



iOS 5 Programming Cookbook

第十七章

iCloud

版本 1.0

翻译时间：2012-09-03

DevDiv 翻译： kyelup cloudhsu 耐心摩卡
wangli2003j3 xiebaochun dymx101
Jimmylianf BeyondVincent

DevDiv 校对： laigb kyelup

DevDiv 编辑： BeyondVincent

写在前面

目前，移动开发被广大的开发者们看好，并大量的加入移动领域的开发。

鉴于以下原因：

- 国内的相关中文资料缺乏
- 许多开发者对 E 文很是感冒
- 电子版的文档利于技术传播和交流

DevDiv.com [移动开发论坛](#) 特此成立了翻译组，翻译组成员具有丰富的移动开发经验和英语翻译水平。组员们利用业余时间，把一些好的相关英文资料翻译成中文，为广大移动开发者尽一点绵薄之力，希望能对读者有些许作用，在此也感谢组员们的辛勤付出。

关于 DevDiv

DevDiv 已成长为国内最具人气的综合性移动开发社区

更多相关信息请访问 [DevDiv 移动开发论坛](#)。

技术支持

首先 DevDiv 翻译组对您能够阅读本文以及关注 DevDiv 表示由衷的感谢。

在您学习和开发过程中，或多或少会遇到一些问题。DevDiv 论坛集结了一流的移动专家，我们很乐意与您一起探讨移动开发。如果您有什么问题和需要技术支持的话，请访问 [DevDiv 移动开发论坛](#) 或者发送邮件到 BeyondVincent@DevDiv.com，我们将尽力所能及的帮助您。

关于本文的翻译

感谢 kyelup、cloudhsu、耐心摩卡、wangli2003j3、xiebaochun、dymx101、jimmylianf 和 BeyondVincent 对本文的翻译，同时非常感谢 laigb 和 kyelup 在百忙中抽出时间对翻译初稿的认真校验，指出了文章中的错误。才使本文与读者尽快见面。由于书稿内容多，我们的知识有限，尽管我们进行了细心的检查，但是还是会存在错误，这里恳请广大读者批评指正，并发送邮件至 BeyondVincent@devdiv.com，在此我们表示衷心的感谢。

读者查看下面的帖子可以持续关注章节翻译更新情况

[\[DevDiv 翻译\]iOS 5 Programming Cookbook 翻译各章节汇总](#)

更多学习资料

我们也为大家准备了 iOS6 新特征的相关内容，请参考下面的帖子：

[iOS6 新特征：参考资料和示例汇总-----持续更新中](#)



目录

写在前面	2
关于 DevDiv	2
技术支持	2
关于本文的翻译	2
更多学习资料	3
目录	4
前言	6
第 1 章 基础入门	7
第 2 章 使用控制器和视图	8
第 3 章 构造和使用 Table View	9
第 4 章 Storyboards	10
第 5 章 并发	11
第 6 章 定位核心与地图	12
第 7 章 实现手势识别的功能	13
第 8 章 网络, JSON, XML 以及 Twitter	14
第 9 章 音频和视频	15
第 10 章 通讯录	16
第 11 章 照相机和图片库	17
第 12 章 多任务	18
第 13 章 Core Data	19
第 14 章 日期, 日程表和事件	20
第 15 章 图形和动画	21
第 16 章 核心运动	22
第 17 章 iCloud	23
17.0. 介绍	23
17.1. 为你的 APP 开启云端同步的功能	23
17.1.1. 问题	23
17.1.2. 方案	23
17.1.3. 讨论	24
17.1.4. 参考	27
17.2. 实现和云端的数据同步	27
17.2.1. 问题	27
17.2.2. 方案	27
17.2.3. 讨论	28
17.2.4. 参考	30
17.3. 实现和云端的目录结构管理	30
17.3.1. 问题	30
17.3.2. 方案	30
17.3.3. 讨论	31
17.3.4. 参考	36

17.4.	查找云端的文件和文件夹	37
17.4.1.	问题	37
17.4.2.	方案	37
17.4.3.	讨论	37
17.5.	同步文档到云端	42
17.5.1.	问题	42
17.5.2.	方案	42
17.5.3.	讨论	42
17.5.4.	参考	52

DevDiv 翻译

前言

参考帖子

[\[DevDiv 翻译\]iOS 5 Programming Cookbook 翻译_前言](#)

DevDiv 翻译

第 1 章 基础入门

参考帖子

[\[DevDiv 翻译\]iOS 5 Programming Cookbook 翻译 第一章 基础入门](#)

DevDiv 翻译

第 2 章 使用控制器和视图

参考帖子

[\[DevDiv 翻译\]iOS 5 Programming Cookbook 翻译 第二章 使用控制器和视图\(上\)](#)

[\[DevDiv 翻译\]iOS 5 Programming Cookbook 翻译 第二章 使用控制器和视图\(下\)](#)

DevDiv 翻译

第 3 章 构造和使用 Table View

参考帖子

[\[DevDiv 翻译\]iOS 5 Programming Cookbook 翻译 第三章 构造和使用 Table View](#)

DevDiv 翻译

第 4 章 Storyboards

参考帖子

[\[DevDiv 翻译\]iOS 5 Programming Cookbook 翻译 第四章 Storyboards](#)

DevDiv 翻译

第 5 章 并发

参考帖子

[\[DEV DIV 翻译\] iOS 5 Programming Cookbook 翻译 第五章 并发](#)

DevDiv 翻译

第 6 章 定位核心与地图

参考帖子

[\[DEV DIV 翻译\] iOS 5 Programming Cookbook 翻译 第六章 核心定位与地图](#)

DevDiv 翻译

第 7 章 实现手势识别的功能

参考帖子

[\[DEVDIV 翻译\] iOS 5 Programming Cookbook 翻译 第七章 实现手势识别](#)

DevDiv 翻译

第 8 章 网络, JSON, XML 以及 Twitter

参考帖子

[\[DEVDIV 翻译\] iOS 5 Programming Cookbook 翻译 第八章 网络, JSON, XML 以及 Twitter](#)

DevDiv 翻译

第 9 章 音频和视频

参考帖子

[\[DEVDIV 翻译\]iOS 5 Programming Cookbook 中文翻译 第九章
音频和视频](#)

DevDiv 翻译

第 10 章 通讯录

参考帖子

[\[DEV DIV 翻译\]iOS 5 Programming Cookbook 中文翻译 第十章
通讯录](#)

DevDiv 翻译

第 11 章 照相机和图片库

参考帖子

[\[DEVDIV 翻译\]iOS 5 Programming Cookbook 中文翻译 第十一章
照相机和图片库](#)

DevDiv 翻译

第 12 章 多任务

参考帖子

[\[DEVDIV 翻译\]iOS 5 Programming Cookbook 中文翻译 第十二章
多任务](#)

DevDiv 翻译

第 13 章 Core Data

参考帖子

[\[DEV DIV 翻译\]iOS 5 Programming Cookbook 中文翻译 第十三章
Core Data](#)

DevDiv 翻译

第 14 章 日期，日程表和事件

参考帖子

[\[DEVDIV 翻译\]iOS 5 Programming Cookbook 中文翻译 第十四章
日期，日程表和事件](#)

DevDiv 翻译

第 15 章 图形和动画

参考帖子

[\[DEV DIV 翻译\]iOS 5 Programming Cookbook 中文翻译 第十五章 图形和动画](#)

DevDiv 翻译

第 16 章 核心运动

参考帖子

[\[DEVDIV 翻译\]iOS 5 Programming Cookbook 中文翻译 第十六章
核心运动](#)

DevDiv 翻译

第 17 章 iCloud

17.0. 介绍

iCloud 是苹果公司做的一个云架构的基础设施，云的概念其实就是为统一化管理存储数据信息服务的。当用户不能直接的来操作他们自己的存储介质的时候，这个时间就可以利用 iCloud 云存储空间来保存和管理自己的数据。目前苹果说有的云数据都是存储在加利福尼亚的云端数据中心。

开发者可以利用云存储个概念为他们的 app 用户提供更好的数据管理程序。从而达到用户可以在不同的终端设备中可以同时管理同一部分数据。现在让我们看一个真实的云端场景使用的例子，比如说你现在开发了一款叫 XYZ 的游戏，一个叫 Sarah 的用户在公司的时候拿 iPhone 开始玩这个游戏，由于你的这个游戏是单机性质类型的，需要升级啊，起名称等数据的保存。但是下班后，Sarah 回家的时候还想玩这个游戏，于是又在 iPad 上下载了一个，发现这个时候又是从 0 级开始的，一切操作都没有了，其实这样的用户体验是很差的，这个时候我们就需要为我们的程序添加云端数据同步的功能，当用户在 iPhone 上玩的时候，用户的数据就会自动的同步到云端中心，当用户在 iPad 上玩的时候，就会把云端中心上的游戏数据同步到 iPad 上来。这样的用户体验就非常的好。

在你准备为你的应用添加云端同步功能的时候，你首先需要为你的 app 开启云端同步的功能。这个就需要你创建正确的认证文件。然后开启云端的功能，你可以通过 17.1 的章节了解到具体是怎么操作的。

17.1. 为你的 APP 开启云端同步的功能

17.1.1. 问题

你准备在你的应用中添加云端同步的功能，你需要了解到你应该做哪些设置相关的操作。

17.1.2. 方案

请参考如下几个步骤进行相关的设置工作。

- 1、用 Xcode 创建一个工程，然后给这个工程设置一个 app 的标识符，例如 com.pixolity.Setting-Up Your-App-For-iCloud（由你的公司名称标识符和产品名字组成）
- 2、登陆苹果官网，进入开发者配置中心，然后为你的应用创建一个 APP ID
- 3、然后再创建一个配置文件，确保这个 Provisioning 配置文件跟你最新的这个 APP ID 是关联在一起的。
- 4、在 Xcode 开发工具之中，选择你的 Target 条目，然后选择 Summary 类别，最后往下拖动，一直到 Entitlements 子项中，然后勾选 enable entitlements 选项。这个操作将会为你的应用做云端配置的初始化动作。

17.1.3. 讨论

为了能够在你的应用中添加云端同步的功能，那么你就需要做一系列的配置工作。下面请从新看一下我们仍然需要做哪些操作。

一.打开 Xcode，选择 New Project.....

