Servlet规范

一、*Servlet*规范介绍

1. Servlet规范来自JAVAEE规范中的一种
2. 作用：
   - 在Servlet规范中，指定【动态资源文件】开发步骤
   - 在Servlet规范中，指定Http服务器调用动态资源文件规则
   - 在Servlet规范中，指定Http服务器管理动态资源文件实例对象规则

二、*Servlet*接口实现类

1. Servlet接口来自于Servlet规范下一个接口，这个接口存在Http服务器所提供的jar包
2. Tomcat服务器下lib文件有一个servlet-api.jar存放Servlet接口（javax.servlet.Servlet接口）
3. Servlet规范中任务，Http服务器能调用的【动态资源文件】必须是一个Servlet接口实现类

```
例子：
class Student{
    //不是动态资源文件，Tomcat无权调用
}

class Teacher implements Servlet{
    //合法动态资源文件，Tomcat有权利调用
    Servlet obj = new Teacher();
    obj.doGet()
}
```

三、*Servlet*接口实现类开发步骤

- 第一步：创建一个Java类继承于HttpServlet，使之成为一个Servlet接口实现类
- 第二部：重写HttpServlet父类两个方法。doGet() 或者 doPost()

  浏览器 ---> get请求方式 ---> oneServlet.doGet();

  浏览器 ---> post请求方式 ---> oneServlet.doPost();
- 第三部：将Servlet接口实现类相关信息【注册】到Tomcat服务器

  【网站】--->【src.main.webapp】--->【Web-INF】---> web.xml

```
<!--将Servlet接口实现类类路径地址交给Tomcat-->
<servlet>
    <servlet-name>mm</servlet-name> <!--声明一个变量存储servlet接口实现类类路径-->
    <servlet-class>com.xukang.controller.OneServlet</servlet-class><!--声明servlet接口实现类类路径-->
</servlet>

Tomcat String mm = "com.xukang.controller.OneServlet"
<!--为了降低用户访问Servlet接口实现类难度，需要设置简短请求别名-->
<servlet-mapping>
    <servlet-name>mm</servlet-name>
    <url-pattern>/one</url-pattern> <!--设置简短请求别名,别名在书写时必须以"/"为开头-->
</servlet-mapping>

<!--
如果现在浏览器向Tomcat索要OneServlet时地址
http://localhost:8080/myWeb/one
-->
```

```
package com.xukang.controller;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
/**
 * 子类继承于父类，父类实现于A接口
```

```
* 此时，子类也是A接口实现类
*
* 抽象类作用：  降低接口实现类对接口实现过程的难度
*           将接口中不需要使用抽象方法就给抽象类进行完成
*           这样接口实现类只需要对接口需要实现的方法进行重写
*
*  Servlet接口
*          init()
*          getServletConfig()
*          getServletInfo()
*          destory() ------ 四个方法对 Servlet接口实现类没用
*          service() ---- 有用
*
*   Tomcat根据Servlet规范调用Servlet接口实现类规则：
*          1. Tomcat有权创建Servlet接口实现类实例对象
*              Servlet oneServlet = new OneServlet();
*          2. Tomcat根据实例对象调用service()方法处理当前请求
*              oneServlet.service();//此时service()方法中this====oneServlet
*
*   OneServlet --> 根据父类实现的service()方法，去重写doGet(),doPost()等方法
*   继承于
*   (abstract)HttpServlet ---> 实现了service()方法
*   继承于
*   (abstract)GenericServlet --> 实现了 init(),destory(),getServletInfo(),getServletConfig() 方法
*   实现于
*   Servlet接口
*
*   通过父类决定在何种情况下调用子类中方法，设计模式中称为模板设计模式
*
*   HttpServlet:
*      service(){
*          if(请求方式 == Get){
*              this.doGet();
*          }else if(请求方式 == POST){
*              this.doPost();
*          }...
*      }
*   OneServlet: doGet()   doPost()
*
*   Servlet oneServlet = new OneServlet();
*   oneServlet.service();
*/
public class OneServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        System.out.println("OneServlet类针对浏览器发送GET请求方式处理");
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        System.out.println("OneServlet类针对浏览器发送POST请求方式处理");
    }
}
```

```
<!--web.xml-->
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <!--servlet接口实现类类路径地址交给Tomcat-->
    <servlet>
        <servlet-name>oneServlet</servlet-name>
        <servlet-class>com.xukang.controller.OneServlet</servlet-class>
    </servlet>

    <!--为servlet接口实现类提供简短别名-->
    <servlet-mapping>
        <servlet-name>oneServlet</servlet-name>
        <url-pattern>/one</url-pattern>
    </servlet-mapping>
</web-app>
```

```
// this 指向问题
public class Father {
    public void service(String method){
        if("get".equals(method)){
            /*
            子类如果重写了doGet()方法，那么this====子类实例对象
            如果没有重写doGet()方法，那么this====Father
            */
            this.doGet();
        }else if("post".equals(method)){
            this.doPost();
        }
```

```
    }
    protected void doPost() {
        System.out.println("Father doPost is running ...");
    }
    protected void doGet() {
        System.out.println("Father doGet is running ...");
    }
}

public class Son extends Father{
    @Override
    protected void doPost() {
        System.out.println("Son doPost is running ...");
    }
    @Override
    protected void doGet() {
        System.out.println("Son doGet is running ...");
    }
}

public class TestMain {
    public static void main(String[] args) {
        Father son = new Son();
        son.service("get");
        son.service("post");
    }
}

/*
输出:
    Son doGet is running ...
    Son doPost is running ...
*/
```

四、*Servlet*对象生命周期

1. 网站中所有的Servlet接口实现类的实例对象，只能由Http服务器负责额创建。开发人员不能手动创建Servlet接口实现类的实例对象。
2. 在默认的情况下，Http服务器接收到对于当前Servlet接口实现类第一次请求时，自动创建这个Servlet接口实现类的实例对象。
   在手动配置情况下，要求Http服务器在启动时自动创建某个Servlet接口实现类的实例对象

```
<servlet>
    <servlet-name>mm</servlet-name> <!--声明一个变量存储servlet接口实现类类路径-->
    <servlet-class>com.bjpowernode.controller.OneServlet</servlet-class>
    <load-on-startup>30<load-on-startup><!--填写一个大于0的整数即可，Tomcat启动时对应的实现类的实例对象被创建出来-->
</servlet>
```

3. 在Http服务器运行期间，一个Servlet接口实现类只能被创建出一个实例对象。
4. 在Http服务器关闭时刻，自动将网站中所有的Servlet对象进行销毁。

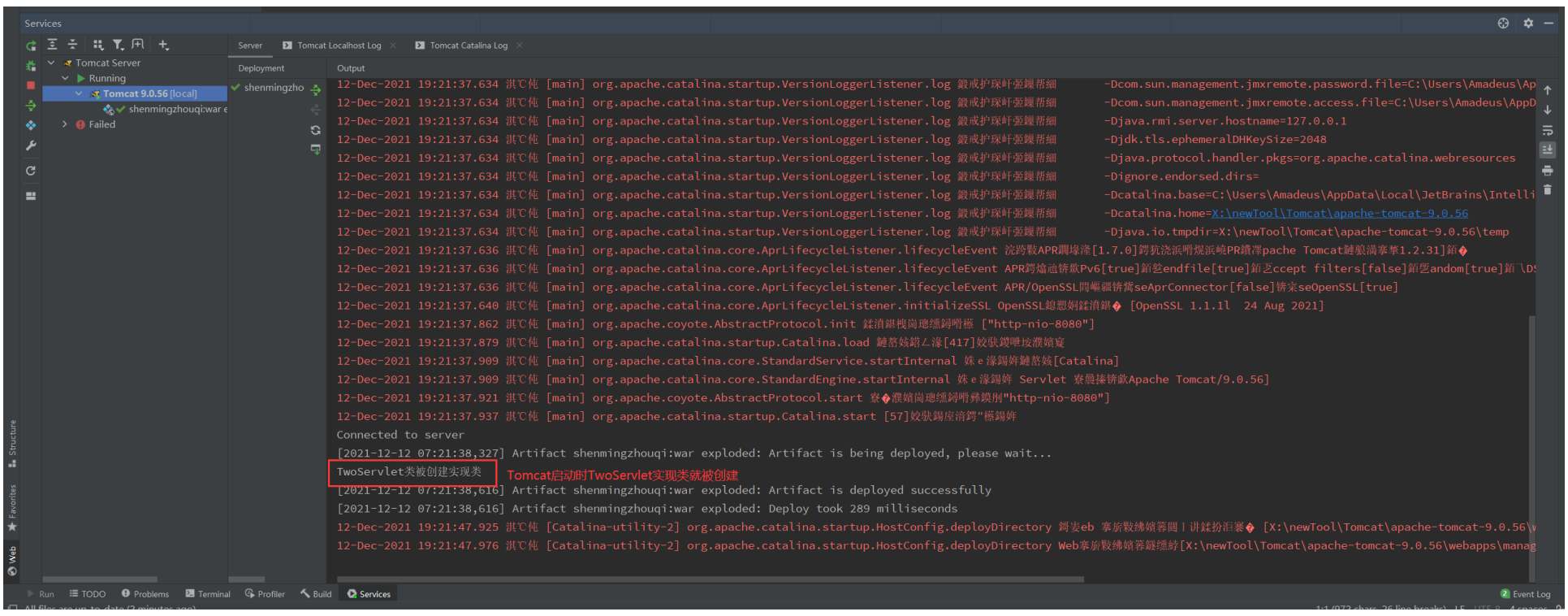
```
package com.example.controller;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
import java.io.IOException;

public class OneServlet extends HttpServlet {
    public OneServlet(){
        System.out.println("OneServlet类被创建实现类");
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("OneServlet doGet is run...");
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    }
}
```

```
package com.example.controller;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class TwoServlet extends HttpServlet {
    public TwoServlet(){
        System.out.println("TwoServlet类被创建实现类");
```

```java
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("TwoServlet doGet is run...");
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    }
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
    <servlet>
        <servlet-name>OneServlet</servlet-name>
        <servlet-class>com.example.controller.OneServlet</servlet-class>
    </servlet>

    <servlet>
        <servlet-name>TwoServlet</servlet-name>
        <servlet-class>com.example.controller.TwoServlet</servlet-class>
        <!--通知Tomcat在启动时负责创建TwoServlet实例对象-->
        <load-on-startup>9</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>OneServlet</servlet-name>
        <url-pattern>/one</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>TwoServlet</servlet-name>
        <url-pattern>/two</url-pattern>
    </servlet-mapping>
</web-app>
```

启动Tomcat后



五、*HttpServletResponse*接口

1. 介绍：
   - HttpServletResponse接口来自于Servlet规范中，在Tomcat中存在于servlet-api.jar
   - HttpServletResponse接口实现类由Http服务器负责提供
   - HttpServletResponse接口负责将 doGet()/doPost() 方法执行结果写入到【响应体】交给浏览器
   - 开发人员习惯于将HttpServletResponse接口修饰的对象称为【响应对象】
2. 主要功能
   - 将执行结果以二进制形式写入到【响应体】
   - 设置响应头中[content-type]属性值，从而控制浏览器使用对应编译器将响应体的 二进制数据 编译为【文字，图片，视频，命令】
   - 设置响应头中【location】属性，将一个请求地址赋值给location，从而控制浏览器向指定服务器发送请求

```java
package com.example.controller;

import javax.servlet.*;
import javax.servlet.http.*;
```

```java
import java.io.IOException;
import java.io.PrintWriter;

public class OneServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String result = "Hello World";//执行结果
        //=====响应对象将结果写入到响应体===== start
        //1.通过响应对象，向Tomcat索要输出流
        PrintWriter out = response.getWriter();
        //2.通过输出流，将执行结果以二进制形式写入到响应体
        out.write(result);
    }
    //doGet()执行完毕
    //Tomcat将响应包推送给浏览器
}
```

```java
package com.example.controller;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;

public class TwoServlet extends HttpServlet {
    /*
    * 问题描述：out.write(50); 浏览器收到数据是2，不是50
    *
    * 问题原因：
    *       out.write()方法可以将【字符】,【字符串】,【ASCII码】写入到响应体
    *       【ASCII码】
    *              a ------- 97
    *              2 ------- 50
    *       out.write()收到数字的时候，会把数字当做ASCII码去执行代码
    *
    * 问题解决方案：
    *       在实际的开发过程中，都是通过out.print()将真实数据写入到响应体.
    * */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        int money = 50;//执行结果

        PrintWriter out = response.getWriter();
        out.write(money);//2
        out.println();
        out.print(money);//50
    }
    //doGet()执行完毕
    //Tomcat将响应包推送给浏览器
}
```

```java
package com.example.controller;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;

public class ThreeServlet extends HttpServlet {
    /*
    * 问题描述：Java<br/>MySql<br/>HTML<br/>
    *         浏览器在接收到响应结果时，将<br/>作为文字内容在
    *         窗口中显示出来，没有将<br/>当做HTml标签命令来执行
    *
    * 问题原因：
    *       浏览器在接收响应包之后，根据【响应头中的 content-type】属性的
    *       值，来采用对应【编译器】对【响应体中二进制内容】进行编译处理
    *
    *       在默认情况下，content-type属性值为"text" content-type = "text";
    *       此时浏览器将会采用【文本编译器】对响应体二进制数据进行解析
    *
    * 解决方案：
    *       一定要在得到输出流之前，通过响应对象对响应头中content-type属性进行一次
    *       重新赋值用于指定浏览器采用正确编译器
    *
    * */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //<br/>为Html中换行命令
        //既有文字信息，又有HTML标签命令
        String result = "Java<br/>MySql<br/>HTML<br/>";
        String resule2 = "动漫<br/>电影<br/>TV<br/>";
        //PrintWriter out = response.getWriter();
        //out.print(result);//Java<br/>MySql<br/>HTML<br/>

        //设置响应头content-type
```

```
        response.setContentType("text/html;charset=utf-8");
        //向Tomcat索要输出流
        PrintWriter out2 = response.getWriter();
        //通过输出流将结果写入到响应体
        out2.print(result);
        out2.print(resule2);
    }
}
```

```
package com.example.controller;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class FourServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String result = "https://www.baidu.com";

        //通过响应对象，将地址赋值给响应头中location属性
        response.sendRedirect(result);//[响应头 location="https://www.baidu.com"]
    }
    /*
    *   浏览器在接收到响应包之后，如果发现响应头中存在location属性
    *   自动通过地址栏向location指定网站发送请求
    *
    *   sendRedirect方法远程控制浏览器请求行为【请求地址，请求方式，请求参数】
    * */
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
    <servlet>
        <servlet-name>OneServlet</servlet-name>
        <servlet-class>com.example.controller.OneServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>TwoServlet</servlet-name>
        <servlet-class>com.example.controller.TwoServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>ThreeServlet</servlet-name>
        <servlet-class>com.example.controller.ThreeServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>FourServlet</servlet-name>
        <servlet-class>com.example.controller.FourServlet</servlet-class>
    </servlet>
    <!--起别名-->
    <servlet-mapping>
        <servlet-name>OneServlet</servlet-name>
        <url-pattern>/one</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>TwoServlet</servlet-name>
        <url-pattern>/two</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>ThreeServlet</servlet-name>
        <url-pattern>/three</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>FourServlet</servlet-name>
        <url-pattern>/four</url-pattern>
    </servlet-mapping>
</web-app>
```

六、*HttpServletRequest*接口

1. 介绍
   - HttpServletRequest接口来自于Servlet规范中，在Tomcat中存在servlet-api.jar
   - HttpServletRequest接口实现类由Http服务器负责提供
   - HttpServletRequest接口负责在doGet()/doPost()方法运行时读取Http请求协议包中信息
   - 开发人员习惯于将HttpServletRequest接口修饰的对象称为【请求对象】
2. 作用
   - 可以读取Http请求协议包中【请求行】信息
   - 可以读取保存在Http请求协议包中【请求头】或则【请求体】中请求参数信息

- 可以代替浏览器向Http服务器申请资源文件调用

```java
/***
 * 读取请求行的信息
 */
public class OneServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //1.通过请求对象，读取【请求行】中【url】信息
        String url = request.getRequestURL().toString();
        //2.通过请求对象，读取【请求行】中【method】信息
        String method = request.getMethod();
        //3.通过请求对象，读取【请求行】中uri信息
        /*
         * URI：资源文件精准定位地址，在请求行并没有URI这个属性。
         *      实际上URL中截取一个字符串，这个字符串格式"/网站名/资源文件名"
         *      URI用于让Http服务器对被访问的资源文件进行定位
         */
        String uri = request.getRequestURI();

        System.out.println("URL "+url);//URL http://localhost:8080/myWeb/one
        System.out.println("method "+method);//method GET
        System.out.println("URI "+uri);//URI /myWeb/one
    }
}
```

```java
public class TwoServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //1.通过请求对象获得【请求头】中【所有请求参数名】
        //将所有请求参数名称保存到一个枚举对象进行返回
        Enumeration<String> parameterNames = request.getParameterNames();
        while(parameterNames.hasMoreElements()){
            String paramName = (String) parameterNames.nextElement();
            //2.通过请求对象读取指定的请求参数的值
            String value = request.getParameter(paramName);
            System.out.println("请求参数名 = " + paramName + ",请求参数值 = " + value);
            //请求参数名 = userName,请求参数值 = jack
            //请求参数名 = password,请求参数值 = 123
        }
    }
}
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <center>
        <a href="/myWeb/two?userName=jack&password=123">通过超链接访问TwoServlet别携带请求参数</a>
    </center>
</body>
</html>
```

```java
/*
 *  问题：
 *      以GET方式发送中文参数内容“老杨是正经人”时，得到正常结果
 *      以POST方式发送中文参数内容"老崔是个男人"，得到【乱码】“è?????????????·???”
 *
 *  原因：
 *      浏览器以GET方式发送请求,请求参数保存在【请求头】,在Http请求协议包到达Http服务器之后，第一件事就是进行解码
 *      请求头二进制内容由Tomcat负责解码，Tomcat9.0默认使用【utf-8】字符集，可以解释一切国家文字
 *
 *      浏览器以POST方式发送请求，请求参数保存在【请求体】,在Http请求协议包到达Http服务器之后，第一件事就是进行解码
 *      请求体二进制内容由当前请求对象（request）负责解码。request默认使用[ISO-8859-1]字符集，一个东欧语系字符集
 *      此时如果请求体参数内容是中文，将无法解码只能得到乱码
 *
 *  解决方案：
 *      在Post请求方式下，在读取请求体内容之前，应该通知请求对象使用utf-8字符集对请求体内容进行一次重新解码
 */
public class ThreeServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //通过请求对象，读取【请求头】参数信息
        String userName = request.getParameter("userName");
        System.out.println("从请求头得到参数值 " + userName);
    }

    @Override
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //通知请求对象，使用utf-8字符集对请求体二进制内容进行一次重写解码
        request.setCharacterEncoding("utf-8");
        //通过请求对象，读取【请求体】参数信息
        String value = request.getParameter("userName");
        System.out.println("从请求体得到参数值 "+value);
    }
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <center>
        <form action="/myWeb/three" method="get">
            请求参数:<input type="text" name="userName" /><br/>
            <input type="submit" value="get方式访问ThreeServlet">
        </form>

        <form action="/myWeb/three" method="post">
          请求参数:<input type="text" name="userName" /><br/>
          <input type="submit" value="post方式访问ThreeServlet">
        </form>
    </center>
</body>
</html>
```

