33 | 固若金汤：限速器与错误处理

2022-12-24 郑建勋 来自北京

《Go进阶 · 分布式爬虫实战》

课程介绍 >

讲述：郑建勋

时长 11:44　大小 10.71M

你好，我是郑建勋。

我们前面的课程，由于一直都是不加限制地并发爬取目标网站，很容易导致被服务器封禁。为了能够正常稳定地访问服务器，我们这节课要给项目增加一个重要的功能：限速器。同时，我们还会介绍在 Go 中进行错误处理的最佳实践。

限速器

先来看限速器。很多情况下，不管你是想防止黑客的攻击，防止对资源的访问超过服务器的承载能力，亦或是防止在爬虫项目中被服务器封杀，我们都需要对服务进行限速。

在爬虫项目中，保持合适的速率也有利于我们稳定地爬取数据。大多数限速的机制是**令牌桶算法（Token Bucket）**来完成的。

令牌桶算法的原理很简单，我们可以想象这样一个场景，你去海底捞吃饭，里面只有 10 个座位，我们可以将这 10 个座位看作是桶的容量。现在，由于座位已经满了，服务员就帮我们叫了个号，我们随即进入到了等待的状态。

一桌客人吃完之后，下一位并不能马上就座，因为服务员还需要收拾饭桌。由于服务员的数量有限，因此即便很多桌客人同时吃完，也不能立即释放出所有的座位。如果每 5 分钟收拾好一桌，那么“1 桌 /5 分钟”就叫做令牌放入桶中的速率。轮到我们就餐时，我们占据了一个座位，也就是占据了一个令牌，这时我们就可以开吃了。

通过上面简化的案例能够看到，令牌桶算法通过控制桶的容量和令牌放入桶中的速率，保证了系统能在最大的处理容量下正常工作。在 Go 中，我们可以使用官方的限速器实现：golang.org/x/time/rate，它提供了一些简单好用的 API。

在 golang.org/x/time/rate 库中，类型 Limit 表示速率，代表每秒钟放入到桶中的令牌个数。NewLimiter 函数中，第一个参数传递的是 Limit 速率，第二个参数 b 表示桶的数量。除此之外，库中还有 Every 函数，它的参数是两个令牌之间的时间间隔，它会转化为对应的 Limit 速率。

复制代码

```
// Limit defines the maximum frequency of some events.  Limit is
// represented as number of events per second.  A zero Limit allows no
// events.
type Limit float64

// NewLimiter returns a new Limiter that allows events up to rate r
// and permits bursts of at most b tokens.
func NewLimiter(r Limit, b int) *Limiter

// Every converts a minimum time interval between events to a Limit.
func Every(interval time.Duration) Limit
```

生成了 Limiter 之后，我们一般会调用 Limiter 的 Wait 方法等待可用的令牌。其中，参数 ctx 可以设置超时退出的时间，它可以避免协程一直陷在堵塞状态中。

复制代码

```
func (lim *Limiter) Wait(ctx context.Context) (err error) {
  return lim.WaitN(ctx, 1)
}
```

如果没有可用的令牌，当前协程会陷入到堵塞状态。我们用一个例子来说明一下 Limiter 的使用方法。

在下面的例子中，rate.NewLimiter 生成了一个限速器，其中桶的最大容量为 2，rate.Limit(1) 表示每隔 1 秒钟向桶中放入 1 个令牌。


```
package main

import (
  "context"
  "fmt"
  "golang.org/x/time/rate"
  "time"
)

func main() {
  limit := rate.NewLimiter(rate.Limit(1), 2)
  for {
    if err := limit.Wait(context.Background()); err == nil {
      fmt.Println(time.Now().Format("2006-01-02 15:04:05"))
    }
  }
}
```

我们用一个 for 循环打印当前时间会发现，前两次打印是在同一秒，这是因为桶中一开始有两个令牌可以用。之后，每一次打印都需要间隔一秒，因为每隔一秒钟才会往桶中填充一个令牌。


```
2022-11-22 22:31:51
2022-11-22 22:31:51
2022-11-22 22:31:52
2022-11-22 22:31:53
2022-11-22 22:31:54
```

我们再尝试使用 rate.Every 来生成 Limit 速率：


```
func main() {
  limit := rate.NewLimiter(rate.Every(500*time.Millisecond), 2)
```

```
  for {
    if err := limit.Wait(context.Background()); err == nil {
      fmt.Println(time.Now().Format("2006-01-02 15:04:05"))
    }
  }
}
```


其中，rate.Every(500*time.Millisecond) 表示每 500 毫秒放入一个令牌，换算过来就是每秒钟放入 2 个令牌。

所以我们可以看到，现在每秒钟都会输出两条记录：

复制代码

```
2022-11-22 22:39:28
2022-11-22 22:39:28
2022-11-22 22:39:29
2022-11-22 22:39:29
2022-11-22 22:39:30
2022-11-22 22:39:30
```

刚才我们已经看到了限速器 Limiter 的基本用法。但有时候我们还会有一些更复杂的需求，例如有多层限速器的需求（细粒度限速器限制每秒的请求，粗粒度限速器限制每分钟、每小时或每天的请求）。

