系统整容纪：责任链设计模式的应用实战（爆灯了，研发工期由45天降为1天）

==========

本文通过介绍使用责任链设计模式的背景和经历，来使得读者加深对于此设计模式的印象，甚至受到一定的启发来对自己当下所参与、所负责的项目进行“整容”，从而提升系统的“美感”。分享工作中的点点滴滴。

一、背景

在下所负责的系统中有这么一个模块，分区模块，直接看这个词的话相信很多人都会疑惑甚至是误解，其实其真正的含义就是“路由”，接下来我简单描述一下何谓“路由”。

相信大家都有过网上购物的经验，每当我们下完订单后，我们都能随时随地的查看订单的物流跟踪状态，而上述的“路由”概念就是指：订单从A地到B地的运输路由线路，例如订单order1要从A运输到目的地F，其可以从A->B->D->F，也可以从A->D->F，至于具体应该走哪条线路，是靠系统中配置的路由以及对应的匹配规则进行筛选出来的。

很长的一段时间里，系统里的路由配置和规则都是静态的（所谓静态就是提前配置好并且几乎是固定不变的），这种做法的弊端很明显，就是成本无法控制，就如同上述所讲的例子，运输路线明明有机会可以减少甚至是可以直送（A地到F地的货物明明可以装满N车，却也不得不按照系统中制定的路线进行运输），但是却被系统中定死的路由规则所限只能多走一些道路，提高了人力作业和运输的成本。

基于此有位大牛发现了降本增效的商机：就是让这块路由线路和规则动起来，使得系统能够更灵活的兼容上述情况，达到资源的极限利用，举个例子：比如有很多订单都是从A要送达目的地F的，系统中静态线路配置的只有A->B->D->F，但是经过系统监测计算发现从A到F的货物量可以装满两车，那此时为这些订单临时生成一条新的线路从A直达F，同时在A场地进行收货、发货等凡是涉及到路由线路匹配的实操环节时，均兼容上此临时路由的场景，这样就能在不改变用户习惯的情况下将整体成本给降下来，并且运输到目的地的效率也得到了大大的提高。

这位大牛提出的方案很好，得到了大家的广泛认可和好评，于是在立项后进入了轰轰烈烈的开发阶段，而在下有幸被委以重任来主导完成此项目的开发交付。

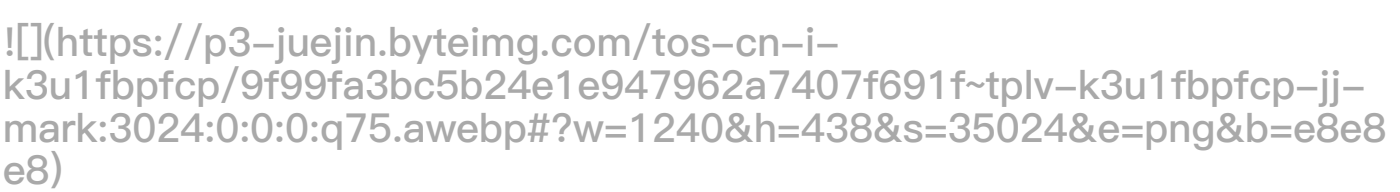
值得一提的是，路由线路这块的改动贯穿了订单的整个实操流程以及一些边边角角的辅助查询、统计报表等功能，场景涉及众多，所以压力还是蛮大的，虽然期间也确实走了不少弯路，但最终的结果是好的，甚至在之后的又来了几个类似变动路由线路的需求，但是基于此次改造后，我们这边都能进行轻松应对，这个在文章后续的效果中会体现一下。

说了这么一大堆，有没有觉得其实都是废话的感觉，哈哈，确实有点啰嗦，接下来让我们来梦回路由，再现一下整个稀碎而又有成就感的改造过程。

二、思路和方法

后续文章中提到的分区是为路由匹配规则的含义

首先来看一张简易的实操流程示意图



在每个场地的实际操作的作业流程中都会涉及到分区匹配规则的情况，同时在一些辅助实操作业的查询功能或报表功能中也会涉及到路由匹配规则，所以一旦分区匹配规则要做变动，那么就会面临如下痛点

首先在业务层面上就会几乎贯穿整个流程，无论是评估、开发、测试等环节都会面临工作量巨大的问题

再次从系统层面上来看，这部分的代码现状也非常不友好，主要体现在：

分区匹配核心业务规则一致，但是代码写法不一，并且分布散乱，让人难以阅读与维护，同时伴随场景遗漏的风险

分区匹配功能目前都是各个实例自行编写并且耦合在各个使用场景中，不具备扩展性，一旦规则变化，改动成本非常高

基于上述痛点，在下决定下定决心将此模块进行重构整治一番，一来通过此次挑战来提高自我、二来也为以后此块规则再次变更的可能做好铺垫，那么具体应该怎么做才能解决上述所提到的痛点呢？我是这么做的

