# 1.redis是什么？

[Redis](http://lib.csdn.net/base/redis) 是Remote Dictionary Server缩写，一个基于内存的高性能key-value[数据库](http://lib.csdn.net/base/mysql)。

# 2.redis解决什么问题？

缓存，高并发，分布式锁，发布订阅，数据存储。

# 3.redis的优点和缺点

## 3.1 优点

3.1.1 支持数据持久化，支持AOF和RDB两种持久化方式

3.1.2 支持主从复制，主机会自动将数据同步到从机，可以进行读写分离。

3.1.3 [数据结构](http://lib.csdn.net/base/datastructure)丰富：除了支持string类型的value外还支持string、hash、set、sortedset、list等数据结构。

## 3.2 缺点

3.2.1 [Redis](http://lib.csdn.net/base/redis)不具备自动容错和恢复功能，主机从机的宕机都会导致前端部分读写请求失败，需要等待机器重启或者手动切换前端的IP才能恢复。

3.2.2 主机宕机，宕机前有部分数据未能及时同步到从机，切换IP后还会引入数据不一致的问题，降低了系统的可用性。

3.2.3 [redis](http://lib.csdn.net/base/redis)的主从复制采用全量复制，复制过程中主机会fork出一个子进程对内存做一份快照，并将子进程的内存快照保存为文件发送给从机，这一过程需要确保主机有足够多的空余内存。若快照文件较大，对集群的服务能力会产生较大的影响，而且复制过程是在从机新加入集群或者从机和主机网络断开重连时都会进行，也就是网络波动都会造成主机和从机间的一次全量的数据复制，这对实际的系统运营造成了不小的麻烦。

3.2.4 Redis较难支持在线扩容，在集群容量达到上限时在线扩容会变得很复杂。为避免这一问题，运维人员在系统上线时必须确保有足够的空间，这对资源造成了很大的浪费。

## 3.3 其他

性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。

丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

原子 – Redis的所有操作都是原子性的，同时Redis还支持对几个操作全并后的原子性执行。

丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

# 4.Redis常用数据类型

常用的Redis数据结构：

String  # 键值对

List      # 列表

Hash    # 哈希

Set      # 集合

Sorted Set  # 有序集合

每种数据结构都有其合适的使用场景，但是谨记 不要手里握着锤子，看什么都是钉子 。另外当存储的数据量较大时，要注意每个操作命令的时间复杂度

## 4.1、String

常用命令: set,get,decr,incr,mget 等。

应用场景：String类型是最简单也是最常用的redis数据结构，key/value格式完全可以取代Memcached作为缓存服务器，单机的测试效果显示redis的效果更好。

set、get：最简单的数据缓存

mset、mget： 批量操作，把数据统一传回客户端，节省网络io时间

decr、incr：计数器

append命令：可以作为时间序列，配合getrange、setrange，对字符串进行操作，目前redis还木有修剪操作

setbit、getbit： 省内存的好命令，可以作为简单的布尔过滤器来判断用户是否执行过某些操作

## 4.2、List

常用命令：lpush,rpush,lpop,rpop,lrange等。

应用场景：List的应用场景很多，应用也相当广泛

lpush、lpop：天然的 队列操作 ，轻松实现队列任务，Celery的存储容器我们选的就是redis

lpush、ltrim： 显示最新 的数据，很好用的！比如：游戏上方的跑马灯，就可以用这两个命令来存储最新的50条记录

还有一些其他操作：堵塞式的blpop， lrange（O(n)）， lindex（O(n)），linsert（O(n)）， llen（O(1)），lrem(O(n))，lset(O(n))

