

Request&Response-授课

1 响应对象

1.1 响应对象概述

1.1.1 关于响应

响应，它表示了服务器端收到请求，同时也已经处理完成，把处理的结果告知用户。简单来说，指的就是服务器把请求的处理结果告知客户端。在B/S架构中，响应就是把结果带回浏览器。

响应对象，顾名思义就是用于在JavaWeb工程中实现上述功能的对象。

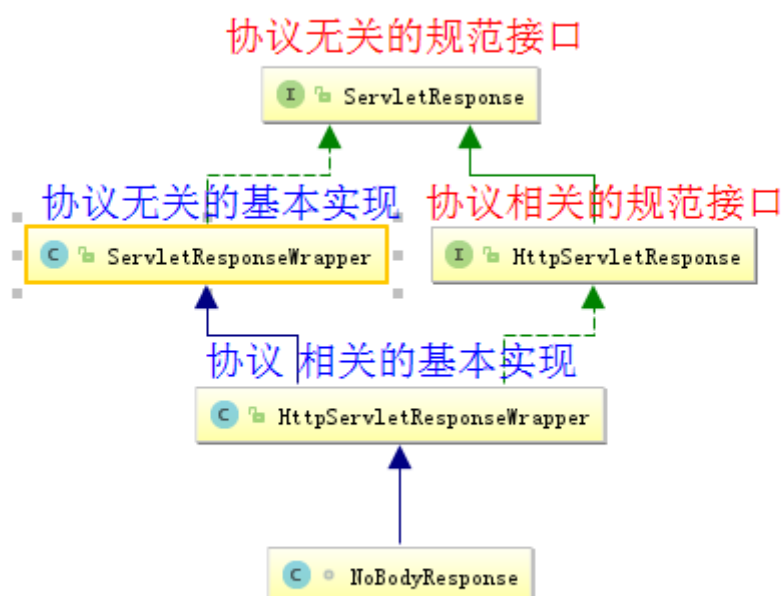
1.1.2 常用响应对象

响应对象也是是Servlet规范中定义的，它包括了协议无关的和协议相关的。

协议无关的对象标准是：ServletResponse接口

协议相关的对象标准是：HttpServletResponse接口

类结构图如下：



我们课程中涉及的响应对象都是和HTTP协议相关的。即使用的是HttpServletResponse接口的实现类。

这里有些同学可能会产生疑问，我们在使用Servlet时，需要定义一个类，然后实现Servlet接口（或者继承它的实现类）。现在我们想要实现响应功能，要不要定义一个类，然后实现HttpServletResponse接口呢？

此问题的答案是否定的，我们无需这么做。我们只需要在自己写的Servlet中直接使用即可，因为这个对象的实现类是由Tomcat提供的，无须我们自定义。同时它还会帮我们对象创建出来并传入doGet和doPost方法中。

1.2 常用方法介绍

在HttpServletResponse接口中提供了很多方法，接下来我们通过API文档，来了解一下这些方法。

Method Summary

All Methods	Instance Methods	Abstract Methods	Default Methods	Deprecated Methods	
Modifier and Type		Method and Description			
void		addCookie(Cookie cookie) 在响应中添加一个指定的cookie (cookie是会话管理技术，我们将在明天讲解)			
void		addDateHeader(String name, long date) 添加日期类型的响应消息头。name是头的名称。date是日期的毫秒值，从1970年开始			
void		addHeader(String name, String value) 添加字符串类型的响应消息头			
void		addIntHeader(String name, int value) 添加整型的响应消息头			
boolean		containsHeader(String name) 判断是否包含指定名称的消息头			
String		encodeRedirectUrl(String url) 重定向，指的是把浏览器地址重新定向到一个新的地址。我们稍后在案例中演示。 Deprecated. 和会话相关的知识点，我们都放到明天来讲解。 URL重写方法，用于重定向目的地禁用Cookie的会话保持。从2.1版本标注过时。用encodeRedirectURL方法代替。			
String		encodeRedirectURL(String url) URL重写方法，用于重定向目的地禁用Cookie的会话保持。			
String		encodeUrl(String url) Deprecated. URL重写方法，用于浏览器禁用Cookie的会话保持。从2.1版本标注过时。用encodeURL代替。			这两个方法和上面两个方法的区别是响应方式是转发还是重定向。
String		encodeURL(String url) URL重写方法，用于浏览器禁用Cookie的会话保持。从2.1版本标注过时。用encodeURL代替。			
String		getHeader(String name) 根据名称，获取响应消息头的内容			
Collection<String>		getHeaderNames() 获取响应消息头名称的集合。			
Collection<String>		getHeaders(String name) 根据名称，获取全部响应消息头内容。它适用于一个消息头，有多个值的情况。			
int		getStatus() 获取当前响应状态码			
default Supplier<Map<String, String>>		getTrailerFields() 获取Trailer消息头的提供商			
void		sendError(int sc) 向客户端发送指定状态码的错误响应，并清除缓冲区。（在HttpServlet的doGet方法中用的就是此方法）			
void		sendError(int sc, String msg) 向客户端发送指定状态的错误响应，同时携带错误描述。它也会清除缓冲区。			
void		sendRedirect(String location) 把客户浏览器地址栏重新定向到一个新的地址上。清除缓冲区。此方法一般被称为：重定向方法。			
void		setDateHeader(String name, long date) 设置日期类型的消息头。name是消息头的名称。date是消息头的值，是从1970年开始的毫秒值。			
void		setHeader(String name, String value) 设置字符串类型的消息头。			
void		setIntHeader(String name, int value) 设置整型的消息头			
void		setStatus(int sc) 设置响应状态码			
void		setStatus(int sc, String sm) Deprecated. 设置响应状态码，包括状态描述，从2.1版本被标注为过时			
default void		setTrailerFields(Supplier<Map<String, String>> supplier) 设置Trailer消息头的提供商			

常用状态码：

状态码	说明
200	执行成功
302	它和307一样，都是用于重定向的状态码。只是307目前已不再使用
304	请求资源未改变，使用缓存。
400	请求错误。最常见的就是请求参数有问题
404	请求资源未找到
405	请求方式不被支持
500	服务器运行内部错误

