## 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中通常需要：

* 接收请求数据；
* 处理请求；
* 完成响应。

　　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

实现Servlet有三种方式：

* 实现javax.servlet.Servlet接口；
* 继承javax.servlet.GenericServlet类；
* 继承javax.servlet.http.HttpServlet类；

　　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（下面内容需要背下来）

|  |
| --- |
| <servlet>  <servlet-name>hello</servlet-name>  <servlet-class>cn.itcast.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“cn.itcast.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！

接下来，我们编译HelloServlet，注意，编译HelloServlet时需要导入servlet-api.jar，因为Servlet.class等类都在servlet-api.jar中。

javac -classpath F:/tomcat6/lib/servlet-api.jar -d . HelloServlet.java

然后把HelloServlet.class放到/helloworld/WEB-INF/classes/目录下，然后启动Tomcat，在浏览器中访问：http://localhost:8080/helloservlet/helloworld即可在控制台上看到输出！

* /helloservlet/WEB-INF/classes/cn/itcast/servlet/HelloServlet.class；

## Servlet接口

###### 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的出生

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cn.itcast.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。请记住， Servlet出生后马上就会调用init()方法，而且一个Servlet的一生。这个方法只会被调用一次。这好比小孩子出生后马上就要去剪脐带一样，而且剪脐带一生只有一次。

我们可以把一些对Servlet的初始化工作放到init方法中！

1.2　Servlet服务

　　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，所以我们才需要把处理请求的代码写到service()方法中！

1.3　Servlet的离去

　　Servlet是不会轻易离去的，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的临终遗言放到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（第五天会详细讲解这两个对象）

ServletRequest和ServletResponse是Servlet#service() 方法的两个参数，一个是请求对象，一个是响应对象，可以从ServletRequest对象中获取请求数据，可以使用ServletResponse对象完成响应。你以后会发现，这两个对象就像是一对恩爱的夫妻，永远不分离，总是成对出现。

ServletRequest和ServletResponse的实例由服务器创建，然后传递给service()方法。如果在service() 方法中希望使用HTTP相关的功能，那么可以把ServletRequest和ServletResponse强转成HttpServletRequest和HttpServletResponse。这也说明我们经常需要在service()方法中对ServletRequest和ServletResponse进行强转，这是很心烦的事情。不过后面会有一个类来帮我们解决这一问题的。

HttpServletRequest方法：

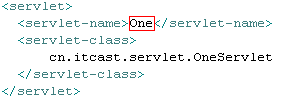
* String getParameter(String paramName)：获取指定请求参数的值；
* String getMethod()：获取请求方法，例如GET或POST；
* String getHeader(String name)：获取指定请求头的值；
* void setCharacterEncoding(String encoding)：设置请求体的编码！因为GET请求没有请求体，所以这个方法只只对POST请求有效。当调用request.setCharacterEncoding(“utf-8”)之后，再通过getParameter()方法获取参数值时，那么参数值都已经通过了转码，即转换成了UTF-8编码。所以，这个方法必须在调用getParameter()方法之前调用！

HttpServletResponse方法：

* PrintWriter getWriter()：获取字符响应流，使用该流可以向客户端输出响应信息。例如response.getWriter().print(“<h1>Hello JavaWeb!</h1>”)；
* ServletOutputStream getOutputStream()：获取字节响应流，当需要向客户端响应字节数据时，需要使用这个流，例如要向客户端响应图片；
* void setCharacterEncoding(String encoding)：用来设置字符响应流的编码，例如在调用setCharacterEncoding(“utf-8”);之后，再response.getWriter()获取字符响应流对象，这时的响应流的编码为utf-8，使用response.getWriter()输出的中文都会转换成utf-8编码后发送给客户端；
* void setHeader(String name, String value)：向客户端添加响应头信息，例如setHeader(“Refresh”, “3;url=http://www.itcast.cn”)，表示3秒后自动刷新到http://www.itcast.cn；
* void setContentType(String contentType)：该方法是setHeader(“content-type”, “xxx”)的简便方法，即用来添加名为content-type响应头的方法。content-type响应头用来设置响应数据的MIME类型，例如要向客户端响应jpg的图片，那么可以setContentType(“image/jepg”)，如果响应数据为文本类型，那么还要台同时设置编码，例如setContentType(“text/html;chartset=utf-8”)表示响应数据类型为文本类型中的html类型，并且该方法会调用setCharacterEncoding(“utf-8”)方法；
* void sendError(int code, String errorMsg)：向客户端发送状态码，以及错误消息。例如给客户端发送404：response(404, “您要查找的资源不存在！”)。

