Broadview®
www.broadview.com.cn

大型网站技术架构

核心原理与案例分析

李智慧 著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

内 容 简 介

本书通过梳理大型网站技术发展历程，剖析大型网站技术架构模式，深入讲述大型互联网架构设计的核心原理，并通过一组典型网站技术架构设计案例，为读者呈现一幅包括技术选型、架构设计、性能优化、Web 安全、系统发布、运维监控等在内的大型网站开发全景视图。

本书不仅适用于指导网站工程师、架构师进行网站技术架构设计，也可用于指导产品经理、项目经理、测试运维人员等了解网站技术架构的基础概念；还可供包括企业系统开发人员在内的各类软件开发从业人员借鉴，了解大型网站的解决方案和开发理念。

图书在版编目（CIP）数据

大型网站技术架构：核心原理与案例分析 / 李智慧著. —北京：电子工业出版社，2013.9
ISBN 978-7-121-21200-0

Ⅰ. ①大… Ⅱ. ①李… Ⅲ. ①网站－建设 Ⅳ.①TP393.092

中国版本图书馆 CIP 数据核字(2013)第 182399 号

责任编辑：徐津平
印　　刷：三河市双峰印刷装订有限公司
装　　订：三河市双峰印刷装订有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编：100036
开　　本：720×1000　1/16　印张：15　字数：240 千字
印　　次：2013 年 9 月第 1 次印刷
印　　数：4000 册　　定价：59.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

2 大型网站架构模式

关于什么是模式，这个来自建筑学的词汇是这样定义的：“每一个模式描述了一个在我们周围不断重复发生的问题及该问题解决方案的核心。这样，你就能一次又一次地使用该方案而不必做重复工作”。模式的关键在于模式的可重复性，问题与场景的可重复性带来解决方案的可重复使用。

> 我们的现实生活中充斥着几乎千篇一律的人生架构模式：读重点学校，选热门专业，进稳定高收入的政府部门和企业，找门当户对的配偶，生一个听话的孩子继续这个模式……但是人生不同于软件，精彩的人生绝不会来自于复制。

也许互联网产品不是随便复制就能成功的，创新的产品更能为用户创造价值。但是网站架构却有一些共同的模式，这些模式已经被许多大型网站一再验证，通过对这些模式的学习，我们可以掌握大型网站架构的一般思路和解决方案，以指导我们的架构设计。

2.1 网站架构模式

为了解决大型网站面临的高并发访问、海量数据处理、高可靠运行等一系列问题与挑战，大型互联网公司在实践中提出了许多解决方案，以实现网站高性能、高可用、易伸缩、可扩展、安全等各种技术架构目标。这些解决方案又被更多网站重复使用，从而逐渐形成大型网站架构模式。

2.1.1 分层

分层是企业应用系统中最常见的一种架构模式，将系统在横向维度上切分成几个部分，每个部分负责一部分相对比较单一的职责，然后通过上层对下层的依赖和调用组成一个完整的系统。

分层结构在计算机世界中无处不在，网络的 7 层通信协议是一种分层结构；计算机硬件、操作系统、应用软件也可以看作是一种分层结构。在大型网站架构中也采用分层结构，将网站软件系统分为应用层、服务层、数据层，如表 2.1 所示。

表 2.1 网站分层架构

应用层	负责具体业务和视图展示，如网站首页及搜索输入和结果展示

服务层	为应用层提供服务支持，如用户管理服务，购物车服务等
数据层	提供数据存储访问服务，如数据库、缓存、文件、搜索引擎等

通过分层，可以更好地将一个庞大的软件系统切分成不同的部分，便于分工合作开发和维护；各层之间具有一定的独立性，只要维持调用接口不变，各层可以根据具体问题独立演化发展而不需要其他层必须做出相应调整。

但是分层架构也有一些挑战，就是必须合理规划层次边界和接口，在开发过程中，严格遵循分层架构的约束，禁止跨层次的调用（应用层直接调用数据层）及逆向调用（数据层调用服务层，或者服务层调用应用层）。

在实践中，大的分层结构内部还可以继续分层，如应用层可以再细分为视图层（美工负责）和业务逻辑层（工程师负责）；服务层也可以细分为数据接口层（适配各种输入和输出的数据格式）和逻辑处理层。

分层架构是逻辑上的，在物理部署上，三层结构可以部署在同一个物理机器上，但是随着网站业务的发展，必然需要对已经分层的模块分离部署，即三层结构分别部署在不同的服务器上，使网站拥有更多的计算资源以应对越来越多的用户访问。

所以虽然分层架构模式最初的目的是规划软件清晰的逻辑结构便于开发维护，但在网站的发展过程中，分层结构对网站支持高并发向分布式方向发展至关重要。因此在网站规模还很小的时候就应该采用分层的架构，这样将来网站做大时才能有更好地应对。

2.1.2 分割

如果说分层是将软件在横向方面进行切分，那么分割就是在纵向方面对软件进行切分。

网站越大，功能越复杂，服务和数据处理的种类也越多，将这些不同的功能和服务分割开来，包装成高内聚低耦合的模块单元，一方面有助于软件的开发和维护；另一方面，便于不同模块的分布式部署，提高网站的并发处理能力和功能扩展能力。

