Redis

尚硅谷 java 研究院

第 1 章 NoSQL 简介

1.1 技术的分类

1) 解决功能性的问题
   Java、Servlet、Jsp、Tomcat、RDBMS、JDBC、Linux、Svn 等
2) 解决扩展性的问题
   Spring、 SpringMVC、SpringBoot、Hibernate、MyBatis 等
3) 解决性能的问题
   NoSQL、Java 多线程、Nginx、MQ、ElasticSearch、Hadoop 等

1.2 WEB1.0 及 WEB2.0

1) Web1.0 的时代,数据访问量很有限，用一夫当关的高性能的单节点服务器可以解决大部分问题.

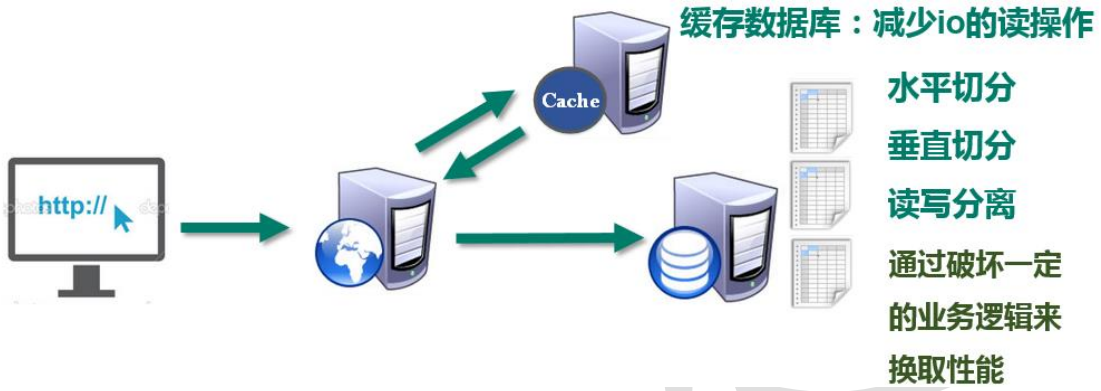
2) Web2.0 时代的到来，用户访问量大幅度提升，同时产生了大量的用户数据，加上后来的智能移动设备的普及，所有的互联网平台都面临了巨大的性能挑战.

1.3 解决服务器 CPU 内存压力

思考: Session 共享问题如何解决?

- 方案一、存在 Cookie 中
  此种方案需要将 Session 数据以 Cookie 的形式存在客户端,不安全，网络负担效率低
- 方案二、存在文件服务器或者是数据库里
  此种方案会导致大量的 IO 操作，效率低.
- 方案三、Session 复制
  此种方案会导致每个服务器之间必须将 Session 广播到集群内的每个节点，Session 数据会冗余，节点越多浪费越大,存在广播风暴问题.
- 方案四、存在 Redis 中
  目前来看，此种方案是最好的。将 Session 数据存在内存中，每台服务器都从内存中读取数据,速度快，结构还相对简单.

1.4 解决 IO 压力



将活跃的数据缓存到 Redis 中，客户端的请求先打到缓存中来获取对应的数据，如果能获取到，直接返回，不需要从 MySQL 中读取。如果缓存中没有，再从 MySQL 数据库中读取数据，将读取的数据返回并存一份到 Redis 中，方便下次读取.

扩展: 对于持久化的数据库来说，单个库单个表存在性能瓶颈，因此会通过水平切分、垂直切分、读取分离等技术提升性能，此种解决方案会破坏一定的业务逻辑，但是可以换取更高的性能.

1.5 NoSQL 数据库概述

1) NoSQL(NoSQL = Not Only SQL)，意即“不仅仅是 SQL”，泛指非关系型的数据库。
   NoSQL 不依赖业务逻辑方式存储，而以简单的 key-value 模式存储。因此大大的增加了数据库的扩展能力。
2) NoSQL 的特点
   不遵循 SQL 标准
   不支持 ACID
   远超于 SQL 的性能。
3) NoSQL 的适用场景
   对数据高并发的读写
   海量数据的读写
   对数据高可扩展性的
4) NoSQL 的不适用场景
   需要事务支持
   基于 sql 的结构化查询存储，处理复杂的关系,需要即席查询。
5) 建议： **用不着 sql 的和用了 sql 也不行的情况，请考虑用 NoSql**

1.6 常用的缓存数据库

1) Memcached

- Memcached
  - 很早出现的NoSql数据库
  - 数据都在内存中，一般不持久化
  - 支持简单的key-value模式
  - 一般是作为缓存数据库辅助持久化的数据库

2) Redis

- Redis
  - 几乎覆盖了Memcached的绝大部分功能
  - 数据都在内存中，支持持久化，主要用作备份恢复
  - 除了支持简单的key-value模式，还支持多种数据结构的存储，比如 list、set、hash、zset等。
  - 一般是作为缓存数据库辅助持久化的数据库

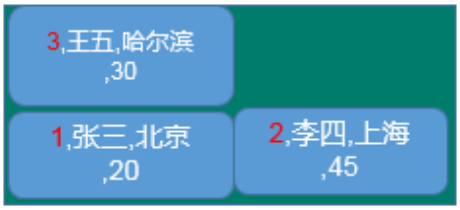
3) mongoDB

- mongoDB
  - 高性能、开源、模式自由(schema free)的文档型数据库
  - 数据都在内存中， 如果内存不足，把不常用的数据保存到硬盘
  - 虽然是key-value模式，但是对value（尤其是json）提供了丰富的查询功能
  - 支持二进制数据及大型对象
  - 可以根据数据的特点替代RDBMS ，成为独立的数据库。或者配合RDBMS，存储特定的数据。

4) 列式数据库

- 先看行式数据库

id	name	city	age
1	张三	北京	20
2	李四	上海	45
3	王五	哈尔滨	30



思考: 如下两条 SQL 的快慢

```
select * from users where id =3
select avg(age) from users
```

- 再看列式数据库

id	name	city	age
1	张三	北京	20
2	李四	上海	45
3	王五	哈尔滨	30

张三,李四,王五	1,2,3
北京，上海，哈尔滨	20,45,30

5) HBase

➢ HBase

- HBase是Hadoop项目中的数据库。它用于需要对大量的数据进行随机、实时的读写操作的场景中。HBase的目标就是处理数据量非常庞大的表，可以用普通的计算机处理超过10亿行数据，还可处理有数百万列元素的数据表。

6) Cassandra

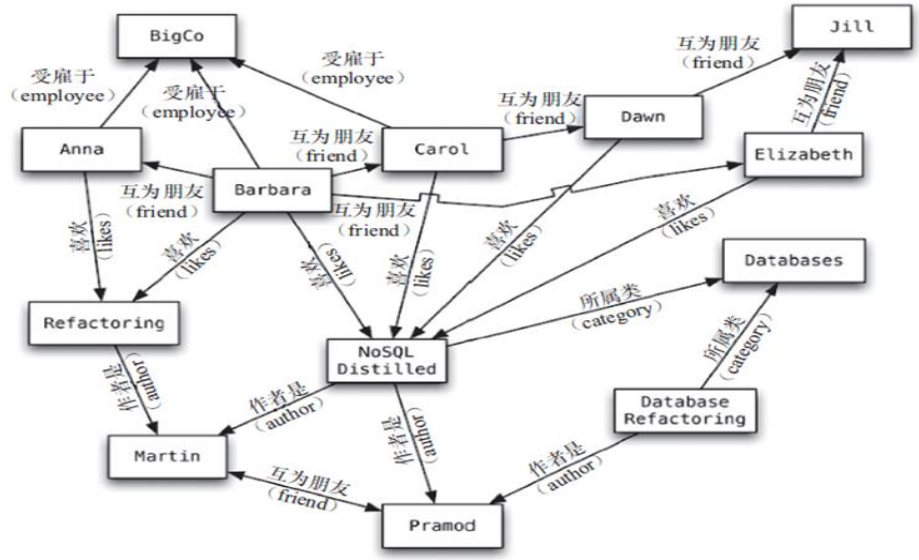
➢ Cassandra

- Apache Cassandra是一款免费的开源NoSQL数据库，其设计目的在于管理由大量商用服务器构建起来的庞大集群上的**海量数据集(数据量通常达到PB级别)**。在众多显著特性当中，Cassandra最为卓越的长处是对写入及读取操作进行规模调整，而且其不强调主集群的设计思路能够以相对直观的方式简化各集群的创建与扩展流程。

