

# V8.0 Redis课程

**课程目标**

* 能够掌握Redis不同数据类型操作
* 能够使用Java API操作Redis
* 能够理解Redis的两种持久化方式
* 能够理解Redis的主从复制架构
* 能够理解Redis的Sentinel架构
* 能够理解Redis集群架构

V8.0 Redis课程 1

1. NoSQL数据库发展历史简介 4

1.1 web系统变迁历史 4

1.2 什么是NoSQL 5

1.3 NoSQL的特点 5

1.4 NoSQL数据库 6

2. Redis介绍 7

2.1 Redis的基本介绍 7

2.2 Redis的应用场景 7

2.3 Redis的特点 8

3. Redis单机环境安装 9

3.1 Windows版Redis安装（了解） 9

3.2 Linux版Redis安装 10

3.3 Redis Desktop Manager 13

4. Redis的数据类型 13

4.1 对字符串string的操作 14

4.2 对hash列表的操作 16

4.3 对list列表的操作 17

4.4 对set集合的操作 20

4.5 对key的操作 22

4.6 对ZSet的操作 23

4.7 对位图BitMaps的操作 24

4.8 对HyperLogLog结构的操作 27

5. Redis Java API操作 30

5.1 创建maven工程并导入依赖 30

5.2 创建包结构和类 31

5.3 连接以及关闭redis客户端 31

5.4 操作string类型数据 32

5.5 操作hash列表类型数据 33

5.6 操作list类型数据 34

5.7 操作set类型的数据 34

6. Redis的持久化 35

6.1 RDB持久化方案 35

6.2 AOF持久化方案 37

6.3 RDB or AOF 39

7. Redis 高级使用 39

7.1 Redis 事务 39

7.2 Redis 过期策略 42

7.3 内存淘汰策略 43

8. Redis的主从复制架构 44

8.1 另外两台服务器安装Redis 45

8.2 启动Redis服务 47

9. Redis中的Sentinel架构 47

9.1 Sentinel介绍 47

9.2 配置哨兵 49

9.3 Redis的sentinel模式代码开发连接 52

10. Redis 集群 54

10.1 引言 54

10.2 Redis Cluster 设计 55

10.3 Redis Cluster 搭建 57

10.4 启动Redis服务 62

10.5 Redis Cluster 管理 68

10.6 JavaAPI操作redis集群 69

11. Redis高频面试题 71

11.1 缓存穿透 71

11.2 缓存击穿 72

11.3 缓存雪崩 72

12. 常见问题 74

## NoSQL数据库发展历史简介

### 什么是NoSQL

* NoSQL最常见的解释是“non-relational”， 很多人也说它是“Not Only SQL”
* NoSQL仅仅是一个概念，泛指非关系型的数据库
* 区别于关系数据库，它们不保证关系数据的ACID特性
* NoSQL是一项全新的数据库革命性运动，提倡运用非关系型的数据存储，相对于铺天盖地的关系型数据库运用，这一概念无疑是一种全新的思维的注入

### NoSQL的特点

#### 应用场景

* 高并发的读写
* 海量数据读写
* 高可扩展性
* 速度快

#### 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）

### NoSQL数据库

#### memcache

* 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介绍

* 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string、 list、set、hash、zset等。
* 一般是作为缓存数据库辅助持久化的数据库
* 现在市面上用得非常多的一款内存数据库

## Redis介绍

redis官网地址：

<https://redis.io/>

中文网站

<http://www.redis.cn/>

### Redis的基本介绍

* Redis是当前比较热门的NoSQL系统之一
* 它是一个开源的、使用ANSI C语言编写的**key-value**存储系统（区别于MySQL的二维表格形式存储）

### Redis的应用场景

#### 取最新N个数据的操作

比如典型的取网站最新文章，可以将最新的5000条评论ID放在Redis的List集合中，并将超出集合部分从数据库获取

#### 排行榜应用，取TOP N操作

这个需求与上面需求的不同之处在于，前面操作以时间为权重，这个是以某个条件为权重，比如按顶的次数排序，可以使用Redis的sorted set，将要排序的值设置成sorted set的score，将具体的数据设置成相应的value，每次只需要执行一条ZADD命令即可。

#### 需要精准设定过期时间的应用

比如可以把上面说到的sorted set的score值设置成过期时间的时间戳，那么就可以简单地通过过期时间排序，定时清除过期数据了，不仅是清除Redis中的过期数据，你完全可以把Redis里这个过期时间当成是对数据库中数据的索引，用Redis来找出哪些数据需要过期删除，然后再精准地从数据库中删除相应的记录。

#### 计数器应用

Redis的命令都是原子性的，你可以轻松地利用INCR，DECR命令来构建计数器系统。

#### Uniq操作，获取某段时间所有数据排重值

这个使用Redis的set数据结构最合适了，只需要不断地将数据往set中扔就行了，set意为集合，所以会自动排重。

#### 缓存

将数据直接存放到内存中，性能优于Memcached，数据结构更多样化。

### Redis的特点

* 高效性
  + Redis读取的速度是110000次/s，写的速度是81000次/s
* 原子性
  + Redis的所有操作都是原子性的，同时Redis还支持对几个操作全并后的原子性执行。
* 支持多种数据结构
  + string（字符串）
  + list（列表）
  + hash（哈希）
  + set（集合）
  + zset(有序集合)
* 稳定性：持久化，主从复制（集群）
* 其他特性：支持过期时间，支持事务，消息订阅。

## Redis单机环境安装

### Linux版Redis安装

#### 下载redis安装包

node1.itcast.cn服务器执行以下命令下载redis安装包

|  |
| --- |
| cd /export/software  wget http://download.redis.io/releases/redis-3.2.8.tar.gz |

#### 解压redis压缩包到指定目录

node1.itcast.cn执行以下命令进行解压redis

|  |
| --- |
| cd /export/software  tar -zxvf redis-3.2.8.tar.gz -C ../server/ |

#### 安装C程序运行环境

node1.itcast.cn执行以下命令安装C程序运行环境

|  |
| --- |
| yum -y install gcc-c++ |

#### 安装较新版本的tcl

下载安装较新版本的tcl

##### 使用压缩包进行安装

node1.itcast.cn执行以下命令下载tcl安装包

|  |
| --- |
| cd /export/softwares  wget http://downloads.sourceforge.net/tcl/tcl8.6.1-src.tar.gz  解压tcl  tar -zxvf tcl8.6.1-src.tar.gz -C ../server/  进入指定目录  cd ../servers/tcl8.6.1/unix/  ./configure  make && make install |

##### 在线安装tcl（推荐）

node1.itcast.cn执行以下命令在线安装tcl

|  |
| --- |
| yum -y install tcl |

#### 编译redis

node1.itcast.cn执行以下命令进行编译：

|  |
| --- |
| cd /export/server/redis-3.2.8/  #或者使用命令 make 进行编译  make MALLOC=libc  make test && make install PREFIX=/export/server/redis-3.2.8 |

##### 修改redis配置文件

node1.itcast.cn执行以下命令修改redis配置文件

|  |
| --- |
| cd /export/server/redis-3.2.8/  mkdir -p /export/server/redis-3.2.8/log  mkdir -p /export/server/redis-3.2.8/data  vim redis.conf  # 修改第61行  bind node1.itcast.cn  # 修改第128行  daemonize yes  # 修改第163行  logfile "/export/server/redis-3.2.8/log/redis.log"  # 修改第247行  dir /export/server/redis-3.2.8/data |

##### 启动redis

node1.itcast.cn执行以下命令启动redis

|  |
| --- |
| cd /export/server/redis-3.2.8/  bin/redis-server redis.conf |

##### 关闭redis

|  |
| --- |
| bin/redis-cli -h node1.itcast.cn shutdown |

##### 连接redis客户端

node1.itcast.cn执行以下命令连接redis客户端

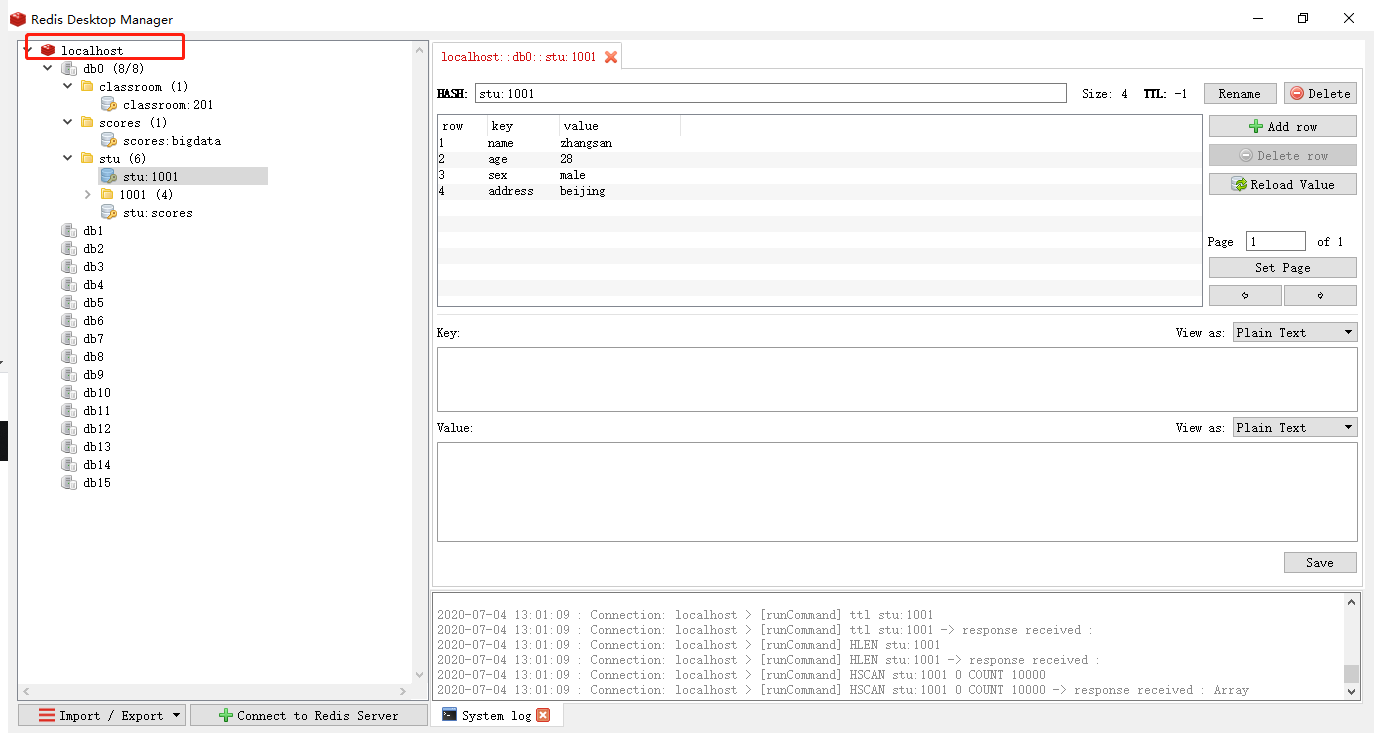
|  |
| --- |
| cd /export/server/redis-3.2.8/  bin/redis-cli -h node1.itcast.cn |

