**IOS扩展**

**一**、**扩展概述**

扩展（Extension）是iOS 8中引入的一个非常重要的新特性。扩展让app之间的数据交互成为可能。用户可以在app中使用其他应用提供的功能，而无需离开当前的应用。

扩展 (Extension) 是 iOS 8 和 OSX 10.10 加入的一个非常大的功能点，开发者可以通过系统提供给我们的扩展接入点 (Extension point) 来为系统特定的服务提供某些附加的功能。对于 iOS 来说，可以使用的扩展接入点有以下几个：

Today 扩展 - 在下拉的通知中心的 "今天" 的面板中添加一个 widget

分享扩展 - 点击分享按钮后将网站或者照片通过应用分享

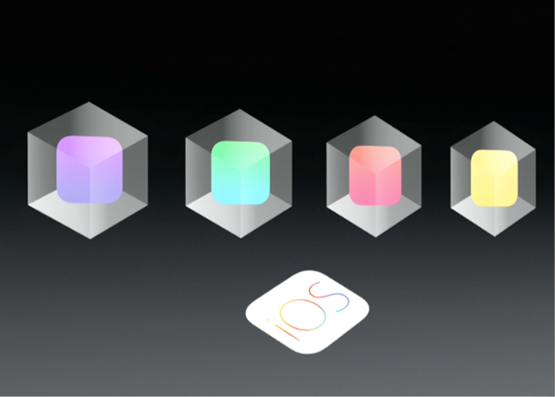
动作扩展 - 点击 Action 按钮后通过判断上下文来将内容发送到应用

照片编辑扩展 - 在系统的照片应用中提供照片编辑的能力

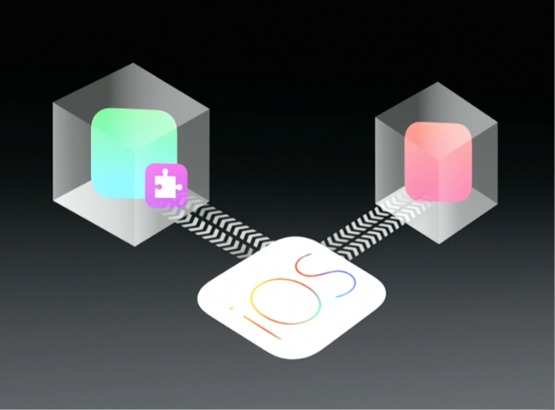
文档提供扩展 - 提供和管理文件内容

自定义键盘 - 提供一个可以用在所有应用的替代系统键盘的自定义键盘或输入法

在iOS 8系统之前，每一个app在物理上都是彼此独立的，app之间不能互访彼此的私有数据。

[](http://static.oschina.net/uploads/img/201409/24105542_c5C1.png)

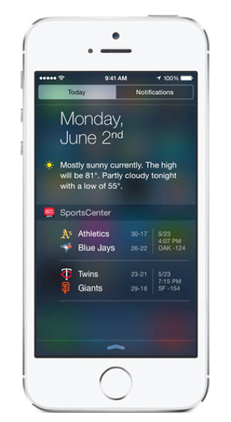
而在引入扩展之后，其他app可以与扩展进行数据交换。基于安全和性能的考虑，每一个扩展运行在一个单独的进程中，它拥有自己的bundle， bundle后缀名是.appex。扩展bundle必须包含在一个普通应用的bundle的内部。

[](http://static.oschina.net/uploads/img/201409/24105542_vm1b.png)

iOS 8系统有6个支持扩展的系统区域，分别是Today、Share、Action、Photo Editing、Storage Provider、Custom keyboard。支持扩展的系统区域也被称为扩展点。

**Today Widget**

对于赛事比分，股票、天气、快递这类需要实时获取的信息，可以在通知中心的Today视图中创建一个Today扩展实现。Today扩展又称为Widget。

[](http://static.oschina.net/uploads/img/201409/24105543_gYrz.png)

**Share**

在iOS 8之前，用户只有Facebook,Twitter等有限的几个分享选项可以选择。如果希望将内容分享到Pinterest，开发者则需要一些额外的努力。在iOS 8中，开发者可以创建自定义的分享选项。

[](http://static.oschina.net/uploads/img/201409/24105543_paHX.png)

**Action**

action在所有支持的扩展点中扩展性最强的一个。它可以实现转换另一个app上下文中的内容。苹果在WWDC大会上演示了一个Bing翻译动作扩展，它可以将在Safari中选中的文本翻译成不同的语言。

[](http://static.oschina.net/uploads/img/201409/24105543_txUV.png)

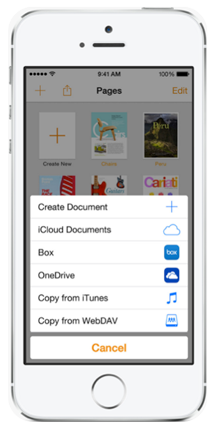
**Photo Editing**

在iOS 8之前，如果你想为你的照片添加一个特殊的滤镜，你需要进入第三方app中，这个过程是相当繁琐的。在iOS 8中，你可以直接在Photos中使用第三方app，如Instagram，VSCO cam、Aviary提供的Photo Editing扩展完成对图片的编辑，而无需离开当前的app。

