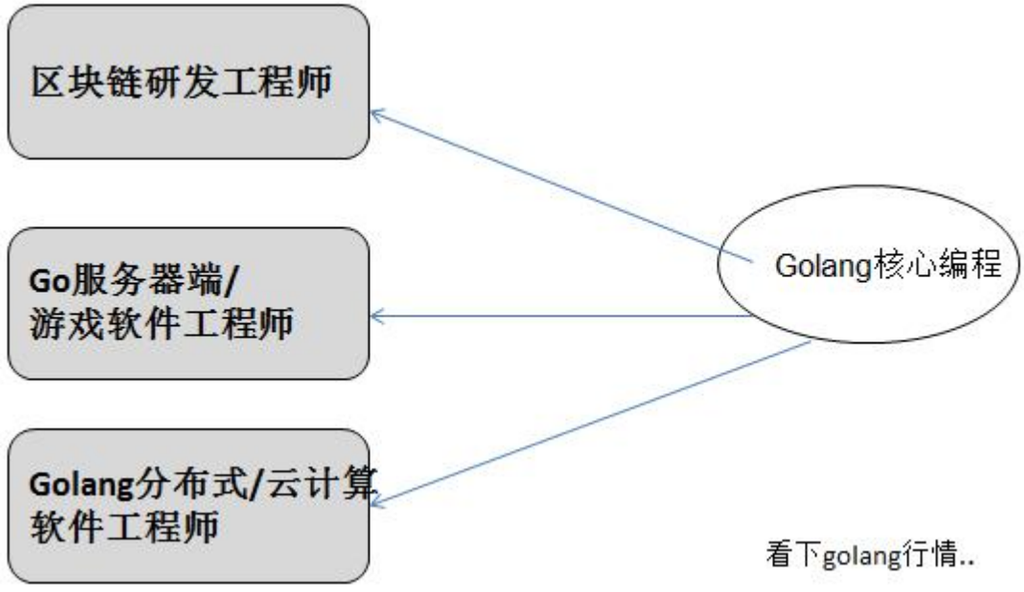
第 1 章 Golang 开山篇

1.1 Golang 的学习方向

Go 语言，我们可以简单的写成 Golang.



1.2 Golang 的应用领域

1.2.1区块链的应用开发

区块链应用

区块链技术，[1] 简称BT（Blockchain technology），也被称之为**分布式账本技术**，是一种互联网数据库技术，其特点是去**中心化**、**公开透明**，让**每个人均可参与数据库记录**

1.2.2后台的服务应用

后端服务器应用

美团后台流量支撑程序

支撑主站后台流量（**排序，推荐，搜索等**），提供**负载均衡，cache，容错，按条件分流**，统计运行指标（qps，latency）等功能-》Golang

仙侠道

产品网址：仙侠道官网 - 心动游戏

应用范围： 游戏服务端（通讯、逻辑、数据存储）

1.2.3云计算/云服务后台应用

云计算/云服务后台应用

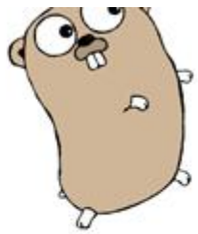
盛大云CDN (内容分发网络)
网址：盛大云计算
应用范围：CDN的调度系统、分发系统、监控系统、短域名服务，CDN内部开放平台、运营报表系统以及其他一些小工具等

京东消息推送云服务/京东分布式文件系统
网址: 京东云
应用范围：后台所有服务全部用go实现
golang的计算能力强。

云有新升,未来无限可

1.3 学习方法的介绍

我对学习Go编程方法的理解，**希望我们能够达成共识**:



金花鼠gordon

1) **高效**而**愉快**的学习
2) 先建立一个**整体框架**，然后**细节**
3) 在实际工作中，要培养用到什么，能够快速学习什么能力
4) 先know how ,再know why【工科】
5) 软件编程是一门 "做中学" 的学科 ,不是会了再做，而是做了才会.
6) 适当的**囫囵吞枣**
7) 学习软件编程是在琢磨别人怎么做，而不是我认为应该怎么做的过程

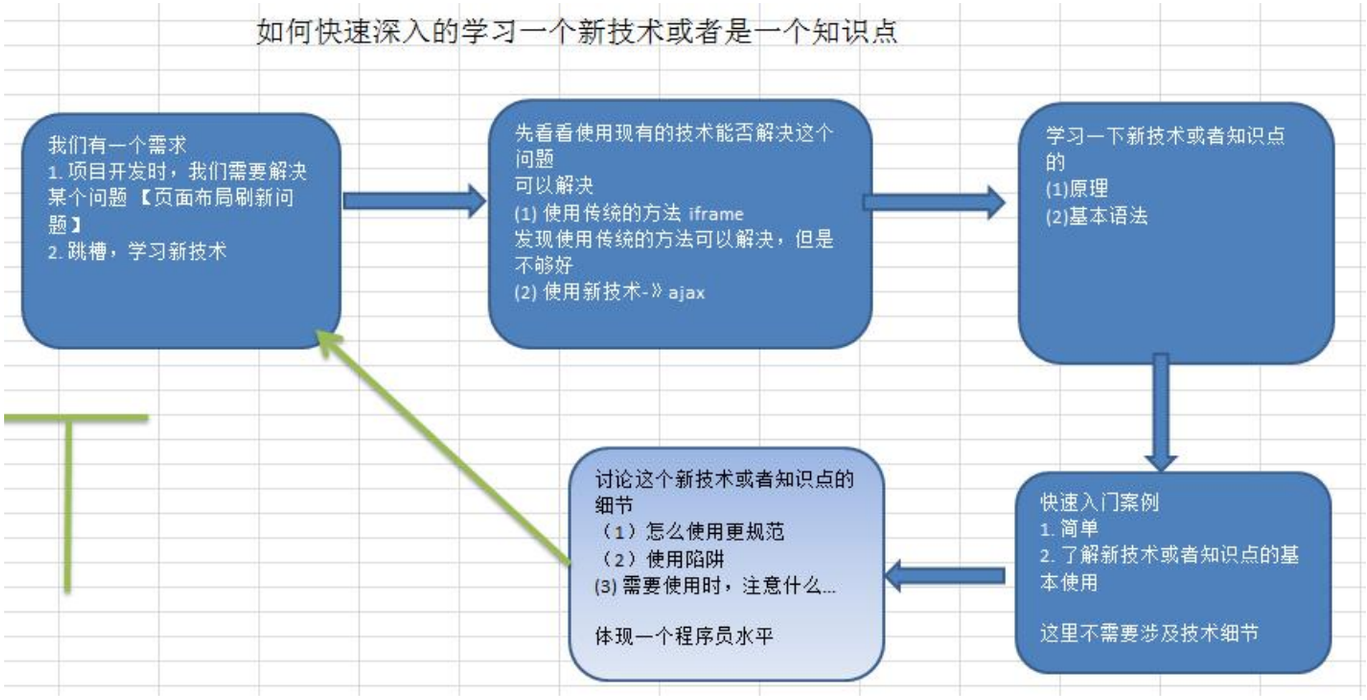
```
for i := 0; i < 10; i++ {
        fmt.Println("hello,world")
}
```

1.4 讲课的方式的说明

1) 努力做到通俗易懂

2) 注重 Go 语言体系，同时也兼顾技术细节

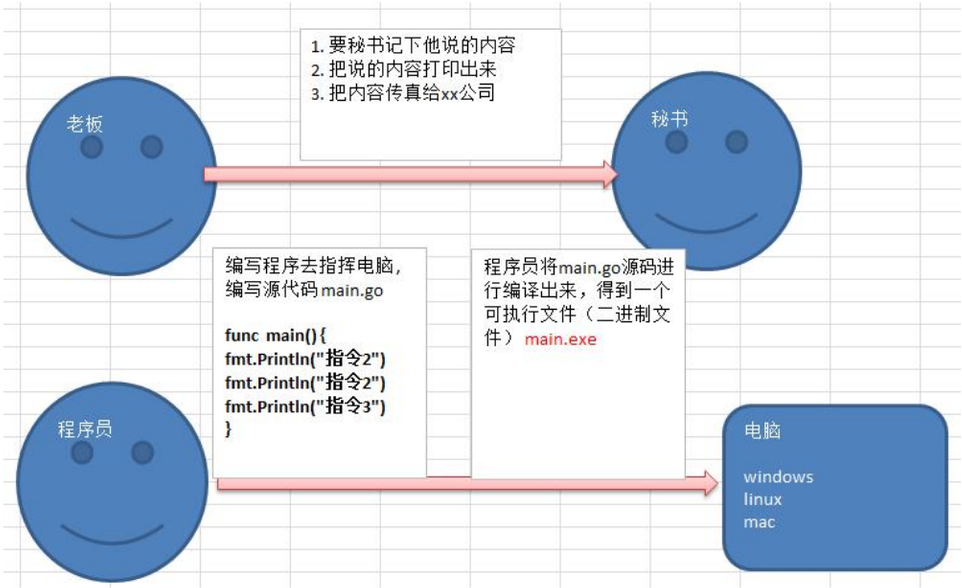
3) 在实际工作中，如何快速的掌握一个技术的分享，同时也是我们授课的思路(怎么讲解或者学习一个技术)。(很多学员反馈非常受用)

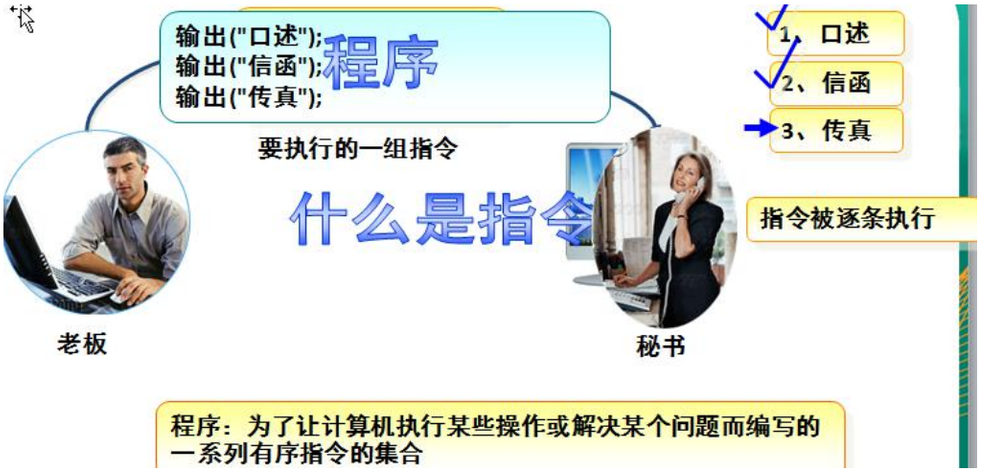


第 2 章 Golang 的概述

2.1 什么是程序

程序：就是完成某个功能的指令的集合。画一个图理解：





2.2 Go 语言的诞生小故事

2.2.1Go 语言的核心开发团队-三个大牛

Ken Thompson(肯·汤普森)：1983年图灵奖（Turing Award）和1998年美国国家技术奖（National Medal of Technology）得主。他与Dennis Ritchie是Unix的原创者。Thompson也发明了后来衍生出C语言的B程序语言，同时也是C语言的主要发明人。2000年

Rob Pike(罗布·派克)：曾是贝尔实验室（Bell Labs）的Unix团队，和Plan 9操作系统计划的成员。他与Thompson共事多年，并共创出广泛使用的UTF-8 字元编码。

Robert Griesemer：曾协助制作Java的HotSpot编译器，和Chrome浏览器的JavaScript引擎V8。

Ken Thompson(肯·汤普森)

Rob Pike(罗布·派克) 1980 奥运会 射箭 银牌 天文学家 【】

2.2.2Google 创造 Golang 的原因

1) 计算机硬**件技术更新频繁**，性能提高很快。目前主流的编程语言发展明显落后于硬件，不能合理利用**多核多CPU**的优势提升软件系统性能。

2) 软件系统复杂度越来越高，维护成本越来越高，目前**缺乏一个足够简洁高效**的编程语言。【现有的编程语言: 1. 风格不统一 2. 计算能力不够 3. 处理大并发不够好】

3) 企业运行维护很多c/c++的项目，c/c++程序运行速度虽然很快，但是编译速度确很慢，同时还存在**内存泄漏**的一系列的困扰需要解决。

2.2.3Golang 的发展历程

- 2007 年，谷歌工程师 Rob Pike, Ken Thompson 和 Robert Griesemer 开始设计一门全新的语言，这是 Go 语言的最初原型。

- **2009** 年 11 月 10 日，Google 将 Go 语言以开放源代码的方式向全球发布。
- **2015** 年 8 月 19 日，Go 1.5 版发布，本次更新中移除了”最后残余的 C 代码”

- 2017 年 2 月 17 日，Go 语言 Go 1.8 版发布。
- 2017 年 8 月 24 日，**Go 语言 Go 1.9 版发布**。 1.9.2 版本
- 2018 年 2 月 16 日，Go 语言 Go 1.10 版发布。

2.3 Golang 的语言的特点

- 简介：

Go 语言保证了既能到达**静态编译语言的安全和性能**，又达到了**动态语言开发维护的高效率**，使用一个表达式来形容 Go 语言：**Go = C + Python**，说明 Go 语言既有 C 静态语言程序的运行速度，又能达到 Python 动态语言的快速开发。

1) 从 C 语言中继承了很多理念，包括表达式语法，控制结构，基础数据类型，调用参数传值，**指针**等等，也保留了和 C 语言一样的编译执行方式及弱化的指针

举一个案例(体验)：

```
//go 语言的指针的使用特点(体验)
func testPtr(num *int) {
    *num = 20
}
```

2) 引入**包的概念**，用于组织程序结构，**Go 语言的一个文件都要归属于一个包**，而不能单独存在。

3) 垃圾回收机制，内存自动回收，不需开发人员管理

4) 天然并发 (**重要特点**)

(1) 从语言层面支持并发，实现简单

(2) goroutine，轻量级线程，可实现大并发处理，高效利用多核。

(3) 基于 CPS 并发模型(Communicating Sequential Processes)实现

5) 吸收了管道通信机制，形成 Go 语言特有的管道 channel 通过管道 channel，可以实现不同的 goroute 之间的相互通信。

6) 函数可以返回多个值。举例：

```
//写一个函数，实现同时返回 和，差
//go 函数支持返回多个值
func getSumAndSub(n1 int, n2 int) (int, int ) {
    sum := n1 + n2 //go 语句后面不要带分号.
    sub := n1 - n2
    return sum , sub
}
```

7) 新的创新：比如切片 slice、延时执行 defer

2.4 Golang 的开发工具的介绍

2.4.1工具介绍

工具介绍：

1) visual studio code Microsoft 产品(VSCode)：一个运行于 Mac OS X、Windows和 Linux 之上的，**默认提供Go 语言的语法高亮**，安装Go语言插件，还可以支持智能提示，编译运行等功能。

2) Sublime Text，可以免费使用，默认也支持Go代码语法高亮，只是保存次数达到一定数量之后就会提示是否购买，点击取消继续用，和正式注册版本没有任何区别

3) Vim：Vim是从vi发展出来的一个**文本编辑器**，代码补全、编译及错误跳转等方便编程的功能特别丰富，在程序员中被广泛使用

4) Emacs：Emacs传说中的神器，她不仅仅是一个编辑器，因为功能强大，可称它为集成开发环境

5) Eclipse IDE工具，开源免费， 并提供GoEclipse插件

6) LiteIDE，LiteIDE是一款专门为**Go语言开发**的跨平台轻量级集成开发环境（IDE），是国人开发的。

7) JetBrains公司的产品：PhpStrom、WebStrom和PyCharm 等IDE工具，都需要安装Go插件。

2.4.2工具选择：

- 如何选择开发工具

我们先选择用 **visual studio code** 或者 vim 文本编辑器本，到大家对 Go 语言有一定了解后，我们再使用 Eclipse 等 IDE 开发工具。

- 这是为什么呢?

1) 更深刻的理解 Go 语言技术,培养代码感。->写代码的感觉。

2) 有利于公司面试。-> 给你纸，写程序

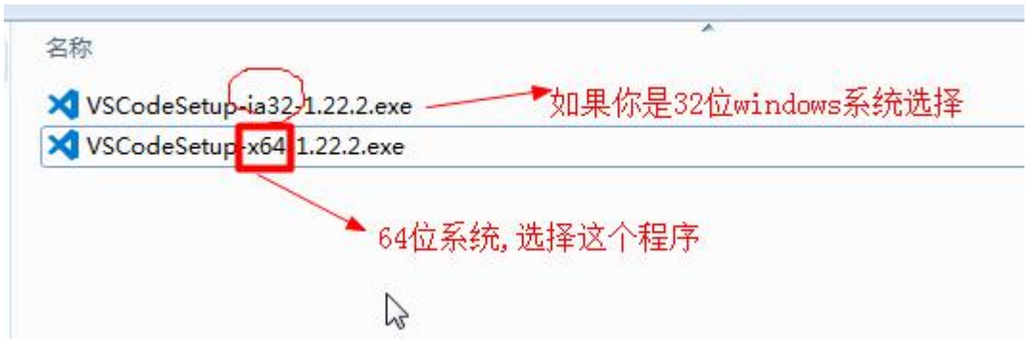
2.4.3 VSCode 的安装和使用

1) 先到下载地址去选择适合自己系统的 VSCode 安装软件

https://code.visualstudio.com/download

2) 演示如何在 windows 下安装 vscode 并使用

步骤 1: 把 vscode 安装文件准备好

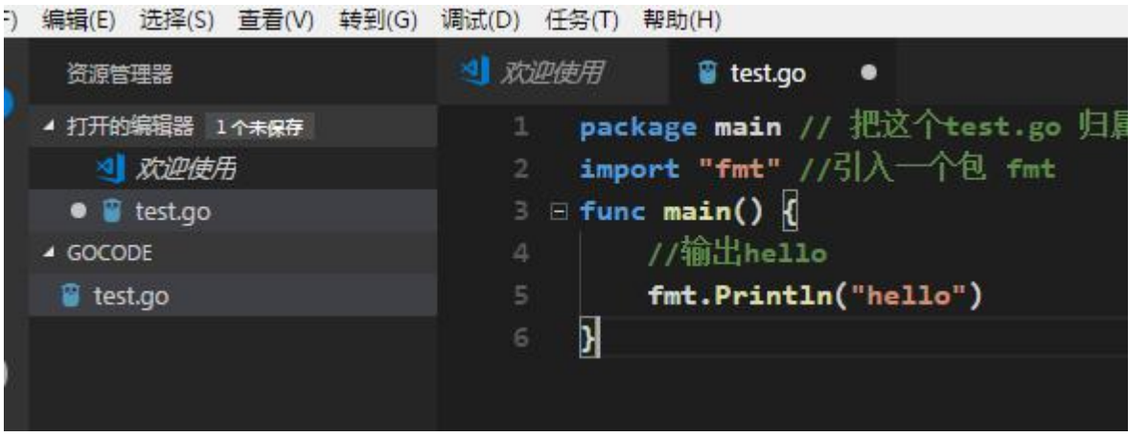


步骤 2：双击安装文件，就可以一步一步安装，我个人的习惯是安装到 d:/programs 目录.

当看到如下界面时，就表示安装成功!

步骤 3: 简单的使用一下 vscode

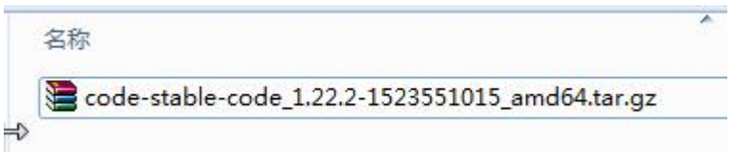
在 d 盘创建了一个文件夹 gocode.



3) 演示如何在 Linux(ubuntu /centos)下安装 vscode 并使用

这里，我们介绍一下我的 linux 的环境:

步骤 1: 先下载 linux 版本的 vscode 安装软件.



步骤 2: 因为我这里使用的是虚拟机的 ubuntu, 因此我们需要先将 vscode 安装软件传输到 ubuntu 下,使用的 xftp5 软件上传。

步骤 3: 如果你是在 ubuntu 下做 go 开发，我们建议将 vscode 安装到 /opt 目录..

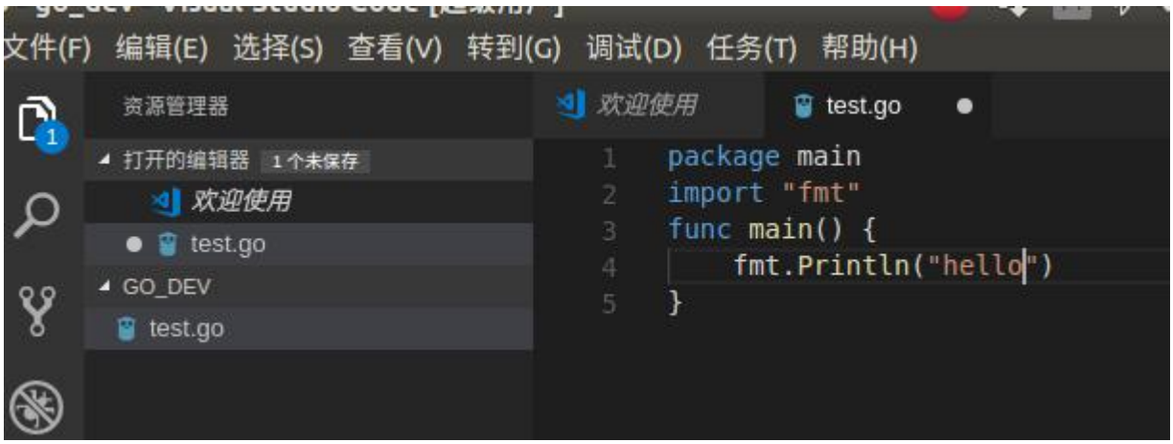
步骤 4：将安装软件拷贝到 /opt

步骤 5: cd /opt 【切换到 /opt】

步骤 6: 将安装文件解决即可

```
root@ubuntu:/opt# tar -zxvf code-stable-code_1.22.2-1523551015_amd64.tar.gz
```

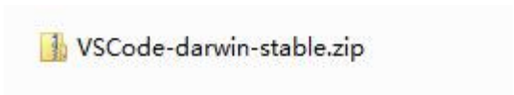
步骤 7: 现在进入解压后的目录，即可运行我们的 vscode



4) 演示如何在 Mac 下安装 vscode 并使用

如果你使用的就是 **mac** 系统，也可以在该系统下进行 go 开发.

步骤 1：下载 mac 版本的 vscode 安装软件



步骤 2：把 vscode 安装软件，传输到 mac 系统

细节：在，默认情况下 mac 没有启动 ssh 服务，所以需要我们启动一下，才能远程传输文件.

mac 本身安装了 ssh 服务，默认情况下不会开机自启

1.启动 sshd 服务：

sudo launchctl load -w /System/Library/LaunchDaemons/ssh.plist

2.停止 sshd 服务：

sudo launchctl unload -w /System/Library/LaunchDaemons/ssh.plist

3 查看是否启动：

sudo launchctl list | grep ssh

如果看到下面的输出表示成功启动了：

————————————————

- 0 com.openssh.sshd

步骤 3：将安装软件解压后即可使用.

进入到这个解压后的文件夹(**图形界面**)，双击即可

步骤 4：编写解简单测试.

在用户所在的目录，创建了 gocode,然后将 test.go 写到这个文件夹下 ..

2.4.4 小结

我们会讲解在 windows， linux , mac 如何安装 vscode 开发工具，并且还会讲解如何在三个系统下安装 go 的 sdk 和如何开发 go 程序。

但是为了学习方便，我们前期选择 Windows 下开发 go。到我们开始讲项目和将区块链时，就会使用 linux 系统。

在实际开发中，也可以在 windows 开发好程序，然后部署到 linux 下。

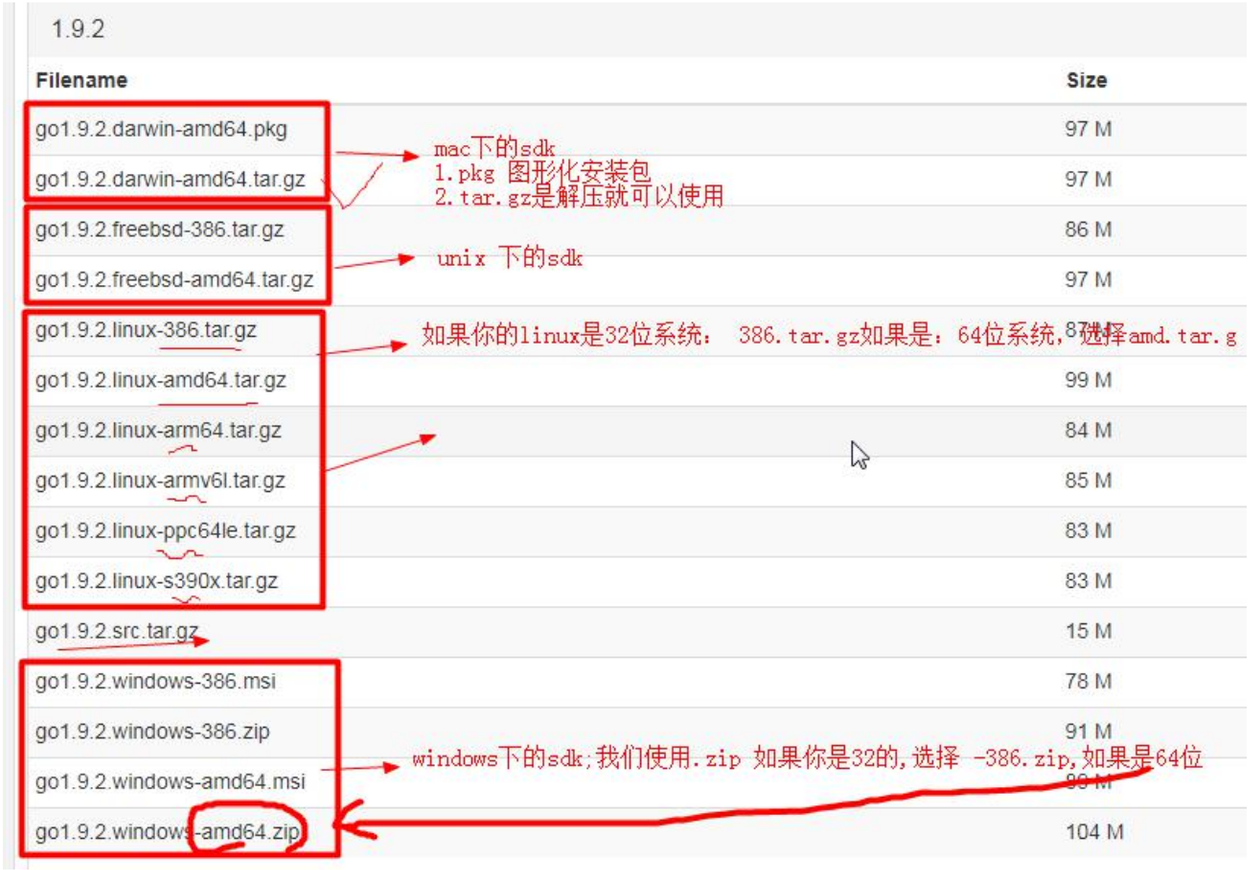
2.5 Windows 下搭建 Go 开发环境-安装和配置 SDK

2.5.1介绍了 SDK

1) SDK 的全称(**S**oftware **D**evelopment **K**it 软件开发工具包)
2) SDK 是提供给开发人员使用的，其中包含了对应开发语言的工具包

2.5.2下载 SDK 工具包

1) Go 语言的官网为：golang.org，因为各种原因，可能无法访问。
2) SDK 下载地址：Golang 中国 https://www.golangtc.com/download
3) 如何选择对应的 sdk 版本



