从微信支付宝支付接口设计谈 API 接口产品的设计经验和最佳实践

互联网有个说法，好的产品会说话，API 接口是一种特殊的产品，主要服务于 B 端企业，是一种典型的 B2B 服务模式。通常通过可配置的开放平台对外提供，设计一款好的 API 接口产品，需要有客户意识，要以服务客户、为客户带来便利、减少客户的对接成本、提高客户对接效率为目标，才能设计出来一款优秀的 API 产品。

我这两年一直忙于一款线上与线下相结合的一站式企业定制支付平台，重新定义了产品的对外 API 的形态，优化了产品体验，提高了产品的对接效率，重构了支付平台应用架构。由于我开发的产品是公司的主打产品，所以，本文中不会使用我开发的产品作为示例来阐述 API 的设计要点，而是通过微信和支付宝支付接口的设计，来谈 API 接口产品的设计经验。目的是把接口设计的各种经验分享出来，免得大家设计不合理的接口，影响开发者使用，或者被开发者吐槽不专业、不好用、太重等等。

开放平台文档的提供形式

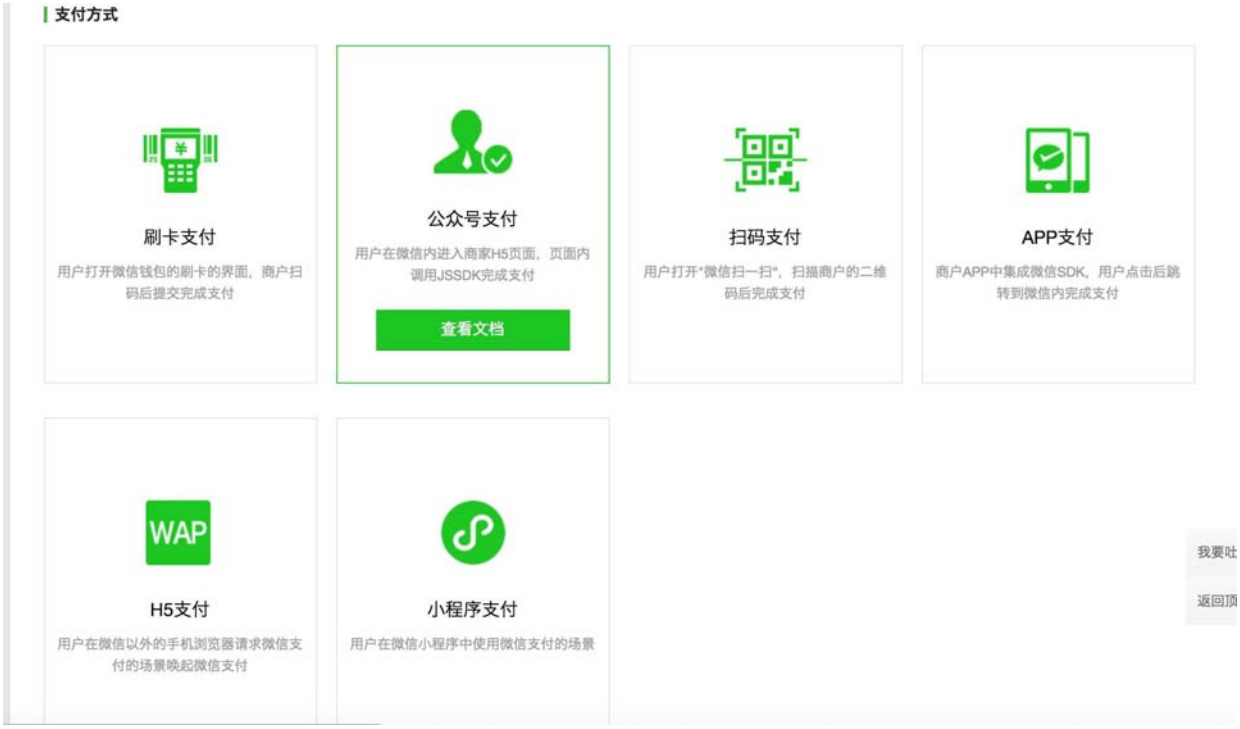
首先，作为客户，要对接一款 API 产品，客户的开发人员首先需要接触的就是对接文档。传统的方式是通过技术支持人员把最新的 PDF 或者 Word 文档交给客户，客户线下阅读和对接，而更好的体验是直接在开放平台上阅读在线文档，这样不但更加方便客户获得文档的路径，还方便 API 平台对文档进行自动升级。在线下传递文档模式下，版本的更新还需要技术支持的介入，这增加了很多成本。

