一、前置

1.要求

1. 接受 板子上的GPIO口高低电平信息 得到 所在楼层 （搭配光电传感器）。
2. 控制 板子上的GPIO口的高低电平，控制继电器，从而控制电梯。
3. socket 和 某台服务器保持长链接。
4. 建议2-1-3的顺序做

2.硬件

2.1 购买地址

1. 树莓派
2. 继电器
3. 光电传感器

2.2 树莓派4B

用户名：pi

密码：ccc111222

根据操作方式配置树莓派系统环境，java方式见参考10

GPIO引脚图40pin

树莓派 40Pin 引脚对照表

wiringPi 编码	BCM 编码	功能名	物理引脚 BOARD编码		功能名	BCM 编码	wiringPi 编码
		3.3V	1	2	5V		
8	2	SDA.1	3	4	5V		
9	3	SCL.1	5	6	GND		
7	4	GPIO.7	7	8	TXD	14	15
		GND	9	10	RXD	15	16
0	17	GPIO.0	11	12	GPIO.1	18	1
2	27	GPIO.2	13	14	GND		
3	22	GPIO.3	15	16	GPIO.4	23	4
		3.3V	17	18	GPIO.5	24	5
12	10	MOSI	19	20	GND		
13	9	MISO	21	22	GPIO.6	25	6
14	11	SCLK	23	24	CE0	8	10
		GND	25	26	CE1	7	11
30	0	SDA.0	27	28	SCL.0	1	31
21	5	GPIO.21	29	30	GND		
22	6	GPIO.22	31	32	GPIO.26	12	26
23	13	GPIO.23	33	34	GND		
24	19	GPIO.24	35	36	GPIO.27	16	27
25	26	GPIO.25	37	38	GPIO.28	20	28
		GND	39	40	GPIO.29	21	29

表格由树莓派实验室绘制 http://shumeipai.nxez.com

升级wiringPi库版本

```
#查看当前版本
gpio -v
#进入tmp目录
cd /tmp
#下载
wget https://project-downloads.drogon.net/wiringpi-latest.deb
#执行安装
sudo dpkg -i wiringpi-latest.deb
#查询所有引脚情况
gpio readall
```

shell操作Gpio

方式一:

```
#列出所有针角
gpio readal
#设置[以writePi编号为1]的GPIO（即GPIO1口） 口为输出模式
gpio mode 1 out
#设置[以BCM编号为18]的GPIO（即GPIO1口）口为输出模式
gpio read 1
#设置当前GPIO1口的电平为0（低电平）
gpio write 1 0
```

方式二:

```
#进入/sys/class/gpio/目录并查看文件
cd /sys/class/gpio/
#将gpio18重定向用户定义设备，生成gpio18目录,BCM?
sudo echo 18 > export
#进入gpio18目录并查看文件
cd gpio18
#设置引脚状态为输入状态
sudo echo in > direction
#查看引脚高低电平
cat value
#设置引脚状态为输出状态
sudo echo out > direction
#设置输出高电平
sudo echo 1 > value
#设置输出低电平
sudo echo 0 > value
#测试完毕之后返回/sys/class/gpio/目录
cd /sys/class/gpio/
#将gpio注销
sudo echo 18 > /sys/class/gpio/unexport
```

树莓派常见命令

```
#关机
sudo shutdown -h now
sudo halt
sudo poweroff
sudo init 0
#重启
sudo reboot
shutdown -r now
```

检测Gpio信号输入

基本:

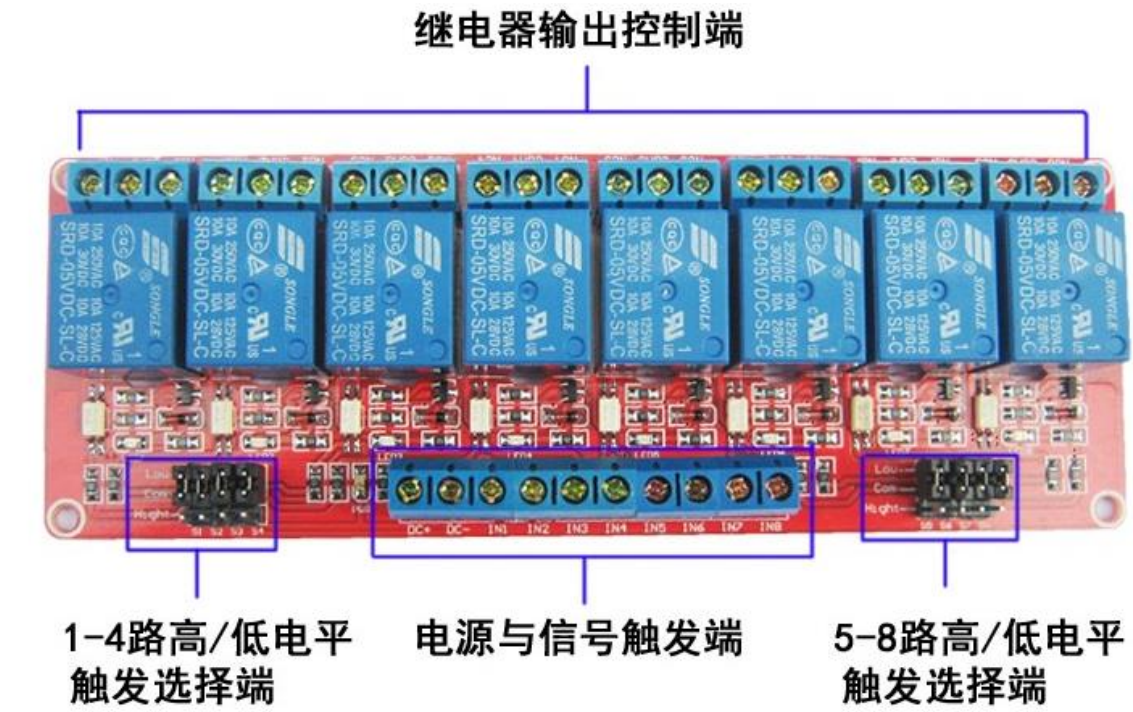
```
#导入 RPi.GPIO 模块
import RPi.GPIO as GPIO
#指定您所使用的方式（必须指定）
GPIO.setmode(GPIO.BOARD)
```

```
或者
GPIO.setmode(GPIO.BCM)
#禁用该警告消息
GPIO.setwarnings(False)
#channel通道编号是基于您所使用的编号系统所指定的（BOARD 或 BCM）
GPIO.setup(channel, GPIO.IN)
#读取 GPIO 针脚的值
GPIO.input(channel)
#在您的脚本结束后进行清理
GPIO.cleanup()
```

进阶

```
import RPi.GPIO as GPIO  ##引入GPIO模块
import time              ##引入time库
detectPin = 4
GPIO.setmode(GPIO.BCM)   ##此处采用的BCM编码 因为T型扩展板也是BCM编码 方便统一
GPIO.setup(detectPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)  ##上拉电阻，高电平
#GPIO.setup(detectPin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) ##下拉电阻，低电平
print(GPIO.input(detectPin))
GPIO.cleanup()
```

2.3 高低电平继电器

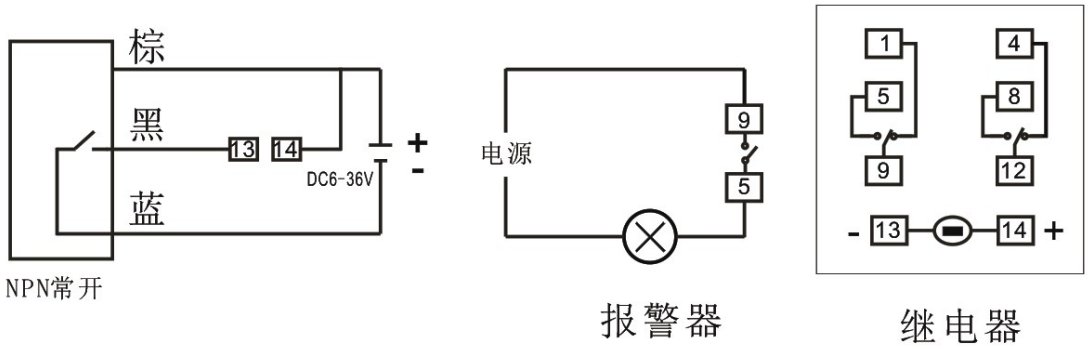


- 模块说明:
  1、模块采用优质继电器，常开接口负载：交流250V/10A，直流30V/10A;
  2、采用贴片光耦隔离，驱动能力强，性能稳定；触发电流5mA；
  3、模块工作电压有5V、9、12V、24V可供选择；
  4、模块的每一路都可以通过跳线设置高电平或低电平触发；
  5、容错设计，即使控制线断，继电器也不会动作；
  6、电源指示灯（绿色），8路继电器状态指示灯（红色）
  7、接口设计人性化，所有接口均可通过接线端子直接连线引出，非常方便

