入职Apifox研发组三个月，我领悟了30个高效开发方法🔥

🌞 前言

- 从去年入职 `Apifox` 研发组至今，已经有快**四个月**的时间了，今天来跟大家分享一下我的感想💭。
- 写这篇文章的契机是在年后我成功通过**三个月试用期**，也算是给自己一个交代，从自身出发来进行反思与总结在这**三个月**学到的东西与一些感想✍。

🤡 我学会了什么

- 在这**三个月内**我更多接受到的是工作或者说是写代码甚至作为程序员**应该掌握的方法**，而这些方法**不在于**你是什么技术栈或者说什么职位都很有用😳，所以今天我就来分享一些我总结的 `30` 个方法,我主要分成 💻代码方面 和 👨💻程序员方面 。

💻 代码方面

```
// life motto
if (sad() === true) {
 sad().stop();
 beAwesome();
}
```


学会表达

- 当你在写一个复杂的表达式甚至需要用到这些表达式来做判断的时候，这时候要养成一个习惯把表达式换成一个变量来表示👀，别人才能语义化的理解你这个表达式。
- 自己写的代码不管你写的再复杂你自己都可以看懂，但是对于别人不同，如果你把你的这些表达式赋值给一个变量,**这样别人只需要知道这个变量是什么意思就行了。**

学会复盘

- 中级与高级的程序员有一个差别是，优秀程序员肯定至少会花一些时间来**清理🧹自己的代码。**
- 这么做是因为，他们知道整洁好看的代码比杂乱无章的代码更容易修改，**甚至他们知道自己几乎无法一开始就写出整洁的代码🤷‍♀️。**
- 虽然我自己也没有完全严苛遵守，但是还是希望大家多去复盘，因为当你去看了你之前写的代码后你会发现很多乐趣（觉得自己写的代码可笑😁）。

拥抱变化

- 永远都不要说我写的功能 **"总是满足我们的需求"** ，在做项目开发特别是公共模块，你要学会**拥抱变化**😢， 永远要考虑变化的情况。
- 养成一个习惯，在做一个公共模块的时候要考虑后面有没有实现变化的可能或者能不能封装成一个 `js` 模块⭕，而不是直接用第三方库❌。
- 特别是我们是做客户端而不是简简单单的单页网页，而这里的公共模块指的是 **"使用者只用用到你的提供出来的 `API` 就知道怎么用了，并不需要使用者去考虑里面的实现"**

学会修BUG

- 很多人在接到一个 `BUG` 需求的时候经常只关注 **"眼前"**，即只关注遇到的问题而没有考虑其原本的意义。
- 你要做的其实不止是修改好这一个**缺陷**而是要去思考😉为什么他会出现这个**缺陷**，一定要关注上下文。
- 在修改一个旧的模块引发的 `BUG` 的时候,我们要保证不影响原来的功能逻辑，考虑清楚再 `commit` ，否则就会出现经常遇到的 **"改了 `BUG` 生成另一个 `BUG`** 😤**"**。

多用结构化数据

- 在你所做的一个组件逻辑**很复杂**的时候，更要考虑清楚它的结构，**多用结构化数据**，定义一个数据结构来存储中间状态，而不应该永远用简单的状态❌。
- **所有复杂的组件你的状态复杂是可以接受的，但是有多个状态是不可接受的。**

- 因为如果明明可以用结构化状态来存储，反而用多个状态组合实现的话，你这个组合关系就很复杂了。
- 确定一个你所需要的数据结构，**所有的操作都以这个数据结构为目标**，这个数据结构可以是一个对象可以是一个数组任何你期望的值，这样最后只需要拿到这个结构化数据来进行简单处理，所有的问题都迎刃而解了。

不要怕错

- 很多人包括我自己在进入一个新的公司或者面对一个别人正在开发的项目难免都会有一个问题：**"不够自信"**。
- 在看到别人的代码时候难免会看见别人明显的错误，而在当下你看来可能对自己编码不自信而不愿意帮忙改正，又怕改了之后被同事责怪，但这是不对的❌。
- 不要怕改错代码，**如果在当时看到一个错误或者命名不规范而你觉得有更好的名字或者重构方法可以及时改正**，就算是改错了那就在 `CodeReview` 的时候提出来让大家提建议。
- 这样在下次见到这段代码时就不要重复努力搞懂逻辑，因为很可能下次还会是你继续改动这个模块。

