返回期刊列表 (/issues/)

iOS 7 的多任务

杨奕洋 (http://weibo.com/ylovesy) 2014/04/21

在 iOS 7 之前，当程序置于后台之后开发者们对他们程序所能做的事情非常有限。除了 VOIP 和基于地理位置特性以外，唯一能做的地方就是使用后台任务（background tasks）让代码可以执行几分钟。如果你想下载比较大的视频文件以便离线浏览，亦或者备份用户的照片到你的服务器上，你都仅能完成一部分工作。

iOS 7 添加了两个新的 API 以便你的程序可以在后台更新界面以及内容。首先是后台获取（Background Fetch），它允许你定期地从网络获取新的内容。第二个 API 就是远程通知（Remote Notifications），这是一个当事件发生时可以让推送通知主动提醒应用的新特性，这两者都为你的应用界面保持最新提供了极大的帮助。在新的后台传输服务 (Background Transfer Service) 中执行定期的任务，也允许你在进程之外可以执行网络传输（下载和上传）工作。

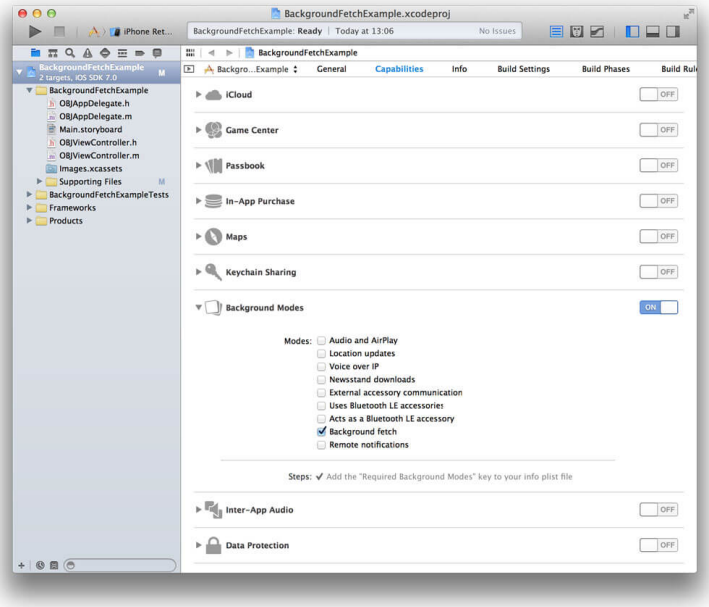
后台获取 (Background Fetch) 和远程通知 (Remote Notification) 基于简单的 ApplicationDelegate 钩子，在应用程序挂起之前的 30 秒时钟时间执行工作。它们不是用于 CPU 频繁工作或者长时间运行任务，而是用来处理长时间运行的网络请求队列，例如下载一部很大的电影，或者执行快速的内容更新。

对用户来说，多任务处理有一点显而易见的改变就是新的应用切换程序 (the new app switcher)，它用来呈现应用到后台时的界面快照。这些快照的存在是有一定理由的--现在你可以在后台完成工作后更新程序快照，以用来呈现新的内容。社交网络、新闻或者天气等应用现在都可以直接呈现最新的内容而不需要用户重新打开应用。我们稍后会介绍如何更新屏幕快照。

后台获取

后台获取是一种智能的轮询机制，它很适合需要经常更新内容的程序，像社交网络，新闻或天气的程序。为了在用户启动程序前提前触发后台获取，系统会根据用户行为唤醒应用程序。举个例子，如果用户经常在下午 1 点使用某个应用程序，系统会学习，适应并在使用周期前执行后台获取。为了减少电池使用，使用设备无线通信的所有应用的后台获取会被合并，如果你向系统报告新数据无法获取，iOS 会适应并使用此信息避免会继续获取。

开启后台获取的第一步是在 info plist 文件中对 UIBackgroundModes (https://developer.apple.com/library/ios/documentation/general/Reference/InfoPlistKeyReference/Articles/iPhoneOSKeys.html#/SW22) 键指定特定的值。最简单的途径是在 Xcode 5 的 project editor 中新的 Capabilities 标签页中设置，这个标签页包含了后台模式部分，可以方便配置多任务选项。



或者，你可以手动编辑这个值

```
<key>UIBackgroundModes</key>
<array>
    <string>fetch</string>
</array>
```

