001. Redis简单介绍

介绍：Redis是一个开源的使用ANSIC语言编写、遵守BSD协议、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API的非关系型数据库。

传统数据库遵循ACID规则。而Nosql（NotOnlySQL的缩写，是对不同于传统的关系型数据库的数据库管理系统的统称）一般为分布式而分布式一般遵循CAP定理。

002. Redis支持的数据类型

String字符串：

格式:setkeyvalue

string类型是二进制安全的。意思是redis的string可以包含任何数据。比如jpg图片或者序列化的对象。

string类型是Redis最基本的数据类型，一个键最大能存储512MB。

Hash（哈希）

格式:hmsetnamekey1value1key2value2

Redishash是一个键值(key=>value)对集合。

Redishash是一个string类型的field和value的映射表，hash特别适合用于存储对象。

List（列表）

Redis列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）

格式:lpushnamevalue

在key对应list的头部添加字符串元素

格式:rpushnamevalue

在key对应list的尾部添加字符串元素

格式:lremnameindex

- redis 是一种高级的 key-value 的存储系统，其中 value 支持五种数据类型。
  1、字符串（String）
  2、哈希（hash）
  3、字符串列表（list）
  4、字符串集合（set）
  5、有序字符串集合（sorted set）
- 而关于 key 的定义呢，需要大家注意的几点：
  1、key 不要太长，最好不要操作 1024 个字节，这不仅会消耗内存还会降低查找效率
  2、key 不要太短，如果太短会降低 key 的可读性
  3、在项目中，key 最好有一个统一的命名规范

003. 什么是Redis持久化？Redis有哪几种持久化方式？优缺点是什么？

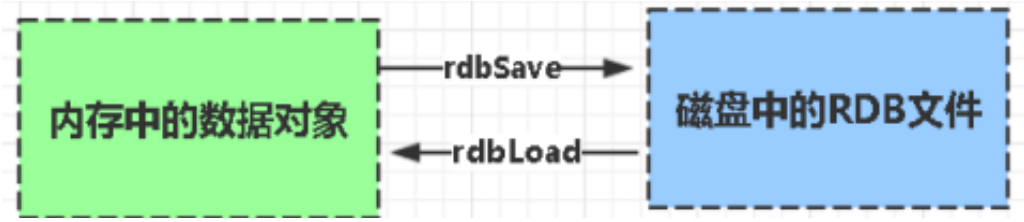
持久化就是把内存的数据写到磁盘中去，防止服务宕机了内存数据丢失。

Redis提供了两种持久化方式:RDB（默认）和AOF

RDB：

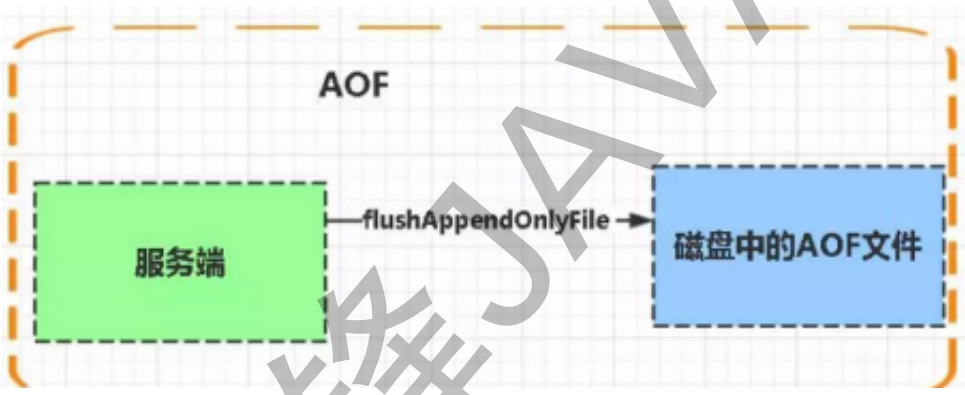
rdb是RedisDataBase缩写

功能核心函数rdbSave(**生成RDB文件**)和rdbLoad（从文件加载内存）两个函数



AOF:

Aof是Append-onlyfile缩写



每当执行服务器(定时)任务或者函数时flushAppendOnlyFile函数都会被调用，这个函数执行以下两个工作

aof写入保存：

WRITE：根据条件，将aof_buf中的缓存写入到AOF文件

SAVE：根据条件，调用fsync或fdatasync函数，将AOF文件保存到磁盘中。

存储结构:

内容是redis通讯协议(RESP)格式的命令文本存储。

比较：

1、aof文件比rdb更新频率高，优先使用aof还原数据。

2、aof比rdb更安全也更大

3、rdb性能比aof好

4、如果两个都配了优先加载AOF

004 redis通讯协议(RESP)，能解释下什么是RESP？有什么特点？

RESP是redis客户端和服务端之前使用的一种通讯协议；

RESP的特点：实现简单、快速解析、可读性好

ForSimpleStringsthefirstbyteofthereplyis"+"回复

ForErrorsthefirstbyteofthereplyis"-"错误

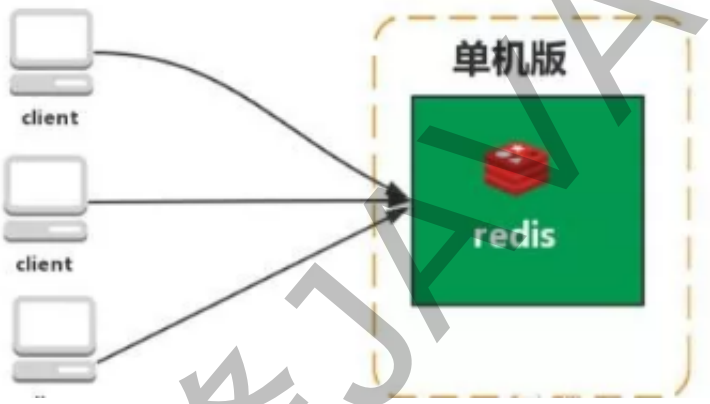
ForIntegersthefirstbyteofthereplyis":"整数

ForBulkStringsthefirstbyteofthereplyis"$"字符串

ForArraysthefirstbyteofthereplyis"*"数组

005. Redis有哪些架构模式？讲讲各自的特点

单机版

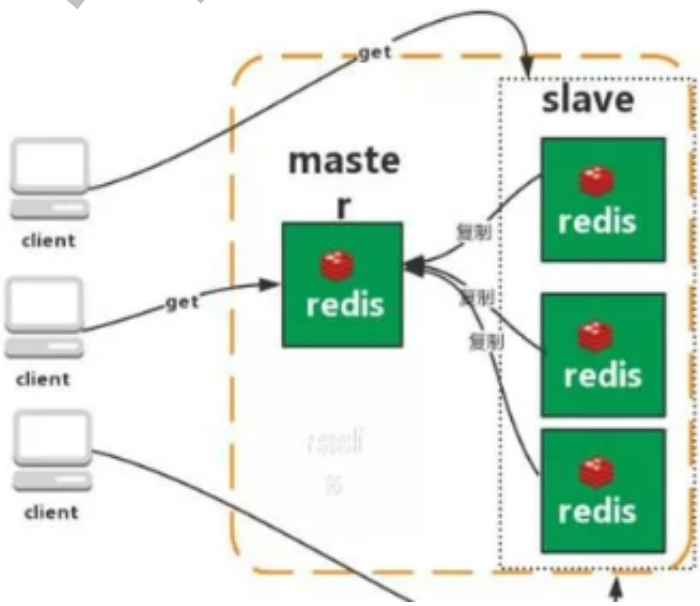


特点：简单

问题：

内存容量有限2、处理能力有限3、无法高可用。

主从复制



Redis的复制（replication）功能允许用户根据一个Redis服务器来创建任意多个该服务器的复制品，其中被复制的服务器为主服务器（master），而通过复制创建出来的服务器复制品则为从服务器（slave）。只要主从服务器之间的网络连接正常，主从服务器两者会具有相同的数据，主服务器就会一直将发生在自己身上的数据更新同步给从服务器，从而一直保证主从服务器的数据相同。

特点：

1、master/slave角色

2、master/slave数据相同

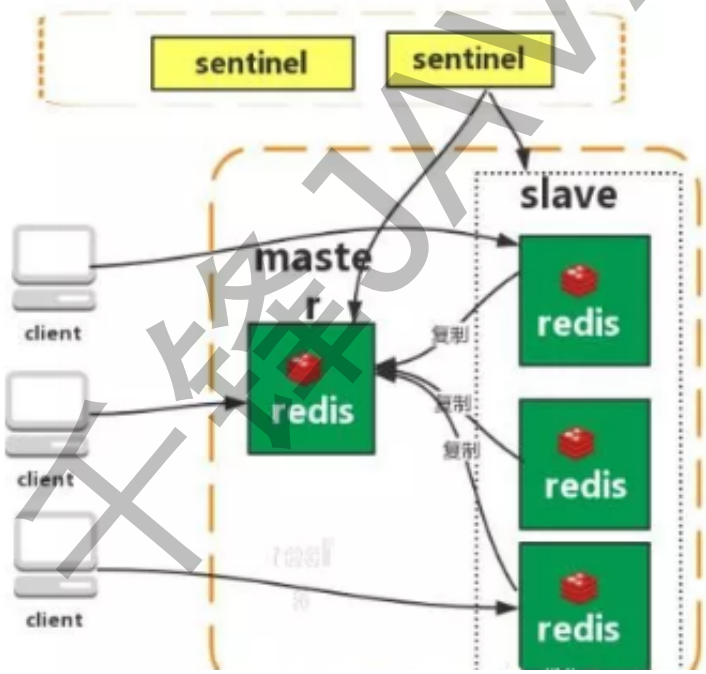
3、降低master读压力在转交从库

问题：

无法保证高可用

没有解决master写的压力

哨兵



Redissentinel是一个分布式系统中监控redis主从服务器，并在主服务器下线时自动进行故障转移。其中三个特性：

监控（Monitoring）： Sentinel会不断地检查你的主服务器和从服务器是否运作正常。

提醒（Notification）： 当被监控的某个Redis服务器出现问题时，Sentinel可以通过API向管理员或者其他应用程序发送通知。

自动故障迁移（Automaticfailover）： 当一个主服务器不能正常工作时，Sentinel会开始一次自动故障迁移操作。

特点：

1、保证高可用

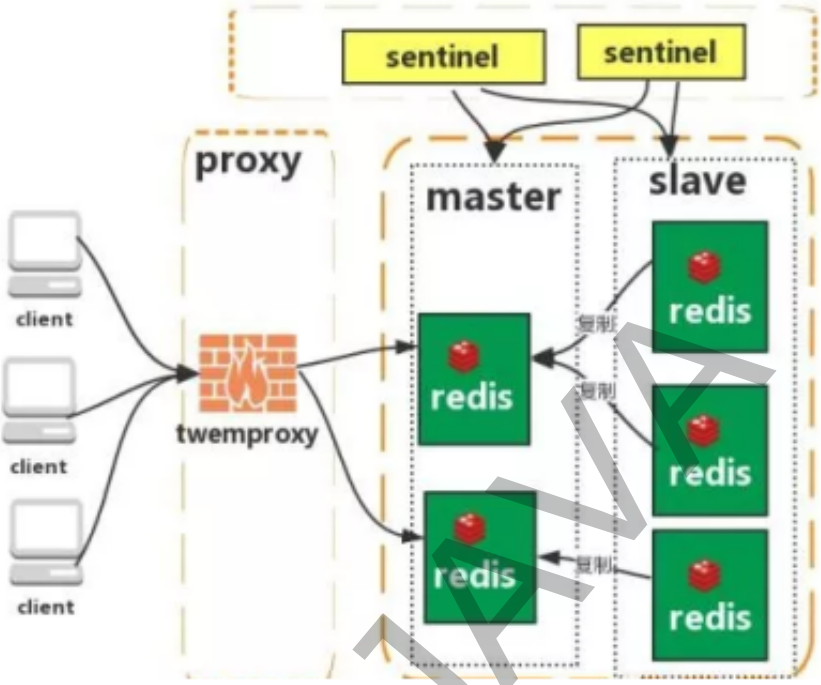
2、监控各个节点

3、自动故障迁移

缺点：主从模式，切换需要时间丢数据

没有解决master写的压力

集群（proxy型）



Twemproxy是一个Twitter开源的一个redis和memcache快速/轻量级代理服务器；Twemproxy是一个快速的单线程代理程序，支持MemcachedASCII协议和redis协议。

特点：1、多种hash算法：MD5、CRC16、CRC32、CRC32a、hsieh、murmur、Jenkins

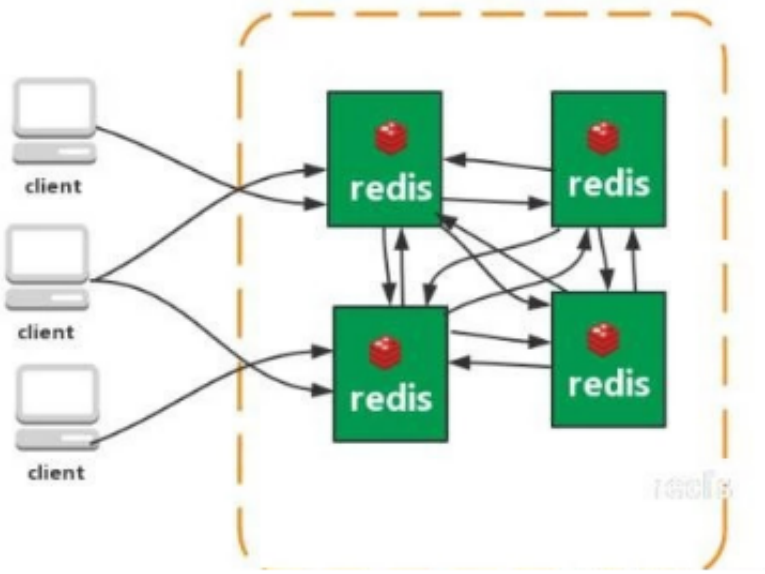
2、支持失败节点自动删除

3、后端Sharding分片逻辑对业务透明，业务方的读写方式和操作单个Redis一致

缺点：增加了新的proxy，需要维护其高可用。

failover逻辑需要自己实现，其本身不能支持故障的自动转移可扩展性差，进行扩缩容都需要手动干预.

集群（直连型）：



从redis3.0之后版本支持redis-cluster集群，Redis-Cluster采用无中心结构，每个节点保存数据和整个集群状态,每个节点都和其他所有节点连接。

特点：

1、无中心架构（不存在哪个节点影响性能瓶颈），少了proxy层。

2、数据按照slot存储分布在多个节点，节点间数据共享，可动态调整数据分布。

3、可扩展性，可线性扩展到1000个节点，节点可动态添加或删除。

4、高可用性，部分节点不可用时，集群仍可用。通过增加Slave做备份数据副本

5、实现故障自动failover，节点之间通过gossip协议交换状态信息，用投票机制完成Slave到Master的角色提升。

缺点：

1、资源隔离性较差，容易出现相互影响的情况。

2、数据通过异步复制,不保证数据的强一致性

006. 什么是一致性哈希算法？什么是哈希槽?

