Redis高可用

**1.Redis分布式集群的结构**

图示

描述已自动生成

实现集群节点的高可用,支持数据的备份功能是高可用的基础;

**2.redis的主从复制**

redis支持数据的主从复制,支持多级主从,一主多从的复制结构,当结构过于复杂时,数据同步的效率就降低了.企业经验,最多2级主从（父、子、孙）,最多一个主6个从节点;

命令和配置文件 **slaveof**

**2.1配置搭建一个redis的一主二从关系**

* 准备3个节点6382,6383,6384

[root@localhost redis-3.2.11]# pwd

/home/software/redis-3.2.11

[root@localhost redis-3.2.11]# cp redis.conf redis6382.conf

[root@localhost redis-3.2.11]# cp redis.conf redis6383.conf

[root@localhost redis-3.2.11]# cp redis.conf redis6384.conf

* 挨个修改端口

[root@localhost redis-3.2.11]# vim redis6382.conf

**输入下面替换命令，保存退出**

:%s/**6379**/6382/g

[root@localhost redis-3.2.11]# vim redis6383.conf

:%s/6379/6383/g

[root@localhost redis-3.2.11]# vim redis6384.conf

:%s/6379/6384/g



文本

低可信度描述已自动生成

* 挨个启动3个节点

[root@localhost redis-3.2.11]# redis-server redis6382.conf

[root@localhost redis-3.2.11]# redis-server redis6383.conf

[root@localhost redis-3.2.11]# redis-server redis6384.conf

* 主从关系的创建

6382为主节点,6383,6384为两个从节点

* 观察节点状态的命令

启动三个客户端分别访问三个节点

图示

低可信度描述已自动生成

127.0.0.1:6382> info replication

# Replication

role:**master**

connected\_slaves:0

master\_repl\_offset:0

repl\_backlog\_active:0

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:0

repl\_backlog\_histlen:0

127.0.0.1:6382>

默认情况下，所有的节点都是master

* 通过修改配置文件做定义主从

**图形用户界面, 文本, 应用程序

描述已自动生成**

* 通过命令挂接主节点

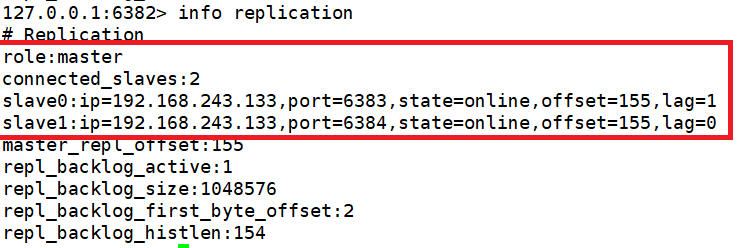
从节点>**slaveof** masterip masterport

127.0.0.1:6383> slaveof 192.168.243.133 6382

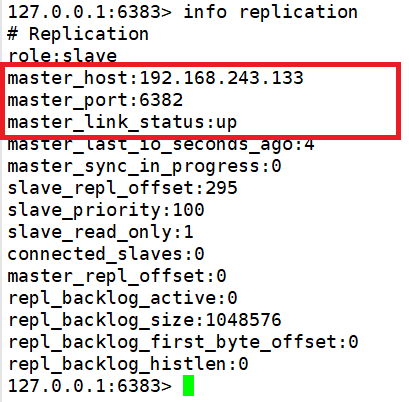
127.0.0.1:6384> slaveof 192.168.243.133 6382

