Кеширование URL запросов в iOS

Тарас Палиенко

- Что дает
- Как работает
- Пример
- Как сделать в iOS чтобы работало
- Разбор подводных камней

Что дает

- Меньше запросов на сервер
- Экономия траффика как клиента так и сервера
- Возможность получить данные в оффлайне
- Быстродействие, особенно при медленном интернете

Как работает

HTTP запросы и ответы используют заголовки (headers) для управления кешированием.

По умолчанию NSURLRequest использует текущее время для того чтобы определить нужно ли возвращать закешированный запрос. Для более точного контроля используют следующие заголовки:

Request cache headers

- If-Modified-Since - соответствует Last-Modified заголовку ответа (response).
- If-None-Match - соответствует Etag заголовку ответа.

Response cache headers

- **Cache-Control** - должен быть в ответе, чтобы включить кеширование на клиенте. Включает в себя такую информацию как **max-age**, **public** или **private**, или **no-cache**.

- **Last-Modified** - соответствует той дате и времени, когда запрашиваемый ресурс был изменен.

- **Etag** - идентификатор запрашиваемого ресурса, зачастую хеш (например, MD5) его свойств. Используется, если время изменения ресурса не очевидно.

NSURLConnectionDelegate

- Когда от сервера получен ответ, у нас есть возможность его изменить в методе **connection:willCacheResponse:**

- Если здесь вернуть nil, ответы не будут кешироваться

- Если не реализовать этот метод, то просто закешируется полученный ответ от сервера

Caching in HTTP

13 Caching in HTTP

HTTP is typically used for distributed information systems, where performance can be improved by the use of response caches. The HTTP/1.1 protocol includes a number of elements intended to make caching work as well as possible. Because these elements are inextricable from other aspects of the protocol, and because they interact with each other, it is useful to describe the basic caching design of HTTP separately from the detailed descriptions of methods, headers, response codes, etc.Caching would be useless if it did not significantly improve performance. The goal of caching in HTTP/1.1 is to eliminate the need to send requests in many cases, and to eliminate the need to send full responses in many other cases. The former reduces the number of network round-trips required for many operations; we use an "expiration" mechanism for this purpose (see section 13.2). The latter reduces network bandwidth requirements; we use a "validation" mechanism for this purpose (see section 13.3).Requirements for performance, availability, and disconnected operation require us to be able to relax the goal of semantic transparency. The HTTP/1.1 protocol allows origin servers, caches,and clients to explicitly reduce transparency when necessary. However, because non-transparent operation may confuse non-expert users, and might be incompatible with certain server applications (such as those for ordering merchandise), the protocol requires that transparency be relaxed

- only by an explicit protocol-level request when relaxed by
  client or origin server
- only with an explicit warning to the end user when relaxed by
  cache or client

Therefore, the HTTP/1.1 protocol provides these important elements:

. Protocol features that provide full semantic transparency when
  this is required by all parties.

2. Protocol features that allow an origin server or user agent to
   explicitly request and control non-transparent operation.
3. Protocol features that allow a cache to attach warnings to
   responses that do not preserve the requested approximation of
   semantic transparency.

A basic principle is that it must be possible for the clients to detect any potential relaxation of semantic transparency.

13.1.1 Cache Correctness

A correct cache MUST respond to a request with the most up-to-date response held by the cache that is appropriate to the request (see sections 13.2.5, 13.2.6, and 13.12) which meets one of the following conditions:

1. It has been checked for equivalence with what the origin server
   would have returned by revalidating the response with the
   origin server (section 13.3);
2. It is "fresh enough" (see section 13.2). In the default case,
   this means it meets the least restrictive freshness requirement
   of the client, origin server, and cache (see section 14.9); if
   the origin server so specifies, it is the freshness requirement
   of the origin server alone.
   If a stored response is not "fresh enough" by the most
   restrictive freshness requirement of both the client and the
   origin server, in carefully considered circumstances the cache
   MAY still return the response with the appropriate Warning
   header (see section 13.1.5 and 14.46), unless such a response
   is prohibited (e.g., by a "no-store" cache-directive, or by a
   "no-cache" cache-request-directive; see section 14.9).
3. It is an appropriate 304 (Not Modified), 305 (Proxy Redirect),
   or error (4xx or 5xx) response message.

If the cache can not communicate with the origin server, then a correct cache SHOULD respond as above if the response can be correctly served from the cache; if not it MUST return an error or warning indicating that there was a communication failure

If a cache receives a response (either an entire response, or a 304 (Not Modified) response) that it would normally forward to the requesting client, and the received response is no longer fresh, the cache SHOULD forward it to the requesting client without adding a new Warning (but without

Наглядный пример, Postman, 200, 304

Как сделать в iOS чтобы работало

```
// википедия
NSURL * url = [NSURL URLWithString:@"http://wikipedia.org"];

// создаем запрос
NSMutableURLRequest* request = [NSMutableURLRequest requestWithURL:url
                                                       cachePolicy:NSURLRequestUseProtocolCachePolicy
                                                   timeoutInterval:30.0];

// достаем закешированный ответ для этого запроса
NSCachedURLResponse * cachedResponse = [[NSURLCache sharedURLCache] cachedResponseForRequest:request];
NSHTTPURLResponse *response = (NSHTTPURLResponse *)cachedResponse.response;

if (response) {

    // устанавливаем соответствующие headers в запрос, если они есть в ответе
    if (response.allHeaderFields[@"last-modified"]) {

        NSString * lastModifiedValue = response.allHeaderFields[@"last-modified"];
        [request setValue:lastModifiedValue forHTTPHeaderField:@"If-Modified-Since"];
    }

    if (response.allHeaderFields[@"Etag"]) {

        NSString * etagValue = response.allHeaderFields[@"Etag"];
        [request setValue:etagValue forHTTPHeaderField:@"If-None-Match"];
    }
}
```

Подводные камни

Подводные камни

- Кеширование в работает в том случае, если **Cache-control** содержит директиву **public**.Значение **private** не дает закешировать ничего.

- Если в **Cache-control** не указан **max-age**, то вместо него будет взято стандартное значение, поэтому **max-age** стоит указывать на сервере явно даже если оно равно 0. Пока не истек **max-age**, iOS даже не будет проверять, изменился ли контент на сервере, и все response будут возвращены из кеша.

- Вы никогда не получите ответ 304 от NSURLConnection, так как эта логика скрыта в реализации. Вместо этого вы получите 200.

- Проверить откуда вы получили ответ –, с сервера или из кеша, – можно с помощью метода **connection:willCacheResponse:** : если этот метод *вызвался* перед получением ответа, значит запрос пришел с сервера, если нет - значит он взят из кеша.

Ссылки

http://nshipster.com/nsurlcache/

http://andrewmarinov.com/ioss-corenetwork-lying/

https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/URLLoadingSystem/Concepts/CachePolicies.html

That's all Folks!