每次只关心一个上下文

- 如果某个模块经常因为不同的原因在不可预知的方向上发生变化，那么这个模块就变的很散乱了。
- 当你需要在原有基础上根据一种条件新增功能的时候，我可能需要更改三个函数，两个变量，这样的话就会很烦躁了，在如今这么内卷的时代再烦躁起来得有多难受😢。
- 我们要养成一种习惯每个函数只干一件事情，**将一个功能按照不同模块划分开来**，分成数个上下文而每次我们修改功能的时候只需要找到对应的模块来修改就很清晰了，我们只需要关注那一个上下文就可以了。

消灭注释

- 尽可能少的添加注释，特别是团队开发，有很多人不理解为什么？🤔我举个最简单的例子:**如果这个函数以后功能变了,除了我们需要改里面的代码还需要帮忙改变函数命名还要一起把之前的注释也改了**，假如一个函数有**5-6行**注释这么多，那你是不是每个都要改?
- 当然在不添加注释的前提下我们要保证我们函数命名变量命名**尽可能语义化**👌。
- 任何你觉得需要注释的地方， `99%` 是因为这段代码不合理。
- 每当你需要用注释来说明什么的时候，我们要做的不是用文字去解释这段代码，**而是把我们需要说明注释的东西写进一个独立函数，并以它的用途取个语义化的名字**✅。
- 甚至我们可以用短短几行的小函数来代替这件事情，哪怕之后调用函数的步骤变多，但是只要每个函数命名足够语义化，就能用代码合理的解释你想要做的一切。

- 函数的长度关键不在于有多长而是在于可读性，**在于"😐如何做"和"😖做些什么"**。

不畏注释

- 我们通过上面的方法可以有效的减少在代码内的注释但是对于别人已经遗留的注释我们应该置之不理吗?
- 如果在修改的功能模块发现原来存在注释，不要去抱怨，这对你没有一点好处，**相反的这可能是个很好的信号去告知我们应该去对这段代码进行修改**，而通过这些注释可以精准的帮我们定位到他的问题所在。

学会命名

- 如果在编写一个函数的时候你无法对其进行命名，**那么这个函数多数是设计不合理的**，就像我们上面提到的，函数里面表达的是 **"如何做"和"做些什么"** ，而我们对函数的命名也切记按照这个原则。
- 当你真正对一个函数准确命名，那么这里面的结构往往是**可读性高**的。
- 做任何业务都**不要把实现写在函数名**上，以后如果别人需要改动函数逻辑，还要帮你把函数名字给改动了。

合理入参

- 对于携带参数的函数的命名也是有讲究的，我们通常**可以使用通用的名字来代表参数命名**而不是使用在当下具体的名称，比如： `apiSelectedKey => selectedKey`
- 修改的好不仅能增加函数的应用范围，还能改变连接一个模块所需的条件，从而去除不必要的耦合。

学会提炼函数

- 在简化代码块的时候，我们最喜欢的就是提炼函数，**提炼函数可以让我们将意图与实现分开**。
- 并以意图命名函数，但是如果发现自己并不能合理命名，说明你不应该提炼这个函数  ，你应该考虑更多。

学会返回

- 上面讲了两个关于函数命名的方法，而往往在小函数内我们对于内部的变量会 **"词穷"**。
- 大部分的函数都可以使用 `result` 作为返回值，在函数开头定义，在函数结尾 `return` ，这样下次看这段代码的人一下就知道要返回的是什么🧐。

考虑时机

- 往往我们刚拿到一个新需求的时候很容易去实现它的功能，甚至不会去调研，修改 `BUG` 也是如此。
- 这是很容易的，任何一个程序员都可以做到，但是在什么时机做这个 **"动作"** 是应该考虑清楚的，点击的时候？依赖变化的时候？函数返回的时候？
- 我就吃过这个亏，**在不恰当的代码恰好完成了需求但是引发了一些不可控的缺陷**，导致我要重复读这整个模块的代码，不仅拖慢了项目进度还让自己多掉了几扎头发👦🌕。

