22.图解支付渠道网关（二）：模型、状态机与流程编排_V20240125

1. 前言
2. 分层 - 简单而有效的架构思想
3. 领域模型
   3.1. L0级别渠道网关领域模型
   3.2. L1级别渠道模型
   3.3. L1级别单据模型
4. 状态机
   4.1. 业务单
   4.2. 流程单
5. 流程编排与流程引擎
6. 结束语

在《图解渠道网关（一）：不只是对接渠道的接口》那篇文章中，介绍了渠道网关的定位，常见渠道类型，产品架构，系统架构等内容。

*今天这篇文章中，主要讲清楚如何抽象定义渠道模型，各种单据模型，状态机设计，流程引擎，以及如何定义一种简单好用好理解的流程脚本等。不过最重要的是给出一个示例：**如何从这些繁复各异的渠道中找出规律，做好抽象，以不变应万变**。里面涉及的一些架构分析思想，对于提升通用架构能力也是非常有益的。*

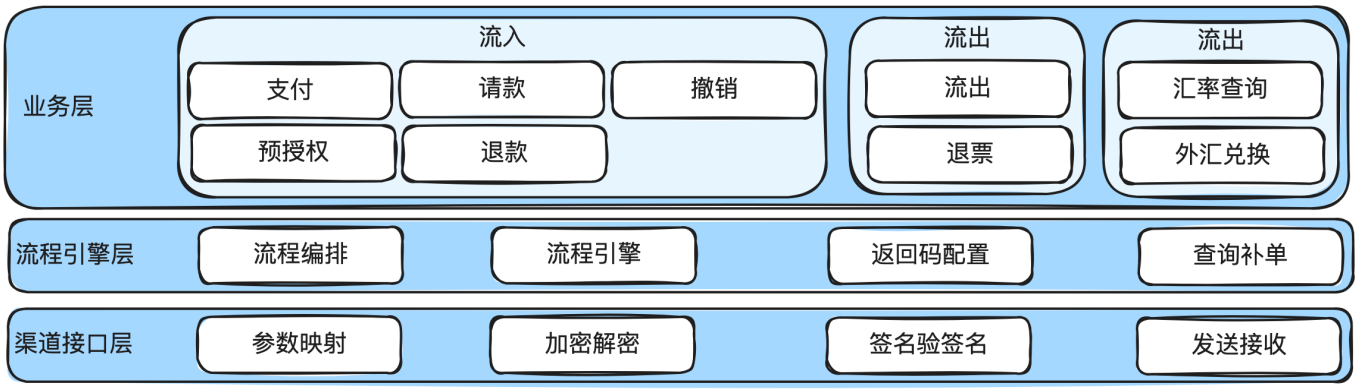
1. 前言

外部渠道是支付系统的资金来源与出口，所有的支付系统都需要对接各种各样的外部渠道，比如银行、第三方支付、外汇等，这些外部渠道形态各异，接口各异，流程也各不相同。

个人经历过好几种渠道接入的方式，最差也是最原始的方式，就是每来一个渠道就硬编码，各自if else满天飞，各渠道间完全无法复用，接入成本高不说，维护成本也高得离谱。

今天，我们深入到渠道网关的核心——如何在多样化的支付渠道中找到共性，实现有效的抽象，以应对不断变化的需求。这不仅是支付系统架构设计的挑战，也是提升通用架构能力的重要课题。