7) Neo4j

➢ Neo4j

- 主要应用：社会关系，公共交通网络，地图及网络拓谱



1.7 数据库排名

http://db-engines.com/en/ranking

DB-Engines 数据库排名

Rank Dec 2018	Rank Nov 2018	Rank Dec 2017	DBMS	Database Model	Score Dec 2018	Score Nov 2018	Score Dec 2017
1.	1.	1.	Oracle	Relational DBMS	1283.22	-17.89	-58.32
2.	2.	2.	MySQL	Relational DBMS	1161.25	+1.36	-156.82
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1040.34	-11.21	-132.14
4.	4.	4.	PostgreSQL	Relational DBMS	460.64	+20.39	+75.21
5.	5.	5.	MongoDB	Document store	378.62	+9.14	+47.85
6.	6.	6.	IBM Db2	Relational DBMS	180.75	+0.87	-8.83
7.	7.	↑ 8.	Redis	Key-value store	146.83	+2.66	+23.59
8.	8.	↑ 10.	Elasticsearch	Search engine	144.70	+1.24	+24.92
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	139.51	+1.08	+13.63
10.	10.	↑ 11.	SQLite	Relational DBMS	123.02	+0.31	+7.82
11.	11.	↓ 9.	Cassandra	Wide column store	121.81	+0.07	-1.40
12.	12.	↑ 15.	Splunk	Search engine	82.18	+1.81	+18.39
13.	13.	↓ 12.	Teradata	Relational DBMS	79.16	-0.14	+4.42
14.	14.	↑ 17.	MariaDB	Relational DBMS	77.26	+4.01	+20.52
15.	15.	↑ 19.	Hive	Relational DBMS	67.38	+2.81	+12.71
16.	16.	↓ 13.	Solr	Search engine	61.35	+0.47	-4.95
17.	17.	↓ 16.	HBase	Wide column store	60.01	-0.40	-3.40
18.	↑ 20.	18.	FileMaker	Relational DBMS	56.65	+0.90	+1.45
19.	19.	↑ 20.	SAP HANA	Relational DBMS	56.31	+0.43	+9.82
20.	↓ 18.	↓ 14.	SAP Adaptive Server	Relational DBMS	55.82	-0.75	-9.86
21.	21.	↑ 22.	Amazon DynamoDB	Multi-model	54.30	+0.48	+17.58
22.	22.	↓ 21.	Neo4j	Graph DBMS	45.56	+2.44	+6.83
23.	23.	23.	Couchbase	Document store	35.59	+0.74	+4.28
24.	24.	24.	Memcached	Key-value store	29.62	-0.14	+1.18

第 2 章 Redis 简介 及 安装

2.1 Redis 是什么

Redis 是一个开源的 key-value 存储系统。和 Memcached 类似，它支持存储的 value 类型相对更多，包括 string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和 hash（哈希类型）。这些数据类型都支持 push/pop、add/remove 及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，Redis 支持各种不同方式的排序。与 memcached 一样，为了保证效率，数据都是缓存在内存中。区别的是 Redis 会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了 master-slave(主从)同步。

2.2 Redis 的应用场景

1) 配合关系型数据库做高速缓存
- 高频次，热门访问的数据，降低数据库 IO
- 高频次，热门访问的数据，降低数据库 IO

2) 由于其拥有持久化能力，利用其多样的数据结构存储特定的数据
- 最新 N 个数据 ➔ 通过 List 实现按自然事件排序的数据

- 排行榜，TopN ➔ 利用 zset(有序集合)
- 时效性的数据，比如手机验证码➔ Expire 过期
- 计数器，秒杀 ➔ 原子性，自增方法 INCR、DECR
- 去除大量数据中的重复数据➔ 利用 set 集合
- 构建队列➔利用 list 集合
- 发布订阅消息系统 ➔ pub/sub 模式

2.3 Redis 官网

1) Redis 官方网站 http://Redis.io
2) Redis 中文官方网站 http://www.Redis.net.cn

2.4 关于 Redis 版本

1) 3.2.5 for Linux
2) 不用考虑在 Windows 环境下对 Redis 的支持
   Redis 官方没有提供对 Windows 环境的支持，是微软的开源小组开发了对 Redis 对 Windows 的支持.

2.5 安装步骤

1) 下载获得 redis-3.2.5.tar.gz 后将它放入我们的 Linux 目录/opt
2) 解压命令:tar -zxvf redis-3.2.5.tar.gz
3) 解压完成后进入目录:cd redis-3.2.5
4) 在 redis-3.2.5 目录下执行 make 命令
- 运行 Make 命令时出现错误,提示 gcc：命令未找到 ,原因是因为当前 Linux 环境中并没有安装 gcc 与 g++ 的环境

5) 安装 gcc 与 g++
- 能上网的情况:
  yum install gcc
  yum install gcc-c++
- 不能上网[建议]
  参考 Linux 课程中<<03_在 VM 上安装 CentOS7>>中的第 40 步骤

6) 重新进入到 Redis 的目录中执行 make distclean 后再执行 make 命令.
7) 执行完 make 后，可跳过 Redis test 步骤，直接执行 make install

2.6 查看默认安装目录 /usr/local/bin

1) Redis-benchmark:性能测试工具，可以在自己本子运行，看看自己本子性能如何(服务启动起来后执行)

2) Redis-check-aof：修复有问题的 AOF 文件，rdb 和 aof 后面讲
3) Redis-check-dump：修复有问题的 dump.rdb 文件
4) Redis-sentinel：Redis 集群使用
5) redis-server：Redis 服务器启动命令
6) redis-cli：客户端，操作入口

2.7 Redis 的启动

1) 默认前台方式启动
- 直接执行 redis-server 即可.启动后不能操作当前命令窗口

2) 推荐后台方式启动
- 拷贝一份 redis.conf 配置文件到其他目录，例如根目录下的 myredis 目录 /myredis
- 修改 redis.conf 文件中的一项配置 daemonize 将 no 改为 yes，代表后台启动
- 执行配置文件进行启动 执行 redis-server /myredis/redis.conf

2.8 客户端访问

1) 使用 redis-cli 命令访问启动好的 Redis
- 如果有多个 Redis 同时启动，则需指定端口号访问 redis-cli -p 端口号

2) 测试验证,通过 ping 命令 查看是否 返回 PONG

2.9 关闭 Redis 服务

1) 单实例关闭
- 如果还未通过客户端访问，可直接 redis-cli shutdown
- 如果已经进入客户端,直接 shutdown 即可.

2) 多实例关闭
- 指定端口关闭 redis-cli -p 端口号 shutdown

2.9 Redis 端口号的由来

1) 端口号来自一位影星的名字 . Alessia Merz

2.10 Redis 默认 16 个库

1) Redis 默认创建 16 个库,每个库对应一个下标,从 0 开始.
通过客户端连接后默认进入到 0 号库，推荐只使用 0 号库.

2) 使用命令 select 库的下标 来切换数据库，例如 select 8

2.11 Redis 的单线程+多路 IO 复用技术

1) 多路复用是指使用一个线程来检查多个文件描述符（Socket）的就绪状态，比如调用 select 和 poll 函数，传入多个文件描述符，如果有一个文件描述符就绪，则返回，否则阻塞直到超时。得到就绪状态后进行真正的操作可以在同一个线程里执行，也可以启动线程执行（比如使用线程池）。
2) Memcached 是 多线程 + 锁.
Redis 是 单线程 + 多路 IO 复用.

第 3 章 Redis 的五大数据类型

3.1 key

keys *	查看当前库的所有键
exists <key>	判断某个键是否存在
type <key>	查看键的类型
del <key>	删除某个键
expire <key> <seconds>	为键值设置过期时间，单位秒
ttl <key>	查看还有多久过期,-1 表示永不过期,-2 表示已过期
dbsize	查看当前数据库中 key 的数量
flushdb	清空当前库
Flushall	通杀全部库

3.2 String

1) String 是 Redis 最基本的类型，你可以理解成与 Memcached 一模一样的类型，一个 key 对应一个 value
2) String 类型是二进制安全的。意味着 Redis 的 string 可以包含任何数据。比如 jpg 图片或者序列化的对象 。
3) String 类型是 Redis 最基本的数据类型，一个 Redis 中字符串 value 最多可以是 512M
4) 常用操作

get <key>	查询对应键值
set <key> <value>	添加键值对

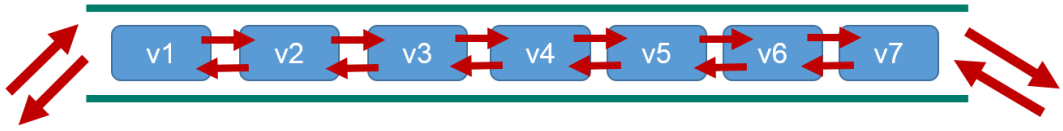
append <key> <value>	将给定的<value>追加到原值的末尾
strlen <key>	获取值的长度
senx <key> <value>	只有在 key 不存在时设置 key 的值
incr <key>	将 key 中存储的数字值增 1 只能对数字值操作，如果为空，新增值为 1
decr <key>	将 key 中存储的数字值减 1 只能对数字之操作，如果为空,新增值为-1
incrby /decrby <key> 步长	将 key 中存储的数字值增减，自定义步长
mset <key1> <value1> <key2> <value2>	同时设置一个或多个 key-value 对
mget <key1> <key2> <key3>	同时获取一个或多个 value
msetnx <key1> <value1> <key2> <value2>	同时设置一个或多个 key-value 对，当且仅当所有给定的 key 都不存在
getrange <key> <起始位置> <结束位置>	获得值的范围,类似 java 中的 substring
setrange <key> <起始位置> <value>	用<value>覆盖<key>所存储的字符串值，从<起始位置>开始
setex <key> <过期时间> <value>	设置键值的同时，设置过去时间，单位秒
getset <key> <value>	以新换旧,设置了新值的同时获取旧值

