iOS 8 出色的跨应用通信效果：解读 Action 扩展

用程序扩展最初于 WWDC 2014 大会上正式亮相，这是一种将 iOS 应用程序功能扩展至系统其它组成部分的途径、而且能够实现更为出色的跨应用通信效果。

应用程序扩展最初于 WWDC 2014 大会上正式亮相，这是一种将 iOS 应用程序功能扩展至系统其它组成部分的途径、而且能够实现更为出色的跨应用通信效果。

举例为说，大家可以利用 Today 扩展创建出能够显示在通知中心之内的功能部件、Sharing 扩展则帮助用户将信息共享至社交网络当中，而 Action 扩展的作用在于允许用户执行当前内容——包括将其以不同方式显示或者对内容作出更改。在今天的上手指南当中，我们将了解如何从零开始创建一项 Action 扩展。

虽然这篇文章并不会对大家的知识储备提出太多硬性要求，但我还是建议读者朋友能够首先阅读一些相关资料，从而在本文指导之后得以更轻松地掌握更多扩展创建知识。

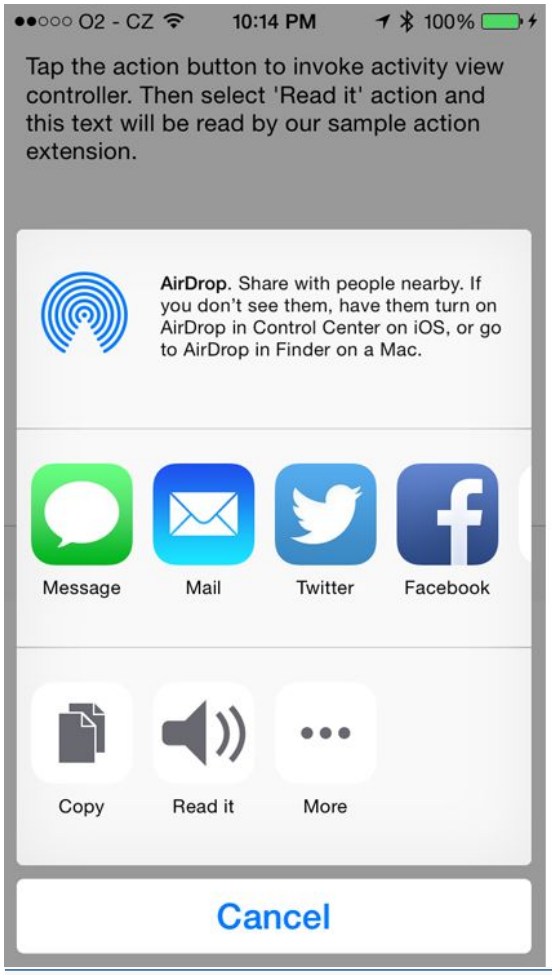
WWDC 大会演讲：为 iOS 及 OS X 创建扩展——第一部分与第二部分

应用程序扩展编程指南

1. 我们要创建什么？

我们将要创建一项名为“Read it”的简单 Action 扩展。这项扩展将把文本内容作为输入信息，并利用 AVFoundation 框架中的语音合成 API 将其朗读出来。我认为整个流程非常适合作为本教程的核心内容，因为在处理当中我们无需引入任何第三方依赖性、也不会产生其它难以处理的问题。

