T SrainijSoftware

齿圏H

[www.epubit.com.cn](http://www.epubit.com.cn)

Spring MVC

学习指南

(第2版）

[美]Paul Deck 著 林仪明译



0 SI

0 IQ s

s§

3AW M量 S

9JVMyosf^l*•置coJli*



图书在版编目（cIP**)数据**

SpringMVC**学习指南：第2版/(美）戴克** (PaulDeck**)著；林仪明译.一北京：人民邮电出版** 社，2017. 5

ISBN 978-7-115-44759-3

I **.①**S"\_II**.①戴…②林…**III**.①**JAVA语言一程序 设计一指南IV.①TP312.8-62

中国版木图书馆CIP数据核字(2017)第031857号

版权声明

Simplified Chinese translation copyright ©2017 by Posts and Telecommunications Press ALL RIGHTS RESERVED

Spring MVC A Tutorial, Second Edition by Paul Deck Copyright © 2017 by Brainy Software Inc.

本书中文简体版由作者Paul Deck授权人民邮电出版社出版。未经出版者书面许可，对本书的任何部分不 得以任何方式或任何手段复制和传播。

版权所有，侵权必宄。

♦著 [美]Paul Deck

译 林仪明

责任编辑陈冀康 责任印制焦志炜

♦人民邮电出版社出版发行 北京市丰台区成寿寺路11号

邮编 100164 电子邮件 [315@ptpress.com.cn](mailto:315@ptpress.com.cn) 网址 <http://www.ptpress.com.cn>

北京鑫正大印刷有限公司印刷

♦开本：800x1000 1/16

印张：21.5

字数：406千字 2017年5月第1版

印数：1 -3 000册 2017年5月北京第1次印刷

著作权合同登记号图字：01-2016-5102号

定价：59.00元

读者服务热线：（010)81055410印装质量热线：（010)81055316  
反盗版热线：（010)81055315  
广告经营许可证：京东工商广字第8052号

内容提要

Spring MVC是Spring框架中用于Web应用快速开发的一个模块，其中的MVC是 Model-View-Controller的缩写。作为当今业界最主流的Web开发框架，Spring MVC已经成为 当前最热门的开发技能，同时也广泛用于桌面开发领域。

本书重在讲述如何通过Spring MVC来开发基于Java的Web应用。全书共包括13章和5 个附录，分别从Spring框架、模型2和MVC模式、Spring MVC介绍、控制器、数据绑定和 表单标签库、转换器和格式化、验证器、表达式语言、JSTL、国际化、上传文件、下载文件 以及应用测试等多个角度介绍了 Spring MVC。附录部分分别介绍了 Tomcat、Spring Tool Suite 和Maven等工具的安装和配置，Servlet、JavaServer Pages和部署描述符的相关参考资料。除 此之外，本书还配有丰富的示例以供读者练习和参考。

本书是一本Spring MVC的教程，内容细致、讲解清晰，非常适合Web开发者和想要使 用Spring MVC开发基于Java的Web应用的读者阅读。

作者简介

Paul Deck是一位资深的Java和Spring专家，开发过大型的企业级应用，并且目前是一 位独立的软件工程师。他是《How Tomcat Works》一书的合著者。

译者简介

林仪明，男，现为IBM高级工程师。2004年毕业于厦门大学软件学院，主要研宄软件架 构、应用中间件。目前在福州生活和工作，先后从事软件技术开发，软件架构设计以及团队 管理等工作，有多年的开发设计和管理经验，目前提供IBM中间件产品支持工作。

、产1 、4

刖目

欢迎阅读本书。

Spring MVC是Spring框架中用于Web应用快速开发的一个模块。Spring MVC的MVC 是 Model-View-Controller E

不仅常见于Web开发，也广泛应用于如Swing和JavaFX等桌面开发。

作为当今业界最主流的Web开发框架，Spring MVC (有时也称Spring Web MVC)的开发技 能相当热门。本书可供想要学习如何通过Spring MVC开发基于Java Web应用的开发人员阅读。

Spring MVC 基于 Spring 框架、Servlet 和 JSP (JavaServer Page)，在掌握这 3 种技术的基 Spring MVC 1 章针对 Spring C

和附录D Servlet和JSP Servlet和JSP,推荐阅读由

BudiKurniawan 所著的《(Sferv/efawd JSP.- dTMton.a/，

接下来，我们将介绍HTTP、基于Servlet和JSP的Web编程，以及本书的章节结构。

HTTP

HTTP使得Web服务器与浏览器之间可以通过互联网或内网进行数据交互。作为一个制定标 准的国际社区，万维网联盟（W3C)负责和维护HTTP。HTTP第一版是HTTP 0.9，随后更新为 HTTP 1.0,之后的版本是HTTP 1.1。HTTP 1.1版本的RFC编号是2616。编写本书时，HTTP 1.1 依然是当前最流行的HTTP版本。当前最新版本是发布于2015年5月的HTTP/2,附表1.1列出了 HTTP針版本及其发布时间，HTTP的第二个主要版本通常表示为HTTP/2,而不是HTTP2。

附表**1.1 HTTP**版本与发布曰期

|  |  |  |
| --- | --- | --- |
| 版本 | 发布时间 | |
| HTTP 0.9 | 1991 年 | |
| HTTP 1.0 | 1996 年 | |
| HTTP 1.1 | 1997 | 1999 |
| HTTP/2 | 2015年5月 | |

Web服务器每天24小时不间断运行，并等待HTTP客户端（通常是Web浏览器）来连 接并请求资源。通常，客户端发起一个连接，服务端不会主动连接客户端。

互联网用户需要通过点击链接或者输入一个URL地址来访问一个资源，以下为两个 示例：

<http://google.com/index.html> <http://facebook.com/index.html>

URL的第一个部分是HTTP，代表所采用的协议。除HTTP外，URL还可以采用其他类 型的协议，以下为两个示例：

<mailto:joe@example.org>

ftp://marketing@ftp.example.org

通常，HTTP的URL格式如下：

protocol://[host.]domain [:port] [/context] [/resource] [?query string **|** path variable]

或者

protocol ://IP Address [:port] [/context] [/resource] [ ?query string \ path variable]

中括号中的内容是可选项。因此，一个最简单的URL是http://yahoo.ca或者是http://192.168.1.9。

需要说明的是，除了输入http://google.com夕卜，还可以用http://173.194.46.35来访问谷歌。 可以用ping命令来获取域名对应的IP地址。

ping google.com

由于IP地址不容易记忆，所以实践中更倾向于使用域名。一台计算机可以托管不止一个 域名，因此，不同的域名可能指向同一个IP。另外，example.com或者example.org无法被注 册，因为它们被保留作为各类文档手册举例使用。

URL中的host部分用来表示在互联网或内网中一个唯一的地址。例如，<http://yahoo.com> (没有host)访问的地址完全不同于<http://mail.yahoo.com> (有host)。多年以来，作为最受欢 迎的主机名，www是默认的主机名。通常，http://www.domainName会被映射到 <http://domainName> 〇

HTTP的默认端口是80端口。因此，对于釆用80端口的Web服务器，无需输入端口号。 有时，Web服务器并未运行在80端口上，此时必须输入相应的端口号。例如，Tomcat服务 器的默认端口号是8080,为了能正确访问服务器，必须提供输入端口号。

<http://localhost:8080/index.html>

localhost作为一个保留关键字，用于指向本机。

URL中的context部分用来代表应用名称，该部分也是可选的。一台Web服务器可以运 行多个上下文（应用），其中一个可以配置为默认上下文。若访问默认上下文中的资源，可以 跳过context部分。

最后，一个context可以有一个或多个默认资源（通常为index.html、index.htm或者 defaulthtm)。一个不带资源名称的URL通常指向默认资源。当存在多个默认资源时，其中最 高优先级的资源将被返回给客户端。

资源名后可以有一个或多个查询语句或者路径参数。查询语句是一个key/value组，多个 查询语句间用“&”符号分隔。路径参数类似于查询语句，但只有value部分，多个value部 分用“/”符号分隔。

接下来，我们介绍HTTP请求和响应。

HTTP请求

一个HTTP请求包含3部分内容。

1. 方法一URI—协议/版本。
2. 请求头信息。
3. 请求正文。

下面为一个HTTP请求示例。

POST /examples/default.j sp HTTP/1.1 Accept: text/plain; text/html Accept-Language:en-gb Connection:Keep-Alive Host:localhost

User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.2.6) Gecko/20100625 Firefox/3.6.6 Content-Length:30

Content-Type: application/x-www-form-urlencoded Accept-Encoding: gzip, deflate

lastName=Blanks&firstName=Mike

请求的第一行 POST /examples/defaultjsp HTTP/1.1 是方法一URI—协议/版本。

请求方法为 POST, URI 为/examples/default.\_jsp，而协议/版本为 HTTP/1.1。

HTTP 1.1规范定义了 7种类型的方法，包括GET、POST、HEAD、OPTIONS、PUT、 DELETE以及TRACE,其中GET和POST广泛应用于互联网。

URI定义了一个互联网资源，通常解析为服务器根目录的相对路径。因此，通常用“/” 符号打头。另外，URL是URI的一个具体类型（详见[http://www.ietf.org/rfc/rfc2396.txt)。](http://www.ietf.org/rfc/rfc2396.txt)%e3%80%82)

HTTP请求包含的请求头信息，包括关于客户端环境以及实体内容等非常有用的信息。 例如，浏览器设置的语言、实体内容长度等。每个header都用回车/换行（即CRLF)分隔。

HTTP请求头信息和请求正文用一行空行分隔，HTTP服务器据此判断请求正文的起始位 置。因此，在一些关于互联网的书籍中，CRLF被作为HTTP请求的第4种组件。

7K例中，请求正文是 lastName=Blanks&firstName=Mike。

在正常的HTTP请求中，请求正文的内容不止如此。

HTTP响应

同HTTP请求一样，HTTP响应也包含3部分。

1. 协议一状态码一描述。
2. 响应头信息。
3. 响应正文。

下面为一个HTTP响应示例。

HTTP/1.1 200 0K

Server: Apache-Coyote/1.1

Date: Thu, 29 Sep 2013 13:13:33 GMT

Content-Type: text/html

Last-Modified: Web, 28 Sep 2013 13:13:12 GMT Content-Length: 112

<html>

<head>

<title>HTTP Response Example</title> </head>

<body>

Welcome to Brainy Software </body>

</html>

类似于HTTP请求报文，HTTP响应报文的第一行说明了 HTTP的版本是1.1，并且请求 结果是成功的（状态代码200为响应成功）。

同HTTP请求报文头信息一样，HTTP响应报文头信息也包含了大量有用的信息。 HTTP响应报文的响应正文是HTML文档。HTTP响应报文的头信息和响应正文也是用 CRLF分隔的。

状态代码200表示Web服务器能正确响应所请求的资源。若一个请求的资源不能被找到 或者理解，则Web服务器将返回不同的状态代码。例如，访问未授权的资源将返回401，而 使用被禁用的请求方法将返回405。完整的HTTP响应状态代码列表详见网址<http://www.w3>. org/Protocols/rfc2616/rfc2616-sec 10.html 〇

Servlet 和 JSP

Java Servlet技术是Java体系中用于开发Web应用的底层技术。1996年，Servlet和JSP 由SUN系统公司发布，以替代CGI技术，作为产生Web动态内容的标准。CGI技术为每一

CPU

的CGI程序。相对于CGI程序，一个Servlet则快多了，这是因为当一个Servlet为响应第一 次请求而创建后，会驻留在内存中，以便响应后续请求。

从Servlet技术出现的那天起，人们开发了大量的Web框架来帮助程序员快速编写 Web应用程序。这些开发框架让开发人员能更关注业务逻辑，减少编写“相似”的代码 片段。尽管如此，开发人员依然需要去理解Servlet技术的基础概念。随后发布的 JavaServer Pages (JSP)技术，是用来帮助简化Servlet开发。尽管实践中会应用一些诸 如Spring MVC、Struct 2或者JSF等强大的开发框架，但如果没有深入理解Servlet和JSP 技术，则无法有效和高效地开发。Servlet是运行在Servlet容器中的Java程序，而Servlet 容器或Servlet引擎相当于一个Web服务器，但是可以产生动态内容，而不仅是静态资 源。Servlet 当前的版本为 3.1，其规范定义可见 JSR (Java Specification Request) 340 ([http://jcp.org/en/jsr/detail?id=340)，基于](http://jcp.org/en/jsr/detail?id=340)%ef%bc%8c%e5%9f%ba%e4%ba%8e) Java 标准版本 6 及以上版本。JSP 2.3 规范定义在 JSR 245 ([http://jcp.org/en/jsr/detail?id=245)。本书假定读者已经了解Java以及面向对象编程技术。](http://jcp.org/en/jsr/detail?id=245)%e3%80%82%e6%9c%ac%e4%b9%a6%e5%81%87%e5%ae%9a%e8%af%bb%e8%80%85%e5%b7%b2%e7%bb%8f%e4%ba%86%e8%a7%a3Java%e4%bb%a5%e5%8f%8a%e9%9d%a2%e5%90%91%e5%af%b9%e8%b1%a1%e7%bc%96%e7%a8%8b%e6%8a%80%e6%9c%af%e3%80%82) 对于 Java 新手，推荐阅读《Java 7: ABeginner’s Tutorial (fourth Edition)》。

一个Servlet是一个Java程序，一个Servlet应用包含了一个或多个Servlet，一个JSP页

面会被翻译并编译成一个Servlet。

一个Servlet应用运行在一个Servlet容器中，它无法独立运行。Servlet容器将来自用户 的请求传递给Servlet应用，并将Servlet应用的响应返回给用户。由于大部分Servlet应用都 会包含一些JSP界面，故称Java Web应用为“Servlet/JSP”应用会更恰当些。

Web用户通过一个诸如正、火狐或者Chrome等Web浏览器来访问Servlet应用。Web 浏览器又称为Web客户端。图1展示了一个典型的Servlet/JSP应用架构。

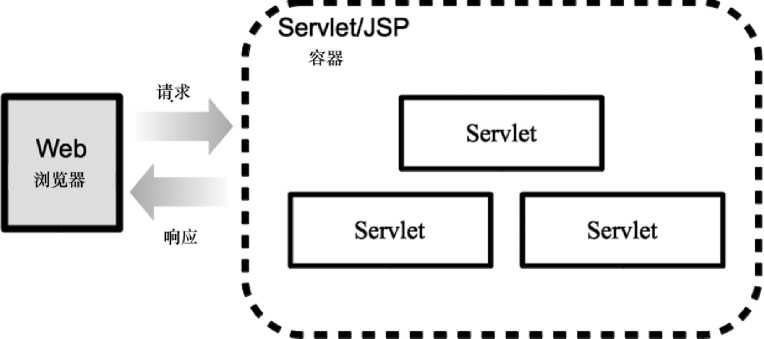


图1 Servlet/JSP应用架构

Web服务端和Web客户端基于HTTP通信。因此，Web服务端往往也称为HTTP服

务端。

一个Servlet/JSP容器是一个能处理Servlet以及静态资源的Web服务端。过去，由于HTTP 服务器更加健壮，人们更愿意将Servlet/JSP容器作为HTTP服务器（如Apache HTTP服务器） 的一个模块来运行，在这种场景下，Servlet/JSP HTTP服务器处理

静态资源。今天，Servlet/JSP容器已经更加成熟，并且被广泛地独立部署。Apache Tomcat和Jetty 作为最流行的Servlet/JSP容器，免费而且开源。下载地址为http://tomcat.apache.org和 http://jetty.codehaus.org。

Servlet和JSP仅是Java企业版众多技术之一，其他Java企业版技术包括JMS、EJB、JSF 和JPA等。Java企业版7 (当前最新版）完整的技术列表见Http://www.oracle.com/technetwork/java/ javaee/tech/index.html 〇

运行一个Java企业版应用需要一个Java企业版容器，常见的容器有GlassFish、JBoss、 Oracle Weblogic以及IBM WebSphere。虽然可以将一个Servlet/JSP应用部署到Java企业版容

器中，但一个Servlet/JSP容器其实就足够了，而且更加轻量级。Tomcat和Jetty不是Java企 业版容器，故它们不能运行EJB或JMS。

下载 Spring 或使用 STS 与 Maven/Gradle

Spring网站建议使用Maven和Gradle来下载Spring库及其依赖包。现在的应用通常有很 多依赖包，并且这些依赖包也有自己的依赖。通过一个依赖管理工具，可以把我们从解析和 下载依赖的工作中解放出来。并且这些工具还可以帮助构建、测试和打包应用。Maven和 Gradle是当下流行的依赖管理系统，使用这两个工具是一个明智的选择。然而当下还有很多 Java 3

程不能要求他们先学习Maven或Gradle，然后才能构建和测试他们的Spring应用程序。因此， 本书所有的例子都有两种发行包：基于Maven的发行包和不依赖Maven/Gradle的发行包。

Maven曾经是Spring支持的唯一的依赖管理系统。Spring开发工具Spring Tool Suites (STS)，是一个基于Eclipse的IDE,捆绑了一个最新的Maven版本。这对于不熟悉Maven

Eclipse,则无须学习Maven就可以应用其好处。然而， 在写作本书的时候，最新版本的STS并不是没有缺点。例如，当使用STS创建Spring MVC 应用程序时，生成的应用程序会因为缺少Servlet部署描述符（web.xml文件）而导致Maven 插件生成一个错误消息。因此开发人员不得不被迫处理一个Maven问题。当然，可以通过简 单地创建一个web.xml文件来解决它。但是，从Servlet 3.0开始，web.xml文件是可选的，并 且可以构建一个没有web.xml文件的Spring MVC应用程序。

下面对于使用Maven/Gradle或不使用工作会分别介绍。

手动下载Spring

如果不想通过Gradle和Maven来构建应用，则应从Spring仓库中下载其类库：

<http://repo.spring.io/release/org/springframework/spring/>

点击以上链接将跳转到最新版本（写作本书时为版本为4.2.5),选择一个zip发行包，发 行包命名格式如下：

spring-framework-x.y.z.RELEASE-dist.zip

其中，x.y.z表示Spring的主要版本和次要版本。

将下载好的zip解压到任意目录。在解压的目录中，包含相应的文档和Java源代码，其 中libs Spring框架开发应用所需的jar文件。

使用 STS 和 Maven/Gradle

如果选择使用Maven或Gradle，贝彳无需手动下载Spring。具体内容，请参考本书附录B。

下载Spring源码

Spring框架是一个开源项目，如果你喜欢冒险或者你想要的尚未发布的最新版本的 Spring，你可以使用Git下载源代码。克隆Spring源代码的命令如下：

git clone git://github.com/spring-projects/spring-framework.git

此外还需使用Gradle来从源代码构建Spring。Git和Gradle工具不是本书的范围。

本书内容简介

第1章：Spnng框架，介绍了最流行的开源框架。

第2章：模型2和MVC模式，讨论了 Spring MVC实现的设计模式。

第3章：Spring MVC介绍，编写了第一个Spring MVC应用。

第4章：基于注解的控制器，讨论了 MVC模式中最重要的一个对象一一控制器。在本 章中，我们将学习如何编写基于注解的控制器，该方式由SpringMVC 2.5版本引入。

第5章：数据绑定和表单标签库，讨论Spring MVC最强大的一个特性，并利用它来展示 表单数据。

第6章：转换器和格式化，讨论了数据绑定的辅助对象类型。

第7章：验证器，展示如何通过验证器来验证用户输入数据。

第8章：表达式语言，介绍了 JSP 2.0中最重要的特性“表达式语言”。该特性的目标是 帮助开发人员编写无脚本的JSP页面，让JSP页面更加简洁而且有效。本章将帮助你学会通 过EL来访问Java Bean和上下文对象。

第9章：JSTL，介绍了 JSP技术中最重要的类库^标准标签库。这是一组帮助处理常见 问题的标签，诸如访问map或集合对象、条件判断、XML处理，以及数据库访问和数据处理。

第10章：国际化，将展示如何用SpringMVC来构建多语言网站。

第11章：上传文件，介绍两种不同的方式来处理文件上传。

第12章：下载文件，介绍如何用编程方式向客户端端传输一个资源。

第13章：应用测试，介绍如何使用Junit、Mockito和Spring MVC Test进行单兀测试和 集成测试。

附录A: Tomcat，介绍如何安装和配置Tomcat。

附录B: Spring Tool Suite和Maven，介绍如何安装并配置Spring Tool Suite，并且开发 和运行Spring MVC应用。

本书内容简介

附录C: Servlet,介绍了 Servlet API并展示几个简单的Servlet应用。本附录重点关注Servlet API 的 4 个包中的两个，艮P javax.servlet 和 javax.servlet.http。

附录D: JavaServer Pages，介绍了 JSP语法，包括指令、脚本元素和actions。

附录E:部署描述符，介绍如何配置Spring MVC应用以便部署。

下载示例应用

本书所有的示例应用压缩包可以通过以下地址下载：

http: //books .brainys〇ftware . com/download 和 http: //www. epubit • com. cn。

目录

第1章Spring框架 1

1. [XML配置文件 3](#bookmark21)
2. [Spring控制反转容器的使用 4](#bookmark28)

1.2.1通过构造器创建一个bean

錢 4

1.2.2通过工厂方法创建一■个bean

\_ 5

[1.2.3销毁方法的使用 6](#bookmark43)

[1.2.4向构造器传递参数 6](#bookmark45)

[1.2.5 Setter方式依赖注入 7](#bookmark52)

1.2.6构造器方式依赖注入……10

1. 雜 10

[第2章模型2和MVC模式 11](#bookmark61)

[2.1模型1介绍 11](#bookmark62)

[2.2模型2介绍 11](#bookmark63)

[2.3模型2之Servlet控制器 13](#bookmark64)

1. [Product 类 14](#bookmark65)
2. [ProductForm 类 15](#bookmark66)
3. [ControllerServlet 类 16](#bookmark72)
4. [Action 类 20](#bookmark81)
5. 视图 20

[2.3.6测试应用 22](#bookmark94)

[2.4模型2之Filter分发器 22](#bookmark95)

1. [&验器 25](#bookmark101)

[2.6依赖注入 31](#bookmark117)

1. /鳩 38

第 3 章 Spring MVC 介绍 39

[3.1采用Spring MVC的好处 39](#bookmark128)

1. Spring MVC 的

D ispatcherServlet 40

1. [Controller 接口 41](#bookmark130)
2. [第一个 Spring MVC 应用 42](#bookmark131)
3. [目录结构 42](#bookmark132)

3.4.2部署描述符文件和Spring

MVC配置文件 43

1. [Controller 类 44](#bookmark136)
2. [View 类 46](#bookmark137)
3. [测试应用 47](#bookmark142)

[3.5视图解析器 47](#bookmark145)

1. ，鳩 49

[第4章基于注解的控制器 50](#bookmark150)

1. [Spring MVC 注解类型 50](#bookmark151)
2. [Controller 注解类型 50](#bookmark152)
3. RequestMapping 注解

麵 51

[4.2编写请求处理方法 54](#bookmark167)

[4.3应用基于注解的控制器 56](#bookmark171)

1. [目录结构 56](#bookmark172)

[4.3.2配置文件 56](#bookmark173)

1. [Controller 类 59](#bookmark176)
2. [View 60](#bookmark180)
3. [测试应用 61](#bookmark182)

4.4应用@八世〇评^3和@呂61^以

浙讎注A 62

[4.5重定向和Flash属性 66](#bookmark196)

[4.6请求参数和路径变量 67](#bookmark197)

1. [@ModelAttribute 69](#bookmark199)
2. /鳩 70

第5章数据绑定和表单标签库. . .+71

[5.1数据绑定概览 71](#bookmark203)

[5.2表单标签库 72](#bookmark206)

[5.2.1表单标签 73](#bookmark210)

1. [input 标签 74](#bookmark211)
2. [password 标签 74](#bookmark212)
3. [hidden 标签 75](#bookmark215)
4. [textarea 标签 75](#bookmark219)
5. [checkbox 标签 76](#bookmark222)
6. [radiobutton 标签 76](#bookmark224)
7. [checkboxes 标签 77](#bookmark227)
8. [radiobuttons 标签 78](#bookmark231)
9. [select 标签 78](#bookmark235)
10. [option 标签 79](#bookmark240)
11. [options 标签 80](#bookmark246)
12. [errors 标签 80](#bookmark250)

[5.3数据绑定范例 81](#bookmark255)

[5.3.1目录结构 81](#bookmark256)

1. [Domain 类 81](#bookmark257)
2. [Service 类 84](#bookmark262)
3. [配置文件 87](#bookmark264)
4. 视 SI 88

[5.3.7测试应用 91](#bookmark277)

5.4 4、结 92

[第6章转换器和格式化 93](#bookmark282)

* 1. [Converter 93](#bookmark283)
  2. [Formatter 98](#bookmark293)
  3. [用 Registrar 注册 Formatter 101](#bookmark300)
  4. 选择 Converter，还是

Formatter 103

* 1. 103

[第7章验证器 104](#bookmark308)

[7.1验证概览 104](#bookmark309)

1. [Spring 验证器 105](#bookmark310)
2. [ValidationUtils 类 106](#bookmark311)
3. [Spring 的 Validator 范例 107](#bookmark312)

[7.5源文件 109](#bookmark317)

1. [Controller 类 109](#bookmark320)

[7.7测试验证器 111](#bookmark326)

1. [JSR 303 验证 112](#bookmark328)
2. [JSR 303 Validator 范例 113](#bookmark335)
3. [小结 116](#bookmark342)

[第8章表达式语言 117](#bookmark343)

[8.1表达式语言简史 117](#bookmark344)

[8.2表达式语言的语法 118](#bookmark345)

1. [关键字 118](#bookmark346)
2. Controller 类 83 8.2.2 []和运算符 119

[8.2.3取值规贝l] 119](#bookmark348)

1. [访问 JavaBean 120](#bookmark349)
2. [EL隐式对象 121](#bookmark350)
3. [pageContext 121](#bookmark353)
4. [initParam 123](#bookmark358)
5. [param 123](#bookmark359)
6. [paramValues 123](#bookmark360)
7. [header 123](#bookmark361)
8. [headerValues 124](#bookmark362)
9. [cookie 124](#bookmark363)
10. applicationScope、

sessionScope、requestScope

和 pageScope 124

[8.5使用其他EL运算符 124](#bookmark365)

1. [算 g 算符 125](#bookmark366)

[8.5.2关系运算符 125](#bookmark367)

[8.5.3逻辑运算符 126](#bookmark368)

[8.5.4条件运算符 126](#bookmark369)

1. [empty 运算符 126](#bookmark370)

[8.5.6字符串连接运算符 126](#bookmark371)

[8.5.7分号操作符 127](#bookmark372)

8.6引用静态属性和静态方法……127

1. 创建 Set、List 和 Map 128

8.8访问列表元素和Map条目……129 8.9操作集合 129

1. [toList 129](#bookmark382)
2. [toArray 130](#bookmark383)
3. [limit 130](#bookmark384)
4. [sort 130](#bookmark385)
5. [average 130](#bookmark386)
6. [sum 131](#bookmark387)
7. [count 131](#bookmark388)
8. [min 131](#bookmark389)
9. [max 131](#bookmark390)
10. [map 131](#bookmark391)
11. [filter 132](#bookmark392)
12. [forEach 132](#bookmark393)

[8.10格式化集合 132](#bookmark394)

1. [使用 HTML 注释 132](#bookmark395)
2. [使用 String.join() 134](#bookmark405)

[8.11格式化数字 134](#bookmark406)

[8.12格式化日期 134](#bookmark407)

8.13如何在JSP 2.0及其更髙版本中

I2SEL 135

8.13.1实现免脚本的JSP

M 135

[8.13.2禁用EL计算 135](#bookmark410)

[8.14 小结 137](#bookmark411)

第 9 章 JSTL 138

1. [下载 JSTL 138](#bookmark413)
2. [JSTL 库 138](#bookmark414)
3. [—般行为 140](#bookmark415)
4. [out 标签 140](#bookmark416)
5. [url 标签 141](#bookmark417)
6. [set 标签 144](#bookmark427)
7. [remove 标签 145](#bookmark433)

[9.4条件行为 146](#bookmark437)

1. [if 标签 146](#bookmark438)
2. choose、when 和 otherwise

标签 147

[9.5 遍历行为 148](#bookmark440)

1. [forEach 标签 148](#bookmark441)
2. [forTokens 标签 157](#bookmark466)

[9.6格式化行为 158](#bookmark471)

1. [formatNumber 标签 158](#bookmark473)
2. [formatDate 标签 160](#bookmark480)
3. [timeZone 标签 162](#bookmark484)
4. [setTimeZone 标签 163](#bookmark489)
5. [parseNumber 标签 163](#bookmark491)
6. [parseDate 标签 165](#bookmark493)
7. MWi 166
8. [contains 函数 166](#bookmark502)
9. containsIgnoreCase

函数 166

1. [endsWith 函数 167](#bookmark508)
2. [escapeXml 函数 167](#bookmark511)
3. [indexOf 函数 167](#bookmark514)
4. [join 函数 167](#bookmark517)
5. [length 函数 168](#bookmark519)
6. [replace 函数 168](#bookmark520)
7. [split 函数 168](#bookmark521)
8. [startsWith 函数 169](#bookmark522)
9. [substring 函数 169](#bookmark525)
10. [substringAfter 函数 169](#bookmark528)
11. substringBefore 函数…..169
12. [toLowerCase 函数 170](#bookmark534)
13. [toUpperCase 函数 170](#bookmark535)
14. [trim 函数 170](#bookmark536)
15. /J、@ 170

[第10章国际化 171](#bookmark537)

1. [语言区域 172](#bookmark538) 10.2国际化Spring MVC应用程序…173

10.2.1将文本组件隔离成

H生文件 174

10.2.2选择和读取正确的

H生文件 175

10.3告诉Spring MVC使用哪个

i吾W關 176

1. [使用 message 标签 177](#bookmark548)
2. [范例 177](#bookmark551)
3. [小结 181](#bookmark558)

第11章上传文件 182

[11.1客户端编程 182](#bookmark561)

1. [MultipartFile 接口 183](#bookmark562)
2. 用 Commons FileUpload

±歡{牛 184

1. [Domain 类 185](#bookmark564)
2. [控制器 185](#bookmark566)

[11.6配置文件 187](#bookmark568)

1. [JSP 页面 188](#bookmark571)

[11.8应用程序的测试 190](#bookmark576)

11.9用Servlet 3及其更髙版本

±舦彳牛 191

[11.10客户端上传 194](#bookmark584)

1. [小结 202](#bookmark595)

[第12章下载文件 203](#bookmark598)

[12.1文件下载概览 203](#bookmark599)

[12.2范例1:隐藏资源 204](#bookmark600)

[12.3范例2:防止交叉引用 207](#bookmark604)

1. [小结 210](#bookmark610)

[第13章应用测试 211](#bookmark611)

1. [单元测试 211](#bookmark612)

[13.2状态测试与行为测试 213](#bookmark613)

[13.3 应用 JUnit 213](#bookmark614)

13.3.1开发一个单元测试……213 13.3.2运行一个单元测试……215 13.3.3通过测试套件来运行全部

或多个单元测试 215

13.4应用测试挡板

(Test Doubles) 216

1. [dummy 217](#bookmark622)
2. [stub 219](#bookmark625)
3. [spy 219](#bookmark628)
4. [fake 221](#bookmark637)
5. [mock 224](#bookmark639)
6. 对 Spring MVC Controller

226

1. MockHttpServletRequest 和

MockHttpServlet Response 226

1. ModelAndViewAssert.…229
2. 应用 Spring MVC Test 进行

\_廳 232

1. [API 233](#bookmark662)
2. Spring MVC 测试类的

附录 A Tomcat 242

1. [1下载和配置Tomcat 242](#bookmark689)
2. [2启动和终止Tomcat 243](#bookmark690)

[A.3定义上下文 243](#bookmark691)

[A.4定义资源 244](#bookmark692)

1. [5安装TLS证书 245](#bookmark693)

附录 B Spring Tool Suite 和 Maven ... 246

1. [1 安装 STS 246](#bookmark695)
2. 2创建一个Spring MVC应用..…247
3. [3 选择Java版本 251](#bookmark708)
4. [4 创建 index.html 文件 252](#bookmark710)

[B.5更新项目 253](#bookmark713)

1. [6运行应用 253](#bookmark716)

[附录 C Servlet 256](#bookmark729)

1. [1 Servlet API 概览 256](#bookmark728)
2. 2 Servlet 257

C.3编写基础的Servlet应用程序…258

1. 3.1编写和编译Servlet类. 259
2. [3.2应用程序目录结构 260](#bookmark734)
3. [3.3 调用 Servlet 261](#bookmark735)
4. 4 ServletRequest 262
5. [5 ServletResponse262](#bookmark736)
6. [6 ServletConfig263](#bookmark737)

絲 234

1. [示例 236](#bookmark676)

13.7修改集成测试中Web

娜径 239

1. [小结 241](#bookmark687)
2. [7 ServletContext266](#bookmark742)
3. [8 GenericServlet 266](#bookmark743)
4. [9 Http Servlets 268](#bookmark751)
5. [9.1 HttpServlet 269](#bookmark752)
6. [9.2 HttpServletResponse 271](#bookmark753)
7. [10处理HTML表单 271](#bookmark754)
8. [11使用部署描述符 277](#bookmark766)
9. [12 小结 280](#bookmark770)

附录 D JavaServer Pages 281

1. [1 JSP 概述 281](#bookmark772)
2. [2 注释 286](#bookmark781)
3. [3 隐式对象 287](#bookmark785)
4. 4 290
5. [4.1 page 指令 290](#bookmark795)
6. [4.2 include 指令 292](#bookmark798)
7. [5脚本元素 293](#bookmark806)
8. [5.1 表达式 294](#bookmark809)
9. [5.2 声明 294](#bookmark810)

[D.5.3禁用脚本元素 298](#bookmark817)

1. 6 谢乍 298
2. [6.1 useBean 298](#bookmark819)
3. 6.2 setProperty 和

getProperty 299

1. [6.3 include 300](#bookmark826)
2. [6.4 forward 301](#bookmark829)
3. [7错误处理 301](#bookmark830)
4. 8 ，鳩 302

[附录E部署描述符 303](#bookmark832)

1. [1 概述 303](#bookmark833)

[E.1.1 核心元素 305](#bookmark834)

1. [1.2 context-param 305](#bookmark835)
2. [1.3 distributable 306](#bookmark836)
3. [1.4 error-page 306](#bookmark837)
4. [1.5 filter 306](#bookmark841)
5. [1.6 filter-mapping 307](#bookmark843)
6. [1.7 listener 308](#bookmark844)
7. 1.8 locale-encoding-mapping-list

和 locale-encoding­mapping 308

1. [1.9 login-config 308](#bookmark846)
2. [1.10 mime-mapping 309](#bookmark847)
3. [1.11 security-constraint 309](#bookmark848)
4. [1.12 security-role 310](#bookmark849)
5. [1.13 Servlet 311](#bookmark850)
6. [1.14 servlet-mapping 313](#bookmark851)
7. [1.15 session-config 313](#bookmark854)
8. [1.16 welcome-file-list 313](#bookmark860)

E.1.17 JSP-Specific Elements…314

[E.1.18 taglib 314](#bookmark863)

[E.1.19 jsp-property-group 315](#bookmark864)

E.2 316

[E.3 Web fragment 317](#bookmark866)

E.4 ，J、g 319

第1章

Spring框架

Spring框架是一个开源的企业应用开发框架，作为一个轻量级的解决方案，它包含20多 个不同的模块。本书主要关注Core、Spring Bean、Spring MVC和Spring MVC Test模块。

本章主要介绍Core和Spring Bean这两个模块，以及它们如何提供依赖注入解决方案。 为方便初学者，本书会深入讨论依赖注入概念的细节。后续介绍开发MVC应用的章节将会 使用到本章介绍的技能。

依赖注入

在过去数年间，依赖注入技术作为代码可测试性的一个解决方案已经被广泛应用。实际 上，Spring、谷歌Guice等框架都采用了依赖注入技术。那么，什么是依赖注入技术？

很多人在使用中并不区分依赖注入和控制反转（I〇C)，尽管Martin Fowler在其文章中已 分析了两者的不同：

<http://martinfowler.com/articles/injection.html>

简单来说，依赖注入的情况如下。

有两个组件A和B，A依赖于B。假定A是一个类，且A有一个方法importantMethod用到 了 B，如下：

public class A {

public void importantMethod() {

B b = ... II get an instance of B b.usefulMethod();



要使用B，类A必须先获得组件B的实例引用。若B是一个具体类，则可通过new关键 字直接创建组件B实例。但是，如果B是接口，且有多个实现，则问题就变得复杂了。我们 固然可以任意选择接口 B的一个实现类，但这也意味着A的可重用性大大降低了，因为无法 采用B的其他实现。

依赖注入是这样处理此类情景的：接管对象的创建工作，并将该对象的引用注入需要该对象的 组件。以上述情况为例，依赖注入框架会分别创建对象A和对象B，将对象B注入到对象A中。

为了能让框架进行依赖注入，程序员需要编写特定的set方法或者构建方法。例如，为了 能将B注入到A中，类A会被修改成如下形式：

public class A { private B b;

public void importantMethod() {

// no need to worry about creating B anymore // B b **=.…//** get an instance of B b.usefulMethod();

public void setB(B b) { this.b = b;

}

}

修改后的类A新增了一个set方法，该方法将会被框架调用，以注入B的一个实例。由 于对象依赖由依赖注入，类A的importantMethod方法不再需要在调用B的useMMethod方 法前去创建B的一个实例。

当然，也可以采用构造器方式注入，如下所示：

public class A { private B b;

public A(B b) { this.b = b;

}

public void importantMethod() {

// no need to worry about creating B anymore // B b = ... II get an instance of B b.usefulMethodO ;

本例中，Spring会先创建B的实例，再创建实例A，然后把B注入到实例A中。

注：

Spring管理的对象称为beans。

通过提供一个控制反转容器（或者依赖注入容器)，Spring为我们提供一种可以“聪明” 地管理Java对象依赖关系的方法。其优雅之处在于，程序员无需了解Spring框架的存在，更 不需要引入任何Spring类型。

从1.0版本开始，Spring就同时支持setter和构造器方式的依赖注入。从2.5版本开始， 通过Autowired注解，Spring支持基于field方式的依赖注入，但缺点是程序必须引入 org.springframework.beans.factory.annotation.Aiitowired，这对 Spring 产生了依赖，这样，程序

无法直接迁移到另一个依赖注入容器间。

使用Spring，程序几乎将所有重要对象的创建工作移交给Spring，并配置如何注入依赖。 Spring支持XML或注解两种配置方式。此外，还需要创建一个ApplicationContext对象，代 表一个 Spring 控制反转容器，org.springframework.context.ApplicationContext 接口 有多个实现， 包括 ClassPathXmlApplicationContext 和 FileSystemXmlApplicationContext。这两个实现都需要 至少一个包含beans信息的XML文件。ClassPathXmlApplicationContext尝试在类加载路径中 加载配置文件，而FileSystemXmlApplicationContext则从文件系统中加载。

下面是从类路径中加载configl.xml和config2.xml的ApplicationContext创建的一个代码示例。

ApplicationContext context = new ClassPathXmlApplicationContext( new String [ ] { "configl .xml1’，"conf ig2 .xml" });

可以通过调用ApplicationContext的getBean方法获得对象。

Product product = context.getBean("product", Product.class);

getBean方法会查询id为product且类型为Product的bean对象。

注=

理想情况下，我们只需在测试代码中创建一个ApplicationContext，应用程序本身无需处理。 对于Spring MVC应用，可以通过一个Spring Servlet来处理ApplicationContext，而无需直接处理。

1. XML配置文件

从1.0版本开始，Spring就支持基于XML的配置；从2.5版本开始，増加了通过注解的

配置支持。下面介绍如何配置XML文件。配置文件的根元素通常为beans:

<?xml version="l. 0" encoding="UTF-8f, ?>

<beans xmlns="<http://www.springframework.org/schema/beansM>xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instancen>xsi : schemaLocation=,,http: //www. springf ramework. org/schema/beans http: //www. springframework.org/schema/beans/spring-beans .xsdT,>

</beans>

如果需要更强的Spring配置能力，可以在schema location属性中添加相应的schema，也 可以指定schema版本：

<http://www.springframework.org/schema/beans/spring-beans-4.2.xsd>

不过，笔者推荐使用默认schema,以便升级spring库时无需修改配置文件。

配置文件既可以是一份，也可以分解为多份，以支持模块化配置。ApplicationContext的实现 类支持读取多份配置文件。另一种选择是，通过一份主配置文件，将该文件导入到其他配置文件。

下面是导入其他配置文件的一个示例：

<?xml version="l. 0" encoding="UTF-8f, ?>

<beans xmlns=,,http: //www. springf ramework. org/schema/beans" xmlns:xsi=M<http://www.w3.org/2001/XMLSchema-instanceM>xsi : schemaLocation=,,http: //www. springframework.org/schema/beans http: //www. springf ramework. org/schema/beans/spring-beans .xsdT,>

<import resource="configl.xmlM/>

<import resource="module2/config2.xml"/> <import resource="/resources/config3.xmlM/>

</beans>

bean元素的配置后面将会详细介绍。

1. Spring控制反转容器的使用

本节主要介绍Spring如何管理bean和依赖关系。

1.2.1通过构造器创建一个bean实例

前面已经介绍，通过调用ApplicationContext的getBean方法可以获取一个bean的实例。

下面的配置文件中定义了一个名为product的bean (见清单1.1)。

清单1.1 一个简单的配置文件

<?xml version="l.0M encoding="UTF-8"?>

<beans xmlns=Mhttp: //www. springf ramework. org/schema/beans11 xmlns : xsi=’’http: //[www.w3](http://www.w3) • org/2001/XMLSchema-instance" xsi:schemaLocation="<http://www.springframework.org/schema/beans> <http://www.springframework.org/schema/beans/spring-beans.xsdn>>

<bean name=l'product11 class=Mspringintro .bean. ProductM/>

</beans>

该bean的定义告诉Spring，通过默认无参的构造器来初始化Product类。如果不存在该 构造器（如果类的编写者重载了构造器，且没有显示声明默认构造器)，则Spring将拋出一个 异常。此外，该无参数的构造器并不要求是public签名。

注意，应采用id或者name属性标识一个bean。为了让Spring创建一个Product实例，

应将bean定义的name值“product”（具体实践中也可以是id值）和Product类型作为参数传

递给 ApplicationContext 的 getfiean 方法。

ApplicationContext context =

new ClassPathXmlApplicationContext( new String[] {Mspring-config.xml"});

Product productl = context .getBean ("product11, Product. class); productl. setName ("Excellent snake oil11);

System.out.printIn("productl: M + productl.getName{));

1.2.2通过工厂方法创建一个bean实例

大部分类可以通过构造器来实例化。然而，Spring还同样支持通过调用一个工厂的方法

来初始化类。

下面的bean定义展示了通过工厂方法来实例化java.time.LocalDate。

<bean id=nlocalDate’’ class="j ava ■ time ■ LocalDate" factory-method=,,nowll/>

本例中采用了 id属性而非name属性来标识bean，采用了 getBean方法来获取LocalDate

实例。

ApplicationContext context =

new ClassPathXmlApplicationContext(

new String[] {"spring-config.xml"});

LocalDate localDate = context.getBean("localDate", LocalDate.class);

1.2.3销毁方法的使用

有时，我们希望一些类在被销毁前能执行一些方法。Spring考虑到了这样的需求。可以 在bean定义中配置destroy-method属性，来指定在销毁前要执行的方法。

下面的例子中，我们配置Spring通过java.util.concurrent.Executors的静态方法newCached ThreadPool 来创建一个 java.uitl.concurrent.ExecutorService 实例，并指定了 destroy-method 属 性值为shutdown方法。这样，Spring会在销毁ExecutorService实例前调用其shutdown方法。

<bean id="executorService" class=Mjava.util.concurrent.Executors" factory-method=,,newCachedThreadPoolM destroy-method=Mshutdown"/>

1.2.4向构造器传递参数

Spring支持通过带参数的构造器来初始化类（见清单1.2)。

清单 1.2 Product 类

package springintro.bean; import j ava.io.Serializable;

public class Product implements Serializable {

private static final long serialVersionUID = 748392348L; private String name; private String description; private float price;

public Product() {

public Product(String name, String description, float price) { this.name = name; this.description = description; this.price = price;

}

public String getName(> { return name;

}

public void setName(String name) { this.name = name;

public String getDescription() {

return description;

}

public void setDescription(String description) { this.description = description;

}

public float getPrice() {

return price;

}

public void setPrice(float price) { this.price = price;

以下的定义展示了如何通过参数名传递参数。

<bean name=MfeaturedProductM class="springintro.bean.ProductM> <constructor-arg name=MnameM value=’’Ultimate Olive OilM/> <constructor-arg name=Mdescription11

value="The purest olive oil on the market"/> <constructor-arg name=MpriceM value=M9.95M/>

</bean>

这样，在创建Product实例时，Spring会调用如下构造器：

public Product(String name, String description, float price) { this.name = name; this.description = description; this.price = price;

}

除了通过名称传递参数外，Spring还支持通过指数方式来传递参数，具体如下:

<bean name=MfeaturedProduct2" class=nspringintro.bean.Productn> <constructor-arg index="On value=,Ultimate Olive OilM/> <constructor-arg index="l"

value="The purest olive oil on the market"/> <constructor-arg index=,,2M value=M 9.95" / >

</bean>

需要说明的是，采用这种方式，对应构造器的所有参数必须传递，缺一不可。

1. Setter方式依赖注入

下面以Employee类和Address类为例，介绍setter方式依赖注入(见清单1.3和清单1.4)。

清单 1.3 Employee 类

package springintro.bean;

public class Employee {

private String firstName;

private String lastName;

private Address homeAddress;

public Employee() {

}

public Employee(String firstName, String lastName, Address homeAddress) { this.firstName = firstName; this.lastName = lastName; this.homeAddress = homeAddress;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) { this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) { this.lastName = lastName;

public Address getHomeAddress() { return homeAddress;

}

public void setHomeAddress(Address homeAddress) { this.homeAddress = homeAddress;

}

0〇verride

public String toStringO {

return firstName + " " + lastName

+ "\n" + homeAddress;

清单1.4 Address类

package springintro.bean;

public class Address { private String linel; private String line2; private String city; private String state; private String zipCode; private String country;

public Address(String linel, String line2, String city, String state, String zipCode, String country) { this.linel = linel; this.Iine2 = line2; this■city = city; this.state = state; this.zipCode = zipCode; this.country = country;

// getters and setters omitted 0〇verride

public String toStringO {  
return linel + "\n"

+ line2 + M\nM + city + M\n"

+ state + + zipCode + "\nn

+ country;

}

}

Employee依赖于Address类，可以通过如下配置来保证每个Employee实例都能包含 Address 实例。

<bean name=MsimpleAddressM class=Mspringintro.bean.AddressM> <constructor-arg name=Mlinel" value=M151 Corner StreetM/> <constructor-arg name=Mline2M value=M"/>

<constructor-arg name=McityM value=MAlbanyM/>

<constructor-arg name=”state” value=MNYn/>

<constructor-arg name=”zipCode" value="99999"/>

<constructor-arg name=Mcountry" value=nUS"/> </bean>

<bean name=Memployeeln class=,,springintro.bean.Employee11 >

<property name="homeAddressM ref= Msimp1eAddressM/>

<property name="firstName" value=MJunior"/>

〈property name=’’lastName" value=,,Moore"/>

</bean>

simpleAddress对象是Address类的一个实例，它通过构造器方式实例化。employeel对象 则通过配置property元素来调用setter方法以设置值。需要注意的是，homeAddress属性配置 的是simpleAddress对象的引用。

被引用对象的配置定义无需早于引用其对象的定义。在本例中，employeel对象可以出现 在simpleAddress对象定义之前。

1.2.6构造器方式依赖注入

清单1.3所示的Employee类提供了一个可以传递参数的构造器，我们还可以将Address 对象通过构造器注入，如下所示：

<bean name=,'employee2" class=,,springintro.bean.Employee">

<constructor-arg name="firstName" value=MSenior"/>

<constructor-arg name="lastName" value="Moore"/>

<constructor-arg name=”homeAddress" ref="simpleAddress"/〉

</bean>

<bean name=MsimpleAddress" class=Mspringintro.bean.Address"> <constructor-arg name=’’linel" value=’’151 Corner Street"/> <constructor-arg name=’’line2" value=’"’/>

<constructor-arg name=,,cityM value=MAlbany"/>

<constructor-arg name=,,stateM value="NY"/>

<constructor-arg name="zipCoden value=”99999”/>

<constructor-arg name="countryn value=nUSM/>

</bean>

1.3小结

本章学习了依赖注入的概念以及基于Spring容器的实践，后续将在此基础之上配置Spring

应用。

第2章

模型2和MVC模式

Java Web应用开发中有两种设计模型，为了方便，分别称为模型1和模型2。模型1是 以页面中心，适合于小应用开发。而模型2基于MVC模式是Java Web应用的推荐架构（简 单类型的应用除外）。

本章将会讨论模型2,并展示4个不同示例应用。第一个应用是一个基本的模型2应用， 采用Servlet作为控制器；第二个应用也是模型2应用，但采用了 Filter作为控制器；第三个 应用引入了验证控件来校验用户的输入；最后一个应用则采用了一个自研的依赖注入器。在 实践中，应替换为Spring。

注

在写作本书时，业界正致力于MVC Web框架的标准化（见JSR 371)。

2.1模型1介绍

第一次学习JSP，通常通过链接方式进行JSP页面间的跳转。这种方式非常直接，但在 中型和大型应用中，这种方式会带来维护上的问题。修改一个JSP页面的名字，会导致页面 中大量的链接需要修正。因此，实践中并不推荐模型1 (但仅2~3个页面的应用除外)。

2.2模型2介绍

模型2基于模型一视图一控制器（MVC)模式，该模式是Smalltalk-80用户交互的核心 概念，那时还没有设计模式的说法，当时称为MVC范式。

一个实现MVC模式的应用包含模型、视图和控制器3个模块。视图负责应用的展示。模

型封装了应用的数据和业务逻辑。控制器负责接收用户输入，改变模型以及调整视图的显示。

模型2中，Servlet或者Filter都可以充当控制器。几乎所有现代Web框架都是模型2的 实现。Struts 1、Spring MVC 和 JavaServer Faces 使用一个 Servlet 作为控制器，而 Struts 2 则 使用一个Filter作为控制器。大部分都采用JSP页面作为应用的视图，当然也有其他技术。而 模型则采用POJO (Plain Old Java Object)。不同于EJB等特定对象，POJO是一个普通对象。 实践中会采用一个JavaBean来持有模型状态，并将业务逻辑放到一个Action类中。

图2.1展示了一个模型2应用的架构图。

/Web容器

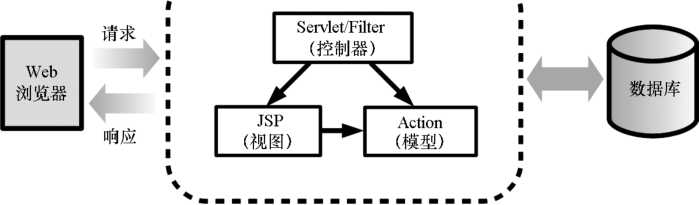


图2.1模型2架构图

每个HTTP请求都发送给控制器，请求中的URI标识出对应的action。action代表了应用 可以执行的一个操作。一个提供了 action的Java对象称为action对象。一个action类可以支 持多个 action (在 Spring MVC 以及 Struts 2 中），或者一个 action (在 Struts 1 中。

看似简单的操作可能需要多个action。如向数据库添加一个产品，需要两个action。

1. 显示一个“添加产品”的表单，以便用户能输入产品信息。
2. 将表单信息保存到数据库中。

如前所述，我们需要通过URI方式告诉控制器执行相应的action。例如，通过发送类似 如下的URI，来显示“添加产品”表单。

<http://domain/appName/input-product>

通过类似如下的URI，来保存产品。

<http://domain/appName/save-product>

控制器会解析URI并调用相应的action，然后将模型对象放到视图可以访问的区域（以 便服务端数据可以展示在浏览器上)。最后，控制器利用RequestDispatcher或者HttpServlet Response.sendRedirect〇方法跳转到视图（JSP页面或者其他资源)。在JSP页面中，用表达式语言以及定制标签显示数据。

注意

调用 RequestDispatcher.forward 方法或者 HttpServletResponse.sendRedirect()方法并不会 停止执行剩余的代码。因此，若forward方法不是最后一行代码，则应显式地返回。

if (action.equals(...)) {

RequestDispatcher rd = request.getRequestDispatcher(dispatchUrl); rd.forward(request, response);

return;//explicitly return. Or else, the code below will be executed

}

//do something else

大多数时候，你将使用RequestDispatcher转发到视图，因为它比sendRedirect更快响应。

这是因为重定向导致服务器向浏览器发送状态代码为302的HTTP响应，并包含新URL。而 浏览器在接收到状态代码302时，根据响应头部中找到的URL向服务器发出新的HTTP请求。 换句话说，重定向需要一个往返，这使其慢于转发。

使用重定向超过转发的优势是什么？通过重定向，你可以将浏览器定向到其他应用程 序，这是转发不能支持的。如果重定向用于在同一应用程序中不同的资源，由于使用与原始 请求URL不同的URL，若用户在响应后意外地按下浏览器的重新加载/刷新按钮，则与原始 请求URL相关联的代码将不会再次执行。例如，你不希望因为用户意外按下她的浏览器的重 新加载或刷新按钮，而导致重新执行诸如信用卡付款的代码。

本章最后一个例子是appdesign4应用程序，它显示了一个重定向的例子。

2.3模型2之Servlet控制器

为了便于对模型2有一个直观的了解，本节将展示一个简单模型2应用。实践中，模型2 的应用非常复杂。

示例应用名为appdesignl，其功能设定为输入一个产品信息。具体为：用户填写产品表单（图 2.2)并提交；示例应用保存产品并展示一个完成页面，显示已保存的产品信息（见图2.3)。

示例应用支持如下两个action。

1. 展示“添加产品”表单。该action将图2.2中的输入表单发送到浏览器上，其对应的 URI应包含字符串input-product。
2. 保存产品并返回如图2.3所示的完成页面，对应的URI必须包含字符串save-product。

Details;

Product Name: Abracadabra Bluray Description: Low cost blur ay player Price: $26.99



图2.2产品表单



G D localhost:8080/appdesign1/save-product ^ 〇 E

The product has been saved.

图2.3产品详细页

示例应用由如下组件构成：

^ webapp T ^css

Jf main.css ，&jsp

lii ProductDetails.jsp ProductFormjsp T WEB-INF T classes ▼ &appdesign1 T {a\* action

SaveProductAction.class T fe- controller

ControUerServietdass ▼色 form

^ ProductForm.class ▼ b model

品 Product.dass

图2.4 app02a目录结构

1. 一个Product类，作为product的领域对象。
2. —个ProductFonn类，封装了 HTML表单的

输入项。

(3 ) —个ControlleiServlet类，本示例应^的控制器。

1. 一个 SaveProductAction 类。
2. 两个 JSP 页面（ProductFormjsp 和 Product Detail.jsp)作为视图。
3. —个CSS文件，定义了两个JSP页面的显示 风格。

示例应用目录结构如图2.4所示。

下面详细介绍示例应用的每个组件。

1. Product 类

Product实例是一个封装了产品信息的JavaBean。Product类（见清单2.1)包含3个属性: productName% description price〇

清单 2.1 Product 类

package appdesignl.model; import java\*io.Serializable;

import java.math.BigDecimal;

public class Product implements Serializable {

private static final long serialVersionUID = 748392348L; private String name; private String description; private BigDecimal price;

public String getName() { return name;

}

public void setName(String name) { this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) { this.description = description;

}

public BigDecimal getPrice<) {

return price;

}

public void setPrice(BigDecimal price) { this.price = price;

}

}

Product类实现了 java.io.Serializable接口，其实例可以安全地将数据保存到HttpSession 中。根据 Serializable 的要求，Product 实现了一个 serialVersionUID 属性。

1. ProductForm 类

表单类与HTML表单相映射，是后者在服务端的代表。ProductForm类（见清单2.2)包 含了一个产品的字符串值。ProductForm类看上去同Product类相似，这就引出一个问题： ProductForm类是否有存在的必要。

实际上，表单对象会传递ServletRequest给其他组件，类似Validator (本章后面会介绍)。 而ServletRequest是一个Servlet层的对象，不应当暴露给应用的其他层。

另一个原因是，当数据校验失败时，表单对象将用于保存和展示用户在原始表单上的输 入。2.5节将会详细介绍应如何处理。

注意：

大部分情况下，一个表单类不需要实现Serializable接口，因为表单对象很少保存在 HttpSession 中。

清单 2.2 ProductForm 类

package appdesignl.form; public class ProductForm {

private String name;

private String description;

private String price;

public String getNameO { return name;

}

public void setName(String name) { this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) { this.description = description;

}

public String getPrice() { return price;

}

public void setPrice(String price) { this.price = price;

1. ControllerServlet 类

ControllerServlet 类(见清单 2.3 自 javax.servlet.[http.HttpServlet](http://http.HttpServlet) 类其 doGet 和 doPost 方法最终调用process方法，该方法是整个Servlet控制器的核心。

可能有人好奇，为何这个Servlet控制器命名为ControllerServlet，实际上，这里遵从了一 个约定：所有Servlet的类名称都带有Servlet后缀。

清单 2.3 ControllerServlet 类 package appdesignl.controuer;

import java.io.IOException;

import j avax.servlet.RequestDispatcher;

import j avax.servlet.ServletException;

import j avax,servlet,annotation.WebServlet;

import j avax.servlet.[http.HttpServlet](http://http.HttpServlet);

import j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest);

import j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse); import appdesignl.action.SaveProductAction; import appdesign1.form.Product Form; import appdesign1.model.Product; import j ava.math.BigDecimal;

@WebServlet(name = "ControllerServlet", urlPatterns = { "/input-product", "/save-product"}) public class ControllerServlet extends HttpServlet {

private static final long serialVersionUID = 1579L;

©Override

public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException { process(request, response);

}

©Override

public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException { process(request, response);

}

private void process(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {

String uri = request.getRequestURI();

/\*

* uri is in this form: /contextName/resourceName,
* for example: /appdesignl/input-product.
* However, in the event of a default context, the
* context name is empty, and uri has this form
* /resourceName, e.g.: /input-product

\*/

int lastIndex = uri.lastIndexOf("/");

String action = uri.substring(lastIndex + 1);

// execute an action

String dispatchUrl = null;

if ("input-product".eauals(action)) {

// no action class, just forward dispatchUrl = "/jsp/ProductForm.jsp";

} else if ("save-product".eauals(action)) {

// create form

ProductForm productForm = new ProductForm();

// populate action properties

productForm.setName(request.getParameter("name")); productForm.setDescription(

request.getParameter("description")); productForm.setPrice(request.getParameter("price"));

// create model

Product product = new Product(); product.setName(productForm.getName()); product.setDescription(productForm.getDescription()); try {

product.setPrice(new Bigdecimal(productForm.getPrice())); } catch (NumberFormatException e) {

}

// execute action method SaveProductAction saveProductAction = new SaveProductAction(); saveProductAction.save(product);

// store model in a scope variable for the view request.setAttribute("product", product); dispatchUrl = "/jsp/ProductDetails.jsp";

} if (dispatchUrl != null) {

RequestDispatcher rd =

request.getRequestDispatcher(dispatchUrl); rd.forward(request, response);

}

}

}

ControllerServlet的process方法处理所有输入请求。首先是获取请求URI和action名称。

String uri = request.getRequestURI(); int lastIndex = uri.lastIndexOf("/");

String action = uri.substring(lastIndex + 1);

在本亦例应用中，action值只会是input-product或save-product。

接着，process方法执行如下步骤。

1. 创建并根据请求参数构建一个表单对象。save-product操作涉及3个属性：name、 description和price。然后创建一个领域对象，并通过表单对象设置相应属性。
2. 执行针对领域对象的业务逻辑。
3. 转发请求到视图（JSP页面)。

process方法中判断action的if代码块如下：

// execute an action

if ("input-product".eauals(action))) {

// no action class, just forward dispatchUrl = "/jsp/ProductForm.jsp";

} else if ("save-product".eauals(action)) { // instantiate action class

}

对于input-product，无需任何操作，而针对save-product，则创建一个ProductForm对象 和Product对象，并将前者的属性值复制到后者。这个步骤中，针对空字符串的复制处理将留 到稍后的2.5节处理。

再次，process方法实例化SaveProductAction类，并调用其save方法。

// create form

ProductForm productForm = new ProductForm();

// populate action properties

productForm.setName(request.getParameter("name")); productForm.setDescription(

request.getParameter("description"));  
productForm.setPrice(request.getParameter("price"));

// create model

Product product = new Product(); product.setName(productForm.getName()); product.setDescription(productForm.getDescription()); try {

product.setPrice(new BigDecimal(productForm.getPrice()));

} catch (NumberFormatException e) {

}

// execute action method SaveProductAction saveProductAction = new SaveProductAction(); saveProductAction.save(product);

然后，将Product对象放入HttpServletRequest对象中，以便对应的视图能访问到。

// store action in a scope variable for the view request.setAttribute("product", product);

最后，process方法转到视图，如果action是product\_input，则转到ProductFormjsp页面，

否则转到 ProductDetails.jsp 页面。

// forward to a view if (dispatchUrl != null) {

RequestDispatcher rd =

request.getRequestDispatcher(dispatchUrl); rd.forward(request, response);

1. Action 类

这个应用中只有一个action类，负责将一个product持久化，例如数据库。这个action类 名为 SaveProductAction (见清单 2.4)。

清单 2.4 SaveProductAction 类

package appdesignl.action;

public class SaveProductAction {

public void save(Product product) {

// insert Product to the database

在这个示例中，SaveProductAction类的save方法是一个空实现。我们会在本章后续章节

中实现它。

2.3.5 mm

示例应用包含两个JSP页面。第一个页面ProductFormjsp对应于input-product操作，第 二个页面 ProductDetails.jsp 对应于 save-product 操作。ProductForm.jsp 以及 ProductDetails.jsp

页面代码分别见清单2.5和清单2.6。

清单 2.5 ProductForm.jsp <!DOCTYPE html>

<html>

<head>

<title>Add Product Form</title>

<style type=,1 text/css">@import url(css/main.css);</style> </head>

<body>

<form method=MpostM action=,,save-product ">

<hl> Add Product

<span>Please use this form to enter product details</span> </hl>

<label>

<span>Product Name: </span>

<input id= "name11 type=,ltextn name=nnamen

placeholder=nThe complete product nameM>

</label>

<label>

<span>Description: </span>

<input id=ndescription" type=Mtextn name=Mdescription" placeholder="Product descriptionn>

</label>

<label>

<span>Price: </label>

<input id="price" name=MpriceM type=MnumberM step=Manyn placeholder=MProduct price in #.## formatM>

</label>

<label>

<span>&nbsp: </span>

<input type=nsubmit">

</label>

</form>

</body>

</html>

注意

不要用HTMLTabel来布局表单，用CSS。

注意

价格输入域的step属性要求浏览器允许输入小数数字。

清单 2.6 ProductDetails.jsp <!DOCTYPE html>

<html>

<head>

<title>Save Product</title>

<style type=,1 text/css">@import url(css/main.css);</style> </head>

<body>

<div id="globaln>

<h4>The product has been saved.</h4>

<p>

<h5>Details:</h5>

Product Name: ${product.name}<br/>

Description: ${product.description}<br/>

Price: $${product.price}

</p>

</div>

</body>

</html>

ProductForm.jsp页面包含了一个HTML表单。ProductDetails.jsp页面通过表达式语言(EL) 访问 HttpServletRequest 所包含的 product 对象。