下面链接是微信支付和支付宝支付 API 文档的首页。

- 微信支付文档
- 支付宝支付文档

大厂的 API 文档首页看起来清晰易懂，给读者的体验是：不需要任何的学习成本，一眼就能找到自己想要找到的内容。

下面是微信支付文档的主页截图。



我们看到微信在主页上罗列了所有的支付方式的文档，这包括刷卡支付、公众号支付、扫码支付、APP 支付、H5 支付、小程序支付，千万不要认为这是简单的罗列，实际上罗列的顺序至关重要，主要取决于微信对各个产品的定位以及市场对各种支付方式的需求多少。排在第1位和第3位的是刷卡支付和扫码支付，这是我们常说的“扫码”和“被扫”支付，显然，这是支付产品中最重要的支付方式，另外，排在第2位的是公众号支付，公众号支付也是微信主打的支付方式，这个支付方式并不是看起来那么的简单，它的背后是内容运营做背书的，微信想让支付和内容运营形成一个闭环，形成一个自包含的生态，因此，大力推广公众号支付，这就是为什么公众号支付的时候需要企业报备，虽然很麻烦，但是价格却很合理的原因。而并不常用的 H5 支付，通常费率很高，小程序支付则是一个新产品，我曾经提出一个创新产品，试图在小程序中对接快捷支付，不考虑微信政策风险，相信现在仍然有市场，目前小程序还只支持微信支付，之所以最后一个是小程序支付，这是因为小程序是微信生态中继微信语音聊天、微信支付、微信公众号的又一大创新，虽然现在还并不那么显眼，但是让我们拭目以待。

下面是支付宝支付文档的主页截图。

我们看到支付宝支付文档首页也同样提供了当面付、APP 支付、手机网站支付和电脑网站支付等四种支付方式。这里的当面付指的是“扫码”和“被扫”，这两种方式被归类成了当面付，比微信所说的刷卡支付和扫码支付更合理，而且名字更加容易理解。说起当面付，除了传统的 POS，那就是“扫码”和“被扫”两种支付方式。然后就是 APP 支付、手机网站支付，最后是电脑网站支付，可见把移动支付的两大支付方式排在了第2位和第3位，优先于电脑网站支付，这也体现了现在是移动互联网时代，也是移动支付的时代。

对于微信支付，支付宝支付还提供了电脑网站支付这种支付方式，这是在互联网时代里，淘宝用户在淘宝买东西，在支付宝 PC 网页端支付的场景的延续，在另外一方面，微信支付独有的公众号模式，在支付宝支付中也没有提供，但是近期支付宝的生活号支付已经开始推广了，具有异曲同工的效果。

清晰易懂的 API 文档

为什么有的文档看起来就像吃烤鱼一样爽，而有的文档看起来很辛苦？这里，我们一定记得写文档是给客户来看的，因此，我们要思考客户想看什么样的文档？客户想怎么看文档？而不是单方面的把一大坨的内容堆砌给客户，心想着：反正都给你了，看不看得懂是你的问题了。

首先，我们先看下微信支付 API 文档。从上一节在主页中我们看到，微信把各种支付方式摆放在微信支付 API 文档的主页，客户想看哪个支付方式，只需要点击即可，这不需要任何的学习成本，而且在光标放到每个支付方式的图标范围内，这个支付方式的图标会自动描框，并且下面显示查看文档的按钮，如何操作则一目了然，根本不需要学习或者思考，这就应了那句话，好的产品是给“傻子”用户设计的。点击进入某种支付方式，左边列表栏是关于这种支付方式的所有功能，点击 API 项目即可查看对接 API 的文档，于是乎，我们轻松的就可以找到我们想要看的 API 对接文档，下面显示的查询订单 API 的文档。API文档包括对接的 URI、参数列表、返回值列表等等。

在 API 文档的最下方，是这个 API 可能产生的错误码，对接支付 API，错误码的处理尤其重要，因此这里直接列在了 API 下面，如下图所示。

错误码

名称	描述	原因	解决方案
ORDERNOTEXIST	此交易订单号不存在	查询系统中不存在此交易订单号	该API只能查提交支付交易返回成功的订单，请商户检查需要查询的订单正确
SYSTEMERROR	系统错误	后台系统返回错误	系统异常，请再调用发起查询

这里看到要想寻找微信支付安装 API，只需要经过两个页面跳转，非常的方便。

支付宝支付 API 文档具有异曲同工的作用，在上节中，从首页的支付方式中直接点击某一个热门产品，当光标放在热门产品的时候，我们也看到了自动描框，这一点给用户可以点击进入的提示很重要，这就是设计指引，让用户使用的时候自然就知道应该怎么来用，而不需要学习成本，这才是好产品和好文档。

点击进入某个支付方式的主页以后，我们看到了和微信一样，使用的左面导航栏。要查看对接的 API 文档，我们需要再次点击左边的 API 列表，进入 API 列表页面，如下图所示。

