## 5.1、处理器拦截器简介

Spring Web MVC的处理器拦截器（如无特殊说明，下文所说的拦截器即处理器拦截器）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。

### 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

**[java]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. **package** org.springframework.web.servlet;
2. **public** **interface** HandlerInterceptor {
3. **boolean** preHandle(
4. HttpServletRequest request, HttpServletResponse response,
5. Object handler)
6. **throws** Exception;
8. **void** postHandle(
9. HttpServletRequest request, HttpServletResponse response,
10. Object handler, ModelAndView modelAndView)
11. **throws** Exception;
13. **void** afterCompletion(
14. HttpServletRequest request, HttpServletResponse response,
15. Object handler, Exception ex)
16. **throws** Exception;
17. }

我们可能注意到拦截器一个有3个回调方法，而一般的过滤器Filter才两个，这是怎么回事呢？马上分析。

preHandle：预处理回调方法，实现处理器的预处理（如登录检查），第三个参数为响应的处理器（如我们上一章的Controller实现）；

     返回值：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；

postHandle：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（模型和视图对象）对模型数据进行处理或对视图进行处理，modelAndView也可能为null。

afterCompletion：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。

### 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，不管你需不需要，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。

**[java]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. **public** **abstract** **class** HandlerInterceptorAdapter **implements** HandlerInterceptor {
2. //省略代码 此处所以三个回调方法都是空实现，preHandle返回true。
3. }

### 5.1.4、运行流程图

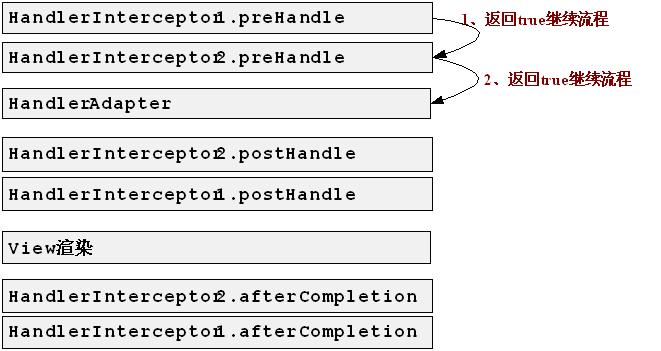


图5-1 正常流程

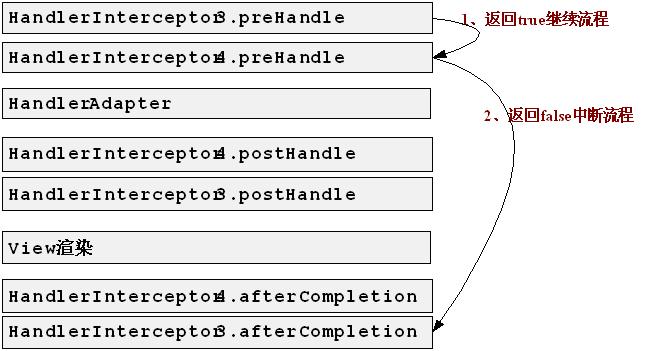


图5-2 中断流程

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]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. //doDispatch方法
2. //1、处理器拦截器的预处理（正序执行）
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
35. triggerAfterCompletion(mappedHandler, interceptorIndex, processedRequest, response, **null**);

注：以上是流程的简化代码，中间省略了部分代码，不完整。

**[java]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. // triggerAfterCompletion方法
2. **private** **void** triggerAfterCompletion(HandlerExecutionChain mappedHandler, **int** interceptorIndex,
3. HttpServletRequest request, HttpServletResponse response, Exception ex) **throws** Exception {
4. // 5、触发整个请求处理完毕回调方法afterCompletion （逆序从1.2中的预处理成功的索引处的拦截器执行）
5. **if** (mappedHandler != **null**) {
6. HandlerInterceptor[] interceptors = mappedHandler.getInterceptors();
7. **if** (interceptors != **null**) {
8. **for** (**int** i = interceptorIndex; i >= 0; i--) {
9. HandlerInterceptor interceptor = interceptors[i];
10. **try** {
11. interceptor.afterCompletion(request, response, mappedHandler.getHandler(), ex);
12. }
13. **catch** (Throwable ex2) {
14. logger.error("HandlerInterceptor.afterCompletion threw exception", ex2);
15. }
16. }
17. }
18. }
19. }

