一 Redis 介绍

1 redis 是什么

redis 是一个存储数据的容器,相当于数据库能存储数据是一样的,实现对数据的增删查改

2 特性

2.A nosql:

not only structured query language 不仅结构化查询语言,一类操作命令的统称 nosql

2.B key-value :

为了处理非结构化数据,使用 key-value 的数据结构处理数据.处理结构化数据的技术--关系型数据库,处理非结构化数据--redis mongdb hdfs

1. 非结构化数据:不同提取公用属性结构的一批数据集合
    1. 日志文件
    2. 网页数据
2. 结构化数据:可以提取公用属性结构的一批数据集合
    1. student 表格,都有名称,学号,id 值等字段

2.C 内存运行

2. redis 在运行过程,数据在内存中处理
    1. redis 使用结构:包含 2 个 服务端运行处理数据的进程
       登录 redis 服务端操作数据的客户端

- 优点缺点:
    - 优点:速度快
    - 缺点:
        - 内存容量,相对于磁盘较小,redis 存储空间资源稀缺
        - 内存断电丢失数据情况,数据容错性比较低

2.D 分布式:

- 一个 redis 节点内存容量小,可以多启动几个 rendis,形成分布式集群解决容量小的问题,相对单个 redis 进程,处理容量呈线性增长(总量还是不如磁盘多),分布式结构中就要解决分布式的问题--数据分片计算

2.E 持久化:

- redis 的数据虽然是在内存处理的,但是 redis 提供持久化的功能,可以按照持久化的机制,将内存数据保存在磁盘一份.即使内存数据断电丢失了,也可以保证数据依然存在继续被使用--增加 redis 容错性.

二 Redis 的安装目录

三 启动 redis 和 redis 脚本

1 服务端启动:

```
redis-server #默认端口启动
redis-server --port 8080 #默认有保护模式，外部连接访问不到
redis-server /home/software/redis-3.2.11/redis.conf #指定配置文件启动
```

退出关闭进程

- ps 查看进程 id 直接杀掉

[root@10-9-104-184 redis-3.2.11]# ps -ef|grep redis
[root@10-9-104-184 redis-3.2.11]# kill pid

- 使用客户端登录后关闭 shutdown

2 客户端启动

- 在 redis 软件中提供客户端登录的脚本开启客户端进程,一个服务端可以被多个启动的客户端进程同时连接使用.

[root@10-9-104-184 redis-3.2.11]# redis-cli

redis-cli 启动一个客户端进程,同时也有选项和参数
-p 表示登录到服务端启动的端口号 默认 6379
-h 表示登录到服务端启动 ip 地址 默认 127.0.0.1

```
[root@10-9-104-184 redis-3.2.11]# redis-cli -p 9000
```

四　Redis 的基础命令

set/get 非常常用的字符串类型数据的写、覆盖/读

以下的命令是不区分数据类型（key-value 类型是总类型，在 redis 中对于 value 数据结构是严格的区分的，存在五种不同的 value 数据类型）

1　keys *

表示在客户端查看当前 redis 服务端内存中所有的数据 key 值。

将已有的数据返回，没有数据时，返回空。这里返回的都是内存中保存的数据 key 值

```
127.0.0.1:6379> set age 100
OK
127.0.0.1:6379> keys *
1) "age"
127.0.0.1:6379>
127.0.0.1:9000> keys *
(empty list or set)
```

```
127.0.0.1:9000> keys *
(empty list or set)
```

注意：不支持分布式结构。不能通过一个 keys * 从一个 redis 服务中查看其它 redis 数据

生产环境（客户使用环境）使用 keys *不合理，造成读数据阻塞（一次读太多数据）

2　exists <key>

表示要查看一下对应的 key 值数据是否在 redis 内存中存在。

```
10.42.175.170:8001> exists name
(integer) 1
10.42.175.170:8001> exists age
 (integer) 0
```

注意：如果判断存在的数据，有读的操作，比如字符串类型的 get，不需要 exists 的存在了呢？不可以使用 get 这种读操作代替 exists 判断存在的操作，因为使用读判断存在，浪费了读数据的带宽。而且 redis 最新版本一个 value 数据可以达到 1GB 大小。

3 expire/pexpire <key> <time>

在 redis 中，可以根据需求对写入的数据设置超时时间，一旦到达超时条件
将会在内存中把数据删除，expire 对某个 key 的数据做秒单位的超时，pexpire 对 key
值做毫秒单位的超时。没有使用相关超时的数据写入时，默认是永久数据。

```
127.0.0.1:9000> set name ShuXinSheng
OK
127.0.0.1:9000> expire name 5 #5 秒删除
(integer) 1
127.0.0.1:9000> pexpire name 5000 #5 秒删除
(integer) 0
```

4 ttl/pttl <key>

在执行设置了超时时间的 key 值上，查看这个 key 的剩余时间。
ttl 操作一个 key 能够看到剩余时间单位是秒，pttl 看得是毫秒。

```
127.0.0.1:9000> set name ShuXinSheng
OK
127.0.0.1:9000> expire name 60
(integer) 1
127.0.0.1:9000> ttl name
(integer) 57
127.0.0.1:9000> pttl name
(integer) 51231
```

