分布式任务系统 cronsun

苏创绩

目录

01 Part One

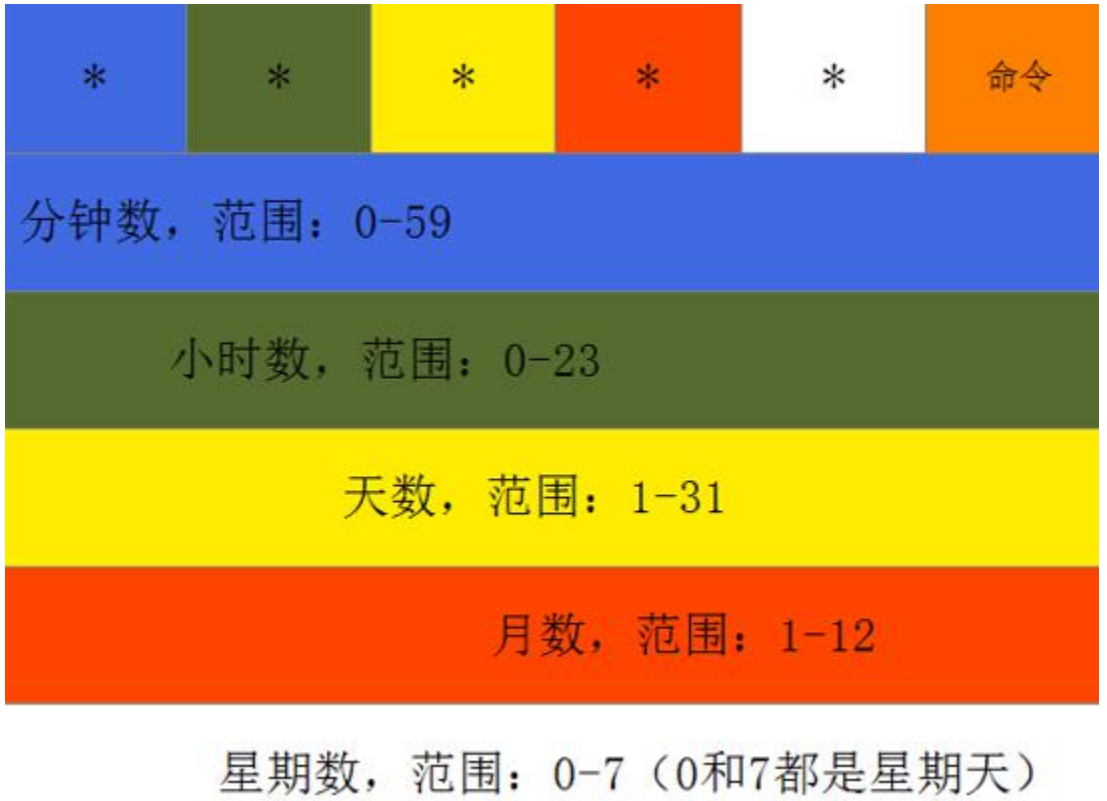
任务系统

任务

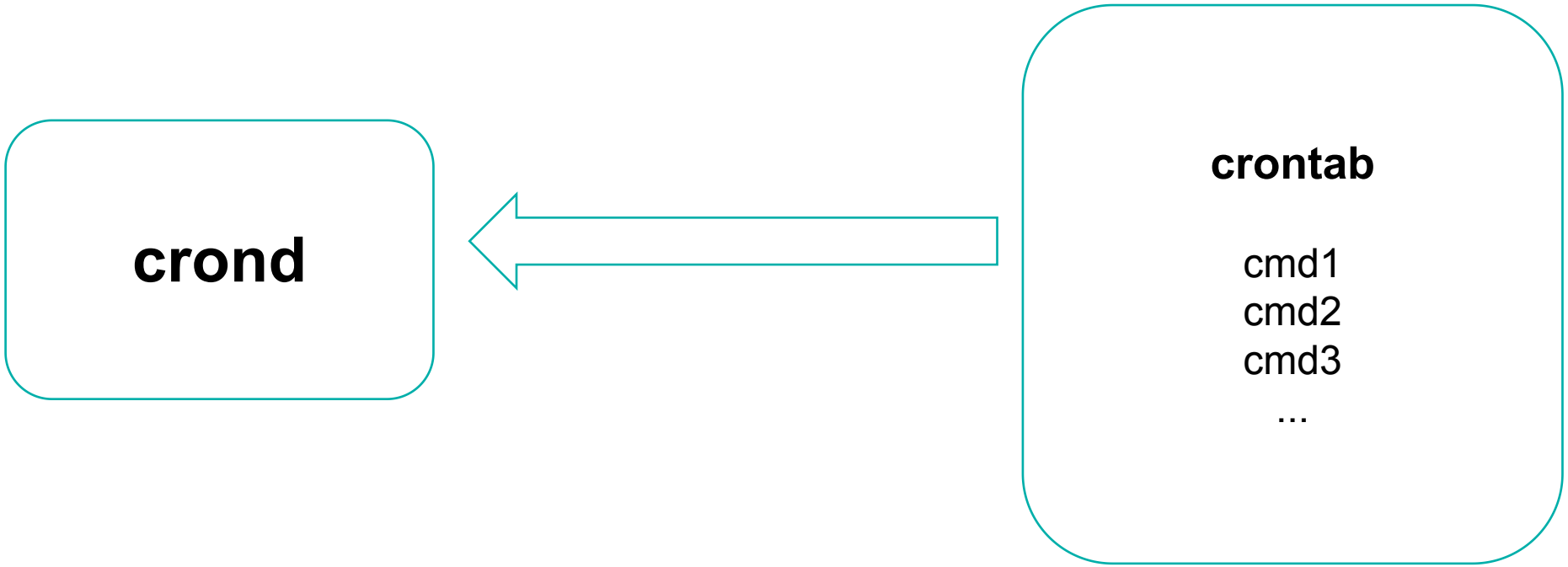
1. 什么时间
2. 什么地点
3. 做什么事

一个简单的任务

0 8 * * * echo "Hello Gophers!"



cron



早期的 cron

V7，1979

1. 在Version 7 Unix里是一个系统服务
2. 只用 root 运行任务
3. 算法简单直接

早期的 cron 运行逻辑

1. 读 /usr/lib/crontab 文件
2. 如果有命令要在当前时间执行，就用 root 用去执行命令
3. Sleep 1 minute
4. 重新从步骤 1 开始

支持多用户的 cron

Unix System V，1983

1. 启动的时候读取所有用户下的 .crontab 文件
2. 计算出每个 crontab 文件里需要执行的命令的下一次执行时间
3. 把这些命令按下一次执行时间排序后放入队列里
4. 进入主循环
   - ① 计算队列里第一个任务的执行时间与当前的时间差
   - ② Sleep 直到第一个任务执行时间
   - ③ 后台执行任务
   - ④ 计算这个任务的下一次执行时间，放回队列，排序

