Go语言 数据操作 MongoDB

MongoDB 简介

MongoDB是一个高性能，开源，无模式的文档型数据库，是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，采用的是类似json的bjson格式来存储数据，因此可以存储比较复杂的数据类型。Mongo最大的特点是他支持的查询语言非常强大，其语法有点类似于面向 对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

下表将帮助您更容易理解Mongo中的一些概念：

SQL术语/概念	MongoDB术语/概念	解释/说明
database	database	数据库
table	collection	数据库表/集合
row	document	数据记录行/文档
column	field	数据字段/域
index	index	索引
table joins		表连接,MongoDB不支持
primary key	primary key	主键,MongoDB自动将_id字段设置为主键

在MongoDB中，指定索引插入比不指定慢很多，这是因为，MongoDB里每一条数据的_id值都是唯一的。

当在不指定id插入数据的时候，其id是系统自动计算生成的。MongoDB通过计算机特征值、时间、进程ID与随机数来确保生成的_id是唯一的。

而在指定id插入时，MongoDB每插一条数据，都需要检查此id可不可用，当数据库中数据条数太多的时候，这一步的查询开销会拖慢整个数据库的插入速度。

MongoDB 使用

MongoDB下载地址：

```
https://www.mongodb.com/download-center/community
```

打开客户端

```
mongo.exe
```

> 注意6.0版本不一样，需要自行添加安装Mongoshell。
>
> mongoDB6没有mong.exe和mongdb.exe,要想通过命令行启动mongoDB需要自己下载一个Mongoshell，下载地址https://www.mongodb.com/try/download/shell，直接下载即可。

创建数据库

```
use go_db;
```

创建集合

```
db.createCollection("student");
```

添加MongoDB依赖

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

连接MongoDB

链接数据库

```
func Connect(ctx context.Context, opts ...*options.ClientOptions)
```

Connect 需要两个参数，一个context和一个options.ClientOptions对象

```
var client *mongo.Client

func initDB() {
    // 设置客户端选项
    clientOptions := options.Client().ApplyURI("mongodb://localhost:27017")
    // 连接 MongoDB
    var err error
    client, err = mongo.Connect(context.TODO(), clientOptions)
    if err != nil {
        fmt.Println("链接失败")
        log.Panic(err)
    }

    // 检查连接
    err = client.Ping(context.TODO(), nil)
    if err != nil {
        log.Panic(err)
    }

    fmt.Println("链接成功")
}
```

上面代码的流程就是 创建 链接对象 option 和 context , 然后写入 mongo.Connect ， Connect 函数返回一个链接对象 和一个错误 对象，如果错误对象不为空，那就链接失败了.
然后我们可以再次测试，链接：client.Ping(context.TODO(), nil)
cilent 对象 Ping 就好了，他会返回一个错误对象，如果不为空，就链接失败了。

创建数据表的链接对象

```
collectionStudent := client.Database("go_db").Collection("student")
```

go_db是数据库，student是数据表

断开链接对象

```
client.Disconnect()
```

如果我们不在使用 链接对象，那最好断开，减少资源消耗

```
err = client.Disconnect(context.TODO())
if err != nil {
    log.Fatal(err)
}
fmt.Println("MongoDB链接已关闭.")
```

操作MongoDB数据库

MongoDB中的JSON文档存储在名为BSON(二进制编码的JSON)的二进制表示中。与其他将JSON数据存储为简单字符串和数字的数据库不同，BSON编码扩展了JSON表示，使其包含额外的类型，如int、long、date、浮点数和decimal128。这使得应用程序更容易可靠地处理、排序和比较数据。

在go.mongodb中有两种族来使用bson数据，分别是D和RAW。

D族是使用原生Go形式来构造一个BSON对象。这个对于使用命令来操作mongoDB是十分有用的。

D()由下面4种类型：

- D:一个BSON文档，这个是有序的。
- M:一个无序的map。它除了无序之外和D是一样的（可以理解为map和bson是可以转换）。
- A:一个BSON形式的数组。
- E:一个D里面的单独元素。(就是文档里的一个元素)

RAW族是被用来判断是否为bytes的一个slice。

你也可以用look up()方法从RAW取得一个元素。这可以在你将BSON转化为另一个形式的数 据时是十分有用的(原文大概意思是可以节省你转化数据时的开销)。

定义学生结构体

```
// 定义学生结构体
type Student struct {
    Name string
    Age  int
}
```

插入单个文档

```
collection.InsertOne()
```

```
// 插入单条数据
```

```
func insertData() {
    // 链接mongodb
    initDB()
    // 成功后断开mongodb
    defer client.Disconnect(context.TODO())

    // 初始化
    s := Student{
        Name: "张三",
        Age:  18,
    }
    // 链接数据表对象
    collection := client.Database("go_db").Collection("student")
    // 插入单条数据
    ior, err := collection.InsertOne(context.TODO(), s)
    if err != nil {
        log.Fatal(err)
    } else {
        fmt.Printf("ior.InsertedID: %v\n", ior.InsertedID)
    }
}
```

