

如何进行代码压缩是页面性能最优（HTTP 压缩）

前端工程师做出的某些决定可以显著的减少 HTTP 请求以及响应的网络传输时间。终端用户的带宽速度，网络服务提供商，距网络交换节点的距离这些因素都不受开发团队的控制。但是仍有其它的因素会影响响应时间。比如，通过压缩 HTTP 响应来减少响应时间。

从 HTTP/1.1 开始，web 客户端就开始支持 HTTP 请求的 Accept-Encoding header。
Accept-Encoding: gzip, deflate

如果 web 服务器在请求中看到这种 header，它就会用客户端列出来的方法来压缩响应内容。web 服务器通过响应中的 Content-Encoding header 通知 web 客户端。Content-Encoding: gzip

Gzip 是目前最流行以及最有效的压缩方法。其它你有可能看到的压缩格式是 deflate，但是它不是很流行以及很有效。

Gzip 通常可以把响应内容大小减少 70%。目前浏览器当中接近 90% 的网络流量支持 gzip。

对于浏览器以及代理来说还有一个已知问题就是浏览器期望的内容和它获取的压缩内容可能不匹配。幸运的是，这种情况随着旧版浏览器的使用率越来越低会越来越少。Apache 模块通过添加适当的不同响应头来解决这个问题。

服务端选择用 Gzip 压缩内容主要依赖于文件类型，但是通常也受限于要决定压缩的内容。大部分 web 站点用 gzip 压缩 html 文档。当然也值的压缩脚本以及样式文件，但是很多网站都没有选择这么做。事实上，可以用 gzip 压缩任何文本响应内容，包括 XML 和 JSON。Image 和 PDF 不建议 gzip 压缩，因为它们都是被压缩过的。试图压缩它们不仅浪费 CPU 也有可能增加文件大小。