2. 分层 – 简单而有效的架构思想



分层设计思想在软件架构设计中无处不在，已经到了日用而不自知的地步。在渠道网关的设计中，也需要用到这个最基础的架构设计思想。

整个渠道网关，通常可以分成三层：**业务层，流程编排层，接口对接层**。

首先是业务层，负责处理上游的业务，对上游来说，渠道网关承担的核心业务就三个：

流入：通过渠道把用户的钱扣到平台的备付金账户。细分有：支付、预授权、请款、退款、撤销等。

流出：通过渠道渠道把平台的备付金账户的钱给到用户，或者平台内部各账户进行调拨。细分有：流出，退票。

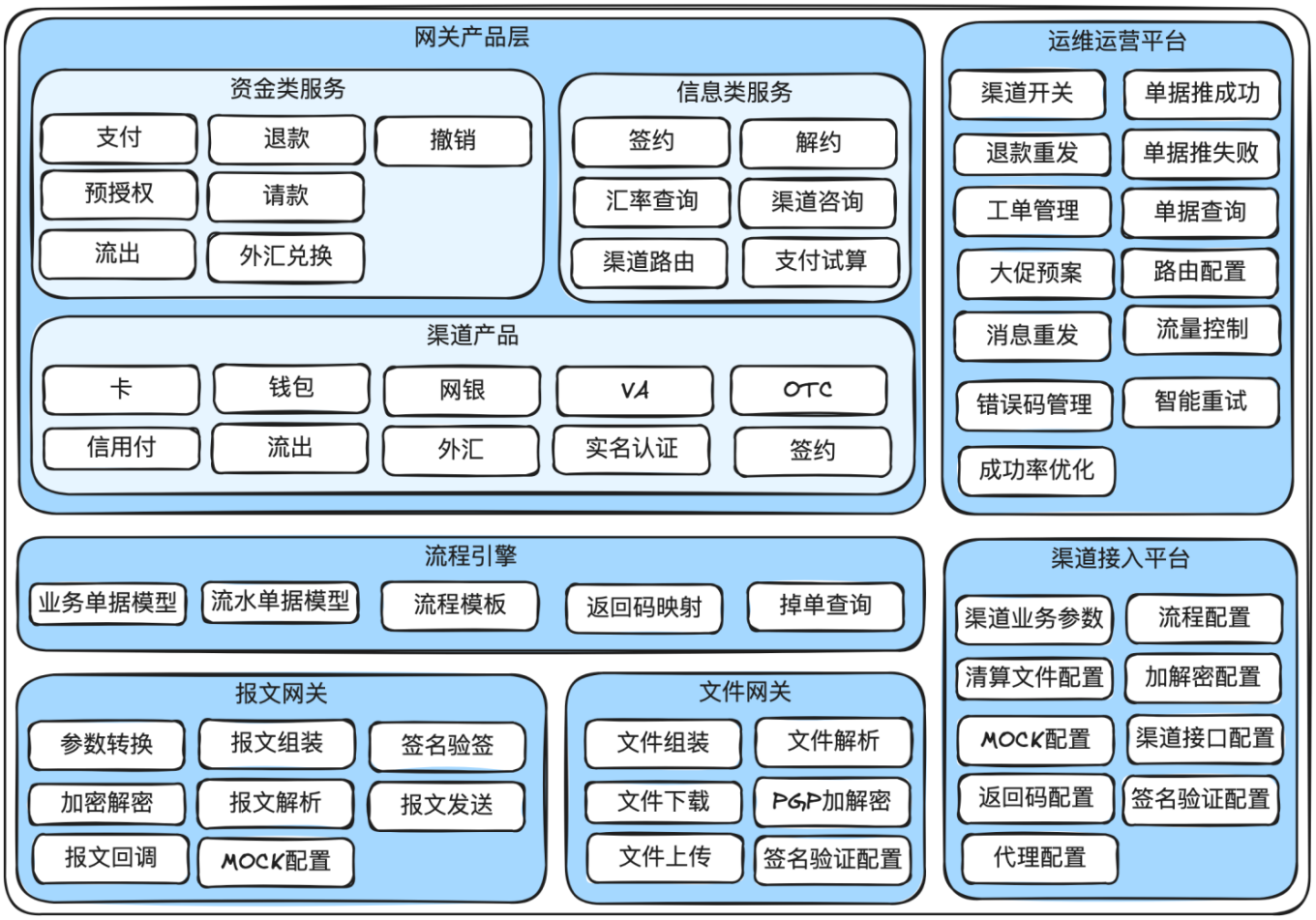
外汇：查询汇率，再做货币兑换。

对于业务层来说，只需要把上游的请求收下来，再往下发流程引擎层，然后等待流程引擎处理结果的推进。

接下来是流程引擎层。为什么要有流程引擎层？