永久数据返回值-1
已删除/不存在的数据返回值-2

5 del <key>

del 可以对指定的 key-value 进行删除操作。

```
127.0.0.1:9000> set name ShuXinSheng
OK
127.0.0.1:9000> del name
(integer) 1
```

6 save

redis 支持持久化，将内存数据，输出到持久化文件，内存数据保存在磁盘上。
redis 重新启动时自动加载保存的持久化文件，将数据恢复回来。
save 命令的调用，就是将内存数据输出到持久化文件中保存。

```
127.0.0.1:6379> save
OK
```

7 flushall

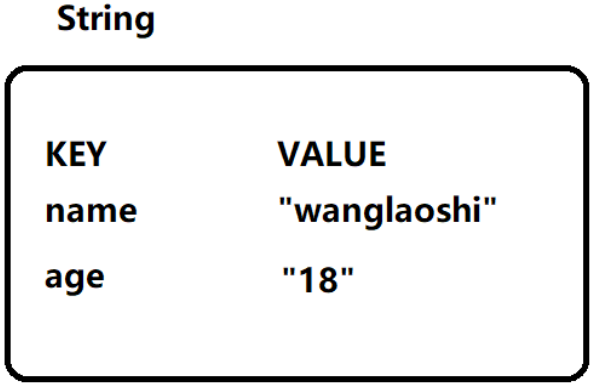
冲刷所有,删除所有数据。将当前 redis 服务的内存数据和持久化文件中的数据全部清空。尽可能只在测试环境使用，不要到生产环境。

```
127.0.0.1:9000> set name ShuXinSHENG
OK
127.0.0.1:9000> KEYS *
1) "name"
127.0.0.1:9000> flushall
OK
127.0.0.1:9000> keys *
(empty list or set)
```

五 Redis 的数据类型

redis 是以 key-value 结构存储数据的，但是根据不同的应用场景，可以使用完全不同的 value 结构存储数据。包括：String 字符串，Hash，List 链表，Set 集合，ZSet 有序集合

1 String 类型



1.A set/get

redis 中可以对字符串类型进行写操作调用 set 命令，也可以在已有数据时，对数据覆盖操作。在 redis 的大量命令都可以携带很多不同的参数选项。使用详细的选项和参数，可以从 redis 官网去查看 连接

EX/PX/NX/XX

存数据和取数据。

```
127.0.0.1:9000> set name ShuXinSheng
OK
127.0.0.1:9000> get name
"ShuXinSheng"
```

```
127.0.0.1:9000> get age
(nil)
```

EX:可以在 set 时直接设置超时秒数

PX:可以在 set 时直接设置超时毫秒数

```
127.0.0.1:6379> set bomb tnt EX 50
OK
127.0.0.1:6379> ttl bomb
(integer) 46
```

NX：在执行 set 时，会判断 redis 中有没有该 key 值，如果有则无法 set，没有则可以 set 成功。表示，只有第一个 set 数据的客户端可以成功，后续都会失败。

```
127.0.0.1:6379> keys *
1) "age"
127.0.0.1:6379> set age 22 NX
(nil)
127.0.0.1:6379> set name wanglaoshi NX
OK
```

XX：在执行 set 时，会判断 redis 中有没有 key 值，有的时候才会 set 成功，没有则不成功。表示，使用 XX 的客户端没有新建的权限。

```
127.0.0.1:6379> keys *
1) "age"
127.0.0.1:6379> set gender male XX
(nil)
127.0.0.1:6379> set age 55 XX
OK
```

1.B incr/incrby decr/decrby

执行计步器，可以增加数值，减少数值。对应 value 字符串数据必须是纯数字

```
127.0.0.1:9000> set age 24
OK
127.0.0.1:9000> incr age
(integer) 25
127.0.0.1:9000> decr age
(integer) 24
127.0.0.1:9000> incrby age 10
(integer) 34
127.0.0.1:9000> decrby age 10
(integer) 24
```

常见的应用使用计步器：

记录排队人数（拿号，自增，叫号后，前剩余人数自减）；

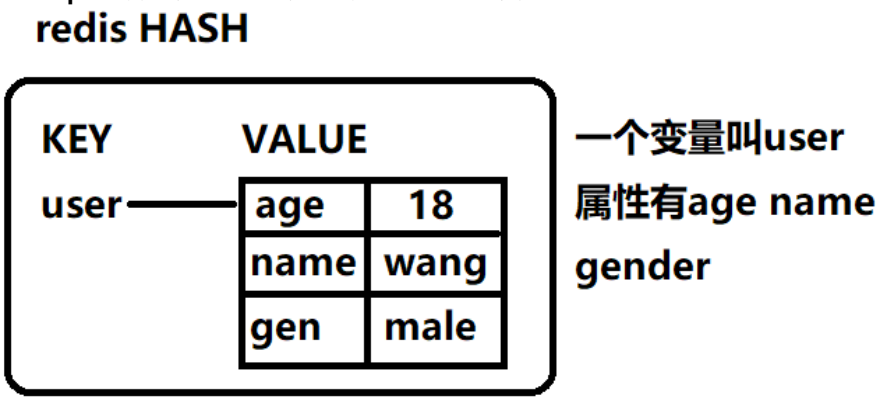
在线人数统计（每秒钟上下变动）

1.C 2.4 数据类型 String 应用场景

一般使用 String 类型的 value 数据实现 缓存的功能。并且可以利用代码的序列化和反序列化的方法，将对象序列化为字符串（user-->{“userName”：“wanglaoshi”}）.在 easymall 中使用序列化将 product 对象变成 json，以商品 id 作为唯一 key 值操作商品在 redis 的缓存数据。