8、模块尺寸：141.5mm * 50mm* 18.5mm（长*宽*高)
9、设有4个固定螺栓孔，孔3.1mm，间距136mm*44.5mm

- 模块接口:
  1、DC+：接电源正极（电压按继电器要求，有5V.9V.12V和24V选择）
  2、DC-：接电源负极
  3、IN1-IN8：根据每一路的设置，均可以高或低电平控制相应继电器吸合
- 高低电平触发选择端:
  1、S1-S8依次为继电器1路-8路的高低电平触发选择;
  2、com与low短接时，相应继电器为低电平触发，com与high端短接时为高电平触发
- 继电器输出端：有24线接口，所有接口均可直接连线引出，方便用户使用
  1、NO1--NO8： 继电器常开接口，继电器吸合前悬空，吸合后与COM短接
  2、COM1--COM8：继电器公用接口
  3、NC1--NC8： 继电器常闭接口，继电器吸合前与COM短接，吸合后悬空

2.4 光电传感器




```
import RPi.GPIO as GPIO  ##引入GPIO模块
import time              ##引入time库
detectPin = 4
GPIO.setmode(GPIO.BCM)   ##此处采用的BCM编码 因为T型扩展板也是BCM编码 方便统一
GPIO.setup(detectPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)  ##上拉电阻，高电平
#GPIO.setup(detectPin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) ##下拉电阻，低电平
print(GPIO.input(detectPin))
GPIO.cleanup()
```

2.5 串口继电器

```
#python方式，注意文件命名不要与serialxaing相同
import serial
ser=serial.Serial('/dev/ttyUSB0',9600)#如果是1就写1
#inr = '01 06 00 01 01 00 D9 9A'
inr = '01 06 00 01 02 00 D9 6A'
a_bytes = bytes.fromhex(inr)
print (a_bytes)
ser.write(a_bytes)
```

java操作见参考15

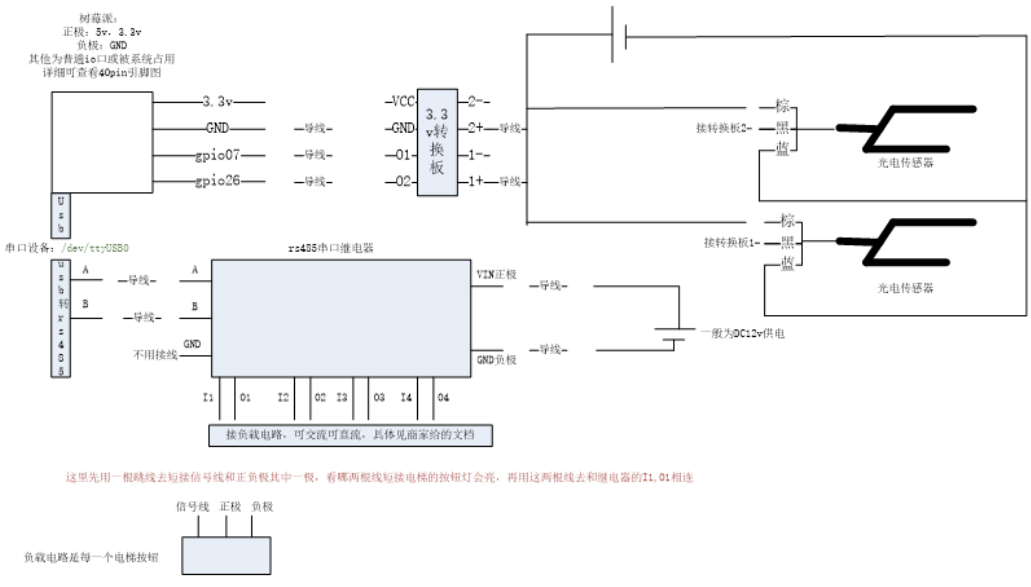
指令计算

```
指令计算


通道1打开 ：01 06 00 01 01 00 D9 9A
通道1关闭 ：01 06 00 01 02 00 D9 6A

通道2打开 ：01 06 00 02 01 00 29 9A
通道2关闭 ：01 06 00 02 02 00 29 6A

一共含有32个输出通道
格式：01 06 00 （通道编号）（01开启，02关闭） 00 最后两组数字由计算得出，计算方式如下

更多指令计算方式见
梯控综合文档\串口继电器商家文档\指令在线计算方式\On-line CRC calculation 2
```