假设我们的爬虫项目希望每分钟只能够访问 10 次目标网站，但是只有每分钟的限制是不够的。因为这样我们可能会一秒钟直接访问 10 次，这样服务器就能直接检测出我们是爬虫机器人了。所以，我们还需要控制一下瞬时的请求量，例如每秒钟访问的频率不超过 0.5 次。这里我也借鉴了《Concurrency in Go》中多层限速器的设计，在新的 Limiter 包中将限速器抽象为了 RateLimiter 接口，golang.org/x/time/rate实现的 Limiter 自动就实现了该接口：

复制代码

```
package limiter

type RateLimiter interface {
  Wait(context.Context) error
  Limit() rate.Limit
}
```

多层限速器对应的 multiLimiter 如下：


```
type multiLimiter struct {
  limiters []RateLimiter
}

func MultiLimiter(limiters ...RateLimiter) *multiLimiter {
  byLimit := func(i, j int) bool {
    return limiters[i].Limit() < limiters[j].Limit()
  }
  sort.Slice(limiters, byLimit)
  return &multiLimiter{limiters: limiters}
}

func (l *multiLimiter) Wait(ctx context.Context) error {
  for _, l := range l.limiters {
    if err := l.Wait(ctx); err != nil {
      return err
    }
  }
  return nil
}

func (l *multiLimiter) Limit() rate.Limit {
  return l.limiters[0].Limit()
}
```

其中，MultiLimiter 函数用于聚合多个 RateLimiter，并将速率由小到大排序。Wait 方法会循环遍历多层限速器 multiLimiter 中所有的限速器并索要令牌，只有当所有的限速器规则都满足后，才会正常执行后续的操作。

最后，我们生成多层限速器并把它放入爬虫任务 Task 中，每一个爬虫任务可能有不同的限速。这里生成速率的函数用 Per 进行了封装，例如 limiter.Per(20, 1*time.Minute) 代表速率是每 1 分钟补充 20 个令牌。


```
func main(){
  //2秒钟1个
  secondLimit := rate.NewLimiter(limiter.Per(1, 2*time.Second), 1)
  //60秒20个
  minuteLimit := rate.NewLimiter(limiter.Per(20, 1*time.Minute), 20)
  multiLimiter := limiter.MultiLimiter(secondLimit, minuteLimit)
```

```
  seeds := make([]*collect.Task, 0, 1000)
  seeds = append(seeds, &collect.Task{
    Property: collect.Property{
      Name: "douban_book_list",
    },
    Fetcher: f,
    Storage: storage,
    Limit:   multiLimiter,
  })
}

func Per(eventCount int, duration time.Duration) rate.Limit {
  return rate.Every(duration / time.Duration(eventCount))
}

