缺省值”部分并确保**数据源 JNDI 名称**的值为空白。如果该值为 **jdbc/Default**，则删除 **jdbc/Default** 并按 Crtl+S 以保存更改。
23. 在“J2EE 层次结构”视图中展开 **EJB 模块**，然后用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 并选择**生成** > **部署和 RMIC 代码**。
24. 选择 **Bonus** bean 并单击**完成**。

下一步是修正模式名以使新 bean 可移植到其它数据库。要进行此修改，请执行以下操作:

1. 在 J2EE 视图中，切换至“J2EE 层次结构”视图。
2. 展开**数据库**，然后展开 **WebSphereCommerceServerExtensionsData**。
3. 用鼠标右键单击模式节点（例如，DB2USER）然后选择**重命名**。
4. 把值设置为 NULLID。

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。WebSphere Studio Application Developer 中的工具用于生成此访问 bean,

这取决于您已经创建的实体 bean（特别的是，仅已经提升到远程接口的方法可用于访问 bean）。要创建 Bonus 实体 bean 的访问 bean，请执行以下操作:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**，然后用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 并选择**新建 > 访问 bean**。“添加访问 bean”窗口打开。
2. 选择**复制帮手**并单击**下一步**。
3. 选择 **Bonus** bean 并单击**下一步**。
4. 从构造函数方法下拉列表，选择 **findByPrimaryKey(com.ibm.commerce.extension.objects.BonusKey)** 作为构造函数方法。
5. 选择“属性帮手”部分的所有属性。
6. 单击**完成**。

可以通过切换到 Business Professional “J2EE 导航器”选项卡 Express “项目导航器”选项卡、展开以下目录来查看新生成的代码:
WebSphereCommerceServerExtensionsData >ejbModule >com.ibm.commerce.extension.objects
会创建名为 BonusAccessBean 的新类和名为 BonusAccessBeanData 的新接口，并显示在数据包中。

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。它还生成通过 IIOP 进行的 RMI 访问所必需的 Java ORB、存根和 tie 类以及主接口和远程接口的存根

要生成部署代码，请执行以下操作: Business Professional

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**，然后用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 并选择**生成 > 部署和 RMIC 代码**。
2. 选择 **Bonus** bean 并单击**完成**。

Express

1. 在“J2EE 层次结构”视图中展开 **EJB 模块** > **WebSphereCommerceServerExtensionsData**。

2. 右键单击 **WebSphereCommerceServerExtensionsData** 并选择**打开方式 > 部署描述符编辑器**。
3. 向下滚动至“JNDI - 缺省值”部分并确保**数据源 JNDI 名称**的值为空白。如果该值为 **jdbc/Default**，则删除 **jdbc/Default** 并按 Crtl+S 以保存更改。

您可以通过切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图来查看新生成的代码。您会看到如下内容:

表 11.

代码类型	类名
容器实现生成代码	EJSCMPBonusHomeBean.java
	EJSRemoteCMPBonus.java
	EJSRemoteCMPBonusHome.java
	EJSFinderBonusBean.java
JDBC 访问代码	EJSJDBCPersisterCMPBonusBean.java
RMI 约束和存根代码	_EJSRemoteCMPBonus_Tie.java
	_Bonus_Stub.java
	_EJSRemoteCMPBonusHome_Tie.java
	_BonusHome_Stub.java

下一步是要通过执行以下操作使用通用测试客户机来测试新企业 bean:

1. 切换至“服务器”视图。
2. 在“服务器配置”视图中，双击 **WebSphereCommerceServer** 服务器并单击**配置**选项卡。
3. 选择**启用通用测试客户机**。保存更改。
4. 在“服务器”视图，用鼠标右键单击 **WebSphereCommerceServer** 服务器并选择**启动**。
5. 在“J2EE 层次结构”中，展开 **EJB 模块 > WebSphereCommerceServerExtensionsData**。
6. 用鼠标右键单击 **Bonus** bean 并选择**在服务器上运行**。
   将打开 IBM 通用测试客户机。
7. 在左窗格中，单击 **Bonus**，然后单击 **Bonus 主页**。
8. 单击 **Bonus create(Long, Integer)** 方法。
9. 在右窗格的 **Long** 字段中，输入 -1000 并在 **Integer** 字段中输入 1000。
10. 单击**调用**，结果就会显示在底端窗格内。

11. 单击**处理对象**以向“引用”窗格添加远程接口，然后您就会在“EJB 引用”下看到输入的值。已在 BONUS 表中创建新记录。
12. 选择 **getMemberId** 方法并单击**调用**，结果 -1000 就会显示在底端窗格内。
13. 关闭测试客户机并停止服务器。

将 Bonus 实体 bean 与 MyNewControllerCmd 集成在一起

在前一部分中，您使用在 WebSphere Studio Application Developer 中生成的测试客户机测试了新的 Bonus 实体 bean。通过这样做，您确定了可以成功更新数据库信息。现在，将 Bonus 实体 bean 与 MyNewControllerCmd 逻辑集成在一起。一旦更新了 Java 代码，将更新 MyNewJSPTemplate.jsp 文件来创建一个允许对客户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 MyNewTaskCmd 任务命令以包含奖金点数的字段和方法；更新 validateParameters 方法；以及添加逻辑以更新用户的奖金点数余额。
2. 将 getResources 方法添加到 MyNewControllerCmdImpl 类，以返回命令使用的资源的列表。此方法附带包含，以用于访问控制目的。
3. 创建新的 BonusDataBean，以便奖金点数可在 JSP 模板中显示。
4. 对新资源创建新的访问控制策略。
5. 修改 MyNewJSPTemplate.jsp 模板以使您可输入用户的奖金点数并显示该用户的奖金点数新余额。

修改 MyNewTaskCmd 接口以包含奖金点数

在本步骤中，您将通过执行以下操作修改 MyNewTaskCmd 接口以指定奖金点数的必需字段和方法:

1. 切换至 Java 视图并展开 **WebSphereCommerceServerExtensionsLogic** 项目。
2. 展开 **com.ibm.commerce.sample.commands\src** 目录。
3. 双击 **MyNewTaskCmd** 接口以查看其源代码。
4. 取消对导入第 2 部分的注解以包含以下数据包:

```
/// Import section 2 ////////////////////////////////////
import com.ibm.commerce.extension.objects.*;
/// End of import section 2 ////////////////////////////
```

5. 取消对第 4 部分的注解，从而将如下代码引入到方法:

```
/// Section 4 ///////////////////////////////////////////////

 public java.lang.Integer getOldBonusPoints();
 public Integer getTotalBonusPoints();
```

```
public void setBonusAccessBean(BonusAccessBean bb);
public BonusAccessBean getBonusAccessBean();


/// End of section 4/////////////////////////////////////////
```

6. 保存更改。

修改 MyNewTaskCmdImpl 以计算奖金点数

MyNewTaskCmdImpl 用作 Bonus 实体 bean 和 MyNewControllerCmd 之间的集成点（因为 MyNewControllerCmd 调用 MyNewTaskCmd）。

要修改 MyNewTaskCmdImpl 以计算奖金点数，请执行以下操作:

1. 选择 **MyNewTaskCmdImpl** 类以查看其源代码。
2. 取消对导入第 2 部分的注解以引入以下数据包:

```
/// Import section 2 ////////////////////////////////////
import com.ibm.commerce.extension.objects.*;
///  End of Import section 2 ////////////////////////////////
```

3. 取消对第 3A 和 3B 部分的注解，以将以下代码引入到类中:

```
//// Section 3A ////////////////////////////////////////////

 private java.lang.Integer oldBonusPoints;
 private java.lang.Integer totalBonusPoints;


 private BonusAccessBean bb = null;

////End of Section 3A ////////////////////////////////////////

//// Section 3B ////////////////////////////////////////////

 public void setBonusAccessBean(BonusAccessBean newBB) {
  bb = newBB;
 }

 public BonusAccessBean getBonusAccessBean(){
  return bb;
 }

    public java.lang.Integer getOldBonusPoints() {
  return oldBonusPoints;
 }

 public Integer getTotalBonusPoints(){
  return totalBonusPoints;
 }


/// End of section 3B ////////////////////////////////////////
```

4. 在“概览”视图中，选择 **validateParameters** 方法并取消对第 2 部分的注解，以将如下代码引入到该方法中:

```
// section 2 ////////////////////////////////////////////////////

 try {
    oldBonusPoints = bb.getBonusPoint();
     } catch (javax.ejb.FinderException e) {
  try {
   // If bb is null, create a new instance
    bb = new BonusAccessBean(new Long(foundUserId), new Integer(0));
    oldBonusPoints = new Integer(0);
     } catch (javax.ejb.CreateException ec) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
     this.getClass().getName(), "validateParameters");
     } catch (javax.naming.NamingException ec) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
     this.getClass().getName(), "validateParameters");
     } catch (java.rmi.RemoteException ec) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
     this.getClass().getName(), "validateParameters");
   }
} catch (javax.naming.NamingException e) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
    this.getClass().getName(), "validateParameters");
} catch (java.rmi.RemoteException e) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
    this.getClass().getName(), "validateParameters");
  } catch (javax.ejb.CreateException e) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
    this.getClass().getName(), "validateParameters");
  }

// end of section 2 ////////////////////////////////////////////////
```

5. 在“概览”视图中，选择 **performExecute** 方法。
6. 在此 performExecute 方法的源代码中，取消对第 2 部分的注解。这将以下代码引入到方法中:

```
/// use BonusAccessBean to update new bonus point
/// Section 2 ////////////////////////////////////////////////////

 int newBP =  oldBonusPoints.intValue() + getInputPoints().intValue();
 totalBonusPoints = new Integer (newBP);
 bb.setBonusPoint(totalBonusPoints)   ;

 try {
   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");
     }

   /// End of section 2 ////////////////////////////////////////////
```

7. 保存更改。

向 MyNewControllerCmdImpl 类添加 getResources 方法

在本部分中，您将新的 getResources 方法添加到 MyNewControllerCmdImpl。此方法返回命令在处理期间使用的资源的列表。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. 双击 **MyNewControllerCmdImpl** 类以打开并查看其源代码。
2. 在源代码中，取消对导入第 2 部分的注解，以将以下数据包引入到类中:

```
/// Import Section 2 ////////////////////////////
import com.ibm.commerce.extension.objects.*;
/// End of Import Section 2 ////////////////////////
```

3. 取消对第 3 部分的注解，以将以下代码引入到该类中:

```
/// Section 3 ////////////////////////////////////////////
/// Create an instance variable of type AccessVector to hold
/// the resources and a BonusAccessBean instance variable for
/// access control purposes.

private AccessVector resources = null;
    private BonusAccessBean bb = null;

/// End of Section 3 ////////////////////////////////////////
```

4. 在源代码中，取消对访问控制部分的注解。此段显示如以下代码片段中所示:

```
/// AccessControl Section ////////////////////////////////////

public AccessVector getResources() throws ECException {

   if (resources == null) {

    /// use UserRegistryAccessBean to check user reference number

 String refNum = null;
 String methodName = "getResources";

  rrb = new UserRegistryAccessBean();

 try {
   rrb = rrb.findByUserLogonId(getUserName());
```

```
            refNum = rrb.getUserId();
        } catch (javax.ejb.FinderException e) {
        throw new ECSystemException(ECMessage._ERR_FINDER_EXCEPTION,
           this.getClass().getName(),methodName,e);
    } catch (javax.naming.NamingException e) {
        throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
           this.getClass().getName(), methodName,e);
    } catch (java.rmi.RemoteException e) {
        throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
           this.getClass().getName(), methodName,e);
     } catch (javax.ejb.CreateException e) {
        throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
           this.getClass().getName(), methodName,e);
     }


    /// find the bonus bean for this registered user

       bb = new com.ibm.commerce.extension.objects.BonusAccessBean();
    try {
        if (refNum != null) {
       bb.setInitKey_memberId(new Long(refNum));
       bb.refreshCopyHelper();
       resources = new AccessVector(bb);
      }
        } catch (javax.ejb.FinderException e) {

      ///doesn't have a bonus object so return the container that
        ///will hold the bonus object when it's created
     resources = new AccessVector(rrb);
      return resources;

    } 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 resources;
    }

    /// End of AccessControl Section ////////////////////////////////
```

5. 保存更改。

修改 MyNewControllerCmdImpl 类的 performExecute 方法

在此步骤中如下修改 MyNewControllerCmdImpl 的 performExecute 方法中的代码，以包含与新 bonus bean 相关的代码:

1. 双击 **MyNewControllerCmdImpl** 类。
2. 在“概览”视图中，选择 **performExecute** 方法。
3. 在源代码中，取消对第 4E、4G 和 4H 部分的注解，以将以下代码引入到方法中:

```
// Section 4E ////////////////////////////////////
/// pass bb instance variable to the task command
    cmd.setBonusAccessBean(bb);
// End of section 4E ////////////////////////////

// Section 4G ////////////////////////////////////
if (cmd.getOldBonusPoints() != null) {
      rspProp.put("oldBonusPoints", cmd.getOldBonusPoints());
}
// End of section 4G ////////////////////////////

// Section 4H ////////////////////////////////////
///Instantiate the bonus data bean , then put it to response properties
 BonusDataBean bdb = new com.ibm.commerce.sample.databeans.BonusDataBean(
      cmd.getBonusAccessBean());
  rspProp.put("bdbInstance", bdb );
// End of section 4H ////////////////////////////////
```

4. 保存更改。

注: 您将看到错误，因为尚未定义 BonusDataBean。这些将在下一部分中得到修正。

创建 BonusDataBean 数据 bean

遵照编程模型，您应该创建一个与新 Bonus 实体 bean 相对应的新数据 bean。虽然不是所有的实体 bean 都必需有一个相应的数据 bean，但如果希望可以显示 JSP 模板中实体 bean 的信息，则出于此目的应创建一个新的数据 bean。

在此方案，将要求您创建一个新的用于扩展 BonusAccessBean 的 BonusDataBean 数据 bean。对于教程的其它部分，我们提供了基本代码，您需要取消对代码各部分的注解。

要把 BonusDataBean 引入您的代码，请执行以下操作:

1. 第一步是导入新数据 bean 的基本代码，如下:
   a. 切换到“Java 透视图”中的 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
   b. 展开 **WebSphereCommerceServerExtensionsLogic** 项目。
   c. 用鼠标右键单击 **src** 文件夹并选择**导入**。“导入”向导打开。
   d. 从**选择导入源**列表中，选择 **Zip 文件**并单击**下一步**。

e. 单击**浏览**（在 **Zip 文件**字段旁）并浏览至样本代码。此文件的位置如下:
yourDirectory\WC_SAMPLE_55.zip
其中 *yourDirectory* 是您将数据包下载至的目录。

f. 单击**全部不选**，然后展开目录并选择要导入的以下文件:

- com\ibm\commerce\sample\databeans\BonusDataBean.java

g. 在**文件夹**字段中，WebSphereCommerceServerExtensionsLogic/src 文件夹已经指定。保留该值。

h. 单击**完成**。

2. 双击 **BonusDataBean** 类以查看其源代码。
3. 在源代码中，取消对第 1 部分的注解，以将以下代码引入到该 bean 中:

```
/// Section 1 ////////////////////////////////////////////

// create fields and accessors (setter/getter methods)

 private java.lang.String userId;
 private java.lang.Integer totalBonusPoints;


 public java.lang.String getUserId() {
  return userId;
 }

 public void setUserId(java.lang.String newUserId) {
  userId = newUserId;

  ////////////////////////////////////
  /// Section A : instantiate BonusAccessbean

  if (userId != null)
   this.setInitKey_memberId(new Long(newUserId));

  ////////////////////////////////////
 }


    public java.lang.Integer getTotalBonusPoints() {
  return totalBonusPoints;
 }
 public void setTotalBonusPoints(java.lang.Integer newTotalBonusPoints) {
  totalBonusPoints= newTotalBonusPoints;
 }


//// End of section 1 ////////////////////////////////////////////
```

4. 下一步，取消对第 2 部分的注解，以将以下部分的代码引入到该 bean 中:

```
/// Section 2////////////////////////////////////////////

// create a new constructor for passing access bean into databean
```

```
// so that JSP can work with the access bean

public BonusDataBean(BonusAccessBean bb) throws com.ibm.commerce.
exception.ECException {
 try {
  super.setEJBRef(bb.getEJBRef());
     } catch (javax.ejb.FinderException e) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
     "BonusDataBean", "BonusDataBean(bb)");
} catch (javax.naming.NamingException e) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
     "BonusDataBean", "BonusDataBean(bb)");
} catch (java.rmi.RemoteException e) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
     "BonusDataBean", "BonusDataBean(bb)");
  } catch (javax.ejb.CreateException e) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
     "BonusDataBean", "BonusDataBean(bb)");
  }
}

//// End of section 2 ////////////////////////////////////////////
```

5. 下一步，取消对第 3 部分的注解，以将以下代码引入到该 bean 中:

```
//// Section 3 ////////////////////////////////////////////////

// set additional data field that is used for
// instantiating BonusAccessbean

 try {

  setUserId(getRequestProperties().getString("taskOutputUserId"));

  try {
   super.refreshCopyHelper();
     } catch (javax.ejb.FinderException e) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
     "BonusDataBean", "populate");
} catch (javax.naming.NamingException e) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
     "BonusDataBean", "populate");
} catch (java.rmi.RemoteException e) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
     "BonusDataBean", "populate");
  } catch (javax.ejb.CreateException e) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
     "BonusDataBean", "populate");
  }


 }
catch (ParameterNotFoundException e){}
```

```
///// End of Section 3 ////////////////////////////////////////
}
```

6. 下一步，取消对第 4 部分的注解，以将以下代码引入到该 bean 中:

```
/// Section 4 ///////////////////////////////////////////

// copy input TypedProperteis to local

 requestProperties = aParam;

/// End of section 4 ///////////////////////////////////
```

7. 保存更改。
8. 右键单击 **WebSphereCommerceServerExtensionsLogic** 项目并选择**重构项目**，来编译已更改的代码。

创建新实体 bean 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 tester@mycompany 用户登录，则该用户仅可修改他自己的奖金点数。

设置 AllUsersUpdateBonusResourceGroup 策略包含以下步骤:

1. 修改访问控制策略以反映您的环境。
2. 使用 acpload 命令装入 SampleACPolicy.xml 文件。
3. 使用 acpnlsload 命令装入 SampleACPolicy_en_US.xml 描述。

要为您的环境定制访问控制策略，请执行以下操作:

1. 要确定“时尚潮流”商店的成员标识值，请执行以下操作:

   - DB2 如果您正在使用 DB2 数据库，请执行以下操作:

     a. 连接开发数据库。

     b. 发出以下 SQL 语句:

        ```
        select member_id from storeent where storeent_id=FF_storeent_ID
        ```

        其中 *FF_storeent_ID* 是您的“时尚潮流”商店的商店实体标识。例如，您可以输入:

        ```
        select member_id from storeent where storeent_id=10001
        ```

        记下成员标识值: _____

   - Oracle 如果您正在使用 Oracle 数据库，请执行以下操作:

     a. 打开 Oracle SQL Plus 命令窗口并连接到您的开发数据库。

     b. 输入以下内容:
        `column member_id format 999999999999999999999999999999999999`
        (thiry-five 9's)
        这样将设置足够宽度的 member_id 列以看到完整值的显示。

     c. 发出以下 SQL 语句:

        ```
        select member_id from storeent where storeent_id=FF_storeent_ID
        ```

        其中 *FF_storeent_ID* 是您的“时尚潮流”商店的商店实体标识。例如，您可以输入:

        ```
        select member_id from storeent where storeent_id=10001
        ```

        记下成员标识值: _____

     d. 通过输入以下内容将 member_id 列的显示宽度复位:

        ```
        column member_id clear
        ```

2. 使用文本编辑器，打开 SampleACPolicy_template.xml 文件（在 *WCDE_installdir*\Commerce\xml\policies\xml 目录中）并把 *FashionFlowMemberId* 替换为步骤 1 中所确定的“时尚潮流”商店的成员标识值。把修改后的文件保存为 SampleACPolicy.xml（在相同的目录中）。

3. 使用文本编辑器，打开 SampleACPolicy_template_en_US.xml 文件（在 *WCDE_installdir*\Commerce\xml\policies\xml 目录中）并把 *FashionFlowMemberId* 替换为“时尚潮流”商店的成员标识值，如步骤 1 中所确定。把修改后的文件保存为 SampleACPolicy_en_US.xml（在相同的目录中）。

4. 在命令提示符下，切换至以下目录:
   WCDE_installdir\commerce\bin
5. 要装入 SampleACPolicy.xml 文件，您必须发出 acpload 命令，此命令具有以下格式:

   ```
   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。在此例中，输入 SampleACPolicy.xml

   例如，可以发出以下命令:

   ```
   acpload Demo_dev db2user db2user SampleACPolicy.xml
   ```

6. 要装入策略描述，必须发出 acpnlsload 命令，它具有以下格式:

   ```
   acpnlsload db_name db_user db_password inputXMLFile
   ```

   例如，可以发出以下命令:

   ```
   acpnlsload Demo_dev db2user db2user SampleACPolicy_en_US.xml
   ```

7. 如果用于测试环境的服务器当前正在运行，您可以使用 WebSphere Commerce 管理控制台的注册表刷新选项来更新访问控制注册表，如下:
   a. 打开 Web 浏览器并输入以下 URL:

      ```
      https://localhost/webapp/wcs/admin/servlet/ToolsLogon?XMLFile=
         adminconsole.AdminConsoleLogon
      ```

   b. 当得到提示时，用站点管理员标识登录。
   c. 选择在**站点**工作并单击**确定**。
   d. 从**配置**菜单中，选择**注册表**。
   e. 单击**更新所有**，过一会后，单击**刷新**以验证更新是否已完成。
   f. 注销并关闭管理控制台窗口。

修改 MyNewJSPTemplate.jsp 模板以包含奖金点数

要修改显示页，请执行以下操作:

1. 在 WebSphere Studio Application Developer 中，切换至 Web 视图。
2. 同时打开 **MyNewJSPTemplate_All.jsp** 和 **MyNewJSPTemplate.jsp** 文件。
3. 把第 9 部分从 MyNewJSPTemplate_All.jsp 文件复制到 MyNewJSPTemplate.jsp 文件。这将以下文本引入到 JSP 模板中:

```
<!-- SECTION 9 -->

<h2><fmt:message key="BonusAdmin" bundle="${tutorial}" /> </h2>

<c:if test="${!empty taskOutputUserId}">
 <ul>
  <li>
   <b>
   <fmt:message key="PointBeforeUpdate" bundle="${tutorial}" />
   <c:out value="${oldBonusPoints}"/>
   </b>
  </li>
  <li>
   <b>
   <fmt:message key="PointAfterUpdate" bundle="${tutorial}" />
      <c:out value="${bdbInstance.bonusPoint}" />
      </b>
     </li>
 </ul>
</c:if>


      <br />
<b><fmt:message key="EnterPoint" bundle="${tutorial}" /></b><p />


<form name="Bonus" action="MyNewControllerCmd">
<table>
 <tr>
  <td>
   <b>Logon ID </b>
  </td>
  <td>
   <input type="text" name="input1" value="<c:out
    value="${userName}"/>" />
  </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 9 -->
```

4. 保存 MyNewJSPTemplate.jsp 文件。

测试集成的 Bonus bean

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。

要测试新逻辑，请执行以下操作:

1. 切换至“服务器”视图。
2. 用鼠标右键单击 **WebSpherCommerceServer** 服务器并选择**启动**（或**重新启动**）。
3. 用鼠标右键单击商店的 **index.jsp** 文件并选择**在服务器上运行**。
   将显示商店主页。
4. 通过执行以下操作，以已注册的用户登录:
   a. 单击**注册**链接。
      将显示“注册”页面。
   b. 在**电子邮件地址**字段中，输入您在第 247 页的『测试用户名验证』中创建的用户的电子邮件地址。
   c. 在**密码**字段，输入该用户的密码并单击**登录**。
   d. 完成登录后，在同一个浏览器中输入以下 URL:

      ```
      http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd?
         input1=user_e-mail&input2=1000
      ```

      其中 *user_e-mail* 是您在第 247 页的『测试用户名验证』中创建的用户的电子邮件地址。将向您显示一个页面，其中包含了先前所有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）。

图 41.

e. 现在，在**登录标识**字段中输入用户的电子邮件地址，并在**奖金点数**字段中输入 500。单击**提交**。会显示与以下类似的页面，该页面显示已更新了奖金点数余额。

图 42.

部署奖金点数逻辑

本部分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“时尚潮流”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。

有许多不同类型的有用资源，它们必须部署到目标 WebSphere Commerce Server。它们包括:

- 控制器命令、任务命令和数据 bean 逻辑
- 企业 bean 逻辑
- JSP 模板和图像文件
- 属性文件和资源束
- 数据库更新包括模式更新（新表）以及命令注册表更新
- 访问控制更新

本节描述如何以*递增*方式将所有这些有用资源部署到目标 WebSphere Commerce Server。这是作为递增部署完成的，与对整个 EAR 文件进行的部署相对。

创建命令和数据 bean JAR 文件

本部分描述如何创建包含控制器命令、任务命令和数据 bean 逻辑的 JAR 文件。

要创建此 JAR 文件，请在开发机上执行以下步骤:

1. 在本地文件系统上创建名为 *drive*:\ExportTemp 的目录。
2. 在 WebSphere Studio Application Developer 中，切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
3. 用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 项目并选择**导出**。
   “导出”向导打开。
4. 在“导出”向导中，请执行以下操作:
   a. 选择 **JAR 文件**并单击**下一步**。
   b. **选择要导出的资源**左下窗格是以项目的名称预先填充的。保留此值不变。
   c. 在右窗格中，确保仅选择了以下资源:
      - .classpath
      - .project

- .serverPreference

d. 确保已选择**导出已生成的类文件和资源**。

e. 不要选择**导出 Java 源文件和资源**。

f. 在**选择导出目的地**字段中，输入要使用的全限定 JAR 文件名。在此例中，输入 *drive*:\ExportTemp\WebSphereCommerceServerExtensionsLogic.jar。请注意，JAR 文件名必须是 WebSphereCommerceServerExtensionsLogic.jar。

g. 单击**完成**。

创建 EJB JAR 文件

要创建 EJB JAR 文件，请执行以下操作:

1. 打开 WebSphere Studio Application Developer 并切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
2. 展开 **WebSphereCommerceServerExtensionsData** 项目。
3. 双击 **EJB 部署描述符**。
4. 在选择“概述”选项卡的情况下，滚动至窗格底部以找到 **WebSphere 绑定**部分。
5. 在**数据源 JNDI 名称**字段，输入目标 WebSphere Commerce Server 的数据源 JNDI 名称。以下是值的示例:

   DB2 `jdbc/WebSphere Commerce DB2 DataSource demo`

   其中目标 WebSphere Commerce Server 使用 DB2 数据库，WebSphere Commerce 实例名称是“demo”

   Oracle `jdbc/WebSphere Commerce Oracle DataSource demo`

   其中目标 WebSphere Commerce Server 使用 Oracle 数据库，WebSphere Commerce 实例名称是“demo”。

数据源 JNDI 名称的值是通过将“jdbc/”添加到目标 WebSphere Commerce Server 的数据源名称来创建的。可通过打开目标 WebSphere Commerce Server 上的 *instanceName*.xml 文件并在该文件中搜索 `DatasourceName=` 来验证数据源名称。

6. 保存部署描述符更改（Ctrl+S）。
7. 在 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图中，用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 项目，并选择**导出**。

   “导出”向导打开。
8. 在“导出”向导中，请执行以下操作:

a. 选择 **EJB JAR 文件**并单击**下一步**。

b. Business Professional **您想要导出什么资源?** 的值是以 EJB 项目的名称预填充的。

Express EJB 项目名称是预填充的。
保留该值不变。

c. Business Professional 在**您要将资源导出到哪里?** 字段中，输入要使用的全限定 JAR 文件名。

Express 对于目的地，输入要使用的全限定 JAR 文件名。
在这种情况下，输入
drive:\ExportTemp\WebSphereCommerceServerExtensionsData.jar。

d. 单击**完成**。

9. JAR 文件创建后，撤销步骤 5 中所作的对本地部署描述符的更改，用以恢复本地测试服务器所必需的设置。

注: 本教程假定您的开发数据库和目标 WebSphere Commerce Server 所使用的数据库是同一种类型。如果部署到不同的数据库类型，您要遵循第 190 页的『使用转换创建 EJB JAR 文件』中包含的指示信息。

导出商店有用资源

要导出商店有用资源，请执行以下操作:

1. 切换到Business Professional “J2EE 导航器”视图Express “项目导航器”视图。
2. 展开 **Stores** 文件夹。
3. 用鼠标右键单击 **Web Content** 文件夹并选择**导出**.
“导出”向导打开。
4. 在“导出”向导中，请执行以下操作:
   a. 选择**文件系统**并单击**下一步**。
   b. 单击**全部不选**。
   c. 选择以导出以下资源:
      - Web Content*FashionFlow_name*\MyNewJSPTemplate.jsp
      - Web Content*FashionFlow_name*\images\male_blueshirt.gif
      - Web Content\WEB-INF\classes*FashionFlow_name*\Tutorial_NLS_en_US.properties
      - Web Content\WEB-INF\lib\jstl.jar
      - Web Content\WEB-INF\lib\standard.jar

d. 选择**为文件创建目录结构**。

e. 在“目录”字段中输入放置这些资源的临时目录。例如，输入 `C:\ExportTemp`

f. 单击**完成**。

封装访问控制策略

在本部分，您将一个为新资源创建的访问控制策略复制到 *drive*:\ExportTemp 目录，如下:

1. 浏览到 *WCDE_installdir*\Commerce\xml\policies\xml 目录。
2. 将以下文件复制到 *drive*:\ExportTemp\ACPolicies 目录中:
   - MyNewViewACPolicy.xml
   - MyNewControllerCmdACPolicy.xml
   - SampleACPolicy_template.xml
   - SampleACPolicy_template_en_US.xml

将有用资源传送到目标 WebSphere Commerce Server

在本步骤中，您将在目标 WebSphere Commerce Server 中创建一个临时目录并把奖金点数有用资源复制到该目录。在后面的步骤中，您将把不同类型的代码置于 WebSphere Commerce 应用程序中的合适位置。

要把文件从开发机复制到目标 WebSphere Commerce Server，请执行以下操作:

1. 在目标 WebSphere Commerce Server 上，创建一个称为 *drive*:ImportTemp 的临时目录。
2. 确定如何从一台计算机向另一台计算机复制文件。您可以通过将目标 WebSphere Commerce Server 上的驱动器映射到开发机，或是使用 FTP 应用程序（如果已配置）来实现这一目的。
3. 从开发机将 *drive*:\ExportTemp 的内容复制到目标 WebSphere Commerce Server 上的 *drive*:\ImportTemp。

停止目标 WebSphere Commerce Server

在开始部署步骤之前，应停止目标 WebSphere Commerce Server。有关停止 WebSphere Commerce Server 的详细信息，请参考 Business Professional *《WebSphere Commerce Studio 安装指南》* 或 Express *《WebSphere Commerce - Express Developer Edition 安装指南》*。

更新目标 WebSphere Commerce Server 上的数据库

在更新目标数据库之前，请为您正在部署定制逻辑的商店验证商店实体标识。以下 SQL 语句可以用来确定该值:

```
select STOREENT_ID from STOREENT where IDENTITY='FashionFlow_name'
```

其中 *FashionFlow_name* 是您正在部署其代码的商店名称。

注册视图

DB2 如果正在使用 DB2 数据库，请执行以下操作注册 MyNewView:

1. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 命令行工具 > 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. 关闭工具设置。
5. 在选中“脚本”选项卡的情况下，通过在脚本窗口中输入以下信息，在 VIEWREG 表中创建必需的条目:

   ```
   connect to targetDB user dbuser using dbpassword;
   insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
      CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE)
   values ('MyNewView',-1, FF_storeent_ID,
      'com.ibm.commerce.command.ForwardViewCommand',
      'com.ibm.commerce.command.HttpForwardViewCommandImpl',
      'docname=MyNewJSPTemplate.jsp','This is my new view for tutorial 1',
      0, null);
   ```

   其中

   - *targetDB* 是目标数据库的名称
   - *dbuser* 是数据库用户
   - *dbpassword* 是数据库用户的密码
   - *FF_storeent_ID* 是基于“时尚潮流”样本商店的商店唯一标识

   单击**执行**图标。保持“命令中心”打开。

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:

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:

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
   CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE)
values ('MyNewView',-1, FF_storeent_ID,
   'com.ibm.commerce.command.ForwardViewCommand',
   'com.ibm.commerce.command.HttpForwardViewCommandImpl',
   'docname=MyNewJSPTemplate.jsp','This is my new view for tutorial 1',
   0, null);
```

其中

- *FF_storeent_ID* 是基于“时尚潮流”样本商店的商店唯一标识。

按 Enter 键运行 SQL 语句。

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。

MyNewView 已注册。

注册新控制器命令

要注册 MyNewControllerCmd，请执行以下操作:

1. DB2 如果您使用的是 DB2 数据库，请执行以下操作注册 MyNewControllerCmd:
   a. 在“命令中心”先选择“脚本”选项卡，然后通过在脚本窗口中输入如下信息，在 URLREG 表中创建必需的条目:

```
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS, DESCRIPTION,
   AUTHENTICATED) values ('MyNewControllerCmd',FF_storeent_ID,
   'com.ibm.commerce.sample.commands.MyNewControllerCmd',
   0, 'This is a new controller command for tutorial one.',null);
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
   CLASSNAME, TARGET)
values (FF_storeent_ID,
   'com.ibm.commerce.sample.commands.MyNewControllerCmd',
   'This is a new controller command for tutorial one.',
   'com.ibm.commerce.sample.commands.MyNewControllerCmdImpl',
   'local');
```

其中

- *targetDB* 是目标数据库的名称
- *dbuser* 是数据库用户
- *dbpassword* 是数据库用户的密码
- *FF_storeent_ID* 是基于“时尚潮流”样本商店的商店唯一标识。

单击**执行**图标。保持“命令中心”打开。

2. Oracle 如果正在使用 Oracle 数据库，请执行以下操作注册 MyNewControllerCmd:

   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 SQL 语句:

```
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS, DESCRIPTION,
   AUTHENTICATED) values ('MyNewControllerCmd',FF_storeent_ID,
   'com.ibm.commerce.sample.commands.MyNewControllerCmd',
   0, 'This is a new controller command for tutorial one.',null);
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
   CLASSNAME, TARGET)
values (FF_storeent_ID,
   'com.ibm.commerce.sample.commands.MyNewControllerCmd',
   'This is a new controller command for tutorial one.',
   'com.ibm.commerce.sample.commands.MyNewControllerCmdImpl','local');
```

   其中

   - *FF_storeent_ID* 是基于“时尚潮流”样本商店的商店唯一标识。

   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。

创建 XBONUS 表

在此步骤中，您将在数据库中创建目标 WebSphere Commerce Server 使用的 XBONUS 表。

DB2 如果正在使用 DB2 数据库，请执行以下操作创建表:

1. 在“脚本”窗口中，输入以下内容:

```
create table XBONUS (MEMBERID BIGINT NOT NULL,
    BONUSPOINT INTEGER NOT NULL, constraint p_xbonus
           primary key (MEMBERID),
    constraint f_xbonus foreign key (MEMBERID)
    references users (users_id) on delete cascade)
```

   其中

   - *targetDB* 是目标数据库的名称

- *dbuser* 是数据库用户
- *dbpassword* 是数据库用户的密码

单击“执行”图标。
XBONUS 表已创建。

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 XBONUS (MEMBERID NUMBER NOT NULL,
    BONUSPOINT INTEGER NOT NULL, constraint p_xbonus
           primary key (MEMBERID),
    constraint f_xbonus foreign key (MEMBERID)
 references users (users_id) on delete cascade);
```

   并按 Enter 键运行 SQL 语句。XBONUS 表已创建。
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。

在目标 WebSphere Commerce Server 上装入访问控制策略

在本步骤中，您将把新资源的访问控制策略装入目标 WebSphere Commerce Server。

要装入新策略，请执行以下操作:

1. 通过执行以下操作，更新访问控制策略以反映特定于目标 WebSphere Commerce Server 的值:
   a. 要确定“时尚潮流”商店的成员标识值，请执行以下操作:
      - DB2 如果您正在使用 DB2 数据库，请执行以下操作:
        1) 连接开发数据库。
        2) 发出以下 SQL 语句:

```
select member_id from storeent where storeent_id=FF_storeent_ID
```

其中 *FF_storeent_ID* 是您的“时尚潮流”商店的商店实体标识。例如，您可以输入:

```
select member_id from storeent where storeent_id=10001
```

记下成员标识值: _____

- Oracle 如果您正在使用 Oracle 数据库，请执行以下操作:
  1) 打开 Oracle SQL Plus 命令窗口并连接到您的开发数据库。
  2) 输入以下内容:
     `column member_id format 999999999999999999999999999999999999`
     (thiry-five 9's)
     这样将设置足够宽度的 member_id 列以看到完整值的显示。
  3) 发出以下 SQL 语句:

     ```
     select member_id from storeent where storeent_id=FF_storeent_ID
     ```

     其中 *FF_storeent_ID* 是您的“时尚潮流”商店的商店实体标识。例如，您可以输入:

     ```
     select member_id from storeent where storeent_id=10001
     ```

     记下成员标识值: _____
  4) 通过输入以下内容将 member_id 列的显示宽度复位:

     ```
     column member_id clear
     ```

b. 浏览到以下目录:
   drive:\ImportTemp\ACPolicies

c. 使用文本编辑器，打开 SampleACPolicy_template.xml 文件并把 *FashionFlowMemberId* 替换为步骤 1a 中所确定的“时尚潮流”商店的成员标识值。把修改后的文件保存为 SampleACPolicy.xml。

d. 使用文本编辑器，打开 SampleACPolicy_template_en_US.xml 文件并把 *FashionFlowMemberId* 替换为步骤 1a 中所确定的“时尚潮流”商店的成员标识值。把修改后的文件保存为 SampleACPolicy_en_US.xml。

2. 把以下访问控制策略复制到 *WC_installdir*\xml\policies\xml 目录:
   - MyNewViewACPolicy.xml
   - MyNewControllerCmdACPolicy.xml
   - SampleACPolicy.xml
   - SampleACPolicy_en_US.xml
3. 在命令提示符下，浏览至以下目录:
   WC_installdir\bin

4. 您必须发出 acpload 命令，此命令具有以下格式:

   ```
   acpload targetDB dbuser dbpassword inputXMLFile
   ```

   其中

   - *targetDB* 是开发数据库的名称。
   - *dbuser* 是数据库用户的名称。
   - *dbpassword* 是数据库用户的密码。
   - *inputXMLFile* 是包含访问控制策略规范的 XML 文件。在此例中，指定 `MyNewViewACPolicy.xml`。

   以下是带有指定变量的命令的示例:

   ```
   acpload Demo_Dev db2admin db2admin MyNewViewACPolicy.xml
   ```

5. 对以下每个访问控制策略重复步骤 4:
   - MyNewControllerCmdACPolicy.xml
   - SampleACPolicy.xml
6. 要装入策略描述（包含在 SampleACPolicy_en_US.xml 中），必须发出 acpnlsload 命令，它具有以下格式:

   ```
   acpnlsload db_name db_user db_password inputXMLFile
   ```

   例如，可以发出以下命令:

   ```
   acpnlsload Demo_dev user password SampleACPolicy_en_US.xml
   ```

7. 检查 *WC_installdir*\xml\policies\xml 目录，获取在装入访问控制策略时可能生成的任何错误日志。

更新目标 WebSphere Commerce Server 上的商店有用资源

在本步骤中，您将用已修改的商店有用资源来更新商店，如下:

1. 备份 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear\Stores.war 目录（其中 *cellName* 通常是机器的主机名，*instanceName* 是 WebSphere Commerce 实例的名称）。
2. 浏览到 *drive*:\ImportTemp\Stores\Web Content 目录。
3. 将 *FashionFlow_name* 和 WEB-INF 文件夹复制到以下目录:
   WAS_installdir\installedApps*cellName*\WC_*instanceName*.ear\Stores.war

更新目标 WebSphere Commerce Server 上的命令和数据 bean JAR 文件

在本步骤中您将更新目标 WebSphere Commerce Server 来使用新目标和数据 bean JAR 文件，如下:

1. 您应该制作现有 JAR 文件的备份副本，如下:

a. 浏览至 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear 目录。
b. 制作 WebSphereCommerceServerExtensionsLogic.jar 文件的副本并将它保存在备份位置。

2. 把新的 WebSphereCommerceServerExtensionsLogic.jar 文件从 *drive*:\ImportTemp 目录复制到 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear 目录。

其中 *instanceName* 是 WebSphere Commerce 实例的名称。

更新目标 WebSphere Commerce Server 上的 EJB JAR 文件

在本步骤中您将更新目标 WebSphere Commerce Server 来使用新 EJB JAR 文件，如下:

1. 您应该制作现有 JAR 文件的备份副本，如下:
   a. 浏览至 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear 目录。
   b. 制作 WebSphereCommerceServerExtensionsData.jar 文件的副本并将它保存在备份位置。
2. 把新的 WebSphereCommerceServerExtensionsData.jar 文件从 *drive*:\ImportTemp 目录复制到 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear 目录
3. 下一步，您必须修改 EJB 部署描述符信息，如下:
   a. 定位此 WebSphere Application Server 单元的部署资源库（META-INF 目录）。这通常采取以下格式:
   WAS_installdir\config\cells*cellName*\applications\WC_*instance_name*.ear\deployments\WC_*instance_name**EJBModuleName*.jar\META-INF。
   以下是它的一个特定示例:
   D:\WebSphere\AppServer\config\cells\myCell\applications\WC_demo.ear\deployments\WC_demo\WebSphereCommerceServerExtensionsData.jar\META-INF
   b. 该目录包含以下文件:
      - ejb-jar.xml
      - ibm-ejb-access-bean.xmi
      - ibm-ejb-jar-bnd.xmi
      - ibm-ejb-jar-ext.xmi
      - MANIFEST.MF

      备份所有这些文件。
   c. 使用工具打开新 WebSphereCommerceServerExtensionsData.jar 文件并查看其内容。

d. 将 meta-inf 目录的内容（上面列出的文件）从该 WebSphereCommerceServerExtensionsData.jar 文件抽取到步骤 3a 的目录中。确保目录结构在您抽取这些文件后保持正确。

4. 在命令行中使用 WebSphere Application Server startServer 命令，重新启动 WebSphere Commerce 实例。关于启动和停止此实例的更多信息，请参阅对应于您的平台和数据库的*《WebSphere Commerce 安装指南》*。

验证目标 WebSphere Commerce Server 上的奖金点数逻辑

在此步骤中，通过执行以下操作，验证已将奖金点数逻辑成功部署到目标 WebSphere Commerce Server:

1. 打开 Web 浏览器并输入 URL，以启动您的基于“时尚潮流”样本商店的商店。
2. 通过执行以下操作，创建新的注册用户:
   a. 单击**注册**。
   b. 再次单击**注册**来创建新客户。
   c. 在注册表单中，在所有必填字段中输入相应的值。例如，在电子邮件字段中，输入 `tester@mycompany`。记下电子邮件地址的值:

      _____。

   d. 一旦输入了值，请单击**提交**。
3. 完成登录后，在同一个浏览器中输入以下 URL:

   ```
   http://hostname/webapp/wcs/stores/servlet/MyNewControllerCmd?
      input1=user_e-mail&input2=1000
   ```

   其中 *user_e-mail* 是您在步骤 2 中创建的用户的电子邮件地址。将向您显示一个页面，其中包含了先前所有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）。

图 43.

4. 现在，在**登录标识**字段中输入用户的电子邮件地址，并在**奖金点数**字段中输入 500。单击**提交**。会显示与以下类似的页面，该页面显示已更新了奖金点数余额。

图 44.

第 11 章 教程: 修改现有的控制器命令

本教程的目的是演示用于修改现有控制器命令的过程。

在这一教程中，您需要把客户购物车中的商品数限制为 5 个或更少。要实现这一解决方案，需要用您自己的实现重设 OrderItemAddCmdImpl，这一实现包括对购物车中的商品数进行检查的逻辑。如果一个客户试图向购物车添加第 6 项商品，就会发生异常。该异常使用新的错误消息。

请注意: 本教程的目的是演示用于修改现有命令逻辑的开发过程。这不意味着将包含所有在购物车中的限制商品的示例。本教程中所用的逻辑是为了教程而简化的。

在本教程中，您将学习以下内容:

- 如何为现有的控制器命令创建新的实现
- 如何更新命令注册表从而可以在应用程序中使用新的实现
- 如何向现有的 WebSphere Commerce 应用程序部署已修改的控制器命令

先决条件

本教程并不要求您完成第 201 页的第 10 章，『教程: 创建新的业务逻辑』。如果您已完成该教程，那么把代码留在工作区没有坏处，因为它不会与本教程冲突。

在开始本教程前，必须已经发布了一个基于“时尚潮流”样本商店的商店。在这个商店里，您必须能够完成一项购买（例如，浏览产品目录，向购物车添加商品，检出并查看订单确认）。

为了完成部署步骤，商店必须同时存在于目标 WebSphere Commerce Server 上。

创建新 MyOrderItemAddCmdImpl 类

在教程本步骤中，您需要创建一个新 MyOrderItemAddCmdImpl 类。要创建这个类，请执行以下操作:

1. 在 WebSphere Studio Application Developer 中，打开 Java 视图（**视窗 > 打开视图 > Java**）。
2. 浏览到 **WebSphereCommerceServerExtensionsLogic** 项目。
3. 浏览到 **src** 目录。

4. 如果您没有完成 第 201 页的第 10 章，『教程: 创建新的业务逻辑』，用鼠标右键单击 **src** 目录并选择**新建 > 数据包**。
   “新 Java 数据包”向导打开。在本向导中，请执行以下操作:
   a. 在**名称**字段，输入 com.ibm.commerce.sample.commands。
   b. 单击**完成**。
5. 用鼠标右键单击 **com.ibm.commerce.sample.commands** 数据包。选择**新建 > 类**。
   “新 Java 类”向导打开。
6. 在“新 Java 类”向导中，执行以下操作:
   a. 在**名称**字段，输入 MyOrderItemAddCmdImpl。
   b. 要指定超类，请单击“超类”字段旁边的**浏览**按钮，然后输入 OrderItemAddCmdImpl。单击**确定**。
   c. 要指定要实现哪一接口，请单击**添加**，然后输入 OrderItemAddCmd 并单击**确定**。
   d. 单击**完成**。

将显示 MyOrderItemAddCmdImpl 类的源代码。

下一步是更新该类中的导入语句，如下:

1. 在“概览”视图中，选择并打开**导入声明**。您会发现有两个导入语句已创建。
2. 向导入语句的源代码中添加以下导入语句:

```
import com.ibm.commerce.exception.ECApplicationException;
import com.ibm.commerce.exception.ECException;
import com.ibm.commerce.exception.ECSystemException;
import com.ibm.commerce.order.objects.OrderAccessBean;
import com.ibm.commerce.ras.ECMessage;
import com.ibm.commerce.sample.messages.MyNewMessages;
```

3. 保存更改。

下一步是添加业务逻辑和异常处理，从而在添加更多订购商品之前确定在客户购物车中是不是已经有多于 5 项商品。如下更新代码:

1. 在“概览”视图中，选择 MyOrderItemAddCmdImpl 类并查看其源代码。当前其中只有如下代码:

```
public class MyOrderItemAddCmdImpl
 extends OrderItemAddCmdImpl
 implements OrderItemAddCmd {

}
```

2. 必须向此类添加一个新的 performExecute 方法。这一方法包括用于检查购物车中商品数的逻辑，如果数量少于 5，则通常会调用超类

（OrderItemAddCmdImpl）的常规 performExecute 方法。如果有 5 项或更多商品，就会发生异常而且用户将无法向购物车添加更多商品。要添加此方法，把以下源代码复制到该类（确保它已包含在表示类结束的最后一个右括号 “}” 之前）：

```
public void performExecute() throws ECException {
  // Get a list of order ids
  String[] orderIds = getOrderId();

  // Check to make sure that an id exists at all
  // if order id exists then get number of items in the order
  // else if no order id exists then execute normal code
  if (orderIds != null && orderIds.length > 0) {
   // An exception should be thrown when trying to add a sixth item
   // to the cart.  Since this code is run before any items are added
   // throw and exception if there are 5 or more items in the cart
   if (itemsInOrder(orderIds[0]) >= 5) {
          throw new ECApplicationException(
     MyNewMessages._ERR_TOO_MANY_ITEMS,
    this.getClass().getName(),
     "performExecute");
   }
   // else perform normal flow
  }
  super.performExecute();
 }
 // get number of items in the order
 protected int itemsInOrder(String orderId) throws ECException {
  try {
   OrderAccessBean order = new OrderAccessBean();
   order.setInitKey_orderId(orderId);
   order.refreshCopyHelper();
   return order.getOrderItems().length;
     } catch (javax.ejb.FinderException e) {
throw new ECSystemException(
    ECMessage._ERR_FINDER_EXCEPTION,
    this.getClass().getName(),
    "itemsInOrder");
     } catch (javax.naming.NamingException e) {
throw new ECSystemException(
    ECMessage._ERR_NAMING_EXCEPTION,
    this.getClass().getName(),
    "itemsInOrder");
} catch (java.rmi.RemoteException e) {
throw new ECSystemException(
    ECMessage._ERR_REMOTE_EXCEPTION,
    this.getClass().getName(),
    "itemsInOrder");
} catch (javax.ejb.CreateException e) {
throw new ECSystemException(
    ECMessage._ERR_CREATE_EXCEPTION,
```

```
        this.getClass().getName(),
        "itemsInOrder");
    }
  }
```

在您将新代码粘贴到类中之后，在源代码中用鼠标右键单击并选择**格式化**以格式化代码。

3. 保存工作。

   注: 将出现警告表示消息信息丢失。这将在以后的步骤中更正。

创建消息信息

新的命令实现使用新的错误消息: _ERR_TOO_MANY_ITEMS。在本部分内，您为新的消息及其相关属性文件创建代码。要导入此代码，请执行以下操作:

1. 展开 **WebSphereCommerceServerExtensionsLogic** 项目。
2. 用鼠标右键单击 **src** 目录并选择**新建 > 数据包**。
   “新 Java 数据包”向导打开。在本向导中，请执行以下操作:
   a. 在**名称**字段，输入 com.ibm.commerce.sample.messages。
   b. 单击**完成**。
3. 用鼠标右键单击 **com.ibm.commerce.sample.messages** 数据包。选择**新建 > 类**。
   “新 Java 类”向导打开。
4. 在“新 Java 类”向导中，执行以下操作:
   a. 在**名称**字段中，输入 MyNewMessages。
   b. 单击**完成**。
5. 双击 **MyNewMessages** 类以查看其源代码。
6. 立即在 public class MyNewMessages 代码行前添加以下导入语句:

```
import com.ibm.commerce.ras.ECMessage;
import com.ibm.commerce.ras.ECMessageSeverity;
import com.ibm.commerce.ras.ECMessageType;
```

7. 在该类中，添加以下代码:

```
// Resouce bundle used to extract the text for an exception
 static final String errorBundle = "MyNewErrorMessages";

 // An ECMessage is used to describe an ECException and is passed
 // into the ECException when thrown
 public static final ECMessage _ERR_TOO_MANY_ITEMS =
      new ECMessage(ECMessageSeverity.ERROR, ECMessageType.USER,
      MyNewMessageKeys._ERR_TOO_MANY_ITEMS, errorBundle);
```

8. 保存更改。
9. 用鼠标右键单击 **com.ibm.commerce.sample.messages** 数据包。选择**新建 > 类**。
   “新 Java 类”向导打开。
10. 在“新 Java 类”向导中，执行以下操作:
    a. 在**名称**字段中，输入 MyNewMessageKeys。
    b. 单击**完成**。
11. 按以下内容定义类中的代码:

```
public class MyNewMessageKeys {
// This class defines the keys used to create new exceptions that are
// thrown by customized code.
 public static final String _ERR_TOO_MANY_ITEMS = "_ERR_TOO_MANY_ITEMS";
}
```

12. 保存更改。
13. 通过用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 项目并选择**构建项目**来编译您的代码。
14. 创建新的将包含消息信息的属性文件，如下所示:
    a. 打开 Web 视图（**视窗 > 打开视图 > Web**）。
    b. 在 **Stores** Web 项目中，展开 **Web Content > WEB-INF > classes** 文件夹。
    c. 用鼠标右键单击 **classes** 文件夹并选择 **新建 > 其它 > 简单 > 文件 > 下一个**来创建一个新的属性文件。
       “新建文件”窗口打开。
    d. 在**文件名**字段，输入 MyNewErrorMessages.properties，然后单击**完成**。
       新的空文件打开。
    e. 将以下文本复制到新文件中:

```
_ERR_TOO_MANY_ITEMS=You are trying to place too many different items
   in one shopping cart.
```

       注: 在上述代码中的断行仅出于显示目的。在一行上输入您的文本。
    f. 保存更改。

修改命令注册表

在本步骤中，您需要修改命令注册表，这样可使用新的 MyOrderItemAddCmdImpl 实现类而不是原始 OrderItemAddCmdImpl 实现类。命令注册表中唯一需要修改的表是 CMDREG 表。在本例中，新实现类用于所有商店。

要修改命令注册表，请执行以下操作:

1. DB2 如果正在使用 DB2 数据库，请执行以下操作注册 MyOrderItemAddCmdImpl:
   a. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 控制中心**）。
   b. 从**工具**菜单中，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 developmentDB user dbuser using dbpassword;
update CMDREG
set CLASSNAME='com.ibm.commerce.sample.commands.MyOrderItemAddCmdImpl'
WHERE INTERFACENAME=
'com.ibm.commerce.orderitems.commands.OrderItemAddCmd'
and storeent_Id=0;
```

   其中

   - *developmentDB* 是开发数据库的名称
   - *dbuser* 是数据库用户
   - *dbpassword* 是数据库用户的密码

   单击**执行**图标。

2. Oracle 如果正在使用 Oracle 数据库，请执行以下操作注册 MyOrderItemAddCmdImpl:
   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 SQL 语句:

```
update CMDREG
set CLASSNAME='com.ibm.commerce.sample.commands.MyOrderItemAddCmdImpl'
WHERE INTERFACENAME=
'com.ibm.commerce.orderitems.commands.OrderItemAddCmd'
and storeent_Id=0;
```

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。

测试 MyOrderItemAddCmdImpl 命令

下一步是测试以确保新逻辑正常工作。要测试该命令，您应该可以向购物车成功添加 5 个商品，但是预期会在试图向购物车添加第 6 个商品时发生错误。

要测试新业务逻辑，请执行以下操作:

1. 切换至“服务器”视图（**视窗 > 打开视图 > 服务器**）。
2. 用鼠标右键单击 **WebSphereCommerceServer** 服务器并选择**启动**（或**重新启动**）。
3. 用鼠标右键单击 Stores\Web Content*FashionFlow_name* 目录下的 **index.jsp** 并选择**在服务器上运行**。
   商店主页显示在 Web 浏览器上。
4. 在商店内购物并向购物车添加 5 个商品。添加第 5 个商品后，您会获得与下面相似的购物车:

图 45.

5. 单击**继续购物**并选择其它商品。对此选定商品，单击**添加至购物车**。您会看到如下错误页:

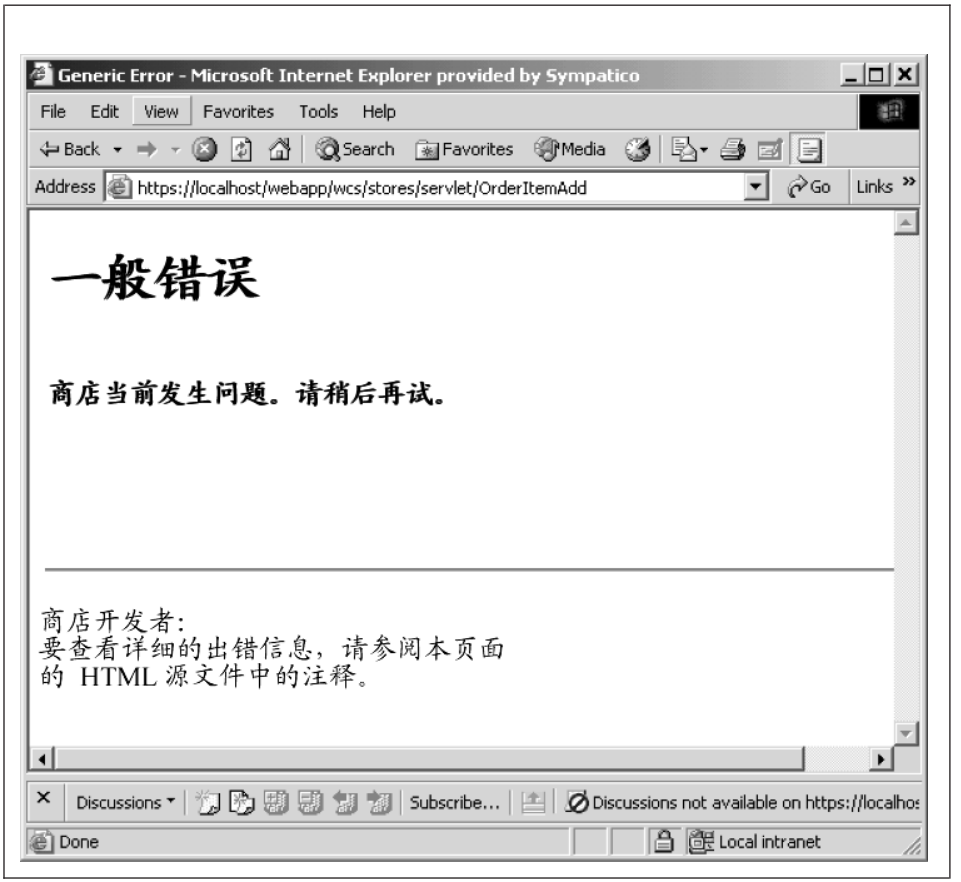

图 46.

查看错误页面源代码。在源代码中，向下滚动以找到以下错误信息:

```
Message Key: _ERR_TOO_MANY_ITEMS
Message: You are trying to place too many different items in one shopping
         cart
```

部署 MyOrderItemAddCmdImpl

在此步骤中，您将向目标 WebSphere Commerce Server 部署已修改的业务逻辑。在此例中，部署由以下高级步骤构成:

1. 创建包含命令逻辑和错误类的 JAR 文件。
2. 导出错误消息属性文件。
3. 把该资源传送到目标 WebSphere Commerce Server。
4. 更新目标 WebSphere Commerce Server 上的命令注册表。
5. 验证目标 WebSphere Commerce Server 上的新逻辑。

创建命令 JAR 文件

遵照 WebSphere Commerce 代码定制策略，将定制命令和数据 bean 置于 WebSphereCommerceServerExtensionsLogic 项目中。因此，当要部署定制代码时，您需注意要把先前定制的代码包含到 JAR 文件中。例如，如果已完成 第 201 页的第 10 章，『教程: 创建新的业务逻辑』，您需注意在创建 JAR 文件以部署 MyOrderItemAddCmdImpl 类时，要包括先前创建的 MyNewControllerCmd 和其它类。

要创建包含 MyOrderItemAddCmdImpl 类的 JAR 文件，请在开发机上执行以下步骤:

1. 在本地文件系统中创建一个称为 *drive*:\ExportTemp2 的目录。
2. 打开 WebSphere Studio Application Developer 并切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
3. 用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 项目并选择**导出**。
   “导出”向导打开。
4. 在“导出”向导中，请执行以下操作:
   a. 选择 **JAR 文件**并单击**下一步**.
   b. **选择要导出的资源**左下窗格是以项目的名称预先填充的。保留此值不变。
   c. 在右窗格中，确保仅选择了以下资源:
      - .classpath
      - .project
      - .serverPreference
   d. 确保已选择**导出已生成的类文件和资源**。
   e. 不要选择**导出 Java 源文件和资源**。
   f. 在**选择导出目的地**字段中，输入要使用的全限定 JAR 文件名。在此例中，输入 *drive*:\ExportTemp2\WebSphereCommerceServerExtensionsLogic.jar。请注意，JAR 文件名必须是 WebSphereCommerceServerExtensionsLogic.jar。
   g. 单击**完成**。

导出消息属性文件

在本部分中，您导出包含新消息文本的属性文件，如下:

1. 切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
2. 展开 **Stores** 文件夹。

3. 用鼠标右键单击 **Web Content** 文件夹并选择**导出**。
   “导出”向导打开。
4. 在“导出”向导中，请执行以下操作:
   a. 选择**文件系统**并单击**下一步**。
   b. 单击**全部不选**。
   c. 选择以导出以下资源:
      Web Content\WEB-INF\classes\MyNewErrorMessages.properties
   d. 选择**为文件创建目录结构**。
   e. 在“目录”字段中输入放置这些资源的临时目录。例如，输入 `C:\ExportTemp2`
   f. 单击**完成**。

将有用资源传送到目标 WebSphere Commerce Server

在本步骤中，您将在目标 WebSphere Commerce Server 中创建一个临时目录并把 MyOrderItemAddCmdImpl 有用资源复制到该目录。

要把文件从开发机复制到目标 WebSphere Commerce Server，请执行以下操作:

1. 在目标 WebSphere Commerce Server 上，创建一个称为 *drive*:ImportTemp2 的临时目录。
2. 确定如何从一台计算机向另一台计算机复制文件。您可以通过将目标 WebSphere Commerce Server 上的驱动器映射到开发机或是使用 FTP 应用程序（如果已配置）来实现这一目的。
3. 从开发机将 *drive*:\ExportTemp2 的内容复制到目标 WebSphere Commerce Server 上的 *drive*:ImportTemp2。

停止目标 WebSphere Commerce Server

在开始部署步骤之前，应停止目标 WebSphere Commerce Server。有关停止目标 WebSphere Commerce Server 的详细信息，请参考 Business Professional *《WebSphere Commerce Studio 安装指南》*或 Express *《WebSphere Commerce - Express Developer Edition 安装指南》*。

更新目标 WebSphere Commerce Server 上的数据库

在本步骤中，您需要修改命令注册表以便它使用新 MyOrderItemAddCmdImpl 实现类。

DB2 如果正在使用 DB2 数据库，请执行以下操作注册 MyOrderItemAddCmdImpl:

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. 关闭工具设置。
5. 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 targetDB user dbuser using dbpassword;
   update CMDREG
   set CLASSNAME='com.ibm.commerce.sample.commands.MyOrderItemAddCmdImpl'
   WHERE INTERFACENAME=
   'com.ibm.commerce.orderitems.commands.OrderItemAddCmd'
   and storeent_Id=0;
   ```

   其中

   - *targetDB* 是目标 WebSphere Commerce Server 使用的数据库的名称
   - *dbuser* 是数据库用户
   - *dbpassword* 是数据库密码

   单击**执行**图标。

Oracle 如果正在使用 Oracle 数据库，请执行以下操作注册 MyOrderItemAddCmdImpl:

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:

   ```
   update CMDREG
   set CLASSNAME='com.ibm.commerce.sample.commands.MyOrderItemAddCmdImpl'
   WHERE INTERFACENAME=
   'com.ibm.commerce.orderitems.commands.OrderItemAddCmd'
   and storeent_Id=0;
   ```

   按 Enter 键运行 SQL 语句。
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。

更新目标 WebSphere Commerce Server 上的命令 JAR 文件

在本步骤中您将更新目标 WebSphere Commerce Server 来使用包含新 MyOrderItemAddCmdImpl 的 JAR 文件，如下:

1. 在命令行中使用 WebSphere Application Server stopServer 命令，停止 WebSphere Commerce 实例。关于启动和停止此实例的信息（如需要的话），请参阅对应于您的平台和数据库的*《WebSphere Commerce 安装指南》*。
2. 您应制作现有 JAR 文件的备份副本，如下:
   a. 浏览至 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear 目录，其中 *cellName* 通常是机器的主机名
   b. 制作 WebSphereCommerceServerExtensionsLogic.jar 文件的副本并将它保存在备份位置。
3. 把新的 WebSphereCommerceServerExtensionsLogic.jar 文件从 *drive*:\ImportTemp2 目录复制到 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear 目录。

其中 *instanceName* 是 WebSphere Commerce 实例的名称。

更新目标 WebSphere Commerce Server 上的消息属性

在此步骤中，您将添加新的消息属性文件到应用程序，如下:

1. 备份 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear\Stores.war 目录（其中 *cellName* 通常是机器的主机名，*instanceName* 是 WebSphere Commerce 实例的名称）。
2. 浏览到 *drive*:\ImportTemp\Stores\Web Content 目录。
3. 复制 WEB-INF 文件夹到以下目录中:
   WAS_installdir\installedApps*cellName*\WC_*instanceName*.ear\Stores.war
4. 在命令行中使用 WebSphere Application Server startServer 命令，重新启动 WebSphere Commerce 实例。

其中 *instanceName* 是 WebSphere Commerce 实例的名称。

验证目标 WebSphere Commerce Server 上的 MyOrderItemAddCmdImpl 逻辑

在本步骤中，您将执行快速检查以验证代码一旦部署就可正常工作。

要验证代码，请执行以下操作:

1. 打开 Web 浏览器并启动 “时尚潮流” 商店。例如，输入以下 URL 以启动商店:

```
http://hostname/webapp/wcs/stores/servlet/FashionFlow/index.jsp
```

其中 *hostname* 是实例的主机名。

2. 在商店内购物并向购物车添加 5 个商品。添加第 5 个商品后，您会获得与下面相似的购物车:

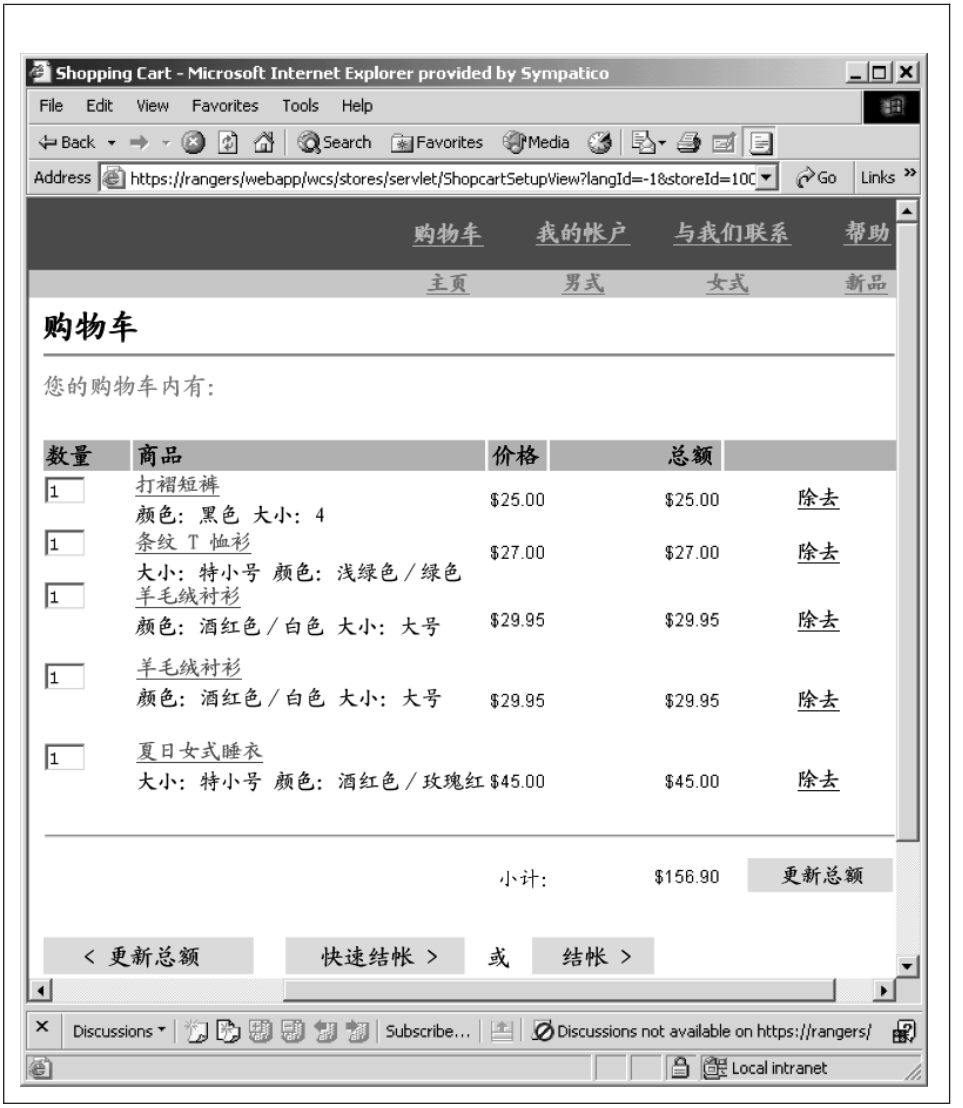

图 47.

3. 单击**继续购物**并选择其它商品。对此选定商品，单击**添加至购物车**。您会看到如下错误页:

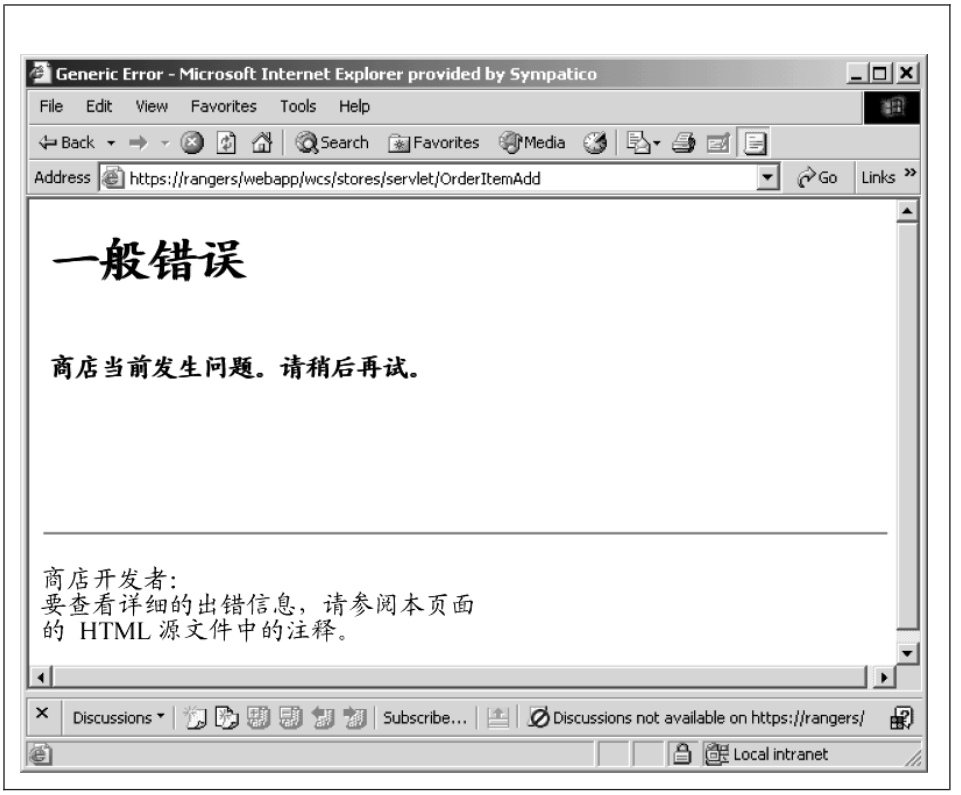

图 48.

查看错误页面源代码。请在源代码中向下滚动，您会发现错误的消息键是_ERR_TOO_MANY_ITEMS。

第 12 章 教程: 扩展对象模型并修改现有的任务命令

在这一教程中，您要满足为订单收集礼物卡信息的要求。必须收集的信息包括接收方的名称、发送方的名称和两个消息。客户提交订单后就要准备收集信息。

因为礼物卡的信息必须存储在数据库中，所以显然需要一个新的数据库表。由于这是订单处理的扩展，因此有两个可能的方法可用于修改对象模型。通常，或者可以修改现有的 WebSphere Commerce 公共实体 bean 并把已修改 bean 中的新字段映射到新表，或者可以创建一个直接映射到新表的新实体 bean。由于该方法需要扩展 Order 实体 bean 而且该 bean 使用 “找到即更新” 类型的 SQL 查询，因此您无法用此方法扩展该 bean。因此，在此情况下您必须创建映射到新表的新实体 bean。

除了新实体 bean，也会扩展现有的 ExtOrderProcessCmdImpl 任务命令。扩展用于实例化与新表对应的新数据 bean，并用于更新数据库中的礼物信息。

本教程包括以下高级任务:

1. 创建并填充新 XORDGIFT 表
2. 创建新 OrderGift 实体 bean
3. 创建 XORDGIFT 模式
4. 创建表定义并将 OrderGift 实体 bean 中的字段映射为 XORDGIFT 表中的列
5. 生成 OrderGift bean 的部署代码和访问 bean
6. 创建新 OrderGiftDataBean
7. 创建新 MyExtOrderProcessCmdImpl 任务命令实现
8. 修改 OrderSubmitForm.jsp 以收集消息信息，并修改 OrderDetailDisplayForm.jsp 以显示消息信息。
9. 测试已修改代码

先决条件

本教程并不要求您已完成某些教程的内容。如果您已完成这些教程，那么把代码留在工作区没有坏处，因为它不会与本教程冲突。

在开始本教程前，必须已经发布了一个基于 “时尚潮流” 样本商店的商店。在这个商店里，您必须能够完成一项购买（例如，浏览产品目录，向购物车添加商品，检出并查看订单确认）。

创建并填充 XORDGIFT 表

在此步骤中，您将创建 XORDGIFT 表。

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 > 命令行工具 > 控制中心**）并单击**脚本**选项卡。
2. 在“脚本”窗口中，输入以下内容:

```
connect to developmentDB user dbuser using dbpassword;

create table XORDGIFT (ORDERSID bigint not null, RECEIPTNAME varchar(50),
SENDERNAME varchar(50), MSGFIELD1 varchar(50), MSGFIELD2 varchar(50),
constraint p_xordgift primary key (ORDERSID),
constraint f_xordgift foreign key (ORDERSID) references ORDERS(ORDERS_ID)
on delete cascade);
insert into XORDGIFT (ORDERSID) (select ORDERS_ID from ORDERS);
```

其中

- *developmentDB* 是开发数据库的名称
- *dbuser* 是数据库用户
- *dbpassword* 是数据库用户的密码

单击“执行”图标。
现在，XORDGIFT 表已创建。

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 XORDGIFT (ORDERSID NUMBER NOT NULL, RECEIPTNAME VARCHAR(50),
SENDERNAME VARCHAR(50), MSGFIELD1 VARCHAR(50), MSGFIELD2 VARCHAR(50),
constraint p_xordgift primary key (ORDERSID),
constraint f_xordgift foreign key (ORDERSID) references ORDERS(ORDERS_ID)
on delete cascade);
insert into XORDGIFT (ORDERSID) (select ORDERS_ID from ORDERS);
```

并按 Enter 键运行 SQL 语句。XORDGIFT 表已创建。

注: 如果其他人先前已使用此数据库运行本示例，则在创建 XORDGIFT 表之前必须发出以下命令:

```
drop table XORDGIFT;
```

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 OrderGift 实体 bean

一旦已经创建数据库表，就已准备好开始创建新的实体 bean。下一步使用 WebSphere Studio Application Developer 创建该 bean。

下一步通过执行以下操作创建新 OrderGift bean:

1. 启动 WebSphere Commerce 开发环境如下:
   - Business Professional **开始 > 程序 > IBM WebSphere Commerce Studio > WebSphere Commerce 开发环境**
   - Express Express **开始 > 程序 > IBM WebSphere - Express Developer Edition > WebSphere Commerce 开发环境**
2. 打开 “J2EE 视图”。
3. 在 “J2EE 层次结构” 视图中，展开 **EJB 模块**。
4. 用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 模块并选择 **新建 > 企业 bean**。
   “企业 bean 创建” 向导打开。
5. 从 **EJB 项目**下拉列表，选择 **WebSphereCommerceServerExtensionsData** 并单击**下一步**。
6. 请在 “创建企业 bean” 窗口执行以下操作:
   a. 选择**带有受容器管理的持久性（CMP）字段的实体 bean**
   b. 在 **bean 名称**字段中，输入 `OrderGift`。
   c. 在**源文件夹**字段中，保留指定的缺省值（ejbModule）。
   d. 在**缺省数据包**字段中，输入 `com.ibm.commerce.extension.objects`。
   e. 单击**下一步**。
7. 在 “CMP 属性” 窗口，请执行以下操作:
   a. 单击**添加**以向表中的以下列添加新字段:
      - ORDERSID

- RECEIPTNAME
- SENDERNAME
- MSGFIELD1
- MSGFIELD2

“创建 CMP 属性”窗口打开。在此窗口中，请执行以下操作:

1) 如下创建 ordersId 字段:

表 12.

参数名称	参数值
名称	ordersId
类型	java.lang.Long **注**: 您必须使用 *java.lang.Long* 数据类型，而不是 *long* 数据类型。
关键字段	选择

单击**应用**。

2) 如下创建 receiptName 字段:

表 13.

参数名称	参数值
名称	receiptName
类型	java.lang.String
使用获取函数和设置函数方法进行访问	选择
将获取函数和设置函数方法提升到远程接口	清除

单击**应用**。

3) 如下创建 senderName 字段:

表 14.

参数名称	参数值
名称	senderName
类型	java.lang.String
使用获取函数和设置函数方法进行访问	选择
将获取函数和设置函数方法提升到远程接口	清除

单击**应用**。

4) 如下创建 msgField1 字段:

表 15.

参数名称	参数值
名称	msgField1
类型	java.lang.String
使用获取函数和设置函数方法进行访问	选择
将获取函数和设置函数方法提升到远程接口	清除

单击**应用**。

5) 如下创建 msgField2 字段:

表 16.

参数名称	参数值
名称	msgField2
类型	java.lang.String
使用获取函数和设置函数方法进行访问	选择
将获取函数和设置函数方法提升到远程接口	清除

单击**应用**。

6) 单击**关闭**以关闭此窗口。

b. 清除**对键类使用单一键属性类型**复选框，然后单击**下一步**。

8. 在 EJB Java 类详细信息窗口，请执行以下操作:
   a. 要选择该 bean 的超类，请单击**浏览**。
   “类型选择”窗口打开。
   b. 在**使用以下任意方法选择一个类**字段中，输入 ECEntityBean 并单击**确定**。
   这选择了 com.ibm.commerce.base.objects.ECEntityBean 为超类。
   c. 单击**完成**。

通过执行以下操作设置新 bean 的隔离级别:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**。
2. 双击 **WebSphereCommerceServerExtensionsData** 项目以便用部署描述符编辑器打开它。
3. 单击**访问**选项卡。
4. 单击“隔离级别”文本框旁的**添加**。
   “添加隔离级别”窗口打开。

5. DB2 选择**可重复读**，然后单击**下一步**。
   Oracle 选择**已提交的读**，然后单击**下一步**
6. 选择 **OrderGift** bean，然后单击**下一步**。
7. 选择 **OrderGift** 来选中其所有方法，然后单击**完成**。
8. 保存工作（Ctrl+S）。

然后通过执行以下操作设置该 bean 的安全性身份:

1. 在部署描述符编辑器中，选择“访问”选项卡。
2. 单击“安全性身份”（方法级别）文本框旁的**添加**。
   “添加安全性身份”窗口打开。
3. 选择**使用 EJB 服务器的身份**，然后单击**下一步**。
4. 选择 **OrderGift** bean，然后单击**下一步**。
5. 选择 **OrderGift** 来选中其所有方法，然后单击**完成**。
6. 保存工作（Ctrl+S）并使编辑器保持打开。

然后通过执行以下操作为 bean 中的方法设置安全性角色:

1. 在部署描述符编辑器中，选择“Assembly 描述符”选项卡。
2. 在“方法许可权”部分，单击**添加**。
3. 选择 **WCSecurityRole** 作为安全角色并单击**下一步**。
4. 从找到的 bean 列表中，选择 **OrderGift** 然后单击 **下一步**。
5. 在“方法”元素页面，单击**应用到所有**，然后单击**完成**。
6. 保存工作（Ctrl+S）然后关闭部署描述符编辑器。

下一步是除去与 WebSphere Studio Application Developer 生成的实体上下文相关的字段和方法。需要删除这些字段的原因是 ECEntityBean 基类对于这些方法会提供其自己的实现。要删除已生成的实体上下文字段和方法，请执行以下操作:

1. 在“J2EE 层次结构”视图中，展开 **WebSphereCommerceServerExtensionsData** 项目。
2. 展开 **OrderGift** EJB 模块，然后双击 **OrderGiftBean**。
3. 在“概览”视图中，请执行以下操作:

   注: Express WebSphere Studio Application Developer 5.1 会提示您删除 getEntityContext() 和 setEntityContext(EntityContext)，这样您就不必像下面所提到的那样手工删除它们。确保选择删除这些项。

   a. 用鼠标右键单击 **myEntityCtx** 字段并选择**删除**。

b. 用鼠标右键单击 **getEntityContext()** 方法，并选择**删除**。

c. 用鼠标右键单击 **setEntityContext(EntityContext)** 方法，并选择**删除**。

d. 用鼠标右键单击 **unsetEntityContext()** 方法并选择**删除**。

4. 保存工作（Ctrl+S）。

应修改生成的 ebjCreate 方法，以便在创建新的 OrderGift bean 时所有参数都是显式设置，如下所示:

1. 在“J2EE 层次结构”视图中，双击 **OrderGiftBean** 类以打开它并查看其源代码。
2. 在“概览”视图中，选择 ejbCreate(Long) 方法。其源代码显示如下:

```
public com.ibm.commerce.extension.objects.OrderGiftKey
   ejbCreate(java.lang.Long ordersId)
   throws javax.ejb.CreateException {
 _initLinks();
      this.ordersId = ordersId;
      return null;
}
```

3. 修改代码使它显示如下:

```
public com.ibm.commerce.extension.objects.OrderGiftKey
   ejbCreate(java.lang.Long ordersId)
   throws javax.ejb.CreateException {
 _initLinks();
      this.ordersId = ordersId;
      this.senderName = null;
      this.receiptName = null;
      this.msgField1 = null;
      this.msgField2 = null;
      return null;
}
```

4. 保存代码更改。

接着，通过执行以下操作向 OrderGift bean 添加新的 ejbCreate 方法:

1. 在“J2EE 层次结构”视图中，双击 **OrderGiftBean** 类以打开它并查看其源代码。
2. 通过向类添加以下代码来创建新的 ejbCreate(Long, String, String, String, String) 方法:

```
public com.ibm.commerce.extension.objects.OrderGiftKey
   ejbCreate(java.lang.Long ordersId,
   java.lang.String receiptName,
   java.lang.String senderName,
   java.lang.String msgField1,
   java.lang.String msgField2)
   throws javax.ejb.CreateException {
 _initLinks();
      this.ordersId = ordersId;
```

```
        this.senderName = senderName;
        this.receiptName = receiptName;
        this.msgField1 = msgField1;
        this.msgField2 = msgField2;
        return null;
    }
```

3. 保存代码更改。
4. 必须将此新的 ejbCreate 方法添加到人机接口。这将使方法在生成的访问 bean 中可用。要向主接口添加方法，请执行以下操作:
   a. 在“概览”视图中，用鼠标右键单击 **ejbCreate(Long, String, String, String, String)** 方法并选择**企业 Bean > 提升至人机接口**。

接着，通过执行以下操作创建新 ejbPostCreate(Long, String, String, String, String) 方法，从而使它拥有与 ejbCreate(Long, String, String, String, String) 方法相同的参数:

1. 双击 **OrderGiftBean** 类以打开它并查看其源代码。
2. 通过向类添加以下代码来创建新的 ejbPostCreate(Long ordersId, String receiptName, String senderName, String msgField1, String msgField2) 方法:

```
public void ejbPostCreate(
   java.lang.Long ordersId,
   java.lang.String receiptName,
   java.lang.String senderName,
   java.lang.String msgField1,
   java.lang.String msgField2)
   throws javax.ejb.CreateException {
   }
```

3. 保存代码更改。

下一步您必须创建 XORDGIFT 表的定义。如果您*没有*完成 第 201 页的第 10 章，『教程: 创建新的业务逻辑』，执行以下操作来创建 XORDGIFT 表定义

1. 打开“数据”视图并切换至“数据定义”视图。
2. 浏览至以下目录:
   WebSphereCommerceServerExtensionsData > ejbModule > META-INF
3. 用鼠标右键单击 **META-INF** 并选择**新数据库定义**。
   “新数据库定义”向导打开。
4. 在**数据库名**字段，输入开发数据库的名称。例如，输入 Demo_Dev。
5. 从数据库供应商类型下拉列表中，为您的开发环境选择适当的数据库类型:

   DB2

   - Business Professional **DB2 通用数据库版本 8.1**

- Express **DB2 通用数据库版本 8.1 精简版**

Oracle **Oracle 9i**

6. 单击**完成**。新数据库定义已创建。
7. 浏览至以下目录:
   WebSphereCommerceServerExtensionsData > ejbModule > META-INF > Schema > *developmentDB*。
8. 用鼠标右键单击 *developmentDB* 并选择**新建 > 新模式定义**。
   “新模式定义”向导打开。
9. 在**模式名称**字段中，输入 NULLID 并单击**完成**。
10. 浏览到 **WebSphereCommerceServerExtensionsData > ejbModule > META-INF > 模式 >** *developmentDB* **> NULLID > 表**。
11. 用鼠标右键单击**表**并选择**新建表定义**。
    “新建表定义”向导打开。
12. 在**表名**字段，输入 XORDGIFT 并单击**下一步**。
13. 向表定义添加键列，如下:

    Business Professional

    a. 单击**添加另一个**。
    b. 在**列名**字段中输入 ORDERSID。
    c. 从**列类型**下拉列表选择以下内容:

       DB2 BIGINT

       Oracle NUMBER

    d. 选择**键列**。
    e. Oracle 在**数字精度**字段，输入 38。
    f. Oracle 将**数字标度**的值保留为 0。

    Express

    a. 单击**添加另一个**。
    b. 在**列名**字段中输入 ORDERSID。
    c. 选择**键列**。
    d. 从**列类型**下拉列表中选择以下内容:

       DB2 BIGINT

       Oracle NUMBER

e. Oracle 在**数字精度**字段，输入 38。

f. Oracle 将**数字标度**的值保留为 0。

14. 向表定义添加附加列，如下:

a. 单击**添加另一个**并用以下属性创建一列:

表 17.

属性	值
列名	RECEIPTNAME
列类型	DB2 VARCHAR Oracle VARCHAR2
可空	选择
字符串长度	50
对于位数据	清除

b. 单击**添加另一个**并用以下属性创建一列:

表 18.

