Go语言语法特性

一、课前准备

> 说明：提前需要让学生做的课前准备，比如环境安装部署、工具安装、插件安装等，需要学生提前做的都放这，并且给出下载链接或信息源；

1. Golang开发环境环境安装就绪；
2. Golang-IDE开发环境安装就绪；

二、课堂主题

> 说明：本堂课的总体概述，明确课堂主题和主体；

本节主要介绍Go语言当中的语法特性，包括Go语言的类型，变量定义和使用，常量用法，指针变量，函数以及函数指针，匿名函数与函数闭包等。

三、课堂目标

> 说明：主要是让学生了解，学了本堂课后，能达到的一个期望值，要量化；

1. 掌握Go语言变量的定义方法；
2. 掌握Go语言常量的使用方法；
3. 掌握Go语言iota的用法；
4. 掌握Go语言函数的声明；
5. 掌握Go语言的匿名函数；
6. 掌握Go语言的函数闭包；
7. 掌握Go语言的循环方式；
8. 掌握Go语言的分支处理；

四、知识点

> 说明：
>
> 1. 知识点是课堂主体，本堂课所有知识点都列出来；
> 2. 预估每个知识点需要讲解的时间；
> 3. 研发逻辑就是讲解逻辑，一般从上往下，遵循：`What - Why - How` 或 `Why - What - How` 思路；

1. 类型与变量（5分钟）

1.1 Go语言的数据类型

Go语言的数据类型定义非常丰富，下面来看看。

- 布尔类型 bool

true 或者 false

- 整型

类型	取值范围	描述
uint8	0 - 255	无符号8位整数
uint16	0 - 65535	无符号 16 位整型
uint32	0 - 4294967295	无符号 32 位整型
uint64	0 - 18446744073709551615	无符号 64 位整型
int8	-128 - 127	有符号8位整数
int16	-32768 - 32767	有符号 16 位整型
int32	-2147483648 - 2147483647	有符号 32 位整型
int64	-9223372036854775808 - 9223372036854775807	有符号 64 位整型

- 浮点型

类型	描述
float32	IEEE-754 32位浮点型数
float64	IEEE-754 64位浮点型数
complex64	32 位实数和虚数
complex128	64 位实数和虚数

备注：IEEE-754为IEEE二进制浮点数算术标准（ANSI/IEEE Std 754-1985）

- 其他类型

类型	描述
byte	类似uint8
rune	类似uint32
uint	32或64位，取决于操作系统
int	与uint一样大小
uintptr	无符号整形，存放指针

1.2 Go语言的变量定义

Go语言中，变量的声明方式如下：

```
var identifier type
```

也可以选择在声明的时候，直接赋值：

```
var identifier [type] = value
```

在类型定义的时候，可以省略type，Go语言可以自动推到数据类型。我们也可以同时定义多个变量，在赋值的时候，需要注意，等号左右的数量、类型要一致。比如：

```
var x, y = 123, "hello"
```

var的定义变量方法没有区域限制，相比而言，Go开发者更喜欢另外一种定义变量且赋值的方式，那就是用":="，":="必须出现函数中，这种方式可以方便的定义变量，包括接收函数返回结果，它利用的仍然是Go语言自动推到类型的优势。

```
a := 123
str := "hello"
```

当然":="还有一个注意事项，使用":="的前提是当前代码段，该变量没有定义过。下面来段代码，跑一跑试试。

```
package main

import (
  "fmt"
)

func main() {
  fmt.Println("hello world")
  var v1 int
  var x, y = 123, "hello"
  a, str := 456, "world"
  fmt.Println(v1, x, y, a, str)
  //定义复数，注意i一定要与数字紧紧相连，否则会被当成字符i处理
  var c1 complex64 = 4 + 3i
  fmt.Println(c1)

}
```

运行结果如下：

```
localhost:day01 teacher$ go run 02-var.go
hello world
0 123 hello 456 world
(4+3i)
```

2. 常量与iota（5分钟）

2.1 常量声明

在Go语言当中，使用const关键字定义常量，所谓常量，就是在程序运行时，值不会被修改的标识。常量的定义格式如下：

```
const identifier [type] = value
```

根据我们的经验，[type]这样被中括号扩起来的代表可以省略，也就是常量也可以支持类型它推导。同样的，常量也可以一次定义多个，这一点与变量是相同的。来看一个示例：

```
package main

import "fmt"

func main() {
  const LENGTH int = 10
  const WIDTH = 5
  const a, b, c = 1, false, "str" //多重赋值

  //注意用 := ,代表定义变量area
  area := LENGTH * WIDTH
  //Printf与Printfln的区别是格式化以及自带换行
  fmt.Printf("area is %d\n", area)
  println(a, b, c)
}
```

