Go语言数据库编程

一、课前准备

> 说明：提前需要让学生做的课前准备，比如环境安装部署、工具安装、插件安装等，需要学生提前做的都放这，并且给出下载链接或信息源；

1. Golang开发环境环境安装就绪；
2. Golang-IDE开发环境安装就绪；
3. 练习Go语言基础语法代码；
4. 熟练掌握Go并发编程；
5. 熟练掌握Go网络编程；
6. MySQL数据库安装并启动；
7. 具备一定SQL编写能力；
8. MongoDB数据库安装并启动；

二、课堂主题

> 说明：本堂课的总体概述，明确课堂主题和主体；

本节主要介绍Go语言当中如何与数据库进行交互，我们本次课程会讲解Go语言与MySQL及MongoDB的交互，当然主要研究的是针对数据库的CURD。

三、课堂目标

> 说明：主要是让学生了解，学了本堂课后，能达到的一个期望值，要量化；

1. 掌握Go语言与MySQL的连接；
2. 掌握Go语言对MySQL数据库的CURD；
3. 掌握Go语言与MongoDB的连接；
4. 掌握Go语言对MongoDB数据库的CURD；

四、知识点

数据库编程是开发人员不得不面对的话题，我们本章将会介绍传统的关系型数据库MySQL的开发以及非关系型数据库MongoDB的开发。

1. MySQL编程

1.1 环境检查

在编写代码之前，我们还是要确保MySQL已经启动了，为了演示的方便，建议大家统一执行下面的脚本。

```
root@linuxkit-025000000001:/# mysql -uroot -proot
Warning: Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.46 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
```

在登陆mysql后，建立一个名为`teacher`的数据库。

```
drop database if exists teacher;
create database teacher character set utf8;
use teacher;
```

在确定mysql可以正常访问后，我们来尝试编写连接mysql，进行CURD的代码。

1.2 连接到mysql

在进行mysql编程时，我们要用到Go语言官方包sql以及mysql的驱动。

```
import (
  "database/sql"

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

mysql驱动引用我们要匿名导入，后续的使用都是通过官方sql来操作。 首先要借助`Open`的函数来打开数据库，这也是访问的前提，函数原型如下：

```
func Open(driverName, dataSourceName string) (*DB, error)
```

- driverName 传入驱动名字，我们传入：`mysql`
- dataSourceName 代表数据源，格式是：`username:password@protocol(address)/dbname?param=value`

如果我们要访问mysql的数据库，参数可以类似这样传递：

```
 db, err := sql.Open("mysql", "root:abc123@tcp(127.0.0.1:3306)/teacher?
charset=utf8")
  if err != nil {
    log.Panic("failed to open mysql ", err)
  }
```

但是需要注意的是，本处的err并非代表连接全部错误，换句话说，当err为nil的时候，并非代表连接没有问题。

我们在Open得到DB这个结构后，我们应调用其内部的Ping函数来判断是否存在问题，如果此时没问题，则代表确实是真的连接数据库成功了！

```
err = db.Ping()
  if err != nil {
    fmt.Println("Failed to ping ", err)
    return
  }
```

1.3 MySQL无结果集操作

在连接到数据库后，我们就可以尝试进行数据库操作，对于关系型数据库来说，也就是执行SQL。在我们编程开发时，SQL可以大体分为2类：有结果集和无结果集。所谓有结果集，就是执行SQL后有数据返回的，比如查询类的SQL都是有结果集的，而另一类就是没有结果返回的，比如create，drop，delete，update等语句，很显然无结果集的处理更简单一些，我们先来处理无结果集的操作。

在Go语言访问mysql的操作里，我们主要用`db.Exec`来执行无结果集函数，当然也并非完全没有反馈，我们可以得到影响的记录数，如果新增时，也会得到一个LastInsertId。

```
func (db *DB) Exec(query string, args ...interface{}) (Result, error)
type Result interface {
  // LastInsertId returns the integer generated by the database
  // in response to a command. Typically this will be from an
  // "auto increment" column when inserting a new row. Not all
  // databases support this feature, and the syntax of such
  // statements varies.
  LastInsertId() (int64, error)

  // RowsAffected returns the number of rows affected by an
  // update, insert, or delete. Not every database or database
  // driver may support this.
  RowsAffected() (int64, error)
}
```

这样，我们就可以将create语句，insert，update等语句都来执行一下试试了。

案例：创建一个学生表，在其插入一条记录，并且修改该记录。

```
create table t_student(name varchar(30),age int, sex varchar(5));
```

1.4 MySQL有结果集操作

对于有结果集的SQL处理，比无结果集显然麻烦一些，除了要执行SQL外，还要处理结果集的问题。 对于查询类的SQL，我们使用db.Query来调用。

```
func (db *DB) Query(query string, args ...interface{}) (*Rows, error)
func (rs *Rows) Next() bool //判断是否存在下一条信息
func (rs *Rows) Scan(dest ...interface{}) error //扫描结果集
func (rs *Rows) Columns() ([]string, error) //获得列
```

在通过Query获得了Rows之后，我们可以用Next来判断是否存在有下一个结果集，Scan来扫描结果集到我们的接收变量中，需要格外注意Rows可能存在多条数据，所以Next是每次判断下一条。

```
rows, err := db.Query("SELECT name FROM t_student WHERE age = ?", age)
if err != nil {
    log.Fatal(err)
```

```
}
for rows.Next() {
    var name string
    if err := rows.Scan(&name); err != nil {
        log.Fatal(err)
    }
    fmt.Printf("%s is %d\n", name, age)
}
if err := rows.Err(); err != nil {
    log.Fatal(err)
}
```

采用这样的逻辑，我们可以处理查询结果的展现了！ 小练习：显示学生表的全部信息。

