**作者：熊忠波**

**Mongoose 是什么?**

一般我们不直接用[**MongoDB**](http://lib.csdn.net/base/mongodb)的函数来操作[**mongodb**](http://lib.csdn.net/base/mongodb)[**数据库**](http://lib.csdn.net/base/mysql) Mongose就是一套操作MongoDB数据库的接口.

**Schema**

一种以文件形式存储的数据库模型骨架，无法直接通往数据库端，也就是说它不具备对数据库的操作能力.可以说是数据属性模型(传统意义的表结构)，又或着是“集合”的模型骨架

/\* 定义一个 Schema \*/

var mongoose = require("mongoose");

var TestSchema = new mongoose.Schema({

name : { type:String },//属性name,类型为String

age : { type:Number, default:0 },//属性age,类型为Number,默认为0

time : { type:Date, default:Date.now },

email: { type:String,default:‘‘}

});

上面这个 TestSchema包含4个属性 [name, age, time, email]

**Model**

由Schema构造生成的模型，除了Schema定义的数据库骨架以外，还具有数据库操作的行为，类似于管理数据库属性、行为的类

var db = mongoose.connect("mongodb://127.0.0.1:27017/test");

// 创建Model

var TestModel = db.model("test1", TestSchema);

test1 数据库中的集合名称, 不存在会创建.

**Entity**

由Model创建的实体，使用save方法保存数据，Model和Entity都有能影响数据库的操作，但Model比Entity更具操作性

var TestEntity = new TestModel({

name : "Lenka",

age : 36,

email: "lenka@qq.com"

});

console.log(TestEntity.name); // Lenka

console.log(TestEntity.age); // 36

**游标**

MongoDB 使用游标返回find的执行结果.客户端对游标的实现通常能够对最终结果进行有效的控制。可以限制结果的数量，略过部分结果，根据任意键按任意顺序的组合对结果进行各种排序，或者是执行其他一些强的操作。

**ObjectId**

存储在mongodb集合中的每个文档（document）都有一个默认的主键\_id，这个主键名称是固定的，它可以是mongodb支持的任何数据类型，默认是ObjectId。

ObjectId是一个12字节的 BSON 类型字符串。按照字节顺序，依次代表： 4字节：UNIX时间戳 3字节：表示运行MongoDB的机器 2字节：表示生成此\_id的进程 3字节：由一个随机数开始的计数器生成的值

**Node.js 中**

package.json 中加入"mongoose": “\*” 字段 npm install 安装依赖.

var mongoose = require("mongoose");

var db = mongoose.connect("mongodb://localhost:27017/test");

然后引用

**API**

var mongoose = require("mongoose");

var db = mongoose.connect("mongodb://localhost:27017/test");

**db - 数据库操作**

1.挂接数据库连接事件,参数1: 也可以是error.

db.connection.on(‘open’, callback);

**Schema - 表结构**

1.构造函数

new mongoose.Schema( { name:{type:String}, age:{type:Number, default:10} } )

2.添加属性

Schema.add( { name: ‘String’, email: ‘String’, age: ‘Number’ } )

3.有时候Schema不仅要为后面的Model和Entity提供公共的属性，还要提供公共的方法

Schema.method( ‘say’, function(){console.log(‘hello’);} ) //这样Model和Entity的实例就能使用这个方法了

4.添加静态方法

Schema.static( ‘say’, function(){console.log(‘hello’);} ) //静态方法，只限于在Model层就能使用

5.追加方法

Schema.methods.say = function(){console.log(‘hello’);}; //静态方法，只限于在Model层就能使用

**model - 文档操作**

1.构造函数, 参数1:集合名称, 参数2:Schema实例

db.model("test1", TestSchema );

2.查询, 参数1忽略,或为空对象则返回所有集合文档

model.find({}, callback);

model.find({},field,callback); 过滤查询,参数2: {’name’:1, ‘age’:0} 查询文档的返回结果包含name , 不包含age.(\_id默认是1)

model.find({},null,{limit:20}); 过滤查询,参数3: 游标操作 limit限制返回结果数量为20个,如不足20个则返回所有.

model.findOne({}, callback); 查询找到的第一个文档

model.findById(‘obj.\_id’, callback); 查询找到的第一个文档,同上. 但是只接受 \_\_id 的值查询

3.创建, 在集合中创建一个文档

Model.create(文档数据, callback))

4.更新,参数1:查询条件, 参数2:更新对象,可以使用MondoDB的更新修改器

Model.update(conditions, update, function(error)

5.删除, 参数1:查询条件

Model.remove(conditions,callback);

**Entity - 文档操作**

1.构造函数, 其实就是model的实例

new TestModel( { name:’xueyou’, age:21 } );

2.创建, 在集合中创建一个文档.

