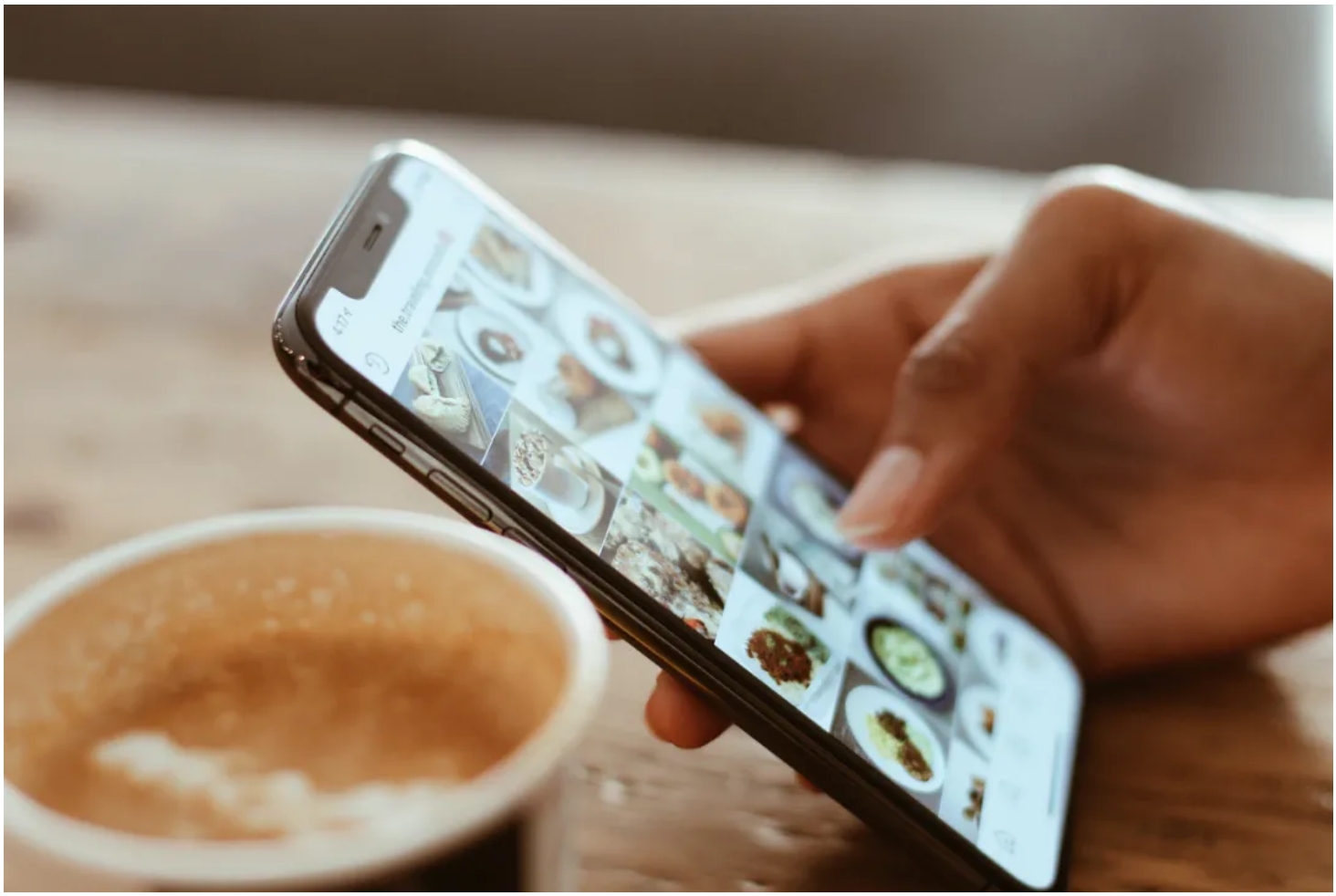
iOS 14 - 使用 PHPicker 选择照片和视频

Harley-xk 知识小集 今天



作者 | Harley-xk
来源 | Harley's Studio，点击阅读原文查看作者更多文章

现状

绝大多数的 App 都要和相册打交道，选择照片或者视频，要么用来发个朋友圈，要么是放到什么地方做个背景。从 AssertLibrary 到 Photos 框架，苹果已经在多年之前就给相册相关的 API 做过一次大升级了。

通过 Photos 框架，只需要通过下面几步就可以从相册获取到内容了

1、获取相册权限
2、建立 UI 展示相册内容
3、响应用户操作，获取照片或视频的 PHAsset 对象
4、通过 PHImageManager 导出照片或者视频