2 hash 类型

hash 在 redis 中底层双层 map 形式存在，key-value 是 map，value 在 hash 结构中又是一个 map。所以他可以对应对象的数据结构。



2.A hset key field value

field 为第二层 map 的 key

```
127.0.0.1:9000> hset user name shuxinsheng
(integer) 1
127.0.0.1:9000> hset user age 24
(integer) 1
```

2.B hget key field

由于 hash 结构双层 map，hget 可以读取到一个属性的值，指定某个 key 的 某个属性读取

```
127.0.0.1:9000> hget user age
"24"
127.0.0.1:9000> hget user name
"shuxinsheng"
```

2.C hkeys/hvals key

hkeys key 从 key 值的 hash 数据结构中将所有的 field 属性名称返回
hvals key 从 key 的 hash 数据结构中将所有的 field 的值返回

```
127.0.0.1:9000> hkeys user
1) "name"
2) "age"
127.0.0.1:9000> hvals user
1) "shuxinsheng"
2) "24"
```

2.D hdel key field

如果想将整个 hash 结构删掉，直接调用 del 全部删除
如果删除的是一个 hash 结构中的某个属性和值，
hdel key field

```
127.0.0.1:9000> hdel user name
(integer) 1
127.0.0.1:9000> hkeys user
1) "age"
```

2.E hincrby

增加指定值

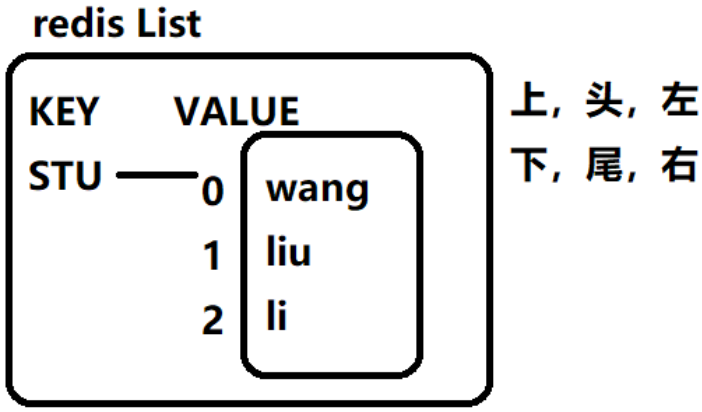
```
127.0.0.1:9000> hget user age
"24"
127.0.0.1:9000> hincrby user age 1
(integer) 25
```

1.A 应用场景

也可以像 String 一样，执行某个项目中的缓存逻辑。不同环境使用不同结构，有不同的效果。
例如，缓存数据对象的结构比较复杂，属性不仅仅是 Integer String 还有数组，list，还有 set，优先使用 String 类型，如果对象属性简单可以使用 hash（造成代码编写复杂）

3 List 双向链表

list 底层双向链表，可以从头和尾部处理数据，实现队列的结构（就为了处理消息队列逻辑）。



3.A lpush/rpush key value

l/r 表示左和右。lpush 从链表头部，插入数据，rpush 从链表的尾部，插入数据

```
127.0.0.1:6379> lpush student wangcuihua
(integer) 1
127.0.0.1:6379> lpush student liuyoucai
(integer) 2
127.0.0.1:6379> rpush student zhangshoufu
(integer) 3
```

形成链表结构顺序：

liuyoucai

wangcuihua

zhangshoufu

3.B lrange key start end

可以对一个 list 链表中的元素范围内的数据读取返回。

lrange student 起始下标 结束下标

```
127.0.0.1:6379> lrange student 0 4
1) "liuyoucai"
2) "wangcuihua"
3) "zhangshoufu"
127.0.0.1:6379> lrange student 1 4
1) "wangcuihua"
2) "zhangshoufu"
```

有时候，咱们并不会确定不知道元素的个数，要想查看所有的元素可以使用-1 的 end 结尾

表示一直到尾部。start=0 end=-1 就可以查询一个 list 所有内容

```
127.0.0.1:6379> lrange student 0 -1
1) "liuyoucai"
2) "wangcuihua"
3) "zhangshoufu"
```

3.C lset key 元素下标 index 修改值

不建议使用，双向链表操作最有效，速度最快是对头尾的操作。
lset 可能从中间操作链表，效率非常低
对 list 的某个下标为 index 的元素值，做修改。

```
127.0.0.1:6379> lrange student 0 -1
1) "liuyoucai"
2) "wangcuihua"
3) "zhangshoufu"
127.0.0.1:6379> lset student 1 haha
OK
127.0.0.1:6379> lrange student 0 -1
1) "liuyoucai"
2) "haha"
3) "zhangshoufu"
127.0.0.1:6379>
```

3.D rpop/lpop key

rpop 从链表的尾部删除元素，并且将删除的元素值返回
lpop 从链表的头部删除元素，将元素值返回（remove）

```
127.0.0.1:6379> lrange student 0 -1
1) "liuyoucai"
2) "haha"
3) "zhangshoufu"
127.0.0.1:6379> rpop student
"zhangshoufu"
127.0.0.1:6379> lpop student
"liuyoucai"
127.0.0.1:6379> lrange student 0 -1
1) "haha"
127.0.0.1:6379>
```

