[Go语言开发分布式任务调度 1](#_Toc18556)

[概述 1](#_Toc14885)

[最终效果 2](#_Toc4187)

[架构搭建 3](#_Toc19481)

[Shell命令原理与使用 6](#_Toc24040)

[Golang运行shell的最小原理 8](#_Toc6352)

[简单实践 9](#_Toc6346)

[开发习惯 9](#_Toc16824)

[实例1: 并没有捕获响应的内容 9](#_Toc23440)

[实例2：完整示例与捕获显示 10](#_Toc25232)

[实例3: 强制结束任务 10](#_Toc22690)

[Cron表达式 12](#_Toc31531)

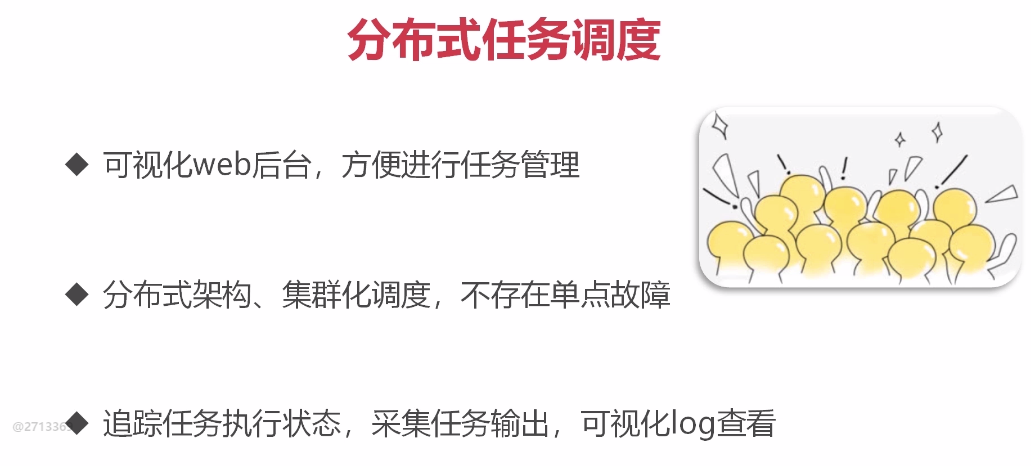
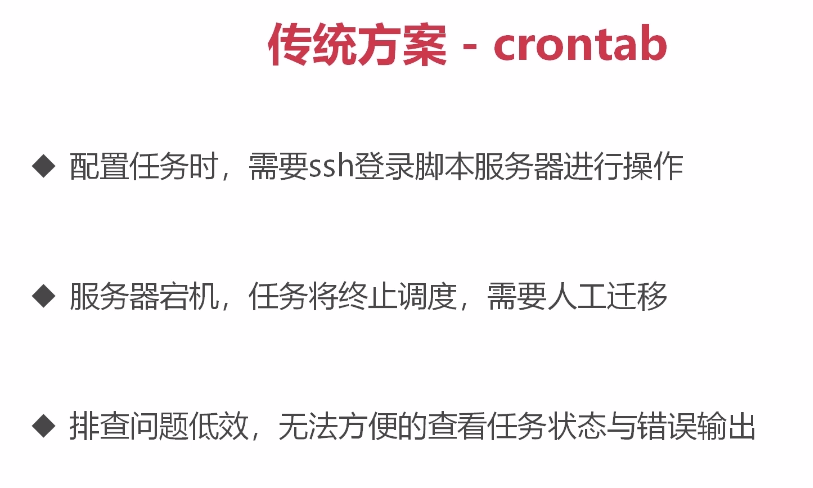
[如何计算下一次的调度时间 13](#_Toc21382)

[使用Cron解析库 14](#_Toc24575)

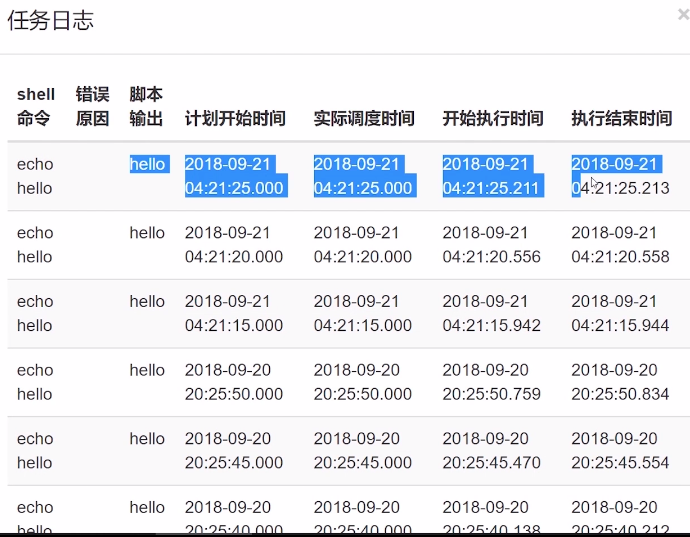
[Cronexpr库的使用例子 14](#_Toc14752)

