2016-08-08 • 能工巧匠集

活久见的重构 – iOS 10 UserNotifications 框架解析

TL;DR

iOS 10 中以前杂乱的和通知相关的 API 都被统一了，现在开发者可以使用独立的 UserNotifications.framework 来集中管理和使用 iOS 系统中通知的功能。在此基础上，Apple 还增加了撤回单条通知，更新已展示通知，中途修改通知内容，在通知中展示图片视频，自定义通知 UI 等一系列新功能，非常强大。

对于开发者来说，相较于之前版本，iOS 10 提供了一套非常易用的通知处理接口，是 SDK 的一次重大重构。而之前的绝大部分通知相关 API 都已经被标为弃用 (deprecated)。

这篇文章将首先回顾一下 Notification 的发展历史和现状，然后通过一些例子来展示 iOS 10 SDK 中相应的使用方式，来说明新 SDK 中通知可以做的事情以及它们的使用方式。

您可以在 WWDC 16 的 Introduction to Notifications 和 Advanced Notifications 这两个 Session 中找到详细信息；另外也不要忘了参照 UserNotifications 的官方文档以及本文的实例项目 UserNotificationDemo。

Notification 历史和现状

碎片化时间是移动设备用户在使用应用时的一大特点，用户希望随时拿起手机就能查看资讯，处理事务，而通知可以在重要的事件和信息发生时提醒用户。完美的通知展示可以很好地帮助用户使用应用，体现出应用的价值，进而有很大可能将用户带回应用，提高活跃度。正因如此，不论是 Apple 还是第三方开发者们，都很重视通知相关的开发工作，而通知也成为了很多应用的必备功能，开发者们都希望通知能带来更好地体验和更多的用户。

但是理想的丰满并不能弥补现实的骨感。自从在 iOS 3 引入 Push Notification 后，之后几乎每个版本 Apple 都在加强这方面的功能。我们可以回顾一下整个历程和相关的主要 API:

- iOS 3 – 引入推送通知 `UIApplication` 的 `registerForRemoteNotificationTypes` 与 `UIApplicationDelegate` 的 `application(_:didRegisterForRemoteNotificationsWithDeviceToken:)`, `application(_:didReceiveRemoteNotification:)`
- iOS 4 – 引入本地通知 `scheduleLocalNotification`, `presentLocalNotificationNow:`, `application(_:didReceive:)`
- iOS 5 – 加入通知中心页面
- iOS 6 – 通知中心页面与 iCloud 同步
- iOS 7 – 后台静默推送 `application(_:didReceiveRemoteNotification:fetchCompletionHandle:)`
- iOS 8 – 重新设计 notification 权限请求，Actionable 通知 `registerUserNotificationSettings(_:)`, `UIUserNotificationAction` 与 `UIUserNotificationCategory`, `application(_:handleActionWithIdentifier:forRemoteNotification:completionHandler:)` 等
- iOS 9 – Text Input action，基于 HTTP/2 的推送请求 `UIUserNotificationActionBehavior`，全新的 Provider API 等

有点晕，不是么？一个开发者很难在不借助于文档的帮助下区分 `application(_:didReceiveRemoteNotification:)` 和 `application(_:didReceiveRemoteNotification:fetchCompletionHandle:)`，新入行的开发者也不可能明白 `registerForRemoteNotificationTypes` 和 `registerUserNotificationSettings(_:)` 之间是不是有什么关系，Remote 和 Local Notification 除了在初始化方式之外那些细微的区别也让人抓狂，而很多 API 都被随意地放在了 `UIApplication` 或者 `UIApplicationDelegate` 中。除此之外，应用已经在前台时，远程推送是无法直接显示的，要先捕获到远程来的通知，然后再发起一个本地通知才能完成显示。更让人郁闷的是，应用在运行时和非运行时捕获通知的路径还不一致。虽然这些种种问题都是由一定历史原因造成的，但不可否认，正是混乱的组织方式和之前版本的考虑不周，使得 iOS 通知方面的开发一直称不上“让人愉悦”，甚至有不少“坏代码”的味道。

另一方面，现在的通知功能相对还是简单，我们能做的只是本地或者远程发起通知，然后显示给用户。虽然 iOS 8 和 9 中添加了按钮和文本来进行交互，但是已发出的通知不能更新，通知的内容也只是在发起时唯一确定，而这些内容也只能是简单的文本。 想要在现有基础上扩展通知的功能，势必会让原本就盘根错节的 API 更加难以理解。

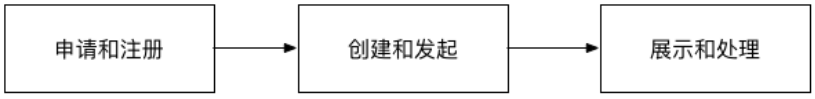
在 iOS 10 中新加入 UserNotifications 框架，可以说是 iOS SDK 发

展到现在的最大规模的一次重构。新版本里通知的相关功能被提取到了单独的框架，通知也不再区分类型，而有了更统一的行为。我们接下来就将由浅入深地解析这个重构后的框架的使用方式。

UserNotifications 框架解析

基本流程

iOS 10 中通知相关的操作遵循下面的流程：



首先你需要向用户请求推送权限，然后发送通知。对于发送出的通知，如果你的应用位于后台或者没有运行的话，系统将通过用户允许的方式 (弹窗，横幅，或者是在通知中心) 进行显示。如果你的应用已经位于前台正在运行，你可以自行决定要不要显示这个通知。最后，如果你希望用户点击通知能有打开应用以外的额外功能的话，你也需要进行处理。

