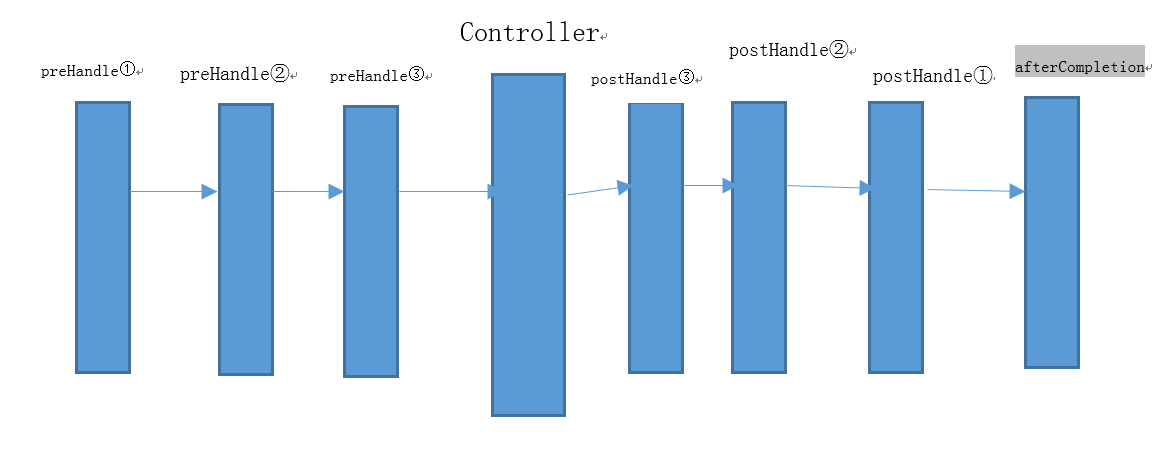
[SpringMVC 拦截器实现原理和登录实现](https://www.cnblogs.com/jianjianyang/p/5009457.html)



**springMVC拦截器的实现一般有两种方式**

     第一种方式是要定义的Interceptor类要实现了Spring的HandlerInterceptor 接口

     第二种方式是继承实现了HandlerInterceptor接口的类，比如Spring已经提供的实现了HandlerInterceptor接口的抽象类HandlerInterceptorAdapter

HandlerInterceptor 接口中定义了三个方法，我们就是通过这三个方法来对用户的请求进行拦截处理的。

**preHandle()**： 这个方法在业务处理器处理请求之前被调用，SpringMVC 中的Interceptor 是链式的调用的，在一个应用中或者说是在一个请求中可以同时存在多个Interceptor 。每个Interceptor 的调用会依据它的声明顺序依次执行，而且最先执行的都是Interceptor 中的preHandle 方法，所以可以在这个方法中进行一些前置初始化操作或者是对当前请求的一个预处理，也可以在这个方法中进行一些判断来决定请求是否要继续进行下去。该方法的返回值是布尔值Boolean 类型的，当它返回为false 时，表示请求结束，后续的Interceptor 和Controller 都不会再执行；当返回值为true 时就会继续调用下一个Interceptor 的preHandle 方法，如果已经是最后一个Interceptor 的时候就会是调用当前请求的Controller 方法。

**postHandle()**：这个方法在当前请求进行处理之后，也就是Controller 方法调用之后执行，但是它会在DispatcherServlet 进行视图返回渲染之前被调用，所以我们可以在这个方法中对Controller 处理之后的ModelAndView 对象进行操作。postHandle 方法被调用的方向跟preHandle 是相反的，也就是说先声明的Interceptor 的postHandle 方法反而会后执行。

**afterCompletion()**：该方法也是需要当前对应的Interceptor 的preHandle 方法的返回值为true 时才会执行。顾名思义，该方法将在整个请求结束之后，也就是在DispatcherServlet 渲染了对应的视图之后执行。这个方法的主要作用是用于进行资源清理工作的。

下面来看我们的Interceptor类

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.web.servlet.ModelAndView;

import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

public class CommonInterceptor extends HandlerInterceptorAdapter{

private final Logger log = LoggerFactory.getLogger(CommonInterceptor.class);

public static final String LAST\_PAGE = "lastPage";