5) 详说 incr key 操作的原子性

- 所谓原子操作是指不会被线程调度机制打断的操作；这种操作一旦开始，就一直运行到结束，中间不会有任何 context switch （切换到另一个线程）。
- 在单线程中， 能够在单条指令中完成的操作都可以认为是" 原子操作"，因为中断只能发生于指令之间。
- 在多线程中，不能被其它进程（线程）打断的操作就叫原子操作。
- **Redis 单命令的原子性主要得益于 Redis 的单线程**
- 思考: java 中 i++是否是原子操作?

3.3 List

1) 单键多值
2) 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导列表的头部（左边）或者尾部（右边）。
3) 它的底层实际是个**双向链表，对两端的操作性能很高，通过索引下标的操作中间的节点性能会较差**
4)



5) 常用操作

lpush/rpush <key> <value1> <value2>	从左边/右边插入一个或多个值。

lpop/rpop <key>	从左边/右边吐出一个值。 值在键在，值光键亡。
rpoplpush <key1> <key2>	从<key1>列表右边吐出一个值，插到<key2>列表左边
lrange <key> <start> <stop>	按照索引下标获得元素(从左到右)
lindex <key> <index>	按照索引下标获得元素(从左到右)
llen <key>	获得列表长度
linsert <key> before <value> <newvalue>	在<value>的后面插入<newvalue> 插入值
lrem <key> <n> <value>	从左边删除 n 个 value(从左到右)

3.4 Set

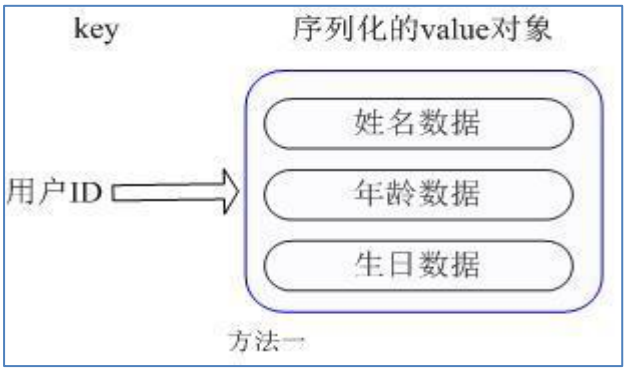
1) Redis set 对外提供的功能与 list 类似是一个列表的功能，特殊之处在于 set 是可以自动排重的，当你需要存储一个列表数据，又不希望出现重复数据时，set 是一个很好的选择，并且 set 提供了判断某个成员是否在一个 set 集合内的重要接口，这个也是 list 所不能提供的
2) Redis 的 Set 是 string 类型的无序集合。它底层其实是一个 value 为 null 的 hash 表,所以添加，删除，查找的复杂度都是 O(1)。
3) 常用操作

sadd <key> <value1> <value2>	将一个或多个 member 元素加入到集合 key 当中，已经存在于集合的 member 元素将被忽略。
smembers <key>	取出该集合的所有值。
sismember <key> <value>	判断集合<key>是否为含有该<value>值，有返回 1，没有返回 0
scard <key>	返回该集合的元素个数。
srem <key> <value1> <value2>	删除集合中的某个元素。
spop <key>	随机从该集合中吐出一个值。
srandmember <key> <n>	随机从该集合中取出 n 个值。 不会从集合中删除
sinter <key1> <key2>	返回两个集合的交集元素。
sunion <key1> <key2>	返回两个集合的并集元素。
sdiff <key1> <key2>	返回两个集合的差集元素。

3.5 Hash

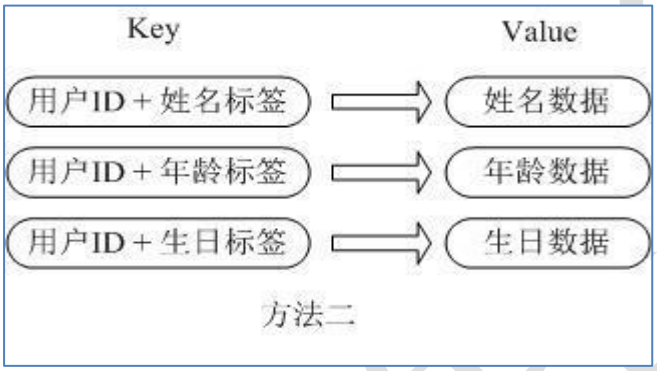
1) Redis hash 是一个键值对集合
2) Redis hash 是一个 string 类型的 field 和 value 的映射表，hash 特别适合用于存储对象。
3) 类似 Java 里面的 Map<String,Object>
4) 分析一个问题：现有一个 JavaBean 对象，在 Redis 中如何存?

- 第一种方案： 用户 ID 为 key ,VALUE 为 JavaBean 序列化后的字符串



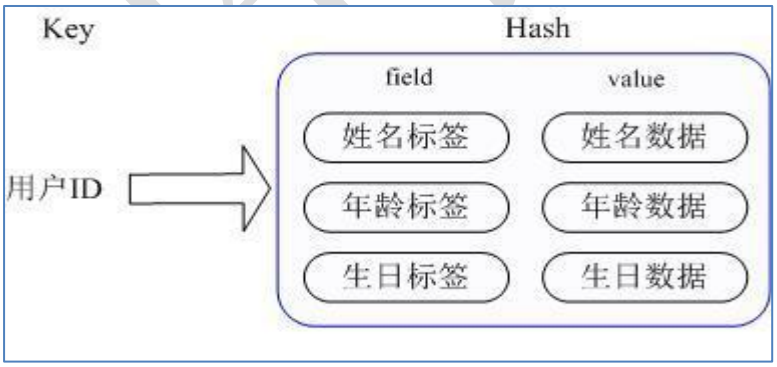
缺点：每次修改用户的某个属性需要，先反序列化改好后再序列化回去。开销较大

- 第二种方案：用户 ID+属性名作为 key，属性值作为 Value.



缺点：用户 ID 数据冗余

- 第三种方案：通过 key(用户 ID) + field(属性标签) 就可以操作对应属性数据了，既不需要重复存储数据，也不会带来序列化和并发修改控制的问题



5) 常用操作

hset <key> <field> <value>	给<key>集合中的 <field>键赋值<value>
hget <key1> <field>	从<key1>集合<field> 取出 value
hmset <key1> <field1> <value1> <field2> <value2>...	批量设置 hash 的值
hexists key <field>	查看哈希表 key 中，给定域 field 是否存在。

hkeys <key>	列出该 hash 集合的所有 field
hvals <key>	列出该 hash 集合的所有 value
hincrby <key> <field> <increment>	为哈希表 key 中的域 field 的值加上增量 increment
hsetnx <key> <field> <value>	将哈希表 key 中的域 field 的值设置为 value ，当且仅当域 field 不存在

3.6 zset (sorted set)

1) Redis 有序集合 zset 与普通集合 set 非常相似，是一个没有重复元素的字符串集合。不同之处是有序集合的每个成员都关联了一个评分（score），这个评分（score）被用来按照从最低分到最高分的方式排序集合中的成员。集合的成员是唯一的，但是评分可以是重复了 。
2) 因为元素是有序的, 所以你也可以很快的根据评分（score）或者次序（position）来获取一个范围的元素。访问有序集合的中间元素也是非常快的,因此你能够使用有序集合作为一个没有重复成员的智能列表。
3) 常用操作

zadd <key> <score1> <value1> <score2> <value2>...	将一个或多个 member 元素及其 score 值加入到有序集 key 当中
zrange <key> <start> <stop> [WITHSCORES]	返回有序集 key 中，下标在<start> <stop>之间的元素 带 WITHSCORES，可以让分数一起和值返回到结果集。
zrangebyscore key min max [withscores] [limit offset count]	返回有序集 key 中，所有 score 值介于 min 和 max 之间(包括等于 min 或 max)的成员。有序集成员按 score 值递增(从小到大)次序排列。
zrevrangebyscore key max min [withscores] [limit offset count]	同上，改为从大到小排列。
zincrby <key> <increment> <value>	为元素的 score 加上增量
zrem <key> <value>	删除该集合下，指定值的元素
zcount <key> <min> <max>	统计该集合，分数区间内的元素个数
zrank <key> <value>	返回该值在集合中的排名，从 0 开始。

4) 思考: 如何利用 zset 实现一个文章访问量的排行榜?