## 4.3、Hash

常用命令：hget,hset,hgetall 等。

应用场景：以前在memcached中如果保存一个大的数据，经常用序列化之后保存，取出来反序列化后使用，即不经济实惠，在高并发下还存在原子性问题，在redis中， 用哈希实现，so easy啦！

hget、hset： 实现一个key对应一个数据集集合 ，数据集集合里包含多个单独的key/value键值对，操作依然是原子性的

hmget、hmset、hgetall： 批量操作，节省网络io时间哦

hincrby: 对哈希里域值，进行原子性的加1

其他操作： hdel（O(n)）、 hkeys（O(n)）、hexits（O(1)）、hvals（O(n)）、hscan（O(n)）

## 4.4、Set

常用命令： sadd,spop,smembers,sunion 等。

应用场景：set与list类似，只是set是经过去重的集合，需要一个不重复的数据结构，就要考虑考虑set

sadd： 存储一个 不重复数据 的数据集合

sunion、sdiff、sinter： 进行 集合处理 ，例如微博中，将一个用户关注的所有人放入set集合中，通过并集、交集、差集操作，实现`共同关注`、`共同喜好`、`二度好友`等功能

其他操作：srem、spop、scard、sismember、smove、srandmember

## 4.5、Sorted Set

常用命令： zadd,zrange,zrem,zcard等

应用场景：set是无序的，而Sorted set 顾名思义，它是有序的，由key、member和score组成，需要一个有序而且不重复的数据结构，就要考虑考虑sorted set

zadd：存储一个按照score排序的数据集合，添加时自动排序，例如： 优先队列 ，普通消息的score为1，重要消息的score为2，然后工作线程可以选择按score的倒序来获取工作任务。让重要的任务优先执行。

zrange、zrangebyscore等等： 按照score顺序 获取数据集，例如：微博的时间流信息，把发布时间作为score。还可以用来 处理过期数据。 后台任务使用ZRANGE…SCORES查询排序集合，取出最新的10个项目。如果发现unix时间已经过期，则在数据库中删除条目。

zrank： 排行榜 功能，score作为投票结果

其他操作：zcard、zcount、zincrby、zrem、zscore，以及set的集合操作

先更到这里，接下来记录这次项目中的部署以及业务中用到的redis经验

# 5.redis 内存划分

## 5.1数据，键值对存储包括5表中数据类型

存储：redisEntity 分为key，value。key 指向具体类型对应的存储指针。比如 string 默认存储在SDS存储区。

value 在redisEntity中存储为redisObject，redisObject包含 值类型，最后一次访问事件，值存储区指针。

## 5.2进程本身运行需要的内存

Redis主进程本身运行占用内存，如代码、常量池等，大约几兆。

## 5.3缓冲内存

缓冲内存包括客户端缓冲区、复制积压缓冲区、AOF缓冲区等;其中，客户端缓存存储客户端连接的输入输出。

## 5.4内存碎片

内存碎片是redis在分配、回收物理内存过程稿中产生。

# 6.redis内部运作机制

## 6.1redis服务器状态 由“redisserver”的数据结构来保存。

里面记录了有多少个数据库，所有客户端状态，当前时间戳，执行命令数，数组索引，已使用内存峰值，关闭服务器标识，是否在执行持久化，慢查询日志ID，监视客户端列表，订阅关系表等