近代的 cron

Linux，1991

1. Vixie cron(Paul Vixie 1987)
2. Version 3 Vixie cron(1993)
3. Version 4.1 ISC Cron(2004)
4. anacron, dcron, fcron

cron 的局限性

1. 单机
2. 无界面
3. 功能比较简单
4. 多机器的情况下任务维护成本较高

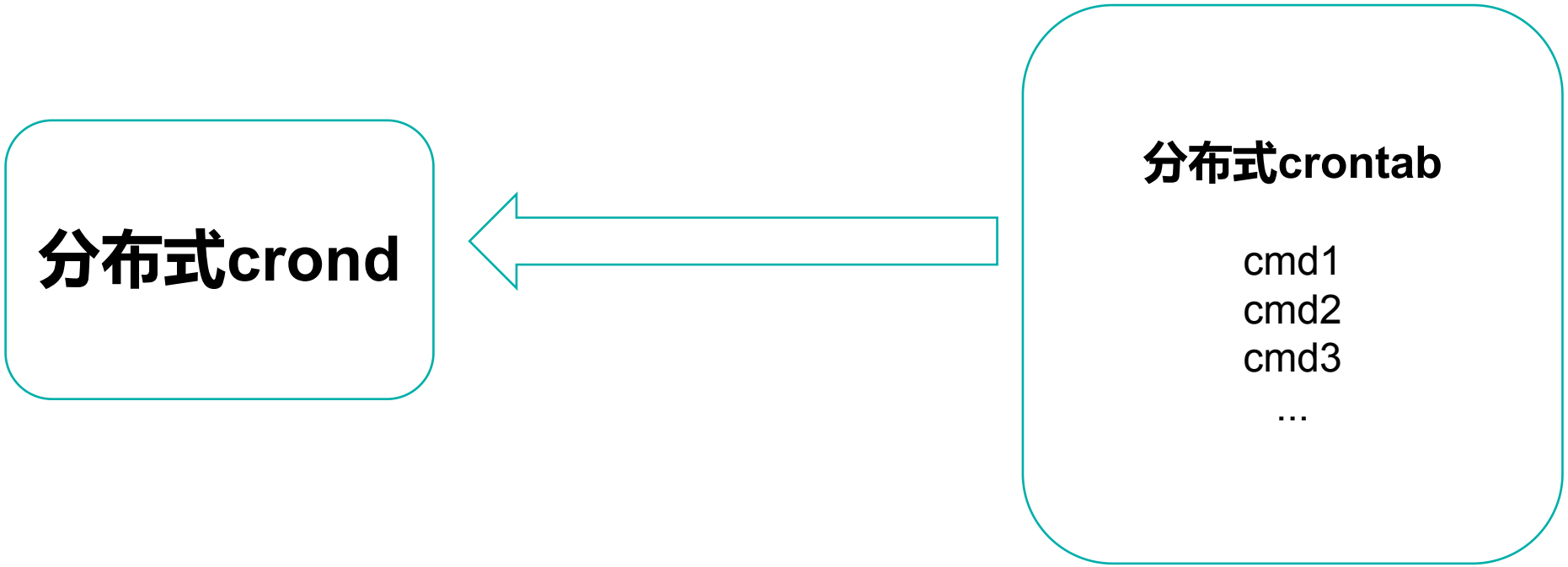
02 Part Two

分布式任务系统

分布式系统的特点

1. 分布性
2. 对等性
3. 并发性
4. 缺乏全局时钟
5. 故障总是会发生

分布式 cron



我只是不爱动，不是懒

市面上的一些任务系统

1. Azkaban
2. Chronos
3. Airflow
4. dkron
5. swoole-crontab
6. Saturn

Azkaban

批量工作流任务调度器(Hadoop)

1. 提供功能清晰，简单易用的 Web UI 界面
2. 提供 job 配置文件快速建立任务和任务之间的依赖关系
3. 提供模块化和可插拔的插件机制，原生支持 command、Java、Hive、Pig、Hadoop
4. …

