Redis

1、NoSQL数据库简介

2、Redis的介绍及安装启动

3、Redis的五大数据类型

4、Redis的相关配置

5、Redis的Java客户端Jedis

6、Redis的事务

7、Redis的持久化

8、Redis的主从复制

9、Redis的集群

NoSQL数据库简介

技术的分类

- 解决功能性的问题

Java、Jsp、 RDBMS
Tomcat、HTML、Linux、
Jdbc、SVN

- 解决扩展性的问题

Struts、Spring、SpringMVC、
Hibernate、Mybatis

- 解决性能的问题

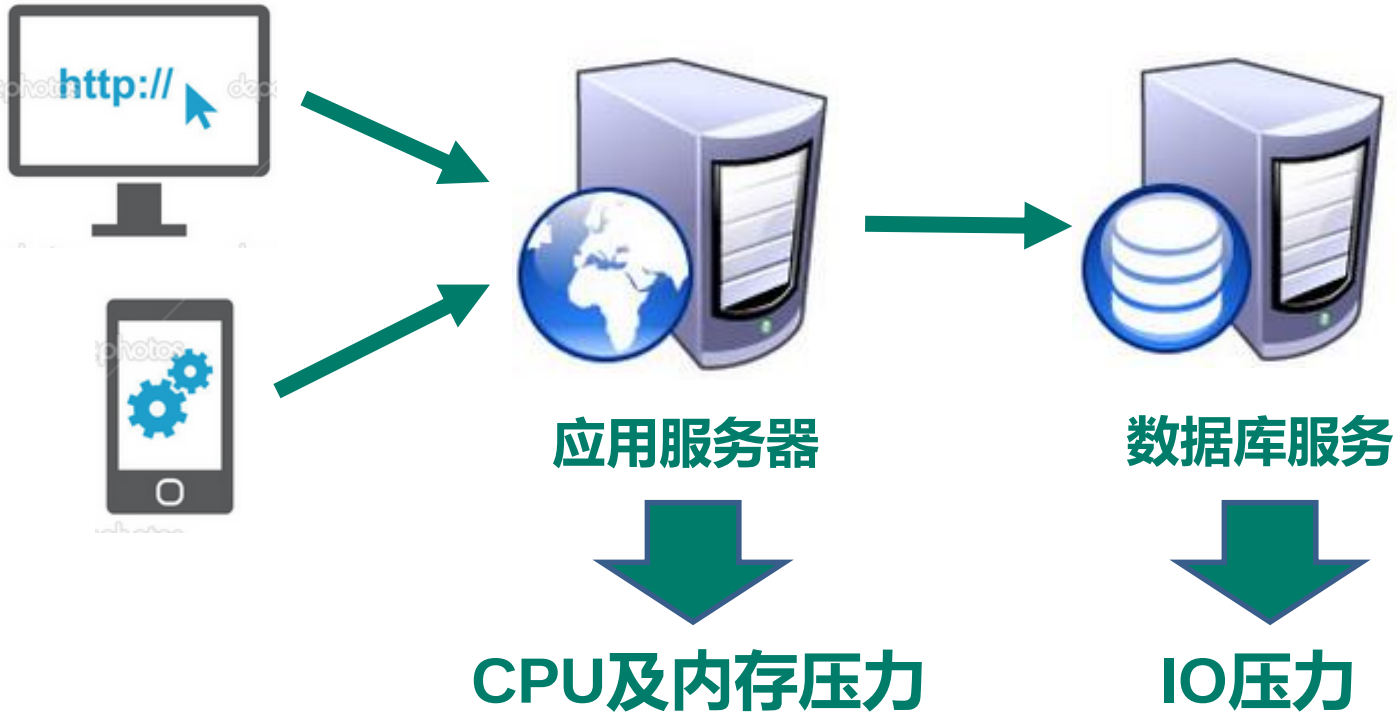
NoSQL、Java线程、Hadoop、
Nginx(负载均衡)、MQ(消息队列)、
ElasticSearch

NoSQL数据库简介

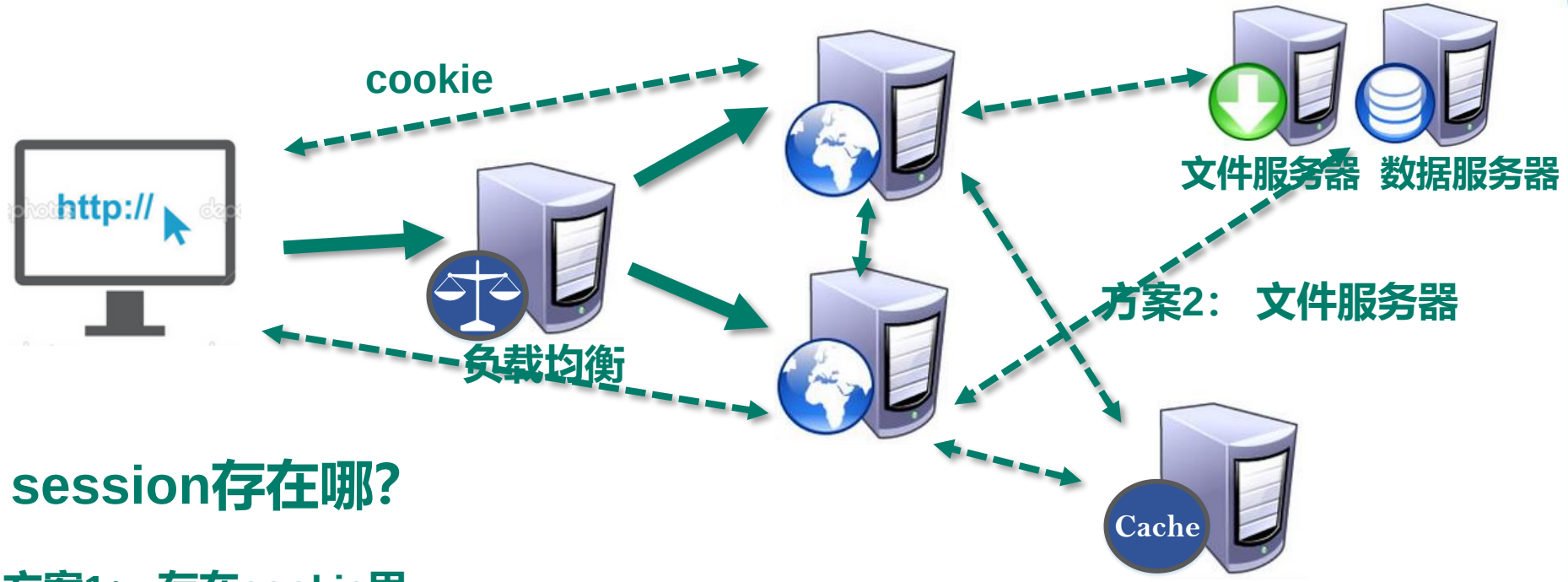
Web1.0的时代，数据访问量很有限，用一夫当关的高性能的单点服务器可以解决大部分问题。

NoSQL数据库简介

随着Web2.0的时代的到来，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。



解决应服务器的cpu和内存压力



session存在哪?

方案1： 存在cookie里

1、不安全

2、网络负担效率低

方案2： 存在文件服务器或者数据库里

1、大量的IO效率问题

方案3：session复制

session数据冗余

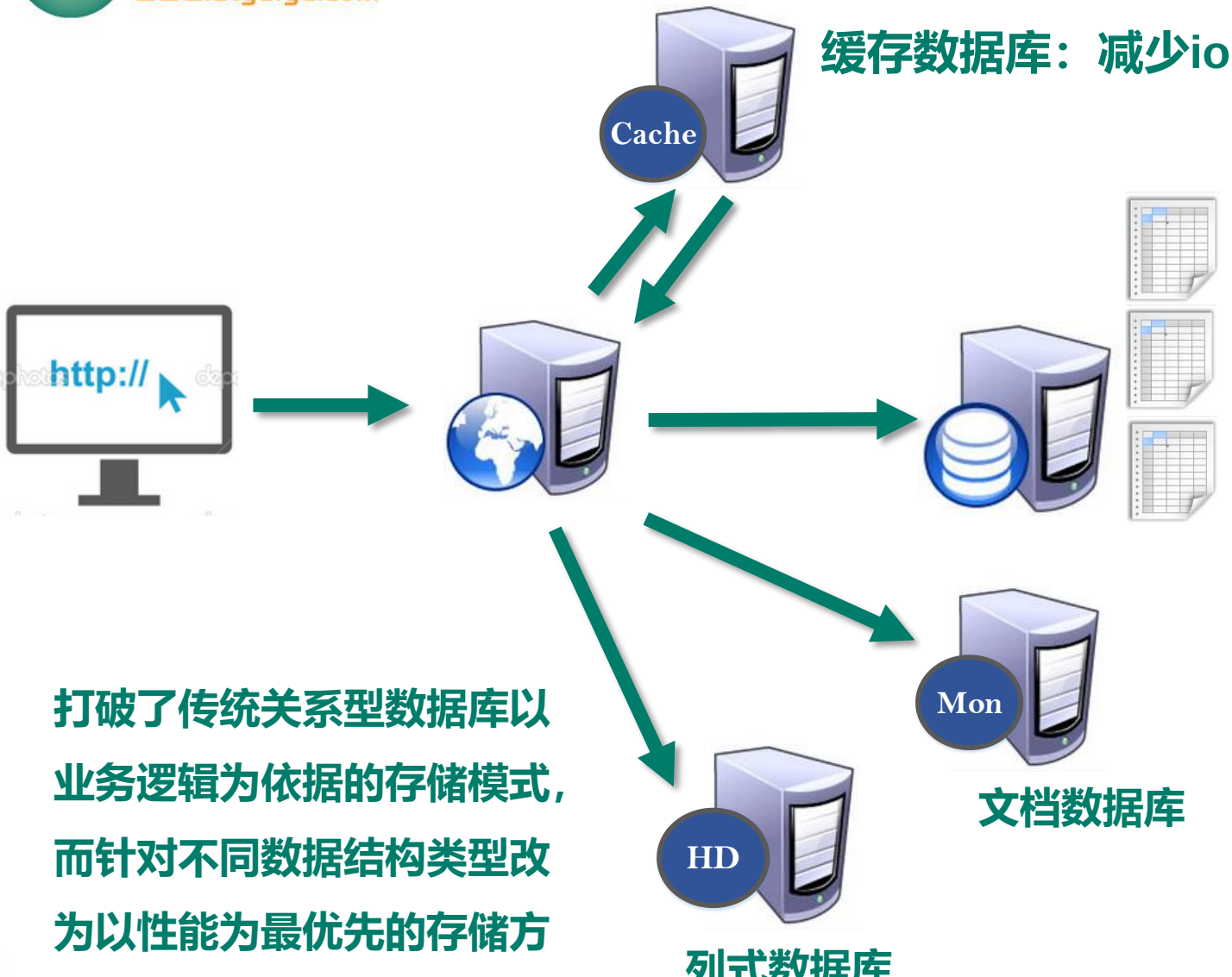
节点越多浪费越大

方案4： 缓存数据库

完全在内存中,速度快

数据结构简单

解决IO的压力



缓存数据库：减少io的读操作

水平切分

垂直切分

读写分离

通过破坏一定的业务逻辑来换取性能

文档数据库

列式数据库

打破了传统关系型数据库以业务逻辑为依据的存储模式，而针对不同数据结构类型改为以性能为最优先的存储方式。

NoSQL数据库概述

- 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。**
- 不遵循SQL标准。
- 不支持ACID。
- 远超于SQL的性能。

