<<构建高性能Web站点>>

图书基本信息

书名：<<构建高性能Web站点>>

13位ISBN编号：9787121093357

10位ISBN编号：7121093359

出版时间：2009 年

出版时间：电子工业出版社

作者：郭欣

页数：402

字数：608000

<<构建高性能Web站点>>

内容概要

本书围绕如何构建高性能Web站点，从多个方面、多个角度进行了全面的阐述，涵盖了Web站点性能优化的几乎所有内容，包括数据的网络传输、服务器并发处理能力、动态网页缓存、动态网页静态化、应用层数据缓存、分布式缓存、Web服务器缓存、反向代理缓存、脚本解释速度、页面组件分离、浏览器本地缓存、浏览器并发请求、文件的分发、数据库I/O优化、数据库访问、数据库分布式设计、负载均衡、分布式文件系统、性能监控等。

在这些内容中充分抓住本质并结合实践，通过通俗易懂的文字和生动有趣的配图，让读者充分并深入理解高性能架构的真相。

同时，本书充分应用跨学科知识和科学分析方法，通过宽泛的视野和独特的角度，将本书的内容展现得更加透彻和富有趣味。

作者简介

郭欣，曾在腾讯网基础平台研发团队，负责诸多Web应用的开发和技术管理，并致力于性能研究和实践推广。
在加入腾讯之前，获得国家系统分析师职称，目前在工作之余从事独立研究，其中包括高性能Web架构和Web敏捷开发框架，并且积极投身开源事业，同时在为Smart Developer系列进

<<构建高性能Web站点>>

书籍目录

章节摘录

第1章 绪论　　1.2 瓶颈在哪里　　相信你一定知道赤壁之战，这是中国历史上一场著名的以少胜多的战役，东吴的任务是击退曹操的进攻，要完成这项任务，可谓“万事俱备，只欠东风”，这时东风便是决胜的瓶颈，所以很多系统论研究专家将其称为“东风效应”，也就是社会心理学里讲的“瓶颈效应”。

之所以称它为瓶颈，是因为尽管东吴做了很多的战前准备，包括蒋干中计导致曹操错杀蔡瑁和张允、诸葛亮草船借箭、东吴苦练水军等，但是仅靠这些仍无法获得最终胜利，还需要最后的东南风才能一锤定音，完成火烧曹军战船的计划。

不过之前的准备工作都是胜利的子因素，而东南风这个关键因素最终和其他子因素一起相互作用，将整个战斗的杀伤力无限放大。

曹操运气不好，遇上东南风，倒了大霉，曹军战船一片火海，这时候东吴需要派出勇猛的陆军部队登岸攻下曹营，可是东吴向来精通水战，几乎没有强大的陆战部队，只有老将黄盖，这如何与曹操的精英骑兵抗衡呢？

这个时候决胜的关键因素变成了刘备的盟军支援，五虎上将各个威猛无比，身怀必杀绝技，此时正是上岸一显身手的好机会，他们不费吹灰之力就将曹军打得落花流水，试想如果没有刘备的支援，赤壁一战胜败可能就扑朔迷离了。

可见，系统性能的瓶颈，是指影响性能的关键因素，这个关键因素随着系统的运行又会发生不断的变化或迁移，比如由于站点用户组成结构的多样性和习惯的差异，导致在不同时段系统的瓶颈各不相同，又如站点在数据存储量或浏览量增长到不同级别时，系统瓶颈也会发生迁移。

一旦找到真正影响系统性能的主要因素，也就是性能瓶颈，就要坚决对其进行调整或优化，因为你不得不这么做。

提示：　　中医是一门关于生命的哲学，也是中国人智慧的结晶，它的光芒在于独到的思辨能力和系统性的分析方法，它认为世间万物都在不停地变化，并赋予它们阴阳状态，包括天地、季节、天气、心理、生理等，而患者的病理也在随之变化，所以，中医会对同一位患者在不同季节进行不同的诊断，找到不同的病因。

同时，在这些关键因素的背后，也存在很多不能忽略的子因素，构成了性能优化的“长尾效应”，也就是说如果你对某个子因素背后的问题进行优化，可能会带来性能上的少许提升，也许不被察觉，但是多个子因素的优化结果也许会叠加在一起，带来性能上可观的提升。

对于诸多子因素的优化，需要稍加谨慎，花点时间考虑这种优化是否值得，以及是否会带来潜在的副作用，还有其他依赖的非技术因素。

然而，不论是关键因素还是子因素，它们的背后都是影响系统性能的问题所在，问题本身并不涉及关键性，只有在不同的系统和应用场景下，才会显示出其是否关键。

本章的其余部分将先列出一些我们经常遇到的问题，并简单介绍我们常用的优化方案，至于这些问题在什么时候是否关键，它们的本质是什么，以及如何调整或优化，在后续章节中我们将结合具体场景来详细探讨包括这些在内的更多主题，这也是本书贯穿始终的线索。

1.3 增加带宽　　当Web站点的网页或组件的下载速度变慢时，一些架构师可能想到的最省事的办法就是增加服务器带宽，因为他们认为是服务器带宽不够用了，对于一些以提供下载服务为主的站点来说也许是这样的，但是对于其他服务的站点，你知道站点当前究竟使用了多少带宽吗？

