# 大型网站架构演变

应用服务器，文件服务器，数据库服务器在同一台主机上

优化1：应用服务与数据服务分离，3台主机工作。

应用服务器需要处理大量业务，需要快速cpu

文件服务器存储用户上传文件，需要大硬盘

数据库服务器需要快速磁盘检索和数据缓存，需要更快的硬盘和更大内存

优化2：80%业务集中在20%数据上，针对此类数据，缓存内存中

缓存在应用服务器本地内存

优点访问速度快

缺点受到应用服务器内存限制，缓存数量有限，还会出现和应用程序争用内存情况

使用集群方式，部署大内存的服务器作为缓存，不受容量限制

优化3：使用应用服务器集群，改善网站并发能力

优化4：网站使用缓存后，绝大部分读操作访问均可不通过数据库完成，但依然有部分读操作(缓存访问不命中，缓存过期等)和

全部写操作需要访问数据库。

主从热备，实现读写分离

优化5：反向代理和CDN加速网站响应，减轻后端服务器负载压力

原理都是缓存

CDN部署在网络提供商的机房，使用户在请求网站时，可以从距离自己最近的网络提供商机房获取数据

反向代理部署在网站的中心机房，当用户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，返回给用户

优化6：nosql和非数据库查询技术如搜索引擎

优化7：业务拆分

将整个网站业务细分，如交易网站分首页，商铺，订单，卖家，买家等分归不同团队管理

应用之间可以通过超链接建立关系(如：首页导航栏每个指向不同地址)，也可以使用消息队列进行数据分发

还可以通过访问同一数据存储系统构成一个关联完整系统

优化8：分布式服务

由于所有应用需要和数据库系统连接，导致数据库连接资源不足，拒绝服务

共同业务提取出来，独立部署，由可复用的业务连接数据库，提供共用业务服务，

而应用系统只需要管理用户界面，通过分布式服务调用共同业务服务完成具体业务操作

大型网站架构技术的核心价值是随网站所需灵活应对

驱动大型网站技术发展的主要是网站的业务发展

技术是用来解决业务问题的，而业务的问题，也可以通过业务的手段去解决

# 大型网站架构模式

分层：将系统横向维度切分成几个部分，每个部分负责单一职责，通过上层对下层依赖和调用组成一个完整系统

如：计算机硬件，操作系统，软件就是一种分层结构

网站软件系统分为

应用层：负责具体业务和视图展示，网站首页及搜索输入和结果显示，可以细分为视图层/业务逻辑层

服务层：为应用层提供服务支持，用户管理服务，购物车服务等，可以细分为数据接口层(适配各种输入输出数据格式)和逻辑处理层

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

分割：纵向对软件切分

将不同功能和服务切分

如：购物业务，切分为机票酒店业务，小商品业务等更细小的粒度

分布式：可以享受更多资源

问题：服务调用通过网络，影响性能

服务器越多，宕机概率也越大，影响其他服务器

保持数据一致性困难

等

分布式方案：

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

改善网站性能和并发性，加快开发，发布速度，减少数据库连接资源消耗

复用共同服务，便于业务功能扩展

分布式静态资源：图片，js等资源独立分布式部署，并采用独立域名，

减轻应用服务压力，通过独立域名加快浏览器并发加载速度

分布式数据和存储：一台电脑无法提供足够存储空间

分布式计算

还有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

集群：对于用户访问集中的模块，如首页，部署集群

缓存：将数据放在距离计算最近的位置以加快处理速度

CDN:内容分发网络，部署在距离终端用户最近的网络提供商，用户的请求先到达网络服务商哪里，在这里缓存网站一些静态资源，最快速度返回给用户

反向代理:缓存网络静态资源，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:应用服务器本地缓存热点数据，应用程序可以再本机内存直接访问数据，无需访问数据库

分布式缓存：通过网络通信访问缓存数据

使用缓存条件

1：数据访问热点不均衡，某些数据会被频繁访问，这些数据放入缓存

2：数据在某个时间段有效，不会很快过期，否则读取了过期的数据产生脏读，影响结果正确性

异步：系统解耦，业务之间消息传递异步调用，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方式异步执行

异步消息队列，如生产者消费者

提高系统可用性：消费者服务器发生故障，数据会在消息队列堆积，生产者依旧可以处理业务请求，消费者服务器好了后继续处理队列数据

加快网站响应速度：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：将突然增加的访问请求数据放入消息队列，等待消费者服务器依次处理