二.在左边的菜单栏中，请一次选择 iOS 目录，然后选择 Application 子条目，然后再右边的视图中选择 Empty Application ,点击 Next.如图 17-1 所示。

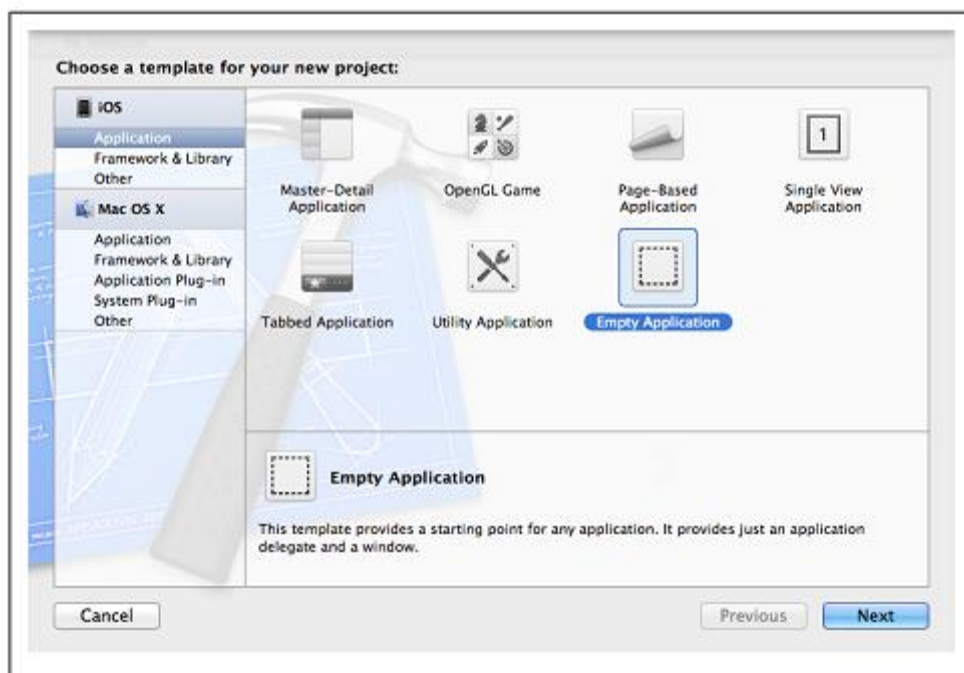


Figure 17-1. Creating an Empty Application to use with iCloud storage

三.如图 17-2 所示，我们创建一个工程，起一个 Product Name，然后再添加一个 Company 标识符。

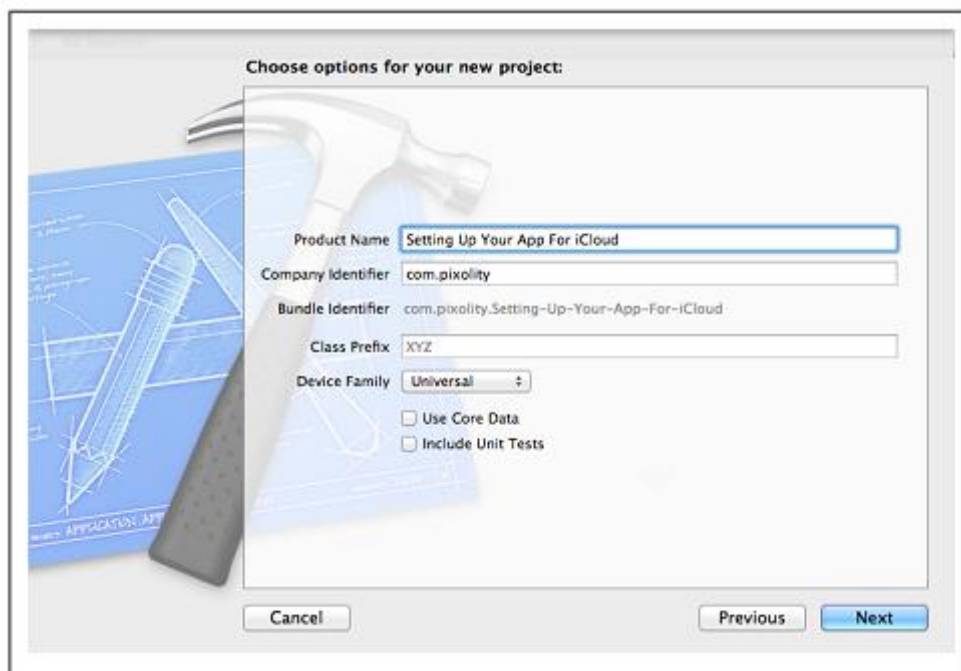


Figure 17-2. Setting the product name and company identifier for a new iCloud app

