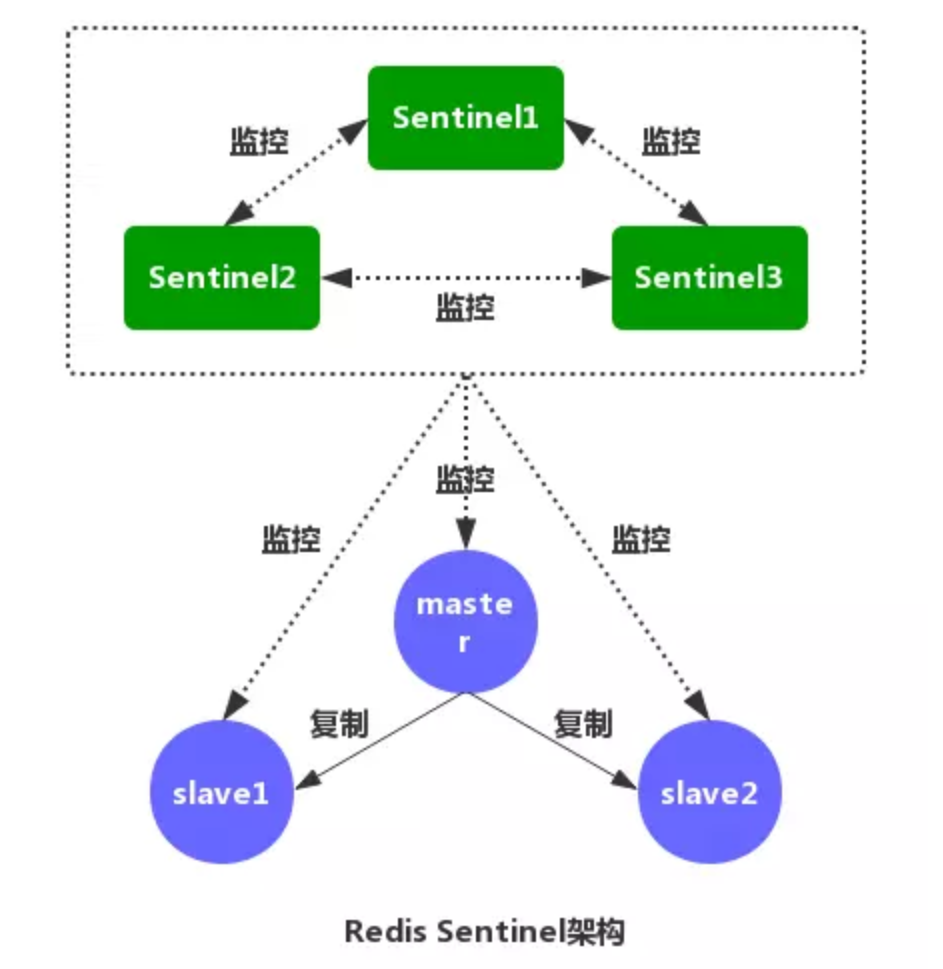
# 一．规划

部署一个master，两个slave，三个哨兵的集群，节点规划如下：

|  |  |  |
| --- | --- | --- |
| 主 | 192.168.61.131 | 6379 |
| 从 | 192.168.61.132 | 6379 |
| 从 | 192.168.61.133 | 6379 |
| 哨兵（sentinel） | 192.168.61.131 | 26379 |
| 哨兵（sentinel） | 192.168.61.132 | 26379 |
| 哨兵（sentinel） | 192.168.61.133 | 26379 |



该图包括Redis的主从复制模式并利用了Sentinel的高可用架构；

# 二．Redis安装

1.下载redis离线包



2.解压后进入目录进行编译

解压目录如下：tar –zxvf redis-5.0.5.tar.gz



编译：

make

安装：

make install PREFIX=/usr/local/redis

然后在/usr/local/redis/目录下就会生成bin目录，目录中文件如下：



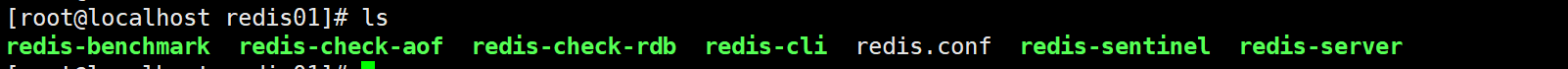
# 三．Redis-sentinel 集群搭建

## Master配置

1.复制/usr/local/redis/bin目录下的文件到自己创建的目录；比如：其中的redis01目录；

2.将解压的redis包中的redis.conf复制到redis01目录中；





3.redis.conf配置文件的修改

|  |
| --- |
| #绑定本机IP  bind 192.168.61.131  #端口默认就行  port 6379  #客户端超时时间  timeout 30  #开启守护进程  daemonize yes  pidfile /var/run/redis\_6379.pid  #日志文件存放位置，提前创建目录  logfile "/var/log/redis/redis.log"  databases 16  always-show-logo yes  save 900 1  save 300 10  #SNAPSHOTTING目录提前创建  dir /data/redis  #使用AOF持久化  appendonly yes  appendfilename "appendonly.aof"  #实时同步  appendfsync always |

