目录

[第一章 大型网站架构演化 3](#_Toc458606308)

[1.1大型网站的关注指标 3](#_Toc458606309)

[1.2大型网站的特点 3](#_Toc458606310)

[1.3 大型网站架构演化发展过程 3](#_Toc458606311)

[1.4大型网站架构演化的价值观 4](#_Toc458606312)

[1.5 网站架构设计误区 4](#_Toc458606313)

[第二章 大型网站架构模式 5](#_Toc458606314)

[2.1 网站架构模式 5](#_Toc458606315)

[2.1.1分层 5](#_Toc458606316)

[2.1.2分割 5](#_Toc458606317)

[2.1.3分布式 5](#_Toc458606318)

[2.1.4集群 5](#_Toc458606319)

[2.1.5缓存 5](#_Toc458606320)

[2.1.6异步 6](#_Toc458606321)

[2.1.7冗余 6](#_Toc458606322)

[2.1.8自动化 6](#_Toc458606323)

[2.1.9安全 6](#_Toc458606324)

[2.2 架构模式在新浪微博的应用 7](#_Toc458606325)

[第三章 大型网站核心架构要素 8](#_Toc458606326)

[3.1 性能 8](#_Toc458606327)

[3.2 可用性 8](#_Toc458606328)

[3.3 伸缩性 8](#_Toc458606329)

[3.4 扩展性 9](#_Toc458606330)

[3.5 安全性 9](#_Toc458606331)

[第四章 瞬时响应 - 高性能架构 10](#_Toc458606332)

[4.1 不同视角下的网站性能 10](#_Toc458606333)

[4.2 性能测试指标 10](#_Toc458606334)

[4.3 性能测试方法 10](#_Toc458606335)

[4.4 Web前端优化 11](#_Toc458606336)

[4.5 应用服务器性能优化(缓存、缓存框架、消息队列) 11](#_Toc458606337)

[第五章 万无一失 - 高可用架构 15](#_Toc458606338)

[5.1 网站可用性度量 15](#_Toc458606339)

[5.2 高可用的网站架构 15](#_Toc458606340)

[5.3 高可用的应用 15](#_Toc458606341)

[5.4 高可用的服务 16](#_Toc458606342)

[5.5 高可用的数据 17](#_Toc458606343)

[5.6 高可用网站的软件质量保证 18](#_Toc458606344)

[5.7 网站运行监控 19](#_Toc458606345)

[第六章 永无止境 - 可伸缩性架构 20](#_Toc458606346)

[6.1 网站架构的伸缩性设计 20](#_Toc458606347)

[6.2 应用服务器集群的伸缩性设计 20](#_Toc458606348)

[HTTP重定向负载均衡 20](#_Toc458606349)

[DNS域名解析负载均衡 20](#_Toc458606350)

[反向代理负载均衡（应用层负载均衡） 21](#_Toc458606351)

[IP负载均衡（网络层负载均衡） 21](#_Toc458606352)

[数据链路层负载均衡 21](#_Toc458606353)

[负载均衡算法 21](#_Toc458606354)

[6.3 分布式缓存集群的伸缩性设计 21](#_Toc458606355)

[6.4 数据存储服务器集群的伸缩性设计 22](#_Toc458606356)

[第七章 随需应变 - 可扩展架构 24](#_Toc458606357)

[7.1构建可扩展的网站架构 24](#_Toc458606358)

[7.2 利用分布式消息队列降低系统耦合性 24](#_Toc458606359)

[7.3布式服务打造可服用的业务平台 24](#_Toc458606360)

[7.4可扩展的数据结构 25](#_Toc458606361)

[7.5 利用开放平台建设网站生态圈 25](#_Toc458606362)

[第八章 固若金汤 - 安全架构 26](#_Toc458606363)

[8.1典型攻击方式 26](#_Toc458606364)

[8.2信息加密技术以及密钥管理 27](#_Toc458606365)

[8.3信息过滤与反垃圾 28](#_Toc458606366)

[8.4电子商务风险控制 28](#_Toc458606367)

[8.5 小结 28](#_Toc458606368)

网站性能优化第一定律：优先考虑使用缓存优化性能。

计算机的任何问题都可以通过增加一个虚拟层来解决。计算机网络的7层协议，每一层都可以看作是下一层的虚拟层；计算机操作系统可以看作是计算机硬件的虚拟层；Java虚拟机可以看作是操作系统的虚拟层；分层的计算机软件架构事实上也是利用虚拟层的概念。

所谓缓存就是将读写比例较高，且很少变化的数据存储在相对较高访问速度的存储介质中。

常见的负载均衡种类：(具体见第六章：永无止境-可伸缩架构)

|  |  |
| --- | --- |
| 1 | Http重定向负载均衡 |
| 2 | DNS域名解析负载均衡 |
| 3 | 反向代理负载均衡 |
| 4 | IP负载均衡(网络层负载均衡) |
| 5 | 数据链路层负载均衡 |

常见的负载均衡算法：(具体见第六章：永无止境-可伸缩架构)

|  |  |
| --- | --- |
| 1 | 轮询 |
| 2 | 加权轮询 |
| 3 | 随机 |
| 4 | 最小连接 |
| 5 | 原地址散列 |

应用服务器集群的Session管理(5.3 高可用的应用)

|  |  |
| --- | --- |
| 1 | Session复制 |
| 2 | Session绑定 |
| 3 | 利用cookie记录session |
| 4 | Session服务器 |

第一章 大型网站架构演化

1.1大型网站的关注指标

高可用

高性能

易扩展

可伸缩

安全

1.2大型网站的特点

高并发，大流量

高可用

海量数据

用户分布广泛，网络情况复杂

安全环境恶劣

需求快速变更，发布频繁

渐进式发展

1.3 大型网站架构演化发展过程

* 初始阶段：一般使用LAMP(Linux，Appache，MySQL，PHP)来搭建，所有资源存放在一台服务器上
* 应用服务和数据服务分离：有独立的数据库服务器（越来越多的用户访问导致性能越来越差，越来越多的数据导致存储空间不足，这时就需要将应用和数据分离）
* 使用缓存改善网站性能，依据是二八定律：80%的业务访问集中在20%的数据上。数据库压力太大导致访问延迟，进而影响整个网站的性能，大部分的业务访问集中在一小部分数据上，把这一小部分数据缓存在内存中，减少数据库压力。① 这里需要考虑哪些数据适合缓存②缓存可以是本地缓存，也可以是远程分布式缓存。
* 使用应用服务器集群改善网站的并发处理能力：单一应用服务器能够处理的请求有限，在网站访问高峰期，应用服务器成为整个网站的瓶颈，使用集群是网站解决高并发、海量数据问题的常用手段。① 如果能通过增加一台服务器的方式来改善负载压力，就可以以同样的方式持续增加服务器来不断改善系统性能，从而实现系统的可伸缩性；②这里需要考虑使用哪些负载均衡的策略。
* 数据库读写分离：使用缓存后仍有一部分读操作和全部的写操作需要访问数据库，在网站用户达到一定规模后，数据库因为负载压力过高而称为网站的瓶颈。① 缓存中的数据，如果更新过快，那么会持续刷新缓存，从而降低性能；②可以利用主流数据库提供的主从热备功能，通过配置两台数据库的主从关系，将一台数据库服务器上的数据同步到另外一台上面。
* 使用反向代理和CDN加速网络响应：① CDN和反向代理的基本原理都是缓存；② CDN部署在网络提供商的机房，用户在请求网络服务时，可以从距离自己最近的网络提供商机房获取数据；③ 反向代理部署在网站的中心机房，当用户的请求到达中心机房后，首先访问的服务器是反向代理服务器，如果反向代理服务器中缓存着用户请求的资源，那么就将其直接返回给用户；④ 使用CDN和反向代理的目的都是尽早返回数据给用户，一方面加快用户访问速度，另一方面也减轻后端服务器的负载压力
* 使用分布式文件系统和分布式数据库系统：① 网站常用的数据库拆分手段是业务分库，即将不同业务的数据库部署到不同的物理服务器上；②分布式数据库是网站数据拆分的最后手段，只有在表达数据规模非常庞大的时候才使用。
* 使用NoSQL和搜索引擎
* 业务拆分：使用分而治之的手段将整个网站业务分成不同的产品线，也会根据产品线划分，将一个网站拆分成许多不同的应用，每个应用独立部署维护。
* 分布式服务，将共用的业务提取出来，独立部署，通过分布式服务调用共用业务服务完成具体业务操作。