Entity.save(callback);

**修改器和更新器还有删除**

**更新修改器:**

‘$inc’ 增减修改器,只对数字有效.下面的实例: 找到 age=22的文档,修改文档的age值自增1

Model.update({’age’:22}, {’$inc’:{’age’:1} } ); 执行后: age=23

‘$set’ 指定一个键的值,这个键不存在就创建它.可以是任何MondoDB支持的类型.

Model.update({’age’:22}, {’$set’:{’age’:’haha’} } ); 执行后: age=’haha’

通过id来更新

ce.findByIdAndUpdate(data.id,{$set: {'price':data.new\_price}})

‘$unset’ 同上取反,删除一个键

Model.update({’age’:22}, {’$unset’:{’age’:’haha’} } ); 执行后: age键不存在

**数组修改器:**

‘$push’ 给一个键push一个数组成员,键不存在会创建

Model.update({’age’:22}, {’$push’:{’array’:10} } ); 执行后: 增加一个 array 键,类型为数组, 有一个成员 10

‘$addToSet’ 向数组中添加一个元素,如果存在就不添加

Model.update({’age’:22}, {’$addToSet’:{’array’:10} } ); 执行后: array中有10所以不会添加

‘$each’ 遍历数组, 和 $push 修改器配合可以插入多个值

Model.update({’age’:22}, {’$push’:{’array’:{’$each’: [1,2,3,4,5]}} } ); 执行后: array : [10,1,2,3,4,5]

‘$pop’ 向数组中尾部删除一个元素

Model.update({’age’:22}, {’$pop’:{’array’:1} } ); 执行后: array : [10,1,2,3,4] tips: 将1改成-1可以删除数组首部元素

‘$pull’ 向数组中删除指定元素

Model.update({’age’:22}, {’$pull’:{’array’:10} } ); 执行后: array : [1,2,3,4] 匹配到array中的10后将其删除

最复杂的，对象中含有数组，数组中含有对象

ctx.body=await wxdb.update({"wx.openid":data.openid},{"$pull":{"xf.shop\_cart":{"id":data.id}}})

"xf" **:** **{**

"shop\_cart" **:** **[**

**{**

"openid" **:** "ogwUO1qiKl-1wlvs-NWsja5ij1js"**,**

"id" **:** "5b0628d78ebc7f22641a5eb6"**,**

"ware\_price" **:** "3788"**,**

"size" **:** "pro"**,**

"color" **:** "白色"

**},**

**{**

"openid" **:** "ogwUO1qiKl-1wlvs-NWsja5ij1js"**,**

"id" **:** "5b065cd4f9dd2722043263bc"**,**

"ware\_price" **:** "4000"**,**

"size" **:** "p20"**,**

"color" **:** "红色"

**}**

**]**

**}**

**条件查询:**

* “$lt” 小于
* “$lte” 小于等于
* “$gt” 大于
* “$gte” 大于等于
* “$ne” 不等于

Model.find({"age":{ "$get":18 , "$lte":30 } } ).limit(NUMBER).skip(NUMBER)

limit指定返回数量 skip跳过多少

查询 age 大于等于18并小于等于30的文档

**或查询 OR:**

* ‘$in’ 一个键对应多个值
* ‘$nin’ 同上取反, 一个键不对应指定值
* “$or” 多个条件匹配, 可以嵌套 $in 使用
* “$not” 同上取反, 查询与特定模式不匹配的文档

Model.find({"age":{ "$in":[20,21,22.’haha’]} } );

查询 age等于20或21或21或’haha’的文档

Model.find({"$or" : [ {’age’:18} , {’name’:’xueyou’} ] }); 查询 age等于18 或 name等于’xueyou’ 的文档

