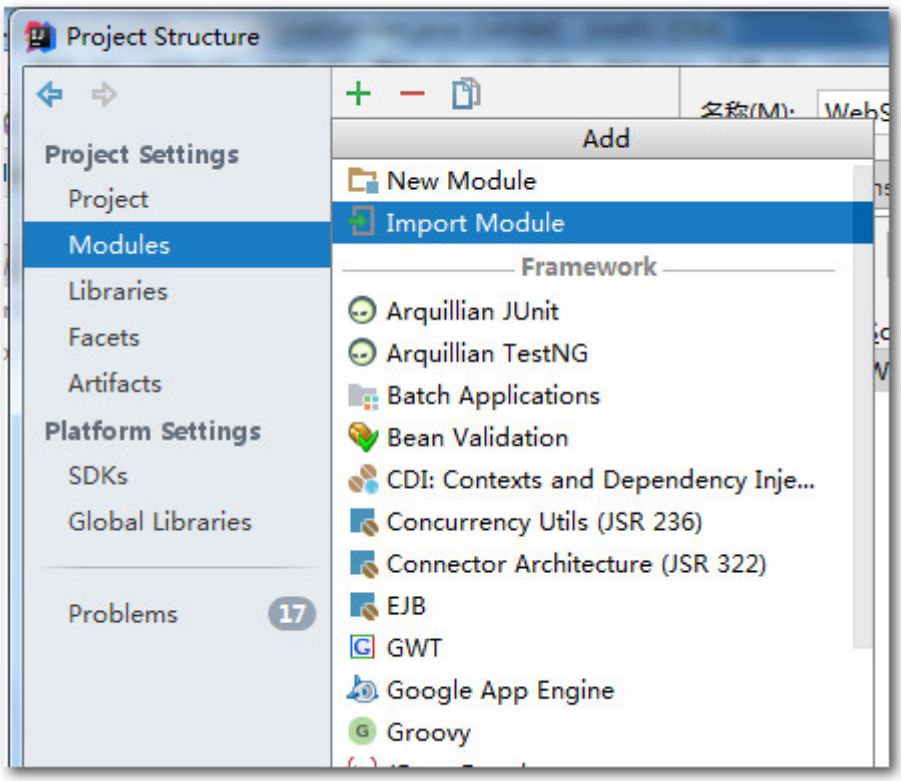
day33-filter_listener

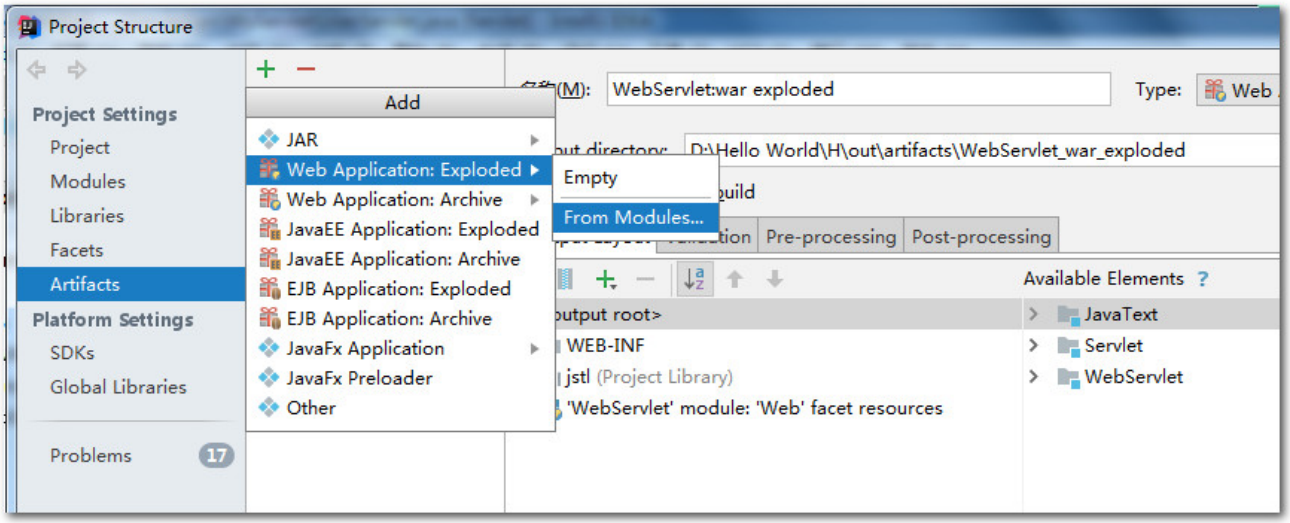
学习目标

