如何更加安全、高效地利用开源项目

在平时的开发过程中，难免会遇到这样那样的难题，或者一些繁琐且不想纯手工完成的功能，对于这些问题，解决的姿势有很多种，可以通过同事间的交流、上网查资料、去官网找文档等，随着开源的推动和完善，寻找合适的开源项目支持，绝对是一个很好的方法。

如今市面上的开源项目鱼龙混杂，并且有一些项目早已停止更新维护，跑demo的时候，怎么用怎么正确，一放入项目，却发现哪哪都不合适，比如低版本下才可以运行，高版本删去一些方法，再或者与一些新技术的包冲突等，在众多开源项目中，我们应该以何种姿势去选择最佳方案？且听我慢慢道来。

PS：目前我主要是做手机客户端开发，以下的例子会举一些日常开发的例子。

正确理解 确定需求

正确理解、确定需求，对开发中选取第三方开源或者原生都是有很大帮助，可以大大降低后期维护成本。

前段时间公司要接入摄像头来完成视频直播功能，采用m3u8格式实现，后期也不会有其它格式接入，从开发者文档中看到，Android原生控件仅支持MP4格式，对于其它格式的兼容，还存在很大问题，也就是说没法完美兼容m3u8。此时，我们仅需要找到m3u8格式支持的方案就好，在搜索的征途中，我们大大地缩小的范围。

兼容性

做客户端开发，对于兼容性的话题，总是有千言万语，从最初写布局到后来的集成，再到后来系统的升级，无时无刻不在与兼容性打交道，尤其是在swift刚诞生的那一两年，每个大版本都会大改api，这就有点扯淡了，搞的开发者心力交瘁，企业也不敢轻易尝试使用。

对于布局的适配的兼容性，今天不做讨论，接着上面视频对接的话题继续撸下去，经过一番对比，我把目标锁定在了Google的ExoPlayer和bilibili的ijkplayer，这两个都是开源项目，都可以在github找到源码，地址如下，点击可跳转：

ijkplayer的github地址：https://github.com/Bilibili/ijkplayer

ExoPlayer的github地址：https://github.com/google/ExoPlayer

我最初的选择方案更是倾向于ijkplayer，对国内开源项目本来就有一种特别的情感，国货当自强，和朋友讨论的时候，几个朋友也是推荐我ijkplayer，这其中也有B站的哥们，但他没参与到这个开源项目中，在最初跑demo的时候，确实可以完美播放m3u8，当我把他放到我们项目中去的时候，发现报错了，再次检查下导入的包，如下：

```
dependencies {
# required, enough for most devices.
compile 'tv.danmaku.ijk.media:ijkplayer-java:0.8.3'
compile 'tv.danmaku.ijk.media:ijkplayer-armv7a:0.8.3'
```

```
        defaultConfig {
            minSdkVersion 21
            targetSdkVersion rootProject.ext.targetSdkVersion
            }
```

最低支持21的版本，和项目有冲突，解决方案有三种，不导入64位的包，或者自己修改源码后重新编译，最后一种不现实，就是改公司项目的最低版本，我们目前还是有一部分Android 4.3的用户。

一定会有更好的方案，嗯，一定会有的，抱着试试看的态度，无意间发现了ExoPlayer。

通过一番查阅资料，在ExoPlayer的开发者文档中可看到

> ExoPlayer's standard audio and video components rely on Android's MediaCodec API, which was released in Android 4.1 (API level 16). Hence they do not work on earlier versions of Android. Widevine common encryption is available on Android 4.4 (API level 19) and higher.

最低支持的版本是Android 4.1，公司目前项目，最低支持是Android 4.2，挺符合我们的口味。

总结：兼容性一直是一个很繁琐的问题，版本更新太快，技术不断更新换代，一些不安全或者没必要的方法，不断的从API中删去，接着迎来了一些新加入的API，合理的考察和调研，可以省去很多弯路。

对于健全性，主要体现在文档的健全性和资料的健全性，如果是一个全新的技术，官方没有提供健全的API，市面上还没有一些集成文档，这类的开源项目，最好别介入到项目中，作为第一个尝试吃蜘蛛的人，可以吃到的是一嘴的苦水。

最初我在测试ijkplayer的时候，首先，简单浏览了下内容，知道了个大概采用什么技术，接着就去找文档，天哪，我只看到了集成时候需要导入的包和编译采用的环境、工具，还有，就是我只找到了sample。这就尴尬了，这么大而优秀的项目，居然没有官方文档，瞬间好感下降到了负一层，去网上搜了下资料，资料还是很健全的，有很多优秀的开发者在集成后，把一些必要的注释就加上去了，看起来一目了然。