* 查看挂接状态

主节点显示从节点信息



从节点显示主节点.主从关系各种数据状态



**2.2主从关系测试**

* 数据同步

主节点6382写数据,set

6382操作

文本

描述已自动生成

观察6383/6384数据

文本

描述已自动生成

文本

描述已自动生成

* 从节点写入数据

图片包含 徽标

描述已自动生成

默认状态(配置文件默认值),从节点只读。

如果需要在从节点写数据(多级主从关系)，可以打开配置文件的readonly的配置

**图形用户界面

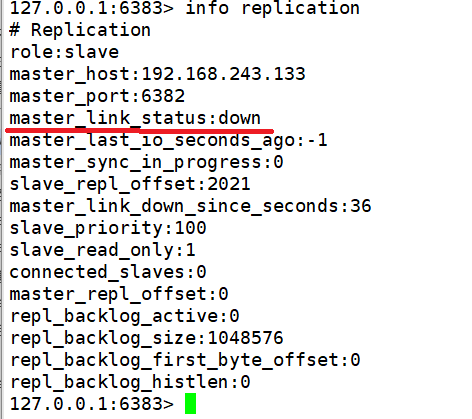
中度可信度描述已自动生成**

* 高可用替换

将6382干掉.观察6383和6384能否有一个自动将slave角色替换成master.

手机屏幕截图

描述已自动生成



在单纯的主从结构下.主节点宕机,从节点什么都不做;**主从结构只是做了数据的备份**

**2.哨兵集群模式(sentinel)**

**2.1介绍和原理**

redis-cluster(Redis3.0)出现之前,绝大部分的redis的使用是围绕哨兵集群展开的;利用单独的进程(特殊redis-server),开启对主从结构的监听;监听主节点,先从主节点调用命令info replication,拿到主从结构的所有信息,保存在内存中,后续每1秒钟发起一次心跳检测(rpc远程通信协议),一旦发现主节点宕机,哨兵集群发起投票选举,最终过半的票数确定事件的结果;

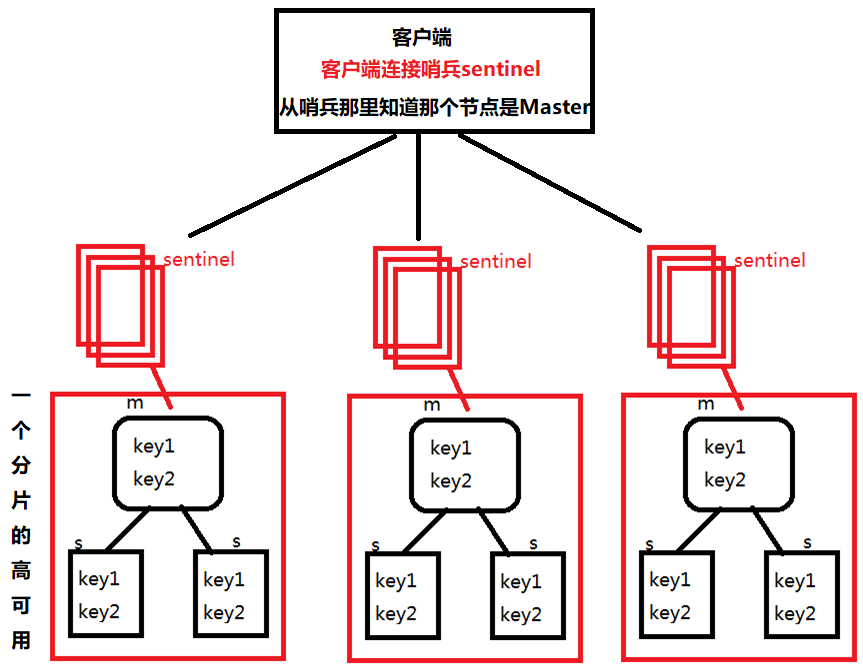
**2.2哨兵集群结构**

**图片包含 游戏机, 钟表

描述已自动生成**

一个哨兵集群,只能维护一个数据分片的高可用结构,不能完成分布式的结构;

分布式结构如下:



客户端连接的是哨兵，从哨兵那里知道那个节点正在提供服务。

**2.3哨兵集群的投票机制**

为什么要投票?为什么过半?

哨兵监听的主从关系.是状态 ,每一次角色的变化,都由哨兵来监控--对所有的主从结构的决策都**需要投票**.