3.可选方案

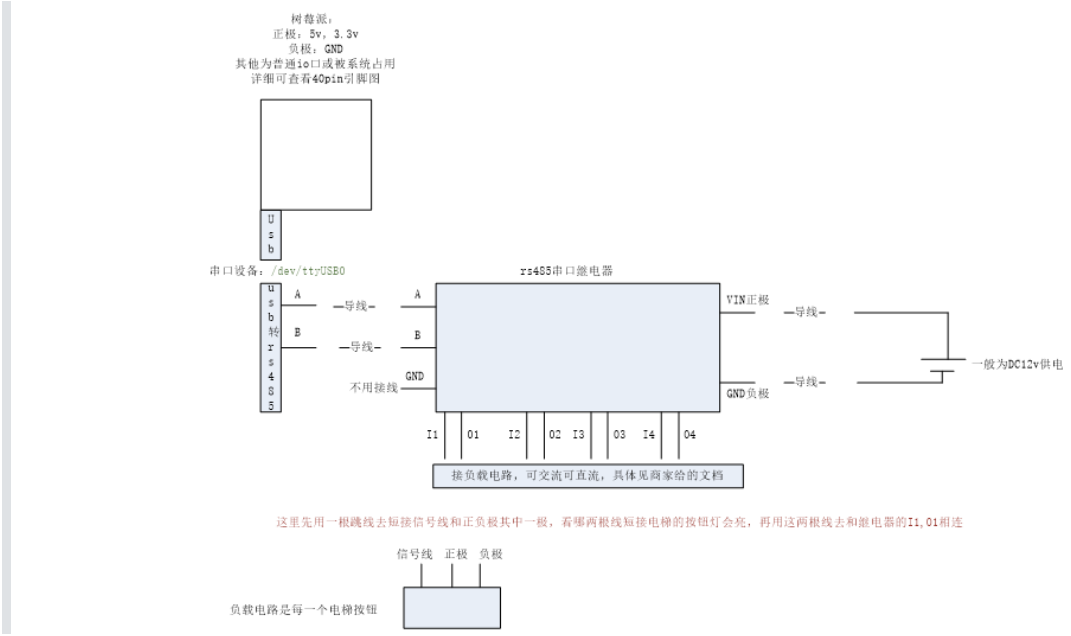
3.1含光电传感器

> 硬件：光电开关+串口继电器+树莓派
>
> 简介：这种方式需要接收光电开关的输入，其作用为通过接收光电传感器的输入获取当前楼层，提高了不稳定性



3.2不含光电传感器

> 硬件：串口继电器+树莓派
>
> 简介：这种方式不需要接收传感器的输入，稳定性较强



二、实施

1. 控制决策

- 树莓派控制楼层计数+1、-1,实际上接入两个光电传感器，来计算实现的,光电传感器就像开关一样，是读取光电传感器的状态，判断电梯有没有动,两个光电传感器一上一下，假设 A在上，B在下，如果A先有一个输入量，B再有一个输入量，A先接触到楼层标记挡杆是上升了一层楼，反之B先有的输入量，则是下楼
- 接收高低电平 是为了判断楼层,pi 先要做的是，给他一个楼层指令，打开电梯门，再去指令楼层
- 两种一个是NPN平层感应器，一个是PNP。具体怎么分辨，原理我就不讲了。说白了就是在遮挡光电的时候，一个输出高电平，一个输出低电平，不遮挡时信号相反。这里使用的是NPN平层感应器

2. 测试

快速调试

```
#杀死端口
fuser -k -n tcp 8080
cd /home/pi/apache-tomcat-9.0.36/webapps/ladder
java -jar
```

开机自启

```
#切换到root角色
sudo su
#修改rc.local文件，这里使用nano进行编辑，这里需要用到两个命令，一个是nano保存Ctrl+O，保存后直接enter表示保存为同名文件，即不修改文件名保存，另一个是Ctrl+X退出
sudo nano /etc/rc.local
#这里以jar包示例，在exit 0前面添加运行代码
sudo java -jar /绝对路径/xx.jar

#范例
sudo java -jar /home/pi/apache-tomcat-9.0.36/webapps/ladder/finalcut.jar
```