/\*\*

\* 在业务处理器处理请求之前被调用

\* 如果返回false

\* 从当前的拦截器往回执行所有拦截器的afterCompletion(),再退出拦截器链

\*

\* 如果返回true

\* 执行下一个拦截器,直到所有的拦截器都执行完毕

\* 再执行被拦截的Controller

\* 然后进入拦截器链,

\* 从最后一个拦截器往回执行所有的postHandle()

\* 接着再从最后一个拦截器往回执行所有的afterCompletion()

\*/

@Override

public boolean preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) throws Exception {

if ("GET".equalsIgnoreCase(request.getMethod())) {

RequestUtil.saveRequest();

}

log.info("==============执行顺序: 1、preHandle================");

String requestUri = request.getRequestURI();

String contextPath = request.getContextPath();

String url = requestUri.substring(contextPath.length()); if ("/userController/login".equals(url)) {

return true;

}else {

String username = (String)request.getSession().getAttribute("user");

if(username == null){

log.info("Interceptor：跳转到login页面！");

request.getRequestDispatcher("/page/index.jsp").forward(request, response);

return false;

}else

return true;

}

}

/\*\*

\* 在业务处理器处理请求执行完成后,生成视图之前执行的动作

\* 可在modelAndView中加入数据，比如当前时间

\*/

@Override

public void postHandle(HttpServletRequest request,

HttpServletResponse response, Object handler,

ModelAndView modelAndView) throws Exception {

log.info("==============执行顺序: 2、postHandle================");

if(modelAndView != null){ //加入当前时间

modelAndView.addObject("haha", "测试postHandle");

}

}

/\*\*

\* 在DispatcherServlet完全处理完请求后被调用,可用于清理资源等

\* 当有拦截器抛出异常时,会从当前拦截器往回执行所有的拦截器的afterCompletion()

\*/

@Override

public void afterCompletion(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex)

throws Exception {

log.info("==============执行顺序: 3、afterCompletion================");

}

}

spring-MVC.xml的相关配置

<!-- 对静态资源文件的访问-->

<!-- <mvc:resources mapping="/images/\*\*" location="/images/"/>

<mvc:resources mapping="/css/\*\*" location="/css/" />

<mvc:resources mapping="/js/\*\*" location="/js/" />

<mvc:resources mapping="/favicon.ico" location="favicon.ico" /> -->

<!--配置拦截器, 多个拦截器,顺序执行 -->

<mvc:interceptors>

<mvc:interceptor>

<!--

/\*\*的意思是所有文件夹及里面的子文件夹

/\*是所有文件夹，不含子文件夹

/是web项目的根目录

-->

**<mvc:mapping path="/\*\*" />**

<!-- 需排除拦截的地址 -->

<!-- <mvc:exclude-mapping path="/userController/login"/> -->

**<bean id="commonInterceptor" class="org.shop.interceptor.CommonInterceptor"></bean>** <!--这个类就是我们自定义的Interceptor -->

</mvc:interceptor>

<!-- 当设置多个拦截器时，先按顺序调用preHandle方法，然后逆序调用每个拦截器的postHandle和afterCompletion方法 -->

</mvc:interceptors>

就这么简单SpringMVC拦截器写好了，**登陆的实现用上一篇filter的代码就可以进行测试了。**

注意：在我测试的时候我用<mvc:resources>不拦截静态资源居然不管用，也不知道是怎么回事，希望有大神指正下应该怎么做

<!-- 对静态资源文件的访问-->  
<mvc:resources mapping="/images/\*\*" location="/images/"/>

<mvc:resources mapping="/css/\*\*" location="/css/" />

<mvc:resources mapping="/js/\*\*" location="/js/" />

<mvc:resources mapping="/favicon.ico" location="favicon.ico" />

所以我只好在web.xml进行了对静态资源不拦截的配置

<!-- 不拦截静态文件 -->

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>/js/\*</url-pattern>

<url-pattern>/css/\*</url-pattern>

<url-pattern>/images/\*</url-pattern>

<url-pattern>/fonts/\*</url-pattern>

</servlet-mapping>