# 全缓存动态页面

## Contents


**1 此文目的** **2**

**2 关于互联网** **2**

**3 阅读对象** **2**

**4 技术问题** **2**

4.1 全局服务器负载均衡 . . . . . 3

4.2 通过合并文件减少网络请求 . . . . . 3

4.3 通过浏览器缓存减少网络请求和数据传输 . . . . . 4

4.4 动态内容读取缓存、写操作入队列 . . . . . 4

4.5 服务器端压缩量大数据 . . . . . 8

4.6 动态内容静态化 . . . . . 8

**5 后记** **8**

# 1 此文目的

漂亮的色彩、绚丽的动画，在输入网址后便能呈现在你的眼前。 互联网无可否认已经融入了我们的生活。

我们可以山寨出iPhone，却很难有属于自己独特的理念，关于事物的思想和心脏。

互联网是现实、艺术、与技术的结合。 本文意在总结作者从业经验的同时， 对互联网的技术范畴提出自己的一点理解。

# 2 关于互联网

像生活中各个领域一样，互联网是个普通的行业，形形色色的商家、个人、组织参与到这个行业。 但互联网行业也有其独特的特点： 创造性、自由性、与平等性（至少技术上是）。

只要你掌握了适当的工具，你可以自由的实现你的想法，自由的供人使用。普通用户使用你的产品， 对比使用世界上任何角落其他产品的方法和成本是一样的。

# 3 阅读对象

这不是一篇关于互联网开发入门的文章，如标题所指，这里总结的，仅仅是作者在web页面高速获取方面的一些经验。

所以，本文的理想读者为具有一定互联网开发经验、并正在被网站的性能问题所困扰的你。

# 4 技术问题

页面打开速度，是影响网站使用价值的一个重要因素。

在无法改变用户自身网络条件的前提下，我们可以从以下几个方面来优化用户的速度体验：

## 4.1 全局服务器负载均衡

简称GSLB。 在网站用户的物理分布范围较广、速度要求高的场合下采用。

通过维护自己的 dns 服务器，将来访用户根据地理位置的不同指向与之对应较近的服务器进行后续数据请求。

比如，山东用户与深圳用户访问同一个新浪页面，虽然用户看到的内容一样，但各自连接的数据服务器， 很可能就不是同一台机器。

GSLB 技术需要比较准确的用户定位数据（比如专业ip库，或者移动设备的位置信息）， 和专业的设备支持。 有兴趣和实力的可以深入了解，由于作者水平有限，在此不作深入介绍。

## 4.2 通过合并文件减少网络请求

用户浏览器和 http 服务器之间从发起请求、建立连接、到开始传送数据，是需要耗费时间的。 特别是在文件数很多的情况下，这上面耗费的时间便值得关注了。

合并文件的原则是小文件合并、同类文件合并。

实际情况一般应用在以下两种文件:

- 网站背景资源图片文件

  将网站的按钮、边框、甚至logo等资源文件合并在一起，通过 css 背景定位属性显示出来。 搜索 css sprites，便可以看到很多相关文章。

- css、js 文本文件

  为了便于开发和维护，网站的 css, js 文件往往会分成多个文件存放。 但放到运营环境的时候， 我们可以通过压缩、合并。 从而也可以达到减少服务器请求的目的。

  有很多方式合并该类文件。 据闻，淘宝便采用了单独的接口来动态完成这一合并工作， 并即时返回合并后的资源文件。 方式有很多种，但效果都是差不多的。 有兴趣的读者不妨深入去了解。

通过合并文件减少网络请求的好处显而易见，但同时，该方式也会增加开发人员的时间成本。 特别是后一种方式，如果不做特殊处理，开发人员在每查看一次修改效果之前，都需要执行某个命令来合并文件。

## 4.3 通过浏览器缓存减少网络请求和数据传输

浏览器 和 http 服务器之间的数据传递，是遵循了一套约定好的规则的。在速度的提高方面， 有以下几点最简单、有效:

- Cache-Control

  在返回页面的 HTTP Header 里输出 Cache-Control， 比如 Cache-Control: max-age=100， 此 header 告诉浏览器，从接受到这条数据开始，100秒内，不要再向我请求同样的地址，当然， 用户的Ctrl+F5指令会让浏览器无视该header。

  需要注意的是，http 服务器上需要设置正确的时间，以确保该指令能被所有浏览器正常执行。

  该 header 对于静态、动态页面同样有效。 静态页面的 header 可以通过 http 服务器添加， 灵活起见，动态页面的该指令最好是程序自己输出。

  有些资料上提及 Expire 指令，也能达到同样的效果。 据个人经验，Expire 指令使用起来不如 Cache-Control 灵活方便。 而且， 大部分浏览器会优先执行 Cache-Control 指令。

