

CH02_Servlet(上)



Servlet(Server Applet)是Java Servlet的简称，称为小服务程序或服务连接器，它是由Java编写的服务器端程序，具有独立于平台和协议的特性，主要用于交互式地浏览和生成数据，生成动态Web内容，是Java Web程序中最核心的组件。

够发布和运行Java Web程序的Web服务器称为Servlet容器，比如Tomcat，Servlet必须运行在Servlet容器中从而能够为各种各样的用户请求提供对应的服务。

一、Servlet的功能

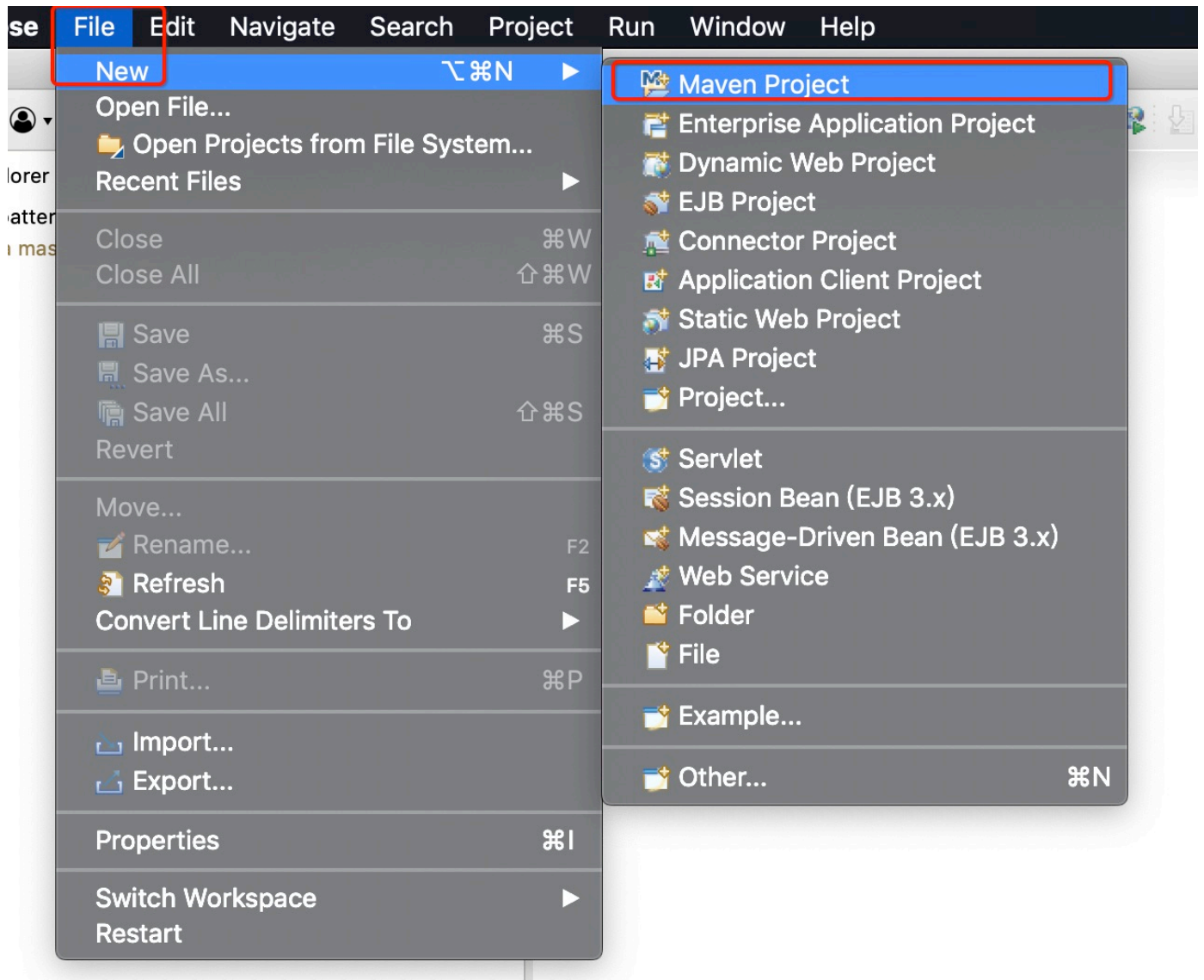
- 能够动态的生成html文档。
- 能够把请求转发给同一个web程序中的其他Servlet组件。
- 能够把请求转发给其他web程序中的Servlet组件。
- 读取和写入客户端的Cookie。
- 访问其他服务器资源，比如数据库。

二、第一个Servlet程序

我们创建一个简单的Servlet，当客户端请求时，输出一段内容到客户端。

首先，我们要创建一个Web项目，这里我们借助Maven来构建管理。

- 新建Maven Web项目:



New Maven Project

New Maven project

Select project name and location

☐ Create a simple project (skip archetype selection)

☒ Use default Workspace location

Location: Browse...

