01｜知识回顾：Go基础知识你真的掌握了吗？

2022-10-11 郑建勋 来自北京

《Go进阶 · 分布式爬虫实战》

讲述：郑建勋

时长 20:48　大小 19.01M

你好，我是郑建勋。

在开篇词我们就提到，这个专栏的目标就是完成一个结合了高并发、分布式、微服务的复杂 Go 语言项目。

构建一个复杂的 Go 项目就和搭建复杂的积木一样。想象一下，当我们想搭建复杂的积木时，首先需要准备良好的环境（宽阔整洁的桌面、收纳盒），拥有基础的要素（各种类型的零件），掌握必要的规则（说明书中零件拼接的规则）。同样的，在构建复杂 Go 语言项目之前，我们也需要掌握一些 Go 语言的基础知识。

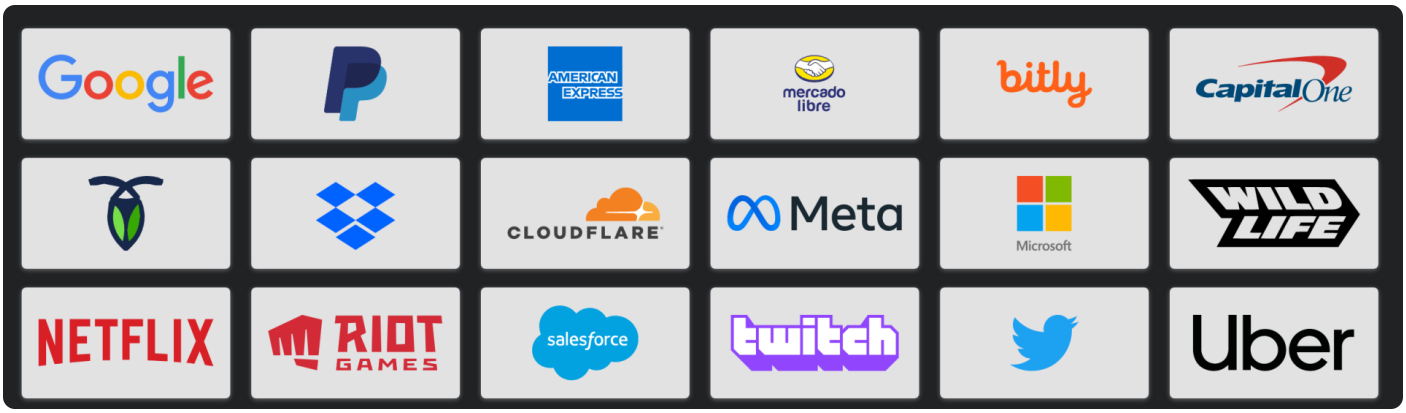
我在实际的工作中发现，即便是互联网大厂的员工，能够体系化掌握 Go 语言用法的人也比较少见。很多人做项目的方式就是直接干，这当然是一种解决问题的思路。但缺少对 Go 语言体系化的了解，也确实限制了我们对语言的使用，同时还很可能为未来埋下隐患。

举个例子，在 Go 项目中不使用通道可能并不妨碍我们完成功能，但如果我们压根不知道 Go 中还有这种更好地实现协程间通信的方式，我们搭建出的项目的质量是值得怀疑的，错误的设计对项目后期的影响是深远的。

也正是因此，我觉得接下来两节课，我有必要先梳理一下 Go 语言的基础知识，让你能够查漏补缺，具备进一步学习的理论基础。

Go 语言的历史与设计理念

对于一门 2009 年才正式开源的高级编程语言来说，Go 语言取得了非凡的成功。Go 已成为云原生领域的流行语言，杀手级的系统 Docker 与 Kubernetes 都是用 Go 编写的，目前国内国外使用 Go 的公司都相当多。



国外使用Go的大公司（来自go.dev）

了解这门语言诞生的时代背景、它的创造者的编程哲学和设计理念，将有助于我们更好地理解这门语言适用的场景及其未来的走向。

实际上，没有语言的时代，只有时代的语言。任何语言都是顺应时代发展的产物。我们过去已经有了很多经典的语言（C、C++, Java、Python），它们都是在特定的时代背景下，为了解决特定的问题而诞生的。

