**5.1、处理器拦截器简介HandlerInterceptor**

Spring Web MVC的处理器拦截器类似于Servlet开发中的过滤器Filter，用于对处理器进行预处理和后处理。

提示：推荐能使用servlet规范中的过滤器Filter实现的功能就用Filter实现，因为HandlerInteceptor只有在Spring Web MVC环境下才能使用，因此Filter是最通用的、最先应该使用的。如登录这种拦截器最好使用Filter来实现。

**5.1.1、常见应用场景**

1、日志记录：记录请求信息的日志，以便进行信息监控、信息统计、计算PV（Page View）等。

2、权限检查：如登录检测，进入处理器检测检测是否登录，如果没有直接返回到登录页面；

3、性能监控：有时候系统在某段时间莫名其妙的慢，可以通过拦截器在进入处理器之前记录开始时间，在处理完后记录结束时间，从而得到该请求的处理时间（如果有反向代理，如apache可以自动记录）；

4、**通用行为**：读取cookie得到用户信息并将用户对象放入请求，从而方便后续流程使用，还有如提取Locale、Theme信息等，只要是**多个处理器都需要的**即可使用拦截器实现。

5、**OpenSessionInView**：如Hibernate，在进入处理器打开Session，在完成后关闭Session。

…………本质也是AOP（面向切面编程），也就是说符合横切关注点的所有功能都可以放入拦截器实现。

**5.1.2、拦截器接口**

|  |
| --- |
| **public** **interface** HandlerInterceptor {  /\*\*  \* 预处理回调方法，实现处理器的预处理（如登录检查）  \* **@param** handler 响应的处理器Controller  \* **@return** true表示继续流程（如调用下一个拦截器或处理器）。  \* false表示流程中断（如登录检查失败），不会继续调用其他的拦截器或处理器，**此时我们需要通过response来产生响应**；  \*/  **boolean** preHandle(HttpServletRequest request, HttpServletResponse response,  Object handler) **throws** Exception;  /\*\*  \* 后处理回调方法，实现处理器的后处理（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，modelAndView也可能为null。  \* **@param** handler  \* **@param** modelAndView  \*/  **void** postHandle(HttpServletRequest request, HttpServletResponse response,  Object handler, ModelAndView modelAndView) **throws** Exception;  /\*\*  \* 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preHandle返回true  \* 的拦截器的afterCompletion  \* **@param** handler  \* **@param** ex  \*/  **void** afterCompletion(HttpServletRequest request,  HttpServletResponse response, Object handler, Exception ex)  **throws** Exception;  } |

**5.1.3、拦截器适配器**

有时候我们可能只需要实现三个回调方法中的某一个，如果实现HandlerInterceptor接口的话，三个方法必须实现，此时spring提供了一个HandlerInterceptorAdapter适配器（一种适配器设计模式的实现），允许我们只实现需要的回调方法。

**5.1.4、运行流程图**

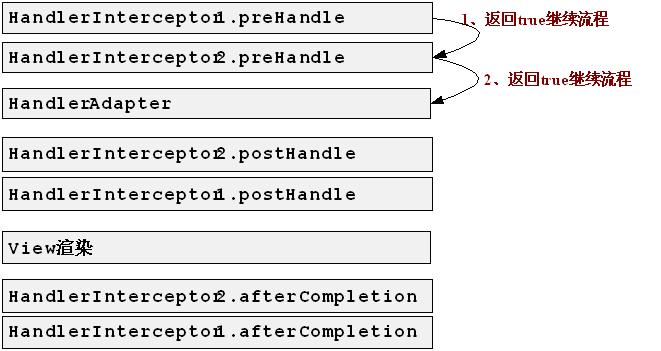


图5-1 正常流程

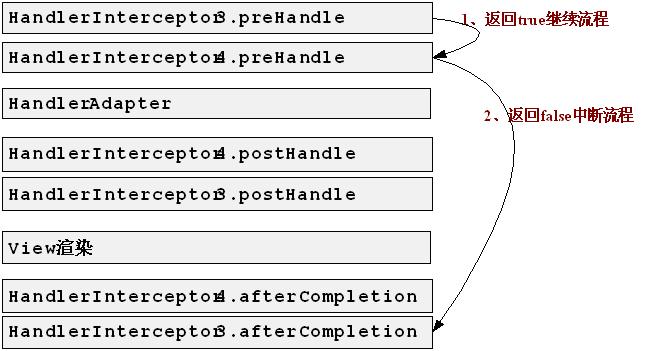


图5-2 中断流程

中断流程中，比如是HandlerInterceptor2中断的流程（preHandle返回false），此处仅调用它之前拦截器的preHandle返回true的afterCompletion方法。

