# 第5章 Spring MVC

本章学习内容

- Spring MVC 简介；
- Spring MVC 程序运行原理；
- Spring MVC 的体系结构；
- 基于注解方式配置控制器；
- Spring MVC 注解详解。

## 5.1 Spring MVC 简介

前面的章节介绍了 Spring IoC、Spring AOP 和 Spring 对 JDBC 的支持。Spring 框架可以说是企业开发中技术体系最全面的一个框架，围绕着 Spring Framework 已经形成了完整的技术体系。目前，基于 Web 的 MVC 框架非常多，发展也很快，如 JSF、Struts 1、Struts 2 和 Spring MVC 等。Struts 1.x 框架在 2001 年出现后成为主流，之后陆续出现了 Struts 2 和 JSF 等框架。从架构上说，Struts 2 是一款非常优秀的软件，几乎成为 Java EE 开发的 MVC 框架的事实标准。但是由于 Struts 2 团队基本不再对其进行更新，只发布补丁，随着时间的推移，特别是 2010 年之后，Struts 2 陆续曝出了多个漏洞。很多网络安全和开发团队都给出了类似的建议：鉴于 Struts 2 至今为止已经多次曝出严重的高危漏洞，如果不是必要，建议开发者以后考虑采用其他类似的 Java 开发框架。所以广大的开发者将目光移向了 Spring MVC。Spring MVC 是后起之秀，从应用上来说要复杂一些，但是它基于 Spring 进行开发，继承了 Spring 的优秀血脉，所以一跃成为采用率最高的 Java EE Web MVC 框架。


视频讲解

## 5.2 第一个 Spring MVC 案例

首先通过一个简单的案例来了解 Spring MVC。

该案例的实现步骤如下。

（1）获取 Spring 框架的 JAR 库文件。

截至 2020 年 5 月，Spring Framework 已发布的稳定版本（GA 版）为 Spring 5.2.6 版，如图 5-1 所示。Spring 官方网站改版后，建议通过 Maven 和 Gradle 下载，若不使用 Maven 和 Gradle 开发项目，可以通过 Spring Framework 官网直接下载，下载路径为 http://repo.springsource.org/libs-release-local/org/springframework/spring。

对于已下载的 ZIP 文件，可以直接解压缩在本地文件夹。由于 Spring 框架的 JAR 包的深度特别深，造成解压后的文件夹名字超过了 WinRAR 或 WinZIP 软件的最长允许范围，在 Windows 操作系统下可能会导致解压失败，如果解压过程中出现问题，可以使用 7ZIP 软件来解压。

解压完成后，在文件夹中找到 libs 文件夹，有 63 个后缀名为 jar 的文件。仔细观察可以发现，每个主题有 3 个文件，例如，核心包文件为 spring-core-5.2.26.RELEASE.jar、

spring-core--5.2.26.RELEASE-javadoc.jar 和 spring-core--5.2.26.RELEASE-sources.jar。其中文件名结尾为 javadoc 的文件是 Java 根据注释自动生成的文档，而文件名结尾为 sources 的文件则是源代码文件。可以将所有的 JAR 文件导入到 Web 项目中，最简单的方法是直接复制到项目的 WebRoot 的 lib 文件夹下。