大型网站分割的粒度可能会很小。比如在应用层，将不同业务进行分割，例如将购物、论坛、搜索、广告分割成不同的应用，由独立的团队负责，部署在不同的服务器上；在同一个应用内部，如果规模庞大业务复杂，会继续进行分割，比如购物业务，可以进一步分割成机票酒店业务、3C 业务，小商品业务等更细小的粒度。而即使在这个粒度上，还是可以继续分割成首页、搜索列表、商品详情等模块，这些模块不管在逻辑上还是物理部署上，都可以是独立的。同样在服务层也可以根据需要将服务分割成合适的模块。

2.1.3 分布式

对于大型网站，分层和分割的一个主要目的是为了切分后的模块便于分布式部署，即将不同模块部署在不同的服务器上，通过远程调用协同工作。分布式意味着可以使用更多的计算机完成同样的功能，计算机越多，CPU、内存、存储资源也就越多，能够处理的并发访问和数据量就越大，进而能够为更多的用户提供服务。

但分布式在解决网站高并发问题的同时也带来了其他问题。首先，分布式意味着服务调用必须通过网络，这可能会对性能造成比较严重的影响；其次，服务器越多，服务器宕机的概率也就越大，一

台服务器宕机造成的服务不可用可能会导致很多应用不可访问，使网站可用性降低；另外，数据在分布式的环境中保持数据一致性也非常困难，分布式事务也难以保证，这对网站业务正确性和业务流程有可能造成很大影响；分布式还导致网站依赖错综复杂，开发管理维护困难。因此分布式设计要根据具体情况量力而行，切莫为了分布式而分布式。

在网站应用中，常用的分布式方案有以下几种。

分布式应用和服务：将分层和分割后的应用和服务模块分布式部署，除了可以改善网站性能和并发性、加快开发和发布速度、减少数据库连接资源消耗外；还可以使不同应用复用共同的服务，便于业务功能扩展。

分布式静态资源：网站的静态资源如 JS，CSS，Logo 图片等资源独立分布式部署，并采用独立的域名，即人们常说的动静分离。静态资源分布式部署可以减轻应用服务器的负载压力；通过使用独立域名加快浏览器并发加载的速度；由负责用户体验的团队进行开发维护有利于网站分工合作，使不同技术工种术业有专攻。

分布式数据和存储：大型网站需要处理以 P 为单位的海量数据，单台计算机无法提供如此大的存储空间，这些数据需要分布式存储。除了对传统的关系数据库进行分布式部署外，为网站应用而生的各种 NoSQL 产品几乎都是分布式的。

分布式计算：严格说来，应用、服务、实时数据处理都是计算，网站除了要处理这些在线业务，还有很大一部分用户没有直观感受的后台业务要处理，包括搜索引擎的索引构建、数据仓库的数据分析统计等。这些业务的计算规模非常庞大，目前网站普遍使用 Hadoop 及其 MapReduce 分布式计算框架进行此类批处理计算，其特点是移动计算而不是移动数据，将计算程序分发到数据所在的位置以加速计算和分布式计算。

此外，还有可以支持网站线上服务器配置实时更新的分布式配置；分布式环境下实现并发和协同的分布式锁；支持云存储的分布式文件系统等。

2.1.4 集群

使用分布式虽然已经将分层和分割后的模块独立部署，但是对于用户访问集中的模块（比如网站的首页），还需要将独立部署的服务器集群化，即多台服务器部署相同应用构成一个集群，通过负载均衡设备共同对外提供服务。

因为服务器集群有更多服务器提供相同服务，因此可以提供更好的并发特性，当有更多用户访问的时候，只需要向集群中加入新的机器即可。同时因为一个应用由多台服务器提供，当某台服务器发生故障时，负载均衡设备或者系统的失效转移机制会将请求转发到集群中其他服务器上，使服务器故障不影响用户使用。所以在网站应用中，即使是访问量很小的分布式应用和服务，也至少要部署两台服务器构成一个小的集群，目的就是提高系统的可用性。

2.1.5 缓存

缓存就是将数据存放在距离计算最近的位置以加快处理速度。缓存是改善软件性能的第一手段，现代 CPU 越来越快的一个重要因素就是使用了更多的缓存，在复杂的软件设计中，缓存几乎无处不在。大型网站架构设计在很多方面都使用了缓存设计。

CDN：即内容分发网络，部署在距离终端用户最近的网络服务商，用户的网络请求总是先到达他的网络服务商那里，在这里缓存网站的一些静态资源（较少变化的数据），可以就近以最快速度返回给用户，如视频网站和门户网站会将用户访问量大的热点内容缓存在 CDN。

反向代理：反向代理属于网站前端架构的一部分，部署在网站的前端，当用户请求到达网站的数据中心时，最先访问到的就是反向代理服务器，这里缓存网站的静态资源，无需将请求继续转发给应用服务器就能返回给用户。

本地缓存：在应用服务器本地缓存着热点数据，应用程序可以在本机内存中直接访问数据，而无需访问数据库。

分布式缓存：大型网站的数据量非常庞大，即使只缓存一小部分，需要的内存空间也不是单机能承受的，所以除了本地缓存，还需要分布式缓存，将数据缓存在一个专门的分布式缓存集群中，应用程序通过网络通信访问缓存数据。

使用缓存有两个前提条件，一是数据访问热点不均衡，某些数据会被更频繁的访问，这些数据应该放在缓存中；二是数据在某个时间段内有效，不会很快过期，否则缓存的数据就会因已经失效而产生脏读，影响结果的正确性。网站应用中，缓存除了可以加快数据访问速度，还可以减轻后端应用和数据存储的负载压力，这一点对网站数据库架构至关重要，网站数据库几乎都是按照有缓存的前提进行负载能力设计的。

