

iOS 大文件下载、断点续传、后台下载 —— HERO博客

原创

hero_wqb

于 2018-05-23 14:40:02 发布

16771

收藏 27

版权

分类专栏:

iOS Objective-C技术分享

文章标签:


杀死进程继续下载

断点下载

多文件同时下载

NSURLSession

封装下载

 iOS Objective... 专栏收录该内容

6 订阅

83 篇文章

订阅专栏

本篇简述一下实现文件下载功能，包含大文件下载，后台下载，杀死进程，重新启动时继续下载，设置下载并发数，监听网络改变等，并在最后附有Demo。

下载功能的实现：

使用的网络连接的类为NSURLSession。该类用以替代NSURLConnection，在iOS7时推出，至此iOS系统才有了后台传输。在初始化NSURLSession前，需要先创建NSURLSessionConfiguration，可以理解是为NSURLSession需要的一个配置。NSURLSessionConfiguration有三种模式：

1. default：可以使用缓存的Cache、Cookie、鉴权。
2. ephemeral，仅内存缓存，不使用缓存的Cache、Cookie、鉴权。
3. background，支持后台传输，需要一个identifier标识，用来重新连接session对象。

创建后台模式NSURLSessionConfiguration：

```
NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration backgroundSessionConfigurationWithIdentifier:@"HWDownloadBackgroundSessionIdentifier"];
```

创建NSURLSession，设置配信息、代理、代理线程：

```
NSURLSession *session = [NSURLSession sessionWithConfiguration:configuration delegate:self delegateQueue:[NSOperationQueue alloc] init];
```

在实现下载前，还需要了解一个很重要的类，NSURLSessionTask，无论下载多少文件，我们只需要初始化一个NSURLSession即可，而每个task对应一个任务，需要通过task才能实现下载，NSURLSessionTask是一个基类，有四个子类：

1. NSURLSessionDataTask：下载时，内容以NSData对象返回，需要我们不断写入文件，但不支持后台传输，切换后台会终止下载，回到前台时在协议方法中输出error，下面贴一下用NSURLSessionDataTask实现断点续传的核心代码：

```
1 // 遵守协议
2 <NSURLSessionDataDelegate>
3 // 创建NSMutableURLRequest
4 NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL URLWithString:[NSString stringWithFormat:@"bytes=%zd-", tmpFileSize] forHTTPMethod:@"GET"];
5 [request setValue:@"application/octet-stream" forHTTPHeaderField:@"Accept"];
6 // 创建NSURLSessionDataTask
7 NSURLSessionDataTask *dataTask = [session dataTaskWithRequest:request];
8 // 开始、继续下载
9 [dataTask resume];
10 // 暂停下载
11 [dataTask suspend];
12 // 取消下载
13 [dataTask cancel];
14
15 // 接收到服务器响应后
```

有几点需要注意，调用cancel方法会立即进入-URLSession: task: didCompleteWithError这个回调；调用suspend方法，即使任务已经暂停，但达到超时时长，也会进入这个回调，可以通过error进行判断；当一个任务调用了resume方法，但还未开始接受数据，这时调用suspend方法是无效的。也可以通过cancel方法实现暂停，只是每次需要重新创建NSURLSessionDataTask。

2. NSURLSessionUploadTask：继承自NSURLSessionDataTask，内容以NSData对象返回，协议方法中可以查看请求时上传内容的过程，支持后台传输。

3. NSURLSessionStreamTask：建立了一个TCP/IP连接，替代NSInputStream/NSOutputStream，新的API可异步读写，自动通过HTTP代理连接远程服务器。

4. NSURLSessionDownloadTask：笔者推荐使用该task实现文件下载，断点续传系统帮我们做了，资源会下载到一个临时文件，下载完成需将文件移动到想要的路径，系统会删除临时路劲文件，暂停时，系统会返回NSData对象，恢复下载时用这个data创建task，支持后台传输，下面重点介绍一下NSURLSessionDownloadTask的使用：

创建NSURLSessionDownloadTask，有两种方式，后面会讲解NSData在哪里获取，其中需要注意一点，在iOS 10.0和iOS 10.1系统中，使用downloadTaskWithResumeData会发生数据错误问题，需要进行额外处理，具体可以在Demo中查看：

```
1 // 根据NSData对象创建，可以继续上次进度下载
2 NSURLSessionDownloadTask *downloadTask = [session downloadTaskWithResumeData:resumeData];
3
4 // 根据NSURLRequest对象创建，开启新的下载
5 NSURLSessionDownloadTask *downloadTask = [session downloadTaskWithRequest:[NSURLRequest requestWithURL:[NSURL URLWithString:@"http://www.baidu.com"]]]];
```