接下来，告诉 iOS 多久进行一次数据获取

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions
{
    [application setMinimumBackgroundFetchInterval:UIApplicationBackgroundFetchIntervalMinimum];

    return YES;
}
```

iOS 默认不进行后台获取，所以你需要设置一个时间间隔，否则，你的应用程序永远不能在后台被唤醒。`UIApplicationBackgroundFetchIntervalMinimum` 这个值要求系统尽可能频繁地去管理你的程序到底什么时候应该被唤醒，但如果你不需要这样的话，你也应该指定一个你想要的的时间间隔。例如，一个天气的应用程序，可能只需要几个小时才更新一次，iOS 将会在后台获取之间至少等待你指定的时间间隔。

如果你的应用允许用户退出登录，那么就没有获取新数据的需要了，你应该把 `minimumBackgroundFetchInterval` 设置为 `UIApplicationBackgroundFetchIntervalNever`，这样可以节省资源。

最后一步是在应用程序委托中实现下列方法：

```
- (void)                application:(UIApplication *)application
  performFetchWithCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler
{
    NSURLSessionConfiguration *sessionConfiguration = [NSURLSessionConfiguration
defaultSessionConfiguration];
    NSURLSession *session = [NSURLSession sessionWithConfiguration:sessionConfiguration];

    NSURL *url = [[NSURL alloc] initWithString:@"http://yourserver.com/data.json"];
    NSURLSessionDataTask *task = [session dataTaskWithURL:url
                                        completionHandler:^(NSData *data, NSURLResponse *response, NSE
rror *error) {

        if (error) {
            completionHandler(UIBackgroundFetchResultFailed);
            return;
        }

        // 解析响应/数据以决定新内容是否可用
        BOOL hasNewData = ...
        if (hasNewData) {
            completionHandler(UIBackgroundFetchResultNewData);
        } else {
            completionHandler(UIBackgroundFetchResultNoData);
        }
    }];

    // 开始任务
    [task resume];
}
```

系统唤醒应用程序后将会执行这个委托方法。需要注意的是，你只有 30 秒的时间来确定获取的新内容是否可用，然后处理新内容并更新界面。30 秒时间应该足够去从网络获取数据和获取界面的缩略图，但是最多只有 30 秒。当完成了网络请求和更新界面后，你应该执行完成的回调。

完成回调的执行有两个目的。首先，系统会估量你的进程消耗的电量，并根据你传递的 `UIBackgroundFetchResult` 参数记录新数据是否可用。其次，当你调用完成的处理代码时，应用的界面缩略图会被采用，并更新应用程序切换器。当用户在应用间切换时，用户将会看到新内容。这种通过 completion handler 来报告并且生成截图的方法，在新的多任务处理 API 中是很常见的。

在实际应用中，你应该将 `completionHandler` 传递到应用程序的子组件，然后在处理完数据和更新界面后调用。

在这里，你可能想知道 iOS 是如何在应用程序后台运行时获得界面截图的，并且想知道应用程序的生命周期与后台获取之间有什么关系。如果应用程序处于挂起状态，系统会先唤醒应用，然后再调用 `application: performFetchWithCompletionHandler:` 。如果应用程序还没有启动，系统将会启动它，然后调用常见的委托方法，包括 `application: didFinishLaunchingWithOptions:` 。你可以把这种应用程序运行的方式想像为用户从 Springboard 启动这个程序，区别仅仅在于界面是看不见的，在屏幕外渲染的。

大多数情况下，无论应用在后台启动或者在前台，你会执行相同的工作，但你可以通过查看 UIApplication 的 applicationState (https://developer.apple.com/library/ios/documentation/uikit/reference/UIApplication_Class/Reference/Reference.html#//apple_ CH3-SW77) 属性来判断应用是不是从后台启动。

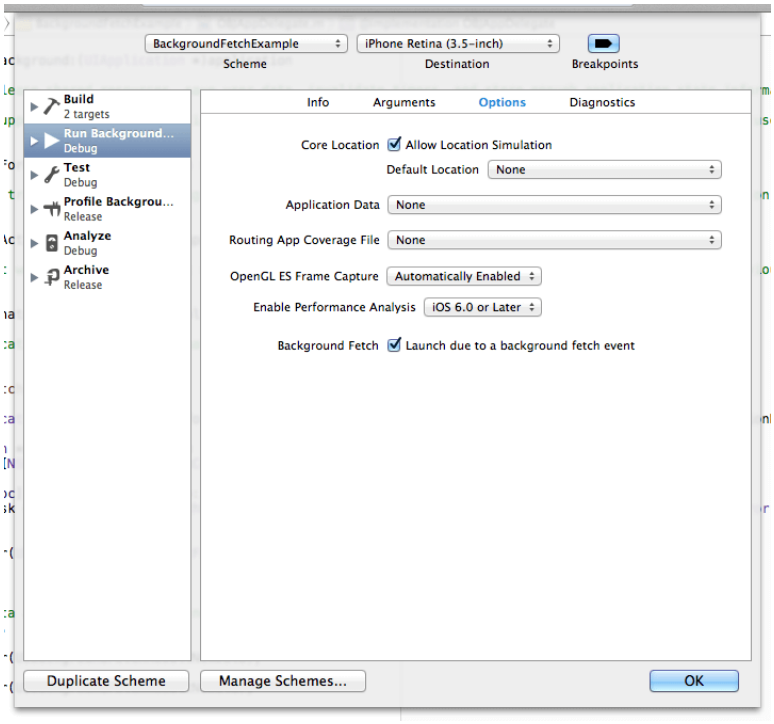
```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions
{
    NSLog(@"Launched in background %d", UIApplicationStateBackground == application.applicationState);

