第3章 Struts基础

学习目的与学习要求

学习目的：深入了解 Struts 2 框架，掌握 Struts 2 框架的工作流程及各个配置文件，Action 类的开发及配置，Model 组件的配置开发和 View 试图组件的配置跳转。

学习要求：熟练应用开发工具搭建和开发基于 Struts 2 框架的应用系统；熟练开发 Action 类及配置调用，正确使用 View 试图组件跳转显示，顺利完成本章节案例开发，并模拟开发其他项目功能。

本章主要内容

本章主要内容包括 MVC 模式概述、MVC 与 Struts 2 的映射、Struts 2 框架的工作流程和配置文件、Action 类的开发和配置、Model 组件和试图组件的开发和配置及 Struts 2 开发步骤。

在 Web 应用开发中，我们经常使用 Struts 框架解决用户接口(UI)层，及其与后端应用层之间的交互。

Struts 是 Craig McClanahan 在 2001 年发布的 Web 框架。经过多年的验证，Struts 越来越稳定和成熟。目前，Struts 推出 2.0 版本的全新框架，它在另一个 MVC 框架 Webwork 的优良基础设计之上进行了一次巨大的升级。

Struts 框架是 MVC 设计模式的一个具体实现，下面介绍 MVC 模式。

3.1 MVC 模式概述

在计算机软件工程领域常常提到设计模式(Design Pattern)。那么，什么是模式(Pattern)呢？一般来说，模式是指一种从一个多次出现的问题背景中抽象出的解决问题的固定方案，而这个问题背景不应该是绝对的，或者说是不固定

的。很多时候，看来不相关的问题，会有相同的问题背景，从而需要应用相同的模式解决。

模式的概念最开始的时候出现在城市建筑领域中。后来，这个概念逐渐被计算机科学所采纳，并在一本广为接受的经典书籍的推动下而流行起来。这本书就是 *Design Patterns*：*Elements of Reusable Object-Oriented Software*（设计模式：可复用面向对象软件元素），由 4 位软件大师合写而成（很多时候，直接用 GoF 指这 4 位作者，GoF 的意思是 Gangs of Four，即四人帮）。

设计模式指的是在软件的建模和设计过程中运用到的模式。设计模式中的很多种方法其实很早就出现了，并且应用得比较多。但是，直到 GoF 的书出来之前，并没有一种统一的认识。或者说，那时并没有对模式形成一个概念，这些方法还仅处在经验阶段，并没有被系统地整理，形成一种理论。

每个设计模式都系统地命名、解释和评价了面向对象系统中一个重要的和重复出现的设计。这样，只要清楚这些设计模式，就可以完全或者很大程度上吸收那些蕴含在模式中的宝贵经验，对面向对象的系统能够有更完善的了解。更重要的是，这些模式都可以直接用来指导面向对象系统中至关重要的对象建模问题。如果有相同的问题背景，直接套用这些模式就可以了，这可以省去很多工作量。

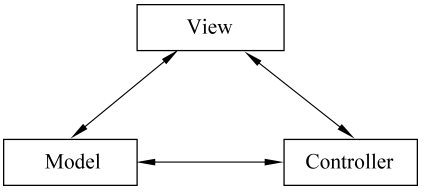

图 3-1 MVC 架构图

MVC 模式就是一种很常见的设计模式。所谓的 MVC 模式，即模型-视图-控制器（Model-View-Controller）模式。MVC 架构图如图 3-1 所示。

1. Model 端

在 MVC 中，模型是执行某些任务的代码，而这部分代码并没有任何逻辑决定用户端的表示方法。Model 只有纯粹的功能性接口，也就是一系列的公共方法，通过这些公共方法，便可以取得模型端的所有功能。

2. View 端

在 MVC 模式里，一个 Model 可以有几个 View 端，而实际上多个 View 端是使用 MVC 的原始动机。使用 MVC 模式可以允许存在多于一个的 View 端，并可以在需要的时候动态注册所需要的 View。

3. Controller 端

MVC 模式的视图端是与 MVC 的控制器结合使用的。当用户端与相应的视图发生交互时，用户可以通过视窗更新模型的状态，而这种更新是通过控制器端进行的。控制器端通过调用模型端的方法更改其状态值。与此同时，控制器端会通知所有注册了的视图刷新用户界面。

那么，使用 MVC 模式有哪些优点呢？MVC 通过以下 3 种方式消除与用户接口和面向对象的设计有关的绝大部分困难：

（1）控制器通过一个状态机跟踪和处理面向操作的用户事件。这允许控制器在必要时创建和破坏来自模型的对象，并且将面向操作的拓扑结构与面向对象的设计隔离开。这个隔离有助于防止面向对象的设计走向歧途。

（2）MVC 将用户接口与面向对象的模型分开。这允许同样的模型不用修改，就可使用许多不同的界面显示方式。除此之外，如果模型更新由控制器完成，那么界面就可以跨应用再使用。

（3）MVC 允许应用的用户接口进行大的变化，而不影响模型。每个用户接口的变化只需

要对控制器进行修改，但是控制器包含很少的实际行为，它是很容易修改的。

面向对象的设计人员在将一个可视化接口添加到一个面向对象的设计中时必须非常小心，因为可视化接口的面向操作的拓扑结构可以大大增加设计的复杂性。

MVC 设计允许一个开发者将一个好的面向对象的设计与用户接口隔离开，允许在同样的模型中使用多个接口，并且允许在实现阶段对接口做大的修改，而不需要对相应的模型进行修改。

3.2 MVC 与 Struts 2 的映射

Struts 的体系结构实现了 Model-View-Controller 设计模式的概念，它将这些概念映射到 Web 应用程序的组件和概念中。

1. 控制器层(Controller)

与 Struts 1 使用 ActionServlet 作为控制器不同，Struts 2 使用了 filter 技术，FilterDispatcher 是 Struts 框架的核心控制器。该控制器负责拦截和过滤所有的用户请求。如果用户请求以 action 结尾，该请求将被转入 Struts 框架进行处理。Struts 框架获得 *.action 请求后，将根据 *.action 请求的名称部分决定调用哪个业务控制 action 类。例如，对于 test.action 请求，调用名为 test 的 action 处理该请求。

Struts 应用中的 action 都被定义在 struts.xml 文件中，在该文件中配置 action 时，主要定义了该 action 的 name 属性和 class 属性，其中 name 属性决定了该 action 处理哪个用户请求，而 class 属性决定了 action 的实现类。例如，＜action name＝"registAction" class＝"com.ascent.action.RegistAction"＞用于处理用户请求的 action 实例，并没有与 Servlet API 耦合，所以无法直接处理用户请求。为此，Struts 框架提供了系列拦截器，该系列拦截器负责将 HttpServletRequest 请求中的请求参数解析出来，传入 Action 中，并回调 Action 的 execute() 方法处理用户请求。关于拦截器的概念，稍后会详细讲解。

2. 显示层(View)

