三、 网络编程之NSURLSession

> *经历了前面的学习，我们发现发送一个应用层的请求是一件非常复杂的工作，并且需要我们封装非常多的东西，比如缓存策略，取消、继续请求，线程操作等等。Apple团队给出了解决方案，2013年（iOS 7）以前，使用NSURLConnection，在iOS 7 的重大更新中，包括了Foundation框架的另外一个网络框架——NSURLSession，替代之前的NSURLConnection。本节我们将学习通过NSURLSession完成一些列的HTTP网络请求。*

目录：

1、NSURLSession简介

1.1 NSURLSession为何物

NSURLSession是一系列类的总称，核心类包括：NSURLRequest、NSCache、NSURLSession、NSURLSessionConfiguration、NSURLSessionTask。

1.2 NSURLSession有何用

NSURLSession帮助我们快速的实现网络请求，并且非常方便的管理网络请求任务，例如后台任务，暂停继续任务等等。

1.3 NSURLSession如何用

NSURLSession 的使用最基本的有3要素：

- NSURLSession：对话对象，可以获取全局的，也可以自己创建。

```
  + (NSURLSession *)sharedSession;
  + (NSURLSession *)sessionWithConfiguration:
(NSURLSessionConfiguration *)configuration;
  + (NSURLSession *)sessionWithConfiguration:
(NSURLSessionConfiguration *)configuration delegate:(id
<NSURLSessionDelegate>)delegate delegateQueue:(NSOperationQueue
*)queue;
```

- NSURLSessionConfiguration：用于配置NSURLSession对象，比如缓存策略等等。有三种常用配置：

```
  + (NSURLSessionConfiguration *)defaultSessionConfiguration;  默认配置
  + (NSURLSessionConfiguration *)ephemeralSessionConfiguration;  短暂型配置
  + (NSURLSessionConfiguration *)backgroundSessionConfigurationWithIdentifier:(NSString *)identifier NS_AVAILABLE(10_10, 8_0);    后台对话型配置
```

    - 默认配置将请求数据持久化保存，程序关闭也不会清除。
    - 短暂型配置会将数据保存在缓存文件夹中，当程序退出时清空。
    - 后台配置将会使NSURLSession对话可以在后台中运行，并在恢复到前台时给程序反馈。
- NSURLSessionTask：用于通过不同的数据获取方式创建不同的任务，通过task管理任务。比如task可以暂停或者继续等等。每一个网络请求都可以看多一个任务，NSURLSessionTask一般不会直接使用，而是使用子类：
    - NSURLSessionDataTask： 一般的数据请求任务
    - NSURLSessionDownloadTask： 文件的下载请求任务
    - NSURLSessionUploadTask： 文件的上传任务

使用时首先设置好NSURLSessionConfiguration，然后根据需求配置初始化一个NSURLSession，最后发起不同的任务即可。下面我们将演示几种请求方式。

2、NSURLSession 使用

首先我们先创建一个默认配置的Session：

```
//session配置对象
NSURLSessionConfiguration *config = [NSURLSessionConfiguration defaultSessionConfiguration];

//session
session = [NSURLSession sessionWithConfiguration:config delegate:self delegateQueue:nil];
```

这里有一步，设置代理的工作，设置代理后，Session所有的task都将回调该代理的代理方法。

所有的任务都需要一个NSURLRequest描述请求内容：

可以这是请求的具体URL，请求头，请求体，超时时间，缓存策略等，一般使用NSURLMutableRequest：

```
    NSURL *url = [NSURL
URLWithString:@"https://api.douban.com/v2/movie/us_box"];

    double timeout = 10;

    NSURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:url
cachePolicy:NSURLRequestUseProtocolCachePolicy timeoutInterval:timeout];
```