第 4 章 Redis 的相关配置

1) 计量单位说明,大小写不敏感

2) include
类似 jsp 中的 include，多实例的情况可以把公用的配置文件提取出来
3) ip 地址的绑定 bind
- 默认情况 bind=127.0.0.1 只能接受本机的访问请求
- 不写的情况下，无限制接受任何 ip 地址的访问
- 生产环境肯定要写你应用服务器的地址
- 如果开启了 protected-mode，那么在没有设定 bind ip 且没有设密码的情况下，Redis 只允许接受本机的相应

4) tcp-backlog
- 可以理解是一个请求到达后至到接受进程处理前的队列.
- backlog 队列总和=未完成三次握手队列 + 已经完成三次握手队列
- 高并发环境 tcp-backlog 设置值跟超时时限内的 Redis 吞吐量决定

5) timeout
一个空闲的客户端维持多少秒会关闭，0 为永不关闭。
6) tcp keepalive
对访问客户端的一种心跳检测，每个 n 秒检测一次，官方推荐设置为 60 秒
7) daemonize
是否为后台进程
8) pidfile
存放 pid 文件的位置，每个实例会产生一个不同的 pid 文件
9) log level
四个级别根据使用阶段来选择，生产环境选择 notice 或者 warning
10) log level
日志文件名称
11) syslog
是否将 Redis 日志输送到 linux 系统日志服务中
12) syslog-ident
日志的标志
13) syslog-facility
输出日志的设备
14) database
设定库的数量 默认 16
15) security
在命令行中设置密码

```
127.0.0.1:6379> config get requirepass
1) "requirepass"
2) ""
127.0.0.1:6379> config set requirepass "123456"
OK
127.0.0.1:6379> config get requirepass
(error) NOAUTH Authentication required.
127.0.0.1:6379> auth 123456
OK
127.0.0.1:6379> get k1
"v1"
127.0.0.1:6379> config set requirepass ""
OK
```

16) maxclient

最大客户端连接数

17) maxmemory

设置 Redis 可以使用的内存量。一旦到达内存使用上限，Redis 将会试图移除内部数据，移除规则可以通过 maxmemory-policy 来指定。如果 Redis 无法根据移除规则来移除内存中的数据，或者设置了“不允许移除”，

那么 Redis 则会针对那些需要申请内存的指令返回错误信息，比如 SET、LPUSH 等。

18) Maxmemory-policy

- volatile-lru：使用 LRU 算法移除 key，只对设置了过期时间的键
- allkeys-lru：使用 LRU 算法移除 key
- volatile-random：在过期集合中移除随机的 key，只对设置了过期时间的键
- allkeys-random：移除随机的 key
- volatile-ttl：移除那些 TTL 值最小的 key，即那些最近要过期的 key
- noeviction：不进行移除。针对写操作，只是返回错误信息

19) Maxmemory-samples

设置样本数量，LRU 算法和最小 TTL 算法都并非是精确的算法，而是估算值，所以你可以设置样本的大小。

一般设置 3 到 7 的数字，数值越小样本越不准确，但是性能消耗也越小。

第 5 章 Redis 的 Java 客户端 Jedis

1) Jedis 所需要的 jar 包 ,可通过 Maven 的依赖引入

Commons-pool-1.6.jar

Jedis-2.1.0.jar

2) 使用 Windows 环境下 Eclipse 连接虚拟机中的 Redis 注意事项

- 禁用 Linux 的防火墙：Linux(CentOS7)里执行命令 ： systemctl stop firewalld.service
- redis.conf 中注释掉 bind 127.0.0.1 ,然后 protect-mode no。

3) Jedis 测试连通性

```
public class Demo01 {
```

```
    public static void main(String[] args) {
        //连接本地的 Redis 服务
        Jedis jedis = new Jedis("127.0.0.1",6379);
        //查看服务是否运行，打出 pong 表示 OK
        System.out.println("connection is OK========>:"+jedis.ping());
    }
}
```

4) 完成一个手机验证码功能

要求:

1、输入手机号，点击发送后随机生成 6 位数字码，2 分钟有效

2、输入验证码，点击验证，返回成功或失败

3、每个手机号每天只能输入 3 次

第 6 章　Redis 事务

6.1 Redis 中事务的定义

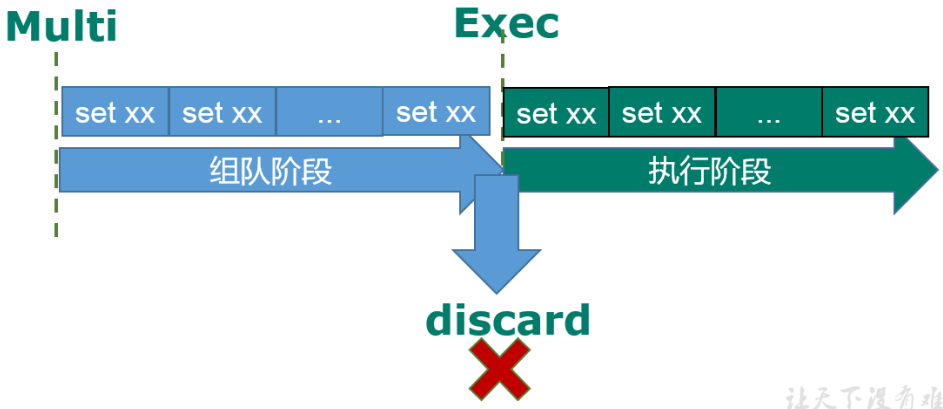
Redis 事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断

Redis 事务的主要作用就是串联多个命令防止别的命令插队

6.2 multi 、exec、discard

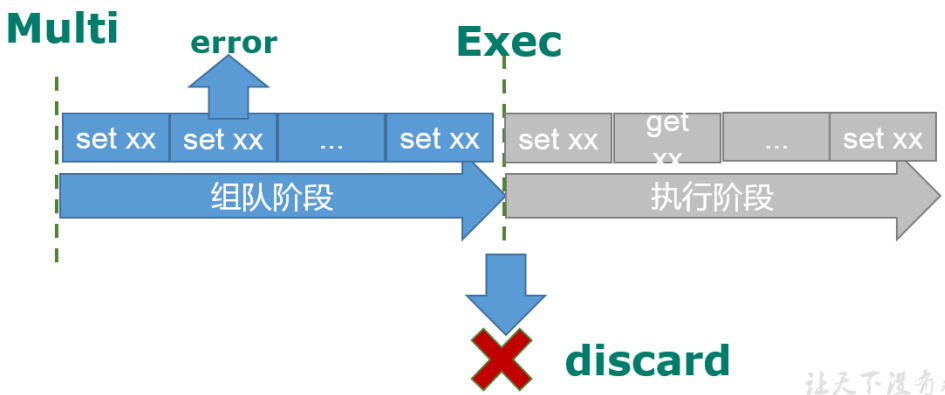
1) 从输入 Multi 命令开始，输入的命令都会依次进入命令队列中，但不会执行，至到输入 Exec 后，Redis 会将之前的命令队列中的命令依次执行。