2.5.3windows 下安装 sdk

1) Windows 下 SDK 的各个版本说明：

Windows 下：根据自己系统是 32 位还是 64 位进行下载：

32 位系统：go1.9.2.windows-386.zip

64 位系统：**go1.9.2.windows-amd64.zip**

2) 请注意：安装路径不要有中文或者特殊符号如空格等

3) SDK 安装目录建议：windows 一般我安装在 d:/programs

4) 安装时，基本上是傻瓜式安装，解压就可以使用

5) 安装看老师的演示：

6) 解压后，我们会看到 d:/go 目录，这个是 sdk

如何测试我们的 go 的 sdk 安装成功。

2.5.4windows 下配置 Golang 环境变量：

- 为什么需要配置环境变量

➢ 为什么要配置环境变量

1) 看一个现象

在dos命令行中敲入go，出现错误提示

2) 原因分析

错误原因：当前执行的程序在当前目录下如果不存在，windows系统会在系统中已有的一个名为path的环境变量指定的目录中查找。如果仍未找到，会出现以上的错误提示。所以进入到 go安装路径\bin目录下，执行go，会看到go参数提示信息

- 配置环境变量介绍

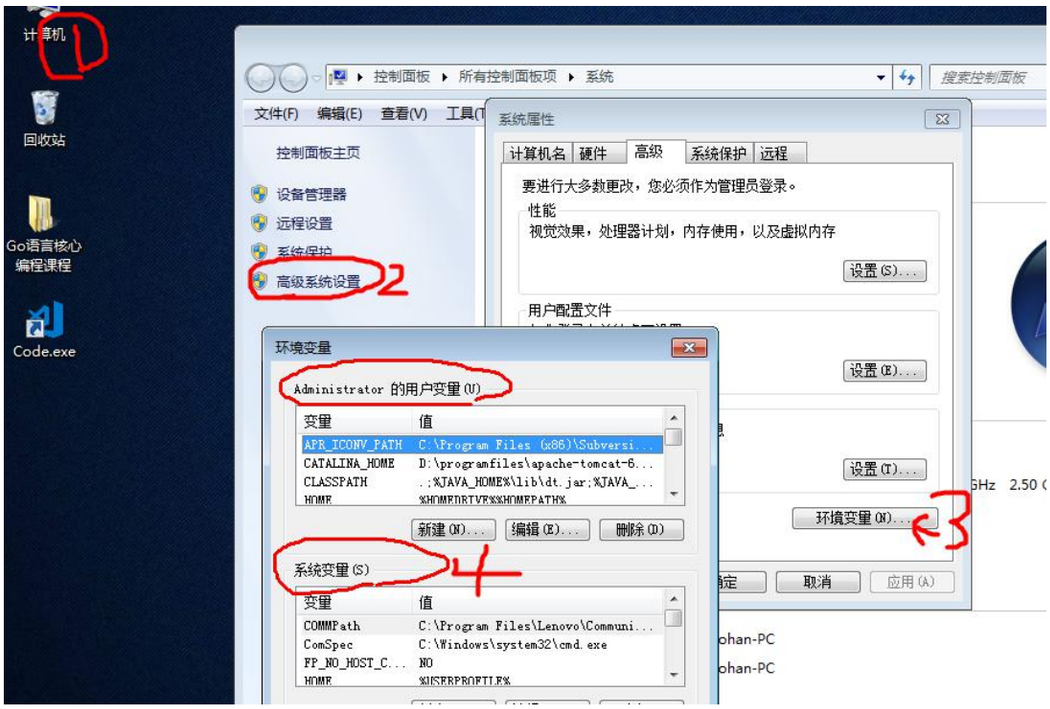
根据 windows 系统在查找可执行程序的原理，可以将 Go 所在路径定义到环境变量中，让系统帮我们去找运行执行的程序，这样在任何目录下都可以执行 go 指令。

- 在 Go 开发中，需要配置哪些环境变量

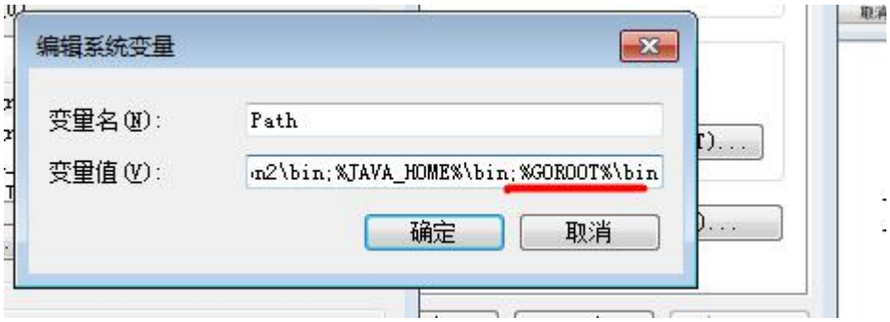
环境变量	说明
GOROOT	指定SDK的安装路径 d:/programs/go
Path	添加SDK的/bin目录
GOPATH	工作目录，将来我们的go项目的工作路径

- 看老师如何配置

步骤 1：先打开环境变量配置的界面



步骤 2: 配置我们的环境变量



对上图的一个说明：

1) Path 这个环境变量不需要在创建，因为系统本身就有，你后面增加即可

2) 增加 Go 的 bin： **;%GOROOT%\bin**

对上图的一个说明

1) GOPATH ：就是你以后 go 项目存放的路径，即工作目录

2) GOPATH：是一个新建的环境变量

- 测试一下我们的环境变量是否配置 ok

```
C:\Users\Administrator>go version
go version go1.9.2 windows/amd64
```

注意：配置环境变量后，需要重新打开一次 dos 的终端，这样环境变量才会生效。

2.6 Linux 下搭建 Go 开发环境-安装和配置 SDK

2.6.1 Linux 下安装 SDK：

1) Linux 下 SDK 的各个版本说明：

Linux 下：根据系统是 32 位还是 64 位进行下载：

32 位系统：go1.9.2.linux-386.tar.gz

64 位系统：**go1.9.2.linux-amd64.tar.gz**

如何确认你的 linux 是多少位：

```
atguigu@ubuntu:~$ uname -a
Linux ubuntu 4.13.0-38-generic #43~16.04.1-Ubuntu SMP Wed Ma
18 x86_64 x86_64 x86_64 GNU/Linux
atguigu@ubuntu:~$
```

2) 请注意：安装路径不要有中文或者特殊符号如空格等

3) SDK 安装目录建议： linux 放在 /opt 目录下

4) 安装时，解压即可，我们使用的是 tar.gz

5) 看老师演示

步骤 1: 将 go1.9.2.linux-amd64.tar.gz 传输到 ubuntu

步骤 2: 将 go1.9.2.linux-amd64.tar.gz 拷贝到 /opt 下

```
igu# cp go1.9.2.linux-amd64.tar.gz  /opt
igu#
```

步骤 3: cd /opt

步骤 4：tar -zxvf go1.9.2.linux-amd64.tar.gz [解压后，就可以看到一个 go 目录]

步骤 5: cd go/bin

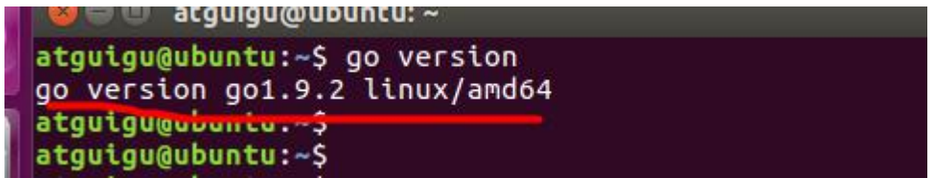
步骤 6：./go version

```
oot@ubuntu:/opt/go/bin# ./go version
go version go1.9.2 linux/amd64
oot@ubuntu:/opt/go/bin#
```

2.6.2Linux 下配置 Golang 环境变量

步骤 1：使用 root 的权限来编辑 vim /etc/profile 文件

步骤 2: 如果需要生效的话，需要注销一下(重新登录)，再使用



2.7 Mac 下搭建 Go 开发环境-安装和配置 SDK

2.7.1mac 下安装 Go 的 sdk

1) Mac 下 SDK 的各个版本说明：

Mac OS 下：只有 64 位的软件安装包

Mac OS 系统的安装包：go1.9.2.darwin-amd64.tar.gz

2) 请注意：安装路径不要有中文或者特殊符号如空格等

3) SDK 安装目录建议： Mac 一般放在用户目录下 go_dev/go 下

4) 安装时，解压即可

5) 看老师的演示步骤

步骤 1： 先将我们的安装文件 go1.9.2.darwin-amd64.tar.gz 上传到 mac

步骤 2: 先在用户目录下，创建一个目录 go_dev，将我们上传的文件 移动到 go_dev 目录

步骤 3： 解压 tar -zxvf go1.9.2.darwin-amd64.tar.gz

步骤 4: 解压后，我们会得到一个目录 go， 进入到 go/bin 就是可以使用

```
atguigudeMac:bin atguigu$ ./go version
go version go1.9.2 darwin/amd64
atguigudeMac:bin atguigu$
```

这里还是有一个问题，就是如果我们不做 bin 目录下，就使用不了 go 程序。因此我们仍然需要配置 go 的环境变量。

2.7.2Mac 下配置 Golang 环境变量：

步骤 1：s 使用 root 用户，修改 /etc/profile 增加环境变量的配置

```
fi
export GOROOT=$HOME/go_dev/go
export PATH=$PATH:$GOROOT/bin
export GOPATH=$HOME/goproject
~
```

步骤 2: 配置完后，需要重新注销用户，配置才会生效.

```
atguigudeMac:~ atguigu$ pwd
/Users/atguigu
atguigudeMac:~ atguigu$ go version
go version go1.9.2 darwin/amd64
atguigudeMac:~ atguigu$
```

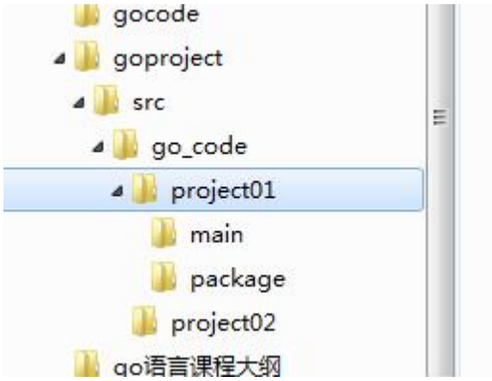
2.8 Go 语言快速开发入门

2.8.1需求

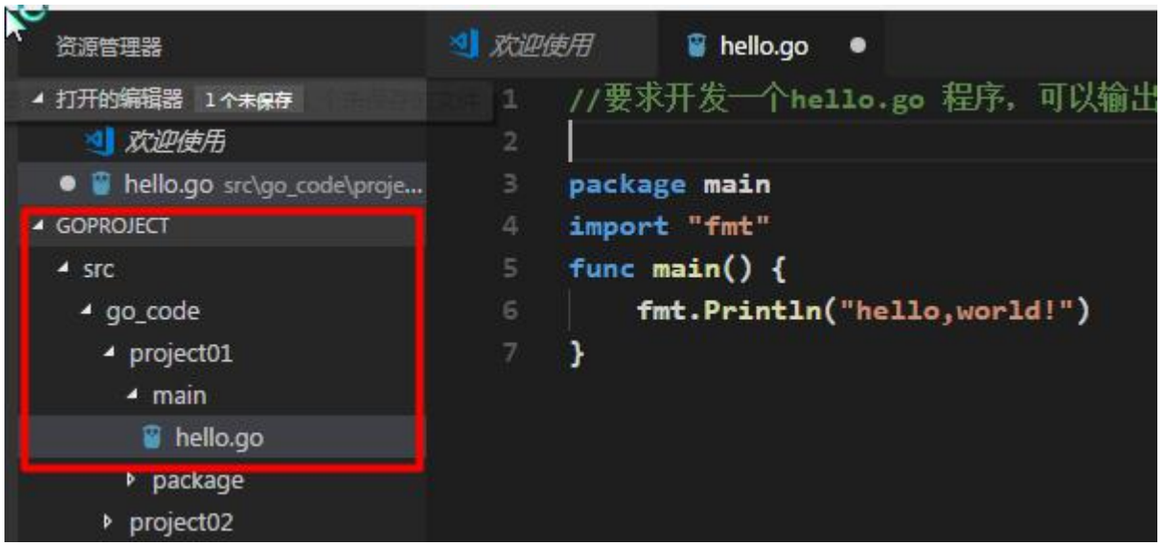
要求开发一个 hello.go 程序，可以输出 "hello,world”

2.8.2开发的步骤

1) 开发这个程序/项目时，go 的目录结构怎么处理.



2) 代码如下：



对上图的说明

(1) go 文件的后缀是 .go

(2) package main

表示该 hello.go 文件所在的包是 main，在 go 中，每个文件都必须归属于一个包。

(3) import “fmt”

表示：引入一个包，包名 fmt，引入该包后，就可以使用 fmt 包的函数，比如：fmt.Println

(4) **func** main() {

}

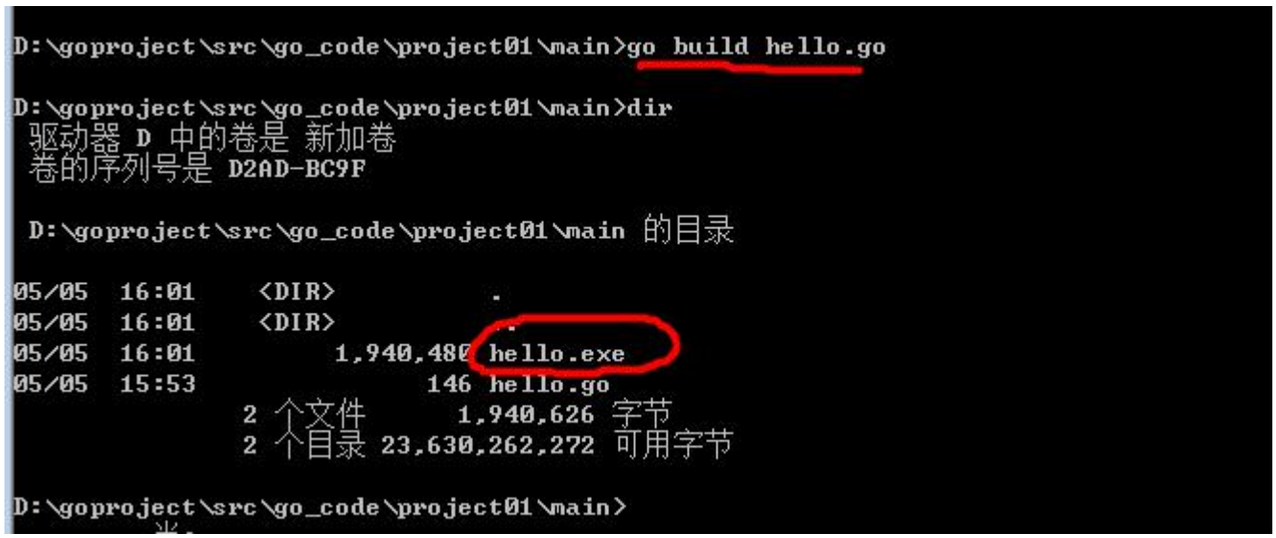
func 是一个关键字，表示一个函数。

main 是函数名，是一个主函数，即我们程序的入口。

(5) fmt.Println(“hello”)

表示调用 fmt 包的函数 Println 输出 “hello,world”

3) 通过 go build 命令对该 go 文件进行编译，生成 .exe 文件.



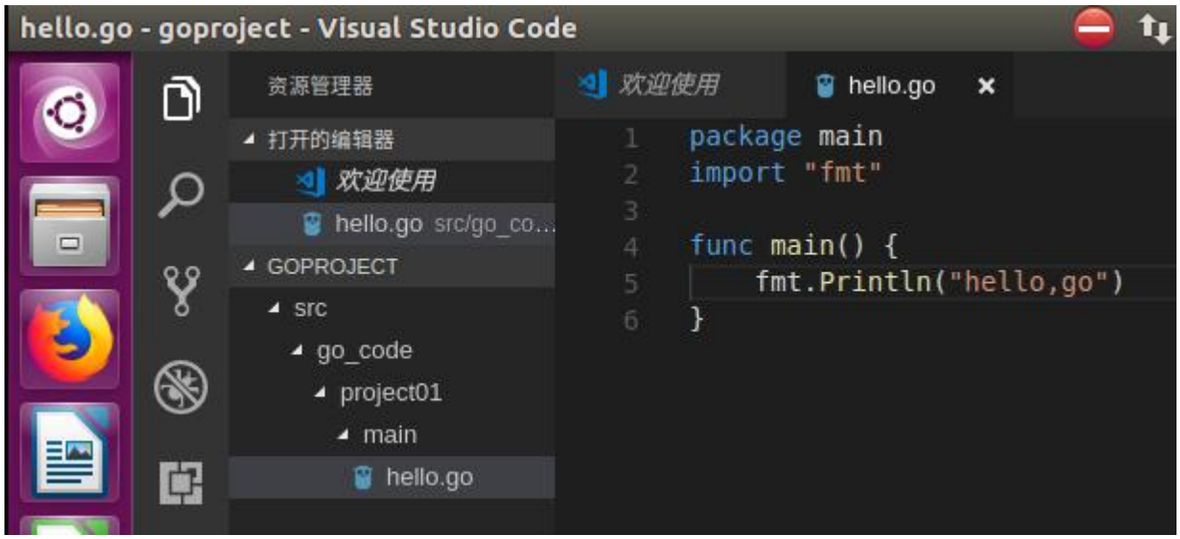
4) 运行 hello.exe 文件即可

5) 注意：通过 go run 命令**可以**直接运行 hello.go 程序 [类似执行一个脚本文件的形式]

2.8.3 linux 下如何开发 Go 程序

说明：linux 下开发 go 和 windows 开发基本是一样的。只是在运行可执行的程序时，是以 **./文件名方式**

演示：在 linux 下开发 Go 程序。



编译和运行 hello.go

```
atguigu@ubuntu:~/goproject/src/go_code/project01/main$ go build hello.go
atguigu@ubuntu:~/goproject/src/go_code/project01/main$ ls
```

```
atguigu@ubuntu:~/goproject/src/go_code/project01/main$ ./hello
hello,go
```

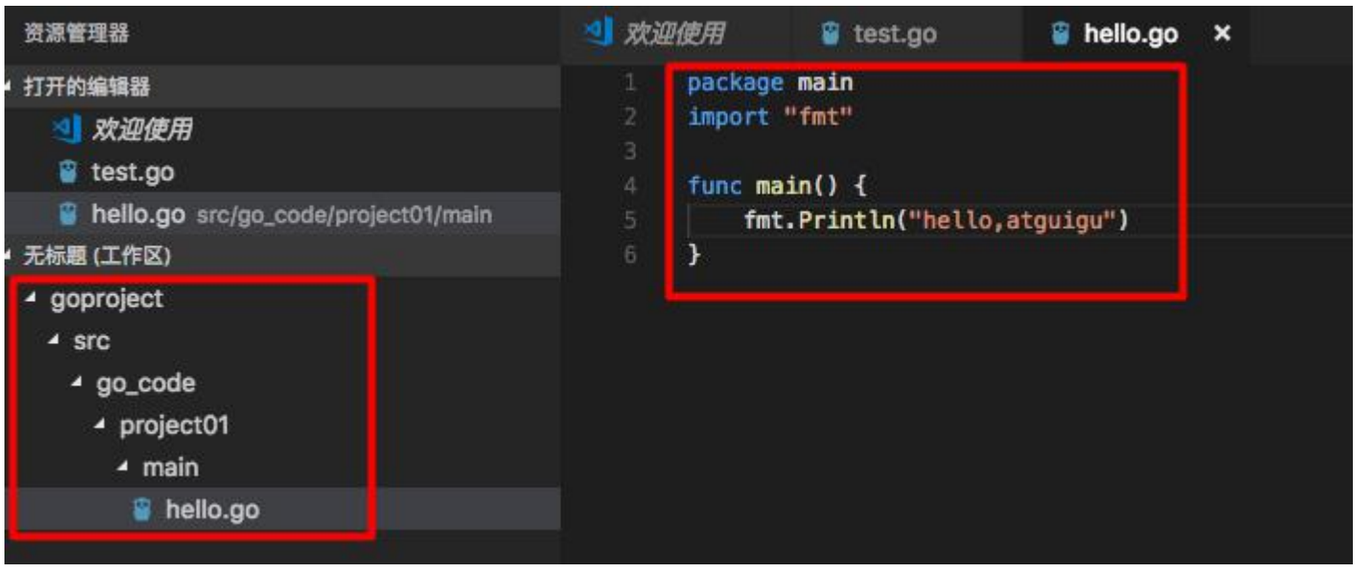
也可以直接使用 go run hello.go 方式运行

```
atguigu@ubuntu:~/goproject/src/go_code/project01/main$ go run hello.go
hello,go
```

2.8.4Mac 下如何开发 Go 程序

- **说明**：在 mac 下开发 go 程序和 windows 基本一样。
- **演示一下**：如何在 mac 下开发一个 hello.go 程序

- 源代码的编写：hello.go



- 编译再运行，直接 go run 来运行

```
Pictures
atguigudeMac:~ atguigu$ cd goproject/src/go_code/project01/main/
atguigudeMac:main atguigu$ ls
hello.go
atguigudeMac:main atguigu$
atguigudeMac:main atguigu$ go build hello.go
atguigudeMac:main atguigu$ ls
hello           hello.go
atguigudeMac:main atguigu$ ./hello
hello,atguigu
atguigudeMac:main atguigu$
```

- 直接 go run 来运行

```
atguigudeMac:main atguigu$ go run hello.go
hello,atguigu
```

2.8.5go 语言的快速入门的课堂练习

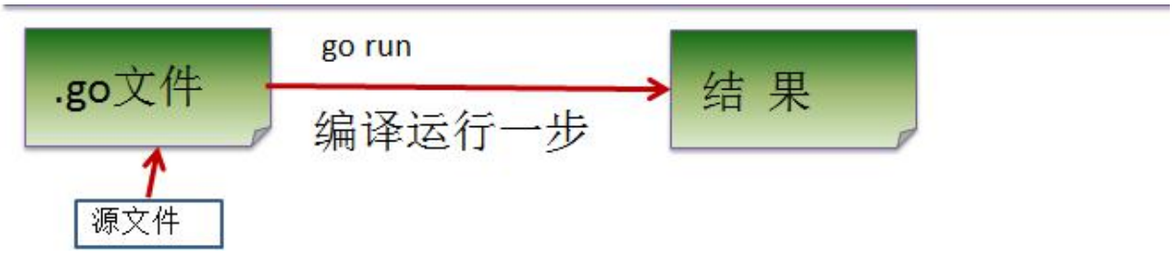
课堂小练习

要求windows下开发一个hi.go 程序，可以输出 "hello,world!" (5min)

2.8.6Golang 执行流程分析

- 如果是对源码编译后，再执行，Go 的执行流程如下图

- 如果我们是对源码直接 执行 go run 源码，Go 的执行流程如下图



- 两种执行流程的方式区别

1) 如果我们先编译生成了可执行文件，那么我们可以将该可执行文件拷贝到没有 go 开发环境的机器上，仍然可以运行

2) 如果我们是直接 go run go 源代码，那么如果要在另外一个机器上这么运行，也需要 go 开发环境，否则无法执行。

3) 在编译时，编译器会将程序运行依赖的库文件包含在可执行文件中，所以，可执行文件变大了很多。

2.8.7编译和运行说明

1) 有了 go 源文件，通过编译器将其编译成机器可以识别的二进制码文件。

2) 在该源文件目录下，通过 go build 对 hello.go 文件进行编译。可以指定生成的可执行文件名，在 windows 下 必须是 .exe 后缀。

```
D:\goproject\src\go_code\project01\main>go build -o myhello.exe hello.go
```

3) 如果程序没有错误，没有任何提示，会在当前目录下会出现一个可执行文件(**windows 下是.exe Linux 下是一个可执行文件**)，该文件是二进制码文件，也是可以执行的程序。

4) 如果程序有错误，编译时，会在错误的那行报错。有助于程序员调试错误.

```
D:\goproject\src\go_code\project01\main>go build  hello.go
# command-line-arguments
.\hello.go:7:2: undefined: fmt.Printnl
```

5) 运行有两种形式

什么是运行

```
atguigu@ubuntu:~/gocode$
atguigu@ubuntu:~/gocode$ ./hello
hello,world!
atguigu@ubuntu:~/gocode$
```

```
D:\programfiles\gocode>hello.exe
hello,world!
```

1) 直接运行生成的可执行Go程序，比如hello.exe
2) 通过运行工具go run 对源代码文件进行运行。

2.8.8Go 程序开发的注意事项

1) Go 源文件以 "go" 为扩展名。

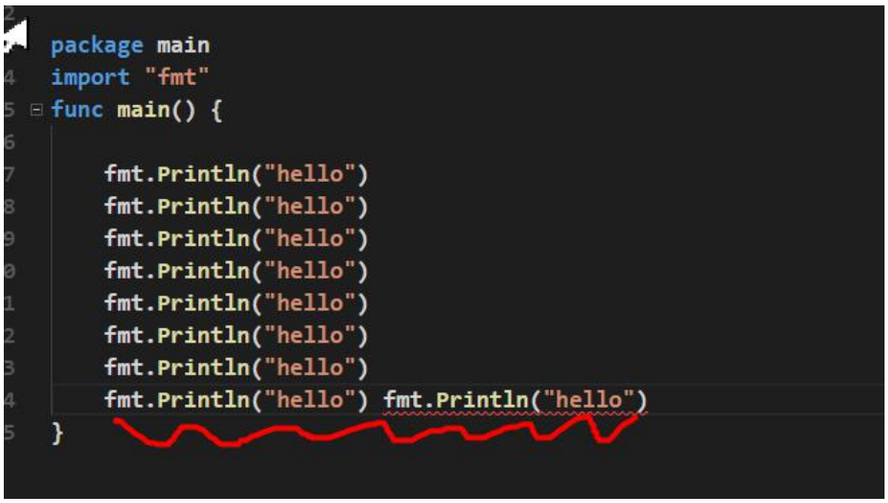
2) Go 应用程序的执行入口是 main()函数。 这个是和其它编程语言（比如 java/c）

3) Go 语言严格区分大小写。

4) Go 方法由一条条语句构成，**每个语句后不需要分号**(Go 语言会在每行后自动加分号)，这也体

现出 Golang 的简洁性。

5) Go 编译器是一行行进行编译的，因此我们一行就写一条语句，不能把多条语句写在同一个，否则报错



6) go 语言**定义的变量**或者 **import 的包**如果没有使用到，代码不能编译通过。

7) 大括号都是成对出现的，缺一不可。

2.9 Go 语言的转义字符(escape char)

说明:常用的转义字符有如下:

1) \t : 表示一个制表符，通常使用它可以排版。

```
import "fmt" //fmt包中提供格式化，输出，输入的函数.
func main() {
    //演示转义字符的使用 \t
    fmt.Println("tom\tjack")
}
```

2) \n ：换行符

3) \\ ：一个\

4) \" ：一个"

5) \r ：一个回车 fmt.Println("天龙八部雪山飞狐\r 张飞");

6) 案例截图

```
package main

import "fmt" //fmt包中提供格式化，输出，输入的函数.
func main() {
    //演示转义字符的使用 \t
    fmt.Println("tom\tjack")

    fmt.Println("hello\nworld")
    fmt.Println("C:\\Users\\Administrator\\Desktop\\Go语言核心编程课程\\资料")
    fmt.Println("tom说\"i love you\"")
    // \r 回车,从当前行的最前面开始输出，覆盖掉以前内容
    fmt.Println("天龙八部雪山飞狐\r张飞厉害")
}
```

- 课堂练习

要求：请使用一**句输出语句**，达到输入如下图形的效果：

```
姓名    年龄    籍贯    住址
john    12      河北    北京
```

5min

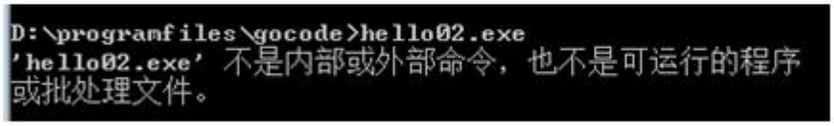
作业评讲：

```
package main
```

```
import "fmt" //fmt 包中提供格式化，输出，输入的函数.
func main() {
    //要求：要求：请使用一句输出语句，达到输入如下图形的效果
    fmt.Println("姓名\t 年龄\t 籍贯\t 地址\njohn\t20\t 河北\t 北京")
}
```

2.10 Golang 开发常见问题和解决方法

2.10.1 文件名或者路径错误



解决方法:源文件名不存在或者写错，或者当前路径错误

2.10.2 小结和提示

学习编程最容易犯的错是语法错误 。Go 要求你必须按照语法规则编写代码。如果你的程序违反了语法规则，例如：忘记了大括号、引号，或者拼错了单词，Go 编译器都会报语法错误，**要求：尝试着去看懂编译器会报告的错误信息**。