作为模型2的一个应用，本示例应用可以通过如下几种方式避免用户通过浏览器直接访 问JSP页面。

•将JSP页面都放到WEB-INF目录下。WEB-INF目录下的任何文件或子目录都受保护， 无法通过浏览器直接访问，但控制器依然可以转发请求到这些页面。

•利用一个servlet filter过滤JSP页面。

•在部署描述符中为JSP页面增加安全限制。这种方式相对容易些，无需编写filter 代码。

2.3.6测试应用

假定示例应用运行在本机的8080端口上，则可以通过如下URL访问应用：

<http://localhost:8080/appdesignl/input-product>

浏览器将显示图2.2的内容。

完成输入后，表单提交到如下服务端URL上：

<http://localhost:8080/appdesignl/save-product>

注意

可以将servlet控制器作为默认主页。这是一个非常重要的特性，使得在浏览器地址栏中仅输 入域名（如<http://example.com> )，就可以访问到该servlet控制器，这是无法通过filter方式完成的。

2.4模型2之Filter分发器

虽然servlet是模型2应用程序中最常见的控制器，但过滤器也可以充当控制器。但请注

意，过滤器没有作为欢迎页面的权限。仅输入域名时不会调用过滤器分派器。Stmts 2使用过滤

器作为控制器，是因为该过滤器也用于提供静态内容。 下面的例子（appdesign2)是一个采用filter分发器

色 webapp

T & css

J<f mainxss

▼办 j$p\_

ProductDetails.jsp ProductForm.jsp ▼ WEB-INF w £3- classes

T 13 appdesign2 w action

SaveProductAction.dass w (tn filter

^ DispatcherFilter.dass w form

iu\* Rroductform.dass w fe model

益 Product, c la »

图2.5 ^>pdesign2目录结构

的模型2应用，目录结构如图2.5所示。

JSP页面和Product类同appdesignl相同，但没有采 用servlet作为控制器，而是使用了一个名为FilterDispatcher

的过滤器（见清单2.7)。

清单 2.7 DispatcherFilter 类

package appdesign2.filter;

import java.io.IOException;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import j avax.servlet.annotation.WebFilter;

import j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest);

import appdesign2.action.SaveProductAction;

import appdesign2.form.ProductForm;

import appdesign2.model.Product;

import java.math.BigDecimal;

©WebFilter(filterName = "DispatcherFilter", urlPatterns = { "/\*" })

public class DispatcherFilter implements Filter {

^Override

public void init(FilterConfig filterConfig) throws ServletException {

^Override

public void destroy() { ©Override

public void doFilter(ServletRequest request,

ServletResponse response, FilterChain filterChain) throws IOException, ServletException { HttpServletRequest req = (HttpServletRequest) request; String uri = req.getRequestURI();

/\*

* uri is in this form: /contextName/resourceName, for
* example /appdesign2/input-product. However, in the
* case of a default context, the context name is empty,
* and uri has this form /resourceName, e.g.:
* /input-product

V

// action processing

int lastlndex = uri . lastlndexOf ;

String action = uri.substring(lastlndex + 1);

String dispatchUrl = null; if ("input-product".equals(action)) {

// do nothing

dispatchUrl = "/j sp/ProductForm.j sp";

}else if ("save-product11.equals (action) ) {

// create form

ProductForm productForm = new ProductForm();

// populate action properties

productForm.setName(request.getParameter("name")); productForm.setDescription(

request.getParameter("description"));  
productForm.setPrice(request,getParameter("price"));

// create model

Product product = new Product();

product.setName(productForm.getName());

product.setDescription(product.getDescription());

try {

product.setPrice(new BigDecimal(productForm.getPrice())); } catch (NumberFormatException e) {

}

// execute action method SaveProductAction saveProductAction = new SaveProductAction(); saveProductAction.save(product);

// store model in a scope variable for the view request,setAttribute("product", product); dispatchUrl = "/j sp/ProductDetails , j sp11 ;

}

// forward to a view if (dispatchUrl != null) {

RequestDispatcher rd = request

.getRequestDispatcher(dispatchUrl); rd.forward(request, response);

} else {

// let static contents pass filterChain.doFilter(request, response);

doFilter方法的内容同appdesignl中process方法。

由于过滤器的过滤目标是包括静态内容在内的所有网址，因此，若没有相应的action则 需要调用 filterChain.doFilter〇 〇

} else {

II let static contents pass filterChain•doFilter(request, response);

}

要测试应用，可以用浏览器访问如下URL:

<http://localhost:8080/appdesign2/input-product>

2.5校验器

在Web应用执行action时，很重要的一个步骤就是进行输入校验。校验的内容可以是简 单的，如检查一个输入是否为空，也可以是复杂的，如校验信用卡号。实际上，因为校验工 作如此重要，Java 社区专门发布了 JSR 303 Bean Validation 以及 JSR 349 Bean Validation 1.1 版 本，将Java世界的输入检验进行标准化。现代的MVC框架通常同时支持编程式和声明式两 种校验方法。在编程式中，需要通过编码进行用户输入校验，而在声明式中，则需要提供包 含校验规则的XML文档或者属性文件。

注意

即使您可以使用HTML5或JavaScript执行客户端输入验证，也不要依赖它，因为精明 的用户可以轻松地绕过它。始终执行服务器端输入验证！

本节的新应用（appdesign3)扩展自appdesignl，但多了一个ProductValidator类（见清单2.8)。 清单 2.8 ProductValidator 类

package appdesign3.validator; import java.util.ArrayList; import java.util.List; import appdesign3.form.ProductForm;

public class ProductValidator {

public List<String> validate(ProductForm productForm) {

List<String> errors = new ArrayList< >();

String name = productForm.getName(); if (name == null || name.trim().isEmpty()) {

errors.add("Product must have a name");

String price = productForm.getPrice(); if (price == null || price.trim().isEmpty()) {

errors.add("Product must have a price");

} else { try {

Float.parseFloat(price);

} catch (NumberFormatException e) {

errors.add(nInvalid price value")；

}

}

return errors;

注意

ProductValidator类中有一个操作ProductForm对象的validate方法，确保产品的名字非 空，其价格是一个合理的数字。validate方法返回一个包含错误信息的字符串列表，若返回 一个空列表，则表示输入合法。

现在需要让控制器使用这个校验器了，清单2.9展示了一个更新后的ControllerServlet,

注意黑体部分。

清单 2.9 新版的 ControllerServlet 类

package appdesign3.controller; import java.io■工〇Exception; import j ava,util.List;

import j avax.servlet.RequestDispatcher; import j avax.servlet.ServletException; import j avax.servlet.annotation,WebServlet; import j avax.servlet.[http.HttpServlet](http://http.HttpServlet); import j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest); import j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse); import appdesign3.action.SaveProductAction; import appdesign3.form.ProductForm; import appdesign3.model.Product; import appdesign3.validator.ProductValidator; import j ava.math.BigDecimal;

@WebServlet (name = "ControllerServlet**11,** urlPatterns = {

"/input-product11, ’’/save-product"})  
public class ControllerServlet extends HttpServlet {

private static final long serialVersionUID = 98279L;

SOverride

public void doGet (HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException { process(request, response);

}

@〇verride

public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException { process(request, response);

private void process(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {

String uri = request.getRequestURI();

/\*

* uri is in this form: /contextName/resourceName,
* for example: /appdesignl/input-product.
* However, in the case of a default context, the
* context name is empty, and uri has this form
* /resourceName, e.g.: /input-product

V

int lastlndex = uri■lastlndex〇f("/");

String action = uri.substring(lastlndex + 1);

String dispatchUrl = null;

if ("input-product".eauals(action)) {

// no action class, Hrele is nathing to be done dispatchUrl = "/j sp/ProductForm.j sp";

} else if (Msave-productM-eaoals(action)) {

// instantiatle action class

ProductForm productForm = new ProductForm();

// populate action properties productForm.setName(

request.getParameter("name")); productForm.setDescription(

request.getParameter("description"));  
productForm.setPrice(request.getParameter("price"));

// validate ProductForm

ProducbValidator productValidator = new ProductValidator();

List<String> errors =

productValidator. validate (productForm); if (errors. isEntpty ()) {

// create product from productForm Product product = new Product(); product. setName (productForm.getName ()); product.setDescription(

productForm.getDescription()); product.setPrice(new BigDecimal (productForm.getPrice()));

// no validation error execute action method SaveProductAction saveProductAction = new SaveProductAction(); saveProductAction.save(product);

// store model in a scope variable for the view request.setAttribute <"product", product); dispatchUrl = "/j sp/ProductDetails.j sp";

} else {

request.setAttribute("errorsn, errors); request.setAttribute{"form", productForm); dispatchUrl = "/j sp/ProductForm.j spM;

// forward to a new if (dispatchUrl != null) {

RequestDispatcher rd =

request.getRequestDispatcher(dispatchUrl); rd.forward(request, response);

新版的ControllerServlet类添加了初始化ProductValidator类并调用其validate方法的代码。

// validate ProductForm

ProductValidator productValidator = new ProductValidator();

List<String> errors =

productValidator .validate (productForm);

validate方法接受一个ProductForm参数，它封装了输入到HTML表单的产品信息。如果 不用ProductForm,则应将ServletRequest传递给验证器。

如果验证成功，validate方法返回一个空列表，在这种情况下，将创建一个产品并传递给

SaveProductAction，然后，控制器将 Product 存储在 ServletContext 中，并转发到 ProductDetails.jsp

页面，显示产品的详细信息。如果验证失败，控制器将错误列表和PmductFonn存储在

ServletContext 中，并返回到 ProductFormjsp。

if (errors,isEmpty()) {

// create Product from ProductForm Product product = new Product(); product.setName(productForm.getName()); product.setDescription(

productForm.getDescription()); product.setPrice(new BigDecimal(productForm.getPrice()));

**//no** validation error, execute action method SaveProductAction saveProductAction = new SaveProductAction(); saveProductAction.save(product);

// store action in a scope variable for the view request.setAttribute("product", product); dispatchurl=M/j sp/ProductDetails.j sp";

} else {

request.setAttribute("errors", errors); request.setAttribute("form", productForm); dispatchurl=M/j sp/ProductForm.j spM;

}

现在，需要修改appdesign3应用的ProductForm.jsp页面（见清单2.10)，使其可以显示

错误信息以及错误的输入。

清单 2.10 ProductForm.jsp 页面 <!DOCTYPE html>

<html>

<head>

<title>Add Product Form</title>

<style type=,1 text/css">@import url(css/main.css);</style>

</head>

<body>

<form method=Mpostn action=nsave-productM>

<hl>Add Product

<span>Please use this form to enter product details</span> </hl>

${empty requestScope.errors? : n<p style:1 color:red1 >’’

+= "Error(s)!"

+=

<!--${requestScope.errors.stream().map(

x -> n—Xli>M+=x+=n</liX!—M) .toList() }—>

${empty requestScope.errors? 11,1 : ,l</ulx/p>"}

<label>

<span>Product Name :</span>

<input id= "name11 type=ntext" name="name”

placeholder=,TThe complete product name" value=T'$ {form, name} T,/>

</label>

<label>

<span>Description :</span>

< input id=T,descriptionT, type=,Ttextf, name=,f descript ion,T placeholder=,TProduct description" value=T, $ { form. description} u />

</label>

<label>

<span>Price :</span>

<input id=T,price,T name=T,price,T type=TTnumber” step=,,anyf, placeholder=,TProduct price in #.## format” value= ”${ form • price} .T/>

</label>

<label>

<span>&nbsp;</span>

<input type=,T submit11/>

</label>

</ form>

</body>

</html>

现在访问 input-product，测拭 appdesign3 应用。

<http://localhost:8> 0 8 0/appdesgin3/input-product

若产品表单提交了无效数据，页面将显示相应的错误信息。图2.6显示了包含两条错误 信息的ProductForm页面。

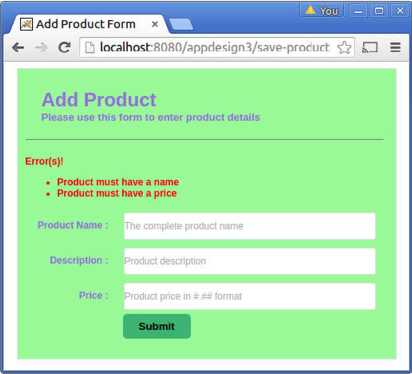


图2.6包含两条错误信息的ProductForm页面

2.6依赖注入

在过去数年间，依赖注入技术作为代码可测试性的一个解决方案已经广泛应用。实际上, Spring、Stmts2等伟大框架都采用了依赖注入技术。那么，什么是依赖注入技术？

关于这个，Martin Fowler写一篇优秀的文章：

<http://martinfowler.com/articles/injection.html>

在Fowler创造术语“依赖注入”之前，术语“控制反转”通常用于表示同样的事情。正 如Fowler在他的文章中指出的，两者不完全相同。

有两个组件A和B，A依赖于B。假定A是一个类，且A有一个方法importantMethod使用 到了 B，如下：

public class A {

public void importantMethod() {

B b =II get an instance of B b.usefulMethod();

}

要使用B，类A必须先获得组件B的实例引用。若B是一个具体类，则可通过new关键 字直接创建组件B实例。但是，如果B是接口，且有多个实现，则问题就变得复杂了。我们 固然可以任意选择接口 B的一个实现类，但这也意味着A的可重用性大大降低了，因为无法 采用B的其他实现。

示例appdesign4使用了一个自制依赖注入器。在现实世界的应用程序中，应该使用Spring。

示例应用程序用来生成PDF。它有两个动作，form和pdf。第一个没有action类，只是 转发到可以用来输入一些文本的表单；第二个生成PDF文件并使用PDFAction类，操作类本 身依赖于生成PDF的服务类。

PDFAction和PDFService类分别见清单2.11和清单2.12。

清单 2.11 PDFAction 类

package action;

import service.PDFService;

public class PDFAction {

private PDFService pdfService;

public void setPDFService(PDFService pdfService) { this.pdfService = pdfService;

}

public void createPDF(String path, String input) { pdfService.createPDF(path, input);

}

}

清单 2.12 PDFService 类

package service; import util.PDFUtil;

public class PDFService {

public void createPDF(String path, String input) {

PDFUtil.createDocument(path, input);

}

}

PDFService 使用了 PDFUtil 类，PDFUtil 最终采用了 Apache PDFBOx 库来创建 PDF 文档，

如果对创建PDF的具体代码有兴趣，可以进一步查看PDFUtil类。

这里的关键在于，如代码2.11所示，PDFAction需要一个PDFService来完成它的工作。 换句话说，PDFAction依赖于PDFService。没有依赖注入，你必须在PDFAction类中实例化 PDFService类，这将使PDFAction更不可测试。除此之外，如果需要更改PDFService的实现, 你必须重新编译PDFAction。

使用依赖注入，每个组件都有注入它的依赖项，这使得测试每个组件更容易。对于在依 赖注入环境中使用的类，你必须使其支持注入。一种方法是为每个依赖关系创建一个set方法。 例如，PDFAction类有一个setPDFService方法，可以调用它来传递PDFService。注入也可以 通过构造方法或类属性进行。

一旦所有的类都支持注入，你可以选择一个依赖注入框架并将它导入你的项目。Spring 框架、Google Guice、Weld 和 PicoContainer 是一些好的选择。

注意

依赖注入的Java规范是JSR 330和JSR 299

appdesign4程序使用Dependencylnjector类（见清单2.13)来替代依赖注入框架（在现实

世界的应用程序中，你会使用一个合适的框架)。这个类专为appdesigM应用设计，可以容易 地实例化。一旦实例化，必须调用其start方法来执行初始化，使用后，应调用其shutdown 方法以释放资源。在此示例中，start和shutdown都为空。

清单 2.13 Dependencylnjector 类

package util;

import action.PDFAction;

import service.PDFService;

public class Dependencylnjector {

public void start() {

// initialization code

public void shutDown() {  
// clean-up code

"

* Returns an instance of type, type is of type Class
* and not String because it1s easy to misspell a class name \*/

public Object getObject(Class type) { if (type == PDFService.class) { return new PDFService();

} else if (type == PDFAction.class) {

PDFService pdfService = (PDFService) getObject(PDFService.class);

PDFAction action = new PDFAction(); action.setPDFService(pdfService); return action;

}

return null;

要从Dependencylnjector获取对象，须调用其getObject方法，并传递目标对象的Class。 Dependencylnjector 支持两种类型，即 PDFAction 和 PDFService。例如，要获取 PDFAction 的

实例，你将通过传递PDFActionxlass来调用getObject:

PDFAction pdfAction = (PDFAction) dependencylnjector .getObject (PDFAction. class );

DependencyLijector (和所有依赖注入框架）的优雅之处在于它返回的对象注入了依赖。如果返 回的对象所依赖的对象也有依赖，则所依赖的对象也会注入其自身的依赖。例如，从

DependencylnjectDr 获取的 PDFAction 已包含 PDFService。无需在 PDFAction 类中自己创建 PDFService。

appdesign4中的servlet控制器如清单2.14所示。请注意，它在其init方法中实例化 Dependencylnjector，其 destroy 方法中调用 Dependencylnjector 的 shutdown 方法。servlet 不再创建它自己的依赖，相反，它从Dependencylnjector获取这些依赖。

清单 2.14 appdesign4 中 ControllerServlet

package servlet; import action.PDFAction; import java•io.IOException; import j avax.servlet.ReadListener; import j avax.servlet.RequestDispatcher; import j avax.servlet.ServletException; import j avax.servlet.annotation.WebServlet; import j avax.servlet.[http.HttpServlet](http://http.HttpServlet); import j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest); import j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse); import j avax.servlet.[http.HttpSession](http://http.HttpSession); import util,Dependencylnjector;

@WebServlet(name = "ControllerServlet", urlPatterns = {

"/form", "/pdf"})

public class ControllerServlet extends HttpServlet { private static final long serialVersionUID = 6679L; private Dependencylnj ector dependencylnj ector;

0〇verride

public void init() {

dependencylnjector = new Dependencylnjector(); dependencylnj ector.start();

@〇verride

public void destroy() {

dependencylnj ector.shutDown();

}

protected void process(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { ReadListener r = null;

String uri = request.getRequestURI();

/\*

* uri is in this form: /contextName/resourceName,
* for example: /applOa/product—input•
* However, in the case of a default context, the
* context name is empty, and uri has this form

\* /resourceName, e.g.: /pdf

V

int lastlndex = uri•lastlndex〇f("/");

String action = uri.substring(lastlndex + 1)； if ("form".equals(action)) {

String dispatchUrl = "/jsp/Form.jspM; RequestDispatcher rd =

request.getRequestDispatcher(dispatchUrl); rd.forward(request, response);

} else if ("pdf".equals(action)) {

HttpSession session = request.getSession(true); String sessionld = session.getld();

PDFAction pdfAction = (PDFAction) dependencylnjector .getObject(PDFAction.class);

String text = request.getParameter("text");

String path = request.getServletContext()

.getRealPath (’’/result”）+ sessionld + n .pdf pdfAction.createPDF(path, text);

// redirect to the new pdf StringBuilder redirect = new StringBuilder();

redirect.append(request.getScheme() + redirect.append(request.getLocalName()); int port = request.getLocalPort(); if (port != 80) {

redirect. append + port);

}

String contextPath = request.getContextPath(); if (!n/M.equals(contextPath)) {

redirect.append(contextPath);

}

redirect.append("/result/M + sessionld + M.pdfM); response.sendRedirect(redirect.toString());

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { process(request, response);

}

@〇verride

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException { process(request, response);

servlet支持两种URL模式，form和pdf。对于表单模式，servlet简单地转发到表单。对 于pdf模式，servlet使用PDFAction并调用其createDocument方法。此方法有两个参数：文 件路径和文本输入。所有PDF存储在应用程序目录下的result目录中，用户的会话标识符用 做文件名，而文本输入作为PDF文件的内容；最后，重定向到生成的PDF文件。以下是创建 重定向URL并将浏览器重定向到新URL的代码：

// redirect to the new pdf StringBuilder redirect = new StringBuilder();

redirect. append (request. getScheme () + //http or https

redirect. append(request.getLocalName()); // the domain int port = request.getLocalPort{); if (port != 80) {

redirect.append(":" + port);

)

String contextPath = request.getContextPath〇; if (!"/M.equals(contextPath)) {

redirect.append(contextPath);

}

redirect.append("/result/" + sessionld + n.pdf")p response.sendRedirect(redirect.toString());

现在访问如下URL来测试appdesign4应用。

<http://localhost:8080/appdesign4/form>

应用将展示一个表单（见图2.7)。

Wf |^| Add Product Form x

4- G D localhost:8080/appdesign4/form ☆ E

**Create PDF**

Please use this form to enter the text

Submit

如果在文本字段中输入一些内容并按提交按钮，服务器将创建一个PDF文件并发送重定 向到浏览器（见图2.8)。



图2.8 PDF文件

请注意，重定向网址将采用此格式。

<http://localhost:8080/appdesign4/result/sessionJd.pdf>

由于依赖注入器，appdesign4中的每个组件都可以独立测试。例如，可以运行清单2.15 中的PDFActionTest类来测试类的createDocument方法。

清单 2.15 PDFActionTest 类

package test;

import action.PDFAction;

import util.Dependencylnjector;

public class PDFActionTest {

public static void main(String[] args) {

Dependencylnjector dependencylnjector = nev? Dependencylnjector(); dependencylnjector.start();

PDFAction pdfAction = (PDFAction) dependencylnjector.getObject( PDFAction.class);

pdfAction.createPDF(H/home/j aneexample/Downloads/1.pdf",

"Testing PDFAction. . . . ;

dependencylnj ector•shutDown 〇/

}

}

如果你使用的是Java 7 EE容器，如Glassfish,可以让容器注入对servlet的依赖。应用 appdesign4中的servlet将如下所示：

public class ControllerServlet extends HttpServlet { @Inject PDFAction pdfAction;

SOverride

public void doGet(HttpServletRequest request,

HttpServletResponse response) throws 工OException, ServletException {

©Override

public void doPost(HttpServletRequest request,

HttpServletResponse response) throws IOException, ServletException {

2.7小结

在本章中，我们学习了基于MVC模式的模型2架构以及如何基于servlet控制器或者filter 分发器开发一个模型2应用。两个示例分别为appdesignl和appdesign2。使用servlet作为过 滤器上的控制器，一个明显的优点是你可以将servlet配置为欢迎页面。在模型2应用程序中， JSP页面通常用做视图，当然也可以使用其他技术，如Apache Velocity和FreeMarker。如果 JSP页面用做模型2体系结构中的视图，那些页面仅用于显示值，并且不应在其中显示脚本 元素。

在本章中，我们还构建了一个简单的MVC框架，其中包含一个验证器并为其配备了一 个依赖注入器。虽然自制的框架是一个良好的学习工具，但未来的MVC项目应基于一个成 熟的MVC框架上，如Spring MVC，而不是试图重复造轮子。

m

o

3早

Spring MVC 介绍

在第2章中，我们学习了现代Web应用程序广泛使用的MVC设计模式，也学习了模型 2架构的优势以及如何构建一个模型2应用。Spring MVC框架可以帮助开发人员快速地开发 MVC应用。

本章首先介绍采用Spring MVC的好处，以及Spring MVC如何加速模型2应用开发；然 后介绍Spring MVC的基本组件，包括Dispatcher Servlet，并学习如何开发一个“传统风格” 的控制器，这是在Spring 2.5版本之前开发控制器的唯一方式。另一种方式将在第4章中介绍。 之所以介绍传统方式，是因为我们可能不得不在基于旧版Spring的遗留代码上工作。对于新 的应用，我们可以采用基于注解的控制器。

此外，本章还会介绍Spring MVC配置，大部分的Spring MVC应用会用一个XML文档 来定义应用中所用到的bean。

3.1采用Spring MVC的好处

若基于某个框架来开发一个模型2的应用程序，我们要负责编写一个Dispatcher servlet 和控制类。其中，Dispatcher servlet必须能够做如下事情。

1. 根据URI调用相应的action。
2. 实例化正确的控制器类。
3. 根据请求参数值来构造表单bean。
4. 调用控制器对象的相应方法。
5. 转向到一个视图（JSP麵）。

Spring MVC是一个包含了 Dispatcher servlet的MVC框架。它调用控制器方法并转发到

视图。使用Spring MVC的第一个好处是，不需要编写Dispatcher servlet。以下是Spring MVC

具有的能加速开发的功能的列表。

* Spring MVC 提供了一个 Dispatcher Servlet，无需额外开发。
* Spring MVC使用基于XML的配置文件，可以编辑，而无需重新编译应用程序。
* Spring MVC实例化控制器，并根据用户输入来构造bean。
* Spring MVC可以自动绑定用户输入，并正确地转换数据类型。例如，Spring MVC能 自动解析字符串，并设置float或decimal类型的属性。
* SpringMVC可以校验用户输入，若校验不通过，则重定向回输入表单。输入校验是可 选的，支持编程方式以及声明方式。关于这一点，SpringMVC内置了常见的校验器。
* Spring MVC是Spring框架的一部分，可以利用Spring提供的其他能力。
* SpringMVC支持国际化和本地化，支持根据用户区域显示多国语言。
* Spring MVC支持多种视图技术。最常见的JSP技术以及其他技术包括Velocity和 FreeMarker。

1. Spring MVC 的 DispatcherServlet

回想一下，第2章建立了一个简单的MVC框架，包含一个充当调度员的servlet。基于Spring MVC，则无需如此。Spring MVC中自带了一个开箱即用的Dispatcher Servlet,该Servlet的全名 是 org.springframework.web.servletDispatcherServlet。

要使用这个servlet，需要在部署描述符（web.xml文件）中使用servlet和servlet-mapping

元素来配置它，如下所示：

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet </servlet-class>

<load-on-startup>l</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name> <!-- map all requests to the DispatcherServlet --> <url-pattern>/</url-pattern>

</servlet-mapping>

servlet元素内的on-startup元素是可选的。如果它存在，则它将在应用程序启动时装载 servlet并调用它的init方法。若它不存在，则在该servlet的第一个请求时加载它。

Dispatcher servlet将使用Spring MVC诸多默认的组件。此外，初始化时，它会寻找在应 用程序的WEB-INF目录下的一个配置文件，该配置文件的命名规则如下：

*servletName-servlet*.xml

其中，servletName是部署描述符中的Dispatcher servlet的名称。如果这个servlet的名称 是SpringMVC，则在应用程序目录的WEB-INF下对应的文件是SpringMVC-servlet.xml。

此外，也可以把Spring MVC的配置文件放在应用程序目录中的任何地方，只要告诉 Dispatcher servlet在哪里找到该文件。我们使用servlet声明下的一个init-param元素来做到这 一点。init-param 元素拥有一个值为 contextConfigLocation 的 param-name 元素，其 param-value 元素则包含配置文件的路径。例如，可以利用init-param元素将默认的文件名和文件路径更改 为 WEB-INF/config/simple-config.xml 〇

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet </servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name> <param-value>/WEB-INF/config/simple-config.xml</param-value> </init-param>

<load-on-startup>l</load-on-startup>

</servlet>

1. Controller 接口

在Spring 2.5前，开发一个控制器的唯一方法是实现org.springframework.web.servlet. mvc.Controller接口。这个接口公开了一个handleRequest方法。下面是该方法的签名：

ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)

其实现类可以访问对应请求的HttpServletRequest和HttpServletResponse，还必须返回一

个ModelAndView对象，它包含视图路径或视图路径和模。

Controller接口的实现类只能处理一个单一动作（action)，而一个基于注解的控制器可 以同时支持多个请求处理动作，并且无须实现任何接口。第4章将会详细介绍。

3.4第一个Spring MVC应用

本章的示例应用程序springmvc-introl展示了基本的Spring MVC应用。该应用程序同第 2章学习的appdesignl应用非常相似，专用于展示Spring MVC是如何工作的。Sringmvc-introl 应用也有两个控制器类似于appdesignl的控制器类。

3.4.1目录结构

图3.1展示了 springmvc-introl的目录结构。注意，WEB-INF/lib目录包含了 Spring MVC 所需要的所有的jar文件。特别需要注意的是spring-webmvc-x.y.z.jar文件，其中包含了 DispatcherServlet 的类。还要注意 Spring MVC 依赖于 Apache Commons Logging 组件，没有它， Spring MVC应用程序将无法正常工作。可以从以下网址下载这个组件：

<http://commons.apache.org/proper/commons-loggins/download_logging.cgi>

& webapp WEB-INF ▼色 classes v & controller

^ Input Product Con troller.class & SaveProductControUer.class v & domain

B Product.class ^ form

& ProductForm.class ▼ &jsp

圍 ProductDetails.jsp li ProductForm.jsp ▼盔 lib

commons-logging-1.1.2.jar ^ spring-beans-4.2.4.RELEASE.jar sprin9-context-4.2.4.RELEASE.jar spring-core-4.2.4.RELEASE.jar ^ spring-expression-4.2.4.RELEASEjar 為 spring-web-4.2ARELEASE.jar spring-web mvc-4.2.4.RELEA5E.jar S3 springmvc-servlet.xml ® web.xml

图 3.1 springmvc-intro 1 的目录结构

该示例应用的所有JSP页面都存放在/WEB-INF/jsp目录下，确保无法被直接访问。

3.4.2部署描述符文件和Spring MVC配置文件

清单3.1部署描述符（web.xml)文件

<?xml version="1. 0" encodings"UTF-8"?>

<web-app version="3.1"

xmlns="<http://xmlns.jcp.org/xml/ns/javaee>" xmlns:xsi="<http://www.w3.org/2> 0 01/XMLSchema-instance" xsi:schemaLocation="<http://xmlns.jcp.org/xml/ns/javaee> ://xmlns.jcp.org/xml/ns/j avaee/web-app\_3\_l. xsd"> <servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet </servlet-class>

<l〇ad—〇n—startup>1</l〇ad—〇n—startup>

</servlet> <servlet-mapping>

<servlet—name>springmvc</servlet—name>

<!—— map all requests to the DispatcherServlet ——> <url—pattern>/</url—pattern>

</servlet—mapping>

</web—app>

这里告诉了 Servlet/ JSP容器，我们将使用Spring MVC的Dispatcher Servlet，并通过 url-pattern元素值配置为“厂’，将所有的URL映射到该servlet。由于servlet元素下没有init-param 元素，所以Spring MVC的配置文件在/WEB-INF文件夹下，并按照通常的命名约定。

清单3.2 Spring MVC配置文件

<?xml version="1. 0" encodings"UTF-8"?>

<beans *xmlns= "*<http://www.springframework.org/schema/beans>*" xmlns:xsi= "*<http://www.w3.org/2001/XMLSchema-instance> *"* xsi:schemaL〇cati〇n= "<http://www.springframework.org/schema/beans> <http://www.springframework.org/schema/beans/spring-beans.xsd>*">*

<bean name=*"/input-produrt"*

class=*"controller.InputProductController"/>*

<bean name=*"/save-product"*

class=*"controller.SaveProductController"/*>

</beans>

这里声明了 InputProductController和SaveProductController两个控制器类，并分别映射到 /product\_input和/product\_save。两个控制器将在3.4.3节中讨论。

1. Controller 类

springmvc-introl 应用程序有InputProductController和 SaveProductController这两个“传统” 风格的控制器，分别实现了 Controller接口。代码分别见清单3.3和清单3.4。

清单 3.3 InputProductController 类

package controller;

import j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest); import j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse); import org.apache,commons.logging.Log; import org.apache.commons.logging.LogFactory; import org.springframework.web.servlet.ModelAndView; import org.springframework.web.servlet.mvc.Controller;

public class InputProductController implements Controller {

private static final Log logger = LogFactory

.getLog(InputProductController.class);

@Override

public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception { logger, info ("InputProductController called11); return new ModelAndView("/WEB-INF/jsp/ProductForm.jspn)；

InputProductController 类的 handleRequest 方法只是返回一个 ModelAndView,包含一个视 图，且没有模型。因此，该请求将被转发到/WEB-INF/jsp/ProductFormjsp页面。

清单 3.4 SaveProductController 类

package controller;

import j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest); import j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse); import org.apache.commons.logging.Log; import org.apache,commons.logging.LogFactory; import org.springframework.web.servlet.ModelAndView; import org.springframework.web,servlet.mvc.Controller; import domain.Product;

import form.ProductForm;

public class SaveProductController implements Controller {

private static final Log logger = LogFactory .getLog(SaveProductController.class);

@Override

public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception { logger.info("SaveProductController called");

ProductForm productForm = new ProductForm();

// populate action properties

productForm.setName(request.getParameter("name")); productForm.setDescription(request.getParameter( "description"));

productForm.setPrice(request.getParameter("price"));

// create model

Product product = new Product();

product.setName{productForm.getName{));

product.setDescription(productForm.getDescription());

try {

product.setPrice(

Float.parseFloat(productForm.getPrice()));

} catch (NumberFormatException e) {

}

// insert code to save Product return new ModelAndView ("/WEB-INF/j sp/ProductDetails . j sp11,

"productM, product);

}

}

SaveProductController 类的 handleRequest 方法中，首先用请求参数创建一个 ProductForm 对象；然后，它根据ProductForm对象创建Product对象。由于ProductForm的price属性是一 个字符串，而其在Product类对应的是一个float，此处类型转换是必要的。在第4章中，我们 将学习在Spring MVC中如何省去ProductForm对象，使事情变得更省力。

SaveProductController 的 handleRequest 方法最后返回的 ModelAndView 模型包含了视图

的路径、模型名称以及模型（product对象)。该模型将提供给目标视图，用于界面显示。

1. View 类

springmvc-introl应用程序中包含两个JSP页面：ProductFormjsp页面（代码见清单3.5) 和 ProductDetails.jsp 页面（见清单 3.6)。

清单 3.5 ProductForm.jsp 页面 <!DOCTYPE HTML>

<html>

<head>

<title>Add Product Form</title>

<style type=Mtext/css">@import url(css/main.css);</style> </head>

<body> <div id="global">

<form action=Msave- Product" method="post">

<fieldset>

<legend>Add a product</legend>

<label for=MnameM>Product Name: </label>

<input type=MtextM id="name" name=nnameM value="" tabindex="1">

<label f or= M descript ion11 >Descript ion: </label>

<input type=’’text" id="description" name=Mdescription" tabindex=" 211 >

<label for=MpriceM>Price: </label>

<input type=MtextM id=npricen name=npriceM tabindex=’’3’’>

<div id="buttonsn>

<label for=Mduminyn> </label>

<input id=Mreset" type="reset" tabindex=M4M>

<input id=11 submit" type=Msubmit" tabindex="5n value="Add Product’’〉

</div>

</fieldset>

</form>

</div>

</body>

</html>

此处不适合讨论HTML和CSS,但需要强调的是清单3.5中的HTML是经过适当设计的，

并且没有使用<table>来布局输入字段。 清单 3.6 ProductDetails.jsp 页面

<!DOCTYPE HTML>

<html>

<head>

<title>Save Product</title>

<style type="text/css">0import url(css/main.css);</style>

</head>

<body>

<div id=”global">

<h4>The product has been saved.</h4>

<p>

<h5>Details: </h.5>

Product Name: ${product.name}<br/>

Description: ${product.description}<br/>

Price: $${product.price}

</p>

</div>

</body>

</html>

ProducfDetailsjsp 页面通过属性名“product”来访问由 SaveProductController 传入的 Product

对象。这里用JSP表达式语言来显示Product对象的各种属性。我们将在第8章中学习JSP EL。

3.4.5测试应用

现在，在浏览器中输入如下URL来测试应用：

<http://localhost:8080/springmvc-introl/input-product>

会看到类似图3.2的产品表单页面，在空字段中输入相应的值后单击“Add Product (添 加产品)”按钮，会在下一页中看到产品属性。

f @ Add Product Form x、

C localhost lOfiO/sprmgmvc intro\*./input product 「Add a produa

□



图3.2 ^pringmvc-introl的产品表单

3.5视图解析器

Spring MVC中的视图解析器负责解析视图。可以通过在配置文件中定义一个MewResolver (如下）来配置视图解析器。

<bean id=11 viewResolver11 class=norg. springframework. web. servlet. ^♦view • InternalResourceViewResolver">

〈property name=，,prefixn value="/WEB-INF/jsp/"/>

<property name=M suffix" value=,!. jsp!,/>

</bean>

视图解析器配置有前缀和后缀两个属性。这样一来，view 路径将缩短。例如，仅需提供“myPage”，而不必再将视图路径 设置为/WEB-INF/jsp/myPage.jsp，视图解析器将会自动增加前缀 和后缀。

**^ webapp ▼^WEB-INF T & classes T controller**

**InputProductControUer.class SaveProductControHer.class T & domain**

总 **Product.class ▼ form**

**ProductForm.class T & conftg**

**® springmvc-config.xml** ▼色 **jsp**

圍 **ProductDetails.jsp il ProductForm.jsp** ▼ ^ Lib

**commons-logging-1.1.2.jar ^ spring-beans-4.2.4.RELEASE.jar ^ sprin9-context-4.2.4.RELEASE.jar ^ spring-core-4.2.4.RELEASE.jar ^ spring-expres5ion-4.2.4.RELEASE.jar ^ spring-web-4.2.4.RELEASE.jar ^ spring-webmvc-4.2.4.RELEASE.jar ® web.xml**

图 3.3 springmvc-into2 文件结构

以 springmvc-intro2 应用为例，该例子同 springmvc-introl

应用类似，但调整了配置文件的名称和路径。此外，它还配置 了默认的视图解析器，为所有视图路径添加前缀和后缀，如图 3.3所示。

springmvc-intro2中，Spring MVC的配置文件被重命名为 springmvc-config.xml 中，并移动到/WEB-INF/config 目录下。为 了让Spring MVC可以正确加载到该配置文件，需要将文件路径 配置到 Spring MVC 的 Dispatcher servlet。清单 3.7 显示了 springmvc-intro2应用的部署描述符（web.xml文件)。

清单3.7 springmvc-intro2应用的部署描述符 <?xml version="l. 0" encoding=,,UTF-8f, ?>

<web-app version=M3.1"

xmlns=M<http://xmlns.j> cp.org/xml/ns/j avaeeM xmlns :xsi=,,http: //www. w3 . org/2001/XMLSchema-instance,, xsi : schemaLocation=,,http: / /xmlns . j cp. org/xml/ns/j avaee ^♦http: //xmlns .jcp.org/xml/ns/javaee/web-app\_3\_l .xsd">

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet </servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>

/WEB-INF/config/springmvc-config.xml </param-value>

</init-param>

<load-on-startup>l</load-on-startup〉

</servlet> <servlet-mapping>

<servlet-name>springmvc</servlet-name> <ur1-pattern>/</url-pattern〉

</servlet-mapping>

</web-app>

需要特别注意的是Web.xml文件中的init-param元素。要使用非默认配置文件的命名和路

径，需要使用名为contextConfigLocation的init-param，其值应为配置文件在应用中的相对路 径（见清单3.8)。

清单3.8 springmvc-intro2的配置文件

<?xml version="l. 0" encoding=,,UTF-8f, ?>

<beans xmlns="<http://www.springframework.org/schema/beans>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi : schemaLocation=,,http: //www. springframework.org/schema/beans <http://www.springframework.org/schema/beans/spring-beans.xsdn>>

<bean name="/input-product"

class="controller.InputProductController"/>

<bean name="/save-product"

class= M controller.SaveProductController M/>

<bean id="viewResolverf,

class="org.springframework.web.servlet.view. ^♦lnternalResourceViewResolver,,>

<property name=Mprefixn value=M/WEB-INF/jsp/M/>

^property name=,,suffix,' value=n .jsp'’/>

</bean>

</beans>

在浏览器中输入如下URL,测试app03b应用：

<http://localhost:8080/springmvc-intro2/input-product>

即可看到如图3.2所示的表单页面。

3.6小结

本章是Spring MVC的入门介绍。我们学习了如何开发一个类似第2章的简单应用。在 Spring MVC中，我们无需编写自己的Dispatcher servlet，并通过实现控制器接口来编写控制 器。这是传统风格的控制器。从Spring 2.5版本开始，Spring提供了一个更好的开发控制器的 方式，如采用注解。第4章会深入介绍这种风格的控制器。

第4章 基于注解的控制器

在第3章中，我们创建了两个采用传统风格控制器的Spring MVC应用程序。其控制器是 实现了 Controller接口的类。Spring 2.5版本引入了一个新途径：使用控制器注释类型。本章 介绍了基于注解的控制器，以及各种对应用程序有用的注释类型。

* 1. Spring MVC注解类型

使用基于注解的控制器的几个优点。其一，一个控制器类可以处理多个动作（而实现了 Controller接口的一个控制器只能处理一个动作）。这就允许将相关的操作写在同一个控制器 类中，从而减少应用程序中类的数量。

其二，基于注解的控制器的请求映射不需要存储在配置文件中。使用RequestMapping注 释类型，可以对一个方法进行请求处理。

Controller和RequestMapping注释类型是Spring MVC API最重要的两个注解类型。本章

重点介绍这两个，并简要介绍了一些其他不太流行的注解类型。

* + 1. Controller 注解类型

org.springframework.stereotype.Controller注解类型用于指示Spring类的实例是一个控制 器。下面是一个带注解@〇01113\*〇1161<的例子。

package com.example.controller; import org.springframework.stereotype;

@Controller

public class CustomerController {

II request-handling methods here

Spring使用扫描机制来找到应用程序中所有基于注解的控制器类。为了保证Spring能找 到你的控制器，需要完成两件事情。首先，需要在Spring MVC的配置文件中声明spring- context， 如下所示：

Cbeans

xmlns:context=n<http://www.springframework.org/schema/context>"

>

然后，需要应用<component-scan/>元素，如下所示：

<context: component-scan base-package=MjbasePacicageM/>

请在<<：011^01^111：-80011/>元素中指定控制器类的基本包。例如，若所有的控制器类都在 com.example.controller及其子包下，则需要写一个如下所示的<component-scan/>元素：

<context:component-scan base-package=Mcom.example.controllerM/>

现在，整个配置文件看上去如下所示：

<?xml version="l.0M encoding=MUTF-8"?>

<beans xmlns=Mhttp: //www, springf ramework. org/schema/beans11 xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instanceM>xmlns:p="<http://www.springframework.org/schema/pM>xmlns: context=Mhttp: / /www. springf ramework, org/schema/context11 xsi:schemaLocation="

<http://www.springframework.org/schema/beans>

<http://www.springframework.org/schema/beans/spring-beans.xsd> <http://www.springframework.org/schema/context> [http://www.springframework.org/schema/context/spring- context](http://www.springframework.org/schema/context/spring-context) .xsdM>

<context:component-scan base-package="com,example.controllerM/>

<!一一 …一一>

</beans>

请确保所有控制都在基本包下，并且不要指定一个;泛的基本包(如指定com.example， 而非com.example.controller，前者就更广泛)，因为这会使得Spring MVC扫描了无关的包。

* + 1. RequestMapping 注解类型

现在，我们需要在控制类的内部为每一个动作开发相应的处理方法。要让Spring知道用 哪一种方法来处理它的动作，需要使用 〇rg.springframework.web.bind.annotation.Request Mapping注解类型映射的URI与方法。

RequestMapping注解类型的作用同其名字所暗示的：映射一个请求和一种方法。可以使 注解一种方法或类。

一个采用@RequestMapping注解的方法将成为一个请求处理方法，并由调度程序在接收 到对应URL请求时调用。

下面是一个RequestMapping注解方法的控制器类。

package com.example.controller;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

©Controller

public class CustomerController {

0RequestMapping(value = "/input-customer M) public String inputCustomer() {

// do something here

return "CustomerForm";

}

}

使用RequestMapping注解的value属性将URI映射到方法。在上面的例子中，我们将 input-customer映射到inputCustomer方法。这样，可以使用如下URL访问inputCustomer方法。

<http://domain/context/input-customer>

由于value属性是RequestMapping注释的默认属性，因此，若只有唯一的属性，则可以 省略属性名称。换句话说，如下两个标注含义相同。

©RequestMapping(value = "/input-customer ")

©RequestMapping (11 /input-customer ’’）

但如果有多个属性时，就必须写入value属性名称。

请求映射的值可以是一个空字符串，此时该方法被映射到以下网址：

<http://domaln/context>

RequestMapping除了具有value属性外，还有其他属性。例如，method属性用来指示该 方法仅处理哪些HTTP方法。

例如，仅当在11111^081'或？111'方法时，才访问到下面的？1'(似88〇此1'方法。

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

@RequestMapping(value="/process-order",

method={RequestMethod.POST, RequestMethod.PUT}) public String processOrder() {

**//do** something here return "OrderForm";

}

若method属性只有一个HTTP方法值，则无需花括号。例如，

@RequestMapping (value:”/process-order11, method=RequestMethod. POST)

如果没有指定method属性值，则请求处理方法可以处理任意HTTP方法。 此外，RequestMapping注解类型也可以用来注解一个控制器类，如下所示：

import org.springframework.stereotype.Controller;

@Controller

@RequestMapping(value="/customern) public class CustomerController {

在这种情况下，所有的方法都将映射为相对于类级别的请求。例如，下面的deleteCustomer

方法。

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

©Controller

@ Re qu e s tMapp ing(M/cust ome r M) public class CustomerController {

@RequestMapping(value=M/deleten,

method={RequestMethod.POST, RequestMethod.PUT}) public String deleteCustomer() {

// do something here return ...;

}

由于控制器类的映射使用“/customer”，而deleteCustomer方法映射为“/delete”，贝!J如F

URL会映射到该方法上。

<http://domain/context/customer/delete>

42编写请求处理方法

每个请求处理方法可以有多个不同类型的参数，以及一个多种类型的返回结果。例如， 如果在请求处理方法中需要访问HttpSession对象，则可以添加的HttpSession作为参数，Spring 会将对象正确地传递给方法。

@RequestMapping (11 /uriv )

public String myMethod(HttpSession session) {

session.addAttribute{key, value);

或者，如果需要访问客户端语言环境和HttpServletRequest对象，则可以在方法签名上包 括这样的参数：

@RequestMapping (11 /uriv )

public String myOtherMethod(HttpServletRequest request,

Locale locale) {

// access Locale and HttpServletRequest here

下面是可以在请求处理方法中出现的参数类型：

* javax.servlet.ServletRequest ^ javax.servlet.[http.HttpServletRequestc](http://http.HttpServletRequestc)
* javax.servlet.ServletResponse 或 javax.servlet.http.HttpServletResponse。
* javax.servlet.http.HttpSession。
* org.springframework.web.context.request.WebRequest ^ org.springframework.web. context.request.NativeWebRequest〇

4.2编写请求处理方法

* java.util.Locale〇
* java.io.InputStream 或 java.io.Reader。
* java.io.Ou^mtStream 或 java.io.Writer。
* java.security.Principal。
* HttpEntity<?>paramters
* java.util.Map/org.springframework.ui.Model/〇
* org.springframework.ui.ModelMap 〇
* org.springframework.web.servlet.mvc.support.RedirectAttributes〇
* org. springframework. validation. Errors / 〇
* org. springframework. validation. BindingResult 〇

•命令或表单对象。

* org.springframework.web.bind.support.SessionStatus。
* org.spring£ramework.web.util.UriComponentsBuilder〇

•带@?乱111'\/|£11^1316、@MatrixVariable、@RequestParam、@RequestHeader、@Requestfiody

注释的对象。

特别重要的是org.springframework.ui.Model类型。这不是一个Servlet API类型，而是一 个包含Map的Spring MVC类型。每次调用请求处理方法时，Spring MVC都创建Model对象 并将其Map注入到各种对象。

请求处理方法可以返回如下类型的对象：

* ModelAndView〇
* Model 〇

•包含模型的属性的Map。

* View〇

•代表逻辑枧图名的String。

* void〇

•提供对Servlet的访问，以响应HTTP头部和内容HttpEntity或ResponseEntity对象。

* Callable。
* DeferredResulto

•其他任意类型，Spring将其视作输出给View的对象模型。

本章后续会展示一个例子，以进一步学习如何开发一个请求处理方法。

4.3应用基于注解的控制器

本章的示例应用amiotatedl基于第2章和第3章的例子重写，展示了包含有两个请求处 理方法的一个控制器类。

annotatedl和前面的应用程序间的主要区别在于，annotated!的控制器类增加了注解 @Controller。此外，Spring配置文件也增加了一些元素，后续小节中会详细介绍。

4.3.1目录结构

图4.1展示了 annotated!的目录结构。注意，annotatedl中只有一个控制器类，而不是两 个，同时新增了一个名为index.html的HTML文件，以便Spring MVC Servlet的URL模式设

置为“/”时，依然可以访问静态资源。

4.3.2配置文件

annotatedl有两个配置文件。第一个为部署描述符（web.xml文件）中注册Spring MVC 的 Dispatcher Servlet。第二个为 springmvc-config.xml，即 Spring MVC 的配置文件。

清单4.1和清单4.2分别展示部署描述符和Spring MVC的配置文件。

清单4.1 annotatedl (web.xml)的部署描述符

<?xml version=" 1.0" encoding=,,UTF-8,' ?>

<web-app version=M3.1"

xmlns=M<http://xmlns.j> cp.org/xml/ns/j avaeeM xmlns :xsi='’http: //www. w3 ■ org/2001/XMLSchema-instance,' xsi:schemaLocation=M<http://xmlns.j> cp.org/xml/ns/j avaee <http://xmlns.j> cp.org/xml/ns/j avaee/web-app\_3\_l .xsdM>

<servlet>

<servlet-name>springTnvc</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet </servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name> <param-value>

/WEB-INF/config/springmvc-config.xml

</param-value>

</init-parara>

<load-on-startup>l</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-narae>springmvc</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web\_app>

图4.1为annotatedl的目录结构。

^ webapp

► fecss ^ ^WEB-INF ^ (b\* classes w controller

& ProductController.dass w ^ domain

Product class w ^ form

ProductForm.class w ^ config

El 5prin9mvc<onrig.xml ►籬 jsp ^ tfy tib

& commons-logging-1.1.2.jdr SI sprin〇-aop-4,2.4.RELEASE.jar spring-beans-4.2.4.RELEASEjar Q spring<ontext-4,2.4,RELEASe.jar St spring-core-4.2.4.RELEASLjar Si sprin9-expressiorv4.2.4.RELEASE.jar Q sp r r ng-web -4.2.4. R E L EASLj a r Q spring>webmvc<2.4 RELEASEjar web.xml

图4.1 annotatedl的目录结构

另外，在部署描述符中的<servlet-mapping^>元素，Spring MVC 的 dispatcher-servlet 的 URL

模式设置为“/”，当URL模式设置为时，意味着所有请求（包括那些用于静态资源）都 被映射到Dispatcher Servlet。为了正确处理静态资源，需要在Spring MVC配置文件中添加一 些 <resources/> 元素。

清单 4.2 springmvc-config.xml 文件

<?xml version=" 1.0" encoding=,,UTF-8M ?>

<beans xmlns=,,http: //www. springf ramework. org/schema/beans" xmlns :xsi="http: //www. w3 . org/2001/X]yILSchema-instance,, xmlns:p="<http://www.springframework.org/schema/pM>xmlns :mvc=,,http: //www. springf ramework. org/schema/mvc" xmlns : context=,,http: //www. springf ramework. org/schema/contextM xsi:schemaLocation=M

<http://www.springframework.org/schema/beans>

<http://www.springframework.org/schema/beans/spring-beans.xsd>

<http://www.springframework.org/schema/mvc>

<http://www.springframework.org/schema/mvc/spring-mvc.xsd>

<http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/springcontext.xsdM>> <context:component-scan base-package="controller"/>

<mvc:annotation-driven/>

<mvc: resources mapping="/css/\*\*" location=’’/css/"/>

<mvc:resources mapping=M/\*.html" location="/"/>

<bean id="viewResolver"

class=,,org. springf ramework. web. servlet .view. ^♦lnternalResourceViewRes〇lver,,>

〈property name=nprefixn value=M/WEB-INF/j sp/M/>

<property name=,,suf f ix" value=" • j sp"/>

</bean>

</beans>

清单4.2 (SpringMVC的配置文件）中最主要的是<component-scan/>元素。这是要指示Spring MVC扫描目标包中的类，本例是controller包。接下去是一个<annotation-driven^>元素和两个 <resources>元素。<annotation~driven/>元素做了很多的事情，其中包括注册用于控制器注解的bean 对象。<resources/>元素贝!j指示Spring MVC哪些静态资源需要单独处理(不通过Dispatcher Servlet)。

在清单4.2的配置文件中，有两个<res〇urCes/>元素。第一个确保在/CSS目录下的所有文 件可见，第二个允许显示所有的.html文件。

注意

如果没有<annotation-driven/>，<^〇1111〇68/>元素会阻止任意控制器被调用。若不需要使 用 resources，则不需要〈annotation-driven/〉元素。

1. Controller 类

如前所述，使用Controller注释类型的一个优点在于：一个控制器类可以包含多个请 求处理方法。如清单4.3所示，ProductController类中有inputProduct和saveProduct两个

方法。

清单 4.3 ProductController 类

package controller;

import java.match.Bigoecimal

import org.apache.commons.logging.Log;

import org.apache,commons.logging.LogFactory;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.RequestMapping; import domain.Product; import form.ProductForm;

@Controller

public class ProductController {

private static final Log logger =

LogFactory.getLog(ProductController.class);

@RequestMapping(value=M/input-product") public String inputProduct() {

logger.info("inputProduct called"); return 11 ProductForm";

@RequestMapping(value=M/save-productM)

public String saveProduct(ProductForm productForm, Model model){ logger.info("saveProduct called");

// no need to create and instantiate a ProductForm

// create Product

Product product = new Product();

product.setName(productForm.getName());

product.setDescription(productForm.getDescription());

try {

product.setPrice(new BigDecimal(productForm.getPrice())); } catch (NumberFormatException e) {

// add product

model.addAttribute("product", product);

return "ProductDetails";

}

}

其中，ProductController 的 saveProduct 方法的第二个参数是 org.springframework.ui.Model 类型。无论是否会使用，Spring MVC都会在每一个请求处理方法被调用时创建一个Model 实例，用于增加需要显示在视图中的属性。例如，通过调用model.addAttribute来添加Product 实例：

model.addAttribute("product", product);

Product实例就可以像被添加到HttpServletRequest中那样访问了。

1. View

annotatedl也有与前面章节示例类似的两个视图：ProductFormjsp页面（见清单4.4)和 ProductDetails.jsp 页面（见清单 4.5)。

清单 4.4 ProductForm.jsp 页面

<!DOCTYPE HTML>

<html>

<head>

<title>Add Product Form</title>

<style type=Mtext/css">@import url(css/main.css);</style>

</head>

<body>

<div id="globaln>

<form action=Msave-productM method="postM>

<fieldset>

<legend>Add a product</legend>

<p>

〈label for=Mname">Product Name: </label> <input type=ntextn id=’’name” name=Mname11 tabindex=Ml">

</p>

<p>

<label for="description">Description: </label> <input type=MtextM id=ndescription" name="description" tabindex=n2’’>

</p>

**<p>**

<label for="price">Price: </label>

<input type=ntext" id="priceM name=,,priceM tabindex=M3M>

</p>

<p id="buttonsM>

<input id="reset" type="reset" tabindex=M4n> <input id=11 submit11 type=Msubmit11 tabindex="5" value="Add Product"〉

</p>

</fieldset>

</form>

</div>

</body>

</html>

清单 4.5 ProductDetails.jsp 页面

<!DOCTYPE HTML>

<html>

<head>

<title>Save Product</title>

<style type=,1 text/css">@import url(css/main.css);</style> </head>

<body>

<div id="globaln>

<h4>The product has been saved.</h4>

<p>

<h5>Details:</h5>

Product Name: ${product.name}<br/>

Description: ${product.description}<br/>

Price: $${product.price}

</p>

</div>

</body>

</html>

4.3.5测试应用

在浏览器中输入如下URL来测试annotatedl。

<http://localhost:8080/annotatedl/input-product>

浏览器会显示Product表单，如图4.2所示。

单击“Add Product”按钮，会调用saveProduct方法。



图 4.2 Product 表单

4.4应用@Autowired和@861\*¥^进行依赖注入

使用Spring框架的一个好处是容易进行依赖注入。毕竟，Spring框架一开始就是一个依 赖注入容器。将依赖注入到Spring MVC控制器的最简单方法是，通过注解@Autowired到字 段或方法。Autowired 注解类型属于 oi]g.springframework.beaiis.factory.annotation 包。

此夕卜，为了能作为依赖注入，类必须要注明为©Service。该类型是oi^.springframework. stereotype包的成员。Service注解类型指示类是一个服务。此外，在配置文件中，还需要添加 —个<00叫。116111：-80013/^>元素来扫描依赖基本包。

<context:component-scan base-package="dependencyPackagen/>

下面以aunotated2应用进一步说明Spring MVC如何应用依赖注入。在aimotated2应用程 序中，ProductController类（见清单4.6)己经不同于annotatedl中的类。

清单 4.6 annotated2 中的 ProductController 类

package controller;

import java.math.BigDecimal;

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.PathVariable; import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.servlet,mvc.support.RedirectAttributes;

import domain.Product;

import form.ProductForm;

import service.ProductService;

SController

public class ProductController {

private static final Log logger = LogFactory •getLog(ProductController.class);

0Autowired

private ProductService productService;

0RequestMapping(value = "/input-product M) public String inputProduct() {

logger.info("inputProduct called"); return n ProductForm";

}

@RequestMapping (value = "/save-product 11, method = RequestMethod. POST) public String saveProduct(ProductForm productForm,

RedirectAttributes redirectAttributes) { logger.info("saveProduct called");

II no need to create and instantiate a ProductForm

// create Product

Product product = new Product();

product.setName{productForm.getName{));

product.setDescription(productForm.getDescription());

try {

product.setPrice(new BigDecimal(productForm.getPrice()));

} catch (NumberFormatException e) { // add product

Product savedProduct = productService.add(product);

redirectAttributes.addFlashAttribute("message",

"The product was successfully added."); return "redirect:/product-view/" + savedProduct.getld();

@RequestMapping(value = "/view-product/{id}M)

public String viewProduct(@PathVariable Long id, Model model) { Product product = productService.get(id);

model.addAttribute("product", product); return 11 ProductView";

}

}

与annotatedl中相比，annotated2中的ProductController类做了一系列的调整。首先是在 如下的私有字段上增加了 @Autowired注解：

@Autowired

private ProductService productService

ProductService是一个提供各种处理产品的方法的接口。为productService字段添加 @Autowired注解会使ProductService的一个实例被注入到ProductController实例中。

清单4.7和清单4.8分别显示了 ProductService接口及其实现类ProductServicelmpl。注意， 为了使类能被Spring扫描到，必须为其标注@861^6。

清单 4.7 ProductService 接口

package service

import domain.Product;

public interface ProductService {

Product add(Product product);

Product get(long id);

清单 4.8 ProductServicelmp丨类

package service; import j ava.math.BigDecimal; import java.util.HashMap; import j ava.util.Map;

import j ava.util.concurrent.atomic.AtomicLong; import org.springframework.stereotype.Service; import domain.Product;

0Service

public class ProductServiceImp1 implements ProductService {

private MapCLong, Product〉 products = new HashMap<Long, Product>(); private AtomicLong generator = new AtomicLong();

public ProductServicelmpl() {

Product product = new Product();

product.setName{MJX1 Power Drill"); product.setDescription(

"Powerful hand drill, made to perfection"); product.setPrice(new BigDecimal(129.99));

add(product);

0〇verride

public Product add(Product product) {

long newld = generator.incrementAndGet(); product.setld(newld); products.put(newld, product); return product;

}

SOverride

public Product get(long id) { return products.get(id);

}

}

如清单4.9所示，annotated2的Spring MVC配置文件中有两个<component-scan/>元素:

一个用于扫描控制器类，另一个用于扫描服务类。

清单4.9 Spring MVC配置文件

<?xml version="l.0M encoding=MUTF-8"?>

<beans xmlns=Mhttp: //www. springframework.org/schema/beans11 xmlns : xsi="http: //www. w3 . org/2001/XMLSchema-instance11 xmlns:p="<http://www.springframework.org/schema/pM>xmlns:mvc="<http://www.springframework.org/schema/mvcM>xmlns:context=M<http://www.springframework.org/schema/context>" xsi:schemaLocation=M

<http://www.springframework.org/schema/beans>

http://www,springframework.org/schema/beans/spring-beans.xsd <http://www.springframework.org/schema/mvc> http://www,springframework.org/schema/mvc/spring-mvc.xsd <http://www.springframework.org/schema/context>

http://www,springframework.org/schema/context/springcontext.xsdM>

<context:component-scan base-package=,lcontrollerM/>

<context:component-scan base-package="service"/>

<mvc:annotation-driven/>

<mvc:resources mapping="/css/\*\*" location=M/css/M/>

<mvc:resources mapping=n/\*.html" location=M/"/>

<bean id=,,viewResolver"

class="org.springframework.web.servlet.view.InternalResourceViewResolver"> 〈property name=,'prefix" value=M/WEB-INF/j sp/"/>

〈property name=Msuffix" value=”•jsp"/>

</bean>

</beans>

4.5重定向和Flash属性

作为一名经验丰富的servlet/JSP程序员，必须知道转发和重定向的区别。转发比重定向 快，因为重定向经过客户端，而转发没有。但是，有时采用重定向更好，若需要重定向到一 个外部网站，则无法使用转发。

使用重定向的另一个场景是避免在用户重新加载页面时再次调用同样的动作。例如，在 annotatedl中，当提交产品表单时，saveProduct方法将被调用，并执行相应的动作。在一个 真实的应用程序中，这可能包括将所述产品加入到数据库中。但是，如果在提交表单后重新 加载页面，saveProduct就会被再次调用，同样的产品将可能被再次添加。为了避免这种情况， 提交表单后，你可能更愿意将用户重定向到一个不同的页面。这个网页任意重新加载都没有 副作用。例如，在annotatedl中，可以在提交表单后，将用户重定向到一个ViewProduct页面。

在annotated!中，ProductController类中的saveProduct方法以如下所示的行结束：

return "redirect:/view-product/n + savedProduct.getld();

这里，使用重定向而不是转发来防止当用户重新加载页面时saveProduct被二次调用。

使用重定向的一个不便的地方是：无法轻松地传值给目标页面。而采用转发，则可以简 单地将属性添加到Model,使得目标视图可以轻松访问。由于重定向经过客户端，所以Model 中的一切都在重定向时丢失。好在，Spring 3.1版本以及更高版本通过Flash属性提供了一种 供重定向传值的方法。

要使用Flash属性，必须在Spring MVC配置文件中有一个<annotation-driven/>元素。然 后，还必须在方法上添加一个新的参数类型org.springframework.web.servlet.mvc.support. RedirectAttributes。清单 4.10 展示了更新后的 saveProduct 方法。

清单4.10使用Flash属性

0RequestMapping(value = "save-product", method = RequestMethod.POST) public String saveProduct(ProductForm productForm,

RedirecttAttributes redirectAttributes) {

logger.info("saveProduct called");

// no need to create and instantiate a ProductForm

II create Product

Product product = new Product();

product.setName(productForm.getName());

product.setDescription(productForm.getDescription());

try {

product.setPrice(new BigDecimal(productForm.getPrice())); } catch (NumberFormatException e) {

II add product

Product savedProduct = productService.add(product);

redirectAttributes.addFlashAttribute("message",

"The product was successfully added.1');

return "redirect:/ view\_product /" + savedProduct.getld();

4.6请求参数和路径变量

请求参数和路径变量都可以用于发送值给服务器。二者都是URL的一部分。请求参数采 用key=value形式，并用“&”分隔。例如，下面的URL带有一个名为productld的请求参数， 其值为3:

<http://localhost:8080/annotated2/view-product?productId=3>

在传统的servlet编程中，可以使用HttpServletRequest的getParameter方法来获取一个请

求参数值：

String productld = httpServletRequest.getParameter("productld”）；

Spring MVC提供了一个更简单的方法来获取请求参数值：使用org.springframework. web.bind.annotation.RequestParam注解类型来注解方法参数。例如，下面的方法包含了 一个获 取请求参数productld值的参数。

public void sendPr〇duct(@RequestParam int productld)

正如你所看到的，@RequestParam注解的参数类型不一定是字符串。

路径变量类似请求参数,但没有key部分，只是一个值。例如，在annotated2中，view-product 动作映射到如下URL:

/view-product/productId

其中的productld是表示产品标识符的整数。在Spring MVC中，productld称为路径变量，

用来发送一个值到服务器。

清单4.11中的viewProduct方法演示了一个路径变量的使用。

清单4.11使用路径变量

@RequestMapping(value = "/view-product/{id}")

public String viewProduct(@PathVariable Long id, Model model) {

Product product = productService.get(id); model.addAttribute("productn, product); return "ProductView";

