# Redis分片技术

## 1.分片的自定义计算(hash取余算法)

单个redis的内存容量上限.很容易达到瓶颈,即使配置数据时使用超时逻辑，也会因为单机down掉导致系统无法登录。

可以利用redis的分布式集群;

分布式技术引入项目之后.要考虑一个问题.数据如何划分

在分布式结构中,如何将数据划分给不同的节点.进行分布式处理的过程—也就是数据分片的计算过程

下面引入一个分布式简单算法完成

当前引入的算法:hash取余算法(利用hashCode完成)

### 1.2 hash取余的基础特性

#### 1.取hashcode

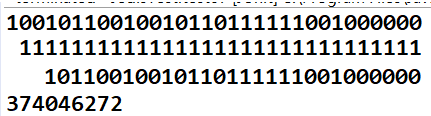
String key,无论key的值是什么

key.hashCode()=int类型的整数(可正可负)

只要key不发生变化(对象的equals方法判断相等),hashCode结果是不发生变动的;

#### 2.取正

* key.hashCode()&Integer.MAX\_VALUE;
* 2个整数值做位的与运算,31位1组成的二进制数对key的hash散列结果(可正可负)做位的与运算,结果必定是正数(31位的保真运算)



#### 3.取余

* (key.hashCode()&Integer.MAX\_VALUE)%3
* 上述公式的结果是 [0,1,2]
* 可以利用hash取余算法,对多个节点的redis集群进行数据分片的计算。其中节点个数就是取余的值N。
* 利用hash取余算法封装好对多个节点的分布式数据分片计算就能实现分布式的存,读,延时,判断存在等操作;

// 引入了多个节点时.如何对海量生成的数据进行分配

@Test

**public** **void** test06() {

// 模拟海量数据生成 key=value

Jedis jedis1 = **new** Jedis("192.168.243.133", 6379);

Jedis jedis2 = **new** Jedis("192.168.243.133", 6380);

Jedis jedis3 = **new** Jedis("192.168.243.133", 6381);

**for** (**int** i = 0; i < 10000; i++) {

String key = UUID.*randomUUID*().toString();

String value = "value\_" + i;

// 根据计算划分数据逻辑代码调用不同的jedis

// 将数据存储在不同的redis服务端,定义的计算逻辑

// 需要在存取,判断存在等方法中完全一致;

// 利用hash取余的公式,每一个key都能得到0,1,2

Integer result = (key.hashCode() & Integer.***MAX\_VALUE***) % 3;

**if** (result == 0) {

jedis1.set(key, value);

}

**if** (result == 1) {

jedis2.set(key, value);

}

**if** (result == 2) {

jedis3.set(key, value);

}

}

}

@Test

**public** **void** test07() {

// 4900e409-4873-4c25-8f37-11d6a3d33e64 =-1773437376

// 73a8f507-26c3-4f3b-a211-2ddb1699891c = 263652173

// String key = UUID.randomUUID().toString();

// System.out.println(key);

Integer hashResult = "4900e409-4873-4c25-8f37-11d6a3d33e64".hashCode();

// System.out.println(hashResult);

System.***out***.println("hashResult-bin:");

System.***out***.println(Integer.*toBinaryString*(hashResult));

System.***out***.println(Integer.*toBinaryString*(Integer.***MAX\_VALUE***));

Integer result = hashResult & Integer.***MAX\_VALUE***;

System.***out***.println(Integer.*toBinaryString*(result));

System.***out***.println(result);

}

图形用户界面, 文本, 应用程序

描述已自动生成

## 2.jedis自带的分片对象

jedis实现底层的hash一致性算法,利用一个对象收集多个节点信息,通过底层hash一致性算法封装,可以跟jedis对象调用方法一样.实际上已经做了hash一致性计算。

我们只要获取一个封装了算法的对象ShardedJedis分片对象即可;

hash一致性算法优点远远大于hash取余算法

@Test

**public** **void** test08() {

// 收集所有节点信息

List<JedisShardInfo> infoList = **new** ArrayList<JedisShardInfo>();

// 将3个节点信息.add到infoList总

infoList.add(**new** JedisShardInfo("192.168.243.133", 6379));

infoList.add(**new** JedisShardInfo("192.168.243.133", 6380));

infoList.add(**new** JedisShardInfo("192.168.243.133", 6381));

ShardedJedis sJedis = **new** ShardedJedis(infoList);

sJedis.set("name", "许老师");

System.***out***.println(sJedis.get("name"));

}

## 3.分片连接池

// 可以创建分片连接池,ShardedJedis对象从连接池获取连接资源

@Test

**public** **void** test09() {

// 收集节点信息

List<JedisShardInfo> infoList = **new** ArrayList<JedisShardInfo>();

// 将3个节点信息.add到infoList总

infoList.add(**new** JedisShardInfo("10.9.100.26", 6379));

infoList.add(**new** JedisShardInfo("10.9.100.26", 6380));

infoList.add(**new** JedisShardInfo("10.9.100.26", 6381));

// 创建一个连接池的配置对象,定义最大连接数

// 最小空闲,最大空闲,最大等待时长

GenericObjectPoolConfig config = **new** GenericObjectPoolConfig();

config.setMaxTotal(200);

config.setMaxIdle(8);

config.setMinIdle(2);

// 配置对象.收集对象创建连接池

ShardedJedisPool pool = **new** ShardedJedisPool(config, infoList);

// 每次操作redis集群时,从pool来获取资源

ShardedJedis sJedis = pool.getResource();

// sJedis.set("location", "华南师范大学南海校区");

System.***out***.println(sJedis.get("location"));

pool.~~returnResource~~(sJedis);

}

Jedis客户端生成连接池，ShardedJedis分片对象通过hash一致性算法进行数据操作。

