

request&response

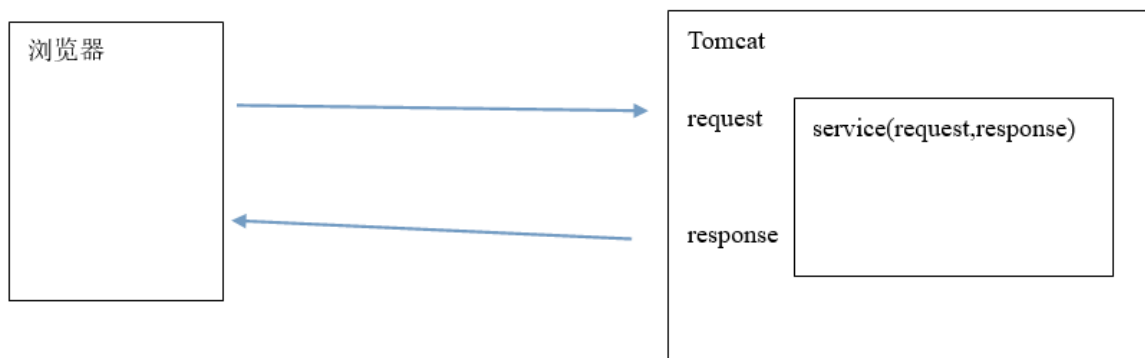
学习目标

- 1.能够使用Response对象操作HTTP响应内容
- 2.能够处理响应乱码
- 3.能够完成文件下载案例
- 4.能够使用Request对象获取HTTP协议请求内容
- 5.能够处理HTTP请求参数的乱码问题
- 6.能够使用Request域对象
- 7.能够使用Request对象做请求转发
- 8.能够完成登录案例

小提示: 学好这2个对象的关键是--理解HTTP协议

Web服务器收到客户端的http请求，会针对每一次请求

分别创建一个用于代表请求的request对象和代表响应的response对象。



第1章 response对象操作响应信息

1.1 response对象概述

response对象原形是`javax.servlet.http.HttpServletResponse`，它继承自`javax.servlet.ServletResponse`，我们先看下官方文档截图：

javax.servlet.http

Interface HttpServletResponse

All Superinterfaces:

[ServletResponse](#)

All Known Implementing Classes:

[HttpServletResponseWrapper](#)

```
public interface HttpServletResponse
extends ServletResponse
```

Implements: [ServletResponse](#)

Implemented by: [HttpServletResponseWrapper](#)

扩展 [ServletResponse](#) 接口以提供特定于 HTTP 的发送响应功能。例如，该接口拥有访问 HTTP 头和 cookie 的方法。

servlet 容器创建 `HttpServletResponse` 对象，并将该对象作为参数传递给 servlet 的 service 方法（doGet、doPost，等等）。

总结：HttpServletResponse对象封装了向客户端发送数据、发送响应头，发送响应状态码的方法。开发人员必须学会使用HttpServletResponse，才能给浏览器发送数据。

1.2 response操作响应行

响应行的组成

响应行是http响应内容的第一行。

响应行一般数据为：HTTP/1.1 200 (tomcat8.5) 或者 HTTP/1.1 200 OK (tomcat7)

响应行分为三个部分： HTTP/1.1: 协议版本

200: 响应状态码

OK: 对响应状态码的解释

常见的响应状态码：

200 OK 请求已成功，请求所希望的响应头或数据体将随此响应返回。出现此状态码是表示正常状态。

302 Move temporarily 重定向，请求的资源临时从不同的 **URI**响应请求。

304 Not Modified 从缓存中读取数据，不从服务器重新获取数据。

403 Forbidden 服务器已经理解请求，但是拒绝执行它，一般在权限不够的时候常见。

404 Not Found 请求失败，请求所希望得到的资源未被在服务器上发现。

405 Method Not Allowed 请求行中指定的请求方法不能被用于请求相应的资源。

500 Internal Server Error 服务器遇到了一个未曾预料的状态，导致了它无法完成对请求的处理。

以上就是我们常用的响应状态码，了解了http协议有关响应行的部分，对于response对象来说，只有响应状态码可以操作，并且我们一般也不操作状态码。

API介绍

```
void setStatus(int sc) 设置响应的状态代码(一般用来设置 1xx 2xx 3xx)
void sendError(int sc) 设置响应的状态代码(一般用来设置 4xx 5xx)
```

注意：状态码的一般不需要我们手动设置

1.3 response操作响应头

常见的响应头介绍

响应头有很多，我们先来看自己大家必须知道的响应头

1. location :

重定向操作：通常告知浏览器马上向该地址发送请求，通常和响应码302 一起使用

2. refresh :

定时刷新操作，指定时间后跳转到指定页面

3. content-encoding :

设置当前数据的压缩格式，告知浏览器以何种压缩格式解压数据

4. content-disposition :

通知浏览器以何种方式获取数据（直接解析数据（网页，图片文本），或者以附件方式（下载文件））

5. content-type :

实体头部用于指示资源的MIME类型（MIME类型：用于提示当前文件的媒体类型，例如图片为：image/png、音频为:audio/ogg）。它的作用与传统上Windows上的文件扩展名相同。该名称源于最初用于电子邮件的MIME标准。）

注意：我们content-type常用的设置一般都是“text/html; charset=utf-8”，其中“text/html”用来设置浏览器以指定文件格式解析数据；“charset=utf-8”用来响应数据的编码表，若不需要设置编码可以不写。

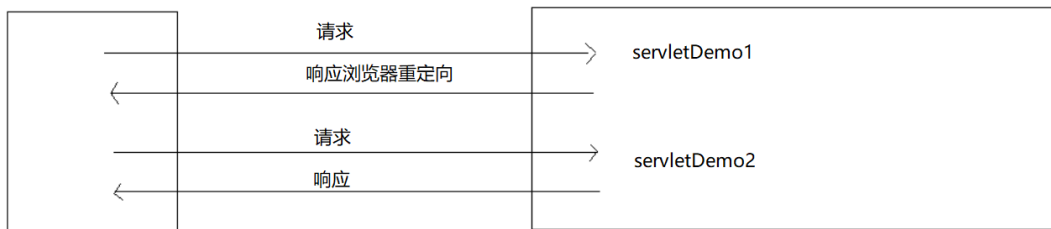
API 介绍

```
void setHeader(String name, String value) 用给定名称和值设置响应头
void sendRedirect(String location) 用类设置重定向
```