```


到这里，我们就不再需要在爬取数据时固定休眠了，只要使用限速器来控制速度的就可以了。

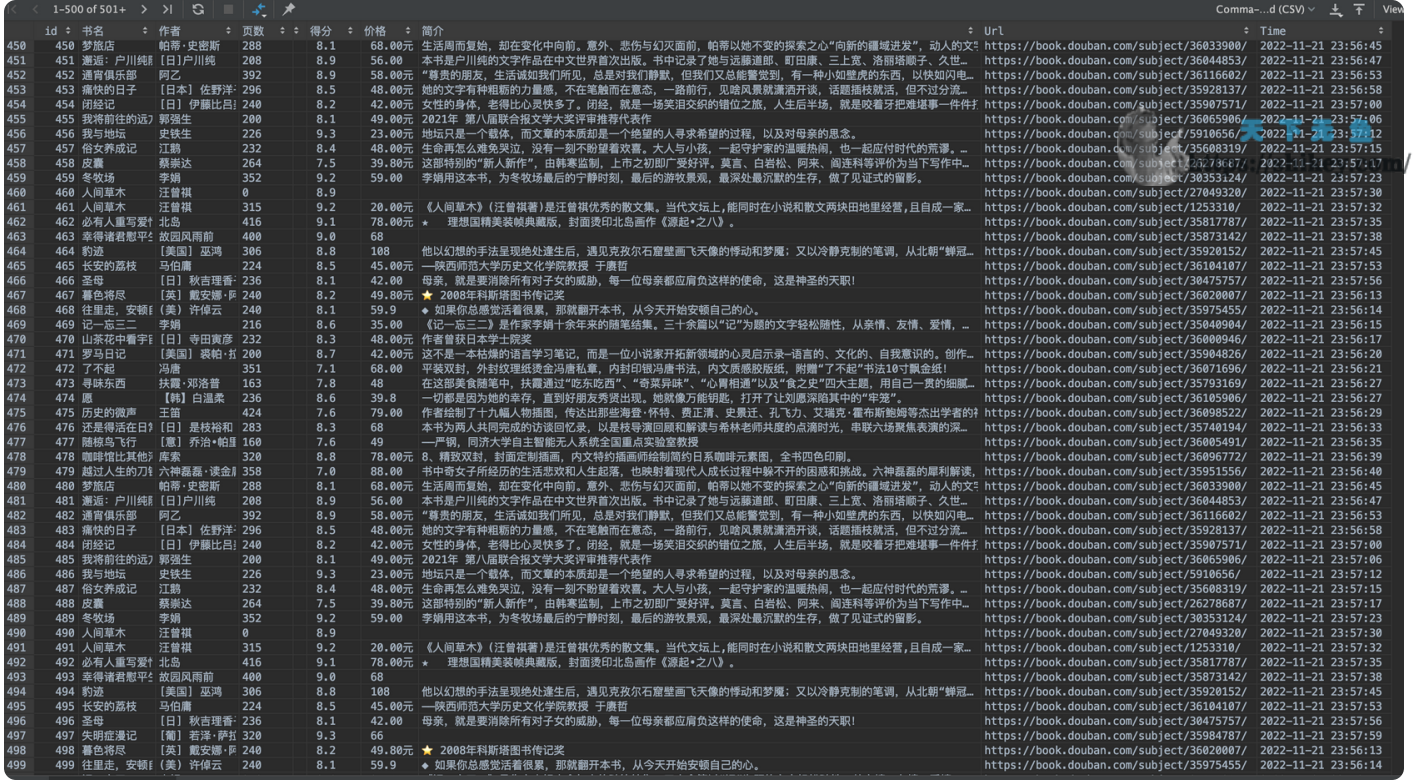
随机休眠

不过，如果使用多层限速器，我们访问服务器的频率会过于稳定。为了模拟人类的行为，我们还可以在限速器的基础上，增加随机休眠。

如下所示，假设设置的 r.Task.WaitTime 为 2 秒，我们这里使用了随机数，获取 0-2000 毫秒之间的任何一个整数作为休眠时间。


```
func (r *Request) Fetch() ([]byte, error) {
  if err := r.Task.Limit.Wait(context.Background()); err != nil {
    return nil, err
  }
  // 随机休眠，模拟人类行为
  sleeptime := rand.Int63n(r.Task.WaitTime * 1000)
  time.Sleep(time.Duration(sleeptime) * time.Millisecond)
  return r.Task.Fetcher.Get(r)
}
```

