你真的会写Podfile吗?--关于Podfile的详细总结

前言

iOS开发会经常用到cocoapods管理第三方，简单、方便、高效。如何集成cocoapods在cocoapods官网和Podfile语法说明会有详细介绍，本文我想介绍的是关于集成cocoapods时会用到的一个文件Podfile文件。

什么是Podfile

Podfile是一个规范，描述了一个或多个一套工程目标的依赖项

一个简单写法:

```
target 'MyApp' do
  pod 'AFNetworking', '~> 3.0'
end
```

这是最简单最普遍的写法，针对MyApp这个target引入AFNetworking这个依赖库，也是大家平时用的最多的一种方式。

下面是个更复杂的一个例子：

```
# 下面两行是指明依赖库的来源地址
source 'https://github.com/CocoaPods/Specs.git'
source 'https://github.com/Artsy/Specs.git'

# 说明平台是ios，版本是9.0
platform :ios, '9.0'

# 忽略引入库的所有警告（强迫症者的福音啊）
inhibit_all_warnings!

# 针对MyApp target引入AFNetworking
# 针对MyAppTests target引入OCMock,
target 'MyApp' do
    pod 'AFNetworking', '~> 3.0'
    target 'MyAppTests' do
       inherit! :search_paths
       pod 'OCMock', '~> 2.0.1'
    end
end
# 这个是cocoapods的一些配置,官网并没有太详细的说明,一般采取默认就好了,也就是不写.
post_install do |installer|
   installer.pods_project.targets.each do |target|
     puts target.name
   end
end
```

主配置

`install!` 这个命令是cocoapods声明的一个安装命令，用于安装引入Podfile里面的依赖库。
`install!` 这个命令还有一些个人设置选项，例如：

```
install! 'cocoapods',
  :deterministic_uuids => false,
  :integrate_targets => false
```

还支持其他的选项：

```
Supported Keys:

:clean

:deduplicate_targets

:deterministic_uuids

:integrate_targets

:lock_pod_sources

:share_schemes_for_development_pods
```

关于以上的配置，官网也没有一个确切的说明，所以我们只需用系统默认即可。

Dependencies（依赖项）

Podfile指定每个target的依赖项

- `pod` 指定特定的依赖库
- `podspec` 可以提供一个API来创建podspecs
- `target` 通过target指定依赖范围

pod - 指定项目的依赖项

依赖项规范是由Pod的名称和一个可选的版本组合一起。

1> 如果后面不写依赖库的具体版本号，那么cocoapods会默认选取最新版本。

`pod 'SSZipArchive'`

2> 如果你想要特定的依赖库的版本，就需要在后面写上具体版本号，格式：

```
pod 'Objection', '0.9'
```

3> 也可以指定版本范围

- `> 0.1` 高于0.1版本（不包含0.1版本）的任意一个版本
- `>= 0.1` 高于0.1版本（包含0.1版本）的任意一个版本
- `< 0.1` 低于0.1版本（不包含0.1版本）的任意一个
- `<= 0.1` 低于0.1版本（包含0.1版本）的任意一个
- `~> 0.1.2` 版本 0.1.2的版本到0.2 ，不包括0.2。这个基于你指定的版本号的最后一个部分。这个例子等效于>= 0.1.2并且 <0.2.0，并且始终是你指定范围内的最新版本。

关于版本形式规范详情请参考链接：语义化版本

Build configurations（编译配置）

默认情况下， 依赖项会被安装在所有target的build configuration中。为了调试或者处于其他原因，依赖项只能在给定的build configuration中被启用。

下面写法指明只有在Debug和Beta模式下才有启用配置

```
pod 'PonyDebugger', :configurations => ['Debug', 'Beta']
```

或者，可以弄白名单只指定一个build configurations。

```
pod 'PonyDebugger', :configuration => 'Debug'
```

注意：默认情况下如果不指定具体生成配置，那么会包含在所有的配置中，如果你想具体指定就必须手动指明。

Subspecs

一般情况我们会通过依赖库的名称来引入，cocoapods会默认安装依赖库的所有内容。
我们也可以指定安装具体依赖库的某个子模块，例如：

```
# 仅安装QueryKit库下的Attribute模块
pod 'QueryKit/Attribute'
# 仅安装QueryKit下的Attribute和QuerySet模块
pod 'QueryKit', :subspecs => ['Attribute', 'QuerySet']
```

Using the files from a local path (使用本地文件)