接下来我们通过几个案例，让大家来练习使用下方法和响应头。

案例1：重定向--使用location响应头实现跳转

重定向概念



案例需求

使用location响应头实现重定向跳转淘宝主页www.taobao.com

案例分析

1. 创建servlet
2. 使用response对象，发送location消息头和302响应码给浏览器

代码实现

```
package cn.itcast.web;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class RedirectServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //设置重定向响应头
        // response.setHeader("location","http://www.taobao.com");
        //设置状态码
        //response.setStatus(302);
        //=====上面代码的简化方式=====
        response.sendRedirect("http://www.taobao.com");
    }
}
```

案例2：3秒钟之后跳转到其他页面

案例需求

在当前页面停留3秒钟之后跳转到京东首页

案例分析

1. 创建RefreshServlet
2. 调用setHeader，设置消息头（"Refresh"," 3;url=<http://www.jd.com>"）

代码实现

```
package cn.itcast.web;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class RefreshServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //自动刷新操作，在3秒后跳转京东主页
        response.setHeader("Refresh", " 3;url=http://www.jd.com");
    }
}
```

另外几个响应头，我们需要学习完response操作响应体之后才可以操作。

1.4 response操作响应体

响应体页面上的要展示的html的代码了

API介绍

`ServletOutputStream` `getOutputStream()` 获取字节输出流
`PrintWriter` `getWriter()` 获取字符输出流

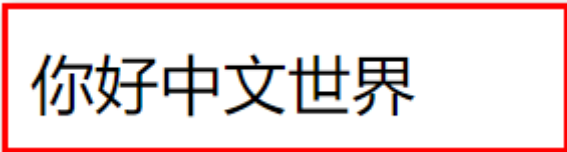
注意:两个流不能同时使用(互斥),这两个流使用完成后如果没有关闭,Servlet容器会帮我们将其关闭

案例1：向浏览器输出中文数据(无乱码)

案例需求

向页面输出中文数据没有乱码

案例效果



你好中文世界

案例分析

1. 创建servlet
2. 使用response对象，调用setContentType方法传入参数：“text/html;charset=utf-8”

3. 使用response对象，向页面输出“你好中文世界”

代码实现

```
package cn.itcast.web;

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

@WebServlet(name = "EncodingServlet", urlPatterns = "/encoding")
public class EncodingServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        response.getWriter().write("你好中文世界");
    }
}
```

案例2：以附件形式下载文件

案例需求

完成文件下载功能。

案例效果

用户点击页面的链接，浏览器开始下载文件。

案例分析

1. 创建一个页面，展示所有要被下载文件的链接
2. 链接将要下载的文件名称，发送给服务器的servlet，让servlet进行处理
3. 服务器加载文件资源
4. 提示浏览器，以下载的方式，获取服务器资源
5. 使用IO的方式，将文件数据输出到浏览器（response.getOutputStream()）

代码实现

1. html页面

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<a href="/day03/download?fileName=1.jpg">1.jpg</a>
<a href="/day03/download?fileName=2.txt">2.txt</a>
</body>
</html>

```

2. servlet演示代码

```

package cn.itcast.web;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import cn.itcast.utils.DownloadUtils;

//urlPatterns = "/download"
public class DownloadServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //1 获取要下载的文件名称
        String fileName = request.getParameter("fileName");
        System.out.println(fileName);
        //2 加载当前文件
        // 注意：需要动态的获取当前文件的目录位置（即使服务器所在目录发生变化，我也可以获
        取到准确位置）
        // 我们需要使用ServletContext，最后一节课讲解，获取资源路径
        ServletContext context = getServletContext();
        String realPath = context.getRealPath("/download");
        File file = new File(realPath, fileName);

        //3 提示浏览器，以下载的方式，获取服务器资源
        //响应消息头设置：
        //Content-Type 设置文件媒体格式    getMimeType: 1.txt 2.jpg 获取文件的后
        缀名
        response.setContentType(getServletContext().getMimeType(fileName));

        //4 处理中文文件名乱码问题（仅做了解，实际项目开发中下载文件不会使用中文名称）
        // 获取浏览器类型，通过请求头中的User-Agent来判断

```

```

        fileName = DownloadUtils.getName(request.getHeader("user-agent"),
        fileName);

        //Content-Disposition 设置要被下载的文件名
        response.setHeader("Content-Disposition", "attachment;filename=" +
        fileName);
        //5 将指定文件使用IO技术, 向浏览器输出
        FileInputStream in = new FileInputStream(file);
        ServletOutputStream out = response.getOutputStream();

        //6 标准IO代码
        byte[] buf = new byte[1024];
        int len = -1;
        while((len = in.read(buf)) != -1) {
            out.write(buf, 0, len);
        }
        in.close();
    }
}

```

案例3：点击切换验证码(扩展)

案例需求

在页面展示登录验证码,点击此验证码可以更换新的验证码

案例分析

1. 创建一个登录页面, 展示验证码图片
2. 配置今天资料中提供的servlet输出验证码图片
3. 设置页面的点击事件, 触发点击事件就重新获取验证码图片

代码实现

1. html页面

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="/day03/login" method="post">
    <table>
        <tr>
            <td>用户名</td>
            <td><input type="text" name="username"/></td>
        </tr>
        <tr>
            <td>密码</td>
            <td><input type="password" name="password"/></td>
        </tr>
    </table>

```



```

        <tr>
            <td>验证码: </td>
            <td><input type="password" name="checkcode"/></td>
        </tr>
        <tr>
            <td></td>
            <td></td>
        </tr>
        <tr>
            <td></td>
            <td><input type="submit" value="登录"/></td>
        </tr>
    </table>
</form>
</body>
</html>

```

2.1 (四个字符)复制验证码servlet,不要忘记在web.xml中配置

```

package cn.itcast.web;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.util.Random;

import javax.imageio.ImageIO;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
//urlPatterns = "/checkcode"
public class CheckcodeServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // 创建画布
        int width = 120;
        int height = 40;
        BufferedImage bufferedImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        // 获得画笔
        Graphics g = bufferedImage.getGraphics();
        // 填充背景颜色
        g.setColor(Color.white);
        g.fillRect(0, 0, width, height);
        // 绘制边框
        g.setColor(Color.red);
        g.drawRect(0, 0, width - 1, height - 1);
        // 生成随机字符
        // 准备数据