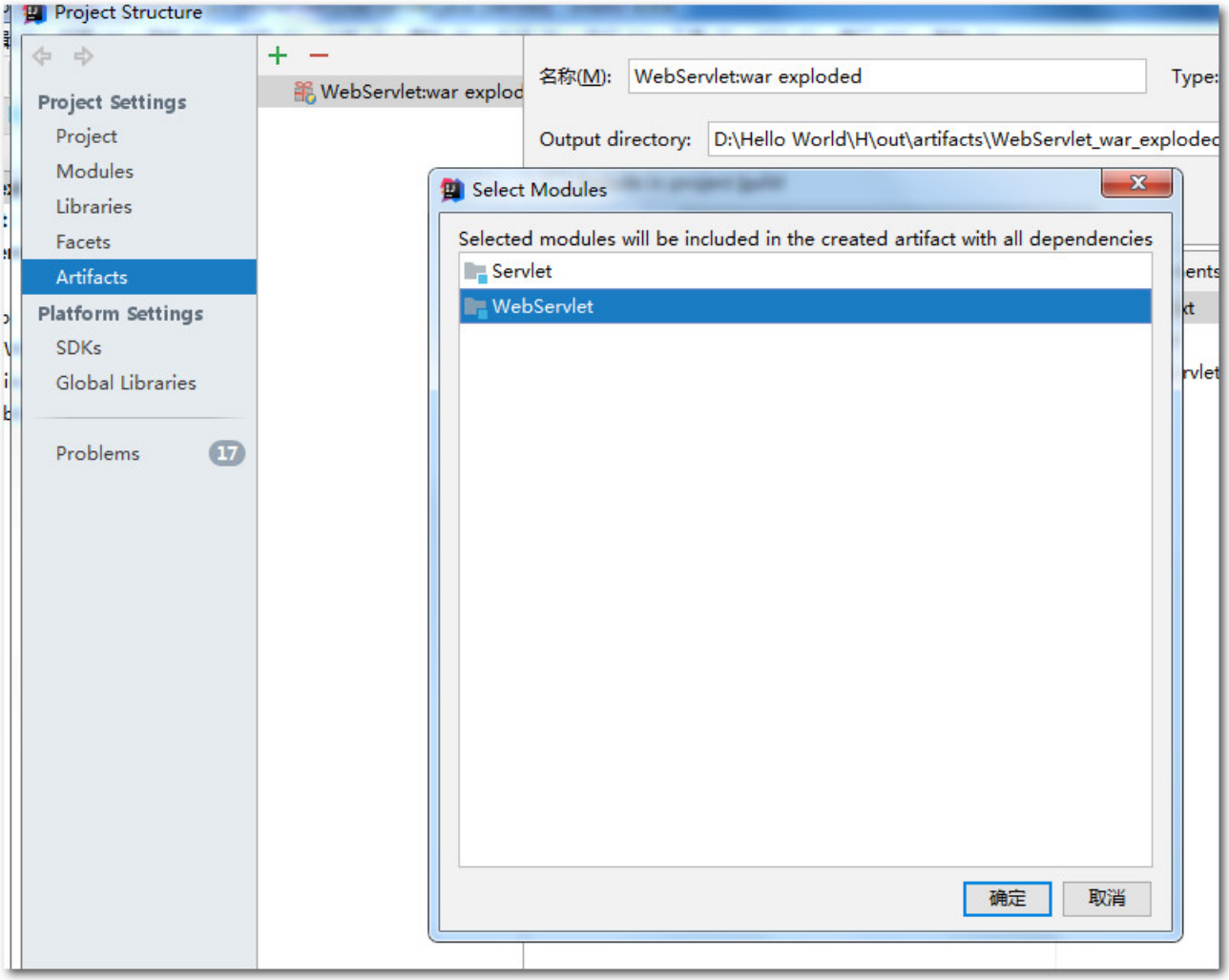
1. 能够说出过滤器的作用
2. 能够编写过滤器
3. 能够说出过滤器生命周期相关方法
4. 能够根据过滤路径判断指定的过滤器是否起作用
5. 能够说出什么是过滤器链
6. 能够编写过滤器解决全局乱码

