# day02

课程目标

创建Servlet

Servlet生命周期====================理解

配置Servlet

如何请求Servlet

## Servlet概述

### 1　什么是Servlet

Servlet是JavaWeb的**三大组件之一**，它属于动态资源。Servlet的作用是处理请求，服务器会把接收到的请求交给Servlet来处理，在Servlet中程序员通常需要：

* 接收请求数据；
  + 举例：登录—带着用户名和密码
* 处理请求；
  + 验证用户名和密码是否正确
* 完成响应。
  + 处理结果以响应协议格式写回客户端（浏览器）

　　例如客户端发出登录请求，或者输出注册请求，这些请求都应该由Servlet来完成处理！Servlet需要我们自己来编写，每个Servlet必须实现javax.servlet.Servlet接口。

### 2　实现Servlet的方式（由我们自己来写！）

实现Servlet有三种方式：

* 实现javax.servlet.Servlet接口；
  + 目的：理解Servlet组件创建和访问的方式及流程
* 继承javax.servlet.GenericServlet类；
* 继承javax.servlet.http.HttpServlet类；============== 常用形式

　　通常我们会去继承HttpServlet类来完成我们的Servlet，但学习Servlet还要从javax.servlet.Servlet接口开始学习。

Servlet.java

|  |
| --- |
| **public** **interface** Servlet {  **public** **void** init(ServletConfig config) **throws** ServletException;  **public** ServletConfig getServletConfig();  **public** **void** service(ServletRequest req, ServletResponse res)  **throws** ServletException, IOException;  **public** String getServletInfo();  **public** **void** destroy();  } |

### 3　创建helloservlet应用

我们开始第一个Servlet应用吧！首先在webapps目录下创建helloservlet目录，它就是我们的应用目录了，然后在helloservlet目录中创建准备JavaWeb应用所需内容：

* 创建/helloservlet/WEB-INF目录；
* 创建/helloservlet/WEB-INF/classes目录；
* 创建/helloservlet/WEB-INF/lib目录；
* 创建/helloservlet/WEB-INF/web.xml文件；

接下来我们开始准备完成Servlet，完成Servlet需要分为两步：

* 编写Servlet类；
* 在web.xml文件中配置Servlet；

HelloServlet.java

|  |
| --- |
| **public** **class** HelloServlet **implements** Servlet {  **public** **void** init(ServletConfig config) **throws** ServletException {}  **public** ServletConfig getServletConfig() {**return** **null**;}  **public** **void** destroy() {}  **public** String getServletInfo() {**return** **null**;}  **public** **void** service(ServletRequest req, ServletResponse res)  **throws** ServletException, IOException {  System.*out*.println("hello servlet!");  }  } |

　　我们暂时忽略Servlet中其他四个方法，只关心service()方法，因为它是用来处理请求的方法。我们在该方法内给出一条输出语句！

web.xml（下面内容最好记住）

|  |
| --- |
| <servlet>  <servlet-name>hello</servlet-name>  <servlet-class>com.hpit.servlet.HelloServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>hello</servlet-name>  <url-pattern>/helloworld</url-pattern>  </servlet-mapping> |

在web.xml中配置Servlet的目的其实只有一个，就是把访问路径与一个Servlet绑定到一起，上面配置是把访问路径：“/helloworld”与“com.hpit.servlet.HelloServlet”绑定到一起。

* <servlet>：指定HelloServlet这个Servlet的名称为hello；
* <servlet-mapping>：指定/helloworld访问路径所以访问的Servlet名为hello。

<servlet>和<servlet-mapping>通过<servlet-name>这个元素关联在一起了！

## Servlet接口

### 1 Servlet的生命周期

所谓xxx的生命周期，就是说xxx的出生、服务，以及死亡。Servlet生命周期也是如此！与Servlet的生命周期相关的方法有：

* void init(ServletConfig)；
* void service(ServletRequest,ServletResponse)；
* void destroy()；

#### 1.1　Servlet的init( )

服务器会在Servlet第一次被访问时创建Servlet，或者是在服务器启动时创建Servlet。如果服务器启动时就创建Servlet，那么还需要在web.xml文件中配置。也就是说默认情况下，Servlet是在第一次被访问时由服务器创建的。