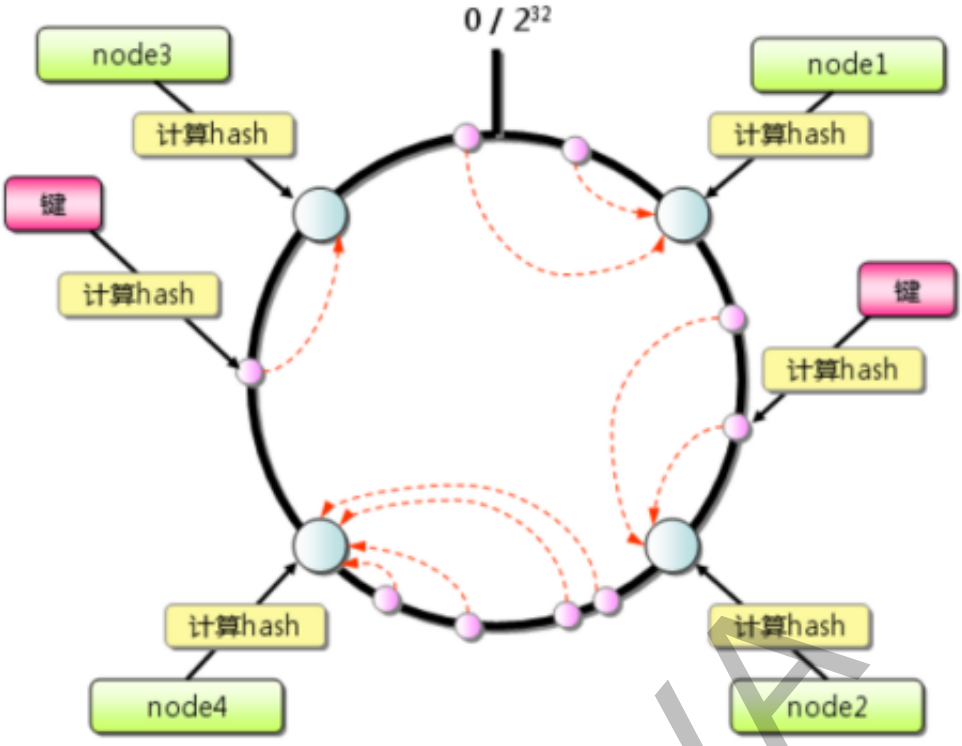
一致性哈希算法在1997年由麻省理工学院的Karger等人在解决分布式Cache中提出的，设计目标是为了解决因特网中的热点(Hotspot)问题，初衷和CARP十分类似。一致性哈希修正了CARP使用的简单哈希算法带来的问题，使得DHT可以在P2P环境中真正得到应用。

但现在一致性hash算法在分布式系统中也得到了广泛应用，研究过memcached缓存数据库的人都知道，memcached服务器端本身不提供分布式cache的一致性，而是由客户端来提供，具体在计算一致性hash时采用如下步骤：

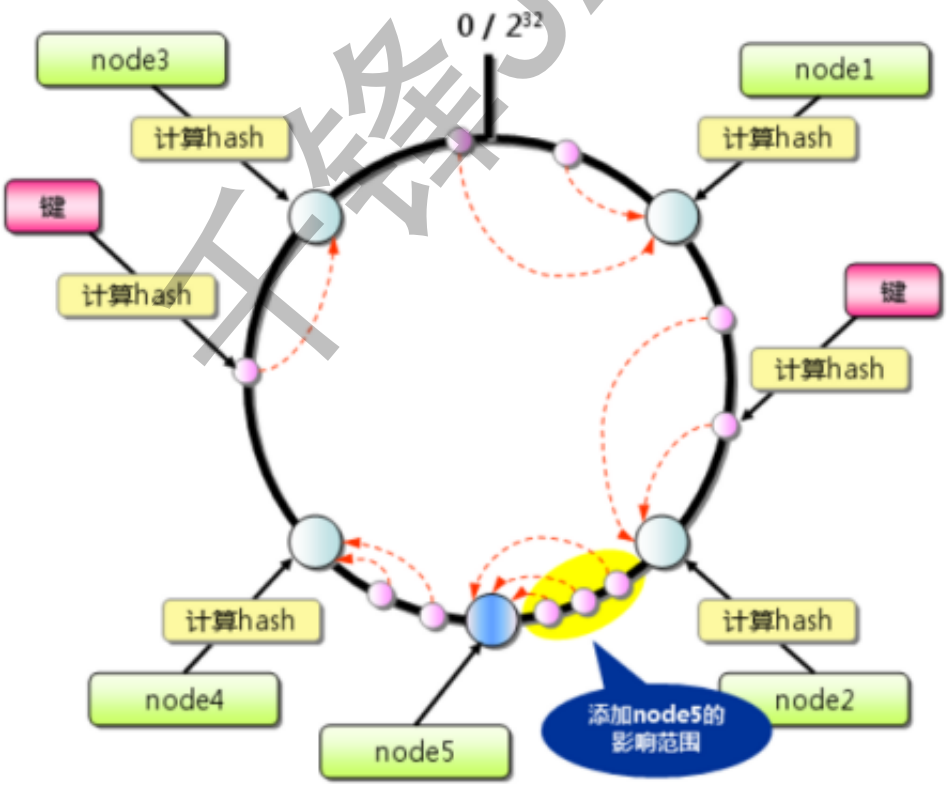
首先求出memcached服务器（节点）的哈希值，并将其配置到0～232的圆（continuum）上。

然后采用同样的方法求出存储数据的键的哈希值，并映射到相同的圆上。

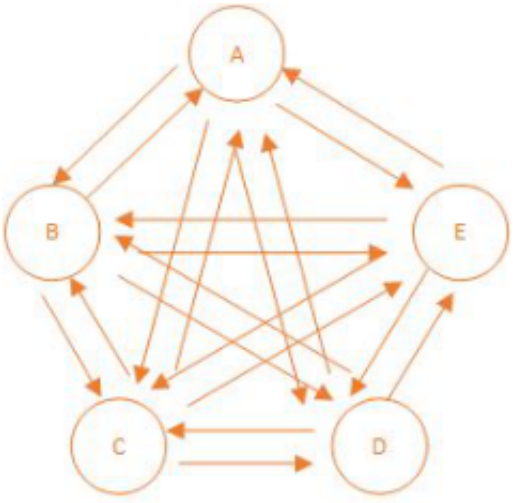
然后从数据映射到的位置开始顺时针查找，将数据保存到找到的第一个服务器上。如果超过232仍然找不到服务器，就会保存到第一台memcached服务器上。



从上图的状态中添加一台memcached服务器。余数分布式算法由于保存键的服务器会发生巨大变化而影响缓存的命中率，但ConsistentHashing中，只有在园（continuum）上增加服务器的地点逆时针方向的第一台服务器上的键会受到影响，如下图所示：



从redis3.0之后版本支持redis-cluster集群，Redis-Cluster采用无中心结构，每个节点保存数据和整个集群状态,每个节点都和其他所有节点连接。



Gossip in Redis Cluster

其结构特点：

1、所有的redis节点彼此互联(PING-PONG机制),内部使用二进制协议优化传输速度和带宽。

2、节点的fail是通过集群中超过半数的节点检测失效时才生效。

3、客户端与redis节点直连,不需要中间proxy层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可。

4、redis-cluster把所有的物理节点映射到[0-16383]slot上（不一定是平均分配）,cluster负责维护node<->slot<->value。

5、Redis集群预分好16384个桶，当需要在Redis集群中放置一个key-value时，根据CRC16(key)mod16384的值，决定将一个key放到哪个桶中。

007. Redis常用命令?

Keyspattern

*表示区配所有

以bit开头的

查看Existskey是否存在

Set

设置key对应的值为string类型的value。

setnx

设置key对应的值为string类型的value。如果key已经存在，返回0，nx是notexist的意思。

删除某个key

第一次返回1删除了第二次返回0

Expire设置过期时间（单位秒）

TTL查看剩下多少时间

返回负数则key失效，key不存在了

Setex

设置key对应的值为string类型的value，并指定此键值对应的有效期。

Mset

一次设置多个key的值，成功返回ok表示所有的值都设置了，失败返回0表示没有任何值被设置。

Getset

设置key的值，并返回key的旧值。

Mget

一次获取多个key的值，如果对应key不存在，则对应返回nil。

Incr

对key的值做加加操作,并返回新的值。注意incr一个不是int的value会返回错误，incr一个不存在的key，则设置key为1

incrby

同incr类似，加指定值，key不存在时候会设置key，并认为原来的value是0

Decr

对key的值做的是减减操作，decr一个不存在key，则设置key为-1

Decrby

同decr，减指定值。

Append

给指定key的字符串值追加value,返回新字符串值的长度。

Strlen

取指定key的value值的长度。

persistxxx(取消过期时间)

选择数据库（0-15库）

Select0//选择数据库

moveage1//把age移动到1库

Randomkey随机返回一个key

Rename重命名

Type返回数据类型

008. 使用过Redis分布式锁么，它是怎么实现的?

先拿setnx来争抢锁，抢到之后，再用expire给锁加一个过期时间防止锁忘记了释放。

如果在setnx之后执行expire之前进程意外crash或者要重启维护了，那会怎么样?

set指令有非常复杂的参数，这个应该是可以同时把setnx和expire合成一条指令来用的!

009.使用过Redis做异步队列么，你是怎么用的？有什么缺点?

一般使用list结构作为队列，rpush生产消息，lpop消费消息。当lpop没有消息的时候，要适当sleep一会再重试。

缺点:

在消费者下线的情况下，生产的消息会丢失，得使用专业的消息队列如rabbitmq等。

能不能生产一次消费多次呢?

使用pub/sub主题订阅者模式，可以实现1:N的消息队列。

010.什么是缓存穿透？如何避免？什么是缓存雪崩？何如避免?

缓存穿透

一般的缓存系统，都是按照key去缓存查询，如果不存在对应的value，就应该去后端系统查找（比如DB）。一些恶意的请求会故意查询不存在的key,请求量很大，就会对后端系统造成很大的压力。这就叫做缓存穿透。

如何避免?

1：对查询结果为空的情况也进行缓存，缓存时间设置短一点，或者该key对应的数据insert了之后清理缓存。

2：对一定不存在的key进行过滤。可以把所有的可能存在的key放到一个大的Bitmap中，查询时通过该bitmap过滤。

缓存雪崩

当缓存服务器重启或者大量缓存集中在某一个时间段失效，这样在失效的时候，会给后端系统带来很大压力。导致系统崩溃。

如何避免?

1：在缓存失效后，通过加锁或者队列来控制读数据库写缓存的线程数量。比如对某个key只允许一个线程查询数据和写缓存，其他线程等待。

2：做二级缓存，A1为原始缓存，A2为拷贝缓存，A1失效时，可以访问A2，A1缓存失效时间设置为短期，A2设置为长期

3：不同的key，设置不同的过期时间，让缓存失效的时间点尽量均匀。

011.redis的事务处理

1、MULTI用来组装一个事务

2、EXEC用来执行一个事务

3、DISCARD用来取消一个事物

4、WATCH用来监视一些key，一点这些key在事务执行之前被改变，则取消事务的执行

012.使用redis有哪些好处?

1、速度快，因为数据存在内存中，类似于HashMap,HashMap的优势就是查找和操作的时间复杂度都是O（1）

2、支持丰富数据类型，支持string，list，set，sortedset，hash

3、支持事务，操作都是原子性，所谓原子性就是对数据的更改要么全部执行，要么全部不执行

4、丰富的特性：可用于缓存，消息，按key设置过期时间，过期后将会自动删除

013.redis相比memcached有哪些优势?

1、memcached所有的值均是简单的字符串，redis作为其其替代者，支持更为丰富的数据类型

2、redis的速度比memcached快很多

3、redis可以持久化其数据

014.redis常见性能问题和解决方案：

1、Master最好不要做任何持久化工作，如RDB内存快照和AOF日志文件

2、如果数据比较重要，某个Slave开启AOF备份数据，策略设置为每秒同步一次

3、为了主从复制的速度和连接的稳定性，Master和Slave最好在同一个局域网内

4、尽量避免在压力河大的主库上增加从库

5、主从复制不要用图状结构，用单向链表结构更为稳定，这样的结构方便解决单点故障问题，

实现slave对master的替换。如果master挂了，可以立刻启动slave做master，其他不变

015.redis怎么和spring进行集成

1.引入jar包

2、配置bean 在application.xml加入如下配置 jedis配置 reids服务器中心 cache配置

不需要加入缓存的类 不需要缓存的方法设置缓存失效时间

016.redis内存数据集大小上升到一定大小的时候，就会执行数据淘汰策略。redis提供6中数据淘汰策略：

1、volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最新最少使用的数据淘汰

2、volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

3、volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

4、allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰

5、allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

6、no-enviction(驱逐)：禁止驱逐数据

017.redis常见的性能问题都有哪些？如何解决?

1、.Master写内存快照，save命令调度rdbSave函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，

会间断性暂停服务，所以Master最好不要写内存快照

2、.MasterAOF持久化，如果不重写AOF文件，这个持久化方式对性能的影响是最小的，但是AOF文件会不断增大，AOF

文件过大会影响Master重启的恢复速度。Master最好不要做任何持久化工作，包括内存快照和AOF日志文件，特别是

不要日用内存快照做持久化，如果数据比较关键，某个Slave开启AOF备份数据，策略为每秒同步一次

3、.Master调用BGREWRITEAOF重写AOF文件，AOF在重写的时候会占大量的CPU和内存资源，导致服务load过高，

出现短暂服务暂停现象。

4、.Reids主从复制的性能问题，为了主从复制的速度和连接的稳定性，Slave和Master最好在同一个局域网内

018.分布式缓存

硬盘上的数据，缓存在别的计算机（不是程序运行的计算机）的内存上

而且可以缓存的计算机的个数不止一个，可以使n个

用户通过访问http服务器，然后访问应用服务器资源，应用服务器调用后端的数据库，

在第一次访问的时候，直接访问数据库，然后将要缓存的内容放入到memcached集群，集群

规模根据缓存文件的大小而定。在第二次访问的时候就直接进入缓存读取，不需要进行

数据库的操作。这个适合数据变化不频繁的场景，比如：互联网站显示的榜单、阅读排行等、

019.redis和memcached的区别

如果简单地比较redis和memcached的区别，大多数都会得到以下观点：

1、redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，hash等数据结构的存储。

2、redis支持数据的备份，即master-slave模式的数据备份

3、redis支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用

在redis中，并不是所有的数据都一直存储在内存中的。这是和memcached相比一个最大的区别.

