# Redis



# 为什么需要配置Redis分布式集群？-解决高性能问题

|  |
| --- |
| 首先，Redis支持配置主从，实现读写分离，配置一主多从+哨兵机制，实现主机挂了，可以在从机中选举一台作为主机，解决单点故障问题。  但是依然会面临下面的问题：   1. 高并发，虽然Redis官方数据是10万/秒，但是如果我们的应用需要达到100万/秒怎么办？ 2. 高容量，一般服务器的内存是在16-256G内存之间，如果要达到500G以上的存储容量，怎么办？ 3. 网络流量的问题，假设一台机器的网卡是千兆网卡，当我们的应用网络流量超过这个上限怎么办？   这些都是实际会面临的问题，那么就需要我们采用另一种方案，Redis分布式集群。 |

# 搭建Redis分布式集群简介

|  |
| --- |
| Redis的集群架构图如下：  C:\Users\zhijun\Documents\My Knowledge\temp\4dedc425-875d-454a-a734-51cd20cfb7a6_4_files\6e54d7ef-7d67-441a-8edf-23d7a93d65c1.jpg  **注意：以上的集群架构是Redis3.0的新特性（RedisCluster）**   * Client与redis节点直接连接,**不需要中间proxy层**. * redis-cluster把所有的物理节点映射到[0-16383]slot（插槽）上,cluster 负责维护，每个节点负责保存部分的插槽信息 * 所有的redis节点彼此互联(PING-PONG机制),内部使用gossip二进制协议优化传输数据 * 节点的失效检测是通过集群中**超过半数**的节点检测失效时才生效.   算法说明（了解即可）：  问题：Redis 集群中内置了 16384 个哈希槽，那他是如何决定将key放入哪个插槽的？  当Redis 集群中放置一个 key-value 时，redis 先对 key 使用 crc16 算法算出一个结果，然后把结果对 16384 求余数，这样每个 key 都会对应一个编号在 0-16383 之间的哈希槽，这样就可以确定数据应该保存到哪个插槽，从而确定应该保存到哪个节点 |

# 搭建Redis分布式

|  |
| --- |
| **真实的情况，应该最少是三台**  **实验的情况，一台，用不同端口来区分**  **操作步骤如下：**   1. 在redis3目录下，复制三份bin 2. 创建redis\_cluster 来保存三份bin6379,bin6380,bin6381   **配置集群之前：注意要删除每个reids目录下dump.rdb等相关数据文件**  修改redis.conf，配置集群信息  2、开启集群，cluster-enabled yes  3、指定集群的配置文件，cluster-config-file "nodes-\*\*\*\*.conf"  这几个\*\*\*就是修改为自己的端口，每个配置文件要不一样    启动每个redis，查看启动的结果    4、用redis-trib.rb搭建集群  因为redis-trib.rb是用Ruby实现的Redis集群管理工具，所以我们需要先安装ruby的环境  4.1、安装ruby  yum -y install zlib ruby rubygems  4.2、安装rubygems的redis依赖  gem install -l redis-3.3.0.gem  4.3、安装好依赖的环境之后，我们就可以来使用脚本命令了  **注意：搭建集群前，需要先将各节点的认证密码先注释掉，这样才能互联**  注意：此ruby脚本文件在我们的**解压缩目录**src下    ./redis-trib.rb create --replicas 0 192.168.10.167:6379 192.168.10.167:6380 192.168.10.167:6381  --replicas 0：指定了从数据的数量为0  观察提示信息，查看集群的分配结果    **查看集群的状态**  通过客户端输入以下命令：  cluster nodes：这个命令可以查看插槽的分配情况  整个Redis提供了16384个插槽，./redis-trib.rb 脚本实现了是将16384个插槽平均分配给了N个节点。  **测试集群：**  使用任意一个redis客户端进行数据的存储和获取，观察结果  redis-cli 只在当前连接的节点上操作  redis-cli -c 会自动帮我们调整到合适的节点上操作  **经验分享：**  如果重新启动集群报以下错误  [ERR] Node 192.168.10.137:6379 is not empty. Either the node already knows other nodes (check with CLUSTER NODES) or contains some key in database 0.  **处理办法：**  需要清除杀掉redis实例，然后删除每个节点下的临时数据文件appendonly.aof，dump.rdb，nodes-6379.conf，然后再重新启动redis实例即可启动集群。  关闭进程，要kill才真正有效  **血的经验教训：**  关于彼此之间实现互联，需要注意的地方：  注意：为了保证集群可以搭建成功，需要先将密码去掉，搭建成功之后，再加回来即可  应该是把protected-mode修改为no，而不是注释掉 |

# 搭建高可用的集群（分布式+集群）

|  |
| --- |
| 问题：上述的集群如果有一个节点挂了，怎么办？  集群中的故障检查机制   1. 当集群中的节点超过半数认为该目标节点疑似下线，那么该节点就会被标记为**下线**； 2. 当集群中的任何一个节点下线，就会导致插槽区有空档，不完整，那么该集群将不可用；   **解决方案：**  在Redis集群中可以使用**主从模式**实现某一个节点的高可用  **目标：**  我们来搭建一个三主，三备的集群  第一：先要将相关的服务停掉，清楚掉相关的文件（**集群配置文件nodes**），  第二：复制多三台实例，bin6382,bin6383,bin6384  第三：启动所有的服务实例      第二：创建新集群，且设置为一主一备  ./redis-trib.rb create --replicas **1** 192.168.10.167:6379 192.168.10.167:6380 192.168.10.167:6381 192.168.10.167:6382 192.168.10.167:6383 192.168.10.167:6384    第三：测试主机挂了之后，集群是否可用  中间的转换大概需要几秒钟才能反应过来  第四：重新恢复失效的节点 |

