

一人我编程累 Lv1

2020年01月04日 阅读 189

[关注](#)

[iOS] iOS中URLRequest的缓存策略

Cache URLRequest

Example 项目地址: github.com/zColdWater/... 下载Demo, 结合Demo一起实践更容易理解。

一，准备工作

在开始之前，我们需要清楚下面的一些问题，才方便我们后面的讲解。

1. URLRequest 涉及的范围

我们一提到 `URLRequest`，我相信很多国内的开发者，首先就会联想到，`HTTP` 请求，然后木有别的了。但是其实 `URLRequest` 是一个很大的概念，它不只服务于 `HTTP` 协议，它还服务于 其他应用协议，比如 `File` 协议，`Data` 协议，自定义协议等等。要么苹果公司为什么不叫它 `HTTPURLRequest` 呢？问题就在于我们平时最常接触的就是 `HTTP` 协议，用来请求服务端的数据用来展示。

[首页](#) ▾[探索掘金](#)

通过上面的文章我们清楚了 `URLRequest` 服务很多协议，那么 `URLRequest.CachePolicy` 的范围是什么呢，很明显，和 `URLRequest` 一样，这个缓存策略也包含上面这些协议，当然 我也不清楚其他协议的缓存策略是什么样子的，比如 `File` 协议，或则别的。但是我很清楚，我们常用的 `HTTP` 协议的缓存策略，这个后面再讲，这里清楚的是，这个缓存策略支持很多协议，我们的 `HTTP` 协议有着自己的缓存策略。

3. HTTP协议的缓存策略

我之前转了一篇台湾作者关于 `HTTP` 协议的缓存策略的文章，文章地址是：

<http://47.99.237.180:8080/articles/2019/11/18/1574050998351.html> 那么 `HTTP` 协议的缓存策略是什么呢？

Note: 首先我们需要清楚的是，`HTTP` 协议的策略是需要 客户端 和 服务端 配合完成的。也就是如果这个策略想要完成，需要双方都有动作，并且 客户端 需要完全配合才行。

我用最直白的话来概述这个原理：

- 首先 客户端 第一次访问 服务器 某一个资源，并且 服务器 和 客户端 协商好，我们都用标准的 `HTTP` 缓存协议。
- 因为是第一次， 客户端 通过 `URL` 来查找，发现本地没有缓存，直接向服务器发起一个 `HTTP` 协议的网络请求。 客户端 的请求头，例如：

```
GET /EC6/poster-share-invite.png HTTP/1.1
Host: fep-sit.nioint.com:5418
Accept: */*
User-Agent: SessionDownload/1 CFNetwork/1107.1 Darwin/19.0.0
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

这其实就是一个普通的网络请求，请求头也是客户端默认的，没有特殊设置。

- 服务器 接到了某一个 客户端 的发来的请求，然后做的直接就看一下这个请求头有没有提供 `HTTP` 协议缓存的相关字段，然后根据 `HTTP` 协议缓存规则，来判断是否返回 状态码304 让客户端用缓存，还是返回 状态码200，让客户端使用服务器返回的新资源，服务器检查请求头的相关字段如下：

并且这个值是从上一次 服务器 返回的响应头里面的 Etag 字段取来的。因为我们客户端是第一次请求，所以没有从之前的服务器响应里面拿到这个值，所以请求头就没有这个字段。

- If-Modified-Since :Tue, 31 Dec 2019 14:57:28 GMT，这个字段表示的是最后一次资源更改的时间，同 If-None-Match 也是从上一轮的服务器响应头中拿到，从 Last-Modified 字段取的。因为第一次请求，所以没有获取到上一次响应头的字段，也就没有带上。

- 服务器开始根据 HTTP 协议规则进行检查，来决定是让客户端使用缓存还是使用服务器下发的资源。

- 服务器的两种响应头：

1. 状态码200（告诉客户端，不要使用缓存，用我给你的新资源）