☒ Add project(s) to working set

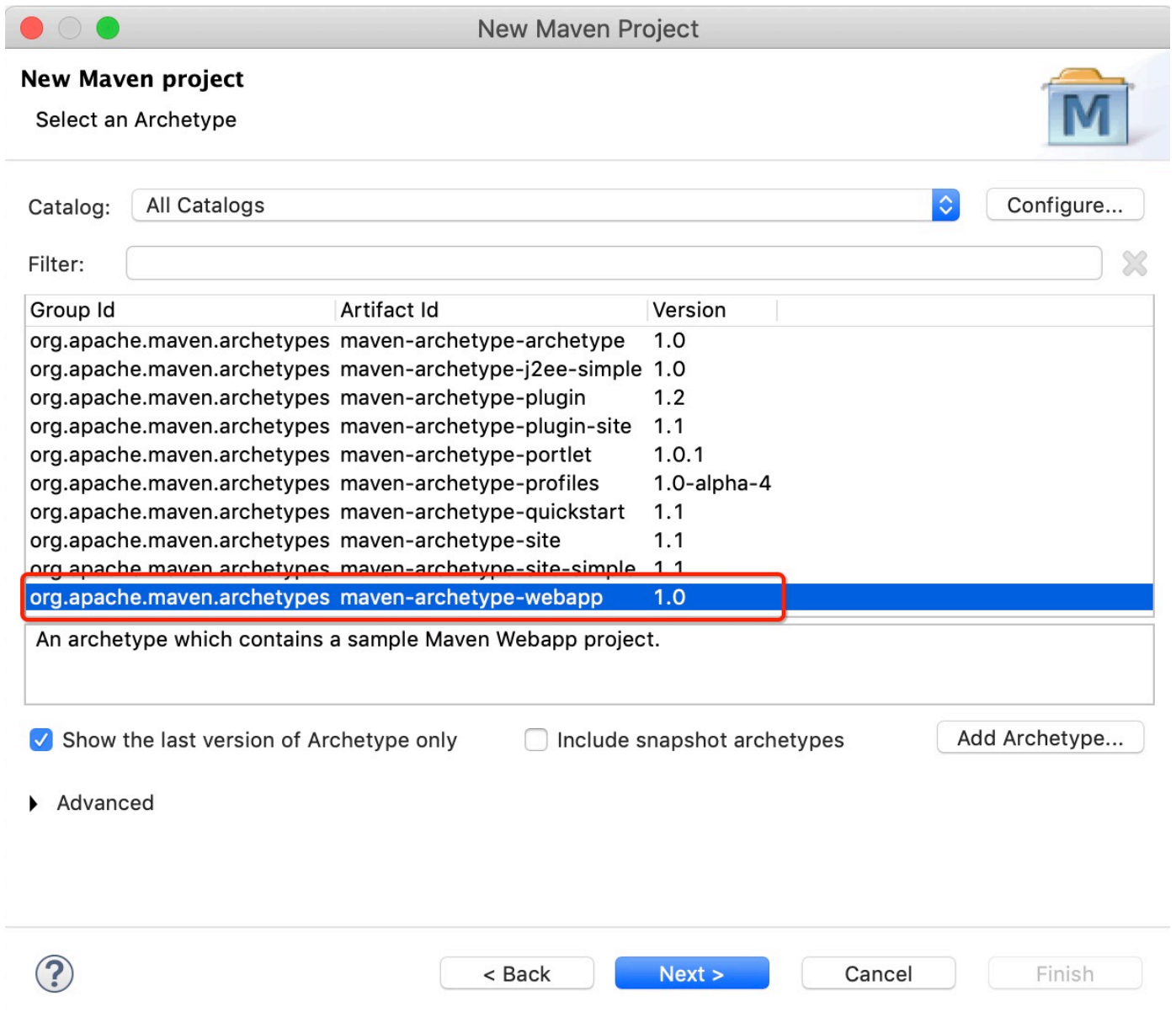
Working set: More...

▶ **Advanced**

可以不选，这里为了eclipse中项目管理方便，选择了web

? < Back Next > Cancel Finish

- 选择maven web项目，点击Next:



- 填写项目信息，点击Finish，项目创建完毕:

New Maven Project

New Maven project

Specify Archetype parameters

Group Id:com.cheer

Artifact Id:servlet

Version:0.0.1-SNAPSHOT

Package:com.cheer

Properties available from archetype:

Name	Value

Add...

Remove

Advanced

?

< Back

Next >

Cancel

Finish

- 点击pom.xml文件，在其中添加servlet-api依赖:

```
<dependencies>
```

```
  <dependency>
```

```
    <groupId>junit</groupId>
```

```
    <artifactId>junit</artifactId>
```

```
    <version>3.8.1</version>
```

```
    <scope>test</scope>
```

```
  </dependency>
```

```
  <dependency>
```

```
    <groupId>javax.servlet</groupId>
```























```
    <artifactId>servlet-api</artifactId>
```

```
    <version>2.5</version>
```

```
  </dependency>
```

```
</dependencies>
```

- 在src/main/java路径下，创建Servlet类：

- ▼  servlet
 - ▶  Deployment Descriptor: First Servlet
 - ▼  Java Resources
 - ▼  src/main/java
 - ▼  com.cheer.pkg1
 - ▶  FirstServlet.java
 - ▶  src/main/resources
 - ▶  src/test/java
 - ▶  Libraries
 - ▶  JavaScript Resources
 - ▶  Deployed Resources
 - ▼  src
 - ▼  main
 - ▶  java
 - ▶  resources
 - ▼  webapp
 - ▼  WEB-INF
 -  web.xml
 -  index.jsp
 - ▶  test
 - ▶  target
 -  pom.xml


```

public class FirstServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        //设置响应正文MIME类型和字符编码
        resp.setContentType("text/html;charset=utf-8");
        //获取PrintWriter对象, Servlet用它来输出字符串形式的正文数据
        PrintWriter out = resp.getWriter();
        //向响应正文输出信息
        out.println("<h1><font color=\"red\">Hello, 这是我的第一个Servlet程序...</font></h1>");
    }
}

```

- 打开web.xml, 在其中配置我们刚刚编写的Servlet类:

```

<!-- 定义创建的Servlet -->
<servlet>
    <!-- 定义Servlet的名字 -->
    <servlet-name>servlet01</servlet-name>
    <!-- 指定Servlet的完整类名 -->
    <servlet-class>com.cheer.pkg1.FirstServlet</servlet-class>
</servlet>

<!-- 为创建的Servlet映射一个URL -->
<servlet-mapping>
    <!-- 指定待映射的Servlet名字 -->
    <servlet-name>servlet01</servlet-name>
    <!-- 指定访问Servlet的相对URL路径 -->
    <url-pattern>/servlet01</url-pattern>
</servlet-mapping>

```

- 把该Web项目部署到Tomcat下, 并启动Tomcat。然后在浏览器中输入地址: <http://localhost:8080/servlet/servlet01> 即可看见页面输出我们在Servlet类中定义的内容。

完整的代码, 参考 [第一个Servlet程序完整代码](#)。

总结下创建Servlet的步骤:

- 添加servlet-api.jar依赖
- 创建Servlet类, 必须继承GenericServlet或者HttpServlet类。
- 在web.xml中配置我们创建的Servlet类。

三、Servlet相关的API

Servlet相关的API可以在 [Oracle官网](#) 上查看。下面我们学习一些主要的API。

1. javax.servlet.Servlet接口

Servlet API的核心是javax.servlet.Servlet接口，所有的Servlet都必须实现该接口。该接口提供了5个方法，其中三个由Servlet容器在Servlet的不同生命周期阶段来调用：

- `init(ServletConfig config)`方法: 该方法负责初始化Servlet对象，Servlet容器在创建完Servlet对象后会立即调用该方法。
- `service(ServletRequest request, ServletResponse response)`方法: 用于响应客户端的请求，为客户端提供服务。容器接收到客户端访问指定的Servlet请求时，就会调用该Servlet的`service`方法。
- `destroy()`方法: 用于释放Servlet对象占用的资源，比如关闭文件输入、输出流，关闭数据库连接等。Servlet容器在Servlet对象结束生命周期时会调用该方法。

