Java 应用与开发
Servlet 编程

王晓东
wangxiaodong@ouc.edu.cn

计算机科学与技术系

November 13, 2018

学习目标

1. 理解 Web 的概念及工作模式，掌握 Java Web 应用的构成。
2. 掌握 Servlet 的概念、体系结构及生命周期管理基本原理。
3. 掌握 Servlet 的编程及配置方法，了解 Servlet 的在 Tomcat 服务器上的部署方式（war）。

大纲

Servlet 是主流 Web 框架的基础

JSP 和 JSF 都是建立在 Servlet 基础之上的，其他 Web 框架如 Struts、WebWork 和 Spring MVC 都是基于 Servlet。

接下来…

Web 基础

Servlet 概述

Servlet 编程

Servlet 生命周期

Servlet 配置

Servlet 部署

Servlet 示例

什么是 Web

- Web 本质上就是 Internet 上所有文档（资源）的集合，如 HTML 网页、CSS、JS、图片、动态网页、声音、视频等。
- Web 文档保存在 Web 站点上，Web 站点驻留在 Web 服务器上。
- 常见 Web 服务器有 Apache、IIS、WebLogic、GlassFish、JBoss 和 Tomcat 等。

Web 文档都有唯一的地址，通过 URL 来进行定位：

协议://IP 地址: 端口/站点名/目录/文件名

```
1  http://210.30.108.30:8080/jycrm/admin/login.jsp
2  ftp://210.30.108.30/software/jdk.zip
```

Web 工作模式

Web 使用请求/响应模式进行工作，Web 服务器不会主动将 Web 文档发送到客户端。

1. 由客户（一般是浏览器）使用 URL 对 Web 文档进行请求；
2. Web 服务器接收并处理请求；
3. 处理结束后将响应内容发送到客户。

Web 工作模式

- Web 请求方式主要有GET、POST、PUT、DELETE 和 HEAD。
- Web 响应一般情况下是 HTML 文档，也可以是其他类型资源。
- Web 使用 MIME (Multipurpose Internet mail Extensions) 标准来确定具体的响应类型。HTTP 响应总体上分为两类：文本类型（纯文本字符、HTML、XML）和二进制原始类型（图片、声音、视频）。

Java Web 应用的构成

- HTML 文档
- CSS
- JavaScript
- 图片文件
- Servlet
- JSP
- JavaBean 类
- Java Lib
- Web 配置文件：/WEB-INF/web.xml

演示 在 Eclipse 中创建一个 Java Dynamic Project。

接下来…

Servlet 概述

什么是 Servlet

- Servlet 是一种 Java Class，它运行在 Java EE 的 Web 容器内，由 Web 容器负责它的对象的创建和销毁，不能直接由其它类对象来调用。
- 当 Web 容器接收到对它的 HTTP 请求时，自动创建 Servlet 对象，并自动调用它的 doPost 或 doGet 方法。

Servlet 的主要功能

- 接收用户 HTTP 请求。
- 取得 HTTP 请求提交的数据。
- 调用 JavaBean 对象的方法。
- 生成 HTML 类型或非 HTML 类型的 HTTP 动态响应。
- 实现其他 Web 组件的跳转，包括重定向和转发。

