1.开发环境搭建

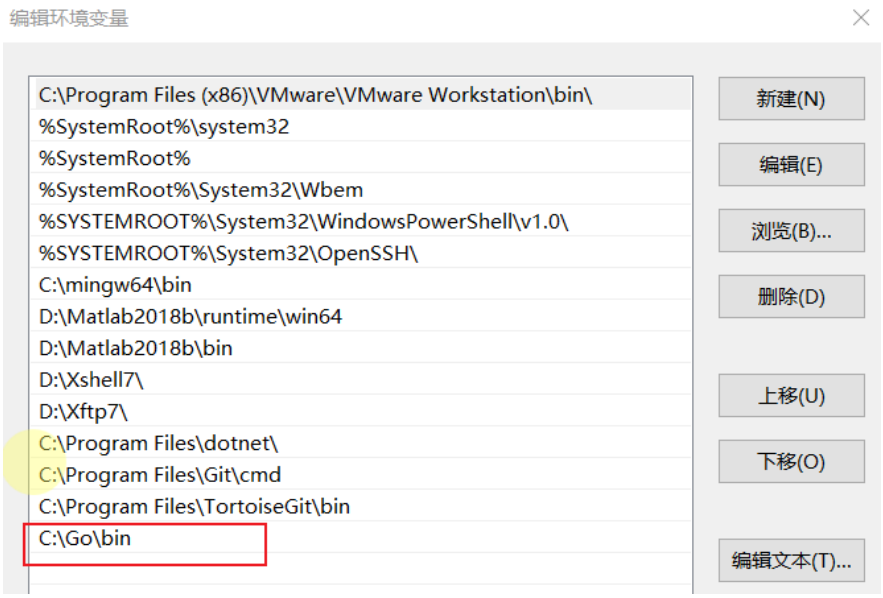
windows平台（我手上只有win设备）

1.安装包下载地址All releases - The Go Programming Language (google.cn)

windows平台的文件后缀为 **.msi**

2.下载完后进行安装，可以自定义路径进行安装，我选择的是C:\Go

3.安装目录下的bin目录，配置到系统变量中的path中，我的安装目录是C:\Go\bin



4.配置GOPATH，GOPATH的变量值即为存储Go语言项目的路径。在 GOPATH 指定的工作目录下，代码总是会保存在 $GOPATH/src 目录下。在工程经过 go build、go install 或 go get 等指令后，会将产生的二进制可执行文件放在 $GOPATH/bin 目录下，生成的中间缓存文件会被保存在 $GOPATH/pkg 下。

5.在网上查找的都是老版本的教程，在新版本中GOROOT目录和GOPATH 目录不要创建同一目录下。GOROOT为go语言源码所在目录，一般都是安装Go的时候自动生成了，不需要再改。

6.配置 GOPROXY，代理配置。Go默认的GOPROXY的值是：GOPROXY=https://proxy.golang.org,direct。这个goproxy在使用go get安装第三方库的时候会报错，导致无法下载成功，所以必须要修改一下。比如改为：https://goproxy.io,direct

在cmd里运行下面的两行

```
go env -w GO111MODULE=on
go env -w GOPROXY=https://goproxy.io,direct
```

7.项目目录配置，在进行Go语言开发的时候，我们的代码总是会保存在$GOPATH/src目录下。在工程经过go build、go install或go get等指令后，会将下载的第三方包源代码文件放在$GOPATH/src目录下， 产生的二进制可执行文件放在 $GOPATH/bin目录下，生成的中间缓存文件会被保存在 $GOPATH/pkg 下。

如果我们使用版本管理工具（Version Control System，VCS。常用如Git）来管理我们的项目代码时，我们只需要添加$GOPATH/src目录的源代码即可。bin 和pkg 目录的内容无需版本控制。当然我们也可以在任意目录创建项目。

