link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=23 sentences=75, words=627

## 1. 缓存作用 #

- 减少了冗余的 数据传输，节省了网费。
- 减少了服务器的负担， 大大提高了网站的 性能
- 加快了客户端加载网页的 速度

## 2. 缓存分类 #

- 强制缓存如果生效，不需要再和服务器发生交互，而对比缓存不管是否生效，都需要与服务端发生交互
- 两类缓存规则可以同时存在，强制缓存优先级高于对比缓存，也就是说，当执行强制缓存的规则时，如果缓存生效，直接使用缓存，不再执行对比缓存规则**2.1 强制缓存#**强制缓存，在缓存数据未失效的情况下，可以直接使用缓存数据，那么浏览器是如何判断缓存数据是否失效呢？ 我们知道，在没有缓存数据的时候，浏览器向服务器请求数据时，服务器会将数据和缓存规则一并返回，缓存规则信息包含在响应header中。

** 2.2 对比缓存 #**

- 对比缓存，顾名思义，需要进行比较判断是否可以使用缓存。
- 浏览器第一次请求数据时，服务器会将缓存标识与数据一起返回给客户端，客户端将二者备份至缓存数据库中。
- 再次请求数据时，客户端将备份的缓存标识发送给服务器，服务器根据缓存标识进行判断，判断成功后，返回304状态码，通知客户端比较成功，可以使用缓存数据。

## 3. 请求流程 #

** 3.1 第一次请求 #**

** 3.2 第二次请求 #**

## 4. 通过最后修改时间来判断缓存是否可用 #

1. Last-Modified：响应时告诉客户端此资源的 最后修改时间
2. If-Modified-Since：当资源过期时（使用Cache-Control标识的max-age），发现资源具有 *Last-Modified*声明，则再次向服务器请求时带上头 *If-Modified-Since*。
3. 服务器收到请求后发现有头 *If-Modified-Since*则与被请求资源的最后修改时间进行比对。若最后修改时间较新，说明资源又被改动过，则 响应最新的资源内容并返回200状态码；
4. 若最后修改时间和 *If-Modified-Since*一样，说明资源没有修改，则响应304表示 未更新，告知浏览器继续使用所保存的缓存文件。

```
let http = require('http');
let fs = require('fs');
let path = require('path');
let mime = require('mime');
http.createServer(function (req, res) {
    let file = path.join(__dirname, req.url);
    fs.stat(file, (err, stat) => {
        if (err) {
            sendError(err, req, res, file, stat);
        } else {
            let ifModifiedSince = req.headers['if-modified-since'];
            if (ifModifiedSince) {
                if (ifModifiedSince == stat.ctime.toGMTString()) {
                    res.writeHead(304);
                    res.end();
                } else {
                    send(req, res, file, stat);
                }
            } else {
                send(req, res, file, stat);
            }
        }
    });
}).listen(8080);
function send(req, res, file, stat) {
    res.setHeader('Last-Modified', stat.ctime.toGMTString());
    res.writeHead(200, { 'Content-Type': mime.getType(file) });
    fs.createReadStream(file).pipe(res);
}
function sendError(err, req, res, file, stat) {
    res.writeHead(400, { "Content-Type": 'text/html' });
    res.end(err ? err.toString() : "Not Found");
}
```

## 5. 最后修改时间存在问题 #

1. 某些服务器不能精确得到文件的 最后修改时间， 这样就无法通过最后修改时间来判断文件是否更新了。
2. 某些文件的修改非常频繁，在秒以下的时间内进行修改. Last-Modified只能 精确到秒。
3. 一些文件的最后修改时间改变了，但是 内容并未改变。 我们不希望客户端认为这个文件修改了。
4. 如果同样的一个文件位于多个 CDN服务器上的时候内容虽然一样，修改时间不一样。

## 6. ETag #

ETag是实体标签的缩写，根据实体内容生成的一段 hash字符串,可以标识资源的状态。当资源发生改变时，ETag也随之发生变化。 ETag是Web服务端产生的，然后发给浏览器客户端。

