一、单节点实例
单节点实例还是比较简单的，平时做个测试，写个小程序如果需要用到缓存的话，启动一个 Redis 还是很轻松的，做为一个 key/value 数据库也是可以胜任的

二、主从模式（master/slaver）
redis 主从模式配置
主从模式：redis 的主从模式，使用异步复制，slave 节点异步从 master 节点复制数据，master 节点提供读写服务，slave 节点只提供读服务（这个是默认配置，可以通过修改配置文件 slave-read-only 控制）。master 节点可以有多个从节点。配置一个 slave 节点只需要在 redis.conf 文件中指定 slaveof master-ip master-port 即可。
从节点开启主从复制，有 3 种方式：

配置文件
在从服务器的配置文件中加入：slaveof<masterip><masterport>
启动命令
redis-server 启动命令后加入：slaveof<masterip><masterport>
客户端命令
Redis 服务器启动后直接通过客户端执行命令：slaveof<masterip><masterport>，则该 Redis 实例成为从节点。
上述 3 种方式是等效的，下面以客户端命令的方式为例，看一下当执行了 slaveof 后，Redis 主节点和从节点的变化。

本示例：一个 master 节点有两个 slave 节点

配置：
1，cd /usr/local/redis/redis-4.0.2

切换到当前 redis 安装路径
2， mkdir config

新建一个文件夹，存放 redis 的配置文件
3，在 config 下，新建三个配置文件，如下：

cd config

vi master-6739.conf

bind 0.0.0.0
port 6379
logfile "6379.log"
dbfilename "dump-6379.rdb"
daemonize yes
rdbcompression yes
vi slave-6380.conf

bind 0.0.0.0
port 6380
logfile "6380.log"
dbfilename "dump-6380.rdb"
daemonize yes
rdbcompression yes
slaveof 192.168.81.135 6379
vi slave-6381.conf

bind 0.0.0.0
port 6381
logfile "6381.log"
dbfilename "dump-6381.rdb"
daemonize yes
rdbcompression yes
slaveof 192.168.81.135 6379
master-6739.conf，为主节点配置文件，slave-6380.conf，slave-6381.conf 为从节点配置文件
在从节点的配置文件中使用：slaveof 指定 master 节点

4，启动三台 reids 服务

[root@localhost redis-4.0.2]# ./src/redis-server config/master-6379.conf
[root@localhost redis-4.0.2]# ./src/redis-server config/slave-6380.conf
[root@localhost redis-4.0.2]# ./src/redis-server config/slave-6381.conf
查看一下 redis 服务

测试主从模式：

a，先分别连上三台 Redis 服务，获取 key 为 name 的值，通过-p 指定连接那个端口的 redis 服务

[root@localhost redis-4.0.2]# ./src/redis-cli -p 6379
127.0.0.1:6379> get name
(nil)
[root@localhost redis-4.0.2]# ./src/redis-cli -p 6380
127.0.0.1:6380> get name
(nil)
[root@localhost redis-4.0.2]# ./src/redis-cli -p 6381
127.0.0.1:6381> get name
(nil)
#获取的值都为空
b，给 master 节点 set 一个 key

[root@localhost redis-4.0.2]# ./src/redis-cli -p 6379
127.0.0.1:6379> set name cmy
OK
127.0.0.1:6379> get name
"cmy"
c，slave 节点直接读取 key 为 name 的值

[root@localhost redis-4.0.2]# ./src/redis-cli -p 6380
127.0.0.1:6380> get name
"cmy"
[root@localhost redis-4.0.2]# ./src/redis-cli -p 6381
127.0.0.1:6381> get name
"cmy"
d，slave 节点只提供读服务，不能进行写入操作

127.0.0.1:6381> set age 23
(error) READONLY You can't write against a read only slave.
注意
使用主从模式时应注意 matser 节点的持久化操作，matser 节点在未使用持久化的情况详情下如果宕机，并自动重新拉起服务，从服务器会出现丢失数据的情况。

首先，禁止 matser 服务持久化

127.0.0.1:6379> CONFIG SET save ""
OK
在 master 节点 set 一个值

127.0.0.1:6379> set age 23
OK
slave 节点可以 get 到 age 的值

127.0.0.1:6380> get age
"23"
关掉 master 节点服务

127.0.0.1:6379> shutdown
not connected>
slave 节点此时仍可以 get 到 age 的值

127.0.0.1:6380> get age
"23"
重启 master 服务，此时获取不到 age 的值

[root@localhost redis-4.0.2]# ./src/redis-server config/master-6379.conf
[root@localhost redis-4.0.2]# ./src/redis-cli -p 6379
127.0.0.1:6379> get age
(nil)
slave 节点此时在获取 age 的值为空，数据丢失

[root@localhost redis-4.0.2]# ./src/redis-cli -p 6380
127.0.0.1:6380> get age
(nil)
数据丢失的原因：因为 master 服务挂了之后，重启服务后，slave 节点会与 master 节点进行一次完整的重同步操作，所以由于 master 节点没有持久化，就导致 slave 节点上的数据也会丢失掉。所以在配置了 Redis 的主从模式的时候，应该打开主服务器的持久化功能。
到这，redis 的主从模式就已经完成了