此外，Servlet接口还提供了2个方法，Java Web程序通过调用这2个方法可以获得Servlet的相关信息：

- `getServletConfig()`方法: 返回一个ServletConfig对象，该对象中包含了Servlet的初始化参数信息。
- `getServletInfo()`: 返回一个包含Servlet创建者、版权和版本等信息的字符串。

在Servlet API中，`javax.servlet.GenericServlet`抽象类实现了Servlet接口，而`javax.servlet.HttpServlet`继承了`GenericServlet`类。当我们开发自己的Servlet时，可以继承`GenericServlet`或者`HttpServlet`类。

javax.servlet.http

Class HttpServlet

```
java.lang.Object
└─ javax.servlet.GenericServlet
    └─ javax.servlet.http.HttpServlet
```

All Implemented Interfaces:

`java.io.Serializable`, [Servlet](#), [ServletConfig](#)

2. javax.servlet.GenericServlet抽象类

GenericServlet类与任何传输协议都无关。Servlet查看该抽象类的API可以发现:

- 该抽象类为Servlet接口提供了通用实现, 此外它还实现了ServletConfig接口和Serializable接口。

```
public abstract class GenericServlet  
    implements Servlet, ServletConfig, java.io.Serializable
```

- 该抽象类中拥有一个私有成员变量ServletConfig config, 当Servlet容器调用init(ServletConfig config)方法时, 会让成员变量config引用由Servlet容器传入的ServletConfig对象。

```
public void init(ServletConfig config) throws ServletException {  
    this.config = config;  
    this.init();  
}
```

- 除了上面的init(ServletConfig config)方法之外, GenericServlet类还提供了该方法的另一种重载形式:

```
public void init() throws ServletException {  
  
}
```

该方法啥也没干, 当子类需要时可以重写该方法。

当我们开发自己的Servlet继承GenericServlet类时, 都可以在自己的Servlet类中重写这两个方法, 如果想要当前Servlet对象和容器传入的ServletConfig对象关联, 可以调用带参数的init方法, 实际开发一般都是重写带参数的init方法。

- GenericServlet类没用实现service方法, 留待子类去实现, 这样子类可以根据客户端的请求提供各种各样的实现。

```
public abstract void service(ServletRequest req, ServletResponse res)  
    throws ServletException, IOException;
```

- GenericServlet实现了destroy()方法, 但是啥都没干。GenericServlet的子类可以根据需要去重写它。

3. javax.servlet.http.HttpServlet抽象类

该类是GenericServlet的子类，但是它提供了与http协议相关的通用实现。HttpServlet对象适合运行在与客户端采用http协议的通信的Servlet容器或者Web服务器中。

在Java Web程序中，自定义的Servlet一般都是继承HttpServlet类。

Http传输协议把用户请求分为post、get、put等方式，HttpServlet类为每一种请求方式都提供了对应的处理方法，比如doPost()、doGet()、doPut()等方法。

```

protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException
{
    String method = req.getMethod();

    if (method.equals(METHOD_GET)) {
        long lastModified = getLastModified(req);
        if (lastModified == -1) {
            // servlet doesn't support if-modified-since, no reason
            // to go through further expensive logic
            doGet(req, resp);
        } else {
            long ifModifiedSince = req.getDateHeader(HEADER_IFMODSINCE);
            if (ifModifiedSince < (lastModified / 1000 * 1000)) {
                // If the servlet mod time is later, call doGet()
                // Round down to the nearest second for a proper compare
                // A ifModifiedSince of -1 will always be less
                maybeSetLastModified(resp, lastModified);
                doGet(req, resp);
            } else {
                resp.setStatus(HttpServletResponse.SC_NOT_MODIFIED);
            }
        }
    }

    } else if (method.equals(METHOD_HEAD)) {
        long lastModified = getLastModified(req);
        maybeSetLastModified(resp, lastModified);
        doHead(req, resp);

    } else if (method.equals(METHOD_POST)) {
        doPost(req, resp);

    } else if (method.equals(METHOD_PUT)) {
        doPut(req, resp);

    } else if (method.equals(METHOD_DELETE)) {
        doDelete(req, resp);

    } else if (method.equals(METHOD_OPTIONS)) {
        doOptions(req, resp);

    } else if (method.equals(METHOD_TRACE)) {
        doTrace(req, resp);

    } else {
        //
        // Note that this means NO servlet supports whatever
        // method was requested, anywhere on this server.
        //
    }
}

```

可以看出，HttpServlet实现了GenericServlet中的service方法，它会根据容器传入的HttpServletRequest对象的getMethod()方法获得客户端的请求方式，然后根据不同的请求方

式调用不同的处理方法，比如doGet()、doPost()等。

所以我们在Java Web程序中，自定义的Servlet都是继承HttpServlet类，一般不用去重写service方法，只需要根据客户端不同的请求方式去重写对应的doXXX方法即可。重写的时候，应该扩大方法的访问权限，即由protected升级为public，这是为了能让重写的方法可以被Servlet容器访问。

4. javax.servlet.ServletRequest接口

该接口表示来自客户端的请求。当Servlet容器接收到客户端请求访问指定的Servlet时，容器会先解析原始请求数据，把它封装成ServletRequest对象。最后当容器调用Servlet对象的service方法时，会把ServletRequest对象传递给该方法。

```
public void service(ServletRequest req, ServletResponse res)  
throws ServletException, IOException;
```

该接口提供了很多方法用于读取客户端的请求数据:

```
1  /**  
2   * 获取请求正文的长度，如果请求正文的长度未知，则返回-1。  
3   */  
4   public int getLength();  
5   /**  
6   * 获得请求正文的MIME类型，如果请求正文的类型未知，则返回null。  
7   */  
8   public String getContentType();  
9   /**  
10  * 获取用于读取请求正文的输入流。  
11  */  
12  public ServletInputStream getInputStream() throws IOException;  
13  /**  
14  * 获取读取字符串形式的请求正文的BufferedReader对象。  
15  */  
16  public BufferedReader getReader() throws IOException;  
17  /**  
18  * 获取服务器的主机名。  
19  */  
20  public String getLocalName();  
21  /**  
22  * 获取服务器的IP地址。
```

```

23  */
24  public String getLocalAddr();
25  /**
26   * 获取服务器的FTP端口号。
27   */
28  public int getLocalPort();
29  /**
30   * 根据给定的参数名，获取客户端请求中的参数值。
31   */
32  public String getParameter(String name);
33  /**
34   * 获取客户端和服务端通信的协议名称以及版本。
35   */
36  public String getProtocol();
37  /**
38   * 获取客户端的IP地址。
39   */
40  public String getRemoteAddr();
41  /**
42   * 获取客户端的主机名。
43   */
44  public String getRemoteHost();
45  /**
46   * 获取客户端的FTP端口号。
47   */
48  public int getRemotePort();