Servlet 概述

与 Servlet 相近的技术

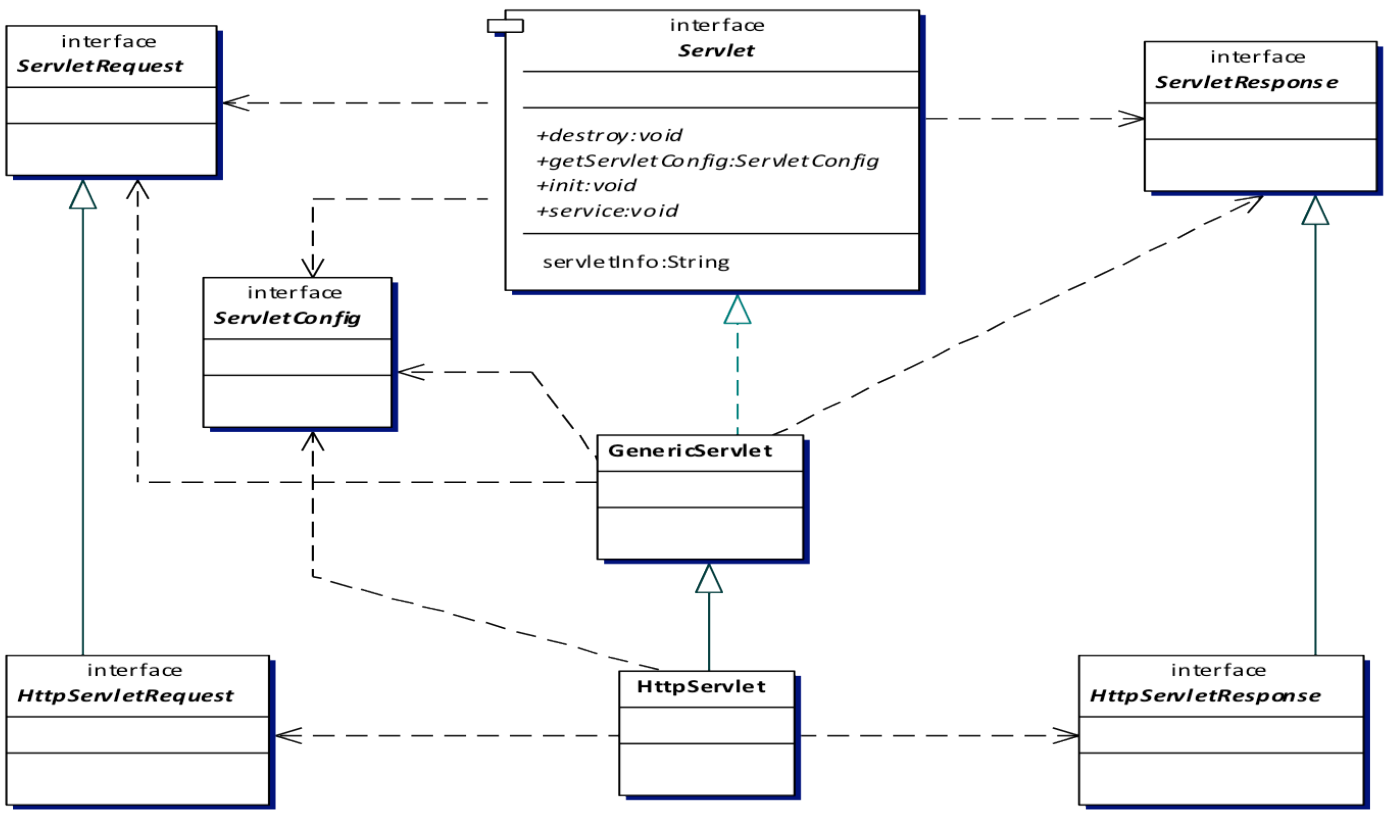
- CGI (Common Gateway Interface)。
- MS 的 HTTP DLL 技术。
- Perl 语言编写的处理代码。

Servlet 的特点

- 使用 Java 语言编写。
- 可以运行在符合 J2EE 规范的所有应用服务器上，实现跨平台运行。
- 单进程、多线程技术，运行速度快，节省服务器资源。

Servlet 体系结构

- javax.servlet 包含支持所有协议的的通用的 Web 组件接口和类；
- javax.servlet.http 包含了支持 HTTP 协议的接口和类。



接下来…

引入包

```
1 import java.io.*;
2 import javax.servlet.*;
3 import javax.servlet.http.*;
```

类定义

编写接收 HTTP 请求并进行 HTTP 响应的 Servlet 需要继承 javax.servlet.http.HttpServlet。

```
1 public class LoginAction extends HttpServlet {
2   // Code goes on.
3 }
```

重写 doGet 方法

父类 HttpServlet 的 doGet 方法是空的，没有实现任何代码，子类需要重写此方法。

```
2  public void doGet(HttpServletRequest request, HttpServletResponse response)
3  throws ServletException, IOException {
4    // Rewrite the method.
5  }
```

