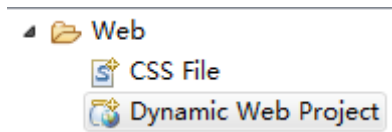
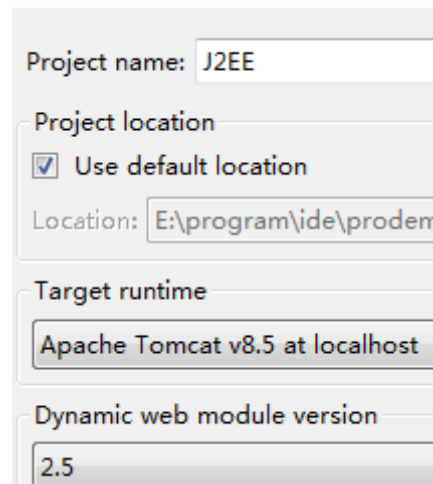


配置tomcat到eclipse中

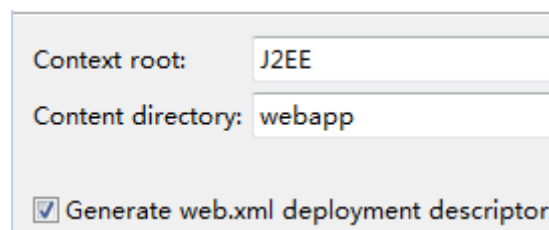
a)



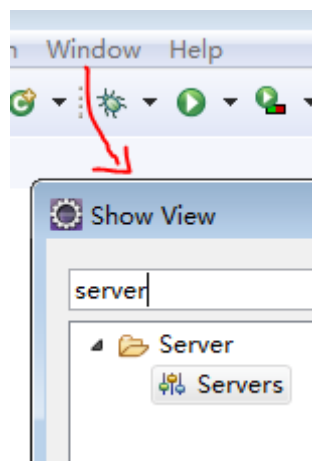
b)



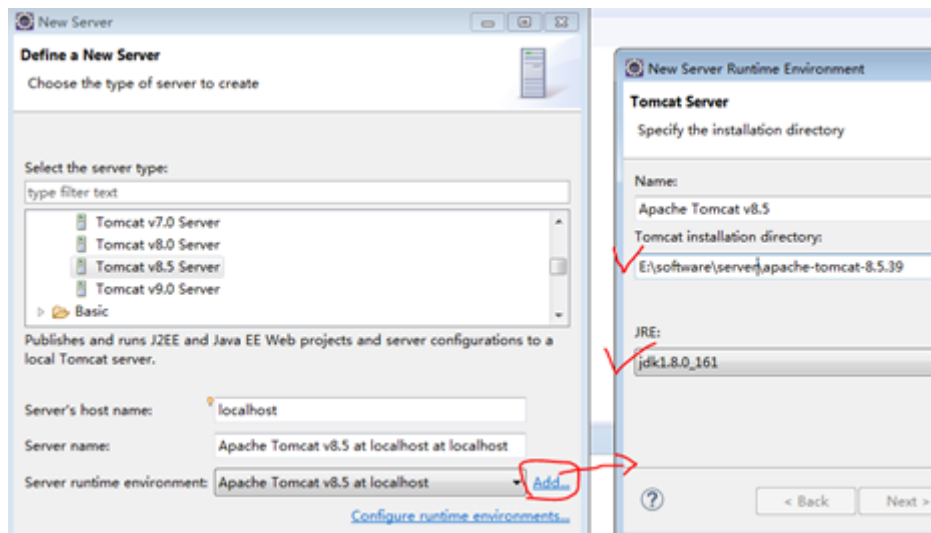
c)



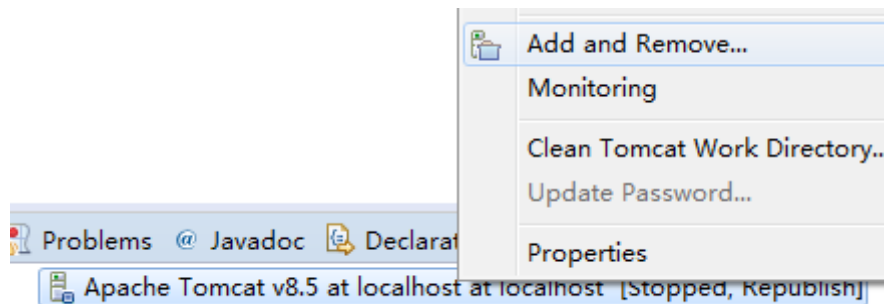
d)



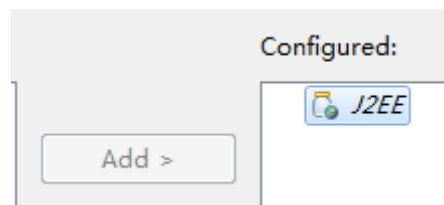
e)



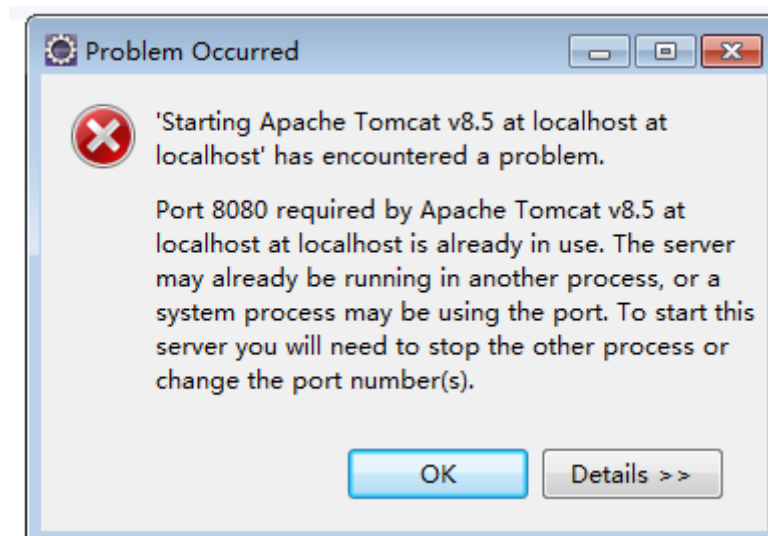
f)



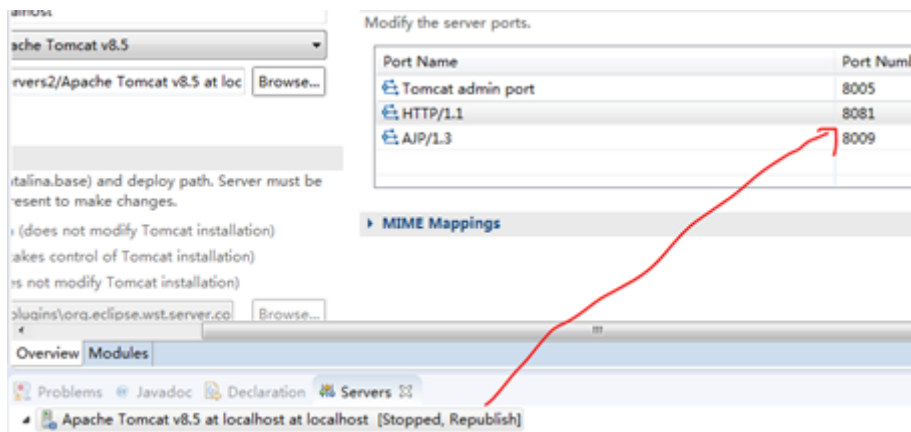
g)



h)



