程序通事

个人博客网站：https://studyidea.cn/

写了一套优雅接口之后，领导让我给大家讲讲这背后的技术原理

Hello，各位小伙伴们，早上好~

上周文章年轻人不讲武德，竟然重构出这么优雅后台 API 接口我们使用 `@ControllerAdvice` 与 `ResponseBodyAdvice` 重构后端的 API 接口，降低了复杂度，减少了重复代码，后续接口开发非常简洁优雅。

知其然而知其所以然，今天这篇文章来聊聊这个注解背后的原理，让我们彻底掌握这个注解，避免后续踩坑。

另外，有个小伙伴看完上篇文章，觉得这个注解的跟 `Spring Interceptor` 功能很类似，再加上之前还学习了 `Servlet` 体系 `Filter` 功能，不知道这几个有什么区别，感觉很混乱。

所以今天这篇文章下面两个部分出发，详细解释一下。

1. `@ControllerAdvice` 与 `ResponseBodyAdvice` 注解原理
2. `Filter` ， `Interceptor` ， `ResponseBodyAdvice` 区别

> 欢迎关注我的公众号：程序通事，获得日常干货推送。如果您对我的专题内容感兴趣，也可以关注我的博客：studyidea.cn

从源码解析背后的原理

上篇文章中我们看到 `ResponseBodyAdvice` 的子类使用 `@ControllerAdvice` 注解，大家有没有好奇，如果我将 `@ControllerAdvice` 换成 `@Controller` 注解，还能达到上篇文章的效果吗?

感兴趣的小伙伴可以自己尝试下，这里小黑哥自己告诉大家结果了，实际测试结果是不行的。

那为什么一定要与 `@ControllerAdvice` 搭配才会生效?

首先我们先查看一下 `@ControllerAdvice` 的源码：

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component
public @interface ControllerAdvice {
```

可以看到这个注解上还存在一个我们非常熟悉的 `@Component` 注解。这里我们可以将 `@ControllerAdvice` 理解成 `@Component` 子类，所以其修饰的类也会成为 Spring 中 `Bean` 。

> ps:大家可以看下 `@Controller` / `@Service` / `@Repository` ,其实也是这个原理。

Spring 容器初始化过程，如果扫描到 `@ControllerAdvice` 注解，将会将其生成一个 `ControllerAdviceBean` Bean。

这个过程代码主要位于 `RequestMappingHandlerAdapter#initControllerAdviceCache` ：

```
private void initControllerAdviceCache() {
    if (getApplicationContext() == null) {
        return;
    }

1   List<ControllerAdviceBean> adviceBeans = ControllerAdviceBean.findAnnotatedBeans(getApplicationContext());

    List<Object> requestResponseBodyAdviceBeans = new ArrayList<>();

    for (ControllerAdviceBean adviceBean : adviceBeans) {
        Class<?> beanType = adviceBean.getBeanType();
        if (beanType == null) {
            throw new IllegalStateException("Unresolvable type for ControllerAdviceBean: " + adviceBean);
        }
        Set<Method> attrMethods = MethodIntrospector.selectMethods(beanType, MODEL_ATTRIBUTE_METHODS);
        if (!attrMethods.isEmpty()) {
            this.modelAttributeAdviceCache.put(adviceBean, attrMethods);
        }
        Set<Method> binderMethods = MethodIntrospector.selectMethods(beanType, INIT_BINDER_METHODS);
        if (!binderMethods.isEmpty()) {
            this.initBinderAdviceCache.put(adviceBean, binderMethods);
        }
2       if (RequestBodyAdvice.class.isAssignableFrom(beanType) || ResponseBodyAdvice.class.isAssignableFrom(beanType)) {
            requestResponseBodyAdviceBeans.add(adviceBean);
        }
    }

    if (!requestResponseBodyAdviceBeans.isEmpty()) {
        this.requestResponseBodyAdvice.addAll( index: 0, requestResponseBodyAdviceBeans);
    }
```

这段代码主要分为两步：

第一步使用 `ControllerAdviceBean#findAnnotatedBeans` 获取所有被 `@ControllerAdvice` 修饰的类。