四.然后你需要选择一个目录来保存你的工程文件，这个看你自己的意愿了。

五.然后我们需要根据我们创建的标识符，来创建我们相对应的 APP ID 配置文件。

六.如图 17-3 所示，这个界面是我们创建 APPID 的界面，具体配置项请参考如下图来填写。

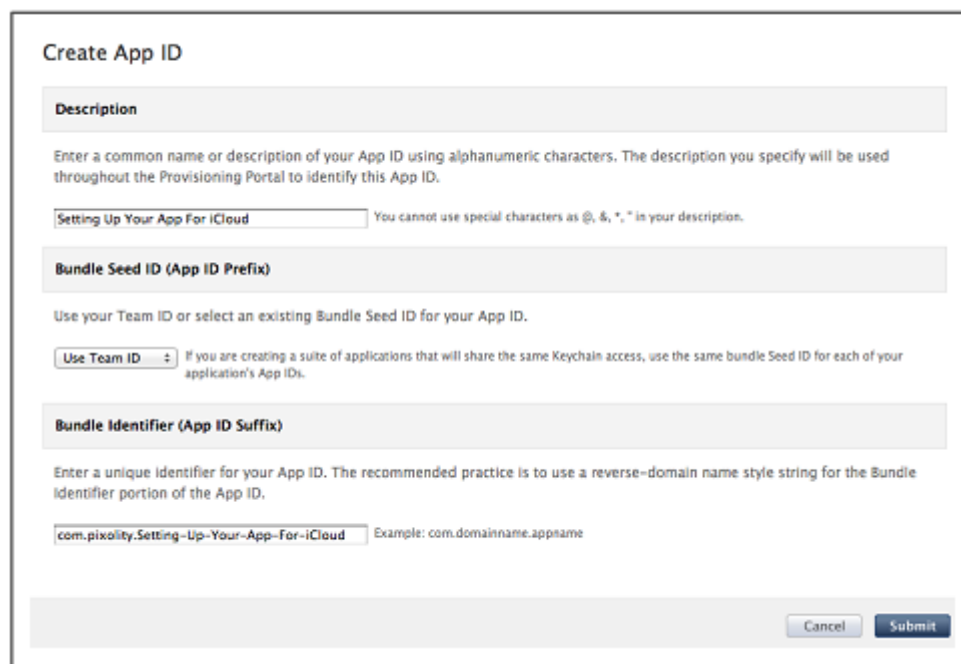


Figure 17-3. Setting up a new App ID for our iCloud app

七.然后你需要创建一个产品配置文件和你的 AppID 进行一个关联。如图 17-4 所示

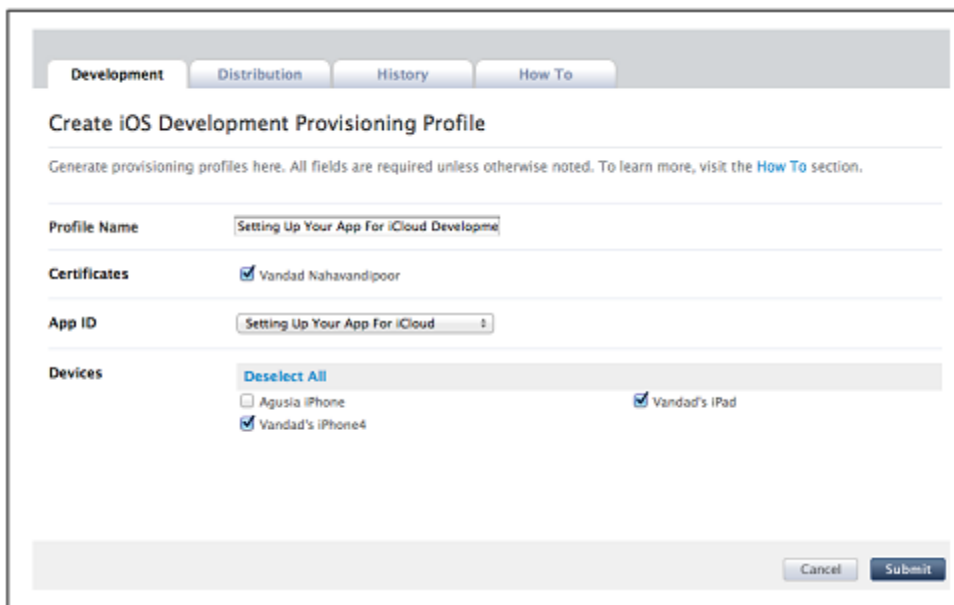


Figure 17-4. Creating a new Development provision profile for iCloud

八. 创建好配置文件了，关联 APPID 之后，我们需要切换到 Distribution 选项卡中，然后再做如下配置。

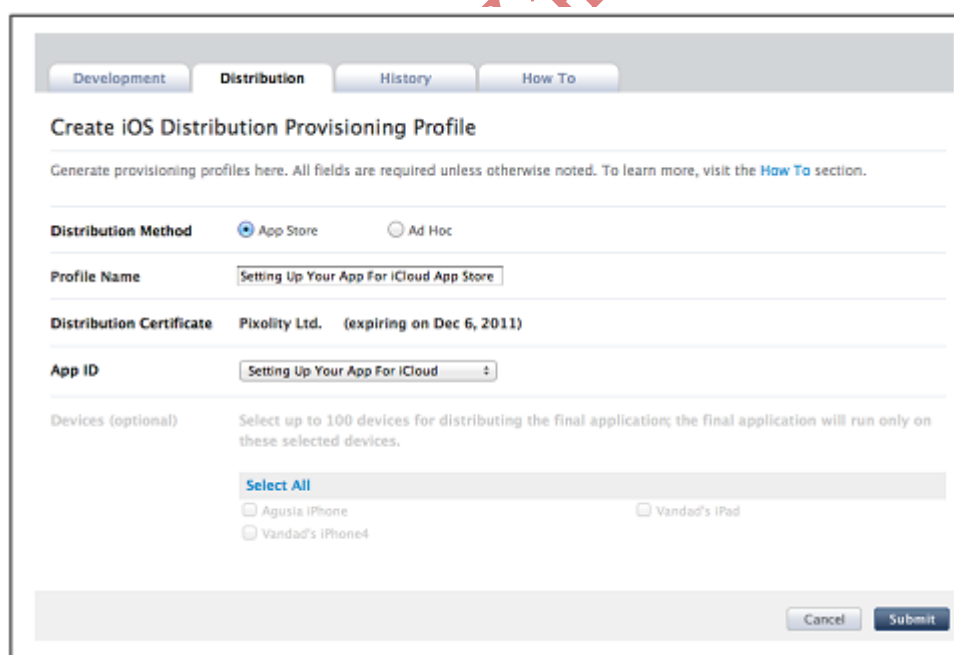


Figure 17-5. Creating a new Distribution provision profile for iCloud

九. 然后创建完成只有刷新界面，在从新进入到 Provisioning 选项卡中，下载刚才创建的 Provisioning File 和 Distribution File 这两个文件。

十. 然后把这两个文件拖动到 itunes 里面，itunes 会自动安装这两个文件的。

十一. 然后进入到 xcode 里面中，选择你的工程，然后会看到类似如下界面。

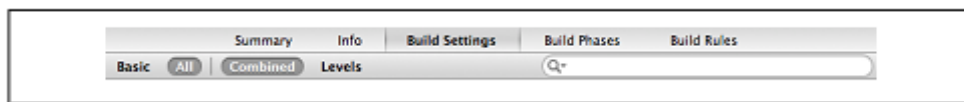


Figure 17-6. The Combined and All bar items should be selected in the Build Settings tab

然后一直往下拖动，知道看到 **Code Signing** 这个条目。在这个条目中，你就需要为你的应用程序添加发布时的配置文件了。如图 17-7 所示。



Figure 17-7. Setting the appropriate provisioning profiles for Debug and Release schemes

Ok，当一起操作完成之后，别忘记最后一步，就是需要添加 iCloud 的基础配置文件。

请切换到 **Summary** 选项卡里面，然后再往下滑动，看到 **Enable Entitlements** 的时候，然后把这个选项卡选住，那么 xcode 就会自动的帮你生成基础的配置文件。

很简单吧。下面几个章节让我们来看一下具体的应用。

17.1.4. 参考

暂无

17.2. 实现和云端的数据同步

17.2.1. 问题

如果你想存储一个 **Dictionary** 类型的表单数据到云端，然后再能通过一个关联的账号实现数据的同步。

17.2.2. 方案

你需要使用到 **NSUbiquitousKeyValueStore** 这个类。



你不必担心你放在云端中心的数据会产生混乱，这个完全是有云端中心来控制的。

17.2.3. 讨论

其实 `NSUbiquitousKeyValueStore` 的用法就跟 `NSUserDefaults` 的用法非常相似。它能用来存储各种类型, `string`, `boolean`, `integer` 等等。其中每一个值都会有一个相对应的 `Key` 来和他保持对应。你可以通过这个 `Key` 值来进行读取的操作。`NSUbiquitousKeyValueStore` 和 `NSUserDefaults` 的区别就是前者保存的数据是保存在云端中心的, 而后者却是把数据保存在本地, 也就是 `a.plist` 文件中的。



在你能够正常使用 `NSUbiquitousKeyValueStore` 这个类来同步数据到云端中心的时候, 你必须要做正常的设置, 请参考 17.1

每一个应用程序在同步数据到云端中心的时候都会使用一个唯一的标识符来作为区别。通常这个唯一的标识符是有三部分组成。

Team ID

这个其实就是你的 iOS 开发者账号作为一个标识符, 当你在注册开发者账号的时候, 苹果公司会自动的给你生成一个唯一的标识符。为了获取到这个标识符, 你需要登录的开发者管理中心, 然后选择你的账户, 然后选择组织信息。最后你就能够看到你的一个 `Team ID`。所有的 `Team ID`

都不可能时一样的, 所以你不必担心这个问题。

Reverse domain-style of company identifier

这个标识符通常是由 `com.COMPANYNAME.APPNAME` 组成。其中 `COMPANYNAME` 就是你们公司的名字, `APPNAME` 就是你的这个应用的名字。例如我的公司名称是 `Pixolity`, 我的应用的名字是 `olity`。那么这个标识符就应该是 `com.piolity.olity`

App identifier and optional suffix

这个标识符通常是作为 `Reverse domain-style of company identifier` 标识符的一个补充的后缀。例如 `Storing and Synchronizing Dictionaries in iCloud` 是我创建的一个云端中心的数据名称, 由于 `Xcode` 中是不允许有空格的, 所以我们经常设置为 `Storing-and-Synchronizing-Dictionaries-in-iCloud`

现在我们假设我们的一起准备工作已经设置完毕。现在我们就来开始学习一下如果通过 `NSUbiquitousKeyValueStore` 来进行和云端中心的数据同步。`NSUbiquitousKeyValueStore` 这个类是有很多个方法的, 下面我们需要做一个详细的解释。

1. `setString:forKey:`

给一个指定的 `Key` 添加一条关联的值。`Key` 的类型必须是要为 `NSString` 类型的。很明显如果是 `NSString` 的子类的, 例如 `NSMutableString` 也是可以存储进来的。

2. `setArray:forKey:`

给一个指定的 `Key` 和一个数组绑定在一起。

3. `setBool:forKey:`

把一个 `Key` 和 `boolean` 的变量绑定在一起

4. `setData:forKey:`

把一个 `Key` 和一个 `Data` 类型的数据绑定在一起。

如上的方法只是做了一个临时的保存, 如果你需要同步到云端中心, 那么你就需要使用

到 `NSUbiquitousKeyValueStore` 这个类的 `synchronize` 这个方法。



我们需要创建一个 `NSUbiquitouskeyValueStore` 对象实例。仅仅需要使用 `[NSUbiquitousKeyValueStore defaultStore]`这个方法就可以了。

有一点我们需要注意的就是，当我们的应用第一次和云端同步的时候，我们要先检查我们的云端中是否保存这样的数据，如果保存了然后我们应该做一个更新。如果没有保存我们再来创建一个新的云端同步数据的储存对象，这样就能防止，当我们拿 **iPhone** 开了书，然后记录的页数，并且保存到了云端上，然后我们在拿 **iPad** 看这本书，然后记录页数，又创建一个到云端中心上。

具体的操作代码如下。

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
    NSUbiquitousKeyValueStore *kvoStore =
    [NSUbiquitousKeyValueStore defaultStore];

    NSString *stringValue = @"My String";
    NSString *stringValueKey = @"MyStringKey";

    BOOL boolValue = YES;
    NSString *boolValueKey = @"MyBoolKey";

