Day25

java课程-李彦伯

Filter过滤器

> filter是对客户端访问资源的过滤，符合条件放行，不符合条件不放行，并且可以对目 标资源访问前后进行逻辑处理

设置步骤

- 创建类实现Filter接口

- 在doFilter方法中编写放行和拦截的代码
- 在web.xml文件中进行配置

```
<filter>
  <display-name>UserFilter</display-name>
  <filter-name>UserFilter</filter-name>
  <filter-class>com.zhiyou100.filter.UserFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>UserFilter</filter-name>
  <url-pattern>/*</url-pattern>
  <!-- <servlet-name>MyServlet01</servlet-name> -->
</filter-mapping>
```

- 对于 `<url-pattern>/*</url-pattern>` 将路径配置为 `/*` 就是拦截所有的请求
- 还可以配置成 `<servlet-name>MyServlet01</servlet-name>` 表示指拦截指定的servlet
- `url-pattern` 和 `servlet-name` 可以同时配置多个

```
<filter-mapping>
  <filter-name>UserFilter</filter-name>
  <servlet-name>MyServlet01</servlet-name>
  <servlet-name>MyServlet02</servlet-name>
  <url-pattern>*.do</url-pattern>
  <url-pattern>*.action</url-pattern>
</filter-mapping>
```

url-pattern

> 可以设置多种方式进行配置url-pattern

- 完全匹配 `/sertvle1`,可以使用 `<servlet-name>sertvle1</servlet-name>` 代替
- 目录匹配 `/aaa/bbb/*` 使用情况较多
  - `/user/*` 访问前台的资源进入此过滤器
  - `/admin/*` 访问后台的资源时执行此过滤器

- 扩展名匹配 `*.abc` `*.jsp` `*.action`
- `/*` 拦截所有,包括静态资源和jsp页面
- 目录匹配和扩展名匹配不能同时使用,会造成服务器无法启动
- **如果有多个filter,按照web.xml中的 `<filter-mapping>` 标签的前后顺序进行过滤,并不依照 `<filter>` 的前后顺序**

dispatcher：访问的方式(了解)

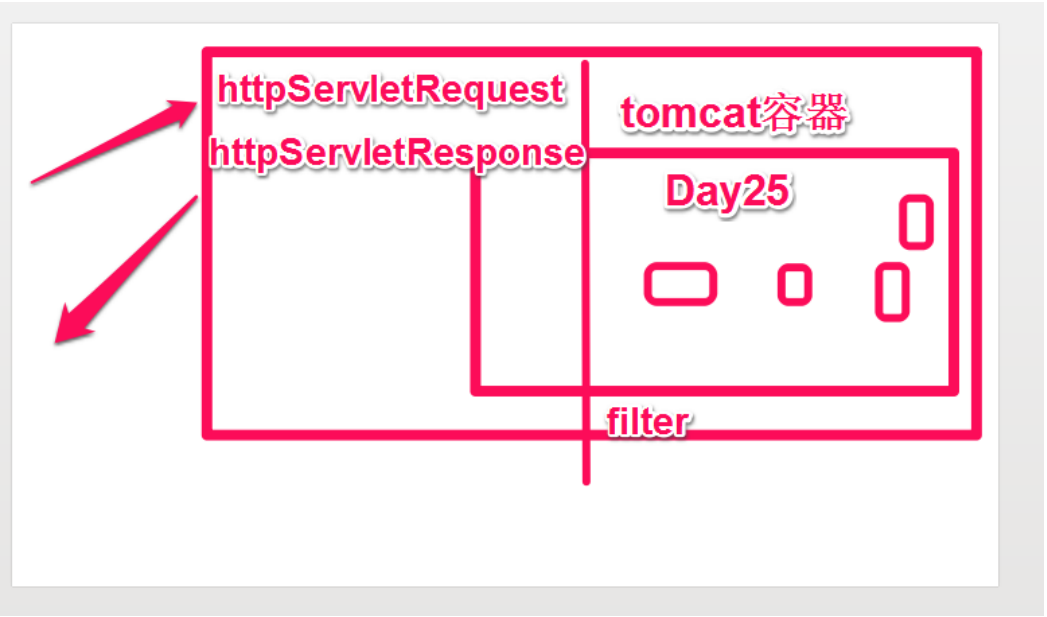
- REQUEST：默认值，代表直接访问某个资源时执行filter
- FORWARD：转发时才执行filter
- INCLUDE: 包含资源时执行filter
- ERROR：发生错误时 进行跳转是执行filter

生命周期

- 服务器启动的时候init方法调用,filter被创建
- 服务器关闭的时候destory方法调用,filter被销毁
- 当有请求过来并符合拦截规则的时候doFilter方法执行.