状态码首位含义：

状态码	说明
1xx	消息
2xx	成功
3xx	重定向
4xx	客户端错误
5xx	服务器错误

1.3 响应对象的使用示例

1.3.1 响应-字节流输出中文问题

```
/**
 * @author 黑马程序员
 * @Company http://www.itheima.com
 */
public class ResponseDemo1 extends HttpServlet {

    /**
     * 演示字节流输出的乱码问题
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

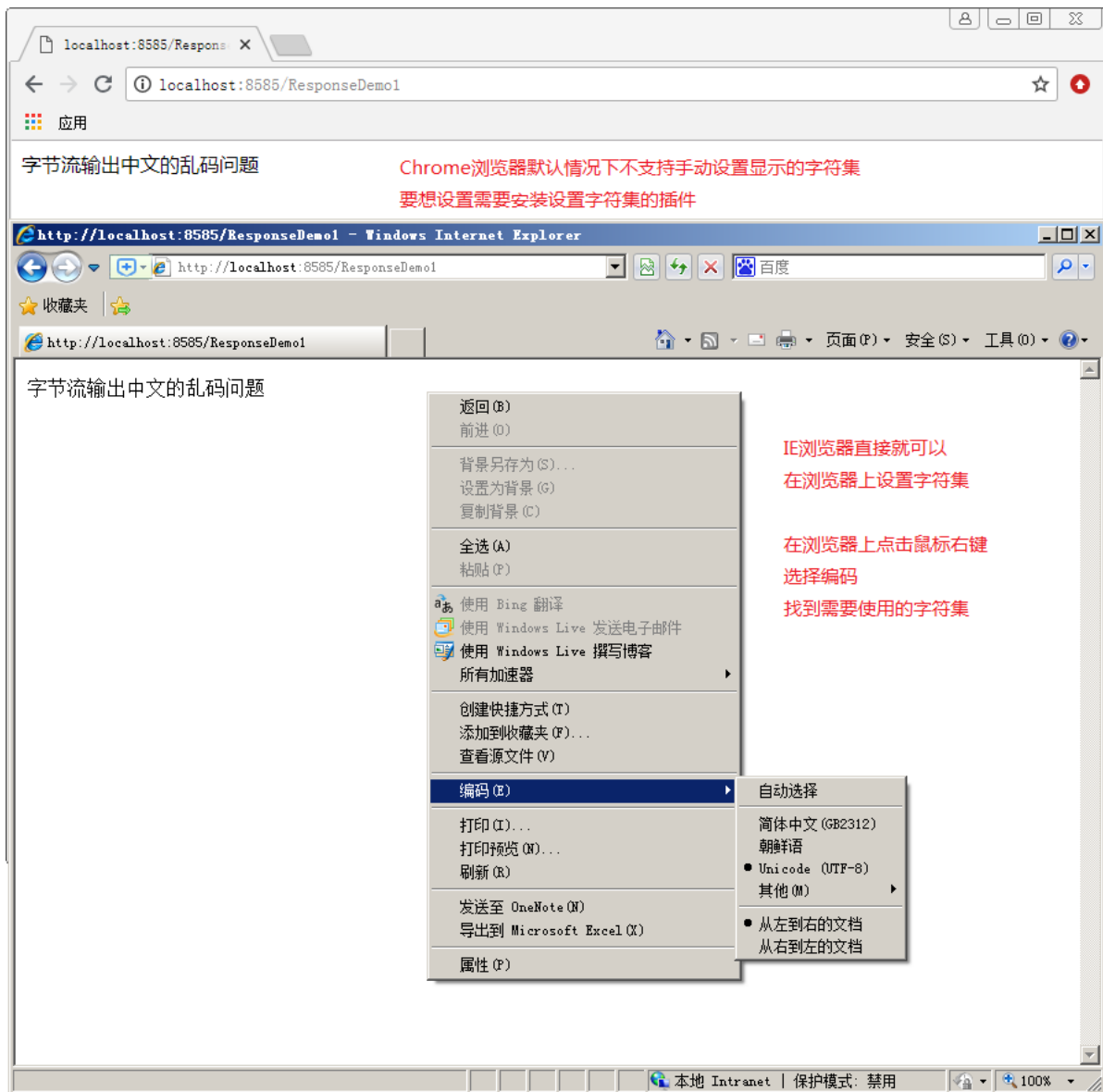
        /**
         * 问题：
         *     String str = "字节流中文乱码问题";
         *     使用字节流输出，会不会产生中文乱码？
         * 答案：
         *     会产生乱码
         * 原因：
         *     String str = "字节流中文乱码问题"; 在保存时用的是IDEA创建文件使用的字符集UTF-8。
         *     到浏览器上显示，chrome浏览器和ie浏览器默认的字符集是GB2312(其实就是GBK)，存和取用的不是同一个码表，就会产生乱码。
         */
    }
}
```

```

    * 引申：
    * 如果产生了乱码，就是存和取用的不是同一个码表
    * 解决办法：
    * 把存和取的码表统一。
    */
    String str = "字节流输出中文的乱码问题";//UTF-8的字符集，此时浏览器显示也需要使用
    UTF-8的字符集。
    //1.拿到字节流输出对象
    ServletOutputStream sos = response.getOutputStream();
    /**
    * 解决办法：
    * 第一种解决办法：
    * 修改浏览器的编码，使用右键--编码--改成UTF-8。（不推荐使用，我们的应用尽量不要
    求用户取做什么事情）
    * ie和火狐浏览器可以直接右键设置字符集。而chrome需要安装插件，很麻烦。
    * 第二种解决办法：（不建议使用，因为不好记）
    * 向页面上输出一个meta标签，内容如下： <meta http-equiv="content-type"
    content="text/html; charset=UTF-8">
    * 其实它就是指挥了浏览器，使用哪个编码进行显示。
    * 第三种解决办法：
    * 设置响应消息头，告知浏览器响应正文的MIME类型和字符集
    * response.setHeader("Content-Type", "text/html; charset=UTF-8");
    * 第四种解决办法：我们推荐使用的办法
    * 它的本质就是设置了一个响应消息头
    * response.setContentType("text/html; charset=UTF-8");
    */
    //第二种解决办法：sos.write("<meta http-equiv='content-type'
    content='text/html; charset=UTF-8'>".getBytes());
    //第三种解决办法：response.setHeader("Content-Type", "text/html; charset=UTF-
    8");
    //第四种解决办法：
    response.setContentType("text/html; charset=UTF-8");
    //2.把str转换成字节数组之后输出到浏览器
    sos.write(str.getBytes("UTF-8"));
}

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

```



1.3.2 响应-字符流输出中文问题

```
/**
 * @author 黑马程序员
 * @Company http://www.itheima.com
 */
public class ResponseDemo2 extends HttpServlet {

    /**
     * 字符流输出中文乱码
     * @param request
     * @param response
     * @throws ServletException
     * @throws IOException
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String str = "字节流输出中文乱码";
        //response.setCharacterEncoding("UTF-8");

        //设置响应正文的MIME类型和字符集
        response.setContentType("text/html;charset=UTF-8");
        //1. 获取字符输出流
```

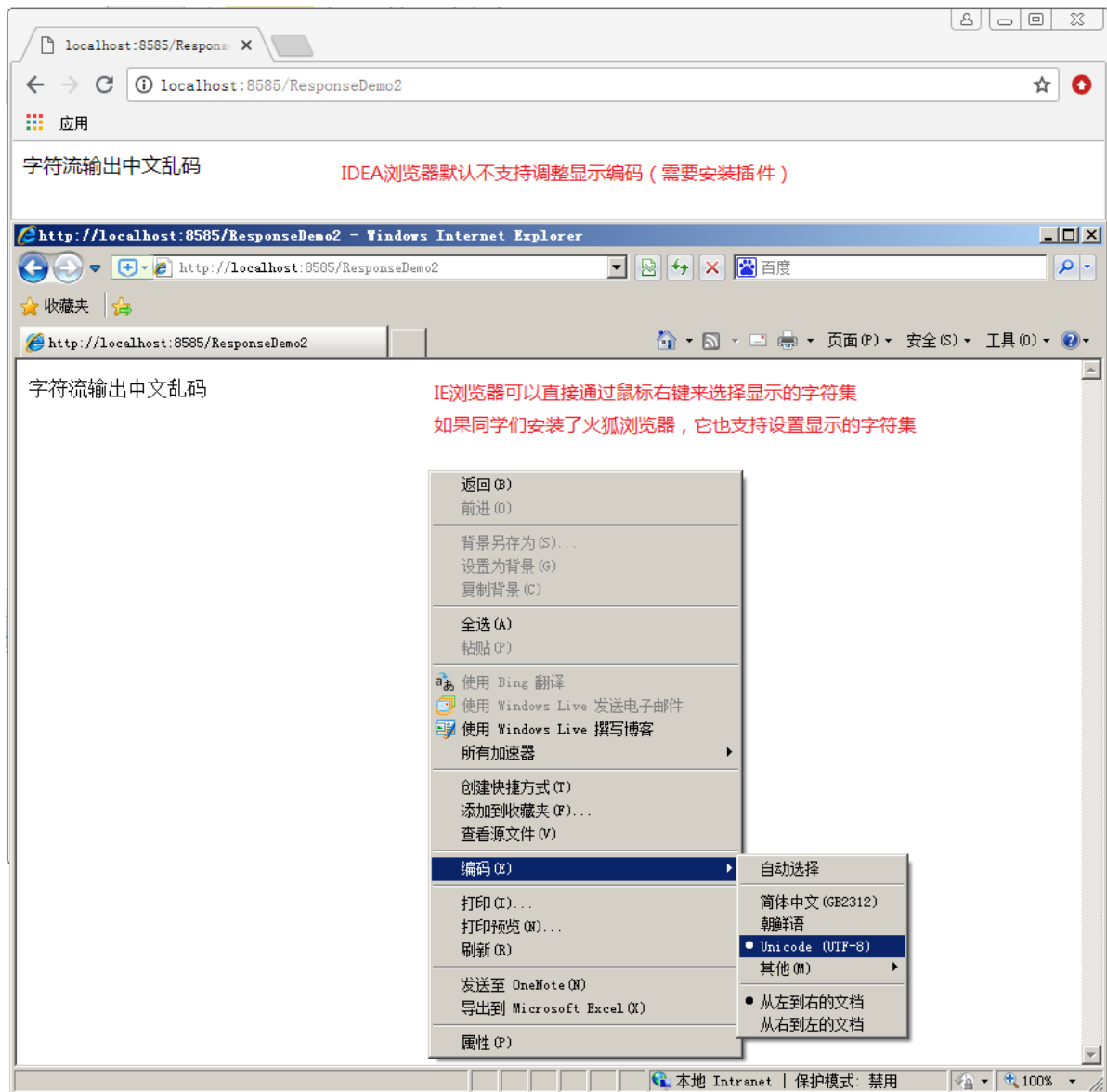
```

PrintWriter out = response.getWriter();
//2.使用字符流输出中文
/**
 * 问题:
 * out.write(str); 直接输出, 会不会产生乱码
 * 答案:
 * 会产生乱码
 * 原因:
 * 存用的什么码表: UTF-8
 * 在浏览器取之前, 字符流PrintWriter已经获取过一次了, PrintWriter它在取的时候出现
了乱码。
 * 浏览器取默认用的是GBK。(本地系统字符集)
 *
 * UTF-8(存)---->PrintWriter ISO-8859-1(取) 乱
 * PrintWritter ISO-8859-1(存)---->浏览器 GBK(取) 乱
 *
 * 解决办法:
 * 改变PrintWriter的字符集, PrintWriter是从response对象中获取的, 其实设置
response的字符集。
 * 注意: 设置response的字符集, 需要在拿流之前。
 * response.setCharacterEncoding("UTF-8");
 *
 * response.setContentType("text/html; charset=UTF-8");
 * 此方法, 其实是做了两件事:
 * 1. 设置响应对象的字符集 (包括响应对象取出的字符输出流)
 * 2. 告知浏览器响应正文的MIME类型和字符集
 */

out.write(str);
}

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

