

63 | 职责链模式（下）：框架中常用的过滤器、拦截器是如何实现的？

2020-03-27 王争

设计模式之美

[进入课程 >](#)

职责链模式（下）

讲述：冯永吉

时长 08:25 大小 7.72M



上一节课，我们学习职责链模式的原理与实现，并且通过一个敏感词过滤框架的例子，展示了职责链模式的设计意图。本质上来说，它跟大部分设计模式一样，都是为了解耦代码，应对代码的复杂性，让代码满足开闭原则，提高代码的可扩展性。

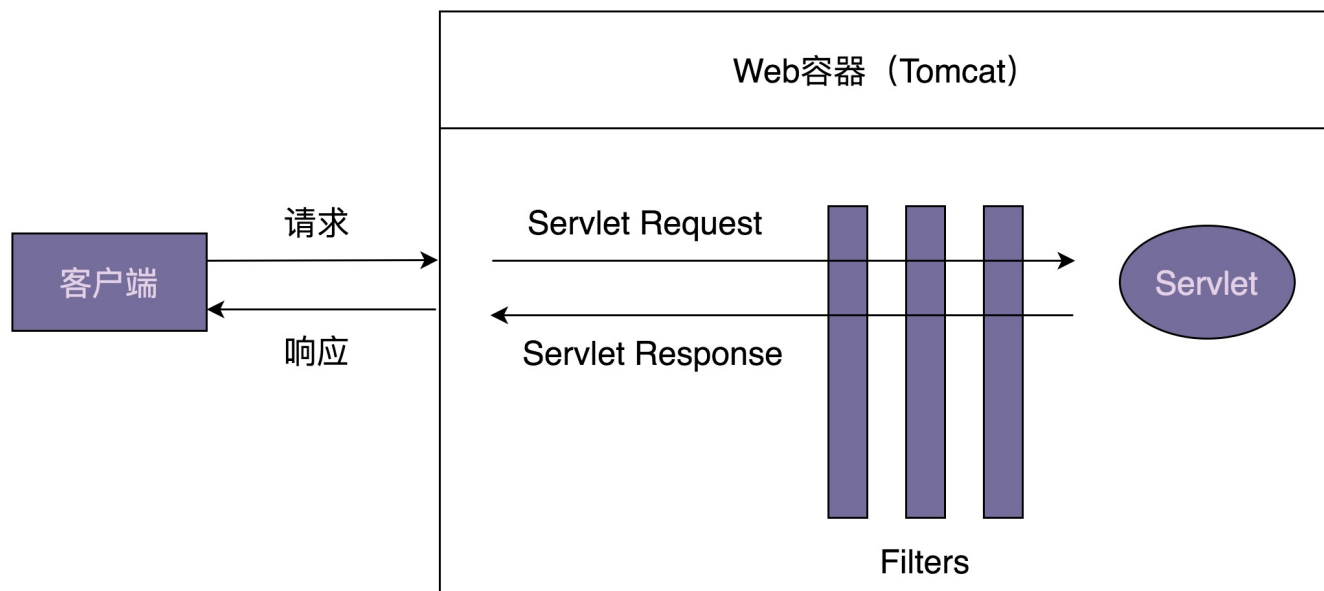
除此之外，我们还提到，职责链模式常用在框架的开发中，为框架提供扩展点，让框架的使用者在不修改框架源码的情况下，基于扩展点添加新的功能。实际上，更具体点来说，职责链模式最常用来开发框架的过滤器和拦截器。今天，我们就通过 Servlet Filter、Spring Interceptor 这两个 Java 开发中常用的组件，来具体讲讲它在框架开发中的应用。



话不多说，让我们正式开始今天的学习吧！

Servlet Filter

Servlet Filter 是 Java Servlet 规范中定义的组件，翻译成中文就是过滤器，它可以实现对 HTTP 请求的过滤功能，比如鉴权、限流、记录日志、验证参数等等。因为它是 Servlet 规范的一部分，所以，只要是支持 Servlet 的 Web 容器（比如，Tomcat、Jetty 等），都支持过滤器功能。为了帮助你理解，我画了一张示意图阐述它的工作原理，如下所示。