然而，在互联网迅猛发展的数十年中，出现了越来越多新的场景与挑战，例如大数据、大规模集群计算、上千万行的服务器代码、更复杂的网络环境、多核处理器开始成为主流……那些成熟但上了年纪的语言没能为新的挑战给出直接的解决方案，Go 语言就在这种时代背景下应运而生了。

正如罗勃·派克（Rob Pike）在 2012 年的演讲中提到的，Go 是为了应对谷歌在软件工程和基础架构上遇到的困难设计出来的。这些困难包括，软件开发开始变得缓慢和笨拙、软件设计

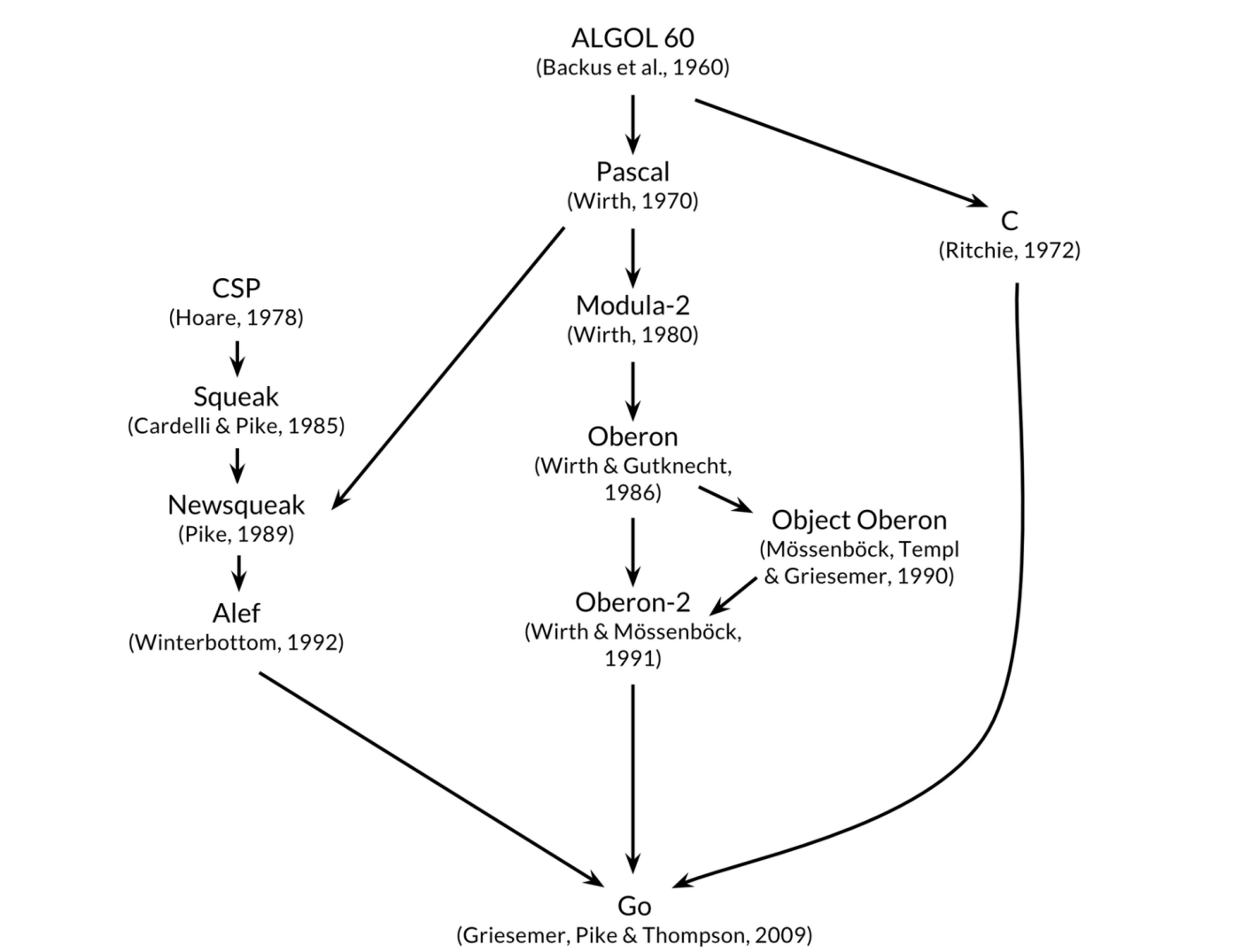
的复杂度越来越高、编译速度越来越慢等等。因此，设计 Go 的目的并不是要探索一种突破性的语言，而是希望 Go 能够专注于软件开发过程本身，成为设计大规模软件项目的优秀工具，为软件开发提供生产力和扩展性。

