NSURLProtocol

Written by Mattt Thompson — November 5th, 2012

iOS is all about networking–whether it's reading or writing state to and from the server, offloading computation to a distributed system, or loading remote images, audio, and video from the cloud.

Because of this, Foundation's URL Loading System is something that every iOS developer would do well to buddy up with.

When given the choice, applications should adopt the highest-level framework possible for what needs to be done. So, if that task is communicating over `http://`, `https://` or `ftp://`, then `NSURLConnection` and friends are a clear choice. Apple's networking classes cover the essentials for modern Objective-C application development, from URL and cache management to authentication & cookie storage:

The URL Loading System

URL Loading	
NSURLConnection	
NSURLRequest	NSMutableURLRequest
NSURLResponse	NSHTTPURLResponse
Cache Management	
NSURLCache	
NSCacheURLRequest	
NSCachedURLResponse	
Authentication & Credentials	
NSURLCredential	

NSURLCredentialStorage
NSURLAuthenticationChallenge
NSURLProtectionSpace
Cookie Storage
NSHTTPCookie
NSHTTPCookieStorage
Protocol Support
NSURLProtocol

Although there's a lot to the URL Loading System, it's designed in a way that hides the underlying complexity, with hooks to provide configuration when needed. Any request going through `NSURLConnection` is intercepted by other parts of the system along the way, allowing for things like cached responses being transparently loaded from disk when available.

Which brings us to this week's topic: `NSURLProtocol`.

`NSURLProtocol` is both the most obscure and the most powerful part of the URL Loading System. It's an abstract class that allows subclasses to define the URL loading behavior of new or existing schemes.

If you aren't already `mindblown.gif`, here are some examples of what this can be used for, *without changing anything else about how requests are loaded*:

- Intercepting HTTP requests to serve images locally from the app bundle resources, if available
- Mocking and stubbing HTTP responses for testing
- Normalizing headers and parameters of outgoing requests
- Signing outgoing streaming media requests
- Creating a proxy server for a local data transformation service with a URL request interface
- Deliberately sending malformed & illegal response data to test the robustness of the application

- Filtering sensitive information from requests or responses
- Implementing an `NSURLConnection`-compatible interface to an existing protocol.

Again, it's important to reiterate that the whole point of `NSURLProtocol` is that you can change everything about the loading behavior of your application without doing anything differently with how your application communicates to the network.

Or, put another way: `NSURLProtocol` is an Apple-sanctioned man-in-the-middle attack.

Subclassing NSURLProtocol

As mentioned previously, `NSURLProtocol` is an abstract class, which means it will be subclassed rather than used directly.

Determining if a Subclass Can Handle a Request

The first task of an `NSURLProtocol` subclass is to define what requests to handle. For example, if you want to serve bundle resources when available, it would only want to respond to requests that matched the name of an existing resource.

This logic is specified in `+canInitWithRequest:`. If `YES`, the specified request is handled. If `NO`, it's passed down the line to the next URL Protocol.

Providing a Canonical Version of a Request

If you wanted to modify a request in any particular way, `+canonicalRequestForRequest:` is your opportunity. It's up to each subclass to determine what "canonical" means, but the gist is that a protocol should ensure that a request has only one canonical form (although many different requests may normalize into the same canonical form).

Getting and Setting Properties on Requests

`NSURLProtocol` provides methods that allow you to add, retrieve, and remove arbitrary metadata to a request object–without the need for a private category or swizzling:

- `+propertyForKey:inRequest:`
- `+setProperty:forKey:inRequest:`
- `+removePropertyForKey:inRequest:`

This is especially important for subclasses created to interact with protocols that have information not already provided by `NSURLRequest`. It can also be useful as a way to pass state between other methods in

your implementation.

Loading Requests

The most important methods in your subclass are `-startLoading` and `-stopLoading`. What goes into either of these methods is entirely dependent on what your subclass is trying to accomplish, but there is one commonality: communicating with the protocol client.

Each instance of a `NSURLProtocol` subclass has a `client` property, which is the object that is communicating with the URL Loading system. It's not `NSURLConnection`, but the object does conform to a protocol that should look familiar to anyone who has implemented `NSURLConnectionDelegate`

`<NSURLProtocolClient>`

- `-URLProtocol:cachedResponseIsValid:`
- `-URLProtocol:didCancelAuthenticationChallenge:`
- `-URLProtocol:didFailWithError:`
- `-URLProtocol:didLoadData:`
- `-URLProtocol:didReceiveAuthenticationChallenge:`
- `-URLProtocol:didReceiveResponse:cacheStoragePolicy:`
- `-URLProtocol:wasRedirectedToRequest:redirectResponse:`
- `-URLProtocolDidFinishLoading:`

In your implementation of `-startLoading` and `-stopLoading`, you will need to send each delegate method to your `client` when appropriate. For something simple, this may mean sending several in rapid succession, but it's important nonetheless.

Registering the Subclass with the URL Loading System

Finally, in order to actually use an `NSURLProtocol` subclass, it needs to be registered into the URL Loading System.

When a request is loaded, each registered protocol is asked "hey, can you handle this request?". The first one to respond with `YES` with `+canInitWithRequest:` gets to handle the request. URL protocols are consulted in reverse order of when they were registered, so by calling `[NSURLProtocol registerClass: [MyURLProtocol class]];` in `-application:didFinishLoadingWithOptions:`, your protocol will have priority over any of the built-in protocols.

Like the URL Loading System that contains it, `NSURLProtocol` is incredibly powerful, and can be used in

exceedingly clever ways. As a relatively obscure class, we've only just started to mine its potential for how we can use it to make our code cleaner, faster, and more robust.

So go forth and hack! I can't wait to see what y'all come up with!

NSMUTABLEHIPSTER

Questions? Corrections? Issues and pull requests are always welcome — NSHipster is made better by readers like you.

Find status information for all articles on the status page.

FOLLOW NSHIPSTER

WRITTEN BY

Mattt Thompson

Mattt Thompson (@mattt) is a writer and developer from the Rustbelt.

NEXT ARTICLE

NSValueTransformer

Of all the Foundation classes, NSValueTransformer is perhaps the one that fared the worst in the shift from OS X to iOS. But you know what? It's ripe for a comeback. With a little bit of re-tooling and some recontextualization, this blast from the past could be the next big thing in your application.

RELATED ARTICLES

- NSDataDetector
- NSCoding / NSKeyedArchiver
- CMDeviceMotion
- NSAssertionHandler

ISSN 2373-9800

文章也可在中文版阅读。