孤傲苍狼

只为成功找方法，不为失败找借口！

javaweb学习总结(五)——Servlet开发(一)

一、Servlet简介

Servlet是sun公司提供的一门用于开发动态web资源的技术。

Sun公司在其API中提供了一个servlet接口，用户若想用发一个动态web资源(即开发一个Java程序向浏览器输出数据)，需要完成以下2个步骤：

1、编写一个Java类，实现servlet接口。

2、把开发好的Java类部署到web服务器中。

按照一种约定俗成的称呼习惯，通常我们也把实现了servlet接口的java程序，称之为Servlet

二、Servlet的运行过程

Servlet程序是由WEB服务器调用，web服务器收到客户端的Servlet访问请求后：

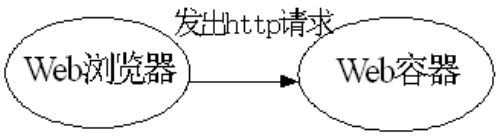
①Web服务器首先检查是否已经装载并创建了该Servlet的实例对象。如果是，则直接执行第④步，否则，执行第②步。

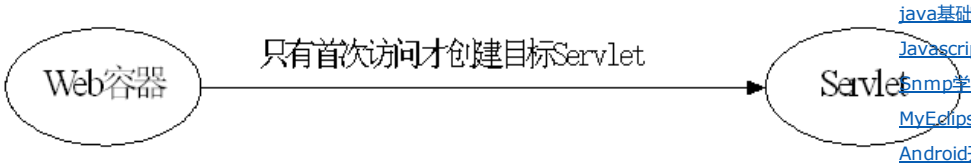
②装载并创建该Servlet的一个实例对象。

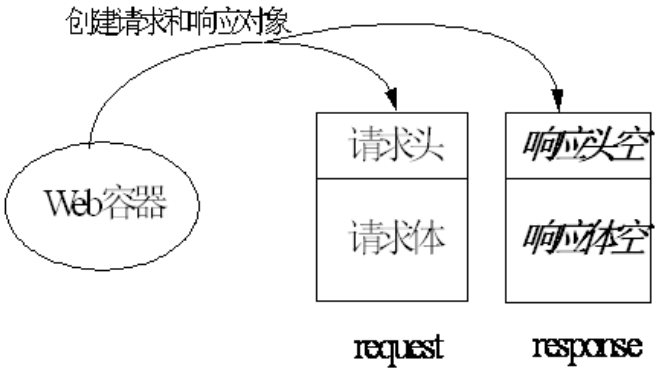
③调用Servlet实例对象的init()方法。

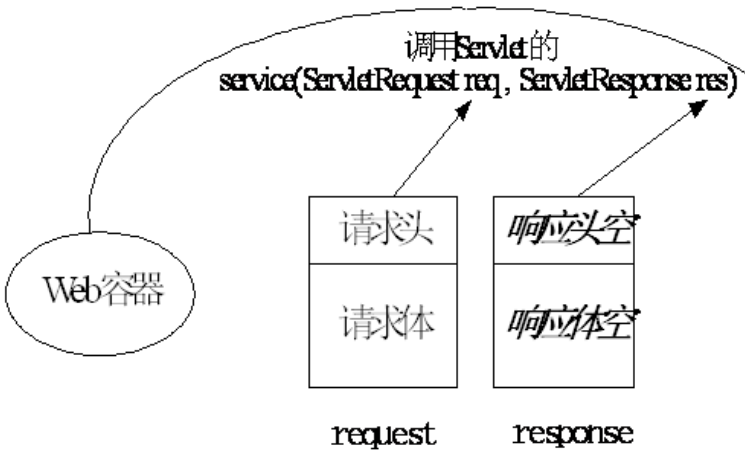
④创建一个用于封装HTTP请求消息的HttpServletRequest对象和一个代表HTTP响应消息的HttpServletResponse对象，然后调用Servlet的service()方法并将请求和响应对象作为参数传递进去。

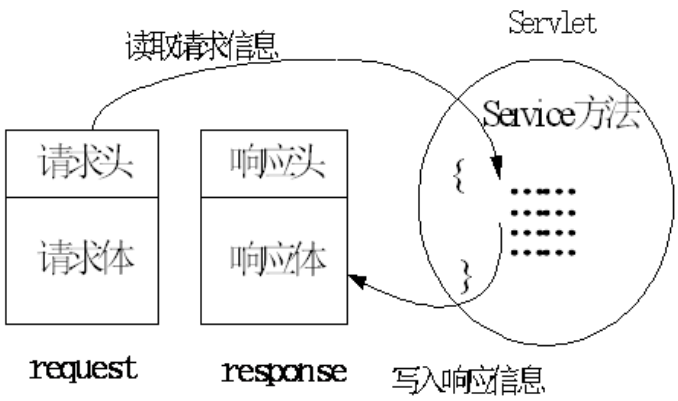
⑤WEB应用程序被停止或重新启动之前，Servlet引擎将卸载Servlet，并在卸载之前调用Servlet的destroy()方法。