以下为我们在创建结束之后所得到的扩展功能效果。大家可以点击此处从 GitHub 上下载到本教程中的创建结果。



2. 创建一项 Action 扩展

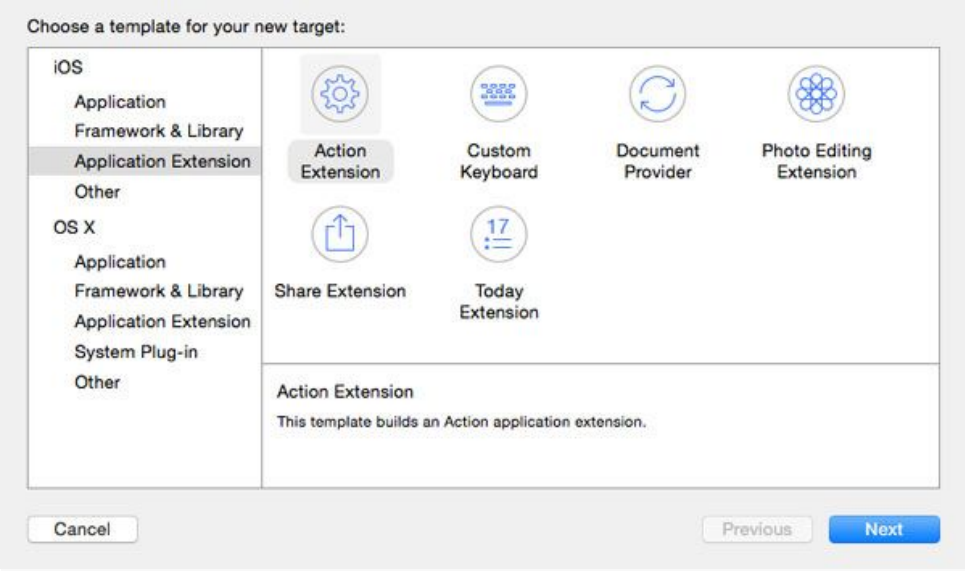
第一步：项目设置

首先要做的是启动 Xcode 6.1 或者更高版本，而后创建一个新项目。在 Xcode 的 File 菜单当中选择 New > Project…，然后从模板列表当中选择 Single View Application。

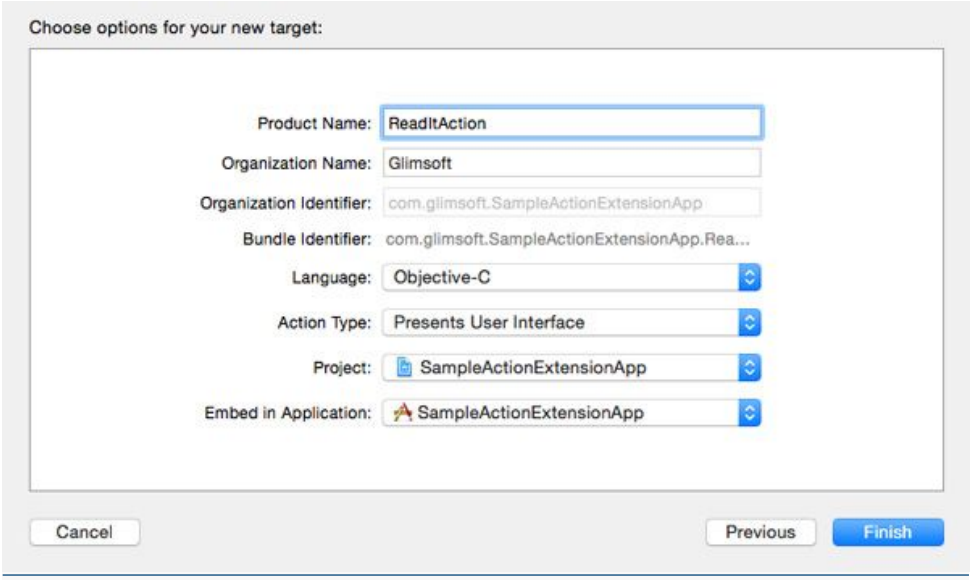
点击 Next 并为我们的项目设定一个 SampleActionExtensionApp 名称。输入一个 Organization Identifier 并将 Devices 类型设置为 iPhone。在本教程当中，我们将使用的编程语言为 Objective-C。

第二步：添加目标

在完成了以上项目创建工作之后，大家接下来可以为 Action 扩展添加一个目标了。从 File 菜单下选择 New > Target…。在左侧面板当中，从 iOS 选项处选择 Application Extension，并在选定 Action Extension 后点击 Next。



