

# 利用 Xcode Extension 开发 Xcode 的插件

关键字：Xcode Source Editor Extensions

## 前言

---

在 Xcode 8 之前，想要为 Xcode 扩展功能的话可以使用一些第三方的插件来实现，而之前的插件都是在程序员在没有官方文档的情况下摸索出来的，而在 iOS 10 之后，苹果推出了一系列的扩展(extension)，其中包括 Xcode 的扩展(Xcode Source Editor Extensions)，算是官方提供的一种为 Xcode 提供功能的方式。

当然，苹果提供这个扩展功能有限，只能为代码编辑区域作一些文字性的替换操作等，而且到现在为止还是极其不稳定的，在 beta 版的 Xcode 8 中，非常大的概率出现功能无效。另外一个非常不好的消息就是在 Xcode 8 上面所有之前的插件都不能使用了。但是，不管怎么说，官方提供的东西始终都是有优势的。

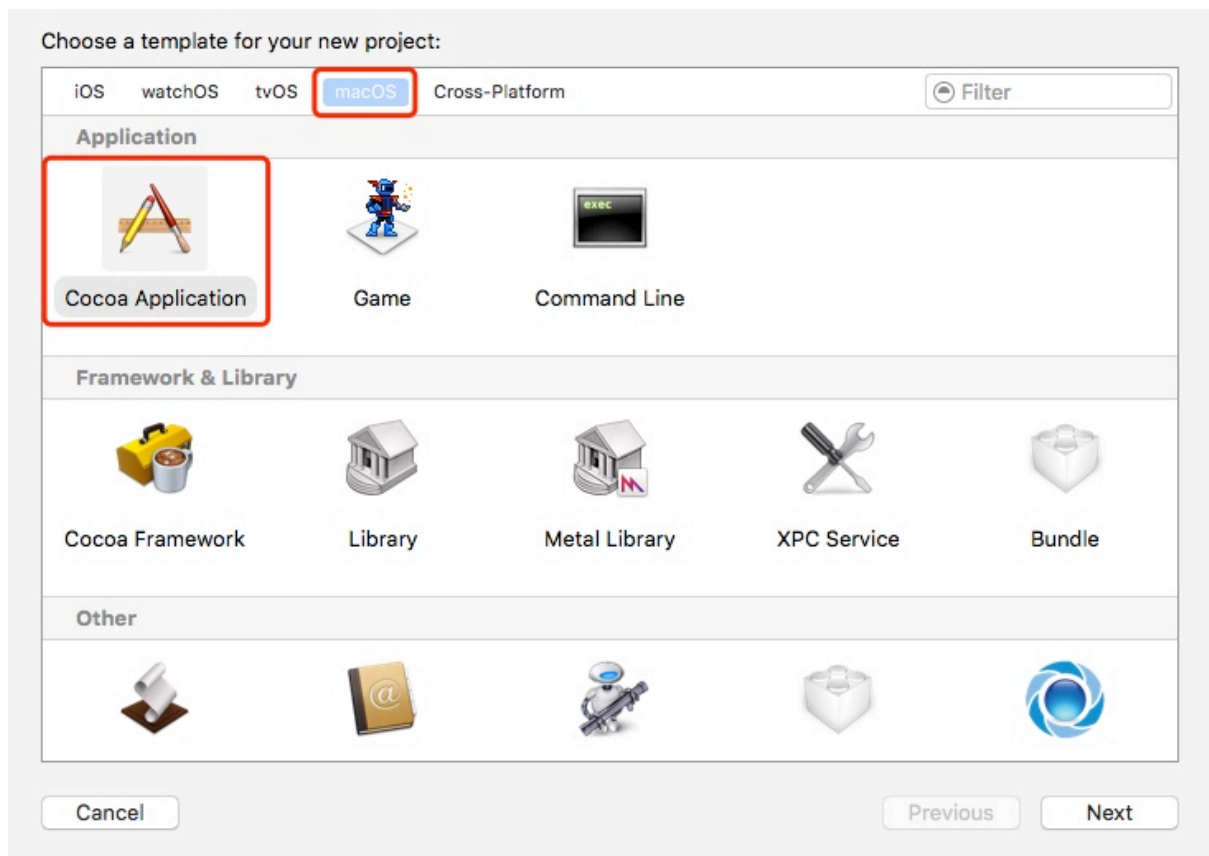
而接下来就介绍一下，在 Xcode 8 上如何实现一个 Editor Extensions。模拟实现一个名字叫 [Xcode Kit](#) 的插件，该插件的主要功能是可以对代码 `快速删除当前行` 和 `快速复制当前行`。

注：以下操作都是在 Xcode 8 beta 3 上演示的，到现在为止，Xcode 8 已经出到第6个 beta 版本了，而在 beta 3 是我测试稍微稳定的一个版本😓