2.11 注释(comment)

2.11.1 介绍注释

用于注解说明解释程序的文字就是注释，**注释提高了代码的阅读性**；

注释是一个程序员必须要具有的良好编程习惯。将自己的思想通过注释先整理出来，再用代码去体现。

2.11.2 在 Golang 中注释有两种形式

1) 行注释

- 基本语法

// 注释内容

- 举例

```
//这是一个main函数，是程序入口
func main() {
    //演示转义字符的使用 \t
    fmt.Println("tom\tjack")

    // 如果希望一次性注释 ctrl + / 第一次表示注释，第二次表示取消注释

    // fmt.Println("hello\nworld")
    // fmt.Println("C:\\Users\\Administrator\\Desktop\\Go语言核心编程课程\\资料")
    // fmt.Println("tom说\"i love you\"")

    // \r 回车,从当前行的最前面开始输出，覆盖掉以前内容
    fmt.Println("天龙八部雪山飞狐\r张飞厉害")
}
```

2) 块注释(多行注释)

- 基本语法

/*

注释内容

*/

- 举例说明

```
/*
fmt.Println("hello\nworld")
fmt.Println("C:\\Users\\Administrator\\Desktop\\Go语言核心编程课程\\资料")
fmt.Println("tom说\"i love you\"")
*/
```

- 使用细节

1) 对于行注释和块注释，被注释的文字，不会被 Go 编译器执行。

2) 块注释里面不允许有块注释嵌套 [注意一下]

2.12 规范的代码风格

2.12.1 正确的注释和注释风格：

1) Go 官方推荐使用行注释来注释整个方法和语句。

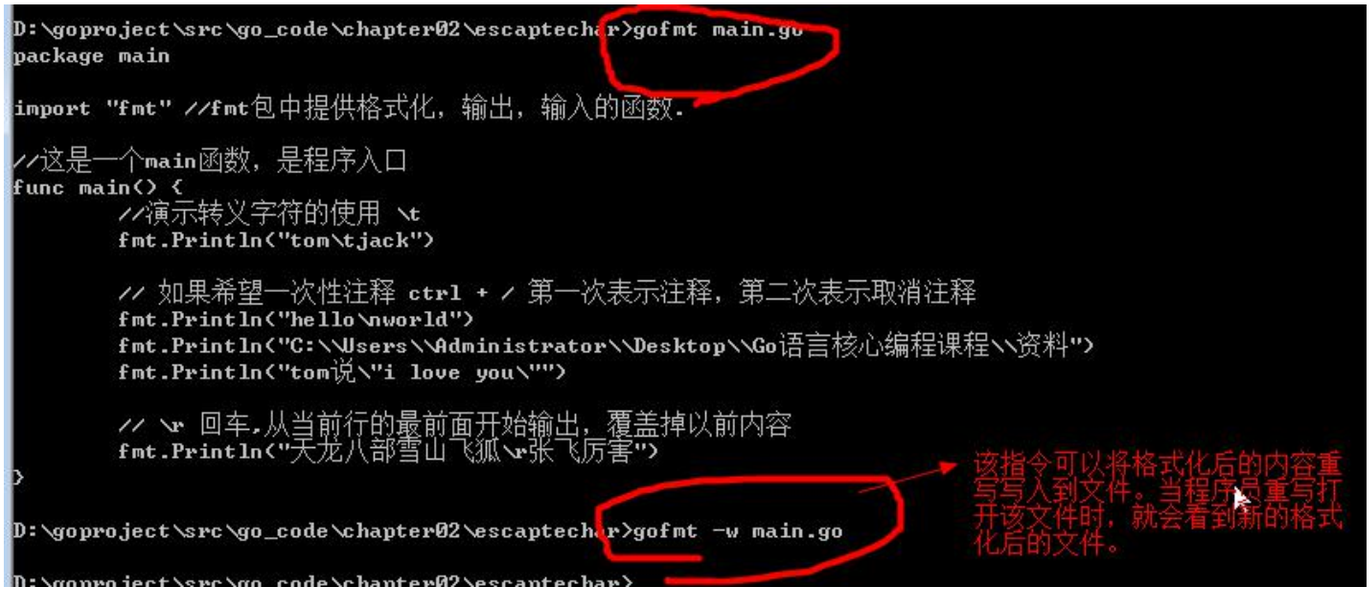
2) 带看 Go 源码

2.12.2 正确的缩进和空白

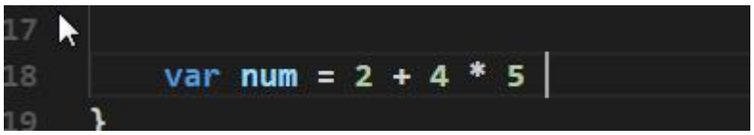
1) 使用一次 **tab 操作**，实现缩进,默认整体向右边移动，时候用 **shift+tab** 整体向左移

看老师的演示：

2) 或者使用 gofmt 来进行格式化 [演示]



3) 运算符两边习惯性各加一个空格。比如：2 + 4 * 5。



4) Go 语言的代码风格.

```
package main
import "fmt"
func main() {
    fmt.Println("hello,world!")
}
```

上面的写法是正确的.

```
package main
import "fmt"
func main()
{
    fmt.Println("hello,world!")
```

}

上面的写法不是正确，Go 语言不允许这样编写。 【Go 语言不允许这样写，是错误的！】

Go 设计者思想: **一个问题尽量只有一个解决方法**

5) 一行最长不超过 80 个字符，超过的请使用换行展示，尽量保持格式优雅

- 举例说明

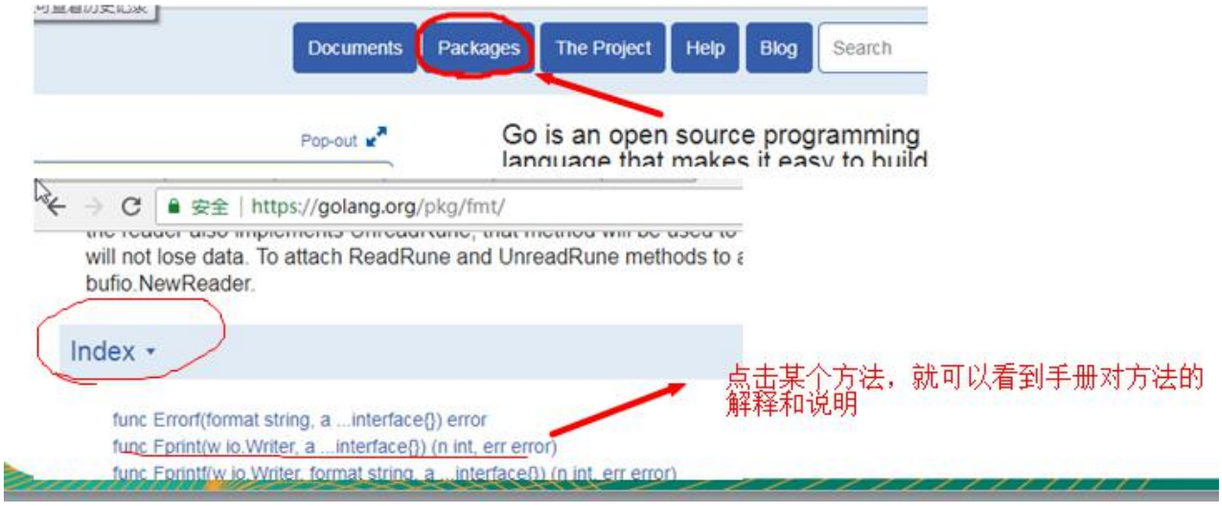
```
// \r 回车,从当前行的最前面开始输出，覆盖掉以前内容
fmt.Println("天龙八部雪山飞狐\r张飞厉害")

fmt.Println("helloworldhelloworldhelloworldhelloworl\n",
    "dhelloworldhelloworldhelloworldhelloworldhelloworldhellowor\n",
    "ldhelloworldhelloworldhelloworldhelloworldhelloworldhelloworl\n",
    "dhelloworldhelloworldhelloworldhelloworldhelloworldhelloworldhel\n",
    "loworldhelloworldhelloworldhelloworldhelloworldhelloworldhellowor\n",
    "ldhelloworldhelloworldhelloworldhelloworldhelloworldhelloworld")
```

2.13 Golang 官方编程指南

- 说明： Golang 官方网站 https://golang.org

- 点击上图的 tour -> 选择 简体中文就可以进入中文版的 Go 编程指南 。

- Golang 官方标准库 API 文档， https://golang.org/pkg 可以查看 Golang 所有包下的函数和使用



- 解释术语：API

  api ： application program interface :应用程序编程接口。

  就是我们 Go 的各个包的各个函数。

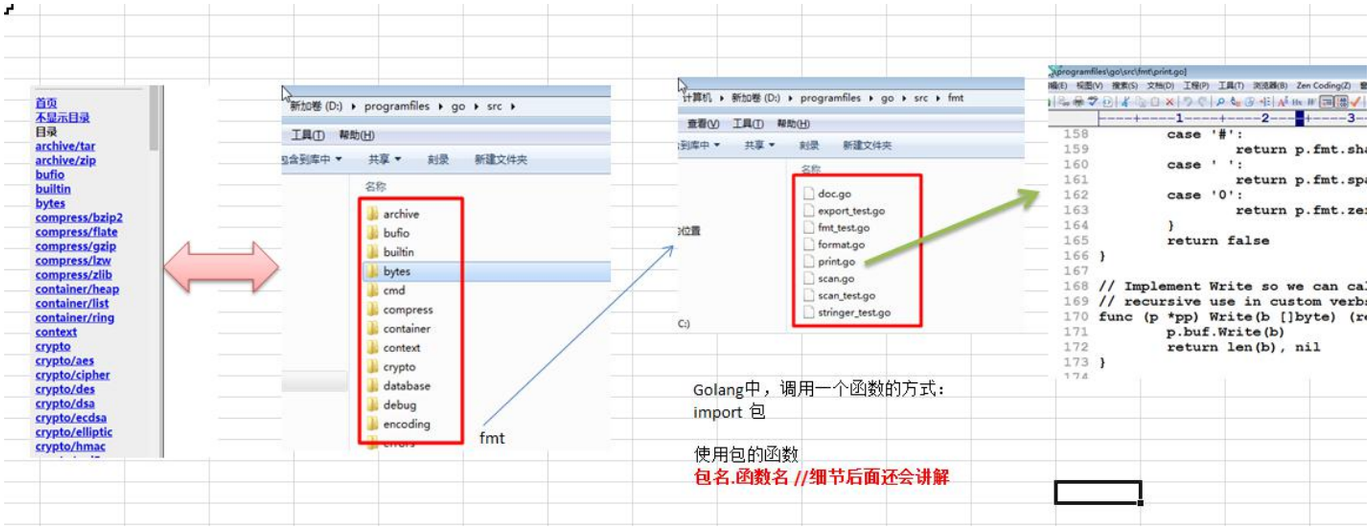
2.14 Golang 标准库 API 文档

1) API （Application Programming Interface,应用程序编程接口）是 Golang 提供的基本编程接口。

2) Go 语言提供了大量的标准库，因此 google 公司 也为这些标准库提供了相应的 API 文档，用于告诉开发者如何使用这些标准库，以及标准库包含的方法。

3) Golang 中文网 在线标准库文档： https://studygolang.com/pkgdoc

4) Golang 的包和源文件和函数的关系简图



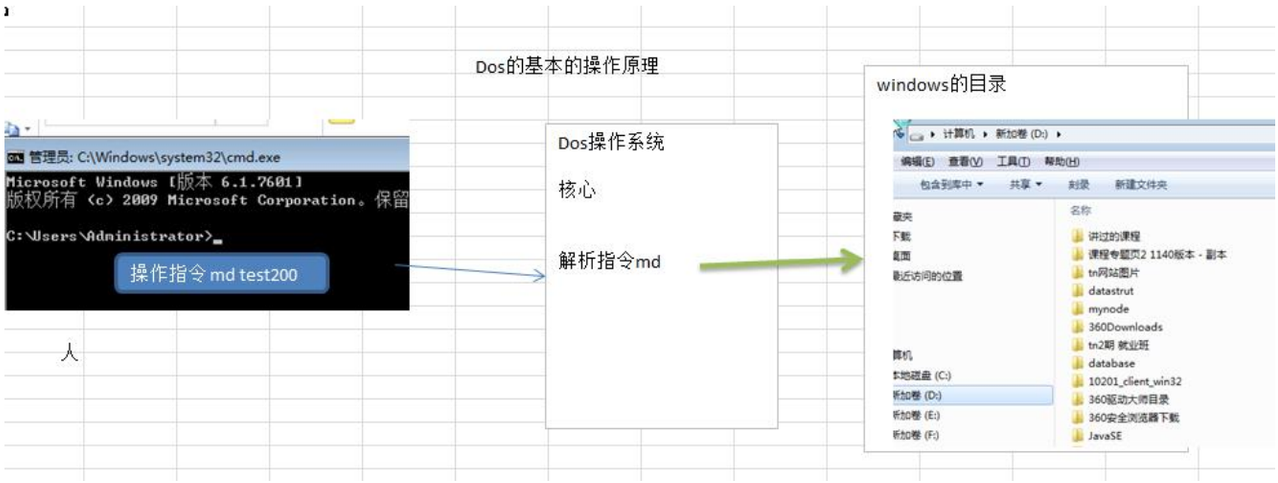
5) 有一个离线版的 Golang_Manual_By_AstaXie_20120522.chm

2.15 Dos 的常用指令(了解)

2.15.1 dos 的基本介绍

Dos： Disk Operating System 磁盘操作系统, 简单说一下 windows 的目录结构

2.15.2 dos 的基本操作原理



2.15.3 目录操作指令

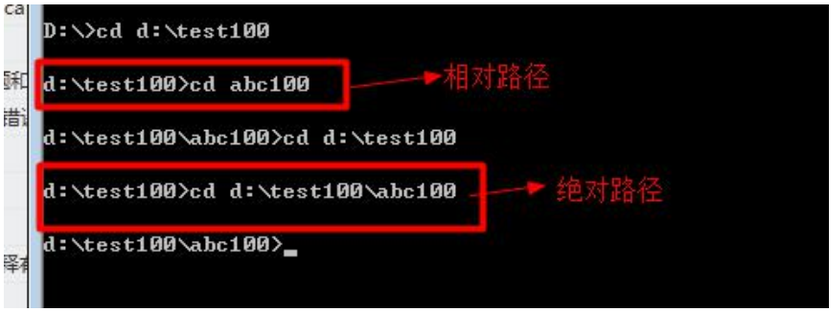
- 查看当前目录是什么

- 切换到其他盘下：盘符号 F 盘

  案例演示：

- 切换到当前盘的其他目录下 (使用**相对路径**和**绝对路径**演示)

  案例演示：



- 切换到上一级：

  案例演示：

- 切换到根目录：

  案例演示：

- 新建目录 md （make directory）

  新建一个目录：

  新建多个目录：

- 删除目录

  删除空目录

```
d:\test200>rd ok100
```

删除目录以及下面的子目录和文件，不带询问

```
d:\test200>rd /q/s ok200
```

删除目录以及下面的子目录和文件，带询问

```
d:\test200>rd /s ok300
ok300, 是否确认(Y/N)? y
```

2.15.4 文件的操作

- 新建或追加内容到文件

  案例演示：

```
d:\test200>echo hello > d:\test100\abc100\abc.txt
```

```
d:\test100\abc100>echo atguigu > abc2.txt
```

- 复制或移动文件

  复制

```
d:\test100\abc100>copy abc.txt d:\test200
已复制         1 个文件。

d:\test100\abc100>copy abc.txt d:\test200\ok.txt
已复制         1 个文件。
```

拷贝时，使用原来文件名

拷贝重写指定名

  移动

```
d:\test100\abc100>move abc.txt f:\
移动了         1 个文件。
```

- 删除文件

  删除指定文件

```
d:\test100\abc100>del abc2.txt

d:\test100\abc100>dir
```

  删除所有文件

```
d:\test100\abc100>del *.txt
```

2.15.5 其它指令

- 清屏

  cls [苍老师]

- 退出 dos

  exit

2.15.6 综合案例

切换到E盘下，新建一个目录 atguigu100（文件夹），切换到该目录，新建a b c三个目录,然后切换到a ，新建文件news.txt,然后复制文件到b下，最后都删除

完成该案例： 10min

2.16 课后练习题的评讲

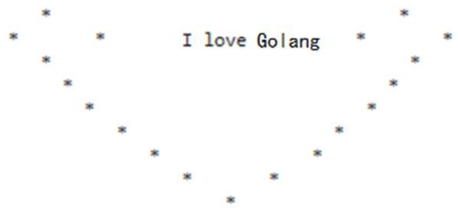
1) 独立编写 Hello,Golang!程序[评讲]

```go
package main
import "fmt"

func main() {
    fmt.Println("hello,Golang")
}
```

2) 将个人的基本信息（姓名、性别、籍贯、住址）打印到终端上输出。各条信息分别占一行

```go
package main
import "fmt"

func main() {
    fmt.Println("姓名\t性别\t籍贯\t住址\ntom\t男\t天津\t北京")
}
```

3) 在终端打印出如下图所示的效果



2.17 本章的知识回顾

- Go 语言的 SDK 是什么?

  SDK 就是软件开发工具包。我们做 Go 开发，首先需要先安装并配置好 sdk.

- Golang 环境变量配置及其作用。

  GOROOT: 指定 go sdk 安装目录。

  Path: 指令 sdk\bin 目录：go.exe　godoc.exe　gofmt.exe

  GOPATH: 就是 golang 工作目录：我们的所有项目的源码都这个目录下。

- Golang 程序的编写、编译、运行步骤是什么？能否一步执行?

  编写：就是写源码

  编译：go build 源码 =》 生成一个二进制的可执行文件

  运行：1. 对可执行文件运行 xx.exe　./可执行文件　2. go run 源码

- Golang 程序编写的规则。

  1) go 文件的后缀 .go

2) go 程序区分大小写

3) go 的语句后，不需要带分号

4) go 定义的变量，或者 import 包，必须使用，如果没有使用就会报错

5) go 中，不要把多条语句放在同一行。否则报错

6) go 中的大括号成对出现，而且风格

```
func main() {
    //语句
}
```

- 简述：在配置环境、编译、运行各个步骤中常见的错误

  对初学者而言，最容易错的地方**拼写错误**。比如文件名，路径错误。拼写错误

第 3 章 Golang 变量

3.1 为什么需要变量

3.1.1一个程序就是一个世界

3.1.2变量是程序的基本组成单位

不论是使用哪种高级程序语言编写程序,变量都是其程序的基本组成单位，比如一个示意图：

```go
func getVal(num1 int, num2 int) (int, int) {
    sum := num1 + num2
    sub := num2 - num2
    return sum, sub
}

func main() {

    sum, sub:= getVal(30, 30)
    fmt.Println("sum=", sum, "sub=", sub)
    sum2, _ := getVal(10, 30)//只取出第一个返回值
    fmt.Println("sum=", sum2)
}
```

比如上图的 sum,sub 都是变量。

3.2 变量的介绍

3.2.1变量的概念

变量相当于内存中一个数据存储空间的表示，你可以把变量看做是一个房间的门牌号，通过门牌号我们可以找到房间，同样的道理，通过变量名可以访问到变量(值)。

3.2.2变量的使用步骤

1) 声明变量(也叫:定义变量)
2) 非变量赋值
3) 使用变量

3.3 变量快速入门案例

看一个案例：

```go
package main
import "fmt"

func main() {
    //定义变量/声明变量
    var i int
    //给i 赋值
    i = 10
    //使用变量
    fmt.Println("i=", i)
}
```

输出：

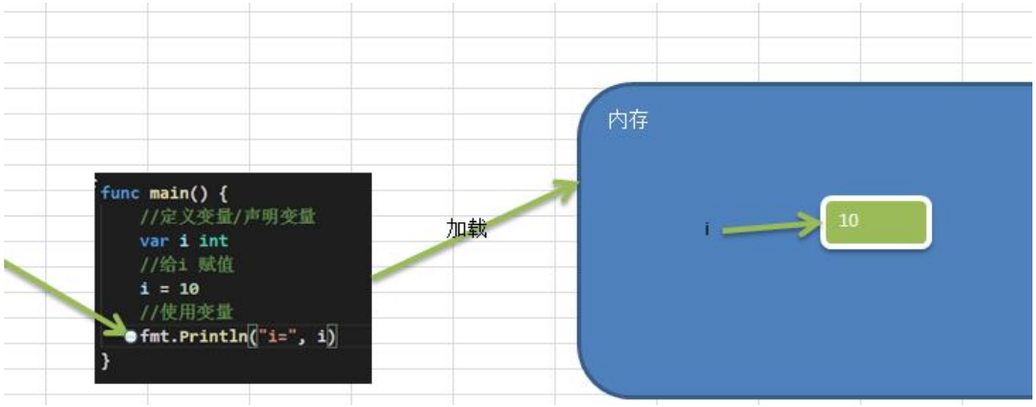
```
D:\goproject\src\go_code\chapter03\demo01>go run main.go
i= 10
```

3.4 变量使用注意事项

1) 变量表示内存中的一个存储区域

2) 该区域有自己的名称（变量名）和类型（数据类型）

示意图：



3) Golang 变量使用的三种方式

(1) 第一种：指定变量类型，**声明后若不赋值，使用默认值**

```
package main
D:\goproject\src\go_code\test.go

func main() {
    //golang的变量使用方式1
    //第一种：指定变量类型，声明后若不赋值，使用默认值
    // int 的默认值是0 ，其它数据类型的默认值后面马上介绍
    var i int
    fmt.Println("i=", i)
}
```

(2) 第二种：根据值自行判定变量类型(**类型推导**)

```
//第二种：根据值自行判定变量类型(类型推导)
var num  = 10.11
fmt.Println("num=", num)
```

(3) 第三种：省略 var, 注意 :=左侧的变量不应该是已经声明过的，否则会导致编译错误

```
//第三种：省略var，注意 :=左侧的变量不应该是已经声明过的，否则会导致编译错误
//下面的方式等价 var name string   name = "tom"
// := 的 :不能省略，否则错误
name := "tom"
fmt.Println("name=", name)
```

4) 多变量声明

在编程中，有时我们需要一次性声明多个变量，Golang 也提供这样的语法

举例说明:

```go
package main
import "fmt"

func main() {

    //该案例演示golang如何一次性声明多个变量
    // var n1, n2, n3 int
    // fmt.Println("n1=",n1, "n2=", n2, "n3=", n3)

    //一次性声明多个变量的方式2
    // var n1, name , n3 = 100, "tom", 888
    // fmt.Println("n1=",n1, "name=", name, "n3=", n3)

    //一次性声明多个变量的方式3，同样可以使用类型推导
    n1, name , n3 := 100, "tom~", 888
    fmt.Println("n1=",n1, "name=", name, "n3=", n3)

}
```

如何一次性声明多个全局变量【在 go 中函数外部定义变量就是全局变量】：

```go

//定义全局变量
var n1 = 100
var n2 = 200
var name = "jack"
//上面的声明方式，也可以改成一次性声明
var (
    n3 = 300
    n4 = 900
    name2 = "mary"
)
```

5) 该区域的数据值可以在同一类型范围内不断变化(重点)

```go
package main
import "fmt"

//变量使用的注意事项
func main() {

    //该区域的数据值可以在同一类型范围内不断变化
    var i int = 10
    i = 30
    i = 50
    fmt.Println("i=", i)
    i = 1.2 //int ,原因是不能改变数据类型

}
```

6) 变量在**同一个作用域**(在一个函数或者在代码块)内不能重名

```go
//i = 1.2 //int ,原因是不能改变数据类型

//变量在同一个作用域(在一个函数或者在代码块)内不能重名
var i int = 59
i := 99
```

7) 变量=**变量名**+**值**+**数据类型**，这一点请大家注意，变量的三要素

8) Golang 的变量如果没有赋初值，编译器会使用默认值, 比如 int 默认值 0 string 默认值为空串，小数默认为 0

3.5 变量的声明，初始化和赋值

声明变量

基本语法：var 变量名 数据类型

var a int 这就是声明了一个变量,变量名是 a

var num1 **float32** 这也声明了一个变量，表示一个单精度类型的小数,变量名是num1

初始化变量

在声明变量的时候，就给值。

var a int = 45 这就是初始化变量a

使用细节：如果声明时就直接赋值，可省略数据类型

var b = 400

给变量赋值

比如你先声明了变量: var num int // 默认0

然后，再给值 num = 780; 这就是给变量赋值.

