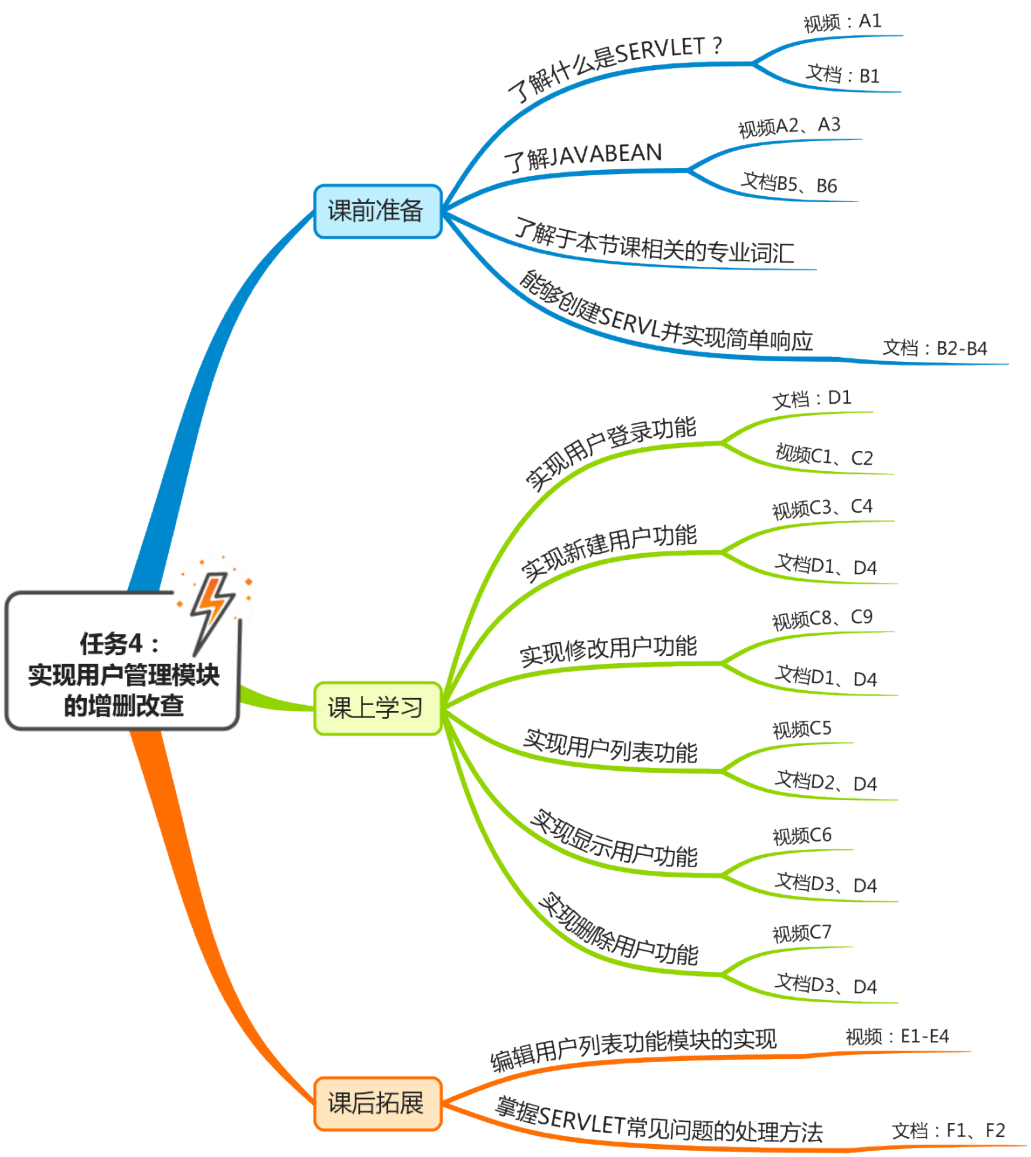
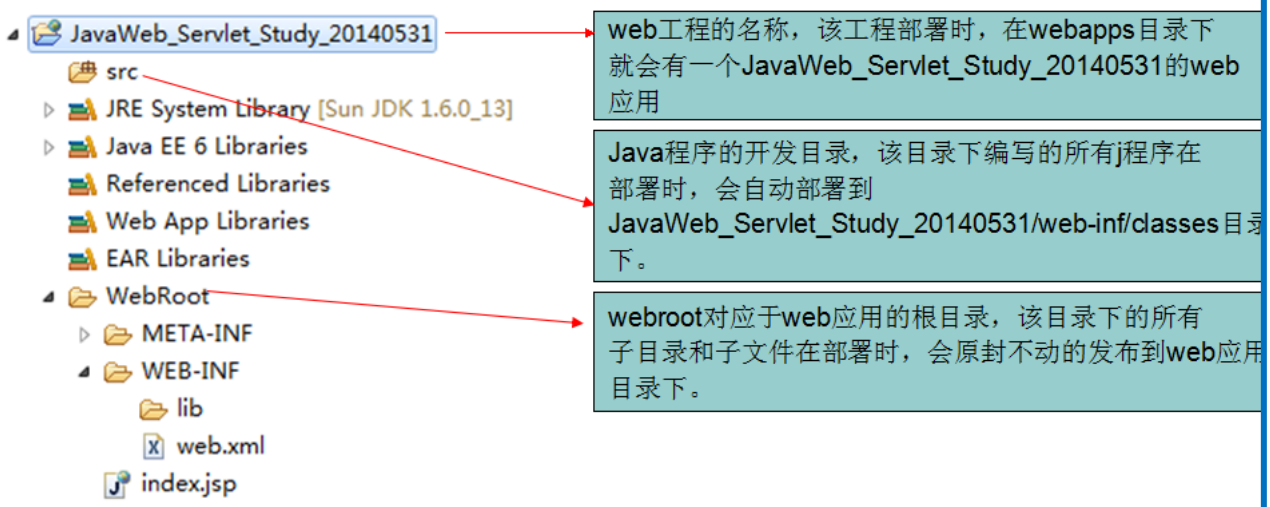
单元 1.1.4 用 Servlet 技术实现用户管理模块的增删改查

<table>
<tr><th colspan="3">【课程导入】</th></tr>
<tr><td colspan="3">在本节课的学习中，我们将学习 Java Servlet 功能，历史版本及配置方法，学习 Servlet 工作原理、Servlet 生命周期、Servlet 核心类和 Servlet 接口的常用方法，学会用 Servlet 会话跟踪、Servlet 会话管理、重定向技术以及 Servlet 和 JSP 共享对象方法。
Java Servlet 是 Java Web 技术的基础。本单元的学习，同学们可以更加深刻的理解和应用 Java Web 开发技术。</td></tr>
<tr><th>知识目标</th><th>能力目标</th><th>素质目标</th></tr>
<tr><td>1.掌握 JSP 中使用 JavaBean 的语法
2.掌握 JavaBean 的作用域
3.掌握 JavaBean 封装方法
4. 掌握 Java Servlet 及使用方法</td><td>1.能够灵活使用 JavaBean 技术封装数据
2.能够用 Servlet 响应来自客户的请求
3.能够深刻理解 JSP+JavaBean+Servlet 开发模式</td><td>1.培养学生的团队意识和团队协作精神，锻炼学生的沟通交流能力;
2.通过项目教学，让学生真切的体验项目分析、设计、管理及实施的全过程;</td></tr>
<tr><th>学习任务</th><th>重点难点</th><th>突破方法</th></tr>
<tr><td>熟练的使用 JSP+JavaBean+Servlet 开发模式完成一个模块的增删改查</td><td>1. 封装用户信息、实现数据库连接的 JavaBean
2. JavaServlet 的编写与部署过程、Servlet 的生命周期、Servlet 接口</td><td>采用翻转课堂、项目导入的教学模式，进行分组讨论、演示动画原理。</td></tr>
</table>

学习导航	
任务1	**任务描述：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：



1、Servlet 接口实现类

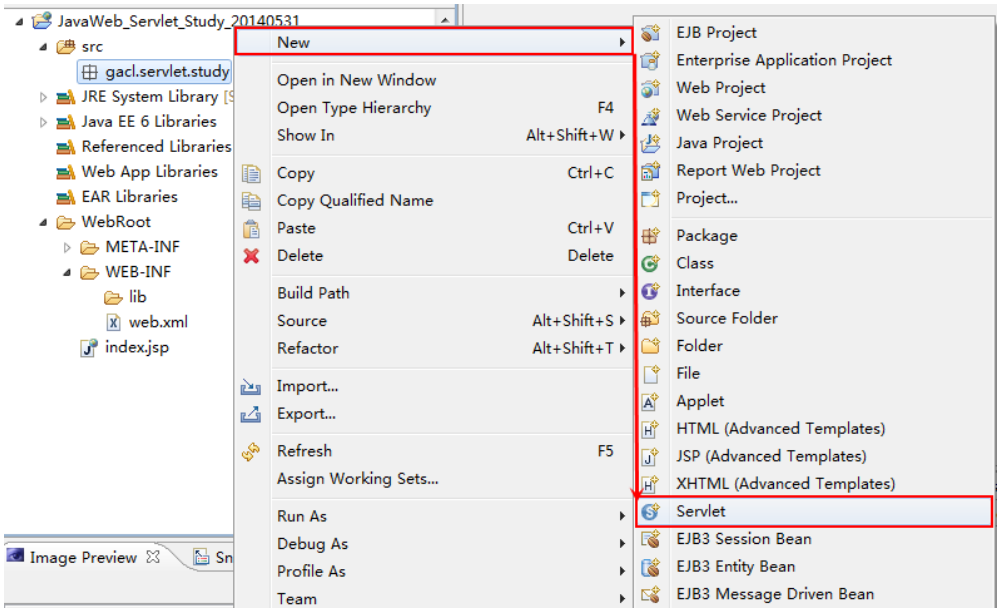
Servlet 接口 SUN 公司定义了两个默认实现类，分别为：GenericServlet、HttpServlet。

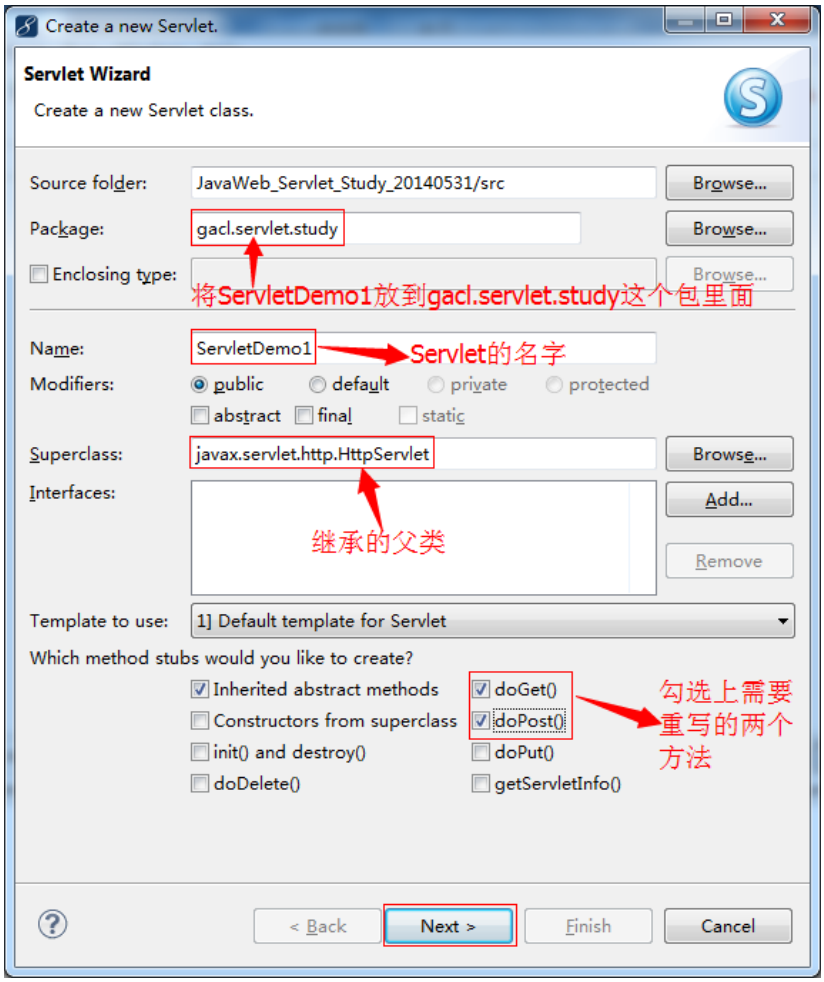
HttpServlet 指能够处理 HTTP 请求的 servlet，它在原有 Servlet 接口上添加了一些与 HTTP 协议处理方法，它比 Servlet 接口的功能更为强大。因此开发人员在编写 Servlet 时，通常应继承这个类，而避免直接去实现 Servlet 接口。

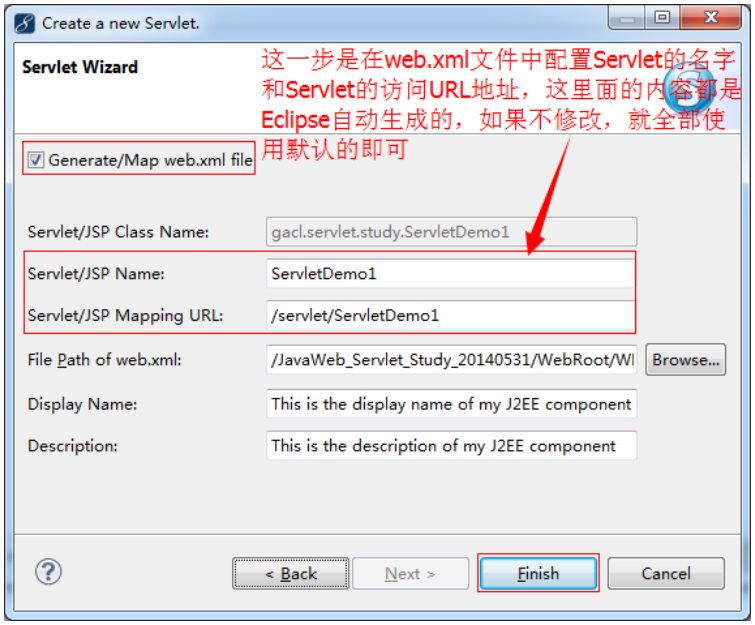
HttpServlet 在实现 Servlet 接口时，覆写了 service 方法，该方法体内的代码会自动判断用户的请求方式，如为 GET 请求，则调用 HttpServlet 的 doGet 方法，如为 Post 请求，则调用 doPost 方法。因此，开发人员在编写 Servlet 时，通常只需要覆写 doGet 或 doPost 方法，而不要去覆写 service 方法。

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：







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      *
```

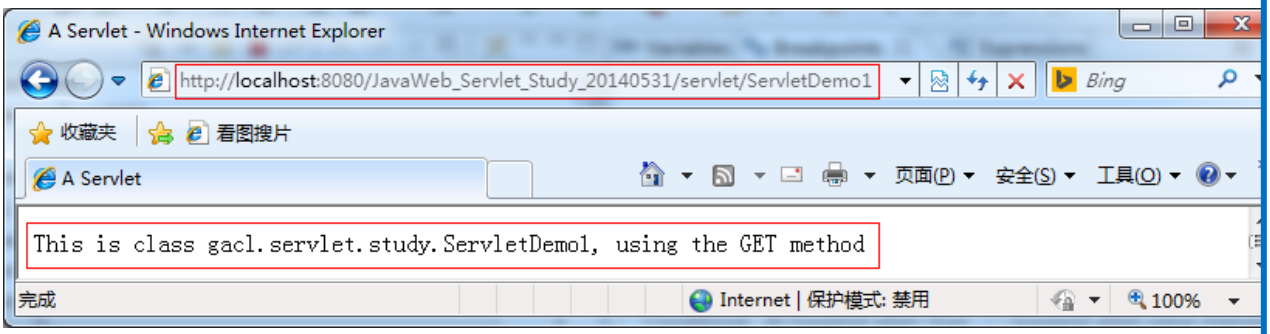
```
46      * @param request the request send by the client to the server
47      * @param response the response send by the server to the client
48      * @throws ServletException if an error occurred
49      * @throws IOException if an error occurred
50      */
51     public void doPost(HttpServletRequest 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：



任务2

任务描述：处理 Servlet 开发时容易出错的几个地方

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：

```
1  <servlet>
2    <servlet-name>ServletDemo1</servlet-name>
3    <servlet-class>gacl.servlet.study.ServletDemo1</servlet-class>
4  </servlet>
5
6  <servlet-mapping>
7    <servlet-name>ServletDemo1</servlet-name>
8    <url-pattern>/servlet/ServletDemo1</url-pattern>
9  </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：

```
1  <servlet>
2    <servlet-name>ServletDemo1</servlet-name>
3    <servlet-class>gacl.servlet.study.ServletDemo1</servlet-class>
4  </servlet>
5
6  <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。

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>
```

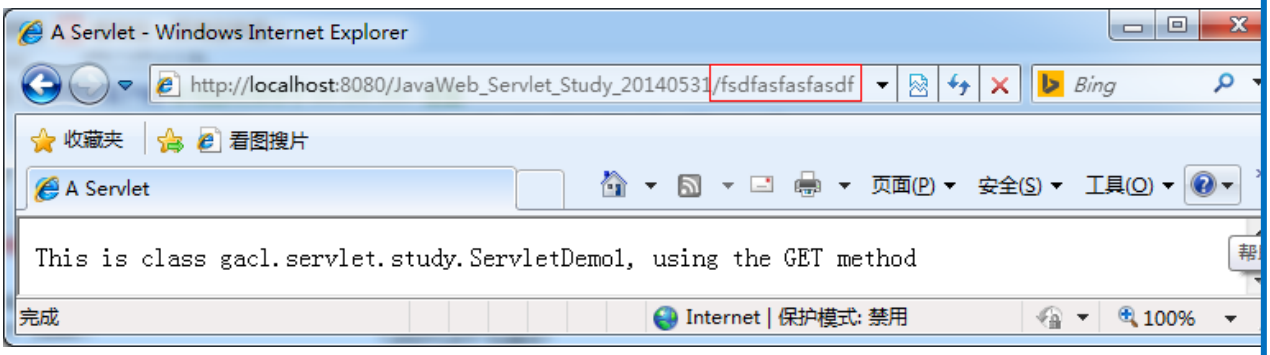
```
<servlet-mapping>
    <servlet-name>
        AnyName
    </servlet-name>
    <url-pattern>
        /action/*
    </url-pattern>
</servlet-mapping>
```

```
<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"

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。

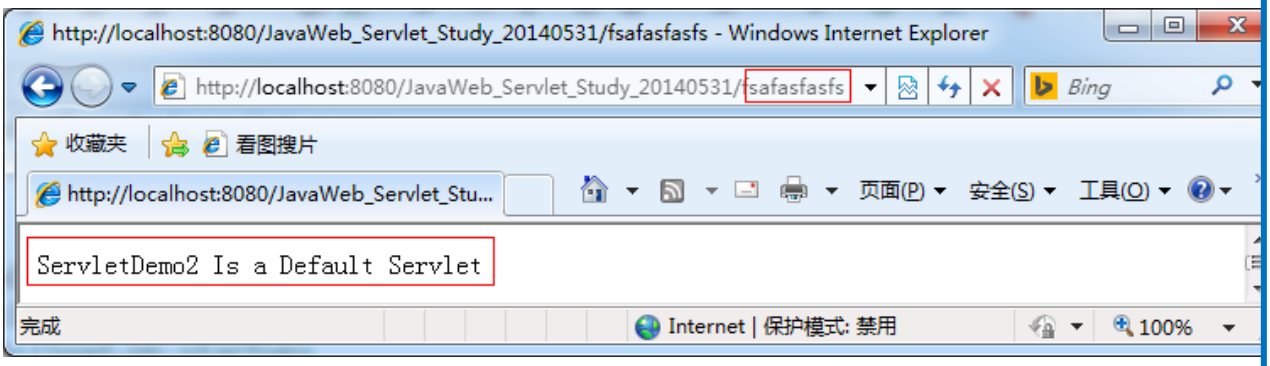
4、缺省 Servlet

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：

```
1  <servlet>
2     <servlet-name>ServletDemo2</servlet-name>
3     <servlet-class>gacl.servlet.study.ServletDemo2</servlet-class>
4     <load-on-startup>1</load-on-startup>
5   </servlet>
6
7   <!-- 将 ServletDemo2 配置成缺省 Servlet -->
8   <servlet-mapping>
9     <servlet-name>ServletDemo2</servlet-name>
10    <url-pattern>/</url-pattern>
11  </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、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;

```

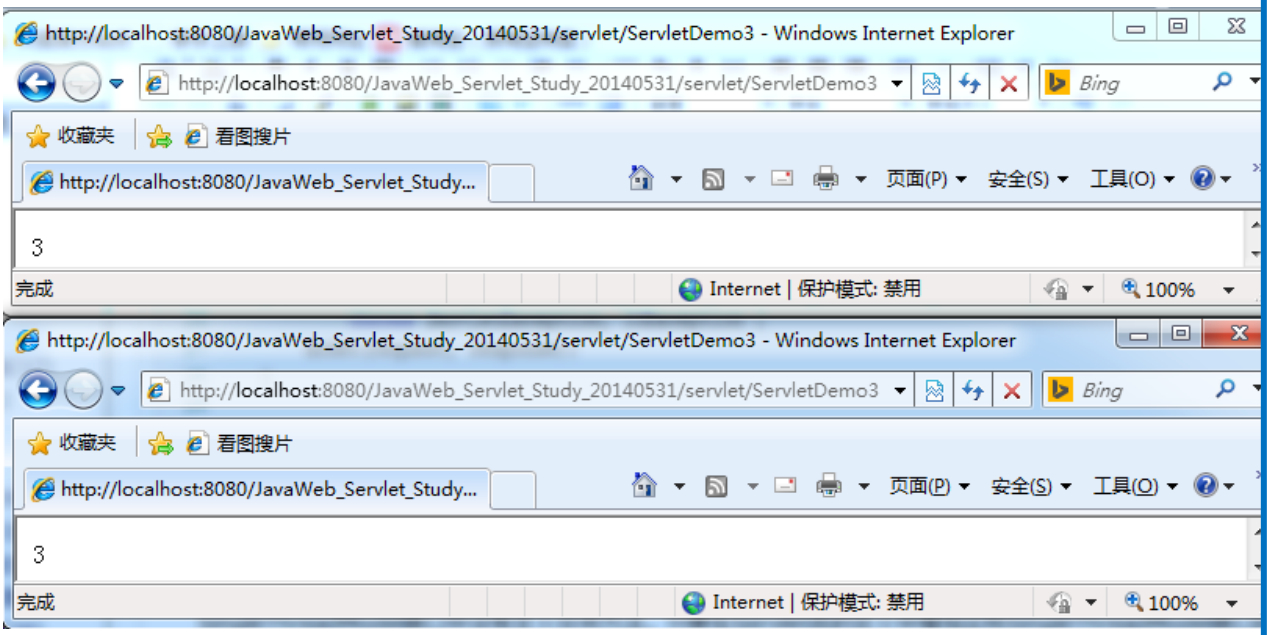
```
 3 import java.io.IOException;
 4
 5 import javax.servlet.ServletException;
 6 import javax.servlet.http.HttpServlet;
 7 import javax.servlet.http.HttpServletRequest;
 8 import javax.servlet.http.HttpServletResponse;
 9
10 public class ServletDemo3 extends HttpServlet {
11
12
13     public void doGet(HttpServletRequest request, HttpServletResponse
response)
14             throws ServletException, IOException {
15
16         /**
17          * 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
18          * 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，
19          * 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，
20          * 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
21          * 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)
22          * 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
23          */
24         int i=1;
25         i++;
26         response.getWriter().write(i);
27     }
28
29     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。



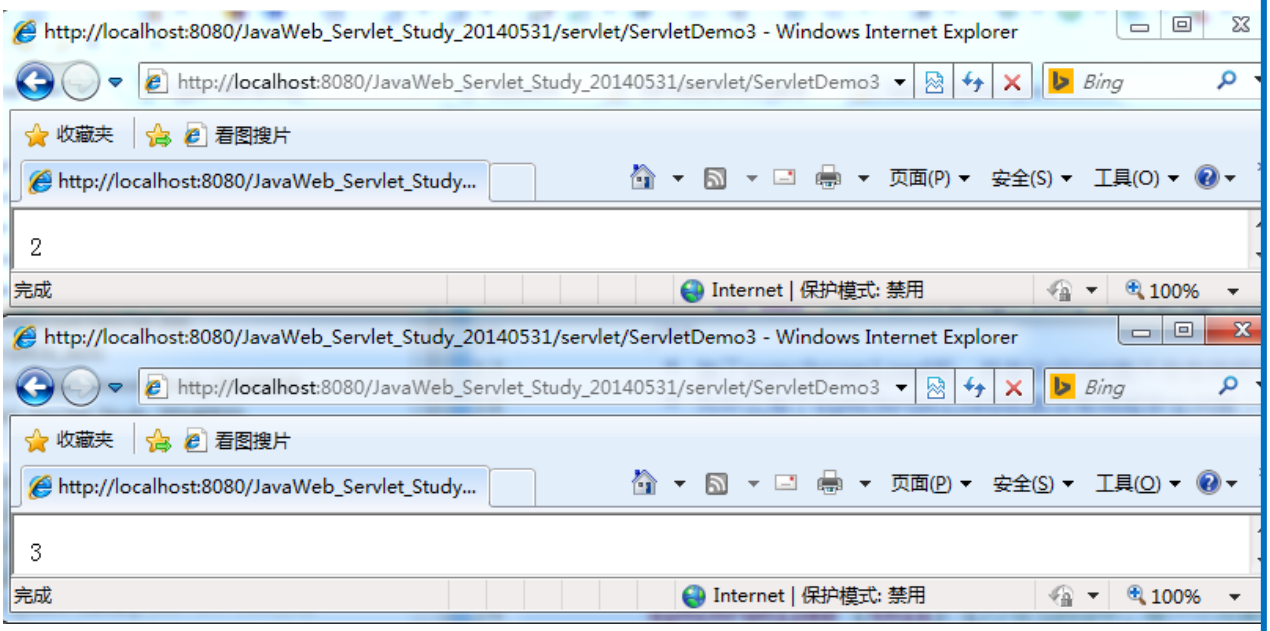
线程安全问题只存在多个线程并发操作同一个资源的情况下，所以在编写 Servlet 的时候，如果并发访问某一个资源(变量，集合等)，就会存在线程安全问题，那么该如何解决这个问题呢？

先看看下面的代码：

```
 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
11 public class ServletDemo3 extends HttpServlet {
12
13     int i=1;
14     public void doGet(HttpServletRequest request, HttpServletResponse
response)
15             throws ServletException, IOException {
16         /**
17          * 加了 synchronized 后，并发访问 i 时就不存在线程安全问题了，
18          * 为什么加了 synchronized 后就没有线程安全问题了呢？
19          * 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
20          * 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，
21          * 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
```

```
22          *
23          */
24         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
25             i++;
26             try {
27                 Thread.sleep(1000*4);
28             } catch (InterruptedException e) {
29                 e.printStackTrace();
30             }
31             response.getWriter().write(i+"");
32         }
33
34     }
35
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）