# 第5章 Spring MVC基础

### 学习目的与学习要求

学习目的：了解 Spring MVC 框架，掌握 Spring MVC 框架的工作流程及组件，熟悉 Spring MVC 原理，包括核心控制器 DispatcherServlet、Controller 控制器、ModelAndView 和视图解析 ViewResolver 等。

学习要求：熟练搭建和开发基于 Spring MVC 框架的应用系统，顺利完成本章案例的开发，并能够开发项目的其他功能。

### 本章主要内容

本章主要内容包括 MVC 模式的概述、MVC 与 Spring MVC 的映射、Spring MVC 框架的工作流程和组件、Spring MVC 原理（包括核心控制器 DispatcherServlet、Controller 控制器、ModelAndView 和视图解析 ViewResolver 等）以及 Spring MVC 开发步骤。

Spring MVC 框架是 MVC 设计模式的一个具体实现，所以先介绍 MVC 模式。

## 5.1 MVC 模式概述

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一再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。

模式的概念开始时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，并且应用得也比较多。但是，直到 GoF 的书出来之前，并没有一种统一的认识。或者说，那时对模式并没有形成一个概念，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，就可以完全或者说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。

MVC 模式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 5-1 所示。

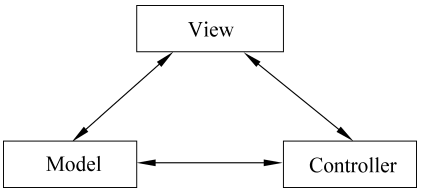


图 5-1　MVC 架构图

**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，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 模式可以允许多于一个的 View 端存在，并可以在需要的时候动态注册需要的 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将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容易地使用多个接口,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。

## 5.2 Spring MVC 概述

Spring MVC 属于 Spring Framework 的后续产品,已经融合在 Spring Web Flow 里。Spring MVC 分离了控制器、模型对象、分派器以及处理程序对象的角色,这让它们更容易进行定制。

Spring 的模型-视图-控制器(MVC)框架核心是围绕一个 DispatcherServlet 设计的,这个 Servlet 会将来自客户端的 HTTP 请求分发给各个用户自定义开发的处理器,同时支持可配置的处理器映射关系、页面视图的渲染、消息本地化和国际化,以及时区设置与页面风格主题渲染等。

处理器被称为 Controller,是在开发应用中注解了@Controller 和@RequestMapping 的 Java 类和方法。由于 Spring MVC 作为 Spring 的 Web MVC 开发组件内置在 Spring 中,所以 Spring 为 Spring MVC 的处理器方法提供了极其多样灵活的配置。Spring 3.0 以后提供了@Controller 注解机制、@PathVariable 注解,以及一些其他的特性,使得开发更加灵活、简洁,同时也可以使用 Spring MVC 进行 RESTful Web 站点和应用的开发。

在 Spring Web MVC 中,可以使用任何 Java 对象作为执行请求处理的命令对象或表单数据处理的返回对象,而无须实现与框架相关的接口或基类。Spring 的数据绑定非常灵活:例如,它会把数据类型不匹配当成可由应用自行处理的运行时验证错误,而非系统错误。之前你可能会为了避免非法的类型转换,在表单对象中使用字符串存储数据,但无类型的字符串无法描述业务数据的真正含义,并且还需要把它们转换成对应的业务对象类型。有了 Spring 的验证机制,再也不必这么做了,直接将业务对象绑定到表单对象上通常是更好的选择。

Spring 的视图解析设计得也异常灵活。控制器一般负责准备一个 Map 模型、填充数据、返回一个合适的视图名等,同时它也可以直接将数据写到响应流中。视图名的解析高度灵活,支持多种配置,包括通过文件扩展名、Accept 内容头、Bean、配置文件等的配置,甚至还可以自己实现一个视图解析器 ViewResolver。模型(Model)其实是一个 Map 类型的接口,彻底地把数据从视图技术中抽象分离了出来。可以与基于模板的渲染技术直接整合,如 JSP、Velocity 和 Freemarker 等,也可以直接生成 XML、JSON、Atom 以及其他多种类型的内容。Map 模型会简单地被转换成合适的格式,如 JSP 的请求属性(attribute)、一个 Velocity 模板的模型等。