七、请求对象与响应对象生命周期

1. 在Http服务器接收到浏览器发送的【Http请求协议包】之后，自动为当前的【Http请求协议包】生成一个【请求对象】和一个【响应对象】
2. 在Http服务器调用doGet/doPost方法时，负责将【请求对象】和【响应对象】作为实参传递到方法，确保doGet/doPost正确执行
3. 在Http服务器准备推送Http响应协议包之前，负责将本次请求关联的【请求对象】和【响应对象】销毁

    ***【请求对象】和【响应对象】生命周期贯穿一次请求的处理过程中

    ***【请求对象】和【响应对象】相当于用户在服务端的代言人

八、在线考试管理系统

1、准备工作

```
任务：在线考试管理系统----用户信息管理模块
子任务:
        用户信息注册
        用户信息查询
        用户信息删除
        用户信息更新
准备工作:
      1.创建用户信息表 Users.frm
      CREATE TABLE Users(
        userId int  primary key auto_increment, #用户编号
        userName varchar(50),    #用户名称
        password varchar(50),    #用户密码
        sex      char(1),        #用户性别 '男' 或者 '女'
        email    varchar(50)     #用户邮箱
      )
      auto_increment,自增序列     i++
      在插入时，如果不给定具体用户编号，此时根据auto_increment的值递增添加
      2.在src下 com.example.entity.Users 实体类
      3.在src下 com.example.util.JdbcUtil 工具类【复用】
      4.在web下WEB-INF下创建lib文件夹  存放mysql提供JDBC实现jar包
```

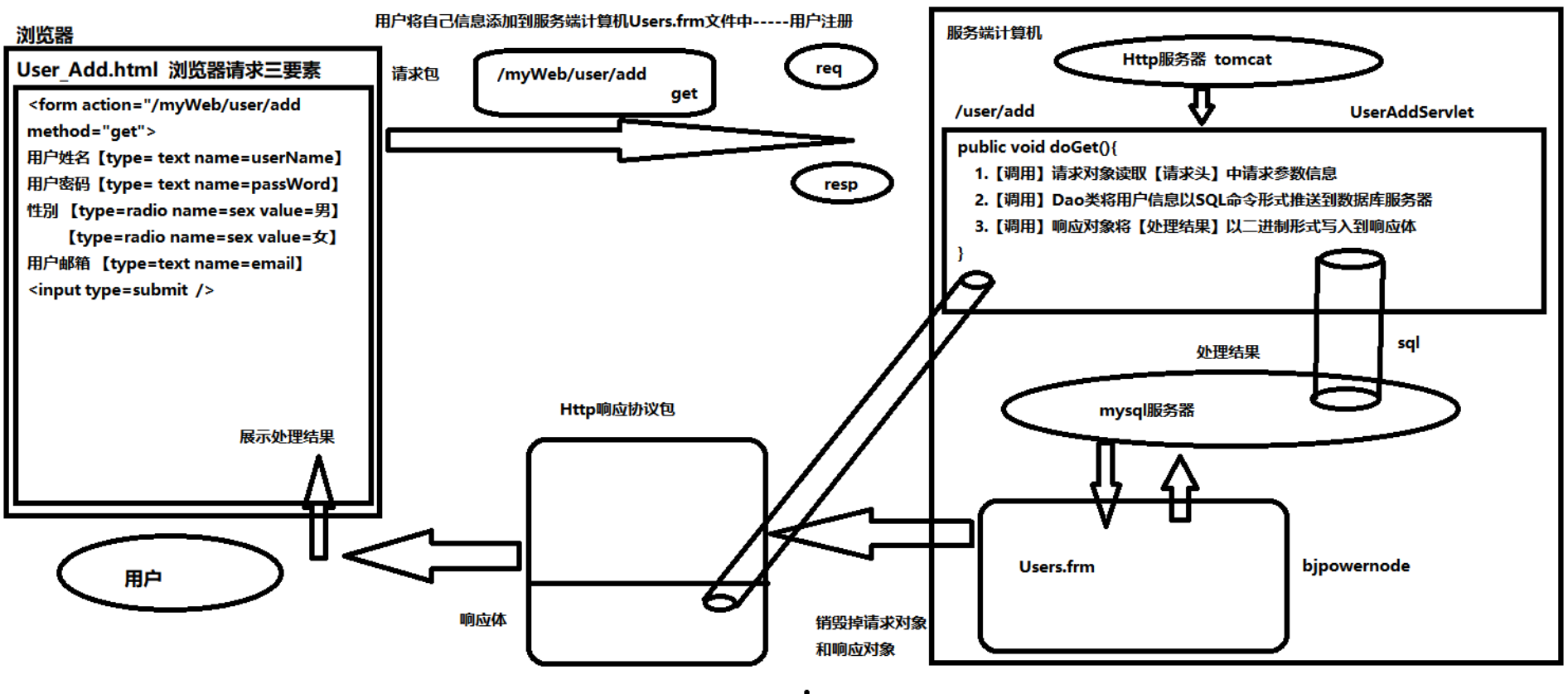
```
//Users类
package com.example.entity;
public class Users {
    private Integer userId;
    private String userName;
    private String passWord;
    private String sex;
    private String email;

    public Users() {
    }
    public Users(Integer userId, String userName, String passWord, String sex, String email) {
        this.userId = userId;
        this.userName = userName;
```

```
        this.passWord = passWord;
        this.sex = sex;
        this.email = email;
    }

    public Integer getUserId() {
        return userId;
    }
    public void setUserId(Integer userId) {
        this.userId = userId;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getPassWord() {
        return passWord;
    }
    public void setPassWord(String passWord) {
        this.passWord = passWord;
    }
    public String getSex() {
        return sex;
    }
    public void setSex(String sex) {
        this.sex = sex;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
}
```

```
package com.example.util;
import java.sql.*;
/**
 * JDBC工具类，简化JDBC编程
 */
public class JdbcUtil {
    /**
     * 工具类的构造方法都是私有的
     * 因为工具类当中的方法都是静态的，不需要new对象，直接采用类名调用
     */
    private JdbcUtil(){}

    // 静态代码块在类加载时执行，并且只执行一次
    // 注册驱动
    static {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    /**
     * 获取数据库连接对象
     * @return 连接对象
     * @throws SQLException
     */
    public static Connection getConnection() throws SQLException {
        Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/bjpowernode","root","111");
        return conn;
    }

    /**
     * 关闭资源
     * @param conn 连接对象
     * @param ps   数据库操作对象
     * @param rs   结果集
     */
    public static void close(Connection conn, Statement ps, ResultSet rs){
        if(rs != null){
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(ps != null){
            try {
                ps.close();
            } catch (SQLException e) {
```

```
                e.printStackTrace();
            }
        }
        if(conn != null){
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

2、用户信息注册流程图



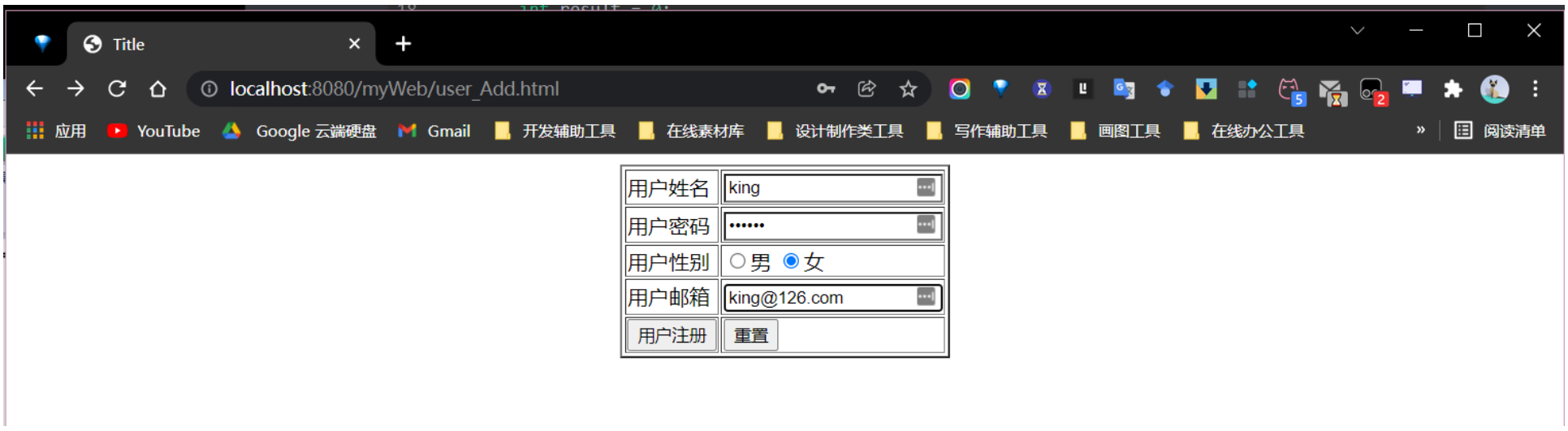
3、User_Add开发

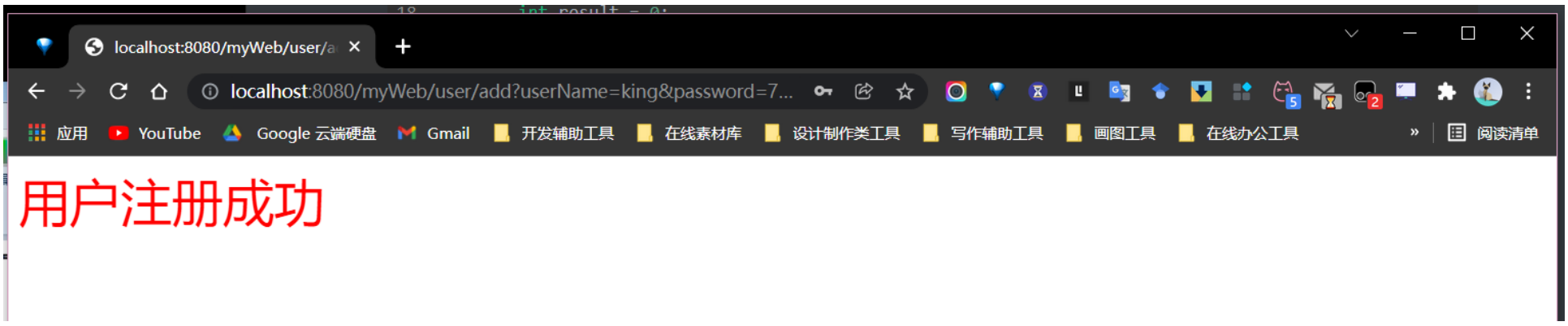
```
<!--user_Add.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <center>
        <form action="/myWeb/user/add" method="get">
            <table border="2">
              <tr>
                <td>用户姓名</td>
                <td><input type="text" name="userName"></td>
              </tr>
              <tr>
                <td>用户密码</td>
                <td><input type="password" name="password"></td>
              </tr>
              <tr>
                <td>用户性别</td>
                <td>
                  <input type="radio" name="sex" value="男" />男
                  <input type="radio" name="sex" value="女" />女
                </td>
              </tr>
              <tr>
                <td>用户邮箱</td>
                <td><input type="text" name="email"></td>
              </tr>
              <tr>
                <td><input type="submit" value="用户注册" /></td>
                <td><input type="reset" name="重置" /></td>
              </tr>
            </table>
        </form>
    </center>
</body>
</html>
```

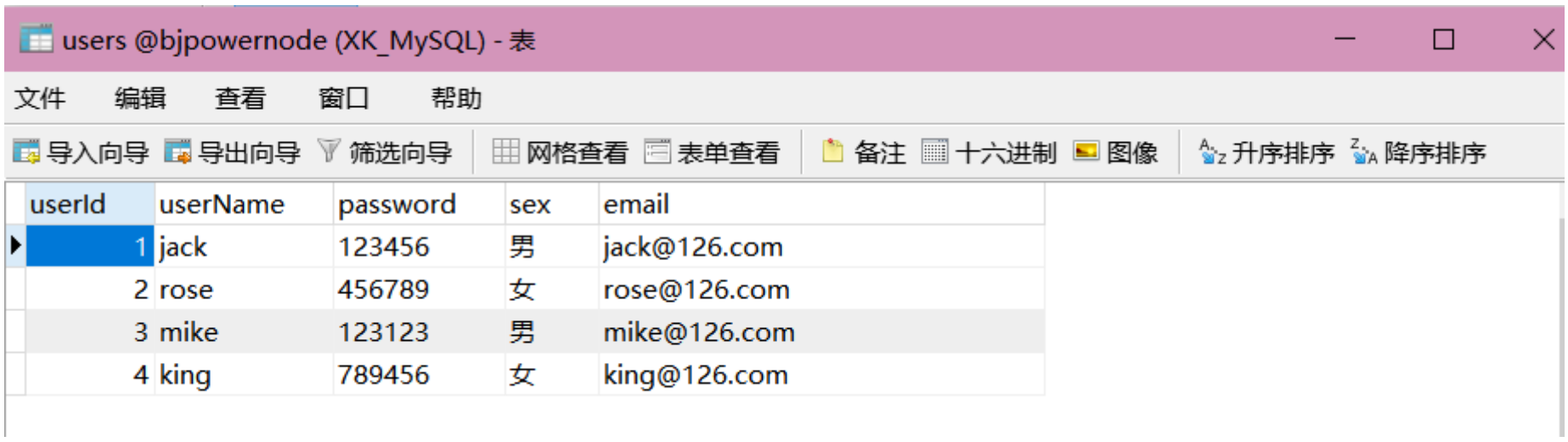
```
package com.example.controller;

import com.example.dao.UserDao;
import com.example.entity.Users;
```

```java

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;

public class UserAddServlet extends HttpServlet {//别名：/user/add
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //1.【调用请求对象】读取【请求头】参数信息，得到用户的信息
        String userName, passWord, sex, email;
        userName = request.getParameter("userName");
        passWord = request.getParameter("password");
        sex = request.getParameter("sex");
        email = request.getParameter("email");
        //2.【调用UserDao】将用户信息填充到INSERT命令并借助JDBC规范发送到数据库服务器
        UserDao userDao = new UserDao();
        Users user = new Users(null, userName, passWord, sex, email);
        int result = userDao.add(user);
        //3.【调用响应对象】将【处理结果】以二进制形式写入到响应体
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = null;
        out = response.getWriter();
        if(result == 1){
            out.print("<font style='color:red;font-size:40'>用户注册成功</font>");
        }else {
            out.print("<font style='color:red;font-size:40'>用户注册失败</font>");
        }
    }
    //Tocmat负责销毁【请求对象】和【响应对象】
    //Tomcat负责将Http响应协议包推送到发起请求的浏览器上
    //浏览器根据响应头content-type指定编译器对响应体二进制内容编辑
    //浏览器将编辑后结果在窗口中展示给用户【结束】
}
```

```java
package com.example.dao;

import com.example.entity.Users;
import com.example.util.JdbcUtil;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class UserDao {
    /**
     * 用户注册
     * @return 1---注册成功，0---注册失败
     */
    public int add(Users user){
        Connection connection = null;
        PreparedStatement ps = null;
        int result = 0;
        try {
            connection = JdbcUtil.getConnection();
            String sql = "insert into users(userName,password,sex,email)" +
                    "values(?,?,?,?)";
            ps = connection.prepareStatement(sql);
            ps.setString(1, user.getUserName());
            ps.setString(2, user.getPassWord());
            ps.setString(3, user.getSex());
            ps.setString(4, user.getEmail());
            result = ps.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }finally {
            JdbcUtil.close(connection, ps, null);
        }
        return result;
    }
}
```

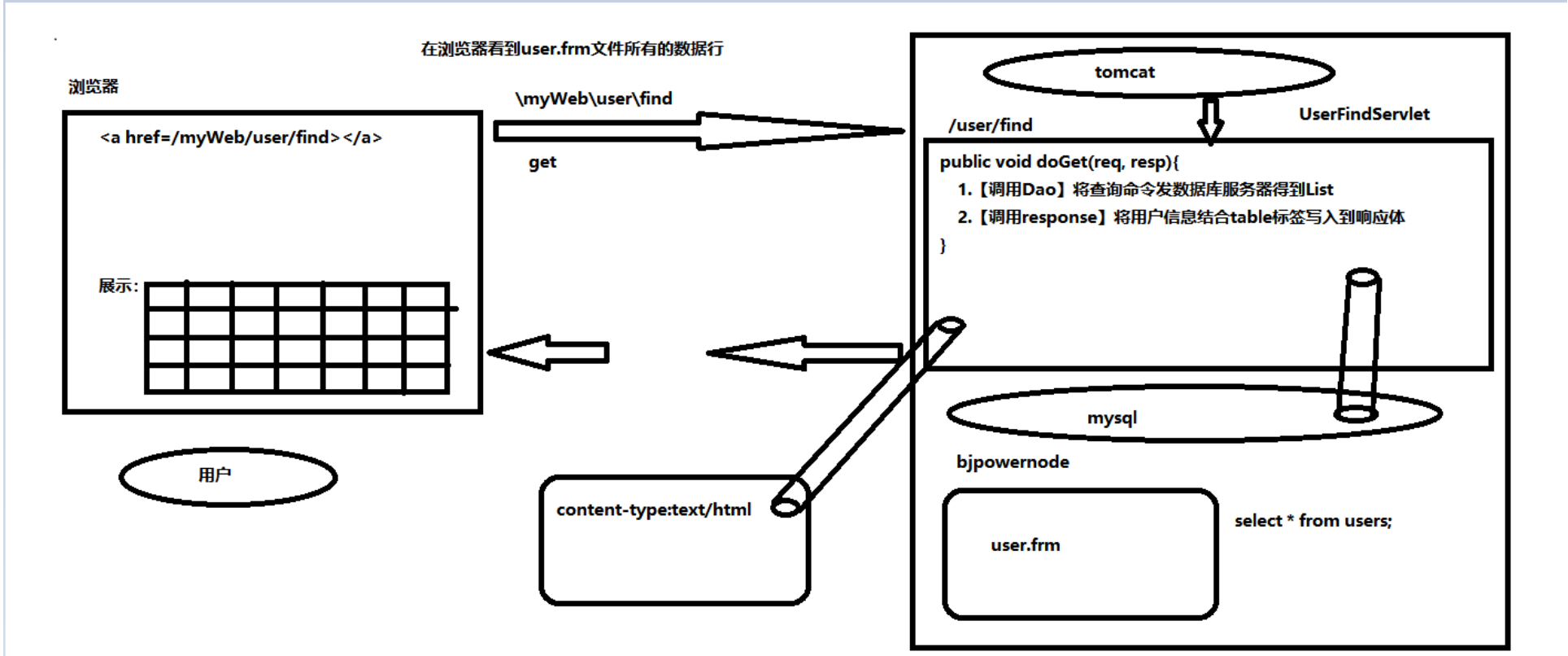






userId	userName	password	sex	email
1	jack	123456	男	jack@126.com
2	rose	456789	女	rose@126.com
3	mike	123123	男	mike@126.com
4	king	789456	女	king@126.com

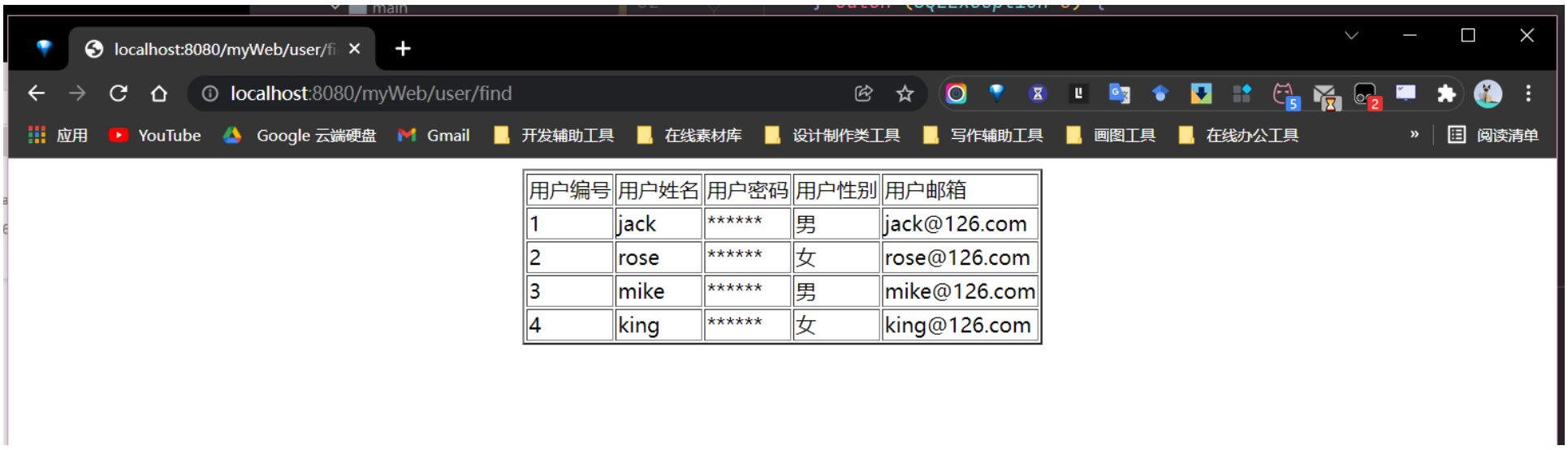
4、user Find开发