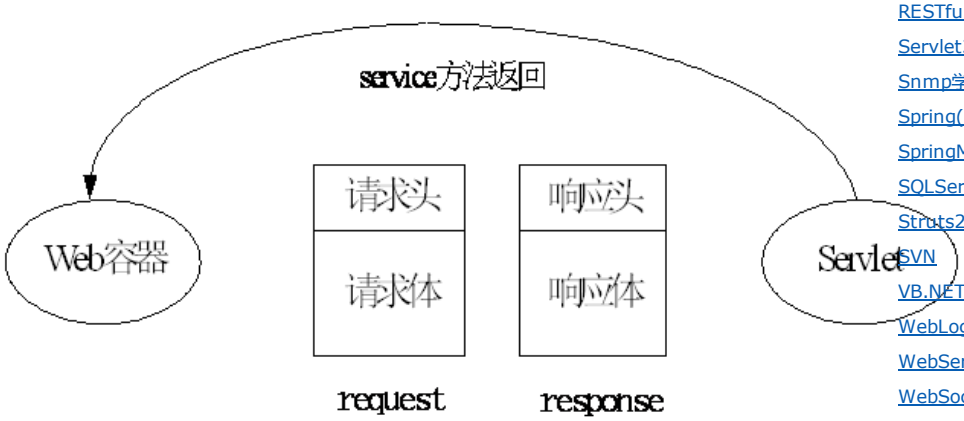


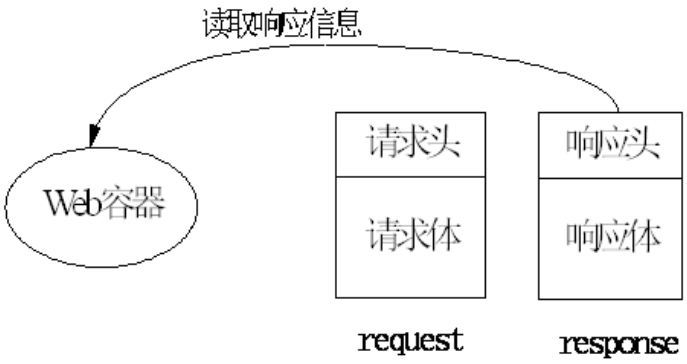


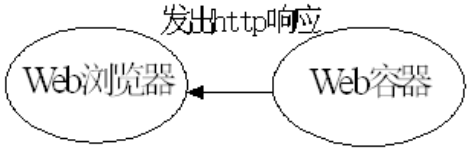




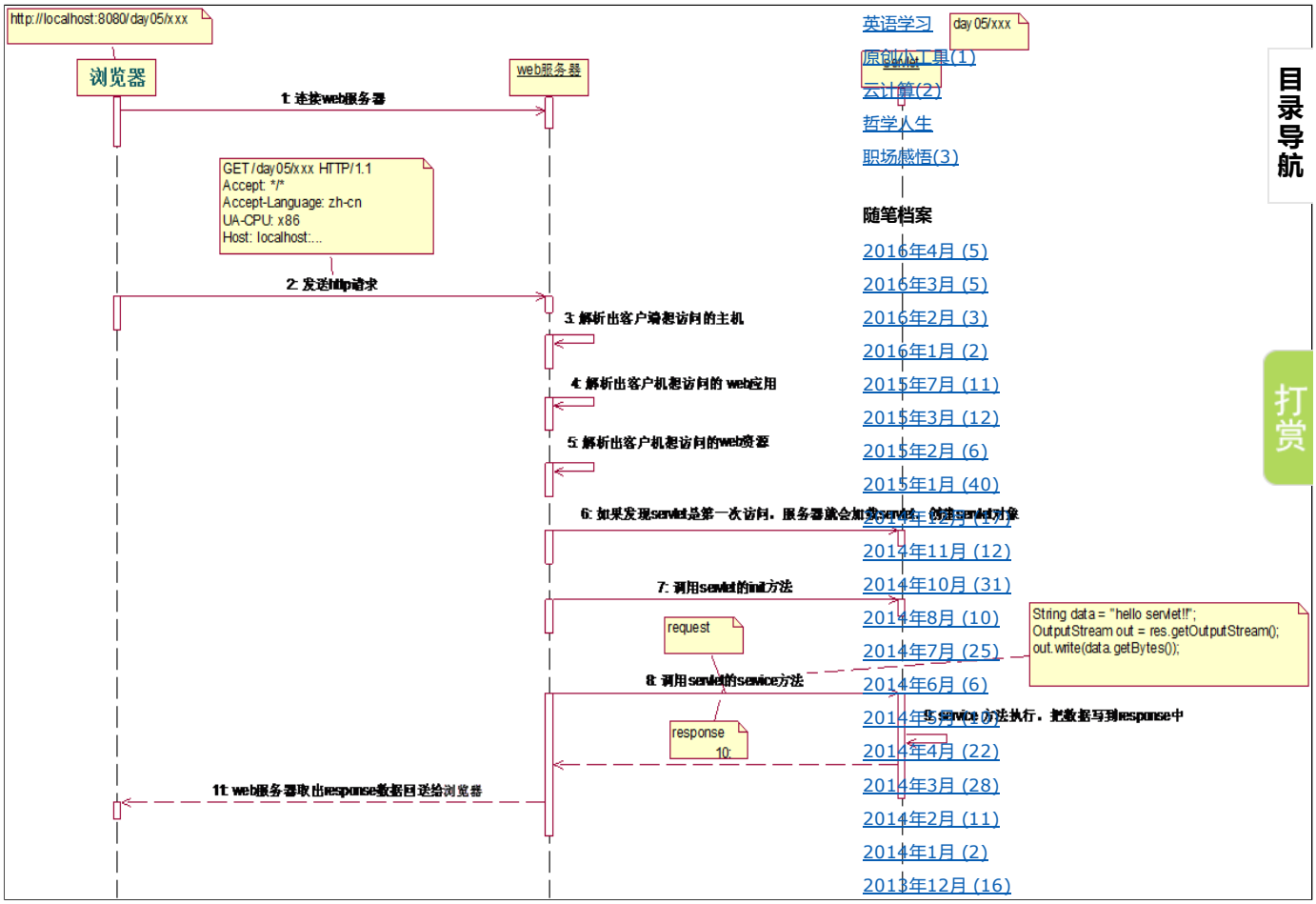






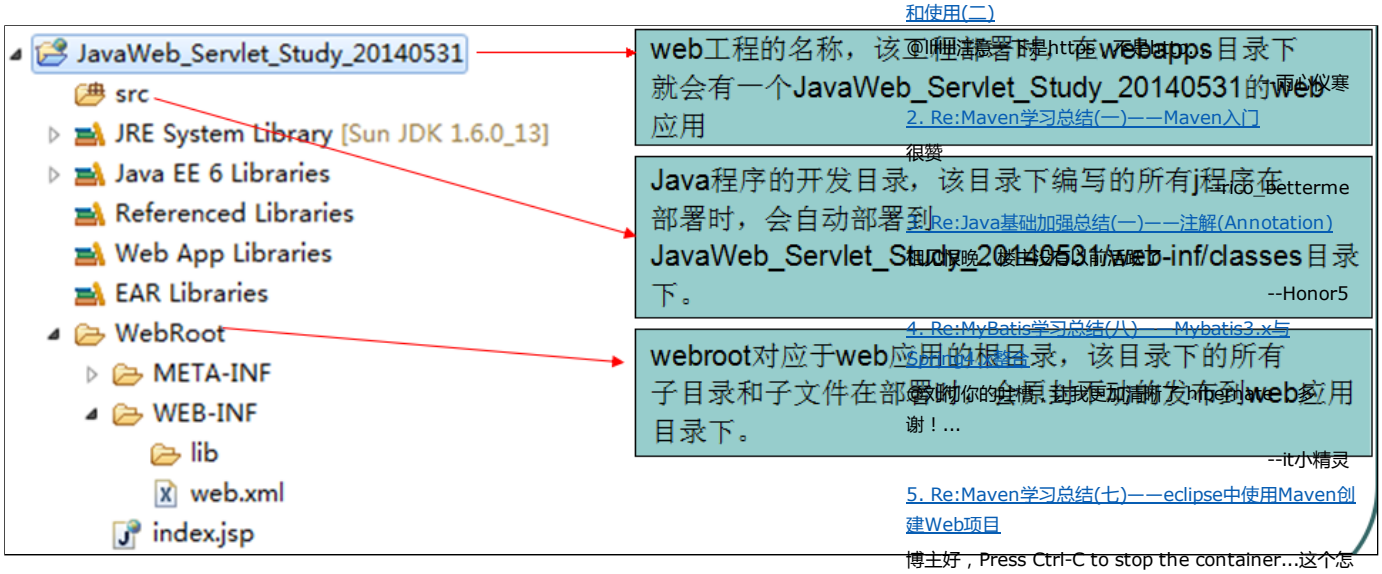


三、Servlet调用图



四、在Eclipse中开发Servlet

在eclipse中新建一个web project工程，eclipse会自动创建下图所示目录结构：