在 API 列表页面，选择某个 API 点击进入 API 文档，文档内容包括对接的 URI、参数列表、返回值列表等等，如下图所示。

从列表中选择了某个 API 后，就可以看到这个 API 的对接文档，文档的最下方是对应的错误处理和可能产生错误码，如下图所示。

我们看到，我们需要跳转3个页面才能看到支付宝支付 API 文档，比微信支付 API 文档要多一个步骤。

命令模式和 RESTful 的抉择

首先，我们查看微信的 API，它使用的是类 RESTful 服务的实现，每个功能都有各自的 URI。

微信 API 的示例如下。

下单：

```
https://api.mch.weixin.qq.com/pay/unifiedorder
```

查单：

```
https://api.mch.weixin.qq.com/pay/orderquery
```

可见，下单和查单的 API 具有相同的 URI 前缀，即 https://api.mch.weixin.qq.com/pay，各自的后缀为 unifiedorder 和 orderquery。

然后，我们再来查看一下支付宝支付的 API 的设计，它使用的是命令模式。

所有的功能都使用同一个 URI。

统一的 URI 为：

```
https://openapi.alipay.com/gateway.do
```

在 HTTP 协议中，提供了一个方法参数：method，通过方法参数来传递具体使用的功能，不同的方法表示使用不同的操作。

下单：

```
alipay.trade.pay
```

查单：

```
alipay.trade.query
```

我们看到，支付宝使用命令模式，只提供了一个统一的 URI，然后通过 method 参数来表示具体调用哪个功能和操作。

对比微信支付的 RESTful 服务和支付宝支付的命令模式，两者各有优缺点。

两种方式的优点如下。

- 对于命令模式，URI 的设计比较简单，只需要设计统一的 URI 即可，具体选择哪个功能都在 method 中来表示，method 参数易于扩展，扩展时不需要对 URI 和以及7层代理的 URI 路由进行修改，只需要对后端代码进行修改即可，对后期增加更多的命令和操作提供了方便。
- 对于 RESTful 模式，每个功能和操作都有一个对应的 URI，对于开发者来说学习起来比较简单，使用起来比较方便，URI 信息是在 HTTP 头上进行传递的，由于 URI 的不同，在7层代理可以灵活配置分流和路由，这通常在迁移的过程中是需要的。

我们看到这两种方式各有优点，实际上，命令模式代表中心化的思想，而 RESTful 模式代表的是分而治之的思想，有的时候优点就是缺点儿，缺点就是优点，其实是个博弈。

命令模式是中心化的思想，比较封闭，虽然对将来的扩展实现比较容易，但是失去了简单易懂的优点。而 RESTful 是分而治之的思想，保持原汁原味的简单 API 的风格，看到 URI 就知道 API 的功能，简单易懂，方便使用。

这里，通过一个比喻来对比命令模式和 RESTful 模式。想象一下桌子上有3个杯子，杯子里面装着水、酒、雪碧，看起来都是透明的，如果每个杯子都没有标签，我们得拿起挨个尝一下，才知道哪个是水、酒或者雪碧，这就像命令模式一样，URI 长得一样，想知道它是干什么的需要进一步看内部的参数，除了看起来麻烦，这在7层代理分流的时候，也是需要解析 HTTP 体才能看到 method 参数，性能偏低，然而，如果我们在杯子上加了标签，我们一眼就看出哪个是水、酒或者雪碧，这就像在7层代理需要分流的时候，直接在 HTTP 头中获取 URI，做不同的事儿。

API 命名规范

好的 API 是会说话的，而不是腼腆不语的、深藏不漏的，因此，API 的设计应该秉着简单、易懂、使用方便的原则来提供。

我们从微信和支付宝的 API 中看到，URI 使用的都是非驼峰式无下划线的命名，例如，订单查询的 URI 为 /orderquery，而不是 order_query，也不是 orderQuery，而参数命名也是采用非驼峰式有下滑线的命名，例如，商户 ID 为 mch_id，而不是 mchId，为什么不用驼峰式命名，因为 HTTP 协议是不区分大小写的，即使使用驼峰式，也没有什么实际的意义。

对于 URI 的设计上，无论是微信支付 API 还是支付宝支付 API，都保持简单明了的风格，例如：

```
https://api.mch.weixin.qq.com/pay/orderquery
https://openapi.alipay.com/gateway.do
```

一般不需要定义版本字段，因为版本的升级一般通过增加参数来解决，并向前兼容，读者也发现一般 API 中定义的版本都是 1.0，从来没有升级过。也不需要把某个固定关键字放到 URI 上，例如 rest。实际上，是不是 RESTful 风格的 API，从 URI 的设计上一目了然，不需要再增加提示信息来表达，假设我们为微信的 URI 增加 restful 的信息，如下所示。


```
https://api.mch.weixin.qq.com/restful/pay/orderquery
```

那么，我们看起来也会觉得这个 URI 显得臃肿，不够简单明了。

在这里，我们强烈推荐所有的 API 产品，无论是 URI 还是参数最好都使用英文来命名，最忌讳的是汉语拼音和英文混用，这样确实显得不够专业，降低了 API 的档次，使用英文的时候尽量使用正确词汇来表达含义，例如，支付中常说的商户是 Merchant，而 Customer 表示的是顾客，也不要把 queue 和 queen 混读，这些内容需要准确把关，才能体现出设计支付产品的专业性，才能服务好客户。