# Go语言开发分布式任务调度

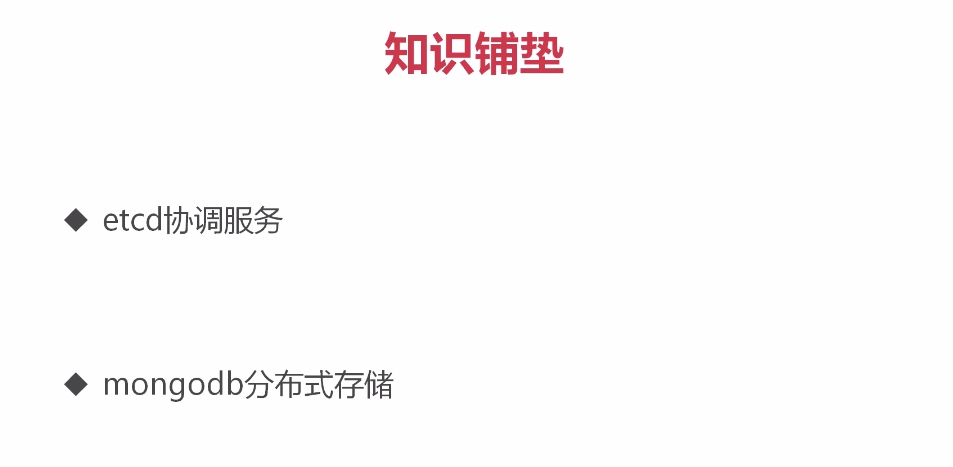
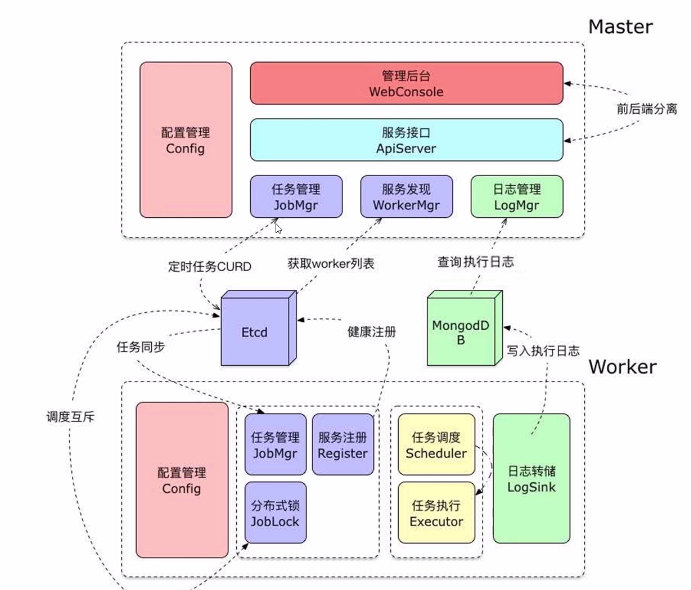
## 概述