因为大部分渠道的流程是不一样的，比如有些渠道，只需要请求渠道一次，就可以完成支付，但是有些渠道可能要请求3次才能完成支付：1）刷新TOKEN；2）换取支付TOKEN）；3）发起真正的支付。而且还可能请求超时，这时需要查询补单。这些操作如果和上面的业务层耦合在一起，将会变得非常复杂。

再往下，**就是接口对接层**，通常一个流入渠道有8到12个接口，比如签名，解约，支付，退款，支付查询，退款查询，支付通知，退款通知，撤销，刷新TOKEN，换取TOKEN，清算文件等。不同的渠道提供的接口不同，报文、签名签验、加密解密等也都不一样。

把上面分层思想完整实现出来，就是我们在*《图解渠道网关：不只是对接渠道的接口（一）》那篇文章中提到的系统架构图，如下：*



其中报文网关和文件网关就是渠道接口对接层的落地。

3. 领域模型

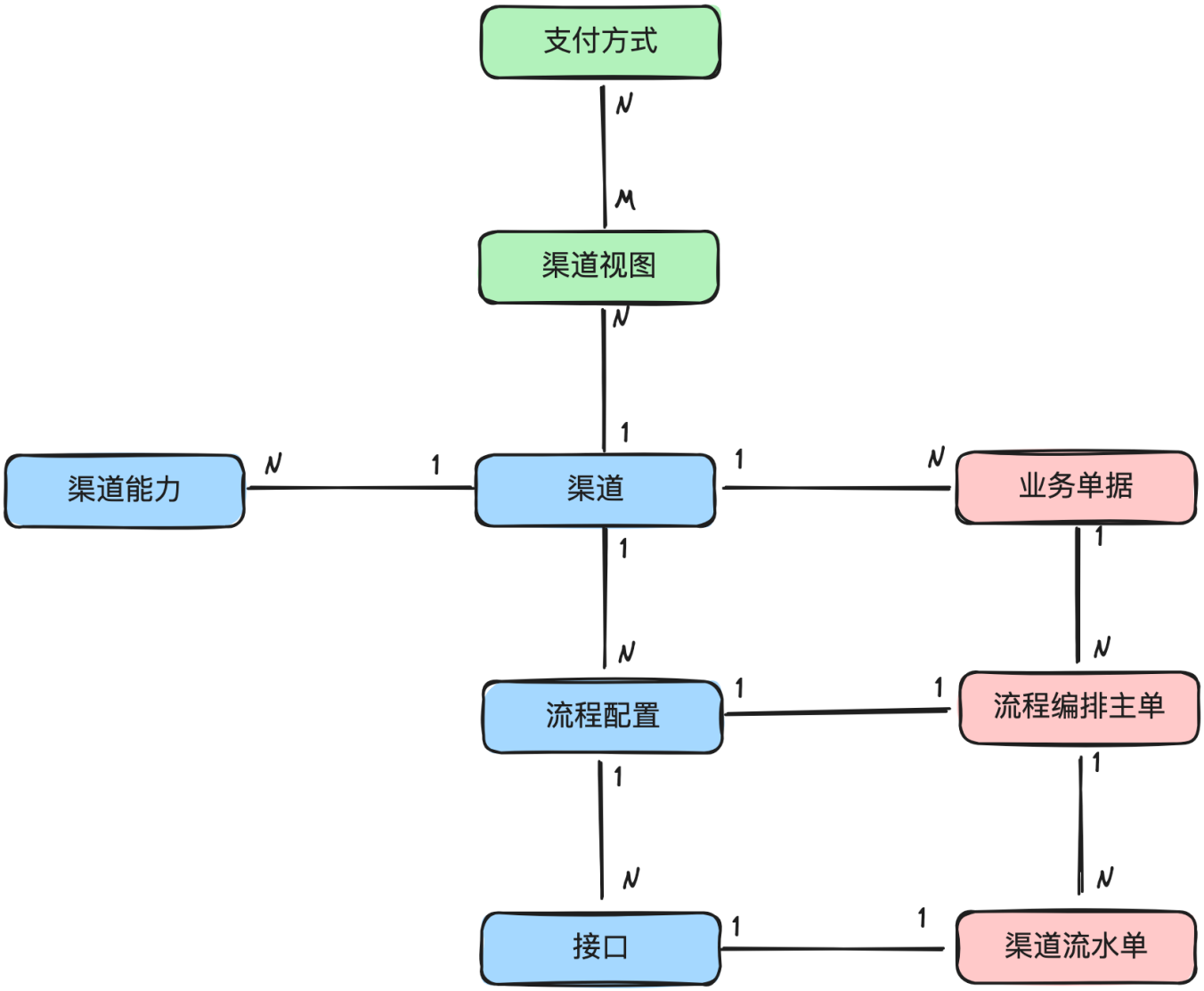
从上面“业务、流程编排、接口分离原则”导出的分层模型，我们可以看到，每层都是自己的领域模型，但是需要有一根主线牵起来，那就是渠道模型。