权限申请

通用权限

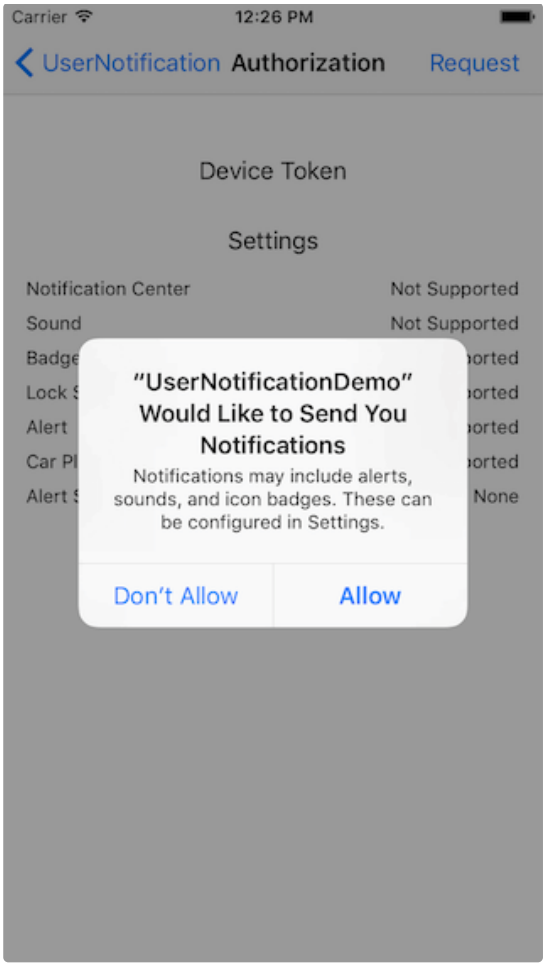
iOS 8 之前，本地推送 (`UILocalNotification`) 和远程推送 (Remote Notification) 是区分对待的，应用只需要在进行远程推送时获取用户同意。iOS 8 对这一行为进行了规范，因为无论是本地推送还是远程推送，其实在用户看来表现是一致的，都是打断用户的行为。因此从 iOS 8 开始，这两种通知都需要申请权限。iOS 10 里进一步消除了本地通知和推送通知的区别。向用户申请通知权限非常简单：

```
UNUserNotificationCenter.current().requestAuthorization(options:
    granted, error in
    if granted {
        // 用户允许进行通知
    }
}
```

当然，在使用 UN 开头的 API 的时候，不要忘记导入 UserNotifications 框架：

```
import UserNotifications
```

第一次调用这个方法时，会弹出一个系统弹窗。



要注意的是，一旦用户拒绝了这个请求，再次调用该方法也不会再进行弹窗，想要应用有机会接收到通知的话，用户必须自行前往系统的设置中为你的应用打开通知，如果不是杀手级应用，想让用户主动去在茫茫多 app 中找到你的那个并专门为你开启通知，往往是不可能的。因此，在合适的时候弹出请求窗，在请求权限前预先进行说明，以此增加通过的概率应该是开发者和策划人员的必修课。相比与直接简单粗暴地在启动的时候就进行弹窗，耐心诱导会是更明智的选择。

远程推送

一旦用户同意后，你就可以在应用中发送本地通知了。不过如果你通过服务器发送远程通知的话，还需要多一个获取用户 token 的操作。你的服务器可以使用这个 token 将用向 Apple Push Notification 的服务器提交请求，然后 APNs 通过 token 识别设备和应用，将通知推给用户。

提交 token 请求和获得 token 的回调是现在“唯二”不在新框架中的 API。我们使用 `UIApplication` 的 `registerForRemoteNotifications` 来注册远程通知，在 `AppDelegate` 的 `application(_:didRegisterForRemoteNotificationsWithDeviceToken)` 中获取用户 token：

```
// 向 APNs 请求 token:
UIApplication.shared.registerForRemoteNotifications()

// AppDelegate.swift
 func application(_ application: UIApplication, didRegisterForRem
    let tokenString = deviceToken.hexString
    print("Get Push token: \(tokenString)")
}
```

获取得到的 `deviceToken` 是一个 `Data` 类型，为了方便使用和传递，我们一般会选择将它转换为一个字符串。Swift 3 中可以使用下面的 `Data` 扩展来构造出适合传递给 Apple 的字符串：

```
extension Data {
    var hexString: String {
        return withUnsafeBytes {(bytes: UnsafePointer<UInt8>) ->
            let buffer = UnsafeBufferPointer(start: bytes, count:
            return buffer.map {String(format: "%02hhx", $0)}.redu
        }
    }
}
```

权限设置

用户可以在系统设置中修改你的应用的通知权限，除了打开和关闭全部通知权限外，用户也可以限制你的应用只能进行某种形式的通知显示，比如只允许横幅而不允许弹窗及通知中心显示等。一般来说你不应该对用户的选择进行干涉，但是如果你的应用确实需要某种特定场景的推送的话，你可以对当前用户进行的设置进行检查：

```
UNUserNotificationCenter.current().getNotificationSettings {
    settings in
    print(settings.authorizationStatus) // .authorized | .denied | .
    print(settings.badgeSetting) // .enabled | .disabled | .notSuppo
    // etc...
}
```

> 关于权限方面的使用，可以参考 Demo 中 `AuthorizationViewController` 的内容。

发送通知

UserNotifications 中对通知进行了统一。我们通过通知的内容 (`UNNotificationContent`)，发送的时机 (`UNNotificationTrigger`) 以及一个发送通知的 `String` 类型的标识符，来生成一个 `UNNotificationRequest` 类型的发送请求。最后，我们将这个请求添

加到 `UNUserNotificationCenter.current()` 中，就可以等待通知到达了：

```
// 1. 创建通知内容
let content = UNMutableNotificationContent()
content.title = "Time Interval Notification"
content.body = "My first notification"

// 2. 创建发送触发
let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5,

// 3. 发送请求标识符
let requestIdentifier = "com.onevcat.usernotification.myFirstNoti

// 4. 创建一个发送请求
let request = UNNotificationRequest(identifier: requestIdentifier

// 将请求添加到发送中心
UNUserNotificationCenter.current().add(request) { error in
    if error == nil {
        print("Time Interval Notification scheduled: \(requestIde
    }
}
```

1. iOS 10 中通知不仅支持简单的一行文字，你还可以添加 `title` 和 `subtitle`，来用粗体字的形式强调通知的目的。对于远程推送，iOS 10 之前一般只含有消息的推送 payload 是这样的：

