03 控制结构：if、for、switch 逻辑语句的那些事儿

在上节课中我留了一个思考题，在一个字符串中查找另外一个字符串是否存在，这个其实是字符串查找的功能，假如我需要在“飞雪无情”这个字符串中查找“飞雪”，可以这么做：

```
i:=strings.Index("飞雪无情","飞雪")
```

这就是 Go 语言标准库为我们提供的常用函数，以供我们使用，减少开发。

这节课我们继续讲解 Go 语言，今天的内容是：Go 语言代码逻辑的控制。

流程控制语句用于控制程序的执行顺序，这样你的程序就具备了逻辑结构。一般流程控制语句需要和各种条件结合使用，比如用于条件判断的 if，用于选择的 switch，用于循环的 for 等。这一节课，我会为你详细介绍，通过示例演示它们的使用方式。

if 条件语句

if 语句是条件语句，它根据布尔值的表达式来决定选择哪个分支执行：如果表达式的值为 true，则 if 分支被执行；如果表达式的值为 false，则 else 分支被执行。下面，我们来看一个 if 条件语句示例：

ch03/main.go

```
func main() {
    i:=10
    if i >10 {
        fmt.Println("i>10")
    } else {
        fmt.Println("i<=10")
    }
}
```

这是一个非常简单的 if……else 条件语句，当 i>10 为 true 的时候，if 分支被执行，否则就执行 else 分支，你自己可以运行这段代码，验证打印结果。

关于 if 条件语句的使用有一些规则：

1. if 后面的条件表达式不需要使用 ()，这和有些编程语言不一样，也更体现 Go 语言的简洁；
2. 每个条件分支（if 或者 else）中的大括号是必须的，哪怕大括号里只有一行代码（如示例）；
3. if 紧跟的大括号 { 不能独占一行，else 前的大括号 } 也不能独占一行，否则会编译不通过；
4. 在 if……else 条件语句中还可以增加多个 else if，增加更多的条件分支。

通过 go run ch03/main.go 运行下面的这段代码，会看到输出了 5<i<=10 ，这说明代码中的 else if i>5 && i<=10 成立，该分支被执行。

ch03/main.go

```
func main() {
    i:=6
    if i >10 {
        fmt.Println("i>10")
    } else if  i>5 && i<=10 {
        fmt.Println("5<i<=10")
    } else {
        fmt.Println("i<=5")
    }
}
```

你可以通过修改 i 的初始值，来验证其他分支的执行情况。

你还可以增加更多的 else if，以增加更多的条件分支，不过这种方式不被推荐，因为代码可读性差，多个条件分支可以使用我后面讲到的 switch 代替，使代码更简洁。

和其他编程语言不同，在 Go 语言的 if 语句中，可以有一个简单的表达式语句，并将该语句和条件语句使用分号 ; 分开。同样是以上的示例，我使用这种方式对其改造，如下面代码所示：

ch03/main.go

```
func main() {
    if i:=6; i >10 {
        fmt.Println("i>10")
    } else if  i>5 && i<=10 {
        fmt.Println("5<i<=10")
    } else {
        fmt.Println("i<=5")
    }
}
```

在 if 关键字之后，i>10 条件语句之前，通过分号 ; 分隔被初始化的 i:=6。这个简单语句主要用来在 if 条件判断之前做一些初始化工作，可以发现输出结果是一样的。

通过 if 简单语句声明的变量，只能在整个 if……else if……else 条件语句中使用，比如以上示例中的变量 i。

switch 选择语句

if 条件语句比较适合分支较少的情况，如果有很多分支的话，选择 switch 会更方便，比如以上示例，使用 switch 改造后的代码如下：

ch03/main.go

```
switch i:=6;{
case i>10:
    fmt.Println("i>10")
case i>5 && i<=10:
    fmt.Println("5<i<=10")
default:
    fmt.Println("i<=5")
}
```

switch 语句同样也可以用一个简单的语句来做初始化，同样也是用分号 ; 分隔。每一个 case 就是一个分支，分支条件为 true 该分支才会执行，而且 case 分支后的条件表达式也不用小括号 () 包裹。

在 Go 语言中，switch 的 case 从上到下逐一进行判断，一旦满足条件，立即执行对应的分支并返回，其余分支不再做判断。也就是说 Go 语言的 switch 在默认情况下，case 最后自

