黑客派

征服恐惧！用 Vim 写 iOS App

作者：noark9

原文链接：https://hacpai.com/article/1487647846821

来源网站：黑客派

我们都知道 VIM 和 Emacs 都是文本编辑器中的上古神器，你也许用 ctags，cscopes 配合 VIM 完成过大型 C 或者 C++ 的开发，你也许配合过其他插件，完成过 JavaScript，python 代码的开发但是很少有人试过 iOS app 的开发吧，毕竟 iOS 的框架包含了很多东西，以及 Objective-C 天生很的 API 名字，让我们没办法把此神器用起来，今天我就来给大家讲下我是怎么使用 VIM 开发 iOS App 的，当然 Emacs 也可以

<h2 id="begin">begin</h2>
使用 VIM 开发 iOS App 并不是特殊的爱好，而是被 Xcode 8 活生生的逼的，刚开始更新了 Xcode 8 以后，Xcode 8 把第三方插件给屏蔽了，导致没有 XVim 给我用了，没有 XVim 以后，发现异常顺手，于是尝试用了一段时间的 AppCode，不得不说 AppCode 是一个非常好的 IDE，但是他有个大的缺点，那就是 Java，JetBrain 家的东西都很不错，唯一缺点就是基于 Java，整个平台都略慢，后我在不断的 Google 过程中，发现了有人竟不知何谓恐惧，竟然使用 VIM 开发 iOS App，最后我学会了这个新姿势 XDDDD
不过目前，只支持 Objective-C 代码的开发，swift 的话，没有解决工程文件自动补全的问题，为目前大家使用的流行的 swift 自动补全工具 SourceKitten 并没有支持 workspace 所以暂时还没用来
以及，目前不支持调试，因为发现 VIM 对调试的支持确实好糟糕...
<h2 id="准备活动">准备活动</h2>
工欲善其事，必先利其器，主角是 VIM 或者 Emacs，少了其他配角和龙套们，也没办法正负恐，我们来看看用到了些什么东西，让我们的 VIM 成为利器的，这里只是点下他们的名，文章后面会链接奉上

  首先我们来看主角队的同学们，他们是征服恐惧的主力

   macOS，没有神话，开发 iOS 还是只能在 Mac 上
   支持 python 的 VIM，可以用 Vim8 或是 neoVim 食用更佳，我就用的 neoVim
   YCMD，其实他是 YouCompleteMe + YouCompelteMeDeamon 的合体，自动补全、定义转等功能，就依赖他了

  接下来我们来看看其他龙套们

   Vbundle，装插件用的，没他，龙套和主角都不用上场了
   unite + unite-outline + unite-outline-objc，提供了方法导航
   auto-pairs，自动补全右括号
   ctrlp，文件搜索跳转
   Ag，字符串搜索工具
   syntastic，语法检查工具
   vim-clang-format，代码格式工具

恩，需要的东西大概就是这些了，VIM 的配置文件，我是基于好久以前 square 开源的 maximu-awesome 的，所以，配合这个食用风味更佳，我的 dot file 也放到了 GitHub 上，欢迎大家 star
<h2 id="进入正题">进入正题</h2>
龙套们，基本都可以通过配置 vbundle 来完成安装，之后只用配置对应的快捷键就好了，这里正题，要搞定难搞的主 YCMD

<h2 id="难搞的主-YCMD">难搞的主 YCMD</h2>
<h3 id="安装-YCMD">安装 YCMD</h3>
YCMD 的安装很简单，主要是需要一定的配置
首先在 VIM 的配置文件中加入下面的内容，更新配置文件并执行 <code>BundleInstall</code> 命令，让 Vbundle 把 YouCompleteMe 插件装上
<pre><code class="language-vim highlight-chroma">Plugin 'Valloric/YouCompleteMe'
</code></pre>
然后到这个路径 <code>~/.vim/bundle/YouCompleteMe</code> 这里是 YouCompleteMe 安装的位置，在这里需要编译 YCM，一条命令就可以搞定

