第7章 MongoDB数据库操作

Node.js开发实战教程

内容导航

Contents

7.1 MongoDB数据库基础

- NoSQL数据库简介
- MongoDB数据库简介
- MongoDB基本概念
- MongoDB的安装和基本使用

7.2 使用原生驱动连接和操作MongoDB数据库

7.3 使用Mongoose操作MongoDB数据库

7.4 实战演练——开发图书使用管理操作接口

Section 00

【学习目标】

（1）了解MongoDB数据库；

（2）安装和使用MongoDB数据库。

Section 01

【NoSQL数据库简介】

▸▸▸ NoSQL数据库类型

- 键值（Key-Value）存储数据库
- 列存储数据库
- 文档型数据库
- 图形（Graph）数据库

▸▸▸ NoSQL数据库应用场合

- 数据模型比较简单。
- 需要灵活性更强的IT系统。
- 对数据库性能要求较高。
- 不需要高度的数据一致性。
- 特定键容易映射到复杂值的环境。

Section 02

【MongoDB数据库简介】

MongoDB的优势

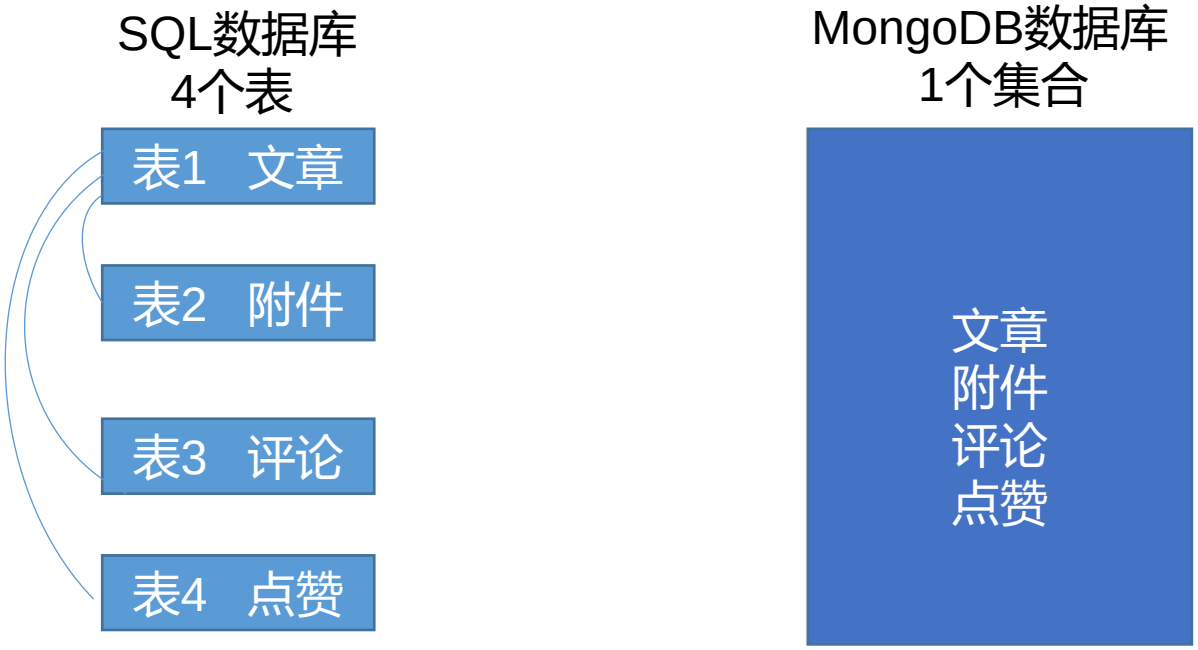
- JSON文档模型
- 动态的数据模式
- 二级索引强大
- 查询功能
- 自动分片
- 水平扩展
- 自动复制
- 高可用
- 文本搜索
- 企业级安全
- 聚合框架MapReduce
- 大文件存储GridFS
- 支持多种编程语言

Section 02

【MongoDB数据库简介】

▸▸▸ MongoDB的应用场合

- 存储网站内容、缓存等大容量、低价值的数据。
- 在高伸缩性的场景中用于对象及JSON数据的存储。



Section 03

【MongoDB基本概念】

▸▸▸ MongoDB与SQL的术语对比

SQL术语	MongoDB术语
Database（数据库）	Database（数据库）
Table（表）	Collection（集合）
Row（行或记录）	Document或BSON Document（文档）
Column（列或字段）	Index（索引）
Table Joins（表联合）	Embedded Documents and Linking（嵌入文档和连接）
Primary Key（主键，将唯一列或列组合定义为主键）	Primary Key（主键，自动设置为_id字段）
Aggregation（聚合）	Aggregation Pipeline（聚合管道）

Section 03

【MongoDB基本概念】

▸▸▸ 数据库

- 一个MongoDB服务器可以建立多个独立的数据库，每个数据库都有自己的集合和权限，不同的数据库也放置在不同的文件中。
- MongoDB中默认的数据库为local。

▸▸▸ 集合

- 集合类似于SQL数据库管理系统中的数据表，是一个MongoDB文档的集合。
- 集合存在于数据库中，没有固定的结构。
- 集合只在文档内容插入之后才会创建。

```
{"site":"www.jd.com"}
{"site":"www.taobao.com","name":"淘宝","hit":125}
{"site":"www.tmall.com","name":"天猫","purchases":23}
```