NoSQL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

NoSQL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（用户自定义查询条件的查询）。

用不着sql的和用了sql也不行的情况，请考虑用NoSql

NoSQL兄弟会—缓存数据库

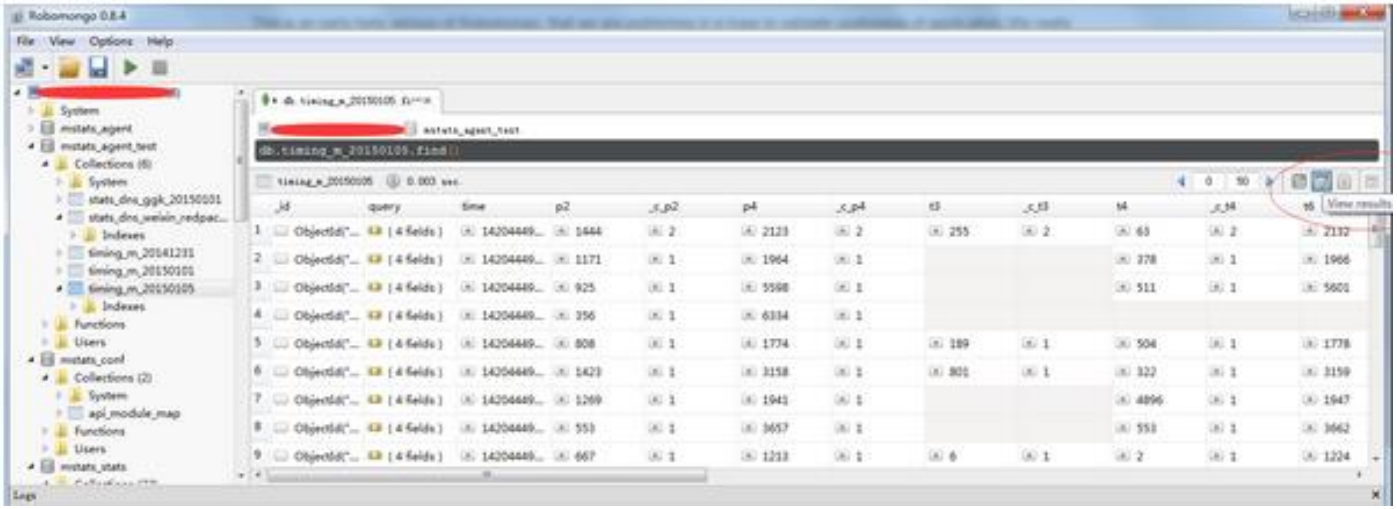
- Memcached
  - 很早出现的NoSql数据库
  - 数据都在内存中，一般不持久化
  - 支持简单的key-value模式
  - 一般是作为缓存数据库辅助持久化的数据库

- **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

NoSQL兄弟会—文档数据库

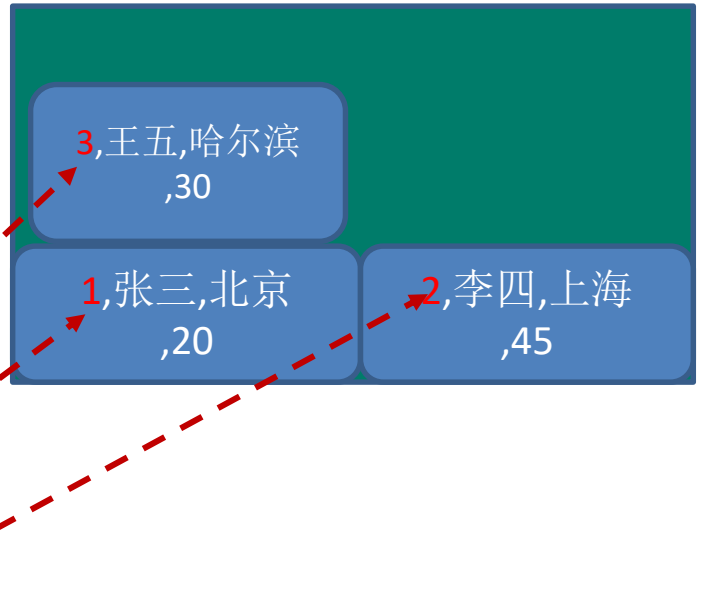
➢ mongoDB

- 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。



- 先看什么是行式存储数据库

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



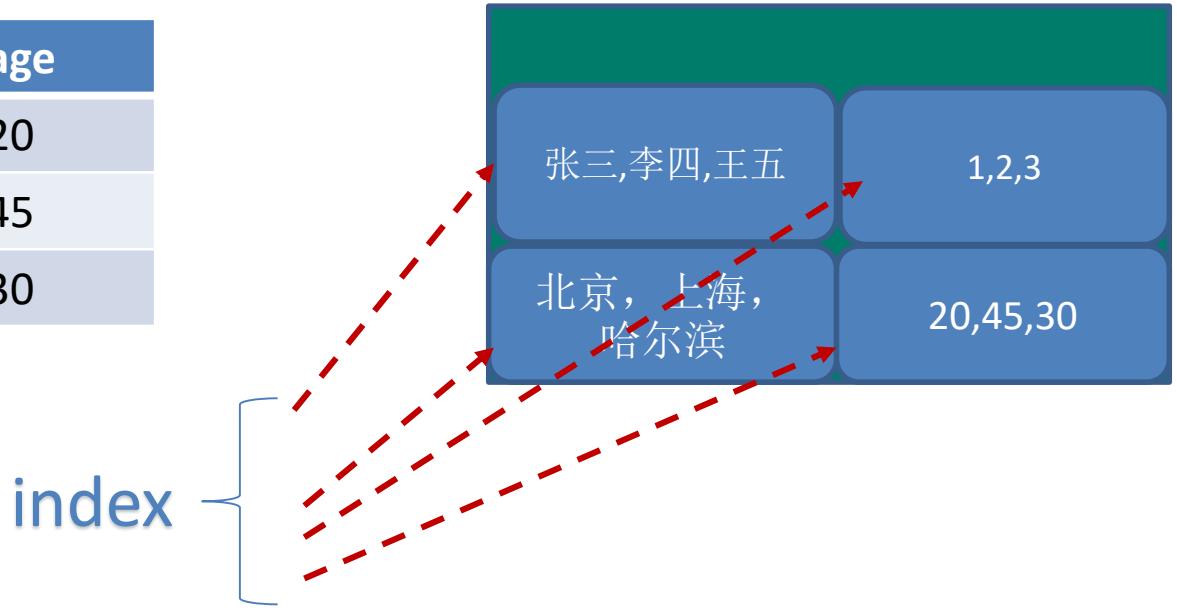
姓名: 王五
年龄: 30
城市: 哈尔滨

select * from users where id =3 快

select avg(age) from users 慢

- 列式存储数据库

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



查询年龄的平均值　　快　　OLAP分析型处理

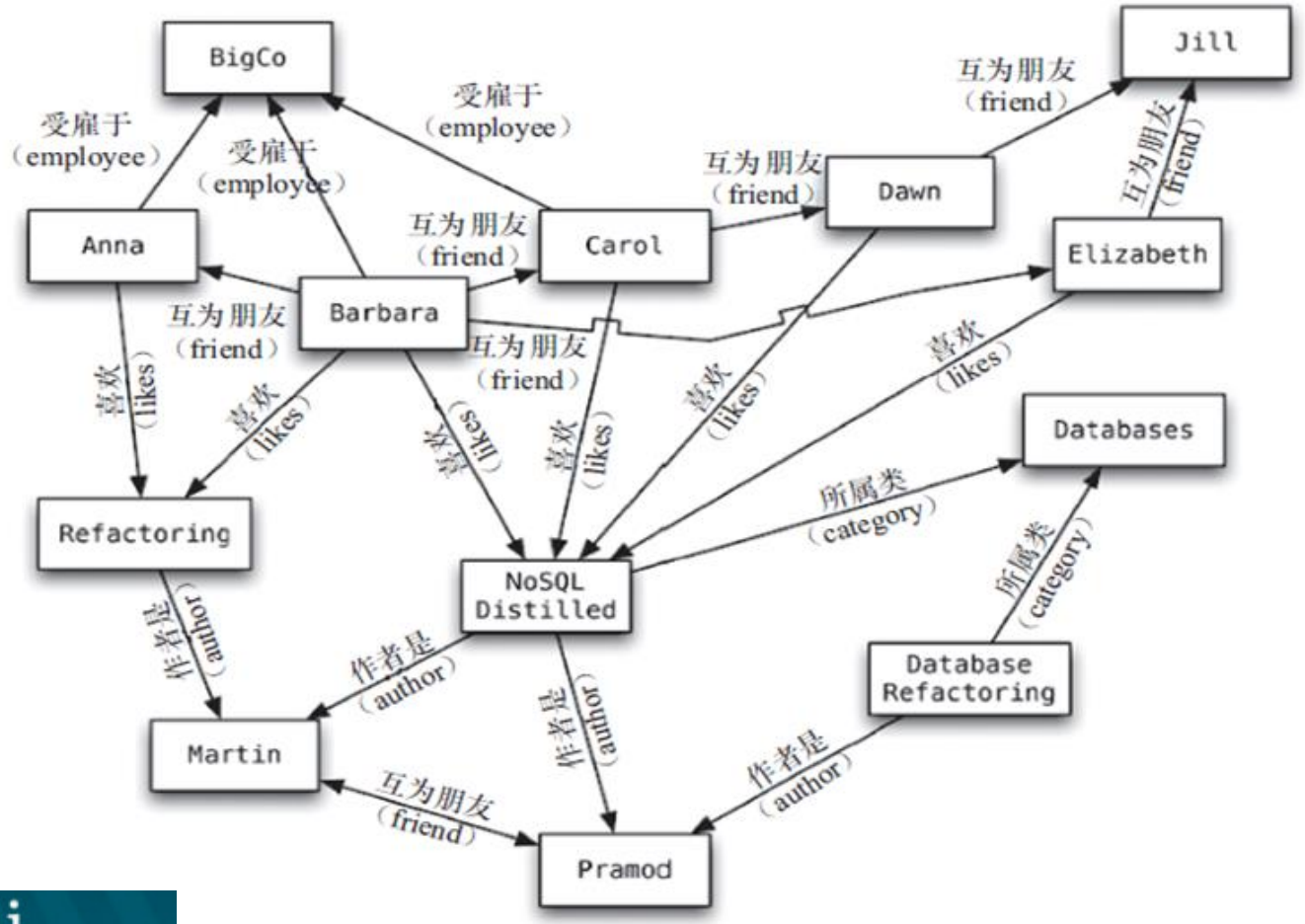
查询id为3的人员信息　　慢　　OLTP事务型处理

NoSQL兄弟会—列式数据库

- 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。

- 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。

NoSQL兄弟会—图关系数据库



Neo4j the graph database

主要应用：社会关系，公共交通网络，地图及网络拓谱

DB-Engines 数据库排名

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

334 systems in ranking, September 2017

Rank			DBMS	Database Model	Score		
Sep 2017	Aug 2017	Sep 2016			Sep 2017	Aug 2017	Sep 2016
1.	1.	1.	Oracle	Relational DBMS	1359.09	-8.78	-66.47
2.	2.	2.	MySQL	Relational DBMS	1312.61	-27.69	-41.41
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1212.54	-12.93	+0.99
4.	4.	4.	PostgreSQL	Relational DBMS	372.36	+2.60	+56.01
5.	5.	5.	MongoDB	Document store	332.73	+2.24	+16.74
6.	6.	6.	DB2	Relational DBMS	198.34	+0.87	+17.15
7.	7.	↑ 8.	Microsoft Access	Relational DBMS	128.81	+1.78	+5.50
8.	8.	↓ 7.	Cassandra	Wide column store	126.20	-0.52	-4.29
9.	9.	↑ 10.	Redis	Key-value store	120.41	-1.49	+12.61
10.	10.	↑ 11.	Elasticsearch	Search engine	120.00	+2.35	+23.52
11.	11.	↓ 9.	SQLite	Relational DBMS	112.04	+1.19	+3.41
12.	12.	12.	Teradata	Relational DBMS	80.91	+1.67	+7.84
13.	13.	↑ 14.	Solr	Search engine	69.91	+2.95	+2.95
14.	14.	↓ 13.	SAP Adaptive Server	Relational DBMS	66.75	-0.16	-2.41
15.	15.	15.	HBase	Wide column store	64.34	+0.82	+6.53
16.	16.	↑ 17.	Splunk	Search engine	62.57	+1.11	+11.28
17.	17.	↓ 16.	FileMaker	Relational DBMS	61.00	+1.35	+5.64
18.	18.	↑ 20.	MariaDB	Relational DBMS	55.47	+0.78	+16.94
19.	↑ 20.	↓ 18.	Hive	Relational DBMS	48.62	+1.31	-0.21
20.	↓ 19.	↓ 19.	SAP HANA	Relational DBMS	48.33	+0.36	+4.91
21.	21.	21.	Neo4j	Graph DBMS	38.42	+0.42	+2.06
22.	22.	↑ 25.	Amazon DynamoDB	Document store	37.82	+0.20	+10.40
23.	23.	↓ 22.	Couchbase	Document store	33.11	+0.14	+4.57
24.	24.	↓ 23.	Memcached	Key-value store	28.94	-1.02	+0.51