现在将 Product Name 设置为 ReadItAction。此外还需要注意其它一些选项，特别是 Action Type。我们稍后再就这一话题进行深入探讨。现在点击 Finish 以创建 Action 扩展。



现在系统会询问大家是否打算激活这套 ReadItAction 项目。暂时点击 Cancel，因为我们之后会通过运行内容应用来安装这一 Action 扩展、而非直接加以激活。

Action 类型

Action 扩展共分为两种类型，其一配备用户界面、另一种则不配备用户界面。大家可能会觉得奇怪——不配备用户界面的 Action 扩展到底有什么实实在在的好处？下面就让我为大家作出解释。

不具备用户界面的 Action 扩展以内容变更为目标作用于当前项目。举例来说，一项 Action 扩展能够去除相片当中的红眼现象，而且其完全无需用户界面作为配合。内容性应用随后可以对这部分经过变更的内容加以运用，在以上实例中即为完成了修正的相片素材。

配备用户界面的 Action 扩展则可以表现为全屏形式或者格式表形式。Action 扩展目标模板采用的是全屏幕形式，因此我们也将在接下来的创建过程中将其作为设计思路。

第三步：用户界面的实现

现在我们已经完成了基础性设置流程，也就是做好了创建用户界面的各项准备工作。下面我们将从应用程序内容开始入手。

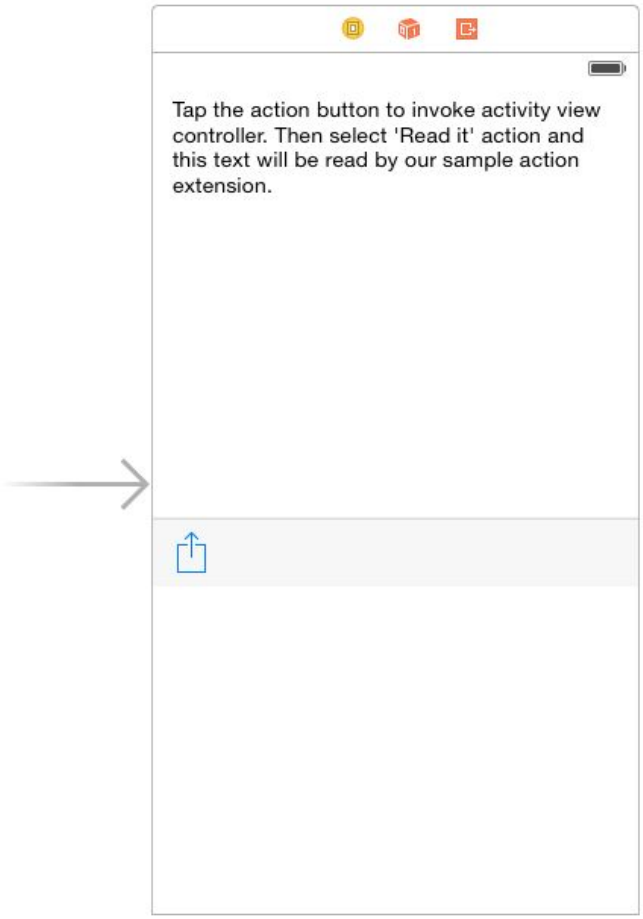
首先点击 Project Navigator 当中 SampleActionExtensionApp 组之下的 Main.storyboard。在右侧面板当中，选择 File Inspector 并取消对 Use size classes 的勾选。请注意，如果大家打算创建的是一款真正的应用程序且不需要对 iPad 提供支持，那么使用 size classes（即尺寸类）可能会是最好的选择。

下面打开 Object Library，并将文本视图与工具栏拖拽至该视图当中。在右侧的 Size Inspectoron 中将文本视图的框体设定为{x:8, y:20, width:304, height:288}。而对于工具栏，我们同样在 Size Inspector 中对其进行设置，具体参数为{x:0, y:308, width:320, height:44}。