当 HTTP 请求为 GET 时自动运行，每次请求都运行一次。

重写 doPost 方法

编写 Servlet 需要重写父类的 doPost 方法。

```
1 public void doPost(HttpServletRequest request, HttpServletResponse response)
2 throws ServletException, IOException {
3   // Rewrite the method.
4 }
```

当请求方式为 POST 时自动运行，每次请求都运行一次。

doGet 和 doPost 方法都接收 Web 容器自动创建的请求对象和响应对象，使得 Servlet 能够解析请求数据和发送响应给客户端。

重写 init 方法

当 Web 服务器创建 Servlet 对象后，会自动调用 init 方法完成初始化功能，一般将耗时的连接数据库和打开外部资源文件的操作放在 init 方法中。
init 方法在 Web 容器创建 Servlet 对象后立即执行，且只执行一次。

```
1 public void init(ServletConfig config) throws ServletException {
2   super.init(config);
3   // 这里放置初始化工作代码.
4 }
```

在 init 方法中使用 Web 容器传递的 config 对象取得 Servlet 的各种配置初始参数，进而使用这些参数完成读取数据库或其他外部资源。

重写 destroy 方法

当 Web 容器需要销毁 Servlet 对象时，一般是 Web 容器停止运行或 Servlet 源代码修改而重新部署时，Web 容器自动运行 destroy 方法完成清理工作，如关闭数据库连接和 I/O 流。

```
1 public void destroy() {
2   try {
3     cn.close();
4   } catch (Exception e) {
5     application.Log("登录处理关闭数据库错误" + e.getMessage());
6   }
7 }
```

代码中 application 为 Web 应用的上下文环境对象。

接下来…

Servlet 的运行过程

1. 用户在浏览器请求 ServletURL 地址。
2. Web 容器接收到请求，检查是 Servlet 请求，将处理交给 Servlet 引擎。
3. Servlet 引擎根据 URL 地址检查是否有 Servlet 映射，如果没有则返回错误信息给浏览器。
4. 有 servlet 映射时，先检查是否有实例在运行。
5. 如果没有实例运行，则创建 Servlet 类的对象，调用其构造方法，然后调用 init() 方法。
6. 如果有实例在运行，则根据请求的方法是 GET 或 POST，自动调 doGet() 或 doPost() 方法。将请求对象和响应对象传给 doGet() 或 doPost() 方法。
7. 在 doGet() 或 doPost() 方法内通过 HttpServletRequest 的请求对象分析出用户发送的请求信息。
8. 按用户的要求进行业务处理。
9. 通过 HttpServletResponse 响应对象向浏览器发送响应信息。

Servlet 的运行过程

1. 用户在浏览器请求 ServletURL 地址。
2. Web 容器接收到请求，检查是 Servlet 请求，将处理交给 Servlet 引擎。
3. Servlet 引擎根据 URL 地址检查是否有 Servlet 映射，如果没有则返回错误信息给浏览器。
4. 有 servlet 映射时，先检查是否有实例在运行。
5. 如果没有实例运行，则创建 Servlet 类的对象，调用其构造方法，然后调用 init() 方法。
6. 如果有实例在运行，则根据请求的方法是 GET 或 POST，自动调 doGet() 或 doPost() 方法。将请求对象和响应对象传给 doGet() 或 doPost() 方法。
7. 在 doGet() 或 doPost() 方法内通过 HttpServletRequest 的请求对象分析出用户发送的请求信息。
8. 按用户的要求进行业务处理。
9. 通过 HttpServletResponse 响应对象向浏览器发送响应信息。

Servlet 的运行过程

1. 用户在浏览器请求 ServletURL 地址。
2. Web 容器接收到请求，检查是 Servlet 请求，将处理交给 Servlet 引擎。
3. Servlet 引擎根据 URL 地址检查是否有 Servlet 映射，如果没有则返回错误信息给浏览器。
4. 有 servlet 映射时，先检查是否有实例在运行。
5. 如果没有实例运行，则创建 Servlet 类的对象，调用其构造方法，然后调用 init() 方法。
6. 如果有实例在运行，则根据请求的方法是 GET 或 POST，自动调 doGet() 或 doPost() 方法。将请求对象和响应对象传给 doGet() 或 doPost() 方法。
7. 在 doGet() 或 doPost() 方法内通过 HttpServletRequest 的请求对象分析出用户发送的请求信息。
8. 按用户的要求进行业务处理。
9. 通过 HttpServletResponse 响应对象向浏览器发送响应信息。

