

13 | 接口规范，是协作的合约

2019-02-01 范学雷



一个软件项目，一般需要交付两类文档。一类文档是面向开发者的，另一类文档是面向最终用户的。这两类文档，由于面向用户的不同，无论是内容还是形式，都有巨大的差异。今天我们先来聊聊面向开发者的文档。下一讲中，我们再接着聊面向最终用户的文档。

区分外部接口和内部实现

为了便于维护和协作，一个软件通常被划分为几个不同的部分。比如我们通常使用的MVC架构，把软件分为模型（**Model**）、视图（**View**）和控制器（**Controller**）三个部分。这样做，可以降低复杂度，让程序结构更加直观。同时，这种架构也很容易对程序进行修改和扩展，并且可以重复利用基础的功能。

不同功能的分离，让程序员之间产生了分工，专业人员可以更聚焦于个人的专长领域。这是一个多赢的局面，也能让软件的质量得到提升。

既然有分工，就要有协作。MVC架构把软件拆分为三块，是分工；而MVC模块之间的调用关系，就是协作。

一个好的软件设计，要区分外部接口和内部实现。外部接口，就是协作的界面，要简单规矩；内部实现，可以是千变万化的复杂小世界。

这种区分无处不在，即使是最普通的API。比如我们常用的InputStream，一旦我们获得这个对象实例，就可以调用它的read()方法。我们不用去关心，它的底层实现是一个文件，一段内存，

还是一个远程连接。`InputStream`的接口定义只有十个方法，短短的500多行代码。

但是它的内部实现却是一个更大的世界，广泛地分布在不同的类库、不同的模块，实现着不同的具体功能，有些实现甚至超出想象的复杂，比如一个安全连接的`_InputStream`的实现，一般有着数万行的代码。

幸运的是，我们区分了接口和实现，调用者就不用去关心这些复杂的实现了，只需要理解接口规范就好。

提高协作效率的最高技巧不是提高沟通技巧，而是要减少沟通的数量，提高沟通的质量，尤其是要减少数量。如果你参加了工作，没完没了的会议，没完没了的文案，都会加深你对这条原则的理解。软件的设计也是这样，外部接口，要少、要小、要描述清楚。

接口规范是协作合约

由于外部接口是协作的界面，是调用者和实现者之间的合约，所以对它就有了更加严格的要求。这里我总结了合约的四个原则：成文、清楚、稳定、变更要谨慎。

具体要怎么实践这些原则呢？

合约要成文

无论对于调用者，还是实现者来说，外部接口的使用都要有章可循，有规可依。如果调用者需要去看实现代码来理解外部接口，那么外部接口和内部实现的分离还有什么用呢？不就背离了外部接口和内部实现分离的初衷吗？这样做既是对实现者的纵容，也是对调用者的无视。

比如说，**Java**的每个版本的API文档和指南，就是**Java**语言的合约。

合约要清楚

合约既然是我们协作的依靠，就一定要清晰可靠、容易遵循，不能有模棱两可的地方。如果接口规范描述不清，既误导调用者，也误导实现者。

如果接口规范复杂难懂，说明接口的设计也很糟糕。

那么接口规范要怎么描述呢？

接口规范主要用来描述接口的设计和函数，包括确认边界条件、指定参数范围以及描述极端状况。比如，参数错了会出什么错误？

这里需要注意的是，接口规范不是我们定义术语、交代概念、提供示例的地方。这些应该在其他文档中解决，比如我们下次要聊的面向最终用户的文档。

合约要稳定

既然是合约，意味着调用者必须依赖于现有的规范。比如`InputStream.read()`这个方法，接口规范描述的是读取一个字节（8-bit），返回值是介于0和255之间的一个整数。如果我们要把这一个规范改成返回值是介于-128到127之间的一个整数，或者是读取一个字符（比如一个汉字），都会对现有的使用代码造成灾难性的影响。