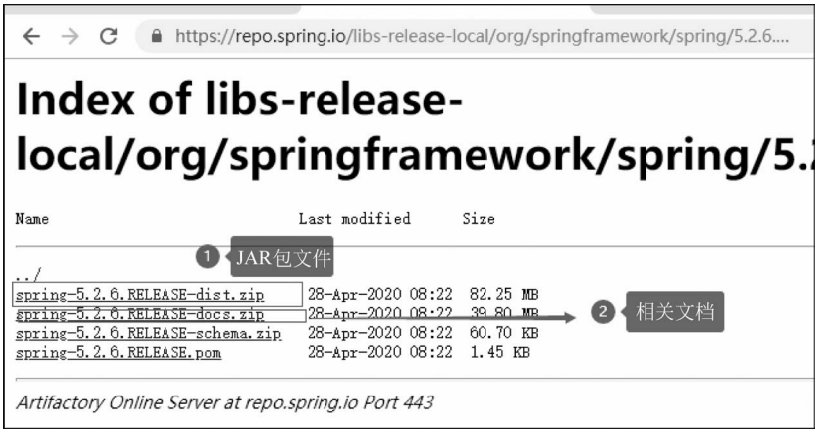


图 5-1　Spring 框架的下载

如果使用 MyEclipse2014 开发工具，可以使用 MyEclipse 内置的 Spring 包，只是版本稍低一些，不过并不影响学习和使用。Spring 5.x 的主要升级在于集成了 Spring Boot 模块，在启动方式上有很大的差别。对于初学者来说 Spring 3.x 更容易上手，所以本书后面依然采用 MyEclipse2014+Spring3.x 的方式来入门。虽然工具和版本更新换代会很快，但是 Spring 框架的思想和原理是稳定的，熟练掌握之后可以很快地完成版本切换。

（2）在 Web 项目中加入 Spring 框架。

通过 MyEclipse2014 新建一个 Web 项目，项目名为 ssmBook_ch6。在新建的 Web 项目中加入 Spring 框架，同时选中 Spring Web 模块以便支持 Spring MVC，如图 5-2 所示。

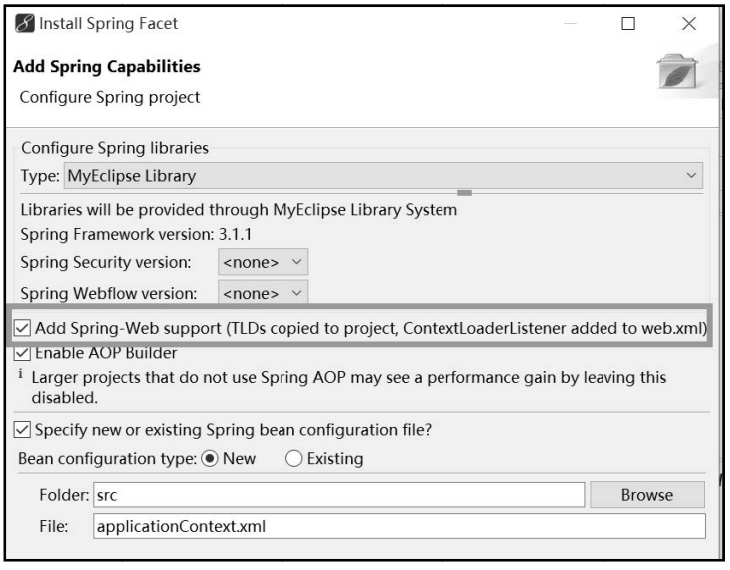


图 5-2　在 MyEclipse 中添加 Spring MVC

（3）配置 Dispatchservlet 和对应的 Servlet。

接下来在项目的 web.xml 配置文件中加入 DispatchServlet 的配置。

Servlet 的名字在这里命名为 springapp，也可以自由定义，但是后续的定义都要修改成对应自定义的名字，如图 5-3 所示。由于 Spring MVC 采用了约定优先于配置的方式，它会将此处的 Servlet 名字加上-servlet.xml 的后缀，以此来查找一个名为 springapp-servlet.xml 的配置文件。