我们也可以指定依赖库的来源地址。如果我们想引入我们本地的一个库，可以这样写：

```
pod 'AFNetworking', :path => '~/Documents/AFNetworking'
```

使用这个选项后，Cocoapods会将给定的文件夹认为是Pod的源，并且在工程中直接引用这些文件。这就意味着你编辑的部分可以保留在CocoaPods安装中，如果我们更新本地AFNetworking里面的代码，cocoapods也会自动更新。

被引用的文件夹可以来自你喜爱的SCM，甚至当前仓库的一个git子模块

注意：Pod的podspec文件也应该被放在这个文件夹当中

From a podspec in the root of a library repository (引用仓库根目录的podspec)

有时我们需要引入依赖库指定的分支或节点，写法如下。

- 引入master分支（默认）

```
pod 'AFNetworking', :git => 'https://github.com/gowalla/AFNetworking.git'
```

- 引入指定的分支

```
pod 'AFNetworking', :git => 'https://github.com/gowalla/AFNetworking.git', :branch => 'dev'
```

- 引入某个节点的代码

```
pod 'AFNetworking', :git => 'https://github.com/gowalla/AFNetworking.git', :tag => '0.7.0'
```

- 引入某个特殊的提交节点

```
pod 'AFNetworking', :git => 'https://github.com/gowalla/AFNetworking.git', :commit => '082f8319af'
```

需要特别注意的是，虽然这样将会满足任何在Pod中的依赖项通过其他Pods 但是podspec必须存在于仓库的根目录中。

从外部引入podspec引入

podspec可以从另一个源库的地址引入

```
pod 'JSONKit', :podspec => 'https://example.com/JSONKit.podspec'
```

podspec

使用给定podspec文件中定义的代码库的依赖关系。如果没有传入任何参数，podspec优先使用根目录，如果是其他情况必须在后面指明。(一般使用默认设置即可)例如：

```
# 不指定表示使用根目录下的podspec，默认一般都会放在根目录下
podspec
# 如果podspec的名字与库名不一样，可以通过这样来指定
podspec :name => 'QuickDialog'
# 如果podspec不是在根目录下，那么可以通过:path来指定路径
podspec :path => '/Documents/PrettyKit/PrettyKit.podspec'
```

target

在给定的块内定义pod的target（Xcode工程中的target）和指定依赖的范围。一个target应该与Xcode工程的target有关联。默认情况下，target会包含定义在块外的依赖，除非指定不使用inherit!来继承（说的是嵌套的块里的继承问题）

- 定义一个简单target `ZipApp`引入`SSZipArchive`库

```
target 'ZipApp' do
  pod 'SSZipArchive'
end
```

- 定义一个`ZipApp`target仅引入`SSZipArchive`库，定义`ZipAppTests`target 引入`Nimble`的同时也会继承`ZipApp`target里面的`SSZipArchive`库

```
target 'ZipApp' do
  pod 'SSZipArchive'
  target 'ZipAppTests' do
    inherit! :search_paths
    pod 'Nimble'
  end
end
```

- target块中嵌套多个子块

```
target 'ShowsApp' do
    # ShowsApp 仅仅引入ShowsKit
    pod 'ShowsKit'
    # 引入 ShowsKit 和 ShowTVAuth
    target 'ShowsTV' do
```

```
        pod 'ShowTVAuth'
    end
    # 引入了Specta和Expecta以及ShowsKit
    target 'ShowsTests' do
        inherit! :search_paths
        pod 'Specta'
        pod 'Expecta'
    end
end
```

抽象target

定义一个新的抽象目标，它可以方便的用于目标依赖继承。

- 简单写法

```
abstract_target 'Networking' do
    pod 'AlamoFire'
    target 'Networking App 1'
    target 'Networking App 2'
end
```

- 定义一种abstract_target包含多个target

```
# 注意：这是个抽象的target也就是说在工程中并没有这个target引入ShowsKit
abstract_target 'Shows' do
    pod 'ShowsKit'
    # ShowsiOS target会引入ShowWebAuth库以及继承自Shows的ShowsKit库
    target 'ShowsiOS' do
        pod 'ShowWebAuth'
    end
    # ShowsTV target会引入ShowTVAuth库以及继承自Shows的ShowsKit库
    target 'ShowsTV' do
        pod 'ShowTVAuth'
    end
    # ShowsTests target引入了Specta和Expecta库，并且指明继承Shows，
所以也会引入ShowsKit
    target 'ShowsTests' do
```

```
        inherit! :search_paths
        pod 'Specta'
        pod 'Expecta'
    end
end
```

abstract! 和 inherit!