```



1.3.3 响应-生成验证码

```
/**
 * @author 黑马程序员
 * @Company http://www.itheima.com
 *
 */
public class ResponseDemo3 extends HttpServlet {

    /**
     * 输出图片
     * @param request
     * @param response
     * @throws ServletException
     * @throws IOException
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        int width = 200;
        int height = 35;
        /**
         * 实现步骤:
         * 1. 创建图像内存对象
         */
    }
}
```

```

        * 2.拿到画笔
        * 3.设置颜色，画矩形边框
        * 4.设置颜色，填充矩形
        * 5.设置颜色，画干扰线
        * 6.设置颜色，画验证码
        * 7.把内存图像输出到浏览器上
        */
//创建内存图像
BufferedImage image = new
BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);//参数：宽度，高度 （指的都是
像素），使用的格式（RGB）
    Graphics g = image.getGraphics();//画笔就一根

//设置颜色
g.setColor(Color.BLUE);
//画边框
g.drawRect(0, 0, width, height);

//设置颜色
g.setColor(Color.GRAY);
//填充矩形
g.fillRect(1, 1, width-2, height-2);

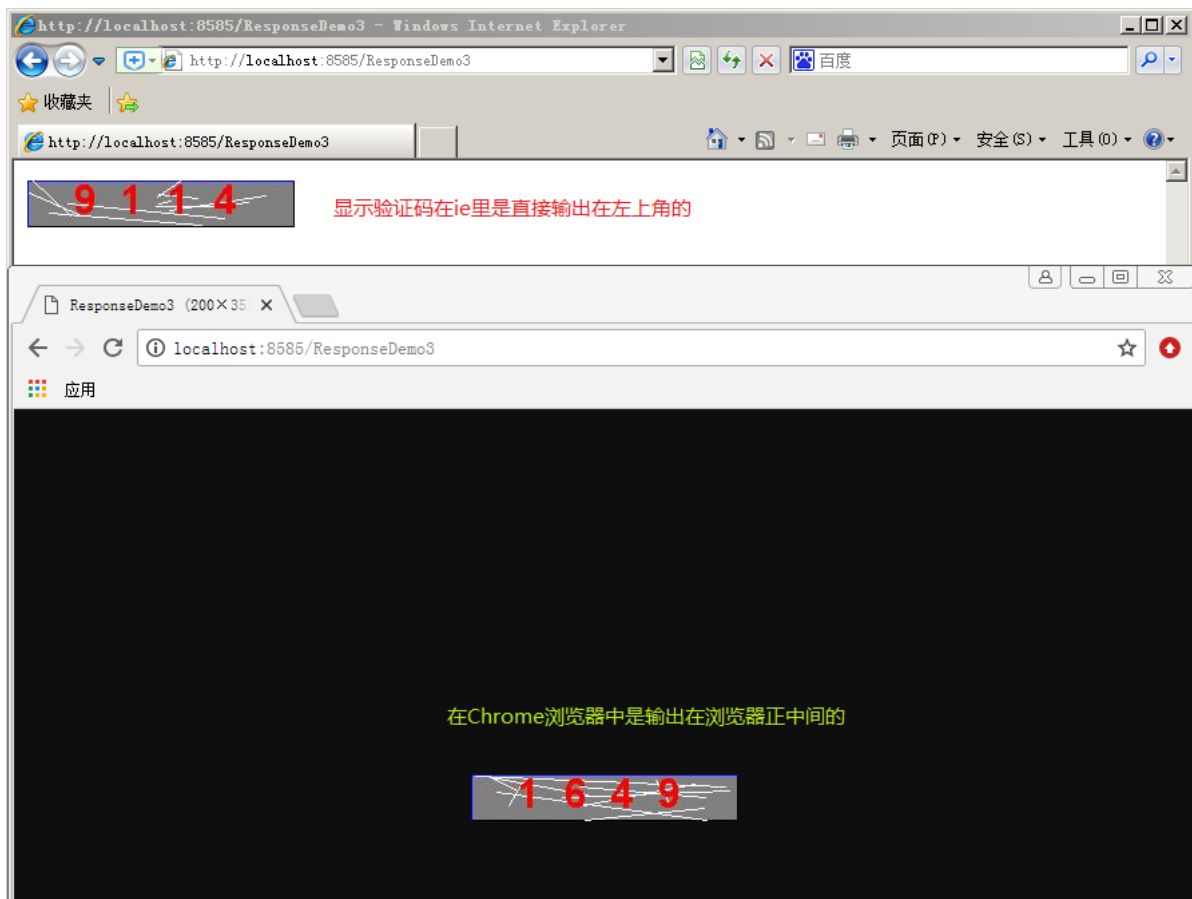
//设置颜色
g.setColor(Color.WHITE);
//拿随机数对象
Random r = new Random();
//画干扰线 10条
for(int i=0;i<10;i++){
    g.drawLine(r.nextInt(width), r.nextInt(height),r.nextInt(width),
r.nextInt(height));
}

//设置颜色
g.setColor(Color.RED);
//改变字体大小
Font font = new Font("宋体", Font.BOLD,30);//参数： 1字体名称。2.字体样式 3.字
体大小
g.setFont(font);//设置字体
//画验证码 4个
int x = 35;//第一个数的横坐标是35像素
for(int i=0;i<4;i++){
    //r.nextInt(10)+" "这种写法效率是十分低的
    g.drawString(String.valueOf(r.nextInt(10)), x, 25);
    x+=35;
}

//输出到浏览器上
//参数： 1.内存对象。2.输出的图片格式。3.使用的输出流
ImageIO.write(image, "jpg", response.getOutputStream());
}

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