2) 组队的过程中可以通过 discard 来放弃组队。

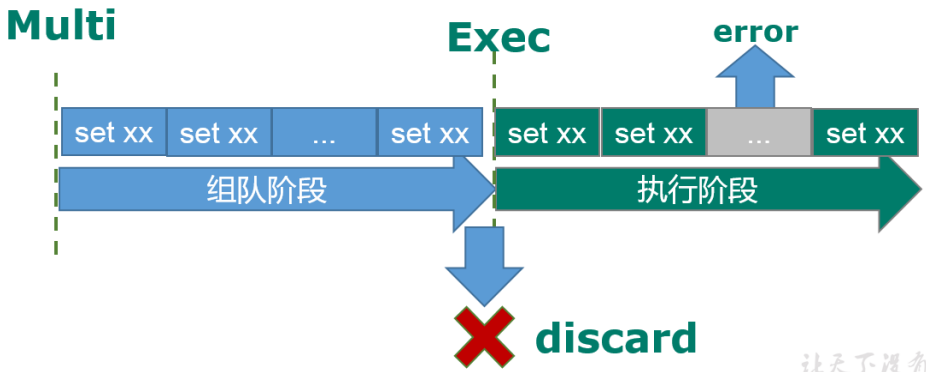


6.3 事务中的错误处理

1) 组队中某个命令出现了报告错误，执行时整个的所有队列会都会被取消。

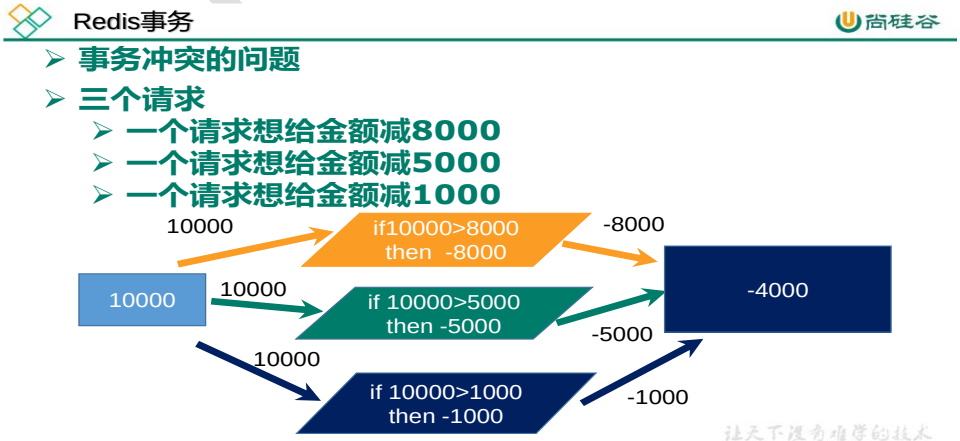


2) 如果执行阶段某个命令报出了错误，则只有报错的命令不会被执行，而其他的命令都会执行，不会回滚。

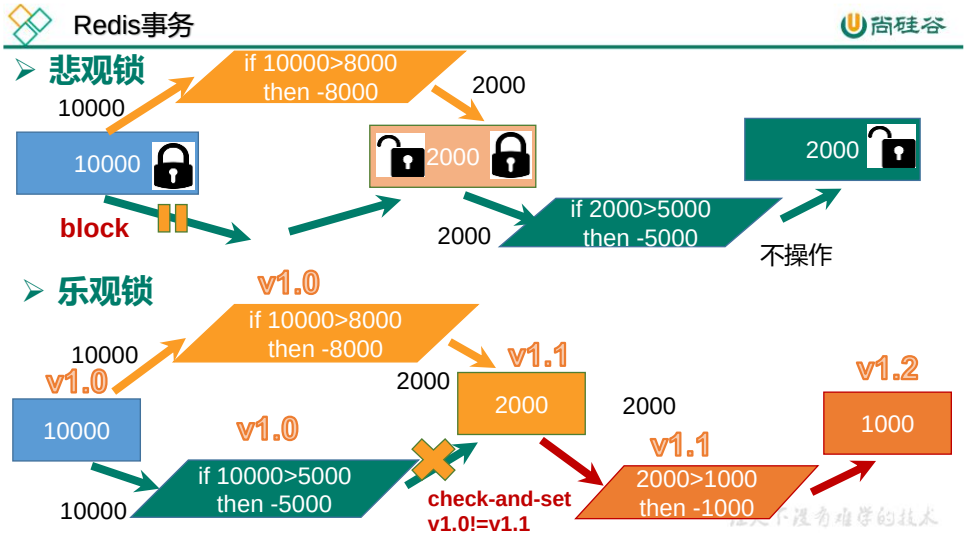


6.4 为什么要做成事务?

1) 想想一个场景: 有很多人有你的账户，同时去参加双十一抢购



2) 通过事务解决问题



悲观锁(Pessimistic Lock)，顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会 block 直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁

乐观锁(Optimistic Lock)，顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。乐观锁适用于多读的应用类型，这样可以提高吞吐量。Redis 就是利用这种 check-and-set 机制实现事务的。

6.5 Redis 事务的使用

1) WATCH key[key....]

在执行 multi 之前，先执行 watch key1 [key2],可以监视一个(或多个) key ，如果在事务执行之前这个(或这些) key 被其他命令所改动，那么事务将被打断。

```
127.0.0.1:6379> WATCH balance
OK
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> DECRBY balance 10
QUEUED
127.0.0.1:6379> INCRBY debt 10
QUEUED
127.0.0.1:6379> EXEC
1) (integer) 60
2) (integer) 40
```

2) unwatch

取消 WATCH 命令对所有 key 的监视。

如果在执行 WATCH 命令之后，EXEC 命令或 DISCARD 命令先被执行了的话，那么就不需要再执行 UNWATCH 了。

3) 三特性

- 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。

- 没有隔离级别的概念

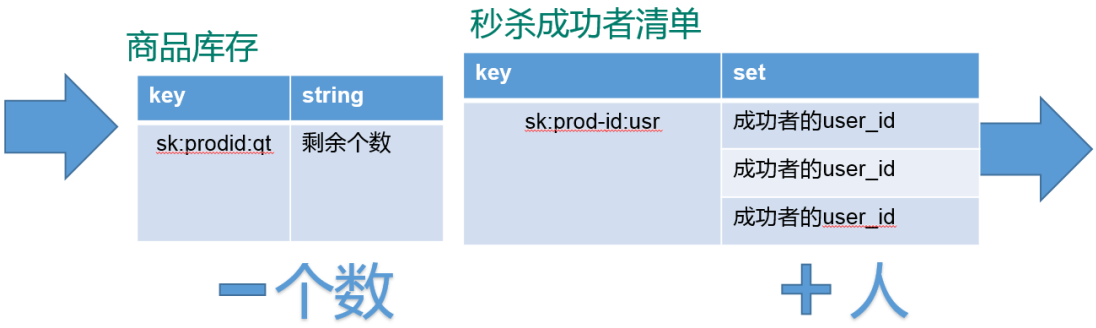
队列中的命令没有提交之前都不会实际的被执行，因为事务提交前任何指令都不会被实际执行，也就不存在“事务内的查询要看到事务里的更新，在事务外查询不能看到”这个让人万分头痛的问题

- 不保证原子性

Redis 同一个事务中如果有一条命令执行失败，其后的命令仍然会被执行，没有回滚

6.6 Redis 事务 秒杀案例

1) 解决计数器和人员记录的事务操作



2) 秒杀并发模拟 ab 工具

- CentOS6 默认安装 ,CentOS7 需要手动安装
- 联网: yum install httpd-tools

无网络: 进入 cd /run/media/root/CentOS 7 x86_64/Packages

顺序安装

apr-1.4.8-3.el7.x86_64.rpm

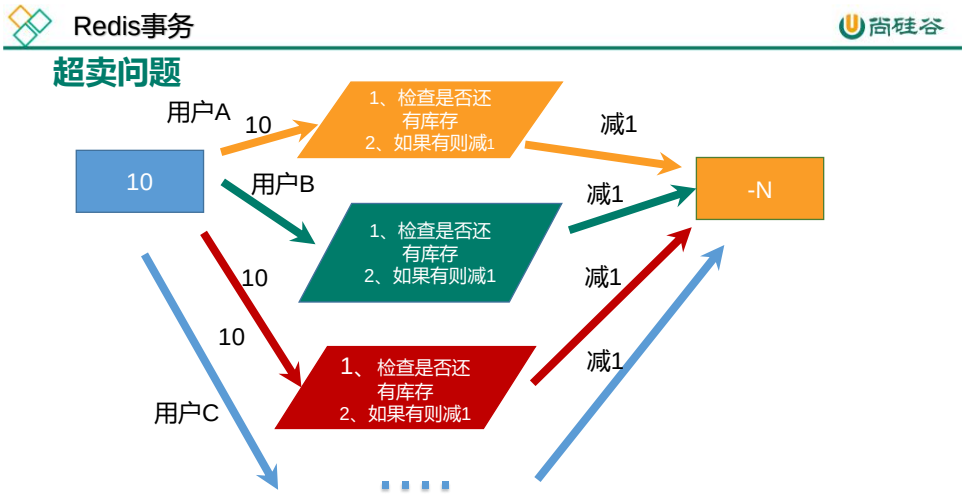
apr-util-1.5.2-6.el7.x86_64.rpm

httpd-tools-2.4.6-67.el7.centos.x86_64.rpm

- ab －n 请求数 -c 并发数 -p 指定请求数据文件

-T “application/x-www-form-urlencoded” 测试的请求

3) 超卖问题



4) 请求超时问题

节省每次连接 redis 服务带来的消耗，把连接好的实例反复利用

连接池参数:

MaxTotal：控制一个 pool 可分配多少个 jedis 实例，通过 pool.getResource()来获取；如果赋值为-1，则表示不限制；如果 pool 已经分配了 MaxTotal 个 jedis 实例，则此时 pool 的状态为 exhausted。

maxIdle：控制一个 pool 最多有多少个状态为 idle(空闲)的 jedis 实例；

MaxWaitMillis：表示当 borrow 一个 jedis 实例时，最大的等待毫秒数，如果超过等待时间，则直接抛 JedisConnectionException；

testOnBorrow：获得一个 jedis 实例的时候是否检查连接可用性（ping()）；如果为 true，则得到的 jedis 实例均是可用的；

5) 遗留问题

- LUA 脚本

Lua 是一个小巧的脚本语言，Lua 脚本可以很容易的被 C/C++ 代码调用，也可以反过来调用 C/C++的函数，Lua 并没有提供强大的库，一个完整的 Lua 解释器不过 200k，所以 Lua 不适合作为开发独立应用程序的语言，而是作为嵌入式脚本语言。

很多应用程序、游戏使用 LUA 作为自己的嵌入式脚本语言，以此来实现可配置性、可扩展性。这其中包括魔兽争霸地图、魔兽世界、博德之门、愤怒的小鸟等众多游戏插件或外挂

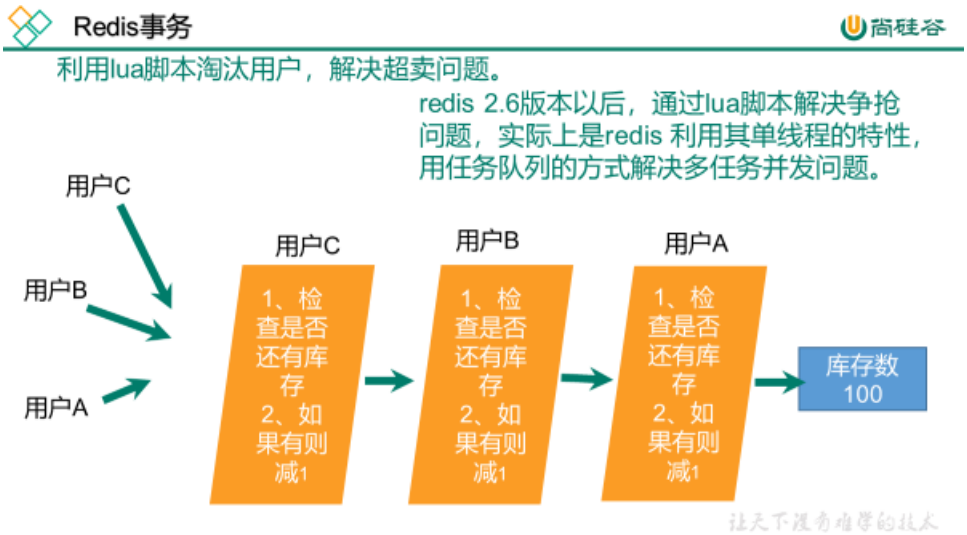
- LUA 脚本在 Redis 中的优势

将复杂的或者多步的 redis 操作，写为一个脚本，一次提交给 redis 执行，减少反复连接 redis 的次数。提升性能。

LUA 脚本是类似 redis 事务，有一定的原子性，不会被其他命令插队，可以完成一些 redis 事务性的操作

但是注意 redis 的 lua 脚本功能，只有在 2.6 以上的版本才可以使用。

- 利用 lua 脚本淘汰用户，解决超卖问题。



第 7 章 Redis 持久化

Redis 提供了 2 个不同形式的持久化方式 RDB 和 AOF

Redis Persistence

Redis provides a different range of persistence options:

- The RDB persistence performs point-in-time snapshots of your dataset at specified intervals.
- the AOF persistence logs every write operation received by the server, that will be played again at server startup, reconstructing the original dataset. Commands are logged using the same format as the Redis protocol itself, in an append-only fashion. Redis is able to rewrite the log on background when it gets too big.
- If you wish, you can disable persistence at all, if you want your data to just exist as long as the server is running.
- It is possible to combine both AOF and RDB in the same instance. Notice that, in this case, when Redis restarts the AOF file will be used to reconstruct the original dataset since it is guaranteed to be the most complete.

The most important thing to understand is the different trade-offs between the RDB and AOF persistence. Let's start with RDB:

7.2 RDB

1) 在指定的时间间隔内将内存中的数据集快照写入磁盘，也就是行话讲的 Snapshot 快照，它恢复时是将快照文件直接读到内存里。
2) 备份是如何执行的
   Redis 会单独创建（fork）一个子进程来进行持久化，会先将数据写入到一个临时文件中，待持久化过程都结束了，再用这个临时文件替换上次持久化好的文件。整个过程中，主进程是不进行任何 IO 操作的，这就确保了极高的性能如果需要进行大规模数据的恢复，且对于数据恢复的完整性不是非常敏感，那 RDB 方式要比 AOF 方式更加的高效。RDB 的缺点是最后一次持久化后的数据可能丢失。
3) 关于 fork

在 Linux 程序中，fork()会产生一个和父进程完全相同的子进程，但子进程在此后多会 exec 系统调用，出于效率考虑，Linux 中引入了“写时复制技术”，一般情况父进程和子进程会共用同一段物理内存，只有进程空间的各段的内容要发生变化时，才会将父进程的内容复制一份给子进程。

4) RDB 保存的文件

在 redis.conf 中配置文件名称，默认为 dump.rdb

```
# The filename where to dump the DB
dbfilename dump.rdb
```

5) RDB 文件的保存路径

默认为 Redis 启动时命令行所在的目录下,也可以修改

```
# The working directory.
#
# The DB will be written inside this directory, with the filename
# above using the 'dbfilename' configuration directive.
#
# The Append Only File will also be created inside this directory
#
# Note that you must specify a directory here, not a file name.
dir ./
```

6) RDB 的保存策略

```
#   In the example below the behaviour will be to save:
#   after 900 sec (15 min) if at least 1 key changed
#   after 300 sec (5 min) if at least 10 keys changed
#   after 60 sec if at least 10000 keys changed
#
```

```
save 900 1
save 300 10
save 60 10000
```

7) 手动保存快照

save: 只管保存，其它不管，全部阻塞

bgsave:按照保存策略自动保存

8) RDB 的相关配置

- stop-writes-on-bgsave-error yes

  当 Redis 无法写入磁盘的话，直接关掉 Redis 的写操作

- rdbcompression yes

  进行 rdb 保存时，将文件压缩

- rdbchecksum yes

  在存储快照后，还可以让 Redis 使用 CRC64 算法来进行数据校验，但是这样做会增加大

约 10%的性能消耗，如果希望获取到最大的性能提升，可以关闭此功能

9) RDB 的备份 与恢复

- 备份:先通过 config get dir 查询 rdb 文件的目录 ， 将*.rdb 的文件拷贝到别的地方
- 恢复: 关闭 Redis，把备份的文件拷贝到工作目录下,启动 redis,备份数据会直接加载。

10) RDB 的优缺点

- 优点: 节省磁盘空间,恢复速度快.
- 缺点: 虽然 Redis 在 fork 时使用了写时拷贝技术,但是如果数据庞大时还是比较消耗性能。在备份周期在一定间隔时间做一次备份，所以如果 Redis 意外 down 掉的话，就会丢失最后一次快照后的所有修改

7.2 AOF

1) 以日志的形式来记录每个写操作，将 Redis 执行过的所有写指令记录下来(读操作不记录)，只许追加文件但不可以改写文件，Redis 启动之初会读取该文件重新构建数据，换言之，Redis 重启的话就根据日志文件的内容将写指令从前到后执行一次以完成数据的恢复工作。

2) AOF 默认不开启，需要手动在配置文件中配置

```
# AOF and RDB persistence can be enabled at the same time without problems.
# If the AOF is enabled on startup Redis will load the AOF, that is the file
# with the better durability guarantees.
#
# Please check http://redis.io/topics/persistence for more information.

appendonly no
```

3) 可以在 redis.conf 中配置文件名称，默认为 appendonly.aof

```
# The name of the append only file (default: "appendonly.aof")

appendfilename "appendonly.aof"
```

AOF 文件的保存路径，同 RDB 的路径一致

4) AOF 和 RDB 同时开启，redis 听谁的？

5) AOF 文件故障备份

AOF 的备份机制和性能虽然和 RDB 不同, 但是备份和恢复的操作同 RDB 一样，都是拷贝备份文件，需要恢复时再拷贝到 Redis 工作目录下，启动系统即加载

6) AOF 文件故障恢复

如遇到 AOF 文件损坏，可通过

redis-check-aof --fix appendonly.aof 进行恢复

7) AOF 同步频率设置

```
#
# If unsure, use "everysec".

# appendfsync always
appendfsync everysec
# appendfsync no
```