冗余：保证服务器宕机时依然可以继续服务，不丢失数据。实现高可用

定期备份，冷热备，主从分离

自动化：无人值守情况下网站正常运行

安全

# 大型网站核心架构要素

性能:响应时间,TPS,系统性能计数器等

浏览器端:浏览器缓存，页面压缩，合理布局界面，减少cookie传输等

CDN/反向代理服务器

应用服务器端：本地缓存/分布式缓存/异步操作将用户请求发送至消息队列等待后续处理/集群

代码层面：多线程、改善内存管理等

数据库服务器端：索引，缓存，sql优化等

高可用

当服务器宕机，服务器依旧可用

冗余解决：部署多台服务器同时访问，数据存储在多台服务器上备份，任意一台宕机无法影响整体使用，也不会数据丢失

应用服务器：前提条件应用服务器上能保存请求的会话信息，否则服务器宕机，会话丢失，将用户请求信息转发到其他服务器也无法完成业务处理

存储服务器：由于存储数据，需要对数据进行实时备份，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的时候数据依然有用

伸缩性：通过不断向集群加入服务器缓解用户并发访问压力和数据存储

衡量架构伸缩性标准

1：是否可以用多台服务器构建集群

2：是否容易向集群添加新的服务器

3：加入新的服务器后是否可以提供和原来的服务器无差别服务

4：集群可容纳的总服务器数量是否有限制

应用服务器集群：只要服务器上不存储数据，服务器对等，使用负载均衡设备就可以向集群添加

缓存服务器集群：加入新的服务器可能导致缓存路由失效，进而导致集群中大部分缓存数据无法访问

缓存的数据可以通过数据库重新加载，但是如果应用已经严重依赖缓存，会导致网站崩溃，

需要改进缓存路由算法保证缓存数据的可访问性

关系数据库：虽然支持数据复制，主从热备等机制，但是很难做到大规模集群可伸缩性，

所以只能在之外实现，通过路由分区等手段将部署由多个数据库的服务器组成集群

扩展性

网站增加新的业务产品时,是否可以实现对现有产品透明无影响,不需要改动or改动较少既有业务功能就能上线新产品

主要手段：事件驱动架构，通常使用消息队列

分布式服务,将业务和可复用服务分离,通过分布式框架调用。

新增产品可以调用可复用的服务实现自身逻辑，对现有产品无影响。

可复用服务升级时，也可以通过提供多版本服务对应用实现透明升级，不需要强制应用同步变更

安全性

架构

## 网站性能测试

用户:关注尽快显示内容

手段:浏览器缓存策略,CDN,反向代理等

开发人员:关注应用程序本身以及相关子系统性能,包括延迟,吞吐量,系统稳定性等,

手段: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

运维人员:基础设施性能和资源利用率,如带宽,服务器硬件,数据中心网络架构等,

手段:建设优化骨干网,使用高性能服务器,利用虚拟化技术优化资源利用等

性能测试指标

响应时间

并发数:系统能够同时处理请求的数目

吞吐量:单位时间内系统处理的请求数量

如:请求数/秒,页面数/秒,访问人数/天,处理的业务数/小时等来衡量

TPS每秒事务数

HPS每秒HTTP请求数

QPS每秒查询数

类似:

吞吐量是每天通过收费处的车辆数目

并发数是高速公路正在行驶的车辆数目

响应时间是车速

性能计数器:描述服务器或操作系统性能的一些数据指标

如:system Load(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,反映系统忙闲程度的重要指标