配合 lpush 和 rpush 实现排队机制,先到先得

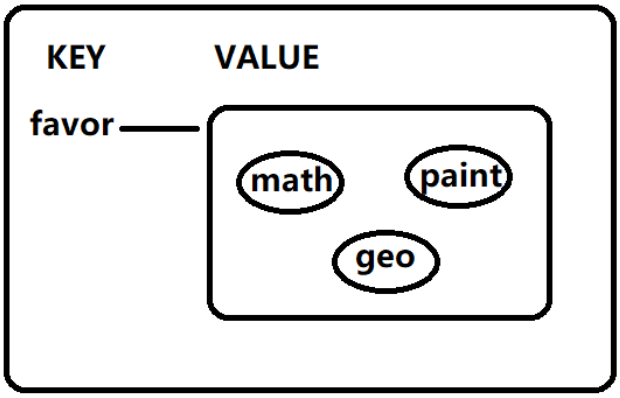
3.E 应用场景

初始目的就是利用 list 的类型实现排队队列的处理逻辑，先来先得，先到先处理。
可以在元素中封装一些消息属性，先进入队列，优先被 pop 出去进行处理。
easymall 使用 list 数据，实现高并发争抢线程资源的权限设置，解决防止线程安全的出现。

4 SET 集合

可以将不同的，不重复的元素值，放到一个没有顺序概念的集合中，实现 value 数据在 redis 的管理

REDIS SET



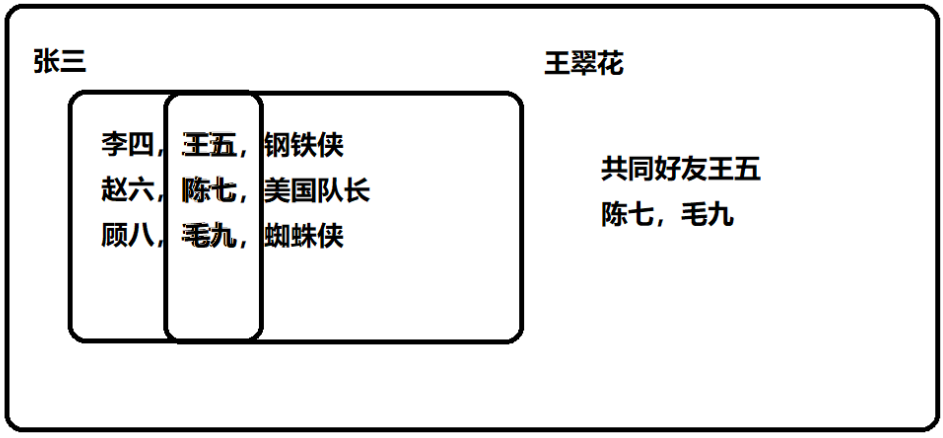
4.A 常用场景

登录系统时，给用户帖的标签。例如：

登录一些明日头条这种推荐文件，推荐网站的系统，注册后选择自己兴趣爱好：军事，数学，历史，天文，登陆后会跟你你选择的内容，随机推荐不同的文章。

直播网站：lol 王者荣耀 dota2 其他，根据兴趣推送相关直播

还可以实现好友保存，计算共同好友



4.B sadd key member

新增元素到集合中。

```
127.0.0.1:6379> sadd favor math english geo
(integer) 3
```

4.C srandmember key [count]

随机在 key 对应 set 集合中选取 count 个元素。

```
127.0.0.1:6379> srandmember favor
"english"
127.0.0.1:6379> srandmember favor 2
1) "math"
2) "geo"
```

4.D srem key 元素值

可以对集合中某个元素进行去除的操作，当删除成功，返回 1，删除失败返回 0

```
127.0.0.1:6379> srem favor math
(integer) 1
127.0.0.1:6379> srem favor mathdasfdads
(integer) 0
127.0.0.1:6379>
```

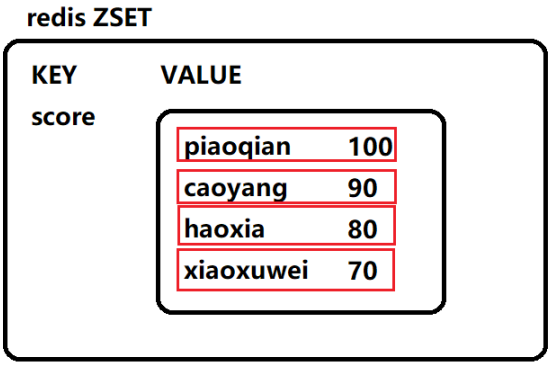
4.E sismember 查看一个元素是否属于这个集合

```
127.0.0.1:6379> sismember favor math
(integer) 0
127.0.0.1:6379> sismember favor english
(integer) 1
127.0.0.1:6379>
```