Redis的介绍及安装

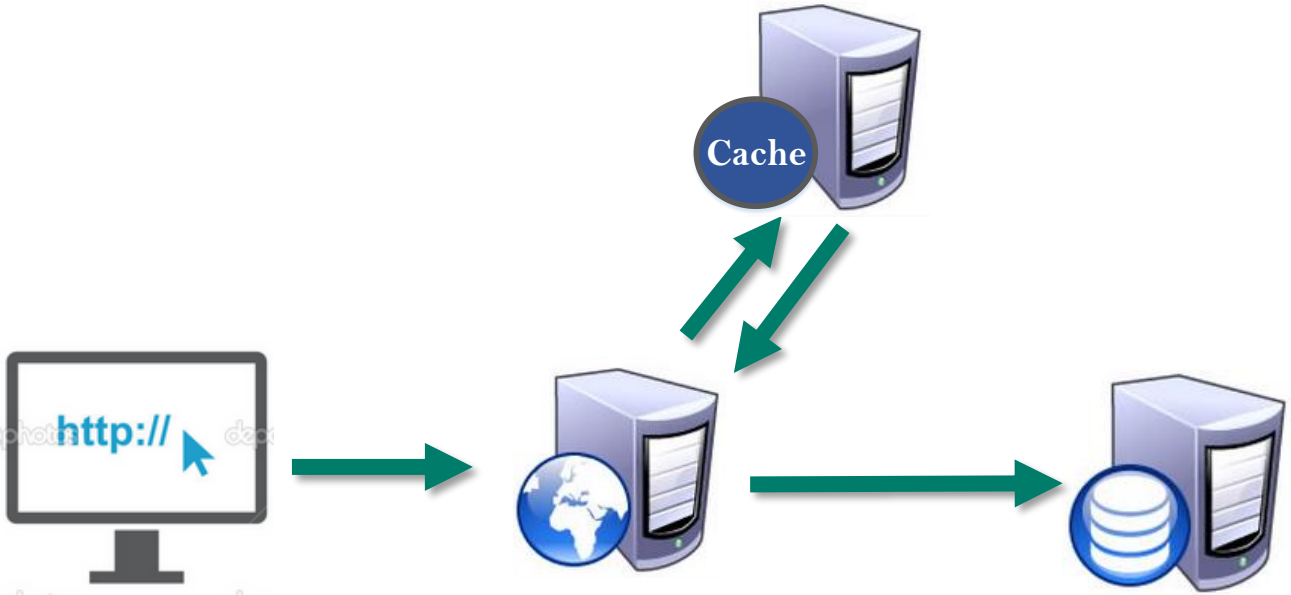
Redis介绍

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。

Redis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
- 分布式架构，做session共享



- 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实现按自然时间排序的数据
排行榜，Top N，	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

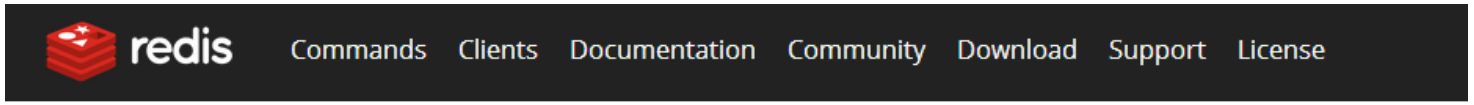
Redis从哪获得

Redis官方网站

http://Redis.io

Redis中文官方网站

http://www.Redis.net.cn/



Redis is an open source (BSD licensed), in-memory **data structure store**, used as database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

Learn more →

Try it

Ready for a test drive? Check this interactive tutorial that will walk you through the most important features of Redis.

Download it

Redis 3.2.3 is the latest stable version. Interested in release candidates or unstable versions? Check the downloads page.

Quick links

Follow day-to-day Redis on Twitter and GitHub. Get help or help others by subscribing to our mailing list, we are 5,000 and counting!

Redis安装

关于安装版本：

3.2.5 for Linux

不用考虑在windows环境下对Redis的支持：

Windows

The Redis project does not officially support Windows. However, the Microsoft Open Tech group develops and maintains this Windows port targeting Win64. Learn more

Redis安装

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命令时出现故障意出现的错误解析：　　gcc：命令未找到

- 能上网：**yum install gcc-c++**
- 不能上网：
  - 执行 cd /media/CentOS_6.8_Final/Packages （路径跟centos5不同） 进入安装包目录
  - 依次执行以下：

    rpm -ivh mpfr-2.4.1-6.el6.x86_64.rpm

    rpm -ivh cpp-4.4.7-17.el6.x86_64.rpm

    rpm -ivh ppl-0.10.2-11.el6.x86_64.rpm

    rpm -ivh cloog-ppl-0.15.7-1.2.el6.x86_64.rpm

    rpm -ivh gcc-4.4.7-17.el6.x86_64.rpm

Redis安装

5、在redis-3.2.5目录下再次执行make命令

Jemalloc/jemalloc.h：没有那个文件

解决方案：运行make distclean之后再 make

6、在redis-3.2.5目录下再次执行make命令

```
    CC redis-check-rdb.o
    CC geo.o
    LINK redis-server
    INSTALL redis-sentinel
    CC redis-cli.o
    LINK redis-cli
    CC redis-benchmark.o
    LINK redis-benchmark
    INSTALL redis-check-rdb
    CC redis-check-aof.o
    LINK redis-check-aof

Hint: It's a good idea to run 'make test' ;)

make[1]: Leaving directory `/opt/redis-3.2.0/src'
[root@jack redis-3.2.0]#
```

Redis Test(可以不用执行)

Redis安装

执行完make后，跳过Redis test 继续执行**make install**

```
[root@jack redis-3.2.0]# make install
cd src && make install
make[1]: Entering directory `/opt/redis-3.2.0/src'

Hint: It's a good idea to run 'make test' ;)

    INSTALL install
    INSTALL install
    INSTALL install
    INSTALL install
    INSTALL install
make[1]: Leaving directory `/opt/redis-3.2.0/src'
[root@jack redis-3.2.0]#
```

Redis目录

查看默认安装目录：usr/local/bin

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

Redis启动

- 启动

1、备份redis.conf：拷贝一份redis.conf到其他目录

2、修改**redis.conf**文件将里面的**daemonize no** 改成 **yes**，让服务在后台启动

3、启动命令：执行 **redis-server /myredis/redis.conf**

Redis启动

4、用客户端访问: Redis-cli

```
[root@jack myredis]# redis-cli
127.0.0.1:6379>
```

- 多个端口可以 Redis-cli –p 6379

5、测试验证： ping

```
127.0.0.1:6379> ping
PONG
127.0.0.1:6379>
```

Redis关闭

- 单实例关闭：**Redis-cli shutdown**

```
root@jack myredis]# redis-server  /myredis/redis.conf
root@jack myredis]# redis-cli shutdown
root@jack myredis]# ps -ef|grep redis
root       6938   3571  0 17:32 pts/0    00:00:00 grep redis
```

- 也可以进入终端后再关闭

```
127.0.0.1:6379> shutdown
not connected>
```

多实例关闭，指定端口关闭:Redis-cli -p 6379 shutdown

Redis相关知识

- 端口6379从何而来

  Alessia **Merz**

- 默认16个数据库，类似数组下标从0开始，初始默认使用0号库

  使用命令 **select <dbid>** 来切换数据库。如: select 8

- 统一密码管理，所有库都是同样密码，要么都OK要么一个也连接不上。

Redis相关知识—单线程

Redis是单线程+多路IO复用技术

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）。

串行 vs 多线程+锁（memcached） vs 单线程+多路IO复用(Redis)

Redis数据类型

- key +
  - string
  - set
  - list
  - hash
  - zset

Redis 五大数据类型 --Key

- keys *
  - 查询当前库的所有键
- 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

Redis 五大数据类型 --Key

- **expire <key> <seconds>**
  - 为键值设置过期时间，单位秒。

- **ttl <key>**
  - 查看还有多少秒过期，-1表示永不过期，-2表示已过期

Redis 五大数据类型 --Key

- **dbsize**
  - 查看当前数据库的**key**的数量
- **flushdb**
  - 清空当前库
- **flushall**
  - 通杀全部库

Redis 五大数据类型 --String

String

- 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。
- 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 。
- 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

Redis 五大数据类型 --String

- 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>
  - 获得值的长度
- setnx <key> <value>
  - 只有在 key 不存在时设置 key 的值

Redis 五大数据类型 --String

- incr <key>
  - 将 key 中储存的数字值增1
  - 只能对数字值操作，如果为空，新增值为1
- decr <key>
  - 将 key 中储存的数字值减1
  - 只能对数字值操作，如果为空，新增值为-1
- incrby / decrby <key> <步长>
  - 将 key 中储存的数字值增减。自定义步长。

Redis 五大数据类型 --String

➢ 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）。

（1） 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。
（2）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

java中的i++是否是原子操作？

Redis 五大数据类型 –string

- **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 key 都不存在。

Redis 五大数据类型 --String

- getrange <key> <起始位置> <结束位置>
  - 获得值的范围，类似java中的substring
  - 包前包后 双引号不算数

- setrange <key> <起始位置> <value>
  - 用 <value> 覆写<key> 所储存的字符串值，从<起始位置>开始。

Redis 五大数据类型 --String

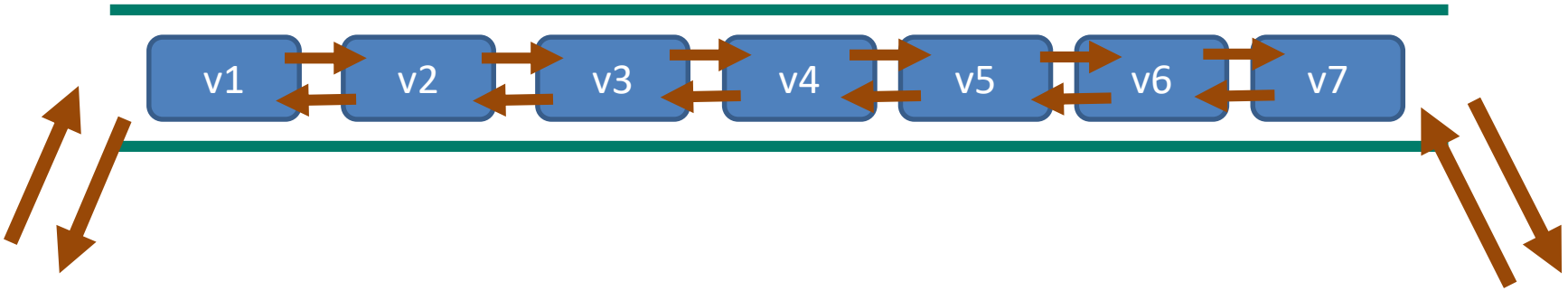
- setex <key> <过期时间> <value>
  - 设置键值的同时，设置过期时间，单位秒。