运行结果如下：

```
localhost:day01 teacher$ go run 03-const.go
area is 50
1 false str
```

2.2 优雅的使用常量

在常量定义中，为了使常量的值更灵活，我们可以使用iota关键字，使用iota后，常量定义可以非常灵活。

```
package main

import (
  "fmt"
)
```

```
//定义业务类型
const (
  login = iota // iota = 0
  logout
  user    = iota + 2 //iota = 2,user = 2+2 = 4
  account = iota * 3 //iota = 3, account = 3*3 = 9
)

const (
  apple, banana = iota + 1, iota + 2 // iota = 0
  peach, pear                        //iota = 1
  orange, mango                      //iota = 2
)

func main() {
  fmt.Println(login, logout, user, account)
  fmt.Println(apple, banana, peach, pear, orange, mango)
}
```

执行结果如下：

```
localhost:day01 teacher$ go run 04-iota.go
0 1 4 9
1 2 2 3 3 4
```

3 Go语言的指针（5分钟）

在C语言里，掌握指针几乎可以掌握一切，Go语言中的指针并没有像C语言里那样妖魔化，指针所表达的就是它指向变量的地址。指针背后的作用往往才是更关键的，那就是值传递与引用传递。

```
  a := 10
  var b *int = &a // b指向a的地址
  fmt.Println(*b) //打印的是10
  *b = 100 //通过b可以修改a的值
  fmt.Println(a, *b) // 此时打印的都是100
```

如果我们想对两个变量的值进行互换，函数像下面这样写，最后得到的是失败的结果。

```
func swap(a , b int) {
  temp := a
  a = b
  b = temp
}
```

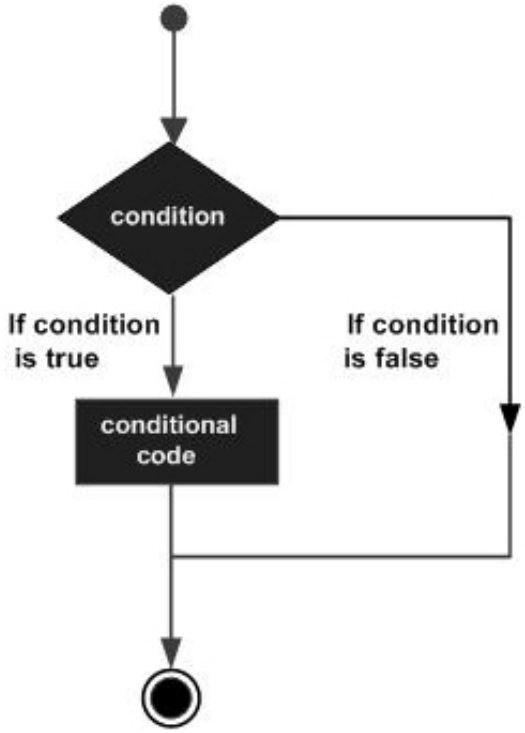
很显然值传递的方式不能解决变量替换的问题，通过指针是可以的。

```
func swap2(a, b *int) {
  temp := *a
  *a = *b
  *b = temp
}
```

当然Go语言的强大特性也可以通过其他方式解决值交换的问题，如果了解了函数的多返回值，到时候不用指针也可以同样解决问题。

4 条件与分支

条件分支是任何语言必不可少的一部分，Go语言的条件分支语法上还真有特殊性。



我们下面来看一个if-else的例子：

```
/*
   author:teacher
   company:teacher
   filename:05-if.go
*/
package main

import (
  "fmt"
)

func main() {
  a := 10
```

```
  if a > 10 { //左括号一定要写在表达式同行，与函数要求相同
    fmt.Println("My God ,a is ", a)
  } else if a < 10 {
    fmt.Println("a is too small")
  } else {
    fmt.Println("a == ", a)
  }
}
```

运行结果如下：

```
localhost:day01 teacher$ go run 05-if.go
a ==  10
```

也要特别注意一点：Go语言中if后的表达式必须是bool型值，否则语法检测不通过。

与其他语言类似，Go语言也支持switch，switch的好处就是可以针对某个变量可能的值进行分支匹配处理，语法如下：

```
switch var1 {
    case val1:
        ...
    case val2:
        ...
    default:
        ...
}
```