```

1.3.4 设置响应消息头-控制缓存

```
/**
 * 设置缓存时间
 * 使用缓存的一般都是静态资源
 * 动态资源一般不能缓存。
 * 我们现在目前只掌握了Servlet，所以用Servlet做演示
 * @author 黑马程序员
 * @Company http://www.itheima.com
 *
 */
public class ResponseDemo4 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String str = "设置缓存时间";
        /**
         * 设置缓存时间，其实就是设置响应消息头: Expires 但是值是一个毫秒数。
         * 使用的是
         * response.setDateHeader();
         *
         * 缓存1小时，是在当前时间的毫秒数上加上1小时之后的毫秒值
         */

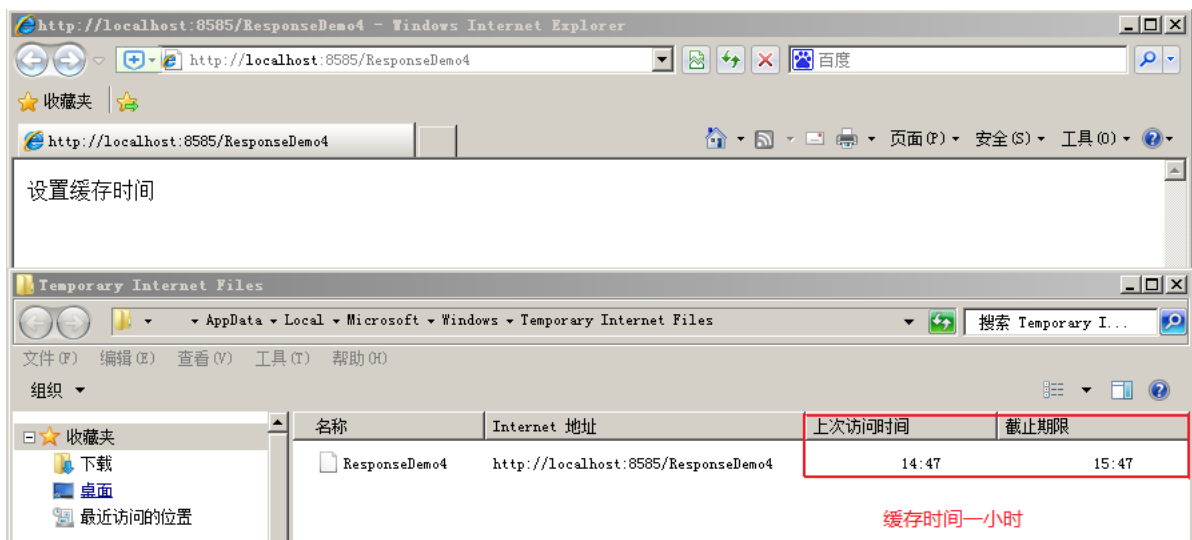
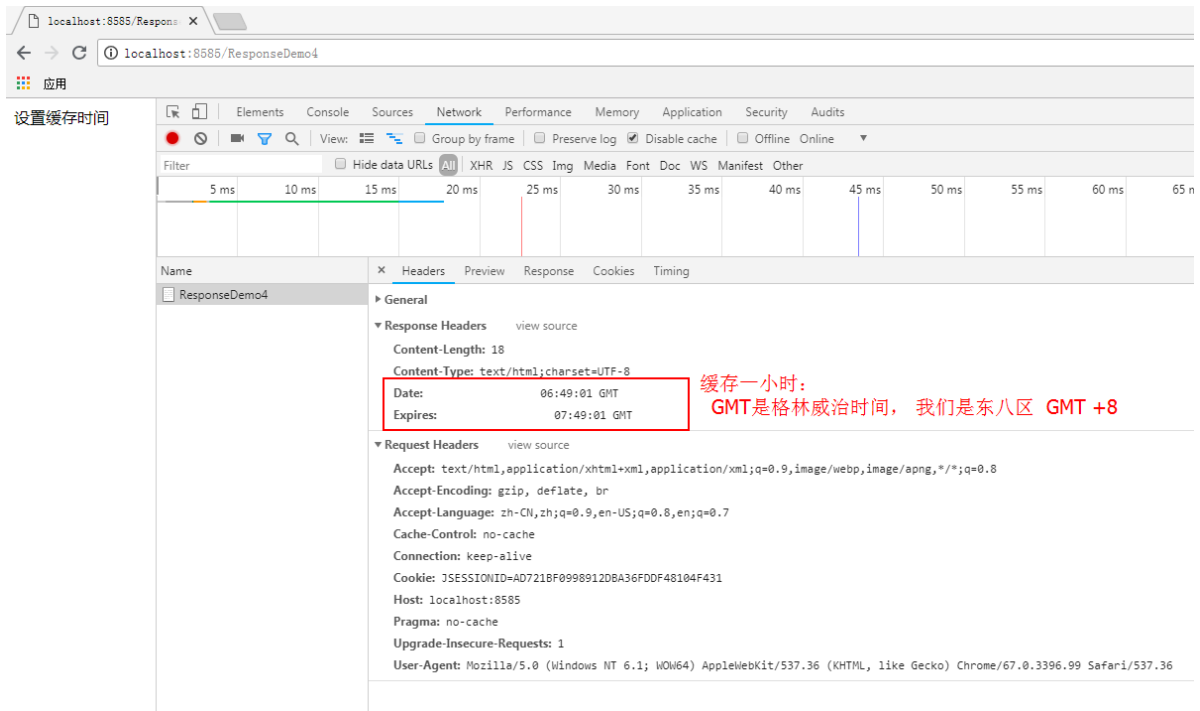
        response.setDateHeader("Expires", System.currentTimeMillis() + 1 * 60 * 60 * 1000);
        response.setContentType("text/html; charset=UTF-8");
        response.getOutputStream().write(str.getBytes());
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
```

```

        throws ServletException, IOException {
            doGet(request, response);
        }
    }
}

```



1.3.5 设置响应消息头定时刷新

```

/**
 * 设置响应消息头:
 * 通过定时刷新演示添加消息头
 * @author 黑马程序员
 * @Company http://www.itheima.com
 */
public class ResponseDemo5 extends HttpServlet {

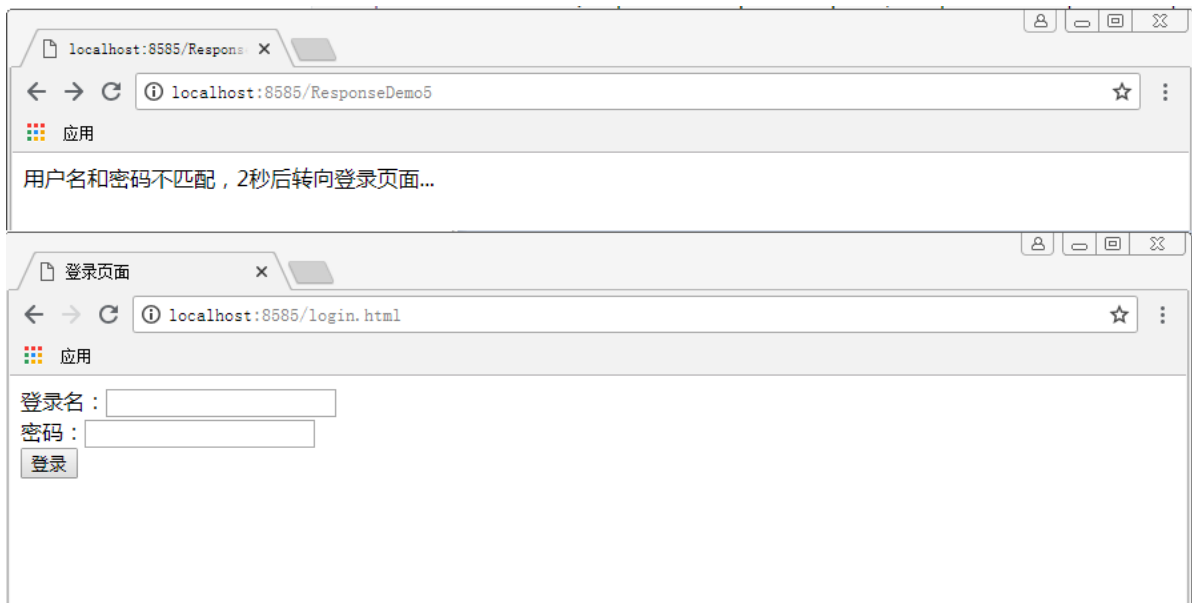
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

```

```
String str = "用户名和密码不匹配, 2秒后转向登录页面...";
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
out.write(str);
//定时刷新, 其实就是设置一个响应消息头
response.setHeader("Refresh", "2;URL=/login.html");//Refresh设置的时间单位
是秒, 如果刷新到其他地址, 需要在时间后面拼接上地址
}

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

}
```



1.3.6 请求重定向：注意地址栏发生改变。

```
/**
 * 设置响应状态码, 实现重定向
 * 重定向的特点:
 * 两次请求, 地址栏改变, 浏览器行为, xxxx
 * @author 黑马程序员
 * @Company http://www.itheima.com
 *
 */
public class ResponseDemo6 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        //1. 设置响应状态码
        response.setStatus(302);
        //2. 定向到哪里去: 其实就是设置响应消息头, Location
        response.setHeader("Location", "ResponseDemo7");

        //使用重定向方法
        response.sendRedirect("ResponseDemo7");//此行做了什么事, 请看上面
    }
}
```

```

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

