

【济南中心】JavaEE就业班同步笔记第一阶段：Listener和Filter

小鲁哥哥 • 2017-3-11 17:22:46

【济南中心】JavaEE就业班同步笔记第一阶段： JavaWeb之高级技术--Listener和Filter

1. 监听器的概述：

1.1.1 什么是监听器：

* 监听器：就是一个Java类,用来监听其他的Java的状态的变化.

1.1.2 监听器的用途：

* 用来监听其他的对象的变化.

* 主要应用在图形化界面中比较多：

* GUI,Android.

1.1.3 监听器的术语：

* 事件源:指的是被监听的对象.(汽车)

* 监听器:指的是监听的对象.(报警器)

* 事件源与监听器的绑定:就是在汽车上安装报警器.

* 事件:指的是事件源的改变.(踹汽车一脚)---主要的功能获得事件源对象.

1.2 WEB中的监听器的概述：

1.2.1 WEB中的监听器：

WEB中的Listener和Filter是属于Servlet规范中的高级的技术.

WEB中的监听器共有三类八种(监听三个域对象)

* 事件源：Servlet中的三个域对象.ServletContext,HttpSession,ServletRequest.

* 监听器：自定义类实现8个接口.

三类八种：

* 一类：监听三个域对象的创建和销毁的监听器：

* ServletContextListener

* HttpSessionListener

- * ServletRequestListener

- * 二类： 监听三个域对象的属性变更的监听器(属性添加,移除,替换):

- * ServletContextAttributeListener

- * HttpSessionAttributeListener

- * ServletRequestAttributeListener

- * 三类： 监听HttpSession中的JavaBean的状态改变(绑定,解除绑定,钝化,活化)

- * HttpSessionBindingListener

- * HttpSessionActivationListener

1.2.3 WEB中的监听器的使用:

编写一个类实现监听器的接口:

通过配置文件配置监听器:

1.3 一类： 监听三个域对象的创建和销毁的监听器:

1.3.1 ServletContextListener:监听ServletContext对象的创建和销毁:

【方法】:

contextDestroyed()

contextInitialized()

【ServletContext对象的创建和销毁】:

- * 创建： 服务器启动的时候,服务器为每个WEB应用创建一个属于该web项目的对象ServletContext.

- * 销毁： 服务器关闭或者项目从服务器中移除的时候.

【应用代码】

```
[mw_shl_code=java,true]public class MyServletContextListener implements ServletContextListener{  
    @Override  
    public void contextInitialized(ServletContextEvent sce) {  
        System.out.println("ServletContext对象被创建了...");  
    }  
    @Override  
    public void contextDestroyed(ServletContextEvent sce) {  
        System.out.println("ServletContext对象被销毁了...");  
    }  
}[/mw_shl_code]
```

配置:

```
<!-- 配置监听器 -->
```

```
[mw_shl_code=xml,true]<listener>
```

```
    <listener-class>com.itheima.weblistener.MyServletContextListener</listener-class>
```

```
</listener>[/mw_shl_code]
```

【企业中的应用:】

* 1.加载框架的配置文件 :Spring框架 ContextLoaderListener

* 2.定时任务调度:

* Timer,TimerTask

1.3.2 HttpSessionListener:监听HttpSession的创建和销毁的监听器:

【方法】

sessionCreated(HttpSessionEvent)

sessionDestroyed(HttpSessionEvent)

【HttpSession何时创建和销毁的】

* 创建:服务器端第一次调用getSession();

* 销毁:

* 非正常关闭服务器(正常关闭session会序列化):

* session过期了默认30分钟.

* 手动调用session.invalidate();

【HttpSession的问题】

* 访问Servlet会不会创建Session : 不会

* 访问JSP会不会创建Session : 会.

* 访问html会不会创建Session : 不会

【应用的代码】

```
[mw_shl_code=java,true]public class MyHttpSessionListener implements HttpSessionListen
```

```
er{
```

```
    @Override
```

```
    public void sessionCreated(HttpSessionEvent se) {
```

```
        System.out.println("HttpSession被创建了...");
```

```
    }
```

```
    @Override
```

```
    public void sessionDestroyed(HttpSessionEvent se) {
```

```
        System.out.println("HttpSession被销毁了...");
    }
}[/mw_shl_code]
```

配置:

```
[mw_shl_code=xml,true]<listener>
    <listener-class>com.itheima.weblistener.MyHttpSessionListener</listener-class>
</listener>[/mw_shl_code]
```

1.3.3 ServletRequestListener:监听ServletRequest对象的创建和销毁的监听器:

【方法】:

requestDestroyed(ServletRequestEvent)

requestInitialized(ServletRequestEvent)

【request对象何时创建和销毁】:

- * 创建: 客户端向服务器发送一次请求,服务器就会创建request对象.
- * 销毁: 服务器对这次请求作出响应后就会销毁request对象.