2.开发工具vscode

1.安装插件go

2.安装插件code runner运行脚本

3.GO语言结构

Go 语言的基础组成有以下几个部分：

包声明

引入包

函数

变量

语句 & 表达式

注释

```
package main // 声明 main 包，表明当前是一个可执行程序

import "fmt" // 导入内置 fmt

func main() { // main函数，是程序执行的入口
    /* 这是我的第一个简单的程序 */
    fmt.Println("Hello, World!")
    // 在终端打印 Hello World!
}
```

程序的各个部分：

1.**package main 定义了包名。**你必须在源文件中非注释的第一行指明这个文件属于哪个包。

package main表示一个可独立执行的程序，每个 Go 应用程序都包含一个名为 main 的包。

2.**import "fmt" 告诉 Go 编译器这个程序需要使用 fmt 包**（的函数，或其他元素），fmt包实现了格式化 IO（输入/输出）的函数。

3.**func main() 是程序开始执行的函数。**main 函数是每一个可执行程序所必须包含的，一般来说都是在启动后第一个执行的函数（如果有 init() 函数则会先执行该函数）。

4.**注释。**方式和C/C++一致，块注释，行注释。

5.**fmt.Println(...) 可以将字符串输出到控制台，并在最后自动增加换行字符 \n。**使用 fmt.Print("hello, world\n") 可以得到相同的结果。

Print 和 Println 这两个函数也支持使用变量，如：fmt.Println(arr)。如果没有特别指定，它们会以默认的打印格式将变量 arr 输出到控制台。

6. 当标识符（包括常量、变量、类型、函数名、结构字段等等）**以一个大写字母开头**，如：Group1，那么使用这种形式的标识符的对象就可以被外部包的代码所使用（客户端程序需要先导入这个包），这被称为导出（像面向对象语言中的 public）；**标识符如果以小写字母开头，则对包外是不可见的，但是他们在整个包的内部是可见并且可用的（像面向对象语言中的protected ）。**

代码组织

Go应用使用**包和模块**组织代码，包对应文件系统就是文件夹，模块就是.go后缀的源文件，一个包中会有多个模块或者多个子包。

文件名与包名没有直接关系，不一定要将文件名与包名定成同一个。

文件夹名与包名没有直接关系，并非需要一致。

同一个文件夹下的文件只能有一个包名，否则编译报错。

GO项目管理工具

早期的Go项目使用GOPATH来管理项目，从golang1.11开始使用gomod管理项目，当然还有第三方模块如govendor等管理项目。

实现步骤：

1. 创建项目。如我在D盘创建GoPro项目 D:\GoPro
2. 初始化项目。打开VSCode，打开项目文件夹，打开终端执行初始化命令 go mod init 项目名：go mod init Golearn。执行成功后会生成go.mod文件。
3. 创建包。如创建hello文件夹。
4. 创建模块。如创建hello.go文件。所有的go模块都是以 ".go" 结尾。
5. 相互调用。就是别的文件可以导入包hello 。 import Gopro/hello
6. 如果操作正确你将在屏幕上看到 "Hello World!" 字样的输出。

执行GO程序

1. 打开编辑器如VSCode，将以上代码添加到编辑器中。
2. 将以上代码保存为 hello.go
3. 打开命令行，并进入程序文件保存的目录中。
4. 输入命令 go run hello.go 并按回车执行代码。
5. 如果操作正确你将在屏幕上看到 "Hello World!" 字样的输出

4.标识符

标识符用来命名变量、类型等程序实体（包括常量、变量、函数、结构体、数组、切片等等）。**一个标识符实际上就是一个或是多个字母(A~Z和a~z)数字(0~9)、下划线_组成的序列，但是第一个字符必须是字母或下划线而不能是数字。标识符区分大小写。**文件名不包含空格或其他特殊字符。

_ 本身就是一个特殊的标识符，被称为空白标识符。它可以像其他标识符那样用于变量的声明或赋值（任何类型都可以赋值给它），但任何赋给这个标识符的值都将被抛弃，因此这些值不能在后续的代码中使用，也不可以使用这个标识符作为变量对其它变量进行赋值或运算。

