

JSP Servlet 2.0/3.0 快速入门系列课程

时间: 2017/11/08

作者: 袁毅雄

邮箱: 896778954@qq.com

目录

JSP Servlet 2.0/3.0 快速入门系列课程.....	1
目录	1
Day01	2
JavaEE.....	2
Tomcat 基础配置.....	2
Tomcat 配置要求.....	2
Tomcat 文件夹结构.....	3
九大内置对象,四大作用域.....	3
简介.....	3
九大内置对象详解.....	4
EL 表达式.....	12
四大作用域详解.....	12
案例.....	12
Day02	13
JavaEE.....	13
JSTL 表达式.....	13
分支(branch).....	13
循环(ciculation).....	13
Servlet	13
生命周期(life cycle)/原理图(schematic)	13
API	14
请求 GET/POST	14
Web2.0 解决方案: web.xml 开发模式.....	14
Web3.0 解决方案: 注解开发模式	14
Day03	16
JavaEE.....	16
Filter	16
生命周期(life cycle)	16
API	16
Web2.0 解决方案: web.xml 开发模式.....	17
Web3.0 解决方案: 注解开发模式	17
四大监听器/Listener	18
简介(intro)	18

四大监听器详解.....	18
Web2.0 解决方案: web.xml 开发模式.....	20
Web3.0 解决方案: 注解开发模式	20

Day01

JavaEE

Tomcat 基础配置

Tomcat 配置要求

JDK 安装好,java 命令有效

JDK 环境变量,javac 命令有效

JAVA_HOME

D:\java\JDK\JDK8

PATH

% JAVA_HOME%\bin

% JAVA_HOME%\lib

Tomcat 安装好

Tomcat 环境变量[可以不配置]


CATALINA_HOME


D:\java\apache-tomcat-8.0.38


Tomcat 文件夹结构


> java > apache-tomcat-8.0.38


名称 ^


 bin


 conf

 lib

 logs

 temp

 webapps

 work

九大内置对象,四大作用域

简介

Object name	Nickname	Type	Action scope
request	请求对象	javax.servlet.HttpServletRequest	Request
response	响应对象	javax.servlet.SrvletResponse	Page
pageContext	页面上下文对象	javax.servlet.jsp.PageContext	Page
session	会话对象	javax.servlet.http.HttpSession	Session
application	应用程序对象	javax.servlet.ServletContext	Application
out	输出对象	javax.servlet.jsp.JspWriter	Page
config	配置对象	javax.servlet.ServletConfig	Page
page	页面对象	javax.lang.Object	Page
exception	例外对象	javax.lang.Throwable	Page

九大内置对象详解

page

作用(effect): 确定页面内容

API

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
```

out

作用(effect): 往页面上输出东西

API

a.jsp

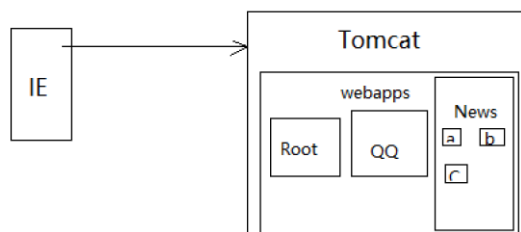
```
<%  
    out.print("<h1>Hello</h1>");  
%>
```

a_jsp.java → a_jsp.class