- getset <key> <value>
  - 以新换旧，设置了新值同时获得旧值。

Redis 五大数据类型 --list

List

- 单键多值
- 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）。
- 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。**



Redis 五大数据类型 --list

- lpush/rpush <key> <value1> <value2> <value3> ....
  - 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。

Redis 五大数据类型 --list

- 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

Redis 五大数据类型 --list

- linsert <key> before <value> <newvalue>
  - 在<value>的前面插入<newvalue> 插入值

- lrem <key> <n> <value>
  - 从左边删除n个value(从左到右)

Redis 五大数据类型 -- set

- 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。

- 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)。

Redis 五大数据类型 -- set

- 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

Redis 五大数据类型 -- set

- 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

Redis 五大数据类型 -- set

- 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。

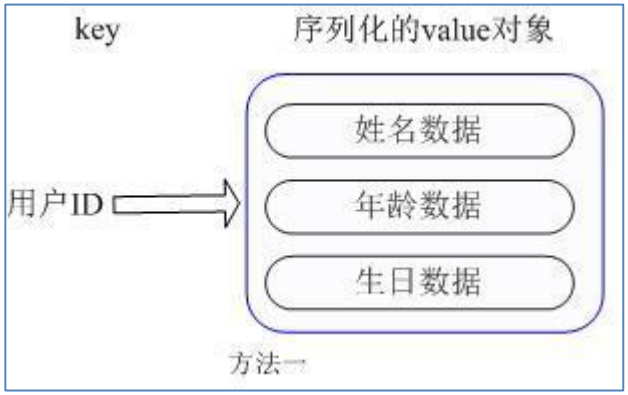
- sdiff <key1> <key2>
  - 返回两个集合的差集元素。
  - 谁在前面，就用谁减去两者的并集 。

hash

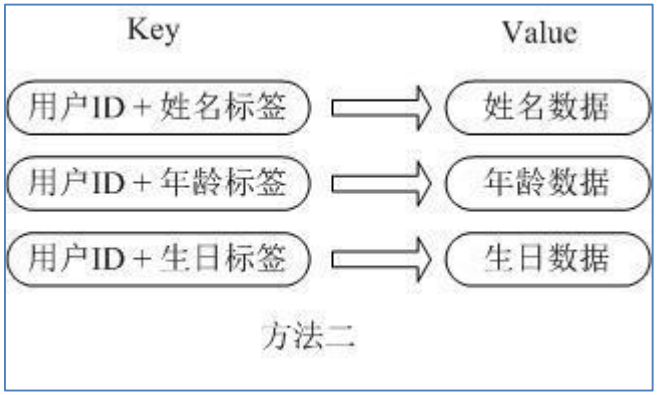
- Redis hash 是一个键值对集合。
- Redis hash是一个string类型的field和value的映射表，hash特别适合用于存储对象。
- 类似Java里面的Map<String,Object>

Redis 五大数据类型 -- hash

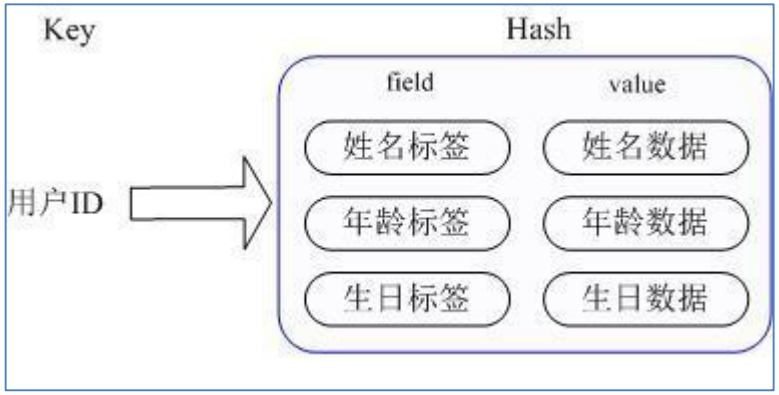
用户ID为查找的key，存储的value用户对象包含姓名，年龄，生日等信息，如果用普通的key/value结构来存储，主要有以下2种存储方式：



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。



用户ID数据冗余



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

Redis 五大数据类型 -- hash

- 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

Redis 五大数据类型 -- hash

- 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 .

zset（sorted set）

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没有成员都关联了一个**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 。

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。

Redis 五大数据类型 -- zset

- 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。
  - zrevrange 反转
- 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。

Redis 五大数据类型 -- zset

- 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。

Redis 五大数据类型 -- zset

- 如何利用zset实现一个文章访问量的排行榜？

Redis的相关配置

Redis 相关配置

- 计量单位说明

```
# Note on units: when memory size is needed, it is possible to specify
# it in the usual form of 1k 5GB 4M and so forth:
#
# 1k => 1000 bytes
# 1kb => 1024 bytes
# 1m => 1000000 bytes
# 1mb => 1024*1024 bytes
# 1g => 1000000000 bytes
# 1gb => 1024*1024*1024 bytes
#
# units are case insensitive so 1GB 1Gb 1gB are all the same.
```

- 大小写不敏感

Redis 相关配置

- include

```
################################## INCLUDES ###################################

# Include one or more other config files here.  This is useful if you
# have a standard template that goes to all Redis servers but also need
# to customize a few per-server settings.  Include files can include
# other files, so use this wisely.
#
# Notice option "include" won't be rewritten by command "CONFIG REWRITE"
# from admin or Redis Sentinel. Since Redis always uses the last processed
# line as value of a configuration directive, you'd better put includes
# at the beginning of this file to avoid overwriting config change at runtime.
#
# If instead you are interested in using includes to override configuration
# options, it is better to use include as the last line.
#
# include /path/to/local.conf
# include /path/to/other.conf
```

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

Redis 相关配置

- 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只允许接受本机的响应

Redis 相关配置

- 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

Redis 相关配置

- timeout

  一个空闲的客户端维持多少秒会关闭，0为永不关闭。

- TCP keepalive

  对访问客户端的一种心跳检测，每个n秒检测一次。

  官方推荐设为60秒。

Redis 相关配置

- daemonize

  是否为后台进程

- pidfile

  存放pid文件的位置，每个实例会产生一个不同的pid文件

Redis 相关配置

- 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

- logfile

  日志文件名称

Redis 相关配置

- syslog

  是否将Redis日志输送到linux系统日志服务中

- syslog-ident

  日志的标志

- syslog-facility

  输出日志的设备

- database

  设定库的数量 默认16

Redis 相关配置

➢ 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
```

Redis 相关配置

- maxclient
  - 最大客户端连接数
- 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。

Redis 相关配置

Maxmemory-policy

（1）volatile-lru：使用LRU算法移除key，只对设置了过期时间的键

（2）allkeys-lru：使用LRU算法移除key

（3）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

（4）allkeys-random：移除随机的key

（5）volatile-ttl：移除那些TTL值最小的key，即那些最近要过期的key

（6）noeviction：不进行移除。针对写操作，只是返回错误信息

Redis 相关配置

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。

Java的Redis客户端Jedis

Redis的Java客户端Jedis

- **Jedis所需要的jar包**
  - Commons-pool-1.6.jar
  - Jedis-2.1.0.jar

Redis的Java客户端Jedis

➢ 用windows中的Eclipse连接虚拟机的Redis的注意事项

- 禁用Linux的防火墙：Linux里执行命令 **service iptables stop**
- redis.conf中注释掉bind 127.0.0.1 ,然后 protect-mode no。

Redis的Java客户端Jedis

➢ 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());
  }
}
```

Redis的Java客户端Jedis

Jedis-API: Key

```
//key
Set<String> keys = jedis.keys("*");
for (Iterator iterator = keys.iterator(); iterator.hasNext();) {
  String key = (String) iterator.next();
  System.out.println(key);
}
System.out.println("jedis.exists====>"+jedis.exists("k2"));
System.out.println(jedis.ttl("k1"));
```

Redis的Java客户端Jedis

Jedis-API: String

```
System.out.println(jedis.get("k1"));
jedis.set("k4","k4_Redis");
System.out.println("----------------------------------------");
jedis.mset("str1","v1","str2","v2","str3","v3");
System.out.println(jedis.mget("str1","str2","str3"));
```

Jedis-API: List

```
List<String> list = jedis.lrange("mylist",0,-1);
    for (String element : list) {
      System.out.println(element);
    }
```

Redis的Java客户端Jedis

Jedis-API: set

```
jedis.sadd("orders","jd001");
jedis.sadd("orders","jd002");
jedis.sadd("orders","jd003");
Set<String> set1 = jedis.smembers("orders");
for (Iterator iterator = set1.iterator(); iterator.hasNext();) {
  String string = (String) iterator.next();
  System.out.println(string);
}
jedis.srem("orders","jd002");
```

Jedis-API: hash

```
jedis.hset("hash1","userName","lisi");
System.out.println(jedis.hget("hash1","userName"));
Map<String,String> map = new HashMap<String,String>();
map.put("telphone","13810169999");
map.put("address","atguigu");
map.put("email","abc@163.com");
jedis.hmset("hash2",map);
List<String> result = jedis.hmget("hash2", "telphone","email");
for (String element : result) {
  System.out.println(element);
}
```

Redis的Java客户端Jedis

Jedis-API: zset

```
jedis.zadd("zset01",60d,"v1");
jedis.zadd("zset01",70d,"v2");
jedis.zadd("zset01",80d,"v3");
jedis.zadd("zset01",90d,"v4");
Set<String> s1 = jedis.zrange("zset01",0,-1);
for (Iterator iterator = s1.iterator(); iterator.hasNext();) {
  String string = (String) iterator.next();
  System.out.println(string);
}
```

- 作业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

Redis事务

Redis的事务

➢ Redis的事务定义

- All the commands in a transaction are serialized and executed sequentially. It can never happen that a request issued by another client is served **in the middle** of the execution of a Redis transaction. This guarantees that the commands are executed as a single isolated operation.

