| COMMENTS

使用Cocoapods创建私有podspec

Cocoapods是非常好用的一个iOS依赖管理工具，使用它可以方便的管理和更新项目中所使用到的第三方库，以及将自己的项目中的公共组件交由它去管理。Cocoapods的介绍及优点本文就不在赘述，我开始使用Cocoapods还是在两年前，那个时候它刚刚出现，网上的资料还非常的少，就连他们自己的HomePage都十分的简单，我就着手尝试着使用了一下，用它管理起第三方库确实是十分的方便顺手。后来它有了更强大的功能就是自己创建podspec，更可以设置私有的库。

春节回来上班，一天的工作结束之后，需要充实下自己，正好项目中有一些公共组件需要从庞大的项目体系中剥离出来，而且年前项目终于从SVN迁移到了Git，真是喜大普奔，大快人心！这样项目使用Cocoapods就有了条件，正好学习一下创建私有的podspec并在项目中部署使用，以及pods的subspec的创建及使用。

整体先说明一下创建一个私有的podspec包括如下那么几个步骤:

1. 创建并设置一个私有的Spec Repo。
2. 创建Pod的所需要的项目工程文件，并且有可访问的项目版本控制地址。
3. 创建Pod所对应的podspec文件。
4. 本地测试配置好的podspec文件是否可用。
5. 向私有的Spec Repo中提交podspec。
6. 在个人项目中的Podfile中增加刚刚制作的好的Pod并使用。
7. 更新维护podspec。

在这一系列的步骤中需要创建两个Git仓库，分别是第一步和第二步（第二步不一定非要是Git仓库，只要是可以获取到相关代码文件就可以，也可以是SVN的，也可以说zip包，区别就是在podspec中的source项填写的内容不同），并且第一步只是在初次创建私有podspec时才需要，之后在创建其他的只需要从第二步开始就可以。本文只介绍在Git环境下的操作，其他环境其他方式暂不说明。

创建私有Spec Repo

先来说第一步，什么是Spec Repo？他是所有的Pods的一个索引，就是一个容器，所有公开的Pods都在这个里面，他实际是一个Git仓库 remote端
在GitHub上，但是当你使用了Cocoapods后他会被clone到本地的~/.cocoapods/repos目录下，可以进入到这个目录看到master文件夹就是这个官方的Spec Repo了。这个master目录的结构是这个样子的

```
.
├── Specs
    └── [SPEC_NAME]
        └── [VERSION]
            └── [SPEC_NAME].podspec
```

因此我们需要创建一个类似于`master`的私有`Spec Repo`，这里我们可以`fork`官方的`Repo`，也可以自己创建，个人建议不`fork`，因为你只是想添加自己的`Pods`，没有必要把现有的公开`Pods`都`copy`一份。所以创建一个`Git仓库`，这个仓库你可以创建私有的也可以创建公开的，不过既然私有的`Spec Repo`，还是创建私有的仓库吧，需要注意的就是如果项目中有其他同事共同开发的话，你还要给他这个`Git仓库`的权限。因为`GitHub`的私有仓库是收费的，我还不是`GitHub`的付费用户，所以我使用了其他`Git`服务，我使用的是`CODING`，当然还有其他的可供选择`开源中国`、`Bitbucket`以及`CSDN`

创建完成之后在`Terminal`中执行如下命令

```
 # pod repo add [Private Repo Name] [GitHub HTTPS clone URL]
$ pod repo add WTSpecs https://coding.net/wtlucky/WTSpecs.git
```

此时如果成功的话进入到`~/.cocoapods/repos`目录下就可以看到`WTSpecs`这个目录了。至此第一步创建私有`Spec Repo`完成。

PS：如果有其他合作人员共同使用这个私有`Spec Repo`的话在他有对应`Git仓库`的权限的前提下执行相同的命令添加这个`Spec Repo`即可。

创建Pod项目工程文件

这个第二步没有什么好介绍的，如果是有现有的组件项目，并且在`Git`的版本管理下，那么这一步就算完成了，可以直接进行下一步了。

如果你的组件还在你冗余庞大的项目中，需要拆分出来或者需要自己从零开始创建一个组件库，那么我建议你使用`Cocoapods`提供的一个工具将第二步与第三步结合起来做。