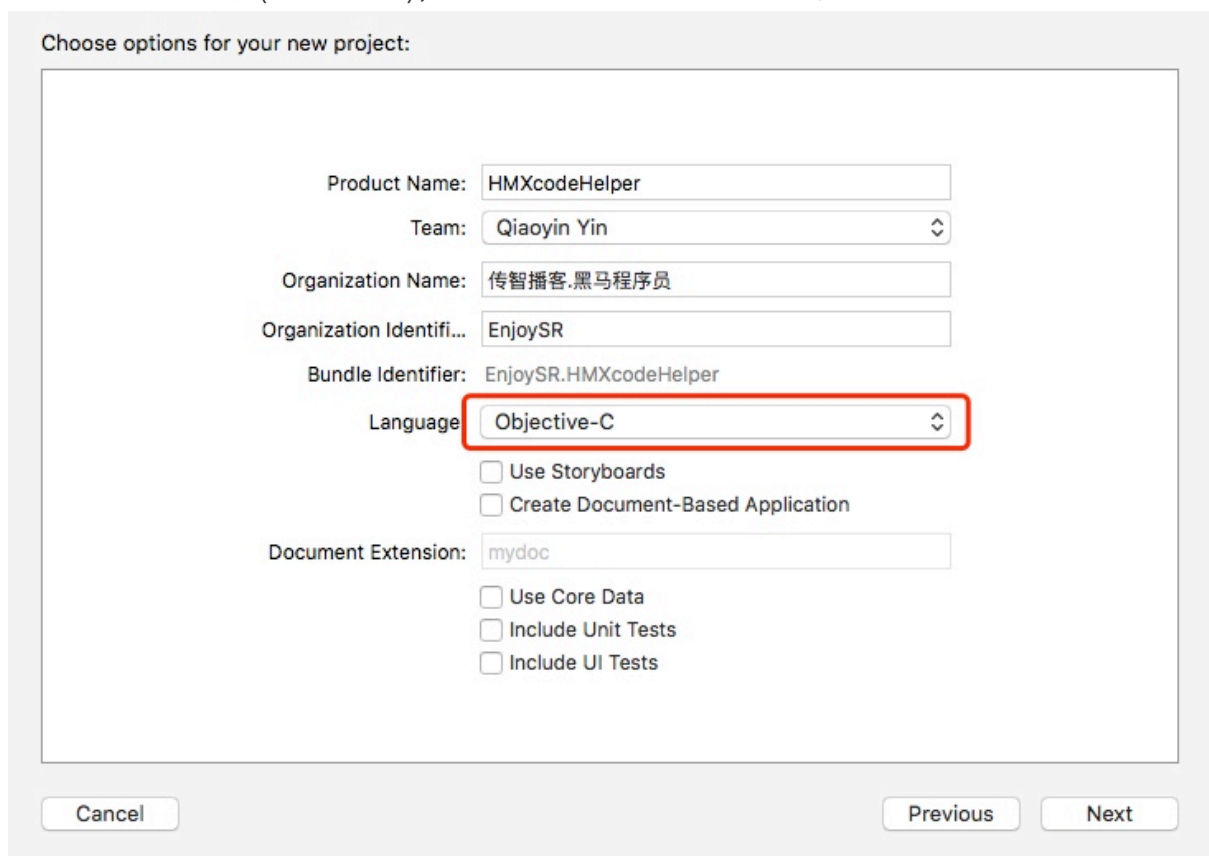
## 创建扩展

---

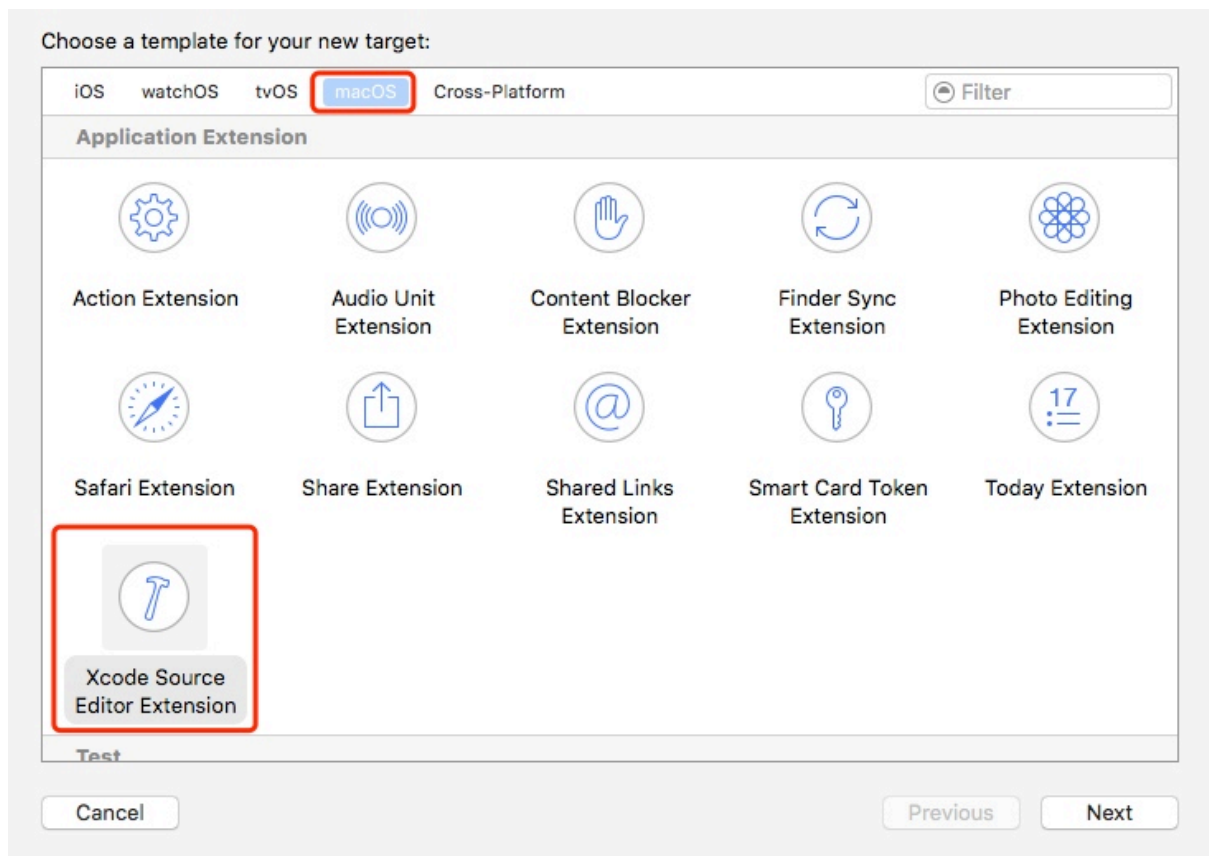
- 创建 Mac OS 的项目，并输入项目名字：



