redis主从复制

# redis主从复制

## 概述

Redis的复制功能是支持多个数据库之间的数据同步。一类是主数据库（master）一类是从数据库（slave），主数据库可以进行读写操作，当发生写操作的时候自动将数据同步到从数据库，而从数据库一般是只读的，并接收主数据库同步过来的数据，一个主数据库可以有多个从数据库，而一个从数据库只能有一个主数据库。

通过redis的复制功能可以很好的实现数据库的读写分离，提高服务器的负载能力。主数据库主要进行写操作，而从数据库负责读操作。

## 主从复制过程



过程：

* 当一个从数据库启动时，会向主数据库发送sync命令，
* 主数据库接收到sync命令后会开始在后台保存快照（执行rdb操作），并将保存期间接收到的命令缓存起来
* 当快照完成后，redis会将快照文件和所有缓存的命令发送给从数据库。
* 从数据库收到后，会载入快照文件并执行收到的缓存的命令。

注意：redis2.8之前版本：当主从数据库同步的时候从数据库因为网络原因断开重连后会重新执行上述操作，不支持断点续传。redis2.8之后支持断点续传。

## redis主从配置

|  |
| --- |
| 方式1：手动修改配置文件  只需要额外修改从节点中redis的配置文件中的slaveof属性即可  bind 127.0.0.1 10.10.43.96  slaveof 172.16.130.169 6379 |
| 方式2：动态设置（命令方式）  slaveof 172.16.130.169 6379 |

注意事项：如果你使用主从复制，那么要确保你的master激活了持久化，或者确保它不会在当掉后自动重启。原因：slave是master的完整备份，因此如果master通过一个空数据集重启，slave也会被清掉。

在配置redis复制功能的时候如果主数据库设置了密码，需要在从数据的配置文件中通过masterauth参数设置主数据库的密码，这样从数据库在连接主数据库时就会自动使用auth命令认证了。相当于做了一个免密码登录。

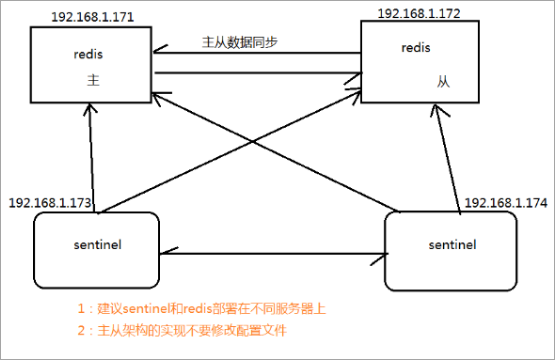
# redis的Sentinel

## sentinel功能

redis的sentinel系统用于管理多个redis服务器，该系统主要执行三个任务：监控、提醒、自动故障转移。

* 监控（Monitoring）： Redis Sentinel实时监控主服务器和从服务器运行状态，并且实现自动切换。
* 提醒（Notification）：当被监控的某个 Redis 服务器出现问题时， Redis Sentinel 可以向系统管理员发送通知， 也可以通过 API 向其他程序发送通知。
* 自动故障转移（Automatic failover）： 当一个主服务器不能正常工作时，Redis Sentinel 可以将一个从服务器升级为主服务器， 并对其他从服务器进行配置，让它们使用新的主服务器。当应用程序连接Redis 服务器时， Redis Sentinel会告之新的主服务器地址和端口。

注意：在使用sentinel监控主从节点的时候，从节点需要是使用动态方式配置的，如果直接修改配置文件，后期sentinel实现故障转移的时候会出问题。



主观下线状态：当一个sentinel认为一个redis服务连接不上的时候，会给这个服务打个标记为下线状态。

客观下线状态：当多个sentinel认为一个redids连接不上的时候，则认为这个redis服务确实下线了。这里的多个sentinel的个数可以在配置文件中设置。

主节点：主观下线和客观下线

从节点：主观下线状态

## sentinel配置

|  |
| --- |
| #1修改sentinel.conf文件  sentinel monitor mymaster 192.168.33.130 6379 2 #最后一个参数视情况决定  http://img.blog.csdn.net/20160503192616532  最后一个参数为需要判定客观下线所需的主观下线sentinel个数，这个参数不可以大于sentinel个数。 |
| #2启动sentinel  redis-sentinel sentinel.conf  http://img.blog.csdn.net/20160503192658763 |