Chronos

Chronos 是一个运行在 Mesos 之上的具有分布式容错特性的作业调度器

1. 可替代 cron
2. 有 UI
3. 支持ISO8601标准，允许更灵活地定义调度时间
4. 支持任务依赖

Dkron

分布式高可用的任务调度系统

1. 易用、有 UI
2. 高可用
3. 可扩展性高，支持大量任务和成千上万结点

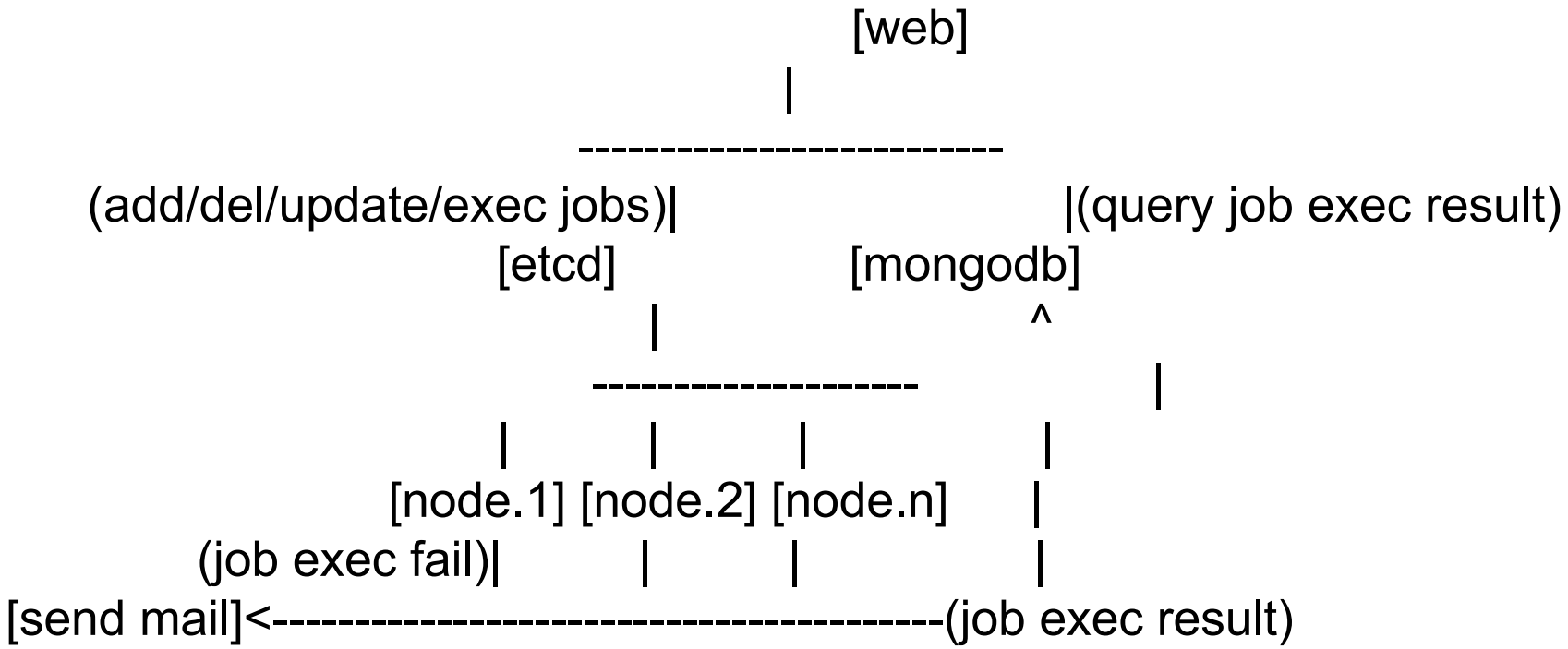
我眼里的“西施”

1. 可替代 cron
2. 分布式、高可用
3. 支持多种任务属性
4. 易用
5. 易部署

03 Part Three

cronsun

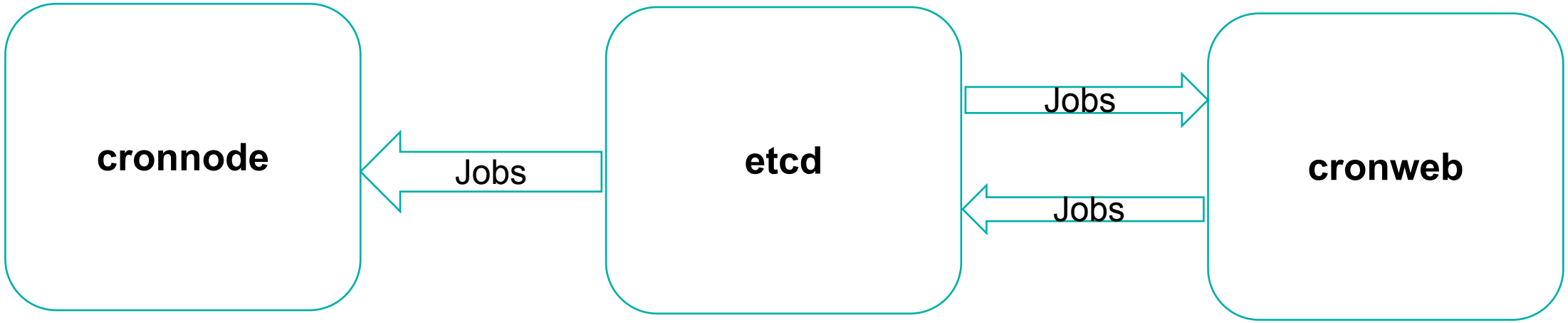
cronsun 架构



cronsun 特性

1. 部署简单
2. Web 界面统一管理任务
3. 任务失败重试
4. 任务失败邮件提醒
5. 多机单任务(防止单机挂掉任务不按时执行)
6. 单机任务并行数限制
7. 执行单次任务
8. 多机器严格的时间间隔任务
9. 支持安全性配置，可以限制任务脚本的后缀和执行用户
10. ...