现在来说一下这个工具，相关的文档介绍是Using Pod Lib Create 就拿我创建的`podTestLibrary`为例子具体讲一下这里是如何操作的，先`cd`到要创建项目的目录然后执行

```
$ pod lib create podTestLibrary
```

之后他会问你四个问题，1.是否需要一个例子工程；2.选择一个测试框架；3.是否基于View测试；4.类的前缀；4个问题的具体介绍可以去看官方文档，我这里选择的是1.yes；

2.Specta/Expecta；3.yes；4.PTL。 问完这4个问题他会自动执行`pod install`命令创建项目并生成依赖。

```
$ tree PodTestLibrary -L 2
PodTestLibrary
├── Example                                  #demo APP
│   ├── PodTestLibrary
│   ├── PodTestLibrary.xcodeproj
│   ├── PodTestLibrary.xcworkspace
│   ├── Podfile                              #demo APP 的依赖描述文件
│   ├── Podfile.lock
│   ├── Pods                                  #demo APP 的依赖文件
│   └── Tests
├── LICENSE                                 #开源协议 默认MIT
├── Pod                                        #组件的目录
│   ├── Assets                              #资源文件
│   └── Classes                                #类文件
├── PodTestLibrary.podspec             #第三步要创建的podspec文件
└── README.md                                  #markdown格式的README

9 directories, 5 files
```

以上是项目生成的目录结构及相关介绍。

接下来就是向`Pod`文件夹中添加库文件和资源，并配置`podspec`文件，我把一个网络模块的共有组件放入`Pod/Classes`中，然后进入`Example`文件夹执行`pod update`命令，再打开项目工程可以看到，刚刚添加的组件已经在`Pods`子工程下`Development Pods/PodTestLibrary`中了，然后编辑demo工程，测试组件，我并没有使用提供的测试框架进行测试，这里就先不介绍了。

*注：*这里需要注意的是每当你向`Pod`中添加了新的文件或者以后更新了`podspec`的版本都需要重新执行一遍`pod update`命令。

测试无误后需要将该项目添加并推送到远端仓库，并编辑`podspec`文件。

通过`Cocoapods`创建出来的目录本身就在本地的`Git`管理下，我们需要做的就是给它添加远端仓库，同样去`GitHub`或其他的`Git`服务提供商那里创建一个私有的仓库，拿到`SSH`地址，然后`cd`到`PodTestLibrary`目录

```
$ git add .
$ git commit -s -m "Initial Commit of Library"
$ git remote add origin git@coding.net:wtlucky/podTestLibrary.git          #添
$ git push origin master     #提交到远端仓库
```

因为`podspec`文件中获取`Git`版本控制的项目还需要`tag`号，所以我们要打上一个`tag`，

```
$ git tag -m "first release" 0.1.0
$ git push --tags      #推送tag到远端仓库
```