}

为了使用路径变量，首先需要在RequestMapping注解的值属性中添加一个变量，该变量 必须放在花括号之间。例如，下面的RequestMapping注解定义了一个名为id的路径变量：

©RequestMapping(value = "/view-product/{id}")

然后，在方法签名中添加一个同名变量，并加i@PathVariable注解。注意清单4.11中 viewProduct的方法签名。当该方法被调用时，请求URL的id值将被复制到路径变量中，并 可以在方法中使用。路径变量的类型可以不是字符串。Spring MVC将尽力转换成一个非字符 串类型。这个Spring MVC的强大功能会在第5章中详细讨论。

可以在请求映射中使用多个路径变量。例如，下面定义了 userid和orderld两个路径变量。

QRequestMapping (value = ’'/view-product/ {userid} / {orderld}")

请直接在浏览器中输入如下URL，来测试viewProduct方法的路径变量。

<http://localhost:8080/annotated2/view-product/l>

有时，使用路径变量时会遇到一个小问题：在某些情况下，浏览器可能会误解路径变量。 考虑下面的URL。

<http://example.com/context/abc>

浏览器会（正确）认为abc是一个动作。任何静态文件路径的解析，如CSS文件，将 使用[http://example.com/context作为基本路径。这就是说，若服务器发送的网页包含如下img](http://example.com/context%e4%bd%9c%e4%b8%ba%e5%9f%ba%e6%9c%ac%e8%b7%af%e5%be%84%e3%80%82%e8%bf%99%e5%b0%b1%e6%98%af%e8%af%b4%ef%bc%8c%e8%8b%a5%e6%9c%8d%e5%8a%a1%e5%99%a8%e5%8f%91%e9%80%81%e7%9a%84%e7%bd%91%e9%a1%b5%e5%8c%85%e5%90%ab%e5%a6%82%e4%b8%8bimg)

元素：

<img src="logo•png"/>

该浏览器将试图通过 <http://example.com/context/logo.png> 来加载 logo.png。

然而，若一个应用程序被部署为默认上下文（默认上下文路径是一个空字符串），则对于 同一个目标的URL，会是这样的：

<http://example.com/abc>

下面是带有路径变量的URL:

<http://example.eom/abc/l>

在这种情况下，浏览器会认为abc是上下文，没有动作。如果在页面中使用<imgsrc=“l〇g〇. png”/>，浏览器将试图通过[http://example.com/abc/logo.png来寻找图像，并且它将找不到该图像。](http://example.com/abc/logo.png%e6%9d%a5%e5%af%bb%e6%89%be%e5%9b%be%e5%83%8f%ef%bc%8c%e5%b9%b6%e4%b8%94%e5%ae%83%e5%b0%86%e6%89%be%e4%b8%8d%e5%88%b0%e8%af%a5%e5%9b%be%e5%83%8f%e3%80%82)

好在，我们有一个简单的解决方案，即通过使用JSTL标记的URL (第8章中将会详细 讨论JSTL)。标签会通过正确解析URL来修复该问题。例如，annotated2中所有的JSP页面 导入的所有CSS，从

<style type=,,text/css,,>@import url (css/main. css) ;</style>

修改为

<style type="text/cssn>

0import url("<c:url value="/css/main.css"/>")；

</style>

若程序部署为默认上下文，链接标签会将该URL转换成如下所示形式：

<style type=,'text/cssn>@import url("/css/main.css");</style>

若程序不在默认上下文中，则它会被转换成如下形式：

<style type="text/cssn>@import url(n/annotated2/css/main.css");

</style>

* 1. @ ModelAttribute

前面谈到Spring MVC在每次调用请求处理方法时，都会创建Model类型的一个实例。 若打算使用该实例，则可以在方法中添加一个Model类型的参数。事实上，还可以使用在方 法中添加ModelAttribute注解类型来访问Model实例。该注解类型也是org.springframework. web.bind.annotation 包的成员。

可以用@ModelAttribute来注解方法参数或方法。带@ModelAttribute注解的方法会将其 输入的或创建的参数对象添加到Model对象中（若方法中没有显式添加)。例如，Spring MVC

将在每次调用submitOrder方法时创建一个Order实例。

0RequestMapping(method = RequestMethod.POST)

public String submitOrder(0ModelAttribute(nnewOrderM) Order order, Model model) {

输入或仓！J建的Order实例将用newOrder键值添加到Model对象中。如果未定义键值名， 则将使用该对象类型的名称。例如，每次调用如下方法，会使用键值order将Order实例添加 到Model对象中。

public String submitOrder(@ModelAttribute Order order, Model model)

@ModelAttribute的第二个用途是标注一个非请求的处理方法。被@ModelAttribute注解 的方法会在每次调用该控制器类的请求处理方法时被调用。这意味着，如果一个控制器类有 两个请求处理方法，以及一个有@M〇delAttribute注解的方法，该方法的调用次数就会比每个 处理请求方法更频繁。

Spring MVC会在调用请求处理方法之前调用注解的方法。带 @ModelAttribute注解的方法可以返回一个对象或一个void类型。如果返回一个对象，则返回 对象会自动添加到Model中。

@ModelAttribute

public Product addProduct(@RequestParam String productld) { return productService.get(productld);

}

若方法返回void，则还必须添加一个Model类型的参数，并自行将实例添加到Model中， 如下面的例子所示。

@ModelAttribute

public void populateModel(0RequestParam String id, Model model) er); model.addAttribute(new Account(id));

4.8小结

在本章中，我们学会了如何编写基于注解的控制器的SpringMVC应用，也学会了各种可 用来注解类、方法或方法的参数的注解类型。

5早

数据绑定和表单标签库

数据绑定是将用户输入绑定到领域模型的一种特性。有了数据绑定，类型总是为String 的HTTP请求参数，可用于填充不同类型的对象属性。数据绑定使得form bean (如前面各章 中的ProductForm实例）变成多余的。

为了髙效地使用数据绑定，还需要Spring的表单标签库。本章着重介绍数据绑定和表单 标签库，并提供范例，展示表单标签库中这些标签的用法。

5.1数据绑定概览

基于HTTP的特性，所有HTTP请求参数的类型均为字符串。在前面的章节中，为了获 取正确的产品价格，不得不将字符串解析成BigDecimal类型。为了便于复习，这里把第4章 中ProductController类的saveProduct方法的部分代码复制过来了。

@RequestMapping(value=nsave—product”）

public String saveProduct(ProductForm productForm, Model model) { logger.info("saveProduct called”）；

// no need to create and instantiate a ProductForm // create Product Product product = new Product(); product.setName(productForm.getName()); product.setDescription(productForm.getDescription()); try {

product.setPrice(new BigDecimal(productForm.getPrice()));

} catch (NumberFormatException e) {

}

之所以需要解析ProductForm中的price属性，是因为它是一个String，却需要用 BigDecimal来填充Product的price属性。有了数据绑定，就可以用下面的代码取代上面的 saveProduct方法部分。

@RequestMapping (value=,,save-productn)

public String saveProduct(Product product, Model model)

有了数据绑定，就不再需要ProductForm类，也不需要解析Product对象的price属性了。

数据绑定的另一个好处是：当输入验证失败时，它会重新生成一个HTML表单。手工编 写HTML代码时，必须记着用户之前输入的值，重新填充输入字段。有了 Spring的数据绑定 和表单标签库后，它们就会替你完成这些工作。

5.2表单标签库

表单标签库中包含了可以用在JSP页面中渲染HTML元素的标签。为了使用这些标签， 必须在JSP页面的开头处声明这个taglib指令。

<%0taglib prefix=,,forIn,,

uri="<http://www.springframework.org/tags/form>" %>

表5.1展示了表单标签库中的标签。

在接下来的小节中，将逐一介绍这些标签。5.3节展示了一个范例应用程序，展示了数据 绑定结合表单标签库的使用方法。

表5.1表单标签库中的标签

|  |  |
| --- | --- |
| 标签 | 描述 |
| form | 渲染表单元素 |
| input | 這染 <input type="text”/> 元素 |
| password | 渲染〈input type=”password”/i> 元素 |
| hidden | 植染〈input type=’’hidden7> 元素 |
| textarea | 濱染textarea元素 |
| checkbox | 這染一个<\_111; type=”checkbox’’/> 元素 |
| checkboxes | 渲染多个<\_1^ type=ncheckbox"/> 元素 |
| radiobutton | 植染一个<\_111; type=”radio”/> 元素 |
| radiobuttons | 潼染多个<1^说type=nradio7>元素 |
| select | 渲染一个选择元素 |
| option | 渲染一个可选元素 |
| options | 渲染一个可选元素列表 |
| errors | 在span元素中渲染字段错误 |

5.2.1表单标签

表单标签用于渲染HTML表单。要使用渲染一个表单输入字段的任何其他标签，必须有 一个form标签。表单标签的属性见表5.2。

表5.2中的所有标签都是可选的。这个表中没有包含HTML属性，如method和action。

C〇\_andName属性或许是其中最重要的属性，因为它定义了模型属性的名称，其中包含 了一个表单支持对象（formbacking object)，其属性将用于填充所生成的表单。如果该属性存 在，则必须在返回包含该表单的视图的请求处理方法中添加相应的模型属性。例如，在本章 配套的tags-demo应用程序中，下列表单标签是在BookAddFormjsp中定义的。

<form:form commandName=llbook" action^11 save-book" method=npost">

</form:form>

表5.2表单标签的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| acceptCharset | 定义服务器接受的字符编码列表 |
| commandName | 暴露表单对象之模型属性的名称，默认为command |
| cssClass | 定义要应用到被渲染form元素的CSS类 |
| cssStyle | 定义要应用到被渲染form元素的CSS样式 |
| htmlEscape | 接受true或者false，表示被渲染的值是否应该进行HTML转义 |
| modelAttribute | 暴露表单支持对象的模型属性名称，默认为command |

BookController类中的inputBook方法，是返回BookAddForm.jsp的请求处理方法。下面 就是inputBook方法。

0RequestMapping (value = 11 /input-book11) public String inputBook(Model model) {

model.addAttribute("book"7 new Book());

return "BookAddForm";

}

此处用book属性创建了一个Book对象，并添加到Model。如果没有Model属性， BookAddFormjsp页面就会抛出异常，因为表单标签无法找到在其commandName属性中指定 的 form backing object。

此外，一般来说，仍然需要使用action和method属性。这两个属性都是HTML属性， 因此不在表5.2之中。

1. input 标签

input标签植染〈input type="text">元素。这个标签最重要的属性是path，它将这个输入字段 绑定到表单支持对象的一个属性。例如，若随附<f〇rm/>标签的commandName属性值为book， 并且input标签的path属性值为isbn，那么，input标签将被绑定到Book对象的isbn属性。

表5.3展示了 input标签的属性。表5.3中的属性都是可选的，其中不包含HTML属性。

表5.3 i叩ut标签的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| cssClass | 定义要应用到被渲染input元素的CSS类 |
| cssStyle | 定义要应用到被渲染input元素的CSS样式 |
| cssErrorClass | 定义要应用到被渲染input元素的CSS类，如果bound属性中包含错误，则覆盖 cssClass属性值 |
| htmlEscape | 接受true或者false,表示是否应该对被渲染的值进行HTML转义 |
| path | 要绑定的属性路径 |

举个例子，下面这个input标签被绑定到表单支持对象的isbn属性。

<form:input id="isbn" path=nisbnM cssErrorClass="errorBox"/>

它将会被渲染成下面的<丨1^111/>元素：

<input type=MtextM id="isbn" name=’’isbn’’/>

cssErrorClass属性不起作用，除非isbn属性中有输入验证错误，并且采用同一个表单重 新显示用户输入，在这种情况下，input标签就会被植染成下面这个input元素。

<input type=,,textM id="isbn" name="isbn" class=’’errorBox"/>

input标签也可以绑定到嵌套对象的属性。例如，下列的input标签绑定到表单支持对象的 category属性的id属性。

<form:input path=Mcategory.idM/>

1. password 标签

password 标签遣染<1^111:1^6=>&88评〇^'/>元素，其属性见表 5.4。password 标签与 input

标签相似，只不过它有一个showPassword属性。

表5.4 password标签的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| cssClass | 定义要应用到被渲染input元素的CSS类 |
| cssStyle | 定义要应用到被渲染input元素的CSS样式 |
| cssEirorClass | 定义要应用到被渲染input元素的CSS类，如果bound属性中包含错误，则覆盖cssClass  属性值 |
| htmlEscape | 接受true或者false,表示是否应该对被渲染的值进行HTML转义 |
| path | 要绑定的属性路径 |
| showPassword | 表示应该显示或遮盖密码，默认值为false |

表5.4中的所有属性都是可选的，这个表中不包含HTML属性。下面是一个password标 签的例子。

<form:password id=MpwdM path=MpasswordM cssClass=MnormallV>

1. hidden 标签

hidden标签植染<^^111: type="hidden"/>元素，其属性见表5.5。hidden标签与input标签相 似，只不过它没有可视的外观，因此不支持cssClass和cssStyle属性。

表5.5 hidden标签的属性

|  |  |  |
| --- | --- | --- |
| 属性 | 描述 | |
| htmlEscape | 接受true或者false， | 表示是否应该对被渲染的值进行HTML转义 |
| path | 要绑定的属性路径 | |

表5.5中的所有属性都是可选的，其中不包含HTML属性。

下面是一个hidden标签的例子。

<form:hidden path="productId"/>

1. textarea 标签

textarea标签遣染一个HTML的textarea元素。Textarea实际上就是支持多行输入的一个 input元素。textarea标签的属性见表5.6。表5.6中的所有属性都是可选的，其中不包含HTML

属性。

表5.6 textarea标签的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| cssClass | 定义要应用到被渲染input元素的CSS类 |
| cssStyle | 定义要应用到被渲染input元素的CSS样式 |
| cssErrorClass | 定义要应用到被渲染input元素的CSS类，如果bound属性中包含错误，则覆盖cssClass  属性值 |
| htmlEscape | 接受true或者false，表示是否应该对被湞染的值进行HTML转义 |
| path | 要绑定的属性路径 |

例如，下面的textarea标签就是被绑定到表单支持对象的note属性。

<form:textarea path=MnoteM tabindex="4n rows=M5M cols="8〇"/>

1. checkbox 标签

checkbox 标签植染〈input type=”checkbox”/>元素^ checkbox 标签的属性见表 5.7。表 5.7

中的所有属性都是可选的，其中不包含HTML属性。

表5.7 checkbox标签的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| cssClass | 定义要应用到被渲染input元素的CSS类 |
| cssStyle | 定义要应用到被渲染input元素的CSS样式 |
| cssErrorClass | 定义要应用到被渲染input元素的CSS类，如果bound属性中包含错误，则覆盖cssClass 属性值 |
| htmlEscape | 接受true或者false，表示是否应该对被渲染的（多个）值进行HTML转义 |
| label | 要作为标签用于被渲染复选框的值 |
| path | 要绑定的属性路径 |

例如，下面的checkbox标签绑定到outOfStock属性：

<form: checkbox path=MoutOfStock" value=,,0ut of StockM/>

1. radiobutton 标签

radiobutton 标签遣染<input type=”radio”/>元素。radiobutton 标签的属性见表 5.8。表 5.8 中的所有属性都是可选的，其中不包含HTML属性。

例如，下列的radiobutton标签绑定到newsletter属性。

Computing Now <form:radiobutton path=Mnewsletter" value=MComputing Now"/> <br/>

Modern Health <form:radiobutton path=Mnewsletter" value="M〇dern Health"/>

表5.8 radiobutton标签的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| cssClass | 定义要应用到被渲染input元素的CSS类 |
| cssStyle | 定义要应用到被渲染input元素的CSS样式 |
| cssEirorClass | 定义要应用到被渲染input元素的CSS类，如果bound属性中包含错误，则覆盖cssClass 属性值 |
| htmlEscape | 接受true或者false,表示是否应该对被渲染的（多个）值进行HTML转义 |
| label | 要作为标签用于被渲染复选框的值 |
| path | 要绑定的属性路径 |

1. checkboxes 标签

checkboxes 标签植染多个<inputtype=ncheckbox7>元素。checkboxes 标签的属性见表 5.9。 表5.9中的属性都是可选的，其中不包含HTML属性。

例如，下面的checkboxes标签将model属性categoryList的内容遣染为复选框。checkboxes

标签允许进行多个选择。

<form:checkboxes path=Mcategoryn items=M${categoryList}"/>

表5.9 checkboxes标签的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| cssClass | 定义要应用到被渲染input元素的CSS类 |
| cssStyle | 定义要应用到被渲染input元素的CSS样式 |
| cssEirorClass | 定义要应用到被渲染input元素的CSS类，如果bound属性中包含错误，则覆盖cssClass 属性值 |
| delimiter | 定义两个input元素之间的分隔符，默认没有分隔符 |
| element | 给每个被渲染的input元素都定义一个HTML元素，默认为“span” |
| htmlEscape | 接受true或者false，表示是否应该对被渲染的（多个）值进行HTML转义 |
| items | 用于生成input元素的对象的Collection、Map或者Array |
| itemLabel | item属性中定义的Collection、Map或者Array中的对象属性，为每个input元素提供标签 |
| itemValue | item属性中定义的Collection、Map或者Array中的对象属性，为每个input元素提供值 |
| path | 要绑定的属性路径 |

1. radiobuttons 标签

radiobuttons 标签植染多个<input type=”radio"/>元素。radiobuttons 标签的属性见表 5.10。

例如，下面的radiobuttons标签将model属性categoryList的内容植染为单选按钮。每次

只能选择一个单选按钮。

<form:radiobuttons path=ncategory” items="${categoryList}"/>

表5.10 radiobuttons标签的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| cssClass | 定义要应用到被渲染input元素的CSS类 |
| cssStyle | 定义要应用到被渲染input元素的CSS样式 |
| cssErrorClass | 定义要应用到被植染input元素的CSS类，如果bound属性中包含错误，则覆盖cssClass 属性值 |
| delimiter | 定义两个input元素之间的分隔符，默认没有分隔符 |
| element | 给每一个被渲染的input元素都定义一个HTML元素，默认为“span” |
| htmlEscape | 接受true或者false,表示是否应该对被渲染的（多个）值进行HTML转义 |
| items | 用于生成input元素的对象的Collection、Map或者Array |
| itemLabel | item属性中定义的Collection、Map或者Array中的对象属性，为每个input元素提供标签 |
| itemValue | item属性中定义的Collection、Map或者Array中的对象属性，为每个input元素提供值 |
| path | 要绑定的属性路径 |

1. select 标签

select标签漬染一个HTML的select元素。被漬染元素的选项可能来自赋予其items属性 的一个Collection、Map、Array，或者来自一个嵌套的option或者options标签。select标签的 属性见表5.11。表5.11中的所有属性都是可选的，其中不包含HTML属性。

items属性特别有用，因为它可以绑定到对象的Collection、Map、Array，为select元素

生成选项。

例如，下面的select标签绑定到表单支持对象的category属性的id属性。它的选项来自 model属性categories。每个选项的值均来自categories collection/map/array的id属性，它的标 签来自name属性。

<form:select id=Mcategory" path=Mcategory.idM

items=M${categories}M itemLabel=nname" itemValue="idn/>

表5.11 select标签的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| cssClass | 定义要应用到被渲染input元素的CSS类 |
| cssStyle | 定义要应用到被渲染input元素的CSS样式 |
| cssErrorClass | 定义要应用到被渲染input元素的CSS类，如果bound属性中包含错误，则覆盖cssClass  属性值 |
| htmlEscape | 接受true或者false,表示是否应该对被渲染的（多个）值进行HTML转义 |
| items | 用于生成input元素的对象的Collection、Map或者Array |
| itemLabel | item属性中定义的Collection、Map或者Array中的对象属性，为每个input元素提供标签 |
| itemValue | item属性中定义的Collection、Map或者Array中的对象属性，为每个input元素提供值 |
| path | 要绑定的属性路径 |

1. option 标签

option标签植染select元素中使用的一个HTML的option元素，其属性见表5.12。表5.12

中的所有属性都是可选的，其中不包含HTML属性。

例如，下面是一个option标签的范例。

表5.12 option标签的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| cssClass | 定义要应用到被渲染input元素的CSS类 |
| cssStyle | 定义要应用到被渲染input元素的CSS样式 |
| cssErrorClass | 定义要应用到被渲染input元素的CSS类，如果bound属性中包含错误，则覆盖cssClass  属性值 |
| htmlEscape | 接受true或者false,表示是否应该对被渲染的（多个）值进行HTML转义 |

<form:select id=McategoryM path=Mcategory.idM items="${categories}" itemLabel=MnameM itemValue=Mid">

〈option value=n0">-- Please select --</option>

</form:select>

这个代码片断是濱染一个select元素，其选项来自model属性categories，以及option

标签。

1. options 标签

options标签生成一个HTML的option元素列表，其属性见表5.13,其中不包含HTML

属性。

tags-demo应用程序展示了一个options标签的范例。

表5.13 options标签的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| cssClass | 定义要应用到被渲染input元素的CSS类 |
| cssStyle | 定义要应用到被渲染input元素的CSS样式 |
| cssEirorClass | 定义要应用到被渲染input元素的CSS类，如果bound属性中包含错误，则覆盖cssClass  属性值 |
| htmlEscape | 接受true或者false,表示是否应该对被渲染的（多个）值进行HTML转义 |
| items | 用于生成input元素的对象的Collection、Map或者Array |
| itemLabel | item属性中定义的Collection、Map或者Array中的对象属性，为每个input元素提供标签 |
| itemValue | item属性中定义的Collection、Map或者Array中的对象属性，为每个input元素提供值 |

1. errors 标签

errors标签渲染一个或者多个HTML的span元素，每个span元素中都包含一个字段错误。 这个标签可以用于显示一个特定的字段错误，或者所有字段错误。

errors标签的属性见表5.14。表5.14中的所有属性都是可选的，其中不包含可能在HTML 的span元素中出现的HTML属性。

例如，下面这个errors标签显示了所有字段错误。

<form:errors path=M\*M/>

下面的errors标签显示了一个与表单支持对象的author属性相关的字段错误。

<form:errors path=MauthorM/>

|  |  |
| --- | --- |
| 表5.14 errors标签的属性 | |
| 属性 | 描述 |
| cssClass | 定义要应用到被渲染input元素的CSS类 |
| cssStyle | 定义要应用到被渲染input元素的CSS样式 |

|  |  |
| --- | --- |
|  | 续表 |
| 属性 | 描述 |
| Delimiter | 分隔多个错误消息的分隔符 |
| element | 定义一个包含错误消息的HTML元素 |
| htmlEscape | 接受true或者felse,表示是否应该对被渲染的（多个）值进行HTML转义 |
| path | 要绑定的错误对象路径 |

5.3数据绑定范例

在表单标签库中利用标签进行数据绑定的例子，参见tags-demo应用程序。这个范例围绕 domain类Book进行。这个类中有几个属性，包括一个类型为Category的category属性。 Category有id和name两个属性。

这个应用程序允许列出书目、添加新书，以及编辑书目。

5.3.1目录结构

图5.1中展示了 tags-demo的目录结构。

**&** webapp

|  |  |
| --- | --- |
|  | ^ & CSS  V&WE\_F v fe classes ^ ^controller  & BookController.dass y domain & Book-class &Catego「y.class v service  i&»B〇okService.class & BookServicelmpLdass w &conHg  springmvc-config.xml ▼ & jsp  B BookAddForm.jsp 1 BookEditForm.jsp H BookLisLjsp ► &lib Si web.xml  图5.1 tags-demo的目录结构 |

1. Domain 类

Book类和Category类是这个应用程序中的domain类，它们分别如清单5.1和清单5.2所示。

清单5.1 Book类

package domain;

import j ava,math.BigDecimal;

import j ava.io.Serializable;

public class Book implements Serializable {

private static final long serialVersionUID = 1L;

private long id;

private String isbn;

private String title;

private Category category;

private String author;

public Book() { public Book(long id, String isbn, String title,

Category category, String author, BigDecimal price) { this.id = id; this.isbn = isbn; this.title = title; this.category = category; this•author = author; this.price = price;

}

// get and set methods not shown

清单 5.2 Category 类

package domain;

import j ava.io.Serializable;

public class Category implements Serializable {

private static final long serialVersionUID = 1L; private int id; private String name;

public Category() { public Category(int id, String name) {

this.id = id; this.name = name;

// get and set methods not shown

}

1. Controller 类

下面的范例为Book提供了一个控制器：BookController类。它允许用户创建新书目、更 新书的详细信息，并在系统中列出所有书目。清单5.3中展示了 BookController类。

清单 5.3 BookController 类

package controller; import java,util.List;

import org.apache.commons.logging.Log; import org. apache . coimnons . logging. LogFactory;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Controller; import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.ModelAttribute; import org.springframework.web.bind.annotation.PathVariable; import org.springframework.web.bind.annotation.RequestMapping; import domain.Book; import domain.Category; import service.BookService;

@Controller

public class BookController {

@Autowired

private BookService bookService;

private static final Log logger =

LogFactory.getLog(BookController.class);

@RequestMapping(value = "/input-book ") public String inputBook(Model model) {

List<Category> categories = bookService.getAUCategories(); model. addAttribute ("categories11, categories); model.addAttribute("book", new Book()); return "BookAddForm";

}

0RequestMapping(value = "/edit-book/{id}M) public String editBook(Model model, @PathVariable long id) {

List<Category> categories = bookService.getAUCategories(); model,addAttribute(McategoriesM, categories);

Book book = bookService.get(id); model.addAttribute("book", book); return "BookEditForm";

}

@RequestMapping(value = "/save-book”）

public String saveBook(@ModelAttribute Book book) {

Category category =

bookService,getCategory(book.getCategory().getld()); book.setCategory(category); bookService,save(book); return "redirect:/list-book";

SRequestMapping(value = "/update-book")

public String updateBook(@ModelAttribute Book book) {

Category category =

bookService.getCategory(book.getCategory<).getld()); book.setCategory(category); bookService.update(book); return "redirect:/list-book";

@RequestMapping(value = "/list-book") public String listBooks(Model model) { logger.info("listBooks");

List<Book> books = bookService . getAUBooks {); model.addAttribute("booksM, books); return "BookList";

}

}

BookController依赖BookService进行一些后台处理。@Autowired注解用于给Book Controller 注入一个 BookService 实现。

@Autowired

private BookService bookService;

1. Service 类

清单5.4和清单5.5分别展示了 BookService接口和BookServicelmpl类。顾名思义， BookServicelmpl 就是实现 BookService。

清单 5.4 BookService 接口

package service; import java.util.List; import domain.Book; import domain.Category;

public interface BookService {

List<Category> getAUCategories (); Category getCategory(int id); List<Book> getAUBooks ();

Book save(Book book);

Book update(Book book);

Book get(long id); long getNextldO;

清单 5.5 BookServicelmp丨类

package service;

import j ava.util.ArrayList;

import j ava.util.List;

import org. springf ramework. stereotype. Servicer- import domain.Book; import domain.Category;

@Service

public class BookServicelmpl implements BookService {

"

\* this implementation is not thread-safe

"

private List<Category> categories; private List<Book> books;

public BookServicelmpl() {

categories = new ArrayList<Category>();

Category categoryl = new Category (1, ’'Computer1’）； Category category2 = new Category(2, "Travel”）； Category category3 = new Category(3, "Health"); categories.add(categoryl); categories.add(category2);

categories.add(category3);

books = new ArrayList<Book>();

books•add(new Book(1L, "9781771970273",

"Servlet & JSP: A Tutorial (2nd Edition)11, categoryl, "Budi Kurniawan1', new BigDecimal ("54.99M)); books.add(new Book(2L, ”9781771970297",

"C#: A Beginner's Tutorial (2nd Edition) ", categoryl, "Jayden Kyn, new BigDecimal("39.99")))；

@〇verride

public List<Category> getAUCategories () {

return categories;

0〇verride

public Category getCategory(int id) {

for (Category category : categories) { if (id == category.getld()) {

return category;

}

}

return null;

@〇verride

public List<Book> getAUBooks () { return books;

}

0〇verride

public Book save(Book book) { book.setId(getNextld{)); books.add(book); return book;

}

@〇verride

public Book get(long id) {

for (Book book : books) {

if (id == book.getldO ) { return book;

return null;

SOverride

public Book update(Book book) { int bookCount = books.size(); for (int i = 0; i < bookCount; i++) {

Book savedBook = books.get(i); if (savedBook.getId() == book.getld()) { books.set(i, book); return book;

}

}

return book;

}

SOverride

public long getNextldO {

// needs to be locked

long id = 〇L;

for (Book book : books) {

long bookld = book.getld(); if (bookld > id) { id = bookld;

}

}

return id + 1;

BookServicelmpl类中包含了一个Book对象的List和一个Category对象的List。这两个 List都是在实例化类时生成的。这个类中还包含了获取所有书目、获取单个书目，以及添加 和更新书目的方法。

5.3.5配置文件

清单5.6展示了 tags-demo中的Spring MVC配置文件。

清单5.6 Spring MVC配置文件

<?xml version="l.0M encoding=MUTF-8"?>

<beans xmlns=M<http://www.springframework.org/schema/beans>" xmlns : xsi="http: //www. w3 . org/2001/XMLSchema-instance11 xmlns:p="http://www.springframework•org/schema/p" xmlns:mvc="<http://www.springframework.org/schema/mvc>" xmlns:context=Mhttp://www.springframework,org/schema/context"

xsi:schemaLocation=M

<http://www.springframework.org/schema/beans>

http://www,springframework.org/schema/beans/spring-beans.xsd http://www,springframework.org/schema/mvc http://www.springframework,org/schema/mvc/spring-mvc,xsd <http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/spring-context>. xsdM>

<context:component-scan base-package="controllerM/>

<context:component-scan base-package="serviceM/>

...<!-- other elements are not shown -->

</beans>

component-scan bean 使得 app05a.controller 包和 app05a.service 包得以被扫描。

5.3.6视图

tags-demo中使用的3个JSP页面如清单5.7、清单5.8和清单5.9所示。BookAddFormjsp 和BookEditForm.jsp页面中使用的是来自表单标签库的标签。

清单 5.7 BookList.jsp 页面

<%@ taglib uri=n<http://java.sun.com/j> sp/j stl/core" prefix=McM %>

<!DOCTYPE HTML>

<html>

<head>

<title>Book List</title>

<style type=Mtext/css">@import url("<c:url value=M/css/main.css"/>M); </style>

</head>

<body>

<div id="globaln>

<hl>Book List</hl>

<a href="<c:url value=lVbook\_input"/>,l>Add Book</a>

<table>

<tr>

<th>Category</th>

<th>Title</th>

<th>ISBN</th>

<th>Author</th>

<th>Price</th>

<th>&nbsp;</th>

</tr>

<c:forEach items=n${books}M var=MbookM> <tr>

<td>${book.category.name}</td>

<td>${book,title}</td>

<td>${book.isbn}</td>

<td>${book.author}</td>

<td>${book.price}</td>

<td><a href=,,book\_edit/$ {book, id} n>Edit</a></td>

</tr>

</c:forEach>

</table>

</div>

</body>

</html>

清单 5.8 BookAddForm.jsp 页面

<%@ taglib prefix=Mform"

uri=M<http://www.springframework.org/tags/form>" %>

<%@ taglib uri=n<http://java.sun.com/j> sp/j stl/core" prefix=McM %> <!DOCTYPE HTML>

<html>

<head>

<title>Add Book Form</title>

<style type=Mtext/css">@import url("<c:url value=M/css/main.cssM/>n);</style>

</head>

<body>

<div id="globaln>

<form:form commandName=Mbook" action=Msave-book" method="post"> <fieldset>

<legend>Add a book</legend>

<P>

<label for=ncategory">Category: </label>

<form:select id=McategoryM path=Mcategory.idM items=M$ {categories}11 itemLabel=l,nameM itemValue=’’id"/>

</p>

<p>

<label for="title">Title: </label>

<form:input id="title" path=Mtitle"/>

</p>

<p>

<label for="author">Author: </label>

<form:input id="author" path="author"/>

</p>

<p>

<label for="isbn">ISBN: </label>

<f orm: input id="isbn" path=’’isbn"/>

</p> <p id="buttonsM>

<input id="reset" type="reset" tabindex=M4n>

<input id=11 submit11 type=Msubmit11 tabindex="5" value="Add Bookn>

</p>

</fieldset>

</form:form>

</div>

</body>

</html>

清单 5.9 BookEditForm.jsp 页面

<%@ taglib prefix=Mform"

uri=M<http://www.springframework.org/tags/form>" %>

<%@ taglib uri=n<http://java.sun.com/j> sp/j stl/core" prefix=McM %> <!DOCTYPE HTML>

<html>

<head>

<title>Edit Book Form</title>

<style type=Mtext/css">@import url("<c:url value=M/css/main.cssM/>n);</style>

</head>

<body>

<div id="global">

<c:url var=〃formAction〃 value=〃/update-book〃 />

<form:£orm commandNaine="book" action="${£ormAction}" method=MpostM>

<fieldset>

<legend>Edit a book</legend>

<form:hidden path=MidM/>

<P>

<label for=ncategory">Category: </label>

<form: select id=11 category" path=Mcategory. id" items=M$ {categories}"

itemLabel=,,nameM itemValue=nidM/>

</p>

<p>

<label for="title">Title: </label>

<f orm: input id="title’’ path=MtitleM/>

</p>

<p>

<label for=nauthorn>Author: </label>

<form:input id=,’author" path="author"/>

</p>

<p>

<label for=Misbn">ISBN: </label>

<f orm: input id=’’isbn’’ path=Misbnn/>

</p>

<p id="buttonsM>

<input id=Mreset" type="reset" tabindex=M4M>

<input id=Msubmit11 type=nsubmit11 tabindex=,,5n value=MUpdate Book">

</p>

</fieldset>

</form:form>

</div>

</body>

</html>

注意，在BookEditFormj sp页面中，表单的action属性为<c :url />的值：

<c:url var=MformActionM value =M/update-bookM/>

<form:forin commandName=Mbook" action=M${formAction}M method=MpostM>

这是因为表单需要定位/update-book映射，并且给action属性一个静态值“update-book”

是有问题的。如果图书ID作为请求参数发送到编辑图书页面，则页面URL将如下所示：

http: //d〇jnain/context/edit-book?id=l

该表单将正确提交到 [http://domain/context/update-book〇](http://domain/context/update-book%e3%80%87)

但是，如果图书ID作为路径变量发送，页面网址将如下所示：

**http:** *//*domain/context/edit-boo'k/Id

并且表单将被提交到：

<http://domain/context/edit-book/update-book>

因此，您应该使用<c:url>来确保表单目标始终正确，无论网页网址如何。遗憾的是，表单的 action属性不能取<c:url/>。因此，您需要创建一个变量formAction并从action属性引用它。

5.3.7测试应用

要想测试这个应用程序范例，请打开以下网页：

<http://localhost:8080/tags-demo/list-books>

图5.2展示了第一次启动这个应用程序时显示的书目列表。

单击“AddBook”链接添加书目，或者单击书籍详情右侧的“Edit”链接来编辑书目。 图5.3展示了 “Add a book”表单。图5.4中展示了 “Edit a book”表单。



W 0 Add Book Form

C 丨 oca 祖 host

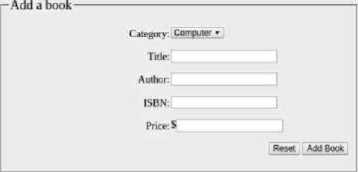


图 **5.3 Add a book** 表单

Pr@Edlt Book Form"™

夺 C l〇calh〇5tS'N：；/Dc；'：MlPrno.rJiC-DOok/Q t? O H • 9 芎



图 **5.4 Edit a book** 表单

5.4小结

本章介绍了数据绑定和表单标签库中的标签。第6章和第7章将讨论如何进一步将数据 绑定与Converter、Formatter以及验证器结合起来使用。

*m*

*o*

6早

转换器和格式化

第5章已经见证了数据绑定的威力，并学习了如何使用表单标签库中的标签。但是，Spring 的数据绑定并非没有任何限制。有案例表明，Spring在如何正确绑定数据方面是杂乱无章的。 例如，Spring总是试图用默认的语言区域将日期输入绑定到java.util.Date。假如想让Spring 使用不同的日期样式，就需要用一个Converter (转换器）或者Formatter (格式化）来协助 Spring完成。

本章着重讨论Convertei•和Formatter的内容。这两者均可用于将一种对象类型转换成另 一种对象类型。Converter是通用元件，可以在应用程序的任意层中使用，而Formatter则是专 门为Web层设计的。

本章有两个示例程序：converter-demo和formatter-demo。两者都使用一个messageSource bean来帮助显示受控的错误消息，这个bean的功能在第10章中介绍。

1. Converter

Spring的Converter是可以将一种类型转换成另一种类型的一个对象。例如，用户输入的 日期可能有许多种形式，如“December 25, 2014” “12/25/2014”和“2014-12-25”，这些都表 示同一个日期。默认情况下，Spring会期待用户输入的日期样式与当前语言区域的日期样式 相同。例如，对于美国的用户而言，就是月/日/年格式。如果希望Spring在将输入的日期字符 串绑定到LocalDate时，使用不同的日期样式，则需要编写一个Converter，才能将字符串转 换成日期。java.time.LocalDate类是Java 8的一个新类型，用来替代java.util.Date。还需使用 新的Date/Time API来替换旧有的Date和Calendar类。

为了仓键 Converter，必须编写实现 org.springframeworkxorexonvertxonverter. Converter 接口的一个Java类。这个接口的声明如下：

public interface Converter<S, T>

这里的S表示源类型，T表示目标类型。例如，为了创建一个可以将Long转换成Date

的Converter，要像下面这样声明Converter类：

public class MyConverter implements Converter<Long, LocalDate> {

}

在类实体中，需要编写一个来自Converter接口的convert方法实现。这个方法的签名 如下：

T convert(S source)

例如，清单6.1展示了一个适用于任意日期样式的Converter。

清单 6.1 String To LocalDate Converter

package converter;

import j ava.time.LocalDate;

import j ava.time.format.DateTimeFormatter;

import j ava.time.format.DateTimeParseException;

import org.springframework.core.convert.converter.Converter;

public class StringToLocalDateConverter implements C〇nverter<String, LocalDate> {

private String datePattern;

public StringToLocalDateConverter(String datePattern) { this.datePattern = datePattern;

@Override

public LocalDate convert(String s) { try {

return LocalDate.parse(s, DateTimeFormatter.ofPattern( datePattern));

} catch (DateTimeParseException e) {

// the error message will be displayed in <form:errors> throw new IllegalArgumentException(

”invalid date format. Please use this patternV’” + datePattern + "V"’）；

注意清单6.1中的Converter方法，它利用传给构造器的日期样式，将一个String转换成 LocalDate〇

为了使用Spring MVC应用程序中定制的Converter，識在Spring MVC配置文件中编写 一个名为 conversionService 的 bean。bean 的类名称必须为 org.springframework.context. support.ConversionServiceFactoryBean。这个 bean 必须包含一个 converters 属性，它将列出要 在应用程序中使用的所有定制Converter。例如，下面的bean声明在清单6.1中注册了

StringToDateConverter 〇

<bean id=,'conversionService" class=,,org. springf ramework. context. support. ^♦ConversionServiceFactoryBean,,>

〈property name=”converters"〉

<list>

<bean class=Mconverter.StringToLocalDateConvertern> <constructor-arg type=M j ava. lang. String1' value=,,MM\_dd\_yyyy" />

</bean>

</list>

</property>

</bean>

随后，要给annotation-driven元素的conversion-service属性赋值bean名称（本例中是 conversionService ),如下所示：

<mvc:annotation-driven

conversion-service=nconversionService"/>

converter-demo 是一个范例应用程序，它利用 StringToLocalDateConverter 将 String 转换成 Employee对象的birthDate属性。Employee类如清单6.2所示。

清单 **6.2 Employee** 类

package domain;

import java.io.Serializable;

import j ava.time.LocalDate;

public class Employee implements Serializable {

private static final long serialVersionUID = -908L;

private long id;

private String firstName;

private String lastName;

private LocalDate birthDate;

private int salaryLevel;

// getters and setters not shown

}

清单6.3中的EmployeeController类是domain对象Employee的控制器。

清单 6.3 converter-demo 中的 EmployeeController 类

package controller;

import org.springframework.stereotype.Controller; import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

import org.springframework.validation.FieldError;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.RequestMapping;

import domain.Employee;

@Controller

public class EmployeeController {

@RequestMapping(value=M/add-employeeM) public String inputEmployee(Model model) { model,addAttribute(new Employee()); return "EmployeeForm";

0RequestMapping(value=M/save-employee")

public String saveEmployee(@ModelAttribute Employee employee, BindingResult bindingResult, Model model) {

if (bindingResult.hasErrors()) {

FieldError fieldError = bindingResult.getFieldError{); return "EmployeeForm";

}

// save employee here

model. addAttribute ("employee11, employee); return "EmployeeDetails";

}

}

EmployeeController 类有 inputEmployee 和 saveEmployee 两个处理请求的方法。 inputEmployee方法返回清单6.4中的EmployeeFormjsp页面。saveEmployee方法取出在提交 Employee 表单时创建的一个 Employee 对象。有了 StringToLocalDateConverter converter，就

不需要劳驾控制器类将字符串转换成日期了

saveEmployee方法的BindingResult参数中放置了 Spring的所有绑定错误。该方法利用 BindingResult记录所有绑定错误。绑定错误也可以利用errors标签显示在一个表单中，如 EmployeeFormjsp 页面所示。

清单 6.4 EmployeeForm.jsp 页面

<%@ taglib prefix=MformMuri="<http://www.springframework.org/tags/form>" %> <%@ taglib uri=n<http://java.sun.com/j> sp/j stl/core" prefix=McM %>

<!DOCTYPE HTML>

<html>

<head>

<title>Add Employee Form</title>

<style type=,,text/css">@import url ("<c:url value=’’/css/main• css"/>") ;</style>

</head>

<body>

<div id="globaln>

<form: form coimnandName=11 employee" action="save-employee" method= Mpostn> <fieldset>

<legend>Add an employee</legend>

<P>

<label for="firstNameM>First Name: </label>

<form:input path=MfirstName" tabindex=M1M/>

</p>

<p>

<label for=MlastNameM>First Name: </label>

<f orm: input path=111 astNameM tabindex=M2M/>

</p>

<P>

<form:errors path=MbirthDateM cssClass="error11/>

</p>

<p>

<label for=llbirthDateM>Date Of Birth: </label>

<form:input path=MbirthDateM tabindex=M3M />

</p>

<p id="buttonsM>

<input id="reset" type="reset" tabindex=M4n>

<input id=Msubmit" type=Msubmit" tabindex="5n value=MAdd Employee">

</p>

</fieldset>

</form:form>

</div>

</body>

</html>

在浏览器中打开以下URL，可以对这个转换器进行测试：

<http://localhost:8080/converter-demo/add-employee>

试着输入一个无效的日期，将会被转到同一个Employee表单，并且可以在表单中看到错 误消息，如图6.1所示。

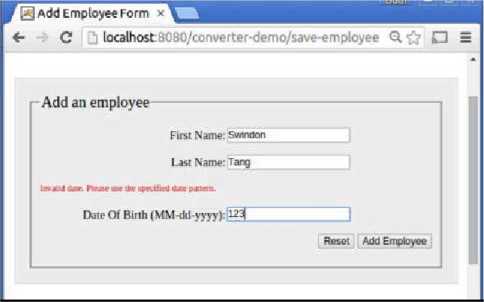


图6.1 Enqjloyee表单中的转换错误

1. Formatter

Formatter就像Converter—样，也是将一种类型转换成另一种类型。但是，Formatter的 源类型必须是一个String，而Converter则适用于任意的源类型。Formatter更适合Web层，而 Converter则可以用在任意层中为了转换Spring MVC应用程序表单中的用户输入，始终应 该选择 Formatter，而不是 Converter。

为 了创建 Formatter，要编写一个实现 oi^.spriiigfi\*amework.fomiat.Fomiatter 接口的 Java 类。 下面是这个接口的声明：

public interface Formatter<!T>

这里的T表示输入字符串要转换的目标类型^该接口有parse和print两个方法，所有实 现都必须覆盖它们。

T parse(String text, java.util.Locale locale)

String print(T object, java.util.Locale locale)

parse方法利用指定的Locale将一个String解析成目标类型。print方法与之相反，它返回

目标对象的字符串表示法。

例如，formatter-demo应用程序中用一个LocalDateFormatter将String转换成Date。其作 用与 converter-demo 中的 StringToLocalDateConverter—样。

DateFormatter类如清单6.5所示。

清单 6.5 Local DateFormatter 类

package formatter;

import java.text.ParseException;

import j ava.time.LocalDate;

import j ava.time.format.DateTimeFormatter;

import j ava.time.format.DateTimeParseException;

import j ava.util.Locale;

import org.springframework.format.Formatter;

public class LocalDateFormatter implements Formatter<LocalDate> {

private DateTimeFormatter formatter; private String datePattern;

public LocalDateFormatter(String datePattern) { this.datePattern = datePattern;

formatter= DateTimeFormatter.ofPattern(datePattern);

0〇verride

public String print(LocalDate date, Locale locale) { return date.format(formatter);

0〇verride

public LocalDate parse(String s, Locale locale) throws ParseException { try {

return LocalDate.parse(s7

DateTimeFormatter,ofPattern(datePattern));

} catch (DateTimeParseException e) {

// the error message will be displayed in <form:errors> throw new IllegalArgumentException(

"invalid date format. Please use this pattern\M + datePattern + M\M");

为了在Spring MVC应用程序中使用Formatter，需要利用名为conversionService的bean 对它进行注册〇 bean 的类名称必须为 org.springframework.format.support.FormattingConversion ServiceFactoryBean。这与converter-demo中用于注册converter的类不同。这个bean可以用一 个formatters属性注册formatter,用一个converters属性注册converter。清单6.6展示了 formatter-demo 的 Spring 配置文件。

清单 6.6 formatter-demo 的 Spring 配置文件

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns=M<http://www.springframework.org/schema/beans>" xmlns :xsi=,,http: //www. w3 . org/2001/XMLSchema-instance,, xmlns:p="<http://www.springframework.org/schema/pM>xmlns :mvc=,,http: //www. springf ramework. org/schema/mvc" xmlns:context=M<http://www.springframework.org/schema/contextM>xsi: schemaLocationy

<http://www.springframework.org/schema/beans>

<http://www.springframework.org/schema/beans/spring-beans.xsd> <http://www.springframework.org/schema/mvc> <http://www.springframework.org/schema/mvc/spring-mvc.xsd> <http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/spring-context.xsdM>>

<context:component-scan base-package="controllerM/>

<context:component-scan base-package^"formatter"/>

<mvc:annotation-driven conversion-service="conversionServiceM/>

<mvc:resources mapping=M/css/\*\*" location="/css/M/>

<mvc: resources mapping=M/\* .html1' location:’’/”/>

<bean id="viewResolver"

class="org.springframework.web.servlet.view. ^♦internalResourceViewResolver1^

<property name="prefix" value=M/WEB-INF/jsp/M />

<property name=,,suf f ixM value=M . j spM />

</bean> <bean id=MconversionService"

class=Morg.springframework.format.support. ^♦FormattingConversionServiceFactoryBeanM>

〈property name=”formatters’’〉

<set>

<bean class=,,formatter.LocalDateFormatter"> <constructor-arg type="j ava.lang.String" value=ffMM-dd-yyyy" />

</bean>

</set>

</property>

</bean>

</beans>

注意，还需要给这个Formatter添加一个component-scan元素。

在浏览器中打开下面的URL，可以测试app06b中的Formatter:

<http://localhost:8080/formatter-demo/add-employee>

1. 用 Registrar 注册 Formatter

注册Formatter的另一种方法是使用Registrar。例如，清单6.7就是注册DateFormatter的

一个例子。

清单 6.7 MyFormatterRegistrar 类

package formatter;

import org.springframework.format.FormatterRegistrar; import org.springframework.format.FormatterRegistry;

public class MyFormatterRegistrar implements FormatterRegistrar {

private String datePattern;

public MyFormatterRegistrar(String datePattern) { this.datePattern = datePattern;

}

0〇verride

public void registerFormatters(FormatterRegistry registry) {

registry.addFormatter(new LocalDateFormatter(datePattern));

// register more formatters here

}

}

有了 Registrar，就不需要在Spring MVC配置文件中注册任何Formatter 了，只在Spring 配置文件中注册Registrar就可以了，如清单6.8所示。

清单 6.8 在 springmvc-config.xml 文件中注册 Registrar

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="<http://www.springframework.org/schema/beans>" xmlns:xsi="<http://www.w3.org/2> 0 01/XMLSchema-instance" xmlns:p="<http://www.springframework.org/schema/p>" xmlns:mvc="<http://www.springframework.org/schema/mvc>" xmlns:context="<http://www.springframework.org/schema/context>" xsi:schemaLocation="

<http://www.springframework.org/schema/beans>

<http://www.springframework.org/schema/beans/spring-beans.xsd>

<http://www.springframework.org/schema/mvc>

<http://www.springframework.org/schema/mvc/spring-mvc.xsd>

<http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/springcontext.xsd>">

<context:component-scan base-package="controller" /> <context:component-scan base-package="formatter" />

<mvc:annotation-driven conversion-service=MconversionService" />

<mvc:resources mapping="/css/\*\*" location="/css/" />

<mvc:resources mappings"/\*.html" location^"/" />

<bean id="viewResolver"

class="org.springframework.web.servlet.view. ^■►internalResourceA/iewResolver n>

〈property name="prefix" value="/WEB-INF/jsp/" />

<property name="suffix" value=".jsp" />

</bean> <bean id="conversionService"

class="org.springframework.format.support.

^■►FormattingConversionServiceFactoryBean">

<property name="formatterRegistrars">

<set>

<bean class="formatter.MyFormatterRegistrar"> <constructor-arg type="java.lang.String" value="MM-dd-yyyy" />

</bean>

</set>

</property>

</bean>

</beans>

6.4 遍馨 Converter,还是 Formatter

Converter是一般工具，可以将一种类型转换成另一种类型。例如，将String转换成 LocalDate，或者将Long转换成LocalDate。Converter既可以用在Web层，也可以用在其他

层中。

Formatter只能将String转换成另一种Java类型。例如，将String转换成LocalDate，但 它不能将Long转换成LocalDate。因此，Formatter适用于Web层。为此，在Spring MVC应 用程序中，选择Formatter比选择Converter更合适。

6.5小结

本章学习了 Converter和Formatter，可以利用它们来引导Spring MVC应用程序中的数据 绑定。Converter是一般工具，可以将任意类型转换成另一种类型，而Formatter则只能将String 转换成另一种Java类型。Formatter更适用于Web层。

第7章 验证器

输入验证是Spring处理的最重要Web开发任务之一。在Spring MVC中，有两种方式可 以验证输入，即利用Spring自带的验证框架，或者利用JSR 303实现。本章将详细介绍这两 种输入验证方法。

本章用两个不同的示例分别介绍两种方式：spring-validator和jsr303-validator。

7.1验证概览

Converter和Formatter作用于字段级。在MVC应用程序中，它们将String转换或格式化 成另一种Java类型，如java.time.LocalDate。验证器则作用于对象级。它决定某一个对象中的 所有字段是否均是有效的，以及是否遵循某些规则。一个典型的Spring MVC应用会同时应用 至丨J formatters/converters 和 validators。

如果一个应用程序中既使用了 Formatter，又有validator (验证器)，那么，应用中的事件 顺序是这样的：在调用Controller期间，将会有一个或者多个Formatter，试图将输入字符串 转换成domain对象中的field值。一旦格式化成功，验证器就会介入。

例如，Order对象可能会有一个shippingDate属性（其类型显然为LocalDate)，它的 值绝对不可能早于今天的日期。当调用OrderController时，DateFormatter会将字符串转 化成Date，并将它赋予Order对象的shippingDate属性。如果转换失败，用户就会被转 回到前一个表单。如果转换成功，则会调用验证器，查看shippingDate是否早于今天的 日期。

现在，你或许会问，将验证逻辑转移到LocalDateFormatter中是否更加明智？因为比较 一下日期并非难事，但答案却是否定的。首先，LocalDateFormatter还可用于将其他字符串

格式化成日期，如birthDate或者purchaseDate。这两个日期的规则都不同于shippingDate。 事实上，比如，员工的出生日期绝对不可能晚于今日。其次，验证器可以检验两个或更多 字段之间的关系，各字段均受不同的Formatter支持。例如，假设Employee对象有birthDate 属性和startDate属性，验证器就可以设定规则，使任何员工的入职日期均不可能早于他的 出生日期。因此，有效的Employee对象必须让它的birthDate属性值早于其startDate值。这 就是验证器的任务。

1. Spring验证器

从一开始，Spring就设计了输入验证，甚至早于JSR 303 (Java验证规范）。因此，Spring 的Validation框架至今都很普遍，尽管对于新项目，一般建议使用JSR 303验证器。

为 了创建 Spring 验证器，要实现 org.springframework.validation.Validator 接口。这个接口 如清单7.1所示，其中有supports和validate两个方法。

清单 7.1 Spring 的 Validator 接口

package org.springframework.validation; public interface Validator {

boolean supports(Class<?> clazz);

void validate(Object target, Errors errors);

}

如果验证器可以处理指定的Class，supports方法将返回true。validate方法会验证目标对 象，并将验证错误填入Errors对象。

Errors 对象是 org.spring£ramework.validation.Errors 接口 的一个实例。Errors 对象中包含了 一系列FieldError和ObjectError对象。FieldError表示与被验证对象中的某个属性相关的一个 错误。例如，如果产品的price属性必须为负数，并且Product对象被验证为负数，那么就需 要创建一个FieldError。例如，在欧洲出售的一本Book,却在美国的网店上购买，那么就会 出现一个 ObjectError。

编写验证器时，不需要直接创建Error对象，因为实例化ObjectError或FieldError花费了 大量的编程精力。这是因为ObjectError类的构造器需要4个参数，FieldError类的构造器则需 要7个参数，如以下构造器签名所示：

ObjectError(String objectName, String[] codes, Object[] arguments,

String defaultMessage)

FieldError(String objectName, String field, Object rejectedValue,

boolean bindingFallure, String[] codes, Object[] arguments,

String defaultMessage)

给Errors对象添加错误的最容易的方法是:在Errors对象上调用一个reject或者rejectValue 方法。调用 reject，会往 FieldError 中添加一个 Objecffirror 和 rejectValue。

下面是reject和rejectValue的部分方法重载：

void reject(String errorCode)

void reject(String errorCode, String defaul tMessage) void rejectValue{String field, String errorCode) void rejectValue(String field, String errorCode,

String defaul tMessage)

大多数时候，只给reject或者rejectValue方法传入一个错误码，Spring就会在属性文件中 查找错误码，获得相应的错误消息。还可以传入一个默认消息，当没有找到指定的错误码时， 就会使用默认消息。

Errors对象中的错误消息，可以利用表单标签库的Errors标签显示在HTML页面中。 错误消息可以通过Spring支持的国际化特性本地化。关于国际化的更多信息，请查看第 10章。

1. ValidationlMs 类

org.springframework.validation.ValidationUtils 类是一个工具，有助于编写 Spring 验证器。

不需要像下面这样编写：

if (firstName == null || firstName.isEmpty{)) {

errors.rej ectValue("price");

}

而是可以利用ValidationlMs类的rejectlfEmpty方法，像下面这样：

ValidationUtils•rej ectlfEmpty <"price");

或者下面这样的代码：

if (firstName == null || firstName.trim().isEmpty()) {

errors • re j ectValue (’’price1');

可以编写成：

ValidationUtils•rejectlfEmptyOrWhitespace("price”)；

下面是 validationUtils 中 rejectlffimpty 和 rejectlffimptyOrWhitespace 方法的方法重载:

public static void rejectlfEmpty(Errors errors, String field,

String *errorCode)*

public static void rejectlfEmpty(Errors errors, String field,

String *errorCode,* Object[] *errorArgs)*

public static void rejectlfEmpty(Errors errors, String field,

String *errorCode,* Object[] *errorArgs,* String *defaultMessage)*

public static void rejectlfEmpty(Errors errors, String field,

String *errorCode,* String *defaultMessage)*

public static void rejectlfEmptyOrWhitespace(Errors errors,

String *field,* String *errorCode)*

public static void rejectlfEmptyOrWhitespace(Errors errors.

String *field,* String *errorCode,* Object[] *errorArgs)*

public static void rejectlfEmptyOrWhitespace(Errors errors,

String *field,* String *errorCode,* Object[] *errorArgs,*

String *defaultMessage)*

public static void rejectlfEmptyOrWhitespace(Errors errors,

String *field,* String *errorCode,* String *defaultMessage)*

此外，ValidationUtils还有一个invoke Validator方法，用来调用验证器。

public static void invokeValidator(Validator validator,

Object obj, Errors errors)

接下来的小节将通过范例来介绍如何使用这个工具。

1. **Spring** 的 **Validator** 范例

spring-validator应用程序中包含一个名为Product Validator的验证器，用于验证Product对

象。Spring-validator 的 Product 类如清单 7.2 所示。ProductValidator 类如清单 7.3 所示。

清单 **7.2 Product** 类

package domain;

import j ava.io.Serializable;

import java.math,BigDecimal;

import j ava.time.LocalDate;

public class Product implements Serializable {

private static final long serialVersionUID = 1L;

private String name;

private String description;

private BigDecimal price;

private LocalDate productionDate;

//getters and setters methods not shown

清单 7.3 ProductValidator 类

package validator;

import j ava.math,BigDecimal;

import j ava.time.LocalDate;

import org.springframework.validation.Errors; import org.springframework.validation.ValidationUtils; import org.springframework.validation.Validator; import domain.Product;

public class ProductValidator implements Validator { @〇verride

public boolean supports(Class<?> klass) {

return Product.class.isAssignableFrom(klass);

@〇verride

public void validate(Object target, Errors errors) {

Product product = (Product) target;

ValidationUtils.rejectlfEmpty(errors, "namen,

"productname.required");

ValidationUtils•rejectIfEmpty(errors, "price","price•required"); ValidationUtils•rejectIfEmpty(errors, "productionDate", "productiondate.required");

BigDecimal price = product.getPrice{);

if (price != null && price.compareTo(BigDecimal.ZERO) < 0) {

errors,rejectValue(MpriceM, "price■negative");

}

Local Date productionDate = product.getProductionDate(); if (productionDate != null) {

if (productionDate.isAfter(LocalDate.now())) { errors.rej ectValue("productionDate",

Mproductiondate.invalid");

ProductValidator验证器是一个非常简单的验证器。它的validate方法会检验Product是否

有名称和价格，并且价格是否不为负数。它还会确保生产日期不晚于今天。

7.5源文件

验证器不需要显式注册，但是如果想要从某个属性文件中获取错误消息，则需要通过声 明messageSource bean，告诉Spring**要去哪里查找这个文件。下面是**app07a**中的** messageSource bean：

<bean id="messageSource" class="org.springframework.context.support.