<pre><code class="language-bash highlight-chroma">./install.py --clang-completer --system-libclang
</code></pre>
<code>--clang-completer</code> 告诉脚本我们需要 clang 的支持，<code>--system-libclang</code> 告诉编译脚本使用系统的 clang，因为之前 clang 升级 4.0 的时候，并没有已经编译好包给我下载，所以这里不用系统 clang 的话，编译脚本会下载一个 clang 3.0，这样就无法支持 iOS 1.0 以后的 sdk 了，因为 iOS 10.0 以后的 sdk 为了支持 swift 引入了一些 clang 3.0 不支持的新语法所以这里要加上 <code>--system-libclang</code>
然后等他编译完成，这样 YCMD 就配置好了，似乎这里看并不是很难搞，其实难搞的是如何在 iS 项目中配置好自动提示
<h3 id="为-Xcode-项目配置-YCMD">为 Xcode 项目配置 YCMD</h3>
这里进入到了真正征服恐惧的地方了
<h4 id="YouCompleteMe-的原理">YouCompleteMe 的原理</h4>
曾经有人说过，编辑器再怎么神器是没办法超过 IDE 的，因为 IDE 是通过编译、解析整个项目所有文件，来达到语法错误提示，自动补全，定义跳转等高级功能的，而 YCMD 就是来弥补这一个距的，YCMD 通过传入完整的编译参数，编译需要提示的文件，来实现自动补全，这样没办法超过 ID 的部分就被抹平了
<h4 id="配置一个项目">配置一个项目</h4>
这里我们配置一个复杂的项目来练练手，首先 YCMD 是不可能通过 Xcode 的项目文件或是 workspace 文件获取到编译参数的，所以这一步需要手来，当然，将来可以做成自动的，因为目前手动的其实很方便，所以现在还没有做成自动的
首先，YCMD 是通过每个项目路径下的 <code>.ycm_extra_conf.py</code> 脚本文件来获取译参数的，这个脚本文件中有一个叫做 <code>FlagsForFile</code> 的函数，我们通过这个函数返某一个特定文件需要的编译参数，一般情况下大部分文件的编译参数是一样的，我们来看一个配置的子
<pre><code class="language-python highlight-chroma">import os
import ycm_core

flags = [
'-resource-dir',</code></pre>

'/Applications/Xcode.app/Contents/Developer/Toolchains/Xcod
Default.xctoolchain/usr/bin/../lib/clang/8.0.0' ,</
pan>