Section 03

【MongoDB基本概念】

▸▸▸ 文档

- 文档是一组键值对。
- 与SQL数据库不同，MongoDB的文档不需要设置相同的字段。
- MongoDB文档中存储的是BSON格式的数据。

SQL数据库表

id 序号	isbn 书号	name 书名	author 作者	press 出版社	price 价格	pubdate 出版日期
1	9787111544937	深入理解计算机系统（原书第3版）	兰德尔 E.布莱恩特	机械工业出版社	139.00	2016-12-01
3	9787115450326	网络操作系统 (第2版)	陈景亮	人民邮电出版社	69.80	2017-06-01
4	9787115483690	CentOS Linux系统管理与运维(第2版)	张金石	人民邮电出版社	69.80	2018-07-01

MongoDB文档


```
_id: ObjectId("5cdb7b5d7d422f0bd0088f3e")
isbn: "9787115450326"
name: "网络操作系统 （第2版）"
author: Array
  0: "陈景亮"
  1: "钟小平"
  2: "宋大勇"
press: "人民邮电出版社"
price: 69.80
pubdate: "2017-07-01"
comments: Array
```

```
_id: ObjectId("5cdb74a97d422f0bd0088f3d")
isbn: "9787111544937"
name: "深入理解计算机系统（原书第3版）"
author: Array
press: "机械工业出版社"
price: 139.00
pubdate: "2016-12-01"
comments: Array
```

```
_id: ObjectId("5cdb7f647d422f0bd0088f3f")
isbn: "9787115483690"
name: "CentOS Linux系统管理与运维(第2版)"
author: Array
press: "人民邮电出版社"
price: 69.80
pubdate: "2018-08-01"
comments: Array
```

Section 03

【MongoDB基本概念】

ObjectId类型的主键

- MongoDB文档必须有一个名为_id的主键。
- 主键的值可以是任何类型的，默认是一个ObjectId对象。MongoDB采用一个ObjectId类型的值作为主键。
- ObjectId是12字节的BSON字符串，一个字节为两位十六进制数，共有24位十六进制数，其组成是固定格式，例如：

  5cdb7f647d422f0bd0088f3f

  - 4个字节表示当前的时间戳；
  - 3个字节代表所在主机唯一标识；
  - 2个字节表示进程ID；
  - 3个字节是一个自动生成的随机数。

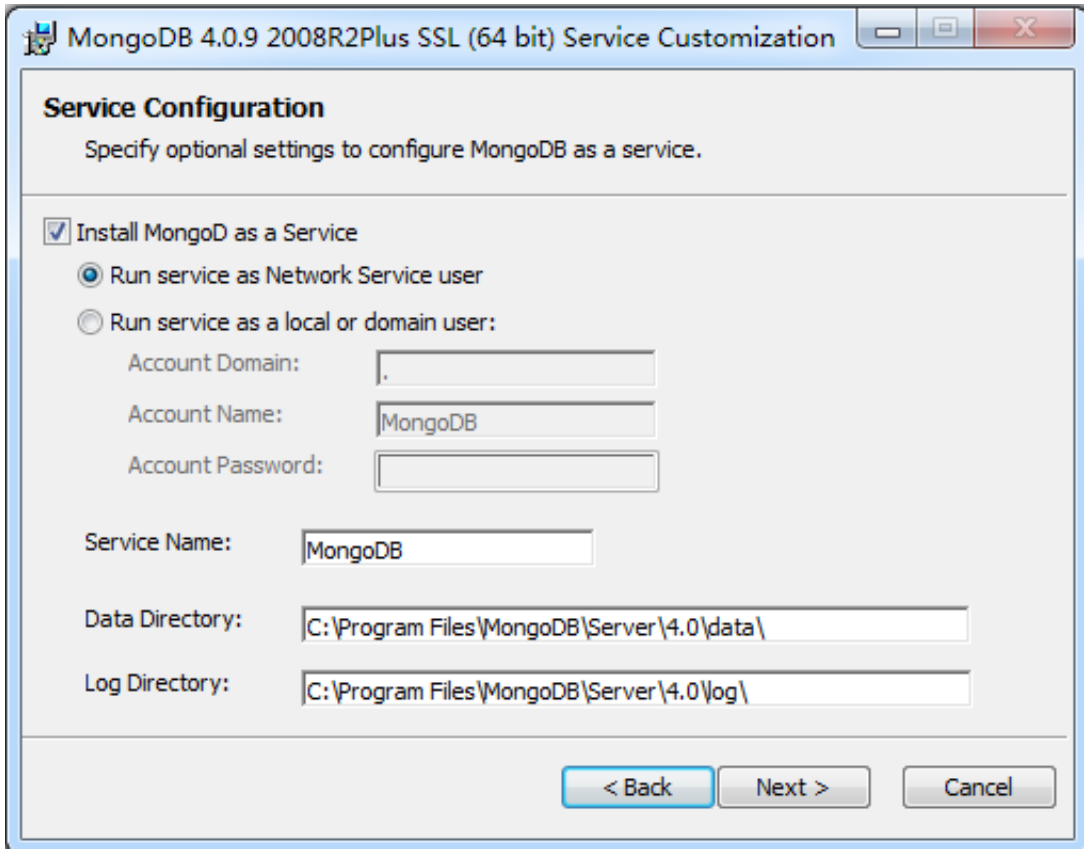
Section 04

【MongoDB的安装和基本使用】

▸▸▸ 安装MongoDB

- 从MongoDB官网下载社区版安装包。
- 确认系统已安装Visual C++ 2015运行库和Microsoft.NET Framework 4.5。
- 运行MongoDB安装包启动安装向导，根据向导提示完成安装和初始配置。