020.请解释一下什么是Nginx?

Nginx(enginex)是一个高性能的HTTP和反向代理服务，也是一个IMAP/POP3/SMTP服务。

Nginx是由伊戈尔·赛索耶夫为俄罗斯访问量第二的Rambler.ru站点（俄文：Рамблер）开发的，第一个公开版本0.1.0发布于2004年10月4日。

其将源代码以类BSD许可证的形式发布，因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。2011年6月1日，nginx1.0.4发布。

Nginx是一款轻量级的Web服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，并在一个BSD-like协议下发行。其特点是占有内存少，并发能力强，事实上nginx的并发能力确实在同类型的网页服务器中表现较好，中国大陆使用nginx网站用户有：百度、京东、新浪、网易、腾讯、淘宝等。

021.Nginx和apache的优缺点

n nginx相对于apache的优点：

Ø 轻量级，同样起web 服务，比apache 占用更少的内存及资源

Ø 抗并发，nginx 处理请求是异步非阻塞的，而apache 则是阻塞型的，在高并发下nginx

Ø 能保持低资源低消耗高性能

Ø 高度模块化的设计，编写模块相对简单

Ø 社区活跃，各种高性能模块出品迅速

n apache 相对于nginx 的优点：

Ø rewrite ，比nginx 的rewrite 强大

Ø 模块超多，基本想到的都可以找到

Ø 少bug ，nginx 的bug 相对较多

n Nginx 配置简洁, Apache 复杂

n 最核心的区别在于apache是同步多进程模型，一个连接对应一个进程；

nginx是异步的，多个连接（万级别）可以对应一个进程

023.请解释Nginx服务器上的Master和Worker进程分别是什么?

Master进程：读取及评估配置和维持

Worker进程：处理请求

```
[root@bruceliu001 ~]# ps -ef | grep nginx
root      7931     1  0 16:11 ?        00:00:00 nginx: master process /opt/myapp/soft/tengine-2.1.0/sb
nobody    8197  7931  0 17:44 ?        00:00:01 nginx: worker process
root      8250  8119  0 18:33 pts/1    00:00:00 grep nginx
```

024.请解释Nginx如何处理HTTP请求。

Nginx使用反应器模式。主事件循环等待操作系统发出准备事件的信号，这样数据就可以从套接字读取，在该实例中读取到缓冲区并进行处理。单个线程可以提供数万个并发连接。

025.Nginx反向代理为什么能够提升服务器性能?

对于后端是动态服务来说，比如Java和PHP。这类服务器(如JBoss和PHP-FPM)的IO处理能力往往不高。

Nginx有个好处是它会把Request在读取完整之前buffer住，这样交给后端的就是一个完整的HTTP请求，从而提高后端的效率，而不是断断续续的传递(互联网上连接速度一般比较慢)。同样，Nginx也可以把

response给buffer住，同样也是减轻后端的压力。

026.Nginx多进程模型是如何实现高并发的?

进程数与并发数不存在很直接的关系。这取决取server采用的工作方式。如果一个server采用一个进程负责一个request的方式，那么进程数就是并发数。那么显而易见的，就是会有很多进程在等待中。等什么？最多的应该是等待网络传输。

Nginx的异步非阻塞工作方式正是利用了这点等待的时间。在需要等待的时候，这些进程就空闲出来待命了。因此表现为少数几个进程就解决了大量的并发问题。apache是如何利用的呢，简单来说：同样的4个进程，如果采用一个进程负责一个request的方式，那么，同时进来4个request之后，每个进程就负责其中一个，直至会话关闭。期间，如果有第5个request进来了。就无法及时反应了，因为4个进程都没干完活呢，因此，一般有个调度进程，每当新进来了一个request，就新开个进程来处理。nginx不这样，每进来一个request，会有一个worker进程去处理。但不是全程的处理，处理到什么程度呢？处理到可能发生阻塞的地方，比如向上游（后端）服务器转发request，并等待请求返回。那么，这个处理的worker不会这么傻等着，他会在发送完请求后，注册一个事件：“如果upstream返回了，告诉我一声，我再接着干”。于是他就休息去了。此时，如果再有request进来，他就可以很快再按这种方式处理。而一旦上游服务器返回了，就会触发这个事件，worker才会来接手，这个request才会接着往下走。

由于webserver的工作性质决定了每个request的大部份生命都是在网络传输中，实际上花费在server机器上的时间片不多。这是几个进程就解决高并发的秘密所在。webserver刚好属于网络io密集型应用，不算是计算密集型。异步，非阻塞，使用epoll，和大量细节处的优化。也正是nginx之所以然的技术基石。

027. nginx负载均衡的4 种方式分配

nginx的upstream目前支持4种方式的分配

1)、轮询（默认）

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。

2)、weight 权重

指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。

2)、ip_hash

每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。

3)、fair（第三方）

按后端服务器的响应时间来分配请求，响应时间短的优先分配。

4)、url_hash（第三方）

nginx内置策略包含加权轮询和ip hash

加权轮询算法分为先深搜索和先广搜索，那么nginx采用的是先深搜索算法，即将首先将请求都分给高权重的机器，直到该机器的权值降到了比其他机器低，才开始将请求分给下一个高权重的机器；

028. Nginx优秀模块 模块设计：

高度模块化设计，除了少量核心代码，其他一切接模块。

官方Nginx共有五大类型模块：核心模块、配置模块、事件模块、HTTP模块、mail模块。

要注意的是：nginx的模块是静态的，添加和删除模块都要对nginx进行重新编译，这一点与Apache的动态模块完全不同。

029. 如何解决惊群现象?

惊群是多个子进程在同一时刻监听同一个端口引起的；

Nginx解决方法：同一个时刻只能有唯一一个worker子进程监听web端口，此时新连接事件只能唤醒唯一正在监听端口的worker子进程。

采用锁，互斥量实现！！

030.为什么要用Nginx?

优点：

跨平台、配置简单

非阻塞、高并发连接：处理2-3万并发连接数，官方监测能支持5万并发

内存消耗小：开启10个nginx才占150M内存，Nginx采取了分阶段资源分配技术

nginx处理静态文件好,耗费内存少

内置的健康检查功能：如果有一个服务器宕机，会做一个健康检查，再发送的请求就不会发送到宕机的服务器了。重新将请求提交到其他的节点上。

节省宽带：支持GZIP压缩，可以添加浏览器本地缓存

稳定性高：宕机的概率非常小

master/worker结构：一个master进程，生成一个或者多个worker进程

接收用户请求是异步的：浏览器将请求发送到nginx服务器，它先将用户请求全部接收下来，再一次性发送给后端web服务器，极大减轻了web服务器的压力

一边接收web服务器的返回数据，一边发送给浏览器客户端

网络依赖性比较低，只要ping通就可以负载均衡

可以有多台nginx服务器

事件驱动：通信机制采用epoll模型

031.为什么Nginx性能这么高?

得益于它的事件处理机制：

异步非阻塞事件处理机制：运用了epoll模型，提供了一个队列，排队解决

032.为什么不使用多线程?

Apache: 创建多个进程或线程，而每个进程或线程都会为其分配cpu和内存（线程要比进程小的多，所以worker支持比perfork高的并发），并发过大会榨干服务器资源。

Nginx: 采用单线程来异步非阻塞处理请求（管理员可以配置Nginx主进程的工作进程的数量）(epoll)，不会为每个请求分配cpu和内存资源，节省了大量资源，同时也减少了大量的CPU的上下文切换。所以才使得Nginx支持更高的并发。

033.Nginx是如何处理一个请求的呢?

首先，nginx在启动时，会解析配置文件，得到需要监听的端口与ip地址，然后在nginx的master进程里面

先初始化好这个监控的socket，再进行listen

然后再fork出多个子进程出来, 子进程会竞争accept新的连接。

此时，客户端就可以向nginx发起连接了。当客户端与nginx进行三次握手，与nginx建立好一个连接后此时，某一个子进程会accept成功，然后创建nginx对连接的封装，即ngx_connection_t结构体

接着，根据事件调用相应的事件处理模块，如http模块与客户端进行数据的交换。

最后，nginx或客户端来主动关掉连接，到此，一个连接就寿终正寝了

034.正向代理

一个位于客户端和原始服务器(origin server)之间的服务器，为了从原始服务器取得内容，客户端向代理发送一个请求并指定目标(原始服务器)，然后代理向原始服务器转交请求并将获得的内容返回给客户端。客户端才能使用正向代理

正向代理总结就一句话：代理端代理的是客户端

035.反向代理

反向代理（Reverse Proxy）方式是指以代理服务器来接受internet上的连接请求，然后将请求，发给内部网络上的服务器

并将从服务器上得到的结果返回给internet上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器

反向代理总结就一句话：代理端代理的是服务端

036.动态资源、静态资源分离

动态资源、静态资源分离是让动态网站里的动态网页根据一定规则把不变的资源和经常变的资源区分开来，动静资源做好了拆分以后，我们就可以根据静态资源的特点将其做缓存操作，这就是网站静态化处理的核心思路

动态资源、静态资源分离简单的概括是：动态文件与静态文件的分离

037.为什么要做动、静分离?

在我们的软件开发中，有些请求是需要后台处理的（如：.jsp,.do等等），有些请求是不需要经过后台处理的（如：css、html、jpg、js等等文件）

这些不需要经过后台处理的文件称为静态文件，否则动态文件。因此我们后台处理忽略静态文件。这会有人又说那我后台忽略静态文件不就完了吗

当然这是可以的，但是这样后台的请求次数就明显增多了。在我们对资源的响应速度有要求的时候，我们应该使用这种动静分离的策略去解决

动、静分离将网站静态资源（HTML，JavaScript，CSS，img等文件）与后台应用分开部署，提高用户访问静态代码的速度，降低对后台应用访问

这里我们将静态资源放到nginx中，动态资源转发到tomcat服务器中

038.负载均衡概念

负载均衡即是代理服务器将接收的请求均衡的分发到各服务器中

负载均衡主要解决网络拥塞问题，提高服务器响应速度，服务就近提供，达到更好的访问质量，减少后台服务器大并发压力

039.什么是 webService?

WebService 是一种跨编程语言和跨操作系统平台的远程调用技术。所谓跨编程语言和跨操作平台，就是说服务端程序采用 java 编写，客户端程序则可以采用其他编程语言编写，反之亦然！跨操作系统平台则是指服务端程序和客户端程序可以在不同的操作系统上。

040.webService相关术语

041.SOAP是什么?

SOAP是simple object access protocal的缩写，即简单对象访问协议。 是基于XML和HTTP的一种通信协议。是webservice所使用的一种传输协议，webservice之所以能够做到跨语言和跨平台，主要是因为XML和HTTP都是独立于语言和平台的。Soap的消息分为请求消息和响应消息，一条SOAP消息就是一个普通的XML文档，包含下列元素：

1、 必需的 Envelope 元素，可把此XML文档标识为一条SOAP消息

2、 可选的 Header 元素，包含头部信息

3、 必需的 Body 元素，包含所有的调用和响应信息

4、 可选的 Fault 元素，提供有关在处理此消息所发生错误的信息

Soap请求消息

Soap响应消息

042.WSDL文档主要有那几部分组成，分别有什么作用?

一个WSDL文档的根元素是definitions元素，WSDL文档包含7个重要的元素：types, import, message, portType, operations, binding和service元素。

1、 definitions元素中一般包括若干个XML命名空间；

2、 Types元素用作一个容器，定义了自定义的特殊数据类型，在声明消息部分（有效负载）的时候，messages定义使用了types元素中定义的数据类型与元素；

3、 Import元素可以让当前的文档使用其他WSDL文档中指定命名空间中的定义；

4、 Message元素描述了Web服务的有效负载。相当于函数调用中的参数和返回值；

5、 PortType元素定义了Web服务的抽象接口，它可以由一个或者多个operation元素，每个operation元素定义了一个RPC样式或者文档样式的Web服务方法；

6、 Operation元素要用一个或者多个messages消息来定义它的输入、输出以及错误；

7、 Binding元素将一个抽象的portType映射到一组具体的协议（SOAP或者HTTP）、消息传递样式（RPC或者document）以及编码样式（literal或者SOAP encoding）

8、 Service元素包含一个或者多个Port元素

每一个Port元素对应一个不同的Web服务，port将一个URL赋予一个特定的binding，通过location实现。

可以使两个或者多个port元素将不同的URL赋给相同的binding。

043.怎么理解UDDI?

UDDI是Universal Description Discovery and Integration的缩写，即统一描述、发现和整合规范。用来注册和查找服务，把web services收集和存储起来，这样当别人访问这些信息的时候就从UDDI中查找，看有没有这个信息存在。

044.Webservice的SEI指什么?

WebService EndPoint Interface（webservice终端[Server端]接口）

就是 WebService服务器端用来处理请求的接口

045.说说你知道的webservice框架，他们都有什么特点?

Webservice常用框架有JWS、Axis2、XFire以及CXF。

下面分别介绍一个这几种Web Service框架的基本概念

1、JWS是Java语言对WebService服务的一种实现，用来开发和发布服务。而从服务本身的角度来看JWS服务是没有语言界限的。但是Java语言为Java开发者提供便捷发布和调用WebService服务的一种途径。

2、Axis2是Apache下的一个重量级WebService框架，准确说它是一个Web Services / SOAP / WSDL 的引擎，是WebService框架的集大成者，它能不但能制作和发布WebService，而且可以生成Java和其他语言版WebService客户端和服务端代码。这是它的优势所在。但是，这也不可避免的导致了Axis2的复杂性，使用过的开发者都知道，它所依赖的包数量和大小都是很惊人的，打包部署发布都比较麻烦，不能很好的与现有应用整合为一体。但是如果你要开发Java之外别的语言客户端，Axis2提供的丰富工具将是你不二的选择。

3、XFire是一个高性能的WebService框架，在Java6之前，它的知名度甚至超过了Apache的Axis2，XFire的优点是开发方便，与现有的Web整合很好，可以融为一体，并且开发也很方便。但是对Java之外的语言，没有提供相关的代码工具。XFire后来被Apache收购了，原因是它太优秀了，收购后，随着Java6 JWS的兴起，开源的WebService引擎已经不再被看好，渐渐的都败落了。

4、CXF是Apache旗下一个重磅的SOA简易框架，它实现了ESB（企业服务总线）。CXF来自于XFire项目，经过改造后形成的，就像目前的Struts2来自WebWork一样。可以看出XFire的命运会和WebWork的命运一样，最终会淡出人们的视线。CXF不但是一个优秀的Web Services / SOAP / WSDL 引擎，也是一个不错的ESB总线，为SOA的实施提供了一种选择方案，当然他不是最好的，它仅仅实现了SOA架构的一部分。

注：对于Axis2与CXF之间的关系，一个是Axis2出现的时间较早，而CXF的追赶速度快。

如何抉择：

1、如果应用程序需要多语言的支持，Axis2应当是首选了；

2、如果应用程序是遵循 spring哲学路线的话，Apache CXF是一种更好的选择，特别对嵌入式的Web Services来说；

3、如果应用程序没有新的特性需要的话，就仍是用原来项目所用的框架，比如 Axis1，XFire，Celtrix或BEA等等厂家自己的Web Services实现，就别劳民伤财了。

046.mq的原理是什么?

