速度快：所有数据都存储在内存中。

Redis后台启动: ./redis-server ../redis.conf

关闭：./redis-cli shutdown

配置文件：redis.conf

JedisConnectionException

<http://blog.csdn.net/u012661010/article/details/73822712>

# 基础知识

## Jedis



## 五种数据类型及其使用场景

<https://mp.weixin.qq.com/s/d3PEkN6Jn3Lv1K5PNLMevA>

**字符串**（String）、字符串列表（list）、有序字符串集合（sorted set）、**哈希**（hash）、字符串集合（set）。

Key定义不要太长，最好不要超过1024，也不能太短，考虑可读性。

### String

这个没啥好说的，最常规的 set/get 操作，Value 可以是 String 也可以是数字。一般做一些复杂的计数功能的缓存。

Value最大为512M。

set key value：赋值

get key：取值

getset key value：先取值后赋值

del key：删除

incr key：将指定值递增1，不存在则初始为0在自增（结果为1），不能转为整型则出错

decr key：递减，没有初始0并减一（结果为-1），不能转型则出错

incrby key num：指定值增加num

decrby key num：指定值减少num

### hash

这里 Value 存放的是结构化的对象，比较方便的就是操作其中的某个字段。

在做**单点登录**的时候，就是用这种数据结构存储用户信息，以 CookieId 作为 Key，设置 30 分钟为缓存过期时间，能很好的模拟出类似 Session 的效果。

String key和String value的map容器

赋值、取值

hset key field value：

hmset key field value [field value …]：新增多个元素

hget key field：

hmget key key …：获取多个元素

hgetall key：获取所有元素

删除

hdel key …：

### list

使用 List 的数据结构，可以做简单的消息队列的功能。另外还有一个就是，可以利用 lrange 命令，做基于 Redis 的分页功能，性能极佳，用户体验好。

按照插入顺序排序的链表

两端添加、查看链表、两端弹出、获取个数、扩展。

lpush key value …：头插入

rpush key value …：右部插入

lrange key start end

lpop key：左边弹出

rpop key：右边弹出

llen key：长度

lrem key count value：删除count个值为value的元素

lset key index value：下标index处设置值为value

linsert key before value insertValue：在值为value的元素之前插入insertVlaue

linsert key after value insertValue：

### set

因为 Set 堆放的是一堆不重复值的集合。所以可以做全局去重的功能。为什么不用 JVM 自带的 Set 进行去重？因为我们的系统一般都是集群部署，使用 JVM 自带的 Set，比较麻烦，难道为了一个做一个全局去重，再起一个公共服务，太麻烦了。另外，就是利用交集、并集、差集等操作，可以计算共同喜好，全部的喜好，自己独有的喜好等功能。

没有排序的字符集合，不能出现重复元素，可在服务器端完成聚合操作，效率高。

交、差和并集运算。

sadd key value …：添加元素

srem key value … ：删除元素

smembers key：查看

sismember key value：判断元素是否存在

sdiff key1 key2：差集运算（仅第一个不同元素）

sinter key1 key2：交集运算

sunion key1 key2：并集运算

scard key：数量

srandmember key：一个随机元素

sdiff/inter/union**store** key1 key2 key3：将key2和key3的差/交/并集存到key1中

### sorted-set

可以做排行榜应用，取 TOP N 操作。Sorted Set 可以用来做延时任务。最后一个应用就是可以做范围查找。

成员在集合中的位置是有序的，排序依赖于每一个元素的“分数”