```

MIME: 多用途互联网邮件扩展（Multipurpose Internet Mail Extensions），最初是为了将纯文本格式的电子邮件扩展到可以支持多种信息格式而定制的。后来被应用到多种协议里，包括我们常用的HTTP协议。MIME的常见形式是一个主类型加一个子类型，用斜线分隔。比如text/html、application/javascript、image/png等，在访问网页时，MIME type帮助浏览器识别一个HTTP请求返回的是什么内容的数据，应该如何打开、如何显示。

ServletRequest接口还提供了一些操作请求范围内属性的方法:

```

1  /**
2   * 根据指定的属性名获取请求范围内匹配的属性值。
3   */
4  public Object getAttribute(String name);
5  /**
6   * 在请求范围内设置一个属性。
7   * @param name: 属性名

```



```
8      * @param o: 属性值
9      */
10     public void setAttribute(String name, Object o);
11     /**
12      * 根据指定的属性名在请求范围内删除一个指定的属性。
13      */
14     public void removeAttribute(String name);
```

5. javax.servlet.http.HttpServletRequest接口

HttpServletRequest继承了ServletRequest接口，前面在讲HttpServlet的时候，HttpServlet类中的doXXX()方法和service方法，都带有一个参数是HttpServletRequest类型的。

```
protected void service(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException
```

HttpServletRequest接口也提供了很多读取http请求中相关信息的方法：

```
1      /**
2       * 获取http请求中的所有cookie。
3       */
4       public Cookie[] getCookies();
5       /**
6       * 获取客户端请求访问的Web程序的URL入口。
7       * 比如客户端请求访问 http://localhost:8080/iweb/index.jsp, 该方法返回/iweb。
8       */
9       public String getContextPath();
10      /**
11      * 获取http请求头部的指定项。
12      */
13      public String getHeader(String name);
14      /**
15      * 获取http请求头部的所有项目名，返回结果时一个枚举Enumeration对象。
16      */
17      public Enumeration getHeaderNames();
18      /**
19      * 获取http请求方式，比如get方式、post方式等。
20      */
21      public String getMethod();
22      /**
23      * 获取http请求的头部的第一行中的URI。
```

```

24 */
25 public String getRequestURI();
26 /**
27 * 返回http请求中的查询字符串，即URL中?后面的内容。
28 * 比如客户端请求访问 http://localhost:8080/iweb/index.do?loginName=ZhangSan
29 * ，那么该方法返回loginName=ZhangSan。
30 */
    public String getQueryString();

```

HttpServletRequestApi.java测试了HttpServletRequest中的大部分方法

6. javax.servlet.ServletResponse接口

Servlet接口的service方法中除了ServletRequest类型的参数，还有一个ServletResponse参数。当Servlet容器接收到客户端请求访问指定的Servlet时，容器会创建一个ServletResponse对象，并传递给service方法：

```

public void service(ServletRequest req, ServletResponse res)
throws ServletException, IOException;

```

该接口提供了很多与生成结果相关的方法：

```

1 /**
2 * 返回响应正文的字符编码，这里指编码格式
3 */
4 public String getCharacterEncoding();
5 /**
6 * 设置响应正文的字符编码，默认编码是ISO-8859-1。
7 */
8 public void setCharacterEncoding(String charset);
9 /**
10 * 设置响应正文的长度。
11 */
12 public void setContentLength(int len);
13 /**
14 * 设置响应正文的MIME类型，比如xml、html等。
15 */
16 public void.setContentType(String type);
17 /**
18 * 获取响应正文的MIME类型，默认类型是为text/plain，即纯文本类型
19 */

```

```

19 public String getContentType();
20 /**
21  * 设置用于存放响应正文数据的缓冲区大小。
22  */
23 public void setBufferSize(int size);
24 /**
25  * 获取用于存放响应正文数据的缓冲区大小。
26  */
27 public int getBufferSize();
28 /**
29  * 清空缓冲区内的正文数据，并清除响应状态代码及响应头。
30  */
31 public void reset();
32 /**
33  * 仅仅清空缓冲区内的正文数据，不清除响应状态代码及响应头。
34  */
35 public void resetBuffer();
36 /**
37  * 手动把缓冲区内的正文数据发送到客户端。
38  */
39 public void flushBuffer() throws IOException;
40 /**
41  * 检测缓冲区内的正文数据是否发送到客户端，true表示已经发送，false表示未发送。
42  */
43 public boolean isCommitted();
44 /**
45  * 获取一个ServletOutputStream对象，Servlet用它来输出二进制的正文数据。
46  */
47 public ServletOutputStream getOutputStream() throws IOException;
48 /**
49  * 获取一个PrintWriter对象，Servlet用它来输出字符串形式的正文数据。
50  */
51 public PrintWriter getWriter() throws IOException;
52

```

如果要设置响应正文的MIME类型和字符编码，必须先调用ServletResponse对象的setContentType () 和setCharacterEncoding () 方法，然后再调用ServletResponse的getOutputStream () 或getWriter () 方法，以及提交缓冲区内的正文数据。只有满足这样的操作顺序，所作的设置才能生效。

7. javax.servlet.http.HttpServletResponse接口

该接口是ServletResponse的子接口，前面在讲HttpServlet的时候，HttpServlet类中的doXXX()方

法和service方法，一个参数是HttpServletRequest类型的，另一个参数就是HttpServletResponse类型的。

```
protected void service(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException  
,
```

该接口提供了一些与HTTP传输协议相关的方法，Servlet可以通过这些方法来设置HTTP响应头部或者向客户端写Cookie:

```
1  /**  
2  * 在http响应头中加入一项内容。  
3  */  
4  public void addHeader(String name, String value);  
5  /**  
6  * 设置http响应头中一项内容，如果响应头中存在这项内容，则旧的内容会被覆盖。  
7  */  
8  public void setHeader(String name, String value);  
9  /**  
10 * 向客户端发送一个代表指定错误的http响应状态代码。  
11 */  
12 public void sendError(int sc) throws IOException;  
13 /**  
14 * 向客户端发送一个代表指定错误的http响应状态代码，并且发送具体的错误信息。  
15 */  
16 public void sendError(int sc, String msg) throws IOException;  
17 /**  
18 * 设置http响应状态代码。  
19 */  
20 public void setStatus(int sc);  
21 /**  
22 * 向http响应中加入一个Cookie。  
23 */  
24 public void addCookie(Cookie cookie);
```