为了达到设计的目标，Go 在充分吸收借鉴优秀开发语言特性的基础上，也审视了这些语言具有的缺陷。**语言设计者使用第一性原理思维，思考复杂挑战背后的本质问题，并尝试用简单的设计来解决它们。**

首先，让我们来看看 Go 从哪些编程语言那里“取了经”。

Go 的祖先

就像生物的杂交有时会让后代产生惊人的优势一样，新的编程语言也会融合过去优秀编程语言的特性和优点，同时这种组合也带来了新的强大的表现力。如下图所示，Go 至少从三个“祖先”中吸取了养分。



来自《The Go Programming Language》

Go 的第一个祖先是 C 语言，其实，Go 有时被描述为 21 世纪的 C 语言，这是因为 Go 在许多方面（表达式语法，控制流语句，基本数据类型，参数的值传递，指针）都和 C 语言比较相似。

同时，Go 和 C 都致力于更接近机器，编译出高效的机器识别的二进制代码。这和 Python 这样的脚本语言以及需要把代码转换为字节码的 Java 有本质的不同。Go 在 C 语言语法的基础上做了许多改进，包括在 if 和 for 中不用加入()，也不用在每个语句的末尾加入;等等。

Go 的第二个祖先来自于 Pascal → Modula-2 → Oberon → Oberon-2 这条语言分支，Go 语法设计中的 package、import、声明以及特殊的方法声明的灵感都来源于此。

Go 的第三个祖先来自于 CSP→ Squeak → Newsqueak → Alef 这条语言分支，Go 从中借鉴了 CSP，并引入了 Channel 用于协程间的通信，这同时也是 Go 区别于其他语言的重要特性。

Go 的特性

除了继承其他语言的优点，Go 在设计之初还考虑了其他语言在当前面临的困境。Go 的特性解决了当前软件开发中的诸多问题，下面我来介绍其中比较重要的几个特性。

- **良好的编译器和依赖设计**

过去，C、C++ 编写的大型项目面临编译缓慢，依赖不受控制的问题。Go 语言为了加快编译的速度，在编译时，如果 package 中有未使用的依赖项会直接报错，这保证了 Go 程序的依赖树是精确的。在构建程序时，Go 不会编译多余的代码，这就最大限度地减少了编译时间。

另外，Go 的编译器还做了大量优化。当编译器执行 import 导入包时，它只需要打开一个与导入包相关的 obj 文件（object file），而不是导入依赖的源代码。这种方式具有扩展性，因此不会随着 Go 代码库的增多导致编译时间的指数级上升。同时，Go 对 obj 文件的结构也做了优化，以便更快速地导入依赖。

Go 依赖管理的另一个特性是不能有循环依赖。因为代码之间的循环纠缠最终会使各个模块之间难以解耦，难以独立管理。取消了循环依赖意味着编译器不必一次性编译大量的源代码。不过，这也意味着我们需要在程序设计的早期就考虑好 package 的边界，然后合理设计 package 的目录，不然，到后期一旦出现循环依赖问题，将很难完美解决。

Go 还对标准库进行了精心的设计，设计者甚至不惜有一些冗余的代码也要避免导入一个繁重的依赖库，这在一定程度上也缩短了编译时间。

- **面向组合而不是继承**

Java、C++ 这样面向对象的语言曾经为软件工程带来了一场深刻的革命，它们通过将事物抽象为对象和对象的行为，并通过继承等方式实现了对象之间的关联。相比面向过程的编程，Java、C++ 这类语言对现实有了更强的解释力，事实也证明了面向对象思想在构建大规模程序中的成功。

