# Redis高级

## Redis中持久化策略

### Redis持久化规则

说明:根据redis的配置文件中所配置的持久化策略,定期持久化.将redis内存中的数据最终写入到**持久化文件**中.当redis意外关闭内存数据清空了.当redis重新启动时,根据配置文件中持久化文件的路径/名称,读取持久化文件.

从持久化文件中恢复内存数据.

## RDB模式

说明:

1. RDB模式是redis中**默认**的持久化策略.该模式能够定期(有时间间隔)将内存中的数据持久化到XXX.rdb文件中.
2. 该模式的效率是最高的.相当于对内存做**快照**处理.
3. 同时数据加密保存.
4. 如果允许少量的数据丢失则首选RDB模式

### RDB模式持久化配置

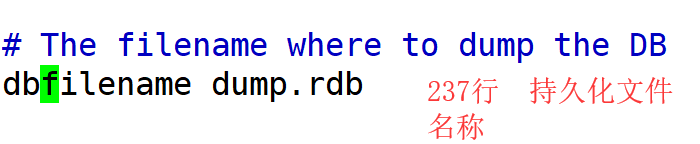
1. 持久化时间间隔

save 900 1 900秒内执行一次set操作 则持久化1次

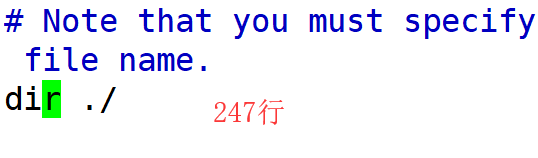
save 300 10 300秒内执行10次set操作,则持久化1次

save 60 10000 60秒内执行10000次set操作,则持久化1次

1. 持久化文件名称



1. 持久化文件存储路径



如果需要修改持久化文件保存位置,则修改该文件 /绝对路径.

## AOF模式

说明:

1. AOF模式通过持久化文件记录了用户的全部的操作过程.该模式可以实现数据的实时备份.
2. AOF模式中持久化文件会很庞大.不易于解析.
3. AOF文件没有加密处理.

特点:

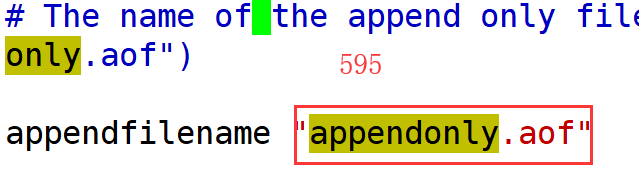
如果对数据完整性有特殊要求,则采用AOF模式.

### AOF模式配置

1. 开启AOF模式 如果改为yes后,redis持久化策略采用AOF模式.



1. AOF持久化文件名称



1. 持久化策略

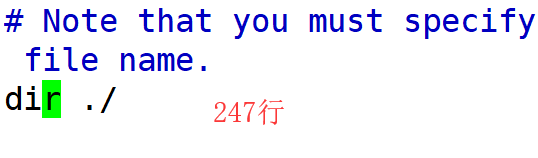
appendfsync always 该配置表示如果用户执行set操作,就持久化1次

appendfsync everysec 每秒持久化一次

appendfsync no 持久化化

1. 文件存储

通过dir文件,执行持久化文件位置.



## Redis中内存策略

### 需求描述

Redis中的数据都保存在内存中.如果内存中一直添加数据,则可能会造成内存填满,内存溢出的现象.需要控制redis的内存大小.

### LRU算法

[内存管理](https://baike.baidu.com/item/%E5%86%85%E5%AD%98%E7%AE%A1%E7%90%86/5633616)的一种页面置换算法，对于在内存中但又不用的[数据块](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%9D%97/107672)（内存块）叫做LRU，操作系统会根据哪些数据属于LRU而将其移出内存而腾出空间来加载另外的数据。

### 内存策略介绍

volatile-lru 在已经设定了超时时间的数据中,选择最少使用的数据进行删除.

allkeys-lru 选择全部值,采用LRU算法进行删除.

volatile-random 在设定了超时时间的数据中随机删除.