1.业务层面上做好充分的评估：体现在开发的详细设计上（这里由于在下对于业务规则非常熟，所以算是是本人的优势拉哈哈），像场景梳理啊、修改方案、甚至在设计中下沉到了具体功能（按钮点击啊、录入框输入后的回车触发啊等等这些细节）的修改逻辑方案以及代码位置，以便让所有参与开发人员以此为指导手册进行快速开发，测试人员以此为指导进行用例编写、产品人员以此为字典加深其自身对于此块业务的理解等等。听起来是不是很牛X，哇哈哈，这里有点吹嘘拉，本文中不着重介绍这里，主要介绍设计模式哈，咱们缓缓继续往下看

2.这里是我们的重头戏哈，系统层面上主要体现在代码上，毕竟说的再好，落实不到代码，看不到效果都是白搭嘛，何况咱又是职位，好了，废话不多说，继续看我表演哈哈

首先将分区匹配核心模块进行统一的收口，在系统中只保留一个实例进行服务提供，既能解决代码分散维护成本高的问题、又能避免场景遗漏的风险

看到这里或许会有人说，你这个会不会带来另外一个问题啊：虽然场景不会遗漏了，达到了一处代码变动，处处场景都会生效，但是会不会改动了某些场景原有的功能特性？能想到这里的同学，确实很细心哈，如果是硬性收口的话的确是会产生这个新问题，所以统一收口这块一定要支持扩展，预留好钩子便于支持各个场景的差异化处理。这么说比较抽象，举个例子：比如通用场景规则是所有单子都按照系统既定配置的分区规则进行运输，但是现在有一些商家开通了一些快速送到目的地的服务，那对于这些商家的单子就不能用现有的通用规则进行分区匹配运输了，要按照新的规则进行分区匹配从而达到快速运输的目的。这就是里边的差异化。

复用现有的数据结构，增加分区类型，并调整对应的sql和service服务（这里不是本篇重点就不展开讲了），在兼容现有生产逻辑的基础上支持分区规则的扩展

结合设计模式的思想调整分区匹配规则的代码结构：采用责任链模式（这里是本篇文章的重点内容）

那么到底什么是责任链模式呢?

大牛给出的定义：使多个对象都有机会处理请求，从而避免了请求的发送者和接受者之间的耦合关系。将这些对象连成一条链，并沿着这条链传递该请求，直到有对象处理它为止。

结合着现有系统的实际业务来讲讲我是如何进行套用的：开篇背景中已经介绍过现有的分区匹配规则为静态分区匹配（如某某业务点到点、某某业务点到范围区域等等，具体业务细则就不展开讲了，只要知道这里边的匹配规则一大堆即可），现在要新增一种规则支持动态（也是各种匹配，就不展开讲了），这里我把每一种分区规则都定义为一种分区类型，而每一个分区类型都定义为一个分区节点，将这些节点穿成一条链，让每个请求都在这条链中找到匹配自己的线路进行运输。

由于业务细则比较敏感，文章中就不具体透露了，不影响重点设计模式的应用理解

结合定义以及上述分析，那么实际情况到底适不适合使用责任链设计模式呢？在下认为只要能够解决上述痛点并且总体的利大于弊，那么就是适用的。首先使用此模式改造后的优点如下：

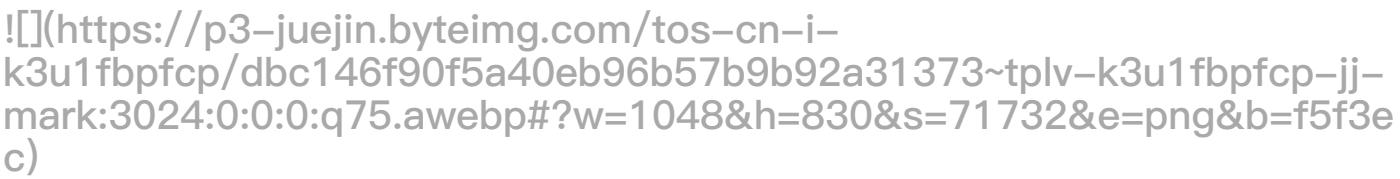
将请求和处理分开了（从与业务实操的耦合中解脱出来，完全不用关心请求是怎么来的，只专注于分区规则的匹配）