cronsun 主要组件



cronnode

1. 节点可以进行分组(label)
2. 节点的状态
   ① 正常结点
   ② 故障节点
   ③ 离线节点

cronweb

1. 管理任务
2. 查询任务执行结果

任务类型

1. 普通任务，和 crontab 中的任务一样。
2. 单机单进程任务，普通的 crontab 任务是单机的，如果执行任务的机器出现问题，任务可能执行失败。cronsun 提供此任务类型是保证有多台机器可以执行一个任务，但在执行任务被执行时，只有一台机器在执行任务。
3. 一个任务执行间隔内允许执行一次，这个类型的任务和单机单进程任务类似，但限制更严格。因为多台机器间，时间可能会不一致，而某些任务要求执行的时间间隔要严格一致时，可以考虑采取这种类型。

任务属性

1. 失败重试
2. 超时设置
3. 安全设置
4. 同时执行任务数设置
5. 分组设置

任务定时器

```
0  1   2   3   4   5
|   |   |   |   |   |
|   |   |   |   |    +------ day of week (0 - 6) (Sunday=0)
|   |   |   |   +------ month (1 - 12)
|   |   |   +-------- day of month (1 - 31)
|   |   +---------- hour (0 - 23)
|   +------------ min (0 - 59)
+-------------- sec (0-59)
```