# Java连接redis集群-Jedis版本

|  |
| --- |
| **public** **class** JedisTest {  @Test  **public** **void** set(){  Set<HostAndPort> nodes = **new** HashSet<HostAndPort>();  nodes.add(**new** HostAndPort("192.168.10.167", 6379));  nodes.add(**new** HostAndPort("192.168.10.167", 6380));  nodes.add(**new** HostAndPort("192.168.10.167", 6381));  nodes.add(**new** HostAndPort("192.168.10.167", 6382));  nodes.add(**new** HostAndPort("192.168.10.167", 6383));  nodes.add(**new** HostAndPort("192.168.10.167", 6384));  JedisCluster jCluster = **new** JedisCluster(nodes);  String result = jCluster.set("kebi", "神一样的男人");  System.***out***.println(result);  String kebi = jCluster.get("kebi");  System.***out***.println(kebi);  }  }  如需要设置密码，请换一另一个构造方法来构造JedisCluster |

# Spring整合Jedis

|  |
| --- |
| 配置spring配置文件  <!-- 配置redis客户端集群版 -->  <bean id=*"jedisCluster"* class=*"redis.clients.jedis.JedisCluster"*>  <constructor-arg>  <set>  <bean class=*"redis.clients.jedis.HostAndPort"*>  <constructor-arg name=*"host"* value=*"192.168.10.167"* />  <constructor-arg name=*"port"* value=*"6379"* />  </bean>  ......  </set>  </constructor-arg>  </bean>  编写单元测试  @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration(locations="classpath:spring/\*.xml")  **public** **class** JedisClusterTest {  @Autowired  **private** JedisCluster jedisCluster;  @Test  **public** **void** set() {  jedisCluster.set("kebi", "good");  System.***out***.println(jedisCluster.get("kebi"));  }  }  此处，我们已经完全可以后端的redis实现通信 |

# Spring-data-redis整合redis

|  |
| --- |
|  |

# 配置连接池对象

|  |
| --- |
| <!-- 配置redis连接池对象 -->  <bean id=*"poolConfig"* class=*"redis.clients.jedis.JedisPoolConfig"*>  <!-- 最大空闲数 -->  <property name=*"maxIdle"* value=*"50"* />  <!-- 最大连接数 -->  <property name=*"maxTotal"* value=*"100"* />  <!-- 最大等待时间 -->  <property name=*"maxWaitMillis"* value=*"20000"* />  </bean> |

# Redis集群配置

|  |
| --- |
| <!-- Redis集群配置 -->  <bean id=*"redisClusterConfiguration"*  class=*"org.springframework.data.redis.connection.RedisClusterConfiguration"*>  <property name=*"clusterNodes"*>  <set>  <bean class=*"org.springframework.data.redis.connection.RedisNode"*>  <constructor-arg name=*"host"* value=*"192.168.10.172"*/>  <constructor-arg name=*"port"* value=*"6379"*/>  </bean>  <bean class=*"org.springframework.data.redis.connection.RedisNode"*>  <constructor-arg name=*"host"* value=*"192.168.10.172"*/>  <constructor-arg name=*"port"* value=*"6380"*/>  </bean>  <bean class=*"org.springframework.data.redis.connection.RedisNode"*>  <constructor-arg name=*"host"* value=*"192.168.10.172"*/>  <constructor-arg name=*"port"* value=*"6381"*/>  </bean>  <bean class=*"org.springframework.data.redis.connection.RedisNode"*>  <constructor-arg name=*"host"* value=*"192.168.10.172"*/>  <constructor-arg name=*"port"* value=*"6382"*/>  </bean>  <bean class=*"org.springframework.data.redis.connection.RedisNode"*>  <constructor-arg name=*"host"* value=*"192.168.10.172"*/>  <constructor-arg name=*"port"* value=*"6383"*/>  </bean>  <bean class=*"org.springframework.data.redis.connection.RedisNode"*>  <constructor-arg name=*"host"* value=*"192.168.10.172"*/>  <constructor-arg name=*"port"* value=*"6384"*/>  </bean>  </set>  </property>  </bean> |

# 配置连接工厂

|  |
| --- |
| <!-- 配置redis连接工厂 -->  <bean id=*"connectionFactory"*  class=*"org.springframework.data.redis.connection.jedis.JedisConnectionFactory"*>  <constructor-arg name=*"poolConfig"* ref=*"poolConfig"*/>  <constructor-arg name=*"clusterConfig"* ref=*"redisClusterConfiguration"*/>  <!-- 密码 -->  <property name=*"password"* value=*"huangguizhao"* />  </bean> |

# 配置模板

|  |
| --- |
| <!-- 配置redis模板对象 -->  <bean id=*"redisTemplate"* class=*"org.springframework.data.redis.core.RedisTemplate"*>  <!-- 配置连接工厂 -->  <property name=*"connectionFactory"* ref=*"connectionFactory"* />  <property name=*"keySerializer"* ref=*"stringRedisSerializer"* />  </bean> |

# 使用模板对象

|  |
| --- |
| 直接使用redisTemplate对象，跟之前操作单机无差别 |

总结

|  |
| --- |
|  |