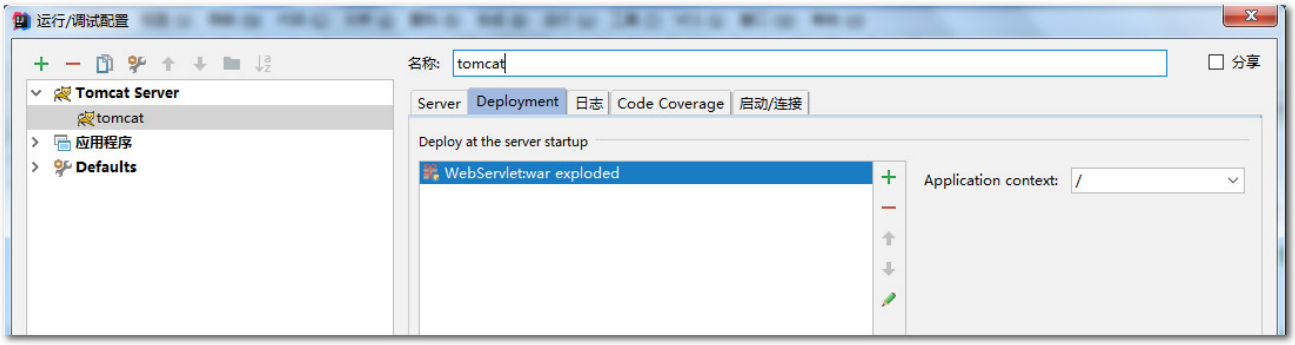
案例一:统一全网站中文乱码的处理

一,需求分析　更换项目服务器部署




Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
17
Add
JAR
Web Application: Exploded
Web Application: Archive
JavaEE Application: Exploded
JavaEE Application: Archive
EJB Application: Exploded
EJB Application: Archive
JavaFx Application
JavaFx Preloader
Other
Empty
From Modules...
WebServlet:war exploded
Type:
D:\Hello World\H\out\artifacts\WebServlet_war_exploded
Pre-processing
Post-processing
Available Elements
JavaText
Servlet
WebServlet
WEB-INF
jstl (Project Library)
'WebServlet' module: 'Web' facet resources


Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
17
WebServlet:war explod
名称(M):
WebServlet:war exploded
Type:
Output directory:
D:\Hello World\H\out\artifacts\WebServlet_war_explodec
Select Modules
Selected modules will be included in the created artifact with all dependencies
Servlet
WebServlet
确定
取消



在整个网站中,可能会有get请求或post请求向服务器提交参数.参数中往往有中文信息.在后台每个Servlet中都需要去处理乱码.

我们想做的是：请求到达Servlet中.就可以直接调用getParameter方法获得请求参数,请求参数已经没有乱码了.

二,技术分析

1.Filter概述

1.1什么是filter

Filter：一个实现了特殊接口(Filter)的Java类.实现对请求资源(jsp,servlet,html,)的过滤的功能.

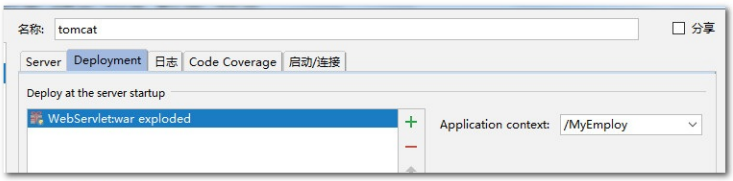
过滤器是一个运行在服务器的程序,优先于请求资源(Servlet或者jsp,html)之前执行. 过滤器是javaweb技术中**最为实用**的技术.

1.2过滤器的作用

对目标资源(Servlet,jsp)进行过滤.

应用场景:登录权限检查,解决网站乱码,过滤敏感字符 ...

