



xiaoweifive的博客

[目录视图](#)[摘要视图](#)[RSS 订阅](#)

个人资料



小伟Five1993

访问: 2527次

积分: 61

等级: [BLOG > 1](#)

排名: 千里之外

原创: 0篇 转载: 25篇

译文: 0篇 评论: 0条

文章搜索



文章分类

[iOS 感应器 \(5\)](#)[iOS 数据持久化 \(6\)](#)[iOS http的get和post请求 \(3\)](#)

文章存档

[2016年01月 \(1\)](#)[2015年08月 \(1\)](#)[2015年07月 \(3\)](#)[2015年06月 \(20\)](#)

阅读排行

[AFNetworking 2.5 \(269\)](#)[iOS 数据持久化三 - NSL \(145\)](#)[iOS 数据持久化四 - SQL \(134\)](#)[iOS-百度地图API用点生 \(133\)](#)[iOS 数据持久化一 - 属性 \(126\)](#)[AFNetworking 2.0 \(112\)](#)[iOS定位原理和使用建议 \(112\)](#)[关于MKMapView - 地图 \(109\)](#)[iOS 数据持久化五 - Core \(105\)](#)[iOS开发系列-地图与定位 \(102\)](#)[【免费公开课】Gulp前端自动化教程](#) [【专家问答】曹钟岩: 探讨C#---新人入门到实战开发](#) [【博客活动】有奖征文--走进VR开发世界](#)

AFNetworking 2.5

2015-06-24 14:55

269人阅读

[评论\(0\)](#)[收藏](#)[举报](#)[目录\(?\)](#)[\[+\]](#)

文档版本

• 2015.3.19 - 创建文档

使用

AFNetworking有2套用于网络操作的API:

- 基于NSURLConnection
- 基于NSURLSession, 要求iOS 7以上版本

通过配置 [CocoaPods subspecs](#) 可挑选需要的模块而无需使用整个AFNetworking。

本文讲解基于NSURLSession的新接口。一个返回数据为JSON格式的HTTP GET请求, 最简单的编程步骤为:

1. 创建AFHTTPSessionManager, 如 `AFHTTPSessionManager *manager = [AFHTTPSessionManager manager]`。
2. 调用manager的 `GET:parameters:success:failure:` 方法进行GET操作, POST、PUT、DELETE操作类似。
 - GET: 请求的URL字符串, 如 `@("http://www.mySite.com/news")`, `http://` 不可省略
 - parameters: 提交给服务器的参数字典, 如 `@{"time": @"2015-3-22"}`
 - success: 请求成功后调用的block, 接受两个参数: 表示请求的task及服务器返回的响应 `responseObject`, 响应的数据类型在默认情况下被AFURLResponseSerialization解析成 Foundation数据类型, 如 `NSDictionary`
 - failure: 请求失败或服务器响应数据解析失败时调用的block, 接受两个参数: 表示请求的task及错误信息 `error`, 可读取 `error` 中描述, 以便定位问题。

需要说明的是, 在 `[AFHTTPSessionManager manager]` 的内部实现中, AFNetworking使用了`[NSURLSessionConfiguration defaultSessionConfiguration]` 配置它, AFNetworking源码如下

```
// AFURLSessionManager.m
-----
- (instancetype)initWithSessionConfiguration:(NSURLSessionConfiguration *)configuration {
    //...
    if (!configuration) {
        configuration = [NSURLSessionConfiguration defaultSessionConfiguration];
    }
    //...
}
```

[关闭](#)

若要对NSURLSessionConfiguration作更多配置, 则需创建NSURLSessionConfiguration, 然后通过 `initWithBaseURL:sessionConfiguration:` 方式创建AFHTTPSessionManager的SessionConfiguration是只读属性, 以语法得到的对象在正常情况下不可修改其sessionConfiguration属性

下面给出创建NSURLSessionConfiguration并配置HTTP头的示例代码

```
NSURLSessionConfiguration *conf = [NSURLSessionConfiguration
conf.HTTPAdditionalHeaders = @{@"MyHeader": @"MyHeaderValue"}];
[[AFHTTPSessionManager alloc] initWithBaseURL:url sessionCo
```

在应用中, 若都访问同一个基站点, 如 `@("http://www.mySite.com/se` 的POST、GET操作只需加上file部分, 如访问 `http://www.mySite.co`