4.1、Servlet接口实现类

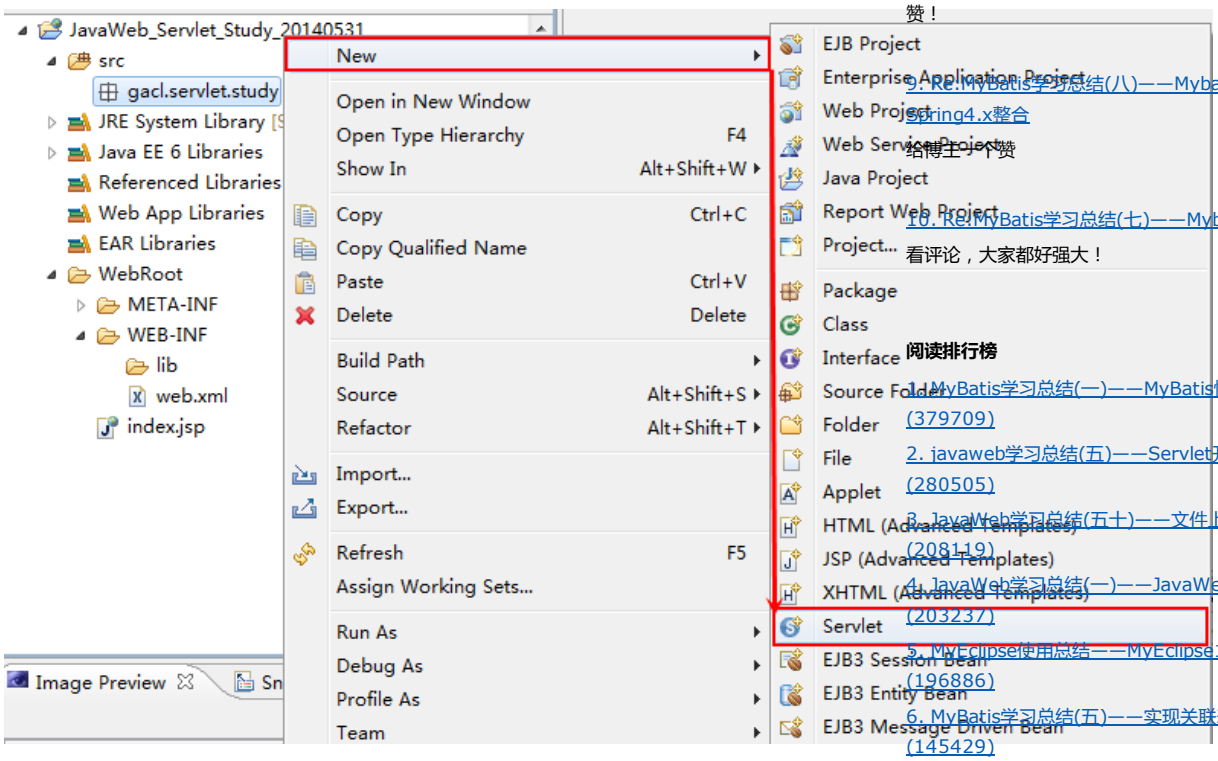
Servlet接口SUN公司定义了两个默认实现类，分别为：GenericServlet、HttpServlet。

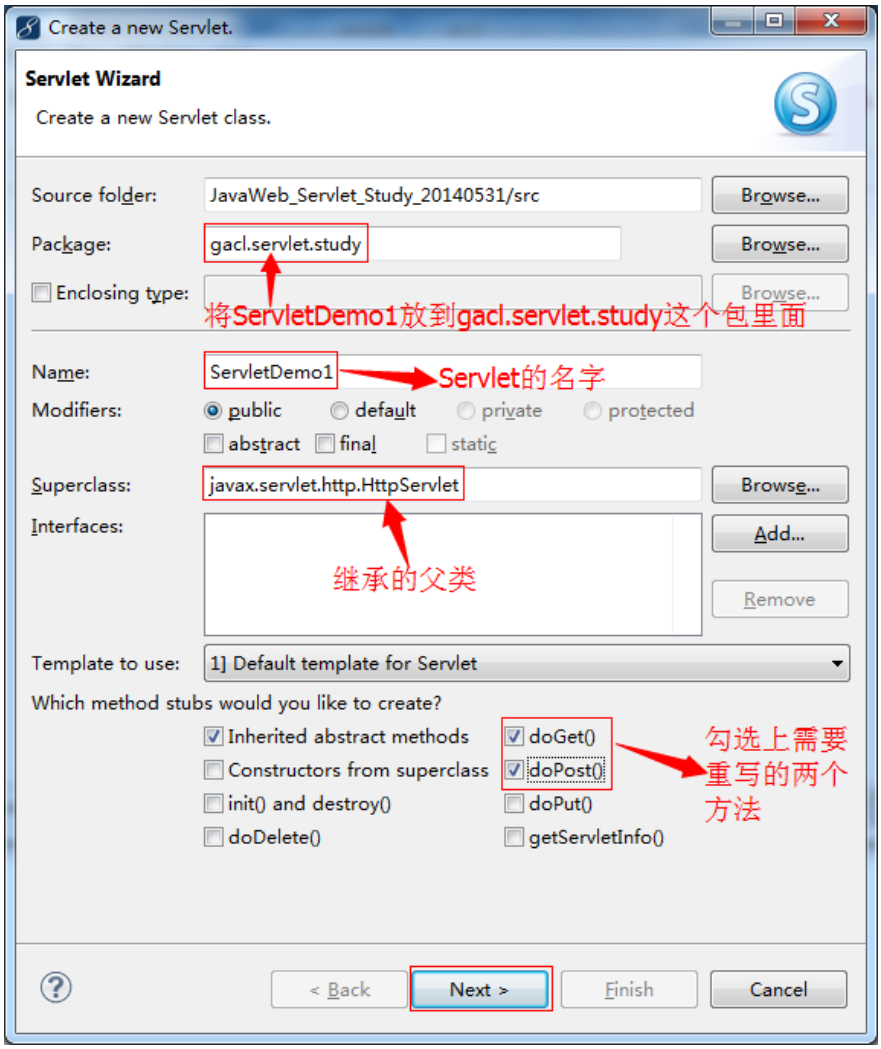
HttpServlet指能够处理HTTP请求的servlet，它在原有Servlet接口上添加了一些与HTTP协议处理方法，它比Servlet接口的功能更为强大。因此开发人员在编写Servlet时，通常应继承这个类，而避免直接去实现Servlet接口。