1.4大型网站架构演化的价值观

网站的价值在于它能为用户提供什么价值，在于网站能做什么，而不在于它是怎么做的。因此对于小型网站来说，最需要做的是为用户提供好的服务来创造价值，得到用户的认可，从而活下去，野蛮生长。

① 大型网站架构技术的核心价值是随网站所需灵活应对， 它是一个演化的过程；

② 驱动大型网站技术发展的主要力量是网站的业务发展，是业务成就了技术，而不是相反。因此要摒弃为了技术而技术的套路。12306的真正问题不在于它的技术架构，而在于它的业务架构：调整业务需求，换一种方式卖票，而不要去搞促销秒杀这种噱头式的游戏（在售票方式上引入排队机制、整点售票调整为分时段售票）。

1.5 网站架构设计误区

① 一味追随大公司的解决方案

② 为了技术而技术

③ 企图用技术解决所有问题

第二章 大型网站架构模式

2.1 网站架构模式

网站设计过程中有一些解决方案被很多网站重复使用，从而逐渐形成大型网站结构模式。

### 2.1.1分层

在横向方向对系统进行切分。

* 分层的挑战在于必须合理规划层次边界和接口
* 分层包括物理分层和逻辑分层两种
* 分为应用层、服务层、数据层

### 2.1.2分割

在纵向方向对系统进行切分。将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。主要是对业务进行

### 2.1.3分布式

* 分层和分割的目的在于小模块便于分布式部署
* 带来的问题：① 分布式意味着服务调用必须通过网络，需要考虑带宽的影响；② 服务器越多，宕机的概率越大
* 常用的分布式方案：① 分布式应用和服务；② 分布式静态资源；③ 分布式数据和存储； ④ 分布式计算；⑤ 分布式配置、分布式锁、分布式文件系统

### 2.1.4集群

即多台服务器部署相同的应用，从而构成一个集群，通过负载均衡设备共同对外提供服务。即使访问量很小的分布式应用和服务，也至少要部署到两台服务器来构成一个小集群，这样可以提高系统的可用性。

### 2.1.5缓存

即将数据放在距离计算最近的位置以加快处理速度

① CDN：即内容分发网络Content Delivery Network。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它的网络服务商哪里，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最大的热点内容缓存在CDN。

在CDN的应用中，从传统意义上来讲，主要是把用户需要访问的内容放到离用户近的地方。

② 反向代理：反向代理属于网站前端结构的一部分，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。

③ 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。

④ 分布式缓存：大型网站的数据量非常大，即使只缓存一小部部分，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。

### 2.1.6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法异步进行协作。

* 通常需要使用消息队列（生产者消费者模式）
* 带来的好处：1） 提高系统可用性； 2） 加快网站响应速度； 3） 消除并发访问高峰

### 2.1.7冗余

* 集群带来的必然结果
* 安全需求的必然结果
* 数据库冷热备份

### 2.1.8自动化

DevOps思维，尽量减少人工干预

自动化发布

自动化代码管理

自动化测试

自动化安全监测

自动化部署

自动化监控

自动化报警

自动化失效转移、恢复

自动化分配资源

......

### 2.1.9安全

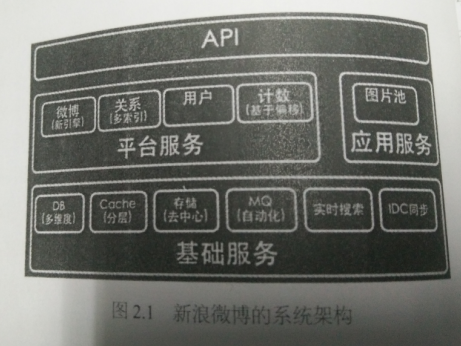
① 通过密码和手机校验进行身份验证；

② 登录、交易等操作需要对网络通信进行加密；

③ 网络服务器上存储的敏感数据如用户信息等也进行加密处理 ；

④ 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。

2.2 架构模式在新浪微博的应用



系统分为三层：基础服务层、平台服务和应用服务层、API和新浪微博的业务层。

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，是整个系统的技术支持。

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。

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继承到新浪微博系统中，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，每个模块都被部署在一组独立的服务器集群上，同远程调用的方式进行依赖访问。

在新浪微博的早起架构中，微博发布使用同步推送模式，用户发表微博后系统会立即将这条微博插入到数据库中所有粉丝的订阅列表中，当用户量比较大时，会引起大量的数据库写操作，超出数据库负载，系统性能急速下降，用户响应延迟。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，非在线用户登录后根据关注列表拉取微博订阅列表。

由于微博刷新频繁，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微博的服务器上，在心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，可以获得良好的系统性能。

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，这些数据中心即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，提高系统的可用性。

第三章 大型网站核心架构要素

3.1 性能

1）一个性能问题可能会导致网站用户严重流失。

2）衡量性能的指标：响应时间、TPS、系统性能计数器等。

3）性能优化：从用户浏览器到数据库，影响用户请求的所有环节都可以进行性能优化 。

① 浏览器端，可以通过浏览器缓存、使用页面压缩、合理布局页面、减少Cookie传输等手段改善性能；使用CDN和反向代理服务器；

② 在应用服务器端，使用本地缓存和分布式缓存减轻数据库负载压力；使用异步、集群；

③ 在代码层面，使用多线程、改善内存管理等手段优化性能；

④ 在数据库服务器端，使用索引、缓存、SQL优化等性能优化手段。

3.2 可用性

1）没有网站可以完美的7\*24运行。

2）衡量高可用目标的标准：假设系统中任何一台或多台服务器宕机以及出现各种不可预测的问题时，系统整体是否可用。

3）网站高可用结构的前提是必然会出现服务器宕机，而高可用设计的目标是当服务器宕机时，服务或者应用依然可用。

4）必要的手段是集群，即冗余。

5）服务器可分为针对应用服务器和存储服务器：

①应用服务器集群中任何一台服务器宕机，只需要把请求转发到另一台应用服务器上即可。应用服务器高可用的前提条件是应用服务器上不能保存请求的会话信息，否则服务器宕机，会话丢失，即使用户请求转发到其他服务器上也无法完成业务处理；