属性	值
列名	SENDERNAME
列类型	DB2 VARCHAR Oracle VARCHAR2
可空	选择
字符串长度	50
对于位数据	清除

c. 单击**添加另一个**并用以下属性创建一列:

表 19.

属性	值
列名	MSGFIELD1
列类型	DB2 VARCHAR Oracle VARCHAR2
可空	选择
字符串长度	50

表 19. (续)

属性	值
对于位数据	清除

d. 单击**添加另一个**并用以下属性创建一列:

表 20.

属性	值
列名	MSGFIELD2
列类型	DB2 VARCHAR Oracle VARCHAR2
可空	选择
字符串长度	50
对于位数据	清除

e. 单击**完成**。

f. Oracle 您必须使用文本编辑器编辑表定义，方法如下:

1) 切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
2) 展开 **WebSphereCommerceServerExtensionsData** 项目。
3) 展开以下内容: **ejbModule > META-INF > 模式**。
4) 用鼠标右键单击 **WebSphereCommerceServerExtensionsData_NULL_XORDGIFT.xmi** 文件并选择**打开工具 > 文本编辑器**。
5) 将出现的所有 SQLNumeric_6 替换为 SQLNumeric_3。
6) 保存更改并关闭文本编辑器。

如果您已完成第 201 页的第 10 章，『教程: 创建新的业务逻辑』，执行以下操作以创建 XORDGIFT 表定义:

1. 打开“数据”视图并切换至“数据定义”视图。
2. 浏览至以下目录:
   WebSphereCommerceServerExtensionsData > ejbModule > META-INF > Schema > *developmentDB* **> NULLID > Tables**
3. 用鼠标右键单击 **Tables** 目录并选择**新建 > 新建表定义**。
   “新表定义”向导打开。

4. 在**表名**字段，输入 XORDGIFT 并单击**下一步**。
5. 向表定义添加键列，如下:

   Business Professional

   a. 单击**添加另一个**。
   b. 在**列名**字段中输入 ORDERSID。
   c. 从**列类型**下拉列表选择以下内容:

      DB2 BIGINT

      Oracle NUMBER

   d. 选择**键列**。
   e. Oracle 在**数字精度**字段，输入 38。
   f. Oracle 将**数字标度**的值保留为 0。

   Express

   a. 单击**添加另一个**。
   b. 在**列名**字段中输入 ORDERSID。
   c. 选择**键列**。
   d. 从**列类型**下拉列表中选择以下内容:

      DB2 BIGINT

      Oracle NUMBER

   e. Oracle 在**数字精度**字段，输入 38。
   f. Oracle 将**数字标度**的值保留为 0。
6. 向表定义添加附加列，如下:
   a. 单击**添加另一个**并用以下属性创建一列:

表 21.

属性	值
列名	RECEIPTNAME
列类型	DB2 VARCHAR Oracle VARCHAR2
可空	选择
字符串长度	50
对于位数据	清除

b. 单击**添加另一个**并用以下属性创建一列:

表 22.

属性	值
列名	SENDERNAME
列类型	DB2 VARCHAR Oracle VARCHAR2
可空	选择
字符串长度	50
对于位数据	清除

c. 单击**添加另一个**并用以下属性创建一列:

表 23.

属性	值
列名	MSGFIELD1
列类型	DB2 VARCHAR Oracle VARCHAR2
可空	选择
字符串长度	50
对于位数据	清除

d. 单击**添加另一个**并用以下属性创建一列:

表 24.

属性	值
列名	MSGFIELD2
列类型	DB2 VARCHAR Oracle VARCHAR2
可空	选择
字符串长度	50
对于位数据	清除

e. 单击**完成**。

7. Oracle 您必须使用文本编辑器编辑表定义，方法如下:

   a. 切换到Business Professional “J2EE 导航器”视图Express “项目导航器”视图。

   b. 展开 **WebSphereCommerceServerExtensionsData** 项目。

   c. 展开以下内容: **ejbModule > META-INF > 模式**。

   d. 用鼠标右键单击 **WebSphereCommerceServerExtensionsData_NULL_XORDGIFT.xmi** 文件并选择 **打开工具 > 文本编辑器**。

   e. 将出现的所有 SQLNumeric_6 替换为 SQLNumeric_3。

   f. 保存更改并关闭文本编辑器。

下一步是把 XORDGIFT 表映射至 OrderGiftBean 实体 bean。因为开发数据库和 XORDGIFT 表已存在，所以使用“在中间会面”映射。

如果您还未完成第 201 页的第 10 章，『教程: 创建新的业务逻辑』，请执行以下操作来创建映射:

1. 在“J2EE 层次结构”视图中，用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 项目并选择 **生成 > EJB 到 RDB 映射**。
   “创建新 EJB/RDB 映射”向导打开。
2. 选择**在中间会面**并单击**下一步**。
3. 选择**按名称和类型匹配**然后单击**完成**。
   Map.mapxmi 编辑器打开。
4. 在“企业 bean”窗格中，展开 **OrderGift** bean。在“表”窗格，展开 **XORDGIFT** 表。
5. 通过执行以下操作把 Bonus bean 中的字段映射为 XORDGIFT 表中的列:
   a. 用鼠标右键单击 **OrderGift** bean 并选择 **按名称匹配**。
6. 通过按 Ctrl+S 保存 Map.mapxmi 文件。关闭该文件。

如果您已完成第 201 页的第 10 章，『教程: 创建新的业务逻辑』，请执行以下操作来创建映射:

1. 在“J2EE 层次结构”视图中，用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 项目并选择**生成 > EJB 到 RDB 映射**。
   Map.mapxmi 编辑器打开。

2. 在“企业 bean”窗格中，展开 **OrderGift** bean。在“表”窗格，展开 **XORDGIFT** 表。
3. 通过执行以下操作把 Bonus bean 中的字段映射为 XORDGIFT 表中的列:
   a. 用鼠标右键单击 **OrderGift** bean 并选择 **按名称匹配**。
4. 通过按 Ctrl+S 保存 Map.mapxmi 文件。关闭该文件。

一旦已经创建 OrderGiftBean 实体，并已正确映射模式，就可以创建实体 bean 的访问 bean。此访问 bean 可使应用程序访问 OrderGift 实体 bean 中包含的信息变得更为简化。WebSphere Studio Application Developer 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 OrderGift 实体 bean 的访问 bean，请执行以下操作:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**，然后用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 并选择**新建 > 访问 bean**。“添加访问 bean”窗口打开。
2. 选择**复制帮手**并单击**下一步**。
3. 选择 **OrderGift** bean 并单击**下一步**。
4. 从**构造函数方法**下拉列表，选择 **findByPrimaryKey(com.ibm.commerce.extension.objects.OrderGiftKey)**。
5. 选择“属性帮手”部分的所有属性。
6. 单击**完成**。

您可通过切换至 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图来查看新生成的代码，方法是展开 **WebSphereCommerceServerExtensionsData** 项目并展开 **ejbModule** 子文件夹，然后展开 **com.ibm.commerce.extension.objects**。在数据包内创建并显示一个称为 OrderGiftAccessBean 的新类和一个称为 OrderGiftAccessBeanData 的新接口。

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。它还生成通过 IIOP 进行的 RMI 访问所必需的 Java ORB、存根和 tie 类以及主接口和远程接口的存根

要生成部署代码，请执行以下操作:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**，然后用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 并选择**生成 > 部署和 RMIC 代码**。
   “部署和 RMIC 代码”窗口打开。
2. 选择 **OrderGift** 并单击**完成**。

您可以通过切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图来查看新生成的代码。您会看到如下内容:

表 25.

代码类型	类名
容器实现生成代码	EJSCMPOrderGiftHomeBean.java
	EJSRemoteCMPOrderGift.java
	EJSRemoteCMPOrderGiftHome.java
	EJSFinderOrderGiftBean.java
JDBC 访问代码	EJSJDBCPersisterCMPOrderGiftBean.java
RMI 约束和存根代码	_EJSRemoteCMPOrderGift_Tie.java
	_OrderGift_Stub.java
	_EJSRemoteCMPOrderGiftHome_Tie.java
	_OrderGiftHome_Stub.java

将 OrderGift 实体 bean 集成到购物流程

在本节中，您将通过执行以下操作把 OrderGift 实体 bean 集成到样本商店的常规购物流程:

1. 创建新 OrderGiftDataBean 数据 bean
2. 创建新 MyExtOrderProcessCmdImpl 任务命令
3. 修改 OrderSubmitForm.jsp 显示页

后面各部分中将详细描述上面每个步骤。

创建 OrderGiftDataBean

必须创建 OrderGiftDataBean 以使 OrderGift 实体 bean 的属性得以显示在 JSP 显示页上。对于教程的其它部分，提供了基本代码，您需要对代码各部分取消注释。

要创建 OrderGiftDataBean，请执行以下操作:

1. 第一步是导入新数据 bean 的基本代码，如下:

a. 确保您已经完成第 202 页的『找到样本代码』中的步骤。

b. 切换到“J2EE 透视图”，然后选择Business Professional “J2EE 导航器”视图

Express “项目导航器”视图。

c. 展开 **WebSphereCommerceServerExtensionsLogic** 项目。

d. 用鼠标右键单击 **src** 文件夹并选择**导入**。
“导入”向导打开。

e. 从**选择导入源**列表中，选择 **Zip 文件**并单击**下一步**。

f. 单击**浏览**（在 **Zip 文件**字段旁）并浏览至样本代码。此文件的位置如下:
yourDirectory\WC_SAMPLE_55.zip
其中 *yourDirectory* 是您将数据包下载至的目录。

g. 单击**全部不选**，然后展开目录并选择要导入的以下文件:

- com\ibm\commerce\sample\databeans\OrderGiftDataBean.java

h. 在**文件夹**字段中，`WebSphereCommerceServerExtensionsLogic/src` 文件夹已经指定。保留该值。

i. 单击**完成**。

一旦您导入 bean 代码，请检查这些代码。

创建 MyExtOrderProcessCmdImpl 类

在本部分中，您将在 OrderProcess 业务流程的结束处添加新逻辑，从而在 XORDGIFT 数据库表中更新与礼物订单相关的信息。提供 ExtOrderProcessCmdImpl 命令作为该业务流程的扩展点。遵循编程模型，要扩展该逻辑，需要创建任务命令的新实现类并在该类中包含新逻辑。然后您必须更新命令注册表，以将新实现类和 ExtOrderProcessCmd 接口相关联。

要创建 MyExtOrderProcessCmdImpl 类，请执行以下操作:

1. 第一步是导入新命令的基本代码，如下:

a. 展开 **WebSphereCommerceServerExtensionsLogic** 项目。

b. 用鼠标右键单击 **src** 文件夹并选择**导入**。
“导入”向导打开。

c. 作为导入源，选择 **Zip 文件**并单击**下一步**。

d. 从**选择导入源**列表中，选择 **Zip 文件**并单击**下一步**。

e. 单击**浏览**（在 **Zip 文件**字段旁）并浏览至样本代码。此文件的位置如下:
yourDirectory\WC_SAMPLE_55.zip
其中 *yourDirectory* 是您将数据包下载至的目录。

f. 单击**全部不选**，然后展开目录并选择要导入的以下文件:

- com\ibm\commerce\sample\commands\ MyExtOrderProcessCmdImpl.java

g. 在**文件夹**字段中，`WebSphereCommerceServerExtensionsLogic/src` 文件夹已经指定。保留该值。

h. 单击**完成**。

一旦您导入此新命令的代码，您可以检查源代码以了解该命令。注意，该命令调用其超类的 performExecute() 方法以确保执行来自该命令的任何处理。然后它包含了设置礼物订单信息到新的数据 bean 中的逻辑。

在本步骤中，您需要修改命令注册表，这样可用使用 MyExtOrderProcessCmdImpl 实现类而不是原始 ExtOrderProcessCmdImpl 实现类。命令注册表中唯一需要修改的表是 CMDREG 表。在本例中，新实现类用于所有商店。

要修改命令注册表，请执行以下操作:

DB2 如果正在使用 DB2 数据库，请执行以下操作注册 MyExtOrderProcessCmdImpl:

1. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 命令行工具 > 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. 关闭工具设置。
5. 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 developmentDB user dbuser using dbpassword;
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
   CLASSNAME, TARGET)
values (FashionFlow_storeent_ID,
   'com.ibm.commerce.order.commands.ExtOrderProcessCmd',
   'This is a new task command for tutorial two.',
   'com.ibm.commerce.sample.commands.MyExtOrderProcessCmdImpl',
   'local');
```

其中

- *developmentDB* 是开发数据库的名称
- *dbuser* 是数据库用户
- *dbpassword* 是数据库用户的密码
- *FashionFlow_storeent_ID* 是样本商店的商店标识

单击**执行**图标。

Oracle 如果正在使用 Oracle 数据库，请执行以下操作注册 MyOrderItemAddCmdImpl:

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:

```
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
   CLASSNAME, TARGET)
values (FashionFlow_storeent_ID,
   'com.ibm.commerce.order.commands.ExtOrderProcessCmd',
   'This is a new task command for tutorial two.',
   'com.ibm.commerce.sample.commands.MyExtOrderProcessCmdImpl',
   'local');
```

   按 Enter 键运行 SQL 语句。
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。

编译更改

注意事项: 请确保没有重构商店 Web 项目。

在本部分中，您将编译已对代码所作的更改，如下:

1. 在 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图中，选择以下项目:
   - WebSphereCommerceServerExtensionsData
   - WebSphereCommerceServerExtensionsLogic
2. 在以上项目突出显示的情况下，用鼠标右键单击并选择**构建项目**。

修改礼物消息的显示页

在本步骤中，您将修改 OrderSubmitForm 和 OrderDetailDisplay 模板，从而使客户可以输入礼物订单的消息信息，以及在摘要页中查看该信息。修改这些页面的策

略是要包括附加的 JSP 模板，这些模板为页面指定新信息。这些新的页面（OrderSubmitFormInclude.jsp 和 OrderDetailDisplayInclude.jsp）使用 JSTL 以显示新的信息。

要修改显示页，请执行以下操作:

1. 切换到 Web 透视图，并使用 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
2. 如果您没有完成第 201 页的第 10 章，『教程: 创建新的业务逻辑』，则必须修改 Stores web 项目的属性，如下:
   a. 用鼠标右键单击 **Stores** Web 项目并选择**属性**。
   b. Business Professional 在左窗格中选择 **Web**，然后从“可用的 Web 项目功能”列表中选择**包含 JSP 标准标记库**。

      Express 在左窗格中选择 **Web 项目功能**，然后从“可用的 Web 项目功能”列表中选择 **JSP 标准标记库**。

      单击**应用**。更新完成后，单击**确定**以关闭属性编辑器。
3. 展开到以下目录:
   Stores\Web Content*FashionFlow_name*。
4. 通过执行以下操作，创建 OrderSubmitForm.jsp 文件的备份副本:
   a. 展开 **ShoppingArea>CheckoutSection>StandardCheckoutSubsection** 目录。
   b. 用鼠标右键单击 **OrderSubmitForm.jsp** 文件并选择**重命名**。
   c. 在“重命名”窗口中，输入 `OrderSubmitForm_bak.jsp` 并单击**确定**。
   d. 在提示时，如果您希望更新此文件的链接，请单击**否**。
5. 通过执行以下操作，创建 OrderDetailDisplay.jsp 文件的备份副本:
   a. 展开 **UserArea>ServiceSection>TrackOrderStatusSubsection** 目录。
   b. 用鼠标右键单击 **OrderDetailDisplay.jsp** 文件并选择**重命名**。
   c. 在“重命名”窗口中，输入 `OrderDetailDisplay_bak.jsp` 并单击**确定**。
   d. 在提示时，如果您希望更新此文件的链接，请单击**否**。
6. 用鼠标右键单击 *FashionFlow_name* 目录并选择**导入**。
   “导入”向导打开。
7. 从**选择导入源**列表中，选择 **Zip 文件**并单击**下一步**。
8. 单击**浏览**（在 **Zip 文件**字段旁）并浏览至样本代码。此文件的位置如下:
   yourDirectory\WC_SAMPLE_55.zip
   其中 *yourDirectory* 是您将数据包下载至的目录。

9. 单击**全部不选**，然后展开目录并选择要导入的以下文件:
   - ShoppingArea\CheckoutSection\StandardCheckoutSubsection\ OrderSubmitForm.jsp
   - ShoppingArea\CheckoutSection\StandardCheckoutSubsection\ OrderSubmitFormInclude.jsp
   - UserArea\ServiceSection\TrackOrderStatusSubsection\ OrderDetailDisplay.jsp
   - UserArea\ServiceSection\TrackOrderStatusSubsection\ OrderDetailDisplayInclude.jsp
10. 在**文件夹**字段，Stores/Web Content/*FashionFlow_name* 文件夹已指定。保留该值。
11. 单击**完成**。

如果您检查 OrderSubmitForm.jsp 文件，您将会发现包含了以下内容:

```
<!-- tutorial start-->
   <jsp:include page="OrderSubmitFormInclude.jsp" flush="true" />
<!-- tutorial done -->
```

这就是将新的 OrderSubmitFormInclude.jsp 文件编入到现有 JSP 模板中的方法。检查 OrderSubmitFormInclude.jsp 以了解如何在您的模板中使用 JSTL 的示例。类似地，检查 OrderDetailDisplay.jsp 和 OrderDetailDisplayInclude.jsp 文件。

您还必须导入包含已修改的 JSP 模板中使用的字符串值的属性文件。此文件名为 ordergift.properties。要导入此文件，执行以下操作:

1. 在Business Professional“J2EE 导航器”视图Express“项目导航器”视图中，展开以下目录:
   Stores > Web Content> WEB-INF >classes > *FashionFlow_name* 目录。
2. 用鼠标右键单击 *FashionFlow_name* 目录并选择**导入**。
   “导入”向导打开。
3. 从**选择导入源**列表中，选择 **Zip 文件**并单击**下一步**。
4. 单击**浏览**（在 **Zip 文件**字段旁）并浏览至样本代码。此文件的位置如下:
   yourDirectory\WC_SAMPLE_55.zip
   其中 *yourDirectory* 是您将数据包下载至的目录。
5. 单击**全部不选**，然后展开目录并选择要导入的以下文件:
   - ordergift.properties
6. 在**文件夹**字段中，Stores/Web Content/WEB-INF/classes/*FashionFlow_name* 文件夹已指定。保留该值。
7. 单击**完成**。

测试新礼物消息功能

在本部分，您将测试新礼物消息功能，如下:

1. 切换至“服务器”视图（**视窗 > 打开视图 > 服务器**）。
2. 启动 Payment Server。
3. 用鼠标右键单击 **WebSphereCommerceServer** 服务器并选择**启动**（或**重新启动**）。
4. 打开 Web 浏览器并输入以下 URL:

   ```
   http://localhost/webapp/wcs/stores/servlet/FashionFlow/index.jsp
   ```

5. 作为新用户登录。例如，单击**注册**然后创建新的用户，或作为现有用户登录。
6. 作为新注册用户浏览商店并向购物车添加商品，然后结束购物。您可以向订单添加一个礼物消息，如以下屏幕快照所示:

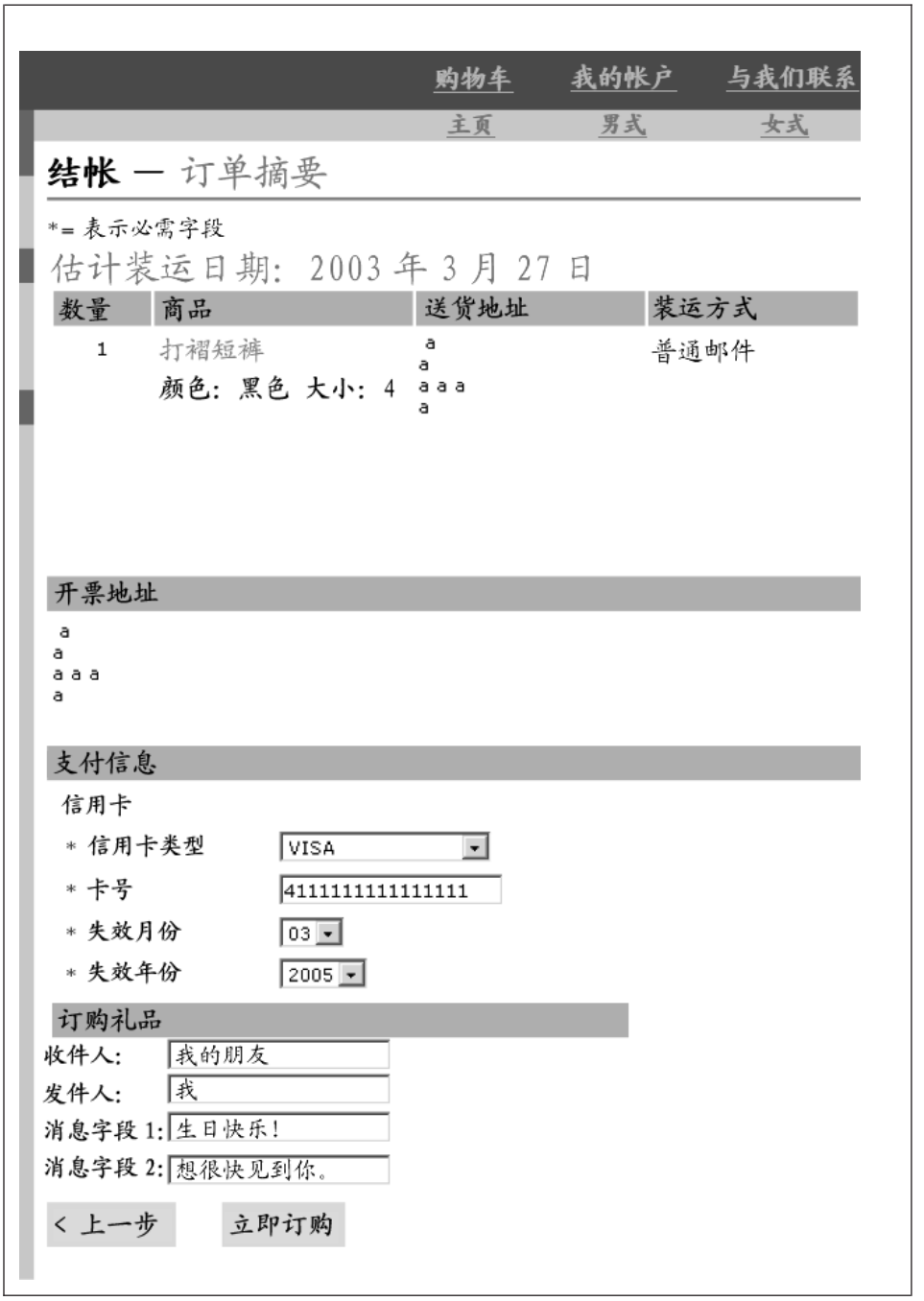

图 49.

注意与此订单相关联的订单号。

7. 单击**我的帐户**。
8. 单击**查看订单**。
9. 选择您在步骤 6 中创建的订单。您将会看到与以下类似的屏幕:

图 50.

部署礼物消息功能

本部分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“时尚潮流”样本商店）。在该商店中，您必须能够完成购物。

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。

有许多不同类型的有用资源，它们必须部署到目标 WebSphere Commerce Server。它们包括:

- 任务命令和数据 bean 逻辑
- 企业 bean 逻辑
- 已更新 JSP 模板
- 数据库更新包括模式更新（新表）以及命令注册表更新

本节描述如何以*递增*方式将所有这些有用资源部署到目标 WebSphere Commerce Server。这是作为递增部署完成的，与对整个 EAR 文件进行的部署相对。

创建命令和数据 bean JAR 文件

本部分描述了如何创建包含控制器命令、任务命令和数据 bean 逻辑的 JAR 文件。

要创建此 JAR 文件,请在开发机上执行以下步骤:

1. 在本地文件系统上创建名为 \ExportTemp3 的目录。
2. 打开 WebSphere Studio Application Developer 并切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
3. 用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 项目并选择**导出**。
   “导出”向导打开。
4. 在“导出”向导中，请执行以下操作:
   a. 选择 **JAR 文件**并单击**下一步**.
   b. **选择要导出的资源**左下窗格是以项目的名称预先填充的。保留此值不变。
   c. 在右窗格中，确保仅选择了以下资源:
      - .classpath
      - .project
      - .serverPreference

d. 确保已选择**导出已生成的类文件和资源**。

e. 不要选择**导出 Java 源文件和资源**。

f. 在**选择导出目的地**字段中，输入要使用的全限定 JAR 文件名。在此例中，输入 *drive*:\ExportTemp3\WebSphereCommerceServerExtensionsLogic.jar。请注意，JAR 文件名必须是 WebSphereCommerceServerExtensionsLogic.jar。

g. 单击**完成**。

创建 EJB JAR 文件

要创建 EJB JAR 文件，请执行以下操作:

1. 打开 WebSphere Studio Application Developer 并切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
2. 展开 **WebSphereCommerceServerExtensionsData** 项目。
3. 双击 **EJB 部署描述符**。
4. 在选择“概述”选项卡的情况下，滚动至窗格底部以找到 **WebSphere 绑定**部分。
5. 在**数据源 JNDI 名称**字段，输入目标 WebSphere Commerce Server 的数据源 JNDI 名称。以下是值的示例:

   DB2 `jdbc/WebSphere Commerce DB2 DataSource demo`
   其中目标 WebSphere Commerce Server 使用 DB2 数据库，WebSphere Commerce 实例名称是“demo”

   Oracle `jdbc/WebSphere Commerce Oracle DataSource demo`
   其中目标 WebSphere Commerce Server 使用 Oracle 数据库，WebSphere Commerce 实例名称是“demo”。

数据源 JNDI 名称的值是通过将“jdbc/”添加到目标 WebSphere Commerce Server 的数据源名称来创建的。可通过打开目标 WebSphere Commerce Server 上的 *instanceName*.xml 文件并在该文件中搜索 DatasourceName= 来验证数据源名称。

6. 保存部署描述符更改（Ctrl+S）。
7. 在 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图中，用鼠标右键单击 **WebSphereCommerceServerExtensionsData** 项目，并选择**导出**。
   “导出”向导打开。
8. 在“导出”向导中，请执行以下操作:
   a. 选择 **EJB JAR 文件**并单击**下一步**。

b. Business Professional **您想要导出什么资源？**的值是以 EJB 项目的名称预填充的。

Express EJB 项目名称是预填充的。

保留该值不变。

c. Business Professional 在**您要将资源导出到哪里？**字段中，输入要使用的全限定 JAR 文件名。

Express 对于目的地，输入要使用的全限定 JAR 文件名。

在这种情况下，输入 *drive*:\ExportTemp3\WebSphereCommerceServerExtensionsData.jar。

d. 单击**完成**。

9. JAR 文件创建后，撤销在步骤 5 中对本地部署描述符所作更改，用以恢复本地测试服务器所必需的设置。

导出商店有用资源

要从 WebSphere Studio Application Developer 导出 OrderSubmitForm 和 OrderDetailDisplay 显示模板，请执行以下操作:

1. 打开 WebSphere Studio Application Developer 并切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
2. 展开 **Stores** 文件夹。
3. 用鼠标右键单击 **Web Content** 文件夹并选择**导出**。
   “导出向导”打开。
4. 在“导出”向导中，请执行以下操作:
   a. 选择**文件系统**并单击**下一步**。
   b. 选择要部署的以下资源:
      - Web Content*FashionFlow_name*\ShoppingArea\CheckoutSection\StandardCheckoutSubsection\ OrderSubmitForm.jsp
      - Web Content*FashionFlow_name*\ShoppingArea\CheckoutSection\StandardCheckoutSubsection\ OrderSubmitFormInclude.jsp
      - Web Content*FashionFlow_name*\UserArea\ServiceSection\TrackOrderStatusSubsection\OrderDetailDisplay.jsp
      - Web Content*FashionFlow_name*\UserArea\ServiceSection\TrackOrderStatusSubsection\ OrderDetailDisplayInclude.jsp
      - Web Content\WEB-INF\lib\jstl.jar
      - Web Content\WEB-INF\lib\standard.jar

- Web Content\WEB-INF\classes*FashionFlow_name*\ordergift.properties

c. 选择**为文件创建目录结构**。

d. 在“目录”字段中输入放置这些资源的临时目录。例如，输入 `C:\ExportTemp3`

e. 单击**完成**。

将有用资源传送到目标 WebSphere Commerce Server

在本步骤中，您将在目标 WebSphere Commerce Server 中创建一个临时目录并把您的礼品订单有用资源复制到该目录。在后面的步骤中，您将把不同类型的代码置于 WebSphere Commerce 应用程序中的合适位置。

要把文件从开发机复制到目标 WebSphere Commerce Server，请执行以下操作:

1. 在目标 WebSphere Commerce Server 上，创建一个称为 *drive*:\ImportTemp3 的临时目录。
2. 确定如何从一台计算机向另一台计算机复制文件。您可以通过将目标 WebSphere Commerce Server 上的驱动器映射到开发机或是使用 FTP 应用程序（如果已配置）来实现这一目的。
3. 从开发机器上将 \ExportTemp3 的内容复制到目标 WebSphere Commerce Server 上的 \ImportTemp3。

停止目标 WebSphere Commerce Server

在开始部署步骤之前，应通过在命令行发出 stopServer 命令来停止目标 WebSphere Commerce Server。有关该命令的详细信息，请参考Business Professional *《WebSphere Commerce Studio 安装指南》*或 Express *《WebSphere Commerce - Express Developer Edition 安装指南》*。

更新目标 WebSphere Commerce Server 上的数据库

注册任务命令实现

要修改命令注册表，请执行以下操作:

1. DB2 如果正在使用 DB2 数据库，请执行以下操作注册 MyExtOrderProcessCmdImpl:
   a. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 命令行工具 > 控制中心**）。
   b. 从**工具**菜单中，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. 关闭工具设置。

e. 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 targetDB user dbuser using dbpassword;
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
   CLASSNAME, TARGET)
values (FashionFlow_storeent_ID,
   'com.ibm.commerce.order.commands.ExtOrderProcessCmd',
   'This is a new task command for tutorial two.',
   'com.ibm.commerce.sample.commands.MyExtOrderProcessCmdImpl',
   'local');
```

其中

- *targetDB* 是目标数据库的名称
- *dbuser* 是数据库用户
- *dbpassword* 是数据库用户的密码
- *FashionFlow_storeent_ID* 是样本商店的商店标识

单击**执行**图标。保持“命令中心”打开。

2. Oracle 如果正在使用 Oracle 数据库，请执行以下操作注册 MyOrderItemAddCmdImpl:

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 SQL 语句:

```
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
   CLASSNAME, TARGET)
values (FashionFlow_storeent_ID,
   'com.ibm.commerce.order.commands.ExtOrderProcessCmd',
   'This is a new task command for tutorial two.',
   'com.ibm.commerce.sample.commands.MyExtOrderProcessCmdImpl',
   'local');
```

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。

创建 XORDGIFT 表

在此步骤中，您将创建 XORDGIFT 表。

DB2 如果正在使用 DB2 数据库，请执行以下操作创建表:

1. 在“脚本”窗口中，输入以下内容:

   ```
   create table XORDGIFT (ORDERSID bigint not null, RECEIPTNAME varchar(50),
   SENDERNAME varchar(50), MSGFIELD1 varchar(50), MSGFIELD2 varchar(50),
   constraint p_xordgift primary key (ORDERSID),
   constraint f_xordgift foreign key (ORDERSID) references ORDERS(ORDERS_ID)
   on delete cascade);
   insert into XORDGIFT (ORDERSID) (select ORDERS_ID from ORDERS);
   ```

   其中

   - *targetDB* 是目标数据库的名称
   - *dbuser* 是数据库用户
   - *dbpassword* 是数据库用户的密码

   单击“执行”图标。
   现在，XORDGIFT 表已创建。

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 XORDGIFT (ORDERSID NUMBER NOT NULL, RECEIPTNAME VARCHAR(50),
   SENDERNAME VARCHAR(50), MSGFIELD1 VARCHAR(50), MSGFIELD2 VARCHAR(50),
   constraint p_xordgift primary key (ORDERSID),
   constraint f_xordgift foreign key (ORDERSID) references ORDERS(ORDERS_ID)
   on delete cascade);
   insert into XORDGIFT (ORDERSID) (select ORDERS_ID from ORDERS);
   ```

   并按 Enter 键运行 SQL 语句。XORDGIFT 表已创建。
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。

更新目标 WebSphere Commerce Server 上的商店有用资源

在本步骤中，您将用已修改的商店有用资源来更新商店，如下:

1. 备份 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear\ Stores.war 目录。
2. 浏览至 \ImportTemp3\Stores\Web Content 目录。
3. 将 *FashionFlow_name* 文件夹复制到以下目录中:
   WAS_installdir\installedApps*cellName*\WC_*instanceName*.ear\ Stores.war

其中 *instanceName* 是 WebSphere Commerce 实例的名称。

更新目标 WebSphere Commerce Server 上的命令和数据 bean JAR 文件

在本步骤中您将更新目标 WebSphere Commerce Server 来使用新目标和数据 bean JAR 文件，如下:

1. 您应制作现有 JAR 文件的备份副本，如下:
   a. 浏览至 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear 目录。
   b. 制作 WebSphereCommerceServerExtensionsLogic.jar 文件的副本并将它保存在备份位置。
2. 将新的 WebSphereCommerceServerExtensionsLogic.jar 文件从 \ImportTemp3 目录复制到 *WAS_installdir*\nstalledApps*cellName*\WC_*instanceName*.ear 目录。

其中 *instanceName* 是 WebSphere Commerce 实例的名称。

更新目标 WebSphere Commerce Server 上的 EJB JAR 文件

在本步骤中您将更新目标 WebSphere Commerce Server 来使用新 EJB JAR 文件，如下:

1. 您应制作现有 JAR 文件的备份副本，如下:
   a. 浏览至 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear 目录。
   b. 制作 WebSphereCommerceServerExtensionsData.jar 文件的副本并将它保存在备份位置。

   其中 *instanceName* 是 WebSphere Commerce 实例的名称。
2. 将新的 WebSphereCommerceServerExtensionsData.jar 文件从 \ImportTemp3 目录复制到 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear 目录
3. 下一步，您必须修改 EJB 部署描述符信息，如下:
   a. 定位这一 WebSphere Application Server 单元的部署资源库（META-INF 目录）。这通常采取以下格式:
      WAS_installdir\config\cells*cellName*
      \applications\WC_*instance_name*.ear\deployments\

WC_*instance_name**EJBModuleName*.jar\META-INF
以下是它的一个特定示例:
D:\WebSphere\AppServer\config\cells\myCell\applications\WC_demo.ear\deployments\WC_demo\WebSphereCommerceServerExtensionsData.jar\META-INF
其中

- mycell 是 WebSphere Application Server 单元的名称
- demo 是 WebSphere Commerce 实例的名称

b. 该目录包含以下文件:

- ejb-jar.xml
- ibm-ejb-access-bean.xmi
- ibm-ejb-jar-bnd.xmi
- ibm-ejb-jar-ext.xmi
- MANIFEST.MF

备份所有这些文件。

c. 使用工具打开新 WebSphereCommerceServerExtensionsData.jar 文件并查看其内容。

d. 把 meta-inf 目录的内容从该 WebSphereCommerceServerExtensionsData.jar 文件解压缩到步骤 3a 的目录中。

4. 在命令行中使用 WebSphere Application Server startServer 命令，重新启动 WebSphere Commerce 实例。

验证目标 WebSphere Commerce Server 上的礼物消息功能

在此部分，通过执行以下操作，验证礼物消息逻辑在目标 WebSphere Commerce Server 上是否正常运作:

1. 打开 Web 浏览器并输入基于“时尚潮流”样本商店的您的商店的 URL。
2. 作为新用户登录。例如，单击“注册”，然后创建用户“shopper”。
3. 作为新注册用户浏览商店并向购物车添加商品，然后结束购物。您可以向订单添加一个礼物消息，如以下屏幕快照所示:

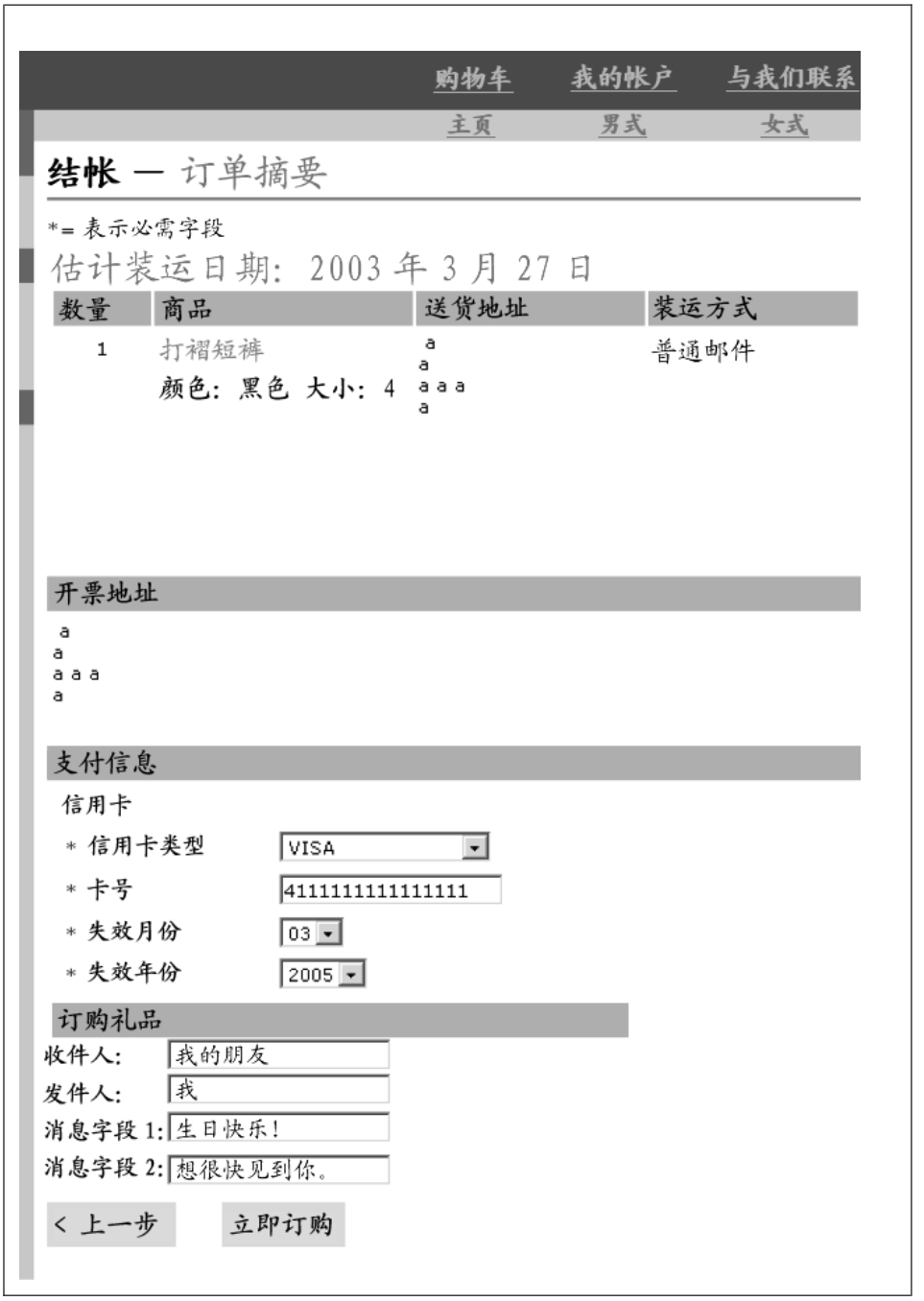

图 51.

注意与此订单相关联的订单号。

4. 单击**我的帐户**。
5. 单击**查看订单**。
6. 选择您在步骤 3 中创建的订单。您将会看到与以下类似的屏幕:

图 52.

第 13 章 教程: 扩展现有的 WebSphere Commerce 实体 bean

在该教程中，您将学习修改现有 WebSphere Commerce 实体 bean 的过程。它使用添加附加驻留信息调查到用户注册过程的方案。在此例中，修改了 User 实体 bean 以包括附加字段，该字段存储用户驻留类型值和位置值。创建了新的数据库表（称为 XHOUSING）并且修改了 User bean 的现有映射信息以包括该新表。

除了新表和实体 bean 更改之外，也创建了新的 MyPostUserRegistrationAddCmdImpl 实现类。这包含处理驻留调查信息和用新信息更新数据库的逻辑。

为了能收集驻留信息及在以后显示结果，修改了 UserRegistrationAddForm.jsp 和 UserRegistrationUpdateForm.jsp 文件。

先决条件

本教程并不要求您已完成某些教程的内容。如果您已完成这些教程，那么把代码留在工作区没有坏处，因为它不会与本教程冲突。

在开始本教程前，必须已经发布了一个基于“时尚潮流”样本商店的商店。在这个商店里，您必须能够完成一项购买（例如，浏览产品目录，向购物车添加商品，检出并查看订单确认）。

创建并填充 XHOUSING 表

在创建实体 bean 的准备中，必须首先创建并填充新的数据库表。要创建的表称为 XHOUSING。

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 > 命令行工具 > 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 developmentDB user dbuser using dbpassword;
create table XHOUSING (MEMBERID bigint not null,
    HOUSINGTYPE integer, LOCATION integer,
   constraint p_xhousing primary key (MEMBERID),
   constraint f_xhousing foreign key (MEMBERID)
    references USERS (USERS_ID) on delete cascade);
insert into XHOUSING (MEMBERID) (select USERS_ID from USERS);
```