   ```
   {
     "aps":{
       "alert":"Test",
       "sound":"default",
       "badge":1
     }
   }
   ```

   如果我们想要加入 `title` 和 `subtitle` 的话，则需要将 `alert` 从字符串换为字典，新的 payload 是：

   ```
   {
     "aps":{
       "alert":{
         "title":"I am title",
         "subtitle":"I am subtitle",
         "body":"I am body"
       },
       "sound":"default",
       "badge":1
     }
   }
   ```

好消息是，后一种字典的方法其实在 iOS 8.2 的时候就已经存在了。虽然当时 `title` 只是用在 Apple Watch 上的，但是设置好 `body` 的话在 iOS 上还是可以显示的，所以针对 iOS 10 添加标题时是可以保证前向兼容的。

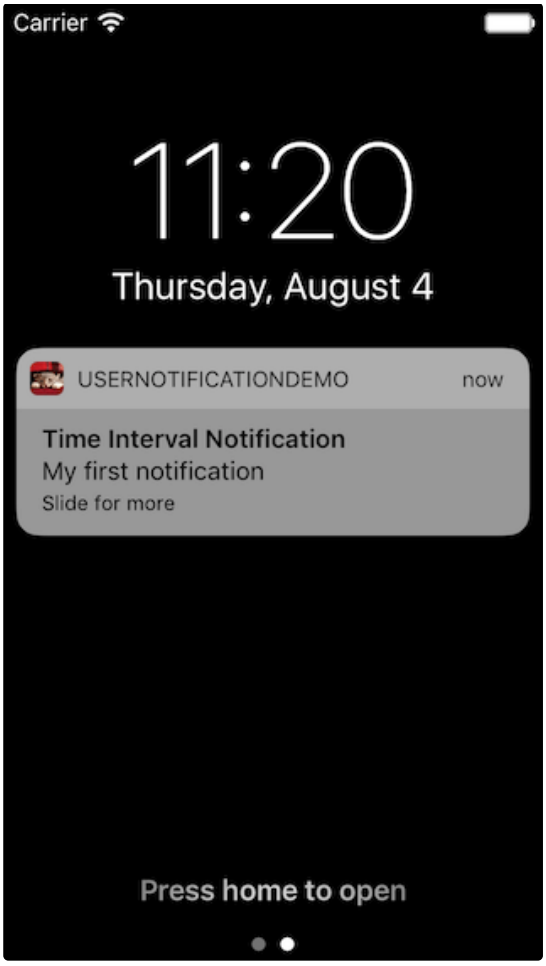
另外，如果要进行本地化对应，在设置这些内容文本时，本地可以使用 `String.localizedUserNotificationString(forKey: "your_key", arguments: [])` 的方式来从 Localizable.strings 文件中取出本地化字符串，而远程推送的话，也可以在 payload 的 alert 中使用 `loc-key` 或者 `title-loc-key` 来进行指定。关于 payload 中的 key，可以参考这篇文档。

2. 触发器是只对本地通知而言的，远程推送的通知的话默认会在收到后立即显示。现在 UserNotifications 框架中提供了三种触发器，分别是：在一定时间后触发 `UNTimeIntervalNotificationTrigger`，在某月某日某时触发 `UNCalendarNotificationTrigger` 以及在用户进入或是离开某个区域时触发 `UNLocationNotificationTrigger`。

3. 请求标识符可以用来区分不同的通知请求，在将一个通知请求提交后，通过特定 API 我们能够使用这个标识符来取消或者更新这个通知。我们将在稍后再提到具体用法。

4. 在新版本的通知框架中，Apple 借用了一部分网络请求的概念。我们组织并发送一个通知请求，然后将这个请求提交给 `UNUserNotificationCenter` 进行处理。我们会在 delegate 中接收到这个通知请求对应的 response，另外我们也有机会在应用的 extension 中对 request 进行处理。我们在接下来的章节会看到更多这方面的内容。

在提交通知请求后，我们锁屏或者将应用切到后台，并等待设定的时间后，就能看到我们的通知出现在通知中心或者屏幕横幅了：



OneV's Den
上善若水，人淡如菊

嗨，我是王巍(@onevcat)，一名来自中国的 iOS / Unity 开发者。现居日本，就职于 LINE。正在修行，探求创意之源。

> 关于最基础的通知发送，可以参考 Demo 中 `TimeIntervalViewController` 的内容。

取消和更新

在创建通知请求时，我们已经指定了标识符。这个标识符可以用来管理通知。在 iOS 10 之前，我们很难取消掉某一个特定的通知，也不能主动移除或者更新已经展示的通知。想象一下你需要推送用户账户内的余额变化情况，多次的余额增减或者变化很容易让用户十分困惑 – 到底哪条通知才是最正确的？又或者在推送一场比赛的比分时，频繁的通知必然导致用户通知中心数量爆炸，而大部分中途的比分对于用户来说只是噪音。

iOS 10 中，UserNotifications 框架提供了一系列管理通知的 API，你可以做到：

- 取消还未展示的通知
- 更新还未展示的通知
- 移除已经展示过的通知
- 更新已经展示过的通知

其中关键就在于在创建请求时使用同样的标识符。

比如，从通知中心中移除一个展示过的通知：

```
let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 3,
let identifier = "com.onevcat.usernotification.notificationWillBe
let request = UNNotificationRequest(identifier: identifier, conte

UNUserNotificationCenter.current().add(request) { error in
    if error != nil {
        print("Notification request added: \(identifier)")
    }
}

delay(4) {
    print("Notification request removed: \(identifier)")
    UNUserNotificationCenter.current().removeDeliveredNotificatio
}
```

类似地，我们可以使用 `removePendingNotificationRequests`，来取消还未展示的通知请求。对于更新通知，不论是否已经展示，都和一开始添加请求时一样，再次将请求提交给 `UNUserNotificationCenter` 即可：

```
// let request: UNNotificationRequest = ...
UNUserNotificationCenter.current().add(request) { error in
    if error != nil {
        print("Notification request added: \(identifier)")
    }
}

delay(2) {
    let newTrigger = UNTimeIntervalNotificationTrigger(timeInterv

    // Add new request with the same identifier to update a notif
    let newRequest = UNNotificationRequest(identifier: identifier
    UNUserNotificationCenter.current().add(newRequest) { error in
        if error != nil {
            print("Notification request updated: \(identifier)")
        }
    }
}
```