对于响应状态代码，该接口提供了静态常量:

```
1  public static final int SC_BAD_REQUEST; //对应的响应状态代码为400。  
2  public static final int SC_FOUND; //对应的响应状态代码为302。  
3  public static final int SC_METHOD_NOT_ALLOWED; //对应的响应状态代码为405。  
4  public static final int SC_NON_AUTHORITATIVE_INFORMATION; //对应的响应状态  
5  代码为203。
```

```
6 public static final int SC_FORBIDDEN; //对应的响应状态代码为403。
7 public static final int SC_NOT_FOUND; //对应的响应状态代码为404。
  public static final int SC_OK; //对应的响应状态代码为200
```

这只http响应正文的MIME类型及其字符编码，有三种方式：

```
1 //第一种方式
2 resp.setContentType("text/html;charset=utf-8");
3 //第二种方式
4 resp.setContentType("text/html");
5 resp.setCharacterEncoding("utf-8");
6 //第三种
7 resp.setHeader("Content-type", "text/html;charset=utf-8");
```

8. javax.servlet.ServletConfig接口

Servlet接口的init方法有一个ServletConfig类型的参数。当Servlet容器初始化一个Servlet对象时，会为这个Servlet对象创建一个ServletConfig对象，ServletConfig对象中包含了Servlet的初始化参数信息，此外，ServletConfig对象还与当前Web应用的ServletConfig对象关联。Servlet容器调用Servlet对象init (ServletConfig config) 方法时，会把ServletConfig对象作为参数传给Servlet对象，init (ServletConfig config) 方法会使得当前Servlet对象与 ServletConfig 对象之间建立关联关系。

ServletConfig接口中定义了一些和Servlet初始化信息有关的方法：

```
1 /**
2  * 根据给定的参数名，获取参数值。
3  */
4 public String getInitParameter(String name);
5 /**
6  * 获取所有的初始化参数名，返回类型是Enumeration枚举。
7  */
8 public Enumeration getInitParameterNames();
9 /**
10 * 获取ServletContext对象。
11 */
12 public ServletContext getServletContext();
13 /**
14 * 获取Servlet的名字即web.xml文件中相应<servlet>元素的<servlet-name>子元素的值。
15 如果没有为Servlet配置<servlet-name>子元素，则返回Servlet类的名字。每个初始化参数包
```

```
16 | 括一对参数名。  
    */  
    public String getServletName();
```

ServletConfigApi.java测试了ServletConfig中的大部分方法

Servlet初始化参数可以在web.xml中配置Servlet对象的时候进行设置，下面的Servlet配置了一个初始化参数name，值是Jim。

```
1 | <servlet>  
2 |     <servlet-name>SecondServlet</servlet-name>  
3 |     <display-name>SecondServlet</display-name>  
4 |     <description></description>  
5 |     <servlet-class>com.cheer.SecondServlet</servlet-class>  
6 |     <init-param>  
7 |         <param-name>name</param-name>  
8 |         <param-value>Jim</param-value>  
9 |     </init-param>  
10 |     <load-on-startup>1</load-on-startup>  
11 | </servlet>
```

HttpServlet类继承GenericServlet类，GenericServlet类实现了ServletConfig接口，因此在HttpServlet类或GenericServlet类以极其子类中都可以直接调用ServletConfig接口中的方法。

9. javax.servlet.ServletContext接口

该接口是Servlet和Servlet容器之间的进行通信的接口。Servlet容器在启动一个web项目的时候，会为它创建一个ServletContext对象。每个Web项目都有唯一一个ServletContext对象，一个Web项目中的所有Servlet都共享这个ServletContext对象，它们可以通过ServletContext对象访问Servlet容器中的各种资源。

该接口提供的方法可以大致分为下面类型：

- 在Web项目范围内存取共享数据

```
1 | /**  
2 |  * 在ServletContext对象中设置一个属性。  
3 |  * @name: 属性名  
4 |  * @object: 属性值
```



```

5  */
6  public void setAttribute(String name, Object object);
7  /**
8   * 根据指定的属性名，在ServletContext对象中查找属性值。
9   */
10 public Object getAttribute(String name);
11 /**
12  * 获取ServletContext对象中的所有的属性名
13  */
14 public Enumeration getAttributeNames();
15 /**
16  * 根据指定的属性名，从ServletContext对象中删除匹配的属性。
17  */
18 public void removeAttribute(String name);

```

- 访问当前Web项目的资源

```

1  /**
2   * 获取当前Web项目的URL入口。
3   */
4  public String getContextPath();
5  /**
6   * 根据指定的参数名，获取当前Web项目范围内匹配的参数值。
7   * 在web.xml文件中，直接在<webapp>根元素下定义的<context-param>元素表示应用范围内的初始化参数。
8   */
9  public String getInitParameter(String name);
10 /**
11  * 获取当前Web项目范围内的所有参数名。
12  */
13 public Enumeration getInitParameterNames();
14 /**
15  * 获取Web应用的名字。即web.xml文件中<display-name>元素的值。
16  */
17 public String getServletContextName();
18 /**
19  * 根据指定的路径，获取一个向其他Web组件转发请求的RequestDispatcher对象。
20  */
21 public RequestDispatcher getRequestDispatcher(String path);

```

- 访问Servlet容器中的其他Web项目

```
1  /**
2  * 根据指定的uri，获取当前Servlet容器中其他Web项目的ServletContext对象。
3  */
4  public ServletContext getContext(String uripath);
```

- 访问Servlet容器的相关信息

```
1  /**
2  * 获取Servlet容器的名字和版本。
3  */
4  public String getServerInfo();
5  /**
6  * 获取Servlet容器支持的Java Servlet API支持的主版本号。
7  */
8  public int getMajorVersion();
9  /**
10 * 获取Servlet容器支持的Java Servlet API支持的次版本号。
11 */
12 public int getMinorVersion();
```