```

```

String data =
"ABCDEFGHGIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890";
// 准备随机对象
Random r = new Random();
// 声明一个变量 保存验证码
String code = "";
// 书写4个随机字符
for (int i = 0; i < 4; i++) {
    // 设置字体
    g.setFont(new Font("宋体", Font.BOLD, 28));
    // 设置随机颜色
    g.setColor(new Color(r.nextInt(255), r.nextInt(255),
r.nextInt(255)));

    String str = data.charAt(r.nextInt(data.length())) + "";
    g.drawString(str, 10 + i * 28, 30);

    // 将新的字符 保存到验证码中
    code = code + str;
}
// 绘制干扰线
for (int i = 0; i < 6; i++) {
    // 设置随机颜色
    g.setColor(new Color(r.nextInt(255), r.nextInt(255),
r.nextInt(255)));

    g.drawLine(r.nextInt(width), r.nextInt(height), r.nextInt(width),
r.nextInt(height));
}

// 将验证码 打印到控制台
System.out.println(code);

// 将验证码放到session中
request.getSession().setAttribute("code_session", code);

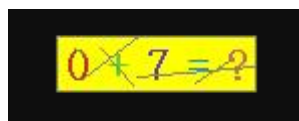
// 将画布显示在浏览器中
ImageIO.write(bufferedImage, "jpg", response.getOutputStream());
}

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

}
...

```

2. 2 或者使用 (数学运算验证码)复制验证码servlet ,别忘记在web.xml中配置



```

package com.itheima.servlet.response;

import javax.imageio.ImageIO;

```

```

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.util.Random;

    //urlPatterns = "/checkcode"
public class CodeServlet extends HttpServlet {

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

        // 使用java图形界面技术绘制一张图片

        int charNum = 5; //5个字符例如 1+2=3
        int width = 20 * 5;
        int height = 28;

        // 1. 创建一张内存图片
        BufferedImage bufferedImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);

        // 2. 获得绘图对象
        Graphics graphics = bufferedImage.getGraphics();

        // 3、绘制背景颜色
        graphics.setColor(Color.YELLOW);
        graphics.fillRect(0, 0, width, height);

        // 4、绘制图片边框
        graphics.setColor(Color.GRAY);
        graphics.drawRect(0, 0, width - 1, height - 1);

        // 5、设置字体颜色和属性
        graphics.setColor(Color.RED);
        graphics.setFont(new Font("宋体", Font.BOLD, 22));

        // 随机对象
        Random random = new Random();

        int n1 = random.nextInt(10); //第一个数字
        int n2 = random.nextInt(10); //第二个数字
        int m = random.nextInt(2)+1; //运算符12代表加减
        String msg = ""; // session中要用到
        String code = ""; //图片中的字符

        switch (m) {
            case 1: msg = String.valueOf(n1+n2); code=""+n1+"+"+n2+"=?"; break; //和
            case 2: msg = String.valueOf(n1>n2?n1-n2:n2-n1); code=(n1>n2)?
(""+n1+"-"+n2+"=?"):(""+n2+"-"+n1+"=?"); break; //大数减去小数的差
        }

        System.out.println("n1 = " + n1);
        System.out.println("n2 = " + n2);

```

```

System.out.println("m = " + m);
System.out.println("code = " + code);
System.out.println("msg = " + msg);

int x = 5; //坐标从5像素开始
for (int i = 0; i < code.length(); i++) { //画验证码图片
    String content = String.valueOf(code.charAt(i));
    graphics.setColor(new Color(random.nextInt(255),
random.nextInt(255), random.nextInt(255)));
    graphics.drawString(content, x, 22);
    x += 20; //右移动20像素
}

// 6、绘制干扰线
graphics.setColor(Color.GRAY);
for (int i = 0; i < 5; i++) {
    int x1 = random.nextInt(width);
    int x2 = random.nextInt(width);

    int y1 = random.nextInt(height);
    int y2 = random.nextInt(height);
    graphics.drawLine(x1, y1, x2, y2);
}

// 释放资源
graphics.dispose();

// 图片输出 ImageIO
ImageIO.write(bufferedImage, "jpg", response.getOutputStream());

}

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

    }

}

```

页面js代码

3. 设置页面点击事件的js代码

```
````javascript
<script>

 window.onload = function () {
 var img = document.getElementById("img");
 img.onclick=function () {
 img.src="/day03/checkcode?r="+new Date().getTime();
 };
 };
</script>
````
```

第2章 request对象获取请求信息

2.1 request对象的基本概念

1. 什么是HttpServletRequest?

HttpServletRequest对象代表客户端的请求，当客户端通过HTTP协议访问服务器时，HTTP请求中的所有信息都封装在这个对象中，开发人员通过这个对象的方法，可以获得客户这些信息。

以下是API文档中的介绍：

```
javax.servlet.http
Interface HttpServletRequest

All Superinterfaces:
    ServletRequest

All Known Implementing Classes:
    HttpServletRequestWrapper

public interface HttpServletRequest
    extends ServletRequest

Implements: ServletRequest
Implemented by: HttpServletRequestWrapper

扩展 javax.servlet.ServletRequest 接口，为 HTTP servlet 提供请求信息。

servlet 容器创建 HttpServletRequest 对象，并将该对象作为参数传递给 servlet 的 service 方法（doGet、doPost，等等）。
```

通过文档阅读可以注意到一个细节javax.servlet.http.HttpServletRequest不是相关请求的顶级接口，它继承自父接口javax.servlet.ServletRequest：

```
javax.servlet
Interface ServletRequest

All Known Subinterfaces:
    HttpServletRequest

All Known Implementing Classes:
    HttpServletRequestWrapper, ServletRequestWrapper

public interface ServletRequest

Implemented by: HttpServletRequest, ServletRequestWrapper

定义将客户端请求信息提供给某个 servlet 的对象。servlet 容器创建 ServletRequest 对象，并将该对象作为参数传递给该 servlet 的 service 方法。
```

通过Request对象进行的常用操作:

- 1) 获取请求数据
- 2) 作为域对象
- 3) 请求转发

2. HttpServlet有许多的API我们从何学起?

答: 我们按照学习http请求组成部分, 按请求行、请求头、请求体顺序学习。

2.2 request获取请求行信息

请求行的组成元素

在http协议中我已经看到了http协议中请求行的内容——分为请求方式、请求路径、协议版本。

在HttpServletRequest概述中我们知道浏览器与请求相关的数据封装在request中, 因此, 接下来我们学习如何使用request对象获取请求行的数据。

API介绍

```
String getMethod()    获取请求方式的类型    如: GET或POST
StringBuffer getRequestURL()  获取客户端发出请求完整URL
String getRequestURI()  获取请求行中的资源名部分
String getContextPath()  获取当前项目根路径
String getProtocol()    获取当前协议的名称和版本
String getRemoteAddr()  获取IP地址
int getLocalPort()     获取端口
```

注:

uri: 统一资源标识符, 用来标识一个资源, 资源路径。(相当于身份证)

url: 统一资源定位符, 是一种具体的URI, 可以用来标识一个资源, 并且指明了如何定位一个资源 (相当于身份证的地址)

如:

http://localhost:9090/day02/demo

url: http://localhost:9090/day02/demo

uri: /day02/demo

使用步骤

1. 创建DemoServlet
2. 在DemoServlet中的doGet或者doPost方法的参数列表, 已经包含了request对象, 调用方法即可。
3. 将数据打印在控制台

演示代码

```
package cn.itcast.web;

import javax.servlet.ServletException;
```

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

@WebServlet(name = "DemoServlet",urlPatterns = "/demo")
public class DemoServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request,response);
    }

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

        /**
         * 1.    String getMethod()
                获取请求方式的类型
         */
        String method = request.getMethod();
        System.out.println("获取请求方式的类型:"+method);

        /**
         * 2.    StringBuffer getRequestURL()
                获取客户端发出请求完整URL
         */
        StringBuffer getRequestURL = request.getRequestURL();
        System.out.println("获取客户端发出请求完整URL:"+getRequestURL());

        /**
         * 3.    String getRequestURI()
                获取请求行中的资源名部分
         */
        String requestURI = request.getRequestURI();
        System.out.println("获取请求行中的资源名部分:"+requestURI);

        /**
         * 4.    String getContextPath()
                获取当前项目路径
         */
        String contextPath = request.getContextPath();
        System.out.println(contextPath);

        /**
         * 5.    String getProtocol()
                获取当前协议的名称和版本
         */
        String getProtocol = request.getProtocol();
        System.out.println("获取当前协议的名称和版本:"+getProtocol);

        /**
         * 6.    String getRemoteAddr()
                获取IP地址
         */
        String getRemoteAddr = request.getRemoteAddr();
        System.out.println("获取IP地址:"+getRemoteAddr);
    }
}
```

```

    /**
     * 7.    int getLocalPort()
           获取端口
     */
    int localPort = request.getLocalPort();
    System.out.println("获取端口:"+localPort);

}
}

```

效果一：测试地址——<http://localhost:9090/day02/demo>

获取请求方式的类型:GET
 获取客户端发出请求完整URL:<http://localhost:9090/day02/demo>
 获取请求行中的资源名部分: /day02/demo
 获取当前项目根路径: /day02
 获取当前协议的名称和版本:HTTP/1.1
 获取IP地址:0:0:0:0:0:0:0:1
 获取端口:9090

效果二：测试地址——<http://127.0.0.1:9090/day02/demo>

获取请求方式的类型:GET
 获取客户端发出请求完整URL:<http://localhost:9090/day02/demo>
 获取请求行中的资源名部分: /day02/demo
 获取当前项目根路径: /day02
 获取当前协议的名称和版本:HTTP/1.1
 获取IP地址:127.0.0.1
 获取端口:9090

问：为什么要测试两次？

答：同学们观察下两次打印的IP地址会发现不一样，因此，注意，Localhost和127.0.0.1效果一致，但是localhost默认使用ipv6本机地址——0:0:0:0:0:0:0:1，而127.0.0.1是ipv4的本机地址。

2.3 request获取请求头信息

获取请求头信息常用的方法

API介绍

`String getHeader(String name)` 以String的形式返回指定请求头的值
`Enumeration getHeaderNames()` 返回此请求包含的所有头名称的枚举

使用步骤

1. 创建DemoServlet2
2. 在DemoServlet2中的doGet或者doPost方法的参数列表，已经包含了request对象。因此，调用方法即可。

3. 将结果打印在控制台

演示代码

```
package cn.itcast.web;

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

@WebServlet(name = "DemoServlet2", urlPatterns = "/demo2")
public class DemoServlet2 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request, response);
    }

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

        /**
         * 1. String getHeader(String name)
            以 String 的形式返回指定请求头的值
         */
        String header = request.getHeader("User-Agent");
        System.out.println("getHeader:"+header);
        System.out.println();
        /**
         * 2. Enumeration getHeaderNames()
            返回此请求包含的所有头名称的枚举
         */
        Enumeration<String> headerNames = request.getHeaderNames();
        while (headerNames.hasMoreElements()){
            System.out.println("getHeaderNames:"+headerNames.nextElement());
        }
    }
}
```

效果:

```
getHeader:Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36
```

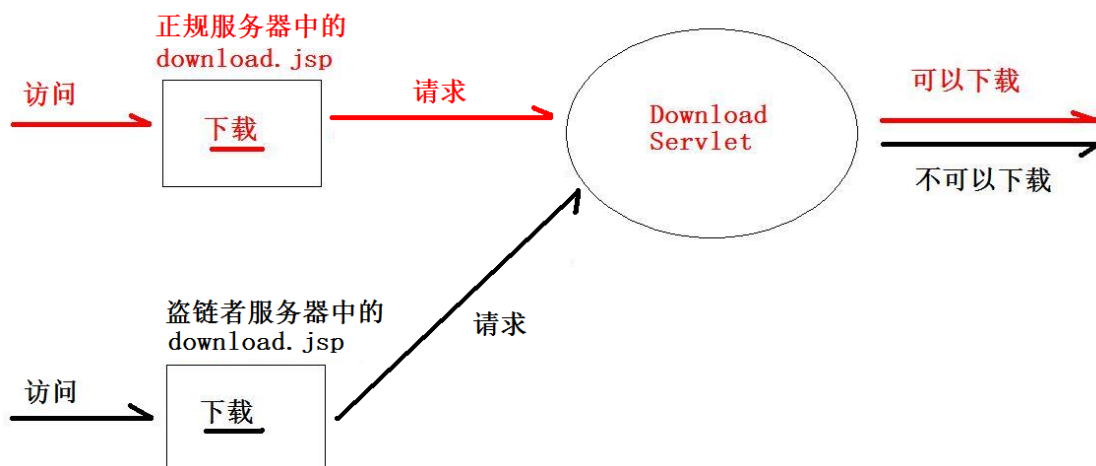
```
getHeaderNames:host  
getHeaderNames:connection  
getHeaderNames:cache-control  
getHeaderNames:user-agent  
getHeaderNames:upgrade-insecure-requests  
getHeaderNames:accept  
getHeaderNames:accept-encoding  
getHeaderNames:accept-language  
getHeaderNames:cookie
```

案例：使用referer实现防盗链

案例需求

1. 问：什么是盗链？

答：如下图所示。



2. 问：如何防止盗链？

答：在上图中用户一共发送两次请求，第一次访问正规服务器中的download.jsp页面，第二次访问盗链者服务器中的download.jsp页面，两个download.jsp页面虽然内容相同，但来源不相同。

如果download.jsp中的请求，来源于盗链者服务器，我们就显示不可以下载；

如果download.jsp中的请求，来源于正规服务器，我们就显示可以下载；

案例效果

两次请求同一个域名，显示可以下载

两次请求不同域名，显示无法下载

案例分析

1. 创建一个DownloadServlet。
2. 使用request对象的getHeader方法获取referer请求头信息。
3. 通过referer判断请求的来源地址，判断是否与当前项目统一。

实现步骤

1. servlet演示代码:

```
/**
 * 使用referer实现防盗链
 * 用户->download.jsp->DownloadServlet
 */
@WebServlet(name = "DownloadServlet", urlPatterns = "/DownloadServlet")
public class DownloadServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        this.doGet(request, response);
    }

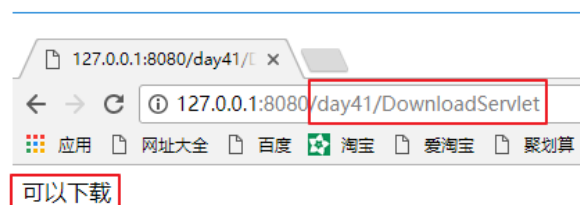
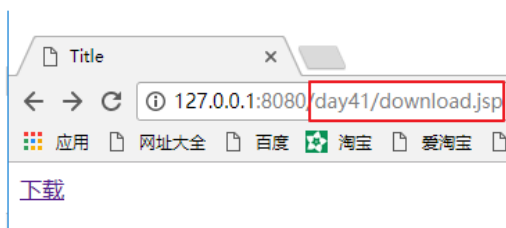
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //处理响应中文乱码-明天讲解
        response.setContentType("text/html;charset=UTF-8");
        //获取输出流 往页面上写内容-明天讲解
        PrintWriter writer = response.getWriter();

        //获取请求头referer
        String referer = request.getHeader("referer");
        //如果请求头referer存在, 且请求来源于正规服务器的download.jsp页面的话
        if(referer != null &&
            referer.equals("http://127.0.0.1:8080/day41/download.jsp")){
            //没有盗链, 在浏览器中显示可以下载
            writer.write("可以下载");
        }else{
            //请求来源于盗链者服务器的download.jsp页面的话
            writer.write("这是盗链, 不可以下载");
        }

        writer.flush();
        writer.close();
    }
}
```

2. 测试:

第一次访问正规服务器中的download.jsp页面, 并发出下载请求, 正规服务器中的DownloadServlet通过验证referer的来源是否合理, 这次下载请求来源合理, 所以显示“可以下载”。



测试:

第二次访问盗链者服务器中的download.jsp页面，并发出下载请求，
正规服务器中的DownloadServlet通过验证referer的来源是否合理，这次下载请求来源不合理，
所以显示“这是盗链，不可以下载”。



案例：获取用户当前使用的浏览器版本

案例需求

获取用户当前使用的浏览器版本

案例效果

当前用户浏览器相关信息: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36

案例分析

1. 创建UserAgentServlet
2. 调用request的getHeader方法，获取消息头User-Agent
3. 打印在控制台上

实现步骤

1. servlet演示代码:

```
package cn.itcast.web;

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

@WebServlet(name = "UserAgentServlet", urlPatterns = "/userAgent")
public class UserAgentServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String header = request.getHeader("User-Agent");
        System.out.println("当前用户浏览器相关信息: "+header);
    }
}
```

2.4 request获取请求体数据

学习完了对请求行和请求头的内容，最后一部分就是请求体了，在请求体中，包含的是用户通过浏览器发送的请求参数，因此，我们主要学习的就是获取请求参数的方法。

获取请求参数使用方法

API介绍

`String getParameter(String name)` 根据表单的name属性 获取对应的值

`String[] getParameterValues(String name)` 获取name相同的所有value 例如复选框。

`Map getParameterMap()` 请求参数名作为key，参数值作为value，封装到map中。

使用步骤

1. 准备html页面：regist.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Title</title>
</head>
<body>
<form action="/day02/regist" method="post">
    用户名: <input type="text" name="username"><br>
    爱 好: <input type="checkbox" name="hobby" value="football">足球
    <input type="checkbox" name="hobby" value="basketball">篮球<br>
    <input type="submit" value="提交">
</form>
</body>
</html>
```

2. 创建GetParameterServlet
3. 在GetParameterServlet中的doGet和doPost方法的参数列表中，已经包含了request对象，调用相应方法即可。

演示代码

```
package cn.itcast.web;

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

```

@WebServlet(name = "GetParameterServlet",urlPatterns = "/getParam")
public class GetParameterServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request,response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        /**
         * 1.    String    getParameter(String name)
                根据表单的name属性 获取对应的值
         * */
        String username = request.getParameter("username");
        System.out.println(username);
        /**
         * 2.    String[]    getParameterValues(String name)
                获取name相同的所有value 例如复选框。
         * */
        String[] hobbies = request.getParameterValues("hobby");
        for (String hobby : hobbies) {
            System.out.println(hobby);
        }