作为一个合格的开发者，对上面这些步骤基本上都能驾轻就熟了。不过虽然看似简单的几步，其实依然存在着巨大的工作量，特别是建立 UI 并展示相册内容这一条，如果要做到细节完善、体验顺畅，基本上属于一个小型 App 的工作量了。所以很多时候我们会选择使用一些第三方的框架，目前可供选择的优秀作品也很多。看起来感觉岁月静好，可以结束工作去泡咖啡了 :)

变化

然而，事情总是在变化的，一向在折腾之路上狂奔的苹果这次依然没有让我们失望，在相册相关的 API 上带来了两个重大变动：

1、相册权限变动

Photos 框架提供了查询相册授权情况的 api，我们可以通过 PHPhotoLibrary 的 authorizationStatus 方法来查询当前的相册授权情况。查询结果 PHAuthorizationStatus 是个枚举，定义如下：

```
public enum PHAuthorizationStatus : Int

    @available(iOS 8, *)
    case notDetermined = 0

    @available(iOS 8, *)
    case restricted = 1

    @available(iOS 8, *)
    case denied = 2
```

```
        @available(iOS 8, *)
        case authorized = 3
    }
```

从 iOS 8 引入 Photos 框架一开始，相册权限就存在这几种状态：

- notDetermined

状态不明确，用户还没有明确授权或拒绝访问，这时候我们一般通过调用 requestAuthorization 方法来显示权限询问弹窗，让用户授权访问。

- restricted

没有访问权限，这个状态表示设备因为特殊原因被禁止访问相册，可能是基于家长控制设置的权限，也可能是当前设备隶属于某个组织，而组织在权限描述文件中禁止了这台设备的相册权限。

这个状态下，用户也无法主动开启相册权限，所以此时唯一能做的就是告诉用户无法获取相册内容。

- denied

禁止访问，这个状态表示用户在上一次询问相册权限时明确选择了禁止。这时候我们可以选择提示用户无法访问，或者提示用户主动去设置中开启权限。

- authorized

允许访问，这个状态表示用户明确同意了相册的授权，此时我们就可以通过 PHAssetCollection 和 PHAsset 相关的 api 来获取相册和其中的照片了。

原本这一切可以很完美的运转，但是如果在 Xcode 12 中查看PHAuthorizationStatus 的定义，会看到从 iOS 14 开始，苹果引入了一个新的权限状态：limited

有限访问权限

苹果在每一年的 WWDC 都特别强调用户隐私，到了今年，苹果终于对相册动手了。现有的方案虽然看起来很完美，但是存在一个重大的隐患：用户往往只需要从相册选

一张照片上传，可此时 App 却获取了整个相册所有照片的数据！！

这时候如果开发者没有守住节操底线，那么用户的所有生活照片甚至是私密照片都存在严重的泄漏风险。

新引入的有限访问权限正是为了解决这个问题。在 iOS 14 下，当我们通过 requestAuthorization 向用户获取权限时，弹窗中多了一个选项：“选择部分照片”，此时系统会在 App 进程之外弹出相册让用户选择授权给当前 App 访问的照片。用户完成选择后，App 通过相册相关的 api 就只能获取到指定的这几张照片，这有效的保护了用户的隐私。

不过，这个设计虽然保护了用户的隐私，但是从 App 的角度来看却引入了新的问题：如果我们还是用原先的那一套流程来获取照片，那么在用户选择了部分权限的情况下，每次弹出相册后用户都无法选择其他的照片，解决办法是再次弹出权限询问，让用户选择或者更换指定的照片，然后再回到我们的相册 UI，继续之后的流程。。。

这一套流程明显变得更加复杂并且奇怪了。。。而且作为一个开发者，始终不能忘了用户是不了解技术细节的，在用户的角度来看，他可能会觉得很奇怪：“为什么我每次都只能选这两张照片？这 App 有什么毛病？”而如果你选择每次都但窗询问权限，那么当用户耐心耗尽的那一天，就是你的 App 被卸载之时。。。

> 基于从 Beta 版就开始使用 iOS 14 的体验，升级之后所有 App 的第一次访问相册时都会弹窗询问权限，目测 iOS 14 应该是重置了所有 App 的相册授权状态。

2、新的相册组件

好在苹果还是给开发者留了一扇窗的，那就是这篇文章要讲的主题，新成员：PHPicker。

其实上面在用户选择部分照片时，我们就已经接触到这玩意儿了：系统在 App 内部额外弹出的相册，本质上就是一个 PHPicker。

苹果在宣传中说它是基于系统的相册 App 的，具有与系统相册一致的 UI 和操作方式，可以保证用户体验的一致性。并且和相册 App 一样，支持通过人物、地点等关键信息来搜索照片。并且 PHPicker 是在独立进程中运行的，与宿主 App 无关，宿主 App 也无法通过截屏 api 来获取当前屏幕上的照片信息。为了保护用户隐私，苹果真的是在各种细节上严防死守。

