```
Java Web：是用Java技术来解决相关web互联网领域的技术总和。
在计算机的世界里，凡是提供服务的一方我们称为服务端（Server），而接受服务的另一方我们称作客户端（Client）。
JavaWeb的三大组件：Servlet,Filter,Listener。
```

一、定义

```
    Servlet是运行在服务器中的小程序，它是动态资源，服务器会把接受到的动态资源请求交给Servlet处理。主要作用就是接受请求、处理请求、响应请求。它本质是一个Java接口，定义了被服务器访问到的规则，它的实现类不需要new对象，就能自动运行。其实就是通过服务器自动创建Servlet对象并调用其方法.
    Servlet实现类需要我们自己编写.每个Servlet必须实现javax.servlt.Servlet接口.而Servlet对象它驻留在服务器内存中.
```

二、实现接口

1、步骤

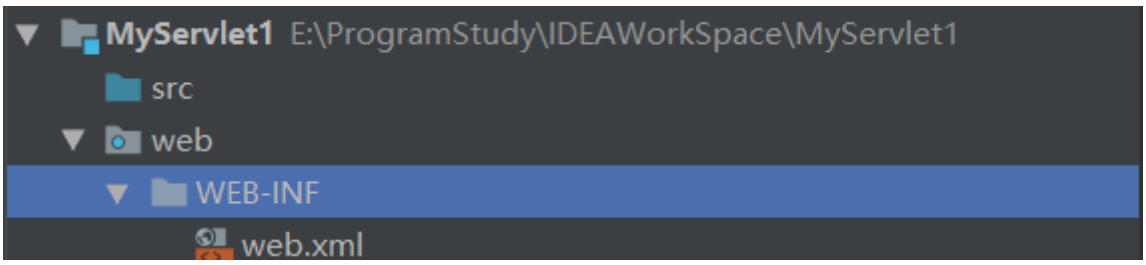
```
    ①创建JavaEE项目

    ②定义一个类实现如下接口

        public class ServletDemo1 implements Servlet

    ③实现接口中的抽象方法

    ④配置Servlet的url路径

    ⑤浏览器输入地址即可访问到此类.
```

2、两种Servlet的url路径配置

```
    在Servlet2.5规范的配置环境下，只能通过xml配置，而在Servlet3.0规范后，就可以进行注解配置 。
```

①XML配置法：2.5

(1)找到下面目录并打开web.xml文件



(2)进行文件配置

```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>标签装有name标签和类标签,<-->
    <servlet>

        <!-->与servlet-mapping的Servlet-name内容相对应<-->
        <servlet-name>
            ServletDemo1
        </servlet-name>

        <!--> 添加ServletDemo1时，必须全类名<-->
        <servlet-class>
            Servlet.ServletDemo1
        </servlet-class>

    </servlet>

    <!-->servlet-mapping是用于servlet的路径映射配置<-->
    <servlet-mapping>

        <!--><servlet-name>标签对应<-->
        <servlet-name>
        ServletDemo1
        </servlet-name>

        <!-->资源访问路径，浏览器输入IP:端口号/虚拟路径/此路径就可以进行访问<-->
        <url-pattern>
        /demo1
        </url-pattern>

    </servlet-mapping>


</web-app>
```

②Servlet3.0规范：注解配置法

好处：不用写xml来进行配置，只需要在实现类上写注解即可.

步骤

1、创建JavaEE项目，选择Servlet的版本3.0以上即可，可以不用创建web.xml，因为后期用注解

2、定义一个类，实现Servlet接口

3、复写方法

4、在类上使用@WebServlet注解，进行配置

配置WebServlet:@WebServlet("资源路径")

如：@WebServlet(urlPatterns="/demo")

```java
import java.io.IOException;
@WebServlet(urlPatterns = "/demo")
public class ServletDemo1 implements Servlet {
    @Override
    public void init(ServletConfig servletConfig) throws ServletException {
        System.out.println("s");
    }
    @Override
    public ServletConfig getServletConfig() { return null; }
    @Override
    public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws ServletException, I
        System.out.println("servlet3.0");
    }
    @Override
    public String getServletInfo() { return null; }
    @Override
    public void destroy() {

    }
}
```

注解内部结构：注解里有value属性，把重要的值赋值给它，最重要的是urlPatterns，所以可以把urlPatterns赋值给value,又因为value当只有一个属性性值的时候，value可以省略，于是，注解就可以变成了@WebServlet("/demo")

```java
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface WebServlet {
    String name() default "";

    String[] value() default {};

    String[] urlPatterns() default {};

    int loadOnStartup() default -1;

    WebInitParam[] initParams() default {};

    boolean asyncSupported() default false;

    String smallIcon() default "";

    String largeIcon() default "";

    String description() default "";
```

③注解配置法补充

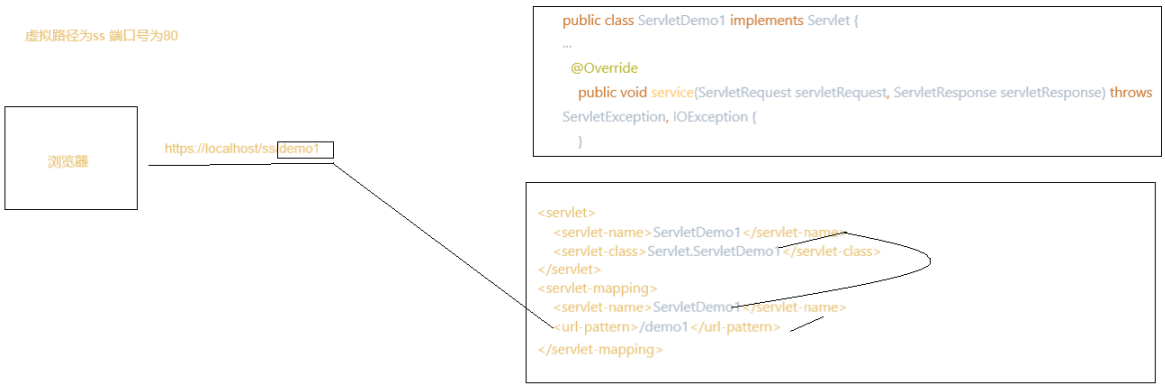
```
        1、一个Servlet可以定义多个访问路径，  通过定义不同路径，可以用不同路径访问同一个资源            如:@WebServlet({"/d4","/ddd","/d2d"})
        2、单路径不同的配置方式：
            ① /xxx：路径匹配
            ② /xxx/xxx:多层路径，目录结构
                /*优先级最低，输入路径如果找不到就访问到此资源
            ③ *.do：扩展名匹配
```

三、Servlet方法

```java
    //初始化，当此对象被创建时，调用此方法，只执行一次
    public void init(ServletConfig servletConfig) ;

    //获取ServletConfig对象，Servlet的配置对象
    public ServletConfig getServletConfig();
```

```
    //用于执行代码的方法.
    public void service(ServletRequest servletRequest, ServletResponse
servletResponse) ;

    //获取servlet的一些信息，版本，作者等等.了解即可.
    public String getServletInfo() ;

    //销毁方法当服务器关闭时，执行此方法.
    public void destroy() ;

    Servlet对象创建后，会一直在服务器内存中，当服务器关闭，对象才会摧毁.
```

四、Servlet原理




```
1、服务器接收到浏览器请求后，解析请求url路径。

2、服务器查找web.xml文件，是否有对应的<url-pattern>标签体内容

3、如果有，则通过<servlet-name>找到对应的<servlet-class>标签内容的全类名

4、标签内写全类名的目的是为了反射，tomcat将通过全类名将类加载进内存Class.forName()，并
new对象xx.newInstance()，接着就进行初始化(初始化init只执行一次)并调用service方法(因为
浏览器请求了Servlet所以就会调用Service方法)，那怎么确定一定会有service方法？因为该类实现
了Servlet接口，如果没有就，就会编译报错.那又怎么知道是Servlet类呢？因为<servlet-class>
标签会对类进行验证，如果不是Servlet就会报错.
```

ServletDemo1.java

```java
import java.io.IOException;

public class ServletDemo1 implements Servlet {
    @Override
    public void init(ServletConfig servletConfig) throws ServletException {
        System.out.println("s");
    }
    @Override
    public ServletConfig getServletConfig() {
        return null;
    }
    @Override
    public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws ServletException, IOExce
            System.out.println("你好");
    }
    @Override
    public String getServletInfo() { return null; }
    @Override
    public void destroy() {

    }
```

web.xml

```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>ServletDemo1</servlet-name>
        <servlet-class>Servlet.ServletDemo1</servlet-class>
        <load-on-startup>1</load-on-startup>
        <!-->指定Servlet的创建时机
                1、第一次被访问时，Servlet被创建
                 负数
                2、服务器启动时，Servlet被创建.
                  0或正整数
         <-->
    </servlet>
    <servlet-mapping>
        <servlet-name>ServletDemo1</servlet-name>
        <url-pattern>/demo1</url-pattern>
    </servlet-mapping>
</web-app>
```