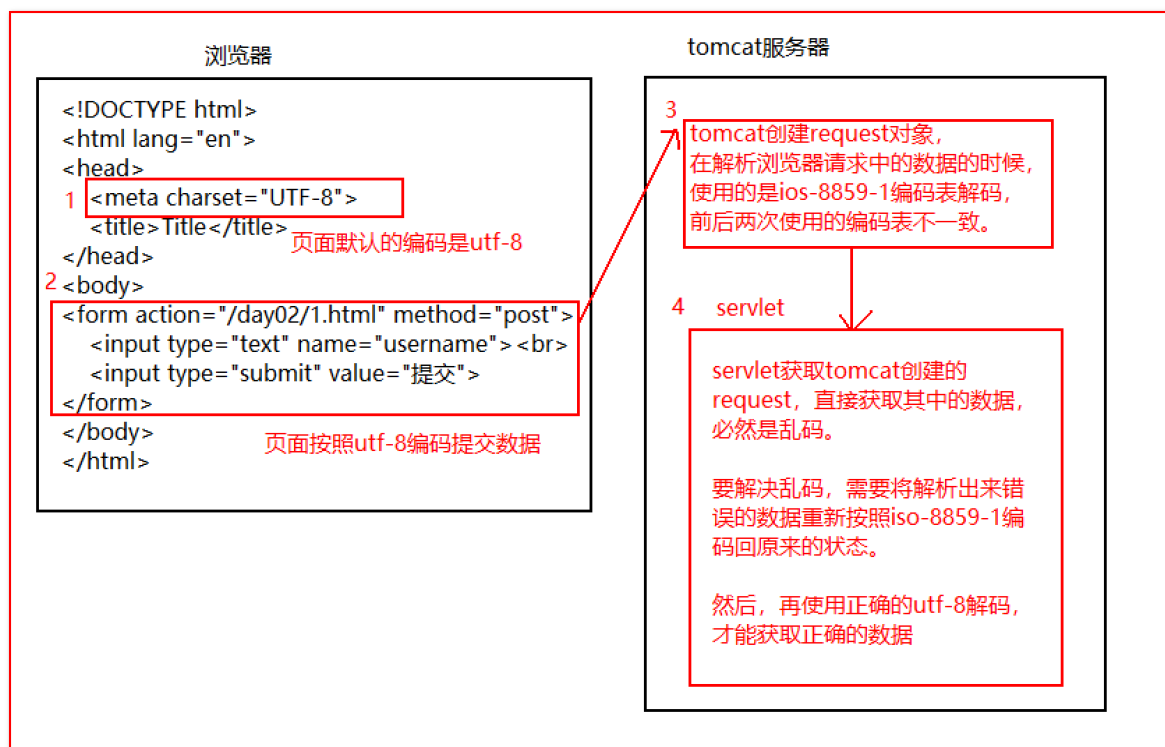
        /**
         * 3.    Map    getParameterMap()
                参数名作为key，参数值作为value，封装到map中。
         * */
        Map<String, String[]> map = request.getParameterMap();
        for (String key : map.keySet()) {
            for (String s : map.get(key)) {
                System.out.println(s);
            }
        }
    }
}

```

乱码解决

请求参数乱码的由来

我们在输入一些中文数据提交给服务器的时候，服务器解析显示出来的一堆无意义的字符，就是乱码。那么这个乱码是如何出现的呢？如下图所示：



有乱码那么必须处理乱码，不同的请求方式处理乱码操作不同。

API介绍

1. `void setCharacterEncoding(String env)`
设置请求体的编码

使用步骤

1. 创建EncodingServlet
2. 在EncodingServlet的doPost或者doGet方法中第一行，调用setCharacterEncoding方法设置编码
3. 然后获取请求参数

注意事项

1. 获取请求参数之后，调用setCharacterEncoding方法无效

演示代码

```
package cn.itcast.web;

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

@WebServlet(name = "EncodingServlet", urlPatterns = "/encoding")
```

```

public class EncodingServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request, response);
    }

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

        //处理post请求乱码
        request.setCharacterEncoding("utf-8");
        String username = request.getParameter("username");
        System.out.println(username);
    }
}

```

案例：使用BeanUtils封装表单提交的数据到javaBean对象中

案例需求

现在我们已经可以使用request对象来获取请求参数，但是，如果参数过多，我们就需要将数据封装到对象。以前封装数据的时候，实体类有多少个字段，我们就需要手动编码调用多少次setXXX方法，因此，我们需要BeanUtils来解决这个问题。

案例效果

使用BeanUtils，完成数据的封装到实体类。

案例分析

1. 设置一个登录页面准备提交表单数据（username、password）
2. 导入BeanUtils相关jar包
3. 创建Servlet获取请求参数
4. 调用BeanUtils.populate方法封装数据

实现步骤

1. 准备登录页面：

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="/day02/login" method="post">
    <table>
        <tr>
            <td>用户名</td>
            <td><input type="text" name="username"/></td>
        </tr>
        <tr>
            <td>密码</td>

