目录

[一、HTTP 协议详解 2](#_Toc31834)

[1. web 交互的基本流程 2](#_Toc16815)

[2. HTTP 的概念和介绍 2](#_Toc213)

[3. HTTP 的交互流程 2](#_Toc5088)

[4. HTTP 协议之请求格式 3](#_Toc12155)

[5. HTTP 协议之请求方式 3](#_Toc19430)

[6. HTTP 协议之响应 4](#_Toc8358)

[二、 Tomcat服务器介绍和使用 5](#_Toc32082)

[1. Tomcat 服务的下载和安装 5](#_Toc3003)

[2.第一个 Web 程序 6](#_Toc5032)

[3. Servlet 介绍 7](#_Toc16665)

[1. Servlet 的生命周期 8](#_Toc3341)

[2.Service和doGet和doPost方法的区别 9](#_Toc138)

[3.Request 对象: 11](#_Toc25461)

[4.Response 对象： 13](#_Toc30782)

[5.请求转发 16](#_Toc24628)

[6.request对象作用域 16](#_Toc22676)

[7.重定向 17](#_Toc13594)

[servlet： 18](#_Toc2883)

[service 31](#_Toc20364)

[pojo： 32](#_Toc20631)

[dao： 35](#_Toc15665)

[四、Cookie学习 39](#_Toc23527)

[五、Session 学习 40](#_Toc30502)

[六、ServletContext 对象 41](#_Toc4209)

[七、ServletConfig对象 42](#_Toc20455)

一、HTTP 协议详解

1. web 交互的基本流程  
客户端根据用户输入的地址信息请求服务器， 服务器在接收到用户的请求后进行处理， 然后将处理结果响应给客户端， 客户端将响应结果展示给用户。  
  
请求： 客户端根据用户地址信息将数据发送给服务器的过程  
响应： 服务器将请求的处理结果发送给浏览器的过程

问题：客户端也就是浏览器的版本是有很多的， 服务器的版本也是有很多的， 如何实现不同版本的浏览器和不同版本的服务器之间的数据交互呢？

解决：规范浏览器和服务器的数据交互的格式。  
实现：HTTP 协议

2. HTTP 的概念和介绍  
概念：超文本传输协议（Hyper Text Transfer Protocol）  
作用：规范了浏览器和服务器的数据交互  
特点：

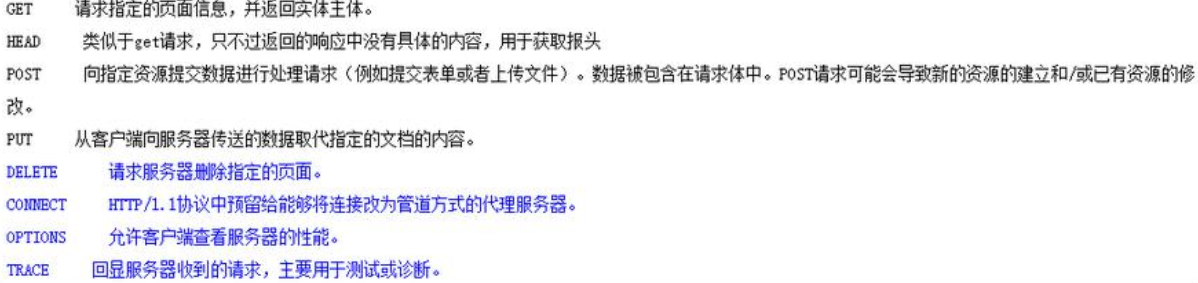
1. 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有get、post。每种方法规定了客户与服务器联系的类型不同。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快  
    ② 灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由Content-Type 加以标记。  
    ③ 无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。  
    ④ 无状态：HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。  
    ⑤ 支持 B/S 及 C/S 模式。  
    ⑥ HTTP1.1 版本后支持可持续连接

3. HTTP 的交互流程  
HTTP 的交互流程一般分为四个步骤(一次完整的请求)：  
步骤一:  
客户端和服务器端建立连接  
步骤二：  
客户端发送请求数据到服务器端(HTTP协议)  
步骤三：  
服务器端接收到请求后进行处理，然后将处理结果响应客户端(HTTP 协议)  
步骤四：  
关闭客户端和服务器端的连接(HTTP1.1 后不会立即关闭)

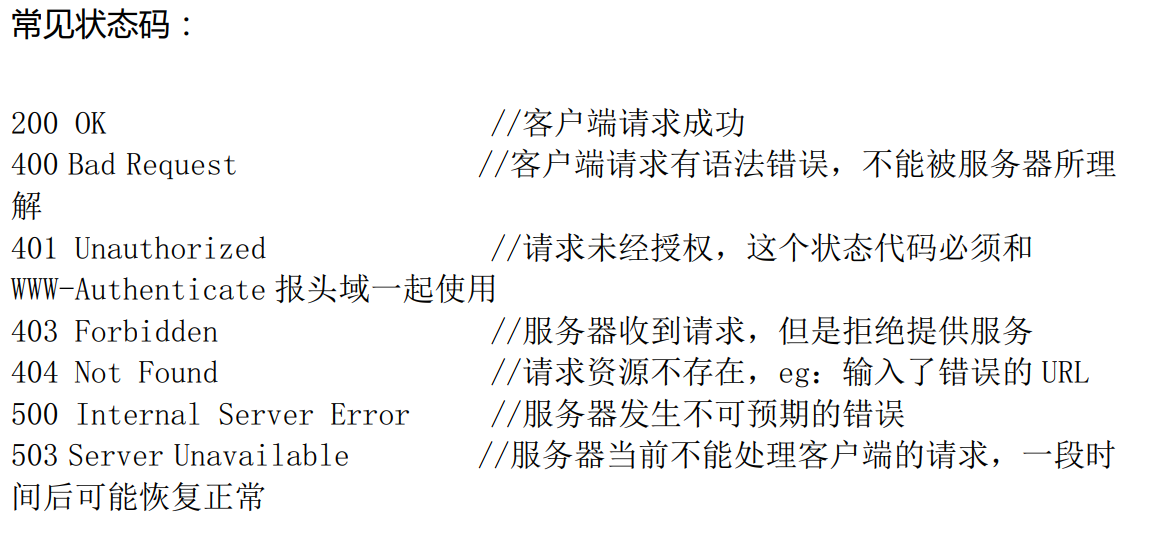
4. HTTP 协议之请求格式  
请求格式的结构：

  
 请求头： 请求方式、 请求的地址和 HTTP 协议版本  
 请求行： 消息报头， 一般用来说明客户端要使用的一些附加信息  
 空行： 位于请求行和请求数据之间， 空行是必须的。  
 请求数据： 非必须。  
注意： 一张网页的内容是极其丰富的， 浏览器会遵循HTTP 请求的格式将有效数据发送给服务器。