五、生命周期

1、被创建

执行init方法，只执行一次。默认情况下，第一次被访问，Servlet被创建

> 也可以配置创建Servlet对象的时机.
> 用<load-on-startup>标签指定创建时机。如果是负数，第一次被访问就会被创建。如果是整数，则服务器启动就会被创建.

```xml
<servlet>
    <servlet-name>ServletDemo1</servlet-name>
    <servlet-class>Servlet.ServletDemo1</servlet-class>
    <load-on-startup>1</load-on-startup>
    <!-->指定Servlet的创建时机
            1、第一次被访问时，Servlet被创建
             负数
            2、服务器启动时，Servlet被创建.
              0或正整数
     <-->
</servlet>
```

```
    而init方法只执行一次，说明一个servlet的实现类只存在一个对象.所以它是单例的，正因为是单例的，所以有线程安全问题.解决这个问题最好的方法就是不要创建成员变量，即使定义了成员变量，也不要对其修改值.
```

2、提供服务

```
Servlet被访问则执行Service方法.
```

3、销毁

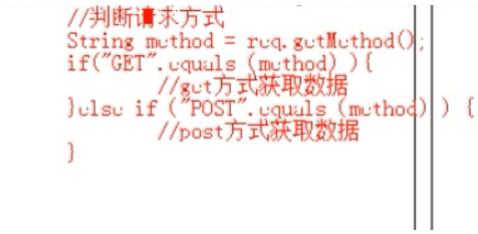
执行destroy方法，只执行一次.

```
    服务器正常关闭时，Servlet被销毁，Servlet销毁前会执行destroy方法.
```

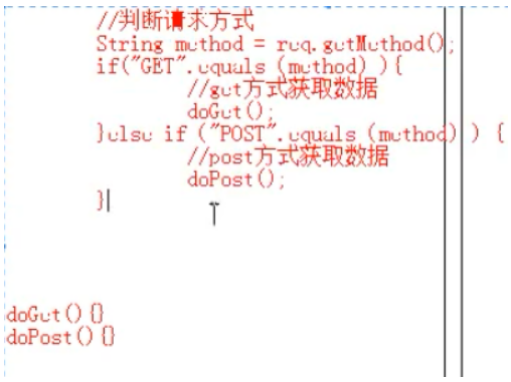
六、Servlet体系

Servlet -- 接口 |实现 GenericServlet -- 抽象类 |继承 HttpServlet -- 抽象类

```
为了简化操作，就出现了如下两个抽象类
    GenericServlet抽象类除了对Service方法作为抽象方法外，其他都是空实现，即我们继承GenericServlet类时，只需要实现Service方法即可。但是我们接受数据请求时，请求格式有很多种，那这样就要判断是哪种格式，写起来就像下面这样；
```




```
 所以HttpServlet抽象类就简化了这些操作，我们只需复写对应格式方法的实现即可

 HttpServlet就相当于这样,在Service写判断代码，从中调用doGet等方法...
```




```
收到哪种请求数据格式，就执行对应的方法
```

七、HTTP

1、定义

Hyper Textsfer protocol：超文本传输协议

```
    超文本是用超链接的方法，将各种不同空间的文字信息组织在一起.
    传输协议：统一了客户端和服务器通信时传输数据的规则。
```

2、特点

```
        1、基于TCP/IP的高级

        2、默认端口号：80

        3、基于请求/响应模型，即一次请求对应一次响应

        4、无状态，即每次请求之间相互独立，不能相互交互数据.
```

3、版本

```
            1.0版本：    每次请求都会建立新的连接，会导致消耗资源且影响传输速度.

            1.1版本：复用连接，传输完后它不会立即断开连接，而是等一会，如果有数据要发送，就复用刚才那个连接，如果没有，则断开连接。
```

4、请求消息

（1）请求方式

```
    请求方式：HTTP协议有7中请求方式，常用的有2种
                        POST：长度无限制，请求数据封装在请求体中
                        GET：长度有限制，请求数据在请求行中，即URL后边加？.
                        如http://bai.com?username=xxx&password=xxx
    除非你设置了Post方式，否则默认方式就会是GET方式。
```

（2）请求数据格式

①请求行：Get方式要用到请求行，Post则在请求体中。

```
            请求方式    URL  HTTP/1.1

            如：GET https//localhost/Wu/login.html    HTTP/1.1
```

②请求头：客户端浏览器告诉服务器一些基本信息（键值对格式）

格式：请求头名称: 请求头值

```
常见请求头
    //告诉服务器要访问哪个URL
    Host: localhost

    //浏览器告诉服务器，我访问你使用的浏览器版本信息
    //* 可以在服务器端获取该头的信息，解决浏览器的兼容性问题
☆   User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:60.0)
Gecko/20100101 Firefox/60.0
```

```
    //能接受到的请求格式，*/*什么格式都可以接受
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

    //可以支持的语言环境 中文-中国 中文-台湾等
    Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2:

    //接收压缩格式
    Accept-Encoding: gzip, deflate:
```

```
☆  Referer:告诉服务器我从哪里来
    如：当被某个网址访问时，就会得到要访问我的那个网址. http://localhost/login.html
        作用：
            1、防盗链
                可以用服务器端判断当前请求是否来自某个URL.如果是，则进入请求资源，如果不是，则响应一个信息过去.
            2、统计工作
                如，有一个网站，想要判断哪里的人来的多，就可以用Referer百度，新浪
if(baidu){baidu++}else{xinlan++};//伪代码.
            运用Referer：做一个Referer的统计工作。
```

```
    Connection: keep-alive ：表示连接状态，这里的意思是连接可以被复用

    Upgrade-Insecure-Requests: 1：升级信息
```

③请求空行

```
    空行，就是用于分割POST请求的请求头，和请求体的。为了排版。
```

④、请求体（正文）

```
封装POST请求消息的请求参数的
参数=xxx 被封装成 参数：xxx
```

5、响应消息

【1】定义

```
    服务器端发送给客户端的数据
```

【2】响应消息数据格式

（1）响应行

```
        ①组成：协议/版本 状态码 状态码描述

        ②响应状态码：服务器告诉客户端本次请求和响应的一个状态

            (1)响应码都是3位数

            (2)响应码分类
```

```
1XX：服务器接受客户端消息，但没有接受完成，服务器以为客户端还有消息来，于是等待一段时间后，最后还没有等到，就发送1xx多状态码，意思是你还有没有消息来.

2XX：成功。代表：200

3xx：重定向。代表：302(重定向)，304(访问缓存)：服务器告诉浏览器本地里有下载好的，去访问缓存.浏览器就会自己去找缓存

4XX:客户端错误。

    * 代表：404（请求路径没有对应的资源）
      405：请求方式没有对应的doXxx方法

5XX:服务器端错误。代表：500(服务器内部出现异常)：服务器内部代码错误.



(3)状态码描述：描述状态码的意思

    如200 OK：即200是响应成功了的意思

    100 continue：100在等待.
```

（2）响应头

①格式：头名称：值

②常见的响应头

I、Content-Type：服务器告诉客户端本次响应体数据格式以及编码格式（两个参数可以只写其中一个XXX;XXX)

```
如Content-Type：text/html;charset=UTF-8

当前响应体是文本内容并且是html内容。那么浏览器收到这个消息后，会自动的拿html解析引擎来解析响应体内容.并且指定了UTF-8编码，那浏览器就会自动将当前页面编码设置为UTF-8;
```

II、Content-disposition：服务器告诉客户端以什么形式打开响应体数据。

```
如果没有设置这个头就会用默认值：in-line，在当前页面打开.也可以设置其他值，如attachment;filename=xxx（文件名称）：以附件形式打开响应体.文件下载会用到这个响应头，filename的意思是下载好的文件文件名就是filename.
```

III、Refresh：自动刷新浏览器， N秒之后刷新本页面或访问指定页面 .

```
    Rfresh:秒;URL
    如果只有秒属性，则说明几秒后刷新.如果秒和URL都有，则说明几秒后跳转到指定页面
```

