redis概括

2018年9月11日 8:56

redis定义:

nosql,key-value数据结构,可持久化,分布式的,缓存的,非关系型 数据库.

nosql: not only structured language 不仅结构化查询语言

结构化数据:一类相同结构的数据集体

非结构化数据:不具备相同特点,相同属性的一类数据集合.网 页数据.

关系型数据库:基于结构化数据,体现数据的关系的数据库 非关系型数据库:可以存储结构化数据,非结构化数据.

oracle mysql是典型的关系型数据库,表格,字段存储

mongoDB redis:key-value(key值可以形成一种自定义的 逻辑,存,取)

可持久化:

redis之前,memoryCache,分布式高可用的缓存技术.

与redis的区别:数据结构没有redis丰富,String,redis具备五 种String,hash,list,set,zset

momoryCache不可以持久化,内存数据一旦丢失,容易造成 雪崩

雪崩/缓存击穿:

海量的请求访问数据,最初来源一定是数据库,缓存内存积累海量数据,后续访问都是被缓存拦截使用.缓存数据一旦丢失, 海量请求涌入数据库,造成数据库宕机,如果在数据库恢复期

分区 day02all 的第 7页

海量请求涌入数据库,造成数据库宕机,如果在数据库恢复期间,缓存数据并未恢复,海量请求继续涌入数据库,数据库会重

复的宕机--重启,这种情况就叫做缓存的雪崩/缓存击穿.

停止系统功能访问.手动将数据从数据库导入恢复的缓存中.

缓存集群redis,memoryCache,都需要防止雪崩的出现 redis就是使用持久化策略,将内存数据保存在磁盘

memoryCache唯一办法就是缓存技术集群永不宕机(容灾). 性能上两种技术相似.

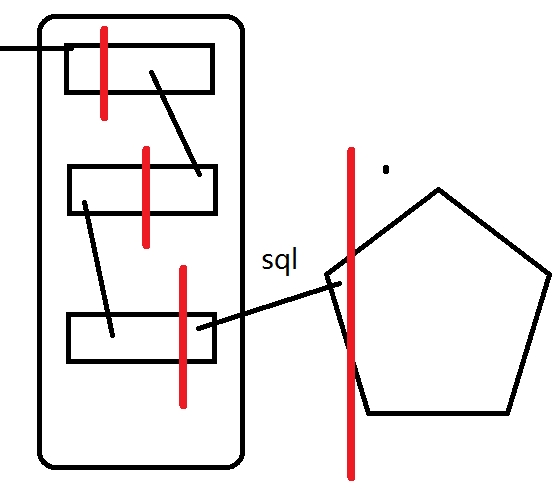
分布式:留到3个节点的分布式简单集群.

数据如何下发.

工程的缓存逻辑

我们京淘阶段的缓存都是在业务层使用

控制层,业务层,持久层,数据库

数据库:

key就是sql语句,同一条sql语句在第一次查询时,组织 resultSet,将sql作为key存储resultSet为value在缓存

分区 day02all 的第 8页

resultSet,将sql作为key存储resultSet为value在缓存

中.(ecache)

持久层:

减少链接数据库次数,减少resultSet数据封装对象的次数 业务层和控制层:

减少调用层次

redis的安装和启动

2018年7月10日 0:11

其他环境安装redis可能缺少插件

redis缺少编译套件GCC，测试套件TCL

问题1：make test 无效

缺少测试使用的tcl

yum -y intall tcl\* 解决问题

问题2：make 无效

缺少gcc

yum -y install gcc

然后清除redis根目录源编译内容

make distclean

然后重新编译安装

make && make install

下载安装包解压(1805云主机,虚拟机不需要上传安装包)

创建管理目录

[root@10-9-62-65 ~]# cd /home/

[root@10-9-62-65 home]# mkdir software

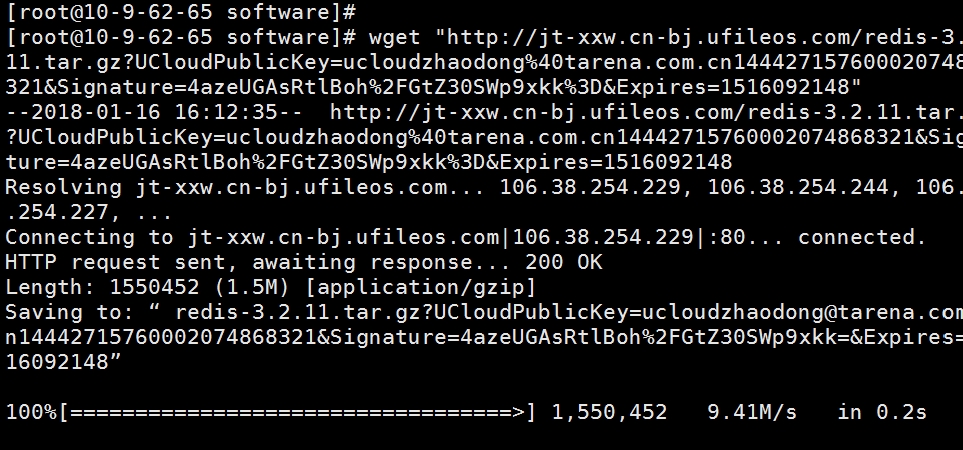
[root@10-9-62-65 home]# cd software/

[root@10-9-62-65 software]#

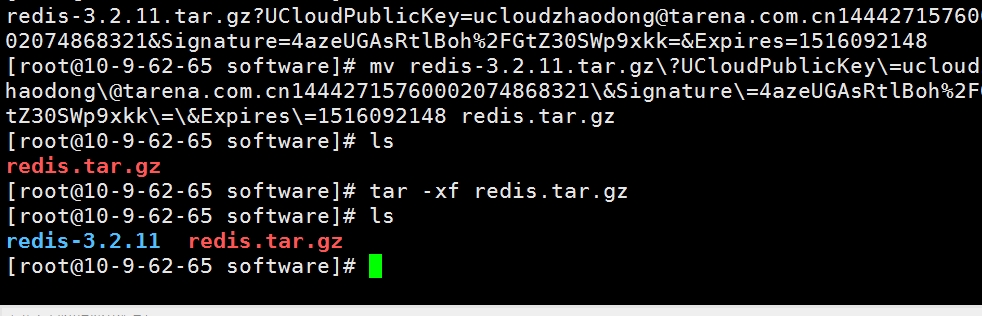
使用wget获取资源(不需要,直接从resources拷贝过来)

redis的tar包,拷贝到software进行解压

分区 day02all 的第 10页

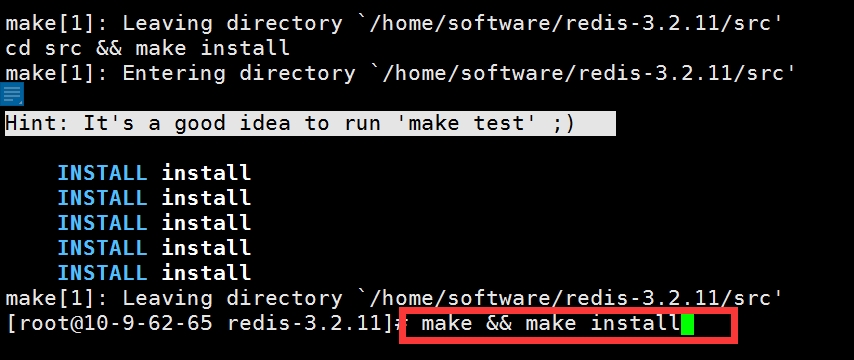
解压

#tar -xf 报名称

进入redis根目录

使用编译 编译安装

#make && make install

启动redis

启动redis的服务,默认启动,后面没有任何参数,启动配置是代码中写死的

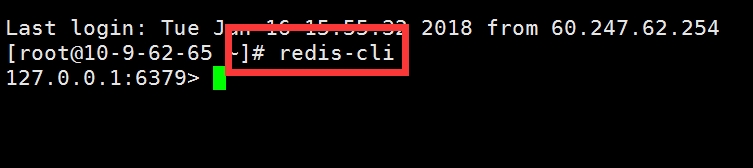
#redis-server

利用默认启动的命令，占用端口，后台守护进程，内部其他配置参数都是写死的

分区 day02all 的第 11页

使用redis需要启动redis客户端 -h指定登录ip 默认是127.0.0.1，-p 指定登录端口默认是6379

#redis-cli

如果想在同一个操作控制台启动服务和客户端

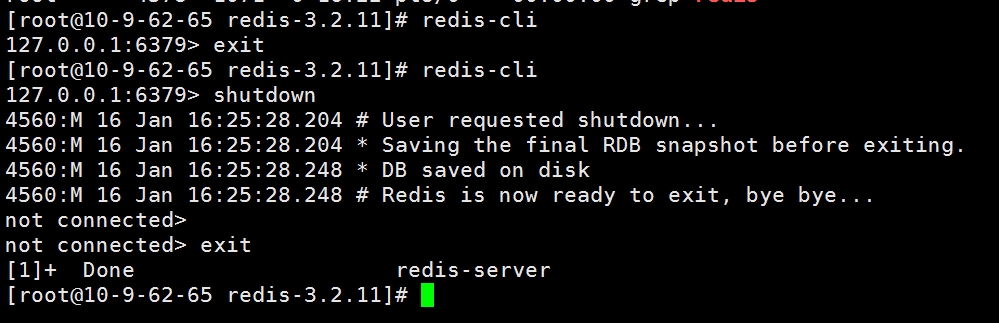
启动客户端时可以使用后台运行命令

#redis-server &

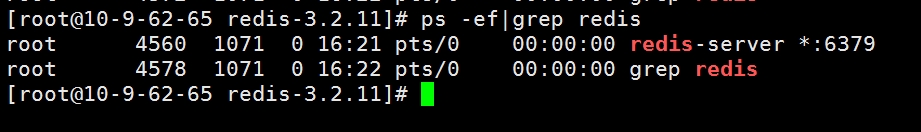
停止redis服务

1. 占用控制台的redis服务直接ctrl+c停止服务
2. 在后台运行服务的时候，登陆客户端

6379>shutdown；

1. 检查后台运行的redis服务

#ps -ef|grep redis;

redis-server表示redis服务

\*：表示能够访问当前redis服务的所有ip地址，都可以

如果列出一系列的ip地址，除这些ip意外的所有访问redis服务的请求

都被拒绝

分区 day02all 的第 12页

redis数据类型和命令

2018年9月11日 8:56

redis的五种数据结构:

String:字符串数据, hanlaoshi,天安门

hash :面向对象的数据. user--age--18

--name--韩老师 --city--北京

list :双向链表:左为上头:右为下尾

{"100","200","300"}

set :集合

zset:有序集合

五种数据结构,是针对的value存储的数据而言的.

String类型(基础名称)

keys \*:查询当前redis服务中存储的全部key值

set [key] [value]:存储数据,value是真正保存的数据,key是自定义的 方便管理数据的入口.

get [key]:获取key对应的value存储数据

select

exists

del

type&&help

分区 day02all的第 14页

flushall

flushdb

expire

pexpire

incr/decr

incrby/decrby

Save

Redis基础命令

2018年8月13日 8:30

另一个qq：3103723160

操作数据：redis具备五种数据类型

String

Hash

List

Set

Zset

String类型（基础命令）

**keys \*** :查看当前redis节点（启动的每一个redis服务 都叫redis节点。）所有已存在的key值。

127.0.0.1:6379> keys \*

(empty list or set)

127.0.0.1:6379> set name hanlaoshi

OK

127.0.0.1:6379> set location beijing

OK

127.0.0.1:6379> keys \*

1. "location"
2. "name"

127.0.0.1:6379>

**set key value:**存储一个key-value结构的数据 （String）,覆盖一个同名key值得数据

127.0.0.1:6379> set age 22

OK

127.0.0.1:6379> keys \*

分区 day02all 的第 16页

127.0.0.1:6379> keys \*

1. "location"
2. "age"
3. "name"

127.0.0.1:6379> get age

"22"

127.0.0.1:6379> set age 33

OK

127.0.0.1:6379> get age

"33"

127.0.0.1:6379>

**get key:**获取当前key对应的value数据

127.0.0.1:6379> get age

"22"

127.0.0.1:6379> set age 33

OK

127.0.0.1:6379> get age

"33"

127.0.0.1:6379>

**select整数：**redis中默认存在16个数据分库

（database），index号0-15，在一个服务器节点上，区分多种类型，多种功能的数据库仓库，默认登

录的是0号分库

127.0.0.1:6379> select 1

OK

127.0.0.1:6379[1]> keys \*

(empty list or set)

分区 day02all 的第 17页

(empty list or set)

127.0.0.1:6379[1]> select 0

OK

127.0.0.1:6379> keys \*

1. "location"
2. "age"
3. "name"

127.0.0.1:6379>

**exists key：**判断当前节点是否包含key的数据。

get也可以完成这个任务。redis在key-value结构的value存储容量是512M。使用get判断存在会先读在判

断数据是否为空，浪费资源

127.0.0.1:6379> exists haha kaka age

(integer) 1

127.0.0.1:6379> exists age

(integer) 1

127.0.0.1:6379> exists haha

(integer) 0

127.0.0.1:6379>

**del key:**删除key值对应的key-value数据；

127.0.0.1:6379> del age

(integer) 1

127.0.0.1:6379> del haha

(integer) 0

127.0.0.1:6379> keys \*

1. "location"
2. "name"

分区 day02all 的第 18页

2) "name"

127.0.0.1:6379>

**type key:查看当前key的类型**

127.0.0.1:6379> type name

string

127.0.0.1:6379>

**help命令名称：一般到官网查询详细讲解；按照案 例**

**save：将当前的内存数据保存到磁盘文件**

**（dump.rdb）**

127.0.0.1:6379> keys \*

1. "location"
2. "name"

127.0.0.1:6379> save

OK

127.0.0.1:6379>

**save前，save后观察redis根目录的dump.rdb大小可以 发现已经存储到持久化文件**

**flushall:将当前redis所有数据清洗，包括持久化文件，内存数据（全部的数据分库）**

127.0.0.1:6379> keys \*

1. "location"
2. "name"

127.0.0.1:6379> select 1

OK

分区 day02all 的第 19页

OK

127.0.0.1:6379[1]> set name wanglaoshi

OK

127.0.0.1:6379[1]> set age 11

OK

127.0.0.1:6379[1]> keys \*

1. "age"
2. "name"

127.0.0.1:6379[1]> save

OK

127.0.0.1:6379[1]> flushall

OK

127.0.0.1:6379[1]> keys \*

(empty list or set)

127.0.0.1:6379[1]> select 0

OK

127.0.0.1:6379> keys \*

(empty list or set)

127.0.0.1:6379>

**flushdb:清空一个分库的数据，不删除持久化文件内容（进行测试使用）**

127.0.0.1:6379[1]> flushdb

OK

127.0.0.1:6379[1]> keys \*

(empty list or set)

127.0.0.1:6379[1]> select 0

OK

127.0.0.1:6379> keys (

分区 day02all 的第 20页

127.0.0.1:6379> keys (

(empty list or set)

127.0.0.1:6379> keys \*

1) "name"

127.0.0.1:6379>

redis中string字符串如果是纯数字，也有一些简单的 数字操作命令

**incr decr key：**自增1，自减1；计步器

127.0.0.1:6379> set num 100

OK

127.0.0.1:6379> incr num

(integer) 101

127.0.0.1:6379> decr num

(integer) 100

127.0.0.1:6379>

**多步计步器：incrby decrby key 整数：对key的**

**value数字自增自减去整数步数**

**127.0.0.1:6379> incrby num 10**

(integer) 110

127.0.0.1:6379> decrby num 10

(integer) 100

127.0.0.1:6379>

**mset mget：**批量操作（不支持多节点分布式，只能 对本节点数据进行处理）

127.0.0.1:6379> mset num1 200 num2 300

num3 400

分区 day02all 的第 21页

num3 400

OK

127.0.0.1:6379> keys \*

1. "num3"
2. "name"
3. "num"
4. "100"
5. "num1"
6. "num2"

127.0.0.1:6379> mget num1 num2 num3

1. "200"
2. "300"
3. "400"

127.0.0.1:6379>

LRU自动删除策略:如果redis内存达到自动删除启动瓶 颈 Last Recent Unused最近最久未使用key

**expire key:对key添加超时过期的设定**

**配合ttl key可以查看 key值剩余时间,已经超时的内容 删除，ttl剩余时间是-2 -1表示永久数据**

127.0.0.1:6379> ttl num1

(integer) -2

127.0.0.1:6379> keys \*

1. "num3"
2. "name"
3. "num"
4. "100"
5. "num2"

127.0.0.1:6379> ttl name

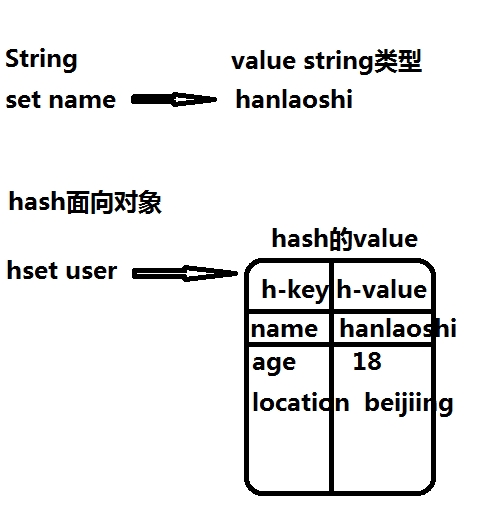
分区 day02all 的第 22页

127.0.0.1:6379> ttl name

(integer) -1

127.0.0.1:6379>

Hash数据类型

**hset key field value:key是 相当于变量名，field是value中的key-value对中的key，但是面向对象时表示一个对象的属**

**性名称， value表示属性的值**

127.0.0.1:6379> hset user username hanlaoshi

(integer) 1

127.0.0.1:6379> hset user age 18

(integer) 1

127.0.0.1:6379> hset user location beijing

(integer) 1

127.0.0.1:6379> type user

hash

127.0.0.1:6379>

分区 day02all 的第 23页

127.0.0.1:6379>

**hget key field**

127.0.0.1:6379> hget user username

"hanlaoshi"

127.0.0.1:6379>

**hexists key field:判断hash类型中的属性值是否存在**

127.0.0.1:6379> hexists user haha

(integer) 0

127.0.0.1:6379> hexists user age

(integer) 1

127.0.0.1:6379>

**hmset和hmget：批量设置，不支持分布式**

127.0.0.1:6379> hmset student age 19 name

wangxiaoxiao gender male

OK

127.0.0.1:6379> hmget student age name gender

1. "19"
2. "wangxiaoxiao"
3. "male"

127.0.0.1:6379>

**Hdel key field:删除属性和值**

127.0.0.1:6379> hdel student age

(integer) 1

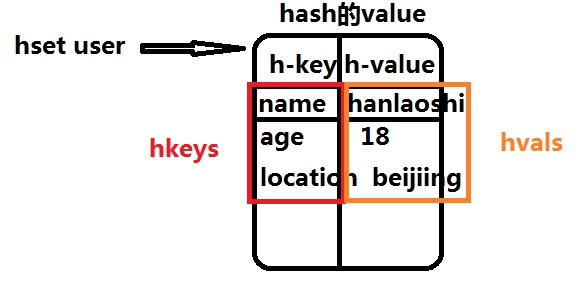
127.0.0.1:6379> hget student age

(nil)

127.0.0.1:6379>

**Hkeys Hvals:单独获取对象的属性名，或者属性的值**

分区 day02all 的第 24页

127.0.0.1:6379> hkeys user

1. "name"
2. "username"
3. "age"
4. "location"

127.0.0.1:6379> hvals user

1. "haha"
2. "hanlaoshi"
3. "18"
4. "beijing"

127.0.0.1:6379>

**Hlen key：获取属性个数（长度）**

127.0.0.1:6379> hlen user

(integer) 4

127.0.0.1:6379> hlen student

(integer) 2

127.0.0.1:6379>

**Hincreby:和String类型特性区别的一个体现**

String对应存储对象数据：需要在redis中修改对象某个属性的数值（10-15）

{“id”:1623125,"age"：10，“name”:hanlaoshi}

分区 day02all 的第 25页

{“id”:1623125,"age"：10，“name”:hanlaoshi}

不能直接在redis中获取age将10变成15，

从redis读出来，转化成对象，对象set属性，重新放 回redis

和hash存储对象数据

user id 1623125

age 10

name hanlaoshi

Hincreby user name 5

List链表数据结构

**list01--{“200”,"300","400"}**

查看链表list和设置链表数据Lpush（左表示上，右表 示下）

**lrange key起始下标 结束下标（展示范围，结束下标**

**是-1表示到尾部）**

127.0.0.1:6379> lpush mylist01 100 200 300

(integer) 3

127.0.0.1:6379> lrange mylist 0 1

(empty list or set)

127.0.0.1:6379> lrange mylist01 0 1

1. "300"
2. "200"

127.0.0.1:6379>

**Rpush key value：**从下向上插入数据

127.0.0.1:6379> rpush mylist01 400

(integer) 4

分区 day02all 的第 26页

(integer) 4

127.0.0.1:6379> lrange mylist01 0 -1

1. "300"
2. "200"
3. "100"
4. "400"

127.0.0.1:6379>

Linsert：从链表中间操作数据,从左侧找到第一个相同 元素插入数据（before after）

linsert mylist01 before 100 one

在mylist01链表中，找到100这个元素，从前插入数 据one 300 200 one 100 400

127.0.0.1:6379> linsert mylist01 before 100 one (integer) 5

127.0.0.1:6379> lrange mylist01 0 -1

1. "300"
2. "200"
3. "one"
4. "100"
5. "400"

127.0.0.1:6379> lpush mylist01 100

(integer) 6

127.0.0.1:6379> lrange mylist01 0 -1

1. "100"
2. "300"
3. "200"
4. "one"
5. "100"

分区 day02all 的第 27页

1. "100"
2. "400"

127.0.0.1:6379> linsert mylist01 after 100 two

(integer) 7

127.0.0.1:6379> lrange mylist01 0 -1

1. "100"
2. "two"
3. "300"
4. "200"
5. "one"
6. "100"
7. "400"

127.0.0.1:6379>

**lset key index value将下标对应的数据修改**

127.0.0.1:6379> lset mylist01 1 2

OK

127.0.0.1:6379> lrange mylist01 0 -1

1. "100"
2. "2"
3. "300"
4. "200"
5. "one"
6. "100"
7. "400"

127.0.0.1:6379>

Lrem:这个没有rrem，删除操作

lrem key count value：从左向右寻找count个value

分区 day02all 的第 28页

lrem key count value：从左向右寻找count个value 相同值元素删除。

count>0从左往右删

count=0 全删

count<0从右往左删

127.0.0.1:6379> lpush mylist01 200 200 200

(integer) 9

127.0.0.1:6379> lrange mylist01 0 -1

1. "200"
2. "200"
3. "200"
4. "2"
5. "300"
6. "200"
7. "one"
8. "100"
9. "400"

127.0.0.1:6379> lrem mylist01 -2 200

(integer) 2

127.0.0.1:6379> lrange mylist01 0 -1

1. "200"
2. "200"
3. "2"
4. "300"
5. "one"
6. "100"
7. "400"

127.0.0.1:6379> lrem mylist01 0 200

(integer) 2

分区 day02all 的第 29页

(integer) 2

127.0.0.1:6379>

ltrim保留链表的范围内的数据元素

ltrim key 起始位置 结束位置（-1表示到尾部）

127.0.0.1:6379> ltrim mylist01 4 6

OK

127.0.0.1:6379> lrange mylist01 0 -1

1) "400"

127.0.0.1:6379>

两个链表的数据交互

lpop从list头部删除元素，返回删除结果（类似remove）

127.0.0.1:6379> lrange mylist01 0 -1

1. "six"
2. "five"
3. "four"
4. "three"
5. "two"
6. "one"
7. "400"

127.0.0.1:6379> lpop mylist01

"six"

127.0.0.1:6379> lrange mylist01 0 -1

1. "five"
2. "four"
3. "three"
4. "two"

分区 day02all 的第 30页

1. "two"
2. "one"
3. "400"

127.0.0.1:6379>

rpoplpush：从第一个list的尾部删除数据，移动到第 二个list的头部添加

127.0.0.1:6379> rpoplpush mylist01 mylist02

"400"

127.0.0.1:6379> lrange mylist02 0 -1

1. "400"
2. "5"
3. "4"
4. "3"
5. "2"
6. "1"

127.0.0.1:6379>

SET类型和Zset（集合和有序集合）

作业：自行完成。

分区 day02all 的第 31页

redis多实例部署

2018年7月10日 0:14

单实例单线程的redis进程不足以高效率使用cpu和内存资源，所以一般来讲redis在同一台机器上要启动多个进程完

成多实例部署；默认占用6379的情况下无法完成直接的3 个实例启动，这里我们需要了解如何通过指定配置文件，将多实例部署在linux上

启动redis服务的命令redis-server 没有加载任何配置文件指定各种各样的配置信息（端口指定，ip绑定，后台运

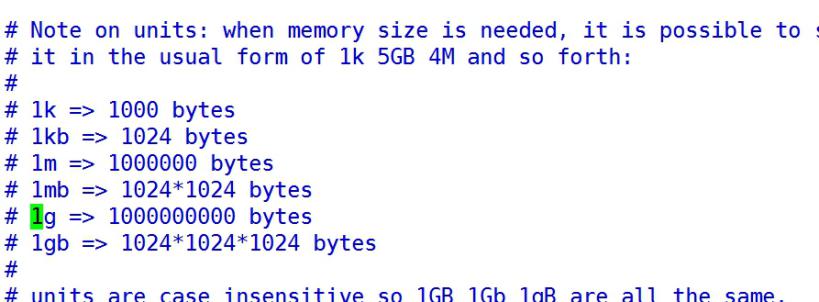
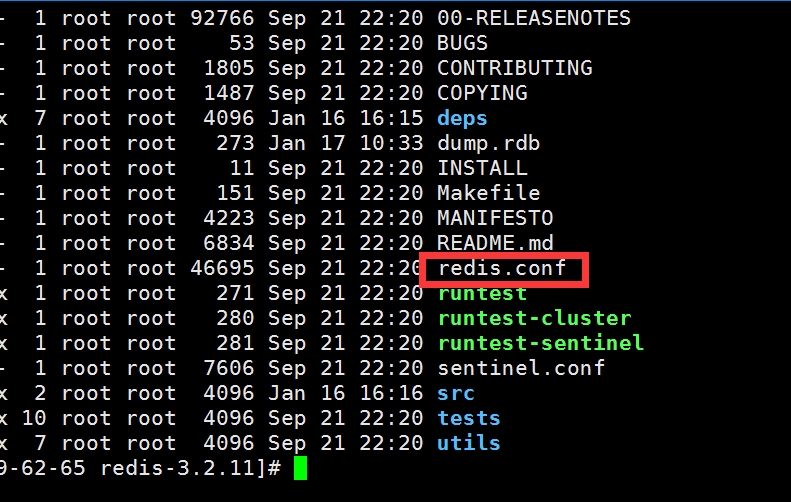
行）

例如在根目录存在一个配置文件的模板（大部分与默认启动的配置相同）redis.conf

#redis-server配置文件的名称

配置文件（在启动时指定的配置文件，核心的配置文件）

/redis根目录/redis.conf

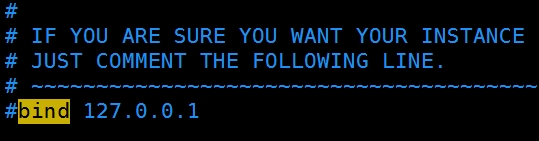
一个redis实例默认占用所有物理内存（上限是物理内存大小）,在实际使用中需要限制大小

配置文件的修改内容

p61 bind 用#注释掉

分区 day02all 的第 32页

p61 bind 用#注释掉

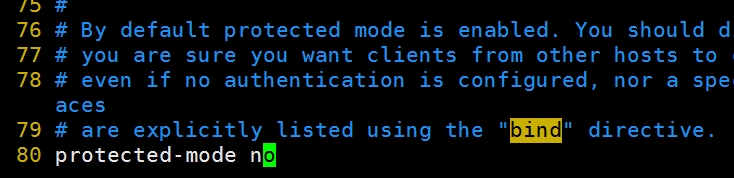
如果需要绑定监听的ip（客户端只有通过被绑定的ip才可以利用

redis-cli -h ip地址链接服务器）

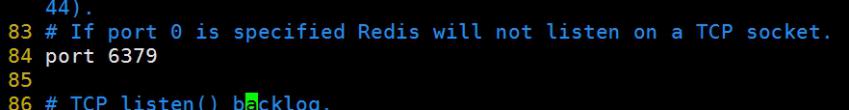
bing 127.0.0.1 106.75.101.219(外网可访问当前服务器的ip)

一旦用#注释bind，没有任何限制，只要可以链接服务器，都允许使用redis

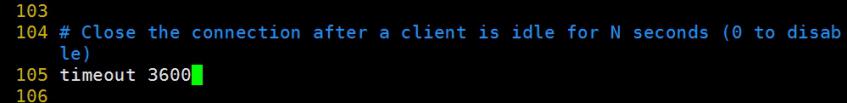
p80 保护模式不启动

保护模式开启，需要登录密码，改成no

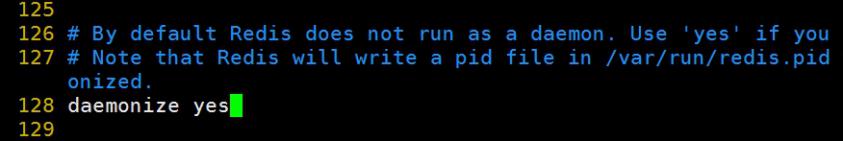
p84 6379是默认端口(要启动其他的redis实例需要修改端口)

p105 当客户端空闲时间达到一小时,就会自动断开连接,0秒表示

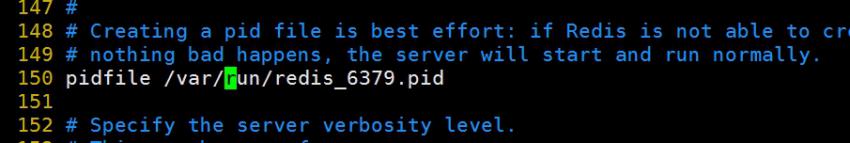
不启用超时配置

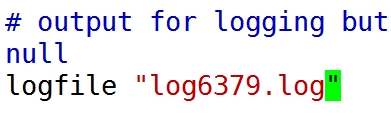
p128 daemonize 设置成yes让redis服务器启动有守护进程管理

(后台执行)

p150 对应不同的redis实例,pid的文件名称需要和端口同名

每个进程在linux或者其他操作系统中都会占用pid号，当系统中的进程过多时，需要查找redis进程号可能比较麻烦，直接打开pid文件查看即可

P163logfile 需要指定，利用端口号命名，放到redis根目录

save 900(秒) 1(变动的数据条数)

当900以内,至少有1条数据变动,看是flush保存数据到文件

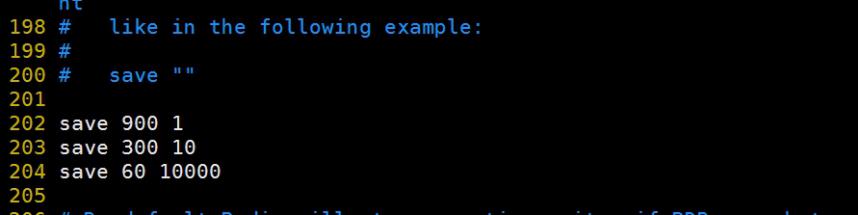
save 300 10

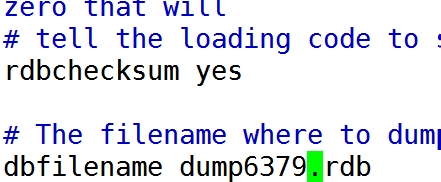
分区 day02all 的第 33页

save 300 10

300秒以内至少10条数据变动,保存文件

save 60 10000

P237左右，指定dump的持久化文件，每个服务单独指向一个文件，重启时，数据不会错乱

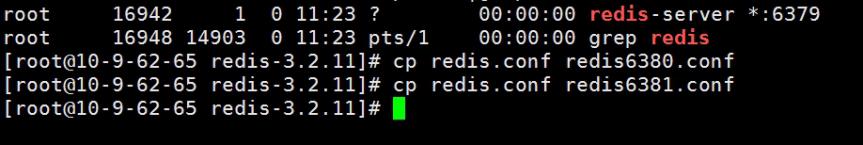
启动第二和第三个redis实例

redis-server redis.conf(指定启动文件)

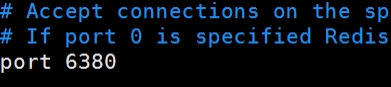
需要第二个实例的配置文件

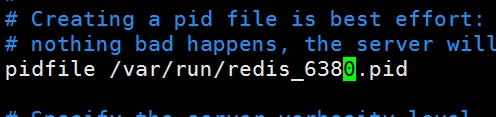
需要第三个实例的配置文件

拷贝redis.conf,用redis6380.conf,redis6381.conf

将拷贝的文件中只修改与端口有关内容

port

pid文件

6381的略

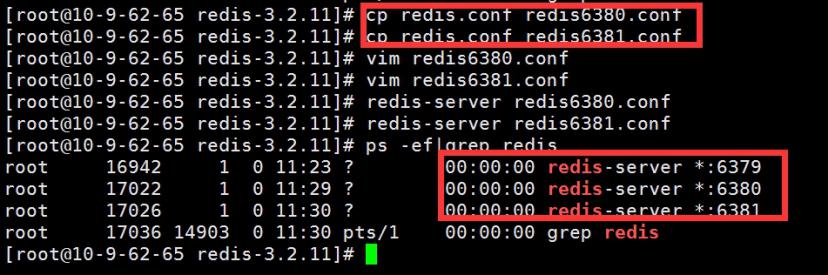
启动另外两个节点

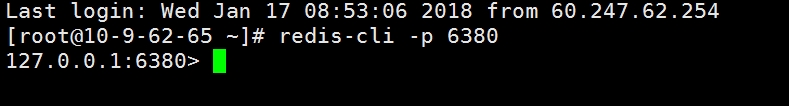
#redis-server redis6380.conf

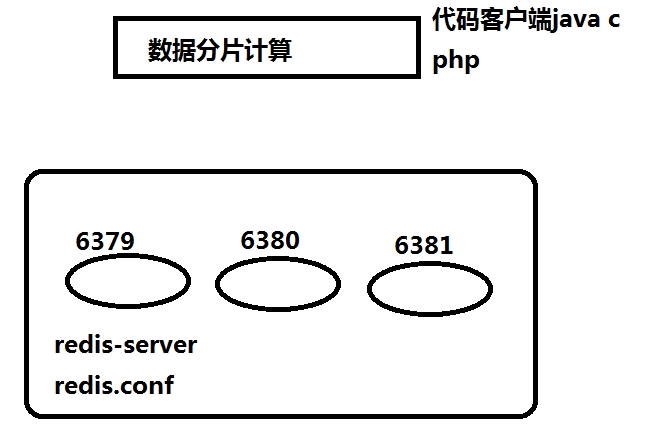
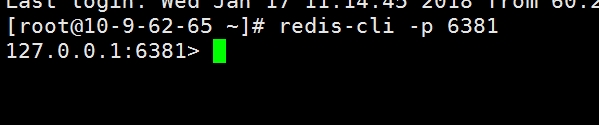
#redis-server redis6381.conf

