开发

驱动

连接数据库是典型的CS编程，服务器端被动等待客户端建立TCP连接，并在此连接上进行特定的应用层协议。但一般用户并不需要了解这些细节，这些都被打包到了驱动库当中，只需要简单的调用打开就可以指定协议连接到指定的数据库。

Go官方不可能也不熟悉不同数据库的协议，所以不可能提供针对不同数据库的驱动程序，往往由各数据库官方或第三方给出不同开发语言的驱动库。但是，为了Go语言可以提前定义操作一个数据库的所有行为（接口）和数据（结构体）的规范，这些定义在database/sql下。

MySQL驱动

- https://github.com/go-sql-driver/mysql 支持 database/sql，**推荐**
- https://github.com/ziutek/mymysql 支持 database/sql，支持自定义接口
- https://github.com/Philio/GoMySQL 不支持 database/sql，支持自定义接口

安装mysql的Go驱动

```
$ go get -u github.com/go-sql-driver/mysql
```

导入

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

注册驱动

```
// github.com/go-sql-driver/mysql/mysql/driver.go 代码中有注册驱动
func init() { // 83行
    sql.Register("mysql", &MySQLDriver{})
}
```

连接

DSN例子 https://github.com/go-sql-driver/mysql#examples

`[username[:password]@][protocol[(address)]]/dbname[?param1=value1&...&paramN=valueN]`

```
connstr := "wayne:wayne@tcp(127.0.0.1:3306)/test"
connstr := "wayne:wayne@tcp/test"
connstr := "wayne:wayne@/test"
```

```
package main
```

```
2
3  import (
4      "database/sql"
5      "fmt"
6      "log"
7      "time"
8
9      _ "github.com/go-sql-driver/mysql" // 1 驱动安装和导入
10 )
11
12 var db *sql.DB
13
14 func init() {
15     // 2 连接数据库
16     var err error
17     db, err = sql.Open("mysql", "wayne:wayne@tcp(localhost:3306)/test") //
   不要使用:=
18     if err != nil {
19         log.Fatal(err)
20     }
21
22     // 参看README.md https://github.com/go-sql-driver/mysql#usage
23     db.SetConnMaxLifetime(time.Second * 30) // 超时时间
24     db.SetMaxOpenConns(0)                   // 设置最大连接数，默认为0表示不限制
25     db.SetMaxIdleConns(10)                  // 设置空闲连接数
26 }
```

db类型是*sql.DB，是一个操作数据库的句柄，底层是一个多协程安全的连接池。

操作

```
1  package main
2
3  import (
4      "database/sql"
5      "fmt"
6      "log"
7      "time"
8
9      _ "github.com/go-sql-driver/mysql" // 1 驱动安装和导入
10 )
11
12 var db *sql.DB
13
14 func init() {
15     // 2 连接数据库
16     var err error
17     db, err = sql.Open("mysql", "wayne:wayne@tcp(localhost:3306)/test") //
   不要使用:=
18     if err != nil {
19         log.Fatal(err)
20     }
21
```

```go
    // 参看README.md
    db.SetConnMaxLifetime(time.Second * 30) // 超时时间
    db.SetMaxOpenConns(0)                   // 设置最大连接数，默认为0表示不限制
    db.SetMaxIdleConns(10)                  // 设置空闲连接数
}

type Emp struct { // 和字段对应的变量或结构体定义，最好和数据库中字段顺序对应
    emp_no     int
    birth_date string
    first_name string
    last_name  string
    gender     int16
    hire_date  string
}

func main() {
    // 3 操作
    // 预编译
    stmt, err := db.Prepare("select * from employees where emp_no > ?")
    if err != nil {
        log.Fatal(err)
    }
    defer stmt.Close()

    // 批量查询
    rows, err := stmt.Query(10018)
    if err != nil {
        log.Fatal(err)
    }
    defer rows.Close()

    emp := Emp{}
    for rows.Next() {
        err = rows.Scan(&emp.emp_no, &emp.birth_date, &emp.first_name,
            &emp.last_name, &emp.gender, &emp.hire_date) // 取字段
        if err != nil {
            log.Fatal(err)
        }
        fmt.Println(emp)
        fmt.Printf("%T %[1]v\n", emp.birth_date)
        t, err := time.Parse("2006-01-02", emp.birth_date) // 时间解析
        if err != nil {
            log.Fatal(err)
        }
        fmt.Printf("%T %[1]v\n", t)
    }
    fmt.Println("~~~~~~~~~~~~~~~~~~~~~~~~~~~")

    // 单行查询
    row := db.QueryRow("select * from employees where emp_no = ?", 10010)
    err = row.Scan(&emp.emp_no, &emp.birth_date, &emp.first_name,
        &emp.last_name, &emp.gender, &emp.hire_date) // 取字段
    if err != nil {
        log.Fatal(err)
    }
```

```
77      fmt.Println(emp)
78  }
```

- 驱动安装和导入，例如`import _ "github.com/go-sql-driver/mysql"`
- 连接数据库并返回数据库操作句柄，例如`sql.Open("mysql", "wayne:wayne@tcp(localhost:3306)/test")`
- 使用db提供的接口函数
- 使用db.Prepare预编译并使用参数化查询
    - 对预编译的SQL语句进行缓存，省去了每次解析优化该SQL语句的过程
    - 防止注入攻击
    - 使用返回的sql.Stmt操作数据库