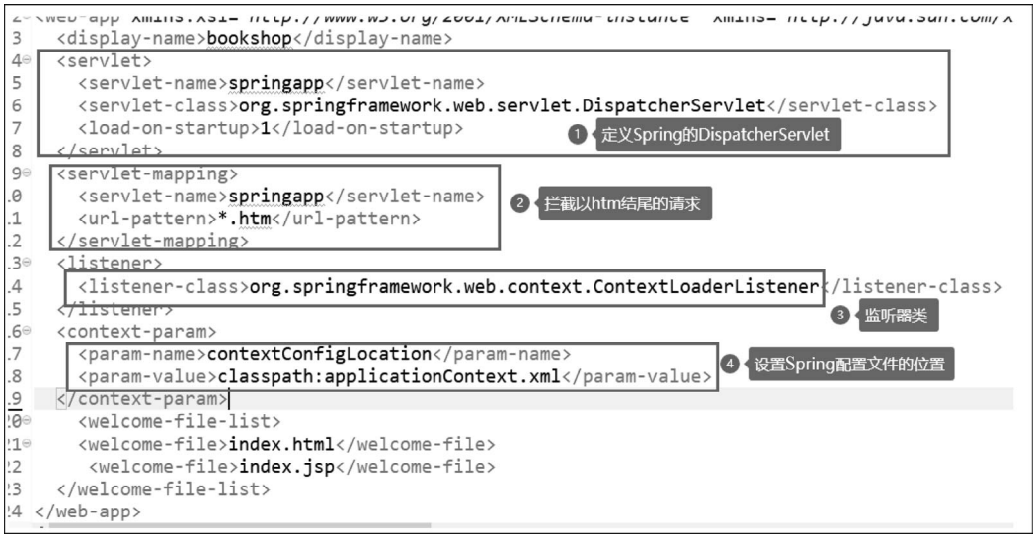


图 5-3　web.xml 配置文件

在与 web.xml 文件相同的位置上，新建一个名为 springapp-servlet.xml 的 xml 文件。该 springapp-servlet.xml 配置文件中定义了用户请求的路径和对应的控制器的映射关系，如图 5-4 所示。

```
<!-- 定义用户请求路径和对应的响应处理类之间的关系 -->
 <bean name="/hello.htm"
 class="com.ssmbook2020.web.ch5.HelloController" />
```

图 5-4　springapp-servlet.xml 配置文件

在这个配置文件中，核心的语句是

`<bean name="/hello.htm" class="org.ssmbook2020.web.ch5.HelloController"/>`。

当用户请求的路径是 hello.htm 时，它告诉 spring 用对应包的 HelloController 类来处理用户的请求。该语句用来取代以前纯 servlet 开发时 servlet 和 url 之间的关系映射。

（4）创建控制器类。

接下来创建 HelloController 的类，它的作用类似于以前的 Servlet。

代码如下所示：

```
//代码清单 5-1 HelloController.java
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
```

```
public class HelloController implements Controller {
    //返回 ModelAndView 对象
    public ModelAndView handleRequest(HttpServletRequest request,
           HttpServletResponse response)
           throws ServletException, IOException{
        //向 Request 域中放入一条信息，给前端 jsp 用
        request.setAttribute("message", "hello,Spring MVC");
        //返回 jsp 的路径
        return new ModelAndView("hello.jsp");
    }
}
```

这个控制器类比原始的 Servlet 更简洁，它继承了 Controller 接口，并实现了 handleRequest 方法。handleRequest 方法的两个参数就是原始的 HttpServletRequest 和 HttpServletResponse。为了演示效果，使用 Request 放入一个字符串信息，然后在 hello.jsp 页面中显示出这个字符串。