（3）响应空行

（4）响应体

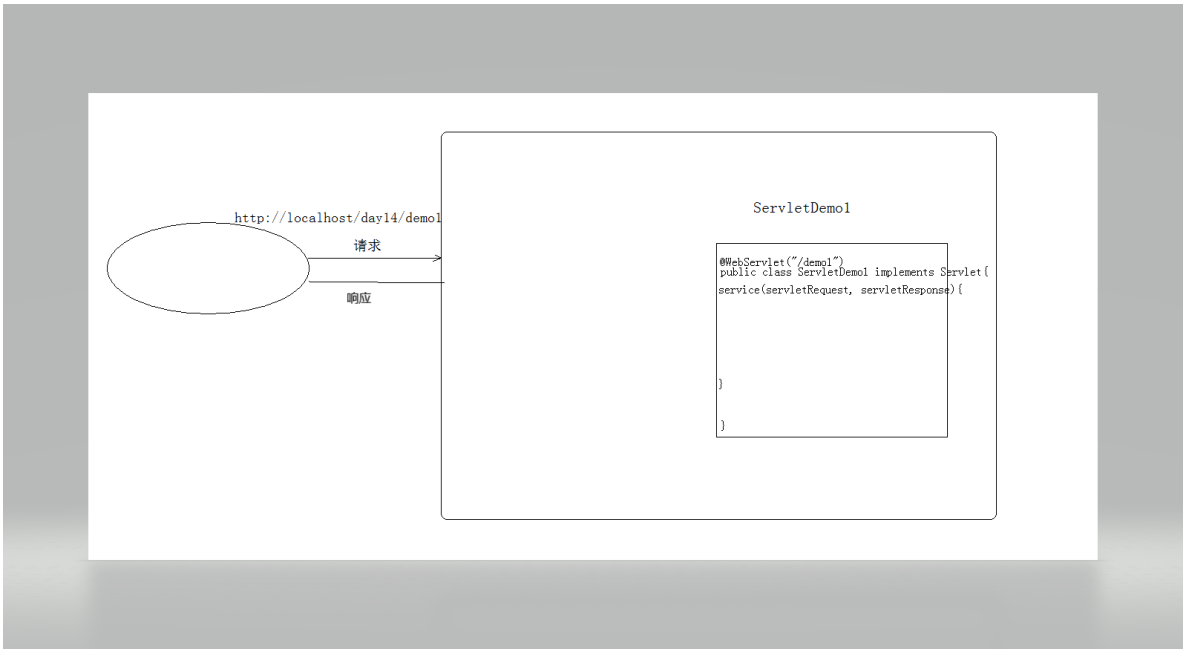
```
        传输的数据.
```

响应体例子:

```
响应字符串格式
HTTP/1.1 200 OK
    Content-Type: text/html;charset=UTF-8
    Content-Length: 101
    Date: Wed, 06 Jun 2018 07:08:42 GMT
<html>
  <head>
    <title>$Title$</title>
  </head>
  <body>
  hello , response
  </body>
</html>
```

八、Request

1、Request与Response对象原理




```
        1. 服务器在接收到客户端的请求之后，会创建request对象和response对象

        2.服务器会通过request对象把客户端的数据，包括请求信息都封装到这个对象里面

        3.然后servlet的service方法通过request对象得到数据，并对数据进行相应的业务处理，最后响应给客户端结果

        4.这个结果我们是通过response来封装的，而response不是直接输出到客户端，而是将数据存入到response缓冲区了，再由服务器去response缓冲区里把所有数据响应给客户端.

        5.当这整个过程结束之后，request和response对象的周期也就结束了，他们的生命范围就是用户的一次请求和得到的一次结果的反馈。

```

```
1. request和response对象是由服务器创建的。
2. request对象是来获取请求消息，response对象是来设置响应消息
```

2、Request体系结构

（1）Request对象继承体系结构

ServletRequest -- 接口 | 继承 HttpServletRequest -- 接口 | 实现 org.apache.catalina.connector.RequestFacade 类(tomcat编写的)

（2）Request功能

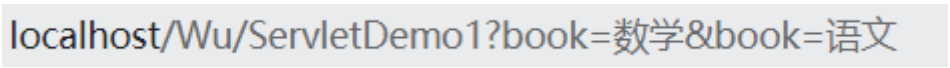
【1】获取请求消息数据

①

```
    getParameter(String name)：通过输入键名，获取值(如果是一个键名对应多个，则输出第一个值)，如果没有对应的键，则返回null，在表单提交中，如果有键，但没有值，则返回空字符串.

    getParametervalues(String  name)：用于多个相同名字复选框中，获取多个值.返回一个数组.一个键名，可能对应多个值,如果没有对应的值，则返回null

    getParameterMap():框架中使用，返回值是一个带有<String,String[]>泛型的Map对象.调用这个方法不用担心Map空指针异常。如果提交的表单都没有键值，request还是会为map赋值.如果提交的键，没有值，则获取到的值是空字符串.
```

②使用

```
<form action="/Wu/ServletDemo1" method="get">
    <input type="checkbox" name="book" value="数学">
    <input type="checkbox" name="book" value="语文">
    <input type="submit" id="sub" value="我的选择">
</form>
```



```java
public class ParameterServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExce
        //使用getParameter获取
        String parameter = request.getParameter( s: "book");
        //使用getParameterVlues方法获取
        String[] books = request.getParameterValues( s: "book");
        //使用getParameterMap方法获取
        Map<String, String[]> parameterMap = request.getParameterMap();
        //输出getParameter的值
        System.out.println(parameter);
        //防止空指针异常
        //遍历getParameterVlues获得的值.
        for(int i=0;(books!=null)&&(i<books.length);i++){
            System.out.println(books[i]);
        }
        //遍历Map的值
        Set<String> keySet = parameterMap.keySet();
        for(String name:keySet){
            String[] values = parameterMap.get(name);
            System.out.print(name+":");
            for(String value:values){
                System.out.print(value+" ");
            }
            System.out.println();
        }
```

```
数学
数学
语文
book:数学 语文
```

【2】获取请求行数据（五角星的方法是重点）

下面的方法基于这个例子：localhost/Wu/RequestDemo1?a=b

```
    ①String   getMethod()：获取请求方式 :   Get

    ②String   getContextPath():获取虚拟目录（☆）:   /RequestDemo1

    ③String   getRequestURI():获取虚拟目录以及后面的地址（☆）:
/Wu/RequestDemo1

    ④StringBuffer getRequestURL()：获取要请求的完整URL.（☆）:
http://localhost/Wu/RequestServletDemo1

    ⑤String getServletPath():获取Servlet路径 : /RequestDemo1

    ⑥String getQueryString():获取get方式请求参数：a=b

    ⑦String getProtocol():获取http协议及版本  : HTTP/1.1
    ⑧String getRemoteAddr()：获取客户机的IP地址
```

【3】获取请求头数据

String getHeader(String name):通过请求头的名称获取请求头的值

Enumeration<String> getHeaderNames():获取所有请求头名称

Enumeration其实是一个迭代器，方法跟迭代器一样。图片是Enumeration的使用例子.

（1）getHeaderNames方法的应用

```
@WebServlet("/RequestServletDemo2")
public class RequestServletDemo2 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExcept
        Enumeration<String> names=request.getHeaderNames();
        while(names.hasMoreElements()){
            String name=names.nextElement();
            String header = request.getHeader(name);
            System.out.println(name+":"+header);
        }
    }
```

```
host:localhost
connection:keep-alive
cache-control:max-age=0
upgrade-insecure-requests:1
user-agent:Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.
sec-fetch-user:?1
accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,applicati
sec-fetch-site:none
sec-fetch-mode:navigate
accept-encoding:gzip, deflate, br
accept-language:zh-CN,zh;q=0.9,en;q=0.8
cookie:JSESSIONID=83BDAAD41657A58ED58AAA70910D380A; Idea-aef28401=1749984e-972b-4585-93f0-74b20e45f74c
```

上面获取不到referee，因为referee是告诉服务器从哪里来，但我没有从别的地方来，我是直接访问服务器.所有获取到的是null.想要获取referee，要通过某个资源链接点到RequestServletDemo1中才行.

（2）user-agent请求头的应用：用于版本兼容。

```
//演示使用请求头数据：user-agent
String agent=request.getHeader("user-agent");
//判断agent的浏览器版本
if(agent.contains("Chrome")){
    //谷歌
    System.out.println("谷歌来了");
}else if(agent.contains("Firefox")){
    //火狐
    System.out.println("火狐来了");
}
```

（3）referee防盗链的应用

```
        String referer = request.getHeader("referer");
            System.out.println(referer);
            //防盗链应用
            if(referer!=null){
               if(referer.contains("/b.html")){
                  System.out.println("欢迎观看");
               }else{
                  System.out.println("请从XXX网站进入");
               }
            }
```

① b.html去访问ReferServlet.

```
<a href="http://localhost/Wu/RefererServlet">s</a>
```

②代码部分

```
@WebServlet("/RefererServlet")
public class RefereeServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExcept
        String referer = request.getHeader( s: "referer");
        System.out.println(referer);
        //防盗链应用
        if(referer!=null){
        if(referer.contains("/b.html")){
            System.out.println("欢迎观看");
        }else{
            System.out.println("请从XXX网站进入");
        }
        }
    }
```

③输出台结果

```
http://localhost/Wu/b.html
欢迎观看
```

【4】获取请求体数据

（1）

```
请求体：只有POST请求方式，才有请求体，在请求体中封装了POST请求的请求参数.

Request对象将请求体封装为流.

    获取请求体步骤：

        1、获取流对象

            如果请求体是字符的数据，如username=zhangsan，那可以获取字符输入流

            如果请求体是除了字符的数据，那就用字节数据流

            BufferedReader getReader():获取字符输入流，用于操作字符数据

            ServletInputStream getInputStream():获取字节输入流，可以操作所有类型数据.
```

2、再从流对象中取数据

3、演示

```
@WebServlet("/PostServletDemo")
public class PostServletDemo extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExc
        BufferedReader reader = request.getReader();
        String value=null;
        //因为流是request对象的，所以不用关闭
        while((value=reader.readLine())!=null){
            System.out.println(value);
        }
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExc
```