Servlet 的运行过程

1. 用户在浏览器请求 ServletURL 地址。
2. Web 容器接收到请求，检查是 Servlet 请求，将处理交给 Servlet 引擎。
3. Servlet 引擎根据 URL 地址检查是否有 Servlet 映射，如果没有则返回错误信息给浏览器。
4. 有 servlet 映射时，先检查是否有实例在运行。
5. 如果没有实例运行，则创建 Servlet 类的对象，调用其构造方法，然后调用 init() 方法。
6. 如果有实例在运行，则根据请求的方法是 GET 或 POST，自动调 doGet() 或 doPost() 方法。将请求对象和响应对象传给 doGet() 或 doPost() 方法。
7. 在 doGet() 或 doPost() 方法内通过 HttpServletRequest 的请求对象分析出用户发送的请求信息。
8. 按用户的要求进行业务处理。
9. 通过 HttpServletResponse 响应对象向浏览器发送响应信息。

Servlet 的运行过程

1. 用户在浏览器请求 ServletURL 地址。
2. Web 容器接收到请求，检查是 Servlet 请求，将处理交给 Servlet 引擎。
3. Servlet 引擎根据 URL 地址检查是否有 Servlet 映射，如果没有则返回错误信息给浏览器。
4. 有 servlet 映射时，先检查是否有实例在运行。
5. 如果没有实例运行，则创建 Servlet 类的对象，调用其构造方法，然后调用 init() 方法。
6. 如果有实例在运行，则根据请求的方法是 GET 或 POST，自动调 doGet() 或 doPost() 方法。将请求对象和响应对象传给 doGet() 或 doPost() 方法。
7. 在 doGet() 或 doPost() 方法内通过 HttpServletRequest 的请求对象分析出用户发送的请求信息。
8. 按用户的要求进行业务处理。
9. 通过 HttpServletResponse 响应对象向浏览器发送响应信息。

Servlet 的运行过程

1. 用户在浏览器请求 ServletURL 地址。
2. Web 容器接收到请求，检查是 Servlet 请求，将处理交给 Servlet 引擎。
3. Servlet 引擎根据 URL 地址检查是否有 Servlet 映射，如果没有则返回错误信息给浏览器。
4. 有 servlet 映射时，先检查是否有实例在运行。
5. 如果没有实例运行，则创建 Servlet 类的对象，调用其构造方法，然后调用 init() 方法。
6. 如果有实例在运行，则根据请求的方法是 GET 或 POST，自动调 doGet() 或 doPost() 方法。将请求对象和响应对象传给 doGet() 或 doPost() 方法。
7. 在 doGet() 或 doPost() 方法内通过 HttpServletRequest 的请求对象分析出用户发送的请求信息。
8. 按用户的要求进行业务处理。
9. 通过 HttpServletResponse 响应对象向浏览器发送响应信息。

Servlet 的运行过程

1. 用户在浏览器请求 ServletURL 地址。
2. Web 容器接收到请求，检查是 Servlet 请求，将处理交给 Servlet 引擎。
3. Servlet 引擎根据 URL 地址检查是否有 Servlet 映射，如果没有则返回错误信息给浏览器。
4. 有 servlet 映射时，先检查是否有实例在运行。
5. 如果没有实例运行，则创建 Servlet 类的对象，调用其构造方法，然后调用 init() 方法。
6. 如果有实例在运行，则根据请求的方法是 GET 或 POST，自动调 doGet() 或 doPost() 方法。将请求对象和响应对象传给 doGet() 或 doPost() 方法。
7. 在 doGet() 或 doPost() 方法内通过 HttpServletRequest 的请求对象分析出用户发送的请求信息。
8. 按用户的要求进行业务处理。
9. 通过 HttpServletResponse 响应对象向浏览器发送响应信息。