提高了系统的灵活性和扩展性（再有新的规则，只需要增加节点即可）

当然了，此模式也有一定的缺点：当责任链比较长的时候，由于每个请求都会遍历整个链条，可能会有性能的问题，同时对于不理解业务的同学调试起来也会感觉比较复杂。

整体看下来，优点是解决了我们当下的痛点，并且利于后续的扩展，而缺点中的性能部分可通过结合模板模式中预留的钩子函数（如果当下请求不适合于当下分区节点规则则跳过）拉最大限度的降低性能问题的影响，同时基于开发人员了解业务也是应当的。那这样看来总体还是利大于弊的。

说了这么多我们先来看看改造前后的分区模块的简单对比示意图吧



虽然图非常简陋，但是其对比含义还是很明显的，改造前：分区匹配和业务处理是耦合在一块的；改造后：分区匹配是一条链并且里边没有业务逻辑处理，从耦合中解脱了出来，也支持扩展。看到这里会有人疑惑：上文中不是说只新增了一个动态匹配规则吗，怎么链中有这么多节点，而且还是两条链。这里我稍作解释下：当下的两条链是经历过许多需求版本变更的，当下看起来区别已是不大，主要是因为其提供的来源业务指定场景不同抽象出来的两条链，互不干扰各自运行，而里边多处本文介绍的那些节点也是后来新增的，直至当下还在系统中使用的节点规则（这也就是文章开篇我提的：万一后边还有规则变更呢。果然还是机智如我，“预言”应验了，在效果中我会讲出这里支持扩展对于缩短工期的重要性）

三、实践过程

相信有不少读者会发现，上面又是说统一收口、又是说结合模板模式最大程度规避性能影响，那你这个责任链一种设计模式是支持不了的吧。

没错，你说对了，大聪明，确实只是使用单一的责任链设计模式远远达不到上文中所说的效果，这里我们也确实将工厂、模板、责任链模式进行了结合使用，工厂用来获取链条的bean，模板用来设置通用方法、方法间的调用以及预留给子类实现的开关、前后置处理、差异化等方法，责任链用来组合各个节点，这里简单抽几个节点将类图展示出来如下

k3u1fbpfcp/c2f1783be390453a8a48c390e071e83f~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1734&h=795&s=404563&e=png&b=2d2d2d)

这段就比较简单了，毕竟就是撸码，上面的类图几乎代码中的实践应用了，也是代码中的核心部分，而在实际调用中均是通过工厂来获取bean链条进行具体分区匹配的。

四、对实践过程的思考和对效果的评价

改造过后的成果主要体现在后续的扩展和维护，就如文章开篇提到的一旦分区匹配规则要做变动就会面临两个痛点，最直接的体现就是在工期上面，第一次接此块的变动新增路由规则需求时，整体实际用的工期就研发侧来说是45天，就更别说一旦遇到BUG，那测试工期也就没保障了（对应上图节点中的动态节点）

但是后来没多久就又再次增加匹配规则的新需求（对应上图节点中的到仓拼车节点），同类的需求，工期缩短近半，优45天降低到了27天，这里还包括此需求中的其他非分区模块的改造，当然了第一版的改造确实有些完美的地方，经过这次需求由进行了优化

![](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/8bf726524bee4401b81f467ea26c357a~tplv-k3u1fbpfcp-jj-

mark:3024:0:0:0:q75.awebp#?w=862&h=140&s=10175&e=png&b=98c091）

接下来的这两个就真正体现到了什么叫做工期消失术，一个是首板分区规则需求，一个是最近B网融合需求中的直发分区规则

直发分区工期2天

首板分区工期就1天

说实话，我在没回看这些数据的时候也没想到有这么大的效果，现在回头一看我也是惊呆了，谁能想到一个设计模式的应用竟能将工期由45天缩短至了1天，这太不可思议了。

当然了其中也还有一些可以改进升级的地方，目前责任链节点的装配都是手动指定的，可改动为自动装配（我在另个业务场景中的改造中已经实现过了，这里也会进行同步改造），再有一个就是要控制节点的数量，如果数量过大则可能需要考虑兼容方案了。

俗话说滴水穿石非一日之功,冰冻三尺非一日之寒，追求强大的工具、新颖的技术固然可行，但是也不要忘了日常工作中的一点一滴的小改动，短时之间可能看不出什么，一旦量变引起了质变，我相信那结果将是非常可观亮眼的。

原文链接: https://juejin.cn/post/7362547943390117897