HttpServlet在实现Servlet接口时，覆写了service方法，该方法体内的代码会自动判断用户的请求方式，如为GET请求，则调用HttpServlet的doGet方法，如为Post请求，则调用doPost方法。因此，开发人员在编写Servlet时，通常只需要覆写doGet或doPost方法，而不要去覆写service方法。

4.2、通过Eclipse创建和编写Servlet

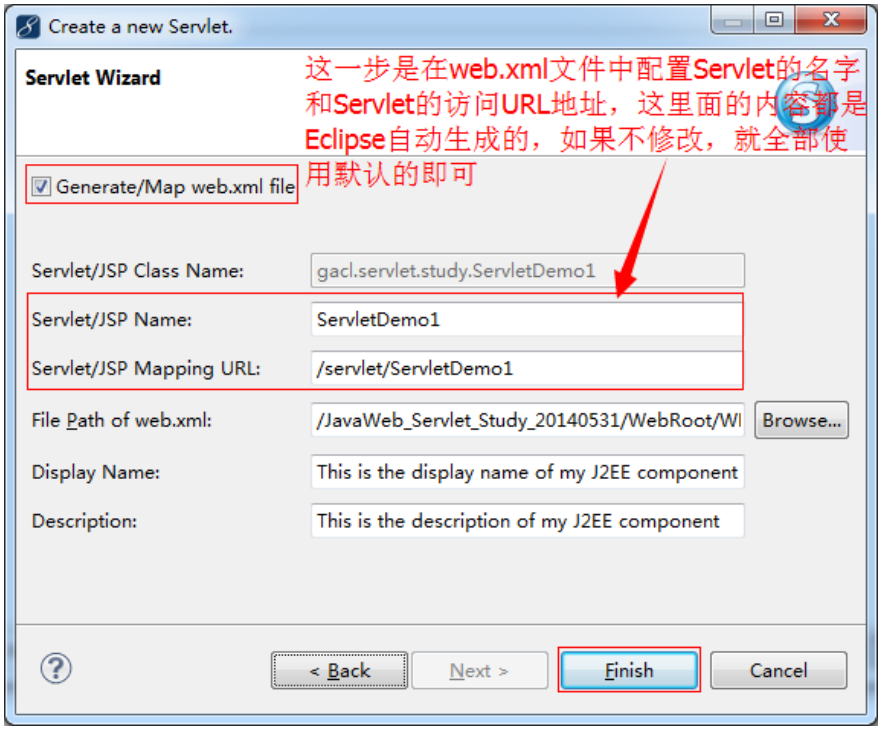
选中gacl.servlet.study包，右键→New→Servlet，如下图所示：





9. javaweb学习总结(四)——Http协议(67)
10. MyBatis学习总结(五)——实现关联表查询(65)

目录导航
打赏



这样，我们就通过Eclipse帮我们创建好一个名字为ServletDemo1的Servlet，创建好的ServletDemo01里面会有如下代码：

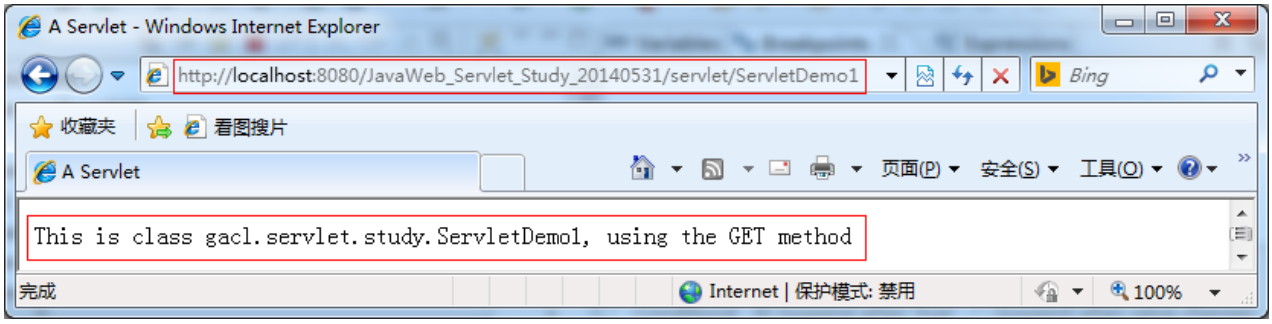
```
1 package gacl.servlet.study;
2
3 import java.io.IOException;
4 import java.io.PrintWriter;
5
6 import javax.servlet.ServletException;
7 import javax.servlet.http.HttpServlet;
8 import javax.servlet.http.HttpServletRequest;
9 import javax.servlet.http.HttpServletResponse;
10
11 public class ServletDemo1 extends HttpServlet {
12
13     /**
14      * The doGet method of the servlet. <br>
15      *
16      * This method is called when a form has its tag value method
equals to get.
17      *
18      * @param request the request send by the client to the server
19      * @param response the response send by the server to the client
20      * @throws ServletException if an error occurred
21      * @throws IOException if an error occurred
22      */
23     public void doGet(HttpServletRequest request,
HttpServletResponse response)
24             throws ServletException, IOException {
25
26         response.setContentType("text/html");
27         PrintWriter out = response.getWriter();
28         out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01
Transitional//EN\">");
29         out.println("<HTML>");
30         out.println("  <HEAD><TITLE>A Servlet</TITLE></HEAD>");
31         out.println("  <BODY>");
32         out.print("    This is ");
33         out.print(this.getClass());
34         out.println(", using the GET method");
35         out.println("  </BODY>");
```

```
36         out.println("</HTML>");
37         out.flush();
38         out.close();
39     }
40 
41     /**
42      * The doPost method of the servlet. <br>
43      *
44      * This method is called when a form has its tag value method
equals to post.
45      *
46      * @param request the request send by the client to the server
47      * @param response the response send by the server to the client
48      * @throws ServletException if an error occurred
49      * @throws IOException if an error occurred
50      */
51     public void doPost(HttpServletRequest request,
HttpServletResponse response)
52             throws ServletException, IOException {
53 
54         response.setContentType("text/html");
55         PrintWriter out = response.getWriter();
56         out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01
Transitional//EN\">");
57         out.println("<HTML>");
58         out.println("  <HEAD><TITLE>A Servlet</TITLE></HEAD>");
59         out.println("  <BODY>");
60         out.print("    This is ");
61         out.print(this.getClass());
62         out.println(", using the POST method");
63         out.println("  </BODY>");
64         out.println("</HTML>");
65         out.flush();
66         out.close();
67     }
68 
69 }
```