②对于存储服务器，当服务器宕机时需要将数据访问转移到可用的服务器上，并进行数据恢复以保证继续有服务器宕机时数据依然可用

3.3 伸缩性

1） 定义：通过不断向集群中加入服务器的手段来缓解不断上升的用户并发访问压力和不断增长的数据存储需求

2）衡量标准：是否可以构建集群；是否可以方便的向集群中添加新的服务器。

3）① 对于应用服务器集群，只有服务器上不保存数据，所有服务器都是对等的；

② 对于缓存服务器集群，加入新的服务器可能会导致缓存路由失效，进而导致集群中大部分缓存数据无法访问，需要改进缓存路由算法保证缓存数据的可访问性；

③ 关系数据库虽然支持数据复制，主从热备等机制，但很难做到大规模集群的可伸缩性，因此关系数据库的集群伸缩性方案必须在数据库之外实现，通过路由分区等手段将部署有多个数据库的服务器组成一个集群；

④ 大部分NOSQL数据库产品，对伸缩性的支持通常都非常好

3.4 扩展性

1) 目标：直接关注网站的功能，保证可以快速响应需求变更

2）衡量标准： 网站增加新的业务产品时，是否对现有业务透明无影响

3）主要手段是事件驱动架构和分布式服务（事件驱动架构通常利用消息队列实现；分布式服务则是将业务和可复用服务分离开来，通过分布式服务框架调用）

3.5 安全性

1）目的：保护网站不受恶意访问和攻击。

2）衡量标准： 针对现存和潜在的各种攻击和窃密手段，是否可以有效的应对

第四章 瞬时响应 - 高性能架构

4.1 不同视角下的网站性能

* 用户视角

1）主要是端到端的感觉：用户在浏览器上直观感受到的网站响应速度是快还是慢。

2）主要通过前端优化的手段来提升用户体验，比如优化页面HTML、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。

* 开发人员视角

1）主要关注应用程序本身以及相关子系统的性能，包括响应延迟、系统吞吐量、并发处理能力、系统稳定性等。

2）主要优化手段： 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、使用代码优化提升程序性能。

* 运维人员视角

1）主要关注基础设施性能和资源利用率。

2）主要优化手段： 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率。

4.2 性能测试指标

从不同的角度看网站就有不同的性能指标。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、吞吐量和性能计数器。

1）响应时间，即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.实践中通常采用的办法是重复请求多次，记录时间然后除以请求的次数。

2）并发数，即系统能够同时处理的请求的数目，也反映了系统的负载特性.

3）吞吐量，即单位时间内系统处理的请求数量，体现系统的整理处理能力.

4）性能计数器， 描述服务器或者操作系统性能的一些数据指标.

4.3 性能测试方法

性能测试反应的是在实际运行过程中，随着并发请求数量的增大，系统的处理能力。

1）性能测试，以系统设计初期规划的性能指标为预期目标，对系统不断增压，测试目标是评估系统性能是否符合需求和达到预期目标.

2）负载测试，对系统不断的增加并发请求，直到系统的某项或者多项性能指标达到安全临界值，目标是测试系统正常运行情况下能够承受的最大访问负载压力.

3）压力测试，超过安全负载的情况下，继续对系统增压，直到系统崩溃或者不能再处理任何请求，目标是测试导致系统崩溃的最大访问负载压力.

4）稳定性测试，在特定硬件、软件、网络情况下，给系统加载一定压力，是系统运行较长一段时间，来观察系统是否稳定.

4.4 Web前端优化

* 浏览器访问优化

1）减少http请求：合并CSS，合并JavaScript、合并图片。

2）使用浏览器缓存：在浏览器端缓存一部分的静态资源文件。

3）启用压缩：服务器端加压缩、客户端解压缩。但会给服务器和浏览器增加压力，在通信宽带良好，而服务器资源不足的情况下需要权衡。

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)

5）减少Cookie传输：太大的cookie会影响传输，应尽量减少cookie中传输的数据量。

* CDN加速

CDN(Content Distribute Network,内容分发网络)本质是缓存，将数据缓存在距离用户最近的地方，使得用户可以通过最短的距离获取数据即所谓的网络访问第一跳。

CDN部署在网络运营商的机房，用户发的请求会首先到达CDN服务器 ，当CDN中存在浏览器请求的资源，那么就会直接从CDN返回给用户，极大的减少了用户等待的响应时间，加快用户的访问速度。CDN能够缓存的一般是静态资源，如图片、文件、CSSS、Script脚本、静态网页等，由于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。

* 反向代理

反向代理的本质也是缓存。反向代理服务器是位于网站机房一侧，代理网站Web服务器接收HTTP请求。当用户的请求到达中心机房后，首先访问的服务器是代理服务器，如果代理服务器中缓存有用户请求的资源(可静态，可动态。)，则直接返回，否则代理服务器将用户的请求进行分发到应用服务器，代理服务器将应用服务器返回的结果做缓存，然后将结果返回给用户。

所以反向代理可以做的事情：① 保护网站安全；② 缓存资源，加速web响应；③实现负载均衡，通过负载均衡构建的应用集群可以提高系统总体处理能力，进而改善网站高并发情况下的性能。

* 1. 应用服务器性能优化(缓存、缓存框架、消息队列)

应用服务器就是处理网站业务的服务器，优化手段主要有缓存、集群、异步等。

* 分布式缓存

1. 缓存的原理

缓存就是将读写比很高、很少变化的数据存储在相对较高访问速度的存储介质中，以供系统处理。①缓存访问速度快，可以加快数据访问时间；②被缓存的数据都是提前计算好的，所以利用缓存也可以减少计算时间。

缓存对提高系统性能有很多好处，但是也要注意不能过分依赖和使用缓存，否则会成为系统的累赘和风险。缓存从本质上来说，就是一个内存hash表，计算KV中key的HashCode对应的Hash表索引，可快速访问Hash表中的数据。

① 缓存需要缓存那些读写比很高、很少变化的数据，一般来说读写比在2:1以上时，缓存才有意义。

② 应用程序读取数据时，首先到缓存中读取，如果缓存不存在或者已失效，再访问数据库，同时将新的数据放入缓存。

③ 缓存也需要注意缓存热点数据。缓存预热指的是在新启动的缓存系统中，在启动时就加载热点数据，这样启动后就可以直接使用。

④ 缓存穿透， 应用持续大量访问不存在的数据，因为这类数据不存在于缓存中，因此会大量访问数据库，从而降低性能

1. 分布式缓存框架

对于分布式缓存来说，目前有两类：

(1) 不同的缓存服务器之间进行通信，例如JBoss Cache（JBoss Cache的分布式缓存在集群中所有服务器中保存相同的缓存数据，带来的问题就是缓存更新信息的时候需要同步到集群所有机器，代价很惊人）。

(2) 不同缓存服务器之间不进行通信，例如Memcached（缓存与应用分离部署，缓存系统部署在一组专门的服务器上，应用程序通过一致性Hash等路由算法选择缓存服务器远程访问缓存数据，一致性Hash算法已成为数据存储伸缩性架构设计的经典范式，正是集群内服务器互不通信使得集群可以做到几乎无限制的线性伸缩，这也正是目前流行的许多大数据技术的基本架构特点。）

