redis FailOver - 1M + 1S

该文档只适用于一种最简单的failover，一台主，一台从。
至于多主多从的failover方案，还未详细查看。

该failover方案是结合keepalived来实现的。
服务器A，服务器B，下面为初始状态

	ServerA	ServerB
keepalived	master	backup
redis	master	slave

服务器A与服务器B的redis配置差别只有appendonly，其余配置均一致（当然，也可以自定义一些其他不一样的配置）

	ServerA	ServerB
appendonly	no	yes

服务器A的keepalived重要配置（其余配置与正常keepalived高可用方案一致）

```
global_defs {
	router_id test_node1
}
###定义检测脚本，用于检测redis状态，从而进行vip的切换
vrrp_script check_redis {
	script "/opt/keepalived_shell/redis/check.sh"
	interval 5
	fall 3
}
vrrp_instance Redis {
	state BACKUP
	interface eth0
	virtual_router_id 101
	priority 200
	advert_int 1
	nopreempt
	track_interface {
		eth0
	}
	authentication {
		auth_type pass
		auth_pass 123456
	}
	virtual_ipaddress {
		192.168.223.200
	}
	track_script {
		check_redis
	}
###定义keepalived实例状态发生变化时执行的脚本，在A上不起什么作用，主要就是输出下日志
而已
	notify_master "/bin/bash /opt/keepalived_shell/redis/master.sh"
	notify_backup "/bin/bash /opt/keepalived_shell/redis/backup.sh"
	notify_fault "/bin/bash /opt/keepalived_shell/redis/fault.sh"
###A、B服务器上的上面3个脚本均一致
}
```

服务器B的keepalived重要配置（其余配置与正常keepalived高可用方案一致）

```
global_defs {
	router_id test_node2
}
###定义检测脚本，用于检测redis状态，从而进行vip的切换
vrrp_script check_redis {
	script "/opt/keepalived_shell/redis/check.sh"
	interval 5
	fall 3
}
vrrp_instance Redis {
	state BACKUP
	interface eth0
	virtual_router_id 101
	priority 150
	advert_int 1
	nopreempt
	track_interface {
		eth0
	}
	authentication {
		auth_type pass
		auth_pass 123456
	}
	virtual_ipaddress {
		192.168.223.200
	}
	track_script {
		check_redis
	}
###定义keepalived实例状态发生变化时执行的脚本，在B上起到了辅助redis切换主备的操作
	notify_master "/bin/bash /opt/keepalived_shell/redis/master.sh"
	notify_backup "/bin/bash /opt/keepalived_shell/redis/backup.sh"
	notify_fault "/bin/bash /opt/keepalived_shell/redis/fault.sh"
###A、B服务器上的上面3个脚本均一致
}
```

keepalived中涉及到的4个脚本，如下

```
/opt/keepalived_shell/redis/check.sh
#!/bin/bash
###redis-cli命令，指定端口
REDIS_CLI='/usr/local/redis-2.8.6/src/redis-cli -p 6379'
```

```
ALIVE=$($REDIS_CLI PING)
[ "$ALIVE" == "PONG" ] && {
 exit 0
} || {
 exit -1
}
```

/opt/keepalived_shell/redis/master.sh

```
#!/bin/bash
###输出日志的记录
log=/opt/keepalived_shell/redis/change.log
###redis-cli命令，指定端口
REDIS_CLI='/usr/local/redis-2.8.6/src/redis-cli -p 6379'
###master redis的ip，由于这里slaveof no one，故这个配置填写no
MASTER_IP=no
###master redis的端口，由于这里slaveof no one，故这个配置填写one
MASTER_PORT=one
echo "$(date +%F-%T) KeepAlived ### Redis Instance changed to MASTER" >> $log
echo "$(date +%F-%T) KeepAlived ### Redis Instance going to slaveof $MASTER_IP
$MASTER_PORT" >> $log
$REDIS_CLI slaveof $MASTER_IP $MASTER_PORT
echo "$(date +%F-%T) KeepAlived ### Redis Instance going to config set appendonly no" >>
$log
$REDIS_CLI config set appendonly no
echo >> $log
```

