

你的浏览器目前处于放大状态，会导致页面显示不正常，你可以键盘按“ctrl+数字0”组合键恢复初始状态。[不再显示](#)

CSIG学习K吧

已加入

文章分享

培训课件

团队活动

调查问卷

团队日历

视频

学习直达区

1.1 方案一 ...

1.2 方案二 ...

2. 代码实现

2.1 结构体

2.1.1 首...

2.1.2 需...

2.1.3 需...

2.2 启动时...

2.3 向时间...

2.4 从时间...

2.5 到期执...

3. 总结

34日 12:06

浏览(36)

收藏(0)

评论(3)

分享

关于作者

damonkli(李凯)

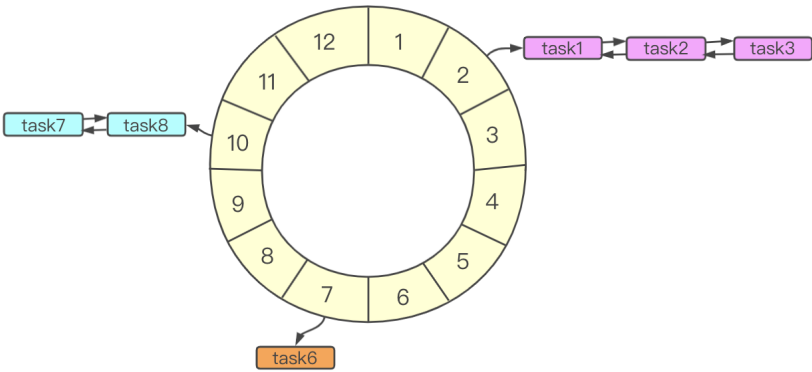
CSIG\云技术运营服务部\工具建设三...

作者文章

• 手把手教你用go实现一个简单timewheel时间轮

1. timewheel简介

timewheel时间轮盘，简单理解就是一个时钟表盘，指针每隔一段时间前进一格，走完一圈是一个周期。而需要执行的任务就放置在表盘的刻度处，当指针走到该位置时，就执行相应的任务。该算法的运用其实是非常的广泛的，在 Netty、Akka、Quartz、ZooKeeper、Kafka 等组件中都存在时间轮的踪影。具体架构如下所示



时间轮是一个环形队列，底层实现就是一个固定长度的数组，数组中的每个元素存储一个双向链表，选择双向链表的原因是在O(1)时间复杂度实现插入和删除操作。而这个双向链表存储的就是要在该位置执行的任务。

举例分析，假设时间轮盘每隔1s前进一格，那么上图中的时间轮盘的周期就是12s，如果此时时间在刻度为0的位置，此时需要添加一个定时任务，需要10s后执行，那么该任务就需要放到刻度10处。当指针到达刻度10时，执行在该位置上，双向链表存储的所有任务。

此时你会有个以为，如果此时我想添加一个20s之后执行的任务，应该怎么添加呢？由于这个任务的延时超过了时间轮盘的周期(12s)，所以单个轮盘已经没法满足需求了，此时有如下两个解决方案

<https://km.woa.com/group/b2training/articles/show/476506>

目录

1. timewhee...	多级时间轮
1.1 方案一 ...	
1.2 方案二 ...	
2. 代码实现	，更贴近了钟表，钟表分为时针、分针、秒针。每过60s，分针前进一格，
2.1 结构体	前进一格。多级时间轮的理念就是分两级甚至更多级轮盘，当一级轮盘走一
2.1.1 首...	进一格，二级轮盘走一圈后，三级轮盘前进一格，依次类推~
2.1.2 需...	
2.1.3 需...	
2.2 启动时...	
2.3 向时间...	cle的参数，例如某个任务需要20s之后执行，那么该任务的circle=1，计算
2.4 从时间...	4，也就是该任务需要在表盘走一圈以后，在位置4处执行。相应的，表盘指
2.5 到期执...	处的任务列表中，所有的任务的circle都-1。如果该处的任务circle==0，那
3. 总结	

本文选取该种万案来实现。

2. 代码实现

接下来从代码层面来介绍整体的实现过程。

2.1 结构体

2.1.1 首先需要有一个TimeWheel的结构体来存储时间轮盘的相关信息

- 1. interval: 时间轮盘的精度，也就是时间轮盘每前进一步，所需要的时间
- 2. slotNums: 时间轮盘总的齿轮数，interval*slotNums就是时间轮盘走一周所花的时间
- 3. currentPos: 时间轮盘指针当前的位置
- 4. ticker: 时钟计时器，定时触发。
- 5. slots: 利用数组来实现时间轮盘，数组中每个元素是个双向链表，来存储要执行的任务
- 6. taskRecords: 针对任务的一个map表，key是任务的key，value是任务对象
- 7. isRunning: 时间轮盘是否是running状态，避免重复启动

```
type TimeWheel struct {
    // 时间轮盘的精度
    interval time.Duration

    // 时间轮盘的齿轮数 interval*slotNums就是时间轮盘转一圈走过的时间
    slotNums int

    // 时间轮盘当前的位置
    currentPos int

    ticker *time.Ticker

    // 时间轮盘每个位置存储的Task列表
    slots []*list.List

    taskRecords *sync.Map

    isRunning bool
}
```

目录

到时间以后执行的Job

1. timewhee...