相对路径



访问首页为http://localhost/MyEmploy/index.jsp

```
<body>
<a href="ms">跳转过滤器</a><br/>
</body>
```

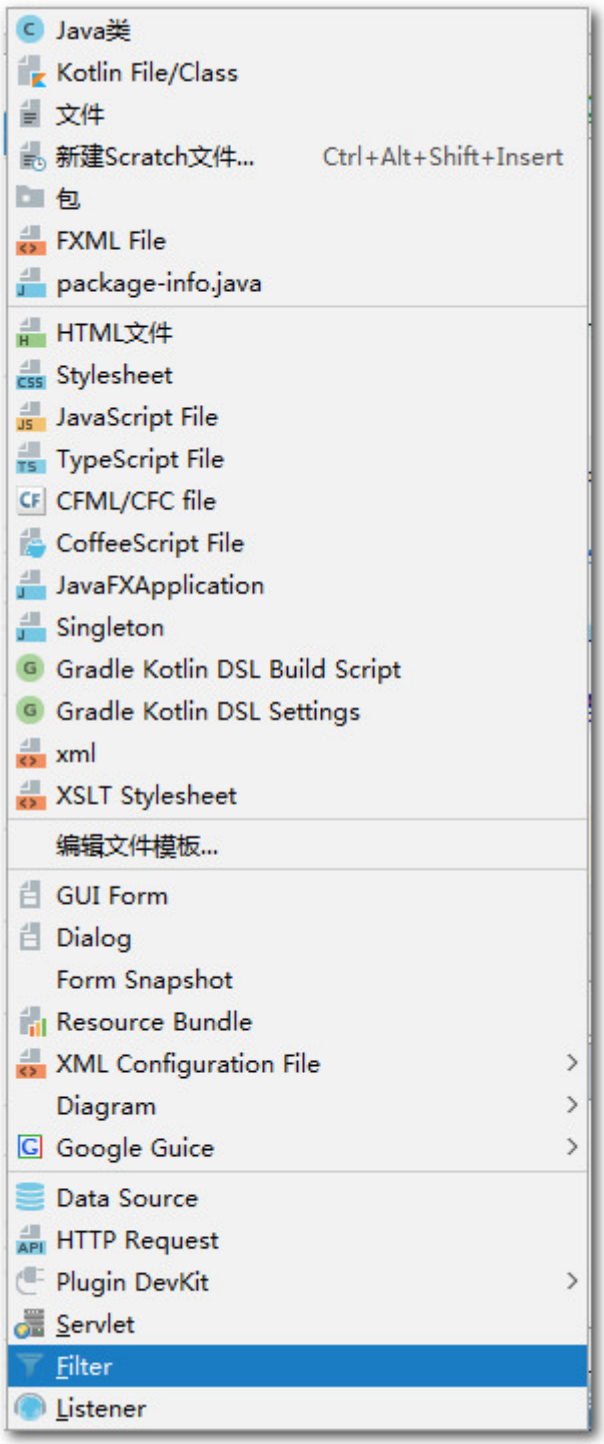
需要跳转的Servlet为http://localhost/MyEmploy/ms
与首页在同一个文件夹下，因此不需要加/

```
@WebServlet("/ms")
public class MyServlet extends HttpServlet
```

2. filter入门

2.1 通过xml配置方式

- 创建一个类实现Filter接口



- 在web.xml对过滤器进行配置

```
<filter>
    <filter-name>FilterUtils</filter-name>
    <filter-class>MyServlet.MyFilter</filter-class>
</filter>
    <filter-mapping>
        <filter-name>FilterUtils</filter-name>
        <url-pattern>/ms</url-pattern>
    </filter-mapping>
```

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
  <head>
    <title>index</title>
  </head>
  <body>
  <a href="ms">跳转过滤器</a><br/>
  </body>
</html>
```

```
public class MyFilter implements Filter {
    public void destroy() {
    }
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws
ServletException, IOException {
        System.out.println("过滤器收到了请求");
        //放行
        chain.doFilter(req,resp);
    }
    public void init(FilterConfig config) throws ServletException {
    }
}
```

```
@WebServlet("/ms")
public class MyServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doGet(request,response);
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        System.out.println("Servlet运行了");
        response.getWriter().print("页面Servlet运行了");
    }
}
```

2.2通过注解方式

- 创建一个类实现Filter接口
- **直接在这个类上面添加注解进行配置**

```
@WebFilter("/ms")
public class MyFilter implements Filter {
    public void destroy() {
    }
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws
ServletException, IOException {
        System.out.println("过滤器收到了请求");
        //放行
        chain.doFilter(req,resp);
    }
    public void init(FilterConfig config) throws ServletException {
    }
}
```

三,乱码思路分析

- **在过滤器中将request通过装饰者或者动态代理进行增强**
- **增强getParameter()方法,使代码变成utf-8编码**

四,代码实现

- **主页**

