# Go 1.16新特性：embed包及其使用详解

## embed是什么

embed是在Go 1.16中新加包。它通过***//go:embed***指令，可以在编译阶段将静态资源文件打包进编译好的程序中，并提供访问这些文件的能力。

## **为什么需要embed包**

* **部署过程更简单。**传统部署要么需要将静态资源与已编译程序打包在一起上传，或者使用docker和dockerfile自动化前者，这在精神上是很麻烦的。
* **确保程序的完整性。**在运行过程中损坏或丢失静态资源通常会影响程序的正常运行。
* **您可以独立控制程序所需的静态资源。**

最常见的方法（例如静态网站的后端程序）要求将程序连同其所依赖的html模板，css，js和图片以及静态资源的路径一起上传到生产服务器。必须正确配置Web服务器，以便用户访问它。

现在，我们将所有这些资源都嵌入到程序中。我们只需要部署一个二进制文件并为程序本身配置它们即可。部署过程已大大简化。

## embed的常用场景

以下列举一些静态资源文件需要被嵌入到程序中的常用场景：

* ***Go模板：****模板文件必须可用于二进制文件（模板文件需要对二进制文件可用）。 对于Web服务器二进制文件或那些通过提供init命令的CLI应用程序，这是一个相当常见的用例。 在没有嵌入的情况下，模板通常内联在代码中。例如示例qbec init的init命令：[https://qbec.io/userguide/tour/#initialize-a-new-qbec-app](https://link.zhihu.com/?target=https://qbec.io/userguide/tour/%23initialize-a-new-qbec-app" \t "https://zhuanlan.zhihu.com/p/_blank)*
* **静态web服务：**有时，静态文件（如index.html或其他HTML，JavaScript和CSS文件之类的静态文件）需要使用golang服务器二进制文件进行传输，以便用户可以运行服务器并访问这些文件。例如示例web server中嵌入静态资源文件：[https://github.com/gobuffalo/toodo/tree/master/assets](https://link.zhihu.com/?target=https://github.com/gobuffalo/toodo/tree/master/assets" \t "https://zhuanlan.zhihu.com/p/_blank)
* **数据库迁移：**另一个使用场景是通过嵌入文件被用于数据库迁移脚本。参考示例数据库迁移文件：[https://github.com/bigpanther/trober/tree/786dc471ea0d9b4a9e934d7e3c192de214f7c173/migrations](https://link.zhihu.com/?target=https://github.com/bigpanther/trober/tree/786dc471ea0d9b4a9e934d7e3c192de214f7c173/migrations" \t "https://zhuanlan.zhihu.com/p/_blank)

## embed的基本使用

embed包是golang 1.16中的新特性，所以，请确保你的golang环境已经升级到了1.16版本。 下面来一起看看embed的基本语法

### embed的基本语法

基本语法非常简单，首先导入***embed包，***然后使用指令***//go:embed***文件名 将对应的文件或目录结构导入到对应的变量上。 例如： 在当前目录下新建文件 version.txt，并输入内容 0.0.1

package main  
  
import (  
 \_ "embed"  
 "fmt"  
)  
  
//***go:embed*** version.txt  
var version string  
func main() {  
 fmt.Printf("version: %q\n", version)  
}

### embed的三种数据类型及使用

在embed中，可以将静态资源文件嵌入到三种类型的变量，分别为：**字符串、字节数组、embed.FS文件类型**

* 将文件内容嵌入到字符串变量中

package main  
  
import (  
 \_ "embed"  
 "fmt"  
)  
  
//***go:embed*** version.txt  
var version string  
  
func main() {  
 fmt.Printf("version %q\n", version)  
}

* 将文件内容嵌入到字节数组变量中

package main  
  
import (  
 \_ "embed"  
 "fmt"  
)  
  
//***go:embed*** version.txt  
var versionByte []byte  
  
func main() {  
 fmt.Printf("version %q\n", string(versionByte))  
}

* 将文件目录结构映射成embed.FS文件类型。embed.FS结构主要有3个对外方法

// Open opens the named file for reading and returns it as an fs.File.  
//  
// The returned file implements io.Seeker when the file is not a directory.  
func (f FS) Open(name string) (fs.File, error) {  
 file := f.lookup(name)  
 if file == nil {  
 return nil, &fs.PathError{Op: "open", Path: name, Err: fs.ErrNotExist}  
 }  
 if file.IsDir() {  
 return &openDir{file, f.readDir(name), 0}, nil  
 }  
 return &openFile{file, 0}, nil  
}  
  
