# 1.cocoapods升级背景

- 目前部分三方库已经不支持低版本cocoapods，比如Flutter。在Flutter开发环境下执行flutter doctor命令可以看到Flutter要求至少1.6.0版本的cocoapods，而我们目前使用的是1.2.1版本。

```
Doctor summary (to see all details, run flutter doctor -v):
[!] Xcode - develop for iOS and macOS (Xcode 10.2)
    ! CocoaPods out of date (1.6.0 is recommended).
        CocoaPods is used to retrieve the iOS and macOS platform sides plugin
        code that responds to your plugin usage on the Dart side.
        Without CocoaPods, plugins will not work on iOS or macOS.
        For more info, see https://flutter.dev/platform-plugins
      To upgrade:
        sudo gem install cocoapods
```

- 经过对比测试，升级高版本的cocoapods可以降低主工程pod update&build总耗时约1分钟，可以提高我们的开发效率。

### 1.1 升级出现的问题

主工程在1.8.4版本cocoapods下执行pod install命令安装组件库并运行工程，编译失败。如图所示，在Pods/MGJ-Categories/UIDevice+MGJKit.h文件中报错。

### 1.2 报错原因查找

一般我们导入组件库报错找不到"xxx.h"文件，可能是因为头文件路径没有配置好。如图所示，找到Pods Target->MGJ-Categories->Build Settings->Search Paths，可以发现所有依赖的头文件路径都已经被添加了，报错与MGJ-Categories头文件路径配置无关。

展开详细日志，可以看到上一层调用者是MGJHangDetector库。

```
In file included from /Users/gemini/Desktop/mogujie-iphone/Pods/MGJHangDetector/MGJHangDetector/Classes/MGJHangDetector.m:10:
/Users/gemini/Desktop/mogujie-iphone/Pods/MGJ-Categories/MGJCategories/Foundation/UIDevice+MGJKit.h:9:9: fatal error: 'AFNetworking/
  AFNetworkReachabilityManager.h' file not found
#import <AFNetworking/AFNetworkReachabilityManager.h>
        ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1 error generated.
```

同样检查Pods Target->MGJHangDetector-Categories->Build Settings->Search Paths，MGJHangDetector库的头文件路径配置是空的，再查看Build Phases->Dependencies，发现该库对MGJ-Categories库的依赖也没有加上。

将高低两个版本的主工程进行对比，发现低版本中MGJHangDetector库同样没有添加MGJ-Categories库的依赖，Framework_Search_Paths也是空的。但展开Header_Search_Paths，其中添加了所有库的搜索路径，且都在Pods/Headers/Public路径下。

而高版本工程的Pods/Headers/Public文件夹下，所有.framework库都没有生成相应的头文件目录。查看源码可以看出新版本cocoapods在这方面做了一些处理。

**1.3 解决方案**

在确定问题后，我们从Podfile、cocoapods源码、podspec三方面入手找到了几种解决方案：

- Podfile中添加hooks

  因为MGJHangDetector库没有添加依赖库的搜索路径，因此我们可以利用hooks手动添加，代码如下，添加后工程成功编译。

```
if target.name == "MGJHangDetector"
    target.build_configurations.each do |config|
        config.build_settings['FRAMEWORK_SEARCH_PATHS'] ||=
['$(inherited)', '${PODS_ROOT}/AFNetworking']
    end
end
```

- cocoapods源码修改

  cocoapods/pod_target.rb中的header_mappings方法，对工程Header生成做了处理，屏蔽了所有".framework"库。因此我们可以注释cocoapods源码，使Pods/Headers/Public目录下生成".framework"库的Header。源码如下：

```
headers.each do |header|
        next if header.to_s.include?('.framework/')

        sub_dir = dir
        if header_mappings_dir
          relative_path =
header.relative_path_from(file_accessor.path_list.root +
header_mappings_dir)
          sub_dir += relative_path.dirname
        end
        mappings[sub_dir] ||= []
        mappings[sub_dir] << header
      end
```

- podspec字段修改

  对MGJPodSpecs源中MGJHangDetector.podspec.json文件的所有字段进行修改排查。尝试补全source_files、public_header_files两个字段值的路径，对Pods安装没有影响。

  添加static_framework字段，并设置值为true，代码如下。pod update后MGJHangDetector库添加上了依赖,并且编译成功。

```
"static_framework" : true
```