（5）建立 jsp 文件。

在 WebRoot 文件夹下新建 hello.jsp 文件，其内容如下。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
  <head>
    <title>hello jsp 页面</title>
  </head>
  <body>
     显示服务器信息如下:${requestScope.message }
  </body>
</html>
```

该步骤的目的是显示HelloController类中放入的信息，可以直接用EL表达式显示出来。

（6）部署运行。

将项目部署到 Tomcat 服务器上，启动 Tomcat 服务器，在浏览器中输入地址 http://localhost:8080/ssmBook_ch6/hello.htm，结果如图 5-5 所示。

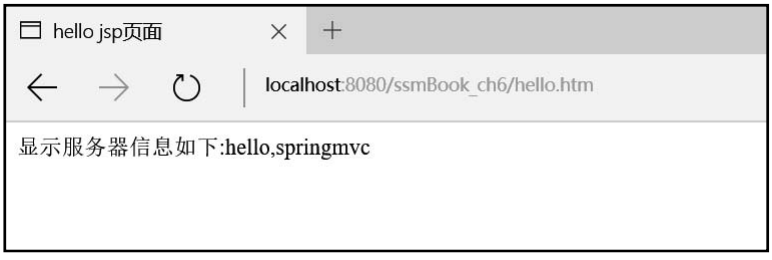


图 5-5　第一个 Spring MVC 的运行结果

# 5.3 Spring MVC 的工作原理与体系结构

## 5.3.1 Spring MVC 程序运行原理

通过第一个 Spring MVC 程序来介绍 Spring MVC 的运行流程。

（1）用户通过浏览器发出请求。

（2）在 web.xml 中 DispatchServlet 拦截*.htm 的请求。

（3）在与 web.xml 相同的路径下查找该 Servlet 对应的 Spring 配置文件，此案例中为 springapp-servlet.xml。

（4）根据 springapp-servlet.xml 配置文件中的 BeanName，找到对应的处理请求的类。在此案例中，用 HelloController 类来响应 hello.htm 请求。

（5）在 HelloController 类中的方法 `handleRequest`，它的作用类似于纯 Servlet 中的 doGet 或者 doPost。注意它的返回值是一个 Spring MVC 中的对象 ModelAndView，这个对象可以用来封装模型和视图。Hello.jsp 是默认的 jsp 页面的名字。这里直接返回页面的名字是不可取的，本章后面将会介绍更合理的方式。

图 5-6 为 Spring MVC 的工作流程图。

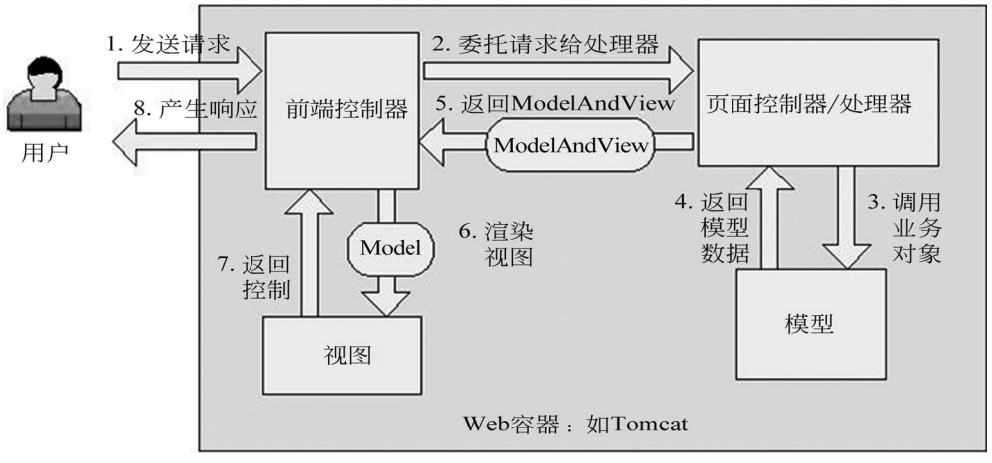


图 5-6 Spring MVC 工作流程图

## 5.3.2 视图解析器

在代码清单 5-1 的 HelloController 类中，handleRequest 方法返回的是一个 jsp 页面的名字。在实际案例中会改为如下的代码。

```
//代码清单 5-2  HelloController.java
package com.ssmbook2020.web.ch5;
import org.springframework.web.servlet.ModelAndView;
```

```
import org.springframework.web.servlet.mvc.Controller;

public class HelloController implements Controller {

    //返回 ModelAndView 对象
    public ModelAndView handleRequest(HttpServletRequest request,
           HttpServletResponse response)
           throws ServletException, IOException {
        //向 request 域中放入一条信息，给前端 jsp 用
        request.setAttribute("message", "hello,Spring MVC");
        //返回 jsp 的路径
        return new ModelAndView("hello");//和代码清单 5-1 的唯一区别!
    }
}
```

在代码 **return new** ModelAndView("hello")中返回的是“hello”字符串而不是“hello.jsp”的文件，为了让程序能正常工作，必须要在 springapp-servlet.xml 中加入 hello 的对应视图的解析方式，即 hello 的对应文件。Spring MVC 通过配置文件给“hello”加上前缀和后缀来指定唯一的物理文件。以下是修改后的 springapp-servlet.xml 文件内容。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">
    <!-- 定义用户请求路径和对应的响应处理类之间的关系 -->
<bean name="/hello.htm" class="com.ssmbook2020.web.ch5.HelloController" />
<!-- 配置一个视图解析器 -->
<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
       <property name="prefix" value="/WEB-INF/jsp/" />
       <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

其中，prefix 属性的值表示地址前缀，suffix 属性的值表示地址后缀，如图 5-7 所示。

```
<!-- 定义用户请求路径和对应的响应处理类之间的关系 -->
 <bean name="/hello.htm"
 class="com.ssmbook2020.web.ch5.HelloController" />