## 5.3 MVC 组件和流程

Spring MVC 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。

Spring MVC 对于 URL 请求的处理框架如图 5-2 所示。

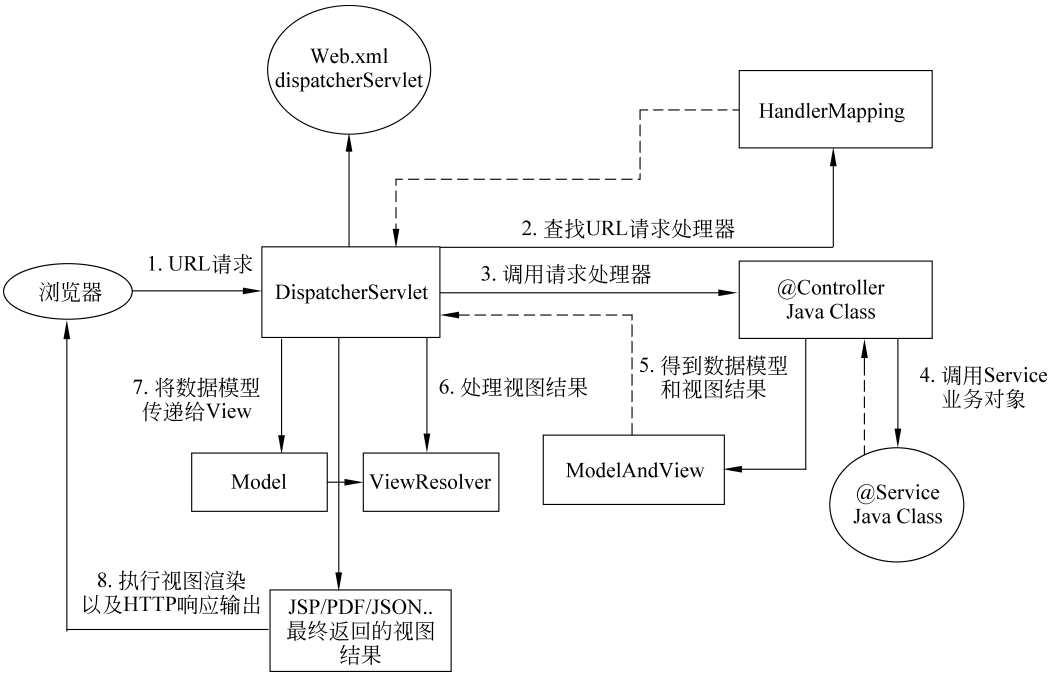


图 5-2　Spring MVC 的体系概图

上述的架构图中包括了 Spring MVC 中的核心功能组件的定义。

(1) DispatcherServlet：Spring MVC 提供的前端控制器，所有的请求都经过它识别，以及负责统一请求分发和响应处理。如 login.action 这样的 URL 请求信息，首先需要 DispatcherServlet 识别请求，然后根据配置信息在 HandlerMapping 中查找处理该 URL 的具体的 Java 类和相应的处理方法。

(2) HandlerMapping：保存了 URL 与 Controller Java 类和方法的映射关系，DispatcherServlet 需要借助该对象定位具体的 Java 对象和方法，以便处理 URL 请求。

(3) Controller：被标记和实现为 Controller 的类通过内部定义的 Java 方法，为并发发出同样 URL 请求的用户处理每个请求，因此实现 Controller 接口时，须保证线程安全并且可重用。Controller 类将调用对应 URL 的 Java 方法处理用户请求，并由开发人员决定是否使用 ModelAndView。一旦 Controller 中的 Java 方法处理完用户请求，则由开发人员决定使用怎样的方式返回 ModelAndView 对象给 DispatcherServlet 前端控制器。

(4) ModelAndView：包含了模型(Model)和视图(View)。模型是处理之后需要展示或者传递到 View 中的数据，如用户信息、登录消息等。View 负责展示动态数据的各种页面技术，如 JSP、PDF、XML、JSON 格式(JavaScript 中的数据对象)等。

(5) ViewResolver：Spring 提供的视图解析器(ViewResolver)在 Web 应用中查找 View(数据展示视图)对象，从而将相应模型中的数据与视图进行渲染(解析处理)，最后将渲染后的结果返回给客户端。

从框架设计整体来说，DispatcherServlet 是整个 Web 应用的控制器，需要在 web.xml 中配置；Controller 是针对具体某个 URL 的单个 Http 请求处理过程中的控制器，是需要开发与配置的 Java 类；而 ModelAndView 是包含了处理 Http 请求过程中的需要返回客户端的数据模型(Model)和在客户端需要展现数据方式的视图(View)两部分内容。