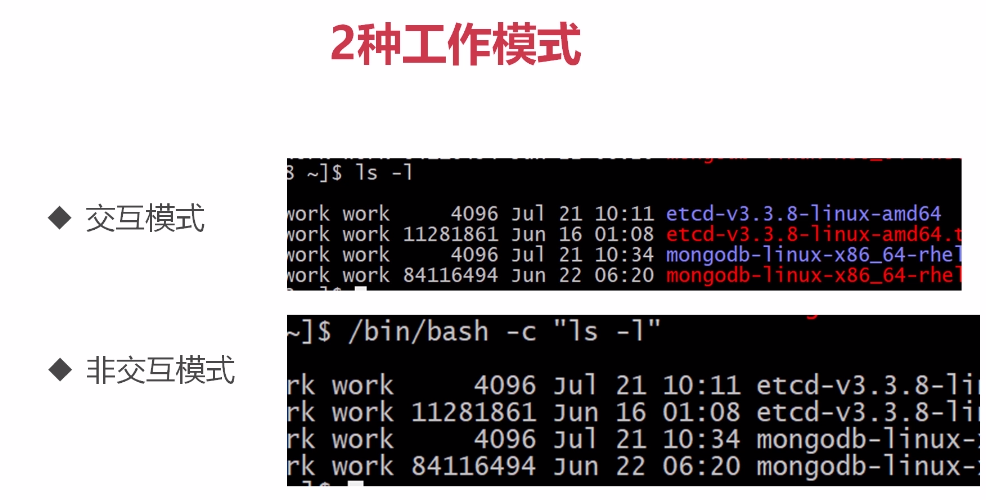
### 最终效果



### 架构搭建

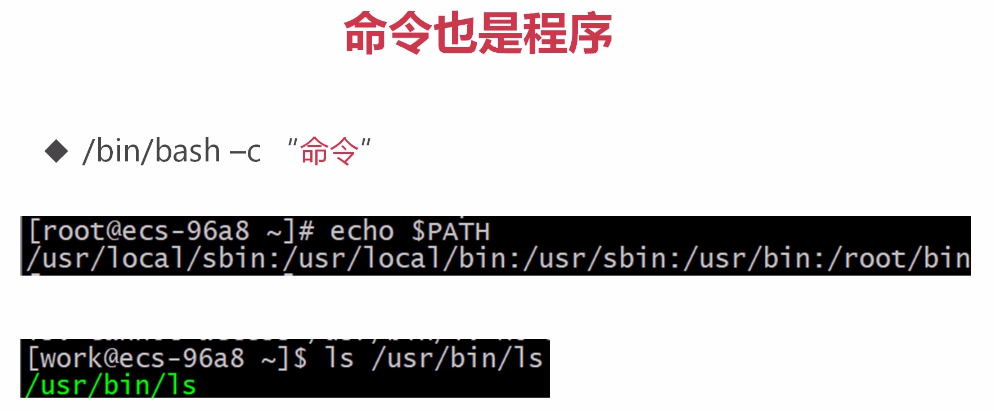


# Shell命令原理与使用

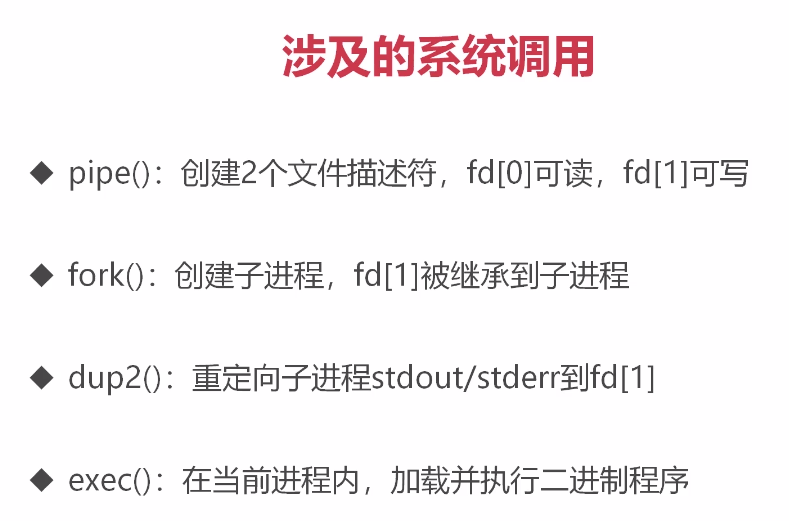
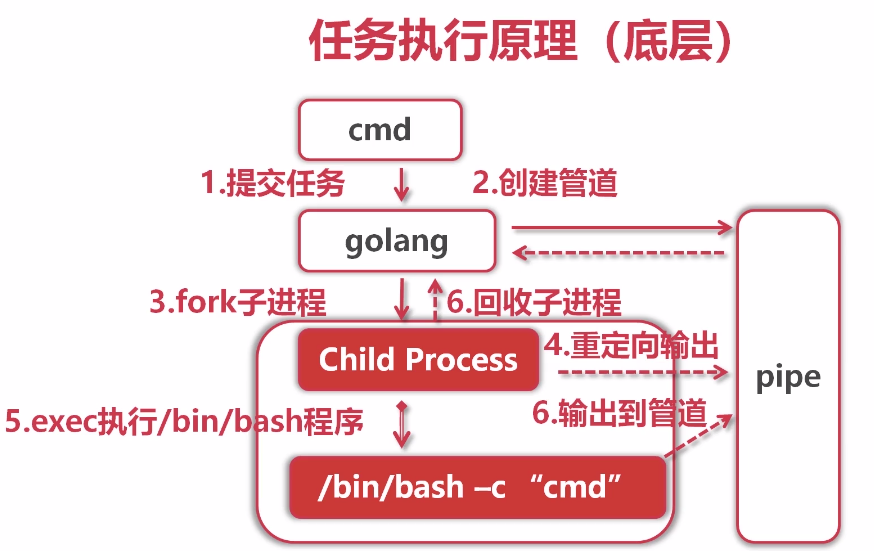