    BOOL mustSynchronize = NO;

    if ([[kvoStore stringForKey:stringValueKey] length] == 0){
        NSLog(@"Could not find the string value in iCloud. Setting..");
        [kvoStore setString:stringValue
                    forKey:stringValueKey];
        mustSynchronize = YES;
    } else {
        NSLog(@"Found the string in iCloud, getting...");
        stringValue = [kvoStore stringForKey:stringValueKey];
    }

    if ([kvoStore boolForKey:boolValueKey] == NO){
        NSLog(@"Could not find the boolean value in iCloud. Setting..");
        [kvoStore setBool:boolValue
                    forKey:boolValueKey];
        mustSynchronize = YES;
    } else {
        NSLog(@"Found the boolean in iCloud, getting...");
        boolValue = [kvoStore boolForKey:boolValueKey];
    }

    if (mustSynchronize){
        if ([kvoStore synchronize]){
            NSLog(@"Successfully synchronized with iCloud.");
        } else {
            NSLog(@"Failed to synchronize with iCloud.");
        }
    }
}
```

```
self.window = [[UIWindow alloc] initWithFrame:
                [[UIScreen mainScreen] bounds]];

self.window.backgroundColor = [UIColor whiteColor];
[self.window makeKeyAndVisible];
return YES;
}
```

在所有的代码和设置完成之后，编译运行我们的工程。
我们将会看到如下的结果。

```
Could not find the string value in iCloud. Setting...
Could not find the boolean value in iCloud. Setting...
Successfully synchronized with iCloud.
```

现在我们就再运行这个程序，我们将会看到如下的结果。

```
Found the string in iCloud, getting...
Found the boolean in iCloud, getting...
```

第一次是新建，而第二次只是做了一个更新，并没有新建。
这样就能把数据同步到云端中心，然后提供各个终端设备对同一应用的数据进行共享。

17.2.4. 参考

暂无

17.3. 实现和云端的目录结构管理

17.3.1. 问题

你想在云端中创建一个专门的文件夹来保存你的指定类型的数据。

17.3.2. 方案

你需要遵循如下几个步骤

1. 确保你已经按照 17.1 的步骤进行了相关的设置。
2. 打开你的工程，然后进入到 **Target** 条目，然后再选择 **Summary** 选项卡。
3. 在 **Summary** 选项卡中，然后往下滑动，进入到 **Entitlements** 类别中，然后找到 **iCloud Container** 列表框。然后添加一个标识符的信息。列入我这边设置的是 `com.pixolity.Creating-and-Managing-Folders-for-Apps-in-iCloud`。这个标识符信息跟你的工程

标识符信息由一点类似。

4. 然后拷贝 iCloud Container 列表中刚才添加的信息，然后作为一个后缀添加在你的 Team ID 后面。

5. 然后我们利用 `NSFileManager` 和 `URLForUbiquityContainerIdentifier`：这个方法进行目录的确认和初始化得工作。

17.3.3. 讨论

有一点我们需要阐述的就是，我们在操作 iCloud 上的目录的文件的时候，并不是直接的来进行一个操作的动作，你只是在本地做了一些操作，然后由云端自己去对云端中心的数据或者目录进行一个同步更新的操作。下面让我们来查看一下代码的具体实现过程。

```
- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    NSFileManager *fileManager = [[NSFileManager alloc] init];

    /* Place your team ID here */
    NSString *teamID = @"TEAM ID";

    NSString *rootFolderIdentifier = [NSString stringWithFormat:
        @"%@.com.pixolity.Creating-and-Managing-Folders-for-Apps-in-iCloud",
        teamID];

    NSURL *containerURL =
        [fileManager URLForUbiquityContainerIdentifier:rootFolderIdentifier];

    NSString *documentsDirectory = [[containerURL path]
        stringByAppendingPathComponent:@"Documents"];

    BOOL isDirectory = NO;
    BOOL mustCreateDocumentsDirectory = NO;

    if ([fileManager fileExistsAtPath:documentsDirectory
        isDirectory:&isDirectory]){
        if (isDirectory == NO){
            mustCreateDocumentsDirectory = YES;
        }
    } else {
        mustCreateDocumentsDirectory = YES;
    }

    if (mustCreateDocumentsDirectory){
        NSLog(@"Must create the directory.");
    }

    NSError *directoryCreationError = nil;

    if ([fileManager createDirectoryAtPath:documentsDirectory
        withIntermediateDirectories:YES
        attributes:nil
        error:&directoryCreationError]){
        NSLog(@"Successfully created the folder.");
    } else {
        NSLog(@"Failed to create the folder with error = %@",
            directoryCreationError);
    }
}
```

```
} else {
    NSLog(@"This folder already exists.");
}

self.window = [[UIWindow alloc] initWithFrame:
               [[UIScreen mainScreen] bounds]];

self.window.backgroundColor = [UIColor whiteColor];
[self.window makeKeyAndVisible];
return YES;
}
```



需要注意的是，我们的这个标识符，也就是 **Container Identifier** 其实可以任意来填写的，这个根据你自己的需要。并没有说一个固定的形式，或者规则。我们这里这么写只是为了更容易理解。

下面我们所需要做的就是对这段代码进行一个重构。

```
- (BOOL) create iCloudDirectory:(NSString *)paramDirectory
    recursiveCreation:(BOOL)paramRecursiveCreation
    teamID:(NSString *)paramTeamID
    iCloudContainer:(NSString *)paramContainer
    finalPath:(NSString **)paramFinalPath{

    BOOL result = NO;

    NSFileManager *fileManager = [[NSFileManager alloc] init];

    NSString *rootFolderIdentifier = [NSString stringWithFormat:
                                     @"%@.%@", paramTeamID, paramContainer];

    NSURL *containerURL =
        [fileManager URLForUbiquityContainerIdentifier:rootFolderIdentifier];

    NSString *documentsDirectory = [[containerURL path]
                                    stringByAppendingPathComponent:@"Documents"];

    if (paramFinalPath != nil){
        *paramFinalPath = documentsDirectory;
    }

    BOOL isDirectory = NO;
    BOOL mustCreateDocumentsDirectory = NO;

    if ([fileManager fileExistsAtPath:documentsDirectory
        isDirectory:&isDirectory]){
        if (isDirectory == NO){
            mustCreateDocumentsDirectory = YES;
        }
    } else {
        mustCreateDocumentsDirectory = YES;
    }

    if (mustCreateDocumentsDirectory){
        NSLog(@"Must create the directory.");
    }
}
```



```

NSError *directoryCreationError = nil;

if ([fileManager createDirectoryAtPath:documentsDirectory
    withIntermediateDirectories:paramRecursiveCreation
    attributes:nil
    error:&directoryCreationError]){
    result = YES;
    NSLog(@"Successfully created the folder.");
} else {
    NSLog(@"Failed to create the folder with error = %@",
        directoryCreationError);
}

} else {
    NSLog(@"This folder already exists.");
    result = YES;
}

return result;
}

- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

    /* Place your Team ID here */
    NSString *teamID = @"TEAM ID";

    NSString *containerID =
    @"com.pixolity.Creating-and-Managing-Folders-for-Apps-in-iCloud";

    NSString *documentsDirectory = nil;

    if ([self createiCloudDirectory:@"Documents"
        recursiveCreation:YES
        teamID:teamID
        iCloudContainer:containerID
        finalPath:&documentsDirectory]){
        NSLog(@"Successfully created the directory in %@", documentsDirectory);
    } else {
        NSLog(@"Failed to create the directory.");
    }

    self.window = [[UIWindow alloc] initWithFrame:
        [[UIScreen mainScreen] bounds]];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}

```

Ok，现在我想大家都能够很好理解了，现在我们在做一下重构，往目录里面添加一个文档。

```

- (BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{

```

```
/* Place your Team ID here */
NSString *teamID = @"TEAM ID";

NSString *containerID =
    @"com.pixolity.Creating-and-Managing-Folders-for-Apps-in-iCloud";

NSString *documentsDirectory = nil;

if ([self createiCloudDirectory:@"Documents"
    recursiveCreation:YES
    teamID:teamID
    iCloudContainer:containerID
    finalPath:&documentsDirectory]){
    NSLog(@"Successfully created the directory in %@", documentsDirectory);

    NSString *stringToSave = @"My String";

    NSString *pathToSave = [documentsDirectory
        stringByAppendingPathComponent:@"MyString.txt"];

    NSError *savingError = nil;

    if ([stringToSave writeToFile:pathToSave
        atomically:YES
        encoding:NSUTF8StringEncoding
        error:&savingError]){
        NSLog(@"Successfully saved the string in iCloud.");
    } else {
        NSLog(@"Failed to save the string with error = %@", savingError);
    }