```java
package com.example.controller;

import com.example.dao.UserDao;
import com.example.entity.Users;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

public class UserFindServlet extends HttpServlet {//别名：/user/find
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //1、[调用UserDao] 将查询命令推送到数据库服务器上，得到所有用户信息【List】
        UserDao userDao = new UserDao();
        List<Users> allUsers = userDao.findAll();
        //2、[调用响应对象] 将用户信息结合<table>标签命令以二进制形式写入到响应体
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        out.print("<table border='2' align='center' >");
        out.print("<tr>");
        out.print("<td>用户编号</td>");
        out.print("<td>用户姓名</td>");
        out.print("<td>用户密码</td>");
        out.print("<td>用户性别</td>");
        out.print("<td>用户邮箱</td>");
        out.print("</tr>");
        for(Users user : allUsers){
            out.print("<tr>");
            out.print("<td>" + user.getUserId() +"</td>");
            out.print("<td>" + user.getUserName() +"</td>");
            out.print("<td>******</td>");
```

```
            out.print("<td>" + user.getSex() +"</td>");
            out.print("<td>" + user.getEmail() +"</td>");
            out.print("</tr>");
        }
        out.print("</table>");
    }
}
```

```
//UserDao
    /**
     * 查询所有用户信息
     * @return list集合存储所有用户信息
     */
    public List<Users> findAll(){
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        List<Users> list = new ArrayList<>();
        try {
            conn = JdbcUtil.getConnection();
            String sql = "select * from users";
            ps = conn.prepareStatement(sql);
            rs = ps.executeQuery();
            while(rs.next()){
                Integer userId = rs.getInt("userId");
                String userName = rs.getString("userName");
                String password = rs.getString("password");
                String sex = rs.getString("sex");
                String email = rs.getString("email");
                Users user = new Users(userId,userName,password,sex,email);
                list.add(user);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            JdbcUtil.close(conn, ps, rs);
        }
        return list;
    }
```

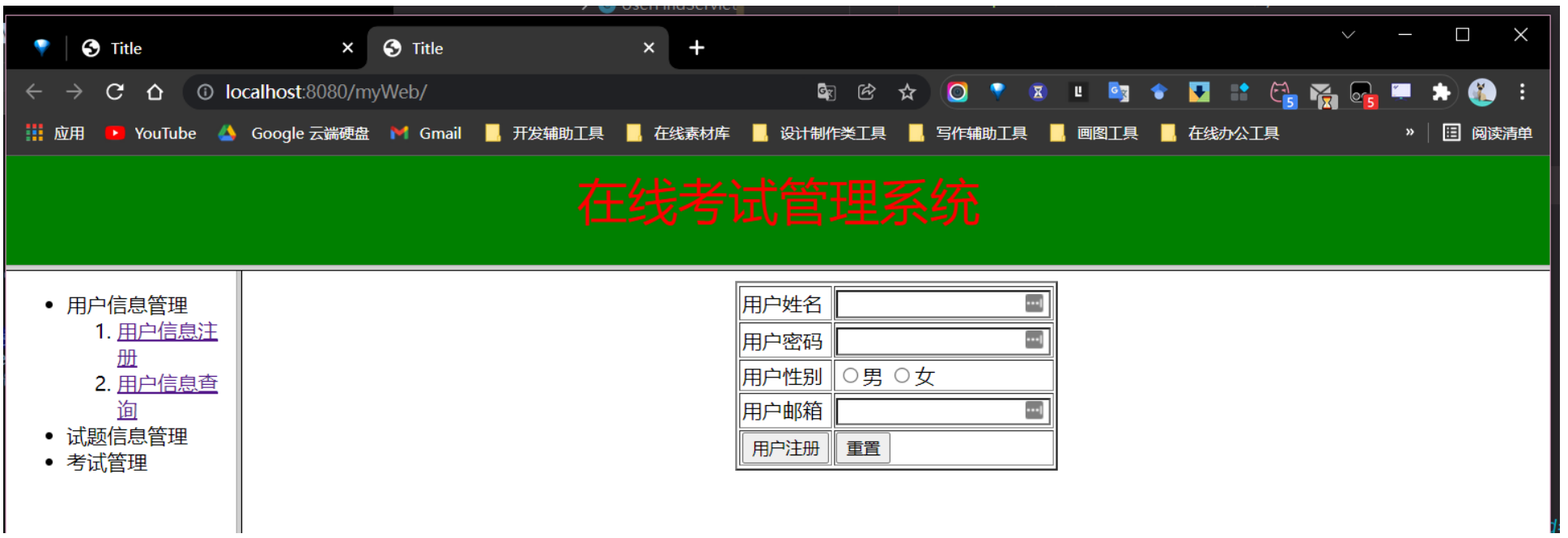


5、导航页面

```
<!-- webapp/index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<frameset rows="15%,85%">
  <frame name="top" src="top.html">
  <frameset cols="15%,85%">
    <frame name="left" src="left.html">
    <frame name="right">
  </frameset>
</frameset>
</html>
```

```html
<!-- webapp/top.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body style="background-color: green">
  <center>
    <font style="color:red;font-size:40px">在线考试管理系统</font>
  </center>
</body>
</html>
```

```html
<!-- webapp/left.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
  <ul>
    <li>用户信息管理
      <ol>
        <li><a href="/myWeb/user_Add.html" target="right">用户信息注册</a> </li>
        <li><a href="/myWeb/user/find" target="right">用户信息查询</a></li>
      </ol>
    </li>
    <li>试题信息管理</li>
    <li>考试管理</li>
  </ul>
</body>
</html>
```



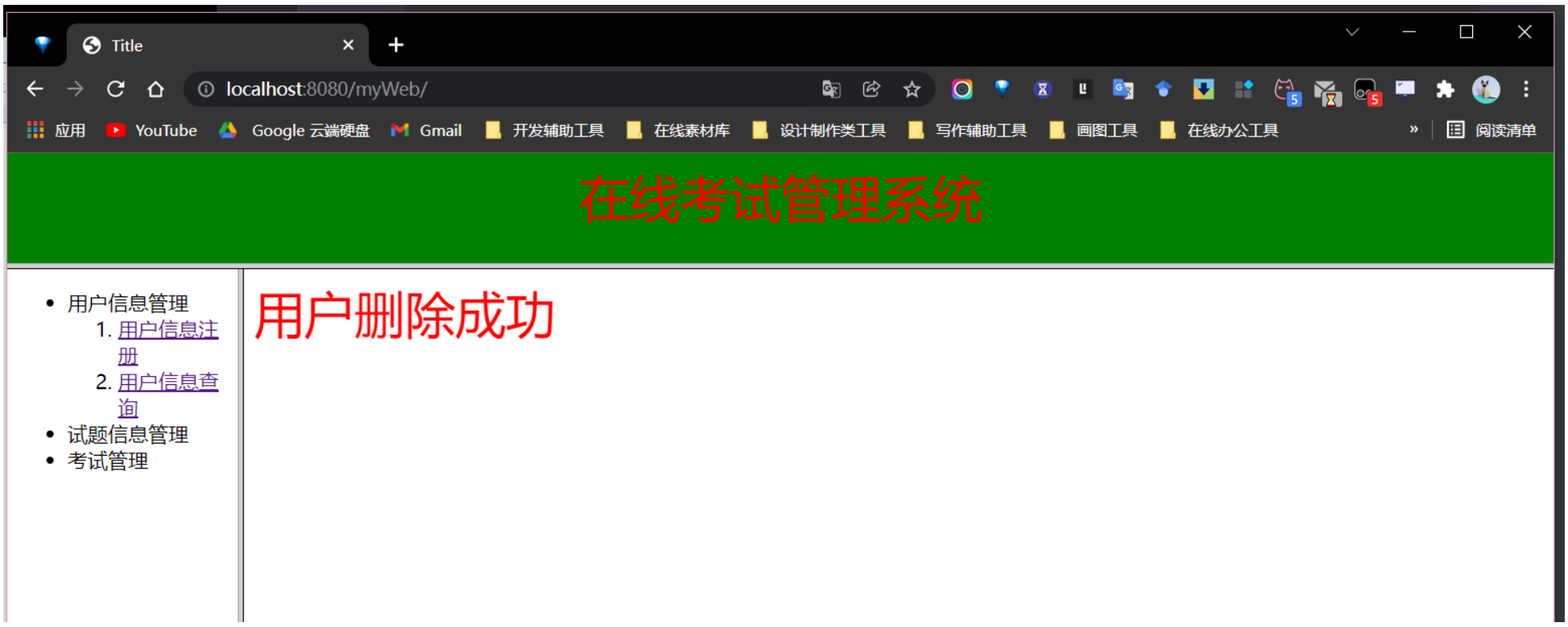
6、user_Delete开发

```java
package com.example.controller;

import com.example.dao.UserDao;
import com.example.entity.Users;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;
```

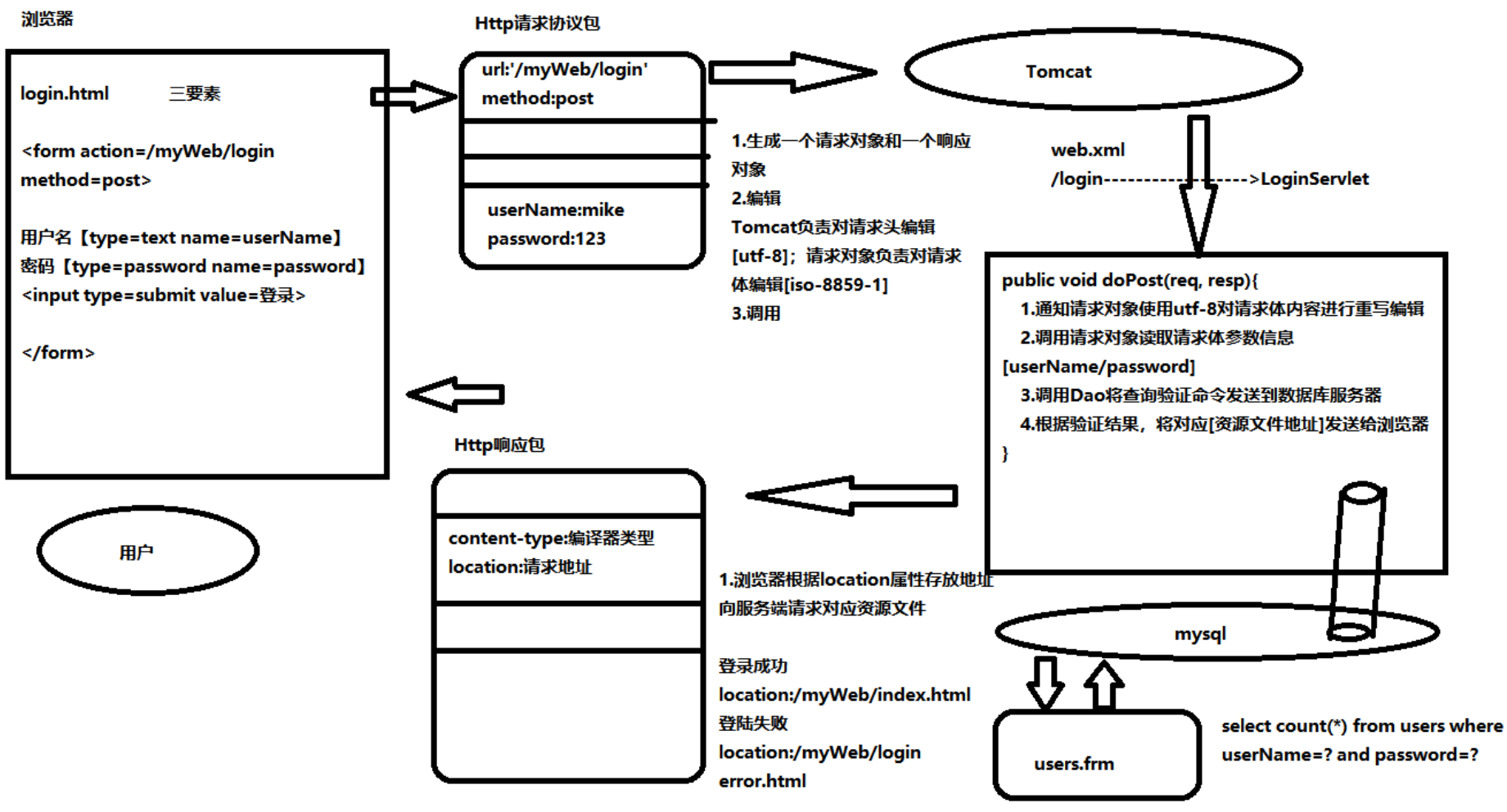
```java
import java.util.List;
public class UserFindServlet extends HttpServlet {//别名：user/find
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //1、[调用UserDao] 将查询命令推送到数据库服务器上，得到所有用户信息【List】
        UserDao userDao = new UserDao();
        List<Users> allUsers = userDao.findAll();
        //2、[调用响应对象] 将用户信息结合<table>标签命令以二进制形式写入到响应体
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        out.print("<table border='2' align='center' >");
        out.print("<tr>");
        out.print("<td>用户编号</td>");
        out.print("<td>用户姓名</td>");
        out.print("<td>用户密码</td>");
        out.print("<td>用户性别</td>");
        out.print("<td>用户邮箱</td>");
        out.print("<td>操作</td>");
        out.print("</tr>");
        for(Users user : allUsers){
            out.print("<tr>");
            out.print("<td>" + user.getUserId() +"</td>");
            out.print("<td>" + user.getUserName() +"</td>");
            out.print("<td>******</td>");
            out.print("<td>" + user.getSex() +"</td>");
            out.print("<td>" + user.getEmail() +"</td>");
            out.print("<td><a href='/myWeb/user/delete?userId=" + user.getUserId() + "' >删除用户</a></td>");
            out.print("</tr>");
        }
        out.print("</table>");
    }
}
```

```java
package com.example.controller;

import com.example.dao.UserDao;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;

public class UserDeleteServlet extends HttpServlet {//别名：/user/delete
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //1、【调用请求对象】读取【请求头】参数（用户编号）
        String userId = request.getParameter("userId");
        //2、【调用UserDao】将用户编号填充到delete命令并发送到数据库服务器
        UserDao userDao = new UserDao();
        int result = userDao.deleteUser(userId);
        //3、【调用响应对象】将处理结果以二进制写入到响应体，交给浏览器
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        if(result == 1){
            out.print("<font style='color:red;font-size:40'>用户删除成功</font>");
        } else {
            out.print("<font style='color:red;font-size:40'>用户删除失败</font>");
        }
    }
}
```

```java
//UserDao
    /**
     * 删除指定用户信息
     * @param userId
     * @return
     */
    public int deleteUser(String userId){
        Connection conn = null;
        PreparedStatement ps = null;
        int result = 0;
        try {
            conn = JdbcUtil.getConnection();
            String sql = "delete from users where userId=?";
            ps = conn.prepareStatement(sql);
            ps.setInt(1, Integer.valueOf(userId));
            result = ps.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            JdbcUtil.close(conn, ps, null);
        }
        return result;
    }
```



7、登录验证