这里，我们在设计一款 API 产品时，我们一般定义如下的规范。

1. URI 采用非驼峰式无下划线的命名，而参数使用非驼峰式有下滑线的命名。
2. 使用 RESTful 风格的 API，为每一个操作 API 显示的定义不同的 URI 来区别。
3. 任何一个 URI 或者参数，一般由最多3个单词组成，尽量使用缩写的单词。
4. 单词要使用英文来命名，英文要尽量准确表达其意，有固定缩写的词就用固定缩写，例如：Identity->ID，没有固定缩写的，缩写的单词通常采用省略单词的元音来构造，例如：method -> mthd。
5. 建设一套适合业务的数据词典，术语按照数据词典命名，在多个 API 产品中保持风格统一，同一产品中对同一事物命名要前后一致。
6. 原则上一个标识符不超过20个字母。

7. 在 URI 的设计上，要简单明了、风格统一、一目了然，不需要类似版本和 REST 等提示信息。
8. 使用的域名必须是英文命名并且具有品牌名称。

这里我们给出几个理想的 URI 设计的案例。

交易 API：

```
/std/trade/order
/std/trade/orderquery
/std/trade/refund
/std/trade/refundquery
/std/trade/orderclose
```

入网 API：

```
/sys/merchant/reginfoadd
/sys/merchant/prodinfoadd
/sys/merchant/agreeinfoquery
```

打款 API：


```
/std/balance/remit
/std/balance/remitquery
```

对账 API：

```
/std/bill/tradedaydownload
/std/bill/trademonthdownload
/std/bill/remitdaydownload
```

数据词典示例如下：

中文名	英文名	URI	参数
用户	user	user	user
商户	merchant	merchant	merchant
商户编号	merchantno	merchantno	merchant_no
主商户编号	parentmerchantno	parentmerchantno	parent_merchant_no
支付	payment	payment	payment

中文名	英文名	URI	参数
订单	order	order	order
查询	query	query	query
出款	remit	remit	remit
退款	refund	refund	rfnd
注册/入网	register	reg	reg

扩展字段的设计

过去的几年里，我负责整个公司核心支付平台的设计评审工作，经常看见小伙伴在 API 上设计扩展字段，未雨绸缪。

小伙伴的想法是好的，为了将来的扩展，先预留字段，等需要的时候，我们就不用再加了，直接使用扩展字段就好了，于是我们看到很多接口有 ext1、ext2 等字段，我们需要这些“万恶”的扩展字段吗？为什么说这是“万恶”的字段，这要从某个生产事故说起。话说曾经就有这么个扩展字段，字段是很久以前的开发者设计的，后来这个字段被不断的使用，终于有一天由于新的开发者不知道这个字段里面放的什么信息，也没有相应的文档说明，就武断的在一个新需求中，把一个新数据放进这个字段，覆盖了原有的数据，于是悲剧就产生了，有用的信息被覆盖了，产生了非常不好的后果。

从设计规则上看，我们不要做这样的未雨绸缪，因为它埋下了祸根，而且是一个不知深浅的祸根，这种问题一旦发生，产生的后果可大可小。

因此，我们一般不需要设计扩展字段，即使设计了扩展字段，等我们有新功能需要新字段的时候，我们也一样需要开发，因为新需求还是会有新逻辑的实现，即使预留了扩展字段也不能达到在不改动代码的前提下实现新需求。

还有些小伙伴愿意使用一个 JSON 格式的扩展字段，有新需求了，就往 JSON 里面放，直到 JSON 字段不堪重负为止，这也是不好的设计实践，需求中需要的重要数据都要单独设计相应的字段显示的传入，而不要用一个 JSON 大字符串来传入，但是有的小伙伴说这个字段是透传的，或者是只读的，都设计出来字段太大了，这类场景需要具体问题具体分析，看看是否这类开放式的非交易类的数据，可以通过别的方法来传递，比如单独的批量接口、单独的接口等等，这里需要更灵活的分而治之的思想。

当客户意识遇上平台意识

这里我们来看下当客户意识碰到了平台意识，会发生什么？客户意识从客户需求出发，从各种客户需求中，挖掘客户真正的核心需求，俗话说，打蛇打七寸、擒贼先擒王，也是这个道理，做事儿要抓住要点，尽可能的满足客户的真实需求。而平台意识则更多是

从提供方角度来设计功能，来满足客户需求，这就造成我们设计的平台有可能没有最高效的满足客户需求，也许客户需要的是个手枪，而我们给的是一个大炮。

这里举个例子，也是我曾经做过的一个设计评审案例，案例里，需求是设计商户对账 API 的提供方法，一般会提供接口下载对账文件、FTP 下载对账文件、商户后台下载对账文件，这里面我们的焦点是通过 API 来提供对账文件，这个 API 应该怎么设计会更好呢？