下面是我通过多层限速器和随机休眠机制成功爬取到的图书数据：



	id	书名	作者	页数	得分	价格	简介	Url	Time
450	450	梦旅店	帕蒂·史密斯	288	8.1	68.00元	生活周而复始，却在变化中向前。意外、悲伤与幻灭面前，帕蒂以她不变的探索之心“向新的疆域进发”，动人的文	https://book.douban.com/subject/36033900/	2022-11-21 23:56:45
451	451	邂逅：户川纯	[日]户川纯	208	8.9	56.00	本书是户川纯的文字作品在中文世界首次出版。书中记录了她与远藤道郎、町田康、三上宽、洛丽塔顺子、久世…	https://book.douban.com/subject/36044853/	2022-11-21 23:56:47
452	452	通宵俱乐部	阿乙	392	8.9	58.00元	“尊贵的朋友，生活诚如我们所见，总是对我们静默，但我们又总能警觉到，有一种小如壁虎的东西，以快如闪电…	https://book.douban.com/subject/36116602/	2022-11-21 23:56:53
453	453	痛快的日子	[日本] 佐野洋·	296	8.5	48.00元	她的文字有种粗砺的力量感，不在笔触而在意态，一路前行，见啥风景就潇洒开谈，话题插枝就活，但不过分流…	https://book.douban.com/subject/35928137/	2022-11-21 23:56:58
454	454	闭经记	[日] 伊藤比吕	240	8.2	42.00元	女性的身体，老得比心灵快多了。闭经，就是一场笑泪交织的错位之旅，人生后半场，就是咬着牙把难堪事一件件	https://book.douban.com/subject/35907571/	2022-11-21 23:57:00
455	455	我将前往的远方	郭强生	200	8.1	49.00元	2021年 第八届联合报文学大奖评审推荐代表作	https://book.douban.com/subject/36065906/	2022-11-21 23:57:06
456	456	我与地坛	史铁生	226	9.3	23.00元	地坛只是一个载体，而文章的本质却是一个绝望的人寻求希望的过程，以及对母亲的思念。	https://book.douban.com/subject/5910656/	2022-11-21 23:57:12
457	457	俗女养成记	江鹏	232	8.4	48.00元	生命再怎么难免哭泣，没有一刻不盼望着欢喜。大人与小孩，一起守护家的温暖热闹，也一起应付时代的荒谬。…	https://book.douban.com/subject/35608319/	2022-11-21 23:57:15
458	458	皮囊	蔡崇达	264	7.5	39.80元	这部特别的“新人新作”，由韩寒监制，上市之初即广受好评。莫言、白岩松、阿来、阎连科等评价为当下写作中…	https://book.douban.com/subject/26278687/	2022-11-21 23:57:17
459	459	冬牧场	李娟	352	9.2	59.00	李娟用这本书，为冬牧场最后的宁静时刻，最后的游牧景观，最深处最沉默的生存，做了见证式的留影。	https://book.douban.com/subject/30353124/	2022-11-21 23:57:23
460	460	人间草木	汪曾祺	0	8.9			https://book.douban.com/subject/27049320/	2022-11-21 23:57:30
461	461	人间草木	汪曾祺	315	9.2	20.00元	《人间草木》(汪曾祺著)是汪曾祺优秀的散文集。当代文坛上,能同时在小说和散文两块田地里经营,且自成一家…	https://book.douban.com/subject/1253310/	2022-11-21 23:57:32
462	462	必有人重写爱情	北岛	416	9.1	78.00元	★ 理想国精美装帧典藏版，封面烫印北岛画作《源起•之八》。	https://book.douban.com/subject/35817787/	2022-11-21 23:57:35
463	463	幸得诸君慰平生	故园风雨前	400	9.0	68		https://book.douban.com/subject/35873142/	2022-11-21 23:57:38
464	464	豹迹	[美国] 巫鸿	306	8.8	108	他以幻想的手法呈现绝处逢生后，遇见克孜尔石窟壁画飞天像的悸动和梦魇；又以冷静克制的笔调，从北朝“蝉冠…	https://book.douban.com/subject/35920152/	2022-11-21 23:57:45
465	465	长安的荔枝	马伯庸	224	8.5	45.00元	—陕西师范大学历史文化学院教授 于赓哲	https://book.douban.com/subject/36104107/	2022-11-21 23:57:53
466	466	圣母	[日] 秋吉理香·	236	8.1	42.00	母亲，就是要消除所有对子女的威胁，每一位母亲都应肩负这样的使命，这是神圣的天职！	https://book.douban.com/subject/30475757/	2022-11-21 23:57:56
467	467	暮色将尽	[英] 戴安娜·珀	240	8.2	49.80元	★ 2008年科斯塔图书传记奖	https://book.douban.com/subject/36020007/	2022-11-21 23:56:13
468	468	往里走，安顿	(美) 许倬云	240	8.1	59.9	◆ 如果你总感觉活着很累，那就翻开本书，从今天开始安顿自己的心。	https://book.douban.com/subject/35975455/	2022-11-21 23:56:14
469	469	记一忘三二	李娟	216	8.6	35.00	《记一忘三二》是作家李娟十余年来的随笔结集。三十余篇以“记”为题的文字轻松随性，从亲情、友情、爱情，…	https://book.douban.com/subject/35040904/	2022-11-21 23:56:15
470	470	山茶花中看字	[日] 寺田寅彦	232	8.3	48.00元	作者曾获日本学士院奖	https://book.douban.com/subject/36000946/	2022-11-21 23:56:17
471	471	罗马日记	[美国] 裘帕·拉	200	8.7	42.00元	这不是一本枯燥的语言学习笔记，而是一位小说家开拓新领域的心灵启示录—语言的、文化的、自我意识的。创作…	https://book.douban.com/subject/35904826/	2022-11-21 23:56:20
472	472	了不起	冯唐	351	7.1	68.00	平装双封，外封纹理纸烫金冯唐私章，内封印银冯唐书法，内文质感胶版纸，附赠“了不起”书法10寸飘金纸！	https://book.douban.com/subject/36071696/	2022-11-21 23:56:21
473	473	寻味东西	扶霞·邓洛普	163	7.8	48	在这部美食随笔中，扶霞通过“吃东吃西”、“奇菜异味”、“心胃相通”以及“食之史”四大主题，用自己一贯的细腻…	https://book.douban.com/subject/35793169/	2022-11-21 23:56:27
474	474	愿	【韩】白温柔	236	8.6	39.8	一切都是因为她的幸存，直到好朋友秀贤出现。她就像万能钥匙，打开了让刘愿深陷其中的“牢笼”。	https://book.douban.com/subject/36105906/	2022-11-21 23:56:27