```
out.write("\r\n");  
out.write("<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" \"http://www.w3.org/TR/html4/loose.dtd\">\r\n");  
out.write("<html>\r\n");  
out.write("<head>\r\n");  
out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-8\">\r\n");  
out.write("<title>Insert title here</title>\r\n");  
out.write("</head>\r\n");  
out.write("<body>\r\n");  
out.write("\t呵呵\r\n");  
out.write("\t");  
  
out.print("<h1>Hello</h1>");  
  
out.write("\r\n");  
out.write("</body>\r\n");  
out.write("</html>");
```

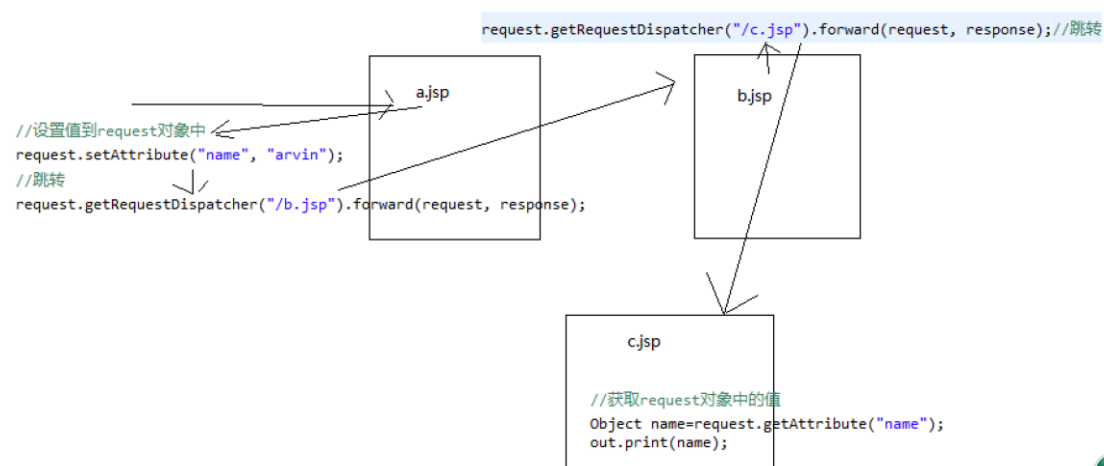
request

原理图(schematic)

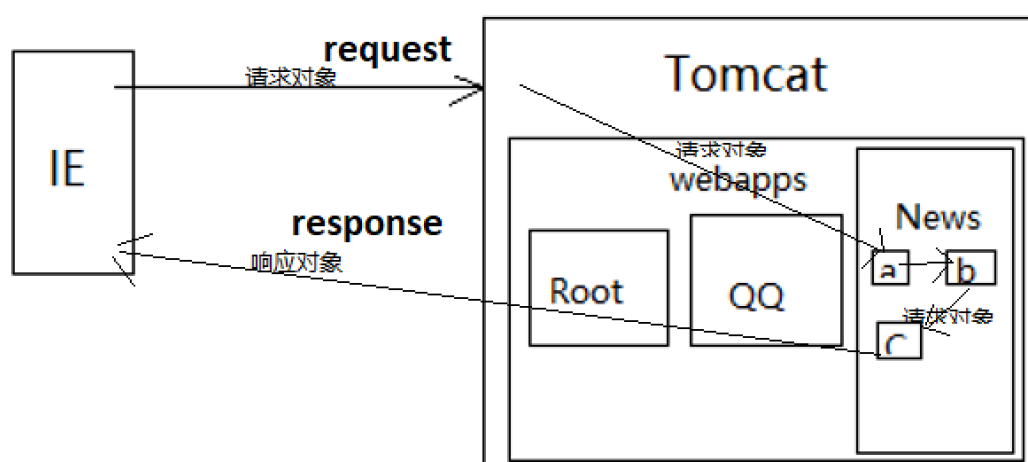


http://localhost:8080/News/a.jsp

Request 实现跳转是服务器内部的周转，数据可以一直携带下去,地址栏不发生改变



一次完整的请求



作用(effect):实现跳转(转发),数据共享,多次跳转地址栏不变

API

```

a.jsp
a
<%
    request.setAttribute("name", "arvin");//设置值到request对象中
    request.getRequestDispatcher("/b.jsp").forward(request, response);//跳转
%>
</body>