    return YES;
}
```

测试后台数据获取

有两种可以模拟后台获取的途径。最简单是从 Xcode 运行你的应用，当应用运行时，在 Xcode 的 Debug 菜单选择 *Simulate Background Fetch*.

第二种方法，使用 scheme 更改 Xcode 运行程序的方式。在 Xcode 菜单的 Product 选项，选择 Scheme 然后选择 Manage Schemes。在这里，你可以编辑或者添加一个新的 scheme，然后选中 *Launch due to a background fetch event* 。如下图：



远程通知

远程通知允许你在重要事件发生时，告知你的应用。你可能需要发送新的即时信息，突发新闻的提醒，或者用户喜爱电视的最新剧集已经可以下载以便离线观看的消息。远程通知很适合用于那些偶尔出现，但却很重要的内容，如果使用后台获取模式中在两次获取间需要等待的时间是不可接受的话，远程通知会是一个不错的选择。远程通知会比后台获取更有效率，因为应用程序只有在需要的时候才会启动。

一条远程通知实际上只是一条普通的带有 `content-available` 标志的推送通知。你可以发送一条带有提醒信息的推送去告诉用户有事请发生了，同时在后台对界面进行更新。但远程通知也可以做到在安静地，没有提醒消息或者任何声音的情况下，只去更新应用界面或者触发后台工作。然后你可以在完成下载或者处理完新内容后，发送一条本地通知。

静默的推送通知有速度限制，所以你可以大胆地根据应用程序的需要发送尽可能多的通知。iOS 和苹果推送服务会控制推送通知多久被递送，发送很多推送通知是没有问题的。如果你的推送通知达到了限制，推送通知可能会被延迟，直到设备下次发送保持活动状态的数据包，或者收到另外一个通知。

发送远程通知

要发送一条远程通知，需要在推送通知的有效负载（payload）设置 content－available 标志。content-available 标志和用来通知 报刊应用（Newsstand）的健值是一样的，因此，大多数推送脚本和库都已经支持远程通知。当你发送一条远程通知时，你可能还想要包含一些通知有效负载中的数据，让你应用程序可以引用事件。这可以为你节省一些网络请求，并提高应用程序的响应度。

我建议在开发的时候，使用 Nomad CLI's Houston (http://nomad-cli.com/#houston) 工具发送推送消息，当然你也可以使用你喜欢的库或脚本。

你可以通过 nomad-cli ruby gem 来安装 Houston

```
gem install nomad-cli
```

然后通过包含在 Nomad 的 apn 实用工具发送一条通知：

```
# Send a Push Notification to your Device
apn push <device token> -c /path/to/key-cert.pem -n -d content-id=42
```

在这里， -n 标志指定应该包含 content-available 健值， -d 标志允许添加我们自定义的数据健值到有效负荷。

通知的有效负荷（payload）结果和下面类似：