而且一个Servlet类型，服务器只创建一个实例对象，例如在我们首次访问http://localhost:8080/helloservlet/helloworld时，服务器通过“/helloworld”找到了绑定的Servlet名称为com.hpit.servlet.HelloServlet，然后服务器查看这个类型的Servlet是否已经创建过，如果没有创建过，那么服务器才会通过反射来创建HelloServlet的实例。当我们再次访问http://localhost:8080/helloservlet/helloworld时，服务器就不会再次创建HelloServlet实例了，而是直接使用上次创建的实例。

在Servlet被创建后，服务器会马上调用Servlet的void init(ServletConfig)方法。Servlet的生命周期中，这个方法只会被调用一次。

我们可以把一些对Servlet的初始化工作放到init方法中！

#### 1.2　Servlet的service( )

　　当服务器每次接收到请求时，都会去调用Servlet的service()方法来处理请求。服务器接收到一次请求，就会调用service() 方法一次，所以service()方法是会被调用多次的。正因为如此，所以我们需要把处理浏览器请求的代码写到service()方法中！

#### 1.3　Servlet的destroy( )

　　Servlet是不会轻易销毁的，通常都是在服务器关闭时Servlet才会离去！在服务器被关闭时，服务器会去销毁Servlet，在销毁Servlet之前服务器会先去调用Servlet的destroy()方法，我们可以把Servlet的被销毁前要执行的代码放到destroy()方法中，例如对某些资源的释放等代码放到destroy()方法中。

1.4　测试生命周期方法

修改HelloServlet如下，然后再去访问http://localhost:8080/helloservlet/helloworld

|  |
| --- |
| **public** **class** HelloServlet **implements** Servlet {  **public** **void** init(ServletConfig config) **throws** ServletException {  System.*out*.println("Servlet被创建了！");  }  **public** ServletConfig getServletConfig() {**return** **null**;}  **public** **void** destroy() {  System.*out*.println("Servlet要离去了！");  }  **public** String getServletInfo() {**return** **null**;}  **public** **void** service(ServletRequest req, ServletResponse res)  **throws** ServletException, IOException {  System.*out*.println("hello servlet!");  }  } |

在首次访问HelloServlet时，init方法会被执行，而且也会执行service方法。再次访问时，只会执行service方法，不再执行init方法。在关闭Tomcat时会调用destroy方法。

### 2　Servlet接口相关类型

在Servlet接口中还存在三个我们不熟悉的类型：

* ServletRequest：service() 方法的参数，它表示请求对象，它封装了所有与请求相关的数据，它是由服务器创建的；
* ServletResponse：service()方法的参数，它表示响应对象，在service()方法中完成对客户端的响应需要使用这个对象；
* ServletConfig：init()方法的参数，它表示Servlet配置对象，它对应Servlet的配置信息，那对应web.xml文件中的<servlet>元素。

#### 2.1　ServletRequest和ServletResponse（后面详细讲解这两个对象）

ServletRequest：指http请求协议

ServletResponse：http响应协议

#### 2.1　ServletConfig===============理解

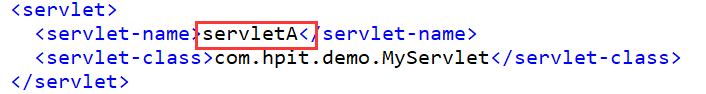
作用：创建serlvet对象时，配置servlet需要的配置信息。类似.Properties文件中

应用场景：解决中文乱码的问题，指定字符编码，编码配置节点中

数据库连接信息配置

框架资源文件配置

ServletConfig对象对应web.xml文件中的<servlet>元素。例如你想获取当前Servlet在web.xml文件中的配置名，那么可以使用servletConfig.getServletName()方法获取！



ServletConfig对象是由服务器创建的，然后传递给Servlet的init()方法，你可以在init()方法中使用它！

* String getServletName()：获取Servlet在web.xml文件中的配置名称，即<servlet-name>指定的名称；
* ServletContext getServletContext()：用来获取ServletContext对象，ServletContext会在后面讲解；
* String getInitParameter(String name)：用来获取在web.xml中配置的初始化参数，通过参数名来获取参数值；
* Enumeration getInitParameterNames()：用来获取在web.xml中配置的所有初始化参数名称；