Golang调用外部程序都是使用执行命令的方式运行非交互模式的方法运行程序的

即 /bin/bash -c “命令”,从而让golang程序可以跑出任意的shell命令



## Golang运行shell的最小原理





## 简单实践

### 开发习惯

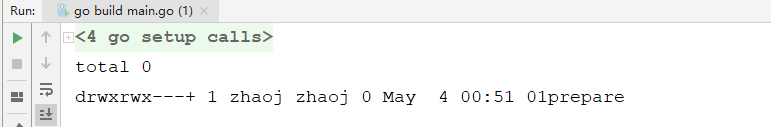
尽量多使用var ()定义变量，减少goto造成的影响

由于在window上不能直接进行linux命令的运行，所以需要借助cygwin64来进行运行，这样就可以在window上进行学习了

### 实例1: 并没有捕获响应的内容

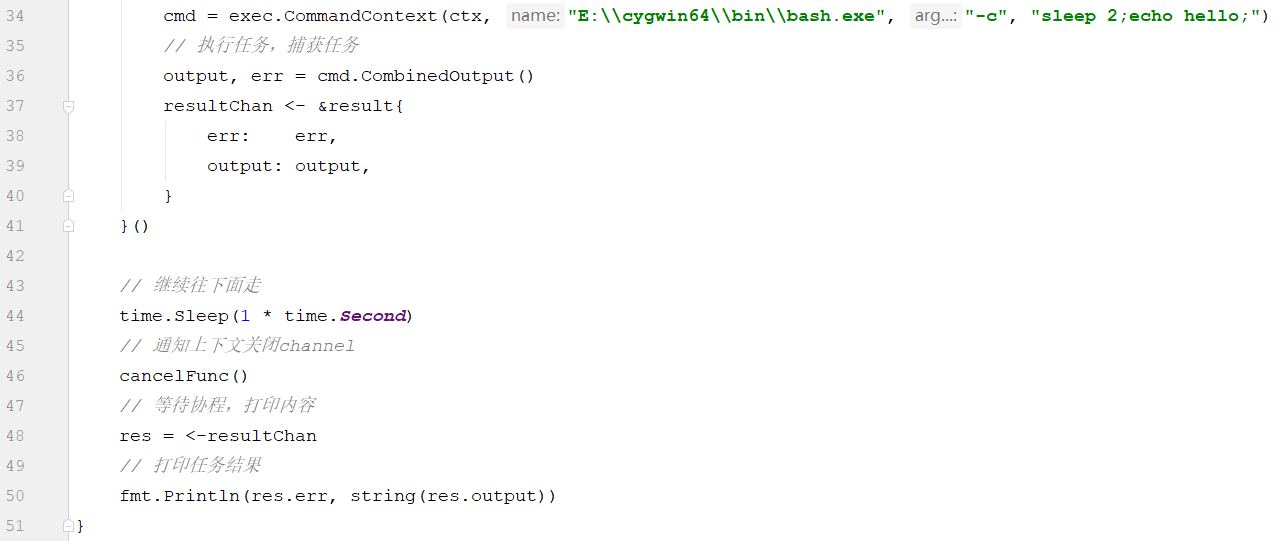


### 实例2：完整示例与捕获显示

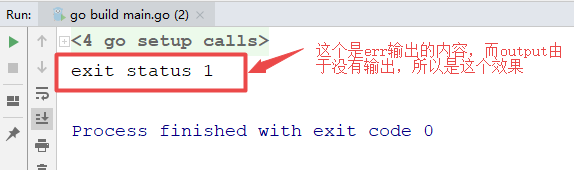


### 实例3: 强制结束任务

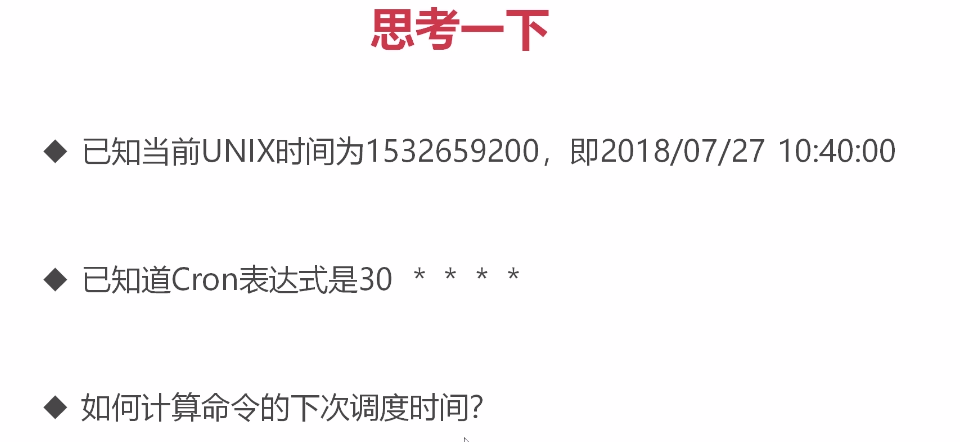
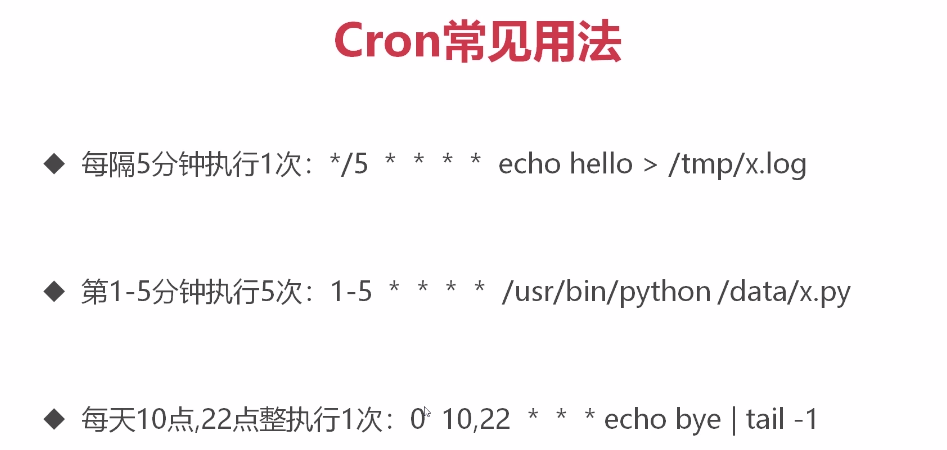
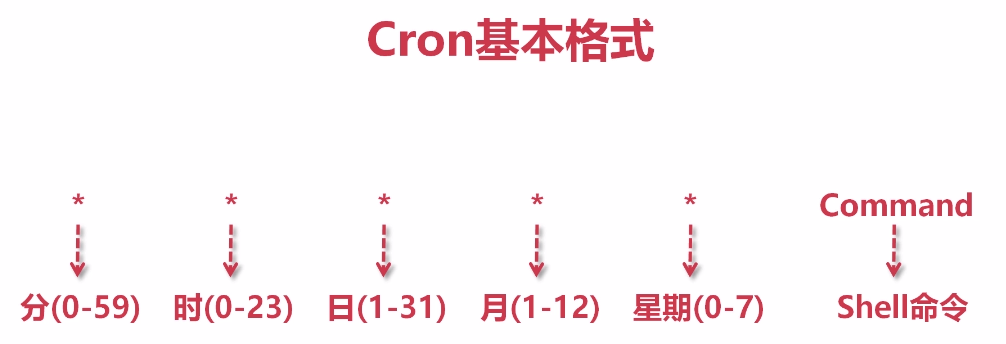
由于1s就通过cancelFunc()将协程杀死，所以会出现下面的效果