这些对象在 Spring MVC 框架中相互配合，用于处理来自客户端的请求，其主要工作流程如下：

① 客户端请求提交到 DispatcherServlet。

② 由 DispatcherServlet 控制器查询一个或多个 HandlerMapping，找到处理请求的 Controller。

③ DispatcherServlet 将请求提交到 Controller。

④ Controller 调用业务逻辑处理后，返回 ModelAndView。

⑤ DispatcherServlet 查询一个或多个 ViewResoler 视图解析器，找到 ModelAndView 指定的视图。

⑥ 视图负责将结果显示到客户端。

先看一个用 MyEclipse 开发的 Spring MVC Web 应用 helloworld 实例，接下来将详细讲解相关理论内容。

(1) 新建一个 Web 项目，项目名称为 Spring MVC_HelloWorld_demo。

选择 File→New→Web Project 命令，如图 5-3 所示。

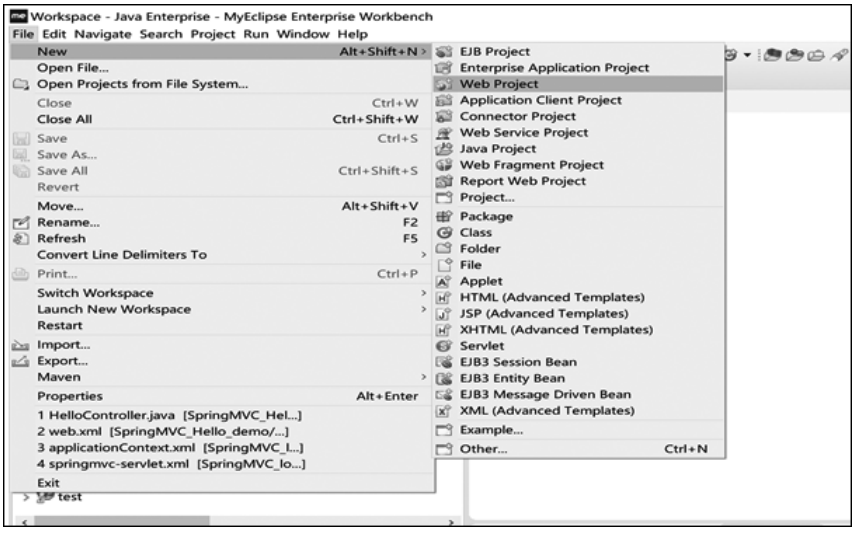


图 5-3 新建 Web Project

在 Project name 中命名为 Spring MVC_HelloWorld_demo，如图 5-4 所示。

单击 Next 按钮，保持默认配置，之后再单击 Next 按钮，勾选 Generate web.xml deployment descriptor，如图 5-5 所示。