@yearly 、@monthly、@weekly、@daily和@hourly

cronsun 应用

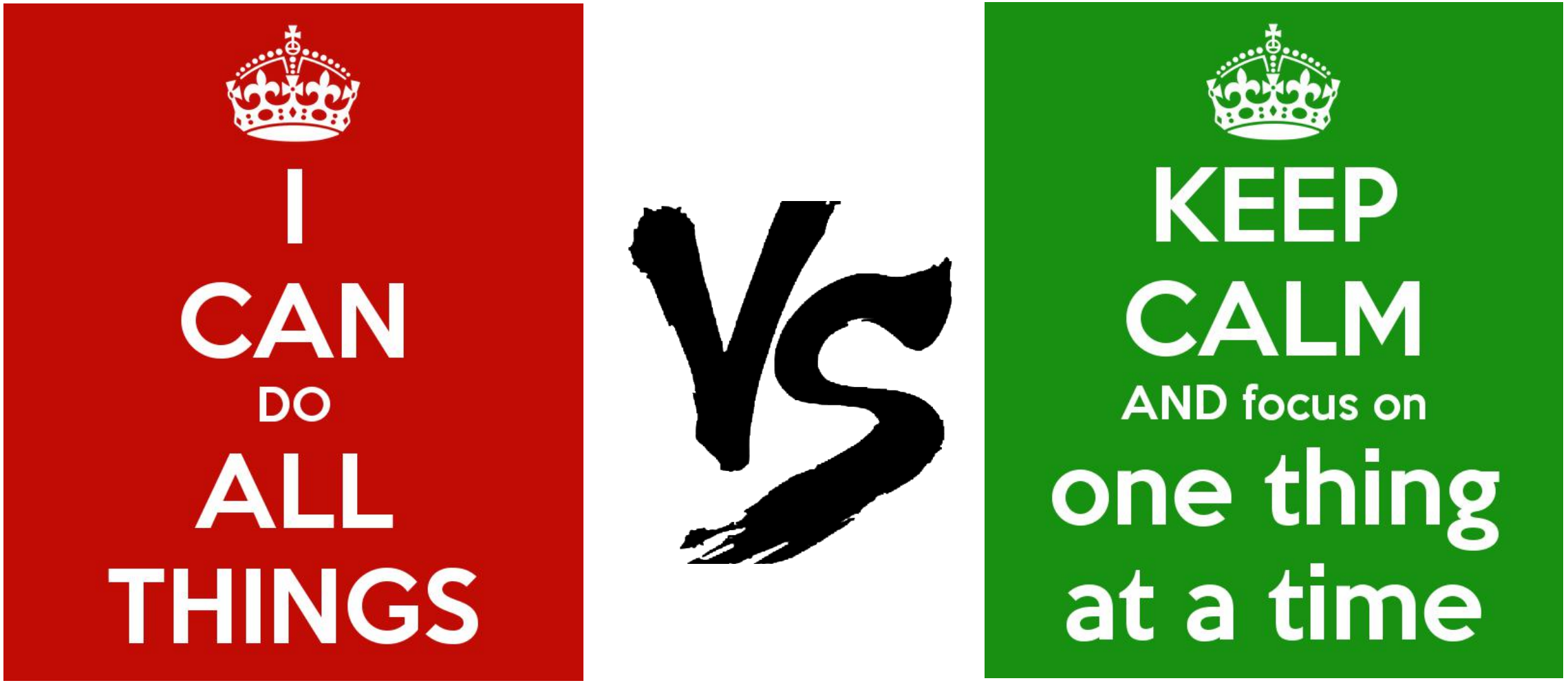
1. 替换 crontab
2. 执行不能单点失败的任务
3. 简单的任务调度

04 Part Four

心得体会

方案选择

目标定位



timer 的使用1

```
select {
    case m := <-c:
        handle(m)
    case <-time.After(1 * time.Second):
        // timed out
}
```

timer 的使用2

```
for {
        select {
        case m := <-c:
                handle(m)
        case <-time.After(1 * time.Second):
                // timed out
        }
}
```

timer 的使用3

```
t := time.NewTimer(1 * time.Second)
for {
        select {
        case m := <-c:
                handle(m)
        case <-t.C::
                // timed out
                t.Reset(1 * time.Second)
        }
}
```

timer 的使用4

```
for {
        select {
        case m := <-c:
                handle(m)
        case <-time.After(1 * time.Second):
                // timed out
        }
}
```

```
t := time.NewTimer(1 * time.Second)
for {
        select {
        case m := <-c:
                handle(m)
        case <-t.C::
                // timed out
                t.Reset(1 * time.Second)
        }
}
```

timer 的使用5

```
t := time.NewTimer(1 * time.Second)
for {
        select {
        case m := <-c:
                handle(m)
        case <-t.C::
                // timed out
        }
        t.Reset(1 * time.Second)
}
```

goroutine 退出1

```
func main() {
    var wg sync.WaitGroup
    wg.Add(1)
    go dosomething()
    wg.Add(1)
    go dosomething()
    wg.Wait()
    fmt.Println("Done")
}
```

```
func dosomething() {
    // do some thing
    wg.Done()
}
```

goroutine 退出2

```
func main() {
    var wg sync.WaitGroup
    wg.Add(1)
    go dosomething()
    wg.Add(1)
    go dosomething()
    waitForStop()
}
```

goroutine 退出3

```
func waitForStop(c, wg) {
    // wait for stop signal
    close(c)
    wg.Wait()
    fmt.Println("Done")
}
```

```
func dosomething(c, wg) {
    for {
        case <-c:
            wg.Done()
            return
        default:
            // do some thing
    }
}
```

goroutine 退出4

```
func main() {
    var wg sync.WaitGroup
    wg.Add(1)
    go dosomething1()
    wg.Add(1)
    go dosomething2()
    waitForStop()
}
```

goroutine 退出4

```
func waitForStop(c, wg) {
    // wait for stop signal
    close(c)
    wg.Wait()
    fmt.Println("Done")
}
```

```
func dosomething1(c, c1, wg) {
    for {
        case <-c:
            wg.Done()
            close(c1)
            return
        default:
            // do some thing
    }
}
```

```
func dosomething2(c1, wg)
{
    for {
        case <-c1:
            wg.Done()
            return
        default:
            // do some thing
    }
}
```