保持可拓展性

- 在拿到一个新需求的时候永远永远不要想着做完就 `ok` ，因为你无法保证以后会不会让你在这个基础上增加新的功能甚至在你 `commit` 后的一个小时，产品的需求如果你没有优秀的产品思维是捉摸不透的😭，所以要给自己留 **"一条后路"**。
- 而这* 而最常见的无非就是各种判断，根据不同的判断来让程序走不同的代码，这时候我们不能写死判断而是多去使用 `map` 结构来保持功能可拓展性😎。

巧用模块

- 如果你发现两个模块之间**需要交流**，或许你可以新建一个模块📦来存放。
- 在做一个需求的时候，如果很多函数跟变量都是**从不同的第三方库引入**，这时候不妨把它归为一个模块，**就算是函数变量名字一摸一样都可以。**
- 只要保证是在这个模块里面导出就可以，这样后面的人更加方便维护这个模块，屏蔽这个实现。
- **使用的人不关心用，只关心结果**，就类似中间层🆗。

👦💻 程序员方面

巧用方法

- 有很多人想入行这个行业，**他要学的就是方法**，就比如说 **《重构》**，**《代码整洁之道》** 这些书讲的都是方法，教的就是方法💡。
- 你一个架构师写的代码好，去到一个新的项目怎么提现他的作用，还是**因为他的方法好，而不是他写过的代码有多好。**
- **只有你的方法好，你后面新产出的代码才会好。**

择善而从

- 当你觉得你在做重复的工作的时候，**一定是你做的方式不对，而不是这个任务不对。**
- 你任何一个任务都有可学习的地方，**我们做的是脑力活动，不存在像搬砖那样的情况**，一定会有更好的方案，只是你没有发现😐。
- 你可以通过研究别人的 app 看很多的源码，看别人的相同效果是怎么做的，**择善而从自己去实现一个新的方案。**
- **当你实现完了之后，你就是这种功能的大神**，以后再去面试别人公司的时候，你就说你在之前的公司，做了 xxx 功能的方案，吸收了业界很多类似 xxx 功能方案的长处，别人就会觉得你很牛逼，但是如果你只是说你之前重复的做 xxx 功能的某些事情那别人直接就把你刷掉了😔。

学会参考

- **作为程序员就是要不断看别人的代码**🧐，哪种代码写得好，哪种写得不好，哪些是值得去参考的，**自己心里要有一个底**，如果你不确定这个好不好，你就百度、谷歌、看源码。
- `API` 设计的过程中除非你是经验丰富的人，否则就得学会参考，任何一个程序员入门的方式都是抄袭，看别人怎么实现看别人代码怎么写。
- **一个入门程序员、一个中级程序员一上来就给你设计一套很舒服的 `API` 这是不可能的**🏅,参考开源大佬的项目,参考别人设计的时候为什么这么做,不能自己纯想。

产品思维

- **有产品思维是好的，能理解产品需求，能与产品进行有效沟通这是优势。**
- 这时候你就需要学会**跳出程序员的维度，拥有多学科交叉的能力**🫂，要去**了解**产品**提这个需求的目的**，因为在没了解的情况下，你按照自己的产品思维去认为这个需求的不同会导致想法过度或不及。
- 在不了解的情况下要多与产品交流，一定不要自己想当然，当你觉得应该怎么去做，**可以带方案去找产品讨论。**

适当做减法

- 你要想把一个需求做到 `100` 分可能是很难的，但是对于用户来说这个需求可能做到 `60` 分已经不错了,**剩下的 `40` 分就需要跟产品沟通什么时候把它做到 `100` 分**💯。
- 这也是所谓的什么时候该干什么样的事情，现阶段你做起来可能会很吃力花费很多时间，但或许 `2` 个月后的你来改这段代码轻而易举。

学会做需求

- **接到一个新需求要列方案、调研**👍，方案有很多种，看起来都很可行，但是实际上合不合适还是需要调研💥，否则做到一半后就要重新更换方案，不然就只能在产品上做妥协。
- 在明确了自己的方案并有一种想法的时候，**多去尝试，尝试使用它。**如果后来发现不太合适，完全可以再重来 完全没有问题，**只要在里面学到了东西，时间就不会白费。**就算你觉得不稳妥，在 `CodeReview` 提出来，自然会被指出错误。
- 当然我们在做一个需求的时候**不能局限于这个需求**，也就是我上面有提到的保持可拓展性，眼光要放长远，以后可能还会有人会用到你写的 `API` 。