做完这些就可以开始编辑`podspec`文件了，它是一个`Ruby`的文件，把编辑器的格式改成`Ruby`就能看到语法高亮，下面我贴上我的`podspec`文件，并在后面以注释的形式说明每个字段的含义，没有涉及到的字段可以去官方文档查阅

```
Pod::Spec.new do |s|
  s.name             = "PodTestLibrary"    #名称
  s.version          = "0.1.0"             #版本号
  s.summary          = "Just Testing."     #简短介绍，下面是详细介绍
  s.description      = <<-DESC
                       Testing Private Podspec.

                       * Markdown format.
                       * Don't worry about the indent, we strip it!
                       DESC
  s.homepage         = "https://coding.net/u/wtlucky/p/podTestLibrary"
  # s.screenshots     = "www.example.com/screenshots_1", "www.example.com/scre
  s.license          = 'MIT'               #开源协议
  s.author           = { "wtlucky" => "wtlucky@foxmail.com" }
  s.source           = { :git => "https://coding.net/wtlucky/podTestLibrary.gi
  # s.social_media_url = 'https://twitter.com/<TWITTER_USERNAME>'

  s.platform     = :ios, '7.0'            #支持的平台及版本
  s.requires_arc = true                   #是否使用ARC，如果指定具体文件，则具体的问

  s.source_files = 'Pod/Classes/**/*'     #代码源文件地址，**/*表示Classes目录及其
  s.resource_bundles = {
    'PodTestLibrary' => ['Pod/Assets/*.png']
  }                                       #资源文件地址

  s.public_header_files = 'Pod/Classes/**/*.h'    #公开头文件地址
  s.frameworks = 'UIKit'                  #所需的framework，多个用逗号隔开
  s.dependency 'AFNetworking', '~> 2.3'   #依赖关系，该项目所依赖的其他库，如果有多个
end
```

编辑完`podspec`文件后，需要验证一下这个文件是否可用，如果有任何`WARNING`或者`ERROR`都是不可以的，它就不能被添加到`Spec Repo`中，不过`xcode`的`WARNING`是可以存在的，验证需要执行一下命令

```
$ pod lib lint
```

当你看到

```
 -> PodTestLibrary (0.1.0)

PodTestLibrary passed validation.
```

时，说明验证通过了，不过这只是这个`podspec`文件是合格的，不一定说明这个`Pod`是可以用的，我们需要在本地做一下验证，这就是第四步的内容了，第四步在具体说明。

创建podspec文件

如果从第二步过来，已经有了现成的项目，那么就需要给这个项目创建一个`podspec`文件，创建它需要执行`Cocoapods`的另外一个命令，官方文档在这里

```
$ pod spec create PodTestLibrary git@coding.net:wtlucky/podTestLibrary.git
```

执行完之后，就创建了一个`podspec`文件，他其中会包含很多内容，可以按照我之前介绍的进行编辑，没用的删掉。编辑完成之后使用验证命令验证一下

```
$ pod lib lint
```

验证无误就可以进入下一步了。

本地测试podspec文件

我们可以创建一个新的项目，在这个项目的`Podfile`文件中直接指定刚才创建编辑好的`podspec`文件，看是否可用。 在`Podfile`中我们可以这样编辑，有两种方式

```
platform :ios, '7.0'

pod 'PodTestLibrary', :path => '~/code/Cocoapods/podTest/PodTestLibrary'      #
pod 'PodTestLibrary', :podspec => '~/code/Cocoapods/podTest/PodTestLibrary/PodT
```

然后执行`pod install`命令安装依赖，打开项目工程，可以看到库文件都被加载到`Pods`子项目中了，不过它们并没有在`Pods`目录下，而是跟测试项目一样存在于`Development Pods/PodTestLibrary`中，这是因为我们是在本地测试，而没有把`podspec`文件添加到`Spec Repo`中的缘故。

在项目中编写代码，测试库文件无误后就可以开始下一步了，提交`podspec`到`Spec Repo`中。

向Spec Repo提交podspec

向`Spec Repo`提交`podspec`需要完成两点一个是`podspec`必须通过验证无误，在一个就是删掉无用的注释（这个不是必须的，为了规范还是删掉吧）。 向我们的私有`Spec Repo`提交`podspec`只需要一个命令

```
$ pod repo push WTSpecs PodTestLibrary.podspec  #前面是本地Repo名字 后面是podspec名
```

完成之后这个组件库就添加到我们的私有`Spec Repo`中了，可以进入到`~/.cocoapods/repos/WTSpecs`目录下查看

```
.
├── LICENSE
├── PodTestLibrary
│   └── 0.1.0
│       └── PodTestLibrary.podspec
└── README.md
```