**澳大利亚买房**

评论排行

- 关于各种项目的SVN 版本 (0)
- iOS 数据持久化——属性 (0)
- iOS 数据持久化三——NSL (0)
- iOS定位原理和使用建议 (0)
- 关于Core Location——ios (0)
- 关于CoreLocation —— 地理 (0)
- 关于MKMapView ——地图 (0)
- CoreMotion框架-iOS设备 (0)
- iOS http get和post(3) (0)
- iOS http get和post(2) (0)

推荐文章

- *Android内存泄露检测工具---LeakCanary的前世今生
- *通过Android源码分析再探观察者模式(二)
- *浅析ZeroMQ工作原理及其特点
- *Rebound-Android的弹簧动画库
- *大型网站架构系列: 缓存在分布式系统中的应用 (二)
- *Hadoop中Map端shuffle源码解析

```
[manager POST:@"menus" parameters:@{@"tableName": @"vip-x"} success:nil failure:nil];
```

若服务器返回的响应数据不是JSON，则需要配置响应序列化器，默认情况下SessionManager的responseSerializer属性值为AFJSONResponseSerializer，且当Content-Type为"application/json"、"text/json"、"text/JavaScript"时，AFNetworking才解析JSON数据。即便服务器端返回的数据为JSON数据，但是Content-Type是"text/plain"，它也不处理，同时产生code=-1016错误。此时给acceptableContentTypes加上服务器返回的类型即可。参见AFNetworking源码：

```
@implementation AFJSONResponseSerializer
-----
- (instancetype)init {
    //...
    self.acceptableContentTypes = [NSSet initWithObjects:@"application/json", @"text/json",
    @"text/javascript", nil];
    //...
}
```

同样，根据业务需要，还可配置requestSerializer属性，指派不同的请求序列化器。

小结上述补充，得到更详细的编程步骤：

1. 创建NSURLSessionConfiguration实例，可配置额外HTTP信息
2. 创建需要的SessionManager，如AFHTTPSessionManager，可配置其requestSerializer responseSerializer属性。
 - 配置请求序列化器，如 [AFHTTPRequestSerializer serializer]
 - 配置响应序列化器，如 [AFHTTPResponseSerializer serializer]
 - 配置响应序列化器的可接受内容类型acceptableContentTypes，如 acceptableContentTypes = [NSSet initWithObjects:@"text/plain", @"text/html", nil]
3. 开始GET、POST、PUT、DELETE等请求，这些请求都返回NSURLSessionTask子类实例，拥有暂停、取消等操作。

获取普通网页文本

```
AFHTTPSessionManager *manager = [[AFHTTPSessionManager manager];
manager.responseSerializer = [AFHTTPResponseSerializer serializer];
manager.responseSerializer.acceptableContentTypes = [NSSet initWithObjects:@"text/html",
@"text/javascript", nil];
[manager GET:@"http://www.baidu.com" parameters:nil success:^(NSURLSessionDataTask *task, id
responseObject) {
    NSLog(@"HTML: %@", [[NSString alloc] initWithData:responseObject
encoding:NSUTF8StringEncoding]);
} failure:^(NSURLSessionDataTask *task, NSError *error) {
    NSLog(@"visit error: %@", error.description);
}];
```

获取JSON数据

请求JSON数据时，因AFHTTPSessionManager默认的responseSerializer为JSON解析类型，则无需额外配置，直接GET请求即可。

```
NSURL *baseUrl = [NSURL URLWithString:@"http://localhost/"];
AFHTTPSessionManager *manager=[AFHTTPSessionManager alloc] initWithBaseURL:baseUrl
sessionConfiguration:config];
NSDictionary *params=[[NSDictionary alloc] initWithObjectsAndKeys:@"8",@"id",nil];
[manager GET:@"json" parameters:params success:^(NSURLSessionDataTask *task, id
responseObject) {
    NSDictionary *object=(NSDictionary *)responseObject;
    NSLog(@"response message: %@",object[@"message"]);
} failure:^(NSURLSessionDataTask *task, NSError *error) {
    NSLog(@"visit error: %@", error);
}];
```

[关闭](#)