```
{
    "aps" : {
        "content-available" : 1
    },
    "content-id" : 42
}
```

iOS 7 添加了一个新的应用程序委托方法，当接收到一条带有 content－available 的推送通知时，下面的方法会被调用：

```
- (void)application:(UIApplication *)application
  didReceiveRemoteNotification:(NSDictionary *)userInfo
        fetchCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler
{
    NSLog(@"Remote Notification userInfo is %@", userInfo);

    NSNumber *contentID = userInfo[@"content-id"];
    // 根据 content ID 进行操作
    completionHandler(UIBackgroundFetchResultNewData);
}
```

和后台抓取一样，应用程序进入后台启动，也有 30 秒的时间去获取新内容并更新界面，最后调用完成的处理代码。我们可以像后台获取那样，执行快速的网络请求，但我们可以使用新的强大的后台传输服务，处理任务队列，下面看看我们如何在任务完成后更新界面。

NSURLSession 和 后台传输服务（Background Transfer Service）

NSURLSession 是 iOS 7 添加的一个新类，它也是 Foundation networking 中的新技术。作为 NSURLConnection 的替代品，一些熟悉的概念和类都保留下来了，例如 NSURL， NSURLRequest 和 NSURLResponse 。所以，你可以使用 NSURLSessionTask 这一 NSURLConnection 的替代品，来处理网络请求及响应。一共有 3 种会话任务：数据，下载和上传。每一种都向 NSURLSessionTask 添加了语法糖，根据你的需要，适当选择一种。

一个 NSURLSession 对象协调一个或多个 NSURLSessionTask 对象，并根据 NSURLSessionTask 创建的 NSURLSessionConfiguration 实现不同的功能。使用相同的配置，你也可以创建多组具有相关任务的 NSURLSession 对象。要利用后台传输服务，你将会使用 [NSURLSessionConfiguration backgroundSessionConfiguration] 来创建一个会话配置。添加到后台会话的任务在外部进程运行，即使应用程序被挂起，崩溃，或者被杀死，它依然会运行。

NSURLSessionConfiguration (https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLSessionConfiguration_class/Reference/R 允许你设置默认的 HTTP 头，配置缓存策略，限制使用蜂窝数据等等。其中一个选项是 discretionary 标志，这个标志允许系统为分配任务进行性能优化。这意味着只有当设备有足够电量时，设备才通过 Wifi 进行数据传输。如果电量低，或者只仅有一个蜂窝连接，传输任务是不会运行的。后台传输总是在 discretionary 模式下运行。

目前为止，我们大概了解了 NSURLSession，以及一个后台会话如何进行，接下来，让我们回到远程通知的例子，添加一些代码来处理后台传输服务的下载队列。当下载完成后，我们会通知用户该文件已经可以使用了。

NSURLSessionDownloadTask

首先，我们先处理一条远程通知，并把一个 NSURLSessionDownloadTask 添加到后台传输服务的队列。在 backgroundURLSession 方法中，我们根据后台会话配置，创建一个 NSURLSession 对象，并把 application delegate 作为会话的委托对象。文档不建议对于相同的标识符 (identifier) 创建多个会话对象，所以我们使用 dispatch_once 来避免潜在的问题：

```
- (NSURLSession *)backgroundURLSession
{
    static NSURLSession *session = nil;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        NSString *identifier = @"io.objc.backgroundTransferExample";
        NSURLSessionConfiguration* sessionConfig = [NSURLSessionConfiguration
backgroundSessionConfiguration:identifier];
        session = [NSURLSession sessionWithConfiguration:sessionConfig
                                                delegate:self
                                           delegateQueue:[NSOperationQueue mainQueue]];
    });

    return session;
}

- (void)           application:(UIApplication *)application
  didReceiveRemoteNotification:(NSDictionary *)userInfo
        fetchCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler
{
    NSLog(@"Received remote notification with userInfo %@", userInfo);

    NSNumber *contentID = userInfo[@"content-id"];
    NSString *downloadURLString = [NSString stringWithFormat:@"http://yourserver.com/downloads/%d.mp3"
, [contentID intValue]];
    NSURL* downloadURL = [NSURL URLWithString:downloadURLString];