```
<form action="ms"method="post">
<input type="text" name="user"><br/>
<input type="password" name="psw"><br/>
  <input type="submit"/>
</form>
```

- **过滤器**

```
@WebFilter("/*")
```

```
public class MyFilter implements Filter {
    public void destroy() {
    }
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws
ServletException, IOException {
        HttpServletRequest hsr= (HttpServletRequest) req;
        HttpServletResponse hsrpon= (HttpServletResponse) resp;
        HttpServletRequest sr = (HttpServletRequest)
Proxy.newProxyInstance(hsr.getClass().getClassLoader(), hsr.getClass().getInterfaces(), new
InvocationHandler() {
            @Override
            public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
                if("getParameter".equals(method.getName())){
                    if("post".equalsIgnoreCase(hsr.getMethod())){
                    hsr.setCharacterEncoding("utf-8");
                    }
                }
                return method.invoke(hsr,args);
            }
        });
        HttpServletResponse hs = (HttpServletResponse)
Proxy.newProxyInstance(hsrpon.getClass().getClassLoader(), hsrpon.getClass().getInterfaces(),
new InvocationHandler() {
            @Override
            public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
                    hsrpon.setContentType("text/html;charset=utf-8");
                return method.invoke(hsrpon,args);
            }
        });
        chain.doFilter(sr,hs);
    }
    public void init(FilterConfig config) throws ServletException {
    }
}
```

- **Servlet**

```
@WebServlet("/ms")
public class MyServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doGet(request,response);
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        System.out.println("Servlet运行
了"+request.getParameter("user")+request.getParameter("psw"));
        response.getWriter().print("页面Servlet运行
了"+request.getParameter("user")+request.getParameter("psw"));
    }
}
```

五,总结

1.Filter的生命周期

1.1Filter生命周期

过滤器从创建到销毁的过程

1.2生命周期方法

init(FilterConfig):初始化

注解方式

```
@WebFilter(urlPatterns = "/*",initParams = @WebInitParam(name = "key",value="aaa"))
public class MyFilter implements Filter {
    public void init(FilterConfig config) throws ServletException {
        System.out.println(config.getInitParameter("key"));
    }
    }
```

配置文件方式

```
<filter>
    <filter-name>My</filter-name>
    <filter-class>MyServlet.MyFilter</filter-class>
    <init-param>
        <param-name>key</param-name>
        <param-value>aaa</param-value>
    </init-param>
</filter>
    <filter-mapping>
        <filter-name>My</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

doFilter(ServletReqeust req,ServletResponse resp,FilterChain chain):执行过滤的方法

destroy():销毁

1.3Filter生命周期描述

1. 当服务器启动的时候，过滤器就被初始化了，执行过滤器的init方法
2. 每当一个请求的路径是满足过滤器的配置路径，那么就会执行一次过滤器的doFilter方法
3. 当服务器停止的时候，销毁过滤器，执行过滤器的destory方法

1.4Servlet服务器启动时创建

```
@WebServlet(value="/ms",loadOnStartup = 1)
```

2.映射路径

假设有一个管理员权限的过滤器，它应该对用户发出的管理员功能的请求进行条件的过滤。但是当用户发出登录、注册等请求的时候，不应该进行过滤。所以我们过滤器，应该有选择的过滤器请求。这就需要学习配置过滤器不同的映射路径，从而让过滤器过滤希望过滤器的请求。

- 完全路径匹配, 以"/"开始

```
/demo01 ---> 过滤器只能拦截路径/demo01;  /aa/a.jsp ---> 只能拦截  aa目录下的a.jsp
```

- 目录匹配:以"/"开始 以 *结束 .

```
/* --->当前项目下的所有的路径都可以拦截;   /aa/*  ---> 可以拦截 /aa/bb, /aa/bb/cc
```

- 扩展名匹配:以"*"开始 例如: *.jsp *.do

```
*.do--->可以拦截路径的后缀是 do的 ;  *.jsp--->只能拦截JSP(所有的)
```

3.拦截方式

有了上面学习的映射路径，我们可以控制过滤器过滤指定的内容，但是我们在访问资源的时候，并不是每次都是之间访问，有时是以转发的方式访问的，这就需要我们要让过滤器可以区分不同的访问资源的方式，有不同的拦截方式。 是通过

DispatcherType 来指定的.

