Go语言import导入包——在代码中使用其他的代码

< 上一页 下一页 >

可以在一个 Go语言源文件包声明语句之后，其它非导入声明语句之前，包含零到多个导入包声明语句。每个导入声明可以单独指定一个导入路径，也可以通过圆括号同时导入多个导入路径。要引用其他包的标识符，可以使用 import 关键字，导入的包名使用双引号包围，包名是从 GOPATH 开始计算的路径，使用 `/` 进行路径分隔。

默认导入的写法

导入有两种基本格式，即单行导入和多行导入，两种导入方法的导入代码效果是一致的。

1) 单行导入

单行导入格式如下：

```
import "包1"
import "包2"
```

2) 多行导入

当多行导入时，包名在 import 中的顺序不影响导入效果，格式如下：

```
import(
    "包1"
    "包2"
    ...
)
```

参考代码 8-1 的例子来理解 import 的机制。

> 本套教程所有源码下载地址：https://pan.baidu.com/s/1ORFVTOLEYYqDhRzeq0zIiQ 提取密码：hfyf

代码 8-1 的目录层次如下：

```
.
└── src
    └── chapter08
        └── importadd
            ├── main.go
            └── mylib
                └── add.go
```

代码8-1 加函数（具体文件：.../chapter08/importadd/mylib/add.go）

```
01.  package mylib
02.
03.  func Add(a, b int) int {
04.      return a + b
05.  }
```

第 3 行中的 Add() 函数以大写 A 开头，表示将 Add() 函数导出供包外使用。当首字母小写时，为包内使用，包外无法引用到。

add.go 在 mylib 文件夹下，习惯上将文件夹的命名与包名一致，命名为 mylib 包。

代码8-2　导入包（具体文件：.../chapter08/importadd/main.go）

```
01.  package main
02.
03.  import (
04.      "chapter08/importadd/mylib"
05.      "fmt"
06.  )
07.
08.  func main() {
09.      fmt.Println(mylib.Add(1, 2))
10.  }
```

代码说明如下：

- 第 4 行，导入 chapter08/importadd/mylib 包。
- 第 9 行，使用 mylib 作为包名，并引用 Add() 函数调用。

在命令行中运行下面代码：

```
export GOPATH=/home/davy/golangbook/code
go install chapter08/importadd
$GOPATH/bin/importadd
```

命令说明如下：

- 第 1 行，根据你的 GOPATH 不同，设置 GOPATH。
- 第 2 行，使用 go install 指令编译并安装 chapter08/code8-1 到 GOPATH 的 bin 目录下。
- 第 3 行，执行 GOPATH 的 bin 目录下的可执行文件 code8-1。

运行代码，输出结果如下：

```
3
```

导入的包之间可以通过添加空行来分组；通常将来自不同组织的包独自分组。包的导入顺序无关紧要，但是在每个分组中一般会根据字符串顺序排列。（gofmt 和 goimports 工具都可以将不同分组导入的包独立排序。）

```
01.  import (
02.      "fmt"
03.      "html/template"
04.      "os"
05.
06.      "golang.org/x/net/html"
07.      "golang.org/x/net/ipv4"
08.  )
```

导入包后自定义引用的包名

如果我们想同时导入两个有着名字相同的包，例如 math/rand 包和 crypto/rand 包，那么导入声明必须至少为一个同名包指定一个新的包名以避免冲突。这叫做导入包的重命名。

```
01.  import (
02.      "crypto/rand"
03.      mrand "math/rand" // 将名称替换为mrand避免冲突
04.  )
```

导入包的重命名只影响当前的源文件。其它的源文件如果导入了相同的包，可以用导入包原本默认的名字或重命名为另一个完全不同的名字。

导入包重命名是一个有用的特性，它不仅仅只是为了解决名字冲突。如果导入的一个包名很笨重，特别是在一些自动生成的代码中，这时候用一个简短名称会更方便。选择用简短名称重命名导入包时候最好统一，以避免包名混乱。选择另一个包名称还可以帮助避免和本地普通变量名产生冲突。例如，如果文件中已经有了一个名为 path 的变量，那么我们可以将"path"标准包重命名为 pathpkg。

每个导入声明语句都明确指定了当前包和被导入包之间的依赖关系。如果遇到包循环导入的情况，Go语言的构建工具将报告错误。

匿名导入包——只导入包但不使用包内类型和数值

如果只希望导入包，而不使用任何包内的结构和类型，也不调用包内的任何函数时，可以使用匿名导入包，格式如下：

```
01.  import (
02.      _ "path/to/package"
03.  )
```