zadd key score value …：增加分数为score的元素value

zscore key value：获得指定元素value的分数

zcard key：成员个数

zrem key value …：删除元素

zrange key from to withscores limit offset rows：范围查询，包括分数

zrevrange key from to withscores：倒序范围查询

zremrangebyrank key from to：范围删除

zremrangebyscire key from to：分数范围删除

zcount key from to：按分数计数

游戏积分排行榜，构建索引数据。

## 键操作

keys \*：查看所有键

keys my?：匹配键为my开头的键

exists key：验证键是否存在

rename key newKey：重命名键

expire key 100：超时时间

ttl key：剩余超时时间

type key：获取key的数据类型

# Redis特性

## 多数据库

一个redis实例最多16个数据库（0-15）。

select index：选择数据库，默认选择0号数据库。

move key index：移动键到指定数据库

## 事务

mulit、exec、discard操纵事务。

mulit：开启事务，之后的命令被存到命令队列。

exec：提交事务

discard：回滚操作

# 持久化

## RDB

在指定的时间间隔内将内存中的数据写入到磁盘

优势：

1. 只包含一个文件，利于文件备份
2. 利于灾难恢复
3. 性能最大化
4. 启动快速

缺点：

宕机情况时数据完全丢失，数据量大时写入磁盘耗时

## AOF

以日志的形式记录每一个操作，启动时通过读取日志重建数据

优势：

1. 更高的数据安全性
2. 日志以追加的形式添加，已有数据不会丢失
3. 日志过大时，Redis能加以操作记录

效率低于RDB，磁盘占用大于RDB。

通过修改配置文件以禁用持久化，同时使用RDB和AOF

# 为什么使用 redis

1. 我们在碰到需要执行耗时特别久，且结果不频繁变动的 SQL，就特别适合将运行结果放入缓存。这样，后面的请求就去缓存中读取，使得请求能够迅速响应。
2. 在大并发的情况下，所有的请求直接访问数据库，数据库会出现连接异常。这个时候，就需要使用 Redis 做一个缓冲操作，让请求先访问到 Redis，而不是直接访问数据库。

# Redis 为什么这么快

1. 纯内存操作
2. 单线程操作，避免了频繁的上下文切换
3. 采用了非阻塞 I/O 多路复用机制

redis-client 在操作的时候，会产生具有不同事件类型的 Socket。在服务端，有一段 I/O 多路复用程序，将其置入队列之中。然后，文件事件分派器，依次去队列中取，转发到不同的事件处理器中。需要说明的是，这个 I/O 多路复用机制，Redis 还提供了 select、epoll、evport、kqueue 等多路复用函数库。

<https://www.cnblogs.com/aspirant/p/6877350.html?utm_source=itdadao&utm_medium=referral>

# Redis 的过期键删除策略和内存淘汰机制

<https://blog.csdn.net/qq_28018283/article/details/80764518>

## 过期键删除策略

Redis 采用的是定期删除+惰性删除策略

### 定时删除

设置键的过期时间时，创建一个 Timer ，当过期时间到临时，立刻删除键。

内存友好型策略，一旦键过期，就会被删除，并释放所占用的内存，Cpu 不友好，当一批数量比较多的键过期时，正好遇上Cpu 紧张的时段，这时候需要的是Cpu处理能力，而不是内存，显然 Cpu 时间用在删除过期键上，会对服务器的响应时间和吞吐量造成影响。另外当前 Redis 时间事件（无序链表O(N)）无法高效处理大量时间事件，所以定时删除并不是一种好的定时删除策略。

### 惰性删除

不管过期的键，在这种策略下，当键在键空间中被取出时，首先检查取出的键是否过期，若过期删除该键，否则，返回该键。

很明显，惰性删除**依赖过期键的被动访问，对于内存不友好**，如果一些键长期没有被访问，会造成内存泄露（垃圾数据占用内存）。我们知道，Redis是依赖内存的，所以惰性删除也不是一个好的策略。

### 定期删除

由定时删除算法，定期的去检查一定的数据库，删除一定的过期键。

通过合理的删除操作执行的时长和频率，达到合理的删除过期键。

Redis 默认每个 100ms随机抽取检查一次。

## RDB和AOF文件

执行 SAVE 或 BGSAVE 时 ，数据库键空间中的过期键不会被保存在RDB文件中

载入RDB文件时：

* Master 载入RDB时，文件中的未过期的键会被正常载入，过期键则会被忽略
* Slave 载入 RDB 时，文件中的所有键都会被载入，当同步进行时，会和Master 保持一致