// ReadDir reads and returns the entire named directory.  
func (f FS) ReadDir(name string) ([]fs.DirEntry, error) {  
 file, err := f.Open(name)  
 if err != nil {  
 return nil, err  
 }  
 dir, ok := file.(\*openDir)  
 if !ok {  
 return nil, &fs.PathError{Op: "read", Path: name, Err: errors.New("not a directory")}  
 }  
 list := make([]fs.DirEntry, len(dir.files))  
 for i := range list {  
 list[i] = &dir.files[i]  
 }  
 return list, nil  
}  
  
// ReadFile reads and returns the content of the named file.  
func (f FS) ReadFile(name string) ([]byte, error) {  
 file, err := f.Open(name)  
 if err != nil {  
 return nil, err  
 }  
 ofile, ok := file.(\*openFile)  
 if !ok {  
 return nil, &fs.PathError{Op: "read", Path: name, Err: errors.New("is a directory")}  
 }  
 return []byte(ofile.f.data), nil  
}

**以下搭建一个简单的静态文件web服务为例。**

首先，在项目根目录下建立如下静态资源目录结构

|-static  
|---js  
|------util.js  
|---img  
|------logo.jpg  
|---index.html

package main  
  
import (  
 "embed"  
 "io/fs"  
 "log"  
 "net/http"  
 "os"  
)  
  
func main() {  
 useOS := len(os.Args) > 1 && os.Args[1] == "live"  
 http.Handle("/", http.FileServer(getFileSystem(useOS)))  
 http.ListenAndServe(":8888", nil)  
}  
  
//***go:embed*** static  
var embededFiles embed.FS  
  
func getFileSystem(useOS bool) http.FileSystem {  
 if useOS {  
 log.Print("using live mode")  
 return http.FS(os.DirFS("static"))  
 }  
  
 log.Print("using embed mode")  
  
 fsys, err := fs.Sub(embededFiles, "static")  
 if err != nil {  
 panic(err)  
 }  
 return http.FS(fsys)  
}

以上代码，分别执行 go run . live 和 go run .，然后在浏览器中运行http://localhost:8888，默认显示static目录下的index.html文件内容。

当然，运行go run . live 和 go run . 的不同之处*在于****编译后的二进制程序文件在运行过程中是否依赖static目录中的静态文件资源。***

以下为验证步骤：

首先，使用编译到二进制文件的方式。

若文件内容改变，输出依然是改变前的内容，说明embed嵌入的文件内容在编译后不再依赖于原有静态文件了。

1、运行go run .

2、修改index.html文件内容为 Hello China

3、浏览器输入 http://localhost:8888 查看输出。输出内容为修改之前的Hello World

其次，使用普通的文件方式。若文件内容改变，输出的内容也改变，说明编译后依然依赖于原有静态文件。

1、go run . live

2、修改index.html文件内容为 delete

3、浏览器输入 http://localhost:8888 查看输出。输出修改后的内容：Hello China

## embed使用中注意事项

* 在使用//go:embed指令的文件都需要导入 embed包。 例如，以下例子 没有导入embed包，即不会正常运行

package main  
  
import (  
 "fmt"  
)  
  
//go:embed file.txt  
var s string  
  
func main() {  
 fmt.Print(s)  
}

* 其次，//go:embed指令只能用在包一级的变量中，不能用在函数或方法级别，像以下程序将会报错：

package main  
  
import (  
 \_ "embed"  
 "fmt"  
)  
  
func main() {  
 //go:embed file.txt  
 var s string  
 fmt.Print(s)  
}

* 第三，当包含目录时，它不会包含以“.”或“*“开头的文件。但是如果使用通配符，比如dir/\*，它将包含所有匹配的文件，即使它们以“."或"*"开头。 请记住，在您希望在Web服务器中嵌入文件但不允许用户查看所有文件的列表的情况下，包含Mac OS的.DS\_Store文件可能是一个安全问题。 出于安全原因，Go在嵌入时也不会包含符号链接或上一层目录。