工具栏当中应当包含一个栏按钮。将其选定，而后在 Attributes Inspector 当中将其 Style 设置为 Plain，并将其 Identifier 设置为 Action。

作为收尾工作,我们需要将文本视图当中的默认文本内容移除并替换为 Tap the action button to invoke activity view controller. Then select 'Read it' action and this text will be read by our sample Action extension.（点击 action 按钮以调用活动视图控制器。而后选择‘Read it’操作，这段文本将由我们的示例 Action 扩展朗读出来。）”

视图控制器的用户界面显示效果现在应当如下图所示：



当然，我们也可以将内容应用程序保留为空白。毕竟我们的目标在于构建一套示例性应用程序扩展，因此该应用本身并不需要真正的执行功能。不过我个人希望向大家展示从应用程序内部实现活动控制器调用有多么轻松，相信各位也能够借此了解如何将更多其它 Action 扩展纳入自己的应用当中。

在点击工具栏当中的按钮时，一套活动视图控制器将会显示出来，而我们则能够从这里对自己的 Action 扩展进行调用。选择这种方式的另一个理由在于，如果大家打算将自己的扩展发布到 App Store 当中，那么其必须要作为一款真正应用程序的组成部分、而应用当然得拥有功能才可以顺利通过苹果官方的审批。

第四步：当前活动视图控制器

接下来，我们需要将一部分代码添加到 ViewController.m 当中。首先在视图控制器的类扩展当中为文本视图创建一套外观，具体代码如下所示。

```
1.      @interface ViewController ()
```

```
2.
3.      @property (nonatomic, weak) IBOutlet UITextView *textView;
4.
5.      @end
```

创建一项名为 actionButtonPressed 的 action，在这里我们将对 UIActivityViewController 实例进行初始化与显示、并将其提供给用户。

```
1.      - (IBAction)actionButtonPressed:(id)sender {
2.          UIActivityViewController *activityVC = [[UIActivityViewController alloc] initWithActivityItems:@[self.textView.text]
3.                                                  applicationActivities:nil];
4.          [self presentViewController:activityVC animated:YES completion:nil];
5.      }
```

让我们再说回 Main.storyboard，通过点击 Control 并将文本视图外观方案从 View Controller Scene 下的 View Controller 对象中将其拖拽至文本视图内、然后在弹出的菜单中选择 textView，这就实现了文本视图外观与文本视图的对接。

为了与 action 方法实现对接，我们需要在工具当中选择该栏按钮并打开 Connections Inspector。将其从 Sent actions 下的 selector 当中拖拽至 View Controller 对象，而后在弹出的菜单内选择 actionButtonPressed: 。

到这里应用程序的用户界面已经设计完成并且正式交付使用，下面我们可以继续进行 Action 扩展创建工作了。

第五步：用户界面的实现

在 Project Navigator 当中，展开 ReadItAction 组并点击 MainInterface.storyboard。大家应该会注意到，该故事板并非空白、其中已经包含有一部分用户界面组件。我们可以对其中一部分加以利用，但具体的图象视图却帮不上什么忙。因此选定该图象视图并通过按下 Delete 键将其移除。

接下来打开 Object Library 并在下方的导航栏中添加一套文本视图。将其框体变更为{x: 8, y: 72, width: 304, height: 300}。最后，双击该导航栏的标题视图并将标题内容设定为“Text reader”。

第六步：ActionViewControlle 的实现

到了这一步，我们需要搞定 Action 扩展的实现工作。在 Project Navigator 当中选择 ActionViewController.m，并对其实施以下几项变更。

下面的导入语句会将一条导入语句添加到 AVFoundation 框架当中，这样我们就能将其语音合成 API 运用在自己的 Action 扩展当中了。

```
1 @import AVFoundation;
```

在 ActionViewController 类的类扩展当中，移除其中的 imageView 外观并添加到我们之前曾经添加至文本视图当中的新外观。

```
@interface ActionViewController ()

@property (nonatomic, strong) IBOutlet UITextView *textView;

@end
```