^♦ReloadableResourceBundleMessageSourceM>

〈property name=’’basename" value="/WEB-INF/resource/messages"/>

</bean>

这个bean实际上是说，错误码和错误消息可以在/WEB-INF/resource目录下的messages, properties文件中找到。

清单7.4展示了 messages.properties文件的内容。

清单 7.4 messages.properties 文件

productname.required.product.name=Please enter a product name

price.required=Please enter a price

price.negative=Price cannot be less than 0

productiondate.required=Please enter a production date

productiondate.invalid=Please ensure the production date is not later

^♦than today

typeMismatch.productionDate=Invalid production date

**7.6 Controller** 类

在Controller类中通过实例化validator类，可以使用Spring验证器。清单7.5中Product- Controller 类的 saveProduct 方法创建 了一个 ProductValidator， 并调用其 validate 方法。 为了检 验该验证器是否生成错误消息，需在BindingResult中调用hasErrors方法。

清单 7.5 ProductController 类

package controller;

import org.apache.commons.logging.Log; import org.apache.commons.logging.LogFactory; import org.springframework.stereotype.Controller; import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

import org.springframework.validation.FieldError;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.RequestMapping;

import domain.Product;

import validator.ProductValidator;

@Controller

public class ProductController {

private static final Log logger = LogFactory .getLog(ProductController.class);

@RequestMapping(value = "/add-product") public String inputProduct(Model model) {

model.addAttribute("product", new Product()); return n ProductFormn;

}

@RequestMapping(value = "/save-product")

public String saveProduct(@ModelAttribute Product product, BindingResult bindingResult, Model model) { ProductValidator productValidator = new ProductValidator(); productValidator.validate(product, bindingResult);

if (bindingResult.hasErrors()) {

FieldError fieldError = bindingResult.getFieldError(); logger.debug("Code:M + fieldError.getCode() + n, field:M

+ fieldError.getField());

return "ProductForm11 ;

// save product here

model.addAttribute("product", product); return "ProductDetails11 ;

使用Spring验证器的另一种方法是：在Controller中编写initBinder方法，并将验证器传 到 WebDataBinder，并调用其 validate 方法。

@org.springframework.web.bind.annotation.InitBinder public void initBinder{WebDataBinder binder) {

// this will apply the validator to all request-handling methods binder.setValidator(new ProductValidator{)); binder.validate();

}

将验证器传到WebDataBinder,会使该验证器应用于Controller类中所有处理请求的方法。 或者利用@javax.validation. Valid对要验证的对象参数进行注解。例如：

public String saveProduct(@Valid @ModelAttribute Product product, BindingResult bindingResult, Model model) {

Valid注解类型是在JSR 303中定义的。关于JSR 303的相关信息，7.8节将会讨论。

7.7测试验证器

要想测试spring-validator中的验证器，在浏览器中打开以下URL:

<http://localhost:8080/spring-validator/add-product>

你将会看到一张空白的Product表。如果单击“Add Product”按钮，没有输入任何值，将 会被转回Product表，并且这次验证器会显示出错消息，如图7.1所示。

W0AdTpr〇duct Form

C LJ localhost： WBO/spring validator/save-product 玟☆ □ E



图 7.1 ProductValidatoi•的效果图

1. **JSR303** 验证

JSR 303 “Bean Validation”（发布于 2009 年 11 月）和 JSR 349 “Bean Validation 1.1 ”（发 布于2013年5月）指定了一整套API，通过注解给对象属性添加约束。JSR 303和JSR 349 可以分别从以下网址下载：

<http://j> cp.org/en/j sr/detail?id=303

<http://j> cp.org/en/j sr/detail?id=34 9

当然，JSR只是一个规范文档，本身用处不大，除非编写了它的实现。对于JSR bean validation，目前有两个实现。第一个实现是Hibernate Validator，目前版本为5, JSR 303和JSR 349它都实现了，可从以下网站下载：

<http://sourceforge.net/proj> ects/hibernate/files/hibernate-validator/

第二个实现是Apache BVal,可从以下网站下载：

<http://bval.apache.org/downloads.html>

JSR 303不需要编写验证器，但要利用JSR 303注解类型嵌入约束。JSR 303约束见 表 7.1。

表7.1 JSR303约束

|  |  |  |
| --- | --- | --- |
| 属性 | 描述 | 范例 |
| @AssertFalse | 应用于boolean属性，该属性值必须为False | @AssertFalse boolean hasChildren; |
| @AssertTrue | 应用于boolean属性，该属性值必须为True | @AssertTrue boolean isEmpty; |
| @DecimalMax | 该属性值必须为小于或等于指定值的小数 | @DecimalMax(” 1 • 1 ’’） BigDecimal price; |
| @DecimalMin | 该属性值必须为大于或等于指定值的小数 | @DecimalMin("0.04n) BigDecimal price; |
| @Digits | 该属性值必须在指定范围内。integer属性定义该数值的最 大整数部分，fraction属性定义该数值的最大小数部分 | @Digits(integer=5, fraction=2) BigDecimal price; |
| @Future | 该属性值必须是未来的一个日期 | @Future  Date shippingDate; |
| @Max | 该属性值必须是一个小于或等于指定值的整数 | @Max(150) int age; |

续表

|  |  |  |
| --- | --- | --- |
| 属性 | 描述 | 范例 |
| @Min | 该属性值必须是一个大于或等于指定值的整数 | @Max(150) int age; |
| @NotNull | 该属性值不能为Null | @NotNull String firstName; |
| @Null | 该属性值必须为Null | @Null  String testString; |
| @Past | 该属性值必须是过去的一个日期 | @Past  Date birthDate; |
|  |  | @P attem(regext=,,\\d {3 |
| @Pattem | 该属性值必须与指定的常规表达式相匹配 | }")  String areaCode; |
| @Size | 该属性值必须在指定范围内 | Size(min=2, max=140) String description; |

一旦了解了 JSR 303 validation的使用方法，使用起来会比Spring验证器还要容易。像使 用Spring验证器一样，可以在属性文件中以下列格式来使用property键，覆盖来自JSR 303 验证器的错误消息：

*constraint. object .property*

例如，为了覆盖以@Size注解约束的Product对象的name属性，可以在属性文件中使用 下面这个键：

Size.product.name

为了覆盖以@Past注解约束的Product对象的productionDate属性，可以在属性文件中使 用下面这个键：

Past.product.productionDate

1. **JSR 303 Validator** 范例

jsr303-validator应用程序展示了 JSR 303输入验证的例子。这个应用程序是对spring- validator 进行修改之后 的版本 ，与之前的版本有一些区别 。首先 ，它没有 ProductValidator 类。 其次，来自Hibernate Validator库的jar文件已经被添加到了 WEB-INF/lib中。

清单7.6 Product类的name和productionDate字段已经用JSR 303注解类型进行了注解。

清单 7.6 Product 类

package domain;

import java,io.Serializable;

import j ava.math.BigDecimal

import j ava.time.LocalDate;

import javax.validation.constraints.Past;

import j avax.validation.constraints.Size;

public class Product implements Serializable {

private static final long serialVersionUID = 78L;

@Size(min=l, max=10) private String name;

private String description; private BigDecimal price;

@Past

private LocalDate productionDate;

// getters and setters not shown

}

在ProductController类的saveProduct方法中，必须用@\^1丨(1对Product参数进行注解，

如清单7.7所示。

清单 7.7 ProductController 类

package controller;

import j avax.validation.Valid;

import org.apache.commons•logging•Log;

import org.apache.commons.logging.LogFactory;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework•validation.BindingResuit;

import org.springframework.validation.FieldError;

import org.springframework,web.bind.annotation,ModelAttribute;

import org.springframework.web.bind.annotation.RequestMapping;

import domain.Product;

©Controller

public class ProductController {

private static final Log logger = LogFactory ■getLog(ProductController.class);

©RequestMapping(value = "/add-product") public String inputProduct(Model model) {

model • addAttribute (’’product1’，new Product ()); return nProductForm1';

©RequestMapping(value = "/save-product”）

public String saveProduct(@Valid @ModelAttribute Product product, BindingResult bindingResult, Model model) {

if (bindingResult.hasErrors{) ) {

FieldError fieldError = bindingResult.getFieldError(); logger. info {"Code: " + fieldError. getCode () + ", object:’’

+ fieldError.getObj ectName() + ", field:"

+ fieldError.getField{)); return "ProductForm";

II save product here

model.addAttribute("product", product); return "ProductDetails";

为了定制来自验证器的错误消息，要在messages.properties文件中使用两个键。 messages.properties 文件如清单 7.8 所示。

清单 7.8 jsr303-validator 中的 messages.properties 文件

productName.required.product.name=Please enter a product name

price.required=Please enter a price

price.negative=Price cannot be less than 0

productionDate.required=Please enter a production date

productionDate.invalid=Please ensure the production date is not later

分than today

typeMismatch.productionDate=Invalid production date Past•productionDate=Production date must be a past date

要想测试jsr303-Validat〇r中的验证器，可以在浏览器中打开以下网址：

<http://localhost:8080/j> sr303-validator/add-product

请注意，在编写本书时，最新版本（版本5.2.4)的Hibernate Validator仍然无法验证使用 @Past或@卩加1沈注解的LocalDate或LocalDateTime类型的字段。因为LocalDate是Date / Time

API中的新类型，应该用来代替java.util.Date。因此，这个项目包含一个自定义验证器和重 写过的 Hibernate 验证器的 ConstraintHelper 类。

**7.10**小结

本章学习了可以在Spring MVC应用程序中使用的两种验证器:Spring MVC验证器和JSR 303验证器。由于JSR 303是正式的Java规范，因此建议新项目使用JSR 303验证器。

第8章 表达式语言

JSP 2.0最重要的特性之一就是表达式语言（EL)，JSP用户可以用它来访问应用程序数据。 由于受到ECMAScript和XPath表达式语言的启发，EL也设计成可以轻松地编写免脚本的JSP 页面。也就是说，页面不使用任何JSP声明、表达式或者scriptlet。第11章会进一步介绍为 何无脚本的的JSP页面是一个最佳实践。

本章介绍如何使用EL表达式在JSP页面中显示数据和对象属性。它涵盖了最新的EL 3.0 版本技术。

本章中的所有示例都可以在本书附带的zip文件中的el-demo项目中找到。

8.1表达式语言简史

JSP2.0最初是将EL应用在JSP标准标签库（JSTL) 1.0规范中。JSP 1.2程序员将标准 库导入到他们的应用程序中，就可以使用EL。JSP 2.0及其更高版本的用户即使没有JSTL， 也能使用EL，但在许多应用程序中，还是需要JSTL的，因为它里面还包含了与EL无关的 其他标签。

JSP 2.1 和 JSP 2.2 中的 EL 要将 JSP 2.0 中的 EL 与 JSF (JavaServer Faces)中定义的 EL 统一起来。JSF是在Java中快速构建Web应用程序的框架，并且是构建在JSP 1.2之上。由 于JSP 1.2中缺乏整合式的表达式语言，并且JSP 2.0 EL也无法满足JSF的所有需求，因此为 JSF 1.0开发出了一款EL的变体。后来这两种语言变体合二为一。

**2013年5月发布了** EL 3.0版本（JSR 341), EL不再是JSP或任何其他技术的一部分， 而是一个独立的规范。EL 3.0添加了对lambda表达式的支持，并允许集合操作。其lambda 支持不需要Java SE 8，Java SE **7即可。**

8.2表达式语言的语法

EL表达式以$丨开头，并以丨结束。EL表达式的结构如下:

${expression}

#{expression}

例如，表达式x+y,可以写成：

${x+y}

或

#{x+y}

${哪}和# {exp}结构都由EL引擎以相同的方式进行计算。然而，当EL未被用作独立引 擎而是使用诸如JSF或JSP的底层技术时，该技术可以不同地解释构造。例如，在JSF中，${exp} 结构用于立即计算，#丨expr丨结构用于延迟计算（即表达式直到系统需要它的值时，才进行计 算)。另一方面，立即计算的表达式，会在JSP页面编译时同时编译，并在执行JSP页面时被执 行。在JSP2.1和更高版本中，#丨exp丨表达式只能在接受延迟表达式的标签属性中使用。

两个表达式可以连接在一起。对于一系列的表达式，它们的取值将是从左到右进行，计 算结果的类型为String，并且连接在一起。假如a+6等于8,等于10,那么这两个表达式 的计算结果将是810:

${a+b}${c+d}

表达式$ {a+b丨and$丨c+d丨的取值结果则是8andl 0。

如果在定制标签的属性值中使用EL表达式，那么该表达式的取值结果字符串将会强制 变成该属性需要的类型：

<my:tag someAttribute=M${expression}M/>

像${这样的字符顺序就表示是一个EL表达式的开头。如果需要的只是文本${，则需要在 它前面加一个转义符，如\${。

8.2.1关键字

以下是关键字，它们不能用作标识符：

and eq gt true instanceof

8.2表达式语言的语法

or ne le false empty

not It ge null div mod

1. []和■运算符

EL表达式可以返回任意类型的值。如果EL表达式的结果是一个带有属性的对象，则可以利 用□或者.运算符来访问该属性。口和.运算符类似；□是比较规范的形式，.运算符则比较快捷。

为了访问对象的属性，可以使用以下任意一种形式：

$ {ob j ect [npropertyName11 ] }

${obj ect.propertyName}

但是，如果propertyName不是有效的Java变量名，只能使用□运算符。例如，下面这两 个EL表达式就可以用来访问隐式对象标题中的HTTP标题host:

${header["host"]}

${header.host}

但是，要想访问accept-language标题，只能使用[]运算符，因为accept-language不是一 个合法的Java变量名。如果用.运算符访问它，将会导致异常。

如果对象的属性碰巧返回带有属性的另一个对象，既可以用[]，也可以用.运算符来 访问第二个对象的属性。例如，隐式对象pageContext是表示当前JSP的PageContext对象。 它有 request 属性，表示 HttpServletRequest。HttpServletRequest 带有 servletPath 属性。 那么，下列几个表达式的结果相同，均能得出pageContext中HttpServletRequest的 servletPath 属性值：

$ {pageContext [ "request"] [ "servletPath11 ] }

${pageContext.request["servletPath"]}

${pageContext.request.servletPath}

${pageContext["request"].servletPath}

要访问HttpSession，可以使用以下语法：

${pageContext.session}

例如，以下表达式会得出session标识符。

${pageContext.session.id}

8.2.3取值规则

EL表达式的取值是从左到右进行的。对于expr-a[expr-b]形式的表达式，其EL表达式的

取值方法如下：

1. 先计算 expr-a 得到 value-a。
2. 如果 value-a 为 null，则返回 null。
3. 然后计算expr-b得到value-b。
4. 如果 value-b 为 null，则返回 null。
5. 如果value-a为java.util.Map，则会查看value-b是否为Map中的一个key。若是，则 返回 value-a.get(value-b)，若不是，则返回 null。
6. 如果value-a为java.util.List，或者假如它是一个array,则要进行以下处理： a.强制value-b为int，如果强制失败，则拋出异常。

b•如果 value-a.get(value-b)拋出 IndexOutOffioundsException，或者假如 Array.get (value-a，value-b)拋出 ArraylndexOutOffio皿dsException，则返回 null。

c•否则，若 value-a 是个 List，则返回 value-a.get(value-b);若 value-a 是个 array， 贝!J返回 Array.get(value-a，value-b)。

1. 如果 value-a 不是一个 Map、List 或者 array,那么，value-a 必须是一个 JavaBean。 在这种情况下，必须强制value-b为String。如果value-b是value-a的一个可读属性，则要调 用该属性的getter方法，从中返回值。如果getter方法拋出异常，该表达式就是无效的，否则，

该表达式有效。

1. 访问 **JavaBean**

利用.或[]运算符，都可以访问bean的属性，其结构如下：

${beanName["propertyName"]}

${beanName.propertyName}

例如，访问myBean的secret属性，可以使用以下表达式：

${myBean•secret}

如果该属性是一个带属性的对象，那么同样也可以利用.或□运算符来访问第二个对象的 该属性。假如该属性是一个Map、List或者array，则可以利用8.2节介绍的访问Map值或List 成员或array元素的同样规则。

1. **EL**隐式对象

在JSP页面中，可以利用JSP脚本来访问JSP隐式对象。但是，在免脚本的JSP页面中， 则不可能访问这些隐式对象。EL允许通过提供一组它自己的隐式对象来访问不同的对象。EL 隐式对象见表8.1。

表8.1 EL隐式对象

|  |  |
| --- | --- |
| 对象 | 描述 |
| pageContext | 这是当前 JSP 的 javax.servletjsp.PageContext |
| initParam | 这是一个包含所有环境初始化参数并用参数名作为key的Map |
| param | 这是一个包含所有请求参数并用参数名作为key的Map。每个key的值就是指定名称的 第一个参数值。因此，如果两个请求参数同名，则只有第一个能够利用param获取值。 要想访问同名参数的所有参数值，可用params代替 |
| paramValues | 这是一个包含所有请求参数并用参数名作为key的Map。每卜key的值就是一个字符串 数组，其中包含了指定参数名称的所有参数值。就算该参数只有一个值，它也仍然会返 回一个带有一个元素的数组 |
| header | 这是一个包含请求标题并用标题名作为key的Map。每卜key的值就是指定标题名称的 第一个标题。换句话说，如果一个标题的值不止一个，则只返回第一个值。要想获得多 个值的标题，得用headerValues对象代替 |
| headerValues | 这是一个包含请求标题并用标题名作为key的Map。铺^ key的值就是一个字符串数组， 其中包含了指定标题名称的所有参数值。就算该标题只有一个值，它也仍然会返回一个 带有一个元素的数组 |
| cookie | 这是一个包含了当前请求对象中所有Cookie对象的Map。Cookie名称就是key名称， 并且每个key都映射到一个Cookie对象 |
| applicationScope | 这是一个包含了 ServletContext对象中所有属性的Map，并用属性名称作为key |
| sessionScope | 这是一个包含了 HttpSession对象中所有属性的Map,并用属性名称作为key |
| requestScope | 这是一个Map,其中包含了当前HttpServletRequest对象中的所有属性，并用属性名称 作为key |
| pageScope | 这是一个Map,其中包含了全页面范围内的所有属性。属性名称就是Map的key |

下面逐个介绍这些对象。

1. pageContext

pageContext对象表示当前JSP页面的javax.servletjsp.PageContext。它包含了所有其他的 JSP隐式对象，见表8.2。

表8.2 JSP隐式对象

|  |  |
| --- | --- |
| 对象 | EL中的类型 |
| request | javax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest) |
| response | javax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse) |
| out | javax.servlet.jsp.Jsp Writer |
| session | javax.servlet.[http.HttpSession](http://http.HttpSession) |
| application | javax.servlet.ServletContext |
| config | javax.servlet.ServletConfig |
| PageContext | javax.servletjsp.PageContext |
| page | javax.servletjsp.HttpJspPage |
| exception | java.lang.Throwable |

例如，可以利用以下任意一个表达式来获取当前的ServletRequest:

${pageContext.request}

${pageContext["request"]

并且，还可以利用以下任意一个表达式来获取请求方法：

${pageContext["request"]["method"]}

${pageContext["request"].method}

${pageContext.request["method"]}

${pageContext.request.method}

表8.3列出了 pageContext.request中一些有用的属性。

表8.3 pageContext.request中一些有用的属性

|  |  |
| --- | --- |
| 属性 | 说明 |
| characterEncoding | 请求的字符编码 |
| contentType | 请求的MIME类型 |
| locale | 浏览器首先locale |
| locales | 所有locale |
| protocol | HTTP 协议，例如：HTTP/1.1 |
| remoteAddr | 客户端IP地址 |
| remoteHost | 客户端IP地址或主机名 |
| scheme | 请求发送方案，HTTP或HTTPS |
| serverName | 服务器主机名 |
| serverPort | 服务器端口 |
| secure | 请求是否通过安全链接传输 |

对请求参数的访问比对其他隐式对象更加频繁；因此，这里提供了 param和paramValues 两个隐式对象。

1. initParam

隐式对象initParam用于获取上下文参数的值。例如，为了获取名为password的上下文参 数值，可以使用以下表达式：

${initParam.password}

或者

${initParam["password"]

1. param

隐式对象param用于获取请求参数值。这个对象表示一个包含所有请求参数的Map。例 如，要获取userName参数，可以使用以下任意一种表达式：

${param.userName}

${param["userName"]}

1. paramValues

利用隐式对象paramValues可以获取一个请求参数的多个值。这个对象表示一个包含所有 请求参数，并以参数名称作为key的Map。每个key的值是一个字符串数组，其中包含了指定 参数名称的所有值。即使该参数只有一个值，它也仍然返回一个带有一个元素的数组。例如， 为了获得selectedOptions参数的第一个值和第二个值，可以使用以下表达式：

${paramValues.selectedOptions[0]}

${paramValues.selectedOptions[1]}

1. header

隐式对象header表示一个包含所有请求标题的Map。为了获取header值，要利用header 名称作为key。例如，为了获取accept-language这个header值，可以使用以下表达式：

${header[naccept-languageM]}

如果header名称是一个有效的Java变量名，如connection,那么也可以使用.运算符： ${header.connection}

1. headerValues

隐式对象headerValues表示一个包含所有请求head并以header名称作为key的Map。但 是，与head不同的是，隐式对象headerValues返回的Map返回的是一个字符串数组。例如， 为了获取标题accept-language的第一个值，要使用以下表达式：

${headerValues[naccept-languageM][0]}

1. cookie

隐式对象cookie可以用来获取一个cookie。这个对象表示当前HttpServletRequest中所有 cookie的值。例如，为了获取名为jsessionid的cookie值，要使用以下表达式：

${cookie.j sessionid.value}

为了获取j sessionid cookie的路径值，要使用以下表达式：

${cookie.j sessionid.path}

8.48 applicationScope、sessionScope、requestScope 和网eScope

隐式对象applicationScope用于获取应用程序范围级变量的值。假如有一个应用程序范围 级变量myVar，就可以利用以下表达式来获取这个属性：

${applicationScope.myVar}

注意，在servlet/JSP编程中，有界对象是指在以下对象中作为属性的对象：PageContext、 ServletRequest、HttpSession 或者 ServletContext。隐式对象 sessionScope、requestScope 和 pageScope 与 applicationScope 相似。但是，其范围分别为 session、request 和 page。

有界对象也可以通过没有范围的EL表达式获取。在这种情况下，JSP容器将返回 PageContext、ServletRequest、HttpSession 或者 ServletContext 中第一个同名的对象。执行顺 序是从最小范围（PageContext)到最大范围（ServletContext)。例如，以下表达式将返回today

引用的任意范围的对象。

${today}

8.5使用其他EL运算符

除了.和[]运算符外，EL还提供了其他运算符：算术运算符、关系运算符、逻辑运算

符、条件运算符，以及empty运算符。使用这些运算符时，可以进行不同的运算。但是，由 于EL的目的是方便免脚本JSP页面的编程，因此，除了关系运算符外，这些EL运算符的用 处都很有限。

8.5.1算术运算符

算术运算符有5种。

•加法（+ )。

•减法（\_)。

•乘法（\*)。

•除法（/和div)。

•取余/取模（％和mod)。

除法和取余运算符都有两种形式，与XPath和ECMAScript是一致的。

注意，EL表达式的计算按优先级从高到低、从左到右进行。下列运算符是按优先级递减 顺序排列的：

• \*/div%mod

这表示\*、/、div、％以及mod运算符的优先级相同，+与-的优先级相同，但第二组运算 符的优先级小于第一组运算符。因此，表达式

${1+2\*3}

的运算结果是7,而不是9。

8.5.2关系运算符

下面是关系运算符列表：

•等于（==和eq)。

•不等于（!=和ne)。

•大于（>和gt)。

•大于或等于（>=和ge)。

•小于（<和11;)。

•小于或等于（<=和le)。

例如，表达式${3=4}返回False, ${“1)”<“<1”}则返回1'1^。

8.5.3逻辑运算符

下面是逻辑运算符列表：

•和（&&和 and)。

•或（|| 和 or)。

•非（！和 not)。

8.5.4条件运算符

EL条件运算符的语法如下：

***${statement? A:B}***

如果statement的计算结果为True，那么该表达式的输出结果就是A，否则为B。

例如，利用下列EL表达式可以测试HttpSession中是否包含名为loggedln的属性。如果 找到这个属性，就显示“You have logged in (您已经登录）”。否则显示“You have not logged in

(您尚未登录)”。

${(sessionScope.loggedIn==null)? "You have not logged **in":**

"You have logged in"}

1. empty 运算符

empty运算符用来检查某一个值是否为null或者empty。下面是一个empty运算符的使用 范例：

${empty X}

如果X为null，或者说X是一个长度为0的字符串，那么该表达式将返回True。如果X 是一个空Map、空数组或者空集合，它也将返回True。否则，将返回False。

8.5.6字符串连接运算符

+ =运算符用于连接字符串。例如，以下表达式打印a + b的值。

$ {a + = b}

8.5.7分号操作符

;操作符用于分隔两个表达式。有关示例，请参阅本章后面的8.10节。

8.6引用静态属性和静态方法

您可以引用在任何Java类中定义的静态字段和方法。但是，在可以在JSP页面中引用 静态字段或方法之前，必须使用page伪指令导入类或类包。jaVa.lang包是一个例外，因为 它是自动导入的。

例如，以下page指令导入java.time包。

<%@page import=Mjava.time.\*" %>

或者，你可以导入单个类，例如

<% ©page import="java.time.LocalDate” ％ >

限制可以引用LocalDate类的静态方法：now方法，如下：

Today is ${LocalDate.now()}

在本章后面的章节中，我们将学习如何格式化日期。

如下是引用类的静态成员和静态字段的另一个例子。

<p>

&radic; <span style = Ktext-decoration: overline;w >&nbsp; 36&nbsp; </ span> =$ {Math, sqrt ( 36 ) }

</p>

<p>

&pi; = ${Math.PI}

</p>

Math是java.lang包下类，无需额外的导入。图8.1展示了以上代码的JSP页面效果。

此外，还有一种导入包的方法，是在ServletContextlistener中 VI = 6.0

以编程方式导入。清单8.1显示了一个监听器，它导入两个包 71 = 3.141592653589793 java.time 和 java.util。 图8.1引用类静态成员

清单8.1编程导入类型

package listener;

import j avax.el.ELContextEvent;

import j avax.servlet.ServletContextEvent;

import j avax.servlet.ServletContextListener;

import j avax.servlet.annotation.WebListener;

import j avax.servlet.j sp.JspFactory;

@WebListener

public class ELImportListener implements ServletContextListener { 0〇verride

public void contextlnitialized(ServletContextEvent event) { JspFactory.getDefaultFactory{).getJspApplicationContext( event.getServletContext()).addELContextListener(

(ELContextEvent e) -> {

e.getELContext().getlmportHandler(). import Package (M j ava. time1'); e.getELContext<).getlmportHandler()• importPackage("java.util");

})；

SOverride

public void contextDestroyed(ServletContextEvent event) {



可以动态的创建Set、List和Map。创建一个Set的语法如下:

*comma-delimited-elements*

例如，如下表达式创建一个5个数字Set:

${{1, 2, 3, 4, 5}}

创建一个List的语法如下

*comma-delimited-elements*

例如，如下表达式创建了一组花名的List:

${ ["Aster”，’’Carnation1', ’’Rose”]}

最后，创建一个Map的语法为:

{ *comma -delimi ted-key-val ue-en tri es}*

8.9操作集合

如下为一组国家及其首都的Maps:

${{"Canada": "Ottawa", "China": "Beijing", "France": "Paris"}}

8.8访问列表元素和Map条目

可以通过索引来访问List,如下表达返回Cities的第一个元素

${cities [ 0] }

可以通过如下方式访问Map

*${map[key] }*

例如，下面的表达式返回“Ottawa”：

${{"Canada": "Ottawa", "China": "Beijing"} ["Canada"]}

8.9操作集合

EL3.0带来了很多新特性。其中一个主要的贡献是操纵集合的能力。你可以通过调用流 方法将集合转换为流来使用此功能。

下面展示如何将列表转换为流，假设myList是一个java.util.List:

${myList.stream()}

大部分流的操作会返回另一个流，因而可以形成链式操作。

$ {MyList.stream () .operation-1 () .operation-2 () .toList() }

在链式操作的末尾，通常调用toList方法，以便打印或格式化结果。

以下小节介绍了你可以对流执行的一些操作

1. toList

toList方法返回一个List，它包含与当前流相同的成员。调用此方法的主要目的是轻松地 打印或操作流元素。下面是一个将列表转换为流并返回列表的示例：

$ {[100, 200, 300]■stream()•toList ()}

当然这个例子没有什么用。稍后在接下来的小节中，你将看到更多的例子。

1. toArray

与toList类似，但返回一个Java数组。同样，在数组中呈现元素通常是有用的，因为许 多Java方法将数组作为参数。这里是一个toArray的例子：

$ {["One") "Two", "Three"].stream().toArray()}

与toList不同，toArray不打印元素。因此，toList更经常使用。

1. limit

limit方法限制流中元素的数量。

名为cities的List包含7个城市：

[Paris, Strasbourg, London, New York, Beijing, Amsterdam, San Francisco]

下面的代码将元素的数量限制为3。

$ {cities.stream{).limit(3).toList()}

执行时，表达式将返回此列表：

[Paris, Strasbourg, London]

如果传递给limit方法的参数大于元素的数量，则返回所有元素。

1. sort

此方法对流中的元素进行排序。例如，这个表达式

$ {cities.stream().sorted()•toList()}

返回如下排序后的列表。

[Amsterdam, Beijing, London, New York, Paris, San Francisco, Strasbourg]

1. average

此方法返回流中所有元素的平均值。其返回值是一个Optional对象，它可能为null。需 要调用get()获取实际值。

此表达式返回4.0:

$ {[1,3,5,7].stream().average().get()}

8.9操作集合

1. sum

此方法计算流中所有元素的总和。例如，此表达式返回16。

$ {[1,3,5,7]•stream().sum(>}

1. count

此方法返回流中元素的数量。例如，此表达式返回7。

$ {cities.stream(),count()}

1. min

此方法返回流的元素中的最小值。同average方法一样，其返回值是一个Optional对象， 因此你需要调用get方法来获取实际值。例如，此表达式返回1。

$ { [1,3,100, 1000] .streamO ,min{) ,get{) }

1. max

此方法返回流的元素中的最大值。同average方法一样，其返回值是一个Optional对象， 因此你需要调用get方法来获取实际值。例如，此表达式返回1000。

$ {[1,3,100, 1000] .stream() .max<)•get()}

1. map

此方法将流中的每个元素映射到另一个流中的另一个元素，并返回该流。此方法接受一 个lambda表达式。

例如，此映射方法使用lambda表达式x->2\*x，这实际上将每个元素乘2,并将它们返 回到新的流中。

$ {[1,3,5].stream().map(x - > 2 \* x).toList()}

返回列表如下：

[2,6,10]

下面是另一个示例，它将字符映射为大写。

$ {cities.stream() ,map(x - > x.toUpperCase ()) .toList ()}

它返回以下列表。

[PARIS, STRASBOURG, LONDON, NEW YORK, BEIJING, AMSTERDAM, SAN FRANCISCO]

1. filter

此方法根据lambda表达式过滤流中的所有元素，并返回包含结果的新流。

例如，以下表达式测试城市是否以“S”开头，并返回所有的结果。

$ {cities.stream() .filter(x - > x.startsWith(MS")) .toList () }

它产生的列表如下所示：

[Strasbourg, San Francisco]

1. forEach

此方法对流中的所有元素执行操作。它返回void。例如，此表达式将城市中的所有元素 打印到控制台。

$ {cities.stream().forEach(x - > System.out.println(x))}

8.10格式化集合

由于EL定义了如何写表达式而不是函数，因此无法直接打印或格式化集合，毕竟，打 印和格式化不是EL负责的领域。然而，打印和格式化是两个不能忽视的重要任务。

如果你是EL 3.0的新手，但熟悉JSP，那么格式化集合最简单的方法就是使用第6章中 讨论的JSTL。然而，具有强大的功能的EL 3.0应足以解决这些问题，并允许我们完全抛弃 JSTL〇

例如，你可以尝试使用forEach，类似于JSTL的forEach。以下代码可以在Tomcat 8上

运行。

$ {cities.stream().forEach(x - > pageContext.out.println(x))}

遗憾的是，这在GlassFish4中不起作用，所以forEach不能通用。

我最终想出的两个解决方案不像forEach那样优雅，但在所有主要的servlet容器上都可 用。第一个解决方案适用于Java SE 7。第二个解决方案可能比第一个更优雅，但只适用于Java SE8。

8.10.1使用HTML注释

List的字符串表示形式如下所示：

[element-1, element-2, ...]

现在，如果我想在HTML中呈现列表元素，需要这样写。

<ul>

<li> element-1 </ li>

<li> element-2 </ li>

</ul>

现在，你可能已经注意到每个元素必须转向<li> element-n </ li>。我怎么做？如果 你一直密切关注，你可能仍然记得map方法可以用来转换每个元素。所以，我会有这样的 内容：

$ {myList•stream().map(x - >”<li>”+ = x + ="</ li>”）•toList ()}

它给了我这样的List:

[<li> element-1 </ li>» <li> element-2 </ li>,".]

足够接近，但仍然需要删除括号和逗号。遗憾的是，你不能控制列表的字符串表示。好 在你可以使用HTML注释。

所以，这里是一个例子：

<lll>

<!-${cities.stream{).rnap(x - >"--> <li>"+ = x + ="</ li> <i--n).toList{)}--> </ul>

结果如下所示:

<ul>

<!--[--><li>Paris</li><!—, —><li>Strasbourg</li><!--,--> <li>London</li>< i--f --><li>New York</li><]--r -->

<li>Beijing</li><!--, --><li>Amsterdam</li><!--,--> <li>San Francisco</li><!--]-->

* Paris
* Strasbourg
* London
* New York
* Beijing
* Amsterdam

\_ San Francisco

图8.2使用HTML注释来  
格式化集合

</ul>

这有效地注释掉了括号和逗号。虽然结果看起来有点凌乱，但 它是有效的HTML,更重要的是，它能工作！

图8.2显示了页面显示结果。

这里有另一个例子，这个例子用表格格式化显示地址信息。

<table>

<tr><th>Street</thXth>City</th></tr>

<!--${addresses.stream〇.map(a->"-->

<trxtd>"+=a.streetName+="</tdxtd>n+=a.city+="</tdx/tr><!--”）.toList ()}

一一>

</table>

1. 使用 String.join()

这第二个解决方案之所以有效，因为EL 3.0允许你引用静态方法。在Java 8中，String 类新增了一些静态方法，其中一个方法是join，这正是我寻找的解决方案。有两个重载join 方法，但这里要用到的一个方法如下所示。

public static String join (CharSequence delimiter,

工terable <? extends CharSequence> elements )

此方法返回用指定分隔符连接在一起的CharSequence元素组成的字符串。而java.util. Collection接口正好扩展了 Iterable。因此，你可以将Collection传递给join方法。

例如，下面是如何将列表格式化为HTML有序列表：

$ {"<ol> <li>"+ = String, join ( "</ li> <li>",城市）+ ="< li> </ ol>"}

此表达式适用于至少有一个元素的集合。如果你可能要处理一个空集合，这里是一个更 好的表达式：

${empty cities? "" : "<ol><li>"

+= String. j oin (,,</lixli>", cities . stream () . sorted () . toList ())

+= "c/lix/ol〉"}

8.11格式化数字

要格式化数字，你可以再次利用EL 3.0允许引用静态方法的能力。String类的format静 态方法可以用来格式化数字。

例如，以下表达式返回带有两个小数点位的数字。

${String.format("%-10.2f%nM, 125.178) }

更多格式化规则可以查阅java.text.DecimalFormat的javadoc文档。

8.12格式化日期

可以通过String.format()来格式化一个date或time。例如：

$ {d = LocalDate.now() .plusDays(2); String.format<M%tB %te, %tY%nn, d, d, d) }

8.13 如何在JSP 2.0及其更高版本中配置EL

首先计算LocalDate.now().plusDays(2)，并将结果复制给变量d，然后用String.format()方 法来格式化LocalDate，引用了 3次变量d。

更多的格式化规则见：

<https://docs.oracle.com/javase/tutorial/java/data/numberformat.html>

8.13如何在JSP 2.0及其更高版本中配置EL

有了 EL、JavaBeans和定制标签，就可以编写免脚本的JSP页面了。JSP2.0及其更高的 版本中还提供了一个开关，可以使所有的JSP页面都禁用脚本。现在，软件架构师可以强制 编写免脚本的JSP页面了。

另一方面，在有些情况下，可能还会需要在应用程序中取消EL。例如，正在使用与JSP 2.0 兼容的容器，却尚未准备升级到JSP 2.0,那么就需要这么做。在这种情况下，可以关闭EL 表达式的计算。

8.13.1实现免脚本的JSP页面

为了关闭JSP页面中的脚本元素，要使用jsp-property-group元素以及url-pattem和 scripting-invalid两个子元素。url-pattem元素定义禁用脚本要应用的URL样式。下面展示如 何将一个应用程序中所有JSP页面的脚本都关闭：

<j sp-config>

<j sp-property-group>

<url-pattern>\*.j sp</url-pattern>

<scripting-invalid>true</scripting-invalid>

</j sp-property-group>

</j sp-config>

注意：

在部署描述符中只能有一个jsp-config元素。如果已经为禁用EL而定义了一个 jsp-property-group，就必须在同一个jsp-config元素下，为禁用脚本而编写jsp-property-group。

8.13.2禁用EL计算

在某些情况下，比如，当需要在JSP 2.0及其更高版本的容器中部署JSP 1.2应用程序时， 可能就需要禁用JSP页面中的EL计算了。此时，出现的EL结构，就不会作为EL表达式进

行计算。目前有两种方式可以禁用JSP中的EL计算。

第一种，可以将page指令的isELIgnored属性设为True，像这样：

<%@ page isELIgnored="trueM %>

isELIgnored属性的默认值为False。如果想在一个或者几个JSP页面中关闭EL表达式计 算，建议使用isELIgnored属性。

第二种，可以在部署描述符中使用jsp-property-group元素。jsp-property-group元素是jsp- config元素的子元素。利用jsp-property-group可以将某些设置应用到应用程序中的一组JSP

页面中。

为了利用jsp-property-group元素禁用EL计算，还必须有url-pattem和el-ignored两个子 元素。url-pattem元素用于定义EL禁用要应用的URL样式。el-ignored元素必须设为True。

下面举一个例子，展示如何在名为n〇EI.jsp的JSP页面中禁用EL计算。

<j sp-config>

<j sp-property-group>

<url-pattern>/noEl.j sp</url-pattern>

<el-ignored>true</el-ignored>

</j sp-property-group>

</j sp-config>

也可以像下面这样，通过给url-pattem元素赋值\*.jsp，来禁用一个应用程序中的所有JSP 页面的EL计算：

<j sp-config>

<j sp-property-group>

<url-pattern>\*.j sp</url-pattern>

<el-ignored>true</el-ignored>

</j sp-property-group>

</j sp-config>

无论是将其page指令的isELIgnored属性设为True，还是其URL与el-ignored为True的 jsp-property-group的URL模式相匹配，都将禁用JSP页面中的EL计算。假如将一个JSP页 面中page指令的isELIgnored属性设为False，URL与在部署描述符中禁用了 EL计算的 JSP页面的模式匹配，那么该页面的EL计算也将被禁用。

此外，如果使用的是与Servlet2.3及其更低版本兼容的部署描述符，那么EL计算已经默 认关闭，即便使用的是JSP2.0及其更高版本的容器，也一样。

**8.14**小结

EL是JSP 2.0及其更高版本中最重要的特性之一。它有助于编写更简短、更高效的JSP 页面，还能帮助编写免脚本的页面。本章介绍了如何利用EL来访问JavaBeans和隐式对象， 还介绍了如何使用EL运算符。本章的最后一个小节介绍了如何在与JSP 2.0及其更髙版本相 关的容器中使用与EL相关的应用程序设置。

第9章

JSTL

JSP标准标签库（JavaServer Pages Standard Tag Library, JSTL)是一个定制标签库的集合，

用来解历Map或集合、条件测试、XML处理，甚至麵库访问和数据操作等常见的问题。

本章要介绍的是JSTL中最重要的标签，尤其是访问有界对象、遍历集合，以及格式化数 字和日期的那些标签。如果有兴趣进一步了解，可以在JSTL规范文档中找到所有JSTL标签 的完整版说明。

注意

随着Expression Language 3.0的发布，所有JSTL核心标记都可以用EL表达式替代。然 而，旧项目仍然包含JSTL，因此掌握JSTL仍然会有帮助。

1. 下载 **JSTL**

JSTL目前的最新版本是1.2,这是由JSR-52专家组在JCP (www.jcp.org)上定义的，在 java.net网站可以下载：

<http://jstl.java.net>

其中，JSTL API和JSTL实现这两个软件是必需下载的。JSTL API中包含javax.servlet. jsp.jstl包，里面包含了 JSTL规范中定义的类型。JSTL实现中包含了实现类。这两个jar文件 都必须复制到应用JSTL的每个应用程序的WEB-INF/lib目录下。

1. **JSTL** 库

JSTL是标准标签库，但它是通过多个标签库来暴露其行为的。JSTL 1.2中的标签可以分

成5类区域，见表9.1。

表9.1 JSTL标签库

|  |  |  |  |
| --- | --- | --- | --- |
| 区域 | 子函数 | URI | 前缀 |
| 核心 | 变量支持 | http ://j ava. sun. com/j sp/j stl/core | c |
| 流控制 |
| URL管理 |
| 其他 |
| XML | 核心 | http ://j ava. sun.com/j sp/j stl/xml | X |
| 流控制 |
| 转换 |
| 国际化 | 语言区域 | http ://j ava. sunxom/j sp/j stl/fint | fint |
| 消息格式化 |
| 数字和日期格式化 |
| 数据库 | SQL | http ://j ava. sun. com/j sp/j stl/sql | sql |
| 函数 | 集合长度 | http ://j ava. sun. com/j sp/j stl/functions | fii |
| 字符串操作 |

在JSP页面中使用JSTL库，必须通过以下格式使用taglib指令:

<%@ taglib uri=nuriM prefix="prefix" %>

例如，要使用Core库，必须在JSP页面的开头处做以下声明：

<%@ taglib uri=n<http://java.sun.com/j> sp/j stl/core" prefix=ncM %>

这个前缀可以是任意的。但是，采用惯例能使团队的其他开发人员以及后续加入该项目 的其他人员更容易熟悉这些代码。因此，建议使用预定的前缀。

注意：

本章中讨论的每一个标签都会在各自独立的小节中做详细的介绍，每一个标签的属性也都 将列表说明。属性名称后面的星号（\* )表示该属性是必需的。加号（+ )表示该属性的rtexprvalue 值为True，这意味着该属性可以赋静态字符串或者动态值（Java表达式、EL表达式，或者通 ii<jsp:attribute>&置的值)。rtexprvalue值为False时，表示该属性只能赋静态字符串的值。

注意：

JSTL 标签的 body content 可以为 empty、JSP 或者 tagdependent。

1. —般行为

下面介绍Core库中用来操作有界变量的3个一般行为：out、set和remove。

1. out 标签

在运算表达式时，out标签是将结果输出到当前的JspWriter。out的语法有两种形式，即 有 body content 和I没有 body content。

<c:out value=Mvalue" [escapeXml=M{true|false}"]

*[default^defaultValue11]* />

<c:out value=Mvalue" [escapeXml=M{true|false}M]> default value </c:out>

注意：

在标签的语法中，[]表示可选的属性。如果值带下划线，则表示为默认值。 out的body content为JSP。out标签的属性见表9.2。

表9.2 out标签的属性

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 类型 | 描述 | |
| value \*+ | 对象 | 要计算的表达式 | |
| escapeXml+ | 布尔 | 表示结果中的字符<、 换成It;等等。 | >、&、'和”将被转换成相应的实体码，如<转 |
| default+ | 对象 | 默认值 | |

例如，下列的out标签将输出有界变量X的值:

<c:out value="${x}"/>

默认情况下，out会将特殊字符<、>、'、”和&分别编写成它们相应的字符实体码&lt;、 &gt;、&#039;、&#034;和&amp;。

在JSP 2.0版本前，out标签是用于输出有界对象值的最容易的方法。在JSP 2.0及其更高 的版本中，除非需要对某个值进行XML转义，否则可以放心地使用EL表达式：

${X}

警告：如果包含一个或多个特殊字符的字符串没有进行XML转义，它的值就无法在浏 览器中正常显示。此外，没有转义的特殊字符，会使网站易于遭受交叉网站的脚本攻击。例如，另!]人可以对它post —个能够自动执行的JavaScript函数/表达式。

out中的defkult属性可以赋一个默认值，当赋予其value属性的EL表达式返回null时，就会 显示默认值。default属性可以赋动态值,如果这个动态值返回null，out就会显示一个空的字符串。

例如，在下面的out标签中，如果在HttpSessioii中没有找到myVar变量，就会显示应用 程序范围的变量myVar值。如果没有找到，则输出一个空的字符串。

<c:out value="${sessionScope.myVar}"

def ault=n$ {applications cope .rayVar,T/>

1. url 标签

url标签是非常有用的简而言之，wl标签执行以下任一操作：

•如果当前上下文路径为“/”（即应用程序部署为默认上下文)，则它将空字符串附加 到指定的路径。

•如果当前上下文路径不是“/”，它会将上下文路径添加到指定的路径。

本节将通过以下一个小的应用程序来解释url标签的重要性，其结构在图9.1中给出。

该应用程序由两个JSP页面mainjsp和admin.jsp组成。 mainjsp文件位于应用程序根目录中，admin.jsp文件位于admin 文件夹中。二者都需要显示在图像文件夹中的两个图像， imagel.png和image2.png。请注意，图像的绝对路径如下。

色 **webapp** v **& admin** El **admin.jsp v & image**

imagel.png image2.png v & include li incl.jspf \_ inc2\_jspf 圍 inc3.jspf \_ inc4.jspf v ^WEB-INF 色 classes ▼為 lib

**s javax.servlet.jsp.jstl-1.2.1.jar** 趨 **javax.servlet.jsp.jstl-api-1.2.1.jar** 圍 mainjsp

图9.1 demo应用的目录结构

http: *//host/context/image/imagel.png*

http: /*/host/context/image/±mage2.png*

因为两个图像从不同位置被引用多次，为了方便使用，用 一个包含文件来引用它们。任何需要显示图像的JSP页面仅需 要将该包含文件添加到文件中即可。include文件夹表明了有4 个JSP页面这么做了。

清单9.1展示了第一个包含文件incl.jspf的内容。

清单9.1 incl.jsp的内容

incl.j sp

< img s rc="image/image1.png"/> <img src=" . . / image/image2 .png'1/

第一个包含文件包含路径是相对于当前页的路径。假设mainjsp页面的URL是http: //host/context/main.jsp，那么这两个图像的URL将被解析为以下URL:

<http://host/context/image/imagel.png> <http://host/context/../image/image2.png>

不难想象，结果不太令人满意。第一个图像能正确显示，但第二个图像不能。

当通过http: //host/context/admin/admin.jsp访问管理页面时，图像的URL解析如下：

<http://host/context/admin/image/imagel.png> <http://host/context/admin/../image/image2.png>

结果，第一个图像将不显示，但第二个图像将显示。

很明显，使用相对路径并不总能工作，因为可以从不同目录中的JSP页面调用包含文件。我们 唯一的希望是使图像URL相对于应用程序本身。所以就有了第二个包含文件:inC2.jspf(见清单9.2)。

清单9.2: inc2.jspf的内容

inc2,j sp

<img src=n/image/imagel.png"/>

<img src=n/irnage/image2 ,pngM/>

这看起来不错，应该能工作了，对吧？遗憾的是，它并不适用于所有情况。这是因为在 开发应用程序时，部署上下文路径通常是未知的。根据应用程序是否部署为默认上下文， admin.jsp页面可能具有以下URL之一：

<http://localhost/j> stl-demo/admin/admin.j sp <http://localhost/admin/admin.j> sp

在这两种情况下，浏览器不知道上下文路径。事实上，在第一个URL的情况下，它认为 应用程序被部署为默认上下文，jstl-demo是一个目录。因此，它将解析到第一个图像的URL 为 [http://localhost/image/imagel.png，这使得图像无法显示。](http://localhost/image/imagel.png%ef%bc%8c%e8%bf%99%e4%bd%bf%e5%be%97%e5%9b%be%e5%83%8f%e6%97%a0%e6%b3%95%e6%98%be%e7%a4%ba%e3%80%82)

仅当应用程序已部署到默认上下文时，才能正确显示这两个图像。然而，这足以说明 inc2.jspf不是正确的解决方案。

url标签是我们的救星。先看下使用了 url标签的inc3.jspf的内容，见清单9.3。

清单9.3: inc3.jspf的内容

< % @ taglib uri =Mhttp://java.sun,com/jsp/jstl/core"prefix **=McM%>** inc3,j sp

<img src =,l<c: url value =,l/image/imagel .png"/>M/>

<irag src ="<c: url value ="/image/image2.png1'/>’V>

这解决了问题，因为url标签在服务器上执行，它知道上下文路径是什么。所以，它可以 正确地解析图像的路径。

还可以使用EL表达式，如清单9.4中的inc4.jspf所示。

清单9.4: inc4.jspf的内容

inc4.j sp

<1 - $ {cp = pageContext.request.contextPath}-->

<img src ="${cp =="/"? " cp}/image/image1.png"/>

<img src ="${cp =="/"? " ”： cp}/image/image2.png"/>

可以使用以下表达式获取上下文路径：

${pageContext.request.contextPath}

相同的表达式将多次使用，所以我还创建了一个快捷方式〇P〇

${cp = pageContext.request.contextPath}

然而，该值仍然会发送到浏览器，这就是为什么它被放在一个HTML注释中。然后，你 只需要测试上下文路径是“/”（默认上下文）还是别的东西。

${cp ==”/，，？ "”： cp}

图9.2显示了非默认上下文中的adminjsp页面。

m@ Admin x

C D localhost:8080/jstl-demo/admin/admin.jsp 〇 E

|  |  |
| --- | --- |
| Admin |  |
| incl .jsp ^ | Sound Activated Clcck  20:15\_  Bt^fny Soitwdte |
| inc2.jsp : | m |
| inc3.jsp | $Qund Activity Clock  20:15\_  Brafrty Soitwar& |
| wmm  inc4.jsp IP ■.— | Sound Activated Clcck  20:15(3  Br^fny Soitwdte |

图9.2图像引用的四种不同方式

正如你可以看到的，只有inc3.jspf和inc4.jspf (分别采用url标签和EL)能够工作。

1. set 标签

利用set标签，可以完成以下工作：

1. 创建一个字符串和一个引用该字符串的有界变量。
2. 创建一个引用现存有界对象的有界变量。
3. 设置有界对象的属性。

如果用set创建有界变量，那么，在该标签出现后的整个JSP页面中都可以使用该变量。

set标签的语法有4种形式。第一种形式用于创建一个有界变量，并用value属性在其中 定义一个要创建的字符串或者现存有界对象。

<c: set value=” value” var=M varNazne"

[scope=n{page|request|session 丨 application}"]/>

这里的scope属性指定了有界变量的范围。

例如，下面的set标签创建了字符串“The wisest fool'并将它赋给新创建的页面范围变量foo:

<c:set var=Mfoon value="The wisest fool’’/>

下面的set标签则创建了一个名为job的有界变量，它引用requestScope对象position。 变量job的范围为page。

<c:set var=njobn value=M${requestScope.position}" scope=Mpagen/>

注意：

最后一个例子可能有点令人费解，因为它创建了一个引用请求范围对象的页面范围变量。 如果清楚有界对象本身并非真的在HttpServletRequest “里面”，就不难明白了。引用（名为 position)其实是指引用该对象。有了上一个例子中的set标签，再创建一个引用相同对象的 有界变量（job)即可。

第二种形式与第一种形式相似，只是要创建的字符串或者要引用的有界对象是作为body content赋值的。

<c:set **var=nvarName^** [scope=" {page | request | session | application}11 ] > **body content** </c:set>

第二种形式允许在body content中有JSP代码。

第三种形式是设置有界对象的属性值。target属性定义有界对象以及有界对象的property 属性。对该属性的赋值是通过value属性进行的。

<c: set target^"target11 property^"propertyName11 value=Mvalue"/>

例如，下面的set标签是将字符串“Tol^o”赋予有界对象address的city属性。

<c:set target="${address}M property=Mcity" value=MTokyo"/>

注意，必须在target属性中用一个EL表达式来引用这个有界对象。

第四种形式与第三种形式相似，只是赋值是作为body content完成的。

<c: set target^" target11 property^MpropertyNameM> body content </c:set>

例如，下面的set标签是将字符串“Beijing”赋予有界对象address的city属性。

<c: set target="$ {address} M property=llcity">Beij ing</c: set>

set标签的属性见表9.3。

表9.3 set标签的属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 描述 |
| value+ | 对象 | 要创建的字符串，或者要引用的有界对象，或者新的属性值 |
| var | 字符串 | 要创建的有界变量 |
| scope | 字符串 | 新创建的有界变量的范围 |
| target+ | 对象 | 其属性要被赋新值的有界对象；这必须是一个JavaBeans实例或者 java.util.Map 对象 |
| property+ | 字符串 | 要被赋新值的属性名称 |

1. remove 标签

remove标签用于删除有界变量，其语法如下：

<c: remove var=" varNa/ne"

[scope=n{page|request|session 丨 application}"]/>

注意，有界变量引用的对象不能删除。因此，如果另一个有界对象也引用了同一个对象， 仍然可以通过另一个有界变量访问该对象。

remove标签的属性见表9.4。

|  |  |  |
| --- | --- | --- |
| 表9.4 remove标签的属性 | | |
| 属性 | 类型 | 描述 |
| var | 字符串 | 要删除的有界变量的名称 |
| scope | 字符串 | 要删除的有界变量的范围 |

举个例子，下面的remove标签删除了页面范围的变量job。

<c: remove var=,,jobM scope=,,pagen/>

9.4条件行为

条件行为用于处理页面输出取决于特定输入值的情况，这在Java中是利用if、if…else 和switch声明解决的。

JSTL中执行条件行为的有4个标签，即if、choose、when和otherwise标签。下面分别

对其进行详细讲解。

1. if 标签

if标签是对某一个条件进行测试，假如结果为Tme，就处理它的body content。测试结果 保存在Boolean对象中，并创建有界变量来引用这个Boolean对象。利用var属性和scope属 性分别定义有界变量的名称和范围。

if的语法有两种形式。第一种形式没有body content。

<c: if test=n test Condi t ion” var=11 var Name11

[scope=n {page | request | session | application} M ] />

在这种情况下，var定义的有界对象一般是由其他标签在同一个JSP的后续阶段进行 测试。

第二种形式中使用了一个body content。

<c: if test=M testCondition [var="varName*11* ]

[scope=n{page|request I session|application}"]> body content </c:if>

body content是JSP，当测试条件的结果为True时，就会得到处理。if标签的属性见 表 9.5〇

表9.5 if标签的属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 描述 |
| test+ | 布尔 | 决定是否处理任何现有body content的测试条件 |
| var | 字符串 | 引用测试条件值的有界变量名称；var的类型为Boolean |
| scope | 字符串 | var定义的有界变量的范围 |

例如，如果找到请求参数user且值为ken，并且找到请求参数password且值为blackcomb， 以下if标签将显示“You logged in successfully (你己经成功登录)”：

<c:if test=n${param.user==1 ken1 && param.password==,blackcombl}">

You logged in successfully.

</c:if>

为了模拟else，下面使用了两个if标签，并使用了相反的条件。例如，如果user和password 参数的值为ken和blackcomb，以下代码片断将显示“You logged in successfully (你已经成功 登录)”。否则，将显示“Loginfailed (登录失败）”。

<c:if test=M${param.user==1 ken1 && param.password==,blackcombf}">

You logged in successfully.

</c:if>

<c:if test=M${! (param.user==1 ken1 && param.password==,blackcomb,)}M>

Login failed.

</c:if>

下面的if标签测试user和password参数值是否分别为ken和blackcomb，并将结果保存 在页面范围的变量loggedln中。之后，利用一个EL表达式，如果loggedln变量值为True， 则显示“You logged in successfully (你已经成功登录）”；如果loggedln变量值为False，则显 示 “Login failed (登录失败）”。

<c:if var=MloggedInM

test=M${param.user==1 ken1 && param.password==lblackcomb,}M/>

${(loggedln)? "You logged in successfully" : "Login failed"}

1. choose、when 和 otherwise 标签

choose和when标签的作用与Java中的关键字switch和case类似。也就是说，它们是用 于为相互排斥的条件执行提供上下文的。choose标签中必须嵌有一个或者多个when柄签，并 且每个when标签都表示一种可以计算和处理的情况。otherwise标签则用于默认的条件块， 假如没有任何一个when标签的测试条件结果为Tme，otherwise就会得到处理。假如是这种 情况，otherwise就必须放在最后一个when之后。

choose和otherwise标签没有属性。when标签必须带有定义测试条件的test属性，用来 决定是否应该处理body content。

举个例子，以下代码是测试参数status的值。如果status的值为fUll，示“You are a flill member (你是正式会员）”。如果这个值为student，则显示“You are a student member (你是 学生会员）”。如果status参数不存在，或者它的值既不是fUU，也不是student，那么这段代码 将不显示任何内容。

<c:choose>

<c:when test=M${param.status==1 full1}">

You are a full member </c:when>

<c:when test=M${param.status==1 student'}M>

You are a student member </c:when>

</c:choose>

下面的例子与前面的例子相似，但它是利用otherwise标签，如果status参数不存在，或 者它的值不是fiill或student，则将显示“Please register (请注册）”。

<c:choose>

<c:when test=M${param.status=='full'}">

You are a full member </c:when>

<c:when test=M${param.status=='student'}M>

You are a student member </c:when>

<c:otherwise〉

Please register </c:otherwise〉

</c:choose>

9.5遍历行为

当需要无数次地遍历一个对象集合时，遍历行为就很有帮助。JSTL提供了 forEach和 forTokens两个执行遍历行为的标签，这两个标签将在接下来的小节中介绍。

1. forEach 标签

forEach标签会无数次地反复遍历body content或者对象集合。可以遍历的对象包括

9.5遍历行为

java.util.Collection和java.util.Map的所有实现，以及对象数组或者基本类型。也可以遍历 java.util.Iterator 和 java.util.Enumeration,但不应该在多个行为中使用 Iterator 或者 Enumeration， 因为无法重置Iterator或者Enumeration。

forEach标签的语法有两种形式。第一种形式是固定次数地重复body content。

<c: forEach [var=n varNamen ] begin=MjbeginM end=Mend11 step=nstep"> body content </c:forEach>

第二种形式用于遍历对象集合。

<c: forEach items=M collection" [var=" varName11 ]

[varStatus=MvarStatusName'' ] [begin=MjbeginM ] [end=Mend1' ]

[step="stepM]> body content </c:forEach>

body content 是 JSP。forEach 标签的属性见表 9.6。

表9.6 forEach标签的属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 描述 |
| var | 字符串 | 引用遍历的当前项目的有界变量名称 |
| items+ | 支持的任意类型 | 遍历的对象集合 |
| varStatus | 字符串 | 保存遍历状态的有界变量名称。类型值为javax.servlet.jsp.jstl.core. LoopTagStatus |
| begin+ | 整数 | 如果指定items,遍历将从指定索引处的项开始，例如，集合中第一 项的索引为0。如果没有指定items,遍历将从设定的索引值开始。 如果指定，begin的值必须大于或者等于0 |
| end+ | 整数 | 如果指定items,遍历将在（含）指定索引处的项结束。如果没有指 定items，遍历将在索引到达指定值时结束 |
| step+ | 整数 | 遍历将只处理间隔指定step的项目，从第一项开始。在这种情况下， step的值必须大于或者等于1 |

例如，下列的forEach标签将显示“1，2, 3, 4, 5”。

<c: forEach var=nxn begin=Mln end=’’5”>

<c:out value=n${x}"/>,

</c:forEach>

下面的forEach标签将遍历有界变量address的phones属性:

<c:forEach var=nphone" items=M${address.phones}M>

${phone}M<br/>

</c:forEach>

对于每一次遍历，forEach标签都将创建一个有界变量，变量名称通过var属性定义。在 本例中，有界变量命名为phone。forEach标签中的EL表达式用于显示phone的值。这个有 界变量只存在于开始和结束的forEach标签之间，一到结束的forEach标签前，它就会被删除。

forEach 标签有一个类型为 javax.servletjsp.jstl.core.LoopTagStatus 的变量 varStatus。 LoopTagStatus接口带有count属性，它返回当前遍历的“次数”。第一次遍历时，status.count 值为1;第二次遍历时，Status.count值为2，依此类推。通过测试status.count%2的余数，可

以知道该标签正在处理的是偶数编号的元素，还是奇数编号的元素。

以iterator应用程序中的BookServlet类和BookListjsp页面为例。如清单9.5所示， BookServer类在其doGet方法中，创建了 3个Book对象，并放入到一个List对象中，并将该 List对象存入ServletRequest属性中，最后转到books.jsp页面，该页面会通过forEach标签遍 历图书集合。Book类如清单9.6所示。

清单 9.5 Books Servlet 类

package servlet; import java■io•I〇Exception; import java.util.ArrayList; import j ava.util.List;

import j avax.servlet.RequestDispatcher; import j avax.servlet.ServletException; import j avax.servlet.annotation.WebServlet; import j avax.servlet.[http.HttpServlet](http://http.HttpServlet); import j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest); import j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse); import appO5a.model.Book ;

@WebServlet (urlPatterns = {’’/books1’})

public class BooksServlet extends HttpServlet {

private static final int serialVersionUID = -234237;

@Override

public void doGet(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException {

List<Book> books = new ArrayList<Book>();

Book bookl = new Book(If978-0980839616M,

"Java 7: A Beginner 1s Tutorial”，

BigDecimal.valueOf(45.00));

Book book2 = new Book (11978-0980331608",

"Struts 2 Design and Programming: A Tutorial",

BigDecimal.valueOf(49.95));

Book book3 = new Book (11978-0975212820",

"Dimensional Data Warehousing with MySQL: A Tutorial", BigDecimal.valueOf(39.95)); books,add(bookl); books.add(book2); books.add(book3);

request.setAttribute("books", books);

RequestDispatcher rd =

request.getRequestDispatcher("/books.j sp"); rd.forward(request, response);

清单9.6 Book类

package domain;

import j ava.math.BigDecimal;

public class Book {

private String isbn; private String title; private BigDecimal price;

public Book(String isbn, String title, BigDecimal price) { this.isbn = isbn; this.title = title; this.price = price;

}

// getters and setters not shown

清单9.7 books.jsp页面

<%@ taglib uri=n<http://java.sun.com/j> sp/j stl/core" prefix=McM %> <!DOCTYPE html>

<html>

<head>

<title>Book List</title>

<style>table, tr, td {

border: lpx solid brown;

}

</style>

</head>

<body>

Books in Simple Table <table>

<tr>

<td>ISBN</td>

<td>Title</td>

</tr>

<c:forEach items="${requestScope.books}M var=Mbookn>

<tr>

<td>${book.isbn}</td>

<td>${book.title}</td>

</tr>

</c:forEach>

</table>

<br/>

Books in Styled Table <table>

<tr style="background:#ababffM>

<td>ISBN</td>

<td>Title</td>

</tr>

<c:forEach items="${requestScope.books}11 var=MbookM varStatus=MstatusM>

<c:if test=M${status.count%2 == 0}n>

<tr style=Mbackground:#eeeeff">

</c:if>

<c:if test=M${status.count%2 != 0}">

<tr style=Mbackground:#dedeff">

</c:if>

<td>${book.isbn}</td>

<td>${book.title}</td>

</tr>

</c:forEach>

</table>

<br/>

ISBNs only:

<c:forEach items="${requestScope.books}M var=Mbookn varStatus="status">

${book.isbn}<c:if test=M${!status.last}">,</c:if>

</c:forEach>

</body>

</html>

注意，books.jsp页面显示了图书3次，第一次是利用没有varStatus属性的forEach标签。

<table>

<tr>

<th>ISBN</th>

<th>Title</th>

</tr>

<c:forEach items="${requestScope books}" var=Mbookn>

<tr>

<td>${book.isbn}</td>

<td>${book.title}</td>

</tr>

</c:forEach>

</table>

第二次是利用有varStatus属性的forEach标签来显示，这是为了根据偶数行或奇数行来 给表格行设计不同的颜色。

<table>

<tr style="background:#ababffM>

<td>ISBN</td>

<td>Title</td>

</tr>

<c:forEach items=" ${ requestScope .books}11 var=MbookM varStatus="status">

<c:if test=M${status.count%2 == 0}">

<tr style=Mbackground:#eeeeff">

</c:if>

<c:if test=M${status.count%2 != 0}">

<tr style=Mbackground:#dedeff">

</c:if>

<td>${book.isbn}</td>

<td>${book.title}</td>

</tr>

</c:forEach>

</table>

最后一个forEach用来展示以逗号分隔的ISBN，status.last用来避免在文末有逗号。

<c:forEach items=M${requestScope.books}M var=nbook" varStatus=nstatusn>

${book.isbn}<c:if test="${!status.last}">,</c:if>

</c:forEach>

利用以下URL可以查看以上范例：

<http://localhost:8080/iterator/books>

其输出结果与图9.3所示的屏幕截图相似。

y@^ookUst

C L localhostSOSO/iterator/books ^1 CU =

Books in Simple Table

|  |  |
| --- | --- |
|  | [ISBN IjTutc |
|  | [978-0980839616||Java 7: A Beginner's Tutorial |
|  | |978-098033i606||Smjis 2 Design and Programming: A Tutorial |
|  | |97R-0975212B2〇||Dimensional Data Warehousing wirh MySQL： A Tutorial |

Books in Styled Table

|  |  |
| --- | --- |
|  | ：ISB1M ]FfiOe~ |
|  | ：978-0980839616i|Java 7: A Beginner's Tutorial |
|  | ：978-0980331606||5truts 2 Design and Programming: A Tutorial |
|  | f978^7521282〇][bimensianal Data Warehousing with MySQL: A Tuttwlalj |

ISBNs only: 97d-0980S39616.978-0980331608.978-0975212820

图 9.3 对 List 使用 forEach

利用forEach还可以遍历Map。要分别利用key和value属性引用一个Map key和一个 Map值。遍历Map的伪代码如下所示：

<c: forEach va.r=nmapltemn items=,,2napn>

${mapltem.key} : ${mapJtem.value}

</c:forEach>

清单9.8展示了 forEach与Map的结合使用。清单9.8中的BigCitiesServlet类将两个Map 实例化，并为它们赋予键/值对。第一个Map中的每一个元素都是一个String/String对，第二 个Map中的每一个元素则都是一个String/String[]对。

清单 9.8 BigCitiesServlet 类

package servlet; import java.io.IOException; import java.util.HashMap; import java.util.Map;

import javax.servlet.RequestDispatcher; import javax.servlet.ServletException; import j avax.servlet.annotation.WebServlet; import j avax.servlet.[http.HttpServlet](http://http.HttpServlet); import j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest); import j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse);

©WebServlet(urlPatterns = {'ybigCities"}〉 public class BigCitiesServlet extends HttpServlet { private static final long serialVersionUID = 1L;

SOverride public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

Map<String, String> capitals =

new HashMap<String, String>(); capitals .put ("Indonesia’1, "Jakarta"); capitals.put("Malaysia", "Kuala Lumpur"); capitals.put("Thailand", "Bangkok"); request.setAttribute("capitals", capitals);

Map<String, String[]> bigCities =

new HashMap<String, String[]>(); bigCities .put ("Australia", new String [ ] { "Sydney11,

"Melbourne”， "Perth"});

bigCities.put("New Zealand", new String[] {"Auckland",

"Christchurch", "Wellington"}); bigCities.put("Indonesia", new String[] {"Jakarta",

"Surabaya", "Medan"}); request.setAttribute("capitals", capitals); request.setAttribute("bigCities", bigCities);

RequestDispatcher rd =

request.getRequestDispatcher("/bigCities.j spM); rd.forward(request, response);

}

}

在doGet方法的结尾处，servlet跳转到bigcities.jsp页面，它利用forEach遍历Map。 bigcities.jsp页面如清单9.9所示。

清单 9.9 bigcities.jsp 页面

<%@ taglib uri=n<http://java.sun.com/j> sp/j stl/core" prefix=McM %>

<!DOCTYPE html>

<html>

<head>

<title>Big Cities</title>

<style>

table, tr, td {

border: lpx solid #aaee77; padding: 3px;

}

</style>

</head>

<body>

Capitals

<table>

<tr style="background:#448755;color:white;font-weight:bold">

<td>Country</td>

<td>Capital</td>

</tr>

<c:forEach items="${requestScope.capitals}M var=MmapItemM>

<tr>

<td>${mapltem.key}</td>

<td>${mapltem.value}</td>

</tr>

</c:forEach>

</table>

<br/>

Big Cities <table>

<tr style="background:#448755;color:white;font-weight:boldn> <td>Country</td>

<td>Cities</td>

</tr>

<c:forEach items="${requestScope.bigCities}" var="mapltem">

<tr>

<td>${mapltem.key}</td>

<td>

<c: forEach items=n${mapltem. value} ’’ var=Mcity,1 varStatus="status">

${city}<c:if test="${!status.last}">,</c:if>

</c:forEach>

</td>

</tr>

</c:forEach>

</table>

</body>

</html>

最重要的是，第二个forEach中还嵌套了另一个forEach:

<c: f orEach items=M$ {requestScope . bigCi ties}11 var="map I tem" >

<c:forEach items=M${mapltem.value}n var="cityM varStatus="statusM>

${city}<c:if test=M${!status.last}M>,</c:if>

</c:forEach>

</c:forEach>

这里的第二个forEach是遍历Map的元素值，它是一个String数组。

登录以下网站可以查看到以上范例：

<http://localhost:8080/iterator/bigCities>

打开网页后，浏览器上应该会以HTML表格的形式，显示出几个国家的首都和大城市,

Capitals

|  |  |
| --- | --- |
| Coutitry | CiipiiaJ |
| Malaysia | Kuala Lumpur |
| Thailand Bangkok | |
| Indonesia | Jakarta |

Big Cities

|  |  |
| --- | --- |
| ■ Country | Cities j |
| New Zealand | Auckland, Chrisichurch, Wellington |
| Australia | Sydney, Melbourne, Perth |
| Endonesia | Jakarta, Surabaya Medan |

**图** 9.4 **对** Map **使用** forEach

1. forTokens 标签

如图9.4所示。



C L 丨〇«丨host8080/iterator/bigCities ☆ C3 E

fbrTokens标签用于遍历以特定分隔符隔开的令牌，其语法如下：

<c:forTokens iterns=HstringOfTokens" delims="delimiters" [var="varNamen] [varStatus=nvarStatusWame"] [begin=,,Jbegi/2,' ] [end="er3dn] [step=n step"]

body content </c:forTokens>

body content 是 JSP。forTokens 标签的属性见表 9.7。

表9.7 forTokens标签的属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 描述 |
| var | 字符串 | 引用遍历的当前项目的有界变量名称 |
| items+ | 支持的任意类型 | 要遍历的token字符串 |
| varStatus | 字符串 | 保存遍历状态的有界变量名称。类型值为javax.servlet.jsp.jstl. core.LoopTagStatus |
| begin+ | 整数 | 遍历的起始索引，此处索引是从0开始的。如有指定，begin的 值必须大于或者等于0 |
| end+ | 整数 | 遍历的终止索引，此处索引是从0开始的 |
| step+ | 整数 | 遍历将以step指定的间隔的token来处理字符串，从第一个token 开始。如有指定，step的值必须大于或者等于1 |
| delims+ | 宇符串 | 一组分隔符 |

下面是一个forTokens范例：

<c: forTokens varyitem" items=ff Argentina, Brazil, Chile" delims=n, n>

<c:out value=M$ {item} M/xbr/>

</c:forTokens>

当将以上forTokens粘贴到JSP中时，它将会产生如下结果：

Argentina

Brazil

Chile

9.6格式化行为

JSTL提供了格式化和解析数字与日期的标签，它们是formatNumber、formatDate、 timeZone、setTimeZone、parseNumber 萍口 parseDate。

1. formatNumber 标签

formatNumber用于格式化数字。你可以根据需要，利用这个标签的各种属性来获得自己 想要的格式。formatNumber■的语法有两种形式。第一种形式没有body content:

<fmt:formatNumber *value=nnumerlcValue"*

[type="{number|currency|percent}"]

*[pattern=11 customPattern11 ]*

*[currencyCode=11 currencyCode11* ]

[c u r re n c yS ymb o 1=" c u rre/3 cySymbo J n ]

[groupingUsed=,1 {true | false}"]

[maxIntegerDigits="inaxIiitegerD:igi ts"]

[minInteger Digi t s=’’lii tegerDi g"i ts"]

*[ma.xFract±onDigits=t1 maxFractionDigi* tsM ]

*[minFractionDigits=^minFractionDigi*tsM]

[var="*varNamev* ]

[scope=n{page|request|session|application}M]

/>

第二种形式有body content:

<fmt:formatNumber [type=M{number|currency|percent}M]

*[pattern=11 customPattern11 ]*

[currencyCode=" curreiicyCode"]

[currencySymbol=M currencySymbol"]

[groupingUsed=,1 {true | false}"]

[maxIntegerDigits="\_maxIiiteger£?:lg'i ts"] [minIntegerDlg±ts=uminIntegerDlgits'' ] [maxFractionDigits=,,inaxFractionDigi tsn ] [minFractionDigits=,,jninFractionDigi tsn ]

[var=M varName11 ]

[scope=" {page | request | session | application}11 ] > numeric value to be formatted </fmt:formatNumber>

body content 是 JSP。formatNumber 标签的属性见表 9.8。

表9.8 formatNumber标签的属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 描述 |
| value+ | 字符串或数字 | 要格式化的数值 |
| type+ | 字符串 | 说明该值是要被格式化成数字、货币，还是百分比。这 个属性值如下：number、currency 和 percent |
| pattem+ | 字符串 | 定制格式化样式 |
| currencyCode+ | 字符串 | ISO 4217 码，见表 9.9 |
| currencySymbol+ | 字符串 | 货币符号 |
| groupingUsed+ | 布尔 | 说明输出结果中是否包含组分隔符 |
| maxIntegerDigits+ | 整数 | 规定输出结果的整数部分最多几位数字 |
| minIntegerDigits+ | 整数 | 规定输出结果的整数部分最少几位数字 |
| maxFractionDigits+ | 整数 | 规定输出结果的小数部分最多几位数字 |
| minFractionDigits+ | 整数 | 规定输出结果的小数部分最少几位数字 |
| var | 字符串 | 将输出结果存为字符串的有界变量名称 |
| scope | 字符串 | var的范围。如果有scope属性，则必须指定var属性 |

formatNumber标签的用途之一就是将数字格式化成货币。为此，可以利用currencyCode 属性来定义一个ISO 4217货币代码。部分ISO 4217货币代码见表9.9。

**表9.9部分**ISO 4217货币代码

|  |  |  |  |
| --- | --- | --- | --- |
| 市别 | ISO 4217 码 | 大单位名称 | 小单位名称 |
| 加拿大元 | CAD | 加元 | 分 |
| 人民币 | CNY | 元 | 角 |
| 欧元 | EUR | 欧元 | 分 |
| 日元 | JPY | 日元 | 钱 |
| 英镑 | GBP | 英镑 | 便士 |
| 美元 | USD | 美元 | 美分 |

FormatNumber标签的用法范例见表9.10。

表9.10 formatNumber标签的用法范例

|  |  |
| --- | --- |
| 行为 | 结果 |
| <fint: formatNumber value="12" type="numberM/> | 12 |
| <fint: formatNumber value="12" type="number" minIntegerDigits=n3 "/> | 012 |
| <fint: formatNumber value=" 12" type="numbern mmFractionDigits="2"/> | 12.00 |
| <fint:formatNumber value="123456.78" pattern^".000"/> | 123456.780 |
| <fint: formatNumber value=,,123456.78,' pattern-'#,#00.0#"/> | 123,456.78 |
| <fint: formatNumber value="12" type="currencyM/> | $12.00 |
| <fint: formatNumber value="12" type="currencyM currencyCode="GBP7> | GBP 12.00 |
| <fint:formatNumber value="0.12M type=Mpercent"/> | 12% |
| <fint:formatNumber value="0.125M type=Mpercent" mmFractionDigits=M2M/> | 12.50% |

注意，在格式化货币时，如果没有定义currencyCode属性，就无法使用浏览器的本地化。

1. formatDate 标签

formatDate标签用于格式化日期，其语法如下：

<fmt:formatDate value=MdateM

[type="{time|date|both}n]

[dateStyle=M{default|short|medium|long|full}M] [timeStyle=M{default|short|medium|long|full}M] [pattern^1 customPattern11 ]

[timeZone=n timeZone11 ]

[var=M varName'1 ]

[scope=n {page | request | session | application}11 ]

body content 为 JSP。formatDate 标签的属性见表 9.11。

表9.11 formatDate标签的属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 描述 |
| value+ | java.util.Date | 要格式化的日期和/或时间 |
| type+ | 字符串 | 说明要格式化的是时间、日期，还是时间与日期部分 都格式化 |

续表

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 描述 |
| dataStyle+ | 字符串 | 预定义日期的格式化样式，遵循java.text.DateFormat  中定义的语义 |
| timeStyle+ | 字符串 | 预定义时间的格式化样式，遵循java.text.DateFormat 中定义的语义 |
| pattem+ | 字符串 | 定制格式化样式 |
| timezone+ | 字符串或 java.util.TimeZone | 定义用于显示时间的时区 |
| var | 字符串 | 将输出结果存储为字符串的有界变量名称 |
| scope | 字符串 | var的范围 |

timeZone属性的可能值，请查看9.6.3小节。

下列代码利用formatDate标签格式化有界变量now引用的java.util.Date对象。

Default: <fmt:formatDate value=n${now}n/>

Short: <fmt:formatDate value="${now}" datestyle="short"/>

Medium: <fmt:formatDate value="${now}M dateStyle="medium"/>

Long: <fmt:formatDate value="${now}" dateStyle=Mlong"/>

Full: <fmt:formatDate value="${now}" datestyle=MfullM/>

下面的formatDate标签用于格式化时间：

Default: <fmt:formatDate type="time" value="${now}"/>

Short: <fmt:formatDate type="timeM value="${now}" timeStyle=MshortM/>

Medium: <fmt:formatDate type=ntime" value="${now}n timeStyle="medium"/>

Long: <fmt: formatDate type=,,timeM value=’’$ {now}’’ timeStyle=Mlongn/> Full: <fmt: formatDate type=,,timeM value=’’$ {now}’’ timestyle=MfullM/>

下面的formatDate标签用于格式化日期和时间：

Default: <fmt:formatDate type=nboth" value=n${now}n/>

Short date short time: <fmt:formatDate type=nbothM value="${now}n datestyle=nshort" timeStyle=MshortM/>

Long date long time format: <fmt:formatDate type="both"  
value="${now}n dateStyle=nlong" timeStyle=nlong"/>

下面的formatDate标签用于格式化带时区的时间：

Time zone CT: <fmt:formatDate type="timeM value="${now}" timeZone=MCTM/><br/>

Time zone HST: <fmt:formatDate type="time" value="${now}"

timeZone=nHST"/><br/>

下面的formatDate标签利用定制模式来格式化日期和时间：

<fmt:formatDate type="bothM value=n${now}" pattern=ndd.MM.yyM/>

<fmt:formatDate type="bothM value="${now}" pattern=Mdd.MM.yyyy"/>

1. timeZone 标签

timeZone标签用于定义时区，使其body content中的时间信息按指定时区进行格式化或

者解析。其语法如下：

<fmt: timeZone value=M timeZone''> body content </fmt:timeZone>

body content是JSP。属性值可以是类型为String或者java.util.TimeZone的动态值。美国 和加拿大时区的值见表9.12。

如果value属性为null或者empty，则使用GMT时区。

下面的范例用timeZone标签格式化带时区的日期。

<fmt:timeZone value=MGMT+l:00">

<fmt: formatDate value=M$ {now} M type=’’bothn dateStyle="full" timeStyle=,,fullM/> </fmt:timeZone>

<fmt:timeZone value=MHSTM>

<fmt: formatDate value=’’$ {now}’’ type=”bothn dateStyle=nfull" timestyle=MfullM/> </fmt:timeZone>

<fmt:timeZone value=MCSTM>

<fmt:formatDate value=M${now}M type=MbothM dateStyle="full" timeStyle=,,fullM/> </fmt:timeZone>

表9.12美国和加拿大时区的值

|  |  |  |
| --- | --- | --- |
| 缩写 | 全名 | 时区 |
| NST | 纽芬兰标准时间 | UTC-3:30 |
| NDT | 纽芬兰夏时制 | UTC-2:30 |
| AST | 大西洋标准时间 | UTC-4 |
| ADT | 大西洋夏时制 | UTC-3 |
| EST | 东部标准时间 | UTC-5 |

续表

|  |  |  |
| --- | --- | --- |
| 缩写 | 全名 | 时区 |
| EDT | 东部夏时制 | UTC-4 |
| ET | 东部时间，如EST或EDT | 氺 |
| CST | 中部标准时间 | UTC-6 |
| CDT | 中部夏时制 | UTC-5 |
| CT | 中部时间，如CST或CDT | 氺 |
| MST | 山地标准时间 | UTC-7 |
| MDT | 山地夏时制 | UTC-6 |
| MT | 山地时间，如MST或MDT | 氺 |
| PST | 太平洋标准时间 | UTC-8 |
| PDT | 太平洋夏时制 | UTC-7 |
| PT | 太平洋时间，如PST或PDT | \* |
| AKST | 阿拉斯加标准时间 | UTC-9 |
| AKDT | 阿拉斯加夏时制 | UTC-8 |
| HST | 夏威夷标准时间 | UTC-10 |

1. setTimeZone 标签

setTimeZone标签用于将指定时区保存在一个有界变量或者时间配置变量中。setTimeZone 的语法如下：

<fmt: setTimeZone value=n timeZone11 [var=M varName}1 ]

[scope=n{page|request|session|application}M]

/>

表9.13展示了 setTimeZone标签的属性。

表9.13 setTimeZone标签的属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 描述 |
| value+ | 字符串或 java.util.TimeZone | 时区 |
| var | 字符串 | 保存类型为java.util.TimeZone的时区的有界变量 |
| scope | 字符串 | var的范围或者时区配置变量 |

1. parseNumber 标签

parseNumber标签用于将以字符串表示的数字、货币或者百分比解析成数字。其语法有两

种形式。第一种形式没有body content:

<fmt:parseNumber value="JiumericValue"

[type="{number|currency|percent}M]

*[pattern=ncustomPattemn* ]

[parseLocale="parseLocaleM ]

[integerOnly=" {true | false} ’’]

[var=M varName11 ]

[scope=n{page|request|session 丨 application}"]

/>

第二种形式有body content:

<fmt :parseNumber [type=M {number | currency | percent}11 ]

*[pattern^1 customPattern*n]

[parseLocale="parseIi〇caleM ]

[integerOnly=" {true | false] ’’]

[var=M varName11 ]

[scope=n {page | request | session | application}11 ] > numeric value to be parsed </fmt:parseNumber>

body content 是 JSP。parseNumber 标签的属性见表 9.14。

下面的parseNumber标签就是解析有界变量quantity引用的值，并将结果保存在有界变量 formattedNumber 中

<fmt:parseNumber var="formattedNumberM type="number" value="$ {quantity} **’’/>**

表9.14 parseNumber标签的属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 描述 |
| value+ | 字符串或数字 | 要解析的字符串 |
| type+ | 字符串 | 说明该字符串是要解析成数字、货币，还是百分比 |
| pattem+ | 字符串 | 定制格式化样式，决定value属性中的字符串要如何 解析 |
| parseLocale+ | 字符串或者 java.util.Locale | 定义locale，在解析操作期间将其默认格式化样式或将 pattern属性定义的样式应用其中 |
| integerOnly+ | 布尔 | 说明是否只解析指定值的整数部分 |
| var | 字符串 | 保存输出结果的有界变量名称 |
| scope | 字符串 | var的范围 |

1. parseDate 标签

parseDate标签以区分地域的格式解析以字符串表示的日期和时间。其语法有两种形式。 第一种形式没有body content:

<fmt:parseDate value="date5tringM [type="{time|date|both}M]

[dateStyle=M{default|short|medium|long|full}M]

[timeStyle=M{default|short|medium|long|full}M]

*[pattern=n customPattern11* ]

[timeZone=n timeZone'' ]

[parseLocale="parseiocaleM ]

[var=M varName'1 ]

[scope=n {page | request | session | application}11 ]

/>

第二种形式有body content:

<fmt:parseDate [type="{time|date|both}M]

[dateStyle=”{default|short|medium|long|full}M]

[timeStyle=M{default|short|medium|long|full}M]

*[pattern^1 customPattern11* ]

[timeZone=M timeZoneu]

[parseLocale="parseIi〇caleM ]

[var=”varName"]

[scope=n {page | request | session | application}11 ] > date value to be parsed </fmt:parseDate>

body content 是 JSP。表 9.15 列出了 parseDate 标签的属性。

表9.15 parseDate标签的属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 描述 |
| value+ | 字符串 | 要解析的字符串 |
| type+ | 字符串 | 说明要解析的字符串中是否包含日期、时间或二者均有 |
| dateStyle+ | 字符串 | 日期的格式化样式 |
| timeStyle+ | 字符串 | 时间的格式化样式 |
| pattem+ | 字符串 | 定制格式化样式，决定要如何解析该字符串 |
| timeZone+ | 字符串或者 java.util.TimeZone | 定义时区，使日期字符串中的时间信息均根据它来解析 |
| parseLocale^ | 字符串或者 java.util.Locale | 定义locale,在解析操作期间用其默认格式化样式，或将 pattern属性定义的样式应用其中 |
| var | 字符串 | 保存输出结果的有界变量名称 |
| scope | 字符串 | var的范围 |

下面的parseDate标签用于解析有界变量myDate引用的日期，并将得到的java.util.Date

保存在一个页面范围的有界变量formattedDate中。

<c:set var=,,myDateM value=M 12/12/2005"/>

<fmt:parseDate var=MformattedDate" type=,,dateM dateStyle=MshortM value=M${myDate}n/>

**9.7**函数

除了定制行为外，JSTL 1.1和JSTL 1.2还定义了一套可以在EL表达式中使用的标准函 数。这些函数都集中放在function标签库中。要使用这些函数，必须在JSP的最前面使用以 下的taglib指令：

<%@ taglib -uri=nhttp: / /java. sun. com/j sp/ j stl/functionsM prefix=,,fnM %>

调用函数时，要以下列格式使用一个EL:

${fn:functionName}

这里的ftinctionName是函数名。

大部分函数都用于字符串操作。例如，length函数用于字符串和集合，并返回集合或者 数组中的项数，或者返回一个字符串的字符数。

1. contains 函数

contains函数用于测试一个字符串中是否包含指定的子字符串。如果字符串中包含该子字 符串，则返回值为True,否则，返回False。其语法如下：

contains*(string/ substring).*

例如，下面这两个EL表达式都将返回True:

<c:set var=,,myStringM value="Hello World’’/>

$ {fn: contains (myString, ’’Hello11) }

${ fn: contains ("Stella Cadente1', "Cadente") }

1. containsIgnoreCase 函数

containsIgnoreCase函数与contains函数相似，但测试是区分大小写的，其语法如下：

containsIgnoreCase(string, substring、

9.7 函数

例如，下列的EL表达式将返回True:

${fn:containsIgnoreCase{"Stella Cadente", "CADENTE")}

1. endsWith 函数

endsWith函数用于测试一个字符串是否以指定的后缀结尾。其返回值是一个Boolean，语 法如下：

endsWith*(string, suffix)*

例如，下列的EL表达式将返回True:

${fn:endsWith("Hello World”， "World")}

1. escapeXm丨函数

escapcXml函数用于给String编码。这种转换与out标签将其escapc；Xml属性设为TVuc 一样。escapeXml的语法如下：

escapeXml(string)

例如，下面的EL表达式：

${fn:escapeXml("Use <br/> to change lines")}

将被渲染成：

Use &lt;br/&gt; to change lines

1. indexOf 函数

indexOf函数返回指定子字符串在某个字符串中第一次出现时的索引。如果没有找到指定 的子字符串，则返回-1。其语法如下：

indexOf(string, substring)

例如，下面的EL表达式将返回7:

${fn:indexOf("Stella Cadente", MCadenteM)}

1. join 函数

join函数将一个String数组中的所有元素都合并成一个字符串，并用指定的分隔符分开， 其语法如下：

join(array, separator)

如果这个数组为null,就会返回一个空字符串。

例如，如果myArray是一个String数组，它带有两个元素“my”和“world”，那么，下 面的EL表达式：

${fn:j oin(myArray,M,M)}

将返回 “ my，world ”。

1. length 函数

length函数用于返回集合中的项数，或者字符串中的字符数，其语法如下：

length{input}

下面的EL表达式将返回14。

${fn:length (’’Stella Cadente", nCadenteM) }

1. replace 函数

replace函数将字符串中出现的所有befoieString都用afeiString替换，并返回结果，其语法如下:

replace*(string, beforeSubstring, afterSubstrlng)*

例如，下面的EL表达式：

${fn:replace ("Stella Cadente", **ne’’**，**nEM)** }

将返回 “StEllaCadEntE”。

1. split 函数

split函数用于将一个字符串分割成一个子字符串数组。它的作用与join相反。例如，下 列代码分割字符串“my，world”，并将结果保存在有界变量split中。随后，利用forEach标签 将split格式化成一个HTML表。

<c:set var=MsplitM value= \*${ fn: split < nmy, world’’，"，"）}\*/>

<table>

<c: forEach var=”substring’1 items=”$ {split}">

<tr><td>$ {substring}</tdx/tr>

</c:forEach>

</table>

结果为：

<table>

<tr><td>my</td></tr> <tr><td>world</td></tr> </table>

1. startsWith 函数

startsWith函数用于测试一个字符串是否以指定的前缀开头，其语法如下：

startsWith*(string, prefix)*

例如，下面的EL表达式将返回True:

$ {fn: startsWith ("Stella Cadente", "St1')}

1. substring 函数

substring函数用于返回一个从指定基于0的起始索引（含）到指定基于0的终止索引的 子字符串，其语法如下：

substring(string, beginlndex, endlndex)

下面的EL表达式将返回“Stel”。

$ {fn: substring ("Stella Cadente11, 0, 4) }

1. substringAfter 函数

substringAfter函数用于返回指定子字符串第一次出现后的字符串部分，其语法如下：

substringAfter*(string, substring)*

例如，下面的EL表达式：

$ {fn: substringAfter ("Stella Cadente", **ne’’）}**

将返回 “ 11a Cadente ”。

1. substringBefore 函数

substringBefore函数用于返回指定子字符串第一次出现前的字符串部分，其语法如下：

substringBefore*(strlng/ substring)*

例如，下面的EL表达式将返回“St”。

${fn:substringBefore("Stella Cadente", Me")}

1. toLowerCase 函数

toLowerCase函数将一个字符串转换成它的小写版本，其语法如下：

toLowerCase(string)

例如，下面的EL表达式将返回“ Stella cadente ”：

${fn:toLowerCase("Stella Cadente")}

1. toUpperCase 函数

toUpperCase函数将一个字符串转换成它的大写版本，其语法如下：

toUpperCase(string)

例如，下面的EL表达式将返回“STELLACADENTE”。

${fn:toUpperCase("Stella Cadente")}

1. trim 函数

trim函数用于删除一个字符串开头和结尾的空白，其语法如下：

trim *(string)*

例如，下面的EL表达式将返回“Stella Cadente”。

${fn:trim(" Stella Cadente " ) }

9J小结

JSTL可以完成一般的任务（如遍历、集合和条件)、处理XML文档、格式化文本、访问 数据库以及操作数据，等等。本章介绍了比较重要的一些标签，如操作有界对象的标签（out、 set、remove)，执行条件测试的标签（if、choose、when、otherwise)，遍历集合或token的标 签（forEach、forTokens)，解析和格式化日期与数字的标签（parseNumber、formatNumber、 parseDate、formatDate等），以及可以在EL表达式中使用的JSTL 1.2函数。

第10章 国际化

在这个全球化的时代，现在比过去更需要能够编写可以在讲不同语言的国家和地区部署 的应用程序。在这方面，需要了解两个术语。第一个术语是国际化，常常缩写为il8n，因为 其单词internationalization以i开头，以n结尾，在它们之间有18个字母。国际化是开发支持 多语言和数据格式的应用程序的技术，无需重写编程逻辑。

第二个术语是本地化，这是将国际化应用程序改成支持特定语言区域（locale)的技术。 语言区域是指一个特定的地理、政治或者文化区域。要考虑到语言区域的一个操作，就称作 区分语言区域的操作。例如，显示日期就是一个区分语言区域的操作，因为日期必须以用户 所在的国家或者地区使用的格式显示。2016年11月15日，在美国显示为11/15/2016,但在 澳大利亚则显示为15/11/2016。与国际化缩写为il8n—样，本地化缩写为110n。

Java谨记国际化的需求，为字符和字符串提供了 Unicode支持。因此，用Java编写国际 化的应用程序是一件很容易的事情。国际化应用程序的具体方式取决于有多少静态数据需要 以不同的语言显示出来。这里有两种方法：

1. 如果大量数据是静态的，就要针对每一个语言区域单独创建一个资源版本。这种 方法一般适用于带有大量静态HTML页面的Web应用程序。这个很简单，不在本章讨论 范围。
2. 如果需要国际化的静态数据量有限，就可以将文本元素，如元件标签和错误消息隔 离为文本文件。每个文本文件中都保存着一个语言区域的所有文本元素译文。随后，应用程 序会自动获取每一个元素。这样做的优势是显而易见的。每个文本元素无需重新编译应用程 序，便可轻松地进行编辑。这正是本章要讨论的技术。

本章将首先解释什么是语言区域，接着讲解国际化应用程序技术，最后介绍一个Spring MVC范例。

10.1语言区域

Java.util.Locale类表示一个语言区域。一个Locale对象包含3个主要元件：language、 country和variant。Language无疑是最重要的部分；但是，语言本身有时并不足以区分一个语 言区域。例如，讲英语的国家有很多，如美国和英国。但是，在美国讲的英语，与在英国用 的英语并非一模一样。因此，必须指定语言国家。再举一个例子，在中国大陆使用的汉语， 与在台湾地区用的汉语也是不完全一样的。

参数variant是一个特定于供应商或者特定于浏览器的代号。例如，用WIN表示Windows， 用MAC表示Macintosh,用P0SIX表示P0SIX。两个variant之间用一个下划线隔开，并将 最重要的部分放在最前面。例如，传统西班牙语，用language、country和variant参数构造一 个 locale 分别是 es，ES，Traditional\_WIN。

构造Locale对象时，要使用Locale类的其中一个构造器：

public Locale(java.lang.String language)

public Locale(java.lang.String language, java.lang.String country)

public Locale(java.lang.String language, j ava.lang.String country, j ava.lang.String variant)

语言代号是一个有效的ISO语言码。表10.1显示了 ISO 639语言码范例。

表10.1 ISO 639语言码范例

|  |  |
| --- | --- |
| 代码 | 语言 |
| de | 德语 |
| el | 希腊语 |
| en | 英语 |
| es | 西班牙语 |
| fr | 法语 |
| hi | 印地语 |
| it | 意大利语 |
| ja | 日语 |
| nl | 荷兰语 |
| pt | 葡萄牙语 |
| ru | 俄语 |
| zh | 汉语 |

参数country是一个有效的ISO国家码，由两个字母组成，ISO 3166 (<http://userpage>. chemie.fiiberlin.de/diverse/doc/ISO\_3166.html)中指定为大写字母。表 10.2 展示了 ISO 3166 国

家码范例。

表10.2 ISO 3166国家码范例

|  |  |
| --- | --- |
| 国家 | 代码 |
| 澳大利亚 | AU |
| 巴西 | BR |
| 加拿大 | CA |
| 中国 | CN |
| 埃及 | EG |
| 法国 | FR |
| 德国 | DE |
| 印度 | IN |
| 墨西哥 | MX |
| 瑞士 | CH |
| 英国 | GB |
| 美国 | US |

例如，要构造一个表示加拿大所用英语的Locale对象，可以像下面这样编写:

Locale locale = new Locale ("en", ’’CA");

此外，Locale类提供了 static final域，用来返回特定国家或语言的语言区域，如CANADA\_ FRENCH、CHINA、CHINESE、ENGLISH、FRANCE、FRENCH、UK、US 等。因此，也可 以通过调用其static域来构造Locale对象。

Locale locale = Locale.CANADA\_FRENCH;

此外，静态的getDefault方法会返回用户计算机的语言区域。

Locale locale = Locale.getDefault();

**10.2**国际化**Spring MVC**应用程序

国际化和本地化应用程序时，需要具备以下条件：

1. 将文本组件隔离成属性文件。
2. 选择和读取正确的属性文件。

下面详细介绍这两个步骤，并进行简单的示范。

10.2.1将文本组件隔离成属性文件

国际化的应用程序是将每一个语言区域的文本元素都单独保存在一个独立的属性文件 中。每个文件中都包含key/value对，并且每个key都唯一表示一个特定语言区域的对象。key 始终是字符串，value则可以是字符串，也可以是其他任意类型的对象。例如，为了支持美国 英语、德语以及汉语，就要有3个属性文件，它们都有着相同的key。

以下是英语版的属性文件。注意，它有greetings和farewell两个key。

greetings = Hello farewell = Goodbye

德国版的属性文件如下：

greetings = Hallo farewell = Ts chilli

汉语版的属性文件如下：

greetings=\u4f60\u597d farewell=\u518d\u89cl

如果你是中文用户，你可以使用任何中文文本编辑器并写入汉字字符。完成后，将文件 转换为Unicode。

现在，要学习java.util.ResourceBundle类。它能使你轻松地选择和读取特定于用户语言区 域的属性，以及查找值。ResourceBundle是一个抽象类，但它提供了静态的getBundle方法，

返回一个具体子类的实例。

ResourceBundle有一个基准名，它可以是任意名称。但是，为了让ResourceBundle正确 地选择属性文件，这个文件名中最好必须包含基准名ResourceBundle，后面再接下划线、语 言码，还可以选择再加一条下划线和国家码。属性文件名的格式如下所示：

basename\_languageCode\_countryCode

例如，假设基准名为MyResources，并且定义了以下3个语言区域：

• US-en；

* DE-de；
* CN-zh〇

那么，就会得到下面这3个属性文件：

* MyResources\_en\_U S .properties ；
* MyResources\_de\_DE.properties ；
* MyResources\_zh\_CN .properties 〇

10.2.2选择和读取正确的属性文件

如前所述，ResourceBundle是一个抽象类。尽管如此，还是可以通过调用它的静态 getBundle方法来获得一个ResourceBundle实例。它的重载签名如下：

public static ResourceBundle getBundle(java.lang.String baseName) public static ResourceBundle getBundle(java.lang.String baseName,

Locale *locale)*

例如：

ResourceBundle rb = ResourceBundle.getBundle("MyResources", Locale.US);

这样将会加载ResourceBundle在相应属性文件中的值。

如果没有找到合适的属性文件，ResourceBundle对象就会返回到默认的属性文件。默认 属性文件的名称为基准名加一个扩展名properties。在这个例子中，默认文件就是MyResources. properties。如果没有找到默认文件，将抛出 java.util.MissingResourceException。

随后，读取值，利用ResourceBundle类的getString方法传入一个key。

public java.lang.String getString(java.lang.String key)

如果没有找到指定key的入口，将会拋出java.util.MissingResourceException。

在 Spring MVC 中，不直接使用 ResourceBundle，而是利用 messageSource bean 告诉 Spring MVC要将属性文件保存在哪里。例如，下面的messageSource bean读取了两个属性

文件。

<bean id="messageSource" class=Morg.springframework.context.support. ReloadableResourceBundleMessageSourceM>

〈property name="basenames" >

<list>

<value>resource/messages</value> <value>resource/labels</value>

</list>

</property>

</bean>

上面的bean定义中用ReloadableResourceBundleMessageSource类作为实现。另一个实现 中包含了 ResourceBundleMessageSource，它是不能重新加载的。这意味着，如果在任意属性文件 中修改了某一个属性key或者value，并且正在使用ResourceBundleMessageSource，那么要使修 改生效，就必须先重启JVM。另一方面，也可以将ReloadableResourceBundleMessageSource设为

可重新加载。

这两个实现之间的另一个区别是：使用ReloadableResourceBundleMessageSource时，是 在应用程序目录下搜索这些属性文件。而使用ResourceBundleMessageSource时，属性文件则 必须放在类路径下，即WEB-INF/class目录下。

还要注意，如果只有一组属性文件，则可以用basename属性代替basenames，像下面这样：

<bean id="messageSource" class="org.springframework.context.support.

分 ResourceBundleMessageSource’'〉

〈property name=’’basename" value=,'re source/me s sage s"/>

</bean>

**10.3**告诉**Spring MVC**使用哪个语言区域

为用户选择语言区域时，最常用的方法或许是读取用户浏览器的accept-language标题值。 accept-language标题提供了关于用户偏好哪种语言的信息。

选择语言区域的其他方法还包括读取某个session属性或者cookie。

在Spring MVC中选择语言区域，可以使用语言区域解析器bean。它有几个实现， 包括：

* AcceptHeaderLocaleResolver ；
* SessionLocaleResolver；

• CookieLocaleResolver。

所有这些实现都是 org.springframework.web.servlet.il8n 包的组成部分。AcceptHeader- LocaleResolver或许是其中最容易使用的一个。如果选择使用这个语言区域解析器，Spring MVC将会读取浏览器的accept-language标题，来确定浏览器要接受哪个（些）语言区域。如

果浏览器的某个语言区域与Spring MVC应用程序支持的某个语言区域匹配，就会使用这个语 言区域。如果没有找到匹配的语言区域，则使用默认的语言区域。

下面是使用 AcceptHeaderLocaleResolver 的 localeResolverbean 定义：

<bean id=,,localeResolver" class=Morg. springf ramework. web . servlet. il8n . AcceptHeaderLocaleResolver"〉

</bean>

10.4使用message标签

在Spring MVC中显示本地化消息的最容易方法是使用Spring的message标签。为了使 用这个标签，要在使用该标签的所有JSP页面最前面声明如下的taglib指令。

<%@taglib prefix=MspringM

uri=nhttp: //www. springf ramework. org/tags,f%>

message标签的属性见表10.3。所有这些属性都是可选的。

表10.3 message标签的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| arguments | 该标签的参数写成一个有界的字符串、一个对象数组或者单个对象 |
| argumentSeparator | 用来分隔该标签参数的字符 |
| code | 获取消息的key |
| htmlEscape | 接受True或者False，表示被渲染文本是否应该进行HTML转义 |
| javaScriptEscape | 接受True或者False，表示被值染文本是否应该进行JavaScript转义 |
| message | MessageSourceResolvable 参数 |
| scope | 保存var属性中定义的变量的范围 |
| text | 如果code属性不存在，或者指定码无法获取消息，所显示的默认文本 |
| var | 用于保存消息的有界变量 |

10.5范例

举例来说，il8n应用程序展示了用localeResolver bean将JSP页面中的消息本地化的方 法。其目录结构如图10.1所示，il8n的Spring MVC配置文件如清单10.1所示。

色webapp T ^css

iMf main.css WEB-INF ► **^** classes ▼色 config

B springmvoconfig.xml ▼色 jsp

E ProductDetails.jsp **111** ProductForm.jsp ^ **to** lib T ^resource

II labels\_fr.properties B labels.properties 圍messages\_en.properties B messages\_\_fr.properties II messages.properties ® web.xml

图**10.1 il8n**的目录结构 清单**10.1 i18n**的**Spring MVC**配置文件

<?xml version=" 1.0" encoding=,,UTF-8"?>

<beans xmlns="<http://www.springframework.org/schema/beansM>xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xmlns:p=M<http://www.springframework.org/schema/pM>xmlns :mvc=,,http: //www. springf ramework. org/schema/mvc" xmlns : context=,,http: //www. springf ramework. org/schema/contextM xsi: schemaLocationy

<http://www.springframework.org/schema/beans>

<http://www.springframework.org/schema/beans/spring-beans.xsd> <http://www.springframework.org/schema/mvc> <http://www.springframework.org/schema/mvc/spring-mvc.xsd> <http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/springcontext.xsdM>>

<context: component-scan base-package=,,controllerM />

<mvc:annotation-driven/>

<mvc:resources mapping=M/css/\*\*" location="/css/M />

<mvc : resources mapping=n/\* .html" location^’’/" />

<bean id="viewRe solver" class = ,,org. springf ramework .web. servlet. view. InternalResourceViewResolver">

〈property name="prefix" value="/WEB-INF/jsp/" />

〈property name="suffix" value="•jsp" />

</bean>

<bean id=,'messageSource" class="org.springframework.context.support. ReloadableResourceBundleMessageSourceM>

〈property nameybasenames” >

<list>

<value>/WEB-INF/resource/messages</value>

<value>/WEB-INF/resource/labels</value>

</list>

</property>

</bean>

<bean id=”localeResolver” class=Morg.springframework.web.servlet.il8n. AcceptHeaderLocaleResolvern>

</bean>

</beans>

这里用到了 messageSource bean 和 localeResolver bean 这两个 bean。messageSource bean 声明 用两个基准名设置了 basenames 属性：/WEB-INF/resource/messages 和/WEB-INF/resource/labels。 localeResolver bean 利用 AcceptHeaderLocaleResolver 类实现消息的本地化。

它支持en和fr两个语言区域，因此每个属性文件都有两种版本。为了实现本地化，

JSP页面中的每一段文本都要用message标签代替。清单10.2展示了 ProductForm.jsp页 面。注意，为了达到调试的目的，当前的语言区域和accept-language标题显示在页面的 最前面。

清单 **10.2 ProductForm.jsp** 页面

<%@ taglib prefix=Mform"

uri=n[http://www.springframework.org/tags/formM%](http://www.springframework.org/tags/formM%25)>

<%@ taglib

prefix=Mspring" uri="[http://www.springframework.org/tags"%](http://www.springframework.org/tags%22%25)>

<%@ taglib uri="http: / /java. sun. com/j sp/j stl/core" pref **ix=,,c,,%>**

<!D0CTYPE HTML>

<html>

<head>

<title><spring: message code=”page • product form, title’’/></tit le>

<style type="text/Gssn>0import url(”<c:url  
value=’Vcss/main• css；</style>

</head>

<body>

<div id=,,global">

Current Locale : ${pageContext.response.locale}

<br/>

accept-language header: $ {header [’’accept-language1'] }

<form: form commandName=MproductM action=’’product\_save" me th o d= M p o s tM >

<fieldset>

<legend><spring:message code="form.name"/></legend>

<p>

<label for=llname"><spring:message

code="label.productNameM text=Mdefault text" />:

</label>

<f orm: input id=,,nameM path=’’namen cssErrorClass=,1 error11/>

<form:errors path=,Inamen cssClass=Merror"/>

</p>

<p>

<label for=11 description"xspring:message code="label.description M/>

</label>

<form:input id=Mdescription" path=MdescriptionM/>

</p>

<p>

<label for=llprice11 ><spring:message code=Mlabel .price\*1 text="default text" />: </label>

<f orm: input id=,’price" path="price’’ cssErrorClass=,1 error11/>

<form:errors path^^price11 cssClass="error’’/>

</p>

<p id="buttonsM>

<input id="reset" type=nreset" tabindex=M4n

value=M<spring:message code=,lbutton. resetn/>M>

<input id=11 submit11 type=Msubmit11 tabindex=,,5M

value=M<spring:message code="button•submitn/>">

</p>

</fieldset>

</form:form>

</div>

</body>

</html>

为了测试il8n的国际化特性，要修改浏览器的accept-language标签。

对于Chrome浏览器，打开“设置”页面，点击“显示高级设置”，点击“语言和输入设 置”，添加并移动语言到列表的顶部。

对于 IE 浏览器，到 Tools >Intemet Options > General (tab) > Languages > Language Preference中修改。在Language Preference窗口中，单击Add按钮添加一种语言。当选择了多 种语言时，为了修改某一种语言的优先值，要使用Move up和Move down按钮。

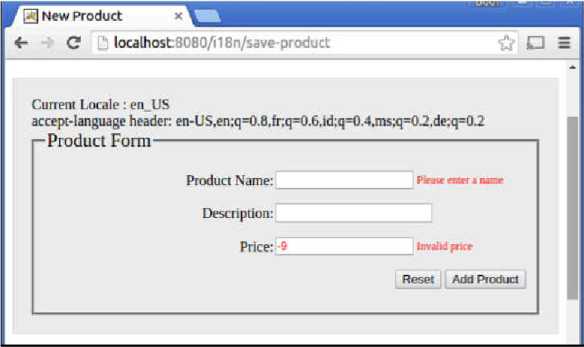
在其他浏览器中修改accept-language标题的说明，请访问以下网址查阅：

<http://www.w3.org/International/questions/qa-lang-priorities.en.php>

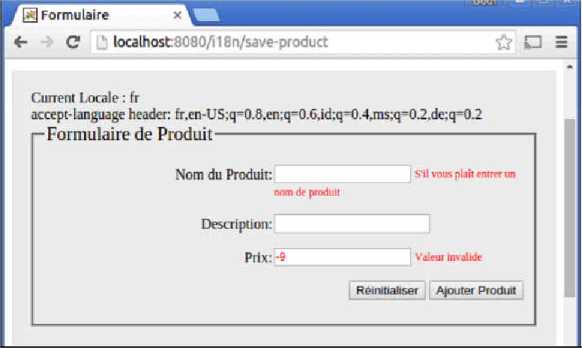
如果要对这个应用程序进行测试，访问以下URL:

<http://localhost:8080/il8n/add-product>

你将会看到Product表的英语版和法语版，分别如图10.2和图10.3所示。



**图10.2语言**g**域为**en**\_**US**的**Product**表**



**图10.3语言区域为**fr**\_\_**CA**的**Product**表**

10.6小结

本章讲解了如何开发国际化的应用首先介绍了 java»util.Locale类和java.util.Resource- Bundle类，然后展示了一个国际化应用程序的例子。

第11章

上传文件

Servlet技术出现之前，文件上传的编程仍然是一项很困难的任务，它涉及在服务器端解 析原始的HTTP响应。为了减轻编程的痛苦，开发人员借助于商业的文件上传组件。值得庆 幸的是，2003 年，Apache Software Foundation 发布了开源的 Commons FileUpload 元件，它很 快成为全球Servlet/JSP程序员的利器。

经过很多年，Servlet的设计人员才意识到文件上传的重要性，并终于成为Servlet 3 的内置特性。Servlet 3的开发人员不再需要将Commons FileUpload元件导入到他们的项 目中去。

为此，在Spring MVC中处理文件上传有两种方法：

1. 使用 Apache Commons FileUpload 组件。
2. 利用Servlet 3.0及其更高版本的内置支持。如果要将应用程序部署到支持Servlet 3.0 及其更高版本的容器中，则只能使用这种方法。

无论选择哪一种方法，都要利用相同的API来处理已经上传的文件。本章将介绍如何在

需要支持文件上传的Spring MVC应用程序中使用Commons FileUpload和Servlet 3文件上传

特性。此外，本章还将展示如何通过HTML 5增强用户体验。

11.1客户端编程

为了上传文件，必须将HTML表格的enctype属性值设为multipart/fomi-data，像下面这样：

**<form act±on="action"** enctype=,'multipart/form-data" **method=MpostM>**

Select a file <input type=,,file" name="**fleldNameu/>**

<input type=nsubmit" value="Upload"/>

</form>

表格中必须包含类型为file的一个input元素，它会显示成一个按钮，点击时，它会打开 一个对话框，用来选择文件。这个表格中也包含了其他的字段类型，如文本区域，或者隐藏字段。

在HTML 5之前，如果想要上传多个文件，必须使用多个文件input元素。但是，在HTML 5 中，通过在input元素中引入多个multiple属性，使得多个文件的上传变得更加简单。在HTML 5 中编写以下任意一行代码，便可生成一个按钮来选择多个文件：

<input type=Mfile" name= nfie 1 dName11 multiple/〉

<input type="file" name="fieldName" multiple=,,multiple"/>

<input type="file" name="fieldName" multiple="’’/>

1. **MultipartFile** 接口

在Spring MVC中处理已经上传的文件十分容易。上传到Spring MVC应用程序中的文件 会被包在一个MultipartFile对象中。你唯一的任务就是，用类型为MultipartFile的属性编写一 个 domain 类。

org.springframework.web.multipart.MultipartFile 接口具有以下方法：

byte[] getBytes{)

它以字节数组的形式返回文件的内容。

String getContentType()

它返回文件的内容类型。

InputStream getInputStream()

它返回一个InputStream，从中读取文件的内容。

String getName{)

它以多部分的形式返回参数的名称。

String getOriginalFilename ()

它返回客户端本地驱动器中的初始文件名。

long getSize ()

它以字节为单位，返回文件的大小。

boolean isEmpty()

它表示被上传的文件是否为空。

void transferTo(File destination)

它将上传的文件保存到目标目录下。

接下来的范例将讲解如何获取控制器中的已上传文件。

1. 用 **Commons FileUpload** 上传文件

只有实现了 Servlet 3.0及其更高版本规范的Servlet容器，才支持文件上传。对版本低于 Servlet 3.0的容器，则需要Apache Commons FileUpload组件，可以从以下网页下载它：

<http://commons.apache.org/proper/commons-fileupload/>

这是一个开源项目，因此是免费的，它还提供了源代码。为了让Commons FileUpload成 功地工作，还需要另一个Apache Commons组件：Apache Commons 10。从以下网页可以下载 到 Apache Commons 10:

http: //coimnons . apache . org/proper/commons-io/

因此，需要将两个JAR文件复制到应用程序的WEB-INF/lib目录下。Commons FileUpload JAR的名称遵循以下模式：

commons-fileupload-x.y.j ar

这里的x是指该软件的大版本，y是指小版本。例如，本章使用的名称是〇)111111〇115- fileupload-1.3.jar。

Commons IO JAR的名称遵循以下模式：

commons-io-x.y.j ar

这里的x是指该软件的大版本，y是指小版本。例如，本章使用的名称是commons- io-2Ajar〇

此外，还需要在Spring MVC配置文件中定义multipartResolver bean。

<bean id=,'multipartResolver"

class = ,,org. springframe work .web .multipart. commons .

CommonsMultipartResolver’’>

<property name=,,maxUploadSizeM value=,,2000000"/>

</bean>

范例uploadl展示了如何利用Apache Commons FileUpload处理已经上传的文件。

这个范例在Servlet 3.0容器中也是有效的。uploadl有一个domain类，即Product类， 它包含了一个MultipartFile对象列表。该例介绍了如何编写一个处理已上传产品图片的 控制器。

1. **Domain** 类

清单11.1展示了 domain类Product。它与前一个例子中的Product类相似，只是清单11.1 中的这个类还具有类型为List<MultipartFile>的 images属性。

清单11.1 经过修改的Product domain类

package domain; import java.io.Serializable; import j ava.math.BigDecimal; import java.util.List;

import j avax.validation.constraints.NotNull;

import j avax.validation.constraints.Size;

import org.springframework.web.multipart.MultipartFile;

public class Product implements Serializable {

private static final long serialVersionUID = 74458L;

@NotNull

@Size(min=l, max=10) private String name;

private String description; private BigDeciimal price; private List<MultipartFile> images;

// getters and setters not shown

**11.5**控制器

uploadl中的控制器如清单11.2所示。这个类中有inputProduct和saveProduct两个处理 请求的方法。inputProduct方法向浏览器发出一个产品表单。saveProduct方法将已上传的图片 文件保存到应用程序目录的image目录下。

清单 11.2 ProductController 类

package controller; import java.io.File; import j ava.io.I〇Excepti〇n; import j ava.util.ArrayList; import j ava.util.List;

import j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest);

import org. apache . coitimons . logging. Log;

import org.apache.commons.logging.LogFactory;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework•validation.BindingResuit;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.multipart.MultipartFile;

import domain.Product;

©Controller

public class ProductController {

private static final Log logger =

LogFactory.getLog(ProductController.class);

@RequestMapping(value = "/input-product") public String inputProduct(Model model) {

model.addAttribute("product", new Product()); return "ProductForm";

0RequestMapping(value = "/save-product")

public String saveProduct(HttpServletRequest servletRequest, @ModelAttribute Product product,

BindingResult bindingResult, Model model) {

List<MultipartFile> files = product.getlmages(); List<String> fileNames = new ArrayList<String>(); if (null != files && files.size() > 0) {

for (MultipartFile multipartFile : files) {

String fileName =

multipartFile.getOriginalFilename(); fileNames.add(fileName);

File imageFile = new

File(servletRequest.getServletContext() .getRealPath「/image’1) , fileName);

try {

multipartFile.transferTo(imageFile); } catch (IOException e) { e.printStackTrace();

// save product here

model.addAttribute("product" , product);

return "ProductDetails";

如清单11.2中的saveProduct方法所示，保存已上传文件是一件很轻松的事情，只需要在 MultipartFile 中调用 transferTo 方法。

11.6配置文件

清单11.3展示了 uploadl的SpringMVC配置文件。

清单11.3 uploadl的Spring MVC配置文件

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns=,,http: //www. springf ramework. org/schema/beans" xmlns :xsi=,,http: //www. w3 . org/2001/XMLSchema-instance,, xmlns:p="<http://www.springframework.org/schema/pM>xmlns :mvc=,,http: //www. springf ramework. org/schema/mvc" xmlns : context=,,http: //www. springf ramework. org/schema/contextM xsi: schemaLocationy

<http://www.springframework.org/schema/beans>

<http://www.springframework.org/schema/beans/spring-beans.xsd> <http://www.springframework.org/schema/mvc> <http://www.springframework.org/schema/mvc/spring-mvc.xsd> <http://www.springframework.org/schema/context> [http://www.springframework.org/schema/context/spring- context](http://www.springframework.org/schema/context/spring-context) .xsd">

<context: component-scan base-package=,,controller" />

<mvc:annotation-driven/〉

<mvc: resources mapping=,,/css/\*\*" location=M/css/M />

<mvc : resources mapping="/\* .html" location=,,/n />

<mvc:resources mappings"/image/\*\*M location^"/image/1' /> <bean id= MviewResolver"

class=Morg.springframework.web.servlet.view. InternalResourceViewResolver">

<property name=,,prefix" value=M/WEB-INF/jsp/M /> <property name=,,suf f ix" value=" • j sp" />

</bean> <bean id=MmultipartResolverM

class=,,org. springframework. web .multipart. commons .

Common sMu It ipartRe solve r’’>

</bean>

</beans>

利用multipartResolver bean的maxUploadSize属性，可以设置能够接受的最大文件容

量。如果没有设置这个属性，则没有最大文件容量限制。没有设置文件容量限制，并不意 味着可以上传任意大小的文件。上传过大的文件时要花很长的时间，这样会导致服务器超 时。为了处理超大文件的问题，可以利用HTML 5 File API将文件切片，然后再分别上传 这些文件。

**11.7 JSP** 页面

用于上传图片文件的ProductForm.jsp页面如清单11.4所示。

清单 11.4 ProductForm.jsp 页面

<%@ taglib prefix=Mform" uri="<http://www.springframework.org/tags/form>" %> <%@ taglib uri="http: / /java . sun. com/j sp/j stl/core" pref ix=,,cM %>

<!DOCTYPE HTML>

<html>

<head>

<title>Add Product Form</title>

<style type="text/css">@import url("<c:url value=’’/css/main• css"/>") ;</style>

</head>

<body>

<div id=,,global">

<form: form coirnnandName=,,productM action="save-product" method=T,post" enctype=,,multipart/form-data,,>

<fieldset>

<legend>Add a product</legend>

<p>

<label for="name">Product Name: </label>

<form: input id="name” path=Mname11 cssErrorClass=,1 error11/>

<form:errors path="namen cssClass=MerrorM/>

</p>

<p>

<label f or=11 description11 >De script ion: </label>

<f orm: input id=11 description" path=MdescriptionM/>

</p>

<p>

<label for="price">Price: </label>

<form:input id="price" path=MpriceM cssErrorClass="error"/>

</p>

<p>

<label for=MimageM>Product Image: </label>

<input type=Mfilen name="images[0]M/>

</p>

<p id="buttons">

<input id="reset" type=nreset" tabindex=M4n>

<input id=Msubmit11 type=nsubmit11 tabindex=,,5n value=MAdd Product">

</p>

</fieldset>

</form:form>

</div>

</body>

</html>

注意表单中类型为file的input元素，它将显示为一个按钮，用于选择要上传的文件。

提交Product表单，将会调用save-product方法。如果这个方法成功地完成，用户将会跳 转到清单11.5所示的ProductDetails.jsp页面。

清单 11.5 ProductDetails.jsp 页面

<%@ taglib uri=nhttp://java.sun,com/j sp/j stl/core" prefix=McM %>

<!D0CTYPE HTML>

<html>

<head>

<title>Save Product</title>

<style type=,,text/css">@import url ("<c:url value=’’/css/main• css"/>") ;</style>

</head>

<body>

<div id="globaln>

<h4>The product has been saved.</h4>

<P>

<h5>Details: </h.5>

Product Name: ${product.name}<br/>

Description: ${product.description}<br/>

Price: $${product.price}

<p>Following files are uploaded successfully.</p> <ol>

<c: forEach items=,,$ {product. images}" var=nimage"> <li>${image.originalFilename}

<img width=,,100n src=,,<c:url value=,'/image/"/> ${image.originalFilename}n/>

</li>

</c:forEach>

</ol>

</p>

</div>

</body>

</html>

ProductDetails.jsp页面显示出已保存的Product的详细信息及其图片。

11.8应用程序的测试

要测试这个应用程序，在浏览器中打开以下网址：

<http://localhost:8080/uploadl/input-product>

你将会看到一个如图11.1所示的“Add aProduct”表单。试着在其中输入一些产品信息， 并选择一个要上传的文件〇

f [3 Add Product Farm

♦ C localhost 3080/up!oad1/input-product



图11.1包含一个文件字段的产品表单

单击“AddProduct”按钮，就会看到如图11.2所示的网页。

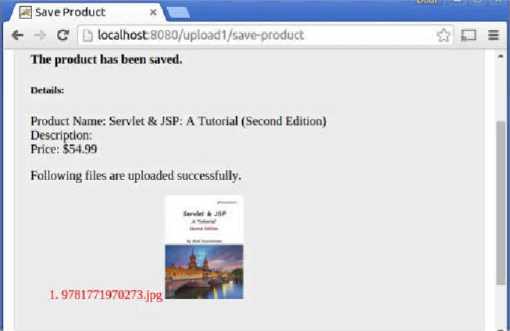


图11.2显示己经上传的图片

如果到应用程序目录的image目录下查看，就会看到己经上传的图片。

11.9用Servlet 3及其更高版本上传文件

有了 Servlet 3,就不需要 Commons FileUpload 和 Commons IO 元件了。在 Servlet 3 及 其以上版本的容器中进行服务器端文件上传的编程，是围绕着注解类型MiiltipartConfig和 javax.servlet.http.Part接口进行的。处理己上传文件的Servlets必须以@MultipartConfig进行

注解。

下列是可能在MultipartConfig注解类型中出现的属性，它们都是可选的。

* maxFikSize:上传文件的最大容量，默认值为-1，表示没有限制。大于指定值的文件 将会遭到拒绝。
* maxRequestSize:表示多部分HTTP请求允许的最大容量，默认值为-1,表示没有 限制。
* location:表示在Part调用write方法时，要将已上传的文件保存到磁盘中的位置。
* fileSizeThreshold:上传文件超出这个容量界限时，会被写入磁盘。

Spring MVC的DispatcherServlet处理大部分或者所有请求。令人遗憾的是，如果不修改 源代码，将无法对Servlet进行注解。但值得庆幸的是，Servlet 3中有一种比较容易的方法， 能使一个 Servlet 变成一个 MultipartConfig Servlet，即给部署描述符（web.xml)中的 Servlet 声明赋值。以下代码与用@MultipartConfig给DispatcherServlet进行注解的效果一样。

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet </servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>

/WEB-INF/config/springmvc-config.xml </param-value>

</init-param>

<multipart-config>

<max-file-size>20848820</max-file-size> <max-request-size>418018841</max-request-size>

<file-size-threshold>l048576</file-size-threshold>

</multipart-config>

</servlet>

此外，还需要在Spring MVC配件文件中使用一个不同的多部分解析器，像下面这样：

<bean id=,,multipartResolver,'

class="org.springframework.web.multipart.support. StandardServletMultipartResolver">

</bean>

upload2应用程序展示了如何在Servlet 3及其更高版本的容器中处理文件上传问题。这是 从uploadl改写过来的，因此，domain和controller类都非常相似。唯一的区别在于，现 在的web.xml文件中包含了一个multipart-config元素。清单11.6展示了 upload2的web.xml

文件。

清单11.6 upload2的web.xm丨文件

<?xml version=" 1.0" encoding=,,UTF-8" ?>

<web-app version=M3.0"

xmlns=,,http: / /java. sun. com/xml/ns/javaee" xmlns :xsi=,,http: //www. w3 . org/2001/XMLSchema-instance,, xsi : schemaLocation=,,http: //java. sun. com/xml/ns/javaee http: //java.sun. com/xml/ns/javaee/web-app\_3\_0 .xsdfT>

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet </servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>

/WEB-INF/config/springmvc-config.xml </param-value>

</init-param>

<load-on-startup>l</load-on-startup>

〈multipart-config>

<max-file-size>20848820</max-file-size> <max-request-size>418018841</max-request-size>

<file-size-threshold>104 857 6</file-size-threshold> </multipart-config>

</servlet> <servlet-mapping>

<servlet-name>springmvc</servlet-name> <ur1-pattern>/</url-pattern〉 </servlet-mapping>

</web-app>

清单11.7展示了 upload2的SpringMVC配置文件。

清单11.7 upload2的Spring MVC配置文件

<?xml version=" 1.0M encoding=,,UTF-8,' ?>

<beans xmlns="<http://www.springframework.org/sGhema/beansM>xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xmlns:p="<http://www.springframework.org/schema/pM>xmlns :mvc=,,http: //www. springf ramework. org/schema/mvc" xmlns : context=,,http: //www. springframework.org/schema/context" xsi: schemaLocationy

<http://www.springframework.org/schema/beans>

<http://www.springframework.org/schema/beans/spring-beans.xsd>

<http://www.springframework.org/schema/mvc>

<http://www.springframework.org/schema/mvc/spring-mvc.xsd>

<http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/spring->

分 context.xsd"> <context:component-scan base-package=,’controller" /> <mvc:annotation-driven /> <mvc:resources mapping="/css/\*\*" location="/css/" /> <mvc : resources mapping="/\* . html" location=,,/n />

<mvc:resources mapping="/image/\* \*" location=n/image/" /> <mvc:resources mapping="/file/\*\*" location=M/file/" /> <bean id=MviewResolverM

class=norg.springframework.web.servlet.view.

