**摘要:** 【问题】 系统架构风格（System Architecture Style）是描述某一特定应用领域中系统组织方式的惯用模式。架构风格定义了一个词汇表和一组约束，词汇表中包含一些构件和连接件类型，而这组约束指出系统是如何将这些构件和连接件组合起来的。软件系统架构风格反映了领域中众多软件系统所共有的结构和语义特性，并指导如何将各个模块和子系统有效地组织成一个完整的系统。软件系统架构风格的共有部分可以使得不同系统共享同一个实现代码，系统能够按照常用的、规范化的方式来组织，便于不同设计者很容易地理解系统架构。 请以“软件系统架构风格”论题，依次从以下三个方面进行论述： 1．概要叙述你参与分析和开发的软件系统开发项目以及你所担任的主要工作。 2．分析软件系统开发中常用的软件系统架构风格有哪些？详细阐述每种风格的具体含义。 3．详细说明在你所参与的软件系统开发项目中，采用了哪种软件系统架构风格，具体实施效果如何。 【摘要】 2018年3月，本人在公司参加了实时分润系统的开发。该项目主要功能是根据不同的分润计算规则，实时计算每笔交易产生的分润金额，代理商可以通过管理平台查看自己的分润情况，调用第三方代付系统将分润金额代付给代理商。本人在该项目中担任了系统架构师的职位，负责该系统的架构设计。本文以该项目为例，主要论述了软件系统架构风格在该系统中的具体应用和效果。在前置服务对接交易系统的模块中，为了降低系统之间的耦合度，采用了隐式调用架构风格；在分润计算模块中，为了自定义分润计算公式，采用了解释器架构风格；在代理商的管理平台模块中，为了方便使用和易于维护升级，采用了B/S架构风格。通过上述架构风格的使用，最终项目顺利上线，持续稳定运行。

2018年3月，本人在公司参加了实时分润计算系统的开发，因公司的交易量逐日变多，传统的手工计算分润方式变得难以支持，而且还只能隔天计算，做不到实时的效果，这种情况下公司决定建设一套实时的分润计算系统。该项目主要模块有前置服务、分润管理计算服务、代付通道对接服务、代理商分润开放平台服务、报表服务、捡漏服务、监控服务等。前置服务是实时接收并记录公司所有产品交易成功的流水数据，必须保证高可用性和可扩展性；分润管理计算服务是实时计算这些交易流水产生的分润金额，公司目前有多个产品，而且后续还会持续的增多，每种产品线根据自己不同的政策、活动等一系列信息，定义多种不同的分润计算规则，这些规则可以在管理平台商进行管理配置；代理商分润开放平台服务是一个web服务，代理商可以通过平台实时查看自己的分润所得明细以及各种统计类报表数据。分润数据在审核部门审核通过后，调用第三方代付接口将分润金额代付给代理商。本人在该系统中担任系统架构师的职位，负责该系统的架构设计。  
软件系统开发中常用的架构风格有以下几种：隐式调用风格、解释器风格、B/S架构风格、微服务风格、数据流风格等。隐式调用风格：构件之间不直接进行调用交互，而是通过事件的触发或广播来进行交互，大大降低了构件之间的耦合度，方便构件的复用、维护和升级。解释器风格：定义一个解释器引擎，来解释或执行自定义的规则或代码，可以方便灵活的自定义业务，但是方便的同时带来了效率低下的问题。B/S架构风格：通过浏览器取代传统的客户端软件与系统进行交互，方便用户使用，易于服务升级拓展。微服务风格：将大的服务拆分成功能单一的小服务，每个小服务可以独立部署，各小服务之间相互协调配合完成整体的服务。管道—过滤器风格：过滤器即为构件，连接件将多个构件连接形成管道，数据从管道一端流入，依次经过所有的过滤器构件进行处理，输出最终结果。合理的使用架构风格，可以为系统提供良好的解决方案、提高软件系统的重用度、便于不同设计者理解系统架构。  
在实时分润计算项目中，我们综合使用了多个软件架构风格，下文将着重讨论隐式调用风格、解释器风格和B/S架构风格在该项目中的具体应用及效果。  