MongoDB安装的服务配置



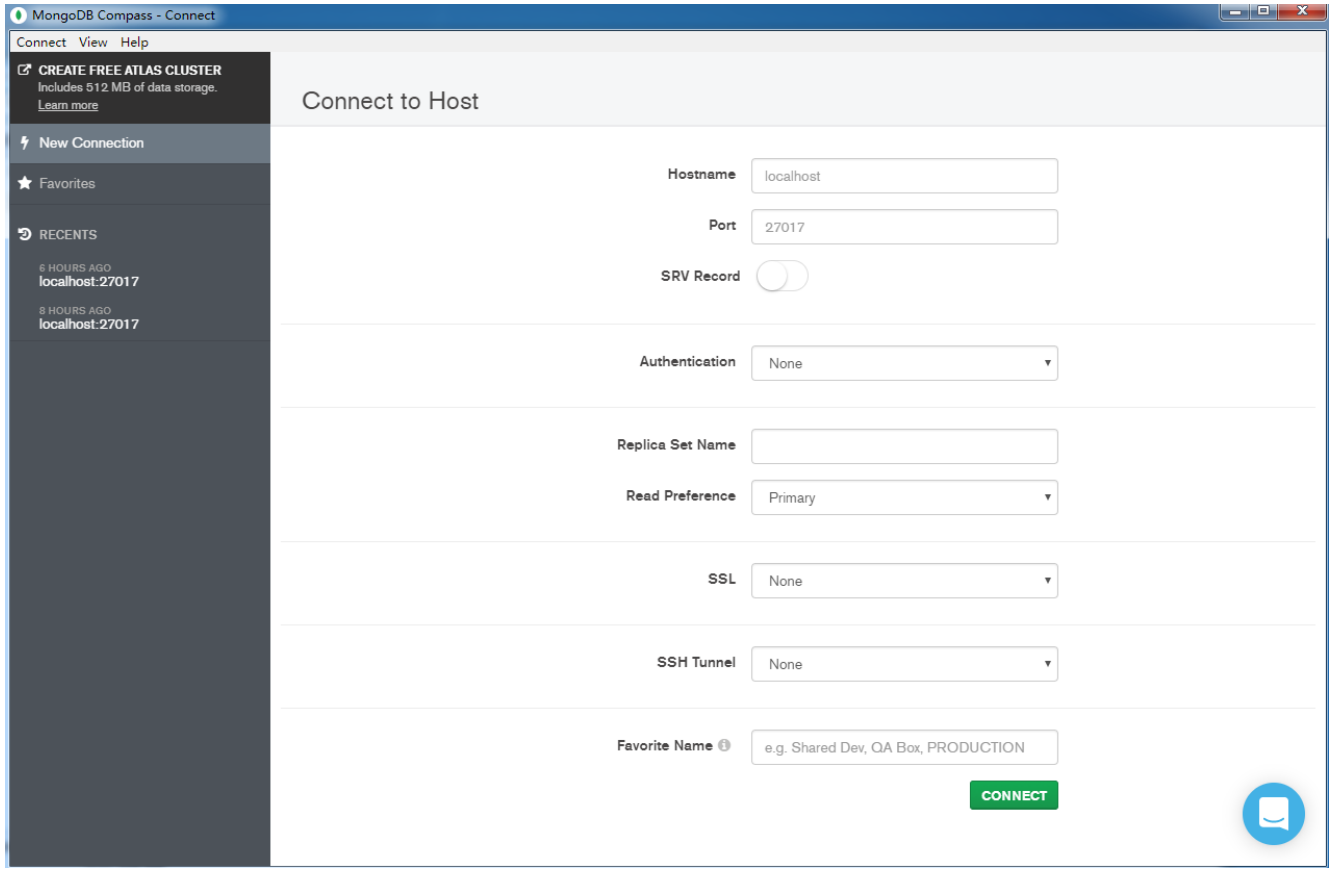
Section 04

【MongoDB的安装和基本使用】

▸▸▸ 安装和使用可视化工具Compass

- 从MongoDB官网下载MongoDB Compass安装包。
- 运行该安装包，默认生成C:\Program Files (x86)\MongoDB Compass Installer\MongoDBCompass.exe文件，再执行该文件正式安装Compass。
- 设置访问MongoDB的连接参数，单击“CONNECT”按钮连接即可。