三、服务器部署

redis

Linux**下载地址：**http://redis.io/download，下载最新稳定版本。

```
# wget http://download.redis.io/releases/redis-6.0.8.tar.gz
# tar xzf redis-6.0.8.tar.gz
# cd redis-6.0.8
# make
```

执行完 **make** 命令后，redis-6.0.8 的 **src** 目录下会出现编译后的 redis 服务程序 redis-server，还有用于测试的客户端程序 redis-cli：

下面启动 redis 服务：

```
# cd src
# ./redis-server
```

注意这种方式启动 redis 使用的是默认配置。也可以通过启动参数告诉 redis 使用指定配置文件使用下面命令启动。

```
# cd src
# ./redis-server ../redis.conf
```

redis.conf 是一个默认的配置文件。我们可以根据需要使用自己的配置文件。

启动 redis 服务进程后，就可以使用测试客户端程序 redis-cli 和 redis 服务交互了。 比如：

```
# cd src
# ./redis-cli
redis> set foo bar
OK
redis> get foo
"bar"
```

启动 Redis

```
# redis-server
#以某配置文件启动，示例
./redis-server ./../redis.conf
```

查看 redis 是否启动?

```
# redis-cli
```

以某个端口启动

```
# redis-cli -p 3306
```

以上命令将打开以下终端：

```
redis 127.0.0.1:6379>
```

127.0.0.1 是本机 IP ，6379 是 redis 服务端口。现在我们输入 PING 命令。

```
redis 127.0.0.1:6379> ping
PONG
```

以上说明我们已经成功安装了redis。

redis集群

一主二从，哨兵检测模式

修改redis.conf配置文件

```
#cd /安装路径
#修改master的redis.conf
#vim redis.conf
```

需要修改的配置文件部分，无密码

```
#绑定本地主机
#bind 127.0.0.1
protected-mode no
port 6379
daemonize yes
pidfile /var/run/redis_6379.pid
logfile "./redis6379.log"
dbfilename dump6379.rdb

#如果是从节点，添加以下内容
replicaof <masterip> <masterport>
#从机只读
replica-read-only yes
```

有密码保护时

```
# 守护进程模式
daemonize yes
#注解掉  bind 127.0.0.1
#protected-mode yes 改为no
protected-mode  no
# 监听端口
port 6379
# pidfile 修改pidfile指向路径
pidfile 根据自己的位置/redis_master/redis_master.pid
# 指明日志文件名
logfile "./redis7001.log"
# 持久化数据库的文件名
dbfilename dump-master.rdb
# 工作目录
dir 根据自己的位置/redis-4.0.6/redis_master/
# 当master服务设置了密码保护时，slav服务连接master的密码
masterauth testmaster123
# 密码验证
requirepass testmaster123
```

```
# redis实例最大占用内存，不要用比设置的上限更多的内存。一旦内存使用达到上限，Redis会根据选定的回收策略
maxmemory 3gb
# volatile-lru -> 根据LRU算法删除带有过期时间的key。
maxmemory-policy volatile-lru
# 如果你有延时问题把这个设置成"yes"，否则就保持"no"，这是保存持久数据的最安全的方式。
no-appendfsync-on-rewrite yes

#如果是从节点，添加以下内容
replicaof <masterip> <masterport>
#从机只读
replica-read-only yes
```