- 访问服务器的文件系统资源

```
1  /**
2  * 获取一个映射到指定路径的URL。
3  */
4  public URL getResource(String path) throws MalformedURLException;
5  /**
6  * 获取一个用于读取参数指定的文件的输入流。
7  */
8  public InputStream getResourceAsStream(String path);
9  /**
10 * 根据指定的虚拟路径，获取文件系统中一个真实的路径。
11 */
12 public String getRealPath(String path);
13 /**
14 * 获取指定文件的MIME类型。
15 */
16 public String getMimeType(String file);
```

- 输出log

```
1  /**
2   * 向Servlet日志文件中写日志。
3   */
4   public void log(String msg);
5   /**
6   * 向Servlet日志文件中写错误日志和异常的堆栈信息。
7   */
8   public void log(String message, Throwable throwable);
```

ServletContextApi.java测试了ServletContext中的大部分方法

ServletConfig接口中定义了getServletContext () 方法。HttpServlet类继承GenericServlet类，而GenericServlet类实现了ServletConfig接口，因此在HttpServlet类或GenericServlet类以及子类中都可以直接调用getServletContext () 方法，从而得到当前Web应用的ServletContext对象。

四、Servlet的生命周期

javax.servlet.Servlet接口提供了三个方法，它们分别代表了Servlet的三个生命周期阶段，具体如下：

- init(): 初始化阶段。
- service(): 运行时阶段。
- destroy(): 销毁阶段。

Servlet的初始化和销毁只会发生一次，因此init()方法和destroy()方法只会被调用一次，而service()方法可以调用多次，取决于客户端的请求次数。

1. 初始化阶段

该阶段包含如下几个小步骤：

- Servlet容器加载Servlet，把它的.class文件中的数据读入内存。Servlet容器加载Servlet有两种方式，一种是Servlet容器启动时加载，还有一种是第一次调用Servlet时再加载。两种加载方式都是通过在web.xml中配置：

```

<servlet>
  <servlet-name>first-servlet</servlet-name>
  <servlet-class>com.cheer.FirstServlet</servlet-class>
  <!-- 表示Servlet是否会被自动加载和加载优先级，数值大于0表示自动加载并且数值越小表示加载优先级越高 -->
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>first-servlet</servlet-name>
  <url-pattern>/firstServlet</url-pattern>
</servlet-mapping>

<servlet>
  <servlet-name>HttpServletRequestAPI</servlet-name>
  <servlet-class>com.cheer.api.HttpServletRequestApi</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>HttpServletRequestAPI</servlet-name>
  <url-pattern>/HttpServletRequestAPI</url-pattern>
</servlet-mapping>

```

自动加载

调用时加载

例子 演示了自动加载和调用加载，当Tomcat启动时，会发现输出MyServlet1和MyServlet2加载的信息，而且MyServlet1在前面，而MyServlet3并没有输出加载信息。这是因为例子中MyServlet1和MyServlet2配置的自动加载，而且MyServlet1的加载优先级高于MyServlet2，MyServlet3配置的调用加载，当我们输入URL: <http://localhost:8080/servlet/MyServlet3>，就会在控制台输出MyServlet加载的信息。

- Servlet容器创建ServletConfig对象。该对象包含了Servlet的初始化信息，比如初始化参数。此外，该对象还与当前Web应用的ServletContext对象关联。比如例子 [ServletContextApi.java](#)测试了ServletContext中的大部分方法。
- Servlet容器创建Servlet对象。
- Servlet容器调用Servlet对象的init方法。

2. 运行时阶段

这是Servlet生命周期中最重要的阶段，该阶段Servlet随时响应客户端的请求。当Servlet容器接收到访问特定Servlet的客户端请求时，Servlet容器创建针对这个请求的ServletRequest对象和ServletResponse对象，接着调用这个Servlet对象的service方法，service方法从ServletRequest对象获取客户端请求信息并处理该请求，通过ServletResponse对象生成响应结果。值得注意的是，当Servlet容器把Servlet的响应结果发送给客户端后，就会销毁ServletRequest对

象和ServletResponse对象。

3. 销毁阶段

当Web应用被终止时，Servlet容器会先调用Web应用中所有Servlet对象的destroy()方法，然后再销毁这些Servlet对象。在destroy()方法的实现中，可以释放Servlet所占用的资源（例如关闭文件输入流和输出流，关闭与数据库的连接等）。此外，Servlet容器还会销毁与Servlet对象关联的ServletConfig对象。

4. 演示案例

参考 [Servlet生命周期演示](#)，当我们在浏览器多次访问ServletLife时，会发现initCount一直是1，证明了init方法只被调用了一次。而serviceCount随着浏览器访问次数一直在增加，证明了service方法调用次数取决于客户端请求次数；destoryCount一直是0，直到我们手动停止Tomcat，会发现Tomcat控制台输出了destroy销毁信息，证明了destroy方法也只会调用一次。

5. Web应用范围内共享数据

要在Web应用范围内共享数据，可以利用ServletContext对象，该对象提供了下面的方法来存取共享数据：

```
1  /**
2   * 在ServletContext对象中设置一个属性。
3   * @name: 属性名
4   * @object: 属性值
5   */
6  public void setAttribute(String name, Object object);
7  /**
8   * 根据指定的属性名，在ServletContext对象中查找属性值。
9   */
10 public Object getAttribute(String name);
11 /**
12  * 根据指定的属性名，从ServletContext对象中删除匹配的属性。
13  */
14 public void removeAttribute(String name);
```

6. 练习

设计一个统计网站访问次数的功能，要求：

- 网站提供多个请求路径。
- 当客户端访问通过任何请求路径访问网站时，都显示：您好，您是本站第xx位访问者！
- 网站有admin，admin拥有一个请求路径重置网站访问次数。

五、监听器javax.servlet.ServletContextListener接口

该接口可以用来监听ServletContext对象的生命周期。当Servlet容器启动或终止时，会触发ServletContextEvent事件，该事件由ServletContextListener来处理，为此ServletContextListener接口提供了两个方法：

```
1  /**
2   * 当Servlet容器启动时调用该方法，调用完该方法后再对Filter过滤器初始化，并且对自动加
3   载的Servlet进行初始化。
4   */
5   public void contextInitialized ( ServletContextEvent sce );
6   /**
7   * 当Servlet容器终止Web应用时调用该方法，在调用该方法之前，Servlet容器会销毁所有的Fi
8   lter和Servlet。
   */
   public void contextDestroyed ( ServletContextEvent sce );
```

当我们自定义一个监听器的时候，必须实现该接口。

我们来实现一个功能，统计网站访问次数，前面练习已经做过，但是当Tomcat重启之后，这个访问次数又归零，为了解决这个问题，我们需要把访问次数存进数据库。此时最理想的做法如下：

