使用代码为 Xcode 工程添加文件

06 MAY 2015

这几天在持续更新 DKNightVersion 这个仓库, 我对这个框架投入了很多的时间和精力.

我还是希望把这个框架的方方面面都做好的, 为框架的使用者提供最好的用户体验, 将开发者的使用成本和使用难度降到最低, 提升开发者的使用体验.

但是在我维护这个框架的过程中还是感觉到如何平衡框架的可扩展性和易用性实在是太难了. 为了减少开发者工作量以达到减少错误的目的, 我在这最近几乎每天 8 小时以上的与 xcodeproj 这个文件打交道, 把原来所需的工作尽可能的降低, 从原来的:

1. ruby generator.rb
2. 拖拽 UIKit 文件到工作区的指定位置
3. 选择指定的 Target

到现在的一个命令:

1. rake

我也付出了很大的努力, 虽然这个过程看起来非常的简单, 只是**将原来的拖拽改变为使用代码来操作**而已. 但是在实际的操作中, 还是**极其麻**

烦的.

麻烦到什么程度呢? 在我每次觉得我要觉得我要成功的时候, 总是会遇到这样那样奇怪的错误或者错误给予我致命一击, 让一切努力都从零开始, 而这种情况在这几天处理这个问题的过程中遇到了 5,6 次.

我很多次都是想过要放弃实现这个功能, 不过到最后, 我还是完成了.

> *很大一部分原因是无法容忍作品中的不完美, 如果我没有想到它, 可能不会去做, 但是既然我想到了这个减少工作量的办法, 而且已经有先例(Cocoapods)完成了, 那么我也一定要实现这个功能.*

而到现在, 我觉得需要写一篇博客记录一下这个过程, 既能够为后人带来一些启发也能少走一些弯路.

Xcode 做了什么

因为 DKNightVersion 使用了 Ruby 脚本来生成 Objective-C 代码, 而在代码生成之后, 文件虽然会出现在你文件夹之中, 但是并不会直接添加到你的工程文件中, 也就是**编译器不知道你需要这些文件**.

我们一般都是通过拖拽来把这些文件添加到工程中合适的位置, 让编译器 Xcode 知道: **我需要这些文件, 帮我"编译"它们**. 这就是你在拖拽的过程中, Xcode 实际上做的事情.

在拖拽的过程中, 它实际上就是改变了一个神奇的文件 `pbxproj`, 这个文件在你的 `xcodeproj` **目录**中, 你需要右键点击 `xcodeproj` 文件

然后会出现这样的菜单, 选中显示包内容

然后你就会看到一个 `project.pbxproj` 文件, 而你对文件进行的操作, **添加和删除**实际上都是对这个文件的修改.

如果你长期使用命令行, 你就会非常自然的发现 `*.xcodeproj` `*.xcodeworkspace` 实际上都是文件夹, 但是后者并不是我们今天所要关心的问题.

我是怎么知道的呢, 我通过反复的尝试, 再删除一些文件的 `refernece`(没有将文件移到垃圾桶), 然后使用 `git diff` 查看所做的修改, 就会发现其他的文件并没有被修改, **唯一被修改的就是这个 `project.pbxproj` 文件**.

- 在我们进行文件的**增加和删除(不包括修改)**的过程中, 实际上就是对 `project.pbxproj` 的修改.

向 Xcode 工程中添加或删除文件

如何在非 GUI 中完成对 Xcode 工程文件的操作呢? 我总共进行了三种不同的尝试.

手搓

作为一个~~闲的蛋疼~~专业的工程师, 我的第一想法是, 这需求肯定很多, 干脆造一个轮子好了, 于是我打开了 `project.pbxproj` 文件. 立刻被里面的数据所淹没了.

```
/* Begin PBXBuildFile section */
        0DFDC45E10D450EE9CA10394 /* UITableViewCell+Back
        186AC5221AF855410095ED95 /* DKNightVersionManage
        18D2288D1AF8BA4F00872BF1 /* UINavigationBar+Anim
```

七百多行的文件, 我在这里只选取了三行. 因为这个文件中的**数据**实在是太多了. 你会看到每一行的前面大概都是一个 **24 位的 16 进制数字** 然后就是一个文件名 `UITableViewCell+BackgroundColor.h` 在最后有一个 `isa` 和一个 `fileRef`.

在大概浏览了一下这个文件的全部内容之后, 我对于手搓这个 script 有了一个初步的想法, 第一步就是生成这个 **24 位的 16 进制数字**. 当然我希望知道苹果是如何生成这一 16 进制数字的.

(注: 我实在无法对这个文件中的任何一部分进行详尽的解释, 不过之后我会介绍几个重要的相关概念)

然而我没有找到相关的资料, 而我自己也没有这个能力来推导出这个神奇的数字是如何生成的.

使用 `xcodeproj`

Xcodeproj 是一个使用 Ruby 来创建和修改 Xcode 工程文件的工具. 我找到它的原因是 Cocoapods 也通过 Ruby 代码向 Xcode 工程中添加文件, 所以我在 Cocoapods 中找到了这一组件.