2.1.6 异步

计算机软件发展的一个重要目标和驱动力是降低软件耦合性。事物之间直接关系越少，就越少被彼此影响，越可以独立发展。大型网站架构中，系统解耦合的手段除了前面提到的分层、分割、分布等，还有一个重要手段是异步，业务之间的消息传递不是同步调用，而是将一个业务操作分成多个阶段，每个阶段之间通过共享数据的方式异步执行进行协作。

在单一服务器内部可通过多线程共享内存队列的方式实现异步，处在业务操作前面的线程将输出写入到队列，后面的线程从队列中读取数据进行处理；在分布式系统中，多个服务器集群通过分布式消息队列实现异步，分布式消息队列可以看作内存队列的分布式部署。

异步架构是典型的生产者消费者模式，两者不存在直接调用，只要保持数据结构不变，彼此功能实现可以随意变化而不互相影响，这对网站扩展新功能非常便利。除此之外，使用异步消息队列还有如下特性。

提高系统可用性。消费者服务器发生故障，数据会在消息队列服务器中存储堆积，生产者服务器可以继续处理业务请求，系统整体表现无故障。消费者服务器恢复正常后，继续处理消息队列中的数据。

加快网站响应速度。处在业务处理前端的生产者服务器在处理完业务请求后，将数据写入消息队列，不需要等待消费者服务器处理就可以返回，响应延迟减少。

消除并发访问高峰。用户访问网站是随机的，存在访问高峰和低谷，即使网站按照一般访问高峰进行规划和部署，也依然会出现突发事件，比如购物网站的促销活动，微博上的热点事件，都会造成网站并发访问突然增大，这可能会造成整个网站负载过重，响应延迟，严重时甚至会出现服务宕机的

情况。使用消息队列将突然增加的访问请求数据放入消息队列中，等待消费者服务器依次处理，就不会对整个网站负载造成太大压力。

但需要注意的是，使用异步方式处理业务可能会对用户体验、业务流程造成影响，需要网站产品设计方面的支持。

2.1.7 冗余

网站需要 7×24 小时连续运行，但是服务器随时可能出现故障，特别是服务器规模比较大时，出现某台服务器宕机是必然事件。要想保证在服务器宕机的情况下网站依然可以继续服务，不丢失数据，就需要一定程度的服务器冗余运行，数据冗余备份，这样当某台服务器宕机时，可以将其上的服务和数据访问转移到其他机器上。

访问和负载很小的服务也必须部署至少两台服务器构成一个集群，其目的就是通过冗余实现服务高可用。数据库除了定期备份，存档保存，实现冷备份外，为了保证在线业务高可用，还需要对数据库进行主从分离，实时同步实现热备份。

为了抵御地震、海啸等不可抗力导致的网站完全瘫痪，某些大型网站会对整个数据中心进行备份，全球范围内部署灾备数据中心。网站程序和数据实时同步到多个灾备数据中心。

2.1.8 自动化

在无人值守的情况下网站可以正常运行，一切都可以自动化是网站的理想状态。目前大型网站的自动化架构设计主要集中在发布运维方面。

发布对网站都是头等大事，许多网站故障出在发布环节，网站工程师经常加班也是因为发布不顺利。通过减少人为干预，使发布过程自动化可有效减少故障。发布过程包括诸多环节。自动化代码管理，代码版本控制、代码分支创建合并等过程自动化，开发工程师只要提交自己参与开发的产品代号，系统就会自动为其创建开发分支，后期会自动进行代码合并；自动化测试，代码开发完成，提交测试后，系统自动将代码部署到测试环境，启动自动化测试用例进行测试，向相关人员发送测试报告，向系统反馈测试结果；自动化安全检测，安全检测工具通过对代码进行静态安全扫描及部署到安全测试环境进行安全攻击测试，评估其安全性；最后进行自动化部署，将工程代码自动部署到线上生产环境。

此外，网站在运行过程中可能会遇到各种问题：服务器宕机、程序 Bug、存储空间不足、突然爆发的访问高峰。网站需要对线上生产环境进行自动化监控，对服务器进行心跳检测，并监控其各项性能指标和应用程序的关键数据指标。如果发现异常、超出预设的阈值，就进行自动化报警，向相关人员发送报警信息，警告故障可能会发生。在检测到故障发生后，系统会进行自动化失效转移，将失效的服务器从集群中隔离出去，不再处理系统中的应用请求。待故障消除后，系统进行自动化失效恢复，重新启动服务，同步数据保证数据的一致性。在网站遇到访问高峰，超出网站最大处理能力时，为了保证整个网站的安全可用，还会进行自动化降级，通过拒绝部分请求及关闭部分不重要的服务将系统负载降至一个安全的水平，必要时，还需要自动化分配资源，将空闲资源分配给重要的服务，扩大其部署规模。

2.1.9 安全

互联网的开放特性使得其从诞生起就面对巨大的安全挑战，网站在安全架构方面也积累了许多模

式：通过密码和手机校验码进行身份认证；登录、交易等操作需要对网络通信进行加密，网站服务器上存储的敏感数据如用户信息等也进行加密处理；为了防止机器人程序滥用网络资源攻击网站，网站使用验证码进行识别；对于常见的用于攻击网站的 XSS 攻击、SQL 注入、进行编码转换等相应处理；对于垃圾信息、敏感信息进行过滤；对交易转账等重要操作根据交易模式和交易信息进行风险控制。