消息队列技术是分布式应用间交换信息的一种技术。消息队列可驻留在内存或磁盘上,队列存储消息直到它们被应用程序读走。通过消息队列，应用程序可独立地执行--它们不需要知道彼此的位置、或在继续执行前不需要等待接收程序接收此消息。

在分布式计算环境中，为了集成分布式应用，开发者需要对异构网络环境下的分布式应用提供有效的通信手段。为了管理需要共享的信息，对应用提供公共的信息交换机制是重要的。

消息队列为构造以同步或异步方式实现的分布式应用提供了松耦合方法。消息队列的API调用被嵌入到新的或现存的应用中，通过消息发送到内存或基于磁盘的队列或从它读出而提供信息交换。消息队列可用在应用中以执行多种功能，比如要求服务、交换信息或异步处理等。

中间件是一种独立的系统软件或服务程序，分布式应用系统借助这种软件在不同的技术之间共享资源，管理计算资源和网络通讯。它在计算机系统中是一个关键软件，它能实现应用的互连和互操作性，能保证系统的安全、可靠、高效的运行。

中间件位于用户应用和操作系统及网络软件之间，它为应用提供了公用的通信手段，并且独立于网络和操作系统。

中间件为开发者提供了公用于所有环境的应用程序接口，当应用程序中嵌入其函数调用，它便可利用其运行的特定操作系统和网络环境的功能，为应用执行通信功能。

如果没有消息中间件完成信息交换，应用开发者为了传输数据，必须要学会如何用网络和操作系统软件的功能，编写相应的应用程序来发送和接收信息，且交换信息没有标准方法，每个应用必须进行特定的编程从而和多平台、不同环境下的一个或多个应用通信。

例如，为了实现网络上不同主机系统间的通信，将要求具备在网络上如何交换信息的知识（比如用TCP/IP的socket程序设计）；为了实现同一主机内不同进程之间的通讯，将要求具备操作系统的消息队列或命名管道(Pipes)等知识。

MQ的通讯模式

1) 点对点通讯：点对点方式是最为传统和常见的通讯方式，它支持一对一、一对多、多对多、多对一等多种配置方式，支持树状、网状等多种拓扑结构。

2) 多点广播：MQ适用于不同类型的应用。其中重要的，也是正在发展中的是"多点广播"应用，即能够将消息发送到多个目标站点(Destination List)。

可以使用一条MQ指令将单一消息发送到多个目标站点，并确保为每一站点可靠地提供信息。MQ不仅提供了多点广播的功能，而且还拥有智能消息分发功能，在将一条消息发送到同一系统上的多个用户时，MQ将消息的一个复制版本和该系统上接收者的名单发送到目标MQ系统。

目标MQ系统在本地复制这些消息，并将它们发送到名单上的队列，从而尽可能减少网络的传输量。

3) 发布/订阅(Publish/Subscribe)模式：发布/订阅功能使消息的分发可以突破目的队列地理指向的限制，使消息按照特定的主题甚至内容进行分发，用户或应用程序可以根据主题或内容接收到所需要的消息。

发布/订阅功能使得发送者和接收者之间的耦合关系变得更为松散，发送者不必关心接收者的目的地址，而接收者也不必关心消息的发送地址，而只是根据消息的主题进行消息的收发。

在MQ家族产品中，MQ Event Broker是专门用于使用发布/订阅技术进行数据通讯的产品，它支持基于队列和直接基于TCP/IP两种方式的发布和订阅。

4) 群集(Cluster)：为了简化点对点通讯模式中的系统配置，MQ提供Cluster(群集)的解决方案。群集类似于一个域(Domain)，群集内部的队列管理器之间通讯时，不需要两两之间建立消息通道，而是采用群集(Cluster)通道与其它成员通讯，从而大大简化了系统配置。

此外，群集中的队列管理器之间能够自动进行负载均衡，当某一队列管理器出现故障时，其它队列管理器可以接管它的工作，从而大大提高系统的高可靠性

047.mq的持久化是怎么做的?

以ActiveMq为例

ActiveMQ消息持久化方式，分别是：文件、mysql数据库、oracle数据库

a.文件持久化：

ActiveMQ默认的消息保存方式，一般如果没有修改过其他持久化方式的话可以不用修改配置文件。

如果是修改过的，打开盘符：\apache-activemq-版本号\conf\activemq.xml，然后找到节点，将其替换成以下代码段

然后修改配置文件（此处演示为spring+ActiveMQ），找到消息发送者所对应的JmsTemplate配置代码块，增加以下配置

以下是JmsTemplate配置完整版

这样就算配置完成了文件持久化方式了，重启项目和ActiveMQ，发送一定消息队列之后关闭ActiveMQ服务，再启动，你可以看到之前发送的消息未消费的依然保持在文件里面，继续让监听者消费。

b.MySQL持久化

首先需要把MySql的驱动放到ActiveMQ的Lib目录下，我用的文件名字是：mysql-connector-java-5.1.27.jar

然后打开盘符：\apache-activemq-版本号\conf\activemq.xml，然后找到节点，将其替换成以下代码段

在配置文件中的broker节点外增加以下代码

这样就算完成了mysql持久化配置了，验证方式同a，打开mysql数据库你能看到三张表，分别是：activemq_acks，activemq_lock，activemq_msgs。

c.Oracle持久化

oracle的配置和mysql一样，在Lib目录下，放入oracle的驱动包，然后配置一下配置文件即可。

048.zookeeper是什么?

ZooKeeper 顾名思义 动物园管理员，他是拿来管大象(Hadoop) 、 蜜蜂(Hive) 、 小猪(Pig) 的管理员，Apache Hbase和 Apache Solr 以及LinkedIn sensei 等项目中都采用到了 Zookeeper。

ZooKeeper是一个分布式的，开放源码的分布式应用程序协调服务，ZooKeeper是以Fast Paxos算法为基础，实现同步服务，配置维护和命名服务等分布式应用

049.zookeeper哪里用到?

Zookeeper是针对大型分布式系统的高可靠的协调系统。由这个定义我们知道zookeeper是个协调系统，作用的对象是分布式系统。为什么分布式系统需要一个协调系统了？理由如下：

开发分布式系统是件很困难的事情，其中的困难主要体现在分布式系统的“部分失败”。“部分失败”是指信息在网络的两个节点之间传送时候，如果网络出了故障，发送者无法知道接收者是否收到了这个信息，而且这种故障的原因很复杂，接收者可能在出现网络错误之前已经收到了信息，也可能没有收到，又或接收者的进程死掉了。

发送者能够获得真实情况的唯一办法就是重新连接到接收者，询问接收者错误的原因，这就是分布式系统开发里的“部分失败”问题。

Zookeeper就是解决分布式系统“部分失败”的框架。Zookeeper不是让分布式系统避免“部分失败”问题，而是让分布式系统当碰到部分失败时候，可以正确的处理此类的问题，让分布式系统能正常的运行。

下面我要讲讲zookeeper的实际运用场景：

场景一：有一组服务器向客户端提供某种服务（例如：我前面做的分布式网站的服务端，就是由四台服务器组成的集群，向前端集群提供服务），我们希望客户端每次请求服务端都可以找到服务端集群中某一台服务器，这样服务端就可以向客户端提供客户端所需的服务。

对于这种场景，我们的程序中一定有一份这组服务器的列表，每次客户端请求时候，都是从这份列表里读取这份服务器列表。那么这分列表显然不能存储在一台单节点的服务器上，否则这个节点挂掉了，整个集群都会发生故障，我们希望这份列表时高可用的。

高可用的解决方案是：这份列表是分布式存储的，它是由存储这份列表的服务器共同管理的，如果存储列表里的某台服务器坏掉了，其他服务器马上可以替代坏掉的服务器，并且可以把坏掉的服务器从列表里删除掉，让故障服务器退出整个集群的运行，而这一切的操作又不会由故障的服务器来操作，而是集群里正常的服务器来完成。

这是一种主动的分布式数据结构，能够在外部情况发生变化时候主动修改数据项状态的数据机构。Zookeeper框架提供了这种服务。这种服务名字就是：统一命名服务，它和javaEE里的JNDI服务很像。

场景二：分布式锁服务。当分布式系统操作数据，例如：读取数据、分析数据、最后修改数据。在分布式系统里这些操作可能会分散到集群里不同的节点上，那么这时候就存在数据操作过程中一致性的问题，如果不一致，我们将会得到一个错误的运算结果，在单一进程的程序里，一致性的问题很好解决，但是到了分布式系统就比较困难，

因为分布式系统里不同服务器的运算都是在独立的进程里，运算的中间结果和过程还要通过网络进行传递，那么想做到数据操作一致性要困难的多。Zookeeper提供了一个锁服务解决了这样的问题，能让我们在做分布式数据运算时候，保证数据操作的一致性。

场景三：配置管理。在分布式系统里，我们会把一个服务应用分别部署到n台服务器上，这些服务器的配置文件是相同的（例如：我设计的分布式网站框架里，服务端就有4台服务器，4台服务器上的程序都是一样，配置文件都是一样），

如果配置文件的配置选项发生变化，那么我们就得一个个去改这些配置文件，如果我们需要改的服务器比较少，这些操作还不是太麻烦，如果我们分布式的服务器特别多，比如某些大型互联网公司的hadoop集群有数千台服务器，那么更改配置选项就是一件麻烦而且危险的事情。

这时候zookeeper就可以派上用场了，我们可以把zookeeper当成一个高可用的配置存储器，把这样的事情交给zookeeper进行管理，我们将集群的配置文件拷贝到zookeeper的文件系统的某个节点上，然后用zookeeper监控所有分布式系统里配置文件的状态，

一旦发现有配置文件发生了变化，每台服务器都会收到zookeeper的通知，让每台服务器同步zookeeper里的配置文件，zookeeper服务也会保证同步操作原子性，确保每个服务器的配置文件都能被正确的更新。

场景四：为分布式系统提供故障修复的功能。集群管理是很困难的，在分布式系统里加入了zookeeper服务，能让我们很容易的对集群进行管理。

集群管理最麻烦的事情就是节点故障管理，zookeeper可以让集群选出一个健康的节点作为master，master节点会知道当前集群的每台服务器的运行状况，一旦某个节点发生故障，master会把这个情况通知给集群其他服务器，从而重新分配不同节点的计算任务。

Zookeeper不仅可以发现故障，也会对有故障的服务器进行甄别，看故障服务器是什么样的故障，如果该故障可以修复，zookeeper可以自动修复或者告诉系统管理员错误的原因让管理员迅速定位问题，修复节点的故障。

大家也许还会有个疑问，master故障了，那怎么办了？zookeeper也考虑到了这点，zookeeper内部有一个“选举领导者的算法”，master可以动态选择，当master故障时候，zookeeper能马上选出新的master对集群进行管理。

下面我要讲讲zookeeper的特点：

zookeeper是一个精简的文件系统。这点它和hadoop有点像，但是zookeeper这个文件系统是管理小文件的，而hadoop是管理超大文件的。

zookeeper提供了丰富的“构件”，这些构件可以实现很多协调数据结构和协议的操作。例如：分布式队列、分布式锁以及一组同级节点的“领导者选举”算法。

zookeeper是高可用的，它本身的稳定性是相当之好，分布式集群完全可以依赖zookeeper集群的管理，利用zookeeper避免分布式系统的单点故障的问题。

zookeeper采用了松耦合的交互模式。这点在zookeeper提供分布式锁上表现最为明显，zookeeper可以被用作一个约会机制，让参入的进程不在了解其他进程的（或网络）的情况下能够彼此发现并进行交互，参入的各方甚至不必同时存在，只要在zookeeper留下一条消息，在该进程结束后，另外一个进程还可以读取这条信息，从而解耦了各个节点之间的关系。

zookeeper为集群提供了一个共享存储库，集群可以从这里集中读写共享的信息，避免了每个节点的共享操作编程，减轻了分布式系统的开发难度。

zookeeper的设计采用的是观察者的设计模式，zookeeper主要是负责存储和管理大家关心的数据，然后接受观察者的注册，一旦这些数据的状态发生变化，Zookeeper 就将负责通知已经在 Zookeeper 上注册的那些观察者做出相应的反应，从而实现集群中类似 Master/Slave 管理模式

050.zookeeper的选主过程?

1 . 接收投票消息。投票消息会包括id,zxid,epoch,state，这四种信息，分别代表

Id: 唯一标识一台机器，存储在myid文件中

Zxid: 标识了本机想要选举谁为leader，是本机目前所见到的最大的id值

Epoch: 逻辑时钟。用于判断选举是否过期

State: 本机的状态信息(包括looking，leading，following，observing)

2 .判断PeerState状态，如果是looking状态，则继续.如果是leading,foolowing,observing则走别的流程

3 .收到票后，会判断发送过来的逻辑时钟是否大于目前的逻辑时钟,如果是说明集群已经进入了新一轮的投票了。

4 .清空投票箱。因为这个之前的投票都是上一次投票期间维护的。

5 . 如果等于目前的逻辑时钟，说明是当前的，则更新最大的leader id和提案id

判断是否需要更新当前自己的选举情况.在这里是根据选举leader id,保存的最大数据id来进行判断的,这两种数据之间对这个选举结果的影响的权重关系是:首先看数据id,数据id大者胜出;其次再判断leader id,leader id大者胜出

判读投票结果代码

6 . 发送通知，通知其他的QuorumPeer更新leader信息.同时将更新后的leader信息放入投票箱

检查是否已经接收到了所有服务器的投票代码参考。如果是的，则设置自己的选择结果

如果没有接收到所有服务器的投票，那判读这个leadId是否得到了一半以后的服务器的投票代码参考，如果是则返回

以上流程描述的是在zookeeper中，参考使用的算法是FastLeaderElection

在zookeeper的的选主的流程，另外还提供了LeaderElection和AuthFastLeaderElection的实现

LeaderElection的实现比较简单。以(id,zxid)做为投票的依据.并且它的实现是同步的，需要等待所有服务器返回后再统计结果。

而相比FastLeaderElection是每次收到回复都会计算投票结果，效率上会比LeaderElection更好一些

051.zookeeper集群之间如何通讯?