Struts 2 框架改变了 Struts 1 只能使用 JSP 作为视图技术的现状(当然，可以少量支持 Velocity 等技术)，它允许使用其他的视图技术(如 FreeMarker、Velocity 等)作为显示层。

当 Struts 2 的控制器调用业务逻辑组件处理完用户请求后，会返回一个字符串，该字符串代表逻辑视图，它并未与任何视图技术关联。

当在 struts.xml 文件中配置 action 时，还要为 action 元素指定系列 result 子元素，每个 result 子元素定义上述逻辑视图和物理视图之间的映射。一般情况下使用 JSP 技术作为视图，故配置 result 子元素时没有 type 属性，默认使用 JSP 作为视图资源，如＜result name＝"error"＞/product/register.jsp＜/result＞。

如果需要在 Struts 2 中使用其他视图技术，则在配置 result 子元素时指定相应的 type 属性即可。例如，type 属性指定值可以是 Velocity。

Struts 显示层包括一个便于创建用户界面的定制标记库。这些标记的使用将在后面讨论。另外我们还会讲解国际化支持、表达式语言等内容。

3. 模型层(Model)

模型层指的是后端业务逻辑处理，它会被 action 调用处理用户请求。当控制器需要获得业务逻辑组件实例时，通常并不会直接获取业务逻辑组件实例，而是通过工厂模式获得业务逻

辑组件的实例;或者利用其他 IoC 容器(如 Spring 容器)管理业务逻辑组件的实例。后面会详细介绍这些技术。

基于 MVC 的系统中的业务逻辑组件还可以细分为两个概念:系统的内部状态以及能够改变状态的行为。可以把状态信息当作名词(事物),把行为当作动词(事物状态的改变),它们使用 JavaBean、EJB 或 Web Service 实现。

这一体系结构中的每个主要组件将在下面详细讨论。在此之前先了解一下 Struts 2 的整体工作流程。

3.3 Struts 2 框架的工作流程和配置文件

3.3.1 Struts 2 框架的工作流程

Struts 2 的工作流程是 WebWork 的升级,而不是 Struts 1 的升级,如图 3-2 所示。

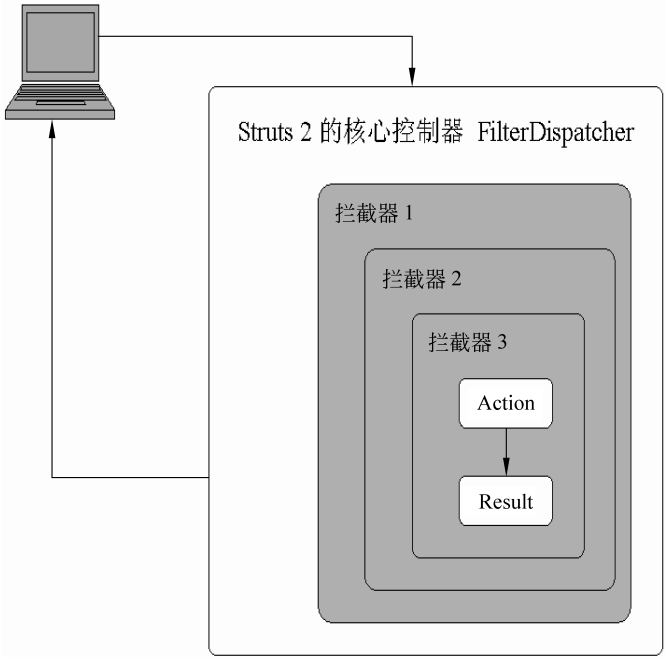

图 3-2 Struts 2 的体系概图

Struts 2 的工作流程如下:

(1) 浏览器发送请求,如请求/regist.action、/reports/myreport.pdf 等。

(2) 核心控制器 FilterDispatcher 根据请求决定调用合适的 Action。

(3) WebWork 的拦截器链自动对请求应用通用功能,如验证、工作流或文件上传等功能。

(4) 回调 Action 的 execute()方法,该方法先获取用户请求参数,然后执行某种业务操作,既可以将数据保存到数据库,也可以从数据库中检索信息。实际上,因为 Action 只是一个控制器,所以它会调用业务逻辑组件(Model)处理用户的请求。

(5) Action 的 execute()方法处理结果信息将被输出到浏览器中,可以是 HTML 页面、图像,也可以是 PDF 文档或者其他文档。Struts 2 支持的视图技术非常多,既支持 JSP,也支持

Velocity、FreeMarker 等模板技术。

要想更详细地掌握 Struts 的核心技术和流程，就要先理解 Struts 的配置文件。

3.3.2 Struts 的配置文件

Struts 2 的配置文件有两份，包括配置 Action 的 struts.xml 文件和配置 Struts 2 全局属性的 struts.properties 文件。接下来分别对它们进行讨论。

1. struts.xml 配置文件

首先了解一下 Struts 2 框架自带的一些配置文件。在 Struts 2 的 jar 包中，会看到有一个 struts-default.xml 文件。这个 struts-default.xml 文件是 Struts 2 框架的默认配置文件，Struts 2 框架每次都会自动加载该文件。struts-default.xml 文件定义了一个名字为 struts-default 的包空间，该包空间里定义了 Struts 2 内建的 Result 类型，配置了大量的核心组件，以及 Struts 2 内建的系列拦截器、由不同拦截器组成的拦截器栈和默认的拦截器引用等。

另外，Struts 2 框架允许以一种“可插拔”的方式安装插件，它们都提供了一个类似 Struts2-xxx-plugin.jar 的文件，例如后面要用到的 Spring 插件，它提供了 Struts2-spring-plugin2.06.jar 文件，只要将该文件放在 Web 应用的 WEB-INF/lib 路径下，Struts 2 框架将自动加载该框架。

对于绝大部分 Struts 2 应用，无须重新定义上面这些配置文件。我们所要关注和管理的是下面的 struts.xml 配置文件。

Struts 框架的核心配置文件就是 struts.xml 配置文件。默认情况下，Struts 2 框架将自动加载放在 WEB-INF/classes 路径下的 struts.xml 文件。该文件主要负责管理应用中的 Action 映射、该 Action 包含的 result 定义等，以及一些其他相关配置。