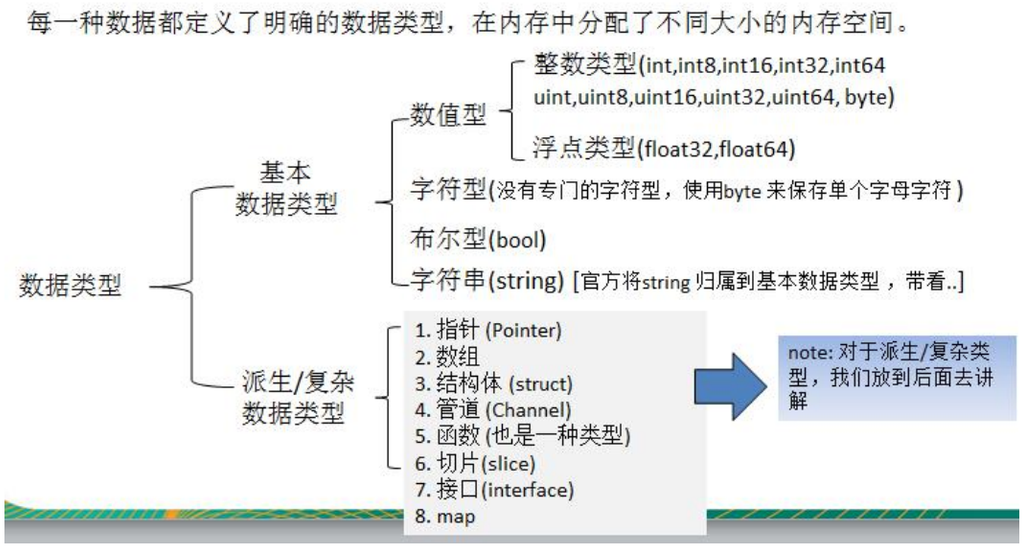
3.6 程序中 +号的使用

1) 当左右两边都是数值型时，则做加法运算

2) 当左右两边都是字符串，则做字符串拼接

```go
package main
import "fmt"

//演示golang中+的使用
func main() {

    var i = 1
    var j = 2
    var r = i + j //做加法运算
    fmt.Println("r=", r)

    var str1 = "hello "
    var str2 = "world"
    var res = str1 + str2 //做拼接操作
    fmt.Println("res=", res)

}
```

3.7 数据类型的基本介绍



3.8 整数类型

3.8.1基本介绍

简单的说，就是**用于存放整数值**的，比如 0, -1, 2345 等等。

3.8.2案例演示

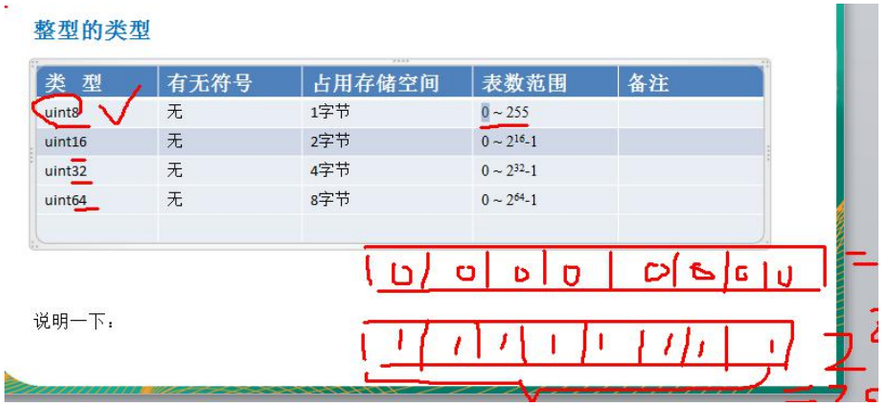
3.8.3整数的各个类型

整型的类型

类 型	有无符号	占用存储空间	表数范围	备注
int8	有	1字节	-128 ~ 127	
int16	有	2字节	$-2^{15} \sim 2^{15}-1$	
int32	有	4字节	$-2^{31} \sim 2^{31}-1$	
int64	有	8字节	$-2^{63} \sim 2^{63}-1$	

```go
//演示golang中整数类型使用
func main() {

    var i int = 1
    fmt.Println("i=", i)

    //测试一下int8的范围 -128~127,
    //其它的 int16, int32, int64,类推。。。
    var j int8 = 127
    fmt.Println("j=", j)

}
```

int 的无符号的类型：



```
//测试一下 uint8的范围(0-255),其它的 uint16, uint32, uint64类推即可
var k uint16 = 255
fmt.Println("k=", k)
```

int 的其它类型的说明:

整型的类型

类 型	有无符号	占用存储空间	表数范围	备注
int	有	32位系统4个字节 64位系统8个字节	$-2^{31} \sim 2^{31}-1$ $-2^{63} \sim 2^{63}-1$	
uint	无	32位系统4个字节 64位系统8个字节	$0 \sim 2^{32}-1$ $0 \sim 2^{64}-1$	
rune	有	与int32一样	$-2^{31} \sim 2^{31}-1$	等价 int32，表示一个 Unicode 码
byte	无	与 uint8 等价	0 ~ 255	当要存储字符时选用byte

```
//int , uint , rune , byte的使用
var a int = 8900
fmt.Println("a=", a)
var b uint = 1
var c byte = 255
fmt.Println("b=", b, "c=", c)
```

3.8.4整型的使用细节

1) Golang 各整数类型分：有符号和无符号，int uint 的大小和系统有关。

2) Golang 的整型默认声明为 int 型

```
//整型的使用细节
var n1 = 100 // ? n1 是什么类型
//这里我们给介绍一下如何查看某个变量的数据类型
//fmt.Printf() 可以用于做格式化输出。
fmt.Printf("n1 的 类型 %T \n", n1)
```

3) 如何在程序查看某个变量的字节大小和数据类型 （使用较多）

```
//如何在程序查看某个变量的占用字节大小和数据类型 （使用较多）
var n2 int64 = 10
//unsafe.Sizeof(n1) 是unsafe包的一个函数，可以返回n1变量占用的字节数
fmt.Printf("n2 的 类型 %T  n2占用的字节数是 %d ", n2, unsafe.Sizeof(n2))
```

4) Golang 程序中整型变量在使用时，遵守保小不保大的原则，即：在保证程序正确运行下，尽量使用占用空间小的数据类型。【如：年龄】

```
//Golang程序中整型变量在使用时，遵守保小不保大的原则，
//即：在保证程序正确运行下，尽量使用占用空间小的数据类型
var age byte = 90
```

5) bit: 计算机中的最小存储单位。byte:计算机中基本存储单元。[二进制再详细说] 1byte = 8 bit

3.9 小数类型/浮点型

3.9.1基本介绍

小数类型就是用于存放小数的，比如 1.2 0.23 -1.911

3.9.2案例演示

```
package main
import (
    "fmt"
)

//演示golang中小数类型使用
func main() {

    var price float32 = 89.12
    fmt.Println("price=", price)
}
```

3.9.3小数类型分类

类 型	占用存储空间	表数范围
单精度float32	4字节	-3.403E38 ~ 3.403E38
双精度float64	8字节	-1.798E308 ~ 1.798E308

对上图的说明:

1) 关于浮点数在机器中存放形式的简单说明,浮点数=符号位+指数位+尾数位

   说明：浮点数都是有符号的.

```
//演示golang中小数类型使用
func main() {

    var price float32 = 89.12
    fmt.Println("price=", price)
    var num1 float32 = -0.00089
    var num2 float64 = -7809656.09
    fmt.Println("num1=", num1, "num2=", num2)
}
```

2) 尾数部分可能丢失，造成精度损失。 -123.0000901

```
//尾数部分可能丢失，造成精度损失。 -123.0000901
var num3 float32 = -123.0000901
var num4 float64 = -123.0000901
fmt.Println("num3=", num3, "num4=", num4)
```

```
D:\goproject\src\go_code\chapter03\floatdemo08>go run main.go
price= 89.12
num1= -0.00089 num2= -7.80965609e+06
num3= -123.00009 num4= -123.0000901
```

   说明：float64 的精度比 float32 的要准确.

   说明：如果我们要保存一个精度高的数，则应该选用 float64

3) 浮点型的存储分为三部分：符号位+指数位+尾数位 在存储过程中，精度会有丢失

3.9.4浮点型使用细节

1) Golang 浮点类型有固定的范围和字段长度，不受具体 OS(操作系统)的影响。

2) Golang 的浮点型默认声明为 float64 类型。

```
//Golang 的浮点型默认声明为float64 类型
var num5 = 1.1
fmt.Printf("num5的数据类型是 %T \n", num5)
```

3) 浮点型常量有两种表示形式

十进制数形式：如：5.12　　.512　　(必须有小数点）

科学计数法形式:如：5.1234e2　= 5.12 * 10 的 2 次方　　5.12E-2　= 5.12/10 的 2 次方

```
//十进制数形式：如：5.12　　.512　　（必须有小数点）
num6 := 5.12
num7 := .123 //=> 0.123
fmt.Println("num6=", num6, "num7=", num7)

//科学计数法形式
num8 := 5.1234e2 // ? 5.1234 * 10的2次方
num9 := 5.1234E2 // ? 5.1234 * 10的2次方 shift+alt+向下的箭头
num10 := 5.1234E-2 // ? 5.1234 / 10的2次方 0.051234
```

4) 通常情况下，应该使用 float64 ，因为它比 float32 更精确。[开发中，**推荐使用 float64**]

3.10 字符类型

3.10.1 基本介绍

Golang 中没有专门的字符类型，如果要存储单个**字符**(字母)，一般使用 **byte 来**保存。

字符串就是一串固定长度的字符连接起来的字符序列。Go 的字符串是由单个字节连接起来的。也就是说对于传统的字符串是由字符组成的，而 **Go 的字符串不同**，它是由**字节**组成的。

3.10.2 案例演示

```go
//演示golang中字符类型使用
func main() {

    var c1 byte = 'a'
    var c2 byte = '0' //字符的0

    //当我们直接输出byte值，就是输出了的对应的字符的码值
    // 'a' ==>
    fmt.Println("c1=", c1)
    fmt.Println("c2=", c2)
    //如果我们希望输出对应字符，需要使用格式化输出
    fmt.Printf("c1=%c c2=%c\n", c1, c2)

    //var c3 byte = '北' //overflow溢出
    var c3 int = '北' //overflow溢出
    fmt.Printf("c3=%c c3对应码值=%d", c3, c3)

}
```

对上面代码说明

1) 如果我们保存的字符在 ASCII 表的,比如[0-1, a-z,A-Z..]直接可以保存到 byte

2) 如果我们保存的字符对应码值大于 255,这时我们可以考虑使用 int 类型保存

3) 如果我们需要安装字符的方式输出，这时我们需要格式化输出，即 fmt.Printf(“%c”, c1)..

3.10.3 字符类型使用细节

1) 字符常量是用单引号(')括起来的单个字符。例如：var c1 byte = 'a' var c2 int = '中' var c3 byte = '9'

2) Go 中允许使用转义字符 '\’ 来将其后的字符转变为特殊字符型常量。例如：var c3 char = ‘\n’ // '\n'表示换行符

3) Go 语言的字符使用 UTF-8 编码，如果想查询字符对应的 utf8 码值 http://www.mytju.com/classcode/tools/encode_utf8.asp

英文字母-1 个字节 汉字-3 个字节

4) 在 Go 中，字符的本质是一个整数，直接输出时，是该字符对应的 UTF-8 编码的码值。

5) 可以直接给某个变量赋一个数字，然后按格式化输出时%c，会输出该数字对应的 unicode 字符

```
//可以直接给某个变量赋一个数字，然后按格式化输出时%c，会输出该数字对应的unicode 字符
var c4 int = 22269 // 22269 -> '国' 120->'x'
fmt.Printf("c4=%c\n", c4)
```

6) **字符类型是可以进行运算的**，相当于一个整数，因为它都对应有 Unicode 码.

```
//字符类型是可以进行运算的，相当于一个整数,运输时是按照码值运行
var n1 = 10 + 'a' //  10 + 97 = 107
fmt.Println("n1=", n1)
```

3.10.4 字符类型本质探讨

1) 字符型 存储到 计算机中，需要将字符对应的码值（整数）找出来

   存储：字符--->对应码值---->二进制-->存储

   读取：二进制----> 码值 ----> 字符 --> 读取

2) 字符和码值的对应关系是通过字符编码表决定的(是规定好)

3) Go 语言的编码都统一成了 utf-8。非常的方便，很统一，再也没有编码乱码的困扰了

3.11 布尔类型

3.11.1 基本介绍

1) 布尔类型也叫 bool 类型，bool 类型数据只允许取值 true 和 false

2) bool 类型占 1 个字节。

3) bool 类型适于**逻辑运算**，一般用于程序流程控制[注：这个后面会详细介绍]：

- if 条件控制语句；
- for 循环控制语句

3.11.2 案例演示

```go
//演示golang中bool类型使用
func main() {
    var b = false
    fmt.Println("b=", b)
    //注意事项
    //1. bool类型占用存储空间是1个字节
    fmt.Println("b 的占用空间 =", unsafe.Sizeof(b) )
    //2. bool类型只能取true或者false

}
```

3.12 string 类型

3.12.1 基本介绍

字符串就是一串固定长度的字符连接起来的字符序列。Go 的字符串是由单个字节连接起来的。Go 语言的字符串的字节使用 UTF-8 编码标识 Unicode 文本

3.12.2 案例演示

```go
//演示golang中string类型使用
func main() {
    //string的基本使用
    var address string = "北京长城 110 hello world!"
    fmt.Println(address)
}
```

3.12.3 string 使用注意事项和细节

1) Go 语言的字符串的字节使用 UTF-8 编码标识 Unicode 文本，这样 Golang 统一使用 UTF-8 编码,中文乱码问题不会再困扰程序员。

2) 字符串一旦赋值了，字符串就不能修改了：在 Go 中字符串是不可变的。

```go
//字符串一旦赋值了，字符串就不能修改了：在Go中字符串是不可变的
var str = "hello"
str[0] = 'a' //这里就不能去修改str的内容，即go中的字符串是不可变的。
```

3) 字符串的两种表示形式

(1) 双引号，会识别转义字符

(2) 反引号，以字符串的原生形式输出，包括换行和特殊字符，可以实现防止攻击、输出源代码等效果

【案例演示】

```
//输出源代码等效果 【案例演示】
str2 := "abc\nabc"
fmt.Println(str2)

//使用的反引号 ``
str3 := `
package main
import (
    "fmt"
    "unsafe"
)

//演示golang中bool类型使用
func main() {
    var b = false
    fmt.Println("b=", b)
    //注意事项
    //1. bool类型占用存储空间是1个字节
    fmt.Println("b 的占用空间 =", unsafe.Sizeof(b) )
    //2. bool类型只能取true或者false

}
`
fmt.Println(str3)
```

4) 字符串拼接方式

```
//字符串拼接方式
var str = "hello " + "world"
str += " haha!"
```

5) 当一行字符串太长时，需要使用到多行字符串，可以如下处理

```
fmt.Println(str)
//当一个拼接的操作很长时，怎么办，可以分行写,但是注意，需要将+保留在上一行.
str4 := "hello " + "world" + "hello " + "world" + "hello " +
"world" + "hello " + "world" + "hello " + "world" +
"hello " + "world"
fmt.Println(str4)
```

3.13 基本数据类型的默认值

3.13.1 基本介绍

在 go 中，数据类型都有一个默认值，当程序员没有赋值时，就会保留默认值，在 go 中，默认值又叫零值。

3.13.2 基本数据类型的默认值如下

数据类型	默认值
整型	0
浮点型	0
字符串	""
布尔类型	false

案例:

```
var a int // 0
var b float32 // 0
var c float64 // 0
var isMarried bool // false
var name string // ""
//这里的%v 表示按照变量的值输出
fmt.Printf("a=%d,b=%v,c=%v,isMarried=%v name=%v",a,b,c,isMarried, name)
```

3.14 基本数据类型的相互转换

3.14.1 基本介绍

Golang 和 java / c 不同，Go 在不同类型的变量之间赋值时**需要显式转换**。也就是说 Golang 中数据类型**不能自动转换**。

3.14.2 基本语法

表达式 T(v) 将值 v 转换为类型 T

T: 就是数据类型，比如 int32，int64，float32 等等

v: 就是需要转换的变量

3.14.3 案例演示

```go
package main
import (
    "fmt"
)

//演示golang中基本数据类型的转换
func main() {

    var i int32 = 100
    //希望将 i => float
    var n1 float32 = float32(i)
    var n2 int8 = int8(i)
    var n3 int64 = int64(i) //低精度->高精度

    fmt.Printf("i=%v n1=%v n2=%v n3=%v", i ,n1, n2, n3)

}
```

3.14.4 基本数据类型相互转换的注意事项

1) Go 中，数据类型的转换可以是从 表示范围小-->表示范围大，也可以 范围大--->范围小

2) 被转换的是变量存储的数据(即值)，变量本身的数据类型并没有变化！

```go
func main() {

    var i int32 = 100
    //希望将 i => float
    var n1 float32 = float32(i)
    var n2 int8 = int8(i)
    var n3 int64 = int64(i) //低精度->高精度

    fmt.Printf("i=%v n1=%v n2=%v n3=%v \n", i ,n1, n2, n3)


    //被转换的是变量存储的数据(即值)，变量本身的数据类型并没有变化
    fmt.Printf("i type is %T\n", i) // int32
```

3) 在转换中，比如将 int64 转成 int8 【-128---127】，编译时不会报错，只是转换的结果是**按溢出处理**，和我们希望的结果不一样。 因此在转换时，需要考虑范围.

```
//在转换中，比如将 int64  转成 int8 【-128---127】 ，编译时不会报错，
//只是转换的结果是按溢出处理，和我们希望的结果不一样
var num1 int64 = 999999
var num2 int8 = int8(num1) //
fmt.Println("num2=", num2)  ——→ num2=63
```

3.14.5 课堂练习

- 练习 1

```
func main() {

    //课堂练习 ， tab键
    var n1 int32 = 12
    var n2 int64
    var n3 int8

    n2 = n1 + 20  //int32 ---> int64 错误
    n3 = n1 + 20  //int32 ---> int8 错误


}
```

如何修改上面的代码，就可以正确.

```
func main() {

    //课堂练习 ， tab键
    var n1 int32 = 12
    var n2 int64
    var n3 int8

    n2 = int64(n1) + 20 //int32 ---> int64 错误
    n3 = int8(n1) + 20 //int32 ---> int8 错误
    fmt.Println("n2=", n2, "n3=", n3)

}
```

隐藏空白

- 练习 2

```
var n1 int32 = 12
var n3 int8
var n4 int8
n4 = int8(n1) + 127  //【编译通过，但是 结果 不是127+12 ，按溢出处理】
n3 = int8(n1) + 128 //【编译不过】
fmt.Println(n4)
```

3.15 基本数据类型和 string 的转换

3.15.1 基本介绍

在程序开发中，我们经常将基本数据类型转成 string,或者将 string 转成基本数据类型。

3.15.2 基本类型转 string 类型

- 方式 1：fmt.Sprintf("%参数", 表达式) **【个人习惯这个，灵活】**

  函数的介绍：

  func Sprintf

  ```
  func Sprintf(format string, a ...interface{}) string
  ```

  Sprintf根据format参数生成格式化的字符串并返回该字符串。

  参数需要和表达式的数据类型相匹配

  fmt.Sprintf().. 会返回转换后的字符串

- 案例演示

```go
//演示golang中基本数据练习转成string使用
func main() {

    var num1 int = 99
    var num2 float64 = 23.456
    var b bool = true
    var myChar byte = 'h'
    var str string //空的str

    //使用第一种方式来转换 fmt.Sprintf方法

    str = fmt.Sprintf("%d", num1)
    fmt.Printf("str type %T str=%q\n", str, str)

    str = fmt.Sprintf("%f", num2)
    fmt.Printf("str type %T str=%q\n", str, str)

    str = fmt.Sprintf("%t", b)
    fmt.Printf("str type %T str=%q\n", str, str)

    str = fmt.Sprintf("%c", myChar)
    fmt.Printf("str type %T str=%q\n", str, str)

}
```

- 方式 2：使用 strconv 包的函数

```
func FormatBool(b bool) string
func FormatFloat(f float64, fmt byte, prec, bitSize int) string
func FormatInt(i int64, base int) string
func FormatUint(i uint64, base int) string
```

- 案例说明

```go
    //第二种方式 strconv 函数
    var num3 int = 99
    var num4 float64 = 23.456
    var b2 bool = true

    str = strconv.FormatInt(int64(num3), 10)
    fmt.Printf("str type %T str=%q\n", str, str)

    // strconv.FormatFloat(num4, 'f', 10, 64)
    // 说明： 'f' 格式 10：表示小数位保留10位 64 :表示这个小数是float64
    str = strconv.FormatFloat(num4, 'f', 10, 64)
    fmt.Printf("str type %T str=%q\n", str, str)

    str = strconv.FormatBool(b2)
    fmt.Printf("str type %T str=%q\n", str, str)
```

```
47      //strconv包中有一个函数Itoa
48      var num5 int64 = 4567
49      str = strconv.Itoa(int(num5))
50      fmt.Printf("str type %T str=%q\n", str, str)
```

3.15.3 string 类型转基本数据类型

- 使用时 strconv 包的函数

func ParseBool(str string) (value bool, err error)
func ParseFloat(s string, bitSize int) (f float64, err error)
func ParseInt(s string, base int, bitSize int) (i int64, err error)
func ParseUint(s string, b int, bitSize int) (n uint64, err error)

- 案例演示

```
func main() {

    var str string = "true"
    var b bool
    // b, _ = strconv.ParseBool(str)
    // 说明
    // 1. strconv.ParseBool(str) 函数会返回两个值 (value bool, err error)
    // 2. 因为我只想获取到 value bool ,不想获取 err 所以我使用_忽略
    b , _ = strconv.ParseBool(str)
    fmt.Printf("b type %T  b=%v\n", b, b)

    var str2 string = "1234590"
    var n1 int64
    var n2 int
    n1, _ = strconv.ParseInt(str2, 10, 64)
    n2 = int(n1)
    fmt.Printf("n1 type %T  n1=%v\n", n1, n1)
    fmt.Printf("n2 type %T n2=%v\n", n2, n2)

    var str3 string = "123.456"
    var f1 float64
    f1, _ = strconv.ParseFloat(str3, 64)
    fmt.Printf("f1 type %T f1=%v\n", f1, f1)

}
```

- 说明一下

note：因为返回的是 int64 或者 float64，如希望要得到int32,float32等如下处理：

```
//如果希望将str->int32的可以这样处理
var num5 int32
num5 = int32(num)
```

int16 ...

3.15.4 string 转基本数据类型的注意事项

在将 String 类型转成 基本数据类型时，**要确保 String 类型能够转成有效的数据**，比如 我们可以把 "123"，转成一个整数，但是不能把 "hello" 转成一个整数，如果这样做，Golang 直接将其转成 0 ，其它类型也是一样的道理. float => 0 bool => false

案例说明：

```
//注意:
var str4 string = "hello"
var n3 int64 = 11
n3, _ = strconv.ParseInt(str4, 10, 64)
fmt.Printf("n3 type %T n3=%v\n", n3, n3)
```

如果没有转成功，n3 = 0 //默认值

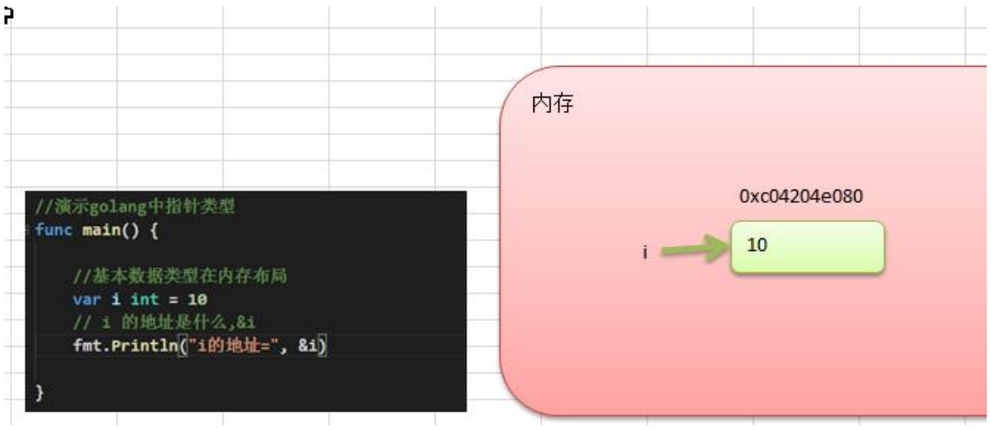
3.16 指针

3.16.1 基本介绍

1) 基本数据类型，变量存的就是值，也叫值类型

2) 获取变量的地址，用&，比如： var num int, 获取 num 的地址：&num

分析一下基本数据类型在内存的布局.

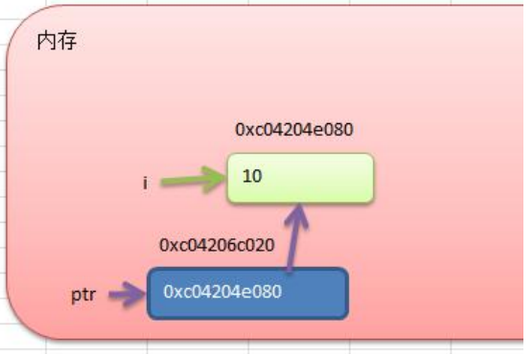


3) 指针类型，指针变量存的是一个地址，这个地址指向的空间存的才是值

比如：var ptr *int = &num

举例说明：指针在内存的布局.

```go
//演示golang中指针类型
func main() {

    //基本数据类型在内存布局
    var i int = 10
    // i 的地址是什么,&i
    fmt.Println("i的地址=", &i)

    //下面的 var ptr *int = &i
    //1. ptr 是一个指针变量
    //2. ptr 的类型 *int
    //3. ptr 本身的值&i
    var ptr *int = &i
    fmt.Printf("ptr=%v\n", ptr)
}
```



4) 获取指针类型所指向的值，使用：*，比如：var ptr *int，使用*ptr 获取 ptr 指向的值

```go
//演示golang中指针类型
func main() {

    //基本数据类型在内存布局
    var i int = 10
    // i 的地址是什么,&i
    fmt.Println("i的地址=", &i)

    //下面的 var ptr *int = &i
    //1. ptr 是一个指针变量
    //2. ptr 的类型 *int
    //3. ptr 本身的值&i
    var ptr *int = &i
    fmt.Printf("ptr=%v\n", ptr)
    fmt.Printf("ptr 的地址=%v", &ptr)
    fmt.Printf("ptr 指向的值=%v", *ptr)

}
```

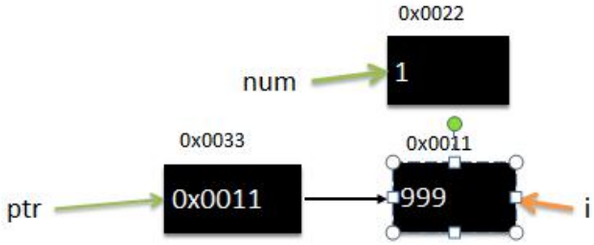
5) 一个案例再说明

举例说明

var num int=1

var i = 999

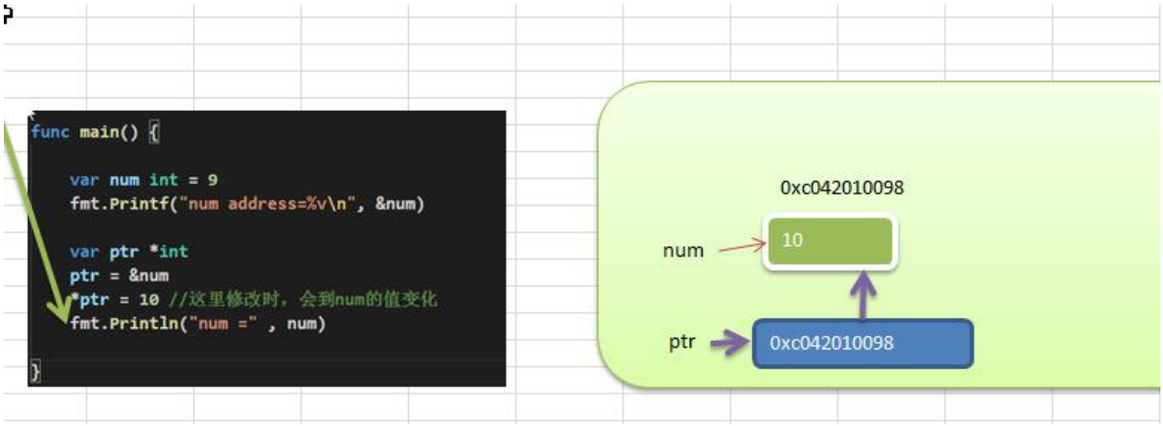
var ptr *int = &i



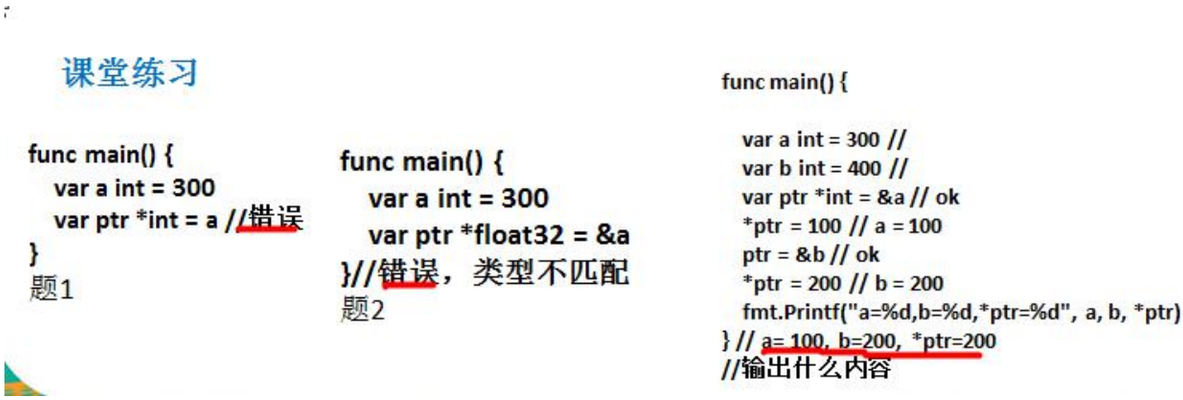
3.16.2 案例演示

1) 写一个程序，获取一个 int 变量 num 的地址，并显示到终端

2) 将 num 的地址赋给指针 ptr，并通过 ptr 去修改 num 的值.