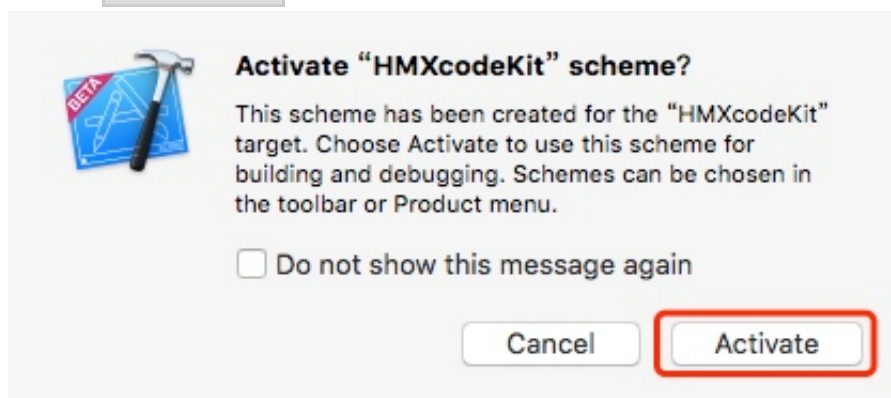
- 输入项目名字 `HMXcodeHelper`，并且选语言为OC（在 Xcode 8 的系列 beta 版本中(beta 1-6)，Swift 语法不停的改变）