## 5.2、入门

具体内容详见工程springmvc-chapter5。

### 5.2.1、正常流程

（1、拦截器实现

**[java]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. **package** cn.javass.chapter5.web.interceptor;
2. //省略import
3. **public** **class** HandlerInterceptor1 **extends** HandlerInterceptorAdapter {//此处一般继承HandlerInterceptorAdapter适配器即可
4. @Override
5. **public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) **throws** Exception {
6. System.out.println("===========HandlerInterceptor1 preHandle");
7. **return** **true**;
8. }
9. @Override
10. **public** **void** postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) **throws** Exception {
11. System.out.println("===========HandlerInterceptor1 postHandle");
12. }
13. @Override
14. **public** **void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) **throws** Exception {
15. System.out.println("===========HandlerInterceptor1 afterCompletion");
16. }
17. }

以上是HandlerInterceptor1实现，HandlerInterceptor2同理 只是输出内容为“HandlerInterceptor2”。

（2、控制器

**[java]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. **package** cn.javass.chapter5.web.controller;
2. //省略import
3. **public** **class** TestController **implements** Controller {
4. @Override
5. **public** ModelAndView handleRequest(HttpServletRequest req, HttpServletResponse resp) **throws** Exception {
6. System.out.println("===========TestController");
7. **return** **new** ModelAndView("test");
8. }
9. }

（3、Spring配置文件chapter5-servlet.xml

**[html]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

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"**/>**

**[html]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

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；

（4、视图页面WEB-INF/jsp/test.jsp

**[plain]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
2. <%System.out.println("==========test.jsp");%>
3. test page

在控制台输出 test.jsp

（5、启动服务器测试

输入网址：<http://localhost:9080/springmvc-chapter5/test>

控制台输出：

**[plain]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. ===========HandlerInterceptor1 preHandle
2. ===========HandlerInterceptor2 preHandle
3. ===========TestController
4. ===========HandlerInterceptor2 postHandle
5. ===========HandlerInterceptor1 postHandle
6. ==========test.jsp
7. ===========HandlerInterceptor2 afterCompletion
8. ===========HandlerInterceptor1 afterCompletion

到此一个正常流程的演示完毕。和图5-1一样，接下来看一下中断的流程。

### 5.2.2、中断流程

（1、拦截器

HandlerInterceptor3和HandlerInterceptor4 与 之前的 HandlerInteceptor1和HandlerInterceptor2一样，只是在HandlerInterceptor4的preHandle方法返回false：

**[java]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. @Override
2. **public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) **throws** Exception {
3. System.out.println("===========HandlerInterceptor1 preHandle");
4. onse.getWriter().print("break");//流程中断的话需要我们进行响应的处理
5. **return** **false**;//返回false表示流程中断

（2、控制器

流程中断不会执行到控制器，使用之前的TestController控制器。

（3、Spring配置文件chapter5-servlet.xml

**[html]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. **<bean** id="handlerInterceptor3"
2. class="cn.javass.chapter5.web.interceptor.HandlerInterceptor3"**/>**
3. **<bean** id="handlerInterceptor4"
4. class="cn.javass.chapter5.web.interceptor.HandlerInterceptor4"**/>**

**[html]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. **<bean** id="handlerInterceptor3"
2. class="cn.javass.chapter5.web.interceptor.HandlerInterceptor3"**/>**
3. **<bean** id="handlerInterceptor4"
4. class="cn.javass.chapter5.web.interceptor.HandlerInterceptor4"**/>**

**[html]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. **<bean** class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"**>**
2. **<property** name="interceptors"**>**
3. **<list>**
4. **<ref** bean="handlerInterceptor3"**/>**
5. **<ref** bean="handlerInterceptor4"**/>**
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；