3.16.3 指针的课堂练习



3.16.4 指针的使用细节

1) 值类型，都有**对应的指针类型**， 形式为 ***数据类型**，比如 int 的对应的指针就是 *int, float32 对应的指针类型就是 *float32, 依次类推。

2) 值类型包括：基本数据类型 **int 系列, float 系列, bool, string 、数组**和**结构体** struct

3.17 值类型和引用类型

3.17.1 值类型和引用类型的说明

1) 值类型：基本数据类型 int 系列, float 系列, bool, string 、数组和结构体 struct

2) 引用类型：指针、slice 切片、map、管道 chan、interface 等都是引用类型

3.17.2 值类型和引用类型的使用特点

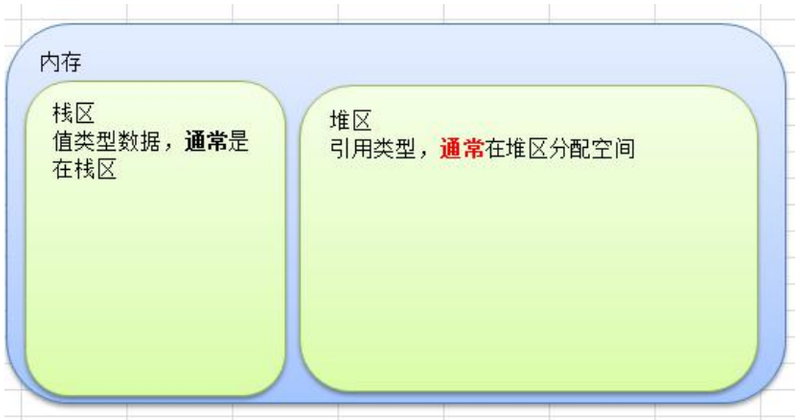
1) 值类型：变量直接存储值，内存通常在栈中分配

示意图：

2) 引用类型：变量存储的是一个地址，这个地址对应的空间才真正存储数据(值)，内存通常在堆上分配，当没有任何变量引用这个地址时，该地址对应的数据空间就成为一个垃圾，由 GC 来回收

示意图：

3) 内存的栈区和堆区示意图



3.18 标识符的命名规范

3.18.1 标识符概念

1) Golang 对各种变量、方法、函数等命名时使用的字符序列称为标识符

2) 凡是自己可以起名字的地方都叫标识符

3.18.2 标识符的命名规则

1) 由 26 个英文字母大小写，0-9 ，_ 组成

2) 数字不可以开头。var num int //ok　　var 3num int //error

3) Golang 中严格区分大小写。

var num int

var Num int

说明：在 golang 中，num 和 Num 是两个不同的变量

4) 标识符不能包含空格。

5) 下划线"_"本身在 Go 中是一个特殊的标识符，称为空标识符。可以代表任何其它的标识符，但是它对应的值会被忽略(比如：忽略某个返回值)。所以**仅能被作为占位符使用，不能作为标识符使用**

6) 不能以系统**保留关键字**作为标识符（一共有 25 个），比如 break，if 等等...

3.18.3 标识符的案例

hello　　// ok

hello12 //ok

1hello // error ,不能以数字开头

h-b　// error ,不能使用 -

x h　// error, 不能含有空格

h_4 // ok

_ab // ok

int // ok，我们要求大家不要这样使用

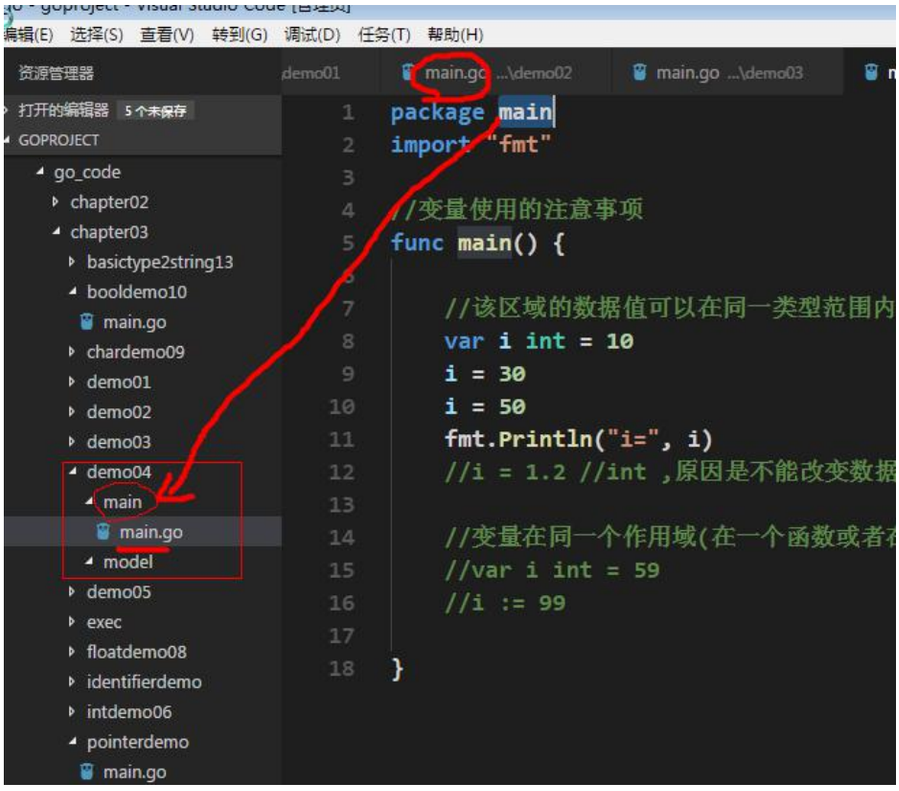
float32 // ok，我们要求大家不要这样使用

_ // error

Abc // ok

3.18.4 标识符命名注意事项

1) 包名：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，不要和标准库不要冲突 fmt



2) 变量名、函数名、常量名：采用驼峰法

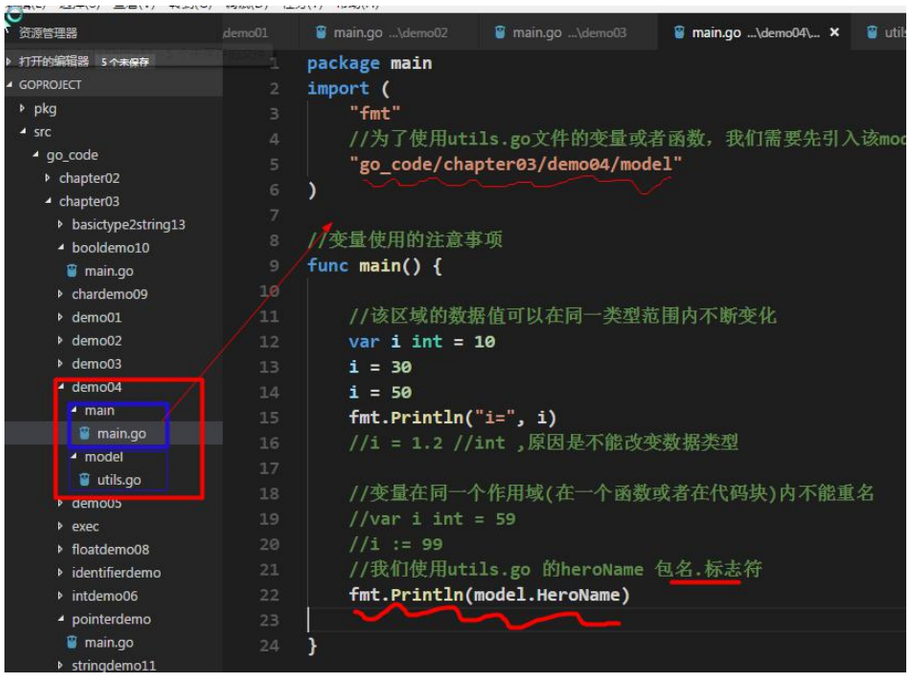
举例：

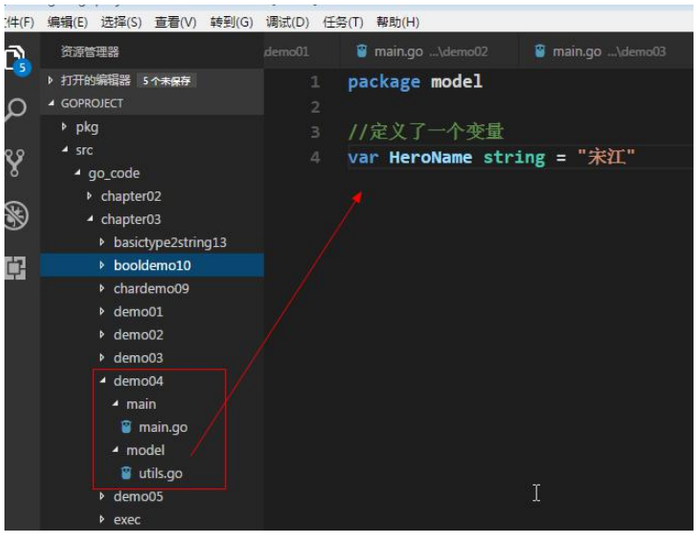
var stuName string = “tom” 形式: xxxYyyyyZzzz ...

var goodPrice float32 = 1234.5

3) 如果变量名、函数名、常量名首字母大写，则可以被其他的包访问；如果首字母小写，则只能在本包中使用（注：可以简单的理解成，**首字母大写是公开的**，首字母小写是私有的），在 golang **没有** public , private 等关键字。

案例演示：





3.19 系统保留关键字

保留关键字介绍

在Go中，为了简化代码编译过程中对代码的解析，其定义的**保留关键字**只有25个。详见如下

break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var

3.20 系统的预定义标识符

预定义标识符介绍

除了保留关键字外，Go还提供了36个预定的标识符，其包括基础数据类型和**系统内嵌函数**

append	bool	byte	cap	close	complex
complex64	complex128	uint16	copy	false	float32
float64	imag	int	int8	int16	uint32
int32	int64	iota	len	make	new
nil	panic	uint64	print	println	real
recover	string	true	uint	uint8	uintprt

第 4 章 运算符

4.1 运算符的基本介绍

运算符是一种特殊的符号，用以表示数据的运算、赋值和比较等

运算符是一种特殊的符号，用以表示数据的运算、赋值和比较等

1) 算术运算符
2) 赋值运算符
3) 比较运算符/关系运算符
4) 逻辑运算符
5) 位运算符
6) 其它运算符

4.2 算术运算符

算术运算符是对数值类型的变量进行运算的，比如：加减乘除。在 Go 程序中使用的非常多

4.2.1 算术运算符的一览表

运算符	运算	范例	结果
+	正号	+3	3
-	负号	-4	-4
+	加	5 + 5	10
-	减	6 - 4	2
*	乘	3 * 4	12
/	除	5 / 5	1
%	取模(取余)	7 % 5	2
++	自增	a=2 a++	a=3
--	自减	a=2 a--	a=1
+	字符串相加	"He" + "llo"	"Hello"

4.2.2 案例演示

- 案例演示算术运算符的使用。

  +, - , * , / , %, ++, -- ，重点讲解 /、%

  自增：++

  自减：--

- 演示 / 的使用的特点

```
//重点讲解 /、%
//说明，如果运算的数都是整数，那么除后，去掉小数部分，保留整数部分
fmt.Println(10 / 4)

var n1 float32 = 10 / 4 //
fmt.Println(n1)

//如果我们希望保留小数部分，则需要有浮点数参与运算
var n2 float32 = 10.0 / 4
fmt.Println(n2)
```

- 演示 % 的使用特点

  // 演示 % 的使用

  // 看一个公式 a % b = a - a / b * b

  fmt.Println("10%3=", 10 % 3) // =1

  fmt.Println("-10%3=", -10 % 3) // = -10 - (-10) / 3 * 3 = -10 - (-9) = -1

  fmt.Println("10%-3=", 10 % -3) // =1

  fmt.Println("-10%-3=", -10 % -3) // =-1

- ++ 和 --的使用

```
// ++ 和 --的使用
var i int = 10
i++ // 等价 i = i + 1
fmt.Println("i=", i) // 11
i-- // 等价 i = i - 1
fmt.Println("i=", i) // 10
```

4.2.3 算术运算符使用的注意事项

1) 对于除号 "/"，它的整数除和小数除是有区别的：整数之间做除法时，只保留整数部分而舍弃小数部分。 例如： x := 19/5 ,结果是 3

2) 当对一个数取模时，可以等价 a%b=a-a/b*b ， 这样我们可以看到 取模的一个本质运算。

3) Golang 的自增自减只能当做一个独立语言使用时，不能这样使用

```
package main
import (
    "fmt"
)
func main() {

    //在golang中，++ 和 -- 只能独立使用.
    var i int = 8
    var a int
    a = i++ //错误，i++只能独立使用
    a = i-- //错误，i--只能独立使用

    if i++ > 0 {
        fmt.Println("ok")
    }
}
```

4) Golang 的++ 和 -- 只能写在变量的后面，不能写在变量的前面，即：只有 a++ a-- 没有 ++a --a

```
var i int = 1
i++
++i // 错误，在golang没有 前++
i--
--i // 错误，在golang没有 前--
fmt.Println("i=", i)
```

5) Golang 的设计者去掉 c / java 中的 自增自减的容易混淆的写法，让 Golang 更加简洁，统一。(强制性的)

4.2.4 课堂练习 1

```
var i int = 1
i = i++
fmt.Println(i);
//问：结果是多少？为什么？
//上面的代码时错误，编译不通过,i = i++


var i int = 10
if i++ > 10 {
fmt.Println("ok")
}
//问：结果是多少？为什么？
//上面的代码时错误，编译不通过,i++ >10
```

4.2.5课堂练习 2

1) 假如还有 97 天放假，问：xx 个星期零 xx 天

2) 定义一个变量保存华氏温度，华氏温度转换摄氏温度的公式为：5/9*(华氏温度-100),请求出华氏温度对应的摄氏温度。

```go
package main
import (
    "fmt"
)
func main() {

    //假如还有97天放假，问：xx个星期零xx天
    var days int = 97
    var week int = days / 7
    var day int = days % 7
    fmt.Printf("%d个星期零%d天\n", week, day)


    //定义一个变量保存华氏温度，华氏温度转换摄氏温度的公式为：
    //5/9*(华氏温度-100),请求出华氏温度对应的摄氏温度
    var huashi float32 = 134.2
    var sheshi float32 = 5.0 / 9 * (huashi - 100)
    fmt.Printf("%v 对应的摄氏温度=%v \n", huashi, sheshi)
}
```

4.3 关系运算符(比较运算符)

4.3.1基本介绍

1) 关系运算符的结果都是 bool 型，也就是要么是 true，要么是 false

2) 关系表达式 经常用在 **if 结构**的条件中或**循环结构**的条件中

4.3.2 关系运算符一览图

运算符	运算	范例	结果
==	相等于	4==3	false
!=	不等于	4!=3	true
<	小于	4<3	false
>	大于	4>3	true
<=	小于等于	4<=3	false
>=	大于等于	4>=3	true

4.3.3 案例演示

```go
package main
import (
    "fmt"
)
func main() {
    //演示关系运算符的使用
    var n1 int = 9
    var n2 int = 8
    fmt.Println(n1 == n2) //false
    fmt.Println(n1 != n2) //true
    fmt.Println(n1 > n2) //true
    fmt.Println(n1 >= n2) //true
    fmt.Println(n1 < n2) //flase
    fmt.Println(n1 <= n2) //flase
    flag := n1 > n2
    fmt.Println("flag=", flag)
}
```

4.3.4 关系运算符的细节说明

细节说明

1) 关系运算符的结果都是 bool 型，也就是要么是 true，要么是 false。

2) 关系运算符组成的表达式，我们称为关系表达式： a > b

3) 比较运算符"=="不能误写成 "=" !!

4.4 逻辑运算符

4.4.1基本介绍

用于连接多个条件（一般来讲就是关系表达式），最终的结果也是一个 bool 值

4.4.2逻辑运算的说明

假定 A 值为 True，B 值为 False

运算符	描述	实例
&&	**逻辑与**运算符。如果两边的操作数**都是** True，则为 True，否则为 False。	(A && B) 为 False
||	**逻辑或**运算符。如果两边的操作数有一个 True，则为 True，否则为 False。	(A || B) 为 True
!	**逻辑非**运算符。如果条件为 True，则逻辑为 False，否则为 True。	!(A && B) 为 True

4.4.3案例演示

```go
//演示逻辑运算符的使用 &&
var age int = 40
if age > 30 && age < 50 {
    fmt.Println("ok1")
}

if age > 30 && age < 40 {
    fmt.Println("ok2")
}
```

```go
//演示逻辑运算符的使用 ||

if age > 30 || age < 50 {
    fmt.Println("ok3")
}

if age > 30 || age < 40 {
    fmt.Println("ok4")
}
```

```go
    //演示逻辑运算符的使用  !

    if age > 30 {
        fmt.Println("ok5")
    }

    if !(age > 30) {
        fmt.Println("ok6")
    }
```

4.4.4注意事项和细节说明

1) **&&也叫短路与：如果第一个条件为 false，则第二个条件不会判断，最终结果为 false**

2) ||也叫**短路或**：如果**第一个条件为 true**，则**第二个条件不会判断**，最终结果为 **true**

3) 案例演示

```go
//声明一个函数(测试)
func test() bool {
    fmt.Println("test....")
    return true
}

func main() {

    var i int = 10
    //短路与
    //说明 因为  i < 9 为 false ,因此后面的 test() 就不执行
    if i < 9 && test() {
        fmt.Println("ok...")
    }

    //说明 因为  i > 9 为 true ,因此后面的 test() 就不执行
    if i > 9 || test() {
        fmt.Println("hello...")
    }
```

4.5 赋值运算符

4.5.1基本的介绍

赋值运算符就是将某个运算后的值，赋给指定的变量。

4.5.2赋值运算符的分类

运算符	描述	实例
=	简单的赋值运算符，将一个表达式的值赋给一个左值	C = A + B 将 A + B 表达式结果赋值给 C
+=	相加后再赋值	C += A 等于 C = C + A
-=	相减后再赋值	C -= A 等于 C = C - A
*=	相乘后再赋值	C *= A 等于 C = C * A
/=	相除后再赋值	C /= A 等于 C = C / A
%=	求余后再赋值	C %= A 等于 C = C % A

运算符	描述	实例
<<=	左移后赋值	C <<= 2 等于 C = C << 2
>>=	右移后赋值	C >>= 2 等于 C = C >> 2
&=	按位与后赋值	C &= 2 等于 C = C & 2
^=	按位异或后赋值	C ^= 2 等于 C = C ^ 2
|=	按位或后赋值	C |= 2 等于 C = C | 2

说明： 这部分的赋值运算涉及到**二进制**相关知识，我们放在讲二进制的时候再回头讲解

4.5.3赋值运算的案例演示

案例演示赋值运算符的基本使用。

1) 赋值基本案例
2) 有两个变量，a 和 b，要求将其进行交换，最终打印结果
3) += 的使用案例
4) 案例

```go
func main() {
    //赋值运算符的使用演示
    // var i int
    // i = 10 //基本赋值

    //有两个变量，a和b，要求将其进行交换，最终打印结果
    // a = 9 , b = 2 ==> a = 2 b = 9
    a := 9
    b := 2
    fmt.Printf("交换前的情况是 a = %v , b=%v \n", a, b)
    //定义一个临时变量
    t := a
    a = b //
    b = t //
    fmt.Printf("交换后的情况是 a = %v , b=%v \n", a, b)

    //复合赋值的操作
    a += 17 // 等价 a = a + 17
    fmt.Println("a=", a)

}
```

4.5.4赋值运算符的特点

1) 运算顺序从右往左

```go
var c int
c = a + 3 // 赋值运算的执行顺序是从右向左
```

2) 赋值运算符的左边 只能是**变量**,右边 可以是**变量、表达式、常量值**

```go
//2)赋值运算符的左边 只能是变量,右边 可以是变量、表达式、常量值
// 表达式：任何有值都可以看做表达式
var d int
d = a //
d = 8 + 2 * 8 // =的右边是表达式
d = test() + 90 //  =的右边是表达式
d = 890 // 890常量
fmt.Println(d)
```

3) 复合赋值运算符等价于下面的效果

比如：a += 3 等价于 a = a + 3

4.5.5面试题

有两个变量，a 和 b，要求将其进行交换，但是不允许使用中间变量，最终打印结果

```go
package main
import (
    "fmt"
)
func main() {

    //有两个变量，a和b，要求将其进行交换，但是不允许使用中间变量，最终打印结果
    var a int = 10
    var b int = 20

    a = a + b //
    b = a - b // b = a + b - b ==> b = a
    a = a - b // a = a + b - a ==> a = b

    fmt.Printf("a=%v b=%v", a, b)
}
```

4.6 位运算符

运算符	描述
&	按位与运算符"&"是双目运算符。 其功能是参与运算的两数各对应的二进位相与。 运算规则是: 同时为1 , 结果为1, 否则为0
|	按位或运算符"|"是双目运算符。 其功能是参与运算的两数各对应的二进位相或 运算规则是: 有一个为1 , 结果为1, 否则为0
^	按位异或运算符"^"是双目运算符。 其功能是参与运算的两数各对应的二进位相异或. 运算规则是: 当二进位不同时, 结果为1, 否则为0
<<	左移运算符"<<"是双目运算符。 其功能把"<<"左边的运算数的各二进位全部左移若干位，高位丢弃，低位补0。左移n位就是乘以2的n次方。
>>	右移运算符">>"是双目运算符。 其功能是把">>"左边的运算数的各二进位全部右移若干位 右移n位就是除以2的n次方 **说明：因为位运算涉及到二进制相关知识，我们仍然放到讲二进制时，在详细讲解**

4.7 其它运算符说明

运算符	描述	实例
&	返回变量存储地址	&a; 将给出变量的实际地址。
*	指针变量	*a; 是一个指针变量

举例说明：

```
:\goproject\src\go_code\chapter04\exec\main.go
import (
    "fmt"
)
func main() {
    //演示一把 & 和 *的使用

    a := 100
    fmt.Println("a 的地址=", &a)

    var ptr *int = &a
    fmt.Println("ptr 指向的值是=", *ptr)
}
```

4.7.1课堂案例

案例 1：求两个数的最大值

```
func main() {

    //求两个数的最大值
    var n1 int = 10
    var n2 int = 40
    var max int
    if n1 > n2 {
        max = n1
    } else {
        max = n2
    }
    fmt.Println("max=", max)

}
```

案例 2：求三个数的最大值

```go
func main() {

    //求两个数的最大值
    var n1 int = 10
    var n2 int = 40
    var max int
    if n1 > n2 {
        max = n1
    } else {
        max = n2
    }
    fmt.Println("max=", max)

    //求出三个数的最大值思路：先求出两个数的最大值，
    //然后让这个最大值和第三数比较，在取出最大值
    var n3 = 45
    if n3 > max {
        max = n3
    }
    fmt.Println("三个数中最大值是=", max)

}
```

4.8 特别说明

举例说明，如果在 golang 中实现三元运算的效果。

```go
    var n int
    var i int = 10
    var j int = 12
    //传统的三元运算
    //n = i > j ? i : j
    if i > j {
        n = i
    } else {
        n = j
    }
    fmt.Println("n=", n) // 12
}
```

4.9 运算符的优先级

4.9.1运算符的优先级的一览表

分类	描述	关联性
后缀	() [] -> . ++ --	左到右
单目	+ - ! ~ (type) * & sizeof	右到左
乘法	* / %	左到右
加法	+ -	左到右
移位	<< >>	左到右
关系	< <= > >=	左到右
相等（关系）	== !=	左到右
按位AND	&	左到右
按位XOR	^	左到右
按位OR	|	左到右
逻辑AND	&&	左到右
逻辑OR	||	左到右
赋值运算符	= += -= *= /= %= >>= <<= &= ^= |=	右到左
逗号	,	左到右

4.9.2对上图的说明

1) 运算符有不同的优先级，所谓优先级就是表达式运算中的运算顺序。如右表，上一行运算符总

优先于下一行。

2) 只有单目运算符、赋值运算符是从右向左运算的。

3) 梳理了一个大概的优先级

1：括号，++, --

2: 单目运算

3：算术运算符

4：移位运算

5：关系运算符

6：位运算符

7：逻辑运算符

8：赋值运算符

9：逗号

4.10 键盘输入语句

4.10.1 介绍

在编程中，需要接收用户输入的数据，就可以使用键盘输入语句来获取。InputDemo.go

4.10.2 步骤 ：

1) 导入 fmt 包

2) 调用 fmt 包的 **fmt.Scanln()** 或者 **fmt.Scanf()**

func Scanln

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

Scanln类似Scan，但会在换行时才停止扫描。最后一个条目后必须有换行或者到达结束位置。

func Scanf

```
func Scanf(format string, a ...interface{}) (n int, err error)
```

Scanf从标准输入扫描文本，根据format 参数指定的格式将成功读取的空白分隔的值保存进成功传递给本函数的参数。返回成功扫描的条目个数和遇到的任何错误。

4.10.3 案例演示：

要求：可以从控制台接收用户信息，【姓名，年龄，薪水，是否通过考试 】。

1) 使用 fmt.Scanln() 获取

```
func main() {

    //要求：可以从控制台接收用户信息，【姓名，年龄，薪水，是否通过考试 】。
    //方式1 fmt.Scanln
    //1先声明需要的变量
    var name string
    var age byte
    var sal float32
    var isPass bool
    fmt.Println("请输入姓名 ")
    //当程序执行到 fmt.Scanln(&name),程序会停止在这里，等待用户输入，并回车
    fmt.Scanln(&name)
    fmt.Println("请输入年龄 ")
    fmt.Scanln(&age)
    fmt.Println("请输入薪水 ")
    fmt.Scanln(&sal)

    fmt.Println("请输入是否通过考试 ")
    fmt.Scanln(&isPass)

    fmt.Printf("名字是 %v \n 年龄是 %v \n 薪水是 %v \n 是否通过考试 %v \n", name, age,

}
```

2) 使用 fmt.Scanf() 获取

```
    //方式2:fmt.Scanf,可以按指定的格式输入
    fmt.Println("请输入你的姓名，年龄，薪水，是否通过考试， 使用空格隔开")
    fmt.Scanf("%s %d %f %t", &name, &age, &sal, &isPass)
    fmt.Printf("名字是 %v \n年龄是 %v \n 薪水是 %v \n 是否通过考试 %v \n", name, age, sal,
```

4.11 进制

对于整数，有四种表示方式：

1) 二进制：0,1 ，满 2 进 1。

   在 golang 中，不能直接使用二进制来表示一个整数，它沿用了 c 的特点。

2) 十进制：0-9 ，满 10 进 1。

3) 八进制：0-7 ，满 8 进 1. 以数字 0 开头表示。

4) 十六进制：0-9 及 A-F，满 16 进 1. 以 0x 或 0X 开头表示。

   此处的 A-F 不区分大小写。

```
6
7  func main() {
8
9      var i int = 5
10     //二进制输出
11     fmt.Printf("%b \n", i)
12
13     //八进制：0-7 ，满8进1. 以数字0开头表示
14     var j int = 011 // 011=> 9
15     fmt.Println("j=", j)
16
17     //0-9及A-F，满16进1. 以0x或0X开头表示
18     var k int = 0x11 // 0x11=> 16 + 1 = 17
19     fmt.Println("k=", k)
20 }
```

4.11.1 进制的图示

进制的图示

十进制	十六进制	八进制	二进制
0	0	0	0
1	1	1	1
2	2	2	10
3	3	3	11
4	4	4	100
5	5	5	101
6	6	6	110
7	7	7	111
8	8	10	1000

进制的图示

十进制	十六进制	八进制	二进制
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111
16	10	20	10000
17	11	21	10001