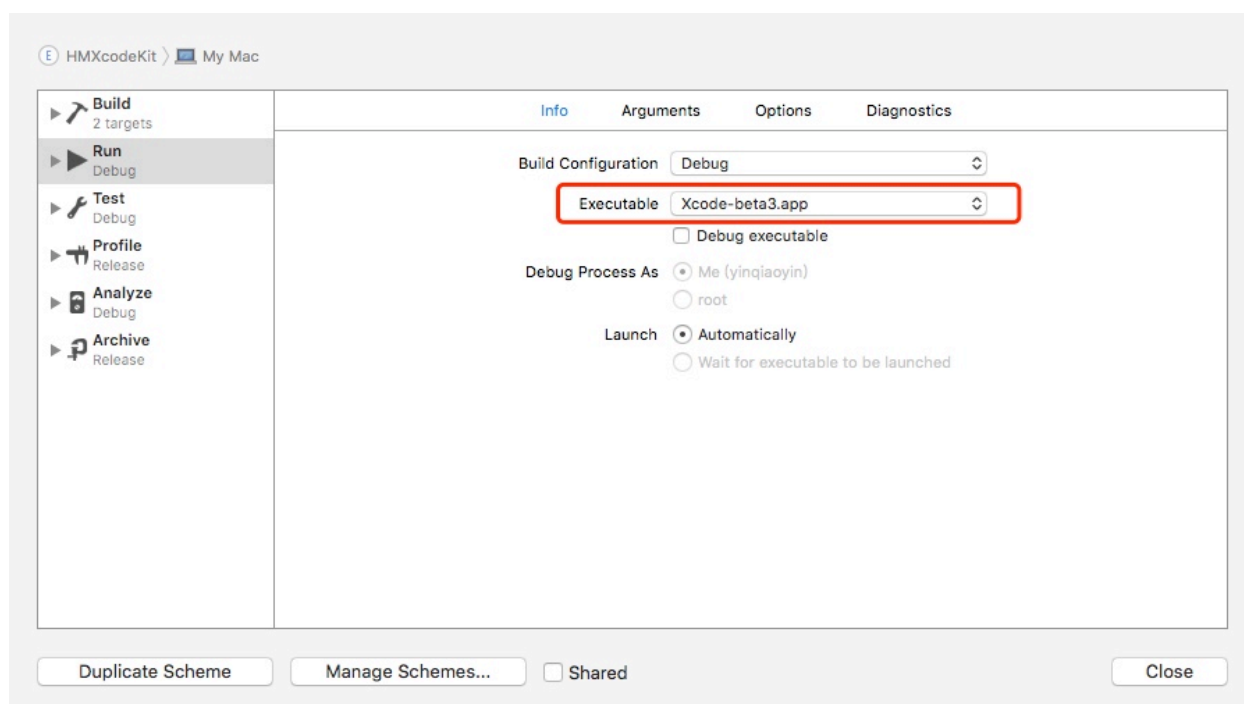
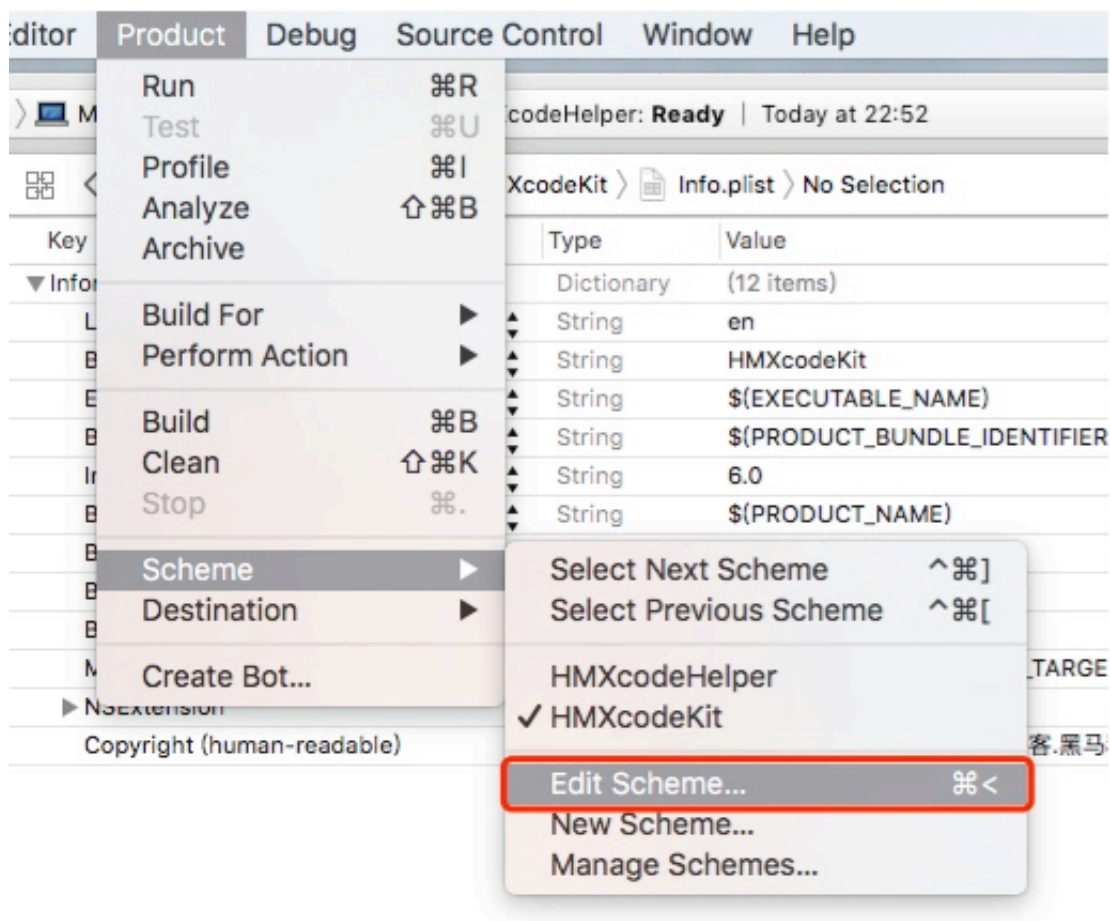
- 创建 target，选择 Xcode Source Extension，输入名字 ``



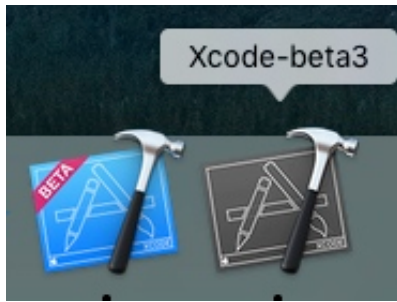
- 创建到该 target 之后，会弹出提示框，提示是否启用刚才创建的方案，选择 **Activate**