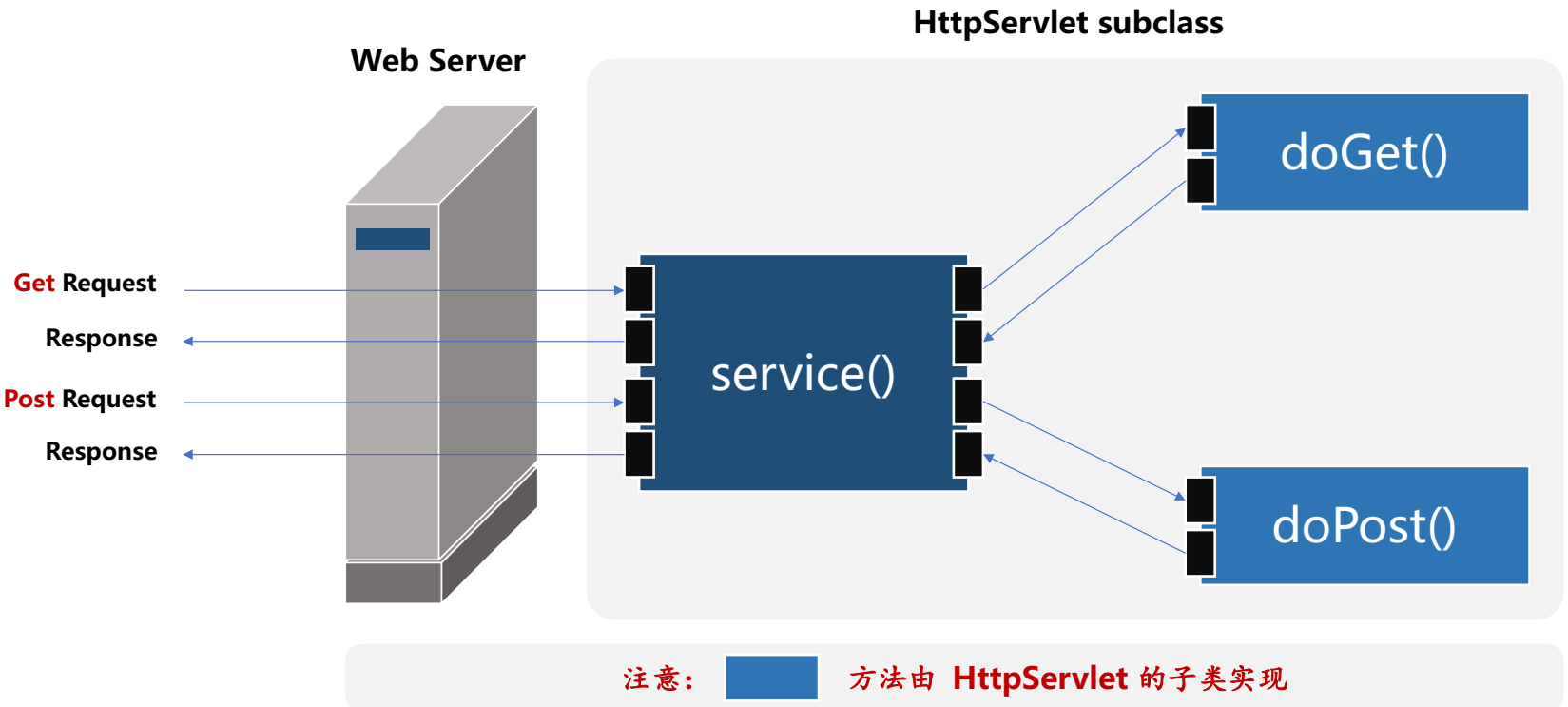
Servlet 的运行过程

1. 用户在浏览器请求 ServletURL 地址。
2. Web 容器接收到请求，检查是 Servlet 请求，将处理交给 Servlet 引擎。
3. Servlet 引擎根据 URL 地址检查是否有 Servlet 映射，如果没有则返回错误信息给浏览器。
4. 有 servlet 映射时，先检查是否有实例在运行。
5. 如果没有实例运行，则创建 Servlet 类的对象，调用其构造方法，然后调用 init() 方法。
6. 如果有实例在运行，则根据请求的方法是 GET 或 POST，自动调 doGet() 或 doPost() 方法。将请求对象和响应对象传给 doGet() 或 doPost() 方法。
7. 在 doGet() 或 doPost() 方法内通过 HttpServletRequest 的请求对象分析出用户发送的请求信息。
8. 按用户的要求进行业务处理。
9. 通过 HttpServletResponse 响应对象向浏览器发送响应信息。