再去看我们的`Spec Repo`远端仓库，也有了一次提交，这个`podspec`也已经被`Push`上去了。

至此，我们的这个组件库就已经制作添加完成了，使用`pod search`命令就可以查到我们自己的库了

```
$ pod search PodTestLibrary

-> PodTestLibrary (0.1.0)
   Just Testing.
   pod 'PodTestLibrary', '~> 0.1.0'
   - Homepage: https://coding.net/u/wtlucky/p/podTestLibrary
   - Source:   https://coding.net/wtlucky/podTestLibrary.git
   - Versions: 0.1.0 [WTSpecs repo]
```

这里说的是添加到私有的`Repo`，如果要添加到`Cocoapods`的官方库了，可以使用`trunk`工具，具体可以查看官方文档

使用制作好的Pod

在完成这一系列步骤之后，我们就可以在正式项目中使用这个私有的`Pod`了只需要在项目的`Podfile`里增加以下一行代码即可

```
$ pod 'PodTestLibrary', '~> 0.1.0'
```

然后执行`pod update`，更新库依赖，然后打卡项目可以看到，我们自己的库文件已经出现在`Pods`子项目中的`Pods`子目录下了，而不再是`Development Pods`。

更新维护podspec

最后再来说一下制作好的`podspec`文件后续的更新维护工作，比如如何添加新的版本，如何删除`Pod`。

我已经制作好了`PodTestLibrary`的`0.1.0`版本，现在我对他进行升级工作，这次我添加了更多的模块到`PodTestLibrary`之中，包括工具类，底层`Model`及`UIKit`扩展等，这里又尝试了一下`subspec`功能，给`PodTestLibrary`创建了多个子分支。

具体做法是先将源文件添加到`Pod/Classes`中，然后按照不同的模块对文件目录进行整理，因为我有四个模块，所以在`Pod/Classes`下有创建了四个子目录，完成之后继续编辑之前的`PodTestLibrary.podspec`，这次增加了`subspec`特性

```
Pod::Spec.new do |s|
  s.name             = "PodTestLibrary"
  s.version          = "1.0.0"
  s.summary          = "Just Testing."
  s.description      = <<-DESC
                       Testing Private Podspec.

                       * Markdown format.
                       * Don't worry about the indent, we strip it!
                       DESC
  s.homepage         = "https://coding.net/u/wtlucky/p/podTestLibrary"
  # s.screenshots     = "www.example.com/screenshots_1", "www.example.com/scre
  s.license          = 'MIT'
  s.author           = { "wtlucky" => "wtlucky@foxmail.com" }
  s.source           = { :git => "https://coding.net/wtlucky/podTestLibrary.gi
  # s.social_media_url = 'https://twitter.com/<TWITTER_USERNAME>'

  s.platform     = :ios, '7.0'
  s.requires_arc = true

  #s.source_files = 'Pod/Classes/**/*'
  #s.resource_bundles = {
  #  'PodTestLibrary' => ['Pod/Assets/*.png']
  #}
  #s.public_header_files = 'Pod/Classes/**/*.h'

  s.subspec 'NetWorkEngine' do |networkEngine|
      networkEngine.source_files = 'Pod/Classes/NetworkEngine/**/*'
      networkEngine.public_header_files = 'Pod/Classes/NetworkEngine/**/*.h'
```

```ruby
        networkEngine.dependency 'AFNetworking', '~> 2.3'
    end

    s.subspec 'DataModel' do |dataModel|
        dataModel.source_files = 'Pod/Classes/DataModel/**/*'
        dataModel.public_header_files = 'Pod/Classes/DataModel/**/*.h'
    end

    s.subspec 'CommonTools' do |commonTools|
        commonTools.source_files = 'Pod/Classes/CommonTools/**/*'
        commonTools.public_header_files = 'Pod/Classes/CommonTools/**/*.h'
        commonTools.dependency 'OpenUDID', '~> 1.0.0'
    end

    s.subspec 'UIKitAddition' do |ui|
        ui.source_files = 'Pod/Classes/UIKitAddition/**/*'
        ui.public_header_files = 'Pod/Classes/UIKitAddition/**/*.h'
        ui.resource = "Pod/Assets/MLSUIKitResource.bundle"
        ui.dependency 'PodTestLibrary/CommonTools'
    end

    s.frameworks = 'UIKit'
    #s.dependency 'AFNetworking', '~> 2.3'
    #s.dependency 'OpenUDID', '~> 1.0.0'
end
```

