# Redis介绍

### 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

### 特性

#### 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

#### 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

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

#### 内存运行

* redis在运行过程，数据在内存中处理
  + redis使用结构：包含2个 服务端运行的处理数据的进程

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

#### 分布式:

* 一个redis节点内存容量小，可以多启动几个redis，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

#### 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。

# Redis的安装目录

# 启动redis和redis脚本

### 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

### 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

# 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）

### 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

重要生产环境（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）

### 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。

### 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 |

### 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

### 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 |

### 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 |

### 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) |

# 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

### String类型

计算机生成了可选文字:
String
KEY
name
age
VALUE
wanglaoshi

#### 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](mhtml:file://E:\达内\互联网框架\day08-redisPM\笔记\day08.mht!https://redis.io/)

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 |

#### 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）

#### 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。

### ha**sh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。

计算机生成了可选文字:
redisHASH
KEY
user
VALUE
age
name
gen
18
wang
male
一个变量叫user
属性有agename
gender

#### 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 |

#### 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" |

#### 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" |

#### 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" |

#### 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 |

#### 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）

### 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）。

计算机生成了可选文字:
redisList
KEYVALUE
STU
0wang
1
上，头，左
下，尾，右

#### 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

#### 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" |

#### 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> |

#### 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

#### 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。

### 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

计算机生成了可选文字:
REDISSET
KEY
favor
VALUE
geo

#### 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

计算机生成了可选文字:
张三
王五
李四
、砾七
赵广
毛九
顾八
铁侠
国队长
蛛侠
王翠花
共同好友王五
陈七，毛九

#### sadd key member

新增元素到集合中。

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

#### 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" |

#### 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> |

#### 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> |

### 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）

计算机生成了可选文字:
redisZSET
KEY
SCOre
VALUE
《ao•an
caoan
haoxia
XIaO)(LIWeI
100
90
80
70

#### 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

#### 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 |

#### 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

1. zrank key member:看看member的排名

|  |
| --- |
| 127.0.0.1:6379> zrank score piaoqian  (integer) 2 |

1. 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> |

1. 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" |

#### zrem key member

将元素从zset类型的数据中删除

# 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")

### maven导入依赖

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

### redis读写测试

#### 客户端对象

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

#### 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"); } |

# 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

计算机生成了可选文字:
easymall 某 有 、 工 扌呈 
n eW Jedis 
easymall 某 有 、 工 扌呈 
n eW Jedis 
easymall 某 有 、 工 扌呈 
n eW Jedis 
easymall 某 有 、 工 扌呈 
n eW Jedis 
宽 占 用 
夂 口 
内 存 容 量 小 

### 相关概念

#### 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.

#### 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.

#### 重要 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

#### 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

#### 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

### 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

### 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

### 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);  } } |

#### 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.

#### 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); } |

# 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.

### 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)

### 计算原理

#### Hash环

虚拟的一个环，范围0-2^32-1

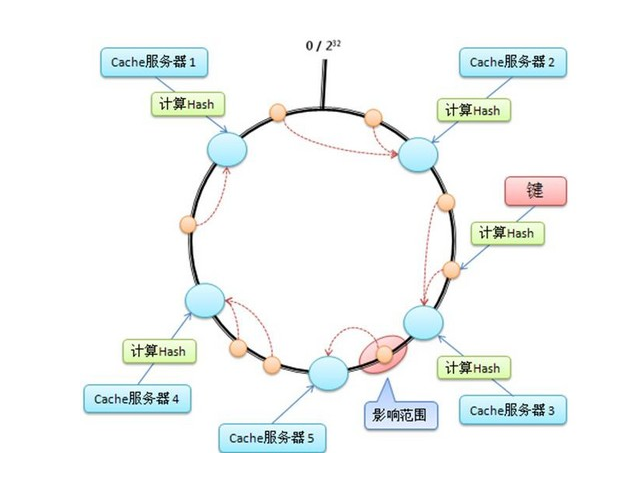
（1）把所有机器编号计算hash对应到这个环上。  
（2）把key也计算hash对应到这个环上。然后在这个环上进行匹配，看这个key和哪台机器匹配。

#### 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。

#### ****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。

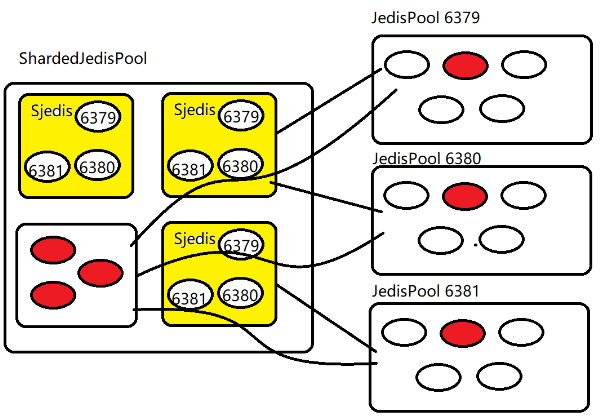
jedis客户端就封装了一致性hash并且引入虚拟节点计算逻辑,160\*weight权重值处理虚拟节点计算,weight默认值是1.

### JAVA实现

#### 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);  } } |

#### 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);  }  } |

### 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.