2.2 架构模式在新浪微博的应用

短短几年时间新浪微博的用户数就从零增长到数亿，明星用户的粉丝数达数千万，围绕着新浪微博正在发展一个集社交、媒体、游戏、电商等多位一体的生态系统。

同大多数网站一样，新浪微博也是从一个小网站发展起来的。简单的 LAMP（Linux+Apache+MySQL+PHP）架构，支撑起最初的新浪微博，应用程序用 PHP 开发，所有的数据，包括微博、用户、关系都存储在 MySQL 数据库中。

这样简单的架构无法支撑新浪微博快速发展的业务需求，随着访问用户的逐渐增加，系统不堪重负。新浪微博的架构在较短时间内几经重构，最后形成现在的架构，如图 2.1 所示。

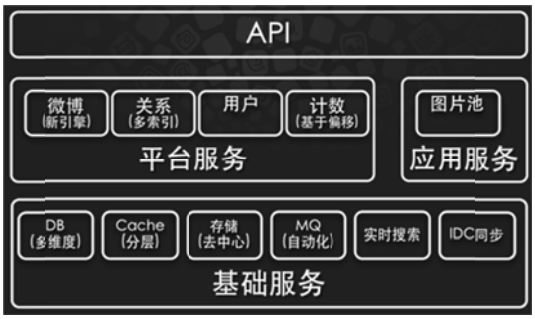

图 2.1　新浪微博的系统架构

（图片来源：http://timyang.net/architecture/weibo/）

系统分为三个层次，最下层是基础服务层，提供数据库、缓存、存储、搜索等数据服务，以及其他一些基础技术服务，这些服务支撑了新浪微博的海量数据和高并发访问，是整个系统的技术基础。

中间层是平台服务和应用服务层，新浪微博的核心服务是微博、关系和用户，它们是新浪微博业务大厦的支柱。这些服务被分割为独立的服务模块，通过依赖调用和共享基础数据构成新浪微博的业务基础。

最上层是 API 和新浪微博的业务层，各种客户端（包括 Web 网站）和第三方应用，通过调用 API 集成到新浪微博的系统中，共同组成一个生态系统。

这些被分层和分割后的业务模块与基础技术模块分布式部署，每个模块都部署在一组独立的服务器集群上，通过远程调用的方式进行依赖访问。新浪微博在早期还使用过一种叫作 MPSS（MultiPort Single Server，单服务器多端口）的分布式集群部署方案，在集群中的多台服务器上，每台都部署多个服务，每个服务使用不同的端口对外提供服务，通过这种方式使得有限的服务器可以部署更多的服务实例，改善服务的负载均衡和可用性。现在网站应用中常见的将物理机虚拟化成多个虚拟机后，在虚拟机上部署应用的方案跟新浪微博的 MPSS 方案异曲同工，只是更加简单，还能在不同虚拟机上使用相同的端口号。

在新浪微博的早期架构中，微博发布使用同步推模式，用户发表微博后系统会立即将这条微博插入到数据库所有粉丝的订阅列表中，当用户量比较大时，特别是明星用户发布微博时，会引起大量的数据库写操作，超出数据库负载，系统性能急剧下降，用户响应延迟加剧。后来新浪微博改用异步推拉结合的模式，用户发表微博后系统将微博写入消息队列后立即返回，用户响应迅速，消息队列消费者任务将微博推送给所有当前在线粉丝的订阅列表中，非在线用户登录后再根据关注列表拉取微博订阅列表。

由于微博频繁刷新，新浪微博使用多级缓存策略，热门微博和明星用户的微博缓存在所有的微博服务器上，在线用户的微博和近期微博缓存在分布式缓存集群中，对于微博操作中最常见的“刷微博”操作，几乎全部都是缓存访问操作，可以获得很好的系统性能。

为了提高系统的整体可用性和性能，新浪微博启用了多个数据中心。这些数据中心既是地区用户访问中心，用户可以就近访问最近的数据中心以加快访问速度，改善系统性能；同时也是数据冗余复制的灾备中心，所有的用户和微博数据通过远程消息系统在不同的数据中心之间同步，提高系统可用性。

同时，新浪微博还开发了一系列自动化工具，包括自动化监控，自动化发布，自动化故障修复等，这些自动化工具还在持续开发中，以改善运维水平提高系统可用性。

由于微博的开放特性，新浪微博也遇到了一系列的安全挑战，垃圾内容、僵尸粉、微博攻击从未停止，除了使用一般网站常见的安全策略，新浪微博在开放平台上使用多级安全审核的策略以保护系统和用户。

2.3 小结

在程序设计与架构设计领域，模式正变得越来越受人关注，许多人寄希望通过模式一劳永逸地解决自己的问题。正确使用模式可以更好地利用业界和前人的思想与实践，用更少的时间开发出更好的系统，使设计者的水平也达到更高的境界。但是模式受其适用场景限制，对系统的要求和约束也很多，不恰当地使用模式只会画虎不成反类犬，不但没有解决原来的老问题，反而带来了更棘手的新问题。

好的设计绝对不是模仿，不是生搬硬套某个模式，而是对问题深刻理解之上的创造与创新，即使是“微创新”，也是让人耳目一新的似曾相识。山寨与创新的最大区别不在于是否抄袭，是否模仿，而在于对问题和需求是否真正理解与把握。