Go 可以实现面向对象的编程，但采用的是一种不同寻常的方式。Go 语言中没有继承，这是因为 Go 语言的设计者认为，继承带来了类型的层次结构。随着程序的发展，继承使代码变得越来越难以变动，这会使代码变得非常脆弱。同时，这也导致开发者容易在前期进行过度设计。

因此，Go 语言中没有基于类型的继承，取而代之的是扁平化的，面向组合的设计。这种正交式的组合不仅更容易构建起复杂的程序，而且在后期功能变动时也有很强的扩展性。

在 Go 语言中，我们可以为任何自定义的类型添加方法，而不仅仅是对象（例如 Java、C++ 中的 Class）。Go 语言中的接口是一种特殊的类型，是其他类型可以实现的方法签名的集合。只要类型实现了接口中的方法签名，就隐式地实现了该接口。这种隐式实现接口的方式被叫做 Duck Typing，这是一种非常有表现力的设计。

- **并发原语**

Go 诞生的时期正是多核 CPU 盛行和互联网野蛮生长的时代，这对高并发提出了新的要求。例如，典型的 Web 服务现在需要处理海量的用户连接，而古老的 C++ 或 Java 在语言级别上缺乏足够的并发支持。但是 Go 语言原生支持并发，Go 在线程之上抽象出了更加轻量级的协程，通过简单的 Go 关键字就可以快速创建协程，并借助 Go 运行时对协程进行管理与调度。

同时，Go 语言实现了 CSP 并发编程模式，通道成为了 Go 语言中的一等公民。通过通道来共享内存的方式屏蔽了很多底层实现细节，不像传统的多线程编程需要开发者了解互斥锁、条件变量及内存屏障等细节。Go 让并发编程变得更加简单了。

- **简单与健壮性**

很多人都觉得 Go 是一门简单的语言，易上手。确实，Go 语言的语法相对简单，少有一些复杂的特性。这种简单性一部分是因为 Go 团队在将新的特性加入语法时非常谨慎，非必要不添加。在其他语言的实践中，一些特性常常带来的是复杂性而不是生产力。这也是为什么 Go 在一开始并没有将泛型加入到语法中。良好的设计是需要经过时间验证的，最终经过社区反复讨论，Go1.18 开始加入了泛型这一特性。

另一方面，Go 为了保证代码的健壮性，屏蔽了一些容易犯错的操作。例如没有隐式的数值转换、没有指针运算、没有类型别名、运行时会检查数组的边界。Go 语言也没有手动的内存管理，而是让运行时托管了无用内存的释放工作，这又被称为垃圾回收。Go 还拥有内存逃逸功能，这意味着我们可以传递栈上变量的地址，而这在 C 语言中会产生类似野指针的问题。

Go 中还有一些设计和语言设计理念有关，例如 Go 中没有继承、不暴露线程的局部存储、不暴露协程的 ID。

- **强大丰富的标准库与工具集**

软件工程需要包含一整套工具，这样才能更好地完成代码编写、代码编译、代码调试、代码分析、代码测试、代码部署等工作。例如，Go 中自带的 go fmt 工具可以完成代码的格式化。go vet 可以报告代码中可能的错误，go doc 则可以用于生成代码的注释文档。在后面，我们还会看到其他优秀的工具与库。

Go 基础知识体系

刚才，我们介绍了 Go 语言的诞生背景以及一些重要特性。接下来，让我们更具体地看一看我们需要具备哪些 Go 语言的基础知识。我把这些知识分为了六个部分：

- 开发环境；
- 基础语法；
- 语法特性；
- 并发编程；
- 项目组织；
- 工具与库。

下面我们分别来看看每一部分的内容。

开发环境

俗话说工欲善其事，必先利其器。一个合格的开发者首先需要准备好自己的开发环境。这主要包括以下五点。

1. 安装语言处理系统（从而能够解释、编译或运行编写的代码）。
2. 配置好 Go 语言的环境变量，包括 GOPATH、GOPROXY。
3. 搭建好舒适的集成开发环境（GoLand、Vim、VSCode 或者 Emacs），以便快速开发代码。挑选集成开发环境需要考虑的因素很多，主要包括下面几点。