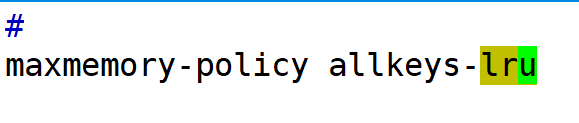
allkeys-random 在所以的key中随机删除.

volatile-ttl 在设定了超时时间的数据中,按照超时时间倒叙排列,删除马上过期的数据.

Noeviction(默认策略) 如果选择该模式,在不删除数据,则直接报错给用户.

### 内存策略修改

560行



## Redis分片技术

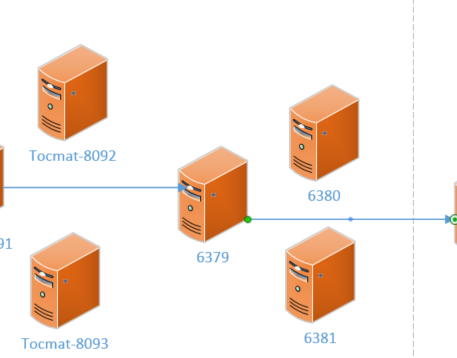
### 业务需求

1. 如果需要动态的扩展内存,单个redis节点上有上限(10M默认-512M-1GB)
2. 如果将数据保存到一个redis节点中,可能会出现丢数据的风险.所以尽可能分开保存.

### Redis分片机制介绍

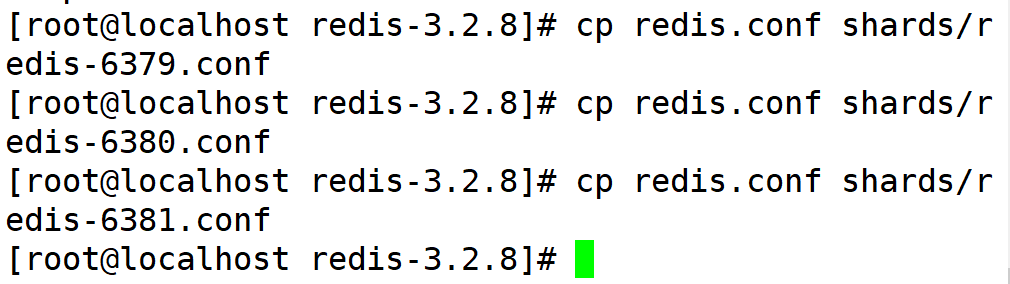
特点:

1. 多台redis节点将内存扩大N倍
2. 多台redis共同唯一所有的数据.每个节点中所保存的数据都不一样的.

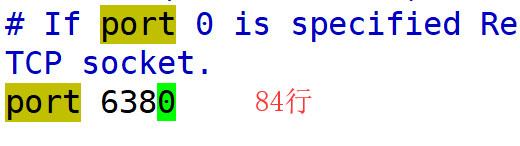


### Redis分片配置步骤

1. 复制配置文件



1. 修改端口号



### 启动多台redis

redis-server redis-6379.conf

[root@localhost shards]# redis-server redis-6380.conf

[root@localhost shards]# redis-server redis-6381.conf

[root@localhost shards]# ps -ef |grep redis

root 9789 1 0 19:41 ? 00:00:00 redis-server \*:6379

root 9794 1 0 19:41 ? 00:00:00 redis-server \*:6380

root 9798 1 0 19:41 ? 00:00:00 redis-server \*:6381

root 9804 7057 0 19:41 pts/2 00:00:00 grep redis

### Redis分片入门案例

@Test

**public** **void** test01(){

List<JedisShardInfo> shards = **new** ArrayList<>();

shards.add(**new** JedisShardInfo("192.168.126.166", 6379));

shards.add(**new** JedisShardInfo("192.168.126.166", 6380));

shards.add(**new** JedisShardInfo("192.168.126.166", 6381));