在<servlet>元素中还可以配置初始化参数：

|  |
| --- |
| <servlet>  <servlet-name>servletA</servlet-name>  <servlet-class>com.hpit.demo.MyServlet</servlet-class>  <init-param>  <param-name>paramName1</param-name>  <param-value>paramValue1</param-value>  </init-param>  <init-param>  <param-name>paramName2</param-name>  <param-value>paramValue2</param-value>  </init-param>  </servlet> |

在servletA中，可以使用ServletConfig对象的getInitParameter()方法来获取初始化参数，例如：

String value1 = servletConfig.getInitParameter(“paramName1”);//获取到paramValue1

## GenericServlet

### 1　GenericServlet概述

GenericServlet是Servlet接口的实现类，我们可以通过继承GenericServlet来编写自己的Servlet。下面是GenericServlet类的源代码：

GenericServlet.java

|  |
| --- |
| **public** **abstract** **class** GenericServlet **implements** Servlet, ServletConfig,  java.io.Serializable {  **private** **static** **final** **long** *serialVersionUID* = 1L;  **private** **transient** ServletConfig config;  **public** GenericServlet() {}  @Override  **public** **void** destroy() {}  @Override  **public** String getInitParameter(String name) {  **return** getServletConfig().getInitParameter(name);  }  @Override  **public** Enumeration<String> getInitParameterNames() {  **return** getServletConfig().getInitParameterNames();  }  @Override  **public** ServletConfig getServletConfig() {  **return** config;  }  @Override  **public** ServletContext getServletContext() {  **return** getServletConfig().getServletContext();  }  @Override  **public** String getServletInfo() {  **return** "";  }  @Override  **public** **void** init(ServletConfig config) **throws** ServletException {  //就不能重写带参init(),覆盖父类init()方法体，config对象丢了  **this**.config = config;  **this**.init();  }  //重写无参init(),  **public** **void** init() **throws** ServletException {}  **public** **void** log(String msg) {  getServletContext().log(getServletName() + ": " + msg);  }  **public** **void** log(String message, Throwable t) {  getServletContext().log(getServletName() + ": " + message, t);  }  @Override  **public** **abstract** **void** service(ServletRequest req, ServletResponse res)  **throws** ServletException, IOException;  @Override  **public** String getServletName() {  **return** config.getServletName();  }  } |

### 2　GenericServlet的init()方法

在GenericServlet中，定义了一个ServletConfig config实例变量，并在init(ServletConfig)方法中把参数ServletConfig赋给了实例变量。然后在该类的很多方法中使用了实例变量config。

如果子类覆盖了GenericServlet的init(StringConfig)方法，那么this.config=config这一条语句就会被覆盖了，也就是说GenericServlet的实例变量config的值为null，那么所有依赖config的方法都不能使用了。如果真的希望完成一些初始化操作，那么去覆盖GenericServlet提供的init()方法，它是没有参数的init()方法，它会在init(ServletConfig)方法中被调用。

### 3　实现了ServletConfig接口

　　GenericServlet还实现了ServletConfig接口，所以可以直接调用getInitParameter()、getServletContext()等ServletConfig的方法。

## HttpServlet

### 1　HttpServlet概述

HttpServlet类是GenericServlet的子类，它提供了对HTTP请求的特殊支持，所以通常我们都会通过继承HttpServlet来完成自定义的Servlet。

### 2　HttpServlet覆盖了service()方法

HttpServlet类中提供了service(HttpServletRequest,HttpServletResponse)方法，这个方法是HttpServlet自己的方法，不是从Servlet继承来的。在HttpServlet的service(ServletRequest,ServletResponse)方法中会把ServletRequest和ServletResponse强转成HttpServletRequest和HttpServletResponse，然后调用service(HttpServletRequest,HttpServletResponse)方法，这说明子类可以去覆盖service(HttpServletRequest,HttpServletResponse)方法即可，这就不用自己去强转请求和响应对象了。

其实子类也不用去覆盖service(HttpServletRequest,HttpServletResponse)方法，因为HttpServlet还要做另一步简化操作，下面会介绍。

HttpServlet.java

|  |
| --- |
| **public** **abstract** **class** HttpServlet **extends** GenericServlet {    **protected** **void** service(HttpServletRequest req, HttpServletResponse resp)  **throws** ServletException, IOException {  ……  //已经将请求对象和响应对象向下转型HTTP协议需要对象。程序员直接用，不用转换了  }  @Override  **public** **void** service(ServletRequest req, ServletResponse res)  **throws** ServletException, IOException {  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);  }  ……  } |