redis.conf 配置项说明如下：

序号	配置项	说明
1	`daemonize no`	Redis 默认不是以守护进程的方式运行，可以通过该配置项修改，使用 yes 启用守护进程（Windows 不支持守护线程的配置为 no ）
2	`pidfile /var/run/redis.pid`	当 Redis 以守护进程方式运行时，Redis 默认会把 pid 写入 /var/run/redis.pid 文件，可以通过 pidfile 指定
3	`port 6379`	指定 Redis 监听端口，默认端口为 6379，作者在自己的一篇博文中解释了为什么选用 6379 作为默认端口，因为 6379 在手机按键上 MERZ 对应的号码，而 MERZ 取自意大利歌女 Alessia Merz 的名字
4	`bind 127.0.0.1`	绑定的主机地址
5	`timeout 300`	当客户端闲置多长秒后关闭连接，如果指定为 0 ，表示关闭该功能
6	`loglevel notice`	指定日志记录级别，Redis 总共支持四个级别：debug、verbose、notice、warning，默认为 notice
7	`logfile stdout`	日志记录方式，默认为标准输出，如果配置 Redis 为守护进程方式运行，而这里又配置为日志记录方式为标准输出，则日志将会发送给 /dev/null
8	`databases 16`	设置数据库的数量，默认数据库为0，可以使用 SELECT 命令在连接上指定数据库id
9	`save <seconds> <changes>` Redis 默认配置文件中提供了三个条件：**save 900 1****save 300 10save 60 10000** 分别表示 900 秒（15 分钟）内有 1 个更改，300 秒（5 分钟）内有 10 个更改以及 60 秒内有 10000 个更改。	指定在多长时间内，有多少次更新操作，就将数据同步到数据文件，可以多个条件配合
10	`rdbcompression yes`	指定存储至本地数据库时是否压缩数据，默认为 yes，Redis 采用 LZF 压缩，如果为了节省 CPU 时间，可以关闭该选项，但会导致数据库文件变的巨大
11	`dbfilename dump.rdb`	指定本地数据库文件名，默认值为 dump.rdb
12	`dir ./`	指定本地数据库存放目录
13	`slaveof <masterip> <masterport>`	设置当本机为 slave 服务时，设置 master 服务的 IP 地址及端口，在 Redis 启动时，它会自动从 master 进行数据同步，在新版本这个标签被替换
14	`masterauth <master-password>`	当 master 服务设置了密码保护时，slav 服务连接 master 的密码

序号	配置项	说明
15	`requirepass foobared`	设置 Redis 连接密码，如果配置了连接密码，客户端在连接 Redis 时需要通过 AUTH 命令提供密码，默认关闭
16	`maxclients 128`	设置同一时间最大客户端连接数，默认无限制，Redis 可以同时打开的客户端连接数为 Redis 进程可以打开的最大文件描述符数，如果设置 maxclients 0，表示不作限制。当客户端连接数到达限制时，Redis 会关闭新的连接并向客户端返回 max number of clients reached 错误信息
17	`maxmemory <bytes>`	指定 Redis 最大内存限制，Redis 在启动时会把数据加载到内存中，达到最大内存后，Redis 会先尝试清除已到期或即将到期的 Key，当此方法处理后，仍然到达最大内存设置，将无法再进行写入操作，但仍然可以进行读取操作。Redis 新的 vm 机制，会把 Key 存放内存，Value 会存放在 swap 区
18	`appendonly no`	指定是否在每次更新操作后进行日志记录，Redis 在默认情况下是异步的把数据写入磁盘，如果不开启，可能会在断电时导致一段时间内的数据丢失。因为 redis 本身同步数据文件是按上面 save 条件来同步的，所以有的数据会在一段时间内只存在于内存中。默认为 no
19	`appendfilename appendonly.aof`	指定更新日志文件名，默认为 appendonly.aof
20	`appendfsync everysec`	指定更新日志条件，共有 3 个可选值：**no**：表示等操作系统进行数据缓存同步到磁盘（快）**always**：表示每次更新操作后手动调用 fsync() 将数据写到磁盘（慢，安全）**everysec**：表示每秒同步一次（折中，默认值）
21	`vm-enabled no`	指定是否启用虚拟内存机制，默认值为 no，简单的介绍一下，VM 机制将数据分页存放，由 Redis 将访问量较少的页即冷数据 swap 到磁盘上，访问多的页面由磁盘自动换出到内存中（在后面的文章我会仔细分析 Redis 的 VM 机制）
22	`vm-swap-file /tmp/redis.swap`	虚拟内存文件路径，默认值为 /tmp/redis.swap，不可多个 Redis 实例共享
23	`vm-max-memory 0`	将所有大于 vm-max-memory 的数据存入虚拟内存，无论 vm-max-memory 设置多小，所有索引数据都是内存存储的(Redis 的索引数据 就是 keys)，也就是说，当 vm-max-memory 设置为 0 的时候，其实是所有 value 都存在于磁盘。默认值为 0