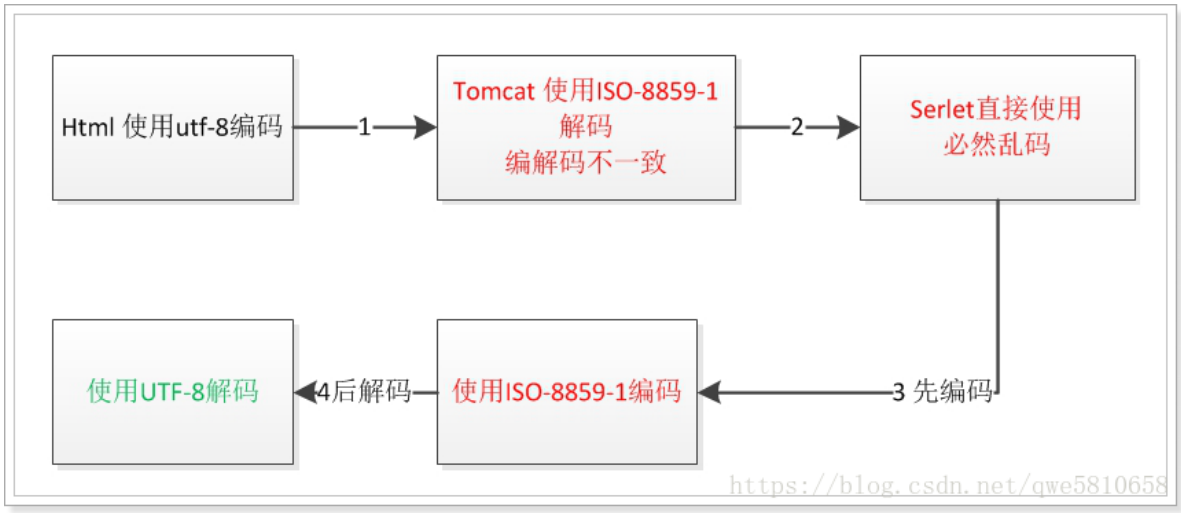
请求参数会放在一行里面.

```
username=zhangsan&password=123
```

```
<form action="/Wu/PostServletDemo" method="post">
    <input type="text" name="username">
    <input type="password" name="password">
    <input type="submit" value="提交">
</form>
```

问题：getReader读取到的数据乱码问题，即使设置了编码方式，它还是会乱码，而用getParameter的方式就不会乱码.

(2)请求参数中文乱码问题



关于编码解码的个人理解：我们写了个html网页，并设置了编码方式，浏览器就会按照这个编码方式去解析，然后呈现页面.但如果没有设置编码，浏览器就会按照默认方式去进行解析.

如果客户端请求服务器并传了参，Tomcat将会按照不同的请求方式去进行解码，Get就以UTF-8去解，POST就以其他解码方式去解.如果编码解码不一致，则会产生乱码.

Get方式：Tomcat8将Get方式的解码方式设置为UTF-8

Post：Tomcat8没将解码方式设置为UTF-8所以Post方式会乱码

解决：根据请求的客户端的编码方式来设置编码方式，如客户端编码方式是UTF-8，则在获取参数前，设置request的编码方式request.setCharacterEncoding("UTF-8");这样解码就会用UTF-8来解码.

【5】Request请求转发

(1) 定义：一种在服务器内部的资源跳转方式.转发其实就是转发给另一个Servlet处理。

(2) 步骤

①通过request对象获取请求转发对象：RequestDispatcher getRequestDispatcher（String path）

②通过RequestDispatcher对象来进行转发：forward(ServletRequest request,ServletResponse response).

(3) 链式编程实现

```
<a href="/Wu/DispatcherDemo1">访问转发....</a>
```

访问转发....

localhost/Wu/DispatcherDemo1

```
@WebServlet("/DispatcherDemo1")
public class DispatcherDemo1 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExc
        System.out.println("DispatcherDemo1被访问了....");
        request.getRequestDispatcher( s: "/DispatcherDemo2").forward(request,response);

    }
```

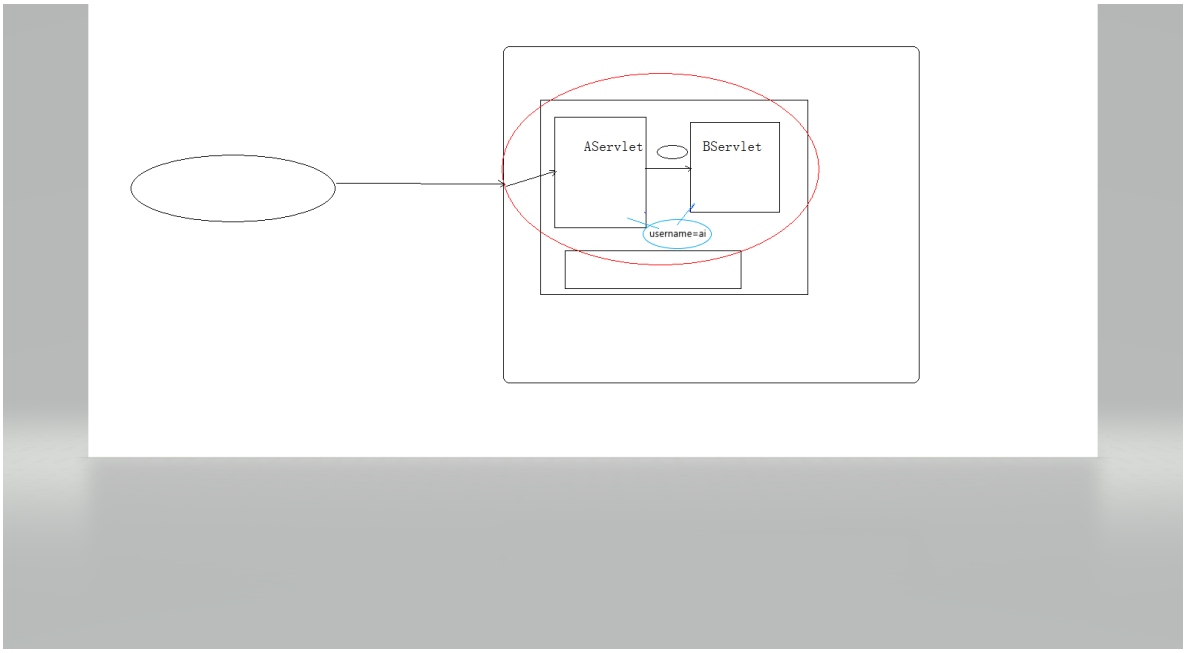
```
@WebServlet("/DispatcherDemo2")
public class DispatcherDemo2 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExc
        System.out.println("DispatcherDemo2被访问了...");
    }
```

(4) 特点

①浏览器地址栏路径不发生变化

②只能转发到当前服务器内部Servlet资源中

③转发是一次请求.



客户端请求服务器资源，服务器里的AServlet资源转发给BServlet资源处理，最终BServlet处理完后，响应给客户端，但结果只有一次请求.AServlet和BServlet在一次请求范围之中，因此可以共享数据。

【6】共享数据：

域对象定义：一个有作用范围的对象，可以在范围内共享数据.

request域：代表一次请求的范围，一般用于请求转发的多个资源中共享数据.范围一次请求内.

①void setAttribute(String name,Object object):存储数据.

如：上图的AServlet接受到username=ai后，转发给BServlet时，要先设置setAtrribute去存储数据，这样BServlet调用getAttribute才能得到数据

②Object getAttribute(String name):通过键获取值

③void removeAttribute(String name):通过键移除键值对.

localhost/Wu/DispatcherDemo1?username=ai

```java
@WebServlet("/DispatcherDemo1")
public class DispatcherDemo1 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExcep
        String username = request.getParameter( s: "username");
        request.setAttribute( s: "username",username);
        request.getRequestDispatcher( s: "/DispatcherDemo2").forward(request,response);

    }
```

```java
@WebServlet("/DispatcherDemo2")
public class DispatcherDemo2 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExce
        Object username = request.getAttribute( s: "username");
        System.out.println(username);
    }
```

【7】获取ServletContext:

ServletContext getServletContext()

【8】转发请求后是否能执行后面的代码?

能，在本页面代码执行到转发语句后，即跳转到指定的页面执行其他代码，执行完毕后返回接着执行转发语句后的代码。但是不会执行原页面后面的response代码.

```java
@WebServlet("/DispatcherDemo1")
public class DispatcherDemo1 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Servle
        request.getRequestDispatcher( s: "/DispatcherDemo2").forward(request,response);
        System.out.println("转发完后会执行到原页面代码");

    }
```

```java
@WebServlet("/DispatcherDemo2")
public class DispatcherDemo2 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletR
        System.out.println("转发...");
    }
```

```
转发...
转发完后会执行到原页面代码
```

情况二：转发后，原页面后面的response代码不会执行到，因为转发给DispatcherDemo2，就由它来响应给浏览器请求.