在实际项目中，我们该如何使用 Servlet Filter 呢？我写了一个简单的示例代码，如下所示。添加一个过滤器，我们只需要定义一个实现 `javax.servlet.Filter` 接口的过滤器类，并且将它配置在 `web.xml` 配置文件中。Web 容器启动的时候，会读取 `web.xml` 中的配置，创建过滤器对象。当有请求到来的时候，会先经过过滤器，然后才由 Servlet 来处理。

复制代码

```
1 public class LogFilter implements Filter {
2     @Override
3     public void init(FilterConfig filterConfig) throws ServletException {
4         // 在创建Filter时自动调用,
5         // 其中filterConfig包含这个Filter的配置参数, 比如name之类的 (从配置文件中读取的)
6     }
7
8     @Override
9     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
10        System.out.println("拦截客户端发送来的请求.");
11        chain.doFilter(request, response);
12        System.out.println("拦截发送给客户端的响应.");
13    }
```

```

14     @Override
15     public void destroy() {
16         // 在销毁Filter时自动调用
17     }
18 }
19 }
20
21 // 在web.xml配置文件中如下配置:
22 <filter>
23     <filter-name>logFilter</filter-name>
24     <filter-class>com.xzg.cd.LogFilter</filter-class>
25 </filter>
26 <filter-mapping>
27     <filter-name>logFilter</filter-name>
28     <url-pattern>/*</url-pattern>
29 </filter-mapping>

```

从刚刚的示例代码中，我们发现，添加过滤器非常方便，不需要修改任何代码，定义一个实现 `javax.servlet.Filter` 的类，再改改配置就搞定了，完全符合开闭原则。那 Servlet Filter 是如何做到如此好的扩展性的呢？我想你应该已经猜到了，它利用的就是职责链模式。现在，我们通过剖析它的源码，详细地看看它底层是如何实现的。

在上一节课中，我们讲到，职责链模式的实现包含处理器接口（`IHandler`）或抽象类（`Handler`），以及处理器链（`HandlerChain`）。对应到 Servlet Filter，`javax.servlet.Filter` 就是处理器接口，`FilterChain` 就是处理器链。接下来，我们重点来看 `FilterChain` 是如何实现的。


不过，我们前面也讲过，Servlet 只是一个规范，并不包含具体的实现，所以，Servlet 中的 `FilterChain` 只是一个接口定义。具体的实现类由遵从 Servlet 规范的 Web 容器来提供，比如，`ApplicationFilterChain` 类就是 Tomcat 提供的 `FilterChain` 的实现类，源码如下所示。

为了让代码更易读懂，我对代码进行了简化，只保留了跟设计思路相关的代码片段。完整的代码你可以自行去 Tomcat 中查看。

```

1 public final class ApplicationFilterChain implements FilterChain {
2     private int pos = 0; //当前执行到了哪个filter
3     private int n; //filter的个数
4     private ApplicationFilterConfig[] filters;
5     private Servlet servlet;

```

 复制代码

```

6
7  @Override
8  public void doFilter(ServletRequest request, ServletResponse response) {
9      if (pos < n) {
10         ApplicationFilterConfig filterConfig = filters[pos++];
11         Filter filter = filterConfig.getFilter();
12         filter.doFilter(request, response, this);
13     } else {
14         // filter都处理完毕后, 执行servlet
15         servlet.service(request, response);
16     }
17 }
18
19 public void addFilter(ApplicationFilterConfig filterConfig) {
20     for (ApplicationFilterConfig filter:filters)
21         if (filter==filterConfig)
22             return;
23
24     if (n == filters.length) { //扩容
25         ApplicationFilterConfig[] newFilters = new ApplicationFilterConfig[n + 1];
26         System.arraycopy(filters, 0, newFilters, 0, n);
27         filters = newFilters;
28     }
29     filters[n++] = filterConfig;
30 }
31 }