以下是项目中 struts.xml 的实例。

```
<?xml version="1.0" encoding="GBK"?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="ascenttech" extends="json-default">
        <action name="registCheckAction" class="com.ascent.action.RegistCheckAction">
            <result type="json"></result>
        </action>
        <action name="registAction" class="com.ascent.action.RegistAction">
            <result>/product/regist_succ.jsp</result>
            <result name="error">/product/register.jsp</result>
        </action>
        <action name="usrLoginAction" class="com.ascent.action.UsrLoginAction">
            <result>/index.html</result>
            <result name="success_1">/product/products.jsp</result>
            <result name="success_2">/product/products.jsp</result>
            <result name="success_3">/product/products_showusers.jsp</result>
            <result name="error">/product/products.jsp</result>
            <result name="input">/product/products.jsp</result>
```

```
        </action>
        <action name="*usrManagerAction" class="com.ascent.action.UsrManagerAction"
method="{1}">
            <result></result>
            <result name="show_users">/product/products_showusers.jsp</result>
            <result name="updateuser">/product/updateuser.jsp</result>
            <result name="updateuser_error">/product/updateuser.jsp</result>
            <result name="changesuperuser_error">/product/changesuperuser.jsp</
result>
            <result name="error">/product/products.jsp</result>
            <result name="admin_order_user">/product/admin_orderuser.jsp</result>
        </action>
           < action  name =" productIdCheckAction "  class =" com. ascent. action.
ProductIdCheckAction">
            <result type="json"></result>
        </action>
          < action  name =" *  ProductManagerAction "  class =" com. ascent. action.
ProductManagerAction" method="{1}">
            <!-- 配置 fileUpload 拦截器 -->
            <interceptor-ref name="fileUpload">
                <!-- 设置上传文件类型 -->
                <param name="allowedTypes">image/bmp,image/png,image/jpg,image/
gif,application/vnd.ms-excel</param>
                <!-- 设置上传文件大小 -->
                <param name="maximumSize">200000</param>
            </interceptor-ref>
            <!-- 必须显示配置引用 struts 默认的拦截器栈:defaultStack -->
            <interceptor-ref name="defaultStack"></interceptor-ref>
            <!-- 设置上传路径 -->
            <param name="savePath">/upload</param>
            <result name="adminproductsshow">/product/admin_products_show.jsp</
result>
            <result name="saveOnesuccess">/product/admin_products_show.jsp</result>
            <!-- 必须设置 input 逻辑视图,拦截器默认出错返回 input -->
            <result name="input">/product/upload_error.jsp</result>
            <result name="updateProduct">/product/update_products_admin.jsp</
result>
            <result name="guestproductsshow">/product/products_show.jsp</result>
            <result name="searchproductshow">/product/products_search_show.jsp</
result>
            <result name="userproducts">/product/userproducts_show.jsp</result>
            <result name="index">/index.jsp</result>
            <result name="productdetail">/product/productdetail.jsp</result>
        </action>
        <action name="*CartManagerAction" class="com.ascent.action.CartManagerAction"
```

```
method="{1}">
            <result name="cartshow">/product/cartshow.jsp</result>
        </action>
        <action name="*OrdersManagerAction" class="com.ascent.action
.OrdersManagerAction" method="{1}">
        <result name="checkoutsucc">/product/checkoutsucc.jsp</result>
        <result name="ordershow">/product/ordershow.jsp</result>
        <result name="adminordershow">/product/admin_ordershow.jsp</result>
        </action>
        <action name="*OrderitemManagerAction" class="com.ascent.action
.OrderitemManagerAction" method="{1}">
        <result name="orderitemshow">/product/orderitem_show.jsp</result>
        </action>
        <action name="*MailManagerAction" class="com.ascent.action
.MailManagerAction" method="{1}">
        <result name="addmailsucc">/product/mailmanager_succ.jsp</result>
        <result name="addmailfail">/product/mailmanager_fail.jsp</result>
        </action>
        <action name="userProductAddAction" class="com.ascent.action
.UserProductAddAction">
            <result type="json"></result>
        </action>
        <action name="clearSession" class="com.ascent.action.ClearSessionAction">
            <result>/index.jsp</result>
        </action>
    </package>
    <!-- 此处用 constant 元素定义常量 -->
    <constant name="struts.devMode" value="true"/>
    <!-- 定义资源文件的位置和类型 -->
    <constant name="struts.custom.i18n.resources" value="properties/myMessages"/>
    <!-- 设置应用使用的解析码 -->
    <constant name="struts.i18n.encoding" value="GBK"/>
    <!-- 设置应用使用的上传解析器类型 -->
    <constant name="struts.multipart.parser" value="jakarta"/>
    <!-- 指定使用按 type 的自动装配策略 -->
    <constant name="struts.objectFactory.spring.autoWire" value="name"/>
</struts>
```

接下来具体看看 struts.xml 的主要内容。

1）包配置

Struts 2 框架中的核心组件是 Action、拦截器等。Struts 2 框架使用包（package）管理 Action 和拦截器等，package 是多个 Action、多个拦截器、多个拦截器引用的集合。

在 struts.xml 文件中，package 元素用于定义包配置，每个 package 元素定义了一个包配置。定义 package 元素时可以指定如表 3-1 所示的 4 个属性。

表 3-1 package 元素属性

属　性	描　述
name	这是一个必填属性，它指定该包的名字，该名字是该包被其他包引用的 key
extends	该属性是一个可选属性，它指定该包继承其他包。继承其他包，可以继承其他包中的 Action 定义、拦截器定义等
namespace	该属性是一个可选属性，它定义该包的命名空间
abstract	该属性是一个可选属性，它指定该包是否是一个抽象包。抽象包中不能包含 Action 定义

这里要特别了解命名空间的概念。考虑到在同一个 Web 应用中可能有同名的 Action，Struts 2 以命名空间的方式管理 Action。同一个命名空间里不能有同名的 Action，不同的命名空间里可以有同名的 Action。Struts 2 的命名空间的作用等同于 Struts 1 里模块的作用，它允许以模块化的方式组织 Action。

Struts 2 不支持单独配置命名空间，而是通过为包指定 namespace 属性来为包里所有的 Action 指定共同的命名空间。

看下面的配置文件代码。

```
<package name="ascenttech" extends="struts-default" namespace="/ascentns">
    <action name="getUsers" class="com.ascent.action.GetUsersAction">
        <result name="login">/login.jsp</result>
        <result name="success">/listUser.jsp</result>
    </action>
</package>
```

当某个包指定命名空间后，该包下所有的 Action 处理 URL 的应该是命名空间＋Action 名。以上面名为 ascentns 的包为例，该包下包含了名为 getUsers 的 Action，则该 Action 处理的 URL 为

```
http://locahost:8080/ascentsdemo/ascentns/getUsers.action
```

如果某个包没有指定 namespace 属性，那么该包使用默认的命名空间，默认的命名空间总是“”。默认命名空间里的 Action 可以处理任何模块下的 Action 请求。也就是说，如果存在 URL 为/ascentns/test.action 的请求，并且/ascentns 的命名空间下没有名为 test 的 Action，则默认命名空间下名为 test 的 Action 也会处理用户请求。

除此之外，Struts 2 还可以指定根命名空间，即通过设置某个包的 namespace＝“/”指定根命名空间。如果请求为/test.action，系统会在根命名空间(“/”)中查找名为 test 的 action，如果在根命名空间中找到了名为 test 的 action，则由该 action 处理用户请求。否则，系统将转入默认命名空间中查找名为 test 的 action，如果默认的命名空间里有名为 test 的 action，则由该 action 处理用户请求；如果两个命名空间里都找不到名为 test 的 Action，则系统出现错误。