[](http://static.oschina.net/uploads/img/201409/24105543_3nBR.png)

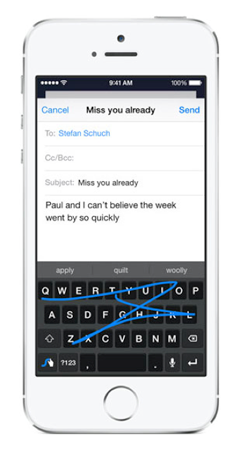
**Storage Provider**

Storage Provider让跨多个文件存储服务之间的管理变得更简单。类似Dropbox、Google Drive等存储提供商通过在iOS 8中提供一个Storage Provider扩展，app直接可以使用这些扩展检索和存储文件而不再需要创建不必要的拷贝。

[](http://static.oschina.net/uploads/img/201409/24105543_QTSz.png)

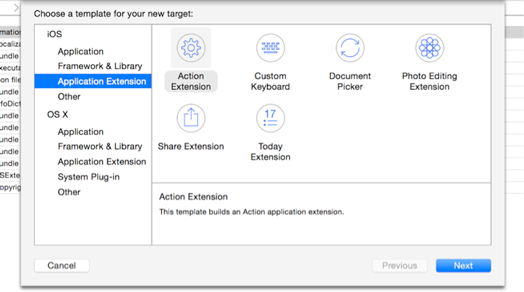
**Custom Keyboard**

苹果公司在2007年率先推出了触摸屏键盘，但一直没多大改进。在这一方面，Android则将键盘权限开放给了第三方开发者，所以出现了许多像Swype，SwiftKey等优秀的键盘输入法。在iOS 8中，苹果终于将键盘权限开发给了第三方开发者，自定义键盘输入法可以让用户在整个系统范围内使用。

[](http://static.oschina.net/uploads/img/201409/24105543_EbzB.png)

**二、创建扩展与发布扩展**

在创建扩展之前，你需要创建一个用来包含扩展的常规的app项目。这个包含扩展的app被称为containing app。在创建好containg app之后，选择File->New->Target菜单，从弹出的对话框中选择一个适当的扩展目标模板。每一个扩展目标模板都包含了与扩展点相关的文件和设置。一个containing app可以包含多个不同类型的扩展。

[](http://static.oschina.net/uploads/img/201409/24105545_IdF4.png)

每一个扩展目标模板包含一个头文件和实现文件，一个Info.plist文件，以及一个storyboard文件。Info.plist文件包含了对扩展的配置信息，其中最重要的键是NSExtension。下面列出了一个NSExtension可能包含的常用键值对。

<key>NSExtension</key>

 <dict>

    <key>NSExtensionAttributes</key>

    <dict>

            <key>NSExtensionActivationRule</key> <!--1-->

            <dict>

                <key>NSExtensionActivationSupportsImageWithMaxCount</key>

                <integer>10</integer>

                <key>NSExtensionActivationSupportsMovieWithMaxCount</key>

                <integer>1</integer>

                 </dict>

            <key>NSExtensionJavaScriptPreprocessingFile</key> <!--2-->

            <string>MyJavaScriptFile</string>

           <key>NSExtensionPointVersion</key>

           <string>1.0</string>

       </dict>

     <key>NSExtensionMainStoryboard</key>  <!--3-->

     <string>MainInterface</string>

       <key>NSExtensionPointIdentifier</key>  <!--4-->

       <string>com.apple.ui-services</string>

     <key>NSExtensionPrincipalClass</key>  <!--5-->

       <string>ActionViewController</string>

</dict>

1)   NSExtensionActivationRule定义了当前的扩展支持的数据类型及数据项个数，例如当前的设置只支持图片格式和视频格式的数据，并且最多不超过10张图片和1个视频。

2)   NSExtensionJavaScriptPreprocessingFile用于配置与脚本交互的JS脚本文件的名字。

3)   NSExtensionMainStoryboard配置扩展的Storyboard文件名。

4)   NSExtensionPointIdentifier用于表示扩展点，每一个扩展点拥有一个唯一的名字。

5)   NSExtensionPrincipalClass配置当扩展启动时，扩展点首先要实例化的类

为了将扩展提交苹果商店，你需要提交你的containg app。并且需要注意，除了扩展必须包含功能以外，同时containg app还需要提供一些功能，而针对OS X平台的扩展则无此限制。当用户安装了你的containg app，containg app中包含的扩展也会一同被安装。