1 1 六室一

type Job func(interface{})

2. 代码实现

2.1 结构体

执行任务的结构体

2.1.1 首...

2.1.2 需... 标识，必须是唯一的

2.1.3 需... 隔多长时间执行

2.2 启动时... 任务的创建时间

2.3 向时间... 轮盘的存储位置，也就是TimeWheel.slots中的存储位置

2.4 从时间... 任务需要执行的Job

2.5 到期执... 要执行的次数，如果需要一直执行，设置成<0的数

3. 总结

// 时间轮盘上需要执行的任务

```
type Task struct {  
    // 用来标识task对象，是唯一的  
    key interface{}  
    // 任务周期  
    interval time.Duration  
    // 任务的创建时间  
    createTime time.Time  
    // 任务在轮盘的位置  
    pos int  
    // 任务需要在轮盘走多少圈才能执行  
    circle int  
    // 任务需要执行的Job，优先级高于TimeWheel中的Job  
    job Job  
    // 任务需要执行的次数，如果需要一直循环执行，设置成<0的数  
    times int  
}
```

2.2 启动时间轮

需要一个函数来启动时间轮盘，该函数需要启动一个线程来执行。启动以后，利用时间定时器，每隔固定时间(TimeWheel.Inteval)，时间轮盘前进一格。

```
// Start 启动时间轮盘  
func (tw *TimeWheel) Start() {  
    tw.ticker = time.NewTicker(tw.interval)  
    go tw.start()  
    tw.isRunning = true  
}
```

// 启动时间轮盘的内部函数

```
func (tw *TimeWheel) start() {  
    for {  
        -  
        -
```

```

        select {
            case <-tw.ticker.C:
                // 通过runTask函数来检查当前需要执行的任务
                tw.checkAndRunTask()
        }
    }
}

```

2.1 结构体

2.1.1 首...

2.1.2 需...

2.1.3 需...

2.2 启动时...

2.3 向时间...

2.4 从时间...

2.5 到期执...

3. 总结

添加任务

务的时候，需要计算任务的pos和circle，也就是在时间轮盘的位置和需要走了两种方式了，一种是基于createdTime来计算，主要用在服务刚刚启动或时候使用。一种是基于任务的周期来计算下次需要执行的时间，这种是在重建计算和添加任务的时候使用。第一种是为了避免在服务重启的时候，相配在同一个位置执行。从而加大任务执行时候的压力。

在计算完pos和circle以后，添加任务主要包括两部分，一个是在taskRecords这个map中存储一下task对象（主要用处是，在删除的时候，获取任务在timewheel表盘的位置），一个是根据计算的pos将task存储在timewheel对应位置的双向队列中（主要用处是在执行任务的时候，获取task需要执行的job）。

```

// 添加任务的内部函数
// @param task      Task   Task对象
// @param byInterval bool   生成Task在时间轮盘位置和圈数的方式，true表示利用
Task.interval来生成，false表示利用Task.createTime生成
func (tw *TimeWheel) addTask(task *Task, byInterval bool) {
    var pos, circle int
    if byInterval {
        pos, circle = tw.getPosAndCircleByInterval(task.interval)
    } else {
        pos, circle =
tw.getPosAndCircleByCreatedTime(task.createdTime, task.interval, task.key)
    }

    task.circle = circle
    task.pos = pos

    element := tw.slots[pos].PushBack(task)
    tw.taskRecords.Store(task.key, element)
}

// 该函数通过任务的周期来计算下次执行的位置和圈数
func (tw *TimeWheel) getPosAndCircleByInterval(d time.Duration) (int, int) {
    delaySeconds := int(d.Seconds())
    intervalSeconds := int(tw.interval.Seconds())
    circle := delaySeconds / intervalSeconds / tw.slotNums
    pos := (tw.currentPos + delaySeconds/intervalSeconds) % tw.slotNums

    // 特殊case，当计算的位置和当前位置重叠时，因为当前位置已经走过了，所以circle需
    要减一
    if pos == tw.currentPos && circle != 0 {

```

```

11 pos == tw.currentPos && circle != 0 {
    circle--
}
return pos, circle
}

// 该函数用任务的创建时间来计算下次执行的位置和圈数
func (tw *TimeWheel) getPosAndCircleByCreateTime(createdTime time.Time, d
time.Duration, key interface{}) (int, int) {

    passedTime := time.Since(createdTime)
    passedSeconds := int(passedTime.Seconds())
    delaySeconds := int(d.Seconds())
    intervalSeconds := int(tw.interval.Seconds())

    circle := delaySeconds / intervalSeconds / tw.slotNums
    pos := (tw.currentPos + (delaySeconds -
(passedSeconds%delaySeconds))/intervalSeconds) % tw.slotNums

    // 特殊case, 当计算的位置和当前位置重叠时, 因为当前位置已经走过了, 所以circle需
要减一
    if pos == tw.currentPos && circle != 0 {
        circle--
    }

    return pos, circle
}