```

ApplicationFilterChain 中的 doFilter() 函数的代码实现比较有技巧，实际上是一个递归调用。你可以用每个 Filter（比如 LogFilter）的 doFilter() 的代码实现，直接替换 ApplicationFilterChain 的第 12 行代码，一眼就能看出是递归调用了。我替换了一下，如下所示。

 复制代码

```

1  @Override
2  public void doFilter(ServletRequest request, ServletResponse response) {
3      if (pos < n) {
4         ApplicationFilterConfig filterConfig = filters[pos++];
5         Filter filter = filterConfig.getFilter();
6         //filter.doFilter(request, response, this);
7         //把filter.doFilter的代码实现展开替换到这里
8         System.out.println("拦截客户端发送来的请求.");
9         chain.doFilter(request, response); // chain就是this
10        System.out.println("拦截发送给客户端的响应.")
11    } else {
12        // filter都处理完毕后, 执行servlet
13        servlet.service(request, response);
14    }

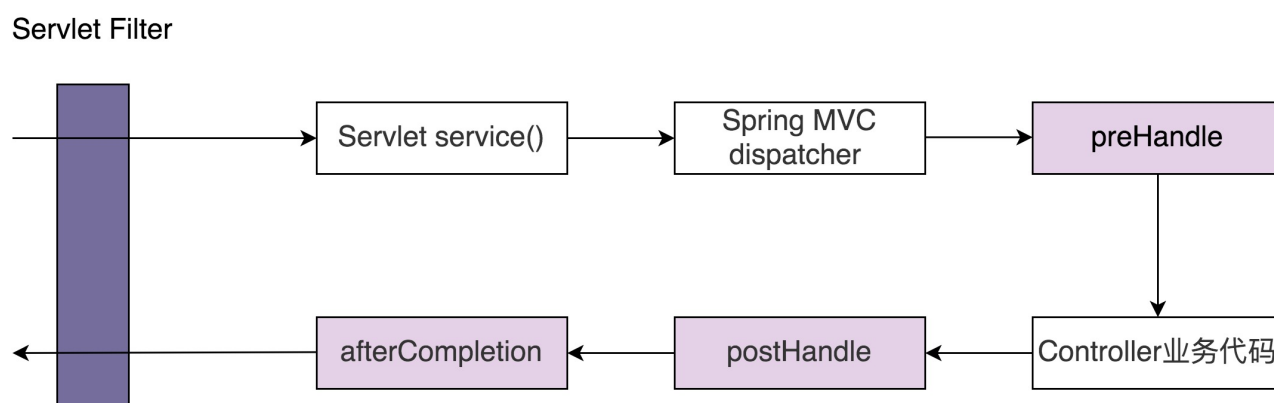
```

这样实现主要是为了在一个 `doFilter()` 方法中，支持双向拦截，既能拦截客户端发送来的请求，也能拦截发送给客户端的响应，你可以结合着 `LogFilter` 那个例子，以及对比待会要讲到的 `Spring Interceptor`，来自自己理解一下。而我们上一节课给出的两种实现方式，都没法做到在业务逻辑执行的前后，同时添加处理代码。

Spring Interceptor

刚刚讲了 `Servlet Filter`，现在我们来讲一个功能上跟它非常类似的东西，`Spring Interceptor`，翻译成中文就是拦截器。尽管英文单词和中文翻译都不同，但这两者基本上可以看作一个概念，都用来实现对 HTTP 请求进行拦截处理。

它们不同之处在于，`Servlet Filter` 是 `Servlet` 规范的一部分，实现依赖于 Web 容器。`Spring Interceptor` 是 `Spring MVC` 框架的一部分，由 `Spring MVC` 框架来提供实现。客户端发送的请求，会先经过 `Servlet Filter`，然后再经过 `Spring Interceptor`，最后到达具体的业务代码中。我画了一张图来阐述一个请求的处理流程，具体如下所示。