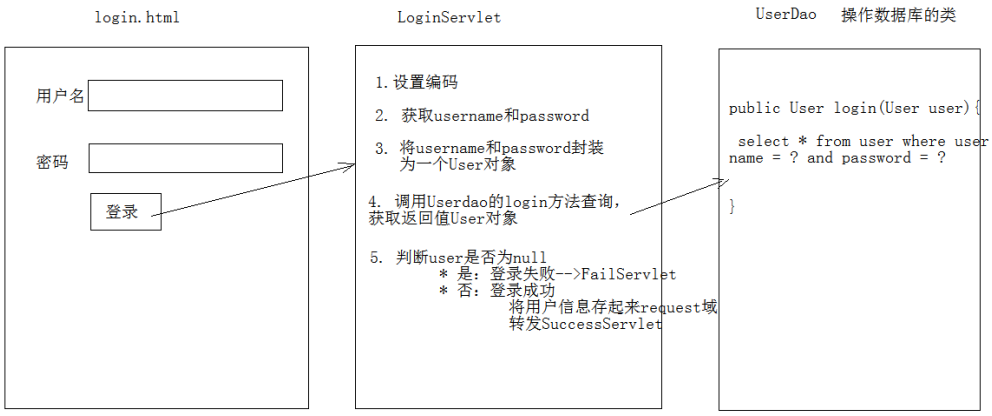
```java
@WebServlet("/DispatcherDemo1")
public class DispatcherDemo1 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throw
        request.getRequestDispatcher( s: "/DispatcherDemo2").forward(request,response);
        response.getWriter().write( s: "不会执行原页面response代码");
        System.out.println("转发后，后面的response不会执行，但可以执行其他代码");
    }
```

```java
@WebServlet("/DispatcherDemo2")
public class DispatcherDemo2 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Servle
        System.out.println("转发...");
        response.getWriter().write( s: "ToWriteWeb");
    }
```

结果： ToWriteWeb

```
转发...
转发后，后面的response不会执行，但可以执行其他代码
```

【9】案例：用户登录




```
1.编写login.html登录页面
    username & password 两个输入框
2.使用Druid数据库连接池技术,操作mysql，day14数据库中user表
3、创建LoginServlet.
4、创建User类，并创建UserDao用JdbcTemplate操作数据库.
5.登录成功跳转到SuccessServlet展示：登录成功！用户名,欢迎您
6.登录失败跳转到FailServlet展示：登录失败，用户名或密码错误
```

```
下图的问题：LoginServlet有一个地方做的不好就是，如果以后要获取很多个参数，则要调用getParameter方法很多次，将参数赋值给User，也要写很多次.为了简化操作，就有BeanUtils工具类出现.
```

修改前：

```java
@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        String username = request.getParameter( s: "username");
        String password=request.getParameter( s: "password");
        User loginUser=new User();
        loginUser.setUsername(username);
        loginUser.setPassword(password);
        UserDao dao=new UserDao();
        User user = dao.login(loginUser);
        if(user!=null){
            request.setAttribute( s: "user",user);
            request.getRequestDispatcher( s: "/SuccessServlet").forward(request,response);

        }else{
            request.getRequestDispatcher( s: "/FailedServlet").forward(request,response);
        }

    }

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

修改后:

```java
@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        Map<String, String[]> map = request.getParameterMap();
        User loginUser=new User();
        try {
            BeanUtils.populate(loginUser,map);
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (InvocationTargetException e) {
            e.printStackTrace();
        }
        UserDao dao=new UserDao();
        User user = dao.login(loginUser);
        if(user!=null){
            request.setAttribute( s: "user",user);
            request.getRequestDispatcher( s: "/SuccessServlet").forward(request,response);

        }else{
            request.getRequestDispatcher( s: "/FailedServlet").forward(request,response);
        }

    }
```

```java
public class User {
    private int id;
    private String name;
    private String gender;
    private int age;
    private String address;
    private String qq;
    private String email;
    private String username;
    private String password;

    public User() {
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", gender='" + gender + '\'' +
                ", age=" + age +
                ", address='" + address + '\'' +
                ", qq='" + qq + '\'' +
```

```java
public class UserDao {
    private JdbcTemplate template=new JdbcTemplate(JDBCUtils.getDataSource());
    public User login(User loginUser){

        //！！！这里是没有发生异常的，用try...catch的目的是防止queryForObject查询不到数据而报错.
        try {
            String sql = "select * from user where username = ? and password = ?";
            User user = template.queryForObject(sql, new BeanPropertyRowMapper<User>(User.class),
                    loginUser.getUsername(), loginUser.getPassword());
            return user;
        } catch (Exception e) {
            return null;
        }
    }
}
```

```java
@WebServlet("/FailedServlet")
public class FailedServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("登录失败");
    }
```

```java
@WebServlet("/SuccessServlet")
public class SuccessServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        User user =(User) request.getAttribute( s: "user");
        System.out.println(user);
    }
```

九、Response

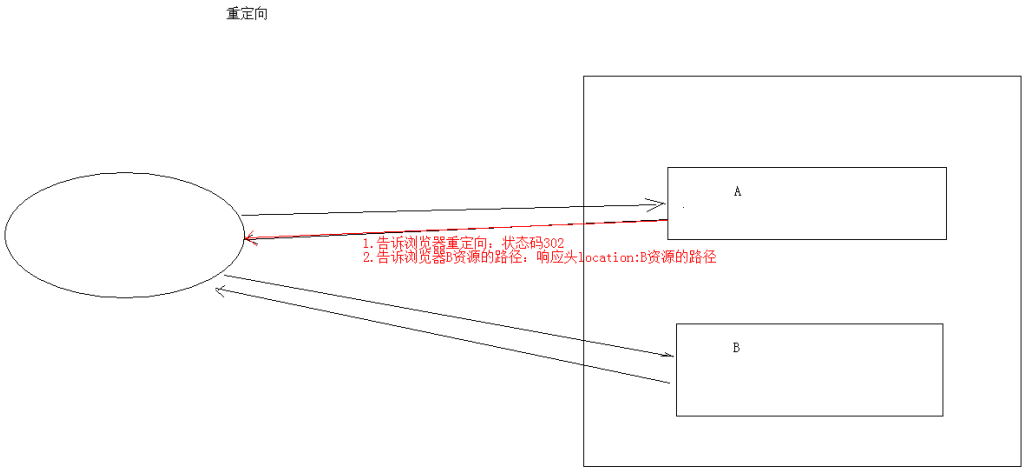
1、定义

response对象用于响应数据给客户端，但是它不是直接输出给客户端，而是将数据弄到response缓冲区，等Service（或doPost、doGet）方法结束，服务器就会将response缓冲区的所有响应数据传给客户端。

2、方法

```
    (1)响应行

        setStatus(int sc):设置状态码

    (2)响应头
        setHeader(String name,String value):设置响应头

        setContentType(String str)：告诉客户端用什么编码以及什么格式解析

    (3)响应体
        PrintWriter getWriter() :获取字符输出流;只能输出字符到浏览器

        ServletOutputStream getOutputStream()：获取字节输出流；可输出任意数据到浏览器.

        setCharacterEncoding(String code)设置字符流的编码方式


```

3、重定向



(1) 定义

客户端请求A资源时，A资源让浏览器去访问B资源，并且A资源给浏览器一个302状态码和B资源路径.浏览器就去请求B资源。

(2) 特点

```
①地址栏地址会发生变化

②重定向可以访问其他服务器的资源(此时要写全URL.)

③重定向是两次请求。不能使用request对象来共享数据
```

(3) 重定向方法

```
sendRedirect(String var1):重定向
```

(4) 重定向操作方式

以后就用第二种.

```
@WebServlet("/AnotherServlet1")
public class AnotherServlet1 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExce
        //第一种
        response.setStatus(302);
        response.setHeader( s: "location", s1: "/Wu/AnotherServlet2");
        //第二种
         response.sendRedirect( s: "/Wu/AnotherServlet2");
    }
```

(5) 重定向之后还会执行后面代码吗?

```
重定向之后的代码会继续执行
当前程序所有代码执行完毕后,才会执行重定向跳转。
```

4、操作输出流与常见问题

```
输出流的write()方法，如果不用ajax接收将数据放在合适的位置，就会在浏览器上生成一个新的页面来显示内容。write方法里可以写标签，到时候浏览器自动解析这些标签.
```

(1) 字符输出流的操作步骤

```


① （1）告诉浏览器要使用的解码方式.