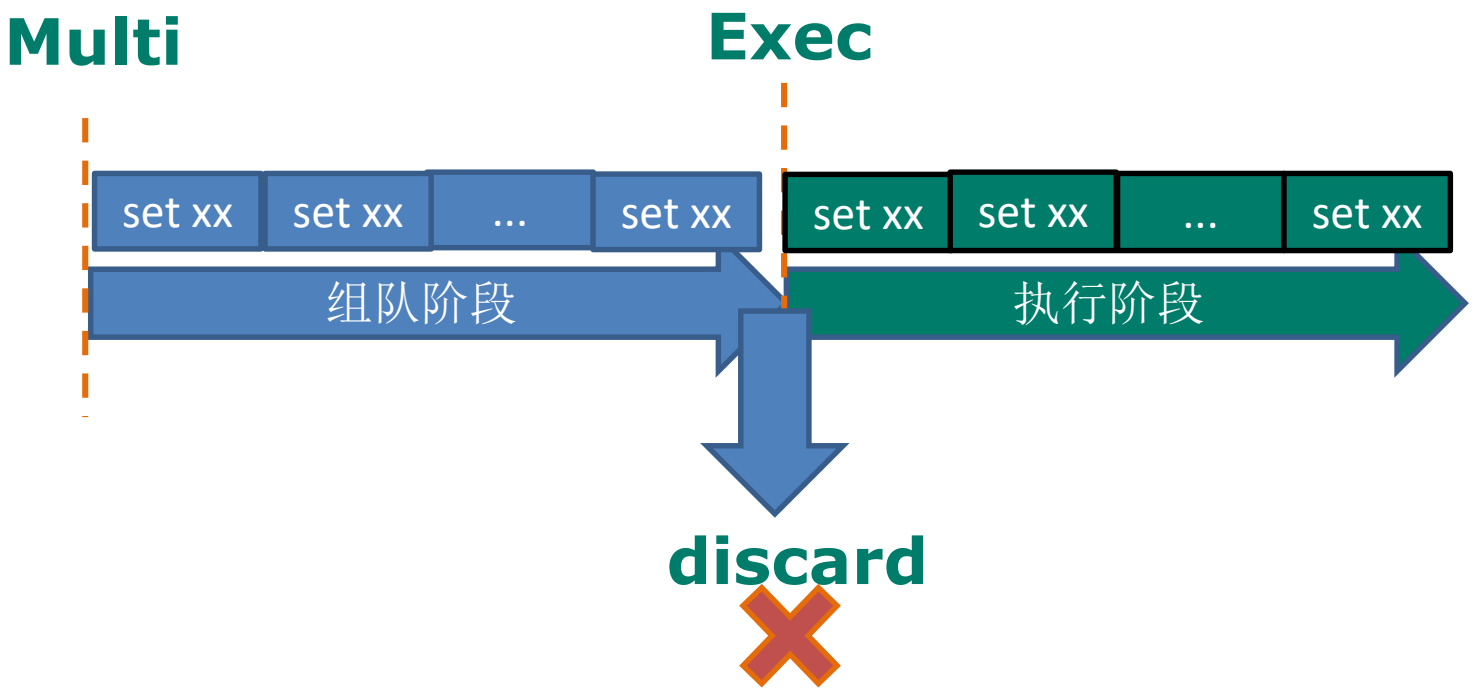
- Redis事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。

