* [首页](https://km.woa.com/?kmref=km_header)
* [发现](https://km.woa.com/discovery?kmref=km_header)
* [悦读](https://km.woa.com/read?kmref=km_header)
* [乐问](https://km.woa.com/q?kmref=km_header)
* [直播](https://tencent.lexiangla.com/lives?company_from=tencent&kmref=km_header)
* [应用](javascript:void(0))
* [我的K吧](javascript:void(0))
* [[](https://km.woa.com/user/rickyrqzhao)](https://km.woa.com/user/rickyrqzhao)

窗体顶端



窗体底端

[](https://km.woa.com/group/wxgTEC)[WXG技术能力提升](https://km.woa.com/group/24938)

* [首页](https://km.woa.com/group/24938)
* [技术好文](https://km.woa.com/group/24938/articles)
* [T族开放技](https://km.woa.com/group/24938/events)
* [T族日历](https://km.woa.com/group/24938/calendars)
* [精选文集](https://km.woa.com/group/24938/knowledges)

目录

一、背景

2.1、拆迁者模式

2.2、绞杀者模式

2.3、修缮者模式

2.4、我选择什么模式

三、确定好迁移顺序

3.1、结合服务内部依赖和功能特点分析服务的风险指数

3.2、服务依赖方越重要风险指数越高

3.3、服务内不同接口风险指数也可能不一样

3.4、服务的风险指数综合决定迁移的优先级

四、如何正确重构

4.1、开发正确的逻辑

4.1.1、协议对等迁移

4.1.2、业务逻辑的对等迁移

4.1.3、风险可控的小范围改造

4.2、测试保证正确性

4.2.1、质量测试

4.2.2、导流验证

4.3、灰度上线

4.3.1、按业务灰度

4.3.2、安商户号灰度

五、如何迁移调用方

5.1、业务方自行更换接口

5.2、统一重置C++客户端

5.3、统一调整PHP适配器

5.3.1、PHP迁移如何处理新旧差异

如何重构有一定历史的服务

* [widyhu](https://km.woa.com/user/widyhu)

* 2021年09月04日 12:21

* 浏览(431)

* [收藏(39)](javascript:void(0);)

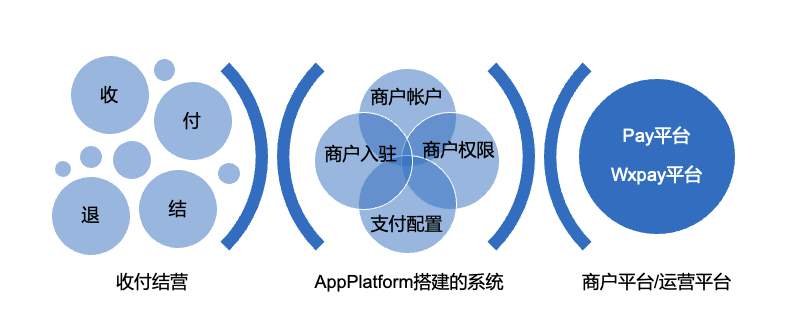
* [评论(3)](javascript:void(0);)

* [分享](javascript:void(0);)

| 导语 本文以appplatform迁移为主线，阐述了一种具有一定历史背景的服务的重构方式。文章基于业界常用的拆迁者、绞杀者、修缮者三种重构模式的特点和优缺点，综合appplatform迁移的实际情况选择合适的迁移方式；基于待迁移系统不可用会影响业务的特点来确定各服务的迁移顺序，从而取得最大迁移价值；然后阐述了在开发、测试和灰度环节如何保证迁移的正确性；最后阐述了如何提高迁移调用方的效率。

**一、背景**

早期微信支付为了匹配开发人员的习惯，快速搭建微信支付的商户体系，使用了appplatform框架搭建了很多服务，这些服务包含有商户帐户、权限、入住、支付配置等，总计43个服务，这些服务渗透到了微信支付收付退结营的各个业务流程中；到后来因运维和开发的流失，appplatform面临无法更新版本，线上服务无法进行正常的日常维护的窘境，这些服务一但出现故障，微信支付很多业务将面临不可用，因此appplatform是微信支付一只巨大的灰犀牛。