所谓渠道模型就是我们对渠道能力的抽象，比如我们可以抽象为：网银、第三方钱包、卡、VA、OTC等，也可抽象为代扣、快捷支付等，还需要把支付币种，限额，退款有效期等业务属性也抽象出来。

下面我们一个个说。

3.1. L0级别渠道网关领域模型

以流入（支付）渠道（通道）为例说明。流出、外汇的思路是差不多的。



补充说明：

渠道：对外部的支付渠道（或者也叫支付通道）进行逻辑抽象，比如支付宝、微信支付、银联，国外的WPG，MGPS等。

渠道能力：把外部渠道提供的基础能力进一步抽象出我们需要的能力，比如支付能力，退款能力等。

流程配置：不同的渠道流程可能是不一样的，有些渠道支付只需要调用渠道一次，有些需要调用多次。

接口：渠道提供的原始接口能力。

渠道视图：我们对渠道配置了很多参数，但是有些参数对于渠道咨询或渠道决策不是必须的，所以再抽象出一个渠道视图，用于渠道咨询或渠道决策。

支付方式：对用户可见的支付方式，可以做各种个性化的定义，和渠道视图没有一对一的映射关系。

业务单据：比如支付、退款、请款等都是业务单据，依赖流程编排主单来推进状态。

流程编排主单：用于流程引擎的推进。每个业务单据需要对应一个流程编程编排主单。依赖渠道流水单的状态来推进状态。

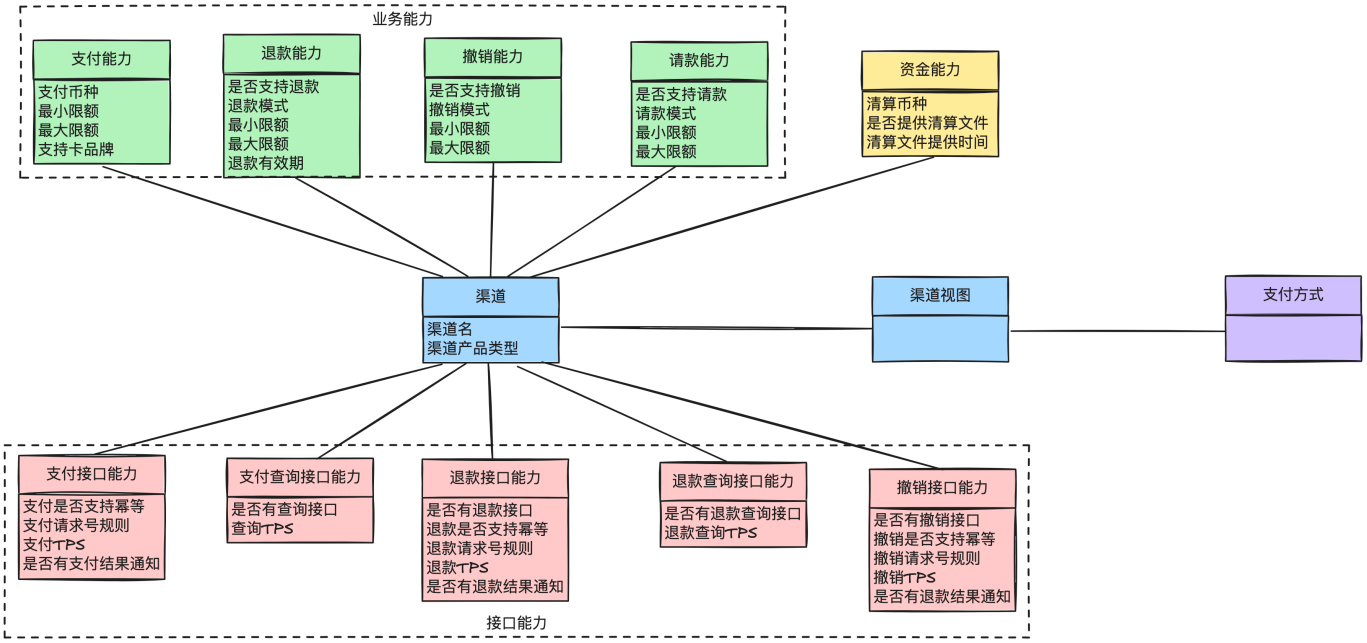
渠道流水单：每次和渠道交互，都生成一个独立的流水单，比如一笔支付，需要调用渠道的支付接口，还可能有多次查询补单，也可能有渠道异步通知回来，每一次交互就会生成一笔流水单。多笔渠道流水单对应一笔流程编排主单。