- Redis事务的主要作用就是串联多个命令防止别的命令插队

➢ Multi、Exec、discard

- 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。

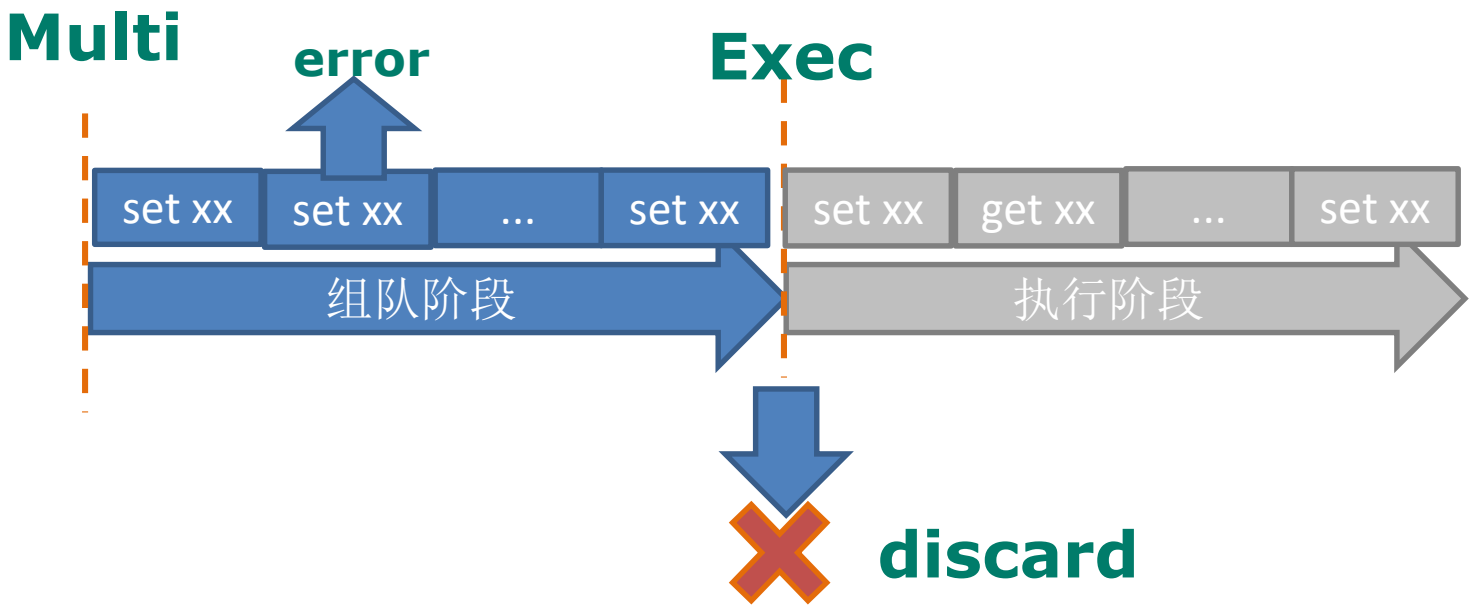
- 组队的过程中可以通过discard来放弃组队。



Redis的事务

- 事务的错误处理

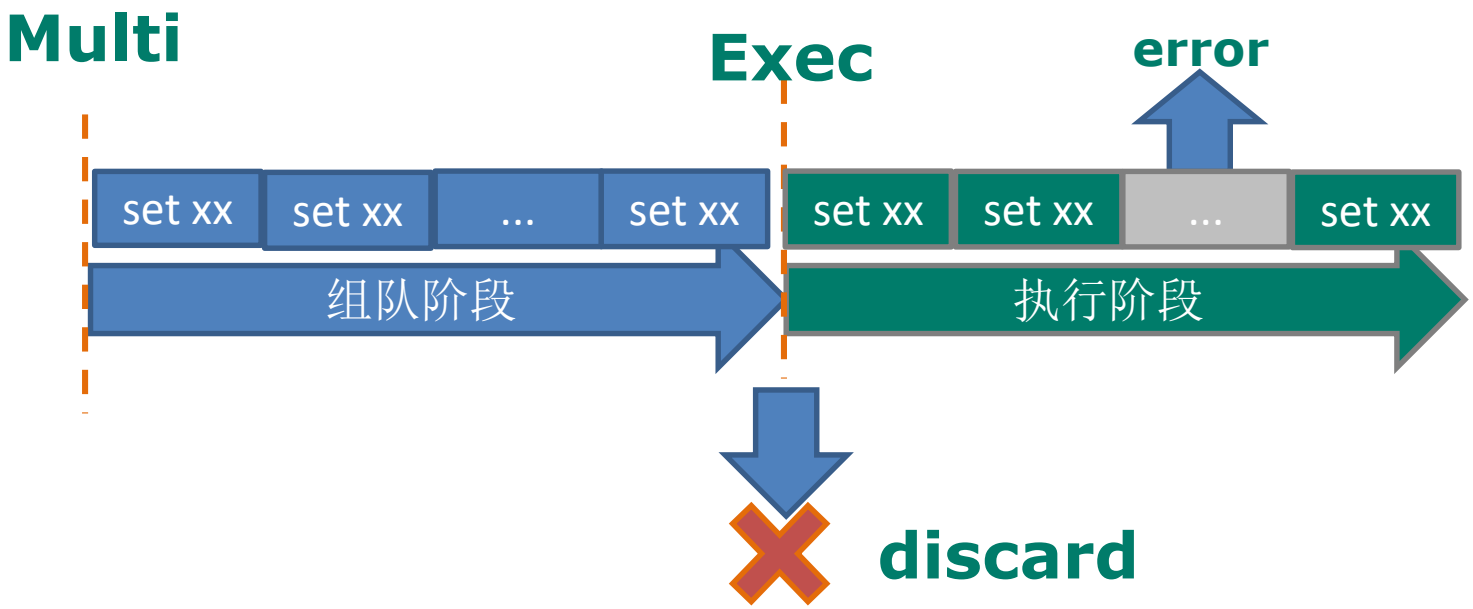
组队中某个命令出现了报告错误，执行时整个的所有队列会都会被取消。



Redis的事务

- 事务的错误处理

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。

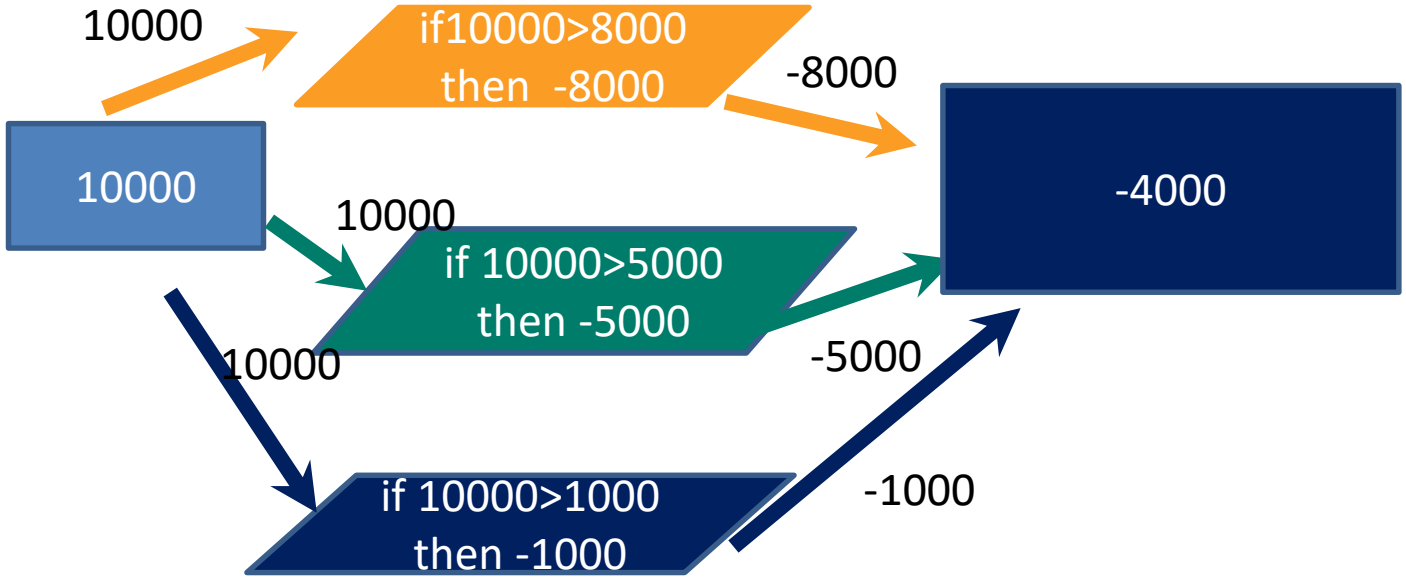


- 为什么要做成事务

想想一个场景：

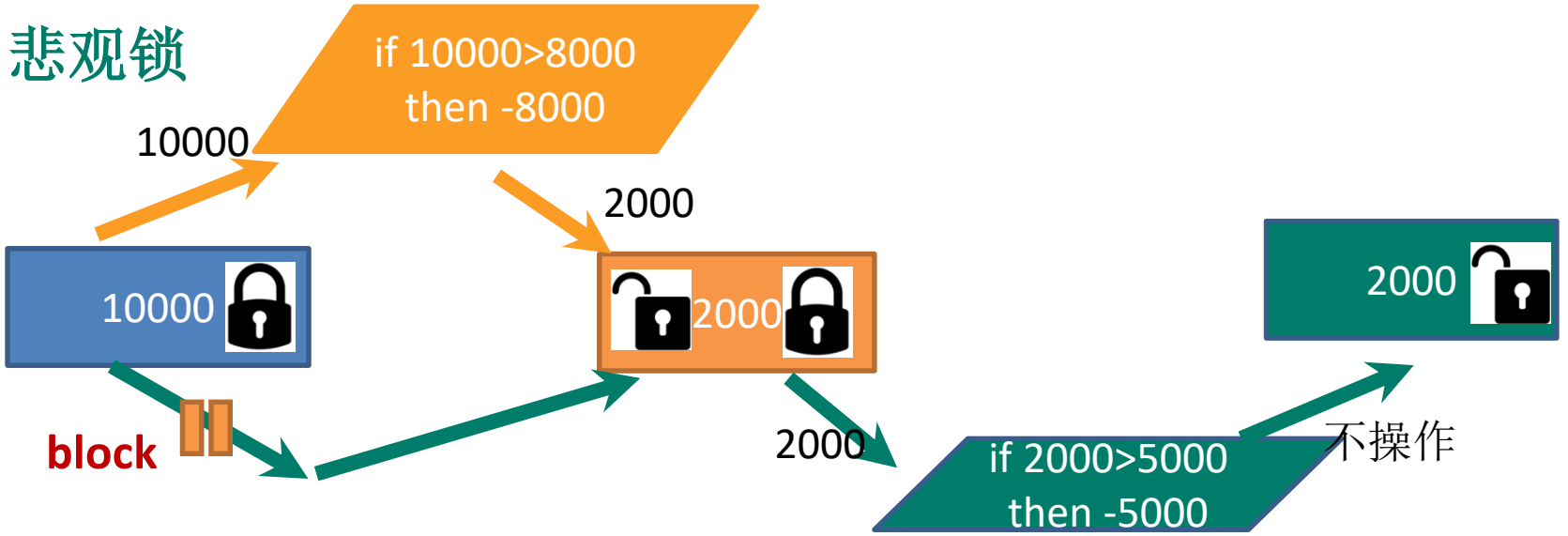
有很多人有你的账户,同时去参加双十一抢购

- 事务冲突的问题
  - 两个请求
    - 一个请求想给金额减**8000**
    - 一个请求想给金额减**5000**
    - 一个请求想给金额减**1000**