/opt/keepalived_shell/redis/backup.sh

```
#!/bin/bash
###输出日志的记录
log=/opt/keepalived_shell/redis/change.log
###redis-cli命令，指定端口
REDIS_CLI='/usr/local/redis-2.8.6/src/redis-cli -p 6379'
###master redis的ip，直接使用虚地址即可！！！
MASTER_IP=192.168.223.200
###master redis的端口
MASTER_PORT=6379
echo "$(date +%F-%T) KeepAlived ### Redis Instance changed to SLAVE" >> $log
echo "$(date +%F-%T) KeepAlived ### Redis Instance going to slaveof $MASTER_IP
$MASTER_PORT" >> $log
$REDIS_CLI slaveof $MASTER_IP $MASTER_PORT
echo "$(date +%F-%T) KeepAlived ### Redis Instance going to config set appendonly yes" >>
$log
$REDIS_CLI config set appendonly yes
echo >> $log
```

/opt/keepalived_shell/redis/fault.sh

```
#!/bin/bash
###输出日志的记录
log=/opt/keepalived_shell/redis/change.log
###redis-cli命令，指定端口
REDIS_CLI='/usr/local/redis-2.8.6/src/redis-cli -p 6379'
###master redis的ip，直接使用虚地址即可！！！
```

```
MASTER_IP=192.168.223.200
###master redis的端口
MASTER_PORT=6379
echo "$(date +%F-%T) KeepAlived ### Redis Instance changed to FAULT" >> $log
echo "$(date +%F-%T) KeepAlived ### Redis Instance going to slaveof $MASTER_IP
$MASTER_PORT" >> $log
$REDIS_CLI slaveof $MASTER_IP $MASTER_PORT
echo "$(date +%F-%T) KeepAlived ### Redis Instance going to config set appendonly yes" >>
$log
$REDIS_CLI config set appendonly yes
echo >> $log
```

下面是整个failover的过程

1.正常状态下是这样的

	A	B
redis	√	√
keepalived	M	S

2.当主redis出现问题，变为下面这样

	A	B
redis	×	√
keepalived	S	M

此时,B上的keepalived由S切换到了M，配置中的/opt/keepalived_shell/redis/master.sh就会自动执行
结果就和脚本中写的一样：1.slaveof no one   2.appendonly no
于是B上的redis就承担了对外提供服务的责任。
由于A上的主redis出现故障的时候是我们不可预测的，因此通过keepalived来实现自动将B升级为主是很合适的。

3.当我们需要将A上的redis恢复成主，即整个架构恢复成上面的正常状态时，需要手动执行下面几个步骤
①恢复A上的redis服务

```
./redis-server ../etc/redis.conf
```

②将B上redis的数据同步到A上的redis
在执行了①之后，由于/opt/keepalived_shell/redis/backup.sh的存在，keepalived会自动替我们做②和③
但是为了安全起见，我们最好还是手动执行一下②和③，执行的同时，注意查看redis日志，确保同步完成！

```
./redis-cli slaveof 192.168.223.101 6379
```

③将A上的redis变回master状态 --- 此时A上的redis只是单方面的master状态，vip还没有切回A上，且B上redis也认为自己是master
在执行了①之后，由于/opt/keepalived_shell/redis/backup.sh的存在，keepalived会自动替我们做②和③
但是为了安全起见，我们最好还是手动执行一下②和③，执行的同时，注意查看redis日志，确保同

步完成！

```
./redis-cli slaveof no one
```

④重启B上的keepalived服务

```
/etc/init.d/keepalived stop
/etc/init.d/keepalived start
```

4.然后，就又回到了正常状态

	A	B
redis	√	√
keepalived	M	S

并且，B上的keepalived由M切换到了S，配置中的/opt/check_shell/redis_2_slave.sh就会自动执行
结果就和脚本中写的一样：1.slaveof A　　2.appendonly yes
于是B就回到了正常的redis slave状态了。