```java
package com.example.controller;

import com.example.dao.UserDao;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class LoginServlet extends HttpServlet {//别名:/myWeb/login
```

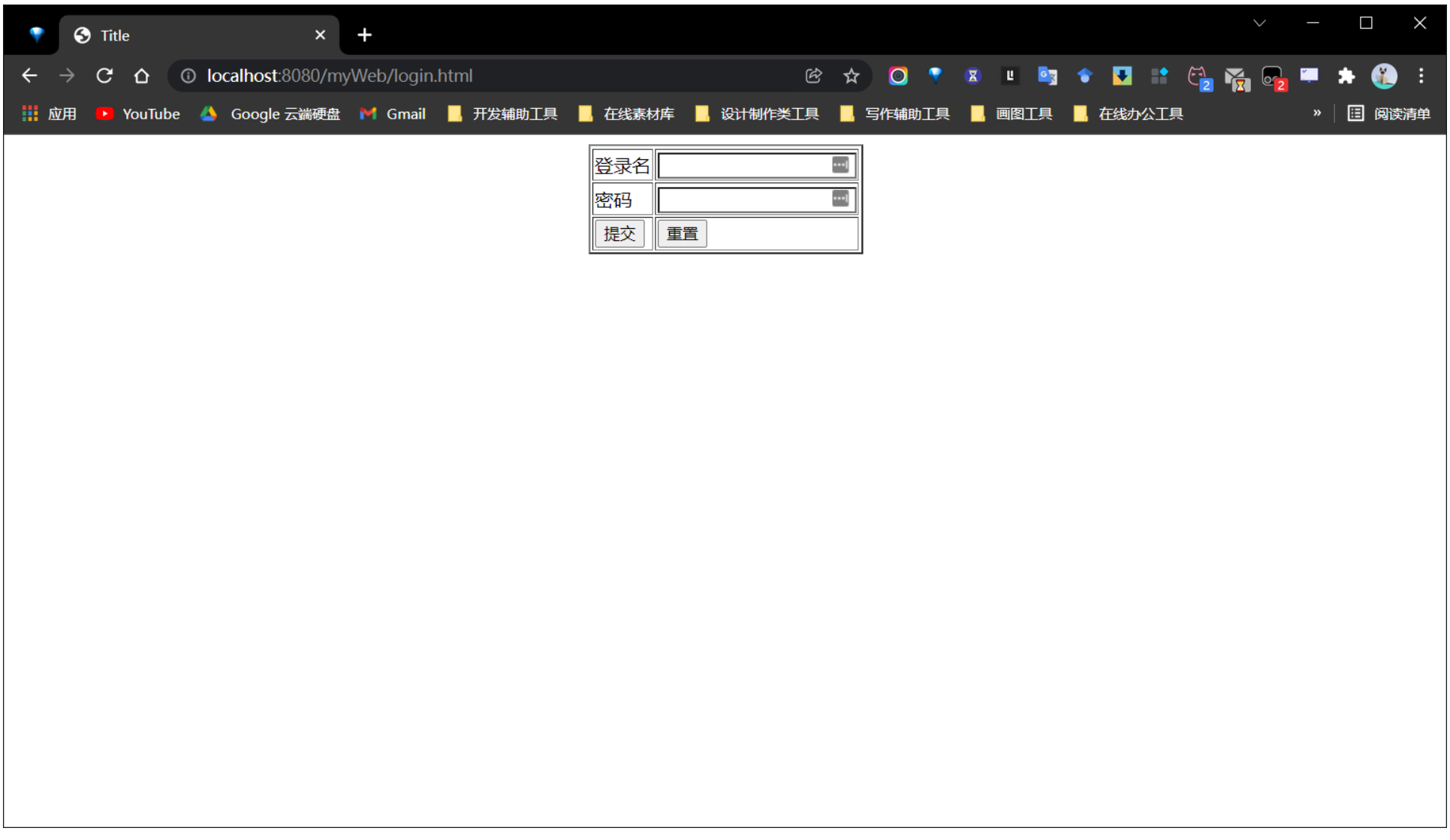
```java
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // 1.调用请求对象对请求体使用utf-8字符集进行重新编辑
        request.setCharacterEncoding("utf-8");
        // 2.调用请求对象读取请求体参数信息
        String userName = request.getParameter("userName");
        String password = request.getParameter("password");
        // 3.调用UserDao将查询验证信息推送到数据库服务器上
        UserDao userDao = new UserDao();
        int result = userDao.ligin(userName, password);
        //4.调用响应对象，根据验证结果将不同资源文件地址写入到响应头，交给浏览器
        if(result == 1){
            response.sendRedirect("/myWeb/index.html");
        } else {
            response.sendRedirect("/myWeb/login_error.html");
        }
    }
}
```

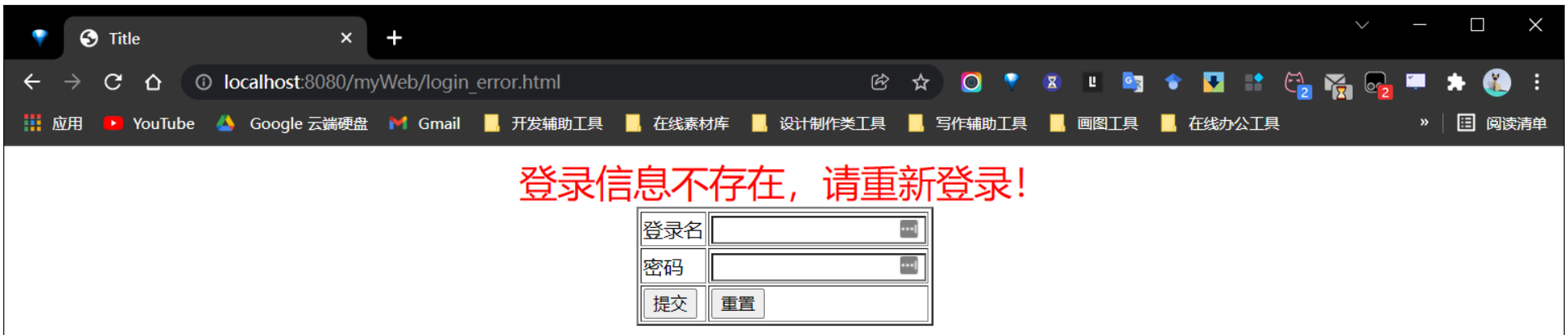
```java
//UserDao.java
    /**
     * 用户登录验证
     * @param userName
     * @param password
     * @return
     */
    public int ligin(String userName, String password){
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        int result = 0;
        try {
            conn = JdbcUtil.getConnection();
            String sql = "select count(*) from users where userName=? and password=?";
            ps = conn.prepareStatement(sql);
            ps.setString(1, userName);
            ps.setString(2, password);
            rs = ps.executeQuery();
            while(rs.next()){
                result = rs.getInt("count(*)");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            JdbcUtil.close(conn, ps, rs);
        }
        return result;
    }
```

```html
<!--login.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <center>
        <form action="/myWeb/login" method="post">
            <table border="2">
                <tr>
                    <td>登录名</td>
                    <td><input type="text" name="userName" /></td>
                </tr>
                <tr>
                    <td>密码</td>
                    <td><input type="password" name="password" /></td>
                </tr>
                <tr>
                    <td><input type="submit" value="提交" /></td>
                    <td><input type="reset" value="重置" /></td>
                </tr>

            </table>
        </form>
    </center>
</body>
</html>
```

```html
<!--login_error.html-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
```

```
</head>
<body>
<center>
  <font style="color: red;font-size: 30px">登录信息不存在，请重新登录！</font>
  <form action="/myWeb/login" method="post">
    <table border="2">
      <tr>
        <td>登录名</td>
        <td><input type="text" name="userName" /></td>
      </tr>
      <tr>
        <td>密码</td>
        <td><input type="password" name="password" /></td>
      </tr>
      <tr>
        <td><input type="submit" value="提交" /></td>
        <td><input type="reset" value="重置" /></td>
      </tr>
    </table>
  </form>
</center>
</body>
</html>
```





九、欢迎资源文件

1. 前提：用户可以记住网站名，但是不会记住网站资源文件名。
2. 默认欢迎资源文件:

   用户发送了一个针对某个网站的【默认请求】时，此时由Http服务器自动从当前网站返回的资源文件

   正常请求： http://localhost:8080/myWeb/index.html

   默认请求： http://localhost:8080/myWeb/
3. Tomcat对于默认欢迎资源文件定位规则
   - 规则位置：Tomcat安装位置/conf/web.xml
   - 规则命令：

```
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

4. 设置当前网站的默认欢迎资源文件规则
   - 规则位置：网站/web/WEB-INF/web.xml
   - 规则命令：

```
<!--自定义默认欢迎资源文件-->
    <welcome-file-list>
        <welcome-file>login.html</welcome-file>
        <welcome-file>user/find</welcome-file><!--servlet作为默认欢迎文件时，开头斜线必须去掉-->
    </welcome-file-list>
```

   - 网站设置自定义默认文件定位规则，此时Tomcat自带定位规则将失效

十、*Http*状态码

1. 介绍：
   - 由三位数字组成的一个符号
   - Http服务器在推送响应包之前，根据本次请求处理情况将Http状态码写入到响应包中【状态行】上
   - 如果Http服务器针对本次请求，返回了对应的资源文件。 通过Http状态码通知浏览器应该如何处理这个结果

     如果Http服务器针对本次请求，无法返回对应的资源文件；通过Http状态码向浏览器解释不能提供服务的原因
2. 分类：
   - 组成　100 --- 599; 分为5个大类
   - 1xx:

最有特征100：通知浏览器本次返回的资源文件, 并不是一个独立的资源文件，需要浏览器在接收响应包之后，继续向Http服务器所要依赖的其他资源文件

- 2xx:

  最有特征200：通知浏览器本次返回的资源文件是一个完整独立资源文件，浏览器在接收到之后不需要所要其他关联文件

- 3xx:

  最有特征302：通知浏览器本次返回的不是一个资源文件内容，而是一个资源文件地址，需要浏览器根据这个地址自动发起请求来索要这个资源文件

  response.sendRedirect('资源文件地址')写入到响应头中locatio；而这个行为导致Tomcat将302状态码写入到状态行

```java
public class OneServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
          String address = "http://www.baidu.com";
          response.sendRedirect(address); //写入到响应头 location
    }
    //Tomcat在推送响应包之前，看到响应体是空，但是响应头location却存放了一个地址。
    //此时Tomcat将302状态码写入到状态行
    //在浏览器接收到响应包之后，因为302状态码，浏览器不会读取响应体内容，自动根据响应头
    //中location的地址发起第二次请求
}
```

- 4xx:

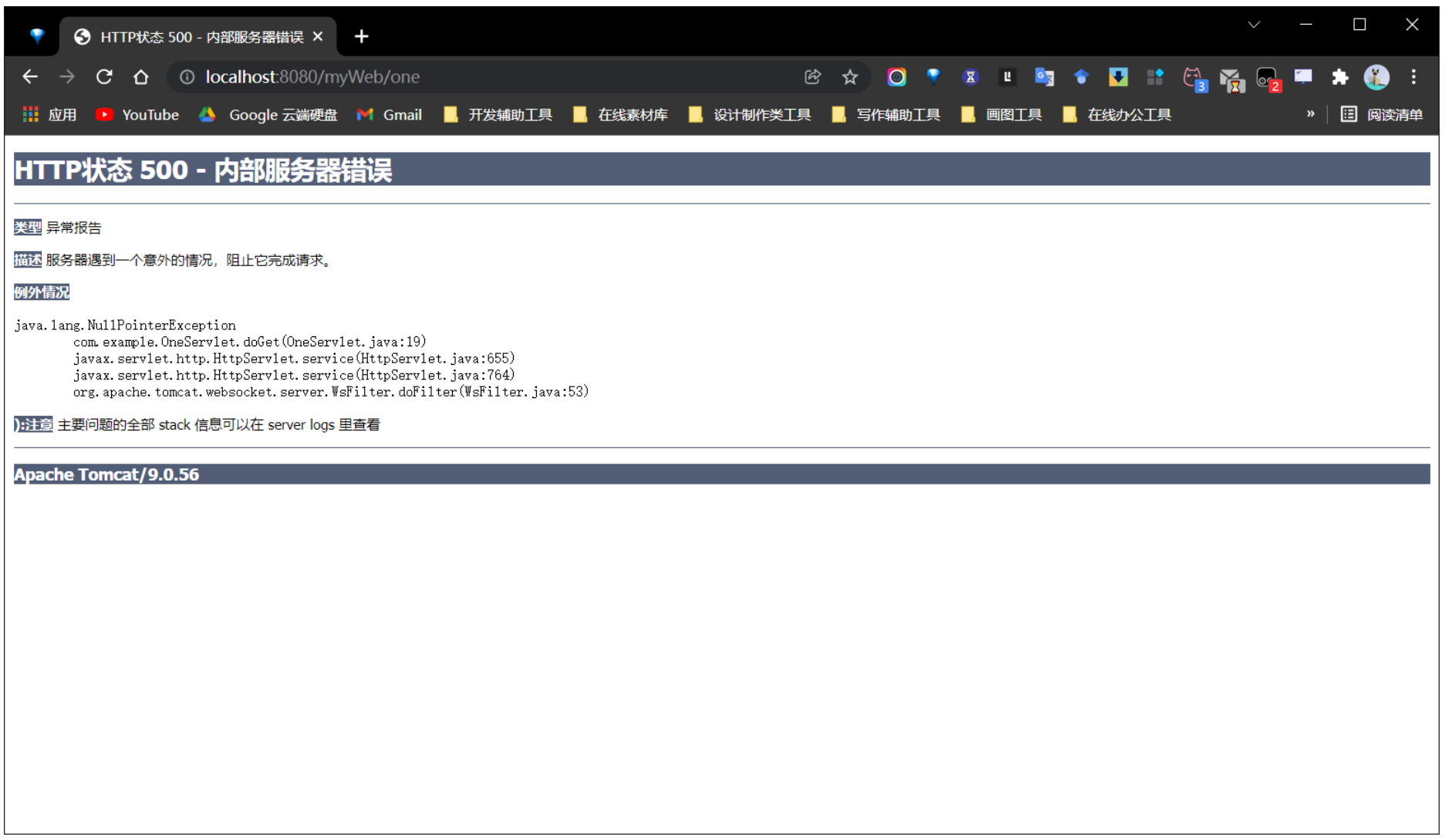
  404：通知浏览器，由于在服务端没有定位到被访问的资源文件,因此无法提供帮助

  405：通知浏览器，在服务端已经定位到被访问的资源文件(Servlet)；但是这个Servlet对于浏览器采用的请求方式不能处理

- 5xx:

  500：通知浏览器，在服务端已经定位到被访问的资源文件(Servlet)；这个Servlet可以接收浏览器采用请求方式，但是Servlet在处理请求期间，由于Java异常导致处理失败

```java
public class OneServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        Map map = new HashMap();
        int num = (int) map.get("key1");//抛出空指针异常
        //int a = null; // null值不可能赋值给int
        //Integer b = null; //所有高级引用类型都可以赋值给null
    }
}
```



十一、多个*Servlet*之间调用规则

1. 前提条件：

   某些来自于浏览器发送请求，往往需要服务端中多个Servlet协同处理。但是浏览器一次只能访问一个Servlet，导致用户需要手动通过浏览器发起多次请求才能得到服务。

   这样增加用户获得服务难度，导致用户放弃访问当前网站

2. 提高用户使用感受规则

   无论本次请求涉及到多少个Servlet,用户只需要【手动】通知浏览器发起一次请求即可

3. 多个Servlet之间调用规则：

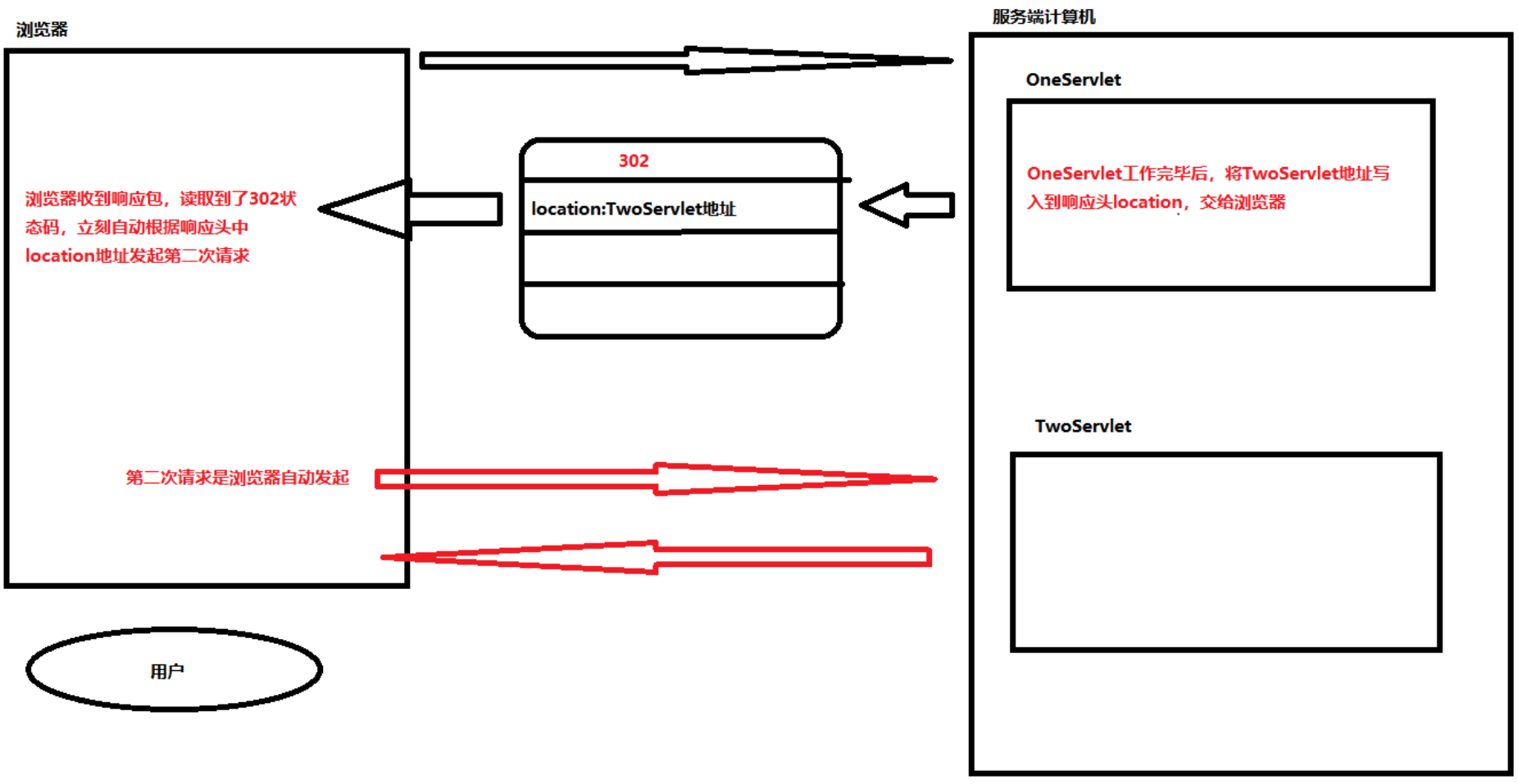
- 重定向解决方案
- 请求转发解决方案

1、重定向解决方案

1. 工作原理：

用户第一次通过【手动方式】通知浏览器访问OneServlet；OneServlet工作完毕后，将TwoServlet地址写入到响应头location属性中，导致Tomcat将302状态码写入到状态行。

在浏览器接收到响应包之后，会读取到302状态。此时浏览器自动根据响应头中location属性地址发起第二次请求，访问TwoServlet去完成请求中剩余任务



2. 实现命令：

response.sendRedirect("请求地址")

将地址写入到响应包中响应头中location属性

```
public class OneServlet extends HttpServlet {//别名：/myWeb/one
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("OneServlet 负责 洗韭菜");
        //重定向解决方案：
        response.sendRedirect("/myWeb/two");// [地址格式：/网站名/资源文件名]
    }
}

public class TwoServlet extends HttpServlet {//别名：/myWeb/two
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("TwoServlet 负责 韭菜炒鸡蛋 ");
    }
}
```

3. 特征

- 请求地址：

  既可以把当前网站内部的资源文件地址发送给浏览器 （/网站名/资源文件名）
  也可以把其他网站资源文件地址发送给浏览器(http://ip地址:端口号/网站名/资源文件名)