实现的结果



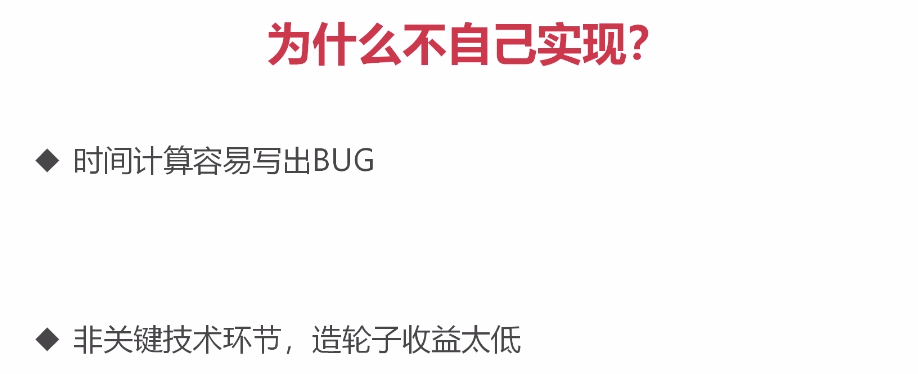
# Cron表达式



## 如何计算下一次的调度时间



## 使用Cron解析库

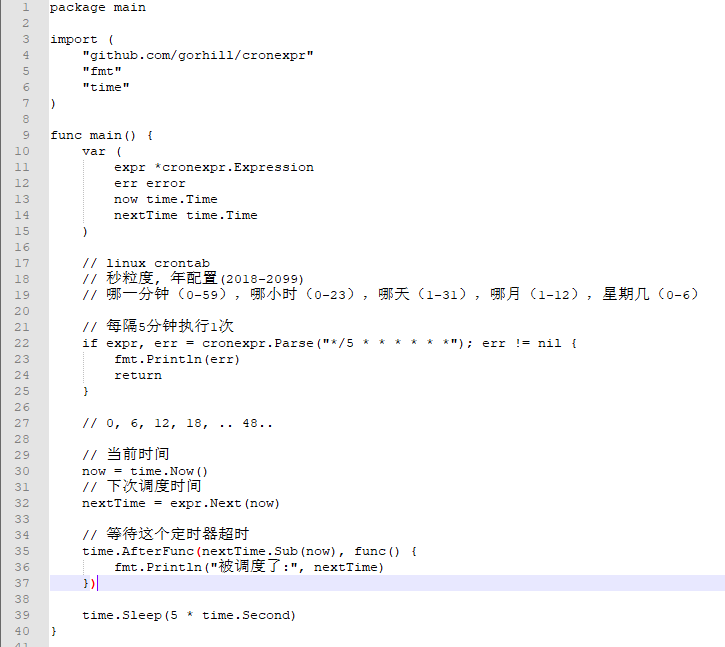


## Cronexpr库的使用例子

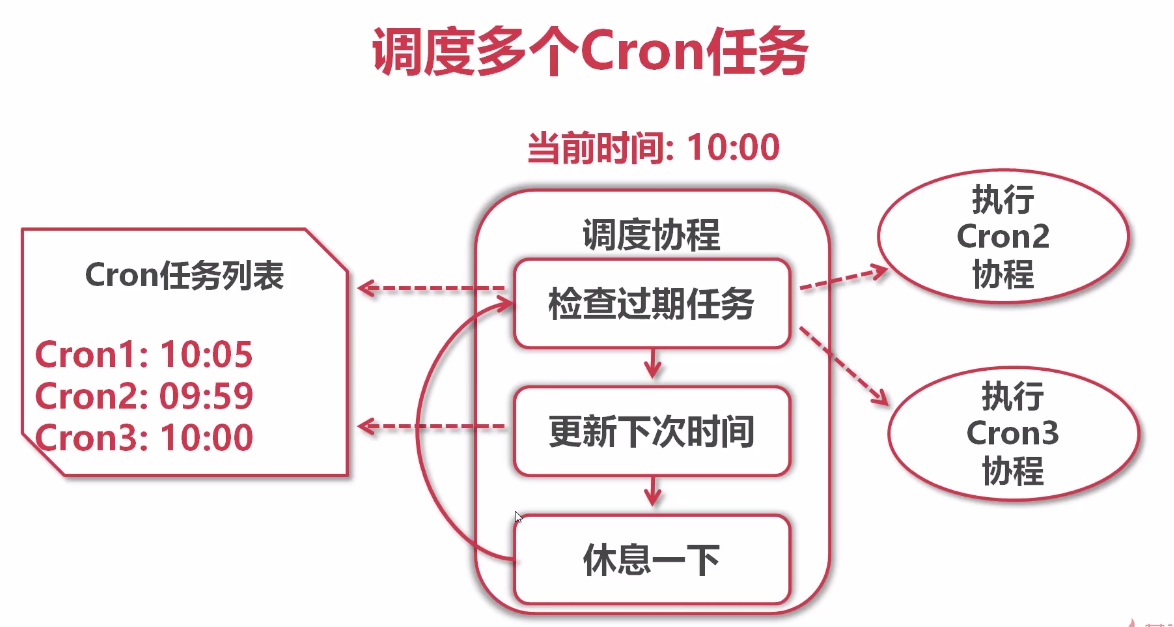
首先下载相关的库

go get github.com/gorhill/cronexpr

例子1：定时任务简单使用



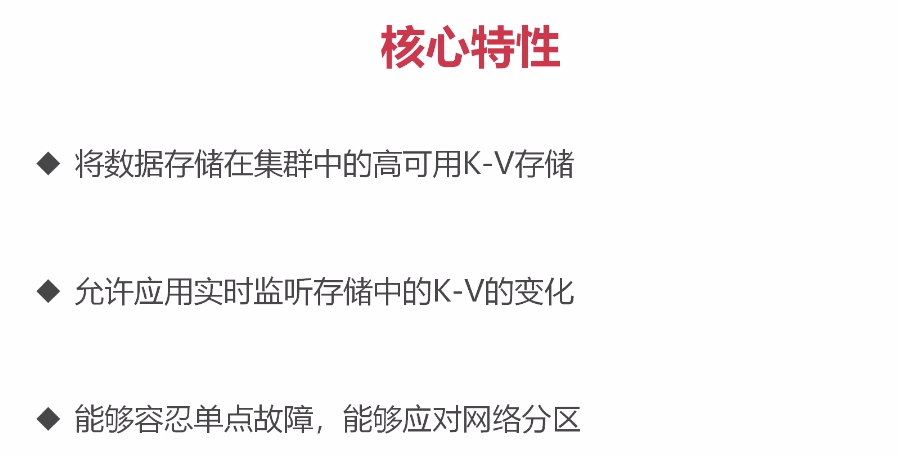
例子2：多个cron任务同时运行的开发





# ETCD协调服务

## etcd功能介绍





## 网络分区

分布式通常假设网络是异步的，意味着网络可能会导致任意的重复、丢失、延迟或者乱序的节点间消息传递。在实际中，TCP状态机会保证节点间消息传递的不丢失、不重复、时序。但是，在Socket级别上，节点接发消息会阻塞，超时等等。

检测到网络失败是困难，因为我们唯一能跟得到其他节点状态的信息就是通过网络来得到，延迟跟网络失败也无从区分。这里就会产生一个基本的网络分区问题:高延迟可以考虑作为失败。当分区产生后，我们没有渠道去了解到其他节点到底发生了什么事: 它们是否还存活？或者已经crash？是否有收到消息？是否正在尝试回应。当网络最终恢复后，我们需要重新建立连接然后尝试解决在不一致状态时的不一致。