在最开始尝试使用这个工具的时候, 发现它的文档是**极其**糟糕. 根本无法直接在文档找到**向 Xcode 工程中添加文件的方法**, 这一简单的需求并没有明确的写在文档中, 这令我十分的不理解.

这个组件的主要功能不就是向工程中添加文件么? 为什么没有写在文档中, 而我所做的就是不断的阅读这个组件的源代码, 尝试理解 `project.pbxproj` 中的东西到底都是什么. 而在这期间, 有几个非常重要的问题需要我们理解.

- Target
- Group
- FileRef

Target

`Target` 到底是什么, 我在以前的 iOS 开发的工作中并没有仔细地考虑这一个问题, 直到我遇到这一**需求**时, 我在尝试理解这背后的意义.

> *A target specifies a product to build and contains the instructions for building the product from a set of files in a project or workspace. A target defines a single product; it organizes the inputs into the build system—the source files and instructions for processing those source files—required to build that product. Projects can contain one or more targets, each of which produces one product.*

Target 指定了一个用于产品(product), 并且包含了从工程中的一些文件中构建产品的命令.

这些命令使用构建设置(build settings)和构建阶段(build phases)的方式来组织, 我们可以在 Xcode 编辑器中改变这些设置.

Group

Group 这个概念和我们平时经常说的 folder 文件夹有很大的差别, 文件夹在我们的日常使用时是一直所接触到的, 而对于 Group, 如果你不使用 Xcode 来编程(不是很清楚别的 IDE 是否有这个功能)的话, 这个概念距离你太远了.

Group 其实是 Xcode 中用来组织文件的一种方式, **它对文件系统没有任何影响**, 无论你创建或者删除一个 Group, 都不会导致 folder 的增加或者移除. 当然如果在你删除时选择 Move to Trash 就是另外一说了.

在 Group 中的文件的关系, 不会与 folder 中的有什么冲突, 它只是 Xcode 为你提供的一种分离关注的方式而已. 但是, 我一般会在开发过程中将不同的模块分到不同的 Group 和 folder 中便于整理.

Group 之间的关系, 也是在 `project.pbxproj` 中定义的, 这个文件中包含了 Xcode 工程中所有 File 和 Group 的关系, 如果你大致浏览过这个文件的话, 你就会对我所说的有所了解.

Group 在我们的工程中就是黄色的文件夹, 而 Folder 是蓝色的文件夹(一般在 Xcode 工程中, 我们不会使用 Folder).

FileRef

FileRef 其实就是 File Reference 的缩写, 当你从 Xcode 中删除一个文件的时候, 它会弹出这样的提示框.

而其中的 `Remove Reference` 选项并不会将这个文件移除到垃圾桶, 而只是会将这个文件的引用从 Xcode 的工程文件中删除.

如果你曾经看过 `Build Phases` 中的内容, 你会发现

- 如果删除的是 `.h` 文件, 它会从 `Build Phases` 中的 **Headers** 部分删除
- 如果删除的是 `.m` 文件, 它会从 `Build Phases` 中的 **Compile Source** 部分删除

但是文件还是会在原来的地方, 因为 Xcode 中所加入到工程的只是文件的一个引用 --- File Ref.

添加文件

我们已经基本了解了阅读这一篇博客所需要的全部知识, 接下来, 我们就需要来分析一下向 Xcode 工程中添加文件所需要的几个步骤.

当我们生成一堆 Objective-C 代码时, 我们的第一步是要将这些文件拖入 Xcode 工程中, 这时 Xcode 会弹出视图询问你是创建 Groups 还是 Folder references, 并询问你要加入到哪个 Target 中.

- 当选择创建 Groups 时, Xcode 就会把 .h 文件加入 Build Phases 中的 Header, 把 .m 文件加入 Compile Sources 中, 并创建一个黄色的文件夹.
- 当选择创建 Folder references 时, Xcode 会把所有的文件加入 Build Phases 中的 Copy Bundles Resources, 不进行任何的编译, 然后创建一个蓝色的文件夹.

我们现在就来使用代码来模拟将文件加入 Xcode 中, 选择 Create Groups 并且添加到指定 Target 的全过程.

在这里我们需要使用 Ruby 的开源框架 xcodeproj 这个框架是著名的开源框架 Cocoapods 的一个组件.

> *Create and modify Xcode projects from Ruby.*

查找 `*.xcodeproj` 文件

lib/xcodeproj/project.rb

在使用 xcodeproj 时, 只需要**查找 `*.xcodeproj` 文件**而不是 `*.pbxproj`. 我们通过调用 `Project` 的类方法来打开文件.

```ruby
project = Xcodeproj::Project.open(path)
```

获取 Target

lib/xcodeproj/project.rb

接下来要获取 Target, 模拟选择 `Add to targets` 这一操作, 在每一个

Project 中都有多个 Target.

一般在我们的项目中, 一般都有一个以 `项目名` 和一个以 `项目名+Test` 命名的 Target.