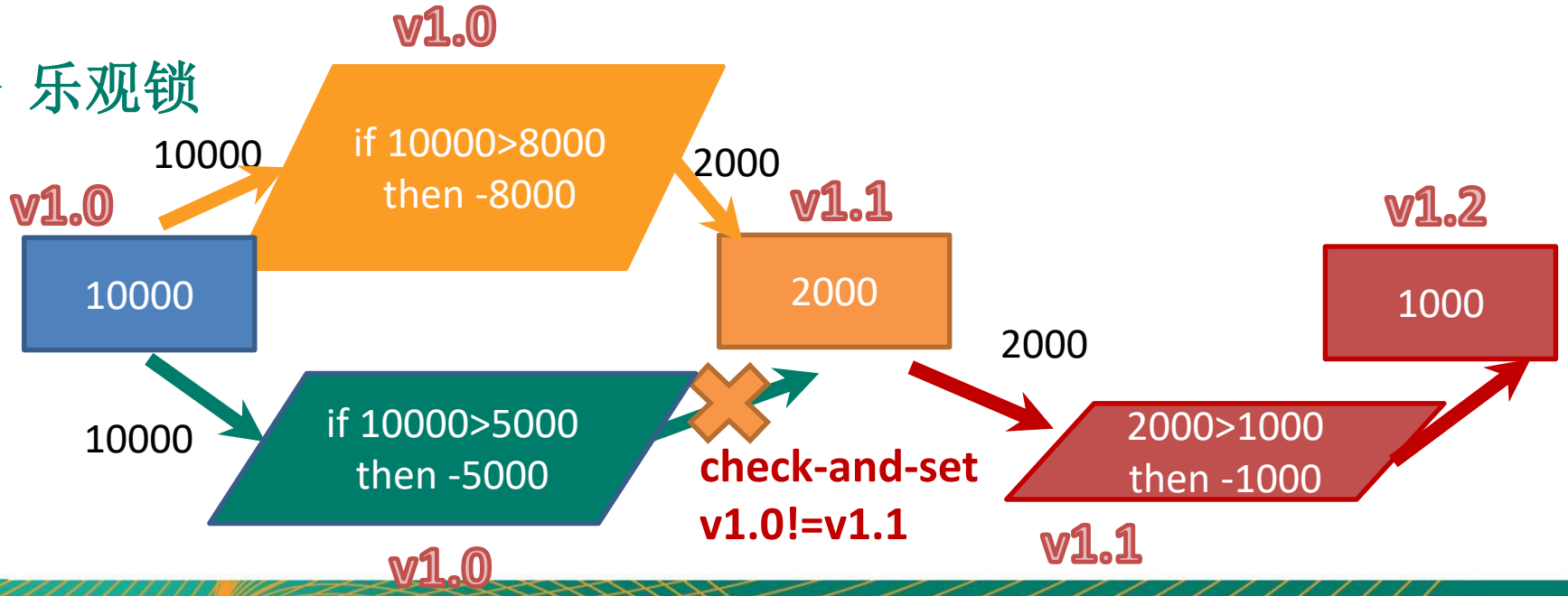


Redis的事务

- 悲观锁



- 乐观锁



Redis的事务

- **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。

Redis的事务

- **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
```

Redis的事务

- 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。

Redis的事务

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

Redis事务--秒杀案例

Redis事务--秒杀案例

解决计数器和人员记录的事务操作

商品库存

key	string
sk:prodid:qt	剩余个数

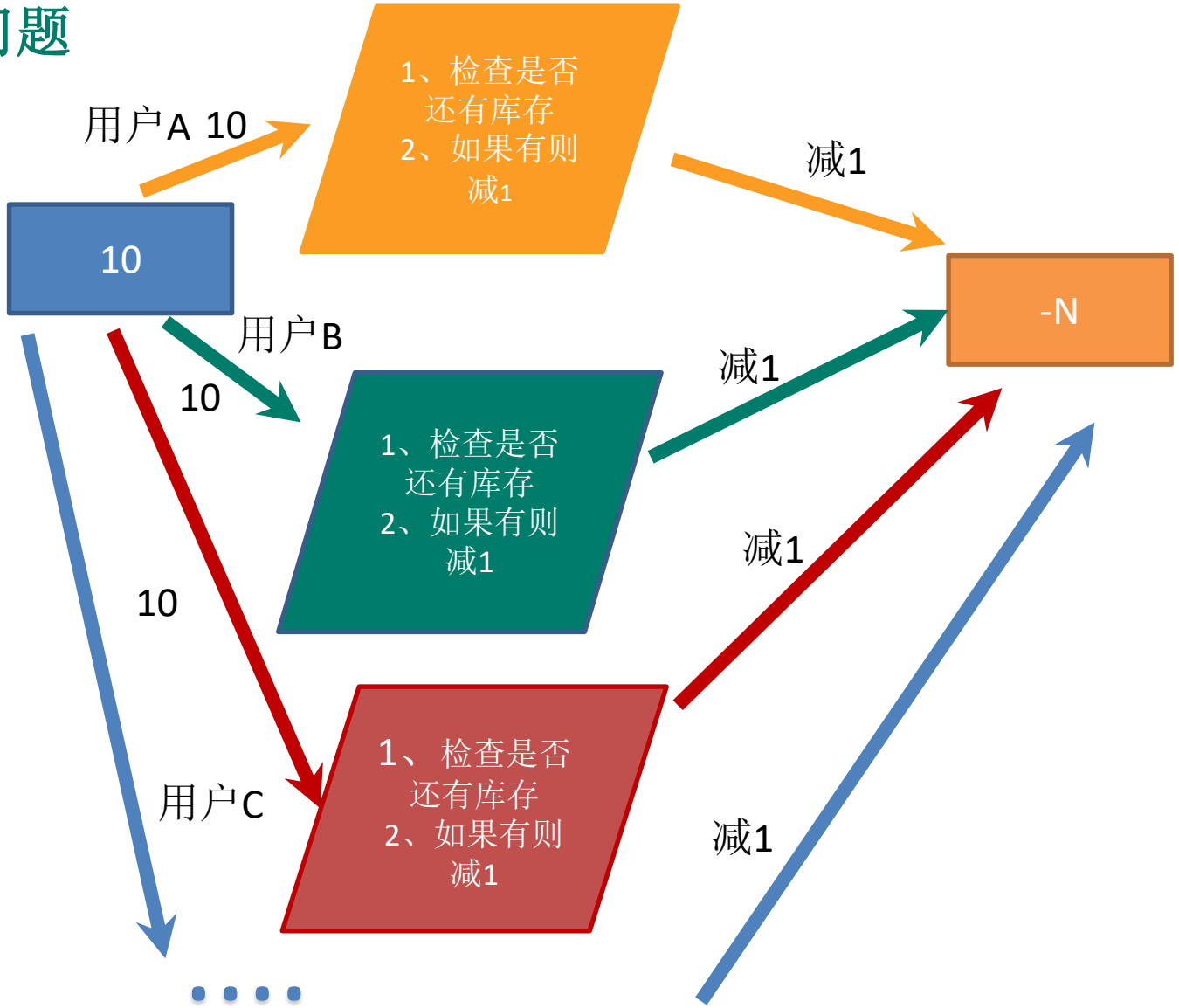
秒杀成功者清单

key	set
sk:prod-id:usr	成功者的user_id
	成功者的user_id
	成功者的user_id

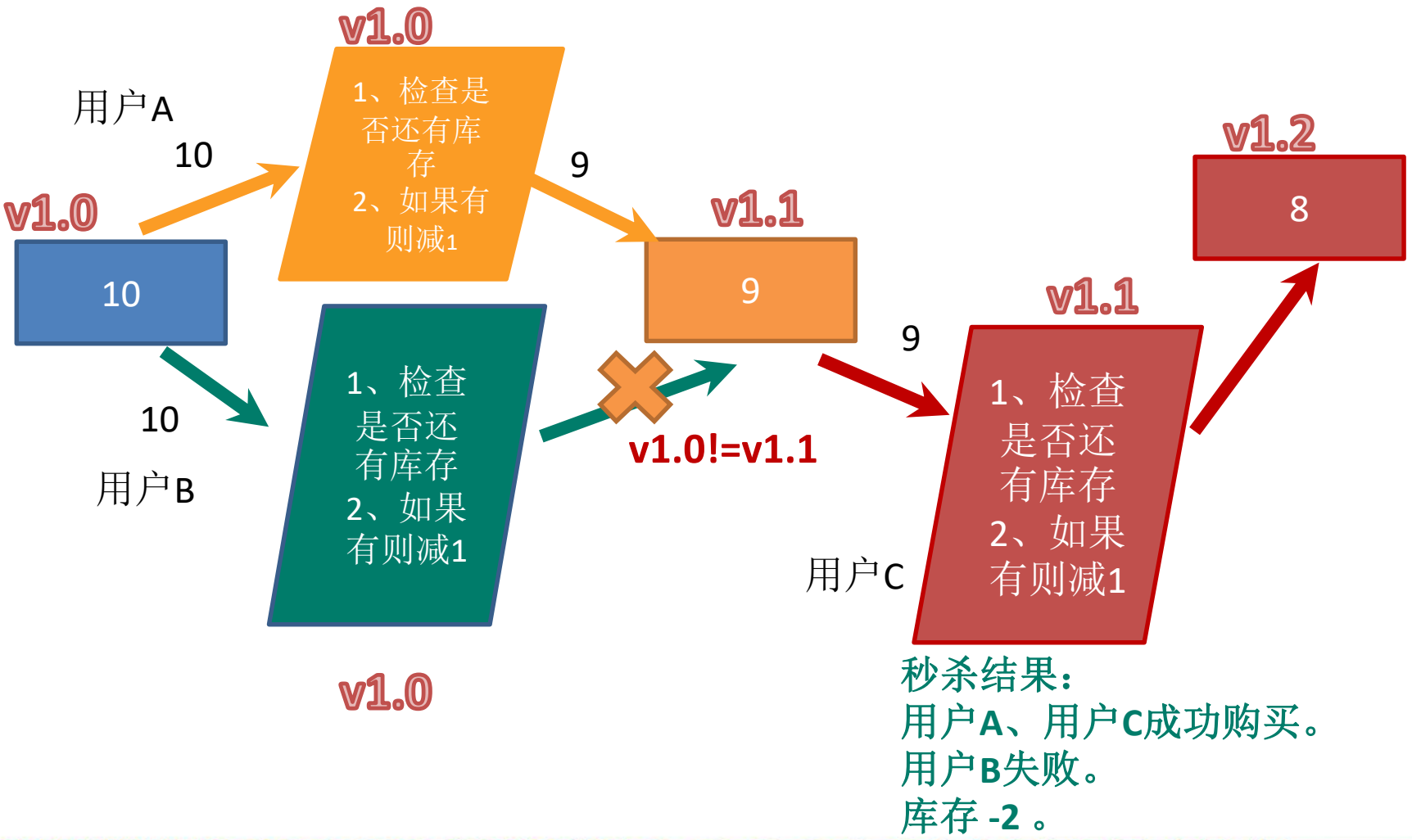
－个数

＋人

超卖问题


用户A 10
1、检查是否还有库存
2、如果有则减1
减1
10
-N
用户B
10
减1
1、检查是否还有库存
2、如果有则减1
减1
10
1、检查是否还有库存
2、如果有则减1
用户C
减1

2、利用乐观锁淘汰用户，解决超卖问题。



Redis的Java客户端Jedis

➢ 链接池

- 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。

- 通过参数管理连接的行为

代码见项目中

链接池参数

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；

➢ 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。

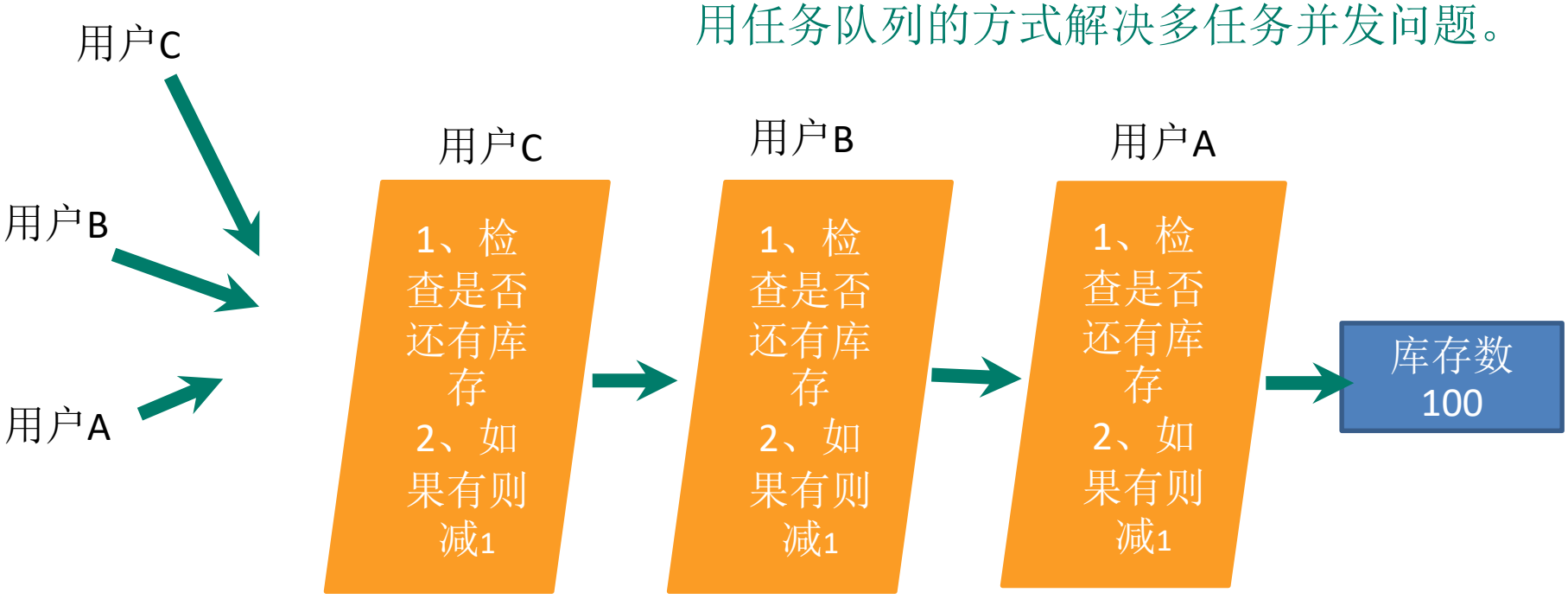
很多应用程序、游戏使用LUA作为自己的嵌入式脚本语言，以此来实现可配置性、可扩展性。这其中包括魔兽争霸地图、魔兽世界、博德之门、愤怒的小鸟等众多游戏插件或外挂。

➢ 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。

redis 2.6版本以后，通过lua脚本解决争抢问题，实际上是redis 利用其单线程的特性，用任务队列的方式解决多任务并发问题。



Redis的持久化

Redis的持久化

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:

- Redis 提供了2个不同形式的持久化方式。
- RDB （Redis DataBase）
- AOF （Append Of File）

Redis的持久化--RDB

RDB

- 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。

Redis的持久化--RDB

➢ 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。

Redis的持久化--RDB

- 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，。

Redis的持久化--RDB

- rdb的保存的文件
  - 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
```

  - rdb文件的保存路径，也可以修改。默认为Redis启动时命令行所在的目录下

```
# 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 ./
```

Redis的持久化--RDB

- 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
```

- 手动保存快照
  - 命令save: 只管保存，其它不管，全部阻塞
  - save vs bgsave

Redis的持久化--RDB

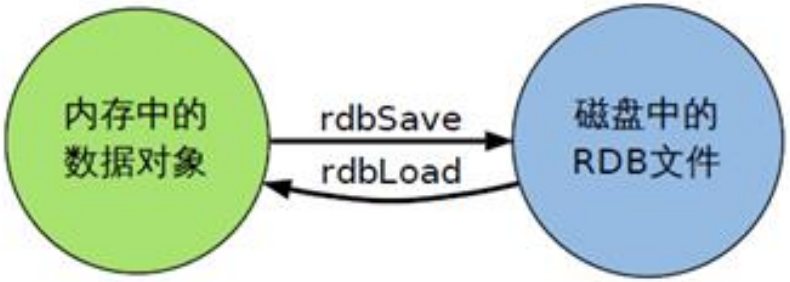
- stop-writes-on-bgsave-error yes
  - 当Redis无法写入磁盘的话，直接关掉Redis的写操作
- rdbcompression yes
  - 进行rdb保存时，将文件压缩
- rdbchecksum yes
  - 在存储快照后，还可以让Redis使用CRC64算法来进行数据校验，但是这样做会增加大约10%的性能消耗，如果希望获取到最大的性能提升，可以关闭此功能

Redis的持久化--RDB

- rdb的备份
  - 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
- rdb的恢复
  - 关闭Redis
  - 先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

Redis的持久化--RDB

- rdb的优点
  - 节省磁盘空间
  - 恢复速度快



- rdb的缺点
  - 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。

Redis的持久化--AOF

AOF

- 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。

Redis的持久化--AOF

- 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
```

- 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。

- AOF和RDB同时开启，redis听谁的？

Redis的持久化--AOF

- 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。
  - AOF和RDB同时开启，系统默认取AOF的数据
- AOF文件故障恢复
  - AOF文件的保存路径，同RDB的路径一致。
  - 如遇到AOF文件损坏，可通过
    redis-check-aof --fix appendonly.aof 进行恢复