很多系统在解决分区时会进入一个特殊的降级操作模式。CAP理论也告诉我们妖么得到一致性要么高可用性，但是很少有数据库系统能够达到CAP理论的极限，多数只是丢失数据。



## etcd原理

### 抽屉理论 -- 大多数理论



### etcd与Raft的关系



### quorum模型

注意:一定要铺开大多数理论的观念

集群一定是一个奇数节点，这样才能获取大多数,即N+1的设备知道了，就说明全部知道了。

#### 第一阶段:同步复制操作，并本地提交阶段，保证N+1个节点都刷新日志，但写入性能较差



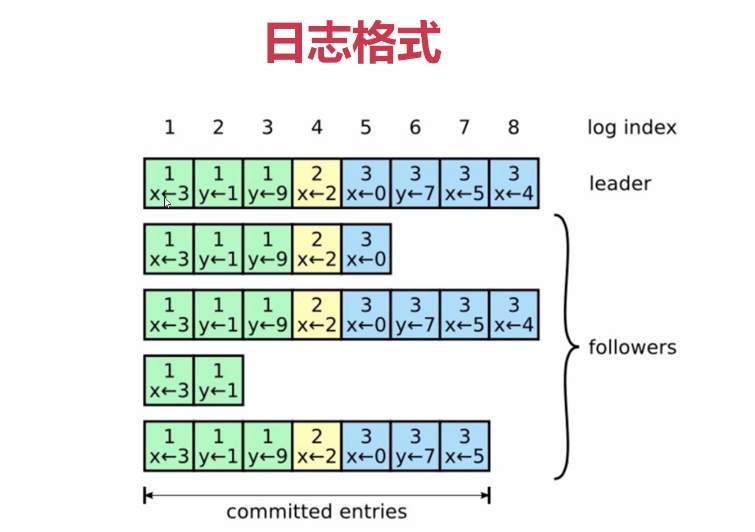
#### 第二节点：异步周期性提交其他follwer操作阶段



### Raft协议是二阶段协议

1. 首先是复制，一旦复制到大多数，就本地提交，就是告诉客户端提交成功
2. 后阶段是异步，是周期性的，只要保证第一阶段，大多数有日志操作记录就可以了

## 日志格式



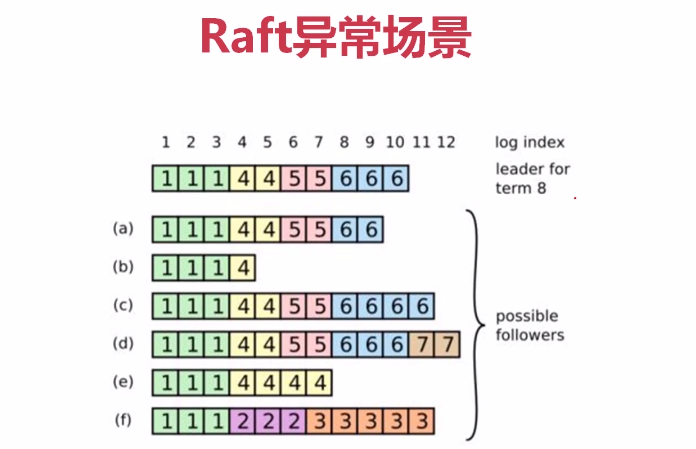
部分缺日志的情况分析，网络不通,日志过不去:

1. 网络延迟
2. 网络分区

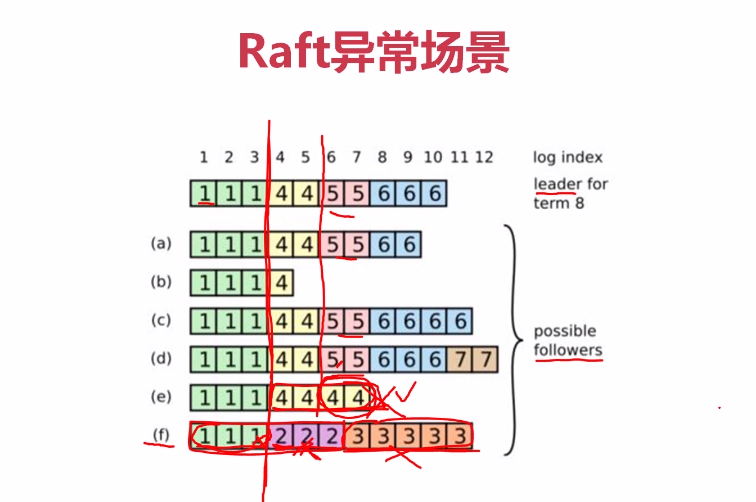
## Raft日志概念



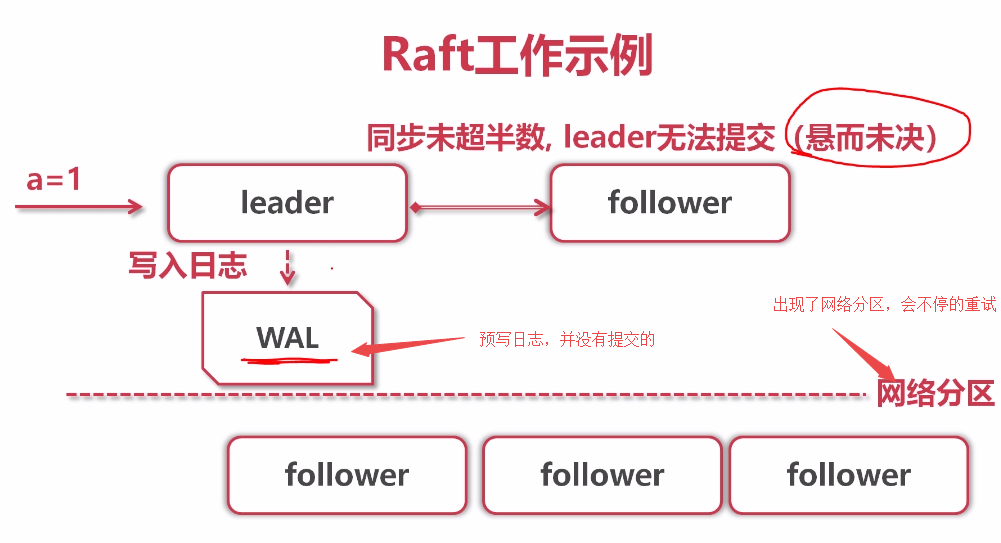
## Raft异常场景



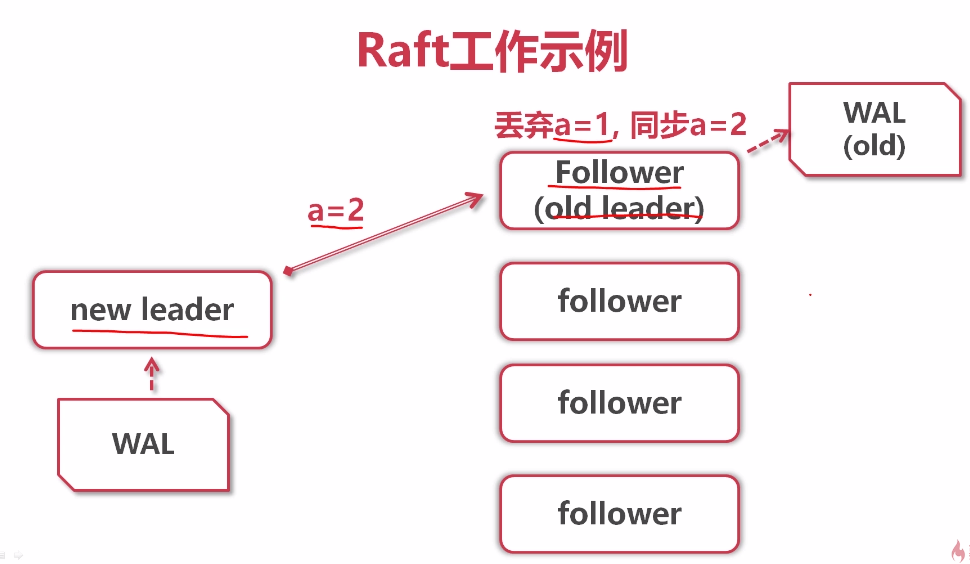
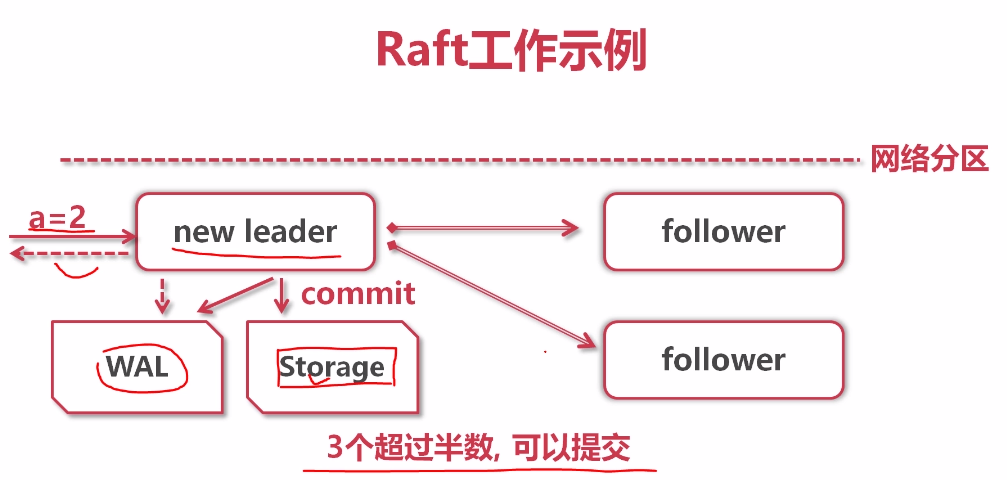