3.2. L1级别渠道模型

外部渠道定义或提供的能力各有差异，我们需要做渠道提供的基础能力做一个适配于我们内部业务的抽象。

以支付渠道为例，可以抽象为：业务能力，资金能力，流程编排参数，接口能力等。业务能力包括支付、退款、撤销等能力。支付能力又可以拆分为支付币种、限额、是否支持冪等，退款能力又可以拆分为是否支持退款，退款有效期，退款限额等。

下面是一个经典的支付渠道抽象：



说明：

渠道：对外部渠道做一个抽象，比如国内微信、支付宝、银联等，国外的WPG，MGPS等。

业务能力：支付、退款、撤销、请款等能力详细描述。比如退款有效期，最小限额等。

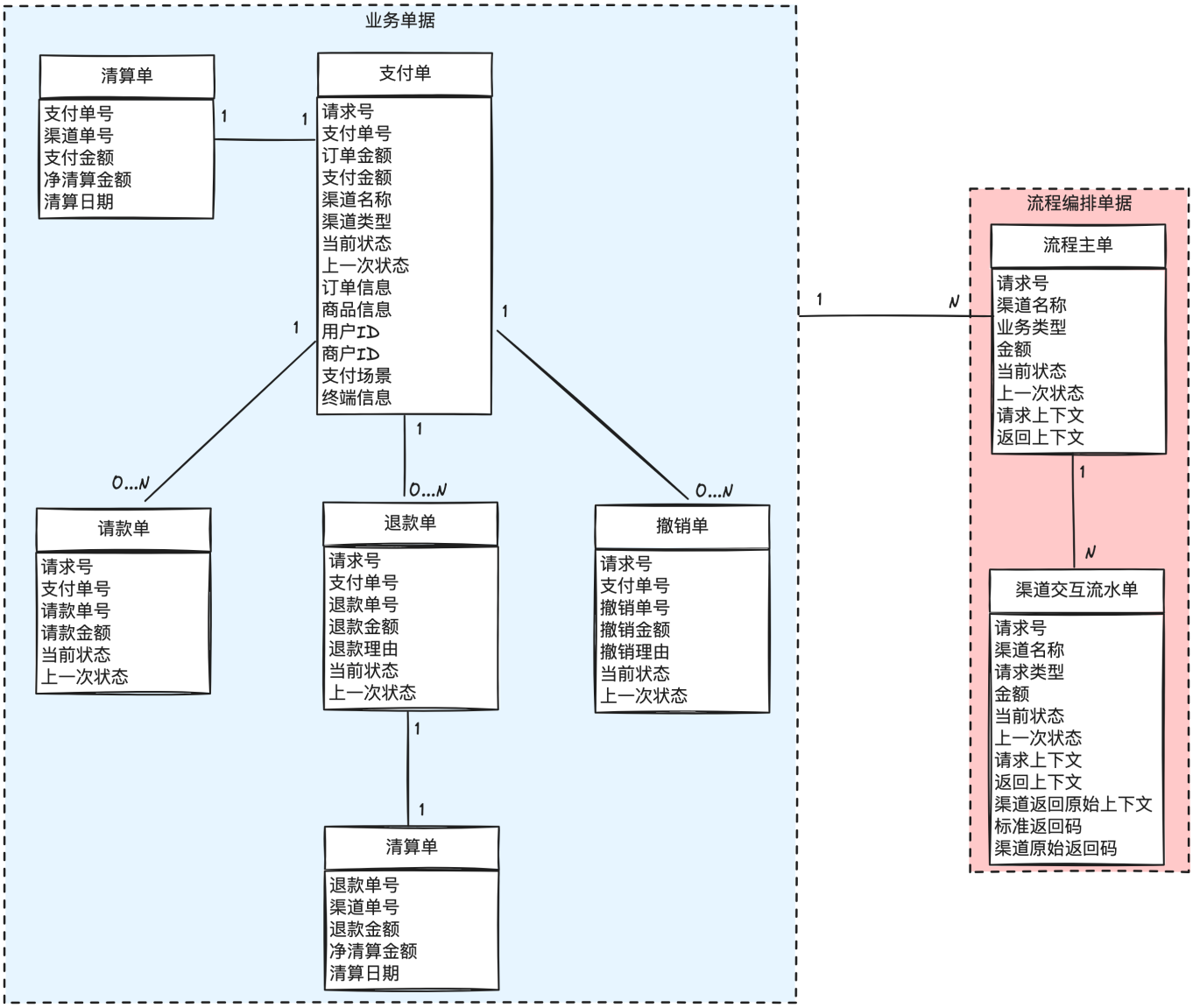
资金能力：清算币种等。

接口能力：描述接口本身的能力，比如渠道的请求号生成规则。特殊情况下可能有短号问题。

图中的渠道视图和支付方式在“L0级别渠道网关领域模型”章节中已经说明。

3.3. L1级别单据模型

因为使用了分层架构，上面的业务层只负责业务逻辑的推进，下面的流程引擎负责与渠道交互的流程推进。所以单据也分为2种类型：业务单与流程编排单。



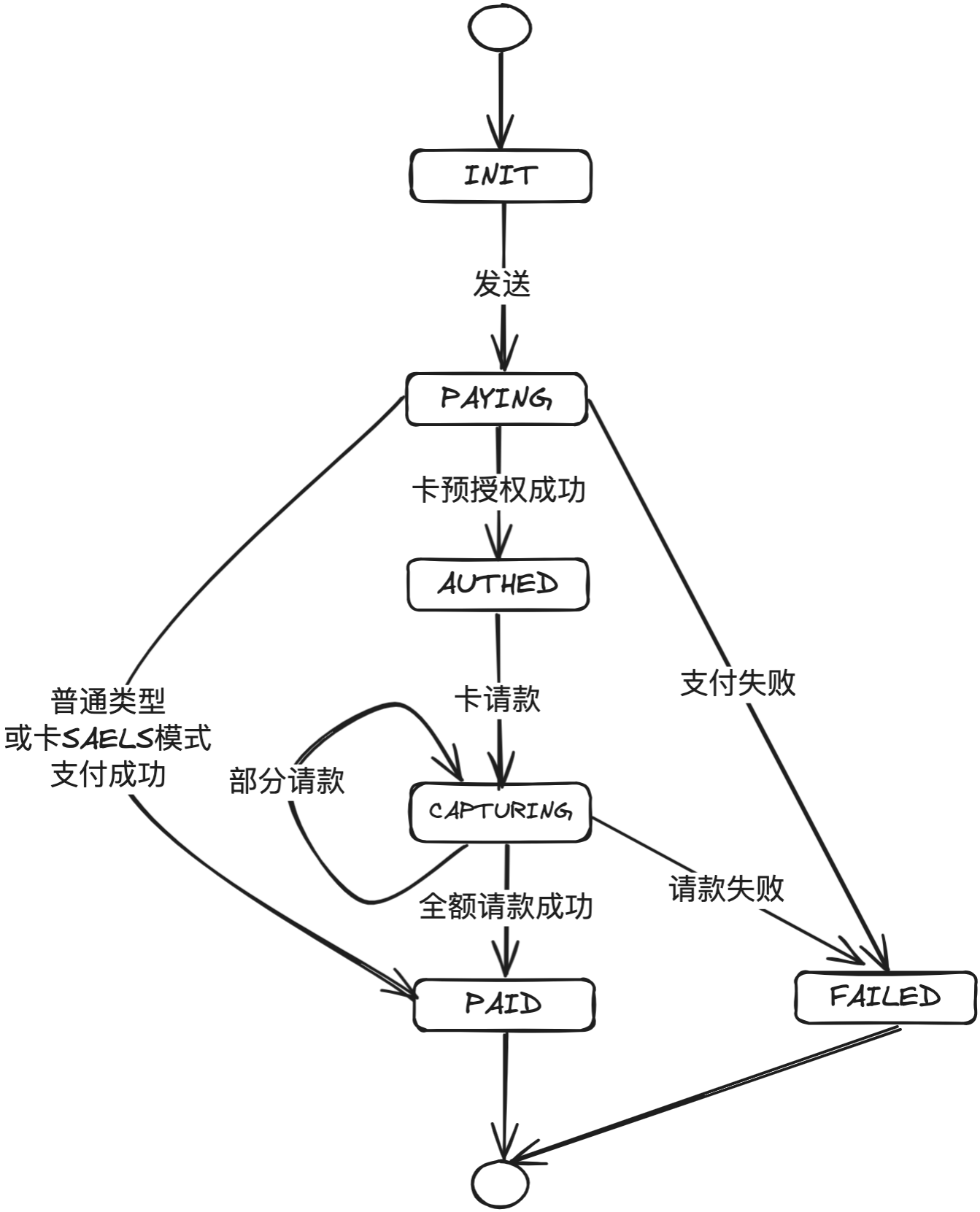
业务单：比如支付、请款、退款、撤销等，是直接承载业务属性的，比如退款时要检查当前的可退金额等。

流程编排单：只有主单和渠道交互流水单两种，承载流程编排。在流程编排单里面，不会做太多的业务校验，只是按预先的脚本配置推进流程，第一步调用什么，第二步调用什么，根据结果码推进状态等。在后面的“流程编排与流程引擎”有详细介绍。