```
/**
 * Find beans annotated with {@link ControllerAdvice @ControllerAdvice} in the
 * given {@link ApplicationContext} and wrap them as {@code ControllerAdviceBean}
 * instances.
 * <p>As of Spring Framework 5.2, the {@code ControllerAdviceBean} instances
 * in the returned list are sorted using {@link OrderComparator#sort(List)}.
 * @see #getOrder()
 * @see OrderComparator
 * @see Ordered
 */
public static List<ControllerAdviceBean> findAnnotatedBeans(ApplicationContext context) {
    List<ControllerAdviceBean> adviceBeans = new ArrayList<>();
    for (String name : BeanFactoryUtils.beanNamesForTypeIncludingAncestors(context, Object.class)) {
        if (!ScopedProxyUtils.isScopedTarget(name)) {
            ControllerAdvice controllerAdvice = context.findAnnotationOnBean(name, ControllerAdvice.class);
            if (controllerAdvice != null) {
                // Use the @ControllerAdvice annotation found by findAnnotationOnBean()
                // in order to avoid a subsequent lookup of the same annotation.
                adviceBeans.add(new ControllerAdviceBean(name, context, controllerAdvice));
            }
        }
    }
    OrderComparator.sort(adviceBeans);
    return adviceBeans;
}
```

第二步将所有实现了 `ResponseBodyAdvice` 接口的 Bean 放入到 `requestResponseBodyAdviceBeans` 集合中，后续将会使用该集合。