Zookeeper的通信架构

在Zookeeper整个系统中，有3中角色的服务，client、Follower、leader。其中client负责发起应用的请求，Follower接受client发起的请求，参与事务的确认过程，在leader crash后的leader选择。

而leader主要承担事务的协调，当然leader也可以承担接收客户请求的功能，为了方便描述，后面的描述都是client与Follower之间的通信，如果Zookeeper的配置支持leader接收client的请求，client与leader的通信跟client与Follower的通信模式完全一样。

Follower与leader之间的角色可能在某一时刻进行转换。一个Follower在leader crash掉以后可能被集群（Quorum）的Follower选举为leader。而一个leader在crash后，再次加入集群（Quorum）将作为Follower角色存在。

在一个集群（Quorum）中,除了在选举leader的过程中没有Follower和leader的区分外，其他任何时刻都只有1个leader和多个Follower。Client、Follower和leader之间的通信架构如下：

Client与Follower之间

为了使客户端具有较高的吞吐量，Client与Follower之间采用NIO的通信方式。当client需要与Zookeeper service打交道时，首先读取配置文件确定集群内的所有server列表，按照一定的load balance算法选取一个Follower作为一个通信目标。

这样client和Follower之间就有了一条由NIO模式构成的通信通道。这条通道会一直保持到client关闭session或者因为client或Follower任一方因某种原因异常中断通信连接。正常情况下, client与Follower在没有请求发起的时候都有心跳检测

Follower与leader之间

Follower与leader之间的通信主要是因为Follower接收到像（create, delete, setData, setACL, createSession, closeSession, sync）这样一些需要让leader来协调最终结果的命令，将会导致Follower与leader之间产生通信。

由于leader与Follower之间的关系式一对多的关系，非常适合client/server模式，因此他们之间是采用c/s模式，由leader创建一个socket server，监听各Follower的协调请求。

集群在选择leader过程中

由于在选择leader过程中没有leader，在集群中的任何一个成员都需要与其他所有成员进行通信，当集群的成员变得很大时，这个通信量是很大的。

选择leader的过程发生在Zookeeper系统刚刚启动或者是leader失去联系后，选择leader过程中将不能处理用户的请求，为了提高系统的可用性，一定要尽量减少这个过程的时间。选择哪种方式让他们可用快速得到选择结果呢?

Zookeeper在这个过程中采用了策略模式，可用动态插入选择leader的算法。系统默认提供了3种选择算法，AuthFastLeaderElection，FastLeaderElection，LeaderElection。

其中AuthFastLeaderElection和LeaderElection采用UDP模式进行通信，而FastLeaderElection仍然采用tcp/ip模式。在Zookeeper新的版本中，新增了一个learner角色，减少选择leader的参与人数。使得选择过程更快。

一般说来Zookeeper leader的选择过程都非常快，通常<200ms。

Zookeeper的通信流程

要详细了解Zookeeper的通信流程，我们首先得了解Zookeeper提供哪些客户端的接口，我们按照具有相同的通信流程的接口进行分组：

Zookeeper系统管理命令

Zookeeper的系统管理接口是指用来查看Zookeeper运行状态的一些命令，他们都是具有4字母构成的命令格式。主要包括：

ruok:发送此命令可以测试zookeeper是否运行正常。

dump：dump server端所有存活session的Ephemeral（临时）node信息。

stat：获取连接server的服务器端的状态及连接该server的所有客服端的状态信息。

reqs: 获取当前客户端已经提交但还未返回的请求。

stmk：开启或关闭Zookeeper的trace level.

gtmk：获取当前Zookeeper的trace level是否开启。

envi: 获取Zookeeper的java相关的环境变量。

srst：重置server端的统计状态

当用户发送这些命令的到server时，由于这些请求只与连接的server相关，没有业务处理逻辑，非常简单。Zookeeper对这些命令采用最快的效率进行处理。这些命令发送到server端只占用一个4字节的int类型来表示不同命令，没有采用字符串处理。当服务器端接收到这些命令，立刻返回结果。

Session创建

任何客户端的业务请求都是基于session存在的前提下。Session是维持client与Follower之间的一条通信通道，并维持他们之间从创建开始后的所有状态。

当启动一个Zookeeper client的时候，首先按照一定的算法查找出follower, 然后与Follower建立起NIO连接。当连接建立好后，发送create session的命令，让server端为该连接创建一个维护该连接状态的对象session。当server收到create session命令，先从本地的session列表中查找看是否已经存在有相同sessionId，

则关闭原session重新创建新的session。创建session的过程将需要发送到Leader，再由leader通知其他follower，大部分Follower都将此操作记录到本地日志再通知leader后，leader发送commit命令给所有Follower，

连接客户端的Follower返回创建成功的session响应。Leader与Follower之间的协调过程将在后面的做详细讲解。当客户端成功创建好session后，其他的业务命令就可以正常处理了。

Zookeeper查询命令

Zookeeper查询命令主要用来查询服务器端的数据，不会更改服务器端的数据。所有的查询命令都可以即刻从client连接的server立即返回，不需要leader进行协调，因此查询命令得到的数据有可能是过期数据。

但由于任何数据的修改，leader都会将更改的结果发布给所有的Follower，因此一般说来，Follower的数据是可以得到及时的更新。这些查询命令包括以下这些命令：

exists:判断指定path的node是否存在，如果存在则返回true，否则返回false.

getData:从指定path获取该node的数据

getACL:获取指定path的ACL。

getChildren:获取指定path的node的所有孩子结点。

所有的查询命令都可以指定watcher，通过它来跟踪指定path的数据变化。一旦指定的数据发生变化（create,delete,modified,children_changed），服务器将会发送命令来回调注册的watcher. Watcher详细的讲解将在Zookeeper的Watcher中单独讲解。

Zookeeper修改命令

Zookeeper修改命令主要是用来修改节点数据或结构，或者权限信息。任何修改命令都需要提交到leader进行协调，协调完成后才返回。修改命令主要包括：

1 . createSession：请求server创建一个session

2 . create：创建一个节点

3 . delete：删除一个节点

4 . setData：修改一个节点的数据

5 . setACL：修改一个节点的ACL

6 . closeSession：请求server关闭session

我们根据前面的通信图知道，任何修改命令都需要leader协调。 在leader的协调过程中，需要3次leader与Follower之间的来回请求响应。并且在此过程中还会涉及事务日志的记录，更糟糕的情况是还有take snapshot的操作。因此此过程可能比较耗时。但Zookeeper的通信中最大特点是异步的，如果请求是连续不断的，Zookeeper的处理是集中处理逻辑，然后批量发送，批量的大小也是有控制的。如果请求量不大，则即刻发送。这样当负载很大时也能保证很大的吞吐量，时效性也在一定程度上进行了保证。

zookeeper server端的业务处理-processor链

Zookeeper通过链式的processor来处理业务请求，每个processor负责处理特定的功能。不同的Zookeeper角色的服务器processor链是不一样的，以下分别介绍standalone Zookeeper server, leader和Follower不同的processor链。

Zookeeper中的processor

AckRequestProcessor:当leader从向Follower发送proposal后，Follower将发送一个Ack响应，leader收到Ack响应后，将会调用这个Processor进行处理。它主要负责检查请求是否已经达到了多数Follower的确认，如果满足条件，则提交commitProcessor进行commit处理

CommitProcessor：从commited队列中处理已经由leader协调好并commit的请求或者从请求队列中取出那些无需leader协调的请求进行下一步处理。

FinalRequestProcessor：任何请求的处理都需要经过这个processor，这是请求处理的最后一个Processor，主要负责根据不同的请求包装不同的类型的响应包。当然Follower与leader之间协调后的请求由于没有client连接，将不需要发送响应（代码体现在if (request.cnxn == null) {return;}）。

FollowerRequestProcessor：Follower processor链上的第一个，主要负责将修改请求和同步请求发往leader进行协调。

PrepRequestProcessor：在leader和standalone server上作为第一Processor，主要作用对于所有的修改命令生成changelog。

ProposalRequestProcessor：leader用来将请求包装为proposal向Follower请求确认。

SendAckRequestProcessor：Follower用来向leader发送Ack响应的处理。

SyncRequestProcessor：负责将已经commit的事务写到事务日志以及take snapshot.

ToBeAppliedRequestProcessor:负责将tobeApplied队列的中request转移到下一个请求进行处理。

052.zookeeper的节点加密是用的什么方式?

ZK的节点有5种操作权限：

CREATE、READ、WRITE、DELETE、ADMIN 也就是 增、删、改、查、管理权限，这5种权限简写为crwda(即：每个单词的首字符缩写)

注：这5种权限中，delete是指对子节点的删除权限，其它4种权限指对自身节点的操作权限

身份的认证有4种方式：

world：默认方式，相当于全世界都能访问

auth：代表已经认证通过的用户(cli中可以通过addauth digest user:pwd 来添加当前上下文中的授权用户)

digest：即用户名:密码这种方式认证，这也是业务系统中最常用的

ip：使用Ip地址认证

设置访问控制：

方式一：（推荐）

1）增加一个认证用户

addauth digest 用户名:密码明文

eg. addauth digest user1:password1

2）设置权限

setAcl /path auth:用户名:密码明文:权限

eg. setAcl /test auth:user1:password1:cdrwa

3）查看Acl设置

getAcl /path

方式二：

setAcl /path digest:用户名:密码密文:权限

注：这里的加密规则是SHA1加密，然后base64编码。

053.分布式锁的实现过程?

当很多进程需要访问共享资源时，我们可以通过zk来实现分布式锁。主要步骤是：

1 .建立一个节点，假如名为：lock 。节点类型为持久节点（PERSISTENT）

2 .每当进程需要访问共享资源时，会调用分布式锁的lock()或tryLock()方法获得锁，这个时候会在第一步创建的lock节点下建立相应的顺序子节点，节点类型为临时顺序节点（EPHEMERAL_SEQUENTIAL），通过组成特定的名字name+lock+顺序号。

3 .在建立子节点后，对lock下面的所有以name开头的子节点进行排序，判断刚刚建立的子节点顺序号是否是最小的节点，假如是最小节点，则获得该锁对资源进行访问。

4 .假如不是该节点，就获得该节点的上一顺序节点，并给该节点是否存在注册监听事件。同时在这里阻塞。等待监听事件的发生，获得锁控制权。

5 .当调用完共享资源后，调用unlock（）方法，关闭zk，进而可以引发监听事件，释放该锁。

实现的分布式锁是严格的按照顺序访问的并发锁

054. Dubbo支持哪些协议?

Dubbo支持dubbo、rmi、hessian、http、webservice、thrift、redis等多种协议，但是Dubbo官网是推荐我们使用Dubbo协议的。

dubbo协议

缺省协议，使用基于mina1.1.7+hessian3.2.1的tbremoting交互。

连接个数：单连接

连接方式：长连接

传输协议：TCP

传输方式：NIO异步传输

序列化：Hessian二进制序列化

适用范围：传入传出参数数据包较小（建议小于100K），消费者比提供者个数多，单一消费者无法压满提供者，尽量不要用dubbo协议传输大文件或超大字符串。

适用场景：常规远程服务方法调用

1、dubbo默认采用dubbo协议，dubbo协议采用单一长连接和NIO异步通讯，适合于小数据量大并发的服务调用，以及服务消费者机器数远大于服务提供者机器数的情况

2、他不适合传送大数据量的服务，比如传文件，传视频等，除非请求量很低。

3、Dubbo协议缺省每服务每提供者每消费者使用单一长连接，如果数据量较大，可以使用多个连接

4、为防止被大量连接撑挂，可在服务提供方限制大接收连接数，以实现服务提供方自我保护

rmi协议

Java标准的远程调用协议。

连接个数：多连接

连接方式：短连接

传输协议：TCP

传输方式：同步传输

序列化：Java标准二进制序列化

适用范围：传入传出参数数据包大小混合，消费者与提供者个数差不多，可传文件。

适用场景：常规远程服务方法调用，与原生RMI服务互操作

1、RMI协议采用JDK标准的java.rmi.*实现，采用阻塞式短连接和JDK标准序列化方式

hessian协议

基于Hessian的远程调用协议。

连接个数：多连接

连接方式：短连接

传输协议：HTTP

传输方式：同步传输

序列化：表单序列化

适用范围：传入传出参数数据包大小混合，提供者比消费者个数多，可用浏览器查看，可用表单或URL传入参数，暂不支持传文件。

适用场景：需同时给应用程序和浏览器JS使用的服务。

1、Hessian协议用于集成Hessian的服务，Hessian底层采用Http通讯，采用Servlet暴露服务，Dubbo缺省内嵌Jetty作为服务器实现。

2、Hessian是Caucho开源的一个RPC框架：http://hessian.caucho.com，其通讯效率高于WebService和Java自带的序列化。

http协议

基于http表单的远程调用协议。参见：[HTTP协议使用说明]

连接个数：多连接

连接方式：短连接

传输协议：HTTP

传输方式：同步传输

序列化：表单序列化

适用范围：传入传出参数数据包大小混合，提供者比消费者个数多，可用浏览器查看，可用表单或URL传入参数，暂不支持传文件。

适用场景：需同时给应用程序和浏览器JS使用的服务

webservice协议

基于WebService的远程调用协议。

连接个数：多连接

连接方式：短连接

传输协议：HTTP

传输方式：同步传输

序列化：SOAP文本序列化

适用场景：系统集成，跨语言调用

1、基于CXF的frontend-simple和transports-http实现。

2、CXF是Apache开源的一个RPC框架：http://cxf.apache.org，由Xfire和Celtix合并而来 。

055. Dubbo集群的切换?

正常情况下，进行系统设计时候，不仅要考虑正常逻辑下代码该如何走，还要考虑异常情况下代码逻辑应该怎么走。当服务消费方调用服务提供方的服务出现错误时候，Dubbo提供了多种容错方案，缺省模式为failover，也就是失败重试。

Failover Cluster：失败重试

当服务消费方调用服务提供者失败后自动切换到其他服务提供者服务器进行重试。这通常用于读操作或者具有幂等的写操作，需要注意的是重试会带来更长延迟。可通过 retries="2" 来设置重试次数（不含第一次）。

接口级别配置重试次数方法 <dubbo:reference retries="2" /> ，如上配置当服务消费方调用服务失败后，会再重试两次，也就是说最多会做三次调用，这里的配置对该接口的所有方法生效