一、隐式调用风格  
前置服务中，需要接收交易系统产生的交易流水数据，进行入库保存以及后续处理。交易系统在完成交易后其实后续还是会有很多的其他业务需要做的，如果每个系统都是需要通过接口报文通知，或者通过rpc的调用通知，那各个系统之间的耦合度还是比较高的，很容易互相影响，针对这一场景，我们决定采用隐式调用架构风格，来降低服务之间的耦合度。经过调研与测试，我们决定采用activeMQ这个开源的消息中间件来实现，利用其发布订阅的功能实现各系统之间的事件订阅与触发，activeMQ是apache的开源软件，其性能和稳定性还是有保障的。交易系统在完成交易后，把事件发布到activeMQ里，实时分润系统的前置服务通过activeMQ订阅了此事件，在交易完成后会收到activeMQ的通知，然后再继续后续其他的业务，如果后续其他业务场景需要用到交易的事件，只需要通过activeMQ订阅即可实现。通过隐式调用风格的使用，大大降低了系统之间的耦合度，为系统的维护和演化带来了方便。  
二、解释器风格  
分润管理计算服务中，计算分润的规则繁多，有多个产品，每个产品又会根据不同的政策有不同的规则，而且每一个规则可能会频繁的修改，新规则生成的频率也高，如果说分润计算规则直接在代码里写死处理的话，可修改性太差，管理和维护的成本也很高。针对这一问题，我们决定采用解释器架构风格来解决。解释器架构风格具有很高的灵活性和可定制性，完全符合我们的这种业务场景。综合调研和测试后，我们采用了drools规则引擎，相应的分润规则计算逻辑使用drools脚本实现，脚本的内容可以通过管理平台进行记录、查看、修改和日常维护，按照产品、政策等信息分目录存放以方便管理。每笔交易需要计算的时候，会先调用分润路由脚本，根据交易流水对应的不同产品线、政策等信息，选择对应的规则计算脚本名称，然后再读取计算脚本内容，通过规则计算出来最终的分润结果。解释器的运行性能会稍微差一点，但是我们只是用到了计算数值，整体性能可以达到预期效果。通过解释器架构风格的使用，大大增强了我们计算规则模块的灵活性。  
三、B/S架构风格  
代理商管理平台，是一个专门提供给代理商使用的服务平台，代理商可以通过该平台查看自己的实时分润明细、查看各种分润统计报表、维护自己的结算信息等操作。公司目前已有大量的代理商，且分布于全国各地，该服务的推广和维护升级将面临巨大的挑战。针对这些场景以及问题，我们决定采用B/S软件架构风格来解决。页面端我们采用了成熟的html5+javaScript+css来实现前端展示逻辑，相关代码文件都是存放在服务端来进行维护的，为服务升级维护奠定了基础。服务端采用了springMvc、mybatis的分层架构来实现后端业务逻辑，降低了各层次代码的耦合度，也易于维护升级。采用B/S软件架构风格后，用户只需要打开浏览器即可使用该平台服务，不用安装本地软件，简化了用户的使用成本。因为所有的页面展示逻辑、业务逻辑都在服务端进行控制维护，如果说服务需要升级，只需要服务端进行更改发布就好，用户端完全是零成本，这样子大大减少了升级维护的成本。  
该项目于2019年3月完成验收与上线，一直持续稳定运行。解释器架构风格的使用给后期分润规则的维护与扩展带来了巨大的方便，后期的分润计算规则在持续进行老规则的修改和新规则的增加，我们只需要在管理平台上操作配置即可，充分发挥了解释器架构风格的灵活性。上线运行后代理商管理平台又经历了几次的功能改造，因为我们采用了B/S的软件架构风格，代理商管理平台的维护和升级还是很容易进行的。  
当然，在我们设计过程中，也存在一些问题和不足，比如代理商管理平台查看统计报表，因为每次都去数据库实时查询，导致数据库压力巨大，后期我们进行了优化改造，每天凌晨定时跑批把结果存起来，方便代理商可以更快的查看报表数据，但是针对当天的数据，因为数据量不大，还是实时的查数据库进行展示。