接口的设计和规范的制定，一定要谨慎再谨慎，小心再小心，反复推敲，反复精简。一旦接口合约制定，公布，然后投入使用，就尽最大努力保持它的稳定，即使这个接口或者合约存在很多不足。

变更要谨慎

世界上哪里有一成不变的东西呢！技术的进步、需求的演进，总是推着我们朝前走。合约也需要跟得上变化。

可是，接口合约毕竟不是租房合约，可以一年一续，每年变更一次。租房合约的变更成本很小，但软件的接口合约变更的影响要严重得多。特别是兼容性问题，稍微一丁点儿的接口规范变化，都可能导致大面积的应用崩溃。越成功的接口，使用者越多，变更的影响也就越大，变更的成本也就变高，变更也就越困难。你可以试着想一想，如果`InputStream.read()`这个方法在Java中删除，会造成多大的影响？会有多少应用瘫痪？

所以，对于接口规范，我们的原则是，能不变更就不变更；必须的变更，一定要反复思量该怎么做才能把影响降到最低。

使用Java Doc

说完了接口规范的几个原则，我们就来讲一下，如何实践这些原则。接口的规范描述，应该怎么组织？

从使用者角度出发，包括接口的调用者和实现者，接口的规范应该便于阅读，便于查找。从制定者的角度出发，接口的规范应该便于定义，便于维护。

JavaDoc就是一种顾及了多方利益的一种组织形式。它通过文档注释的形式，在接口声明的源代码定义和描述接口规范。这种和源代码结合的方式，可以方便我们维护接口规范，也有利于保持接口规范和接口声明的一致性。

JavaDoc工具可以把文档注释，转换为便于阅读为HTML文档。这样就方便规范的使用者阅读了。

当然，也不是所有的规范，都一定要使用JavaDoc的形式，特别是冗长的规范。如果有两种以上不同形式的规范组织文档，我建议一定要互相链接、引用。比如，冗长的规范可以单独放在一个文件里。然后，在Java Doc对应的文件里，加上改规范的链接。

比如下面的例子中，“Java Security Standard Algorithm Names Specification”就是一个独立的，

较长的规范文档。当需要使用这个文档的时候，就要在对应的接口中指明该文档的位置，这样方便用户进行检索。

上面的文档注释，经过JavaDoc的处理，就变成了便于用户阅读的文字。

`protected Signature(String algorithm)`

Creates a Signature object for the specified algorithm.

Parameters:

algorithm - the standard string name of the algorithm. See the Signature section in the Java Security Standard Algorithm Names Specification for information about standard algorithm names.

谁来制定接口合约？

这本来不是一个问题。但是由于我们选择在源代码中，需要通过文档注释表达接口合约，这就成了一个很严肃的问题。

源代码的维护者，是不是对接口合约拥有无限的修改权利呢？

肯定不是的。

既然是合约，就是大家都认可并且接受的规范和细节，只有形成共识才能编辑和修订。合约的编写和修订，一般不应该由源代码的维护者一人决定，而应该由参与各方充分沟通和协商。

“三个臭皮匠，顶个诸葛亮”，我们要充分尊重参与各方的能力，信任充分的沟通可以成就更好的规范。

一个软件项目，不管大小，只要参与者超过两个，都要讨论清楚彼此之间的分工协作方式。这当然也包括，讨论清楚如何制定、修改程序接口。

比如，OpenJDK的接口制定和修订，就一定要经过下面的步骤：

1. 起草接口规范，或者起草提议的修订规范；
2. 找相关领域的专家，审议草案，并根据评审意见，修改接口规范；
3. 如果领域专家审议通过，提交兼容性和规范性审查程序；并根据审查意见，相应地修改接口规范；
4. 兼容性和规范性审查通过，修改接口合约；
5. 按照议定的接口规范，编写最终的实现的代码。