5 ZSET 有序集合

在 set 基础之上，实现了排序的方式，元素也是不可以重复，就是在元素的数据上绑定了一个

评分的数字（实际应用场景中，评分可以不同业务意义，例如点击次数，例如播放量，例如投票数量等）



5.A 常用应用场景

网站各种排名，都可以使用 ZSET 有序集合

视频网站：热播剧，top10。

热搜：话题搜索次数。

小说网站：订阅量排序，月票排序

5.B zadd key score member

可以将一个元素绑定一个分数后，写入到一个有序集合中

```
127.0.0.1:6379> zadd score 98 piaoqian
(integer) 1
127.0.0.1:6379> zadd score 97 haoxia
(integer) 1
127.0.0.1:6379> zadd score 80 caoyang
(integer) 1
127.0.0.1:6379> zadd score 15000 xiaoxuwei
(integer) 1
```

5.C rank/range 排序查询

在有序集合中主要就是为了体现排序，所以使用命令最多的查询方式就是排序相关内容

a. zrank key member:看看 member 的排名

```
127.0.0.1:6379> zrank score piaoqian
(integer) 2
```

a. zrange key start stop：查询排名从 start 开始到 stop 范围内所有元素

起始排名是 0，可以对 stop=-1 查到末尾

```
127.0.0.1:6379> zrange score 0 -1
1) "caoyang"
2) "haoxia"
3) "piaoqian"
4) "xiaoxuwei"
```

```
127.0.0.1:6379>
```

a. zrangebyscore key minscore maxscore:在上限评分和下限评分之间的所有元素和排序

```
127.0.0.1:6379> zrangebyscore score 50 200
1) "caoyang"
2) "haoxia"
3) "piaoqian"
```

5.D zrem key member

将元素从 zset 类型的数据中删除

六 JAVA 中使用 Redis 的客户端 jedis

redis 支持非常丰富的客户端语言访问使用.包括 java.java 语言里也有多种不同的 redis 客户端,jedis 只是其中一种.其他:lettuce,redission.Jedis 中的 api 方法几乎和 redis 命令一致的.比如: set name haha, set("name","haha")

1 maven 导入依赖

```
<!--redis 缓存-->
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
</dependency>
```

2 redis 读写测试

2.A 客户端对象

```
public void test01(){
    Jedis jedis = new Jedis("10.42.175.170", 9000);
    System.out.println(jedis.set("test","2"));
}
```

2.B 客户端连接池

初始化连接数量:创建连接池时,默认一开始连接对象个数

最小空闲数量:连接池中的连接对象空闲的最小个数,小于最小空闲时(连接不够了),将会创建满足最大空闲的数量的连接对象.

最大空闲数量:连接池中连接对象空闲的最大个数,大于最大空闲时(连接建多了),将会把超过最大空闲的个数的连接删除

最大连接数:按照上述逻辑,繁忙状态的连接池,会不断创建连接对象,有上限,这个上限就是连接最大数

```
@Test
public void test02() throws InterruptedException {
    //连接池配置对象
    JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();
    //连接池最小可用连接
    jedisPoolConfig.setMinIdle(2);
    //连接池对象
    JedisPool jedisPool = new
JedisPool(jedisPoolConfig,"10.42.175.170", 6379);
    //获取客户端对象
    Jedis resource = jedisPool.getResource();
    //存数据
    resource.set("age","100");
}
```