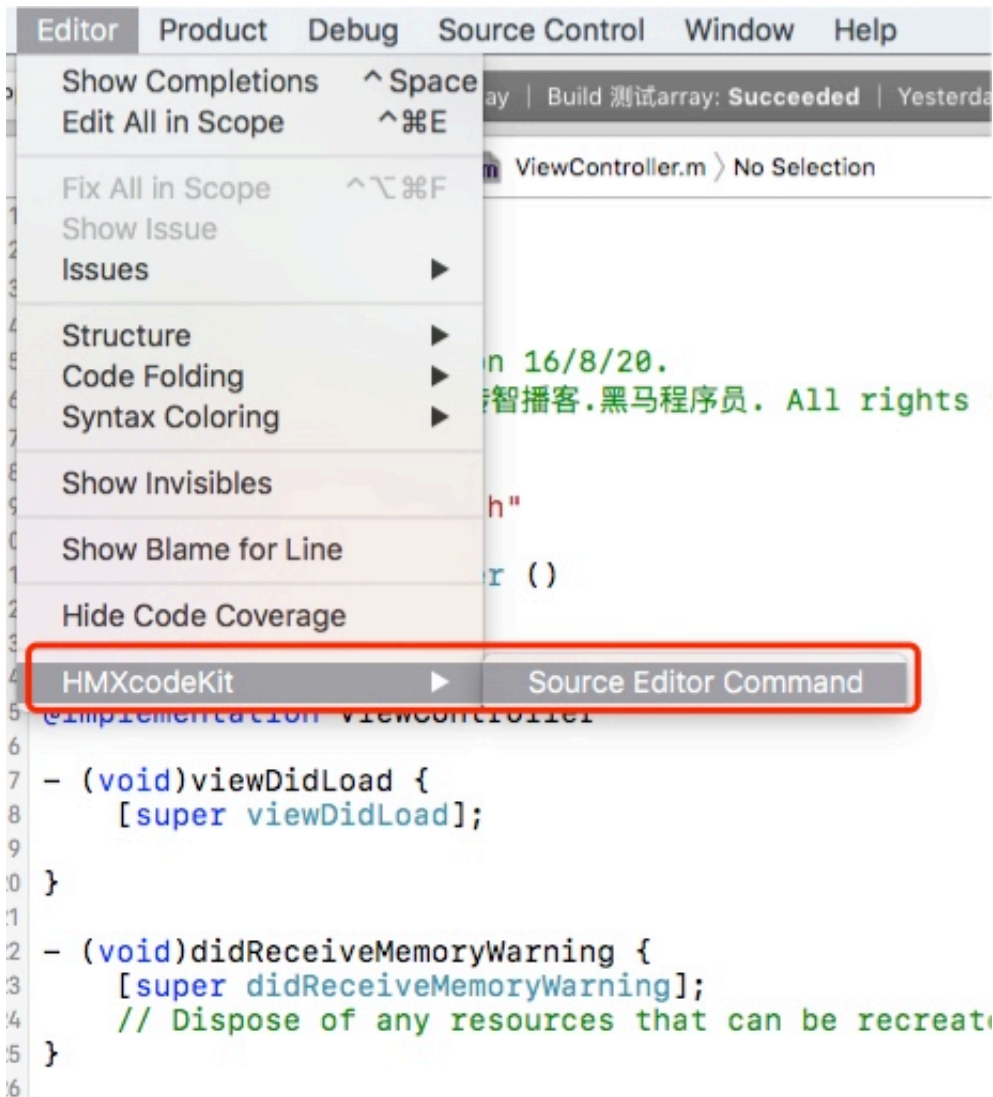
- 接下来可以 **command + r** 运行该扩展，运行的时候，会提示选择使用哪个程序去执行它，选择 Xcode 8 beta 3 运行，也可以设置该 target 的 **scheme**，进去之后设置 **Executable**，选择对应的 Xcode 再运行(如下图)。



- 运行起来之后，会出现一个黑色的 Xcode 图标在 Dock 中，该 Xcode 为调试插件的 Xcode



- 随机打开一个项目，然后打开一个代码文件，Xcode 菜单栏中的 **Editor** 菜单中会出现 **HMXcodeKit**，其子菜单是 **Source Editor Command**