2.1　ServletConfig

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>One</servlet-name>  <servlet-class>cn.itcast.servlet.OneServlet</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>//配置了两个初始化参数  </servlet> |

在OneServlet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  //int实现了Servlet的init(ServletConfig)方法，把参数config赋给了本类的成员config，然后再调用本类自己的无参的init()方法。  **public** **void** init(ServletConfig config) **throws** ServletException {  **this**.config = config;  **this**.init();  }  **public** **void** init() **throws** ServletException {}这个方法是GenericServlet自己的方法，而不是从Servlet继承下来的。当我们自定义Servlet时，如果想完成初始化作用就不要再重复init(ServletConfig)方法了，而是应该去重写init()方法。因为在GenericServlet中的init(ServletConfig)方法中保存了ServletConfig对象，如果覆盖了保存ServletConfig的代码，那么就不能再使用ServletConfig了。  **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 {  ……  }  @Override  **public** **void** service(ServletRequest req, ServletResponse res)  **throws** ServletException, IOException {  HttpServletRequest request;  HttpServletResponse response;  **try** {  request = (HttpServletRequest) req;  response = (HttpServletResponse) res;  //强转  } **catch** (ClassCastException e) {  **throw** **new** ServletException("non-HTTP request or response");  }  service(request, response);// 调用上面service()方法  }  ……  } |

###### 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

###### 1　Servlet与线程安全

因为一个类型的Servlet只有一个实例对象，那么就有可能会现时出一个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中便宜创建成员变量，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>cn.itcast.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>cn.itcast.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>cn.itcast.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。

###### 3　<url-pattern>

<url-pattern>是<servlet-mapping>的子元素，用来指定Servlet的访问路径，即URL。它必须是以“/”开头！

1. 可以在<servlet-mapping>中给出多个<url-pattern>，例如：

|  |
| --- |
| <servlet-mapping>  <servlet-name>AServlet</servlet-name>  <url-pattern>/AServlet</url-pattern>  <url-pattern>/BServlet</url-pattern>  </servlet-mapping> |

那么这说明一个Servlet绑定了两个URL，无论访问/AServlet还是/BServlet，访问的都是AServlet。

1. 还可以在<url-pattern>中使用通配符，所谓通配符就是星号“\*”，星号可以匹配任何URL前缀或后缀，使用通配符可以命名一个Servlet绑定一组URL，例如：

* <url-pattern>/servlet/\*<url-patter>：/servlet/a、/servlet/b，都匹配/servlet/\*；
* <url-pattern>\*.do</url-pattern>：/abc/def/ghi.do、/a.do，都匹配\*.do；
* <url-pattern>/\*<url-pattern>：匹配所有URL；

请注意，通配符要么为前缀，要么为后缀，不能出现在URL中间位置，也不能只有通配符。例如：/\*.do就是错误的，因为星号出现在URL的中间位置上了。\*.\*也是不对的，因为一个URL中最多只能出现一个通配符。

注意，通配符是一种模糊匹配URL的方式，如果存在更具体的<url-pattern>，那么访问路径会去匹配具体的<url-pattern>。例如：

|  |
| --- |
| <servlet>  <servlet-name>hello1</servlet-name>  <servlet-class>cn.itcast.servlet.Hello1Servlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>hello1</servlet-name>  <url-pattern>/servlet/hello1</url-pattern>  </servlet-mapping>  <servlet>  <servlet-name>hello2</servlet-name>  <servlet-class>cn.itcast.servlet.Hello2Servlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>hello2</servlet-name>  <url-pattern>/servlet/\*</url-pattern>  </servlet-mapping> |

　　当访问路径为http://localhost:8080/hello/servlet/hello1时，因为访问路径即匹配hello1的<url-pattern>，又匹配hello2的<url-pattern>，但因为hello1的<url-pattern>中没有通配符，所以优先匹配，即设置hello1。