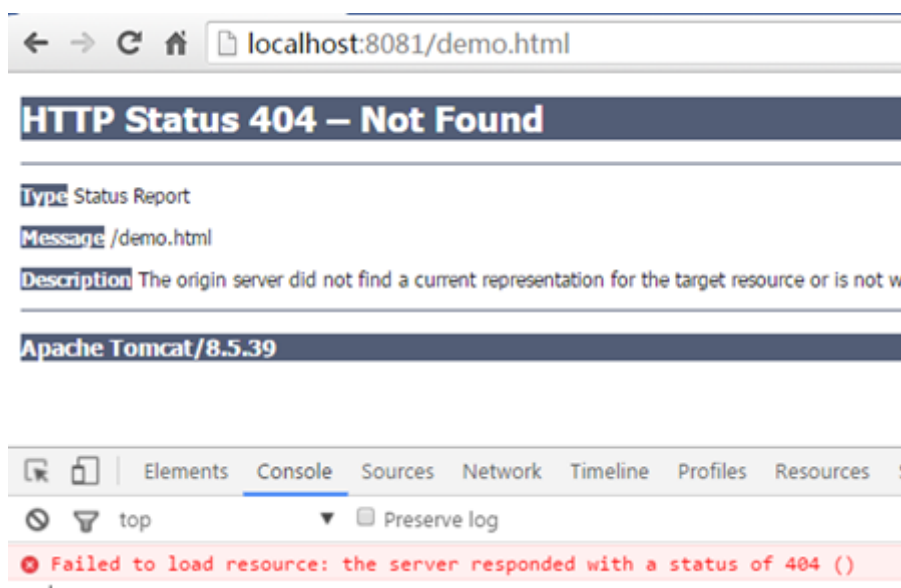
i) 双击并改动端口号



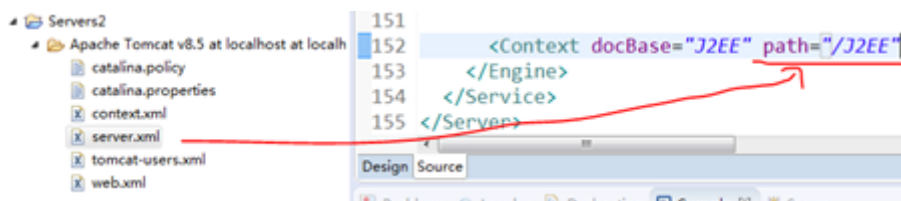
j) 启动成功

1/11/20, 20:12:02 11.018.018.018
信息: Server startup in 5429 ms

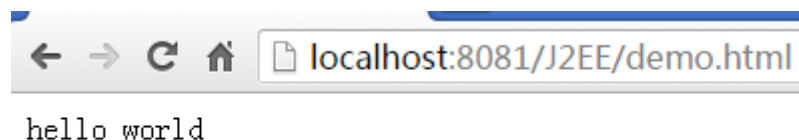
k) 运行页面时f12打开调试窗口



l)



m)




页面的响应事件需要和后台的java代码关联起来

n) Servlet是sun用来开发动态web资源的技术，在开发的api中提供了servlet接口，约定俗成把实现了servlet接口的java类称之为servlet，用户想要开发java代码去响应页面动作需要完成两个步骤

- 编写一个java类实现一个servlet接口
- 把开发好的java类部署到web服务器中

o) 缓缓开发过程

i.  src

ii. Name:

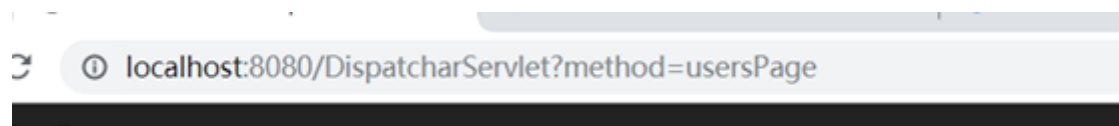
iii.

```
package com.j2ee.servlet;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LoginServlet extends HttpServlet{
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) {
        String username=req.getParameter("username");
        String password=req.getParameter("password");
        System.out.println(username+".."+password);
    }
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) {
        doGet(req,resp);
    }
}
```

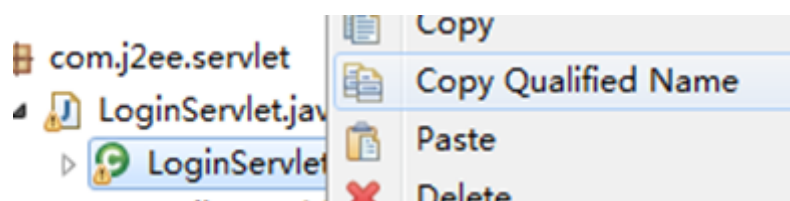
iv.

```
<html>
<body>
    <form action="/J2EE/LoginServlet" method="post">
        <input type="text" name="username"/><br/>
        <input type="password" name="password"/><br/>
        <input type="submit" value="登录"/>
    </form>
</body>
</html>
```

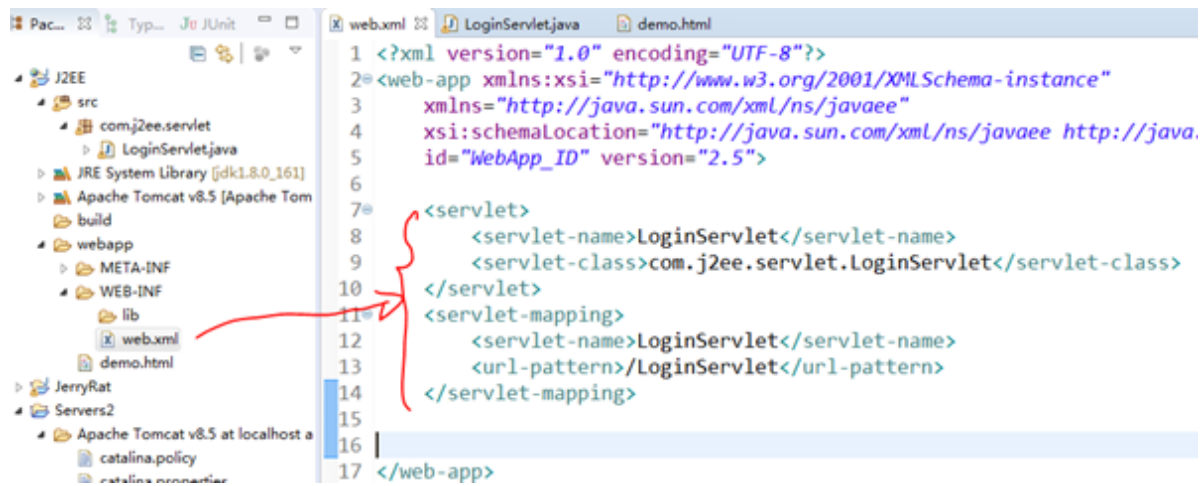
判定是浏览器请求服务器的资源时，第一个url的斜杠，代表服务器，斜杠后的第一个单词是服务器下挂的application，第二个斜杠后才是application下面具体的业务java代码（servlet）