开发语言

为什么选择 Go？

简单、开发效率高

执行效率也不错

1. 两个人
2. 主要工作之余
3. 一个月
4. 第一个可用版

部署简单，只需一个二进制文件，无依赖

清清爽爽，干干净净

适合自己的才是最好的

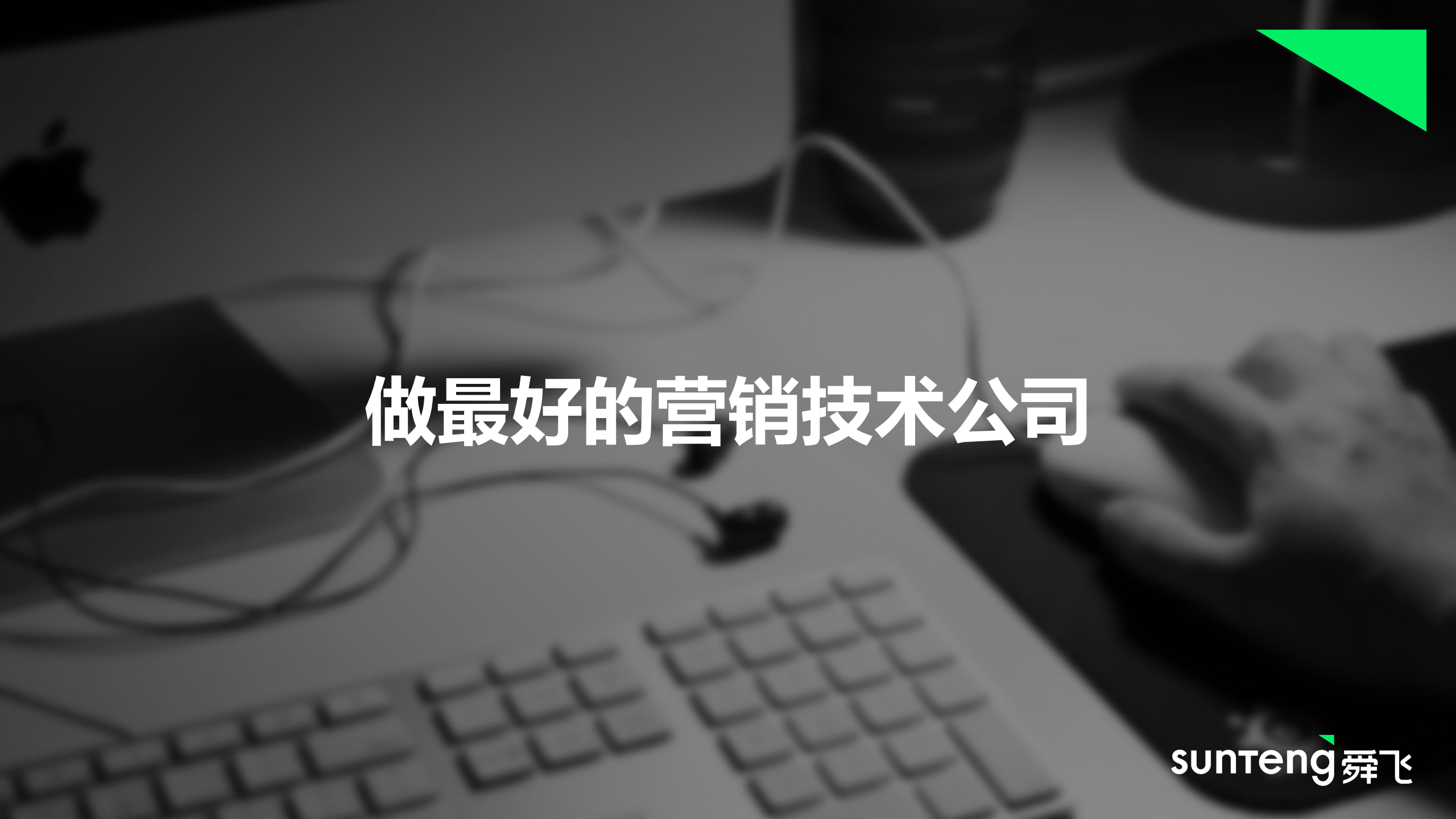
愿程序员的世界里没有纷争

感谢

Q&A

https://github.com/shunfei/cronsun

jobs@sunteng.com


做最好的营销技术公司