- 请求次数：

  浏览器**至少**发送两次请求，但是只有第一次请求是用户手动发送；后续请求都是浏览器自动发送的。

- 请求方式：

  重定向解决方案中，通过地址栏通知浏览器发起下一次请求，因此通过重定向解决方案调用的资源文件接收的请求方式一定是【GET】

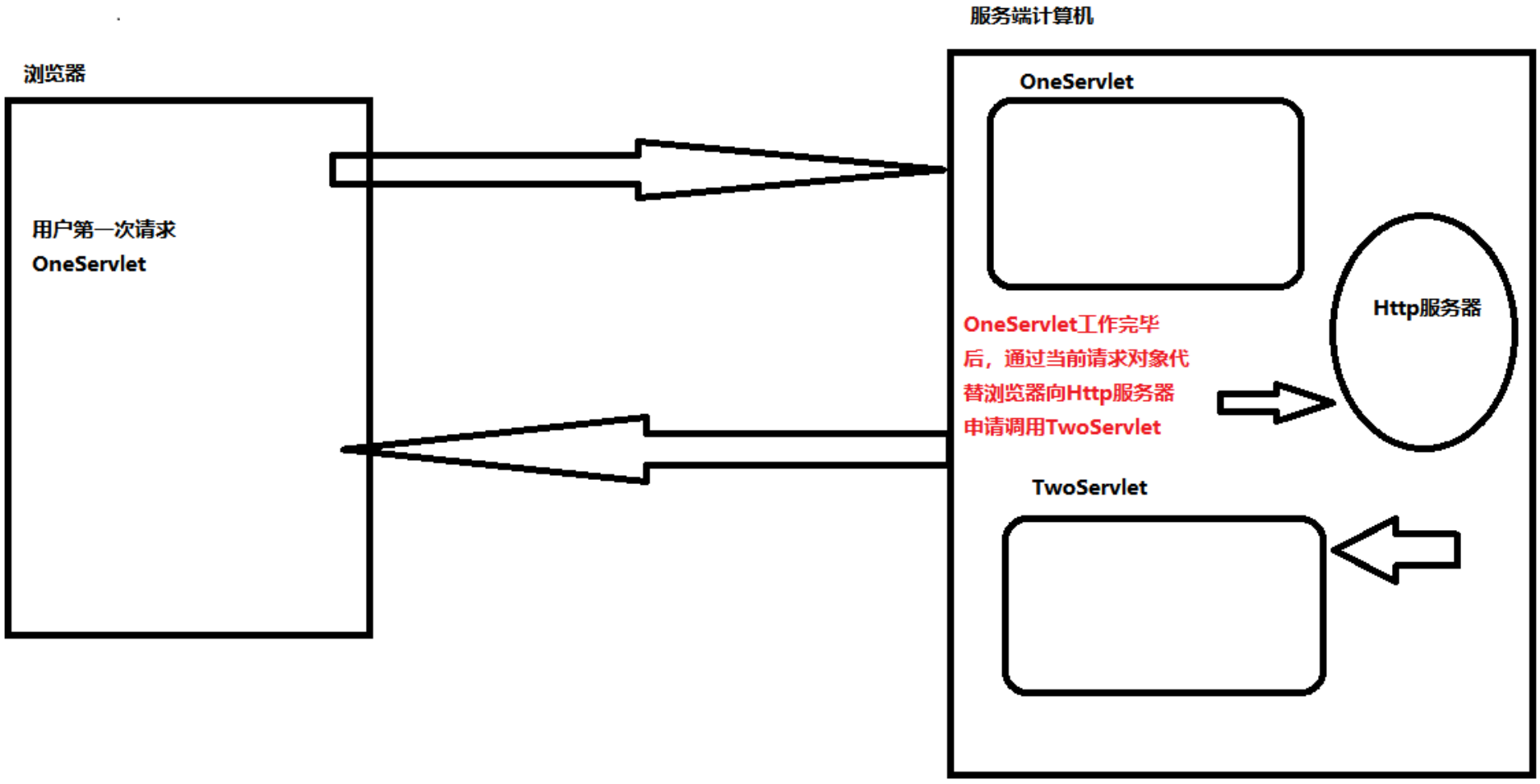
4. 缺点：

重定向解决方案需要在浏览器与服务器之间进行多次往返，大量时间消耗在往返次数上，增加用户等待服务时间

2、请求转发解决方案

1. 工作原理：

用户第一次通过手动方式要求浏览器访问OneServlet；OneServlet工作完毕后，通过当前的请求对象代替浏览器向Tomcat发送请求，申请调用TwoServlet。Tomcat在接收到这个请求之后，自动调用TwoServlet来完成剩余任务。



2. 实现命令：请求对象代替浏览器向Tomcat发送请求

```
//1.通过当前请求对象生成资源文件申请报告对象
RequestDispatcher report = request.getRequestDispatcher("/资源文件名");//一定要以"/"为开头
//2.将报告对象发送给Tomcat
report.forward(当前请求对象，当前响应对象);
```

```
public class OneServlet extends HttpServlet {//别名：/myWeb/one
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("OneServlet 开始工作.....");
        //请求转发方案
        //1.通过当前请求对象生成资源文件申请报告对象
        RequestDispatcher report = request.getRequestDispatcher("/two");
        //2.将报告对象发送给Tomcat
        report.forward(request, response);
    }
}

public class TwoServlet extends HttpServlet {//别名：/myWeb/two
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("OneServlet 工作完毕后，TwoServlet 开始工作.....");
    }
}
```

3. 优点

- 无论本次请求涉及到多少个Servlet,用户只需要手动通过浏览器发送一次请求
- Servlet之间调用发生在服务端计算机上，节省服务端与浏览器之间往返次数增加处理服务速度

4. 特征：

- 请求次数：在请求转发过程中，浏览器只发送一次请求
- 请求地址：

  只能向Tomcat服务器申请调用当前网站下资源文件地址
  request.getRequestDispathcer("/资源文件名") **不要写网站名**
- 请求方式：

  在请求转发过程中，浏览器只发送一个了个Http请求协议包；参与本次请求的所有Servlet共享同一个请求协议包，因此这些Servlet接收的请求方式与浏览器发送的请求方式保持一致

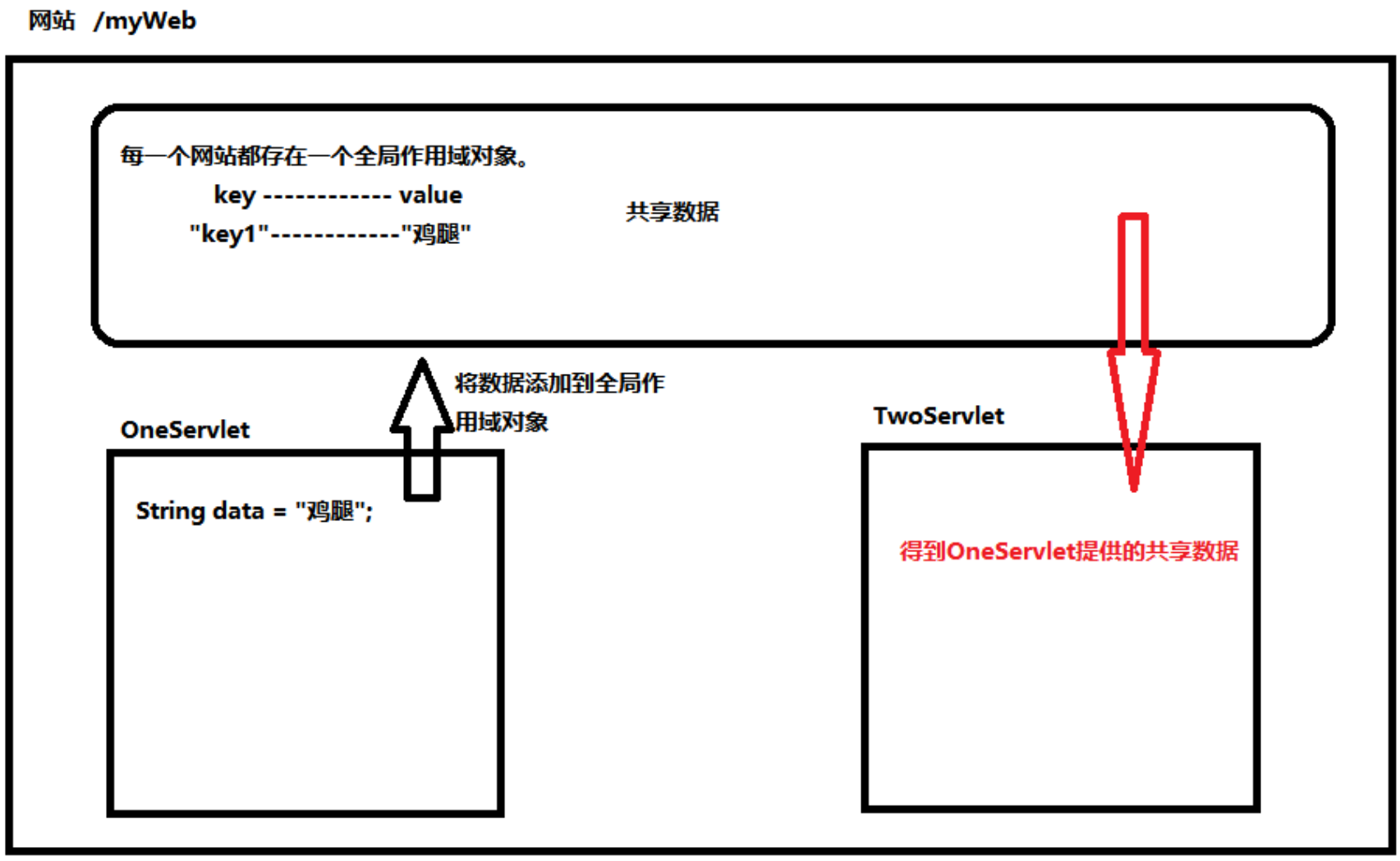
十二、多个*Servlet*之间数据共享实现方案

- 数据共享：OneServlet工作完毕后，将产生数据交给TwoServlet来使用
- Servlet规范中提供四种数据共享方案
  1. ServletContext接口
  2. Cookie类
  3. HttpSession接口
  4. HttpServletRequest接口

1、ServletContext接口

1. 介绍：
   - 来自于Servlet规范中一个接口。在Tomcat中存在servlet-api.jar，在Tomcat中负责提供这个接口实现类
   - 如果两个Servlet来自于同一个网站。彼此之间通过网站的ServletContext实例对象实现数据共享
   - 开发人员习惯于将ServletContext对象称为【全局作用域对象】
2. 工作原理：

   每一个网站都存在一个全局作用域对象。这个全局作用域对象【相当于】一个Map，在这个网站中OneServlet可以将一个数据存入到全局作用域对象，当前网站中其他Servlet此时都可以从全局作用域对象得到这个数据进行使用



3. 全局作用域对象生命周期
   - 在Http服务器启动过程中，自动为当前网站在内存中创建一个全局作用域对象
   - 在Http服务器运行期间时，一个网站只有一个全局作用域对象
   - 在Http服务器运行期间，全局作用域对象一直处于存活状态
   - 在Http服务器准备关闭时，负责将当前网站中全局作用域对象进行销毁处理
   - 局作用域对象生命周期贯穿网站整个运行期间
4. 命令实现： 【同一个网站】OneServlet将数据共享给TwoServlet

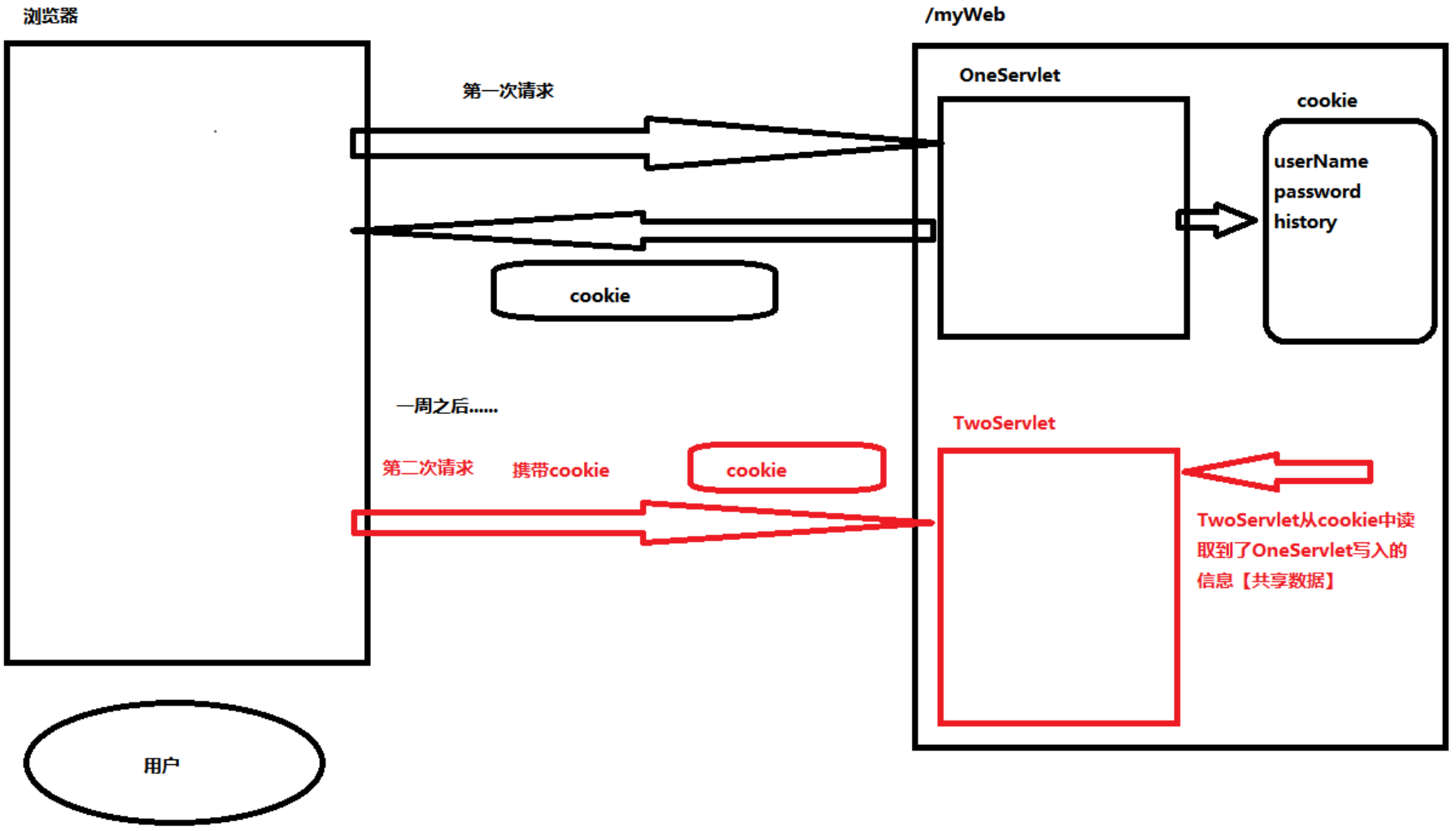
```
OneServlet{
    public void doGet(HttpServletRequest request,HttpServletResponse response){
        //1.通过【请求对象】向Tomcat索要当前网站中【全局作用域对象】
        ServletContext application = request.getServletContext();
        //2.将数据添加到全局作用域对象作为【共享数据】
        application.setAttribute("key1",数据);
    }
}

TwoServlet{
    public void doGet(HttpServletRequest request,HttpServletResponse response){
        //1.通过【请求对象】向Tomcat索要当前网站中【全局作用域对象】
        ServletContext application = request.getServletContext();
        //2.从全局作用域对象得到指定关键字对应数据
        Object 数据 =  application.getAttribute("key1");
    }
}
```

2、Cookie

1. 介绍：
   - Cookie来自于Servlet规范中一个工具类，存在于Tomcat提供servlet-api.jar中
   - 如果两个Servlet来自于同一个网站，并且为同一个浏览器/用户提供服务，此时借助于Cookie对象进行数据共享
   - Cookie存放当前用户的私人数据，在共享数据过程中提高服务质量
   - 在现实生活场景中，Cookie相当于用户在服务端得到【会员卡】
2. 工作原理：

```
用户通过浏览器第一次向myWeb网站发送请求申请OneServlet。OneServlet在运行期间创建一个Cookie存储与当前用户相关数据，OneServlet工作完毕后，【将Cookie写入到响应头】交还给当前浏览器。
```

浏览器收到响应响应包之后，将cookie存储在浏览器的缓存，一段时间之后，用户通过【同一个浏览器】再次向【myWeb网站】发送请求申请TwoServlet时。【浏览器需要无条件的将myWeb网站之前推送过来的Cookie，写入到请求头】发送过去，此时TwoServlet在运行时，就可以通过读取请求头中cookie中信息，得到OneServlet提供的共享数据



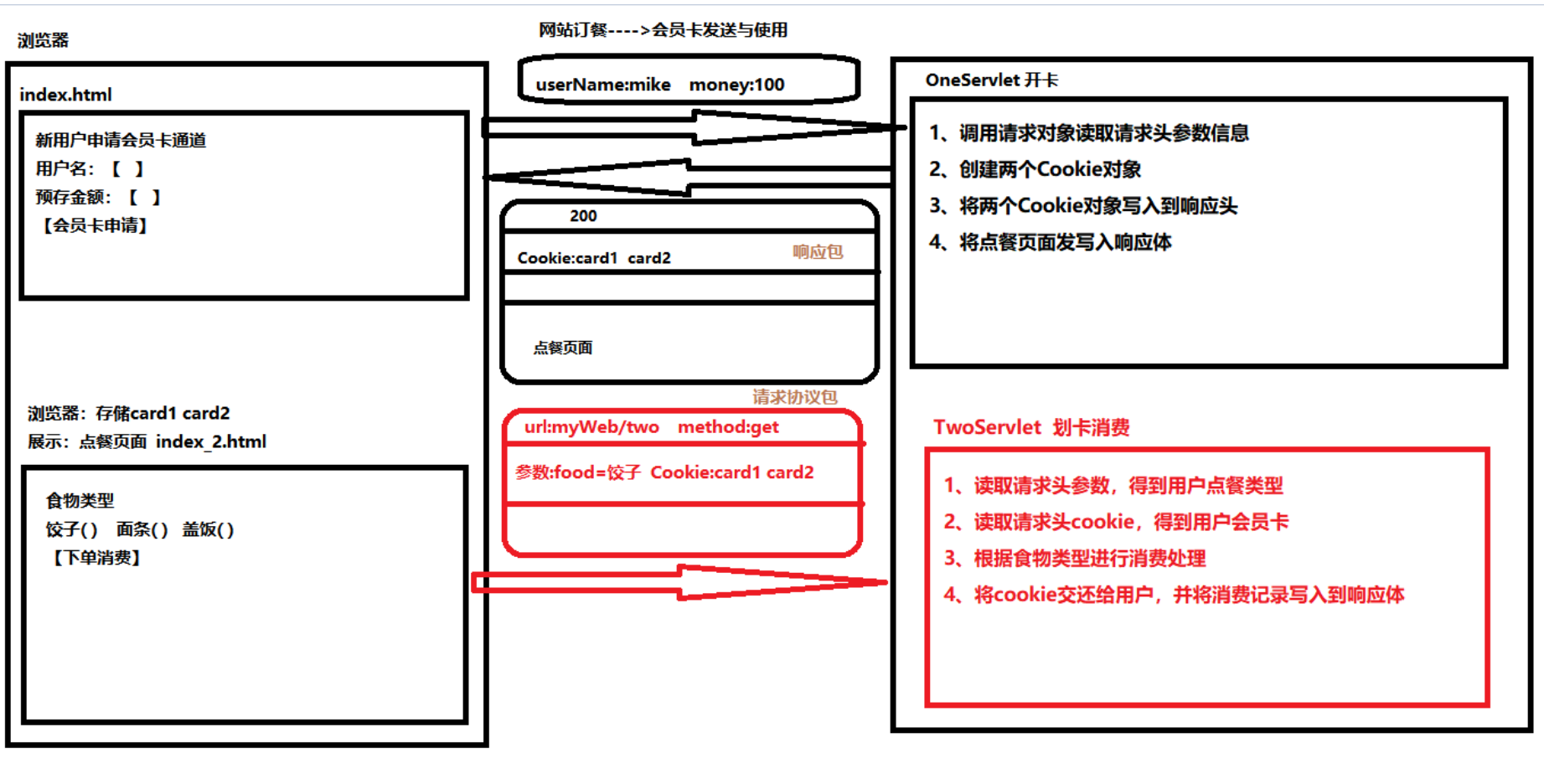
3. 实现命令

同一个网站 OneServlet 与 TwoServlet 借助于Cookie实现数据共享

```
OneServlet{
    public void doGet(HttpServletRequest request,HttpServletResponse resp){
        //1.创建一个cookie对象，保存共享数据（当前用户数据）
        Cookie card = new Cookie("key1","abc");
        Cookie card1= new Cookie("key2","efg");
        //****cookie相当于一个map
        //****一个cookie中只能存放一个键值对
        //****这个键值对的key与value只能是String
        //****键值对中key不能是中文
        //2.【发卡】将cookie写入到响应头，交给浏览器
        resp.addCookie(card);
        resp.addCookie(card1)
    }
}
```

```
浏览器/用户       <---------响应包 【200】
                                【cookie: key1=abc; key2=eft】
                               【】
                               【处理结果】

浏览器向myWeb网站发送请求访问TwoServlet---->请求包 【url:/myWeb/two method:get】
                                              【
                                                请求参数: xxxx
                                                Cookie: key1=abc;key2=efg
                                              】
                                             【】
                                             【】

TwoServlet{
    public void doGet(HttpServletRequest request,HttpServletResponse resp){
        //1.调用请求对象从请求头得到浏览器返回的Cookie
         Cookie  cookieArray[] = request.getCookies();
        //2.循环遍历数据得到每一个cookie的key 与 value
        for(Cookie card:cookieArray){
            String key =   card.getName(); //读取key  "key1"
            Strign value = card.getValue();//读取value "abc"
            //提供较好的服务。。。。。。。。
        }
    }
}
```