###### 4　web.xml文件的继承（了解）

每个完整的JavaWeb应用中都需要有web.xml，但我们不知道所有的web.xml文件都有一个共同的父文件，它在Tomcat的conf/web.xml路径。

conf/web.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"ISO-8859-1"*?>  <web-app xmlns=*"http://java.sun.com/xml/ns/javaee"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*  *http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*  version=*"3.0"*>  <servlet>  <servlet-name>default</servlet-name>  <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>  // DefaultServlet 当访问路径不存在时，会执行该Servlet！其实我们在访问index.html时也是在执行这个Servlet。  <init-param>  <param-name>debug</param-name>  <param-value>0</param-value>  </init-param>  <init-param>  <param-name>listings</param-name>  <param-value>false</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet>  <servlet-name>jsp</servlet-name>  <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>  <init-param>  <param-name>fork</param-name>  <param-value>false</param-value>  </init-param>  <init-param>  <param-name>xpoweredBy</param-name>  <param-value>false</param-value>  </init-param>  <load-on-startup>3</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>default</servlet-name>  <url-pattern>/</url-pattern>//匹配所有URL，也就是说用户访问的URL路径没 有匹配的页面时，那么执行的就是名为default的Servlet，即 org.apache.catalina.servlets.DefaultServlet  </servlet-mapping>  <servlet-mapping>  <servlet-name>jsp</servlet-name>  <url-pattern>\*.jsp</url-pattern>//任何URL后缀为jsp的访问，都会执行名为jsp的Servlet，即org.apache.jasper.servlet.JspServlet  <url-pattern>\*.jspx</url-pattern>  </servlet-mapping>  <session-config>  <session-timeout>30</session-timeout>// session的默认超时时间为30分钟，后面讲session时再深入。  </session-config>  <!-- 这里省略了大概4000多行的MIME类型的定义,这里只给出两种MIME类型的定义 -->  //MIME类型用来标识网络上资源的媒体类型，这里举例为bmp和html两种MIME类型。  <mime-mapping>  <extension>bmp</extension>  <mime-type>image/bmp</mime-type>  </mime-mapping>  <mime-mapping>  <extension>htm</extension>  <mime-type>text/html</mime-type>  </mime-mapping>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>//在应用的web.xml中如果没有对<welcome-file-list>进行覆盖，那么默认主页为index.html、index.html、index.jsp  </web-app> |

## ServletContext

###### 1　ServletContext概述

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；
* ServletContext对象的销毁是在服务器关闭时完成的。

　　 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

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；

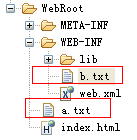
###### 4　获取应用初始化参数

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 ...>  ...  // 配置了两个应用初始化参数  <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();//获取ServletContext对象  String value1 = context.getInitParameter("paramName1");  String value2 = context.getInitParameter("paramName2");//通过参数名，获取参数值  System.*out*.println(value1 + ", " + value2);    Enumeration names = context.getInitParameterNames();//获取所有应用初始化参数名称  **while**(names.hasMoreElements()) {  System.*out*.println(names.nextElement());  } |

###### 5　获取资源相关方法



5.1　获取真实路径

还可以使用ServletContext对象来获取Web应用下的资源，例如在hello应用的根目录下创建a.txt文件，现在想在Servlet中获取这个资源，就可以使用ServletContext来获取。

* 获取a.txt的真实路径：String realPath = servletContext.getRealPath(“/a.txt”)，realPath的值为a.txt文件的绝对路径：F:\tomcat6\webapps\hello\a.txt；
* 获取b.txt的真实路径：String realPath = servletContext.getRealPath(“/WEB-INF/b.txt”)；

5.2　获取资源流

不只可以获取资源的路径，还可以通过ServletContext获取资源流，即把资源以输入流的方式获取：

* 获取a.txt资源流：InputStream in = servletContext.getResourceAsStream(“/a.txt”)；
* 获取b.txt资源流：InputStream in = servletContext.getResourceAsStream(“/WEB-INF/b.txt”)；

5.3　获取指定目录下所有资源路径

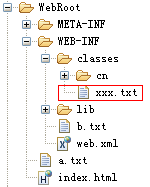
还可以使用ServletContext获取指定目录下所有资源路径，例如获取/WEB-INF下所有资源的路径：

|  |
| --- |
| Set set = context.getResourcePaths("/WEB-INF");  System.*out*.println(set); |
| [/WEB-INF/lib/, /WEB-INF/classes/, /WEB-INF/b.txt, /WEB-INF/web.xml] |

注意，本方法必须以“/”开头！！！

## 获取类路径下资源

　　这里要讲的是获取类路径下的资源，对于JavaWeb应用而言，就是获取classes目录下的资源。



|  |
| --- |
| InputStream in = **this**.getClass().getResourceAsStream("/xxx.txt");  System.*out*.println(IOUtils.*toString*(in)); |
| InputStream in = **this**.getClass().getClassLoader().getResourceAsStream("xxx.txt");  System.*out*.println(IOUtils.*toString*(in)); |

* Class类的getResourceAsStream(String path)：
* 路径以“/”开头，相对classes路径；
* 路径不以“/”开头，相对当前class文件所有路径，例如在cn.itcast.servlet.MyServlet中执行，那么相对/classes/cn/itcast/servlet/路径；
* ClassLoader类的getResourceAsStream(String path)：
* 相对classes路径；