### Redis Desktop Manager

一款基于Qt5的跨平台Redis桌面管理软件，支持：Windows 7+、Mac OS X 10.10+、 Ubuntu 14+，特点： C++ 编写，响应迅速，性能好。

|  |
| --- |
| 下载地址：[http://docs.redisdesktop.com/en/latest/install/#windows](http://docs.redisdesktop.com/en/latest/install/" \l "windows) |

安装客户端，连接本地Redis服务：



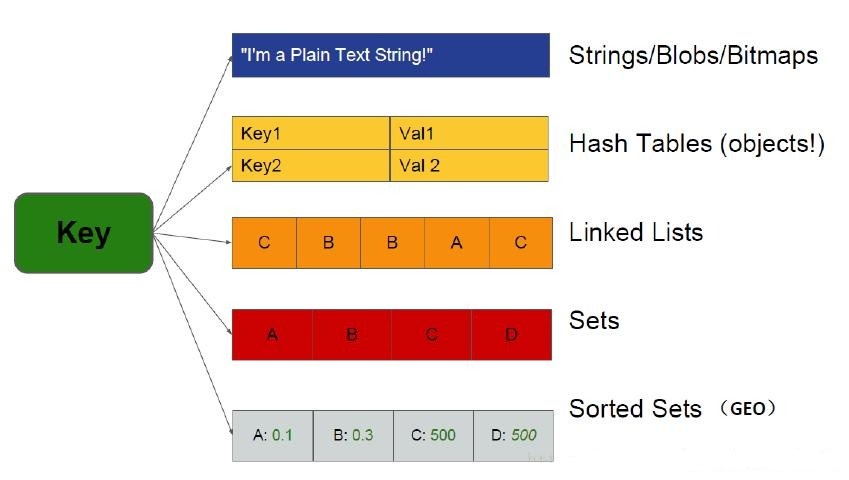
备注说明：Redis Desktoo Manager老版本免费，新版本收费。

## Redis的数据类型

redis当中一共支持五种数据类型，分别是：

* string字符串
* list列表
* set集合
* hash表
* zset有序集合

通过这五种不同的数据类型，可以实现各种不同的功能，也可以应用在各种不同的场景。



Redis当中各种数据类型结构如上图：

Redis当中各种数据类型的操作

<https://www.runoob.com/redis/redis-keys.html>

### 对字符串string的操作

下表列出了常用的 redis 字符串命令

|  |  |  |
| --- | --- | --- |
| 序号 | 命令及描述 | 示例 |
| 1 | **[SET key value](https://www.runoob.com/redis/strings-set.html)** 设置指定 key 的值 | 示例：SET hello world |
| 2 | **[GET key](https://www.runoob.com/redis/strings-get.html)** 获取指定 key 的值。 | 示例：GET hello |
| 3 | **[MGET key1 [key2..]](https://www.runoob.com/redis/strings-mget.html)** 获取所有(一个或多个)给定 key 的值。 | 示例：MGET hello world |
| 4 | **[SETEX key seconds value](https://www.runoob.com/redis/strings-setex.html)** 将值 value 关联到 key ，并将 key 的过期时间设为 seconds (以秒为单位)。 | 示例：SETEX hello 10 world3 |
| 5 | **[SETNX key value](https://www.runoob.com/redis/strings-setnx.html)** 只有在 key 不存在时设置 key 的值。 | 示例：SETNX itcast redisvalue |
| 6 | **[MSET key value [key value ...]](https://www.runoob.com/redis/strings-mset.html)** 同时设置一个或多个 key-value 对。 | 示例：MSET itcast2 itcastvalue2 itcast3 itcastvalue3 |
| 7 | **[INCR key](https://www.runoob.com/redis/strings-incr.html)** 将 key 中储存的数字值增一。 | 示例：  set itcast7 1  INCR itcast7  GET itcast7 |
| 8 | **[INCRBY key increment](https://www.runoob.com/redis/strings-incrby.html)** 将 key 所储存的值加上给定的增量值（increment） | 示例：INCRBY itcast7 2  get itcast7 |

### 对hash列表的操作

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。

Redis 中每个 hash 可以存储 232 - 1 键值对（40多亿）

下表列出了 redis hash 基本的相关命令：

|  |  |  |
| --- | --- | --- |
| 序号 | 命令及描述 | 示例 |
| 1 | **[HSET key field value](https://www.runoob.com/redis/hashes-hset.html)** 将哈希表 key 中的字段 field 的值设为 value 。 | 示例：HSET key1 field1 value1 |
| 3 | **[HMSET key field1 value1 [field2 value2 ]](https://www.runoob.com/redis/hashes-hmset.html)** 同时将多个 field-value (域-值)对设置到哈希表 key 中。 | 示例：HMSET key1 field3 value3 field4 value4 |
| 4 | **[HGET key field](https://www.runoob.com/redis/hashes-hget.html)** 获取存储在哈希表中指定字段的值。 | 示例：HGET key1 field4 |
| 5 | **[HKEYS key](https://www.runoob.com/redis/hashes-hkeys.html)** 获取所有哈希表中的字段 | 示例：HKEYS key1 |
| 6 | **[HINCRBY key field increment](https://www.runoob.com/redis/hashes-hincrby.html)** 为哈希表 key 中的指定字段的整数值加上增量 increment 。 | 示例：  HSET key2 field1 1  HINCRBY key2 field1 1  HGET key2 field1 |

### 对list列表的操作

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）

一个列表最多可以包含 232 - 1 个元素 (4294967295, 每个列表超过40亿个元素)。

下表列出了列表相关的基本命令：

|  |  |  |
| --- | --- | --- |
| **序号** | **命令及描述** | **示例** |
| 1 | **[LPUSH key value1 [value2]](https://www.runoob.com/redis/lists-lpush.html" \t "_blank)** 将一个或多个值插入到列表头部 | 示例：LPUSH list1 value1 value2 |
| 2 | **LRANGE key start stop**  查看list当中所有的数据 | 示例：LRANGE list1 0 -1 |
| 3 | **[LPOP key](https://www.runoob.com/redis/lists-lpop.html" \t "_blank)** 移出并获取列表的第一个元素 | 示例：LPOP list1 |

### 对set集合的操作

* Redis 的 Set 是 String 类型的无序集合。集合成员是唯一的，这就意味着集合中不能出现重复的数据
* Redis 中集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是 O(1)。
* 集合中最大的成员数为 232 - 1 (4294967295, 每个集合可存储40多亿个成员)。

下表列出了 Redis 集合基本命令：

|  |  |  |
| --- | --- | --- |
| **序号** | **命令及描述** | **示例** |
| 1 | **[SADD key member1 [member2]](https://www.runoob.com/redis/sets-sadd.html)** 向集合添加一个或多个成员 | 示例：SADD set1 setvalue1 setvalue2 |
| 2 | **[SMEMBERS key](https://www.runoob.com/redis/sets-smembers.html)** 返回集合中的所有成员 | 示例：SMEMBERS set1 |
| 3 | **[SCARD key](https://www.runoob.com/redis/sets-scard.html)** 获取集合的成员数 | 示例：SCARD set1 |

### 对key的操作

下表给出了与 Redis 键相关的基本命令：

|  |  |  |
| --- | --- | --- |
| 序号 | 命令及描述 | 示例 |
| 1 | **[DEL key](https://www.runoob.com/redis/keys-del.html)** 该命令用于在 key 存在时删除 key。 | 示例：del itcast5 |
| 3 | **[EXISTS key](https://www.runoob.com/redis/keys-exists.html)** 检查给定 key 是否存在。 | 示例：exists itcast |
| 4 | **[EXPIRE key](https://www.runoob.com/redis/keys-expire.html) seconds** 为给定 key 设置过期时间，以秒计。 | 示例：expire itcast 5 |

### 对ZSet的操作

* Redis有序集合和集合一样也是string类型元素的集合,且不允许重复的成员
* 它用来保存需要排序的数据，例如排行榜，一个班的语文成绩，一个公司的员工工资，一个论坛的帖子等。
* 有序集合中，每个元素都带有score（权重），以此来对元素进行排序
* 它有三个元素：key、member和score。以语文成绩为例，key是考试名称（期中考试、期末考试等），member是学生名字，score是成绩。

|  |  |  |
| --- | --- | --- |
| **#** | **命令及描述** | **示例** |
| 1 | **ZADD key score1 member1 [score2 member2]** 向有序集合添加一个或多个成员，或者更新已存在成员的分数 | 向ZSet中添加页面的PV值  ZADD pv\_zset 120 page1.html 100 page2.html 140 page3.html |
| 2 | ZCARD key  获取有序集合的成员数 | 获取所有的统计PV页面数量  ZCARD pv\_zset |

## Redis Java API操作

Redis不仅可以通过命令行进行操作，也可以通过JavaAPI操作，通过使用Java API来对Redis数据库中的各种数据类型操作。

### 创建maven工程并导入依赖

#### 创建Maven工程

|  |  |
| --- | --- |
| groupId | cn.itcast |
| artifactId | redis\_op |

#### 导入POM依赖

|  |
| --- |
| <dependencies>  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>2.9.0</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.testng</groupId>  <artifactId>testng</artifactId>  <version>6.14.3</version>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.0</version>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>UTF-8</encoding>  <!-- <verbal>true</verbal>-->  </configuration>  </plugin>  </plugins>  </build> |

### 创建包结构和类

1. 在test目录创建 cn.itcast.redis.api\_test 包结构
2. 创建RedisTest类

### 连接以及关闭redis客户端

因为后续测试都需要用到Redis连接，所以，我们先创建一个JedisPool用于获取Redis连接。此处，我们基于TestNG来测试各类的API。使用@BeforeTest在执行测试用例前，创建Redis连接池。使用@AfterTest在执行测试用例后，关闭连接池。

实现步骤：

1. 创建JedisPoolConfig配置对象，指定最大空闲连接为10个、最大等待时间为3000毫秒、最大连接数为50、最小空闲连接5个
2. 创建JedisPool
3. 使用@Test注解，编写测试用例，查看Redis中所有的key
   1. 从Redis连接池获取Redis连接
   2. 调用keys方法获取所有的key
   3. 遍历打印所有key

|  |
| --- |
| **import** java.util.Set;  **public class** RedisTest {  **private** JedisPool **jedisPool**;  **private** JedisPoolConfig **config**;   @BeforeTest  **public void** redisConnectionPool(){  **config** = **new** JedisPoolConfig();  **config**.setMaxIdle(10);  **config**.setMaxWaitMillis(3000);  **config**.setMaxTotal(50);  **config**.setMinIdle(5);  **jedisPool** = **new** JedisPool(**config**, **"node1.itcast.cn"**, 6379);  }   @Test  **public void** testConnect() {  Jedis jedis = **jedisPool**.getResource();  Set<String> keySet = jedis.keys(**"\*"**);  **for** (String s : keySet) {  System.***out***.print(s + **" "**);  }  }   @AfterTest  **public void** closePool(){  **jedisPool**.close();  } } |