4

瞬时响应：网站的高性能架构

什么叫高性能的网站？

两个网站性能架构设计方案：A 方案和 B 方案，A 方案在小于 100 个并发用户访问时，每个请求的响应时间是 1 秒，当并发请求达到 200 的时候，请求的响应时间将骤增到 10 秒。B 方案不管是 100 个并发用户访问还是 200 个并发用户访问，每个请求的响应时间都差不多是 1.5 秒。哪个方案的性能好？如果老板说“我们要改善网站的性能”，他指的是什么？

同类型的两个网站，X 网站服务器平均每个请求的处理时间是 500 毫秒，Y 网站服务器平均每个请求的处理时间是 1000 毫秒，为什么用户却反映 Y 网站的速度快呢？

网站性能是客观的指标，可以具体体现到响应时间、吞吐量等技术指标，同时也是主观的感受，而感受则是一种与具体参与者相关的微妙的东西，用户的感受和工程师的感受不同，不同的用户感受也不同。

4.1 网站性能测试

性能测试是性能优化的前提和基础，也是性能优化结果的检查和度量标准。不同视角下的网站性能有不同的标准，也有不同的优化手段。

4.1.1 不同视角下的网站性能

软件工程师说到网站性能的时候，通常和用户说的不一样。

1．用户视角的网站性能

从用户角度，网站性能就是用户在浏览器上直观感受到的网站响应速度快还是慢。用户感受到的时间，包括用户计算机和网站服务器通信的时间、网站服务器处理的时间、用户计算机浏览器构造请求解析响应数据的时间，如图 4.1 所示。

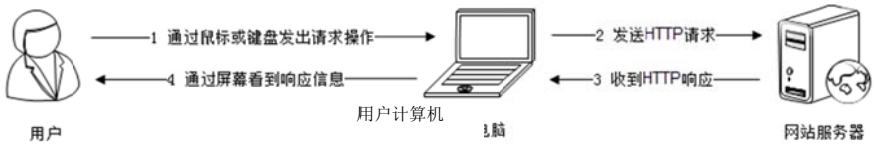

图 4.1　用户视角的网站性能

不同计算机的性能差异，不同浏览器解析 HTML 速度的差异，不同网络运营商提供的互联网宽带服务的差异，这些差异最终导致用户感受到的响应延迟可能会远远大于网站服务器处理请求需要的时间。

在实践中，使用一些前端架构优化手段，通过优化页面 HTML 式样、利用浏览器端的并发和异步特性、调整浏览器缓存策略、使用 CDN 服务、反向代理等手段，使浏览器尽快地显示用户感兴趣的内容、尽可能近地获取页面内容，即使不优化应用程序和架构，也可以很大程度地改善用户视角下的网站性能。

2．开发人员视角的网站性能

开发人员关注的主要是应用程序本身及其相关子系统的性能，包括响应延迟、系统吞吐量、并发处理能力、系统稳定性等技术指标。主要的优化手段有使用缓存加速数据读取，使用集群提高吞吐能力，使用异步消息加快请求响应及实现削峰，使用代码优化手段改善程序性能。

3．运维人员视角的网站性能

运维人员更关注基础设施性能和资源利用率，如网络运营商的带宽能力、服务器硬件的配置、数据中心网络架构、服务器和网络带宽的资源利用率等。主要优化手段有建设优化骨干网、使用高性价比定制服务器、利用虚拟化技术优化资源利用等。

4.1.2 性能测试指标

不同视角下有不同的性能标准，不同的标准有不同的性能测试指标，从开发和测试人员的视角，网站性能测试的主要指标有响应时间、并发数、吞吐量、性能计数器等。

1．响应时间

指应用执行一个操作需要的时间，包括从发出请求开始到收到最后响应数据所需要的时间。响应时间是系统最重要的性能指标，直观地反映了系统的“快慢”。表 4.1 列出了一些常用的系统操作需要的响应时间。

表 4.1　常用系统操作响应时间表

操　　作	响应时间
打开一个网站	几秒
在数据库中查询一条记录（有索引）	十几毫秒
机械磁盘一次寻址定位	4 毫秒
从机械磁盘顺序读取 1MB 数据	2 毫秒
从 SSD 磁盘顺序读取 1MB 数据	0.3 毫秒
从远程分布式缓存 Redis 读取一个数据	0.5 毫秒
从内存中读取 1MB 数据	十几微秒
Java 程序本地方法调用	几微秒
网络传输 2KB 数据	1 微秒

（部分数据来源：http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html）

测试程序通过模拟应用程序，记录收到响应和发出请求之间的时间差来计算系统响应时间。但是记录及获取系统时间这个操作也需要花费一定的时间，如果测试目标操作本身需要花费的时间极少，比如几微秒，那么测试程序就无法测试得到系统的响应时间。实践中通常采用的办法是重复请求，比如一个请求操作重复执行一万次，测试一万次执行需要的总响应时间之和，然后除以一万，得到单次请求的响应时间。

2．并发数

指系统能够同时处理请求的数目，这个数字也反映了系统的负载特性。对于网站而言，并发数即网站并发用户数，指同时提交请求的用户数目。

与网站并发用户数相对应的还有网站在线用户数（当前登录网站的用户总数）和网站系统用户数（可能访问系统的总用户数，对多数网站而言就是注册用户数）。其数量比较关系为：

网站系统用户数>>网站在线用户数>>网站并发用户数

在网站产品设计初期，产品经理和运营人员就需要规划不同发展阶段的网站系统用户数，并以此为基础，根据产品特性和运营手段，推算在线用户数和并发用户数。这些指标将成为系统非功能设计的重要依据。