- abstract! 指示当前的target是抽象的，因此不会直接链接Xcode target。
- inherit! 设置当前target的继承模式。例如：

```
target 'App' do
   target 'AppTests' do
      inherit! :search_paths
   end
end
```

Target configuration (目标项配置)

使用target 配置来控制的cocoapods生成project。
开始时详细说明您正在使用什么平台上。工程文件里允许您具体说明哪些项目的链接。

platform

platform用于指定应建立的静态库的平台。CocoaPods提供了默认的平台版本配置：

- iOS->4.3
- OS X->10.6
- tvOS->9.0
- watchOS->2.0

如果部署目标需要iOS < 4.3，armv6体系结构将被添加到ARCHS。
例如:

```
#指定具体平台和版本
platform :ios, '4.0'
platform :ios
```

project

如果没有显示的project被指定，那么会默认使用target的父target指定的project作为目标。如果如果没有任何一个target指定目标，那么就会使用和Podefile在同一目录下的project。同样也能够指定是否这些设置在release或者debug模式下生效。为了做到这一点，你必须指定一个名字和:release/:debuge关联起来

Examples:
Specifying the user project

```
# MyGPSApp这个target引入的库只能在FastGPS工程中引用
target 'MyGPSApp' do
    project 'FastGPS'
    ...
end
# 原理同上
target 'MyNotesApp' do
    project 'FastNotes'
    ...
end
```

使用自定义的编译配置

```
project 'TestProject', 'Mac App Store' => :release, 'Test' => :debug
```

inhibit_all_warnings!（强迫症者的福音）

inhibit_all_warnings! 屏蔽所有来自于cocoapods依赖库的警告。你可以全局定义，也能在子target里面定义，也可以指定某一个库：

```
# 隐藏SSZipArchive的警告而不隐藏ShowTVAuth的警告
pod 'SSZipArchive', :inhibit_warnings => true
pod 'ShowTVAuth', :inhibit_warnings => false
```

use_frameworks!

通过指定use_frameworks!要求生成的是framework而不是静态库。
如果使用use_frameworks!命令会在Pods工程下的Frameworks目录下生成依赖库的framework
如果不使用use_frameworks!命令会在Pods工程下的Products目录下生成.a的静态库

Workspace

默认情况下，我们不需要指定，直接使用与Podfile所在目录的工程名一样就可以了。如果要指定另外的名称，而不是使用工程的名称，可以这样指定：

`workspace 'MyWorkspace'`

Source

source是指定pod的来源。如果不指定source，默认是使用CocoaPods官方的source。(建议使用默认设置)

```
CocoaPods Master Repository
# 使用其他来源地址
source 'https://github.com/artsy/Specs.git'
# 使用官方默认地址（默认）
source 'https://github.com/CocoaPods/Specs.git'
```

Hooks

Podfile提供了hook机制，它将在安装过程中调用。hook是全局性的，不存储于每个target中。

Plugin

指定应在安装期间使用的插件。使用此方法指定应在安装期间使用的插件，以及当它被调用时，应传递给插件的选项。例如：

```
# 指定在安装期间使用cocoapods-keys和slather这两个插件
plugin 'cocoapods-keys', :keyring => 'Eidolon'
plugin 'slather'
```

pre_install

当我们下载完成，但是还没有安装之时，可以使用hook机制通过pre_install指定要做更改，更改完之后进入安装阶段。
格式如下：

```
pre_install do |installer|
    # 做一些安装之前的更改
end
```

post_install

当我们安装完成，但是生成的工程还没有写入磁盘之时，我们可以指定要执行的操作。比如，我们可以在写入磁盘之前，修改一些工程的配置：

```
post_install do |installer| installer.pods_project.targets.each
 do |target|
        target.build_configurations.each do |config|
            config.build_settings['GCC_ENABLE_OBJC_GC'] = 'supp
orted'
        end
    end
end
```

def

我们还可以通过def命令来声明一个pod集：

```
def 'CustomPods'
   pod 'IQKeyboardManagerSwift'
end
```

然后，我们就可以在需要引入的target处引入：

```
target 'MyTarget' do
   CustomPods
end
```

这么写的好处是：如果有多个target，而不同target之间并不全包含，那么可以通过这种方式来分开引入。