### 操作string类型数据

1. 添加一个string类型数据，key为pv，用于保存pv的值，初始值为0
2. 查询该key对应的数据
3. 修改pv为1000
4. 实现整形数据原子自增操作 +1
5. 实现整形该数据原子自增操作 +1000

|  |
| --- |
| @Test **public void** stringOpTest() {  Jedis connection = **jedisPool**.getResource();  // 1. 添加一个string类型数据，key为pv，初始值为0  connection.set(**"pv"**, **"0"**);   // 2. 查询该key对应的数据  System.***out***.println(**"原始pv为:"** + connection.get(**"pv"**));   // 3. 修改pv为1000  connection.set(**"pv"**, **"1000"**);  System.***out***.println(**"修改pv为:"** + connection.get(**"pv"**));   // 4. 实现整形数据原子自增操作 +1  connection.incr(**"pv"**);  System.***out***.println(**"pv自增1："** + connection.get(**"pv"**));   // 5. 实现整形该数据原子自增操作 +1000  connection.incrBy(**"pv"**, 1000);  System.***out***.println(**"pv自增1000："** + connection.get(**"pv"**));  } |

### 操作hash列表类型数据

1. 往Hash结构中添加以下商品库存
   1. iphone11 => 10000
   2. macbookpro => 9000
2. 获取Hash中所有的商品
3. 新增3000个macbookpro库存
4. 删除整个Hash的数据

参考代码：

|  |
| --- |
| @Test **public void** hashOpTest() {  Jedis connection = **jedisPool**.getResource();  // 1. 往Hash结构中添加以下商品库存  // a) iphone11 => 10000  // b) macbookpro => 9000  connection.hset(**"goodsStore"**, **"iphone11"**, **"10000"**);  connection.hset(**"goodsStore"**, **"macbookpro"**, **"9000"**);   // 2. 获取Hash中所有的商品  Map<String, String> keyValues = connection.hgetAll(**"goodsStore"**);  **for** (String s : keyValues.keySet()) {  System.***out***.println(s + **" => "** + keyValues.get(s));  }   // 3. 修改Hash中macbookpro数量为12000  // 方式一：  // connection.hset("goodsStore", "macbookpro", "12000");  // 方式二：  connection.hincrBy(**"goodsStore"**, **"macbookpro"**, 3000);  System.***out***.println(**"新增3000个库存后：macbookpro => "** + connection.hget(**"goodsStore"**, **"macbookpro"**));   // 4. 删除整个Hash的数据  connection.del(**"goodsStore"**); } |

### 操作list类型数据

1. 向list的左边插入以下三个手机号码：18511310001、18912301231、18123123312
2. 从右边移除一个手机号码
3. 获取list所有的值

|  |
| --- |
| @Test **public void** listOpTest() {  Jedis connection = **jedisPool**.getResource();  // 1. 向list的左边插入以下三个手机号码：18511310001、18912301231、18123123312  connection.lpush(**"telephone"**, **"18511310001"**, **"18912301231"**, **"18123123312"**);   // 2. 从右边移除一个手机号码  connection.rpop(**"telephone"**);   // 3. 获取list所有的值  List<String> telList = connection.lrange(**"telephone"**, 0, -1);  **for** (String tel : telList) {  System.***out***.print(tel + **" "**);  } } |

### 操作set类型的数据

使用set来保存uv值，为了方便计算，将用户名保存到uv中。

1. 往一个set中添加页面 page1 的uv，用户user1访问一次该页面
2. user2访问一次该页面
3. user1再次访问一次该页面
4. 最后获取 page1的uv值

|  |
| --- |
| @Test **public void** setOpTest() {  Jedis connection = **jedisPool**.getResource();   // 1. 往一个set中添加页面 page1 的uv，用户user1访问一次该页面  connection.sadd(**"page1"**, **"user1"**);   // 2. user2访问一次该页面  connection.sadd(**"page1"**, **"user2"**);   // 3. user1再次访问一次该页面  connection.sadd(**"page1"**, **"user1"**);   // 4. 最后获取 page1的uv值  Long uv = connection.scard(**"page1"**);  System.***out***.println(**"page1页面的UV为："** + uv); } |

## Redis的持久化

由于redis是一个内存数据库，所有的数据都是保存在内存当中的，内存当中的数据极易丢失，所以redis的数据持久化就显得尤为重要，在redis当中，提供了两种数据持久化的方式，分别为RDB以及AOF，且Redis默认开启的数据持久化方式为RDB方式。

### RDB持久化方案

#### 介绍

Redis会定期保存数据快照至一个rbd文件中，并在启动时自动加载rdb文件，恢复之前保存的数据。可以在配置文件中配置Redis进行快照保存的时机：

|  |
| --- |
| save [seconds] [changes] |

意为在seconds秒内如果发生了changes次数据修改，则进行一次RDB快照保存，例如

|  |
| --- |
| save 60 100 |

会让Redis每60秒检查一次数据变更情况，如果发生了100次或以上的数据变更，则进行RDB快照保存。可以配置多条save指令，让Redis执行多级的快照保存策略。Redis默认开启RDB快照。也可以通过SAVE或者BGSAVE命令手动触发RDB快照保存。SAVE 和 BGSAVE 两个命令都会调用 rdbSave 函数，但它们调用的方式各有不同：

* SAVE 直接调用 rdbSave ，阻塞 Redis 主进程，直到保存完成为止。在主进程阻塞期间，服务器不能处理客户端的任何请求。
* BGSAVE 则 fork 出一个子进程，子进程负责调用 rdbSave ，并在保存完成之后向主进程发送信号，通知保存已完成。 Redis 服务器在BGSAVE 执行期间仍然可以继续处理客户端的请求。

#### RDB方案优点

1. 对性能影响最小。如前文所述，Redis在保存RDB快照时会fork出子进程进行，几乎不影响Redis处理客户端请求的效率。
2. 每次快照会生成一个完整的数据快照文件，所以可以辅以其他手段保存多个时间点的快照（例如把每天0点的快照备份至其他存储媒介中），作为非常可靠的灾难恢复手段。
3. 使用RDB文件进行数据恢复比使用AOF要快很多

#### RDB方案缺点

1. 快照是定期生成的，所以在Redis crash时或多或少会丢失一部分数据
2. 如果数据集非常大且CPU不够强（比如单核CPU），Redis在fork子进程时可能会消耗相对较长的时间，影响Redis对外提供服务的能力

#### RDB配置

1. 修改redis的配置文件

|  |
| --- |
| cd /export/server/redis-3.2.8/  vim redis.conf  # 第202行  save 900 1  save 300 10  save 60 10000  save 5 1 |

这三个选项是redis的配置文件默认自带的存储机制。表示每隔多少秒，有多少个key发生变化就生成一份dump.rdb文件，作为redis的快照文件

例如：save 60 10000 表示在60秒内，有10000个key发生变化，就会生成一份redis的快照

1. 重新启动redis服务

每次生成新的dump.rdb都会覆盖掉之前的老的快照

|  |
| --- |
| ps -ef | grep redis  bin/redis-cli -h node1.itcast.cn shutdown  bin/redis-server redis.conf |

### AOF持久化方案

#### 介绍

采用AOF持久方式时，Redis会把每一个写请求都记录在一个日志文件里。在Redis重启时，会把AOF文件中记录的所有写操作顺序执行一遍，确保数据恢复到最新。

#### 开启AOF

AOF默认是关闭的，如要开启，进行如下配置：

|  |
| --- |
| # 第594行  appendonly yes |

#### 配置AOF

AOF提供了三种fsync配置：always/everysec/no，通过配置项[appendfsync]指定：

1. **appendfsync no**：不进行fsync，将flush文件的时机交给OS决定，速度最快
2. **appendfsync always**：每写入一条日志就进行一次fsync操作，数据安全性最高，但速度最慢
3. **appendfsync everysec**：折中的做法，交由后台线程每秒fsync一次

#### AOF rewrite

随着AOF不断地记录写操作日志，因为所有的写操作都会记录，所以必定会出现一些无用的日志。大量无用的日志会让AOF文件过大，也会让数据恢复的时间过长。不过Redis提供了AOF rewrite功能，可以重写AOF文件，只保留能够把数据恢复到最新状态的最小写操作集。

AOF rewrite可以通过BGREWRITEAOF命令触发，也可以配置Redis定期自动进行：

|  |
| --- |
| auto-aof-rewrite-percentage 100  auto-aof-rewrite-min-size 64mb |

* Redis在每次AOF rewrite时，会记录完成rewrite后的AOF日志大小，当AOF日志大小在该基础上增长了100%后，自动进行AOF rewrite
* auto-aof-rewrite-min-size最开始的AOF文件必须要触发这个文件才触发，后面的每次重写就不会根据这个变量了。该变量仅初始化启动Redis有效。

#### AOF优点

1. 最安全，在启用appendfsync为always时，任何已写入的数据都不会丢失，使用在启用appendfsync everysec也至多只会丢失1秒的数据
2. AOF文件在发生断电等问题时也不会损坏，即使出现了某条日志只写入了一半的情况，也可以使用redis-check-aof工具轻松修复
3. AOF文件易读，可修改，在进行某些错误的数据清除操作后，只要AOF文件没有rewrite，就可以把AOF文件备份出来，把错误的命令删除，然后恢复数据。

#### AOF的缺点

1. AOF文件通常比RDB文件更大
2. 性能消耗比RDB高
3. 数据恢复速度比RDB慢

Redis的数据持久化工作本身就会带来延迟，需要根据数据的安全级别和性能要求制定合理的持久化策略：

* AOF + fsync always的设置虽然能够绝对确保数据安全，但每个操作都会触发一次fsync，会对Redis的性能有比较明显的影响
* AOF + fsync every second是比较好的折中方案，每秒fsync一次
* AOF + fsync never会提供AOF持久化方案下的最优性能

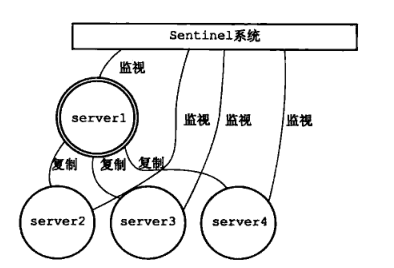
使用RDB持久化通常会提供比使用AOF更高的性能，但需要注意RDB的策略配置

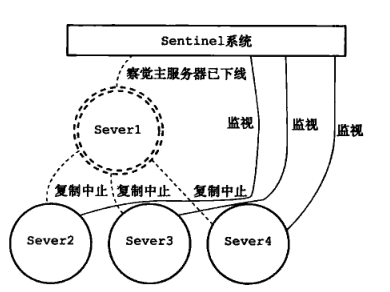
## Redis中的Sentinel架构

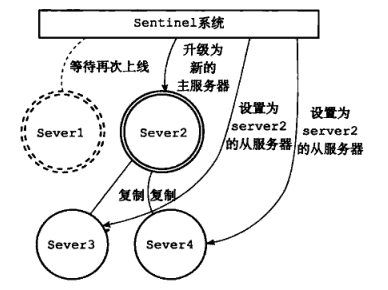
### Sentinel介绍

Sentinel（哨兵）是Redis的高可用性解决方案：由一个或多个Sentinel实例 组成的Sentinel系统可以监视任意多个主服务器，以及这些主服务器属下的所有从服务器，并在被监视的主服务器进入下线状态时，自动将下线主服务器属下的某个从服务器升级为新的主服务器。