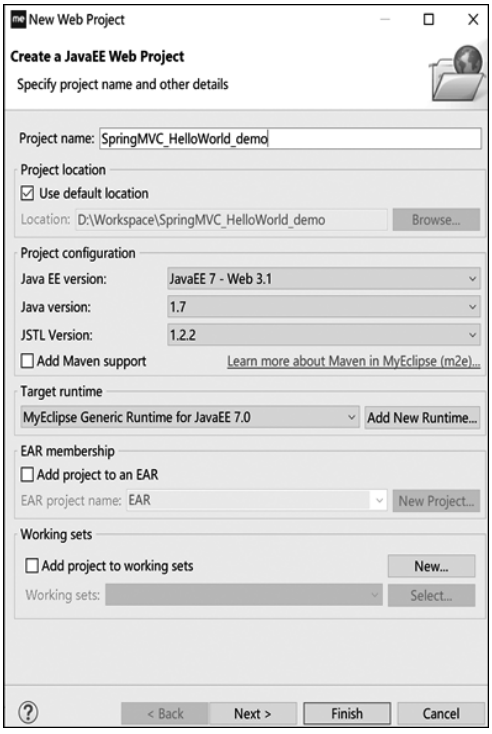


图 5-4　命名 Web Project

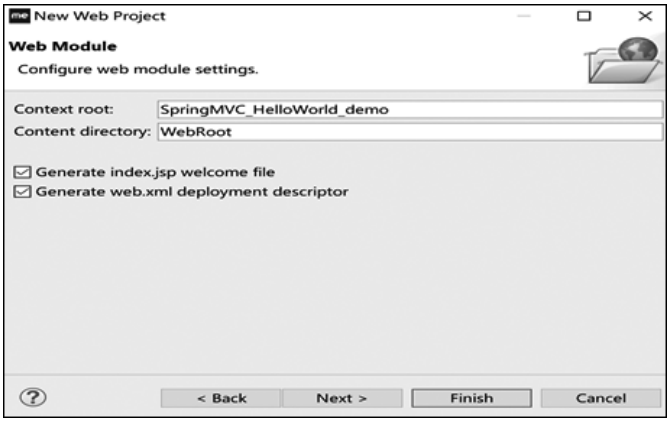


图 5-5　勾选 Generate web.xml deployment descriptor

最后单击 Finish 按钮。

(2) 添加 Spring MVC 相关 jar 包。右击 Spring MVC_HelloWorld_demo 项目，从弹出的快捷菜单中选择 Configure Facets...→Install Spring Facet，如图 5-6 所示。

之后出现图 5-7。

保持默认设置，单击 Next 按钮，出现图 5-8。

保持默认设置，单击 Next 按钮，出现图 5-9。

不需修改(这里需要 Spring 4.1.0 Libraries 中的 Core、Facets 和 Spring Web jar 包)，单击 Finish 按钮。

