Golang 的 mysql 驱动的解析 （连接过程）

作者：lucianolixin
原文链接：https://ld246.com/article/1573025534337
来源网站：链滴

前言：go中import用于导入包

进入正题之前先介绍一个小知识点：

比如：

```
import  "fmt"
```

在代码中就可以使用fmt包中的方法，如：

```
fmt.Println("hello world")
```

1、使用点操作引入包时，可以省略包前缀：

```
import . "fmt"
```

注意上面 fmt前多了 . 字符。代码中使用时：

```
Println("hello world")
```

前缀fmt就不需要了。

2、别名操作可以给包起个小名儿。如：

```
import  f "fmt"

f.Println("hello world")
```

3、_操作

由于go在引入包时调用包的init方法。所以使用_操作，主要是为了使用包的init函数，一般用在数据方面的包中：

```
import (
"database/sql"
_ "github.com/ziutek/mymysql/godrv"
)
```

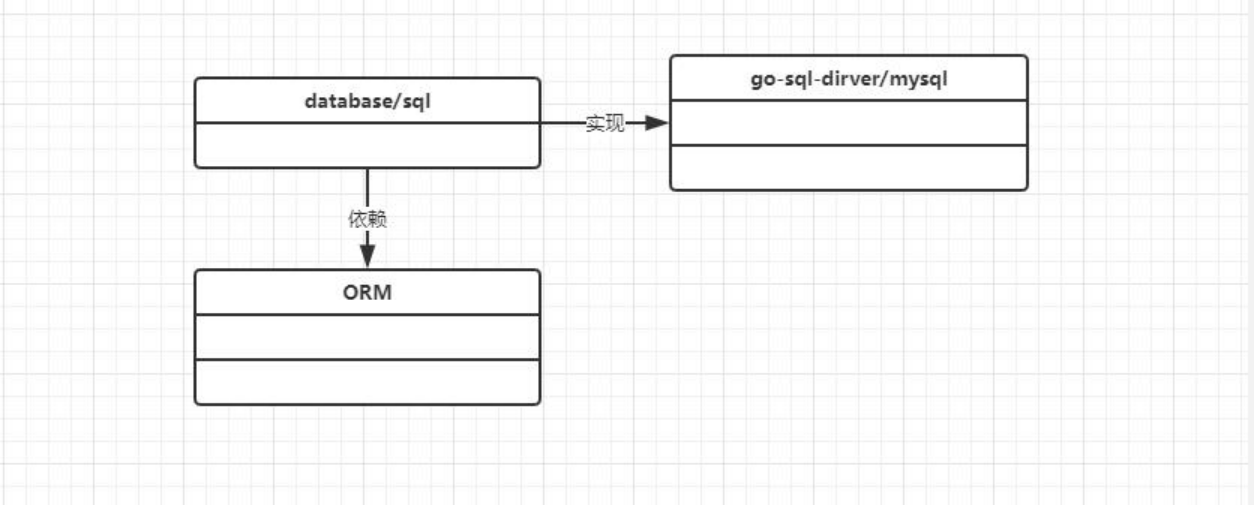
这个可以避免go编译时提示引入了包但未使用。

MYSQL驱动的工作原理

SQL驱动接口

Golang的官方标准库提供了database/sql，sql包提供了保证SQL或类SQL数据库的泛用接口。使用sql包时必须注入（至少）一个数据库驱动。

```
Goals of the sql and sql/driver packages:
Provide a generic database API for a variety of SQL or SQL-like
databases.  There currently exist Go libraries for SQLite, MySQL,
and Postgres, but all with a very different feel, and often
a non-Go-like feel.
```



1. sql/driver的目的是提供一个通用的数据库抽象接口，现在已经存在SQLite，MySQL等，但是他们设计风格完全不同。

2. 考虑到通用SQL的可移植的情况，而特定的SQL特性，和扩展应该被应用程序单独实现。

3. 使用sql/dirver包，用户不需要考虑并发和线程安全的问题，通过 *sql.DB 可以在不同的goroutine之间共享，而不需要额外的同步。

MySQL驱动

go-sql-dirver/mysql是一个MySQL数据库的驱动， databases/sql中定义的泛化接口的实现。

它需要自行安装：

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

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

db, err := sql.Open("mysql", "user:password@/dbname")
```

导入go-sql-dirver/mysql包，这里用到前言中的最后一种导入包的情况，目的是运行mysql包中init数，把mysql驱动注册到database/sql中，注册的名称是“mysql”，对象是一个实现了Open方法结构体MysqlSQLDirver。

```
// in mysql package
func init() {

    sql.Register("mysql", &MySQLDriver{})

}

// in sql package
func Register(name string, driver driver.Driver) {

    driversMu.Lock()
    defer driversMu.Unlock()
    if driver == nil {
        panic("sql: Register driver is nil")
    }
    if _, dup := drivers[name]; dup {
        panic("sql: Register called twice for driver " + name)
    }
    drivers[name] = driver
}
```

注册函数用到了Sync包中的同步工具。

同步的用途有两个，一个是避免多个线程在同一时刻操作同一个数据块，另一个是协调多个线程，以免它们在同一时刻执行同一个代码块

在 Go 语言中，可供我们选择的同步工具并不少。其中，最重要且最常用的同步工具当属互斥量（mutual exclusion，简称 mutex。sync包中的Mutex就是与其对应的类型，该类型的值可以被称为互斥或者互斥锁。

每当线程或者goroutine进去临界区，抢占资源的时候就要进行lock操作，当离开临界区，unlock操。解锁最好用defer关键字。防止程序崩溃无法解锁的情况导致后续goroutine的阻塞。

```
// Open new Connection.
// See https://github.com/go-sql-driver/mysql#dsn-data-source-name for how
// the DSN string is formated
```