通过订阅指定的频道信息，当服务器出现故障得时候通知管理员。客户端可以将 Sentinel 看作是一个只提供了订阅功能的 Redis 服务器，你不可以使用 PUBLISH 命令向这个服务器发送信息，但你可以用 SUBSCRIBE 命令或者 PSUBSCRIBE 命令， 通过订阅给定的频道来获取相应的事件提醒。

一个频道能够接收和这个频道的名字相同的事件。 比如说， 名为 +sdown 的频道就可以接收所有实例进入主观下线（SDOWN）状态的事件。

## sentinel的一些命令

|  |
| --- |
| #sentinel的基本状态信息  INFO  #列出所有被监视的主服务器，以及这些主服务器的当前状态  SENTINEL masters  #列出给定主服务器的所有从服务器，以及这些从服务器的当前状态  SENTINEL slaves <master name>  #返回给定名字的主服务器的 IP 地址和端口号  SENTINEL get-master-addr-by-name <master name>  #重置所有名字和给定模式 pattern 相匹配的主服务器。重置操作清除主服务器目前的所有状态， 包括正在执行中的故障转移， 并移除目前已经发现和关联的， 主服务器的所有从服务器和 Sentinel 。  SENTINEL reset <pattern>  #当主服务器失效时， 在不询问其他 Sentinel 意见的情况下， 强制开始一次自动故障迁移，但是它会给其他sentinel发送一个最新的配置，其他sentinel会根据这个配置进行更新  SENTINEL failover <master name> |

## java操作sentinel

|  |
| --- |
| import java.util.HashSet;  //需要在pom.xml文件中引入jedis依赖  import redis.clients.jedis.HostAndPort;  import redis.clients.jedis.Jedis;  import redis.clients.jedis.JedisPoolConfig;  import redis.clients.jedis.JedisSentinelPool;    public class SentinelTest {    public static void main(String[] args) {  // 使用HashSet添加多个sentinel  HashSet<String> sentinels = new HashSet<String>();  // 添加sentinel主机和端口  sentinels.add("192.168.33.131:26379");    // 创建config  JedisPoolConfig poolConfig = new JedisPoolConfig();  // 控制一个pool最多有多少个状态为idle(空闲的)的jedis实例。  poolConfig.setMaxIdle(10);  // 控制一个pool最多有多少个jedis实例。  poolConfig.setMaxTotal(100);  // 表示当borrow(引入)一个jedis实例时，最大的等待时间，如果超过等待时间，则直接抛出JedisConnectionException；  poolConfig.setMaxWaitMillis(2000);  // 在borrow一个jedis实例时，是否提前进行validate操作；如果为true，则得到的jedis实例均是可用的；  poolConfig.setTestOnBorrow(true);    // 通过Jedis连接池创建一个Sentinel连接池  JedisSentinelPool pool = new JedisSentinelPool("mymaster", sentinels,poolConfig);  // 获取master的主机和端口  HostAndPort currentHostMaster = pool.getCurrentHostMaster();  System.out.println(currentHostMaster.getHost() + "--"+ currentHostMaster.getPort());  // 从Sentinel池中获取资源  Jedis resource = pool.getResource();  // 打印资源中key为name的值  System.out.println(resource.get("name"));  // 关闭资源  resource.close();  }  } |

# redis集群

## Redis集群简介

redis集群是一个无中心的分布式Redis存储架构，可以在多个节点之间进行数据共享，解决了Redis高可用、可扩展等问题。redis集群提供了以下两个好处

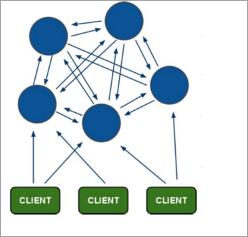
* 将数据自动切分(split)到多个节点
* 当集群中的某一个节点故障时，redis还可以继续处理客户端的请求。

一个 Redis 集群包含 16384 个哈希槽（hash slot），数据库中的每个数据都属于这16384个哈希槽中的一个。集群使用公式 CRC16(key) % 16384 来计算键 key 属于哪个槽。集群中的每一个节点负责处理一部分哈希槽。

集群中的主从复制：集群中的每个节点都有1个至N个复制品，其中一个为主节点，其余的为从节点，如果主节点下线了，集群就会把这个主节点的一个从节点设置为新的主节点，继续工作。这样集群就不会因为一个主节点的下线而无法正常工作。

注意：如果某一个主节点和他所有的从节点都下线的话，redis集群就会停止工作了。redis集群不保证数据的强一致性，在特定的情况下，redis集群会丢失已经被执行过的写命令。

