AFNetworking 3.0 Migration Guide

Jump to bottom

Jeff Kelley edited this page Nov 23, 2017 · 19 revisions

AFNetworking 3.0 is the latest major release of AFNetworking, a delightful networking library for iOS, macOS, watchOS, and tvOS. To keep this library maintainable moving forward, 3.0 removes all support for the now deprecated `NSURLConnection` based APIs. If your project was previously using these APIs, it is recommended that you now upgrade to the `NSURLSession` based APIs. This guide will step you through that process.

This guide is provided in order to ease the transition of existing applications using AFNetworking 2.X to the latest APIs, as well as explain the design and structure of new and changed functionality.

New Requirements: iOS 7, Mac OS X 10.9, watchOS 2, tvOS 9, & Xcode 7

AFNetworking 3.0 officially supports iOS 7+, Mac OS X 10.9+, watchOS 2+, tvOS 9, and Xcode 7. If you'd like to use AFNetworking in a project targeting an older SDK version, please check the README for compatibility information.

NSURLConnection APIs Have Been Removed

AFNetworking 1.0 was built on `NSURLConnection` based APIs, and AFNetworking 2.0 provided the option of either using the `NSURLConnection` based APIs, or the newer `NSURLSession` based APIs. AFNetworking 3.0 is now built exclusively on top of the `NSURLSession` based APIs, which lowers the maintenance burden going forward, while allowing for support of any additional features around future enhancements provided by Apple to `NSURLSession`. As of Xcode 7, `NSURLConnection` API has been officially deprecated by Apple. While the API will continue to function, no new features will be added, and Apple has advised all network based functionality to leverage `NSURLSession` going forward.

AFNetworking 2.x will continue to receive critical bug and security fixes, but no new functionality will be added going forward. The Alamofire Software Foundation recommends that all projects migrate to the `NSURLSession` based APIs going forward.

Classes Removed

The following classes have been removed from AFNetworking 3.0:

- `AFURLConnectionOperation`
- `AFHTTPRequestOperation`
- `AFHTTPRequestOperationManager`

Modified Classes

The following classes contained internal implementations based on `NSURLConnection` APIs. They have been refactored to use `NSURLSession` APIs going forward.

- `UIImageView+AFNetworking`
- `UIWebView+AFNetworking`
- `UIButton+AFNetworking`

Migration

AFHTTPRequestOperationManager Based Code

If you were previously using `AFHTTPRequestOperationManager`, you will need to migrate to `AFHTTPSessionManager`. Several components are reused between these classes, including the following:

- `securityPolicy`
- `requestSerializer`
- `responseSerializer`

The following shows a simple example of migration to `AFHTTPSessionManager`. Note that the HTTP verb methods return `NSURLSessionTask` rather than `AFHTTPRequestOperation`, and the success and failure blocks pass in an `NSURLSessionTask` rather than an `AFHTTPRequestOperation`.

AFNetworking 2.x

```
AFHTTPRequestOperationManager *manager = [AFHTTPRequestOperationManager manager
[manager GET:@"http://example.com/resources.json" parameters:nil success:^(AFHT
    NSLog(@"JSON: %@", responseObject);
} failure:^(AFHTTPRequestOperation *operation, NSError *error) {
    NSLog(@"Error: %@", error);
}];
```

AFNetworking 3.x

```
AFHTTPSessionManager *manager = [AFHTTPSessionManager manager];
[manager GET:@"http://example.com/resources.json" parameters:nil progress:nil s
    NSLog(@"JSON: %@", responseObject);
} failure:^(NSURLSessionTask *operation, NSError *error) {
    NSLog(@"Error: %@", error);
}];
```

AFHTTPRequestOperation Based Code

Unlike `NSURLConnection` objects, which each share application-wide settings for session management, cache policies, cookie storage, and URL protocols, `NSURLSession` objects can configure these all individually. Once a session is initialized with a particular configuration, it can dispatch tasks to fetch data, and upload or download files.

With AFNetworking 2.0, leveraging `AFHTTPRequestOperation`, it was possible to create a single request with no additional overhead and fetch remote data. `NSURLSession` requires slightly more overhead, in order to gain access to functionality for the request itself.

Going forward, making a single request will require the creation of an `AFHTTPSessionManager`, followed by creating a task and starting it.

AFNetworking 2.x

```
NSURL *URL = [NSURL URLWithString:@"http://example.com/resources/123.json"];
NSURLRequest *request = [NSURLRequest requestWithURL:URL];
```

```
AFHTTPRequestOperation *op = [[AFHTTPRequestOperation alloc] initWithRequest:re
op.responseSerializer = [AFJSONResponseSerializer serializer];
[op setCompletionBlockWithSuccess:^(AFHTTPRequestOperation *operation, id respo
    NSLog(@"JSON: %@", responseObject);
} failure:^(AFHTTPRequestOperation *operation, NSError *error) {
    NSLog(@"Error: %@", error);
}];
[[NSOperationQueue mainQueue] addOperation:op];
```

AFNetworking 3.x

```
NSURL *URL = [NSURL URLWithString:@"http://example.com/resources/123.json"];
AFHTTPSessionManager *manager = [AFHTTPSessionManager manager];
[manager GET:URL.absoluteString parameters:nil progress:nil success:^(NSURLSess
    NSLog(@"JSON: %@", responseObject);
} failure:^(NSURLSessionTask *operation, NSError *error) {
    NSLog(@"Error: %@", error);
}];
```

Note that `NSURLSession` is not built upon `NSOperation`. If you have an app that heavily relies on the `NSOperation` aspects of `AFURLConnectionOperation`, there may be additional rework needed for your application to wrap `AFHTTPSessionManager` with an `NSOperation` based approach.

Tracking Progress

AFNetworking 3.0 uses `NSProgress` to track progress for NSURLSessionTasks. In addition to the block based convenience methods for upload and download tasks on `AFURLSessionManager` and GET and POST methods on `AFHTTPSessionManager`, `AFURLSessionManager` exposes helper methods to access `NSProgress` objects for each task:

- `uploadProgressForTask:`
- `downloadProgressForTask:`

Using KVO, progress can then be tracked for each task, assuming the response includes a `Content-Length` header. Please see #3187 for additional information around the progress changes in AFNetworking 3.0

UIKit Migrations

Image downloading has been refactored to follow the architecture from AlamofireImage with the new `AFImageDownloader` class. The UIButton and UIImageView categories now delegate image download responsiblities to this class, and each expose a shared reference to a downloader allowing for custom behavior if necessary. The actual methods for downloading remote images from the category interface remain unchanged.

The UIWebView category has been refactored to use a shared `AFHTTPSessionManager` for its network requests.

UIAlertView Category has been removed

The `UIAlertView` category has been removed in AFNetworking 3.0 since it is now deprecated. There are no plans to provide support for `UIAlertController` going forward, as this is logic that should be handled by the application, not this library.

Streamlined Example

The example project has been streamlined to support all available targets, including watchOS, tvOS, and iOS extensions.

Help

If you have any additional questions around migration, feel free to open a documentation issue on Github to get more clarity added to this guide.

▸ Pages 11

AFNetworking

- Getting Started with AFNetworking
- AFNetworking FAQ
- 2.0 Migration Guide
- 3.0 Migration Guide
- AFNetworking Extensions

Clone this wiki locally

```
https://github.com/AFNetworking/AFNetworking.wiki.git
```

Clone in Desktop