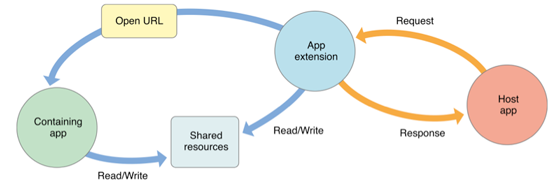
**三、理解扩展如何运作**

在安装扩展之后，扩展并不会自动运行，用户必须执行特定的操作来启用扩展。如果是Today扩展，用户可以在通知中心的Today视图中编辑启用扩展。如果是自定义键盘扩展，用户需要在系统设置的通用选项下的键盘选项中启用自定义键盘扩展。而如果是Share扩展，用户只需点击系统提供的分享按钮[http://static.oschina.net/uploads/img/201409/24105545_U1fx.png](http://static.oschina.net/uploads/img/201409/24105545_U1fx.png)，即可在分享列表中找到分享扩展。

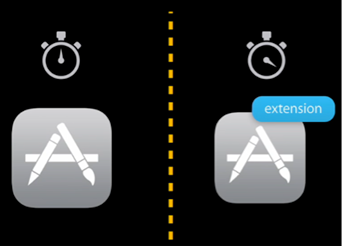
一个扩展并不是一个app，它的生命周期和运行环境不同于普通app。在生命周期方面，扩展的生命周期从用户在另一个app中选择了扩展开始，一直到扩展完成了用户的请求生命周期结束。在运行环境方面，扩展的限制要比普通app更严格，扩展的可用内存上限以及可用的API都比普通app要少。严格限制扩展的内存是因为在同一时间可能会有多个扩展同时运行，如Widget扩展。如果API声明包含NS\_EXTENSION\_UNAVAILABLE宏，则此API在扩展中将不可用，常见的API如：

+ (UIApplication \*)sharedApplication NS\_EXTENSION\_UNAVAILABLE\_IOS("Use view controller based solutions where appropriate instead.");

调用扩展的应用称为host app，对于Widget扩展，host app就是Today。host app会在扩展的有效生命周期内定义一个扩展上下文。通过扩展上下文，host app可以和扩展互传数据。注意，扩展只和host app直接通信，扩展与containg app以及containing app与host app之间不存在通信关系，如果扩展需要打开containg app，则通过自定义URL scheme方式实现，而不是直接向containg app发送消息。三者的关系见下图：

[](http://static.oschina.net/uploads/img/201409/24105545_FHs0.png)

扩展是一个单独的个体。扩展拥有独立的target，独立的bundle文件，独立的运行进程，独立的地址空间。这意味着即使你的containing app不在运行，系统也可以启动扩展。或者你的containing app处于挂起状态，同样不会影响扩展的运行。所以系统可以单独对扩展执行优化。扩展与containg app的关系：

[](http://static.oschina.net/uploads/img/201409/24105546_ieS9.png)

**四、设计扩展过程中常见的几个问题**

1. containg app与扩展如何通过扩展上下文互传数据

在iOS 8中，UIViewController新增了一个扩展上下文属性extensionContext。来处理containing app与扩展之间的通信，上下文的类型是NSExtensionContext。假设你现在需要在host app中将一张图片传递给扩展做滤镜处理，host app中的代码如下：

    UIActivityViewController \*activityViewController = [[UIActivityViewController alloc] initWithActivityItems:@[[self.imageView image]] applicationActivities:nil];

    [self presentViewController:activityViewController animated:YES completion:nil];

当用户在弹出的Action列表中选择了扩展，扩展将被启动，然后在扩展的viewDidLoad方法中，通过extensionContext检索host app传回的数据项。扩展中的代码如下：

- (void)viewDidLoad {

    [super viewDidLoad];

**NSExtensionItem** \*imageItem = [self.**extensionContext**.inputItems firstObject];

    if(!imageItem){

        return;

    }

**NSItemProvider** \*imageItemProvider = [[imageItem attachments] firstObject];

    if(!imageItemProvider){

        return;

    }

   // 检查是否包含文本

    if([imageItemProvider hasItemConformingToTypeIdentifier:(NSString \*)kUTTypeImage]){

        [imageItemProvider loadItemForTypeIdentifier:**(NSString \*)kUTTypeImage** options:nil completionHandler:^(**UIImage \*image**, NSError \*error) {

            if(image){

                dispatch\_async(dispatch\_get\_main\_queue(), ^{

                    self.imageView.image = image;

                });

            }

        }];

    }

}

上述代码中，extensionContext表示一个扩展到host app的连接。通过extionContent，你可以访问一个NSExtensionItem的数组，每一个NSExtensionItem项表示从host app传回的一个逻辑数据单元。你可以从NSExtensionItem项的attachments属性中获得附件数据，如音频，视频，图片等。每一个附件用NSItemProvider实例表示。上述代码中NSItemProvider的loadItemForTypeIdentifier实例方法的第一个参数是(NSString \*)kUTTypeImage，如果你需要处理的是文本类型，参数则为(NSString \*)kUTTypeText，相应的处理代码则变成：

if([itemProvider hasItemConformingToTypeIdentifier:**(NSString \*)kUTTypeText**]){