在项目中，我们该如何使用 `Spring Interceptor` 呢？我写了一个简单的示例代码，如下所示。`LogInterceptor` 实现的功能跟刚才的 `LogFilter` 完全相同，只是实现方式上稍有区别。`LogFilter` 对请求和响应的拦截是在 `doFilter()` 一个函数中实现的，而 `LogInterceptor` 对请求的拦截在 `preHandle()` 中实现，对响应的拦截在 `postHandle()` 中实现。

```

2
3  @Override
4  public boolean preHandle(HttpServletRequest request, HttpServletResponse response) {
5      System.out.println("拦截客户端发送来的请求.");
6      return true; // 继续后续的处理
7  }
8
9  @Override
10 public void postHandle(HttpServletRequest request, HttpServletResponse response) {
11     System.out.println("拦截发送给客户端的响应.");
12 }
13
14 @Override
15 public void afterCompletion(HttpServletRequest request, HttpServletResponse response) {
16     System.out.println("这里总是被执行.");
17 }
18 }
19
20 //在Spring MVC配置文件中配置interceptors
21 <mvc:interceptors>
22     <mvc:interceptor>
23         <mvc:mapping path="/*"/>
24         <bean class="com.xzg.cd.LogInterceptor" />
25     </mvc:interceptor>
26 </mvc:interceptors>

```

同样，我们还是来剖析一下，Spring Interceptor 底层是如何实现的。

当然，它也是基于职责链模式实现的。其中，HandlerExecutionChain 类是职责链模式中的处理器链。它的实现相较于 Tomcat 中的 ApplicationFilterChain 来说，逻辑更加清晰，不需要使用递归来实现，主要是因为它将请求和响应的拦截工作，拆分到了两个函数中实现。HandlerExecutionChain 的源码如下所示，同样，我对代码也进行了一些简化，只保留了关键代码。

 复制代码

```

1 public class HandlerExecutionChain {
2     private final Object handler;
3     private HandlerInterceptor[] interceptors;
4
5     public void addInterceptor(HandlerInterceptor interceptor) {
6         initInterceptorList().add(interceptor);
7     }
8
9     boolean applyPreHandle(HttpServletRequest request, HttpServletResponse response) {
10         HandlerInterceptor[] interceptors = getInterceptors();
11         if (!ObjectUtils.isEmpty(interceptors)) {

```

```

12     for (int i = 0; i < interceptors.length; i++) {
13         HandlerInterceptor interceptor = interceptors[i];
14         if (!interceptor.preHandle(request, response, this.handler)) {
15             triggerAfterCompletion(request, response, null);
16             return false;
17         }
18     }
19 }
20 return true;
21 }
22
23 void applyPostHandle(HttpServletRequest request, HttpServletResponse response
24     HandlerInterceptor[] interceptors = getInterceptors();
25     if (!ObjectUtils.isEmpty(interceptors)) {
26         for (int i = interceptors.length - 1; i >= 0; i--) {
27             HandlerInterceptor interceptor = interceptors[i];
28             interceptor.postHandle(request, response, this.handler, mv);
29         }
30     }
31 }
32
33 void triggerAfterCompletion(HttpServletRequest request, HttpServletResponse r
34     throws Exception {
35     HandlerInterceptor[] interceptors = getInterceptors();
36     if (!ObjectUtils.isEmpty(interceptors)) {
37         for (int i = this.interceptorIndex; i >= 0; i--) {
38             HandlerInterceptor interceptor = interceptors[i];
39             try {
40                 interceptor.afterCompletion(request, response, this.handler, ex);
41             } catch (Throwable ex2) {
42                 logger.error("HandlerInterceptor.afterCompletion threw exception", ex2);
43             }
44         }
45     }
46 }
47 }

```

在 Spring 框架中，DispatcherServlet 的 doDispatch() 方法来分发请求，它在真正的业务逻辑执行前后，执行 HandlerExecutionChain 中的 applyPreHandle() 和 applyPostHandle() 函数，用来实现拦截的功能。具体的代码实现很简单，你自己应该能脑补出来，这里就不罗列了。感兴趣的话，你可以自行去查看。

重点回顾

好了，今天的内容到此就讲完了。我们一块来总结回顾一下，你需要重点掌握的内容。

职责链模式常用在框架开发中，用来实现框架的过滤器、拦截器功能，让框架的使用者在不需要修改框架源码的情况下，添加新的过滤拦截功能。这也体现了之前讲到的对扩展开放、对修改关闭的设计原则。