2) Action 配置

struts.xml 中最重要的是关于 Action 的配置。Action 是 Struts 2 的基本“程序单位”。

(1) 基本配置。

配置 Action 时，需要指定该 Action 的实现类，并定义 Action 处理结果与视图资源之间的映射关系。

下面是 struts.xml 配置文件的示例：

```
<package name="ascenttech" extends="json-default">
    …
        <action name="usrLoginAction" class="com.ascent.action.UsrLoginAction">
            <result>/index.html</result>
            <result name="success_1">/product/products.jsp</result>
            <result name="success_2">/product/products.jsp</result>
            <result name="success_3">/product/products_showusers.jsp</result>
            <result name="error">/product/products.jsp</result>
            <result name="input">/product/products.jsp</result>
        </action>
    …
</package>
```

前面提到，Struts 2 使用包组织 Action，因此，Action 定义是放在包定义下完成的，定义 Action 通过使用 package 下的 Action 子元素完成。定义 Action 时，至少需要指定它的 name 属性，该 name 属性既是该 Action 的名字，也是它需要处理的 URL 的一部分。

注意：Struts 2 的 Action 名字就是它所处理的 URL 的前半部分。与 Struts 1 不同，Struts 1 的 Action 配置中的 name 属性指定的是该 Action 关联的 ActionForm，而 path 属性才是该 Action 处理的 URL。Struts 2 去除了这些易混淆的地方，Action 的 name 属性等同于 Struts 1 中 Action 的 path 属性。

除此之外，通常还需要为 Action 元素指定一个 class 属性，它指定了该 Action 的实现类，如＜action name＝"usrLoginAction" class＝"com.ascent.action.UsrLoginAction"＞。

前面提到过，Action 只是一个业务控制器，它在处理完用户请求后，需要将指定的视图资源呈现给用户。因此，配置 Action 时，应该配置逻辑视图和物理视图资源之间的映射。这是通过＜result…/＞元素定义的，每个＜result…/＞元素定义逻辑视图和物理视图之间的一次映射。关于 result 的配置，后面有更详细的讲解。

另外，还可以为 action 元素指定 method 属性。Struts 框架允许一个表单元素里包含多个按钮，分别提交给不同的处理逻辑。Struts 2 提供了一种处理方法，即将一个 Action 处理类定义成多个逻辑 Action。如果在配置＜action…/＞元素时指定 action 的 method 属性，则可以让 Action 类调用指定方法，而不是用 execute()方法处理用户请求。

```
<action name="login" class="com.ascent.action.LoginAction" method="login" />
  …
</action>
<action name="regist" class=" com.ascent.action .LoginAction" />
  …
</action>
```

上面定义了 login 和 regist 两个逻辑 Action，它们对应的物理处理类都是 com.ascenttech.action.LoginAction。login 和 regist 两个 Action 虽然有相同的处理类，但处理逻辑不同，它通过 method()方法指定，其中名为 regist 的 Action 对应的处理逻辑为默认的 exeute()方法，而名为 login 的 Action 对应的逻辑为指定的 login()方法。

再次看上面 struts.xml 文件中两个＜action…/＞元素的定义，发现两个 action 定义的绝

大部分相同，因此这种定义有大量冗余。为了解决这个问题，Struts 2 还有另一种形式的动态方法调用，即使用通配符的方式。

(2) 使用通配符。

在配置<action…/>元素时，可以指定 name、class 和 method 属性，这 3 个属性都可支持通配符，这种使用通配符的方式是动态方法调用的一种形式。当使用通配符定义 Action 的 name 属性时，相当于一个 action 元素定义多个逻辑 Action。

以下举例说明：

```
<action name="*Action" class="com.ascent.action.LoginAction" method="{1}">
  …
</action>
```

解释上面代码的含义：上面定义的不是一个普通的 action，而是定义了一系列的 action，只要 URL 是 *Action.action 的模式，都可以通过该 Action 进行处理，但该 Action 定义了一个表达式{1}，该表达式的值就是 name 属性值中的第一个 * 的值。

例如，如果用户请求的 URL 是 loginAction.action，则调用该 action 的 login()方法；如果用户请求的 URL 是 registAction.action，则调用该 action 的 regist()方法。LoginAction 类不再包含默认的 execute()方法，而是包含了 regist()和 login()两个方法，这两个方法与 execute()方法除了方法名不同外，其他完全相同。

除此之外，表达式也可出现在<action…/>元素的 class 属性中，即 Struts 2 允许将一系列 Action 类配置成一个<action…/>元素。例如：

```
<action name="*Action" class="com.ascent.action.{1}Action">
…
</action>
```

上面的<action…/>定义片段定义了一系列的 Action，这些 Action 名字应该匹配 *Action 模式，没有指定 method 属性，所以总是使用 execute()方法处理用户请求。但 class 属性值使用了表达式，上面配置片段的含义是，如果有 URL 为 RegistAction.action 请求，将匹配 *Action 模式，而交给该 Action 处理，其第一个 * 的值为 Regist，该 Regist 传入 class 属性值，即该 Action 的处理类为 com.ascent.action.RegistAction。

如果需要，Struts 2 允许在 class 属性和 method 属性中同时使用表达式。看如下的配置片段：

```
<action name="*_*" class="com.ascent.action.{1}Action" method="{2}">
```

当一个 action 为 Product_update.action 的时候，将调用 ProductAction 的 update()方法处理用户请求。现在的问题是，当用户请求的 URL 同时匹配多个 Action 时，究竟由哪个 Action 处理用户请求？

如果有 URL 为 abcAction.action 的请求，struts.xml 文件有名为 abcAction 的 Action，则一定由该 Action 处理用户请求；如果 struts.xml 文件没有名为 abcAction 的 Action，则搜索 name 属性值匹配 abcAction 的 Action，例如 name 为 *Action 或 *，*Action 并不会比 * 更优先匹配 abcAction 的请求，而是先找到哪个 Action，就先由哪个 Action 处理用户的请求。因此，应该将名为 * 的 Action 配置在最后，否则 Struts 2 将使用该 Action 处理所有希望使用

模式匹配的请求。

在 AscentWeb 项目的 struts.xml 中，使用了通配符：

```
<action name="*CartManagerAction" class="com.ascent.action.CartManagerAction"
method="{1}">
            <result name="cartshow">/product/cartshow.jsp</result>
        </action>
        <action name="*OrdersManagerAction" class="com.ascent.action.
OrdersManagerAction" method="{1}">
          <result name="checkoutsucc">/product/checkoutsucc.jsp</result>
          <result name="ordershow">/product/ordershow.jsp</result>
          <result name="adminordershow">/product/admin_ordershow.jsp</result>
        </action>
        <action name="*OrderitemManagerAction" class="com.ascent.action.
OrderitemManagerAction" method="{1}">
          <result name="orderitemshow">/product/orderitem_show.jsp</result>
        </action>
```