    [imageItemProvider loadItemForTypeIdentifier:**(NSString \*)kUTTypeText** options:nil completionHandler:^(**NSAttributedString \*string**, NSError \*error) {

        if (string) {

            // 在这里处理文本

        }

    }];

}

当扩展处理完host app传回的图片数据后，它需要将处理好的数据再传给host app。在扩展中的代码如下：

-(IBAction)done:(id)sender{

    NSExtensionItem\* extensionItem = [[NSExtensionItem alloc] init];

    [extensionItem setAttachments:@[[[NSItemProvider alloc] initWithItem:[self.imageView image] typeIdentifier:(NSString\*)kUTTypeImage]]];

    [self.extensionContext completeRequestReturningItems:@[extensionItem] completionHandler:nil];

}

最后一步是host app接收来自扩展传回的数据，在host app中的代码如下：

[activityViewController **setCompletionWithItemsHandler**:^(NSString \*activityType, BOOL completed, NSArray \*returnedItems, NSError \* error){

        if([returnedItems count] > 0){

            NSExtensionItem\* extensionItem = [returnedItems firstObject];

            NSItemProvider\* imageItemProvider = [[extensionItem attachments] firstObject];

            if([imageItemProvider hasItemConformingToTypeIdentifier:(NSString \*)kUTTypeImage]){

                [imageItemProvider loadItemForTypeIdentifier:(NSString \*)kUTTypeImage options:nil completionHandler:^(UIImage \*item, NSError \*error) {

                    if(item && !error){

                        dispatch\_async(dispatch\_get\_main\_queue(), ^{

                            [self.imageView setImage:item];

                        });

                    }

                }];

            }

        }

    }];

上述代码主要是通过设置一个completionBlock处理数据回调。

注意，所有的扩展都是一个UIViewController。所以UIViewController的所有生命周期方法，如viewWillAppear：、viewWillDisappear：等在扩展中都是可以使用的。

2. 如何在扩展中打开containing app

在一般情况下，扩展和containing app不存在通信关系。但是有时候需要在扩展中打开containing app，如iOS 7中预置的日历Widget。在常规的app中，可以使用如下代码在A app中打开B app：

    if ([[UIApplication sharedApplication] canOpenURL:[NSURL URLWithString:customURL]]) {

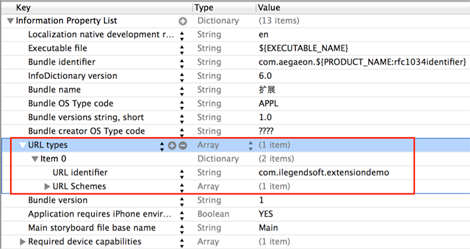
        [[UIApplication sharedApplication] openURL:[NSURL URLWithString:customURL]];

    }

但是之前有讲到，sharedApplication API在扩展中被禁止使用，所以为了实现同样的功能，NSExtensionContext定义了一个新的方法用来打开containing app：

- (void)openURL:(NSURL \*)URL completionHandler:(void (^)(BOOL success))completionHandler;

在调用此方法之前，需要在containg app中定义一个自定义URL Scheme。定义方法可参见[链接](http://www.cocoachina.com/applenews/devnews/2014/0522/8514.html)，最终的结果如下图：

[](http://static.oschina.net/uploads/img/201409/24105546_XvaH.png)

在扩展中打开containing app的代码如下：

- (IBAction)openContainingApp:(id)sender {

    NSURL \*url = [NSURL URLWithString:@"ExtensionDemo://"];

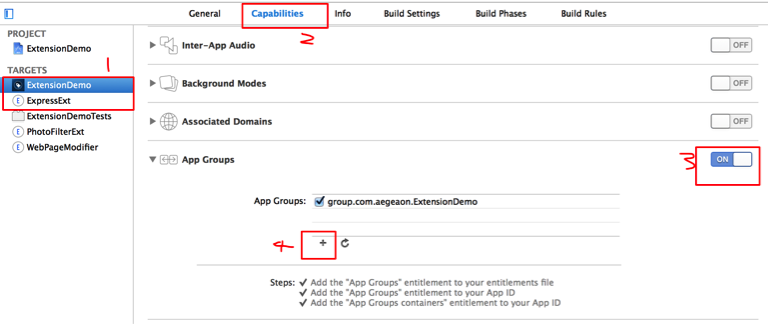
    [self.extensionContext openURL:url completionHandler:^(BOOL success) {

    }];

}

3. 如何实现containing app与扩展共享数据

扩展和containing app各自拥有自己的数据容器，虽然扩展内嵌在containing app的内部，但是它们并不可以互访彼此的数据。为了实现containing app与扩展的数据共享，苹果在iOS 8中引入了一个新的概念——App Group。为了开启App Group，找到你的containing app目标，在右侧找到Capabilities标签，定位到App Groups分组，如下图所示。

[](http://static.oschina.net/uploads/img/201409/24105546_W6H9.png)