```

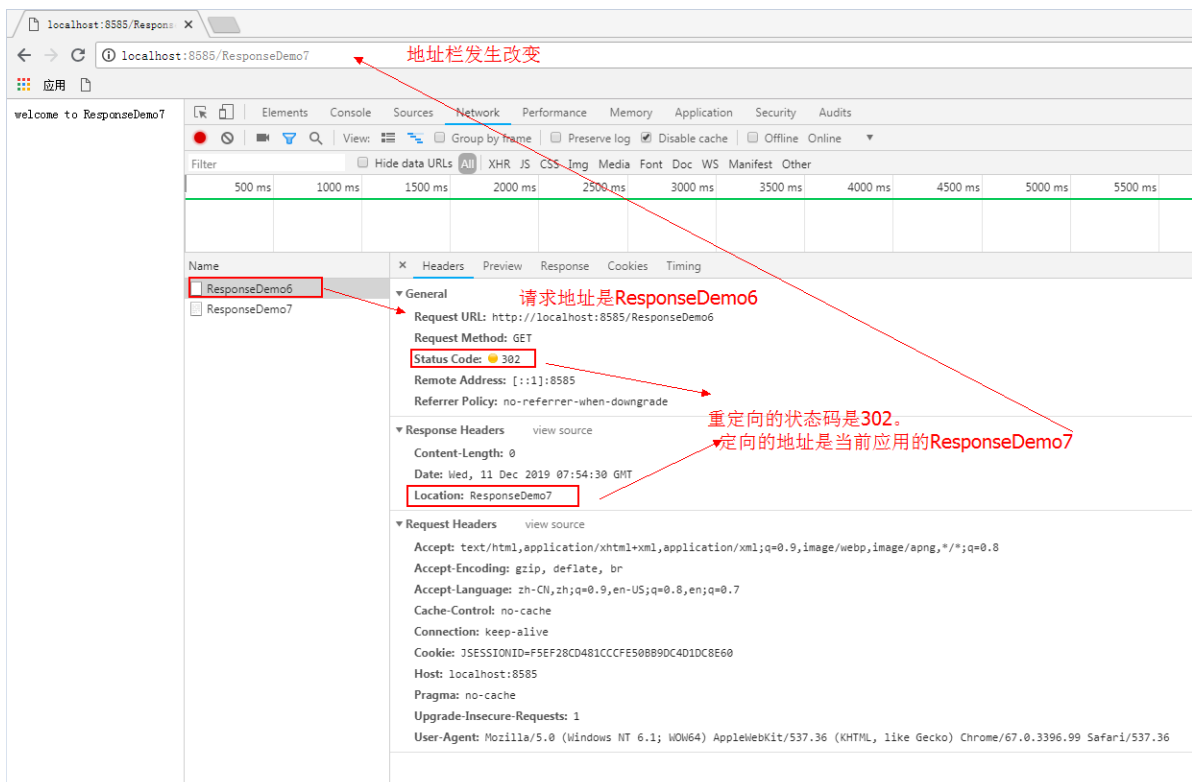
```

/**
 * 重定向的目的地
 * @author 黑马程序员
 * @Company http://www.itheima.com
 */
public class ResponseDemo7 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.getWriter().write("welcome to ResponseDemo7");
    }

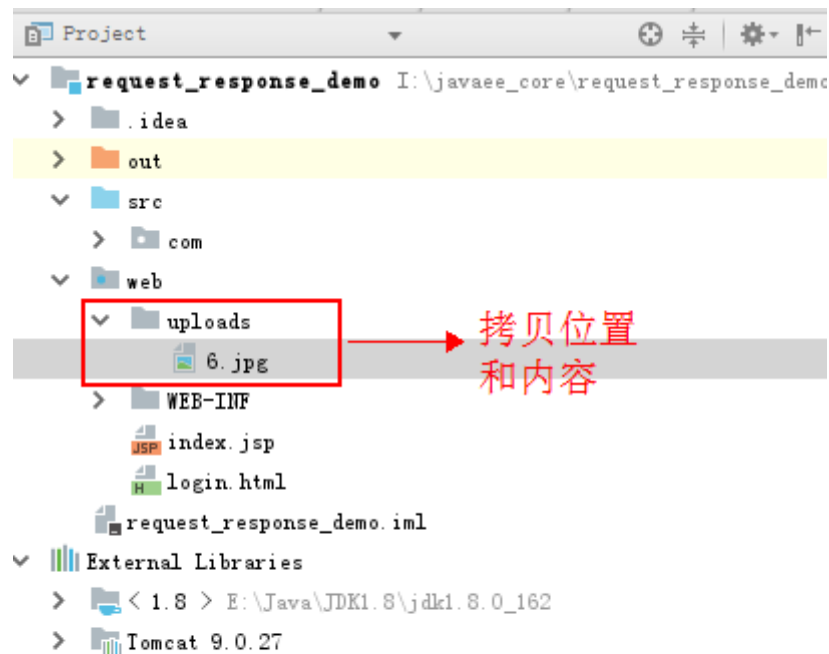
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}

```



1.3.7 响应和消息头组合应用-文件下载

首先，在工程的web目录下新建一个目录uploads，并且拷贝一张图片到目录中，如下图所示：



文件下载的Servlet代码如下：

```
/**
 * 文件下载
 * @author 黑马程序员
 * @Company http://www.itheima.com
 *
 */
public class ResponseDemo8 extends HttpServlet {

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

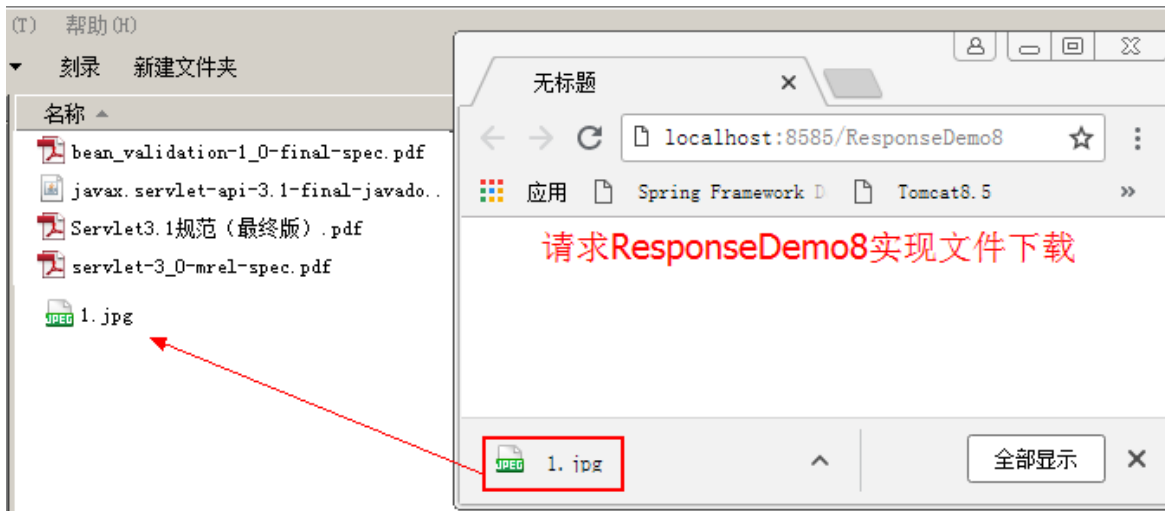
        /**
         * 文件下载的思路：
         * 1. 获取文件路径
         * 2. 把文件读到字节输入流中
         * 3. 告知浏览器，以下载的方式打开（告知浏览器下载文件的MIME类型）
         * 4. 使用响应对象的字节输出流输出到浏览器上
         */
        //1. 获取文件路径（绝对路径）
        ServletContext context = this.getServletContext();
        String filePath = context.getRealPath("/uploads/6.jpg");//通过文件的虚拟路
        径，获取文件的绝对路径
        //2. 通过文件路径构建一个字节输入流
        InputStream in = new FileInputStream(filePath);
        //3. 设置响应消息头
        response.setHeader("Content-Type", "application/octet-stream");//注意下载
        的时候，设置响应正文的MIME类型，用application/octet-stream
        response.setHeader("Content-Disposition",
            "attachment;filename=1.jpg");//告知浏览器以下载的方式打开
        //4. 使用响应对象的字节输出流输出
        OutputStream out = response.getOutputStream();
        int len = 0;
        byte[] by = new byte[1024];
```

```

        while((len = in.read(by)) != -1){
            out.write(by, 0, len);
        }
        in.close();
    }

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

```



1.3.8 响应对象注意事项

第一：response得到的字符流和字节流互斥，只能选其一

第二：response获取的流不用关闭，由服务器关闭即可

```

/**
 * 使用Response对象获取流时候的注意事项：
 * 1. 我们使用response获取的流，可以不用关闭。服务器会给我们关闭。
 * 2. 在response对象中，字节流和字符流互斥，输出的时候，只能选择一个
 * @author zhy
 */
public class ResponseDemo9 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String str = "test";
        response.getOutputStream().write(str.getBytes());
        //response.getWriter().write(str);
        // response.getOutputStream().write("haha".getBytes());
    }

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

```

```
}

9  /**
10 * 使用Response对象获取流时候的注意事项：
11 * 1. 我们使用response获取的流，可以不用关闭。服务器会给我们关闭。
12 * 2. 在response对象中，字节流和字符流互斥，输出的时候，只能选择一个
13 * @author zhy
14 */
15
16 public class ResponseDemo9 extends HttpServlet {
17
18     public void doGet(HttpServletRequest request, HttpServletResponse response)
19         throws ServletException, IOException {
20         String str = "test";
21         response.getOutputStream().write(str.getBytes());
22         response.getWriter().write(str);
23     }
24 }
```

在ResponseDemo9中的doGet方法中添加断点。当我们发送请求，请求到达Servlet之后，被断点捕获，然后逐步执行，就会执行到下面的代码中。

localhost:8585
localhost:8585/ResponseDemo9

非法状态异常：这个response已经调用过getOutputStream()。（也就不允许在调用getWriter()方法了）
} catch (IOException | ServletException | RuntimeException e) { e: "java.lang.IllegalStateException: getOutputStr
throw e; e: "java.lang.IllegalStateException: getOutputStream() has already been called for this response"