Failfast Cluster：快速失败

当服务消费方调用服务提供者失败后，立即报错，也就是只调用一次。通常这种模式用于非幂等性的写操作。

Failsafe Cluster：失败安全

当服务消费者调用服务出现异常时，直接忽略异常。这种模式通常用于写入审计日志等操作。

Failback Cluster：失败自动恢复

当服务消费端用服务出现异常后，在后台记录失败的请求，并按照一定的策略后期再进行重试。这种模式通常用于消息通知操作。

Forking Cluster：并行调用

当消费方调用一个接口方法后，Dubbo Client会并行调用多个服务提供者的服务，只要一个成功即返回。这种模式通常用于实时性要求较高的读操作，但需要浪费更多服务资源。可通过 forks="2" 来设置最大并行数。

Broadcast Cluster：广播调用

当消费者调用一个接口方法后，Dubbo Client会逐个调用所有服务提供者，任意一台调用异常则这次调用就标志失败。这种模式通常用于通知所有提供者更新缓存或日志等本地资源信息。

如上，Dubbo本身提供了丰富的集群容错模式，但是如果您有定制化需求，可以根据Dubbo提供的扩展接口Cluster进行定制。在后面的消费方启动流程章节会讲解何时/如何使用的集群容错。

失败重试策略实现分析

Dubbo中具体实现失败重试的是FailoverClusterInvoker类

056.Dubbo的负载均衡策略

当服务提供方是集群的时候，为了避免大量请求一直落到一个或几个服务提供方机器上，从而使这些机器负载很高，甚至打死，需要做一定的负载均衡策略。Dubbo提供了多种均衡策略，缺省为random，也就是每次随机调用一台服务提供者的机器。

Dubbo提供的负载均衡策略

· Random LoadBalance：随机策略。按照概率设置权重，比较均匀，并且可以动态调节提供者的权重。

· RoundRobin LoadBalance：轮询策略。轮询，按公约后的权重设置轮询比率。会存在执行比较慢的服务提供者堆积请求的情况，比如一个机器执行的非常慢，但是机器没有挂调用（如果挂了，那么当前机器会从Zookeeper的服务列表删除），当很多新的请求到达该机器后，由于之前的请求还没有处理完毕，会导致新的请求被堆积，久而久之，所有消费者调用这台机器上的请求都被阻塞。

· LeastActive LoadBalance：最少活跃调用数。如果每个提供者的活跃数相同，则随机选择一个。在每个服务提供者里面维护者一个活跃数计数器，用来记录当前同时处理请求的个数，也就是并发处理任务的个数。所以如果这个值越小说明当前服务提供者处理的速度很快或者当前机器的负载比较低，所以路由选择时候就选择该活跃度最小的机器。如果一个服务提供者处理速度很慢，由于堆积，那么同时处理的请求就比较多，也就是活跃调用数目越大，这也使得慢的提供者收到更少请求，因为越慢的提供者的

活跃度越来越大。

· ConsistentHash LoadBalance：一致性Hash策略。一致性Hash，可以保证相同参数的请求总是发到同一提供者，当某一台提供者挂了时，原本发往该提供者的请求，基于虚拟节点，平摊到其他提供者，不会引起剧烈变动。

057.linux常用的命令有哪些?

关机 (系统的关机、重启以及登出)

shutdown -h now 关闭系统(1)

logout 注销

文件和目录

cd /home 进入 '/ home' 目录'

pwd 显示工作路径

ls 查看目录中的文件

mkdir dir1 创建一个叫做 'dir1' 的目录'

rm -f file1 删除一个叫做 'file1' 的文件'

mv dir1 new_dir 重命名/移动 一个目录

cp file1 file2 复制一个文件

文件搜索

find / -name file1 从 '/' 开始进入根文件系统搜索文件和目录

locate *.ps 寻找以 '.ps' 结尾的文件 - 先运行 'updatedb' 命令

whereis halt 显示一个二进制文件、源码或man的位置

which halt 显示一个二进制文件或可执行文件的完整路径

磁盘空间

df -h 显示已经挂载的分区列表

ls -lSr |more 以尺寸大小排列文件和目录

打包和压缩文件

tar -cvf archive.tar file1 创建一个非压缩的 tarball

zip file1.zip file1 创建一个zip格式的压缩包

unzip file1.zip 解压一个zip格式压缩包

058.如何获取java进程的pid?

pgrep -l java

059.如何获取某个进程的网络端口号?

netstat/lsof

netstat命令用于显示与IP、TCP、UDP和ICMP协议相关的统计数据，一般用于检验本机各端口的网络连接情况

-a 显示一个所有的有效连接信息列表(包括已建立的连接，也包括监听连接请求的那些连接)

-n 显示所有已建立的有效连接

-t tcp协议

-u udp协议

-l 查询正在监听的程序

-p 显示正在使用socket的程序识别码和程序名称

例如:netstat -ntupl|grep processname

如何只查询tomcat的连接?

netstat -na|grep ESTAB |grep 80 |wc-l

netstat -na|grep ESTAB |grep 8080 |wc-l

常用端口介绍:

端口：21

服务:FTP服务器所开放的端口，用于上传、下载。

端口: 22

服务:ssh

端口: 80

服务:HTTP 用于网页浏览

端口：389

服务：LDAP ILS 轻型目录访问协议和NetMeetingInternet Locator Server

端口：443

服务：网页浏览端口 能提供加密和通过安全端口传输的另一种HTTP

端口：8080

服务：代理端口

打开终端，执行如下命令，查看各进程占用端口情况：

ps -ef|wc -l //查看后台运行的进程总数

ps -fu csvn //查看csvn进程

netstat -lntp //查看开启了哪些端口

netstat -r //本选项可以显示关于路由表的信息

netstat -a //本选项显示一个所有的有效连接信息列表

netstat -an|grep 8080

netstat -na|grep -i listen //可以看到目前系统侦听的端口号

netstat -antup //查看已建立的连接进程，所占用的端口。

netstat -anp|grep1487

lsof -i:1487

查看哪些进程打开了指定端口1487

060.如何实时打印日志?

tail -f messages

061.如何统计某个字符串行数?

要统计一个字符串出现的次数，这里现提供自己常用两种方法：

\1. 使用vim统计

用vim打开目标文件，在命令模式下，输入

:%s/objStr//gn

即可

\2. 使用grep：

grep -o objStr filename|wc -l

如果是多个字符串出现次数，可使用：

grep -o 'objStr1|objStr2' filename|wc -l #直接用| 链接起来即可

062.大型网站在架构上应当考虑哪些问题?

答：

分层：分层是处理任何复杂系统最常见的手段之一，将系统横向切分成若干个层面，每个层面只承担单一的职责，然后通过下层为上层提供的基础设施和服务以及上层对下层的调用来形成一个完整的复杂的系统。

计算机网络的开放系统互联参考模型（OSI/RM）和Internet的TCP/IP模型都是分层结构，大型网站的软件系统也可以使用分层的理念将其分为持久层（提供数据存储和访问服务）、业务层（处理业务逻辑，系统中最核心的部分）和表示层（系统交互、视图展示）。

需要指出的是：（1）分层是逻辑上的划分，在物理上可以位于同一设备上也可以在不同的设备上部署不同的功能模块，这样可以使用更多的计算资源来应对用户的并发访问；（2）层与层之间应当有清晰的边界，这样分层才有意义，才更利于软件的开发和维护。

分割：分割是对软件的纵向切分。我们可以将大型网站的不同功能和服务分割开，形成高内聚低耦合的功能模块（单元）

。在设计初期可以做一个粗粒度的分割，将网站分割为若干个功能模块，后期还可以进一步对每个模块进行细粒度的分割，这样一方面有助于软件的开发和维护，另一方面有助于分布式的部署，提供网站的并发处理能力和功能的扩展。

分布式：除了上面提到的内容，网站的静态资源（JavaScript、CSS、图片等）也可以采用独立分布式部署并采用独立的域名，这样可以减轻应用服务器的负载压力，也使得浏览器对资源的加载更快。

数据的存取也应该是分布式的，传统的商业级关系型数据库产品基本上都支持分布式部署，而新生的NoSQL产品几乎都是分布式的。当然，网站后台的业务处理也要使用分布式技术，例如查询索引的构建、数据分析等，这些业务计算规模庞大，可以使用Hadoop以及MapReduce分布式计算框架来处理。

集群：集群使得有更多的服务器提供相同的服务，可以更好的提供对并发的支持。

缓存：所谓缓存就是用空间换取时间的技术，将数据尽可能放在距离计算最近的位置。使用缓存是网站优化的第一定律。我们通常说的CDN、反向代理、热点数据都是对缓存技术的使用。

异步：异步是实现软件实体之间解耦合的又一重要手段。异步架构是典型的生产者消费者模式，二者之间没有直接的调用关系，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，这对网站的扩展非常有利。

使用异步处理还可以提高系统可用性，加快网站的响应速度（用Ajax加载数据就是一种异步技术），同时还可以起到削峰作用（应对瞬时高并发）。"；能推迟处理的都要推迟处理"是网站优化的第二定律，而异步是践行网站优化第二定律的重要手段。

冗余：各种服务器都要提供相应的冗余服务器以便在某台或某些服务器宕机时还能保证网站可以正常工作，同时也提供了灾难恢复的可能性。冗余是网站高可用性的重要保证。

063.你使用过的应用服务器优化技术有哪些?

① 分布式缓存：缓存的本质就是内存中的哈希表，如果设计一个优质的哈希函数，那么理论上哈希表读写的渐近时间复杂度为O(1)。缓存主要用来存放那些读写比很高、变化很少的数据，这样应用程序读取数据时先到缓存中读取，如果没有或者数据已经失效再去访问数据库或文件系统，并根据拟定的规则将数据写入缓存。

对网站数据的访问也符合二八定律（Pareto分布，幂律分布），即80%的访问都集中在20%的数据上，如果能够将这20%的数据缓存起来，那么系统的性能将得到显著的改善。当然，使用缓存需要解决以下几个问题：

- 频繁修改的数据；

- 数据不一致与脏读；

- 缓存雪崩（可以采用分布式缓存服务器集群加以解决，memcached是广泛采用的解决方案）；

- 缓存预热；

- 缓存穿透（恶意持续请求不存在的数据）。

② 异步操作：可以使用消息队列将调用异步化，通过异步处理将短时间高并发产生的事件消息存储在消息队列中，从而起到削峰作用。电商网站在进行促销活动时，可以将用户的订单请求存入消息队列，这样可以抵御大量的并发订单请求对系统和数据库的冲击。目前，绝大多数的电商网站即便不进行促销活动，订单系统都采用了消息队列来处理。

③ 使用集群。

④ 代码优化：

- 多线程：基于Java的Web开发基本上都通过多线程的方式响应用户的并发请求，使用多线程技术在编程上要解决线程安全问题，主要可以考虑以下几个方面：

A. 将对象设计为无状态对象（这和面向对象的编程观点是矛盾的，在面向对象的世界中被视为不良设计），这样就不会存在并发访问时对象状态不一致的问题。

B. 在方法内部创建对象，这样对象由进入方法的线程创建，不会出现多个线程访问同一对象的问题。使用ThreadLocal将对象与线程绑定也是很好的做法，这一点在前面已经探讨过了。C. 对资源进行并发访问时应当使用合理的锁机制。

- 非阻塞I/O： 使用单线程和非阻塞I/O是目前公认的比多线程的方式更能充分发挥服务器性能的应用模式，基于Node.js构建的服务器就采用了这样的方式。Java在JDK 1.4中就引入了NIO（Non-blocking I/O）,在Servlet 3规范中又引入了异步Servlet的概念，这些都为在服务器端采用非阻塞I/O提供了必要的基础。

- 资源复用：资源复用主要有两种方式，一是单例，二是对象池，我们使用的数据库连接池、线程池都是对象池化技术，这是典型的用空间换取时间的策略，另一方面也实现对资源的复用，从而避免了不必要的创建和释放资源所带来的开销。

064.什么是XSS攻击？什么是SQL注入攻击？什么是CSRF攻击？

答：

XSS（Cross Site Script，跨站脚本攻击）是向网页中注入恶意脚本在用户浏览网页时在用户浏览器中执行恶意脚本的攻击方式。

跨站脚本攻击分有两种形式：反射型攻击（诱使用户点击一个嵌入恶意脚本的链接以达到攻击的目标，目前有很多攻击者利用论坛、微博发布含有恶意脚本的URL就属于这种方式）和持久型攻击（将恶意脚本提交到被攻击网站的数据库中，用户浏览网页时，恶意脚本从数据库中被加载到页面执行，

QQ邮箱的早期版本就曾经被利用作为持久型跨站脚本攻击的平台）。XSS虽然不是什么新鲜玩意，但是攻击的手法却不断翻新，防范XSS主要有两方面：消毒（对危险字符进行转义）和HttpOnly（防范XSS攻击者窃取Cookie数据）。

SQL注入攻击是注入攻击最常见的形式（此外还有OS注入攻击（Struts 2的高危漏洞就是通过OGNL实施OS注入攻击导致的）），当服务器使用请求参数构造SQL语句时，恶意的SQL被嵌入到SQL中交给数据库执行。

SQL注入攻击需要攻击者对数据库结构有所了解才能进行，攻击者想要获得表结构有多种方式：

（1）如果使用开源系统搭建网站，数据库结构也是公开的（目前有很多现成的系统可以直接搭建论坛，电商网站，虽然方便快捷但是风险是必须要认真评估的）；

（2）错误回显（如果将服务器的错误信息直接显示在页面上，攻击者可以通过非法参数引发页面错误从而通过错误信息了解数据库结构，Web应用应当设置友好的错误页，一方面符合最小惊讶原则，一方面屏蔽掉可能给系统带来危险的错误回显信息）；

（3）盲注。防范SQL注入攻击也可以采用消毒的方式，通过正则表达式对请求参数进行验证，此外，参数绑定也是很好的手段，这样恶意的SQL会被当做SQL的参数而不是命令被执行，JDBC中的PreparedStatement就是支持参数绑定的语句对象，从性能和安全性上都明显优于Statement。

CSRF攻击（Cross Site Request Forgery，跨站请求伪造）是攻击者通过跨站请求，以合法的用户身份进行非法操作（如转账或发帖等）。CSRF的原理是利用浏览器的Cookie或服务器的Session，盗取用户身份，其原理如下图所示。

防范CSRF的主要手段是识别请求者的身份，主要有以下几种方式：

（1）在表单中添加令牌（token）；

（2）验证码；

（3）检查请求头中的Referer（前面提到防图片盗链接也是用的这种方式）。

令牌和验证都具有一次消费性的特征，因此在原理上一致的，但是验证码是一种糟糕的用户体验，不是必要的情况下不要轻易使用验证码，目前很多网站的做法是如果在短时间内多次提交一个表单未获得成功后才要求提供验证码，这样会获得较好的用户体验

065.分布式环境怎么保存用户状态?