4. 应用实例_订餐会员卡



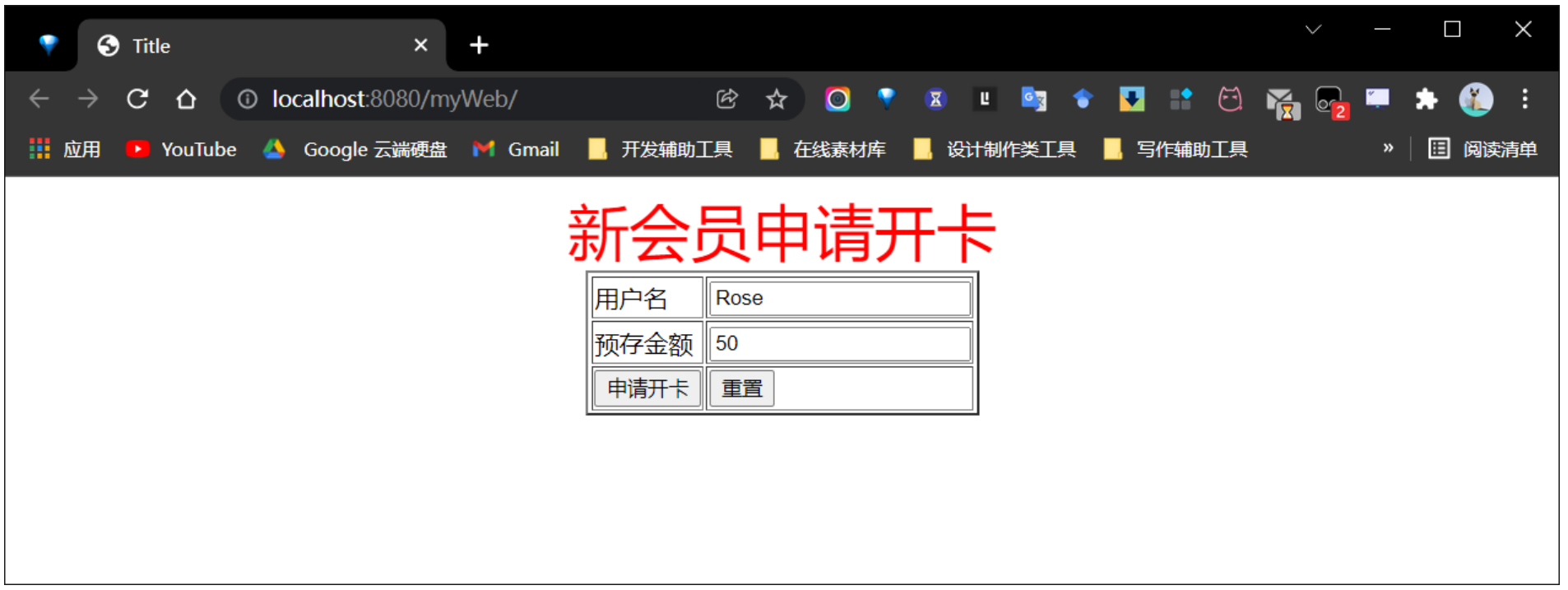

```java
public class OneServlet extends HttpServlet {//别名：myWeb/one
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //1、调用请求对象读取【请求头】参数信息
        String userName = request.getParameter("userName");
        String money = request.getParameter("money");
        //2、开卡
        Cookie card1 = new Cookie("userName", userName);
        Cookie card2 = new Cookie("money", money);
        //3、发卡，将cookie写入到【响应头】中交给浏览器
        response.addCookie(card1);
        response.addCookie(card2);
        //4、通知Tomcat将【点餐页面】内容写入到【响应体】交给浏览器
        request.getRequestDispatcher("/index_2.html").forward(request, response);
    }
}
```

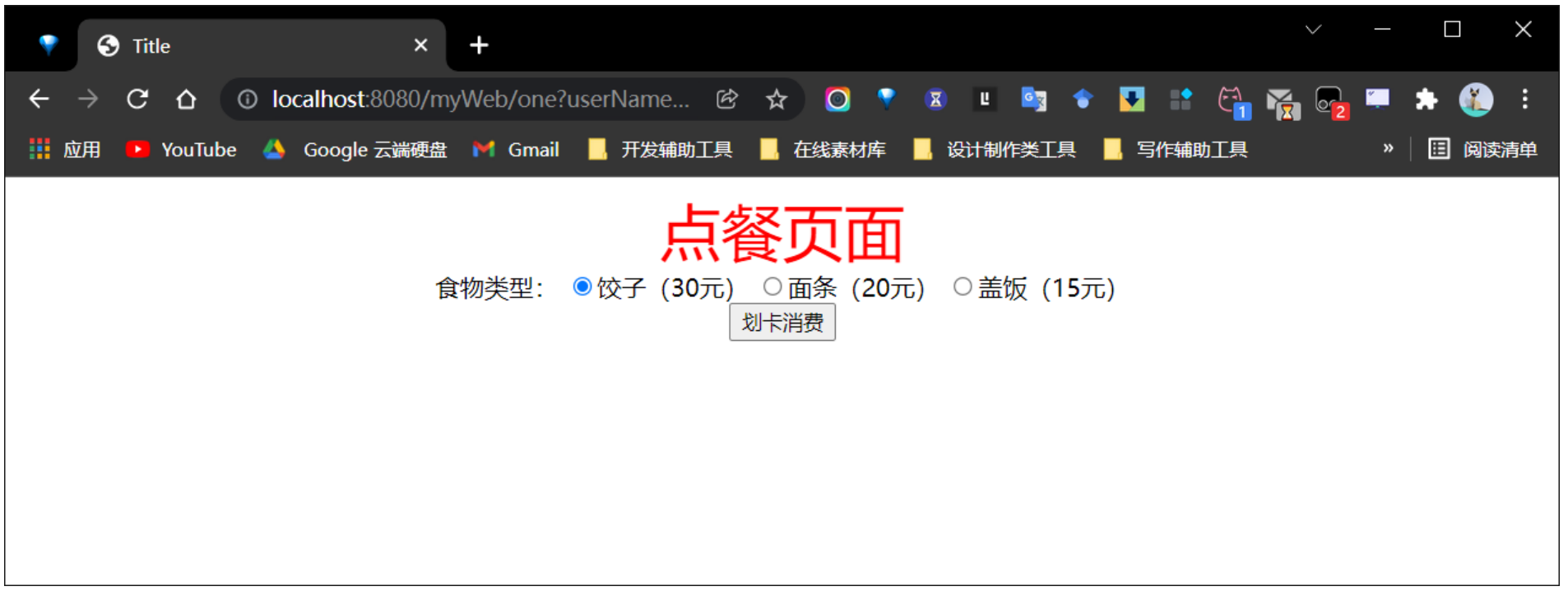
```java
public class TwoServlet extends HttpServlet {//别名：myWeb/two
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        int jiaozi_money = 30;
        int gaifan_money = 15;
        int miantiiao_money = 20;
        String userName = null;
        int money = 0, xiaofei = 0, balance = 0;
        Cookie newCard = null;
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        //1、读取请求头参数信息，得到用户点餐食物类型
        String food = request.getParameter("food");
        //2、读取请求中Cookie
        Cookie[] cookieArray = request.getCookies();
        //3、消费刷卡
        for (Cookie card : cookieArray){
            String key = card.getName();
            String value = card.getValue();
            if("userName".equals(key)){
                userName = value;
            }else if("money".equals(key)){
                money = Integer.valueOf(value);
                if("饺子".equals(food)){
                    if(jiaozi_money > money){
                        out.print("用户" + userName + "余额不足，请充值。");
                    } else {
                        newCard = new Cookie("money", (money - jiaozi_money) + "");
                        xiaofei = jiaozi_money;
                        balance = money - jiaozi_money;
                    }
                } else if("面条".equals(food)){
                    if(miantiiao_money > money){
                        out.print("用户" + userName + "余额不足，请充值。");
                    } else {
                        newCard = new Cookie("money", (money - miantiiao_money) + "");
                        xiaofei = miantiiao_money;
                        balance = money - miantiiao_money;
                    }
                } else if("盖饭".equals(food)){
                    if(gaifan_money > money){
                        out.print("用户" + userName + "余额不足，请充值。");
```

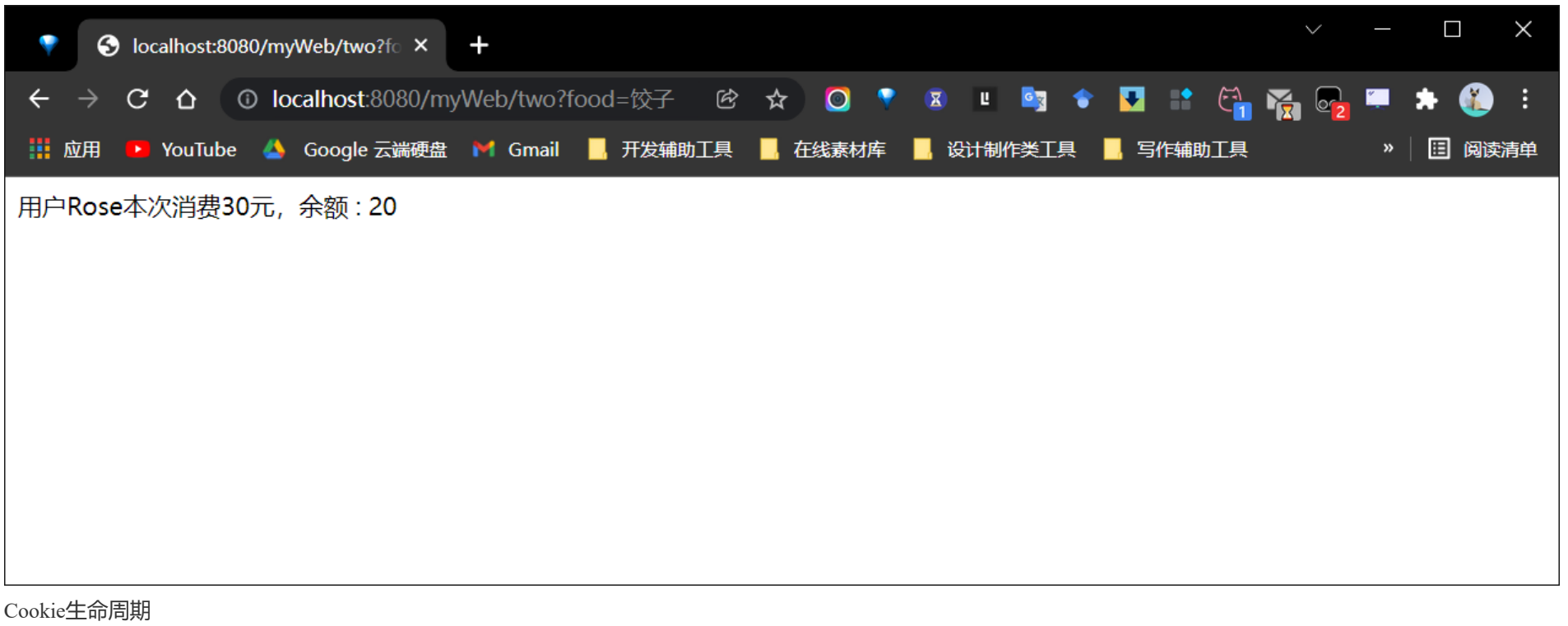
```
                    } else {
                        newCard = new Cookie("money", (money - gaifan_money) + "");
                        xiaofei = gaifan_money;
                        balance = money - gaifan_money;
                    }
                }
            }
        }
        //4、将用户会员卡返还给用户
        response.addCookie(newCard);
        //5、将消费记录写入到响应体
        out.print("用户" + userName + "本次消费" + xiaofei + "元，余额 : " + balance);
    }
}
```

```
<!--index.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
  <center>
    <font style="color: red;font-size: 40px">新会员申请开卡</font>
    <form action="/myWeb/one" method="get">
      <table border="2">
        <tr>
          <td>用户名</td>
          <td><input type="text" name="userName" /></td>
        </tr>
        <tr>
          <td>预存金额</td>
          <td><input type="text" name="money" /></td>
        </tr>
        <tr>
          <td><input type="submit" value="申请开卡" /></td>
          <td><input type="reset" /></td>
        </tr>
      </table>
    </form>
  </center>
</body>
</html>
```

```
<!--index_2.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
  <center>
    <font style="color:red;font-size: 40px">点餐页面</font>
    <form action="/myWeb/two">
      食物类型:
      <input type="radio" name="food" value="饺子">饺子（30元）
      <input type="radio" name="food" value="面条">面条（20元）
      <input type="radio" name="food" value="盖饭">盖饭（15元）<br>
      <input type="submit" value="划卡消费">
    </form>
  </center>
</body>
</html>
```







5. Cookie生命周期

   - 在默认情况下，Cookie对象存放在浏览器的缓存中。因此只要浏览器关闭，Cookie对象就被销毁掉
   - 在手动设置情况下，可以要求浏览器将接收的Cookie存放在客户端计算机上硬盘上，同时需要指定Cookie在硬盘上存活时间。在存活时间范围内，关闭浏览器、关闭客户端计算机、关闭服务器都不会导致Cookie被销毁。在存活时间到达时，Cookie自动从硬盘上被删除

```
cookie.setMaxAge(60); //cookie在硬盘上存活1分钟
```

3、HttpSession接口

1. 介绍

   - HttpSession接口来自于Servlet规范下一个接口。存在于Tomcat中servlet-api.jar，其实现类由Http服务器提供。Tomcat提供实现类存在于servlet-api.jar
   - 如果两个Servlet来自于同一个网站，并且为同一个浏览器/用户提供服务，此时借助于HttpSession对象进行数据共享
   - 开发人员习惯于将HttpSession接口修饰对象称为【会话作用域对象】

2. HttpSession 与 Cookie 区别

   - 存储位置：一个在天上，一个在地下

     Cookie：存放在客户端计算机（浏览器内存/硬盘）
     HttpSession：存放在服务端计算机内存

   - 数据类型：

     Cookie对象存储共享数据类型只能是String
     HttpSession对象可以存储任意类型的共享数据Object

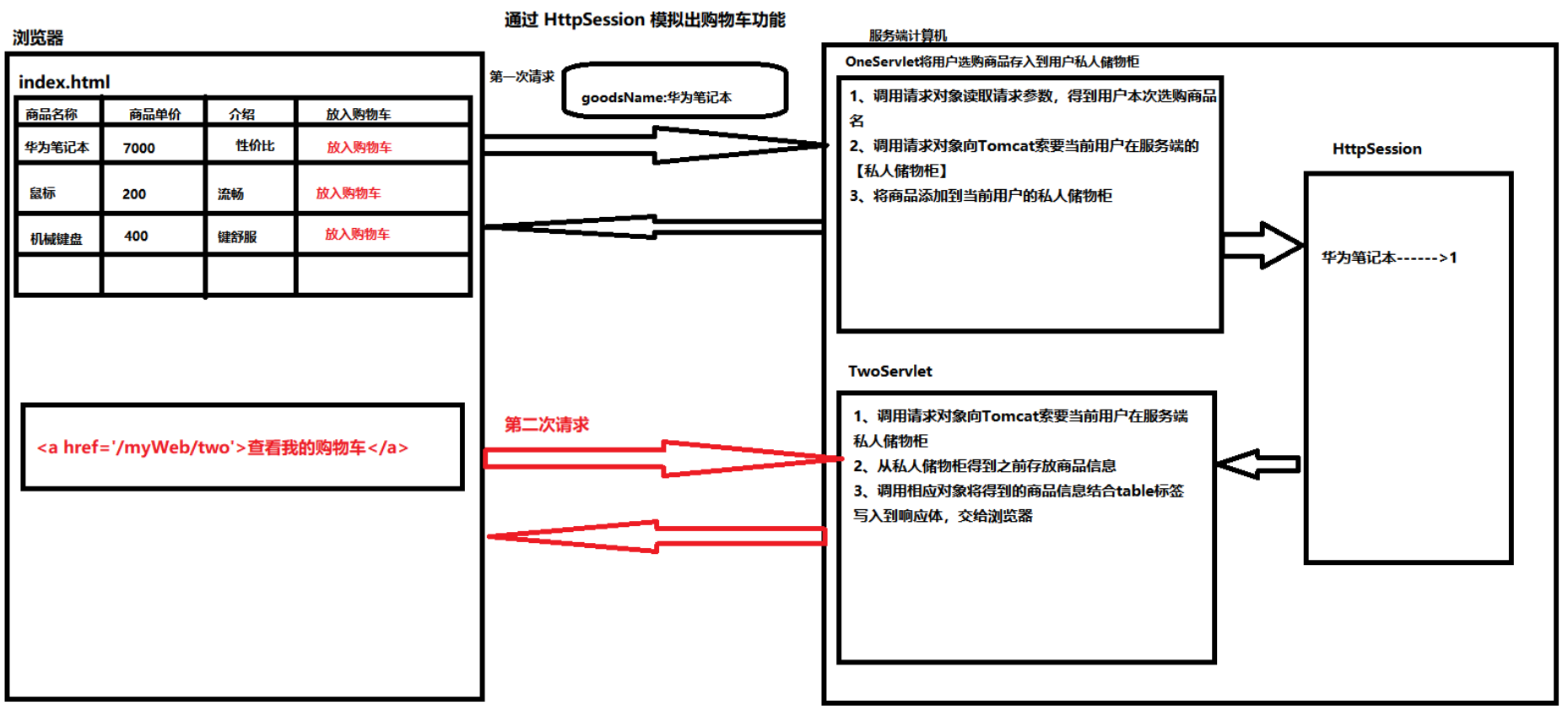
- 数据数量:
  一个Cookie对象只能存储一个共享数据
  HttpSession使用map集合存储共享数据，所以可以存储任意数量共享数据
- 参照物:
  Cookie相当于客户在服务端【会员卡】
  HttpSession相当于客户在服务端【私人保险柜】