## 6.2redis数据库

每一个redisDb对应一个redis数据库，

redisDB包含数据库键空间字典，过期字典，字典正在被监视的键

在读取一个键的时候，更新命中次数和未命中次数

在读取一个键的时候，服务器会更新键的LRU属性值（最后一次使用时间，使用‘object idletime’命令可以查看键的闲置时间）

在读取一个键时，发现该键已过去，删除这个键

如果有客户端命令监视了某个键后，服务器在改键被修改后，标记这个键为脏，从而让事务程序注意到这个键被修改了。

服务器每次修改一个键后，都会对脏键计数器值加一，这个计数器会触发服务器的持久化及复制操作。

如果服务器开启了数据库通知功能，在对键进行修改后，服务器将按配置发送相应的数据库通知

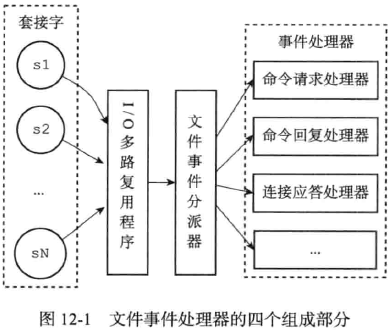
## 6.3redis客户端

redisserver中clients是一个链表，保存所有链接到当前服务器的客户端状态，每个客户端状态使用类型为“redisclient”的数据结构。

redisclient中记录客户端当前使用的数据库，客户端正在使用的套接字符串，客户端名字，当前执行命令及参数，是否通过验证，事务状态，包含一个事务队列

## 6.4redis事件

redis服务器是一个事件驱动程序，包含 文件事件和时间事件



## 6.5redis服务器初始化步骤

初始化数据服务器状态结构 redisServer

加载用户配置

初始化数据结构，主要是为服务器状态中的一些数据结构分配内存

还原数据库状态 若启用了AOF持久化功能 则载入AOF文件，否则载入RDB。根据文件还原数据库。

执行事件循环 准备接收处理客户端指令

## 6.6redis命令

发送命令请求

读取命令请求

命令执行器-查找命令

命令执行器-执行预备操作

命令执行器-调用命令的实现函数

命令执行器-执行后续工作

将命令回复发送给客户端

客户端接受并打印命令回复

# 7.redis应用场景

## 7.1 缓存

## 7.2 分布式锁

分布式锁的目的是为了在多服务器多服务运行的情况下，保证有且只有一台服务器上的一个服务能够访问受限制的资源。

比如在秒杀场景中，减库存操作。一个商品在库存中有100个，小明买了2个。正确的方式应该是100-2=98。但实际情况可能是，在-2库存的时候，有很多用户同时买了这个商品。由于网络原因，有的用户减库存的操作快。当小明查询剩余库存100个的时候，还没有进行-2，其他用户就已经减少了库存。在这种情况下，再用100-2明显库存是错误的。

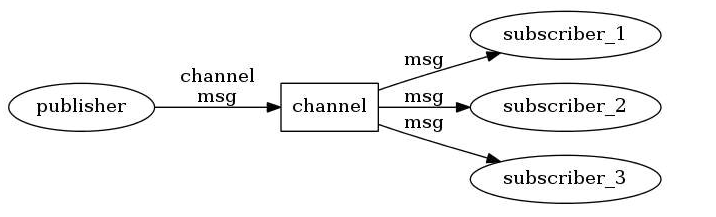
为了解决这种问题，我们需要在查询到当前库存前，先将库存锁住，即其他用户暂时不能减库存，小明减-2库存后，释放锁，其他用户才能进行减库存操作。

要达到以上目的，我们要确保分布式锁具备4个条件 ：

1. 互斥性。在任意时刻，只有一个客户端能持有锁。
2. 不会发生死锁。当有一个客户端在持有锁的时候崩溃了而没有及时释放锁，也要保证后续其他的客户端能够获得锁。
3. 具有容错性。即分布式锁加锁的服务器具有高可用，一台服务器挂掉后不影响正常使用。
4. 解锁和解锁必须是同一个客户端，客户端不能把别人加的锁解了。

## 7.3 pub/sub

Redis提供了发布订阅功能，可以用于消息的传输，Redis的发布订阅机制包括三个部分，发布者，订阅者和Channel。

  
发布者和订阅者都是Redis客户端，Channel则为Redis服务器端，发布者将消息发送到某个的频道，订阅了这个频道的订阅者就能接收到这条消息。Redis的这种发布订阅机制与基于主题的发布订阅类似，Channel相当于主题。

## 7.4 事务

### ***7.4.1、基本概念***

**1）什么是redis的事务？**

简单理解，可以认为redis事务是一些列redis命令的集合，并且有如下两个特点：

a）事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。

b）事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。

**2）事务的性质ACID**

一般来说，事务有四个性质称为ACID，分别是原子性，一致性，隔离性和持久性。

a）原子性atomicity：redis事务保证事务中的命令要么全部执行要不全部不执行。有些文章认为redis事务对于**执行错误**不回滚违背了原子性，是偏颇的。

b）一致性consistency：redis事务可以保证命令失败的情况下得以回滚，数据能恢复到没有执行之前的样子，是保证一致性的，除非redis进程意外终结。

c）隔离性Isolation：redis事务是严格遵守隔离性的，原因是redis是单进程单线程模式，可以保证命令执行过程中不会被其他客户端命令打断。

d）持久性Durability：redis事务是不保证持久性的，这是因为redis持久化策略中不管是RDB还是AOF都是异步执行的，不保证持久性是出于对性能的考虑。