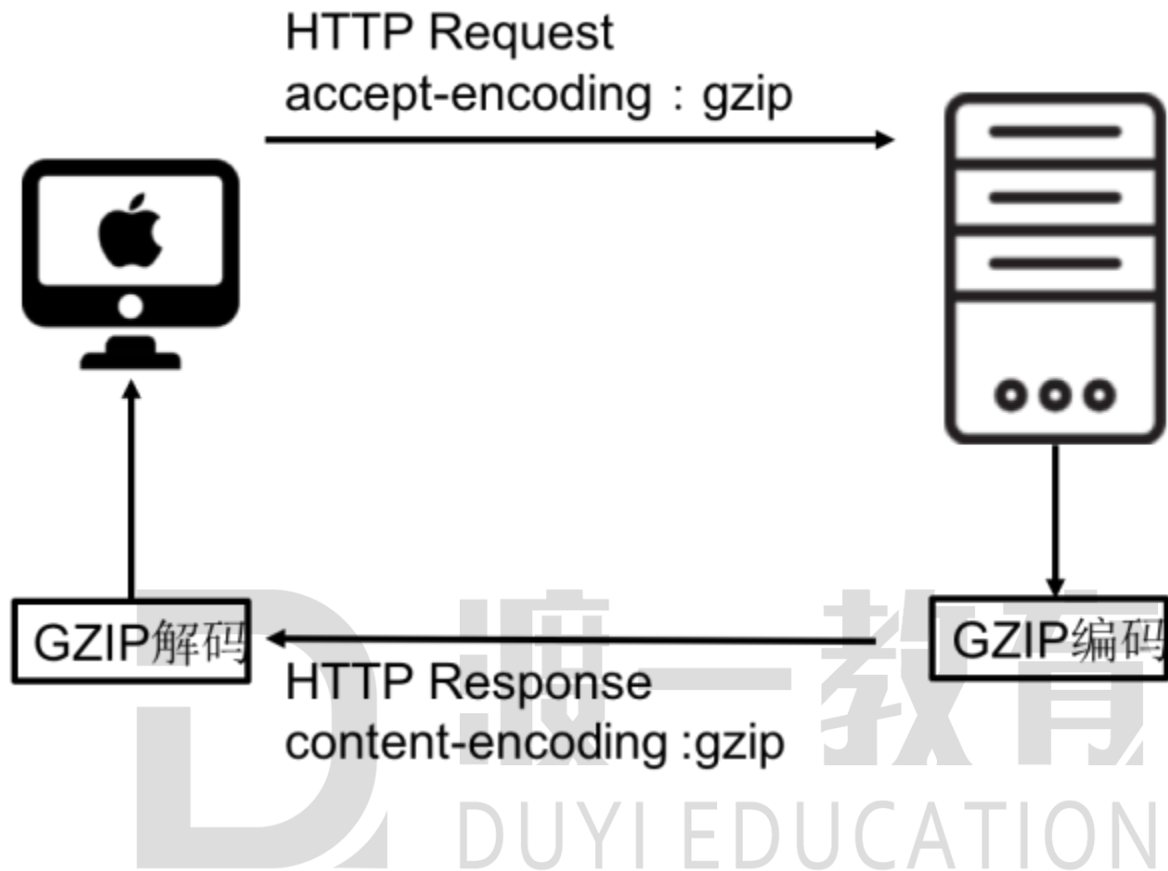
用 gzip 压缩尽可能多的文件类型是减少页面大小提升用户体验的一种简单方法。

下面来深入的了解一下 gzip。

gzip

gzip 是 GNUzip 的缩写，最早用于 UNIX 系统的文件压缩。HTTP 协议上的 gzip 编码是一种用来改进 web 应用程序性能的技术，web 服务器和客户端（浏览器）必须共同支持 gzip。目前主流的浏览器，Chrome, firefox, IE 等都支持该协议。常见的服务器如 Apache, Nginx, IIS 同样支持 gzip。

gzip 压缩比率在 3 到 10 倍左右，可以大大节省服务器的网络带宽。而在实际应用中，并不是对所有文件进行压缩，通常只是压缩静态文件。



- 1) 浏览器请求 url，并在 request header 中设置属性 `accept-encoding: gzip`。表明浏览器支持 gzip。
- 2) 服务器收到浏览器发送的请求之后，判断浏览器是否支持 gzip，如果支持 gzip，则向浏览器传送压缩过的内容，不支持则向浏览器发送未经压缩的内容。一般情况下，浏览器和服务器都支持 gzip，response headers 返回包含 `content-encoding: gzip`。
- 3) 浏览器接收到服务器的响应之后判断内容是否被压缩，如果被压缩则解压缩显示页面内容。

下面以淘宝为例，验证一下开启 gzip 的效果。客户端（浏览器）请求 <http://www.taobao.com/>。本次测试使用的浏览器为 Chrome，打开控制台查看网络信息可以看到 request headers 中包含：`accept-encoding: gzip, deflate, sdch`，表明 chrome 浏览器支持这三种压缩。这里值得一提的是 `accept-encoding` 中添加的另外两个压缩方式

deflate 和 sdch。deflate 与 gzip 使用的压缩算法几乎相同，这里不再赘叙。sdch 是 Shared Dictionary Compression over HTTP 的缩写，即通过字典压缩算法对各个页面中相同的内容进行压缩，减少相同的内容的传输。sdch 是 Google 推出的，目前只在 Google Chrome, Chromium 和 Android 中支持。

Request Headers

```
:host: www.taobao.com
:method: GET
:path: /
:scheme: https
:version: HTTP/1.1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
accept-encoding: gzip, deflate, sdch
accept-language: zh-CN,zh;q=0.8,en;q=0.6,zh-TW;q=0.4
cache-control: max-age=0
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36
```