对象,线程数,内存使用,cpu使用,磁盘与网络I/O等

性能测试方法

性能测试:验证系统资源可接受范围内,是否达到性能预期

负载测试:增加并发请求,测试系统最大能承受负载压力

压力测试:超过负载情况下,对系统继续施压,直到系统崩溃,获得系统最大承受能力

稳定性测试:给系统加载一定业务压力,运行一段时间，检验系统是否稳定。

由于不同时间内请求的压力是不均匀的,所以稳定性测试应不均匀给系统施加压力

性能优化策略

性能分析:对请求每个环节排查

性能优化:web前端优化,应用服务器优化,存储服务器优化

WEB前端优化

浏览器访问优化

减少HTTP请求:合并CSS,合并js,合并图片(通过CSS偏移响应鼠标点击操作构造不同URL)。

浏览器缓存:缓存静态资源,通过Cache-Control和Expires属性,设置缓存时间

静态资源变化需要及时更新到浏览器上,方法有

改变文件名,即更新js文件而不是js文件内容,生成新的js文件,并更新引用

更新静态资源时,采用一个个更新,并有间隔时间

避免用户浏览器大量缓存失效,造成服务器负载剧增,网络堵塞情况

启用压缩:服务端对文件压缩,浏览器对文件解压,html/CSS/JS启用GZIP,

CSS放在页面最上面，js放在页面最下面

减少cookie传输和cookie数据量:静态资源采用独立域名访问,避免请求静态资源发送cookie,减少cookie传输

CDN加速:能够缓存的一般是静态资源

反向代理:保护网站安全,缓存动静资源,可实现负载均衡

等

应用服务器性能优化

分布式缓存

缓存:本质内存hashTable

合理使用缓存:

频繁修改的数据:只缓存读写比在2:1的数据

没有热点的访问:只缓存热点数据

数据不一致与脏读:设置失效时间or及时更新缓存

缓存可用性:搭建集群

缓存预热:在缓存系统启动时把热点数据加载好

热点数据是缓存系统使用LRU(最近最久未用算法)对不断访问的数据筛选淘汰出来的

缓存穿透:因为不恰当业务Or高并发访问不存在的数据,而也没有缓存,所有请求会落到数据库上

解决:将不存在数据缓存起来,value=null

分布式缓存架构:

1:更新同步的分布式缓存(jboss cache)

jboss cache通常和应用程序部署同一台服务器,应用程序可快速从本地获取数据

缺点:受限服务器内存空间,集群规模较大时,缓存更新信息需要同步到集群所有机器。

2:不互相通信的分布式缓存(memcache)

异步操作

用户请求后数据发送给消息队列后立即返回,在由消息队列的消费者进程从消费队列中获取数据,异步写入数据库

处理速度远超数据库

缺点:数据写入消息队列后立即返回给用户,数据在后续的业务校验中写数据库可能失败,

如:订单提交后,订单数据写入消息队列,不能立即返回用户订单提交成功,需要消息队列执行完成后通知用户订单状态

任何可以晚点做的事都应该晚点做

集群

代码优化

多线程:解决网络响应,需要等待磁盘操作,采用多线程提高任务并发度,系统吞吐能力

(任务执行时间/任务执行时间-IO等待时间)\*CPU内核数=启动线程数

解决线程安全问题(多线程并发对某资源进行修改)

1:对象设计为无状态对象,无状态指对象本身不存储状态信息(对象无成员变量Or成员变量也是无状态对象)