【问题】:

访问一个Servlet会不会创建request对象:会

访问一个JSP会不会创建request对象:会

访问一个HTML会不会创建request对象:会

【应用的代码】

```
[mw_shl_code=java,true]public class MyServletRequestListener implements ServletRequest
Listener{
    @Override
    public void requestDestroyed(ServletRequestEvent sre) {
        System.out.println("ServletRequest被销毁了...");
    }
    @Override
    public void requestInitialized(ServletRequestEvent sre) {
        System.out.println("ServletRequest被创建了...");
    }
}[/mw_shl_code]
```

配置:

```
[mw_shl_code=xml,true]<listener>
```

```
<listener-class>com.itheima.weblistener.MyServletRequestListener</listener-class>
</listener>[/mw_shl_code]
```

1.4 二类:监听三个域对象的属性变更的监听器:(属性添加,移除,替换)

1.4.1 ServletContextAttributeListener:监听ServletContext对象的属性变更:

attributeAdded(ServletContextAttributeEvent)

attributeRemoved(ServletContextAttributeEvent)

attributeReplaced(ServletContextAttributeEvent)

1.4.2 HttpSessionAttributeListener:监听HttpSession中的属性变更:

attributeAdded(HttpSessionBindingEvent)

attributeRemoved(HttpSessionBindingEvent)

attributeReplaced(HttpSessionBindingEvent)

1.4.3 ServletRequestAttributeListener: 监听ServletRequest对象的属性变更的:

attributeAdded(ServletRequestEvent)

attributeRemove(ServletRequestEvent)

attributeReplaced(ServletRequestEvent)

1.5 三类: 监听HttpSession中的JavaBean的对象的属性改变的监听器

第三类监听器很特殊,不需要进行配置的.作用在JavaBean上的监听器.JavaBean可以自己感知到在Session中的状态.

1.5.1 HttpSessionBindingListener:监听HttpSession中的JavaBean的绑定和解除绑定的

valueBound(event)

valueUnbound(event)

1.5.2 HttpSessionActivationListener:监听HttpSession中的JavaBean的钝化和活化的.

* sessionDidActivate(); :--活化 (反序列化)

* sessionWillPassivate(); :--钝化 (序列化到硬盘)

***** 优化Session:

* 通过配置<Context>标签配置定时session序列化.

* 在tomcat/conf/context.xml中配置<Context> :在tomcat中所有的虚拟主机和虚拟路径都会按照这个配置执行.

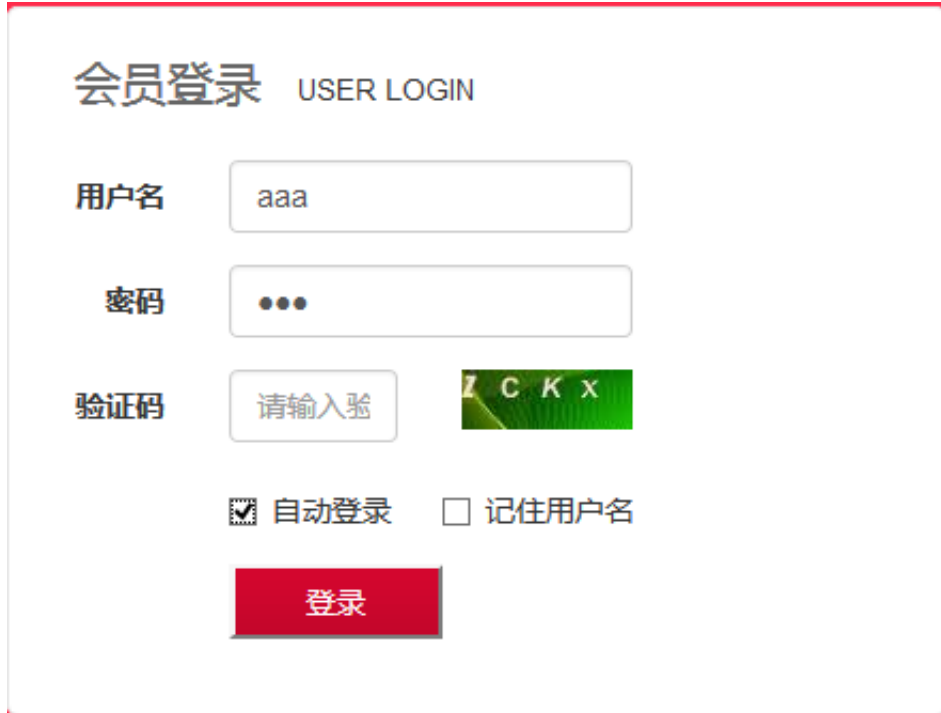
* 在tomcat/conf/Catalina/localhost/context.xml配置<Context> :在tomcat中的localhost虚拟主机中的所有虚拟路径按照这个配置执行.

* 在当前的工程下的META-INF/context.xml配置<Context> :当前这个工程按照配置执行.

2. 案例一：自动登录案例.

2.1 需求:


在各式网站都会看到自动登录的功能,在登录页面中勾选了自动登录的复选框,那么下次访问网站首页的时候,可以不需要进行登录.



会员登录 USER LOGIN

用户名

密码

验证码 

☒ 自动登录 ☐ 记住用户名

2.2 分析:

2.2.1 技术分析:

【Cookie技术】

* 利用Cookie记住用户的用户名和密码.