```
select * from t_student;
```

2. MongoDB编程

MongoDB是一种文档型数据库，对于文档式数据存储有着天然的优势，在Go语言之中要对MongoDB进行访问，需要使用MongoDB关于Go语言的驱动，之前可以选择gopkg.mgo.v2，由于其已经不再维护，所以现在我们多数要使用官方的驱动。

```
go.mongodb.org/mongo-driver/mongo
```

在学习操作MongoDB的思路上，与MySQL是类似的。

2.1 连接到MongoDB

在连接之前，首先要确保MongoDB是启动的，在使用`mongo`测试可以连接到MongoDB后，就可以使用官方驱动库的API来尝试连接数据库了！

```
import (
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
    "go.mongodb.org/mongo-driver/mongo/readpref"
)

client, err :=
mongo.NewClient(options.Client().ApplyURI("mongodb://localhost:27017"))
```

可以使用client进行数据库连接，不过也可以用mongo包自带的connect函数进行连接，通常连接时，需要设置一个超时时间：

```
ctx, _ := context.WithTimeout(context.Background(), 10*time.Second)
client, err := mongo.Connect(ctx,
options.Client().ApplyURI("mongodb://localhost:27017"))
```

与MySQL类似，在连接后，同样可以用Ping函数来测试连接是否正常。

```
ctx, _ = context.WithTimeout(context.Background(), 2*time.Second)
err = client.Ping(ctx, readpref.Primary())
```

- Primary 代表主节点

2.2 集合操作

由于MongoDB的结构是`库-->集合-->document`，而库的创建方面非常open，因此我们的操作基本都是以集合为主。集合本身也无须特别创建，在新增文档时就会同时创建该集合（如果该集合不存在的情况下）。

```
collection := client.Database("school").Collection("student")
```

我们通过client的指针可以很方便的得到Dtabase对象，而Database对象可以得到Collection对象。

针对文档的操作，我们需要bson结构，因此要导入bson包：

```
"go.mongodb.org/mongo-driver/bson"
```

- 新增文档

```
ctx, _ = context.WithTimeout(context.Background(), 5*time.Second)
res, err := collection.InsertOne(ctx, bson.M{"name": "pi", "value": 3.14159})
id := res.InsertedID
```

Collection对应Insert相关函数，就可以向集合内部添加文档，注意文档需要用bson.M来构造对象。

- 修改文档

```
func (coll *Collection) UpdateOne(ctx context.Context, filter interface{},
update interface{},opts ...*options.UpdateOptions) (*UpdateResult, error)
```

与mongo客户端操作类似，如果要精确修改某条记录（非覆盖），最好使用`$set`操作符。

```
collection.UpdateOne(ctx, bson.M{"name": "pi"}, bson.M{"$set": bson.M{"name":
"yy"}})
```

- 删除文档

```
func (coll *Collection) DeleteOne(ctx context.Context, filter interface{},
  opts ...*options.DeleteOptions) (*DeleteResult, error) {

  return coll.delete(ctx, filter, true, rrOne, opts...)
}
```

- 查看文档

查看文档使用Find方法，可以一次获取多条，filter仍然代表过滤条件。

```
func (coll *Collection) Find(ctx context.Context, filter interface{},
  opts ...*options.FindOptions) (*Cursor, error)
func (c *Cursor) Next(ctx context.Context) bool
func (c *Cursor) Decode(val interface{}) error
```

查看文档主要要处理结果集，Cursor就是我们要处理的结果，与处理MySQL结果集类似，我们使用Next与Decode相配合就可以解决问题。

```
ctx, _ = context.WithTimeout(context.Background(), 30*time.Second)
cur, err := collection.Find(ctx, bson.D{})
if err != nil { log.Fatal(err) }
defer cur.Close(ctx)
for cur.Next(ctx) {
   var result bson.M
   err := cur.Decode(&result)
   if err != nil { log.Fatal(err) }
   // do something with result....
}
if err := cur.Err(); err != nil {
  log.Fatal(err)
}
```

bson.D{} 构造的实际是一个空对象，而作为查询条件就代表了所有数据，从处理上来看，最终就是把查询结果再次变成Bson格式。

五、拓展点

> 说明：
>
> 1. 典型面试题、笔试题；
> 2. 新技术 or 经验分享；
> 3. 未来计划、行业趋势分享；

六、总结

> 说明：
>
> 1. 回顾本堂课所有知识点；
> 2. 提示`注意点`和`重点`；
> 3. 提示学习方法；
> 4. 提示哪些需要记、哪些需要背、哪些代码需要敲；

- 注意核心API
- 注意数据库编程学习的套路
- 熟练编写MySQL与MongoDB的增删改查

七、大作业

> 说明：
>
> 1. 给出明确的作业要求，形成文字、图片或测试题等形式；
> 2. 给出明确的解答方式；
> 3. 频次低，可设计为：1次/周 或 1次/2周；

可从下面的检测题题库任意选择。

八、集中答疑

> 说明：完成本堂课所有知识点的讲解后，由学员集中提问，讲师逐一解答；

九、检测题

> 说明：
>
> 1. 针对本堂课，设计5-10个题，由学员课下完成（课下刷题）；
> 2. 出题范围可从检查点里挑选，可以是面试题或笔试题；
> 3. 题型要求是 单选、多选、判断、填空中的一种或多种；
> 4. 频次高，原则上每次课都要有检测题；

十、下节课预告

> 说明：
>
> 1. echo框架的使用；
> 2. echo框架打造博客系统；