哨兵节点需要连接主从结构(涉及到网络通信),只有一个哨兵节点的管理结构,不可信，只有一个哨兵节点，当哨兵连接主从结构的网络异常时，会误判为主从结构的节点出问题。

所以必须引入哨兵集群，管理主从结构.毕竟超过一半的节点都误判了的几率极小;

* 集群容忍度(宕机容忍度)

哨兵集群容忍度:可以允许宕机的个数,就是集群的容忍度;

5个哨兵集群节点:为了达到过半选举至少3个节点;容忍度是2

6个哨兵集群节点:为了过半至少剩余4个节点,容忍度是2

2n-1个集群的容忍度和2n是一样的;一般过半机制的集群,选择都是奇数个;

**2.4搭建哨兵集群监听主从结构**

搭建3个哨兵节点,管理6382,6383/6384的主从结构

* 修改哨兵的模板配置文件(在根目录sentinel.conf)

配置三个哨兵的端口**26379 26380 26381**

文本

描述已自动生成

**P15** bind的ip地址绑定--**保持注释**

**图形用户界面

低可信度描述已自动生成**

**P17** 释放注释掉的protected-mode

**图片包含 应用程序

描述已自动生成**

**P21** 保持模板文件中26379默认端口号

**图片包含 图示

描述已自动生成**

**P69** 监听主节点的配置核心内容

图形用户界面

低可信度描述已自动生成

sentinel monitor 配置命令,表示开始进行对\*\*的监听工作

mymaster 自定义的名称,表示当前哨兵监听的一个主从结构的代号,也是其他哨兵判断是否是同一个集群的依据

host 主节点的ip地址

port 主节点的端口号

最后面的2表示主观**下限票数**, 当集群哨兵不断宕机之后,最少剩下的节点数量

* 拷贝三份

图片包含 形状

描述已自动生成

* 修改sentinel02.conf和sentinel03.conf,端口号修改为26380 26381

文本

描述已自动生成

:%s/26379/26381/g

* 启动三个哨兵集群(注意要先启动主从结构中的三个节点：6382,6382.6384)

每个哨兵占用一个窗口，为了看日志信息

[root@localhost redis-3.2.11]# redis-sentinel sentinel01.conf

[root@localhost redis-3.2.11]# redis-sentinel sentinel02.conf

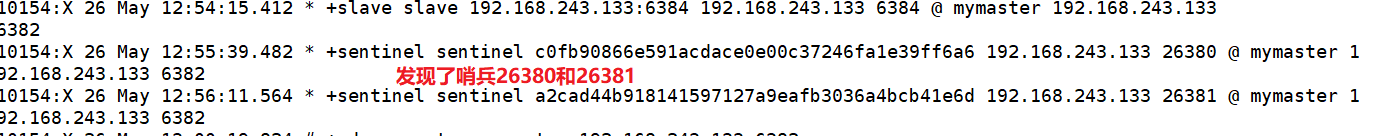
[root@localhost redis-3.2.11]# redis-sentinel sentinel03.conf