Memcached设计简单、性能优异、高效的内存管理、互不通信的服务器集群、海量数据可伸缩的架构。Memcached使用TCP协议(UDP也支持)。

* 异步操作

使用消息队列可以将调用异步化，可改善网站的扩展性和网站的性能。使用消息队列后，用户请求的数据发送给消息队列后立即返回，再由消息队列的消费者进程从消息队列中获取数据，异步写入数据库。

一般会使用消息队列，带来的额外好处是会削平峰值（消息队列+通知），将短时间内高并发产生的事务消息存储在消息队列中，从而削平高峰期的并发事务。

在使用异步消息队列之后，需要额外适当修改业务流程进行配合，比如订单提交自后，订单数据写到消息队列中，不能立即返回用户订单提交成功 ，需要在消息队列的订单消费者进程真正处理完该订单，甚至商品出库之后，再通过电子邮件或SMS消息通知用户订单成功，避免交易纠纷。

* 使用集群

使用负载均衡技术为一个应用构建一个由多台服务器组成的服务器集群，将并发访问请求分发到多台服务器上处理，避免单一服务器由于并发请求数量过大导致响应缓慢。

* 代码优化

1. 多线程

利用多线程与IO阻塞交替执行，最大限度利用CPU资源。

需要注意线程安全问题：① 将对象设计成无状态对象；② 使用局部对象；③ 并发访问资源时使用锁

1. 资源复用

主要是单例和资源池（对象池）

1. 数据结构，选择合适的算法
2. 垃圾回收，合理设置垃圾回收策略，合理设置老年代和新生代的大小。

* 存储性能优化

1. 机械硬盘 vs 固态硬盘

机械硬盘：常用、连续访问和随机访问时性能差别大。

固态硬盘：无机械装置、数据存储在可持久记忆的硅晶体上、可像内存一样被随机访问。缺点是不太成熟，可靠性与性价比有待提升。

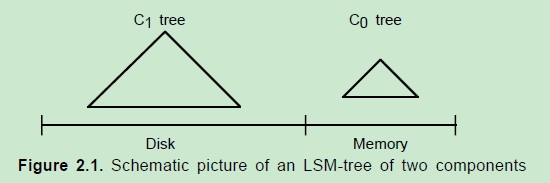
1. B+树 vs LSM树

关系型数据库采用B+树索引提高数据的查询效率。目前数据库多采用两级索引的B+树，树的层次最多三层。因此可能需要5次磁盘访问才可以更新一条记录(三次磁盘访问获得数据索引及行ID，然后再进行一次数据文件读操作和一次文件写操作)。

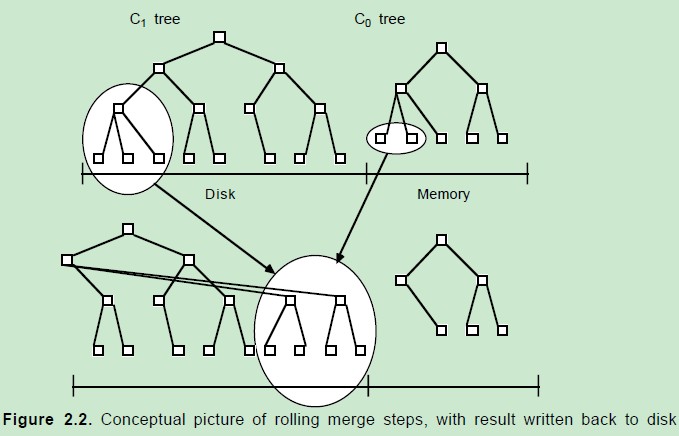
LSM(Log Structured Merge Tree)树，就是将对数据的修改增量保持在内存中，达到指定的大小限制后将这些修改操作批量写入磁盘，读取时需要合并磁盘中的历史数据和内存中最近的修改操作。LSM树的优势在于有效地规避了磁盘随机写入问题，但读取时可能需要访问较多的磁盘文件。

LSM-tree的主要思想是划分不同等级的树。以两级树为例，可以想象一份索引数据由两个树组成，一棵树存在于内存，一棵树存在于磁盘。内存中的树可以不一定是B-树，可以是其他的树，例如AVL树。因为数据大小是不同的，没必要牺牲CPU来达到最小的树高度。而存在于磁盘的树是一棵B-树。

http://bofang.iteye.com/blog/1676698



数据首先会插入到内存中的树。当内存中的树中的数据超过一定阈值时，会进行合并操作。合并操作会从左至右遍历内存中的树的叶子节点与磁盘中的树的叶子节点进行合并，当被合并的数据量达到磁盘的存储页的大小时，会将合并后的数据持久化到磁盘，同时更新父亲节点对叶子节点的指针。



 之前存在于磁盘的叶子节点被合并后，旧的数据并不会被删除，这些数据会拷贝一份和内存中的数据一起顺序写到磁盘。

1. RAID vs HDFS

RAID(廉价磁盘冗余阵列)：主要是为了改善磁盘的访问延迟，增强磁盘的可用性和容错能力。

RAID技术在传统关系数据库及文件系统中应用比较广泛，但是在大型网站比较喜欢使用的是NoSQL，以及分布式文件系统。

HDFS以块(Block)为单位管理文件内容，一个文件被分割成多个Block，当应用程序写文件时，每写完一个Block，HDFS就将其自动复制到另外两台机器上，保证每个Block有三个副本，即使有两台服务器宕机，数据依然可以访问。

在HDFS中有两个重要的角色，NameNode(名字服务节点)和DataNode(数据存储节点)。NameNode在整个HDFS中只部署一个实例，提供元数据服务，相当于操作系统中的文件分配表(FAT)，管理文件名Block的分配，维护整个文件系统的目录树结构。DataNode则部署在HDFS集群中其他所有服务器上，提供真正的数据存储服务。

第五章 万无一失 - 高可用架构

网站的可用性描述网站可以有效访问的特性，它不同于易用性

* 1. 网站可用性度量

(1) 网站不可用时间 = 故障修复时间点 - 故障发现时间点

(2) 网站年度可用性指标 = （1 - 网站不可用时间/年度总时间）\* 100%

一般以几个9来表示，2个9是基本可用，网站年度不可用时间小于88小时；3个9是较高可用，网站年度不可用时间小于9小时；4个9是具有自动恢复能力的高可用，网站年度不可用时间小于53分钟；5个9是极高可用性，网站年度不可用时间小于5分钟。

5.2 高可用的网站架构

* 设计目标

保证服务器硬件故障时服务依然可用、数据依然保存并能够被访问。

* 主要手段

数据和服务的冗余备份以及失效转移，一旦服务器宕机，就将服务切换至其他可用的服务器上。

* 应用层、服务层、数据层针对高可用的解决方案

典型的分层模型是把网站分为应用层、服务层和数据层；各层之间具有相对独立性，应用层主要负责具体业务逻辑处理，服务层负责提供可复用的服务，数据层负责数据的存储与访问。

位于应用层的服务器通常为了应对高并发的访问请求，会通过负载均衡设备将一组服务器组成一个集群共同对外提供服务，当负载均衡设备通过心跳检测等手段监控到某台应用服务器不可用时，就将其从集群列表中删除，并将请求分发到集群中其他可用的服务器上，使整个集群保持可用，从而实现高可用。