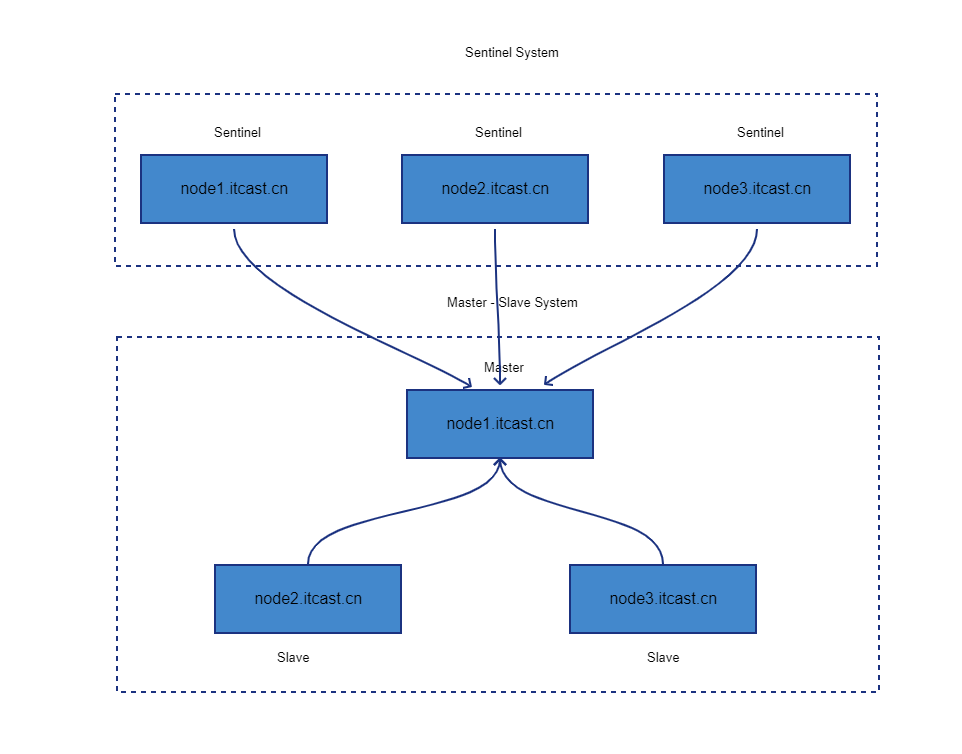
例如：



在Server1 掉线后：

升级Server2 为新的主服务器：

### 配置哨兵



#### 三台机器修改哨兵配置文件

三台机器执行以下命令修改redis的哨兵配置文件

|  |
| --- |
| cd /export/server/redis-3.2.8  vim sentinel.conf |

**配置监听的主服务器**

1. 修改node1.itcast.cn的sentinel.conf文件

|  |
| --- |
| #修改第15行， bind配置，每台机器修改为自己对应的主机名  bind node1.itcast.cn  # 在下方添加配置，让sentinel服务后台运行  daemonize yes  #修改第71行，三台机器监控的主节点，现在主节点是node1.itcast.cn服务器  sentinel monitor mymaster node1.itcast.cn 6379 2 |

参数说明

* sentinel monitor代表监控
* mymaster代表服务器的名称，可以自定义
* node1.itcast.cn代表监控的主服务器，6379代表端口
* 2代表只有两个或两个以上的哨兵认为主服务器不可用的时候，才会进行failover操作。

如果Redis是有密码的，需要指定密码

|  |
| --- |
| # sentinel author-pass定义服务的密码，mymaster是服务名称，123456是Redis服务器密码  # sentinel auth-pass <master-name> <password> |

1. 分发到node2.itcast.cn和node3.itcast.cn

|  |
| --- |
| scp sentinel.conf node2.itcast.cn:$PWD  scp sentinel.conf node3.itcast.cn:$PWD |

1. 分别修改配置中bind的服务器主机名

node2.itcast.cn

|  |
| --- |
| cd /export/server/redis-3.2.8  vim sentinel.conf  # 修改第18行  bind node2.itcast.cn |

node3.itcast.cn

|  |
| --- |
| cd /export/server/redis-3.2.8  vim sentinel.conf  # 修改第18行  bind node3.itcast.cn |

#### 三台机器启动哨兵服务

|  |
| --- |
| cd /export/server/redis-3.2.8  bin/redis-sentinel sentinel.conf |

三台服务器的进程信息：

|  |
| --- |
| **node1.itcast.cn**  [root@node1 redis-3.2.8]# ps aux | grep redis  root 18911 0.0 0.0 136920 2456 ? Ssl 08:58 0:04 bin/redis-server node1.itcast.cn:6379  root 19112 0.0 0.0 149232 5152 pts/1 S+ 09:16 0:00 vim redis.conf  root 20544 0.1 0.0 135728 2328 ? Ssl 10:48 0:00 **bin/redis-sentinel node1.itcast.cn:26379 [sentinel**]  root 20548 0.0 0.0 112712 960 pts/3 S+ 10:48 0:00 grep --color=auto redis  **node2.itcast.cn**  [root@node2 redis-3.2.8]# ps aux | grep redis  root 26260 0.0 0.1 139200 4456 ? Ssl 10:34 0:00 bin/redis-server node2.itcast.cn:6379  root 26421 0.1 0.0 139204 2440 ? Ssl 10:48 0:00 **bin/redis-sentinel node2.itcast.cn:26379 [sentinel]**  root 26438 0.0 0.0 112812 972 pts/1 S+ 10:49 0:00 grep --color=auto redis  **node3.itcast.cn**  [root@node3 redis-3.2.8]# ps aux | grep redis  root 22325 0.0 0.0 135992 2376 ? Ssl 10:34 0:00 bin/redis-server node3.itcast.cn:6379  root 22463 0.1 0.0 135836 2384 ? Ssl 10:48 0:00 **bin/redis-sentinel node3.itcast.cn:26379 [sentinel]**  root 22475 0.0 0.0 112812 972 pts/1 S+ 10:49 0:00 grep --color=auto redis |

#### node1服务器杀死redis服务进程

|  |
| --- |
| 查看Sentinel master的状态  bin/redis-cli -h node2.itcast.cn -p 26379  使用ping命令检查哨兵是否工作，如果正常会返回PONG  node2.itcast.cn:26379> ping  PONG  node2.itcast.cn:26379> info  ... ... ...  # Sentinel  sentinel\_masters:1  sentinel\_tilt:0  sentinel\_running\_scripts:0  sentinel\_scripts\_queue\_length:0  sentinel\_simulate\_failure\_flags:0  **master0:name=mymaster,status=ok,address=192.168.88.100:6379,slaves=2,sentinels=3** |

使用kill -9命令杀死redis服务进程，模拟redis故障宕机情况

过一段时间之后，就会在node2.itcast.cn与node3.itcast.cn服务器选择一台服务器来切换为主节点

|  |
| --- |
| node2.itcast.cn:26379> **info**  ... ... ...  # Sentinel  sentinel\_masters:1  sentinel\_tilt:0  sentinel\_running\_scripts:0  sentinel\_scripts\_queue\_length:0  sentinel\_simulate\_failure\_flags:0  master0:name=mymaster,status=ok,address=**192.168.88.102:6379**,slaves=2,sentinels=3 |

### Redis的sentinel模式代码开发连接

通过哨兵连接，要指定哨兵的地址，并使用JedisSentinelPool来创建连接池。

实现步骤：

1. 在 cn.itcast.redis.api\_test 包下创建一个新的类 ReidsSentinelTest
2. 构建JedisPoolConfig配置对象
3. 创建一个HashSet，用来保存哨兵节点配置信息（记得一定要写端口号）
4. 构建JedisSentinelPool连接池
5. 使用sentinelPool连接池获取连接

|  |
| --- |
| **public class** ReidsSentinelTest {  **private** JedisPoolConfig **jedisPoolConfig**;  **private** JedisSentinelPool **jedisSentinelPool**;   @BeforeTest  **public void** beforeTest() {  // 1. 构建JedisPoolConfig配置对象  **jedisPoolConfig** = **new** JedisPoolConfig();  **jedisPoolConfig**.setMaxTotal(50);  **jedisPoolConfig**.setMaxIdle(10);  **jedisPoolConfig**.setMinIdle(5);  **jedisPoolConfig**.setMaxWaitMillis(10000);   // 2. 创建一个HashSet，用来保存哨兵节点配置信息  HashSet<String> sentinelNodeSet = **new** HashSet<>(Arrays.*asList*(**"node1.itcast.cn:26379"**, **"node2.itcast.cn:26379"**, **"node3.itcast.cn:26379"**));   // 3. 构建JedisSentinelPool连接池  **jedisSentinelPool** = **new** JedisSentinelPool(**"mymaster"**, sentinelNodeSet, **jedisPoolConfig**);   }   @Test  **public void** keysOpTest() {  // 使用sentinelPool连接池获取连接  Jedis connection = **jedisSentinelPool**.getResource();  Set<String> keySet = connection.keys(**"\*"**);   **for** (String key : keySet) {  System.***out***.println(key);  }  }   @AfterTest  **public void** afterTest() {  **jedisSentinelPool**.close();  } } |

## Redis 集群

Redis最开始使用**主从模式做集群**，若master宕机需要手动配置slave转为master；后来为了**高可用提出来哨兵模式**，该模式下有一个哨兵监视master和slave，若master宕机可自动将slave转为master，但它也有一个问题，就是不能动态扩充；所以在Redis 3.x提出**cluster集群**模式。

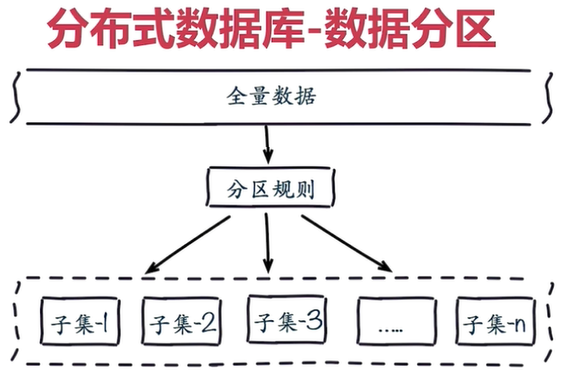
### 引言

​Redis Cluster是Redis官方提供的Redis集群功能，为什么要实现Redis Cluster？

1. 主从复制不能实现**高可用**
2. 随着公司发展，用户数量增多，并发越来越多，业务需要更高的QPS，而主从复制中单机的**QPS**可能无法满足业务需求；
3. 数据量的考虑，现有服务器内存不能满足业务数据的需要时，单纯向服务器添加内存不能达到要求，此时需要考虑**分布式**需求，把数据分布到不同服务器上；
4. 网络流量需求，业务的流量已经超过服务器的网卡的上限值，可考虑使用分布式来进行**分流；**
5. 离线计算，需要中间环节**缓冲**等其他需求；