（4、视图页面

流程中断，不会执行到视图渲染。

（5、启动服务器测试

输入网址：<http://localhost:9080/springmvc-chapter5/test>

控制台输出：

**[plain]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. ===========HandlerInterceptor3 preHandle
2. ===========HandlerInterceptor4 preHandle
3. ===========HandlerInterceptor3 afterCompletion

此处我们可以看到只有HandlerInterceptor3的afterCompletion执行，否和图5-2的中断流程。

而且页面上会显示我们在HandlerInterceptor4 preHandle 直接写出的响应“break”。

## 5.3、应用

### 5.3.1、性能监控

如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。

实现分析：

1、在进入处理器之前记录开始时间，即在拦截器的preHandle记录开始时间；

2、在结束请求处理之后记录结束时间，即在拦截器的afterCompletion记录结束实现，并用结束时间-开始时间得到这次请求的处理时间。

问题：

我们的拦截器是单例，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？

解决方案是使用ThreadLocal，它是线程绑定的变量，提供线程局部变量（一个线程一个ThreadLocal，A线程的ThreadLocal只能看到A线程的ThreadLocal，不能看到B线程的ThreadLocal）。

代码实现：

**[java]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. **package** cn.javass.chapter5.web.interceptor;
2. **public** **class** StopWatchHandlerInterceptor **extends** HandlerInterceptorAdapter {
3. **private** NamedThreadLocal<Long>  startTimeThreadLocal =
4. **new** NamedThreadLocal<Long>("StopWatch-StartTime");
5. @Override
6. **public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response,
7. Object handler) **throws** Exception {
8. **long** beginTime = System.currentTimeMillis();//1、开始时间
9. startTimeThreadLocal.set(beginTime);//线程绑定变量（该数据只有当前请求的线程可见）
10. **return** **true**;//继续流程
11. }
13. @Override
14. **public** **void** afterCompletion(HttpServletRequest request, HttpServletResponse response,
15. Object handler, Exception ex) **throws** Exception {
16. **long** endTime = System.currentTimeMillis();//2、结束时间
17. **long** beginTime = startTimeThreadLocal.get();//得到线程绑定的局部变量（开始时间）
18. **long** consumeTime = endTime - beginTime;//3、消耗的时间
19. **if**(consumeTime > 500) {//此处认为处理时间超过500毫秒的请求为慢请求
20. //TODO 记录到日志文件
21. System.out.println(
22. String.format("%s consume %d millis", request.getRequestURI(), consumeTime));
23. }
24. }
25. }

NamedThreadLocal：Spring提供的一个命名的ThreadLocal实现。

在测试时需要把stopWatchHandlerInterceptor放在拦截器链的第一个，这样得到的时间才是比较准确的。

### 5.3.2、登录检测

在访问某些资源时（如订单页面），需要用户登录后才能查看，因此需要进行登录检测。

流程：

1、访问需要登录的资源时，由拦截器重定向到登录页面；

2、如果访问的是登录页面，拦截器不应该拦截；

3、用户登录成功后，往cookie/session添加登录成功的标识（如用户编号）；

4、下次请求时，拦截器通过判断cookie/session中是否有该标识来决定继续流程还是到登录页面；

5、在此拦截器还应该允许游客访问的资源。

拦截器代码如下所示：

**[java]** [view plain](http://blog.csdn.net/hongjun1847/article/details/20550341) [copy](http://blog.csdn.net/hongjun1847/article/details/20550341)

[在CODE上查看代码片](https://code.csdn.net/snippets/218760)

1. @Override
2. **public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response,
3. Object handler) **throws** Exception {
4. //1、请求到登录页面 放行
5. **if**(request.getServletPath().startsWith(loginUrl)) {
6. **return** **true**;
7. }
9. //2、TODO 比如退出、首页等页面无需登录，即此处要放行 允许游客的请求
11. //3、如果用户已经登录 放行
12. **if**(request.getSession().getAttribute("username") != **null**) {
13. //更好的实现方式的使用cookie
14. **return** **true**;
15. }
17. //4、非法请求 即这些请求需要登录后才能访问
18. //重定向到登录页面
19. response.sendRedirect(request.getContextPath() + loginUrl);
20. **return** **false**;
21. }

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。