```

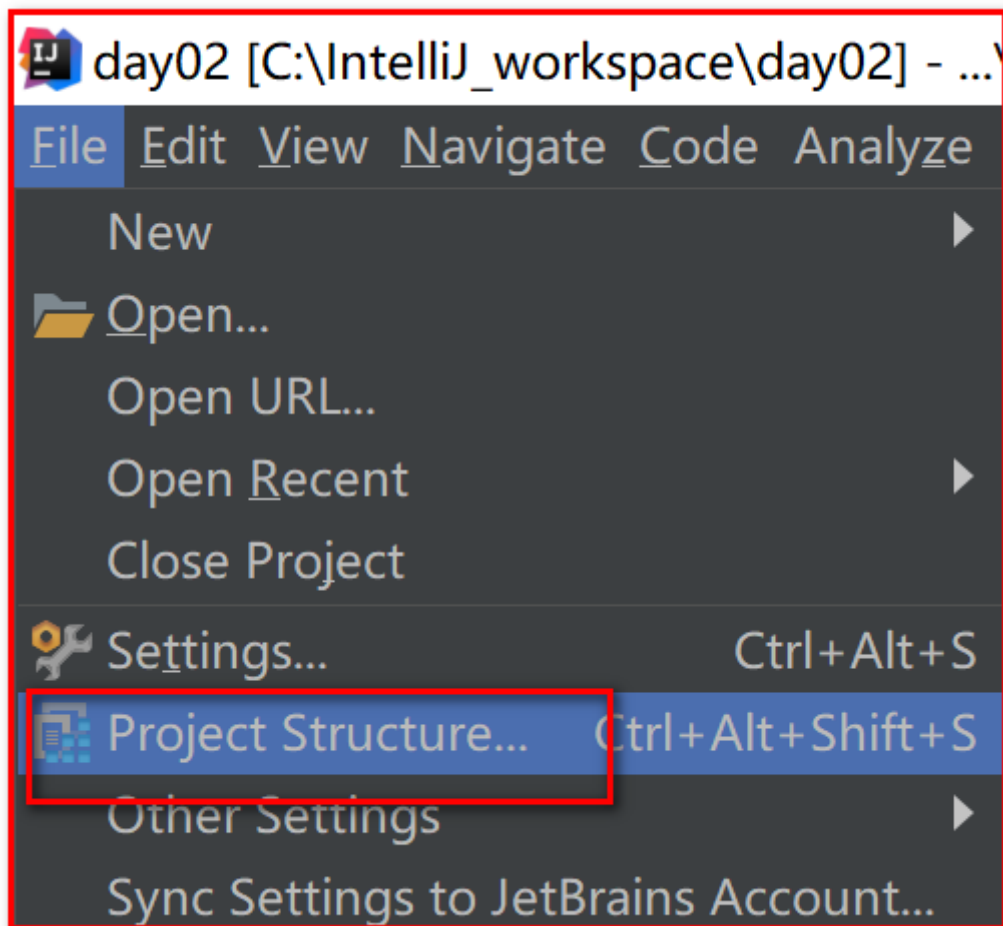


```
<td><input type="password" name="password"/></td>
</tr>
<tr>
<td></td>
<td><input type="submit" value="登录"/></td>
</tr>
</table>
</form>
</body>
</html>
```

2. 在web目录下创建WEB-INF\lib文件夹，导入BeanUtils相关jar包

```
commons-beanutils-1.8.3.jar
commons-logging-1.1.1.jar
```

3. 注意：导入完成还要关联jar包到项目




```

package cn.itcast.domain;

public class User {

    private int id;
    private String username;
    private String password;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    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;
    }
    @Override
    public String toString() {
        return "User [id=" + id + ", username=" + username + ", password=" +
password + "]\n";
    }
}

```

5. servlet代码：封装表单数据到User对象

```

package cn.itcast.web;

import cn.itcast.domain.User;
import org.apache.commons.beanutils.BeanUtils;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.util.Map;

@WebServlet(name = "LoginServlet",urlPatterns = "/login")
public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request,response);
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

```

```

//获取请求参数
Map<String, String[]> map = request.getParameterMap();
//创建要封装数据的对象
User user = new User();
//封装前打印一次数据
System.out.println("封装数据前: "+user);
try {
    //封装数据
    BeanUtils.populate(user, map);
} catch (Exception e) {
    e.printStackTrace();
}

//封装后打印一次数据
System.out.println("封装数据后: "+user);
}
}

```

效果:

```

User [id=0, username=null, password=null]
User [id=0, username=tom, password=123]

```

2.5 request的其他作用（域对象和转发）

域对象

问：什么是域对象？

答：域对象是一个容器，这种容器主要用于servlet与servlet之间的数据传输使用的。

今天讲解的request域对象,就可以在**一次请求中的多个servlet之间**进行数据共享。

域对象的API介绍

1. `void setAttribute(String name, Object o)` 设置数据到域中
2. `Object getAttribute(String name)` 从域中获取数据
3. `void removeAttribute(String name)` 从域中移除数据

域对象的使用

1. 创建ScopeServlet
2. 调用request对象存（setAttribute）取（getAttribute）删（removeAttribute）方法
3. 在保存和删除方法调用完成之后，都使用获取方法获取数据，打印在控制台上

注意事项

以上三个方法都是操作request域对象中的数据，与请求参数无关。

演示代码

```
package cn.itcast.web;

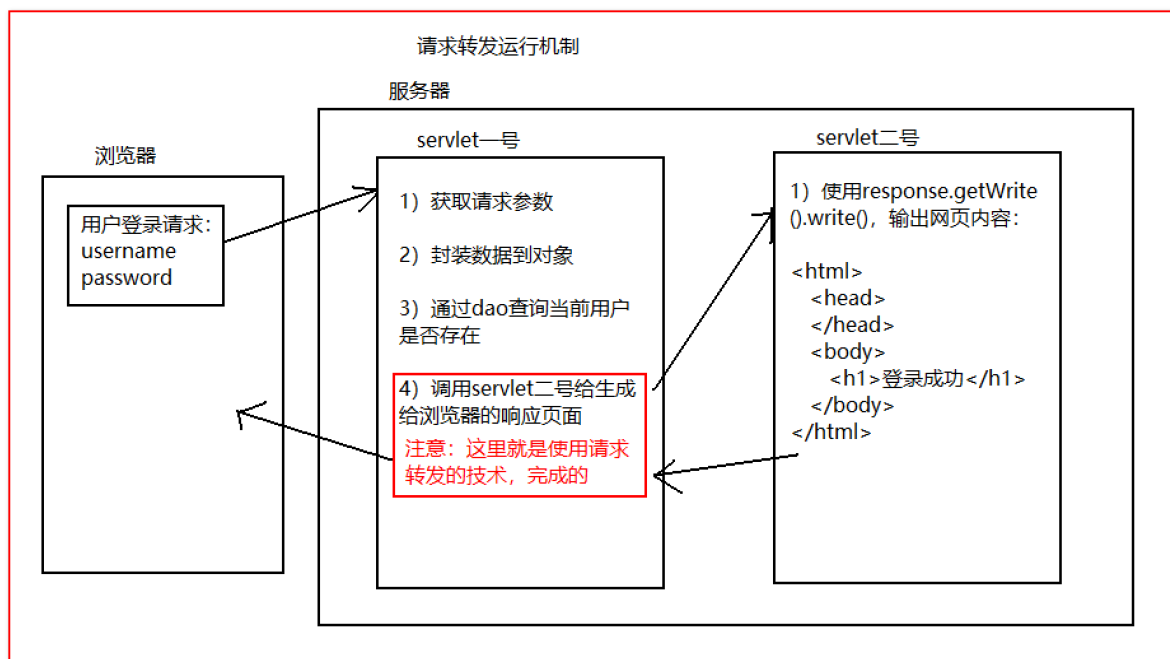
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(name = "ScopeServlet", urlPatterns = "/scope")
public class ScopeServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //存入数据
        request.setAttribute("name", "张三");
        //获取数据
        String name1 = (String) request.getAttribute("name");
        System.out.println(name1);
        //移除数据
        request.removeAttribute("name");
        String name2 = (String) request.getAttribute("name");
        System.out.println(name2);
    }
}
```