5. HTTP 协议之请求方式  
http1.0 定义了三种请求方法： get , post和head方法。  
http1.1 新增了五种请求方法： options,put,delete,trace和connect方法。

  
get 和 post 请求方式的区别:  
get 请求方式：  
 请求数据会以？的形式隔开拼接在请求头中，不安全，没有请求实体部分。  
http协议虽然没有规定请求数据的大小，但是浏览器对 URL 的长度是有限制的，所以 get 请求不能携带大量的数据。  
post 请求方式：  
 请求数据在请求实体中进行发送，在URL中看不到具体的请求数据，安全。 适合数据量大的数据发送。

6. HTTP 协议之响应  
常见响应状态码含义：  
 其由三个十进制数字组成，第一个十进制数字定义了状态码的类型，后两个数字没有分类的作用。http状态码共分为5种类型：

1. Tomcat服务器介绍和使用

问题：

学习了 java 编程之后， java 代码的一个很重要的作用就是进行数据的  
处理， 但是目前来说我们运行编写的代码， 只有一次性， 也就是运行完毕后， 如果需要再次运行则需要再次手动启动代码的执行。 但是我们无法提前用户会何时发送请求， 也就无法决定我们编写的 java 代码应该什么时候启动运行。 而且手动运行也变得不现实。

解决：

那么根据 java 的网络编内容， 我们是不是可以使用代码编写一个容器  
(对象) 呢？该容器可以根据用户的请求来启动并运行我们编写的数据逻辑代  
码。 答案是可以的。

实现：  
 服务器

解释：  
 所谓服务器其实就是代码编写的一个可以根据用户请求实时的调用执行  
对应的逻辑代码的一个容器。 在普通用户看来就是一个安装程序。 我们只需要将服务器在操作系统上进行安装， 并将我们事先编写好的逻辑处理代码根据规则放到服务器的指定位置， 启动服务器， 那么服务器就自动的会根据接收到请求调用并执行对象的逻辑代码进行处理。

1. Tomcat 服务的下载和安装  
注意： 尽量不要解压在中文目录中  
目录结构介绍：  
 \bin 存放启动和关闭 Tomcat 的可执行文件  
 \conf 存放 Tomcat 的配置文件  
 \lib 存放库文件  
 \logs 存放日志文件  
 \temp 存放临时文件  
 \webapps 存放 web 应用  
 \work 存放 JSP 转换后的 Servlet 文件

校验安装：  
 打开bin目录，然后双击执行startup.bat文件，打开浏览器在地址栏中输入 localhost：8080/如果出现tomcat广告页安装成功

2.第一个 Web 程序  
 1、 打开 Myeclipse 并创建工作空间  
 2、 点击 file 选择—>new—>web project—>输入项目名， 点击 finish  
 3、 在 src 下创建包 com.bjsxt.servlet.  
 4、 在包下创建一个普通 java 类 MyServlet，并继承 HttpServlet  
 5、 在 MyServlet 类中覆写 service 方法。

1. **package** com.bjsxt.servlet;
2. **import** java.io.IOException;
3. **import** javax.servlet.ServletException;
4. **import** javax.servlet.http.HttpServlet;
5. **import** javax.servlet.http.HttpServletRequest;
6. **import** javax.servlet.http.HttpServletResponse;
7. **public** **class** MyServlet **extends** HttpServlet{
8. @Override
9. **protected** **void** service(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {
10. resp.getWriter().write("this is my first servlet.");
11. System.out.println("this is my first servlet.");
12. }
13. }

6、在 webRoot 下的 WEB-INF 下找到 web.xml文件并配置， 如下。

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee"
4. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5. http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID"
6. version="2.5"**>**
7. <!--配置Servlet -->
8. <!--配置servlet类路径 -->
9. **<servlet>**
10. **<servlet-name>**my**</servlet-name>**
11. **<servlet-class>**com.bjsxt.servlet.MyServlet**</servlet-class>**
12. **</servlet>**
13. <!--配置访问方式 -->
14. **<servlet-mapping>**
15. **<servlet-name>**my**</servlet-name>**
16. **<url-pattern>**/my**</url-pattern>**
17. **</servlet-mapping>**
18. **</web-app>**

7、 打开 tomcat 的安装目录，在webapps目录下新建文件夹，文件夹名为 project，并将项目源码webRoot下的内容全部复制到该文件夹下。  
 8、 启动 tomcat 服务器并在地址栏中输入：localhost:8080/project/my

总结:  
 MyEclipse 是开发工具， 通过此工具便于程序员的代码编写。真正运行的代码不是 MyEclipse 中编写的代码， 而是 tomcat 服务器中部署好的代码。Tomcat 会根据请求自动调用对应的代码进行请求处理。

3. Servlet 介绍  
问题：  
 服务器在接收到浏览器的请求后，会自动调用对应的逻辑代码进行请求处理。 但是逻辑代码是由程序员编写并放到服务器中， 那么服务器怎么知道该怎  
么调用并调用哪个类和哪个方法来进行请求处理。

解决：  
 程序员在编写代码的时候如果能够按照服务器能够识别的规则进行编写，浏览器按照指定的规则进行发送请求，那么服务器就可以调用并执行响应的逻辑  
代码进行请求处理了。

实现：  
 Servlet技术

概念：  
 狭义的 Servlet 是指 Java 语言实现的一个接口，广义的 Servlet 是指任何实现了这个 Servlet 接口的类，一般情况下，人们将 Servlet 理解为后者。Servlet 运行于支持 Java 的应用服务器中。 从原理上讲， Servlet 可以响应任何类型的请求， 但绝大多数情况下 Servlet 只用来扩展基于 HTTP 协议的 Web 服务器。