1. 客户端想判断缓存是否可用可以先获取缓存中文档的 *ETag*，然后通过 *If-None-Match*发送请求给Web服务器询问此缓存是否可用。
2. 服务器收到请求，将服务器的中此文件的 *ETag*,跟请求头中的 *If-None-Match*相比较,如果值是一样的,说明缓存还是最新的,Web服务器将发送 *304 Not Modified*响应码给客户端表示缓存未修改过，可以使用。
3. 如果不一样则Web服务器将发送该文档的最新版本给浏览器客户端

```
let http = require('http');
let fs = require('fs');
let path = require('path');
let mime = require('mime');
let crypto = require('crypto');
http.createServer(function (req, res) {
    let file = path.join(__dirname, req.url);
    fs.stat(file, (err, stat) => {
        if (err) {
            sendError(err, req, res, file, stat);
        } else {
            let ifNoneMatch = req.headers['if-none-match'];
            let etag = crypto.createHash('sha1').update(stat.ctime.toGMTString() + stat.size).digest('hex');
            if (ifNoneMatch) {
                if (ifNoneMatch == etag) {
                    res.writeHead(304);
                    res.end();
                } else {
                    send(req, res, file, etag);
                }
            } else {
                send(req, res, file, etag);
            }
        }
    });
}).listen(8080);
function send(req, res, file, etag) {
    res.setHeader('ETag', etag);
    res.writeHead(200, { 'Content-Type': mime.lookup(file) });
    fs.createReadStream(file).pipe(res);
}
function sendError(err, req, res, file, etag) {
    res.writeHead(400, { "Content-Type": 'text/html' });
    res.end(err ? err.toString() : "Not Found");
}
```

## 7. 如何干脆不发请求 #

- 浏览器会将文件缓存到Cache目录，第二次请求时浏览器会先检查Cache目录下是否含有该文件，如果有，并且还没到 **Expires**设置的时间，即文件还没有过期，那么此时浏览器将直接从 **Cache**目录中读取文件，而不再发送请求
- **Expires**是服务器响应消息头字段，在响应 *http*请求时告诉浏览器在过期时间前浏览器可以直接从浏览器缓存取数据，而无需再次请求,这是HTTP1.0的内容，现在浏览器均默认使用HTTP1.1,所以基本可以忽略
- **Cache-Control**与 **Expires**的作用一致，都是指明当前资源的有效期，控制浏览器是否直接从浏览器缓存取数据还是重新发请求到服务器取数据,如果同时设置的话，其优先级高于 *Expires*
  *** 7.1 Cache-Control #*
- private 客户端可以缓存
- public 客户端和代理服务器都可以缓存
- max-age=60 缓存内容将在60秒后失效
- no-cache 需要使用对比缓存验证数据,强制向源服务器再次验证
- no-store 所有内容都不会缓存，强制缓存和对比缓存都不会触发

> Cache-Control:private, max-age=60, no-cache

```
let http = require('http');
let fs = require('fs');
let path = require('path');
let mime = require('mime');
let crypto = require('crypto');
http.createServer(function (req, res) {
    let file = path.join(__dirname, req.url);
    console.log(file);

    fs.stat(file, (err, stat) => {
        if (err) {
            sendError(err, req, res, file, stat);
        } else {
            send(req, res, file);
        }
    });
}).listen(8080);
function send(req, res, file) {
    let expires = new Date(Date.now() + 60 * 1000);
    res.setHeader('Expires', expires.toUTCString());
    res.setHeader('Cache-Control', 'max-age=60');
    res.writeHead(200, { 'Content-Type': mime.lookup(file) });
    fs.createReadStream(file).pipe(res);
}
function sendError(err, req, res, file, etag) {
    res.writeHead(400, { "Content-Type": 'text/html' });
    res.end(err ? err.toString() : "Not Found");
}
```