返回期刊列表 (/issues/)

照片框架

MicroCai (https://www.zybuluo.com/MicroCai/note/51120) 2015/03/03

介绍

每天，用 iPhone 拍摄的照片 (https://www.flickr.com/cameras#brands)数量超过了任何相机。每年 iOS 设备上的显示效果变得越来越好，回到 iPad 刚出现 (http://youtu.be/_KN-5zmvjAo?t=17m7s)还没有 Retina 显示屏的时代，大屏幕的杀手级功能之一就是可以展示用户照片和浏览器照片库。自从相机成为 iPhone 最重要和最受欢迎的功能开始，对能管理和加工用户照片库中宝贵的照片的应用程序和工具就有着巨大的需求。

直到 2014 年夏天前，开发者只能用 AssetsLibrary 框架 (https://developer.apple.com/library/ios/documentation/AssetsLibrary/Reference/ALAssetsLibrary_Class/#//apple_ref/doc/uid/T CH1-SW57)访问日益增长的用户的照片库。几年以来，相机应用和照片应用发生了显著的变化，增加了许多新特性，包括按时刻来组织照片的方式。但与此同时，AssetsLibrary 框架却没有跟上步伐。

随着 iOS 8 的到来，苹果给我们提供了一个现代化的框架 —— PhotoKit，它比 AssetsLibrary 表现更好，并且拥有让应用和设备照片库无缝工作的特性。

概要

我们将从框架对象模型的鸟瞰图开始：实体和实体间的关系，获取实体的实例，以及使用获取的结果进行工作。

除此之外，我们的讲解还会涉及到一些在使用 AssetsLibrary 时，尚未对开发者开放的资源元数据。

然后我们会讨论加载资源的图像数据：过程本身，大量可用的选项，一些陷阱和边界案例。

最后，我们会谈谈通过外部参与者来观察照片库的变化，学习如何创建和提交我们自己修改的改变。

PhotoKit 对象模型

PhotoKit 定义了与系统的 Photos 应用内展现给用户的模型对象相一致的实体图表。这些照片实体都是轻量级的不可变对象。所有的 PhotoKit 对象都是继承自 `PHObject` 抽象基类，其公共接口只提供了一个 `localIdentifier` 属性。

`PHAsset` 表示用户照片库中一个单独的资源，用以提供资源的元数据。

成组的资源叫做资源集合，用 `PHAssetCollection` 类表示。一个单独的资源集合可以是照片库中的一个相册或者一个时刻，或者是一个特殊的“智能相册”。这种智能相册包括所有的视频集合，最近添加的项目，用户收藏，所有连拍照片等等 (https://developer.apple.com/library/prerelease/ios/documentation/Photos/Reference/PHAssetCollection_Class/index.html#//ap 是 `PHCollection` 的子类。

`PHCollectionList` 表示一组的 `PHCollections` 。因为它本身就是 `PHCollection` ，所以集合列表可以包含其他集合列表，它们允许复杂的集合继承。实际上，我们可以在照片应用的时刻栏目中看到它：照片 --- 时刻 --- 精选 --- 年度，就是一个例子。

获取 (Fetch) 照片实体

获取 vs. 枚举

那些熟悉 AssetsLibrary 框架的开发者可能会记得 AssetsLibrary 可以用一些特定属性来找到需要的资源，其中一个必须**枚举**用户资源库来获得匹配的资源。不得不承认，这个 API 虽然提供了一些缩小搜索域(https://developer.apple.com/library/ios/documentation/AssetsLibrary/Reference/ALAssetsGroup_Class/index.html#//apple_ref/的方法，但还是十分低效。

而与之形成鲜明对比，PhotoKit 实体的实例是通过**获取**得到的。那些熟悉 Core Data 的人，会觉得和 PhotoKit 在概念和描述都比较接近。

获取请求

获取操作是由上面描述的实体的类方法实现的。要使用哪个类/方法，取决于问题所在范围和你展示与遍历照片库的方式。所有获取方法的命名都是相似的：`class func fetchXXX(..., options: PHFetchOptions) -> PHFetchResult`。`options` 参数给了我们一个对结果进行过滤和排序的途径，这和 `NSFetchRequest` 的 `predicate` 与 `sortDescriptors` 参数类似。