连接到MongoDB服务器

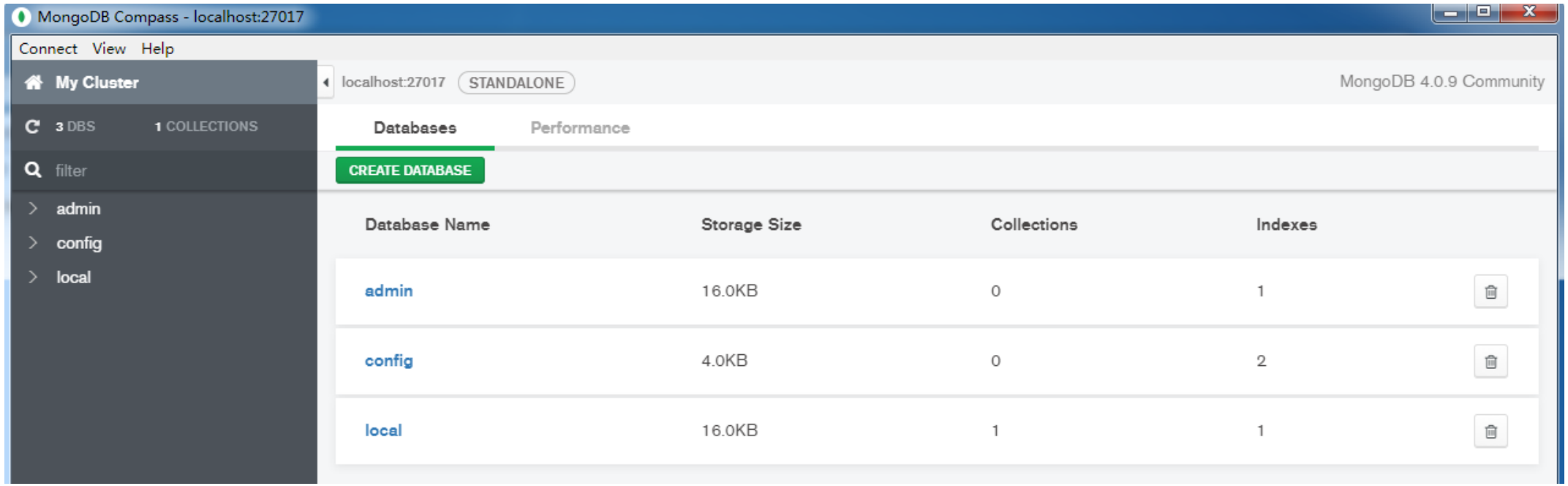


Section 04

【MongoDB的安装和基本使用】

▸▸▸ 安装和使用可视化工具Compass

- 连接到MongoDB之后，即可以使用可视化方法管理和操作MongoDB数据库。



Section 04

【MongoDB的安装和基本使用】

▸▸▸ 使用命令行界面

```
C:\Windows\system32>mongo localhost
MongoDB shell version v4.0.9
connecting to: mongodb://127.0.0.1:27017/localhost?gssapiServiceName=mongodb
(此处省略)
> db          #当前数据库
localhost    #默认数据库
> show dbs     #显示当前的数据库列表
admin     0.000GB
config    0.000GB
local     0.000GB
>
```

▸▸▸ 导出和导入数据

- 导出为JSON格式的命令行用法：

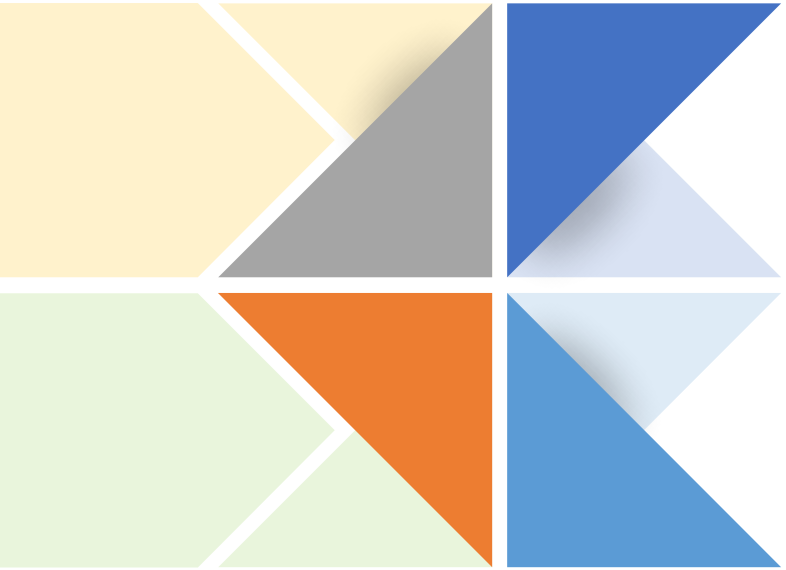
  mongoexport -d 数据库名 -c 集合名 –o 目的文件名（带路径和扩展名.json）