示例代码如下：

```

package main

import (
  "fmt"
)

func main() {
  var fruit string
  fmt.Println("Please input a fruit's name:")
  fmt.Scanf("%s", &fruit)
  switch fruit {
  case "banana":
    fmt.Println("I want 2 banana!")
  case "orange":
    fmt.Println("I want 3 orange!")
  case "apple":
    fmt.Println("I want an apple!")
```

```
  case "pear":
    fmt.Println("I do not like pear!")
  default:
    fmt.Println("Are you kidding me?")
  }
}
```

5 Go语言的循环

Go语言种循环没有while关键字，取而代之的都是由for来处理，for支持三种方式：

1. for init; condition; post { }
2. for condition { }
3. for {}

我们还是先看例子：

```
/*
   author:teacher
   company:teacher
   filename:07-for.go
*/

package main

import (
  "fmt"
  "time"
)

func main() {
  //计算1+2+3+……+100 = ?
  //第一种方式
  sum := 0
  i := 0
  for i = 1; i <= 100; i++ {
    sum += i
  }
  fmt.Println("sum is ", sum)
  //第二种方式
  i = 1
  sum = 0
  for i <= 100 {
    sum += i
    i++
  }
  fmt.Println("sum is ", sum)
  //死循环 - 开启刷屏模式
  for {
```

```
    fmt.Println("heihei")
    time.Sleep(time.Second * 1) //每次执行睡眠1s
  }
}
```

执行结果如下，最后按ctrl+c中断该进程：

```
localhost:day01 teacher$ go run 07-for.go
sum is  5050
sum is  5050
heihei
heihei
heihei
heihei
heihei
heihei
^Csignal: interrupt
```

6 函数

6.1 函数声明

函数是针对某些特定功能的封装，便于多次调用，函数也是语言当中非常重要的部分，Go语言函数的格式如下：

```
func function_name( [parameter list] ) [return_types] {
   函数体
}
```

说明：

- func 函数关键字
- function_name 函数名称
- parameter list 参数列表，根据设计需要，提供参数的顺序和类型要求，可以为空
- return_types 返回类型，支持0或多个返回类型，如果超过一个类型时，需要用小括号
- 函数体 函数的功能实现部分

来看一个例子：

```
/*
   author:teacher
   company:teacher
   filename:08-func.go
*/

package main
```

```
import (
  "fmt"
)

func main() {
  //函数调用，同时获得2个返回值
  sum, sub := add_sub(32, 21)
  fmt.Println(sum, sub)
  //获得函数指针，此时addsubptr相当于 func add_sub(a int, b int) (int, int)
  addsubptr := add_sub
  //通过函数指针的方式调用
  sum1, sub1 := addsubptr(1, 2)
  fmt.Println(sum1, sub1)
}

func add_sub(a int, b int) (int, int) {
  return a + b, a - b
}
```

例子中，我们实现了一个函数 add_sub，它有2个参数，a和b，这种写法属于常规写法，如果相邻参数相同时，我们也可以简化类型，写成下面这样：

```
func add_sub(a, b int) (int, int) {
  return a + b, a - b
}
```

6.2 函数作为参数

此外，函数也可以作为函数的参数进行传递。

```
/*
   author:teacher
   company:teacher
   filename:09-func2.go
*/

package main

import (
  "fmt"
)

func main() {
  a := math(10, 20, add) //传入add函数，求和
  b := math(10, 20, sub) //传入sub函数，求差
  fmt.Println(a, b)
}
```

```
func add(a int, b int) int {
  return a + b
}
func sub(a int, b int) int {
  return a - b
}

//函数作为特殊的类型也可以当作参数,调用时要求f参数必须是 func(a, b int) int 这样类型的函数
func math(a, b int, f func(a, b int) int) int {
  return f(a, b)
}
```

上述例子math函数的第三个参数f就是一个函数，调用时它可以是add，也可以是sub，前提是这2个函数的声明必须与f的类型一致。执行结果如下：

```
localhost:day01 teacher$ go run 09-func2.go
30 -10
```

