还在用策略模式解决 if-else？Map+函数式接口方法才是YYDS！

本文介绍策略模式的具体应用以及Map+函数式接口如何“更完美”的解决 if-else的问题。

需求

最近写了一个服务：根据优惠券的类型resourceType和编码resourceId来 查询 发放方式grantType和领取规则

实现方式：

- 根据优惠券类型resourceType -> 确定查询哪个数据表
- 根据编码resourceId -> 到对应的数据表里边查询优惠券的派发方式grantType和领取规则

优惠券有多种类型，分别对应了不同的数据库表：

- 红包 —— 红包发放规则表
- 购物券 —— 购物券表
- QQ会员
- 外卖会员

实际的优惠券远不止这些，这个需求是要我们写一个业务分派的逻辑

第一个能想到的思路就是if-else或者switch case：

```
switch(resourceType){
 case "红包":
  查询红包的派发方式
  break;
 case "购物券":
  查询购物券的派发方式
  break;
 case "QQ会员" :
  break;
 case "外卖会员" :
  break;
 ......
 default : logger.info("查找不到该优惠券类型resourceType以及对应的派发方式");
  break;
}
```

如果要这么写的话， 一个方法的代码可就太长了，影响了可读性。（别看着上面case里面只有一句话，但实际情况是有很多行的）

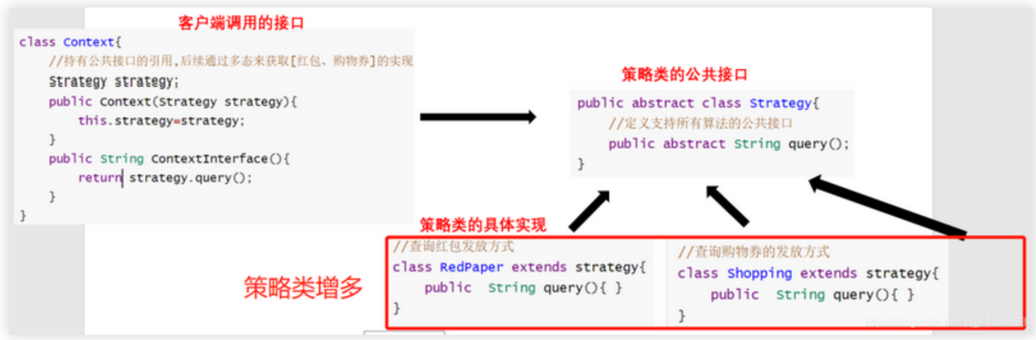
而且由于 整个 if-else的代码有很多行，也不方便修改，可维护性低。

> 基于 Spring Cloud Alibaba + Gateway + Nacos + RocketMQ + Vue & Element 实现的后台管理系统 + 用户小程序，支持 RBAC 动态权限、多租户、数据权限、工作流、三方登录、支付、短信、商城等功能
>
> - 项目地址：https://github.com/YunaiV/yudao-cloud
> - 视频教程：https://doc.iocoder.cn/video/

策略模式

策略模式是把 if语句里面的逻辑抽出来写成一个类，如果要修改某个逻辑的话，仅修改一个具体的实现类的逻辑即可，可维护性会好不少。

以下是策略模式的具体结构



策略模式在业务逻辑分派的时候还是if-else，只是说比第一种思路的if-else 更好维护一点。

```
switch(resourceType){
 case "红包":
  String grantType=new Context(new RedPaper()).ContextInterface();
  break;
 case "购物券":
  String grantType=new Context(new Shopping()).ContextInterface();
  break;

 ......
 default : logger.info("查找不到该优惠券类型resourceType以及对应的派发方式");
  break;
```

但缺点也明显：

- 如果 if-else的判断情况很多，那么对应的具体策略实现类也会很多，上边的具体的策略实现类还只是2个，查询红包发放方式写在类RedPaper里边，购物券写在另一个类Shopping里边；那资源类型多个QQ会员和外卖会员，不就得再多写两个类？有点麻烦了
- 没法俯视整个分派的业务逻辑

Map+函数式接口

用上了Java8的新特性lambda表达式

- 判断条件放在key中
- 对应的业务逻辑放在value中

这样子写的好处是非常直观，能直接看到判断条件对应的业务逻辑

> 需求：根据优惠券(资源)类型resourceType和编码resourceId查询派发方式grantType

上代码：

```java
@Service
public class QueryGrantTypeService {

    @Autowired
    private GrantTypeSerive grantTypeSerive;
    private Map<String, Function<String,String>> grantTypeMap=new HashMap<>();

    /**
     *  初始化业务分派逻辑,代替了if-else部分
     *  key: 优惠券类型
     *  value: lambda表达式,最终会获得该优惠券的发放方式
     */
    @PostConstruct
    public void dispatcherInit(){
        grantTypeMap.put("红包",resourceId->grantTypeSerive.redPaper(resourceId));
        grantTypeMap.put("购物券",resourceId->grantTypeSerive.shopping(resourceId));
        grantTypeMap.put("qq会员",resourceId->grantTypeSerive.QQVip(resourceId));
    }

    public String getResult(String resourceType){
        //Controller根据 优惠券类型resourceType、编码resourceId 去查询 发放方式grantType
        Function<String,String> result=getGrantTypeMap.get(resourceType);
        if(result!=null){
         //传入resourceId 执行这段表达式获得String型的grantType
            return result.apply(resourceId);
        }
        return "查询不到该优惠券的发放方式";
    }
}
```