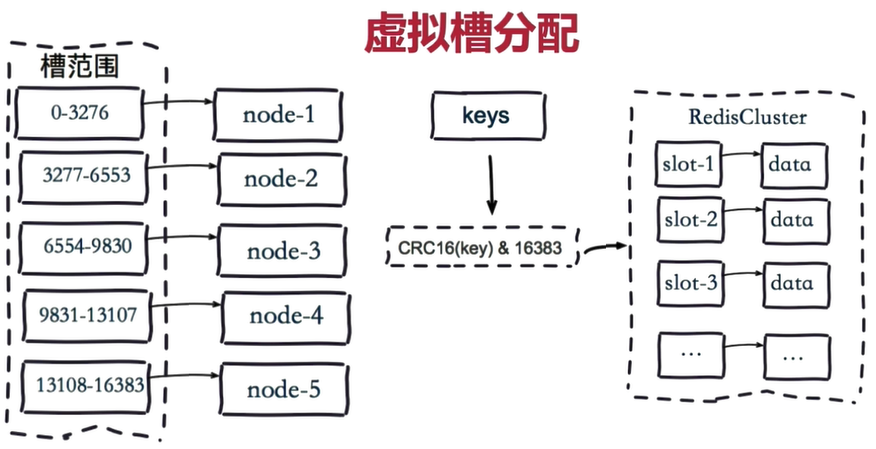
在存储引擎框架（MySQL、HDFS、HBase、Redis、Elasticsearch等）中，只要数据量很大时，单机无法承受压力，最好的方式就是：**数据分布**进行存储管理。

**对Redis 内存数据库来说：全量数据，单机Redis节点无法满足要求，按照分区规则把数据分到若干个子集当中。**



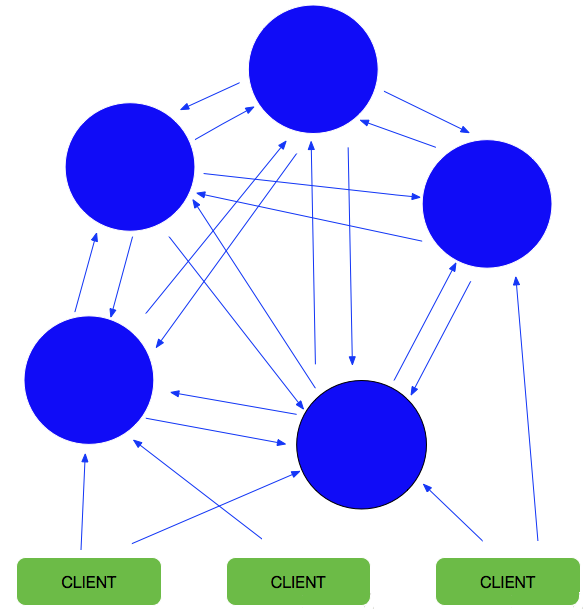
常用数据分布方式：

1. **顺序分布**：关系型数据库设计
   * 比如：1到100个数字，要保存在3个节点上，按照顺序分区，把数据平均分配三个节点上，此时1号到33号数据保存到节点1上，34号到66号数据保存到节点2上，67号到100号数据保存到节点3上
2. **哈希分布**：
   * 例如1到100个数字，对每个数字进行哈希hash运算，然后对每个数的哈希结果除以节点数进行取余，余数为1则保存在第1个节点上，余数为2则保存在第2个节点上，余数为0则保存在第3个节点，这样可以保证数据被打散，同时保证数据分布的比较均匀。
   * 哈希分布方式分为三个分区方式：
     + **节点取余分区**：比如有100个数据，对每个数据进行hash运算之后，与节点数进行取余运算，根据余数不同保存在不同的节点上
     + **一致性哈希分区**：将所有的数据当做一个token环，token环中的数据范围是0到2的32次方。然后为每一个数据节点分配一个token范围值，这个节点就负责保存这个范围内的数据。
     + **虚拟槽分区**：虚拟槽分区是Redis Cluster采用的分区方式，预设虚拟槽，每个槽就相当于一个数字，有一定范围。每个槽映射一个数据子集，一般比节点数大。Redis Cluster中预设虚拟槽的范围为0到16383。



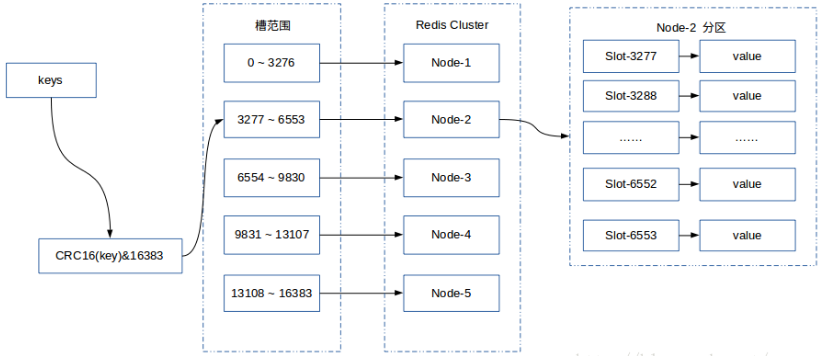
### Redis Cluster 设计

Redis Cluster是分布式架构，有多个节点，每个节点都负责进行数据读写操作，每个节点之间会进行通信。Redis Cluster采用**无中心结构**，每个节点保存数据和整个集群状态，每个节点都和其他所有节点连接。



结构特点：

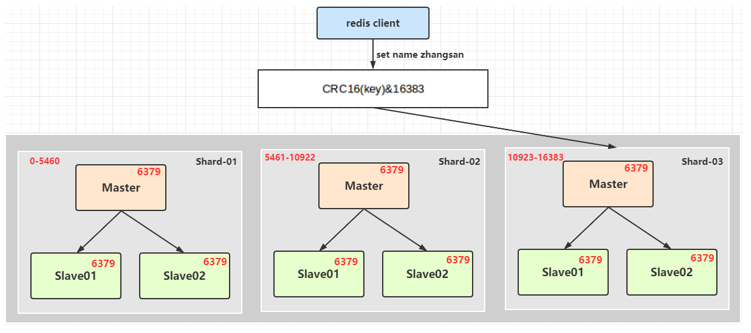
* 所有的redis节点彼此互联(PING-PONG机制)，内部使用**二进制协议**优化传输速度和带宽；
* 节点的fail是通过集群中**超过半数**的节点检测失效时才生效；
* 客户端与redis节点直连，不需要中间proxy层，客户端不需要连接集群所有节点，连接集群中任何一个可用节点即可；
* redis-cluster 把所有的物理节点映射到**[0-16383]slot**上（不一定是平均分配）,cluster 负责维护**node<->slot<->value**；
* Redis集群预分好**16384**个桶（Slot），当需要在 Redis 集群中放置一个 key-value 时，根据 CRC16(key) & 16384的值，决定将一个key放到哪个桶中；



Redis 集群的优势：

* **缓存永不宕机**：启动集群，永远让集群的一部分起作用。主节点失效了子节点能迅速改变角色成为主节点，整个集群的部分节点失败或者不可达的情况下能够继续处理命令；
* **迅速恢复数据**：持久化数据，能在宕机后迅速解决数据丢失的问题；
* Redis可以使用所有机器的内存，变相扩展性能；
* 使Redis的计算能力通过简单地增加服务器得到成倍提升，Redis的网络带宽也会随着计算机和网卡的增加而成倍增长；
* Redis集群没有中心节点，不会因为某个节点成为整个集群的性能瓶颈;
* 异步处理数据，实现快速读写；

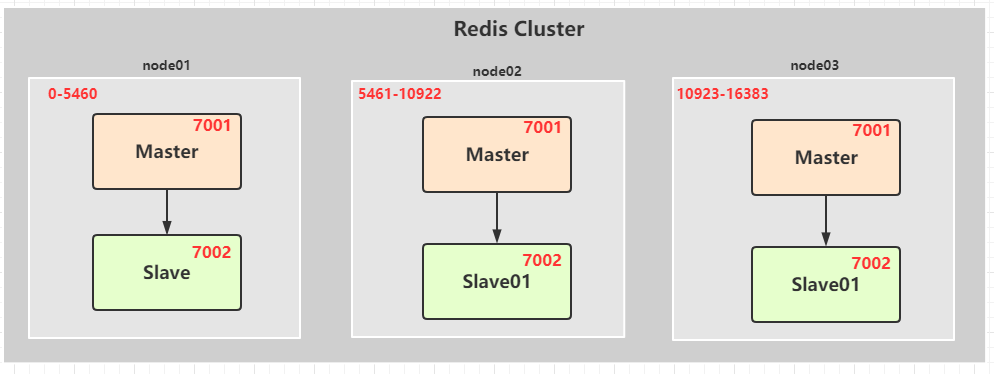
Redis 3.0以后，节点之间通过去中心化的方式提供了完整的**sharding(数据分片)**、**replication(复制机制、Cluster具备感知准备的能力)**、**failover解决方案**。



### Redis Cluster 搭建

Redis3.0及以上版本实现，集群中至少应该有**奇数个节点**，所以**至少有三个节点**，官方推荐**三主三从**的配置方式。**Redis 3.x和Redis4.x 搭建集群是需要手动安装ruby组件的，比较麻烦。**

2018年十月 Redis 发布了稳定版本的 5.0 版本，推出了各种新特性，其中一点是放弃 Ruby的集群方式，改为 使用 C语言编写的**redis-cli**的方式，是集群的构建方式复杂度大大降低。Redis cluster tutorial：<https://redis.io/topics/cluster-tutorial> 。



基于Redis-5.0.8版本，在三台机器上搭建6个节点的Redis集群：**三主三从架构**。

#### 环境准备

关闭以前Redis主从复制和哨兵模式监控的所有服务，备注：如果以前没有安装过Redis服务，不用执行此步骤操作。

|  |
| --- |
| # ============= node1.itcast.cn、node2.itcast.cn和node3.itcast.cn =============  # 关闭哨兵服务SentinelServer  **ps -ef|grep redis**  **kill -9 哨兵的进程ID**  # 关闭Redis服务  **redis-cli -h node1.itcast.cn -p 6379 SHUTDOWN**  **redis-cli -h node2.itcast.cn -p 6379 SHUTDOWN**  **redis-cli -h node3.itcast.cn -p 6379 SHUTDOWN** |

安装Redis编译环境：GCC和TCL。

|  |
| --- |
| yum -y install gcc-c++ tcl |

#### 上传和解压

将Redis-5.0.8软件安装包上传至 **/export/software** 目录，并解压与安装。

|  |
| --- |
| # node01, 上传安装包至/export/softwares  **cd /export/software**  **rz**  # 解压  **cd /export/software**  **chmod u+x redis-5.0.8.tar.gz**  **tar -zxvf redis-5.0.8.tar.gz -C /export/server/** |

#### 编译安装

编译Redis 源码，并安装至【**/export/server/redis-5.0.8-bin**】目录。

|  |
| --- |
| # node01, 编译、安装、创建软连接  # 进入源码目录  **cd /export/server/redis-5.0.8**  # 编译  **make**  # 安装至指定目录  **make PREFIX=/export/server/redis-5.0.8-bin install**  # 创建安装目录软连接  **cd /export/server**  **ln -s redis-5.0.8-bin redis** |

配置环境变量（如果以前安装过Redis，配置过环境变量，就不用配置）。

|  |
| --- |
| # 配置环境变量  **vim /etc/profile**  # ======================== 添加如下内容 ========================  **# REDIS HOME**  **export REDIS\_HOME=/export/server/redis**  **export PATH=:$PATH:$REDIS\_HOME/bin**  # 执行生效  **source /etc/profile** |

