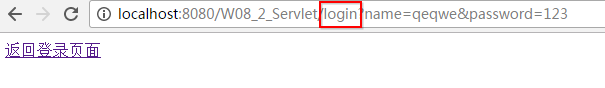
# 一 Servlet注册

由于浏览器客户端是通过URL地址访问web服务器中的资源，所以Servlet程序若想被外界访问，必须把servlet程序映射到一个URL地址上，这个工作在web.xml文件中使用<servlet>元素和<servlet-mapping>元素完成。

<servlet>元素用于注册Servlet，它包含有两个主要的子元素：<servlet-name>和<servlet-class>，分别用于设置Servlet的注册名称和Servlet的完整类名。

<servlet-mapping>元素用于映射已注册的Servlet的对外访问路径，它包含有两个子元素：<servlet-name>和<url-pattern>

****

****

<servlet>

<servlet-name>Hello</servlet-name>

<servlet-class>com.rr.servlet.HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Hello</servlet-name>

<url-pattern>/Hello</url-pattern>

</servlet-mapping>

同一个Servlet可以被映射到多个URL上，即多个<servlet-mapping>元素的<servlet-name>子元素的设置值可以是同一个Servlet的注册名。

在Servlet映射到的URL中也可以使用\*通配符，如果使用\*，只能有两种固定的格式：

一种格式是“\*.扩展名”，例如：

<url-pattern>\*.do</url-pattern>

<url-pattern>\*.action</url-pattern>

另一种格式是以正斜杠（/）开头并以“/\*”结尾，例如：

<url-pattern>/\*</url-pattern>

<url-pattern>/Hello/\*</url-pattern>

Web.xml针对web应用程序的配置文件

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* id=*"WebApp\_ID"* version=*"2.5"*>

<display-name>W07\_Servlet</display-name>

<welcome-file-list>

<!-- 当url中没有具体访问的资源时，会从该应用中找符合下面列表中名称的资源，并调用 -->

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

<!-- 访问资源时，比如 http://localhost:8080/W07\_Servlet/Hello

会从<servlet-mapping>中找匹配url-pattern的值

然后可以得到对应servlet-name

通过该name在<servlet>中找相同的<servlet-name>

如果找到，获取对应的<servlet-class>,

访问资源时，具体调用的哪个类，就可以找到

-->

<servlet>

<servlet-name>hhh</servlet-name>

<servlet-class>com.rr.servlet.HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>hhh</servlet-name>

<url-pattern>/Hello</url-pattern>

<!-- 要求匹配的资源后缀必须是.do,\*表示通配符，任意内容都可以 ,不能写成/\*.do-->

<!-- <url-pattern>\*.do</url-pattern> -->

<!-- 匹配任意资源 不能直接写\*-->

<!-- <url-pattern>/\*</url-pattern> -->

<!-- 下面写法错误 -->

<!-- <url-pattern>/a/\*.do</url-pattern> -->

</servlet-mapping>

<!-- 一个<servlet>可以对应多个<servlet-mapping> -->

<servlet-mapping>

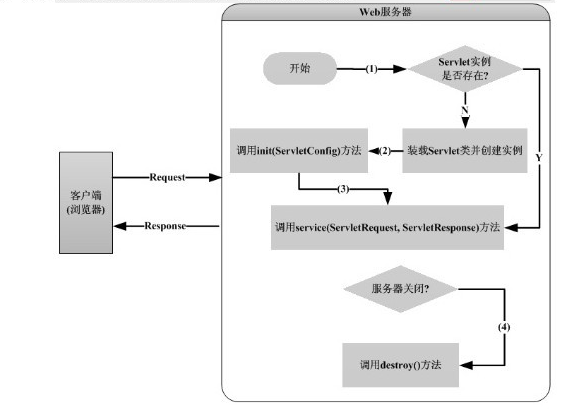
<servlet-name>hhh</servlet-name>

<url-pattern>/hello</url-pattern>

</servlet-mapping>

# 二 Servlet的生命周期

## 1生命周期



针对客户端的多次Servlet请求，通常情况下，服务器只会创建一个Servlet实例对象，也就是说一个Servlet的实例对象一旦创建，它就会驻留在内存中，为后续的其它请求服务，直至web容器退出，servlet实例对象才会销毁。

Servlet的生命周期主要有四个阶段：

1. 创建对象

对同一个servlet，只会创建一次

1. 初始化

创建对象后，init方法，只会调用一次，会获取一个ServletConfig对象

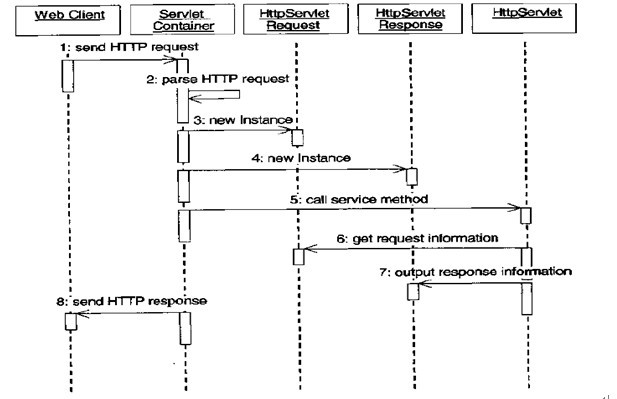
1. Service

每调用一次servlet资源，都会调用该方法。对于每次访问请求，都会创建一个新的HttpServletRequest请求对象和一个新的HttpServletResponse响应对象，然后将这两个对象作为参数传递给它调用的Servlet的service()方法，在该方法中，根据请求方式，决定调用doGet还是doPost方法