插入多条文档

```
collection.InsertMany()
```

不同的是接受一个 切片作为数据集合:

```
// 插入多条数据
func insertManyData() {
    // 链接mongodb
    initDB()
    // 成功后断开mongodb
    defer client.Disconnect(context.TODO())
    // 初始化
    s := Student{
        Name: "王五",
        Age:  23,
    }
    s1 := Student{
        Name: "李四",
        Age:  20,
    }
    // 声明成切片
    stus := []interface{}{s, s1}

    // 链接数据表对象
    collection := client.Database("go_db").Collection("student")
    // 插入多条数据
    ior, err := collection.InsertMany(context.TODO(), stus)
    if err != nil {
        log.Fatal(err)
    } else {
        fmt.Printf("ior.InsertedIDs: %v\n", ior.InsertedIDs...)
```

```
    }
}
```

更新单个文档

```
collection.UpdateOne()
```

如果有多个满足条件的，只更新第一条

```
// 修改单条数据
func updatetData() {
    // 链接mongodb
    initDB()
    // 成功后断开mongodb
    defer client.Disconnect(context.TODO())
    // 链接数据表对象
    collection := client.Database("go_db").Collection("student")

    // filter: 包含查询操作符的文档，可以用来选择要查询的文档
    // 查询到name=李四的文档
    filter := bson.D{{Key: "name", Value: "李四"}}
    // 修改name 为张三
    update := bson.D{
        {Key: "$set", Value: bson.D{{Key: "name", Value: "张三"}}},
    }
    ur, err := collection.UpdateOne(context.TODO(), filter, update)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Printf("ur.ModifiedCount: %v\n", ur.ModifiedCount)
}
```

更新多个文档

```
collection.UpdateMany()
```

```
// 修改多条数据
func updatetManyData() {
    // 链接mongodb
    initDB()
    // 成功后断开mongodb
    defer client.Disconnect(context.TODO())
    // 链接数据表对象
    collection := client.Database("go_db").Collection("student")

    // 查询到name=张三的文档
    filter := bson.D{{Key: "name", Value: "张三"}}
    // 修改age加一岁 $inc增加  $set设置成
    update := bson.D{{Key: "$inc",
        Value: bson.D{
            {Key: "age", Value: 1},
        },
    }}
```

```
    ur, err := collection.UpdateMany(context.TODO(), filter, update)

    if err != nil {
        log.Fatal(err)
    }
    fmt.Printf("ur.ModifiedCount: %v\n", ur.ModifiedCount)
}
```

查询单个文档

```
collection.FindOne()
```

```
// 查找单个文档
func findData() {
    // 链接mongodb
    initDB()
    // 成功后断开mongodb
    defer client.Disconnect(context.TODO())
    // 链接数据表对象
    collection := client.Database("go_db").Collection("student")
    //查找成功赋值
    var s Student
    // 查找name=王五
    filter := bson.D{{Key: "name", Value: "王五"}}
    err := collection.FindOne(context.TODO(), filter).Decode(&s)
    if err != nil {
        log.Fatal(err)
    } else {
        fmt.Println(s)
    }
}
```

> 查找文档需要一个filter文档， 以及一个指针在它里边保存结果的解码

查询多个文档

```
collection.Find()
```

```
// 查找多个文档
func findManyData() {
    // 链接mongodb
    initDB()
    // 成功后断开mongodb
    defer client.Disconnect(context.TODO())
    // 链接数据表对象
    collection := client.Database("go_db").Collection("student")

    // 查找name=张三
    filter := bson.D{{Key: "name", Value: "张三"}}

    cursor, err := collection.Find(context.TODO(), filter)
    if err != nil {
        log.Fatal(err)
```

```
    }
    //关闭上下文
    defer cursor.Close(context.TODO())
    // 定义切片
    var students []Student
    err = cursor.All(context.TODO(), &students)

    for _, student := range students {
        fmt.Println(student)
    }
}
```

> 查找文档需要一个filter文档， 以及一个指针在它里边保存结果的解码

复合查询

$regex 模糊查询

```
filter := bson.M{"name": bson.M{"$regex":"张"}}
```

in($in) 包含 和 no in($nin) 不包含

```
filter := bson.M{"name": bson.M{"$in":[]string{"张三","李四"}}}
```

and($and) 和

```
filter := bson.M{"$and": []bson.M{{"name": "张三"}, {"age": 18}}}
```

or($or) 或

```
filter := bson.M{"$or": []bson.M{{"name": "张三"}, {"age": 20}}}
```