```
HTTP/1.1 200 OK
X-Powered-By: Express
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Tue, 31 Dec 2019 14:57:28 GMT
ETag: W/"20f9a-16f5c76370d"
Content-Type: image/gif
Content-Length: 135066
Date: Wed, 01 Jan 2020 01:56:35 GMT
Proxy-Connection: keep-alive
```

2. 状态码304（告诉客户端你用自己的缓存即可）

这里注意的是，在iOS当中，你不需要亲自处理 304 的情况，如果你使用了默认的缓存策略，也就是使用HTTP协议本身的缓存策略，系统的网络框架比如 URLSession 或者 URLConnection 会自动的将这个 304 处理成 200，这样方便了开发者逻辑处理，开发者只需要知道 资源获取成功，就可以了。

```
HTTP/1.1 304 Not Modified
X-Powered-By: Express
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Tue, 31 Dec 2019 14:57:28 GMT
ETag: W/"20f9a-16f5c76370d"
Date: Wed, 01 Jan 2020 01:59:25 GMT
Proxy-Connection: keep-alive
```

- 服务器响应头里与HTTP协议缓存策略相关的字段：

[首页](#)[探索掘金](#)

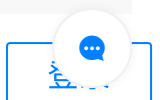
100s之内再去请求，是不会发起真正的网络请求的，客户端的网络层框架会自动返回状态码200，上一次的缓存数据)

2. **Last-Modified** : Tue, 31 Dec 2019 14:57:28 GMT (这个字段是告诉客户端，这个资源最后一次更新的时间，让客户端保存好，下一次请求的时候，在请求头里面带上这个值，请求头的那个字段就是 **If-Modified-Since**，这里的规则是这样的，如果请求头里的 **If-Modified-Since** 时间点早于服务器的 **Last-Modified** 时间点，服务器会返回200，让客户端需要更新最新资源，如果反过来，或者相同，服务器会下发304，让客户端使用缓存。)
 3. **ETag** : W/"20f9a-16f5c76370d" (这个字段告诉客户端，这个值是这个资源的唯一id，如果服务器上面有新资源，我们会更新这个值，客户端要保存好，下次请求的时候带上，下次请求头里面这个 **If-None-Match** 字段就是保存的上次响应头里面的 **Etag** 字段。它的规则是，如果客户端请求头中的 **If-None-Match** 值不与服务器里面的 **Etag** 一致，就返回200，让客户端使用新资源，只有当相等的情况，会返回304，让客户端使用缓存。)
- 服务器检查完请求头发现这个客户端没有带上资源缓存信息，那么服务器就认为客户端不想使用 **HTTP** 协议缓存策略，返回200，把资源也一同返回。

```
HTTP/1.1 200 OK
X-Powered-By: Express
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Tue, 31 Dec 2019 14:57:28 GMT
ETag: W/"20f9a-16f5c76370d"
Content-Type: image/gif
Content-Length: 135066
Date: Wed, 01 Jan 2020 01:56:35 GMT
Proxy-Connection: keep-alive
```

- 客户端第一次拿到资源后，先将服务器告诉自己的缓存信息，保存起来，**Cache-Control: public, max-age=0** 读取一下 **max-age**，发现值等于0，这是告诉我应该每次都发真正的请求，然后再存一下 **Last-Modified** 上次更新的时间，再存一下 **ETag**，这个告诉我们资源的唯一id。
- 然后 **客户端** 开始发起第二次请求，请求头如下：

```
GET /demo.gif HTTP/1.1
Host: 152.136.154.126:3000
If-None-Match: W/"20f9a-16f5c76370d"
Accept: */*
If-Modified-Since: Tue, 31 Dec 2019 14:57:28 GMT
```