后来在调研Google的ExoPlayer的时候，发现有了质的差距，如下：

ExoPlayer 源码地址：

https://github.com/google/ExoPlayer

ExoPlayer api地址：

http://google.github.io/ExoPlayer/doc/reference/

ExoPlayer 开发者指南：

https://google.github.io/ExoPlayer/guide.html

这真的太棒了，集成的时候，可以解决少走很多弯路，遇到问题，也有了一些解决方案，扩展的时候，也为自己增加了几分信心。

想起了中学时候学校的一句标语：“细节决定成败，态度决定高度”。在平时使用或者学习第三方开源项目的时候，实现功能并不是第一要素，更应该关注API和资料的健全性，这点，我一直很欣赏Google和square，Google先不提，在学习square的retrofit的时候，API还是给了我很大的帮助，里面写的确实挺详细，一目了然，每个注解都有详细的说明，让学习者感到更加亲切。

实现原理

确实是这样子的，就像三年多前在一家外包公司的时候，那时候市面上还没有太多关于即时聊天的文档，当时的第三方也没那么多选择，公司决定使用xmpp + openfire去实现，而那时候我们团队没人接触过这一块内容，项目在爬行中推动，一个加好友的功能，把一个同事搞了好几天（这不否认那位同事的能力和粗心，因为英语的问题，没去看官方文档，同时也没过多的做调研）。

我很欣赏曹操，很欣赏他的疑人不用，用人不疑。如果曹操是一个程序员，一定是一位很优秀的代码家，以曹操的性格，一定会把一门技术吃透，然后再学以致用，举一反三，尽可能把一门技术发挥到极致。

性能

不管采用什么样的开源项目，性能绝对是一个很重要的参考，就算这个项目写得再好，功能齐全，一旦性能有问题，这将是致命一击，就像一个失去双手的运动员，长、短跑都是第一，此时让他去参加乒乓球比赛，再牛逼的教练也是一脸懵逼。

回到刚才的话题，再ExoPlayer和ijkplayer中的选择，一大部分原因是因为性能方面的问题，性能问题，直接把矛头指向了硬解码和软解码的区别。

> 硬解码：就是调用GPU的专门模块进行解码，由显卡核心GPU来对视频进行解码工作。
>
> 软解码：通过软件让CPU来对视频进行解码处理。

相信对于软解码和硬解码，大部分人都不陌生，这里不做多与赘述。

一图胜千言，做了一下对比：

	内存消耗	性能	支持格式	流畅度	画质	功耗	总功耗
硬解码	低	好	多	好	差	高	低
软解码	高	差	无限制	差	好	低	高

在处于性能过剩的时代，CPU已经很难处于负荷状态，选择软解码或者硬解码都是没有谁对谁错，刚刚图上已经贴出和标记两者的优点，根据项目需要选择。

当时选择硬解码的ExoPlayer，是因为只需要播放m3u8格式的视频，画面上没有那么高的追求，对于这样的需求，硬解码更符合公司的口味和用户的体验，至少可以节省更多的电量。

功能与扩展

对于产品经常改需求，这是常有的事，我们更多的时间不应该是放在和产品经理撕逼，而是如何更好的应对这个问题。

刚学软件开发的时候，书上就提到，好的程序员，会更好的考虑代码的可维护性、可重用性、可扩展性和灵活性。

再接入一个第三方之前，熟悉它的功能是在所难免的事，就是因为某个和某几个功能吸引才导致我们采用这个第三方，为了方便以后扩展，还是应该多读几遍开发者文档，尽可能多的了解和熟悉内部结构，在开发者文档，一般都会有详细的说明。

对于扩展性，这个就离不开文档和源码了，总不能自己莫名其妙的写代码，这不现实，我觉得每个合格的程序员都应该养成阅读源码的好习惯，这对自身的提高和功能的扩展都有很大的帮助。

集成性

对于集成性，我第一反应就是想到了微信和支付宝支付，这简直就是天壤之别，两年前的时候，我和一个同事分别接入支付宝和微信支付，我接入支付宝，加上阅读文档和跑demo，不到一个小时就跑通了，而他接入微信支付的时候，接了一天还没搞定，然后和我说微信支付的各种坑，当时也是半信半疑，就算坑了点，也不至于需要一天多吧，就这么点内容。后来有一次接了个私活，也是需要接入支付宝和微信支付，支付宝的文档

总结

如何更加安全、高效地利用第三方开源项目，为了提高以后代码的可维护性和可扩展性，我们应该更多的去调研和阅读开发者文档，磨刀不误砍柴工，一个好的开发者，应该把更多的时间放在思考和调研，而不是速度完成需求，然后把更多的时间放在改 bug。