- 从JSON文件导入的命令行用法：

  mongoimport -d 数据库名 -c 集合名 --file 需要导入的文件

内容导航

Contents



Section 00

【学习目标】

（1）了解MongoDB原生驱动；

（2）使用驱动连接和访问MongoDB数据库。

Section 01

【连接到MongoDB】

▸▸▸ 安装mongodb包

- 原生MongoDB驱动的包名为mongodb。
- 安装该包：

```
cnpm install mongodb --save
```

Section 01

【连接到MongoDB】

▸▸▸ 建立到MongoDB的连接

```
const MongoClient = require('mongodb').MongoClient; //导入模块获取连接客户端
const url = 'mongodb://localhost:27017';   // 连接URL
const dbName = 'testmgdb';  // 数据库名称
MongoClient.connect(url, {useNewUrlParser: true}, function(err, client) {
  if (err) throw err;
  console.log("成功连接到MongoDB服务器");
  const db = client.db(dbName); //选择一个数据库
  client.close();   //关闭连接
});
```

Section 02

【添加MongoDB文档】

▸▸▸ 往集合中添加单个文档

```
const MongoClient = require('mongodb').MongoClient;
const url = 'mongodb://localhost:27017';
const dbName = 'testmgdb';
//连接到MongoDB
MongoClient.connect(url, {useNewUrlParser: true},function(err, client) {
  if (err) throw err;
  const db = client.db(dbName);  //选择数据库
  const coll = db.collection('bookinfo'); // 选择一个集合(表)
//执行添加文档操作
  const myobj = { "isbn":"9787115474582","name":"Docker实践","author":"尹恩•米尔
","press":"人民邮电出版社","price":79.00,"pubdate":"2018-02-01"};
  col.insertOne(myobj, function(err, res) {
    if (err) throw err;
    console.log("文档添加成功");
    console.log(res); //输出结果
    client.close();  //关闭连接
  });
});
```

Section 02

【添加MongoDB文档】

▸▸▸ 往集合中添加多个文档

```
const myobj = [{"isbn":"9787115428028","name":"Python编程从入门到实践","author":"袁国忠","press":"人民邮电出版社","price":"89.00","pubdate":"2016-07-01"},
{"isbn":"97871153735557","name":"数学之美（第二版）","author":"吴军","press":"人民邮电出版社","price":"49.00","pubdate":"2014-11-01"}];
db.collection('bookinfo').insertMany(myobj, function(err, res) {
    if (err) throw err;
    console.log("插入的文档数为: " + res.insertedCount);
});
```

Section 03

【查询MongoDB文档】

▸▸▸ 基本查询

- 基本查询示例：

```
db.collection('bookinfo').find({}).toArray(function(err, docs) {
  if (err) throw err;
  console.log("找到下列记录：");
  console.log(docs);
  client.close();
});
```

- 条件查询示例：

```
var filterStr = {"press": '人民邮电出版社'};  // 查询条件
db.collection('bookinfo').find(filterStr).toArray(function(err, docs) {
  if (err) throw err;
  console.log("找到下列记录：");
  console.log(docs);
  client.close();
});
```

Section 03

【查询MongoDB文档】

▸▸▸ 使用查询操作符

- 示例：查找出age大于30的文档

```
{ "age": { "$gt": 30 }}
```

- 示例：复杂的组合查询

```
{ $and: [ { $or: [{a: 1}, {b: 1}] }, { $or: [{c: 1}, {d: 1}] } ]}
```

▸▸▸ 返回指定的查询字段

- 示例

```
find({},{ projection:{'isbn':1,'name':1}})
```

▸▸▸ 查询结果排序

- 示例

```
find({},{projection:{'isbn':1,'name':1},sort:{'isbn':-1}})
```

Section 03

【查询MongoDB文档】

▸▸▸ 查询结果分页

- 使用limit选项限制返回的文档数

```
find({},{limit:2})
```

- 使用skip选项指定跳过的文档数

```
find({},{ skip:2 , limit:2 })
```

- 组合使用skip选项和limit选项可以实现查询结果分页

```
var pageSize = 5;          //每页数据条数
var currentPage = 2;       //当前页数
var skipNum = {currentPage -1} * pageSize;   //跳过条数
find({},{ skip: skipNum , limit: pageSize })
```

Section 04

【更改MongoDB文档】

- 使用updateMany()方法来更改符合条件的文档，用法如下：

  updateMany(filter, update, options, callback)

- 示例

```
var filterStr = { press: '人民邮电出版社'};  // 更改条件
 var updateStr = { $set  : { "url" : "http://www.ptress.com.cn" }};
 db.collection('bookinfo').updateMany(filterStr, updateStr,function(err, docs) {
   if (err) throw err;
   console.log("更改文档数为：  "+ docs.result.nModified);
   client.close();
 });
```

Section 05

【删除MongoDB文档】

- 删除一个文档可使用deleteOne()方法。
- 删除多个文档可以使用deleteMany()方法。
- 示例

```
var filterStr = {"isbn":"9787301299487"};  // 删除条件
 db.collection('bookinfo').deleteMany(filterStr,function(err, docs) {
   if (err) throw err;
   console.log("删除文档数为：  "+ docs.result.n);
   client.close();
 });
});
```

Section 06

【为MongoDB集合创建索引】

- 使用createIndex()方法创建索引，用法如下：

```
createIndex(fieldOrSpec, options, callback)
```

- 示例

```
db.collection('bookinfo').createIndex({ "isbn": 1 }, null,function(err, results) {
    if (err) throw err;
    console.log(results);
    client.close();
  });
});
```