比较函数

- `!=` ($ne)
- `>` ($gt)
- `<` ($lt)
- `>=` ($gte)
- `<=` ($lte)

```
filter := bson.M{"age": bson.M{"$gt": 18}}
```

```
// 复合查询
func findComplexData() {
    // 链接mongodb
    initDB()
    // 成功后断开mongodb
    defer client.Disconnect(context.TODO())
    // 链接数据表对象
    collection := client.Database("go_db").Collection("student")

    // 模糊查找name like张
```

```
    // filter := bson.M{"name": bson.M{"$regex": "张"}}

    // name包含张三和李四的
    // filter := bson.M{"name": bson.M{"$in": []string{"张三", "李四"}}}

    // name=张三 and age=18
    // filter := bson.M{"$and": []bson.M{{"name": "张三"}, {"age": 18}}}

    // name=张三 或者or age=20
    // filter := bson.M{"$or": []bson.M{{"name": "张三"}, {"age": 20}}}

    // 年龄age>18
    filter := bson.M{"age": bson.M{"$gt": 18}}

    cursor, err := collection.Find(context.TODO(), filter)
    if err != nil {
        log.Fatal(err)
    }
    //关闭上下文
    defer cursor.Close(context.TODO())
    // 定义切片
    var students []Student
    err = cursor.All(context.TODO(), &students)

    for _, student := range students {
        fmt.Println(student)
    }
}
```

聚类聚合函数

- $sum 计算总和。
- $avg 计算平均值。
- $min 获取集合中所有文档对应值得最小值。
- $max 获取集合中所有文档对应值得最大值。
- $first 根据资源文档的排序获取第一个文档数据。
- $last 根据资源文档的排序获取最后一个文档数据。
- $push 在结果文档中插入值到一个数组中。
- $addToSet 在结果文档中插入值到一个数组中，但不创建副本。

定义最大时间

```
opts := options.Aggregate().SetMaxTime(2 * time.Second)
```

定义查询语句

```
groupStage := bson.D{{Key: "$group", Value: bson.D{{Key: "_id", Value: "$major"},
{Key: "ageAvg", Value: bson.D{{Key: "$avg", Value: "$age"}}}}}}
```

查询

```
result, err := collection.Aggregate(context.TODO(), mongo.Pipeline{groupStage},
opts)
```

赋值，注意这里类型可以自己定义，也可以直接用bson.M

```
var students []bson.M
err = result.All(context.TODO(), &students)
```

```
// 复合查询
func findGroupData() {
    // 链接mongodb
    initDB()
    // 成功后断开mongodb
    defer client.Disconnect(context.TODO())
    // 链接数据表对象
    collection := client.Database("go_db").Collection("student")

    // 定义最大时间
    opts := options.Aggregate().SetMaxTime(2 * time.Second)

    //查询语句 age和
    // groupStage := bson.D{{Key: "$group", Value: bson.D{{Key: "_id", Value:
"$major"}, {Key: "ageSum", Value: bson.D{{Key: "$sum", Value: "$age"}}}}}

    //查询语句 age平均值
    // groupStage := bson.D{{Key: "$group", Value: bson.D{{Key: "_id", Value:
"$major"}, {Key: "ageAvg", Value: bson.D{{Key: "$avg", Value: "$age"}}}}}

    // 查询语句 age最小值
    // groupStage := bson.D{{Key: "$group", Value: bson.D{{Key: "_id", Value:
"$major"}, {Key: "ageMin", Value: bson.D{{Key: "$min", Value: "$age"}}}}}

    // 查询语句 age最大值
    groupStage := bson.D{{Key: "$group", Value: bson.D{{Key: "_id", Value:
"$major"}, {Key: "ageMax", Value: bson.D{{Key: "$max", Value: "$age"}}}}}

    // 查询
    result, err := collection.Aggregate(context.TODO(),
mongo.Pipeline{groupStage}, opts)
    if err != nil {
        log.Fatal(err)
    }
    //关闭上下文
    defer result.Close(context.TODO())

    //bson.M
    var students []bson.M
    err = result.All(context.TODO(), &students)

    // // 自定义切片
    // var students []Student
    // err = result.All(context.TODO(), &students)

    for _, student := range students {
        fmt.Println(student)
    }
}
```

删除单个文档

```
collection.DeleteOne()
```

```
// 删除单个文档
func deleteData() {
    // 链接mongodb
    initDB()
    // 成功后断开mongodb
    defer client.Disconnect(context.TODO())
    // 链接数据表对象
    collection := client.Database("go_db").Collection("student")

    // 删除name=王五的数据
    filter := bson.D{{Key: "name", Value: "王五"}}
    dr, err := collection.DeleteOne(context.TODO(), filter)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Printf("dr.DeletedCount: %v\n", dr.DeletedCount)
}
```

删除多个文档

```
collection.DeleteMany()
```

```
// 删除多个文档
func deleteManyData() {
    // 链接mongodb
    initDB()
    // 成功后断开mongodb
    defer client.Disconnect(context.TODO())
    // 链接数据表对象
    collection := client.Database("go_db").Collection("student")
    // 删除name=张三的数据
    filter := bson.D{{Key: "name", Value: "张三"}}
    dr, err := collection.DeleteMany(context.TODO(), filter)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Printf("dr.DeletedCount: %v\n", dr.DeletedCount)
}
```