2:使用局部对象

3:并发访问资源时用锁

资源复用:减少开销很大的系统资源创建和销毁,如数据库连接,网络通信连接,线程,复杂对象等

单例模式

对象池

数据结构:hashTable

Time33算法:对字符串逐字符迭代\*33求得hashcode值

md5计算:原始字符串使用MD5获得信息指纹,hash计算获得hashcode

垃圾回收:JVM垃圾回收机制

存储性能优化

固态硬盘(快速顺序读写,慢速随机读写) VS SSD硬盘

B+树 VS LSM树

RAID(改善磁盘访问延迟,增强磁盘可用性和容错能力) VS HDFS

# 网站高可用架构

网站可用性的度量与考核

网站可用性度量

网站不可用时间=故障修复时间点-故障发现时间点

网站年度可用性指标=(1-网站不可用时间/年度总时间)\*100%

网站可用性考核

故障分=故障时间(分钟)\*故障权重

高可用网站架构

主要目的: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访问

数据和服务冗余备份及失效转移

高可用应用

应用层

无状态性:不保存业务上下文信息(web称之为session),仅根据每次请求提交的数据进行相应的业务逻辑处理

应用服务器集群的session管理

1:session复制:服务器开启web容器的session复制功能,在集群中的几台服务器中间同步session对象

使得每台服务器都保存session信息,任何一台宕机都不会导致session数据丢失

集群多,不适用

2:session绑定(源地址hash算法实现):ip-hash,会话粘滞

一台服务器宕机,该session不存在

3:cookie记录session

受cookie大小限制,每次传输都需要传cookie,影响性能

用户关闭cookie,访问异常。

4:session服务器:分布式缓存,数据库等,sso

高可用服务策略

1:分级管理

重要的服务用好设备,核心服务和数据部署不同地域数据中心

2:超时设置,应用程序中设置服务调用超时时间,超时通信框架抛出异常,根据异常选择继续重试or请求转移

3:异步调用,队列

4:服务降级,网站访问高峰时,性能下降,通过各种手段提升资源可用率

拒绝服务:拒绝优先级低应用调用,随机拒绝部分请求

关闭功能:关闭不重要服务

5:幂等性设计:应用服务调用失败后,会将调用请求重新发送到其他服务器。有时调用服务后完成了操作只是来不及响应

当判断调用失败后,重新调用其他服务器时,设计到转账操作,则会出现问题

针对一样设置的结果不用理会

针对转账操作可以使用服务调用有效性校验

高可用的数据

含义

数据持久化:写入数据时持久化存储,还需要将数据备份1个或多个副本

数据可访问性:一个数据存储设备损坏,需要将数据访问切换到另外一个设备上,时间较长的话，期间不可访问

数据一致性:数据有多份副本情况,如果网络,服务器,软件出现问题,会导致副本部分写入成功,造成副本之间数据不一致

数据强一致:各个副本的数据在物理存储中总是一致,数据更新操作结果和操作响应总是一致的,操作失败就是失败,而不是不确定状态

数据用户一致:各个副本数据可能不一致,终端用户访问时,通过纠错和校验机制,确定一个一致且正确数据给用户

数据最终一致:存储数据不一致,用户访问也不一致(访问多次结果不同),系统经过一段时间的自我恢复和修正,数据最终一致

数据不一致出现于高并发,集群状态不稳定(故障恢复,集群扩容等)

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,数据可用性,分区耐受性(系统具有跨网络分区的伸缩性)

数据备份:冷备:定期复制到存储介质

同步热备:多份副本写操作同步完成

异步热备:master/slave

失效转移:

失效确认

访问转移

数据恢复

# 网站伸缩性架构

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or缩小网站服务处理能力

网站伸缩性设计成2类

根据功能进行物理分离实现伸缩,不同服务器部署不同任务

纵向分离(分层后分离):将业务处理流程上的不同部分分离部署,实现系统伸缩性

横向分离(业务分割后分离):不同业务模块分离部署,实现系统伸缩性

单一功能通过集群实现伸缩,集群内多台服务器部署相同任务,提供相同功能

集群伸缩性分为应用服务器集群伸缩性,数据服务器集群伸缩性

应用服务器集群伸缩性设计

HTTP重定向负载均衡

DNS域名解析负载均衡

反向代理负载均衡:一切依靠自身转发完成,转发都在HTTP协议层面，也称之为应用层负载均衡

web服务器不直接对外访问,所以不需要外部地址.反向代理服务器需要双网卡和内部外部2套IP地址

ip负载均衡:在网络层通过修改请求目标地址进行负载均衡

用户请求到达负载均衡服务器后-->负载均衡服务器算出真实web服务器,将数据目的ip修改为真实web服务器ip

真实web服务器处理完成后响应负载均衡服务器-->负载均衡服务器将数据包源地址修改自身iP发送用户

2种真实web服务器响应数据包返回给负载均衡服务器

1:负载均衡服务器修改目的IP地址,将数据源地址设为自身ip(原地址转换SNAT)

2:负载均衡服务器作为真实物理服务器集群的网关服务器

数据链路层负载均衡:通信协议链路层修改mac地址进行负载均衡

真实物理服务器ip和负载均衡服务器ip一样

用户请求负载均衡服务器-->负载均衡服务器将请求数据目的mac地址修改为真实服务器地址

数据传输到真实服务器地址-->完成后发送响应数据到网站网关服务器-->发送用户

响应过程中不需要负载均衡服务器

LVS产品支持

常见负载均衡算法

步骤

1:根据负载均衡算法和web服务器列表计算得到集群其中一台web服务器地址

2:请求数据发送到该对应服务器上

轮询/加权轮询

随机/加权随机

最少连接/加权:记录每个应用服务器正在处理连接数,将新请求分发到最少连接服务器上

原地址散列(ip-hash)