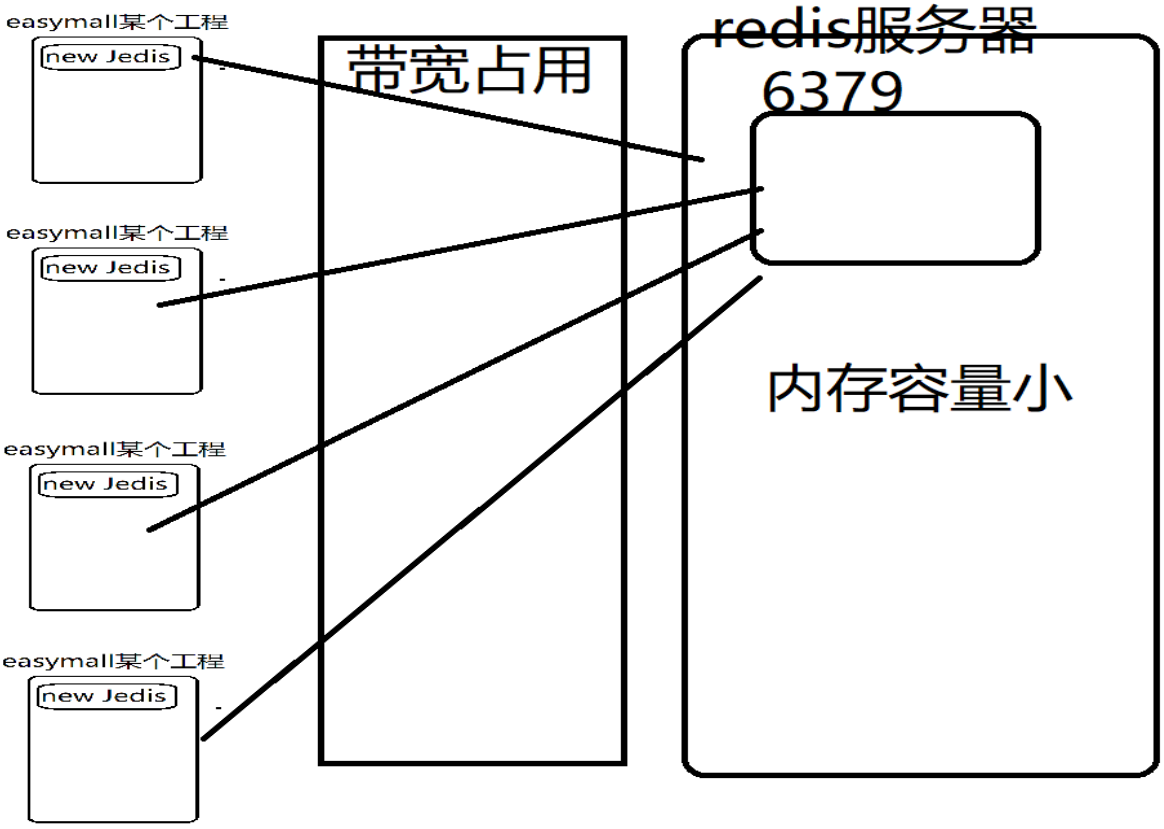
七 分布式计算—Hash 取余

当前使用一个 redis 的进程处理业务逻辑,已经可以实现所有内容,就是用

jedis 对象操作 redis.但是对于一个系统中如果只使用 1 个 redis 进程,不足以支持海量数据

不足以支持对 redis 高并发访问.需要引入分布式结构.

内存容量小、带宽占用高



1 相关概念

1.A 分布式

将原有一个进程,系统处理的任务,分发给不同的进程和系统.

1.B 微服务

分布式的一种,将一个大系统,切分成了多个不同功能小系统.

1.C ★ 数据处理分布式:

数据切分/数据分片计算;

当整体数据要用不同的多个 redis 节点进行读写操作时，就需要引入数据分片计算。同时需要计算逻辑支持单调性

1.D 保真运算:

通过二进制的位的与运算,保存原值的 N 位二进制,保存 8 位 叫做 8 位保真运算,16 位保真运算

1.E **单调性**

某一条数据切分到了 redis 哪个节点, 读取时就要在到哪个节点去读.往哪写的,从哪读

2 Hash 取余运算

经典的分布式计算方法.

对数据的 key 值做计算,对应到一个 redis 节点,只要 redis 集群不发生变化,key 值永远对应这个节点

3 JAVA 的 Hash 取余公式

```
@Test
public void test05(){
     //节点个数
    int n=3;
     String key="key";
     //hashcode
     int hashCode = key.hashCode();
     //取正(取真)
     hashCode=hashCode&Integer.MAX_VALUE;
     //取余 i 一定等于 0 1 2 中的一个
    int i = hashCode % 3;
     System.out.println(i);//2
}
```

key.hashCode()：对 key 值做 java 的 hash 散列计算,将一个内存对象映射到一个整数区间的整数值[-21 亿--21 亿]

&Integer.MAX_VALUE：目的取正(绝对值也可以实现比较慢).散列值是正整数,结果不变,如果是负数,变成正整数.key 值不变,得到对应不变的正整数.

%n：计算结果[0,1,2,3,...,n-1]

有 n 个节点,对节点进行下标设置,0-->n-1 对应计算取余结果

key 的 hash 取余得到 0,到 0 号节点存储

key 的 hash 取余得到 1,到 1 号节点存储

...

key 的 hash 取余得到 n-1,到最后一个节点存储

读取的计算和存储的计算都会经过 hash 取余计算后对应同一个节点--单调性.但是不能保证绝对的平均,会有数据倾斜

4 测试 Redis

```
@Test
public void test06(){
```

```
    ArrayList<Jedis> jedisArrayList = new ArrayList<Jedis>();
    jedisArrayList.add(new Jedis("10.42.175.170", 6379));
    jedisArrayList.add(new Jedis("10.42.175.170", 6380));
    jedisArrayList.add(new Jedis("10.42.175.170", 6381));
    //节点个数
   int n=jedisArrayList.size();
    for (int i = 0; i <1000 ; i++) {
        //随机生成 key
        String key=UUID.randomUUID().toString();
        int result=(key.hashCode()&Integer.MAX_VALUE)%n;
        Jedis jedis = jedisArrayList.get(result);
        //存数据
       jedis.set(key,""+i);
    }
}
```

4.A 结论:

经过 hash 取余能保证单调性,在哪存,就能在哪儿取， key 值不变,对应节点也不变;

但是只能在节点集群不发生变动时使用,一旦发生变动,3 个节点变成 4 个,5 个节点变成 3 个,不能保证单调性了.

集群节点,分片数量可能发生变动,这种环境不宜使用 hash 取余,会破坏单调性,集群节点/分片个数越多,新增,删除时破坏的单调性越强.redis 集群就是不断实现现行扩展—hash 取余 不能在 redis 集群使用.

4.B 连接池

```
@Test
public void test09(){
    //自定义属性 最大连接,最大空闲等
   JedisPoolConfig config=new JedisPoolConfig();
    config.setMaxTotal(50);
    config.setMaxIdle(10);
    config.setMinIdle(2);
    JedisPool pool=new JedisPool(config,"10.9.100.26",9000);
    //通过连接池,获取连接资源
   Jedis jedis = pool.getResource();
    jedis.set("name","舒老师");
    System.out.println(jedis.get("name"));
    //使用完资源,将资源还回连接池
   pool.returnResource(jedis);
}
```

八 一致性 Hash

既然 hash 取余无法解决扩容、缩容时数据的单调性,但一致性 hash 可以解决这个问题---在分布式计算实现单调性后,在分片个数发生变动时,尽可能少的去影响数据单调性.