5.字符串连接

Go 语言的字符串连接可以通过 + 实现：

```
package main

import "fmt"

func main() {
    var name string
    name = "hello world "
    fmt.Println(name+"linx")
}
#实际输出结果：hello world linx
```

6.关键字

Go 代码中会使用到的 25 个关键字或保留字：

Go 语言还有 36 个预定义标识符：

程序一般由关键字、常量、变量、运算符、类型和函数组成。

程序中可能会使用到这些分隔符：括号 ()，中括号 [] 和大括号 {}。
程序中可能会使用到这些标点符号：.、,、;、:和 …。
程序的代码通过语句来实现结构化。每个语句不需要像 C 家族中的其它语言一样以分号 ; 结尾，因为这些工作都将由 Go 编译器自动完成。
如果你打算将多个语句写在同一行，它们则必须使用 ; 人为区分，但在实际开发中我们并不鼓励这种做法。

7.GO的命名规范

Go是一门区分大小写的语言。

命名规则涉及变量、常量、全局函数、结构、接口、方法等的命名。 Go语言从语法层面进行了以下限定：任何需要对外暴露的名字必须以大写字母开头，不需要对外暴露的则应该以小写字母开头。
1. 当命名（包括常量、变量、类型、函数名、结构字段等等）以一个大写字母开头，如：Analysize，那么使用这种形式的标识符的对象就可以被外部包的代码所使用（客户端程序需要先导入这个包），这被称为导出（像面向对象语言中的 public）；
2. **命名如果以小写字母开头，则对包外是不可见的**，但是他们在整个包的内部是可见并且可用的（像面向对象语言中的 private ）

包名称

保持package的名字和目录保持一致，尽量采取有意义的包名，简短，有意义，尽量和标准库不要冲突。包名应该为**小写单词**，不要使用下划线或者混合大小写。

```
package domain
package main
```

文件命名

尽量采取有意义的文件名，简短，有意义，**应该为小写单词，使用下划线分隔各个单词。**

```
approve_service.go
```

结构体命名

1. 采用**驼峰命名法**，首字母根据访问控制大写或者小写

2. struct 申明和初始化格式采用多行，例如下面：

```
type MainConfig struct {
    Port string `json:"port"`
    Address string `json:"address"`
}
config := MainConfig{"1234", "123.221.134"}
```

接口命名

1. 命名规则基本和上面的结构体类型
2. 单个函数的结构名以 “er” 作为后缀，例如 Reader , Writer 。

```
type Reader interface {
    Read(p []byte) (n int, err error)
}
```

变量命名

和结构体类似，变量名称一般遵循驼峰法，首字母根据访问控制原则大写或者小写，但遇到特有名词时，需要遵循以下规则：

1. 如果变量为私有，且特有名词为首个单词，则使用小写，如 appService
2. 若变量类型为 bool 类型，则名称应以 Has, Is, Can 或 Allow 开头

```
var isExist bool
var hasConflict bool
var canManage bool
var allowGitHook bool
```

常量命名

常量均需使用全部大写字母组成，并使用下划线分词

```
const APP_URL = "https://www.baidu.com"
```

如果是枚举类型的常量，需要先创建相应类型：

```
type Scheme string
const (
    HTTP Scheme = "http"
    HTTPS Scheme = "https"
)
```

错误处理

1.错误处理的原则就是不能丢弃任何有返回err的调用，不要使用 _ 丢弃，必须全部处理。接收到错误，要么返回err，或者使用log记录下来。
2.尽早return：一旦有错误发生，马上返回
3.尽量不要使用panic，除非你知道你在做什么
4.错误描述如果是英文必须为小写，不需要标点结尾
5.采用独立的错误流进行处理

```
// 错误写法
if err != nil {
    // error handling
} else {
    // normal code
}
// 正确写法
if err != nil {
    // error handling
    return // or continue, etc.
}
// normal code
```