其中

- *developmentDB* 是开发数据库的名称
- *dbuser* 是数据库用户
- *dbpassword* 是数据库用户的密码

单击“执行”图标。
您应该看到一条消息，指示 SQL 语句已成功完成。

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 XHOUSING (MEMBERID number not null,
    HOUSINGTYPE integer, LOCATION integer,
   constraint p_xhousing primary key (MEMBERID),
   constraint f_xhousing foreign key (MEMBERID)
    references USERS (USERS_ID) on delete cascade);
insert into XHOUSING (MEMBERID) (select USERS_ID from USERS);
```

并按 Enter 键运行 SQL 语句。XHOUSING 表已创建。

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。

将一个新的字段添加到 User 实体 bean

在这一部分，添加两个新的字段到 User 实体 bean，这些 bean 是用于捕获住房信息的。新的字段名为 housingType 和 location。这些字段最终映射为 XHOUSING 表的 HOUSINGTYPE 和 LOCATION 列。

要添加这些新字段，请执行以下操作:

1. 启动 WebSphere Commerce 开发环境如下:
   - Business Professional **开始 > 程序 > IBM WebSphere Commerce Studio > WebSphere Commerce 开发环境**

- Express Express **开始 > 程序 > IBM WebSphere - Express Developer Edition > WebSphere Commerce 开发环境**

2. 切换到服务器视图并确保 WebSphereCommerceServer 测试服务器已停止。
3. 切换至 J2EE 视图并选择“J2EE 层次结构”视图。
4. 展开 **EJB 模块**，然后双击 **Member-MemberManagementData** EJB 项目。这一操作会打开 EJB 部署描述符编辑器。
5. 单击 **Beans** 选项卡并从显示的 bean 列表中选择 **User** bean。
6. 单击 CMP 字段文本框旁边的**添加**。
   “创建 CMP 属性”窗口打开。
7. 使用以下属性创建 CMP 字段:
   a. 在**名称**字段中，输入 `housingType`。
   b. 在“类型”字段中，输入 `java.lang.Integer`。
   c. 启用**以 getter 和 setter 方法访问**。
   d. 清除**将 getter 和 setter 方法升级到远程界面**复选框。（这会自动清除使 getter 成为只读选项的选项。）
   e. 单击**确定**。
8. 再次单击**添加**以使用以下属性创建另一个 CMP 字段:
   a. 在**名称**字段中，输入 `location`。
   b. 在“类型”字段中，输入 `java.lang.Integer`。
   c. 启用**以 getter 和 setter 方法访问**。
   d. 清除**将 getter 和 setter 方法升级到远程界面**复选框。（这会自动清除使 getter 成为只读选项的选项。）
   e. 单击**确定**。

   这些新字段会显示在 CMP 字段列表中。
9. 保存更改，然后关闭 EJB 部署描述符编辑器。

更新模式和表映射信息

在以下部分中，您将使用新的 XHOUSING 表更新 User 模式，创建新表的外键关系，并在 User 实体 bean 各字段与 XHOUSING 表各列之间创建一个表映射。使用这一方法扩展对象模型，代码看上去象是将新的列直接添加到了 USERS 表。

创建 XHOUSING 表的表定义

要创建 XHOUSING 表定义以及创建 USERS 表和 XHOUSING 表之间的外键关系，请执行以下操作:

1. 在“J2EE 层次结构”视图中，展开**数据库**。
2. 展开 **Member-MemberManagementData** 数据库,然后展开 **NULLID** 模式。
3. 用鼠标右键单击**表**并选择**新建 > 新建表定义**。
   “表定义”窗口打开。
4. 在**表名**字段中，输入 XHOUSING 并单击**下一步**。
5. 向表定义添加键列，如下:

   Business Professional

   a. 单击**添加另一个**.
   b. 在**列名**字段中输入 ORDERSID。
   c. 从**列类型**下拉列表选择以下内容:

      DB2 BIGINT

      Oracle NUMBER

   d. 选择**键列**。
   e. Oracle 在**数字精度**字段，输入 38。
   f. Oracle 将**数字标度**的值保留为 0。

   Express

   a. 单击**添加另一个**.
   b. 在**列名**字段中输入 ORDERSID。
   c. 选择**键列**。
   d. 从**列类型**下拉列表中选择以下内容:

      DB2 BIGINT

      Oracle NUMBER

   e. Oracle 在**数字精度**字段，输入 38。
   f. Oracle 将**数字标度**的值保留为 0。
6. 向表定义添加附加列，如下:
   a. 单击**添加另一个**并用以下属性创建一列:

表 26.

属性	值
列名	HOUSINGTYPE
列类型	INTEGER

表 26. (续)

属性	值
可空	选择

b. 单击**添加另一个**并用以下属性创建一列:

表 27.

属性	值
列名	LOCATION
列类型	INTEGER
可空	选择

c. 单击**下一步**。

7. 在**主键名**字段中输入 p_xhousing 并单击**下一步**。
8. 单击**添加另一个**以添加外键。指定以下值:
   a. 在**外键名**字段中，输入 f_xhousing。
   b. 从**删除时**下拉列表中选择 **CASCADE**
   c. 从**目标表**下拉列表中，选择 **NULLID.USERS**
   d. 在“源列”窗格，单击 **MEMBERID** 并单击 **>** 添加外键。
9. 单击**完成**。
10. Oracle 您必须使用文本编辑器编辑表定义，方法如下:
    a. 切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
    b. 展开 **Member-MemberManagementData** 项目。
    c. 展开以下内容: **ejbModule > META-INF > 模式**。
    d. 用鼠标右键单击 **Member-MemberManagementData_NULL_XHOUSING.xmi** 文件并选择**打开工具 > 文本编辑器**。
    e. 将出现的所有 SQLNumeric_6 替换为 SQLNumeric_3。
    f. 保存更改并关闭文本编辑器。

新的 Member-MemberManagementData_NULL_XHOUSING 表定义可以通过展开 Member-MemberManagementData/NULLID 下的 Tables 文件夹查看。

创建 XHOUSING 表映射

在本部分中，您将创建 XHOUSING 表中的两列（HOUINGTYPE 和 LOCATION）和 User 实体 bean 中的两个字段（housingType 和 location）间的映射。

要创建此映射，请执行以下操作:

1. 切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
2. 展开以下文件夹: **Member-MemberManagementData > ejbModule > META-INF**。
3. 双击 **Map.mapxmi** 文件。
4. Express 在“企业 bean”窗格中展开 **Member** 组，然后用鼠标右键单击 **User** 实体 bean。
5. 在“表”窗格中，突出显示以下表:
   - **MEMBER**
   - **USERS**
   - **XHOUSING**

   （按住 Ctrl 键可一次选择多个表。）
6. Business Professional 在“企业 bean”窗格中（在编辑器顶部），展开 **Member** 组，然后用鼠标右键单击 **User** 实体 bean 并选择**创建映射**。

   Express 在“企业 bean”窗格中（在编辑器顶部），用鼠标右键单击 **User** 实体 bean 并选择**创建映射**。
7. 展开 **User** 实体 bean。
8. 在“表”窗格中，展开 **XHOUSING** 表，这样就可以查看其列。
9. 突出显示 **housingType** bean 属性并把它拖动到 **HOUSINGTYPE** 列以创建映射。
10. 突出显示 **location** bean 属性并把它拖动到 **LOCATION** 列以创建映射。
11. 保存更改，并关闭 Map.mapxmi 文件。

更新映射文件

1. 展开以下文件夹: **Member-MemberManagementData > ejbModule > META-INF**。
2. 用鼠标右键单击 **Map.mapxmi** 文件并选择**打开工具 > 文本编辑器**。
3. 找到以下字符串:

   ```
   User_EJB
   ```

   在这之下两行，您可以看到:

   ```
   <discriminatorValues>User</discriminatorValues>
   ```

如果您发现上述行，那么您必须将其更改为

```
<discriminatorValues>'U'</discriminatorValues>
```

4. 保存更改。

生成访问 bean 和部署代码

由于已经修改 User 实体 bean，因此必须重新生成它的访问 bean 和部署代码。

要重新生成访问 bean，请执行以下操作:

1. 在“J2EE 层次结构”视图中，展开 **EJB 模块**。
2. 用鼠标右键单击 **Member-MemberManagementData** EJB 模块并选择**访问 bean> 编辑访问 Bean**。
3. 选择 **CopyHelper** 并单击**下一步**。
4. 在“选择 EJB 项目”窗口，单击**下一步**。
5. 在“复制帮手访问 bean”窗口，请执行以下操作:
   a. 从企业 bean 下拉列表，选择 **User**。
   b. 从**构造函数方法**下拉列表中，选择 **findByPrimaryKey(com.ibm.commerce.user.objects.MemberKey)**。
   c. 从属性帮手列表中，确保 **housingType** 和 **location** 属性已和所有其它属性一同选定。
6. 单击**完成**。
7. 用鼠标右键单击 **Member-MemberManagementData** EJB 模块并选择**访问 bean> 重新生成访问 Bean**。
8. 单击**全部选中**，然后单击**完成**。

要重新生成部署代码，请执行以下操作:

1. 用鼠标右键单击 **Member-MemberManagementData** EJB 模块并选择**生成 > 部署和 RMIC 代码**。
2. 单击**全部选中**，然后单击**完成**。

创建 MyPostUserRegistrationAddCmdImpl 实现

在这一步中，您需要扩展 UserRegistrationAddCmd 控制器命令以包含用于分析新住房信息的新逻辑，这些信息是在注册表单中收集到的。这一扩展是通过创建新的实现 PostUserRegistrationCmd 接口的 MyPostUserRegistrationAddCmdImpl 实现类完成的。创建这一新的实现类后，必须更新命令注册表以反映这一更改。其它教程都提供了这一新命令的代码。

要实现新的 MyPostUserRegistrationAddCmdImpl 实现类，请执行以下操作:

1. 确保您已经完成第 202 页的『找到样本代码』中的步骤。
2. 在 WebSphere Studio Application Developer 中，打开 Java 视图（**视窗 > 打开视图 > Java**）。
3. 展开 **WebSphereCommerceServerExtensionsLogic** 项目。
4. 用鼠标右键单击 **src** 文件夹并选择**导入**。
   “导入”向导打开。
5. 从**选择导入源**列表中，选择 **Zip 文件**并单击**下一步**。
6. 单击**浏览**（在 **Zip 文件**字段旁）并浏览至样本代码。此文件的位置如下:
   yourDirectory\WC_SAMPLE_55.zip
   其中 *yourDirectory* 是您将数据包下载至的目录。
7. 单击**全部不选**，然后展开目录并选择导入以下文件:
   - com\ibm\commerce\sample\commands\ MyPostUserRegistrationAddCmdImpl.java
8. 在**文件夹**字段中，WebSphereCommerceServerExtensionsLogic/src 文件夹已经指定。保留该值。
9. 单击**完成**。
10. 展开 **com.ibm.commerce.sample.commands** 数据包。
11. 双击新的 **MyPostUserRegistrationAddCmdImpl** 类。
12. 取消对第 1 部分的注解。这些代码设置变量并创建这些变量的 getter 和 setter:

```
/// Section 1 ////////////////
 Integer housingType = null;
 Integer location = null;

 public Integer getHousingType() {
  return housingType;
 }

 public Integer getLocation() {
  return location;
 }

 public void setHousingType(Integer newHousingType) {
  housingType = newHousingType;
 }

 public void setLocation(Integer newLocation) {
  location = newLocation;
 }
/// End of Section 1 ////////////////
```

13. 取消对第 2 部分的注解。这将以下代码引入到类中:

```
/// Section 2 ////////////////
public void performExecute() throws ECException {
  super.performExecute();

  // Set the needed fields before processing the survey
  setHousingType(requestProperties.getInteger("housingType", 0));
  setLocation(requestProperties.getInteger("location", 0));

  processSurvey();
 }

/// End of Section 2 ////////////////
```

上述代码调用超类的 performExecute 方法，这样将执行 PostUserRegistrationAddCmdImpl 的常规逻辑。一旦这些完成，就会调用新的 processSurvey 方法。

14. 取消对第 3 部分的注解，将以下代码引入到类中:

```
/// Section 3 ////////////////

 private void processSurvey() throws ECException {

  try {
   // load up the user data
   UserAccessBean abUser = new UserAccessBean();
   abUser.setInitKey_MemberId(commandContext.getUserId().toString());
   abUser.refreshCopyHelper();

   // store the new attributes
   abUser.setHousingType(getHousingType());
   abUser.setLocation(getLocation());

   abUser.commitCopyHelper();
     } catch (javax.ejb.FinderException e) {
throw new ECSystemException(
    ECMessage._ERR_FINDER_EXCEPTION,
    this.getClass().getName(),
    "processSurvey");
     } catch (javax.naming.NamingException e) {
throw new ECSystemException(
    ECMessage._ERR_NAMING_EXCEPTION,
    this.getClass().getName(),
    "processSurvey");
} catch (java.rmi.RemoteException e) {
throw new ECSystemException(
    ECMessage._ERR_REMOTE_EXCEPTION,
    this.getClass().getName(),
    "processSurvey");
} catch (javax.ejb.CreateException e) {
throw new ECSystemException(
    ECMessage._ERR_CREATE_EXCEPTION,
    this.getClass().getName(),
    "processSurvey");
```

```
    }

   }
/// End of Section 3 ////////////////
```

15. 保存更改。
16. 用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 项目并选择**构建项目**。

修改命令注册表

必须修改命令注册表，以便在购物流程中使用新的实现类。

要修改命令注册表，请执行以下操作:

1. DB2 如果正在使用 DB2 数据库，请执行以下操作注册 MyPostUserRegistrationAddCmdImpl:
   a. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 控制中心**）。
   b. 从**工具**菜单中，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 developmentDB user dbuser using dbpassword;
insert into CMDREG values (FashionFlow_storeent_Id,
'com.ibm.commerce.usermanagement.commands.PostUserRegistrationAddCmd'
'Command for modified user bean tutorial',
'com.ibm.commerce.sample.commands.MyPostUserRegistrationAddCmdImpl',
null, null, 'Local');
```

   其中

   - *developmentDB* 是开发数据库的名称
   - *dbuser* 是数据库用户
   - *dbpassword* 是数据库用户的密码
   - *FashionFlow_storeent_Id* 是商店的唯一商店实体标识。

   单击**执行**图标。

2. Oracle 如果正在使用 Oracle 数据库，请执行以下操作注册 MyPostUserRegistrationAddCmdImpl:
   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 SQL 语句:

```
insert into CMDREG values (FashionFlow_storeent_Id,
'com.ibm.commerce.usermanagement.commands.PostUserRegistrationAddCmd'
'Command for modified user bean tutorial',
'com.ibm.commerce.sample.commands.MyPostUserRegistrationAddCmdImpl',
null, null, 'Local');
```

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。

修改 JSP 模板以收集并显示住房信息

在这一步中，您需要修改 UserRegistrationAddForm 和 UserRegistrationUpdateForm 模板，这样客户在登录时才能输入住房信息，并且能在摘要页面中查看住房信息。修改这些页面的策略是要包括附加的 JSP 模板，这些模板为页面指定新信息。这些新的页面（UserRegistrationAddFormInclude.jsp 和 UserRegistrationUpdateFormInclude.jsp）使用 JSTL 显示新信息。

要修改这些页面，请执行以下操作:

1. 切换至 Web 视图。
2. 如果您没有完成第 201 页的第 10 章，『教程: 创建新的业务逻辑』，则必须修改 Stores web 项目的属性，如下:
   a. 用鼠标右键单击 **Stores** Web 项目并选择**属性**。
   b. Business Professional 在左窗格中选择 **Web**，然后从“可用的 Web 项目功能”列表中选择**包含 JSP 标准标记库**。

      Express 在左窗格中选择 **Web 项目功能**，然后从“可用的 Web 项目功能”列表中选择 **JSP 标准标记库**。

      单击**应用**。更新完成后，单击**确定**以关闭属性编辑器。
3. 展开到以下目录:
   Stores\Web Content*FashionFlow_name*。
4. 通过执行以下操作，创建 UserRegistrationAddForm.jsp 文件的备份副本:
   a. 展开 **UserArea>AccountSection>RegistrationSubsection** 目录。
   b. 用鼠标右键单击 **UserRegistrationAddForm.jsp** 文件并选择**重命名**。

c. 在“重命名”窗口中，输入 UserRegistrationAddForm_bak.jsp 并单击**确定**。

d. 在提示时，如果您希望更新此文件的链接，请单击**否**。

5. 通过执行以下操作，创建 UserRegistrationUpdateForm.jsp 文件的备份副本:
   a. 展开 **UserArea>AccountSection>RegistrationSubsection** 目录。
   b. 用鼠标右键单击 **UserRegistrationUpdateForm.jsp** 文件并选择**重命名**。
   c. 在“重命名”窗口中，输入 UserRegistrationUpdateForm_bak.jsp 并单击**确定**。
   d. 在提示时，如果您希望更新此文件的链接，请单击**否**。
6. 用鼠标右键单击 *FashionFlow_name* 目录并选择**导入**。
   “导入”向导打开。
7. 从**选择导入源**列表中，选择 **Zip 文件**并单击**下一步**。
8. 单击**浏览**（在 **Zip 文件**字段旁）并浏览至样本代码。此文件的位置如下:
   yourDirectory\WC_SAMPLE_55.zip
   其中 *yourDirectory* 是您将数据包下载至的目录。
9. 单击**全部不选**，然后展开目录并选择要导入的以下文件:
   - UserArea\AccountSection\RegistrationSubsection\ UserRegistrationAddForm.jsp
   - UserArea\AccountSection\RegistrationSubsection\ UserRegistrationAddFormInclude.jsp
   - UserArea\AccountSection\RegistrationSubsection\ UserRegistrationUpdateForm.jsp
   - UserArea\AccountSection\RegistrationSubsection\ UserRegistrationUpdateFormInclude.jsp
10. 在**文件夹**字段，Stores/Web Content/*FashionFlow_name* 文件夹已指定。保留该值。
11. 单击**完成**。

如果您检查 UserRegistrationAddForm.jsp 文件，您将会发现该教程中已经添加了以下部分:

```
<%-- Add for tutorial --%>
 <tr>
      <td colspan="3">
         <jsp:include page="UserRegistrationAddFormInclude.jsp" flush="true" />
  </td>
 </tr>
<%-- End of tutorial --%>
```

您也可以检查 UserRegistrationAddFormInclude.jsp 文件了解如何使用 JSTL 收集新信息。类似地，检查 UserRegistrationAddForm.jsp 和 UserRegistrationAddFormInclude.jsp 文件。

您还必须导入包含已修改的 JSP 模板中使用的字符串值的属性文件。此文件名为 Housing.properties。要导入此文件，执行以下操作:

1. 在 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图中展开以下目录:
   Stores > Web Content> WEB-INF >classes > *FashionFlow_name* 目录。
2. 用鼠标右键单击 *FashionFlow_name* 目录并选择**导入**。
   “导入”向导打开。
3. 从**选择导入源**列表中，选择 **Zip 文件**并单击**下一步**。
4. 单击**浏览**（在 **Zip 文件**字段旁）并浏览至样本代码。此文件的位置如下:
   yourDirectory\WC_SAMPLE_55.zip
   其中 *yourDirectory* 是您将数据包下载至的目录。
5. 单击**全部不选**，然后展开目录并选择要导入的以下文件:
   - Housing.properties
6. 在**文件夹**字段中，`Stores/Web Content/WEB-INF/classes/`*`FashionFlow_name`* 文件夹已指定。保留该值。
7. 单击**完成**。

测试已修改的代码

下一步，您必须通过执行以下操作来测试修改后的注册信息:

1. 切换至“服务器”视图。
2. 用鼠标右键单击 **WebSphereCommerceServer** 测试服务器并选择**启动**。
3. 用鼠标右键单击 Stores\Web Content*FashionFlow_name* 目录下的 **index.jsp** 并选择**在服务器上运行**。
4. 当商店的主页打开时，单击**注册**。
5. 再次单击**注册**来创建新用户。
6. 您将看到已修改的注册页面，现在已具有居住情况调查，如以下屏幕快照所示:

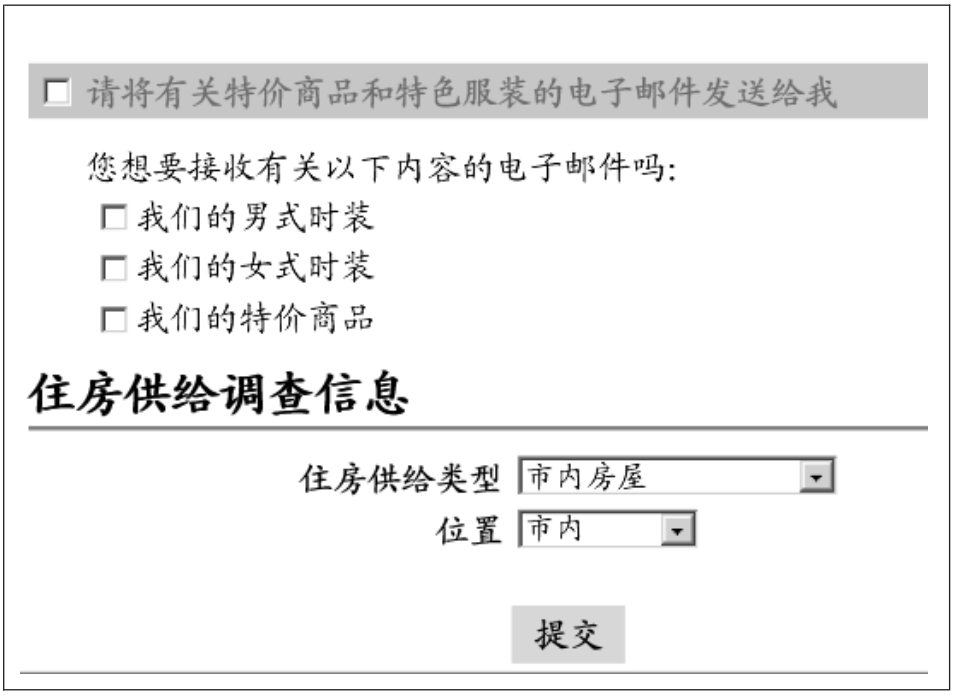

图 53.

7. 为新用户输入相应信息，并单击**提交**。
8. 在信息提交后，单击**更改个人信息**来验证已捕获居住情况信息。您将看到一个屏幕，显示您的调查响应的摘要，如下:

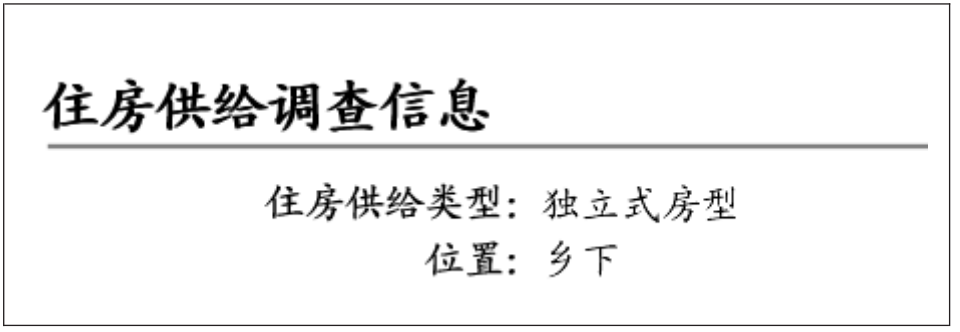

图 54.