6.3 匿名函数与函数闭包

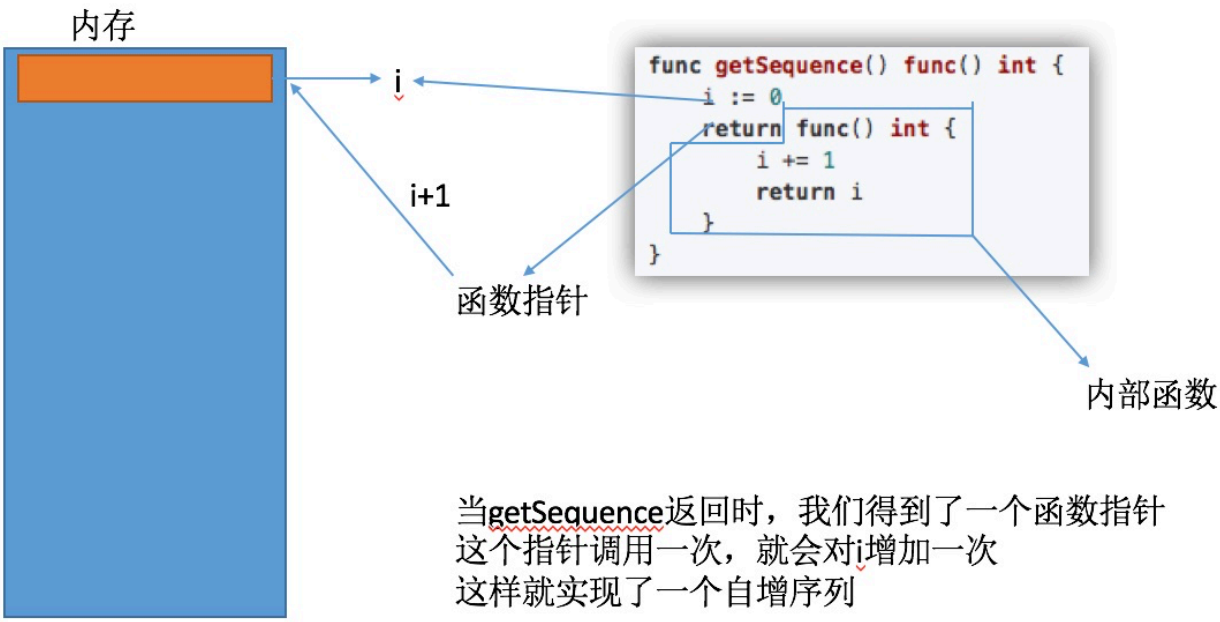
有的时候，我们为了支持一个功能而实现一个函数，但是还不想给这段功能实现起一个名字，我们可以定义匿名函数，Go语言也是借鉴了其他语言的特点，在语法上支持了匿名函数，匿名函数与后文介绍Go并发相互配合，杀伤力巨大。

```
func(a, b int) int {
    return a + b
}(3, 4)
```

注意上述代码为一个匿名函数执行的代码段，它与普通函数的区别是没有函数名称，并且在{}之外还有调用参数传递，如果没有参数，直接放()就可以了。

由于对匿名函数的支持，再加上Go语言的函数也可以作为返回值，所以在Go语言中可以支持闭包，所谓闭包就是能够读取其他函数内部变量的函数。

可以先看下图：




```
/*
   author:teacher
   company:teacher
   filename:10-func-closure.go
*/

package main

import "fmt"

func getSequence() func() int {
  i := 0
  return func() int {
    i += 1
    return i
  }
}

func main() {
  // nextNumber 为一个函数，函数 i 为 0
  nextNumber := getSequence()

  // 调用 nextNumber 函数，i 变量自增 1 并返回
  fmt.Println(nextNumber())
  fmt.Println(nextNumber())
  fmt.Println(nextNumber())

}
```