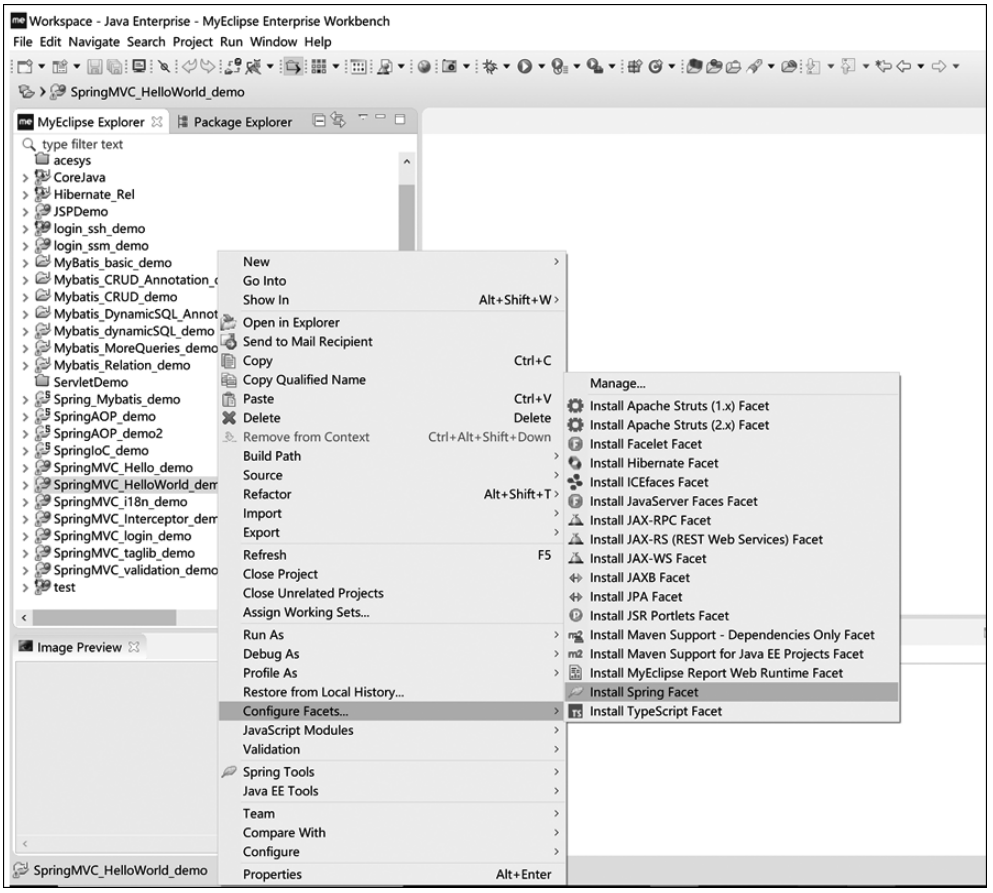


图 5-6　Install Spring Facet 界面 1

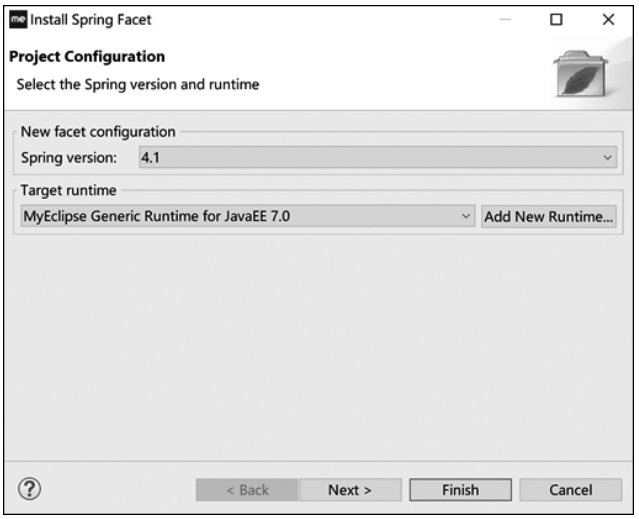


图 5-7　Install Spring Facet 界面 2

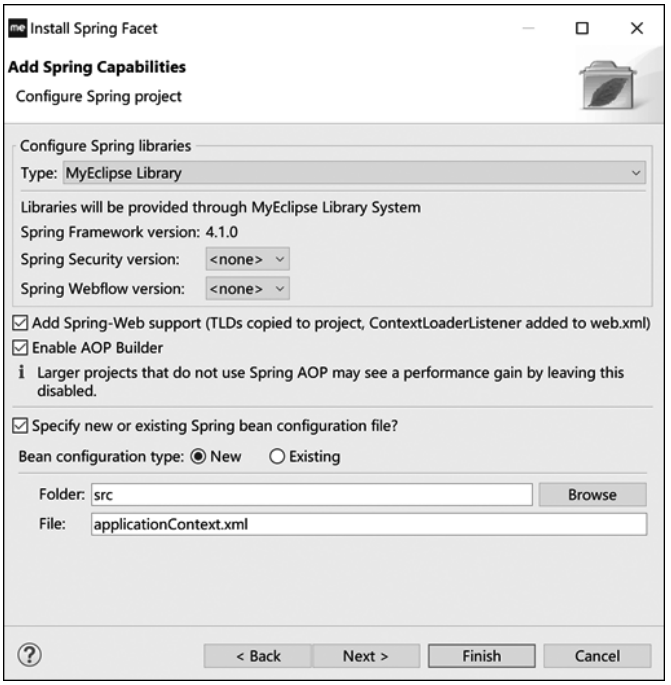


图 5-8　Install Spring Facet 界面 3

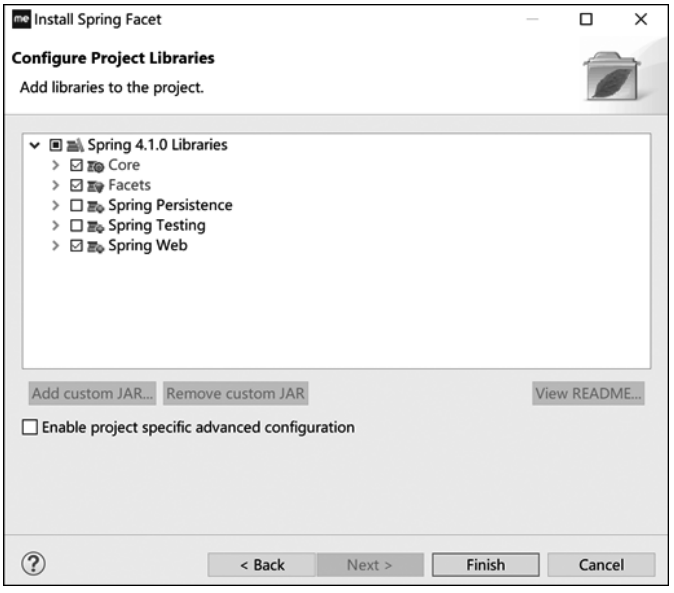


图 5-9　Install Spring Facet 界面 4

(3) 建立项目目录。

右击 src,从弹出的快捷菜单中选择 New→Package,如图 5-10 所示。

命名包名为 com.ascent,如图 5-11 所示,之后单击 Finish 按钮。

我们将在这个包里存放 Java 代码。

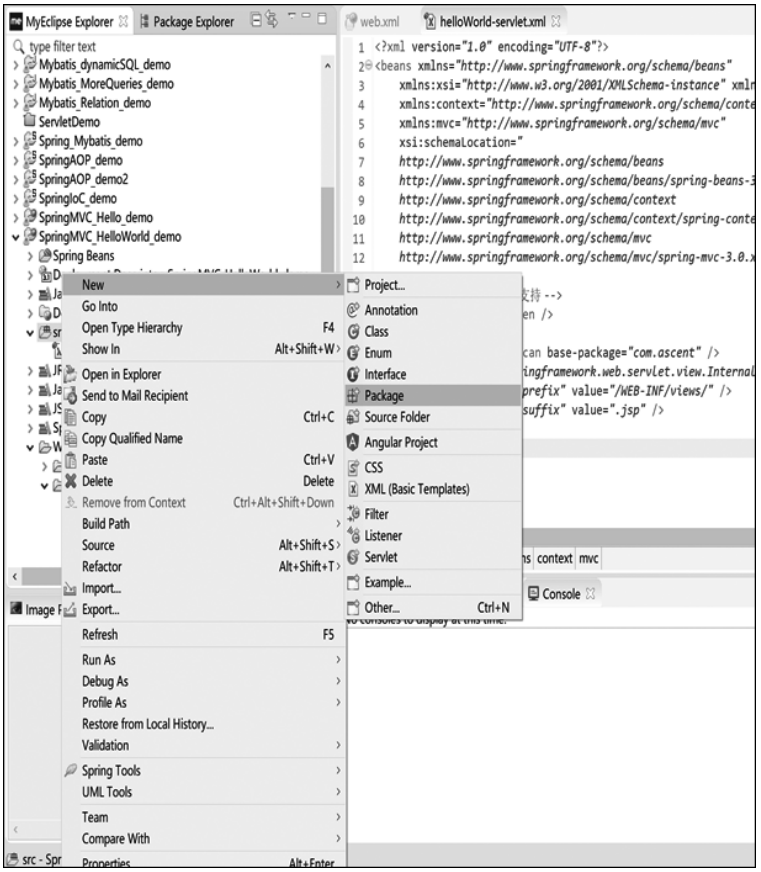


图 5-10 新建 Package

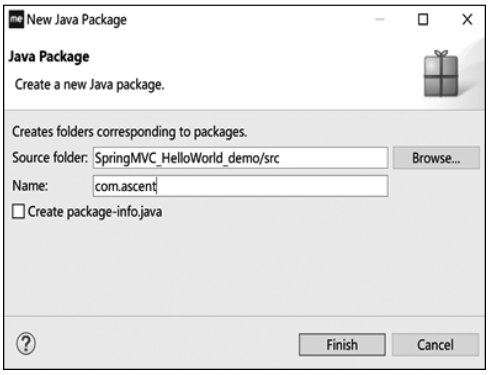


图 5-11 命名 Package

(4) 修改 web.xml,添入以下内容。

```
<servlet>
    <servlet-name>helloWorld</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
```

```
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>helloWorld</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