3.1DispatcherType.REQUEST

默认值,过滤从浏览器发送过来的请求和重定向 不过滤转发

3.2 DispatcherType.FORWARD

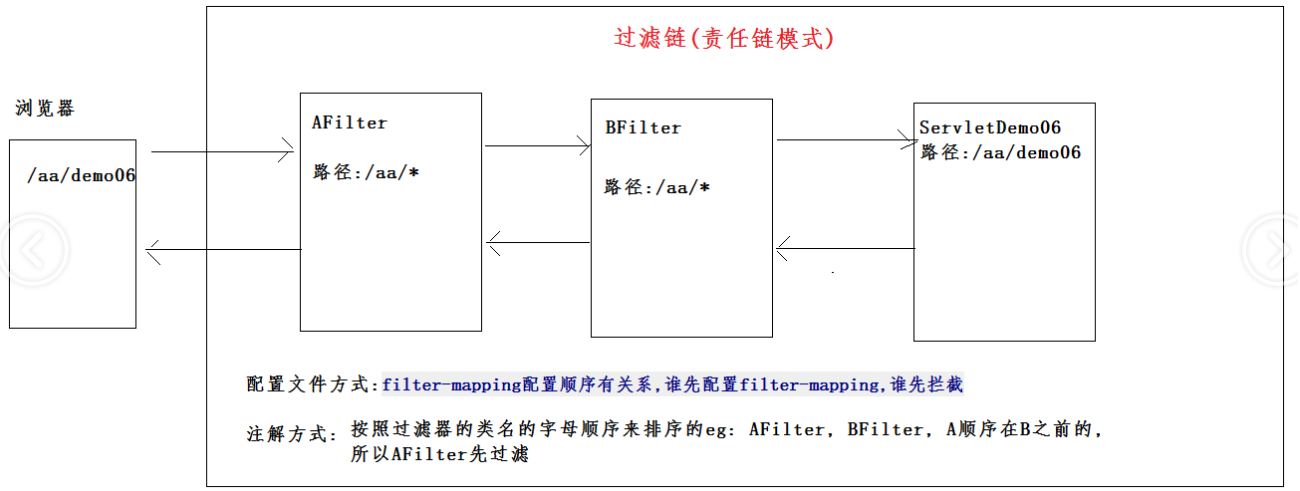
只过滤转发过来的请求

```
/*只拦截转发*/
/*@WebFilter(urlPatterns = "/demo05",dispatcherTypes={DispatcherType.FORWARD})*/

/*转发,重定向,请求都会拦截,逗号隔开*/
@WebFilter(urlPatterns = "/demo05",dispatcherTypes=
{DispatcherType.FORWARD,DispatcherType.REQUEST})
public class FilterDemo05 implements Filter {

    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws
ServletException, IOException {
        System.out.println("FilterDemo05()  收到了请求...");
        chain.doFilter(req, resp);
    }
    public void destroy() {
    }
    public void init(FilterConfig config) throws ServletException {
    }
```

```
}
```

4.过滤器链

过滤器链作用：当一个filter收到请求的时候,调用chain.doFilter才可以访问下一个匹配的filter,若当前的filter是最后一 个filter,调用chain.doFilter才能访问目标资源

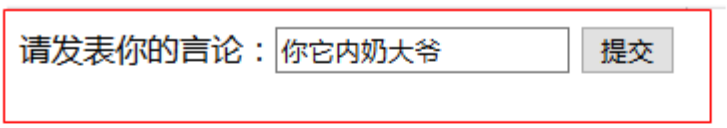


案例二:非法字符

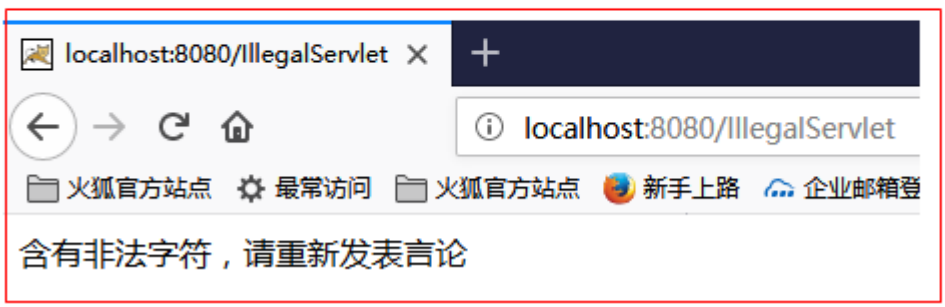
一,需求分析

当用户发出非法言论的时候，提示用户言论非法。

效果:



页面输入



响应

二,思路分析

1.创建一个表单用于发表言论。

2.创建一个txt文件，其中存入非法字符。

3.创建一个Filter，拦截请求。在init方法中将txt文件中的非法字符读取到内存中。

4.获取请求中的参数，对请求的参数进行非法字符的校验。

5.如果言论中不含有非法字符，就放行。

6.如果言论中含有非法字符，就拦截，并且提示用户非法言论。

三,代码实现

- 首页

```
<a href="/MyEmploy/form.jsp">过滤器</a>
```

- 表单

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <form method="post" action="pin">
        请发表你的言论：<input type="text" name="message">
        <input type="submit" value="提交">
    </form>
</body>
</html>
```

- Servlet

```
@WebServlet("/pin")
public class PinServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doGet(request,response);
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html;charset=utf-8");
        String message = request.getParameter("message");
        response.getWriter().print(message);
    }
}
```

- Filter

```
@WebFilter("/pin")
```

```
public class PinFilter implements Filter {
    List<String> list=new ArrayList<>();
    public void destroy() {
    }
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws
ServletException, IOException {
        HttpServletRequest hreq= (HttpServletRequest) req;
        HttpServletResponse hres= (HttpServletResponse) resp;
        hreq.setCharacterEncoding("utf-8");
        hres.setContentType("text/html;charset=utf-8");
        String message = hreq.getParameter("message");
        for (String word : list) {
            if(message != null && message.trim().contains(word)){
                //直接给用户提示
                hres.getWriter().print("您发布的言论里面包含敏感字符,请重新输入!!!");
                return;
            }
        return;
        }
        chain.doFilter(hreq, hres);
    }
    public void init(FilterConfig config)throws ServletException {
        try {
            ServletContext sc=config.getServletContext();
            InputStream is = sc.getResourceAsStream("/IllegalWords.txt");
            BufferedReader bw=new BufferedReader(new InputStreamReader(is,"utf-8"));
            String str;
            while((str = bw.readLine())!=null){
                list.add(str);
            }
            System.out.println(list);
            bw.close();
            is.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

案例三:监听ServletContext的创建和销毁

一,案例需求

监听ServletContext域对象的创建和销毁.

问题: ServletContext什么时候创建,什么时候销毁?