通过统计分析，这43个服务的现状如下：

（1）总计代码行数超80w行（不包含生成代码）

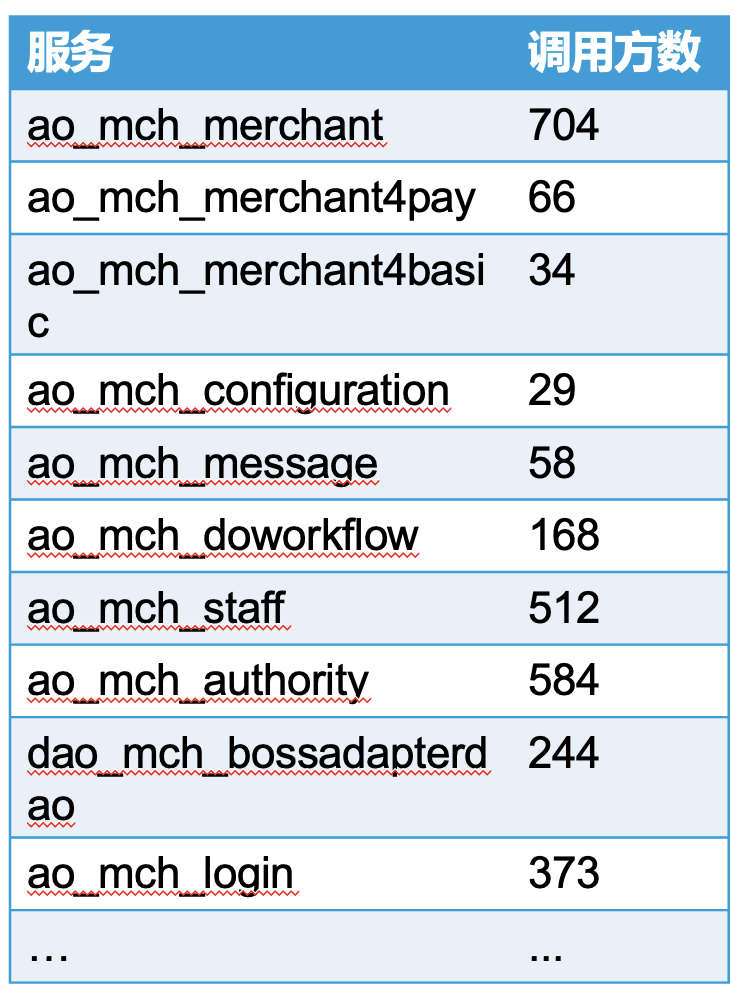
（2）全是2b的业务，业务复杂性较高

（3）这些服务被包括支付、付款、退款、结算以及营销等业务广泛依赖，依赖性项达3544项

（4）各个业务一直都在正常持续向前发展，2019年总计提交代码次数达346次

面对不能发版本，不能进行线上维护的情况，这些appplatform服务随时都有可能出现重大事故，若微信支付是一辆快速飞奔的列车，appplatform可能就是他不怎么好使的轮子，能用但随时都有可能脱轨，我们需要做的是在飞速奔跑的列车下平稳地将这些不怎么好使的轮子更换掉，使其更稳更快运行。也就是把这些appplatofrm服务给重构掉。那么我们怎么去重构呢？

以下表格是部分服务的调用方数



**二、业界一般如何重构历史服务**

业界对复杂系统进行重构常用处理模式有拆迁者模式、绞杀者模式和修缮者模式三种。我们可以分析业界的方法，再结合我们的业务特点选择合适的重构方式。

**2.1、拆迁者模式**

拆迁者模式是指单独组建一波人，完全实现一个新的系统，用于替代旧的系统，新系统开发和替换期间，两套系统都运行，直到新系统完全替代旧系统之后下线旧的系统。

特点：完全重构一个新系统，用于替代旧的系统；

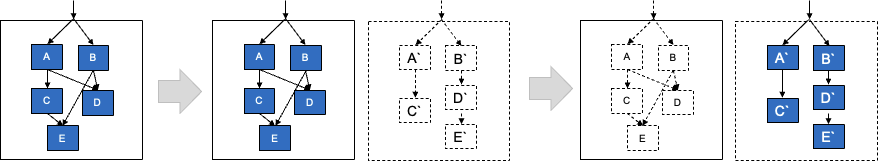
好处：

（1）新系统与旧系统没有瓜葛，彻底解决历史包袱；

缺点：

（1）新系统中业务需求容易遗漏；

（2）期间两套系统需要维护，人力消耗较大



**2.2、绞杀者模式**

绞杀者模式则是开发新需求的时候，新建一个模块，然后将原来旧的模块替换掉，依此反复直到重构完所有的模块

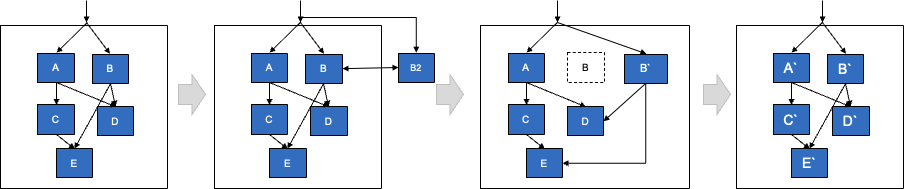
特点：在遗留系统之外，构建新服务，并且使用新服务，保持原有的遗留系统不变  
优点：