1 介绍

核心算法还是 hash 散列,是 1997,由麻省理工大学团队创建的一种数学计算模型. 任意计算机对象,散列映射无符号的 32 位二进制区间[0-43 亿](0 到 2 的 32 次方-1)

2 计算原理

2.A Hash 环

虚拟的一个环，范围 0-2^32-1

（1）把所有机器编号计算 hash 对应到这个环上。

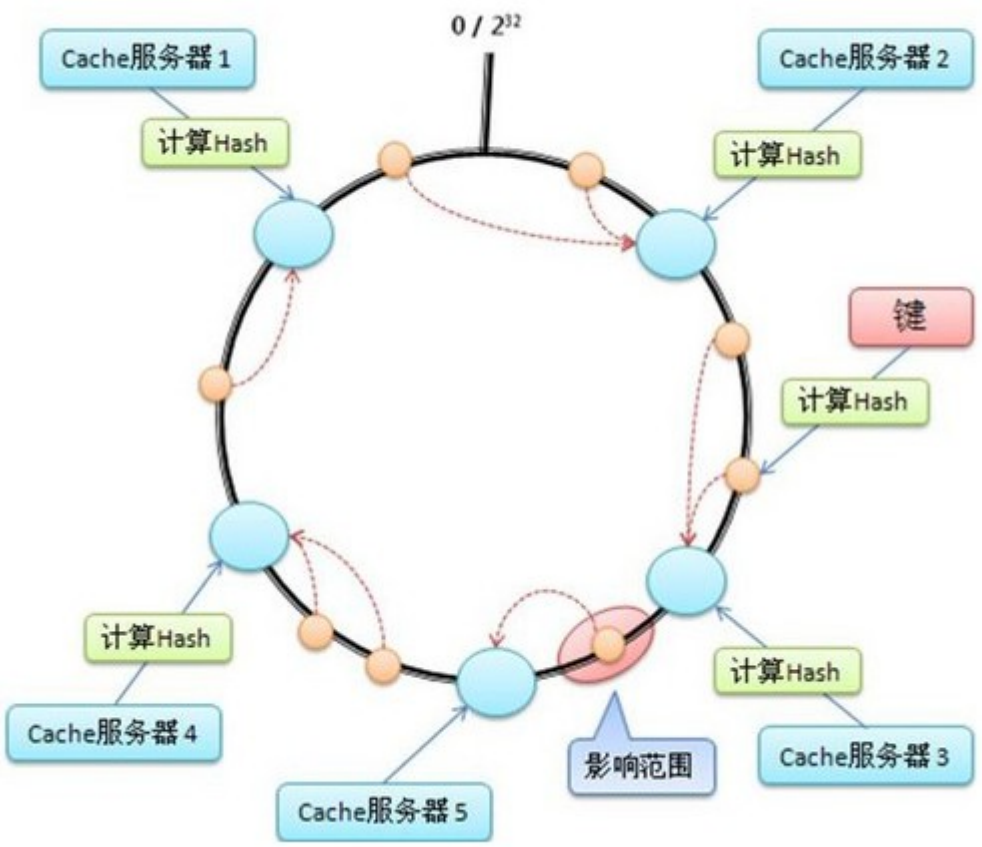
（2）把 key 也计算 hash 对应到这个环上。然后在这个环上进行匹配，看这个 key 和哪台机器匹配。

2.B 具体原理

cache 服务器为 redis 服务器

假设 key、机器计算完成 hash 后在换上对应的关系如下：

在 key 的 hash 对应的环上的位置沿着顺时针方向寻找第一个 redis 服务器，找到后就将(key-value)存入该机器，如下图的 key 存到 cache 服务器 3



这里的关键点是：当你增加/减少机器时，其他机器在环上的位置并不会发生改变。这样只有增加的那台机器、或者减少的那台机器附近的数据会失效(相对于 Hash 取余明显减少了对数据的单调性影响)，其他机器上的数据都还是有效的。

2.C 数据倾斜问题

上图中的 Cache 服务器在环上均匀分布，但这是理想情况，实际上当你机器不多的时候，很可能出现几台机器在环上面贴的很近，不是在环上均匀分布。这将会导致大部分数据，都会集中在某 1 台机器上。

为了解决这个问题，可以引入“虚拟机器”的概念，也就是说：1 台机器，我在环上面计算出多个位置。怎么弄呢？ 假设用机器的 ip 来 hash，我可以在 ip 后面加上几个编号, ip_1, ip_2, ip_3, … 把 1 台物理机器生个多个虚拟机器的编号。

数据首先映射到“虚拟机器上”，再从“虚拟机器”映射到物理机器上。因为虚拟机器可以很多，在环上面均匀分布，从而保证数据均匀分布到物理机器上面。

jedis 客户端就封装了一致性 hash 并且引入虚拟节点计算逻辑,160*weight 权重值处理虚拟节点计算,weight 默认值是 1.