带 break。这和其他编程语言不一样，比如 C 语言在 case 分支里必须要有明确的 break 才能退出一个 case。Go 语言的这种设计就是为了防止忘记写 break 时，下一个 case 被执行。

那么如果你真的有需要，的确需要执行下一个紧跟的 case 怎么办呢？Go 语言也考虑到了，提供了 fallthrough 关键字。现在看个例子，如下面的代码所示：

ch03/main.go

```
switch j:=1;j {
case 1:
    fallthrough
case 2:
    fmt.Println("1")
default:
    fmt.Println("没有匹配")
}
```

以上示例运行会输出 1，如果省略 case 1: 后面的 fallthrough，则不会有任何输出。

不知道你是否可以发现，和上一个例子对比，这个例子的 switch 后面是有表达式的，也就是输入了 ;j，而上一个例子的 switch 后只有一个用于初始化的简单语句。

当 switch 之后有表达式时，case 后的值就要和这个表达式的结果类型相同，比如这里的 j 是 int 类型，那么 case 后就只能使用 int 类型，如示例中的 case 1、case 2。如果是其他类型，比如使用 case "a" ，会提示类型不匹配，无法编译通过。

而对于 switch 后省略表达式的情况，整个 switch 结构就和 if……else 条件语句等同了。

switch 后的表达式也没有太多限制，是一个合法的表达式即可，也不用一定要求是常量或者整数。你甚至可以像如下代码一样，直接把比较表达式放在 switch 之后：

ch03/main.go

```
switch 2>1 {
case true:
    fmt.Println("2>1")
case false:
    fmt.Println("2<=1")
}
```

可见 Go 语言的 switch 语句非常强大且灵活。

for 循环语句

当需要计算 1 到 100 的数字之和时，如果用代码将一个个数字加起来，会非常复杂，可读性也不好，这就体现出循环语句的存在价值了。

下面是一个经典的 for 循环示例，从这个示例中，我们可以分析出 for 循环由三部分组成，其中，需要使用两个 ; 分隔，如下所示：

ch03/main.go

```
sum:=0
for i:=1;i<=100;i++ {
    sum+=i
}
fmt.Println("the sum is",sum)
```

其中：

1. 第一部分是一个简单语句，一般用于 for 循环的初始化，比如这里声明了一个变量，并对 i:=1 初始化；
2. 第二部分是 for 循环的条件，也就是说，它表示 for 循环什么时候结束。这里的条件是 i<=100；
3. 第三部分是更新语句，一般用于更新循环的变量，比如这里 i++，这样才能达到递增循环的目的。

需要特别留意的是，Go 语言里的 for 循环非常强大，以上介绍的三部分组成都不是必须的，可以被省略，下面我就来为你演示，省略以上三部分后的效果。

如果你以前学过其他编程语言，可能会见到 while 这样的循环语句，在 Go 语言中没有 while 循环，但是可以通过 for 达到 while 的效果，如以下代码所示：

ch03/main.go

```
sum:=0
i:=1
for i<=100 {
    sum+=i
    i++
```

```
}
fmt.Println("the sum is",sum)
```

这个示例和上面的 for 示例的效果是一样的，但是这里的 for 后只有 i<=100 这一个条件语句，也就是说，它达到了 while 的效果。

在 Go 语言中，同样支持使用 continue、break 控制 for 循环：

1. continue 可以跳出本次循环，继续执行下一个循环。
2. break 可以跳出整个 for 循环，哪怕 for 循环没有执行完，也会强制终止。

现在我对上面计算 100 以内整数和的示例再进行修改，演示 break 的用法，如以下代码：

ch03/main.go

```
sum:=0
i:=1
for {
    sum+=i
    i++
    if i>100 {
        break
    }
}
fmt.Println("the sum is",sum)
```

这个示例使用的是没有任何条件的 for 循环，也称为 for 无限循环。此外，使用 break 退出无限循环，条件是 i>100。

总结

这节课主要讲解 if、for 和 switch 这样的控制语句的基本用法，使用它们，你可以更好地控制程序的逻辑结构，达到业务需求的目的。

这节课的思考题是：任意举个例子，练习 for 循环 continue 的使用。

Go 语言提供的控制语句非常强大，本节课我并没有全部介绍，比如 switch 选择语句中的类型选择，for 循环语句中的 for range 等高级能力。这些高级能力我会在后面的课程中逐一介绍，接下来要讲的集合类型，就会详细地为你演示如何使用 for range 遍历集合，记得来听课！