475	475	历史的微声	王笛	424	7.6	79.00	作者绘制了十九幅人物插图，传达出那些海登·怀特、费正清、史景迁、孔飞力、艾瑞克·霍布斯鲍姆等杰出学者的…	https://book.douban.com/subject/36098522/	2022-11-21 23:56:29
476	476	还是得活在日常	[日] 是枝裕和	283	8.3	68	本书为两人共同完成的访谈回忆录，以是枝导演回顾和解读与希林老师共度的点滴时光，串联六场聚焦表演的深…	https://book.douban.com/subject/35740194/	2022-11-21 23:56:33
477	477	随椋鸟飞行	[意] 乔治•帕里	160	7.6	49	—严钢，同济大学自主智能无人系统全国重点实验室教授	https://book.douban.com/subject/36005491/	2022-11-21 23:56:35
478	478	咖啡馆比其他	库索	320	8.8	78.00元	8、精致双封，封面定制插画，内文特约插画师绘制简约日系咖啡元素图，全书四色印刷。	https://book.douban.com/subject/36096772/	2022-11-21 23:56:39
479	479	越过人生的刀	六神磊磊·读金	358	7.0	88.00	书中奇女子所经历的生活悲欢和人生起落，也映射着现代人成长过程中躲不开的困惑和挑战。六神磊磊的犀利解读，	https://book.douban.com/subject/35951556/	2022-11-21 23:56:40
480	480	梦旅店	帕蒂·史密斯	288	8.1	68.00元	生活周而复始，却在变化中向前。意外、悲伤与幻灭面前，帕蒂以她不变的探索之心“向新的疆域进发”，动人的文	https://book.douban.com/subject/36033900/	2022-11-21 23:56:45
481	481	邂逅：户川纯	[日]户川纯	208	8.9	56.00	本书是户川纯的文字作品在中文世界首次出版。书中记录了她与远藤道郎、町田康、三上宽、洛丽塔顺子、久世…	https://book.douban.com/subject/36044853/	2022-11-21 23:56:47
482	482	通宵俱乐部	阿乙	392	8.9	58.00元	“尊贵的朋友，生活诚如我们所见，总是对我们静默，但我们又总能警觉到，有一种小如壁虎的东西，以快如闪电…	https://book.douban.com/subject/36116602/	2022-11-21 23:56:53
483	483	痛快的日子	[日本] 佐野洋·	296	8.5	48.00元	她的文字有种粗砺的力量感，不在笔触而在意态，一路前行，见啥风景就潇洒开谈，话题插枝就活，但不过分流…	https://book.douban.com/subject/35928137/	2022-11-21 23:56:58
484	484	闭经记	[日] 伊藤比吕	240	8.2	42.00元	女性的身体，老得比心灵快多了。闭经，就是一场笑泪交织的错位之旅，人生后半场，就是咬着牙把难堪事一件件	https://book.douban.com/subject/35907571/	2022-11-21 23:57:00
485	485	我将前往的远方	郭强生	200	8.1	49.00元	2021年 第八届联合报文学大奖评审推荐代表作	https://book.douban.com/subject/36065906/	2022-11-21 23:57:06
486	486	我与地坛	史铁生	226	9.3	23.00元	地坛只是一个载体，而文章的本质却是一个绝望的人寻求希望的过程，以及对母亲的思念。	https://book.douban.com/subject/5910656/	2022-11-21 23:57:12
487	487	俗女养成记	江鹏	232	8.4	48.00元	生命再怎么难免哭泣，没有一刻不盼望着欢喜。大人与小孩，一起守护家的温暖热闹，也一起应付时代的荒谬。…	https://book.douban.com/subject/35608319/	2022-11-21 23:57:15
488	488	皮囊	蔡崇达	264	7.5	39.80元	这部特别的“新人新作”，由韩寒监制，上市之初即广受好评。莫言、白岩松、阿来、阎连科等评价为当下写作中…	https://book.douban.com/subject/26278687/	2022-11-21 23:57:17
489	489	冬牧场	李娟	352	9.2	59.00	李娟用这本书，为冬牧场最后的宁静时刻，最后的游牧景观，最深处最沉默的生存，做了见证式的留影。	https://book.douban.com/subject/30353124/	2022-11-21 23:57:23
490	490	人间草木	汪曾祺	0	8.9			https://book.douban.com/subject/27049320/	2022-11-21 23:57:30
491	491	人间草木	汪曾祺	315	9.2	20.00元	《人间草木》(汪曾祺著)是汪曾祺优秀的散文集。当代文坛上,能同时在小说和散文两块田地里经营,且自成一家…	https://book.douban.com/subject/1253310/	2022-11-21 23:57:32
492	492	必有人重写爱情	北岛	416	9.1	78.00元	★ 理想国精美装帧典藏版，封面烫印北岛画作《源起•之八》。	https://book.douban.com/subject/35817787/	2022-11-21 23:57:35
493	493	幸得诸君慰平生	故园风雨前	400	9.0	68		https://book.douban.com/subject/35873142/	2022-11-21 23:57:38
494	494	豹迹	[美国] 巫鸿	306	8.8	108	他以幻想的手法呈现绝处逢生后，遇见克孜尔石窟壁画飞天像的悸动和梦魇；又以冷静克制的笔调，从北朝“蝉冠…	https://book.douban.com/subject/35920152/	2022-11-21 23:57:45
495	495	长安的荔枝	马伯庸	224	8.5	45.00元	—陕西师范大学历史文化学院教授 于赓哲	https://book.douban.com/subject/36104107/	2022-11-21 23:57:53
496	496	圣母	[日] 秋吉理香·	236	8.1	42.00	母亲，就是要消除所有对子女的威胁，每一位母亲都应肩负这样的使命，这是神圣的天职！	https://book.douban.com/subject/30475757/	2022-11-21 23:57:56
497	497	失明症漫记	[葡] 若泽·萨拉	320	9.3	66		https://book.douban.com/subject/35984787/	2022-11-21 23:57:59
498	498	暮色将尽	[英] 戴安娜·珀	240	8.2	49.80元	★ 2008年科斯塔图书传记奖	https://book.douban.com/subject/36020007/	2022-11-21 23:56:13
499	499	往里走，安顿	(美) 许倬云	240	8.1	59.9	◆ 如果你总感觉活着很累，那就翻开本书，从今天开始安顿自己的心。	https://book.douban.com/subject/35975455/	2022-11-21 23:56:14