#ps -ef|grep redis

分区 day02all 的第 34页

指定端口登录客户端redis-cli -p [端口号] -h [ip] #redis-cli -p 6380

#redis-cli -p 6381

分区 day02all 的第 35页

redis的java客户端代码jedis

2018年9月11日 16:34

利用jedis客户端段,实现代码操作redis服务

pom的依赖

1. jedis链接redis服务

//1 使用jedis链接redis的服务

@Test

public void test01(){

//通过生成一个jedis对象,构造方法时传入ip 端口等内容 Jedis jedis=new Jedis("106.75.120.140",6379);

jedis.set("school", "tedu");

System.out.println(jedis.get("school"));

}

1. 代码模拟缓存获取商品信息的步骤

//2 模拟缓存逻辑在系统中的步骤

@Test

public void test02(){

System.out.println("用户开始请求访问商品:1238126");

//生成一个可以操作的全局唯一的key值,存储对应商品信 息的key-

//value就是商品数据

String key="Item\_"+"1238122";

String value="";

分区 day02all 的第 36页

//判断key是否在缓存存在,如果存在,不需要访问数据库, 直接返回数据

//如果不存在,需要访问数据库,从数据库获取数据后存入

缓存,等待后续访问使用

Jedis jedis=new Jedis("106.75.120.140",6379);

if(jedis.exists(key)){

System.out.println("商品信息,由缓存拦截,用户获取 商品信息");

value=jedis.get(key);

System.out.println("商品信息:"+value);

}else{

System.out.println("缓存数据未命中");

System.out.println("从数据库获取商品信息"); value="考面包机";

System.out.println("数据库商品:"+value);

//将数据存到缓存供后续访问

jedis.set(key, value);

}

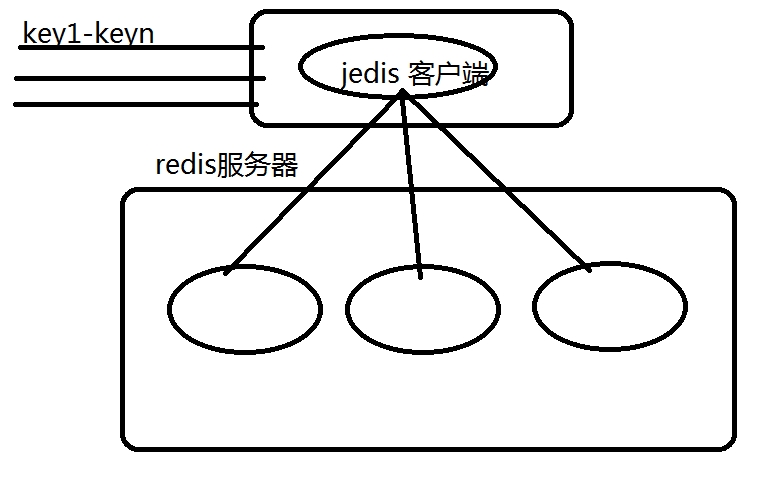
}

1. 数据分片之自定义分片计算

上两步操作没有涉及到6380 6381的操作

如果使用3个节点来存储大量数据,分布式存储.

分区 day02all 的第 37页

海量数据,经过代码客户端,需要下发到不同的redis节点进行存储,需要一种下分数据的计算逻辑,这种逻辑的不同,会导致最终3个节点存储的数据结果不一样;这种下分的逻辑实现在代码中 是一套计算方法,在数据的分布式结构中称之为数据分片的计 算.

//实现自定义的数据分片计算

@Test

public void test03(){

//数据分片计算需要根据数据,key值得取值不同,定义 对应的计算逻辑

//for循环模拟key-value数据对的海量生成 100条

Jedis jedis1=new Jedis("106.75.120.140",6379); Jedis jedis2=new Jedis("106.75.120.140",6380); Jedis jedis3=new Jedis("106.75.120.140",6381); for(int i=0;i<100;i++){

String key="item\_key\_"+i;

String value="item\_value\_"+i;

//如何将这100条下分

if(i<30){//存到6379

jedis1.set(key, value);

分区 day02all 的第 38页

jedis1.set(key, value);

}else if(i<66){//6380

jedis2.set(key, value);

}else if(i<100){

jedis3.set(key, value);

}else{

jedis1.set(key, value);

}}}

如何实现自定义数据分片:

1. 需要对你的数据有所了解
2. key值在不同的领域,取值范围是不同的.
3. 数据量必须固定才可以实现负载均衡(因为计算逻辑有问题)

自定义数据分片的缺点:

一旦数据增加,会造成某个,某几个节点的压力过大(数据倾 斜)

key值取值范围变动,计算方法也需要随之变动.

一旦数据变动,代码改,key值取值范围变动,代码也得变动

1. 数据分片之hash取余计算
2. jedis的数据分片
3. jedis的数据分片连接池

分区 day02all 的第 39页

分区 day02all 的第 40页