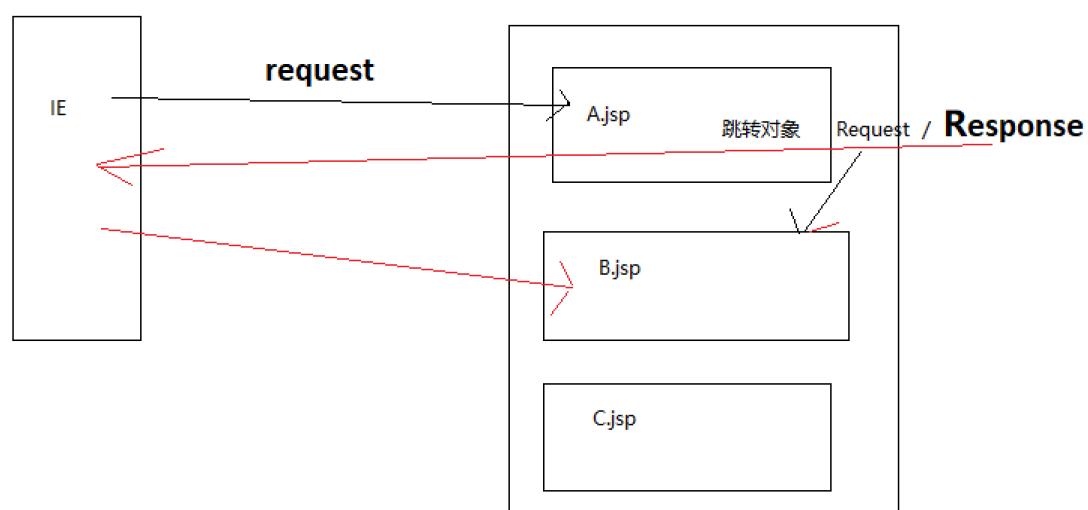
b.jsp
<body>
b
<%
    request.getRequestDispatcher("/c.jsp").forward(request, response);//跳转
%>
</body>

c.jsp
<body>
c
<%
    //可以取到值
    Object name=request.getAttribute("name");//获取request对象中的值
    out.print(name);
%>
</body>

```

response

原理图(schematic)



作用(effect): 实现页面跳转(重定向),地址栏发生改变,数据不共享

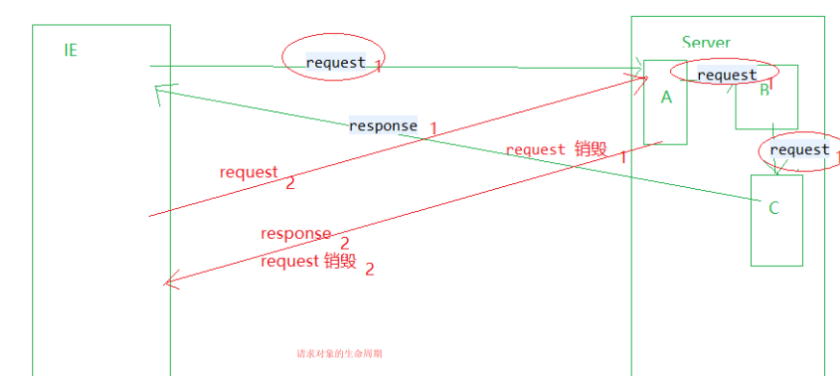
API

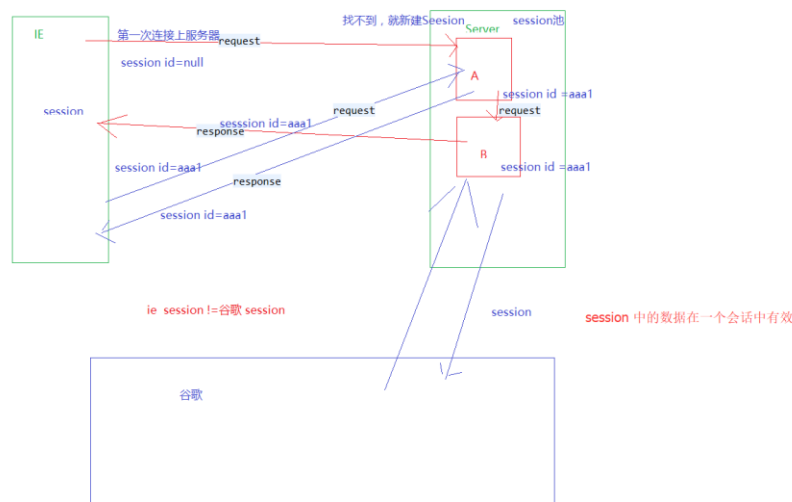
```
*a.jsp
<title>Insert title here</title>
</head>
<body>
  a
  <%
    request.setAttribute("name", "arvin");//设置值到request对象中
    response.sendRedirect("b.jsp");//跳转
  %>
</body>
</html>