远程推送可以进行通知的更新，在使用 Provider API 向 APNs 提交请求时，在 HTTP/2 的 header 中 `apns-collapse-id` key 的内容将被作为该推送的标识符进行使用。多次推送同一标识符的通知即可进行更新。

> 对应本地的 `removeDeliveredNotifications`，现在还不能通过类似的方式，向 APNs 发送一个包含 collapse id 的 DELETE 请

> 求来删除已经展示的推送，APNs 服务器并不接受一个 DELETE 请求。不过从技术上来说 Apple 方面应该不存在什么问题，我们可以拭目以待。现在如果想要消除一个远程推送，可以选择使用后台静默推送的方式来从本地发起一个删除通知的调用。关于后台推送的部分，可以参考我之前的一篇关于 iOS7 中的多任务 的文章。
>
> 关于通知管理，可以参考 Demo 中 `ManagementViewController` 的内容。为了能够简单地测试远程推送，一般我们都会用一些方便发送通知的工具，Knuff 就是其中之一。我也为 Knuff 添加了 `apns-collapse-id` 的支持，你可以在这个 fork 的 repo 或者是原 repo 的 pull request 中找到相关信息。

处理通知

应用内展示通知

现在系统可以在应用处于后台或者退出的时候向用户展示通知了。不过，当应用处于前台时，收到的通知是无法进行展示的。如果我们希望在应用内也能显示通知的话，需要额外的工作。

`UNUserNotificationCenterDelegate` 提供了两个方法，分别对应如何在应用内展示通知，和收到通知响应时要如何处理的工作。我们可以实现这个接口中的对应方法来在应用内展示通知：

```
class NotificationHandler: NSObject, UNUserNotificationCenterDele
    func userNotificationCenter(_ center: UNUserNotificationCente
                        willPresent notification: UNNotification,
                        withCompletionHandler completionHandler: @
    {
        completionHandler([.alert, .sound])

        // 如果不想显示某个通知，可以直接用空 options 调用 completio
        // completionHandler([])
    }
}
```

实现后，将 `NotificationHandler` 的实例赋值给 `UNUserNotificationCenter` 的 `delegate` 属性就可以了。没有特殊理由的话，AppDelegate 的 `application(_:didFinishLaunchingWithOptions:)` 就是一个不错的选择：

```
class AppDelegate: UIResponder, UIApplicationDelegate {
    let notificationHandler = NotificationHandler()
    func application(_ application: UIApplication, didFinishLaunc
        UNUserNotificationCenter.current().delegate = notificatio
```

```
        return true
    }
}
```

对通知进行响应

`UNUserNotificationCenterDelegate` 中还有一个方法，`userNotificationCenter(_:didReceive:withCompletionHandler:)`。这个代理方法会在用户与你推送的通知进行交互时被调用，包括用户通过通知打开了你的应用，或者点击或者触发了某个 action (我们之后会提到 actionable 的通知)。因为涉及到打开应用的行为，所以实现了这个方法的 delegate 必须在 `applicationDidFinishLaunching:` 返回前就完成设置，这也是我们之前推荐将 `NotificationHandler` 尽早进行赋值的理由。

一个最简单的实现自然是什么也不错，直接告诉系统你已经完成了所有工作。

```
func userNotificationCenter(_ center: UNUserNotificationCenter, d
    completionHandler()
}
```

想让这个方法变得有趣一点的话，在创建通知的内容时，我们可以在请求中附带一些信息：

```
let content = UNMutableNotificationContent()
content.title = "Time Interval Notification"
content.body = "My first notification"

content.userInfo = ["name": "onevcat"]
```

在该方法里，我们将获取到这个推送请求对应的 response，`UNNotificationResponse` 是一个几乎包括了通知的所有信息的对象，从中我们可以再次获取到 `userInfo` 中的信息：

```
func userNotificationCenter(_ center: UNUserNotificationCenter, d
    if let name = response.notification.request.content.userInfo[
        print("I know it's you! \(name)")
    }
    completionHandler()
}
```

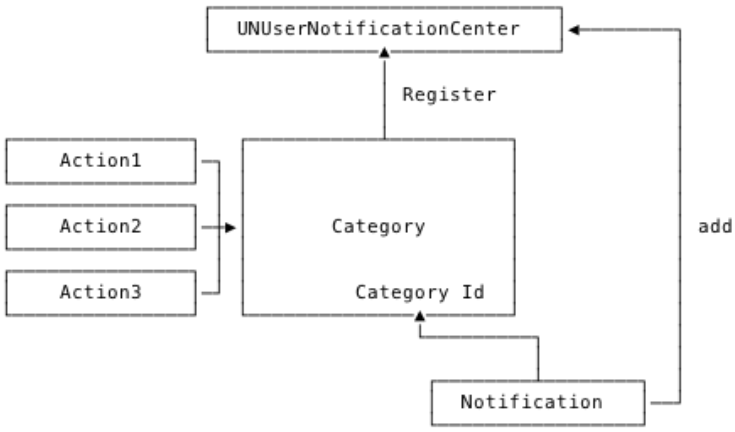
更好的消息是，远程推送的 payload 内的内容也会出现在这个 `userInfo` 中，这样一来，不论是本地推送还是远程推送，处理的路

径得到了统一。通过 `userInfo` 的内容来决定页面跳转或者是进行其他操作，都会有很大空间。

Actionable 通知发送和处理

注册 Category

iOS 8 和 9 中 Apple 引入了可以交互的通知，这是通过将一簇 action 放到一个 category 中，将这个 category 进行注册，最后在发送通知时将通知的 category 设置为要使用的 category 来实现的。