} else {
    NSLog(@"Failed to create the directory.");
}

self.window = [[UIWindow alloc] initWithFrame:
    [[UIScreen mainScreen] bounds]];

self.window.backgroundColor = [UIColor whiteColor];
[self.window makeKeyAndVisible];
return YES;
}
```

点击运行我们的应用程序，然后再进入到 **setting** 中，进入到 **iCloud** 的选项中，你将会看到 17-9 图显示的效果。（注意我们这里的保存并不是一个真正的保存到了云端上，并没有同步到云端上，具体如何同步到云端上，请参考我们的 17.5 小节）



图 17-9 我们程序的文档数据在 iPad 的设置程序里罗列出来

现在我们在点击 UNKNOWN 选项，然后将会看到 17-10 的效果。



图 17-10 我们在磁盘上存储的字符串的确同步于 iCloud

17.3.4. 参考 暂无

17.4. 查找云端的文件和文件夹

17.4.1. 问题

你想查找和你当前应用相关的云端的数据或者是目录结构。

17.4.2. 方案

你需要学会使用 `NSMetadataQuery` 这个类。

17.4.3. 讨论

OS X 的开发人员可能对 `NSMetadataQuery` 这个类比较熟悉，这个类允许开发者能够查询一些文件或者是目录结构。在 iOS 中，我们同样提供了这样一个类让用户来和云端中心的数据和目录结构进行一个交互。

下面让我们看一下具体的代码该如何操作。

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController
@property (nonatomic, strong) NSMetadataQuery *metadataQuery;
@end

#import "ViewController.h"
@implementation ViewController
@synthesize metadataQuery;
...
```

首先当我们的视图界面在加载的时候，我们就应该启动一个搜索对象，进行云端中心的目录和文件的搜索。

```
- (void)viewDidLoad{

    [super viewDidLoad];

    /* Listen for a notification that gets fired when the metadata query
    has finished finding the items we were looking for */
    [[NSNotificationCenter defaultCenter]
    addObserver:self
    selector:@selector(handleMetadataQueryFinished:)
    name:NSMetadataQueryDidFinishGatheringNotification

    object:nil];

    // Do any additional setup after loading the view, typically from a nib.
    self.metadataQuery = [[NSMetadataQuery alloc] init];
    NSArray *searchScopes = [[NSArray alloc] initWithObjects:
        NSMetadataQueryUbiquitousDocumentsScope, nil];
    [self.metadataQuery setSearchScopes:searchScopes];
    NSPredicate *predicate = [NSPredicate predicateWithFormat:
        @"%K like %@",
```

```

        NSMetadataItemFSNameKey,
        @"*"];
[self.metadataQuery setPredicate:predicate];
if ([self.metadataQuery startQuery]){
    NSLog(@"Successfully started the query.");
} else {
    NSLog(@"Failed to start the query.");
}
}

- (void)viewDidUnload{
    [super viewDidUnload];
    // Release any retained subviews of the main view.
    [[NSNotificationCenter defaultCenter] removeObserver:self];
    [self setMetadataQuery:nil];
}

- (NSURL *) urlForDocumentsFolderInCloud{

    NSURL *result = nil;

    #error Put your TEAM ID here
    const NSString *TeamID = @"YOUR TEAM ID";

    NSString *containerID = [[NSBundle mainBundle] bundleIdentifier];

    NSString *teamIDAndContainerID = [[NSString alloc] initWithFormat:@"% @.% @",
                                     TeamID,
                                     containerID];

    NSFileManager *fileManager = [[NSFileManager alloc] init];

    NSURL *appiCloudContainerURL =
    [fileManager URLForUbiquityContainerIdentifier:teamIDAndContainerID];

    result = [appiCloudContainerURL URLByAppendingPathComponent:@"Documents"
                                     isDirectory:YES];

    if ([fileManager fileExistsAtPath:[result path]] == NO){

        /* The Documents directory does NOT exist in our app's iCloud container;
        attempt to create it now */

        NSError *creationError = nil;
        BOOL created = [fileManager createDirectoryAtURL:result
                                     withIntermediateDirectories:YES
                                     attributes:nil
                                     error:&creationError];

        if (created){
            NSLog(@"Successfully created the Documents folder in iCloud.");
        } else {
            NSLog(@"Failed to create the Documents folder in iCloud. Error = % @",
                  creationError);
        }
    }
}

```

```
        result = nil;
    }

} else {
    /* the Documents directory already exists in our app's iCloud container;
       we don't have to do anything */
}

return result;
}

- (NSURL *) urlForRandomFileInDocumentsFolderInIcloud{

    NSURL *result = nil;

    NSInteger randomNumber = arc4random() % NSUIntegerMax;

    NSString *randomFileName = [[NSString alloc] initWithFormat:@"%llu.txt",
                               (unsigned long)randomNumber];

    /* Check in the metadata query if this file already exists */
    __block BOOL fileExistsAlready = NO;
    [self.metadataQuery.results enumerateObjectsUsingBlock:
     ^(NSMetadataItem *item, NSUInteger idx, BOOL *stop) {
        NSString *itemFileName = [item valueForKey:NSMetadataItemFSNameKey];
        if ([itemFileName isEqualToString:randomFileName]){
            NSLog(@"This file already exists. Aborting...");
            fileExistsAlready = YES;
            *stop = YES;
        }
    }];

    if (fileExistsAlready){
        return nil;
    }

    result = [[self urlForDocumentsFolderInIcloud]
              URLByAppendingPathComponent:randomFileName];

    return result;
}

- (NSURL *) urlForRandomFileInDocumentsFolderInAppSandbox
    :(NSString *)paramFileName{

    NSURL *result = nil;

    NSString *documentsFolderInAppSandbox =
    [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                         NSUserDomainMask,
                                         YES) objectAtIndex:0];

    NSString *filePath = [documentsFolderInAppSandbox
```

```
        stringByAppendingPathComponent:paramFileName];

result = [NSURL fileURLWithPath:filePath];

return result;
}

- (void) enumerateMetadataResults:(NSArray *)paramResults{

    [paramResults enumerateObjectsUsingBlock:
    ^(NSMetadataItem *item, NSUInteger index, BOOL *stop) {

        NSString *itemName = [item valueForKey:NSMetadataItemFSNameKey];
        NSURL *itemURL = [item valueForKey:NSMetadataItemURLKey];
        NSNumber *itemSize = [item valueForKey:NSMetadataItemFSSizeKey];

        NSLog(@"Item name = %@", itemName);
        NSLog(@"Item URL = %@", itemURL);
        NSLog(@"Item Size = %llu",
            (unsigned long long)[itemSize unsignedLongLongValue]);

    }];
}
```

在最后我们需要实现 `handleMetadataQueryFinished`，这个方法，这个方法是当有请求数据返回后调用的方法。

```
- (void) handleMetadataQueryFinished:(id)paramSender{

    NSLog(@"Search finished");

    if ([paramSender object] isEqual:self.metadataQuery) == NO){
        NSLog(@"An unknown object called this method. Not safe to proceed.");
        return;
    }

    /* Stop listening for notifications as we are not expecting anything more */
    [[NSNotificationCenter defaultCenter] removeObserver:self];

    /* We are done with the query, let's stop the process now */
    [self.metadataQuery disableUpdates];
    [self.metadataQuery stopQuery];

    [self enumerateMetadataResults:self.metadataQuery.results];

    if ([self.metadataQuery.results count] == 0){
        NSLog(@"No files were found.");
    }

    NSURL *urlForFileIniCloud = [self urlForRandomFileInDocumentsFolderIniCloud];

    if (urlForFileIniCloud == nil){
        NSLog(@"Cannot create a file with this URL. URL is empty.");
        return;
    }
}
```



```

NSString *fileName = [[[urlForFileInCloud path]
                        componentsSeparatedByString:@"[/"] lastObject];

NSURL *urlForFileInAppSandbox =
[self urlForRandomFileInDocumentsFolderInAppSandbox:fileName];

NSString *fileContent =
[[NSString alloc] initWithFormat:@"Content of %@",
 [[self urlForRandomFileInDocumentsFolderInCloud] path]];

/* Save the file temporarily in the app bundle and then move
it to the cloud */
NSError *writingError = nil;
BOOL couldWriteToAppSandbox =
[fileContent writeToURL:urlForFileInAppSandbox path]
    atomically:YES
    encoding:NSUTF8StringEncoding
    error:&writingError];

/* If cannot save the file, just return from method because it won't make
any sense to continue as we, ideally, should have stored the file in iCloud
from the app sandbox but here, if an error has occurred,
we cannot continue */
if (couldWriteToAppSandbox == NO){
    NSLog(@"Failed to save the file to app sandbox. Error = %@", writingError);
    return;
}

NSFileManager *fileManager = [[NSFileManager alloc] init];

/* Now move the file to the cloud */
NSError *ubiquitousError = nil;
BOOL setUbiquitousSucceeded =
[fileManager setUbiquitous:YES
    itemAtURL:urlForFileInAppSandbox
    destinationURL:urlForFileInCloud
    error:&ubiquitousError];

if (setUbiquitousSucceeded){
    NSLog(@"Successfully moved the file to iCloud.");
    /* The file has been moved from App Sandbox to iCloud */
} else {
    NSLog(@"Failed to move the file to iCloud with error = %@",
        ubiquitousError);
}
}

```

当如上代码添加完成之后，我们编译运行我们的工程，你将会看到类似 17-11 图所示的内容。



我们需要在查询之前先创建一些文件。然后我们需要在 ViewDidLoad 方法开始调用的时候开始我们的查询动作。因此，如果你我们在打开我们应用的时候，我们有点击 Home 件进入到主界面，那么这个

时间的创建工作就被打断了。为了确保我们在每次进入的时候都能创建一些新的文件，我们应该每次手动的切掉这个进程（由于上次的错误操作点击了 HOME），然后再重新进入我们的应用。让这个创建的过程在重新执行一遍。