2 请求对象

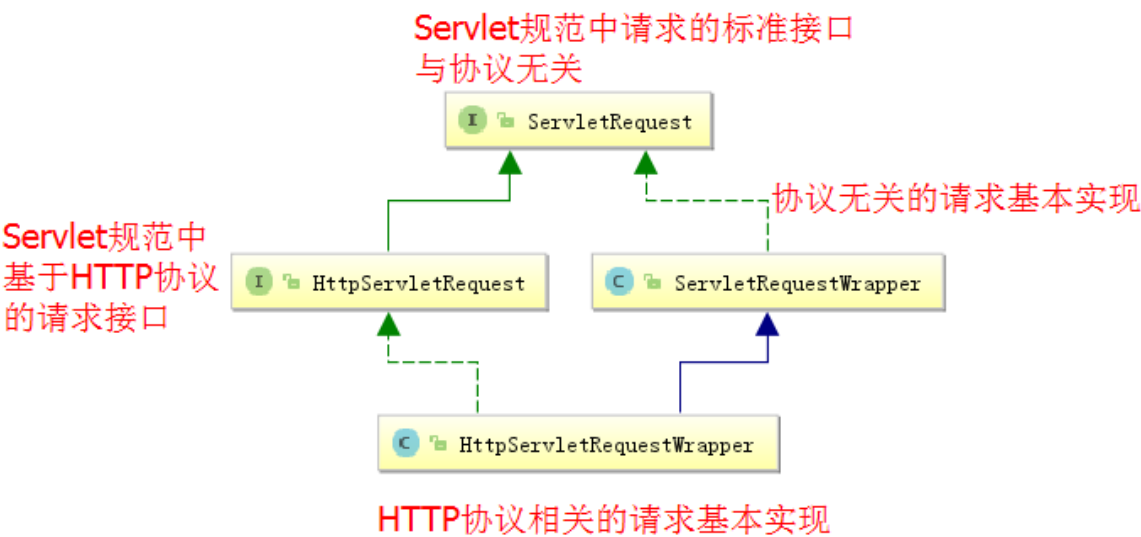
2.1 请求对象概述

2.1.1 关于请求

请求，顾名思义，就是使用者希望从服务器端索取一些资源，向服务器发出询问。在B/S架构中，就是客户浏览器向服务器发出询问。在我们的JavaEE工程中，客户浏览器发出询问，要遵循HTTP协议所规定的。

请求对象，就是在JavaEE工程中，用于发送请求的对象。我们常用的对象就是ServletRequest和HttpServletRequest，它们的区别就是是否和HTTP协议有关。

2.1.2 常用请求对象



2.2 常用方法介绍

All Methods	Instance Methods	Abstract Methods	Default Methods	Deprecated Methods
Modifier and Type		Method and Description		
boolean		<code>authenticate(HttpServletResponse response)</code> 使用为ServletContext配置的容器登录机制来验证当前用户的请求，Servlet3.0规范中的新方法。		
String		<code>changeSessionId()</code> 改变当前请求对应的会话ID，它只会改变session的ID，不会新创建HttpSession对象。Servlet3.1版本加入的新方法		
String		<code>getAuthType()</code> 返回用于保护servlet的验证方法名称。 所有servlet容器都支持 basic，form和client_certificate验证，并且可能还支持digest验证。		
String		<code>getContextPath()</code> 获取当前项目访问的URL。就是我们在IDEA里面部署项目时配置的	Application context: <input type="text" value="/"/>	
Cookie[]		<code>getCookies()</code> 获取当前请求所携带的所有Cookie对象。（Cookie也是会话管理的技术，知识我们在明天的课程中讲解）		
long		<code>getDateHeader(String name)</code> 根据消息头的名称，获取日期类型的请求消息头的值。得到的是一个从1970年开始计算的毫秒值。		
String		<code>getHeader(String name)</code> 根据名称，获取字符串类型的请求消息头的值		
Enumeration<String>		<code>getHeaderNames()</code> 获取请求消息头名称的枚举。返回的是所有请求消息头的名称。		
Enumeration<String>		<code>getHeaders(String name)</code> 根据消息头名称，获取消息头的值。它和getHeader的区别是，它是一个消息头对应多个值。		
default HttpServletMapping		<code>getHttpServletMapping()</code> 它是Servlet4.0规范的内容。由于我们用的是JavaEE8，配合Tomcat9，所以能看到Servlet4.0规范的内容 返回当前请求所在的Servlet的HttpServletMapping。		
int		<code>getIntHeader(String name)</code> 根据消息头名称，返回int类型的消息头的值		
String		<code>getMethod()</code> 返回请求的方式。例如:GET,POST,PUT等等。		
Part		<code>getPart(String name)</code> 表单的enctype取值为：multipart/form-data时使用此方法 根据给定的名称，获取多部分表单数据的内容。如果没有指定名称的内容，则返回null。它是Servlet3.0规范加入的。		
Collection<Part>		<code>getParts()</code> 获取多部分表单数据中的所有part的集合。要求表单的enctype取值为：multipart/form-data		
String		<code>getPathInfo()</code> 返回与客户端发出此请求时发送的URL相关联的任何额外路径信息。		
String		<code>getPathTranslated()</code> 返回servlet名称之后、查询字符串（QueryString）之前的任何额外路径信息，并将其转换为实际路径。		
String		<code>getQueryString()</code> 通常GET方式请求URL是：http://localhost:8585/ServletDemo?username=test&age=18 获取请求URL中携带参数的字符串，包括参数名称以及参数的值。这段内容就是QueryString		
String		<code>getRemoteUser()</code> 如果用户已通过身份验证，则返回发出此请求的用户的登录名；如果用户未通过身份验证，则返回null。		
String		<code>getRequestedSessionId()</code> 获取当前请求关联的会话ID。		
String		<code>getRequestURI()</code> http://localhost:8585/ServletDemo?username=test&age=18 获取的是此部分内容，如果配置了项目的访问路径，那么也会获取到。		
StringBuffer		<code>getRequestURL()</code> http://localhost:8585/ServletDemo?username=test&age=18 获取当前请求的URL。 获取的是此部分内容		
String		<code>getServletPath()</code> 获取Servlet的映射路径。就是url-pattern标签配置的内容，或者@WebServlet注解的value属性的取值。		
HttpSession		<code>getSession()</code> 获取当前请求关联的会话对象，如果没有的话就创建一个新的会话。		
HttpSession		<code>getSession(boolean create)</code> 如果参数是true，它和getSession()方法是一样的。如果参数是false，在当前请求没有关联的会话时，直接返回null。		
default Map<String, String>		<code>getTrailerFields()</code> 获取HTTP中Trailer报头信息。它也是Servlet4.0规范新加入的内容。		
Principal		<code>getUserPrincipal()</code> 返回包含当前已验证用户名称的java.security.Principal对象。		
boolean		<code>isRequestedSessionIdFromCookie()</code> 检查请求的会话ID是否作为cookie传入		
boolean		<code>isRequestedSessionIdFromUrl()</code> Deprecated. 检查请求的会话ID是否作为请求URL的一部分传入。此方法从2.1版本之后被标注过时了。		
boolean		<code>isRequestedSessionIdFromURL()</code> 检查请求的会话ID是否作为请求URL的一部分传入。		
boolean		<code>isRequestedSessionIdValid()</code> 检查当前请求关联的会话ID是否仍然有效。		
default boolean		<code>isTrailerFieldsReady()</code> 指示是否可以使用getTraileFields()读取Trailer字段，默认值是false，不允许。		
boolean		<code>isUserInRole(String role)</code> 用于判断认证后的用户是否属于逻辑role中的成员		
void		<code>login(String username, String password)</code> 验证提供的用户名和密码，然后将已验证的用户与请求关联。它是 Servlet 3.0规范提供的方法。		
void		<code>logout()</code> 从请求中移除认证成功的用户。它是Servlet3.0规范加入的方法。		
default PushBuilder		<code>newPushBuilder()</code> 获取推送器对象。以从当前请求发出服务器推送响应。它是Servlet4.0规范的新方法		
<T extends HttpUpgradeHandler> T		<code>upgrade(Class<T> handlerClass)</code> 为给定类创建HttpUpgradeHandler实例，并将其用于http协议升级处理。它是Servlet3.1规范中新加入的方法		