注册一个 category 非常容易：

```
private func registerNotificationCategory() {
    let saySomethingCategory: UNNotificationCategory = {
        // 1
        let inputAction = UNTextInputNotificationAction(
            identifier: "action.input",
            title: "Input",
            options: [.foreground],
            textInputButtonTitle: "Send",
            textInputPlaceholder: "What do you want to say...")

        // 2
        let goodbyeAction = UNNotificationAction(
            identifier: "action.goodbye",
            title: "Goodbye",
            options: [.foreground])

        let cancelAction = UNNotificationAction(
            identifier: "action.cancel",
            title: "Cancel",
            options: [.destructive])

        // 3
        return UNNotificationCategory(identifier:"saySomethingCat
    }()

    UNUserNotificationCenter.current().setNotificationCategories(
}
```

1. `UNTextInputNotificationAction` 代表一个输入文本的 action，你可以自定义框的按钮 title 和 placeholder。你稍后会使用 `identifier` 来对 action 进行区分。
2. 普通的 `UNNotificationAction` 对应标准的按钮。
3. 为 category 指定一个 `identifier`，我们将在实际发送通知的时候用这个标识符进行设置，这样系统就知道这个通知对应哪个 category 了。

当然，不要忘了在程序启动时调用这个方法进行注册：

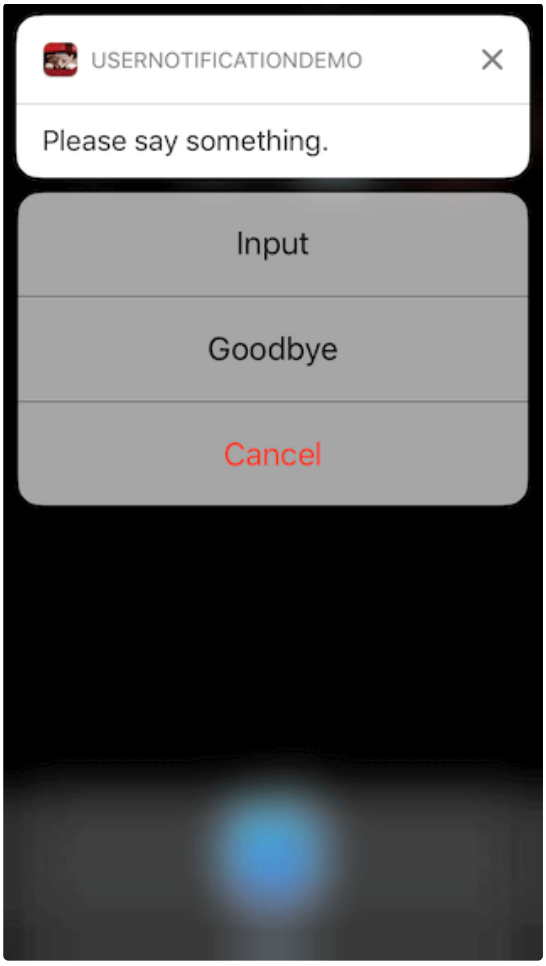
```
func application(_ application: UIApplication, didFinishLaunching
    registerNotificationCategory()
    UNUserNotificationCenter.current().delegate = notificationHan
    return true
}
```

发送一个带有 action 的通知

在完成 category 注册后，发送一个 actionable 通知就非常简单了，只需要在创建 `UNNotificationContent` 时把 `categoryIdentifier` 设置为需要的 category id 即可：

```
content.categoryIdentifier = "saySomethingCategory"
```

尝试展示这个通知，在下拉或者使用 3D touch 展开通知后，就可以看到对应的 action 了：



远程推送也可以使用 category，只需要在 payload 中添加 `category` 字段，并指定预先定义的 category id 就可以了：

```
{
  "aps":{
    "alert":"Please say something",
    "category":"saySomething"
  }
}
```

处理 actionable 通知

和普通的通知并无二致，actionable 通知也会走到 `didReceive` 的 delegate 方法，我们通过 request 中包含的 `categoryIdentifier` 和 response 里的 `actionIdentifier` 就可以轻易判定是哪个通知的哪个操作被执行了。对于 `UNTextInputNotificationAction` 触发的 response，直接将它转换为一个 `UNTextInputNotificationResponse`，就可以拿到其中的用户输入了：

```
func userNotificationCenter(_ center: UNUserNotificationCenter, d

    if let category = UserNotificationCategoryType(rawValue: resp
        switch category {
        case .saySomething:
```

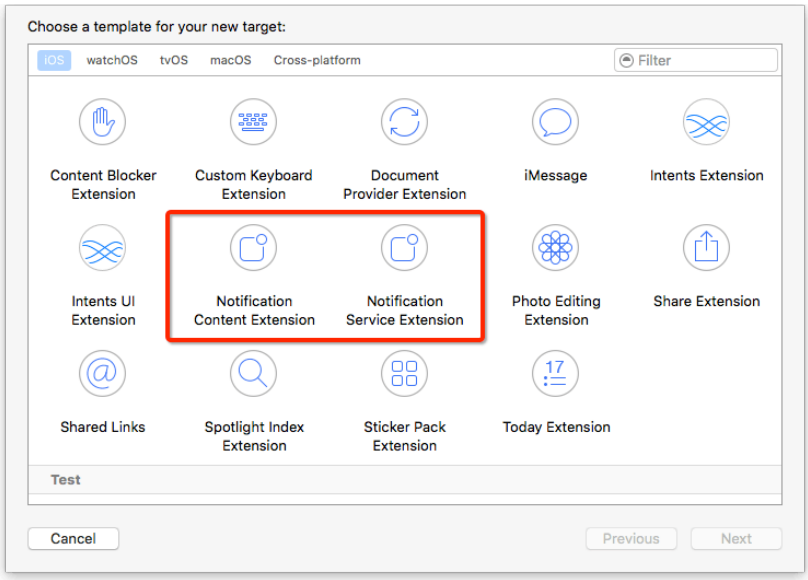
```
            handleSaySomthing(response: response)
        }
    }
    completionHandler()
}

private func handleSaySomthing(response: UNNotificationResponse)
    let text: String

    if let actionType = SaySomethingCategoryAction(rawValue: resp
        switch actionType {
        case .input: text = (response as! UNTextInputNotification
        case .goodbye: text = "Goodbye"
        case .none: text = ""
        }
    } else {
        // Only tap or clear. (You will not receive this callback
        text = ""
    }

    if !text.isEmpty {
        UIAlertController.showConfirmAlertFromTopViewController(m
    }
}
```