- 设计一个表用来存储访问次数，并插入一条记录，初始化值为0。

```
1  create table web_access_count(access_count int);
2  insert into web_access_count values(0);
```

- 设计一个监听器类，AccessCountListener。
- 在web.xml中配置该监听器。
- 当Tomcat启动时，在监听器的contextInitialized的方法中从数据库中获取原来的访问次数，并把它设置成web应用范围内共享。这样所有的Servlet被请求时都可以获取该值，然后加1

显示在网页。

- 当Tomcat终止运行时，在监听器的contextDestroyed方法中，把当前访问次数存进数据库。

例子详细代码。

六、禁止页面被客户端缓存

很多浏览器为了能够快速向用户展示请求的页面，会把来自服务端的网页存放在客户端的缓存中，如果用户多次请求同一个页面，并且客户端缓存中存在该页面，那么浏览器会直接从缓存中获取该页面，不需要从服务器获取。

但是浏览器的缓存机制一般只适合保存服务器端的静态页面或者一些不包含敏感数据的网页，下面的情景不适合客户端缓存：

- 网页包含随时会被更新的内容，此时如果客户端缓存，有可能浏览器展示的是过期的网页。
- 网页中包含敏感信息，比如账号、Email内容等，此时如果客户端缓存，有可能带来安全问题。

上面两种场景，我们需要禁止浏览器缓存，此时我们可以通过HttpServletResponse对象来禁止，有三种方式：

```
1  /**
2   * 适合采用HTTP/1.0的浏览器，HTTP/1.0采用非持久连接，一个TCP连接只传送一个Web对象
3   */
4   response.addHeader("Pragma", "no-cache");
5   /**
6   * 适合采用HTTP/1.1的浏览器，HTTP/1.1采用持久连接，一个TCP连接可以传送多个Web对象
7   */
8   response.setHeader("Cache-Control", "no-cache");
9   /**
10  * 适合采用HTTP/1.0的浏览器，也适合采用HTTP/1.1的浏览器，适合所有浏览器，推荐这种方式
11  来禁用客户端缓存
12  */
    response.setHeader("Expires", 0);
```

七、过滤器javax.servlet.Filter

1. 过滤器介绍

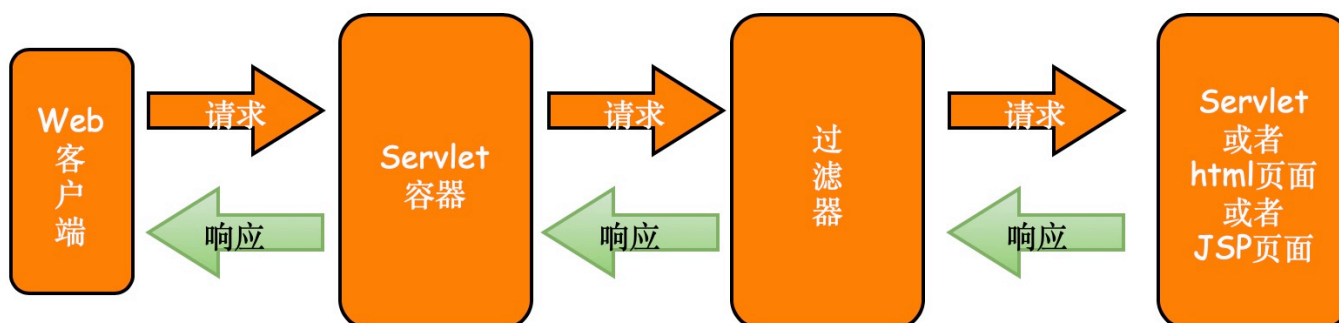
不同的Web组件在响应客户端请求的时候，有可能会完成一些共同的操作，例如身份验证、加密、日志记录等等。如果我们在每个Web组件中都写上这些共同的操作，就会导致代码重复。

此时，过滤器就会派上用场，过滤器能够对部分客户端请求先进行预处理，然后再把请求转发给各自的Web组件，等Web组件生成响应结构后，过滤器也能够对响应结果进行处理(比如检查、修改等操作)，然后再把处理后的结果发送给客户端。

上升到代码层面，过滤器能够对Servlet容器传递给Web组件的ServletRequest和ServletResponse对象进行处理，过滤器本身并不能产生ServletRequest和ServletResponse对象，它只为Web组件提供滤过功能：

- 在Web组件被调用之前检查ServletRequest对象，修改请求头和请求正文的内容，或者对请求进行预处理操作。
- Web组件被调用之后检查ServletResponse对象，修改响应头和响应正文。

这里的Web组件可以是Servlet、JSP页面或者html页面。



2. 过滤器特点

- 过滤器可以检查ServletRequest和ServletResponse对象，并且利用ServletRequestWrapper和ServletResponseWrapper类来修改ServletRequest和ServletResponse对象。
- 可以再web.xml中为过滤器映射指定的URL，当客户端访问指定的URL时才会触发过滤器。
- 多个过滤器可以串联在一起，进行协同过滤。

3. 创建过滤器

所有的自定义过滤器都必须实现javax.servlet.Filter接口，这样自定义过滤器必须实现该接口的三个方法：

```

1  /**
2   * 过滤器的初始化方法。
3   * 当Web应用启动时，Servlet容器先创建包含了过滤器配置信息的FilterConfig对象，
4   * 然后创建Filter对象，接着调用Filter对象的init (FilterConfig config) 方法。
5   * 在该方法中可通过config参数来读取web.xml文件中为过滤器配置的初始化参数。
6   */
7   public void init(FilterConfig filterConfig) throws ServletException;
8   /**
9   * 该方法完成实际的过滤操作。
10  * 当客户请求访问的URL与为过滤器映射的URL匹配时，
11  * Servlet容器将先调用过滤器的doFilter () 方法。
12  * FilterChain参数用于访问后续过滤器或者Web组件。
13  */
14  public void doFilter ( ServletRequest request, ServletResponse response,
15  FilterChain chain ) throws IOException, ServletException;
16  /**
17  * Servlet容器在销毁过滤器对象前调用该方法，在这个方法中可以释放过滤器占用的资源。
18  */
19  public void destroy();

```

过滤器由Servlet容器创建，它的生命周期如下：

- 初始化阶段: 当Web应用启动时，Servlet容器会加载过滤器类，创建过滤器配置对象（FilterConfig）和过滤器对象，并调用过滤器对象的init（FilterConfig config）方法。
- 运行阶段: 当客户请求访问的URL与为过滤器映射的URL匹配时，Servlet容器将先调用过滤器的doFilter()方法。
- 销毁阶段: 当Web应用终止时，Servlet容器会先调用过滤器对象的destroy()方法，然后销毁过滤器对象。