因为我们创建了`subspec`所以项目整体的依赖`dependency`，源文件`source_files`，头文件`public_header_files`，资源文件`resource`等都移动到了各自的`subspec`中，每个`subspec`之间也可以有相互的依赖关系，比如`UIKitAddition`就依赖于`CommonTools`。

编辑完成之后，在测试项目里`pod update`一下，几个子项目都被加进项目工程了，写代码验证无误之后，就可以将这个工程`push`到远端仓库，并打上新的`tag`->`1.0.0`。

最后再次使用`pod lib lint`验证编辑好的`podsepc`文件，没有自身的`WARNING`或者`ERROR`之后，就可以再次提交到`Spec Repo`中了，命令跟之前是一样的

```
$ pod repo push WTSpecs PodTestLibrary.podspec
```

之后再次到`~/.cocoapods/repos/WTSpecs`目录下查看

```
.
├── LICENSE
├── PodTestLibrary
│   ├── 0.1.0
│   │   └── PodTestLibrary.podspec
│   └── 1.0.0
```

```
└── PodTestLibrary.podspec
└── README.md

3 directories, 4 files
```

已经有两个版本了，使用`pod search`查找得到的结果为

```
$ pod search PodTestLibrary

-> PodTestLibrary (1.0.0)
   Just Testing.
   pod 'PodTestLibrary', '~> 1.0.0'
   - Homepage: https://coding.net/u/wtlucky/p/podTestLibrary
   - Source:   https://coding.net/wtlucky/podTestLibrary.git
   - Versions: 1.0.0, 0.1.0 [WTSpecs repo]
   - Sub specs:
     - PodTestLibrary/NetWorkEngine (1.0.0)
     - PodTestLibrary/DataModel (1.0.0)
     - PodTestLibrary/CommonTools (1.0.0)
     - PodTestLibrary/UIKitAddition (1.0.0)
```

完成这些之后，在实际项目中我们就可以选择使用整个组件库或者是组件库的某一个部分了，对应的`Podfile`中添加的内容为

```
platform :ios, '7.0'

pod 'PodTestLibrary/NetWorkEngine', '1.0.0'  #使用某一个部分
pod 'PodTestLibrary/UIKitAddition', '1.0.0'

pod 'PodTestLibrary', '1.0.0'   #使用整个库
```

最后介绍一下如何删除一个私有`Spec Repo`，只需要执行一条命令即可

```
$ pod repo remove WTSpecs
```

这样这个`Spec Repo`就在本地删除了，我们还可以通过

```
$ pod repo add WTSpecs git@coding.net:wtlucky/WTSpecs.git
```

再把它给加回来。

如果我们要删除私有`Spec Repo`下的某一个`podspec`怎么操作呢，此时无需借助`Cocoapods`，

只需要`cd`到`~/.cocoapods/repos/WTSpecs`目录下，删掉库目录

```
wtlucky@wtluckydeMacBook-Pro:~/.cocoapods/repos/WTSpecs$ rm -Rf PodTestLibrary
```

然后在将`Git`的变动`push`到远端仓库即可

```
wtlucky@wtluckydeMacBook-Pro:~/.cocoapods/repos/WTSpecs$ git add --all .
wtlucky@wtluckydeMacBook-Pro:~/.cocoapods/repos/WTSpecs$ git ci -m "remove unus
wtlucky@wtluckydeMacBook-Pro:~/.cocoapods/repos/WTSpecs$ git push origin master
```