Tips:

- 在 Pod 项目中, 每一个 Pod 都会对应一个 Target, 然后会有一个 `Pods-项目名` Target.

而在这里我想要获得的就是以项目名命名的 Target, 一般都是 `targets` 数组中的第一个.

```ruby
target = project.targets.first
```

创建 Group

lib/xcodeproj/project/object/group.rb

在获取 Target 之后, 需要创建或者获取一个文件即将被添加进去的 `group`, 我一般使用 `find_subpath` 这个方法

```ruby
def find_subpath(path, should_create = false)
```

它能比较快捷的根据路径名寻找 `group`, 如果当前的 `group` 不存在,

它还会递归地创建(可选).

```ruby
group = project.main_group.find_subpath(File.join('DKNig
```

因为这个方法是一个 `group` 的实例方法, 所以先通过 `main_group` 获取主 `group`, 然后再调用这个方法, 最后会返回指定的 `group`. 在工程中创建这样一种的结构.

在成功获取之后还**需要把 `group` 的 `source_tree` 设置成 `'SOURCE_ROOT'`**, 这样在加入到 Build Phases 的时候, 它会从工程文件的根目录下开始寻找你所添加的文件, 不会出现一些非常奇怪的问题.

```ruby
group.set_source_tree('SOURCE_ROOT')
```

向 `group` 中添加文件

lib/xcodeproj/project/object/group.rb

我们在获取 `group` 之后就可以向其中添加文件了. 在这时使用 `new_reference` 方法, 为文件创建一个 `FileRef` 添加到 `group` 中.

```ruby
file_ref = group.new_reference(file_path)
```

这样这个文件就添加到了 `group` 中, 会出现在工程中的导航栏中.

但是这个文件并没有被添加到 `Build Phases` 中, 无论你是编译还是作为资源来使用, Xcode 都会提示你无法找到这个文件. 我们还需要把这个文件加入到 `Build Phases` 中.

将文件加入 `Build Phases`

lib/xcodeproj/project/object/native_target.rb

在前面获取到的 `Target` 在这一步就开始起了作用, 我们需要获取 `Target` 的 `Build Phase` 并将在上面添加的文件添加到 `Build Phase` 中.

```ruby
target.add_file_references([file_ref])
```

`add_file_references` 就负责把一组 `FileRef` 添加到对应的 `Build Phases` 中, `source_build_phase` `headers_build_phase` `resource_build_phase` `framework_build_phase` 在 GUI 中你可以找到这四者对应的 section.

而 `add_file_references` 方法自动为你把 `FileRef` 添加到合适的 phase 中.

但是从 `Build Phase` 中移除文件就需要手动获取这些 `*_build_phase` 然后从中调用 `remove_reference` 来删除文件或者资源.

```ruby
target.source_build_phase.remove_file_reference(file_ref
target.headers_build_phase.remove_file_refernece(file_re
```

保存 Project

在最后, 我们只需要调用 `save` 方法来保存整个工程就好了.

```ruby
project.save
```

总结

```ruby
project = Xcodeproj::Project.open(path)
target = project.targets.first
group = project.main_group.find_subpath(File.join('DKNig
group.set_source_tree('SOURCE_ROOT')
file_ref = group.new_reference(file_path)
target.add_file_references([file_ref])
project.save
```

其实到现在为止, 我感觉到使用代码向 Xcodeproj 中添加文件是很简单的事情, 那是因为, 首先有 Xcodeproj 这样文档糟糕但是功能还是比较齐全的第三方框架, 而且这是我在几天不停地阅读源代码, 不停被坑, 一点一点尝试才摸索出来的结果, 都是泪啊...不想多说了...不过最后把这个问题解决之后, ~~自我感觉还是蛮好的~~还是挺高兴的. 嗯, 就这样.

Draveness

iOS Developer / Rails / Elixir

Maine, USA *draveness.me*

Share this post

blog comments powered by Disqus

READ THIS NEXT

DKNightVersion 的实现 --- 如何为 iOS 应用添加夜间模式

最新: [成熟的夜间模式解决方案](http://draveness.me/night) **注意: 这篇文章已经过时了, 最新版本的 2.3.0 实现已经更改了.** 在很多重阅读或者需要在夜间观看的软件其实都会把夜间模式当做一个 App 所需要具备的特性. 而如何在不改变原有的架构, 甚至不改变原有的代码的基础上, 就能为应用优雅地添加夜间模式就成为一个在很多应用开发的过程中不得不面对的一个问题. 就是以上事情的驱动, 使我思考如何才能使用一种优雅并且简洁

YOU MIGHT ENJOY

程序员修炼之道 Tips

这一篇文章其实就是记录程序员修炼之道中的所有 Tips, 我讲会在之后的每周实践两个 Tip, 并对这两个 Tips 进行补充和说明自己的体会, 最终成为书中所说的卓有成效的程序员. Tip 1: Care About Your Craft 关心你的技艺...

的方法解决这一问题....