Golang作为一个实用主义的编程语言，非常注重性能，在语言特性上天然支持并发，它有多种并发模型，通过流水线模型系列文章，你会更好的使用Golang并发特性，提高你的程序性能。

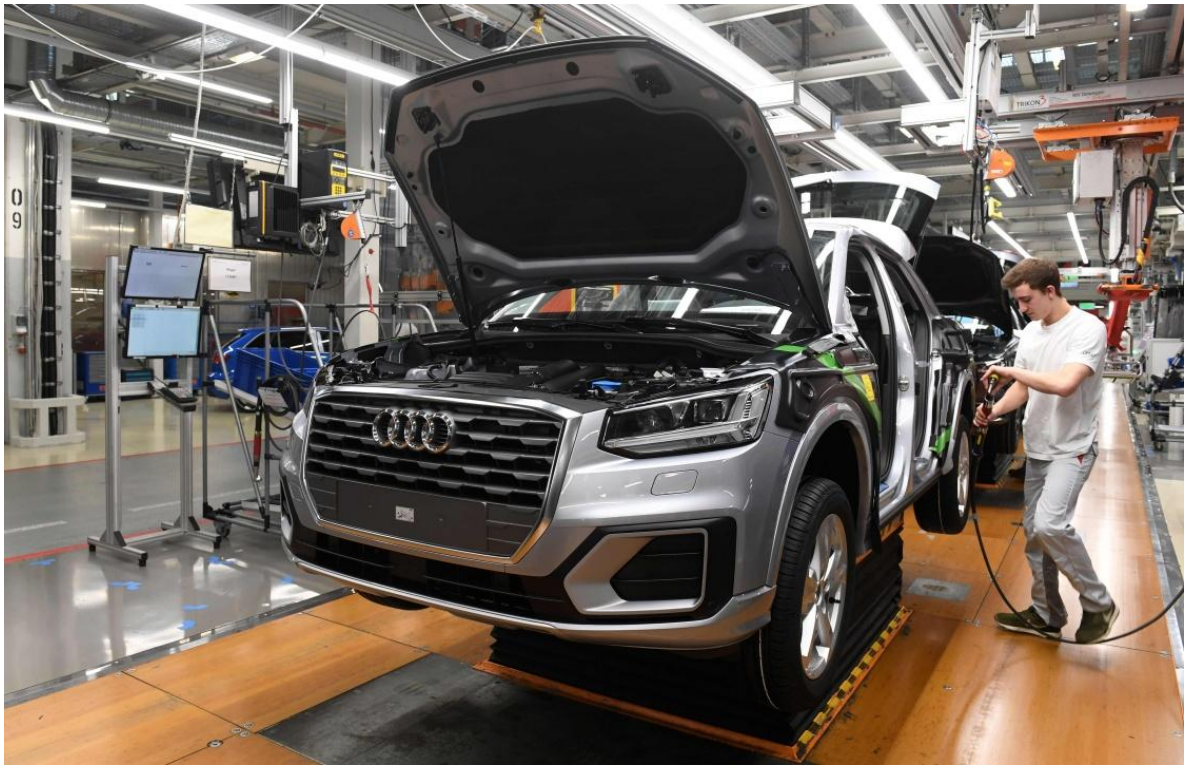
这篇文章主要介绍流水线模型的流水线概念，后面文章介绍流水线模型的FAN-IN和FAN-OUT，最后介绍下如何合理的关闭流水线的协程。

Golang的并发核心思路

Golang并发核心思路是关注数据流动。数据流动的过程交给channel，数据处理的每个环节都交给goroutine，把这些流程画起来，有始有终形成一条线，那就能构成流水线模型。

但我们先从简单的入手。

流水线并不是什么新奇的概念，它能极大的提高生产效率，在当代社会流水线非常普遍，我们用的几乎任何产品（手机、电脑、汽车、水杯），都是从流水线上生产出来的。以汽车为例，整个汽车流水线要经过几百个组装点，而在某个组装点只组装固定的零部件，然后传递给下一个组装点，最终一台完整的汽车从流水线上生产出来。



Golang的并发模型灵感其实都来自我们生活，对软件而言，高的生产效率就是高的性能。

在Golang中，流水线由多个阶段组成，每个阶段之间通过channel连接，每个节点可以由多个同时运行的goroutine组成。

从最简单的流水线入手。下图的流水线由3个阶段组成，分别是A、B、C，A和B之间是通道`aCh`，B和C之间是通道`bCh`，A生成数据传递给B，B生成数据传递给C。

流水线中，第一个阶段的协程是**生产者**，它们只生产数据。最后一个阶段的协程是**消费者**，它们只消费数据。下图中A是生成者，C是消费者，而B只是中间过程的处理者。

举个例子，设计一个程序：计算一个整数切片中元素的平方值并把它打印出来。非并发的方式是使用for遍历整个切片，然后计算平方，打印结果。

我们使用流水线模型实现这个简单的功能，从流水线的角度，可以分为3个阶段：

1. 遍历切片，这是生产者。
2. 计算平方值。
3. 打印结果，这是消费者。

下面这段代码：

- `producer()`负责生产数据，它会把数据写入通道，并把它写数据的通道返回。
- `square()`负责从某个通道读数字，然后计算平方，将结果写入通道，并把它的输出通道返回。
- `main()`负责启动producer和square，并且还是消费者，读取suqre的结果，并打印出来。

```
package main

import (
    "fmt"
)

func producer(nums ...int) <-chan int {
    out := make(chan int)
    go func() {
        defer close(out)
        for _, n := range nums {
            out <- n
        }
    }()
    return out
}

func square(inCh <-chan int) <-chan int {
    out := make(chan int)
    go func() {
        defer close(out)
        for n := range inCh {
            out <- n * n
        }
    }()

    return out
}

func main() {
    in := producer(1, 2, 3, 4)
    ch := square(in)

    // consumer
    for ret := range ch {
        fmt.Printf("%3d", ret)
    }
```

```
    fmt.Println()
}
```

结果：

```
➜  awesome git:(master) ✗ go run hi.go
1  4  9 16
```

这是一种原始的流水线模型，这种原始能让我们掌握流水线的思路。

流水线的特点

1. 每个阶段把数据通过channel传递给下一个阶段。
2. 每个阶段要创建1个goroutine和1个通道，这个goroutine向里面写数据，函数要返回这个通道。
3. 有1个函数来组织流水线，我们例子中是main函数。

如果你没了解过流水线，建议自己把以上的程序写一遍，如果遇到问题解决了，那才真正掌握了流水线模型的思路。