每个哨兵占用一个窗口，为了看日志信息

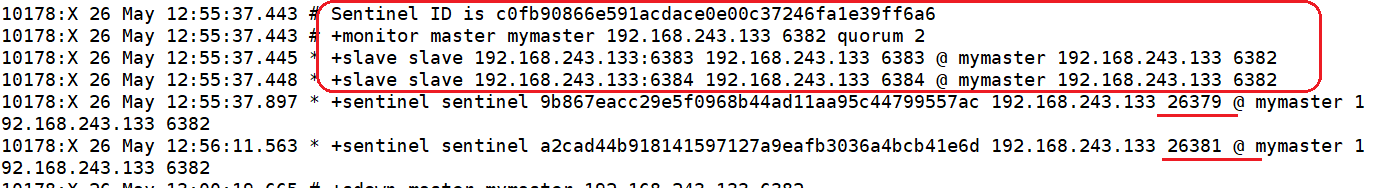
启动后查看日志：



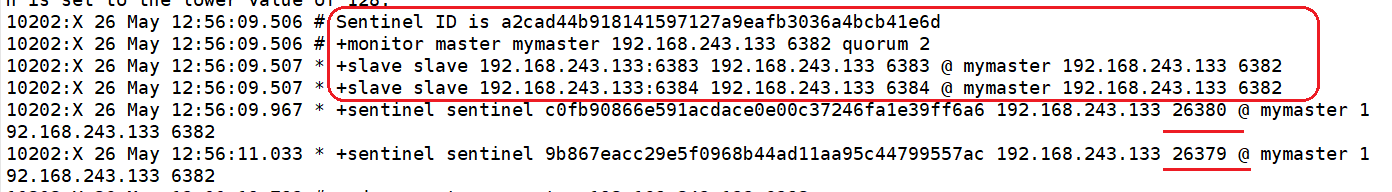
后面是互相发现



26380哨兵的日志：



26381哨兵的日志：

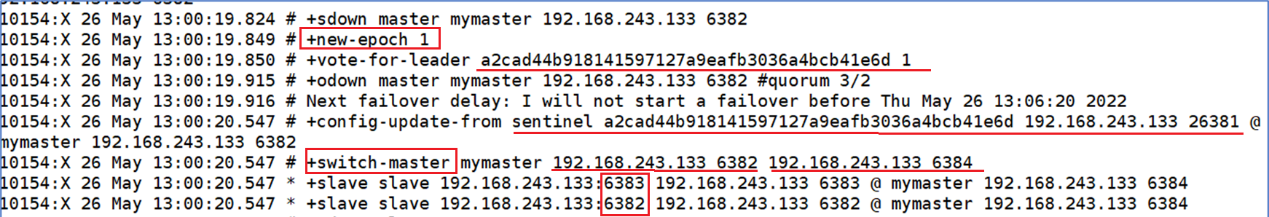


如果Down掉其中一个哨兵节点。剩下两个也够投票。

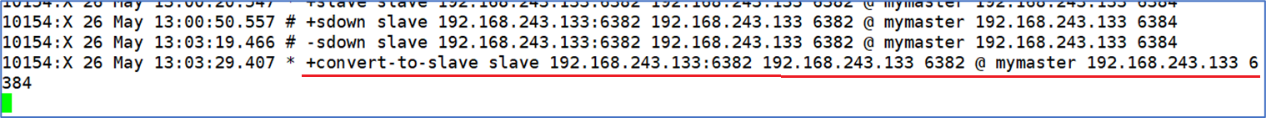
**2.5对哨兵集群高可用测试**

* 将6382 kill掉,观察日志内容(哨兵日志)

哨兵发现宕机.开始投票,通过,确定宕机. 投票选举新的主节点;



主节点一旦恢复,重新添加到集群以从节点角色继续提供服务.



查看哨兵日志文件，可查看到哨兵启动后，其监听过程中出现的各种各样的事件。

[root@localhost redis-3.2.11]# vim sentinel01.conf



sentinel current-epoch 的作用：当 slave A 发现其所属的 master 下线时，就会试图发起故障转移流程。首先就是增加 current-epoch的值，这个增加后的 current-epoch的值是所有集群节点中最大的。然后slave A 向所有节点发起拉票请求，请求其他 节点投票给自己，使自己能成为新的 master。其他节点收到包后，发现发送者的 current-epoch比自己的 current-epoch 大，就会更新自己的 current-epoch，并在尚未投票的情况下，投票给 slave A，表示同意使其成为新的 master。

**2.6重启哨兵集群**

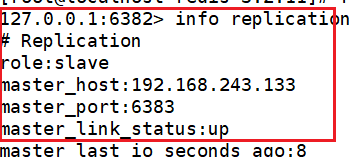
* **确定主从结构**

**重启哨兵集群时，必须先确定主从结构正确。**

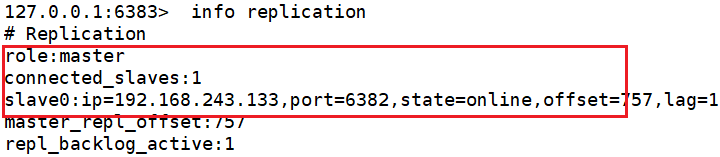
由于在此之前，我kill掉了6384，6383变成了master,然后重启虚拟机，所以重启redis哨兵集群时，应该按下面步骤完成

1）分别启动6382，6383，6384节点，打开三个窗口，进行客户端连接，分别通过info replication查看信息，观察结构是否正确。要看到正确的主从结构信息，否则要重新修改配置文件

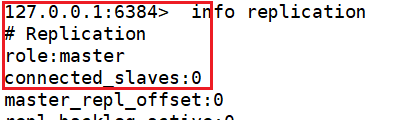
查看6382，发现其挂接在6383上。6383是Master.