```
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

可以看到，客户端将缓存资源的信息带上来了，在 `If-None-Match` , `If-Modified-Since` 。

- **服务器** 再次接到这个请求头，和自己的对资源的信息对比，发现 `If-None-Match` 和 `ETag` 一致的，发现 `Last-Modified` 和 `If-Modified-Since` 一致的，那么客户端的资源就和服务器是一致的，我应该告诉客户端304状态码，让客户端使用缓存就可以了。响应头如下：

```
HTTP/1.1 304 Not Modified
X-Powered-By: Express
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Tue, 31 Dec 2019 14:57:28 GMT
ETag: W/"20f9a-16f5c76370d"
Date: Wed, 01 Jan 2020 03:02:57 GMT
Proxy-Connection: keep-alive
```

自此，一次完整的HTTP协议缓存策略应用完成，我们可以看到，客户端第一次发起请求，第二次发起请求，请求头里面的变化。和服务器如何对请求头做出的校验和响应。

我知道还有 `Expire` 字段，也是缓存相关的，但是HTTP1.1之后设置Cache-Control里面的max-age，可以覆盖它，如果他们同时存在，所以这里不再概述，如果你感兴趣，可以WIKI，或者 <http://47.99.237.180:8080/articles/2019/11/18/1574050998351.html>

希望我可以说明清楚这个过程，因为下面iOS下面的网络框架就会涉及到。这个是HTTP协议的缓存策略，意味着，只要使用HTTP协议的客户端，不管是浏览器，还是移动端，还是其他，都使用。

即使现在不是特别清楚也没关系，我会在文章中附上，整个过程的Demo，可以运行Demo，来了解每一步。

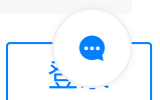
4. 出了一个小插曲

在验证过程中除了一些小插曲，比如我已经为服务器的静态资源，设置资源的过期时间，在响应头当中， `Cache-Control: max-age=xxx` ，发现在移动端上，是OK的，在xxx秒之内再次访问，真的没有发起网络请求，但是在 **Chrome** 当中我发现直接输入资源地址，它好像会忽略过期时间一样，都会发起真正的网络请求，我看了小半天，我终于发现了，这个问题出在哪了，下面我来讲一下。



首页 ▾

探索掘金



1. 移动端下: 使用URLRequest的默认缓存策略, 缓存策略使用协议本身的缓存策略, 设置超时时间工作正常。
2. Web端: 使用 **标签** 或者 **XHR** 发起的网络请求, 设置超时时间工作正常。
3. Web端: 使用浏览器直接访问资源, 发现即使服务器设置了超时时间, 还是会发起网络请求。

总结下来: 只有 **3** 显示的不正常, 问题可能就是因为直接在浏览器里面输入资源地址去访问资源, 忽略超时时间, 可能由于某些原因吧, 所以每次才都会发起网络请求, 而且Chrome和Safari的表现行为还不一样, 所以我们如果想验证HTTP协议的缓存策略其实可以忽略掉 **3**, 验证 **1** 和 **2** 即可。

二, iOS下, URLRequest.CachePolicy和URLCache

上面我们主要介绍了一些事先需要了解的知识, 比如iOS里面的URLRequest和URLRequest.CachePolicy使用范围, 还有最重要, 也是最复杂的 **HTTP** 协议缓存策略。

0. CachePolicy 与 URLCache 的关系

CachePolicy: 顾名思义这个是缓存策略的意思, 我下面会仔细说它, 这里我们知道它代表一种缓存的策略就OK。

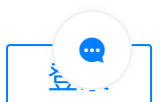
URLCache: 这个其实就是我们缓存的对象, 也就是我们使用了某种缓存策略, 把内容存储的地方和配置存储在哪地方, 等等, 或者说管理URL缓存的对象。

小结: **CachePolicy** 用于选择一种缓存策略, **URLCache** 管理和设置缓存对象。

1. 有哪些地方可以设置 CachePolicy

首先先了解 **CachePolicy** 有哪些策略可选:

- **useProtocolCachePolicy** // 使用协议缓存, 比如你的 **URL** 是 **HTTP** 协议的, 那就是使用 **HTTP** 协议的缓存策略, 就是我上面讲的, 根据请求和响应头的关键字, 进行缓存的处理。这也是(默认的策略)。
- **reloadIgnoringLocalCacheData** // 完全忽略本地缓存, 直接从服务器拿资源。
- **reloadIgnoringLocalAndRemoteCacheData** // 这个实例是未实现的, 你不应该使用
- **reloadIgnoringCacheData** // 这个选项已经被 reloadIgnoringLocalCacheData 选项所替代了。
- **returnCacheDataElseLoad** // 如果本地有缓存就使用缓存, 不管HTTP协议上缓存上的那些max-age, expire 过期时间等, 没有缓存再去远端请求拿数据。这个协议存在的问题就是, 如果使用HTTP协议的缓存策略, 服务器没办法告诉它, 它手头的这份资料是旧的。



- `reloadRevalidatingCacheData` // 这个实例是未实现的，你不应该使用它。

其实苹果给的默认策略，一点毛病都没有，根据HTTP协议的缓存策略来配置是最好的，有缓存使用缓存，没有缓存，或者缓存过期，来请求服务器的资源。也就是你根本毛都不用去设置，但是我们经常会被问及，为啥我服务器换了图片，你移动端iOS不更新呢，如果你是默认策略，我可以很负责的告诉你，你可以狠狠的怼回去，你懂不懂HTTP协议的缓存策略，我这是官方的实现，而不是，不明道理，婆婆诺诺，好，我改成一切都从服务器获取。不懂的哥们还以为你技术不过关呢，不过说句实话，确实许多人真的不太清楚 HTTP协议的缓存策略。

有哪些地方可以设置缓存策略 `URLRequestCache` 呢？其实看名字你就应该知道，这东西肯定是给 `URLRequest` 设置的，我每一个网络请求，都会有一个 `URLRequest`，但是，为了方便，苹果会有两个地方给你设置这个策略的地方，分别是：

- `URLSessionConfiguration`: 所有通过 `URLSession` 发出去的 `URLRequest` 都会带上这个策略。

```
swift
let url = URL(string: "http://152.136.154.126:3002/demo.gif")!
var request = URLRequest(url: url)
let config = URLSessionConfiguration.default

// 这这这 ↓ 这这这 ↓ 这这这 ↓ 这这这 ↓
config.requestCachePolicy = .reloadIgnoringLocalCacheData

let session = URLSession(configuration: config)
```

- `URLRequest`: 设置这个 `URLRequest` 的缓存策略，这个就没什么说的了。

```
swift
let url = URL(string: "http://152.136.154.126:3002/demo.gif")!
var request = URLRequest(url: url)

// 这这这 ↓ 这这这 ↓ 这这这 ↓ 这这这 ↓
request.cachePolicy = .returnCacheDataDontLoad

let config = URLSessionConfiguration.default
let session = URLSession(configuration: config)
```

Q 疑问🤔？同时设置，`URLSessionConfiguration` 和 `URLRequest` 的缓存策略，系统到底该用哪个呢？

A 答案: `URLRequest` 的缓存策略会覆盖掉 `URLSessionConfiguration` 的缓存策略。

[首页](#)[探索掘金](#)

区分 `URLCache` 和 `NSCache` nshipster.com/nsurlcache/ 这两东西不是一回事哈，看名字你也知道了，一个是专门缓存URL的类，一个是类似Map，字典的存储结构，用来缓存内存对象的。

上面也说了，`URLCache` 这个对象，其实是用于管理URL的缓存的，比如放在哪里呀，大小设置多少呀。

都哪些地方可以设置呢？有俩地方可以设置，分别是：

- `URLSessionConfiguration`: 单独为这个 `URLSession` 配置缓存对象，大小，路径等，如果单独为 `URLSessionConfiguration` 配置了缓存对象，由这个 `URLSession` 发出去的 `URLRequest`，都会使用这个缓存配置，不会使用全局的缓存对象。

```
let config = URLSessionConfiguration.default
config.requestCachePolicy = .reloadIgnoringLocalCacheData

let cache = URLCache(memoryCapacity: 4 * 1024 * 1024, diskCapacity: 20 * 1024 * 1024)
config.urlCache = cache
```