除此之外，我们还需要对 ActionViewController 类中的 viewDidLoad 方法作出以下几项变更。

```
30 - (void)viewDidLoad {
    [super viewDidLoad];

    //从扩展背景信息中获取我们打算处理的项目。
    // 在我们的 Action extension 当中, 我们只需要一个输入项目（即文本）, 因此我们使用数组中的第一个项目。
    NSExtensionItem *item = self.extensionContext.inputItems[0];
    NSItemProvider *itemProvider = item.attachments[0];

    if ([itemProvider hasItemConformingToTypeIdentifier:(NSString *)kUTTypePlainText]) {

        // 这是一段纯文本！
        __weak UITextView *textView = self.textView;

        [itemProvider loadItemForTypeIdentifier:(NSString *)kUTTypePlainText options:nil completionHandler:^(NSString *item, NSError *error) {

            if (item) {
```

```
                [[NSOperationQueue mainQueue] addOperationWithBlock:^{

                    [textView setText:item];

                    //设置语音合成并加以启动
                    AVSpeechSynthesizer *synthesizer = [[AVSpeechSynthesizer alloc]init];
                    AVSpeechUtterance *utterance = [AVSpeechUtterance speechUtteranceWithString:textView.text];
                    [utterance setRate:0.1];
                    [synthesizer speakUtterance:utterance];
                }];
            }
        }];
    }
}
```

整个实现过程还是非常容易理解的。在 viewDidLoad 当中，我们得到了输入文本内容、将其分配给文本视图、而后再创建一个能够将其朗读出来的语音合成对象。

第七步：配置 Action 扩展

虽然我们已经接近整个项目创建流程的尾声，但仍有一些需要着重关注的细节问题。首先，我们需要将故事板中的文本视图与我们此前已经创建完成的外观方案相对接。

打开 MainInterface.storyboard 并将文本视图与 Image 场景相对接，正如我们之前在 Main.storyboard 当中完成的操作一样。

接下来我们还需要为 Action 扩展指定所能支持的数据类型。在这一次的实例当中，我们只需要支持纯文本数据即可。展开 Supporting Files 组并选择 Info.plist。在 Info.plist 当中，遵循 NSExtension > NSExtensionAttributes > NSExtensionActivationRule 导航流程，最后将 NSExtensionActivationRule 的类型由 String 变更为 Dictionary。

在已经展开的 dictionary 当中，点击旁边的+号按钮，从而添加一个新的子级键。将其名称设置为 NSExtensionActivationSupportsText，类型设定成 Boolean，而值则取为 YES。这样一来，我们就能确保自己的 Action 扩展只在输入项目包含文本内容时才会显示出来。

仍然是在 Info.plist 当中，我们要将 Bundle Display Name 变更为 Read It。这样看起来会更加清晰。下图所示为 Info.plist 文件当中相关部分的设置结果：

▼NSExtension	Dictionary	(3 items)
▼NSExtensionAttributes	Dictionary	(1 item)
▼NSExtensionActivationRule	Dictionary	(1 item)
NSExtensionActivationSupportsText	Boolean	YES
NSExtensionMainStoryboard	String	MainInterface
NSExtensionPointIdentifier	String	com.apple.ui-services