(3) 处理结果。

前面已经提到，Action 仅负责处理用户请求，它只是一个控制器，不能也不应该直接提供对浏览者的响应。当 Action 处理完用户请求后，处理结果应该通过视图资源实现，而控制器应该控制将哪个视图资源呈现给浏览者。

Action 处理完用户请求后，将返回一个普通字符串，整个普通字符串就是一个逻辑视图名。struts.xml 中包含逻辑视图名和物理视图之间的映射关系，一旦收到 Action 返回的某个逻辑视图名，系统就会把对应的物理视图呈现给浏览者。

相对于 Struts 1 框架而言，Struts 2 的逻辑视图不再是 ActionForward 对象，而是一个普通字符串，这样的设计更有利于将 Action 类与 Struts 2 框架分离，提供了更好的代码复用性。

除此之外，Struts 2 还支持多种结果映射，实际资源不仅可以是 JSP 视图资源，也可以是 FreeMaker 或 Velocity 等视图资源，甚至可以将请求转给下一个 Action 处理，形成 Action 的链式处理。

① 处理结果配置。

Struts 2 通过在 Struts.xml 文件中使用＜result…/＞元素配置结果。根据＜result…/＞元素所在位置的不同，Struts 2 提供了两种结果。

- 局部结果：将＜result…/＞作为＜action…/＞元素的子元素配置。
- 全局结果：将＜result…/＞作为＜global-result…/＞元素的子元素配置。

先介绍局部结果，它的作用范围是对特定的某个 Action 有效。局部结果是通过在＜action…/＞元素中指定＜result…/＞元素配置的，一个＜action…/＞元素可以有多个＜result…/＞元素，这表示一个 action 可以对应多个结果。

最典型的＜result…/＞配置片段如下：

```
<action name="usrLoginAction" class="com.ascent.action.UsrLoginAction">
    <result>/index.html</result>
    <result name="success_1">/product/products.jsp</result>
```

```
    <result name="success_2">/product/products.jsp</result>
    <result name="success_3">/product/products_showusers.jsp</result>
    <result name="error">/product/products.jsp</result>
    <result name="input">/product/products.jsp</result>
</action>
```

注：还可以使用＜param…/＞子元素配置结果，其中＜param…/＞元素的 name 属性可以为如下两个值。

- location：该参数指定了该逻辑视图对应的实际视图资源。
- parse：该参数指定是否允许在实际视图名字中使用 OGNL 表达式，该参数值默认为 true。如果设置该参数值为 false，则不允许在实际视图名中使用表达式。通常无须修改该属性值。

下面了解一下全局结果。Struts 2 的＜result…/＞元素配置，也可放在＜global-results…/＞元素中配置，当在＜global-results…/＞元素中配置＜result…/＞元素时，该＜result…/＞元素配置了一个全局结果，全局结果的作用范围对所有的 Action 都有效。

如果一个 Action 里包含了与全局结果里同名的结果，则 Action 里的局部 Action 会覆盖全局 Action。也就是说，当 Action 处理用户请求结束后，会首先在本 Action 里的局部结果里搜索逻辑视图对应的结果，只有在 Action 里的局部结果里找不到逻辑视图对应的结果，才会到全局结果里搜索。

② Struts 2 支持的处理结果类型。

Struts 2 支持使用多种视图技术，如 JSP、Velocity 和 FreeMarker 等。当一个 Action 处理用户请求结束后，仅返回一个字符串，这个字符串就是逻辑视图名，但该逻辑视图并未与任何的视图技术及任何的视图资源关联。实际上，结果类型决定了 Action 处理结束后，下一步将执行哪种类型的动作。

Struts 2 的结果类型要求实现 com.opensymphony.xwork.Result，这个结果是所有 Action 执行结果的通用接口。如果需要自己的结果类型，应该提供一个实现该接口的类，并且在 struts.xml 文件中配置该结果类型。

Struts 2 的 Struts-default.xml 和各个插件中的 Struts-plugin.xml 文件中提供了一系列的结果类型，表 3-2 列出的就是 Struts 2 支持的结果类型。

表 3-2 Struts 2 支持的结果类型

结 果 类 型	描 述
Chain 结果类型	Action 链式处理的结果类型
Chart 结果类型	用于整合 JFreeChart 的结果类型
dispatcher 结果类型	用于 JSP 整合的结果类型
freemarker 结果类型	用于 freemarker 整合的结果类型
httpheader 结果类型	用于控制特殊的 HTTP 行为的结果类型
Jasper 结果类型	用于 JasperReports 整合的结果类型
Jsf 结果类型	用于与 JSF 整合的结果类型
redirect 结果类型	用于直接重定向到其他 URL 的结果类型

续表

结 果 类 型	描　　述
redirect-action 结果类型	用于直接重定向到 Action 的结果类型
Stream 结果类型	用于向浏览器返回一个 InputStream(一般用于文件下载)
Tiles 结果类型	用于与 Tiles 整合的结果类型
Velocity 结果类型	用于与 Velocity 整合的结果类型
XSLT 结果类型	用于与 XML/XSLT 整合的结果类型
plaintext 结果类型	用于显示某个页面的源代码的结果类型

上面一共列出 14 种类型，其中 dispatcher 结果类型是默认的类型，也就是说，如果省略了 type 属性，默认 type 属性为 dispatcher，它主要用于与 JSP 页面整合。下面重点介绍 plaintext、redirect 和 redirect-action 3 种结果类型。

a. plaintext 结果类型

这个结果类型并不常用，因为它的作用太过局限：它主要用于显示实际视图资源的源代码。在 struts.xml 文件中采用如下的配置片段：

```
<result type="plaintext">
    <param name="location">/welcome.jsp</param>
    <!--设置字符集编码-->
    <param name="charset">gb2312</param>
</result>
```

这里使用了 plaintext 结果类型，系统将把视图资源的源代码呈现给用户。如果在 welcome.jsp 页面的代码中包含了中文字符，使用 plaintext 结果将会看到乱码。为了解决这个问题，Struts 2 通过＜param name＝"charset"＞gb2312＜/param＞元素设置使用特定的编码解析页面代码。

b. redirect 结果类型

这种结果类型与 dispatcher 结果类型相对，dispatcher 结果类型是将请求 forward(转发)到指定的 jsp 资源；而 redirect 结果类型则意味着将请求 redirect(重定向)到指定的视图资源。

dispatcher 结果类型与 redirect 结果类型的差别主要是转发和重定向的差别；重定向的效果是重新产生一个请求，因此所有的请求参数、请求属性、Action 实例和 Action 中封装的属性全部丢失。

完整地配置一个 redirect 的 Result，可以指定如下两个参数：

- location：该参数指定 Action 处理完用户请求后跳转的地址。
- parse：该参数指定是否允许在 location 参数值中使用表达式，该参数默认为 true。

c. redirect-action 结果类型

当一个 Action 处理结束后，直接将请求重定向(是重定向，不是转发)到另一个 Action 时，应该使用 redirect-action 结果类型。配置 redirect-action 结果类型时，可以指定如下两个参数：