这些代码都是Eclipse自动生成的，而web.xml文件中也多了<servlet></servlet>和<servlet-mapping></servlet-mapping>两对标签，这两对标签是配置ServletDemo1的，如下图所示：

```
<servlet>
  <description>This is the description of my J2EE component</description>
  <display-name>This is the display name of my J2EE component</display-name>
  <servlet-name>ServletDemo1</servlet-name>
  <servlet-class>gacl.servlet.study.ServletDemo1</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>ServletDemo1</servlet-name>
  <url-pattern>/servlet/ServletDemo1</url-pattern>
</servlet-mapping>
```

然后我们就可以通过浏览器访问ServletDemo1这个Servlet，如下图所示：



五、Servlet开发注意细节

5.1、Servlet访问URL映射配置

由于客户端是通过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。一个<servlet-mapping>元素用于映射一个已注册的Servlet的一个对外访问路径，它包含有两个子元素：<servlet-name>和<url-pattern>，分别用于指定Servlet的注册名称和Servlet的对外访问路径。例如：

```
<servlet>
  <servlet-name>ServletDemo1</servlet-name>
  <servlet-class>gacl.servlet.study.ServletDemo1</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>ServletDemo1</servlet-name>
  <url-pattern>/servlet/ServletDemo1</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。 例如：

```
<servlet>
    <servlet-name>ServletDemo1</servlet-name>
    <servlet-class>gacl.servlet.study.ServletDemo1</servlet-class>
 </servlet>

 <servlet-mapping>
    <servlet-name>ServletDemo1</servlet-name>
    <url-pattern>/servlet/ServletDemo1</url-pattern>
 </servlet-mapping>
<servlet-mapping>
    <servlet-name>ServletDemo1</servlet-name>
    <url-pattern>/1.htm</url-pattern>
 </servlet-mapping>
  <servlet-mapping>
    <servlet-name>ServletDemo1</servlet-name>
    <url-pattern>/2.jsp</url-pattern>
 </servlet-mapping>
  <servlet-mapping>
    <servlet-name>ServletDemo1</servlet-name>
    <url-pattern>/3.php</url-pattern>
 </servlet-mapping>
  <servlet-mapping>
    <servlet-name>ServletDemo1</servlet-name>
    <url-pattern>/4.ASPX</url-pattern>
 </servlet-mapping>
```

通过上面的配置，当我们想访问名称是ServletDemo1的Servlet，可以使用如下的几个地址去访问：

http://localhost:8080/JavaWeb_Servlet_Study_20140531**/servlet/ServletDemo1**

http://localhost:8080/JavaWeb_Servlet_Study_20140531**/1.htm**

http://localhost:8080/JavaWeb_Servlet_Study_20140531**/2.jsp**

http://localhost:8080/JavaWeb_Servlet_Study_20140531**/3.php**

http://localhost:8080/JavaWeb_Servlet_Study_20140531**/4.ASPX**

ServletDemo1被映射到了多个URL上。

5.2、Servlet访问URL使用*通配符映射

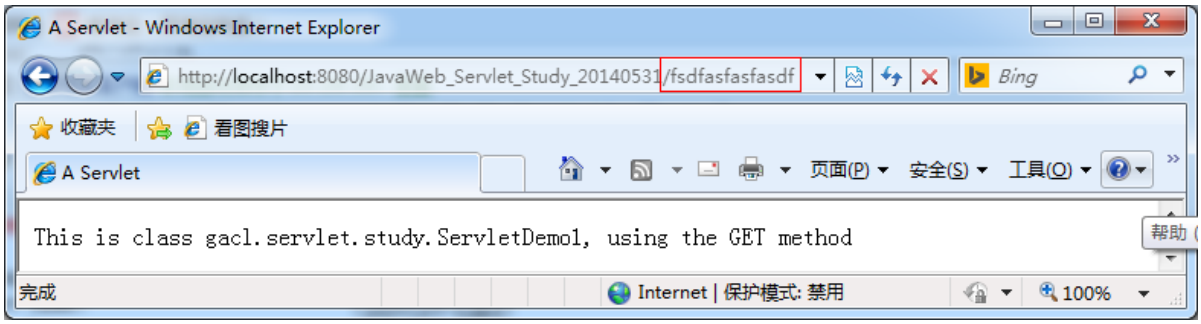
在Servlet映射到的URL中也可以使用*通配符，但是只能有两种固定的格式：一种格式是"*.扩展名"，另一种格式是以正斜杠（/）开头并以"/*"结尾。例如：

```
<servlet-mapping>
    <servlet-name>
        AnyName
    </servlet-name>
    <url-pattern>
        *.do
    </url-pattern>
</servlet-mapping>
```

```
<servlet-mapping>
    <servlet-name>
        AnyName
    </servlet-name>
    <url-pattern>
        /action/*
    </url-pattern>
</servlet-mapping>
```

```
<servlet>
    <servlet-name>ServletDemo1</servlet-name>
    <servlet-class>gacl.servlet.study.ServletDemo1</servlet-class>
  </servlet>