特点：  
 运行在支持 java的应用服务器上Servlet 的实现遵循了服务器能够识别的规则，也就是服务器会自动的根据请求调用对应的 servlet 进行请求处理。简单方便，可移植性强

使用：  
 1、 创建普通的 java 类并继承 HttpServlet  
 2、 覆写 service 方法  
 3、 在 service 方法中书写逻辑代码即可  
 4、 在 webRoot 下的 WEB-INF 文件夹下的 web.xml

文件中配置 servlet：  
 url：http://localhost:8080/project/my2  
组成:  
 服务器地址:端口号/虚拟项目名/servlet的别名  
 URI：虚拟项目名 /servlet的别名  
浏览器发送请求到服务器，服务器根据请求URL地址中的URI信息在webapps目录下找到对应的项目文件夹，然后在web.xml中检索对应的servlet，找到后调用并执行Servlet。  
  
  
三、Servlet的使用

1. Servlet 的生命周期  
 从第一次调用， 到服务器关闭。 如果在 web.xml 中配置了 load-on-startup 则是从服务器开启到服务器关闭。  
 代码示例：

package com.bjsxt.servlet;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

 /\*\*

 \* Servlet的生命周期：

 \*      1、从第一次调用到服务器关闭。

 \*      2、如果在web.xml中配置了load-on-startup，生命周期为从服务器启动到服务器关闭

 \* 注意：

 \*      init方法是对Servlet进行初始化的一个方法，会在Servlet第一次加载进行存储时执行

 \*      destory方法是在servlet被销毁时执行，也就服务器关闭时。 \*

 \*/

public class ServletLife extends HttpServlet {

    //初始化方法，在servlet第一次加载内容的时候被调用

    @Override

    public void init() throws ServletException {

        System.out.println("servlet初始化完成");

    }

    //service方法，真正处理请求的方法

    @Override

    protected void service(HttpServletRequest req, HttpServletResponse resp)

            throws ServletException, IOException {

        resp.getWriter().write("servlet life");

        System.out.println("servlet life");

    }

    @Override

    public void destroy() {

        System.out.println("我被销毁了...");

    }

}

Web.xml 配置:

1. **<servlet>**
2. **<description>**This is the description of my J2EE component**</description>**
3. **<display-name>**This is the display name of my J2EE component**</display-name>**
4. **<servlet-name>**ServletLife**</servlet-name>**
5. **<servlet-class>**com.bjsxt.servlet.ServletLife**</servlet-class>**
6. **<load-on-startup>**1**</load-on-startup>**
7. **</servlet>**
8. **<servlet-mapping>**
9. **<servlet-name>**my**</servlet-name>**
10. **<url-pattern>**/my**</url-pattern>**
11. **</servlet-mapping>**
12. **<servlet-mapping>**
13. **<servlet-name>**ServletLife**</servlet-name>**
14. **<url-pattern>**/life**</url-pattern>**
15. **</servlet-mapping>**

2.Service和doGet和doPost方法的区别  
Service 方法:  
 不管是 get 方式还是 post 方式的请求，如果Servlet类中有servic方法，则优先调用 Service 方法。  
doGet 方法:  
 在没有service方法的情况下如果是get方式的请求所调用的处理请求的方法  
doPost 方法:  
 在没有service方法的情况下若是post方式的请求所调用的处理请求的方法  
Servlet 的常见错误总结：  
 \* Servlet的常见错误：  
 \* 404错误:资源未找到  
 \* 原因一： 在请求地址中的servlet的别名书写错误。  
 \* 原因二： 虚拟项目名称拼写错误  
 \* 500错误： 内部服务器错误  
 \* 错误一：  
 \* java.lang.ClassNotFoundException: com.bjsxt.servlet.ServletMothod  
 \* 解决：  
 \* 在web.xml中校验servlet类的全限定路径是否拼写错误。  
 \* 错误二：  
 \* 因为service方法体的代码执行错误导致  
 \* 解决：  
 \* 根据错误提示对service方法体中的代码进行错误更改。  
 \* 405错误:请求方式不支持  
 \* 原因：  
 \* 请求方式和servlet中的方法不匹配所造成的。  
 \* 解决：  
 \* 尽量使用 service 方法进行请求处理， 并且不要再 service 方法中调用父类的service。

package com.bjsxt.servlet;

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 ServletMethod extends HttpServlet {

    @Override

    protected void service(HttpServletRequest req, HttpServletResponse resp)

            throws ServletException, IOException {

        int i=5/0;

        System.out.println("我是service");

        super.service(req, resp);

    }

    @Override

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)

            throws ServletException, IOException {

        System.out.println("我是doGet方法");

    }

    @Override

    protected void doPost(HttpServletRequest req, HttpServletResponse resp)

            throws ServletException, IOException {

        System.out.println("我是doPost方法");

    }

}

3.Request 对象:  
问题：  
 浏览器发起请求到服务器，会遵循HTTP协议将请求数据发送给服务器。  
那么服务器接受到请求的数据改怎么存储呢?不但要存，而且要保证完成性。

解决：  
 使用对象进行存储， 服务器每接受一个请求， 就创建一个对象专门的存储此次请求的请求数据。

实现：  
 request 对象

解释：  
 服务器接收到浏览器的请求后，会创建一个Request对象，对象中存储了此次请求相关的请求数据。 服务器在调用 Servlet 时会将创建的Request对象作为实参传递给Servlet的方法，比如：service 方法。

使用：  
 获取请求头数据  
 获取请求行数据  
 获取用户数据

package com.bjsxt.servlet;

import java.io.IOException;

import java.util.Enumeration;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