ShardedJedis shardedJedis =

**new** ShardedJedis(shards);

shardedJedis.set("1807","学习redis分片");

System.***out***.println("获取数据:"+shardedJedis.get("1807"));

}

### 链接池操作

@Test

**public** **void** test01(){

List<JedisShardInfo> shards = **new** ArrayList<>();

shards.add(**new** JedisShardInfo("192.168.126.166", 6379));

shards.add(**new** JedisShardInfo("192.168.126.166", 6380));

shards.add(**new** JedisShardInfo("192.168.126.166", 6381));

JedisPoolConfig config = **new** JedisPoolConfig();

config.setMaxTotal(500);

config.setMaxIdle(20);//设定最大空闲链接

//实现链接池的操作

ShardedJedisPool pool =

**new** ShardedJedisPool(config, shards);

/\*ShardedJedis shardedJedis =

new ShardedJedis(shards);\*/

ShardedJedis shardedJedis =

pool.getResource();

shardedJedis.set("1807","学习redis分片");

System.***out***.println("获取数据:"+shardedJedis.get("1807"));

//shardedJedis.close(); //将链接直接关闭.

pool.returnResource(shardedJedis);//将链接还回池中.

}

## Spring整合redis分片

### 编辑配置文件

<!--spring实现分片的配置 -->

<bean id=*"poolConfig"* class=*"redis.clients.jedis.JedisPoolConfig"*>

<property name=*"maxTotal"* value=*"1000"*/>

</bean>

<bean id=*"info1"* class=*"redis.clients.jedis.JedisShardInfo"*>

<constructor-arg name=*"host"* value=*"192.168.126.166"*/>

<constructor-arg name=*"port"* value=*"6379"*/>

</bean>

<bean id=*"info2"* class=*"redis.clients.jedis.JedisShardInfo"*>

<constructor-arg name=*"host"* value=*"192.168.126.166"*/>

<constructor-arg name=*"port"* value=*"6380"*/>

</bean>

<bean id=*"info3"* class=*"redis.clients.jedis.JedisShardInfo"*>

<constructor-arg name=*"host"* value=*"192.168.126.166"*/>

<constructor-arg name=*"port"* value=*"6381"*/>

</bean>

<bean id=*"jedisPool"* class=*"redis.clients.jedis.ShardedJedisPool"*>

<constructor-arg name=*"poolConfig"* ref=*"poolConfig"*/>

<constructor-arg name=*"shards"*>

<list>

<ref bean=*"info1"*/>

<ref bean=*"info2"*/>

<ref bean=*"info3"*/>

</list>

</constructor-arg>

</bean>

### 编辑工具类

@Service

**public** **class** RedisService {

//有的工程需要，有的工程不需要。设置required=false，有就注入，没有就不注入。

@Autowired(required=**false**)

**private** ShardedJedisPool jedisPool;

**public** String get(String key){

ShardedJedis jedis = jedisPool.getResource();

String result = jedis.get(key);

jedisPool.returnResource(jedis);

**return** result;

}

**public** **void** set(String key,String value){

ShardedJedis jedis = jedisPool.getResource();

jedis.set(key,value);

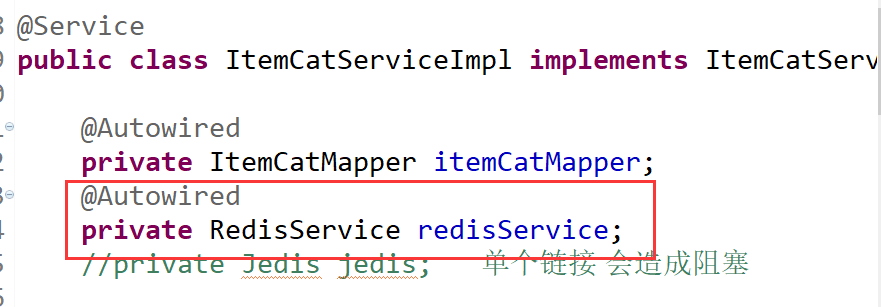
jedisPool.returnResource(jedis);