3 JAVA 实现

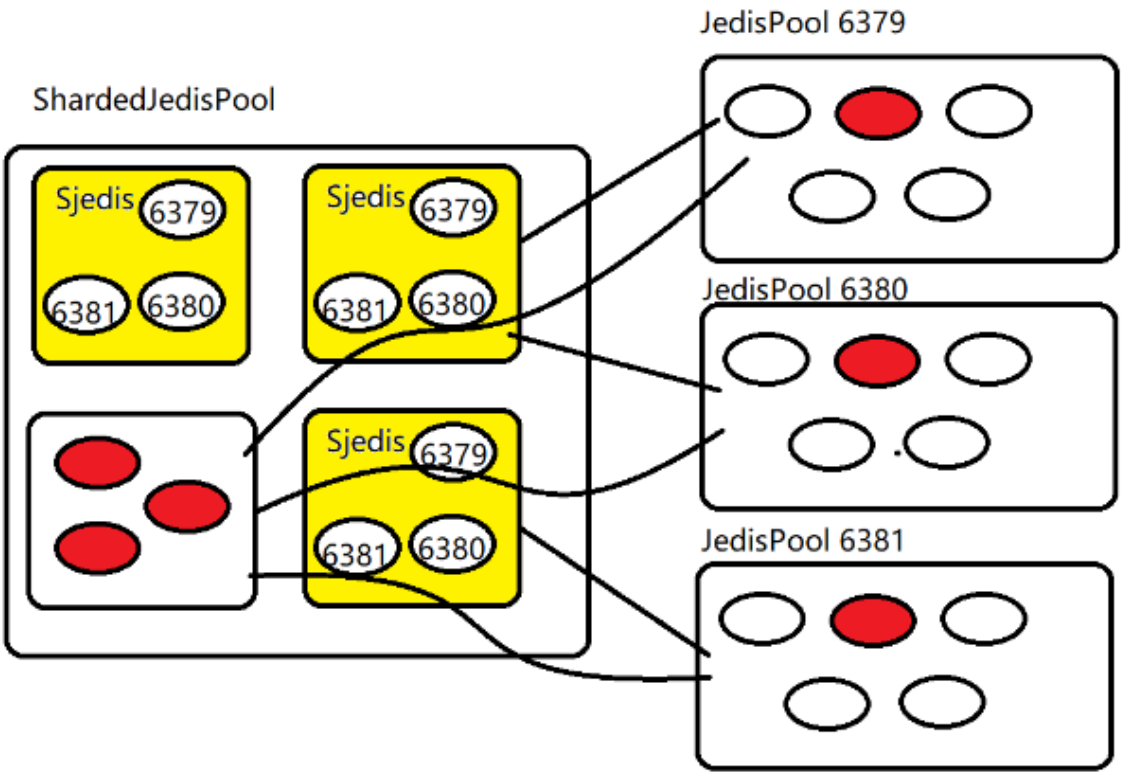
3.A 单个分片对象(SharedJedis)

```
@Test
public void test07(){
    //收集集群所有节点信息
    ArrayList<JedisShardInfo> jedisArrayList = new
ArrayList<JedisShardInfo>();
    jedisArrayList.add(new JedisShardInfo("10.42.175.170", 6379));
    jedisArrayList.add(new JedisShardInfo("10.42.175.170", 6380));
    jedisArrayList.add(new JedisShardInfo("10.42.175.170", 6381));

    //创建一个封装了一致性 hash 算法的分片对象
    ShardedJedis shardedJedis = new ShardedJedis(jedisArrayList);

    //生成 1000 key 看看数据分布情况
    for (int i = 0; i <1000 ; i++) {
        //随机生成 key
        String key=UUID.randomUUID().toString();
        //存数据
       shardedJedis.set(key,""+i);
    }
}
```

3.B 连接池(SharedJedisPool)




```
@Test
 public void test08(){
     //收集集群所有节点信息
    ArrayList<JedisShardInfo> jedisArrayList = new
```

```
ArrayList<JedisShardInfo>();
        jedisArrayList.add(new JedisShardInfo("10.42.175.170",
6379));
        jedisArrayList.add(new JedisShardInfo("10.42.175.170",
6380));
        jedisArrayList.add(new JedisShardInfo("10.42.175.170",
6381));

/*        //创建一个封装了一致性 hash 算法的分片对象
        ShardedJedis shardedJedis = new
ShardedJedis(jedisArrayList);*/
        //连接池配置
       JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();
        //创建一个封装了一致性 hash 算法的分片对象的连接池
       ShardedJedisPool shardedJedis = new
ShardedJedisPool(jedisPoolConfig,jedisArrayList);
        //生成 1000 Key 看看数据分布情况
       for (int i = 0; i <1000 ; i++) {
            //随机生成 key
            String key=UUID.randomUUID().toString();
            //存数据
          shardedJedis.getResource().set(key,""+i);
        }
    }
```

4 你能描述一下一致性 hash 吗

基础:hash 环,散列计算,任何计算机内存对象都能映射到这个[0-43 亿]区间

数据计算:redis 的客服务端节点,将节点信息映射大环中,将数据 key 值映射到环中

对应关系:key 的整数顺时针寻找最近节点.

数据平衡性:引入虚拟节点解决数据平衡性,每个真是节点都会生成一批虚拟节点,虚拟节点个数越多,平衡性越好.