上面的代码先判断通知响应是否属于 "saySomething"，然后从用户输入或者是选择中提取字符串，并且弹出一个 alert 作为响应结果。当然，更多的情况下我们会发送一个网络请求，或者是根据用户操作更新一些 UI 等。

> 关于 Actionable 的通知，可以参考 Demo 中 `ActionableViewController` 的内容。

Notification Extension

iOS 10 中添加了很多 extension，作为应用与系统整合的入口。与通知相关的 extension 有两个：Service Extension 和 Content Extension。前者可以让我们有机会在收到远程推送的通知后，展示之前对通知内容进行修改；后者可以用来自定义通知视图的样式。



截取并修改通知内容

`NotificationService` 的模板已经为我们进行了基本的实现：

```
class NotificationService: UNNotificationServiceExtension {

    var contentHandler: ((UNNotificationContent) -> Void)?
    var bestAttemptContent: UNMutableNotificationContent?

    // 1
    override func didReceive(_ request: UNNotificationRequest, wi
        self.contentHandler = contentHandler
        bestAttemptContent = (request.content.mutableCopy() as? U

        if let bestAttemptContent = bestAttemptContent {
            if request.identifier == "mutableContent" {
                bestAttemptContent.body = "\(bestAttemptContent.b
            }
            contentHandler(bestAttemptContent)
        }
    }

    // 2
    override func serviceExtensionTimeWillExpire() {
        // Called just before the extension will be terminated by
        // Use this as an opportunity to deliver your "best attem
        if let contentHandler = contentHandler, let bestAttemptCo
            contentHandler(bestAttemptContent)
        }
    }
}
```

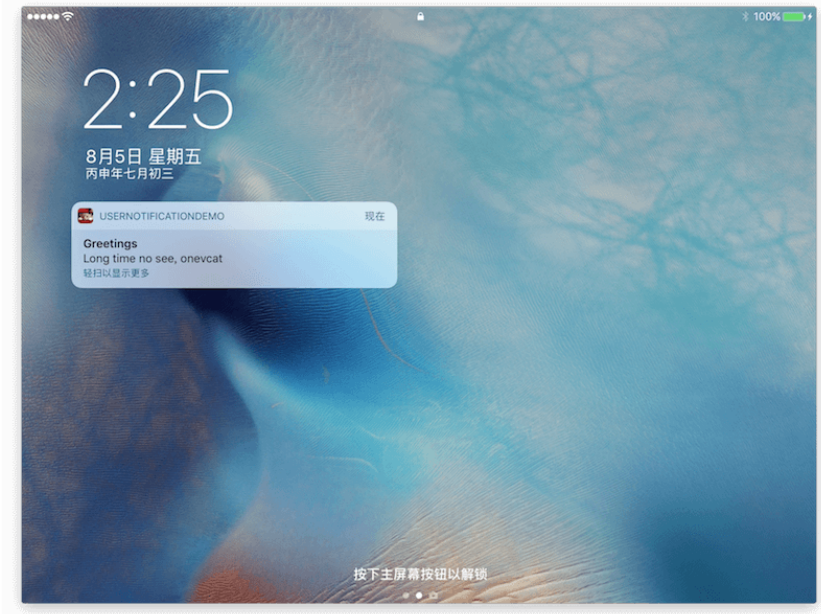
1. `didReceive:` 方法中有一个等待发送的通知请求，我们通过修改这个请求中的 content 内容，然后在限制的时间内将修改后的内容调用通过 `contentHandler` 返还给系统，就可以显示这个修改过的通知了。
2. 在一定时间内没有调用 `contentHandler` 的话，系统会调用这个

> 方法，来告诉你大限已到。你可以选择什么都不做，这样的话系统将当作什么都没发生，简单地显示原来的通知。可能你其实已经设置好了绝大部分内容，只是有很少一部分没有完成，这时你也可以像例子中这样调用 `contentHandler` 来显示一个变更“中途”的通知。

Service Extension 现在只对远程推送的通知起效，你可以在推送 payload 中增加一个 `mutable-content` 值为 1 的项来启用内容修改：

```
{
  "aps":{
    "alert":{
      "title":"Greetings",
      "body":"Long time no see"
    },
    "mutable-content":1
  }
}
```

这个 payload 的推送得到的结果，注意 body 后面附上了名字。



使用在本机截取推送并替换内容的方式，可以完成端到端 (end-to-end) 的推送加密。你在服务器推送 payload 中加入加密过的文本，在客户端接到通知后使用预先定义或者获取过的密钥进行解密，然后立即显示。这样一来，即使推送信道被第三方截取，其中所传递的内容也还是安全的。使用这种方式来发送密码或者敏感信息，对于一些金融业务应用和聊天应用来说，应该是必备的特性。