很多设置也可以在NSURLSessionConfiguration进行统一的配置，后面无需再针对每一个任务单独配置，两者不冲突。

2.1 数据任务——NSURLSessionDataTask

上节中我们使用了豆瓣的API获取json数据，这一类的请求任务就属于数据任务，我们可以使用NSURLSessionDataTask完成该工作：

```
    NSURL *url = [NSURL
URLWithString:@"https://api.douban.com/v2/movie/us_box"];

    double timeout = 10;

    NSURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:url
cachePolicy:NSURLRequestUseProtocolCachePolicy timeoutInterval:timeout];

    NSURLSessionDataTask *data_task = [session dataTaskWithRequest:request
completionHandler:^(NSData *  data, NSURLResponse *  response, NSError *
error) {

        NSLog(@"%@",[[NSString alloc] initWithData:data
encoding:NSUTF8StringEncoding]);

    }];

    [data_task resume];
```

2.2 下载任务——NSURLSessionDownloadTask

上节课中我们请求一个图片地址，这一类其实就是下载任务，下面我们从微博服务器中获取一张图片：

```
NSURL *url = [NSURL
URLWithString:@"http://ww1.sinaimg.cn/bmiddle/005wayyCgw1ew6yyvcv7vj30fa0fz4
0g.jpg"];

double timeout = 10;

NSURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:url
cachePolicy:NSURLRequestReturnCacheDataElseLoad timeoutInterval:timeout];


NSURLSessionDownloadTask *download = [session
downloadTaskWithRequest:request completionHandler:^(NSURL *  location,
NSURLResponse *  response, NSError *  error) {

    NSLog(@"%@",location);
//        block 完成后将删除tmp文件

}];

[download resume];
```

2.2.1 下载进度

在下载时，UI经常需要监听下载进度，NSURLSessionDownloadTask给了很好的解决方案，
创建任务时我们不能使用block的方式： NSURL *url = [NSURL
URLWithString:@"http://ww1.sinaimg.cn/bmiddle/005wayyCgw1ew6yyvcv7vj30fa0fz40g.jpg"];

```
double timeout = 10;

NSURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:url
cachePolicy:NSURLRequestReturnCacheDataElseLoad timeoutInterval:timeout];

/*
NSURLSessionDownloadTask *download = [session
downloadTaskWithRequest:request completionHandler:^(NSURL *  location,
NSURLResponse *  response, NSError *  error) {

    NSLog(@"%@",location);
//        block 完成后将删除tmp文件

}];

 */

NSURLSessionDownloadTask *download = [session
downloadTaskWithRequest:request];

[download resume];
```

同时我们需要添加代理方法:

```
//每次下载一定的数据后调用
- (void)URLSession:(NSURLSession *)session downloadTask:
(NSURLSessionDownloadTask *)downloadTask
      didWriteData:(int64_t)bytesWritten
 totalBytesWritten:(int64_t)totalBytesWritten
totalBytesExpectedToWrite:(int64_t)totalBytesExpectedToWrite{

    NSLog(@"%ld / %ld",totalBytesWritten,totalBytesExpectedToWrite);

}
//下载完毕后调用
- (void)URLSession:(NSURLSession *)session downloadTask:
(NSURLSessionDownloadTask *)downloadTask
didFinishDownloadingToURL:(NSURL *)location{
    NSLog(@"%@",location);
```

}

2.3 上传任务——NSURLSessionUploadTask

上传任务一般为POST请求方式，需要设置NSURLRequest的请求体，然后把请求体作为发送的数据发送出去。

2.3.1 POST请求

POST请求体的Content-type有很多种，最常用的是form-data 即表单数据。拼接form表单的方式如下：

```
/*

 Content-type: multipart/form-data, boundary=AaB03x

 以下是请求体内容

 --AaB03x
 content-disposition: form-data; name="field1"

 Joe Blow
 --AaB03x
 content-disposition: form-data; name="pics"; filename="file1.txt"
 Content-Type: text/plain

 ... contents of file1.txt ...
 --AaB03x--
 */
```