}

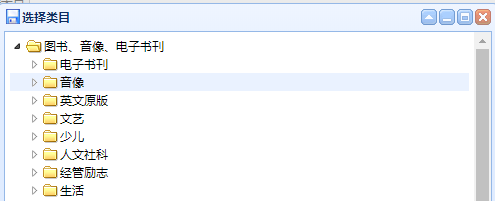
}

### 修改业务类

注入工具类对象



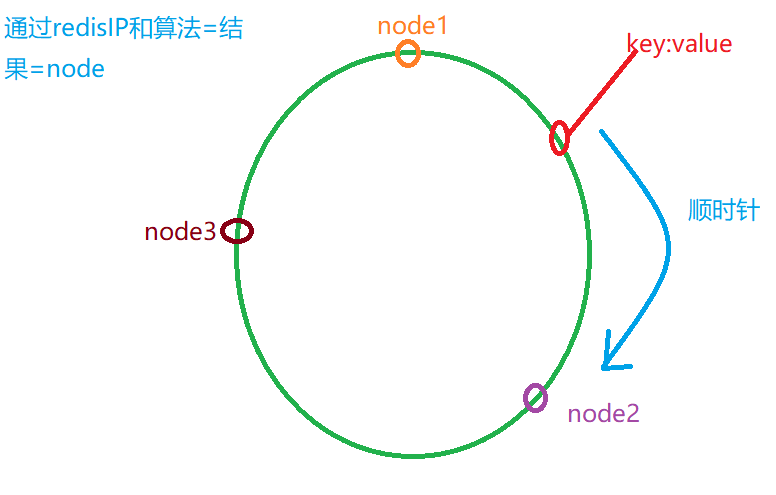
之后修改方法.展现结果



## Hash一致性算法

### Redis数据存储策略

2^32=21亿

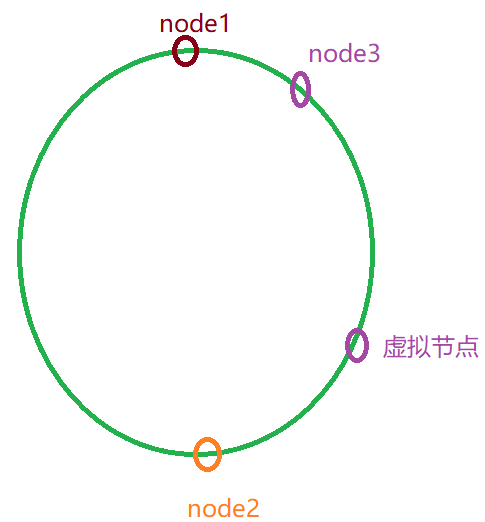


### 均衡性

说明:采用均衡性的特点让数据尽可能均匀.如果分配不均则采用虚拟节点争抢数据.最终达到相对平衡.

问题:Hash一致算法中存在的问题,如果计算的结果node节点分布不均匀.则可能会出现数据超过最大内存的现象.

解决:尽可能的让节点均匀分配数据.



### 单调性

说明:当节点数量**增加**时,数据会重新进行挂载,但是尽可能保证原有的数据不变.

特列:

如果使用分片的过程中,分片的节点突然宕机.会导致映射的内存缺失.直接导致整个分片将不能正常使用.

### 分散性

特点:由于**分布式的原因**.有些系统**不能够看到全部的内存空间**.会导致**一个key出现多个位置**的现象(不好的)

### 负载

说明:负载是分散性的另一种讨论,可能会出现**一个位置有多个key**的现象.