淘宝 request header

▼ Response Headers

```
age: 74
cache-control: max-age=0, s-maxage=100
content-encoding: gzip
content-type: text/html; charset=utf-8
date: Mon, 02 Jan 2017 10:18:40 GMT
eagleid: 65e3d0e514833523200992238e
server: nginx
status: 200 OK
strict-transport-security: max-age=31536000
timing-Allow-Origin: *
vary: Ali-Detector-Type, X-CIP-PT
version: HTTP/1.1
via: cache6.l2cn8[0,200-0,H], cache13.l2cn8[0,0], cache10.cn417[0,200-0,H], cache1.cn417[0,0]
x-cache: HIT TCP_MEM_HIT dirn:-2:-2
x-swift-cachetime: 92
x-swift-savetime: Mon, 02 Jan 2017 10:17:34 GMT
```

淘宝 response header

在企业级应用中，通常被使用到的服务器有 nginx, Apache 等。nginx 是取代 Apache 的高性能服务器，本文接下来的内容会介绍一下在 Nginx 中如何开启 gzip。

Nginx 中开启 gzip:

如果服务端接口使用 nodejs 和 express，那么开启 nginx 非常简单。启用 `compress()` 中间件即可并在 `nginx.conf` 中添加 gzip 配置项即可，`express.compress()` gzip 压缩中间件，通过 `filter` 函数设置需要压缩的文件类型。压缩算法为 gzip/deflate。这个中间件应该放置在所有的中间件最前面以保证所有的返回都是被压缩的。如果使用 java 开发，需要配置 `filter`。

下面详细介绍一下如何在 `nginx.conf` 中配置 gzip。配置的 gzip 参数图所示：

```
#gzip on;  
gzip on;  
gzip_comp_level 9;  
gzip_min_length 100;  
gzip_types text/plain text/css application/xml application/javascript;  
gzip_vary on;
```

gzip 参数

添加完参数后，运行 `nginx -t` 检查一下语法，若语法检测通过，则开始访问 url 检测 gzip 是否添加成功。以下为我所使用的 gzip 配置的作用。

- 1) `gzip on`: 开启 gzip。
- 2) `gzip_comp_level`: gzip 压缩比。
- 3) `gzip_min_length`: 允许被压缩的页面最小字节数。
- 4) `gzip_types`: 匹配 MIME 类型进行压缩，`text/html` 默认被压缩。

http 与 gZip

gZip 文件怎么通讯

我们传输压缩文件给别人时候一般都带着后缀名 `.rar`，`.zip` 之类，对方在拿到文件后根据相应的后缀名选择不同的解压方式然后去解压文件。我们在 `http` 传输时候解压文件的这个角色的扮演者就是我们使用的浏览器，但是浏览器怎么分辨这个文件是什么格式，应该用什么格式去解压呢？

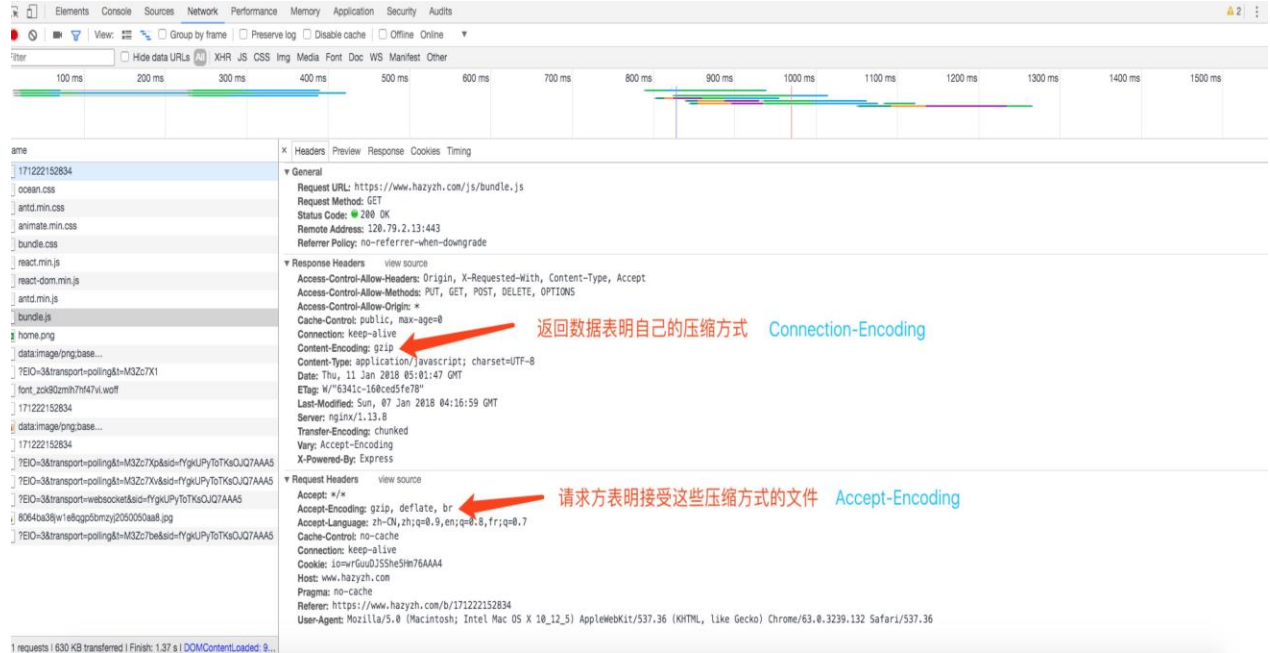
在 `http / 1.0` 协议中关于服务端发送的数据可以配置一个 `Content-Encoding` 字段，这个字段用于说明数据的压缩方法