分布式Session的几种实现方式

1 .基于数据库的Session共享

2 .基于NFS共享文件系统

3 .基于memcached 的session，如何保证 memcached 本身的高可用性?

4 . 基于resin/tomcat web容器本身的session复制机制

5 . 基于TT/Redis 或 jbosscache 进行 session 共享。

6 . 基于cookie 进行session共享

066.NIO框架：dubbo的实现原理?

client一个线程调用远程接口，生成一个唯一的ID（比如一段随机字符串，UUID等），Dubbo是使用AtomicLong从0开始累计数字的

将打包的方法调用信息（如调用的接口名称，方法名称，参数值列表等），和处理结果的回调对象callback，全部封装在一起，组成一个对象object

向专门存放调用信息的全局ConcurrentHashMap里面put(ID, object)

将ID和打包的方法调用信息封装成一对象connRequest，使用IoSession.write(connRequest)异步发送出去

当前线程再使用callback的get()方法试图获取远程返回的结果，在get()内部，则使用synchronized获取回调对象callback的锁， 再先检测是否已经获取到结果，如果没有，然后调用callback的wait()方法，释放callback上的锁，让当前线程处于等待状态。

服务端接收到请求并处理后，将结果（此结果中包含了前面的ID，即回传）发送给客户端，客户端socket连接上专门监听消息的线程收到消息，分析结果，取到ID，再从前面的ConcurrentHashMap里面get(ID)，从而找到callback，将方法调用结果设置到callback对象里。

监听线程接着使用synchronized获取回调对象callback的锁（因为前面调用过wait()，那个线程已释放callback的锁了），再notifyAll()，唤醒前面处于等待状态的线程继续执行（callback的get()方法继续执行就能拿到调用结果了），至此，整个过程结束。

当前线程怎么让它“暂停”，等结果回来后，再向后执行?

答：先生成一个对象obj，在一个全局map里put(ID,obj)存放起来，再用synchronized获取obj锁，再调用obj.wait()让当前线程处于等待状态，然后另一消息监听线程等到服 务端结果来了后，再map.get(ID)找到obj，再用synchronized获取obj锁，再调用obj.notifyAll()唤醒前面处于等待状态的线程。

正如前面所说，Socket通信是一个全双工的方式，如果有多个线程同时进行远程方法调用，这时建立在client server之间的socket连接上会有很多双方发送的消息传递，前后顺序也可能是乱七八糟的，server处理完结果后，将结果消息发送给client，client收到很多消息，怎么知道哪个消息结果是原先哪个线程调用的?

答：使用一个ID，让其唯一，然后传递给服务端，再服务端又回传回来，这样就知道结果是原先哪个线程的了。

067.请解释下列 10 个 shell 命令的用途

top、ps、mv、find、df、cat、chmod、chgrp、grep、wc

top：该命令提供了实时对系统处理器状态的监控，它能够实时显示系统中各个进程的资源占用情况。该命令可以按照对 CPU、内存使用和执行时间对系统任务进程进行排序显示，同时还可以通过交互式命令进行设定显示。

ps：显示系统进程在瞬间的运行动态。 mv：文件/目录改名或变更存储位置。 find：在指定的路径下查找指定文件。

df：检查磁盘空间占用情况。

cat：将文件的内容打印到标准输出。

chmod：改变文件的权限。 chgrp：改变文件所属组。

grep：过滤文本，根据指定的字符串，对文件的每一行进行搜索，如找到，则输出该行内容。

wc：统计指定文件中的字节数、字数、行数，并将统计结果显示输出。

相关题目：写出 **15** 个以上你所知道的常用的 **Linux** 命令和它的功能。

ls：列出目录 cp：复制 rm：删除

cat：将文件的内容打印到标准输出 mkdir： 建立目录

tar：打包压缩 ps：查看进程 top：查看机器使用情况 df：检查磁盘空间占用情况

find：在指定路径下查找指定文件 grep：过滤文本

cd：改变当前工作目录 mount：挂载/卸载指定的文件系统

ifconfig：配置网络或显示当前网络接口状态 telnet:：远程登录

068.vi编辑器中，选中、复制、粘贴、删除的命令各是什么

选中：v（以字符为单位），V（以行为单位）复制：y，如 yy，nyy，y1G，yG，y0，y$

粘贴：p（粘贴在光标之后），P（粘贴在光标之前）删除：d，如 dd，ndd，d1G，dG，d$,d0

069.获取文件行数 (酷讯)

wc -l filename

070.输入文件的最后 5 行到另一个文件中

tail -n 5 file1 >> file2

071.查找文件中包含 hello 的行 (酷讯)

grep hello filename

072.查找当前目录下所有目录名为CVS的子目录的命令 (酷讯)

find ./CVS -maxdepth 1 -type d -print

073.如何让一个程序在后台运行并把输入定向到指定的文件 (酷讯)

可使用命令：nohup 程序 >> 文件名 2>&1 & （注意，2>&1 要放在输出文件名的后面）

074.如何把一个文件的内容添加到另一个文件的末尾 (酷讯)

cat file1 >> file2

075.如何实时的显示一个文件的输出 (酷讯)

使用 more 或者 less

076.定时执行一个程序的方法有什么 (酷讯)

可以使用 at 或者 crontab，其中 at 是处理仅执行一次就结束调度的命令，适用于突发性工作，而 crontab 将会循环一直进行下去，适用于例行性工作。

077.vi 编辑器中，如何替换指定的字符串 (酷讯) 使用 s 命令，

例如：

：n1,n2s/word1/word2/g，将第 n1 行与 n2 行之间的 word1 替换为 word2

：1,$s/word1/word2/g，从第一行到最后一行中的 word1 替换为 word2

078.当更新后，cvs中文件有冲突时。如何判断哪些你编辑的内容和更新下来的内容 (酷讯)

使用 CVS 时，如果出现冲突，双击冲突的文件（标识为红色），显示文件比较窗口。其中蓝色为需要传入的更新，其中灰色为需要传出的修改，其中红色为需要手工解决的冲突。

可以首先将蓝色需要传入的更新，然后手工解决红色部分，自行选择保留左边还是右边，并选择 mark as merge 将文件标志为合并，此时将以左边窗口中的内容为依据向服务器提示传出，即灰色和红色部分都会以左边窗口的内容为依据向服务器提交。

如果冲突文件打开无任何的颜色提示，可选择 update and override，使用服务器上的文件更新本地文件。

079.查看磁盘空间使用率的 Linux 命令是什么？查看有哪些系统进程正在运行命令是什么？检测 Linux 性能（cpu，磁盘 io，内存，网络等）都用到哪些命令？ （卓望）

查看磁盘空间使用率的命令：df

查看系统进程命令：top

检测 Linux 性能的命令：ps，free，vmstat，netstat

080.Linux 下终止一个进程用什么命令？打包压缩和解包用什么命令？软连接和硬链接有什么区别？建立软连接的命令是？ （卓望）

终止进程的命令：kill 打包压缩和解包命令：tar 建立软连接的命令：ln -s 源文件 目标链接名

硬链接是通过文件系统的 inode 来产生新文件名，而不是新文件，它只是在某个目录新建一条文件名链接到某 inode 号码的管理记录而已，而软链接是创建一个独立的文件，该文件会让数据的读取指向它链接的那个文件的文件名，类似于 windows 中的快捷方式。

081.说说 Linux 下的 find 命令和 grep 命令的区别

find：在磁盘/分区中找到文件，可以配 type 可以配 size time 等，通过文件名或文件大小或访问时间找到指定文件。

grep：查找文件里符合条件的字符串，并把匹配的行打印出来，可以使用正则表达式。

082.有 mail.log 的一个文档，内容为若干邮件地址，其中用'\n'将邮件地址分隔。要求从中挑选出 sina.com 的邮件地址（包括从文件读取、过滤到列印出来）。

mail.log 内容如下： james@sina.com.cn jack@163.com zhansan@sohu.com

lisi@hotmail.com

wangwu@gmail.com 参考答案：

cat mail.log | grep sina.com

或者：

#!/bin/bash

while read row ; do case $row in

sina.com) echo $row ;; esac done < mail.log

083.简述 Tcp 协议的三次握手过程。 （亿邮）

TCP 是主机对主机层的传输控制协议，提供可靠的连接服务，采用三次握手确认建立一个连接：

第一次握手：建立连接时，客户端发送 syn 包(syn=j)到服务器，并进入 SYN_SEND 状态，等待服务器确认；

第二次握手：服务器收到 syn 包，必须确认客户的 SYN（ack=j+1），同时自己也发送一个 SYN 包（syn=k），即 SYN+ACK 包，此时服务器进入 SYN_RECV 状态；

第三次握手：客户端收到服务器的 SYN＋ACK 包，向服务器发送确认包 ACK(ack=k+1)，此包发送完毕，客户端和服务器进入 ESTABLISHED 状态，完成三次握手。

完成三次握手，客户端与服务器开始传送数据。

084.高并发量网站解决方案

一个小型的网站，可以使用最简单的html静态页面就实现了，配合一些图片达到美化效果，所有的页面均存放在一个目录下，这样的网站对系统架构、性能的要求都很简单。随着互联网业务的不断丰富，网站相关的技术经过这些年的发展，已经细分到很细的方方面面，尤其对于大型网站来说，所采用的技术更是涉及面非常广，从硬件到软件、编程语言、数据库、WebServer、防火墙等各个领域都有了很高的要求，已经不是原来简单的html静态网站所能比拟的。

大型网站，比如门户网站，在面对大量用户访问、高并发请求方面，基本的解决方案集中在这样几个环节：使用高性能的服务器、高性能的数据库、高效率的编程语言、还有高性能的Web容器。这几个解决思路在一定程度上意味着更大的投入。

1、HTML静态化

其实大家都知道，效率最高、消耗最小的就是纯静态化的html页面，所以我们尽可能使我们的网站上的页面采用静态页面来实现，这个最简单的方法其实也是最有效的方法。但是对于大量内容并且频繁更新的网站，我们无法全部手动去挨个实现，于是出现了我们常见的信息发布系统CMS，像我们常访问的各个门户站点的新闻频道，甚至他们的其他频道，都是通过信息发布系统来管理和实现的，信息发布系统可以实现最简单的信息录入自动生成静态页面，还能具备频道管理、权限管理、自动抓取等功能，对于一个大型网站来说，拥有一套高效、可管理的CMS是必不可少的。

除了门户和信息发布类型的网站，对于交互性要求很高的社区类型网站来说，尽可能的静态化也是提高性能的必要手段，将社区内的帖子、文章进行实时的静态化、有更新的时候再重新静态化也是大量使用的策略，像Mop的大杂烩就是使用了这样的策略，网易社区等也是如此。

同时，html静态化也是某些缓存策略使用的手段，对于系统中频繁使用数据库查询但是内容更新很小的应用，可以考虑使用html静态化来实现。比如论坛中论坛的公用设置信息，这些信息目前的主流论坛都可以进行后台管理并且存储在数据库中，这些信息其实大量被前台程序调用，但是更新频率很小，可以考虑将这部分内容进行后台更新的时候进行静态化，这样避免了大量的数据库访问请求。

2、图片服务器分离

大家知道，对于Web服务器来说，不管是Apache、IIS还是其他容器，图片是最消耗资源的，于是我们有必要将图片与页面进行分离，这是基本上大型网站都会采用的策略，他们都有独立的、甚至很多台的图片服务器。这样的架构可以降低提供页面访问请求的服务器系统压力，并且可以保证系统不会因为图片问题而崩溃。

在应用服务器和图片服务器上，可以进行不同的配置优化，比如apache在配置ContentType的时候可以尽量少支持、尽可能少的LoadModule，保证更高的系统消耗和执行效率。

3、数据库集群、库表散列

大型网站都有复杂的应用，这些应用必须使用数据库，那么在面对大量访问的时候，数据库的瓶颈很快就能显现出来，这时一台数据库将很快无法满足应用，于是我们需要使用数据库集群或者库表散列。

在数据库集群方面，很多数据库都有自己的解决方案，Oracle、Sybase等都有很好的方案，常用的MySQL提供的Master/Slave也是类似的方案，您使用了什么样的DB，就参考相应的解决方案来实施即可。

上面提到的数据库集群由于在架构、成本、扩张性方面都会受到所采用DB类型的限制，于是我们需要从应用程序的角度来考虑改善系统架构，库表散列是常用并且最有效的解决方案。

我们在应用程序中安装业务和应用或者功能模块将数据库进行分离，不同的模块对应不同的数据库或者表，再按照一定的策略对某个页面或者功能进行更小的数据库散列，比如用户表，按照用户ID进行表散列，这样就能够低成本的提升系统的性能并且有很好的扩展性。

sohu的论坛就是采用了这样的架构，将论坛的用户、设置、帖子等信息进行数据库分离，然后对帖子、用户按照板块和ID进行散列数据库和表，最终可以在配置文件中进行简单的配置便能让系统随时增加一台低成本的数据库进来补充系统性能。

4、缓存

缓存一词搞技术的都接触过，很多地方用到缓存。网站架构和网站开发中的缓存也是非常重要。这里先讲述最基本的两种缓存。高级和分布式的缓存在后面讲述。

架构方面的缓存，对Apache比较熟悉的人都能知道Apache提供了自己的缓存模块，也可以使用外加的Squid模块进行缓存，这两种方式均可以有效的提高Apache的访问响应能力。

网站程序开发方面的缓存，Linux上提供的Memory Cache是常用的缓存接口，可以在web开发中使用，比如用Java开发的时候就可以调用MemoryCache对一些数据进行缓存和通讯共享，一些大型社区使用了这样的架构。另外，在使用web语言开发的时候，各种语言基本都有自己的缓存模块和方法，PHP有Pear的Cache模块，Java就更多了，.net不是很熟悉，相信也肯定有。

5、镜像

镜像是大型网站常采用的提高性能和数据安全性的方式，镜像的技术可以解决不同网络接入商和地域带来的用户访问速度差异，比如ChinaNet和EduNet之间的差异就促使了很多网站在教育网内搭建镜像站点，数据进行定时更新或者实时更新。在镜像的细节技术方面，这里不阐述太深，有很多专业的现成的解决架构和产品可选。也有廉价的通过软件实现的思路，比如Linux上的rsync等工具。

6、负载均衡

负载均衡将是大型网站解决高负荷访问和大量并发请求采用的高端解决办法。　　负载均衡技术发展了多年，有很多专业的服务提供商和产品可以选择，我个人接触过一些解决方法，其中有两个架构可以给大家做参考。

（1）、硬件四层交换

第四层交换使用第三层和第四层信息包的报头信息，根据应用区间识别业务流，将整个区间段的业务流分配到合适的应用服务器进行处理。

第四层交换功能就像是虚IP，指向物理服务器。它传输的业务服从的协议多种多样，有HTTP、FTP、NFS、Telnet或其他协议。这些业务在物理服务器基础上，需要复杂的载量平衡算法。在IP世界，业务类型由终端TCP或UDP端口地址来决定，在第四层交换中的应用区间则由源端和终端IP地址、TCP和UDP端口共同决定。

在硬件四层交换产品领域，有一些知名的产品可以选择，比如Alteon、F5等，这些产品很昂贵，但是物有所值，能够提供非常优秀的性能和很灵活的管理能力。“Yahoo中国”当初接近2000台服务器，只使用了三、四台Alteon就搞定了。

(2)、软件四层交换

大家知道了硬件四层交换机的原理后，基于OSI模型来实现的软件四层交换也就应运而生，这样的解决方案实现的原理一致，不过性能稍差。但是满足一定量的压力还是游刃有余的，有人说软件实现方式其实更灵活，处理能力完全看你配置的熟悉能力。

软件四层交换我们可以使用Linux上常用的LVS来解决，LVS就是Linux Virtual Server，他提供了基于心跳线heartbeat的实时灾难应对解决方案，提高系统的强壮性，同时可供了灵活的虚拟VIP配置和管理功能，可以同时满足多种应用需求，这对于分布式的系统来说必不可少。

一个典型的使用负载均衡的策略就是，在软件或者硬件四层交换的基础上搭建squid集群，这种思路在很多大型网站包括搜索引擎上被采用，这样的架构低成本、高性能还有很强的扩张性，随时往架构里面增减节点都非常容易。

对于大型网站来说，前面提到的每个方法可能都会被同时使用到，这里介绍得比较浅显，具体实现过程中很多细节还需要大家慢慢熟悉和体会。有时一个很小的squid参数或者apache参数设置，对于系统性能的影响就会很大。

7、最新：CDN加速技术

什么是CDN?