请求转发

什么是请求转发？



API介绍

1. `RequestDispatcher` `getRequestDispatcher(String path)` 获取请求转发器（`request`对象方法）
2. `void forward(ServletRequest request, ServletResponse response)` 将请求转发到另一个资源（`servlet`）上（`RequestDispatcher`对象的方法）

使用步骤

1. 先通过请求对象获取转发器
2. 再调用转发器转发方法，转发请求

演示代码

1. `DispatcherServlet`:

```
package cn.itcast.web;

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

@WebServlet(name = "DispatcherServlet", urlPatterns = "/dispatcher")
public class DispatcherServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("对用户请求第一次处理");
        request.setAttribute("result", "test_data");
        request.getRequestDispatcher("/test").forward(request, response);
    }
}
```

2. `TestServlet`:

```
package cn.itcast.web;

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

```
import java.io.IOException;

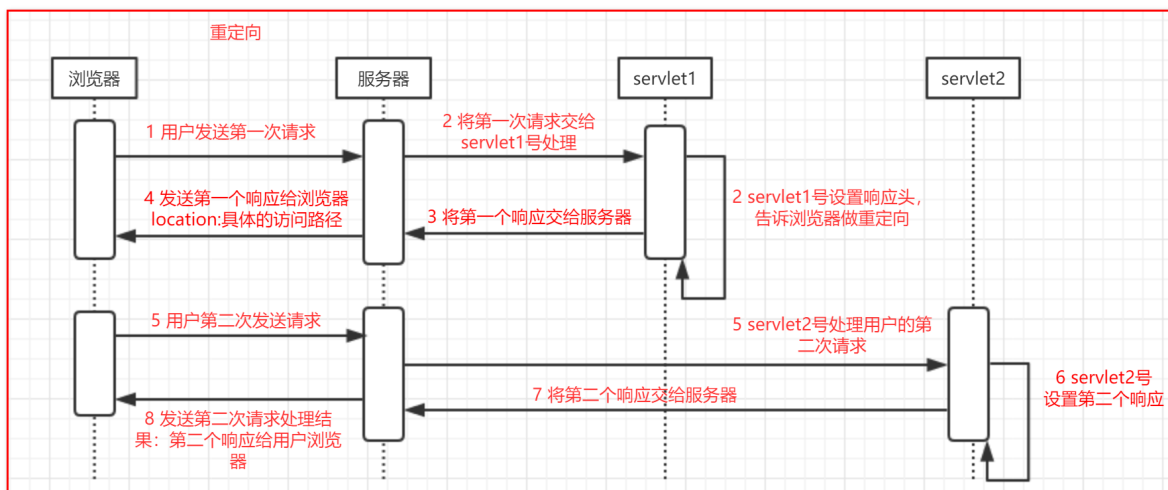
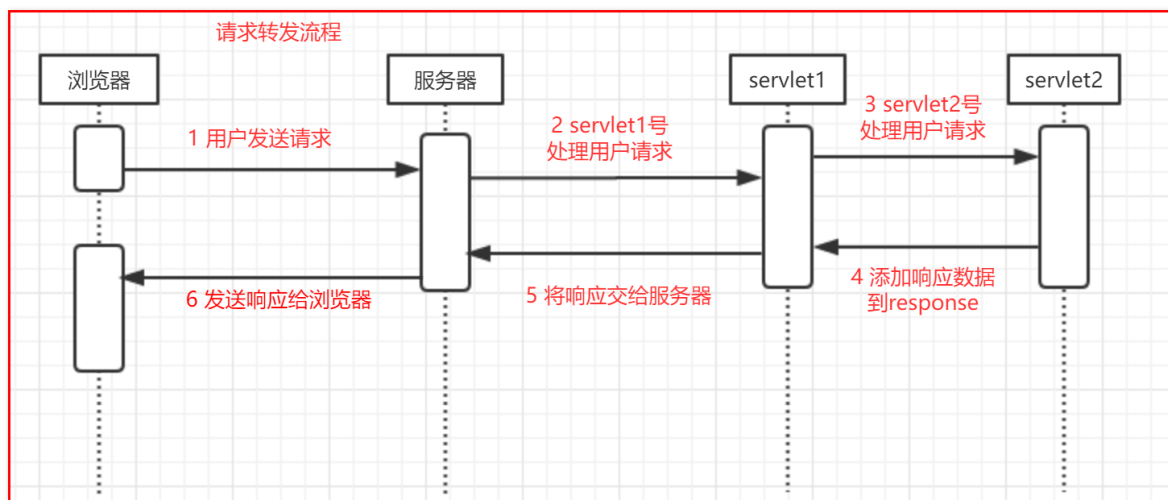
@WebServlet(name = "TestServlet",urlPatterns = "/test")
public class TestServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        System.out.println("对用户请求第二次处理");
        String result = (String) request.getAttribute("result");
        response.getWriter().write(result);
    }
}
```

请求转发与请求重定向的区别

完成重定向操作之后，同学们心里有一个疑问：之前学习的转发和现在的重定向都可以完成跳转，那么他们之间有什么区别呢？

我们将转发和重定向的流程制作了时序图给大家展示：



通过上图，我们可以简单总结几点转发和重定向的区别：

1. 转发在一次请求中完成，重定向是两次请求
2. 转发操作发生在服务器内部，重定向是浏览器执行操作
3. 转发地址栏不变（只有一次请求，一个地址），重定向，地址栏变化（两次请求，两个地址）
4. 转发可以在一次请求中共享数据，重定向不行（重定向两次请求，每次都会创建新的request对象）。