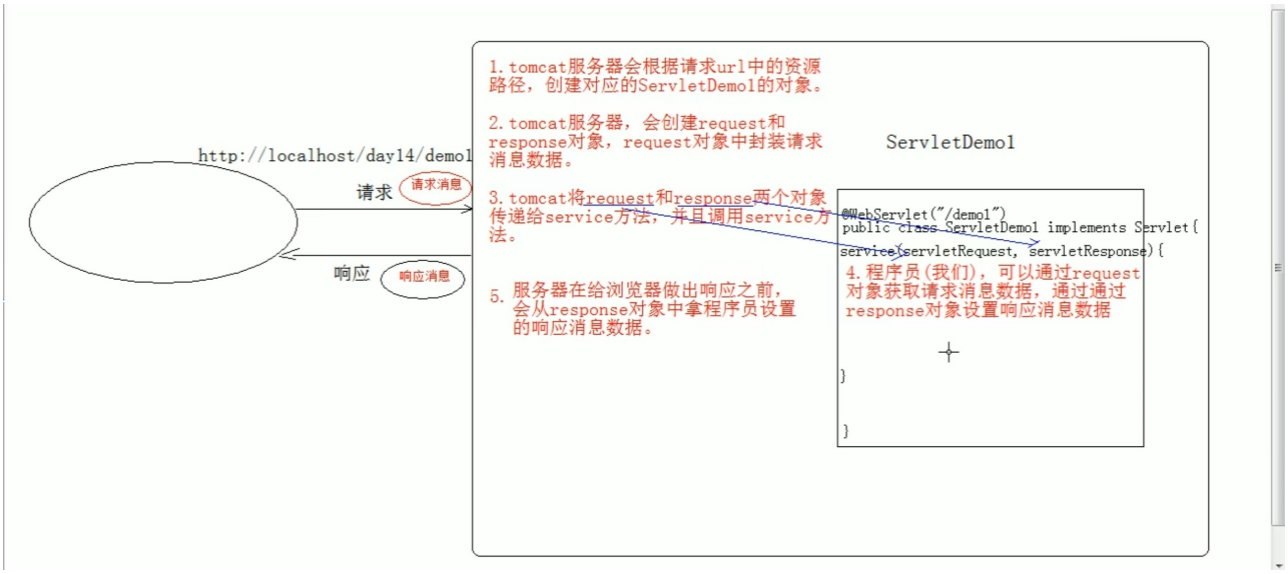
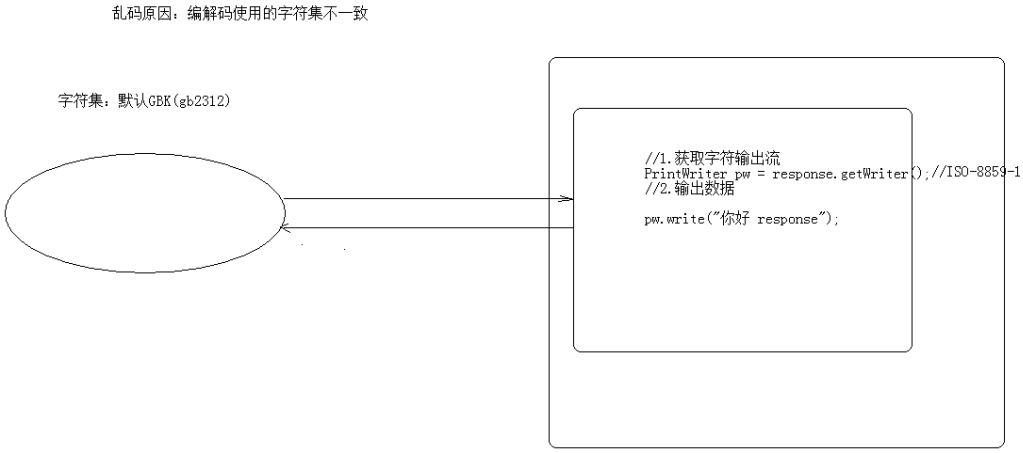
Response对象