 \* request对象学习：

 \*      作用：request对象中封存了当前请求的所有请求信息

 \*      使用：

 \*          获取请求头数据

 \*                  req.getMethod();//获取请求方式

 \*                  req.getRequestURL();//获取请求URL信息

 \*                  req.getRequestURI();//获取请求URI信息

 \*                  req.getScheme();//获取协议

 \*          获取请求行数据

 \*                  req.getHeader("键名");//返回指定的请求头信息

 \*                  req.getHeaderNames();//返回请求头的键名的枚举集合

 \*          获取用户数据

 \*                  req.getParameter("键名");//返回指定的用户数据

 \*                  req.getParameterValues("键名");//返回同键不同值的请求数据(多选)，返回的数组。

 \*                  req.getParameterNames()//返回所有用户请求数据的枚举集合

 \*          注意：

 \*              如果要获取的请求数据不存在，不会报错，返回null。

 \*

 \*      注意：

 \*          request对象由tomcat服务器创建，并作为实参传递给处理请求的servlet的service方法。

 \* @author MyPC

 \*

 \*/

public class RequestServlet extends HttpServlet {

    @Override

    protected void service(HttpServletRequest req, HttpServletResponse resp)

            throws ServletException, IOException {

        //获取请求头数据

                //获取请求方式

                    String method = req.getMethod();

                    System.out.println(method);

                //获取请求URL

                    StringBuffer url=req.getRequestURL();

                    System.out.println(url);

                    //获取URI

                    String uri=req.getRequestURI();

                    System.out.println(uri);

                //获取协议

                    String h=req.getScheme();

                    System.out.println(h);

        //获取请求行数据

                    //获取指定的请求行信息

                    String value=req.getHeader("aaa");

                    System.out.println(value);

                    //获取所有的请求行的键的枚举

                    Enumeration e = req.getHeaderNames();

                    while(e.hasMoreElements()){

                        String name=(String) e.nextElement();

                        String value2=req.getHeader(name);

                        System.out.println(name+":"+value2);

                    }

        //获取用户数据

                    String name=req.getParameter("uname");

                    String pwd=req.getParameter("pwd");

                    System.out.println(name+":"+pwd);

                    //String fav=req.getParameter("fav");

                    String[] favs=req.getParameterValues("fav");

                    if(favs!=null){

                        for(String fav:favs){

                            System.out.println(fav);

                        }

                    }

                    //获取所有的用户请求数据的键的枚举集合---req.getParameterNames()

    }

}

4.Response 对象：  
问题：  
 在使用 Request 对象获取了请求数据并进行处理后，处理的结果如何显  
示到浏览器中呢？

解决：  
 使用 Response对象

解释：  
 服务器在调用指定的 Servlet 进行请求处理的时候，会给 Servlet 的方法传递两个实参request和response。其中request中封存了请求相关的请求数据， 而 response 则是用来进行响应的一个对象。

使用：  
 设置响应头  
 设置响应编码格式  
 设置响应实体  
 请求乱码问题解决:  
 使用 String 进行重新编码：  
 uname=new String(uname.getBytes("iso8859-1"),"utf-8");

package com.bjsxt.servlet;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

 \* Respone对象学习：

 \*      作用：

 \*          用来响应数据到浏览器的一个对象

 \*      使用：

 \*          设置响应头

 \*              setHeader(String name,String value);//在响应头中添加响应信息，但是同键会覆盖

 \*              addHeader(String name,String value);//在响应头中添加响应信息，但是不会覆盖。

 \*          设置响应状态

 \*              sendError(int num,String msg);//自定义响应状态码。

 \*          设置响应实体

 \*              resp.getWrite().write(String str);//响应具体的数据给浏览器

 \*      设置响应编码格式：

 \*              resp.setContentType("text/html;charset=utf-8");

 \*      总结：

 \*          service请求处理代码流程：

 \*              设置响应编码格式

 \*              获取请求数据

 \*              处理请求数据

 \*                  数据库操作（MVC思想）

 \*              响应处理结果

 \* @author MyPC

 \*

 \*/

public class ResponseServlet extends HttpServlet {

    @Override

    protected void service(HttpServletRequest req, HttpServletResponse resp)

            throws ServletException, IOException {

            //获取请求信息

                //获取请求头

                //获取请求行

                //获取用户数据

            //处理请求

            //响应处理结果

                //设置响应头

                resp.setHeader("mouse", "two fly birds");

                resp.setHeader("mouse", "bjsxt");

                resp.addHeader("key", "thinkpad");

                resp.addHeader("key", "wollo");

                //设置响应编码格式

                    //resp.setHeader("content-type", "text/html;charset=utf-8");

                    //resp.setContentType("text/plain;charset=utf-8");

                    //resp.setContentType("text/xml;charset=utf-8");

                    resp.setContentType("text/html;charset=utf-8");

                //设置响应状态吗

                    //resp.sendError(404, "this Method is not supported");

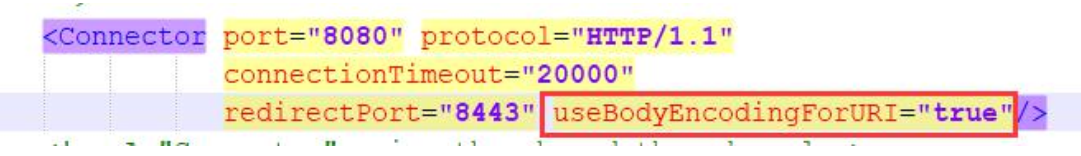
                //设置响应实体

                    resp.getWriter().write("<b>今天天气真好，适合学习</b>");

    }

}

get 方式请求：  
 在 service 方法中使用: req.setCharacterEncoding(“utf-8”)；  
 在 tomcat 服务器目录下的conf文件下找到 server.xml文件，打开进  
行如下配置:



post 方式请求：  
 在service方法中使用: req.setCharacterEncoding(“utf-8”)；

流程总结:  
 Servlet 的使用流程:  
 设置请求编码格式  
 设置响应编码格式  
 获取请求信息  
 处理请求信息  
 响应处理结果

数据流转流程:  
 浏览器------>服务器------->数据库  
 浏览器<------服务器<-------数据库

5.请求转发

问题：  
 服务器在接收到浏览器的请求后，仅仅使用一个Servlet进行请求处理，会造成不同的Servlet 逻辑代码冗余，Servlet的职责不明确。

解决：  
 使用请求转发。

req.getRequestDispatchar(“要转发的地址”).forward(req,resp);