3. 命令实现：同一个网站（myWeb）下OneServlet将数据传递给TwoServlet

```
OneServlet{
    public void doGet(HttpServletRequest request,HttpServletResponse response){
        //1.调用请求对象向Tomcat索要当前用户在服务端的私人储物柜
        HttpSession session = request.getSession();
         //2.将数据添加到用户私人储物柜
        session.setAttribute("key1",共享数据)
    }
}
```

```
//浏览器访问/myWeb中TwoServlet
TwoServlet{
    public void doGet(HttpServletRequest request,HttpServletResponse response){
        //1.调用请求对象向Tomcat索要当前用户在服务端的私人储物柜
        HttpSession session = request.getSession();
         //2.从会话作用域对象得到OneServlet提供的共享数据
        Object 共享数据 = session.getAttribute("key1");
    }
}
```

4. 应用实例_模拟购物车功能



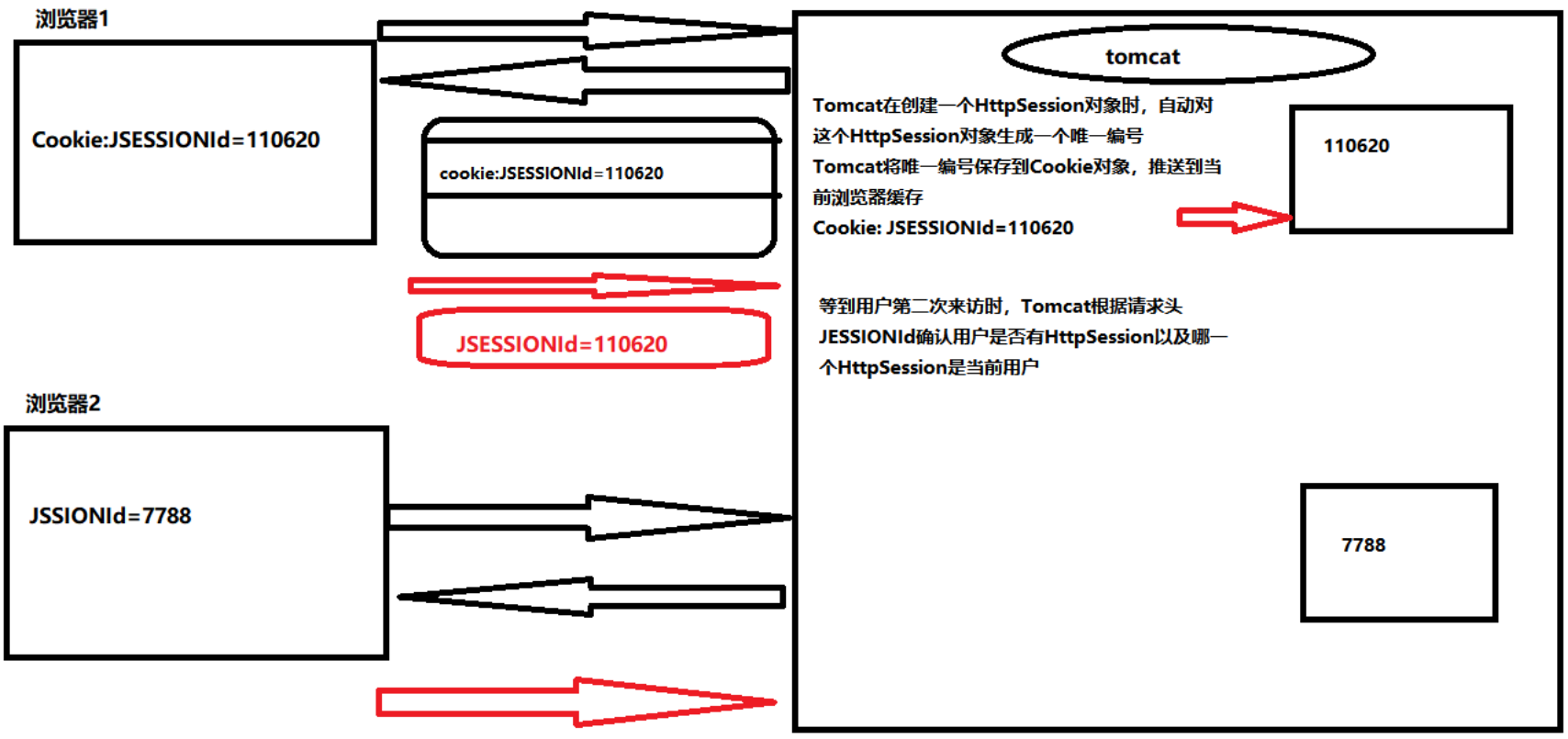

```
public class OneServlet extends HttpServlet {//别名：/one
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //1.调用请求对象，读取请求头参数，得到用户选择商品名
        String goodsName = request.getParameter("goodsName");
        //2.调用请求对象，向Tomcat索要当前用户在服务端的私人储物柜
        HttpSession session = request.getSession();
        //session.setMaxInactiveInterval(5);
        //3.将用户选购商品添加到当前用户私人储物柜
        Integer goodsNum = (Integer)session.getAttribute(goodsName);
        if(goodsNum == null){
            session.setAttribute(goodsName, 1);
        }else{
            session.setAttribute(goodsName, goodsNum+1);
        }
    }
}
```

```
public class TwoServlet extends HttpServlet {//别名：/two
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //1.调用请求对象，向Tomcat索要当前用户在服务端私人储物柜
        HttpSession session = request.getSession();
        //2.将session中所有的key读取出来，存放一个枚举对象
        Enumeration goodsNames = session.getAttributeNames();
        while(goodsNames.hasMoreElements()){
            String goodsName =(String) goodsNames.nextElement();
            int goodsNum = (int) session.getAttribute(goodsName);
            System.out.println("商品名称 "+goodsName+" 商品数量 "+goodsNum);
```

```
        }
    }
}
```

```
<!--index.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
  <table border="2" align="center">
    <tr>
      <td>商品名称</td>
      <td>商品单价</td>
      <td>供货商</td>
      <td>放入购物车</td>
    </tr>
    <tr>
      <td>华为笔记本pro13</td>
      <td>7000</td>
      <td>华为</td>
      <td><a href="/myWeb/one?goodsName=华为笔记本pro13">放入购物车</a></td>
    </tr>
    <tr>
      <td>罗技G304鼠标</td>
      <td>200</td>
      <td>罗技</td>
      <td><a href="/myWeb/one?goodsName=罗技G304鼠标">放入购物车</a></td>
    </tr>
    <tr>
      <td>ikbc机械键盘</td>
      <td>300</td>
      <td>ikbc</td>
      <td><a href="/myWeb/one?goodsName=ikbc机械键盘">放入购物车</a></td>
    </tr>
    <tr>
      <td colspan="4" align="center">
        <a href="/myWeb/two">查看我的购物车</a>
      </td>
    </tr>
  </table>
</body>
</html>
```

5. Http服务器如何将用户与HttpSession关联起来



6. getSession()与getSession(false)
   - getSession()：如果当前用户在服务端已经拥有了自己的私人储物柜，要求tomcat将这个私人储物柜进行返回；如果当前用户在服务端尚未拥有自己的私人储物柜，要求tocmat为当前用户创建一个全新的私人储物柜。
   - getSession(false)：如果当前用户在服务端已经拥有了自己的私人储物柜，要求tomcat将这个私人储物柜进行返回；如果当前用户在服务端尚未拥有自己的私人储物柜，此时Tomcat将返回null。
7. HttpSession生命周期
   - 用户与HttpSession关联时使用的Cookie只能存放在浏览器缓存中；
   - 在浏览器关闭时，意味着用户与他的HttpSession关系被切断；
   - 由于Tomcat无法检测浏览器何时关闭，因此在浏览器关闭时并不会导致Tomcat将浏览器关联的HttpSession进行销毁
   - 为了解决这个问题，Tomcat为每一个HttpSession对象设置【空闲时间】，这个空闲时间默认30分钟，如果当前HttpSession对象空闲时间达到30分钟，此时Tomcat认为用户已经放弃了自己的HttpSession，此时Tomcat就会销毁掉这个HttpSession
8. HttpSession空闲时间手动设置

```
<!--在当前网站/web/WEB-INF/web.xml-->
<session-config>
    <session-timeout>5</session-timeout> <!--当前网站中每一个session最大空闲时间5分钟-->
</session-config>
```

4、HttpServletRequest接口

1. 介绍
   - 在同一个网站中，如果两个Servlet之间通过【请求转发】方式进行调用，彼此之间共享同一个请求协议包。而一个请求协议包只对应一个请求对象；因此servlet之间共享同一个请求对象，此时可以利用这个请求对象在两个Servlet之间实现数据共享
   - 在请求对象实现Servlet之间数据共享功能时，开发人员将请求对象称为【请求作用域对象】
2. 命令实现：OneServlet通过请求转发申请调用TwoServlet时，需要给TwoServlet提供共享数据

```
OneServlet{
    public void doGet(HttpServletRequest req,HttpServletResponse response){
        //1.将数据添加到【请求作用域对象】中attribute属性
        req.setAttribute("key1",数据); //数据类型可以任意类型Object
         //2.向Tomcat申请调用TwoServlet
        req.getRequestDispatcher("/two").forward(req,response)
    }
}
```

```
TwoServlet{
    public void doGet(HttpServletRequest req,HttpServletResponse response){
        //从当前请求对象得到OneServlet写入到共享数据
        Object 数据 = req.getAttribute("key1");
    }
}
```

```
public class OneServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
         //1.将数据添加到请求作用域对象，作为共享数据
          request.setAttribute("key1", "hello World");
         //2.代替浏览器，向Tomcat索要TwoServlet来完成剩余任务
          request.getRequestDispatcher("/two").forward(request, response);
    }
}

public class TwoServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
         // 从同一个请求作用域对象得到OneServlet写入到共享数据
         String value =(String)request.getAttribute("key1");
         System.out.println("TwoServlet得到共享数据 "+value);
    }
```

```
}
```

十三、监听器接口

1. 介绍
   - 一组来自于Servlet规范下接口，共有8个接口。在Tomcat存在servlet-api.jar包
   - 监听器接口需要由开发人员亲自实现，Http服务器提供jar包并没有对应的实现类
   - 监听器接口用于监控【作用域对象生命周期变化时刻】以及【作用域对象共享数据变化时刻】
2. 作用域对象：
   - 在Servlet规范中，认为在服务端内存中可以在某些条件下为两个Servlet之间提供数据共享方案的对象，被称为【作用域对象】
   - Servlet规范下作用域对象:
     1. ServletContext：全局作用域对象
     2. HttpSession：会话作用域对象
     3. HttpServletRequest：请求作用域对象
3. 监听器接口实现类开发规范：三步
   - 根据监听的实际情况，选择对应监听器接口进行实现
   - 重写监听器接口声明【监听事件处理方法】
   - 在web.xml文件将监听器接口实现类注册到Http服务器
4. ServletContextListener接口
   1. 作用：通过这个接口合法的检测全局作用域对象被初始化时刻以及被销毁时刻
   2. 监听事件处理方法：
      - public void contextInitlized(): 在全局作用域对象被Http服务器初始化被调用
      - public void contextDestory(): 在全局作用域对象被Http服务器销毁时候触发调用

```java
package com.example.listener;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
public class OneListener implements ServletContextListener {
    @Override
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("恭喜恭喜，来世走一遭！");
    }
    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("兄弟莫怕，20年后又是一条好汉！");
    }
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <!--将监听器接口实现类注册到Tomcat-->
    <listener>
        <listener-class>com.example.listener.OneListener</listener-class>
    </listener>
</web-app>
```

5. ServletContextAttributeListener接口:
   1. 作用：通过这个接口合法的检测全局作用域对象共享数据变化时刻
   2. 监听事件处理方法：
      - public void contextAdd(): 在全局作用域对象添加共享数据
      - public void contextReplaced(): 在全局作用域对象更新共享数据
      - public void contextRemove(): 在全局作用域对象删除共享数据

```java
package com.example.listener;

import javax.servlet.ServletContextAttributeEvent;
import javax.servlet.ServletContextAttributeListener;

public class OneListener implements ServletContextAttributeListener {
    @Override
    public void attributeAdded(ServletContextAttributeEvent scae) {
        System.out.println("新增共享数据");
    }
    @Override
    public void attributeRemoved(ServletContextAttributeEvent scae) {
        System.out.println("删除共享数据");
    }
    @Override
    public void attributeReplaced(ServletContextAttributeEvent scae) {
        System.out.println("更新共享数据");
    }
```

```
}
```

```
package com.example.controller;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
public class OneServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        ServletContext application = request.getServletContext();
        application.setAttribute("key1",100); //新增共享数据
        application.setAttribute("key1",200); //更新共享数据
        application.removeAttribute("key1");  //删除共享数据
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
      <!--注册监听器接口实现类-->
      <listener>
          <listener-class>com.example.listener.OneListener</listener-class>
      </listener>
    <servlet>
        <servlet-name>OneServlet</servlet-name>
        <servlet-class>com.example.controller.OneServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>OneServlet</servlet-name>
        <url-pattern>/one</url-pattern>
    </servlet-mapping>
</web-app>
```

6. 全局作用域对象共享数据变化时刻

```
ServletContext application = request.getServletContext();
application.setAttribute("key1",100); //新增共享数据
application.setAttribute("key1",200); //更新共享数据
application.removeAttribute("key1");  //删除共享数据
```

7. 在线考试管理系统通过监听器优化

   原因：JDBC规范中，Connection创建与销毁最浪费时间

   解决方案：通过全局作用域对象得到Connection

```
package com.example.listener;

import com.example.util.JdbcUtil;

import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class OneListener implements ServletContextListener {
    @Override
    public void contextInitialized(ServletContextEvent sce) {
        //在Tomcat启动时，预先创建20个Connection，在UserDao.add方法执行时
        //将实现创建好connection交给add方法
        Map<Connection, Boolean> map = new HashMap();
        for(int i = 0; i < 20; i++){
            try {
                Connection conn = JdbcUtil.getConnection();
                System.out.println("在Http服务器启动时，创建Connection" + conn);
                map.put(conn, true);//true表示这个通道处于空闲状态，false表示通道正在被使用
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        //为了在Http服务器运行期间，一直都可以使用20个Connection，将Connection保存
        //到全局作用域对象当中
        ServletContext servletContext = sce.getServletContext();
        servletContext.setAttribute("connections", map);
    }//map被销毁

    @Override
```

```
    public void contextDestroyed(ServletContextEvent sce) {
        //在Http服务器关闭时刻，将全局作用域对象中20个Connection对象销毁
        ServletContext servletContext = sce.getServletContext();
        Map<Connection,Boolean> map = (Map<Connection,Boolean>) servletContext.getAttribute("connections");
        Iterator<Connection> it = map.keySet().iterator();
        while (it.hasNext()){
            Connection conn = it.next();
            if(conn != null){
                try {
                    conn.close();
                    System.out.println(conn + "已销毁! ");
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

```
//JdbcUtil.java
//原则：开闭原则
//通过方法重载来写新的方法

    /**
     * 通过全局作用域对象得到Connection
     * @param request
     * @return
     */
    public static Connection getConnection(HttpServletRequest request) throws SQLException {
        //1、通过请求对象，得到全局作用域对象
        ServletContext application = request.getServletContext();
        //2、从全局作用域对象得到map
        Map<Connection,Boolean> map = (Map<Connection,Boolean>) application.getAttribute("connections");
        //3、从Map得到一个处于空闲状态的Connection
        Iterator<Connection> it = map.keySet().iterator();
        Connection conn = null;
        while(it.hasNext()){
            conn = it.next();
            if(map.get(conn)){
                map.put(conn,false);
                break;
            }
        }
        return conn;
    }

    /**
     * 关闭资源
     * @param conn 连接对象
     * @param ps   数据库操作对象
     * @param rs   结果集
     */
    public static void close(HttpServletRequest request, Connection conn, Statement ps, ResultSet rs){
        if(rs != null){
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(ps != null){
            try {
                ps.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(conn != null){
            ServletContext application = request.getServletContext();
            Map<Connection,Boolean> map = (Map<Connection, Boolean>) application.getAttribute("connections");
            map.put(conn,true);//将Connection通道重新设为空闲状态
        }
    }
```

```
//UserDao.java
//原则：开闭原则
//通过方法重载来写新的方法

    /**
     * 用户注册,全局作用域对象
     * @return 1---注册成功，0---注册失败
     */
    public int add(Users user, HttpServletRequest request){
        //JDBC规范中，Connection创建与销毁最浪费时间
        Connection connection = null;
        PreparedStatement ps = null;
```

```
        int result = 0;
        try {
            connection = JdbcUtil.getConnection(request);
            String sql = "insert into users(userName,password,sex,email)" +
                    "values(?,?,?,?)";
            ps = connection.prepareStatement(sql);
            ps.setString(1, user.getUserName());
            ps.setString(2, user.getPassWord());
            ps.setString(3, user.getSex());
            ps.setString(4, user.getEmail());
            result = ps.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }finally {
            JdbcUtil.close(request, connection, ps, null);
        }
        return result;
    }
.......
```

```
<!--web.xml-->
    <!--注册监听器-->
    <listener>
        <listener-class>com.example.listener.OneListener</listener-class>
    </listener>
```

十四、*Filter (*过滤器接口*)*

1. 介绍
   - 来自于Servlet规范下接口，在Tomcat中存在于servlet-api.jar包
   - Filter接口实现类由开发人员负责提供，Http服务器不负责提供
   - Filter接口在Http服务器调用资源文件之前，对Http服务器进行拦截
2. 具体作用
   - 拦截Http服务器，帮助Http服务器检测当前请求合法性
   - 拦截Http服务器，对当前请求进行增强操作
3. Filter接口实现类开发步骤：三步
   - 创建一个Java类实现Filter接口
   - 重写Filter接口中doFilter()方法
   - web.xml将过滤器接口实现类注册到Http服务器

```
package com.example.filter;

import javax.servlet.*;
import java.io.IOException;
import java.io.PrintWriter;

public class OneFilter implements Filter {
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException,
ServletException {
        //1.通过拦截请求对象得到请求包参数信息，从而得到来访用户的真实年龄
        String age = servletRequest.getParameter("age");
        //2.根据年龄，帮助Http服务器判断本次请求合法性
        if(Integer.valueOf(age) < 70){ //合法请求
            //将拦截请求对象和响应对象交还给Tomcat,由Tomcat继续调用资源文件
            filterChain.doFilter(servletRequest, servletResponse);//放行
        }else{
            //过滤器代替Http服务器拒绝本次请求
            servletResponse.setContentType("text/html;charset=utf-8");
            PrintWriter out = servletResponse.getWriter();
            out.print("<center><font style='color:red;font-size:40px'>大爷，珍爱生命啊!</font></center>");
        }
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <!--将过滤器类文件路径交给Tomcat-->
    <filter>
        <filter-name>oneFilter</filter-name>
        <filter-class>com.example.filter.OneFilter</filter-class>
    </filter>
    <!--通知Tomcat在调用何种资源文件时需要被当前过滤器拦截-->
    <filter-mapping>
        <filter-name>oneFilter</filter-name>
        <url-pattern>/mm.jpg</url-pattern>
```