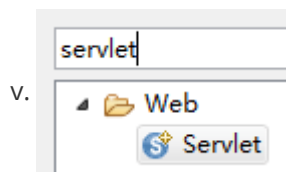
拷贝servlet的全路径名



按照——映射关系，对xml文件进行配置，将刚才拷贝的路径搞进去



p) 速速开发过程

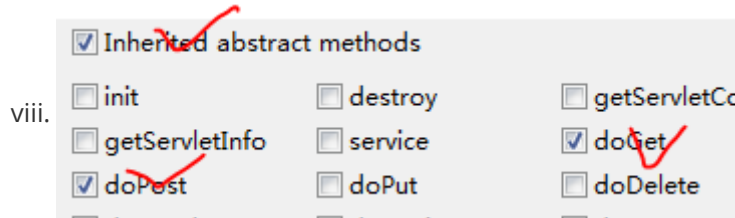
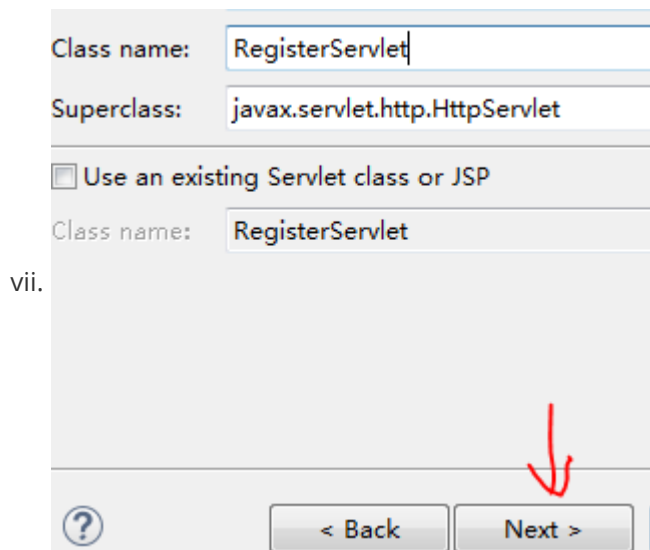


vi.

```

<form action="/J2EE/RegisterServlet" method="post">
  <input type="text" name="username"/><br/>
  <input type="password" name="password"/><br/>
  <input type="submit" value="注册"/>
</form>

```



ix. 写业务代码

```

public class RegisterServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) {
        String username=request.getParameter("username");
        String password=request.getParameter("password");
        System.out.println(username+"."+password);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) {
        doGet(request, response);
    }
}

```

x. 检查是否配置完整

```

<servlet>
    <description></description>
    <display-name>RegisterServlet</display-name>
    <servlet-name>RegisterServlet</servlet-name>
    <servlet-class>com.j2ee.servlet.RegisterServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>RegisterServlet</servlet-name>
    <url-pattern>/RegisterServlet</url-pattern>
</servlet-mapping>

```

资源路径详解（URL统一资源定位符）

1、url 进入服务器后，先进web.xml进行匹配，没有匹配时进入webapp 目录进行页面名称匹配，如果都没有，返回 404。

2、WEB-INF目录下，无法通过浏览器直接输入资源定位访问，这里是绝对安全的，只能是服务器内部访问该WEB-INF目录下的资源。

3、浏览器访问服务器资源

- i. / 代表服务器
- ii. 任何访问路径建议在最头部都加上 /
- iii. 多个不同映射可以访问同一个资源

```

<servlet>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>com.j2ee.servlet.LoginServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/RegisterServlet</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>

```

- iv. *通配符

*.扩展名 *.do *.action ...

/ 开头 以 / 结尾 //login/* ...

v. 案例练习

1. 对于如下的一些映射关系：

Servlet1 映射到 /abc/*

Servlet2 映射到 /*

Servlet3 映射到 /abc

Servlet4 映射到 .do

问题：

当请求URL为“/abc/a.html”，“/abc/”和“/”都匹配，哪个servlet响应
Servlet引擎将调用Servlet1。

当请求URL为“/abc”时，“/abc/”和“/abc”都匹配，哪个servlet响应
Servlet引擎将调用Servlet3。

当请求URL为“/abc/a.do”时，“/abc/”和“.do”都匹配，哪个servlet响应
Servlet引擎将调用Servlet1。

当请求URL为“/a.do”时，“/”和“.do”都匹配，哪个servlet响应
Servlet引擎将调用Servlet2。

当请求URL为“/xxx/yyy/a.do”时，“/”和“.do”都匹配，哪个servlet响应
Servlet引擎将调用Servlet2

2. 原则，谁像选谁

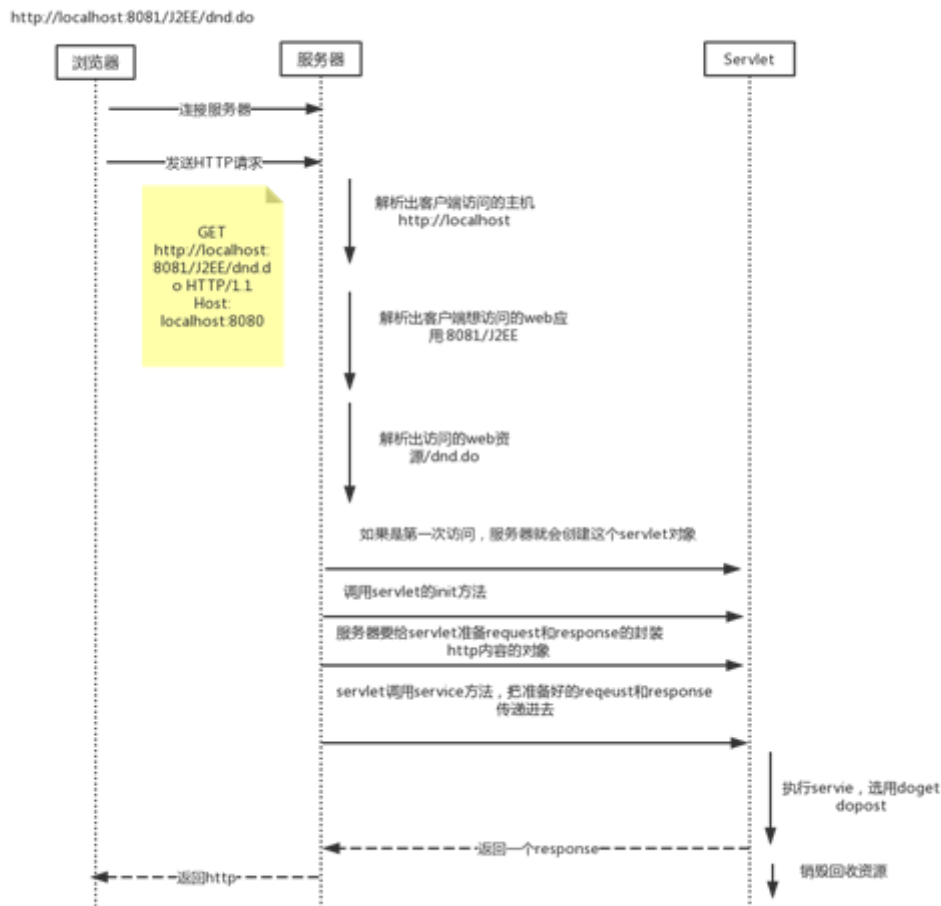
4、 服务器内部访问服务器资源

i. / 代表项目名称

Servlet的生命周期（单线程）

线程问题

servlet只会被初始化一次，没有特别处理的情况下，不应该在成员变量处放置可以被改变值的成员变量，线程处理方式有两种，加锁，交给框架



描述流程:

web服务器接收到客户端的servlet访问请求

- Web容器首先检查是否已经装载了该servlet的实例对象, 如果没有执行ii, 如果有执行iv
- 实例化一个servlet并加载到服务器中的集合中存储
- 调用servlet的init方法
- 服务器创建一个用于封装http请求消息的HttpServletRequest对象和一个代表http响应消息的 HttpServletResponse对象, 接着调用servlet的service方法, 并把request和response作为参数传递进去
- Web应用程序在被卸载或者停止之前, 会卸载servlet引擎, 并且在卸载引擎之前会调用servlet的destory () 方法

运用forward方法只能重定向到同一个Web应用程序中的一个资源。而sendRedirect方法可以让你重定向到任何URL。

Servletconfig

在 servlet 的配置文件 web.xml 中, 可以使用多个 <init-param> 标签, 作为 servlet 初始化的参数, 是写在 <servlet> 标签里面

- 1、配置web.xml中的init-param.


```

<servlet>
  <servlet-name>LoginServlet</servlet-name>
  <servlet-class>com.j2ee.servlet.LoginServlet</servlet-class>
  <!-- config -->
  <init-param>
    <param-name>name</param-name>
    <param-value>tom</param-value>
  </init-param>
  <init-param>
    <param-name>password</param-name>
    <param-value>123456</param-value>
  </init-param>
</servlet>

```

2、完成servlet的config初始化.

```

//定义一个config
private ServletConfig sc;

@Override
public void init(ServletConfig config) throws ServletException {
  this.sc=config;
  //遍历init的所有内容
  Enumeration<String> e=this.sc.getInitParameterNames();
  while(e.hasMoreElements()) {
    String name=e.nextElement();
    String value=this.sc.getInitParameter(name);
    System.out.println(name+"-->"+value);
  }
}

```

3、完成业务开发.

```

protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
  String url=req.getRequestURL().toString();
  System.out.println(url);
  String username=req.getParameter("username");
  String password=req.getParameter("password");

  if(username==null||username.equals("")) {
    //从config中取出默认值
    username=this.sc.getInitParameter("name");
    password=this.sc.getInitParameter("password");
  }
}

```

Servletcontext

1、Servletcontext代表当前应用，web容器在启动的时候，会给每个应用都创建一个servletcontext对象，context的引用被config持有（维护），可以通过config.getContext的方式获取context对象，由于所有servlet都使用同一个context，通常用来做通讯使用，可称为context域

```

Demo1Servlet.java  web.xml
1  e com.j2ee.servlet;
2
3  java.io.IOException;
4
5  javax.servlet.ServletContext;
6  javax.servlet.ServletException;
7  javax.servlet.http.HttpServlet;
8  javax.servlet.http.HttpServletRequest;
9  javax.servlet.http.HttpServletResponse;
10 class Demo1Servlet extends HttpServlet {
11
12 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
13     //准备一串数据做传输
14     String temp="hello world";
15     ServletContext context=this.getServletContext().getServletContext();
16     context.setAttribute("message",temp);
17 }

```

2、context 做 servlet 的通讯域

```

Demo2Servlet.java
1  : com.j2ee.servlet;
2
3  java.io.IOException;
4
5  javax.servlet.ServletContext;
6  javax.servlet.ServletException;
7  javax.servlet.http.HttpServlet;
8  javax.servlet.http.HttpServletRequest;
9  javax.servlet.http.HttpServletResponse;
10 class Demo2Servlet extends HttpServlet {
11
12 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
13     //准备一串数据做传输
14     ServletContext context=this.getServletContext().getServletContext();
15     String temp=(String) context.getAttribute("message");
16     System.out.println("demo1-->" + temp);
17 }

```

3、 localhost:8081/J2EE/Demo1Servlet

4、 localhost:8081/J2EE/Demo2Servlet

5、如何在配置中给整个web应用做配置呢？ 从配置文件中读取context配置

```

import java.io.IOException;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class Demo3Servlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        ServletContext sc=this.getServletContext();
        String url=sc.getInitParameter("url");
        //打印到浏览器上
        response.getWriter().print("<a href=' " + url + "' >点我</a>");
    }
}

2<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
3<context-param>
4<param-name>url</param-name>
5<param-value>https://www.baidu.com</param-value>
6</context-param>
7<servlet>
8<servlet-name>LoginServlet</servlet-name>
9<servlet-class>com.j2ee.servlet.LoginServlet</servlet-class>
10<init-param>
11<param-name>name</param-name>
12<param-value>tom</param-value>
13</init-param>
14<init-param>
15<param-name>password</param-name>
16<param-value>123456</param-value>
17</init-param>

```

6、Context如何实现两个servlet的跳转功能？ RequestDispatcher

```

ServletContext sc=this.getServletContext();
RequestDispatcher rd=sc.getRequestDispatcher("/Demo5Servlet");//得到http请求中的分发路径
RequestDispatcher rd=sc.getRequestDispatcher("/demo.html");
RequestDispatcher rd=sc.getRequestDispatcher("/WEB-INF/demo.html");
rd.forward(request, response);//forward转发

```



7、读取文本内容

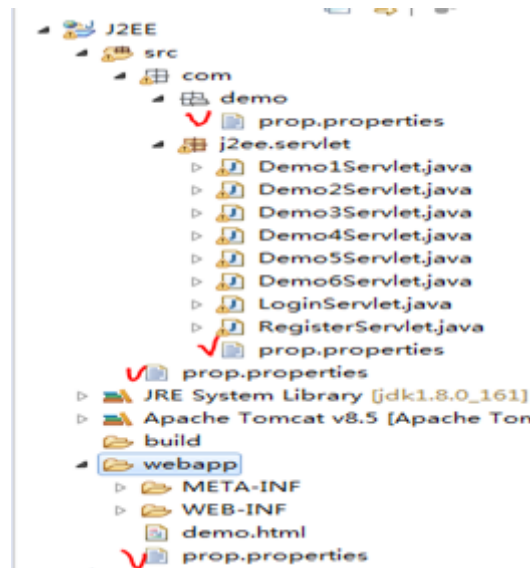
```
1      protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
2          //      method1();
3          //      method2();
4          //      method3();
5          method4();
6      }
7      //类装载器读取资源文件
8      //TODO   jvm高级
9      //webapp
10     private void method4()throws IOException{
11         //servlet.prop.properties
12         Properties prop=new Properties();
13         //获取文件输入流
14         InputStream
in=this.getServletContext().getResourceAsStream("/prop.properties");
15         prop.load(in);
16         String name=prop.getProperty("name");
17         System.out.println(name);
18     }
19     //src
20     private void method3()throws IOException{
21         //servlet.prop.properties
22         Properties prop=new Properties();
23         //获取文件输入流
24         InputStream in=this.getServletContext().getResourceAsStream("/WEB-
INF/classes/prop.properties");
25         prop.load(in);
26         String name=prop.getProperty("name");
27         System.out.println(name);
28     }
29     //com.demo
30     private void method2()throws IOException{
31         //servlet.prop.properties
32         Properties prop=new Properties();
33         //获取文件输入流
34         InputStream in=this.getServletContext().getResourceAsStream("/WEB-
INF/classes/com/demo/prop.properties");
35         prop.load(in);
36         String name=prop.getProperty("name");
37         System.out.println(name);
38     }
39     //j2ee.servlet
40     private void method1() throws IOException{
41         //servlet.prop.properties
42         Properties prop=new Properties();
43         //获取文件输入流
44         InputStream in=this.getServletContext().getResourceAsStream("/WEB-
INF/classes/com/j2ee/servlet/prop.properties");
```

```

45     prop.load(in);
46     String name=prop.getProperty("name");
47     System.out.println(name);
48 }
49