4.11.2 进制转换的介绍

- **第一组（其它进制转十进制）：**

1) 二进制转十进制
2) 八进制转十进制
3) 十六进制转十进制
4) 示意图

- **第二组：(十进制转其它进制)**

1) 十进制转二进制
2) 十进制转八进制
3) 十进制转十六进制
4) 示意图

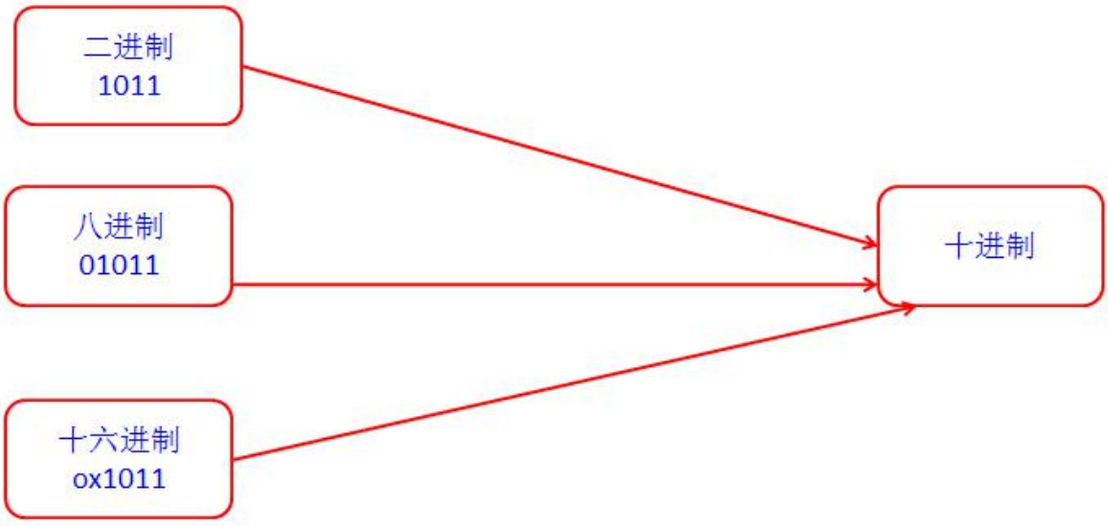
- **第三组 (二进制转其它进制)**

1) 二进制转八进制
2) 二进制转十六进制
3) 示意图

- **第四组 (其它进制转二进制)**

1) 八进制转二进制
2) 十六进制转二进制
3) 示意图

4.11.3 其它进制转十进制



4.11.4 二进制如何转十进制

134 = 4 * 1 + 3 * 10 + 1 * 10 * 10 = 4 + 30 + 100 = 134

规则：从最低位开始（右边的），将每个位上的数提取出来，乘以2的(位数-1)次方，然后求和。

案例：请将 二进制：1011 转成十进制的数

1011 = 1 * 1 + 1 * 2 + 0 * 2 * 2 + 1 * 2 * 2 * 2 = 1 + 2 + 0 + 8 = 11

4.11.5 八进制转换成十进制示例

规则：从最低位开始（右边的），将每个位上的数提取出来，乘以8的(位数-1)次方，然后求和。

案例：请将 0123 转成十进制的数

0123 = 3 * 1 + 2 * 8 + 1 * 8 * 8 = 3 + 16 + 64 = 83

4.11.6 16 进制转成 10 进制

规则：从最低位开始，将每个位上的数提取出来，乘以16的(位数-1)次方，然后求和。

案例：请将 0x34A 转成十进制的数

0x34A = 10 * 1 + 4 * 16 + 3 * 16 * 16 = 10 + 64 + 768 = 842

4.11.7 其它进制转 10 进制的课堂练习

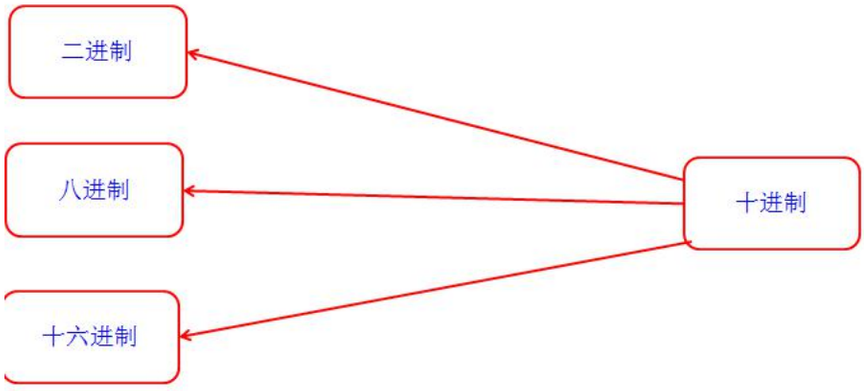
- 课堂练习：请将

  二进制： 110001100 转成 十进制

  八进制： 02456 转成十进制

  十六进制： 0xA45 转成十进制

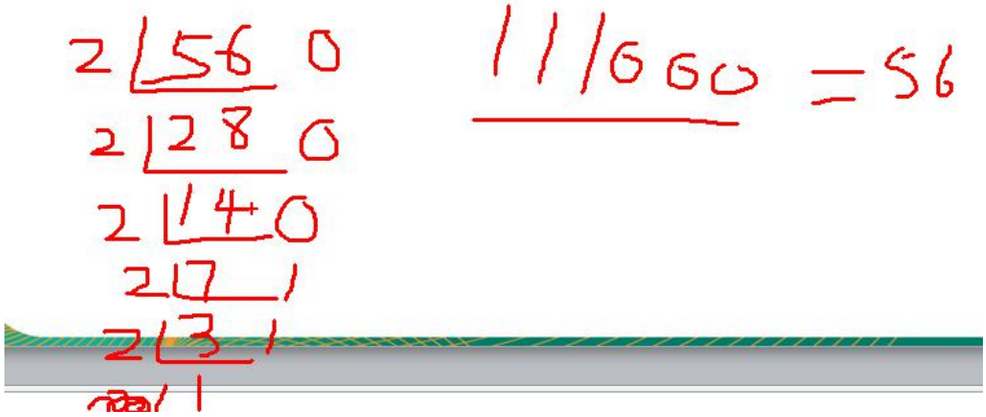
4.11.8 十进制如何转成其它进制



4.11.9 十进制如何转二进制

规则：将该数不断除以2，直到商为0为止，然后将每步得到的余数倒过来，就是对应的二进制。

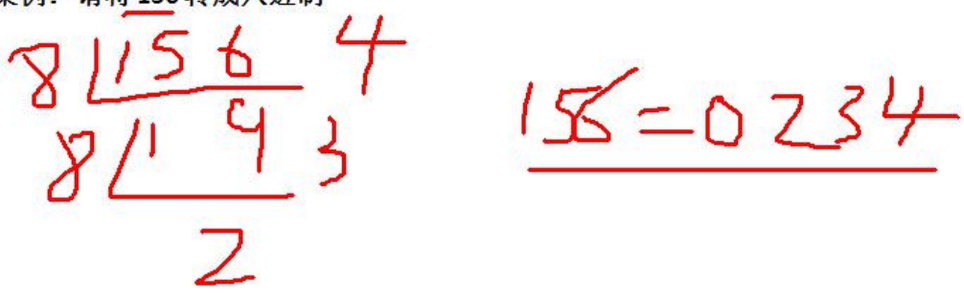
案例：请将 56 转成二进制



4.11.10 十进制转成八进制

规则：将该数不断除以8，直到商为0为止，然后将每步得到的余数倒过来，就是对应的八进制。

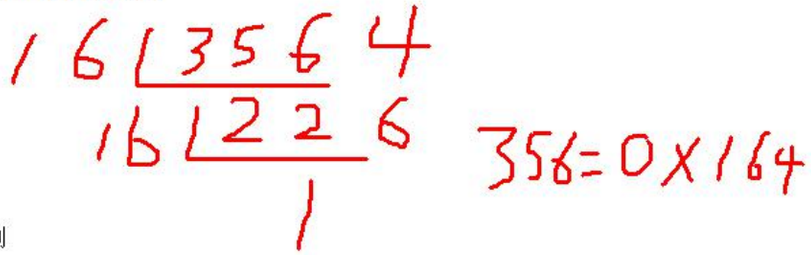
案例：请将 156 转成八进制



4.11.11 十进制转十六进制

规则：将该数不断除以16，直到商为0为止，然后将每步得到的余数倒过来，就是对应的十六进制。

案例：请将 356 转成十六进制



课堂练习：请将
123 转成 二进制
678 转成八进制
8912 转成十六进制

4.11.12 课堂练习

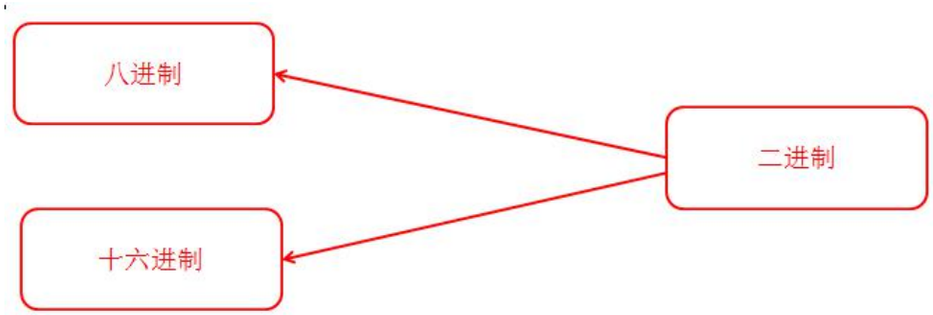
课堂练习：请将

123 转成 二进制

678 转成八进制

8912 转成十六进制

4.11.13 二进制转换成八进制、十六进制



4.11.14 二进制转换成八进制

规则：将二进制数每三位一组(从低位开始组合)，转成对应的八进制数即可。
案例：请将 二进制：11010101 转成八进制
11010101 = 0325

4.11.15 二进制转成十六进制

规则：将二进制数每四位一组(从低位开始组合)，转成对应的十六进制数即可。
案例：请将 二进制：11010101 转成十六进制
11010101 = 0xD5

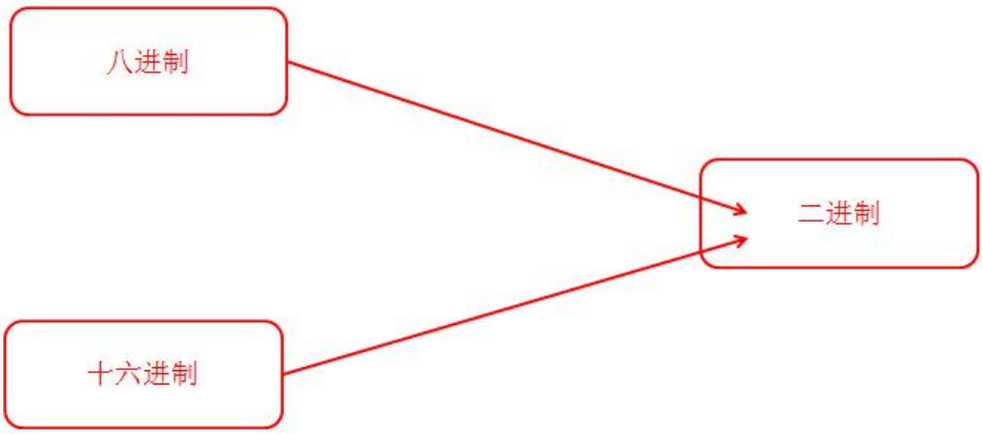
- 课堂练习

  课堂练习：请将

  二进制：11100101 转成 八进制

  二进制：1110010110 转成 十六进制

4.11.16 八进制、十六进制转成二进制



4.11.17 八进制转换成二进制

规则：将八进制数每1位，转成对应的一个3位的二进制数即可。
案例：请将 0237 转成二进制
0237 = 10011111

4.11.18 十六进制转成二进制

规则：将十六进制数每1位，转成对应的一个4位的二进制数即可。
案例：请将 0x237 转成二进制
0x237 = 1000110111

4.12 位运算

4.12.1 位运算的思考题

1) 请看下面的代码段，回答 a,b,c,d 结果是多少?

```
func main() {

    var a int = 1 >> 2
    var b int = -1 >> 2
```

```
    var c int = 1 << 2
    var d int = -1 << 2
    //a,b,c,d 结果是多少
    fmt.Println("a=", a)
    fmt.Println("b=", b)
    fmt.Println("c=", c)
    fmt.Println("d=", d)

}
```

2) 请回答在 Golang 中，下面的表达式运算的结果是:

```
func main() {

    fmt.Println(2&3)
    fmt.Println(2|3)
    fmt.Println(13&7)
    fmt.Println(5|4) //?
    fmt.Println(-3^3) //?

}
```

4.12.2 二进制在运算中的说明

二进制是逢 2 进位的进位制，0、1 是基本算符。

现代的电子计算机技术全部采用的是二进制，因为它只使用 0、1 两个数字符号，非常简单方便，

易于用电子方式实现。计算机内部处理的信息，都是采用二进制数来表示的。二进制（Binary）数用 0 和 1 两个数字及其组合来表示任何数。进位规则是“逢 2 进 1”，数字 1 在不同的位上代表不同的值，按从右至左的次序，这个值以二倍递增。

在计算机的内部，运行各种运算时，都是以二进制的方式来运行。

4.12.3 原码、反码、补码

网上对原码,反码,补码的解释过于复杂，我这里精简6句话:

对于有符号的而言:
1) 二进制的最高位是符号位: 0表示正数,1表示负数
1 ===》 [0000 0001] -1===>[1000 0001]

2) 正数的原码，反码，补码都一样
3) 负数的反码=它的原码符号位不变，其它位取反(0->1,1->0)
1===> 原码 [0000 0001] 反码 [0000 0001] **补码** [0000 0001]
-1===> 原码[1000 0001] 反码[1111 1110] **补码** [1111 1111]

4) 负数的补码=它的反码+1
5) 0的反码，补码都是0
6) 在**计算机运算的时候**，都是以**补码的方式来运算**的.
1+1 1-1 = 1 + (-1)

4.12.4 位运算符和移位运算符

- Golang 中有 3 个位运算

  分别是”按位与&、按位或|、按位异或^,它们的运算规则是：

  按位与&　　：　　两位全为 1 ，结果为 1，否则为 0

  按位或|　　：　　两位有一个为 1，结果为 1，否则为 0

  按位异或 ^　　：　两位一个为 0,一个为 1，结果为 1，否则为 0

- **案例练习**

  比如：2&3=? 2|3=? 2^3=?

```go
package main
import (
    "fmt"
)
func main() {

    //位运算的演示
    fmt.Println(2&3) // 2
    fmt.Println(2|3) // 3
    fmt.Println(2^3) // 3
    fmt.Println(-2^2) //-4

}
```

- Golang 中有 2 个移位运算符：

  >>、<< 右移和左移,运算规则:

  右移运算符 >>：**低位溢出,符号位不变,并用符号位补溢出的高位**

  左移运算符 <<: **符号位不变,低位补 0**

- **案例演示**

  a := 1 >> 2 // 0000 0001 =>0000 0000 = 0

```
c := 1 << 2    // 0000 0001 ==> 0000 0100 => 4
```

第 5 章 程序流程控制

5.1 程序流程控制介绍

在程序中，程序运行的流程控制决定程序是如何执行的，是我们必须掌握的，主要有三大流程控制语句。

1) 顺序控制
2) 分支控制
3) 循环控制

5.2 顺序控制

程序从上到下逐行地执行，中间没有任何判断和跳转。

一个案例说明，必须下面的代码中，没有判断，也没有跳转.因此程序按照默认的流程执行，即顺序控制。

```
7     //假如还有97天放假，问：xx个星期零xx天
8     var days int = 97
9     var week int = days / 7
10    var day int = days % 7
11    fmt.Printf("%d个星期零%d天\n", week, day)
12
13
14    //定义一个变量保存华氏温度，华氏温度转换摄氏温度的公式为：
15    //5/9*(华氏温度-100),请求出华氏温度对应的摄氏温度
16    var huashi float32 = 134.2
17    var sheshi float32 = 5.0 / 9 * (huashi - 100)
18    fmt.Printf("%v 对应的摄氏温度=%v \n", huashi, sheshi)
```

5.2.1 顺序控制的一个流程图

5.2.2顺序控制举例和注意事项

Golang 中定义变量时采用合法的前向引用。如：

```
func main() {
    var num1 int = 10 //声明了 num1
    var num2 int = num1 + 20 //使用 num1
    fmt.Println(num2)
}
```

错误形式：

```
func main() {
    var num2 int = num1 + 20 //使用 num1
    var num1 int = 10 //声明 num1 (×)
    fmt.Println(num2)
}
```

5.3 分支控制

5.3.1分支控制的基本介绍

分支控制就是让程序有选择执行。有下面三种形式

1) 单分支

2) 双分支

3) 多分支

5.3.2单分支控制

- 基本语法

```
if 条件表达式{
        执行代码块
}
```

说明：当条件表达式为ture时，就会执行{}的代码。
注意{}是必须有的，就算你只写一行代码。

- 应用案例

请大家看个案例[ifDemo.go]:

编写一个程序,可以输入人的年龄,**如果**该同志的年龄大于 18 岁,则输出 "你年龄大于 18,要对自己的行为负责!"　　需求---[分析]---->代码

代码:

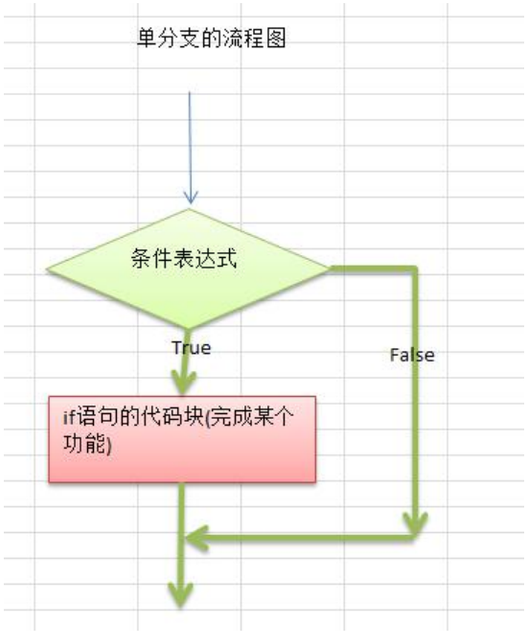
```
func main() {

    //请大家看个案例[ifDemo.go]:
    //编写一个程序,可以输入人的年龄,如果该同志的年龄大于18岁,则输出 "你年龄大
    //于18,要对自己的行为负责!"

    //分析
    //1.年龄 ==> var age int
    //2.从控制台接收一个输入 fmt.Scanln(&age)
    //3.if判断

    var age int
    fmt.Println("请输入年龄:")
    fmt.Scanln(&age)

    if age > 18 {
        fmt.Println("你年龄大于18,要对自己的行为负责!")
    }
}
```

输出的结果：

```
D:\goproject\src\go_code\chapter05\ifdemo>go run main.go
请输入年龄:
20
你年龄大于18,要对自己的行为负责!

D:\goproject\src\go_code\chapter05\ifdemo>go run main.go
请输入年龄:
1
```

- 单分支的流程图

  流程图可以用**图形方式**来更加清晰的描述程序执行的流程。



- 单分支的细节说明

```
//golang支持在if中，直接定义一个变量，比如下面
if age := 20; age > 18 {
    fmt.Println("你年龄大于18,要对自己的行为负责!")
}
```

5.3.3 双分支控制

- 基本语法

```
if 条件表达式{
        执行代码块1
}else{
        执行代码块2
}
```

说明：当条件表达式成立，即执行代码块1，否则执行代码块2。{} 也是必须有的。

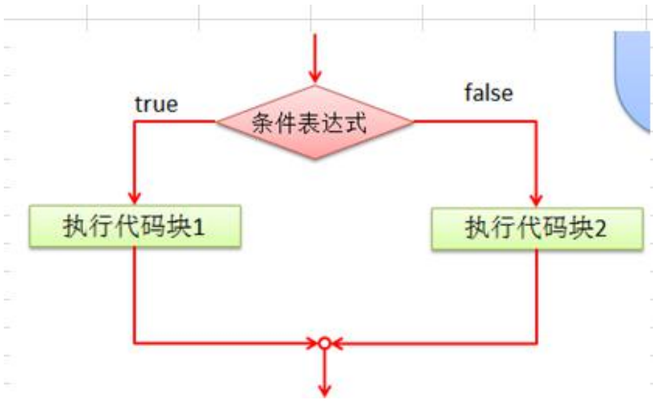
- 应用案例

  请大家看个案例[IfDemo2.go]:

  编写一个程序,可以输入人的年龄,**如果**该同志的年龄大于 18 岁,则输出 “你年龄大于 18,要对自己的行为负责!”。**否则** ,输出”你的年龄不大这次放过你了.”

```go
func main() {

    //请大家看个案例[IfDemo2.go]:
    //编写一个程序,可以输入人的年龄,如果该同志的年龄大于18岁,则输出 “你年龄大于18,要对
    //自己的行为负责!”。否则 ,输出”你的年龄不大这次放过你了.”

    //思路分析
    //1. 年龄 ===》 var age int
    //2. fmt.Scanln接收
    //3. if --- else

    //代码
    var age int
    fmt.Println("请输入年龄:")
    fmt.Scanln(&age)

    if age > 18 {
        fmt.Println("你年龄大于18....")
    } else {
        fmt.Println("你的年龄不大这次放过你了")
    }

}
```

- 双分支的流程图的分析




```
if age > 18 {
    fmt.Println("你年龄大于18....")
} else {
    fmt.Println("你的年龄不大这次放过你了")
}
```

对双分支的总结

1. 从上图看 条件表达式就是 age >18
2. 执行代码块 1 ===> fmt.Println("你的年龄大于 18") ..
3. 执行代码块 2 ===> fmt.Println("你的年龄不大....") .
4. **强调一下** 双分支只会执行其中的一个分支。

5.3.4单分支和双分支的案例

1) 对下列代码，若有输出，指出输出结果。

```
var x int = 4
var y int = 1
if (x > 2) {
   if (y > 2) {
      fmt.Println(x + y)
   }
      fmt.Println("atguigu")
} else {
   fmt.Println("x is = ", x)
}
//输出结果是 atguigu
```

2) 对下列代码，若有输出，指出输出结果。

```
var x int = 4
if x > 2
    fmt.Println("ok")
else
   fmt.Println("hello")

//编译错误，原因没有 {}
```

3) 对下列代码，若有输出，指出输出结果。

```
var x int = 4
if x > 2 {
    fmt.Println("ok")
}
else {
  fmt.Println("hello")
}
//编程错误，原因是 else 不能换行
```

4) 对下列代码，若有输出，指出输出结果。

```
var x int = 4
   if (x > 2) {
      fmt.Println("ok~")
   } else {
      fmt.Println("hello")
   }
//正确，输出 ok~
//虽然正确，但是我们要求大家 if (x > 2) 要求写成 if x > 2 { ...
```

5) 编写程序，声明 2 个 int32 型变量并赋值。判断两数之和，如果大于等于 50，打印“hello world!

```
func main() {
    //编写程序，声明2个int32型变量并赋值。判断两数之和，如果大于等于50，打印“hello world!”

    //分析
    //1. 变量
    //2. 单分支

    var n1 int32 = 10
    var n2 int32 = 50
    if n1 + n2 >= 50 {
        fmt.Println("hello,world")
    }
}
```

6) 编写程序，声明 2 个 float64 型变量并赋值。判断第一个数大于 10.0，且第 2 个数小于 20.0，打印两数之和。

```
19      //编写程序，声明2个float64型变量并赋值。判断第一个数大于10.0，
20      //且第2个数小于20.0，打印两数之和
21
22      var n3 float64 = 11.0
23      var n4 float64 = 17.0
24      if n3 > 10.0 && n4 < 20.0 {
25          fmt.Println("和=", (n3 + n4) )
26      }
```

7) 【选作】定义两个变量 int32，判断二者的和，是否能被 3 又能被 5 整除，打印提示信息

```
var num1 int32 = 10
var num2 int32 = 5
if (num1 + num2) % 3 == 0 &&  (num1 + num2) % 5 == 0 {
    fmt.Println("能被3又能被5整除")
}
```

8) 判断一个年份是否是闰年，闰年的条件是符合下面二者之一：(1)年份能被 4 整除，但不能被 100 整除；(2)能被 400 整除

```go
//8)判断一个年份是否是闰年，闰年的条件是符合下面二者之一：
//(1)年份能被4整除，但不能被100整除；(2)能被400整除
var year int = 2019
if (year % 4 == 0 && year % 100 !=0) || year % 400 == 0 {
    fmt.Println(year, "是润年~")
}
```

5.3.5 多分支控制

- 基本语法

```
if 条件表达式1 {
        执行代码块1
} else if 条件表达式2 {
        执行代码块2
}
.....
else {
        执行代码块n
}
```

对上面基本语法的说明

1) 多分支的判断流程如下:

(1) 先判断条件表达式 1 是否成立，如果为真，就执行代码块 1

(2) 如果条件表达式 1 如果为假，就去判断条件表达式 2 是否成立， 如果条件表达式 2 为真，就执行代码块 2

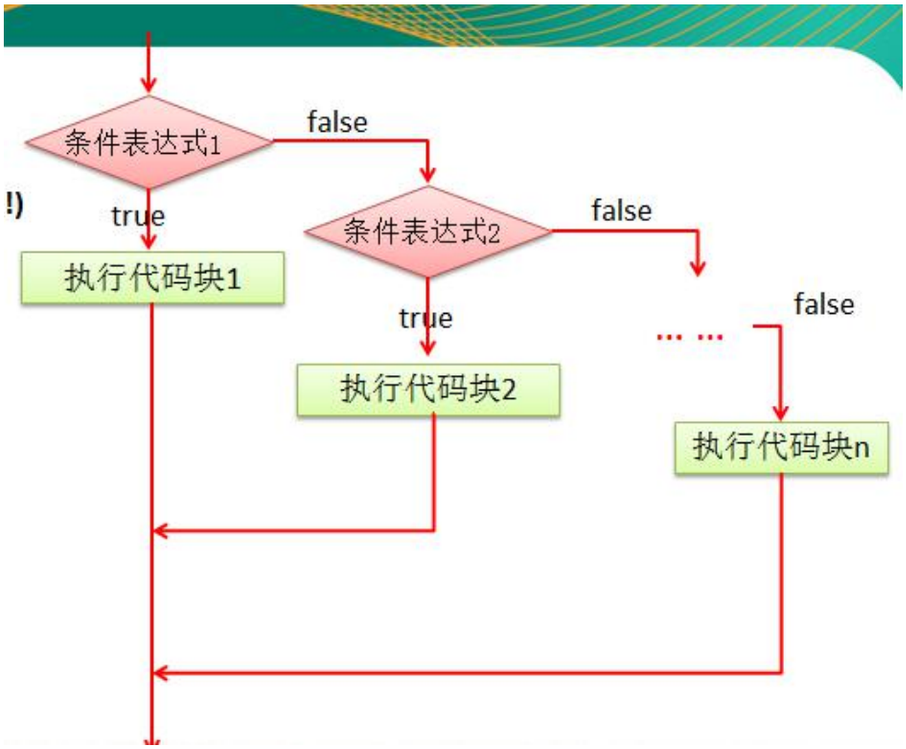
(3) 依次类推.