位于服务层的服务器和应用层的服务器类似，也是通过集群方式实现高可用，只是这些服务器被应用层通过分布式服务调用框架访问，分布式服务调用框架会在应用层客户端程序中实现软件负载均衡，并通过服务注册中心对提供服务的服务器进行心跳检测，发现有服务不可用，立即通过客户端程序修改服务访问列表，剔除不可用的服务器。

位于数据层的服务器比较特殊，数据服务器上存储着数据，为了保证服务器宕机时数据不丢失，数据访问服务不中断，需要在数据写入时进行数据同步赋值，将数据写入多台服务器上，实现数据冗余备份。当数据服务器宕机时，应用程序将访问切换到有备份数据的服务器上。

5.3 高可用的应用

* 应用的无状态性

应用层主要处理网站应用的业务逻辑，因此有时也称作业务逻辑层。应用的一个主要特点是应用的无状态性。

无状态应用： 应用服务器不保存业务的上下文信息，而仅根据每次请求提交的数据进行相应的业务逻辑处理，多个服务实例之间完全对等，请求提交到任何一个服务器上，处理的结构都是相同的。

* 负载均衡

通过负载均衡进行无状态服务的失效转移。

主要使用在业务量和数据量较高的情况下，当单台服务器不足以承担所有的负载压力时，通过负载均衡手段，将流量和数据分摊到一个集群组成的多台服务器上， 以提升整体的负载处理能力。

* 应用服务器集群的Session管理

① Session复制

应用服务器开启Web容器的Session复制功能，在集群中的几台服务器之间同步Session对象，使每台服务器上都保存所有用户的Session信息。

好处：不必固定特定的应用服务器；坏处：只有有session发生改变，应用服务器之间就需要同步，因此效率低下，大型网站的集群中通常不采用这种做法。

② Session绑定

Session绑定可以利用负载均衡的源地址Hash算法实现，将Session绑定在某台特定服务器上，保证每次用户的请求都被与其绑定的服务器所处理。也叫会话粘滞，不符合对系统高可用的要求，因此很少使用这这种方法。

③ 利用Cookie记录Session

将Session记录在客户端，每次请求服务器时，将Session放在请求中发送给服务器，服务器处理完请求后再将修改过的Session响应给客户端。对浏览器就是利用其支持的Cookie记录Session。

缺点是Cookie记录的内容大小有限；每次请求都传输Cookie会影响性能；如果用户关闭Cookie访问就不正常。

④ Session服务器

利用独立部署的Session服务器（集群）统一管理Session，这种解决方案事实上是将应用服务器的状态分离，分为无状态的应用服务器和有状态的Session服务器。然后针对这两种服务器的不同特性分别设计其架构，对于有状态的Session服务器，一种比较简单的方法是利用分布式缓存、数据库等，在这些产品的基础上进行包装，使其符合Session的存储和访问要求，如果业务场景对Session管理由比较高的要求，则需要开发专门的Session服务器管理平台。

5.4 高可用的服务

服务层负责提供可复用的服务，可复用的服务也是无状态的服务，可使用类似负载均衡的失效转移策略实现高可用的服务。

* 分级管理

运维上将服务器进行分级管理，核心应用和服务优先使用更好的硬件。同时在服务部署上也进行必要的隔离，避免故障的连锁反应。

* 超时设置

在应用程序中设置服务调用的超时时间，一旦超时，通信框架就抛出异常，应用程序根据服务调度策略，可选择重试或将请求转移到提供相同服务的其他服务器上。

* 异步调用

应用对服务的调用通过消息队列等异步方式完成，避免一个服务失败导致整个应用请求失败的情况。

* 服务降级

为了保证核心应用和功能的正常执行，可采用降级手段如拒绝服务和关闭服务。

拒绝服务：拒绝低优先级应用的调用。

关闭服务：关闭部分不重要的服务。如淘宝双十一是关闭了评价和确认收货等非核心服务。

* 幂等性设计

在服务层需要保证服务重复调用和调用一次产生的结果相同，即服务具有幂等性。有些服务是天然幂等性的，比如用户将性别设为女，不管设置多少次，都还是女性。但是转账这种操作就必须通过交易编号等信息进行服务调用有效性校验，只有有效的操作才可以继续进行。

5.5 高可用的数据

保护网站的数据就是保护企业的命脉。

* 主要手段

数据备份和失效转移。数据备份保证数据有多个副本，任意副本的失效都不会导致数据的永久丢失，从而实现数据完全的持久化。失效转移机制则保证当一个数据副本不可访问时，可以快速切换访问数据的其他副本。

* CAP原理

一个提供数据服务的存储系统无法同时满足数据一致性(Consistency)、数据可用性(Availibility)、分区耐受性(Parition Tolerance,系统具有跨网络分区的伸缩性)这三个条件。

* 数据一致性分类(从强到弱)

① 数据强一致：各个副本的数据在物理存储中总是一致的。

② 数据用户一致：数据在物理存储中的各个副本的数据可能是不一致的，但是终端用户访问时，通过纠错和校验机制，可以确定一个一直的且正确的数据返回给用户。

③ 数据最终一致：物理上存的副本也不一样，用户访问的也不一样，但可经过一段时间的自我恢复和修正，数据最终会达到一致。

* 数据备份

① 冷备份

定期将数据复制到某种存储介质(磁盘、光盘…)上并物理存档保管，如果系统存储损坏，那么就从冷备份中的存储设备中恢复数据。

冷备的优点是简单和廉价，成本和技术难度较低，缺点是不能保证数据最终一致(从上个备份点开始后更新的数据就会永久丢失)。

② 热备份

1. 异步热备

应用程序写数据库的时候 ，只写一份，然后存储系统会异步地写其他副本。

1. 同步热备

应用程序写数据库的时候会同时写到多个副本。

* 失效转移

当数据服务器集群中任意一台服务器宕机时，应用程序针对这台服务器的所有读写操作都会转移到其他的数据服务器上，保证数据访问不会失败。失效转移操作有三个步骤：失效确认、访问转移、数据恢复。

Step1：失效确认

失效确认的手段有：心跳检测、应用程序访问失败报告。

Step2：访问转移

应用程序根据配置切换到对等的服务器上，如果存储是不对等的，则需要重新计算路由，选择对等的存储服务器。

Step3：数据恢复

因为某台服务器宕机，所以数据存储的副本数目会减少，必须将副本的数目恢复到系统设定的值，否则，再有服务器宕机时，就可能出现无法访问转移，数据永久丢失的情况，因此系统需要从健康的服务器复制数据，将数据副本数目恢复到设定值）

5.6 高可用网站的软件质量保证

* 网站发布

它的过程和服务器宕机效果箱单，其对系统可用性的影响也类似。一般采取批量更新的方式进行，不会一次关掉集群中的全部服务器

* 自动化测试

目前大部分网站采用web自动化测试技术。一般使用Selenium来进行测试。Thoughtworks开发，运行在浏览器中 ，模拟用户操作进行测试，因此Selenium可以同时完成web功能测试和浏览器兼容测试。

* 预发布验证

预发布服务器是一种特殊用途的服务器，它和线上的正式服务器唯一的区别是没有配置在负载均衡服务器上，外部用户无法访问。预发布服务器和线上正式服务器都部署在相同的物理环境中，使用相同的线上配置，依赖相同的外部服务。如果预发布服务器上执行的测试验证都正确的，基本上可以确保在线上正式服务器部署时也是没有问题的。