使用异步复制（asynchronous replication）是 Redis 集群可能会丢失写命令的其中一个原因，有时候由于网络原因，如果网络断开时间太长，redis集群就会启用新的主节点，之前发给主节点的数据就会丢失。



## Redis集群安装配置

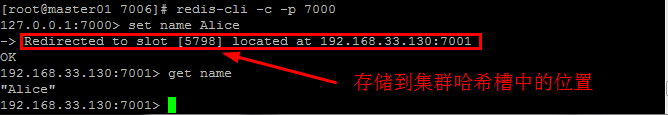
|  |
| --- |
| daemonize yes  port 6379  cluster-enabled yes  cluster-config-file nodes.conf  cluster-node-timeout 5000 |
| 要让集群正常运作至少需要三个主节点  我们这里就简单在一台主机上创建6个redis节点来演示集群配置，实际生产环境中需要每个节点一台主机。我们要创建的6个redis节点，其中三个为主节点，三个为从节点，对应的redis节点的ip和端口对应关系如下：  192.168.33.130:7000  192.168.33.130:7001  192.168.33.130:7002  192.168.33.130:7003  192.168.33.130:7004  192.168.33.130:7005  #1首先我们创建6个以端口为名称的文件夹（由于每个redis节点启动的时候，都会在当前文件夹下创建快照文件，所以我们需要创建每个节点的启动目录）  mkdir 7000  mkdir 7001  mkdir 7002  mkdir 7003  mkdir 7004  mkdir 7005  #2接下来把每个节点启动所需要的配置文件拷贝到相应的启动目录  cp redis.conf 7000  cp redis.conf 7001  cp redis.conf 7002  cp redis.conf 7003  cp redis.conf 7004  cp redis.conf 7005  #3然后我们进入每个启动目录，修改之前拷贝的redis.conf文件中的端口port 为上面列出的对应端口。最终每个节点的配置类似于：  daemonize yes  port 6379 #只有端口不同，其他相同  cluster-enabled yes  cluster-config-file nodes.conf  cluster-node-timeout 5000  #4进入每个启动目录，以每个目录下的redis.conf文件启动  http://img.blog.csdn.net/20160503193225784  使用命令查看redis节点是否启动：ps -ef | grep redis  http://img.blog.csdn.net/20160503193239753  #5创建集群命令  redis-trib.rb create --replicas 1 192.168.33.130:7000 192.168.33.130:7001 192.168.33.130:7002 192.168.33.130:7003 192.168.33.130:7004 192.168.33.130:7005  注意：执行上面的命令的时候可能会报错，因为是执行的ruby的脚本，需要ruby的环境。所以我们需要安装ruby的环境，这里推荐使用yum安装：  yum install ruby  yum install rubygems  gem install redis  http://img.blog.csdn.net/20160503193550973  http://img.blog.csdn.net/20160503193601213  这里输入yes，最后结果：  http://img.blog.csdn.net/20160503193631526  #6接下来我们使用命令进入集群环境  redis-cli -c -p 7000 |