```
func (d MySQLDriver) Open(dsn string) (driver.Conn, error) {
    var err error

    // New mysqlConn
    mc := &mysqlConn{
        maxAllowedPacket: maxPacketSize,
        maxWriteSize:     maxPacketSize - 1,
    }
    mc.cfg, err = ParseDSN(dsn)
    if err != nil {
        return nil, err
    }
    mc.parseTime = mc.cfg.ParseTime
    mc.strict = mc.cfg.Strict

    // Connect to Server
    if dial, ok := dials[mc.cfg.Net]; ok {
        mc.netConn, err = dial(mc.cfg.Addr)
    } else {
        nd := net.Dialer{Timeout: mc.cfg.Timeout}
        mc.netConn, err = nd.Dial(mc.cfg.Net, mc.cfg.Addr)
    }
    if err != nil {
        return nil, err
    }

    // Enable TCP Keepalives on TCP connections
    if tc, ok := mc.netConn.(*net.TCPConn); ok {
        if err := tc.SetKeepAlive(true); err != nil {
            // Don't send COM_QUIT before handshake.
            mc.netConn.Close()
            mc.netConn = nil
            return nil, err
        }
    }

    mc.buf = newBuffer(mc.netConn)

    // Set I/O timeouts
    mc.buf.timeout = mc.cfg.ReadTimeout
    mc.writeTimeout = mc.cfg.WriteTimeout

    // Reading Handshake Initialization Packet
    cipher, err := mc.readInitPacket()
    if err != nil {
        mc.cleanup()
        return nil, err
    }

    // Send Client Authentication Packet
    if err = mc.writeAuthPacket(cipher); err != nil {
        mc.cleanup()
        return nil, err
    }
```

```
    // Handle response to auth packet, switch methods if possible
    if err = handleAuthResult(mc, cipher); err != nil {
        // Authentication failed and MySQL has already closed the connection
        // (https://dev.mysql.com/doc/internals/en/authentication-fails.html).
        // Do not send COM_QUIT, just cleanup and return the error.
        mc.cleanup()
        return nil, err
    }

    if mc.cfg.MaxAllowedPacket > 0 {
        mc.maxAllowedPacket = mc.cfg.MaxAllowedPacket
    } else {
        // Get max allowed packet size
        maxap, err := mc.getSystemVar("max_allowed_packet")
        if err != nil {
            mc.Close()
            return nil, err
        }
        mc.maxAllowedPacket = stringToInt(maxap) - 1
    }
    if mc.maxAllowedPacket < maxPacketSize {
        mc.maxWriteSize = mc.maxAllowedPacket
    }

    // Handle DSN Params
    err = mc.handleParams()
    if err != nil {
        mc.Close()
        return nil, err
    }

    return mc, nil
}
```

golang在进行数据库操作，一般来说我们使用Open函数创建一个数据库（操作）句柄： func Open(riverName, dataSourceName string) (DB, error) 我们知道，返回的DB是一个具有零到多个底层连的连接池，可以安全的被多个go程使用，并且会维护自身的闲置连接。所以一般来说open函数我们调用一次，几乎很少需要关闭DB。 连接池可以使得连接更好的控制，于是简单配置甚至是不做什么置就可以用了，事实好像也是如此，连上数据库就都ok了。但是！你很可能会遇到下面这样的错误: mysql] 2017/01/15 12:23:43 packets.go:124: write tcp 127.0.0.1:45697->127.0.0.1:3306：write: broken pipe 其原因很有可能就是你的连接池没有做好相关配置，或者配置了却设置的值不对，那么时的原因大概有以下几点： 1.最大连接数大于数据库服务器端配置的最大连接数，多余的连接在被使到的时候出现连接超时 2.网络抖动：每次连接的时候提交或维护大量数据，此时网络不稳定导致连接时 3.占用的资源未被释放，我们一般只open一次，获取一个*DB,但是在使用stmt，rows或者是tx时用完没有关闭，耗尽资源也有可能导致连接超时。 对于出现的3中问题，你可以试试下面的解决方案: 1.对于最大连接数的问题，调用函数Db.SetMaxOpenConns()设置值（小于数据库配置的最大连接） .对于维护大量数据，可以尝试采用事务操作，若失败，回滚。然后重试。 3.使用defer关键字，在启操作的时候直接在下一行加上defer *.close()函数return的时候会执行相关的关闭函数。 如果你尝试以上的解决方案，还是发现偶尔会出现连接超时的错误的话，很可能是因为数据库本身对连接有一个时时间的设置，如果超时时间到了数据库会单方面断掉连接，此时再用连接池内的连接进行访问就会错，连接池中的每个连接都维护了一个创建时间，取连接的时候会自动检查的，你可以试试调用db的etConnMaxLifetime（）方法来给db设置一个超时时间，时间小于数据库的超时时间即可。 以上仅

个人的一下看法，若有错误之处，欢迎批评指出。如果你尝试了以上的解决方案，还是发现偶尔会出连接超时的错误的话，很可能是因为数据库本身对连接有一个超时时间的设置，如果超时时间到了数库会单方面断掉连接，此时再用连接池内的连接进行访问就会出错，

自己看了一下databas/sql和go-sql-driver的源码，出现这个错误之后官方提供的方法是会进行两次重连操作，如果两次重连都没有成功，就创建一个新的SQL链接，所以出现这个错误日志没什么影响当然最好的做法是设置一下链接的空闲超时时间，设置的比Mysql服务器超时时间短就可以了，设置这个之后就不会报这个错误了。当然不设也可以，设置了的话程序处理会更优雅一点！

https://blog.csdn.net/pangudashu/article/details/54291558#commentBox

https://github.com/go-sql-driver/mysql/wiki/Examples