获取结果

你可能已经注意到了这些获取操作不是异步的。它们返回了一个 `PHFetchResult` 对象，可以用类似 `NSArray` 的接口来访问结果内的集合。它会按需动态加载内容并且缓存最近请求的内容。这个行为和设置了 `batchSize` 属性的 `NSFetchRequest` 返回的结果数组相似。对于 `PHFetchResult` 来说，没有办法用参数来指定这个行为，但是官网文档(https://developer.apple.com/library/prerelease/ios/documentation/Photos/Reference/PHFetchResult_Class/index.html)保证 “即使在处理大量的返回结果时，依然能够有最好的表现”。

就算满足请求的照片库内容发生了改变，获取方法所返回的 `PHFetchResult` 对象是不会自动更新。在后面的小节 (http://objccn.io/issue-21-4#The-Times-They-Are-A-Changin)中，我们会介绍如何对返回的 `PHFetchResult` 对象的改变进行观察并处理更新内容。

临时集合 (Transient Collections)

你可能会发现你已经设计了一个可以操作资源集合的组件，并且你还希望它能够处理任意一组的资源。PhotoKit 通过临时资源集合，让我们可以轻松做到这点。

你可以通过 `PHAsset` 对象数组或是包含资源的 `PHFetchResult` 对象来创建临时资源集合。创建的操作在 `PHAssetCollection` 的 `transientAssetCollectionWithAssets(...)` 和 `transientAssetCollectionWithFetchResult(...)` 工厂方法内完成。这些方法创建出来的对象可以像其它的 `PHAssetCollection` 对象一样使用。尽管如此，这些集合不会被存储到用户照片库，自然也不会在照片应用中展示出来。

和资源集合相似，你可以用 `PHCollectionList` 中的 `transientCollectionListWithXXX(...)` 工厂方法来创建临时集合列表。

当你要合并两个获取请求时，你就会发现这个东西非常有用。

照片元数据

正如文章开头提到的，PhotoKit 提供了额外的关于用户资源的元数据，而这些数据在以前使用 ALAssetsLibrary 框架中是没有办法访问，或者很难访问到。

HDR 和全景照片

你可以使用照片资源的 `mediaSubtypes` 属性验证资源库中的图像在捕捉时是否开启了 HDR，拍摄时是否使用了相机应用的全景模式。

收藏和隐藏资源

要验证一个资源是否被用户标记为收藏或被隐藏，只要检查 PHAsset 实例的 favorite 和 hidden 属性即可。

连拍模式照片

对于一个资源，如果其 PHAsset 的 representsBurst 属性为 true，则表示这个资源是一系列连拍照片中的代表照片 (多张照片是在用户按住快门时拍摄的)。它还有一个属性是 burstIdentifier，如果想要获取连拍照片中的剩余的其他照片，可以通过将这个值传入 fetchAssetsWithBurstIdentifier(...) 方法来获取。

用户可以在连拍的照片中做标记；此外，系统也会自动用各种试探来标记用户可能会选择的潜在代表照片。这个元数据是可以通过 PHAsset 的 burstSelectionTypes 属性来访问。这个属性是用三个常量组成的位掩码：.UserPick 表示用户手动标记的资源，.AutoPick 表示用户可能标记的潜在资源，.None 表示没有标记的资源。

这个屏幕快照显示了，照片应用是如何在连拍的照片中自动标记用户可能标记的潜在资源。

照片加载

在处理用户照片库的过去几年中，开发者创造了上百 (如果没有上千) 的小技巧来提高照片加载和展示的效率。这些技巧处理请求的派发和取消，图像大小的修改和裁剪，缓存等等。PhotoKit 提供了一个可以用更加便捷和现代的 API 做了所有这些操作的类： `PHImageManager` 。

请求图像

图像请求是通过 `requestImageForAsset(...)` 方法派发的。这个方法接受一个 `PHAsset` ，可以设置返回图像的大小和图像的其它可选项 (通过 `PHImageRequestOptions` 参数对象设置)，以及结果回调 (result handler)。这个方法的返回值可以用来在所请求的数据不再被需要时取消这个请求。