## Slave配置

1.复制/usr/local/redis/bin目录下的文件到某一目录；比如：其中的redis01目录；

2.将解压的redis包中的redis.conf复制到redis01目录中；

可以直接使用scp命令将master的redis.conf文件上传到slave服务器中，有两个参数需要修改下：如下面配置中标红的地方

两个slave的配置是一样的，就是bind ip不一样；

|  |
| --- |
| #绑定本机IP  bind 192.168.61.132  #端口默认就行  port 6379  #客户端超时时间  timeout 30  #开启守护进程  daemonize yes  pidfile /var/run/redis\_6379.pid  #日志文件存放位置，提前创建目录  logfile "/var/log/redis/redis.log"  databases 16  always-show-logo yes  save 900 1  save 300 10  #SNAPSHOTTING目录提前创建  dir /data/redis  #使用AOF持久化  appendonly yes  appendfilename "appendonly.aof"  #实时同步  appendfsync always  #指定主服务IP 端口  replicaof 192.168.61.131 6379  #如果slave 无法与master 同步，设置成slave不可读，方便监控脚本发现问题。  replica-serve-stale-data no |

## Sentinel配置

1.复制/usr/local/redis/bin目录下的文件到某一目录；

2.将解压的redis包中的sentinel.conf复制到目录中；

配置文件如下，然后三个服务器上分别放置一份；

|  |
| --- |
| port 26379  #开启守护进程  daemonize yes  pidfile /var/run/redis-sentinel.pid  #指定日志文件存放位置  logfile "/var/log/redis/sentinel/"  dir /tmp  #指定主的IP 端口 “2”代表有两个哨兵进程发现主服务器宕机便选取一个从服务器为主  sentinel monitor mymaster 192.168.61.131 6379 2  sentinel down-after-milliseconds mymaster 30000  sentinel parallel-syncs mymaster 1  sentinel failover-timeout mymaster 180000  sentinel deny-scripts-reconfig yes |

依次启动主，两从，三哨兵；

启动主从命令：./redis-server redis.conf

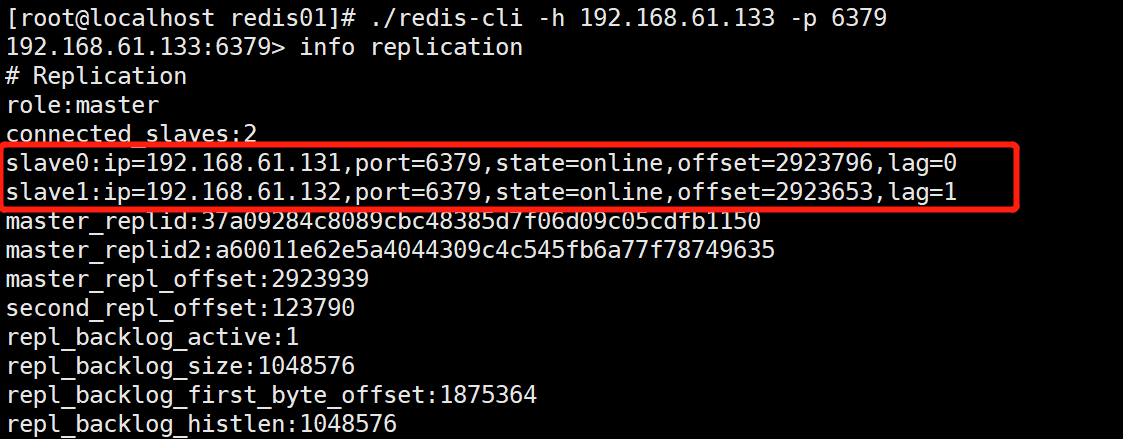
启动sentinel命令：./redis-server sentinel.conf --sentinel