'-x objective-c' ,</sp
n>

'-arch armv7' ,</spa
>

'-fmessage-length=0' ,
/span>

'-fmodules',

'-gmodules',

'-fdiagnostics-show-note-include-stack'<span class="hi
hlight-p">,

'-fmacro-backtrace-limit=0' ,
/span>

'-D__arm__=1' ,</sp
n>

'-D__IPHONE_OS_VERSION_MIN_REQUIRED=80000'<sp
n class="highlight-p">,

'-std=gnu99' ,</spa
>

'-fobjc-arc' ,

'-Wnon-modular-include-in-framework-module'<span c
ass="highlight-p">,

'-Werror=non-modular-include-in-framework-module'<
pan class="highlight-p">,

'-Wno-trigraphs' ,</
pan>

'-fpascal-strings' ,</
pan>

'-Os' ,

'-fno-common' ,</s
an>

'-Wno-missing-field-initializers'<span class="highlight-
">,

'-Wno-missing-prototypes' ,
/span>

'-Werror=return-type' ,
/span>

'-Wunreachable-code' ,
/span>

'-Wno-implicit-atomic-properties'<span class="highligh
-p">,

'-Werror=deprecated-objc-isa-usage',
'-Werror=objc-root-class'
/span>
'-Wno-arc-repeated-use-of-weak',
'-Wduplicate-method-match'
/span>
'-Wno-missing-braces'
/span>
'-Wparentheses'
pan>
'-Wswitch'
'-Wunused-function'
/span>
'-Wno-unused-label'
/span>
'-Wno-unused-parameter'
/span>
'-Wunused-variable'
/span>
'-Wunused-value'
span>
'-Wempty-body'
pan>
'-Wconditional-uninitialized'
/span>
'-Wno-unknown-pragmas'
/span>
'-Wno-shadow'
an>
'-Wno-four-char-constants'
/span>
'-Wno-conversion'
/span>
'-Wconstant-conversion'
/span>
'-Wint-conversion'
/span>
'-Wbool-conversion'
/span>
'-Wenum-conversion'
/span>
'-Wshorten-64-to-32'

/span>
'-Wpointer-sign' ,</
pan>
'-Wno-newline-eof' ,
/span>
'-Wno-selector' ,</s
an>
'-Wno-strict-selector-match' ,
/span>
'-Wundeclared-selector' ,
/span>
'-Wno-deprecated-implementations'<span class="highl
ght-p">,
'-DOBJC_OLD_DISPATCH_PROTOTYPES=0'<span class=
highlight-p">,
'-isysroot' ,
'/Applications/Xcode.app/Contents/Developer/Platforms/iPhone
imulator.platform/Developer/SDKs/iPhoneSimulator.sdk' ,
/span>
'-fstrict-aliasing' ,</
pan>
'-Wprotocol' ,</spa
>
'-Wdeprecated-declarations' ,
/span>
'-miphoneos-version-min=8.0'<span class="highlight-p
>,
'-g' ,
'-Wno-sign-conversion' ,
/span>
'-Wno-infinite-recursion' ,
/span>
'-fembed-bitcode-marker' ,
/span>
'-I/Applications/Xcode.app/Contents/Developer/Platforms/iPhon
Simulator.platform' ,
'-I/Applications/Xcode.app/Contents/Developer/Platforms/iPhon
Simulator.platform/Developer/SDKs/iPhoneSimulator.sdk/usr/include'<span class="h
ghlight-p">,
'-I/Applications/Xcode.app/Contents/Developer/Platforms/iPhon
Simulator.platform/Developer/SDKs/iPhoneSimulator.sdk/System/Library/Frameworks'</spa
> ,
'-I/Applications/Xcode.app/Contents/Developer/Toolchains/Xco
eDefault.xctoolchain/usr/bin/../lib/clang/8.0.0/include' ,

/span>

'-F/Users/apple/Documents/Developer/CloudLifeWorkspace/iOS Develop/Project_iOS/project',

'-MMD',

'-MT',

'-MF',

]

SOURCE_EXTENSIONS = ['.cpp', '.cxx', '.cc', '.c', '.m', '.mm']

HEADER_EXTENSIONS = ['.hpp', '.hxx', '.hh', '.h']

def FlagsForFile(filename, **wargs):

staticFlags = flags

return {

'flags': staticFlags,

'do_cache': True

}

</code></pre>

<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>

<!-- 黑客派PC帖子内嵌-展示 -->

<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></ins>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<p>这里上面的代码可以当作 <code>.ycmd_extra_conf</code> 文件的一个最小模板，也就是如

我们的项目里面没有子目录，没有第三方库，那么使用这个已经可以为 iOS 项目提供自动提示了

颤抖吧凡人，如此这般的编译条件，需要你能够把 Xcode 项目配置中的编译参数完完全全翻译来才行，但是我也是凡人，所以这个不是我写的，而是有方法生成的，方法如下
把需要增加自动提示的项目用 Xcode 打开，然后编译，然后如下图所示，找到编译信息然后找项目中的一个文件，注意不要是 Pod 中的文件，最右边，有个三条横线的按钮，点开他，没错，你到了完整的编译参数，下图中 <code>EXPORT PATH =....</code> 下面的所有内容就是完整的编参数