**3）redis事务的错误**

使用事务时可能会遇上以下两种错误：

a）入队错误：事务在执行 [EXEC](http://doc.redisfans.com/transaction/exec.html#exec) 之前，入队的命令可能会出错。比如说，命令可能会产生语法错误（参数数量错误，参数名错误，等等），或者其他更严重的错误，比如内存不足（如果服务器使用 maxmemory 设置了最大内存限制的话）。

b）执行错误：命令可能在 [EXEC](http://doc.redisfans.com/transaction/exec.html#exec) 调用之后失败。举个例子，事务中的命令可能处理了错误类型的键，比如将列表命令用在了字符串键上面，诸如此类。

注：第三种错误，redis进程终结，本文并没有讨论这种错误。

### ***7.4.2、redis事务的用法***

redis事务是通过[MULTI](http://doc.redisfans.com/transaction/multi.html#multi)，[EXEC](http://doc.redisfans.com/transaction/exec.html#exec)，[DISCARD](http://doc.redisfans.com/transaction/discard.html#discard)和WATCH四个原语实现的。

[MULTI](http://doc.redisfans.com/transaction/multi.html#multi)命令用于开启一个事务，它总是返回OK。

[MULTI](http://doc.redisfans.com/transaction/multi.html#multi)执行之后，客户端可以继续向服务器发送任意多条命令，这些命令不会立即被执行，而是被放到一个队列中，当[EXEC](http://doc.redisfans.com/transaction/exec.html#exec)命令被调用时，所有队列中的命令才会被执行。

另一方面，通过调用[DISCARD](http://doc.redisfans.com/transaction/discard.html#discard)，客户端可以清空事务队列，并放弃执行事务。

下面给出几种事务场景。

**1）正常执行**

[复制代码](javascript:void(0);)

127.0.0.1:6379> MULTI

OK

127.0.0.1:6379> SET key1 1

QUEUED

127.0.0.1:6379> HSET key2 field1 1

QUEUED

127.0.0.1:6379> SADD key3 1

QUEUED

127.0.0.1:6379> EXEC

1) OK

2) (integer) 1

3) (integer) 1

[复制代码](javascript:void(0);)

[EXEC](http://doc.redisfans.com/transaction/exec.html#exec) 命令的回复是一个数组，数组中的每个元素都是执行事务中的命令所产生的回复。 其中，回复元素的先后顺序和命令发送的先后顺序一致。

当客户端处于事务状态时，所有传入的命令都会返回一个内容为 QUEUED 的状态回复（status reply），这些被入队的命令将在 [EXEC](http://doc.redisfans.com/transaction/exec.html#exec)命令被调用时执行。

**2）放弃事务**

当执行 [DISCARD](http://doc.redisfans.com/transaction/discard.html#discard) 命令时，事务会被放弃，事务队列会被清空，并且客户端会从事务状态中退出：

[复制代码](javascript:void(0);)

127.0.0.1:6379> MULTI

OK

127.0.0.1:6379> SET key1 1

QUEUED

127.0.0.1:6379> DISCARD

OK

127.0.0.1:6379> EXEC

(error) ERR EXEC without MULTI

[复制代码](javascript:void(0);)

**3）入队错误回滚**

[复制代码](javascript:void(0);)

127.0.0.1:6379> MULTI

OK

127.0.0.1:6379> set key1 1

QUEUED

127.0.0.1:6379> HSET key2 1

(error) ERR wrong number of arguments for 'hset' command

127.0.0.1:6379> SADD key3 1

QUEUED

127.0.0.1:6379> EXEC

(error) EXECABORT Transaction discarded because of previous errors.