**类型查询:**

null 能匹配自身和不存在的值, 想要匹配键的值 为null, 就要通过 “$exists” 条件判定键值已经存在 “$exists” (表示是否存在的意思)

Model.find(“age” : { “$in” : [null] , “exists” : true } ); 查询 age值为null的文档

Model.find({name: {$exists: true}},function(error,docs){

//查询所有存在name属性的文档

});

Model.find({telephone: {$exists: false}},function(error,docs){

//查询所有不存在telephone属性的文档

});

**正则表达式:**

MongoDb 使用 Prel兼容的正则表达式库来匹配正则表达式

find( {"name" : /joe/i } )  
查询name为 joe 的文档, 并忽略大小写

find( {"name" : /joe?/i } ) 查询匹配各种大小写组合

**查询数组:**

Model.find({"array":10} ); 查询 array(数组类型)键中有10的文档, array : [1,2,3,4,5,10] 会匹配到

Model.find({"array[5]":10} ); 查询 array(数组类型)键中下标5对应的值是10, array : [1,2,3,4,5,10] 会匹配到

‘$all’ 匹配数组中多个元素

Model.find({"array":[5,10]} ); 查询 匹配array数组中 既有5又有10的文档

‘$size’ 匹配数组长度

Model.find({"array":{"$size" : 3} } ); 查询 匹配array数组长度为3 的文档

‘$slice’ 查询子集合返回

Model.find({"array":{"$skice" : 10} } ); 查询 匹配array数组的前10个元素

Model.find({"array":{"$skice" : [5,10] } } ); 查询 匹配array数组的第5个到第10个元素

**where**

用它可以执行任意javacript语句作为查询的一部分,如果回调函数返回 true 文档就作为结果的一部分返回

find( {"$where" : function(){

for( var x in this ){

//这个函数中的 this 就是文档

}

if(this.x !== null && this.y !== null){

return this.x + this.y === 10 ? true : false;

}else{

return true;

}

} } )

简化版本

find( {"$where" : "this.x + this.y === 10" } )

find( {"$where" : " function(){ return this.x + this.y ===10; } " } )

# 高阶

关于push或者pull深度操作

const result\_pull=await wxdb.update({"wx.openid":ware\_query\_resp.openid},{"$pull":{"xf.pay":{"out\_trade\_no":ware\_query\_resp.out\_trade\_no}}})

//xf是一个对象，pay是一个数组 包含的{}，out\_trade\_no是这个对象的字段

聚合

xf是对象，pay是数组 用$unwind 拆分这个数组

用$match 选定条件，因为已经被拆分，所以原本xf.pay的数组对象，已经变成单条的常规对象，可以直接点出。

$project 可以限定输出的字段，默认\_id 是输出的，可以限制大对象的输出，也可以制定更为精准的xf.pay的输出，1是输出，0是不输入

data=JSON.parse(data)

                 const result\_await=await wxdb.aggregate([

                    { $unwind : "$xf.pay"},

                    { $match : {"wx.openid":data.openid,'xf.pay.pay':'await'}},

                    { $project : {"xf.pay":1}},

                 ])

                 console.log(result\_await)

关于不返回所有数组而只是匹配的这条

模型

**{**

"\_id" **:** ObjectId**(**"5b88efef4124660b15ea0648"**),**

"wx" **:** **{**

"openid" **:** "ogwUO1qiKl-1wlvs-NWsja5ij1js"**,**

"nickname" **:** "叫我东方叔叔"**,**

"sex" **:** NumberInt**(**1**),**

"language" **:** "zh\_CN"**,**

"city" **:** "杭州"**,**

"province" **:** "浙江"**,**

"country" **:** "中国"**,**

"headimgurl" **:** "http://thirdwx.qlogo.cn/mmopen/vi\_32/Q2LT8JFWSggCHRbpKuZdQibl2C7HMfoJicVTOPYVdNysTtVQqQDAbbcxhibrT6C3aevc43W3ow5DfUSkQtunocxtg/132"**,**

