# Redis的主从和哨兵机制-解决高可用问题

## 单机存在的问题

|  |
| --- |
| 1. 高可用的问题 2. 高性能的问题   本篇文章，主要来解决高可用的问题 |

## 配置主从关系

|  |
| --- |
| 1. 在从机执行命令的方式来配置主从关系   在从机执行slaveof ip port 就会自动从主机开始复制   1. 通过配置文件的方式来配置主从   slaveof ip port  Slave-read-only yes   1. 在节点上执行如下命令，查看复制状态信息   info replication  Replication:复制   1. 测试验证结果：   在主节点操作数据，在从节点查看是否有同步   1. 断开复制：   执行slaveof no one 则会解除关系   1. 解决安全性问题   主节点设置auth  从节点设置masterauth   1. 解决传输的延迟问题   Repl-disable-tcp-nodelay，当关闭时，主节点产生的命令无论大小都会及时发送给从节点，这样，延迟会变小，但网络带宽消耗较大，适用于主从部署在同个机房的情况  而如果是跨机房部署，建议打开，这样主节点会合并较小的TCP数据包从而节省带宽，默认发送时间间隔为40毫秒  是否禁用tcp不延迟传输：no  就是不禁用 |

## 主从复制的流程

|  |
| --- |
| 主要分为6个步骤：   1. 从节点保存主节点信息 2. 主从建立socket连接 3. 从节点发送ping命令，等待主节点返回pong回应 4. 权限验证 5. 主从连接正常后，开始同步数据集，首次建立复制，是全量复制的方式 6. 持续的主从复制，后续主节点发生数据变更，会继续给从节点发送命令，此处采用增量复制   说明：  主节点和从节点都会记录一个叫复制偏移量的值，来确定从哪开始继续复制。  通过在主从节点执行info repplication 对比offset的值即可  这个也是检查主从复制的健康度依据 |

## 主从的多种拓扑结构

|  |
| --- |
| 1. 一主一从 2. 一主多从   适用于读多写少的场景，如果是写多读少的场景，多个从节点会导致主节点写命令的多次同步而过度消耗网络带宽   1. 树状主从结构   就是将从节点变成多层，这样可以减少主节点需要传输给从节点的数据量 |

# 哨兵

## 为什么需要哨兵模式

|  |
| --- |
| 主从模式存在一个问题，如果主节点挂了，那么需要人工的方式来将某个从节点设置为主节点，这样的手工维护方式不太智能。  人工处理，要经历以下几个步骤：   1. 选择一个从节点，执行slaveof no one，使其成为新主节点 2. 从节点成为主节点后，更新应用方的主节点信息，然后重启应用方 3. 其他从节点，跟新的主节点建立主从关系 slaveof ip port 4. 等原先的主节点恢复后，让它去复制新的主节点   Redis从2.8开始，通过哨兵机制Redis Sentinel解决了这个问题，从而达到高可用。  说明：正确来说应该是2.6版就提供，但是功能还不是很健全，所以投入实际项目应用，建议使用2.8以上版本 |

## 哨兵Redis Sentinel的架构图

|  |
| --- |
| 哨兵架构的本质：就是在主从的架构基础上多了监控的机制 |

## 哨兵机制的工作步骤

|  |
| --- |
| 1. 每个哨兵节点通过定时监控发现主节点出现故障 2. 多个哨兵节点通过主节点的故障达成一致，并选择出一个哨兵节点作为领导者负责本次故障转移处理 3. 哨兵领导节点执行故障转移，整个过程自动完成，无需人工干预 |

## 搭建哨兵架构

|  |
| --- |
| 注意：哨兵节点也是Redis节点，只是他们不存储数据  整体上，  我们至少需要3个Redis节点来做哨兵，构成一主多从，因为哨兵的领导者选举需要至少一半加一个节点，就是过半数机制。  再需要3个redis节点来做主从，实现读写分离    搭建步骤：   1. 搭建一主多从的读写分离架构，这个跟之前没有差异。      1. 配置sentinel节点   端口分别为26379,26380,26381  **拷贝解压缩目录下的sentinel.conf 到bin目录下，并执行修改**  **注意：如果要让远程连接，则127.0.0.1要修改成实际的服务器IP地址**         1. 启动哨兵服务（**此项需要自己添加**）        1. 然后关闭主节点，观察故障的自动处理结果   **注意，自动恢复需要一定的处理时间，并不是立马生效** |

## 通过哨兵查看主从情况

|  |
| --- |
| 注意要通过哨兵的端口登录，26379  Sentinel masters  Sentinel master <master name>  Sentinel slaves <master name> |

## 哨兵总结

|  |
| --- |
|  |

## Spring配置哨兵模式

|  |
| --- |
| <!-- 哨兵配置 -->  <bean id="sentinelConfig" class="org.springframework.data.redis.connection.RedisSentinelConfiguration">  <!-- 配置master主服务的名称 -->  <property name="master">  <bean class="org.springframework.data.redis.connection.RedisNode">  <property name="name" value="mymaster"/>  </bean>  </property>  <!-- 配置哨兵服务 -->  <property name="sentinels">  <set>  <bean class="org.springframework.data.redis.connection.RedisNode">  <constructor-arg name="host" value="192.168.10.171"/>  <constructor-arg name="port" value="26379"/>  </bean>  <bean class="org.springframework.data.redis.connection.RedisNode">  <constructor-arg name="host" value="192.168.10.171"/>  <constructor-arg name="port" value="26380"/>  </bean>  <bean class="org.springframework.data.redis.connection.RedisNode">  <constructor-arg name="host" value="192.168.10.171"/>  <constructor-arg name="port" value="26381"/>  </bean>  </set>  </property>  </bean>    <!-- 配置redis连接工厂 -->  <bean id="connectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory">  <!-- 配置哨兵 -->  <constructor-arg name="sentinelConfig" ref="sentinelConfig"/>  <!-- 连接池配置 -->  <constructor-arg name="poolConfig" ref="poolConfig"/>  <!-- 密码 -->  <property name="password" value="root"/>  </bean> |

## 开放哨兵访问权限

|  |
| --- |
|  |
| 问题一：哨兵无法远程访问，需要打开哨兵配置的保护模式  问题二：连接上哨兵之后，无法获取redis连接，需要将哨兵配置中的127.0.0.1也要修改为实际的服务器地址，否则客户端从哨兵获取的redis主机地址也会是127.0.0.1 |