第八步

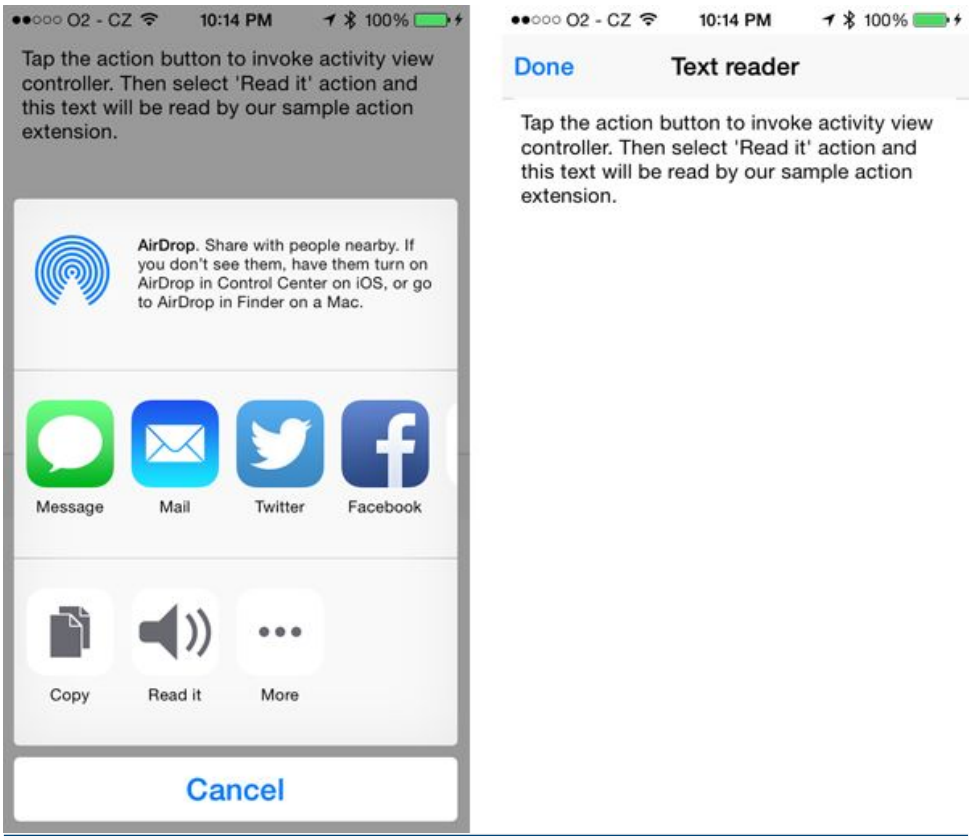
作为画龙点睛之笔，大家可以为 Action 扩展添加一个图标。在 Project Navigator 当中，选择该项目并在目标之下选定 ReadItAction 目标。在 App Icons and Launch Images 部分中的 General 标签下点击 App Icons Source 旁边的 Use Asset Catalog。根据提示，我们接下来需要点击 Migrate。而后慎用至资产目录并将以下图标拖拽至 iPhone App iOS 7，8 60pt 2x 位置。

完成应用程序的构建与运行步骤，看看一切是否像我们预期的那样运转正常。不过还有一个需要关注的问题：如果声音图标没有被正常显示在 Action 扩展之内，大家需要确保主 Images.xcassets 文件已经被正确复制到了扩展目标当中。

要完成这项操作，我们需要在 Project Navigator 当中选定该项目并从 Targets 列表当中选择 ReadItAction 目标。打开屏幕顶端的 Build Phases 标签并展开 Copy Bundle Resources 步骤。如果 Images.xcassets 文件并未出现在资源列表当中，那么点击其中的小加号将其手动添加到列表之内。

3. 运行及测试

最后要做的就是运行应用程序并尝试其各功能的起效情况。以下显示的两幅截图为运行当中的扩展外观。大家也可以尝试通过记事本应用调用该活动视图控制器，并让我们的扩展读取之前曾经记录过的文本内容。除此之外，我们不妨在相片应用当中打开活动列表，这时大家会发现自己的扩展并没有被列中其中。没错，这正好符合我们之前为其设置的激活规则。



总结

在今天的教程当中，大家了解了如何构建一套简单的 Action 扩展。我们还涉及到了一些基础性知识，即如何运用 AVFoundation 框架当中的语音合成 API。如果大家有举创建出更多扩展方案，也可以点击此处查看由 Cesar Tessarin 带来的 Today 扩展创建指南。

英文原文：http://code.tutsplus.com/tutorials/ios-8-how-to-build-a-simple-action-extension--cms-2279