（1）能完整还原有需求；

（2）系统可稳定提供价值  
缺点：

（1）重构时间跨度很大；

（2）系统有一定的迭代成本，会影响交付率



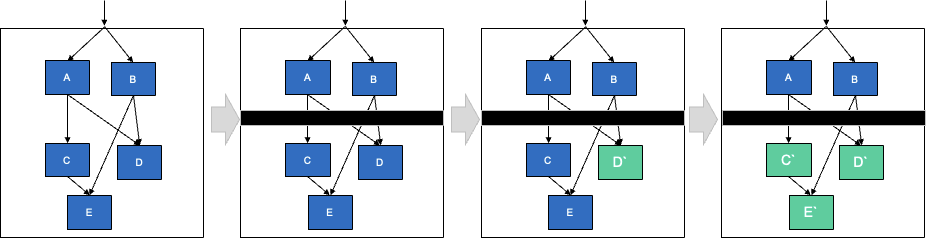
**2.3、修缮者模式**

修缮者模式则是将系统内部隔离出部分，通常是需要修缮的部分，然后对这部分进行抽象、重构和替换

特点：修缮者模式改造只会在系统的内部，不改变系统的外在表现  
优点：

（1）能完整还原有需求：

（2）系统外部无感知  
（3）重构与业务需求随时可切换支持  
缺点：  
（1）重构时间跨度很大  
（2）技术上有比较大的迭代成



**2.4、我选择什么模式**

对于这三种重构模式，对比完这些模式的优缺点之后，来看看我们业务的特点：

（1）我们需要从appplatform迁移到svrkit，接口定义需要调整，不能在服务内部进行重构；

（2）其次appplatform服务无法进行版本更新；

（3）再者业务也需要持续发展，需求不会中断；

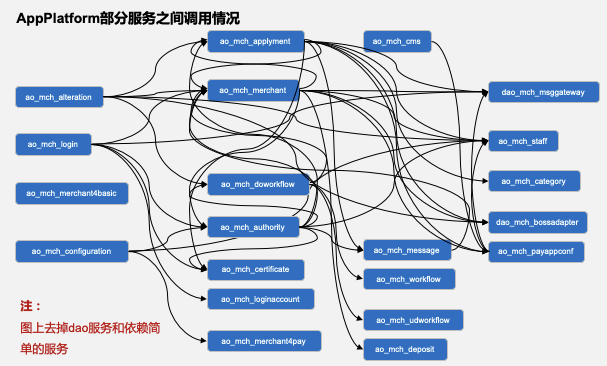
综合这三种模式的特点，我们比较适合的是绞杀者模式，但是我并不想等到需求来的时候再去重构对应的模块，那样会影响需求交互效率，所以也参考了修缮者模式的优点，主动从系统中圈出需要重构的部分出来重构，直到重构完成，所以我选择的是技术驱动型的绞杀者模式。



**三、确定好迁移顺序**

分析好待迁移服务的内部和外部依赖关系，给每个模块设定一个迁移价值，执行时可以从价值高的开始，从而做到迁移价值的最大化；价值这个值根据业务系统的特征可以选不同的维度，如本次appplatform迁移，主要是为了应对无法更新版本、无法正常维护的情况，迁移哪个服务能快速减少appplatform的不确定性对关键系统的威胁将是一个及其重要的参考点，因此快速降低风险的一个重要指标，们从待迁移服务的内部关系、调用方、接口功能三个方面进行分析，识别各服务的风险指数，最终确定迁移顺序。

**3.1、结合服务内部依赖和功能特点分析服务的风险指数**



上图是抽取部分服务画的系统调用的关系网图，通过对内部依赖关系进行分析，我们将服务划分成了三大类，分别是协议适配服务、只读服务和读写业务服务

（1）读写服务为业务的领域服务或者流程，其中又有比较特别的dao服务，这种服务往往处于关系网的最底层，除了依赖db外，不会再依赖其他的appplatform服务；