上述代码位于v0.2.8。

错误处理

介绍了限速器，我们再来看一看 Go 项目中另一个重要的话题：错误处理。在 Go 中，错误处理也是被人吐槽得比较多的地方。很多批评者根据自己过去的编程语言的经验，觉得在 Go 中有很多下面这样的错误处理语法。

复制代码

```
if err != nil{
  return err
}
```

确实，在日常实践中，代码常常会变得像下面这样，让人感觉不太优雅。

复制代码

```
func doSomething() error {
  if err := foo(); err != nil {
    return err
  }
  if err := bar(); err != nil {
    return err
  }
  if err := baz(); err != nil {
```

```
    return err
  }
  return nil
}
```

然而，Go 这种错误处理方式实际上是深思熟虑的结果，它也是有软件工程的经验作为指导的。像 Java 或 C++ 中错误处理的 try…catch 语句被实践证明面临可读性差、难以精准地处理错误等问题。而在 Go 语言中，错误处理的哲学是强制调用者检查错误，这就保证了代码的可读性和健壮性。

虽然在某些情况下，这确实使 Go 代码变得冗长，但幸运的是，我们可以使用一些技术来最大程度地降低错误处理的重复性。

基本的错误处理方式

由于 Go 中允许多返回值，因此最常见的错误处理方式是将函数的最后一个返回值作为 error 接口，接口中的 Error 方法返回错误的信息。

复制代码

```
type error interface {
    Error() string
}
```

这种方式会用 errors.New 函数来生成一个新的错误，其中参数为本次的错误信息。

复制代码

```
func (r *Request) Check() error {
  if r.Depth > r.Task.MaxDepth {
    return errors.New("max depth limit reached")
  }
  return nil
}
```

错误的信息需要清晰表明错误，方便定位问题，例如通过 `open not_here.txt: no such file or directory`，我们可以清楚地知道打开的文件不存在。

errors.New 的实现非常简单，它生成了一个内置的 errorString 结构体，而 errorString 则实现了 Error() 方法。

复制代码

```
func New(text string) error {
  return &errorString{text}
}

type errorString struct {
  s string
}

func (e *errorString) Error() string {
  return e.s
}
```

我们也可以采用 fmt.Errorf 来做一些格式化的输出：

复制代码

```
func (b BrowserFetch) Get(request *Request) ([]byte, error) {
  client := &http.Client{
    Timeout: b.Timeout,
  }
  if b.Proxy != nil {
    transport := http.DefaultTransport.(*http.Transport)
    transport.Proxy = b.Proxy
    client.Transport = transport
  }
  req, err := http.NewRequest("GET", request.Url, nil)
  if err != nil {
    return nil, fmt.Errorf("get url failed:%v", err)
  }
  ...
}
```

fmt.Errorf 的好处是，我们可以在之前错误的基础上，附加一些额外的错误信息。

由于接口的零值为 nil，因此在处理函数返回的错误时，我们要通过 error != nil 来判断是否有错误产生：

复制代码

```
func (s *Crawler) CreateWork() {
```

```
  for {
    ...
    body, err := req.Fetch()
    if err != nil {
      s.Logger.Error("can't fetch ",
        zap.Error(err),
        zap.String("url", req.Url),
      )
      s.SetFailure(req)
      continue
    }
}
```


此外，在进行错误处理时，我们还要谨慎地处理自定义的错误。我们之前在介绍接口时提到过，当接口为 nil 时，意味着接口内部的动态类型和动态数据都为 nil。所以下面 foo 函数返回的 err 不为 nil：


```
func foo() error {
  var err *os.PathError
  return err
}
func main() {
  err := foo()
  fmt.Println(err != nil) // true
}
```

另一个类似的例子也是如此：


```
func GenerateError(flag bool) error {
    var genErr StatusErr
    if flag {
        genErr = StatusErr{
            Status: NotFound,
        }
    }
    return genErr
}

func main() {
    err := GenerateError(true) // **true**
    fmt.Println(err != nil)
    err = GenerateError(false) // **true**
    fmt.Println(err != nil)
```

```
16 }
```

那我们应该怎么处理返回的错误呢？应该直接 return err 让上层区处理？还是应该先对错误进行一些特殊的处理?

在实践中我们经常看到一些对错误反复、无意义的处理，最典型的就是日志处理。我们当前的项目也存在这个问题，当爬取网站超时时，会在多个地方打印错误信息，这时常是多余的。

复制代码

```
1 {"level":"ERROR","ts":"2022-11-22T00:22:57.549+0800","caller":"collect/collect.
2 {"level":"ERROR","ts":"2022-11-22T00:22:57.549+0800","caller":"engine/schedule.
```

但如果我们只是 return err，又容易失去一些函数堆栈的关键信息，变得只知道最终的错误信息，不知道函数的调用链是如何触发这一问题的。其实，我们可以使用 fmt.Errorf 包装额外的错误信息来解决这一问题。

错误链处理方式

在实践中，我们可能还会遇到下面这样的特殊情况。例如 tasks 是任务的列表，我们希望所有任务都执行，即便前面任务执行失败也不退出。但是下面这样的写法就只能得到最后一个错误的信息：

复制代码

```go
func doSomething() error {
  var reserr error
  for _, task := range tasks {
    if err = f(task);err != nil{
      reserr = err
    }
  }

  return reserr
}
```

还有一种情况是希望能够检测到错误链中的某一类特定错误。举一个例子，foo 函数通过读取 Read 来读取文件中的信息，当读取到最后，Read 函数会返回特定的错误类型：io.EOF。而在一些场景需要检测特殊的错误类型来进行额外的逻辑处理。

```
func foo() error {
  f, _ := os.Open(file)
  if _, err = f.Read(b); err != nil {
    fmt.Println(err == io.EOF) // true
  }
}
```