1. 销毁

关闭tomcat服务器，或者重新加载应用，都会调用destroy方法

## 2 请求流程



## 3 Servlet自动加载

<servlet>

<description></description>

<display-name>LifeServlet</display-name>

<servlet-name>LifeServlet</servlet-name>

<servlet-class>com.rr.life.LifeServlet</servlet-class>

<!-- tomcat服务器启动时，创建对应的servlet对象

值越大，优先级越高-->

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

</servlet>

# 三 Servlet的线程安全

Servlet运行多线程环境下，而且servlet是单例的

synchronized (ThreadServlet.class) {

//System.out.println("你是第" + count + "位访客");

response.getWriter().write("你是第" + count + "位访客");

try {

//休眠5秒

Thread.sleep(5000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

count++;

}

# 四 ServletConfig

一个servlet对应一个servletconfig对象

<servlet>

<description></description>

<display-name>ConfigServlet</display-name>

<servlet-name>ConfigServlet</servlet-name>

<servlet-class>com.rr.servletconfig.ConfigServlet</servlet-class>

<!-- servlet的初始化参数 -->

<init-param>

<param-name>user</param-name>

<param-value>zhangsan</param-value>

</init-param>

<init-param>

<param-name>pwd</param-name>

<param-value>12345</param-value>

</init-param>

</servlet>

在Servlet的配置文件中，可以使用一个或多个<init-param>标签为servlet配置一些初始化参数。

当servlet配置了初始化参数后，web容器在创建servlet实例对象时，会自动将这些初始化参数封装到ServletConfig对象中，并在调用servlet的init方法时，将ServletConfig对象传递给servlet。进而，程序员通过ServletConfig对象就可以得到当前servlet的初始化参数信息。

# 五 ServletContext

一个web应用对应一个ServletContext对象, 它代表当前web应用

## 1 获取web应用路径

//获取servletcontext对象

ServletContext servletContext = **this**.getServletContext();

// /W08\_2\_Servlet

System.***out***.println(servletContext.getContextPath());

//通过request对象，获取应用的路径

System.***out***.println(request.getContextPath());

## 2 web应用的初始化参数

/\*

\*<!-- 针对整个应用的初始化参数 -->

<context-param>

<param-name>AAA</param-name>

<param-value>aaa</param-value>

</context-param>

\*

\* \*/

String v1 = servletContext.getInitParameter("AAA");

System.***out***.println(v1);

## 3 ServletContext域对象

request域、session域

ServletContext servletContext = **this**.getServletContext();

//向servletcontext域中写入数据

//在整个web应用的不同的资源之间共享数据

servletContext.setAttribute("name", "zhangsan");

Dog d = **new** Dog();

d.setAge(2);

d.setName("wangcai");

servletContext.setAttribute("dog", d);

# 六 转发和重定向

转发：一个web资源收到客户端请求后，通知服务器去调用另外一个web资源进行处理，称之为请求转发。

//request域

request.setAttribute("age", 10);

//转发

request.getRequestDispatcher("/ForwardServlet2").forward(request, response);

重定向：一个web资源收到客户端请求后，通知浏览器去访问另外一个web资源，称之为请求重定向。

//重定向

//response.sendRedirect("/W08\_2\_Servlet/RedirectServlet2");

//response.sendRedirect("RedirectServlet2");

//重定向时，不能共享request域中的数据

request.setAttribute("id", 100);

//推荐如下写法

response.sendRedirect(request.getContextPath() + "/RedirectServlet2");

重定向和转发区别：

1)重定向的访问过程结束后，浏览器地址栏中显示的URL会发生改变；转发过程结束后，浏览器地址栏保持初始的URL地址不变。

2)重定向时，通过响应的结果告诉浏览器去重新发出请求访问另外一个资源；而转发是在服务器端内部将请求转发给另外一个资源。

3)转发时，调用者与被调用者之间共享相同的request对象和response对象，它们属于同一个访问请求和响应过程；而重定向时调用者与被调用者使用各自的request对象和response对象，它们属于两个独立的访问请求和响应过程。

4)重定向时，重定向的URL以“/”开头，它是相对于整个WEB站点的根目录；转发时的URL以“/”开头，它是相对于当前WEB应用程序的根目录。

5)转发只能将请求转发给同一个WEB应用中的资源；而重定向还可以重定向到同一个站点上的其他应用程序中的资源，甚至是使用绝对URL重定向到其他站点的资源。