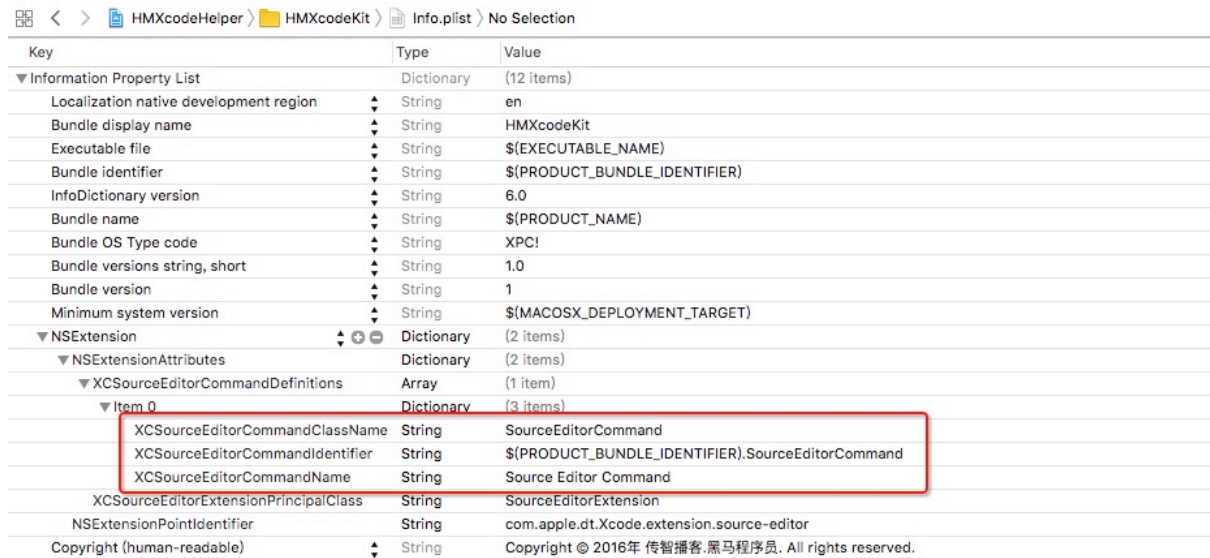


有很大的机率该菜单 **HMXcodeKit** 是 Disable 的，估计是 Xcode 的 bug

- 到此，一个 Xcode 的扩展就创建完成了，可能到这个时候就有疑问了，那个菜单里面菜单名字是怎么来的呢？在哪儿配置的呢，那么接下



我们来看一个刚才我们创建的那个 `HMXcodeKit` 的 target 里面的 `info.plist` 文件，关键参数如下：



Key	Type	Value
Information Property List	Dictionary	(12 items)
Localization native development region	String	en
Bundle display name	String	HMXcodeKit
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	XPC!
Bundle versions string, short	String	1.0
Bundle version	String	1
Minimum system version	String	\$(MACOSX_DEPLOYMENT_TARGET)
NSExtension	Dictionary	(2 items)
NSExtensionAttributes	Dictionary	(2 items)
XCSourceEditorCommandDefinitions	Array	(1 item)
Item 0	Dictionary	(3 items)
XCSourceEditorCommandClassName	String	SourceEditorCommand
XCSourceEditorCommandIdentifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER).SourceEditorCommand
XCSourceEditorCommandName	String	Source Editor Command
XCSourceEditorExtensionPrincipalClass	String	SourceEditorExtension
NSExtensionPointIdentifier	String	com.apple.dt.Xcode.extension.source-editor
Copyright (human-readable)	String	Copyright © 2016年 传智播客.黑马程序员. All rights reserved.

- `XCSourceEditorCommandClassName`
  - 代表该菜单触发执行哪一个类的函数
- `XCSourceEditorCommandIdentifier`
  - 代表该菜单功能对应的标识符，可以区分不同功能，设置成 `hmDeleteLine`，后面会用到。
- `XCSourceEditorCommandName`
  - 代表该菜单对应的菜单显示文字，可以在此配置，比如我们在此配置成 `Delete Line`

## 关键的类、属性、方法解释

- 查看 `SourceEditorExtension.m` 文件，其中默认有两个方法，已注释：

```
// 当该扩展启动的时候，可以在此做一些操作
- (void)extensionDidFinishLaunching
{
    // If your extension needs to do any work at launch, implement this optional method.
}
```

```
// 该方法返回类似于上图中 item0 的配置信息，如果你不想在 info.plist 中配置的话，也可以在这个方法里面返回
- (NSArray <NSDictionary <XCSourceEditorCommandDefinitionKey, id> *> *)commandDefinitions
{
    // If your extension needs to return a collection of command definitions that differs from those in its Info.plist, implement this optional property getter.
    return @[];
}
```

- 查看 `SourceEditorCommand.m` 文件，其中有一个方法：

```
- (void)performCommandWithInvocation:(XCSourceEditorCommandInvocation *)invocation completionHandler:(void (^)(NSError * _Nullable nilOrError))completionHandler
{
    // Implement your command here, invoking the completion handler when done. Pass it nil on success, and an NSError on failure.
    completionHandler(nil);
}
```

当点击 Xcode 中的扩展菜单的时候，会触发这个方法。处理完毕之后调用 `completionHandler`，如果没有错误回传 `nil`，否则将错误信息回调。也就是接下来我们要实现的代码就写在这个里面。该方法中传入一个类型为 `XCSourceEditorCommandInvocation` 的参数，其中有一个关键的属性：`buffer`(类型为`XCSourceTextBuffer`)，这个`buffer`就是当前代码编辑区域的文字信息，而我们最主要操作的对象就是它。