- 有没有语法高亮？语法高亮是必不可少的功能，这也是为什么每个开发工具都提供配置文件，让我们自定义配置的原因。

- 有没有较好的项目文件纵览和导航能力？我们希望可以同时编辑多个源文件并设置书签，能够匹配括号，能够跳转到某个函数或类型的定义部分。
- 有没有完美的查找和替换功能？替换之前最好还能预览结果。
- 当有编译错误时，双击错误提示能否跳转到发生错误的位置？
- 能否跨平台运作？比如，能够在 Linux、Mac OS X 和 Windows 下工作，这样我们就可以只专注于一个开发环境了。

此外，我们还需要确保集成开发环境具有如下功能：

- 能够通过插件架构来轻易扩展和替换某个功能；
- 拥有断点、检查变量值、单步执行、按照过程顺序执行标识库中代码的能力；
- 能够方便地存取最近使用过的文件或项目；
- 拥有对包、类型、变量、函数和方法的智能代码补全功能；
- 能够方便地在不同的 Go 环境之间切换；
- 针对一些特定的项目有项目模板（如 Web 应用、App Engine 项目等），这样能够更快地开始开发工作。

4. 合格的开发者需要熟悉编辑器中的快捷键，例如上移 (Up)、下移 (Down)、右移 (Right)、左移 (Left)、复制当前或选中行（Duplicate Line or Selection）、提取选中内容为函数（Extract Method)，还有众多的快捷键我在这里就不赘述了。

需要提到的是，仅仅凭借经验或者查看官方文档是很难将这些快捷键掌握得足够好。例如，初学者很习惯在鼠标和键盘之间来回切换，而这常常是非常低效的开发方式。所以，我建议你带着问题去学习快捷键。

比如，我需要移动到本行的末尾，比较容易想到的方法是移动鼠标或者按住键盘的右键不放。但如果我们稍微想得更多一点，就会发现有一个使用频率很高的快捷键可以解决这个问题，那就是：Ctrl+E（Move Caret to Line End）。当然还有一些快捷键包含一些技巧，很多人可能压根就不知道还能这么用，例如“让一个类型快速地实现一个接口”。这些技巧就需要系统地看攻略了。

5. 掌握 Go 的一些命令行工具，特别是一些基础的命令。只要在命令行中执行 Go，就有多种子命令可供选择。这些命令及其含义如下：

command	说 明
go bug	打开浏览器，报告错误信息
go build	编译源代码
go clean	移除目标文件和缓存文件
go env	打印Go环境信息
go fix	旧版本代码修正为新版本
go fmt	格式化源文件
go generate	扫描特殊注释，用于自动生成Go文件
go get	添加指定版本的依赖
go list	列出指定代码包的信息
go mod	依赖管理工具
go run	编译并运行源代码
go test	测试代码
go tool	运行特殊的Go工具
go version	打印Go版本
go vet	静态扫描代码，报告代码中可能的错误

基础语法

Go 中要掌握的基础语法和其他的高级语言是类似的。它包括了变量与类型、表达式与运算符、基本控制结构、函数和复合类型。

变量与类型

先来看变量，我们需要掌握的内容如下。

- 变量的声明与赋值。特别是在 Go 函数中使用相当频繁的变量赋值语句 `:=`，其将在编译时对类型进行自动推断。
- Go 中的内置类型。

```
int  int8  int16  int32  int64
uint  uint8  uint16  uint32  uint64  uintptr
float32  float64  complex128  complex64
bool  byte  rune  string
```

- 变量的命名规则。
- 变量的生命周期。需要了解变量何时存在，何时消亡。
- 变量的作用域。Go 的词法范围使用花括号{...}作为分割。根据作用域的范围大小，可以分为全局作用域、包作用域、文件作用域、函数作用域。

表达式与运算符

除了需要对变量有深入理解外，你还需要掌握表达式与运算符，帮助程序完成基础的运算。运算符包括：

- 算术运算符；
- 关系运算符；
- 逻辑运算符；
- 位运算符；
- 赋值运算符；
- 地址运算符。

当这些运算符同时存在时，还需要考虑运算符优先级的顺序：

```
优先级(由高到低)          操作符
    5                  *  /  %  <<  >>  &  &^
    4                  +  -  |  ^
    3                  ==  !=  <  <=  >  >=
    2                  &&
    1                  ||
```