 Report View

 戳这个按钮

 我们要的编译信息

右键，copy，找到你熟悉的记事本，参数都是空格分隔的，所以，这里我们把它转换成上面代中的形式，并把不需要的去掉，比如这里我注释了 <code>-fmodules</code> 和 <code>-gmodues</code> 因为这样编译没办法使用 module
把这些加入到配置文件中后，再打开 VIM 你就会发现自动提示变得非常好用了，不过这里还会一个问题，如果编译的时候，有无法找到 <code>UIKit</code>，这是因为 YCMD 默认引入了 maOS SDK 的路径，导致了编译时 clang 认为我们编译目标是 macOS，所以如下图需要修改 <code>~/vim/bundle/YouCompleteMe/third_party/ycmd/ycmd/completers/cpp/flags.py</code> 文件去掉默认引入 macOS sdk 的编译参数，这样就好了

<h4 id="效果图们">效果图们</h4>
至此，最主要的问题已经被我们解决了，接下来看下效果如何

 首先是自动提示

 查找文件

 方法名称大纲

 搜索字符串

 语法检查


<h2 id="好处">好处</h2>
这么做自然不是为了花样炫技，更多的是为了探索 VIM 的使用，以及更多了解 Xcode 项目

 完整的 VIM 环境，写代码再也不用碰鼠标了
 VIM 插件 + 不满的地方自己动手丰衣足食
 快速，跟 VIM 比速度，笑话
 方便快接的字符串搜索，替换操作
 学了点 Python
 附赠一个 C/C++ 的开发环境

<h2 id="已知问题">已知问题</h2>
虽然目前已经可以达到写代码的程度了，但是还是有很多问题，如下：

 没办法 debug
 不支持 Xcode 中的 group 展示
 头文件的提示有问题
 不能使用 <code>@import</code> 的导入，会报语法错误
 xib，storyboard 自然是不行的，我的做法是 Xcode 里面拖 UI + 关联 Outlet + Debug，VIM 中做大量的代码编辑操作
 <code>[]</code> 方括号的匹配没有 Xcode 那么智能
 delegate 或是全局的自动提示，需要使用 ctrl + 空格打开，并且有时候会有点慢
 delegate 的自动提示有时候需要输入前面半部分的方法名才会有，比如上面截图的 tableView 需要先输入 <code>- (void)tableView:(UITableView *)tableView </code> 再使用 ctrl + 空格才有非常好的自动提示，当然输入前面这部分也会有一定的提示，整体上看能接受

上面的问题都是一直以来我没有解决的问题，大家要是发现有破解的方法，欢迎联系
<h2 id="end">end</h2>
折腾这么多，相信大家已经可以用 VIM 敲 iOS 的代码了，虽然虽然当初开始折腾的时候，踩了多坑，比如 clang 3 升级 clang 4 后，原来的配置都不能用了，但是收获挺多的也并不是这么一篇文能够说完的，除了 VIM 大家也可以试试 Emacs 下的配置，我用的 Spacemacs，添加了 ycmd 的 layr，配置后也有了相同的效果
也欢迎大家丢砖
<h2 id="参考资料">参考资料</h2>

 YouCompleteMe: 自动补全插件

/li>
 Vbundle: 插件管理插件
 Unite: 一个通用的显示插件，可以用显示各种东西，比如文件列表，buffer 列表，outline
 Unite-outline: Unite 插件的 outline 插件
 Unite-outline-objc: Unite outline 插件的 Objective-C 插件
 auto-pairs: 自动补全括号的插件</l>
 ctrlp: 文件查找插件
 Ag: 字符串查找插件
 syntastic: 语法检查插件
 vim-clang-format: clang format 格式化插件
 VimAwesome: 方便的 VIM 插件导航网站
 maximum-awesome: 一比较有名的 VIM 配置，包括了上面的 Vbundle，Ag，ctrlp，syntastic 等插件，以及一些很方便的置，我的配置文件是基于这个配置的，使用前先安装这个
 我的配置文件: 我的配置文件，除了上文的内容，还加了些 PHP，python，js 等开发配置，以及一些自己觉得用起来方便的配置