Redis的持久化--AOF

AOF同步频率设置

- 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。

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

# appendfsync always
appendfsync everysec
# appendfsync no
```

Redis的持久化--AOF

➢ 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的持久化--AOF

- 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。

Redis的持久化--AOF

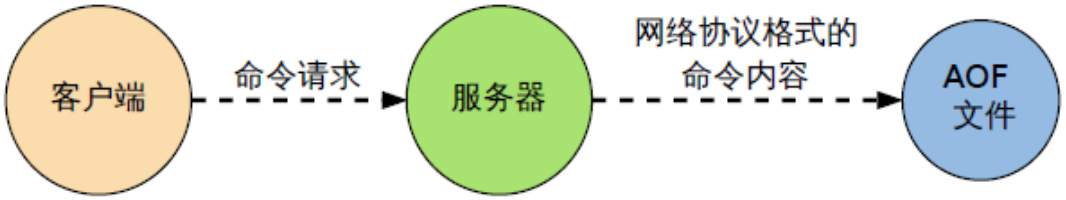
➢ 何时重写

- 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。

Redis的持久化--AOF



- AOF的优点
  - 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。
- AOF的缺点
  - 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。
  - 存在个别Bug，造成恢复不能。

Redis的持久化--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。

Redis的主从复制

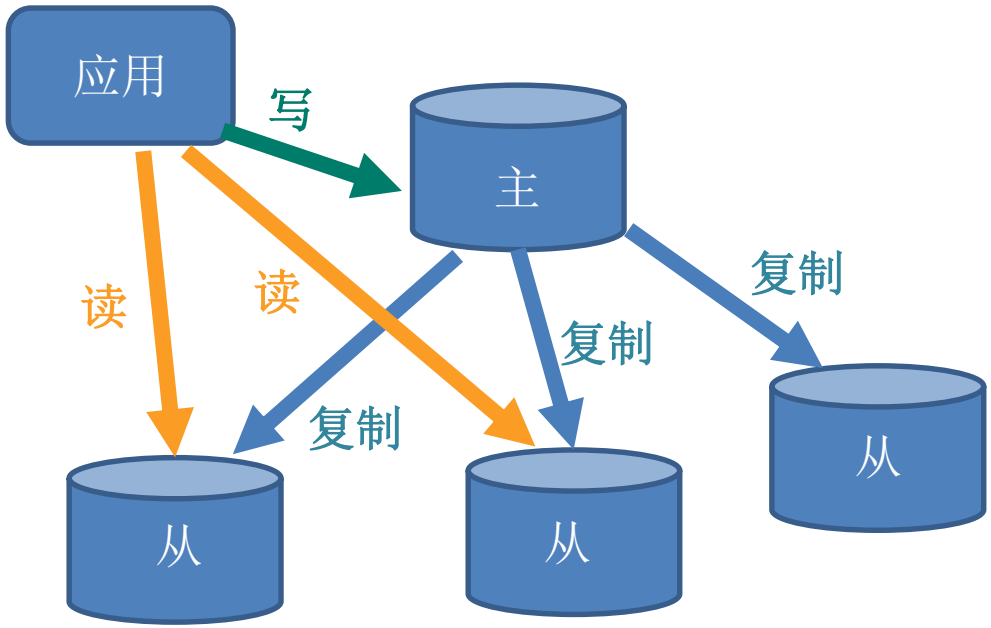
Redis的主从复制

➢ 是什么

- 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**

➢ 用处

- **读写分离，性能扩展**
- **容灾快速恢复**



Redis的主从复制

- 配从(服务器)不配主(服务器)
  - 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

Redis的主从复制

- **info replication**
  - 打印主从复制的相关信息

- **slaveof <ip> <port>**
  - 成为某个实例的从服务器

Redis的主从复制

➢ 一主二仆模式演示

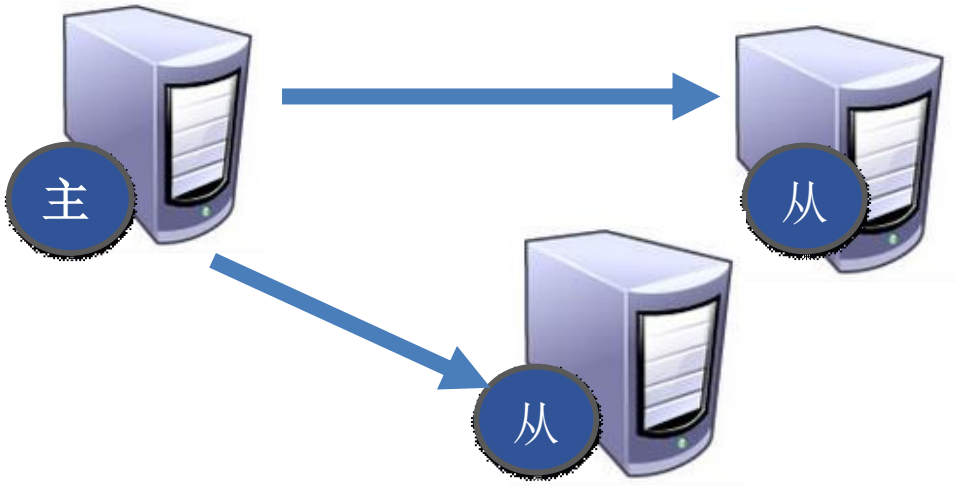
1 切入点问题？slave1、slave2是从头开始复制还是从切入点开始复制?从头开始复制比如从k4进来，那之前的123是否也可以复制

2 从机是否可以写？set可否?

3 主机shutdown后情况如何？从机是上位还是原地待命 原地待命

4 主机又回来了后，主机新增记录，从机还能否顺利复制？可以

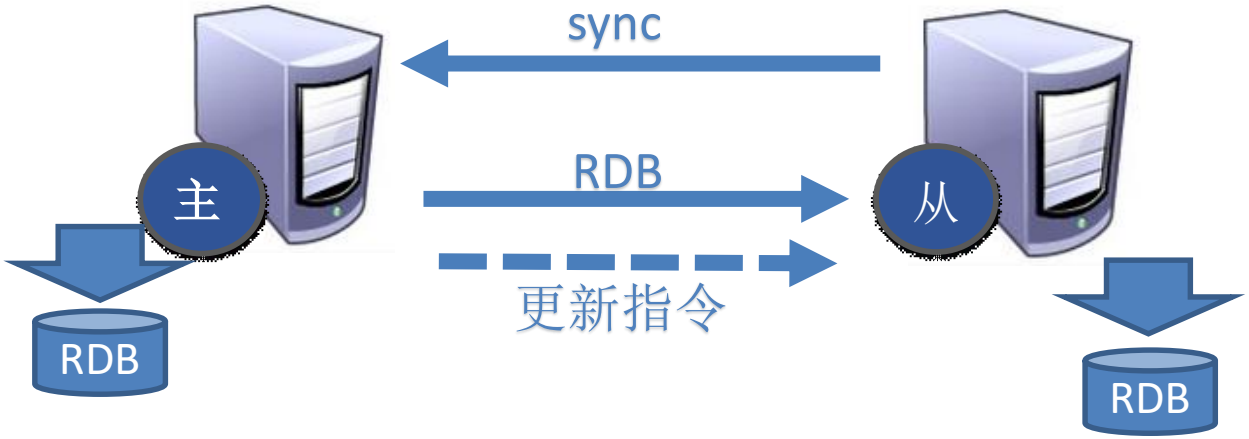
5 其中一台从机down后情况如何？变成主机依照原有它能跟上大部队吗？不能，需要在配置文件中添加slaveof 127.0.0.1 6379



Redis的主从复制

➢ 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



Redis的主从复制

➢ 薪火相传

- 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。
- 用 slaveof <ip> <port>
- 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

Redis的主从复制

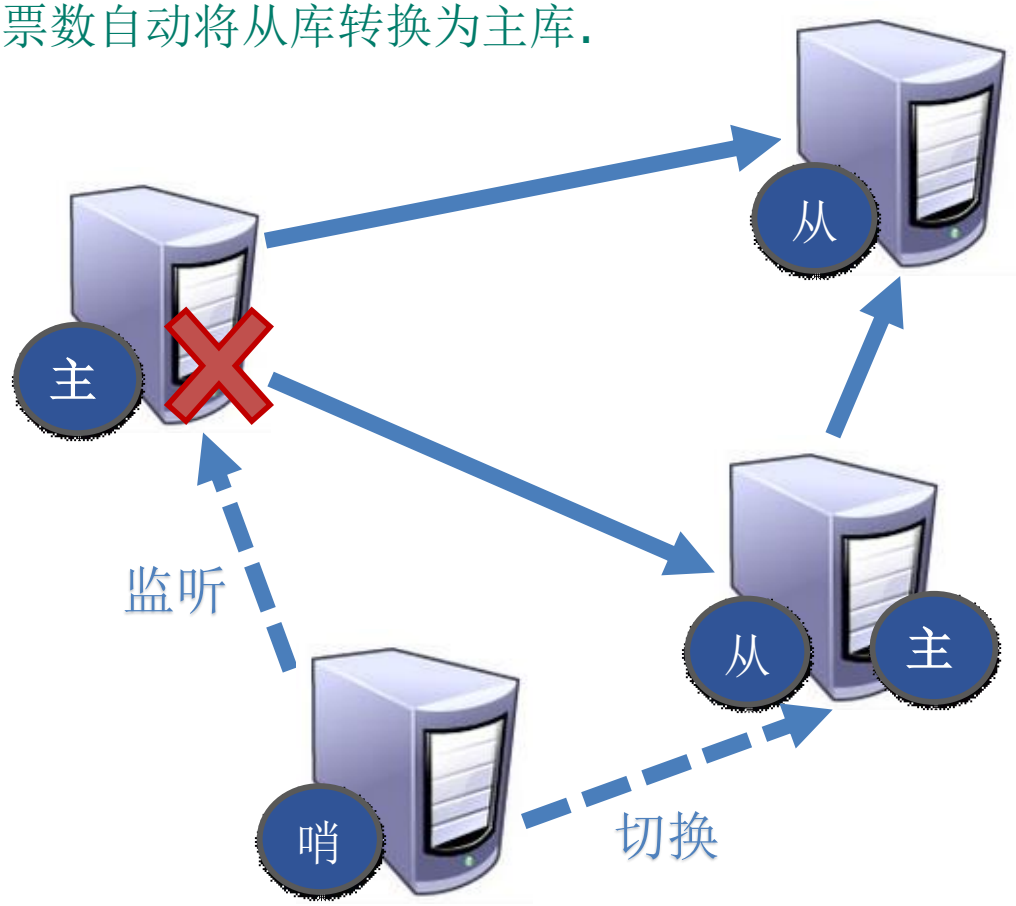
➢ 反客为主

- 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。
- 当原来的master又复活之后，后面的slave和原来的mamaster就没关系了

Redis的主从复制

➢ 哨兵模式(sentinal)

- 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.



Redis的主从复制

➢ 配置哨兵

- 调整为一主二仆模式
- 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。

Redis的主从复制

- 启动哨兵
  - 执行redis-sentinel /myredis/sentinel.conf

Redis的主从复制

➢ 故障恢复

新主登基

从下线的主服务的所有从服务里面挑选一个从服务，将其转成主服务

选择条件依次为：

1、选择优先级靠前的

2、选择偏移量最大的

3、选择runid最小的从服务

群仆俯首

挑选出新的主服务之后，sentinel 向原主服务的从服务发送 slaveof 新主服务 的命令，复制新master

旧主俯首

当已下线的服务重新上线时，sentinel会向其发送slaveof命令，让其成为新主的从

优先级在redis.conf中slave-priority 100

偏移量是指获得原主数据最多的

每个redis实例启动后都会随机生成一个40位的runid

redis的集群

➢问题

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

什么是集群

- 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。

- 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。

➢ 安装ruby环境

1、**依次执行**在安装光盘下的Package目录(/media/CentOS_6.8_Final/Packages)下的rpm包：

- 执行rpm -ivh compat-readline5-5.2-17.1.el6.x86_64.rpm
- 执行rpm -ivh ruby-libs-1.8.7.374-4.el6_6.x86_64.rpm
- 执行rpm -ivh ruby-1.8.7.374-4.el6_6.x86_64.rpm
- 执行rpm -ivh ruby-irb-1.8.7.374-4.el6_6.x86_64.rpm
- 执行rpm -ivh ruby-rdoc-1.8.7.374-4.el6_6.x86_64.rpm
- 执行rpm -ivh rubygems-1.3.7-5.el6.noarch.rpm

2、拷贝redis-3.2.0.gem到/opt目录下

3、执行在opt目录下执行 gem install --local redis-3.2.0.gem

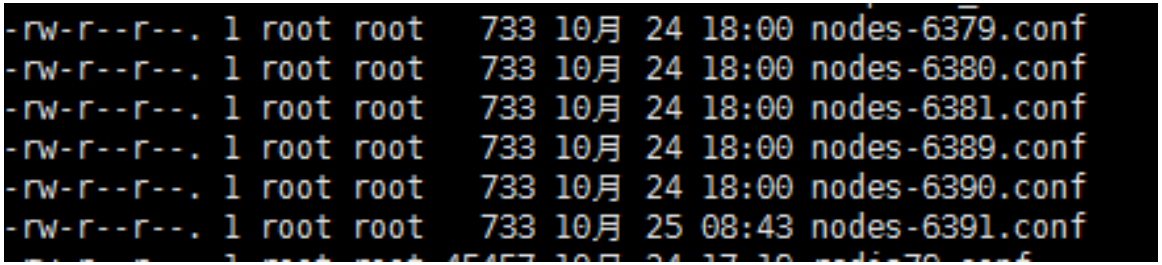
- 制作6个实例，6379,6380,6381,6389,6390,6391
  - 拷贝多个redis.conf文件
  - 开启daemonize yes
  - Pid文件名字
  - 指定端口
  - Log文件名字
  - Dump.rdb名字
  - Appendonly 关掉或者换名字

➢ 安装redis cluster配置修改

- cluster-enabled yes　打开集群模式
- cluster-config-file nodes-6379.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。

➢ 将六个节点合成一个集群

- 组合之前，请确保所有redis实例启动后，nodes-xxxx.conf文件都生成正常。



- 合体：
- cd /opt/redis-3.2.5/src
- ./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
  - 此处不要用127.0.0.1， 请用真实IP地址

➢ 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
```

➢ 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。

➢ 什么是slots

```
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

- 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。

➢ 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。

➢ 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。

➢ 故障恢复

- 如果主节点下限？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

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"));
    }

}
```

Redis 集群提供了以下好处：

- 实现扩容
- 分摊压力
- 无中心配置相对简单

Redis 集群的不足：

- 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。