Article

Accessing cached data

Control how URL requests make use of previously cached data.

Overview

The URL Loading System caches responses both in memory and on disk, improving performance and reducing network traffic.

The `URLCache` class is used for caching responses from network resources. Your app can directly access the shared cache instance by using the `shared` property of `URLCache`. Or, you can create your own caches for different purposes, setting distinct caches on your `URLSessionConfiguration` objects.

Set a cache policy for URL requests

Each `URLRequest` instance contains a `URLRequest.CachePolicy` object to indicate if and how caching should be performed. You can change this policy to control caching for the request.

For convenience, `URLSessionConfiguration` has a property called `requestCachePolicy`; all requests created from sessions that use this configuration inherit their cache policy from the configuration.

The behaviors of the various policies are described in doc:accessing-cached-data#Table-1. This table shows the policies' respective preferences for loading from cache or from the originating source, like a server or the local file system. Currently, only HTTP and HTTPS responses are cached. For FTP and file URLs, the only effect of a policy is to determine whether the request is allowed to access the originating source.

Cache policy	Local cache	Originating source
NSURLRequest.CachePolicy.reloadIgnoringLocalCacheData	Ignored	Accessed exclusively
NSURLRequest.CachePolicy.returnCacheDataDontLoad	Accessed exclusively	Ignored
NSURLRequest.CachePolicy.returnCacheDataElseLoad	Tried first	Accessed only if needed
NSURLRequest.CachePolicy.useProtocolCachePolicy	Depends on protocol	Depends on protocol

For an explanation of how `useProtocolCachePolicy` is implemented for HTTP and HTTPS, see `NSURLRequest.CachePolicy`. `useProtocolCachePolicy` is the default value for a `URLRequest` object.

> **Note**
>
> `useProtocolCachePolicy` caches HTTPS responses to disk, which may be undesirable for securing user data. You can change this behavior by manually handling caching behavior, as described in Manage caching programmatically.

Access the cache directly

You can get or set the cache object used by a `URLSession` object by using the `urlCache` property of the session's `configuration` object.

To look for the cached response to a given request, call `cachedResponse(for:)` on the cache. If cached data exists for the request, this call returns a `CachedURLResponse` object; otherwise, it returns `nil`.

You can inspect resources used by the cache. The properties `currentDiskUsage` and `diskCapacity` represent the file system resources used by the cache, and `currentMemoryUsage` and `memoryCapacity` represent memory use.

You can remove cached data for individual items with `removeCachedResponse(for:)`. You can also clear out many cached items simultaneously with `removeCachedResponses(since:)`, which removes cached items past a given date, or `removeAllCachedResponses()`, which wipes the entire cache.

Manage caching programmatically

You can write to the cache programmatically, with the `storeCachedResponse(_:for:)` method, passing in a new `CachedURLResponse` object and a `URLRequest` object.

Typically, you manage the caching of a response while it's being handled by a `URLSessionTask` object. To manage caching on a per-response basis, implement the `urlSession(_:dataTask:willCacheResponse:completionHandler:)` method of the `URLSessionDataDelegate` protocol. Note that this delegate method is called only for uploads and data tasks, and is not called for sessions with a background or ephemeral configuration.

The delegate receives two parameters: a `CachedURLResponse` object and a completion handler. Your delegate *must* call this completion handler directly, passing in one of the following:

- The provided `CachedURLResponse` object, to cache the proposed response as-is
- `nil`, to prevent caching
- A newly created `CachedURLResponse` object, typically based on the provided object, but with a modified `storagePolicy` and `userInfo` dictionary, as you see fit

The following example shows an implementation of `urlSession(_:dataTask:willCacheResponse:completionHandler:)`, which intercepts responses to HTTPS requests and allows the responses to be stored in the in-memory cache only.

Handling the urlSession(_:dataTask:willCacheResponse:completionHandler:) callback

```
func urlSession(_ session: URLSession, dataTask: URLSessionDataTask,
                willCacheResponse proposedResponse: CachedURLResponse,
                completionHandler: @escaping (CachedURLResponse?) -> Void) {
    if proposedResponse.response.url?.scheme == "https" {
        let updatedResponse = CachedURLResponse(response: proposedResponse.response,
                                                data: proposedResponse.data,
                                                userInfo: proposedResponse.userInfo,
                                                storagePolicy: .allowedInMemoryOnly)
        completionHandler(updatedResponse)
    } else {
        completionHandler(proposedResponse)
    }
}
```

See Also

Cache behavior

class CachedURLResponse

A cached response to a URL request.

class URLCache

An object that maps URL requests to cached response objects.