```
if (RequestBodyAdvice.class.isAssignableFrom(beanType) || ResponseBodyAdvice.class.isAssignableFrom(beanType)) {
    requestResponseBodyAdviceBeans.add(adviceBean);
}
```

支付系统(12)
dubbo(10)
Spring(8)
IDEA(8)
Redis(5)
系统设计(4)
开发工具(4)
对账系统(4)
踩坑经历(4)
更多

积分与排名
积分 - 251436
排名 - 4045

随笔分类
IDEA(6)
Spring(6)
第三方支付(1)

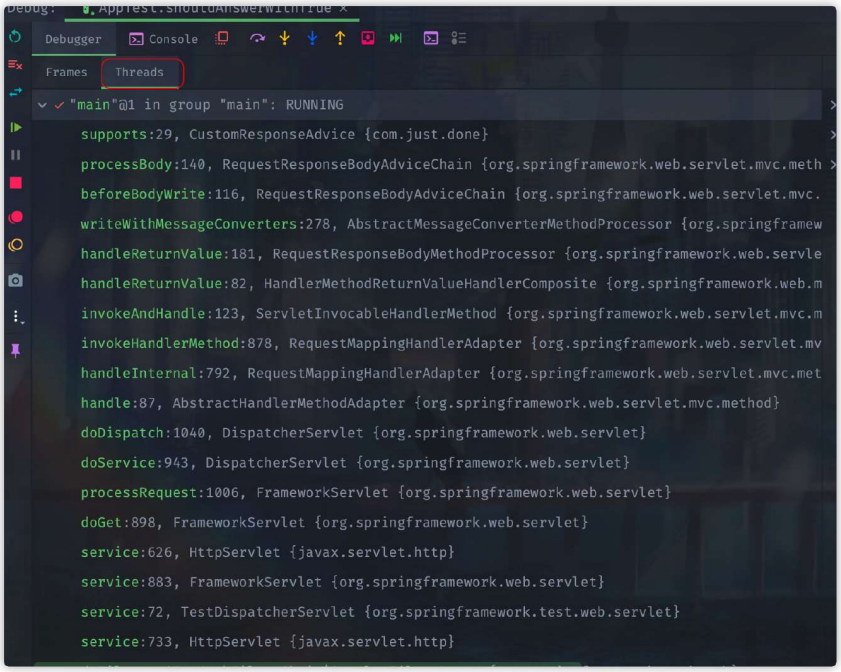
随笔档案
2022年1月(3)
2021年12月(2)
2021年7月(2)
2021年5月(3)

3
升级成为会员

这就解释了为什么实现 `ResponseBodyAdvice` 接口的子类一定要与 `@ControllerAdvice` 一起使用的原因了。

接下来我们来看下 `ResponseBodyAdvice` 的执行流程。

这里教给大家一个代码调试的小技巧，当我们不知道一个类在源码中如何被调用的时候，我们可以使用 IDEA 代码调试功能，然后查看代码调用栈。



如上面的所示，我们可以很清楚观察 `ResponseBodyAdvice` 调用关系。这里的类调用关系相对还是比较复杂，下面给大家简化一下。

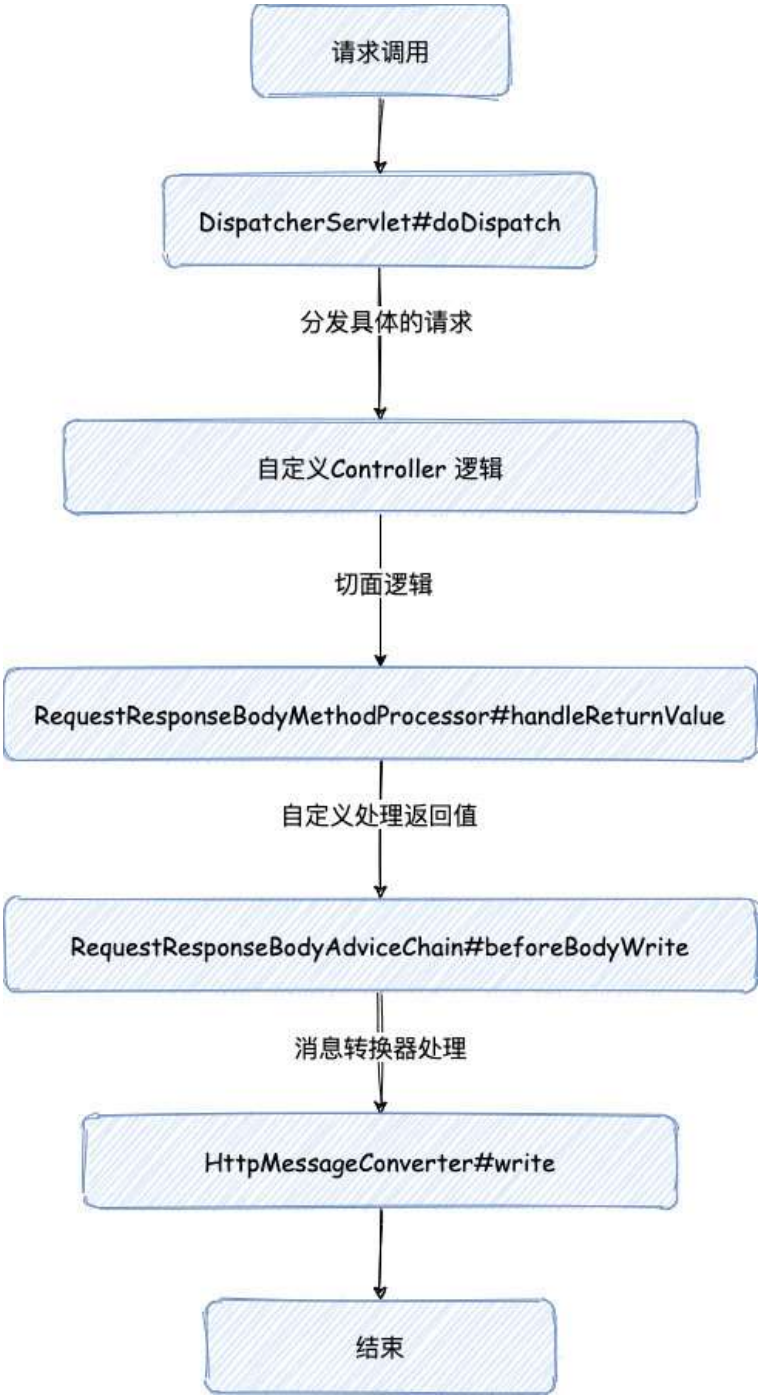
2021年2月(1)
2021年1月(1)
2020年12月(4)
2020年11月(2)
2020年10月(2)
2020年9月(5)
2020年8月(3)
2020年7月(4)
2020年6月(6)
2020年5月(4)
2020年4月(7)
更多

阅读排行榜

1. 设计数据库 ER 图太麻烦？不妨试试这两款工具，自动生成数据库 ER 图！！！(18124)
2. SM 国密算法踩坑指南(17805)
3. IDEA Debug 无法进入断点的解决方法(15603)
4. 面试官再问我如何保证 RocketMQ 不丢失消息,这回我笑了！(13487)
5. Spring 注解编程之 AnnotationMetadata(12052)

评论排行榜

1. 求求你了，不要再



前面的逻辑就不说了，就是 Spring MVC 通用流程。重点逻辑位于 `RequestResponseBodyAdviceChain` ，我们具体看下源码：

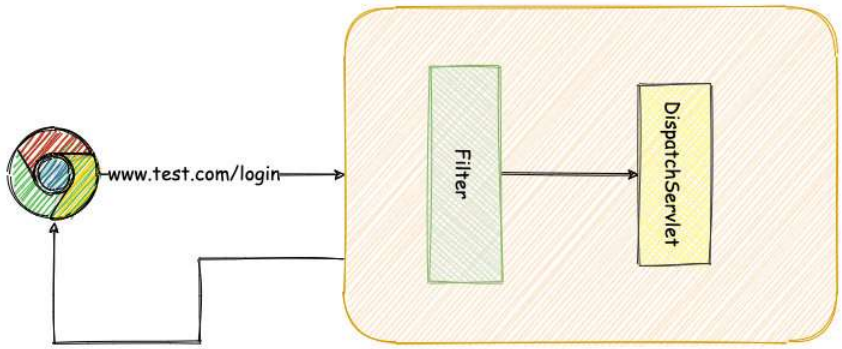
```
@Nullable
private <T> Object processBody(@Nullable Object body, MethodParameter returnType, MediaType contentType,
        Class<? extends HttpMessageConverter<?>> converterType,
        ServerHttpRequest request, ServerHttpResponse response) {

