下载安装

2019年6月15日 星期六　　下午7:23

1. golang.org/x

方法一： GO111MODULE:

很多库都依赖golang.org/x，go get在下载某些库时会自动下载这些依赖性的库，但由于众所周知的原因，国内访问不了。因此需要配置代理：

A global proxy for go modules: https://goproxy.io/

```
# Enable the go modules feature
export GO111MODULE=on
# Set the GOPROXY environment variable
export GOPROXY=https://goproxy.io
```

注：GO111MODULE 新增了go modules，是一个全新的包管理工具。

方法二：如果想用之前的包管理系统:

需要先在src中创建golang.org/x/目录，再到https://github.com/golang/中下载golang.org中的包，并把包放置在golang.org/x/目录下。

db

2019年7月5日 星期五　　上午10:28

gorm

2019年7月5日 星期五　　上午10:29

https://juejin.im/post/5ce2a5f3e51d455d86719f77 GORM模型定义，字段映射等

Go语言的标准库sql/database包操作数据库的过程，虽然使用sql/database包操作数据也是挺方便的，但是需要自己写每一条SQL语句，因此我们可能会自己再度进行封装，以便更好地使用，而使用现有Go语言开源ORM框架则是代替自己封装的一个更好的方式。

ORM，即对象关系映射(Object Relational Mapping)，可以简单理解为将关系型数据库中的数据表映射为编程语言中的具体的数据类型(如struct)，而GORM库就是一个使用Go语言实现的且功能非常完善易使用的ORM框架。

安装和使用：

go get -u github.com/jinzhu/gorm

import "github.com/jinzhu/gorm"

GORM框架支持MySQL,SQL Server,Sqlite3,PostgreSQL四种数据库驱动，如果我们要连接这些数据库，则需要导入不同的驱动包及定义不同格式的DSN(Data Source Name)。

MySQL

1. 导入

import _ "github.com/jinzhu/gorm/dialects/mysql"

//或者//import _ "github.com/go-sql-driver/mysql"

复制代码

2. DSN

//user指用户名，password指密码,dbname指数据库名

"user:password@(ip:port)/dbname?charset=utf8&parseTime=True&loc=Local"

eg: source_name="root:1d3B9qRT@f@(10.236.158.96:12339)/zhuji_logic?charset=utf8mb4&parseTime=true&loc=Asia%2FShanghai"

sql.Open函数实际上是返回一个连接池对象，不是单个连接。在open的时候并没有去连接数据库，只有在执行query、exce方法的时候才会去实际连接数据库。在一个应用中同样的库连接只需要保存一个sql.Open之后的db对象就可以了，不需要多次open。

连接池的实现关键在于SetMaxOpenConns和SetMaxIdleConns，其中：
SetMaxOpenConns用于设置最大打开的连接数，默认值为0表示不限制。
SetMaxIdleConns用于设置闲置的连接数。

Find & Scan

2019年7月7日 星期日　　下午2:12

http://gorm.io/docs/query.html

Find 可以加条件；Scan不可以。
Scan results into another struct.

Pb

2019年7月15日 星期一　　下午3:02

一、说明

https://segmentfault.com/a/1190000009277748

在go中使用protobuf，有两个可选用的包goprotobuf（go官方出品）和 gogoprotobuf，gogoprotobuf完全兼容google protobuf，它生成的代码质量和编解码性能均比goprotobuf高一些。

官方：https://github.com/protocolbuffers/protobuf

gogoprotobuf：https://github.com/gogo/protobuf/

There are several ways to use gogoprotobuf, but for all you need to install go and protoc. After that you can choose:

- Speed
  protoc --gofast_out=. myproto.proto

- More Speed and more generated code
- Most Speed and most customization

二、安装goprotobuf

https://developers.google.com/protocol-buffers/docs/gotutorial

1.安装protocol compiler

brew install protobuf //默认安装最新版本

protoc --version

2.Run the following command to install the Go protocol buffers plugin:

```
go get -u github.com/golang/protobuf/protoc-gen-go
```

protoc需要使用protoc-gen-go，因此需要把protoc-gen-go放到PTAH路径下。

3.运行compiler

```
protoc -I=$SRC_DIR --go_out=$DST_DIR $SRC_DIR/addressbook.proto
```

the source directory (where your application's source code lives – the current directory is used if you don't provide a value),
the destination directory (where you want the generated code to go; often the same as `$SRC_DIR`),
the path to your `.proto`

protoc --go_out=. *.proto

4.note

- proto3中删除了required和optional
  https://stackoverflow.com/questions/31801257/why-required-and-optional-is-removed-in-protocol-buffers-3

Config

2019年7月30日 星期二　　上午10:35

github.com/spf13/viper

2019年7月30日 星期二　　上午10:35

```
go get -u github.com/spf13/viper
```

命令行

2019年7月18日 星期四　　下午3:43

go标准库flag包

2019年7月22日 星期一　　上午9:54

alecthomas/kingpin

2019年7月18日 星期四　　下午3:44

https://github.com/alecthomas/kingpin
2.5k

github.com/spf13/cobra

2019年7月30日 星期二　　上午10:32

Cobra 库提供简单接口创建类似git和go的命令行接口。
Cobra 也是一个应用程序，用来生成应用框架，开发以 Cobra 为基础的应用。

相关概念：

cobra基于commands, arguments, flags的结构。

```
$ go get -u github.com/spf13/cobra/cobra
```

Web

2019年8月9日 星期五　　下午4:25

gin

2019年8月9日 星期五　　下午4:26

1.post请求也可以把参数放在url中带到后端

获取post请求的参数：

- gin.Context.Query //从url获取

```
// Query returns the keyed url query value if it exists,
// otherwise it returns an empty string `("")`.
// It is shortcut for `c.Request.URL.Query().Get(key)`
//     GET /path?id=1234&name=Manu&value=
//            c.Query("id") == "1234"
//            c.Query("name") == "Manu"
//            c.Query("value") == ""
//            c.Query("wtf") == ""
```

- gin.Context.PostForm() //从body获取

```
// PostForm returns the specified key from a POST urlencoded form or multipart form
// when it exists, otherwise it returns an empty string `("")`.
```

2.多个handler，每个handler按先后顺序执行

```
screen.POST("/debug", Auth, LoginAllow, CheckFrequent, sh.Debug)
```

//可以在前面的handler中call gin.context.Abort() 来跳过后续的所有handler。//一般用于鉴权。

net/http

2019年8月9日 星期五　　下午4:26

https://studygolang.com/articles/9467

一切的基础: ServeMux(多路复用器，Multiplexor) 和 Handler(处理器)

1.ServeMux 本质上是一个HTTP请求路由器（或者叫多路复用器），它把收到的请求与一组预先定义的 URL 路径列表做对比，然后在匹配到路径的时候调用关联的Handler。

2.Handler负责输出HTTP响应的头和正文。任何满足了http.Handler接口的对象都可作为一个处理器。对象只要实现该方法即可：ServeHTTP(http.ResponseWriter, *http.Request)。
Http包自带了几个函数作为常用的处理器，比如FileServer，NotFoundHandler 和 RedirectHandler。

3.自定义处理器 custom handlers

4.将函数作为处理器

5.更便利的DefaultServerMux
为了简便起见，net/http包提供了一个全局的ServeMux实例DefaultServeMux，以及包级别的注册函数http.Handle和http.HandleFunc。要让DefaultServeMux作为服务器的主处理程序，只需将nil传给ListenAndServer。

DefaultServerMux 并没有什么的特殊的地方，DefaultServerMux 就是我们之前用到的 ServerMux，只是它随着 net/http 包初始化的时候被自动初始化了而已。
net/http 包提供了一组快捷方式来配合 DefaultServeMux：http.Handle 和 http.HandleFunc。这些函数与我们之前看过的类似的名称的函数功能一样，唯一的不同是他们将处理器注册到 DefaultServerMux，而之前我们是注册到自己创建的 ServeMux。