今天，我们通过 Servlet Filter、Spring Interceptor 两个实际的例子，给你展示了在框架开发中职责链模式具体是怎么应用的。从源码中，我们还可以发现，尽管上一节课中我们有给出职责链模式的经典代码实现，但在实际的开发中，我们还是要具体问题具体对待，代码实现会根据不同的需求有所变化。实际上，这一点对于所有的设计模式都适用。

课堂讨论

1. 前面在讲代理模式的时候，我们提到，Spring AOP 是基于代理模式来实现的。在实际的项目开发中，我们可以利用 AOP 来实现访问控制功能，比如鉴权、限流、日志等。今天我们又讲到，Servlet Filter、Spring Interceptor 也可以用来实现访问控制。那在项目开发中，类似权限这样的访问控制功能，我们该选择三者（AOP、Servlet Filter、Spring Interceptor）中的哪个来实现呢？有什么参考标准吗？
2. 除了我们讲到的 Servlet Filter、Spring Interceptor 之外，Dubbo Filter、Netty ChannelPipeline 也是职责链模式的实际应用案例，你能否找一个你熟悉的并且用到职责链模式的框架，像我一样分析一下它的底层实现呢？

欢迎留言和我分享你的想法。如果有收获，欢迎你把这篇文章分享给你的朋友。

打卡 3 道题 「免费」领课程

🕒 3月30日-4月5日



【点击】图片, 立即领取

© 版权归极客邦科技所有, 未经许可不得传播售卖。页面已增加防盗追踪, 如有侵权极客邦将依法追究其法律责任。

上一篇 62 | 职责链模式 (上) : 如何实现可灵活扩展算法的敏感信息过滤框架?

下一篇 64 | 状态模式: 游戏、工作流引擎中常用的状态机是如何实现的?

精选留言 (24)

💬 写留言



cricket1981

2020-03-27

Filter 可以拿到原始的http请求, 但是拿不到你请求的控制器和请求控制器中的方法的信息; Interceptor 可以拿到你请求的控制器和方法, 却拿不到请求方法的参数; Aop 可以拿到方法的参数, 但是却拿不到http请求和响应的对象

展开 ∨

💬 1

👍 14



筱乐乐哦

2020-03-27

1、个人感觉权限的话, 属于api的调用, 应该放在调用链比较靠前的位置, 早发现早处理, 所以用Servlet Filter会更好一些吧, 如果是rpc层的话, 例如dubbo, 就需要 在实现fil

ter的时候通过order吧filter得优先级提高一些，让这个filter先执行，个人感觉哈

2、Dubbo Filter的核心处理逻辑在ProtocolFilterWrapper类下的buildInvokerChain这个方法中，属于把所有的filter的类对象搞成一个list，通过遍历list去调用所有的filter，N...

展开 ▾



13



PCMD

2020-03-27

针对问题1而言，其实要实现一个鉴权的过滤器，通过以上3种方式都是可以去实现的，然而从粒度，场景，和方式上边有所区别，主要采取用哪个，还是有业务来决定去用，没有统一的参考标准。比如要对所有的web接口，进行统一的权限处理，不需要区分动作，写或者读，所有一视同仁，这种情况下，servlet的更加适合。针对一些存在状态的，比如做一些统一的去参数转换，cookie转uid之类，以及通用检验uid是否符合当前权限，则...

展开 ▾



13



小晏子

2020-03-27

首先需要明确“访问控制功能”的粒度，如果访问控制功能要精确到每个请求，那么要使用AOP，AOP可以配置每个controller的访问权限。而Spring interceptor和servlet filter的粒度会粗一些，控制HttpRequest, HttpResponse的访问。另外servlet filter不能够使用Spring容器资源，只能在容器（如tomcat）启动时调用一次，而Spring Interceptor是一个Spring的组件，归Spring管理，配置在Spring文件中，因此能使用Spring里的任...