   <servlet-mapping>
    <servlet-name>ServletDemo1</servlet-name>
   <url-pattern>/*</url-pattern>
```

*可以匹配任意的字符，所以此时可以用任意的URL去访问ServletDemo1这个Servlet，如下图所示：



对于如下的一些映射关系：

Servlet1 映射到 /abc/*

Servlet2 映射到 /*

Servlet3 映射到 /abc

Servlet4 映射到 *.do

问题：

当请求URL为“/abc/a.html”，“/abc/*”和“/*”都匹配，哪个servlet响应

Servlet引擎将调用Servlet1。

当请求URL为“/abc”时，“/abc/*”和“/abc”都匹配，哪个servlet响应

Servlet引擎将调用Servlet3。

当请求URL为“/abc/a.do”时，“/abc/*”和“*.do”都匹配，哪个servlet响应

Servlet引擎将调用Servlet1。

当请求URL为“/a.do”时，“/*”和“*.do”都匹配，哪个servlet响应

Servlet引擎将调用Servlet2。

当请求URL为“/xxx/yyy/a.do”时，“/*”和“*.do”都匹配，哪个servlet响应
Servlet引擎将调用Servlet2。

匹配的原则就是"谁长得更像就找谁"

5.3、Servlet与普通Java类的区别

Servlet是一个供其他Java程序（Servlet引擎）调用的Java类，它不能独立运行，它的运行完全由Servlet引擎来控制和调度。

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Servlet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。

在Servlet的整个生命周期内，Servlet的init方法只被调用一次。而对一个Servlet的每次访问请求都导致Servlet引擎调用一次servlet的service方法。对于每次访问请求，Servlet引擎都会创建一个新的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，service方法再根据请求方式分别调用doXXX方法。

如果在<servlet>元素中配置了一个<load-on-startup>元素，那么WEB应用程序在启动时，就会装载并创建Servlet的实例对象、以及调用Servlet实例对象的init()方法。

举例：

```
<servlet>
   <servlet-name>invoker</servlet-name>
   <servlet-class>
      org.apache.catalina.servlets.InvokerServlet
   </servlet-class>
   <load-on-startup>1</load-on-startup>
</servlet>
```

用途：为web应用写一个InitServlet，这个servlet配置为启动时装载，为整个web应用创建必要的数据库表和数据。

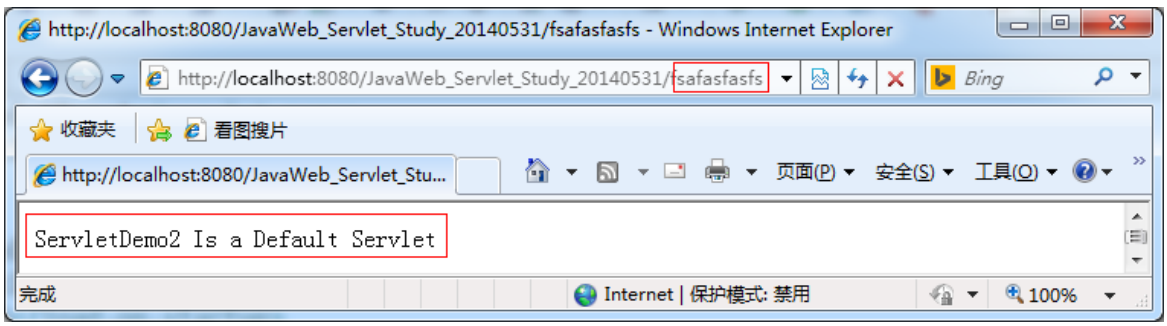
5.4、缺省Servlet

如果某个Servlet的映射路径仅仅为一个正斜杠（/），那么这个Servlet就成为当前Web应用程序的缺省Servlet。

凡是在web.xml文件中找不到匹配的<servlet-mapping>元素的URL，它们的访问请求都将交给缺省Servlet处理，也就是说，缺省Servlet用于处理所有其他Servlet都不处理的访问请求。 例如：

```
<servlet>
    <servlet-name>ServletDemo2</servlet-name>
    <servlet-class>gacl.servlet.study.ServletDemo2</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <!-- 将ServletDemo2配置成缺省Servlet -->
  <servlet-mapping>
    <servlet-name>ServletDemo2</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
```

当访问不存在的Servlet时，就使用配置的默认Servlet进行处理，如下图所示：



在<tomcat的安装目录>\conf\web.xml文件中，注册了一个名称为org.apache.catalina.servlets.DefaultServlet的Servlet，并将这个Servlet设置为了缺省Servlet。

```
    <servlet>
        <servlet-name>default</servlet-name>
        <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
        <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>

 <!-- The mapping for the default servlet -->
    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

当访问Tomcat服务器中的某个静态HTML文件和图片时，实际上是在访问这个缺省Servlet。

5.5、Servlet的线程安全问题

当多个客户端并发访问同一个Servlet时，web服务器会为每一个客户端的访问请求创建一个线程，并在这个线程上调用Servlet的service方法，因此service方法内如果访问了同一个资源的话，就有可能引发线程安全问题。例如下面的代码：