查看6383，发现6383是Master.,其上面只挂接在6382。

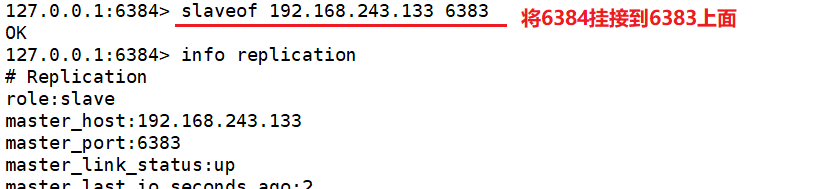


查看6384，发现其自己是Master.

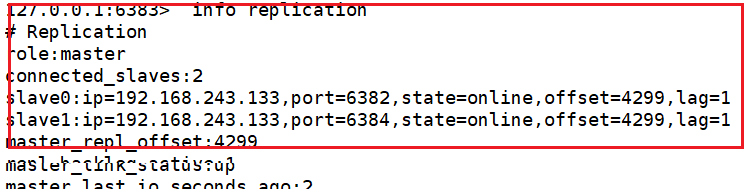


通过查看信息我们发现6383是Master,6382挂接在6383上，6384是Master,没有节点挂接在它上面。发现三个节点的主从关系不正常，需要修改

将6384挂节到6383上面，使主从关系正常。



再次查看6383节点：



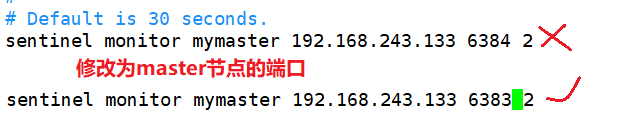
发现主从结构正常。

* **哨兵进程**

初始化配置文件，分别查看三个哨兵节点的配置文件。

* 确定配置文件的sentinel monitor mymaster 主节点ip 主节点port 2

[root@localhost redis-3.2.11]# vim sentinel01.conf



[root@localhost redis-3.2.11]# vim sentinel02.conf

[root@localhost redis-3.2.11]# vim sentinel03.conf

确定sentinel02.conf和sentinel03.conf中的

sentinel monitor mymaster 主节点ip 主节点port 2 正确

* 尾部的日志给删除

删除sentinel01.conf、sentinel02.conf和sentinel03.conf尾部的日志。

文本

描述已自动生成

* 重新启动sentinel

重启三个sentinel节点。

[root@localhost redis-3.2.11]# redis-sentinel sentinel01.conf

[root@localhost redis-3.2.11]# redis-sentinel sentinel02.conf

[root@localhost redis-3.2.11]# redis-sentinel sentinel03.conf

**2.7 jedis连接操作sentinel集群的api**

开启26379、26380、26381端口。

文本

描述已自动生成

连接哨兵集群操作主从,高可用结构

**在运行下面测试案例之前，确保三个主从节点和三个哨兵节点都启动。**

图形用户界面, 文本, 应用程序

描述已自动生成

文本

中度可信度描述已自动生成

**2.8将哨兵集群的高可用结构整合springboot**

* properties中准备使用的参数

哨兵的节点们nodes

连接池属性们 maxTotal idle

主从结构的代号

* @Configuration编辑配置类

@ConfigurationProperties读取properties数据到私有属性

* 构造一个初始化JedisSentinelPool对象的初始化方法,将读过来的数据在初始化方法中使用返回一个JedisSentinelPool对象,只需要在代码中注入这个对象使用即可。