开始、继续下载用NSURLSessionTask的resume方法，暂停下载用下面方法，这里拿到回调的NSData，保存，可以通过它来创建task实现继续下载：

```
1 [downloadTask cancelByProducingResumeData:^(NSData * _Nullable resumeData) {
2     model.resumeData = resumeData;
3 }];
```

遵守协议，实现相应协议方法：

NSURLSessionDownloadDelegate:

```
1 /**
2  接收到服务器返回数据，会被调用多次，可获取文件大小，进度，计算速度等
3
4  @param bytesWritten 当次写入文件大小
5  @param totalBytesWritten 已写入文件大小
6  @param totalBytesExpectedToWrite 文件总大小
7  */
8 - (void)URLSession:(NSURLSession *)session downloadTask:(NSURLSessionDownloadTask *)downloadTask didReceiveData:(NSData *)data {
9 {
10     // 计算进度
11     model.progress = 1.0 * totalBytesWritten / totalBytesExpectedToWrite;
12 }
13
14 // 下载完成
15 - (void)URLSession:(NSURLSession *)session downloadTask:(NSURLSessionDownloadTask *)downloadTask didCompleteWithResumeData:(NSData *)resumeData {}
```

NSURLSessionTaskDelegate，注意调用cancel、cancelByProducingResumeData:方法也会调用：

```
1 // 请求完成，有错误时，error有值
2 - (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task didCompleteWithError:(NSError *)error {}
```

后台下载：

到这里，已经可以通过NSURLSessionDownloadTask实现断点续传了，下面介绍如何实现后台下载，其实非常简单，一共三步：

1. 创建NSURLSession时，需要创建后台模式NSURLSessionConfiguration，上面已经介绍过了。
2. 在AppDelegate中实现下面方法，并定义变量保存completionHandler代码块：

```
1 // 应用处于后台，所有下载任务完成调用
2 - (void)application:(UIApplication *)application handleEventsForBackgroundURLSession:(NSString *)identifier completionHandler:(void (^)(void))completionHandler {
3 {
4     _backgroundSessionCompletionHandler = completionHandler;
5 }
6 }
```

3. 在下载类中实现下面NSURLSessionDelegate协议方法，其实就是先执行完task的协议，保存数据、刷新界面之后再执行在AppDelegate中保存的代码块：

```
1 // 应用处于后台，所有下载任务完成及NSURLSession协议调用之后调用
2 - (void)URLSessionDidFinishEventsForBackgroundURLSession:(NSURLSession *)session {
3 {
4     dispatch_async(dispatch_get_main_queue(), ^{
5         AppDelegate *appDelegate = (AppDelegate *)[UIApplication sharedApplication].delegate;
6         if (appDelegate.backgroundSessionCompletionHandler) {
7             void (^completionHandler)(void) = appDelegate.backgroundSessionCompletionHandler;
8             appDelegate.backgroundSessionCompletionHandler = nil;
9
10            // 执行block，系统后台生成快照，释放阻止应用挂起的断言
11            completionHandler();
12        }
13    });
14 }
```

程序终止，再次启动继续下载：

后台下载实现之后，再看一下如何实现进程杀死后，再次启动时继续下载，在应用程序被杀掉时，系统会自动保存应用下载session信息，重新启动应用时，如果创建和之前相同identifier的session，系统会找到对应的session数据，并响应-URLSession: task: didCompleteWithError:方法，打印Error输出如下：

```
error: Error Domain=NSURLErrorDomain Code=-999 "(null)" UserInfo={NSURLErrorBackgroundTaskCancelledReasonKey=0, NSErrorFailingURLStringKey=https://www.apple.com/105/media/cn/iphone-x/2017/01df5b43-28e4-4848-bf20-490c34a926a7/films/feature/iphone-x-feature-cn-20170912_1280x720h.mp4, NSErrorFailingURLKey=https://www.apple.com/105/media/cn/iphone-x/2017/01df5b43-28e4-4848-bf20-490c34a926a7/films/feature/iphone-x-feature-cn-20170912_1280x720h.mp4, NSURLSessionDownloadTaskResumeData=<CFData 0x7ff401097c00 [0x104cb6bb0]>{Length = 6176, capacity = 6176, bytes = 0x3c3f786d6c20766572736966e3d2231... 2f706c6973743e0a}}
```