[复制代码](javascript:void(0);)

对于入队错误，redis 2.6.5版本后，会记录这种错误，并且在执行EXEC的时候，报错并回滚事务中所有的命令，并且终止事务。

**3）执行错误放过**

[复制代码](javascript:void(0);)

127.0.0.1:6379> MULTI

OK

127.0.0.1:6379> HSET key1 field1 1

QUEUED

127.0.0.1:6379> HSET key2 field1 1

QUEUED

127.0.0.1:6379> EXEC

1) (error) WRONGTYPE Operation against a key holding the wrong kind of value

2) (integer) 1

[复制代码](javascript:void(0);)

当遇到执行错误时，redis放过这种错误，保证事务执行完成。

这里要注意此问题，与mysql中事务不同，在redis事务遇到执行错误的时候，不会进行回滚，而是简单的放过了，并保证其他的命令正常执行。这个区别在实现业务的时候，需要自己保证逻辑符合预期。

### ***7.4.3、使用WATCH***

[WATCH](http://doc.redisfans.com/transaction/watch.html#watch) 命令可以为 Redis 事务提供 check-and-set （CAS）行为。

被 [WATCH](http://doc.redisfans.com/transaction/watch.html#watch) 的键会被监视，并会发觉这些键是否被改动过了。 如果有至少一个被监视的键在 [EXEC](http://doc.redisfans.com/transaction/exec.html#exec) 执行之前被修改了， 那么整个事务都会被取消， [EXEC](http://doc.redisfans.com/transaction/exec.html#exec) 返回空多条批量回复（null multi-bulk reply）来表示事务已经失败。

[复制代码](javascript:void(0);)

127.0.0.1:6379> WATCH key1

OK

127.0.0.1:6379> set key1 2

OK

127.0.0.1:6379> MULTI

OK

127.0.0.1:6379> set key1 3

QUEUED

127.0.0.1:6379> set key2 3

QUEUED

127.0.0.1:6379> EXEC

(nil)

[复制代码](javascript:void(0);)

使用上面的代码， 如果在 [WATCH](http://doc.redisfans.com/transaction/watch.html#watch) 执行之后， [EXEC](http://doc.redisfans.com/transaction/exec.html#exec) 执行之前， 有其他客户端修改了 key1 的值， 那么当前客户端的事务就会失败。 程序需要做的， 就是不断重试这个操作， 直到没有发生碰撞为止。