要解决这个问题，我们首先可能会想到在 foo 函数中使用 fmt.Errorf 包裹信息，但是很快会发现，现在已经失去了 io.EOF 这个原始的错误类型。

其实，为了解决这类问题，Go 标准库为我们实现了类似错误包装的机制，它可以将多个错误组成一个错误链。要使用错误包装机制有一个非常简单的方法，那就是在使用 fmt.Errorf 的时候加入一个特殊的格式化符号 %w，在下面这段代码中， `fmt.Errorf("read failed,%w", err)` 将错误 err 进行了包装。

这时，我们可以通过 errors.Is 来判断当前错误中是否包含了原始的 io.EOF 错误，errors.Is 会遍历整个错误链并查找是否有相同的错误。

```
func foo() error {
  f, _ := os.Open(file)
  if _, err = f.Read(b); err != nil {
    warpErr := fmt.Errorf("read failed,%w", err)
    fmt.Println(errors.Is(warpErr, io.EOF)) // true
  }
}
```


另外，errors.Unwrap 还可以对错误进行解包，如下所示：

```
func foo() error {
  f, _ := os.Open(file)
  if _, err = f.Read(b); err != nil {
    warpErr := fmt.Errorf("read failed,%w", err)
    err = errors.Unwrap(warpErr)
    fmt.Println(err == io.EOF) //  true
  }
}
```

一般我们会使用 errors.Is 来判断错误链中是否包含了指定的错误，而不是直接使用下面的==来判断。

复制代码

```
if err == ErrSomething {
  ...
}
```

要注意，尽量不要在自己的 API 中返回 `syscall.ENOENT` 以及 `io.EOF` 这样的特殊类型。因为这通常意味着调用者需要依赖我们代码库当中定义的特定类型。这在标准库中没有问题，但是如果是第三方库返回了一个新的错误类型，或者使用 fmt.Errorf 等方式进行了包裹，这种相等关系在后续就不再成立了。同时，这种方式还可能带来不必要的包之间的依赖。

减少错误处理的实践

为了避免冗长的错误处理，我们可以使用一些最佳实践。

例如下面的 HTTP 服务器中，每一个路由函数都需要处理错误，这看起来很繁琐。

复制代码

```
func main() {
    http.HandleFunc("/users", viewUsers)
    http.HandleFunc("/companies", viewCompanies)
}

func viewUsers(w http.ResponseWriter, r *http.Request) {
    ...
    if err := userTemplate.Execute(w, user); err != nil {
        http.Error(w, err.Error(), 500)
    }
}

func viewCompanies(w http.ResponseWriter, r *http.Request) {
    ...
    if err := companiesTemplate.Execute(w, companies); err != nil {
        http.Error(w, err.Error(), 500)
    }
}
```

这时候，我们可以用一个类似中间件的函数 appHandler 来统一处理错误，改造如下：

复制代码

```go
func main() {
    http.HandleFunc("/users", appHandler(viewUsers))
    http.HandleFunc("/companies", appHandler(viewCompanies))
}

func viewUsers(w http.ResponseWriter, r *http.Request) {
    return userTemplate.Execute(w, user)
}

func viewCompanies(w http.ResponseWriter, r *http.Request) {
    return companiesTemplate.Execute(w, companies)
}

type appHandler func(http.ResponseWriter, *http.Request) error

func (fn appHandler) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    if err := fn(w, r); err != nil {
        http.Error(w, err.Error(), 500)
    }
}
```


第二个最佳实践是使用 defer 来减少错误处理。

例如，下面的函数在错误返回时，包装函数的逻辑是相同的。

复制代码

```go
func DoSomeThings(val1 int, val2 string) (string, error) {
    val3, err := doThing1(val1)
    if err != nil {
        return "", fmt.Errorf("in DoSomeThings: %w", err)
    }
    val4, err := doThing2(val2)
    if err != nil {
        return "", fmt.Errorf("in DoSomeThings: %w", err)
    }
    result, err := doThing3(val3, val4)
    if err != nil {
        return "", fmt.Errorf("in DoSomeThings: %w", err)
    }
    return result, nil
}
```

如果我们把这里的函数改造为使用 defer，会更为简洁：

```go
func DoSomeThings(val1 int, val2 string) (_ string, err error) {
    defer func() {
        if err != nil {
            err = fmt.Errorf("in DoSomeThings: %w", err)
        }
    }()
    val3, err := doThing1(val1)
    if err != nil {
        return "", err
    }
    val4, err := doThing2(val2)
    if err != nil {
        return "", err
    }
    return doThing3(val3, val4)
}
```


panic

最后，我们再来讲讲一类特殊的错误：panic。当发生算术除 0 错误、内存无效访问、数组越界等问题时，会触发程序 panic，导致程序异常退出并打印出函数的堆栈信息。