```

2.4 从时间轮删除任务

删除任务需要两部分，一部分是从taskRecords中删除任务记录，一部分是从时间轮盘的双向链表中删除任务记录。

```

// 删除任务的内部函数
func (tw *TimeWheel) removeTask(task *Task) {
    // 从map结构中删除
    val, _ := tw.taskRecords.Load(task.key)
    tw.taskRecords.Delete(task.key)

    // 通过TimeWheel.slots获取任务的
    currentList := tw.slots[task.pos]
    currentList.Remove(val.(*list.Element))
}

```

2.5 到期执行任务

时间轮盘的currentPos记录当前的位置，当时间轮盘前进一格的时候，就会检查当前位置的双向链表，检查他们的circle是否为0，如果为0就执行任务，如果不为零，circle--。执行完成后，将任务从双向链表中删除。

目录

- 1. timewheel...
 - 1.1 方案一 ...
 - 1.2 方案二 ...

// 检查该轮盘点位上的Task，看哪个需要执行

```
func (tw *TimeWheel) checkAndRunTask() {

    // 获取该轮盘位置的双向链表
    currentList := tw.slots[tw.currentPos]

    if currentList != nil {
        for item := currentList.Front(); item != nil; {
            task := item.Value.(*Task)
            // 如果圈数>0，表示还没到执行时间，更新圈数
            if task.circle > 0 {
                task.circle--
                item = item.Next()
                continue
            }

            // 执行任务时，Task.job是第一优先级，然后是TimeWheel.job
            if task.job != nil {
                go task.job(task.key)
            } else {
                fmt.Println(fmt.Sprintf("The task %d don't
have job to run", task.key))
            }

            // 执行完成以后，将该任务从时间轮盘删除
            next := item.Next()
            tw.taskRecords.Delete(task.key)
            currentList.Remove(item)

            item = next

            // 重新添加任务到时间轮盘，用Task.interval来获取下一次执行的
            // 轮盘位置

            if task.times != 0 {
                if task.times < 0 {
                    tw.addTask(task, true)
                } else {
                    task.times--
                    tw.addTask(task, true)
                }
            } else {
                // 将任务从taskRecords中删除
                tw.taskRecords.Delete(task.key)
            }
        }
    }
}
```

```
}

// 轮盘前进一步
if tw.currentPos == tw.slotNums-1 {
    tw.currentPos = 0
} else {
    tw.currentPos++
}
}

2.1.2 需...
2.1.3 需...
2.2 启动时...
2.3 向时间...
2.4 从时间...
2.5 到期执... 司轮盘所需要的必要操作，更多的优化请参考我的github源代码
```

3. 总结 [/lk668/timewheel](#)。源代码丰富了以下操作：
- timewheel
- 2. 添加函数来获取timewheel的单例模式，从而保证timewheel的唯一性
 - 3. 添加检查timewheel是否running的函数
 - 4. 为TimeWheel结构体也添加了job参数，但是其优先级低于task，主要用于大部分task需要执行相同任务的情况，此时就没必要为每个task设置job。只需要为TimeWheel结构体设置一个job即可

参考

- <https://github.com/lk668/timewheel>
- <https://www.jianshu.com/p/cdaea7eadba8>
- <https://github.com/nosixtools/timewheel>
- <https://mp.weixin.qq.com/s/DcyXPGxXFYcXCQJII1INpg>

最后更新于 2021-07-04 12:06


本文申报

如果觉得我的文章对您有用，请随意赞赏



仅供内部学习与交流，未经公司授权切勿外传

分类：其他文章分享 标签：timewhee...(1) 时间轮盘(1) GoLang(1) 定时任务(1)



本文专属二维码，扫一扫还能分享朋友圈

想要微信公众号推广本公众号？[点击获取链接](#)

目录

1. timewhee...

1.1 方案一 ...

1.2 方案二 ...

我顶 (1)

收藏 (0)

2. 代码实现

选中文章内容可快速反馈

2.1 结构体

转载

收录

评论 (3)

反馈

2.1.1 首...

2.1.2 需...

2.1.3 需...

2021-07-04 12:30:31

2.2 启动时...

不能使用Go原生的timer?

2.3 向时间...

复 (2)

2.4 从时间...

2.5 到期执...

nkli (楼主)

2021-07-04 21:25:34

会比较多，每个任务的周期不一定是一样的，执行时间也不一定相同~

3. 总结

复



damonkli (楼主)

2021-07-04 21:27:10

所以需要一种数据结构来存储和执行任务~

回复



切换到更多功能

发表评论