图像的尺寸和裁剪

奇怪的是，对返回图像的尺寸定义和裁剪的参数是分布在两个地方的。 `targetSize` 和 `contentMode` 这俩参数会被直接传入 `requestImageForAsset(...)` 方法内。这个 content Mode 和 UIView 的 `contentMode` 参数类似，决定了照片应该以按比例缩放还是按比例填充的方式放到目标大小内。注意：如果不对照片大小进行修改或裁剪，那么方法参数是 `PHImageManagerMaximumSize` 和 `PHImageContentMode.Default` 。

此外， `PHImageRequestOptions` 还提供了一些方式来确定图像管理器该**以怎样的方式**来重新设置图像大小。 `resizeMode` 属性可以设置为 `.Exact` (返回图像必须和目标大小相匹配)， `.Fast` (比 .Exact 效率更高，但返回图像可能和目标大小不一样) 或者 `.None` 。还有个值得一提的是， `normalizedCroppingMode` 属性让我们确定图像管理器应该如何裁剪图像。注意：如果设置了 `normalizedcroppingMode` 的值，那么 `resizeMode` 需要设置为 `.Exact` 。

请求递送和进度

默认情况下，如果图像管理器决定要用最优策略，那么它会在将图像的高质量版本递送给你之前，先传递一个较低质量的版本。你可以通过 `deliveryMode` 属性来控制这个行为；上面所描述的默认行为的值为 `.Opportunistic` 。如果你只想要高质量的图像，并且可以接受更长的加载时间，那么将属性设置为 `.HighQualityFormat` 。如果你想要更快的加载速度，且可以牺牲一点图像质量，那么将属性设置为 `.FastFormat` 。

你可以使用 `PHImageRequestOptions` 的 `synchronous` 属性，让 `requestImage...` 系列的方法变成同步操作。注意：当 `synchronous` 设为 `true` 时， `deliveryMode` 属性就会被忽略，并被当作 `.HighQualityFormat` 来处理。

在设置这些参数时，一定要考虑到你的一些用户有可能开启了 iCloud 照片库，这点非常重要。PhotoKit 的 API 不一定会对设备的照片和 iCloud 上照片进行区分 —— 它们都用同一个 `requestImage` 方法来加载。这意味着任意一个图像请求都有可能是一个通过蜂窝网络来进行的非常缓慢的网络请求。当你要用 `.HighQualityFormat` 或者做一个同步请求的时候，要牢记这个。注意：如果你想要确保请求不经过网络，可以将 `networkAccessAllowed` 设为 `false` 。

另一个和 iCloud 相关的属性是 `progressHandler` 。你可以将它设为一个 PHAssetImageProgressHandler (https://developer.apple.com/library/ios/documentation/Photos/Reference/PHImageRequestOptions_Class/index.html#//apple_ 的 block，当从 iCloud 下载照片时，它就会被图像管理器自动调用。

资源版本

PhotoKit 允许应用对照片进行无损的修改。对编辑过的照片，系统会对单独保存一份原始照片的拷贝和针对应用的调整数据。当用图像管理器获取资源时，你可以指定哪个版本的图像资源应该通过 result handler 被递送。这可以通过设置 `version` 属性来做到： `.Current` 会递送包含所有调整和修改的图像； `.Unadjusted`

会递送未被施加任何修改的图像； `.Original` 会递送原始的、最高质量的格式的图像 (例如 RAW 格式的数据。而当将属性设置为 `.Unadjusted` 时，会递送一个 JPEG)。

你可以在 Sam Davies 的文章《照片扩展》 (http://objccn.io/issue-21-5/)中，阅读框架中更多关于这方面的内容。

结果回调 (result handler)

结果回调是一个包含了一个 `UIImage` 变量和一个 `info` 字典作为参数的 block。根据参数和请求的选项，在请求的整个生命周期，它可以被图像管理器多次调用。