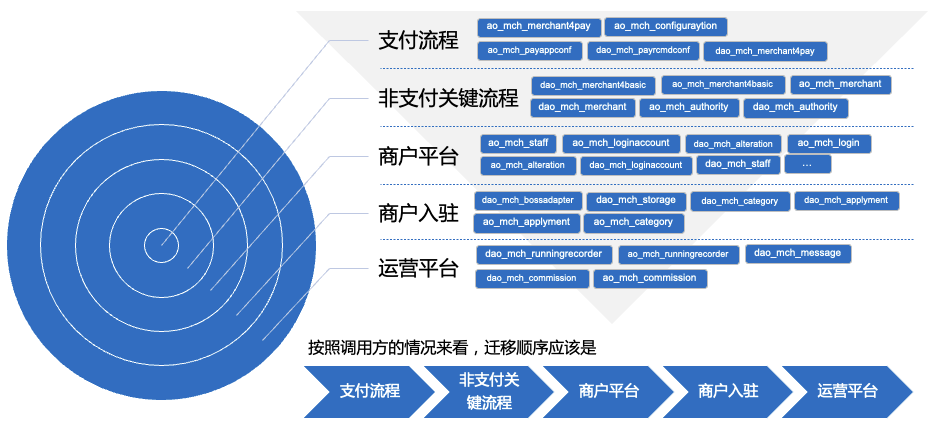
（2）协议适配服务往往是扇入比较高的服务；

（3）而只读服务一般是为了可用性而专门搭建的；

一般来说，依赖方越多风险指数就越大，而有特殊可用性指标的风险指数最大。按照这个维度分类的话，迁移顺序应该是：

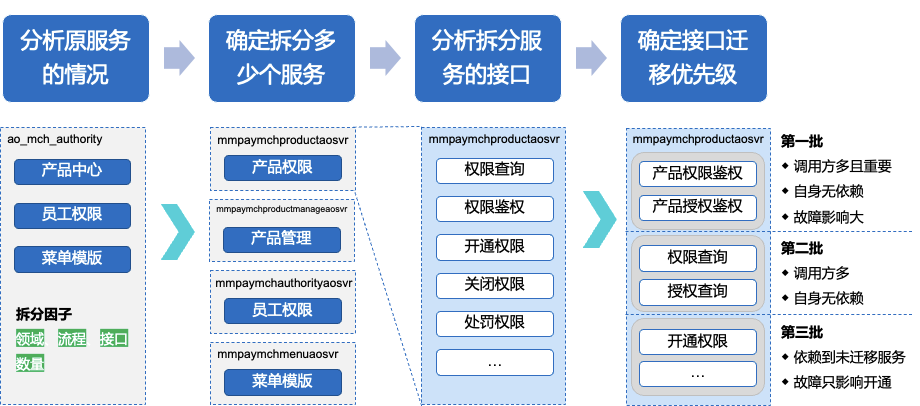
只读服务 > 协议适配服务 > 读写服务

**3.2、服务依赖方越重要风险指数越高**

****

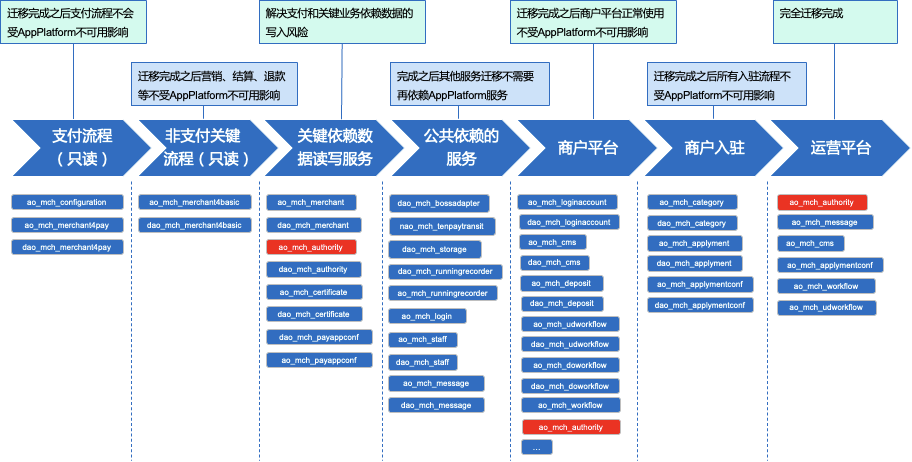
除了从待迁移服务自身进行分析之外，我们还可以从服务的调用方来进行分析，所有的调用方，从重要程度上来划分，可以分为支付流程、非支付关键流程、商户平台、商户入驻和运营平台。待迁移的服务根据各自的被依赖的情况，分门别类地划入这五个级别中。这五个级别中重要程度越高，代表着他的风险指数越高，那么我们越应该优先解决他，使得迁移得到最大的收益价值，从这个角度来看，迁移的顺序应该是：支付流程依赖的服务，非支付关键流程依赖服务、商户平台依赖的服务、商户入驻依赖的服务，最后是运营平台依赖的服务

**3.3、服务内不同接口风险指数也可能不一样**

****

之前appplatform服务过于粗重，从只有43个服务，但却有80w行业务代码就可以窥探服务的大小，这些服务往往里面包含了好几个领域的东西，这些不同的业务映射出来之后重要级别是不一样的，例如权限服务，里面包含了产品权限、员工权限、还有菜单模版，产品中心还分管理端接口和鉴权端接口，产品权限鉴权有很多关键业务在依赖，而管理端仅仅是运营平台配置权限才使用，因此这里应该将其划分成多个服务，各个服务以自己的优先级顺序来迁移。