Servlet 的运行过程

1. 用户在浏览器请求 ServletURL 地址。
2. Web 容器接收到请求，检查是 Servlet 请求，将处理交给 Servlet 引擎。
3. Servlet 引擎根据 URL 地址检查是否有 Servlet 映射，如果没有则返回错误信息给浏览器。
4. 有 servlet 映射时，先检查是否有实例在运行。
5. 如果没有实例运行，则创建 Servlet 类的对象，调用其构造方法，然后调用 init() 方法。
6. 如果有实例在运行，则根据请求的方法是 GET 或 POST，自动调 doGet() 或 doPost() 方法。将请求对象和响应对象传给 doGet() 或 doPost() 方法。
7. 在 doGet() 或 doPost() 方法内通过 HttpServletRequest 的请求对象分析出用户发送的请求信息。
8. 按用户的要求进行业务处理。
9. 通过 HttpServletResponse 响应对象向浏览器发送响应信息。

Servlet 处理流程



接下来…

Servlet 配置

- Servlet 作为 Web 组件可以处理 HTTP 请求/响应，因此对外要求一个唯一的 URL 地址。
- Servlet 是一个 Java 类文件，不像 JSP 那样直接存放在 Web 目录下就能获得 URL 请求访问地址。
- Servlet 必须在 Web 的配置文件/WEB-INF/web.xml中进行配置和映射才能响应 HTTP 请求。
- Servlet 的配置分为**声明和映射**两个步骤。

Servlet 配置

❖ Servlet 声明

通知 Web 容器 Servlet 的存在。

```
<servlet>
  <servlet-name>loginaction</servlet-name>
  <servlet-class>ouc.java.servlet.LoginAction</servlet-class>
</servlet>
```

<servlet-name> 声明 Servlet 的名字，要求在一个 web.xml 文件内名字唯一。

<servlet-class> 指定 Servlet 的全名，即包名. 类名。

Servlet 配置

Servlet 初始参数

在 Servlet 的声明中可以配置 Servlet 初始参数，如数据库的 Driver、URL、账号和密码等信息。在 Servlet 中可以读取这些信息，避免在 Servlet 代码中定义这些信息，修改时无需重新编译 Servlet。

```
<servlet>
  <init-param>
    <param-name>driver</param-name>
    <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
  </init-param>
</servlet>
```

在 Servlet 中取得以上定义的参数的方法：

```
String driver = config.getInitParameter("driver");
```

Servlet 配置

Servlet 启动时机

在配置 Servlet 时，可以指示 Servlet 跟随 Web 容器一起自动启动。这时，Servlet 就可以在没有请求的情形下，进行实例化和初始化，完成特定任务。自启动 Servlet 的配置语法：

```
1 <load-on-startup>2</load-on-startup>
```

数字越小越先启动，0 表示紧跟 Web 容器启动后第一个启动。

Servlet 配置

❖ Servlet 映射

- 任何 Web 文档在 Internet 上都要有一个 URL 地址才能被请求访问。
- Servlet 不能像 JSP 一样直接放在 Web 的发布目录上，需要单独映射 URL 地址。
- 在/WEB-INF/web.xml中进行 Servlet 的 URL 映射。

映射语法

```
1 <servlet-mapping>
2   <servlet-name>servlet name</servlet-name>
3   <url-pattern>URL</url-pattern>
4 </servlet-mapping>
```

其中，servlet name 与 Servlet 声明中的名称要一致。

Servlet 配置

❖ Servlet 映射

映射地址方式 ❶ 绝对地址方式映射

```
1 <servlet-mapping>
2   <servlet-name>LoginAction</servlet-name>
3   <url-pattern>/login.action</url-pattern>
4 </servlet-mapping>
```

Servlet 配置

❖ Servlet 映射

映射地址方式 ❷ 匹配目录模式映射方式

```
<servlet-mapping>
  <servlet-name>MainAction</servlet-name>
  <url-pattern>/main/*</url-pattern>
</servlet-mapping>
```