当然了，你的软件项目，也许和OpenJDK有巨大的差异。你要找到适合自己公司和项目的，接口合约制定和修改的适当方式。

小结

对于接口规范，我们要有意识地使用下面的这条原则：

接口规范是使用者和实现者之间的合约。

我们在工作过程中，如果有和接口相关的迷惑或者争执，可以多想一想上面的这条原则。

一起来动手

2018年12月25日，部分开发者突然发现他们开发的Web网页的界面发生了变化，按钮上方出现“积雪”。这超出开发者的脑洞和认知，难道是圣诞老人的礼物，或者是黑客的祝福？经过探索发现这是前端UI组件库Ant Design（简称antd）提前埋入一个未经声明的“彩蛋”。事件迅速发酵，引起了巨大争议。

前人的危机都是后人的财富。该怎么做，才可以避免类似的事情？欢迎你在讨论区留言，我们一起把这个事件转化成我们的见识和能力。

也欢迎点击“[请朋友读](#)”，把这篇文章分享给你的朋友或者同事，一起来交流。



代码精进之路

你写的每一行代码都是你的名片

范学雷

Oracle 首席软件工程师
Java SE 安全组成员
OpenJDK 评审成员



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有[现金](#)奖励。



MOV AX,0

👍 9

关于更新接口的问题，我所知的做法一般有两种：

1.对于外部部门调用的公开接口，如果有修改，需要提前知会各部门负责人。负责人委派一名同事进行对接，我们协调好接口规范整理出文档。在近期版本，上线这个新接口，但不马上废弃旧接口，只是标注@Deprecated。等待所有部门在后续版本替换完新接口后，检查接口调用情况，确认没有任何调用后进行移除；

2.如果是作为开放平台公开出去的接口，或在更改接口实现逻辑前需要额外流程（比如DB变更、数据源切换等），需要加入类似如下的逻辑：

```
if (isNewProcess()) {  
    return executeByNewProcess();  
}  
return executeByOldProcess();
```

目前公司是使用携程的Apollo配置中心实现公共配置，比如近期我们遇到一个查询会员账户总余额&积分统计的DB慢查询问题。我们将查询数据源改为从大数据获取，但是在大数据可能出错或挂掉的情况，就可能导致一系列问题。所以我在apollo配置了三个开关：

a.所有数据从大数据获取(boolean)

b.从大数据获取统计信息的商户ID(list)

c.所有统计数据直接返回0的商户ID(list)

容错性非常重要，如果大数据方面数据不可靠/接口挂掉，切到直查DB后，针对会员数很多的大商户，还需要直接返回0禁止DB的慢查询拖垮库。

同理，开放平台的接口，如果修改在线上的应用具有不确定性，一定要有后手，可以换回旧逻辑。测试环境通过，并不代表线上也通过！

2019-02-01

作者回复

非常棒的经验，多谢！

2019-02-01



周锐

👍 3

提供/调用接口甩锅指南：

1、提供接口：a、打印传入参数；b、对参数做验证，不合规就回抛异常；c、返回之前打印返回结果。

2、调用接口：a、调用前打印调用参数；b、调用后打印返回结果。

2019-02-01

作者回复

检查参数还不够吗？

2019-02-02



小文

👍 0

唉 我做游戏开发的从没有写过这个东东

2019-02-18

作者回复

忧桑，没接触过游戏开发，一点也不懂。游戏不开发公共接口吗？

2019-02-18



秦凯

👍 0

在接口声明中将“积雪”（节日）的样式作为配置参数，并且默认为不应用。只有当开发者主动配置时才会应用节日特效。并且将此特性记录到使用规范文档中供使用者参考，好让使用者可以清晰明了的使用API。

2019-02-14

作者回复

这是一个好办法！ "默认为不应用"是一个常用的解决兼容性问题的办法。

2019-02-14



逆风飞翔

👍 0

请问一下专栏内容有印象笔记版本吗

2019-02-01

作者回复

我没用过印象笔记，应该没有这个版本。

2019-02-01