SpringBoot实现拦截器的方法

****springboot中实现拦截器的两种方式，分别为：实现HandlerInterceptor接口和使用servlet的filter拦截器。****

# 一.实现HandlerInterceptor

1. 创建一个拦截器实现HandlerInterceptor接口

|  |
| --- |
| import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import org.springframework.stereotype.Component;  import org.springframework.web.servlet.HandlerInterceptor;  import org.springframework.web.servlet.ModelAndView;  @Component  public class LoginInterceptor implements HandlerInterceptor {  /\*\*  \* 预处理回调方法，实现处理器的预处理  \* 返回值：true表示继续流程；false表示流程中断，不会继续调用其他的拦截器或处理器  \*/  @Override  public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)  throws Exception {  System.out.println("开始拦截.........");  //业务代码  return false;  }  /\*\*  \* 后处理回调方法，实现处理器（controller）的后处理，但在渲染视图之前  \* 此时我们可以通过modelAndView对模型数据进行处理或对视图进行处理  \*/  @Override  public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,  ModelAndView modelAndView) throws Exception {  // TODO Auto-generated method stub  }  /\*\*  \* 整个请求处理完毕回调方法，即在视图渲染完毕时回调，  \* 如性能监控中我们可以在此记录结束时间并输出消耗时间，  \* 还可以进行一些资源清理，类似于try-catch-finally中的finally，  \* 但仅调用处理器执行链中  \*/  @Override  public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)  throws Exception {  // TODO Auto-generated method stub  }  } |

2 新建配置类来管理拦截器，并将之前的拦截器注入其中

|  |
| --- |
| package wmq.fly.mybatis.interceptor;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.context.annotation.Configuration;  import org.springframework.web.servlet.config.annotation.InterceptorRegistry;  import org.springframework.web.servlet.config.annotation.WebMvcConfigurationSupport;  @Configuration  public class MvcInterceptorConfig extends WebMvcConfigurationSupport{  @Autowired  private LoginInterceptor loginInterceptor;  @Override  protected void addInterceptors(InterceptorRegistry registry) {  // 多个拦截器组成一个拦截器链  // addPathPatterns 用于添加拦截规则，/\*\*表示拦截所有请求  // excludePathPatterns 用户排除拦截  registry.addInterceptor(loginInterceptor).addPathPatterns("/\*\*")  .excludePathPatterns("/stuInfo/getAllStuInfoA","/account/register");  super.addInterceptors(registry);  }  } |

# 二.使用servlet的filter拦截器

|  |
| --- |
| package wmq.fly.mybatis.filter;  import java.io.IOException;  import java.util.Arrays;  import javax.servlet.Filter;  import javax.servlet.FilterChain;  import javax.servlet.FilterConfig;  import javax.servlet.ServletException;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  import javax.servlet.annotation.WebFilter;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import org.springframework.stereotype.Component;  @Component  @WebFilter(urlPatterns="/\*\*",filterName="loginFilter")  public class LoginFilter implements Filter{  //排除不拦截的url  private static final String[] excludePathPatterns = { "/stuInfo/getAllStuInfoA"};  @Override  public void init(FilterConfig filterConfig) throws ServletException {  // TODO Auto-generated method stub  }  @Override  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)  throws IOException, ServletException {  HttpServletRequest req = (HttpServletRequest)request;  HttpServletResponse res = (HttpServletResponse)response;  // 获取请求url地址，不拦截excludePathPatterns中的url  String url = req.getRequestURI();  if (Arrays.asList(excludePathPatterns).contains(url)) {  //放行，相当于第一种方法中LoginInterceptor返回值为true  chain.doFilter(request, response);  }  System.out.println("开始拦截了................");  //业务代码  }  @Override  public void destroy() {  // TODO Auto-generated method stub  }  } |

# 通过拦截器实现登录功能

## **一，pom.xml的配置**

    这里很简单，先引入spring-boot-starter-parent，parent 是父模块，由父模块统一进行 spring-boot 版本管理，dependencies 中与 spring-boot 启动绑定的包不需要再指定版本。

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.3.RELEASE</version>  </parent>    <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies> |

二，新建WebConfigurer

    1，如图，新建的 config 包，用来装初始化文件，在配置之下新建 WebConfigurer。

    2，WebConfigurer需要实现 WebMvcConfigurer 这个接口，并实现里面的两个方法。（在老版本的 spring-boot 中使用的是WebMvcConfigurerAdapter，新版本中已过时！！！还有不能通过继承 WebMvcConfigurationSupport 这个类来实现，这样会在某些情况下失效！！！），第二个 addInterceptors 方法用来注册添加拦截器。