方法详解

> jdk1.8中支持接口中定义非抽象方法,使用修饰符default进行修饰,init和destory都是default修饰,所以我们实现接口的时候可以不用实现这两个方法



- init方法

```
@Override
public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("init");
        //获取过滤器的名字
        System.out.println(filterConfig.getFilterName());
    //获取过滤器的初始化参数       System.out.println(filterConfig.getInitPara
meter("exclude"));
    //获取servletContext
        System.out.println(filterConfig.getServletContext());
}
```

- doFilter方法
  - ServletRequest/ServletResponse：每次在执行doFilter方法时 web容器负责创建一个request和一个response对象作为doFilter的参数传递进来。该request个该response就是在访问目标资源的service方法时的request和response。
  - FilterChain：过滤器链对象，通过该对象的doFilter方法可以放行该请求

```
public void doFilter(ServletRequest request, ServletResponse response, Fil
terChain chain)
        throws IOException, ServletException {
    HttpServletRequest req = (HttpServletRequest) request;//实质上就是tomca
t组装的HttpServletRequest对象
    HttpServletResponse res = (HttpServletResponse) response;//实质上就是to
mcat组装的HttpServletResponse对象
    System.out.println(req.getServletPath());
    chain.doFilter(request, response);//用于放行
}
```

- destory方法
  - filter对象销毁时执行

```
public void destroy() {
        System.out.println("destory");
}
```

应用

权限限制

> 完成crm中用户不登陆,不能直接浏览应用中的内容

```
public void doFilter(ServletRequest request, ServletResponse response, Fil
terChain chain) throws IOException, ServletException {
HttpServletRequest req = (HttpServletRequest) request;
HttpServletResponse res = (HttpServletResponse) response;

User u = (User)req.getSession().getAttribute("user");
    String path = req.getServletPath();
    if(path.equals("/index.jsp")||path.equals("/login")){
        chain.doFilter(request, response);
        return;
    }
    if(u == null){
         res.sendRedirect(req.getContextPath()+"/index.jsp");
         return;
     }
        chain.doFilter(request, response);
}
```

全局编码

> 设置request和response的编码为UTF-8

```
public void doFilter(ServletRequest request, ServletResponse response, Fil
terChain chain) throws IOException, ServletException {
    req.setCharacterEncoding("UTF-8");
    res.setContentType("text/html; charset=UTF-8");
}
```

tomcat内置编码过滤器设置

```
 <filter>
        <filter-name>setCharacterEncodingFilter</filter-name>
        <filter-class>org.apache.catalina.filters.SetCharacterEncodingFilt
er</filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>setCharacterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

Listener监听器

> 监听器就是监听某个对象的的状态变化的组件,其主要是监听三个域对象(ServletContext域,HttpSession域,ServletRequest域)的创建和销毁以及域对象内的属性的改变,一共有6+2个

/	ServletContext域	HttpSession域	ServletRequest域
域对象的创建和销毁	ServletContextListener	HttpSessionListener	ServletRequestListener
域对象内的属性的改变	ServletContextAttributeListener	HttpSessionAttributeListener	ServletRequestAttributeListener

ServletContextListener

> 监听ServletContext域的创建和销毁

设置步骤

- 写一个监听器类去实现监听器接口
- 覆盖接口方法
- 在web.xml中进行配置

```
public class ServletContextListenerTest implements ServletContextListener
{

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("context创建");
    }
    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("context销毁");
    }
}
```

```
  <listener>
    <listener-class>com.zhiyou100.listener.ServletContextListenerTest</lis
tener-class>
  </listener>
```

应用范围

> 加载配置文件,spring中加载配置文件使用

HttpSessionListener

> 监听session的创建和销毁

```
public class HttpSessionListenerTest implements HttpSessionListener {

    @Override
    public void sessionCreated(HttpSessionEvent se) {
        System.out.println("session创建");
    }
    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        System.out.println("session销毁");
    }