其中，path/to/package 表示要导入的包名，下画线 `_` 表示匿名导入包。

匿名导入的包与其他方式导入包一样会让导入包编译到可执行文件中，同时，导入包也会触发 init() 函数调用。

包在程序启动前的初始化入口：init

在某些需求的设计上需要在程序启动时统一调用程序引用到的所有包的初始化函数，如果需要通过开发者手动调用这些初始化函数，那么这个过程可能会发生错误或者遗漏。我们希望在被引用的包内部，由包的编写者获得代码启动的通知，在程序启动时做一些自己包内代码的初始化工作。

例如，为了提高数学库计算三角函数的执行效率，可以在程序启动时，将三角函数的值提前在内存中建成索引表，外部程序通过查表的方式迅速获得三角函数的值。但是三角函数索引表的初始化函数的调用不希望由每一个外部使用三角函数的开发者调用，如果在三角函数的包内有一个机制可以告诉三角函数包程序何时启动，那么就可以解决初始化的问题。

Go 语言为以上问题提供了一个非常方便的特性：init() 函数。

init() 函数的特性如下：

- 每个源码可以使用 1 个 init() 函数。
- init() 函数会在程序执行前（main() 函数执行前）被自动调用。

- 调用顺序为 main() 中引用的包，以深度优先顺序初始化。

例如，假设有这样的包引用关系：main→A→B→C，那么这些包的 init() 函数调用顺序为：

```
C.init→B.init→A.init→main
```

说明：

- 同一个包中的多个 init() 函数的调用顺序不可预期。
- init() 函数不能被其他函数调用。

理解包导入后的init()函数初始化顺序

Go 语言包会从 main 包开始检查其引用的所有包，每个包也可能包含其他的包。Go 编译器由此构建出一个树状的包引用关系，再根据引用顺序决定编译顺序，依次编译这些包的代码。

在运行时，被最后导入的包会最先初始化并调用 init() 函数。

通过下面的代码理解包的初始化顺序。

代码8-3　包导入初始化顺序入口（.../chapter08/pkginit/main.go）

```
01.  package main
02.
03.  import "chapter08/code8-2/pkg1"
04.
05.  func main() {
06.
07.      pkg1.ExecPkg1()
08.  }
```

代码说明如下：

- 第 3 行，导入 pkg1 包。
- 第 7 行，调用 pkg1 包的 ExecPkg1() 函数。

代码8-4　包导入初始化顺序pkg1（.../chapter08/pkginit/pkg1/pkg1.go）

```
01.  package pkg1
02.
03.  import (
04.      "chapter08/code8-2/pkg2"
05.      "fmt"
06.  )
07.
08.  func ExecPkg1() {
09.
10.      fmt.Println("ExecPkg1")
11.
12.      pkg2.ExecPkg2()
13.  }
14.
15.  func init() {
16.      fmt.Println("pkg1 init")
17.  }
```

代码说明如下：

- 第 4 行，导入 pkg2 包。
- 第 8 行，声明 ExecPkg1() 函数。
- 第 12 行，调用 pkg2 包的 ExecPkg2() 函数。
- 第 15 行，在 pkg1 包初始化时，打印 pkg1 init。

代码8-5　包导入初始化顺序pkg2（.../chapter08/pkginit/pkg2/pkg2.go）

```
01.  package pkg2
```

```
02.
03.  import "fmt"
04.
05.  func ExecPkg2() {
06.      fmt.Println("ExecPkg2")
07.  }
08.
09.  func init() {
10.      fmt.Println("pkg2 init")
11.  }
```

代码说明如下：

- 第 5 行，声明 ExecPkg2() 函数。
- 第 10 行，在 pkg2 包初始化时，打印 pkg2 init。

执行代码，输出如下：

```
pkg2 init
pkg1 init
ExecPkg1
ExecPkg2
```

< 上一页 下一页 >

所有教程

C语言入门 C语言编译器 C语言项目案例 数据结构 C++ STL C++11 socket GCC GDB
Makefile OpenCV Qt教程 Unity 3D UE4 游戏引擎 Python Python并发编程 TensorFlow
Django NumPy Linux Shell Java教程 设计模式 Java Swing Servlet JSP教程 Struts2
Maven Spring Spring MVC Spring Boot Spring Cloud Hibernate Mybatis MySQL
MySQL函数 NoSQL Redis MongoDB HBase Go语言 C# MATLAB JavaScript Bootstrap
CSS教程 PHP 汇编语言 TCP/IP vi命令 Android教程 区块链 Docker 大数据 云计算
编程笔记 资源下载 VIP视频 关于我们

相关文章

Java多维数组
Shell变量测试与内容置换
UE4新建项目
UE4设置人物移动和人物视角
汇编语言置位和清除单个CPU标志位
常用32位编程调用规范简介

Hibernate avg方法：计算某一列的平均值
JavaScript中的几个重要概念
TiDB数据库的存储原理（非常详细）
C++ STL vector容器迭代器用法详解