分布式缓存集群伸缩设计

由于分布式缓存集群中缓存数据不同,所以缓存访问请求不能再缓存服务器中任意一台处理

所以造成新上线的没缓存数据,下线的还换成数据.制约伸缩性

设计目标:新加入缓存服务器后应使整个缓存服务器集群中已经缓存的数据尽可能还被访问到

解决方法1:网站访问量最少时扩容缓存服务器集群,使用模拟请求预热

路由算法

hash取余:命中率低至(N/N+1)

一致性hash算法(一致性hash环):通常使用二叉查找树实现。命中率高至99%

缺点:新加入节点只影响附近最近的节点，其他节点还是负载那么多

添加虚拟层,每个物理服务器节点虚拟为一组缓存服务器,然后根据hash值放置hash环上

命中高达75%,同时解决资源分配不合理问题

数据存储服务器集群伸缩性设计

关系数据库集群伸缩:主从复制,读写分离

分库:不同业务数据表部署在不同数据库集群上,垮库不能join)

分表:一张表拆开分别存储在多个数据库中,cobar,amoeba)

数据迁移(利用hash一致算法)

nosql数据库伸缩

# 网站的可扩展架构

扩展性:对现有系统最小影响下,系统功能可持续扩展or提升的能力

伸缩性:系统通过增加or减少自身资源规模的方式增强or减少自己计算处理事务的能力

增减成比例,称为线性扩展。通常使用集群伸缩

构建可扩展的网站架构

降低耦合提交复用

低耦合:分层,分割

聚合方式:分布式消息队列,分布式服务

利用分布式消息队列降低系统耦合性

事件驱动架构:通过在低耦合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

生产者消费者模式:如分布式消息队列activeQ,利用Mysql写入时间排序,也可实现

利用分布式服务打造可复用业务平台

巨无霸系统缺点

编译,部署困难

代码分支管理困难

数据库连接耗尽

新增业务困难

解决方案:拆分,独立部署,降低系统耦合性

纵向拆分:大应用拆分多个小应用,业务独立则独立部署

横向拆分:复用业务拆分,独立部署为分部署服务,新增业务只需调用分布式服务即可

webservice与企业级分布式服务

缺点:

臃肿注册与发现机制

低效XML序列化

开销较高的HTTP远程通信

复杂部署与维护

大型网站分布式服务的需求与特点

负载均衡/失效转移/高效远程通信/整合异构系统(不同语言实现的平台)

对应用最少侵入/版本控制(对低版本的服务兼容)/实时监控

分布式服务框架设计

dubbo:使用nio

可扩展的数据结构

设计表需要指出表的字段名,类型,僵硬了数据结构和无法面对复杂需求

解决1(不可取):sql冗余字段应对需求

解决2(推荐):列族

hbase:创建表时只需指出列族名,无需指定字段,可以再数据写入时指定,数据接口可扩展,查询时可指定任意字段名和值查询

利用开放平台建设网站生态圈

开放平台:网站内部服务封装成调用接口开放,供第三方使用

开放平台架构设计:

API接口:restful,webservice,RPC等

协议转换:各种api输入转换可识别形式,并将内部服务的返回封装成api形式

安全:身份校验,权限控制,资源分配

审计:记录第三方访问情况,监控计费,日志

路由:各种访问路由映射到具体内部服务

流程:一组离散服务组织成一个上下文相关的新服务,隐藏服务细节,提供统一接口供第三方用

# 网站的安全架构

安全与防护

XSS攻击(垮站点脚本攻击):篡改网页,注入恶意HTML脚本,用户浏览时,控制用户浏览进行恶意操作

反射攻击:攻击者诱使用户点击一个嵌入恶意脚本的链接,达到攻击目的

如:偷取用户cookie,密码,伪造交易,窃取情报

持久性攻击: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,恶意脚本被包含在正常页面中,达到攻击目的

如:论坛,博客等

防守

消毒:对HTML危险字符转义,>转"&gt"

HTTPOnly:浏览器禁止页面js访问带有httpOnly属性的cookie,防止xss攻击者窃取cookie

cookie添加httpOnly属性即可

注入攻击

OS注入

SQL注入:需要攻击者对数据库结构有了解,攻击者获取数据库表结构信息手段如下

开源/错误回显/

解决:

消毒:正则匹配过滤(如drop table)