InternalResourceViewResolver">

<property name=,,prefix" value=M/WEB-INF/jsp/M /> <property name=,,s'uf f ix" value=" . j sp" />

</bean> <bean id="multipartResolver"

class="org.springframework.web.multipart.support. 分 StandardservletMultipartResolver">

</bean>

</beans>

如果要对这个应用程序进行测试，请在浏览器中访问以下URL:

<http://localhost:8080/upload2/input-product>

11.10客户端上传

虽然Servlet 3中的文件上传特性使文件上传变得十分容易，只需在服务器端编程即可， 但这对提升用户体验却毫无帮助。单独一个HTML表单并不能显示进度条，或者显示已经成 功上传的文件数量。开发人员采用了各种不同的技术来改善用户界面，例如，单独用一个浏 览器线程对服务器发出请求，以便报告上传进度，或者利用像Java applets、Adobe Flash、 Microsoft Silverlight这样的第三方技术。

这些第三方技术可以工作，但都在一定程度上存在限制。今天Java applets和Silverlight 几乎死了，Chrome 不再允许 applet 和 Silverlight, Microsoft 取代 Internet Explorer 的新浏览器 Edge根本就不支持插件。

你仍然可以使用Flash,因为Chrome仍然可以运行它，Edge已经集成了它。然而，现在 越来越多的人选择拥抱HTML5。

HTML 5在其DOM中添加了一个File API。它允许访问本地文件。与Java小程序、Adobe Flash、Microsoft Silverlight相比，HTML 5似乎是针对客户端文件上传局限性的最佳解决方案。 令人遗憾的是，在编写本书时，IE 9尚未完全支持这个API，但可以利用最新版的Firefox、 Chrome和Opera浏览器来测试下面的例子。

为了证明 HTML 5 的威力，upload2 中的 html5.jsp 页面采用了 JavaScript 和 HTML 5 File API来提供报告上传进度的进度条。upload2应用程序中也复制了一份MultipleUploadsServlet 类，用于在服务器中保存已上传的文件。但是，JavaScript不在本书讨论范围之内，因此这里 只做简单的说明。

简言之，我们关注的是HTML 5 input元素的change事牛，当input元素的值发生改变时， 就会触发它。本书还关注HTML 5在XMLHttpRequest对象中添加的progress事件。 XMLHttpRequest自然是AJAX的骨架。当异步使用XMLHttpRequest对象上传文件时，就会 持续地触发progress事件，直到上传进度完成或取消，或者直到上传进度因为出错而中断。 通过监听progress事件，可以轻松地监测文件上传操作的进度。

upload2中的Html5FileUploadController类能够将已经上传的文件保存到应用程序目录 的file目录下。清单11.8中的UploadedFile类展示了一个简单的domain类，它只包含一个 属性。

清单 11.8 UploadedFile 的 domain 类

package domain;

import java.io.Serializable;

import org.springframework.web.multipart.MultipartFile;

public class UploadedFile implements Serializable { private static final long serialVersionUID = 1L; private MultipartFile multipartFile; public MultipartFile getMultipartFile() { return multipartFile;

}

public void setMultipartFile(MultipartFile multipartFile) { this.multipartFile = multipartFile;

Html5FileUploadController 类如清单 11.9 所示。 清单 11.9 Html5FileUploadController 类

package controller; import j ava•io.File; import j ava■io•工〇Exception;

import j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest); import org.apache,commons.logging.Log; import org.apache.commons.logging.LogFactory; import org.springframework.stereotype.Controller; import org.springframework.ui.Model;

import org.springframework•validation.BindingResuit; import org.springframework.web.bind.annotation.ModelAttribute; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.multipart.MultipartFile;

import domain.UploadedFile;

©Controller

public class Html5FileUploadController {

private static final Log logger = LogFactory

.getLog(Html5FileUploadController.class);

@RequestMapping(value = "/html5n) public String inputProduct() {

return nHtml5";

0RequestMapping(value = "/file\_upload")

public void saveFile(HttpServletRequest servletRequest, 0ModelAttribute UploadedFile uploadedFile,

BindingResult bindingResult, Model model) {

MultipartFile multipartFile =

uploadedFile.getMultipartFile();

String fileName = multipartFile.getOriginalFilename(); try {

File file = new File(servletRequest.getServletContext() .getRealPath("/file"), fileName); multipartFile.transferTo(file);

} catch (IOException e) { e.printStackTrace{);

Html5FileUploadController中的saveFile方法将已经上传的文件保存到应用程序目录中的 file目录下。

清单11.10所示的html5.jsp页面中包含的JavaScript代码允许用户选择多个文件，并且一

键单击即可全部上传。这些文件本身将同时上传。

清单 11.10 html5.jsp 页面

<!D0CTYPE HTML>

<html>

<head>

<script>

var totalFileLength, totalUploaded, fileCount, filesUploaded;

function debug(s) {

var debug = document.getElementByld{1 debug1); if (debug) {

debug.innerHTML = debug.innerHTML + 1<br/>1 + s;

}

}

function onUploadComplete(e) {

totalUploaded += document.getElementByld(1 files 1).

files[filesUploaded].size; filesUploaded++;

debug(1 complete ' + filesUploaded + M of " + fileCount); debug(1totalUploaded: 1 + totalUploaded); if (filesUploaded < fileCount) { uploadNext();

} else {

var bar = document.getElementById(lbar,);

bar.style.width = 1100%1;

bar.innerHTML = 1100% complete 1;

alert(1 Finished uploading file(s) 1)；

}

}

function onFileSelect(e) {

var files = e.target.files; // FileList object var output **=[];** fileCount = files.length; totalFileLength = 0; for (var i=0; i<fileCount; i++) {

var file = files[i]; output.push{file.name, 1 (1,

file.size, 1 bytes, 1,

file.lastModifiedDate.toLocaleDateString(), 1)

)；

output.push(1<br/>1); debug(1 add 1 + file,size); totalFileLength += file.size;

}

document.getElementByld(1selectedFiles1).innerHTML = output,j oin{11)；

debug(1totalFileLength:1 + totalFileLength);

function onUploadProgress(e) {

if (e.lengthComputable) {

var percentComplete = parselnt(

(e.loaded + totalUploaded) \* 100

/ totalFileLength);

var bar = document.getElementByld(1bar1); bar.style.width = percentComplete + 1 % 1; bar.innerHTML = percentComplete + 1 % complete1; } else {

debug(1 unable to compute1);

function onUploadFailed(e) {

alert("Error uploading file");

}

function uploadNext() {

var xhr = new XMLHttpRequest(); var fd = new FormDataO;

var file = document.getElementByld(1 files 1).

files[filesUploaded]; fd.append("multipartFile", file); xhr.upload.addEventListener(

"progress", onUploadProgress, false); xhr.addEventListener("load", onUploadComplete, false); xhr.addEventListener ("error11, onUploadFailed, false); xhr.open("POST", Mfile\_uploadM); debug(1 uploading 1 + file-name); xhr.send(fd);

}

function startUpload() {

totalUploaded = filesUploaded = 0; uploadNext();

}

window.onload = function() {

document■getElementByld(1 files 1)■addEventListener <

1 change 1, onFileSelect, false); document.getElementByld(1uploadButton1).

addEventListener(1 click1, startUpload, false);

}

</script>

</head>

<body>

<hl>Multiple file uploads with progress bar</hl>

<div id=1progressBar1 style=1 height:20px;border:2px solid green’〉

<div id=fbar!

style=1 height:100%;background:#33dd33;width:0%1>

</div>

<form>

<input type=llfile" id=”files” multiple/〉

<br/>

<output id=nselectedFilesMx/output>

<input id="uploadButton" type="button" value=MUploadn/>

</form>

<div id=,debug1

style=1 height:lOOpx;border:2px solid green;overflow:auto1>

</div>

</body>

</html>

html5.jsp页面的用户界面中主要包含了一个名为progressBar的div元素、一个表单和另 一个名为debug的div元素。也许你已经猜到了，progressBar div用于展示上传进度，debug用

于调试信息。表单中有一个类型为file的input元素和一个按钮。

这个表单中有两点需要注意。第一，是标识为files的input元素，它有一个multiple属性， 用于支持多文件选择。第二，这个按钮不是一个提交按钮。因此，单击它并不会提交表单。 事实上，脚本是利用XMLHttpRequest对象来完成上传的。

下面来看JavaScript代码。我们假定读者已经具备一定的脚本语言知识。

执行脚本时，它做的第一件事就是为这4个变量分配空间：

var totalFileLength, totalUploaded, fileCount, filesUploaded;

totalFileLength变量保存要上传的文件总长度。totalUploaded是指目前已经上传的字节数。 fileCount中包含了要上传的文件数量。filesUploaded表示已经上传的文件数量。

随后，当窗口完全下载后，便调用赋予window.onload的函数。

window,onload = function() {

document.getElementByld(1 files 1) .addEventListener(

1 change1, onFileSelect, false); document.getElementByld(1uploadButton1).

addEventListener(1 click1, startUpload, false);

}

这段代码将files input元素的change事件映射到onFileSelect函数，将按钮的click事件 映射到 startUpload。

每当用户从本地目录中修改了不同的文件时，都会触发change事件。与该事件相关的事件 处理器只是在一个output元素中输出已选中的文件的名称和容量。下面是一个事件处理器的例子:

function onFileSelect (e) {

var files = e.target.files; // FileList object var output **=[];** fileCount = files.length; totalFileLength =0; for (var i=0; i<fileCount; i++) { var file = files[i]; output.push(file.name, 1 (1,

file.size, 1 bytes, 1,

file.lastModifiedDate.toLocaleDateString(), 1)

)；

output.push(1<br/>1); debug(1 add 1 + file.size); totalFileLength += file.size;

}

document.getElementByld(1selectedFiles1).innerHTML = output.join (11)；

debug{1totalFileLength:1 + totalFileLength);

当用户单击Upload按钮时，就会调用startUpload函数，并随之调用uploadNext函数。 uploadNext上传已选文件列表中的下一个文件。它首先创建一个XMLHttpRequest对象和一个 FormData对象，并将接下来要上传的文件添加到它的后面。

var xhr = new XMLHttpRequest(); var fd = new FormData();

var file = document.getElementByld(1 files 1)- files[filesUploaded]; fd.append(MmultipartFileM, file);

随后，uploadNext 函数将 XMLHttpRequest 对象的 progress 事件添加到 onUploadProgress，

并将 load 事件和 error 事件分别添加到 onUploadComplete 和 onUploadFailed。

xhr.upload.addEventListener(

"progress", onUploadProgress, false); xhr.addEventListener("load", onUploadComplete, false); xhr.addEventListener("error", onUploadFailed, false);

接下来，打开一个服务器连接，并发出FormData。

xhr.open("POST", Mfile\_uploadM); debug(1 uploading 1 + file.name); xhr.send(fd);

在上传期间，会重复地调用onUploadProgress函数，让它有机会更新进度条。更新包括

计算已经上传的总字节数比率，计算已选择文件的字节数，拓宽progressBardiv元素里面的 div 素 〇

function onUploadProgress(e) { if (e.lengthComputable) {

var percentComplete = parselnt(

(e.loaded + totalUploaded) \* 100 / totalFileLength);

var bar = document.getElementByld(1 bar1)； bar,style.width = percentComplete + 1 % \*; bar.innerHTML = percentComplete + 1 % complete1; } else {

debug(1 unable to compute 1);

上传完成时，调用onUploadComplete函数。这个事件处理器会增加totalUploaded，即 已经完成上传的文件容量，并添加filesUploaded值。随后，它会查看已经选中的所有文件 是否都已经上传完毕。如果是，则会显示一条消息，告诉用户文件上传已经成功完成。如 果不是，则再次调用uploadNext。为了便于阅读，将onUploadComplete函数重新复制到

这里。

function onUploadComplete(e) {

totalUploaded += document.getElementByldf1files1)• files[filesUploaded].size; filesUploaded++;

debug (1 complete 1 + filesUploaded + ’’ of n + fileCount); debug{1totalUploaded: 1 + totalUploaded); if (filesUploaded < fileCount) { uploadNext ();

} else {

var bar = document.getElementByld(1 bar1)；

bar.style.width = \*100%';

bar.innerHTML = 1100% complete 1;

alert(1 Finished uploading file (s) 1)；

}

}

利用下面的URL可以对上述应用程序进行测试：

<http://localhost:8080/upload2/html5>

选择几个文件，并单击Upload按钮，将会看到一个进度条，以及上传文件的信息，屏 幕截图如图11.3所示。

W 0 l〇calhost:8〇8〇/up. 、V

C C locaEhost：8〇80/uplMd2/html5 ☆ £□=

Multiple file uploads with progress bar

100% complete

Clxwse Files 10 files

AbstractPiocessorxlass (6158 bytes, 16/03/2016}

AbstraaPT〇cessor.java(6919 bytes, 18/08/2015)

AbstraciPTOtocolxlass (13737 bytes, 16/03/2016)

Abstract ProtocoLjava (33923 bytes, 1&08/2015)

AbslractPTOtocolSAbstractConneclionHandler.dass (11231 bytes, 16/03/2016) AbstraaPTOtocolSRecydedProcessors.class (2756byies. 16/03/2016) AaionCodexIass (3853 bytesf 16/03/2016)

ActiortCode.java (6632 bytes, 18/08/2015)

AcdtmHookxlass (190 bytes, 16/03/2016)

ActionHcx)kjava(1628 bytes, 18/08/2015)

jmsinj

add 6158 add 6919 add 13737 add 33923

图11.3带进度条的文件上传

11-11小结

**本章介绍了如何在**SpringMVC**应用程序中处理文件上传。处理已上传的文件有两种方 法，即利用**CommonsFileUpload**组件，或者利用**Servlet **3本地文件上传特性。本章提供的范 例展示了如何使用这两种方法。**

**本章还介绍了如何利用**HIML**5支持多文件上传，并利用**FileAPI**提升客户端的用户体验。**

第12章 下载文件

像图片或者HTML文件这样的静态资源，在浏览器中打开正确的URL即可下载。只要 该资源是放在应用程序的目录下，或者放在应用程序目录的子目录下，而不是放在WEB-INF 下，Servlet/JSP容器就会将该资源发送到浏览器。然而，有时静态资源是保存在应用程序目 录之外，或者是保存在某一个数据库中，或者有时需要控制它的访问权限，防止其他网站交 叉引用它。如果出现以上任意一种情况，都必须通过编程来发送资源。

简言之，通过编程进行的文件下载，使你可以有选择地将文件发送到浏览器。本章将介 绍如何通过编程把资源发送到浏览器，并举两个范例。

12.1文件下载概览

为了将像文件这样的资源发送到浏览器，需要在控制器中完成以下工作：

1. 对请求处理方法使用void返回类型，并在方法中添加HttpServletResponse参数。
2. 将响应的内容类型设为文件的内容类型。Content-Type标题在某个实体的body中定 义数据的类型，并包含媒体类型和子类型标识符。欲了解标准的内容类型，请访问 [http://www.iana.org/assignments/media-types。如果不清楚内容类型，并且希望浏览器始终显示](http://www.iana.org/assignments/media-types%e3%80%82%e5%a6%82%e6%9e%9c%e4%b8%8d%e6%b8%85%e6%a5%9a%e5%86%85%e5%ae%b9%e7%b1%bb%e5%9e%8b%ef%bc%8c%e5%b9%b6%e4%b8%94%e5%b8%8c%e6%9c%9b%e6%b5%8f%e8%a7%88%e5%99%a8%e5%a7%8b%e7%bb%88%e6%98%be%e7%a4%ba) Sava As (另存为）对话框，则将它设为APPLICATION/OCTET-STREAM。这个值是不区分 大小写的。
3. 添加一个名为 Content-Disposition 的 HTTP 响应标题，并赋值 attachment; filename= fileName，这里的fileName是默认文件名，应该出现在File Download (文件下载）对话框中。

它通常与文件同名，但是也并非一定如此。

例如，以下代码将一个文件发送到浏览器：

FilelnputStream fis = new FilelnputStream(file);

BufferedlnputStream bis = new BufferedlnputStream(fis); byte[] bytes = new byte[bis.available()]; response.setContentType(contentType);

OutputStream os = response.getOutputStream(); bis.read(bytes); os.write(bytes);

为了编程将一个文件发送到浏览器，首先要读取该文件作为FilelnputStream,并将内容 加载到一个字节数组。随后，获取HttpServletResponse的OutputStream,并调用其write方法

传入字节数组。

将文件发送到HTTP客户端的更好方法是使用JavaNIO的Files.copyG方法：

Path file = Paths.get(...);

Files.copy(file, response.getOutputStream());

代码更短，运行速度更快。

12.2范例1:隐藏资源

download应用程序展示了如何向浏览器发送文件。在这个应用程序中，由ResourceController 类处理用户登录，并将一个secret.pdf文件发送给浏览器。secretpdf文件放在WEB-INF/data 目录下，因此不可能直接访问。只有得到授权的用户，才能看到它。如果用户没有登录，应 用程序就会跳转到登录页面。

清单12.1中的ResourceController类提供了一个控制器，负责发送secret.pdf文件。只有 当用户的HttpSession中包含一个loggedln属性时，表示该用户已经成功登录，才允许该用户 访问。

清单 **12.1 ResourceController** 类

package controller; import java•io•工OException; import java.nio.file.Files; import j ava.nio.file.Path; import j ava.nio.file.Paths;

import j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest); import j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse); import j avax.servlet.[http.HttpSession](http://http.HttpSession); import org.apache.commons.logging.Log;

12.2范例1:隐藏资源

import org.apache.commons.logging.LogFactory; import org.springframework.stereotype.Controller; import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.ModelAttribute; import org.springframework,web.bind.annotation,RequestMapping; import domain.Login;

QController

public class ResourceController {

private static final Log logger =

LogFactory.getLog(ResourceController.class);

@RequestMapping (value=’’/login”）

public String login(@ModelAttribute Login login, HttpSession session, Model model) { model.addAttribute("login”， new Login()); if (Mpauln.equals(login,getUserName{)) &&

"secret11. equals (login, getPassword {) ) ) { session.setAttribute("loggedln", Boolean.TRUE); return ’’Main’1;

} else {

return "LoginForm";

@RequestMapping(value=M/download-resourceM) public String downloadResource(HttpSession session,

HttpServletRequest request, HttpServletResponse response, Model model) { if (session == null ||

session.getAttribute("loggedln") == null) { model,addAttribute("login", new Login{)); return "LoginForm11;

}

String dataDirectory = request,

getServletContext().getRealPath("/WEB-INF/datan);

Path file = Paths . get (dataDirectory, "secret .pdf11); if (Files.exists(file)) {

response.setContentType(napplication/pdfM); response.addHeader("Content-Disposition",

"attachment; filename=secret.pdfM);

try {

Files.copy(file, response.getOutputStream()); } catch (IOException ex) {

return null;

控制器中的第一个方法login,将用户带到登录表单。

LoginForm.jsp页面如清单12.2所示。

清单 12.2 LoginForm.jsp 页面

<%@ taglib pref ix=Mform11 uri=Mhttp: //www. springframework.org/tags/form" %> <%@ taglib prefix=ncM uri=M<http://j> ava.sun.com/j sp/j stl/coreM %>

<!DOCTYPE HTML>

<html>

<head>

<title>Login</title>

<style type=Mtext/css">@import url("<c:url value=”/css/main.css"/>n);</style>

</head>

<body>

<div id="globaln>

<form:form commandName=Mloginn action=Mlogin" method=MpostM>

<fieldset>

<legend>Login</legend>

<p>

<label for=lluserNameM>User Name: </label>

<f orm: input id=,,userNameM path=MuserNameMcssErrorClass="error"/>

</p>

<p>

<label for=Mpassword">Password: </label>

< form: pas sword id=npas sword” path=’’pas sword" cssErrorClass:11 error11/>

</p>

<p id="buttons">

<input id="reset" type="reset" tabindex="4">

<input id=11 submit11 type=Msubmit11 tabindex="5" value="Login">

</p>

</fieldset>

</form:form>

</div>

</body>

</html>

12.3范例2:防止交叉引用

成功登录所用的用户名和密码必须在login方法中进行硬编码。例如，用户名必须为paul, 密码必须为secret。如果用户成功登录，他就会被转到Mainjsp页面（清单12.3)。Mainjsp页 面中包含了一个链接，用户可以单击它来下载文件。

清单12.3 Main.jsp页面

<%@ taglib ■uri="http: / /java . sun. com/j sp/j stl/coref, pref ix=,,cM %>

<!DOCTYPE HTML>

<html>

<head>

<title>Download Page</title>

<style type=,,text/cssM>@import url ("<c:url value=’’/css/main.css■’/>") ;</style>

</head>

<body>

<div id=nglobaln>

<h4>Please click the link below.</h4>

<P>

<a href=,,resource\_downloadn>Download</a>

</p>

</div>

</body>

</html>

ResourceController 类中的第二个方法 downloadResource，它通过验证 session 属性 loggedln 是否存在，来核实用户是否已经成功登录。如果找到该属性，就会将文件发送给浏览器。如 果没有找到，用户就会转到登录页面。注意，如果使用Java 7或其更高版本，则可以使用其 新的try-with-resources特性，从而更加安全地处理资源。

通过调用以下URL中的FileDownloadServlet，可以测试download应用程序。

<http://localhost:8080/download/login>

12.3范例2:防止交叉引用

心怀叵测的竞争对手有可能通过交叉引用“窃取”你的网站资产，例如，将你的资料 公然放在他的网站上，好像那些东西原本就属于他的一样。如果通过编程控制，使得只有 当referer标题中包含你的域名时才发出资源，就可以防止那种情况发生。当然，那些心意 坚决的窃贼仍然有办法下载到你的东西，但是绝不会像以前那样不费吹灰之力就能得到。

download应用程序利用清单12.4中的ImageController类，使得仅当referer标题不为null

时，才将图片发送给浏览器。这样可以防止仅在浏览器中输入网址就能下载图片的情况发生。 清单 12.4 ImageController 类

package controller; import java.io.IOException; import j ava,nio.file.Files; import j ava.nio.file.Path; import j ava.nio.file.Paths;

import j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest);

import j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse);

import org.apache,commons.logging.Log;

import org.apache.commons.logging.LogFactory;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestHeader;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

SController

public class ImageController {

private static final Log logger =

LogFactory.getLog(ImageController.class);

@RequestMapping(value=M/image\_get/{id}method = RequestMethod.GET) public void getlmage(@PathVariable String id,

HttpServletRequest request, HttpServletResponse response, @RequestHeader String referer) { if (referer != null) {

String imageDirectory = request.getServletContext().

getRealPath(M/WEB-INF/imageM);

Path file = Paths.get(dataDirectory, id + ".jpg"); if (Files.exists(file)) {

response . setContentType ("image/jpg11); try {

Files■copy{file, response,getOutputStream());

} catch (IOException ex) { e.printStackTrace();

原则上，ImageController类的作用与ResourceController无异。getlmage方法开头处的if 语句，可以确保只有当referer标题不为imll时，才发出图片。

清单 12.5 images.html 文件

利用清单12.5中的imageS.litnil文件，可以对这个应用程序进行测试。

<!DOCTYPE html>

<html>

<head>

<title>Photo Gallery</title>

</head>

<body>

<img src= "get-image/l,,/>

<img src="get-image/2n/>

<img src="get-image/3"/>

<img src="get-image/4"/>

<img src="get-image/5"/>

<img src="get-image/6"/>

<img src="get-image/7,'/>

<img src="get-image/8"/>

<img src="get-image/9”/>

<img src="get-image/10"/>

</body>

</html>

要想看到ImageServlet的效果，请在浏览器中打开以下网址:

<http://localhost:8080/download/images.html>

图12.1展示了使用ImageServlet后的效果。

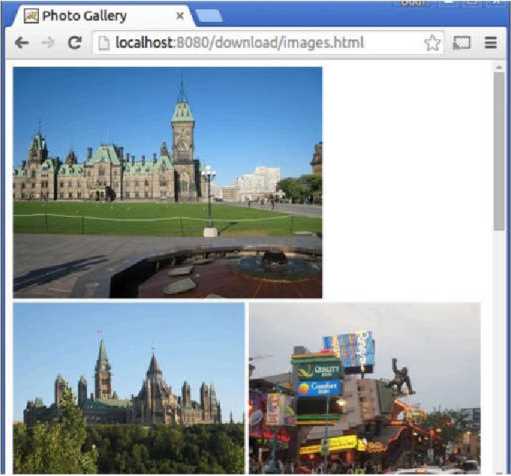


图12.1使用ImageServlet后的效果

**12.4**小结

本章学习了如何在Spring MVC应用程序中通过编程来控制文件的下载，还学习了如何选 择文件，以及如何将它发送给浏览器。

第13章 应用测试

测试在软件开发中的重要性不言而喻。测试的主要目的是尽早发现错误，最好是在代码 开发的同时。逻辑上认为，错误发现的越早，修复的成本越低。如果你在编程时发现错误， 可以立即更改代码；如果队友在你将代码发布到共享存储库之后发现了错误，则现在至少涉 及两个人；想象一下，当软件发布后，客户发现错误后所需要的修复成本会更大。

在软件开发中有许多不同的测试。其中两个是单元测试和集成测试。你通常从单元测试 开始测试类中的单个方法，然后进行集成测试，以测试不同的模块是否可以无缝协同工作。

本章中的示例使用JUnit测试框架以及Spring-test模块。Spring-test模块中的API可用于单 元测试和集成测试。你可以在org.springframework.test及其子包以及org.springframework.mock.\* 包中找到Spring测试相关的类型。本章附带的示例分为两个项目，称为单元测试和集成测试。 第一个项目包含类和单元测试。第二个项目包含类和集成测试。

13.1单元测试

单元测试的理想情况是为每个类创建一个测试类，并为类中的每个方法创建一个测试方 法，像getter和setter方法这样的简单方法除外，它们直接从字段返回值或赋值给字段。

在测试用语中，被测试的类称为被测系统（SUT)。

单元测试旨在快速且多次运行。单元测试仅验证代码本身，而不涉及它的依赖，其任何 依赖应该被帮助对象替代，这将在本章后面解释。涉及依赖性的测试通常在集成测试中完成， 而不是单元测试。

乍一看，写单元测试看起来像不必要的额外工作。毕竟，你可以使用main方法从类本身 内测试一个类。但是，你应该考虑单元测试的好处。首先，在单独的测试类中编写测试代码

不会混淆你的类；第二，单元测试可以用于回归测试，在一些逻辑发生变化时，以确保一切 仍然工作；单元测试的另一个主要优点是，它可以在持续集成设置中自动化测试。持续集成 是指一种开发方法，当程序员将他们的代码提交到到共享库时，每次代码提交将触发一次自 动构建并运行所有单元测试。持续集成可以尽早地检测问题。

在单元测试中，类使用new运算符实例化。不依赖Spring框架的依赖注入容器来创建bean。

让我们来看看清单13.1中的类。

清单13.1被测试类

package com.example,util; public class MyUtility {

public int methodl(int a, int b) { ... } public long method2(long a) { ... }

为了对这个类进行单元测试，创建一个如清单13.2所示的类。注意每个方法应该至少有 一个测试方法。

清单13.2测试类

package com.example.util; public class MyUtilityTest {

public void testMethodl() {

MyUtility utility = new MyUtility(); int result = utility.methodl(100, 200);

// assert that result equals the expected value

public void testMethod2() {

MyUtility utility = new MyUtility(); long result = utility.method2(100L);

// assert that result equals the expected value

}

}

单元测试有些约定俗成，首先是将测试类命名为与带有Test的SUT相同的名称。因此， MyUtility的测试类应命名为MyUtilityTest;其次，测试类的包路径应与SUT相同，以允许前 者访问后者的受保护和默认成员。但是，测试类应位于不同于测试的类的源文件夹下。

13.3 应用 JUnit

测试方法没有返回值。在测试方法中，你实例化要测试的类，调用要测试的方法并验证 结果。为了使测试类更容易编写，你应该使用测试框架，例如JUnit或TestNG。本章介绍用 JUnit编写测试类的例子，JUnit是Java的事实上的标准单元测试框架。

13.2状态测试与行为测试

大多数时候，你只关心方法是否返回正确的结果。这称为状态测试。但是有时候，你还需 要执行行为测试（也称为交互测试)，以确保方法的行为正确。例如，你可能想要验证方法以正 确的顺序调用其他方法。或者，你可能需要确保你的方法在操作期间调用另一个方法正好n次。

你将在本章后面学习如何执行状态测试和行为测试。

1. 应用 JUnit

对于单元测试，我推荐使用JUnit,你可以从http://junit.org下载它。点击下载并安装链接 后，你将被重定向到下载页面。你需要下载junitjar和hamcrest-core.jar文件。后者是JUnit 的依赖项，目前版本为4.12。

如果使用Maven或STS，请将此元素添加到pom.xml文件以下载JUnit及其依赖关系。

<dependency>

<groupId> junit </ groupId>

<artifactld> junit </ artifactld>

<version> 4.12 </ version〉

<scope>test</scope>

</ dependency〉

JUnit也广泛用于集成测试，你将在本章后面了解这一点。

你可以通过阅读JUnit网站上的文档，了解有关JUnit的更多信息以及如何编写JUnit测试。

13.3.1开发一个单元测试

编写单元测试很容易。使用@org.junit.TeSt简单地注解所有测试方法。此外，你可以通过 使用@ orgjunitBefore注解来创建初始化方法。初始化方法在调用任何测试方法之前调用， 你可以编写准备测试方法的代码，例如创建将由测试方法使用的对象。

你还可以通过使用@ orgjunitAfter注解方法来创建清理方法。清理方法在测试类中的所

有测试方法都被执行之后调用，并且可以用来释放测试期间使用的资源。 清单13.3显示了需要进行单元测试的Calculator类。 清单 13.3 Calculator 类

package com,example; public class Calculator {

public int add(int a, int b) { return a + b;

public int subtract(int a, int b) {

return a

**b;**

清单13.4显示了 Calculator类的单元测试类。 清单13.4单元测试类

package com.example; import org.junit.After; import org.junit.Assert; import org.junit.Before; import org.junit.Test;

public class CalculatorTest {

@Before

public void init() {

}

0After

public void cleanup{) {

}

0Test

public void testAdd{) {

Calculator calculator = new Calculator(); int result = calculator.add(5, 8);

Assert.assertEquals(13, result);

@Test

public void testSubtract{) {

Calculator calculator = new Calculator(); int result = calculator.subtract(5, 8);

Assert.assertEquals(-3**,** result);

13.3 应用 JUnit

CalculatorTest类有两个测拭方法、一个初始化方法（用@Before注解）和一个清除方法 (用@Aiter\_ )。org.junit.Assert类提供用于声明结果的静态方法。例如，assertEquals方法 用来比较两个值。

13.3.2运行一个单元测试

Eclipse和STS知道一个类是否是一个JUnit测试类。要运行测试类，请右键单击包资源管理 器中的测磯，然后选择运行方式> JUnit测试。或者，使用编辑器打开测織，然后按CM+F11键。

测试完成后，如果JUnit视图尚未打开，Eclipse或STS将打开它。如果单元测试成功完 成，你将在JUnit视图中看到一个绿色条，如图13.1所示。

如果其中一个测试失败，你将看到一个红色条，如图13.2所示。

Package Type Hie ^ Navigato ^ JUnit S3 口 □ aG al % 1 - ^

Finished after O.OZ2 seconds Runs: 1/1 s Errors: 0 0 Failures: 0

**^ com.example.MyClassTest [Runner: JUnit 4] (0.000 s)**

(3 Package |j Type Hie ^ Navigato JUnit

办分，翩 % ©> i Finished after 0.053 seconds

Runs: 1/1 Q Errors: 0 Q Failures: 1

v is com.example.MyClassTest [Runner: JUnit 4] (0.001 s) iiltestAdd (0.001 s)

=Failure Trace | ^sj g£

JS java.lang.AssertionError: expected:<63> but was:<6>

**Failure Trace**

**=**at com.example.MyClassTesLte5tAdd(MyClassTest.java:24)

图13.2 —个没通过的测试

图13.1 —个通过的测试

13.3.3通过测试套件来运行全部或多个单元测试

在有十几个类的一个小项目中，你将有十几个测试类。在一个更大的项目中，你会有更 多测试类。在Eclipse或STS中，运行一个测试类很容易，但如何运行所有的测试类？

使用JUunit的解决方案非常简单。创建一个Java类并使ffl@RunWith (Suite.class)和 @SuiteClasses〇注解它。后者应该列出你想运行的所有类和其他套件测试。清单13.5显示了一 个示例。类体可以留空。

清单13.5 —个测试套件

package com.example;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)

0SuiteClasses({ MyTestl•class, MyTest2•class }) public class MyTestSuite {

}

如果使用ci服务器，则可以将服务器设置为每次提交新代码或修改的代码时自动运行单 元测试。

**13.4**应用测试挡板**（Test Doubles)**

被测系统（SUT)很少孤立存在。通常为了测试一个类，你需要依赖。在测试中，你的 SUT所需要的依赖称为协作者。

协作者经常被称为测试挡板@的其他对象取代。测试挡板是GerardMeszaros在他的书《xUnit Test Patterns: Refaltoning your code》中创造的一个术语。他对该术语的解释可以在这个网页中找 到：<http://xunitpattems.com/Test%20Double.html>

使用测试挡板有几个原因。

•在编写测试类时，真正的依赖还没有准备好。

* 一些依赖项，例如 HttpServletRequest 和 HttpServletResponse 对象，是从 servlet 容器

获取的，而自己创建这些对象将会非常耗时。

•一些依赖关系启动和初始化速度较慢。例如，DA0对象访问数据库导致单元测试执行很慢。

测试挡板在单元测试中广泛使用，也用于集成测试。当前有许多用于创建测试挡板的框 架。Spring也有自己的类来仓键测试挡板。

模拟框架可用于创建测试挡板和验证代码行为。这里有一些流行的框架。

* Mockito。
* EasyMock〇

①译者注：Test Doubles又翻译为测试替身，本书翻译为测试挡板。

* jMock〇

除了上面的库，Spring还附带了创建模拟对象的类。在下一节中，我们将使用Spring类 来测试SpringMVC控制器。在本节中，我们将学习使用Mockito。

Mockito无法直接下载，所以你必须使用Maven。但是，本章的示例项目包括Mockito发 布包（一个mockito.jar文件）及其依赖（一个objenesis.jar文件）。

要使用Maven下载Mockito，请将以下依赖关系添加到pom.xml文件中。

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactld>

<version>2.0.43-beta</version>

<type>pom</type>

</dependency>

在开始写测试挡板之前，你应该先学习理论知识。如下是测试挡板的5种类型。

參 dummy;

* stub；

參 spy;

* fake；
* mock〇

这些类型中的每一种将在下面的小节中解释。

1. dummy

dummy是最基本的测试挡板类型。一个dummy是一个协作者的实现，它不做任何事情， 并且不改变SUT的行为。它通常用于使SUT可以实例化。dummy只在开发的早期阶段使用。

例如，考虑清单13.6中的ProductServicelnqjl类。这个类依赖于传递给构造函数的ProductDAO。 清单 13.6 ProductServicelmp丨类

package com,example.service; import com.example.dao.ProductDAO;

public class ProductServiceImp1 implements ProductService {

private ProductDAO productDAO;

public ProductServicelmpl(ProductDAO productDAOArg) { if (productDAOArg == null) {

throw new NullPointerException("ProductDAO cannot be null.");

}

this.productDAO = productDAOArg;

0〇verride

public BigDecimal calculateDiscount() {

return productDAO.calculateDiscount{);

}

0〇verride

public boolean isOnSale(int productld) { return productDAO.isOnSale(productld);

}

}

ProductServicelmpl类需要一个非空的ProductDAO对象来实例化。同时，要测试的方 法不使用ProductDAO。因此，可以创建一个如清单13.7所示的dummy对象，只需让 ProductServicelmpl 可实例化。

清单 13.7 ProductDAODummy 类

package com.example.dummy; import j ava.math.BigDecimal; import com.example.dao.ProductDAO;

public class ProductDAODummy implements ProductDAO { public BigDecimal calculateDiscount() { return null;

}

public boolean isOnSale(int productld) { return false;

在d\_y类中的方法实现什么也不做，它的返回值也不重要，因为这些方法从未使用过。 清单13.8显示了一个可以运行的测试类

清单 13.8 ProductDAOTest 类

package com.example.dummy;

import static org.junit.Assert.assertNotNull; import org.junit.Test;

import com.example.dao.ProductDAO;

import com.example.service.ProductService;

import com.example.service.ProductServicelmpl;

public class ProductServicelmplTest {

0Test

public void testCalculateDiscount() {

ProductDAO productDAO = new ProductDAODummy();

ProductService productService =

new ProductServicelmpl(productDAO); assertNotNull(productService);

}

}

1. stub

像dummy—样，stub也是依赖接口的实现。和dummy不同的是，stub中的方法返回硬 编码值，并且这些方法被实际调用。

清单13.9显示了一个stub，可用于测试清单13.6中的ProductServicelmpl类。

清单 13.9 ProductDAOStub 类

package com.example.stub; import j ava.math.BigDecimal; import com.example.dao.ProductDAO;

public class ProductDAOStub implements ProductDAO { public BigDecimal calculateDiscount() { return new BigDecimal(14);

}

public boolean isOnSale(int productld) { return false;

}；

}

1. spy

spy是一个略微智能一些的stub，因为spy可以保留状态。考虑下面的汽车租赁应用程序， 其中包含一个GarageService接口和一个GarageServicelmpl类，分别在清单13.10和清单13.11中。

清单 13.10 GarageService 接口

package com,example.service; import com.example.Car;

public interface GarageService {

Car rent();

}

清单 13.11 GarageServicelmp 丨类

package com.example.service;

import com.example.Car;

import com.example.dao.GarageDAO;

public class GarageServiceImp1 implements GarageService { private GarageDAO garageDAO;

public GarageServicelmpl(GarageDAO garageDAOArg) { this.garageDAO = garageDAOArg;

}

public Car rent() {

return garageDAO.rent <);

}

}

GarageService 接口只有一个方法：rent。GarageServicelmpl 类是 GarageService 的一个实 现，并且依赖一个 GarageDAO。GarageServicelmpl 中的 rent 方法调用 GarageDAO 中的 rent 方法。GarageDAO的实现应该返回一个car,如果还有一辆汽车在车库；或者返回null,如果

没有更多的汽车。

由于GarageDAO的真正实现还没有完成，清单13.12中的GarageDAOSpy类被用作测试

挡板。它是一个spy，因为它的方法返回一个硬编码值，并且它通过一个carCount变量来保 存车库里的车数。

清单 13.12 GarageDAOSpy 类

package com.example,spy;

import **coitk** example ■ Car ;

import com.example.dao.GarageDAO;

public class GarageDAOSpy implements GarageDAO { private int carCount = 3;

@〇verride

public Car rent() {

if (carCount == 0) {

return null;

} else {

carCount——;

return new Car();

清单13.13显示了使用GarageDAOSpy测试GarageServicelmpl类的一个测试类。 清单 13.13 GarageServicelmpITest 类

package com.example.spy;

import com.example.Car;

import com.example,dao.GarageDAO;

import com.example,service,GarageService;

import com.example,service.GarageServicelmpl;

import org.junit.Test;

import static org.junit.Assert.\*;

public class GarageServicelmpITest {

@Test

public void testRentCar() {

GarageDAO garageDAO = new GarageDAOSpy();

GarageService garageService = new GarageServicelmpl(garageDAO); Car carl = garageService.rent();

Car car2 = garageService.rent();

Car car3 = garageService.rent();

Car car4 = garageService.rent();

assertNotNull(carl); assertNotNull(car2); assertNotNull(car3); assertNull(car4);

由于在车库只有3辆车，spy只能返回3辆车，当第四次调用其rent方法时，返回null。

1. fake

fake的行为就像一个真正的合作者，但不适合生产，因为它走“捷径”。内存存储是一个 fake的完美示例，因为它的行为像一个DAO,但不会将其状态保存到硬盘驱动器。

例如，分别考虑清单13.14和清单13.15中的Member和MemberServicelmpl类。Member 类包含成员的标识符和名称。MemberServicelmpl类可以将成员添加到商店并检索所有存储的 成员。

清单 13.14 Member 类

package com.example.model; public class Member { private int id; private String name;

public Member(int idArg, String nameArg) { this.id = idArg; this.name = nameArg;

}

public int getld() { return id;

public void setld(int idArg) { this.id = idArg;

}

public String getName() { return name;

}

public void setName(String nameArg) { this.name = nameArg;

清单 13.15 MemberServicelmp丨类

package com.example.service; import java.util.List; import com.example.dao.MemberDAO; import com.example.model.Member;

public class MeraberServicelmpl implements MemberService { private MemberDAO memberDAO;

public void setMemberDAO(MemberDAO memberDAOArg) { this.memberDAO = memberDAOArg;

@Override

public void add(Member member) { memberDAO.add(member);

}

@〇verride

public List<Member> getMembers() {

return memberDAO.getMembers();

MemberServicelmpl依赖于MemberDAO。但是，由于没有可用的MemberDAO实现，你 可以创建一个MemberDAO的fake实现，以便可以立即测试MemberServicelmpl。清单13.16 显示了这样一个fake。它将成员存储在ArrayList中，而不是持久化存储。因此，不能在生产 中使用它，但是对于单元测试是足够的。

清单 13.16 MemberDAOFake 类

package com.example.fake; import java.util.ArrayList; import j ava.util.List; import com.example.dao.MemberDAO; import com.example.model.Member;

public class MemberDAOFake implements MemberDAO {

private List<Member> members = new ArrayListo{);

@Override

public void add(Member member) { members.add(member);

0〇verride

public List<Member> getMembers() { return members;

}

}

清单13.17显示了一个测试类，它使用MemberDAOFake作为MemberDAO的测试挡板 类来测试 MemberServicelmpl。

清单 13.17 MemberServicelmpITest 类

package com.example.service;

import org.junit.Assert;

import org.junit.Test;

import com.example.dao.MemberDAO;

import com.example.fake.MemberDAOFake;

import com.example.model.Member;

public class MemberServicelmpITest {

@Test

public void testAddMember() {

MemberDAO memberDAO = new MemberDAOFake(); memberDAO. add (new Member (1, "John Diet’’））；

memberDAO.add(new Member(2, "Jane Biteman"));

Assert.assertEquals(2, memberDAO.getMembers().size());

1. mock

mock在理念上不同于其他测试挡板。使用dummy、stub、spy和fake来进行状态测试， 即验证方法的输出。而使用mock来执行行为（交互）测试，以确保某个方法真正被调用，或 者验证一个方法在执行另一个方法期间被调用了一定的次数。

例如，考虑清单13.18中的MathUtil类。

清单13.18 MathUti丨类

package com.example; public class MathUtil {

private MathHelper mathHelper; public MathUtil(MathHelper mathHelper) { this.mathHelper = mathHelper;

}

public MathUtil() {

public int multiply(int a, int b) { int result = 0;

for (int i = 1; i <= a; i++) {

result = mathHelper.add(result, b);

}

return result;

}

}

MathUtil有一个方法multiply。它非常直接，使用多个add方法。换句话说，3 x 8计算为8 + 8 + 8。MathUtil类甚至不知道如何执行add。为此，它依赖于MathHelper对象，其类在列 表13.19中结出。

清单 13.19 MathHelper 类

package com.example; public class MathHelper {

public int add(int a, int b) {

return a + b;

测试所关心的不是multiply方法的结果，而是找出方法是否如预期一样运行。因此，在 计算3 x 8时，它应该调用MathHelper.add〇 3次。清单13.20显示了一个使用MathHelper模 拟的测试类。Mockito是一个流行的模拟框架，用于创建模拟对象。我将在本章稍后介绍关于 Mockito的更多知识。在这一节中，我只是想告诉你这个概念。

清单 13.20 MathUtilTest 类

package com.example;

import static org.mockito.Mockito.mock; import static org.mockito.Mockito.times; import static org.mockito-Mockito.verify; import static org.mockito.Mockito.when; import org.junit.Test;

public class MathUtilTest {

0Test

public void testMultiply() {

MathHelper mathHelper = mock(MathHelper.class); for (int i = 0; i < 10; i++) {

when(mathHelper.add(i \* 8, 8)).thenReturn(i \* **8+8);**

}

MathUtil mathUtil = new MathUtil(mathHelper); mathUtil.multiply(3, 8); verify(mathHelper, times(1))•add(0, 8); verify(mathHelper, times(1)).add(8, 8); verify(mathHelper, times(1)).add(16, 8);

}

}

使用Mockito创建mock对象非常简单，只需调用org.mockito.Mockito类的静态mock方 法即可。下面展示如何创建MathHelper mock对象。

MathHelper mathHelper = mock ( MathHelper.class );

接下来，你需要使用when方法准备mock对象。基本上，你告诉它，给定使用这组参数 的方法调用，mock对象必须返回这个值。例如，这条语句是说如果调用mathHelper.add(10， 20)，返回值必须是10 + 20:

when(mathHelper.add (10> 20) ) .thenReturn(10 + 20);

对于此测试，你准备具有十组参数的mock对象（但不是所有的参数组都会被使用）。

for (int i = 0; i < 10; i++) {

when(mathHelper.add(i \* 8, 8)).thenReturn(i \* 8 + 8);

}

然后创建要测试的对象并调用其参数。

MathUtil mathUtil = new MathUtil (mathHelper); mathUtil.multiply (3, 8);

接下来的3条语句是行为测试。为此，调用verify方法：

verify (mathHelper, times (1)) . add ( 〇5 8); verify (mathHelper, times (1)) .add ( 8, 8); verify (mathHelper, times (1)) .add (16，8 );

第一条语句验证mathHelper.add (0，8)被调用了一次。第二条语句验证mathHelper.add (8,8)被调用一次，第三条语句验证mathHelper.add (16，8)也被调用一次。

1. 对 **Spring MVC Controller** 单元测试

你已经学习了如何在SpringMVC应用程序中测试各个类。但Controller有点不同，因为 它们通常与 Servlet API 对象（如 HttpServletRequest、HttpServletResponse、HttpSession 等）

交互。在许多情况下，你将需要模拟这些对象以正确测试控制器。

像Mockito或EasyMock这样的框架是可以模拟任何Java对象的通用模拟框架。你必须 自己配置生成的对象（使用一系列when语句。Spring-Test模拟对象是专门为使用Spring而 构建的，并且与真实对象更接近，更容易使用。

以下小节讨论了一些更重要的单元测试控制器类型。

1. MockHttpServletRequest 和 MockHttpServletResponse

当调用控制器时，你可能需要传递HttpServletRequest和HttpServletResponse 〇在生产环境中， 两个对象都由servlet容器本身提供。在测试环境中，你可以使用org.springframework.mock.web 包中的 Spring MockHttpServletRequest 和 MockHttpServletResponse 类。

这两个类很容易使用。你可以通过调用其无参构造函数来创建实例：

MockHttpServletRequest request = new MockHttpServletRequest(); MockHttpServletResponse response = new MockHttpServletResponse();

MockHttpServletRequest 类实现了 javax.servlet.http.HttpServletRequest，并允许你将实例配 置将看起来像一个真正的HttpServletRequest。它提供了方法来设置HttpServletRequest中的所

有属性以及获取其属性的值。表13.1显示了它的一些方法。

表 13.1 MockHttpServletRequest 的主要方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| addHeader | 添加一个HTTP请求头 |
| addParameter | 添加一个请求参数 |
| getAttribute | 返回一个属性 |
| getAttributeNames | 返回包含了全部属性名的一个Enumeration对象 |
| getContextPath | 返回上下文路径 |
| getCookies | 返回全部的cookies |
| setMethod | 设置HTTP方法 |
| setParameter | 设置一个参数值 |
| setQueryString | 设置查询语句 |
| setRequestURI | 设置请求URI |

MockHttpServletResponse 实现了 javax.servlethttp.HttpServletResponse，并提供了配置实 例的其他方法。表13.2显示了其一些主要的方法。

表 13.2 MockHttpServletResponse 的主要方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| addCookie | 添加一个cookie |
| addHeader | 添加一个HTTP响应头 |
| getContentLength | 返回内容长度 |
| getWriter | 返回Writer |
| getOutputStream | 返回 ServletOutputStream |

考虑清单 13.21 的 VideoController 类。 清单 13.21 VideoController 类

package com.example.controller; import javax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest); import j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse); import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

SController

public class VideoController {

@RequestMapping(value = "/mostViewed")

public String getMostViewed(HttpServletRequest request, HttpServletResponse response) {

Integer id = (Integer) request.getAttribute("id"); if (id == null) {

response.setStatus(500);

} else if (id == 1) {

request.setAttribute("viewed", 100);

} else if (id == 2) {

request.setAttribute("viewed", 200);

}

return "mostViewed";

}

}

VideoController类的getMostViewed方法中，若请求属性id存在且值为1或2，则添加请 求属性“viewed”。否则，不添加请求属性。

清单13.22中的VideoControllerTest类使用两个测试方法验证这一点。 清单 13.22 VideoControllerTest 类

package com.example.controller;

import org.junit.Test;

import static org.junit.Assert.\*;

import org.springframework.mock.web.MockHttpServletRequest; import org.springframework.mock.web.MockHttpServletResponse;

public class VideoControllerTest {

@Test

public void testGetMostViewed() {

VideoController VideoController = new VideoController{); MockHttpServletRequest request = new MockHttpServletRequest(); request.setRequestURI("/mostViewed"); request. setAttribute < "id’1, 1);

MockHttpServletResponse response = new MockHttpServletResponse{);

VideoController.getMostViewed(request, response); assertEquals(200, response.getStatus <));

assertEquals(10〇L, (int) request.getAttribute("viewed"));

@Test

public void testGetMostViewedWithNoId() {

VideoController videoController = new VideoController(); MockHttpServletRequest request = new MockHttpServletRequest(); request. setRequestURI {'VmostViewed1');

MockHttpServletResponse response = new MockHttpServletResponse{);

videoController.getMostViewed(request, response); assertEquals(50〇, response.getStatus()); assertNull(request.getAttribute("viewed"));

}

}

testGetMostViewed 方法实例化 VideoController 类并创建两个 mock 对象，一个 MockHttp ServletRequest 和一个 MockHttpServletResponse。它还设置请求 URL 并向 MockHttpServletRequest 添加属性“id”。

VideoController videoController = new VideoController();

MockHttpServletRequest request = new MockHttpServletRequest();

request.setRequestURI(lVmostViewedn);

request ■ setAttribute ("id’1, 1);

MockHttpServletResponse response = new MockHttpServletResponse();

然后调用VideoController的getMostView方法，传递mock对象，然后验证响应的状态码 为200，请求包含一个值为100的“viewed”属性。

videoController.getMostViewed(request, response);

assertEquals(200, response.getStatus());

assertEquals(100L, (int) request.getAttribute("viewed"));

VideoControllerTest中的第二个方法类似方法一，但不会向MockHttpServletRequest对象 添加“id”属性。因此，在调用控制器的方法时，它接收HTTP响应状态代码500，并且在 MockHttpServletRequest 对象中没有 “viewed” 属性。

1. ModelAndViewAssert

ModelAndViewAssert 类是 org.springframework.web.servlet 包的一部分，是另一个有用的 Spring类，用于测试从控制器的请求处理方法返回的ModelAndView。回想一下第4章中介绍 过，ModelAndView是请求处理方法可以返回的类型之一，是包含有关请求处理方法的模型和 视图的信息的一个bean。

表13.3列举了 ModelAndViewAssert的一些主要方法。

表 13.3 ModelAndViewAssert 的主要方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| assertViewName | 检查ModelAndView的视图名称是否与预期名称匹配 |
| assertModelAttributeAvailable | 检查ModelAndView的模型是否包含具有预期模型名称的属性 |
| assertModelAttributeValue | 检查ModelAndView模型是否包含具有指定名称和值的属性 |
| assertSortAndCompareList-ModelAttribute | 对ModelAndView的列表排序，然后将其与预期列表进行比较 |

例如，考虑清单13.23中的Book类。这是一个简单的bean类，有4个属性：isbn、title、 author 和 pubDate

清单13.23 Book类

package com.example.model; import j ava.time•LocalDate;

public class Book {

private String isbn; private String title; private String author; private LocalDate pubDate;

public Book(String isbn, LocalDate pubDate) { this.isbn = isbn; this.pubDate = pubDate;

}

public Book(String isbn, String title, String author, LocalDate pubDate) { this.isbn = isbn; this.title = title; this.author = author; this.pubDate = pubDate;

}

// getters and setters not shown to save space

0〇verride

public boolean equals(Object otherBook) {

return isbn.equals(((Book)otherBook),getlsbn());

清单13.24中的BookController类是一个Spring MVC控制器，它包含一个请求处理方法

getLatestTitles。该方法接受pubYear路径变量，并返回一个ModelAndView，如果pubYear的 值为“2016”，它将包含书籍列表。

清单 13.24 BookController 类

package com.example.controller; import j ava.time.LocalDate; import j ava.util.Arrays; import j ava.util.List;

import org.springframework.stereotype.Controller; import org.springframework.web.bind.annotation.PathVariable; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.servlet.ModelAndView; import com.example.model.Book;

@Controller

public class BookController {

0RequestMapping(value = "/latest/{pubYear}M) public ModelAndView getLatestTitles(

@PathVariable String pubYear) {

ModelAndView mav = new ModelAndView ("Latest Titles11);

if (M2016n.equals(pubYear)) {

List<Book> list = Arrays.asList(

new Book(M0001nf "Spring MVC: A Tutorial",

"Paul Deck",

LocalDate.of(2016, 6, 1)), new Book(M0002n, MJava Tutorial’1,

"Budi Kurniawan11, LocalDate.of (2016, 11, 1) ) f new Book {M0003n, nSQL", "Will Biteman11,

LocalDate.of(2016, 12, 12))); mav.getModel().put("latest", list);

}

return mav;

测试BookController的一种简单方式是使用ModelAndViewAssert中的静态方法，如清 单 13.25 中的 BookControllerTest 类所示。

清单 13.25 BookControllerTest 类

package com.example.controller;

import static org.springframework.test.web.ModelAndViewAssert.\*; import j ava.time.LocalDate;

java.util.Arrays; java.util.Comparator; java.util.List; org.junit.Test;

org.springframework.web.servlet.ModelAndView; com.example.model.Book;

import

import

import

import

import

import

public class BookControllerTest {

0Test

public void test() {

BookController bookController = new BookController(); ModelAndView mav = bookController.getLatestTitles<n2016”）； assertViewName (mav, "Latest Titles1’）； assertModelAttributeAvailable(mav, "latest");

List<Book> expectedList = Arrays.asList(

new Book("◦002", LocalDate•of(2016, 11, 1)), new Book("OOOl", LocalDate.of(2016, 6, 1)), new Book("0003", LocalDate.of(2016, 12, 12))); assertAndReturnModelAttributeOfType (mav, ’'latest11, expectedList.getClass{));

Comparator<Book> pubDateComparator =

(a, b) -> a.getPubDate().compareTo《b.getPubDate()); assertSortAndCompareListModelAttribute(mav, "latest", expectedList, pubDateComparator);

**13.6**应用**Spring MVC Test**进行集成测试

集成测试用来测试不同的模块是否可以一起工作。它还确保两个模块之间数据的传递。 使用Spring框架依赖注入容器，必须检查bean依赖注入。

若没有合适的工具，集成测试可能需要很多时间。想象一下，如果你正在建立一个网上商店， 你必须使用网络浏览来测试购物车是否正确计算。每次更改代码时，你必须启动浏览器，登录系 统，将几个项目添加到购物车，并检查总数是否正确。每次迭代可以轻易地花费5分钟！

好在，Spring提供了一个用于集成测试的模块：Spring Test。

Spring 的 MockHttpServletRequest、MockHttpServletResponse 和 ModelAndViewAssert 类 适用于对Spring MVC控制器进行单元测试，但它们缺少与集成测试相关的功能。例如，它们 直接调用请求处理方法，无法测试请求映射和数据绑定。它们也不测试bean依赖注入，因为 SUT类使用new运算符实例化。

对于集成测试，您需要一组不同的Spring MVC测试类型。以下小节讨论集成测试的API 并提供一个示例。

13.6.1 API

作为Spring的一个模块，Spring MVC Test提供了一些实用程序类，可以方便地在Spring MVC 应用程序上执行集成测试。bean是使用Spring依赖注入器创建的，并从ApplicationContext中获 取，就像在一个真正的Spring应用程序中一样。

MockMvc 类位于 org.springframework.test.web.servlet 包下，是 Spring MVC Test 中的主类，

用于帮助集成测试。此类允许你使用预定义的请求映射来调用请求处理方法。这里是一种常 见的创建MockMvc实例的方法：

MockMvc mockMvc = MockMvcBuilders . webAppContextSetup( webAppContext) .build (); 这里，webAppContext 是 WebApplicationContext 实例的一个引用，WebApplicationContext 是第2章中讨论的ApplicationContext的子类，每个Spring开发人员都应该熟悉。要获取一个 WebApplicationContext，你必须在测试类中声明这一点。

@Autowired private WebApplicationContext webAppContext;

MockMvcBuilder读取一个Spring配置文件或为测试类定义的文件。我将讨论如何指定测 试类的配置文件，但首先我想讨论MockMvc。

MockMvc是一个非常简单的类。事实上，它只有一个方法：perform，用于通过URI间 接调用Spring MVC控制器。

perform方法具有以下签名：

ResultActions perform (RequestBuilder requestBuilder)

要测试请求处理方法，你需要创建一个RequestBuilder。好在，MockMvcRequestBuilders 类提供了与HTTP方法具有相同名称的静态方法：get、post、head、put、patch、delete等。要 使用HTTP GET方法测试控制器，你可以调用get方法，要使用POST测试，则调用post方法。

这些静态方法也很容易使用，你只需要传递一个字符串——控制器的请求处理方法的URL

例如，要调用名为getEmployee的请求处理方法，你将编写如下代码：

ResultActions resultActions = mockMvc .perform (get (u/ getEmployee’’））；

当然，你必须导入MockMvcRequestBuilders的静态get方法。

要验证测试是否成功，你需要调用ResultActions的andExpect方法。andExpect方法签名如下:

ResultAction andExpect (ResultMatcher matcher)

注意，andExpect返回ResultActions的另一个实例，这意味着可以链式调用多个AndExpect，

你可以在稍后的示例中看到这一点。

MockMvcResultMatchers 了静态方法来轻松创建 ResultMatcher。MockMvcResultMatclieis

属于 org.springframework.test.web.servlet.result 包。表 13.4 显示了它的一些方法。

表 13.4 MockMvcResultMatchers 的主要方法

|  |  |  |
| --- | --- | --- |
| 方法 | 返回类型 | 描述 |
| cookie | CookieResultMatchers | 返回一个ResultMatchers，用来断言cookie值 |
| header | HeaderResultMatchers | 返回一个ResultMatchers,用来断言HTTP响应头部 |
| model | ModelResultMatchers | 返回一个ResultMatchers，用来断言请求处理的模型 |
| status | StatusResultMatchers | 返回一个ResultMatchers,用来断言HTTP响应状态 |
| view | ViewResultMatchers | 返回一个ResultMatchers，用来断言请求处理的视图 |

例如，要确保控制器方法的请求映射正确，可以使用状态方法：

mockMvc.perform(get("/ getBook")).andExpect(status().isOk());

isOK方法断言响应状态代码是200,可以看到MockMvc及其相关类使得集成测试控制 器变的非常容易。

1. Spring MVC测试类的框架

了解了 Spring MVC Test中的一些重要的API,现在来看下Spring MVC测试类的框架。

import org.junit.After; import org.junit.Before; import org.junit.Test; import org.junit.runner.RunWith;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.test.context.ContextConfiguration;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import org.springframework.test.context.web.WebAppConfiguration;

import org.springframework.test,web.servlet.MockMvc;

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import org.springframework.web.context.WebApplicationContext;

@RunWith(SpringJUnit4ClassRunner.class)

0WebAppConfiguration SContextConf iguration (11. . . M)

public class ProductControllerTest {

SAutowired

private WebApplicationContext webAppContext; private MockMvc mockMvc;

0Before

public void setup() {

mockMvc = MockMvcBuilders.webAppContextSetup(webAppContext).build();

@After

public void cleanup(){

@Test

public void testl() throws Exception { **mockMvc.performandExpect(...);**

0Test

public void test2{) throws Exception { **mockMvc.performadnExpect(...);**

首先，看下你要导入的类型。在导入列表的顶部，是来自JUnit和Spring MVC Test的类 型。像单元测试类一样，SpringMVC测试类可以包括用@Before和@After注解的方法。两种 注解类型都是JUnit的一部分。

接下去，测试框架开始与单元测试有所不同。首先是测试类运行器。你需要一个 SpringJUnit4ClassRunner.class 在⑧尺而贾池注解内：

@RunWith(SpringJUnit4ClassRunner.class)

这个runner允许你使用spring。

然后，你需要添加如下注解类型：

@WebAppConfiguration @ContextConf iguration (11. , . M)

WebAppConfiguration注解类型用于声明为集成测试加载的ApplicationContext应该是

WebApplicationContext类型。ContextConfiguration注解类型告诉测试运行器如何加载和配置 WebApplicationContext 〇

除以上，测试类中还需要两个对象：

private WebApplicationContext webAppContext;

private MockMvc mockMvc;

13.6.3示例

以下示例展示如何对Spring MVC控制器开展集成测试。

清单13.26中的配置文件展示了将被扫描的包。这个文件是一个典型的Spring MVC配置 文件，但减去了任何资源映射和视图解析器。但是，你可以使用实际的配置文件。

清单 13.26 test-config.xm丨文件

<?xml version="l.0M encoding=MUTF-8"?>

<beans xmlns=Mhttp: //www. springframework.org/schema/beans11 xmlns:xsi=M<http://www.w3.org/2001/XMLSchema-instance>" xmlns:p=n<http://www.springframework.org/schema/pn>xmlns:mvc=M<http://www.springframework.org/schema/mvc>" xmlns : context=Mhttp: / /www. springf ramework. org/schema/context11 xsi:schemaLocation="

<http://www.springframework.org/schema/beans>

<http://www-springframework.org/schema/beans/spring-beans-xsd>

<http://www.springframework.org/schema/mvc>

<http://www.springframework.org/schema/mvc/spring-mvc.xsd>

<http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/spring-context.xsd>"> <context:component-scan base-package=McontrollerM/>

<context:component-scan base-package=nservice"/>

<mvc:annotation—driven/>

</beans>

此示例中的SUT是清单13.27中的EmployeeController类。类中只有一个请求处理方法， getHighestPaid，它映射到/highest-paid。

清单 13.27 EmployeeController 类

package controller;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Controller; import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestMapping; import service.EmployeeService; import domain.Employee;

@Controller

public class EmployeeController {

SAutowired

EmployeeService employeeService;

@RequestMapping(value=M/highest-paid/{category}M)

public String getHighestPaid(@PathVariable int category, Model model)

{

Employee employee = employeeService.getHighestPaidEmployee( category); model.addAttribute("employee", employee); return "success";

清单 13.28 展示了 EmployeeController 的测试类。 清单 13.28 EmployeeControllerTest 类 package com.example.controller;

import static org,springframework.test.web.servlet.request. ^3 MockMvcRequestBuilders.get;

import static org,springframework.test.web.servlet.result. zz MockMvcResultHandlers.print;

import static org.springframework.test.web.servlet.result. zz MockMvcResultMatchers.model;

import static org.springframework.test.web.servlet.result. n MockMvcResultMatchers.status;

org.junit.After;

**import**

**import**

**import**

**import**

**import**

**import**

**import**

**import**

**import**

**import**

**import**

org.junit.Before;

org.junit.Test;

org.junit.runner.RunWith;

org.springframework.beans.factory.annotation.Autowired;

org.springframework.test.context.ContextConfiguration;

org.springframework.test.context.j unit4.SpringJUnit4ClassRunner;

org.springframework.test,context.web.WebAppConfiguration;

org.springframework.test.web.servlet.MockMvc;

org.springframework.test.web.servlet.setup.MockMvcBuilders;

org.springframework.web.context.WebApplicationContext;

@RunWith(SpringJUnit4ClassRunner.class) @WebAppConfiguration

@ContextConfiguration("test-config.xml")

public class EmployeeControllerTest {

SAutowired

private WebApplicationContext webAppContext; private MockMvc mockMvc;

0Before

public void setup {) {

this.mockMvc = MockMvcBuilders.webAppContextSetup(webAppContext) .build();

@After

public void cleanup() {

0Test

public void testGetHighestPaidEmployee() throws Exception { mockMvc.perform(get("/highest-paid/2M))

.andExpect(status () .isOk())

.andExpect(model().attributeExists("employee"))

.andDo(print());

}

}

EmployeeControllerTest 类包含一个 setUp 方法，它创建一个 MockMvc 对象。 testGetHighestPaidEmployee方法执行测试，并期望响应状态代码为200，并且模型具有 employee 属性。

测试方法还调用andDo(print〇)在响应对象中打印各种值。如果你的测试成功通过，你应 该会看到类似的结果。

MockHttpServletRequest:

HTTP Method = GET Request UR工=/highest-paid/2 Parameters = {}

Headers = {}

Handler:

Type = controller.EmployeeController Method = public java-lang.String

controller.EmployeeController.getHighestPaid(int,org.springframe- work.ui.Model)

Async:

Async started = false

Async result = null

Resolved Exception:

Type = null

ModelAndView:

View name = View = Attribute = value = errors =

success

null

employee Xiao Ming []

**($**200000**)**

FlashMap:

Attributes = null

MockHttpServletResponse:

Status = 200 Error message = null Headers = {}

Content type = null Body =

Forwarded URL = success Redirected URL = null Cookies =[]

13.7修改集成测试中Web根路径

默认情况下，使用@WebAppConfiguration注解的Spring集成测试类将使用相对于项目目 录的/src/main/webapp目录作为根目录。这是一个Maven标准布局。如果你不使用STS，而是 使用不依赖于Maven的IDE (如Eclipse或NetBeans )，在某些罕见情况下，这可能会出现问 题。例如，如果你需要使用ServletContext.getRealPath〇的值，你将获得相对于/src/main/webapp 的值，而不是相对于你的Web应用程序目录的值。

然而，你也不能使用/src/main/webapp/WEB-INF/classes目录作为Eclipse中的输出目录， 因为这意味着将其嵌套在源目录中，而在Eclipse是禁止这样做的。好在，可以将实际应用程序 目录传递到@WebAppConfiguration来修复此问题。例如，以下将让任何WebApplicationContext 使用/ approotdir作为应用程序目录。

0WebAppConfiguration("/approotdir")

清单13.29展示了一个WebAppController类，其有一个方法返回在线资源的真实路径。

清单 13.29 WebAppController 类

package controller;

import j avax.servlet.ServletContext;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Controller; import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.RequestMapping;

©Controller

public class WebAppController {

// ServletContext cannot be used as a method parameter, inject // instead.