没有复制到其他节点就不会提交，一般来说就是节点down掉了



### Raft选举规则



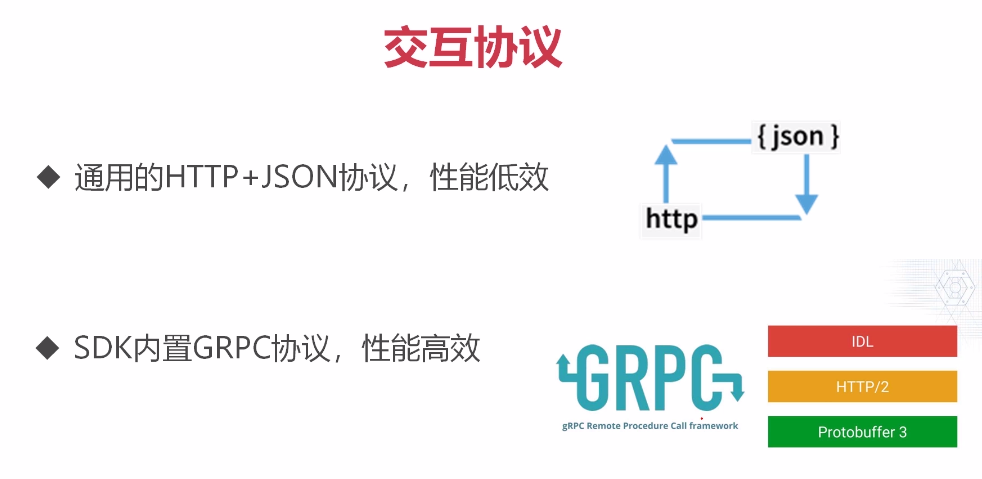
过了大多数的复制后，就进行本地提交，同时告诉客户端，我已经内容保存起来了.

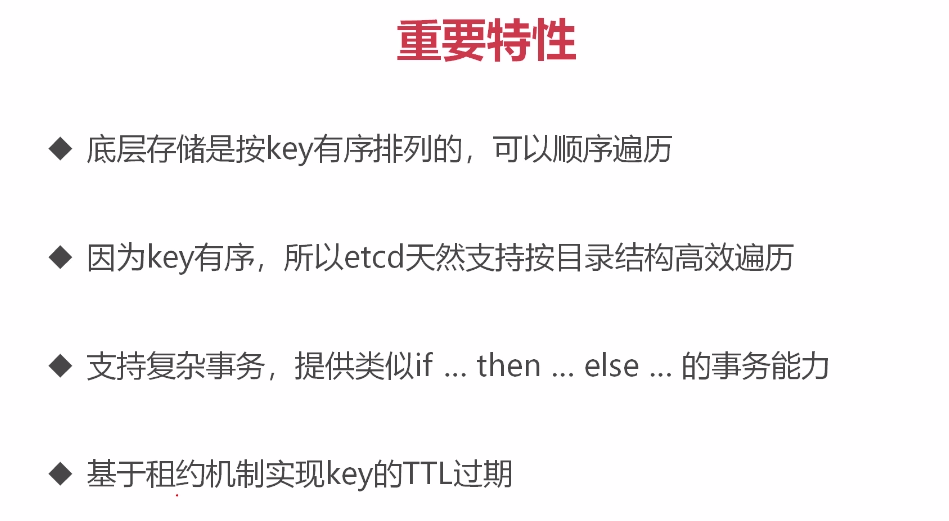


#### Raft保证的内容



### Etcd交互模式





### Key的有序存储