### 3　doGet()和doPost()

在HttpServlet的service(HttpServletRequest,HttpServletResponse)方法会去判断当前请求是GET还是POST，如果是GET请求，那么会去调用本类的doGet()方法，如果是POST请求会去调用doPost()方法，这说明我们在子类中去覆盖doGet()或doPost()方法即可。

|  |
| --- |
| **public** **class** AServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  System.*out*.println("hello doGet()...");  }  } |
| **public** **class** BServlet **extends** HttpServlet {  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  System.*out*.println("hello doPost()...");  }  } |

## Servlet细节

* 不要在Servlet中创建成员变量！创建局部变量即可！
* 可以创建无状态成员！
* 可以创建有状态的成员，但状态必须为只读的！

### 1　Servlet与线程安全

因为一个类型的Servlet只有一个实例对象，那么就有可能会现同一个Servlet同时处理多个请求，那么Servlet是否为线程安全的呢？答案是：“不是线程安全的”。这说明Servlet的工作效率很高，但也存在线程安全问题！

所以我们不应该在Servlet中创建成员变量，因为可能会存在一个线程对这个成员变量进行写操作，另一个线程对这个成员变量进行读操作。

### 2　让服务器在启动时就创建Servlet

默认情况下，服务器会在某个Servlet第一次收到请求时创建它。也可以在web.xml中对Servlet进行配置，使服务器启动时就创建Servlet。

|  |
| --- |
| <servlet>  <servlet-name>hello1</servlet-name>  <servlet-class>com.hpit.servlet.Hello1Servlet</servlet-class>  <load-on-startup>0</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>hello1</servlet-name>  <url-pattern>/hello1</url-pattern>  </servlet-mapping>  <servlet>  <servlet-name>hello2</servlet-name>  <servlet-class> com.hpit.servlet.Hello2Servlet</servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>hello2</servlet-name>  <url-pattern>/hello2</url-pattern>  </servlet-mapping>  <servlet>  <servlet-name>hello3</servlet-name>  <servlet-class> com.hpit.servlet.Hello3Servlet</servlet-class>  <load-on-startup>2</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>hello3</servlet-name>  <url-pattern>/hello3</url-pattern>  </servlet-mapping> |

在<servlet>元素中配置<load-on-startup>元素可以让服务器在启动时就创建该Servlet，其中<load-on-startup>元素的值必须是大于等于0的整数，它的使用是服务器启动时创建Servlet的顺序。上例中，根据<load-on-startup>的值可以得知服务器创建Servlet的顺序为Hello1Servlet、Hello2Servlet、Hello3Servlet。

## ServletContext（重要）

一个项目只有一个ServletContext对象！

我们可以在N多个Servlet中来获取这个唯一的对象，使用它可以给多个Servlet传递数据！

这个对象在Tomcat启动时就创建，在Tomcat关闭时才会销毁！

### ServletContext概述

指实际存在的一个Web应用，所以web程序发布服务器上面，服务器产生一个ServletContext

服务器会为每个应用创建一个ServletContext对象：

* ServletContext对象的创建是在服务器启动时完成的；init()初始化的方法
* ServletContext对象的销毁是在服务器关闭时完成的。destroy()

　　 ServletContext对象的作用是在整个Web应用的动态资源之间共享数据！例如在AServlet中向ServletContext对象中保存一个值，然后在BServlet中就可以获取这个值，这就是共享数据了。

### 2　获取ServletContext

* ServletConfig#getServletContext()；
* GenericServlet#getServletContext();
* HttpSession#getServletContext()
* ServletContextEvent#getServletContext()

在Servlet中获取ServletContext对象：

* 在void init(ServletConfig config)中：ServletContext context = config.getServletContext();，ServletConfig类的getServletContext()方法可以用来获取ServletContext对象；

在GenericeServlet或HttpServlet中获取ServletContext对象：

* GenericServlet类有getServletContext()方法，所以可以直接使用this.getServletContext()来获取；

|  |
| --- |
| public class MyServlet implements Servlet {  public void init(ServletConfig config) {  ServletContext context = config.getServletContext();  }  …  } |
| public class MyServlet extends HttpServlet {  public void doGet(HttpServletRequest request, HttpServletResponse response) {  ServletContext context = this.getServletContext();  }  } |