---
http请求时，如果 SetHeader("Content-Type","application/json")，则请求body直接转换为json即可；
如果 SetHeader("Content-Type", "application/x-www-form-urlencoded")，则请求body格式为`name=vanyar&age=27`，需要对请求体（或部分请求体）进行urlencoded（*request.URI.RawQuery*）转换。
bodyStr := "c=auto_write&plt_id=" + PLT_ID + "&req_data=" + *url.QueryEscape(string(autoWriterBuffer))* + "&token=" + token + "&uid=" + req.AppId
---

总结：

1.http.Handle

```
// Handle registers the handler for the given pattern in the DefaultServeMux.
func Handle(pattern string, handler Handler) { DefaultServeMux.Handle(pattern, handler) }
```

2.http.Handler

```
type Handler interface {
	ServeHTTP(ResponseWriter, *Request)
}
```

3.http.HandleFunc

```
// HandleFunc registers the handler function for the given pattern in the DefaultServeMux.
func HandleFunc(pattern string, handler func(ResponseWriter, *Request)) {
	DefaultServeMux.HandleFunc(pattern, handler)
```

```
}
```

4.http.HandlerFunc

```
// The HandlerFunc type is an adapter to allow the use of ordinary functions as HTTP handlers.
// If f is a function with the appropriate signature, HandlerFunc(f) is a Handler that calls f.
type HandlerFunc func(ResponseWriter, *Request)
```

5.http.NewServeMux

```
// NewServeMux allocates and returns a new ServeMux.
func NewServeMux() *ServeMux { return new(ServeMux) }
```

6.http.ListenAndServe

```
// ListenAndServe listens on the TCP network address addr and then calls Serve with handler to handle requests on incoming connections.
// Accepted connections are configured to enable TCP keep-alives.
// The handler is typically nil, in which case the DefaultServeMux is used.
// ListenAndServe always returns a non-nil error.
func ListenAndServe(addr string, handler Handler) error {
	server := &Server{Addr: addr, Handler: handler}
	return server.ListenAndServe()
}
```

gopkg.in/resty.v1

2019年9月9日 星期一　　下午2:36

Package resty provides Simple HTTP and REST client library for Go.

TLS

2019年12月24日 星期二　　下午8:16

https://zhuanlan.zhihu.com/p/26684081

sql

2019年7月2日 星期二　　下午5:41

NULL

2019年9月18日 星期三　　上午10:11

https://github.com/golang/go/wiki/SQLInterface

Dealing with NULL

If a database column is nullable, one of the types supporting null values should be passed to Scan.

For example, if the name column in the names table is nullable:

```
var name NullString
err := db.QueryRowContext(ctx, "SELECT name FROM names WHERE id = $1",
id).Scan(&name)
...
if name.Valid {
        // use name.String
} else {
        // value is NULL
}
```

Only NullBool, NullFloat64, NullInt64 and NullString are implemented in database/sql. Implementations of database-specific null types are left to the database driver.

时间问题

2019年9月18日 星期三　　上午10:10

https://studygolang.com/articles/9880

使用mysql的时间字段遇到如下两个问题

1.使用go-sql-driver来连接mysql数据库，获取的时区默认是UTC +0的，与本地的东八区是有区别，在业务处理中会出现问题

2.获取mysql中的日期，是string类型，需要在代码中用time.Parse进行转化

`time.Time`

解决方案：

在连接的dsn中，添加`parseTime=true` 和`loc=Local`，此处的`local`可以换为具体的时区`(Asia/Shanghai)`

"zhuji_logic":"root:m4EU3z!X2z@(10.59.216.87:14578)/zhuji_logic?charset=utf8mb4&parseTime=true&loc=Asia%2FShanghai"