    for (ResponseBodyAdvice<?> advice : getMatchingAdvice(returnType, ResponseBodyAdvice.class)) {
        if (advice.supports(returnType, converterType)) {
            body = ((ResponseBodyAdvice<T>) advice).beforeBodyWrite((T) body, returnType,
                    contentType, converterType, request, response);
        }
    }
    return body;
}
```

> 嗯呐嗯呐，请忽略上图的 ③

其实逻辑非常简单，遍历所有的 `ResponseBodyAdvice` 的子类，首先调用其 `supports` 判断是否支持，如果支持的调用的 `beforeBodyWrite` 修改返回信息。

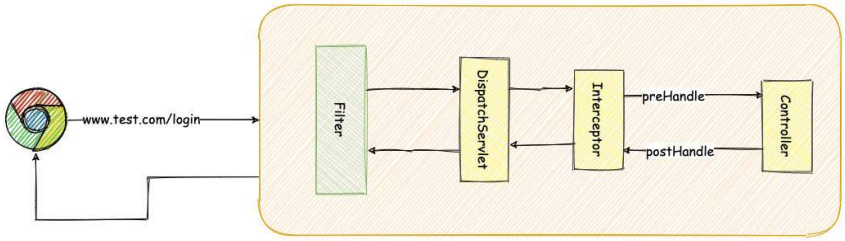
`Filter` 、 `Interceptor` 、 `ResponseBodyAdvice` 区别

`Filter` 属于 Servlet 组件，所有请求将会先进入 `Filter` ，判断通过之后才会在进入到真正的具体的请求中。



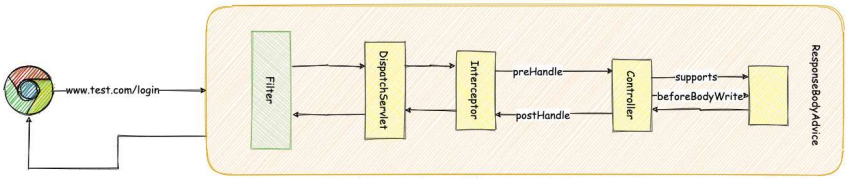
上图代表是用 Spring MVC 的一个 Web 项目，所有请求将会先进入到 `Filter` ，通过之后才会进入到 SpringMVC 中最重要的组件 `DispatchServlet` 。

而 `Interceptor` 是 SpringMVC 的组件，它的作用实际上与 `Filter` 类似， 只不过的它的作用是位于自定义的 `Controller` 前后。



不管是 `Filter` 还是 `Interceptor` ，它们的作用方法域内只能拿到 `ServletResponse` 的参数，这个时候返回值已经被写入 `ServletResponse` ，我们很难再去修改。

而 `ResponseBodyAdvice` 作用时机位于写入之前，所以这个时候可以很容易拿到原值进行修改。



总结

SpringMVC 初始化的过程中，将会扫描所有带有 `@ControllerAdvice` 注解的类，将其生成为 `ControllerAdviceBean` 。如果这类刚好为 `ResponseBodyAdvice` 接口的子类，Spring 将会为其单独保存起来，后续将会封装到的 `RequestResponseBodyAdviceChain` ，使用责任链的模式对请求、响应进行处理。

最后我们解释了一下 `Filter` ， `Interceptor` ， `ResponseBodyAdvice` 区别，从作用范围上来讲：

```
Filter>Interceptor>ResponseBodyAdvice
```

但是前两者没办法修改返回值（时机太晚），只有后者才可以真正在返回值返回之前做到修改。

好了，今天文章就到这里了，下次我们分享一下如何写出优雅的 Dubbo 接口，下次见。

> 欢迎关注我的公众号：程序通事，获得日常干货推送。如果您对我的专题内容感兴趣，也可以关注我的博客：studyidea.cn

标签: Java， Spring

--忧郁的熊猫人

3. Re:SM 国密算法踩坑指南

请问有签名公钥证书后, 怎么获取自身证书对应的私钥信息呢

--习惯沉淀

4. Re:贞炸了！上线之后，消息收不到了！

『今天的问题主要由于 VIP 端口无法连接，从而导致消费端无法正常消费消息。』准确的说法不是 VIP 端口无法连接，导致生产者发送消息失败，消费端才无法消费到消息吗？...

--小小配角

5. Re:巧用 Base62 解决字段太短的问题

感觉还是58进制使用起来比较方便，避开了四个容易混淆的字符。

--吸氧羊与肥汰狼

楼下小黑哥
粉丝 - 147 关注 - 11
+加关注

« 上一篇： 年轻人不讲武德，竟然重构出这么优雅后台 API 接口
» 下一篇： 数据库读写分离这个坑，你应该踩过吧？

posted @ 2020-12-02 08:57 楼下小黑哥 阅读(1635) 评论(5) 编辑 收藏 举报