或者使用 ./redis-sentinel sentinel.conf （推荐）

进入master，查看集群信息：

./redis-cli -h 192.168.61.133 -p 6379

命令：Info replication



## 故障模拟

可以手动kill掉master，然后从slave进入查看，会出现新的master；

然后将刚刚kill掉的redis再重新添加进来，会成为slave节点；

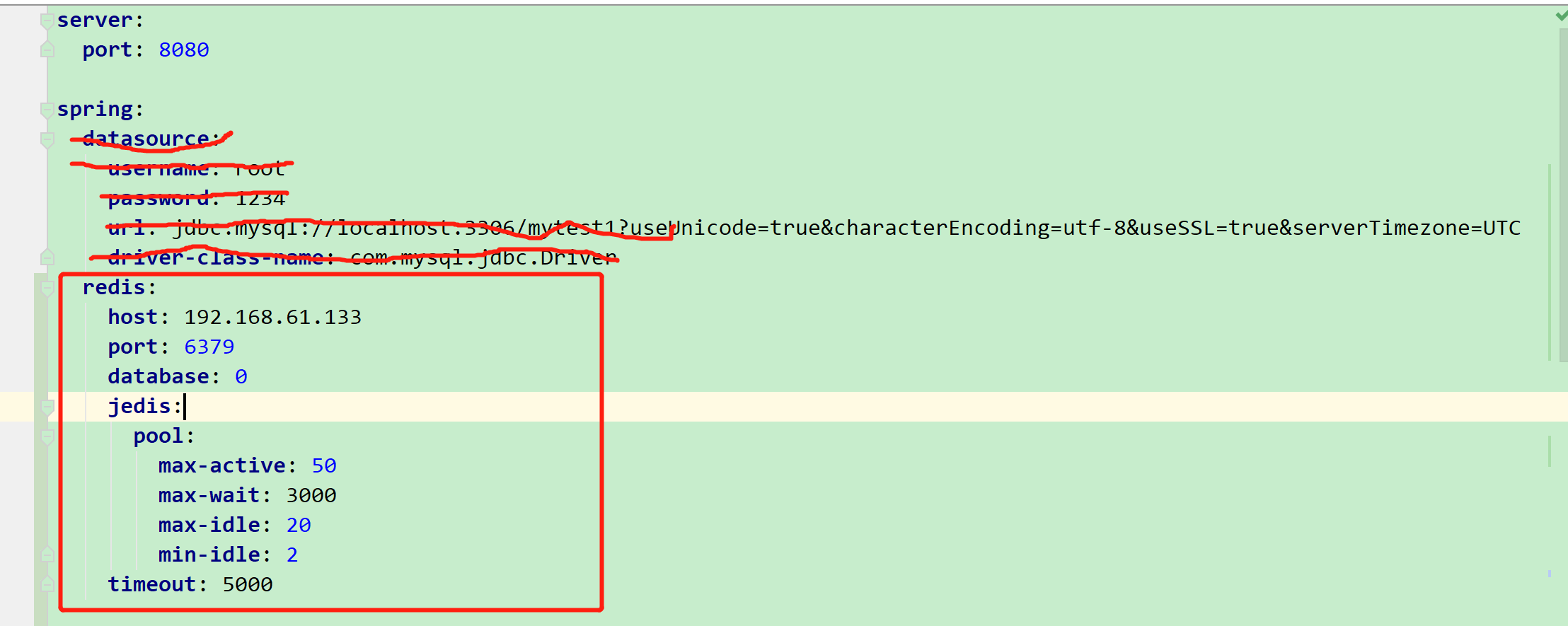
# 四．Springboot使用Redis

添加依赖：

*<!-- redis依赖包 -->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-redis</**artifactId**>  
</**dependency**>

单机配置：

application.yaml配置:其中方框内的配置，这里的host需要配置master，不能配slave的；



集群配置：



以下是操作redis的工具类：

@Component  
**public class** RedisUtils {  
   
 @Autowired  
 **private** RedisTemplate<String, String> **redisTemplate**;  
   
 */\*\*  
 \* 读取缓存  
 \*   
 \** ***@param key*** *\** ***@return*** *\*/* **public** String get(**final** String key) {  
 **return redisTemplate**.opsForValue().get(key);  
 }  
   
 */\*\*  
 \* 写入缓存  
 \*/* **public boolean** set(**final** String key, String value) {  
 **boolean** result = **false**;  
 **try** {  
 **redisTemplate**.opsForValue().set(key, value);  
 result = **true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** result;  
 }  
   
 */\*\*  
 \* 更新缓存  
 \*/* **public boolean** getAndSet(**final** String key, String value) {  
 **boolean** result = **false**;  
 **try** {  
 **redisTemplate**.opsForValue().getAndSet(key, value);  
 result = **true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** result;  
 }  
   
 */\*\*  
 \* 删除缓存  
 \*/* **public boolean** delete(**final** String key) {  
 **boolean** result = **false**;  
 **try** {  
 **redisTemplate**.delete(key);  
 result = **true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** result;  
 }  
}

# 五．K8s中Redis集群的搭建

使用开源chart搭建：

<https://github.com/helm/charts/tree/master/stable/redis-ha>

kubectl exec -it existing-llama-redis-ha-server-0 sh -n default

redis-cli -h existing-llama-redis-ha.default.svc.cluster.local