链滴

Go1.16 初体验，怎样使用 //go:embed

作者：xhaoxiong
原文链接：https://ld246.com/article/1626138282555
来源网站：链滴

2月16日，Go1.16版本发布了。对于我们普通开发者来说，本次版本发布了一些有趣的特性，这里列了重要的几点：

- 新增了embed包，在编译时通过使用 //go:embed 指令可以进行嵌入文件的访问，即将文件嵌入二进制包中。
- 增加了对 macOS ARM64 的支持（Apple silicon）。
- 默认开启 Go modules。
- 修复了一些bug和改进一些问题，如构建速度提升25%，内存使用量降低15%。
- io/util 包被弃用，所有方法被移至io 和os 包。

具体详细的发布日志移步：go1.16，本篇文章关注的是如何使用：//go:embed。

embed功能说明

embed能帮我们做什么？一句话概括就是将静态资源文件嵌入到编译的二进制文件中。

这样做有什么优势？个人认为比较重要是保证一个应用的完整性。比如：

1. 比如一个Web应用，包含了很多image和html，一般情况下我们需要将所有的文件和编译好的二制文件拷贝到同一机器。如果是分布式应用还会带来更多的拷贝过程。当然如果使用如docker容器镜方式，是会简化拷贝过程，但也会增加一些负担，如：打包过程。
2. 比如一个App应用，本身会携带很多如音频、图片小文件。一般情况下在安装过程中我们需要将许多多的小文件进行拷贝，我们知道磁盘I/O瓶颈比较大的，安装时间长会给用户带来不好的体验。
3. 比如一个游戏应用。
4. 比如一个WebAssembly应用等等。

当然上面举的例子只是从一些方面来考虑，具体的打包部署方式需要综合考虑多个因素，如当前公司自动化运维体系。

embed能帮我们保证一个应用的整体性和完整性，我觉得对于强迫症的开发者来说一定是个福利，哈。下面来看看embed的使用方法。

embed基础用法

通过 官方文档 我们知道embed嵌入的三种方式：string、bytes和FS（File Systems）。

//go:embed 基本用法是：

```
package main

import "embed"

//go:embed hello.txt
var s string

//go:embed hello.txt
var b []byte

//go:embed hello.txt
```

```
//go:embed assets
var f embed.FS

func main() {
  print(s)

  print(string(b))

  data, _ := f.ReadFile("hello.txt")
  print(string(data))
}
```

1、导入 embed 包，如果没有使用 embed.FS 需要显示的导入：

```
import _ "embed"
```

2、匹配文件 //go:embed <匹配模式> <匹配模式>...，匹配模式符合 path.Match 方式。

（1）匹配模式是相对位置，如：

```
├── assets
│   ├── .gitkeep
│   ├── _home.html
│   └── index.html
├── hello.txt
└── main.go
```

匹配 index.html 则使用 //go:embed assets/index.html 即可，不能使用 . 和 ..（如 ./hello.txt）。

（2）可以匹配多个，以空格隔开，如 //go:embed hello.txt assets/index.html。也可以重复，避免配长度过长：

```
//go:embed hello.txt
//go:embed assets/index.html
var f embed.FS
```

（3）[]byte 和 string 只能匹配单个文件。如果文件名称有空格可使用双引号 " 或者反引号 ```。 （4 如果 //go:embed assets匹配的是一个目录，那么该目录中所有文件都将递归的嵌入，除了以 .或 * 头的文件。 （5）匹配目录中的所有内容，使用统配 *，包括以 .和 * 开头的文件。

3、匹配的变量只能是全局变量。

embed进阶用法

Go1.16 为了对 embed 的支持也添加了一个新包 io/fs。两者结合起来可以像之前操作普通文件一样。

常规文件操作

如通过 embed 进行常规的文件目录读取，文件递归遍历等：

```
//go:embed hello.txt
//go:embed hello.txt assets/*
var f embed.FS
```

...

```
entries, err := f.ReadDir(".")
if err != nil {
  panic(err)
}
for _, entry := range entries {
  info, err := entry.Info()
  if err != nil {
    panic(err)
  }
  fmt.Println(info.Name(), info.Size(), info.IsDir())
}
```

Web文件系统

通过原生go http服务，我们将静态资源文件嵌入到二进制中，做静态文件服务器：

```
package main

import (
  "embed"
  "net/http"
)

//go:embed hello.txt assets/*
var f embed.FS

func main() {
  http.Handle("/static/", http.StripPrefix("/static/", http.FileServer(http.FS(f))))

  http.ListenAndServe(":8080", nil)
}
```

通过常见的Web服务框架，提供文件的访问：

```
package main

import (
  "embed"
  "net/http"

  "github.com/gin-gonic/gin"
)

//go:embed hello.txt assets/*
var f embed.FS

func main() {
  e := gin.Default()

  e.StaticFS("/static/", http.FS(f))
  e.Run(":8080")
}
```

其它web框架各自可以试试。

模版操作

通过 embed 方式嵌入模版，渲染模版：

```
├── main.go
└── tmpl
    ├── en.tmpl
    └── zh.tmpl
```

```
package main

import (
    "embed"
    "fmt"
    "html/template"
    "net/http"
)

//go:embed tmpl/*.tmpl
var f embed.FS

func main() {
  t, err := template.ParseFS(f, "tmpl/*.tmpl")
  if err != nil {
    panic(err)
  }

  // /hello?lang=xx.tmpl
  http.HandleFunc("/hello", func(w http.ResponseWriter, r *http.Request) {
    r.ParseForm()

    t.ExecuteTemplate(w, r.FormValue("lang"), nil)
  })

  http.ListenAndServe(":8080", nil)
}
```

版本嵌入

通常我们需要将我们的版本打包到二进制文件中，以便确定我们的版本信息。embed 之前我们可以取通过 -ldflags 的方法将版本动态的赋值到变量。现在，我们可以通过 embed 方式赋值啦。

```
# version_dev.go
// +build !prod

package main

var version string = "dev"

# version_prod.go
// +build prod
```

原文链接：Go1.16 初体验，怎样使用 //go:embed

```
package main

import (
  _ "embed"
)

//go:embed version.txt
var version string
```

执行命令：

```
$ go run .
Version "dev"

$ go run -tags prod .
Version "0.0.1"
```

Deepzz

Permalink to Go1.16初体验，怎样使用 //go:embed