链滴

经历心得分享：用 Web 框架 Gin 开发 API，但业务返回的 JSON 太大，调研了一圈，最后自己写了个中间件做 gzip 压缩，记录下调研和开发调优时的心得

作者：nanmu42

原文链接：https://ld246.com/article/1576226084467

来源网站：链滴

<h2 id="背景">背景</h2>
最近在做一个 web 版的展示大屏，前端靠 HTTP(S)+JSON 和后端交互，部分图形是密集的地点位和时间序列，HTTP 返回数据量较大，公网上加载速度不佳。
<h2 id="调研">调研</h2>
考虑压缩 HTTP 返回，选了 gzip 这个常规选项。
gzip 在压缩时，得考虑几点：

内容类型是否对压缩友好：例如 JPEG 本身已经压缩过，再次压缩收益太小，费力不讨好；
内容大小是否值得压缩：太小的内容压缩效果不好（甚至有可能比原文还大，毕竟有字典等开销，另外小内容本身传输时间短，直接放过去还能节约 CPU 和内存。

<h2 id="先考察了-Nginx">先考察了 Nginx</h2>

Nginx 的压缩一般是用 <code>ngx_http_gzip_module</code>；
根据 <code>Content-Type</code> 判断内容类型是否对压缩友好，要压缩的类型由用户定义
根据返回的 <code>Content-Length</code> 判断内容大小是否值得压缩；

当返回的 JSON 大于 2KB 时，Golang 会使用 chunk 的形式传输，这个时候没有 <code>Content-Length</code>，<code>ngx_http_gzip_module</code> 不会进行内容压缩。
<h2 id="又考察了下-gin-contrib-gzip">又考察了下 gin-contrib/gzip</h2>

根据扩展名判断内容类型是否对压缩友好；
根据 Path 判断是否启用压缩；
不支持判断内容大小是否值得压缩。

根据 Path 来判断确实可行，但太死板，业务和中间件耦合了。
<h2 id="自己造了个轮子">自己造了个轮子</h2>
<h2 id="特性">特性</h2>
看了 gin-contrib/gzip 和 Caddy 的实现后，我造了个自己的轮子，欢迎试用、Star 以及反馈
仓库地址： https://github.com/nanmu42/gzip
文档： <a href="https://ld246.com/forward?goto=https%3A%2F%2Fithub.com%2Fnanmu42%2Fgzip%2Fblob%2Fmaster%2FREADME.Chinese.md" target="_blank rel="nofollow ugc">https://github.com/nanmu42/gzip/blob/master/README.Chinese.md</>

支持 Gin 和 标准库 net/http ；
支持基于 <code>Content-Type</code>、<code>Content-Length</code>、扩展名判断否压缩；
启用压缩的阈值用户可以自定义；
压缩级别用户可以自定义；
不压缩已经压缩过的返回，不压缩 Head 请求的返回，不影响 HTTP Upgrade ；
中间件初始化简单，集成容易；

<h2 id="更进一步">更进一步</h2>
还记得返回的 JSON 大于 2KB 时，Golang 会使用 chunk 的形式传输，这个时候没有 Content-ength 带来无法判断内容是否值得压缩的问题吗？
我取了个巧，如果 Content-Length 不存在，中间件会去观察 <code>http.ResponseWriter.Wr

te(data []byte)</code> 的第一次调用时的 <code>len(data)</code>，如果此时 <code>len(data</code> 已经大于启用压缩的阈值，那么可以安全地开始压缩。
<h2 id="调优">调优</h2>
这里分享在造轮子时的两个调优点。
此部分可配合项目各阶段 benchmark 食用： https://github.com/nanmu42/gzip/blob/a0b9dac85d4a0a72f4a2183d3b9bfadf215f2168/docs/benchmarks.md
<h2 id="AC-自动机">AC 自动机</h2>
原本我使用 <code>Strings.Contains()</code> 配合循环来判断文件后缀 /MIME 是否在支持压缩的列表中，但 benchmark 下来效果不太好。做了一些搜索后发现 Cloudflare 实现了一个 AC 自动机来做这个事情。和维护者聊了聊之后，用了它的一个 fork： https://github.com/signalsciencs/ac
<h2 id="toc_h2_9"><code>Sync.Pool</code></h2>
<code>Sync.Pool</code> 用来做对象重用，以降低系统内存分配和 Go 垃圾回收的压力，一始我只对 gzip.Writer 做了对象重用，但发现中间件对内存的影响还有一些大，后来我用了第二个 <code>Sync.Pool</code> 重用 wrapper，内存使用量和 CPU 时间都有了可观的改善。
两个调优之后，CPU 时间下降为调优前的 40%，内存使用量下降为原先的一半。