(4) 如果所有的条件表达式不成立，则执行 else 的语句块。

2) else 不是必须的。

3) **多分支只能有一个执行入口**。

- 看一个多分支的流程图(更加清晰)



- 多分支的快速入门案例

  岳小鹏参加 Golang 考试，他和父亲岳不群达成承诺：

  如果：

  成绩为 100 分时，奖励一辆 BMW；

  成绩为(80，99]时，奖励一台 iphone7plus；

  当成绩为[60,80]时，奖励一个 iPad；

  其它时，什么奖励也没有。

  请从键盘输入岳小鹏的期末成绩，并加以判断

  代码如下：

```go
// 其它时，什么奖励也没有。
// 请从键盘输入岳小鹏的期末成绩，并加以判断

//分析思路
//1. score 分数变量 int
//2. 选择多分支流程控制
//3. 成绩从键盘输入 fmt.Scanln

var score int
fmt.Println("请输入成绩:")
fmt.Scanln(&score)

//多分支判断
if score == 100 {
    fmt.Println("奖励一辆BMW")
} else if score > 80 && score <= 99 {
    fmt.Println("奖励一台iphone7plus")
} else if score >= 60 && score <= 80 {
    fmt.Println("奖励一个 iPad")
} else {
    fmt.Println("什么都不奖励")
}
```

对初学者而言，有一个使用陷阱.

```go
//使用陷阱.... 只会输出ok1...

var n int = 10
if n > 9 {
    fmt.Println("ok1") //输出 ok1
} else if  n > 6 {
    fmt.Println("ok2")
} else if n > 3 {
    fmt.Println("ok3")
} else {
    fmt.Println("ok4")
}
```

- 多分支的课堂练习

➢ 案例演示2

```
func main() {
    var b bool = true
    if b == false {        //如果写成 b = false; 能编译通过吗？如果能，结果是？
        fmt.Println("a")
    } else if b {
        fmt.Println("b")
    } else if !b {
        fmt.Println("c")//c
    } else {
        fmt.Println("d")
    }
}
//输出结果是b ,
//如果写成 b = false; 能编译通过吗？如果能，结果是？[编程错误，if的条件表达式不能是 赋值语句]
```

案例 3:

➢ 案例演示3

求$ax^2+bx+c=0$方程的根。a,b,c分别为函数的参数，如果：$b^2-4ac>0$，则有两个解；$b^2-4ac=0$，则有一个解；$b^2-4ac<0$，则无解；

提示1：$x_1=(-b+sqrt(b^2-4ac))/2a$

$x_2=(-b-sqrt(b^2-4ac))/2a$

提示2：math.Sqrt(num); 可以求平方根 需要引入 math包

测试数据: 3 ,100, 6

代码:

```go
//分析思路
//1. a,b,c 是三个float64
//2. 使用到给出的数学公式
//3. 使用到多分支
//4. 使用math.Squr方法 =》手册

//走代码
var a float64 = 2.0
var b float64 = 4.0
var c float64 = 2.0

m := b * b - 4 * a * c
//多分支判断
if m > 0 {
    x1 := (-b + math.Sqrt(m)) / 2 * a
    x2 := (-b - math.Sqrt(m)) / 2 * a
    fmt.Printf("x1=%v x2=%v", x1, x2)
} else if m == 0 {
    x1 := (-b + math.Sqrt(m)) / 2 * a
    fmt.Printf("x1=%v", x1)
} else {
    fmt.Println("无解...")
}
```

➢ 案例演示4

大家都知道，男大当婚，女大当嫁。那么女方家长要嫁女儿，当然要提出一定的条件：高：180cm以上；富：财富1千万以上；帅：是。条件从控制台输入。

1) 如果这三个条件同时满足，则：“我一定要嫁给他!!!”
2) 如果三个条件有为真的情况，则：“嫁吧，比上不足，比下有余。”
3) 如果三个条件都不满足，则：“不嫁！”

var height int32 | var money float32 | var handsome bool

代码：

```go
//分析思路
//1. 应该设计三个变量 var height int32  | var money float32 | var handsome bool
//2. 而且需要从终端输入 fmt.Scanln
//3. 使用多分支if--else if -- else
var height int32
var money float32
var handsome bool

fmt.Println("请输入身高(厘米)")
fmt.Scanln(&height)
fmt.Println("请输入财富(千万)")
fmt.Scanln(&money)
fmt.Println("请输入是否帅(true/false)")
fmt.Scanln(&handsome)

if height > 180 && money > 1.0 && handsome {
    fmt.Println("我一定要嫁给他!!!")
} else if height > 180 || money > 1.0 || handsome {
    fmt.Println("嫁吧，比上不足，比下有余")
} else {
    fmt.Println("不嫁....")
}
```

5.3.6嵌套分支

- 基本介绍

  在一个分支结构中又完整的嵌套了另一个完整的分支结构，里面的分支的结构称为内层分支外面的分支结构称为外层分支。

- 基本语法

```
基本语法
if 条件表达式{
    if 条件表达式{
    }else{
    }
}
```

说明：嵌套分支不宜过多，建议控制在3层内。

- 应用案例 1

  参加百米运动会，如果用时 8 秒以内进入决赛，否则提示淘汰。并且根据性别提示进入男子组或女

子组。【可以让学员先练习下】，输入成绩和性别。

代码:

```
//分析思路
//1. 定义一个变量，来接收跑步使用秒数. float64
//2. 定义一个变量，来接收性别string
//3. 因为判断是嵌套的判断，因此我们会使用嵌套分支

var second float64

fmt.Println("请输入秒数")
fmt.Scanln(&second)

if second <= 8 {
    //进入决赛
    var gender string
    fmt.Println("请输入性别")
    fmt.Scanln(&gender)
    if gender == "男" {
        fmt.Println("进入决赛的男子组")
    } else {
        fmt.Println("进入决赛的女子组")
    }
} else {
    fmt.Println("out...")
}
```

- 应用案例 2

出票系统：根据淡旺季的月份和年龄，打印票价 [考虑学生先做]

4_10 旺季：

成人（18-60）：60

儿童（<18）:半价

老人（>60）:1/3

淡季：

成人：40

其他：20

代码：

```go
    //分析思路
    //1.month age 的两个变量 byte
    //2.使用嵌套分支

    var month byte
    var age byte
    var price float64 = 60.0
    fmt.Println("请输入游玩月份")
    fmt.Scanln(&month)
    fmt.Println("请输入游客的年龄")
    fmt.Scanln(&age)

    if month >= 4 && month <= 10 {
        if age > 60 {
            fmt.Printf("%v月 票价 %v 年龄 %v ", month, price / 3 ,  age)
        } else if age >= 18 {
            fmt.Printf("%v月 票价 %v 年龄 %v ", month, price,  age)
        } else {
            fmt.Printf("%v月 票价 %v 年龄 %v ", month, price / 2,  age)
        }
    } else {
        //淡季
        if age >= 18 && age < 60 {
            fmt.Println("淡季成人 票价 40")
        } else {
            fmt.Println("淡季儿童和老人  票价 20")
        }
    }
}
```

5.4 switch 分支控制

5.4.1基本的介绍

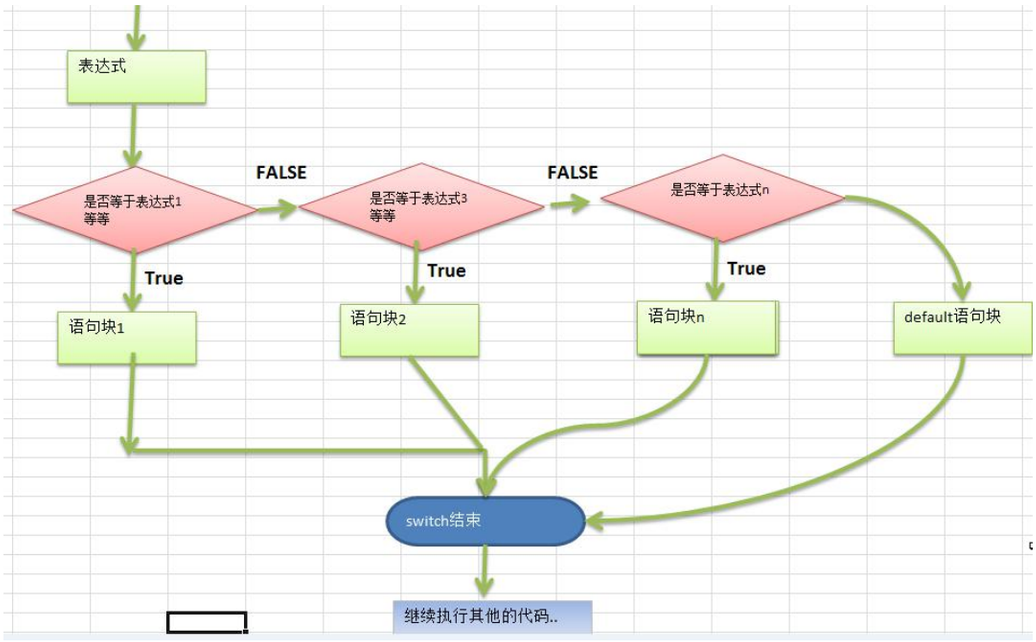
1) switch 语句用于基于不同条件执行不同动作，每一个 case 分支都是唯一的，从上到下逐一测试，直到匹配为止。

2) 匹配项后面也**不需要再加 break**

5.4.2基本语法

```
switch 表达式 {

case 表达式1, 表达式2, ... :
        语句块1

case 表达式3, 表达式4, ... :
        语句块2
//这里可以有多个case语句

default:
        语句块
}
```

5.4.3 switch 的流程图



➢ 对上图的说明和总结

1) switch 的执行的流程是，先执行表达式，得到值，然后和 case 的表达式进行比较，如果相等，就匹配到，然后执行对应的 case 的语句块，然后退出 switch 控制。

2) 如果 switch 的表达式的值没有和任何的 case 的表达式匹配成功，则执行 default 的语句块。执行

后退出 switch 的控制.

3) golang 的 case 后的表达式可以有多个，使用 逗号 间隔.

4) golang 中的 case 语句块不需要写 break，因为默认会有,即在默认情况下，当程序执行完 case 语句块后，就直接退出该 switch 控制结构。

5.4.4switch 快速入门案例

- 案例：

请编写一个程序，该程序可以接收一个字符，比如: a,b,c,d,e,f,g a 表示星期一，b 表示星期二 … 根据用户的输入显示相依的信息.要求使用 switch 语句完成

- 代码

```
//分析思路
//1. 定义一个变量接收字符
//2. 使用switch完成
var key byte
fmt.Println("请输入一个字符 a,b,c,d,e,f,g")
fmt.Scanf("%c", &key)

switch key {
    case 'a':
        fmt.Println("周一, 猴子穿新衣")
    case 'b':
        fmt.Println("周二, 猴子当小二")
    case 'c':
        fmt.Println("周三, 猴子爬雪山")
    //...
    default:
        fmt.Println("输入有误...")

}
```

5.4.5switch 的使用的注意事项和细节

1) case/switch 后是一个表达式(即：**常量值**、变量、一个有**返回值的函数**等都可以)

```go
//写一个非常简单的函数
func test(char byte) byte {
    return char + 1
}

func main() {

    //  案例:
    // 请编写一个程序，该程序可以接收一个字符，比如: a,b,c,d,e,f,g
    // a表示星期一，b表示星期二 …  根据用户的输入显示相依的信息.

    // 要求使用 switch 语句完成

    //分析思路
    //1. 定义一个变量接收字符
    //2. 使用switch完成
    var key byte
    fmt.Println("请输入一个字符 a,b,c,d,e,f,g")
    fmt.Scanf("%c", &key)

    switch test(key)+1 { //将语法现象
        case 'a':
            fmt.Println("周一，猴子穿新衣")
        case 'b':
            fmt.Println("周二，猴子当小二")
        case 'c':
            fmt.Println("周三，猴子爬雪山")
        //...
        default:
            fmt.Println("输入有误...")
    }


}
```

2) case 后的各个表达式的值的数据类型，必须和 switch 的表达式数据类型一致

```go
var n1 int32 = 20
var n2 int64 = 20
switch n1 {
    case n2 :  // 错误，原因是 n2的数据类型和n1不一致
        fmt.Println("ok1")
    default :
        fmt.Println("没有匹配到")
}
```

3) case 后面可以带多个表达式，使用逗号间隔。比如 case 表达式 1, 表达式 2 ...

```go
var n1 int32 = 5
var n2 int32 = 20
switch n1 {
    case n2, 10, 5 :  // case 后面可以有多个表达式
        fmt.Println("ok1")
    default :
        fmt.Println("没有匹配到")
}
```

4) case 后面的表达式如果是常量值(字面量)，则要求不能重复

```go
var n1 int32 = 5
var n2 int32 = 20
switch n1 {
    case n2, 10, 5 :  // case 后面可以有多个表达式
        fmt.Println("ok1")
    case 5 : // 错误，因为前面我们有常量5，因此重复，就会报错
        fmt.Println("ok2~")
    default :
        fmt.Println("没有匹配到")
}
```

5) case 后面不需要带 break，程序匹配到一个 case 后就会执行对应的代码块，然后退出 switch，如果一个都匹配不到，则执行 default

6) default 语句不是必须的.

7) switch 后也可以不带表达式，类似 if --else 分支来使用。【案例演示】

```go
//switch 后也可以不带表达式，类似 if --else分支来使用。【案例演示】
var age int = 10

switch {
    case age == 10 :
        fmt.Println("age == 10")
    case age == 20 :
        fmt.Println("age == 20")
    default :
        fmt.Println("没有匹配到")
}

//case 中也可以对 范围进行判断
var score int = 90
switch {
    case score > 90 :
        fmt.Println("成绩优秀..")
    case score >=70 && score <= 90 :
        fmt.Println("成绩优良...")
    case score >= 60 && score < 70 :
        fmt.Println("成绩及格...")
    default :
        fmt.Println("不及格")
}
```

8) switch 后也可以直接声明/定义一个变量，分号结束，不推荐。 【案例演示】

```go
//switch 后也可以直接声明/定义一个变量，分号结束，不推荐

switch grade := 90; { // 在golang中，可以这样写
    case grade > 90 :
        fmt.Println("成绩优秀~..")
    case grade >=70 && grade <= 90 :
        fmt.Println("成绩优良~...")
    case grade >= 60 && grade < 70 :
        fmt.Println("成绩及格~...")
    default :
        fmt.Println("不及格~")
}
```

9) switch 穿透-fallthrough ，如果在 case 语句块后增加 fallthrough ,则会继续执行下一个 case，也叫 switch 穿透

```go
//switch 的穿透 fallthrought
var num int = 10
switch num {
    case 10:
        fmt.Println("ok1")
        fallthrough //默认只能穿透一层
    case 20:
        fmt.Println("ok2")
        fallthrough
    case 30:
        fmt.Println("ok3")
    default:
        fmt.Println("没有匹配到..")
}
```

10) Type Switch：switch 语句还可以被用于 type-switch 来判断某个 interface 变量中实际指向的变量类型 【还没有学 interface，先体验一把】

```go
var x interface{}
var y = 10.0
x = y
switch i := x.(type) {
    case nil:
    fmt.Printf(" x 的类型~ :%T",i)
    case int:
    fmt.Printf("x 是 int 型")
    case float64:
    fmt.Printf("x 是 float64 型")
    case func(int) float64:
    fmt.Printf("x 是 func(int) 型")
    case bool, string:
    fmt.Printf("x 是 bool 或 string 型" )
    default:
    fmt.Printf("未知型")
}
```

5.4.6switch 的课堂练习

1) 使用 switch 把小写类型的 char 型转为大写(键盘输入)。只转换 a, b, c, d, e. 其它的输出“other”。

```go
func main(){
    //1)使用 switch 把小写类型的 char型转为大写(键盘输入)。
    //只转换 a, b, c, d, e. 其它的输出 “other”。

    var char byte
    fmt.Println("请输入一个字符..")
    fmt.Scanf("%c", &char)

    switch char {
        case 'a':
            fmt.Println("A")
        case 'b':
            fmt.Println("B")
        case 'c':
            fmt.Println("C")
        case 'd':
            fmt.Println("D")
        case 'e':
            fmt.Println("E")
        default :
            fmt.Println("other")
    }

}
```

2) 对学生成绩大于 60 分的，输出“合格”。低于 60 分的，输出“不合格”。(注：输入的成绩不能大于 100)

```go
    //2)对学生成绩大于60分的，输出“合格”。低于60分的，输出“不合格”。
    //(注：输入的成绩不能大于100)

    var score float64
    fmt.Println("请输入成绩")
    fmt.Scanln(&score)

    switch int(score / 60) {
        case 1:
            fmt.Println("及格")
        case 0:
            fmt.Println("不及格")
        default:
            fmt.Println("输入有误..")
    }
```

3) 根据用户指定月份，打印该月份所属的季节。3,4,5 春季 6,7,8 夏季 9,10,11 秋季 12, 1, 2 冬季

```go
//3)根据用户指定月份，
//打印该月份所属的季节。3,4,5 春季 6,7,8 夏季  9,10,11 秋季 12, 1, 2 冬季

var month byte
fmt.Println("请输入月份")
fmt.Scanln(&month)
switch month {
    case 3, 4, 5 :
        fmt.Println("spring")
    case 6, 7, 8 :
        fmt.Println("summer")
    case 9, 10, 11 :
        fmt.Println("autumn")
    case 12, 1, 2 :
        fmt.Println("winter")
    default:
        fmt.Println("输入有误..")
}
```

5.4.7switch 和 if 的比较

总结了什么情况下使用 switch ,什么情况下使用 if

1) 如果判断的**具体数值不多**，而且符合整数、浮点数、字符、字符串这几种类型。建议使用 **swtich** 语句，简洁高效。

2) 其他情况：对**区间判断**和结果为 **bool** 类型的判断，使用 if，**if 的使用范围更广**。

5.5 for 循环控制

5.5.1基本介绍

听其名而知其意。就是让我们的一段代码循环的执行。

5.5.2一个实际的需求

- 请大家看个案例 [forTest.go]:

  编写一个程序, 可以打印 10 句

  "你好，尚硅谷!"。请大家想想怎么做?

- 使用传统的方式实现

```
func main() {
    //输出10句 "你好，尚硅谷"

    // fmt.Println("你好，尚硅谷")
    // fmt.Println("你好，尚硅谷")
    // fmt.Println("你好，尚硅谷")
    // fmt.Println("你好，尚硅谷")
    // fmt.Println("你好，尚硅谷")
    // fmt.Println("你好，尚硅谷")
    // fmt.Println("你好，尚硅谷")
    // fmt.Println("你好，尚硅谷")
```

- for 循环的快速入门

```
//golang中，有循环控制语句来处理循环的执行某段代码的方法->for循环
//for循环快速入门
for i := 1; i <= 10; i++ {
    fmt.Println("你好，尚硅谷", i)
}
```

5.5.3 for 循环的基本语法

- 语法格式

for 循环变量初始化; 循环条件; 循环变量迭代 {

循环操作(语句)

}

- 对上面的语法格式说明

1) 对 for 循环来说，有四个要素：
2) 循环变量初始化
3) 循环条件
4) 循环操作(语句) ,有人也叫循环体。
5) 循环变量迭代

- for 循环执行的顺序说明：

1) 执行循环变量初始化，比如 i := 1
2) 执行循环条件， 比如 i <= 10

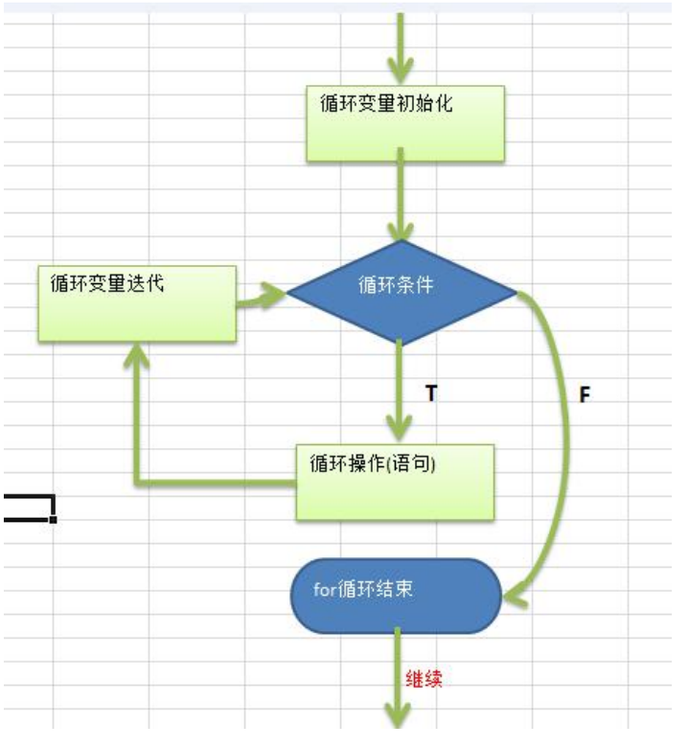
3) 如果循环条件为真，就执行循环操作 :比如 fmt.Println(“....”)

4) 执行循环变量迭代 ，比如 i++

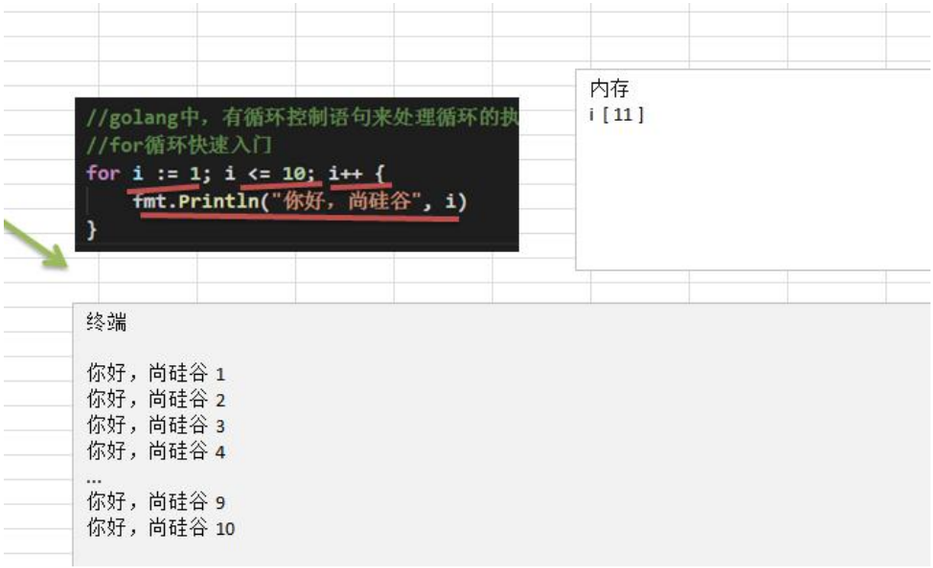
5) 反复执行 2, 3, 4 步骤，直到 循环条件为 False ，就退出 for 循环。

5.5.4 for 循环执行流程分析

- for 循环的流程图



- 对照代码分析 for 循环的执行过程



5.5.5 for 循环的使用注意事项和细节讨论

1) **循环条件**是返回一个**布尔值**的表达式

2) for 循环的第二种使用方式

```
for  循环判断条件  {
    //循环执行语句
}
```

将变量初始化和变量迭代写到其它位置

- 案例演示：

```
//for循环的第二种写法
j := 1 //循环变量初始化
for j <= 10 { //循环条件

    fmt.Println("你好，尚硅谷~", j)
    j++ //循环变量迭代
}
```

3) for 循环的第三种使用方式

```
for  {
    //循环执行语句
}
```

上面的写法**等价 for ; ; {}** 是一个**无限循环**， 通常需要**配合 break 语句**使用

```go
//for循环的第三种写法，这种写法通常会配合break使用
k := 1
for {  // 这里也等价 for ; ; {
    if k <= 10 {
        fmt.Println("ok~~", k)
    } else {
        break //break就是跳出这个for循环
    }
    k++
}
```

4) Golang 提供 for-range 的方式，可以方便遍历字符串和数组(注: 数组的遍历，我们放到讲数组的时候再讲解) ，**案例说明如何遍历字符串**。

- 字符串遍历方式 1-传统方式

```go
//字符串遍历方式1-传统方式
var str string = "hello,world!"
for i := 0; i < len(str); i++ {
    fmt.Printf("%c \n", str[i]) //使用到下标...
}
```

- 字符串遍历方式 2-for - range

```
fmt.Println()
//字符串遍历方式2-for-range
str = "abc~ok"
for index, val := range str {
    fmt.Printf("index=%d, val=%c \n", index, val)
}
```

- 上面代码的细节讨论

如果我们的字符串含有中文，那么传统的遍历字符串方式，就是错误，会出现乱码。原因是传统的对字符串的遍历是按照**字节来遍历**，而一个汉字在 utf8 编码是对应 **3 个字节**。

如何解决 需要要将 str 转成 []rune 切片.=> 体验一把

```
//字符串遍历方式1-传统方式
var str string = "hello,world!北京"
str2 := []rune(str) // 就是把 str 转成 []rune
for i := 0; i < len(str2); i++ {
    fmt.Printf("%c \n", str2[i]) //使用到下标...
}
```

对应 for-range 遍历方式而言，是按照字符方式遍历。因此如果有字符串有中文，也是 ok

```
fmt.Println()
//字符串遍历方式2-for-range
str = "abc~ok上海"
for index, val := range str {
    fmt.Printf("index=%d, val=%c \n", index, val)
}
```

5.5.6 for 循环的课堂练习

1) 打印 1~100 之间所有是 9 的倍数的整数的个数及总和

```go
func main() {
    //打印1~100之间所有是9的倍数的整数的个数及总和

    //分析思路
    //1. 使用for循环对 max 进行遍历
    //2. 当一个数%9 ==0 就是9的倍数
    //3. 我们需要声明两个变量 count 和 sum 来保存个数和总和
    var max uint64 = 100
    var count uint64 = 0
    var sum uint64 = 0
    var i uint64 = 1
    for ; i <= max; i++ {
        if i % 9 == 0 {
            count++
            sum += i
        }
    }
    fmt.Printf("count=%v sum=%v\n", count, sum)
}
```

2) 完成下面的表达式输出 ，6 是可变的。

```
0  +  6  =  6
1  +  5  =  6
2  +  4  =  6
3  +  3  =  6
4  +  2  =  6
5  +  1  =  6
6  +  0  =  6
```

```go
fmt.Println("---------------------------------")
//完成下面的表达式输出 ，6是可变的
var n int = 60
for i := 0; i <= n; i++ {
    fmt.Printf("%v + %v = %v \n", i, n - i, n)
}
```

5.6 while 和 do..while 的实现

Go 语言没有 while 和 do...while 语法，这一点需要同学们注意一下，如果我们需要使用类似其它语言(比如 java / c 的 while 和 do...while)，**可以通过 for 循环来实现其使用效果**。

5.6.1 while 循环的实现

```
循环变量初始化
for {
   if 循环条件表达式 {
      break //跳出for循环..
   }
   循环操作(语句)
   循环变量迭代
}
说明:
```

- 说明上图

  1) for 循环是一个无限循环

  2) break 语句就是跳出 for 循环

- 使用上面的 while 实现完成输出 10 句”hello,wrold”

```go
func main(){

    //使用while方式输出10句 "hello,world"
    //循环变量初始化
    var i int = 1
    for {
        if i > 10 { //循环条件
            break // 跳出for循环,结束for循环
        }
        fmt.Println("hello,world", i)
        i++ //循环变量的迭代
    }

    fmt.Println("i=", i)

}
```

5.6.2do..while 的实现

```
循环变量初始化
for {
   循环操作(语句)
   循环变量迭代
   if 循环条件表达式 {
      break //跳出for循环..
   }
}
说明:
```

- 对上图的说明

  1) 上面的循环是先执行，在判断，因此至少执行一次。

  2) 当循环条件成立后，就会执行 break, break 就是跳出 for 循环，结束循环.