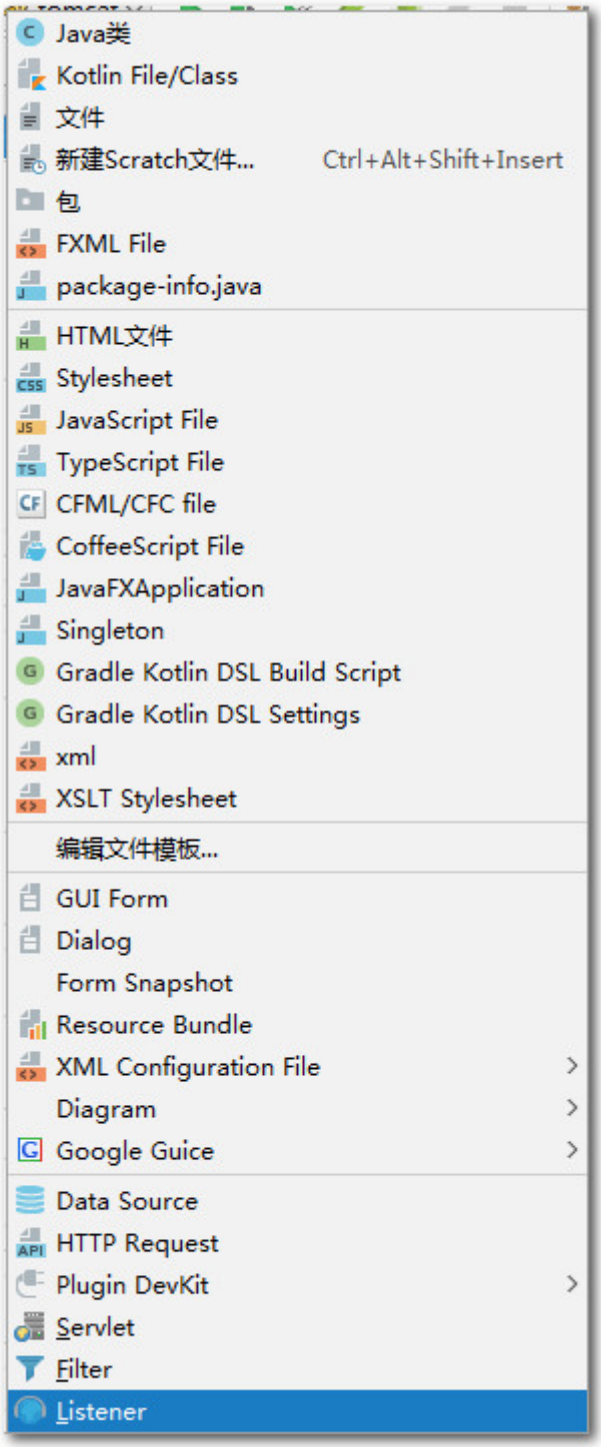
服务器启动的时候创建

服务器正常关闭的时候销毁

我这个案例表面上是监听ServletContext的创建和销毁, 实际上是监听服务器启动和关闭的

二,技术分析

1.Listener的概述



1.1什么是Listener

监听器就是一个Java类,用来监听其他的JavaBean的变化

在javaweb中监听器就是监听**三个域对象**的状态的。request,session,servletContext(application)

1.2监听器的应用

主要在Swing编程

在Android大量应用

JS里面的事件

1.3监听器的术语

eg: 一个狗仔拍明星出轨

事件源 :被监听的对象.(目标对象) 明星

监听器对象：监听的对象. 狗仔

事件：事件源行为的称呼. 出轨

注册(绑定):将"监听器"对象注册给"事件源". 狗仔和明星进行绑定(去楼下蹲点,跟踪..)

事件对象:在"监听器对象"中获得"事件源" 在狗仔的镜头下可以找到出轨证据

2.ServletContextListener

2.1概述

ServletContextListener是监听ServletContext对象的创建和销毁的

问题 :ServletContext对象何时创建和销毁:

创建：服务器启动时候(前提是这个项目在这个服务器里面). 服务器为每个WEB应用创建一个单独的ServletContext.

销毁：服务器关闭的时候,或者项目从服务器中移除.

2.2ServletContextListener的使用

2.2.1编写步骤

1.编写一个类,实现ServletContextListener接口

2.在web.xml里面注册注册listener

2.2.2代码实现

- xml配置

```
    <listener>
        <listener-class>TomListener</listener-class>
    </listener>
```

- JAVA代码

```
public class TomListener implements ServletContextListener{
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("服务器启动");
    }
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("服务器关闭");
    }
}
```

2.3ServletContextListener企业中应用

初始化工作.

加载配置文件:Spring框架,ContextLoaderListener

常见设计模式:

- 单例模式
- 装饰者
- 代理(动态代理)
- 工厂模式
- 构建模式
- 责任链
- 类模板模式(模板方法)
- 观察者

MD5加密

```
import java.security.MessageDigest;

public  class Md5Util {
    private Md5Util(){}
    public static String encodeByMd5(String password) throws Exception{
        MessageDigest md5 = MessageDigest.getInstance("MD5");
        byte[] byteArray = md5.digest(password.getBytes());
        return byteArrayToHexString(byteArray);
    }

    private static String byteArrayToHexString(byte[] byteArray) {
        StringBuffer sb = new StringBuffer();
        for(byte b : byteArray){
            String hex = byteToHexString(b);
            sb.append(hex);
        }
        return sb.toString();
    }
    private static String byteToHexString(byte b) {
        int n = b;
        if(n < 0){
```

```
            n = 256 + n;
        }
        int d1 = n / 16;
        int d2 = n % 16;
        return hex[d1] + hex[d2];
    }
    private static String[] hex =
{"0","1","2","3","4","5","6","7","8","9","a","b","c","d","e","f"};
}
```

```
import MD5.Md5Util;
public class Demo {
    public static void main(String[] args) throws Exception {
        String str="123456";
        System.out.println("加密前"+str);
        String s = Md5Util.encodeByMd5(str);
        System.out.println("加密后"+s);
    }
}
```