现实中，经常看到某些网站，特别是电商类网站，市场推广人员兴致勃勃地打广告打折促销，用户兴致勃勃地去抢购，结果活动刚一开始，就因为并发用户数超过网站最大负载而响应缓慢，急性子的用户不停刷新浏览器，导致系统并发数更高，最后以服务器系统崩溃，用户浏览器显示“Service is too busy”而告终。出现这种情况，有可能是网站技术准备不充分导致，也有可能是运营人员错误地评估并发用户数导致。

测试程序通过多线程模拟并发用户的办法来测试系统的并发处理能力，为了真实模拟用户行为，测试程序并不是启动多线程然后不停地发送请求，而是在两次请求之间加入一个随机等待时间，这个时间被称作思考时间。

3．吞吐量

指单位时间内系统处理的请求数量，体现系统的整体处理能力。对于网站，可以用“请求数/秒”或是“页面数/秒”来衡量，也可以用“访问人数/天”或是“处理的业务数/小时”等来衡量。TPS（每秒事务数）是吞吐量的一个常用量化指标，此外还有 HPS（每秒 HTTP 请求数）、QPS（每秒查询数）等。

在系统并发数由小逐渐增大的过程中（这个过程也伴随着服务器系统资源消耗逐渐增大），系统吞吐量先是逐渐增加，达到一个极限后，随着并发数的增加反而下降，达到系统崩溃点后，系统资源耗尽，吞吐量为零。

而这个过程中，响应时间则是先保持小幅上升，到达吞吐量极限后，快速上升，到达系统崩溃点后，系统失去响应。系统吞吐量、系统并发数及响应时间之间的关系将在本章后面内容中介绍。

系统吞吐量和系统并发数，以及响应时间的关系可以形象地理解为高速公路的通行状况：吞吐量是每天通过收费站的车辆数目（可以换算成收费站收取的高速费），并发数是高速公路上的正在行驶的车辆数目，响应时间是车速。车辆很少时，车速很快，但是收到的高速费也相应较少；随着高速公路上车辆数目的增多，车速略受影响，但是收到的高速费增加很快；随着车辆的继续增加，车速变得越来越慢，高速公路越来越堵，收费不增反降；如果车流量继续增加，超过某个极限后，任何偶然因素都会导致高速全部瘫痪，车走不动，费当然也收不着，而高速公路成了停车场（资源耗尽）。

网站性能优化的目的，除了改善用户体验的响应时间，还要尽量提高系统吞吐量，最大限度利用服务器资源。

4．性能计数器

它是描述服务器或操作系统性能的一些数据指标。包括 System Load、对象与线程数、内存使用、CPU 使用、磁盘与网络 I/O 等指标。这些指标也是系统监控的重要参数，对这些指标设置报警阈值，当监控系统发现性能计数器超过阈值时，就向运维和开发人员报警，及时发现处理系统异常。

System Load 即系统负载，指当前正在被 CPU 执行和等待被 CPU 执行的进程数目总和，是反映系统忙闲程度的重要指标。多核 CPU 的情况下，完美情况是所有 CPU 都在使用，没有进程在等待处理，所以 Load 的理想值是 CPU 的数目。当 Load 值低于 CPU 数目的时候，表示 CPU 有空闲，资源存在浪费；当 Load 值高于 CPU 数目的时候，表示进程在排队等待 CPU 调度，表示系统资源不足，影响应用程序的执行性能。在 Linux 系统中使用 top 命令查看，该值是三个浮点数，表示最近 1 分钟，10 分钟，15 分钟的运行队列平均进程数。如图 4.2 所示。

```
top - 16:36:49 up 1 day,  5:53,  7 users,  load average: 0.14, 0.20, 0.16
```

图 4.2　在 Linux 命令行查看系统负载

4.1.3　性能测试方法

性能测试是一个总称，具体可细分为性能测试、负载测试、压力测试、稳定性测试。

性能测试

以系统设计初期规划的性能指标为预期目标，对系统不断施加压力，验证系统在资源可接受范围

内，是否能达到性能预期。

负载测试

对系统不断地增加并发请求以增加系统压力，直到系统的某项或多项性能指标达到安全临界值，如某种资源已经呈饱和状态，这时继续对系统施加压力，系统的处理能力不但不能提高，反而会下降。

压力测试

超过安全负载的情况下，对系统继续施加压力，直到系统崩溃或不能再处理任何请求，以此获得系统最大压力承受能力。

稳定性测试

被测试系统在特定硬件、软件、网络环境条件下，给系统加载一定业务压力，使系统运行一段较长时间，以此检测系统是否稳定。在不同生产环境、不同时间点的请求压力是不均匀的，呈波浪特性，因此为了更好地模拟生产环境，稳定性测试也应不均匀地对系统施加压力。

性能测试是一个不断对系统增加访问压力，以获得系统性能指标、最大负载能力、最大压力承受能力的过程。所谓的增加访问压力，在系统测试环境中，就是不断增加测试程序的并发请求数，一般说来，性能测试遵循如图 4.3 所示的抛物线规律。