特点：  
 一次请求  
 地址栏信息不改变。

6.request对象作用域  
问题：

使用请求转发后，不同的 Servlet 之间怎么进行数据的共享呢？或者说数据怎么从一个 servlet 流转给另外一个 Servlet 呢？

解决：

使用request对象的作用域

使用：  
 request.setAttribute(object name,Object value);  
 request.getAttribute(Object obj)

作用：

解决了一次请求内的不同 Servlet 的数据(请求数据+其他数据)共享问题。

作用域：基于请求转发，一次请求中的所有Servlet共享。

注意：  
 使用 Request 对象进行数据流转， 数据只在一次请求内有效。

特点：  
 服务器创建  
 每次请求都会创建  
 生命周期一次请求

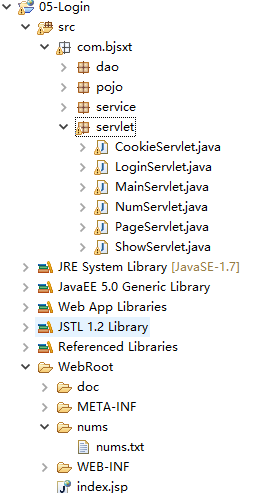
7.重定向

问题：  
 如果当前的请求， Servlet 无法进行处理怎么办?  
 如果使用请求转发， 造成表单数据重复提交怎么办？

解决：  
 使用重定向

使用：  
 response.sendRedirect(“路径”).  
 本地路径为： uri

网络路径为： 定向资源的 URL 信息

特点：  
 两次请求，两个request对象  
 浏览器地址栏信息改变  
 避免表单重复提交  
  
。

### **servlet：**

　　CookieServlet：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | package com.bjsxt.servlet;    import java.io.IOException;    import javax.servlet.ServletException;  import javax.servlet.http.Cookie;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    import com.bjsxt.pojo.User;  import com.bjsxt.service.LoginService;  import com.bjsxt.service.impl.LoginServiceImpl;  /\*\*   \* Cookie信息校验   \*      判断请求中是否携带正确的Cookie信息   \*      如果有则校验Cookie信息是否正确   \*          如果校验正确则直接响应主页面给用户   \*          如果校验不正确则响应登录页面给用户   \*      没有则请求转发给登录页面   \* @author MyPC   \*/  public class CookieServlet extends HttpServlet {      @Override      protected void service(HttpServletRequest req, HttpServletResponse resp)              throws ServletException, IOException {          //设置请求编码格式              req.setCharacterEncoding("utf-8");          //设置响应编码格式              resp.setContentType("text/html;charset=utf-8");          //获取请求信息              //获取Cookie信息              Cookie[] cks=req.getCookies();          //处理请求信息              if(cks!=null){                  //遍历Cookie信息                      String uid="";                      for(Cookie c:cks){                          if("uid".equals(c.getName())){                              uid=c.getValue();                          }                      }                  //校验UID是否存在                      if("".equals(uid)){                          //请求转发                          req.getRequestDispatcher("page").forward(req, resp);                          return;                      }else{                          //校验UID用户信息                              //获取业务层对象                              LoginService ls=new LoginServiceImpl();                              User u=ls.checkUidService(uid);                              if(u!=null){                                  //将用户数据存储到session对象中                                  req.getSession().setAttribute("user",u);                                  //网页计数器自增                                  int nums=(int) this.getServletContext().getAttribute("nums");                                  nums+=1;                         this.getServletContext().setAttribute("nums", nums);                                  //重定向                                  resp.sendRedirect("/login/main");                                  return;                              }else{                                  //请求转发                                  req.getRequestDispatcher("page").forward(req, resp);                                  return;                              }                      }                }else{                  //响应处理结果                      //请求转发                      req.getRequestDispatcher("page").forward(req, resp);                      return;              }      }  } |