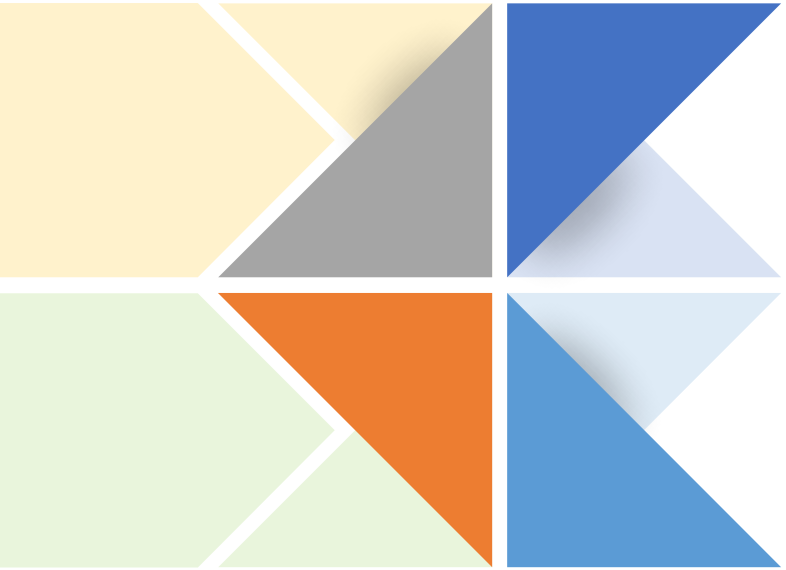
Section 07

【以“同步”方式编写MongoDB操作代码】

```
const MongoClient = require('mongodb').MongoClient;
const url = 'mongodb://localhost:27017';  // 连接URL
const dbName = 'testmgdb';    // 数据库名称
const collName = 'bookinfo';    // 集合名称
async function getConnect() {   //声明建立连接的async函数
    try {
        let connect = await MongoClient.connect(url, {useNewUrlParser: true});
        return connect;
    } catch (err) {
        throw err;
    }
}
async function getBookInfo() {    //声明查询操作的async函数
    // “顺序”执行以下步骤
    let connect = await getConnect();  //执行建立连接的async函数
    let coll = connect.db(dbName).collection(collName);  //得到集合对象
    let result = await coll.find({}).toArray();  //查询文档
    connect.close();   //关闭连接
    return result;    //返回查询结果（这是Promise对象）
  }
getBookInfo().then(data => {   //执行查询操作async函数，并继续处理Promise对象
    console.log("查询的数据:", JSON.stringify(data, null, 4)); // 获取返回的内容
}).catch(error => {
    console.log(error);   // 捕获错误
});
```

内容导航

Contents



使用原生驱动连接和操作MongoDB数据库

Section 00

【学习目标】

（1）了解Mongoose对象模型库；

（2）掌握使用Mongoose操作MongoDB数据库的用法。

Mongoose是针对Node.js异步环境为MongoDB数据库提供的对象模型库。Mongoose基于官方的原生MongoDB驱动node- mongodb-native实现，封装了MongoDB数据的常用操作方法，使Node.js程序操作MongoDB数据库变得更加灵活简单。

Section 01

【Mongoose基本概念】

▸▸▸ 模式

- 每个模式映射到一个MongoDB集合，定义该集合的结构（类似于SQL数据库的表结构）。
- 模式是一种以文件形式存储的模板，仅仅是集合的模型骨架。

▸▸▸ 模型

- 模型是基于模式定义构建的，封装了数据属性和行为的类。
- 模式是静态的定义，必须将模式编译为模型才能用于数据访问。

▸▸▸ 实例

- 实例是指模型（类）的实例，类似SQL数据库的记录，是由模型创建的实体。
- 每个MongoDB文档就是一个实例，Mongoose实现了实例对MongoDB文档的一一映射。
- 模式和模型是定义部分，而实例是模型实例化后创建的对象，是真正要操作的对象。

Section 02

【使用Mongoose的基本步骤】

▸▸▸ 安装包并导入相应模块

- 安装Mongoose

```
cnpm install mongoose  --save
```

- 在程序中导入该模块

```
const mongoose = require('mongoose');
```

▸▸▸ 建立到MongoDB的连接

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://127.0.0.1:27017/newtest', {useNewUrlParser: true, useUnifiedTopology: true });
const conn = mongoose.connection;
conn.on('error', function(error){
    console.log('数据库连接失败：'+error);
});
conn.once('open', function() {
    console.log('数据库连接成功');
});
```

Section 02

【使用Mongoose的基本步骤】

▸▸▸ 定义模式

- 定义模式主要指定对应的MongoDB集合的字段和字段类型。
- 文档中每个属性的类型会被转换为模式定义对应的模式类型（SchemaType）。

```
var UserSchema = new mongoose.Schema({
    name: String,
    pwd: String,
    age: Number,
    date:{type: Date,default: Date.now }
});
```

- 可以直接声明模式类型为某一种类型，或者使用含有type属性的对象来声明类型。
- 可以指定的其他属性还有验证器、索引等。

Section 02

【使用Mongoose的基本步骤】

▸▸▸ 创建模型

- 创建模型是将模式编译成模型，使用mongoose.model(modelName, schema)方法基于模式定义生成一个模型类，以对应于MongoDB集合。