`Content-Encoding: gzip`
`Content-Encoding: compress`
`Content-Encoding: deflate`

客户端在接受到返回的数据后去检查对应字段的信息，然后根据对应的格式去做相应的解码。客户端在请求时，可以用 `Accept-Encoding` 字段说明自己接受哪些压缩方法。

Accept-**Encoding**: gzip, deflate

我们在浏览器的控制台中可以看到请求的相关信息



兼容性

提到浏览器作为一个前端就不由自主的会想一个问题，会不会有浏览器不支持呢。HTTP/1.0 是 1996 年 5 月发布的。好消息是基本不用考虑兼容性的问题，几乎所有浏览器都支持它。值得一提的是 ie6 的早起版本中存在一个会破坏 gZip 的错误，后面 ie6 本身在 WinXP SP2 中修复了这个问题，而且用这个版本的用户数量也很少。

谁去压缩文件

这件事看起来貌似只能服务端来做，我们在网上看到最多的也是诸如 nginx 开启 gZip 配置之类的文章，但是现在前端流行 spa 应用，用 react, vue 之类的框架时候总伴随这一套自己的脚手架，一般用 webpack 作为打包工具，其中可以配置插件如 [compression-webpack-plugin](#) 可以让我们把生成文件进行 gZip 等压缩并生成对应的压缩文件，而我们应用在构架时候有可能也会在服务区和前端文件中放置一层 node 应用来进行接口鉴权和文件转发。nodejs 中我们熟悉的 express 框架中也有一个 [compression](#) 中间件，可以开启 gZip, 一时间看的人眼花缭乱，到底应该用谁怎么用呢？

服务端响应请求时候压缩

其实 nginx 压缩和 node 框架中用中间件去压缩都是一样的，当我们点击网页发送一个请求时候，我们的服务端会找到对应的文件，然后对文件进行压缩返回压缩后的内容【当然可以利用缓存减少压缩次数】，并配置好我们上面提到的 Content-Encoding 信息。对于一些应用在构架时候并没有上游代理层，比如服务端就一层 node 就可以直接用自己本身的压缩插件对文件进行压缩，如果上游配有有 nginx 转发处理层，最好交给 nginx 来处理这些，因为它们有专门为此构建的内容，可以更好的利用缓存并减小开销（很多使用 c 语言编写的）。

我们看一些 nginx 中开启 gZip 压缩的一部分配置

```
# 开启 gzip
gzip on;
# 启用 gzip 压缩的最小文件，小于设置值的文件将不会压缩
gzip_min_length 1k;
# gzip 压缩级别，1-10，数字越大压缩的越好，也越占用 CPU 时间，后面会有详细说明
gzip_comp_level 2;
# 进行压缩的文件类型。javascript 有多种形式。其中的值可以在 mime.types 文件中找到。
gzip_types text/plain application/javascript application/x-javascript text/css
application/xml text/javascript;
```