我们利用过滤器来实现下面的功能：

- 登陆黑名单，当登陆账号在黑名单列表中，我们将显示: 该账号已经被禁用。

实现过程：

- 设计一个登陆页面login_1.html，用来给用户登陆。
- 设计一个过滤器LoginFilter，过滤处于黑名单中的账号。
- 设计登陆Web组件LoginServlet。
- 在web.xml中配置过滤器和Servlet。

完整案例代码

4. 多个过滤器串联

当Web项目中存在多个过滤器的时候，可以把它们串联起来协同工作。Servlet容器根据过滤器在web.xml中配置的先后顺序，依次调用它们的doFilter方法。假设有2个过滤器MyFilter1和MyFilter2，并且MyFilter1在MyFilter2前面调用，它们的doFilter方法结构如下：

```
1 //TODO: 执行chain.doFilter()之前的代码
2 chain.doFilter();
3 //TODO: 执行chain.doFilter()之后的代码
```

案例代码

启动Tomcat，在浏览器中输入"/"，可以看到控制台输出：

```
1 执行MyFilter1中的doFilter中...
2 执行MyFilter2中的doFilter中...
3 执行MyFilter2中的doFilter完毕...
4 执行MyFilter1中的doFilter完毕...
```

八、用Annotation标注配置Servlet和Filter

从Servlet3开始，为了简化Java Web开发，可以不用在web.xml中配置Web组件，比如配置Servlet、过滤器、监听器等。可以直接在相关的类中用Annotation注解来配置发布信息。

注意: Servlet3以下不支持注解方式配置Servlet。

1. 注解方式配置Servlet

利用注解@WebServlet，@WebServlet还标注了一些属性：

- name: 指定Servlet的名字，相当于，如果没有指定，则用Servlet类的完整类名。
- urlPatterns: 指定Servlet的URL匹配模式，相当于。
- loadOnStartup: 设置Servlet的加载顺序，相当于。
- initParams: 指定Servlet的初始化参数，相当于。
- asyncSupported: 声明Servlet是否支持异步处理，相当于。
- description: 设置Servlet的描述信息，相当于。
- displayName: 设置Servlet的显示名，相当于。

其中，urlPatterns属性有2种设置方式：

```
1 //第一种方式
2 @WebServlet("/myServlet")
3 //第二种方式
4 @WebServlet(urlPatterns={"/myServlet1","/myServlet2"})
5 @WebServlet(urlPatterns={"/myServlet1"})
```

一般我们习惯第一种方式

我们用Annotation方式开发一个Servlet，设置成自动加载，带有两个初始化参数，映射2个url。
[完整例子代码](#)。

2. 注解方式配置监听

用注解方式配置监听很简单，只需要在类上加上@WebListener即可。

```
17
18 @WebListener
19 public class WebServerTimeLogListener implements ServletContextListener {
20
```

课堂练习：

我们设计个监听器来记录服务器的启动和关闭时间：

- 创建表web_server_time_log。
- 创建监听器WebServerTimeLogListener。

3. 注解方式配置过滤器

用注解@WebFilter来配置过滤器，该标记附带以下属性：

- filterName: 指定过滤器的名字，相当于。
- urlPatterns: 指定一组待过滤的URL，相当于。
- value: 等价于urlPatterns，两者不可同时使用。
- initParams: 设置过滤器的初始化参数。相当于。
- asyncSupported: 设置过滤器是否支持异步模式，相当于。
- Description: 设置过滤器的描述信息，相当于。
- displayName: 指定过滤器的显示名，相当于。
- dispatcherTypes: 设置过滤器的调用模式，取值包括: ASYNC、ERROR、FORWARD、

INCLUDE、REQUEST，该属性可以取多个值，值之间逗号分隔。

过滤器调用模式取值介绍：

DispatcherType.ASYNC: 当待过滤的目标资源被异步访问时，Web容器会先调用该过滤器。

DispatcherType.ERROR: 当待过滤的目标资源是通过声明式异常处理机制被访问时，Web容器会先调用该过滤器。

DispatcherType.FORWARD: 当待过滤的资源是通过RequestDispatcher的forward方式被访问时，也就是通过请求转发的方式被访问时，Web容器会先调用该过滤器。

DispatcherType.INCLUDE: 当待过滤的资源是通过RequestDispatcher的include方式被访问时，也就是通过请求包含的方式被访问时，Web容器会先调用该过滤器。

DispatcherType.REQUEST: 当客户端直接请求访问待过滤的目标资源时，Web容器会先调用该过滤器。

我们用注解的方式来实现黑名单过滤，参考 [注解过滤器例子](#)。

九、关于http请求参数包含中文字符编码

上面例子中，当浏览器端发送的userName请求参数中包含中文字符，那么服务器端有可能会读到乱码。这是因为有些浏览器采用“ISO-88591”字符编码，如果浏览器端与服务器端对请求参数采用不同的字符编码进行解析，就会造成乱码问题。

解决方式：

- Get方式: 对读取到的请求参数进行字符编码转换。

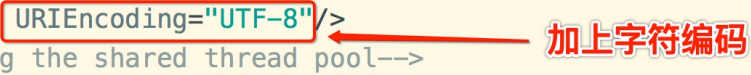
```
@Override
public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    String userName = req.getParameter("userName");
    if(userName != null) {
        userName = new String(userName.getBytes("ISO-8859-1"), "GB2312"); //把请求参数转换为GB2312编码
    }
}
```

- Post方式: 对读取到的请求参数进行字符编码转换。

```
@Override
public void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    req.setCharacterEncoding("GB2312"); //设置Web容器以GB2312编码接收数据
    String userName = req.getParameter("userName");
}
```

- 利用ServletContext对象的getRequestCharacterEncoding () 和 setRequestCharacterEncoding () 方法，来分别读取或设置当前Web应用中请求正文数据的字符编码。这两个方法是从ServletAPI4版本才开始引进的。
- 在Web服务器中配置，比如Tomcat，在server.xml中如下配置：


```
63 <Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1"
64     redirectPort="8443" URIEncoding="UTF-8"/>
65 <!-- A "Connector" using the shared thread pool-->
66 <!--
```



四种方式中，推崇第四种，前提是可以更改Web服务器配置。因为这种方式会自动把URI中的请求参数转换为UTF-8，可以说一劳永逸。