- 始终同步，每次 Redis 的写入都会立刻记入日志
- 每秒同步，每秒记入日志一次，如果宕机，本秒的数据可能丢失。
- 把不主动进行同步，把同步时机交给操作系统。

8) Rewrite

- AOF 采用文件追加方式，文件会越来越大为避免出现此种情况，新增了重写机制,当 AOF 文件的大小超过所设定的阈值时，Redis 就会启动 AOF 文件的内容压缩，只保留可以恢复数据的最小指令集.可以使用命令 bgrewriteaof。
- Redis 如何实现重写
  AOF 文件持续增长而过大时，会 fork 出一条新进程来将文件重写(也是先写临时文件最后再 rename)，遍历新进程的内存中数据，每条记录有一条的 Set 语句。重写 aof 文件的操作，并没有读取旧的 aof 文件，而是将整个内存中的数据库内容用命令的方式重写了一个新的 aof 文件，这点和快照有点类似。
- 何时重写
  重写虽然可以节约大量磁盘空间，减少恢复时间。但是每次重写还是有一定的负担的，因此设定 Redis 要满足一定条件才会进行重写。

```
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
```

  系统载入时或者上次重写完毕时，Redis 会记录此时 AOF 大小，设为 base_size,如果 Redis 的 AOF 当前大小>= base_size +base_size*100% (默认)且当前大小>=64mb(默认)的情况下，Redis 会对 AOF 进行重写。

9) AOF 的优缺点

- 优点:
  备份机制更稳健，丢失数据概率更低。
  可读的日志文本，通过操作 AOF 稳健，可以处理误操作。
- 缺点:
  比起 RDB 占用更多的磁盘空间
  恢复备份速度要慢
  每次读写都同步的话，有一定的性能压力。

7.3 RDB 和 AOF 用哪个好

- 官方推荐两个都启用。
- 如果对数据不敏感，可以选单独用 RDB
- 不建议单独用 AOF，因为可能会出现 Bug。
- 如果只是做纯内存缓存，可以都不用

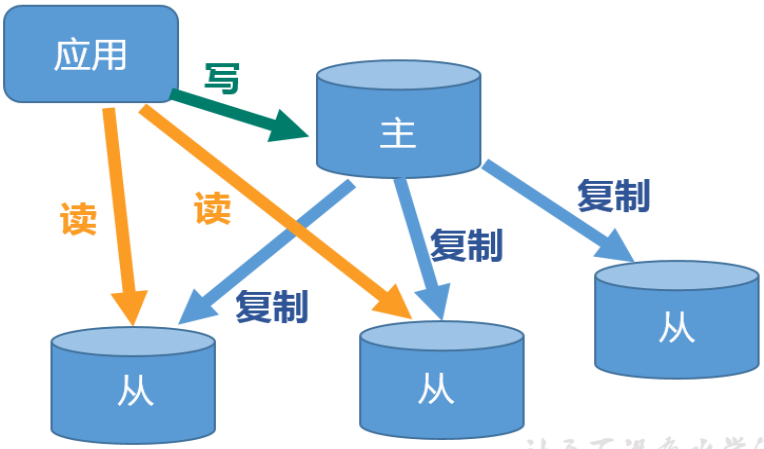
第 8 章 Redis 主从复制

8.1 什么是主从复制

主从复制，就是主机数据更新后根据配置和策略，自动同步到备机的 master/slaver 机制，Master 以写为主，Slave 以读为主。

8.2 主从复制的目的

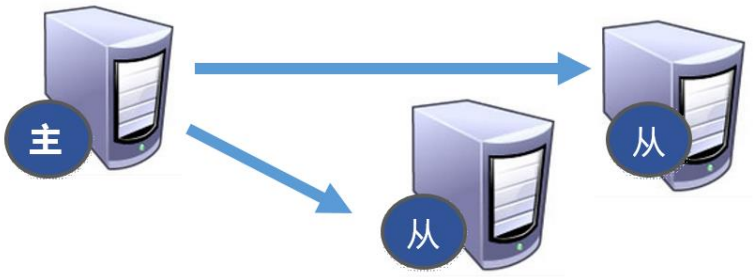
1) 读写分离，性能扩展
2) 容灾快速恢复
3)



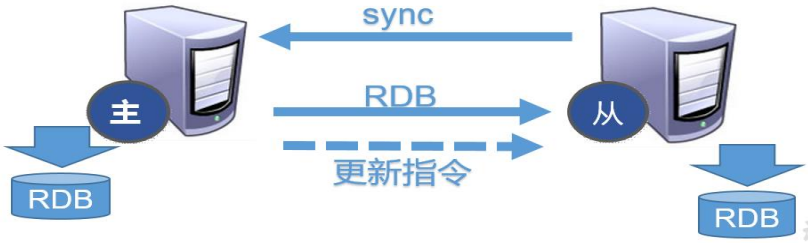
8.3 主从配置

1) 原则: 配从不配主
2) 步骤: 准备三个 Redis 实例，一主两从
   拷贝多个 redis.conf 文件 include
   开启 daemonize yes
   Pid 文件名字 pidfile
   指定端口 port
   Log 文件名字
   Dump.rdb 名字 dbfilename
   Appendonly 关掉或者换名字
3) info replication 打印主从复制的相关信息
4) slaveof <ip> <port> 成为某个实例的从服务器

8.4 一主二从模式演示



1) 相关问题:
   - 切入点问题？slave1、slave2 是从头开始复制还是从切入点开始复制?比如从 k4 进来，那之前的 123 是否也可以复制
   - 从机是否可以写？set 可否？
   - 主机 shutdown 后情况如何？从机是上位还是原地待命
   - 主机又回来了后，主机新增记录，从机还能否顺利复制
   - 其中一台从机 down 后情况如何？依照原有它能跟上大部队吗？
2) 复制原理
   - 每次从机联通后，都会给主机发送 sync 指令
   - 主机立刻进行存盘操作，发送 RDB 文件，给从机
   - 从机收到 RDB 文件后，进行全盘加载
   - 之后每次主机的写操作，都会立刻发送给从机，从机执行相同的命令



8.5 薪火相传模式演示

1) 上一个 slave 可以是下一个 slave 的 Master，slave 同样可以接收其他 slaves 的连接和同步请求，那么该 slave 作为了链条中下一个的 master, 可以有效减轻 master 的写压力,去中心化降低风险.
   中途变更转向:会清除之前的数据，重新建立拷贝最新的
   风险是一旦某个 slave 宕机，后面的 slave 都没法备份

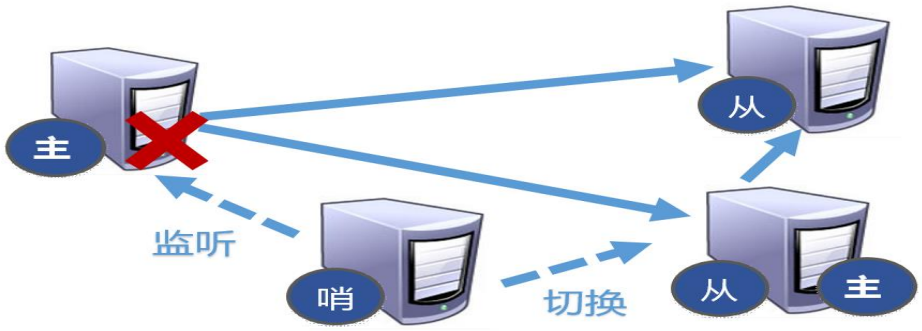
2) 反客为主(小弟上位)

当一个 master 宕机后，后面的 slave 可以立刻升为 master，其后面的 slave 不用做任何修改。

用 slaveof no one 将从机变为主机。

3) 哨兵模式 sentinel (推举大哥)

反客为主的自动版，能够后台监控主机是否故障，如果故障了根据投票数自动将从库转换为主库.