如果单个 if 语句块的业务逻辑有很多行的话，我们可以把这些 业务操作抽出来，写成一个单独的Service，即：

```java
//具体的逻辑操作

@Service
public class GrantTypeSerive {

    public String redPaper(String resourceId){
        //红包的发放方式
        return "每周末9点发放";
    }
    public String shopping(String resourceId){
        //购物券的发放方式
        return "每周三9点发放";
    }
    public String QQVip(String resourceId){
        //qq会员的发放方式
        return "每周一0点开始秒杀";
    }
}
```

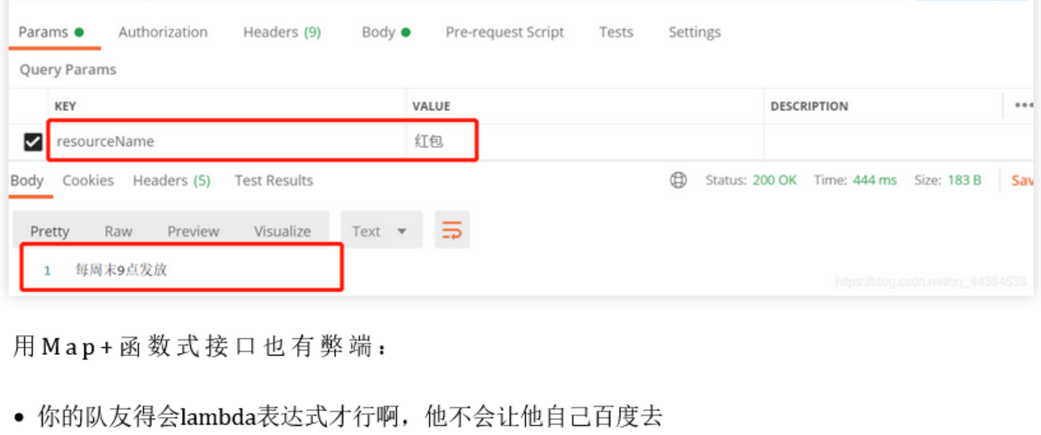
入参String resourceId是用来查数据库的，这里简化了，传参之后不做处理。

用http调用的结果：

```java
@RestController
public class GrantTypeController {

    @Autowired
    private QueryGrantTypeService queryGrantTypeService;

    @PostMapping("/grantType")
    public String test(String resourceName){
        return queryGrantTypeService.getResult(resourceName);
    }
}
```



用Map+函数式接口也有弊端：

- 你的队友得会lambda表达式才行啊，他不会让他自己百度去

最后捋一捋本文讲了什么

策略模式通过接口、实现类、逻辑分派来完成，把 if语句块的逻辑抽出来写成一个类，更好维护。

Map+函数式接口通过Map.get(key)来代替 if-else的业务分派，能够避免策略模式带来的类增多、难以俯视整个业务逻辑的问题。

欢迎加入我的知识星球，一起探讨架构，交流源码。加入方式，**长按下方二维码噢**：

已在知识星球更新源码解析如下：

《精尽面试题（包括答案）》

01. Dubbo 面试题
02. Netty 面试题
03. Spring 面试题
04. Spring MVC 面试题
05. Spring Boot 面试题
06. Spring Cloud 面试题
07. MyBatis 面试题
08. 消息队列面试题
09. RocketMQ 面试题
10. RabbitMQ 面试题
11. Kafka 面试题
12. 缓存面试题
13. Redis 面试题
14. MySQL 面试题
15.【分库分表】面试题
16.【分布式事务】面试题
17. Elasticsearch 面试题
18. MongoDB 面试题
19. 设计模式面试题
20. Java【基础】面试题
21. Java【集合】面试题
22. Java【并发】面试题
23. Java【虚拟机】面试题
24. Linux 面试题
25. Git 面试题
26. 计算机网络面试题
27. Maven 面试题
28. Jenkins 面试题
29. Zookeeper 面试题
30. Nginx 面试题
31. 数据结构与算法面试题