*b.jsp
<title>Insert title here</title>
</head>
<body>
  b
  <%
    //取不到值
    Object name=request.getAttribute("name");
    out.print(name);
  %>
</body>
```

session

原理图(schematic)





作用(effect): 存储需要较长时间用时的数据

API

```
a.jsp
<%
//设置值到request对象中
request.setAttribute("requestName", "arvin-request");
//设置值到session对象中
session.setAttribute("sessionName", "jack-session");
//重定向
//response.sendRedirect("b.jsp");
//转发
request.getRequestDispatcher("/b.jsp").forward(request, response);
%>

b.jsp
<%
//获取request对象中的值
Object requestName=request.getAttribute("requestName");
out.print("requestName:"+requestName);

//获取session对象中的值
Object sessionName=session.getAttribute("sessionName");
out.print("sessionName:"+sessionName);
%>
</body>
```



```
<body>
  a
  <br/>
  <%
    String sessionId=session.getId();
    out.print("sessionId:"+sessionId);
  %>
  <br/>
  <a href="b.jsp">to_b</a>
</body>
</html>
</b.jsp>
<body>
  b
  <br/>
  <%
    String sessionId=session.getId();
    out.print("sessionId:"+sessionId);
  %>
  <a href="a.jsp">to_a</a>
</body>
<%
  //销毁session
  session.invalidate();
%>
<%
  //session 默认30分钟后自动销毁

  //销毁session
  //session.invalidate();

  //定时销毁session
  session.setMaxInactiveInterval(5); //5秒
%>
```

application

作用(effect):存储应用程序中长期缓存的数据

API



```
a.jsp
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    A
    <%
        application.setAttribute("applicationName", "Lucy");
    %>
</body>
</html>

b.jsp
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    B
    <%=application.getAttribute("applicationName") %>
</body>
</html>
```

exception

作用(effect):JSP 错误控制

pageContext

作用(effect):获取其他八个内置对象

API

```
<%  
pageContext.getSession();  
pageContext.getRequest();  
%>
```

config

作用(effect):读取配置的信息

API

```
@WebServlet(  
    urlPatterns = "/a",  
    initParams = {  
        @WebInitParam(name = "encoding", value = "UTF-8"),  
        @WebInitParam(name = "fileUrl", value = "D:xxx/xxx/xx.doc")  
    }  
)  
...  
@Override  
public void init(ServletConfig config) throws ServletException {  
    System.out.println("-----init");  
  
    String encoding = config.getInitParameter("encoding");  
    String fileUrl = config.getInitParameter("fileUrl");  
  
    System.out.println("encoding:" + encoding);  
    System.out.println("fileUrl:" + fileUrl);  
}
```

EL 表达式

四大作用域详解

Action scope	Gets the value inside the scope	General way	Scope of scope
Page	<code>\${pageScope.attr}</code>	<code>\${ attr}</code>	
Request	<code>\${requestScope.attr}</code>	<code>\${ attr}</code>	
Session	<code>\${sessionScope.attr}</code>	<code>\${ attr}</code>	
Application	<code>\${applicationScope.attr}</code>	<code>\${ attr}</code>	

案例