图 4.3 中的横坐标表示消耗的系统资源，纵坐标表示系统处理能力（吞吐量）。在开始阶段，随着并发请求数目的增加，系统使用较少的资源就达到较好的处理能力（a～b 段），这一段是网站的日常运行区间，网站的绝大部分访问负载压力都集中在这一段区间，被称作性能测试，测试目标是评估系统性能是否符合需求及设计目标；随着压力的持续增加，系统处理能力增加变缓，直到达到一个最大值（c 点），这是系统的最大负载点，这一段被称作负载测试。测试目标是评估当系统因为突发事件超出日常访问压力的情况下，保证系统正常运行情况下能够承受的最大访问负载压力；超过这个点后，再增加压力，系统的处理能力反而下降，而资源消耗却更多，直到资源消耗达到极限（d 点），这个点可以看作是系统的崩溃点，超过这个点继续加大并发请求数目，系统不能再处理任何请求，这一段被称作压力测试，测试目标是评估可能导致系统崩溃的最大访问负载压力。

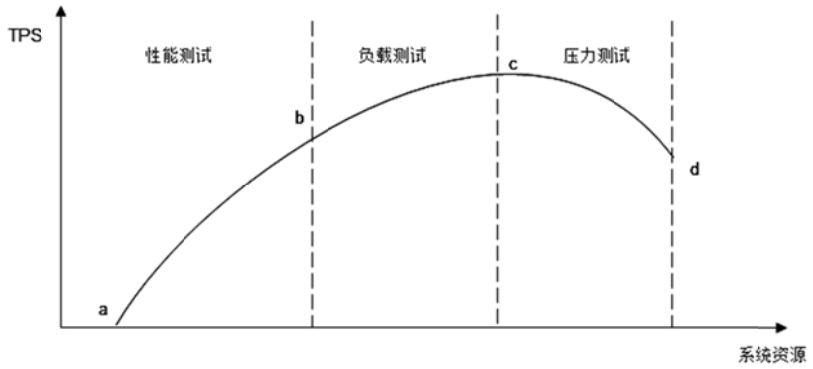

图 4.3　性能测试曲线

性能测试反应的是系统在实际生产环境中使用时，随着用户并发访问数量的增加，系统的处理能力。与性能曲线相对应的是用户访问的等待时间（系统响应时间），如图 4.4 所示。

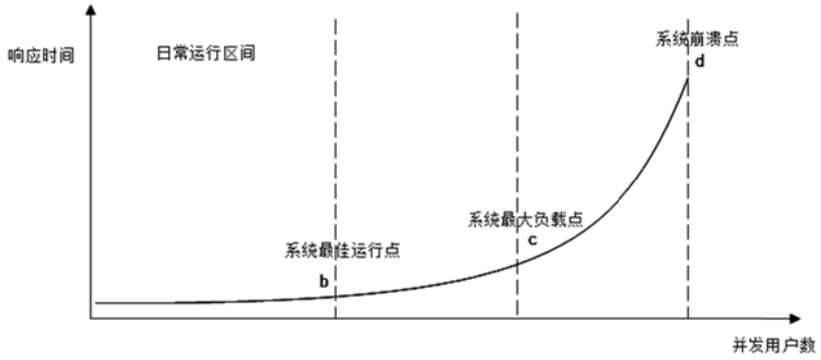

图 4.4　并发用户访问响应时间曲线

在日常运行区间，可以获得最好的用户响应时间，随着并发用户数的增加，响应延迟越来越大，直到系统崩溃，用户失去响应。

4.1.4　性能测试报告

测试结果报告应能够反映上述性能测试曲线的规律，阅读者可以得到系统性能是否满足设计目标和业务要求、系统最大负载能力、系统最大压力承受能力等重要信息 ,表 4.2 是一个简单示例。

表 4.2　性能测试结果报告

并 发 数	响应时间（ms）	TPS	错误率（%）	Load	内 存（GB）	备　　注
10	500	20	0	5	8	性能测试
20	800	30	0	10	10	性能测试
30	1000	40	2	15	14	性能测试
40	1200	45	20	30	16	负载测试
60	2000	30	40	50	16	压力测试
80	超时	0	100	不详	不详	压力测试

4.1.5　性能优化策略

如果性能测试结果不能满足设计或业务需求，那么就需要寻找系统瓶颈，分而治之，逐步优化。

1．性能分析

大型网站结构复杂，用户从浏览器发出请求直到数据库完成操作事务，中间需要经过很多环节，如果测试或者用户报告网站响应缓慢，存在性能问题，必须对请求经历的各个环节进行分析，排查可能出现性能瓶颈的地方，定位问题。

排查一个网站的性能瓶颈和排查一个程序的性能瓶颈的手法基本相同 :检查请求处理的各个环节的日志，分析哪个环节响应时间不合理、超过预期；然后检查监控数据，分析影响性能的主要因素是内存、磁盘、网络、还是 CPU，是代码问题还是架构设计不合理，或者系统资源确实不足。

2．性能优化

定位产生性能问题的具体原因后，就需要进行性能优化，根据网站分层架构，可分为 Web 前端性能优化、应用服务器性能优化、存储服务器性能优化 3 大类。

4.2 Web 前端性能优化

一般说来 Web 前端指网站业务逻辑之前的部分，包括浏览器加载、网站视图模型、图片服务、CDN 服务等，主要优化手段有优化浏览器访问、使用反向代理、CDN 等。

4.2.1 浏览器访问优化

1．减少 http 请求

HTTP 协议是无状态的应用层协议，意味着每次 HTTP 请求都需要建立通信链路、进行数据传输，而在服务器端，每个 HTTP 都需要启动独立的线程去处理。这些通信和服务的开销都很昂贵，减少 HTTP 请求的数目可有效提高访问性能。