《精尽学习指南（包括视频）》

00. 精尽学习指南 —— 路线
01. Dubbo 学习指南
02. Netty 学习指南
03. Spring 学习指南
04. Spring MVC 学习指南
05. Spring Boot 学习指南
06. Spring Cloud 学习指南
06. Spring Cloud Alibaba 学习指南
07. MyBatis 学习指南
08. Hiberante 学习指南
09. RocketMQ 学习指南
10. RabbitMQ 学习指南
11. Kafka 学习指南
12. Redis 学习指南
13. MySQL 学习指南
14. MongoDB 学习指南
15. Elasticsearch 学习指南
16. 设计模式学习指南
17. Java【基础】学习指南
18. Java【并发】学习指南
19. Java【虚拟机】学习指南
21. Linux 学习指南
22. 数据结构与算法学习指南
23. 计算机网络学习指南
24. Maven 学习指南
25. Jenkins 学习指南
26. Git 学习指南
27. Intellij IDEA 学习指南
28. Docker 学习指南
29. Kubernetes 学习指南
30. Zookeeper 学习指南
31. Nginx 学习指南
32. 任务调度学习指南
33. React 学习指南
34. Vue 学习指南

《Dubbo 源码解析》

01. 调试环境搭建
02. 项目结构一览
03. 配置 Configuration
04. 核心流程一览
05. 拓展机制 SPI
06. 线程池 ThreadPool
07. 服务暴露 Export
08. 服务引用 Refer
09. 注册中心 Registry
10. 动态编译 Compile
11. 动态代理 Proxy
12. 服务调用 Invoke
13. 调用特性
14. 过滤器 Filter
15. NIO 服务器
16. P2P 服务器
17. HTTP 服务器
18. 序列化 Serialization
19. 集群容错 Cluster
20. 优雅停机 Shutdown
21. 日志适配 Logging
22. 状态检查 Status
23. 监控中心 Monitor
24. 管理中心 Admin
25. 运维命令 QOS
26. 链路追踪 Tracing
27. Spring Boot 集成
28. Spring Cloud 集成

... 一共 **73+** 篇

《Spring Cloud 源码解析》

01. 网关 Spring Cloud Gateway 25 篇
02. 注册中心 Eureka 23 篇
03. 熔断器 Hystrix 9 篇
04. 配置中心 Apollo 32 篇
05. 链路追踪 SkyWalking 38 篇
06. 调度中心 Elastic Job 24 篇

《Netty 源码解析》

01. 调试环境搭建
02. NIO 基础
03. Netty 简介
04. 启动 Bootstrap
05. 事件轮询 EventLoop
06. 通道管道 ChannelPipeline
07. 通道 Channel
08. 字节缓冲区 ByteBuf
09. 通道处理器 ChannelHandler
10. 编解码 Codec
11. 工具类 Util

... 一共 **61+** 篇

《MyBatis 源码解析》

01. 调试环境搭建
02. 项目结构一览
03. MyBatis 初始化
04. SQL 初始化
05. SQL 执行
06. 插件体系
07. Spring 集成

... 一共 **34+** 篇

01. 调试环境搭建
02. 项目结构一览
03. SpringApplication
04. 自动配置
05. Condition
06. ServletWebServerApplicationContext
07. ReactiveWebServerApplicationContext
08. ApplicationContextInitializer
09. ApplicationListener
... 一共 **15+** 篇

13. FlashMapManager 组件
... 一共 **24+** 篇

《数据库实体设计》

01. 商品模块
02. 交易模块
03. 营销模块
04. 公用模块
... 一共 **17+** 篇（包括**代码**）

《JDK 源码解析》

01. 调试环境搭建（一）入门
02. 调试环境搭建（一）进阶
03. 集合（一）ArrayList
04. 集合（二）LinkedList
05. 集合（三）HashMap
06. 集合（四）LinkedHashMap
07. 集合（五）HashSet
08. 集合（六）TreeMap
09. 集合（七）TreeSet
... 目前 9+ 篇（**努力更新中**）

《Redis 源码解析》

01. Redis 调试环境搭建
02. Redisson 调试环境搭建
03. Redisson 限流器 RateLimiter
04. Redisson 可重入分布式锁 ReentrantLock
05. Redisson 可靠分布式锁 RedLock
... 目前 9+ 篇（随缘**更新中**）

《分布式事务 Seata 源码解析》

01. 调试环境搭建
02. 项目结构一览
... 目前 2+ 篇（**努力更新中**）

最近更新《芋道 SpringBoot 2.X 入门》系列，已经 101 余篇，覆盖了 MyBatis、Redis、MongoDB、ES、分库分表、读写分离、SpringMVC、Webflux、权限、WebSocket、Dubbo、RabbitMQ、RocketMQ、Kafka、性能测试等等内容。

提供近 3W 行代码的 SpringBoot 示例，以及超 6W 行代码的电商微服务项目。

获取方式：点“**在看**”，关注公众号并回复 **666** 领取，更多内容陆续奉上。

文章有帮助的话，在看，转发吧。
谢谢支持哟 (*^__^*)

Read more