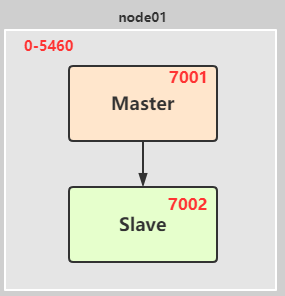
#### 拷贝配置文件

从Redis-5.0.8源码目录下拷贝配置文件：**redis.conf**至Redis 安装目录。

|  |
| --- |
| # ====================== node01 上操作 ======================  # 拷贝配置文件  **cd /export/server/redis-5.0.8**  **cp redis.conf /export/server/redis** |

#### 修改配置文件

每台机器上启动2个Redis服务，一个**主节点服务：7001**，一个**从节点服务：7002**，如下图所示：



在Redis安装目录下创建7001和7002目录，分别存储Redis服务配置文件、日志及数据文件。

|  |
| --- |
| # 创建目录：7001和7002  **cd /export/server/redis**  **mkdir -p 7001 7002** |

拷贝配置文件：**redis.conf**至7001目录，并重命名为**redis\_7001.conf**。

|  |
| --- |
| cd /export/server/redis  cp redis.conf 7001/redis\_7001.conf |

编辑配置文件：**redis\_7001.conf**，内容如下：

|  |
| --- |
| **cd /export/server/redis/7001**  **vim redis\_7001.conf**  ## =========================== 修改内容说明如下 ===========================  ## 69行，配置redis服务器接受链接的网卡  bind 0.0.0.0  ## 88行，关闭保护模式  protected-mode no  ## 92行，设置端口号  port 7001  ## 136行，redis后台运行  daemonize yes  ## 158行，Redis服务进程PID存储文件名称  pidfile /var/run/redis\_7001.pid  ## 171行，设置redis服务日志存储路径  logfile "/export/server/redis-5.0.8-bin/7001/log/redis.log"  ## 263行，设置redis持久化数据存储目录  dir /export/server/redis-5.0.8-bin/7001/data/  ## 699行，启动AOF方式持久化  appendonly yes  ## 832行，启动Redis Cluster  cluster-enabled yes  ## 840行，Redis服务配置保存文件名称  cluster-config-file nodes-7001.conf  ## 847行，超时时间  cluster-node-timeout 15000 |

创建日志目录和数据目录：

|  |
| --- |
| mkdir -p /export/server/redis/7001/log  mkdir -p /export/server/redis/7001/data |

配置7002端口号启动Redis服务，操作命令如下：

|  |
| --- |
| ## 拷贝配置文件  **cd /export/server/redis**  **cp 7001/redis\_7001.conf 7002/redis\_7002.conf**  ## 修改配置文件：redis\_7002.conf  cd /export/server/redis/7002  vim redis\_7002.conf  # 进入vim编辑之后，执行以下代码将7001全部替换成7002  :%s/7001/7002/g # 表示:%s/old/new/g g表示全部替换  # 创建目录  **mkdir -p /export/server/redis/7002/log**  **mkdir -p /export/server/redis/7002/data** |

#### 发送安装包

将node1.itcast.cn上配置好的Redis安装包，发送至node2.itcast.cn和node3.itcast.cn，每台机器运行2个Redis服务，端口号分别为7001和7002，具体命令如下：

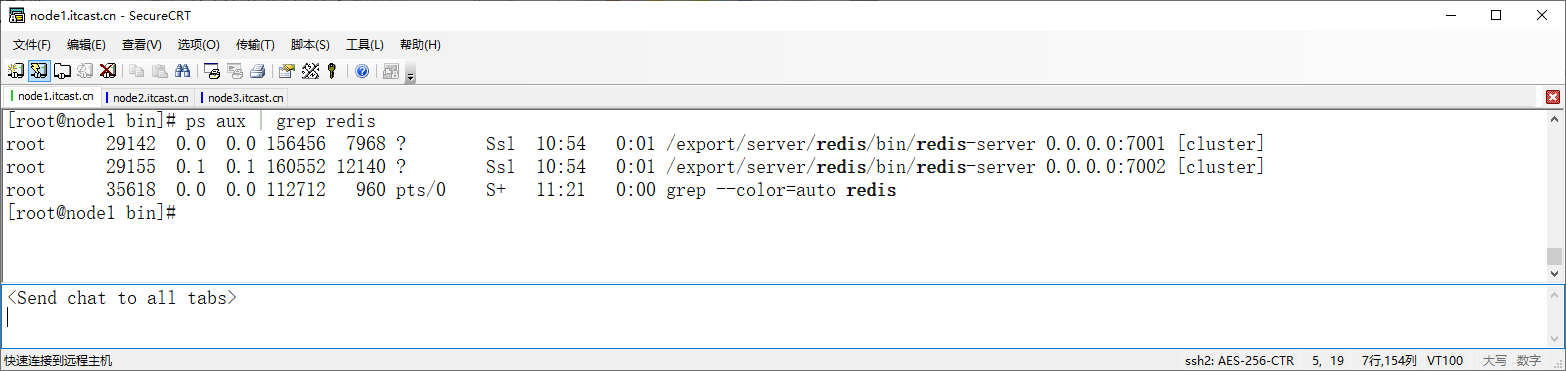
|  |
| --- |
| # 发送安装包  **cd /export/server**  **scp -r redis-5.0.8-bin root@node2.itcast.cn:$PWD**  **scp -r redis-5.0.8-bin root@node3.itcast.cn:$PWD**  # 在node2和node3创建软连接  **cd /export/server**  **ln -s redis-5.0.8-bin redis**  # 配置环境变量  **vim /etc/profile**  **# ======================== 添加如下内容 ========================**  **# REDIS HOME**  **export REDIS\_HOME=/export/server/redis**  **export PATH=:$PATH:$REDIS\_HOME/bin**  **# 执行生效**  **source /etc/profile** |

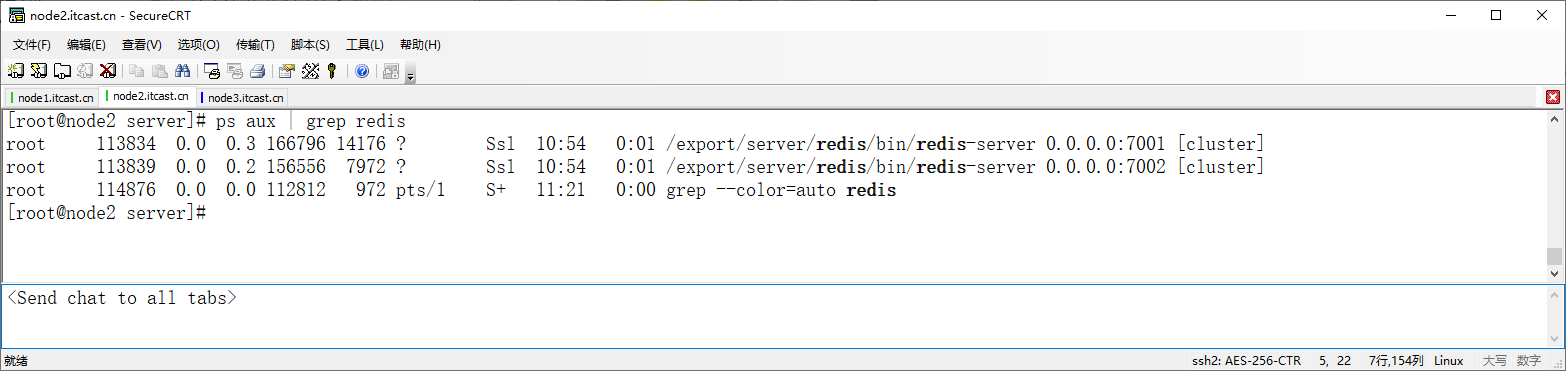
### 启动Redis服务

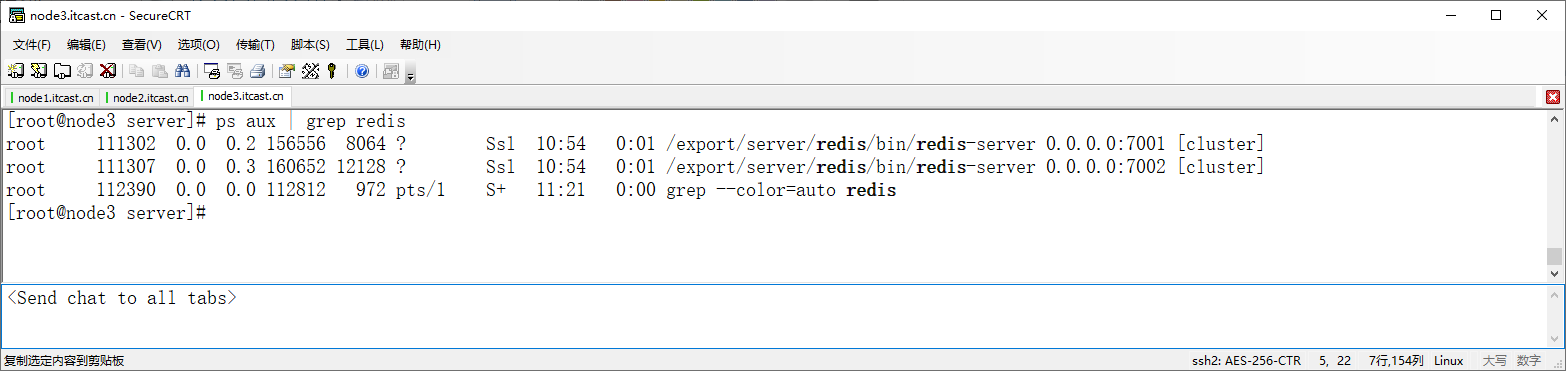
在三台机器node01、node02和node03，分别启动6个Redis服务，命令如下：

|  |
| --- |
| # 启动7001端口Redis服务  **/export/server/redis/bin/redis-server /export/server/redis/7001/redis\_7001.conf**  # 启动7002端口Redis服务  **/export/server/redis/bin/redis-server /export/server/redis/7002/redis\_7002.conf** |

Redis服务启动完成以后，查看如下：







#### 启动集群

Redis5.x版本之后，通过redis-cli客户端命令来进行创建集群，注意：**Redis对主机名解析不友好，使用IP地址**。

|  |
| --- |
| # 任意选择一台机器执行如下命令，创建集群  **/export/server/redis/bin/redis-cli --cluster create 192.168.88.100:7001 192.168.88.101:7002 192.168.88.102:7001 192.168.88.100:7002 192.168.88.101:7001 192.168.88.102:7002 --cluster-replicas 1** |