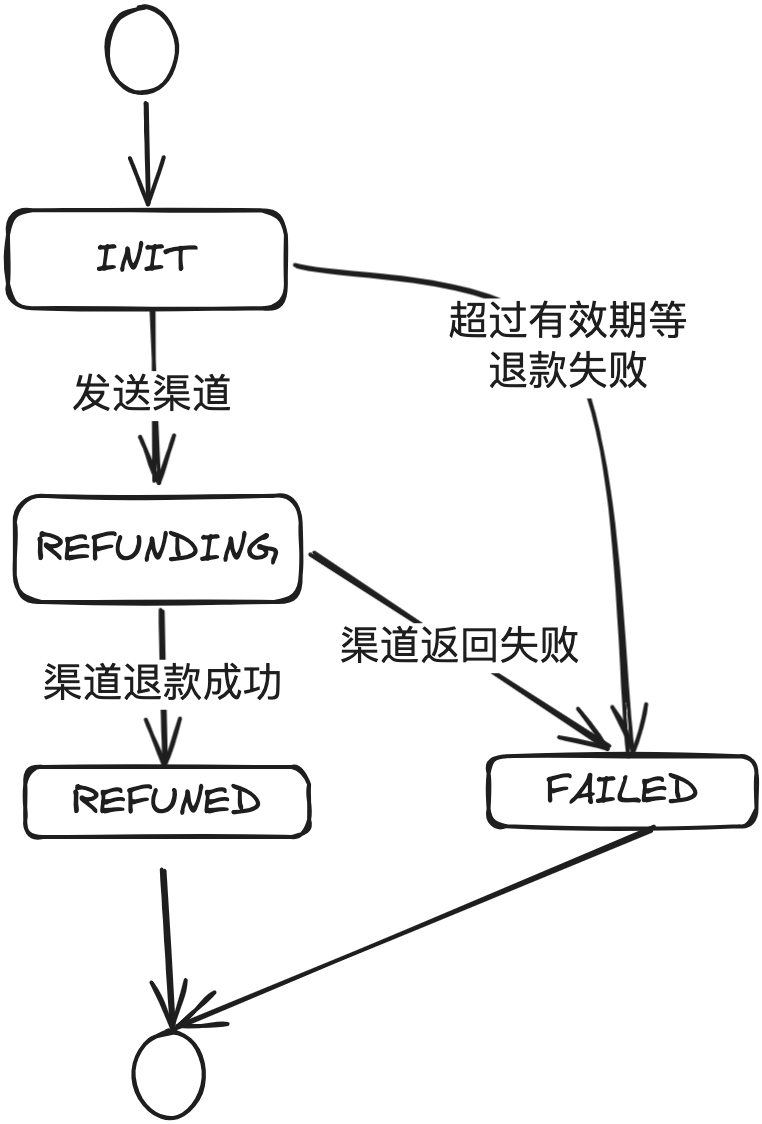
4. 状态机

4.1. 业务单

支付：只有卡支付的预授权模式，才会有预授权成功和请款中状态。其它的支付没有这两个状态。



退款：如果超过退款有效期，直接置为“退款失败”，所以有INIT直接到FAILED的情况。

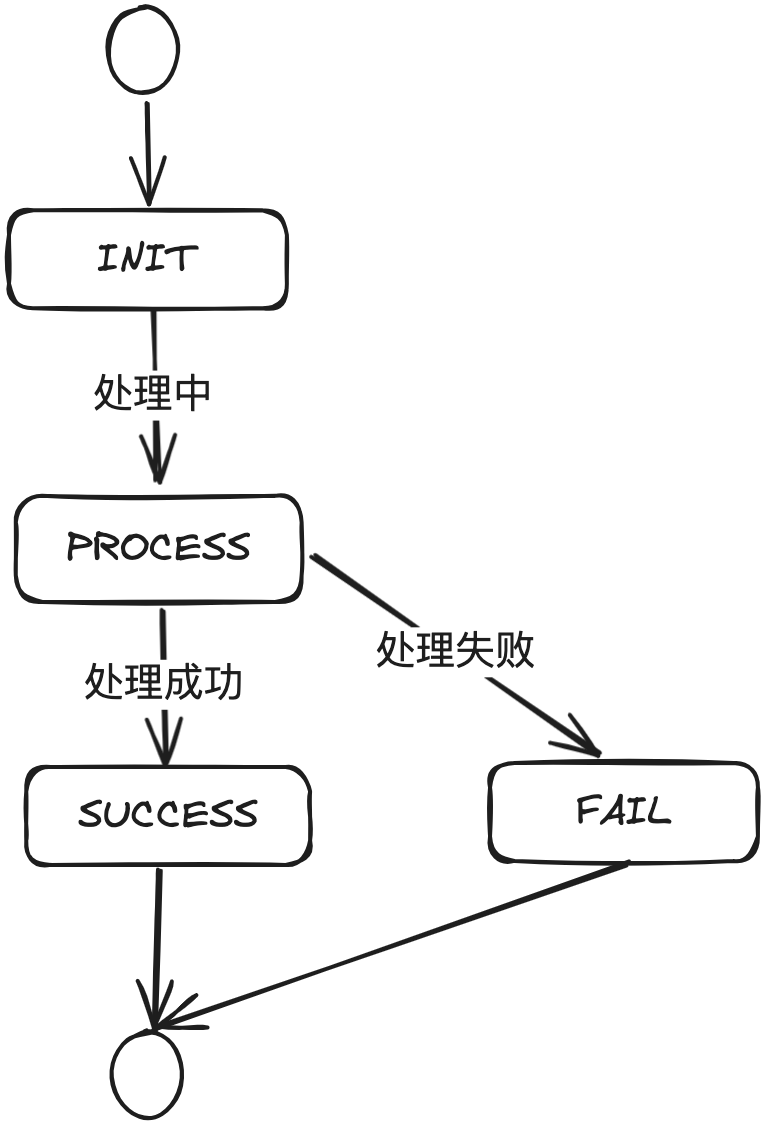


撤销单和退款单比较像，略。

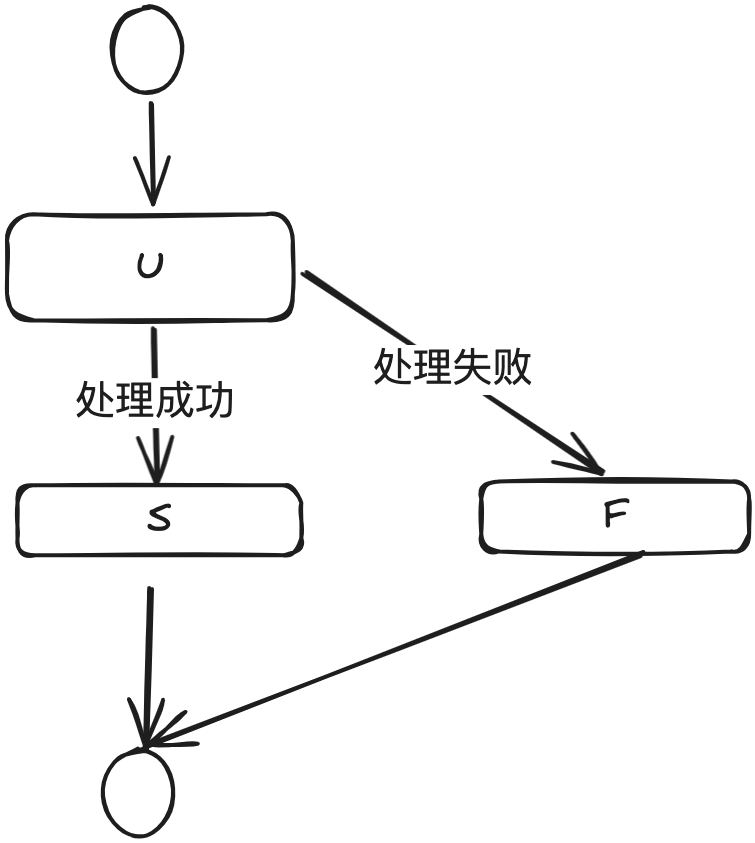
4.2. 流程单

对于流程单来说，主单只有：初始，处理中，成功，失败，流水单则只有：未知、成功、失败。原因在于流程单并不需要感知业务状态。

流程编排主单：初始，处理中，成功，失败。



渠道交互流水单：未知，成功，失败。



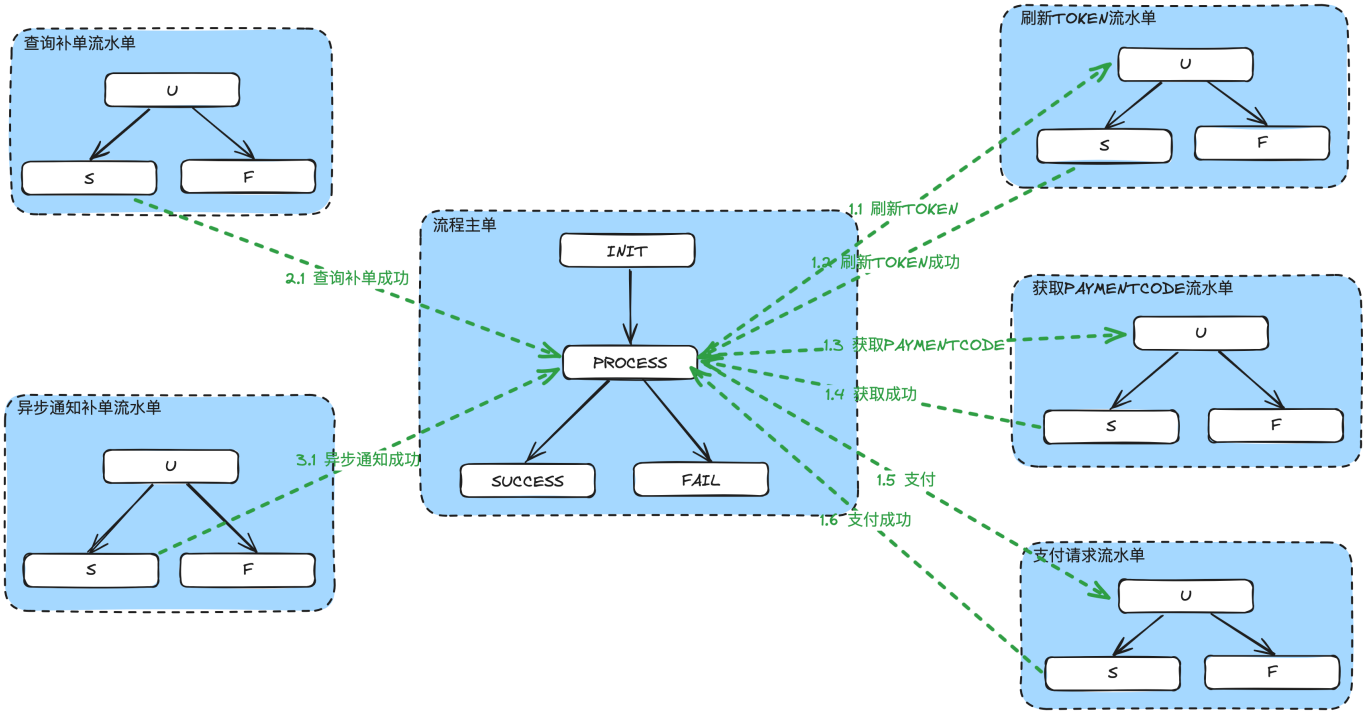
5. 流程编排与流程引擎

前面有说到，对于渠道交互来说，有些渠道的支付只需要一次交互，有些渠道的支付需要三次交互，这些差异由流程编排层来吃掉。上层的业务层只负责业务状态的推进。

流程引擎有很多成熟的方案，而且很多都声称自己的是轻量级的，比如**Activiti，JBMP，**大家可以根据自己的需要来选择。

不过对支付流程来说，仍然太重了，且大家都是做技术的，什么轮子都想自己造一个，对吧，我们也不例外，经历过的几家公司都是自己造的自己的流程引擎轮子，哈哈。

先举个稍微复杂的例子：渠道需要3次交互（刷新token，获取paymentCode，最后支付），提供查询和异步通知接口。主单和流水单的推进大概如下：



图画得不太好，大家将就着看。

然后我们定义一种自己的语言来描述上面的流程，其实很简单，也很清晰，如下：

```java
public PayNeedRefreshTokenAndGetPaymentCodeFlow implements SimpleFlow {

    /**
     * 支付流程，创建后，先推进到处理中，再刷新TOKEN，获取PAYMENTCODE，最后支付
     * 主单的状态为：INIT，PROCESS，SUCCESS，FAIL
     * 流水单的状态为：U，F，S
     */
    @Override
    public void config(FlowConfig config) {
        // 创建后，通过事件CREATE推进到处理中
        config.sourceStatus(INIT).on(CREATE, PROCESS)
        // 然后请求刷新TOKEN，如果失败，就推进流程主单到失败
        .request(refreshToken, subProcess().when(F).transTo(FAIL)