"privilege" **:** **[**

**]**

**},**

"register\_date" **:** ISODate**(**"2018-08-31T07:36:15.288+0000"**),**

"\_\_v" **:** NumberInt**(**0**),**

"phone\_number" **:** "17625402448"**,**

"xf" **:** **{**

"address" **:** **[**

**{**

"sheng" **:** "1"**,**

"dizhi" **:** "1"**,**

"youbian" **:** "1"**,**

"xingming" **:** "1"**,**

"phone" **:** "1"

**}**

**],**

"pay" **:** **[**

**{**

"price" **:** "0.01"**,**

"size" **:** "100元"**,**

"color" **:** "默认"**,**

"store\_user" **:** "2"**,**

"ware\_name" **:** "梦幻西游点卡"**,**

"ems" **:** "0.01"**,**

"logo" **:** "../cloud/2/QQ截图20180524095857.png"**,**

"user" **:** "1"**,**

"address" **:** "11"**,**

"phone" **:** "17625402448"**,**

"openid" **:** "oBO4y1i8XzZn-2LuGsO3uZfWBrRE"**,**

"out\_trade\_no" **:** "9755153595280275686173"**,**

"pay" **:** "支付成功"

**},**

**{**

"price" **:** "0.01"**,**

"size" **:** "100元"**,**

"color" **:** "默认"**,**

"store\_user" **:** "2"**,**

"ware\_name" **:** "梦幻西游点卡"**,**

"ems" **:** "0.01"**,**

"logo" **:** "../cloud/2/QQ截图20180524095857.png"**,**

"user" **:** "1"**,**

"address" **:** "11"**,**

"phone" **:** "17625402448"**,**

"openid" **:** "oBO4y1i8XzZn-2LuGsO3uZfWBrRE"**,**

"out\_trade\_no" **:** "24b9153595361117543106"**,**

"pay" **:** "支付成功"

**},**

**{**

"price" **:** "0.01"**,**

"size" **:** "100元"**,**

"color" **:** "默认"**,**

"store\_user" **:** "2"**,**

"ware\_name" **:** "梦幻西游点卡"**,**

"ems" **:** "0.01"**,**

"logo" **:** "../cloud/2/QQ截图20180524095857.png"**,**

"user" **:** "1"**,**

"address" **:** "11"**,**

"phone" **:** "17625402448"**,**

"openid" **:** "oBO4y1i8XzZn-2LuGsO3uZfWBrRE"**,**

"out\_trade\_no" **:** "15bf153595380973000823"**,**

"pay" **:** "支付成功"

**},**

**{**

"price" **:** "0.01"**,**

"size" **:** "100元"**,**

"color" **:** "默认"**,**

"store\_user" **:** "2"**,**

"ware\_name" **:** "梦幻西游点卡"**,**

"ems" **:** "0.01"**,**

"logo" **:** "../cloud/2/QQ截图20180524095857.png"**,**

"user" **:** "1"**,**

"address" **:** "11"**,**

"phone" **:** "17625402448"**,**

"openid" **:** "oBO4y1i8XzZn-2LuGsO3uZfWBrRE"**,**

"out\_trade\_no" **:** "1fa5153595408923208140"**,**

"pay" **:** "await"

**},**

**{**

"price" **:** "0.01"**,**

"size" **:** "100元"**,**

"color" **:** "默认"**,**

"store\_user" **:** "2"**,**

"ware\_name" **:** "梦幻西游点卡"**,**

"ems" **:** "0.01"**,**

"logo" **:** "../cloud/2/QQ截图20180524095857.png"**,**

"user" **:** "1"**,**

"address" **:** "11"**,**

"phone" **:** "17625402448"**,**

"openid" **:** "oBO4y1i8XzZn-2LuGsO3uZfWBrRE"**,**

"out\_trade\_no" **:** "9454153595421954679719"**,**

"pay" **:** "await"

**}**

**]**

**},**

"wxapp" **:** "oBO4y1i8XzZn-2LuGsO3uZfWBrRE"

**}**

**$符是核心，不使用$符号，就变成了泛查询**

let result\_message=await wxdb.findOne({'xf.pay.out\_trade\_no':'9755153595280275686173'},{'xf.pay.$.pay':1})

    console.log(result\_message.xf.pay[0].pay)