然后选择你需要共享数据的扩展目标，重复执行一次操作，注意两次的App Group名要相同，不要添加新的条目。当开启App Group后，你可以使用NSUserDefaults方法访问共享区域，如下述代码，注意**不是**[NSUserDefaults standardUserDefaults]：

\_sharedUserDefault= [[NSUserDefaults alloc] initWithSuiteName:@"group.com.aegeaon.ExtensionDemo"];

你也可以使用NSFileManager的containerURLForSecurityApplicationGroupIdentifier方法访问共享数据区：

- (BOOL)saveTextByNSFileManager {

    NSError \*err = nil;

    NSURL \*containerURL = [[NSFileManager defaultManager] containerURLForSecurityApplicationGroupIdentifier:@"group.wangzz"];

    containerURL = [containerURL URLByAppendingPathComponent:@"Library/Caches/good"];

    NSString \*value = \_textField.text;

    BOOL result = [value writeToURL:containerURL atomically:YES encoding:NSUTF8StringEncoding error:&err];

    if (!result) {

        NSLog(@"%@",err);

    } else {

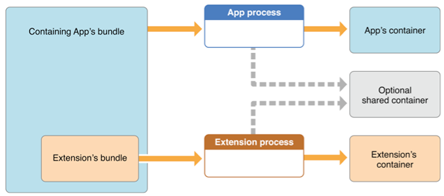
        NSLog(@"save value:%@ success.",value);

    }

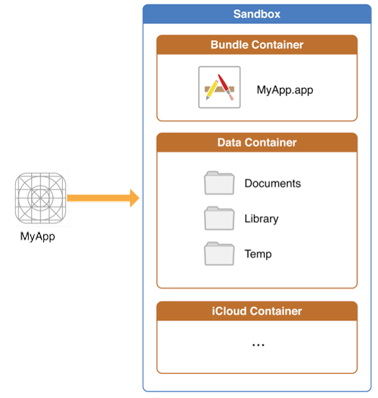
    return result;

}

App Group区域在containing app与扩展之间所处的关系图：

[](http://static.oschina.net/uploads/img/201409/24105546_2U3B.png)

你可能注意到了，在Xcode 6中iPhone模拟器的位置已经发生了变化。与此同时，在[iOS 8 release Note](https://developer.apple.com/library/prerelease/ios/releasenotes/General/RN-iOSSDK-8.0/index.html" \l "//apple_ref/doc/uid/TP40014223-CH1-DontLinkElementID_38)中有提到，app的沙盒结构已经发生了改变，现在它被划分成了三个容器，Bundle容器、Data容器、iCloud容器。iOS 8 app沙盒目录结构如下图：

[](http://static.oschina.net/uploads/img/201409/24105546_tt5k.png)

为了具体了解沙盒目录的布局，使用如下代码分别在containing app和扩展中打印出App Group目录，app bundle目录，以及Document目录：

- (void)logAppPath

{

    //app group路径

    NSURL \*containerURL = [[NSFileManager defaultManager] containerURLForSecurityApplicationGroupIdentifier:@"group.com.aegeaon.ExtensionDemo"];

    NSLog(@"app group:\n%@",containerURL.path);

    //打印可执行文件路径

    NSLog(@"bundle:\n%@",[[NSBundle mainBundle] bundlePath]);

    //打印documents

    NSArray \*paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);

    NSString \*path = [paths objectAtIndex:0];

    NSLog(@"documents:\n%@",path);

}

在containing app中执行logAppPath方法的结果如下：

app group:

/Users/aegaeon/Library/Developer/CoreSimulator/Devices/**72CA5D31-9509-4076-BC94-BF4D29DC0151**/data/Containers/**Shared**/AppGroup/**5B4CFBD8-D95D-4F01-9268-D9F79792147D**

bundle:

/Users/aegaeon/Library/Developer/CoreSimulator/Devices/**72CA5D31-9509-4076-BC94-BF4D29DC0151**/data/Containers/**Bundle**/Application/**EED1F771-A8AD-4A97-97F3-2B0A57936C17**/ExtensionDemo.app

documents:

/Users/aegaeon/Library/Developer/CoreSimulator/Devices/**72CA5D31-9509-4076-BC94-BF4D29DC0151**/data/Containers/**Data**/Application/**95DBF43A-8B4B-426C-9A3A-C1745FCB3FA2**/Documents

在扩展中执行logAppPath方法的结果如下：

app group:

/Users/aegaeon/Library/Developer/CoreSimulator/Devices/**72CA5D31-9509-4076-BC94-BF4D29DC0151**/data/Containers/**Shared**/AppGroup/**5B4CFBD8-D95D-4F01-9268-D9F79792147D**

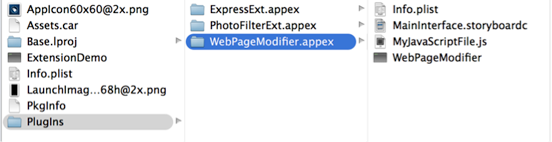
bundle:

/Users/aegaeon/Library/Developer/CoreSimulator/Devices/**72CA5D31-9509-4076-BC94-BF4D29DC0151**/data/Containers/**Bundle**/Application/**EED1F771-A8AD-4A97-97F3-2B0A57936C17**/ExtensionDemo.app/PlugIns/ExpressExt.appex

documents:

~/Documents

其中标注为红色的意思是每次运行目录名都会发生变化。标注为绿色的表示文件名不会变化的，标准为橘色也验证了iOS 8中沙盒目录被划分的说法。其中也可以看出扩展扩展名为appex,它包含在containing app的PlugIns目录内。下图展示了扩展目录在Finder中的结构：

[](http://static.oschina.net/uploads/img/201409/24105547_lsU3.png)

**以下是真机打出的路径，黑色为扩展路径，红色为主程序路径。可以看出app group路径相同，bundle路径和document路径不同**

**app group:**

**/private/var/mobile/Containers/Shared/AppGroup/94B5C6D4-C017-4B4F-98C3-E79284D968B0**

**/private/var/mobile/Containers/Shared/AppGroup/94B5C6D4-C017-4B4F-98C3-E79284D968B0**

**bundle:**

**/private/var/mobile/Containers/Bundle/Application/4A2720EA-BBEC-4856-938D-1DF61619396D/AutoNavi.app/PlugIns/extension.appex**

**/private/var/mobile/Containers/Bundle/Application/4669EB59-2879-44BE-8449-04E4A36D2FC0/AutoNavi.app**

**documents:**

**/var/mobile/Containers/Data/PluginKitPlugin/058937F2-3959-4541-B00D-7E1CC1223D09/Documents**

**/var/mobile/Containers/Data/Application/375FD59E-9559-48F3-B68F-6C24432B31C3/Documents**

4. 如何让扩展访问到网页内容

在WWDC上，苹果演示了在Safari for iOS中使用Bing Action扩展将当前页面翻译为其他语言。考虑一下，为了完成这个功能，扩展和浏览器之间一定要建立一个连接，浏览器负责将选中的文本发给扩展，扩展将翻译的结果发回浏览器。为了实现这个机制，这里需要借助一个Javascript脚本，使用JS脚本可以访问网页的DOM。脚本的内容很简单，只包含两个方法，脚本文件名为MyJavaScriptFile.js。代码如下：

var MyExtensionJavaScriptClass = function() {};

MyExtensionJavaScriptClass.prototype = {

    run: function(arguments) {

        arguments.completionFunction({"baseURI": document.baseURI});

    },

    finalize: function(arguments) {

        var newContent = arguments["content"];

        document.write(newContent);

    }

};

var ExtensionPreprocessingJS = new MyExtensionJavaScriptClass;

其中包含一个run()和finalize()方法。当Safari一加载好你的JS文件，就会立即调用run方法，当你在扩展中调用了completeRequestReturningItems:expirationHandler:completion:方法，Safari会调用finalize()方法。在run()方法中，Safari提供了一个arguments参数，使用它可以利用键值对的形式将数据传给扩展。在上述代码中，传给扩展的键值对是：@{@"baseURI" : document.baseURI}。在finalize()方法中，当你调用了completeRequestReturningItems:expirationHandler:completion:方法，方法第一个参数的值会传给finalize()方法的arguments形参。在上述代码中，finalize()接收到参数后，将内容写入了当前的文档。

为了Safari能够调用正确调用到JS文件，需要在扩展的Info.plist文件中添加如下配置：

<key>NSExtensionAttributes</key>

 <dict>

      <key>NSExtensionJavaScriptPreprocessingFile</key>

      <string>MyJavaScriptFile</string>

  </dict>

在你的扩展中，为了取得从JS脚本传回的键值对，你需要为NSItemProvider的方法loadItemForTypeIdentifier:options:completionHandler:指定kUTTypePropertyList数据类型，在取得返回的键值字典后，使用NSExtensionJavaScriptPreprocessingResultsKey键取值，代码如下：

    NSExtensionContext \*context = self.extensionContext;

    NSExtensionItem \*item = context.inputItems.firstObject;

    NSItemProvider \*provider = item.attachments.firstObject;

    [provider loadItemForTypeIdentifier:(NSString \*)**kUTTypePropertyList**

                                options:nil

                      completionHandler:^(id<NSSecureCoding> item, NSError \*error) {

                          NSDictionary \*results = (NSDictionary \*)item;

                          NSString \*baseURI = [[results objectForKey:**NSExtensionJavaScriptPreprocessingResultsKey**] objectForKey:@"baseURI"];

                          NSLog(@"%@", baseURI);

                      }];

为了在扩展中将处理后的结果传给脚本，你需要使用NSItemProvider的initWithItem:typeIdentifier:包装键值对。代码如下：

NSExtensionItem \*extensionItem = [[NSExtensionItem alloc] init];