启动集群日志信息如下：

|  |
| --- |
| >>> Performing hash slots allocation on 6 nodes...  ## ====== 进行Slot槽范文划分 ======  Master[0] -> Slots 0 - 5460  Master[1] -> Slots 5461 - 10922  Master[2] -> Slots 10923 – 16383  ## ====== 主从分配，一主一从 ======  Adding replica 192.168.88.101:7001 to 192.168.88.100:7001  Adding replica 192.168.88.102:7002 to 192.168.88.101:7002  Adding replica 192.168.88.100:7002 to 192.168.88.102:7001  ## ====== 主从节点配对信息 ======  M: 97e1d381d59561e075ac813e2df7fed00114687e 192.168.88.100:7001  slots:[0-5460] (5461 slots) master  M: 6e0353e92377a71d691a853152673a8774d11dc2 192.168.88.101:7002  slots:[5461-10922] (5462 slots) master  M: d9bf2ac8eec5637ed7ec50061419ff6b951eef0b 192.168.88.102:7001  slots:[10923-16383] (5461 slots) master  S: b679ac2df0df7509ffd3a1d3b460cb9e13f9dbfa 192.168.88.100:7002  replicates d9bf2ac8eec5637ed7ec50061419ff6b951eef0b  S: b8a76df88aafb19ce38232d0b4c1daf12370f257 192.168.88.101:7001  replicates 97e1d381d59561e075ac813e2df7fed00114687e  S: c038b9e271f5b5dba47d6ef81c4f49548a9f3698 192.168.88.102:7002  replicates 6e0353e92377a71d691a853152673a8774d11dc2  ## ====== 是否同意上述划分，一般都是yes ======  Can I set the above configuration? (type 'yes' to accept): yes  ## ====== 节点配置更新，进入集群，主从节点，高可用 ======  >>> Nodes configuration updated  >>> Assign a different config epoch to each node  >>> Sending CLUSTER MEET messages to join the cluster  Waiting for the cluster to join  ...  >>> Performing Cluster Check (using node 192.168.88.100:7001)  M: 97e1d381d59561e075ac813e2df7fed00114687e 192.168.88.100:7001  slots:[0-5460] (5461 slots) master  1 additional replica(s)  S: c038b9e271f5b5dba47d6ef81c4f49548a9f3698 192.168.88.102:7002  slots: (0 slots) slave  replicates 6e0353e92377a71d691a853152673a8774d11dc2  S: b8a76df88aafb19ce38232d0b4c1daf12370f257 192.168.88.101:7001  slots: (0 slots) slave  replicates 97e1d381d59561e075ac813e2df7fed00114687e  S: b679ac2df0df7509ffd3a1d3b460cb9e13f9dbfa 192.168.88.100:7002  slots: (0 slots) slave  replicates d9bf2ac8eec5637ed7ec50061419ff6b951eef0b  M: 6e0353e92377a71d691a853152673a8774d11dc2 192.168.88.101:7002  slots:[5461-10922] (5462 slots) master  1 additional replica(s)  M: d9bf2ac8eec5637ed7ec50061419ff6b951eef0b 192.168.88.102:7001  slots:[10923-16383] (5461 slots) master  1 additional replica(s)  [OK] All nodes agree about slots configuration.  >>> Check for open slots...  >>> Check slots coverage...  [OK] All 16384 slots covered. |

#### 测试集群

在任意一台机器，使用**redis-cli**客户端命令连接Redis服务：

|  |
| --- |
| **redis-cli -c -p 7001** |

输入命令：**cluster nodes**（查看集群信息）和**info replication**（主从信息）：

|  |
| --- |
| 127.0.0.1:7001> **cluster nodes**  c038b9e271f5b5dba47d6ef81c4f49548a9f3698 192.168.88.102:7002@17002 slave 6e0353e92377a71d691a853152673a8774d11dc2 0 1598239340178 6 connected  b8a76df88aafb19ce38232d0b4c1daf12370f257 192.168.88.101:7001@17001 slave 97e1d381d59561e075ac813e2df7fed00114687e 0 1598239338161 5 connected  b679ac2df0df7509ffd3a1d3b460cb9e13f9dbfa 192.168.88.100:7002@17002 slave d9bf2ac8eec5637ed7ec50061419ff6b951eef0b 0 1598239337000 4 connected  6e0353e92377a71d691a853152673a8774d11dc2 192.168.88.101:7002@17002 master - 0 1598239338000 2 connected 5461-10922  d9bf2ac8eec5637ed7ec50061419ff6b951eef0b 192.168.88.102:7001@17001 master - 0 1598239339170 3 connected 10923-16383  97e1d381d59561e075ac813e2df7fed00114687e 192.168.88.100:7001@17001 myself,master - 0 1598239338000 1 connected 0-5460  127.0.0.1:7001> **info replication**  # Replication  role:master  connected\_slaves:1  slave0:ip=192.168.88.101,port=7001,state=online,offset=840,lag=1  master\_replid:e669abc55325a3ea3f4273aeed92c3370568fb34  master\_replid2:0000000000000000000000000000000000000000  master\_repl\_offset:854  second\_repl\_offset:-1  repl\_backlog\_active:1  repl\_backlog\_size:1048576  repl\_backlog\_first\_byte\_offset:1  repl\_backlog\_histlen:854 |

测试数据，设置Key值和查询Key的值。

|  |
| --- |
| **# node1(192.168.88.153:7001)，redis-cli登录**  127.0.0.1:7001> **KEYS \***  (empty list or set)  127.0.0.1:7001> **set k1 v1**  -> Redirected to slot [12706] located at 192.168.88.102:7001 # 自动定向到node03上主服务  OK  192.168.88.102:7001> **set k2 v2**  -> -> Redirected to slot [449] located at 192.168.88.100:7001 # 自动定向到node02上主服务  OK  192.168.88.100:7001> **set k3 v3**  OK  192.168.88.100:7001> **get k1**  -> Redirected to slot [12706] located at 192.168.88.102:7001 # 自动定向到node03上主服务  "v1"  192.168.88.102:7001> **get k2**  -> Redirected to slot [449] located at 192.168.88.100:7001 # 自动定向到node03上主服务  "v2"  192.168.88.102:7001> **get k3**  "v3"  192.168.88.100:7001> **KEYS \***  1) "k3"  2) "k2" |

#### 启动关闭集群

编写脚本，方便启动和关闭Redis集群：**redis-cluster-start.sh**和**redis-cluster-stop.sh**。

* 进入Redis安装目录中bin目录，创建脚本文件

|  |
| --- |
| **cd /export/server/redis-5.0.8-bin/bin/**  **touch redis-cluster-start.sh**  **touch redis-cluster-stop.sh**  # 给以执行权限  **chmod u+x redis-cluster-start.sh**  **chmod u+x redis-cluster-stop.sh** |

* 启动集群：**redis-cluster-start.sh**

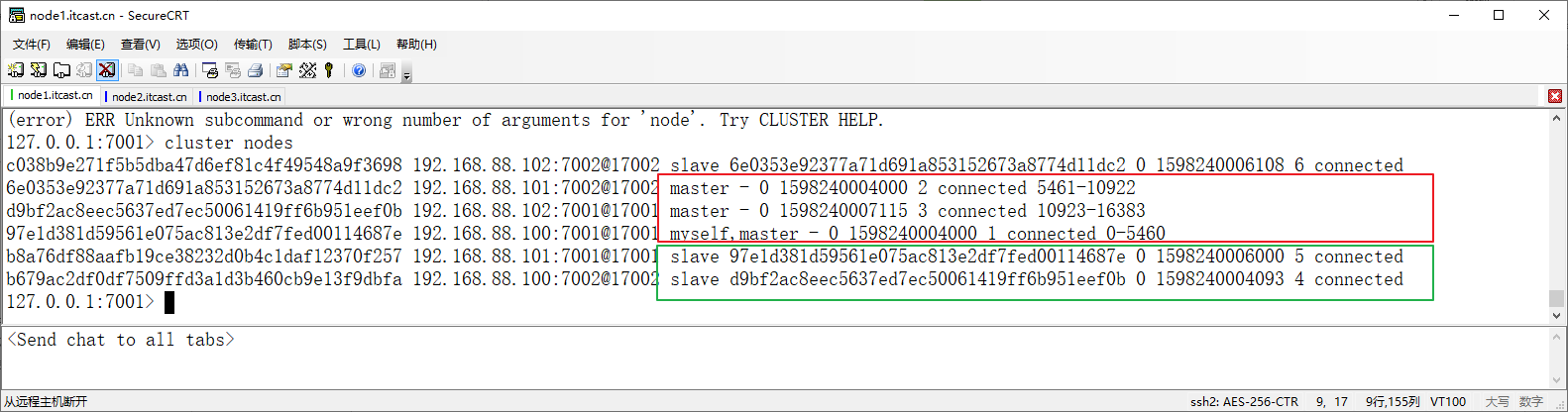
|  |
| --- |
| **vim redis-cluster-start.sh**  **#!/bin/bash**  **REDIS\_HOME=/export/server/redis**  # Start Server  ## node1.itcast.cn  **ssh node1.itcast.cn "${REDIS\_HOME}/bin/redis-server /export/server/redis/7001/redis\_7001.conf"**  **ssh node1.itcast.cn "${REDIS\_HOME}/bin/redis-server /export/server/redis/7002/redis\_7002.conf"**  ## node02  **ssh node2.itcast.cn "${REDIS\_HOME}/bin/redis-server /export/server/redis/7001/redis\_7001.conf"**  **ssh node2.itcast.cn "${REDIS\_HOME}/bin/redis-server /export/server/redis/7002/redis\_7002.conf"**  ## node03  **ssh node3.itcast.cn "${REDIS\_HOME}/bin/redis-server /export/server/redis/7001/redis\_7001.conf"**  **ssh node3.itcast.cn "${REDIS\_HOME}/bin/redis-server /export/server/redis/7002/redis\_7002.conf"** |

* 关闭集群：**redis-cluster-stop.sh**

|  |
| --- |
| vim redis-cluster-stop.sh  #!/bin/bash  **REDIS\_HOME=/export/server/redis**  # Stop Server  ## node01  **${REDIS\_HOME}/bin/redis-cli -h node1.itcast.cn -p 7001 SHUTDOWN**  **${REDIS\_HOME}/bin/redis-cli -h node1.itcast.cn -p 7002 SHUTDOWN**  ## node02  **${REDIS\_HOME}/bin/redis-cli -h node2.itcast.cn -p 7001 SHUTDOWN**  **${REDIS\_HOME}/bin/redis-cli -h node2.itcast.cn -p 7002 SHUTDOWN**  ## node03  **${REDIS\_HOME}/bin/redis-cli -h node3.itcast.cn -p 7001 SHUTDOWN**  **${REDIS\_HOME}/bin/redis-cli -h node3.itcast.cn -p 7002 SHUTDOWN** |

#### 主从切换

测试Redis Cluster中主从服务切换，首先查看集群各个服务状态：

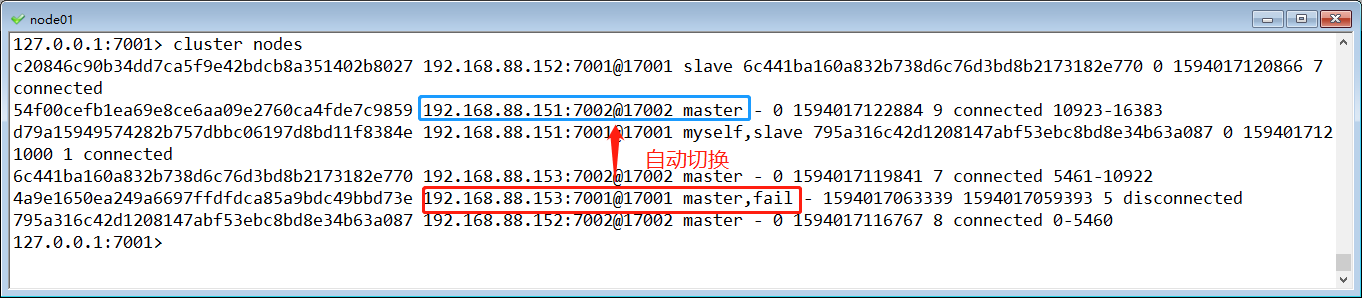


在node3上将7001端口Redis 服务关掉：**SHUTDOWN**

**../redis-cli -h node3.itcast.cn -p 7001 SHUTDOWN**

|  |
| --- |
| [root@node3 server]# ps aux | grep redis  root 112830 0.1 0.3 161164 14084 ? Ssl 11:32 0:00 /export/server/redis/bin/redis-server 0.0.0.0:7002 [cluster]  root 112993 0.0 0.0 112812 972 pts/1 S+ 11:36 0:00 grep --color=auto redis |