             // 刷新成功，就获取支付码，如果失败，就推进流程主单到失败
             .when(S).request(getPaymentCode, subProcess().when(F).transTo(FAIL)
                  // 获取支付码成功，就支付
                  when(S).request(pay, subProcess().when(S).transTo(SUCCESS).when(F).transTo(FAIL))));

        // 支付回调通知
        config.sourceStatus(PROCESS).callback(payNotify, subProcess().when(S).transTo(SUCCESS).when(F).transTo(FAIL));

        // 定时任务梯度查询补单
        config.suoreceStatus(PROCESS).query(payQuery, subProcess().when(S).transTo(SUCCESS).when(F).transTo(FAIL));
    }
}
```

如果上面的比较复杂，我们看一下简单的普通支付，如下：

```java
public PayCommonFlow implements SimpleFlow {

    /**
     * 支付流程，创建后，先推进到处理中，然后支付
     * 主单的状态为：INIT，PROCESS，SUCCESS，FAIL
     * 流水单的状态为：U，F，S
     */
    @Override
    public void config(FlowConfig config) {
        // 创建后，先推进到处理中
        config.sourceStatus(INIT).on(CREATE, PROCESS)
        // 支付，如果失败，就推进流程主单到失败，如果成功，就推进主单到成功
        .request(pay, subProcess().when(S).transTo(SUCCESS).when(F).transTo(FAIL));

        // 支付回调通知
        config.sourceStatus(PROCESS).callback(payNotify, subProcess().when(S).transTo(SUCCESS).when(F).transTo(FAIL));

        // 定时任务梯度查询补单