    NSURLRequest *request = [NSURLRequest requestWithURL:downloadURL];
    NSURLSessionDownloadTask *task = [[self backgroundURLSession] downloadTaskWithRequest:request];
    task.taskDescription = [NSString stringWithFormat:@"Podcast Episode %d", [contentID intValue]];
    [task resume];

    completionHandler(UIBackgroundFetchResultNewData);
}
```

我们使用 NSURLSession 类方法创建一个下载任务，配置请求，并提供说明供以后使用。因为所有会话任务一开始处于挂起状态，你必须谨记要调用 [task resume] 保证开始了任务。

现在，我们需要实现 NSURLSessionDownloadDelegate 的委托方法，当下载完成时，调用回调函数。如果你需要处理认证或会话生命周期的其他事件，你可能还需要实现 NSURLSessionDelegate 或 NSURLSessionTaskDelegate 的方法。你应该阅读 Apple 的 Life Cycle of a URL Session with Custom Delegates (https://developer.apple.com/library/ios/documentation/cocoa/Conceptual/URLLoadingSystem/NSURLSessionConcepts/NSUF CH2-SW42) 文档，它讲解了所有类型的会话任务的完整生命周期。

NSURLSessionDownloadDelegate 中的委托方法全部是必须实现的，尽管在这个例子中我们只需要用到 [NSURLSession downloadTask:didFinishDownloadingToURL:]。任务完成下载时，你会得到一个磁盘上该文件的临时 URL。你必须把这个文件移动或复制你的应用程序空间，因为当你从这个委托方法返回时，该文件将从临时存储中删除。

```
#Pragma Mark – NSURLSessionDownloadDelegate

– (void)         URLSession:(NSURLSession *)session
              downloadTask:(NSURLSessionDownloadTask *)downloadTask
  didFinishDownloadingToURL:(NSURL *)location
{
    NSLog(@"downloadTask:%@ didFinishDownloadingToURL:%@", downloadTask.taskDescription, location);

    // 用 NSFileManager 将文件复制到应用的存储中
    // ...

    // 通知 UI 刷新
}

– (void)  URLSession:(NSURLSession *)session
        downloadTask:(NSURLSessionDownloadTask *)downloadTask
   didResumeAtOffset:(int64_t)fileOffset
  expectedTotalBytes:(int64_t)expectedTotalBytes
{
}

– (void)         URLSession:(NSURLSession *)session
              downloadTask:(NSURLSessionDownloadTask *)downloadTask
              didWriteData:(int64_t)bytesWritten totalBytesWritten:(int64_t)totalBytesWritten
  totalBytesExpectedToWrite:(int64_t)totalBytesExpectedToWrite
{
}
```

当后台会话任务完成时，如果你的应用程序仍然在前台运行，上面的代码已经足够了。然而，在大多数情况下，你的应用程序可能是没有运行的，或者在后台被挂起。在这些情况下，你必须实现应用程序委托的两个方法，这样系统就可以唤醒你的应用程序。不同于以往的委托回调，该应用程序委托会被调用两次，因为您的会话和任务委托可能会收到一系列消息。app delegate 的： `handleEventsForBackgroundURLSession：` 方法会在这些 `NSURLSession` 委托的消息发送前被调用，然
后， `URLSessionDidFinishEventsForBackgroundURLSession` 在随后被调用。在前面的方法中，包含了一个后台完成的回调（ `completionHandler` ），并在后面的方法中执行回调以便更新界面:

```
- (void)                  application:(UIApplication *)application
  handleEventsForBackgroundURLSession:(NSString *)identifier completionHandler:(void (^)())completionH
andler
{
    // 你必须重新建立一个后台 seesiong 的参照
    // 否则 NSURLSessionDownloadDelegate 和 NSURLSessionDelegate 方法会因为
    // 没有 对 session 的 delegate 设定而不会被调用。参见上面的 backgroundURLSession
    NSURLSession *backgroundSession = [self backgroundURLSession];

    NSLog(@"Rejoining session with identifier %@ %@", identifier, backgroundSession);

    // 保存 completion handler 以在处理 session 事件后更新 UI
    [self addCompletionHandler:completionHandler forSession:identifier];
}