展开 ▾



4



Xs.Ten

2020-03-27

即时通讯里面的消息分发可以用到责任链模式。可以添加不同的分发规则来分发不同的消息类型到各个消息处理器。

展开 ▾



4



袁帅

2020-03-27

1、权限应该使用servlet filter，servlet filter 是对早被执行的，可为所有request加拦截，如果仅仅想对web请求加权限，那么使用spring interceptor

展开 ▾



2



Geek 54edc1
2020-03-27

思考题一：首先要区分三者的特点，Spring AOP的使用粒度是类，是对类的一个包装；servlet filter和spring interceptor主要是对HttpRequest、HttpResponse做处理，servlet filterChain的实现依赖于具体的Web容器，而spring interceptor和spring AOP都依赖于spring框架，servlet filter在一个函数里拦截请求和响应，而spring interceptor将请求、响应的拦截分成了两个函数；其次，针对特定的应用场景，选择适合的。

展开 ▾



1



LiG
2020-03-27

老师，请问Servlet Filter中采用递归方式调用职责链中元素，Spring Interceptor中采用数组变量方式调用职责链中元素，这两种调用方式，是基于什么考虑的，有什么优劣呢？



1



++
2020-03-27

安卓的网络请求框架 okhttp中使用了责任链

展开 ▾



1



test
2020-03-27

偏业务的使用AOP，全局的使用servlet filter

展开 ▾



1



Yang
2020-03-27

但在实际的开发中，我们还是要具体问题具体对待，代码实现会根据不同的需求有所变化。实际上，这一点对于所有的设计模式都适用。
这句话很精辟



1



李小四
2020-03-31

设计模式_63:

作业

1. 这个问题确实不懂，看了大家的回答，了解到对于颗粒度的考虑。

2. 有一个Java/Kotlin的网络库叫Okhttp，也用到了职责链模式，用来做一些日志记录等。

...

展开 ▾



jiajia

2020-03-30

okhttp

展开 ▾



Geek_27a248

2020-03-30

比如鉴权，限流，日志三个方面的话，鉴权可以使用spring interceptor，spring管理的获取的信息比较多，方便做更细的鉴权；限流的话可以使用servlet filter，执行比较靠前，最大程度降低后方请求数量。日志的话使用aop，最大细度的记录日志信息。不知道这样理解的对不对，对这些的理解还是比较片面的不够深入，还有待加深

展开 ▾



朱晋君

2020-03-29

1.如果是一个全局的鉴权，用servlet 自有的filter是最好的，实现也简单，如果是有特性的鉴权，用拦截器或者aop注解的方式实现比较好
2.mybatis的拦截器就用了职责链模式，InterceptorChain类作为职责链类，Interceptor接口的实现类作为handler类，加@Intercepts注解来定义

展开 ▾



楊_宵夜

2020-03-29

针对问题1，一把泪水想起了项目中的坑. 个人觉得最大的不同还是生效粒度的问题.

1. Servlet Filter是针对Servlet容器里的方法都能生效. 就是说Servlet容器里就算要把Spring换成别的框架，鉴权代码依然能生效.

2. Spring开头的就只能在Spring中生效，

2.1. 但更好还是在interceptor，因为interceptor天然的设计背景就是[在请求前，在相应...

展开 ▾



木木



2020-03-29

Mybatis中的拦截器也使用到责任链模式，只不过是动态代理的方式，在多个同类型拦截器中通过遍历的方式为被拦截对象生成代理对象，然后使用生成的代理对象作为参数继续生成代理对象，直至遍历结束，拿到最外层代理对象，触发invoke方法完成链式拦截器传递

展开 ▾



webmin

2020-03-29

AOP、Servlet Filter、Spring Interceptor这三者可以从不同权限检查的范围大小的视角来应用：

1. Servlet Filter

运维部门需要对只供内部访问的服务进行IP限制或访问审查时，在容器这一层增加一个Filter，在发布时发布系统自动加挂这个Filter，这样对上层应用就是透明的，内网IP地址段...

展开 ▾



徐旭

2020-03-28

老师讲得赞👍

展开 ▾



Geek_3b1096

2020-03-28

细读

展开 ▾