* 代码控制SVN Git

① 主干开发，分支发布：代码修改都在主干上进行，需要发布的时候从主干上拉一个分之进行发布，该分之即成为一个发布版本。

② 分支开发，主干发布，这是目前使用的主流方式。任何修改都不能在主干上直接进行，需要开发一个新功能或者修复一个Bug时，从主干拉一个分之进行开发，开发完成之后合并到主干，然后从主干进行发布，主干上的代码永远是最新发布的版本。

两种方式各有优缺点：第一种反映目前整个应用的状态，一目了然，便于管理和控制，也利于持续集成开发，但是在发布的时候可能会由于有些项目开发才一半导致不能及时发布已经成功的其他项目。第二种方式各个分支独立进行，互不干扰，可以使不同发布周期的开发在同一应用中进行。可以避免第一种的缺点，因此目前主流的网站都是使用这个方式。

* 自动化发布

火车模型：将每个应用的发布过程看做一次火车旅程，火车定点运行，期间有若干站点，每一站都进行例行检查，不通过的项目下车，通过的项目继续坐着火车旅行，直到火车到达终点。

实际中，可能所有项目在途中都下车了，这样火车不得不回到原点，等待问题解决后再来一次

另一种可能是火车上的重点项目如果失败，那么整趟火车需要返回。

人的干预越少，自动化程度越高，引入故障的可能性就越小。

由于火车发布模型是基于规则驱动的流程，所以这个流程可以自动化。采用火车发布模型的网站会开发一个自动化发布的工具实现发布过程的自动化。根据相应驱动流程，自动构造代码分支，进行代码合并，执行发布脚本。

* 灰度发布

大型网站都会使用灰度发布模式，将集群服务器分成若干部分，每天只发布一部分服务器，观察运行稳定没有故障，第二天继续发布一部分服务器，持续几天你才把整个集群全部发布完毕，期间如果发现问题，只需要回滚已发布的一部分服务器即可。

灰度发布也常用于用户测试，即在部分服务器上发布新版本，其余服务器上保持老版本(或另一个版本)，然后监控用户操作行为，收集用户体验报告，比较用户对两个版本的满意度，最后确定要发布的版本。

5.7 网站运行监控

* 监控数据采集

① 用户行为日志收集：分为服务器端日志收集和客户端浏览器日志收集。

② 服务器性能监控：收集服务器性能指标，如系统Load、内存占用、磁盘IO、网络IO等。尽早做出故障预警，及时判断应用状况，防患于未然。

③ 运行数据报告：

* 监控管理

① 系统报警

② 失效转移

③ 自动优雅降级

网站在监控管理基础之上实现自动优雅降级，是网站柔性架构的理想状态。

第六章 永无止境 - 可伸缩性架构

网站伸缩性： 在不需要改变网站的软硬件设计，仅仅通过改变部署的服务器数量就可以扩大或者缩小网站的服务处理能力。

最重要的技术手段就是使用服务器集群，通过不断地向集群中添加服务器来增强整个集群的处理能力，这就是网站系统的伸缩性架构。只要技术上可以做到向服务器集群中加入服务器的数量和集群的处理能力成线性关系，那么网站就可以以此手段不断提升自己的规模。

6.1 网站架构的伸缩性设计

* 不同功能进行物理分离实现伸缩

不同的服务器部署不同的服务，提供不同的功能。根据功能进行物理分离实现伸缩，有两种形式的分割：纵向分离(分层后分离)和横向分离(业务分割后分离)。

* 单一功能通过集群规模实现伸缩

集群内的多台服务器部署相同的针对某个功能的服务，提供相同的功能。

集群伸缩性又分为应用服务器集群伸缩性和数据服务器集群伸缩性，这两种集群由于对数据状态管理的不同，技术实现上也不同。而数据服务器集群也可分为缓存数据服务器集群和存储数据服务器集群。

6.2 应用服务器集群的伸缩性设计

如果HTTP请求分发装置可以感知或者可以配置集群的服务器数量，可以及时发现集群中新上线或下线的服务器，并能向新上线的服务器分发请求，停止向已下线的服务分发请求，就实现了应用服务器集群的伸缩性。这个HTTP请求分发装置被称作负载均衡服务器。

### HTTP重定向负载均衡

HTTP重定向服务器是一台普通的应用服务器，唯一的功能就是根据用户的HTTP请求计算一台真实的web服务器地址，并将该web服务器地址写入HTTP重定向响应中返回给用户浏览器。

优点：简单；缺点：浏览器需要两次请求服务器才能完成一次访问，性能较差，而且重定向服务器自身的处理能力也有可能成为瓶颈。

实践中不常用。

### DNS域名解析负载均衡

利用DNS处理域名解析请求的同时进行负载均衡处理的一种方案。

优点：将负载均衡的工作转交给DNS，省掉了网站管理维护负载均衡服务器的麻烦；DNS支持基于地理位置的域名解析，加快用户访问速度，改善性能。

缺点：DNS是多级解析，修改了某个记录之后全部生效需要一段时间。

### 反向代理负载均衡（应用层负载均衡）

大多数反向代理服务器同时提供负载均衡的功能，管理一组web服务器，将请求根据负载均衡算法转发到不同web服务器上。这就要求反代理服务器配置双网卡和内部外部两套IP地址。

反向代理服务器转发请求在HTTP协议层面，因此也叫应用负载均衡。优点：与反向代理合在一起，部署简单；缺点：反向代理是所有请求和响应的中转站，性能可能会成为瓶颈。

### IP负载均衡（网络层负载均衡）

在网络层通过修改请求目标地址进行负载均衡。关键在于如何把真实物理web服务器响应数据包返回给负载均衡服务器。

### 数据链路层负载均衡

在通信协议的数据链路层修改mac地址进行负载均衡。

负载均衡数据分发过程中不修改IP地址，只修改目的IP地址，通过配置真实物理服务器集群所有机器虚拟IP和负载均衡服务器IP地址一致，从而达到不修改数据包的源地址和目的地址就可以进行数据分发的目的。

使用三角传输模式的链路层负载均衡是目前使用最广的一种负载均衡手段。

### 负载均衡算法

① 轮询

② 加权轮询

③ 随机

④ 最小链接

⑤ 原地址散列

6.3 分布式缓存集群的伸缩性设计

分布式缓存集群伸缩性设计的主要目标是新加入缓存服务器后应使整个缓存服务器集群中已缓存的数据尽可能还被访问到。

* Memcached分布式缓存集群的访问模型

用户程序通过Memcached客户端访问Memcached服务器集群，Memcached客户端主要由一组API、Memcached服务器集群路由算法、Memcached服务器集群列表以及通信模块构成

路由算法负责根据应用程序输入的缓存数据KEY计算得到应该将数据写入到Memcached的哪台服务器（写缓存）或者应该从哪台服务器读数据（读缓存）。通过使用相同的路由算法和服务器列表，只要应用程序提供相同的KEY，Memcached客户端客户端总是访问相同的服务器去读取数据，只要服务器中还缓存着该数据，就能保证缓存命中。

* Memcached分布式缓存集群的伸缩性挑战

挑战主要针对路由算法，当集群扩容时，如何保证路由算法可以得到新加入的服务器？

简单的路由算法可以使用余数Hash:用服务器数目除以缓存数据Key的Hash值，余数为服务器列表下标编号。但是有明显的缺点是当分布式缓存集群需要扩容的时候，缓存命中率就会变得很低。