**3.4、服务的风险指数综合决定迁移的优先级**



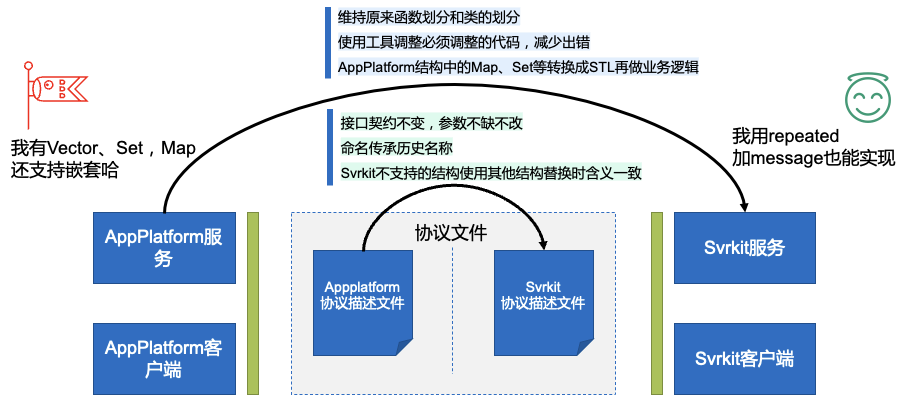
通过对待迁移服务内部关系、功能以及依赖方进行分析总结，我们划定了7个迁移优先级，分别是：支付流程（只读）、非支付关键流程（只读）、关键依赖数据读写服务、公共依赖的服务、商户平台、商户入驻、运营平台，这七个优先级的完成分别对应着迁移的一个里程碑，例如支付流程只读这个节点完成之后，支付流程就彻底解除了appplatform服务的依赖，即使appplatorm故障也不会导致下单不可用。

**四、如何正确重构**

通过上述分析，我们通过关联各个服务甚至是服务下各个接口发生不可用对业务系统带来的影响面来确定风险指数，确定了其迁移的顺序，根据迁移顺序执行，同样的迁移进度，剩余风险却降到最低。旦真实执行时，如何保证正确性呢？我将会从开发、测试和发布三个环节来阐述重构的正确性

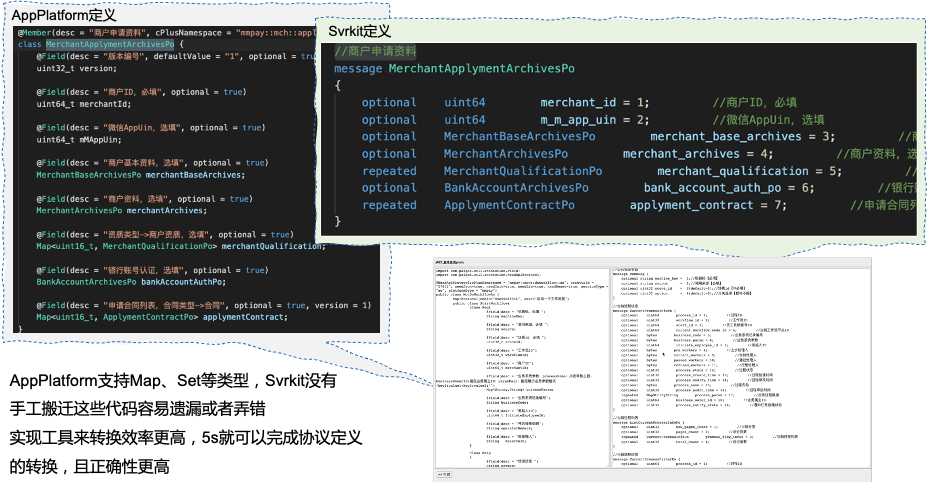
**4.1、开发正确的逻辑**

首先我们来看看开发环节我们怎么保证正确性，appplatform和svrkit框架的思路都是使用协议文件生成客户端和服务端的框架代码，然后开发基于生成的代码进行业务逻辑开发。所以重构的开发工作分为两个步骤：协议对等迁移和业务逻辑的对等迁移。



**4.1.1、协议对等迁移**

首先协议上要维持契约不变，参数上做到不缺不改，变量命名上传承历史名称，另外在协议定义上，appplatform支持map、set、bitset等这些复杂的数据类型，而svrkit则没有，因此这里不得不进行一些调整，调整需要保证表达的含义是一致的，并且加上注释说明怎么和旧的进行映射转换。认真分析appplatform和svrkit在接口协议上的差异，可以实现一个工具来进行转换，减少出错率提高开发效率。



**4.1.2、业务逻辑的对等迁移**

在逻辑代码迁移上，为了保证正确性，我们需要尽可能维持原有函数的划分和类的划分，重复的代码调整可以使用工具来执行，appplatform结构中的map和set等在svrkit下可以定义同等的map和set用于处理业务，完成之后在映射回svrkit结构中。appplatform和svrkit最大的差异是在命名风格上以及appplatform支持map、set等这些复杂的数据类型上，而业务代码里面数据字段的读取和设置构成了业务代码的很大篇幅，可以实现工具，能够选择指定代码一键转换成svrkit结构的用法