谈谈我认为主从模式的必要性：

主从模式的一个作用是备份数据，这样当一个节点损坏（指不可恢复的硬件损坏）时，数据因为有备份，可以方便恢复。
另一个作用是负载均衡，所有客户端都访问一个节点肯定会影响 Redis 工作效率，有了主从以后，查询操作就可以通过查询从节点来完成。

对主从模式必须的理解（结论已经验证过，可以自行验证）：

一个 Master 可以有多个 Slaves
默认配置下，master 节点可以进行读和写，slave 节点只能进行读操作，写操作被禁止
不要修改配置让 slave 节点支持写操作，没有意义，原因一，写入的数据不会被同步到其他节点；原因二，当 master 节点修改同一条数据后，slave 节点的数据会被覆盖掉
slave 节点挂了不影响其他 slave 节点的读和 master 节点的读和写，重新启动后会将数据从 master 节点同步过来
master 节点挂了以后，不影响 slave 节点的读，Redis 将不再提供写服务，master 节点启动后 Redis 将重新对外提供写服务。
master 节点挂了以后，不会 slave 节点重新选一个 master
对有密码的情况说明一下，当 master 节点设置密码时：
客户端访问 master 需要密码
启动 slave 需要密码，在配置中进行配置即可
客户端访问 slave 不需要密码

主从节点的缺点

主从模式的缺点其实从上面的描述中可以得出：
master 节点挂了以后，redis 就不能对外提供写服务了，因为剩下的 slave 不能成为 master
这个缺点影响是很大的，尤其是对生产环境来说，是一刻都不能停止服务的，所以一般的生产坏境是不会单单只有主从模式的。所以有了下面的 sentinel 模式。

三、sentinel 模式
Redis 哨兵模式，用现在流行的话可以说就是一个“哨兵机器人”，给“哨兵机器人”进行相应的配置之后，这个"机器人"可以 7*24 小时工作，它能能够自动帮助你做一些事情，如监控，提醒，自动处理故障等。

Redis-sentinel 简介

Redis-sentinel 是 Redis 的作者 antirez，因为 Redis 集群的被各大公司使用，每个公司要写自己的集群管理工具，于是 antirez 花了几个星期写出了 Redis-sentinel。

Redis 的 Sentinel 系统用于管理多个 Redis 服务器（instance），Redis 的 Sentinel 为 Redis 提供了高可用性。使用哨兵模式创建一个可以不用人为干预而应对各种故障的 Redis 部署。

该系统执行以下三个任务：

监控（Monitoring）：Sentinel 会不断地检查你的主服务器和从服务器是否允许正常。
提醒（Notification）：当被监控的某个 Redis 服务器出现问题时，Sentinel 可以通过 API 向管理员或者其他应用程序发送通知。
自动故障迁移（Automatic failover）：（1）当一个主服务器不能正常工作时，Sentinel 会开始一次自动故障迁移操作，他会将失效主服务器的其中一个从服务器升级为新的主服务器，并让失效主服务器的其他从服务器改为复制新的主服务器；（2）客户端试图连接失败的主服务器时，集群也会向客服端返回新主服务器的地址，是的集群可以使用新主服务器代替失效服务器。
sentinel 的分布式特性

Redis Sentinel 是一个分布式系统，你可以在一个架构中运行多个 Sentinel 进程（progress），这些进程使用流言协议（gossip protocols)来接收关于主服务器是否下线的信息， 并使用投票协议（agreement protocols）来决定是否执行自动故障迁移， 以及选择哪个从服务器作为新的主服务器。

单个 sentinel 进程来监控 redis 集群是不可靠的，当 sentinel 进程宕掉后(sentinel 本身也有单点问题，single-point-of-failure)整个集群系统将无法按照预期的方式运行。所以有必要将 sentinel 集群，这样有几个好处：

有一些 sentinel 进程宕掉了，依然可以进行 redis 集群的主备切换；
如果只有一个 sentinel 进程，如果这个进程运行出错，或者是网络堵塞，那么将无法实现 redis 集群的主备切换（单点问题）;
如果有多个 sentinel，redis 的客户端可以随意地连接任意一个 sentinel 来获得关于 redis 集群中的信息
一个健壮的部署至少需要三个哨兵实例。

三个哨兵实例应该放置在客户使用独立方式确认故障的计算机或虚拟机中。例如不同的物理机或不同可用区域的虚拟机。【本次讲解是一个机器上进行搭建，和多级是一个道理

背景

最近项目需求，接触到了 Redis 的搭建，简单记录下搭建过程中遇到的坑

总体配置

192.168.1.100:6379 -> master
192.168.1.101:6379 -> slave
192.168.1.102:6379 -> slave
192.168.1.100:26379 -> sentinel
192.168.1.101:26379 -> sentinel
192.168.1.102:26379 -> sentinel