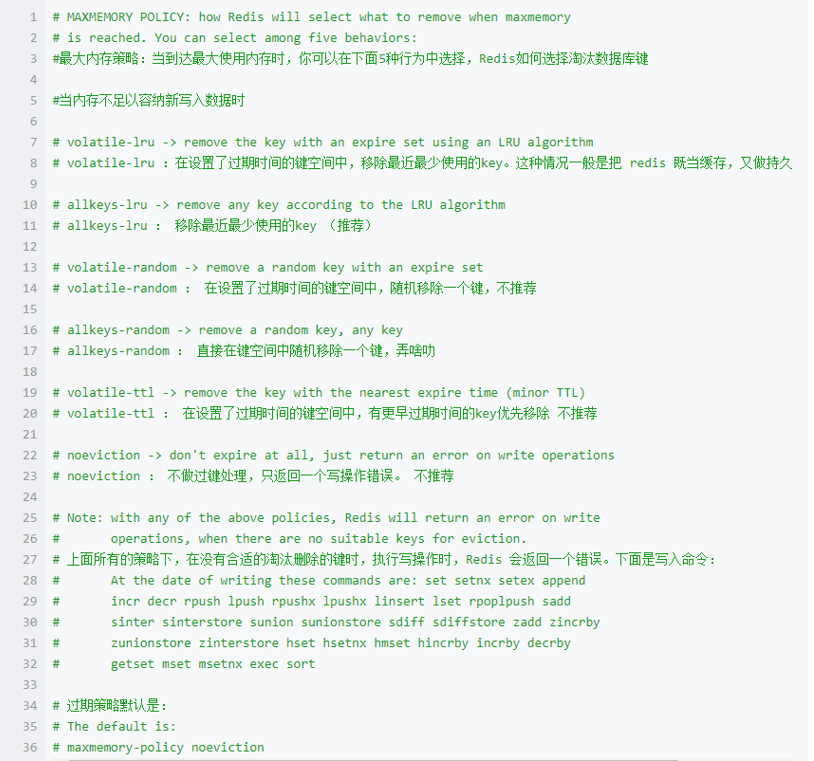
AOF 日志的方式会记录 redis 的每一步操作，所以键空间键过期的状态会被记录到 AOF 文件，当过期键发生释放删除时，DEL 也会被同步到 AOF 文件中。

重新生成 AOF文件时

执行 BGREWRITEAOF 时 ，数据库键中过期的键不会被记录到 AOF 文件中。

## 内存淘汰机制

如果定期删除没删除 Key。然后你也没即时去请求 Key，也就是说惰性删除也没生效。这样，Redis的内存会越来越高。那么就应该采用内存淘汰机制。



# Redis 和数据库双写一致性问题

<https://blog.csdn.net/hjm4702192/article/details/80518922>

一致性问题是分布式常见问题，还可以再分为最终一致性和强一致性。如果对数据有强一致性要求，不能放在redis（缓存）中。

采取正确更新策略，先更新数据库，再删缓存。其次，因为可能存在删除缓存失败的问题，提供一个补偿措施即可，例如利用消息队列。

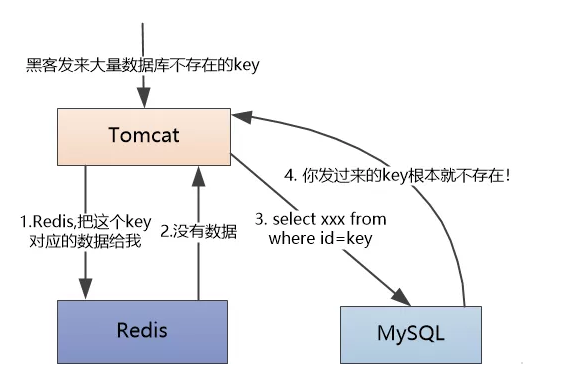
1. 理论上来说，给缓存设置过期时间，是保证最终一致性的解决方案。
2. 先更新数据库，再删除缓存

* 失效：应用程序先从cache取数据，没有得到，则从数据库中取数据，成功后，放到缓存中。
* 命中：应用程序从cache中取数据，取到后返回。
* 更新：先把数据存到数据库中，成功后，再让缓存失效。

# 如何应对缓存穿透和缓存雪崩问题

## 缓存穿透

故意去查询那些一定不存在的数据，缓存中肯定没有，于是请求一定会发到MySQL去执行，在流量大时，MySQL就挂掉了。



## 缓存雪崩

缓存同一时间大面积的失效，这个时候又来了一波请求，结果请求都怼到数据库上，从而导致数据库连接异常。

# Redis 的并发竞争 Key 问题

同时有多个子系统去 Set 一个 Key

1. Redis 事务机制
2. set操作没有顺序要求时，使用分布式锁，子服务都去抢锁，抢到锁就做 set 操作即可
3. 有顺序要求，分布式锁增加时间戳，或set 方法变成串行访问

# 旁路缓存（Cache Aside Pattern）

<https://blog.csdn.net/z50l2o08e2u4aftor9a/article/details/81008933>

## 读实践

先读cache，再读db

如果，cache hit，则直接返回数据

如果，cache miss，则访问db，并将数据set回缓存

## 写实践

淘汰缓存，而不是更新缓存

先操作数据库，再淘汰缓存