下载图片并写入文件

```
responseSerializer = [AFImageResponseSerializer serializer];
UIImage *secImg = responseObject;
NSData *pngData = UIImageJPEGRepresentation(img, 0.4);
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask,
YES);
NSString *filePathName =[[paths objectAtIndex:0] stringByAppendingPathComponent:str];
[pngData writeToFile:filePathName atomically:YES];
```

下载文件

可通过创建NSMutableURLRequest实例并配置下载方式为GET或POST。

```
NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration
defaultSessionConfiguration];
AFURLSessionManager *manager = [[AFURLSessionManager alloc]
initWithSessionConfiguration:configuration];
NSURL *URL = [NSURL URLWithString:@"http://www.baidu.com/img/bdlogo.png"];
NSURLRequest *request = [NSURLRequest requestWithURL:URL];
NSURLSessionDownloadTask *downloadTask = [manager downloadTaskWithRequest:request progress:nil
destination: ^NSURL * (NSURL *targetPath, NSURLResponse *response) {
    NSURL *documentsDirectoryURL = [[NSFileManager defaultManager]
URLForDirectory:NSDocumentDirectory inDomain:NSUserDomainMask appropriateForURL:nil create:NO
error:nil];
    return [documentsDirectoryURL URLByAppendingPathComponent:[response suggestedFilename]];
} completionHandler:^(NSURLResponse *response, NSURL *filePath, NSError *error) {
    NSLog(@"File downloaded to: %@", filePath);
}];
[downloadTask resume];
```

NSURLSessionDownloadTask可用于网络请求的取消、暂停和恢复。

默认下载操作是挂起的，必须手动恢复下载，所以发送 `[NSURLSessionDownloadTask resume]`。

下载大文件

下载大文件时，最好直接从流写到文件中，尽量不要加载到内存，否则在高速网络的情况下，我们的峰值内存压力从而崩溃。这个问题在使用AFHTTPRequestOperation会出现，解决办法是配置其 `outputStream` 属性，如

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask,
YES);
NSString *path = [[paths objectAtIndex:0] stringByAppendingPathComponent:name];
operation.outputStream = [NSOutputStream outputStreamToFileAtPath:path append:NO];
```

而AFURLSessionManager是直接写入文件，不存在上述问题。

```
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:candidateURL];
// 设置HTTP请求方法为HEAD，下载网络请求的头文件，这样可在下载前得到文件的大小信息。
[request setHTTPMethod:@"HEAD"];
// 读取下载任务的countOfBytesExpectedToReceive属性得到预估需下载的字节数
[NSURLSessionDownloadTask countOfBytesExpectedToReceive]
```

参考：

[calculating total progress of downloading multiple file with AFNetworking](#)

批量下载

当提交多个下载任务时，即批量下载，在AFURLSessionManager没提供批量任务完成的通知机制，但AFURLConnectionOperation提供了 `[batchOfRequestOperations:progressBlock:completionBlock:]` 方法。另外，也可将任务放到同一个GCD Dispatch Group中，但是，放到GCD中的任务不可取消。

分段断点下载（多线程断点下载）

像迅雷一样，同一个文件分成多个多线程下载，最后合成一个文件。所有的关键设置全在NSMutableRequest中，参考代码：

```
NSMutableURLRequest *request=[[NSMutableURLRequest alloc] initWithURL:[NSURL
URLWithString:@"yourURL"] cachePolicy:NSURLRequestReloadIgnoringLocalAndRemoteCacheData
timeoutInterval:10];
[request setHTTPMethod:@"GET"];
[request addValue:[NSString stringWithFormat:@"bytes=%llu-", yourDownloadedBytes]
forHTTPHeaderField:@"Range"];
```

每段下载后需要合并文件，由NSFileHandle处理，它可随机写入，参考代码：

```
NSFileHandle *file=[NSFileHandle fileHandleForWritingAtPath:yourFilePath];
if (file) {
    [file seekToFileOffset:yourOffset];
    [file writeData:yourDownloadedData];
    [file closeFile];
}
```

关闭

参考：

[IMIDDownloader](#)

[AFNetworking实现程序重新启动时的断点续传](#)

上传文件

```
NSMutableURLRequest *request =
[[AFHTTPRequestSerializer serializer] multipartFormRequestWithMethod:@"POST"
URLString:@"http://localhost/upload" parameters:nil]
```