ServletResponse的子接口是HttpServletRespons，setHeader方法是在HttpServletRespons里的

```
* 功能：设置响应消息
    1. 设置响应行
        1. 格式：HTTP/1.1 200 ok
        2. 设置状态码的方法：setStatus(int sc)
    2. 设置响应头：setHeader(String name, String value)
    3. 设置响应体：
        * 使用步骤：
            1. 获取输出流
                * 字符输出流方法及返回对象：PrintWriter getWriter()
                    注：通过字符输出流只能输出字符数据，返回对象是打印流
                * 字节输出流方法及返回对象：ServletOutputStream
getOutputStream()
                    注：通过字节输出流可以输出任意数据，可以把
ServletOutputStream流当作OutputStream流
            2. 使用输出流，将数据输出到客户端浏览器
```

完成重定向--案例：

```
完成重定向
        * 重定向：资源跳转的方式
        * 代码实现：
            //1. 设置状态码为302
            response.setStatus(302);
            //2.设置响应头location

response.setHeader("location","/ResponseSelf_war_exploded/responseD
emo22");
   //简单的重定向方法

response.sendRedirect("/ResponseSelf_war_exploded/responseDemo222")
;
        * forward（即转发） 和  redirect（即重定向） 区别：
                * 重定向的特点:redirect
                    1. 地址栏发生变化
                    2. 重定向可以访问其他站点(服务器)的资源
                    3. 重定向是两次请求，所以不能使用request对象来共享数据
                * 转发的特点：forward（之前讲的）
                    1. 转发地址栏路径不变
                    2. 转发只能访问当前服务器下的资源
                    3. 转发是一次请求，所以可以使用request对象来共享数据
        * 路径写法：
            1. 路径分类
```

1. 相对路径：通过相对路径不可以确定唯一资源
    * 如：./index.html
    * 不以/开头，以.开头路径

    * 规则：找到当前资源和目标资源之间的相对位置关系
        * ./：当前目录
        * ../:后退一级目录
2. 绝对路径：通过绝对路径可以确定唯一资源
    * 如：http://localhost/day15/responseDemo2
/day15/responseDemo2
    * 以/开头的路径

    * 规则：判断定义的路径是给谁用的？判断请求将来从哪儿发出
        * 给客户端浏览器使用：需要加虚拟目录(项目的访问路径)
            * 建议虚拟目录动态获取：
request.getContextPath()
            * <a> ，<form> ，重定向...
        * 给服务器使用：不需要加虚拟目录
            * 转发路径

- 按f12，点击网络，然后网址框搜索http://localhost:8080/ResponseSelf_war_exploded/responseDemo111，会看见两次请求：

服务器输出字符数据到浏览器--案例

服务器输出字符数据到浏览器：
    * 步骤：
        1. 获取字符输出流
        2. 输出数据

    * 注意：

* 乱码问题（乱码的原始是编码与解码用的码表不一致；输出数据相当于编码的过程，在浏览器显示相当于一个解码的过程；浏览器打开后没人知道用的是什么字符集，浏览器打开后默认的字符集跟我们当前操作系统的语言环境有关系，我们现在用的是中文的windos，那么打开浏览器后默认的字符集或者编码表是GBK（即gb2312），中文意思是“国标中文码表”，可以在ie浏览器中右键显示）

（注：如果流对象不是获取得到的，而是本地new出来的字符流，如果本地电脑是中文操作系统则这个流也是GBK的；但获取出来的流对象是tomcat返回给我们的，因为response是由tomcat实现的，tomcat用的编码是ISO-8859-1，也就是说response获得的这个流编码是ISO-8859-1，即GBK与ISO-8859-1不匹配则乱码）：

1. 设置该流的默认编码
2. 告诉浏览器响应体使用的编码
3. PrintWriter pw = response.getWriter();获取的流的默认编码是ISO-8859-1

```
//简单的形式，设置编码，是在获取流之前设置
response.setContentType("text/html;charset=utf-8");
```