**4.1.3、风险可控的小范围改造**

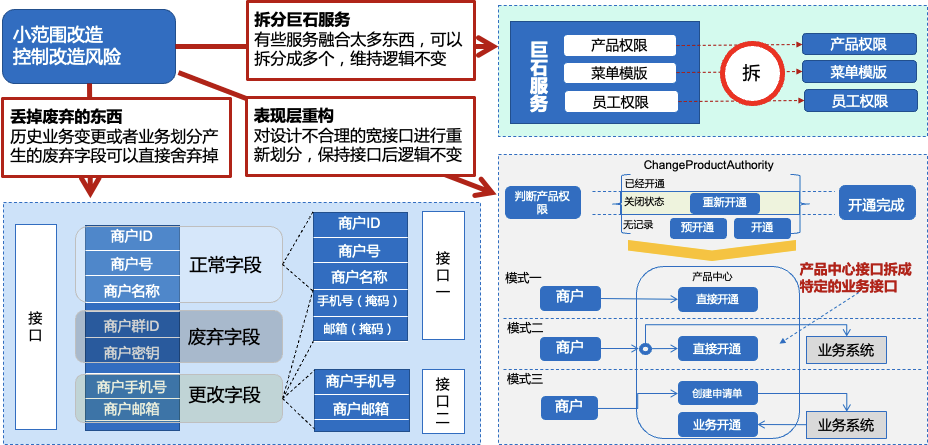
Appplatform服务承载了微信支付商户系统6年的发展，期间必然或多或少产生一些技术债务，存在一些忍受了多年的不合理地方，如果付出代价较小，在迁移过程中理所当然应该给改造好。所以我们可以在控制风险的前提下进行小范围的改造。主要改造有三个方向：

（1）丢掉废弃的历史包袱，包括历史废弃字段和接口等；

（2）拆分巨石服务，这里巨石服务是指服务耦合过多的东西，甚至有些完全没有关系的东西也躺在一个服务里面；

（3）对表现层进行重构，主要用于重构之前提供的宽接口，这些宽接口语义不明确，业务方使用门槛极高，可以对接口进行拆分重定义，但维持接口后面的逻辑不变，达到重构表现层的目的。

上面三个方法都只对系统进行了可控的改造，保证程序的正确性。

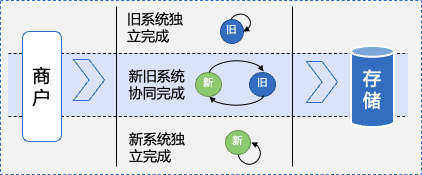


**4.2、测试保证正确性**

接下来来看看测试环节我是如何保证正确性的。首先测试没有银弹，改测试的点都需要去测试，逃不掉，测试环节保证正确性，主要采用了两种测试方法，质量测试和导流验证。

**4.2.1、质量测试**

整个迁移的过程中难免有段时间是新旧系统共存，一个业务流程的完成可能在新旧系统上来回进行了多次切换，因此在质量测试的时候需要考虑两个方面：一是功能是否与原来一致，也就是测试逻辑的正确性；二是新旧接口是否兼容，即兼容性测试。



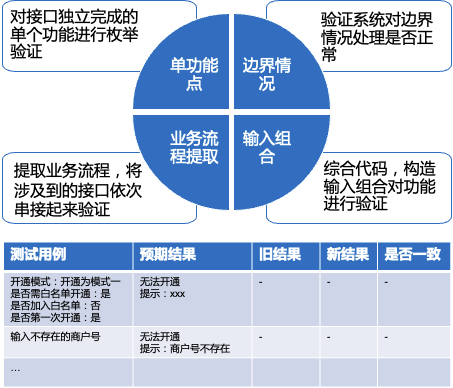
* **兼容性测试**

我们先来看看兼容性测试，我们可以将流程拆成几个部分，然后枚举每个环节在新旧的情况，列出用例矩阵，然后依次执行用例，看是否符合预期；



* **功能逻辑测试**

另外一个就是逻辑测试，我们可以从单个功能点、边界情况、输入组合以及提取业务流程四个方面收集测试用，制定出用例矩阵表，逐一验证是否符合预期。

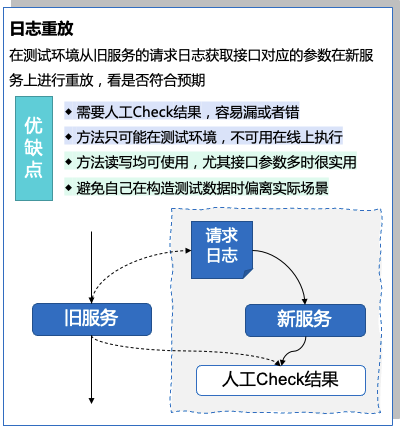


**4.2.2、导流验证**

前面说的质量测试通过完整的测试用例来保证功能的正确性和兼容性，但我们没办法保证所有的用例都能覆盖到，尤其是字段非常多的读写接口，这时我们可以采用导流测试来弥补。具体的导流测试，我采用了两种，一是日志重放；二是流量导入验证。