于是，开发人员中就出现了两个思路。

其中一种思路，把 API 对账进行抽象和泛化，设计了一套完善的文件服务平台，可以上传和下载通用的文件，当然，这个文件服务平台对于下载对账文件也是一定能支持的，因此，就设计了如下的 API。

```
BizSystem bs = ...;
String fileId = bs.getReconciliationFileName(merchantNo, date);

FileServicePlatform fsp = ...;
fsp.downloadFile(fileId, LOCAL_DRIVE_FOLDER);
```

在这个 API 上，客户需要首先从业务系统根据商编和日期来获取对账文件 ID，然后，使用对账文件 ID 再到文件服务平台中下载对账文件。

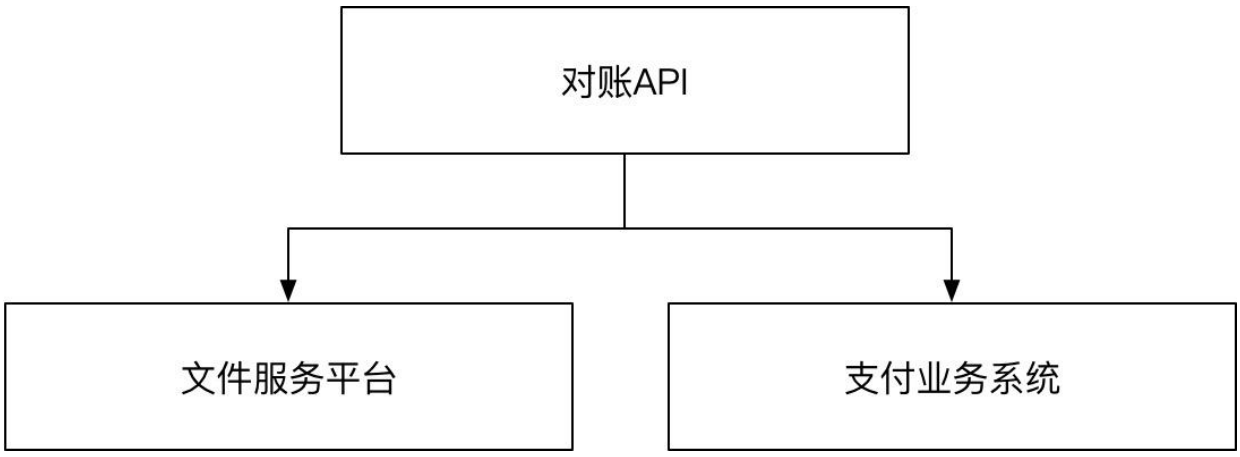
另外一种思路则更具有客户意识，拥有这种思路的人从客户需求出发，他们想更好的服务客户，解决客户的痛点，既然客户想做的是下载对账文件，我们要尽可能让客户以最简单的方式拿到对账文件，因此，他们设计了如下的 API。

```
ReconciliationSystem rs = ...;
rs.getReconciliationFile(merchantNo, date, LOCAL_DRIVE_FOLDER);
```

我们看到，在后面的这个设计中，只要提供了商户的商编、日期和本地磁盘目录，就可以直接下载文件，相比较而言，前面方法虽然设计了文件服务平台，但是需要两步才能下载一个对账文件，并且需要客户直接使用文件服务平台的 API，而客户并不关注文件服务平台。

那么，到底哪个方案更好呢？其实这也是个博弈，没有最好，只有更好，有的时候我们需要结合两种方法。

更好的方法是我们对外设计的系统要进行分层，对内将不同的功能分到不同的层次或者系统中，对外要针对客户来包装内部的分层和服务。



从上图中我们看到，我们可以实现一个文件服务平台，但是对外提供 API 的时候，还是要秉着简单、易用的原则，就像上面的第2种思路一样，这样就可以有效的避免：客户需要一只手枪，我们给予一门大炮。

因此，当客户意识遇到平台意识的时候，我们不要着急，他们是不冲突的，有效的结合二者，会产出更有效的解决方案。

我们看到支付宝通过查询对账 API 下载地址，将对账单下载引流到文件服务平台进行下载，而微信提供同步下载对账单的接口，但是，我相信微信已经对对账单下载和交易系统的 API 进行了隔离，因为使用的 URI 不同，在7层代理上实现隔离是个很容易的事儿。

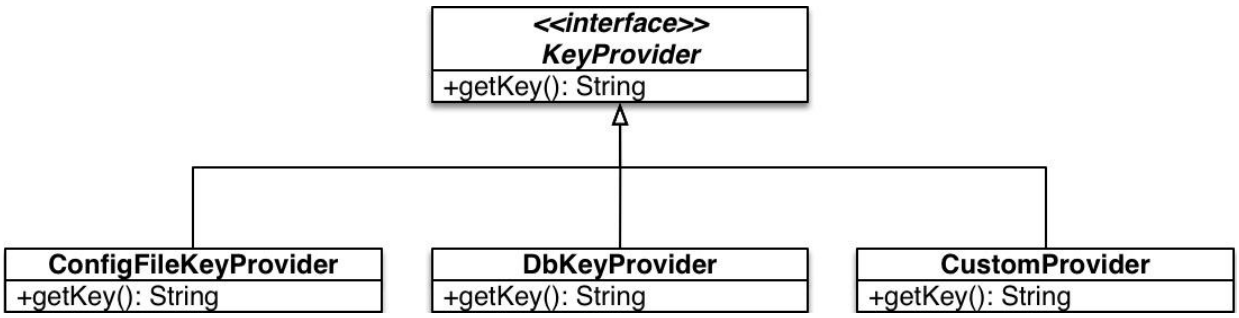
- 访问这里，查看微信支付对账单下载 API。
- 访问这里，查看支付宝支付对账单下载 API。