服务器输出字节数据到浏览器--案例

服务器输出字节数据到浏览器：

* 步骤：
    1. 获取字节输出流
    2. 输出数据

验证码--案例

1. 本质：图片,验证码也是一张图片，只不过这张验证码是在内存里面，我么可以使用内存里的数据把它写到页面上去
2. 目的：防止恶意表单注册

- 流的内容也是响应体，如下：

ResponseDemo41 Tomcat 8.0.52

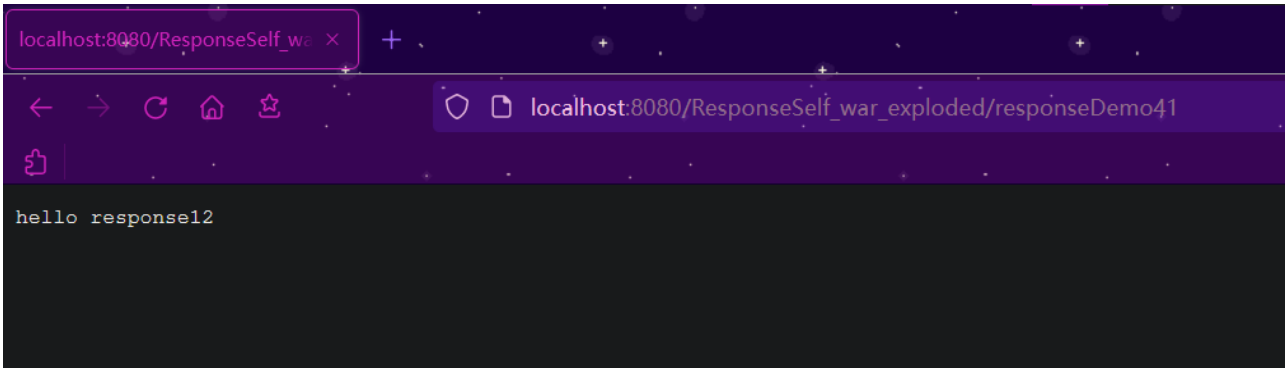
ResponseDemo41.java

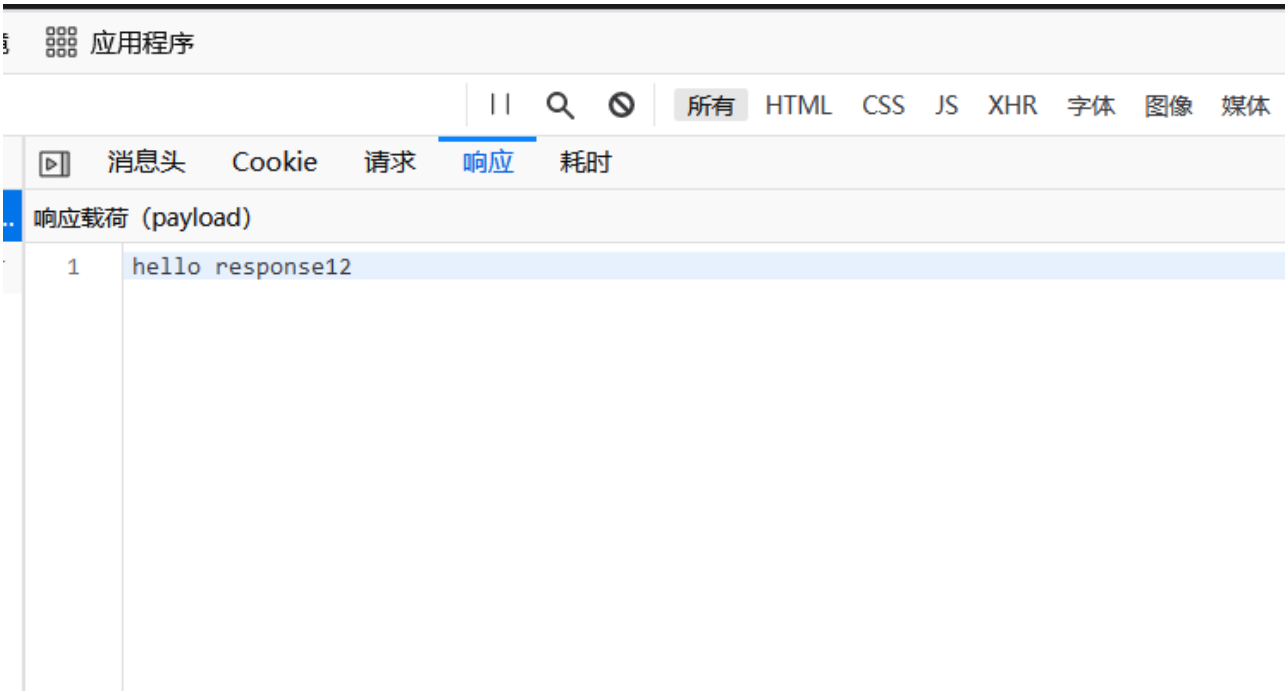
```java
package cn.itcast.web.servlet;

import ...

@WebServlet("/responseDemo41")
public class ResponseDemo41 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
        PrintWriter pw = response.getWriter();        //1.获取字符输出流
        pw.write( s: "hello response12"); //2.输出数据
        /* PrintWritery有两种输出数据的方式，一种用write,一种用println方法；
        write方法不能自动刷新，但是println方法可以自动刷新，自己就可以把数据写出缓冲区；
        目前这个PrintWriter是response获取的，response获取的将来PrintWeiter是不需要去刷新这个流，即使调用write这个方法也不需要刷新，
        response在一次响应完成了之后会自动地被销毁，同时它获得的流也会被自动的关闭掉，所以yongwrite方法也可以到达不刷新就可以写缓冲区的效果*/
    }

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





ServletContext对象：