- 首先在请求头中要明确 请求体的格式和间隔符（间隔符只有form-data才使用）
- 然后两个空白行
- 然后是表单中的数据

具体拼接代码如下：

```
//生成请求体
+(NSData*)httpFormDataBodyWithBoundary:(NSString*)boundary
                                params:(NSDictionary*)params
                             fieldName:(NSString*)fieldName
                              fileName:(NSString*)fileName
                       fileContentType:(NSString*)fileContentType
                                  data:(NSData*)fileData{

    NSString*preBoundary = [NSString stringWithFormat:@"--%@",boundary];

    NSString*endBoundary = [NSString stringWithFormat:@"--%@--",boundary];


    NSMutableString *body = [[NSMutableString alloc] init];

    //遍历keys
    for(NSString*key in params)
    {
        //得到当前key
```

```
        //如果key不是pic，说明value是字符类型，比如name：Boris
        //添加分界线，换行 必须使用  \r\n
        [body appendFormat:@"%@\r\n",preBoundary];
        //添加字段名称，换2行
        [body appendFormat:@"Content-Disposition: form-data;
name=\"%@\"\r\n\r\n",key];
        //添加字段的值
        [body appendFormat:@"%@\r\n",[params objectForKey:key]];

    }

    ////添加分界线，换行
    [body appendFormat:@"%@\r\n",preBoundary];
    //声明pic字段，文件名为boris.png
    [body appendFormat:@"Content-Disposition: form-data; name=\"%@\";
filename=\"%@\"\r\n",fieldName,fileName];
    //声明上传文件的格式
    [body appendFormat:@"Content-Type: %@\r\n\r\n",fileContentType];

    //声明结束符：--wxk--
    NSString *end=[[NSString alloc]initWithFormat:@"\r\n%@",endBoundary];
    //声明myRequestData，用来放入http body


    NSMutableData *myRequestData=[NSMutableData data];
    //将body字符串转化为UTF8格式的二进制
    [myRequestData appendData:[body
dataUsingEncoding:NSUTF8StringEncoding]];
    //将image的data加入
    [myRequestData appendData:fileData];
    //加入结束符--wxk--
    [myRequestData appendData:[end dataUsingEncoding:NSUTF8StringEncoding]];




    return myRequestData;
}
```

拼接出请求体后，就可以发送数据了。下面使用微博上传图片的接口测试：

```
https://upload.api.weibo.com/2/statuses/upload.json
```

注意该接口需要我们填写至少三个参数在请求体中：1.access_token 用户token 2.status 用户发送的微博 3.data 微博图片。

代码如下：

```
NSURL *url = [NSURL
URLWithString:@"https://upload.api.weibo.com/2/statuses/upload.json"];

double timeout = 100;

NSMutableURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:url
cachePolicy:NSURLRequestUseProtocolCachePolicy timeoutInterval:timeout];

[request setHTTPMethod:@"POST"];


//分隔符
NSString *boundary = @"---wxk---";


//设置请求头
//设置HTTPHeader中Content-Type的值
NSString *content=[[NSString alloc]initWithFormat:@"multipart/form-data;
boundary=%@",boundary];
//设置HTTPHeader
[request setValue:content forHTTPHeaderField:@"Content-Type"];



//请求体
NSData *data = [NSData dataWithContentsOfFile:[[NSBundle mainBundle]
pathForResource:@"PS_iOS" ofType:@"jpg"]];


NSDictionary *params = @{@"access_token":kWB_Token,
                                        @"status":@"upload session test"};
NSData *httpBody = [ViewController httpFormDataBodyWithBoundary:boundary
params:params fieldName:@"pic" fileName:@"PS_iOS.jpg"
fileContentType:@"image/png" data:data];


//设置Content-Length
[request setValue:[NSString stringWithFormat:@"%ld", [httpBody length]]
forHTTPHeaderField:@"Content-Length"];



NSURLSessionUploadTask * upload_task = [session
uploadTaskWithRequest:request fromData:httpBody completionHandler:^(NSData
*data, NSURLResponse *response, NSError *error) {

    NSLog(@"%@",response);

}];
[upload_task resume];
```