　　LoginServlet：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130 | package com.bjsxt.servlet;    import java.io.IOException;    import javax.servlet.ServletContext;  import javax.servlet.ServletException;  import javax.servlet.http.Cookie;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import javax.servlet.http.HttpSession;    import com.bjsxt.pojo.User;  import com.bjsxt.service.LoginService;  import com.bjsxt.service.impl.LoginServiceImpl;  /\*\*   \* 请求中文乱码解决:   \*  1、使用String进行数据重新编码   \*          uname=new String(uname.getBytes("iso8859-1"),"utf-8");   \*  2、使用公共配置   \*      get方式：   \*          步骤一：req.setCharacterEncoding("utf-8");   \*          步骤二：   \*              在tomcat的目录下的conf目录中修改server.xml文件：在Connector标签中增加属性 useBodyEncodingForURI="true"   \*      post方式：   \*          req.setCharacterEncoding("utf-8");   \* Servlet流程总结:   \*      浏览器发起请求到服务器(请求)   \*      服务器接受浏览器的请求，进行解析，创建request对象存储请求数据   \*      服务器调用对应的servlet进行请求处理，并将request对象作为实参传递给servlet的方法   \*      servlet的方法执行进行请求处理   \*              //设置请求编码格式   \*              //设置响应编码格式   \*              //获取请求信息   \*              //处理请求信息   \*                  //创建业务层对象   \*                  //调用业务层对象的方法   \*              //响应处理结果   \*请求转发学习：   \*      作用:实现多个servlet联动操作处理请求，这样避免代码冗余，让servlet的职责更加明确。   \*      使用：   \*              req.getRequestDispatcher("要转发的地址").forward(req, resp);   \*              地址：相对路径，直接书写servlet的别名即可。   \*      特点:   \*          一次请求，浏览器地址栏信息不改变。   \*      注意：   \*          请求转发后直接return结束即可。   \*reuqet作用域：   \*      解决了一次请求内的servlet的数据共享问题   \*重定向：   \*      解决了表单重复提交的问题，以及当前servlet无法处理的请求的问题。   \*      使用:   \*          resp.sendRedirect(String uri);   \*      示例:   \*          resp.sendRedirect("/login/main");   \*      特点：   \*          两次请求，两个request对象。   \*          浏览器地址栏信息改变   \*      时机：   \*          如果请求中有表单数据，而数据又比较重要，不能重复提交，建议使用重定向。   \*          如果请求被Servlet接收后，无法进行处理，建议使用重定向定位到可以处理的资源。   \*解决主页面用户名显示为null的问题：   \*      原因：   \*          因为在用户登录成功后使用重定向显示主页面，两次请求，而用户的信息   \*          在第一次请求中，第二次请求中没有用户数据，所以显示为null   \*      解决:   \*          使用session   \*使用ServletContext对象完成网页计数器   \*      在用户登录校验中创建计数器并自增，然后存储到ServletContext对象中   \*      在主页面里取出计数器数据显示给用户即可。   \*   \* @author MyPC   \*/  public class LoginServlet extends HttpServlet {      @Override      protected void service(HttpServletRequest req, HttpServletResponse resp)              throws ServletException, IOException {          //设置请求编码格式:              req.setCharacterEncoding("utf-8");          //设置响应编码格式              resp.setContentType("text/html;charset=utf-8");          //获取请求信息              String uname=req.getParameter("uname");              //uname=new String(uname.getBytes("iso8859-1"),"utf-8");//使用String进行数据重新编码              String pwd=req.getParameter("pwd");              System.out.println(uname+":"+pwd);          //处理请求信息              //获取业务层对象              LoginService ls=new LoginServiceImpl();              User u=ls.checkLoginService(uname, pwd);              System.out.println(u);          //响应处理结果          if(u!=null){                  //创建Cookie信息实现三天免登录。                      Cookie c=new Cookie("uid", u.getUid()+"");                      //设置Cookie的有效期                      c.setMaxAge(3\*24\*3600);                      c.setPath("/login/ck");                      //添加Cookie信息                      resp.addCookie(c);                  //请求转发                      //req.getRequestDispatcher("main").forward(req, resp);                  //将数据存储到session对象中                      HttpSession hs=req.getSession();                      hs.setAttribute("user",u);                  //创建网页计数器                      //获取计数数据                      ServletContext sc=this.getServletContext();                      if(sc.getAttribute("nums")!=null){                          int nums=Integer.parseInt((String) sc.getAttribute("nums"));                          //计数器自增                          nums+=1;                          //再次存储到ServletContext对象中                          sc.setAttribute("nums", nums);                      }else{                          sc.setAttribute("nums", 1);                      }                    //重定向                  resp.sendRedirect("/login/main");                  return;              }else{                  //使用request对象实现不同Servlet的数据流转                  req.setAttribute("str", "用户名或密码错误");                  //使用请求转发                  req.getRequestDispatcher("page").forward(req, resp);                  return;              }      }  } |

　　MainServlet：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | package com.bjsxt.servlet;    import java.io.IOException;    import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    import com.bjsxt.pojo.User;    public class MainServlet extends HttpServlet {      @Override      protected void service(HttpServletRequest req, HttpServletResponse resp)              throws ServletException, IOException {          //设置请求编码格式              req.setCharacterEncoding("utf-8");          //设置响应编码格式              resp.setContentType("text/html;charset=utf-8");          //获取请求信息              //获取session中的数据              User u=(User) req.getSession().getAttribute("user");              //获取网页浏览次数              int nums=(int) this.getServletContext().getAttribute("nums");          //处理请求信息          //响应处理结果              resp.getWriter().write("<html>");              resp.getWriter().write("<head>");              resp.getWriter().write("</head>");              resp.getWriter().write("<body>");              resp.getWriter().write("<h3>欢迎"+u.getUname()+"访问尚学堂管理系统</h3>");              resp.getWriter().write("当前网页浏览次数为:"+nums);              resp.getWriter().write("<hr>");              resp.getWriter().write("<form action='show' method='get'>");              resp.getWriter().write("<input type='submit' value='查看个人信息'>");              resp.getWriter().write("</form>");              resp.getWriter().write("</body>");              resp.getWriter().write("</html>");      }  } |

　　NumServlet：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83 | package com.bjsxt.servlet;    import java.io.BufferedReader;  import java.io.BufferedWriter;  import java.io.FileReader;  import java.io.FileWriter;  import java.io.IOException;    import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;    public class NumServlet extends HttpServlet {        //覆写init初始化方法，将数据读取到ServletContext对象中      @Override      public void init() throws ServletException {          //获取文件中的计数器数据              //获取文件路径              String path=this.getServletContext().getRealPath("/nums/nums.txt");              //声明流对象              FileReader fr=null;              BufferedReader br=null;              try {                  fr=new FileReader(path);                  br=new BufferedReader(fr);                  String nums=br.readLine();                  System.out.println(nums);                  this.getServletContext().setAttribute("nums", nums);              } catch (Exception e) {                  e.printStackTrace();              }finally{                  try {                      fr.close();                  } catch (IOException e) {                      // TODO Auto-generated catch block                      e.printStackTrace();                  }                  try {                      br.close();                  } catch (IOException e) {                      // TODO Auto-generated catch block                      e.printStackTrace();                  }              }      }        //覆写销毁方法，存储计数器到文件中      @Override      public void destroy() {          //获取网页计数器          int nums=(int) this.getServletContext().getAttribute("nums");          //获取文件路径          String path=this.getServletContext().getRealPath("/nums/nums.txt");          //声明流对象          BufferedWriter bw=null;          FileWriter fw=null;          try {              fw=new FileWriter(path);              bw=new BufferedWriter(fw);              bw.write(nums+"");              bw.flush();          } catch (Exception e) {              e.printStackTrace();          }finally{              try {                  fw.close();              } catch (IOException e) {                  // TODO Auto-generated catch block                  e.printStackTrace();              }              try {                  bw.close();              } catch (IOException e) {                  // TODO Auto-generated catch block                  e.printStackTrace();              }          }      }          } |