部署居住情况调查逻辑

本部分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“时尚潮流”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。

有许多不同类型的有用资源，它们必须部署到目标 WebSphere Commerce Server。它们包括:

- 命令逻辑
- 已修改的企业 bean 逻辑
- JSP 模板
- 属性文件
- 数据库更新包括模式更新（新表）以及命令注册表更新

本节描述如何以*递增*方式将所有这些有用资源部署到目标 WebSphere Commerce Server。

创建命令 JAR 文件

本节描述如何创建包含新 MyPostUserRegistrationAddCmdImpl 逻辑的 JAR 文件。

要创建此 JAR 文件,请在开发机上执行以下步骤:

1. 在本地文件系统上创建名为 \ExportTemp4 的目录。
2. 打开 WebSphere Studio Application Developer 并切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
3. 用鼠标右键单击 **WebSphereCommerceServerExtensionsLogic** 项目并选择 **导出**。
   “导出”向导打开。
4. 在“导出”向导中，请执行以下操作:
   a. 选择 **JAR 文件**并单击**下一步**.
   b. **选择要导出的资源**左下窗格是以项目的名称预先填充的。保留此值不变。
   c. 在右窗格中，确保仅选择了以下资源:
      - .classpath
      - .project
      - .serverPreference

d. 确保已选择**导出已生成的类文件和资源**。

e. 不要选择**导出 Java 源文件和资源**。

f. 在**选择导出目的地**字段中，输入要使用的全限定 JAR 文件名。在此例中，输入 *drive*:\ExportTemp4\WebSphereCommerceServerExtensionsLogic.jar。请注意，JAR 文件名必须是 WebSphereCommerceServerExtensionsLogic.jar。

g. 单击**完成**。

创建 EJB JAR 文件

要创建 EJB JAR 文件，请执行以下操作:

1. 打开 WebSphere Studio Application Developer 并切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
2. 展开 **Member-MemberManagementData** 项目。
3. 双击 **EJB 部署描述符**。
4. 在选择“概述”选项卡的情况下，滚动至窗格底部以找到 **WebSphere 绑定**部分。
5. 在**数据源 JNDI 名称**字段，输入目标 WebSphere Commerce Server 的数据源 JNDI 名称。以下是值的示例:

   DB2 `jdbc/WebSphere Commerce DB2 DataSource demo`
   其中目标 WebSphere Commerce Server 使用 DB2 数据库，WebSphere Commerce 实例名称是“demo”

   Oracle `jdbc/WebSphere Commerce Oracle DataSource demo`
   其中目标 WebSphere Commerce Server 使用 Oracle 数据库，WebSphere Commerce 实例名称是“demo”。
6. 保存部署描述符更改（Ctrl+S）。
7. 在 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图中，用鼠标右键单击 **Member-MemberManagementData** 项目，并选择**导出**。“导出”向导打开。
8. 在“导出”向导中，请执行以下操作:

   a. 选择 **EJB JAR 文件**并单击**下一步**。

   b. Business Professional **您想要导出什么资源?** 的值是以 EJB 项目的名称预填充的。

   Express EJB 项目名称是预填充的。
   保留该值不变。

c. Business Professional 在**您要将资源导出到哪里?** 字段中，输入要使用的全限定 JAR 文件名。

Express 对于目的地，输入要使用的全限定 JAR 文件名。在这种情况下，输入 *drive*:\ExportTemp4\Member-MemberManagementData.jar。

d. 单击**完成**。

9. JAR 文件创建后，撤销步骤 5 中所作的对本地部署描述符的更改，用以恢复本地测试服务器所必需的设置。

导出商店有用资源

要导出已修改的 JSP 模板和新属性文件，请执行以下操作:

1. 打开 WebSphere Studio Application Developer 并切换到 Business Professional “J2EE 导航器”视图 Express “项目导航器”视图。
2. 展开 **Stores** 文件夹。
3. 用鼠标右键单击 **Web Content** 文件夹并选择**导出**。“导出”向导打开。
4. 在“导出”向导中，请执行以下操作:
   a. 选择**文件系统**并单击**下一步**。
   b. 选择要部署的以下资源:
      - Web Content*FashionFlow_name*\UserArea\AccountSection\RegistrationSubsection\UserRegistrationAddForm.jsp
      - Web Content*FashionFlow_name*\UserArea\AccountSection\RegistrationSubsection\UserRegistrationAddFormInclude.jsp
      - Web Content*FashionFlow_name*\UserArea\AccountSection\RegistrationSubsection\UserRegistrationUpdateForm.jsp
      - Web Content*FashionFlow_name*\UserArea\AccountSection\RegistrationSubsection\UserRegistrationUpdateFormInclude.jsp
      - Web Content\WEB-INF\lib\jstl.jar
      - Web Content\WEB-INF\lib\standard.jar
      - Web Content\WEB-INF\classes*FashionFlow_name*\Housing.properties

   c. 选择**为选定文件创建目录结构**。
   d. 在“目录”字段中输入放置这些资源的临时目录。例如，输入 C:\ExportTemp4
   e. 单击**完成**。

将有用资源传送到目标 WebSphere Commerce Server

在本步骤中，您将在目标 WebSphere Commerce Server 创建一个临时目录并把您的居住情况调查有用资源复制到此目录中。在后面的步骤中，您将把不同类型的代码置于 WebSphere Commerce 应用程序中的合适位置。

要把文件从开发机复制到目标 WebSphere Commerce Server，请执行以下操作:

1. 在目标 WebSphere Commerce Server 上，创建一个称为 \ImportTemp4 的临时目录。
2. 确定如何从一台计算机向另一台计算机复制文件。您可以通过将目标 WebSphere Commerce Server 上的驱动器映射到开发机或是使用 FTP 应用程序（如果已配置）来实现这一目的。
3. 从开发机器上将 \ExportTemp4 的内容复制到目标 WebSphere Commerce Server 上的 \ImportTemp4。

停止目标 WebSphere Commerce Server

在开始部署步骤之前，应通过在命令行发出 stopServer 命令来停止目标 WebSphere Commerce Server。有关该命令的详细信息，请参考Business Professional *《WebSphere Commerce Studio 安装指南》*或 Express *《WebSphere Commerce - Express Developer Edition 安装指南》*。

更新目标 WebSphere Commerce Server 上的数据库

创建 XHOUSING 表

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 > 命令行工具 > 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 developmentDB user dbuser using dbpassword;
create table XHOUSING (MEMBERID bigint not null,
    HOUSINGTYPE integer, LOCATION integer,
   constraint p_xhousing primary key (MEMBERID),
   constraint f_xhousing foreign key (MEMBERID)
    references USERS (USERS_ID) on delete cascade);
insert into XHOUSING (MEMBERID) (select USERS_ID from USERS);
```

其中

- *developmentDB* 是开发数据库的名称
- *dbuser* 是数据库用户
- *dbpassword* 是数据库用户的密码

单击“执行”图标。
您应该看到一条消息，指示 SQL 语句已成功完成。

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 XHOUSING (MEMBERID number not null,
    HOUSINGTYPE integer, LOCATION integer,
   constraint p_xhousing primary key (MEMBERID),
   constraint f_xhousing foreign key (MEMBERID)
    references USERS (USERS_ID) on delete cascade);
insert into XHOUSING (MEMBERID) (select USERS_ID from USERS);
```

并按 Enter 键运行 SQL 语句。XHOUSING 表已创建。

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。

注册任务命令

要修改命令注册表，请执行以下操作:

1. DB2 如果正在使用 DB2 数据库，请执行以下操作注册 MyPostUserRegistrationAddCmdImpl:
   a. 打开 DB2 控制中心（**开始 > 程序 > IBM DB2 > 控制中心**）。
   b. 从**工具**菜单中，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 developmentDB user dbuser using dbpassword;
insert into CMDREG values (FashionFlow_storeent_Id,
'com.ibm.commerce.usermanagement.commands.PostUserRegistrationAddCmd'
'Description',
'com.ibm.commerce.sample.commands.MyPostUserRegistrationAddCmdImpl',
null, null, 'Local');
```

其中

- *developmentDB* 是开发数据库的名称
- *dbuser* 是数据库用户
- *dbpassword* 是数据库用户的密码
- *FashionFlow_storeent_Id* 是商店的唯一商店实体标识。

单击**执行**图标。

2. Oracle 如果正在使用 Oracle 数据库，请执行以下操作注册 MyPostUserRegistrationAddCmdImpl:
   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 SQL 语句:

```
insert into CMDREG values (FashionFlow_storeent_Id,
'com.ibm.commerce.usermanagement.commands.PostUserRegistrationAddCmd'
'Description',
'com.ibm.commerce.sample.commands.MyPostUserRegistrationAddCmdImpl',
null, null, 'Local');
```

   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。

更新目标 WebSphere Commerce Server 上的商店有用资源

在本步骤中，您将用已修改的商店有用资源来更新商店，如下:

1. 备份 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear\ Stores.war 目录（其中 *cellName* 通常是机器的主机名）。
2. 浏览至 :\ImportTemp4\Stores\Web Content 目录。
3. 将 *FashionFlow_name* 和 WEB-INF 文件夹复制到以下目录: *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear\Stores.war

其中 *instanceName* 是 WebSphere Commerce 实例的名称。

更新目标 WebSphere Commerce Server 上的命令 JAR 文件

在本步骤中您将更新目标 WebSphere Commerce Server 来使用新命令 JAR 文件，如下:

1. 您应制作现有 JAR 文件的备份副本，如下:
   a. 浏览至 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear 目录。
   b. 制作 WebSphereCommerceServerExtensionsLogic.jar 文件的副本并将它保存在备份位置。
2. 将新的 WebSphereCommerceServerExtensionsLogic.jar 文件从 \ImportTemp4 目录复制到 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear 目录

其中 *instanceName* 是 WebSphere Commerce 实例的名称。

更新目标 WebSphere Commerce Server 上的 EJB JAR 文件

在本步骤中您将更新目标 WebSphere Commerce Server 来使用新 EJB JAR 文件，如下:

1. 您应制作现有 JAR 文件的备份副本，如下:
   a. 浏览至 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear 目录。
   b. 制作 Member-MemberManagementData.jar 文件的副本并将它保存在备份位置。
2. 将新的 Member-MemberManagementData.jar 文件从 \ImportTemp4 目录复制到 *WAS_installdir*\installedApps*cellName*\WC_*instanceName*.ear 目录
3. 下一步，您必须修改 EJB 部署描述符信息，如下:
   a. 定位这一 WebSphere Application Server 单元的部署资源库（META-INF 目录）。这通常采取以下格式:
   WAS_installdir\config\cells*cellName*\applications\WC_*instance_name*.ear\deployments\WC_*instance_name**EJBModuleName*.jar\META-INF。
   以下是它的一个特定示例:
   D:\WebSphere\AppServer\config\cells\myCell\applications\WC_demo.ear\deployments\WC_demo\Member-MemberManagementData.jar\META-INF
   其中
   - myCell 是 WebSphere Application Server 单元的名称
   - demo 是 WebSphere Commerce 实例的名称
   - Member-MemberManagementData 是定制的 EJB 模块的名称

   b. 该目录包含以下文件:
   - ejb-jar.xml
   - ibm-ejb-access-bean.xmi
   - ibm-ejb-jar-bnd.xmi

- ibm-ejb-jar-ext.xmi
- MANIFEST.MF

备份所有这些文件。

c. 使用工具打开新 Member-MemberManagementData.jar 文件并查看其内容。

d. 将 meta-inf 目录的内容从此 Member-MemberManagementData.jar 文件解压缩到步骤 3a 的目录中。

4. 在命令行中使用 WebSphere Application Server startServer 命令，重新启动 WebSphere Commerce 实例。

验证目标 WebSphere Commerce Server 上的居住情况调查逻辑

在此步骤中，通过执行以下操作，验证已成功部署到目标 WebSphere Commerce Server 的居住情况调查逻辑:

1. 打开 Web 浏览器并输入 URL，以启动您的基于 “时尚潮流” 样本商店的商店。
2. 当商店的主页打开时，单击**注册**。
3. 再次单击**注册**来创建新用户。
4. 您将看到已修改的注册页面，现在已具有居住情况调查，如以下屏幕快照所示:

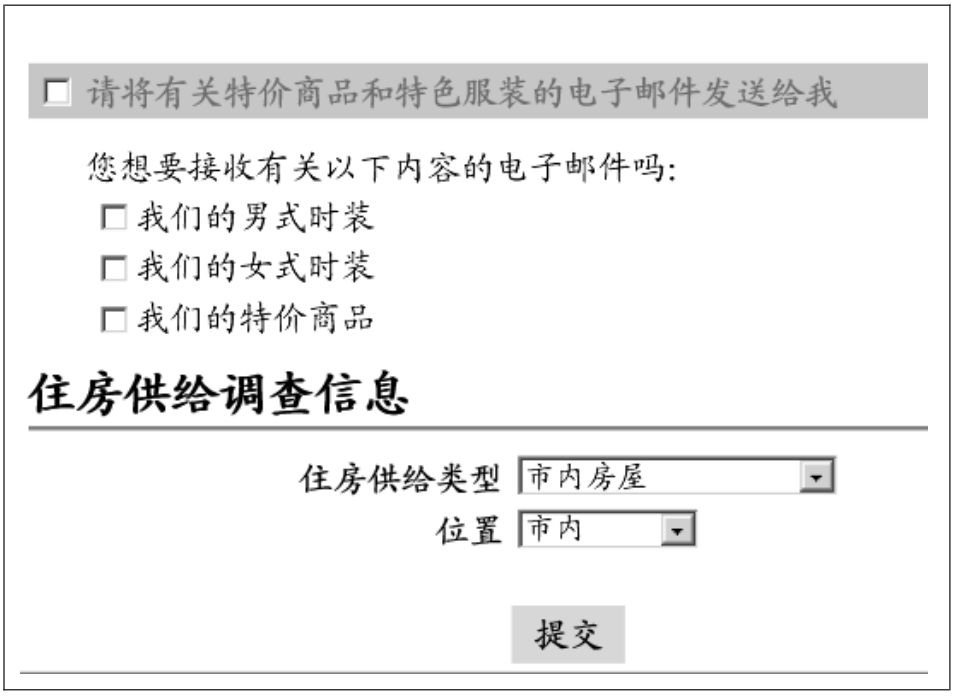

图 55.

5. 为新用户输入相应信息，并单击**提交**。
6. 在信息提交后，单击**更改个人信息**来验证已捕获居住情况信息。您将看到一个屏幕，显示您的调查响应的摘要，如下:

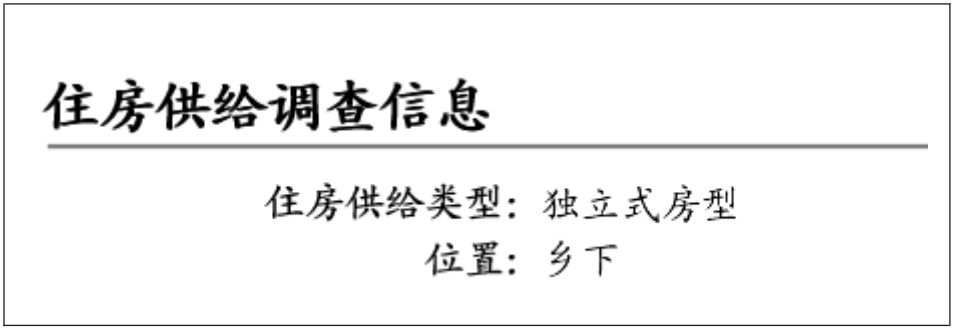

图 56.

第 5 部分 附录

附录 A. 配置 WebSphere Commerce 开发环境中的 WebSphere Commerce 组件跟踪

此附录描述当在 WebSphere Commerce 开发环境中运行时如何为各种 WebSphere Commerce 组件启用跟踪。要启用组件跟踪，请执行以下操作:

1. 如果需要，请打开 WebSphere Commerce 开发环境（**开始 > 程序 > IBM WebSphere Commerce 开发环境 > WebSphere Commerce 开发环境**）。
2. 切换至“服务器”视图。
3. 在“服务器”视图中，用鼠标右键单击 **WebSphereCommerceServer** 并选择**停止**（如果服务器当前正在运行的话）。
4. 在服务器配置视图，展开**服务器配置**文件夹。
5. 双击 **WebSphereCommerceServer**。
   WebSphereCommerceServer 编辑器打开。
6. 选择“跟踪”选项卡。
7. 选择**启用跟踪**。
8. 在**跟踪字符串**文本框中，指定应启用跟踪的组件。使用 WebSphere JRas 扩展跟踪记录器标识值，该值后跟“=all=enabled”。关于这些值的完整列表，请参阅《*WebSphere Commerce 管理指南*》的“配置”主题。多个组件间应当用冒号（:）分隔。例如，要为 SERVER 和 RAS 组件启用跟踪，如下指定跟踪字符串:

   ```
   com.ibm.websphere.commerce.WC_SERVER=all=enabled:
      com.ibm.websphere.commerce.WC_RAS=all=enabled
   ```

   请注意，断行只是出于显示目的。
9. 保存更改（Ctrl+S）。

输出文件

缺省情况下，输出日志文件名为 activity.log。此文件位于以下目录中:
workspace_dir\.metadata\.plugin\com.ibm.etools.server.core\tmp0\logs

由于 activity.log 文件是一个二进制文件，因此使用 Log Analyzer 来阅读此文件。一旦您启用组件跟踪，WebSphere JRas 也会以纯文本格式将日志条目和跟踪条目写入跟踪输出文件。

有关配置 WebSphere Commerce 开发环境中的 Log Analyzer 工具的信息，请参考 Business Professional *《WebSphere Commerce Studio 安装指南》* 或 Express *《WebSphere Commerce - Express Developer Edition 安装指南》*。

此外，消息还会在 WebSphere Studio Application Developer 的控制台视图中显示。

附录 B. 何处可以找到更多信息

可通过各种渠道获取有关 WebSphere Commerce 开发环境系统及其组件的不同格式的更多信息。以下部分指出可用的信息及其访问方式。

WebSphere Commerce 开发环境信息

以下是 WebSphere Commerce 开发环境信息的来源:

- 『WebSphere Commerce 开发环境联机帮助』
- 第 368 页的『WebSphere Commerce Web 站点』
- 第 368 页的『WebSphere Developer Domain』
- 第 368 页的『IBM 红皮书』

WebSphere Commerce 开发环境联机帮助

WebSphere Commerce 开发环境联机信息是在 WebSphere Commerce 开发环境中创建和发布商店的主要信息源。

要查看 WebSphere Commerce 开发环境联机帮助，请执行以下操作:

1. 通过选择**开始 → 程序 → IBM WebSphere Commerce Studio → WebSphere Commerce 开发环境**，启动 WebSphere Commerce 开发环境。
2. 从**帮助**菜单中选择**帮助内容**。

注: 如果您参考『WebSphere Commerce 开发环境联机帮助』中有关多平台的指示信息，请确保您遵循 WebSphere Commerce 开发环境的指示信息。当帮助页面包含有关多平台的信息时，特定于 WebSphere Commerce 开发环境的信息以以下图标指示:

如果未提供特定于 WebSphere Commerce 开发环境的信息，请确保您遵循特定于 Windows 的指示信息，当帮助页面包含有关多平台的指示信息时，用以下图标来指示有关 Windows 的指示信息:

WebSphere Commerce Web 站点

WebSphere Commerce 开发环境产品信息可从 WebSphere Commerce Web 站点获得。关于更多产品信息，请参阅以下 URL:

http://www.ibm.com/software/webservers/commerce/library/

WebSphere Developer Domain

在 WebSphere Developer Domain 的 WebSphere Commerce Zone 中还可以获取关于 WebSphere Commerce 开发环境和 WebSphere Commerce 的其它信息:

http://www.ibm.com/websphere/developer/zones/commerce/

IBM 红皮书

WebSphere Commerce 开发环境和 WebSphere Commerce 信息可从以下 IBM 红皮书™ Web 站点获取:

http://www.ibm.com/redbooks

WebSphere Studio Application Developer 信息

以下是 WebSphere Studio Application Developer 的信息源:

- 『WebSphere Studio Application Developer 联机帮助』
- 『WebSphere Studio Application Developer Web 站点』
- 第 369 页的『WebSphere Developer Domain』
- 第 369 页的『IBM 红皮书』

WebSphere Studio Application Developer 联机帮助

WebSphere Studio Application Developer 联机帮助是关于如何在 WebSphere Studio Application Developer 中执行任务的主要信息源。

要查看 WebSphere Studio Application Developer 联机帮助，请执行以下操作:

1. 通过选择**开始 → 程序 → IBM WebSphere Studio → Application Developer 5.0**，启动 WebSphere Commerce 开发环境。
2. 从**帮助**菜单中选择**帮助内容**。

WebSphere Studio Application Developer Web 站点

WebSphere Studio Application Developer 产品信息可从 WebSphere Studio Application Developer Web 站点获得:

http://www.ibm.com/software/ad/studioappdev/library/

WebSphere Developer Domain

在 WebSphere Developer Domain 的 WebSphere Studio Zone 中的 WebSphere Studio Application Developer 页面上，可以获取关于 WebSphere Studio Application Developer 的其它信息:

`http://www.ibm.com/websphere/developer/zones/studio/appdev/`

IBM 红皮书

WebSphere Studio Application Developer 信息可从以下 IBM 红皮书 Web 站点获取:

`http://www.ibm.com/redbooks`

声明

本信息是为在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 代表咨询。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

本条款不适用英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；这些更改将编入本资料的新版本中。IBM 可以随时对本资料中描述的产品和／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。那些 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:

本信息包含用源语言表示的样本应用程序，这些样本说明不同操作平台上的编程方法。若为是为按照在编写样本程序的操作平台上的应用程序编程接口（API）进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例并未在所有条件下作全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。用户若为是为了按照 IBM 应用程序编程接口开发、使用、经销或分发应用程序，则可以任何形式复制、修改和分发这些样本程序，而无须向 IBM 付费。

凡这些样本程序的每份拷贝或其任何部分或任何衍生品，都必须包括如下版权声明:

如果您正以软拷贝格式查看本信息，图片和彩色图例可能无法显示。

商标和服务标记

IBM 徽标和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
AIX
AS/400
DB2
DB2 Universal Database
@server
IBM
iSeries
WebSphere

Windows 是 Microsoft Corporation 在美国和／或其它国家或地区的商标或注册商标。

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

中国印刷