```
<%  
    //页面上面不允许出现一行java代码  
    request.setAttribute("a", "request");  
    session.setAttribute("b", "session");  
    application.setAttribute("c", "application");  
%>
```

<!-- 准确取出某一个作用域下的属性 -->

```
<b/>${requestScope.a }  
<b/>${sessionScope.b }  
<b/>${applicationScope.c }
```

<!-- 在四作用域之间从小往大的方向去查找属性 -->

```
<b/>${a }<!-- page --request -->  
<b/>${b }<!-- page --request--session-->  
<b/>${c }<!-- page --request--session--application-->
```

Day02

JavaEE

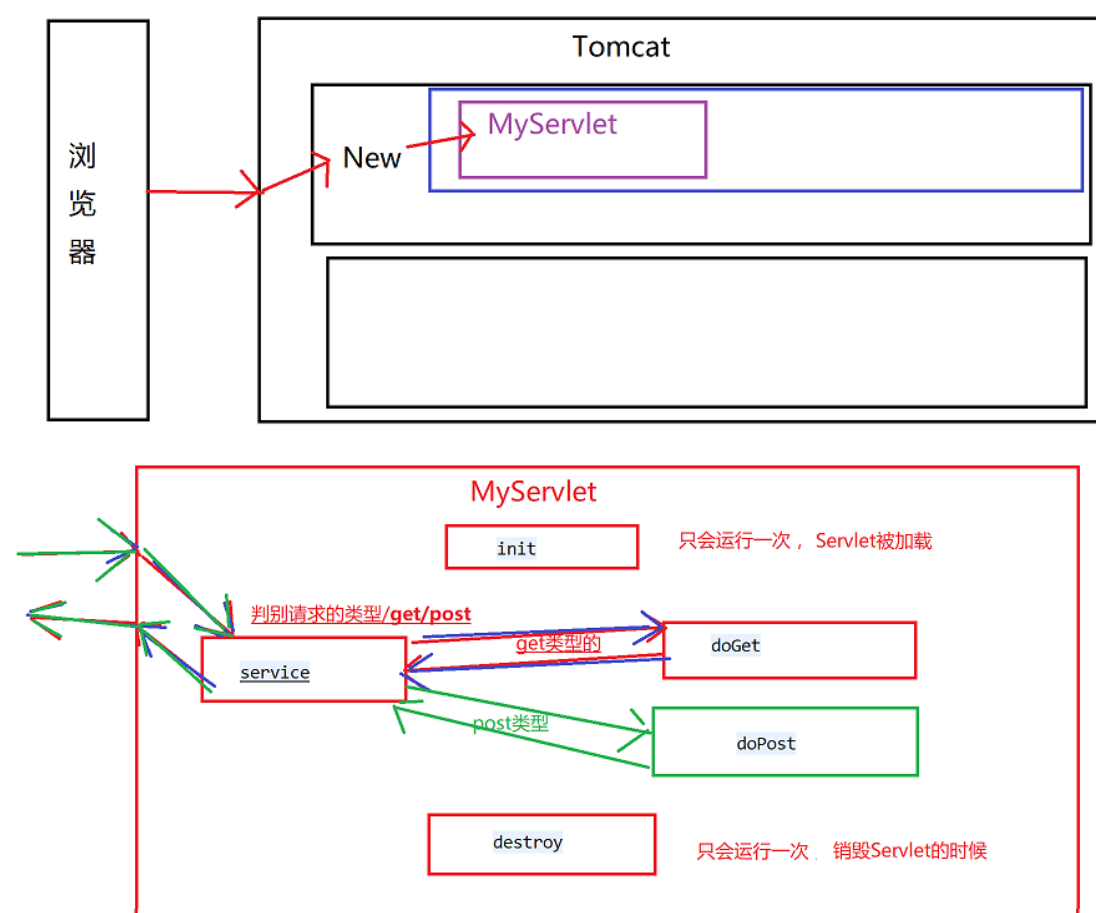
JSTL 表达式

分支(branch)

循环(ciculation)

Servlet

生命周期(life cycle)/原理图(schematic)

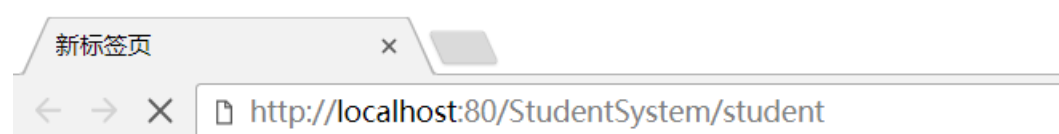


API

请求 GET/POST

GET

地址栏直接编写 URL,默认就是 GET



表单请求,指定请求类型 **method="get"**