- Last-Modified

  Last-Modified 是浏览器与 http 服务器之间约定的另一个指令，通常不需要我们特意配置， 多数服务器和浏览器能自行完成该套指令的动作，没有修改过的文件，不会重复传送相同的数据（请求还是会发起的）。

通过浏览器缓存，减少网络请求、或者网络数据传输。 是提高页面响应速度最简单、有效的方法。 很多网站，用户打开第一个页面很慢，后续页面却很快的原因就在此。

返回合适的 HTTP Header 信息，能让你的网站速度提升达到意想不到的效果。

## 4.4 动态内容读取缓存、写操作入队列

这里，我们到达了服务器响应这个步骤。

静态页面的响应基本不是问题（只要配置得当，静态页面不应该是网站的性能瓶颈），这个小节主要描述如何提高动态页面的响应时间。

本文其他内容大部分都有特定标准，只要尊章行事便可，已无太多的优化和提升空间。 但本章所涉及的内容， 便是各网站技术差异化的体现。

用 firebug 观察各网站动态页面的请求，有个“等待响应”时间，便是本节要讨论的内容。

从服务器接受到浏览器的请求开始，到服务器开始返回数据给浏览器为止的这段时间，便是等待响应时间。

用百度和谷歌搜索相同的几个生僻字组合，我们会发现，前者的等待时间基本上是后者的 3 至 4 倍。 百度和谷歌的差距尚且如此， 其他水平不一的网站就更不用说了。 据作者个人经验，相同的内容，相同的 http 服务器， 优化模式 和 普通模式 响应时间差距在8倍左右。 前者在数十毫秒，后者在数百毫秒。

上面提到的优化模式，一言以蔽之，便是本文的标题：全缓存动态页面。

在相同工作量下，服务器的性能瓶颈，依次是磁盘、网络（高速内网） 、内存 、cpu。

在开发过程中，尽量减少与磁盘和网络的数据交互、不要进行特别费时的系统调用、然后程序没有严重的时耗漏洞， 基本上，我们都能比较容易的做到页面的优化响应。

我们来阐述下高速动态页面响应值得注意的几个方面：

- http服务器

  各种 http 服务器性能讨论一直是很多开发人员比较关注的问题。

  很多个人和团体开发了专有服务器。 作者认为，除了诸如长连接、静态小页面等专用的 http 服务器外， 各种开源 http 服务器已能满足普通动态页面响应需求。 诸如 nginx 或 lighttpd 在性能上都表现很不错。

- 响应模式

  响应模式是指 http 服务器以何种方式响应浏览器的动态页面请求。 为了获得更快的响应速度， 强烈建议让你的程序运行在 FastCGI 模式响应下。 FastCGI 只是普通 cgi 模式的一种改进， 该模式可以省掉 http 服务器对每个请求都 fork 我们编写的程序的开销。

  多数情况下，各种解释型语言编写的程序，不用特殊配置，默认运行于 FastCGI 模式。

- 程序结构

  无论使用什么语言或者技术，输出动态页面的程序，本质上都是 MVC 的程序结构。 controller 根据输入参数， 从 model 返回数据， 并返回给 view 输出。 这也是一个 cgi 需要做的所有事情。

控制器、数据模块、和模板引擎 这三个部分各自都有很多专业的开源软件来完成其的功能。 也有些开源框架替我们完成了所有这些操作的封装，开发者只需按照其约定的方法调用类、或者函数即可。

对于需要高速响应的页面程序，个人觉得使用开源的小工具搭建属于自己的框架更加适合，原因有两个方面:

- 具体的事应该交给专业的人来做

  这也是 unix 系统的设计哲学。 虽然 web 框架远没有 unix 系统复杂， 但毕竟也没有简单到一套框架便能满足各种应用的地步。

- 某个零件不顺手可以随意更换

  如果你对觉得模板引擎不好用，或者目前的控制器处理 URL 参数的方式不能满足需求，对于用小工具组成的框架， 很容易进行部件更换，如果是开源的，甚至很容易进行扩展和定制。

互联网程序框架需要一个逐步完善的过程。 一个高速、易用、灵活的程序框架，需要开发者投入大量的时间和精力。

网络上有很多各种语言开发好的各种框架供我们选用，学习成本也不是太大。 如果我们只是需要解决问题， 动态生成一个页面。 有经验的开发者三五天便能掌握使用。 正因为学习成不不高， 很多开发人员便喜欢研究各种新型的互联网技术和框架，一来开拓技术视野、二来也显得自己什么都会。 无可否认，这种现象在身边既然普遍存在，便有其合理性，商业社会都崇尚时间效率、喜欢追赶潮流，担心与潮流脱节、被社会抛弃。 社会需要这样的角色，同时，也需要对事物追求甚深的角色。 后者很少，显得有点异类，做事方法比起前者来显得有点呆拙， 时不时会被别人嗤笑其迂腐。 还是那句话，社会同样需要这种角色，所以，如果你已经是后者，请坦然承受吧。