解决方法： 在网站访问量最少的时候扩容，然后通过模拟请求的方法逐渐预热缓存，使得缓存服务器中的数据重新分布

* 分布式缓存的一致性Hash算法

一致性Hash算法通过一个叫作Hash环的数据结构实现KEY到缓存服务器的Hash映射。

首先构造一个长度为0~232的整数环，根据节点名称的Hash值，将缓存服务器节点放置在这个Hash环上，然后根据缓存的数据的KEY值计算得到其Hash值，然后在Hash环上顺时针查找距离这个KEY值的Hash值最近的缓存服务器节点，完成Key值到服务器的Hash映射查找。

这样做的话 ，在往集群中添加节点的时候只影响到了其中的一小段。但是可能带来的问题是，集群中的各个节点的负载将会产生不均衡。采用的解决办法就是将每台物理缓存服务器虚拟为一组虚拟缓存服务器，将缓存服务器的Hash值放置在Hash环上，KEY在环上先找到虚拟服务器节点，再得到服务器的信息。

6.4 数据存储服务器集群的伸缩性设计

数据存储服务器必须保证数据的可靠存储，任何情况下都必须保证数据的可用性和正确性

* 关系数据库集群的伸缩性设计

利用主从结构实现读写分离，读写操作针对的是不同的库，然后定时架构两个库进行数据同步。

根据不同业务的数据，放到不同的数据库集群中，即数据库分库。对于特别大的表，进行分表处理

目前比较成熟的支持数据分表的分布式关系数据库产品主要有Amoeba和Cobar。Cobar是一个支持分布式关系数据库访问代理，介于应用服务器和数据库服务器之间。

* NoSQL数据库的伸缩性设计

NoSQL主要指非关系的、分布式的数据库设计模式。一般而言，NoSQL数据库产品都放弃了关系数据库的两个重要特点：结构化查询和事务一致性(ACID),而强化其他一些大型网站更关注的特性：高可用性和可伸缩性。

HBase为可伸缩海量数据存储而设计的，实现面向在线业务的实时数据访问延迟。HBase的伸缩性主要依赖其可分裂的Hregion以及可伸缩性的分布式文件系统HDFS实现。

第七章 随需应变 - 可扩展架构

可扩展性：在对现有系统影响最小的情况下，系统功能可持续扩展或者提升的能力

实现可扩展的手段：低耦合，高内聚

7.1构建可扩展的网站架构

开发低耦合系统是软件设计的终极目标之一，这一目标驱动着软件开发技术的创新与发展。

低耦合的系统更容易扩展，低耦合的模块更容易复用，一个低耦合的系统也会让开发过程和维护变得更加轻松和容易管理。如何分解系统的各个模块、如何定义各个模块的接口、如何复用组合不同的模块构造成一个完整的系统，这是软件设计中最有挑战的部分。

设计网站可扩展架构的核心思想是模块化，并在此基础上，降低模块之间的耦合性，提高模块的复用性。

7.2 利用分布式消息队列降低系统耦合性

* 事件驱动架构（Event Driven Architecture）

定义：通过在低耦合的模块之间传输事件消息，以保持模块的松散耦合，并借助事件消息的通信完成模块间合作。典型的场景是生产着消费者模型，消费者和生产者之间没有直接耦合。

* 分布式消息队列

队列是一种先进先出的数据结构，分布式消息队列可以看作将这种数据结构部署到独立的服务器上，应用程序可以通过远程访问接口使用分布式消息队列，进行消息存取操作，进而实现分布式的异步调用。

1. 消息生产者应用程序通过远程访问接口将消息推送给消息队列服务器，消息队列服务器将消息写入本地内存队列后立即返回成功响应给消息生产者。消息队列服务器根据消息订阅列表查找订阅该消息的消息消费者应用程序，将消息队列中的消息按照先进先出的原则将消息通过远程通信接口发送给消息消费者程序。
2. 在伸缩性方面，由于消息队列服务器上的数据可以被看作是被即时处理的，因此类似于无状态的服务器，伸缩性设计比较简单，将新务器加入分布式消息队列集群中，通知生产者服务器更改消息队列服务器列表即可；
3. 在可用性方面，为了避免消费者进程处理缓慢，分布式消息队列服务器内存空间不足造成的问题，如果内存队列已满，会将消息写入磁盘，消息推送模块在将内存对消息处理完以后，将磁盘内容加载到内存队列继续处理；
4. 为了避免消息队列服务器宕机造成消息丢失，会将消息成功发送到消息队列的消息存储在消息生产者服务器，等消息真正被消息消费者服务器处理后才删除消息，在消息队列服务器宕机后，生产者服务器会选择分布式消息队列服务器集群中其他的服务器发布消息。

7.3分布式服务打造可复用的业务平台

使用分布式服务是降低系统耦合性的一个重要手段。分布式消息队列通过消息对象分解系统耦合性，不同子系统处理同一个消息；那么分布式服务则是通过接口分解系统耦合性，不同子系统通过相同的接口，描述进行服务调用。

要将超大型的、复杂系统拆分成可独立部署的模块，从而降低耦合性。拆分可以分为纵向拆分和横向拆分。

纵向拆分：将一个大应用拆分为多个小应用，如果新增业务较为独立，那么就直接将其设计部署为一个独立的Web应用系统。

横向拆分：将复用的业务拆分出来，独立部署为分布式服务，新增业务只需要调用这些分布式服务，不需要依赖具体的模块代码，即可快速搭建一个应用系统，而模块内业务逻辑变化的时候，只要接口一致就不会影响业务程序和其他模块。

大型网站分布式服务的需求与特点：负载均衡、失效转移、高效的远程通信、整合异构系统、对应用最小侵入、版本管理、实时监控。

* Web Service与企业分布式服务

Web Service比较臃肿，可以考虑使用REST，者使用开源的解决方案，例如Dubbo

7.4可扩展的数据结构

使用NoSQL来替换关系型数据库。

7.5 利用开放平台建设网站生态圈

大型网站为了更好地服务自己的用户，开发更多的增值服务，会把网站内部的服务封装成一些调用接口开放出去 ，供外部的第三方开发者使用，这个提供开放接口的平台被称作开放平台。

第八章 固若金汤 - 安全架构

8.1典型攻击方式

全球大约70%的Web应用攻击来自XSS攻击和SQL注入攻击。

* XSS攻击（跨站脚本攻击）

黑客通过篡改网页，注入恶意HTML脚本，在用户浏览网页时，控制用户浏览器进行恶意操作的一种攻击方式。

① 反射型：攻击者诱使用户点击一个嵌入恶意脚本的链接，达到攻击的目的；②持久型：黑客提交含有恶意脚本的请求，保存在被攻击的web站点的数据库中，用户浏览网页时，恶意脚本被包含在正常页面中，达到攻击的目的。

解决方法：①消毒：消毒几乎是所有网站最必备的XSS防攻击手段。在网站中对脚本进行过滤和消毒处理即对某些html危险字符转义。② HttpOnly：最早由微软提出，即浏览器禁止页面JavaScript访问带有HttpOnly属性的Cookie。HttpOnly并不是直接对抗XSS攻击的，而是防止XSS攻击者窃取Cookie，对于存放敏感信息的Cookie，如用户认证信息等，可以通过对该Cookie添加HttpOnly属性，避免被攻击脚本窃取。