这里将 DispatcherServlet 命名为 helloWorld，并且让它在 Web 项目一启动就加载。接下来需要在 WEB-INF 目录下创建一个 helloWorld-servlet.xml 的 Spring 配置文件，此文件名的命名规则为：在 servlet-name 名称后面加上-servlet.xml。

(5) 添加 helloWorld-servlet.xml 配置文件，内容如下。

```
<?xml version="1.0" encoding="UTF-8"? >
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

    <!--默认的注解映射的支持 -->
    <mvc:annotation-driven />
<!--启用自动扫描  -->
<context:component-scan base-package="com.ascent" />
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
</bean>
</beans>
```

添加了 mvc 名称空间，接下来启用了 Spring 的自动扫描，并且设置了默认的注解映射支持。其中 base-package="com.ascent" 是我们之前建立的 package 名。

配置文件中的 Bean 的类型是 Spring MVC 中最常用的一种视图解析器。prefix 属性是指视图前缀，suffix 是视图后缀，这里配置的是.jsp，我们在控制器的方法 sayHello()中返回的是 hello，结合这里的配置，对应的完整的视图是/WEB-INF/views/hello.jsp。

(6) 编写 HelloWorldController 文件。

右击 com.ascent 包，从弹出的快捷菜单中选择 New→Class，如图 5-12 所示。

在 Name 处填入 HelloWorldController，单击 Finish 按钮，如图 5-13 所示。