批量处理

2019年9月18日 星期三　　上午10:17

mysql批量处理：
https://blog.yumaojun.net/2017/06/19/mysql-performance-for-bulk-action/

go module

2019年12月5日 星期四　　下午8:12

https://segmentfault.com/a/1190000016703769
官网：https://github.com/golang/go/wiki/Modules

⚠ macos 命令行下直接执行 go build / go run 不会下载依赖，而是会报错，找不到包。

1.常见的包管理工具

- govendor
- dep
- glide
- godep

这些包管理工具都是基于GOPATH或者vendor目录，并不能很好的解决不同版本依赖问题。Modules是在GOPATH之外一套新的包管理方式。

优势：

- 自动下载依赖包
- 不需要将代码再放入$GOPATH/src
- 所有来自第三方的包都会指定版本（使用dep是无法指定第三方包的版本的）
- 对于已经转移的包，可以用replace在go.mod文件中替换，不需要修改代码

Go Modules 是 Go 语言的一种依赖管理方式，该 feature 是在 Go 1.11 版本中出现的，由于最近在做的项目中，团队都开始使用 go module 来替代以前的 Godep，Kubernetes 也从 v1.15 开始采用 go module 来进行包管理，所以有必要了解一下 go module。go module 相比于原来的 Godep，go module 在打包、编译等多个环节上有着明显的速度优势，并且能够在任意操作系统上方便的复现依赖包，更重要的是 go module 本身的设计使得自身被其他项目引用变得更加容易，这也是 Kubernetes 项目向框架化演进的又一个重要体现。

2.如何激活go modules

首先要把go升级到1.11。
升级后，可以设置通过一个环境变量GO111MODULE来激活modules：

- GO111MODULE=off，go命令行将不会支持module功能，寻找依赖包的方式将会沿用旧版本那种通过vendor目录或者GOPATH模式来查找。
- GO111MODULE=on，go命令行会使用modules，而一点也不会去GOPATH目录下查找。
- GO111MODULE=auto，默认值，go命令行将会根据当前目录来决定是否启用module功能。这种情况下可以分为两种情形：当前目录在GOPATH/src之外且该目录包含go.mod文件，或者当前文件在包含go.mod文件的目录下面。

GO111MODULE=auto enables module-mode if any go.mod is found, even inside GOPATH. (Prior to Go 1.13, GO111MODULE=auto would never enable module-mode inside GOPATH/src).

当module功能启用时，GOPATH在项目构建过程中不再担当import的角色，但它仍然存储下载的依赖包，具体位置在$GOPATH/pkg/mod。

3.初始化mod

go help mod

Go mod provides access to operations on modules.

Note that support for modules is built into all the go commands,
not just 'go mod'. For example, day-to-day adding, removing, upgrading,
and downgrading of dependencies should be done using 'go get'.
See 'go help modules' for an overview of module functionality.

Usage:

```
go mod <command> [arguments]
```

The commands are:

```
download    download modules to local cache
edit        edit go.mod from tools or scripts
graph       print module requirement graph
init        initialize new module in current directory
tidy        add missing and remove unused modules
vendor      make vendored copy of dependencies
verify      verify dependencies have expected content
why         explain why packages or modules are needed
```

Use "go help mod <command>" for more information about a command.

- mkdir helloworld && cd helloworld

- 编写main函数
  Eg:

```
package main

import (
        "net/http"
        "github.com/labstack/echo"
)

func main() {
        e := echo.New()
        e.GET("/", func(c echo.Context) error {
                return c.String(http.StatusOK, "Hello, World!")
        })
        e.Logger.Fatal(e.Start(":1323"))
}
```

- go mod init helloworld //初始化mod,系统生成了一个go.mod的文件

  go.mod 文件一旦创建后，它的内容将会被 go toolchain 全面掌控。go toolchain 会在各类命令执行时，比如 go get、go build、go mod 等修改和维护 go.mod 文件。

