第6讲 Servlet技术

1. Servlet基础

- Java Servlet 是运行在 Web 服务器或应用服务器上的程序，它是作为来自 Web 浏览器或其他 HTTP 客户端的请求和 HTTP 服务器上的数据库或应用程序之间的中间层。
- 一个Web服务器中可以同时存在多个Servlet，根据客户端请求的URL地址来调用某个Servlet的服务为客户端产生响应

2. Servlet代码结构

- 声明一个Servlet类时，需要继承 javax.servlet.http.HttpServlet类
- HttpServlet主要定义的方法：
  - init()：初始化方法
  - service()：接受客户端的请求，根据请求类型调用相应的doGet(), doPost()等方法
  - destroy()：销毁方法
- 例6-1：FirstServlet

3. Servlet生存周期

- Servlet 生命周期可被定义为从创建直到毁灭的整个过程。以下是 Servlet 遵循的过程：
  - Servlet 通过调用 init () 方法进行初始化。
  - Servlet 调用 service() 方法来处理客户端的请求。
  - Servlet 通过调用 destroy() 方法终止（结束）。
  - 最后，Servlet 是由 JVM 的垃圾回收器进行垃圾回收的。

init()方法

- init()方法只在某个Servlet被部署到服务器之后，用户第一次调用时，系统创建Servlet 时被调用一次，在后续每次用户请求时不再调用
- init()方法主要进行对数据进行初始化，这些数据将被用于 Servlet 的整个生命周期。

service()方法

- service() 方法是执行实际任务的主要方法。
- Web 服务器调用 service() 方法来处理来自客户端浏览器的请求，并把格式化的响应写回给客户端。
- 每次服务器接收到一个 Servlet 请求时，服务器会产生一个新的线程并调用服务。service() 方法会检查 HTTP 请求类型（GET、POST、PUT、DELETE 等），并在适当的时候调用 doGet、doPost、doPut，doDelete 等方法。

destroy()方法

- destroy() 方法只会被调用一次，在 Servlet 生命周期结束时被调用。
- destroy() 方法可以使得 Servlet 关闭数据库连接、停止后台线程、把 Cookie 列表或点击计数器写入到磁盘，并执行其他类似的清理活动。
- 在调用 destroy() 方法之后，servlet 对象被标记为垃圾回收，最后由Java运行环境的垃圾回收机制进行回收。

4. Servlet配置

- 在编写完Servlet后，需要适当的配置，以便告诉Web服务器Servlet的访问名称和访问路径
- 两种方法进行配置
  - 在web.xml配置文件中进行配置
  - 使用注解进行配置

(1) 在web.xml中进行配置

- 使用<servlet>标签声明一个Servlet对象

```
<servlet>
    <servlet-name>SecondServlet</servlet-name>
    <servlet-class>servlets.SecondServlet</servlet-class>
</servlet>
```

- 使用<servlet-mapping>标签声明访问某一个Servlet对象的URL

```
<servlet-mapping>
  <servlet-name>SecondServlet</servlet-name>
  <url-pattern>/SecondServlet</url-pattern>
</servlet-mapping>
```

- 例6-2

(2) 使用@WebServlet注解

注解支持的属性：
- asyncSupported: 声明Servlet是否支持异步操作模式
- displayName: 该Servlet的显示名，通常配合工具使用
- initParams: 指定一组Servlet初始化参数
- loadOnStartup: 指定Servlet的加载顺序，
- name: 指定Servlet的name属性，等价于<servlet-name>，如果没有显式指定，则该Servlet的取值即为类的全名
- urlPatterns: 指定一组Servlet的URL匹配模式(虚拟路径)
- value: 该属性等价于urlPatterns属性。两个属性不能同时使用

举例：

```
@WebServlet(name="firstServlet", urlPatterns={"/firstServlet"})
@WebServlet(“/firstServlet”)          //默认为访问的URL
```

例6-1，4-13

5. Servlet过滤器

- Servlet过滤器是客户端与目标资源间的中间层组件，用于拦截客户端的请求与响应信息
- Servlet 过滤器是可用于 Servlet 编程的 Java 类，可以实现以下目的：
  - 在客户端的请求访问后端资源之前，拦截这些请求。
  - 在服务器的响应发送回客户端之前，处理这些响应。

多个过滤器示意图

多个过滤器的应用

Servlet过滤器编写

- 过滤器是一个实现了 javax.servlet.Filter 接口的 Java 类。
- javax.servlet.Filter 接口定义了三个方法

方　法	说　明
public void init(FilterConfig filterConfig)	过滤器的初始化方法，容器调用此方法完成过滤的初始化。对于每一个Filter实例，此方法只被调用一次
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)	此方法与Servlet的service()方法相类似，当请求及响应交给过滤器时，过滤器调用此方法进行过滤处理
public void destroy()	在过滤器生命周期结束时调用此方法，用于释放过滤器所占用的资源

在web.xml文件中配置过滤器

1. 声明过滤器对象

```
<filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>servlets.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>utf-8</param-value>
    </init-param>
</filter>
```

2. 映射过滤器

```
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

<dispatcher>的可选值及说明

可选值	说明
REQUEST	当客户端直接请求时，通过过滤器进行处理
INCLUDE	当客户端通过RequestDispatcher对象的include()方法请求时，通过过滤器进行处理
FORWARD	当客户端通过RequestDispatcher对象的forward()方法请求时，通过过滤器进行处理
ERROR	当声明式异常产生时，通过过滤器进行处理

过滤器举例

例6-3

6. Servlet与JSP的区别

- Servlet向客户端产生的所有输出都需要使用程序代码完成；而JSP页面可直接使用HTML标签进行输出。
- Servlet适合向客户端提供大量数据和少量HTML代码，JSP适合向客户端提供少量数据和大量HTML代码
- Servlet中需要调用Servlet API接口处理HTTP请求，而在JSP页面中，则直接提供了内置对象进行处理。
- Servlet的使用需要进行一定的配置，而JSP文件通过“.jsp”扩展名部署在容器之中，容器对其自动识别，直接编译成Servlet进行处理。