- `XCSourceTextBuffer` 中有以下两个重要的属性：
  - `lines` 数组，表示代码编辑区域的每一行代码文字
  - `selections` 数组，表示代码编辑区域中用户选中的信息，其数组中元素的类型是 `XCSourceTextRange`

```

/** The lines of text in the buffer, including line endings. Line
breaks within a single buffer are expected to be consistent. Adding
a "line" that itself contains line breaks will actually modify the
array as well, changing its count, such that each line added is a
separate element. */
@property (readonly, strong) NSMutableArray <NSString *> *lines;

/** The text selections in the buffer; an empty range represents an
insertion point. Modifying the lines of text in the buffer will
automatically update the selections to match. */
@property (readonly, strong) NSMutableArray <XCSourceTextRange
*> *selections;

```

- 而 `XCSourceTextRange` 中就两个类型为 `XCSourceTextPosition` 的结构体属性：start / end

```

// XCSourceTextRange 定义：
@interface XCSourceTextRange : NSObject <NSCopying>
// 开始选中的位置
@property XCSourceTextPosition start;
// 结束选中的位置
@property XCSourceTextPosition end;
@end

// 而 XCSourceTextPosition 的定义是：
typedef struct {
    // 代表当前位置是第几行
    NSInteger line;
    // 代表当前位置是第几列
    NSInteger column;
} XCSourceTextPosition;

```

那么到此为止，已经有一点体会的就是我们可以通过这些类知道当前代码编辑区域的文字内容以及选中状态。接下来就去实现 `快速删除当前行` 和



快速复制当前行。

## 实现功能

- 来到 `SourceEditorCommand.m` 新建方法 `deleteLines:`，在此方法中实现删除一行逻辑：

```
// 删除一行
- (void)deleteLines:(XCSourceEditorCommandInvocation *)invocation {

    // 查看是否有选中的行
    XCSourceTextRange *textRange = invocation.buffer.selections.firstObject;

    // 如果选中为空或者行数为0，直接跳过
    if (textRange == nil || invocation.buffer.lines.count == 0) {
        return;
    }

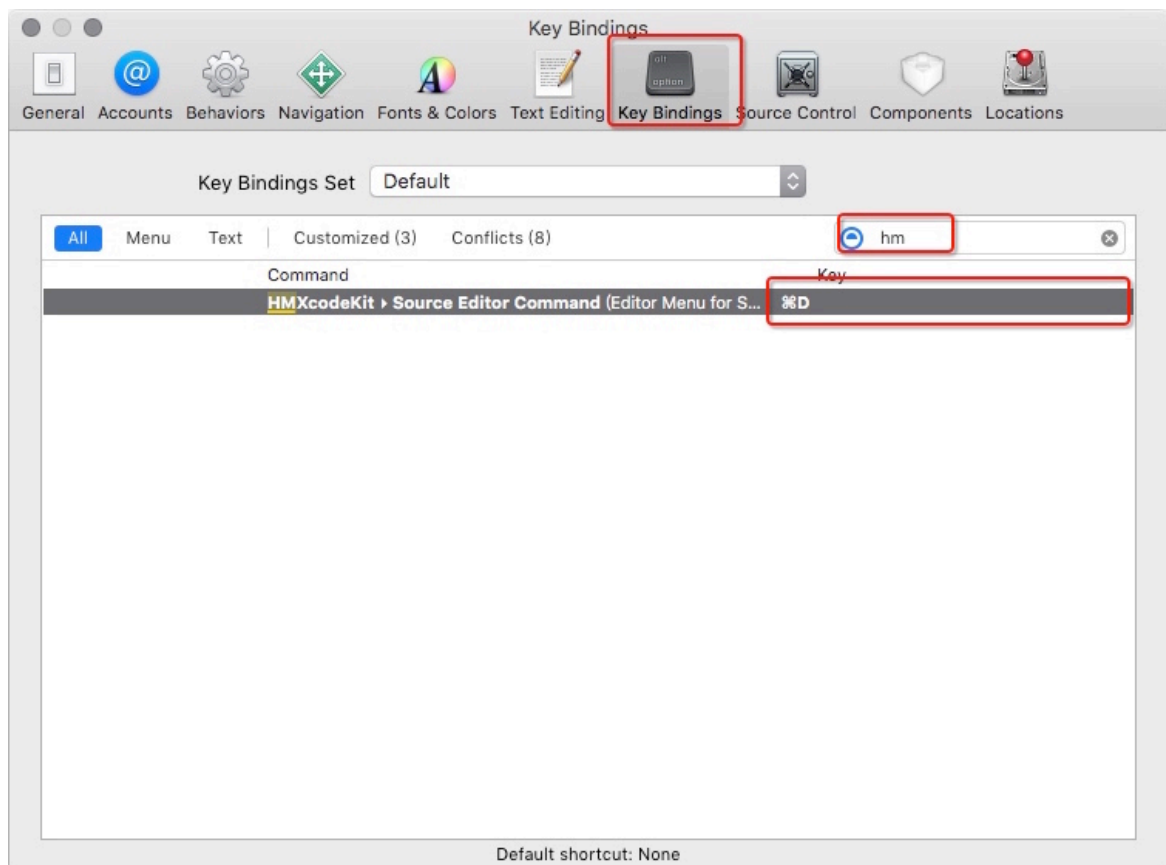
    // 确定开始删除的行数
    NSInteger loc = textRange.start.line;
    // 确定一共要删除多少行
    NSInteger len = textRange.end.line - loc + 1;
    // 通过范围初始化一个索引集
    NSIndexSet *indexSet = [NSIndexSet indexSetWithIndexesInRange:NSMakeRange(loc, len)];
    // 删了indexSet对应范围的行
    [invocation.buffer.lines removeObjectsAtIndexes:indexSet];
    // 重置选中范围
    textRange.end = textRange.start;
}
```