2.3.2 上传进度

上传进度也需要使用代理方法：

```
//发送数据时持续调用
- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
                                 didSendBodyData:(int64_t)bytesSent
                                  totalBytesSent:(int64_t)totalBytesSent
                        totalBytesExpectedToSend:
(int64_t)totalBytesExpectedToSend;

//发送完成
- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
                           didCompleteWithError:(NSError *)error;
```

2.4 下载之断点续传

首先暂停任务时使用如下的方法：

```
[self.resumableTask cancelByProducingResumeData:^(NSData *resumeData) {
    // 如果是可恢复的下载任务，应该先将数据保存到partialData中，注意在这里不要调用
cancel方法
    self.partialData = resumeData;
    self.resumableTask = nil;
}];
```

这里我们使用 self.partialData 保存了当前数据，在需要继续下载时使用如下方法：

```
        self.resumableTask = [self.currentSession
downloadTaskWithResumeData:self.partialData];
```

在继续下载时，回回调以下代理方法：

```
- (void)URLSession:(NSURLSession *)session
      downloadTask:(NSURLSessionDownloadTask *)downloadTask
 didResumeAtOffset:(int64_t)fileOffset
expectedTotalBytes:(int64_t)expectedTotalBytes {

    NSLog(@"NSURLSessionDownloadDelegate: Resume download at %lld",
fileOffset);

}
```

2.5 后台任务

后台任务需要我们创建不同的Session用于后台对话：

```
  NSURLSessionConfiguration *config = [NSURLSessionConfiguration
backgroundSessionConfigurationWithIdentifier:kBackgroundSessionID];
  backgroundSess = [NSURLSession sessionWithConfiguration:config
delegate:self delegateQueue:nil];
  backgroundSess.sessionDescription = kBackgroundSession;
```

需要注意的是应用程序之间可以共享Identifier，所以需要注意Identifier的唯一性，确保其他的应用程序不会干扰到我们的程序。

在切到后台之后，Session的Delegate不会再收到，Task相关的消息，直到所有Task全都完成后，系统会调用ApplicationDelegate的application:handleEventsForBackgroundURLSession:completionHandler:回调，之后“汇报”下载工作，对于每一个后台下载的Task调用Session的Delegate中的URLSession:downloadTask:didFinishDownloadingToURL:（成功的话）和URLSession:task:didCompleteWithError:（成功或者失败都会调用）。

处理的操作如下：

```
//App的Delegate

- (void)application:(UIApplication *)application
handleEventsForBackgroundURLSession:(NSString *)identifier

  completionHandler:(void (^)())completionHandler

{

      self.backgroundSessionCompletionHandler = completionHandler;

}

@end



//Session的Delegate

@implementation ViewController

- (void)URLSessionDidFinishEventsForBackgroundURLSession:(NSURLSession
*)session

{

    APLAppDelegate *appDelegate = (APLAppDelegate *)[[UIApplication
sharedApplication] delegate];

    if (appDelegate.backgroundSessionCompletionHandler) {

        void (^completionHandler)() =
appDelegate.backgroundSessionCompletionHandler;

        appDelegate.backgroundSessionCompletionHandler = nil;

        completionHandler();

    }

    NSLog(@"All tasks are finished");

}
```

还有一个情况是当应用程序处于后台下载时，用户杀死了程序，这时之前在NSURLSessionConfiguration设置的NSString类型的ID起作用了，当ID相同的时候，一旦生成Session对象并设置Delegate，马上可以收到上一次关闭程序之前没有汇报工作的Task的结束情况（成功或者失败）。但是当ID不相同，这些情况就收不到，因此为了不让自己的消息被别的应用程序收到，或者收到别的应用程序的消息，起见ID还是和程序的Bundle名称绑定上比较

好，至少保证唯一性。

3、 iOS 9 更新内容

3.1 iOS9更新之——HTTPS

3.2 iOS9更新之——NSURLSessionStreamTask