```
const User = mongoose.model('User', UserSchema);
```

- 直到模型所使用的数据库连接打开，集合才会被创建或删除。
- 每个集合都有一个绑定的连接。如果模型是通过调用mongoose.model()方法生成的，它使用的是Mongoose的默认连接。
- 自行创建的连接需要使用连接对象的model()函数来代替mongoose.model()方法，例如：

```
var conn = mongoose.createConnection('mongodb://localhost:27017/test');
var User = conn.model('User', UserSchema);
```

Section 02

【使用Mongoose的基本步骤】

▸▸▸ 实例化模型并执行数据操作

- 实例化模型，即创建实体。

```
var user = new User({
    name: '王强',
    pwd: '123456',
    age:32
});
```

- 进行实例操作，调用save()方法在数据库集合中存入一个文档。

```
 //保存数据
user.save(function(err, doc){
    if (err) {
      console.log('save error:' + err);
    }
    console.log('save sucess \n' + doc);
    //查找数据进行验证
    Model.find({name:'laowang'},(err,doc)=>{
        console.log(doc);
    })
});
```

Section 02

【使用Mongoose的基本步骤】

▶▶▶ 测试

程序代码

```
const mongoose = require('mongoose');
//建立连接
mongoose.connect('mongodb://127.0.0.1:27017/newtest',
{useNewUrlParser: true, useUnifiedTopology: true });
const db = mongoose.connection;
……
//定义模式
var UserSchema = new mongoose.Schema({
……
});
// 创建模型
var User = mongoose.model('User', UserSchema);
// 实例化模型
var user = new User({
……
});
//添加数据
user.save(function(err, doc){
……
});
```

执行结果

```
C:\nodeapp\ch07\mongoosetest>node mgsabc.js
数据库连接成功
save sucess
{ _id: 5ce6a84dfcc433045c75828b,
  name: '王强',
  pwd: '123456',
  age: 32,
  date: 2019-05-23T14:03:57.828Z,
  __v: 0 }
[ { _id: 5ce6a6fc010d7b117cc5530a,
    name: '王强',
    pwd: '123456',
    age: 32,
    date: 2019-05-23T13:58:20.864Z,
    __v: 0 }]
```

Section 03 【文档操作】

创建文档

- 使用save()方法

```
var User = mongoose.model('User', UserSchema);
var laoli = new User({name: 'laoli', pwd: '673456',age: 30 });
laoli.save(function (err) {
  if (err) return handleError(err);
});
```

- 改用模型的静态方法create()，其用法如下：

```
Model.create(docs, [options], [callback])
```

- 示例

```
User.create({name: 'laoli', pwd: '673456',age: 30 }, function (err, laoli) {
  if (err) return handleError(err);
})
```

- 改用静态方法insertMany()一次性添加多个文档，例如：

```
User.insertMany([{name: 'laoli', pwd: '673456',age: 30 }], function(err) {
});
```

Section 03 【文档操作】

▸▸▸ 查询文档

➢ 执行查询的方式

- 传入回调函数参数，Mongoose立即执行查询，并将查询结果传给回调函数。

```
var User = mongoose.model('User', UserSchema);
//查找name匹配'laoli'的文档，输出name和age字段
User.findOne({'name': 'laoli' },'name age',function(err, user){
  if (err) return handleError(err);
  console.log(user);
});
```

- 使用查询构建器，不传递回调函数参数，返回一个Query实例（对象）。

```
var query = User.findOne({ 'name': 'loali' });
query.select('name age');
query.exec(function (err, user) {  //执行查询操作
  if (err) return handleError(err);
  console.log(user);
});
```

Section 03 【文档操作】

查询文档

- Query对象的then()方法

```
const q = MyModel.updateMany({}, { isDeleted: true }, function() {
  console.log('Update 1');
});
q.then(() => console.log('Update 2'));
q.then(() => console.log('Update 3'));
```

- 流式处理MongoDB查询结果

```
var cursor = User.find({ age: 30 }).cursor();
cursor.on('data', function(doc) {
  // 对每个文档调用一次
});
cursor.on('close', function() {
  // 完成时调用
});
```

Section 03 【文档操作】

更改文档

- 使用findById()方法配合save()方法

```
User.findById(id, function (err, user) {
  if (err) return handleError(err);
  user.age = 30;
  user.save(function (err, updatedUser) {
    if (err) return handleError(err);
    console.log(updatedUser);
  });
});
```

- 使用模型的静态方法update()：

```
User.update({ _id: id }, { $set: { age: 30 }}, callback);
```

- 静态方法updateMany()可以修改多个文档，语法格式如下：

```
Model.updateMany()([conditions], [update], [options], [callback])
```

Section 03 【文档操作】

删除文档

- 模型的remove()静态方法

```
User.remove({ age: 32 }, function (err) {
  if (err) return handleError(err);
});
```

覆盖文档

- 用set()方法覆盖整个文档

```
User.findById(id, function (err, user) {
  if (err) return handleError(err);
  // 'otherUser'成为'user'的拷贝
  otherUser.set(user);
});
```

Section 03 【文档操作】

▸▸▸ 自定义文档操作方法

➢ 自定义实例方法