```
    </filter-mapping>
</web-app>
```

过滤器对请求对象进行增强服务

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
  <center>
    <form action="/myWeb/one" method="post">
      参数:<input type="text" name="userName" /><br/>
      <input type="submit" value="Post方式访问OneServlet">
    </form>
    <form action="/myWeb/two" method="post">
      参数:<input type="text" name="userName" /><br/>
      <input type="submit" value="Post方式访问TwoServlet">
    </form>
  </center>
</body>
</html>
```

```
package com.example.controller;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class OneServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //直接从请求体读取请求参数
        String userName = request.getParameter("userName");
        System.out.println("OneServlet 从请求体得到参数 " + userName);
    }
}
```

```
package com.example.controller;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class TwoServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //直接从请求体读取请求参数
        String userName = request.getParameter("userName");
        System.out.println("TwoServlet 从请求体得到参数 " + userName);
    }
}
```

```
package com.example.filter;

import javax.servlet.*;
import java.io.IOException;

public class OneFilter implements Filter {
    //通知拦截的请求对象，使用UTF-8字符集对当前请求体信息进行一次重新编辑
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException,
ServletException {
        servletRequest.setCharacterEncoding("utf-8");//增强
        filterChain.doFilter(servletRequest, servletResponse);
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
    <servlet>
        <servlet-name>OneServlet</servlet-name>
        <servlet-class>com.example.controller.OneServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>TwoServlet</servlet-name>
        <servlet-class>com.example.controller.TwoServlet</servlet-class>
    </servlet>
    <servlet-mapping>
```

```
        <servlet-name>OneServlet</servlet-name>
        <url-pattern>/one</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>TwoServlet</servlet-name>
        <url-pattern>/two</url-pattern>
    </servlet-mapping>

    <filter>
        <filter-name>oneFilter</filter-name>
        <filter-class>com.example.filter.OneFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>oneFilter</filter-name>
        <url-pattern>/*</url-pattern><!--通知Tomcat在调用所有资源文件之前都需要调用OneFilter进行拦截-->
    </filter-mapping>
</web-app>
```

4. Filter拦截地址格式

- 命令格式:

```
<filter-mapping>
    <filter-name>oneFilter</filter-name>
    <url-pattern>拦截地址</url-pattern>
</filter-mapping>
```

- 命令作用：拦截地址通知Tomcat在调用何种资源文件之前需要调用OneFilter过滤进行拦截
- 要求Tomcat在调用某一个具体文件之前，来调用OneFilter拦截

```
 <url-pattern>/img/mm.jpg</url-pattern>
```

- 要求Tomcat在调用某一个文件夹下所有的资源文件之前，来调用OneFilter拦截

```
<url-pattern>/img/*</url-pattern>
```

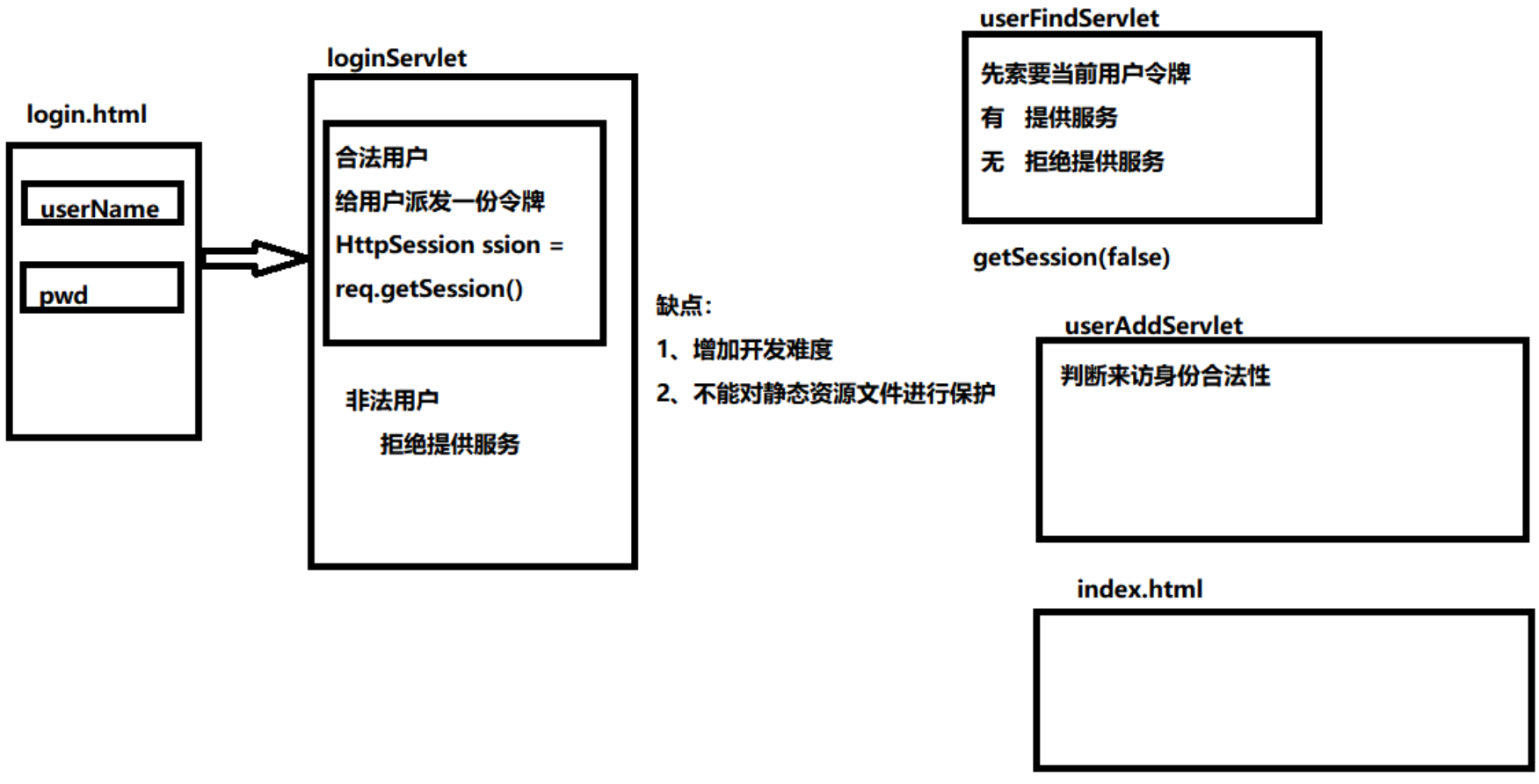
- 要求Tomcat在调用任意文件夹下某种类型文件之前，来调用OneFilter拦截

```
<url-pattern>*.jpg</url-pattern>
```

- 要求Tomcat在调用网站中任意文件时，来调用OneFilter拦截

```
<url-pattern>/*</url-pattern>
```

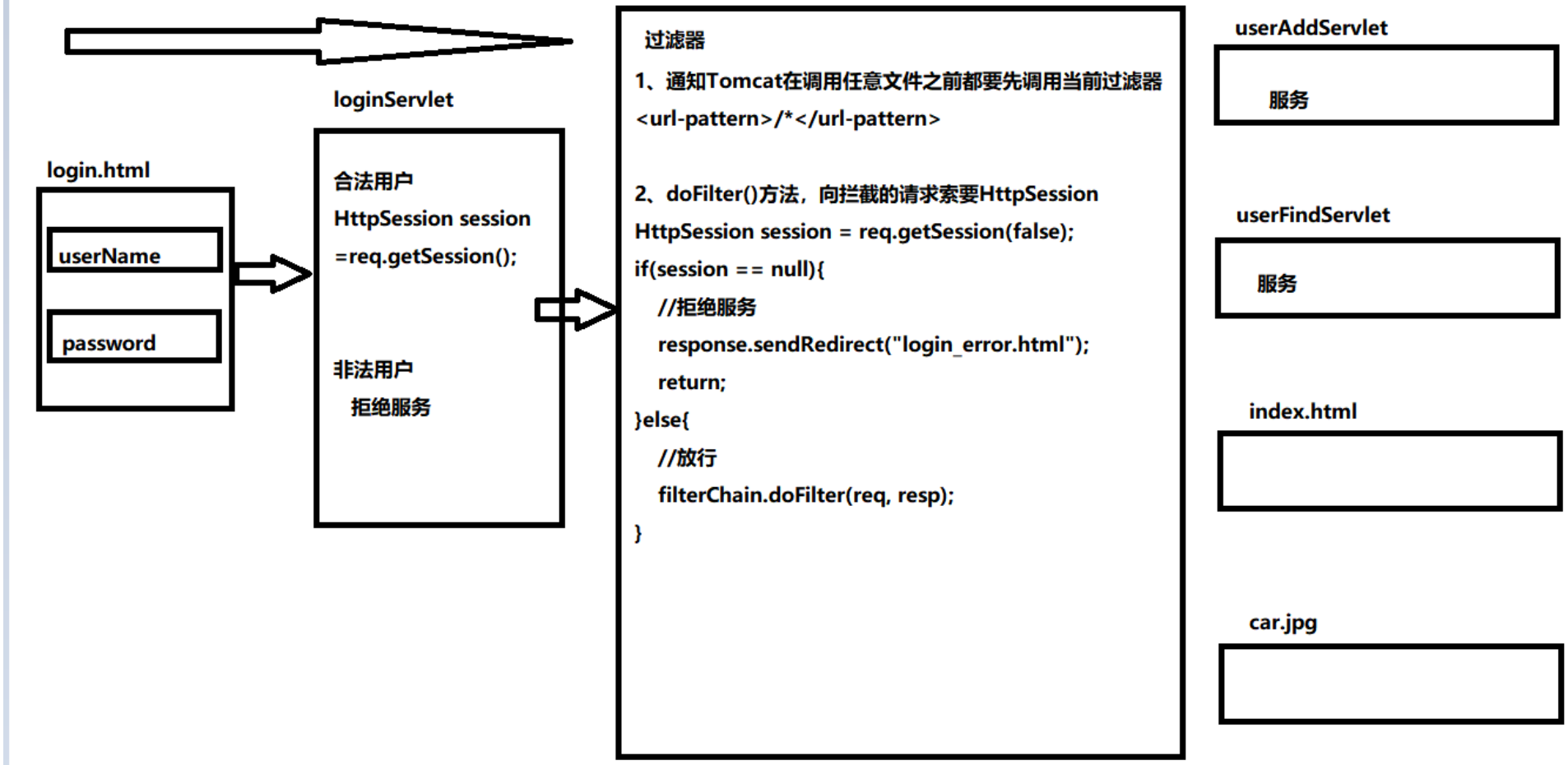
5. 在线考试管理系统_过滤器防止用户恶意登录行为




```
public class LoginServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // 1.调用请求对象对请求体使用utf-8字符集进行重新编辑
        request.setCharacterEncoding("utf-8");
        // 2.调用请求对象读取请求体参数信息
        String userName = request.getParameter("userName");
        String password = request.getParameter("password");
        // 3.调用UserDao将查询验证信息推送到数据库服务器上
        UserDao userDao = new UserDao();
```

```
        int result = userDao.ligin(userName, password, request);
        //4.调用响应对象，根据验证结果将不同资源文件地址写入到响应头，交给浏览器
        if(result == 1){
            //在判定来访用户身份合法后，通过请求对象向Tomcat申请为当前用户申请一个HttpSession
            HttpSession session = request.getSession();
            response.sendRedirect("/myWeb/index.html");
        } else {
            response.sendRedirect("/myWeb/login_error.html");
        }
    }
}
```

```
public class UserFindServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //索要当前用户在服务端HttpSession
        HttpSession session = request.getSession(false);
        if(session == null){
            response.sendRedirect("/myWeb/login_error.html");
            return;
        }

        //提供服务
        //1、[调用UserDao] 将查询命令推送到数据库服务器上，得到所有用户信息【List】
        UserDao userDao = new UserDao();
        List<Users> allUsers = userDao.findAll(request);
        //2、[调用响应对象] 将用户信息结合<table>标签命令以二进制形式写入到响应体
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        out.print("<table border='2' align='center' >");
        out.print("<tr>");
        out.print("<td>用户编号</td>");
        out.print("<td>用户姓名</td>");
        out.print("<td>用户密码</td>");
        out.print("<td>用户性别</td>");
        out.print("<td>用户邮箱</td>");
        out.print("<td>操作</td>");
        out.print("</tr>");
        for(Users user : allUsers){
            out.print("<tr>");
            out.print("<td>" + user.getUserId() +"</td>");
            out.print("<td>" + user.getUserName() +"</td>");
            out.print("<td>******</td>");
            out.print("<td>" + user.getSex() +"</td>");
            out.print("<td>" + user.getEmail() +"</td>");
            out.print("<td><a href='/myWeb/user/delete?userId=" + user.getUserId() + "' >删除用户</a></td>");
            out.print("</tr>");
        }
        out.print("</table>");
    }
}
```



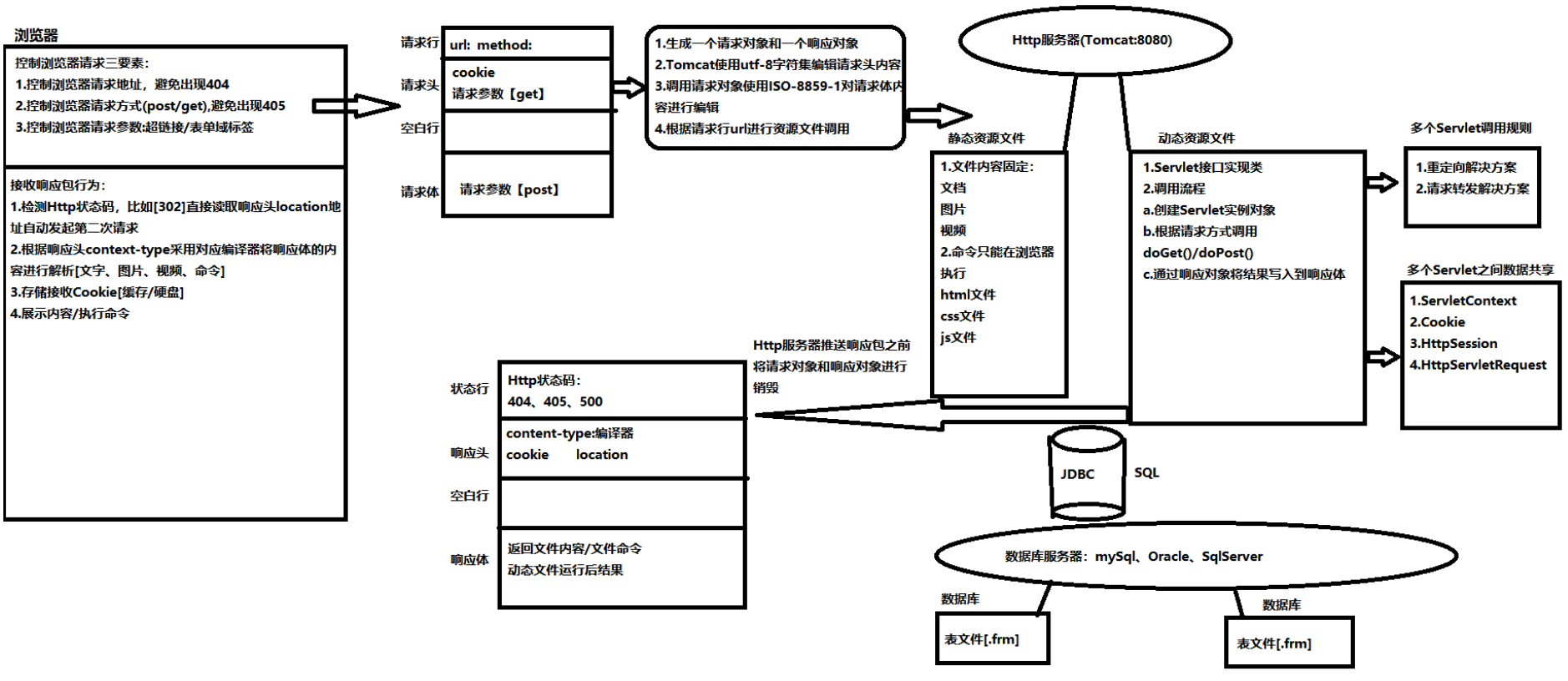

```
public class LoginServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // 1.调用请求对象对请求体使用utf-8字符集进行重新编辑
        request.setCharacterEncoding("utf-8");
        // 2.调用请求对象读取请求体参数信息
```

```
        String userName = request.getParameter("userName");
        String password = request.getParameter("password");
        // 3.调用UserDao将查询验证信息推送到数据库服务器上
        UserDao userDao = new UserDao();
        int result = userDao.ligin(userName, password, request);
        //4.调用响应对象，根据验证结果将不同资源文件地址写入到响应头，交给浏览器
        if(result == 1){
            //在判定来访用户身份合法后，通过请求对象向Tomcat申请为当前用户申请一个HttpSession
            HttpSession session = request.getSession();
            response.sendRedirect("/myWeb/index.html");
        } else {
            response.sendRedirect("/myWeb/login_error.html");
        }
    }
}
```

```
package com.example.filter;

import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;

public class OneFilter implements Filter {
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException,
ServletException {
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        //1.调用请求对象读取请求包中请求行URI，了解用户访问的资源文件是谁
        String uri = request.getRequestURI();//[/网站名/资源文件名 /myWeb/login.html 或 /myWeb/login 或 ...]
        //2.如果本次请求资源文件与登录相关【login.html 或者 LoginServlet】 此时应该无条件放行
        // /myWeb/ 表示初始打开Tomcat时传进来的参数
        if(uri.indexOf("login") != -1 || "/myWeb/".equals(uri)){
            filterChain.doFilter(servletRequest, servletResponse);
            return;
        }
        //3.如果本次请求访问的是其他资源文件，需要得到用户在服务端的HttpSession
        HttpSession session = request.getSession(false);
        if(session != null){
            filterChain.doFilter(servletRequest, servletResponse);
            return;
        }
        //4.拒绝请求
        request.getRequestDispatcher("/login_error.html").forward(servletRequest, servletResponse);
    }
}
```

```
    <!--注册过滤器-->
    <filter>
        <filter-name>oneFilter</filter-name>
        <filter-class>com.example.filter.OneFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>oneFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

十五、第三版互联网通信流程图


浏览器
控制浏览器请求三要素:
1.控制浏览器请求地址，避免出现404
2.控制浏览器请求方式(post/get),避免出现405
3.控制浏览器请求参数:超链接/表单域标签
接收响应包行为:
1.检测Http状态码，比如[302]直接读取响应头location地址自动发起第二次请求
2.根据响应头context-type采用对应编译器将响应体的内容进行解析[文字、图片、视频、命令]
3.存储接收Cookie[缓存/硬盘]
4.展示内容/执行命令
请求行
请求头
空白行
请求体
url: method:
cookie
请求参数【get】
请求参数【post】
1.生成一个请求对象和一个响应对象
2.Tomcat使用utf-8字符集编辑请求头内容
3.调用请求对象使用ISO-8859-1对请求体内容进行编辑
4.根据请求行url进行资源文件调用
Http服务器(Tomcat:8080)
静态资源文件
1.文件内容固定:
文档
图片
视频
2.命令只能在浏览器执行
html文件
css文件
js文件
动态资源文件
1.Servlet接口实现类
2.调用流程
a.创建Servlet实例对象
b.根据请求方式调用
doGet()/doPost()
c.通过响应对象将结果写入到响应体
多个Servlet调用规则
1.重定向解决方案
2.请求转发解决方案
多个Servlet之间数据共享
1.ServletContext
2.Cookie
3.HttpSession
4.HttpServletRequest
Http服务器推送响应包之前将请求对象和响应对象进行销毁
状态行
响应头
空白行
响应体
Http状态码:
404、405、500
content-type:编译器
cookie location
返回文件内容/文件命令
动态文件运行后结果
JDBC
SQL
数据库服务器：mySql、Oracle、SqlServer
数据库
表文件[.frm]
数据库
表文件[.frm]