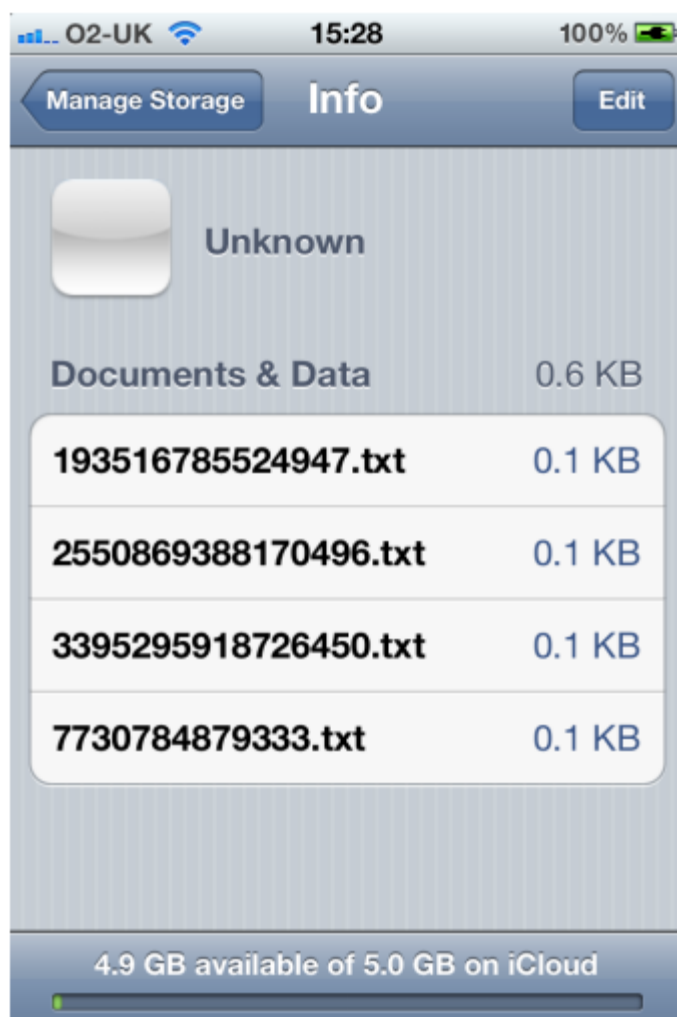


图 17-11 我们程序在 iCloud 创建的随机文件列表

17.5. 同步文档到云端

17.5.1. 问题

你想在你的应用中提供一用户往云端创建文档数据，然后再不同的设备之间进行同步的功能。

17.5.2. 方案

采用 `UIDocument` 这个类进行处理。

17.5.3. 讨论

由于用户在设备上安装了很多个软件，在操作软件的时候也会产生大量的数据。因此每一个 app 都应该做一定的处理，防止任何数据都往云端中心上保存。举个例子说，就是用户在用 safari 浏览器进行网页浏览的时候，浏览器会自动的给我们做一些数据的缓存然后保存在本地的沙盒中，这样可以为用户提供更好的体验以便下次访问能更快的加载。其实这个数据并不是用户自己产生的，而是 app 自己产生的，这部分的数据我们一般的不建议保存到云端中心上去，即使要保存也要提醒用户，哪些数据会保存哪些不会保存。由于每个用户的云端中心的存储空间是有限的，所以我们要存储一些有利的数据，而把一些 app 自己产生的数据保存在本地的沙盒环境中就足够了。

学习 ios 开发这么久，我们一直在我们的应用中所作的数据的存储都是保存在本地的也就是 itongchang 我们所说的应用的沙箱。那么我们现在需要学习一下第二种操作方式，那就是把数据都保存在服务器中心中。我们叫做云端中心。

现在让我们看一下具体该怎么处理，假设我们已经做好了基础的设置，（17.1）小结的设置。

现在我们需要通过我们的应用往云端中心中创建一个 Documents folder。并保存一个 UserDocument.txt 的文件在这个目录中。

1.打开 xcode，然后选择 File---->New----->New File .

2.如图 17-12 所示，然后选择 iOS 目录下的 Cocoa Touch 子项。然后再右边选择.Objective-C 类别。

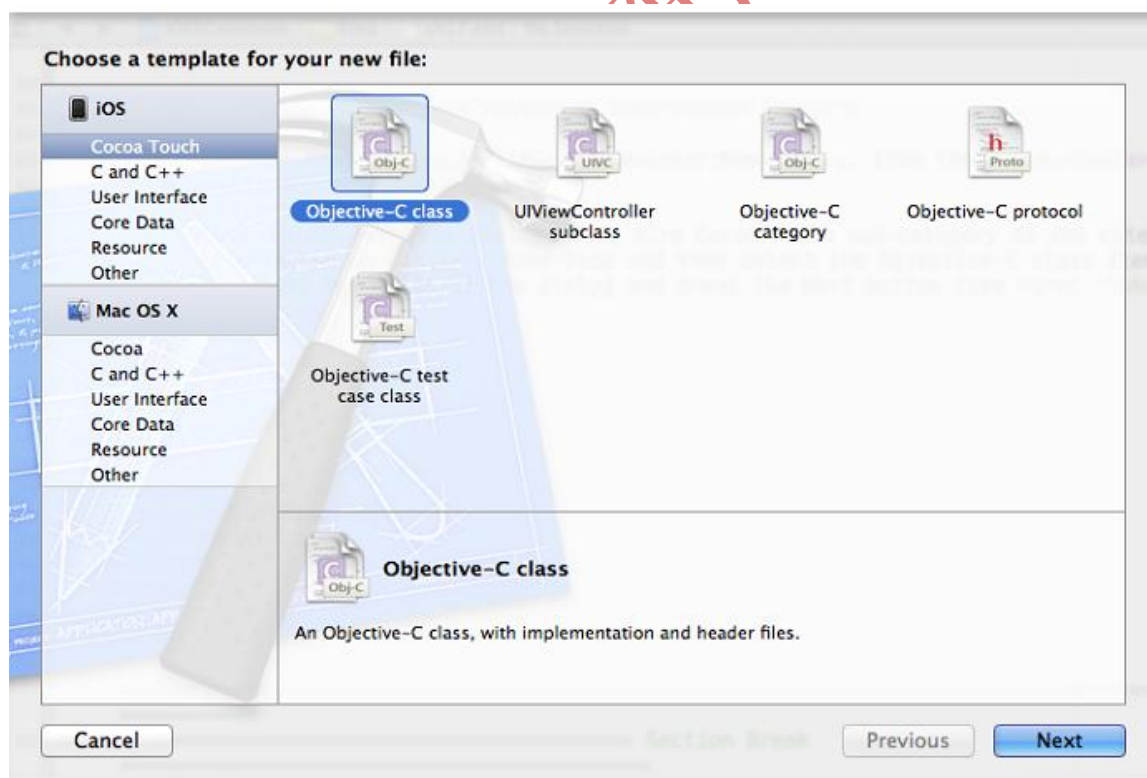


Figure 17-12. Beginning to create a new document class

3.然后如图 17-13 所示，设置如下内容。

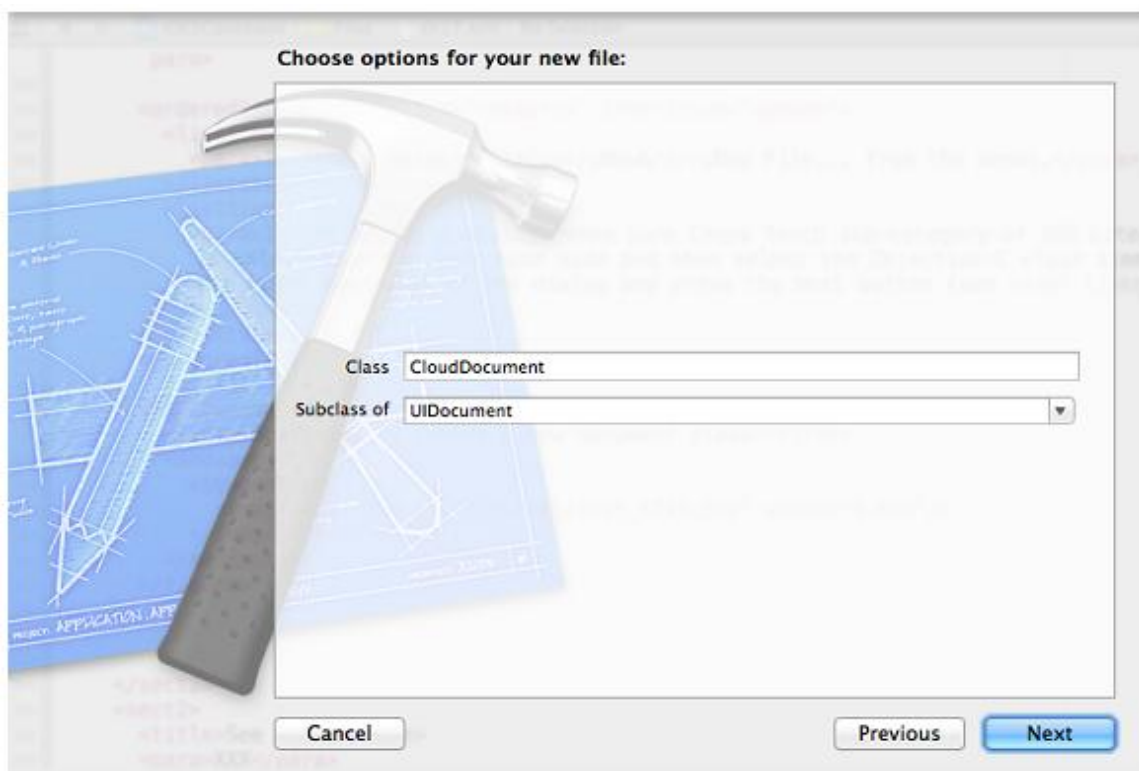


Figure 17-13. Subclassing UIDocument

- 4.如图 17-14 所示，然后选择一个目录保存。
- 5.