在通知中展示图片/视频

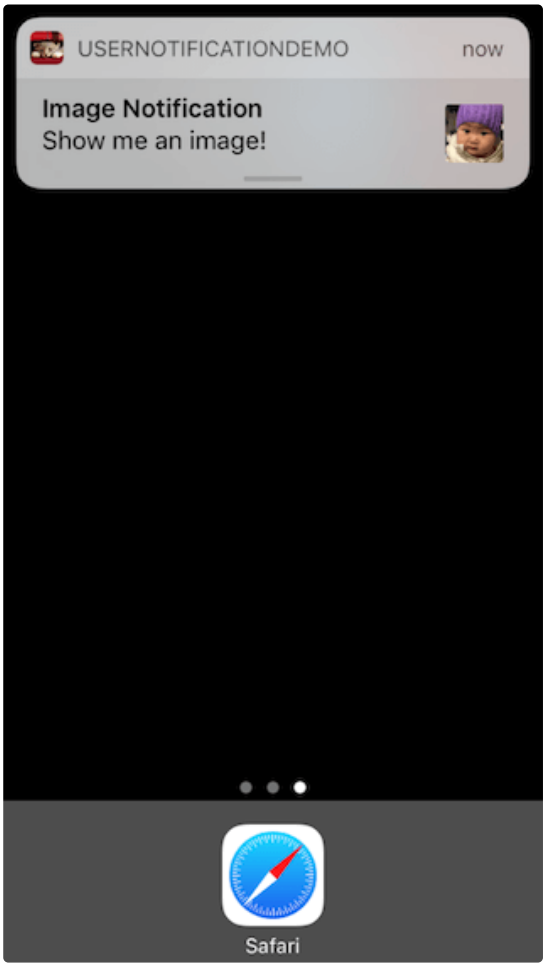
相比于旧版本的通知，iOS 10 中另一个亮眼功能是多媒体的推送。开

发者现在可以在通知中嵌入图片或者视频，这极大丰富了推送内容的可读性和趣味性。

为本地通知添加多媒体内容十分简单，只需要通过本地磁盘上的文件 URL 创建一个 `UNNotificationAttachment` 对象，然后将这个对象放到数组中赋值给 content 的 `attachments` 属性就行了：

```
let content = UNMutableNotificationContent()
content.title = "Image Notification"
content.body = "Show me an image!"

if let imageURL = Bundle.main.url(forResource: "image", withExten
   let attachment = try? UNNotificationAttachment(identifier: "im
{
    content.attachments = [attachment]
}
```

在显示时，横幅或者弹窗将附带设置的图片，使用 3D Touch pop 通知或者下拉通知显示详细内容时，图片也会被放大展示：



除了图片以外，通知还支持音频以及视频。你可以将 MP3 或者 MP4 这样的文件提供给系统来在通知中进行展示和播放。不过，这些文件都有尺寸的限制，比如图片不能超过 5MB，视频不能超过 50MB 等，不过对于一般的能在通知中展示的内容来说，这个尺寸应该是绰绰有余了。关于支持的文件格式和尺寸，可以在文档中进行确认。在创建 `UNNotificationAttachment` 时，如果遇到了不支持的格式，SDK 也会抛出错误。

通过远程推送的方式，你也可以显示图片等多媒体内容。这要借助于上一节所提到的通过 Notification Service Extension 来修改推送通知内容的技术。一般做法是，我们在推送的 payload 中指定需要加载的图片资源地址，这个地址可以是应用 bundle 内已经存在的资源，也可以是网络的资源。不过因为在创建 `UNNotificationAttachment` 时我们只能使用本地资源，所以如果多媒体还不在本地的话，我们需要先将其下载到本地。在完成 `UNNotificationAttachment` 创建后，我们就可以和本地通知一样，将它设置给 `attachments` 属性，然后调用 `contentHandler` 了。

简单的示例 payload 如下：

```
{
```

```
  "aps":{
    "alert":{
      "title":"Image Notification",
      "body":"Show me an image from web!"
    },
    "mutable-content":1
  },
  "image": "https://onevcat.com/assets/images/background-cover.jp
}
```

`mutable-content` 表示我们会在接收到通知时对内容进行更改，`image` 指明了目标图片的地址。

在 `NotificationService` 里，加入如下代码来下载图片，并将其保存到磁盘缓存中：

```
private func downloadAndSave(url: URL, handler: @escaping (_ loca
    let task = URLSession.shared.dataTask(with: url, completionHa
        data, res, error in

        var localURL: URL? = nil

        if let data = data {
            let ext = (url.absoluteString as NSString).pathExtens
            let cacheURL = URL(fileURLWithPath: FileManager.defau
            let url = cacheURL.appendingPathComponent(url.absolut

            if let _ = try? data.write(to: url) {
                localURL = url
            }
        }

        handler(localURL)
    })

    task.resume()
}
```

然后在 `didReceive:` 中，接收到这类通知时提取图片地址，下载，并生成 attachment，进行通知展示：

```
if let imageURLString = bestAttemptContent.userInfo["image"] as?
   let URL = URL(string: imageURLString)
{
    downloadAndSave(url: URL) { localURL in
        if let localURL = localURL {
            do {
                let attachment = try UNNotificationAttachment(ide
                bestAttemptContent.attachments = [attachment]
            } catch {
                print(error)
            }
        }
        contentHandler(bestAttemptContent)
```

```
    }
}
```

关于在通知中展示图片或者视频，有几点想补充说明：

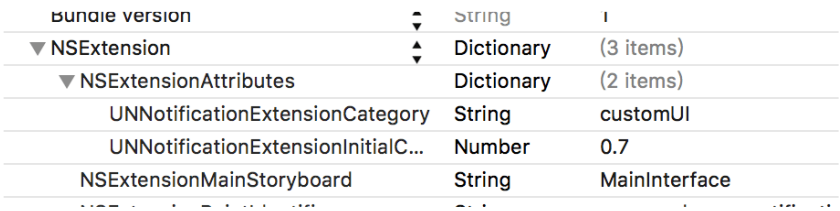
- `UNNotificationContent` 的 `attachments` 虽然是一个数组，但是系统只会展示第一个 attachment 对象的内容。不过你依然可以发送多个 attachments，然后在要展示的时候再重新安排它们的顺序，以显示最符合情景的图片或者视频。另外，你也可能会在自定义通知展示 UI 时用到多个 attachment。我们接下来一节中会看到一个相关的例子。
- 在当前 beta (iOS 10 beta 4) 中，`serviceExtensionTimeWillExpire` 被调用之前，你有 30 秒时间来处理和更改通知内容。对于一般的图片来说，这个时间是足够的。但是如果你推送的是体积较大的视频内容，用户又恰巧处在糟糕的网络环境的话，很有可能无法及时下载完成。
- 如果你想在远程推送来的通知中显示应用 bundle 内的资源的话，要注意 extension 的 bundle 和 app main bundle 并不是一回事儿。你可以选择将图片资源放到 extension bundle 中，也可以选择放在 main bundle 里。总之，你需要保证能够获取到正确的，并且你具有读取权限的 url。关于从 extension 中访问 main bundle，可以参看这篇回答。
- 系统在创建 attachement 时会根据提供的 url 后缀确定文件类型，如果没有后缀，或者后缀无法不正确的话，你可以在创建时通过 `UNNotificationAttachmentOptionsTypeHintKey` 来指定资源类型。
- 如果使用的图片和视频文件不在你的 bundle 内部，它们将被移动到系统的负责通知的文件夹下，然后在当通知被移除后删除。如果媒体文件在 bundle 内部，它们将被复制到通知文件夹下。每个应用能使用的媒体文件的文件大小总和是有限制，超过限制后创建 attachment 时将抛出异常。可能的所有错误可以在 `UNError` 中找到。
- 你可以访问一个已经创建的 attachment 的内容，但是要注意权限问题。可以使用 `startAccessingSecurityScopedResource` 来暂时获取以创建的 attachment 的访问权限。比如：

```
let content = notification.request.content
if let attachment = content.attachments.first {
    if attachment.url.startAccessingSecurityScopedResource()
        eventImage.image = UIImage(contentsOfFile: attachmer
        attachment.url.stopAccessingSecurityScopedResource()
    }
}
```