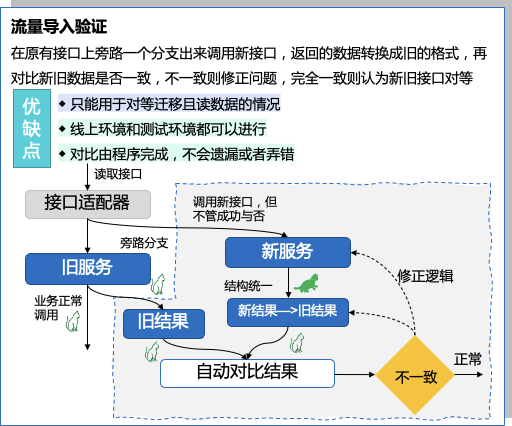
* 日志重放

这种方法是在测试环境将老接口的请求日志提取出来，构造重入条件，之后编辑成新服务的参数重放，在新系统上看是否符合预期，这种方法的好处是读写都可以使用，特别是在参数非常多的情况下，比较实用。能有效避免自己构造的数据偏离实际情况，不足点就是需要人工check容易弄错，另外这种方式只能使用在测试环境执行；



* 流量导入验证

是指在旧服务的调用方，我们一般在php的调用方，旁路一个分支出来，使用相同参数调用svrkit服务，取到结果之后转换成appplatform的格式，再和appplatform接口返回的数据进行对比，如果不一致则找到原因，修正之后再继续对比，如此反复直到完全一致，该方法的好处是不影响主流程可以在线上部署，另外对比由程序完成，不存在看错或者看漏的情况，不足就是这种方式只能用于读数据，写数据使用起来是有风险的。

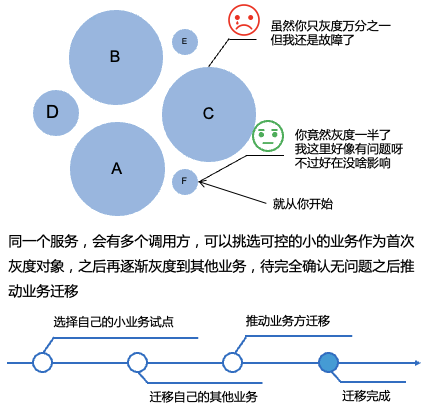


**4.3、灰度上线**

完成测试之后，则来到了版本更新的最后一个环节，灰度上线，为了近可能减少影响，我们迁移的新服务，都采用了两个灰度维度，按业务灰度和按商户号灰度。

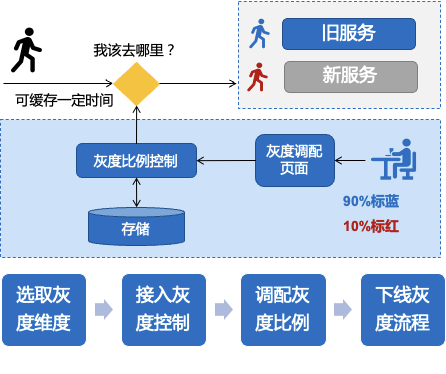
**4.3.1、按业务灰度**

按业务灰度和按商户灰度，按业务灰度是指我们选择一个自己能掌控的且影响面最小的业务作为首次灰度对象，当业务稳定没有问题之后再试点更多的业务，当完全确认没问题之后再推动业务方进行迁移；

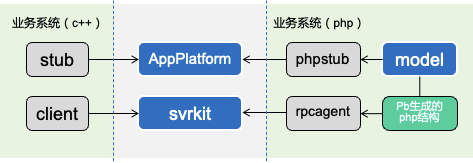


**4.3.2、安商户号灰度**

按商户号灰度是指在一次灰度中，只让自己想要的商户能进入到新的系统执行，其余的都维持在旧的系统上，因我们的微服务都是无状态的多机部署，按商户号灰度我们需要能做到多机同时生效，因此搭建了灰度控制系统，业务方接入灰度控制之后，可以通过运营平台直接按尾号控制灰度量。



**五、如何迁移调用方**



对等服务搭建完成之后，需要对调用方进行迁移，这个也是耗时最大的一部分工作，调用方分为两大类，c++调用方和php调用方。为了提高迁移的效率，我们制定了三种迁移方法：业务方自行更换接口、统一重置C++客户端、统一调整PHP适配器。

**5.1、业务方自行更换接口**

将新接口整理好给到业务方，业务方自行排期将接口由旧接口迁移到新接口，这种方式因风险低且能彻底摒弃appplatform框架，因此也是我们推荐的方式。因此他的优缺点如下：