```

8、类似这样的代码

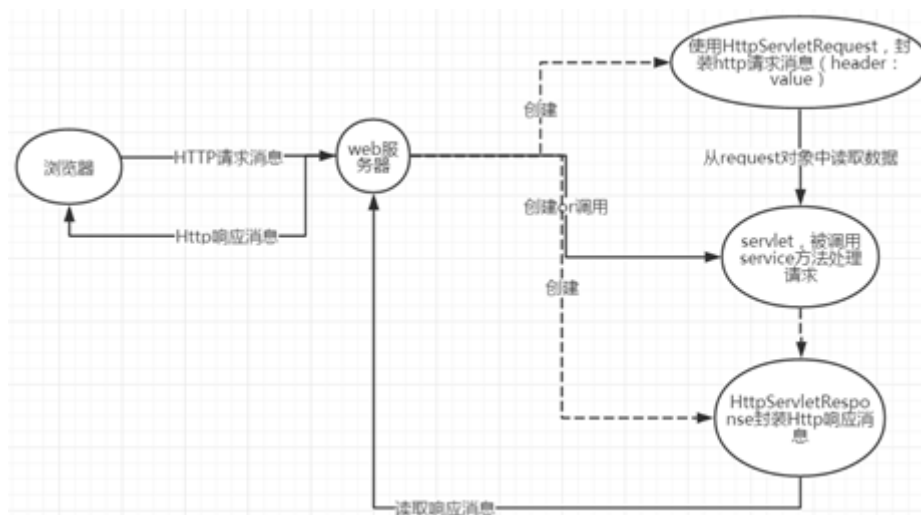


```

//获取文件输入流
InputStream in=this.getServletContext().getResourceAsStream("/prop.properties");
prop.load(in);
String name=prop.getProperty("name");
System.out.println(name);

```

Request与response



分析项目案例核心：

- i. Request: url的解析, url跳转, url的传参, 表单传参
- ii. Response: 服务器命令浏览器基于某些规范, 做规定资源内容 (命令浏览器做事-显示网页, 播放mp3, 显示图片, 下载文件, 定时器自动跳转页面...)
- iii. 涉及到的板块关联越多, 规范越多, 该板块就越重要, 建议前期花时间解决—举例: response

Response

getMethod(); 获得请求方式

getRequestURL(); 返回客户端发出请求时的完整URL。

getRequestURI(); 返回请求行中的资源名部分。

getContextPath(); 当前应用的虚拟目录

getQueryString(); 返回请求行中的参数部分。

Response案例

- iv. 字节字符流输出数据
 - v. 中文乱码处理
 - vi. 定时自动刷新
 - vii. 下载图片
 - viii. 下载文件中文乱码处理
 - ix. 随机图片
 - x. 验证码
 - xi. 重定向
 - xii. 防盗链
 - xiii. 压缩处理
- r) Request案例

Request

RequestDemo01

```
1 package com.j2ee.servlet.request;
2
3 import java.io.IOException;
4 import java.io.PrintWriter;
5
6 import javax.servlet.ServletException;
7 import javax.servlet.http.HttpServlet;
8 import javax.servlet.http.HttpServletRequest;
9 import javax.servlet.http.HttpServletResponse;
10
11 /**
12  * 通过request对象获取客户端请求信息
13  *  getRequestURL方法返回客户端发出请求时的完整URL。
14  *  getRequestURI方法返回请求行中的资源名部分。
15  *  getQueryString 方法返回请求行中的参数部分。
16  *  getPathInfo方法返回请求URL中的额外路径信息。额外路径信息是请求URL中的位于Servlet的路径之后和查询参数之前的内容，它以“/”开头。
17  *  getRemoteAddr方法返回发出请求的客户机的IP地址。
18  *  getRemoteHost方法返回发出请求的客户机的完整主机名。
19  *  getRemotePort方法返回客户机所使用的网络端口号。
```

```

20     getLocalAddr方法返回WEB服务器的IP地址。
21     getLocalName方法返回WEB服务器的主机名。
22     */
23     public class RequestDemo01 extends HttpServlet {
24
25         public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
26             /**
27              * 1. 获得客户机信息
28              */
29             String requestUrl = request.getRequestURL().toString();//得到请求的资
源
30             String requestUri = request.getRequestURI().toString();//得到请求的url
地址
31             String queryStrig = request.getQueryString();//得到请求的URL地址中附带的
参数
32             String remoteAdd = request.getRemoteAddr();//得到访问记得ip地址
33             String remoteHost = request.getRemoteHost();//得到
34             int remotePort = request.getRemotePort();
35             String remoteUser = request.getRemoteUser();//
36             String method = request.getMethod();//得到请求url地址时使用的请求方法
37             String pathInfo = request.getPathInfo();
38             String localaddr =request.getLocalAddr();//获取web服务器的ip地址
39             String localName = request.getLocalName();//q获取web服务器的主机名
40
41
42             //乱码问题
43             response.setCharacterEncoding("UTF-8");
44             response.setHeader("content-type", "text/html;charset=UTF-8");
45             //显示到浏览器
46             PrintWriter out = response.getWriter();
47             out.write("获取到的客户端的信息如下: ");
48             out.write("<hr/>");
49             out.write("请求的资源: "+requestUrl);
50             out.write("<br/>");
51             out.write("请求的Url地址: "+requestUri);
52             out.write("<br/>");
53             out.write("请求的URL地址中附带的参数: "+queryStrig);
54             out.write("<br/>");
55             out.write("来访问者的ip地址: "+remoteAdd);
56             out.write("<br/>");
57             out.write("来访问者的主机名: "+remoteHost);
58             out.write("<br/>");
59             out.write("使用的端口号: "+remotePort);
60             out.write("<br/>");
61             out.write("remoteUser: "+remoteUser);
62             out.write("<br/>");
63             out.write("请求者使用的请求方式: "+method);
64             out.write("<br/>");
65             out.write("pathInfo: "+pathInfo);
66             out.write("<br/>");
67             out.write("获取web服务器的ip地址: "+localaddr);
68             out.write("<br/>");
69             out.write("获取web服务器的名称: "+localName);
70
71         }
72