 <!-- 配置一个视图解析器 -->
 <bean
 class="org.springframework.web.servlet.view.InternalResourceViewResolver">
     <property name="prefix" value="/WEB-INF/jsp/" />
     <property name="suffix" value=".jsp" />
 </bean>
```

① 路径前缀

② 路径后缀

图 5-7　增加视图解析器

这样 hello 就会对应到 /WebRoot/WEB-INF/jsp/hello.jsp 文件，当然文件位置也要有对应变化，要把 jsp 文件放到 WEB-INF 文件夹下。这样有更好的安全性，因为用户无法直接访问 Tomcat 服务器下项目中 WEB-INF 下的文件，可以起到一定的保护作用。

## 5.3.3 Spring MVC 的体系结构

下面详细介绍 Spring MVC 的体系结构，图 5-8 为 Spring MVC 体系结构图。

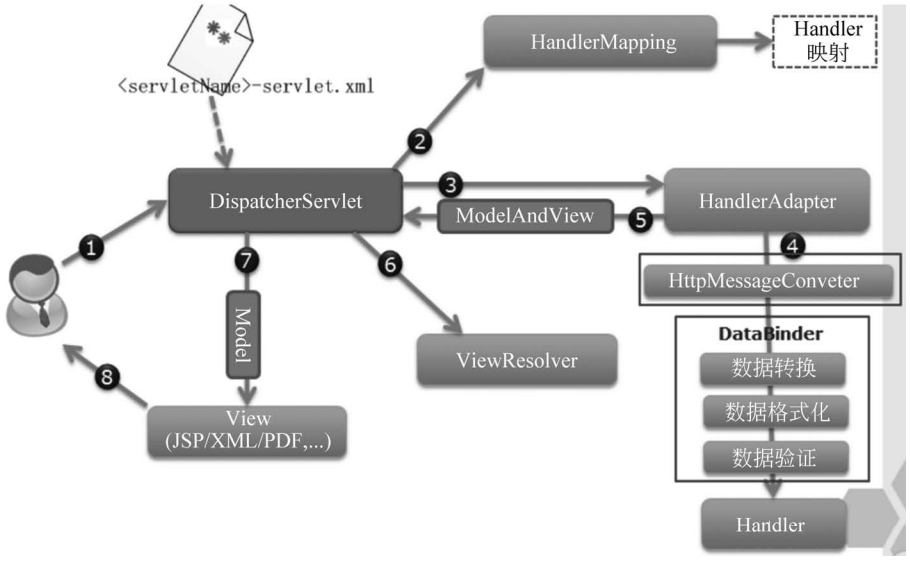


图 5-8 Spring MVC 体系结构图

分析 Spring MVC 体系结构图，可以得到整个流程如下。

（1）用户向服务器发送请求，Spring 前端控制 Servlet，请求被 DispatcherServlet 捕获。

（2）DispatcherServlet 对请求 URL 进行解析，得到请求资源标识符（URI）；然后根据该 URI 调用 HandlerMapping，获得该 Handler 配置的所有相关的对象（包括 Handler 对象和 Handler 对象对应的拦截器）；最后以 HandlerExecutionChain 对象的形式返回。

（3）DispatcherServlet 根据获得的 Handler，选择一个合适的 HandlerAdapter。如果成功获得 HandlerAdapter，此时将开始执行拦截器的 preHandler(...)方法。

（4）提取 Request 中的模型数据，填充 Handler 输入参数，开始执行 Handler(Controller)。在填充 Handler 输入参数的过程中，Spring 将根据配置做一些额外的工作。

① HttpMessageConveter：将请求消息（如 JSON、XML 等数据）转换成一个对象，将对象转换为指定的响应信息。

② 数据转换：对请求消息进行数据转换，如 String 转换成 Integer、Double 等。

③ 数据根式化：对请求消息进行数据格式化，如将字符串转换成格式化数字或格式化日期等。

④ 数据验证：验证数据的有效性（长度、格式等），将验证结果存储到 BindingResult 或 Error 中。

（5）Handler 执行完成后，向 DispatcherServlet 返回一个 ModelAndView 对象。

（6）根据返回的 ModelAndView，选择一个适合的 ViewResolver（必须是已经注册到 Spring 容器中的 ViewResolver)返回给 DispatcherServlet 。

（7）ViewResolver 结合 Model 和 View 来渲染视图。

（8）将渲染结果返回给客户端。

## 5.4 基于注解的控制器配置

在 5.2 节已经学习了 Spring MVC 的基本概念，对 Spring MVC 有了基本的了解。配置控制器的传统方式是采用 XML 配置形式。随着现在基于注解的方式在 Java EE 项目中逐渐流行，Spring MVC 2.5 版本后也支持基于注解的方式来配置控制器。

与 5.2 节基于 xml 配置的初始步骤相同，加入 Spring MVC 的包，并在项目的 web.xml 配置文件中加入 DispatchServlet 的配置。在与 web.xml 文件相同的位置，新建一个名为 springapp-servlet.xml 的 xml 文件。该 springapp-servlet.xml 配置文件中定义了用户请求的路径和对应的控制器的映射关系。

基于注解的控制器配置的实现步骤如下。

（1）修改控制器类

修改之前的 HelloController 类。为了和 5.2 节的 HelloController 区分，将这个类定义为 HelloController2。代码如下所示。