对于客户意识和平台意识，我想给大家提供另外一个案例——在我们对客户提供的 SDK 中，如何初始化秘钥的问题。

如果我们从技术平台的角度来思考，通常我们会实现一个秘钥提供者接口（KeyProvider），然后可以灵活的插入接口的实现，使用者可以实现这样的一个接口来提供秘钥，这看起来很酷，也是一种模板回调的设计模式，但是对于使用者来说并不一定很简单。

首先，使用者得开发一个类来提供秘钥，这个类还需要读取文件或者数据库来获得秘钥，这对有些开发者来说可能是一个负担。因此，从客户意识来说，这是一个半成品，并不像设计模式本身那样闪闪发光。

因此，我们推荐从客户意识来考虑如何实现这个需求，我们可以提供几个默认的实现，一个秘钥配置方式对应一个实现类，例如，从配置文件中取得秘钥（ConfigFileKeyProvider），从数据库中取得秘钥（DbKeyProvider），这样，客户只需要通过配置选择其中的一种使用接口，对于那些20%的特殊客户的特殊需求，他们可以开发一个定制化的秘钥提供类（CustomProvider）来从一个非常特殊的地方来获取秘钥，比如说加密机，这也是支持的，但是这显然不是80%的通用场景。我们设计任何系统，平台化的内容都是80%的通用需求，而我们的平台要对20%的专用需求留下扩展点，如下图所示。



因此，使用回调模式获取密钥是没问题的，但是需要为80%的用户提供通用的解决方案，并留出扩展点来实现20%的非通用需求。

B2B API 的安全策略

API 接口属于典型的 B2B 的产品，对安全的要求比较特殊，需要验证企业的身份，这和用户端 App 的安全技术栈不同，B2B 的 API 产品通常通过签名和加密来保证安全。

要实施签名和加密安全策略，通常我们会进行抉择是用签名还是加密呢？用对称加密还是非对称加密？

这还是要从需求说起，安全需求也是需求的一种，我们总结安全需求无非这几种：

1. 防篡改
2. 防偷窥
3. 防泄漏
4. 防抵赖
5. 防中间人攻击

要满足这些安全需求，我们采用不同的安全策略和方法，我们从密码学的历史开始说起，它一共经历了三个时代。

1.算法加密时代

在很久以前，没有各种加密算法，为了防止软件被盗用，通常在软件中预留了一段加密算法，如果用户输入的注册码满足算法的需求，就通过了校验，这个时代叫做算法加密时代，以算法为核心来验证身份，但是这种方法有明显的缺陷，只要有足够强大的技术力量，算法都是可以被破解的。

2.对称加密时代

这一时代，发明了对称加密算法，加解密方约定同一个秘钥，加密方用密匙加密，解密方用同一个密匙解密，如果可以成功解密，说明加解密方都有权限访问数据。对称加密有两个应用场景，一个就是签名，一个就是用来加密，对称加密和对称签名都是使用的对称加密算法，但是目的不一样，签名就想盖章一样，防止被篡改，而对称加密则是为了防止偷窥和防止泄漏。但是这一时代的对称加密算法也有个明显的缺点，就是加解密双方约定了同一个秘钥，加解密双方理论上都是可以抵赖的，如果秘钥泄露了，都可以说是对方泄露的，是无法判断是谁把秘钥泄露了。这一时代著名的算法有 3DES、AES。

3.非对称加密时代

到了非对称加密时代，解决了对称加密带来的抵赖问题，这一时代可以生成1对秘钥，1对秘钥由公钥和私钥组成，公钥加密数据只能有私钥来解密，私钥加密数据只能由公钥来解密，前者正式加密的应用场景，而后者是签名的应用场景。著名的 SSL 就是使用的非对称加密。这一时代著名的算法有 RSA。

了解了这个背景，我们现在来回顾一开始我们提出的安全需求，要满足防篡改，只需要对称的签名即可，要满足防泄露和防偷窥，只要使用对称的加密即可，但是要满足防止抵赖，必须使用非对称签名和非对称加密。

但是要满足防止中间人攻击，那么，我们需要使用双向的非对称安全验证，通常使用双向的 SSL 来保证。

在支付行业里面的实践中，为了安全以及合规需求，我们通常需要使用非对称签名来保证数据不被篡改，使用非对称加密保证敏感数据不泄露，如果有出款需求，我们一般使用非对称的双向 SSL 证书来保证出款的安全。