序号	配置项	说明
24	`vm-page-size 32`	Redis swap 文件分成了很多的 page，一个对象可以保存在多个 page 上面，但一个 page 上不能被多个对象共享，vm-page-size 是要根据存储的 数据大小来设定的，作者建议如果存储很多小对象，page 大小最好设置为 32 或者 64bytes；如果存储很大大对象，则可以使用更大的 page，如果不确定，就使用默认值
25	`vm-pages 134217728`	设置 swap 文件中的 page 数量，由于页表（一种表示页面空闲或使用的 bitmap）是在放在内存中的，，在磁盘上每 8 个 pages 将消耗 1byte 的内存。
26	`vm-max-threads 4`	设置访问swap文件的线程数,最好不要超过机器的核数,如果设置为0,那么所有对swap文件的操作都是串行的，可能会造成比较长时间的延迟。默认值为4
27	`glueoutputbuf yes`	设置在向客户端应答时，是否把较小的包合并为一个包发送，默认为开启
28	`hash-max-zipmap-entries 64` `hash-max-zipmap-value 512`	指定在超过一定的数量或者最大的元素超过某一临界值时，采用一种特殊的哈希算法
29	`activerehashing yes`	指定是否激活重置哈希，默认为开启（后面在介绍 Redis 的哈希算法时具体介绍）
30	`include /path/to/local.conf`	指定包含其它的配置文件，可以在同一主机上多个Redis实例之间使用同一份配置文件，而同时各个实例又拥有自己的特定配置文件

```
#哨兵
# port <sentinel-port>
port 8001

# 守护进程模式
daemonize yes

# 指明日志文件名
logfile "./sentinel1.log"

# 工作路径，sentinel一般指定/tmp比较简单
dir ./

# 哨兵监控这个master，在至少quorum个哨兵实例都认为master down后把master标记为odown
# （objective down客观down；相对应的存在sdown，subjective down，主观down）状态。
# slaves是自动发现，所以你没必要明确指定slaves。
# 当前Sentinel节点监控 127.0.0.1 7001 这个主节点
# 2代表判断主节点失败至少需要2个Sentinel节点节点同意
# mymaster是主节点的别名
sentinel monitor MyMaster 自己服务的ip地址 7001 2

#当Sentinel节点集合对主节点故障判定达成一致时，Sentinel领导者节点会做故障转移操作，选出新的主节点，原来的从节点会向新的主节点发起复制操作，限制每次向新的主节点发起复制操作的从节点个数为1
```

```
sentinel parallel-syncs MyMaster 1

# master或slave多长时间（默认30秒）不能使用后标记为s_down状态。
# 每个Sentinel节点都要定期PING命令来判断Redis数据节点和其余Sentinel节点是否可达，如果超过30000毫秒且没有回复，则判定不可达
sentinel down-after-milliseconds MyMaster 1500

# 若sentinel在该配置值内未能完成failover操作（即故障时master/slave自动切换），则认为本次failover失败。
sentinel failover-timeout MyMaster 10000

# 设置master和slaves验证密码
# sentinel auth-pass <master-name> <password>
sentinel auth-pass MyMaster testmaster123

sentinel config-epoch MyMaster 15
#除了当前哨兵，还有哪些在监控这个master的哨兵，不用配置，启动后自动发现
#自动出现以下信息，一定要注意这里！！！！
sentinel known-sentinel MyMaster 127.0.0.1 8002
0aca3a57038e2907c8a07be2b3c0d15171e44da5
sentinel known-sentinel MyMaster  127.0.0.1 8003
ac1ef015411583d4b9f3d81cee830060b2f29862
```

```
第三步：  多个哨兵配置
其他哨兵只需要复制sentinel1.conf配置文件，重命名以后，需要如下改变如下配置即可：

# port <sentinel-port>
port 8002
# 指明日志文件名
logfile "./sentinel2.log"

第四步：  启动哨兵

启动哨兵：redis.sentinel  ./sentinel1.conf  或使用 redis.server ./sentinel1.conf  --sentinel
        redis.sentinel  ./sentinel2.conf
        redis.sentinel  ./sentinel3.conf
```

nginx

安装步骤：https://www.nginx.cn/install

使用：https://blog.csdn.net/blackbattery/article/details/104630481

四、参考

1. 通过继电器控制灯
2. 树莓派 继电器 实现led简单控制
3. 树莓派GPIO控制使用教程
4. 树莓派 GPIO按钮开关 原理与实现
5. 树莓派4b控制继电器，继电器控制小LED灯
6. 如何用树莓派控制 GPIO 引脚并操作继电器
7. 详解继电器
8. 树莓派驱动继电器点亮灯

9. 命令操作GPIO
10. JAVA操作GPIO
11. 使用树莓派GPIO控制继电器
12. 树莓派基础实验4：继电器实验
13. springboot本地项目开机自启1
14. springboot本地项目开机自启2
15. HEX格式转换Java