2.3 请求对象的使用示例

2.3.1 请求对象常用方法1-获取各种路径

```
/**
 * 请求对象的各种信息获取
 * @author 黑马程序员
 * @Company http://www.itheima.com
 */
public class RequestDemo1 extends HttpServlet {

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

        //本机地址：服务器地址
        String localAddr = request.getLocalAddr();
        //本机名称：服务器名称
        String localName = request.getLocalName();
        //本机端口：服务器端口
        int localPort = request.getLocalPort();
        //来访者ip
        String remoteAddr = request.getRemoteAddr();
        //来访者主机
        String remoteHost = request.getRemoteHost();
        //来访者端口
        int remotePort = request.getRemotePort();
        //统一资源标识符
        String URI = request.getRequestURI();
        //统一资源定位符
        String URL = request.getRequestURL().toString();
        //获取查询字符串
        String queryString = request.getQueryString();
        //获取Servlet映射路径
        String servletPath = request.getServletPath();

        //输出内容
        System.out.println("getLocalAddr() is :"+localAddr);
        System.out.println("getLocalName() is :"+localName);
        System.out.println("getLocalPort() is :"+localPort);
        System.out.println("getRemoteAddr() is :"+remoteAddr);
        System.out.println("getRemoteHost() is :"+remoteHost);
        System.out.println("getRemotePort() is :"+remotePort);
        System.out.println("getRequestURI() is :"+URI);
        System.out.println("getRequestURL() is :"+URL);
        System.out.println("getQueryString() is :"+queryString);
        System.out.println("getServletPath() is :"+servletPath);
    }

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

2.3.2 请求对象常用方法2-获取请求头信息

```
/**
 * 获取请求消息头
 * @author 黑马程序员
 * @Company http://www.itheima.com
 */
public class RequestDemo2 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        //1.根据名称获取头的值    一个消息头一个值
        String value = request.getHeader("Accept-Encoding");
        System.out.println("getHeader():"+value);

        //2.根据名称获取头的值    一个头多个值
        Enumeration<String> values = request.getHeaders("Accept");
        while(values.hasMoreElements()){
            System.out.println("getHeaders():"+values.nextElement());
        }

        //3.获取请求消息头的名称的枚举
        Enumeration<String> names = request.getHeaderNames();
        while(names.hasMoreElements()){
            String name = names.nextElement();
            String value1 = request.getHeader(name);
            System.out.println(name+":"+value1);
        }
    }

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

2.3.3 请求对象常用方法3-获取请求参数（非常重要）

在本小节，我们会讲解HttpServletRequest对象获取请求参数的常用方法，以及把获取到的请求参数封装到实体类中的方式。首先，我们先来创建一个Servlet对象

```
/**
 * 封装请求正文到javabean（数据模型）
 * @author 黑马程序员
 * @Company http://www.itheima.com
 */
public class RequestDemo3 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        /**
         * 把下面
         *      1) 获取请求参数
         *      2) 封装请求参数到实体类中
         * 中定义的test1到test8逐个添加到此处来运行即可。
         */
    }
}
```

```

        */
    }

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

```

接下来，我们在来准备一个表单页面：

```

<html>
<head>
    <title>login to request demo 3</title>
</head>
<body>
<form action="/day10_1122_requestresponse/RequestDemo3" method="post">
    用户名: <input type="text" name="username" /><br/>
    密码: <input type="password" name="password" /><br/>
    性别: <input type="radio" name="gender" value="1" checked>男
         <input type="radio" name="gender" value="0">女
    <br/>
    <input type="submit" value="注册" />
</form>
</body>
</html>

```

现在，我们开始分析HttpServletRequest对象用于获取请求参数的方法：

1) 获取请求参数

getParameter()方法的示例代码

```

/**
 * 获取请求正文，一个名称对应一个值。
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
private void test1(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //1. 获取请求正文
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    String gender = request.getParameter("gender");
    System.out.println(username+","+password+","+gender);
}

```

没有使用确认密码

getParameterValues()方法的示例代码

```

/**
 * 获取请求正文，一个名称可能对应多个值
 * @param request
 * @param response
 * @throws ServletException

```

使用了确认密码

```

    * @throws IOException
    */
    private void test2(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        //1. 获取请求正文
        String username = request.getParameter("username");
        String[] password = request.getParameterValues("password");//当表单中有多个名称
        是一样时，得到是一个字符串数组
        String gender = request.getParameter("gender");
        System.out.println(username+" "+Arrays.toString(password)+" "+gender);
    }

```

```

<html>
<head>
    <title>login to request demo 4</title>
</head>
<body>
<form action="/day10_1122_requestresponse/RequestDemo4" method="post"
enctype="multipart/form-data">
    用户名: <input type="text" name="username" /><br/>
    密码: <input type="password" name="password" /><br/>
    确认密码: <input type="password" name="password" /><br/>
    性别: <input type="radio" name="gender" value="1" checked>男
        <input type="radio" name="gender" value="0">女
    <br/>
    <input type="submit" value="注册" />
</form>
</body>
</html>

```

getParameterNames()方法的示例代码

```

/**
 * 获取请求正文，一个名称一个值。但是先要获取正文名称的枚举（key的枚举）
 * 确认密码
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
private void test3(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //1. 获取请求正文名称的枚举
    Enumeration<String> names = request.getParameterNames();
    //2. 遍历正文名称的枚举
    while(names.hasMoreElements()){
        String name = names.nextElement();
        String value = request.getParameter(name);
        System.out.println(name+"."+value);
    }
}

```

总结:

以上三个方法可以获取表单提交过来的请求参数。

参数的名称是一个字符串，参数的值可能是一个字符串，也可能是一个字符串数组。

2) 封装请求参数到实体类中

我们通过上面的方法可以获取到请求参数，但是如果参数过多，在进行传递时，方法的形参定义将会变得非常难看。此时我们应该用一个对象来描述这些参数，它就是实体类。这种类的定义，从基础阶段我们就开始使用了。在基础阶段，我们做过一个学生管理系统，用到了一个Student的类，它就是用于描述一个学生的实体类。

我们现在要做的就是将表单中提交过来的数据填充到实体类中。

第一种：最简单直接的封装方式

```
/**
 * 封装请求正文到User对象中                                没有使用确认密码
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
private void test4(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //1. 获取请求正文
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    String gender = request.getParameter("gender");
    //2. 创建一个User对象
    User user = new User();
    System.out.println("封装前: "+user.toString());
    //3. 把请求正文封装到user对象中
    user.setUsername(username);
    user.setPassword(password);
    user.setGender(gender);
    System.out.println("封装后: "+user.toString());
}
```

第二种：使用反射方式封装

此种封装的使用要求是，表单 `<input>` 标签的name属性取值，必须和实体类中定义的属性名称一致。

```
/**
 * 封装请求正文到javabean中                                没有使用确认密码
 * 使用反射+内省实现数据模型的封装
 * 内省：是sun公司推出的一套简化反射操作的规范。把javabean中的元素都封装成一个属性描述器。
 *      属性描述器中会有字段信息，get和set方法（取值或存值）
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
private void test5(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //1. 获取请求正文名称的枚举
    Enumeration<String> names = request.getParameterNames();
    User user = new User();
    System.out.println("封装前: "+user.toString());
    //2. 遍历正文名称的枚举
    while(names.hasMoreElements()){
        String name = names.nextElement();
```

```

String value = request.getParameter(name);
try{
    //1.拿到User对象中的属性描述器。是谁的属性描述器：是由构造函数的第一个参数决定的。第二个参数是指定javabeen的字节码
    PropertyDescriptor pd = new PropertyDescriptor(name, User.class);//参数指的就是拿哪个类的哪个属性的描述器
    //2.设置javabeen属性的值
    Method method = pd.getWriteMethod();
    //3.执行方法
    method.invoke(user, value);//第一个参数是指的给哪个对象，第二个参数指的是赋什么值
}catch(Exception e){
    e.printStackTrace();
}
}
System.out.println("封装后: "+user.toString());
}

```

第三种：使用反射封装，同时请求参数的值是一个数组

此种方式其实就是针对请求参数中包含name属性相同的参数，例如：密码和确认密码，还有爱好。

```

/**
 * 获取请求正文的关系映射Map<String,String[]> 使用确认密码
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
private void test6(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //1.获取请求正文的映射关系
    Map<String,String[]> map = request.getParameterMap();
    //2.遍历集合
    for(Map.Entry<String,String[]> me : map.entrySet()){
        String name = me.getKey();
        String[] value = me.getValue();
        System.out.println(name+": "+Arrays.toString(value));
    }
}

```

当我们把请求参数获取出来之后，就要考虑如何针对数组的反射了，具体代码如下：

```

/**
 * 封装请求正文到javabeen。使用的是反射+内省 使用了确认密码
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
private void test7(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //1.获取请求正文的映射关系
    Map<String,String[]> map = request.getParameterMap();
    Users user = new Users();
    System.out.println("封装前: "+user.toString());
    //2.遍历集合
}