以何种方式提供范围查询 API

一款好的支付 API 产品一般包含下单、查询订单、退款、查询退款等核心操作，但是有的时候来自于客户压力，客户想按照某一范围来查询订单，例如下单时间范围，那么这样的范围查询真的是支付平台应该提供的功能吗？

在这里，我们不提倡为客户提供这样的范围查询，这其实是在为客户做行业的业务订单系统，客户的业务订单系统应该保存所有的订单信息，我们提供的订单查询和退款查询数据交易系统的订单查询接口，是单笔查询，必须指定客户订单号或者我们支付系统的订单号，不应该按照时间范围查询。这可以参考微信和支付宝的 API，两者都没有提供类似的接口。

那么有的时候，确实有这样的客户需求，客户就是想把我们的支付系统当做他们的订单系统，他们需要查询订单了来到我们支付平台来查询，而且还可能是大客户，我们得罪不起，因为除了微信支付宝以后，其他的支付公司都属于乙方，并没有多少的话语权，“yeap, fair enough”，这是个合理的需求，也是一个真实的客户需求，因此，我们确实应该给予实现。

但是我们需要思考在哪里实现，我们不应该在交易系统中对外提供范围查询，因为这会影响交易系统的性能，严重情况下范围查询会拖垮交易系统，因此这类的查询系统应该在支付核心的上层系统中定义业务订单系统，在订单系统中为商户提供多样化的查询需求，并与交易系统分离。由于这需要一定的开发成本和运维、运营成本，因此，我建议与客户洽谈进行一定的收费，这等于我们在为客户做深入行业的业务订单系统。

最后，如果我们真的为客户提供了范围查询，一定要进行分页处理，并提供一定的限流措施，因为在严重情况下范围查询的数据量比较大时，会产生巨大的流量，可能会堵塞机房内外的专线。

以何种方式来使用 APPID

从微信和支付宝的产品设计中，我们看到均使用了 APPID，当一个平台做的越来越大，从开放平台对外提供的 API 产品也越来越多，不同的 API 产品之间的权限需要进行控制，不能让 A 产品的客户访问 B 产品功能，这就需要对不同的产品提供不同的 ID，APPID 就是满足这个需求的。

一个客户可以开通多个产品，每个产品对应一个 APPID，在使用 API 和我们的开放平台对接的时候，必须提供 APPID，来验证这个客户是否开通了这个产品，这是很好的一个隔离模式的实践。

然而，问题来了，在中小型公司里，多数时候只有一款核心的应用通过 API 开放平台来提供的，每次商户提供了商编还得提供 APPID，或者让商户除了记住商编，再记住那么长的一个 APPID 也是一个难事儿。

因此，这里还是应用二八原则，对于通用的80%的需求，也就是只开通一种产品的客户，我们允许客户选择一种默认的产品，这样客户只要传递商户 ID 即可，我们自动为这类商户选择模式的产品，方便商户对接，减少商户的对接成本。对于开通多个产品的商户，他们可以传递 APPID。传递 APPID 的，就使用传递的 APPID 来校验权限，这样既满足了通用需求，又满足了专用需求。

撤销、关闭和退款

在做设计评审的时候，总有小伙伴来问我，什么时候设计撤销、关闭和退款，其实这是3个非常容易混淆的接口。

- 退款，一般分成用户发起的退款以及差错退款，前者是支付成功后，用户主动退款，而后者是支付成功后，发现重复支付或者有差错，系统发起的退款。
- 关闭，订单超过了有效期自动关闭，不再接受支付成功的回调消息。或者订单未到有效期，商户端主动发起关闭，关闭的订单还没有支付成功。
- 撤销，无论支付是否成功，都可以发起撤销，发起撤销后，支付成功的交易要退款，支付未成功的交易，则保证不会再次成功。

根据上面的定义，我们看到，退款是在支付成功的前提下发起的，而撤销则是在不知道支付是否成功的前提下发起的，订单关闭是在订单一定没有支付成功前提下发起的。

但是，在有些产品设计上，这几个功能有着千丝万缕的关系，例如，有的支付产品通过撤销来包装退款，还有通过退款来包装撤销，这需要在具体的场景下具体分析。

那么设计这几个功能的黄金原则是：

1. 退款是一个通用的业务功能，只要有用户需求，我们就应该对外提供，内部不管是用银行提供的退款接口还是撤销接口都可以。
2. 撤销是针对面对面产品来提供的，一般是在面对面的交易过程中，支付状态未知，但是用户着急获取结果，商户端发起撤销来终止交易。