        config.suoreceStatus(PROCESS).query(payQuery, subProcess().when(S).transTo(SUCCESS).when(F).transTo(FAIL));
    }
}
```

简单说明：

config.sourceStatus(STAUTS)：起始状态。

on(EVENT, STATUS)：通过事件推进到新的状态。

request(action, subProcess().when(status).transTo(STATUS)：请求动作，比如请求渠道支付，再根据结果推进主单的状态。

callback(action, subProcess().when(status).transTo(STATUS)：回调动作，根据渠道异步通知的结果推进主单的状态。

query(action, subProcess().when(status).transTo(STATUS)：查询动作，根据查询结果推进主单的状态。

接下来就是，如何构建一个流程引擎，能解析并执行上面的脚本，只需要写几个类就可以了。

通过上面的分析，我们可以看到，和外部渠道（支付通道）再复杂的流程，只要分解一下，就很简单，就那么几个步骤组合，然后写一个好理解的脚本，加几个类就可以处理得相当妥当，既不需要引入Activiti那么复杂的流程引擎，也不用写一堆不好理解的XML配置文件。

再补充说一点，**流水单怎么判断是推进到F（FIAL失败）还是S（SUCCESS成功），还是啥事不干的U（UNKNOW未知）呢**？很简单，就是通过渠道返回码的映射。明确渠道明确场景明确返回码映射到指定的状态就行。这个很简单，就不废笔墨了。

6. 结束语

今天主要讲了如何抽象定义渠道模型，渠道网关内部核心的领域模型，状态机设计，流程引擎，一种还不错的流程脚本定义等。

不过这些都是次要的，**最重要的是我们要想办法探寻事物的本质是什么，再做适当的抽象，复杂的事情要想办法往简单地做**。

比如渠道千千万万，我们只需要抽象我们需要的能力，形成自己的渠道能力模型。渠道的交互千差万别，我们只需要使用分层的架构思想，让各层各司其职，瞬间就简单了很多。当看到需要流程编排时，也不用想着立马要引入各种流程引擎框架，或许加几个简单的类就可以搞定。

渠道网关的解构暂时先告一段落。后面有时间再补上报文网关、文件网关、渠道路由等这些核心部件的解构。

这是《百图解码支付系统设计与实现》专栏系列文章中的第（22）篇。和墨哥（隐墨星辰）一起深入解码支付系统的方方面面。

欢迎转载。

Github（PDF文档全集，不定时更新）：https://github.com/yinmo-sc/Decoding-Payment-System-Book

公众号：隐墨星辰。

有个小群不定时解答一些问题或知识点，有兴趣的同学可先加微信（yinmo_sc）后进入，添加微信请备注：加支付系统设计与实现讨论群。