这种形式的锁被称作乐观锁， 它是一种非常强大的锁机制。 并且因为大多数情况下， 不同的客户端会访问不同的键， 碰撞的情况一般都很少， 所以通常并不需要进行重试。

## 7.5 nosql数据库设计

1. 自增主键设计

incr key

每次运行都将得到一个唯一的id，如此我们就完成了自增主键的设计。

1. 字段设计

Nosql数据库字段设计灵活即用即存，不需要向关系型数据库一样先定义后使用。

1. 查询索引设计

# 8.安装部署

## 8.1.安装

官网 <http://redis.io>

1. 下载

wget http://download.redis.io/releases/redis-4.0.2.tar.gz

1. 解压

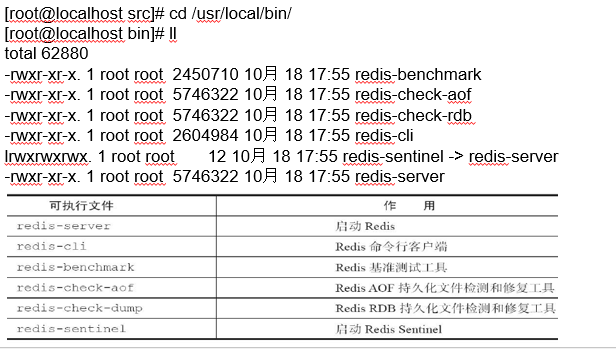
tar xzvf redis-4.0.2.tar.gz

1. 编译安装

make & make install

## 8.2.启动

make install命令执行后，默认安装路径/usr/local/bin



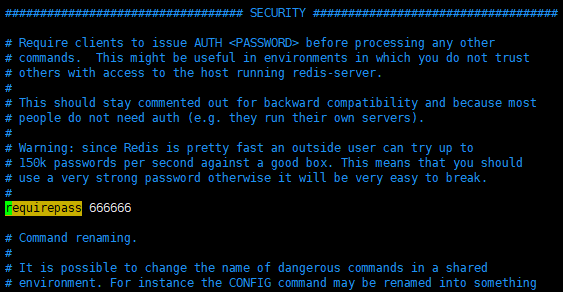
停止 

## 8.3.配置

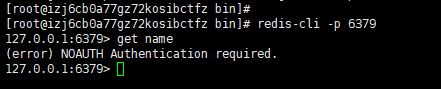
从解压的文件夹中拷贝一份redis.conf到当前目录，也可以自定义目录。

配置redis服务器验证密码

vim redis.conf #配置默认密码666666

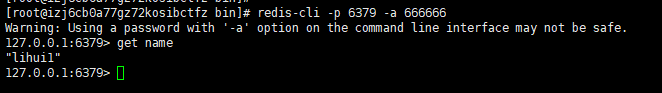


验证密码：



不使用密码能够登录进去，但是不能读取数据。

redis-cli –p 6379 –a 666666 #-a 表示输入密码



## 8.4.主从配置

计划：一主一从，两个redis实例，复制redis.conf一份。都改名分别叫做redis6379.conf,redis16379.conf

主机默认端口号：6379，配置文件redis6379.conf

从机默认端口号：16379，配置文件redis16379.conf

修改redis16379.conf

slaveof 127.0.0.1 6379

masterauth 666666

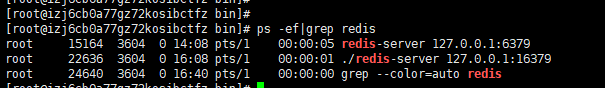
requirepass 666666

启动redis主机、从机

redis-server redis6379.conf

redis-server redis16379.conf

输入ps –ef|grep redis



## 8.5.redis集群，哨兵模式

redis哨兵（sentinel）实现主从切换

由于服务器资源，此处我们配置1个哨兵进程：

vim sentinel.conf

port 26379

sentinel monitor mymaster 127.0.0.1 6000 1 #1这里是当master挂掉后，计算哪一台哨兵升级成master的权值。

sentinel auth-pass mymaster 123456

启动sentinel服务

redis-server sentinel.conf --sentinel

## 8.6.redis 持久化

Redis有两种持久化方式AOF和RDB

RDB优势：

* RDB是redis数据库在一个时间节点产生的单个的备份文件。RDB文件非常适合备份。
* RDB非常适合灾难恢复，可以将单个压缩文件传输到远端数据中心，进行恢复。
* RDB最大限度的提高了redis的性能，因为redis父进程为了保证单线程，fork一个子进程进行备份操作。
* 于AOF相比，RDB允许使用大数据集更快的重启。

RDB缺点

* 按时间点备份容易丢失故障点到备份时间点的数据
* 需要经常fork，数据集大且CPU性能不佳，可能会导致redis停止对客户端的服务几毫秒甚至是一秒中。

AOF优势

* 使用AOF redis更持久：可以选择不同的fsync策略（fsync使用后台线程执行，主线程在没有fsync线程执行的时候尝试写入）。
* 偶发性问题， AOF在恢复时，遇到阻塞命令恢复时产生的数据集和原有的数据集有微小差异。此问题只在测试环境偶发过。

RDB持久化配置

redis.conf

save 900 1 #15分钟内 有一个key变动备份。

save 300 10 #5 分钟内 有10个key变动备份

save 60 1000 #60秒内 有1000个key变动备份

dbfilename "dump.rdb" #持久化文件名称

dir "/data/dbs/redis/6379" #持久化数据文件存放的路径

AOF持久化配置：

Vim redis.conf

[redis@iZ]$ more ~/redis/conf/redis.conf

dir "/data/dbs/redis/6379" #AOF文件存放目录

appendonly yes #开启AOF持久化，默认关闭

appendfilename "appendonly.aof" #AOF文件名称（默认）

appendfsync no #AOF持久化策略

auto-aof-rewrite-percentage 100 #触发AOF文件重写的条件（默认）

auto-aof-rewrite-min-size 64mb #触发AOF文件重写的条件（默认）

# 9.客户端应用