参数绑定

CSRF攻击(跨站点请求伪造):攻击者通过跨站请求,以合法用户身份进行非法操作,如转账交易,发表评论等

解决:识别请求者身份

表单token:表单中添加随机数作为token,每次请求相应页面token也不同,从正常页面请求会包含token,伪造的则无法获得,服务器检测是否有token即可

验证码:

Referer check:Http的refer域记录请求来源,检查请求来源,验证合法性(如:反盗链)

其他攻击和漏洞

错误回显(Error Code)/HTML注释/

文件上传:解决设置上传文件白名单,只允许上传可靠文件类型,修改文件名,使用专门存储等

路径遍历:攻击者在请求的url中使用相对路径,遍历系统未开放的目录和文件。

解决:静态资源独立部署,独立域名,其他文件不使用静态URL访问,动态参数不包含文件路径信息

web应用防火墙

modsecurity:开源web应用防火墙,探测攻击并保护web应用程序

网站安全漏洞扫描

模拟黑客攻击

信息加密技术与秘钥安全管理

信息加密技术分为3类:

单列散列加密:密码加密,加salt,如md5,sha

对称加密:cookie加密,通信加密,如des,rc

非对称加密:信息安全传输,数字签名,rsa

密钥安全管理:

密钥和算法放在独立服务器,甚至做成专用硬件,对外提供加密和解密服务

(推荐)加解密算法放在应用系统中,密钥放在独立服务中,实际存储时密钥分成数片,加密后保存在不同存储介质中。

信息过滤与反垃圾

文本匹配:对敏感词转义成其他or拒绝发表

方法:正则(效率一般)

Trie数:双数组trie

构建多级hash表文本匹配

分类算法:批量将已分类的邮件样本输入分类算法训练,获得垃圾邮件分类模型,利用分类算法结合分类模型对垃圾邮件识别

如:贝利叶分类算法,

TAN算法

ARCS算法

还可以用文档类别分类

黑名单:将被报告的垃圾邮箱地址加入黑名单,针对邮件发件人在黑名单中查找

消息去重.如将文章标题Or文章关键段落记录黑名单,减少搜索引擎收录重复等

hash表实现

布隆过滤器

电子商务风险控制

账户风险/卖家买家风险/交易风险

风控

规则引擎:交易某些指标满足,被认为高风险欺诈(如:用户来自高欺诈地区,登录地点变)

交易规则可以用if..else判断,还可以用规则引擎

将业务规则和规矩规则逻辑相分离的技术

统计模型:分类算法,机器学习算法,

# 网站秒杀架构设计

技术挑战

对现有网站业务冲击

高并发下的应用,数据库负载

增加的网络以及服务器带宽

直接下单

策略

秒杀系统独立部署

秒杀商品页面静态化,缓存在CDN,反向代理以及浏览器上

租借秒杀活动网络带宽

动态生成随机下单页面URL,在下单页面url加入由服务器端生成的随机数作为参数,在秒杀开始时才能办到

架构设计

页面尽量简单

点击购买后按钮变暗,不可以点击

下单表单简单,购买数量只能一个且不能更改

送货地址付款方式使用用户默认,允许订单提交后修改

随机用户提交表单后进入秒杀结束页面

如何控制秒杀商品页面购买按钮的点亮

服务端构造响应页面输出

js控制,秒杀开始时生成新的js被浏览器加载

控制只允许固定次数的用户秒杀

memcache统计次数，超过次数显示秒杀结束页面

# 大型网站典型故障案例分析

写日志引发故障

应用程序自己的日志输入配置和第三方组件日志输出分别配置

log配置文件,日志输出级别至少warn

关闭第三方日志输出

高并发访问数据库故障

首页不应该访问数据库,首页需要的数据从缓存服务器和搜索引擎服务器获取

首页静态

高并发锁引发故障

锁谨慎用

缓存故障

缓存不仅仅改善性能,而是成为网站架构不可或缺的一部分时,对缓存管理上升到其他服务器一样级别

应用启动不同步故障

启动后台在启动前台

大文件读写独占磁盘故障

根据不同文件类型和用途进行管理,小图片不能和大文件共用存储

滥用生产环境

访问线上生产环境规范

不规范流程

提交代码时确认没有提交不该提交代码

加强code review

不好编程习惯

空指针判断

输入对象尽量保证null,必要时构造空对象