（2）获取字符输出流。在Tomcat中，字符输出流的默认编码方式为ISO-8859-1

（3）设置默认编码方式。但其实只要设置了setContentType("text/html;charset=utf-8")，它不仅会向浏览器告诉要用UTF-8来解码，还会将Tomcat内部的码表设置为UTF-8码表。所以设置默认编码setCharacterEncoding方式可用可无。（setContentType必须设置MIME属性。）

（4）输出到浏览器。

② 操作
```

```java
@WebServlet("/ToResponse")
public class ToResponse extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOEx
        //告知浏览器要使用的编码方式，不区分大小写
        response.setContentType("text/html;charset=utf-8");
        //设置流的编码，将默认的ISO-8859-1 设置为utf-8
        response.setCharacterEncoding("UTF-8");
        //打印输出字符流，无需刷新
        PrintWriter pw = response.getWriter();
        pw.write( s: "你好");
        //能写html标签
        pw.write( s: "<h1>你好啊 Response</h1>");
    }
```

你好

你好啊 Response

字符输出流中文乱码问题：Tomcat默认的编码是ISO 8859-1，输出数据时，Tomcat会依据ISO 8859-1码表给我们的数据编码，再输出到浏览器，而中文不支持这个码表，所以就出现了乱码问题.

(2) 字节输出流的操作步骤

①

（1）告知浏览器编码方式

（2）获取字节输出流

（3）输出数据前获取字节数组并对字节数组进行编码，而用getBytes()获取到字节数组的默认是GBK编码的字节数组，但是为了程序对各个国家语言通用性，应该用utf-8来编码.

② 操作

```java
<a href="/Wu/ResponseDemo1">response</a>
```

```java
    //告知浏览器要使用的编码方式
     response.setContentType("text/html;charset=utf-8");
     //获取字节输出流
    ServletOutputStream outputStream = response.getOutputStream();
    //输出
    outputStream.write("<h1>你好 Response</h1>".getBytes( charsetName: "UTF-8"));
}
```

你好 Response

---

(3) 总结输出流中文乱码问题(获取流之前设置)

步骤：①告知浏览器用什么编码解析

```
        方式1：response的setHeader("content-type","text/html;charset=某编码");

        方式2：使用 response的setContentType(String str);
                  setContentType("text/html;charset=某编码");

          ②    【1】字符输出流
                  由于设置了setContentType("text/html;charset=某编码")，所以就可以解决中文乱码.

                【2】字节输出流

                  对getByte进行设置。

        ③用字符输出流的write方法输出.
```

5、路径

（1）路径分类

① 相对路径

```
规则：确定访问当前资源和目标资源之间的相对位置关系.
```

不以"/"开头的

```
      例如：http://localhost:8080/hello1/pages/a.html中的超链接和表单如下：
```

```
相对路径:
<formaction= "index.html">
  <input type= "submit"value= "表单3"/>
</form>
```

```
        相对当前页面的路径，即最终访问的路径为：
        http://localhost:8080/hello1/pages/index.html；
    ② 绝对路径：通过绝对路径可以确定唯一资源，以/开头的路径称为绝对路径.
    如：http://localhot/day15/responseDemo2，可以简化为/day15/responseDemo2.
```

（2）虚拟路径加不加?

①是否要加虚拟路径，主要是判断定义的路径是给谁用的？判断请求从哪儿发出.

I、客户端浏览器使用：需要加虚拟目录

如 a 标签 form表单 重定向

II、给服务器使用：不需要加虚拟路径(如：请求转发)

```
        重定向要加虚拟目录，是因为浏览器请求AServlet后，AServlet告诉浏览器BServlet地址，接着浏览器就按照URL去找BServlet,所以是由浏览器发出的请求.
```

（3）动态虚拟路径

request的getContextPath()：获取虚拟目录；（/XXX）

```
//动态获取虚拟目录
String contextPath = request.getContextPath();

//简单的重定向方法
response.sendRedirect( s: contextPath+"/responseDemo2");
```

```
1    动态虚拟目录的出现就是为了防止将来改虚拟目录后，原先的路径就不能访问了，所以才会有动态虚拟目录的出现。其中html页面不能写动态虚拟目录，但JSP可以。
```

（4）结论

①强烈建议使用“/”开头的路径

②超链接、表单、重定向：以“/”开头的的路径相对于主机根目录【http://localhost:8080/】

转发、包含、：以“/”开头的的路径相对项目根目录【http://localhost:8080/项目名称/】

③、注意：不带“/”的相对路径，是相对于访问到当前文件的路径，而不是当前文件所在的目录。

6、Response注意要点

（1）互斥

```
1    对于一次请求，Response的getOutputStream方法和getWriter方法是互斥，只能调用其一，特别注意forward后也不要违反这一规则。
```

（2）response输出原理

```
1    利用Response输出数据的时候，并不是直接将数据写给浏览器，而是写到了Response的缓冲区中，等到整个service方法执行完后，由服务器拿出response中的信息组成响应消息返回给浏览器。
```

（3）流关闭

```
1    service方法执行完后，服务器会自己检查Response获取的OutputStream或者Writer是否关闭，如果没有关闭，服务器自动帮你关闭，一般情况下不要自己关闭这两个流。
```

7、常见应用

（1）实现一个验证码

（2）自动刷新

(1)几秒自动刷新页面

```
1    以规定的时间让页面刷新，更新资源，让浏览器实现自动刷新，那肯定又是修改消息头了。
2
3    response.setHeader("Refresh", "3");将响应头Refresh修改为X秒，此处为3秒.每三秒自动刷新，即每三秒就向服务器请求.
```

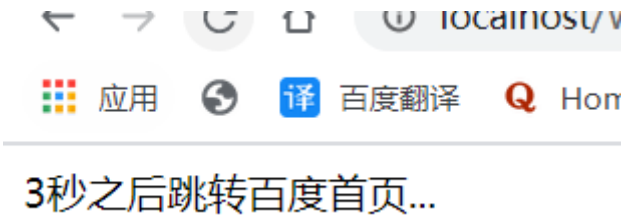
```java
@WebServlet("/AutoFresh")
public class AutoFresh extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throw
        response.setHeader( s: "refresh", s1: "3");
        response.getWriter().write( s: System.currentTimeMillis()+"");
    }
```

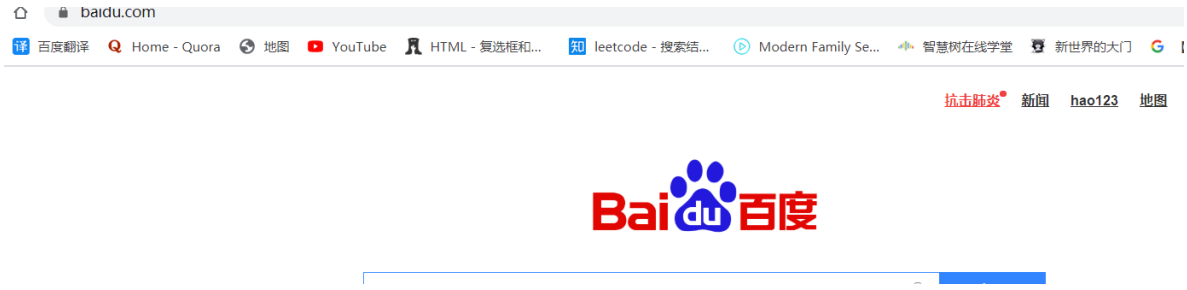
1582627513736

1582627526403

（2）几秒跳转到指定页面

```java
@WebServlet("/AutoFresh")
public class AutoFresh extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)

        response.setContentType("text/html;charset=utf-8");
        response.getWriter().write( s: "3秒之后跳转百度首页...");
        response.setHeader( s: "refresh", s1: "3;https://www.baidu.com/");
    }
```





（3）禁止缓存

十、ServletContext

1、定义

> 它是一个接口，由getServletContext()方法实现.代表整个web应用（工程），可以和程序的容器(服务器)来通信，它的域范围是整个web项目.（应用范围），

2、生命周期

> 服务器创建而创建，服务器销毁而销毁，用这个对象要谨慎，因为在里面存储的数据多了，会对内存产生很大的压力。

3、获取方式

```
（1）通过request对象获取
   request.getServletContext();
（2）通过HttpServlet获取
   this.getServletContext();
只要在同一个服务器内，无论哪种方法获取到的ServletContext对象都是同一个的.
```

```
//法一，用request获取
ServletContext servletContext = request.getServletContext();
//法二，用继承HttpServlet的类来获取.
ServletContext servletContext1 = this.getServletContext();
System.out.println(servletContext==servletContext1);
```

4、MIME类型

（1）定义

```
在互联网通信过程中定义的一种文件数据类型
```

（2）格式

```
大类型/小类型

   如：text/html：纯文本的，里面定义的是html形式的.

       image/jpeg：图片，为jpeg类型的
```

（3）作用

```
用于告知浏览器要响应的数据是什么类型的数据，然后让浏览器用相应的解析引擎去解析他们.
```

5、功能：

（1）获取MIME类型

```
String  getMimeType(String file):根据文件后缀名获取相应的MIME类型
```

```
String file="a.jpg";
String file1="b.html";
ServletContext context = this.getServletContext();
System.out.println(context.getMimeType(file));
System.out.println(context.getMimeType(file1));
```

```
image/jpeg
text/html
```

（2）作为域对象去共享数据

```
        ①setAttribute(String name,Object value)

        ②getAttribute(String name)

        ③removeAttribute(String name)