现在，我们来看下微信与支付宝的设计。

对于微信公众号支付，属于在线的移动支付，我们看见提供了退款和订单关闭的功能，如下图所示。

对于支付宝的支付，属于在线 PC 支付，我们看见同样提供了退款和订单关闭的功能，如下图所示。

电脑网站支付API列表

更新时间：2017-11-

此列表包含该产品所涉及的所有接口，点击"查看文档"可查看接口的公共请求参数，业务请求参数，返回参数，其他语言请求示例以及错误码等。

接口英文名	接口中文名	API文档
alipay.trade.page.pay	统一收单下单并支付页面接口	查看文档
alipay.trade.refund	统一收单交易退款接口	查看文档
alipay.trade.fastpay.refund.query	统一收单交易退款查询接口	查看文档
alipay.trade.query	统一收单线下交易查询接口	查看文档
alipay.trade.close	统一收单交易关闭接口	查看文档
alipay.data.dataservice.bill.downloadurl.query	查询对账单下载地址	查看文档

而对于属于面对面支付的刷卡支付，提供了退款和撤销等功能，如下图所示。


术语
支付账户
接口规则
刷卡支付
场景介绍
验证密码规则
案例介绍
商户侧流程
API列表
提交刷卡支付
查询订单
撤销订单
申请退款
查询退款

下载对账单

而对于支付宝当面付的产品，也提供了退款和撤销等功能，如下图所示。

当面付API列表

更新时间：2017-06-26

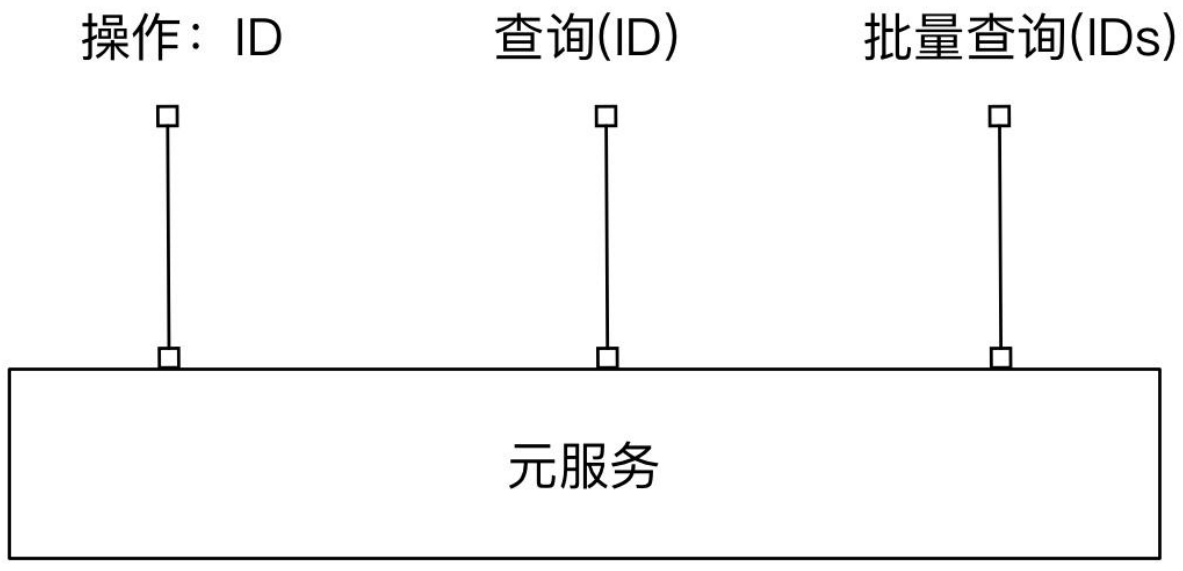
接口英文名	接口中文名	API文档
alipay.trade.pay	统一收单交易支付接口（条码支付）	查看文档
alipay.trade.query	统一收单线下交易查询	查看文档
alipay.trade.cancel	统一收单交易撤销接口	查看文档
alipay.trade.refund	统一收单交易退款接口	查看文档
alipay.trade.precreate	统一收单线下交易预创建（扫码支付）	查看文档
alipay.data.dataservice.bill.downloadurl.query	查询对账单下载地址	查看文档
monitor.heartbeat.syn	交易保障接口	查看文档

同步还是异步的抉择

这里，我们来看下接口设计的核心问题，到底使用同步还是使用异步会更好。实际上，同步和异步也是一个博弈，各有优缺点，没有哪个会更好，只不过都会有各自的场景，同步更多应用在事务较小，返回较快的场景，效率较高，异步通常应用在事务较大，业务逻辑处理比较复杂，不能在有限时间内返回结果的场景。有的时候我们需要结合同步和异步两种方式来满足业务需求。

在支付场景里，我们通常使用两种情况的结合来满足业务场景，通常我们会提供下单接口，然后提供同步的查询接口来查询订单的最新状态，这是最终一致性的查询模式。

查询模式如下图所示。



同时，我们会提供异步的回调接口，异步的通知商户支付交易的状态，如果商户对接了回调通知，则能够自动的得到支付成功的通知。

因此，有了同步的查单接口以及异步的通知接口，商户的业务系统实现起来将会更加简单和可靠。

我们看下微信和支付宝的设计案例。

在支付宝的当面付中，既提供了订单查询又提供了异步通知接口，如下图所示。

同样，在微信公众号支付中，既提供了订单查询又提供了异步通知接口，如下图所示。