- actionName：该参数指定重定向的 Action 名。
- namespace：该参数指定需要重定向的 Action 所在的命名空间。

下面是一个使用 redirect-action 结果类型的配置实例：

```
<result type=" redirect-action">
    <!--指定 action 的命名空间-->
    <param name="namespace">/ss</param>
    <!--指定 action 的名字-->
    <param name="actionName">login </param>
</result>
```

③ 动态结果。

动态结果的意思是在指定实际视图资源时使用了表达式语法，通过这种语法可以允许 Action 处理完用户请求后，动态转入实际的视图资源。

实际上，Struts 2 不仅允许在 class 属性、name 属性中使用表达式，还可以在＜action…/＞元素的＜result…/＞子元素中使用表达式。下面提供了一个通用 Action，该 Action 可以配置成如下形式：

```
<action name="*">
  <result>/{1}.jsp</result>
</action>
```

在上面的 Action 定义中，Action 的名字是一个 *，即它可以匹配任意的 Action，即所有的用户请求都可通过该 Action 处理。因为没有为该 Action 指定 class 属性，即该 Action 使用 ActionSupport 作为处理类，而且因为该 ActionSupport 类的 execute()方法返回 success 的字符串，即该 Action 总是直接返回 result 中指定的 JSP 资源，JSP 资源使用表达式生成资源名。上面 Action 定义的含义是：如果请求 a.action，则进入 a.jsp；如果请求 b.action，则进入 b.jsp 页面……以此类推。

另外，在配置＜result…/＞元素时，还允许使用 OGNL 表达式，这种用法允许让请求参数决定结果。在配置＜result…/＞元素时，不仅可以使用 ${0}表达式形式指定视图资源，还可以使用 ${属性名}的方式指定视图资源。在后面这种配置方式下，${属性名}里的属性名就是对应 Action 实例里的属性。例如：

```
<result type="redirect">edit.action?productName=${myProduct.name}</result>
```

对于上面的表达式语法，要求 action 中必须包含 myProduct 属性，并且 myProduct 属性必须包含 name 属性，否则 ${myProduct.name}表达式的值为 null。

3) include(包含)配置

在大部分应用里，随着应用规模的增加，系统中的 Action 数量也会大量增加，导致 struts.xml 配置文件变得非常臃肿。为了避免这种情况，可以将一个 struts.xml 配置文件分解成多个配置文件，然后在 struts.xml 文件中包含其他配置文件。通过这种方式，Struts 2 提供了一种模块化的方式管理 struts.xml 配置文件，体现了软件工程中“分而治之”的原则。

Struts 2 默认只加载 WEB-INF/class 下的 struts.xml 文件，所以必须通过 struts.xml 文件包含其他配置文件。

在 struts.xml 文件中包含其他配置文件通过＜include…/＞元素完成，配置＜include…/＞元素需要指定一个必需的属性，该属性指定了被包含配置文件的文件名。被包含的 struts 配置文件也是标准的 Struts 2 配置文件，一样包含 DTD 信息、Struts 2 配置文件的根元素等信息。

通常，将 Struts 2 的所有配置文件都放在 Web 应用的 WEB-INF/classes 路径下，strust.xml 文件包含了其他的配置文件，Struts 2 框架自动加载 strust.xml 文件，从而完成加载所有配置信息。

4）Bean 配置

Struts 2 框架是一个可扩展性的框架。对于框架的大部分核心组件，Struts 2 并不是直接以硬编码的方式写在代码中，而是以自己的 IoC（控制反转）容器管理框架的核心组件。关于 IoC，第 10 章将会详细讲解。

Struts 2 框架以可配置的方式管理 Struts 的核心组件，从而允许开发者可以很方便地扩展该框架的核心组件。当开发者需要扩展，或者替换 Struts 2 的核心组件时，只提供自己的组件实现类，并将该组件实现类部署在 Struts 2 的 IoC 容器中即可。

通常使用＜bean/＞元素在 struts.xml 文件中定义 Bean。bean 元素属性见表 3-3。

表 3-3　bean 元素属性

属　性	描　述
class	这个属性是一个必填属性，它指定了 Bean 实例的实现类
Type	这个属性是一个可选属性，它指定了 Bean 实例实现的 Struts 2 规范，该规范通常是通过某个接口实现的，因此该属性的值通常是一个 Struts 2 接口。如果需要将 Bean 实例作为 Struts 2 组件使用，则应该指定该属性值
Name	该属性指定了 Bean 实例的名字，对于有相同 type 类型的多个 Bean，它们的 name 属性不能相同。这个属性也是一个可选属性
Scope	该属性指定 Bean 实例的作用域。该属性是一个可选属性，属性值只能是 default、singleton、request、session 或 thread 中之一
Static	该属性指定 Bean 是否使用静态方法注入。通常，当指定了 type 属性时，该属性不应该指定为 true
optional	该属性指定该 Bean 是否为一个可选的 Bean，该属性是一个可选属性

在 struts.xml 文件中定义 Bean 时，通常有如下两个作用：

- 创建该 Bean 的实例，将该实例作为 Struts 2 框架的核心组件使用。
- Bean 包含的静态方法需要一个值注入。

在第一种用法下，因为 Bean 实例往往是作为一个核心组件使用的，因此需要告诉 Struts 容器该实例的作用——就是该实例实现了哪个接口，这个接口往往定义了该组件必须遵守的规范。

第二种用法则可以很方便地允许不创建某个类的实例，却可以接受框架常量。在这种用法下，通常需要设置 static＝"true"。

注意：对于绝大部分 Struts 2 应用而言，无须重新定义 Struts 2 框架的核心组件，也就无须在 struts.xml 文件中定义 Bean。

5）常量配置

在 struts.xml 文件中配置常量是一种指定 Struts 2 属性的方式。稍后会介绍如何在 struts.properties 文件中配置 Struts 2 属性，这两种方式的作用基本相似。通常推荐在 struts.xml 文件中定义 Struts 2 属性，而不是在 struts.properties 文件中定义 Struts 2 属性的方式，这主要是为了保持与 WebWork 的向后兼容性。另外，还可以在 web.xml 文件中配置 Struts 2

常量。

通常，Struts 2 框架按如下搜索顺序加载 Struts 2 常量：

- struts-default.xml：该文件保存在 struts2-2.0.6.jar 文件中。
- struts-plugin.xml：该文件保存在 struts2-xxx-2.0.6.jar 等 Struts 2 插件 jar 文件中。
- struts.xml：该文件是 Web 应用默认的 Struts 2 配置文件。
- struts.properties：该文件是 Web 应用默认的 Struts 2 配置文件。
- web.xml：该文件是 Web 应用的配置文件。

上面指定了 Struts 2 框架搜索 Struts 2 常量顺序，如果在多个文件中配置了同一个 Struts 2 常量，则后一个文件中配置的常量值会覆盖前面文件中配置的常量值。