学会问问题

- 养成一个习惯，当你要问人问题的时候，要知道每个人沟通能力不一样的，当你达到一种水平，觉得你能几句话能把一个问题说的很清晰，你可以不写，**如果你觉得你没有那个水平，你要写一下你的文字然后发给别人**，沟通协调能力也是在职场中需要锻炼的⚡。

- 很多时候你不能靠问，你必须要带方案，**如果你靠问别人给了你方案，那永远都是别人教你写代码。**
- 如果你自己带了方案，别人觉得方案不错，或者别人有更好的会提出来，当然别人花 `30s` 提出来的方案肯定不是最佳的，这是你负责的模块，你熟悉的逻辑，你想出来的方案肯定会更好，要在此抉择。

多用快捷键

- 花时间去看快捷键，千万不要在 `CodeReview` 或者技术分享会的时候找文件点来点去，在平时就要**养成追求"快"** 的习惯

一段时间干好一件事

- 这个点我以前一直都在提到，任何事情你都无法做到完美兼顾，**你要知道一段时间内能把一件事情干好就已经很不错了。**
- 如果你现在想要换工作，那就安心复习准备面试，如果你要学某个技术，那就围绕着它一直学，你可以给自己定个时间，**一个月？两个月？半年？** 🕐 没有关系，只要你在这段时间内用心去做了，时间会给你答案。

动手前提

- 在做公共业务和普通页面当你需要用到一个第三方库的时候，中级和高级的有一个差别是高级要**先大概把文档过一遍**，这样当你**遇到问题了可以快速定位，而不是遇到问题了再去找，再去看源码。**
- 这样不会花费很多很多时间，如果当你看文档的时候卡住了，你可以选择尝试看别人总结好解析好的的文章📖，说不定就帮你解决了问题，要记住看文章永远是比看源码快的。

快速定位问题

- 当你遇到一个 `BUG` 的时候一定要思考，不能莽做，缕清前因后果，思考为什么会引发这样的现象。
- 最合适的解决问题的路径是：**看上下文->看issue->看文档->看文章->看源码。**

⏳ 时间不知不觉 我们后知后觉

> 一生或许只是几页，不断在修改和誊抄着的诗稿，从青丝到白发，有人还在灯下。 ——席慕蓉

- 谁说不是呢？**时间如流水，一去不复返**，认真过好每一天也是我们力所能及的事情了。
- **在漫漫程序员历史长河中，几个月不过是浪花一朵**，学会时间管理对我们也极其重要。
- 我特别喜欢我老大的一句话：**"如果你是我招进来的而在你出去后你没有任何提升，我觉得我自己是失职的"**
- 在这**三个月**的时间里我**大多数的时间都花在了熟悉公司业务和代码规范上面**，在开始我深知我对未尝试的技术栈的不熟，对产品的不熟，再到后面逐渐熟练使用，我觉得对我自己成长真的不是一星半点😝。
- 这里的小伙伴真的是把一个产品当成自己的宝贝，细心呵护用规范的代码浇灌成长，我在大学当模特队队长的时候也一直在跟队员强调：我们能够在一起**不是因为我，不是因为你，不是因为他，而是"我们"**，只有我们达成了这种共识，才会越走越长远😶🌫️。
- 除了平时自愿留在公司学习，一周我会画一个晚上的时间回家整理 `CodeReview` 提到的点，也会抽两天早点回家看看书写写文，毕竟劳逸结合才是最好嘛。

- 我从来没说过我技术有多好，甚至我觉得我的技术跟很多人比起来差远了，可能屏幕前的你能够感同身受，**时间不停，我们当然也不能停滞不前。**
- 很多人都说**面试造火箭工作拧螺丝**，但是实际上工作中确实会造火箭，就比如我们，做的都是很有意思的事情，一群人都会忘乎所以的去往一个方向去做去成长，**但是在造火箭🚀的途中难免会拧一下螺丝🔩**，这时候我们就需要知道用什么螺丝，框架的底层是什么，做了什么就很重要了。
- 以上我所总结的方法都是这三个月切切实实学到的，**无论是在书里还是在工作中，基本上每一个方法都对应一个小故事**，这些我完完全全可以拆分成 `30` 篇文章来写，但是还是毫无保留的写出来了，如果大家对小故事感兴趣的话以后我也可以分享出来😄。