配置哨兵

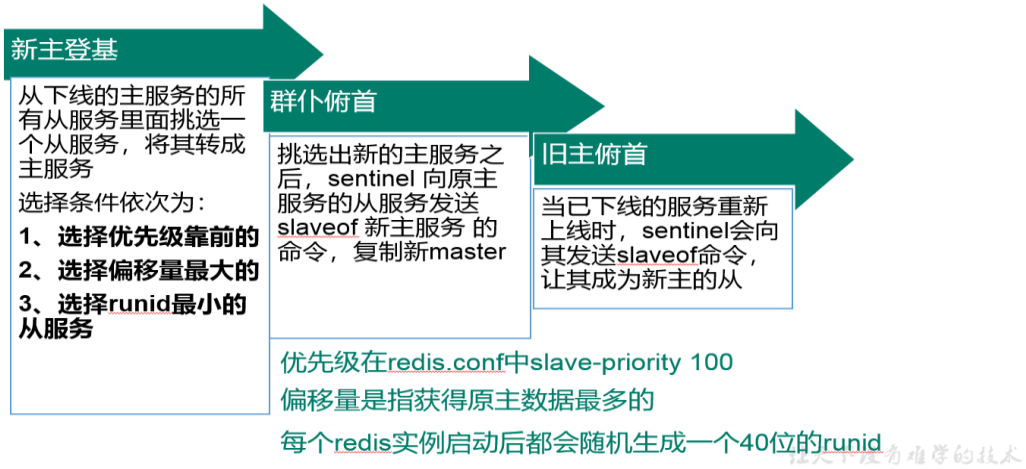
- 调整为一主二从模式
- 自定义的/myredis 目录下新建 sentinel.conf 文件
- 在配置文件中填写内容

  sentinel monitor mymaster 127.0.0.1 6379 1

  其中 mymaster 为监控对象起的服务器名称， 1 为 至少有多少个哨兵同意迁移的数量。
- 启动哨兵

  执行 redis-sentinel /myredis/sentinel.conf

8.6 故障恢复



第 9 章 Redis 集群

9.1 问题

1) 容量不够，redis 如何进行扩容？
2) 并发写操作， redis 如何分摊？

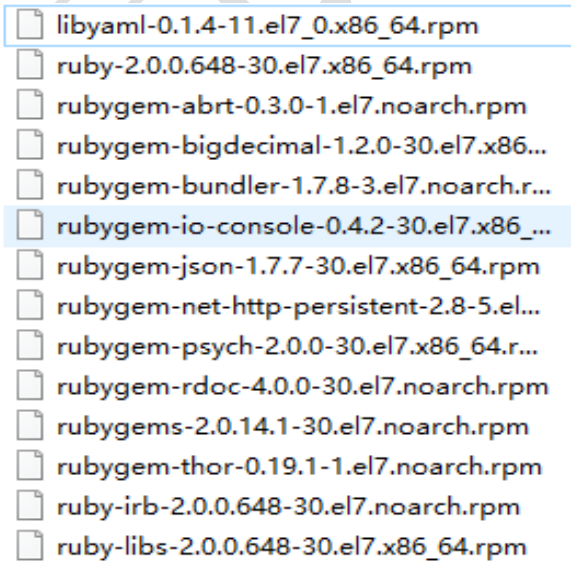
9.1 什么是集群

1) Redis 集群实现了对 Redis 的水平扩容，即启动 N 个 redis 节点，将整个数据库分布存储在这 N 个节点中，每个节点存储总数据的 1/N。
2) Redis 集群通过分区（partition）来提供一定程度的可用性（availability）： 即使集群中有一部分节点失效或者无法进行通讯， 集群也可以继续处理命令请求

9.2 集群方案

9.3 安装 ruby 环境

1) 能上网:
   执行 yum install ruby
   执行 yum install rubygems
2) 不能上网:
   - cd /run/media/root/CentOS 7 x86_64/Packages 获取如下 rpm 包



   - 拷贝到/opt/rpmruby/目录下，并 cd 到此目录

- 执行：rpm -Uvh *.rpm --nodeps –force 按照依赖安装各个 rpm 包
- 按照依赖安装各个 rpm 包
- 执行在 opt 目录下执行 gem install --local redis-3.2.0.gem

9.4 准备 6 个 Redis 实例

1) 准备 6 个实例 6379,6380,6381,6389,6390,6391
   拷贝多个 redis.conf 文件
   开启 daemonize yes
   Pid 文件名字
   指定端口
   Log 文件名字
   Dump.rdb 名字
   Appendonly 关掉或者换名字
2) 再加入如下配置
   cluster-enabled yes 打开集群模式
   cluster-config-file nodes-端口号.conf 设定节点配置文件名
   cluster-node-timeout 15000 设定节点失联时间，超过该时间（毫秒），集群自动进行主从切换

9.5 合体

1) 将 6 个实例全部启动，nodes-端口号.conf 文件都生成正常
2) 合体
   - 进入到 cd /opt/redis-3.2.5/src
   - 执行
     ./redis-trib.rb create --replicas 1
     192.168.31.211:6379 192.168.31.211:6380 192.168.31.211:6381
     192.168.31.211:6389 192.168.31.211:6390 192.168.31.211:6391
   - 注意: IP 地址修改为当前服务器的地址，端口号为每个 Redis 实例对应的端口号.

9.6 集群操作

1) 以集群的方式进入客户端
   redis-cli -c -p 端口号
2) 通过 cluster nodes 命令查看集群信息

```
127.0.0.1:6379> cluster nodes
8239a824b6921de2ef7d121f4ec3665d40e622fd 127.0.0.1:6389 slave 6f31531d29d9909879fe422c557d3e0099f1d954 0 1477304189478 4 connected
2659bc05326a4360ebd381bc0b3d1fcc890b3365 127.0.0.1:6391 slave f8f56dc13fd04bb45d117f8654875d477c990f07 0 1477304190488 6 connected
1b49db8f826ceeb641d3e7e97f16d988ad292808 127.0.0.1:6390 slave 79e2edab30c626dc81de529783bdf3e3f6ae80e0 0 1477304191499 5 connected
6f31531d29d9909879fe422c557d3e0099f1d954 127.0.0.1:6379 myself,master - 0 0 1 connected 0-5460
79e2edab30c626dc81de529783bdf3e3f6ae80e0 127.0.0.1:6380 master - 0 1477304192509 2 connected 5461-10922
f8f56dc13fd04bb45d117f8654875d477c990f07 127.0.0.1:6381 master - 0 1477304192509 3 connected 10923-16383
```

3) redis cluster 如何分配这六个节点

一个集群至少要有三个主节点。

选项 --replicas 1 表示我们希望为集群中的每个主节点创建一个从节点。

分配原则尽量保证每个主数据库运行在不同的 IP 地址，每个从库和主库不在一个 IP 地址上。

4) 什么是 slots

- 一个 Redis 集群包含 16384 个插槽（hash slot）， 数据库中的每个键都属于这 16384 个插槽的其中一个， 集群使用公式 CRC16(key) % 16384 来计算键 key 属于哪个槽， 其中 CRC16(key) 语句用于计算键 key 的 CRC16 校验和 。
- 集群中的每个节点负责处理一部分插槽。举个例子，如果一个集群可以有主节点，其中：

  节点 A 负责处理 0 号至 5500 号插槽。

  节点 B 负责处理 5501 号至 11000 号插槽。

  节点 C 负责处理 11001 号至 16383 号插槽

5) 在集群中录入值

- 在 redis-cli 每次录入、查询键值，redis 都会计算出该 key 应该送往的插槽，如果不是该客户端对应服务器的插槽，redis 会报错，并告知应前往的 redis 实例地址和端口.
- redis-cli 客户端提供了 –c 参数实现自动重定向。

如 redis-cli -c –p 6379 登入后，再录入、查询键值对可以自动重定向。

- 不在一个 slot 下的键值，是不能使用 mget,mset 等多键操作。
- 可以通过{}来定义组的概念，从而使 key 中{}内相同内容的键值对放到一个 slot 中去

6) 查询集群中的值

- CLUSTER KEYSLOT <key> 计算键 key 应该被放置在哪个槽上。
- CLUSTER COUNTKEYSINSLOT <slot> 返回槽 slot 目前包含的键值对数量
- CLUSTER GETKEYSINSLOT <slot> <count> 返回 count 个 slot 槽中的键

7) 故障恢复

- 如果主节点下线？从节点能否自动升为主节点？
- 主节点恢复后，主从关系会如何？
- 如果所有某一段插槽的主从节点都当掉，redis 服务是否还能继续?

  redis.conf 中的参数 cluster-require-full-coverage

9.7 集群的 Jedis 开发

```
public class JedisClusterTest {
  public static void main(String[] args) {

    Set<HostAndPort> set =new HashSet<HostAndPort>();
    set.add(new HostAndPort("192.168.31.211",6379));
    JedisCluster jedisCluster=new JedisCluster(set);
```

```
        jedisCluster.set("k1", "v1");
        System.out.println(jedisCluster.get("k1"));
    }
}
```

9.8 Redis 集群的优缺点

- 优点
  实现扩容
  分摊压力
  无中心配置相对简单
- 缺点
  多键操作是不被支持的
  多键的 Redis 事务是不被支持的。lua 脚本不被支持。
  由于集群方案出现较晚，很多公司已经采用了其他的集群方案，而代理或者客户端分片的方案想要迁移至 redis cluster，需要整体迁移而不是逐步过渡，复杂度较大。