}
```

```
  <listener>
    <listener-class>com.zhiyou100.listener.HttpSessionListenerTest</listen
er-class>
  </listener>
```

应用范围

> 统计网站的访问数量

HttpSessionListener

> 监听request的创建和销毁

```
public class ServletRequestListenerTest implements ServletRequestListener
{
    @Override
    public void requestInitialized(ServletRequestEvent sre) {
        System.out.println("请求发送");
    }
    @Override
    public void requestDestroyed(ServletRequestEvent sre) {
        System.out.println("请求销毁");
    }

}
```

```
   <listener>
    <listener-class>com.zhiyou100.listener.ServletRequestListenerTest</lis
tener-class>
  </listener>
```

ServletContextAttributeListener域中数据的变化

```java
public class ServletContextAttributeListenerTest implements ServletContext
AttributeListener {
    @Override
    public void attributeAdded(ServletContextAttributeEvent scae) {
        System.out.println(scae.getName());//获取放到域中的name
        System.out.println(scae.getValue());//获取放到域中的value
    }
    @Override
    public void attributeRemoved(ServletContextAttributeEvent scae) {
        System.out.println(scae.getName());//获取删除的name
        System.out.println(scae.getValue());//获取删除value
    }
    @Override
    public void attributeReplaced(ServletContextAttributeEvent scae) {
        System.out.println(scae.getName());//获取修改前name
        System.out.println(scae.getValue());//获取修改前value
    }
}
```

```xml
  <listener>
    <listener-class>com.zhiyou100.listener.ServletContextAttributeListener
Test</listener-class>
  </listener>
```

HttpSessionAttributeListener域中数据的变化(和上述相同)

ServletRequestAttributeListener域中数据的变化(和上述相同)

session中的绑定的对象相关的监听器(对象感知监听器)

> 对象放在session中一共有四种状态
>
> 1. 绑定状态：就一个对象被放到session域中
> 2. 解绑状态：就是这个对象从session域中移除了
> 3. 钝化状态：是将session内存中的对象持久化（序列化）到磁盘
> 4. 活化状态：就是将磁盘上的对象再次恢复到session内存中

绑定与解绑的监听器HttpSessionBindingListener

> 此接口是需要对象去实现,并且不需要去注册

```
public class User implements HttpSessionBindingListener{
        private  int age;
        private  String name;
         public int getAge() {
             return age;
         }
         public void setAge(int age) {
             this.age = age;
         }
         public String getName() {
             return name;
         }
         public void setName(String name) {
             this.name = name;
         }
         @Override
         public void valueBound(HttpSessionBindingEvent event) {
             System.out.println("绑定");
         }
         @Override
         public void valueUnbound(HttpSessionBindingEvent event) {
             System.out.println("解绑");
         }
}
```

钝化与活化的监听器HttpSessionActivationListener

> 实质上就是session中对象的序列化和反序列化,可用于服务器的优化,所以需要将放在session中的对象实现此接口,如果需要对象存在磁盘中,需要当前类实现Serializable接口

```
public class User implements HttpSessionActivationListener,Serializable{
        private  int age;
        private  String name;
         public int getAge() {
             return age;
         }
         public void setAge(int age) {
             this.age = age;
         }
         public String getName() {
             return name;
         }
         public void setName(String name) {
             this.name = name;
         }
         @Override
         public void sessionWillPassivate(HttpSessionEvent se) {
             System.out.println("---------------------------------------钝
化了");
         }
         @Override
         public void sessionDidActivate(HttpSessionEvent se) {
             System.out.println("---------------------------------------活
化了");
         }
         @Override
         public String toString() {
             return "User [age=" + age + ", name=" + name + "]";
         }
}
```

服务器的优化处理

> 通过一个xml文件进行配置session中的对象多久被钝化
>
> - 在META-INF中创建一个名字为context.xml文件
> - 将下面代码拷贝进文件

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
 <!-- maxIdleSwap:session中的对象多长时间不使用就钝化单位是分钟 -->
 <!-- directory:钝化后的对象的文件写到磁盘的哪个目录下  配置钝化的对象文件在work/catalina/localhost/钝化文件 -->
 <!--org.apache.catalina.session.PersistentManager的钝化引擎-->
 <!--org.apache.catalina.session.FileStore 用于存储文件-->
 <Manager className="org.apache.catalina.session.PersistentManager" maxIdleSwap="1">
  <Store className="org.apache.catalina.session.FileStore" directory="bigdata14" />
 </Manager>
</Context>
```