这些带宽都用到哪里了呢？

如何计算站点现在和可预见未来使用的带宽？

带宽增加后下载速度就可以加快吗？

使用独享带宽和共享带宽的本质区别是什么？

如何节省带宽？

还有，你可能会忍无可忍地问，究竟什么是带宽？

对于带宽的概念，如果你没有仔细阅读计算机网络教材中的描述，我敢肯定你一定是完全凭借自己的理解来认识它的，因为这个词实在是太有创意了，也实在太容易从字面理解了，但是这些认识从

本质上讲是完全错误的，正是基于这种误解，很多人都无法完全解答上述那一连串问题，导致在所有涉及带宽的问题上，只能依靠经验和猜想。

在后续章节中，我们将通过介绍数据的网络传输原理，彻底揭开带宽的本质，以及数据传输响应时间的依赖因素和计算方法。

搞清楚这些一点都不困难，它们是一个优秀架构师必须掌握的基础知识。

1.4 减少网页中的HTTP请求　　我们知道Web站点中几乎任何一个网页都包含了多个组件，每个组件都需要下载、计算或渲染，毫无疑问，这些行为都会消耗时间。

那么如果可以让网页减少这些行为，应该就可以加快网页的展示速度，这是毫无疑问的，但是往往我们需要在优雅的网页表现和性能之间权衡取舍，这也许是美和快之间的博弈，找到最优的均衡点至关重要，我们为此做了很多尝试和努力：　　设计更加简单的网页，使其包含较少的图片和脚本，但是这可能牺牲了美观和用户交互。

将多个图片合并为一个文件，利用CSS背景图片的偏移技术呈现在网页中，避免了多个图片的下载。

合并JavaScript脚本或者CSS样式表：　　充分利用HTTP中的浏览器端Cache策略，减少重复下载。

很显然，这些技巧都来自于Web网页前端的优化，在后续章节中我们会有所涉及，但是不作为本书的重点来介绍，本书将更加偏重于站点服务器端的性能改善和规模扩展。

1.5 加快服务器脚本计算速度　　我想大多数涉及性能问题的站点都会使用各种各样的服务器端脚本语言，比如主流的PHP、Ruby、Python、ASP.NET、JSP等，这些脚本语言用来编写动态内容或者后台运行的小程序，已经成了几乎所有站点的首选。

而曾经使用C++编写动态内容的经历也让我记忆犹新，除了每天都在感叹c++的严谨和优雅之外，我找不到其他任何好处。

我们知道，用这些脚本语言编写的程序文件需要通过相应的脚本解释器进行解释后生成中间代码，然后依托在解释器的运行环境中运行。

所以生成中间代码的这部分时间又成为大家为获取性能提升而瞄准的一个目标，对于一些拥有较强商业支持的脚本语言，比如ASENET和JSP，均有内置的优化方案，比如解释器对某个脚本程序第一次解释的时候，将中间代码缓存起来，以供下次直接使用。

对于开源类的脚本语言，也有很多第三方组件来提供此类功能，比如PHP的APC组件等。

使用这些组件进行脚本优化真的那么有用吗？

不同的应用效果是否有所不同呢？

在后续章节中我们会详细探讨。

1.6 使用动态内容缓存　　动态内容技术就像Web开发领域的一场工业革命，它带来了产业升级和Web开发者的地位提升，在过去相当长一段时间里，大家普遍认为一个站点的技术含量主要体现在后台的动态程序上，所以很多工程师都会带着虚荣心警告你：“请叫我后台开发工程师。”事实上这种概念和偏见已经开始逐渐被历史抛弃，但这不是我们此刻讨论的重点。

自动态内容技术产生后，聪明的工程师们为了减少动态内容的重复计算，想到了截取动态内容的胜利果实，将动态内容的HTML输出结果缓存起来，在随后的一段时间内当有用户访问时便跳过重复的动态内容计算而直接输出。

在实际应用中，动态内容缓存可能是大家使用得最多的技术，但是并不见得所有的动态内容都适合使用网页缓存，缓存带来的性能提升恰恰与有些动态数据实时交互的需求形成矛盾，这是非常尴尬的，而解决该问题的唯一途径不是技术本身，而是你如何权衡。

另一方面，缓存的实现还涉及了一系列非常现实的问题，即成千上万的缓存文件如何存储？

缓存的命中率如何？

缓存的过期策略如何设计？

在拥有多台Web服务器的分布式站点上应用动态内容缓存需要考虑什么呢？

在后续章节中我们将详细地探讨这些问题。

1.7 使用数据缓存　　动态内容缓存是将数据和表现整体打包，一步到位，但就像快餐店里的组合

套餐一样，有时候未必完全合乎我们的口味。
当我们意识到在自己的站点中，某些动态内容的计算时间其实主要消耗在一些烦人的特殊数据上，这些数据或者更新过于频繁，或者消耗大量的I/O等待时间，比如对关系数据库中某字段的频繁更新和读取，这时我们为了提高缓存的灵活性和命中率，以及性能的要求，便开始考虑数据缓存。

更加细粒度的数据缓存避免了过期时大量相关网页的整体更新，比如很多动态内容都包含了一段公用的数据，如果我们将整个页面全部缓存，那么假如这段数据频繁更新导致频繁过期，无疑会使得所有网页都要频繁地重建缓存，这对网页的其他部分内容似乎很不公平。

……

<<构建高性能Web站点>>

版权说明