减少 HTTP 的主要手段是合并 CSS、合并 JavaScript、合并图片。将浏览器一次访问需要的 JavaScript、CSS 合并成一个文件，这样浏览器就只需要一次请求。图片也可以合并，多张图片合并成一张，如果每张图片都有不同的超链接，可通过 CSS 偏移响应鼠标点击操作，构造不同的 URL。

2．使用浏览器缓存

对一个网站而言，CSS、JavaScript、Logo、图标这些静态资源文件更新的频率都比较低，而这些文件又几乎是每次 HTTP 请求都需要的，如果将这些文件缓存在浏览器中，可以极好地改善性能。通过设置 HTTP 头中 Cache-Control 和 Expires 的属性，可设定浏览器缓存，缓存时间可以是数天，甚至是几个月。

在某些时候，静态资源文件变化需要及时应用到客户端浏览器，这种情况，可通过改变文件名实现，即更新 JavaScript 文件并不是更新 JavaScript 文件内容，而是生成一个新的 JS 文件并更新 HTML 文件中的引用。

使用浏览器缓存策略的网站在更新静态资源时，应采用批量更新的方法，比如需要更新 10 个图标文件，不宜把 10 个文件一次全部更新，而是应一个文件一个文件逐步更新，并有一定的间隔时间，以免用户浏览器突然大量缓存失效，集中更新缓存，造成服务器负载骤增、网络堵塞的情况。

3．启用压缩

在服务器端对文件进行压缩，在浏览器端对文件解压缩，可有效减少通信传输的数据量。文本文件的压缩效率可达 80%以上，因此 HTML、CSS、JavaScript 文件启用 GZip 压缩可达到较好的效果。但是压缩对服务器和浏览器产生一定的压力，在通信带宽良好，而服务器资源不足的情况下要权衡考虑。

4．CSS 放在页面最上面、JavaScript 放在页面最下面

浏览器会在下载完全部 CSS 之后才对整个页面进行渲染，因此最好的做法是将 CSS 放在页面最上面，让浏览器尽快下载 CSS。JavaScript 则相反，浏览器在加载 JavaScript 后立即执行，有可能会阻塞整个页面，造成页面显示缓慢，因此 JavaScript 最好放在页面最下面。但如果页面解析时就需要用到 JavaScript，这时放在底部就不合适了。

5．减少 Cookie 传输

一方面，Cookie 包含在每次请求和响应中，太大的 Cookie 会严重影响数据传输，因此哪些数据需要写入 Cookie 需要慎重考虑，尽量减少 Cookie 中传输的数据量。另一方面，对于某些静态资源的访问，如 CSS、Script 等，发送 Cookie 没有意义，可以考虑静态资源使用独立域名访问，避免请求静态资源时发送 Cookie，减少 Cookie 传输的次数。

4.2.2 CDN 加速

CDN（Content Distribute Network，内容分发网络）的本质仍然是一个缓存，而且将数据缓存在离用户最近的地方，使用户以最快速度获取数据，即所谓网络访问第一跳，如图 4.5 所示。

由于 CDN 部署在网络运营商的机房，这些运营商又是终端用户的网络服务提供商，因此用户请求路由的第一跳就到达了 CDN 服务器，当 CDN 中存在浏览器请求的资源时，从 CDN 直接返回给浏览器，最短路径返回响应，加快用户访问速度，减少数据中心负载压力。

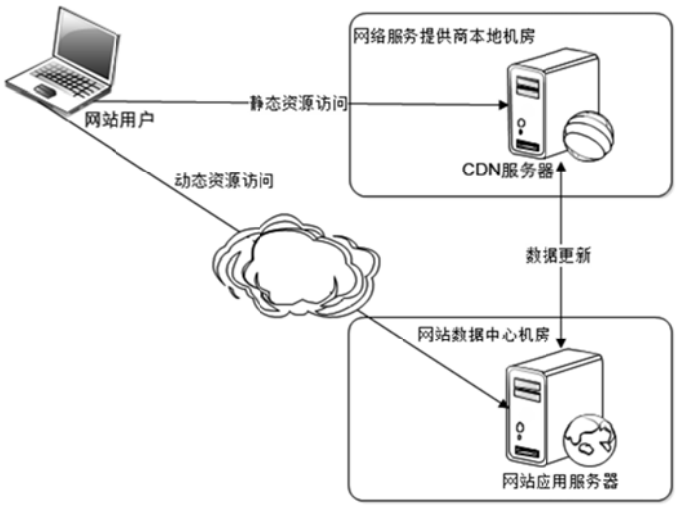

图 4.5　利用 CDN 的网站架构

CDN 能够缓存的一般是静态资源，如图片、文件、CSS、Script 脚本、静态网页等，但是这些文件访问频度很高，将其缓存在 CDN 可极大改善网页的打开速度。

4.2.3 反向代理

传统代理服务器位于浏览器一侧，代理浏览器将 HTTP 请求发送到互联网上，而反向代理服务器位于网站机房一侧，代理网站 Web 服务器接收 HTTP 请求。如图 4.6 所示。