在这个配置中，只要以/main 开头的任何 URL 都能请求此 Servlet。

Servlet 配置

❖ Servlet 映射

映射地址方式 ❸ 匹配扩展名模式映射方式

```
1 <servlet-mapping>
2   <servlet-name>MainAction</servlet-name>
3   <url-pattern>*.action</url-pattern>
4 </servlet-mapping>
```

以上配置中扩展名为 action 的任何请求均被此 Servlet 响应。

注意：不能混合使用以上两种配置模式，否则会在 Web 项目部署并运行时产生运行时错误。

如以下配置是错误的：

```
1 <servlet-mapping>
2   <servlet-name>MainAction</servlet-name>
3   <url-pattern>/main/*.action</url-pattern>
4 </servlet-mapping>
```

接下来…

Web 基础

Servlet 概述

Servlet 编程

Servlet 生命周期

Servlet 配置

Servlet 部署

Servlet 示例

Servlet 部署

编译好的 Servlet class 文件应该放到指定的 Web 应用目录下，才能被 Web 容器找到，这个目录为：
/WEB-INF/classes/package/FileName.class
例如 Servlet 类 LoginAction：

```
1 package ouc.java.servlet;
2 public class LoginAction extends HttpServlet {
3   //
4 }
```

存放目录为：/WEB-INF/classes/ouc/java/servlet/LoginAction.class

接下来…

Servlet 示例

Eclipse

New Project ➟ Web ➟ Dynamic Web Project ➟ Next >

Project name: sample.servlet
Target runtime
Apache Tomcat v8.0
Dynamic web module version
3.0
Configuration
Default Configuration for Apache Tomcat v8.0

➟ Next >

Servlet 示例

Source folder on build path:

src

➠ Next >

✔ Generate web.xml deployment descriptor

➠ Finish

Servlet 示例

WebContent/WEB-INF/web.xml

Add following statements between <web-app> and </web-app>.

```
1   <servlet>
2     <servlet-name>HelloServlet</servlet-name>
3     <servlet-class>ouc.javaweb.HelloServlet</servlet-class>
4     <load-on-startup>0</load-on-startup>
5   </servlet>

7   <servlet-mapping>
8     <servlet-name>HelloServlet</servlet-name>
9     <url-pattern>/hello</url-pattern>
10  </servlet-mapping>
```

Servlet 示例

Java Resources/src

Create java class file named “HelloServlet.java”.

```
1  package ouc.javaweb;
2
3  import java.io.IOException;
4  import java.io.PrintWriter;
5  import javax.servlet.ServletException;
6  import javax.servlet.http.HttpServlet;
7  import javax.servlet.http.HttpServletRequest;
8  import javax.servlet.http.HttpServletResponse;
9
10 public class HelloServlet extends HttpServlet {
11   public void doGet(HttpServletRequest request, HttpServletResponse response)
12   throws ServletException, IOException {
13     response.setContentType("text/html");
14     response.setCharacterEncoding("UTF-8");
15     PrintWriter out = response.getWriter();
16     out.println("<html>");
17     out.println("<head><title>A␣Servlet␣Sample</title></head>");
18     out.println("<body>");
19     out.println("<h1>Hello,␣Servlet!</h1>");
20     out.println("A␣Servlet␣is␣a␣Java-based␣server-side␣web␣technology.␣");
21     out.println("</body></html>"););
22     out.flush();
23     out.close();
24   }
25 }
```

Servlet 示例

sample.servlet ➟ 鼠标右键 ➟ Run as ➟ Run on Server
➟ Choose an existing server ➟ Tomcat v8.0 Server at localhost
➟ Finish
在浏览器中请求页面http://localhost:8080/sample.servlet/hello。

本节习题

❖ 问答题

1. Servlet 和一般 Java 类的区别是什么？
2. 简述 Servlet 的生命周期。
3. 简述 Servlet 与 URL 地址的映射方式（包括 web.xml 配置和基于注解）。

❖ 小编程

1. 编写一个能够计数访问次数的 Servlet，每次请求次数增加 1，并显示当前总访问次数。

THE END

wangxiaodong@ouc.edu.cn