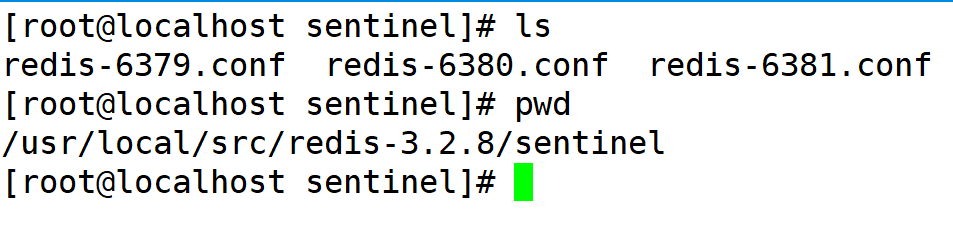
课下自行学习.

## Redis主从配置

### 实现步骤

如果redis需要实现高可用.前提条件就是实现数据的同步.

1. 拷贝配置shards文件夹 改名为sentinel



1. 主从搭建策略

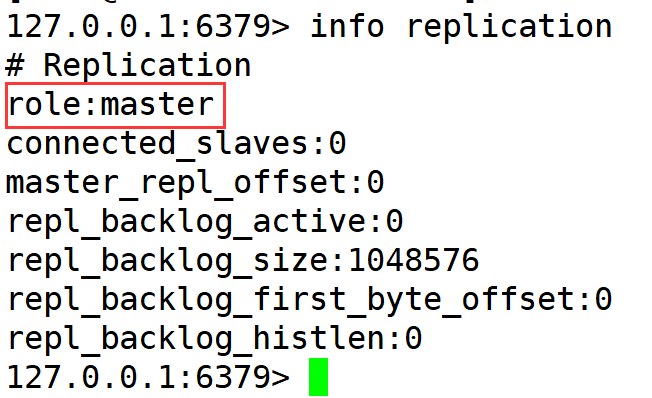
6379主机.

6380从机

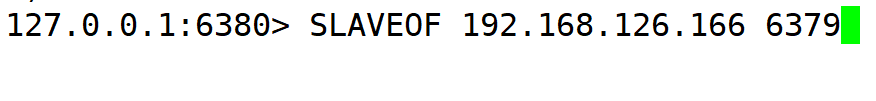
6381从机

1. 状态信息检查

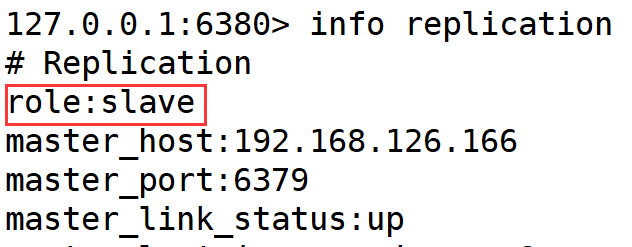
info replication



1. 主从挂载



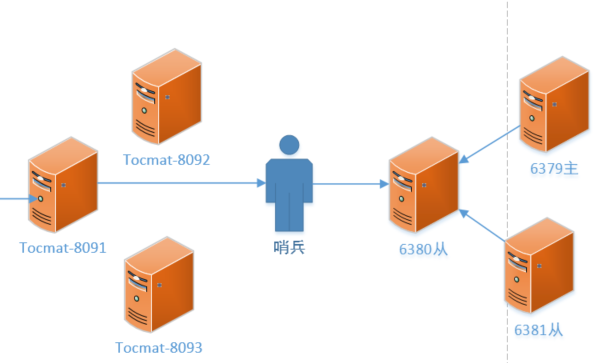
1. 检查主从挂载状态



## 哨兵机制

### 哨兵的作用

通过哨兵机制,**可以实现redis的高可用**,如果主节点宕机后,由哨兵通过选举机制,选择一台从机.同时修改其他机器的配置文件,配置新的主机.