 ServletContext对象域范围：共享所有用户所有请求的数据，一个用户请求存储数据后并退出，再来一次请求还能得到上一个用户存储的数据.
```

(3) 获取项目存储的真实路径

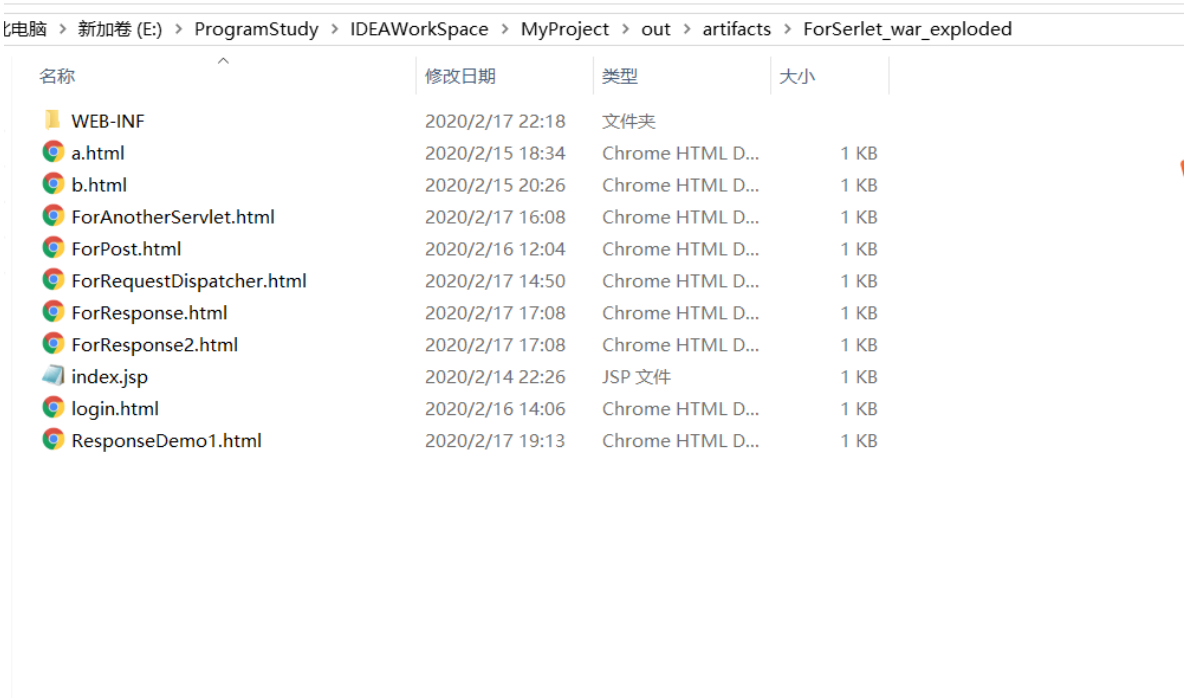
```
        String getRealPath(String path) :获取项目真实路径（部署后的）
        在使用此方法时，要以"/" 开头，否则会找不到路径，导致NullPointerException。它还可以获得文件夹!不管存不存在，只要逻辑上存在就可以获得!
```

I、以下都这个目录为基础

E:/ProgramStudy/IDEAWorkSpace/MyProject/out/artifacts/ForSerlet_war_exploded/

```
/代表着整个Web目录根路径
即/代表
E:/ProgramStudy/IDEAWorkSpace/MyProject/out/artifacts/ForSerlet_war_exploded
```

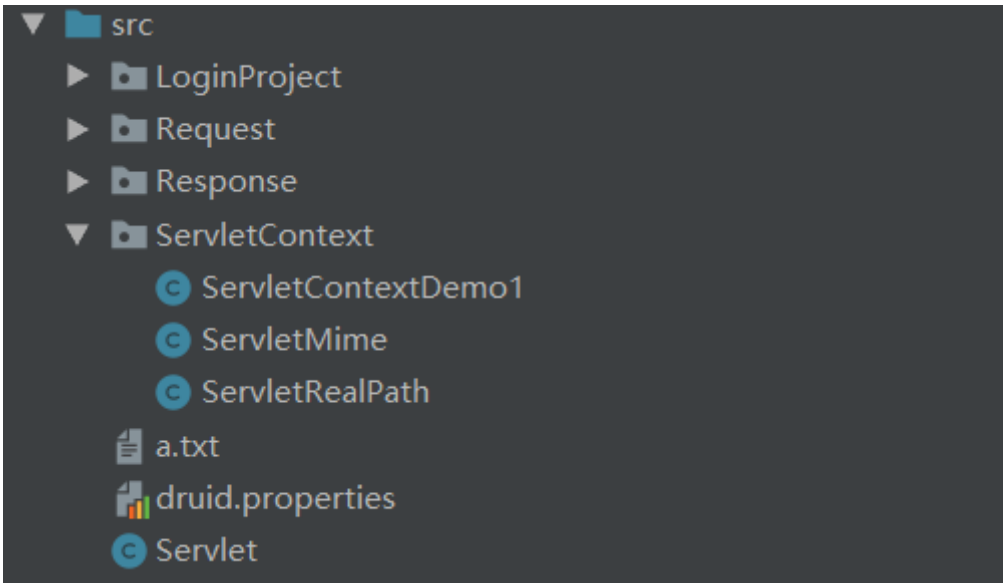
```
项目存储真实地址.
```



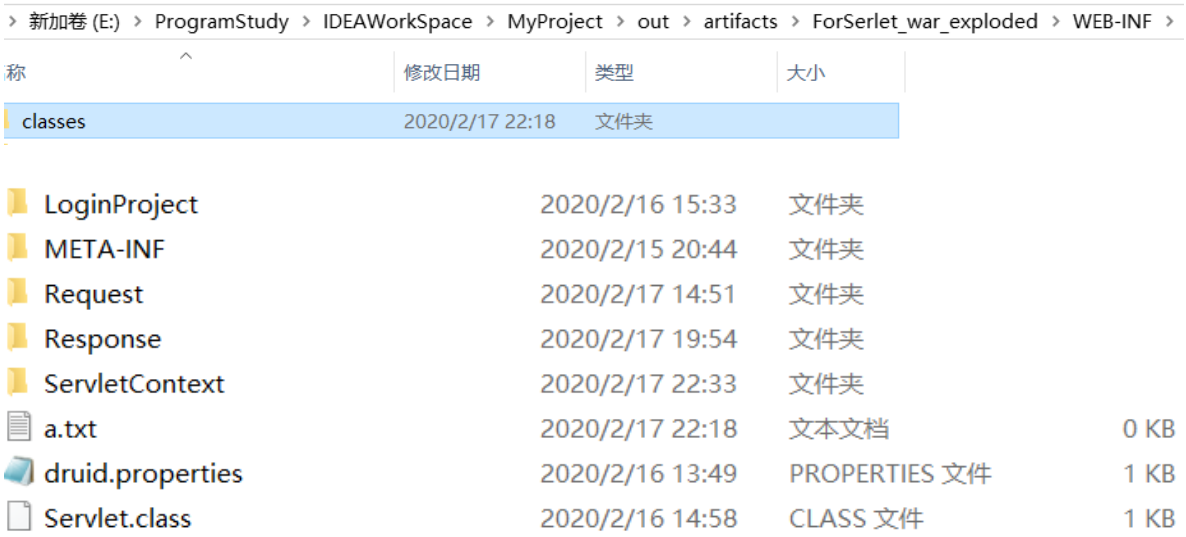
II、获取不同位置的资源

①获取WEB-INF下的a.txt真实路径

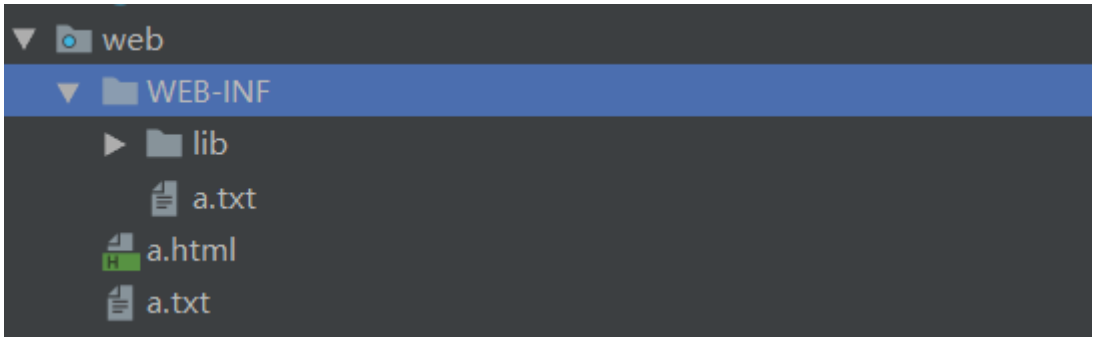
②获取src下的a.txt真实路径



src目录下的东西最后会存在WEB-INF的classes目录下



③获取Web目录下的a.txt



III、代码

因为/代表整个WEB目录，而WEB-INF在WEB目录下，而src下的目录在WEB-INF的classes目录里.