- 内存缓存

  前面提到过，内存读写的开销是所有读写开销中最小的一个。 所以，我们可以将相同URL、相同参数的返回数据，存放到内存里， 在内容有修改之前的读取，便可以直接返回内存中的内容给浏览器。 甚至，我们可以通过前面的浏览器缓存手段， 将内容缓存到浏览器客户端（只是这种方法不能在内容修改后即时返回更新数据）。

  内存缓存的开源工具主要是 memcached，还有一些诸如 redis, nmdb 等数据后台，兼具了内存缓存的功能。 甚至 mysql, pg 等本身也具有某种程度上的读缓存，只是不够灵活。

- 数据后台

数据后台负责数据的存放和获取。 目前比较常用的数据后台分为两大技术分支： SQL 和 NoSQL。

两大分支分别有多种开源实现，读者可以有多种选择。 NoSQL 技术出现得比前者晚一点， 多数有自有的数据格式规范和操作方式。 有诸如操作简单、易于学习的优点， 很多新兴的网站都使用这种技术来实现。 由于我本人没有这方面的实践经验，故不做介绍。

SQL 领域常见的有 mysql 和 postgresql，前者使用广泛，易于掌握；后者可定制性强。 经个人使用，两者实无太多差别，况且，如果程序只做简单的内存缓存，一台数据库服务器最多能支持每天数百万访问量的应用。 从而，我们需要在数据库服务器和 http 服务器 中间，增加一套缓冲系统。 这也是本节标题是“数据后台” 而不是“数据库后台”的原因。 数据后台，包含了数据库系统和这套缓冲系统。

缓冲系统，主要包含读操作的内存缓存，和写操作的队列缓存（也是内存缓存）。 读缓存很好理解，跟 memcached 的技术差不多，相同的输入参数，从内存中直接读取相同的输出。 写队列主要是实现对数据库写操作的队列缓存，以缓解数据库压力。

本文作者在 nmdb 的基础上实现了这套缓存，详见
https://github.com/bigml/cmoon/tree/master/event
该系统以插件的方式支持各业务应用的开发，每个业务一个线程 + 一个写队列 + 一片读缓存内存。 在本地局域网环境下，最大支持每秒3万次读写（通过调整内核参数应该可以继续优化）。 2G内存的服务器上， 最大支持百万次写缓存。 同时，该系统支持多台服务器提供同一个应用的分布式扩展， 能够通过多台机器来有效分担高密度应用的压力。

高速数据后台，除了满足速度和稳定性以外，还需要提供易用的接口 api 供各应用调用。 这样对于程序的开发效率， 和日后的维护效率，都能得到有效保障。

对于接口的易用性，是个讨论广泛，而且由来已久的话题。 对于他的理解，已经超出了本文的预定范围， 不过我相信只要读者足够用心，必定能够遇到最适合自己和团队的接口方式。

- 其他耗时

  web程序其他耗时常见于以下几点：

  - 文件读取

    读取文件（配置文件、模板文件之类），相对于毫秒级的响应，也是个耗时的操作。 所以，我们建议使用 FastCGI 模式， 这样在程序初始化时将所有需要读取的文件读入内存，响应每个请求时，便不需要重复读取。

- 系统调用

  以下系统调用非常费时，请谨慎使用:

  * system(), exec(), fork() 等创建新进程的系统调用
  * fopen() fwrite() 等磁盘文件操作（包括写日志文件）
  * time() 获取系统时间

- 网络连接

  tcp 方式的请求（诸如数据库请求），是需要建立连接的。 这样的操作，对于毫秒级的响应，同样显得非常费时。 故，我们可以在程序初始化时，将 tcp 的 socket 连接建立好， 并设置好超时重连。

### 4.5 服务器端压缩量大数据

如今的浏览器，大部分已支持接受压缩后的数据，在客户端解压后渲染。所以，对于数据量特别大的返回， 我们可以配置 http 服务器进行文本压缩，再进行传送。 这在用户网络环境不是很理想的条件下对页面响应速度也有很大改善。

### 4.6 动态内容静态化

前面提到过，静态页面不应该是整个网站的性能瓶颈。 所以，对于阅读偏向性页面，我们可以将动态页面生成静态内容以响应用户请求。

## 5 后记

以上便是作者在全缓存动态页面方面的一些体会，很少涉及到具体的实现方案。 其实只要在开发过程中注意尽量减少磁盘读写、 网络请求、和费时的系统调用。 便应该能够轻松满足日访问千万级的应用需求，并且能够获得不错的开发效率。

由于作者水平有限，文中难免有遗漏和错误之处。 你可以通过 bigmaliang@163.com 与作者取得联系予以指正。 有关互联网的其他探讨，也可以通过此邮箱与我取得联系。