- (void)URLSessionDidFinishEventsForBackgroundURLSession:(NSURLSession *)session
{
    NSLog(@"Background URL session %@ finished events.\n", session);

    if (session.configuration.identifier) {
        // 调用在 -application:handleEventsForBackgroundURLSession: 中保存的 handler
        [self callCompletionHandlerForSession:session.configuration.identifier];
    }
}

- (void)addCompletionHandler:(CompletionHandlerType)handler forSession:(NSString *)identifier
{
    if ([self.completionHandlerDictionary objectForKey:identifier]) {
        NSLog(@"Error: Got multiple handlers for a single session identifier.  This should not happen.
\n");
    }

    [self.completionHandlerDictionary setObject:handler forKey:identifier];
}

- (void)callCompletionHandlerForSession: (NSString *)identifier
{
    CompletionHandlerType handler = [self.completionHandlerDictionary objectForKey: identifier];

    if (handler) {
        [self.completionHandlerDictionary removeObjectForKey: identifier];
        NSLog(@"Calling completion handler for session %@", identifier);

        handler();
    }
}
```

如果当后台传输完成时，应用程序不再停留在前台，那么，对于更新程序界面来说，这个两步处理过程是必要的。此外，如果当后台传输完成时，应用程序根本没有在运行，iOS 将会在后台启动该应用程序，然后前面的应用程序和会话的委托方法会在 `application:didFinishLaunchingWithOptions:` 方法被调用之后被调用。

配置和限制

我们简单地体验了后台传输的强大之处，但你应该深入文档，阅读 `NSURLSessionConfiguration` 部分，以便最好地满足你的使用场景。例如， `NSURLSessionTasks` 通过 `NSURLSessionConfiguration` 的 `timeoutIntervalForResource` 属性，支持资源超时特性。你可以使用这个特性指定你允许完成一个传输所需的最长时间。内容只在有限的时间可用，或者在用户只有有限 Wifi 带宽的时间内无法下载或上传资源的情况下，你也可以使用这个特性。

除了下载任务， `NSURLSession` 也全面支持上传任务，因此，你可能会在后台将视频上传到服务器，这保证用户不需要再像 iOS 6 那样保持应用程序前台运行。如果当传输完成时你的应用程序不需要在后台运行，一个比较好的做法是，把 `NSURLSessionConfiguration` 的 `sessionSendsLaunchEvents` 属性设置为 `NO` 。高效利用系统资源，是一件让 iOS 和用户都高兴的事。

最后，我们来说一说使用后台会话的几个限制。作为一个必须实现的委托，您不能对 `NSURLSession` 使用简单的基于 block 的回调方法。后台启动应用程序，是相对耗费较多资源的，所以总是采用 HTTP 重定向。后台传输服务只支持 HTTP 和 HTTPS，你不能使用自定义的协议。系统会根据可用的资源进行优化，在任何时候

你都不能强制传输任务在后台进行。

另外，要注意的是在后台会话中， `NSURLSessionDataTasks` 是完全不支持的，你应该只出于短期的，小请求为目的使用这些任务，而不是用来下载或上传。

总结

iOS 7 中强大的多任务和网络 API 为现有应用和新应用开启了一系列全新的可能性。如果你的应用程序可以从进程外的网络传输和数据中获益，那么尽情地使用这些美妙的 API。一般情况下，你可以就像你的应用正在前台运行那样去实现后台传输，并进行适当的界面更新，而这里绝大多数的工作都已经为你完成了。

- 使用适当的新 API 来为你的应用程序提供内容。
- 尽可能早地调用 completion handler 以提高效率。
- 让 completion handler 为应用程序更新界面快照。

扩展阅读

- WWDC 2013 session “What’s New with Multitasking” (https://developer.apple.com/wwdc/videos/?id=204)
- WWDC 2013 session “What’s New in Foundation Networking” (https://developer.apple.com/wwdc/videos/?id=705)
- URL Loading System Programming Guide (https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/URLLoadingSystem/URLLoadingSystem.html#

原文 Multitasking in iOS 7 (http://www.objc.io/issue-5/multitasking.html)

译文 iOS 7系列译文：iOS7的多任务处理 - 博客 - 伯乐在线 (http://blog.jobbole.com/51660/)

校对 Nicholas Tau (http://www.taofengping.com)

译者简介

杨奕洋 (http://weibo.com/ylovesy)
微信 iOS 工程师