## redis集群操作

|  |
| --- |
| #1查看集群中的所有主节点信息  redis-cli -c -p 7000 cluster nodes [| grep master]  http://img.blog.csdn.net/20160503193808661  #2redis集群添加节点  根据添加节点类型的不同，有两种方法来添加新节点   * 主节点：如果添加的是主节点，那么我们需要创建一个空节点，然后将某些哈希槽移动到这个空节点里面 * 从节点：如果添加的是从节点，我们也需要创建一个空节点，然后把这个新节点设置成集群中某个主节点的复制品。   第一步：首先把需要添加的节点启动，创建7006目录，拷贝7000中的redis.conf到7006中，然后修改端口port为7006，修改好后进入7006目录启动这个节点：  redis-server redis.conf  第二步：执行以下命令，将这个新节点添加到集群中：  redis-trib.rb add-node 192.168.33.130:7006 192.168.33.130:7000  http://img.blog.csdn.net/20160503193931202  第三步：执行命令查看刚才新增的节点  redis-cli -c -p 7000 cluster nodes  http://img.blog.csdn.net/20160503193955858  第四步：增加了新的节点之后，这个新的节点可以成为主节点或者是从节点。  -- 将这个新增节点变成从节点，前面我们已经把这个新节点添加到集群中了，现在我们要让新节点成为192.168.33.130:7001的从节点，只需要执行下面的命令就可以了，命令后面的节点ID就是192.168.33.130:7001的节点ID。（注意，这个从节点哈希槽必须为空，如果不为空，则需要转移掉哈希槽使之为空）  redis-cli -c -p 7006 cluster replicate a246963893faf03c45cc19ef4188f82f5393bfef  http://img.blog.csdn.net/20160503194045078  使用下面命令来确认一下192.168.33.130:7006是否已经成为192.168.33.130:7001的从节点。  redis-cli -p 7000 cluster nodes | grep slave | grep a246963893faf03c45cc19ef4188f82f5393bfef  http://img.blog.csdn.net/20160503194109616  -- 将这个新增节点变成主节点，使用redis-trib程序，将集群中的某些哈希槽移动到新节点里面，这个新节点就成为真正的主节点了。执行下面的命令对集群中的哈希槽进行移动：redis-trib.rb reshard 192.168.33.130:7000  命令执行后，系统会提示我们要移动多少哈希槽，这里移动1000个  http://img.blog.csdn.net/20160503194143163  然后还需要指定把这些哈希槽转移到哪个节点上  http://img.blog.csdn.net/20160503194159944  输入我们刚才新增的节点的ID：d113e0f033c98e2f6b88fb93e6e98866256d85c4  然后需要我们指定转移哪几个几点的哈希槽，  http://img.blog.csdn.net/20160503194229569  输入all 表示从所有的主节点中随机转移，凑够1000个哈希槽  然后再输入yes，redis集群就开始分配哈希槽了。  http://img.blog.csdn.net/20160503194246112  至此，一个新的主节点就添加完成了，执行命令查看现在的集群中节点的状态  redis-cli -c -p 7000 cluster nodes  http://img.blog.csdn.net/20160503194313629 |
| #3Redis集群删除节点  第一步：如果删除的节点是主节点，这里我们删除192.168.33.130:7006节点，这个节点有1000个哈希槽,首先要把节点中的哈希槽转移到其他节点中，执行下面的命令：redis-trib.rb reshard 192.168.33.130:7000  系统会提示我们要移动多少哈希槽，这里移动1000个，因为192.168.33.130:7006节点有1000个哈希槽。  http://img.blog.csdn.net/20160503194344992  第二步：然后系统提示我们输入要接收这些哈希槽的节点的ID，这里使用192.168.33.130:7001的节点ID  http://img.blog.csdn.net/20160503194431570  最后输入done表示输入完毕。  最后一步，使用下面的命令把这个节点删除  redis-trib.rb del-node 192.168.33.130:7000 d113e0f033c98e2f6b88fb93e6e98866256d85c4 //最后一个参数为需要删除的节点ID  如果是从节点，直接删除即可  redis-trib.rb del-node 192.168.33.130:7000 d113e0f033c98e2f6b88fb93e6e98866256d85c4 //最后一个参数为需要删除节点的ID  http://img.blog.csdn.net/20160503194541774 |

## java操作redis集群

向Redis集群中存入键值：



|  |
| --- |
| 代码示例：  import java.util.HashSet;  //需要再pom.xml中引入jedis依赖  import redis.clients.jedis.HostAndPort;  import redis.clients.jedis.JedisCluster;  import redis.clients.jedis.JedisPool;  import redis.clients.jedis.JedisPoolConfig;    public class RedisCluster {  public static void main(String[] args) {  //初始化集合，用于装下面的多个主机和端口  HashSet<HostAndPort> nodes = new HashSet<HostAndPort>();    //创建多个主机和端口实例  HostAndPort hostAndPort = new HostAndPort("192.168.33.130", 7000);  HostAndPort hostAndPort1 = new HostAndPort("192.168.33.130", 7001);  HostAndPort hostAndPort2 = new HostAndPort("192.168.33.130", 7002);  HostAndPort hostAndPort3 = new HostAndPort("192.168.33.130", 7003);  HostAndPort hostAndPort4 = new HostAndPort("192.168.33.130", 7004);  HostAndPort hostAndPort5 = new HostAndPort("192.168.33.130", 7005);    //添加多个主机和端口到集合中  nodes.add(hostAndPort);  nodes.add(hostAndPort1);  nodes.add(hostAndPort2);  nodes.add(hostAndPort3);  nodes.add(hostAndPort4);  nodes.add(hostAndPort5);    //创建config  JedisPoolConfig poolConfig = new JedisPoolConfig();  //通过config创建集群实例  JedisCluster jedisCluster = new JedisCluster(nodes,poolConfig);  //获取集群中的key为name键的值  String str = jedisCluster.get("name");  System.out.println(str);  }  }  http://img.blog.csdn.net/20160503194700400 |