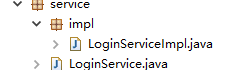
　　PageServlet：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | package com.bjsxt.servlet;    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 PageServlet extends HttpServlet {      @Override      protected void service(HttpServletRequest req, HttpServletResponse resp)              throws ServletException, IOException {          //设置响应编码格式              resp.setContentType("text/html;charset=utf-8");          //获取请求信息          //处理请求          //响应处理结果              //获取request作用域数据                  String str=(String) req.getAttribute("str")==null?"":(String) req.getAttribute("str");                  resp.getWriter().write("<html>");                  resp.getWriter().write("<head>");                  resp.getWriter().write("</head>");                  resp.getWriter().write("<body>");                  resp.getWriter().write("<font color='red' size='20px'>"+str+"</font>");                  resp.getWriter().write("<form action='login' method='get'>");                  resp.getWriter().write("用户名:<input type='text' name='uname' value=''/><br/>");                  resp.getWriter().write("密码:<input type='password' name='pwd' value=''/><br/>");                  resp.getWriter().write("<input type='submit'  value='登录'/><br/>");                  resp.getWriter().write("</form>");                  resp.getWriter().write("</body>");                  resp.getWriter().write("</html>");              }      } |

　　ShowServlet：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | package com.bjsxt.servlet;    import java.io.IOException;    import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import javax.servlet.http.HttpSession;    import com.bjsxt.pojo.User;    public class ShowServlet extends HttpServlet {      @Override      protected void service(HttpServletRequest req, HttpServletResponse resp)              throws ServletException, IOException {          //设置请求编码格式              req.setCharacterEncoding("utf-8");          //设置响应编码格式              resp.setContentType("text/html;charset=utf-8");          //获取请求信息              HttpSession hs=req.getSession();              if(hs.getAttribute("user")==null){                  //重定向                  resp.sendRedirect("/login/ck");                  return;              }              User u=(User) hs.getAttribute("user");          //处理请求信息          //响应处理结果              resp.getWriter().write("<html>");              resp.getWriter().write("<head>");              resp.getWriter().write("</head>");              resp.getWriter().write("<body>");              resp.getWriter().write("<table border='1px'>");              resp.getWriter().write("<tr>");              resp.getWriter().write("<td>用户名</td>");              resp.getWriter().write("<td>"+u.getUname()+"</td>");              resp.getWriter().write("</tr>");              resp.getWriter().write("<tr>");              resp.getWriter().write("<td>密码</td>");              resp.getWriter().write("<td>"+u.getPwd()+"</td>");              resp.getWriter().write("</tr>");              resp.getWriter().write("</table>");              resp.getWriter().write("</body>");              resp.getWriter().write("</html>");      }  } |

### **service**



 　　LoginServiceImpl：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | package com.bjsxt.service.impl;    import com.bjsxt.dao.LoginDao;  import com.bjsxt.dao.impl.LoginDaoImpl;  import com.bjsxt.pojo.User;  import com.bjsxt.service.LoginService;    public class LoginServiceImpl implements LoginService{      //创建Dao层过度向      LoginDao ld=new LoginDaoImpl();      //校验用户登录信息      @Override      public User checkLoginService(String uname, String pwd) {            return ld.checkLoginDao(uname, pwd);      }      //校验Cookie信息      @Override      public User checkUidService(String uid) {          return ld.checkUidDao(uid);      }    } |

　　LoginService：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | package com.bjsxt.service;    import com.bjsxt.pojo.User;    public interface LoginService {      //校验用户登录信息      User checkLoginService(String uname,String pwd);      //校验用户Cookie信息      User checkUidService(String uid);  } |

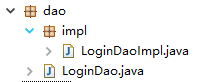
### **pojo：**

IMG_257

　　User：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75 | package com.bjsxt.pojo;    public class User {      private int uid;      private String uname;      private String pwd;      public int getUid() {          return uid;      }      public void setUid(int uid) {          this.uid = uid;      }      public String getUname() {          return uname;      }      public void setUname(String uname) {          this.uname = uname;      }      public String getPwd() {          return pwd;      }      public void setPwd(String pwd) {          this.pwd = pwd;      }      @Override      public String toString() {          return "User [uid=" + uid + ", uname=" + uname + ", pwd=" + pwd + "]";      }      @Override      public int hashCode() {          final int prime = 31;          int result = 1;          result = prime \* result + ((pwd == null) ? 0 : pwd.hashCode());          result = prime \* result + uid;          result = prime \* result + ((uname == null) ? 0 : uname.hashCode());          return result;      }      @Override      public boolean equals(Object obj) {          if (this == obj)              return true;          if (obj == null)              return false;          if (getClass() != obj.getClass())              return false;          User other = (User) obj;          if (pwd == null) {              if (other.pwd != null)                  return false;          } else if (!pwd.equals(other.pwd))              return false;          if (uid != other.uid)              return false;          if (uname == null) {              if (other.uname != null)                  return false;          } else if (!uname.equals(other.uname))              return false;          return true;      }      public User() {          super();          // TODO Auto-generated constructor stub      }      public User(int uid, String uname, String pwd) {          super();          this.uid = uid;          this.uname = uname;          this.pwd = pwd;      }          } |

### **dao：**



　　LoginDao：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | package com.bjsxt.dao;    import com.bjsxt.pojo.User;    public interface LoginDao {      //根据用户名和密码获取用户信息      User checkLoginDao(String uname,String pwd);      //根据uid获取用户信息      User checkUidDao(String uid);  } |