@Autowired

private ServletContext ServletContext;

0RequestMapping(value=M/getWebAppDirM)

public String getWebAppDirectory(Model model) {

model.addAttribute("webAppDirM, ServletContext.getRealPath(M/n))； return "success";

}

}

清单13.30展示了一个用于集成测试WebAppController类的测试类。注意传递给 @WebAppConfiguration 注解的值。

清单 13.30 WebAppControllerTest 类

package com.example.controller;

import static org.springframework.test.web.servlet.request. ^3 MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.result. ] MockMvcResultHandlers.print;

import static org.springframework.test.web.servlet.result. za MockMvcResultMatchers.model;

import static org.springframework.test.web.servlet.result. MockMvcResultMatchers.status;

org.junit.After;

**import**

**import**

**import**

**import**

**import**

**import**

**import**

**import**

**import**

org.junit.Before;

org.junit.Test;

org.junit.runner.RunWith;

org.springframework.beans.factory.annotation.Autowired;

org.springframework.test.context.ContextConfiguration;

org.springframework.test.context.j unit4.SpringJUnit4ClassRunner;

org.springframework.test.context.web.WebAppConfiguration;

org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.setup.MockMvcBuilders; import org .springframework.web.context.WebApplicationGontext;

@RunWith(SpringJUnit4ClassRunner.class)

@WebAppConf iguration (n /webapp11)

SContextConfiguration("test-config.xml") public class WebAppControllerTest {

0Autowired

private WebApplicationContext webAppContext; private MockMvc mockMvc;

0Before

public void setup() {

this.mockMvc =

MockMvcBuilders.webAppContextSetup(webAppContext).build();

@After

public void cleanup{) { @Test

public void testWebAppDir() throws Exception { mockMvc.perform{get(”/getWebAppDir”））

.andExpect(status().isOk())

.andExpect (model () . attributeExists (MwebAppDirT,)) .andDo(print());

13.8小结

测试是软件开发中的一个重要步骤，你应该在开发周期中尽早地执行单元测试和集成测 试这两种类型的测试。单元测试用于类的功能性验证。在单元测试中，所涉及依赖通常被测 试挡板替换，其可以包括dummy、stub、spy、fake和mock对象。JUnit是一个流行的用于单 元测试的框架，并且通常与mocking框架（如Mockito或EasyMock)结合使用。

集成测试用于确保同一应用程序中的不同模块可以一起工作，同时确保请求映射和数据 绑定也可以工作。Spring Test MVC是一个Spring模块，它提供了一组API，可以轻松地对Spring 应用程序执行集成测试。在本章中，你还学习了如何使用SpringTestMVC。

附录A

Tomcat

Tomcat是当今最流行的Servlet/JSP容器。它是免费、成熟、开源的。为了运行本书附带 的范例应用程序，需要Tomcat 7或其更髙版本，或者其他兼容的Servlet/JSP容器才行。附录 A将介绍如何快速安装和配置Tomcat。

1. 1下载和配置Tomcat

首先，从http://tomcat.apache.org网站下载Tomcat的最新版本。选用zip或gz格式的最 新二进制发行版本。Tomcat 8需要用JRE 7或JDK7来运行。早期版本的Tomcat要求是用JDK 来运行，因为需要一个java编译器来编译JSP页面。从5.5开始，Tomcat捆绑了 Eclipse的java 编译器，因此不再需要JDK 了。

下载了 zip或者gz文件后，要进行解压，随后就能在安装目录下看到几个目录。

在bin目录中，可以看到启动和终止Tomcat的程序。webapps目录很重要，因为可以在 那里部署应用程序。此外，conf目录中还包含了配置文件，包括server.xml和tomcat-users.xml 文件。lib目录也值得关注，因为其中包含了编译Servlet和定制标签所需的Servlet和JSP API。

解压完zip或者gz文件后，应设置JRE\_HOME为JRE安装目录（或设置JAVA\_HOME 环境变量为JDK安装目录)。设置方式很简单，在bin目录中创建一个setenv.sh (Linux或Mac OSX环境）或setenv.bat (Windows环境)。该文件包含一行代码，将会在Tomcat启动时执行。

如下是一个serenv.sh文件示例：

JRE\_HOME=/opt/j ava/j dkl.7.0\_7 9

setenv.bat文件如下：

set JAVA\_HOME=c:\Program Files\Java\jdkl.7.◦\_〇6

对于Windows用户，还可以下载对应的Windows安装版本，安装起来会容易一些。

1. 2启动和终止Tomcat

下载并解压好Tomcat二进制版本文件后，就可以运行startup.bat文件（Windows)或 startup.sh文件（UNIX/Linux/Mac 0S)来启动Tomcat。这两个文件都放在Tomcat安装目录的 bin目录下。默认情况下，Tomcat在端口 8080运行，因此可以在浏览器中打开以下网址：

<http://localhost:8080>

终止Tomcat时，运行bin目录下的shutdown.bat文件（Windows)或者shutdown.sh文件 (UNIX/Linux/Mac OS )。

1. 3定义上下文

要将Servlet/JSP应用程序部署到Tomcat，需要显式或隐式定义一个Tomcat上下文。在 Tomcat中，每一个Tomcat上下文都表示一个Web应用程序。

显式定义Tomcat上下文有几种方法，包括：

•在 Tomcat 的 conf/Catalina/localhost 目录下创建一个 XML 文件。

•在 Tomcat 的 conf/server.xml 文件中添加一个 Context 元素。

如果决定给每一个上下文都创建一个XML文件，那么这个文件名就很重要，因为上下 文路径是从文件名衍生得到的。例如，把一个commerce.xml文件放在conf/Catalina/localhost 目录下，那么应用程序的上下文路径就是commerce，并且可以利用以下URL调用一个资源：

<http://localhost:8080/commerce/resourceName>

上下文文件中必须包含一个Context元素，作为它的根元素。这个元素大多没有子元素， 它是该文件中唯一的元素。例如，下面就是一个示例上下文文件，其中只有一行代码：

〈Context docBase=MC:/apps/commercen reloadable=MtrueM/>

这里唯一必要的属性是docBase，它用来定义应用程序的位置。reloadable属性是可选的， 但是如果存在，并且它的值设为true,那么一旦应用程序中Java类文件或者其他资源有任何 増加、减少或者更新，Tomcat都会侦测到，并且一旦侦测到这类变化，Tomcat就会重新加载

附录 A Tomcat

应用程序。在部署期间，建议将reloadable值设为True，在生产期间，则不建议这么做。

当把上下文文件添加到指定目录时，Tomcat就会自动加载应用程序。当删除这个文件时， Tomcat就会自动卸载应用程序。

定义上下文的另一种方法是在confserver.xml文件中添加一个Context元素。为此，要先 打开文件，并在Host元素下创建一个Context元素。与前一种方法不同的是，此处定义上下 文需要给上下文路径定义path属性。下面举一个例子：

<Host nameyiocalhost" appBase=,,webappsM unpackWARs=Mtrue" aut〇Deploy=’’true">

<Context path=n/commerce,!

docBase=,TC: /apps/coInmerce,, reloadable="true"

/>

</Host>

一般来说，不建议通过serverxinl来管理上下文，因为只有重启Tomcat后，更新才能生 效。不过，如果有很多应用程序需要测试，你也许会觉得使用servaxml比较理想，因为可以 在一个文件中同时管理所有的应用程序。

最后，通过将一个war文件或者整个应用程序复制到Tomcat的webapps目录下，还可以 隐式地部署应用程序。

关于Tomcat上下文的更多信息，请通过以下网址查阅：

<http://tomcat.apache.org/tomcat-8.0-doc/config/context.html>

1. 4定义资源

定义一个JNDI资源，应用程序便可以在Tomcat上下文定义中使用。资源用Context元 素目录下的Resource元素表示。

例如，为了添加一个打开MySQL数据库连接的DataSource资源，首先要添加下面这个 Resource 元素：

〈Context [path= n /appNa/rjeM ] docBase=” . . •

〈Resource na.me=11 jdbc/dataSourceName11 auth="Container" type=M javax. sql. DataSource\*'

username=" " password="..."

driverClassName=,,com.mysql. jdbc. Driver" url=M..

/>

</Context>

关于Resource元素的更多信息，请到以下网址查阅：

<http://tomcat.apache.org/tomcat-8.0-doc/j> ndi-resources-howto.html

1. 5安装TLS证书

Tomcat支持TLS，并且用它确保机密数据的传输，如身份证号码和信用卡信息等。利用 KeyTool程序生成一个public/private键对，同时选择一家可信任的授权机构，来创建和签发 数字证书。

一旦收到证书，并将它导入到keystore后，下一步就是在服务器上安装证书了。如果使 用的是Tomcat，将keystore复制到复制服务器上的某个位置，并对Tomcat进行配置即可。随 后，打开conf/server.xml文件，并在<service>下添加以下Connector元素。

<Connector port=,,443M minSpareThreads=M5" maxSpareThreads=,,75,, enableLookupsytrue" disableUploadTimeout=Mtrue,, acceptCount=,,100" maxThreads=n 2 0 0 n

scheme=,,https,1 s e cure="true"

SSLEnabled=n true" keystoreFile=" /path/to/keys tore11 keyAlias=T, example. comT, keystorePass="01secret02%%%" clientAuth=" false'1 sslProtocol="TLS"

/>

以上粗体字部分的代码与SSL有关。请确保未使用SSLEnabled属性，或确保将其设置 为false,以防止旧浏览器使用不再安全的SSL。

附录B

Spring Tool Suite 和 Maven

Spring提供了自己的集成开发环境（IDE)，称为Spring Tool Suite (STS)，它可能是构建 Spring应用程序的最佳IDE 了。STS捆绑Maven作为其默认依赖项管理工具，因此你不需要 单独安装Maven。

本附录提供了有关如何使用STS和Maven的简短教程。

1. 1安装STS

Spring Tool Suite是一个基于Eclipse的集成开发环境（IDE),它非常优秀，可能是使用 Spring框架的最佳IDE。它同时也是一个一站式工具箱，包括开发Spring MVC应用程序所需 的库和应用程序，例如，它附带Maven和Pivotal tc Server的开发版本（一个Tomcat的定制 版)。如果你还没有决定使用哪个IDE,我建议你试试STS。

要开始使用STS，请首先从此网页下载：

<https://spring.io/tools/sts>

STS采用zip文件分发。下载zip文件后，将其解压到你选择的目录。zip文件包含一个 sts-bundle目录，其下又有3个目录：

1. 法律文件，包含各种工具的许可协议。
2. pivotal-te-server-developer-x.y.z.RELEASE。包含 Pivotal tc 月艮务器的开发人员版本 x.y.z。
3. sts-x.y.z.RELEASE。包含STS，其中x.y.z代表为STS的主版本和次要版本。

Windows版本的启动程序为STS.exe文件，Linux版本的启动程序为STS文件。双击exe 文件或从命令行运行STS文件以启动STS。第一次运行STS，系统将提示你选择工作区。单 击“确定”，你将看到STS主页面，如图B.1所示。

|  |  |  |
| --- | --- | --- |
| Spring - Dashboard - Spring Toof Suite — □ X File Edit Navigate Search Project Run Window Help  \* %v - 0 C1 - --  | Quick Access | :哲 | [ \* Spring | | |
| B Pac... ^ ° B | (J) Dashboard ^ | 1=3 B | Outline 2Z a B |
| B笃丨V y | Sprinq Dashboard ； Search spring.ID q Subscribe | ▽  An outline is not available. |
|  |  |  |
|  | Create Feeds  l^i Java Project i'i% Sorina Leaacv Project ^ This Week in Sprrnq - Februa  ^ Welcome to another inttallrr ^ ^ in Spring! It's been a busy we  〆 the Spring team, as you're at Thanks for instating STS 3.7.2 finished my presentation at t  Pivotal has released an update to Spring Tool Suite v show and now... (Feb 16, 20^ J 1 O SprinaAMOP 1.6.0 Mi...^ncT |  |
| ▼ Help and Documentation  We are plesseci to 3nnounce  Communitv Suoport Forums New and Moteworthv the first milestone of the 1.6  AMQP. Some highlights of tl  Issue and Buq Tracker Extensions jar spnng-rabbrt-test contain SorinaSource Commercial Suacort Product Paae tG help with …【Feb 16, 2016 1 | Spring ExpL. S3 = □  e 驾1 v\*Uaz ▽ |
| Dashboard Extensions; |
|  | < > |  |
| El 你 〇 p \_ E  No servers are | Q Console £2 二 barkers . Progress 一，’ 0 ▼巴▼ 1=11 □ No consoles to display at this time. |  |
|  |
|  | | |

图B.l STS欢迎界面

尽管已更改过图标，这个主窗口可能还是会让你想起Eclipse。现在你可以随时创建自己 的Spring MVC应用程序了。

1. 2创建一个SpringMVC应用

STS严重依赖于Maven，并允许你创建使用Maven管理依赖关系的应用程序。要仓(J建 Spring MVC应用程序，请按照下列步骤操作。

1. 点击Hie > New > MavenProject，你会看到如图B.2所示的项目对话框。
2. 勾选 “Create a simple project ( skip archetype selection)”〇
3. 点击Next按钮继续配置项目，将出现如图B.3所示的配置项。
4. 在“Group Id”输入框中输入包名，例如“com.example”；在“Artifact Id”输入框中 输入项目名，例如“firstSpringMVC”。

|  |  |
| --- | --- |
| 1 \_ New Maven Project | □ X |
| New Maven project | 偷 |
| Select project name and bestion |
| 0 Create a simple project (skip archetype selection) |  |
| 0 Use default Workspace location |  |
| Location； | Brows竺… |
| 1 1 Add project(s) to working set | |
| Workinq \_：ct: 1 | More.., |

► Advanced

(2) Next > Finish CUrtceT

**图**B.2**新**Maven**项目对话框**

n **New Mavtn Project**

New Maven project

Configure project

□

**M|**

Aitifjct **Group Id;**

**Artifact Id: firstSpnngMVC Version: I 0.0.1-SNAPSHOT**

**com.example**

**Packaging: war**

**Name:**

Description:

**Parent Project Group Id:**

**Artifact Id**： **Version:**



Advanced

Back

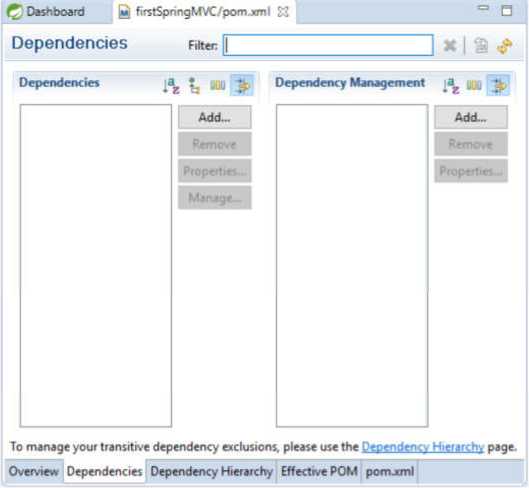
**Nort**

Finish Cancel

**图**B.3**配置项目**

1. 在“Packaging”下拉框中选择“war”，告诉STS你将创建一个Spring MVC应用。 war是servlet/JSP应用的文件扩展名。
2. 点击Finish按钮，你将在Package Explorer中看到你的项目（见图B.4)。
3. STS将会创建一个目录结构如图B.4所示的工程和一个pom.xml文件（maven配置文 件。现在，你需要编辑p〇m.xml，以便让maven来处理依赖。双击pom.xml文件，用默认编 辑器打开文件。编辑器有多个选项卡，提供查看和编辑文件的不同方式。选项卡名称显示在 对话框的底部。其中一个选项卡是“依赖项”选项卡，允许你管理项目所需的依赖项。单击 依赖项选项卡，你将看到如图B.5所示的对话框。

v ^ firstSprrngMVC src/main/javi i3 src/main/resources src/test/jjva i3 sfc/test/resourc«



图B.5通过依赖选项卡添加依赖

> A System Library [J2SE-1.5}

v- ^ src

* & main ^ test

v target

* m2e^Hrtp Ipl pomjtml

图B.4新创建的目录结构

1. 对于初学者，通过Dependencies选项卡添加 依赖是一种相对容易的方式，我将在这里展示如何 做。典型的Spring MVC应用程序至少需要3个库： Servlet API、Spring MVC 库和 JSTL。要添加 Servlet API，请单击“Dependecies”窗格中的Add按钮（而 不是“Denpendenly Management”窗格上的添加按 钮），将扭f “ Select Dependency” 对话(6 (见图 B.6 )。

|  |  |
| --- | --- |
| r.\_. Select Dependency | □ X |
| Group Id: \*| javax.servlet |  |
| Artifact Id: \* javax.servlet-api |  |
| Version: | 5,1,0 | Scope: iprovided i v | |

Enter group Id, artifactld or sha1 prefix ar pattern [\*)\

■ Index downloads are di&abled, search results may be incomplete.

Search Results:

要输入的值如下。

**Group Id: javax.servlet Artifact Id: javax.servlet-api Version: 3.1.0 Scope: provided**

® QIC ■ CanceE

图 **B.6** 添加 **Servlet API**

provided表示当应用程序分发时，该库将由容器提供，不需要包含在war文件中。

完成输入后，单击“OK”按钮，关闭该对话框，你将返回到“Dependencies”选项卡。 9.接下去，用同样的方式添加SpringMVC库，点击Add按钮，输入如下值：

Group Id: org.springframework Artifact Id: spring-webmvc Version: 4.2.4.RELEASE Scope: compile

10.最后，添加JSTL库，点击Add按钮，输入如下值:

Group Id: javax.servlet

Artifact Id: jstl Version: 1.2 Scope: runtime

firstSpringMVC src/main/java ^ src/main/resources i3 src/test/java C3 src/test/resources

JRE System Library [J25E-1.5] v E\*, Maven Dependencies

〇'〇 javax.servlet-api-3.1.0.jar C:\Users\j. 〇ia spring-webmvc-4.2.4.RELEASE.jar 1 ion\* spring-beans-4.2.4.RELEASE.jar - C:\L 〇'〇 spring-context-4.2.4.RELEASE.jar - C： 5pring-aop-4.2ARELEASE.jar C:\Us« > (p aopalliance-I.O.jar - J U:e>： .jayden'1 i〇ia spring-core-4.2.4,RELEASE.jar C: .Us 〇ia commons-logging-1.2.jar - C: ,User: spring-expression-4.2.4.RELEASE.jar - mm spring-web-4.2.4.RELEASE.jar - C:\Uc .〇,〇 jst卜 1.2.jar-'二：丄,

* ^ src
* ^ target @ pom.xml

图**B.7 Maven**依赖项

11.按 Ctrl + S 或点击 File > Save 保存 pom.xml。这样， STS将尝试通过下载依赖项来构建项目。你需要连接到互联 网，除非你以前使用Maven下载过所需的库。

Maven完成下载依赖项后，你将在Package Explorer中看 到一个 Maven Dependencies 文件夹（见图 B.7 )。

pom.xml文件仍然有错误，因为它找不到web.xml。要 纠正此问题，请双击Package Explorer中的pom.xml文件以 重新打开编辑器，然后单击pom.xml选项卡。在pom.xml文 件中查找<dependencies>元素，并在其上添加以下内容。注意， failOnMissingWebXml元素用来打开或关闭由缺少web.xml

文件引起的错误消息。

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactld>maven-war-plugin</artifactld>

<configuration>

<failOnMissingWebXml>false</failOnMissingWebXml> 〈/configuration〉

</plugin>

</plugins>

</build>

1. 3选择Java版本

您可以通过向pom.xml文件添加属性来为应用程序选择Java版本。请按照下列步骤操作。

1. 点击Overview选项卡，如图B.8所示
2. 点击属性下拉列表右侧的“Creat”按钮。将打开“Add Property”对话框，如图B.9

所示。

**h** \*sprin9-mvc-5howcase/po

C Dashboard

ervicw

tlfact

oup id: com.exampte

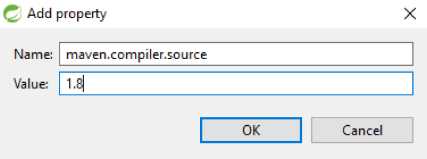
Ufact Id: • HrstSpringMVC

rsion： 1 .(XO\*6UILt>SNAPSHOT

ckaging: war v

**\_>rop«rtt«s**

java-version: 1.7 create



图B.9选择Java版本

oraspringframework-vefsion: 4.2.2.RELE/

Remove

org.springframework.security-version: 4.C

New module dement

**^odutcs**

Overview Depervdencies Dependency Hierarchy “

图B.8 Overview选项卡

3•在 “Name” 输入框中输入 “maven.compiler.source”，在 “Value” 输入框中输入 “1.8”。 4.保存pom.xml文件。

如果STS报错，那是因为它找不到JRE 1.8,你可以通过执行如下步骤告诉它在哪里找 到 Java。

1 •点击 Window > Preferences。

1. 在左窗格中选择Java> Installed JRE，然后单击Add按钮。
2. 选择Standard VM，然后点击Next。你将看到如图B.10所示的对话框。
3. 点击“Directory”按钮并浏览到JRE主目录。
4. 点击Finish按钮。您将在己安装的JRE列表中看到JDK (见图B.11)。

|  |  |
| --- | --- |
| > C:\Program Files\Java\jdkl.8.0„66\jre\lib\resource ^ | Add JARs.» |
| CAProgram File5\Java\jdkt.8.0\_66\)re\lib\rt.jar |  |
| > O\Pf〇gnmFil«\Jtvj\jdM4,0\_WVi^MibVj$5ej«r | JjviSKi&C \_ |
| .〇\Pf〇gram Fil«\Java\jdk1.8.0\_66Vjre\lib\jctjar | Sount1 |
| ■ C:\Program File5\Java\jdkl.8.0^66Vjre\libVchar5et3. |  |
| ■ CAProgram File5\Java\jdk1.8.0-56\jre\lib\jfrjar | Eternal -nnolatiom… |
| i-« C;\Pf〇gra m FilesVIava\jdk 1 ^.0\_66Vjre\ lib\ ott\acc e |  |
| C:\PfDgram Files\Java\jdlcl.8.0\_&&\jre\lib\«t\cldrd | Rc〇)iM |
| »• C:\Program Files\Java\jdk1.8.0a.&6\jre\lib\€xt\dnsn |  |
| C:\Program Files\Java\jdkl.8.0..66\jre\libVexl\;accc |  |
| ^5 C;\Pf〇gram Files\Java\jdk1.8.0\_66\jre\lib\ext\jfxrt.j |  |
| CAProgram Files\Javs\jdkl.8.0\_&6\jre\lib\€xt\local v |  |
| < > | Restore Default |

JRI system libfiritc

fin-iiti I Csn<cd

C AddJRE

**D X**

JRE Definition

Specify attributes for a JRE

home C:\Pragnm Frles\Java\jdk1.8.0,66 I [ Di，ect敗**,••]**

name [ jdkl ^0^.66 |

Default YM arguments Variables.^



\* **B«ck**

**图**B**.10浏览**JRE**目录**

'\_ Preferences

n x

| type filter text

AspectJ Compile! ^

* Cloud Foundry Code Recommen Data Managemer
* Help
* Install/Update v Java

Appearance Build Path

* Code Style
* Compiler
* Debug
* Editor Installed JREs JUnit

Properties File v

⑦

Installed JREs

Add, remove or edit JRE definitions. By default, the checked JRE is added to the build path of newly created Java projects.

|  |  |  |
| --- | --- | --- |
| Name Location | Type | Add... |
| !0 m jdk1\_ C:\Proqram FH«\Java\i— | Standard — j | Edit,.. |
|  | | Duplicate...  Remove  Search... |

Installed JREs:

V

OK

Cancel

A

图B.ll己安装的JRE

1. 点击OK按钮。
2. 4 创建 index.html 文件

要完成示例应用程序，你需要仓1j建一个index.html文件并将其保存在src/main/webapp下。

清单B.1显示了一个简单的HTML页面。

清单**B.1 index.htm**丨文件

<!D〇CTYPE html>

<html>

<head>

<title>First Spring MVC app</title></head> <body>

Welcome

</body>

</html>

1. 5更新项目

在运行应用程序之前，你还要更新项目。右键单击Package Explorer中的项目图标，然后 单击 Maven > Update Project。

之后，Package Explorer中的项目将如图B.12所示。

v» Z3 firstSpringMVC 1^5 src/main/java L? src/main/resources i® src/test/java

src/test/resources

* 献 Maven Dependencies

1^, JRE System Library [JavaSE-1.8J v ^ src v 色 main

v webapp

匕:」index.html

^ test

* target @ pom.xml Servers

**图**B**.12更新后的项目**

1. 6运行应用

你需要一个Tomcat或其他的servlet容器来运行应用程序。如果你还没有安装Tomcat, 请现在安装。安装方式请参见附录A。

请按照下列步骤运行应用：

1.右键单击项目并选择Run As > Run on Server。你将看到Run On Server对话框，如 图B.13所示。

Run On Server

$\*lKt which stfwer to trst

How do you want to select the itiverl 〇 Choose an easting sefver ($^anu«ily dd^ine « new server

Select the server type

^3 jiddiiignj'l -r：r .fi -rdjprtft R^fr^lh

[ type fitter text

Tomtit v?^3

^ Tomc«t vS.O Server \_

Publishes #nd run.s J25E #nd EE W«b projects #nd server configutili^ns to s Tomcat seivev.

^ localhost |

Sever s host name. **Stfver nar^c**

| Tomcat vfi.0 Stfvtf at local host (2)

Scfver iunlime envuonmenf Apache Tomcat v8.0 n Always use this server whfffi running thisfiroject

**Cortfigute funtirrtg tfivarorvitieflts.-**



Next >



Finish

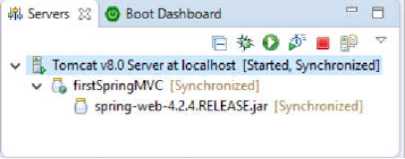
**Cancel**

图 B.13 选择 Tomcat

1. 选择服务器类型，例如Apache下的Tomcat v8.0。如果没有看到Tomcat实例，请单 击“Add”链接（图B.13中圈出），然后浏览到Tomcat目录。
2. 在“Server Host Name”输入项中输入主机名。
3. 单击Finish按钮。

STS将启动Tomcat并运行你的应用程序。

要管理Tomcat及其上部署的所有应用程序，请打开STS中的服务器视图（见图B.14)。



图B.14管理Tomcat实例和应用的服务器视图

图B.15显示了 STS中的Web浏览器，其中显示了应用的默认页面。

|  |  |  |
| --- | --- | --- |
| y tndexhtml | 籲 RrstSpnn...泛 5 | =□ |
| m ^ | Http://IocalK〇st8080/firstSpringMVC/ | ^ B |
| Welcome | | |

图**B.15**你的第一个**Web**应用 现在，你可以按照每章中的教程来添加动态内容了。

附录c

Servlet

Servlet是开发Servlet的主要技术。掌握Servlet API是成为一名强大的Java Web开发者 的基本条件。你必须熟悉Servlet API中定义的70多种类型。“70”这个数字听起来很多，但 是如果一次只学一种类型，其实并不困难。

本附录介绍了 ServletAPI，并教你如何编写Servlet。所有示例代码都可以在servletapil、 servletapi2 和 servletapi3 中找到。

1. l Servlet API 概览

Servlet API 有以下 4 个 Java 包：

* javax.servlet，其中包含定义Servlet和Servlet容器之间契约的类和接口。
* javax.servlet.http，其中包含定义HTTP Servlet和Servlet容器之间契约的类和接口。
* javax.servlet.annotation,其中包含用于 Servlet、filter、listener 的注解。它还为被注解

元件定义元数据。

* javax.servletdescriptor，其中包含提供程序化登录Web应用程序配置信息的类型。

本附录主要关注javax.servlet和javax.servlet.http的成员。

图C.1中展示了 javax.servlet中的主要类型。

Servlet技术的核心是Servlet，它是所有Servlet类必须直接或间接实现的一个接口。在编 写实现Servlet的Servlet类时，直接实现它。在扩展实现这个接口的类时，间接实现它。

Servlet接口定义了 Servlet与Servlet容器之间的契约。这个契约归结起来就是，Servlet 容器将Servlet类载入内存，并在Servlet实例上调用具体的方法。在一个应用程序中，每种 Servlet类型只能有一个实例。



用户请求致使Servlet容器调用Servlet的Service方法，并传入一个ServletRequest实例 和一个ServletResponse实例。ServletRequest中封装了当前的HTTP请求，因此，Servlet开发 人员不必解析和操作原始的HTTP数据。ServletResponse表示当前用户的HTTP响应，使得

将响应发回给用户变得十分容易。

对于每一个应用程序，Servlet容器还会创建一个ServletContext实例。这个对象中封装了 上下文（应用程序）的环境详情。每个上下文只有一个ServletContext。每个Servlet实例也都 有一个封装Servlet配置的ServletConfig〇

下面来看Servlet接口。上面提到的其他接口，将在本章的其他小节中讲解。

1. 2 Servlet

Servlet接口中定义了以下5个方法：

void init(ServletConfig con fig) throws ServletException

void service(ServletRequest request, ServletResponse response) throws ServletException, java•i〇•IOException

void destroy()

j ava.lang•String getServletInf〇（）

ServletConfig getServletConfig()

注意，编写Java方法签名的惯例是，对于和包含该方法的类型不处于同一个包中的类型， 要使用全类名。正因为如此，在Service方法javax.servlet.ServletException的签名中（与Servlet 位于同一个包中），是没有包信息的，而java.io.Exception则是要写成完整名称的。

init、service和destroy是生命周期方法。Servlet容器根据以下规则调用这3个方法：

* init，当请求Servlet时，Servlet容器会第一时间调用这个方法。这个方法在后续请求 中不会再被调用。利用这个方法调用初始化代码。调用这个方法时，Servlet容器会传 入一个ServletConfig。一般来说，你会将ServletConfig赋给一个类级变量，因此这个 对象可以通过Servlet类的其他位置来使用。
* service，每当请求Servlet时，Servlet容器就会调用这个方法。编写代码时，假设Servlet 要在这里被请求。第一次请求Servlet时，Servlet容器调用init方法和Service方法。 后续的请求将只调用service方法。
* destroy，当要销毁Servlet时，Servlet容器就会调用这个方法。当要卸载应用程序， 或者当要关闭Servlet容器时，就会发生这种情况。一般会在这个方法中编写清除 代码。

Servlet中的另外两个方法是非生命周期方法，即getServletlnfo和getServletConfig。

* getServletlnfo,这个方法会返回Servlet的描述。你可以返回有用或为null的任意字符串。
* getServletConfig，这个方法会返回由Servlet容器传给init方法的ServletConfig。彳堤， 为了让getServletConfig返回一个非皿11值，必须将传给init方法的ServletConfig赋 给一个类级变量。ServletConfig将在C.6小节中讲解。

注意线程安全性。Servlet实例会被一个应用程序中的所有用户共享，因此不建议使用类 级变量，除非它们是只读的，或者是java.util.concurrentatomic包的成员。

C.3将介绍如何编写Servlet实现。

1. 3编写基础的Servlet应用程序

其实，编写Servlet应用程序出奇简单。只需要创建一个目录结构，并把Servlet类放在某 个目录下。本节将教你如何编写一个名为servletapil的Servlet应用程序。最初，它会包含一个 Servlet，即MyServlet,其效果是向用户发出一条问候。

要运行Servlets，还需要一个Servlet容器。Tomcat是一个开源的Servlet容器，它是免费 的，并且可以在任何能跑Java的平台上运行。如果你到现在都还没有安装Tomcat，应该去看 看附录A，并安装一个。

1. 3.1编写和编译Servlet类

确定你的机器上有了 Servlet容器后，下一步就要编写和编译一个Servlet类。本例中的 Servlet类是MyServlet，如清单C.1所示。按照惯例，Servlet类的名称要以Servlet作为后缀。

清单 C.1 MyServlet 类

package servletapil;

import java.io.I〇Excepti〇n;

import j ava.io.PrintWriter;

import javax.servlet.Servlet;

import j avax.servlet.ServletConfig;

import j avax,servlet,ServletException;

import j avax.servlet.ServletRequest;

import j avax.servlet.ServletResponse;

import j avax.servlet.annotation.WebServlet;

@WebServlet (name = ’’MyServlet”，urlPatterns = { "/myn }) public class MyServlet implements Servlet {

private transient ServletConfig servletConfig;

0〇verride

public void init(ServletConfig servletConfig) throws ServletException { this.servletConfig = servletConfig;

@〇verride

public ServletConfig getServletConfig() {

return servletConfig;

@〇verride

public String getServletlnfo() {

return "My Servlet";

@〇verride

public void service(ServletRequest request,

ServletResponse response) throws ServletException, IOException {

String servletName = servletConfig.getServletName(); response.setContentType(Mtext/htmln);

PrintWriter writer = response.getWriter(); writer.print("<!DOCTYPE html>M

+ M<html>M

+ M<body>Hello from " + servletName + M</body>M + M</html>M);

}

0〇verride

public void destroy() {

}

}

看到清单c.i中的代码时，你可能首先注意到的是下面这个注解：

@WebServlet (name = ’’MyServlet”，urlPatterns = { "/my" })

WebServlet注解类型用来声明一个Servlet。命名Servlet时，还可以暗示容器，是哪个 URL调用这个Servlet。name属性是可选的，如有该属性，通常用Servlet类的名称。重要的是 urlPatterns属性，它也是可选的，但是一般都是有的。在MyServlet中，urlPatterns告诉容器， /my样式表示应该调用Servlet。

注意，URL样式必须用一个正斜杆开头。

Servlet的init方法只被调用一次，并将private transient变量ServletConfig设为传给该方 法的 ServletConfig 对象。

private transient ServletConfig ServletConfig;

©Override

public void init(ServletConfig ServletConfig) throws ServletException { this.ServletConfig = ServletConfig;

}

如果想通过Servlet内部使用ServletConfig，只需要将被传入的ServletConfig赋给一个类变量。

Service方法发送字符串“Hello from MyServlet”给浏览器。对于每一个针对Servlet进来 的HTTP请求，都会调用Service方法。

为了编译Servlet,必须将Servlet API中的所有类型都放在你的类路径下。Tomcat中带有 servlet-api.jar文件，其中包含了 javax.servlet的成员以及javax.servlet.http包。这个压缩文件 放在Tomcat安装目录下的lib目录中。

1. 3.2应用程序目录结构

Servlet应用程序必须在某一个目录结构下部署。图C.2展示了 Web的应用程序目录。

这个目录结构最上面的那个Web目录，就是应用程序目录。在应 用程序目录下，是WEB-INF目录。它有两个子目录：

* classes。Servlet类及其他Java类必须放在这里面。类以下的 目录反映了类包的结构。

^3 web

* Q META-INF
* WEB-INF **^ ^3 classes**

**T** ^ **servletapH**

® MyServlet.ctass

图C.2应用程序目录

* lib。Servlet应用程序所需的jar文件要在这里部署。但Servlet API的jar文件不需要 在这里部署，因为Servlet容器已经有它的备份。在这个应用程序中，lib目录是空的。 空的lib目录可以删除。

Servlet/JSP应用程序一般都有JSP页面、HTML文件、图片文件以及其他资料。这些应 该放在应用程序目录下，并且经常放在子目录下。例如，所有的图片文件可以放在一个image 目录下，所有的JSP页面可以放在jsp目录下，等等。

放在应用程序目录下的任何资源，用户只要输入资源URL，都可以直接访问到。如果想让 某一个资源可以被Servlet访问，但不可以被用户访问，那么就要把它放在WEB-INF目录下。

现在，准备将应用程序部署到Tomcat。使用Tomcat时，一种部署方法是将应用程序目 录复制到Tomcat安装目录下的webapps目录中。也可以通过在Tomcat的conf目录中编辑 server.xml文件来实现部署，或者单独部署一个XML文件，这样就不需要编辑server.xml 了。 其他的Servlet容器可能会有不同的部署规则。关于如何将Servlet/JSP应用程序部署到Tomcat的 详细信息，请查阅附录A。

部署Servlet/JSP应用程序时，建议将它部署成一个war文件。war文件其实就是以war 作为扩展名的jar文件。利用带有JDK或者类似WinZip工具的jar软件，都可以创建war文 件。然后，将war文件复制到Tomcat的webapps目录下。当开始启动Tomcat时，Tomcat就 会自动解压这个war文件。部署成war文件在所有Servlet容器中都适用。

1. 3.3 调用 Servlet

要测试这个Servlet，在浏览器中打开下面的URL:

<http://localhost:8080/servletapil/my>

V 1^1 l〇€alhost:SOSO/bl/rrnc^L

〇 D localhost::808O/bl/my =

HeDo from My Servlet

其输出结果应该类似于图C.3。

恭喜，你已经成功编写了第一个Servlet应用 程序！

图C.3 MyServlet的响应

CA ServIetRequest

对于每一个HTTP请求，Servlet容器都会创建一个ServletRequest实例，并将它传给Servlet 的Service方法。ServletRequest封装了关于这个请求的信息。

ServletRequest接口中有如下一些方法。

public int getContentLength()

返回请求主体的字节数。如果不知道字节长度，这个方法就会返回-1。

public j ava.lang.String getContentType()

返回请求主体的MIME类型，如果不知道类型，则返回null。

public j ava.lang.String getParameter(j ava.lang.String name)

返回指定请求参数的值。

public java.lang.String getProtocol()

返回这个HTTP请求的协议名称和版本。

getParameter是ServletRequest中最常用的方法。该方法通常用于返回HTML表单域的值。 在C.10小节中，我们将会学到如何获取表单值。

getParameter也可以用于获取查询字符串的值。例如，利用下面的URI调用Servlet:

<http://domain/context/servletName?id=123>

用下面这个语句，可以通过Servlet内部获取id值：

String id = request.getParameter("id");

注意，如果该参数不存在，getParameter将返回null。

除了 getParameter 夕卜，还可以使用 getParameterNames、getParameterMap 和 getParameterValues

获取表单域名、值以及查询字符串。这些方法的使用范例请参阅C.9小节。

1. 5 ServletResponse

javax.servlet.ServletResponse 接口表示一个 Servlet 响应。在调用 Servlet 的 Service 方法前，

Servlet容器首先创建一个ServletResponse，并将它作为第2个参数传给Service方法。 ServletResponse隐藏了向浏览器发送响应的复杂过程。

在ServletResponse中定义的方法之一是getWriter方法，它返回了一个可以向客户端发送 文本的java.io.PrintWriter。默认情况下，PrintWriter对象使用ISO-8859-1编码。

在向客户端发送响应时，大多数时候是将它作为HTML发送。因此，你必须非常熟悉 HTML。

注意：

还有一个方法可以用来向浏览器发送输出，它就是getOutputStream。但这个方法是用 于发送二进制数据的，因此，大多数情况使用的是getWriter,而不是getOutputStream。

在发送任何HTML标签前，应该先调用setContentType方法，设置响应的内容类型，并 将“text/html”作为一个参数传入。这是在告诉浏览器，内容类型为HTML。在没有内容类型 的情况下，大多数浏览器会默认将响应渲染成HTML。但是，如果没有设置响应内容类型， 有些浏览器就会将HTML标签显示为普通文本。

1. 6 ServletConfig

当Servlet容器初始化Servlet时，Servlet容器会给Servlet的init方'法传入一个ServletConfig。 ServletConfig封装可以通j^@WebServlet或者部署描述符传给Servlet的配置信息。这样，传 入的每一条信息就叫一个初始参数。一个初始参数有key和value两个元件。

为了从Servlet内部获取到初始参数的值，要在Servlet容器传给Servlet的init方法的 ServletConfig 中调用 getlnitParameter 方法。getlnitParameter 的方法签名如下：

j ava.lang.String getlnitParameter(j ava.lang.String name)

此外，getlnitParameterNames方法则是返回所有初始参数名称的一个Enumeration。

j ava.util.Enumeration<j ava.lang.String> getlnitParameterNames()

例如，为了获取contactName参数值，要使用下面的方法签名：

String contactName = ServletConfig.getlnitParameter("contactName");

除 getlnitParameter 和 getlnitParameterNames 之外，ServletConfig 还提供了另一个很有用 的方法：getServletContext。利用这个方法可以从Servlet内部获取ServletContext。关于这个

对象的深入探讨，请查阅C.7小节。

下面举一个 ServletConfig 的范例，在 servletapil 中添加一个名为 ServletConfigDemoServlet 的Servlet。这个新的Servlet如清单C.2所示。

清单 C.2 ServletConfigDemoServlet 类

package servletapil;

import java.io■工〇Exception;

import j ava.io.PrintWriter;

import javax.servlet.Servlet;

import j avax.servlet.ServletConfig;

import j avax.servlet.ServletException;

import j avax.servlet.ServletRequest;

import j avax.servlet.ServletResponse;

import j avax.servlet.annotation.WeblnitParam;

import j avax.servlet.annotation.WebServlet;

@WebServlet(name = "ServletConfigDemoServlet", urlPatterns = { "/servletConfigDemo" }, initParams = {

@WebInitParam(name=MadminM, value=”Harry Taciak"),

@WebInitParam(name="email”， value="admin@example.com”）

}

)

public class ServletConfigDemoServlet implements Servlet { private transient ServletConfig servletConfig;

0〇verride

public ServletConfig getServletConfig() { return servletConfig;

@Override

public void init(ServletConfig servletConfig) throws ServletException { this.servletConfig = servletConfig;

0〇verride

public void service(ServletRequest request,

ServletResponse response)

throws ServletException, IOException {

ServletConfig servletConfig = getServletConfig();

String admin = servletConfig.getlnitParameter("admin"); String email = servletConfig.getlnitParameter("email");

response. setContentType {"text/html11); PrintWriter writer = response.getWriter{); writer.print("<!DOCTYPE html>H + "<html>"

+ "<body>"

+ ''Admin:n + admin + n<br/>Email:" + email + "c/bodyx/htmlV’）；

^Override

public String getServletlnfo() {

return "ServletConfig demo";

QOverride

public void destroy() {

如清单C.2所示，在@WebServlet的initParams属性中，给Servlet传入了两个初始参数 (admin 和 email)。

@WebServlet(name = "ServletConfigDemoServlet", urlPatterns = { "/servletConfigDemo" }, initParams = {

0WeblnitParam(name—"admin"f value—"Harry Taciak"), @WebInitParaia(name="email", value=T,admin@exan^>le. con")

用下面这个 URL,可以调用 ServletConfigDemoServlet:

http://localhost:808O/servletapil/servletConfigDemo

其结果类似于图C.4。

I . iaiar\_

V ■ localhost:8080/sen. ^

\* C D localhost:8080/5ervletapi1/servletConfigDemo £□=

Admin:Harry Taciak [Emailradniin@exaniple.com](mailto:Emailradniin@exaniple.com)

图 C.4 ServletConfigDemoServlet 效果展示

另一种方法是，在部署描述符中传入初始参数。在这里使用部署描述符，比使ffi@WebServlet 更容易，因为部署描述符是一个文本文件，不需要重新编译Servlet类，就可以对它进行编辑。

部署描述符将在C.11节详细讲解。

1. 7 ServletContext

ServletContext表示Servlet应用程序。每个Web应用程序只有一个上下文。在将一个 应用程序同时部署到多个容器的分布式环境中时，每台Java虚拟机只有一个ServletContext

对象。

通过在 ServletConfig 中调用 getServletContext 方法，可以获得 ServletContext。

有了 ServletContext，就可以共享可以从应用程序中的所有资料处访问到的信息，并且可 以动态注册Web对象。前者将对象保存在ServletContext中的一个内部Map中。保存在 ServletContext中的对象称作属性。

ServletContext中的下列方法负责处理属性:

j ava.lang.Obj ect getAttribute(j ava.lang.String name) java.util.Enumeration<java.lang.String> getAttributeNames() void setAttribute(java.lang.String name, j ava.lang.Obj ect object) void removeAttribute(java.lang.String name)

1. 8 GenericServlet

前面的例子展示了如何通过实现Servlet接口来编写Servlet。但你注意到没有？它们必须 给Servlet中的所有方法都提供实现，即便其中有一些实现压根就没有包含任何代码。此外， 还需要将ServletConfig对象保存到类级变量中。

值得庆幸的是，GenericServlet抽象类出现了。本着面向对象编程的尽可能使代码简单的 原则，GenericServlet实现了 Servlet和ServletConfig，并完成以下任务：

•将init方法中的ServletConfig赋给一个类级变量，以便可以通过调用getServletConfig

获取。

•为Servlet接口中的所有方法提供默认的实现。 •提供方法，包围ServletConfig中的方法。

GenericServlet通过将ServletConfig赋给init方法中的类级变量servletConfig，来保存

ServletConfig〇 下面就是 GenericServlet 中的 init 实现。

public void init(ServletConfig servletConfig) throws ServletException { this.ServletConfig = ServletConfig; this.init();

}

但是，如果在类中覆盖了这个方法，就会调用Servlet中的init方法，并且还必须调用 super.init(servletConfig)来保存 ServletConfig。为了避免上述麻烦，GenericServlet 提供了第 2 个init方法，它不带参数。这个方法是在ServletConfig被赋给ServletConfig后，由第1个init

方法调用。

public void init(ServletConfig ServletConfig) throws ServletException { this.servletConfig = ServletConfig; this.init();

}

这意味着，可以通过覆盖没有参数的init方法来编写初始化代码，ServletConfig则仍然由 GenericServlet 实例保存。

清单 C.3 中的 GenericServletDemoServlet 类是对清单 C.2 中 ServletConfigDemoServlet 类 的改写。注意，这个新的Servlet扩展了 GenericServlet，而不是实现Servlet。

清单 C.3 GenericServletDemoServlet 类

package servletapil;

import java■io•I〇Exception;

import j ava.io.PrintWriter;

import javax.servlet.GenericServlet;

import j avax.servlet.ServletConfig;

import j avax.servlet.ServletException;

import j avax.servlet.ServletRequest;

import j avax.servlet.ServletResponse;

import j avax.servlet.annotation,WeblnitParam;

import j avax.servlet.annotation,WebServlet;

@WebServlet (name = "GenericServletDemoServlet11, urlPatterns = { "/generic" }, initParams = {

0WebInitParam(name=MadminM, value=MHarry Taciak")/ @WebInitParam (name=Memail", value=Madinin0example . com11)

public class GenericServletDemoServlet extends GenericServlet {

private static final long serialVersionUID = 62500890L;

QOverride

public void service(ServletRequest request,

ServletResponse response)

throws ServletException, IOException {

ServletConfig servletConfig = getServletConfig();

String admin = servletConf ig. getlnitParameter (''admin");

String email = servletConfig•getlnitParameter("email”）； response.setContentType("text/html");

PrintWriter writer = response.getWriter(); writer.print(n<!DOCTYPE html>"

+ ,,<html><head></head><body>,,

+ "Admin:M + admin + ”<br/>Email:” + email + M</body></html>");

}

}

可见，通过扩展GenericServlet,就不需要覆盖那些不打算改变的方法。因此，代码 变得更加整洁。在清单C.3中，唯一被覆盖的方法是Service方法。而且，不必亲自保存 ServletConfig。

用下面这个URL调用Servlet，其结果应该与ServletConfigDemoServlet相似。

<http://localhost:8080/servletapil/generic>

即使GenericServlet是对Servlet —个很好的加强，但它也不常用，因为它毕竟不像 HttpServlet那么髙级。HttpServlet才是主角，在现实的应用程序中广泛使用。关于它的详情， 请查阅C.9节。

**C.9 Http Servlets**

即便不是全部，至少也是大多数应用程序都要与HTTP结合起来使用。这意味着可以利 用HTTP提供的特性。javax.servlet.http包是Servlet API中的第2个包，其中包含了用于编写 Servlet应用程序的类和接口。javax.servlethttp中的许多类型都覆盖了 javax.servlet中的类型。

图C.5展示了 javax.servlet.http中的主要类型。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | «interface» javax.sen/let Servlet |  | «interface»  javax.servlet ServletRequest | | |  | «interface»  javax.sen/let ServletResponse |
|  | A  i  i  A |  |  | f |  | |  |
| javax.servlet.GenericServlet | | |  | «interface»  HttpServletRequest |  | | «interface»  HttpServletResponse |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HttpServlet |  | «interface»  HttpSession |  | Cookie |

图C.5 javax.servlet.http中的主要类型

1. 9.1 HttpServlet

HttpServlet 类覆盖了 javax.servlet.GenericServlet 类。使用 HttpServlet 时，还要借助分别 代表 Servlet 请求和 Servlet 响应的 HttpServletRequest 和 HttpServletResponse 对象。 HttpServletRequest 接 口扩展 javax.servlet.ServletRequest，HttpServletResponse 扩展 javax.servlet. ServletResponse〇

HttpServlet覆盖GenericServlet中的Service方法，并通过下列签名再添加一个Service方法:

protected void service(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, java•i〇•IOException

新Service方法和javax.servlet.Servlet中Service方法之间的区别在于，前者接受 HttpServletRequest 和 HttpServletResponse，而不是 ServletRequest 和 ServletResponse。

像往常一样，Servlet 容器调用 javax.servlet.Servlet 中原始的 Service 方法。HttpServlet 中

的方法编写如下：

public void service(ServletRequest req, ServletResponse res) throws ServletException,工〇Excepti〇n {

HttpServletRequest request;

HttpServletResponse response; try {

request = (HttpServletRequest) req; response = (HttpServletResponse) res;

} catch (ClassCastException e) {

throw new ServletException("non-HTTP request or response");

service(request, response);

}

原始的Service方法将Servlet容器的request和response对象分别转换成HttpServletRequest 和HttpServletResponse，并调用新的Service方法。这种转换总是会成功的，因为在调用Servlet 的 Service 方法时，Servlet 容器总会传入一个 HttpServletRequest 和一个 HttpServletResponse， 预备使用 HTTP。即便正在实现 javax.servletServlet，或者扩展 javax.servletGenericServlet， 也可以将传给 Service 方法的 servlet request 和 servlet response 分别转换成 HttpServletRequest 和 HttpServletResponse。

然后，HttpServlet中的Service方法会检验用来发送请求的HTTP方法（通过调用 request.getMethod)，并调用以下方法之一：doGet、doPost、doHead、doPut、doTrace、doOptions 和doDelete。这7种方法中每一种方法都表示一个HTTP方法。doGet和doPost是最常用 的。因此，不再需要覆盖Service方法了，只要覆盖doGet或者doPost，或者二者都覆盖 即可。

总之，HttpServlet有两个特性在GenericServlet中尚未体现出来。

•不覆盖Service方法，而是覆盖doGet或者doPost，或者覆盖doGet和doPost。在少 数情况下，还会覆盖以下任意方法：doHead、doPut、doTrace、doOptions和doDelete。

•与 HttpServletRequest 和 HttpServletResponse 共用，而不是与 ServletRequest 和 ServletResponse 共用。

HttpServletRequest 表示 HTTP 环境中的 Servlet 请求。它扩展 javax.servlet.ServletRequest

接口，并添加了几个方法。新增的部分方法如下：

j ava.lang•String getContextPath()

返回表示请求上下文的请求RUI部分。

Cookie[] getCookies()

返回一个Cookie对象数组。

java.lang.String getHeader(java.lang.String name)

返回指定HTTP标题的值。

java.lang.String getMethod()

返回生成这个请求的HTTP方法名称。

j ava \_lang•String getQueryString()

返回请求URL中的查询字符串。

HttpSession getSession()

返回与这个请求相关的会话对象。如果没有，将创建一个新的会话对象。

HttpSession getSession(boolean create)

返回与这个请求相关的会话对象。如果有，并且create参数为True,将创建一个新的会 话对象。

C.9.2 HttpServletResponse

HttpServletResponse表示HTTP环境中的Servlet响应。下面是其中定义的部分方法：

void addGookie(Cookie cookie)

给这个响应对象添加一个cookie。

void addHeader(java.lang.String name, j ava.lang.String value)

给这个响应对象添加一个header。

void sendRedirect(java.lang.String location)

发送一条响应码，将浏览器跳转到指定的位置。

下面的章节将进一步学习这些方法。

**C.10**处理**HTML**表单

一个Web应用程序中几乎总会包含一个或者多个HTML表单，供用户输入值。你可以轻 松地将一个HTML表单从一个Servlet发送到浏览器。当用户提交表单时，在表单元素中输入 的值就会被当作请求参数发送到服务器。

HTML输入字段（文本字段、隐藏字段或者密码字段）或者文本区的值，会被当作字符串发 送到服务器。空的输入字段或者文本域区会发送空的字符串。因此，有输入字段名称的 ServletRequestgetParameter 绝对不会返回 null。

HTML的select元素也向header发送了一个字符串。如果select元素中没有任何选项被 选中，那么就会发出所显示的这个选项值。

包含多个值的select元素（允许选择多个选项并且用<selectmultiple>表示的select元素） 发出一个字符串数组，并且必须通过SelectRequest.getParameterValues进行处理。

复选框比较奇特。选中的复选框会发送字符串“on”到服务器。未选中的复选框则不向 服务器发送任何内容，ServletRequestgetParameter(fieldName)返回 null 〇

单选框将被选中按钮的值发送到服务器。如果没有选择任何按钮，将没有任何内容被发 送到服务器，并且 ServletRequest.getParameter(fieldName)返回 null。

如果一个表单中包含多个输入同名的元素，那么所有值都会被提交，并且必须利用 ServletRequest.getParameterValues 来获取它们。ServletRequestgetParameter将只返回最后一个值。

清单C.4中的FormServlet类展示了如何处理HTML表单。它的doGet方法将一个Order 表单发送到浏览器。它的doPost方法获取到所输入的值，并将它们输出。这个Servlet就是 servletapi2应用程序的一部分。

清单 C.4 FormServlet 类

package servletapi2;

import java•io.IOException;

import j ava.io.PrintWriter;

import j ava.util.Enumeration;

import j avax.servlet.ServletException;

import j avax.servlet.annotation.WebServlet;

import j avax.servlet.[http.HttpServlet](http://http.HttpServlet);

import j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest);

import j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse);

@WebServlet(name = "FormServlet", urlPatterns = { "/form" }) public class FormServlet extends HttpServlet {

private static final long serialVersionUID = 54L; private static final String TITLE = "Order Form’1;

0〇verride

public void doGet (HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { response.setContentType("text/html");

PrintWriter writer = response.getWriter(); writer.println(n<!DOCTYPE html>M); writer.println(n<html>M); writer.println(M<head>M);

writer.println(M<title>M + TITLE + M</title></head>M); writer. println (M<bodyXhl>n + TITLE + M</hl>M);

writer.println("<form method=1 post1>M);

writer.println(M<table>M);

writer.println(M<tr>");

writer.println(n<td>Name:</td>M);

writer .println (n<tdxinput name=1 name1 /></td>M);

writer.println(n</tr>n);

writer.println(M<tr>");

writer.println("<td>Address:</td>");

writer. println (M<tdxtextarea name=1 address 1 ’’

+ Mcols=1401 rows=151></textarea></td>n); writer.println(M</tr>M); writer.println(n<tr>"); writer .println (ll<td>Country:</td>"); writer.println(M<td><select name=1 country1>n); writer.println{M<option>United States</option>n); writer.println("<option>Canada</option>"); writer .println (n</selectX/td>M); writer.println <"</tr>"); writer.println(n<tr>");

writer.println(n<td>Delivery Method:</td>M); writer. println (n<tdxinput type=1 radio 1 " +

Mname=1deliveryMethod1n + M value=1 First Class 1 />First Class’1); writer.println("<input type=1 radio 1 M +

Mname=1 deliveryMethod1 ’’

+ Mvalue=1 Second Class 1/>Second Class</td>M); writer.println(n</tr>n); writer.println(n<tr>");

writer.println(M<td>Shipping Instructions:</td>"); writer. println {M<tdxtextarea name=1 instruction 1 11 + Mcols=1401 rows=151></textarea></td>M); writer.println(n</tr>M); writer.println(M<tr>"); writer.println(M<td>&nbsp;</td>M);

writer.println(M<td><textarea name=1 instruction 1 "

+ Mcols=1401 rows=151></textarea></td>n); writer.println(M</tr>M); writer.println(n<tr>");

writer.println(n<td>Please send me the latest " + "product catalog:</td>n); writer. println (M<tdxinput type=1 checkbox1 M +

Mname=1catalogRequest1/></td>M); writer.println <"</tr>"); writer.println(M<tr>"); writer.println(n<td>&nbsp;</td>M); writer. println (n<tdxinput type=1 reset1 />n +

"<input type=1 submit1/></td>M); writer.println <"</tr>"); writer.println(M</table>M); writer.println(n</form>M); writer.println(n</body>M); writer.println(n</html>M);

0〇verride

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { response.setContentType("text/html");

PrintWriter writer = response.getWriter(); writer. println {M<html>,1); writer. println ("<head>11);

writer .println (M<title>M + TITLE + n</titleX/head>M); writer.println(M</head>M);

writer. println (If<body><hl>" + TITLE + M</hl>M);

writer.println(n<table>M);

writer.println(n<tr>");

writer.println(M<td>Name:</td>n);

writer.println(M<td>" + request.getParameter("name")

+ M</td>M);

writer.println <"</tr>");

writer.println(M<tr>");

writer .println (ll<td>Address:</td>");

writer.println(n<td>" + request.getParameter("address") + n</td>n);

writer.println("</tr>");

writer.println(M<tr>");

writer.println(M<td>Country:</td>");

writer. println (M<td>" + request • getParameter (’’country1’） + M</td>M);

writer.println <"</tr>"); writer.println(M<tr>");

writer.println(M<td>Shipping Instructions:</td>"); writer.println(n<td>");

String[] instructions = request

.getParameterValues("instruction"); if (instructions != null) {

for (String instruction : instructions) { writer.println(instruction + M<br/>M);

}

}

writer.println(n</td>M);

SIZ

</ 4 0uihu **t** =auiHu q.nduT> **<i**4**sodi=p**〇**n^aui uaog:>**

°盤說陂锾梁荔亩挲1WXH+裒甲寻潑牟P£)〇P明岜Ml璐

uij〇g:/STd^^sxAJ0S/〇8〇8 ：^soq-[^D〇x// :d^^q

:PIAJ3SUU〇d宙齚扣但‘Ufl明厘土宙

(il<IUI^q/>ii) U-[^UTJ：d'

/ (il<Apoq/>ii) ux^uT^d-

/ (ii<ATp/>ii) UI^UT^d-

{

{

■ (“</JGt>“ + sn-[HA^JHd) ux^UTjd' Jsq.TjM

**} (ssn"[P**八**uipjpd : 6uxjq.s) ^**〇**I**

**i** (suiHjstui'ejpd) '

^sanbaz = sanxHAUJHJPd [ ] **Butj^s i** (u : u + auipjsi^^^d) ux^UT^d'

/ () ^USUISXa^XBU ■ S0UIEJS[J040UIHJHd = SUIHNUIHJHd 5**utj^s }** ( () sq\_ii3ui3\_[39J：〇i^S'Bt[• ssuiHNjaq^smej'Bd)

/ () \*

q.sanbaj = saui'ejsiaaq.aui'ej'ed <5uTjq.s>u〇Tq.ejauinua **■' (“</JC[>**〇**JU**工 **£>nq0Q“ > uTq\_UTj**：**d.**

■ (»< \* %06 : 9ZTs-^u〇J -xd〇^ : do^-UTbJHUij,

+ «\*PPP# PTT〇s xdx : japjoq. ATp>u) uxq.ux' J9q.T

• („〇iqe^/>ii) UI^UT^d' **i** („<^^/>„)UI^UT^d- **•(»<P^/>»)UI^UT^d-**

{

-(usaA„) ui^u-pjd-

} 9SI0 {

• (a〇N»i) UT^UT^d-

} (xinu == (^SBnbB's.box^^vou) ■ q.S0nb3J) **jt**

/(U<P^>»)UI^UT^d-

* (»<pq./>：^ssnb0H 5〇x^^^〇<pq.>u) uxq.uxjd • Jsq.T^M

/ (u<jq.>u) ux^uxad' jaq.TJM ^(U<J^/>U)UI^UTjd'

’（“<P3/>“ +

u<P4>u ) UI^UTJ：d '

* (»<pq./>：p〇qq.3W A^SAT-[0a<pq.>ii) uxq.uT^d ■

；**(u<j^>u)**ux^uT^d- •(»<J^/>»)UI^UT^d-

吉蓽11ALLH垂帑〇r〇

<textarea name=1 address1 cols='401 rows=151></textarea>

<select name='country'>n);

<option>United States</option>

<option>CanadLa</option>

</select>

<input type=Tradio' name='deliveryMethod1 value='First Class'/>

<input type=Tradio' name='deliveryMethod1 value='Second Class"/>

<textarea name=1 instruction1 cols='40' rows=15'></textarea>

<textarea name=,instruction1 cols='401 rows=15'></textarea>

<input type=Tcheckbox\* name=1catalogRequest'/>

<input type=Treset'/>

<input type='submit1/>

</form>

表单的方法设为post，确保当用户提交表单时，使用HTTP POST方法。它的action属性 缺省，表示该表单会被提交给请求它时使用的相同的URL〇

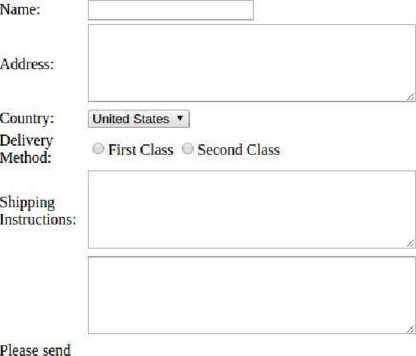
图C.6展示了一个空的Order表单。

现在，填写表单，并单击Submit按钮。在表单中输入的值，将利用HTTP POST方法发 送给服务器，这样就会调用Servlet的doPost方法。因此，你将会看到如图C.7所示的那些值。



**G** D localhost:8080/servlGtapi2/form ^ □ **E**

Order Form



me the latest

product

catalog:

| Reset | Submit |



<-• G D localhost:8080/sGrvlGtapi2/form ^ □ E

|  |  |
| --- | --- |
| Order Form | |
| Name; | John Lilas |
| Address: | 4413 Example Court Orlando Florida |
| Country: | United States |
|  |  |
| Shipping Instructions; | Don’t disturb the dog |
| Delivery Method: | First Class |
| Catalog Request : | Yes |
| Debug Info |  |
| name: John Lilas |  |
| address: 4413 Example Court Orlando Florida | |
| country: United States |  |
| deliveryMethod: First Class | |
| instnidion: Leave at door | |
| Don't disturb the dog |  |
| catalogRequest: on |  |
|  | |

图**C.6 —**个空的**Order**表单

图C.7在Order表单中输入的值

C. 11使用部署描述符

C.11使用部署描述符

正如在前面的例子中所见到的，编写和部署Servlet都是很容易的事情。部署的一个方面 是用一个路径配置Servlet的映射。在这些范例中，是利用WebServlet标注类型，用一个路径 映射了一个 Servlet。

利用部署描述符是配置Servlet应用程序的另一种方法，部署描述符总是命名为web.xml，并 且放在WEB-INF目录下。本章介绍了如何创建一个名为servletapi3的Servlet应用程序，并 为它编写了一个web.xml。

servletapi3 有 SimpleServlet 和 WelcomeServlet 两个 Servlet，还有一个要映射 Servlets 的部署 描述符。清单C.5和清单C.6分别展示了 SimpleServlet和WelcomeServlet。注意，Servlet类 是没有用@WebServlet注解的。部署描述符如清单C.7所示。

清单C.5未注解的SimpleServlet类

package servletapi3;

import java•io•工OException;

import j ava.io.PrintWriter;

import j avax.servlet.ServletException;

import j avax.servlet.[http.HttpServlet](http://http.HttpServlet);

import j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest);

import j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse);

public class SimpleServlet extends HttpServlet {

private static final long serialVersionUID = 8946L;

©Override

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { response.setContentType("text/html");

PrintWriter writer = response.getWriter(); writer . print ("< ! DOCTYPE html><html><head></head>,, + ,,<body>Simple Servlet</body></html");

清单C.6未注解的WelcomeServlet类

package servletapi3;

j ava.io•工OException;

j ava.io.PrintWriter;

import

import

import

import

import

import

j avax.servlet.ServletException;

j avax.servlet.[http.HttpServlet](http://http.HttpServlet);

j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest);

j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse);

public class WeicomeServlet extends HttpServlet {

private static final long serialVersionUID = 27126L;

@Override

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { response.setContentType("text/html");

PrintWriter writer = response.getWriter(); writer . print ("< ! DOCTYPE htmlXhtmlXheadX/head〉" + "<body>Welcome</body></htinl>’’）；

清单C.7部署描述符

<?xml version=" 1.0M encoding=,,UTF-8,' ?>

<web-app version=M3.ln

xmlns=M<http://xmlns.j> cp.org/xml/ns/j avaee" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi : schemaLocation=,,http: / /xmlns . j cp. org/xml/ns/j avaee http: //xmlns .jcp.org/xml/ns/javaee/web-app\_3\_l .xsdM>

<servlet>

<servlet-name>SimpleServlet</servlet-name> <servlet-class>servletapi3.SimpleServlet</servlet-class> <load-on-startup>10</load-on-startup>

</servlet>

<servlet-mapping〉

<servlet-name>SimpleServlet</servlet-name> <ur1-pattern>/simple</ur1-pattern>

</servlet-mapping>

<servlet>

<servlet-name>WelcomeServlet</servlet-name> <servlet-class>servletapi3.WeicomeServlet</servlet-class> <load-on-startup>20</load-on-startup〉

C.11使用部署描述符

</servlet> <servlet-mapping>

<servlet-name>WelcomeServlet</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>

</web-app>

使用部署描述符有诸多好处。其一，可以将没有对等元素的元素放在@WebServlet中， 如load-on-startup元素。这个元素在应用程序启动时加载Servlet，而不是在第一次调用Servlet 时加载。使用l〇ad~on-startup意味着第一次调用Servlet所花的时间并不比后续的调用长。如 果Servlet的init方法需要花费一些时间才能完成的话，这项功能就特别有用。

使用部署描述符的另一个好处是，如果需要修改配置值，如Servlet路径，则不需要重新 编译Servlet类。

此外，可以将初始参数传给一个Servlet,并且不需要重新编译Servlet类，就可以对它们 进行编辑。

部署描述符还允许覆盖在Servlet标注中定义的值。Servlet上的WebServlet注解如果同时 也在部署描述符中进行声明，那么它将不起作用。但是，在有部署描述符的应用程序中，却 不在部署描述符中注解的Servlet时，则仍然有效。这意味着，可以注解Servlet,并在同一个 应用程序的部署描述符中声明这些Servlet。

图C.8展示了有部署描述符的servlet^)i3的目录结构。这个目录结构与servletapil的目录结 构没有太大区别。唯一的区别在于，sexvletapi3在WEB-INF目录中有一个web.xml文件（部署描 述符)。

G web ► □ META-INF ^ <3 WEB-INF v <3 classes ▼ G servletapB

0 SirnpleServlet.class [il WelcomeServlet.class Q web.xml

**图**C**.8有部署描述符的**servletapi3**的目录结构**

现在，在部署描述符中声明SirnpleServlet和WelcomeServlet，可以利用这些URL来访问

它们：

<http://localhost:8080/servletapi3/siraple> <http://localhost:8080/servletapi3/welcome>

C.12小结

Servlet技术是Java EE的一部分。所有Servlet都在Servlet容器中运行，容器和Servlet 之间的契约釆用javax.servlet.Servlet接口的形式。javax.servlet包还提供实现Servlet的 GenericServlet抽象类。这是一个方便的类，你可以扩展以创建一个Servlet。然而，大多数现 代Servlet将在HTTP环境中工作。因此，对javax.servlet.http.HttpServlet类进行子类化更有意 义。HttpServlet 类本身是 GenericServlet 的子类。

附录D

JavaServer Pages

Servlet有两个缺点是无法克服的：首先，写在Servlet中的所有HTML标签必须包含Java 字符串，这使得处理HTTP响应报文的工作十分繁琐；第二，所有的文本和HTML标记是硬 编码，导致即使是表现层的微小变化，如改变背景颜色，也需要重新编译。