> 关于 Service Extension 和多媒体通知的使用，可以参考 Demo 中 `NotificationService` 和 `MediaViewController` 的内容。

自定义通知视图样式

iOS 10 SDK 新加的另一个 Content Extension 可以用来自定义通知的详细页面的视图。新建一个 Notification Content Extension，Xcode 为我们准备的模板中包含了一个实现了 `UNNotificationContentExtension` 的 `UIViewController` 子类。这个 extension 中有一个必须实现的方法 `didReceive(_:)`，在系统需要显示自定义样式的通知详情视图时，这个方法将被调用，你需要在其中配置你的 UI。而 UI 本身可以通过这个 extension 中的 MainInterface.storyboard 来进行定义。自定义 UI 的通知是和通知 category 绑定的，我们需要在 extension 的 Info.plist 里指定这个通知样式所对应的 category 标识符：



系统在接收到通知后会先查找有没有能够处理这类通知的 content extension，如果存在，那么就交给 extension 来进行处理。另外，在构建 UI 时，我们可以通过 Info.plist 控制通知详细视图的尺寸，以及是否显示原始的通知。关于 Content Extension 中的 Info.plist 的 key，可以在这个文档中找到详细信息。

虽然我们可以使用包括按钮在内的各种 UI，但是系统不允许我们对这些 UI 进行交互。点击通知视图 UI 本身会将我们导航到应用中，不过我们可以通过 action 的方式来对自定义 UI 进行更新。`UNNotificationContentExtension` 为我们提供了一个可选方法 `didReceive(_:completionHandler:)`，它会在用户选择了某个 action 时被调用，你有机会在这里更新通知的 UI。如果有 UI 更新，那么在方法的 `completionHandler` 中，开发者可以选择传递 `.doNotDismiss` 来保持通知继续被显示。如果没有继续显示的必要，可以选择 `.dismissAndForwardAction` 或者 `.dismiss`，前者将把通知的 action 继续传递给应用的 `UNUserNotificationCenterDelegate` 中的 `userNotificationCenter(:didReceive:withCompletionHandler)`，而后者将直接解散这个通知。

如果你的自定义 UI 包含视频等，你还可以实现 `UNNotificationContentExtension` 里的 `media` 开头的一系列属性，它将为你提供一些视频播放的控件和相关方法。

> 关于 Content Extension 和自定义通知样式，可以参考 Demo 中 `NotificationViewController` 和 `CustomizeUIViewController` 的内容。

总结

iOS 10 SDK 中对通知这块进行了 iOS 系统发布以来最大的一次重构，很多“老朋友”都被标记为了 deprecated：

iOS 10 中被标为弃用的 API

- UILocalNotification
- UIMutableUserNotificationAction
- UIMutableUserNotificationCategory
- UIUserNotificationAction
- UIUserNotificationCategory
- UIUserNotificationSettings
- handleActionWithIdentifier:forLocalNotification:
- handleActionWithIdentifier:forRemoteNotification:
- didReceiveLocalNotification:withCompletion:
- didReceiveRemoteNotification:withCompletion:

等一系列在 `UIKit` 中的发送和处理通知的类型及方法。

现状以及尽快使用新的 API

相比于 iOS 早期时代的 API，新的 API 展现出了高度的模块化和统一特性，易用性也非常好，是一套更加先进的 API。如果有可能，特别是如果你的应用是重度依赖通知特性的话，直接从 iOS 10 开始可以让你充分使用在新通知体系的各种特性。

虽然原来的 API 都被标为弃用了，但是如果你需要支持 iOS 10 之前的系统的话，你还是需要使用原来的 API。我们可以使用

```
if #available(iOS 10.0, *) {
    // Use UserNotification
}
```

的方式来指针对 iOS 10 进行新通知的适配，并让 iOS 10 的用户享受到新通知带来的便利特性，然后在将来版本升级到只支持 iOS 10 以上时再移除掉所有被弃用的代码。对于优化和梳理通知相关代码来说，新 API 对代码设计和组织上带来的好处足以弥补适配上的麻烦，而且它还能为你的应用提供更好的通知特性和体验，何乐不为呢？

更早的文章

关于 iOS 10 中 ATS 的问题

本文于 2016 年 11 月 28 日按照 Apple 最新的文档和 Xcode 8 中的表现进行了部分更新。WWDC 15 提出的 ATS (App Transport Security) 是 Apple 在推进网络通讯安全的一个重要方式。在 iOS 9 和 OS X 10.11 中，默认情况下非 HTTPS 的网络访问是被禁止的。当然，因为这样的推进影响面非常广，作为缓冲，我们可以在 Info.plist 中添加 NSAppTransportSecurity 字典并且将 NSAllo.....

2016-06-17 • 能工巧匠集 继续阅读