基本控制结构

程序并不都是一行一行顺序执行的，还可能根据条件跳转到其他语句执行，这就涉及到基本控制结构了。理论和实践表明，无论多复杂的算法，都可以通过顺序、选择、循环 3 种基本控制结构构造出来。

下面给出 Go 语言基本控制结构的几种形式：

- if else 语句；


```go
if{

}else if {

}else {

}
```

- switch 语句；


```go
switch var1 {
    case val1:
        ...
    case val2,val3:
        ...
    default:
        ...
}
```

- 4 种 for 循环语句。

1）完整的 C 风格的 for 循环


```
for i := 0; i < 10; i++ {
    fmt.Println(i)
}
```

2）只有条件判断的 for 循环

复制代码

```
i := 1
for i < 100 {
        fmt.Println(i)
        i = i * 2
}
```

3）无限循环的 for 循环

复制代码

```
func main() {
    for {
        fmt.Println("Hello")
    }
}
```

4）for-range 循环

复制代码

```
evenVals := []int{2, 4, 6, 8, 10, 12}
for i, v := range evenVals {
    fmt.Println(i, v)
}
```

函数

高级语言都离不开函数这个概念。通过函数，我们可以给一连串的复合操作定义一个名字，把它们作为一个操作单元。函数是一种定义过程的强大抽象技术，它可以帮助我们构建大规模的

程序。在 Go 语言中，函数也具有一些和其他语言不太一样的特性，例如函数是一等公民。对于函数的基本用法，需要掌握：

- 基本的函数声明；

```
func name(parameter-list) (result-list) {
    body
}
```

- 函数的多返回值特性；

```
func div (a,b int) (int,error){
    if b == 0 {
     return 0, errors.New("b cat't be 0")
    }
    return a/b,nil
}
```

- 可变参数函数。

```
func Println(a ...interface{}) (n int, err error)
```

另外，我们还需要掌握像递归这样复杂的函数形式，了解它的使用场景和执行过程。

```
func f(n int) int {
  if n == 1 {
    return 1
  }
  return n * f(n-1)
}
```

函数作为一等公民拥有一些灵活的特性。

- 函数作为参数时，可以提升程序的扩展性。


```
package main
import (
    "fmt"
)
// 遍历切片的每个元素，通过给定函数进行元素访问
func visit(list []int, f func(int)) {
    for _, v := range list {
        f(v)
    }
}

func main() {
    // 使用匿名函数打印切片内容
    visit([]int{1, 2, 3, 4}, func(v int) {
        fmt.Println(v)
    })
}
```

- 函数作为返回值时，一般在闭包和构建功能中间件时使用得比较多，在不修改过去核心代码的基础上，用比较小的代价增加了新的功能。


```
func logging(f http.HandlerFunc) http.HandlerFunc{
  return func(w http.ResponseWriter, r *http.Request) {
    log.Println(r.URL.Path)
    f(w,r)
  }
}
```

- 函数作为值时，可以用来提升服务的扩展性。


```
var opMap = map[string]func(int, int) int{
    "+": add,
    "-": sub,
    "*": mul,
    "/": div,
}

f := opMap[op]
```

```
9  f()
```

最后要强调的是，Go 函数调用时参数是值传递，在调用过程中修改函数参数不会影响到原始的值。

复合类型

如果说函数是对功能的抽象，那么复合类型带来了数据的抽象。

高级程序语言帮助我们把简单类型组合起来，形成了复合类型。复合类型将我们对程序设计的抽象提到一个新的高度。它增加了程序的模块化程度，并增强了语言的表达能力。例如，我们要处理分数。分数有分子和分母之分，如果我们只有基础的数据类型，这种处理将变得繁琐。而如果有了复合类型。我们就可以将分子和分母看做一个整体了。

为了形成复合数据，编程语言应该提供某种“胶水”，方便将数据对象组合起来，形成更复杂的数据对象。复合类型正是这样的胶水，它让程序更易于设计、维护和修改。

Go 语言中内置的复合类型包括：**数组、切片、哈希表，以及用户自定义的结构体。**

相比于数组，在 Go 语言中使用最多的是切片。切片基础语法需要掌握下面这些内容。

- 声明与赋值。

复制代码

```
1  var slice1 []int
2  numbers:= []int{1,2,3,4,5,6,7,8}
3  var x = []int{1, 5: 4, 6, 10: 100, 15}
```