swift

- `URLCache`: 类方法提供了全局的 `URLCache` 配置，当 `URLSessionConfiguration` 没有另外设置的情况下，会使用全局的缓存作为自己的缓存。

```
URLCache.shared = URLCache(memoryCapacity: 0, diskCapacity: 0, diskPath: nil)
```

swift

Q 疑问🤔？如果同时设置 `全局缓存` 和 `URLSessionConfiguration` 缓存，系统有哪个缓存？

A 回答: 肯定使用 `URLSessionConfiguration` 的缓存，忽略全局缓存。

3. 几种特殊场景

1. `URLSessionConfiguration` 的缓存对象，空间设置为0。

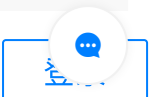
```
let url = URL(string: "http://152.136.154.126:3002/demo.gif")!
var request = URLRequest(url: url)
let config = URLSessionConfiguration.default

// 为这个Session单独设置缓存，并且空间都为0，因为都是用默认缓存策略，也就是使用`HTTP协议`缓存，
let cache = URLCache(memoryCapacity: 0, diskCapacity: 0)
```

swift

首页 ▾

探索掘金




```
let task = session.dataTask(with: request) { (data, resp, error) in
    let httpResp = resp as! HTTPURLResponse
    print("response (StatusCode):\("\(httpResp.statusCode)")")
    DispatchQueue.main.async {
        let httpResponse = resp! as! HTTPURLResponse
        for (key,value) in httpResponse.allHeaderFields {
            print("\(key):\("\(value)")")
        }
        self.imageView.image = UIImage.gifImageWithData(data: data! as NSData)
    }
}
task.resume()
```

2. URLCache全局缓存对象，空间设置为0。

```
swift
// 因为URLSessionConfiguration的缓存没有特殊设置，所以使用全局缓存，又因为全局缓存空间设置成0，又因
URLCache.shared = URLCache(memoryCapacity: 0, diskCapacity: 0, diskPath: nil)
let url = URL(string: "http://152.136.154.126:3002/demo.gif")!
var request = URLRequest(url: url)
let config = URLSessionConfiguration.default
let session = URLSession(configuration: config)
let task = session.dataTask(with: request) { (data, resp, error) in
    let httpResp = resp as! HTTPURLResponse
    print("response (StatusCode):\("\(httpResp.statusCode)")")
    DispatchQueue.main.async {
        let httpResponse = resp! as! HTTPURLResponse
        for (key,value) in httpResponse.allHeaderFields {
            print("\(key):\("\(value)")")
        }
        self.imageView.image = UIImage.gifImageWithData(data: data! as NSData)
    }
}
task.resume()
```

三，关于WKWebView和UIWebView里面的请求缓存是怎样的。

关于 **WebView** 这块是这样的，你设置的缓存策略，是页面里面的标签的缓存策略，也就是，你虽然设置了缓存策略是：不使用本地缓存，只是HTML加载的那些标签使用这个策略，比如 ``，但是在HTML的JS发起的XHR，或者Fetch请求，使用的还是默认缓存策略，你不会改变到他们。这是我



首页 ▾

探索掘金



拦截工具 项目地址: [github.com/zColdWater/...](https://github.com/zColdWater/)。

一，如果你想让这个网站的标签都使用默认的缓存策略，你就不需要另外设置。也就是使用HTTP协议的缓存策略。

```
let url = URL(string: "https://www.baidu.com/")
var request = URLRequest(url: url!)
webView.load(request)
```

swift

二，如果你想让这个网站的资源 **标签** 使用特定的缓存策略来请求，你可以这样处理。

```
let url = URL(string: "https://www.baidu.com/")
let request = URLRequest(url: self.url!, cachePolicy: .reloadIgnoringLocalCacheData)
webView.load(request)
```

swift

小结: 如果文章不够直观，可以下载Example项目，运行查看哦。

四，总结

其实对于iOS当中的 **URLRequest** 缓存，苹果的默认策略就是非常合理的选择，但是想要合理的使用这个默认策略，你需要了解 **HTTP协议** 的缓存策略，对于 **WebView** 的缓存策略我们也说明了，我希望可以帮到之前对这个缓存策略不清楚的童鞋，不要遇到问题一股脑就设置成不使用缓存了。合理的使用缓存是最好的选择。