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.InterceptorRegistry;

import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration

public class WebConfigurer implements WebMvcConfigurer {

// 这个方法是用来配置静态资源的，比如html，js，css，等等

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

}

// 这个方法用来注册拦截器，我们自己写好的拦截器需要通过这里添加注册才能生效

@Override

public void addInterceptors(InterceptorRegistry registry) {

}

}

三，新建登陆拦截器

    如下图所示，在 config 包下新建一个 intercepors 包，用来装拦截器。然后在拦截器下新建 LoginInterceptor，用来拦截验证登陆。

    2，每一个拦截器有需要实现的 HandlerInterceptor 接口，这个接口有三个方法，每个方法会在请求调用的不同时期完成，因为我们需要在接口调用之前拦截请求判断是否登陆，所以这里需要使用 preHandle 方法，在里面写验证逻辑，最后返回 true 或者 false，确定请求是否合法。记住加 @Component 注解，我们需要在上一步的 WebConfigurer 类中注入。用户是自定义的用户类，大家可以自己定义，我这里就不贴出来了。

import com.impte.study.domain.po.User;

import org.springframework.lang.Nullable;

import org.springframework.stereotype.Component;

import org.springframework.web.servlet.HandlerInterceptor;

import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

@Component

public class LoginInterceptor implements HandlerInterceptor {

//这个方法是在访问接口之前执行的，我们只需要在这里写验证登陆状态的业务逻辑，就可以在用户调用指定接口之前验证登陆状态了

public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {

//每一个项目对于登陆的实现逻辑都有所区别，我这里使用最简单的Session提取User来验证登陆。

HttpSession session = request.getSession();

//这里的User是登陆时放入session的

User user = (User) session.getAttribute("user");

//如果session中没有user，表示没登陆

if (user == null){

//这个方法返回false表示忽略当前请求，如果一个用户调用了需要登陆才能使用的接口，如果他没有登陆这里会直接忽略掉

//当然你可以利用response给用户返回一些提示信息，告诉他没登陆

return false;

}else {

return true; //如果session里有user，表示该用户已经登陆，放行，用户即可继续调用自己需要的接口

}

}

public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, @Nullable ModelAndView modelAndView) throws Exception {

}

public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, @Nullable Exception ex) throws Exception {

}

}

四，在WebConfigurer中添加拦截器

    1，首先将 LoginInterceptor 注入到 WebConfigurer 中。

@Autowired

private LoginInterceptor loginInterceptor;

    2，然后在 WebConfigurer 中的 addInterceptors 中添加拦截器，使其生效。

@Override

public void addInterceptors(InterceptorRegistry registry) {

// addPathPatterns("/\*\*") 表示拦截所有的请求，

// excludePathPatterns("/login", "/register") 表示除了登陆与注册之外，因为登陆注册不需要登陆也可以访问

registry.addInterceptor(loginInterceptor).addPathPatterns("/\*\*").excludePathPatterns("/login", "/register");

super.addInterceptors(registry); //较新Spring Boot的版本中这里可以直接去掉，否则会报错

}

    3，addPathPatterns 用来设置拦截路径，excludePathPatterns 用来设置白名单，也就是不需要触发这个拦截器的路径。

    完整代码：

import com.impte.study.config.interceptors.LoginInterceptor;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.InterceptorRegistry;

import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration

public class WebConfigurer implements WebMvcConfigurer {

@Autowired

private LoginInterceptor loginInterceptor;

// 这个方法是用来配置静态资源的，比如html，js，css，等等

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

}

// 这个方法用来注册拦截器，我们自己写好的拦截器需要通过这里添加注册才能生效

@Override

public void addInterceptors(InterceptorRegistry registry) {

// addPathPatterns("/\*\*") 表示拦截所有的请求，

// excludePathPatterns("/login", "/register") 表示除了登陆与注册之外，因为登陆注册不需要登陆也可以访问

registry.addInterceptor(loginInterceptor).addPathPatterns("/\*\*").excludePathPatterns("/login", "/register");

}

}

五，总结

    我在这里只是用登陆的例子来展现来展现拦截器的基本使用，拦截器用途很广，比如可以用来进行接口权限控制。如果我的文章对你有帮助，请关注一波，CSDN新人驾到，还望各路大神多多照顾！