接下来看一下DispatcherServlet内部到底是如何工作的吧：

**Java代码  [收藏代码](javascript:void())**

1. //doDispatch方法
2. //1、处理器拦截器的预处理（正序）
3. HandlerInterceptor[] interceptors = mappedHandler.getInterceptors();
4. **if** (interceptors != **null**) {
5. **for** (**int** i = 0; i < interceptors.length; i++) {
6. HandlerInterceptor interceptor = interceptors[i];
7. **if** (!interceptor.preHandle(processedRequest, response, mappedHandler.getHandler())) {
8. //1.1、失败时触发afterCompletion的调用
9. triggerAfterCompletion(mappedHandler, interceptorIndex, processedRequest, response, **null**);
10. **return**;
11. }
12. interceptorIndex = i;//1.2、记录当前预处理成功的索引
13. }
14. }
15. //2、处理器适配器调用我们的处理器
16. mv = ha.handle(processedRequest, response, mappedHandler.getHandler());
17. //当我们返回null或没有返回逻辑视图名时的默认视图名翻译（详解4.15.5 RequestToViewNameTranslator）
18. **if** (mv != **null** && !mv.hasView()) {
19. mv.setViewName(getDefaultViewName(request));
20. }
21. //3、处理器拦截器的后处理（逆序）
22. **if** (interceptors != **null**) {
23. **for** (**int** i = interceptors.length - 1; i >= 0; i--) {
24. HandlerInterceptor interceptor = interceptors[i];
25. interceptor.postHandle(processedRequest, response, mappedHandler.getHandler(), mv);
26. }
27. }
28. //4、视图的渲染
29. **if** (mv != **null** && !mv.wasCleared()) {
30. render(mv, processedRequest, response);
31. **if** (errorView) {
32. WebUtils.clearErrorRequestAttributes(request);
33. }
34. //5、触发整个请求处理完毕回调方法afterCompletion （逆序）
35. triggerAfterCompletion(mappedHandler, interceptorIndex, processedRequest, response, **null**);

**（3、Spring配置文件chapter5-servlet.xml**

**java代码：**

**Java代码  [收藏代码](javascript:void())**

1. <bean name="/test" **class**="cn.javass.chapter5.web.controller.TestController"/>
2. <bean id="handlerInterceptor1"
3. **class**="cn.javass.chapter5.web.interceptor.HandlerInterceptor1"/>
4. <bean id="handlerInterceptor2"
5. **class**="cn.javass.chapter5.web.interceptor.HandlerInterceptor2"/>

**java代码：**

**Java代码  [收藏代码](javascript:void())**

1. <bean **class**="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping">
2. <property name="interceptors">
3. <list>
4. <ref bean="handlerInterceptor1"/>
5. <ref bean="handlerInterceptor2"/>
6. </list>
7. </property>
8. </bean>

interceptors：指定拦截器链，拦截器的执行顺序就是此处添加拦截器的顺序；

**5.3、应用**

**5.3.1、性能监控**

需要记录一下请求的处理时间，得到一些慢请求（如处理时间超过500毫秒），从而进行性能改进。

一般的反向代理服务器如apache都具有这个功能，但此处我们演示一下使用拦截器怎么实现。

|  |
| --- |
| 问题： 拦截器是单例，因此不管用户请求多少次都只有一个拦截器实现，即线程不安全，那我们应该怎么记录时间呢？  解决方案： ThreadLocal |
| // 作为拦截器链的第一个，这样得到的时间才是比较准确的  **public** **class** StopWatchHandlerInterceptor **extends** HandlerInterceptorAdapter {  // Spring提供的一个命名的ThreadLocal实现  **private** NamedThreadLocal<Long> startTimeThreadLocal = **new** NamedThreadLocal<Long>("StopWatch-StartTime");  @Override  **public** **boolean** preHandle(HttpServletRequest request,  HttpServletResponse response, Object handler) **throws** Exception {  startTimeThreadLocal.set(System.*currentTimeMillis*());  **return** **true**; // 继续流程  }  @Override  **public** **void** afterCompletion(HttpServletRequest request,  HttpServletResponse response, Object handler, Exception ex)  **throws** Exception {  **long** beginTime = startTimeThreadLocal.get();// 开始时间  **long** endTime = System.*currentTimeMillis*(); // 结束时间  **long** consumeTime = endTime - beginTime; // 消耗的时间  **if** (consumeTime > 500) {// 此处认为处理时间超过500毫秒的请求为慢请求  // **TODO** 记录到日志文件  System.***out***.println(String.*format*("%s consume %d millis",  request.getRequestURI(), consumeTime));  }  }  } |

**5.3.2、登录检测**

|  |
| --- |
| **public** **boolean** preHandle(HttpServletRequest request,  HttpServletResponse response, Object handler) **throws** Exception {  **final** String LOGIN\_URL = "登录页面url";  // 1、请求到登录页面 放行  **if** (request.getServletPath().startsWith(LOGIN\_URL)) {  **return** **true**;  }  // 2、**TODO** 比如退出、首页等页面无需登录， 允许游客的请求  // 3、如果用户已经登录 放行  **if** (request.getSession().getAttribute("username") != **null**) {  // 更好的实现方式的使用cookie  **return** **true**;  }  // 4、非法请求 即这些请求需要登录后才能访问,重定向到登录页面  response.sendRedirect(request.getContextPath() + LOGIN\_URL);  **return** **false**;  } |