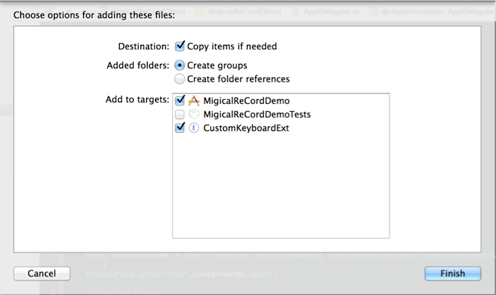
extensionItem.attachments = @[[[NSItemProvider alloc] initWithItem: @{**NSExtensionJavaScriptFinalizeArgumentKey**: @{@"content":@"Hello World"}} typeIdentifier:(NSString \*)**kUTTypePropertyList**]];

 [[self extensionContext] completeRequestReturningItems:@[extensionItem] expirationHandler:nil completion:nil];

5. 如何在containing app与扩展之间共享代码

iOS 8中，你可以内嵌一个framework文件在扩展和containing app之间共享代码。假设你希望在你的containing app与扩展之间共享图片处理的代码，此时你可以将代码打包成framework文件，内嵌到两个目标中。对于内嵌框架中的代码，确保不包含扩展不允许使用的API。

如何将代码打包成framework文件这里就不敖述了，感兴趣的同学可以参见：<http://blog.sina.com.cn/s/blog_407fb5bc01013v6s.html>。当你创建好.framework文件后，你可以直接将.framework文件同时拖入containing app和扩展中，如下图所示：

[](http://static.oschina.net/uploads/img/201409/24105547_fyQ0.png)

这里使用公司ILSLib目录下的的MagicalRecord21.framework文件作为素材，讲解如何在containing app和自定义键盘扩展之间实现共享Core Data数据库。在你的扩展和containing app中中配置好引用头文件。分别在containing app的AppDelegate文件的application: didFinishLaunchingWithOptions: launchOptions与自定义键盘扩展的UIInputViewController子类文件中viewDidLoad方法中添加如下代码：

[MagicalRecord setupCoreDataStackWithStoreNamed:@"demo.sqlite"];

上述代码分别对containing app和扩展执行Core Data栈初始化，其中包括数据模型、sqlite存储文件等配置。运行containing app，此时AppDelegate中的数据库配置代码会被执行，接着打开系统设置中的通用选项下的键盘选项，在这里启用自定义键盘。然后回到containing app，切换到自定义键盘扩展，此时自定义键盘扩展中viewDidLoad方法中的数据库配置代码执行，但是控制台出现错误提示：

CoreData: error: **-addPersistentStoreWithType:SQLite**configuration:(null) URL:~/Library/Application%20Support/CustomKeyboardExt/demo.sqlite -- file:/// options:(null) ... returned error Error Domain=NSCocoaErrorDomain Code=512 "The operation couldn’t be completed. (Cocoa error 512.)" UserInfo=0x7b48a720 {reason=Failed to create file; code = 2} with userInfo dictionary {

    reason = "**Failed to create file**; code = 2";

}

上述错误表示在扩展的~/Library/Application%20Support/CustomKeyboardExt/demo.sqlite目录创建.sqlite文件失败。翻阅MagicalRecord源代码（需要从github重新下载源代码，.framework看不到源代码），其中在创建.sqlite存储文件路径的代码中会发现：

+ (NSURL \*) MR\_urlForStoreName:(NSString \*)storeFileName {

NSArray \*paths = [NSArray arrayWithObjects:[self MR\_applicationDocumentsDirectory], [self MR\_applicationStorageDirectory], nil];

    NSFileManager \*fm = [[NSFileManager alloc] init];

    for (NSString \*path in paths) {

        NSString \*filepath = [path stringByAppendingPathComponent:storeFileName];

        if ([fm fileExistsAtPath:filepath]) {

            return [NSURL fileURLWithPath:filepath];

        }

    }

    //set default url

    return [NSURL fileURLWithPath:[[self **MR\_applicationStorageDirectory**] stringByAppendingPathComponent:storeFileName]];

}

其中MR\_applicationStorageDirectory方法返回的是Application Support目录，而这个目录是处在Library目录内的。上文中已经讲过，扩展没有Documents目录，同样也是没有Library目录。所以文件创建会发生失败。为了实现扩展与containing app之间共享.sqlite文件，这里需要将.sqlite文件创建在App Group区域。问题是MagicalRecord21.framework文件只暴露了头文件，无法对其源文件中的MR\_urlForStoreName:方法做修改。这里使用Objective-C的动态运行时技术——Method Swizzling，在运行时将MR\_urlForStoreName:方法的实现使用新的实现进行替换。 （**注：这里可以直接给setupCoreDataStackWithStoreNamed方法传递一个包含文件路径的URL类型参数实现修改.sqlite文件的存放位置,methodSwizzling只是另一种通用处理方法**)