- 使用 append 往切片中添加元素。

复制代码

```
1  y := []int{20, 30, 40}
2  x = append(x, y...)
```

- 切片的截取。

```
numbers:= []int{1,2,3,4,5,6,7,8}
// 从下标2 一直到下标4，但是不包括下标4
numbers1 :=numbers[2:4]
// 从下标0 一直到下标3，但是不包括下标3
numbers2 :=numbers[:3]
// 从下标3 一直到结尾
numbers3 :=numbers[3:]
```


当然，在 Go 语言中，切片其实是非常容易犯错的数据结构。但这需要结合切片的原理才能理解得比较深刻，在后面的课程中，我还会详细介绍。

对于 Map 哈希表，我们需要掌握哈希表的优势和它的使用场景，掌握它的基本语法。

- Map 声明与初始化。


```
var hash map[T]T
var hash = make(map[T]T,NUMBER)
var country = map[string]string{
"China": "Beijing",
"Japan": "Tokyo",
"India": "New Delhi",
"France": "Paris",
"Italy": "Rome",
}
```

- Map 的两种访问方式。


```
v := hash[key]
v,ok := hash[key]
```

- Map 赋值与删除。


```
m := map[string]int{
    "hello": 5,
    "world": 10,
```

```
}
delete(m, "hello")
```


自定义结构体是对程序进行数据抽象、提高编程语言的表达能力的强有力的工具。

例如，假设现在我们想实现一个分数的加法逻辑。如果没有自定义结构体的抽象，可能的实现方式是下面这样：


```
func add(n1 int,d1 int,n2 int,d2 int) (int,int){
    return (n1*d2 + n2*d1), (d1*d2)
}
```

在这里，函数的参数是一长串的分子与分母。

当我们调用 add 函数时，还需要保证正确地传递了每一个参数，例如第一个参数为第一个数字的分子，第二个参数为第二个数字的分母……这也意味着 add 函数的调用者不仅仅需要小心地排列其传递的参数，还需要关注 add 函数内存的执行和返回的细节。自定义结构体可以为我们解决这样的问题。

这时候，我们就需要自定义结构体并掌握它的基本语法了。这些基本语法包括：

- 结构体声明与赋值；


```
type Nat struct {
    n  int
    d  int
}
var nat Nat
nat := Nat{
    2,
    3
}
nat.n = 4
natq := Nat{
    d:  3,
    n:  2,
}
```

- 匿名结构体，经常在测试或者在 JSON 序列化反序列化等场景使用；


```
var person struct {
    name string
    age  int
    pet  string
}

pet := struct {
    name string
    kind string
}{
    name: "Fido",
    kind: "dog",
}
```

- 结构体的可比较性；

类 型	可比较性	说 明
布尔	可比较	-
整数	可比较	-
浮点	可比较	-
复数	可比较	-
字符串	可比较	-
指针	可比较	如果两个指针指向相同的变量，或者两个指针均为nil，则它们相等。
通道	可比较	如果两个通道是由相同的make函数调用创建的，或者两个通道都为nil，则它们相等。
接口	可比较	如果两个接口具有相同的动态类型和相同的动态值，或者两个接口都为nil，则它们相等。
结构体	可比较	如果结构体的所有字段都是可比较的，则它们的值是可比较的。
数组	可比较	如果数组元素的类型可比较，则数组可比较。如果两个数组对应的元素都相等，则它们相等。
切片	不可比较	-
函数	不可比较	-
哈希表	不可比较	-

同时，我们也需要了解使用结构体的场景，掌握应该何时使用结构体来构建复杂数据模型的抽象。

总结

Go 语言专注于软件开发过程本身，致力于成为设计大规模软件项目的优秀工具，为软件开发提供生产力和扩展性。这节课，我们回顾了 Go 语言的历史、诞生背景、发展历程与设计理念，温习了 Go 语言的开发环境和基础语法。开发环境和基础语法只是 Go 基础知识体系中的两环。除此之外，我们还应该掌握语法特性、并发编程、项目组织、工具与库这些内容，这也是我们下节课的重点。