在不同的文件中配置常量的方式不一样，但不管在哪个文件中，配置 Struts 2 常量都需要指定两个属性：常量 name 和常量 value。

其中，在 struts.xml 文件中通过元素 constant 配置常量。配置常量需要指定两个必填的属性。

- name：该属性指定了常量 name。
- value：该属性指定了常量 value。

如果需要指定 Struts 2 的国际化资源文件的 baseName 为 mess，则可以在 strust.xml 文件中使用如下的代码片段：

```
<?xml  version="1.0"  encoding="UTF-8 "  ?>
<!--指定 Struts 2 的 DTD 信息-->
<!DOCTYPE Struts  PUBLIC
    "-//Apache  Software  Foundation//DTD  Struts  Configuration  2.0//EN"
     "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <!--通过 constant 元素配置 Struts 2 的属性-- >
    <constant name="struts.custom.i18n.resources" value="properties/myMessages"/>
    …
</struts>
```

上面的代码片段中配置了一个常用属性 struts.custom.i18n.resources，该属性指定应用所需的国际化资源文件的 baseName 为 properties/myMessages。

对于 struts.properties 文件而言，该文件的内容就是系列的 key-value 对，其中每个 key 对应一个 Struts 2 常量 name，而每个 value 对应一个 Struts 2 常量 value。关于 struts.properties 配置文件，稍后会详细介绍。

在 web.xml 文件中配置 Struts 2 常量，可通过＜filter＞元素的＜int-param＞子元素指定，每个＜int-param＞元素配置了一个 Struts 2 常量。

实际开发中不推荐将 Struts 2 常量配置在 web.xml 文件中。毕竟，采用这种配置方式配置常量需要更多的代码量，而且降低了文件的可读性。通常推荐将 Struts 2 常量集中在 strust.xml 文件中进行管理。

6）拦截器配置

拦截器其实就是 AOP（面向方面编程）的编程思想。关于面向方面编程，将会在第 11 章详细讲解。拦截器允许在 Action 处理之前，或者 Action 处理结束之后，插入开发者自定义的

代码。

在很多情况下，需要在多个 Action 中进行相同的操作，如权限控制，此处就可以使用拦截器检查用户是否登录，用户的权限是否足够（当然，也可以借助 Spring 的 AOP 框架完成）。通常，使用拦截器可以完成如下操作。

- 进行权限控制（检查浏览者是否为登录用户，并且有足够的访问权限）。
- 跟踪日志（记录每个浏览者所请求的每个 Action）。
- 跟踪系统的性能瓶颈（可以通过记录每个 Action 开始处理时间和结束时间，从而取得耗时较长的 Action）。

Struts 2 也允许将多个拦截器组合在一起，形成一个拦截器栈。一个拦截器栈可以包含多个拦截器，多个拦截器组成下一个拦截器栈。对于 Struts 2 系统而言，多个拦截器组成的拦截器栈对外也表现成一个拦截器。

定义拦截器之前，必须先定义组成拦截器栈的多个拦截器。Struts 2 把拦截器栈当成拦截器处理，因此拦截器和拦截器栈都放在＜interceptors…/＞元素中定义。

下面是拦截器的定义片段：

```
<interceptors>
    <interceptor name="log" class="cc.dynasoft.LogInterceptor" />
    <interceptor name="authority" class="cc.dynasoft. Authority Interceptor" />
    <interceptor name="timer" class="cc.dynasoft.TimerInterceptor" />
    <interceptor-stack name="default">
      <interceptor-ref name=" authority" />
    <interceptor-ref name=" timer" />
    </interceptor>
    …
</interceptors>
```

一旦定义拦截器和拦截器栈之后，在 Action 中使用拦截器或拦截器栈的方式是相同的。

```
<action name="login" class="cc.dynasoft.LoginAction">
…
   <interceptor-ref name="log" />
</action>
```

在我们的项目中配置了 fileUpload 拦截器，如下所示：

```
<action name=" * ProductManagerAction" class="com.ascent.action.ProductManagerAction"
method="{1}">
    <!-- 配置 fileUpload 拦截器 -->
    <interceptor-ref name="fileUpload">
        <!-- 设置上传文件类型 -->
        <param name="allowedTypes">image/bmp,image/png,image/jpg,image/gif,
application/vnd.ms-excel </param>
        <!-- 设置上传文件的大小 -->
        <param name="maximumSize">200000</param>
    </interceptor-ref>
    <!-- 必须显示配置引用 struts 默认的拦截器栈:defaultStack -->
```

```
    <interceptor-ref name="defaultStack"></interceptor-ref>
    ...
</action>
```

2. struts.properties 配置文件

除 struts.xml 核心文件外，Struts 2 框架还包含一个 struts.properties 文件，该文件通常放在 Web 应用的 WEB-INF/classes 路径下。它定义了 Struts 2 框架的大量属性，开发者可以通过改变这些属性满足个性化应用的需求。

struts.properties 中定义的 Struts 2 属性见表 3-4。

表 3-4 struts.properties 中定义的 Struts 2 属性

属 性	描 述
struts.configuration	该属性指定加载 Struts 2 配置文件的配置文件管理器。该属性的默认值是 org.apache.Struts2.config.DdfaultConfiguration，这是 Struts 2 默认的配置文件管理器。如果需要实现自己的配置管理器，开发者开发一个实现 configuration 接口的类，该类可以自己加载 Struts 2 配置文件
Struts.locale	指定 Web 应用的默认 Locale
Struts.i18n.encoding	指定 Web 应用的默认编码集。该属性对于处理中文请求参数非常有用，对于获取中文请求参数值，应该将该属性值设置为 GBK 或者 GB2312。 提示：当设置该参数为 GBK 时，相当于调用 httpservletrequest 的 setcharacterencoding()方法
struts.objectFactory	指定 Struts 2 默认的 objectFactory bean，该属性的默认值是 Spring
struts.objectFactory.spring.autoWire	指定 Spring 框架的自动装配模式，该属性的默认值是 name，即默认根据 Bean 的 name 属性自动装配
struts.objectFactory.spring.useClassCache	该属性指定整合 Spring 框架时，是否缓存 Bean 实例。该属性只允许使用 true 和 false 两个属性值，它的默认值是 true。通常不建议修改该属性值
struts.objectTypeDeterminer	该属性指定 Struts 2 的类型检测机制，通常支持 tiger 和 notiger 两个属性值
Struts.multipart.parser	该属性指定处理 multipart/form-data 的 MIME 类型（文件上传）请求的框架。该属性支持 cos、pell 和 jakarta 等属性值，即分别对应使用 cos 的文件上传框架、pell 上传及 common-fileupload 文件上传框架。该属性的默认值为 jakarta。 如果需要使用 cos 或者 pell 的文件上传方式，则应将对应的 JAR 文件复制到 Web 应用中。例如，使用 cos 上传方式，则需要自己下载 cos 框架的 JAR 文件，并将该文件放在 WEB-INF/lib 路径下
Struts.multipart.savedir	该属性指定上传文件的临时保存路径，默认值是 javax.servlet.context.tempdir
Struts.multipart.maxsize	该属性指定 Struts 2 文件中整个请求内容允许的最大字节数
Sturts.custom.properties	该属性指定 Struts 2 应用加载用户自定义的属性文件，该自定义属性文件指定的属性不会覆盖 struts.properties 文件中指定的属性。如果需要加载多个自定义属性文件，则多个自定义属性文件的文件名以英文逗号(,)隔开