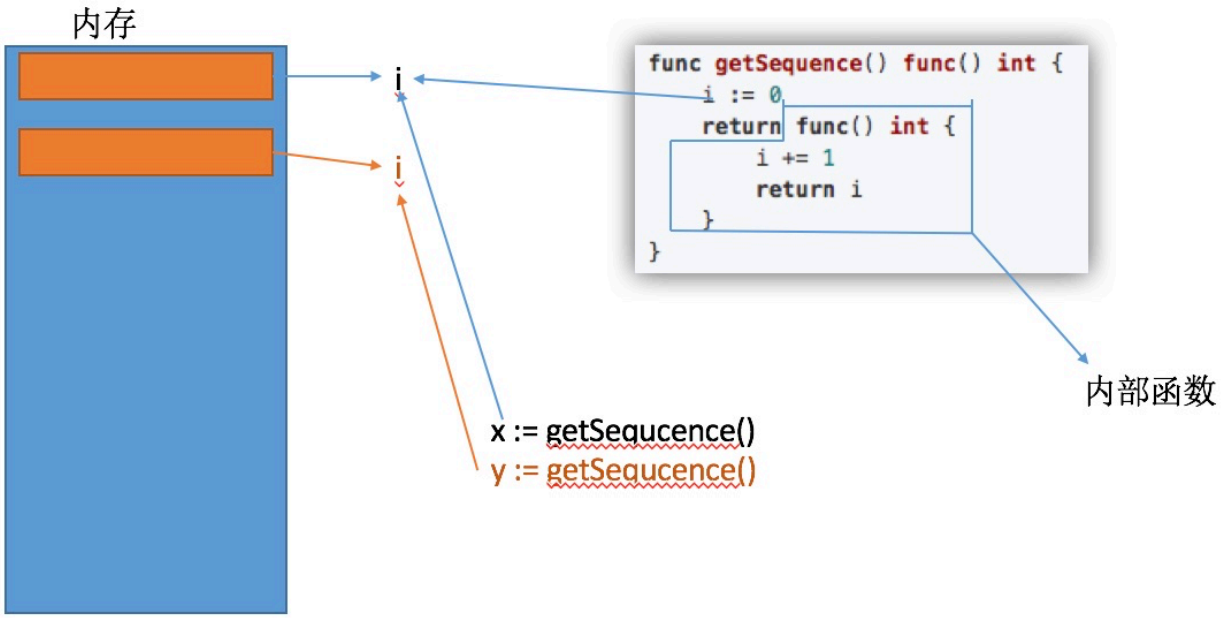
执行结果如下：

```
localhost:day01 teacher$ go run 10-func-closure.go
1
2
3
```

在上述代码中，getSequence函数内部变量i正常的生命周期是函数执行结束后，在getSequence内部又返回一个匿名函数，由于这个匿名函数返回的是i的值，这样就导致i仍然在内存区域中不会释放，当使用nextNumber := getSequence()也就拿到了这个匿名函数，每调用一次，i的值都会增加，就类似于实现了数据库里面的自增序列，那么再定义一个nextNumber1也是等于getSequence()，它代表的序列是与nextNumber一致呢，还是重新开始呢？执行一下，可以思考一下为什么？

```
/*
   author:teacher
   company:teacher
   filename:10-func-closure.go
*/

package main

import "fmt"

func getSequence() func() int {
  i := 0
  return func() int {
    i += 1
    return i
  }
}

func main() {
  // nextNumber 为一个函数，函数 i 为 0
  nextNumber := getSequence()

  // 调用 nextNumber 函数，i 变量自增 1 并返回
  fmt.Println(nextNumber())
  fmt.Println(nextNumber())
  fmt.Println(nextNumber())

  fmt.Println("--------------------") //华丽的分割线
  nextNumber1 := getSequence()
  fmt.Println(nextNumber1())
  fmt.Println(nextNumber1())

}
```

从执行结果来看，nextNumber1又从1开始重新开始了，虽然他们函数内部变量名字都是i，但是在不同的函数调用内部，i所在的内存区域也是不同的，因而nextNumber和nextNumber1代表的必然是不同的序列。



6.4 小结

Go语言的函数非常灵活，尤其是匿名函数，函数指针，让我们有更多选择。

五、拓展点

> 说明：
>
> 1. 典型面试题、笔试题；
> 2. 新技术 or 经验分享；
> 3. 未来计划、行业趋势分享；

六、总结

> 说明：
>
> 1. 回顾本堂课所有知识点；
> 2. 提示`注意点`和`重点`；
> 3. 提示学习方法；
> 4. 提示哪些需要记、哪些需要背、哪些代码需要敲； 少即是多，化繁为简是Go语言的设计宗旨，这一点在循环处理上，在函数声明设计上，在变量定义中都体现的非常明显。Go语言的前期学习需要关注以下要点：

- 语法格式与代码格式化
- := 的变量定义方式

- for循环的不同用法
- 函数的声明
- 匿名函数
- 函数闭包

七、大作业

> 说明：
>
> 1. 给出明确的作业要求，形成文字、图片或测试题等形式；
> 2. 给出明确的解答方式；
> 3. 频次低，可设计为：1次/周 或 1次/2周；

可从下面的检测题题库任意选择。

八、集中答疑

> 说明：完成本堂课所有知识点的讲解后，由学员集中提问，讲师逐一解答；

九、检测题

> 说明：
>
> 1. 针对本堂课，设计5-10个题，由学员课下完成（课下刷题）；
> 2. 出题范围可从检查点里挑选，可以是面试题或笔试题；
> 3. 题型要求是 单选、多选、判断、填空中的一种或多种；
> 4. 频次高，原则上每次课都要有检测题；

十、下节课预告

> 说明：
>
> 1. Go语言面向对象编程；
> 2. Go语言的interface；