```

```

73     public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
74         doGet(request, response);
75     }
76
77 }
78

```

RequestDemo02

```

1  package com.j2ee.servlet.request;
2
3  import java.io.IOException;
4  import java.io.PrintWriter;
5  import java.util.Enumeration;
6  import javax.servlet.ServletException;
7  import javax.servlet.http.HttpServlet;
8  import javax.servlet.http.HttpServletRequest;
9  import javax.servlet.http.HttpServletResponse;
10 /**
11  * 获取客户端请求头信息
12  * 客户端请求头:
13  *   getHeader(string name)方法:String
14  *   getHeaders(String name)方法:Enumeration
15  *   getHeaderNames()方法
16  */
17 public class RequestDemo02 extends HttpServlet {
18
19     public void doGet(HttpServletRequest request, HttpServletResponse
response)
20         throws ServletException, IOException {
21         //设置字符以UTF-8的形式编写到客户端
22         response.setCharacterEncoding("UTF-8");
23         //要求浏览器怎么做
24         response.setHeader("content-type", "text/html;charset=UTF-8");
25         PrintWriter out = response.getWriter();
26
27         //获取所有的请求头
28         Enumeration<String> reqHeadInfos = request.getHeaderNames();
29         out.write("获取到客户端的所有请求头信息如下: ");
30         out.write("<hr/>");
31         while(reqHeadInfos.hasMoreElements()) {
32             String headName= reqHeadInfos.nextElement();
33             //根据请求头的名字获取对应的请求头的值
34             String headValue = request.getHeader(headName);
35             out.write(headName+": "+headValue);
36             out.write("<br/>");//段标签
37         }
38         out.write("<br/>");//换行
39         out.write("获取到客户端的Accept-Encoding请求头的值: ");
40         out.write("<hr/>");
41         //获取Accept-Encoding请求头对应的值
42         String value = request.getHeader("Accept-Encoding");
43         out.write(value);
44
45         //获取指定的值(与上一步结果相同)
46         Enumeration<String> e = request.getHeaders("Accept-Encoding");

```



```

47         while(e.hasMoreElements()) {
48             String string = e.nextElement();
49             System.out.println(string);
50         }
51     }
52
53     public void doPost(HttpServletRequest request, HttpServletResponse
response)
54         throws ServletException, IOException {
55         doGet(request, response);
56     }
57
58 }
59
60

```

RequestDemo03

```

1  package com.j2ee.servlet.request;
2
3  import java.io.IOException;
4  import java.io.UnsupportedEncodingException;
5  import java.text.MessageFormat;
6  import java.util.Enumeration;
7  import java.util.Map;
8
9  import javax.servlet.ServletException;
10 import javax.servlet.http.HttpServlet;
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13
14 /**
15  * 获取客户端通过Form表单提交上来的参数
16  */
17 public class RequestDemo03 extends HttpServlet {
18     @Override
19     protected void doGet(HttpServletRequest request, HttpServletResponse
response)
20         throws ServletException, IOException {
21         metchod01(request, response);
22     }
23     /**
24      * getParameter(String)方法(常用)
25      * getParameterValues(String name)方法(常用)
26      * @param request
27      * @param response
28      * @throws IOException
29      */
30     private void metchod01(HttpServletRequest request, HttpServletResponse
response) throws IOException {
31         //设置客户端以utf-8编码提交数据,
32         request.setCharacterEncoding("UTF-8");
33         //获取表单的编号
34         String userid = request.getParameter("userid");
35         //获取用户名
36         String userName = request.getParameter("username");
37         //获取密码

```



```

38     String usesrpass = request.getParameter("userpass");
39     //获取性别
40     String sex = request.getParameter("sex");
41     //获取部门
42     String dept = request.getParameter("dept");
43     //获取兴趣爱好（多选）
44     String[] insts = request.getParameterValues("inst");
45     //获取填写的说明信息
46     String note = request.getParameter("note");
47     //获取隐藏域的内容
48     String hiddenField = request.getParameter("hiddenField");
49     String str = "";
50     /*
51      * 获取数组数据的技巧，可以避免insts数组为null时引发的空指针异常错误！
52      */
53     for(int i=0;insts!=null&&i<insts.length;i++) {
54         if(i==insts.length-1) {
55             str +=insts[i];
56         }else {
57             str+=insts[i];
58         }
59     }
60     //
61     String htmlStr = "<table>"+
62         "<tr><td>填写的编号: </td><td>{0}</td></tr>"+
63         "<tr><td>填写的用户名: </td><td>{1}</td></tr>"+
64         "<tr><td>填写的密码: </td><td>{2}</td></tr>"+
65         "<tr><td>填写的性别: </td><td>{3}</td></tr>"+
66         "<tr><td>填写的部门: </td><td>{4}</td></tr>"+
67         "<tr><td>填写的兴趣: </td><td>{5}</td></tr>"+
68         "<tr><td>填写的说明: </td><td>{6}</td></tr>"+
69         "<tr><td>隐藏域的内容: </td><td>{7}</td></tr>"+
70         "</table>";
71     htmlStr=MessageFormat.format(htmlStr, userid,//
72         userName,usesrpass,sex,dept,insts,note,hiddenField);
73     response.setCharacterEncoding("UTF-8");
74     response.setContentType("text/html;charset=UTF-8");
75     response.getWriter().write(htmlStr);
76 }
77 /**
78  * 在服务器端使用getParameterNames获取接收表单参数
79  * @param request
80  * @param response
81  * @throws IOException
82  */
83 private void metchod02(HttpServletRequest request, HttpServletResponse
response) throws IOException {
84     request.setCharacterEncoding("UTF-8");
85     response.setCharacterEncoding("UTF-8");
86     response.setHeader("content-type", "text/html;charset=utf-8");
87     Enumeration<String> paramNames = request.getParameterNames();
88     while(paramNames.hasMoreElements()) {
89         String name = paramNames.nextElement();
90         String value = request.getParameter(name);
91         // System.out.println(name+">"+value);
92         // System.out.println(MessageFormat.format("{0}={1}",
name,value));

```

```

93         response.getWriter().write(MessageFormat.format("{0}={1}"
<br/>", name,value));
94     }
95
96 }
97
98 /**
99  * 使用map获取表单值
100  * @param request
101  * @param response
102  * @throws IOException
103  */
104 private void metchod03(HttpServletRequest request, HttpServletResponse
response)
105     throws IOException {
106     // 设置字符编码
107     response.setCharacterEncoding("UTF-8");
108     request.setCharacterEncoding("UTF-8");
109     response.setHeader("content-type", "text/html;charset=utf-8");
110     // 获取封装对象
111     Map<String, String[]> paramMap = request.getParameterMap();
112     for (Map.Entry<String, String[]> entry : paramMap.entrySet()) {
113         String paramName = entry.getKey();
114         String paramValue = "";
115         String[] paramValueArr = entry.getValue();
116         for (int i = 0; paramValueArr != null && i <
paramValueArr.length; i++) {
117             if(i==paramValueArr.length-1) {
118                 paramValue+=paramValueArr[i];
119             }else {
120                 paramValue+=paramValueArr[i]+" ";
121             }
122         }
123     }
124     // System.out.println(paramName+">"+paramValue);
125     // System.out.println(MessageFormat.format("{0}={1}",
paramName,paramValue));
126     response.getWriter().write(MessageFormat.format("{0}={1}"
<br/>", paramName,paramValue));
127 }
128
129 }
130
131 @Override
132 protected void doPost(HttpServletRequest request, HttpServletResponse
response)
133     throws ServletException, IOException {
134     doGet(request, response);
135 }
136
137 }
138