 　　LoginDaoImpl：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112 | package com.bjsxt.dao.impl;    import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import java.sql.SQLException;    import com.bjsxt.dao.LoginDao;  import com.bjsxt.pojo.User;    public class LoginDaoImpl implements LoginDao{        @Override      public User checkLoginDao(String uname, String pwd) {          //声明Jdbc对象          Connection conn=null;          PreparedStatement ps=null;          ResultSet rs=null;          //声明数据存储对象          User u=null;          try {              //加载驱动              Class.forName("com.mysql.jdbc.Driver");              //获取连接对象              conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/407","root","1234");              //创建Sql命令              String sql="select \* from t\_user where uname=? and pwd=?";              //创建Sql命令对象              ps=conn.prepareStatement(sql);              //给占位符赋值              ps.setString(1, uname);              ps.setString(2, pwd);              //执行              rs=ps.executeQuery();              //遍历执行结果              while(rs.next()){                  u=new User();                  u.setUid(rs.getInt("uid"));                  u.setUname(rs.getString("uname"));                  u.setPwd(rs.getString("pwd"));              }              //关闭资源          } catch (Exception e) {              e.printStackTrace();          }finally{              try {                  rs.close();              } catch (SQLException e) {                  // TODO Auto-generated catch block                  e.printStackTrace();              }              try {                  ps.close();              } catch (SQLException e) {                  // TODO Auto-generated catch block                  e.printStackTrace();              }              try {                  conn.close();              } catch (SQLException e) {                  // TODO Auto-generated catch block                  e.printStackTrace();              }          }          //返回          return u;      }      //根据UID获取用户信息      @Override      public User checkUidDao(String uid) {              //声明Jdbc对象                  Connection conn=null;                  PreparedStatement ps=null;                  ResultSet rs=null;                  //声明数据存储对象                  User u=null;                  try {                      //加载驱动                      Class.forName("com.mysql.jdbc.Driver");                      //获取连接对象                      conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/407","root","1234");                      //创建Sql命令                      String sql="select \* from t\_user where uid=?";                      //创建Sql命令对象                      ps=conn.prepareStatement(sql);                      //给占位符赋值                      ps.setString(1, uid);                      //执行                      rs=ps.executeQuery();                      //遍历执行结果                      while(rs.next()){                          u=new User();                          u.setUid(rs.getInt("uid"));                          u.setUname(rs.getString("uname"));                          u.setPwd(rs.getString("pwd"));                      }                      //关闭资源                  } catch (Exception e) {                      e.printStackTrace();                  }finally{                      try {                          rs.close();                      } catch (SQLException e) {                          // TODO Auto-generated catch block                          e.printStackTrace();                      }                      try {                          ps.close();                      } catch (SQLException e) {                          // TODO Auto-generated catch block                          e.printStackTrace();                      }                      try {                          conn.close();                      } catch (SQLException e) {                          // TODO Auto-generated catch block                          e.printStackTrace();                      }                  }                  //返回                  return u;      }  } |

四、Cookie学习

问题：  
 HTTP 协议是没有记忆功能的，一次请求结束后，相关数据会被销毁。如果第二次的请求需要使用相同的请求数据怎么办呢？难道是让用户再次请求书写吗?

解决：  
 使用 Cookie 技术

解释：  
 Cookie技术其实是浏览器端的数据存储技术，解决了不同请求需要使用相同的请求数据的问题。我们把请求需要共享的请求数据，存储在浏览器端，避免用户进行重复的书写请求数据。但是哪些数据需要使用 Cookie 技术存储起来是一个主观问题， 需要在后台进行响应的时候来告诉浏览器， 有些数据其他请求还会使用，需要存储起来。

特点：  
 浏览器端的数据存储技术  
 适合少量数据  
 键值对  
 不安全

使用：  
 Cookie 数据存储

临时存储：  
 不设置 cookie 信息的存储时间， 周期为一次会话，存储在浏览器内存中

定时存储：

设置存储时间， 周期为时间设置，存储在用户电脑中。

五、Session 学习  
问题：  
 Request 对象解决了一次请求内的不同 Servlet 的数据共享问题， 那么一个用户的不同请求的处理需要使用相同的数据怎么办呢?

解决：  
 使用 session 技术。

原理：  
 用户使用浏览器第一次向服务器发送请求，服务器在接受到请求后，调用对应的 Servlet 进行处理。 在处理过程中会给用户创建一个 session 对象，用来存储用户请求处理相关的公共数据，并将此 session 对象的 JSESSIONID 以 Cookie 的形式存储在浏览器中(临时存储，浏览器关闭即失效)。用户在发起第二次请求及后续请求时， 请求信息中会附带 JSESSIONID， 服务器在接收到请求后，调用对应的 Servlet 进行请求处理， 同时根据 JSESSIONID 返回其对应的 session 对象。

特点：

Session 技术是依赖 Cookie 技术的服务器端的数据存储技术。由服务器进行创建每个用户独立拥有一个 session默认存储时间为 30 分钟

作用：

解决了一个用户的不同请求的数据共享问题。

使用：  
 创建 Session 对象  
 存储数据到 session 对象  
 获取 session 对象  
 获取数据从 session 对象  
 如果获取 session 中不存在的数据返回 null。

注意：  
 只要不关闭浏览器， 并且 session 不失效的情况下， 同一个用  
户的任意请求在项目的任意Servlet中获取到的都是同一个session  
对象。

作用域：  
 一次会话

六、ServletContext 对象

问题：  
 Request 解决了一次请求内的数据共享问题， session 解决了用户不同请求的数据共享问题， 那么不同的用户的数据共享该怎么办呢？

解决：  
 使用 ServletContext 对象

作用：  
 解决了不同用户的数据共享问题

原理：  
 ServletContext 对象由服务器进行创建，一个项目只有一个对象。不管在项目的任意位置进行获取得到的都是同一个对象，那么不同用户发起的请求获取到的也就是同一个对象了， 该对象由用户共同拥有。

特点：  
 服务器进行创建  
 用户共享  
 一个项目只有一个

生命周期：  
 服务器启动到服务器关闭

作用域：  
 项目内

使用：  
 获取 ServletContext 对象  
 使用作用域进行共享数据流转  
 获取 web.xml 中的全局配置  
 获取 webroot 下项目资源流对象  
 获取 webroot 下资源绝对路径

案例： 网页浏览器次数统计， 详见源码

七、ServletConfig对象

问题：  
 使用 ServletContext 对象可以获取 web.xml 中的全局配置文件，在 web.xml 中每个Servlet也可以进行单独的配置，那么该怎么获取配置信息呢？

解决：  
 使用 ServletConfig 对象

作用：  
 ServletConfig 对象是Servlet的专属配置对象，每个Servlet都单独拥有一个 ServletConfig对象，用来获取web.xml中的配置信息。

使用：  
 获取 ServletConfig对象  
 获取 web.xml中servlet的配置信息

课程介绍：  
 每个 web 项目下都具备 web.xml 文件。通过前面的学习，我们对web.xml 文件有了一定了解，但是还不够，本节课就 web.xml 文件进行总结。同时对Tomcat下的server.xml 进行介绍。