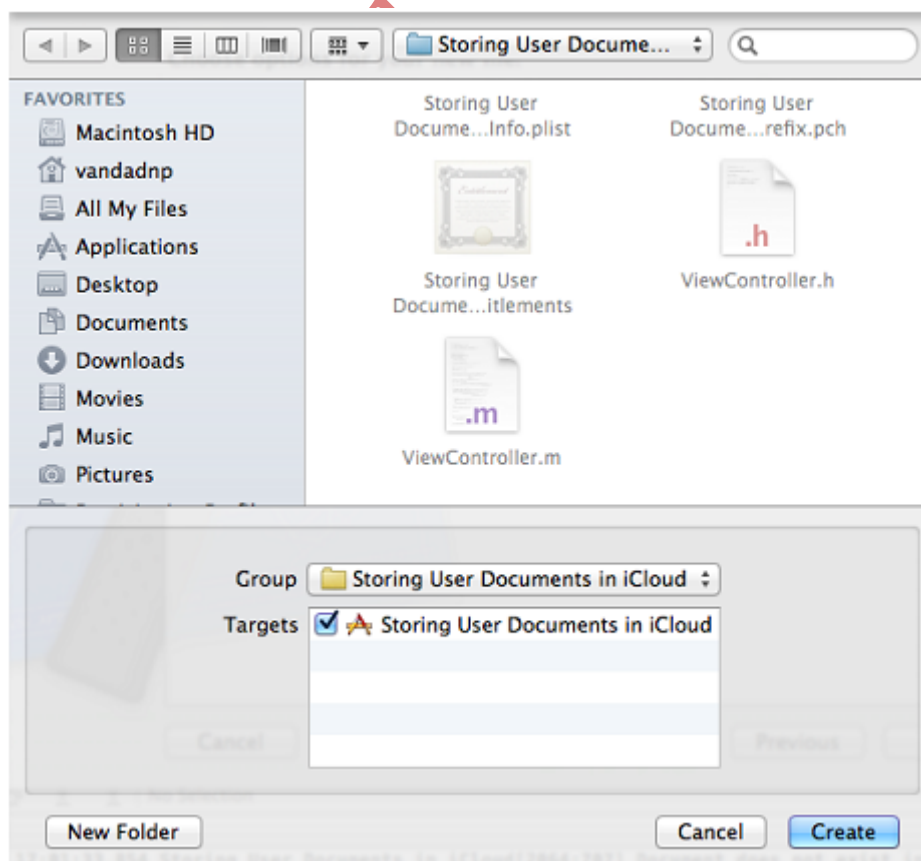


Figure 17-14. Saving the subclass to disk

所有东西都准备好了，下面让我们看一下具体的代码该如何实现。

```
#import <UIKit/UIKit.h>
@class CloudDocument;
@protocol CloudDocumentProtocol<NSObject>
- (void) cloudDocumentChanged:(CloudDocument *)paramSender;
@end
@interface CloudDocument : UIDocument
@property (nonatomic, strong) NSString *documentText;
@property (nonatomic, weak) id<CloudDocumentProtocol> delegate;
/* Designated Initializer */
- (id) initWithFileURL:(NSURL *)paramURL
    delegate:(id<CloudDocumentProtocol>)paramDelegate;
@end
```

如上我们分别实现了一个协议，让我们来看看具体作用。

CloudDocumentProtocol

通过这个协议我们可以实现存储数据的同步于更新，以及云端中心和设备之间的一个认证。

```
#import "CloudDocument.h"
@implementation CloudDocument
@synthesize documentText;
@synthesize delegate;
...
```

我们将需要实现协议的方法，并进行相关值的初始化动作。

```
- (id) initWithFileURL:(NSURL *)paramURL
    delegate:(id<CloudDocumentProtocol>)paramDelegate {

    self = [super initWithFileURL:paramURL];

    if (self != nil) {

        if (paramDelegate == nil) {
            NSLog(@"Warning: no delegate is given.");
        }

        delegate = paramDelegate;
    }

    return self;
}
- (id) initWithFileURL:(NSURL *)paramURL {
    return [self initWithFileURL:paramURL
        delegate:nil];
}
```

就像前几节提到的，我们需要实现 `contentType:error:` 这个方法。

```
- (id) contentType:(NSString *)typeName
    error:(NSError *__autoreleasing *)outError{

    if ([self.documentText length] == 0){
        self.documentText = @"New Document";
    }

    return [self.documentText dataUsingEncoding:NSUTF8StringEncoding];
}
```

下面我们关注一下 `loadFromContents ofType:error:` 这个方法的具体操作过程。

```
- (BOOL) loadFromContents:(id)contents
    ofType:(NSString *)typeName
    error:(NSError *__autoreleasing *)outError{

    NSData *data = (NSData *)contents;

    if ([data length] == 0){
        self.documentText = @"New Document";
    } else {
        self.documentText = [[NSString alloc] initWithData:data
                                                            encoding:NSUTF8StringEncoding];
    }

    if ([delegate respondsToSelector:@selector(cloudDocumentChanged:)]) {
        [delegate cloudDocumentChanged:self];
    }

    return YES;
}
```

在如上的这个方法中，我们指定当前的协议实现类为当前对象，当文档将会有变化的时候，将会自动的调用更新的一个工作。

紧接着，我们需要找到我们的 `UserDocument.txt` 文件在云端的位置，如果没有的话我们就需要指定一个位置进行创建。如我们 17.3 小节所讲的。

```
- (NSURL *) urlForDocumentsDirectoryInCloud{

    NSURL *result = nil;

    #error Replace this with your own Team ID
    NSString *teamID = @"TEAM ID";

    #error Replace this with your own app identifier
    NSString *containerID = @"com.pixolity.Storing-User-Documents-in-iCloud";

    NSString *teamIDAndContainerID = [NSString stringWithFormat:@"%s.%s",
                                     teamID,
                                     containerID];

    NSFileManager *fileManager = [[NSFileManager alloc] init];
```

```

NSURL *iCloudURL = [fileManager
                    URLForUbiquityContainerIdentifier:teamIDAndContainerID];

NSURL *documentsFolderURLIniCloud = [iCloudURL URLByAppendingPathComponent:@"Documents"
                                     isDirectory:YES];

/* If it doesn't exist, create it */
if ([fileManager fileExistsAtPath:[documentsFolderURLIniCloud path]] == NO){
    NSLog(@"The documents folder does NOT exist in iCloud. Creating...");
    NSError *folderCreationError = nil;
    BOOL created = [fileManager createDirectoryAtURL:documentsFolderURLIniCloud
                                     withIntermediateDirectories:YES
                                     attributes:nil
                                     error:&folderCreationError];

    if (created){
        NSLog(@"Successfully created the Documents folder in iCloud.");
        result = documentsFolderURLIniCloud;
    } else {
        NSLog(@"Failed to create the Documents folder in iCloud. Error = %@",
              folderCreationError);
    }
} else {
    NSLog(@"The Documents folder already exists in iCloud.");
    result = documentsFolderURLIniCloud;
}

```

我们需要使用 `urlForDocumentsDirectoryIniCloud` 这个方法返回的一个 URL，来为我们的 `UserDocument.txt` 进行操作。

```

- (NSURL *) urlForFileInDocumentsDirectoryIniCloud{

    return [[self urlForDocumentsDirectoryIniCloud]
            URLByAppendingPathComponent:@"UserDocument.txt"];

}

```

现在让我们回到我们的试图控制器中，然后进行一些操作。

1. 声明一个 `CloudDocument` 的实例对象。
2. 一个 `UITextView` 的录入框，这样可以让用户录入一些信息然后我们保存在我们的文档中间。
3. 声明一个 `NSMetadataQuery` 对象，进行一个查找匹配的动作

```

#import <UIKit/UIKit.h>
#import "CloudDocument.h"
@interface ViewController : UIViewController
    <CloudDocumentProtocol, UITextViewDelegate>
@property (nonatomic, strong) CloudDocument *cloudDocument;
@property (nonatomic, strong) UITextView *textViewCloudDocumentText;
@property (nonatomic, strong) NSMetadataQuery *metadataQuery;
@end

```


现在我们需要创建一个 UIView 和相关的 UITextView

```
- (void) setupTextView{
    /* Create the text view */

    CGRect textViewRect = CGRectMake(20.0f,
                                     20.0f,
                                     self.view.bounds.size.width - 40.0f,
                                     self.view.bounds.size.height - 40.0f);

    self.textViewCloudDocumentText = [[UITextView alloc] initWithFrame:
                                       textViewRect];

    self.textViewCloudDocumentText.delegate = self;
    self.textViewCloudDocumentText.font = [UIFont systemFontOfSize:20.0f];
    [self.view addSubview:self.textViewCloudDocumentText];
}
```