JavaServer Pages (JSP)解决了上述两个问题。同时，JSP不会取代Servlet，相反，它们 具有互补性。现代的Java Web应用会同时使用Servlet和JSP页面。撰写本书时，JSP的最新 版本是2.3。

本附录概述了 JSP技术，并讨论了在JSP页面中，隐式对象以及3个语法元素（指令、 脚本元素和动作)，还讨论了错误处理。

注意

一种好设计方法是：不要在JSP页面中编写Java代码。除非应用程序只包含一个或两 个简单的JSP页面，并且永远不会增长；否则，你应该采用模型2，因为它规定JSP页面仅 用于显示Java对象中的值。

1. **l JSP**概述

JSP页面本质上是一个Servlet。然而，用JSP页面开发比使用Servlet更容易，主要有两 个原因。首先，不必编译JSP页面；其次，JSP页面是一个以jsp为扩展名的文本文件，可以 使用任何文本编辑器来编写它们。

JSP页面在JSP容器中运行，一个Servlet容器通常也是JSP容器。例如，Tomcat就是一 个 Servlet/JSP 容器。

当一个JSP页面第一次被请求时，Servlet/JSP容器主要做以下两件事情：

1. 把JSP页面转换到JSP页面实现类，该实现类是一个实现javax.servletjsp.JspPage接 口或子接口 javax.servletjsp.HttpJspPage 的 Java 类。JspPage 是 javax.servlet.Servlet 的子接口，

这使得每一个JSP页面都是一个Servlet。该实现类的类名由Servlet/JSP容器生成。如果出现 转换错误，则相关错误信息将被发送到客户端。

1. 如果转换成功，Servlet/JSP容器随后编译该Servlet类，并装载和实例化该类，像其 他正常的Servlet —样执行生命周期操作。

对于同一个JSP页面的后续请求，Servlet/JSP容器会先检查JSP页面是否被修改过。如 果是，则该JSP页面会被重新转换、编译并执行。如果不是，则执行已经在内存中的JSP Servlet。 这样一来，一个JSP页面的第一次调用的实际花费总比后来的花费多，因为它涉及转换和编 译。为了解决这个问题，可以执行下列动作之一：

•配置应用程序，使所有的JSP页面在应用程序启动时被调用（实际上也可视为转换和 编译)，而不是在第一次请求时调用。

•预编译JSP页面，并将其部署为Servlet。

JSP自带的API包含4个包：

* javax.servlet.jsp。包含Servlet/JSP容器用于将JSP页面转换为Servlet的核心类和接 口。其中的两个重要成员是JspPage和HttpJspPage接口。所有的JSP页面实现类必 须实现JspPage或HttpJspPage接口。在HTTP环境下，实现HttpJspPage接口是显而

易见的选择。

* javax.servletjsp.tagext。包括用于开发自定义标签的类型。
* javax.el。提供了统一表达式语言的API。
* javax.servlet.jsp.el。提供了一组必须由Servlet/JSP容器支持以便在JSP页面中使用表

达式语言的类。

除了 javax.servletjsp.tagext，我们很少直接使用JSPAPI。事实上，编写JSP页面时，我 们更关心Servlet API，而非JSP API。当然，我们还需要掌握JSP语法，本附录后续会进一步 说明。JSP API在开发JSP容器或JSP编译器时被广泛使用。

可以在以下网址查看JSP API:

<https://docs.oracle.com/javaee/7/api/javax/servlet/jsp/> package-surranary.html

JSP页面可以包含模板数据和语法元素。这里，语法元素是一些具有特殊意义的JSP转 换符。例如，“<%”是一个元素，因为它表示在JSP页面中的Java代码块的开始。“％>”也 是一个元素，因为它是Java代码块的结束符。除语法元素之外的一切都是模板数据。模板数 据会原样发送给浏览器。例如，JSP页面中的HTML标记和文字都是模板数据。

清单D.1给出一个名为welcome.jsp的JSP页面。它是发送一个客户问候的简单页面。注 意，同Servlet相比，JSP页面是如何更简单地完成同样的事情的。

清单 D.1 welcome.jsp

<!DOCTYPE html>

<html>

<headxtitle>Welcome</title></head>

<body>

Welcome

</body>

</html>

在Tomcat中，welcome.jsp页面在第一次请求时被转换成名为welcomejsp的Servlet。你 可以在Tomcat工作目录下的子目录中找到生成的Servlet，该Servlet继承自org.apache.jasper. runtime.HttpJspBase，这是一个抽象类，它继承自 javax.servlet[http.HttpServlet](http://http.HttpServlet) 并实现了 javax. servletjsp.HttpJspPage 〇

下面是为welcome.jsp生成的Servlet。如果觉得不好理解，可以先跳过它。当然，能够理

解它更好。

package org.apache.j sp; import j avax.servlet.\*; import j avax.servlet.http.\*; import j avax.servlet.j sp.\*;

public final class welcome」sp extends

org.apache.j asper.runtime.HttpJspBase

implements org.apache.j asper.runtime.JspSourceDependent {

private static final javax.servlet.jsp.JspFactory \_jspxFactory = javax.servlet.j sp.JspFactory.getDefaultFactory();

private static java.util.Map<j ava.lang.String,java.lang.Long>

\_j spx\_dependants;

private j avax.el.ExpressionFactory \_el\_expressionfactory; private org.apache.tomcat.InstanceManager \_j sp\_instancemanager;

public java.util.Map<java.lang.String,java.lang.Long> getDependants() {

return \_jspx\_dependants;

public void \_jsplnit(> {

\_el\_expressionfactory =

\_j spxFactory•getJspApplicationContext( getServletConfig().getServletContext())

.getExpressionFactory();

\_j sp\_instancemanager =

org•apache.j asper.runtime.InstanceManagerFactory .getlnstanceManager(getServletConfig());

public void \_jspDestroy() {

}

public void \_jspService(final

javax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest) request, final

javax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse) response)

throws j ava.io.IOException, j avax.servlet.ServletException {

final javax.servlet.j sp.PageContext pageContext;

j avax.servlet.[http.HttpSession](http://http.HttpSession) session = null;

final j avax.servlet.ServletContext application;

final j avax.servlet.ServletConfig config;

javax.servlet.jsp.JspWriter out = null;

final java.lang.Object page = this;

javax.servlet.jsp.JspWriter \_jspx\_out = null;

javax. servlet. j sp. PageContext \_j spx\_page\_context = null;

try {

response.setContentType{"text/html");

pageContext = \_jspxFactory.getPageContext (this, request, response, null, true, 8192, true);

\_j spx\_page\_context = pageContext; application = pageContext.getServletContext(); config = pageContext.getServletConfig(); session = pageContext.getSession(); out = pageContext.getOut{);

\_jspx\_out = out;

out.write(M<html>\nM);

out.write{M<headxtitle>Welcome</title></head>\nM); out.write(M<body>\nM);

out.write("WelcomeNn"); out.write (,,</body>\n"); out.write (,,</html>");

} catch (j ava.lang.Throwable t) { if (!(t instanceof

j avax.servlet.j sp.SkipPageException)){ out = \_jspx\_out;

if (out != null && out.getBufferSize () !=〇）

try {

out.clearBuffer();

} catch (j ava.io.IOException e) {

}

if (\_jspx一page\_context != null)

\_j spx\_page\_context.handlePageException (t);

}

} finally {

\_j spxFactory. releasePageContext (\_j spx\_page\_context);

正如我们在上面的代码中看到的，JSP页面的主体是JspService方法。这个方法被定义在 HttpJspPage中，并被HttpJspBase的service方法调用。下面的代码来自HttpJspBase类。

public final void service(HttpServletRequest request,

HttpServletResponse response) throws ServletException,

IOException {

\_jspService(request, response);

}

要覆盖init和destroy方法，可以参见D.5。

一个JSP页面不同于一个Servlet的另一个方面是，前者不需要添加注解或在部署描述符 配置映射URL。应用程序目录中的每一个JSP页面可以直接在浏览器 a web

► **a** META-INF

中输入路径页面来访问。图D.l给出了 D1应用程序的目录结构 Q WEB-INF

0 weLcome.jsp

D1应用程序的结构非常简单，由一个空的WEB-INF目录和图ai D1应用程序的 welcome.jsp页面构成。 目录结构

可以通过如下URL访问welcome.jsp页面。

<http://localhost:8080/j> spdemo/welcome.j sp

注意

添加新的JSP界面后，无需重启Tomcat。

清单D.2展示了如何在JSP页面中使用Java代码来生成动态页面。清单D.2中的 todaysDate.jsp页面显示了今天的曰期。

清单 D.2 todaysDate.jsp 页面

<%@page import=n j ava. util. Dateff%>

<%0page import=n j ava. text. DateFormat,f%>

<!D〇CTYPE html>

<html>

<headxtitle>Today1 s date</title></head>

<body>

<%

DateFormat dateFormat =

DateFormat.getDatelnstance(DateFormat.LONG);

String s = dateFormat. format (new Date〇); out .println ("Today is ’’ + s);

%>

</body>

</html>

todaysDatejsp页面将几个HTML标签和字符串“今天是”以及今天的日期发送到浏览器。

请注意两件事情。首先，Java代码可以出现在JSP页面中的任何位置，并通过“<%”和 “％>”包括起来。其次，可以使用page指令的import属性导入在JSP页面中使用的Java类型， 如果没有导入的类型，必须在代码中写出Java类的全路径名称。

<%...%>块称为scriplet，并在D.5部分进一步讨论。page将在C.4部分详细讨论。

现在可以通过如下URL访问todaysDate.jsp页面。

<http://localhost:8080/j> spdemo/todaysDate.j sp

1. **2**注释

为jsp页面添加注释是一个良好的习惯。jsp支持两种不同的注释格式。

1. JSP注释。该注释记录页面中做了什么。
2. HTML/XHTML注释。这些注释将会发送到浏览器。

JSP注释以开始，以结束。下面是一个例子：

<%-- retrieve products to display 一一%>

JSP注释不会被发送到浏览器端，也不会被嵌套。

HTML/XHTML注释语法如下：

<!-- *[comments here*]——>

一个HTML/XHTML注释不会被容器处理，会原样发送给浏览器。HTML/XHTML注释 的一个用途是用来确定JSP页面本身。

<!-- this is /j sp/store/displayProducts.j spf -->

当一个应用中存在多个jsp片段时，这会特别有用。开发人员可以很容易地通过在浏览 器中查看HTML源代码来找出是哪一个JSP页面或片段产生了相应的HTML片段。

D.3隐式对象

Servlet容器会传递几个对象给它运行的Servlet。例如，可以通过Servlet的service方法获 取 HttpServletRequest 和 HttpServletResponse 对象，并且可以通过 init 方法访问到 ServletConfig 对象。此外，可以通过调用HttpServletRequest对象的getSession方法访问到HttpSession

对象。

在JSP中，可以通过使用隐式对象来访问上述对象。表D.1展示了 JSP隐式对象。

**表**D**.1** JSP**隐式对象**

|  |  |
| --- | --- |
| 对象 | 类型 |
| request | javax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest) |
| response | javax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse) |
| out | javax.servlet.j sp Jsp Writer |
| session | javax.servlet.[http.HttpSession](http://http.HttpSession) |
| application | j avax. servlet. S ervletContext |
| config | javax.servlet.ServletConfig |
| pageContext | javax.servlet.j sp.PageContext |
| page | javax.servlet.j sp.Http JspPage |
| exception | java.lang.Throwable |

以request为例，该隐式对象代表Servlet/JSP容器传递给Servlet服务方法的Ht^ServletRequest 对象。可以将request理解为指向HttpServletRequest对象的一个引用变量。下面的代码示例，

从HttpServletRequest对象中返回username参数值。

<%

String userName = request.getParameter(nuserNameM);

%>

pageContext用于javax.servlet.jsp.PageContext。它提供了有用的上下文信息，并通过其自 说明的方法来访问各种 Servlet 相关对象，如 getRequest、getResponse、getServletContext、 getServletConfig和getSession。当然，这些方法在脚本中不是非常有用，因为可以更直接地通 过隐式对象来访问 request、response、session 和 application。

此外，PageContext提供了另一组有趣的方法:用于获取和设置属性的方法，即getAttribute 方法和setAttribute方法。属性值可被存储在4个范围之一：页面、请求、会话和应用程序。 页面范围是最小范围，这里存储的属性只在同一个JSP页面可用。请求范围是指当前的 ServletRequest。会话范围指当前的HttpSession中。应用程序范围是指应用的ServletContext中。

PageContext 的 setAttribute 方法签名如下：

public abstract void setAttribute(java.lang.String name, java.lang.Object value, int scope)

其中，scope的取值范围为PageContext对象的最终静态int值：PAGE\_SCOPE、REQUEST\_ SCOPE、SESSION\_SCOPE 和 APPLICATION\_SCOPE。

若要保存一个属性到页面范围，可以直接使用setAttribute重载方法。

public abstract void setAttribute(java.lang.String name, j ava.lang.Obj ect value)

如下脚本将一个属性保存到ServletRequest中。

<%

//product is a Java object

pageContext.setAttribute{"productM, product,

PageContext.REQUEST\_SCOPE);

%>

同样效果的Java代码如下：

<%

request.setAttribute("product", product);

%>

隐式对象out引用了一个javax.servletjsp.JspWriter对象，这类似于你在调用HttpServletResponse 的getWriter方法时得到java.io.PrintWriter。可以通过调用它的print方法将消息发送到浏览

器。例如:

out.println("Welcome");

清单D.3中的implicitObjects.jsp页面展示了部分隐式对象的使用。

清单 D.3 implicitObjects.jsp 页面

<%@page import=njava.util.EnumerationM%>

<!DOCTYPE html>

<html>

<headxtitle>JSP Implicit Objects</title></head>

<body>

<b>Http headers:</b><br/>

<%

for (Enumeration<String> e = request.getHeaderNames(); e.hasMoreElements(); ){

String header = e.nextElement();

out.println(header + ": " + request.getHeader(header) +

,,<br/>,1);

}

%>

<hr/>

<%

out .println (’’Buffer size : M + response. getBufferSize () +

,,<br/>n);

out.println ("Session id: " + session . getld () + ,,<br/>"); out.println("Servlet name: " + config.getServletName() +

,,<br/>n);

out.println("Server info: " + application.getServerlnfo());

%>

</body>

</html>

可以通过访问如下URL来调用implicitObjects.jsp页面：

<http://localhost:8080/j> spdemo/implicitObj ects.j sp

该页面产生了如下内容：

**Http headers:**

host: localhost:8080 connection: keep-alive

accept:text/html,application/xhtml+xml,application/xml;q=〇\_ 9,image/webp, \_ \*/\*;q=0.8

user-agent: Mozilla/5.0 (XI1; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.130 Safari/537.36

accept-encoding: gzip, deflate, sdch

accept-language: en-us **,**en;q=0**.**8,id;q=0**.**6,ms;q=0**.**4,fr;q=0**.**2**,**de;q=0**.**2 cookie: JSESSI〇NID=4E3D1A4994B7F7ED5D7B96C2E3CF3BDE;

Buffer size: 8192

Session id: A4EC1E2FFCE1377FD6DF6545BDDD7909 Servlet name: jsp

Server info: Apache Tomcat/8.0.20

在浏览器中具体看到的内容，取决于所使用的浏览器及其环境。

注意，在默认情况下，JSP编译器会将JSP页面的内容类型设为text/html。如果要使用不 同的类型，则需要调用response.setContentType()或者使用页面指令（详情请参考D.4小节） 来设置内容类型。例如，将内容类型设置为text/json如下所示：

response**.** setContentType (Mtext/j son1');

还要注意的是，页面隐式对象是表示当前的jsp页面，jsp页面的设计者一般不使用它。

**D.4**指令

指令是JSP语法元素的第一种类型。它们指示JSP转换器如何把JSP页面转换为Servlet。 JSP 2.3定义了多个指令，但只有page和include是最重要的，本节会详细讨论。

1. 4.1 page 指令

可以使用page指令来控制JSP转换器转换当前JSP页面的某些方面。例如，可以告诉JSP 用于转换隐式对象out的缓冲器的大小、内容类型，以及需要导入的Java类型，等等。

page指令的语法如下：

<%@ page *attributel="valuel" attribute2="value2” •••* %>

@和page间的空格不是必需的，attributel、attribute】等是page指令的属性。如下是page

指令属性的列表。

• import:定义一个或多个本页面中将被导入和使用的java类型。例如:import^njava.util.Listn 将导入List接口。可以使用通配符来引入整个包，类似import=njava.util.\*’’。可 以通过在两个类型间加入“，”分隔符来导入多个类型，如import=”java.util.ArrayList， java.util.Calendar，java.io.PrintWriter”。此外，JSP 默认导入如下包:java.lang、javax.servlet、 javax.servlet.http、javax.servlet.j sp 〇

* session:值为True，本页面加入会话管理；值为False则相反。默认值为True，访问 该页面时，若当前不存在javax.servlet.http.HttpSession实例，则会创建一个。
* buffer:以kB为单位，定义隐式对象out的缓冲大小。必须以kB后缀结尾。默认大 小为8kB或更大（取决于JSP容器)。该值可以为none,这意味着没有缓冲，所有数 据将直接写入PrintWriter。
* autoFlush:默认值为True。若值为True，则当输出缓冲满时会自写入输出流。而值为 False，贝！J仅当调用隐式对象的flush方法时，才会写入输出流。因此，若缓冲溢出， 则会抛出异常。
* isThreadSafe:定义该页面的线程安全级别。不推荐使用JSP参数，因为使用该参数 后，会生成一些Servlet容器已过期的代码。
* info:指定生成的Servlet类的getServletlnfo方法的返回值。
* errorPage:定义当出错时用来处理错误的页面。
* isErrorPage:标识本页是一个错误处理页面。
* contentType:定义本页面隐式对象response的内容类型，默认是text/html。
* pageEncoding:定义本页面的字符编码，默认是ISO-8859-1。
* isELIgnored:配置是否忽略EL表达式。EL是Expression Language的缩写。
* language:定义本页面的脚本语言类型，默认是Java,遊JSP 2.2中是唯一的合法值。
* extends:定义JSP实现类要继承的父类。这个属性较少使用，仅在非常特殊理由下

使用。

* deferredSyntaxAllowedAsLiteral:定义是否解析字符串中出现“#丨”符号，默认是False。

”是一个表达式语言的起始符号，因而很重要。

* trimDirectiveWhitespaces:定义是否不输出多余的空格/空行，默认是False。

大部分page指令可以出现在页面的任何位置，但当page指令包含contentType或pageEncoding 属性时，其必须出现在Java代码发送任何内容之前。这是因为内容类型和字符编码必须在发 送任何内容前设定。

page指令也可以出现多次，但出现多次的指令属性必须具有相同的值。不过，import属 性例外，多个包含import属性的page指令的结果是累加的。例如，以下page指令将同时导

入 java.util.ArrayList 和 java.util.Date 类型。

<%@page import="java.util.ArrayList"%>

<%@page import=n java.util. Date11 %>

如下写法的效果一样：

<%@page import=njava.util.ArrayList, java.util.DateM%>

一个page指令可以同时有多个属性。下面的代码设定了 session属性和buffer属性。

<%@page session="false11 buffer=M16kbM%>

1. 4.2 include 指令

可以使用include指令将其他文件中的内容包含到当前JSP页面。一个页面中可以有多个 include指令。若存在一个内容会在多个不同页面中使用或一个页面不同位置使用的情况，则 将该内容模块化到一个include文件会非常有用。

include指令的语法如下：

<%@ include file=,lurl"%>

其中，@和include间的空格不是必需的，URL为被包含文件的相对路径，若URL以一个 斜杠（/)开始，贝SJ该URL为文件在服务器上的绝对路径，否则为当前JSP页面的相对路径。

JSP转换器处理include指令时，将指令替换为指令所包含文件的内容。换句话说，若存 在清单D.4的copyrightjspf文件，以及主文件清单D.5的main.jsp页面：

清单D.4 copyright.jspf包含文件

<hr/>

&copy;2015 Brainy Software Inc.

<hr/>

清单D.5 main.jsp页面

<!D0CTYPE html>

<html>

<headxtitle>Including a file</title></head> <body>

This is the included content: <hr/>

<%0 include file="copyright.jspfM%>

</body>

</html>

则在main.jsp页面中应用include指令和编写如下页面的效果是一样的：

<!D〇CTYPE html>

<html>

<headxtitle>lncluding a file</title></head>

<body>

This is the included content: <hr/>

<hr/>

&copy;2015 Brainy Software Inc.

<hr/>

</body>

</html>

如上示例中，为保证include指令能正常工作，copyrightjspf文件必须同main.jsp位于相 同的目录。按照惯例，以JSPF为扩展名的文件代表JSP fragement。虽然JSP fragement现在 被称为JSP segment,但为保证一致性，JSPF后缀名依然被保留。

注意，include指令也可以包含静态HTML文件。

此外，include动作（类似于include指令）会在D.6 —节讨论。理解两者之间的区别非常 重要，具体细微的差别参见D.6节的解释。

D.5脚本元素

一个脚本程序是一个java代码块，以<%符号开始，以％>符号结束。以清单D.6的 scriptletTestjsp 页面为例：

清单**D.6**使用脚本程序（**scriptletTest.jsp)**

<%@page import="java.util.EnumerationM%>

<!D0CTYPE html>

<html>

<headxtitle>Scriptlet example</title></head>

<body>

<b>Http headers :</bxbr/>

<%-- first scriptlet —一%>

<%

for (Enumeration<String> e = request.getHeaderNames(); e.hasMoreElements(); ){

String header = e.nextElement();

out.println (header + 11: " + request.getHeader (header) +

,,<br/>f,);

String message = "Thank you.11 ;

%>

<hr/>

<%-- second scriptlet --%>

<%

out.printin(message);

%>

</body>

</html>

在上述清单D.6的JSP页面中，有两个脚本程序，需要注意的是定义在一个脚本程序中 的变量可以被其后续的脚本程序使用。

脚本程序的第一行代码可以紧接<%标记，最后一行代码也可以紧接％>标记，不过，这 会降低代码的可读性。

1. 5.1表达式

每个表达式都会被JSP容器执行，并使用隐式对象out的打印方法输出结果。表达式以 “<%=”开始，并由“％>”结束。例如，在下面一行文本中，黑体字为一个表达式。

Today is <%=java.util.Calendar.getlnstance().getTime()%>

注意，表达式无需分号结尾。

JSP 容器首先执行 java.util.Calendar.getInstance〇.getTime〇，并将计算结果传递给 out.print〇，

这与如下脚本程序的效果一样：

Today is

<%

out.print(java.util.Calendar.getlnstance().getTime());

%>

1. 5.2声明

可以声明能在JSP页面中使用的变量和方法。声明以“<%!”开始，并以“％>”结束。 例如，清单D.7的declarationTestjsp页面展示了一个JSP页面，该页面声明了一个名为 getTodaysDate 的方法。

清单 D.7 使用声明（declarationTestjsp)

<%!

public String getTodaysDate() {

return new java.util.Date();

%>

<!DOCTYPE html>

<html>

<headxtitle>Declarations</ title></head>

<body>

Today is <%=getTodaysDate()%>

</body>

</html>

在jsp页面中，一个声明可以出现在任何地方，并且一个页面可以有多个声明。

可以使用声明来重写JSP页面，实现类的init和destroy方法。通过声明jsplnit方法，来 重写init方法。通过声明jspDestroy方法，来重写destory方法。这两种方法说明如下。

• jsplnit。这种方法类似于javax.servletServlet的init方法。JSP页面在初始化时调 用jsplnit。不同于init方法，jsplnit没有参数。还可以通过隐式对象config访问 ServletConfig 对象。

• jspDestroy。这种方法类似于Servlet的destroy方法，在JSP页面将被销毁时调用。 清单D.8呈现的lifeCycle.jsp页面演示了如何重写jsplnit和jspDestroy。

清单 D.8 lifeCycle.jsp 页面

<%!

public void jsplnit() {

System, out .print In ("jsplnit

}

public void jspDestroy() {

System.out.printIn("jspDestroy

}

%>

<!DOCTYPE html>

<html>

<head><title>jsplnit and jspDestroy</title></head>

<body>

Overriding jsplnit and jspDestroy </body>

</html>

lifeCycle.jsp页面会被转换成如下的Servlet:

package org,apache-jsp; import j avax.servlet.\*; import j avax.servlet.http.\*;

import j avax.servlet.j sp.\*;

public final class lifeCycle\_jsp extends

org.apache.j asper.runtime.HttpJspBase

implements org.apache.j asper.runtime.JspSourceDependent {

public void jsplnit() {

System, out .printIn (M j splnit ■■■’’）；

}

public void jspDestroyO {

System.out.printIn(njspDestroy ••.”）；

}

private static final javax.servlet.jsp.JspFactory \_jspxFactory =  
j avax.servlet.j sp.JspFactory.getDefaultFactory();

private static java.util.Map<java.lang.String,java.lang.Long>

\_j spx\_dependants;

private j avax,el.ExpressionFactory \_el\_expressionfactory; private org.apache.tomcat.InstanceManager \_j sp\_instancemanager;

public java.util.Map<java.lang.String,java.lang.Long> getDependants() { return \_jspx\_dependants;

public void \_jsplnit () {

\_el\_expressionfactory =

\_j spxFactory.getJspApplicationContext( getServletConfig().getServletContext())

.getExpressionFactory();

\_jsp\_instancemanager =

org.apache.j asper.runtime.InstanceManagerFactory .getlnstanceManager(getServletConfig());

}

public void \_jspDestroy() {

}

public void \_jspService(final

j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest) request, final j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse) response) throws j ava.io.IOException, j avax.servlet.ServletException {

final javax.servlet.j sp.PageContext pageContext;

javax.servlet.[http.HttpSession](http://http.HttpSession) session = null;

final j avax.servlet.ServletContext application;

final j avax.servlet.ServletConfig config;

j avax.servlet.j sp.JspWriter out = null;

final java.lang.Object page = this;

javax.servlet.jsp.JspWriter \_jspx\_out = null;

javax. servlet. j sp. PageContext \_j spx\_page\_context = null;

try {

response.setContentType(ntext/htmlM);

pageContext = \_jspxFactory.getPageContext (this, request, response, null, true, 8192, true);

\_j spx\_page\_context = pageContext; application = pageContext.getServletContext(); config = pageContext.getServletConfig(); session = pageContext.getSession(); out = pageContext.getOut{);

\_jspx\_out = out;

out.write(M\nM); out.write(If<!DOCTYPE html>\nn); out .write (n<htinl>\nM);

out .write (M<headxtitle>j splnit and jspDestroy" + "</title></head>\nM); out.write(n<body>\nM);

out.write("Overriding jsplnit and jspDestroy\nM); out.write(M</body>\nM); out•write("</html>");

} catch (java.lang.Throwable t) { if (!(t instanceof

j avax.servlet.j sp.SkipPageException)){ out = \_jspx\_out;

if (out != null && out.getBufferSize () != 0)

try {

out.clearBuffer();

} catch (java.io.IOException e) {

}

if (\_jspx\_page\_context != null)

\_j spx\_page\_context.handlePageException(t);

}

} finally {

\_j spxFactory. releasePageContext (\_j spx\_page\_context);

注意生成的Servlet类中的jsplnit和jspDestroy方法。

现在可以通过如下URL访问lifeCycle.jsp:

<http://localhost:8080/j> spdemo/lifeCycle.j sp

第一次访问页面时，可以在控制台上看到“jsplnit...”，并且在Servlet/JSP容器关闭时看 到 “jspDestory".”。

1. 5.3禁用脚本元素

随着JSP 2.0对表达式语言的加强，推荐的做法是：在JSP页面中用EL访问服务器端对 象且不写Java代码。因此，从JSP2.0起，可以通过在部署描述符中 义一个scripting-invalid元素，来禁用脚本元素。

<j sp-config>

<j sp-property-group>

<url-pattern>\*•j sp</url-pattern>

<scripting-invalid>true</scripting-invalid〉

</j sp-property-group>

</j sp-config>

**D.6**动作

动作是第三种类型的语法元素，它们被转换成Java代码来执行操作，如访问一个Java对 象或调用方法。本节仅讨论所有JSP容器支持的标准动作。除了标准之外，还可以创建自定 义标签来执行某些操作。

下面是一些标准的动作。

1. 6.1 useBean

useBean将创建一个关联Java对象的脚本变量。这是早期分离表示层和业务逻辑的努力 之一。随着其他技术的发展，如自定义标签和表达语言，现在很少使用useBean方式。

清单D.9的useBeanTestjsp页面是一个示例，它创建一个java.util.Date实例，并赋值给 名为today的脚本变量，然后在表达式中使用它。

清单 D.9 useBeanTestjsp 页面 <!DOCTYPE html> <html>

<head>

<title>useBean</title>

</head>

<body>

<j sp:useBean id=11 today" class=" java ■ util ■ Date"/>

<%=today%>

</body>

</html>

在Tomcat中，上述代码会被转换为如下代码：

java.util.Date today = null;

today = (java.util.Date) spx\_page\_context•getAttribute("today”，

javax.servlet.j sp.PageContext.REQUEST\_SCOPE); if (today == null) {

today = new java.util.Date();

\_j spxjage\_context■setAttribute("today", today,

j avax.servlet.j sp.PageContext.REQUEST\_SCOPE);

}

访问这个页面，会输出当前的日期和时间。

1. 6.2 setProperty 和 getProperty

setProperty动作可对一个Java对象设置属性，而getProperty则会输出Java对象的一个属性。 清单D.11中的getSetPropertyTestjsp页面展示了如何设置和输出在清单D.10中定义的Employee 类实例的firstName属性。

清单 D.10 Employee 类

package j spdemo; public class Employee { private String id; private String firstName; private String lastName;

public String getldt) { return id;

}

public void setld(String id) { this.id = id;

}

public String getFirstName() {

return firstName;

public void setFirstName(String firstName) { this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) { this.lastName = lastName;

清单 D.11 getSetPropertyTest.jsp 页面

<!DOCTYPE html>

<html>

<head>

<title>getProperty and setProperty</title>

</head>

<body>

<j sp:useBean id=11 employee" class=M j spdemo. EmployeeM/>

< j sp: set Proper ty name=" employee” property=n firstName" value=’’Abigail" / > First Name: <j sp:getProperty name=,,employeen property=nfirstNameM/>

</body>

</html>

1. 6.3 include

include动作用来动态地引入另一个资源。可以引入另一个JSP页面，也可以引入一个

Servlet或一个静态的HTML页面。例如，清单D.12的jspIncludeTestjsp页面使用include动 作来引入menu.jsp页面。

清单 D.12 jspIncludeTestjsp 页面

<!DOCTYPE html>

<html>

<head>

<title>Include action</title>

</head>

<body>

<j sp:include page=Mj spf/menu.j spM>

<jsp:param name=MtextM value=MHow are you?M/>

</j sp:include>

</body>

</html>

这里，理解include指令和include动作非常重要。对于include指令，资源引入发生在页 面转换时，即当JSP容器把页面转换为生成的Servlet时。而include动作，资源引入发生在 请求页面时。因此，使用include动作是可以传递参数的，而include指令不支持。

第二个不同是，include指令对引入的文件扩展名不做特殊要求。但include动作，若引入 的文件需以JSP页面处理，则其文件扩展名必须是JSP。若使用jspf为扩展名，则该页面被 当作静态文件。

1. 6.4 forward

forward动作把当前页面转向到其他资源。下面的代码将从当前页转向到loginjsp页面。

<j sp:forward page=Mj spf/login.j spM>

<jsp:param name=,,textM value=MPlease login’’/〉

</j sp:forward〉

D.7错误处理

JSP提供了很好的错误处理能力。除了在Java代码中可以使用try语句，还可以指定一个 特殊页面一一当应用页面遇到未捕获的异常时，用户将看到一个精心设计的网页来解释发生 了什么，而不是用户无法理解的一条错误信息。

请使用page指令的isErrorPage属性（属性值必须为True)来标识一个JSP页面是错误

页面。清单D.13展示了一个错误处理程序。

清单 D.13 errorHandler.jsp 页面

<%@page isErrorPage=MtrueM%>

<!DOCTYPE html><html>

<headxtitle>Error</title></head>

<body>

An error has occurred. <br/>

Error message:

<%

out.println(exception.toString());

%>

</body>

</html>

其他需要防止未捕获的异常的页面必须使用page指令的errorPage属性，来指向错误处

理页面。例如，清单D.14中的buggy.jsp页面就使用了清单D.13的错误处理程序。

清单 D.14 buggy.jsp 页面

<%@page errorPage=,,errorHandler. j spM%>

Deliberately throw an exception

<%

Integer.parselnt("Throw me");

%>

运行的buggy.jsp页面会抛出一个异常。不过，我们不会看到由Servlet/JSP容器生成的错 误消息。相反，会看到errorHandler.jsp页面的内容。

**D.8**小结

JSP是构建在Java Web应用程序的第二种技术，是Servlet技术的补充，而不是Servlet技 术的替代。一个精心设计的Java Web应用程序会同时使用Servlet和JSP。

在本附录中，我们学习了 JSP是如何工作的，以及如何编写JSP页面。现在，我们已经 知道JSP的隐式对象，并能在JSP页面使用3个语法元素：指令、脚本元素和动作。

附录E 部署描述符

部署一个Servlet 3或Servlet 3.1应用程序是一件轻而易举的事。通过Servlet注解类型， 对于不太复杂的应用程序，可以部署没有描述符的Servlet/JSP应用程序。尽管如此，在需要 更加精细配置的情况下，仍然需要部署描述符。首先，部署描述符必须被命名为web.xml并 且位于WEB-INF目录下，Java类必须放置在WEB-INF/classes目录下，而Java类库则必须位 于WEB-INF/lib目录下。所有的应用程序资源必须包装成一个以war为后缀的jar文件。

本附录会讨论部署和部署描述符，这是一个应用程序的重要组成部分。

1. **1**概述

在Servlet 3之前，部署工作必然涉及部署描述符，即web.xml文件，我们在该文件中配 置应用程序的各个方面。但在Servlet 3中，部署描述符是可选的，因为我们可以使用标注来 映射一个URL模式的资源。不过，若存在如下场景，则依然需要部署描述符。

•需要传递初始参数给ServletContext。

•有多个过滤器，并要指定调用顺序。

•需要更改会话超时设置。

•要限制资源的访问，并配置用户身份验证方式。

清单E.1展示了部署描述符的框架。它必须被命名为web.xml且合并在应用目录的WEB-INF 目录下。

清单E.1部署描述符的框架

<?xml version="l. 0M encoding=,,ISO-885 9-l"?>

<web-app version=M3.1" xmlns="<http://xmlns.j> cp.org/xml/ns/j avaee"

xmlns :xsi=,,http: //www. w3 . org/2001/XMLSchema-instance,, xsi : schemaLocation=,,http: //xmlns . j cp. org/xml/ns/j avaee http: //xmlns .jcp.org/xml/ns/javaee/web-app\_3\_l .xsdM

[metadata-complete=w true 丨 false” ]

</web-app>

xsi:schemaLocation属性指定了模式文档的位置，以便可以进行验证。version属性指定了 Servlet规范的版本。

可选的metadata-complete属性指定部署描述符是否是完整的，若值为True，贝！JServlet/JSP 容器将忽略Servlet注解。若值为False或不存在，则容器必须检查类文件的Servlet注解，并 扫描web fragments文件。

web-app元素是文档的根元素，并且可以具有如下子元素：

* Servlet 声明。
* Servlet 映射。
* ServletContext的初始化参数。

•会话配置。

•监听器类。

•过滤器定义和映射。

* MIME类型映射。

•欢迎文件列表。

•错误页面。

* JSP特定的设置。
* JNDI设置。

每个元素的配置规则可见web-app\_3\_l.xsd文档，可以从如下网站下载：

<http://xmlns.jcp.org/xml/ns/javaee/web-app_3_l.xsd>

web-app\_3\_l.xsd包括另一种模式（webcommon\_3\_l.xsd)，其中包含了大部分信息。可从

如下网站下载：

http: //xmlns . j cp. org/xml/ns/ j avaee/web-common\_3\_l. xsd

webcommon\_3\_l.xsd包括以下两种模式：

* javaee\_7.xsd，定义了其他Java共享公共元素EE7的部署类型（EAR、JAR和RAR)。
* jsp\_2\_7.xsd，定义要素配置的一部分，JSP根据JSP2.3规范中的应用。

本节列出了在部署描述符中常见的Servlet和jsp元素，但不包括那些不在Servlet或JSP 规范中的Java EE元素。

1. 1.1核心元素

本节将详细介绍各重要元素的细节。web-app的子元素可以以任何顺序出现。某些元素， 如 session-config、jsp-config 和 login-config 只能出现一次，而另一些元素，如 Servlet、filter 和welcome-file-list可以出现很多次。

后续几个小节会分别描述元素下的一级元素。若要查找#<web-app>下的非 一级元素，请查找其父元素。例如，taglib元素在“jsp-config”下，而load-on-startup在Servlet

下。本节后续小节按字母顺序排序。

1. 1.2 context-param

可用context-param元素传值给ServletContext。这些值可以被任何Servlet/JSP页面读取。 context-param元素由名称/值对构成，并可以通过调用ServletContext的getlnitParameter方法 来读取。可以定义多个context-param元素，每个参数名在本应用中必须唯一。ServletContext. getInitParameterNames()方法会返回所有的参数名称。

context-param 元素必须包含一个 param-name 元素和一个 param-value 元素。param-name 定义参数名，而param-value定义参数值。另有一个可选的元素，即description元素，用来描

述参数。

下面是context-param元素的两个例子。

<context-param>

<param-name>location</param-name>

<param-value>localhost</param-value>

</context-param>

<context-param>

<param-name>port</param-name>

<param-value>8080</param-value>

附录E部署描述符

<description>The port number used</description> </context-param>

1. 1.3 distributable

若定义了 distributable元素，则表明应用程序已部署到分布式的Servlet/JSP容器。 distributable元素必须是空的。例如，下面是一个distributable例子。

〈distributable/〉

1. 1.4 error-page

error-page元素包含一个HTTP错误代码与资源路径或Java异常类型与资源路径之间的映 射关系。error-page元素定义容器在特定HTTP错误或异常时应返回的资源路径。

error-page元素由如下成分构成：

* error-code，指定一个HTTP错误代码。
* exception-type,指定Java的异常类型（全路径名称)。
* location，指定要被显示的资源位置。该元素必须以“/”开始。

下面的配置告诉Servlet/JSP容器，当出现HTTP 404时，显示位于应用目录下的error.html

页面。

<error-page>

<error-code>404</error-code>

<location>/error,html</location>

</error-page>

下面的配置告诉Servlet/JSP容器，当发生ServletException时，显示exception.html页面。

<error-page>

<exception-type>j avax.servlet.ServletException</exception-type> <location>/exception.html</location>

</error-page>

1. 1.5 filter

filter指定一个Servlet过滤器。该元素至少包括一个filter-name元素和一个filter-class元素。 此夕卜，它也可以包含以下元素：icon、display-name、discription、init-param 以及 async-supported。

filter-name元素定义了过滤器的名称。过滤器名称必须全局唯一。filter-class元素指定过滤

器类的全路径名称。可由init-param元素来配置过滤器的初始参数（类似于<〇»^\1^31"3111>)， 一个过滤器可以有多个init-param。

下面是 Upper Case Filter 和 Image Filter 这两个 filter 元素。

<filter>

<filter-name>Upper Case Filter</filter-name>

<filter-class>com.example.UpperCaseFilter</filter-class>

</filter>

<filter>

<filter-name>Image Filter</filter-name>

<filter-class>com.example.ImageFiIter</filter-class>

<init-param>

<param-name>frequency</param-name>

<param-value>190 9</param-value>

</init-param>

<init-param>

<param-name>resolution</param-name>

<param-value>1024</param-value>

</init-param>

</filter>

1. 1.6 filter-mapping

过滤器映射元素是指定过滤器要被映射到的一个或多个资源。过滤器可以被映射到 Servlet或者URL模式。将过滤器映射到Servlet会致使过滤器对该Servlet产生作用。将过滤 器映射到URL模式，则会使其对所有URL与该URL模式匹配的资源进行过滤。过滤的顺序 与过滤器映射元素在部署描述符中的顺序一致。

过滤器映射元索中包含一个filter-name元素和一个url-pattem元素或者servlet-name元素。 filter-name元素的值必须与利用filter元素声明的某一个过滤器名称相匹配。

下面的例子中是两个过滤器元素和两个过滤器映射元素。

<filter>

<filter-name>Logging Filter</filter-name>

<filter-class>com.example.LoggingFilter</filter-class>

</filter>

<filter>

<filter-name>Security Filter</filter-name>

<filter-class>com.example.SecurityFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>Logging Filter</filter-name>

<servlet-name>FirstServlet</servlet-name>

</filter-mapping>

<filter-mapping>

<filter-name>Security Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

1. 1.7 listener

listener元素用来注册一个侦听器。其子元素listener-class包含监听器类的全路径名。如 下是一个示例：

<listener>

<listener-class>com.example.AppListener</listener-class>

</listener>

1. 1.8 locale-encoding-mapping-list 禾口 locale-encoding-mapping

locale-encoding-mapping-list 元素包含了一个或多个 locale-encoding-mapping 元素。每个 locale-encoding-mapping定义了 locale以及编码的映射，分另!J用locale以及encoding元素定义。 locale元素的值必须是在ISO 639中定义的语言编码，如en，或者是采用“语言编码\_国家编 码”格式，如en\_US。其中，国家编码值必须在ISO 3166中定义。

如下是一个示例：

<locale-encoding-mapping-list>

<locale-encoding-mapping>

<locale>ja</locale>

<encoding>Shift\_JIS</encoding>

</locale-encoding-mapping>

</locale-encoding-mapping-list>

1. 1.9 login-config

login-config 元素包括 auth-method、realm-name 以及 form-login-config 元素，每个元素都

是可选的。

auth-method 元素定义了认证方式，可选值为 BASIC、DIGEST、FORM 和 CLIENT-CERT。 realm-name元素定义了用于BASIC以及DIGEST认证方式的realm名称。

E.1概述

form-login-config则定义了用于FORM认证方式的登录页面和失败页面。若没有采用 FORM认证方式，则该元素被忽略。

form-login-config 元素包括 form-login-page 和 form-error-page 两个子元素。其中，form- login-page配置了显示登录页面的资源路径，路径为应用目录的相对路径，且必须以“/”开 始。form-error-page则配置了登录失败时显示错误页面的资源路径。同样，路径为应用目录的 相对路径，且必须以‘7”开始。

下面是一个示例：

<login-config>

<auth-method>DIGEST</auth-method>

<realm-name>Members Only</realm-name>

</login-config>

另一个示例如下：

<login-config>

<auth-method>FORM</auth-method>

<form-login-config>

<form-login-page>/loginForm.j sp</form-login-page>

<form-error-page>/errorPage.j sp</form-error-page>

</form-login-config>

</login-config>

E.1.10 mime-mapping

mime-mapping元素用来把一个MIME类型映射到一个扩展名。该元素由一个extension 元素和一个mime-type元素组成。示例如下：

<mime-mapping>

<extension>txt</extension>

<miitie-type>text/plain</inime-type>

</mime-mapping>

E.1.11 security-constraint

security-constraint元素允许对一组资源进行限制访问。

security-constraint元素有如下子元素：一个可选的display-name元素、一个或多个 web-resource-collection 元素、可选的 auth-constraint 元素和一个可选的 user-data-constraint 元素。

web-resource-collection元素标识了一组需要进行限制访问的资源集合。这里，你可以定义

URL模式和所限制的HTTP方法。如果没有定义HTTP方法，则表示应用于所有HTTP方法。

auth-constraint元素指明哪些角色可以访问受限制的资源集合。如果没有指定，贝!J应用于

所有角色。

user-data-constraint元素用于指示在客户端和Servlet/JSP容器传输的数据是否保护。

web-resource-collection 元素包含一个 web-resource-name 元素、一个可选的 description 元 素、零个或多个url-pattem元素，以及零个或多个http-method元素。

web-resource-name元素指定受保护的资源名称。

http-method 元素指定 HTTP 方法，如 GET、POST 或 TRACE。

autii-constraint元素包含一个可选的description元素、零个或多个role-name元素。role-name

元素指定角色名称。

user-data-constraint 元素包含一个可选的 description 元素和一个 transport-guarantee 元素。 transport-guarantee 元素的取值范围如下：NONE、INTEGRAL 或 CONFIDENTIAL。NONE 表

示该应用程序不需要安全传输保障。INTEGRAL意味着服务器和客户端之间的数据在传输过 程中不能被篡改。CONFIDENTIAL意味着必须加密传输数据。大多数情况下，安全套接字层 (SSL)会应用于 INTEGRAL 或 CONFIDENTIAL。

下面是一个例子：

<security-constraint>

<web-resource-collection>

<web-resource-name>Members Only</web-resource-name> <url-pattern>/members/\*</url-pattern>

</web-resource-collection>

<auth-constraint>

<role-name>payingMember</role-name>

</auth-constraint>

</security-constraint>

<login-config>

<auth-method>Digest</auth-method>

〈realm—name>Digest Access Authentication</realm—name〉

</login-config>

E.1.12 security-role

security-role元素声明用于安全限制的安全角色。这个元素有一个可选的description元素 和role-name元素。下面是一个例子：

<security-role>

<role-name>payingMember</role-name>

</security-role>

E.1.13 Servlet

Servlet元素用来配置Servlet，包括如下子元素：

•一个可选的icon元素。

參一个可选的description元素。

•可选的display-name元素。

* 一个 servlet-name 元素。
* 一个 servlet-class 元素或一个 jsp-file 元素。

•零个或更多的init-param元素。

* 一个可选的load-on-startup元素。

•可选的run-as元素。

•可选的enabled元素。

•可选的 async-supported 元素。

•可选的 multipart-config 元素。

•零个或多个security-role-ref元素。

一个Servlet元素至少必须包含一个servlet-name元素和一个servlet-class元素，或者一个 servlet-name 元素和一个 jsp-file 元素。

servlet-name元素定义的Servlet名称在应用程序中必须是唯一的。

servlet-class元素指定的类名为全路径名。

jsp-file元素指定JSP页面的路径，该路径是应用程序的相对路径，必须以“/”开始。

init-param的子元素可以用来传递一个初始参数给Servlet。init-param元素的构成同 context-param〇

可以使用load-on-startup元素在当Servlet/JSP容器启动时自动加载Servlet。加载一个

Servlet是指实例化Servlet和调用它的init方法。使用此元素可以避免由于加载Servlet而导致 对第一个请求的响应延迟。如果该元素指定了 jsp-file元素，则JSP文件被预编译成Servlet， 并加载该Servlet。

load-on-startup可以指定用一个整数值来指定加载顺序。例如，如果有两个Servlet且都包 含一个load-on-startup元素，则值小的Servlet优先加载。若没有指定值或值为负数，则由Web 容器决定如何加载。若两个Servlet具有相同的load-on-startup值，则加载Servlet的顺序不能

确定。

run-as用来覆盖调用EJB的安全标识。角色名是当前Web应用程序定义的安全角色之一。

security-role-ref元素将在调用Servlet的isUserlnRole方法时角色名映射到应用程序定义 的安全角色。security-role-ref元素包含一个可选的description元素、一个role-name元素和一 个 role-link 元素。

role-link元素用于将安全角色映射到一个已定义的安全角色，它必须包含一个在security-role 元素中定义的安全角色。

async-supported元素是一个可选的元素，其值可以是True或False。它表示Servlet是否

支持异步处理。

enabled元素也是一个可选的元素，它的值可以是True或False。设置此元素为False，则 禁用这个Servlet。

例如，映射安全角色“PM”与角色名字“payingMember”的配置如下：

<security-role-ref>

<role-name>PM</role-name>

<role-link>payingMember</role-1ink>

</security-role-ref>

这样，若属于 payingMember 角色的用户调用 Servlet 的 isUserlnRole (“payingMember”)

方法，则结果为真。

下面是Servlet元素的两个例子：

<servlet>

<servlet-name>UploadServlet</servlet-name>

<servlet-class>com.brainysoftware.UploadServlet</servlet-class> <load-on-startup>10</load-on-startup>

</servlet>

<servlet>

<servlet-name>SecureServlet</servlet-name>

<servlet-class>com.brainysoftware.SecureServlet</servlet-class> <load-on-startup>20</load-on-startup>

</servlet>

E.1.14 servlet-mapping

servlet-mapping元素将一个Servlet映射到一个URL模式。该元素必须有一个servlet-name 元素和url-pattem元素。

下面的 servlet-mapping 元素映射一个 Servlet 到/first。

<servlet>

<servlet-name>FirstServlet</servlet-name>

<servlet-class>com.brainysoftware.FirstServlet</servlet-class> </servlet>

<servlet-mapping>

<servlet-name>FirstServlet</servlet-name>

<url-pattern>/first</url-pattern>

</servlet-mapping>

E.1.15 session-config

session-config元素定义了用于javax.servlethttp.HttpSession实例的参数。此元素可包含一 个或更多的以下内容：session-timeout、cookie-config 或 tracking-mode。

session-timeout元素指定会话超时间隔（分钟)。该值必须是整数。如果该值是零或负数， 则会话将永不超时。

cookie-config元素定义了跟踪会话仓!J建的cookie的配置。

tracking-mode元素定义了跟踪会话模式，其有效值是COOKIE、URL或SSL。

下面定义的session-config元素使得应用的HttpSession对象在12分钟不活动后失效。

<session-config>

<session-timeout>12</session-timeout> </session-config>

E.1.16 welcome-file-list

welcome-file-list元素指定当用户在浏览器中输入的URL不包含一个Servlet名称或JSP 页面或静态资源时所显示的文件或Servlet。

welcome-file-list元素包含一个或多个welcome-file元素。welcome-file元素包含默认的文 件名。如果在第一个welcome-file元素指定的文件没有找到，则Web容器将尝试显示第二个， 直到最后一个。

下面是一个welcome-file-list元素的例子。

<welcome-file-list>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.html</welcome-file>

<welcome-file>index.j sp</welcome-file>

</welcome-file-list>

下面的示例，第一个welcome-file元素指定了一个在应用程序目录下的index.html;第二 个welcome-file为servlet目录下的欢迎文件。

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>servlet/welcome</welcome-file>

</welcome-file-list>

E.1.17 JSP-Specific Elements

<\^哪口>元素下的jsp-config元素，可以指定JSP配置。它可以具有零个或多个标签taglib 和零个或多个jsp-property-group元素。下面首先介绍taglib元素，然后介绍jsp-property-group

元素。

E.1.18 taglib

taglib元素定义了 JSP定制标签库。taglib元素包含一个taglib-uri元素和taglib-location 元素。taglib-uri元素定义了 Servlet/JSP应用程序所用的标签库的URL其值相当于在部署描 述符路径。

taglib-location元素指定TLD文件的位置。

下面是一个taglib元素的例子。

<j sp-config>

<taglib>

<taglib-uri>

<http://brainysoftware.com/taglib/complex>

</taglib-uri>

<taglib-location>/WEB-INF/j sp/complex,tld </taglib-location>

</taglib>

</j sp-config>

E.1.19 jsp-property-group

jsp-property-group中的元素可为一组JSP文件统一配置属性。使用<』8口^«^117甘〇叩>子

元素可做到以下几点：

•指7K EL显7K是否忽略。

•指示脚本元素是否允许。

•指明页面的编码信息。

•指示一个资源是JSP文件（XML编写）。

•预包括和代码自动包含。 jsp-property-group包含如下子元素：

* 一个可选的description元素。

參一个可选的display-name元素。

•一个可选的icon元素。

•一个或多个url-pattem元件。

•一个可选的el-ignored元素。

* 一个可选的page-encoding元素。
* 一个可选的 scripting-invalid 元素。

•一个可选的is-xml元素。

•零个或多个include-prelude元素。

•零个或多个include-code元素。

url-pattem元素用来指定可应用相应属性配置的URL模式。

el-ignored元素值为True或False。True值表示在E配URL模式的JSP页面中，EL表达

式无法被计算，该元素的默认值是False。

page-encoding元素指定JSP页面的编码。page-encoding的有效值和页面的pageEncoding

的有效值相同。若page-encoding指定值与匹配URL模式的JSP页面中的pageEncoding属性 值不同时，则会产生一个转换时错误。同样，若page-encoding指定值与XML文档声明的编 码不同，也会产生一个转换时错误。

scripting-invalid元素值为True或False〇 True值是指匹配URL模式的JSP页面不支持<% scripting %>语法。scripting-invalid 元素的默认值是 False。

is-xml元素值为True或False，True表示四配URL模式的页面是JSP文件。

include-prelude元素值为相对于Servlet/JSP应用的相对路径。若设置该元素，则匹配URL

模式的JSP页面开头处会自动包含给定路径文件（同include指令)。

include-coda元素值为相对于Servlet/JSP应用的相对路径。若设置该元素，则匹配URL 模式的JSP页面结尾处会自动包含给定路径文件（同include指令)。

在下面的例子中，jsp-property-group配置所有的JSP页面无法执行EL表达式。

<j sp-config>

<j sp-property-group>

<url-pattern>\*.j sp</url-pattern>

<el-ignored>true</el-ignored>

</j sp-property-group>

</j sp-config>

在下面的例子中，jsp-property-group配置所有的JSP页面不支持<% scripting %>语法。

<j sp-config>

<j sp-property-group>

<url-pattern>\*.j sp</url-pattern>

<scripting-invalid>true</scripting-invalid>

</j sp-property-group>

</j sp-config>

1. **2**部署

从Servlet 1.0开始，可以很方便地部署一个Servlet/JSP应用程序。仅需要将应用原始目 录结构压缩成一个WAR文件。可以在JDK中使用jar工具或使用流行的工具，如WinZip。 需要确保压缩文件有.war扩展名。如果使用WinZip，则在压缩完成后重命名文件。

war文件必须包含所有库文件、类文件、HTML文件、JSP页面、图像文件以及版权声明 (如果有的话）等，但不包括Java源文件。任何人都可以获取一个war文件的副本，并部署

到一个Servlet/JSP容器上。

E.3 Web Fragment

Servlet 3添加了 web fragment特性，用来为已有的Web应用部署插件或框架。web fragment 被设计成部署描述符的补充，而无需编辑web.xml文件。一个Web fragment基本上包含了常 用的Web对象，如Servlet，过滤器和监听器，其他资源如JSP页面和静态图像的包文件（jar 文件)。一个web fragment也可以有一个描述符，类似的部署描述符的XML文档。web fragment 描述符必须命名为web-fragment.xml，并位于包的META-INF目录下。一个web fragment描 述符可能包含可出现在部署描述符web-app元素下的任意元素，再加上一些web fragment的 特定元素。一个应用程序可以有多个web fragment。

清单E.2显示了 web fragment描述符，以黑体形式突出显示与部署描述符之间的不同内 容。在web fragment的根元素必须是web-fragment元素，其可以有metadata-complete属性。 如果metadata-complete属性的值为True，则包含在web fragment中所有的类注释将被跳过。

清单**E.2 web fragment.xml**文件的框架

<?xml version="l.0" encodings"ISO-8859-1"?>

<web-fragment version="3.1M xmlns=,,http: //xmlns . j cp . org/xml/ns/j avaeen xmlns :xsi=,,http: //www. w3 . org/2001/XMLSchema-instance,, xsi : schemaLocation=,,http: / /xmlns . j cp. org/xml/ns/j avaee <http://xmlns.j> cp.org/xml/ns/j avaee/web-fragment\_3\_l.xsd" [metadata-complete=Mtrue|false"]

</web-fragment>

作为一个例子，在fragmentdemo应用程序中包含的fragmentjar文件是一个web fragment。 该jar文件己经导入到WEB-INF/lib目录下。本实例的重点不在于ftagmentdemo，而是web ftagment 项目。该项目包含一个 Servlet (fragment.servletFragmentServlet，见清单 E.3 )和 webfragment.xml 文件（见清单E.4)。

清单 E.3 FragmentServlet 类

package fragment.servlet; import java■io■工OException;

j ava.io.PrintWriter ; j avax.servlet.ServletException; j avax.servlet.[http.HttpServlet](http://http.HttpServlet); j avax.servlet.[http.HttpServletRequest](http://http.HttpServletRequest); j avax.servlet.[http.HttpServletResponse](http://http.HttpServletResponse);

public class FragmentServlet extends HttpServlet {

import

import

import

import

import

private static final long serialVersionUID = 94〇L;

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IQException {

response.setContentType("text/html"); PrintWriter out = response.getWriter(); out.println("A plug-in")；

清单 E.4 webfragment.xml 文件

<?xml version="l.0" encoding="ISO-8859-1"?>

<web-f ragment version="3.1" xmlns=nhttp : //xmlns . j cp . org/xml/ns/j avaee1' xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi:schemaLocation="<http://xmlns.j> cp.org/xml/ns/j avaee <http://xmlns.j> cp.org/xml/ns/j avaee/web-fragment\_3\_l. xsd">

<servlet>

<servlet-name>FragmentServlet</servlet-name>

<servlet-class>fragment.servlet.FragmentServlet</servletclass> </servlet>

<servlet-mapping>

<servlet-name>FragmentServlet</servlet-name>

<url-pattern>/fragment</url-pattern〉

</servlet-mapping>

</web-fragment〉

FragmentServlet是发送一个字符串到浏览器的一个简单的Servlet。web-fragment.xml文件注 册并映射该Servlet。fragment.jar文件结构如图E.1所示。

**&** fragment J **&** servlet

Eli FragmentServlet. cl ass **&** META-INF

**®** web-fragment.xml

图E. 1 fragment.jar文件结构

使用如下URL调用测试该Servlet。

<http://localhost:8080/fragmentdemo/fragment>

可以看到Fragment Servlet的输出。

**E.4**小结

本附录介绍了如何配置和部署Servlet/JSP应用程序。本附录首先介绍一个典型应用的目 录结构，然后详细阐释了部署描述符。

发布一个应用程序有两种方式：一种是以目录结构的形式；另一种是打包成一个单一的 war文件进行部署。

Spring MVC学习指南 (第2版）

作为当今业界最主流的Web开发框架，Spring MVC已经成为当前最热门的开发技能，同时也广泛用 于桌面开发领域。

本书重在讲述如何通过Spring MVC来开发基于Java的Web应用。

本书包括以下内容：

■ Spring和Spring MVC简介;

■模型2和MVC模式；

■转换器、格式化和验证器; ■ JSTL；

■上传文件和下载文件。

■数据绑定和表单标签库；

-表达式语言；

■国际化；

■丰富的示例可供读者练习和参考。

本书所有示例应用可以通过如下链接下载：

<http://books.brainysoftware.com/download> 和 http://www.epubit.com.cn。

作者简介

Paul Deck是一位资深的Spring框架开发人员，他还是《How Tomcat Works》一书的合著者。

ISBN 978-7-115-44759-3

4：

„人民邮电出版社 **B** [www.epubit.com.cn](http://www.epubit.com.cn)



异步社区 [www.epubit.com.cn](http://www.epubit.com.cn) 新浪微博@人邮异步社区 投稿/反馈邮箱 [contact@epubit.com.cn](mailto:contact@epubit.com.cn)



9 "7 8 7 1 1 5 11 4 A 7 5 9 3n>

分类建议：计算机/$欠件开发/Java 人民邮电出版社网址：[www.ptpress.com.cn](http://www.ptpress.com.cn)

IDEA默认快捷方式

1. Ctrl + Space   
完成类、方法、变量名称的自动输入,这个快捷键是我最经常使用的快捷键了，它可以完成类、方法、变量名称的自动录入，很方便   
2. Ctrl + N（Ctrl + Shift + N）   
跳转到指定的java文件（其它文件）这个功能很方便，至少我不用每回都在一长串的文件列表里找寻我想要编辑的类文件和jsp文件了   
3. Ctrl + B   
跳转到定义处这个就不用多说了，好象是个IDE就会提供的功能   
4. Ctrl + Alt + T   
用\*来围绕选中的代码行（ \* 包括if、while、try catch等）这个功能也很方便，把我以前要做的：①先写if-else，②然后调整代码的缩进格式，还要注意括号是否匹配了，现在用这个功能来做，省事多了（不过让我变得越来越懒了）   
5. Ctrl + Alt + B   
跳转到方法实现处这个也算是很普遍的功能了，就不多说了。   
6. Ctrl + W   
按一个word来进行选择操作在IDEA里的这个快捷键功能是先选择光标所在字符处的单词，然后是选择源   
代码的扩展区域。举例来说，对下边这个语句java.text.SimpleDateFormat formatter = new java.text.SimpleDateFormat("yyyy-MM-dd HH:mm");当光标的位置在双引号内的字符串中时，会先选中这个字符串，然后是等号右边的表达式，再是整个句子。我一般都是在对代码进行重新修改的时候使用   
它来选择出那些长长的复合表达式，很方便：）   
7. Shift + F1   
在浏览器中显示指定的java docs,这个也应该是几乎所有的java ide都提供的功能，就不多说了。   
8. Ctrl + Q   
在editor window中显示java docs这个功能很方便--因为有时仅仅是忘记了自己编写的方法中的某个参数的含义，此时又不想再起一个浏览器来查看java doc，此时这个功能的好处就体现出来了   
9. Ctrl + /   
注释/反注释指定的语句,这个功能很象PB中提供的一个功能，它可以注释和反注释你所选择的语句（使用单行注释符号"//"），你也可以用Ctrl + Shift + / 来进行多行语句的注释（即使用多行注释符号"/\* ... \*/"）   
10. F2/Shift + F2   
跳转到下/上一个错误语句处IDEA提供了一个在错误语句之间方便的跳转的功能，你使用这个快捷键可以快捷在出错的语句之间进行跳转。   
11. Shift + F6   
提供对方法、变量的重命名对IDEA提供的Refector功能我用得比较少，相比之下这个功能是我用得最多的了。对于这个功能没什么可说的了，确实很方便，赶快试一试吧。   
12. Ctrl + Alt + L   
根据模板格式化选择的代码,根据模板中设定的格式来format你的java代码，不过可惜的是只对java文件有效   
13. Ctrl + Alt + I   
将选中的代码进行自动缩进编排这个功能在编辑jsp文件的时候也可以工作，提供了一个对上边格式化代码功能的补充。   
14. Ctrl + Alt + O   
优化import自动去除无用的import语句，蛮不错的一个功能。   
15. Ctrl + ]/[   
跳转到代码块结束/开始处,这个功能vi也有，也是很常用的一个代码编辑功能了。   
16.Ctrl+E   
可以显示最近编辑的文件列表   
17.Shift+Click   
可以关闭文件   
18.Ctrl+Shift+Backspace   
可以跳转到上次编辑的地方   
19.Ctrl+F12   
可以显示当前文件的结构   
20.Ctrl+F7   
可以查询当前元素在当前文件中的引用，然后按F3可以选择   
21.Ctrl+Shift+N   
可以快速打开文件   
22.Alt+Q   
可以看到当前方法的声明   
23.Ctrl+P   
可以显示参数信息   
25.Alt+Insert   
可以生成构造器/Getter/Setter等   
26.Ctrl+Alt+V   
可以引入变量。例如把括号内的SQL赋成一个变量   
27.Alt+Up and Alt+Down   
可在方法间快速移动   
28.Alt+Enter   
可以得到一些Intention Action，例如将”==”改为”equals()”   
29.Ctrl+Shift+Alt+N   
可以快速打开符号   
30.Ctrl+Shift+Space   
在很多时候都能够给出Smart提示   
31.Alt+F3   
可以快速寻找   
32.Ctrl+O   
可以选择父类的方法进行重写   
33.Ctrl+Alt+Space   
是类名自动完成   
34. Ctrl+J   
Live Templates!   
35.Ctrl+Shift+F7   
可以高亮当前元素在当前文件中的使用   
30.Ctrl+Alt+Up /Ctrl+Alt+Down   
可以快速跳转搜索结果   
31.Ctrl+Shift+J   
可以整合两行   
32.Alt+F8是计算变量值   
  
Ctrl+D 复制上一行或复制选定   
Ctrl+Alt+L 格式化代码   
Alt+Shift+Insert 列编辑   
  
装上UpperLowerCapitalize后   
Alt+P // to uppercase   
Alt+L // to lowercase 

Alt+C // 首字母大写