```
ServletContext context = this.getServletContext();
//获取WEB目录下的a.txt
String s = context.getRealPath( s: "/a.txt");
//获取WEB-INF目录下的a.txt
String s1 = context.getRealPath( s: "/WEB-INF/a.txt");
//获取src目录下的a.txt
String s2 = context.getRealPath( s: "/WEB/INF/classes/a.txt");
System.out.println(s);
System.out.println(s1);
System.out.println(s2);
```

```
E:\ProgramStudy\IDEAWorkSpace\MyProject\out\artifacts\ForSerlet_war_exploded\a.txt
E:\ProgramStudy\IDEAWorkSpace\MyProject\out\artifacts\ForSerlet_war_exploded\WEB-INF\a.txt
E:\ProgramStudy\IDEAWorkSpace\MyProject\out\artifacts\ForSerlet_war_exploded\WEB\INF\classes\a.txt
```

十一、综合案例

1、验证码

（1）本质：图片

（2）目的：防止恶意表单注册

（3）步骤：

①定义一个页面，用img标签指定请求Servlet地址.

②定义一个Servlet，用于响应img图片

I、定义一个图片对象

II、获取一个画笔对象，画图片

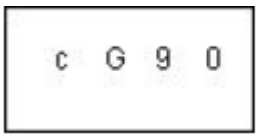
III、使用ImageIO.write方法响应图片

（4）代码：

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

```
    <script>
        window.onload=function () {
            var img = document.getElementById("img1");
            img.onclick=function () {
                //加一个时间，防止访问同样的链接时，浏览器对其进行缓存
                var date=new Date().getTime();
                img.src="/Wu/YzmServlet?"+date;
            }

        }
    </script>
</head>
<body>
    <img src="/Wu/YzmServlet" id="img1">
</body>
</html>
```

```
@WebServlet("/YzmServlet")
public class YzmServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
      int width=100;
      int high=50;
      //定义一个BufferedImage对象，设置图片大小，以及色域 图片会在内存中生成.
      BufferedImage bi=new BufferedImage(width,high,BufferedImage.TYPE_INT_RGB);
        //获取画笔对象
        Graphics graphics = bi.getGraphics();
        //设置画笔颜色
            graphics.setColor(Color.WHITE);
            //填充矩形
            graphics.fillRect(0,0,width,high);
            //设置画笔颜色
            graphics.setColor(Color.BLACK);
            //画矩形边框，因为画的边框占一个像素，会超出，所以要-1；
            graphics.drawRect(0,0,width-1,high-1);
            //获取随机数
            String str="1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
            for(int i=1;i<=4;i++){
                int index= (int)(Math.random()*str.length());
                graphics.drawString(str.charAt(index)+"",width/5*i,high/2);
            }
            //输出.
        ImageIO.write(bi,"jpg",response.getOutputStream());


    }

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

```

结果 

2、文件下载

（1）文件下载需求

①点击超链接

②点击超链接显示下载提示框

③下载

（2）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。所以不满足需求，排除。

② content-disposition:attachment;filename=xxx:打开就能弹出下载提示框，并下载，而谷歌浏览器不会弹出提示框，满足需求。

```
    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.设置filename值的编码方式，可以解决中文乱码
    4. 指定response的响应头： content-disposition:attachment;filename=xxx
    5. 将数据写出到response输出流
```

（3）问题

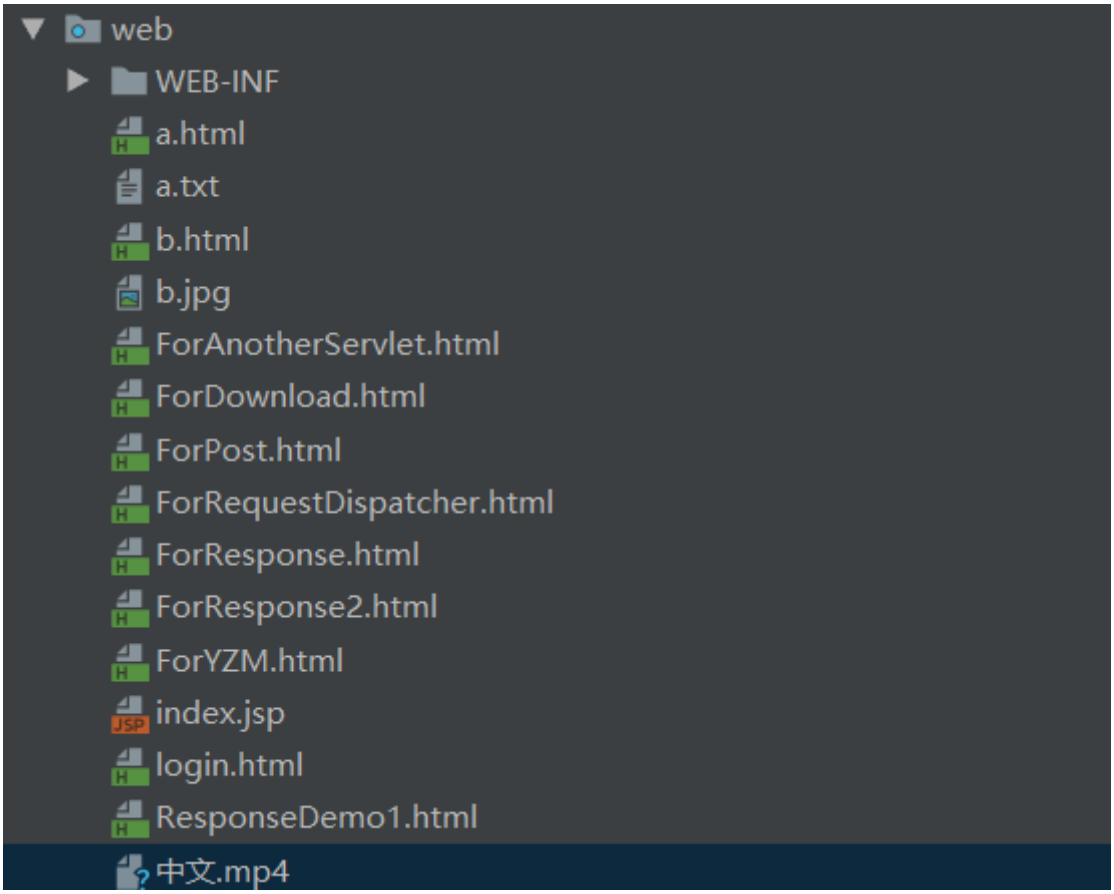
I、 中文文件名乱码问题，原因：头使用的编码不一样而到导致乱码.

II、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编码方式.

III、目录结构



IIII、代码实现

```
<a href="/Wu/DownloadServlet?Filename=中文.mp4">视频</a>
<a href="/Wu/DownloadServlet?Filename=b.jpg">图片</a>
```

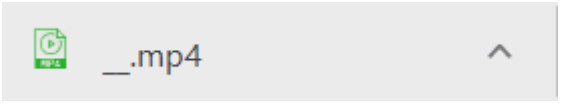
下图是工具类，不需要记，也不需要写

```
public class DownLoadUtils {

    public static String getFileName(String agent, String filename) throws UnsupportedEncodingException {
        if (agent.contains("MSIE")) {
            // IE浏览器
            filename = URLEncoder.encode(filename, enc: "utf-8");
            filename = filename.replace( target: "+", replacement: " ");
        } else if (agent.contains("Firefox")) {
            // 火狐浏览器
            BASE64Encoder base64Encoder = new BASE64Encoder();
            filename = "=?utf-8?B?" + base64Encoder.encode(filename.getBytes( charsetName: "utf-8")) + "?=";
        } else {
            // 其它浏览器
            filename = URLEncoder.encode(filename, enc: "utf-8");
        }
        return filename;
    }
}
```

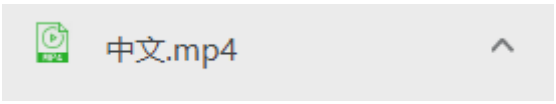
```
@WebServlet("/DownloadServlet")
public class DownloadServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //获取参数
         String filename = request.getParameter("Filename");
         //获取ServletContext对象
```

```
        ServletContext sc = request.getServletContext();
        //获取真实路径
        String realPath = sc.getRealPath("/"+filename);
        //使用字节输入流
        FileInputStream fis=new FileInputStream(realPath);
        //获取字符输出流
        ServletOutputStream os = response.getOutputStream();
        //建议设置响应头Content-Type的mime属性，目的是为了告诉浏览器是什么类型文件.XXX;XXX可以只设置其中一个
        String mimeType = sc.getMimeType(realPath);
        response.setHeader("Content-type",mimeType);

        //设置filename的编码方式，需要传入浏览器版本信息和filename，目的是要和响应的头编码一致,解决乱码问题.没加下面这两行代码就会中文乱码.
        String agent = request.getHeader("user-agent");
        filename = DownLoadUtils.getFileName(agent, filename);
        //设置Content-disposition响应头，用于告诉浏览器附件下载
        response.setHeader("Content-disposition","attachement;Filename="+filename);
        //响应的浏览器
        byte[] b=new byte[1024];
        int lenth=0;
        while((lenth=fis.read(b))!=-1){
            os.write(b,0,lenth);
        }
        //关闭流
        fis.close();
    }
```

未设置filename编码前



设置filename编码后



下载的文件，因为谷歌浏览器没有弹出提示框，而其他浏览器有.