`info` 字典提供了关于当前请求状态的信息，比如：

- 图像是否必须从 iCloud 请求 (如果你初始化时将 `networkAccessAllowed` 设置成 `false` ，那么就必须重新请求图像) —— `PHImageResultIsInCloudKey` 。
- 当前递送的 `UIImage` 是否是最终结果的低质量格式。当高质量图像正在下载时，这个可以让你给用户先展示一个预览图像 —— `PHImageResultIsDegradedKey` 。
- 请求 ID (可以便捷的取消请求)，以及请求是否已经被取消 —— `PHImageResultRequestIDKey` 和 `PHImageCancelledKey` 。
- 如果没有图像提供给 result handler，字典内还会有一个错误信息 —— `PHImageErrorKey` 。

这些值可以让你更新你的 UI 来告知用户，和上面讨论到的 `progressHandler` 一起，来表示出它们的加载状态。

缓存

当图像即将要展示在屏幕上时，比如当要在一组滚动的 collection 视图上展示大量的资源图像的缩略图时，预先将一些图像加载到内存中有时是非常有用的。PhotoKit 提供了一个 `PHImageManager` 的子类来处理这种特定的使用场景 —— `PHImageCachingManager` 。

`PHImageCachingManager` 提供了一个关键方法 —— `startCachingImagesForAssets(...)` 。你传入一个 `PHAssets` 类型的数组，一些请求参数，以及一些请求单个图像时即将用到的可选项。此外，还有一些方法可以让你通知缓存管理器来停止缓存特定资源列表，以及停止缓存所有图像。

`allowsCachingHighQualityImages` 属性可以让你指定图像管理器是否应该准备高质量图像。当缓存一个较短和不变的资源列表时，默认 `true` 的属性表现很好。但当要在 collection 视图上快速滑动时做缓存操作的话，最好将它设置成 `false` 。

注意：以我的经验，当用户正在有大量资源的 collection 视图上极其快速的滑动时，使用缓存管理器会损害滑动的表现效果。为这种特定的使用场景定制一个缓存行为是极其重要的。缓存窗口的大小，移动缓存窗口的时机和频率， `allowsCachingHighQualityImages` 的属性值 —— 这些参数都要在目标硬件上的真实照片库中仔细地调节，并测试表现效果。更进一步，你可以考虑在用户行为的基础上，动态的设置这些参数。

请求图像数据

最后，除了请求普通的 `UIImage` 之外， `PHImageManager` 提供了另一个方法可以返回 `NSData` 对象类型的资源数据，包括它的通用类型标识符，图像的展示方向。这个方法返回了这个资源的最多的信息。

沧海桑田，万物变迁

我们已经讨论了在用户照片库中请求资源的元数据，但是到目前为止，我们没提及如何更新我们获取的数据。照片库本质上是一大堆可变的状态，而第一节中提到的照片实体是不可变的对象。PhotoKit 可以让你接收你需要的、关于照片库变动的所有信息，以正确更新你的缓存状态。

观察变化

首先，你需要通过共享的 `PHPhotoLibrary` 对象，用 `registerChangeObserver(...)` 方法注册一个变化观察者 (这个观察者要遵从 `PHPhotoLibraryChangeObserver` 协议)。只要另一个应用或者用户在照片库中做的修改**影响了你在变化前获取的任何资源或资源集合**的话，变化观察者的 `photoLibraryDidChange(...)` 方法都会被调用。这个方法只有一个 `PHChange` 类型的参数，你可以用它来验证这些变化是否和你所感兴趣的获取对象有关联。

更新获取的结果

`PHChange` 提供了几个方法，让你可以通过传入任何你感兴趣的 `PHObject` 对象或 `PHFetchResult` 对象来追踪它们的变化。这几个方法是 `changeDetailsForObject(...)` 和 `changeDetailsForFetchResult(...)`。如果没有任何变化，这些方法会返回 `nil`，否则你可以借助 `PHObjectChangeDetails` 或 `PHFetchResultChangeDetails` 对象来观察这些变化。