既然在用户授权时可以在 App 中弹出这个选择器，那么我们的 App 是否可以主动发起这个弹窗呢？答案是：PHPickerViewController

3、PHPickerViewController

PHPickerViewController 是整个 PHPicker 组件的核心，PHPicker 隶属于 PhothsUI 库，使用之前需要先导入：

```
import PhotosUI
```

先来查看这个类的定义：

```
@available(iOS 14, *)
public class PHPickerViewController : UI
}

@available(iOS 14, *)
extension PHPickerViewController {

    /// The configuration passed in duri
    public var configuration: PHPickerCo

    /// The delegate to be notified.
    weak public var delegate: PHPickerVi

    /// Initializes new picker with the
    public convenience init(configuratio
}
```

太简单了！这个类的定义中居然没有任何声明，只是表示了一下自己继承自 UIViewControlelr，以及在扩展中暴露了两个属性和一个构造器。。。

PHPickerConfiguration

查看 PHPickerConfiguration 的定义，发现这里东西稍微多了一点：

- preferredAssetRepresentationMode: PHPickerConfiguration.AssetRepresentationMode

presentationMode 用来指定导出结果的表示形式，默认值是 automatic, 此时系统会自动执行一些转码之类的操作，并且在注释中明确说了这个模式的实际表现会在之后的发布中修改。。。

果然这种表述就是坑的表现：目前官方论坛上有用户反馈，使用了缺省的 automatic 模式后，从相册导出视频的过程变得巨慢：

> It is incredibly slow to return local videos (in the range of several seconds, unacceptable).

It's likely that the picker is transcoding the video for you on the fly. You can set `preferredAssetRepresentationMode` to `.current` to avoid transcoding if your app can handle HEVC videos.

对此官方的回复是 automatic 模式可能会在导出时对视频进行转码处理，如果 App 本地能够处理 HEVC 格式的视频，可以指定为 current 模式来跳过转码的过程。

> 这一条是我在集成过程中遇到的最大的坑之一了，实际测试中，两分钟的原始视频（大约200到300M），使用 automatic 自动转码模式，导出过程耗时达到了难以置信的 5分钟！而改为 current 模式跳过转码之后，导出过程几乎无感（5s以内）。由于我们的 App 本身会将导出的视频压缩转码后再上传，因此果断选择了 current 模式

关于 compatible 模式，注释中用了充满玄学的说法: 尽量选择最合适的形式。由于 PHPicker 相关的资料目前还太少，这个模式的运作方式暂时还摸不清楚，欢迎掉过坑的同学来补充 :)

- selectionLimit

这个设置很好理解，限制用户最多可以选择的数量

- filter: PHPickerFilter

filter 提供了有限的筛选模式设置，目前可以指定筛选照片、视频、LivePhoto 几种类型，或者任何他们的组

合。这一点上目前还是比较欠缺的，比如说发送朋友圈时需要限制用户只能上传一分钟以内的视频，原来我们具有完全相册访问权限的时候，可以在展示的时候直接过滤掉超过一分钟的视频，或者将他们标志为不可选。目前 PHPicker 并没有提供更详细的筛选配置，所以应对这种需求折中的方法是将视频导出之后获取视频的时长，如果超过限制则提示用户。

弹出选择器

弹出窗口其实没什么好说的，PHPickerViewController 本身还是一个 ViewController，设置好相应的属性和 delegate 后，简单的调用 present 就可以了。

- delegate: PHPickerViewControllerDelegate

PHPickerViewController 依然是通过 delegate 属性来向宿主 App 返回用户选择的结果。

看定义同样很简单，PHPickerViewControllerDelegate 只定义了一个方法：

```
public protocol PHPickerViewControllerDe

    /// Called when the user completes a
    ///
    /// The picker won't be automaticall
    func picker(_ picker: PHPickerViewCo
}
```

用户完成选择后，该方法会触发，用户选择的结果会以 PHPickerResult 数组的形式传入，每一个 PHPickerResult 都对应一个照片、视频或者 LivePhoto 的数据。

> 需要注意的是，注释中明确说明了 PHPickerViewController 不会自动关闭，用户需要在选择完毕后，自行调用 dismiss 方法来关闭选择器。

PHPickerResult 的定义一如既往的的“简洁”，只有 itemProvider 和 assetIdentifier 两个属性，剩下的是继承的 Equatable 和 Hashable 协议的实现。

- assetIdentifier

  这是选中对象对应的 PHAsset 的 id，可以用这个 id 通过 PHAset 的相关 api 来进行其他操作。不过苹果在 WWDC 的介绍视频中明确强调了如果要访问 PHAsset 的额外信息，依然需要获取到相册权限后才可以执行。