搭建步骤

1.安装 redis

```
# 解压
tar -xvf /usr/local/redis-3.2.11.tar.gz

mkdir -p /usr/local/redis/bin

cp
/usr/local/redis/src/{redis-benchmark,redis-check-aof,redis-check-rdb,redis-cli,redis-sentinel,redis-server,redis-trib.rb} /usr/local/redis/bin

mkdir -p /u01/redis/{6379/{log,data,pid,conf},26379/{log,data,pid,conf}

# 添加环境变量
echo "export PATH=/usr/local/redis/bin:$PATH" >> /etc/profile
source /etc/profile
```

2.redis-6379 配置

redis 节点配置基本如下，把如下配置分别 cp 到三台虚拟机的 /u01/redis/6379/conf/redis_6379.conf

```
bind 0.0.0.0
protected-mode no
daemonize yes
pidfile "/u01/redis/6379/pid/redis_6379.pid"
port 6379
tcp-backlog 511
timeout 0
```

```
tcp-keepalive 0
loglevel notice
logfile "/u01/redis/6379/log/redis_6379.log"
databases 16
stop-writes-on-bgsave-error yes
rdbcompression yes
rdbchecksum yes
dbfilename "dump.rdb"
dir "/u01/redis/6379/data"
slave-serve-stale-data yes
slave-read-only yes
repl-diskless-sync no
repl-diskless-sync-delay 5
repl-disable-tcp-nodelay no
slave-priority 100
min-slaves-to-write 1
min-slaves-max-lag 10
appendonly no
appendfilename "appendonly.aof"
appendfsync everysec
no-appendfsync-on-rewrite no
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
aof-load-truncated yes
lua-time-limit 5000
slowlog-log-slower-than 10000
slowlog-max-len 128
latency-monitor-threshold 0
notify-keyspace-events ""
hash-max-ziplist-entries 512
hash-max-ziplist-value 64
list-max-ziplist-entries 512
```

启动服务

```
# 在三台虚拟机上分别执行
redis-server /u01/redis/6379/conf/redis_6379.conf
```

建立主从关系

```
# 在 192.168.1.101
redis-cli -p 6379 SLAVEOF 192.168.1.100 6379

# 在 192.168.1.102
redis-cli -p 6379 SLAVEOF 192.168.1.100 6379
```

查看 Replication

192.168.1.101:6379> info replication
Replication
role:master
connected_slaves:2
min_slaves_good_slaves:2
slave0:ip=192.168.1.102,port=6379,state=online,offset=9577826,lag=1
slave1:ip=192.168.1.103,port=6379,state=online,offset=9577965,lag=0
master_repl_offset:9577965
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:8529390
repl_backlog_histlen:1048576

192.168.1.102:6379> info replication
Replication
role:slave
master_host:192.168.1.101
master_port:6379
master_link_status:up
master_last_io_seconds_ago:0
master_sync_in_progress:0
slave_repl_offset:9600220
slave_priority:100
slave_read_only:1
connected_slaves:0
min_slaves_good_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0

192.168.1.103:6379> info replication
Replication
role:slave
master_host:192.168.1.101
master_port:6379
master_link_status:up
master_last_io_seconds_ago:0
master_sync_in_progress:0
slave_repl_offset:9612675

slave_priority:100
slave_read_only:1
connected_slaves:0
min_slaves_good_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0

3.sentinel-6379 配置

sentinel 节点配置基本如下，把如下配置分别 cp 到三台虚拟机的 /u01/redis/26379/conf/sentinel_26379.conf

sentinel monitor mymaster 后监控的是 redis 中的 master 节点，也就是 192.168.1.100，所以这个文件在三台机器上是相同的
port 26379
bind 0.0.0.0
daemonize yes
protected-mode no
dir "/u01/redis/26379/tmp"
logfile "/u01/redis/26379/log/sentinel_26379.log"
sentinel monitor mymaster 192.168.1.100 6379 1
等待启动完毕后观察/u01/redis/26379/conf/sentinel_26379.conf 文件变化

查看 sentinel 状态用 info sentinel
redis-cli -h 192.168.1.100 -p 26379 info sentinel

Sentinel
sentinel_masters:1
sentinel_tilt:0
sentinel_running_scripts:0
sentinel_scripts_queue_length:0
sentinel_simulate_failure_flags:0
master0:name=zhuanche01,status=ok,address=192.168.1.100:6379,slaves=2,sentinels=3

总结

我搭建的时候遇到了 192.168.1.101、192.168.1.102 上的 sentinel 启动后一段时间出错的问题，后来发现是没有监控 master
再就是出问题了多看 log

四、cluster 模式

cluster 的出现是为了解决单机 Redis 容量有限的问题，将 Redis 的数据根据一定的规则分配到多台机器。对 cluster 的一些理解：

cluster 可以说是 sentinel 和主从模式的结合体，通过 cluster 可以实现主从和 master 重选功能，所以如果配置两个副本三个分片的话，就需要六个 Redis 实例。
因为 Redis 的数据是根据一定规则分配到 cluster 的不同机器的，当数据量过大时，可以新增机器进行扩容

这种模式适合数据量巨大的缓存要求，当数据量不是很大使用 sentinel 即可。