首先需要为自定义键盘扩展创建一个Category文件NSPersistentStore+Tracking.h/m，.m文件中的完整的代码如下：

#import "NSPersistentStore+Tracking.h"

#import <objc/runtime.h>

#import <MagicalRecord21/CoreData+MagicalRecord.h>

static NSString \* const kGroupName = @"group.com.aegeaon.ExtensionDemo";

static NSString \* const kContainingDirectory = @"CoreDataStore/";

@implementation NSPersistentStore (Tracking)

+ (void)load {

    static dispatch\_once\_t onceToken;

    dispatch\_once(&onceToken, ^{

        Class class = [self class];

        SEL originalSelector = @selector(MR\_urlForStoreName:);

        SEL swizzledSelector = @selector(ILS\_urlForStoreName:);

        SwizzleClassMethod(class, originalSelector, swizzledSelector);

    });

}

void SwizzleClassMethod(Class c, SEL orig, SEL new) {

    Method origMethod = class\_getClassMethod(c, orig);

    Method newMethod = class\_getClassMethod(c, new);

    c = object\_getClass((id)c);

    if(class\_addMethod(c, orig, method\_getImplementation(newMethod), method\_getTypeEncoding(newMethod)))

        class\_replaceMethod(c, new, method\_getImplementation(origMethod), method\_getTypeEncoding(origMethod));

    else

        method\_exchangeImplementations(origMethod, newMethod);

}

#pragma mark - Method Swizzling

+ (NSURL \*) ILS\_urlForStoreName:(NSString \*)storeFileName {

    NSURL \*storeURL = [[NSFileManager defaultManager] **containerURLForSecurityApplicationGroupIdentifier**:kGroupName];

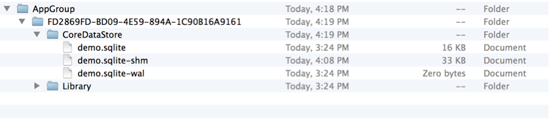
    storeURL = [storeURL URLByAppendingPathComponent:[kContainingDirectory stringByAppendingString:storeFileName]];

    return storeURL;

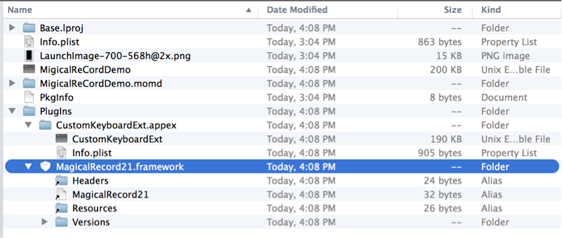
}

@end

在当前的代码一载入内存，load方法将被执行，它比AppDelegate的application: didFinishLaunchingWithOptions: launchOptions方法要先被执行，上述代码会将MR\_urlForStoreName:的实现替换成ILS\_urlForStoreName:，在ILS\_urlForStoreName:方法中，使用NSFileManager的containerURLForSecurityApplicationGroupIdentifier方法设定App Group，最终的.sqlite文件将保存在App Group目录内的CoreDataStore目录下。同样需要为containing app中使用此方法，可以直接将NSPersistentStore+Tracking.h/m拖入containing app目标内。再次运行自定义键盘扩展，数据库文件已成功保存到App Group中。如下图：

[](http://static.oschina.net/uploads/img/201409/24105547_w3mC.png)

同时被共享的代码框架MagicalRecord21.framework被containg app和扩展共享，双方共用一个框架文件，如下图：

[](http://static.oschina.net/uploads/img/201409/24105547_f9sJ.png)

6. 如何在扩展中处理长时间任务

用户希望在扩展完成他们的任务之后能够立即返回到host app中。但是如果扩展执行的任务是一个长时间任务，比如下载。在这种情况下，需要使用NSURLSession来创建一个下载session，并初始化一个后台下载任务。当扩展初始化了上传下载任务后，就算是完成了host app的请求，扩展就可以被终止。这不会影响到任务的结果。如果当后台任务完成后，你的扩展不在运行，系统将在后台启动你的contaiing app并调用appdelegate的aplication:handleEventsForBackgroundURLSession:completionHandler:方法。为了在扩展中初始化一个后台的NSURLSession任务，你必须设置一个containing app和扩展都可以访问的共享容器。

相关代码如下：

NSURLSession \*mySession = [self configureMySession];

NSURL \*url = [NSURL URLWithString:@"http://www.example.com/LargeFile.zip"];

NSURLSessionTask \*myTask = [mySession downloadTaskWithURL:url];

[myTask resume];

- (NSURLSession \*) configureMySession {

    if (!mySession) {

        NSURLSessionConfiguration\* config = [NSURLSessionConfiguration backgroundSessionConfigurationWithIdentifier:@“com.mycompany.myapp.backgroundsession”];

        config.sharedContainerIdentifier = @"com.mycompany.myappgroupidentifier";

        mySession = [NSURLSession sessionWithConfiguration:config delegate:selfdelegateQueue:nil];

    }

    return mySession;

}