不存在线程安全问题的代码：

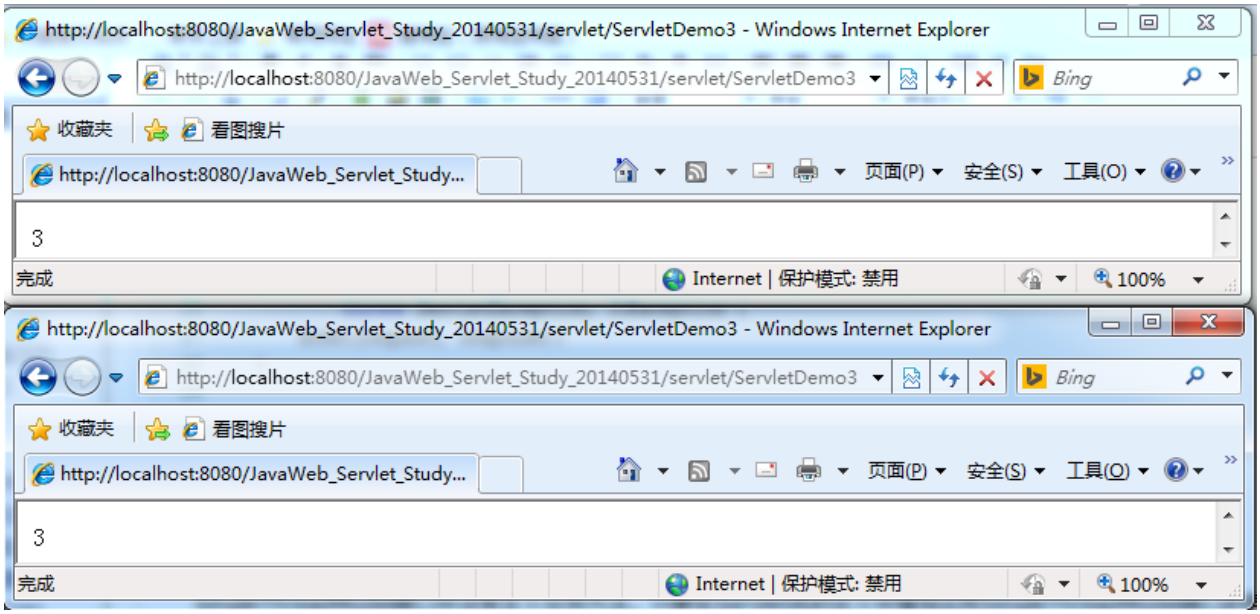
```
package gacl.servlet.study;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ServletDemo3 extends HttpServlet {


    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        /**
         * 当多线程并发访问这个方法里面的代码时，会存在线程安全问题吗
         * i变量被多个线程并发访问，但是没有线程安全问题，因为i是doGet方法里面的局部变量，
         * 当有多个线程并发访问doGet方法时，每一个线程里面都有自己的i变量，
         * 各个线程操作的都是自己的i变量，所以不存在线程安全问题
         * 多线程并发访问某一个方法的时候，如果在方法内部定义了一些资源(变量，集合等)
         * 那么每一个线程都有这些东西，所以就不存在线程安全问题了
         */
        int i=1;
        i++;
        response.getWriter().write(i);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
30             throws ServletException, IOException {
31         doGet(request, response);
32     }
33 
34 }
```

存在线程安全问题的代码：

```
 1 package gacl.servlet.study;
 2 
 3 import java.io.IOException;
 4 
 5 import javax.servlet.ServletException;
 6 import javax.servlet.http.HttpServlet;
 7 import javax.servlet.http.HttpServletRequest;
 8 import javax.servlet.http.HttpServletResponse;
 9 
10 public class ServletDemo3 extends HttpServlet {
11 
12     int i=1;
13     public void doGet(HttpServletRequest request,
HttpServletResponse response)
14             throws ServletException, IOException {
15 
16         i++;
17         try {
18             Thread.sleep(1000*4);
19         } catch (InterruptedException e) {
20             e.printStackTrace();
21         }
22         response.getWriter().write(i+"");
23     }
24 
25     public void doPost(HttpServletRequest request,
HttpServletResponse response)
26             throws ServletException, IOException {
27         doGet(request, response);
28     }
29 
30 }
```

把i定义成全局变量，当多个线程并发访问变量i时，就会存在线程安全问题了，如下图所示：同时开启两个浏览器模拟并发访问同一个Servlet，本来正常来说，第一个浏览器应该看到2，而第二个浏览器应该看到3的，结果两个浏览器都看到了3，这就不正常。



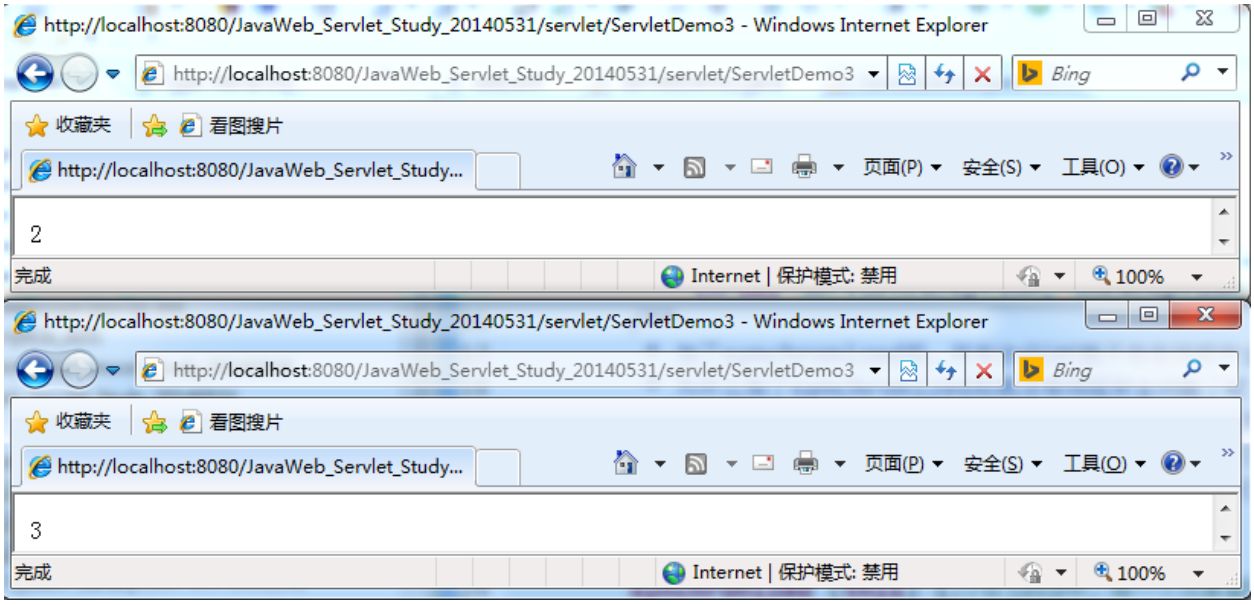
线程安全问题只存在多个线程并发操作同一个资源的情况下，所以在编写Servlet的时候，如果并发访问某一个资源(变量，集合等)，就会存在线程安全问题，那么该如何解决这个问题呢？

先看看下面的代码：

```
package gacl.servlet.study;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class ServletDemo3 extends HttpServlet {

    int i=1;
    public void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        /**
         * 加了synchronized后，并发访问i时就不存在线程安全问题了，
         * 为什么加了synchronized后就没有线程安全问题了呢？
         * 假如现在有一个线程访问Servlet对象，那么它就先拿到了Servlet对象的
那把锁
         * 等到它执行完之后才会把锁还给Servlet对象，由于是它先拿到了Servlet
对象的那把锁，
         * 所以当有别的线程来访问这个Servlet对象时，由于锁已经被之前的线程拿
走了，后面的线程只能排队等候了
         *
         */
        synchronized (this) {//在java中，每一个对象都有一把锁，这里的this
指的就是Servlet对象
            i++;
            try {
                Thread.sleep(1000*4);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            response.getWriter().write(i+"");
        }

    }