CDN的全称是内容分发网络。其目的是通过在现有的Internet中增加一层新的网络架构，将网站的内容发布到最接近用户的网络“边缘”，使用户可以就近取得所需的内容，提高用户访问网站的响应速度。

CDN有别于镜像，因为它比镜像更智能，或者可以做这样一个比喻：CDN=更智能的镜像+缓存+流量导流。因而，CDN可以明显提高Internet网络中信息流动的效率。从技术上全面解决由于网络带宽小、用户访问量大、网点分布不均等问题，提高用户访问网站的响应速度。

CDN的类型特点

CDN的实现分为三类：镜像、高速缓存、专线。

镜像站点（Mirror Site），是最常见的，它让内容直接发布，适用于静态和准动态的数据同步。但是购买和维护新服务器的费用较高，还必须在各个地区设置镜像服务器，配备专业技术人员进行管理与维护。对于大型网站来说，更新所用的带宽成本也大大提高了。

高速缓存，成本较低，适用于静态内容。Internet的统计表明，超过80%的用户经常访问的是20%的网站的内容，在这个规律下，缓存服务器可以处理大部分客户的静态请求，而原始的服务器只需处理约20%左右的非缓存请求和动态请求，于是大大加快了客户请求的响应时间，并降低了原始服务器的负载。

CDN服务一般会在全国范围内的关键节点上放置缓存服务器。

专线，让用户直接访问数据源，可以实现数据的动态同步。

CDN的实例

举个例子来说，当某用户访问网站时，网站会利用全球负载均衡技术，将用户的访问指向到距离用户最近的正常工作的缓存服务器上，直接响应用户的请求。

当用户访问已经使用了CDN服务的网站时，其解析过程与传统解析方式的最大区别就在于网站的授权域名服务器不是以传统的轮询方式来响应本地DNS的解析请求，而是充分考虑用户发起请求的地点和当时网络的情况，来决定把用户的请求定向到离用户最近同时负载相对较轻的节点缓存服务器上。

通过用户定位算法和服务器健康检测算法综合后的数据，可以将用户的请求就近定向到分布在网络“边缘”的缓存服务器上，保证用户的访问能得到更及时可靠的响应。

由于大量的用户访问都由分布在网络边缘的CDN节点缓存服务器直接响应了，这就不仅提高了用户的访问质量，同时有效地降低了源服务器的负载压力。

附：某CDN服务商的服务说明

采用GCDN加速方式

采用了GCDN加速方式以后，系统会在浏览用户和您的服务器之间增加一台GCDN服务器。浏览用户访问您的服务器时，一般静态数据，如图片、多媒体资料等数据将直接从GCDN服务器读取，使得从主服务器上读取静态数据的交换量大大减少。

为VIP型虚拟主机而特加的VPN高速压缩通道，使用高速压缩的电信<==>网通、电信<==>国际（HK）、网通<==>国际（HK）等跨网专线通道，智能多线，自动获取最快路径，极速的动态实时并发响应速度，实现了网站的动态脚本实时同步，对动态网站有一个更加明显的加速效果。

每个网络运营商（电信、网通、铁通、教育网）均有您服务器的GCDN服务器，无论浏览用户是来自何处，GCDN都能让您的服务器展现最快的速度！另外，我们将对您的数据进行实时备份，让您的数据更安全

085.代码优化

1、合理使用缓存使用

提高性能最好最快的办法当然是通过缓存来改善，对于任何一个web开发者都应该善用缓存。Asp.net下的缓存机制十分强大，用好缓存机制可以让我们极大的改善web应用的性能。

1.页面缓存

2.部分页面缓存

3.使用DataSource缓存

4.Cache对象

2、避免数据库频繁连接

1.能采用SQL或直接存储过程一次执行的尽量不要用代码多次执行

2、及时关闭数据库连接

3、适当采取配置文件文件存频繁使用文件

1.对不经常更改并且数据量小的可采用xml或者配置文件设置

4、资源文件上传大小验证

1.严格验证上传图片大小

2、严格控制上传Flash动画和视频大小

5、尽量避开访问高峰期，进行数据作业和数据服务

1.执行定时任务尽量避开访问高峰期

2、对应固定报表2可以采取预定格式，避开高峰自动提取

6、数据查询采用真分页

1.需要多少数据取多少数据

7、建设页面跳转覆盖

1.尽量采取弹框或切换选项方式展示数据，避免来回刷新列表重新获大量数据

8、取数据不要查询全部字段

1.查询数据尽量不要SELECT *

086.数据库优化

1、 查询出的数据量过大（可以采用多次查询，其他的方法降低数据量），尽量采取分页查询数据

2、锁或者死锁(这也是查询慢最常见的问题，是程序设计的缺陷)

3、返回了不必要的行和列

用OR的字句可以分解成多个查询，并且通过UNION链接多个查询。它们的速度只与是否使用索引有关，如果查询需要用到联合索引，用UNION all执行的效率更高。

4、如果是使用like进行查询的话，简单的使用index是不行的，但是全文索引，耗空间。 like 'a%' 使用索引 like '%a' 不使用索引用 like '%a%' 查询时，查询耗时和字段值总长度成正比,所以不能用CHAR类型，而是VARCHAR。对于字段的值很长的建全文索引。

5、 尽量将数据的处理工作放在服务器上，减少网络的开销，如使用存储过程。存储过程是编译、优化过，并且被组织到一个执行规划里，且存储在数据库中的SQL语句（存储过程是数据库服务器端的一段程序），是控制流语言的集合，速度当然快。

6、将需要查询的结果预先计算好放在表中，查询的时候再Select。这在SQL7.0以前是最重要的手段。例如计算商品购买小计计算。

7、没有必要时不要用DISTINCT和ORDER BY，这些动作可以改在客户端执行。它们增加了额外的开销。这同UNION和UNION ALL一样的道理。

8、一次更新多条记录比分多次更新每次一条快,就是说批处理好

9、用临时表，尽量用结果集和Table类性的变量来代替它,Table 类型的变量比临时表好

10、数据库设计：数据库内所有表结构均添加索引

调整原因：

近日数据库压力很大，经查有些大数据量表的查询速度很慢，导致数据库服务器CPU一直持续90%-100%，将这些表添加索引后，CPU很快变正常。

根据查询条件,建立索引,优化索引、优化访问方式，限制结果集的数据量。注意填充因子要适当（最好是使用默认值0）。索引应该尽量小，使用字节数小的列建索引好（参照索引的创建）,不要对有限的几个值的字段建单一索引如性别字段

11、 将大数据表做分库、分区处理:

具体操作如下:

1）、将大数据表与主数据库分离，单独新建一个数据库，然后将这些表做分区；

2）、将数据插入到消息队列内，后台利用windows计划任务执行（5分钟执行一次）C#控制台程序将消息队列内的数据批量（消息队列内有50000条记录，一次性插入到数据表内）插入到相应的数据表内；

调整原因:

例如：用户访问日志，每次用户访问一个页面的时候我们之前的操作是直接将数据插入数据库，这样做对数据库的访问及操作太大，严重影响其他数据插入、查询的效率，利用分库、分区、消息队列完成此操作的好处是用户访问页面的时候不直接对数据库操作，而是在消息队列内积累一定数量的数据后批量插入数据库，只执行一次数据库操作，而且因为数据库分离的原因，对其他的查询及插入不会有影响；

087.分布式优化

1、分布式架构-独立站点开发

模块化结构化开发，实现多资源分站点，数据分库，为后期实现分布式部署做准备，主要分为以下几部分:

web站点:

1.web前端站点

2.图片、文件资源站点

3.管理端站点 4.数据接口站点

数据库:

1.业务数据库

2.访问信息数据库、日志

前期访问量和数据量较小可采取单台或小数目台数服务器部署，后期大数据量采取多web站点多数据服务器方式进行部署。

2、分布式部署-CDN分发式网络

CDN的全称是Content Delivery Network，即内容分发网络。其目的是通过在现有的Internet中增加一层新的网络架构，将网站的内容发布到最接近用户的网络"边缘"，使用户可以就近取得所需的内容，解决Internet网络拥塞状况，提高用户访问网站的响应速度。从技术上全面解决由于网络带宽小、用户访问量大、网点分布不均等原因，解决用户访问网站的响应速度慢的根本原因。

该项为收费项目。

3、分布式部署-软负载均衡

采用nginx进行分流，nginx为轻量级的http服务与反向代理服务器软件，由于其并发能力较强，并且体积很小，所以被称为轻量级http服务软件。

nginx的特色功能有：

(1).URL rewrite：URL重写

(2).reverse proxy：反向代理

(3). 做缓存服务器

(4). 实现对web服务的负载均衡

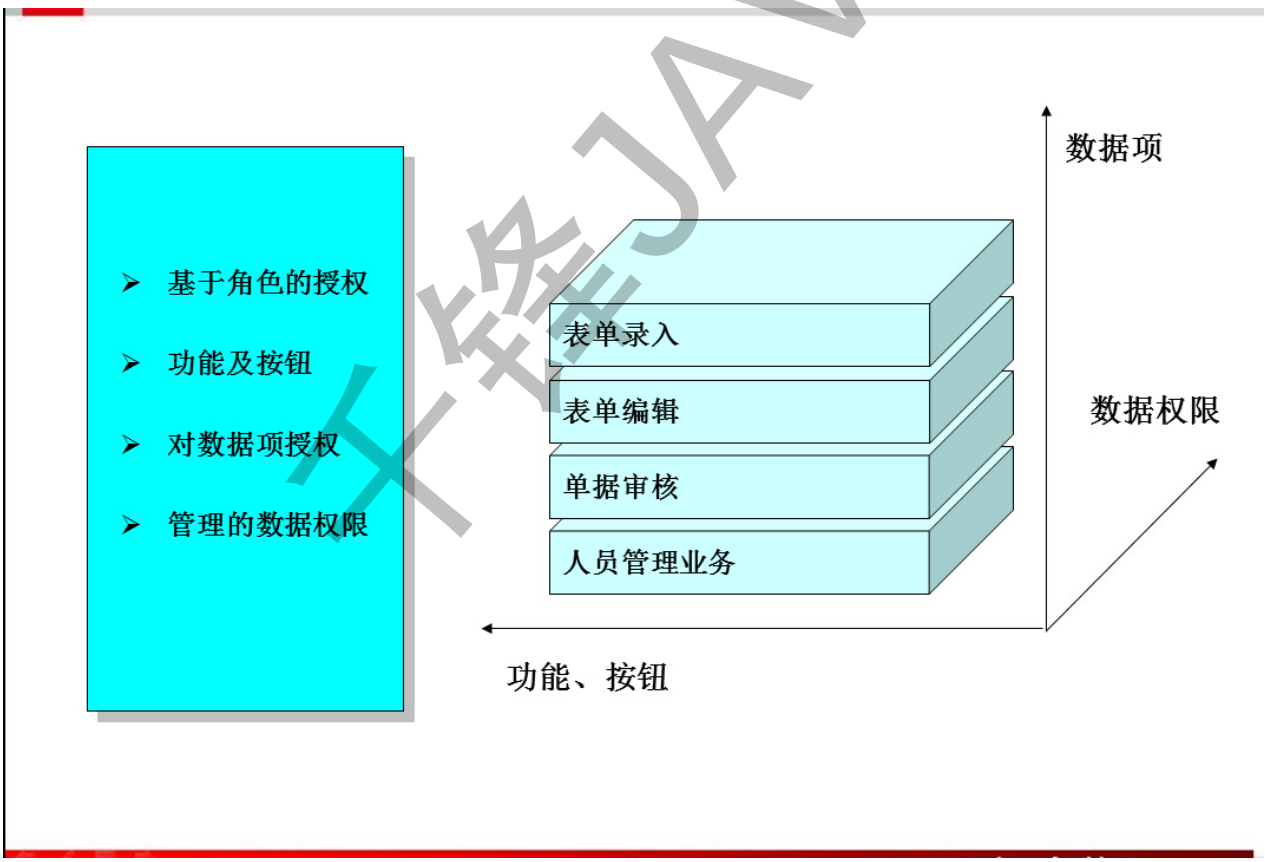
(5). 安装第三方插件，实现健康状态监测

(6).其他功能

088.安全优化

1、权限管理

从模块、表单、数据审核、功能按钮全面数据安全验证及管理。



2、ip验证

数据接口访问进行IP校验

3、登录、操作日志、程序安全日志

系统所有用户登录、操作全部日志记录。

程序安全日志操作可查看我之前写过[LogHelper 日志记录帮助类]。

4、SQL注入校验过滤

a、表单控件js前端校验，特殊字符过滤

b、采用Global.asax的Application_BeginRequest事件过滤敏感字符。

c、request请求过滤

特殊字符过滤可查看我之前写过[采用Global.asax的Application_BeginRequest事件过滤敏感字符]。

5、验证规范

a、前端js代码验证

b、后端程序代码验证

c、数据库约束

6、动态验证码

a、邮件动态验证码验证

b、短信动态验证码验证

7、验证码

登录、注册或相关表单采取输入验证码，避免恶意攻击