课后题

最后，我也给你留一道思考题。

假设我们需要学习一门新的语言，你认为我们需要把握哪些核心的知识，又应该沿着什么样的路径去学习呢？

欢迎你在留言区与我交流讨论，我们下节课再见！

上一篇 开篇词丨聚沙成塔，构建高性能、分布式爬虫项目

下一篇 02丨内有乾坤：Go语言六大基础知识体系

精选留言 (6)

自由 康

2022-10-12 来自北京

学习一门新语言，我认为首先是动手。如果有其他语言的基础，那么重点是掌握它的编程惯例，也就是这么语言与其他语言的不同点以及最佳实践。学习 Go，官方的白皮书是必读的，同时官方的博客也写的很好。

1.我学习 Go，首先是 https://quii.gitbook.io/learn-go-with-tests/ 这个项目，强力推荐，边写边学，了解作者提到的变量声明、方法、函数、结构体、复合结构等，还能学习 TDD 的编程模式。

这个项目 1 天就能练完。

2.接下来就是看官方白皮书，也就是 《Go语言圣经》https://books.studygolang.com/gopl-zh/，6 天就能读完。

3.此时自己肯定摩拳擦掌，想要学一个框架，做一个小项目练练手（作者就是带你做项目，哈哈），我个人不推荐 Gee，不知道为什么，好多初学者都先回了解到 Gee 这个项目，这个项

目已经是僵尸项目了，同时它做的事情就是 HTTP 相关的，例如前缀路由树、分组控制等。想写项目练习，巩固 Go ，我推荐学习极客兔兔的《七天用Go从零实现系列》https://geektutu.com/post/gee.html，选择感兴趣的写一写吧！

4. Go 里面并没有像 Java Spring 一样“一统天下”的框架（或许是因为很多都是 Java 转语言的新 Gopher，所以我这样说?），学习框架，我首先推荐的是Go标准库，当然，标准库没有扎实的基础很难读懂，但是它很巧妙，你可以把阅读所有标准库，作为你的目标，言归正传，框架我推荐 Bilibil 开源的 Kratos 项目，https://go-kratos.dev/en/docs/，大厂背书、丰富的技术选型、活跃的社区、优秀的工程化思想。

5. 接下来，推荐阅读 《Go 语言高级编程》，它的内容，对于面试造火箭的你一定有不小的帮助，重点阅读 Channl 和 Slice，对于GC垃圾回收、运行时、内存逃逸我目前没有阅读到特别通俗易懂的作品。

6. 极客时间毛剑老师《Go 进阶训练营》

7. Happy Coding! 参加开源项目! 提高自己的技术影响力，为 China Gopher 发出更多的声音!

作者回复:

共 3 条评论 >

21

jeffery

2022-10-11 来自北京

买就对了！干货十足！配合go底层原理剖析更nice

作者回复: 冲鸭！

3

看海

2022-10-13 来自北京

学习一门新语言，我会选择先会用，然后再去看源码，了解其内部执行原来，包括但不限于编译过程，垃圾回收，运行时分析等等

1

Seven

2022-10-13 来自四川

Q1: 需要把握哪些核心的知识?

A1: 基本的语法这个是必须的；对比自己已掌握的语言，观察分析异同；思考语言诞生背景，应用场景，来看语言提供的API ，来纵览语言设计者的机制设计理念。

Q2: 应该沿着什么样的路径去学习呢？

A2: 找一个可以实践的项目来学习，如果一个语言不好玩，感觉没有学下去的必要。强迫自己去做一件不认可、得不到满足和反馈的事，挺没劲的。

1

大梧桐树

2022-10-12 来自广东

我觉得核心的东西，线程，内存模型，垃圾回收，编译方式，网络编程，调试定位，测试。学习路径就看个人，有人喜欢由浅入深，有人喜欢一上来就看源码，也有人从入门到放弃

1

徐海浪

2022-10-12 来自广东

思考题: 学习语言基本语法，语言适用场景和特性，然后在做项目实战中深入学习。

1