```
<form action="http://localhost:80/StudentSystem/student" method="get">
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
```

POST

表单请求,指定请求类型 **method="post"**

```
<form action="http://localhost:80/StudentSystem/student" method="post">
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
```

Web2.0 解决方案: web.xml 开发模式

Web3.0 解决方案: 注解开发模式

备注:

无需 web.xml 文件,直接使用 **@WebServlet** 注解

示例:

```
//http://ip:port/projectName/resource
//http://localhost:80/StudentSystem/student
@WebServlet("/student")
public class StudentServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        this.doPost(req, resp);
    }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    }
}
```

@WebServlet 注解详解

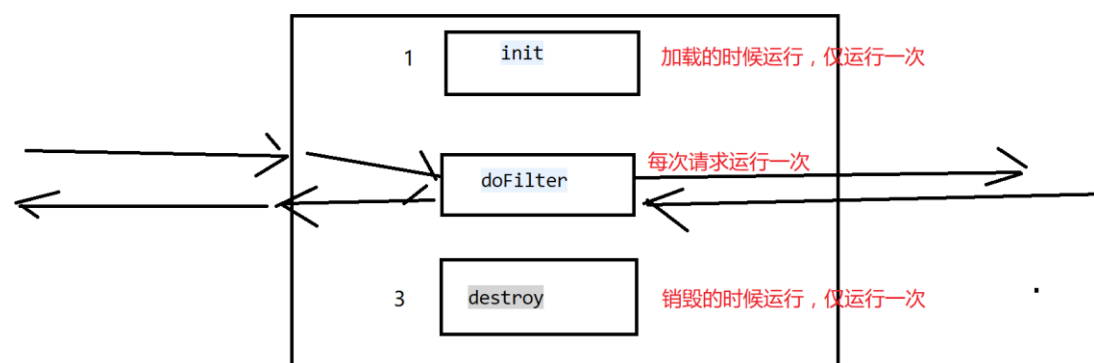
Attribute	Describe	Case
urlPatterns	请求地址栏的路径	@WebServlet(urlPatterns="/student", initParams={ @WebInitParam(name="encoding",value="UTF-8") }, description="注解版本的Servlet", loadOnStartup=2) public class StudentServlet extends HttpServlet { }
initParams	给配置对象设置数据	
description	当前 Servlet 的描述	
loadOnStartup	加载级别默认-1,大于等于 0 时容器启动不便会加载,数字越大加载优先级越高	

Day03

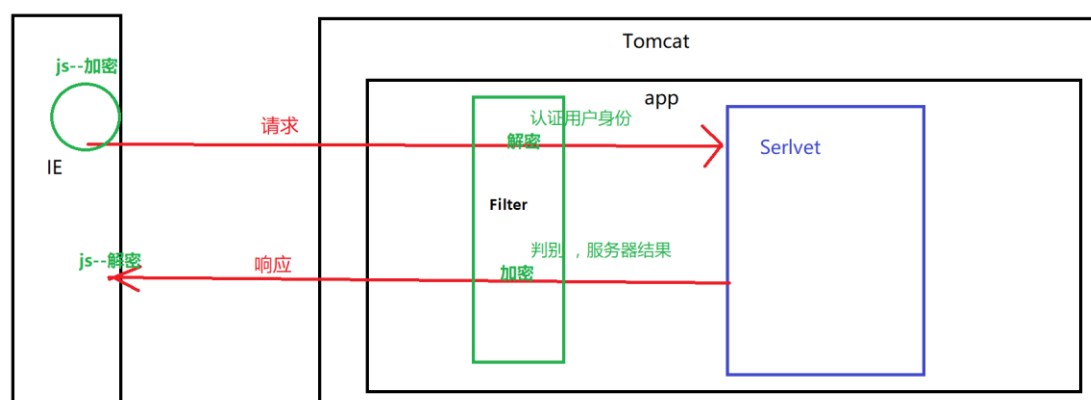
JavaEE

Filter

生命周期(life cycle)



原理图(schematic)



API