- 在 `SourceEditorCommand.m` 中的

`performCommandWithInvocation:completionHandler:` 方法中调用该方法:

```
- (void)performCommandWithInvocation:(XCSourceEditorCommandInvocation *)invocation completionHandler:(void (^)(NSError * _Nullable nilOrError))completionHandler{
    // 执行删除一行
    [self deleteLines:invocation];
    // 执行回调
    completionHandler(nil);
}
```

- 运行测试, 当点击 `Editor` -> `HMXcodeKit` -> `Delete Line` 的时候会把光标当前停留的行给删除。但是这样效率太慢, 我们可以为其配置一个快捷键, 步骤为:
  - 点击 `Preference` -> `Key Bindings`
  - 搜索 `HM` 找到 `Delete Line` 的菜单项
  - 双击该行后面的空白处, 设置快捷键为 `command + d`, 此时会产生快捷键冲突, 点击右边小红点。
  - 将另外一个与之冲突的快捷键点击减号清空
  - 再次测试。快捷键也可以实现删除一行



- 实现快速复制一行，需要在 `info.plist` 中的 `XCSourceEditorCommandDefinitions` 节点再添加一个菜单的配置，注意 `Identifier` 设置成 `hmDuplicateLine`，其关键代码为：

▼ NSExtension	Dictionary	(2 items)
▼ NSExtensionAttributes	Dictionary	(2 items)
▼ XCSourceEditorCommandDefinitions	Array	(2 items)
▼ Item 0	Dictionary	(3 items)
XCSourceEditorCommandClassName	String	SourceEditorCommand
XCSourceEditorCommandIdentifier	String	hmDeleteLine
XCSourceEditorCommandName	String	Delete Line
▼ Item 1	Dictionary	(3 items)
XCSourceEditorCommandClassName	String	SourceEditorCommand
XCSourceEditorCommandIdentifier	String	hmDuplicateLine
XCSourceEditorCommandName	String	Duplicate Line
XCSourceEditorExtensionPrincipalClass	String	SourceEditorExtension
NSExtensionPointIdentifier	String	com.apple.dt.Xcode.extension.source-editor

现在运行起来，在 Xcode 的 `Editor` 菜单中的 `HMXcodeKit` 就有两个子菜单，分别为 `Delete Line` 和 `Duplicate Line`

- 来到 `SourceEditorCommand.m` 新建方法 `duplicateLines:`，在此方法中实现快速复制一行：

```
// 复制一行
- (void)duplicateLines:(XCSourceEditorCommandInvocation *)invocation {
```

```

// 查看是否有选中的行
XCTSourceTextRange *textRange = invocation.buffer.selection
s.firstObject;
// 如果选中为空或者行数为0, 直接跳过
if (textRange == nil || invocation.buffer.lines.count == 0
) {
    return;
}
// 确定开始复制的行数
NSInteger loc = textRange.start.line;
// 确定一共要复制多少行
NSInteger len = textRange.end.line - loc + 1;
// 通过范围初始化一个索引集
NSIndexSet *indexSet = [NSIndexSet indexSetWithIndexesInRa
nge:NSMakeRange(loc, len)];
// 通过indexSet获取到选中的这几行
NSArray *selectLines = [invocation.buffer.lines objectsAtI
ndexes:indexSet];
// 将选中的行插入到指定的位置
[invocation.buffer.lines insertObjects:selectLines atIndex
es:indexSet];
}

```

- 在 `SourceEditorCommand.m` 中的 `performCommandWithInvocation:completionHandler:` 方法中根本不同的 `Identifier` 做不同的操作:

```

- (void)performCommandWithInvocation:(XCSourceEditorCommandInv
ocation *)invocation completionHandler:(void (^)(NSError * _Nu
llable nilOrError))completionHandler {
    if ([invocation.commandIdentifier isEqualToString:@"hmDele
teLine"]) {
        // 执行删除当前光标所在行
        [self deleteLines:invocation];
    } else if ([invocation.commandIdentifier isEqualToString:@"h

```

```
mDuplicateLine")) {  
    // 执行复制当前光标所在行  
    [self duplicateLines:invocation];  
}  
// 回调completionHandler 如果传入nil，代表没有错误。如果出现错误直接回调对应的错误  
completionHandler(nil);  
}
```

- 按照上面方式设置其对应的快捷键为 `command + option + ↓` 并运行测试