希望我可以说的清楚明白，如有不准确或者不对的地方，希望大家指正。

Example 项目地址: [github.com/zColdWater/...](https://github.com/zColdWater/) 下载Demo，结合Demo一起实践更容易理解。

URLRequestCacheDemo.xcodeproj	add: server example	3 hours ago
URLRequestCacheDemo.xcworkspace	add: urlrequestcache example	23 hours ago
URLRequestCacheDemo	add: server example	3 hours ago
Podfile	add: server example	3 hours ago
Podfile.lock	add: server example	3 hours ago
README.md	Update README.md	5 minutes ago

README.md

URLRequestCacheExample

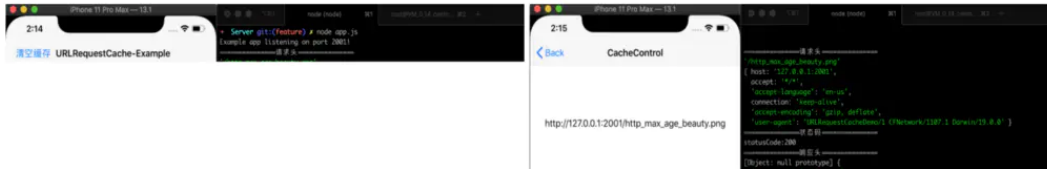
Explain what's meaning URLRequestCachePolicy in iOS

讲解文章地址: <http://47.99.237.180:8080/articles/2020/01/01/1577848007883.html> (不好意思, 域名还差几天才能转到阿里云。腾讯云实在有点。。。算了)

这个项目是为了结合文章理解, iOS当中的URLRequestCache和Policy的使用, 项目中分为iOS和Server两个端, 先将Server端开启, 然后运行iOS端, 分别进入不同的缓存策略, 观看终端的输出, 来理解缓存是如何工作的。

Example Screenshot


Example项目截图



参考: blackpixel.com/writing/201...

关注下面的标签, 发现更多相似文章

iOS



一人我编程累

Lv1 开发

获得点赞 11 · 获得阅读 5,065

关注

安装掘金浏览器插件

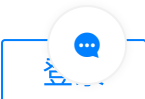
打开新标签页发现好内容, 掘金、GitHub、Dribbble、ProductHunt 等站点内容轻松获取。快来安装掘金浏览器插件获取高质量内容吧!

输入评论...



首页

探索掘金



橘子不酸丶 · 2天前 · iOS

iOS优化篇之App启动时间优化

👍 32 💬 1

老司机技术周报 · 1天前 · iOS / SwiftUI

【WWDC20】10037 - SwiftUI 中的 App 要领

👍 8 💬

FengyunSky · 4天前 · iOS

一文读懂iOS线程调用栈原理

👍 12 💬

ZenonHuang · 7天前 · iOS

iOS 的自动构建流程

👍 61 💬 7

JeremyHuang37 · 1天前 · iOS

【译】自定义 Collection View Layout -- 一个简单的模板

👍 💬 2

掘金酱 · 27天前 · iOS / Android / 前端 / 后端 / 程序员

🏆 掘金征文 | 2020与我的年中总结

👍 94 💬 109

Chouee · 3天前 · iOS

造轮子 - UITableView字母索引条

👍 7 💬 1

路过看风景 · 2天前 · iOS

CocoaPods原理 及 组件化

👍 3 💬

阿里巴巴淘系技术 · 5天前 · iOS

Apple Widget：下一个顶级流量入口？

👍 9 💬



首页 ▾

探索掘金



IOS事件处理 UIResponder

 2