```
//代码清单 5-3 HelloController2.java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
//控制器注解
@Controller
public class HelloController2{
    //返回 ModelAndView 对象
    @RequestMapping(value="/helloController2")
    public ModelAndView handleRequest(HttpServletRequest request,
           HttpServletResponse response)
           throws ServletException, IOException {
        //向 Request 域中放入一条信息，给前端 jsp 用
        request.setAttribute("message", "hello,springmvc");
        //返回 jsp 的路径
        return new ModelAndView("hello");
    }
}
```

这个类和 5.2 节的 HelloController 类有以下两点不同。

① 该类不需要继承其他类，只有@Controller 注解。

② handleRequest 方法前有一个@RequestMapping(value="/helloController2")注解。

（2）修改 springapp-servlet.xml 文件，添加如图 5-9 所示的内容。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.1.xsd">
<!-- 包扫描器，自动扫描以com.ssmbook2020开头的包 -->
    <context:component-scan base-package="com.ssmbook2020"/>
    <mvc:annotation-driven />

    <!-- 配置一个视图解析器 -->
```

1 新增context和mvc的命名空间

2 注解包扫描配置

图 5-9　添加对注解的支持

在 MyEclipse 中，可以通过可视化的方式添加 XML 中的命名空间（XML Namespace），如图 5-10 所示。打开 springapp-servlet.xml 文件，在选项卡一栏中由 Source 模式切换到 Namespaces 模式，再勾选 context 和 mvc 选项，对应的 springapp-servlet.xml 文件中就会加上 context 和 mvc 命名空间。

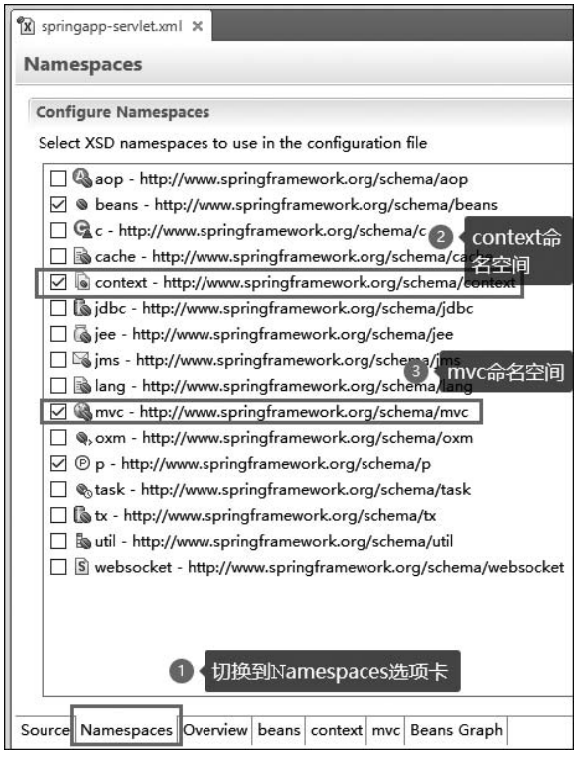


图 5-10　可视化添加命名空间

（3）部署运行。

将项目部署到 Tomcat 服务器，启动 Tomcat 服务器，在浏览器中输入地址 http://localhost: