

Cookie&SessionJsp-授课

1 会话技术

1.1 会话管理概述

1.1.1 什么是会话

这里的会话，指的是web开发中的一次通话过程，当打开浏览器，访问网站地址后，会话开始，当关闭浏览器（或者到了过期时间），会话结束。

举个例子：

例如，你在给家人打电话，这时突然有送快递的配送员敲门，你放下电话去开门，收完快递回来后，通话还在保持中，继续说话就行了。

1.1.2 会话管理作用

什么时候会用到会话管理呢？最常见的就是购物车，当我们登录成功后，把商品加入到购物车之中，此时我们无论再浏览什么商品，当点击购物车时，那些加入的商品都仍在购物车中。

在我们的实际开发中，还有很多地方都离不开会话管理技术。比如，我们在论坛发帖，没有登录的游客身份是不允许发帖的。所以当我们登录成功后，无论我们进入哪个版块发帖，只要权限允许的情况下，服务器都会认识我们，从而让我们发帖，因为登录成功的信息一直保留在服务器端的会话中。

通过上面的两个例子，我们可以看出，它是为我们共享数据用的，并且是在不同请求间实现数据共享。也就是说，如果我们需要在多次请求间实现数据共享，就可以考虑使用会话管理技术了。

1.1.3 会话管理分类

在JavaEE的项目中，会话管理分为两类。分别是：客户端会话管理技术和服务端会话管理技术。

客户端会话管理技术

它是把要共享的数据保存到了客户端（也就是浏览器端）。每次请求时，把会话信息带到服务器，从而实现多次请求的数据共享。

服务端会话管理技术

它本质仍是采用客户端会话管理技术，只不过保存到客户端的是一个特殊的标识，并且把要共享的数据保存到了服务端的内存对象中。每次请求时，把这个标识带到服务器端，然后使用这个标识，找到对应的内存空间，从而实现数据共享。

1.2 客户端会话管理技术

1.2.1 Cookie概述

1) 什么是Cookie

它是客户端浏览器的缓存文件，里面记录了客户浏览器访问网站的一些内容。同时，也是HTTP协议请求和响应消息头的一部分（在HTTP协议课程中，我们备注了它很重要）。

2) Cookie的API详解

作用

它可以保存客户浏览器访问网站的相关内容（需要客户端不禁用Cookie）。从而在每次访问需要同一个内容时，先从本地缓存获取，使资源共享，提高效率。

Cookie的属性

属性名称	属性作用	是否重要
name	cookie的名称	必要属性
value	cookie的值（不能是中文）	必要属性
path	cookie的路径	重要
domain	cookie的域名	重要
maxAge	cookie的生存时间。	重要
version	cookie的版本号。	不重要
comment	cookie的说明。	不重要

细节

Cookie有大小，个数限制。每个网站最多只能存20个cookie，且大小不能超过4kb。同时，所有网站的cookie总数不超过300个。

当删除Cookie时，设置maxAge值为0。当不设置maxAge时，使用的是浏览器的内存，当关闭浏览器之后，cookie将丢失。设置了此值，就会保存成缓存文件（值必须是大于0的,以秒为单位）。

3) Cookie涉及的常用方法

创建Cookie



```
/**
 * 通过指定的名称和价值构造一个Cookie
 *
 * Cookie的名称必须遵循RFC 2109规范。这意味着，它只能包含ASCII字母数字字符，
 * 不能包含逗号、分号或空格或以$字符开头。
 * 创建后无法更改cookie的名称。
 *
 * 该值可以是服务器选择发送的任何内容。
 * 它的价值可能只有服务器才感兴趣。
 * 创建之后，可以使用setValue方法更改cookie的值。
 */
public Cookie(String name, String value) {
    validation.validate(name);
    this.name = name;
    this.value = value;
}
```

向浏览器添加Cookie

```
HttpServletResponse.java
36 public interface HttpServletResponse extends HttpServletResponse {
37
38     /**
39      * Adds the specified cookie to the response. This method can be called
40      * multiple times to set more than one cookie.
41      *
42      * @param cookie
43      *         the Cookie to return to the client
44      */
45     public void addCookie(Cookie cookie);
46
```

```
/**
 * 添加Cookie到响应中。此方法可以多次调用，用以添加多个Cookie。
 */
public void addCookie(Cookie cookie);
```

从服务器端获取Cookie

```
HttpServletRequest.java
73 /**
74  * Returns an array containing all of the Cookie objects the
75  * client sent with this request. This method returns null if
76  * no cookies were sent.
77  *
78  * @return an array of all the Cookies included with this
79  *         request, or null if the request has no cookies
80  */
81 public Cookie[] getCookies();
82
83 /**
```

```
/**
 * 这是HttpServletRequest中的方法。
 * 它返回一个Cookie的数组，包含客户端随此请求发送的所有Cookie对象。
 * 如果没有符合规则的cookie，则此方法返回null。
 */
public Cookie[] getCookies();
```

1.2.2 Cookie的Path细节：浏览器什么时候带给服务器，什么时候不带

1) 需求说明

创建一个Cookie，设置Cookie的path，通过不同的路径访问，从而查看请求携带Cookie的情况。

2) 案例目的

通过此案例的讲解，同学们可以清晰的描述出，客户浏览器何时带cookie到服务器端，何时不带。

3) 案例步骤

第一步：创建JavaWeb工程

沿用第一个案例中的工程即可。

第二步：编写Servlet

```
/**
 * Cookie的路径问题
 * 前期准备：
```

```

* 1.在demo1中写一个cookie到客户端
* 2.在demo2和demo3中分别去获取cookie
*     demo1的Servlet映射是    /servlet/PathQuestionDemo1
*     demo2的Servlet映射是    /servlet/PathQuestionDemo2
*     demo3的Servlet映射是    /PathQuestionDemo3
*
* @author 黑马程序员
* @Company http://www.itheima.com
*
*/
public class PathQuestionDemo1 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        //1.创建一个Cookie
        Cookie cookie = new Cookie("pathquestion","CookiePathQuestion");
        //2.设置cookie的最大存活时间
        cookie.setMaxAge(Integer.MAX_VALUE);
        //3.把cookie发送到客户端
        response.addCookie(cookie);//setHeader("Set-Cookie","cookie的值")
    }

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

```

```

/**
* 获取Cookie，名称是pathquestion
* @author 黑马程序员
* @Company http://www.itheima.com
*
*/
public class PathQuestionDemo2 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        //1.获取所有的cookie
        Cookie[] cs = request.getCookies();
        //2.遍历cookie的数组
        for(int i=0;cs!=null && i<cs.length;i++){
            if("pathquestion".equals(cs[i].getName())){
                //找到了我们想要的cookie，输出cookie的值
                response.getWriter().write(cs[i].getValue());
                return;
            }
        }
    }

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

```

```

/**
 * 获取Cookie, 名称是pathquestion
 * @author 黑马程序员
 * @Company http://www.itheima.com
 */
public class PathQuestionDemo3 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        //1. 获取所有的cookie
        Cookie[] cs = request.getCookies();
        //2. 遍历cookie的数组
        for(int i=0;cs!=null && i<cs.length;i++){
            if("pathquestion".equals(cs[i].getName())){
                //找到了我们想要的cookie, 输出cookie的值
                response.getWriter().write(cs[i].getValue());
                return;
            }
        }
    }

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

```

第三步：配置Servlet

```

<!--配置Cookie路径问题案例的Servlet-->
<servlet>
    <servlet-name>PathQuestionDemo1</servlet-name>
    <servlet-
class>com.itheima.web.servlet.pathquestion.PathQuestionDemo1</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>PathQuestionDemo1</servlet-name>
    <url-pattern>/servlet/PathQuestionDemo1</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>PathQuestionDemo2</servlet-name>
    <servlet-
class>com.itheima.web.servlet.pathquestion.PathQuestionDemo2</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>PathQuestionDemo2</servlet-name>
    <url-pattern>/servlet/PathQuestionDemo2</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>PathQuestionDemo3</servlet-name>
    <servlet-
class>com.itheima.web.servlet.pathquestion.PathQuestionDemo3</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>PathQuestionDemo3</servlet-name>

```

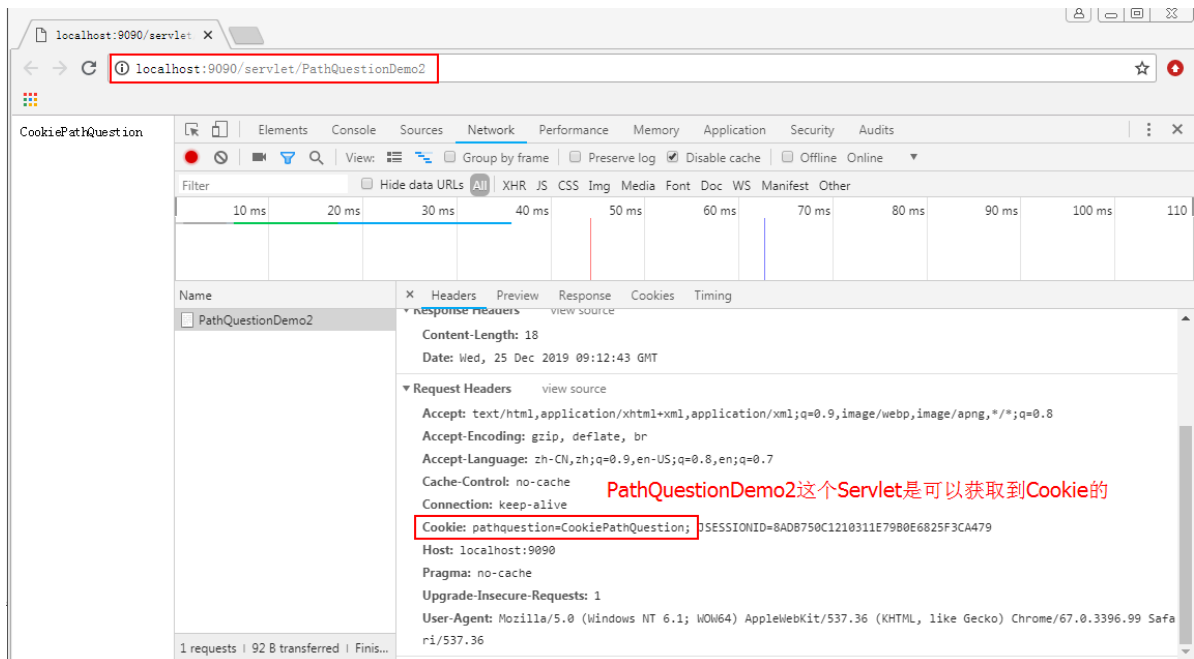
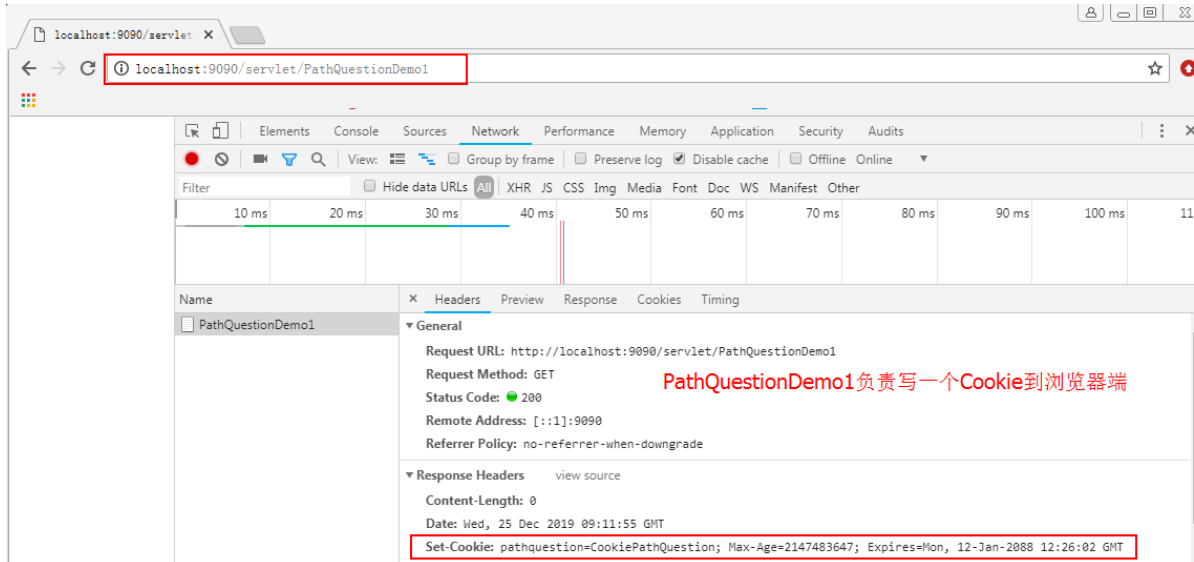
```
<url-pattern>/PathQuestionDemo3</url-pattern>
</servlet-mapping>
```

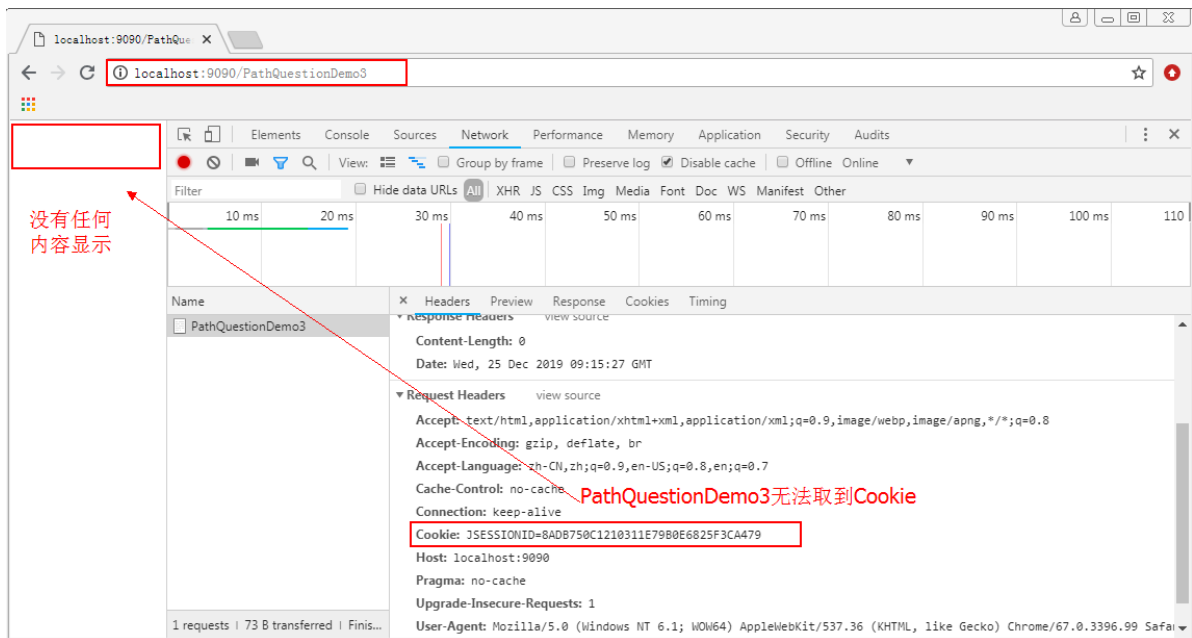
第四步：部署工程

沿用第一个案例中的工程部署即可。

4) 测试结果

通过分别运行PathQuestionDemo1, 2和3这三个Servlet, 我们发现由demo1写Cookie, 在demo2中可以取到, 但是到了demo3中就无法获取了, 如下图所示:





5) 路径问题的分析及总结

问题:

demo2和demo3谁能取到cookie?

答案:

demo2能取到, demo3取不到

分析:

首先, 我们要知道如何确定一个cookie?

那就是使用cookie的三个属性组合: **domain+path+name**

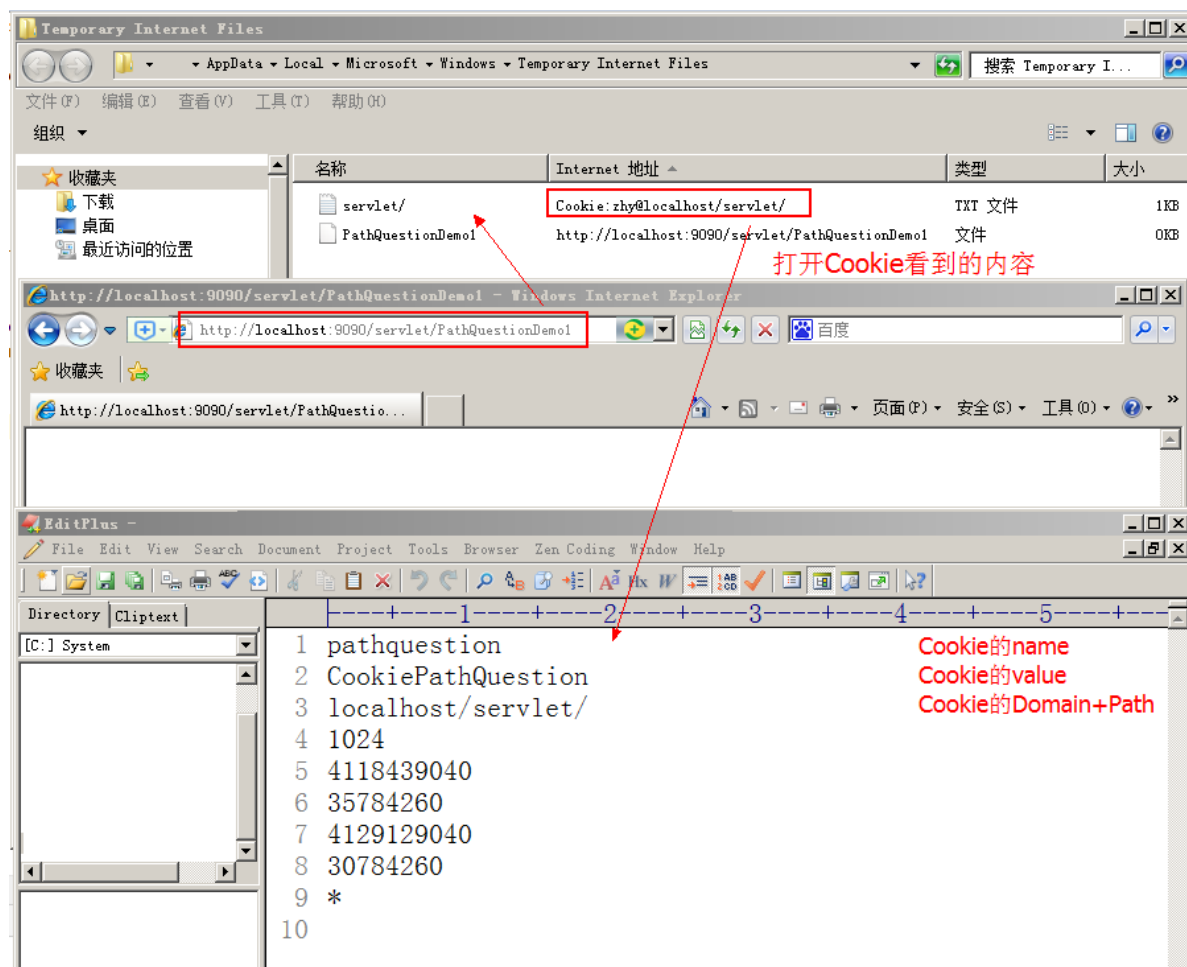
这里面, 同一个应用的domain是一样的, 在我们的案例中都是localhost.

并且, 我们取的都是同一个cookie, 所以name也是一样的, 都是pathquestion.

那么, 不一样的只能是path了。但是我们没有设置过cookie的path属性, 这就表明path是有默认值的。

接下来, 我们打开这个cookie来看一看, 在ie浏览器访问一次PathQuestionDemo1这个Servlet:

Cookie中的内容:



我们是通过demo1写的cookie，demo1的访问路径是：<http://localhost:9090/servlet/PathQuestionDemo1>

通过比较两个路径：请求资源地址和cookie的path，可以看出：cookie的path默认值是：请求资源URI，没有资源的部分（在我们的案例中，就是没有PathQuestionDemo1）。

客户端什么时候带cookie到服务器，什么时候不带？

就是看请求资源URI和cookie的path比较。

请求资源URI.startsWith(cookie的path) 如果返回的是true就带，如果返回的是false就不带。

简单的说：就是看谁的地址更精细

比如：Cookie的path： /国家 /省份 /城市

不带	请求资源URI	:	/国家	/省份		
带	请求资源URI	:	/国家	/省份	/城市	/区县

请求资源URI：/国家 /省份 .startsWith(Cookie的path： /国家 /省份 /城市) false 不带

请求资源URI：/国家 /省份 /城市 /区县 .startsWith(Cookie的path： /国家 /省份 /城市) true 带

在我们的案例中：

访问URL	URI部分	Cookie的Path	是否携带Cookie	能否取到Cookie
PathQuestionDemo2	/servlet/PathQuestionDemo2	/servlet/	带	能取到
PathQuestionDemo3	/PathQuestionDemo3	/servlet/	不带	不能取到

1.3 服务端会话管理概述

1.3.1 HttpSession概述

1) HttpSession对象介绍

它是Servlet规范中提供的一个接口。该接口的实现由Servlet规范的实现提供商提供。我们使用的是Tomcat服务器，它对Servlet规范进行了实现，所以HttpSession接口的实现由Tomcat提供。该对象用于提供一种通过多个页面请求或访问网站来标识用户并存储有关该用户的信息的方法。简单说它就是一个服务端会话对象，用于存储用户的会话数据。

同时，它也是Servlet规范中四大域对象之一的会话域对象。并且它也是用于实现数据共享的。但它与我们之前讲解的应用域和请求域是有区别的。

域对象	作用范围	使用场景
ServletContext	整个应用范围	当前项目中需要数据共享时，可以使用此域对象。
ServletRequest	当前请求范围	在请求或者当前请求转发时需要数据共享可以使用此域对象。
HttpSession	会话返回	在当前会话范围中实现数据共享。它可以在多次请求中实现数据共享。

2) HttpSession的获取

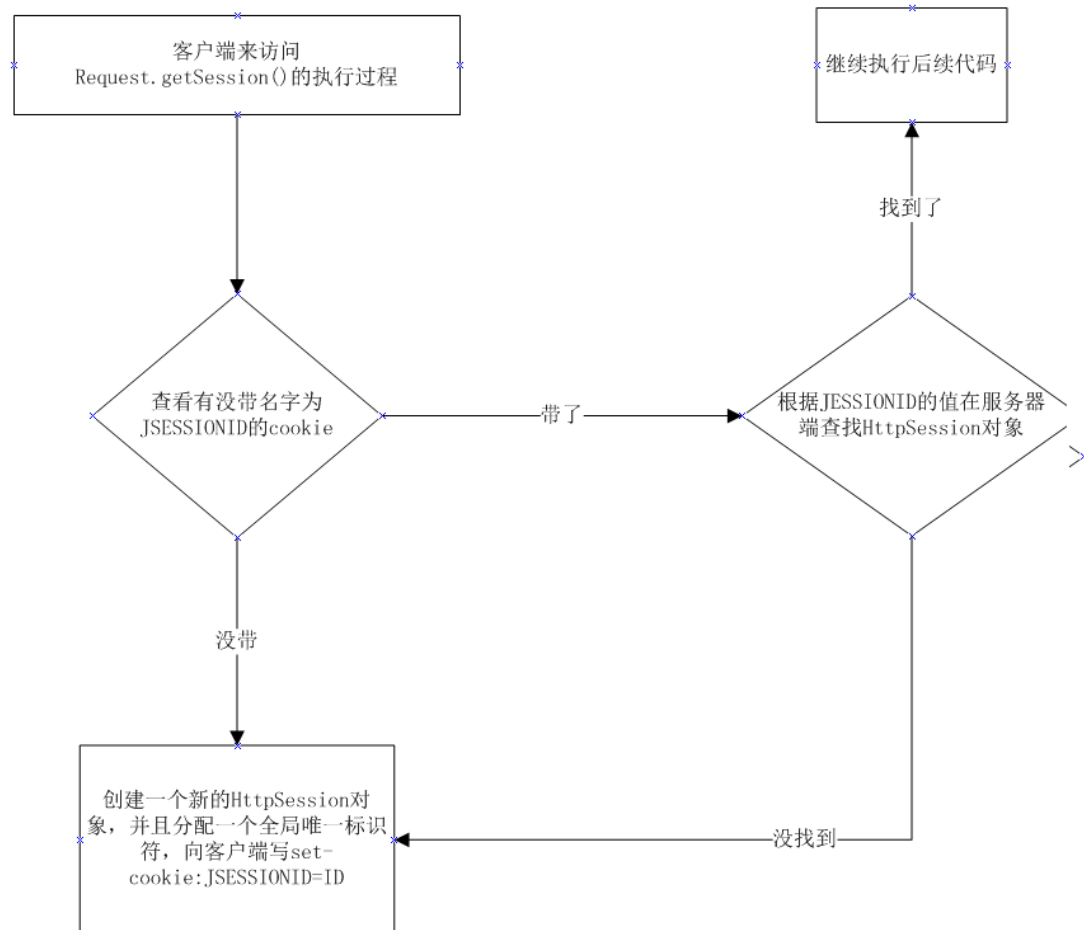
获取HttpSession是通过HttpServletRequest接口中的两个方法获取的，如下图所示：

```

397  /**
398  * Returns the current HttpSession associated with this request
399  * or, if there is no current session and create is true,
400  * returns a new session.
401  *
402  * If create is false and the request has no valid
403  * HttpSession, this method returns null.
404  *
405  * To make sure the session is properly maintained, you must call this
406  * method before the response is committed. If the container is using
407  * cookies to maintain session integrity and is asked to create a new
408  * session when the response is committed, an IllegalStateException is
409  * thrown.
410  *
411  * @param create
412  *         true to create a new session for this request if
413  *         necessary; false to return null if
414  *         there's no current session
415  * @return the HttpSession associated with this request or
416  *         null if create is false
417  *         and the request has no valid session
418  * @see #getSession()
419  */
420  public HttpSession getSession(boolean create);
421
422  /**
423  * Returns the current session associated with this request, or if the
424  * request does not have a session, creates one.
425  *
426  * @return the HttpSession associated with this request
427  * @see #getSession(boolean)
428  */
429  public HttpSession getSession();

```

这两个方法的区别：



需要注意的是：
`request.getSession(带参数)`；
参数是 `Boolean` 类型，表示当没找到指定 `session` 后是否自动创建。
`true` 是自动创建——也是默认值。
`false` 是不自动创建：在调用 `request.getSession(false)`；没找到时返回 `Null`

3) HttpSession的常用方法

Method Summary

All Methods	Instance Methods	Abstract Methods	Deprecated Methods
Modifier and Type		Method and Description	
Object		getAttribute(String name)	根据名称，从会话域中获取数据。
Enumeration<String>		getAttributeNames()	获取会话域中所有属性的名称。
long		getCreationTime()	获取HttpSession对象的创建时间，它是从1970年开始的毫秒值
String		getId()	获取会话的唯一标识。就是JSESSIONID的值。
long		getLastAccessedTime()	获取HttpSession的最后访问时间，它也是从1970年开始的毫秒值。
int		getMaxInactiveInterval()	获取最大间隔时间。
ServletContext		getServletContext()	获取ServletContext对象
HttpSessionContext		getSessionContext()	Deprecated. As of Version 2.1, this method is deprecated and has no replacement. It will be removed in a future version of the Java Servlet API.
Object		getValue(String name)	Deprecated. As of Version 2.2, this method is replaced by getAttribute(java.lang.String).
String[]		getValueNames()	Deprecated. As of Version 2.2, this method is replaced by getAttributeNames()
void		invalidate()	让HttpSession立即失效。
boolean		isNew()	判断会话是否是新的。当客户端不加入会话，或者不知道有此会话时，返回true。
void		putValue(String name, Object value)	Deprecated. As of Version 2.2, this method is replaced by setAttribute(java.lang.String, java.lang.Object)
void		removeAttribute(String name)	根据名称，移除会话域中的数据。
void		removeValue(String name)	Deprecated. As of Version 2.2, this method is replaced by removeAttribute(java.lang.String)
void		setAttribute(String name, Object value)	把名称和值设置到会话域中。
void		setMaxInactiveInterval(int interval)	设置最大间隔时间。

1.3.2 HttpSession的入门案例

1) 需求说明

在请求HttpSessionDemo1这个Servlet时，携带用户名信息，并且把信息保存到会话域中，然后从HttpSessionDemo2这个Servlet中获取登录信息。

2) 案例目的

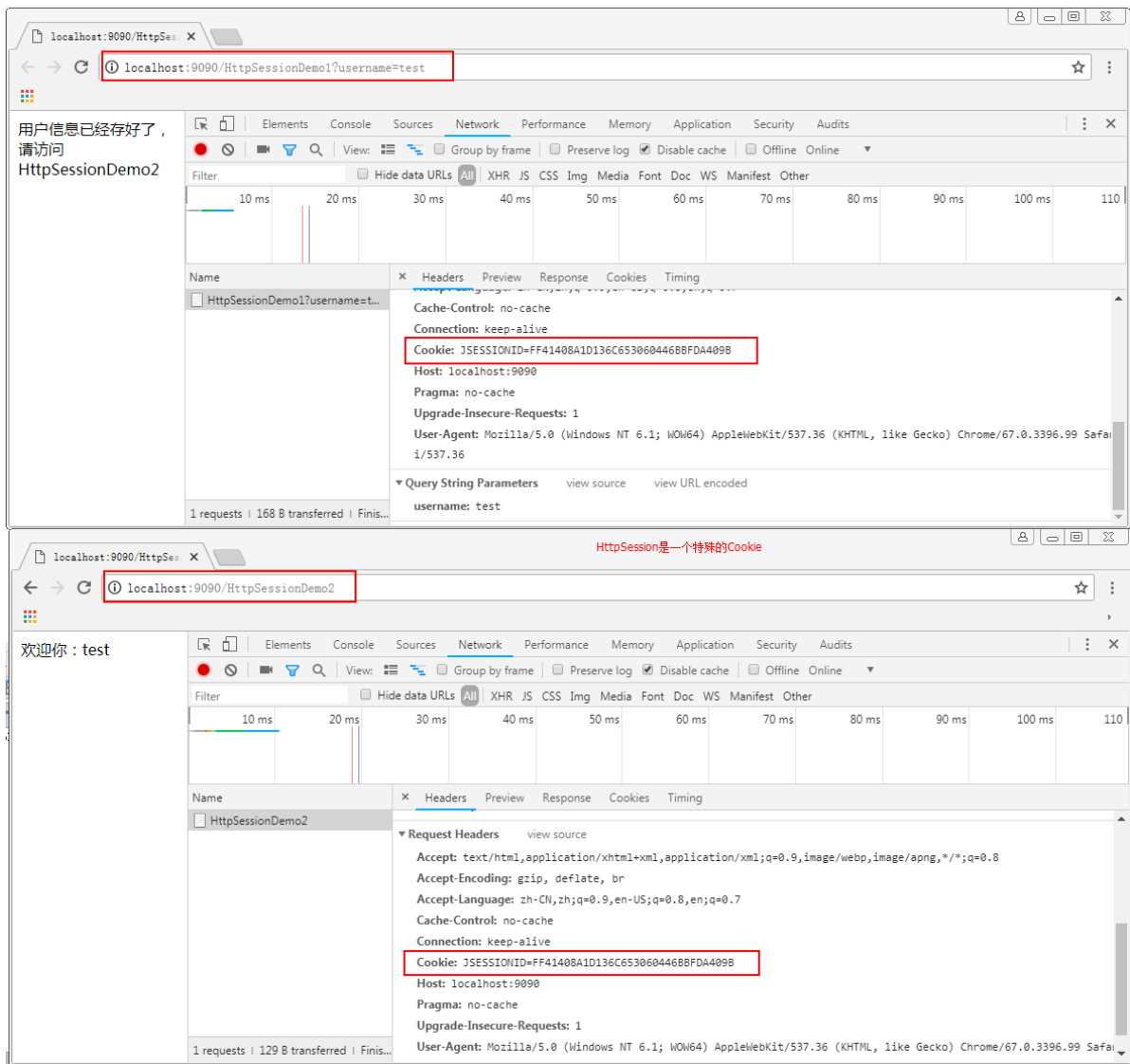
通过本案例的讲解，同学们可以清楚的认识到的作用，即多次请求间的数据共享。因为是两次请求，请求域肯定不一样了，所以不能用请求域实现。

最终掌握HttpSession对象的获取和使用。

3) 原理分析

HttpSession，它虽然是服务端会话管理技术的对象，但它本质仍是一个Cookie。是一个由服务器自动创建的特殊的Cookie，Cookie的名称就是JSESSIONID，Cookie的值是服务器分配的一个唯一的标识。

当我们使用HttpSession时，浏览器在没有禁用Cookie的情况下，都会把这个Cookie带到服务器端，然后根据唯一标识去查找对应的HttpSession对象，找到了，我们就可以直接使用了。下图就是我们入门案例中，HttpSession分配的唯一标识，同学们可以看到两次请求的JSESSIONID的值是一样的：



1.3.3 HttpSession的钝化和活化

什么是持久态

把长时间不用，但还不到过期时间的HttpSession进行序列化，写到磁盘上。

我们把HttpSession持久态也叫做钝化。（与钝化相反的，我们叫活化。）

什么时候使用持久化

第一种情况：当访问量很大时，服务器会根据getLastAccessTime来进行排序，对长时间不用，但是还没到过期时间的HttpSession进行持久化。

第二种情况：当服务器进行重启的时候，为了保持客户HttpSession中的数据，也要对HttpSession进行持久化

注意

HttpSession的持久化由服务器来负责管理，我们不用关心。

只有实现了序列化接口的类才能被序列化，否则不行。

2 页面技术

2.1 JSP基础

2.1.1 JSP简介

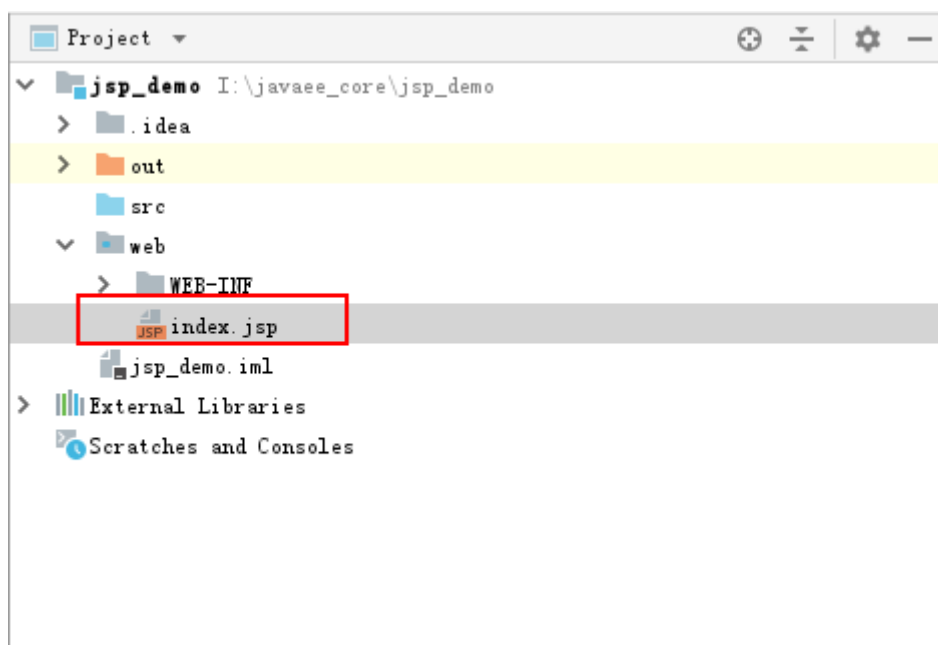
JSP全称是Java Server Page，它和Servlet一样，也是sun公司推出的一套开发动态web资源的技术，称为JSP/Servlet规范。JSP的本质其实就是一个Servlet。

2.1.2 JSP和HTML以及Servlet的适用场景

类别	适用场景
HTML	只能开发静态资源，不能包含java代码，无法添加动态数据。
Servlet	写java代码，可以输出页面内容，但是很不方便，开发效率极低。
JSP	它包括了HTML的展示技术，同时具备Servlet输出动态资源的能力。但是不适合作为控制器来用。

2.1.3 JSP简单入门

创建JavaWeb工程



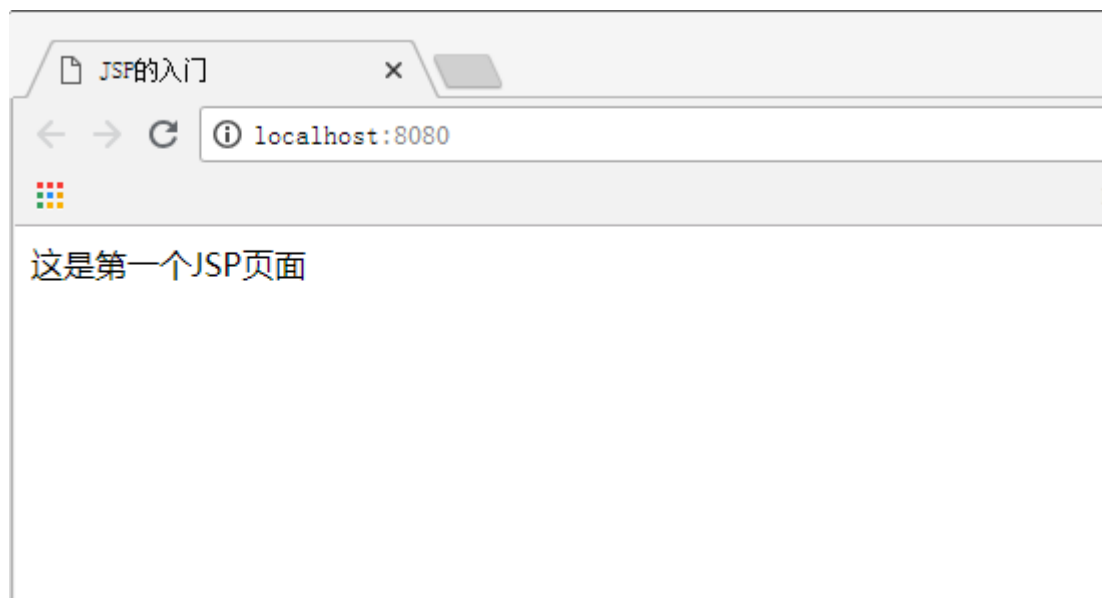
在index.jsp中填写内容

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
<title>JSP的入门</title>
</head>
<body>
    这是第一个JSP页面
</body>
</html>
```

部署项目

沿用会话管理工程的部署方式即可。

测试运行



2.1.4 JSP说明

写在之前：明确JSP就是一个Servlet。是一个特殊的Servlet。

JSP的原理：

客户端提交请求

——Tomcat服务器解析请求地址

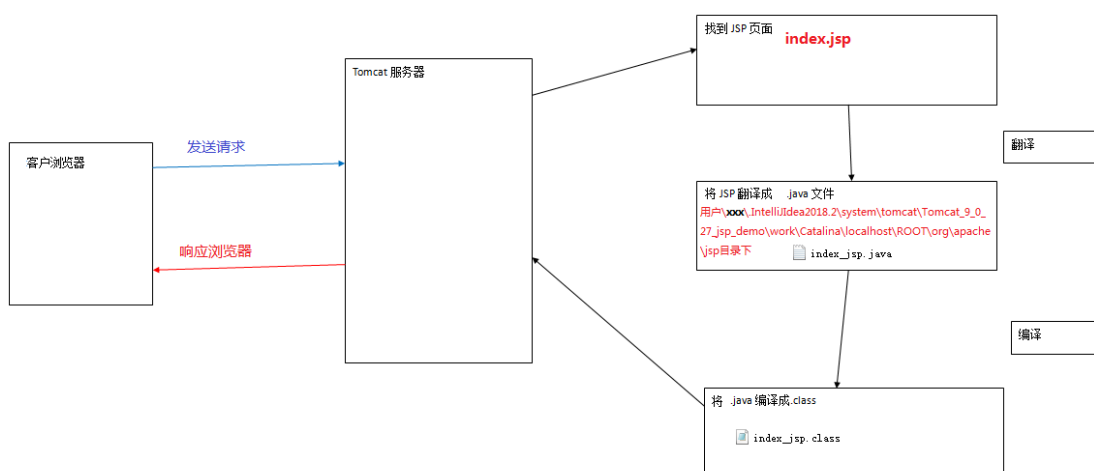
——找到JSP页面

——Tomcat将JSP页面翻译成Servlet的java文件

——将翻译好的.java文件编译成.class文件

——返回到客户浏览器上。

1) 执行过程分析图



2) JSP的.java文件内容分析

当我们打开index.jsp翻译的java文件看到的就是 `public final class index_jsp extends org.apache.jasper.runtime.HttpJspBase` 类的声明，然后我们在Tomcat的源码中找到类的声明，如下图：

```
package org.apache.jasper.runtime;

import java.io.IOException;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.jsp.HttpJspPage;

import org.apache.jasper.compiler.Localizer;

/**
 * This is the super class of all JSP-generated servlets.
 *
 * @author Anil K. Vijendran
 */
public abstract class HttpJspBase extends HttpServlet implements HttpJspPage {
```

这张图一出场，就表明我们写的JSP它本质就是一个HttpServlet了。


```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class index_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent,
        org.apache.jasper.runtime.JspSourceImports {

    out = pageContext.getOut();
    _jspx_out = out;


    out.write("\r\n");
    out.write("<html>\r\n");
    out.write("<head>\r\n");
    out.write("<title>JSP的入门</title>\r\n");
    out.write("</head>\r\n");
    out.write("<body>\r\n");
    out.write("    这是第一个JSP页面\r\n");
    out.write("</body>\r\n");
    out.write("</html>");
```



```
graph TD
    Servlet --> JspPage
    JspPage --> HttpJspPage
```

```
public interface HttpJspPage extends JspPage {

    /** The _jspService() method corresponds to the body of the JSP page. This
     * method is defined automatically by the JSP container and should never
     * be defined by the JSP page author.
     */
    <p>
    * If a superclass is specified using the extends attribute, that
    * superclass may choose to perform some actions in its service() method
    * before or after calling the _jspService() method. See using the extends
    * attribute in the JSP_Engine chapter of the JSP specification.
    *
    * @param request Provides client request information to the JSP.
    * @param response Assists the JSP in sending a response to the client.
    * @throws ServletException Thrown if an error occurred during the
    * processing of the JSP and that the container should take
    * appropriate action to clean up the request.
    * @throws IOException Thrown if an error occurred while writing the
    response for this page.
    */
    public void _jspService(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException;
}
```



同时，我们在index_jsp.java文件中找到了输出页面的代码，并且在浏览器端查看源文件，看到的内容是一样的。这也就是说明，我们的浏览器上的内容，在通过jsp展示时，本质都是用out.write()输出出来的。

讲到这里，我们应该清楚的认识到了，JSP它是一个特殊的Servlet，主要是用于展示动态数据。它展示的方式是用流把数据输出出来，而我们在使用JSP时，涉及HTML的部分，都与HTML的用法一致，这部分称为jsp中的模板元素，在开发过程中，先写好这些模板元素，因为它们决定了页面的外观。

2.2 JSP应用

2.2.1 JSP语法

1) Java代码块

在jsp中，可以使用java脚本代码。形式为：**<% 此处写java代码 %>**

但是，在实际开发中，极少使用此种形式编写java代码。同时需要注意的是：

```
<%  
量  
%>
```

在里面写java程序脚本需要注意：这里面的内容由tomcat负责翻译，翻译之后是service方法的成员变

示例：

```
<!--Java代码块-->  
<% out.println("这是Java代码块");%>  
<hr/>
```

2) JSP表达式

在jsp中，可以使用特定表达式语法，形式为：**<%=表达式%>**

jsp在翻译完后是out.print(表达式内容);

所以：<%out.print("当前时间");%>和<%= "当前时间"%>是一样的。

在实际开发中，这种表达式语法用的也很少使用。

示例：

```
<!--JSP表达式-->  
<%= "这是JSP表达式"%><br/>  
就相当于<br/>  
<%out.println("这是没有JSP表达式输出的");%>
```

3) JSP声明

在JSP中也可以声明一些变量，方法，静态方法，形式为：**<%! 声明的内容 %>**

使用JSP声明需要注意：

```
<%!  
需要注意的是： 写在里面的内容将会被tomcat翻译成全局的属性或者类方法。  
%>
```

示例：

```
<!--JSP声明-->  
<%! String str = "声明语法格式";%>  
<%=str%>
```

4) JSP注释

在使用JSP时，它有自己的注释，形式为：**<!--注释--%>**

需要注意的是：

在jsp中可以使用html的注释，但是只能注释html元素，不能注释java程序片段和表达式。同时，被html注释部分会参与翻译，并且会在浏览器上显示

jsp的注释不仅可以注释java程序片段，也可以注释html元素，并且被jsp注释的部分不会参与翻译成.java文件，也不会浏览器上显示。

示例：

```
<%--JSP注释--%>
<!--HTML注释-->
```

5) 语法的示例

JSP语法完整示例代码

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>JSP语法</title>
</head>
<body>

<!--Java代码块-->
<% out.println("这是Java代码块");%>
<hr/>

<!--JSP表达式-->
<%= "这是JSP表达式"%><br/>
就相当于<br/>
<%out.println("这是没有JSP表达式输出的");%>

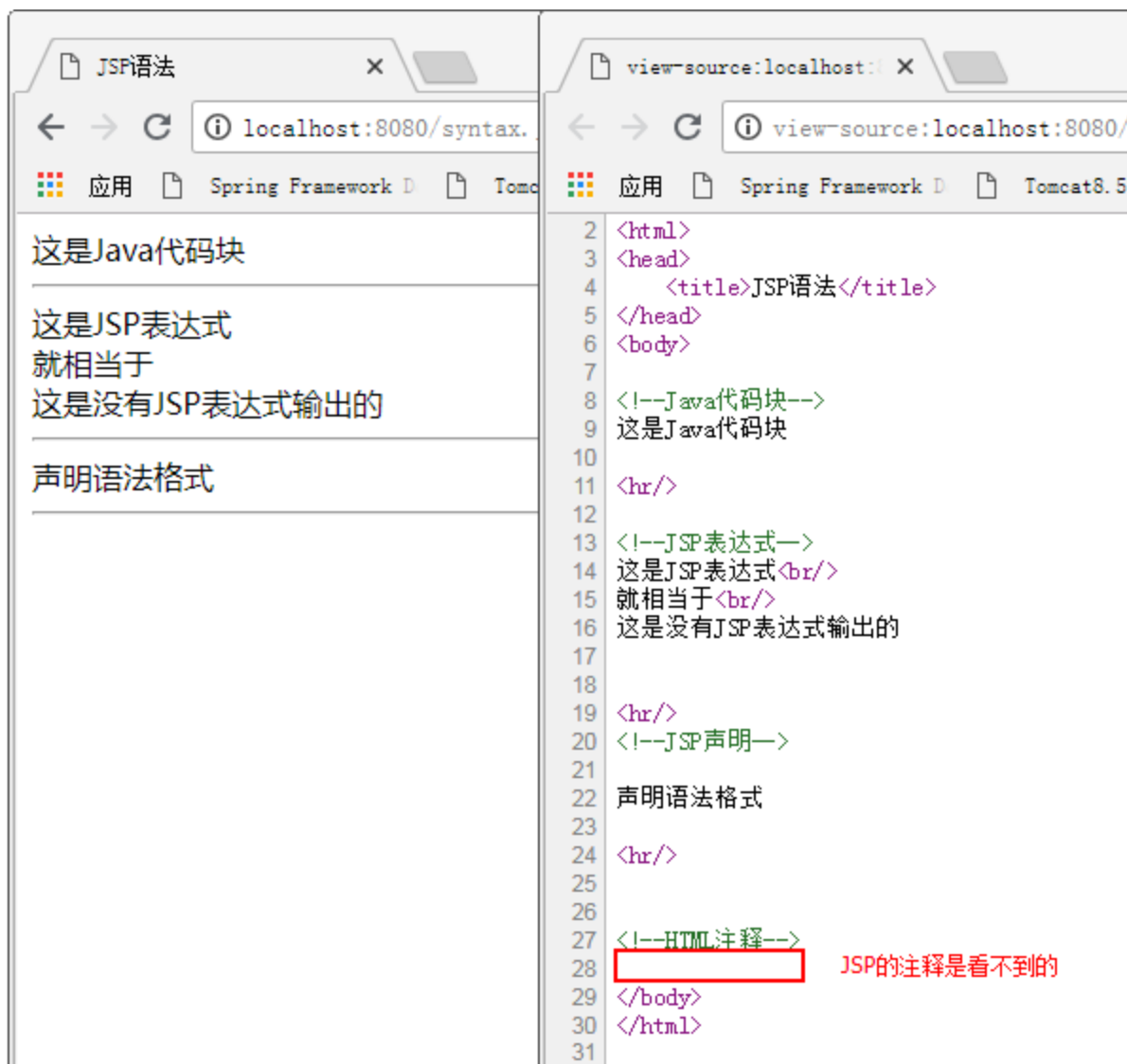
<hr/>
<!--JSP声明-->
<%! String str = "声明语法格式";%>
<%=str%>

<hr/>

<%--JSP注释--%>
<!--HTML注释-->

</body>
</html>
```

JSP语法运行结果



2.2.2 JSP指令

1) page指令

language:告知引擎，脚本使用的是java，默认是java，支持java。不写也行。

extends:告知引擎，JSP对应的Servlet的父类是哪个，不需要写，也不需要改。

import:告知引擎，导入哪些包（类）。

注意：引擎会自动导入：

`java.lang.*; javax.servlet.*; javax.servlet.http.*; javax.servlet.jsp.*`

导入的形式：

`<%@page import="java.util.Date,java.util.UUID"%>`或者：

`<%@page import="java.util.Date"%>`

`<%@page import="java.util.UUID"%>` 用Eclipse：Alt+/ 自动导入

session:告知引擎是否产生HttpSession对象，即是否在代码中调用request.getSession()。默认是true。

buffer:jspWriter用于输出JSP内容到页面上。告知引擎，设定他的缓存大小。默认8kb。

errorPage:告知引擎，当前页面出现异常后，应该转发到哪个页面上（路径写法：/代表当前应用）

小贴士：当在errorpage上使用了isErrorPage=true之后，ie8有时候不能正常显示

配置全局错误页面: web.xml

```
<error-page>
    <exception-type>java.lang.Exception</exception-type>
    <location>/error.jsp</location>
</error-page>
<error-page>
    <error-code>404</error-code>
    <location>/404.html</location>
</error-page>
```

当使用了全局错误页面,就无须再写errorPage来实现转到错误页面,而是由服务器负责跳转到错误页面。

isErrorPage: 告知引擎,是否抓住异常。如果该属性为true,页面中就可以使用exception对象,打印异常的详细信息。默认值是false。

contentType: 告知引擎,响应正文的MIME类型。contentType="text/html;charset=UTF-8"

相当于response.setContentType("text/html;charset=UTF-8");

pageEncoding: 告知引擎,翻译jsp时(从磁盘上读取jsp文件)所用的码表。pageEncoding="UTF-8"相当于告知引擎用UTF-8读取JSP

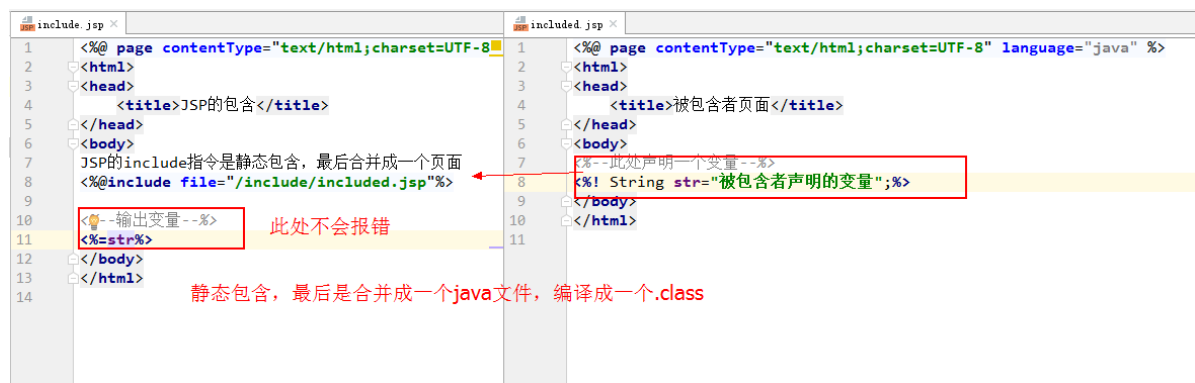
isELIgnored*: 告知引擎,是否忽略EL表达式,默认值是false,不忽略。

2) include指令

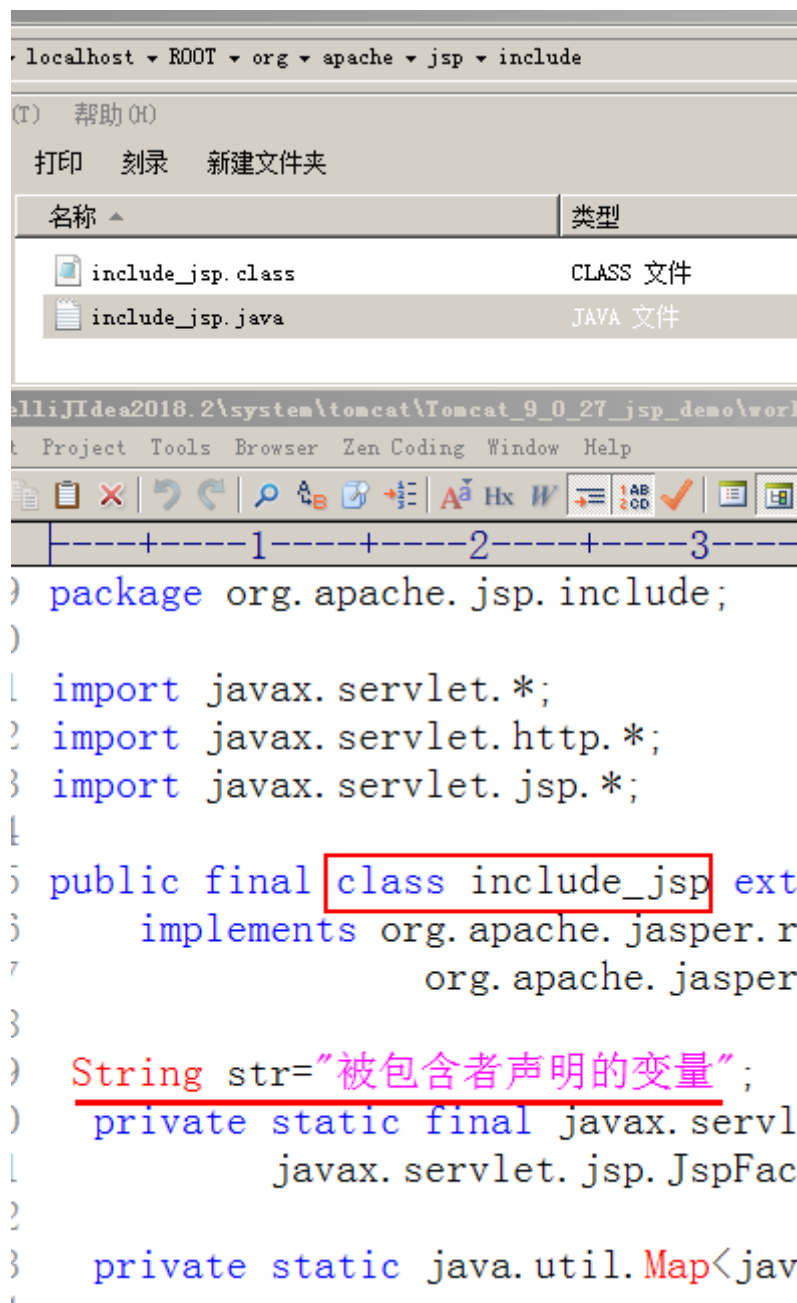
语法格式: <%@include file="" %>该指令是包含外部页面。

属性: file, 以/开头,就代表当前应用。

使用示例



静态包含的特点



3) taglib指令

语法格式: <%taglib uri="" prefix=""%>

作用: 该指令用于引入外部标签库。html标签和jsp标签不用引入。

属性:

uri: 外部标签的URI地址。

prefix: 使用标签时的前缀。

2.2.3 JSP细节

1) 九大隐式对象

什么是隐式对象呢? 它指的是在jsp中, 可以不声明就直接使用的对象。它只存在于jsp中, 因为java类中的变量必须要先声明再使用。其实jsp中的隐式对象也并非未声明, 只是它是在翻译成java文件时声明的。所以我们在jsp中可以直接使用。

隐式对象名称	类型	备注
request	javax.servlet.http.HttpServletRequest	
response	javax.servlet.http.HttpServletResponse	
session	javax.servlet.http.HttpSession	Page指令可以控制开关
application	javax.servlet.ServletContext	
page	Java.lang.Object	当前jsp对应的servlet引用实例
config	javax.servlet.ServletConfig	
exception	java.lang.Throwable	page指令有开关
out	javax.servlet.jsp.JspWriter	字符输出流，相当于printwriter
pageContext	javax.servlet.jsp.PageContext	很重要

2) PageContext对象

简介

它是JSP独有的对象，Servlet中没有这个对象。本身也是一个域（作用范围）对象，但是它可以操作其他3个域对象中的属性。而且还可以获取其他8个隐式对象。

生命周期

它是一个局部变量，所以它的生命周期随着JSP的创建而诞生，随着JSP的结束而消失。每个JSP页面都有一个独立的PageContext。

常用方法

Method Summary	
All Methods	Instance Methods
Abstract Methods	Concrete Methods
Modifier and Type	Method and Description
abstract void	forward(String relativeUriPath) 使用当前请求和响应对象实现转发
ErrorData	getErrorData() 提供对错误信息的方便访问。
abstract Exception	getException() 异常对象（异常）的当前值。
abstract Object	getPage() 页面对象的当前值（在Servlet环境中，这是javax.Servlet.Servlet的一个实例）。
abstract ServletRequest	getRequest() 获取当前请求对象
abstract ServletResponse	getResponse() 获取当前响应对象
abstract ServletConfig	getServletConfig() 获取Servlet的配置对象
abstract ServletContext	getServletContext() 获取应用域对象
abstract HttpSession	getSession() 获取当前会话对象
abstract void	handlePageException(Exception e) 此方法旨在通过将异常转发到此JSP的指定错误页来处理未处理的“页”级异常
abstract void	handlePageException(Throwable t) 它和上面的方法作用用于，只是支持的参数是一个Throwable对象
abstract void	include(String relativeUriPath) 根据提供的路径，包含其他的资源。它是动态包含。
abstract void	include(String relativeUriPath, boolean flush) 它和上面的方法作用一样。当flush为true时，和上面的方法一模一样。当为false时，不会刷出out的内容。
abstract void	initialize(Servlet servlet, ServletRequest request, ServletResponse response, String errorPageURL, boolean needsSession, int bufferSize, boolean autoFlush) 用于初始化PageContext对象
BodyContent	pushBody() 返回一个BodyContent对象，里面保存的是当前JspWriter中的内容。
abstract void	release() 重置PageContext的内部状态，释放所有内部引用

在上图中，同学们发现没有页面域操作的方法，其实是定义在了PageContext的父类JspContext中，如下图所示：

Method Summary

All Methods	Instance Methods	Abstract Methods	Concrete Methods	Deprecated Methods
Modifier and Type		Method and Description		
abstract Object		findAttribute(String name)	根据名称，从四大域中查找数据。查找方式是由小到大。找不到时，返回Null。	
abstract Object		getAttribute(String name)	根据名称，从页面域中获取数据	
abstract Object		getAttribute(String name, int scope)	根据名称和指定的返回获取数据。scope指的是四大域之一。	
abstract Enumeration<String>		getAttributeNamesInScope(int scope)	根据指定的返回，获取域中所有属性的名称	
abstract int		getAttributeScope(String name)	根据名称，获取当前名称出现的作用域	
abstract ELContext		getELContext()	获取EL表达式的上下文对象	
abstract ExpressionEvaluator		getExpressionEvaluator()	Deprecated. As of JSP 2.1, replaced by JspApplicationContext.getExpressionFactory()	
abstract JspWriter		getOut()	获取JSP的输出对象	
abstract VariableResolver		getVariableResolver()	Deprecated. As of JSP 2.1, replaced by ELContext.getELResolver(), which can be obtained by jspContext.getELContext().getELResolver().	
JspWriter		popBody()	返回由匹配的pushBody()保存的前一个JspWriter“out”，并更新JspContext的page scope属性命名空间中“out”属性的值。	
JspWriter		pushBody(Writer writer)	返回一个用于发送输出的新JspWriter对象	
abstract void		removeAttribute(String name)	根据名称移除页面域中的数据	
abstract void		removeAttribute(String name, int scope)	根据名称和作用范围，移除指定域中的数据	
abstract void		setAttribute(String name, Object value)	根据名称和值，在页面域中添加数据	
abstract void		setAttribute(String name, Object value, int scope)	根据名称和值，以及作用范围，在指定域中添加数据	

3) 四大域对象

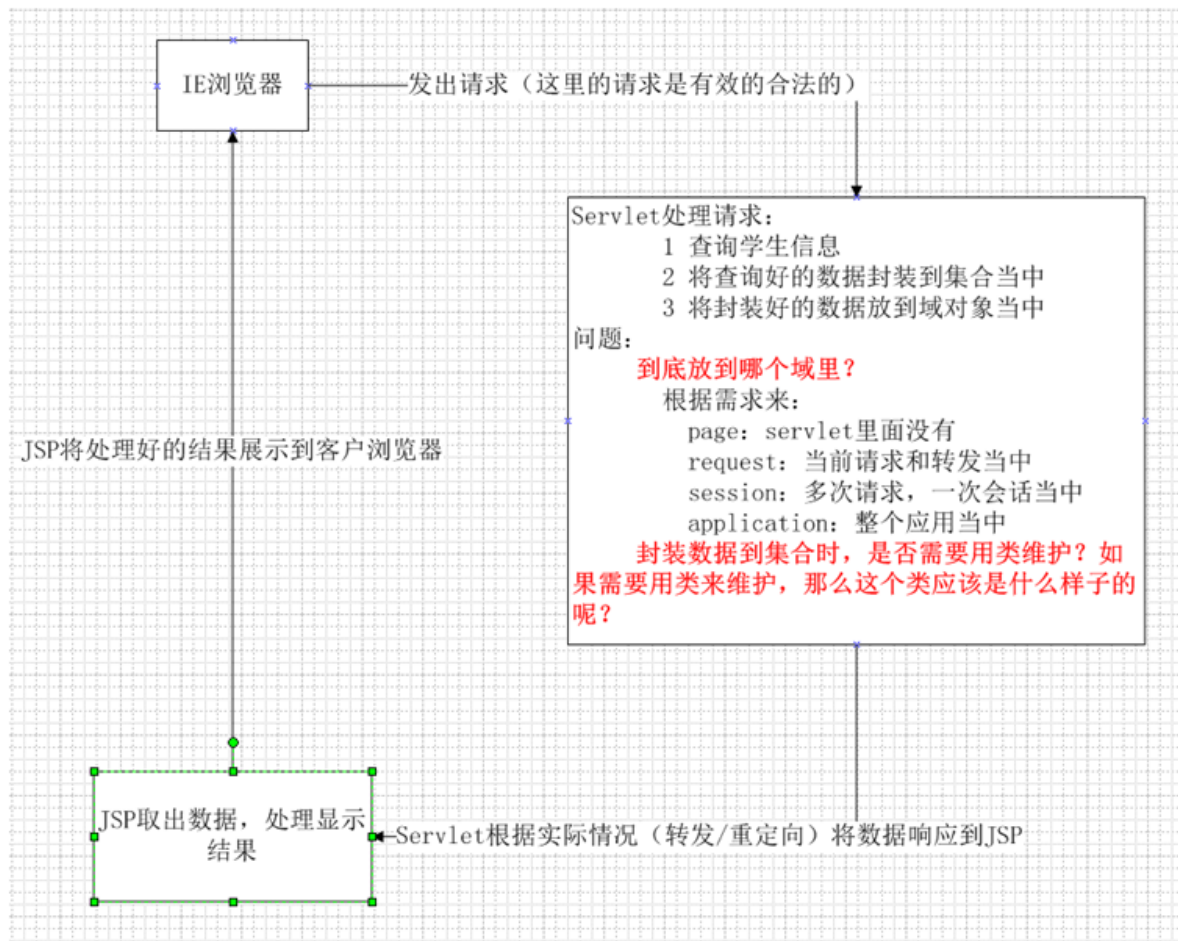
域对象名称	范围	级别	备注
PageContext	页面范围	最小，只能在当前页面用	因范围太小，开发中用的很少
ServletRequest	请求范围	一次请求或当期请求转发用	当请求转发之后，再次转发时请求域丢失
HttpSession	会话范围	多次请求数据共享时使用	多次请求共享数据，但不同的客户端不能共享
ServletContext	应用范围	最大，整个应用都可以使用	尽量少用，如果对数据有修改需要做同步处理

2.2.4 JSP最佳实战-MVC模型

Servlet：擅长处理业务逻辑，不擅长输出显示界面。在web开发中多用于控制程序逻辑（流程）。所以我们称之为：控制器。

JSP：擅长显示界面，不擅长处理程序逻辑。在web开发中多用于展示动态界面。所以我们称之为：视图。

例如:



M: model , 通常用于封装数据, 封装的是数据模型。

V: view , 通常用于展示数据。动态展示用jsp页面, 静态数据展示用html。

C: controller , 通常用于处理请求和响应。一般指的是Servlet。

3 综合案例-学生管理系统升级

3.1 登录功能实现

3.1.1 创建一个web项目, 在 web 目录下创建一个 index.jsp。

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
<title>学生管理系统首页</title>
</head>
<body>
<!--
获取会话域中的数据
如果获取到了则显示添加和查看功能的超链接
如果没获取到则显示登录功能的超链接
-->
<% Object username = session.getAttribute("username");
    if(username == null) {
%>
<a href="/stu/login.jsp">请登录</a>
<%} else {%>
<a href="/stu/addStudent.jsp">添加学生</a>
```

```

        <a href="/stu/listStudentServlet">查看学生</a>
    <%}%>
</body>
</html>

```

3.1.2 在 web 目录下创建一个 login.jsp。实现登录页面

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>学生登录</title>
</head>
<body>
    <form action="/stu/loginStudentServlet" method="get" autocomplete="off">
        姓名: <input type="text" name="username"> <br>
        密码: <input type="password" name="password"> <br>
        <button type="submit">登录</button>
    </form>
</body>
</html>

```

3.1.3 创建 LoginStudentServlet，获取用户名和密码

```

package com.itheima.servlet;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * 学生登录
 */
@WebServlet("/loginStudentServlet")
public class LoginStudentServlet extends HttpServlet{
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        //1. 获取用户名和密码
        String username = req.getParameter("username");
        String password = req.getParameter("password");

        //2. 判断用户名
        if(username == null || "".equals(username)) {
            //2.1 用户名为空 重定向到登录页面
            resp.sendRedirect("/stu/login.jsp");
            return;
        }
    }
}

```

```

//2.2用户名不为空 将用户名存入会话域中
req.getSession().setAttribute("username",username);

//3.重定向到首页index.jsp
resp.sendRedirect("/stu/index.jsp");
}

@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    doGet(req,resp);
}
}

```

3.2添加功能实现

3.2.1 在 web 目录下创建一个 addStudent.jsp，实现添加学生的表单项

```

<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>添加学生</title>
</head>
<body>
<form action="/stu/addStudentServlet" method="get" autocomplete="off">
    学生姓名: <input type="text" name="username"> <br>
    学生年龄: <input type="number" name="age"> <br>
    学生成绩: <input type="number" name="score"> <br>
    <button type="submit">保存</button>
</form>
</body>
</html>

```

3.2.2 创建 AddStudentServlet，获取学生信息并保存到文件中

```

package com.itheima.servlet;

import com.itheima.bean.Student;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

/*
实现添加功能

```

```

*/
@WebServlet("/addStudentServlet")
public class AddStudentServlet extends HttpServlet{
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        //1. 获取表单中的数据
        String username = req.getParameter("username");
        String age = req.getParameter("age");
        String score = req.getParameter("score");

        //2. 创建学生对象并赋值
        Student stu = new Student();
        stu.setUsername(username);
        stu.setAge(Integer.parseInt(age));
        stu.setScore(Integer.parseInt(score));

        //3. 将学生对象的数据保存到d:\\stu.txt文件中
        BufferedWriter bw = new BufferedWriter(new
        FileWriter("d:\\stu.txt",true));
        bw.write(stu.getUsername() + "," + stu.getAge() + "," + stu.getScore());
        bw.newLine();
        bw.close();

        //4. 通过定时刷新功能响应给浏览器
        resp.setContentType("text/html;charset=UTF-8");
        resp.getWriter().write("添加成功。2秒后自动跳转到首页...");
        resp.setHeader("Refresh","2;URL=/stu/index.jsp");
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        doGet(req,resp);
    }
}

```

3.3 查看学生功能

3.3.1 创建 ListStudentServlet, 读取文件中的学生信息到集合中

- 1、将集合添加到会话域中
- 2、重定向到 listStudent.jsp 页面上

```

package com.itheima.servlet;

import com.itheima.bean.Student;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;

```

```

import javax.servlet.http.HttpServletResponse;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

/*
    实现查看功能
*/
@WebServlet("/listStudentServlet")
public class ListStudentServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        //1.创建字符输入流对象，关联读取的文件
        BufferedReader br = new BufferedReader(new FileReader("d:\\stu.txt"));

        //2.创建集合对象，用于保存Student对象
        ArrayList<Student> list = new ArrayList<>();

        //3.循环读取文件中的数据，将数据封装到Student对象中。再把多个学生对象添加到集合中
        String line;
        while((line = br.readLine()) != null) {
            //张三,23,95
            Student stu = new Student();
            String[] arr = line.split(",");
            stu.setUsername(arr[0]);
            stu.setAge(Integer.parseInt(arr[1]));
            stu.setScore(Integer.parseInt(arr[2]));
            list.add(stu);
        }

        //4.将集合对象存入会话域中
        req.getSession().setAttribute("students",list);

        //5.重定向到学生列表页面
        resp.sendRedirect("/stu/listStudent.jsp");
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        doGet(req,resp);
    }
}

```

3.3.2 在 web 目录下创建一个 listStudent.jsp

```

<%@ page import="com.itheima.bean.Student" %>
<%@ page import="java.util.ArrayList" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>查看学生</title>
</head>

```

```
<body>
  <table width="600px" border="1px">
    <tr>
      <th>学生姓名</th>
      <th>学生年龄</th>
      <th>学生成绩</th>
    </tr>
    <% ArrayList<Student> students = (ArrayList<Student>)
session.getAttribute("students");
    for(Student stu : students) {
      <%>
      <tr align="center">
        <td><%=stu.getUsername()%></td>
        <td><%=stu.getAge()%></td>
        <td><%=stu.getScore()%></td>
      </tr>
      <%}%>
    </table>
  </body>
</html>
```