在 Go 语言中，有以下两个内置函数可以处理程序的异常情况：

```go
func panic(interface{})
func recover() interface{}
```

panic 函数传递的参数为空接口 interface{}，它可以存储任何形式的错误信息并进行传递，然后在异常退出时打印出来。

```go
panic(42)
panic("unreachable")
panic(Error("cannot parse"))
```

Go 程序在 panic 时并不会直接导致程序异常退出，它会终止当前正在正常执行的函数，执行 defer 函数并逐级返回。

例如，对于函数调用链 a()→b()→c()，当函数 c 发生 panic 后，会返回函数 b。此时，函数 b 也像发生了 panic 一样返回函数 a。函数 c、b、a 中的 defer 函数都将正常执行。

复制代码

```
func a() {
  defer fmt.Println("defer a")
  b()
  fmt.Println("after a")
}
func b() {
  defer fmt.Println("defer b")
  c()
  fmt.Println("after b")
}
func c() {
  defer fmt.Println("defer c")
  panic("this is panic")
  fmt.Println("after c")
}
func main() {
  a()
}
```

如下所示，当函数 c 触发了 panic 后，所有函数中的 defer 语句都会被正常调用，并且在 panic 时打印出堆栈信息。

复制代码

```
defer c
defer b
defer a
panic: this is panic
goroutine 1 [running]:
main.c()
    bookcode/panic/panic_chain.go:19 +0x95
main.b()
    bookcode/panic/panic_chain.go:13 +0x96
main.a()
    bookcode/panic/panic_chain.go:7 +0x96
main.main()
    bookcode/panic/panic_chain.go:24 +0x20

```

通常，我们希望能够捕获这样的错误，然后让程序继续正常执行。要捕获这种异常，我们需要将 defer 与内置 recover 函数结合起来使用。

```
func executePanic() {
  defer func() {
    if errMsg := recover(); errMsg != nil {
      fmt.Println(errMsg)
    }
    fmt.Println("This is recovery function...")
  }()
  panic("This is Panic Situation")
  fmt.Println("The function executes Completely")
}
func main() {
  executePanic()
  fmt.Println("Main block is executed completely...")
}
```


从下面这段输出可以看出，尽管有 panic，main 函数仍然在正常执行后才退出。

```
This is Panic Situation
This is recovery function...
Main block is executed completely...
```

在实践中，我们常常看到一些开发者为了避免异常退出在各个函数调用的地方都加上了 recover 捕获，但其实这是没有必要的。借助上面的这个特性，我们只需要在最上层函数捕获异常就可以了。

有些同学可能会想，那我直接在 main 函数里加一个 recover 函数，这样捕获异常不就可以了吗？但这是不对的，因为我们讲的这个特性只适用于某一个单独的协程。所以我们应该对每一个重要的协程的最上层函数进行捕获，这样就可以避免程序的异常退出了。

下面这段代码中，我们在 CreateWork 方法中捕获到 panic 并打印到了日志中。一般这种日志会被额外的监控系统发现并报警。

```
func (s *Crawler) CreateWork() {
  defer func() {
    if err := recover(); err != nil {
      s.Logger.Error("worker panic",
        zap.Any("err", err),
```

```
            zap.String("stack", string(debug.Stack())))
        }
    }()
...
}
```


总结

好了， 总结一下。

这节课，我们介绍了两个重要的特性限速器与错误处理。限速器通过令牌桶机制，帮助我们减少了爬虫对目标服务器的压力，并且使用了多层限速器对爬虫进行精细化的管理。

而对于错误处理，Go 语言吸收了当前软件开发中遇到的经验与教训，其哲学是函数的调用者理应对错误进行处理，但这个思路确实有时候会带来错误处理冗长的问题。错误处理目前有许多最佳的实践，其中包括了错误的包装，错误中间件以及通过 defer 来减少错误的处理等。

最后我们还介绍了 panic，在关键的功能最上层使用 defer+recover 可以防止程序异常退出。不过要注意，对异常进行捕获时，只能对单个协程生效。

课后题

学完这节课，给你留一道课后题。

错误处理是 Go 语言中被吐槽很多的话题，但是这其实是深思熟虑的设计结果。你可以查阅相关资料，谈一谈在 Go 中如何更好地进行错误处理。

欢迎你在留言区与我交流讨论，我们下节课见。

上一篇 32丨存储引擎：数据清洗与存储

下一篇 34丨服务注册与监听：Worker节点与etcd交互

精选留言 (3)

写留言

viclilei

2023-01-09 来自广东

老师，为什么我按照代码写完后，请求数据还是有1s20条左右。可以帮忙判断我大概是什么地方写错了吗？

共 1 条评论 >

卢承灏

2022-12-26 来自广东

老师，appHandler 讲解那里，viewUsers还有下面那个方法是不是漏了error 返回

Realm

2022-12-24 来自浙江

在程序中应该避免使用野生的goroutine带来的panic，可以放在一个有个异常捕获的Go函数中来执行goroutine。姿势如下：

```
func Go(f func()) {
	go func() {
		// defer recover 捕获panic
		defer func() {
			if err := recover(); err != nil {
				log.Printf("panic: %+v", err)
			}
		}()

		f()
	}()
}

func main() {
	f := func() {
		panic("xxx")
```

```
        }
        Go(f)

        time.Sleep(1 * time.Second)
}

```