### 3　域对象的功能

ServletContext是JavaWeb四大域对象之一：

* PageContext；
* ServletRequest；
* HttpSession；
* ServletContext；

所有域对象都有存取数据的功能，因为域对象内部有一个Map，用来存储数据，下面是ServletContext对象用来操作数据的方法：

* void setAttribute(String name, Object value)：用来存储一个对象，也可以称之为存储一个域属性，例如：servletContext.setAttribute(“xxx”, “XXX”)，在ServletContext中保存了一个域属性，域属性名称为xxx，域属性的值为XXX。请注意，如果多次调用该方法，并且使用相同的name，那么会覆盖上一次的值，这一特性与Map相同；
* Object getAttribute(String name)：用来获取ServletContext中的数据，当前在获取之前需要先去存储才行，例如：String value = (String)servletContext.getAttribute(“xxx”);，获取名为xxx的域属性；
* void removeAttribute(String name)：用来移除ServletContext中的域属性，如果参数name指定的域属性不存在，那么本方法什么都不做；
* Enumeration getAttributeNames()：获取所有域属性的名称；

### 4　获取应用初始化参数

* Servlet也可以获取初始化参数，但它是局部的参数；也就是说，一个Servlet只能获取自己的初始化参数，不能获取别人的，即初始化参数只为一个Servlet准备！
* 可以配置公共的初始化参数，为所有Servlet而用！这需要使用ServletContext才能使用！

还可以使用ServletContext来获取在web.xml文件中配置的应用初始化参数！注意，应用初始化参数与Servlet初始化参数不同：

web.xml

|  |
| --- |
| <web-app ...>  ...  <context-param>  <param-name>paramName1</param-name>  <param-value>paramValue1</param-value>  </context-param>  <context-param>  <param-name>paramName2</param-name>  <param-value>paramValue2</param-value>  </context-param>  </web-app> |
| ServletContext context = **this**.getServletContext();  String value1 = context.getInitParameter("paramName1");  String value2 = context.getInitParameter("paramName2");  System.*out*.println(value1 + ", " + value2);    Enumeration names = context.getInitParameterNames();  **while**(names.hasMoreElements()) {  System.*out*.println(names.nextElement());  } |

### 5　练习：访问量统计

一个项目中所有的资源被访问都要对访问量进行累加！

创建一个int类型的变量，用来保存访问量，然后把它保存到ServletContext的域中，这样可以保证所有的Servlet都可以访问到！

* 最初时，ServletContext中没有保存访问量相关的属性；
* 当本站第一次被访问时，创建一个变量，设置其值为1；保存到ServletContext中；
* 当以后的访问时，就可以从ServletContext中获取这个变量，然后在其基础之上加１。
* 获取ServletContext对象，查看是否存在名为count的属性，如果存在，说明不是第一次访问，如果不存在，说明是第一次访问；
* 第一次访问：调用Servletcontext的setAttribute()传递一个属性，名为count，值为1；
* 第2~N次访问：调用ServletContext的getAttribute()方法获取原来的访问量，给访问量加1，再调用Servletcontext的setAttribute()方法完成设置。

相信各位一定见过很多访问量统计的网站，即“本页面被访问过XXX次”。因为无论是哪个用户访问指定页面，都会累计访问量，所以这个访问量统计应该是整个项目共享的！很明显，这需要使用ServletContext来保存访问量。

|  |
| --- |
| //获取ServletContext对象  ServletContext application = **this**.getServletContext();  //获取ServletContext对象中的count属性  Integer count = (Integer)application.getAttribute("count");  **if**(count == **null**) {  //如果在ServletContext中不存在count属性，那么设置count的值为1，表示第一次被访问。  count = 1;  } **else** {  //如果在ServletContext中存在count属性，说明以前被访问过，那么让count在原来的基础上加1。  count++;  }  response.setContentType("text/html;charset=utf-8");  //向客户端响应本页面被访问的次数。  response.getWriter().print("<h1>本页面一共被访问" + count + "次！</h1>");  //保存count的值到ServletContext对象中。  application.setAttribute("count", count); |

## 作业

完成课堂案例

理解Servlet调用过程，设计生命周期，

ServletConfig基本方法使用

HttpServlet doGet和doPost调用原理

## 预习

Request和Response

Request:获取客户提交的数据

Response：重定向

响应输出方式