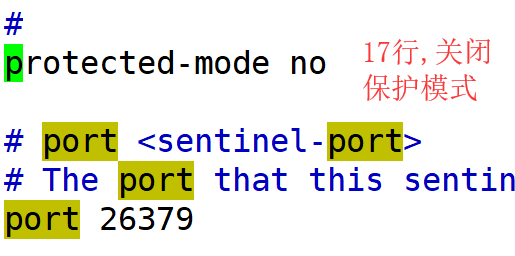


## 编辑哨兵配置文件

### 准备工作

1. 将哨兵的配置文件导入sentinel文件夹中

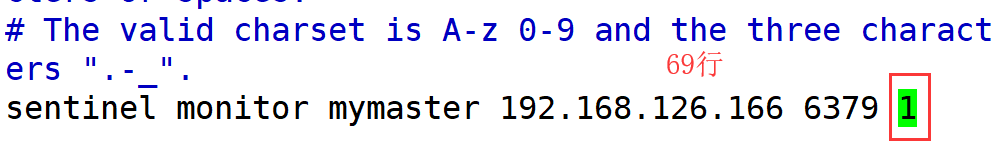
### 关闭保护模式



### 修改哨兵的监听

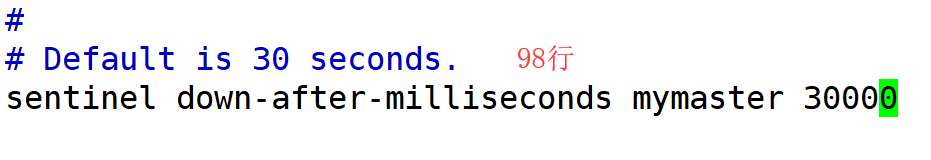
sentinel monitor mymaster 192.168.126.166 6379 2

2:表示2票同意该选举生效.

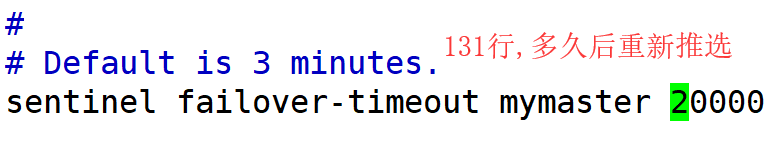


### 修改选举时间

当主机宕机多久后开始选举

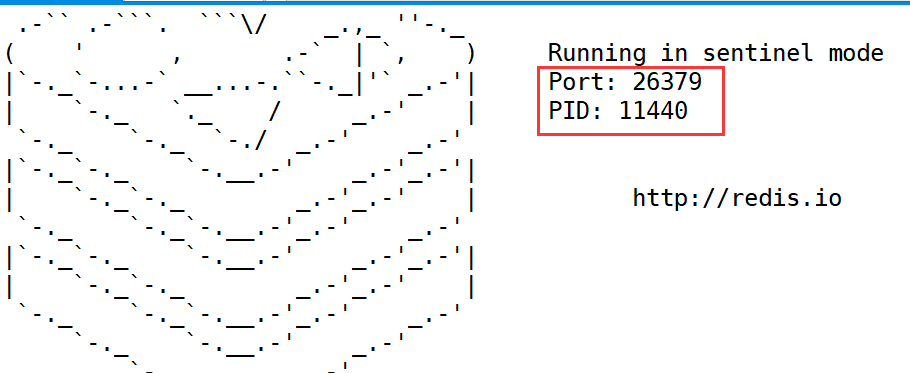


### 修改推选失败超时时间



### 哨兵测试

启动命令: redis-sentinel sentinel.conf



测试步骤:

1. 关闭redis主机,10秒后检查redis哨兵是否正常选举.
2. 将主机重新启动,检查是否降级为从机.

### 哨兵的入门案例

@Test

**public** **void** test01(){

/\*\*

\* masterName:主机的变量名称

\* sentinels:表示哨兵的链接

\* new HostAndPort("192.168.126.166",26379).toString()

\*

\* 转化后的结果

\* 192.168.126.166:26379

\*/

Set<String> sentinels = **new** HashSet<>();

sentinels.add("192.168.126.166:26379");

JedisSentinelPool sentinelPool =

**new** JedisSentinelPool("mymaster", sentinels);

Jedis jedis = sentinelPool.getResource();

jedis.set("bb","哨兵操作");

System.***out***.println(jedis.get("bb"));

}