调度单个cron

- 这写代码的老师都喜欢用var 声明变量后使用
- 下次调度时间: nextTime = expr.Next(now)
- 超时时间计算：nextTime.Sub(now)
- 超时后执行回调函数：time.AfterFunc

```
//  【github.com/gorhill/cronexpr】 这个包功能对应Linux的定时任务;但是这个包支持到秒级
/*
    秒---> 0-59
    分---> 0-59
    时---> 0-23  0表示0点
    日---> 1-31
    月---> 1-12
    周---> 0-6   0表示周日
    年---> 1-12
*/
```

```
package main

import (
    "fmt"
    "time"

    "github.com/gorhill/cronexpr"
)
func main() {

    var (
        err      error
        expr     *cronexpr.Expression
        now      time.Time // 当前时间
        nextTime time.Time // 下次调度时间
    )

    // 每5s执行一次
    if expr, err = cronexpr.Parse("*/5 * * * * * *"); err != nil {
        fmt.Println(err)
        return
    }

    // 当前时间
    now = time.Now()

    // 下次调度时间 [时间规律是0,6,12,18,...,58],而不是根据启动时间计算的
    nextTime = expr.Next(now)

    // 等待这定时器超时
    // 超时时间计算：nextTime.Sub(now)
    time.AfterFunc(nextTime.Sub(now), func() {
        // 超时就会执行这回调函数
        fmt.Println("被调度了", nextTime)
    })
```

```
    time.Sleep(6 * time.Second)
}
```

调度多个cron任务

```
package main

import (
    "fmt"
    "time"

    "github.com/gorhill/cronexpr"
)

// 任务结构体--封装
type CronJob struct {
    expr     *cronexpr.Expression // 时间表达式
    nextTime time.Time            // 下次调度时间
}

// 声明一些全局变量
var (
    cronJob       *CronJob              // job名字
    expr          *cronexpr.Expression // 时间表达式
    now           time.Time             // 当前时间
    scheduleTable map[string]*CronJob  //key：任务名字, 调度表 存放所有定时任务
)

func main() {

    // 需要有一个调度协程,让他定时检测所有的Corn任务,谁过期了就执行谁

    // 当前时间
    now = time.Now()

    // 创建内存,map不可以直接使用,这里不给分配内存
    scheduleTable = make(map[string]*CronJob)

    // 定义两个cornjob:  每5s执行一次;  每57执行一次
    expr = cronexpr.MustParse("*/5 * * * * * *")
    // 将调度器CronJob初始化且赋值给cronJob
    cronJob = &CronJob{
        expr:     expr,
        nextTime: expr.Next(now),
    }

    // 任务注册到调度表
    scheduleTable["job1"] = cronJob

    // 定义两个cornjob:  每5s执行一次;  每57执行一次
    expr = cronexpr.MustParse("*/5 * * * * * *")
    // 将调度器CronJob初始化且赋值给cronJob
    cronJob = &CronJob{
        expr:     expr,
        nextTime: expr.Next(now),
    }
```

```
    // 任务注册到调度表
    scheduleTable["job2"] = cronJob

    // 启动调用协程
    go func() {
        var (
            jobName string
            cronJob *CronJob
            now     time.Time
        )
        // 定时检查一下调度任务
        for {
            now = time.Now()
            // 判断是否过期（如果下次调度时间小于等于当前时间，说明已经过期了）
            for jobName, cronJob = range scheduleTable {
                if cronJob.nextTime.Before(now) || cronJob.nextTime.Equal(now) {
                    // 启动一个协程,执行这任务
                    go func(jobName string) {
                        fmt.Println("执行：", jobName)
                    }(jobName)

                    // 计算下一次调度时间
                    cronJob.nextTime = cronJob.expr.Next(now)
                    fmt.Println(jobName, "下次执行时间是：", cronJob.nextTime)
                }
            }
            // 睡眠100 毫秒（不让它占用过多cpu）
            select {
            case <-time.NewTimer(100 * time.Millisecond).C: //将在100 毫秒可读，返
回
            }

            // 简单的  time.Sleep(100 *time.Second)
        }
    }()

    time.Sleep(100 * time.Second)
}
```

```
# 实际效果
job1 下次执行时间是：  2022-03-15 00:54:45 +0800 CST
执行：  job1
job2 下次执行时间是：  2022-03-15 00:54:45 +0800 CST
执行：  job2
执行：  job1
job1 下次执行时间是：  2022-03-15 00:54:50 +0800 CST
job2 下次执行时间是：  2022-03-15 00:54:50 +0800 CST
执行：  job2
job1 下次执行时间是：  2022-03-15 00:54:55 +0800 CST
job2 下次执行时间是：  2022-03-15 00:54:55 +0800 CST
执行：  job1
执行：  job2
执行：  job1
job1 下次执行时间是：  2022-03-15 00:55:00 +0800 CST
执行：  job2
job2 下次执行时间是：  2022-03-15 00:55:00 +0800 CST
```

```
执行： job1
job1 下次执行时间是： 2022-03-15 00:55:05 +0800 CST
job2 下次执行时间是： 2022-03-15 00:55:05 +0800 CST
```