```

```
36     public void doPost(HttpServletRequest request,
HttpServletResponse response)
37             throws ServletException, IOException {
38         doGet(request, response);
39     }
40
41 }
```

现在这种做法是给Servlet对象加了一把锁，保证任何时候都只有一个线程在访问该Servlet对象里面的资源，这样就不存在线程安全问题了，如下图所示：



这种做法虽然解决了线程安全问题，但是编写Servlet却万万不能用这种方式处理线程安全问题，假如有9999个人同时访问这个Servlet，那么这9999个人必须按先后顺序排队轮流访问。

针对Servlet的线程安全问题，Sun公司是提供有解决方案的：**让Servlet去实现一个SingleThreadModel接口，如果某个Servlet实现了SingleThreadModel接口，那么Servlet引擎将以单线程模式来调用其service方法。**

查看Sevlet的API可以看到，SingleThreadModel接口中没有定义任何方法和常量，**在Java中，把没有定义任何方法和常量的接口称之为标记接口**，经常看到的一个最典型的标记接口就是"**Serializable**"，这个接口也是没有定义任何方法和常量的，标记接口在Java中有什么用呢？主要作用就是给某个对象打上一个标志，告诉JVM，这个对象可以做什么，比如实现了"**Serializable**"接口的类的对象就可以被序列化，还有一个"Cloneable"接口，这个也是一个标记接口，在默认情况下，Java中的对象是不允许被克隆的，就像现实生活中的人一样，不允许克隆，但是只要实现了"Cloneable"接口，那么对象就可以被克隆了。

让**Servlet实现了SingleThreadModel接口，**只要在Servlet类的定义中增加实现SingleThreadModel接口的声明即可。

对于实现了SingleThreadModel接口的Servlet，Servlet引擎仍然支持对该Servlet的多线程并发访问，其采用的方式是产生多个Servlet实例对象，并发的每个线程分别调用一个独立的Servlet实例对象。

实现SingleThreadModel接口并不能真正解决Servlet的线程安全问题，因为Servlet引擎会创建多个Servlet实例对象，而真正意义上解决多线程安全问题是指一个Servlet实例对象被多个线程同时调用的问题。事实上，在Servlet API 2.4中，已经将SingleThreadModel标记为Deprecated（过时的）。

分类: JavaWeb学习总结

标签: JavaWeb学习总结

孤傲苍狼
关注 - 88
粉丝 - 10004
+加关注

« 上一篇：javaweb学习总结(四)——Http协议
» 下一篇：javaweb学习总结(六)——Servlet开发(二)

posted on 2014-07-04 16:30 孤傲苍狼 阅读(280511) 评论(69) 编辑 收藏

< Prev 1 2

目录导航

打赏

评论

#51楼 2016-10-27 16:07 盛开后花园

给楼主赞一个！

支持(0) 反对(0)

#52楼 2016-11-07 15:56 大虫Ed

楼主的这篇文章讲的真是，不再好更多一点.

无论是初学入门还是会开发了再回头来看，收获满满.

赞!

支持(0) 反对(0)

#53楼 2016-11-09 16:23 龙豆豆

谢谢楼主

支持(0) 反对(0)

#54楼 2016-11-16 11:27 小不了

请问孤狼大神，文章可以转载吗》会注明来源的~

支持(0) 反对(0)

#55楼 2016-11-16 14:19 智在千里

没有讲怎么部署开发环境，一直报错

支持(0) 反对(0)

#56楼 2016-11-16 15:36 智在千里

看不下去了，环境不一样

支持(0) 反对(0)

#57楼 2016-11-24 21:41 adolfmc

感谢，

支持(0) 反对(0)

#58楼 2017-02-26 10:46 simpleDi

我只想说 比我们老师ppt讲的通俗易懂

支持(1) 反对(0)

#59楼 2017-03-03 11:22 十年如依

楼主真心大牛 感谢感谢

支持(0) 反对(0)

#60楼 2017-03-03 15:32 求学小明

楼主大牛，照着做了一遍收获很多，感谢

支持(0) 反对(0)

#61楼 2017-03-06 11:21 vashzx

您好，问个问题。实际操作时发现。您的项目目录结构与我用eclipse创建的不同，请问您是建立的dynamic web project吗？请问您eclipse版本是什么？

支持(0) 反对(0)

#62楼 2017-03-06 17:12 ChengHui5690

楼主，怎么刚开始创建servlet文件时用eclipse然后一下子又换成了MyEclipse？

支持(0) 反对(0)

#63楼 2017-03-07 11:26 vashzx

@ ChengHui5690
原来如此！我原以为认为楼主一直用的eclipse，原来后面用的myeclipse，怪不得目录结构和步骤都不一样~

支持(0) 反对(0)

#64楼 2017-03-29 09:37 爱喝牛奶的猫

刚学Servlet,老是卡在URL上，看了楼主写的豁然开朗，很开心，谢谢楼主！

支持(0) 反对(0)

#65楼 2017-03-29 14:54 alittlecomputer

您好，跟您学JavaWeb，谢谢您这么细致的博客。
有一点不足的是后面几个的代码应该为
response.getWriter().print(i);
而不是response.getWriter().write(i+"");

支持(0) 反对(0)

#66楼 2017-06-05 16:15 babyla

楼主好强大，受益匪浅~

支持(0) 反对(0)

#67楼 2017-06-08 09:30 学思危

臧害学习Java这个还真要看你的努力程度和你所选择的学校了，就像我朋友在华清远见学习了四个月在这期间自己也够努力学校也教的好最后还是找到工作了，加油小伙好好学，不会失败的。

支持(0) 反对(0)

#68楼 2017-06-08 21:26 Mr.袋鼠

楼主你好，我在照着做例子时 (int i=1;
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {

i++;
try {
Thread.sleep(1000*4);
} catch (InterruptedException e) {
e.printStackTrace();
}
response.getWriter().write(i+"");
})

发现您贴出的两个结果图都是 3 ，我实际操作了一下，发现结果不一样，我认为，i是两个线程公用的，那么就不可能出现都是数字3的情况。

期待您的回复！

支持(0) 反对(0)

#69楼 2017-06-16 10:17 学思危

有很多Javaweb的东西都是由Java开饭的人员来做的，只要学好JavaJavaweb一定也有进步就像我朋友在http://javaee.3g-edu.org/?lbj学习完后进了一家公司做Java开发，但是在工作中也会做些Javaweb 的东西，他还说辛亏老师有交过，

支持(0) 反对(0)

< Prev 1 2