```

RequestDemo04

```
1 package com.j2ee.servlet.request;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8
9 public class RequestDemo04 extends CharcaterServlet {
10     //request与response已经被乱码解决方案解决好勒
11     protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
12         String name = request.getParameter("username");
13         System.out.println("RequestDemo04>>>" + name);
14         //转发 传递数据转发之前设值
15         request.setAttribute("name", name);
16         request.getRequestDispatcher("/RequestDemo05").forward(request,
response);
17     }
18
19     protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
20         doGet(request, response);
21     }
22
23 }
24
```

RequestDemo05

```
1 package com.j2ee.servlet.request;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8
9 public class RequestDemo05 extends HttpServlet {
10     protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
11         String name = (String) request.getAttribute("name");
12         System.out.println("RequestDemo05>>>" + name);
13     }
14     protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
15         doGet(request, response);
16     }
17
18 }
19
```

CharcaterServlet

```
1 package com.j2ee.servlet.request;
2
3 import java.io.IOException;
4
5 import javax.servlet.ServletException;
6 import javax.servlet.ServletRequest;
7 import javax.servlet.ServletResponse;
8 import javax.servlet.http.HttpServlet;
9 import javax.servlet.http.HttpServletRequest;
10 import javax.servlet.http.HttpServletRequestWrapper;
11 import javax.servlet.http.HttpServletResponse;
12
13 /**
14  *自定义框架真正实现完全乱码处理
15  * @author 18285
16  *
17  */
18 //自定义框架真正实现完全乱码处理
19 //存储页面html的格式 html打开页面的格式 服务器解析http数据包解析格式 servlet处
理request中的数据格式 (get, post)
20 //response中数据需要格式解析 服务器对response数据包格式解析 response规定浏览器格
式解析 往数据中写数据需要格式解析
21 //存入数据库需要格式解析 读取数据库中数据要格式解析 读出来解析到哪个端口 (客户端) 需要
格式解析 ... 代码文件有存储格式 数据库文件有存储格式
22 //数据库中有一些配置也含有数据解析的默认值
23
24 //request从浏览器带服务器, 服务器到servlet get与post两种提交方式进行数据解析
25 public class CharcaterServlet extends HttpServlet {
26     private String defalutCharset = "UTF-8";
27     //重写service
28     @Override
29     public void service(ServletRequest req, ServletResponse res) throws
ServletException, IOException {
30         // ServletRequest 转为 HttpServletRequest
31         // ServletResponse 转为 HttpServletResponse
32         HttpServletRequest request;
33         HttpServletResponse response;
34         MyCharServletEncodingRequest requestWrapper;
35         try {
36             // HttpServletRequest extends ServletRequest
37             request = (HttpServletRequest) req; // 接口向下转型
38             response = (HttpServletResponse) res; // 接口向下转型
39
40             //初始化编码格式
41             String charset =
this.getServletConfig().getInitParameter("charset");
42             if (charset == null) {
43                 charset = defalutCharset;
44             }
45             // request的post提交解码
46             request.setCharacterEncoding(charset);
47             //
48             response.setHeader("content-type", "text/html;charset=" +
charset);
49             // 简便写法
```

```

50         // response.setContentType("text/html;charset="+charset);
51
52         // request的get提交解码
53         // 使用自定义子类，包装原来的HttpServletRequest，需要复写getParameter
54         ()
55         requestWrapper = new MyCharServletEncodingRequest(request,
56         charset);
57         } catch (Exception e) {
58             throw new ServletException("non-HTTP request or response");
59         }
60         //传入父类的方法service(HttpServletRequest req, HttpServletResponse
61         resp)
62         //这个方法是对method的一个判断（get、post）
63         service(requestWrapper, response);
64     }
65
66     // 使用内部类
67     /*
68     * 1.实现与被增强对象相同的接口 2、定义一个变量记住被增强对象 3、定义一个构造器，接
69     收被增强对象 4、覆盖需要增强的方法
70     * 5、对于不想增强的方法，直接调用被增强对象（目标对象）的方法
71     */
72     class MyCharServletEncodingRequest extends HttpServletRequestWrapper {
73         private HttpServletRequest request;
74         private String targetCharset;
75
76         public MyCharServletEncodingRequest(HttpServletRequest request,
77         String targetCharset) {
78             super(request);
79             this.request = request;
80             this.targetCharset = targetCharset;
81         }
82
83         /**
84         * 重写getParameter方法 ServletRequestWrapper>>>getParameter()
85         */
86         @Override
87         public String getParameter(String name) {
88             try {
89                 // 获取参数的值
90                 String value = this.request.getParameter(name);
91                 if (null == value) {
92                     return null;
93                 }
94                 // 如果不是以get方式提交数据的，就直接返回获取到的值
95                 if (!this.request.getMethod().equalsIgnoreCase("get")) {
96                     return value;
97                 } else {
98                     // 如果是以get方式提交数据的，这就直接返回获取到值
99                     value = new String(value.getBytes(targetCharset),
100                     this.request.getCharacterEncoding());
101                     return value;
102                 }
103             } catch (Exception e1) {
104                 throw new RuntimeException(e1);
105             }
106         }
107     }

```

Cookie-response层面

<https://www.cnblogs.com/muzongyan/archive/2010/08/30/1812552.html>

<https://www.cnblogs.com/xdp-gacl/p/3803033.html>

<https://www.cnblogs.com/whgk/p/6422391.html>

s) 创建cookie打开一个浏览器就是一个cookie

t) 如何向客户端写Cookie: CookieDemo01

u) 服务器如何得到客户端传来的Cookie CookieDemo02 Cookie[] getCookies() 获取浏览器发来的信息 (name=value)

```
Cookie cookie = new Cookie("cookieName", "cookieValue");
response.addCookie(cookie);
```

设置生命周期: 以秒为单位

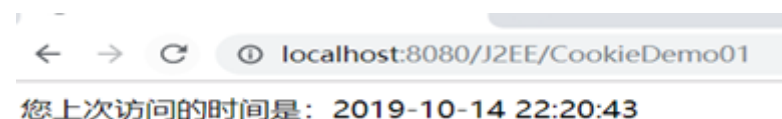
```
cookie.setMaxAge(3600);
```

expiry=-1: 代表浏览器关闭后, 也就是会话结束后, cookie就失效了, 也就没有了。

expiry>0: 代表浏览器关闭后, cookie不会失效, 仍然存在。并且会将cookie保存到硬盘中, 直到设置时间过期才会被浏览器自动删除。

expiry=0: 删除cookie。不管是之前的expiry=-1还是expiry>0, 当设置expiry=0时, cookie都会被浏览器给删除。

v) 读取cookie



使用cookie记录用户上一次访问的时间: CookieDemo01

w) 删除cookie

```
cookie.setMaxAge(0); // ==0时, 用户退出浏览器时就删除cookie
```

x) cookie中存取中文

```
Cookie cookie = new Cookie("userName", URLEncoder.encode("鱼鱼", "UTF-8"))
```

y) api

getName() 获得名称, cookie中的key

getValue() 获得值, cookie中的value

setValue(java.lang.String newValue) 设置内容, 用于修改key对应的value值。

setMaxAge(int expiry) 设置有效时间【】

setPath(java.lang.String uri) 设置路径【】

setDomain(java.lang.String pattern) 设置域名, 一般无效, 有浏览器自动设置,

z) 中文存储使用

URLEncoder:编码

URLDecoder:解码

//发送cookie

```
Cookie cookie = new Cookie(URLEncoder.encode("哈哈"),URLEncoder.encode("呵呵"));
```

```
response.addCookie(cookie);
```