- go build //下载依赖并生成go.sum文件

  go.sum不是一个锁文件，是一个模块版本内容的校验值，用来验证当前缓存的模块。go.sum包含了直接依赖和间接依赖的包的信息，比go.mod要多一些。

4.go.mod

有四种指令：module，require，exclude，replace。

module：模块名称

require：依赖包列表以及版本

exclude：禁止依赖包列表（仅在当前模块为主模块时生效）

replace：替换依赖包列表 （仅在当前模块为主模块时生效）

由于某些已知的原因，并不是所有的package都能成功下载，比如：golang.org下的包。modules 可以通过在 go.mod 文件中使用 replace 指令替换成github上对应的库，比如：

```
replace (
	golang.org/x/crypto v0.0.0-20190313024323-a1f597ede03a => github.com/golang/crypto
	v0.0.0-20190313024323-a1f597ede03a
)
```

或者

```
replace golang.org/x/crypto v0.0.0-20190313024323-a1f597ede03a => github.com/golang/crypto
v0.0.0-20190313024323-a1f597ede03a
```

5.其他命令

go mod tidy //拉取缺少的模块，移除不用的模块。

go mod download //下载依赖包

go mod graph //打印模块依赖图

go mod vendor //将依赖复制到vendor下

go mod verify //校验依赖

go mod why //解释为什么需要依赖

go list -m -json all //依赖详情

6.vendor模式

go mod是不推荐使用vendor目录的，而是直接使用source或cache中的包。

module mode下默认忽略vendor目录。通过flag-mod=vendor设置vendor模式，依赖只从顶层的vendor中查找。可以通过环境变量GOFLAGS=-mod=vendor来设置flag。

How do I use vendoring with modules? Is vendoring going away?

The initial series of vgo blog posts did propose dropping vendoring entirely, but feedback from the community resulted in retaining support for vendoring.

In brief, to use vendoring with modules:

- *go mod vendor* resets the main module's vendor directory to include all packages needed to build and test all of the module's packages based on the state of the go.mod files and Go source code.
- By default, go commands like ***go build*** ignore the vendor directory when in module mode.
- The ***-mod=vendor*** flag (e.g., go build -mod=vendor) instructs the go commands to use the main module's top-level vendor directory to satisfy dependencies. The go commands in this mode therefore ignore the dependency descriptions in go.mod and assume that the vendor directory holds the correct copies of dependencies. Note that only the main module's top-level vendor directory is used; vendor directories in other locations are still ignored.
- Some people will want to routinely opt-in to vendoring by setting a GOFLAGS=-mod=vendor environment variable.

Older versions of Go such as 1.10 understand how to consume a vendor directory created by go mod vendor, as do Go 1.11 and 1.12+ when module mode is disabled. Therefore, vendoring is one way for a module to provide dependencies to older versions of Go that do not fully understand modules, as well as to consumers that have not enabled modules themselves.

If you are considering using vendoring, it is worthwhile to read the "Modules and vendoring" and "Make vendored copy of dependencies" sections of the tip documentation.

Modules and vendoring

When using modules, the go command completely ignores vendor directories.
By default, the go command satisfies dependencies by downloading modules from their sources and using those downloaded copies (after verification, as described in the previous section). To allow interoperation with older versions of Go, or to ensure that all files used for a build are stored together in a single file tree, 'go mod vendor' creates a directory named vendor in the root directory of the main module and stores there all the packages from dependency modules that are needed to support builds and tests of packages in the main module.
To build using the main module's top-level vendor directory to satisfy dependencies (disabling use of the usual network sources and local caches), use 'go build -mod=vendor'. Note that only the main module's top-level vendor directory is used; vendor directories in other locations are still ignored.

go micro

2019年11月29日 星期五　　上午10:32

https://micro.mu/docs/cn/go-micro.html
https://www.kancloud.cn/linimbus/go-micro/529030