续表

属　　性	描　　述
Struts.mapper.class	指定将 HTTP 请求映射到指定 Action 的映射器，Struts 2 提供了默认的映射器 org.pache.struts2.dispatcher.mapper.defaultactionmapper。默认映射器根据请求的前缀与 Action 的 name 属性完成映射
Struts.action.extension	该属性指定需要 Struts 2 处理的请求后缀，默认值是 action，即所有匹配 *.action 的请求都由 Struts 2 处理。如果用户需要指定多个请求后缀，则多个后缀之间以英文逗号(,)隔开
Struts.serve.static	该属性设置是否通过 JAR 文件提供静态内容服务，该属性只支持 true 和 false 属性值，该属性的默认属性值是 true
Struts.serve.static.browsercache	该属性设置浏览器是否缓存静态内容。当应用处于开发阶段时，若希望每次请求都获得服务器的最新响应，则可设置该属性为 false
Struts.enable.dynamicmethodinvocation	该属性设置 Struts 2 是否支持动态方法调用，默认值是 true。如果需要关闭动态方法调用，则可设置该属性为 false
Struts.enable.slashesinactionnames	该属性设置 Struts 2 是否允许在 Action 名中使用斜线，默认值是 false。如果开发者希望允许在 Action 名中使用斜线，则可设置该属性为 true
Struts.tag.altsyntax	该属性指定是否允许在 Struts 2 标签中使用表达式语法，因为通常需要在标签中使用表达式语法，故此属性应设置为 true。该属性的默认值是 true
Struts.devmode	该属性用于设置 Struts 2 应用是否使用开发模式。如果设置该属性为 true，则可以在应用出错时显示更多、更友好的出错提示。该属性只接受 true 和 false 两个值，默认值是 false。通常，应用在开发阶段，将该属性设置为 true，当进入产品发布阶段后，则该属性设置为 false
Struts.i18n.reload	该属性用于设置是否每次 HTTP 请求到达时，系统都重新加载资源文件，默认值是 false。在开发阶段将该属性设置为 true 会更有利于开发，但在产品发布阶段应将该属性设置为 false。 提示：在开发阶段将该属性设置为 true，可以在每次请求时重新加载国际化资源文件，从而让开发者看到实时开发效果；在产品发布阶段将该属性设置为 false，是为了提供响应性能，每次请求都重新加载资源文件会大大降低应用的性能
Struts.ui.theme	该属性指定视图标签默认的视图主题，默认值是 xhtml
Struts.ui.templateDir	该属性指定视图主题所需要模板文件的位置，默认值是 template，即默认加载 template 路径下的模板文件
Struts.ui.templateSuffix	该属性指定模板文件的后缀，默认值是 ftl。该属性还允许使用 ftl、vm 或 jsp，分别对应 FreeMarker、Velocity 和 JSP 模板
Struts.configuration.xml.reload	该属性设置当 struts.xml 文件改变后，系统是否自动重新加载该文件，默认值是 false
Struts.velocity.configfile	该属性指定 Velocity 框架所需的 velocity.properties 文件的位置，默认值是 velocity.properties
Struts.velocity.toolboxlocation	该属性指定 Velocity 框架的 Context 位置，如果该框架有多个 Context，则多个 Context 之间以英文逗号(,)隔开
Struts.velocity.toolboxlocation	该属性指定 Velocity 框架的 toolbox 的位置
Struts.url.http.port	该属性指定 Web 应用所在的监听端口。该属性通常没有太大的用户，只是当 Struts 2 需要生成 URL 时(如 Url 标签)，该属性才提供 Web 应用的默认端口

续表

属　　性	描　　述
Struts.url.https.port	该属性类似于 Struts.url.http.port 属性的作用，区别是该属性指定的是 Web 应用的加密服务端口
Struts.url.includeparams	该属性指定 Struts 2 生成 URL 时是否包含请求参数。该属性接受 none、get 和 all 3 个属性值，分别对应于不包含、仅包含 GET 类型请求参数和包含全部请求参数
Struts.custom.i18n.resources	该属性指定 Struts 2 应用所需要的国际化资源文件，如果有多份国际化资源文件，则多个资源文件的文件名以英文逗号(,)隔开
Struts.dispatcher.parametersWorkaround	某些 Java EE 服务器不支持 HttpServletRequest 调用 getparameterMap()方法，此时可以设置该属性值为 true 解决该问题。该属性的默认值是 false。对于 WebLogic、Orion 和 OC4J 服务器，通常应该设置该属性为 true
Struts.freemarker.manager.classname	该属性指定 Struts 2 使用的 FreeMarker 管理器，默认值是 org.apache.struts2.views.freemarker.FreemarkerManager，这是 Struts 2 内建的 FreeMarker 管理器
Struts.freemarker.wrapper.altMap	该属性只支持 true 和 false 两个属性值，默认值是 true。通常无须修改该属性值
Struts.xslt.nocache	该属性指定 XSLT Result 是否使用样式表缓存。当应用处于开发阶段时，该属性通常设置为 true；当应用处于产品使用阶段时，该属性通常设置为 false
Struts.configuration.files	该属性用于指定 Struts 2 框架默认加载的配置文件，如果需要指定默认加载多个配置文件，则多个配置文件的文件名之间以英文逗号(,)隔开。该属性的默认值为 Struts-default.xml，struts-plugin.xml，struts.xml，看到该属性值，读者应该明白为什么 Struts 2 框架默认加载 struts.xml 文件

有时开发者不喜欢使用额外的 struts.properties 文件。前面提到，Struts 2 允许在 struts.xml 文件中管理 Struts 2 属性，在 struts.xml 文件中管理 Struts 2 属性，在 struts.xml 文件中通过配置 constant 元素，一样可以配置这些属性。前面已经提到，建议尽量在 strust.xml 文件中配置 Struts 2 常量。

3.4 创建 Controller 组件

Struts 的核心是 Controller 组件。它是连接 Model 和 View 组件的桥梁，也是理解 Struts 架构的关键。正如前面提到的，Struts 2 的控制器由两个部分组成：FilterDispatcher 和业务控制器 Action。

3.4.1 FilterDispatcher

任何 MVC 框架都需要与 Web 应用整合，这就离不开 web.xml 文件，只有配置在 web.xml 文件中，Filter/servlet 才会被应用加载。对于 Struts 2 框架而言，需要加载 FilterDispatcher。因为 Struts 2 将核心控制器设计成 filter，而不是一个 servlet，故为了让 Web 应用加载 FilterDispatcher，需要在 web.xml 文件中配置 FilterDispatcher。