应用构建时候压缩

既然服务端都可以做了为什么 webpack 在打包前端应用时候还有这样一个压缩插件呢，我们可以在上面 nginx 配置中看到 gzip_comp_level 2 这个配置项，上面也有注释写道 1-10 数字越大压缩效果越好，但是会耗费更多的 CPU 和时间，我们压缩文件除了减少文件体积大小外，也是为了减少传输时间，如果我们将压缩等级配置的很高，每次请求服务端都要压缩很久才返回信息回来，不仅服务器开销会增大很多，请求方也会等的不耐烦。但是现在的 spa 应用既然文件都是打包生成的，那如果我们在打包时候就直接生成高压压缩等级的文件，作为静态资源放在服务器上，接收到请求后直接把压缩的文件内容返回回去会怎么样呢？

webpack 的 compression-webpack-plugin 就是做这个事情的，配置起来也很简单只需要在装置中加入对应插件，简单配置如下

```
const CompressionWebpackPlugin = require('compression-webpack-plugin');


webpackConfig.plugins.push(
  new CompressionWebpackPlugin({
    asset: '[path].gz[query]',
    algorithm: 'gzip',
    test: new RegExp('\\.(js|css)$'),
```

```

    threshold: 10240,
    minRatio: 0.8
  })
)

```

webpack 打包完成后生成打包文件外还会额外生成 .gz 后缀的压缩文件



common.js	1.18 kB	0	[emitted]	common
index.js	2.34 MB	1, 0	[emitted]	index
index.css	406 kB	1, 0	[emitted]	index
index.css.gz	60.5 kB		[emitted]	
index.js.gz	543 kB		[emitted]	

那么这个插件的压缩等级是多少呢，我们可以在源码中看到默认的 level 是 9

```

...
const zlib = require('zlib');
this.options.algorithm = zlib[this.options.algorithm];
...
this.options.compressionOptions = {
  level: options.level || 9,
  flush: options.flush
  ...
}

```

可以看到压缩使用的是 zlib 库，而 zlib 分级来说，默认是 6，最高的级别就是 9 Best compression (also zlib.Z_BEST_COMPRESSION)，因为我们只有在线上项目时候才回去打包构建一次，所以我们在构建时候使用最高级的压缩方式压缩多耗费一些时间对我们来说根本没任何损耗，而我们在服务器上也不用再去压缩文件，只需要找到相应已经压缩过的文件直接返回就可以了。

服务端怎么找到这些文件

在应用层面解决这个问题还是比较简单的，比如上述压缩文件会产生 index.css, index.js 的压缩文件，在服务端简单处理可以判断这两个请求然后给予相对应的压缩文件。以 node 的 express 为例

```

...
app.get(['/index.js', '/index.css'], function (req, res, next) {
  req.url = req.url + '.gz'
  res.set('Content-Encoding', 'gzip')
}

```



```
res.setHeader("Content-Type", generateType(req.path)) // 这里要根据请求文件  
设置 content-type  
next()  
})
```

上面我们可以给请求返回 gZip 压缩后的数据了，当然上面的局限性太强也不可取，但是对于处理这个方面需求也已经有很多库存在，express 有 [express-static-gzip](#) 插件 koa 的 koa-static 则默认自带对 gZip 文件的检测，基本原理就是对请求先检测 .gz 后缀的文件是否存在，再去根据结果返回不同的内容。

哪些文件可以被 gZip 压缩

gZip 可以压缩所有的文件，但是这不代表我们要对所有文件进行压缩，我们写的代码（css, js）之类的文件会有很好的压缩效果，但是图片之类文件则不会被 gzip 压缩太多，因为它们已经内置了一些压缩，一些文件（比如一些已经被压缩的像.zip 文件那种）再去压缩可能会让生成的文件体积更大一些。当然已经很小的文件也没有去压缩的必要了。