```

constructingBodyWithBlock:^(id<AFMultipartFormData> formData) {
    [formData appendPartWithFileURL:uploadFilePath
                      name:@"file"
                      fileName:@"filename.jpg"
                      mimeType:@"image/jpeg"
                      error:nil];

} error:nil];

AFURLSessionManager *manager =
[[AFURLSessionManager alloc]
 initWithSessionConfiguration:[NSURLSessionConfiguration defaultSessionConfiguration]];
NSProgress *progress = nil;
NSURLSessionUploadTask *uploadTask =
[manager uploadTaskWithStreamedRequest:request
                      progress:&progress
                      completionHandler:^(NSURLResponse *response, id responseObject,
                                          NSError *error) {
    if (error) {
        NSLog(@"Error: %@", error);
    } else {
        NSLog(@"%@ %@", response, responseObject);
    }
}];

[uploadTask resume];

```

NSProgress获取上传进度

UIKit拓展

之前 AFNetworking 中的所有 UIKit category 都被保留并增强，还增加了一些新的 category。

加载图片

```

#import <UIImageView+AFNetworking.h>
NSURL *URL = [NSURL URLWithString:@"http://www.baidu.com/img/bdlogo.png"];
[uiImageViewInstance setImageWithURL:URL];

```

其余加载图片方法

```

- setImageWithURL:
- setImageWithURL:placeholderImage:
- setImageWithURLRequest:placeholderImage:success:failure:
- cancelImageRequestOperation

```

AFNetworkActivityIndicatorManager

在请求操作开始、停止加载时，自动开始、停止状态栏上的网络活动指示图标。

UIImageView+AFNetworking

显示图片前剪裁或者加滤镜的功能。增加了 imageResponseSerializer 属性，可以轻松地向 image view 上的图像自动调整大小或应用滤镜。比如，AFCoreImageSerializer 可以在 response 的图像显示之前应用 Core Image filter。

UIButton+AFNetworking

类似 UIImageView+AFNetworking，从远程资源加载 image 和 backgroundImage。

UIActivityIndicatorView+AFNetworking

根据指定的请求操作和会话任务的状态自动开始、停止 UIActivityIndicatorView。

UIProgressView+AFNetworking

自动跟踪某个请求或会话任务的上传/下载进度。

UIWebView+AFNetworking

支持网络请求的进度和内容转换。

关闭

网络监控

AFNetworkReachabilityManager可以单独使用，很方便，用于监控网络变化。
比如，可以在App启动后执行下面操作，启动监控器：

```

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    [[AFNetworkReachabilityManager sharedManager] startMonitoring];
}

```

在ViewController中：

```

- (void)viewDidLoad {

```

```
[[AFNetworkReachabilityManager sharedManager]
 setReachabilityStatusChangeBlock:^(AFNetworkReachabilityStatus status) {
     NSLog(@"Reachability: %@", AFStringFromNetworkReachabilityStatus(status));
     });
}
```

监听网络变化，做出相应的操作，比如弹出提示框。

几种网络状态

- AFNetworkReachabilityStatusUnknown = -1, // 未知
- AFNetworkReachabilityStatusNotReachable = 0, // 无连接
- AFNetworkReachabilityStatusReachableViaWWAN = 1, // 3G 花钱
- AFNetworkReachabilityStatusReachableViaWiFi = 2, // WiFi

常见问题

code=-1005

端口或网络断开连接，可尝试查看端口是否可用及重启模拟器。

code=-1016

acceptableContentTypes缺少支持的项，在AFURLResponseSerialization.h中搜索

```
self.acceptableContentTypes
```

或直接配置实例，加上缺少的项，如@"text/html",@"text/plain"。

code=3840

```
Error Domain=NSCocoaErrorDomain Code=3840 "The operation couldn't be completed. (Cocoa error 3840.)" (JSON text did not start with array or object and option to allow fragments not set.)
UserInfo=0x9152780 {NSDebugDescription=JSON text did not start with array or object and option to allow fragments not set.}
```

添加语句 `manger.responseSerializer = [AFHTTPResponseSerializerserializer];` 即可。

编码问题

假设服务器返回a，客户端收到<61>。当用浏览器去请求时发现响应头 `Content-Type: text/html;charset=UTF-8`，但是，用AFNetwork请求时为 `Content-Type:text/plain;charset=ISO-8859-1`。

无需修改AFURLResponseSerialization.h，只需修改manager的序列化配置即可，如

```
manager.requestSerializer = [AFHTTPRequestSerializerserializer];
manager.responseSerializer = [AFHTTPResponseSerializerserializer];
```

然后，把收到的responseObject转换一下编码

```
NSString *correctResponseObject = [[NSString alloc] initWithData:responseObject
encoding:NSUTF8StringEncoding];
```

2.0的一些变化

AFNetworking 2.0 的目标是调整原始设计中的一些奇怪的地方，同时添加强大的新架构，帮助新一代的应用程序变得更为强大。

Rocket技术

AFNetwork 2.0 遵循 Rocket 技术，Rocket 是在现有的 REST 服务器之上，通过一些 Web 标准（如 `Server-Sent Events`，`JSON Patch`），实现实时的数据更新。

模块化

AFNetworking 1.0 被批评的一个地方是，它有点臃肿。其实 1.0 在类的层次上很具有模块化，但文件封装的不够方便，没办法单独分离出某个功能模块。随着时间的推移，特别是像 AFHTTPClient 这样的类，会显得很臃肿（创建请求，序列化请求参数，响应和解析，创建和管理各种操作，监控网络的可用性等都堆在一起）。

关闭

兼容 NSURLSession

在 iOS7 中 NSURLConnection 被 NSURLSession 取代，但 NSURLConnection 并没有被 deprecated，在一段时间内依然可用。不过，NSURLSession 才是未来，它解决了 NSURLConnection 的很多缺点。有人可能会说有 NSURLSession 还需要 AFNetworking 么，二者确实有重叠的地方，但 AFNetworking 作为一个更上层的抽象类，能提供的更多。2.0 兼容并扩展了 NSURLSession，铺平其中艰难的路线，最大限度的提高了易用性。

1.0迁移2.0

[《AFNetworking 2.0 迁移指南》] (<https://github.com/AFNetworking/AFNetworking/wiki/AFNetworking-2.0-Migration-Guide>)

提高

NSURLConnection

- AFURLConnectionOperation - 它继承于 NSOperation，负责管理 NSURLConnection，实现它的 delegate 方法。
- AFHTTPRequestOperation - 它继承于 AFURLConnectionOperation，专门用于创建 HTTP 请求。2.0 的主要区别就是可以直接使用它，而不用再继承它，原因将会在下文的 Serialization 处解释。
- AFHTTPRequestOperationManager - 封装 HTTP 请求的常见方式，GET / POST / PUT / DELETE / PATCH

NSURLSession

- AFURLSessionManager - 创建和管理 NSURLSession 对象，以及此对象的数据和下载/上传等任务，实现对象和任务的代理方法。NSURLSession 的 API 本身有一些局限，AFURLSessionManager 能使其变得更好用。

AFHTTPSessionManager - 它继承于 AFURLSessionManager，封装了 HTTP 请求的常见方式，GET / POST / PUT / DELETE / PATCH

总结

为了支持最新的 NSURLSession 接口，同时兼容旧的 NSURLConnection，2.0 的核心组件将“网络请求”和“任务处理”分离。AFHTTPRequestOperationManager 和 AFHTTPSessionManager 提供相似的功能 切换很方便，所以从 **iOS 6** 移植到 iOS 7 会很容易。之前绑在 AFHTTPClient 里的 serialization、security、reachability 模型都被分离了出来，基于 NSURLSession 和 NSURLConnection 的 API 都可复用此模型

序列化

2.0 架构的一个突破就是，请求和解析的可序列化。序列化的灵活性允许在网络层添加更多的商业逻辑，并更容易定制之前内置的默认行为。

- 符合这个协议的对象用于处理请求，它将请求参数转换为 query string 或是 entity body 的形式，并设置必要的 header。那些不喜欢 AFHTTPClient 使用 query string 编码参数的家伙，
- 符合这个协议的对象用于验证、序列化响应及相关数据，转换为有用的形式，比如 JSON 对象、图像、甚至基于 Mantle 的模型对象。相比没完没了地继承 AFHTTPClient，现在 AFHTTPRequestOperation 有一个 responseSerializer 属性，用于设置合适的 handler。同样的，再也没有没用的受 NSURLProtocol 启发的 request operation 类注册，取而代之的还是很棒的 responseSerializer 属性。

安全

AFNetworking 支持 **SSL pinning**。这对涉及敏感数据的 App 很重要。

AFSecurityPolicy - 这个类通过特定证书和公共密钥评估链接的安全性和可信任度。在你的 App bundle 中添加服务器证书有助于防止“中间人攻击”。

可达性

另一个从 AFHTTPClient 中分离的功能是网络的可达性。现在你可以单独使用它，或者作为 AFHTTPRequestOperationManager / AFHTTPSessionManager 的一个属性来使用。

AFNetworkReachabilityManager - 负责监控当前的网络可达性，当网络的可达性发生改变时，提供相应的 callback 和通知。

实时

AFEventSource - 用 Objective-C 实现的 EventSource DOM API。客户端和服务端建立持久 HTTP 连接，服务器会把新的 Event 实时推给客户端。客户端收到的信息格式是 **JSON Patch**，然后 JSON Patch 被转化为 AFJSONPatchOperation 对象的数组。可以将这些 patch operation 应用到之前从服务器获取的持久性数据集。示例代码参考：

```
NSURL *URL = [NSURL URLWithString:@"http://example.com"];
AFHTTPSessionManager *manager = [[AFHTTPClient alloc] initWithBaseURL:URL];
[manager GET:@"resources" parameters:nil success:^(NSURLSessionResponse *response, id responseObject) {
    [resources addObjectsFromArray:responseObject[@"resources"]];
    [manager SUBSCRIBE:@"resources" usingBlock:^(NSArray *operations, NSError *error) {
        for (AFJSONPatchOperation *operation in operations) {
            switch (operation.type) {
                case AFJSONAddOperationType:
                    [resources addObject:operation.value];
                    break;
                default:
                    break;
            }
        }
    } error:nil];
} failure:nil];
```

[关闭](#)

缓存策略

NSURLRequest 默认的缓存策略是 NSURLRequestUseProtocolCachePolicy，网络请求是否用缓存是由 HTTP Cache-Control 决定，而在实际开发中由于种种原因(服务端为了简化系统等)，接口的缓存时间都设置的非常长或

者不准，这种情况下就会出现服务端数据更新但是 AFN 拿到的还是旧数据，因为他直接读的缓存。

得益于 AFN 优秀的架构设计，这个问题也很好解决，继承 AFHTTPClient 然后重写

```
requestWithMethod:path:parameters::  
  
- (NSMutableURLRequest *)requestWithMethod:(NSString *)method path:(NSString *)path parameters:  
(NSDictionary *)parameters  
{  
    NSMutableURLRequest *request = [super requestWithMethod:method path:path  
parameters:parameters];  
    [request setCachePolicy:NSURLRequestReloadIgnoringLocalCacheData];  
  