再次查看集群状态信息：



重新启动node03上7001端口Redis服务，查看集群状态信息

|  |
| --- |
| 127.0.0.1:7001> **cluster nodes**  c038b9e271f5b5dba47d6ef81c4f49548a9f3698 192.168.88.102:7002@17002 slave 6e0353e92377a71d691a853152673a8774d11dc2 0 1598240006108 6 connected  6e0353e92377a71d691a853152673a8774d11dc2 192.168.88.101:7002@17002 master - 0 1598240004000 2 connected 5461-10922  **d9bf2ac8eec5637ed7ec50061419ff6b951eef0b 192.168.88.102:7001@17001 master - 0 1598240007115 3 connected 10923-16383**  97e1d381d59561e075ac813e2df7fed00114687e 192.168.88.100:7001@17001 myself,master - 0 1598240004000 1 connected 0-5460  b8a76df88aafb19ce38232d0b4c1daf12370f257 192.168.88.101:7001@17001 slave 97e1d381d59561e075ac813e2df7fed00114687e 0 1598240006000 5 connected  b679ac2df0df7509ffd3a1d3b460cb9e13f9dbfa 192.168.88.100:7002@17002 slave d9bf2ac8eec5637ed7ec50061419ff6b951eef0b 0 1598240004093 4 connected  127.0.0.1:7001> redis-cli –h node3.itcast.cn –p 7001 SHUTDOWN  127.0.0.1:7001> cluster nodes  c038b9e271f5b5dba47d6ef81c4f49548a9f3698 192.168.88.102:7002@17002 slave 6e0353e92377a71d691a853152673a8774d11dc2 0 1598240223000 6 connected  6e0353e92377a71d691a853152673a8774d11dc2 192.168.88.101:7002@17002 master - 0 1598240224000 2 connected 5461-10922  **d9bf2ac8eec5637ed7ec50061419ff6b951eef0b 192.168.88.102:7001@17001 master,fail - 1598240171708 1598240170299 3 disconnected**  97e1d381d59561e075ac813e2df7fed00114687e 192.168.88.100:7001@17001 myself,master - 0 1598240224000 1 connected 0-5460  b8a76df88aafb19ce38232d0b4c1daf12370f257 192.168.88.101:7001@17001 slave 97e1d381d59561e075ac813e2df7fed00114687e 0 1598240224734 5 connected  **b679ac2df0df7509ffd3a1d3b460cb9e13f9dbfa 192.168.88.100:7002@17002 master - 0 1598240223000 7 connected 10923-16383** |

### Redis Cluster 管理

**redis-cli**集群命令帮助：

|  |
| --- |
| [root@node01 ~]# redis-cli --cluster help  Cluster Manager Commands:  create host1:port1 ... hostN:portN  --cluster-replicas <arg>  check host:port  --cluster-search-multiple-owners  info host:port  fix host:port  --cluster-search-multiple-owners  reshard host:port  --cluster-from <arg>  --cluster-to <arg>  --cluster-slots <arg>  --cluster-yes  --cluster-timeout <arg>  --cluster-pipeline <arg>  --cluster-replace  rebalance host:port  --cluster-weight <node1=w1...nodeN=wN>  --cluster-use-empty-masters  --cluster-timeout <arg>  --cluster-simulate  --cluster-pipeline <arg>  --cluster-threshold <arg>  --cluster-replace  add-node new\_host:new\_port existing\_host:existing\_port  --cluster-slave  --cluster-master-id <arg>  del-node host:port node\_id  call host:port command arg arg .. arg  set-timeout host:port milliseconds  import host:port  --cluster-from <arg>  --cluster-copy  --cluster-replace  help  For check, fix, reshard, del-node, set-timeout you can specify the host and port of any working node in the cluster. |

在实际项目中可能由于Redis Cluster中节点宕机或者增加新节点，需要操作命令管理，主要操作如下。



### JavaAPI操作redis集群

连接Redis集群，需要使用JedisCluster来获取Redis连接。

实现步骤：

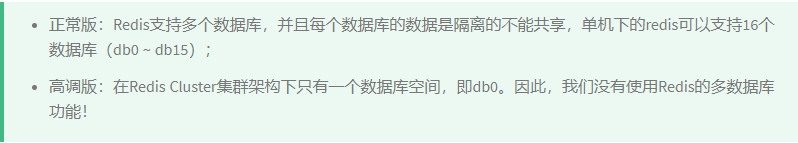
1. 在cn.itcast.redis.api\_test包下创建一个新的类：RedisClusterTest
2. 创建一个HashSet<HostAndPort>，用于保存集群中所有节点的机器名和端口号
3. 创建JedisPoolConfig对象，用于配置Redis连接池配置
4. 创建JedisCluster对象
5. 使用JedisCluster对象设置一个key，然后获取key对应的值

|  |
| --- |
| **public class** RedisClusterTest {  **private** JedisPoolConfig **jedisPoolConfig**;  **private** JedisCluster **jedisCluster**;   @BeforeTest  **public void** beforeTest() {  // 1. 创建一个HashSet<HostAndPort>，用于保存集群中所有节点的机器名和端口号  HashSet<HostAndPort> hostAndPortSet = **new** HashSet<>();  hostAndPortSet.add(**new** HostAndPort(**"node1.itcast.cn"**, 7001));  hostAndPortSet.add(**new** HostAndPort(**"node1.itcast.cn"**, 7002));  hostAndPortSet.add(**new** HostAndPort(**"node2.itcast.cn"**, 7001));  hostAndPortSet.add(**new** HostAndPort(**"node2.itcast.cn"**, 7002));  hostAndPortSet.add(**new** HostAndPort(**"node3.itcast.cn"**, 7001));  hostAndPortSet.add(**new** HostAndPort(**"node3.itcast.cn"**, 7002));   // 2. 创建JedisPoolConfig对象，用于配置Redis连接池配置  **jedisPoolConfig** = **new** JedisPoolConfig();  **jedisPoolConfig**.setMaxIdle(10);  **jedisPoolConfig**.setMinIdle(5);  **jedisPoolConfig**.setMaxWaitMillis(5000);  **jedisPoolConfig**.setMaxTotal(50);   // 3. 创建JedisCluster对象  **jedisCluster** = **new** JedisCluster(hostAndPortSet);  }   @Test  **public void** clusterOpTest() {  // 设置一个key  **jedisCluster**.set(**"pv"**, **"1"**);   // 获取key  System.***out***.println(**jedisCluster**.get(**"pv"**));  }   @AfterTest  **public void** afterTest() {  **try** {  **jedisCluster**.close();  } **catch** (IOException e) {  System.***out***.println(**"关闭Cluster集群连接失败！"**);  e.printStackTrace();  }  } } |

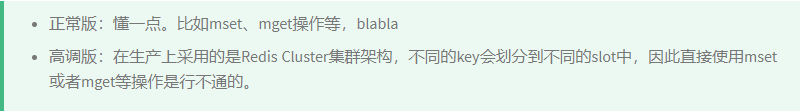
### Redis集群面试题

从Redis 3.0发布提供Redis Cluster以后，经历Redis 4.x、Redis5.x和Redis 6.x一系列版本，Redis Cluster更加成熟、稳定，推荐企业使用此种架构，通常公司也是使用此种架构。如果使用Redis Cluster集群，面试中碰到的问题有一些坑，还望注意。

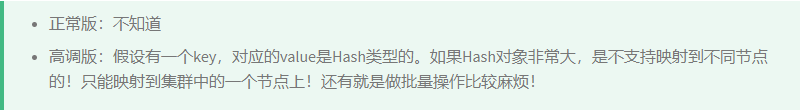
* 问题一：Redis的**多数据库**机制，了解多少？



* 问题二：懂Redis的**批量操作**么？



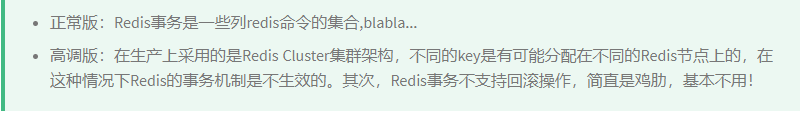
* 问题三：Redis集群机制中，你觉得有什么**不足**的地方吗？



* 问题四：在Redis集群模式下，如何进行**批量**操作？



* 问题五：懂Redis**事务**么？



## 常见问题

### make: cc：命令未找到，make: \*\*\* [adlist.o] 错误 127

解决方法：安装gcc，命令如下：yum install gcc

### Redis编译错误Killing still running Redis server 4966

|  |
| --- |
| Killing still running Redis server 4966 Killing still running Redis server 4971 Killing still running Redis server 4976 Killing still running Redis server 4978 Killing still running Redis server 4980 Killing still running Redis server 4983 Killing still running Redis server 4990 Killing still running Redis server 4991 Killing still running Redis server 4998 Killing still running Redis server 5001 Killing still running Redis server 5014 Killing still running Redis server 5134 Killing still running Redis server 5187 Killing still running Redis server 5208 Killing still running Redis server 5224 Killing still running Redis server 5253 Killing still running Redis server 5265 make[1]: \*\*\* [test] Error 1 make[1]: Leaving directory `/opt/redis/redis-4.0.2/src' make: \*\*\* [test] Error 2 |

解决方案：

|  |
| --- |
| vim tests/integration/replication-2.tcl  将after 1000修改为after 10000 |

after是tcl脚本中的命令，表示延迟程序执行或者在后台执行命令。

### MISCONF Redis is configured to save RDB snapshots

异常信息：

|  |
| --- |
| redis.clients.jedis.exceptions.JedisDataException: MISCONF Redis is configured to save RDB snapshots, but is currently not able to persist on disk. Commands that may modify the data set are disabled. Please check Redis logs for details about the error. |

解决方案：

因为强制关闭Redis快照导致不能持久化。可以使用kill -9再次强制关闭掉redis，然后在重新启动。

### Connection error: Connection refused

修改配置文件/etc/redis.conf，并注释掉bind 127.0.0.1这一行

### Connection: Connection error: The remote host closed the connection

修改配置文件/etc/redis.conf，将protected-mode为no。

### Node 192.168.88.101:7002 is not empty. Either the node already knows other nodes (check with CLUSTER NODES) or contains some key in database 0.

1) 先将redis 进程干掉   ps -ef | grep redis kill pid

2) 将每个节点下aof、rdb、nodes.conf本地备份文件删除,redis.conf   appendfilename ；  
3) 之后再执行脚本，成功执行；