```

```

for(Map.Entry<String,String[]> me : map.entrySet()){
    String name = me.getKey();
    String[] value = me.getValue();
    try{
        //1.拿到User对象中的属性描述器。是谁的属性描述器：是由构造函数的第一个参数决定的。第二个参数是指定javabean的字节码
        PropertyDescriptor pd = new PropertyDescriptor(name, Users.class);//
        参数指的就是拿哪个类的哪个属性的描述器
        //2.设置javabean属性的值
        Method method = pd.getWriteMethod();
        //3.执行方法
        //判断参数到底是几个值
        if(value.length > 1){//最少有2个元素
            method.invoke(user, (Object)value);//第一个参数是指的给哪个对象，第二个参数指的是赋什么值
        }else{
            method.invoke(user, value);//第一个参数是指的给哪个对象，第二个参数指的是赋什么值
        }
    }catch(Exception e){
        e.printStackTrace();
    }
}
System.out.println("封装后: "+user.toString());
}

```

当我们写完此种封装方式之后，同学们可以发现，我们绝大多数封装都可以使用这段代码来实现。并且，无论是谁来写这段通用的封装代码，其代码内容都是大同小异的。**那么，我们就可以得出一个很有趣的结论：一般遇到这种情况时，肯定有人帮我们写好了，我们只需要用就行了。**我们后面还会遇到类似这样的情况。

此时，帮我们写好这段封装代码的是apache软件基金会，我们前面学习的tomcat也是它提供的。它里面有一个开源工具包集合commons，里面有很多开源工具类，今天我们就来讲解第一个：**commons-beanutils**。

第四种：使用apache的commons-beanutils实现封装

实现代码：

```

/**
 * 终极方法：使用beanutils实现请求正文封装到javabean中 使用了确认密码
 * 要想使用beanutils，需要先导包
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
private void test8(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    Users user = new Users();
    System.out.println("封装前: "+user.toString());
    try{
        BeanUtils.populate(user, request.getParameterMap());//就这一句话
    }catch(Exception e){
        e.printStackTrace();
    }
    System.out.println("封装后: "+user.toString());
}

```

```
}
```

2.3.4 用流的形式读取请求信息

我们除了使用2.3.3小节中获取请求参数之外，还可以使用下面代码中的 方式来获取：

```
/**
 * 使用流的方式读取请求正文
 * @author 黑马程序员
 * @Company http://www.itheima.com
 */
public class RequestDemo4 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        //1.获取请求正文的字节输入流
        ServletInputStream sis = request.getInputStream();
        //2.读取流中的数据
        int len = 0;
        byte[] by = new byte[1024];
        while((len = sis.read(by)) != -1){
            System.out.println(new String(by,0,len));
        }
    }

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

2.3.5请求正文中中文编码问题

关于请求中文乱码问题，我们需要分开讨论，第一是POST请求方式，第二是GET方式。

1) POST方式请求

在POST方式请求中，我们的乱码问题可以用如下代码解决：

```
/**
 * 请求正文的中文乱码问题
 * @author 黑马程序员
 * @Company http://www.itheima.com
 */
public class RequestDemo5 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        //1.获取请求正文
        /*POST方式：
        * 问题：
        * 取的时候会不会有乱码
        * 答案：
        * 获取请求正文，会有乱码问题。
        * 是在获取的时候就已经乱码了。
        */
    }
}
```



```

    * 解决办法：
    *   是request对象的编码出问题了
    *   设置request对象的字符集
    *   request.setCharacterEncoding("GBK");它只能解决POST的请求方式，GET方式解决不了
    * 结论：
    *   请求正文的字符集和响应正文的字符集没有关系。各是各的
    */
    request.setCharacterEncoding("UTF-8");
    String username = request.getParameter("username");
    //输出到控制台
    System.out.println(username);
    //输出到浏览器：注意响应的乱码问题已经解决了
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.write(username);
}

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

```

2) GET方式请求

GET方式请求的正文是在地址栏中，在Tomcat8.5版本及以后，Tomcat服务器已经帮我们解决了，所以不会有乱码问题了。

而如果我们使用的不是Tomcat服务器，或者Tomcat的版本是8.5以前，那么GET方式仍然会有乱码问题，解决方式如下：（以下代码了解即可，因为我们现在使用的是Tomcat9.0.27版本）

```

/**
 * 在Servlet的doGet方法中添加如下代码
 */
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    /**
     * GET方式：正文在地址栏
     * username=%D5%C5%C8%FD
     * %D5%C5%C8%FD是已经被编过一次码了
     *
     * 解决办法：
     *   使用正确的码表对已经编过码的数据进行解码。
     *   就是把取出的内容转成一个字节数组，但是要使用正确的码表。（ISO-8859-1）
     *   再使用正确的码表进行编码
     *   把字节数组再转成一个字符串，需要使用正确的码表，是看浏览器当时用的是什码表
     */
    String username = request.getParameter("username");
    byte[] by = username.getBytes("ISO-8859-1");
    username = new String(by, "GBK");

    //输出到浏览器：注意响应的乱码问题已经解决了
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
}

```

```

        out.write(username);
    }

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

```

2.3.6 请求转发（与重定向的区别）

在实际开发中，重定向和请求转发都是我们要用到的响应方式，那么他们有什么区别呢？我们通过下面的示例来看一下：

```

/**
 * 重定向特点：
 * 两次请求，浏览器行为，地址栏改变，请求域中的数据会丢失
 * 请求转发：
 * 一次请求，服务器行为，地址栏不变，请求域中的数据不丢失
 *
 * 请求域的作用范围：
 * 当前请求（一次请求），和当前请求的转发之中
 * @author 黑马程序员
 * @Company http://www.itheima.com
 */
public class RequestDemo6 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        //1.拿到请求调度对象
        RequestDispatcher rd = request.getRequestDispatcher("/RequestDemo7");//如果是给浏览器看的，/可写可不写。如果是给服务器看的，一般情况下，/都是必须的。
        //放入数据到请求域中
        request.setAttribute("CityCode", "bj-010");
        //2.实现真正的转发操作
        rd.forward(request, response);//实现真正的转发操作
    }

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

}

```

```

/**
 * 转发的目的地
 * @author 黑马程序员
 * @Company http://www.itheima.com
 */
public class RequestDemo7 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        //获取请求域中的数据
        String value = (String)request.getAttribute("CityCode");
        response.getWriter().write("welcome to request demo 7    "+value);
    }
}

```

```

    }

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

```

2.3.7 请求包含

在实际开发中，我们可能需要把两个Servlet的内容合并到一起响应浏览器，而同学们都知道HTTP协议的特点是一请求，一响应的方式。所以绝对不可能出现有两个Servlet同时响应方式。那么我们就需要用到请求包含，把两个Servlet的响应内容合并输出。我们看具体使用示例：

```

/**
 * 请求包含
 *
 * 它是把两个Servlet的响应内容合并输出。
 * 注意：
 * 这种包含是动态包含。
 *
 * 动态包含的特点：
 * 各编译各的，只是最后合并输出。
 * @author 黑马程序员
 * @Company http://www.itheima.com
 */
public class RequestDemo8 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.getWriter().write("I am request demo8 ");
        //1.拿到请求调度对象
        RequestDispatcher rd = request.getRequestDispatcher("/RequestDemo9");
        //2.实现包含的操作
        rd.include(request, response);
    }

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

```

```

/**
 * 被包含者
 * @author 黑马程序员
 * @Company http://www.itheima.com
 */
public class RequestDemo9 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.getWriter().write("include request demo 9 ");
    }
}

```

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

}
```

2.3.8 细节问题

请求转发的注意事项：负责转发的Servlet，转发前后的响应正文丢失，由转发目的地来响应浏览器。

请求包含的注意事项：被包含者的响应消息头丢失。因为它被包含起来了。

3 案例中的使用

3.1 案例的需求及环境

3.1.1 案例需求介绍

在昨天的课程中，我们实现了浏览器发送请求，由Servlet来接收。今天，我们继续对学生管理系统进行升级，通过Servlet来实现学生的新增，删除，修改，查询操作。

新增：Create

查询：Read

修改：Update

删除：Delete

每个单词取第一个字母，组成了CRUD。所以，同学们今后看到CRUD操作，指的就是增删改查。

今天案例的CRUD，我们只关注Servlet接收请求和处理响应，不用过多的去关注真正增删改查操作（因为，我们目前还是把学生信息写到文件中，等web5天课程结束，我们会讲解数据库，它是我们保存数据这类问题的终极解决方案，而保存文件只是个替代品，我们没必要在替代品处消耗太多精力）。