- nodependence源中移除bug

为了加快编译速度，主工程添加有加速专用source。找到mgjpodspec-nodependence源中MGJHangDetector.podspec.json文件，发现dependencies字段被置空了。可以确定这是一个bug，导致高版本与低版本cocoapods中主工程都出现依赖没有添加的问题。

将mgjpodspec-nodependence源中的MGJHangDetector库移除，pod update后依赖成功添加，编译工程不再报错。

**综上：**几个方案对比，显然对Podfile、cocoapods源码、podspec进行修改都是有成本的，会对swift开发等造成一定影响。而删除加速源中MGJHangDetector的podspec是没有成本的，因此可以确定为最终方案。

#### 1.4 依赖添加失败原因

1. Podfile中source的优先级是由上至下的，相同版本的库以第一个找到该库的源为准。
2. Podfile中所有的源如下，可以看到mgjpodspec-nodependence源处在第一位。

```
# 主客加速专用source，组件工程中禁止添加这一行
source 'http://gitlab.mogujie.org/ios-team/mgjpodspec-nodependence.git'
# 三方库framework源，注释该行即可使用源码
source 'http://gitlab.mogujie.org/ios-team/3rdpartyframeworkspec.git'
source 'http://gitlab.mogujie.org/ios-team/mgjpodspec-framework.git'
source 'http://gitlab.mogujie.org/ios-team/mgjpodspecs.git'
source "http://gitlab.mogujie.org/CocoapodsRepos/CocoapodsRepoSpecs.git"
# IM 源
source 'http://gitlab.mogujie.org/im/IMPodSpecs.git'
```

## 2. 升级对业务的影响

> cocoapods升级可以降低我们主工程pod update&build总耗时接近1分钟，约13%。以下列出了1.2.1版本和1.8.4版本各部分耗时对比测试。其中pod update主要包括Analyzing、Downloading、Generating、Integrating四个过程，其它的耗时基本可以忽略不计。build主要包括precompile、linking、run script三个过程，其它的耗时同样可以忽略不计。测试电脑型号是MacBook Pro (13-inch, 2015,16G内存)。

#### 2.1 编译时间对比

| | 1.8.4 | 1.2.1 | 平均耗时对比 |
|---|---|---|---|
| 总时长 | 298.8s | 300s | 平均编译总时长基本相同 |
| precompile | 10.8s | 11.7s | precompile耗时基本相同 |
| linking | 86.5s | 84.2s | linking耗时基本相同 |
| run script | 143.3s | 124.8s | run script耗时增加15% |

**结论**:编译总时长基本接近，1.8.4版本在run script过程耗时较长一些。

#### 2.2 重复编译时间对比

> 重复编译的意思是指执行pod update并完成第一次编译后，不做clean、pod update等操作，只修改工程代码，继续编译测试。

| | 1.8.4 | 1.2.1 | 平均耗时对比 |
|---|---|---|---|
| 总时长 | 87.6s | 236.3s | 总耗时减少63% |
| precompile | - | - | 都不需要precompile |
| linking | 79.0s | 87.1s | linking耗时减少9% |
| run script | - | 137.2s | 1.8.4不需要run script |

**结论**:1.2.1版本重复编译，会重新进行linking和run script过程。而1.8.4版本重复编译不需要重新进行run script过程，所以编译总耗时减少63%。但每次执行pod update后，整体编译过程都会重新进行。

**2.3 pod update耗时对比**

| | 1.8.4 | 1.2.1 | 平均耗时对比 |
|---|---|---|---|
| 总时长 | 74s | 131s | 总耗时降低43.5% |
| Analyzing dependencies | 9s | 29.5s | Analyzing耗时降低69.5% |
| downloading | 11s | 19s | - |
| Generating Pods project | 39s | 60s | Generating耗时降低35% |
| Integrating client project | 3s | 9s | Integrating耗时降低66.7% |

**结论**:1.8.4版本在pod update耗时上降低了43.5%。除了downloading耗时与更新包数据及网速相关，Analyzing、Generating、Integrating等过程都有不同比例的耗时降低。

**2.4 耗时总结**

经过详细的对比测试及计算后，pod update&build整体耗时降低了57s，约13.2%。重复编译耗时降低148.7s，约63%，即减少了run script过程的时间。综上可见，升级高版本的cocoapods对我们的业务开发效率也是有积极影响的。