```
@WebFilter(  
    urlPatterns="/*",  
    initParams={
```



```
        @WebInitParam(name="encoding",value="UTF-8")
    }
)
public class EnCodingFilter implements Filter {
    private String encoding;

    @Override
    public void destroy() {
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain
filterChain)
        throws IOException, ServletException {
        request.setCharacterEncoding(encoding);
        response.setCharacterEncoding(encoding);
        filterChain.doFilter(request, response);
    }

    @Override
    public void init(FilterConfig config) throws ServletException {
        encoding=config.getInitParameter("encoding");
    }
}
```

Web2.0 解决方案：web.xml 开发模式

Web3.0 解决方案：注解开发模式

```
@WebFilter(
    urlPatterns="/*",
    initParams={
        @WebInitParam(name="encoding",value="UTF-8")
    }
)
public class EnCodingFilter implements Filter {
```

四大监听器/Listener

简介(intro)

Listener	Function
HttpSessionListener	Session created and destroyed
HttpSessionAttributeListener	Session attribute add and remove add replace
ServletContextListener	Application initialized add destroyed
ServletContextAttributeListener	Application attribute add and remove add replace

四大监听器详解

HttpSessionListener

API

```
@Override
public void sessionCreated(HttpSessionEvent arg0) {
    System.out.println("session 构建成功");
}

@Override
public void sessionDestroyed(HttpSessionEvent arg0) {
    System.out.println("session 销毁成功");
}
```

HttpSessionAttributeListener

API

```
@Override
public void attributeAdded(HttpSessionBindingEvent arg0) {
    System.out.println("session开始添加数据");
}

@Override
public void attributeRemoved(HttpSessionBindingEvent arg0) {
    System.out.println("session开始删除数据");
}
```

```
}

@Override
public void attributeReplaced(HttpSessionBindingEvent arg0) {
    System.out.println("session开始替换数据");
}
```

ServletContextListener

API

```
@Override
public void contextDestroyed(ServletContextEvent arg0) {
    //application 销毁
}

@Override
public void contextInitialized(ServletContextEvent arg0) {
    //application构建
}
```

ServletContextAttributeListener

API

```
@Override
public void attributeAdded(ServletContextAttributeEvent arg0) {
    //application 属性添加
}

@Override
public void attributeRemoved(ServletContextAttributeEvent arg0) {
    //application 属性删除
}

@Override
public void attributeReplaced(ServletContextAttributeEvent arg0) {
    //application 属性覆盖
}
```

Web2.0 解决方案：web.xml 开发模式

Web3.0 解决方案：注解开发模式

```
@WebListener //注解  
public class DemoListener  
implements HttpSessionListener, HttpSessionAttributeListener,  
ServletContextListener, ServletContextAttributeListener {
```