下面我们将会做一个监听的动作，当用户来时录入信息的时候就会弹出键盘录入信息。

```
- (void) listenForKeyboardNotifications{
    /* As we have a text view, when the keyboard shows on screen, we want to
    make sure our textview's content is fully visible so start
    listening for keyboard notifications */
    [[NSNotificationCenter defaultCenter]
     addObserver:self
     selector:@selector(handleKeyboardWillShow:)
     name:UIKeyboardWillShowNotification
     object:nil];

    [[NSNotificationCenter defaultCenter]
     addObserver:self
     selector:@selector(handleKeyboardWillHide:)
     name:UIKeyboardWillHideNotification
     object:nil];
}
```

然后就是我们前面几个小节讲到的需要实现相关的方法。

```
- (void) startSearchingForDocumentIniCloud{
    /* Start searching for existing text documents */
    self.metadataQuery = [[NSMetadataQuery alloc] init];
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"%K like %@",
        NSMetadataItemFSNameKey,
        @"*"];
    [self.metadataQuery setPredicate:predicate];
    NSArray *searchScopes = [[NSArray alloc] initWithObjects:
        NSMetadataQueryUbiquitousDocumentsScope,
        nil];
    [self.metadataQuery setSearchScopes:searchScopes];

    NSString *metadataNotification = NSMetadataQueryDidFinishGatheringNotification;
    [[NSNotificationCenter defaultCenter] addObserver:self
                                           selector:@selector(handleMetadataQueryFinished:)
                                           name:metadataNotification
                                           object:nil];
}
```



```

                                object:nil];

[self.metadataQuery startQuery];
}

- (void)viewDidLoad{
    [super viewDidLoad];
    [self listenForKeyboardNotifications];
    self.view.backgroundColor = [UIColor brownColor];
    [self setupTextView];
    [self startSearchingForDocumentIniCloud];
}

- (void)viewDidUnload
{
    [[NSNotificationCenter defaultCenter] removeObserver:self];
    [self setTextViewCloudDocumentText:nil];
    [self setMetadataQuery:nil];
    [super viewDidUnload];
}

```

其中 startSearchingForDocumentIniCloud 这个方法中，我们开始需要添加一个监听 NSMetadataQueryDidFinishGatheringNotification。

```

- (void) handleMetadataQueryFinished:(NSNotification *)paramNotification{

    /* Make sure this is the metadata query that we were expecting... */
    NSMetadataQuery *senderQuery = (NSMetadataQuery *)[paramNotification object];

    if ([senderQuery isEqual:self.metadataQuery] == NO){
        NSLog(@"Unknown metadata query sent us a message.");
        return;
    }

    [self.metadataQuery disableUpdates];

    /* Now we stop listening for these notifications because we don't really
    have to, any more */
    NSString *metadataNotification =
        NSMetadataQueryDidFinishGatheringNotification;

    [[NSNotificationCenter defaultCenter] removeObserver:self
                                                name:metadataNotification
                                                object:nil];

    [self.metadataQuery stopQuery];

    NSLog(@"Metadata query finished.");

    /* Let's find out if we had previously created this document in the user's
    cloud space because if yes, then we have to avoid overwriting that
    document and just use the existing one */
    __block BOOL documentExistsIniCloud = NO;
    NSString *FileNameToLookFor = @"UserDocument.txt";

    NSArray *results = self.metadataQuery.results;

    [results enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
    NSMetadataItem *item = (NSMetadataItem *)obj;
    NSURL *itemURL = (NSURL *)[item valueForKey:NSMetadataItemURLKey];
    NSString *lastComponent = (NSString *)[itemURL pathComponents] lastObject];
    if ([lastComponent isEqualToString:FileNameToLookFor]){

```

```

        if ([itemURL isEqual:[self urlForFileInDocumentsDirectoryIniCloud]]){
            documentExistsIniCloud = YES;
            *stop = YES;
        }
    }
    });

    NSURL *urlOfDocument = [self urlForFileInDocumentsDirectoryIniCloud];
    self.cloudDocument = [[CloudDocument alloc] initWithFileURL:urlOfDocument
                                                             delegate:self];

    __weak ViewController *weakSelf = self;

    /* If the document exists, open it */
    if (documentExistsIniCloud){
        NSLog(@"Document already exists in iCloud. Loading it from there...");
        [self.cloudDocument openWithCompletionHandler:^(BOOL success) {
            if (success){
                ViewController *strongSelf = weakSelf;
                NSLog(@"Successfully loaded the document from iCloud.");
                strongSelf.textViewCloudDocumentText.text =
                    strongSelf.cloudDocument.documentText;
            } else {
                NSLog(@"Failed to load the document from iCloud.");
            }
        }
    ];

    } else {
        NSLog(@"Document does not exist in iCloud. Creating it...");

        /* If the document doesn't exist, ask the CloudDocument class to
        save a new file on that address for us */
        [self.cloudDocument saveToURL:[self urlForFileInDocumentsDirectoryIniCloud]
                             forSaveOperation:UIDocumentSaveForCreating
                             completionHandler:^(BOOL success) {
            if (success){
                NSLog(@"Successfully created the new file in iCloud.");
                ViewController *strongSelf = weakSelf;

                strongSelf.textViewCloudDocumentText.text =
                    strongSelf.cloudDocument.documentText;

            } else {
                NSLog(@"Failed to create the file.");
            }
        }
    ];

    }
}

```

其中我们的 text view 如果右边的时候，我们就会把这个变化的信息追加到我们的文档中去，这个时候我们就需要实现 UITextViewDelegate 这个协议的 textViewDidChange 这个方法。

```

- (void) textViewDidChange:(UITextView *)textView{
    self.cloudDocument.documentText = textView.text;
    [self.cloudDocument updateChangeCount:UIDocumentChangeDone];
}

```

在所有多将要处理完成的时候，我们不要忘记，我们要时间键盘监听的事件

。

```
- (void) cloudDocumentChanged:(CloudDocument *)paramSender{
    self.textViewCloudDocumentText.text = paramSender.documentText;
}
- (void) handleKeyboardWillShow:(NSNotification *)paramNotification{

    NSDictionary *userInfo = [paramNotification userInfo];

    NSValue *animationCurveObject =
    [userInfo valueForKey:UIKeyboardAnimationCurveUserInfoKey];

    NSValue *animationDurationObject =
    [userInfo valueForKey:UIKeyboardAnimationDurationUserInfoKey];

    NSValue *keyboardEndRectObject =
    [userInfo valueForKey:UIKeyboardFrameEndUserInfoKey];

    NSInteger animationCurve = 0;
    double animationDuration = 0.0f;
    CGRect keyboardEndRect = CGRectMake(0, 0, 0, 0);

    [animationCurveObject getValue:&animationCurve];
    [animationDurationObject getValue:&animationDuration];
    [keyboardEndRectObject getValue:&keyboardEndRect];

    UIWindow *window = [[[UIApplication sharedApplication] delegate] window];

    /* Convert the frame from window's coordinate system to
    our view's coordinate system */
    keyboardEndRect = [self.view convertRect:keyboardEndRect
                                   fromView:window];

    [UIView beginAnimations:@"changeTextViewContentInset"
                     context:NULL];
    [UIView setAnimationDuration:animationDuration];
    [UIView setAnimationCurve:(UIViewAnimationCurve)animationCurve];

    CGRect intersectionOfKeyboardRectAndWindowRect =
    CGRectIntersection(window.frame, keyboardEndRect);

    CGFloat bottomInset = intersectionOfKeyboardRectAndWindowRect.size.height;

    self.textViewCloudDocumentText.contentInset = UIEdgeInsetsMake(0.0f,
                                                                    0.0f,
                                                                    bottomInset,
                                                                    0.0f);

    [UIView commitAnimations];
}
- (void) handleKeyboardWillHide:(NSNotification *)paramNotification{

    if (UIEdgeInsetsEqualToEdgeInsets(self.textViewCloudDocumentText.contentInset,
                                      UIEdgeInsetsZero)){
        /* Our text view's content inset is intact so no need to reset it */
        return;
    }
}
```

```
}

NSDictionary *userInfo = [paramNotification userInfo];

NSValue *animationCurveObject =
[userInfo valueForKey:UIKeyboardAnimationCurveUserInfoKey];

NSValue *animationDurationObject =
[userInfo valueForKey:UIKeyboardAnimationDurationUserInfoKey];

NSInteger animationCurve = 0;
double animationDuration = 0.0f;

[animationCurveObject getValue:&animationCurve];
[animationDurationObject getValue:&animationDuration];

[UIView beginAnimations:@"changeTextViewContentInset"
               context:NULL];
[UIView setAnimationDuration:animationDuration];
[UIView setAnimationCurve:(UIViewAnimationCurve)animationCurve];

self.textViewCloudDocumentText.contentInset = UIEdgeInsetsZero;

[UIView commitAnimations];
}
```

然后编译运行我们的程序，我们最好能够在两个设备中都运行一下我们的程序，这样我们就能够更好的检验我们的文件和文件夹是否同步到云端中心上去了。当第一台设备运行完成之后，一般稍微过个几秒第二台设备会自动的把文件同步下来。

17.5.4. 参考

暂无。



点击这里访问: DevDiv.com 移动开发论坛