* 注入攻击

① SQL注入攻击：攻击者在HTTP请求中注入恶意SQL命令，服务器用请求参数构造数据库SQL命令时，恶意SQL被一起构造，并在数据库中执行。

SQL注入需要攻击者对数据库结构有所了解才能进行。

解决方法：1） 消毒； 2） 参数绑定

② OS注入攻击：攻击者注入OS命令，利用程序漏洞，达到攻击的目的。

* CSRF攻击（跨站点请求伪造）

CSRF(Cross Site Request Forgery跨站点请求伪造),攻击者通过跨站请求，在用户不知情的情况下，以合法用户的身份进行非法操作。其核心是利用了浏览器Cookie或服务器Session策略，盗取用户身份。

解决方法： 识别请求者身份：1） 表单Token(在请求参数中增加随机数的办法来阻止攻击者获得所有请求参数)； 2） 验证码(请求提交时，需要用户输入验证码，以避免在用户不知情的情况下被攻击者伪造请求)； 3） Referer check(HTTP请求头的Referer域中记录着请求来源，可通过检查请求来源，验证其是否合法)

* 其他攻击方式

① Error Code，可能显示异常堆栈，从而暴露危险信息，解决方法：使用统一的500页面。

② HTML注释，注释可能会暴露危险信息，解决方法：code review或者自动扫描。

③ 文件上传，可能上传病毒文件，解决方法：设置上传文件白名单，只允许上传指定类型的文件。

④ 路径遍历， 在URL中使用相对路径，遍历系统未开放的目录和文件，解决方法： 将资源文件部署在独立的服务器上，使用独立域名。

* Web应用防火墙

ModSercurity是一个开源的Web应用防火墙，探测攻击并保护Web应用程序，既可以嵌入到web应用服务器中，也可以作为一个独立的应用程序启动。

ModSercurity采用处理逻辑与攻击规则集合分离的架构模式。处理逻辑(执行引擎)负责请求和响应的拦截过滤，规则加载执行等功能。而攻击规则集合则负责描述对具体攻击的规则定义、模式识别、防御策略等功能。

处理逻辑比较稳定，规则集合需要不断针对漏洞进行升级，是一种可扩展的架构设计。

* 网站安全漏洞扫描

网站安全漏洞扫描工具是根据内置规则，构造具有攻击性的URL请求，模拟黑客攻击行为，用以发现网站安全漏洞的工具。

8.2信息加密技术以及密钥管理

* 单项散列加密，包括MD5、SHA等

通过对不同输入长度的信息进行散列计算，得到固定长度的输出，这个散列计算过程是单向的，即不能对固定长度的输出进行计算从而获得输入信息；利用单向散列加密的特性，可以进行密码加密保存；为了加强单向散列计算的安全性，还会给散列算法加点salt，salt相当于加密的密钥，增加破解的难度。

* 对称加密， 包括DES算法、RC算法等

对称加密是指加密和解密使用的是同一个密钥，对称加密通常用在信息需要安全交换或存储的场合，如Cookie加密、通信加密。

优点是算法间断，加解密效率高，系统开销小，适合对大量数据加密；缺点是加解密使用同一个秘钥，远程通信的情况下如何安全的交换秘钥是个难题，如果秘钥丢失，那么所有的加密信息也就没有秘密可言了。

* 非对称加密， 包括RSA算法等

非对称加密和解密使用的密钥不是同一个，用公钥加密的信息必须用私钥才能解开，反之，用私钥加密的信息只有用公钥才能解开；非对称加密技术通常用在信息安全传输、数字签名等场合。HTTPS传输中浏览器使用的数字证书实质上是经过权威机构认证的非对称加密的公钥。

① 加密过程为信息发送者A通过公开渠道获得消息接收者B的公钥，对提交信息进行加密，然后通过非安全传输通道将密文信息发送给B，B得到密文信息后，用自己的私钥对信息进行解密，获得原始的明文信息，即使密文信息在传输过程中遭到窃取，窃取者没有解密密钥也无法还原明文。

② 数字签名过程为签名者用自己的私钥对信息进行加密，然后发送给对方，接收方用签名者的公钥对信息进行解密，获得原始明文信息，由于私钥只有签名者拥有，因此该信息是不可抵赖的，具有签名的性质。

* 密钥安全管理

① 将密钥和算法放在一个独立的服务器上，甚至做成一个专用的硬件设置，对外提供加密和解密服务。

② 将加解密算法放在应用系统中，密钥则放在独立服务器中，在存储时，将密钥切分成数片，分别存储在不同的介质中。

应用程序调用秘钥安全管理系统提供的加解密服务接口对信息进行加解密，该接口实现了常用的加解密算法并可根据需求任意扩展。

8.3信息过滤与反垃圾

常见的信息过滤与反垃圾手段有文本匹配、分类算法、黑名单等。

* 文本匹配

文本匹配主要解决敏感词过滤的问题。通常网站维护一份敏感词表，如果用户发表的信息含有列表中的敏感词，则进行消毒处理(将敏感词进行转义变换)或拒绝发表。

快速判断用户信息中是否有敏感词的方法有：正则表达式(用于敏感词少且发布的信息文本长短较短)、双数组Trie算法(本质是确定一个有限状态自动机，然后输入数据进行状态转移)、构造多级Hash表进行文本匹配。有些时候需要对输入的输入的信息进行降噪处理之后再进行匹配。

* 分类算法

先将批量已分类的邮件样本输入分类算法进行训练，得到一个垃圾邮件分类模型，然后利用分类算法结合分类模型对待处理邮件进行识别。(即先训练后识别)

常用的分类算法有贝叶斯算法(假设特征值职安健是独立的)、TAN算法(对贝叶斯算法增加特征值的关联依赖)、ARCS算法(对关联规则进行聚类挖掘)。

分类算法除了用于反垃圾，还可用以信息自动分类。

* 黑名单

将被报告的垃圾邮箱地址放入黑名单，然后针对邮件的发件人在黑名单列表中查找，如果查找成功，则过滤该邮件。黑名单也可以用于信息去重，将如文章标题或者文章关键段落记录在黑名单中，以减少搜索引擎收录重复信息等功能。

黑名单可以通过Hash表或BloomFilter实现。

8.4电子商务风险控制

交易安全是电子商务网站的底线。

* 风险

账户风险、买家风险、卖家风险、交易风险。

* 风控

大型电商网站都配备有专门的风控团队进行风险控制，风控的手段也包括自动和人工两种。机器自动识别为高风险的交易和信息会发送给风控审核人员进行人工审核。

机器自动风控的技术手段主要有规则引擎和统计模型。

规则引擎是一种将业务规则和规则处理逻辑相分离的技术，业务规则文件由运营人员通过管理界面编辑，当需要修改规则时，无需更改代码发布程序，即可实时使用新规则。而规则处理逻辑则调用规则处理输入的数据。

统计模型是目前大型网站更倾向于使用的自动风控手段。风控领域使用的统计模型是使用前面提到的分类算法或者更复杂的机器学习算法进行智能统计。

8.5 小结

网站的相对安全是通过提高攻击门槛达到的。世界上没有固若金汤的网站安全架构，架构师只能每天都打起百分百的精神，预防可能的漏洞或者攻击。