- 案例演示

  使用上面的 do...while 实现完成输出 10 句”hello,ok”

```go
//使用的do...while实现完成输出10句"hello,ok"
var j int = 1
for {
    fmt.Println("hello,ok", j)
    j++ //循环变量的迭代
    if j > 10 {
        break //break 就是跳出for循环
    }
}
```

5.7 多重循环控制(重点，难点)

5.7.1 基本介绍

1) 将一个循环放在另一个循环体内，就形成了嵌套循环。在外边的 for 称为外层循环在里面的 for 循环称为内层循环。【**建议一般使用两层，最多不要超过 3 层**】

2) 实质上，嵌套循环就是把内层循环当成外层循环的循环体。当只有内层循环的循环条件为 false 时，才会完全跳出内层循环，才可结束外层的当次循环，开始下一次的循环。

3) 外层循环次数为 **m** 次，内层为 **n** 次，则内层循环体实际上需要执行 **m*n** 次

5.7.2应用案例

1) 统计 3 个班成绩情况，每个班有 5 名同学，求出各个班的平均分和所有班级的平均分[学生的成绩从键盘输入]

编程时两大绝招

(1) 先易后难， 即将一个复杂的问题分解成简单的问题。

(2) 先死后活

代码:

```go
package main
import "fmt"
func main(){

    //1)统计3个班成绩情况，每个班有5名同学，
    //求出各个班的平均分和所有班级的平均分[学生的成绩从键盘输入]

    //分析实现思路
    //1. 统计1个班成绩情况，每个班有5名同学，求出该班的平均分【学生的成绩从键盘输入】=》先易后
    //2. 学生数就是5个 [先死后活]
    //3. 声明一个sum 统计班级的总分

    //分析实现思路2
    //1. 统计3个班成绩情况，每个班有5名同学，求出每个班的平均分【学生的成绩从键盘输入】
    //2. j 表示第几个班级
    //3. 定义一个变量存放总成绩

    //分析实现思路3
    //1. 我们可以把代码做活
    //2. 定义两个变量，表示班级的个数和班级的人数


    //走代码实现
    var classNum int = 2
    var stuNum int = 5
    var totalSum float64 = 0.0
```

```
    var totalSum float64 = 0.0
    for j := 1; j <= classNum; j ++ {
        sum := 0.0
        for i := 1; i <= stuNum; i++ {
            var score float64
            fmt.Printf("请输入第%d班 第%d个学生的成绩 \n", j, i)
            fmt.Scanln(&score)
            //累计总分
            sum += score
        }

        fmt.Printf("第%d个班级的平均分是%v\n", j, sum / float64(stuNum) )
        //将各个班的总成绩累计到totalSum
        totalSum += sum
    }

    fmt.Printf("各个班级的总成绩%v 所有班级平均分是%v\n", totalSum, totalSum / float64(stuN

}
```

2) 统计三个班及格人数，每个班有 5 名同学

对上面的代码进行了一点修改.

```
    //统计三个班及格人数，每个班有5名同学
    //分析思路
    //1. 声明以变量 passCount 用于保存及格人数

    //走代码实现
    var classNum int = 2
    var stuNum int = 5
    var totalSum float64 = 0.0
    var passCount int = 0
    for j := 1; j <= classNum; j ++ {
        sum := 0.0
        for i := 1; i <= stuNum; i++ {
```

```
for j := 1; j <= classNum; j ++ {
    sum := 0.0
    for i := 1; i <= stuNum; i++ {
        var score float64
        fmt.Printf("请输入第%d班 第%d个学生的成绩 \n", j, i)
        fmt.Scanln(&score)
        //累计总分
        sum += score
        //判断分数是否及格
        if score >= 60 {
            passCount++
        }
    }
```

3) 打印金字塔经典案例

使用 for 循环完成下面的案例请编写一个程序，可以接收一个整数,表示层数，打印出金字

- 分析编程思路
- 走代码

```
package main
import "fmt"
func main() {

    //使用 for 循环完成下面的案例请编写一个程序，可以接收一个整数,表示层数，打印出金字塔

    //编程思路
    //1. 打印一个矩形
    /*

        ***
        ***
        ***
    */

    //2. 打印半个金字塔
    /*
        *    1 个 *
        **   2 个 *
        ***  3 个 *
    */

    //3 打印整个金字塔
    /*
        *     1层 1 个*  规律: 2 * 层数 - 1   空格 2 规律 总层数-当前层数
       ***    2层 3 个*  规律: 2 * 层数 - 1   空格 1 规律 总层数-当前层数
```

```
27      *****   3层 5 个*  规律: 2 * 层数 - 1   空格 0 规律 总层数-当前层数
28  */
29  //4 将层数做成一个变量, 先死后活
30  //var totalLevel int
31
32  //5 打印空心金字塔
33  /*
34      *
35     * *
36    *****
37     分析: 在我们给每行打印*号时, 需要考虑是打印 * 还是打印 空格
38     我们的分析的结果是, 每层的第一个和最后一个是打印*, 其它就应该是空的, 即输出空格
39     我们还分析到一个例外情况, 最后层(底层)是全部打*
40
41  */
42
43  var totalLevel int = 20
44
45  //i 表示层数
46  for i := 1; i <= totalLevel; i++ {
47      //在打印*前先打印空格
48      for k := 1; k <= totalLevel - i; k++ {
49          fmt.Print(" ")
50      }
51
52      //j 表示每层打印多少*
53      for j :=1; j <= 2 * i - 1; j++ {
54          if j == 1 || j == 2 * i - 1 || i == totalLevel {
55              fmt.Print("*")
56          } else {
57              fmt.Print(" ")
58          }
59
60      }
61      fmt.Println()
62  }
63
64
65
66  }
```

4) 打印出九九乘法表

```
1 * 1 = 1
1 * 2 = 2   2 * 2 = 4
1 * 3 = 3   2 * 3 = 6   3 * 3 = 9
1 * 4 = 4   2 * 4 = 8   3 * 4 = 12  4 * 4 = 16
1 * 5 = 5   2 * 5 = 10  3 * 5 = 15  4 * 5 = 20  5 * 5 = 25
1 * 6 = 6   2 * 6 = 12  3 * 6 = 18  4 * 6 = 24  5 * 6 = 30  6 * 6 = 36
1 * 7 = 7   2 * 7 = 14  3 * 7 = 21  4 * 7 = 28  5 * 7 = 35  6 * 7 = 42  7 * 7 = 49
1 * 8 = 8   2 * 8 = 16  3 * 8 = 24  4 * 8 = 32  5 * 8 = 40  6 * 8 = 48  7 * 8 = 56  8 * 8 = 64
1 * 9 = 9   2 * 9 = 18  3 * 9 = 27  4 * 9 = 36  5 * 9 = 45  6 * 9 = 54  7 * 9 = 63  8 * 9 = 72  9 * 9 = 81
```

代码:

```go
//打印出九九乘法表
//i 表示层数
var num int = 9
for i := 1; i <= num; i++ {
    for j := 1; j <= i; j++ {
        fmt.Printf("%v * %v = %v \t", j, i, j * i)
    }
    fmt.Println()
}
```

5.8 跳转控制语句-break

5.8.1看一个具体需求，引出 break

随机生成 1-100 的一个数，直到生成了 99 这个数，看看你一共用了几次?

分析：编写一个无限循环的控制，然后**不停的随机生成数**，当生成了 99 时，就退出这个无限循环

==》break 提示使用

这里我们给大家说一下，如下随机生成 1-100 整数.

```go
//在go中，需要生成一个随机额种子，否则返回的值总是固定的.
// time.Now().Unix() : 返回一个从 1970 1-1 0:0:0 到现在的一个秒数
rand.Seed(time.Now().Unix())
fmt.Println("n", rand.Intn(100)+1)
```

5.8.2break 的快速入门案例

```go
func main() {

    //我们为了生成一个随机数，还需要个rand设置一个种子.
    //time.Now().Unix() : 返回一个从1970:01:01 的0时0分0秒到现在的秒数
    //rand.Seed(time.Now().Unix())
    //如何随机的生成1-100整数
    //n := rand.Intn(100) + 1 // [0 100)
    //fmt.Println(n)

    //随机生成1-100的一个数，直到生成了99这个数，看看你一共用了几次
    //分析思路:
    //编写一个无限循环的控制，然后不停的随机生成数，当生成了99时，就退出这个无限循环==》break
    var count int = 0
    for {
        rand.Seed(time.Now().UnixNano())
        n := rand.Intn(100) + 1
        fmt.Println("n=", n)
        count++
        if (n == 99) {
            break //表示跳出for循环
        }
    }

    fmt.Println("生成 99 一共使用了 ", count)
}
```

5.8.3基本介绍：

break 语句用于终止某个语句块的执行，用于中断当前 for 循环或跳出 switch 语句。

5.8.4基本语法：

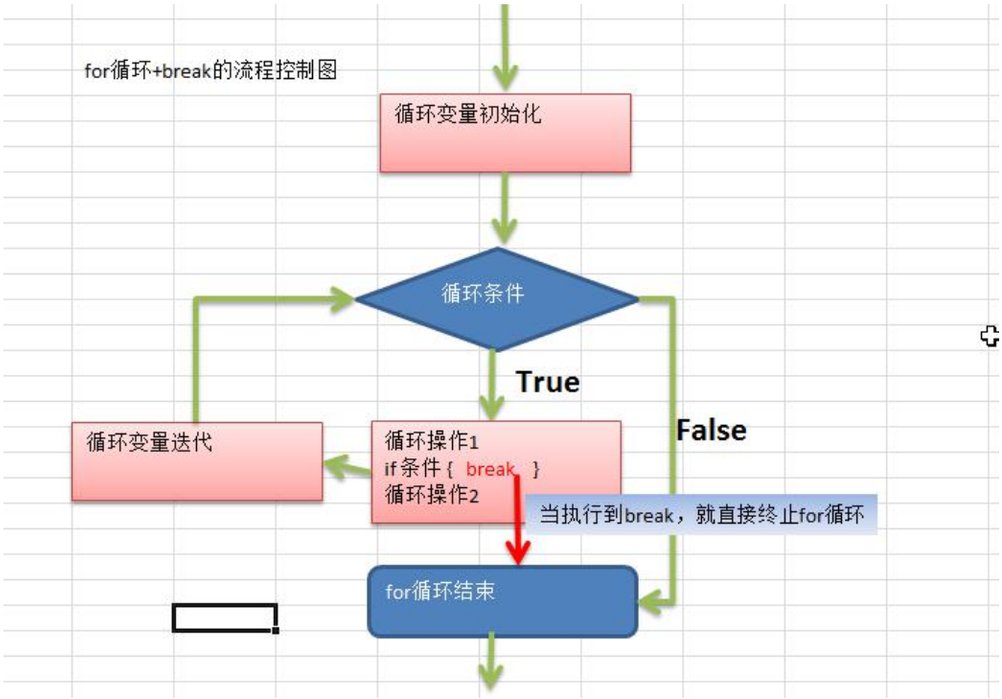
{ ……

break

……

}

5.8.5以 for 循环使用 break 为例,画出示意图



5.8.6break 的注意事项和使用细节

1) break 语句出现在多层嵌套的语句块中时，可以**通过标签**指明要终止的是哪一层语句块

2) 看一个案例

```
//这里演示一下指定标签的形式来使用 break
lable2:
for i := 0; i < 4; i++ {
    //lable1: // 设置一个标签
    for j := 0; j < 10; j++ {
        if j == 2 {
            //break // break 默认会跳出最近的for循环
            //break lable1
            break lable2 // j=0 j=1
        }
        fmt.Println("j=", j)
    }
}
```

3) 对上面案例的说明

(1) break 默认会跳出最近的 for 循环

(2) break 后面可以指定标签，跳出标签对应的 for 循环

5.8.7课堂练习

1) 100 以内的数求和，求出 当和 第一次大于 20 的当前数

```
    //100以内的数求和，求出 当和 第一次大于20的当前数
    sum := 0
    for i := 1; i <= 100; i++ {
        sum += i //求和
        if sum > 20 {
            fmt.Println("当sum>20时，当前数是", i)
            break
        }
    }
```

2) 实现登录验证，有三次机会，如果用户名为”张无忌” ,密码”888”提示登录成功，否则提示还有几次机会.

```
//实现登录验证，有三次机会，如果用户名为"张无忌" ,密码"888"提示登录成功，
//否则提示还有几次机会.

var name string
var pwd string
var loginChance = 3 //
for i := 1 ; i <= 3; i++ {
    fmt.Println("请输入用户名")
    fmt.Scanln(&name)
    fmt.Println("请输入密码")
    fmt.Scanln(&pwd)

    if name == "张无忌" && pwd == "888" {
        fmt.Println("恭喜你登录成功!")
        break
    } else {
        loginChance--
        fmt.Printf("你还有%v次登录机会，请珍惜\n", loginChance)
    }
}

if loginChance == 0 {
    fmt.Println("机会用完，没有登录成功!")
}
```

5.9 跳转控制语句-continue

5.9.1基本介绍：

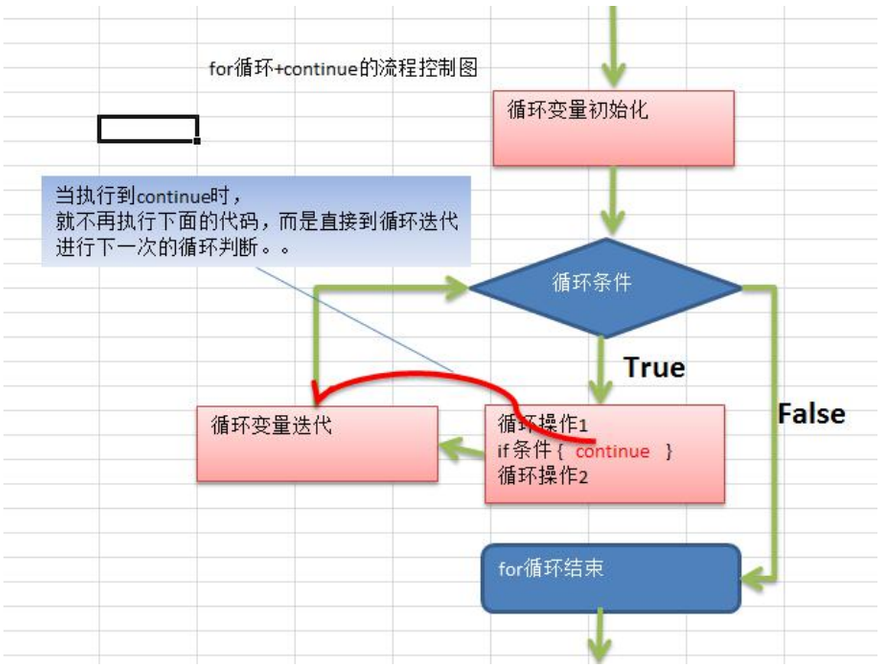
continue 语句用于**结束本次循环**，继续**执行下一次循环**。

continue 语句出现在多层嵌套的循环语句体中时，**可以通过标签指明要跳过的是哪一层循环** ，这个和前面的 break 标签的使用的规则一样.

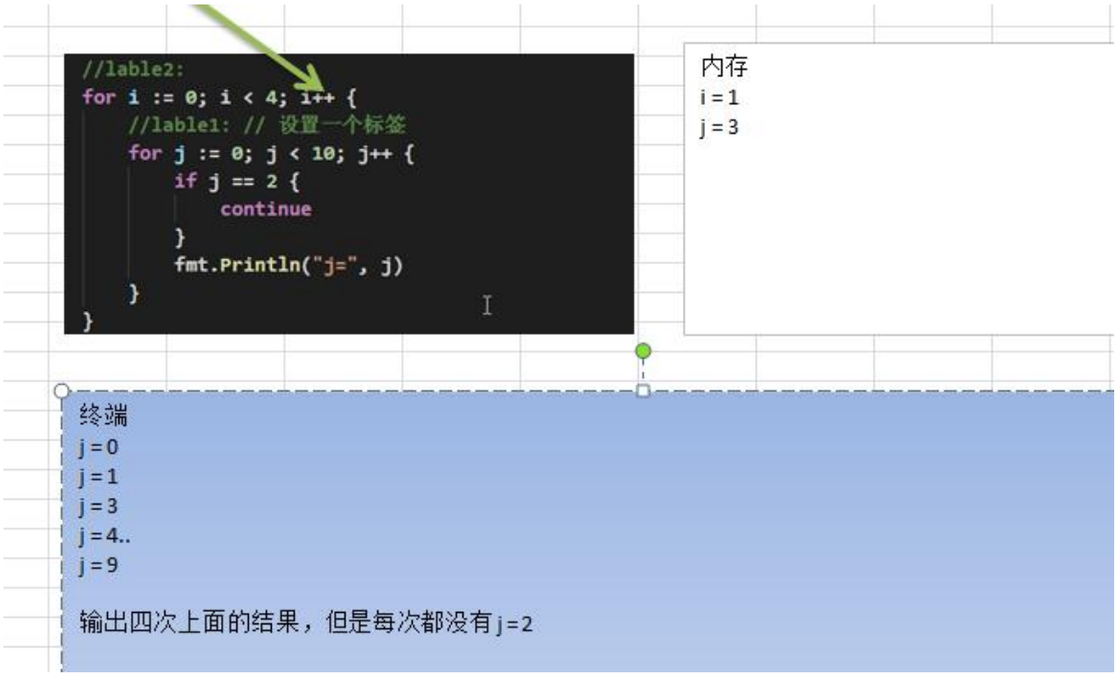
5.9.2基本语法：

```
{
    ……
    continue
    ……
}
```

5.9.3continue 流程图



5.9.4案例分析 continue 的使用



5.9.5continu 的课堂练习

- 练习 1

```
for i := 0; i<13; i++ {
if i == 10 {
continue;
}
fmt.Println("i=", i)
}

0 1 2 3 4 5 6 7 8 9 11 12
```

```
for i := 0; i<2; i++ {
for j := 1; j<4; j++ {
if j == 2 {
continue;
}
fmt.Println( "i=",i,"j=",j)
}
}
i = 0 j = 1
i = 0 j = 3
i = 1 j =1
i = 1 j = 3
```

```
here:
for i:=0; i<2; i++ {
for j:=1; j<4; j++ {
if j==2 {
continue here
}
fmt.Println("i=",i,"j=",j)
}
}
i = 0 j = 1
i = 1 j = 1
```

- continue 实现 打印 1——100 之内的奇数[要求使用 for 循环+continue]

  代码：

```go
package main
import "fmt"
func main() {
    //continue实现 打印1—100之内的奇数[要求使用for循环+continue]

    for i := 1; i <= 100; i++ {
        if i % 2 == 0 {
            continue
        }
        fmt.Println("奇数是", i)
    }
}
```

- 从键盘读入个数不确定的整数，并判断读入的正数和负数的个数，输入为 0 时结束程序

```go
    //从键盘读入个数不确定的整数，并判断读入的正数和负数的个数，输入为0时结束程序

    var positiveCount int // 正数的个数
    var negativeCount int // 负数个数
    var num int
    for {
        fmt.Println("请输入一个整数")
        fmt.Scanln(&num)
        if num == 0 {
            break //终止for循环
        }

        if num > 0 {
            positiveCount++
            continue//结束本次循环，进入下次循环
        }
        negativeCount++
    }
    fmt.Printf("正数个数是%v 负数的个数是%v\n", positiveCount, negativeCount)
```

- 课后练习题(同学们课后自己完成)：

  某人有 100,000 元,每经过一次路口，需要交费,规则如下:

  当现金>50000 时,每次交 5%

  当现金<=50000 时,每次交 1000

  编程计算该人可以经过多少次路口,使用 for break 方式完成

5.10 跳转控制语句-goto

5.10.1 goto 基本介绍

1) Go 语言的 goto 语句可以无条件地转移到程序中指定的行。

2) goto 语句通常与条件语句配合使用。可用来实现条件转移，跳出循环体等功能。

3) 在 Go 程序设计中**一般不主张使用 goto 语句**， 以免造成程序流程的混乱，使理解和调试程序都产生困难

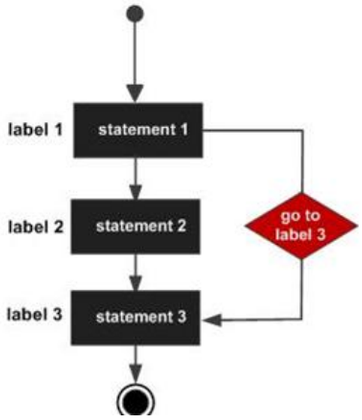
5.10.2 goto 基本语法

goto label

.. .

label: statement

5.10.3 goto 的流程图



5.10.4 快速入门案例

```go
func main() {

    var n int = 30
    //演示goto的使用
    fmt.Println("ok1")
    if n > 20 {
        goto label1
    }
    fmt.Println("ok2")
    fmt.Println("ok3")
    fmt.Println("ok4")
    label1:
    fmt.Println("ok5")
    fmt.Println("ok6")
    fmt.Println("ok7")
}
```

5.11 跳转控制语句-return

5.11.1 介绍：

return 使用在方法或者函数中，表示跳出所在的方法或函数，在**讲解函数的时候，会详细的介绍**。

```go
func main() {

    for i:=1; i<=10; i++ {

        if i==3 {
            return
        }
        fmt.Println("哇哇", i)
    }

    fmt.Println("Hello World!")
}
```

说明

1) 如果 return 是在普通的函数，则表示跳出该函数，即不再执行函数中 return 后面代码，也可以理解成终止函数。

2) 如果 return 是在 main 函数，表示终止 main 函数，也就是说终止程序。

第 6 章 函数、包和错误处理

6.1 为什么需要函数

6.1.1 请大家完成这样一个需求:

输入两个数,再输入一个运算符(+,-,*,/)，得到结果.。

6.1.2 使用传统的方法解决

- 走代码

```
4   func main() {
5
6       //请大家完成这样一个需求:
7       //输入两个数,再输入一个运算符(+,-,*,/)，得到结果..
8
9       //分析思路....
10      var n1 float64 = 1.2
11      var n2 float64 = 2.3
12      var operator byte = '-'
13      var res float64
14      switch operator {
15          case '+':
16              res = n1 + n2
17          case '-':
18              res = n1 - n2
19          case '*':
20              res = n1 * n2
21          case '/':
22              res = n1 / n2
23          default:
24              fmt.Println("操作符号错误...")
25      }
26      fmt.Println("res=", res)
27  }
```

- 分析一下上面代码问题

1) 上面的写法是可以完成功能，但是代码冗余
2) 同时不利于代码维护
3) 函数可以解决这个问题

6.2 函数的基本概念

为完成某一功能的程序指令(语句)的集合,称为函数。

在 Go 中,函数分为: 自定义函数、系统函数(查看 Go 编程手册)

6.3 函数的基本语法

```
func 函数名（形参列表）（返回值列表）{
        执行语句...
        return 返回值列表
}
```

1) 形参列表：表示函数的输入
2) 函数中的语句：表示为了实现某一功能代码块
3) 函数可以有返回值,也可以没有

6.4 快速入门案例

使用函数解决前面的计算问题。

走代码:

```go
//将计算的功能，放到一个函数中，然后在需要使用，调用即可
func cal(n1 float64, n2 float64, operator byte) float64 {

    var res float64
    switch operator {
        case '+':
            res = n1 + n2
        case '-':
            res = n1 - n2
        case '*':
            res = n1 * n2
        case '/':
            res = n1 / n2
        default:
            fmt.Println("操作符号错误...")
    }
    return res
}
```

```go
func main() {
    //请大家完成这样一个需求:
    //输入两个数,再输入一个运算符(+,-,*,/)，得到结果..
    //分析思路....
    var n1 float64 = 1.2
    var n2 float64 = 2.3
    var operator byte = '+'
    result := cal(n1, n2, operator)
    fmt.Println("result=", result)

}
```

调用函数

6.5 包的引出

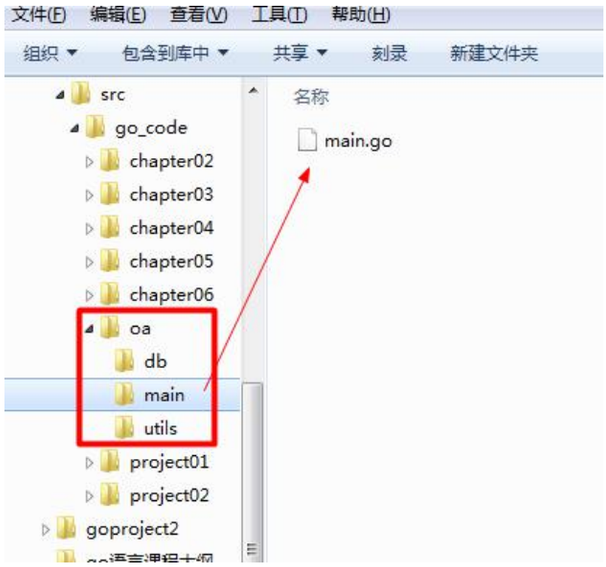
1) 在实际的开发中，我们往往需要在不同的文件中，去调用其它文件的定义的函数，比如 main.go 中，去使用 utils.go 文件中的函数，如何实现？ -》包

2) 现在有两个程序员共同开发一个 Go 项目,程序员 xiaoming 希望定义函数 Cal ,程序员 xiaoqiang 也想定义函数也叫 Cal。两个程序员为此还吵了起来,怎么办? -》包

6.6 包的原理图

包的本质实际上就是创建不同的文件夹，来存放程序文件。

画图说明一下包的原理



6.7 包的基本概念

说明：go 的每一个文件都是属于一个包的，也就是说 go 是以包的形式来管理文件和项目目录结构的

6.8 包的三大作用

区分相同名字的函数、变量等标识符

当程序文件很多时,可以很好的管理项目

控制函数、变量等访问范围，即作用域

6.9 包的相关说明

- 打包基本语法

  package 包名

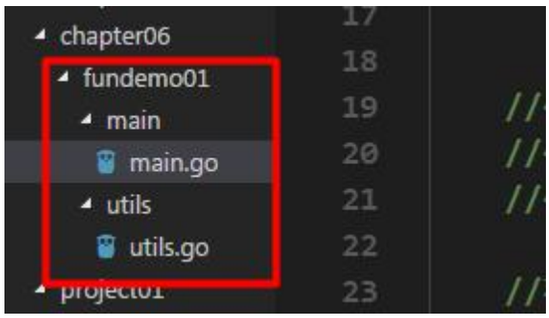
- 引入包的基本语法

  import "包的路径"

6.10 包使用的快速入门

包快速入门-Go 相互调用函数，我们将 func Cal 定义到文件 utils.go，将 utils.go 放到一个包中，当其它文件需要使用到 utils.go 的方法时，可以 import 该包，就可以使用了. 【为演示：新建项目目录结构】

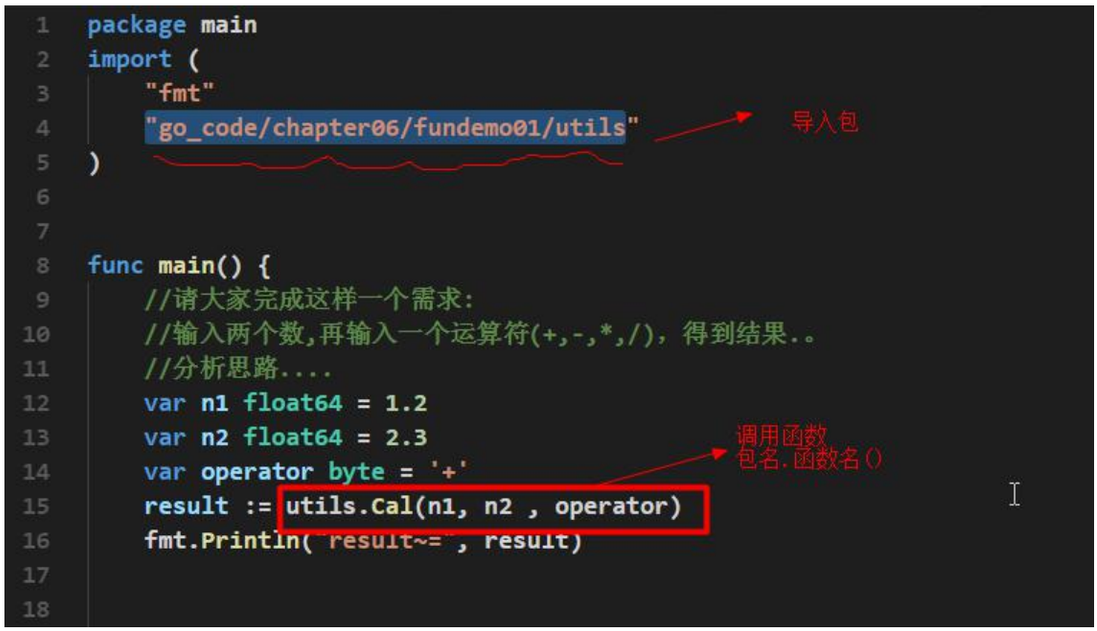
代码演示：



utils.go 文件

```go
package utils
import (
    "fmt"
)
//将计算的功能，放到一个函数中，然后在需要使用，调用即可
//为了让其它包的文件使用Cal函数，需要将C大小类似其它语言的public
func Cal(n1 float64, n2 float64, operator byte) float64 {

    var res float64
    switch operator {
        case '+':
            res = n1 + n2
        case '-':
            res = n1 - n2
        case '*':
            res = n1 * n2
        case '/':
            res = n1 / n2
        default:
            fmt.Println("操作符号错误...")
    }
    return res
}
```

main.go 文件




```go
package main
import (
    "fmt"
    "go_code/chapter06/fundemo01/utils"
)

func main() {
    //请大家完成这样一个需求:
    //输入两个数,再输入一个运算符(+,-,*,/),得到结果..
    //分析思路....
    var n1 float64 = 1.2
    var n2 float64 = 2.3
    var operator byte = '+'
    result := utils.Cal(n1, n2 , operator)
    fmt.Println("result~=", result)

}
```

6.11 包使用的注意事项和细节讨论

1) 在给一个文件打包时，该包对应一个文件夹，比如这里的 utils 文件夹对应的包名就是 utils, 文件的包名通常和文件所在的文件夹名一致，一般为小写字母。

2) 当一个文件要使用其它包函数或变量时，需要先引入对应的包

- 引入方式 1：import "包名"
- 引入方式 2：

```
import (
    "包名"
    "包名"
)
```

- package 指令在 文件第一行，然后是 import 指令。
- 在 import 包时，路径从 $GOPATH 的 src 下开始，不用带 src , 编译器会自动从 src 下开始引入

3) 为了让其它包的文件，可以访问到本包的函数，则该**函数名的首字母需要大写**，类似其它语言的 public ,这样才能跨包访问。比如 utils.go 的

```
func Cal(num1 int, num2 int, operator string) {
    result := 0
    switch operator {
        case "+":
            result = num1 + num2
        case "-":
```

4) 在访问其它包函数，变量时，其语法是 包名.函数名， 比如这里的 main.go 文件中

```
utils.Cal(90, 80, "+")
utils.Cal(90, 80, "-")
```

5) 如果包名较长，Go 支持给包取别名， 注意细节：取别名后，原来的包名就不能使用了

```
package main
import (
    "fmt"
    util "go_code/chapter06/fundemo01/utils"
)
```

说明: 如果给包取了别名，**则需要使用别名来访问该包的函数和变量**。

6) 在同一包下，不能有相同的函数名（也不能有相同的全局变量名），否则报重复定义

7) 如果你要编译成一个可执行程序文件，就需要将这个包声明为 main，即 package main .这个就是一个语法规范，如果你是写一个库 ，包名可以自定义