//获得cookie中文内容

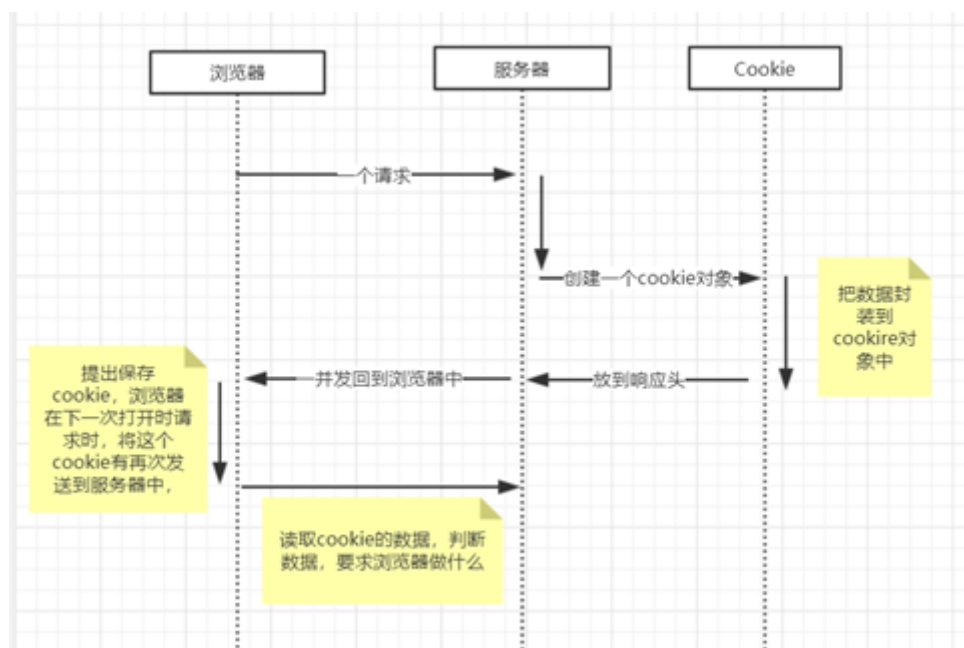
```
URLDecoder.decoder(request.getCookie().getName);    //获取key
```

```
URLDecoder.decoder(request.getCookie().getValue);    //获取value
```

aa) 使用场景

记住用户名、历史记录、登录状态信息记录 (在浏览器中)

bb) 执行流程



Session

伪码分析执行过程

1.第一次访问创建Session对象，给Session对象分配一个唯一的ID，叫JSESSIONID。

```
new HttpSession();
```

2.把JSESSIONID作为Cookie的值发送给浏览器保存

```
Cookie cookie = new Cookie("JSESSIONID", sessionId);  
response.addCookie(cookie);
```

3.第二次访问的时候，浏览器带着JSESSIONID的cookie访问服务器

4.服务器得到JSESSIONID，在服务器的内存中搜索是否存放对应编号的session对象

5.如果找到对应编号的session对象，直接返回该对象

6.如果找不到对应编号session对象，创建新的session对象，继续走1的流程

结论通过JSESSION的cookie值在服务器找session对象

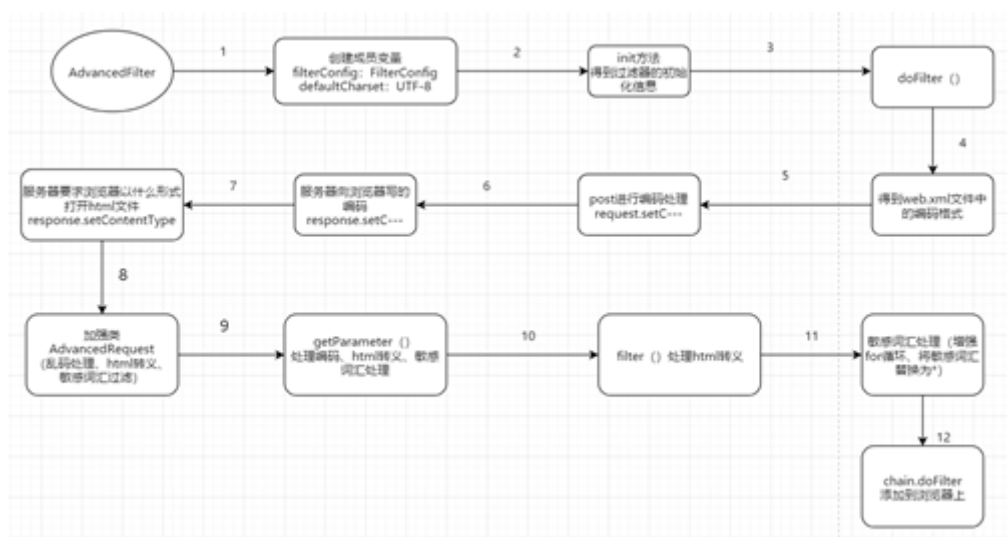
总结：

数据保存在客户端的Cookie技术

数据保存在服务端的Session技术

上传、下载、拦截

Filter过滤、拦截



对web服务器管理的所有web资源：例如jsp, Servlet,

静态图片文件或静态 html 文件等进行拦截，从而实现一些特殊的功能。

例如实现URL级别的权限访问控制、过滤敏感词汇、压缩响应信息等一些高级功能。

服务器初始化时filter就初始化了

Filter接口中有一个`doFilter`方法，当我们编写好Filter，并配置对哪个web资源进行拦截后，

WEB服务器每次在调用web资源的service方法之前，都会先调用一下filter的`doFilter`方法，

因此，在该方法内编写代码可达到如下目的：

调用目标资源之前，让一段代码执行。

是否调用目标资源（即是否让用户访问web资源）。

调用目标资源之后，让一段代码执行。

Filter开发步骤：

编写java类实现Filter接口，并实现其doFilter方法。

在 web.xml 文件中使用<filter>和<filter-mapping>元素对编写的filter类进行注册，并设置它所能拦截的资源

Jsp

动态页面，就是服务器向浏览器写html文件，里面的变量可以根据实际应用场景进行改变

Jsp 9大内置对象

request、response、session、application、out、pageContext、config、page和exception;

jsp 7大动作

include、useBean、getProperty、setProperty、param、forward、plugin

jsp跟servlet的关系：

jsp经编译后就变成了servlet，所以说jsp本质就是servlet，jvm只能识别java的类，不能识别jsp代码，web容器将jsp的代码编译成jvm能够识别的java类。