    return request;  
}
```

Response 类型判断

以 AFJSONRequestOperation 为例，只有 Content-Type 是 @"application/json", @"text/json",
@"text/javascript" 或 URL pathExtension 是 json 的才会被认为是 JSON 类型，其他都不认。很多服务端接口都没有 Content-Type 返回或者直接丢一个 text/html，请求也不是 json 结尾，但返回内容确实是 JSON 数据，这时候 AFN 就很无力。

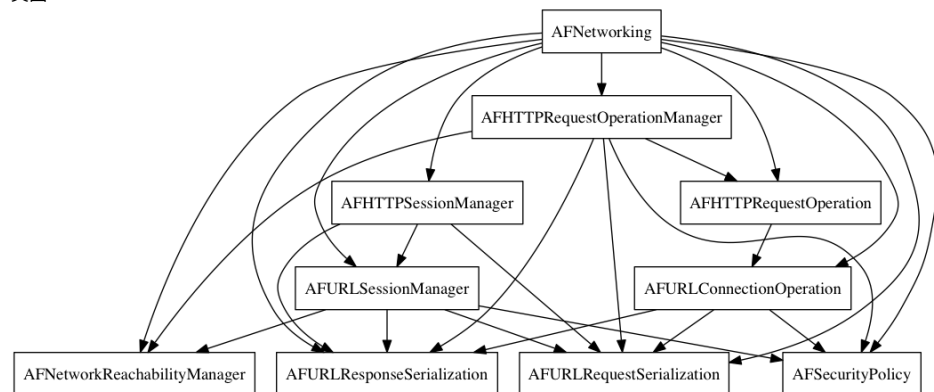
URL加载系统

NSURLConnection 是 Foundation URL 加载系统的基石。一个 NSURLConnection 异步地加载一个 NSURLRequest 对象，调用 delegate 的 NSURLConnection / NSHTTPURLResponse 方法，其 NSData 被发送到服务器或从服务器读取；delegate 还可用来处理 NSURLConnectionChallenge、重定向响应、或是决定 NSCachedURLResponse 如何存储在共享的 NSURLConnection 上。

NSOperation 是抽象类，模拟单个计算单元，有状态、优先级、依赖等功能，可以取消。

源码阅读

类图



增强模块

AFNetworking-Extensions

同类型第三方库

- [STHTTPRequest](#) - 基于 NSURLConnection，支持 synchronous+asynchronous blocks，支持文件上传，非常简单轻量的封装，值得一试。

引伸话题

MVCNetworking

参考

- [AFNetworking 2.0](#)
- [What I Learned From AFNetworking's GitHub Issues](#)
- [What I Learned From AFNetworking's GitHub Issues](#)视频
- [AFNetwork 2.0在请求时报错code=-1016 和 3840](#)
- [使用AFNetworking, SDWebImage和OHHTTPStubs](#)
- [AFNetworking 2.0 来了](#)
- [AFNetworking 学习笔记](#)
- [AFNetworking 学习笔记二](#)
- [AFNetworking2.0源码解析 1](#)
- [Clang Diagnostics](#)
-

关闭

顶 踩
0 0

上一篇 [iOS开发系列--Objective-C之KVC、KVO](#)

下一篇 [AFNetworking 2.0](#)

参考知识库



JavaScript知识库

1456 关注 | 500 收录



jQuery知识库

268 关注 | 232 收录



大型网站架构知识库

975 关注 | 532 收录



AngularJS知识库

484 关注 | 262 收录



Swift知识库

1692 关注 | 386 收录

猜你在找

[从零练就iOS高手实战班](#)

[Android开发精品课程【Java核心知识】](#)

[老郭全套iOS开发课程【UI技术】](#)

[基于Spring MVC+MyBatis+FreeMarker+OSCache网上商城](#)

[PDF神器-Adobe Acrobat Pr](#)

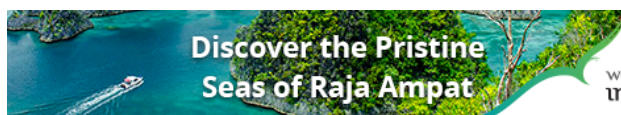
[ARC相关](#)

[Receiver type X for instance message is a forward](#)

[手把手教你ARCiOSMac开发ARC入门和使用](#)

[iOS9适配系列教程](#)

[iOS9的新特性](#)



wonderful
indonesia

Waigeo Island,
Raja Ampat
EXPLORE NOW!
www.indonesia.travel

[查看评论](#)

暂无评论

[发表评论](#)

用户 名: ALDRIDGE1

评论内容:

提交

关闭


* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

[全部主题](#) [Hadoop](#) [AWS](#) [移动游戏](#) [Java](#) [Android](#) [iOS](#) [Swift](#) [智能硬件](#) [Docker](#) [OpenStack](#)
[VPN](#) [Spark](#) [ERP](#) [IE10](#) [Eclipse](#) [CRM](#) [JavaScript](#) [数据库](#) [Ubuntu](#) [NFC](#) [WAP](#) [jQuery](#)

BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC
coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap

[公司简介](#) | [招贤纳士](#) | [广告服务](#) | [银行汇款帐号](#) | [联系方式](#) | [版权声明](#) | [法律顾问](#) | [问题报告](#) | [合作伙伴](#) | [论坛反馈](#)

[网站客服](#) [杂志客服](#) [微博客服](#) webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持
京 ICP 证 09002463 号 | Copyright © 1999-2014, CSDN.NET, All Rights Reserved 

关闭