`PHObjectChangeDetails` 提供了一个对最新的照片实体对象的引用，以及告诉你对象的图像数据是否曾变化过、对象是否曾被删除过的布尔值。

`PHFetchResultChangeDetails` 封装了施加在你之前通过获取所得到的 `PHFetchResult` 上的变化的信息。`PHFetchResultChangeDetails` 是为了尽可能简化 CollectionView 或 TableView 的更新操作而设计的。它的属性恰好映射到你在使用一个典型的 CollectionView 的 update handler 时所需要的信息。注意，若要正确的更新 `UITableView` / `UICollectionView`，你必须以正确顺序来处理变化，那就是：**RICE** —— **r**emovedIndexes，**i**nsertedIndexes，**c**hangedIndexes，**e**numerateMovesWithBlock (如果 `hasMoves` 为 `true` 的话)。另外，`PHFetchResultChangeDetails` 的 `hasIncrementalChanges` 属性可以被设置成 `false`，这意味着旧的获取结果应该全部被新的值代替。这种情况下，你应该调用 `UITableView/UICollectionView` 的 `reloadData`。

注意：没有必要以集中的方式处理变化。如果你应用中的多个组件需要处理照片实体，那么它们每个都要有自己的 `PHPhotoLibraryChangeObserver`。接着组件就能靠自己查询 `PHChange` 对象，检测是否需要 (以及如何) 更新它们自己的状态。

随风而变

现在我们已经知道了如何观察用户和其他应用造成的变化，我们来尝试一下自己进行改变。

改变存在的对象

用 PhotoKit 在照片库做改变，说到底其实是先创建了一个链接到某个资源或者资源集合的变化请求对象，再设置请求对象的相关属性或调用合适的方法来描述你想要提交的变化。这个必须通过 `performChanges(...)` 方法，在提交到共享的 `PHPhotoLibrary` 的 block 内完成。注意：你需要准备好在 `performChanges` 方法的 completion block 里处理失败的情况。虽然处理的是能被多个参与者 (如你的应用，用户，其他应用，照片扩展等) 改变的状态，但这个方式能提供安全性，也相对易用。

想要修改资源，需要创建一个 `PHAssetChangeRequest` (https://developer.apple.com/library/ios/documentation/Photos/Reference/PHAssetChangeRequest_Class/index.html#//apple_r。然后你就可以修改创建创建日期，资源位置，以及是否将隐藏资源，是否将资源看做用户收藏等。此外，你还可以从用户的库里删除资源。

类似地，若要修改资源集合或集合列表，需要创建一个 `PHAssetCollectionChangeRequest` (https://developer.apple.com/library/ios/documentation/Photos/Reference/PHAssetCollectionChangeRequest_Class/index.html 或 `PHCollectionListChangeRequest` (https://developer.apple.com/library/ios/documentation/Photos/Reference/PHCollectionListChangeRequest_Class/index.html#/ 对象。然后你就可以修改集合标题，添加或删除集合成员，或者完全删除集合。

在你的变化提交到用户照片库前，系统会向用户展示一个明确的获取权限的警告框。

创建新对象

创建一个新资源的做法和修改已存在的资源类似。只要用 `creationRequestForAssetFromXXX(...)` 工厂方法，来创建变化请求，并传入资源图像数据 (或一个 URL)。如果你需要对新建的资源做额外的修改，你可以用创建变化请求的 `placeholderForCreatedAsset` 属性。它会返回一个可用的 placeholder 来代替“真实的” `PHAsset` 引用。

结论

我已经讨论了 PhotoKit 的基础知识，但仍然还有非常多的东西等着我们去发掘。你可以通过查看示例的各处的代码 (https://developer.apple.com/library/ios/samplecode/UsingPhotosFramework/Introduction/Intro.html#//apple_ref/doc/uid/TP400 观看 WWDC session (https://developer.apple.com/videos/wwdc/2014/?id=511) 视频学习更多内容，发掘更深的知识，然后写一些自己的代码！PhotoKit 为 iOS 开发者开启了通往新世界可能性，在未来的数月或者数年里，我们肯定会看到更多基于这个基础构建的富有创造性的优秀产品。

原文 The Photos Framework (http://www.objc.io/issue-21/the-photos-framework.html)

译者简介

MicroCai (https://www.zybuluo.com/MicroCai/note/51120)
会修电脑的单身狗，努力学习 iOS