```
1882        </mime-mapping>
1883        <mime-mapping>
1884            <extension>htc</extension>
1885            <mime-type>text/x-component</mime-type>
1886        </mime-mapping>
1887        <mime-mapping>
1888            <extension>htke</extension>
1889            <mime-type>application/vnd.kenameaapp</mime-type>
1890        </mime-mapping>
1891        <mime-mapping>
1892            <extension>htm</extension>
1893            <mime-type>text/html</mime-type>
1894        </mime-mapping>
1895        <mime-mapping>
1896            <extension>html</extension>
1897            <mime-type>text/html</mime-type>
1898        </mime-mapping>
1899        <mime-mapping>
1900            <extension>hvd</extension>
1901            <mime-type>application/vnd.yamaha.hv-dic</mime-type>
```

1. 概念：代表整个web应用，可以和程序的容器(服务器，即服务器tomcat)来通信
2. 获取：
    1. 方式一：通过request对象获取

        ```
        request.getServletContext();
        ```

    2. 方式二：通过HttpServlet获取

        ```
        this.getServletContext();
        ```

3. 功能：
    1. 获取MIME类型：
        * MIME类型:在互联网通信过程中定义的一种文件数据类型
            * 格式： 大类型/小类型 text/html image/jpeg

        * 获取：String getMimeType(String file)
        * 各种存储类型实质上是从服务器tomcat里获取的（如果不想用getMimeType方法获取，在web.xml的爹里找也行），可在D:\apache-tomcat-8.0.52-64\conf里的web.xml里看，它是是所有项目的web.xml的爹
    2. 它是一个域对象：共享数据
        1. setAttribute(String name,Object value)
        2. getAttribute(String name)
        3. removeAttribute(String name)

        * ServletContext对象范围（它是最大的范围）：所有用户所有请求的数据
    3. 获取文件的真实(服务器)路径 （即out下面的）
        1. 方法：String getRealPath(String path)

            ```
            String b = context.getRealPath("/b.txt");//web目录下资源访问

            System.out.println(b);
            ```

```
            String c = context.getRealPath("/WEB-INF/c.txt");//WEB-INF目录下的资源访问
            System.out.println(c);

            String a = context.getRealPath("/WEB-INF/classes/a.txt");//src目录下的资源访问
            System.out.println(a);
```

文件下载需求--案例：

```
* 文件下载需求：
    1. 页面显示超链接
    2. 点击超链接后弹出下载提示框
    3. 完成图片文件下载
* 分析：
    1. 超链接指向的资源如果能够被浏览器解析，则在浏览器中展示，如果不能解析，则弹出下载提示框。不满足需求
    2. 任何资源都必须弹出下载提示框
    3. 使用响应头设置资源的打开方式：
        * content-disposition:attachment;filename=xxx
* 步骤：
    1. 定义页面，编辑超链接href属性，指向Servlet，传递资源名称filename
    2. 定义Servlet
        1. 获取文件名称
        2. 使用字节输入流加载文件进内存
        3. 指定response的响应头： content-disposition:attachment;filename=xxx
        4. 将数据写出到response输出流
* 问题：
    * 中文文件问题
        * 解决思路：
            1. 获取客户端使用的浏览器版本信息
            2. 根据不同的版本信息，设置filename的编码方式不同
```