可以看到，有几点有用的信息：

- 1) error.localizedDescription为"(null)"，打印结果为"The operation couldn't be completed. (NSURLErrorDomain error -999.)"。
- 2) [error.userInfo objectForKey:NSURLErrorBackgroundTaskCancelledReasonKey]有值。
- 3) 返回了NSURLSessionDownloadTaskResumeData。

综上进程杀死后，再次启动继续下载的思路就是，重启时，创建相同identifier的session，在-URLSession: task: didCompleteWithError:方法中拿到resumeData，用resumeData创建task，就可以恢复下载。

再说明一下，另外一种不可取的思路，在AppDelegate中进程杀死时会调用-applicationWillTerminate:方法，在这里task调用cancelByProducingResumeData:方法暂停正在下载的任务，但是这个方法的回调需要时间，还没有执行到代码块进程就已经终止了。

并发数设置：

下面介绍一下下载并发数的设置：NSURLSession本身就支持多任务同时下载，它会根据性能内部控制同时下载的个数，最多5个。一个任务对应一个NSURLSessionDownloadTask，所以想多任务同时下载，需要创建多个task，可以用数组或字典保存。我们定义变量去记录当前下载文件个数及用户设置的最大下载个数。

监听网络改变：[用AFN监听，可以点击这里查看](#)

为了增加用户体验，往往在设置中会给用户一个选项，选择蜂窝网络下是否允许下载。

NSURLSessionConfiguration本身就有一个属性allowsCellularAccess，默认为YES，允许蜂窝网络下载。如果不需要用户随时变更这个选项，是可以这个属性。但是对于正在下载的任务，修改这个属性是无效的，即我们已经通过session创建了task对象，开启了任务，再试图用session.configuration.allowsCellularAccess = NO;去修改这个选项是无效的。如果一定要用这个属性修改这个选项，那么只能重新创建session：

```
1 // 重新创建后台NSURLSessionConfiguration，并且identifier需要改变，不能与之前一样
2 NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration backgroundSe
3 // 修改是否允许蜂窝网络下载
4 configuration.allowsCellularAccess = NO;
5 // 重新创建NSURLSession
6 _session = [NSURLSession sessionWithConfiguration:configuration delegate:self dele
```

所以我们创建NSURLSessionConfiguration时把allowsCellularAccess设为YES，然后定义一个变量去控制是否允许蜂窝网络下载，在网络状态改变及用户设置修改这个选项之后，调用暂停、开启任务。

数据保存：[用FMDB存储数据，可以点击这里查看](#)

下载速度计算：

声明两个变量，一个记录时间，一个记录在特定时间内接收到的数据大小，在接收服务器返回数据的-URLSession: downloadTask: didWriteData: totalBytesWritten: totalBytesExpectedToWrite:方法中，统计接收到数据的大小，达到时间限时，计算速度=数据/时间，然后清空变量，为方便数据库存储，这里用的时间戳：

```
1 - (void)URLSession:(NSURLSession *)session downloadTask:(NSURLSessionDownloadTask
2 {
3     // 记录在特定时间内接收到的数据大小
4     model.intervalFileSize += bytesWritten;
5
6     // 获取上次计算时间与当前时间间隔
7     NSInteger intervals = [[NSDate date] timeIntervalSinceDate:[NSDate dateWithTim
8     if (intervals >= 1) {
9         // 计算速度
10         model.speed = model.intervalFileSize / intervals;
11
12         // 重置变量
13         model.intervalFileSize = 0;
14         model.lastSpeedTime = [[NSNumber numberWithInt:[NSDate date] timeInter
15     }
16 }
```

Demo效果图：



这里在模型中加入了一个变量，记录任务加入准备下载的时间，用于计算任务开始的先后顺序，如上图3，开启任务08、09、10、11、12，暂停，依次开启10、11、12、08、09，然后将最大并发数由5改为2，暂停的应该为12、08、09三个任务，当10下载完成，开启的应该是12而不是08。

Demo下载链接：<https://github.com/HeroWqb/HWDownloadDemo>

写博客的初心是希望大家共同交流成长，博主水平有限难免有偏颇之处，欢迎批评指正。