- itemProvider

NSItemProvider 是一个 iOS 8.0 就存在了的 api，上一次使用还是在 iOS 11 引入 Drag & Drop 的时候了。不过用法依然是一致的：故名思议，这个对象就是用来向我们提供另一个对象的（好像有点绕？） 通过 itemProvider 的 api，我们可以获取到最终的结果，不过这里有点小麻烦：针对照片、视频和 LivePhoto 三种媒体类型，分别要使用不同的 api 来获取

获取照片

获取照片比较简单，通过 NSItemProvider 的 loadObject 方法，并且指定 Class 类型为 UIImage，

就可以在回调中得到 UIImage 类型的照片了：

```
provider.loadObject(ofClass: UIImage.sel
  // do someting with results
}
```

获取 **LivePhoto**

获取 LivePhoto 与获取照片类似，只是需要将 UIImage 替换为 PHLivePhoto。之后你可以通过 PHLivePhotoView 来显示。或者通过 PHAssetResourceManager 获取 LivePhoto 的原始数据。

获取视频

苹果在 WWDC 视频中只演示了如果使用 PHPickerViewController 获取相册照片，但对于如何获取视频只字未提，这就比较尴尬了。目前跟进 PHPicker 的开发者还不多，基本上都搜不到相关资料，

一番折腾之后，终于在官方论坛零星找到了几条关于获取视频的讨论。

查看了相关对话之后，总算摸清了获取视频的套路，要稍微复杂一点：框架开发者明确指出需要使用 loadFileRepresentation 方法来加载大文件，例如视频：

```
provider.loadFileRepresentation(forTypeI
  // do something with results
}
```

loadFileRepresentation 的使用方式与 UIImage 类似，但需要额外传入一个参数 forTypeIdentifier 来指定文件类型，指定为 public.movie 可以覆盖相册中的 .mov 和 .mp4 类型。

与照片不同的是，这个 api 返回的是一个 URL 类型的临时文件路径，苹果在这个 API 的说明中指出：系统会把请求的文件数据复制到这个路径对应的地址，并且在回调执行完毕后删除临时文件。

Summary

Asynchronously writes a copy of the provided, typed data to a temporary file, returning a Progress object.

Declaration

```
func loadFileRepresentation(forTypeIdentifier typeIdentifier:
String, completionHandler: @escaping (URL?, Error?) -> Void) ->
Progress
```

Discussion

This method writes a copy of the file's data to a temporary file, which the system deletes when the completion handler returns.

> 如果你需要在异步线程中对这个文件进行处理，那么需要再复制一次，将文件放到不会被系统自动删除的路径下，并且在处理完毕后自行删除。

关于 **iCloud**

iOS 相册提供了 iCloud 同步功能，如果用户开启了相册同步，那么相册中的照片、视频或者 LivePhoto 有可能会被上传到 iCloud，而本地只保存有缩略图，当请求某张照片时，相册会先从 iCloud 下载，然后再返回数据。

原先具有完整访问权限时，App 可以获得资源是否存在 iCloud 的状态，并且在下载时获得进度信息。由于 PHPicker 向 App 隐藏了所有隐私信息，因此我们无法再得知资源的 iCloud 同步状态，PHPicker 会自动从

iCloud 下载资源，并且完成之后通过 delegate 回调将数据返回。

不过这里有另外一个坑，不知道是因为工期问题还是苹果员工偷懒，目前宿主 App 是无法从 PHPicker 中获取到 iCloud 的下载进度信息的：

> There doesn't appear to be a way to track iCloud download progress (required for larger videos).
>
> It's currently not supported. Please file a Feedback request if possible.

这是 PHPicker 的另一个大坑，这种情况下，如果用户选择了比较大的视频，或者是网络状态不好的话，视频导出依然需要耗费非常长的时间，并且没有进度信息！只能期待 iOS 的后续版本能够将将这一环补充完整了。

总结

PHPicker 的集成本身很简单，但是存在一些坑需要注意。简单总结一下优缺点：

优点

- 保护用户隐私（初期可以作为 App 的宣传点？）
- 不需要频繁询问相册权限，对于用户体验提升较大
- 行为逻辑与系统相册保持一致，降低了用户的学习成本
- 集成简单，在最低支持版本 iOS 14 之后，可以抛弃权限验证相关代码和第三方照片库（似乎很遥远）

缺点

- 太过简单，缺少细节的筛选配置比如视频时长等
- iCloud 大文件下载缺少进度信息（这个缺陷似乎把优点中的用户体验提升干掉了。。。）

> iOS 14 正式版刚上线不久，目前还没有看到适配的大厂 App，可能也是在权衡利弊之中。不过从长远来看，PHPicker 必然会将那些第三方的相册组件扫进历史的垃圾堆的 :)

阅读原文