【Filter:过滤器的概述】

什么是过滤器Filter:可以过滤从客户端向服务器发送的请求.

过滤器的使用:

* 进行IP的过滤,脏话过滤,自动登录,响应压缩...

使用过滤器:

* 编写一个类实现Filter接口:

* 配置过滤器:

【过滤器的生命周期】:了解

过滤器的创建和销毁:

* 创建:服务器启动的时候.

* 销毁:服务器关闭的时候.

【FilterConfig:过滤器的配置对象】：

getFilterName()

getInitParameter()

getInitParameterNames()

getServletContext()

代码:

```
[mw_shl_code=java,true]public void init(FilterConfig filterConfig) throws ServletException {  
    // 获得当前的Filter的名称:  
    String filterName = filterConfig.getFilterName();  
    System.out.println(filterName);  
    // 获得初始化参数:  
    String username = filterConfig.getInitParameter("username");  
    String password = filterConfig.getInitParameter("password");  
    System.out.println(username+" "+password);  
    // 获得所有的初始化参数的名称:  
    Enumeration<String> en = filterConfig.getInitParameterNames();  
    while(en.hasMoreElements()){  
        String name = en.nextElement();  
        String value = filterConfig.getInitParameter(name);  
        System.out.println(name+" "+value);  
    }  
}[/mw_shl_code]
```

【FilterChain:过滤器链】

过滤器链中的过滤器的执行的顺序与<filter-mapping>的配置顺序有关.

* doFilter(request,response); -- 放行,放行到下一个过滤器中,如果没有下一个过滤器,到达目标资源.

【Filter相关的配置】

<url-pattern>的配置:

* 完全路径匹配 : 以 / 开始 /demo4/demo1.jsp

* 目录匹配 : 以 / 开始 以 * 结束. /* /demo1/*

* 扩展名匹配 : 不能以 / 开始 以 * 开始. *.do *.action

<servlet-name>的配置:根据Servlet的名称拦截Servlet.

<dispatcher>的配置:

* REQUEST :默认值.

* FORWARD :转发.

* INCLUDE :包含.

* ERROR :错误页面跳转.(全局错误页面)

2.3 代码实现:

【步骤一】 : 创建数据库和表:

```
[mw_shl_code=shell,true]create database web_16;
```

```
use web_16;
```

```
create table user(
```

```
    id int primary key auto_increment,
```

```
    username varchar(20),
```

```
    password varchar(20),
```

```
    nickname varchar(20),
```

```
    type varchar(10)
```

```
);[/mw_shl_code]
```

```
[mw_shl_code=java,true]insert into user values (null,'aaa','111','张凤','user');
```

```
insert into user values (null,'bbb','111','如花','user');
```

```
insert into user values (null,'ccc','111','张芙蓉','user');[/mw_shl_code]
```

【步骤二】 : 导入jar包和工具类:

【步骤三】 : 创建包结构及常用类:

```
└─ com.itheima
    └─ autologin
        └─ dao
        └─ domain
        └─ service
        └─ web
            └─ filter
            └─ servlet
```

【步骤四】 : 登录功能.

【步骤五】：利用COokie记住用户名和密码

【步骤六】：实现自动登录的过滤器

3. 案例二：通用的字符集编码的过滤器.

3.1 需求:

在一个网站上,通常会提交带有中文的数据,GET/POST请求都有可能提交中文数据.通常情况下在Servlet中处理中文乱码.现在能不能将乱码的处理交给过滤器完成.只需要在Servlet中关心参数的接收就可以了.

只需要在Servlet中调用request.getParameter();接收参数就可以,而不关心到底get/post如何处理乱码.

3.2 分析:

【增强request中的getParameter方法】

继承 :控制这个类构造.

装饰者模式 :增强的类和被增强类实现相同的接口,增强的类中获得到被增强的类的引用.

* 缺点: 接口中方法太多.

动态代理 :被增强的类实现接口就可以.

3.3 代码实现:

```
[mw_shl_code=java,true]public class MyHttpServletRequestWrapper extends HttpServletRequestWrapper{
    private HttpServletRequest request;
    public MyHttpServletRequestWrapper(HttpServletRequest request) {
        super(request);
        this.request = request;
    }
    @Override
    public String getParameter(String name) {
        // 根据请求方式不同,去处理:
        // 获得请求方式:
        String method = request.getMethod();
        if("get".equalsIgnoreCase(method)){
            String value = null;
            try {
```

```
        value = new String(request.getParameter(name).getBytes("ISO-8859-1"),"UTF-8")
;
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
    return value;
}else if("post".equalsIgnoreCase(method)){
    try {
        request.setCharacterEncoding("UTF-8");
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}
return super.getParameter(name);
}
}[/mw_shl_code]
```



回帖

— 3条回帖 —



lvshen9

差差差

收藏了，谢谢分享

沙发 • 2017-3-12 08:55:57

回帖



来自宇宙超级黑马专属安卓客户端



wangrongyu

好帖要顶

藤椅 • 2017-3-14 12:24:59

回帖



wllpeter

谢谢分享

板凳 • 2017-4-19 22:09:44

回帖