优点：

（1）去掉appplatform相关的东西，彻底抛开appplatform的包袱

（2）业务方比较了解业务的情况，无论测试还是风险评估方面会更全面，不容易出错

（3）灰度上业务方控制更加灵活

缺点：

（1）以来业务方的排期计划，迁移周期会非常长

**5.2、统一重置C++客户端**

对于c++我们可以重载生成的stub，将参数转成svrkit参数之后调用svrkit接口，取到结果之后再转换成appplatform结构返回，这样业务方不需要进行业务流程改动，更换头文件修改变量定义，重编上线即可。他的优缺点如下：

优点：

（1）统一迁移，不依赖业务方的排期，迁移速度更快

缺点：

（1）appplatform还遗漏在业务代码里面，如果业务是个经常更新的业务，对后续维护不利

（2）另外灰度上面也不好控制

**5.3、统一调整PHP适配器**

相比于c++，php因语言特性，迁移上面能做更多的事情，我们可以调整php的适配层，将其迁移到svrkit，结构转换成与之前一样的数组返回，这样业务方无需任何改动，可以做到无感知地迁移。这种方式的优缺点如下：

优点：

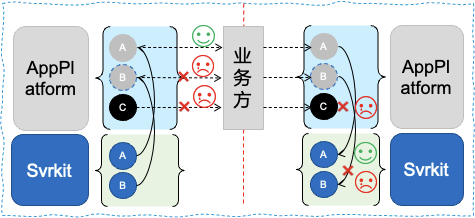
（1）无需业务做任何改动即可以完成迁移

（2）可以根据业务来做灰度

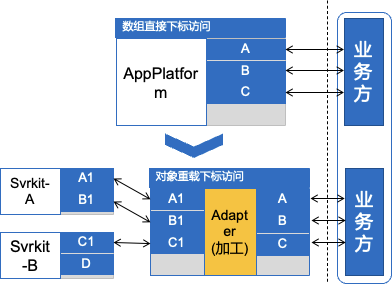
缺点：

（1）风险极高，很容易出错

**5.3.1、PHP迁移如何处理新旧差异**



php调用方的特点就是非常多，因为早期很多业务流程都在php中串接，前面提到了php可以统一从适配层进行迁移，但是如果对应的接口字段定义发生了变化，例如手机号拆分到了独立的接口获取，这种情况我们怎么办？



面对上面的疑问，我们的做法是定义一个类，从php的数组继承，重写下标访问的实现方法，通过get或者set调用字段的名称来决定从哪里获取数据，这样即使数据结构后面挂了多个数据来源，对于使用的业务方也是没有任何感知的。

最后更新于 2021-09-04 12:21

本文申报

如果觉得我的文章对您有用，请随意赞赏

赏

仅供内部学习与交流，未经公司授权切勿外传

分类：[文章](https://km.woa.com/group/wxgTEC/articles?cat_id=179295" \t "_blank) 标签：[重构](https://km.woa.com/group/wxgTEC/articles?tag_id=823" \o "重构" \t "_blank)(1) [迁移](https://km.woa.com/group/wxgTEC/articles?tag_id=3622)(1)



本文专属二维码，扫一扫还能分享朋友圈

想要微信公众号推广本文章？[点击获取链接](javascript:void(0);)

我顶 (12)

收藏 (39)

* [转载](javascript:void(0);)

* [收录](javascript:void(0);)
* [评论](javascript:void(0);)(3)
* [反馈](javascript:void(0);)

分享到

大家评论

[](https://km.woa.com/user/biaozhou)

[biaozhou](https://km.woa.com/user/biaozhou)

2021-09-04 12:58:44

赞！！

 顶 [回复](javascript:void(0);)

[](https://km.woa.com/user/bobbli)

[bobbli](https://km.woa.com/user/bobbli)

2021-09-04 17:44:02

👍

 顶 [回复](javascript:void(0);)

[](https://km.woa.com/user/xpniceli)

[xpniceli](https://km.woa.com/user/xpniceli)

2021-09-08 10:30:01

好文🐂 学习一个

 顶 [回复](javascript:void(0);)

[](https://km.woa.com/user/rickyrqzhao)



[切换到更多功能](javascript:void(0);)

关于作者

[](https://km.woa.com/user/widyhu)

[widyhu(胡忠云)](https://km.woa.com/user/widyhu)

WXG\商户产品部\商户基础开发组员工

作者文章

* [如何重构有一定历史的服务](https://km.woa.com/posts/show/520629?kmref=author_post)
* [vscode上基于protobuf补全和跳转的神器-cpptips](https://km.woa.com/posts/show/469575?kmref=author_post" \o "vscode上基于protobuf补全和跳转的神器-cpptips" \t "_blank)
* [商户基础多维监控接入指引](https://km.woa.com/posts/show/433901?kmref=author_post)
* [关于APSNET部署到IIS服务器下证书不能使用的解决方案](https://km.woa.com/posts/show/220197?kmref=author_post)

Copyright©1998-2021 Tencent Inc. All Rights Reserved

腾讯公司研发管理部 版权所有

[广告申请](https://km.woa.com/chartlets/km/) [反馈问题](javascript:void(0);)

[424/464/576 ms]