```
var animalSchema = new Schema({                    //定义关于动物的模式
    name: String, type: String
});
animalSchema.methods.findSimilarTypes = function (cb) {//为实例定义查找类型相同的方法
    return this.model('Animal').find({ type: this.type }, cb);
}
```

- 所有的animalSchema实例都可使用findSimilarTypes()方法

```
var AnimalModel = mongoose.model('Animal', animalSechema);//创建模型
var dog = new AnimalModel({ type: '狗狗' });  //实例化模型
dog.findSimilarTypes(function (err, dogs) {  //实例调用findSimilarTypes()方法
    console.log(dogs);
});
```

Section 03 【文档操作】

▸▸▸ 自定义文档操作方法

- 自定义模型静态方法

```
animalSchema.statics.findByName = function (name, cb) { //声明按名查找的静态方法
    this.find({ name: new RegExp(name, 'i') }, cb);
}
```

- 使用findByName ()静态方法：

```
var AnimalModel = mongoose.model('Animal', animalSchema); //创建模型
AnimalModel.findByName('海豚', function (err, animals) { //模型调用静态方法
    console.log(animals);
});
```

Section 04

【数据验证】

▶▶▶ 内置的验证器

```
var breakfastSchema = new Schema({   //定义一个关于早餐的模式
  eggs: {
    type: Number,
    min: [6, '鸡蛋太少'],     //最小值验证，低于6将报出'鸡蛋太少'消息
    max: 12                  //最大值验证
  },
  bacon: {
    type: Number,
    required: [true, '为何没有咸肉?']    //必需字段，提供验证的错误信息
  },
  drink: {
    type: String,
    enum: ['咖啡', '茶'],    //枚举范围
    required: function() {
      return this.bacon > 3;       //定制返回值(根据bacon的数量)确定是否必需字段    }
  }
});
```

Section 04 【数据验证】

自定义验证器

```
var userSchema = new Schema({   //定义一个关于用户的模式
  phone: {
    type: String,
    validate: {      //自定义验证器检查电话号码
      validator: function(v) {
        return /\d{3}-\d{3}-\d{4}/.test(v);
      },
      message: props => '${props.value} 不是有效的电话号码!'
    },
    required: [true, '要求提供用户的电话号码']
  }
});
```

Section 04 【数据验证】

▶▶▶ 验证错误信息

```
var toySchema = new Schema({   //定义一个关于玩具的模式
  color: String,
  name: String
});
var validator = function(value) {  //定义一个玩具颜色的验证器函数
  return /red|white|gold/i.test(value);
};
//引用验证器函数检查玩具颜色
toySchema.path('color').validate(validator,'颜色 '{VALUE}' 是无效的', '无效颜色');
toySchema.path('name').validate(function(v) { //匿名的验证器函数用于检查玩具品名
  if (v !== 'Turbo Man') {
    throw new Error('需要的是奥特曼');
  }
  return true;
}, '品名 '{VALUE}' 不合格');
```

Section 05 【中间件】

▸▸▸ 中间件类型

- 文档中间件，this指向当前文档。
- 查询中间件，this指向当前查询对象。
- 聚合中间件，this指向当前聚合对象，仅支持aggregate操作。
- 模型中间件，this指向当前模型。此类中间件支持的模型操作只有insertMany()。

所有类型的中间件都支持pre和post钩子。

中间件在模式级别定义。

Section 05 【中间件】

pre钩子

- pre钩子在指定方法执行之前绑定。当每个中间件调用next()方法时，pre中间件函数会依次执行

```
var schema = new Schema(..);
schema.pre('save', function(next) {
  // 执行任务
  next();
});
```

- 可以返回一个Promise对象：

```
schema.pre('save', function() {
  return doStuff().
    then(() => doMoreStuff());
});
```

- 可以使用async/await

```
schema.pre('save', async function() {
  await doStuff();
  await doMoreStuff();
});
```

Section 05 【中间件】

post钩子

- post钩子相当于事件监听的绑定。post中间件会在所有钩子方法及pre中间件执行完毕后执行

```
schema.post('init', function(doc) {
  console.log('%s 已经从数据库初始化', doc._id);
});
schema.post('validate', function(doc) {
  console.log('%s 已被验证（但还没保存）', doc._id);
});
schema.post('save', function(doc) {
  console.log('%s 已经保存', doc._id);
});
schema.post('remove', function(doc) {
  console.log('%s 已被删除', doc._id);
});
```

Section 06 【子文档】

▸▸▸ 概述

- Mongoose子文档有两种不同的概念：子文档数组和单个嵌套子文档。

```
var childSchema = new Schema({ name: 'string' });
var parentSchema = new Schema({
  // 子文档数组
  children: [childSchema],
  // 子文档嵌套
  child: childSchema
});
```

Section 06 【子文档】

什么是子文档

- 子文档与普通文档类似，其嵌套模式可以有自己的中间件、自定义验证逻辑、虚拟属性以及其他顶层模式可用的特性。
- 子文档不会单独保存，会随顶级父文档的保存而保存。
- 子文档跟普通文档一样具有save()和validate()中间件。

```
var Parent = mongoose.model('Parent', parentSchema);
var parent = new Parent({ children: [{ name: 'Matt' }, { name: 'Sarah' }] })
parent.children[0].name = 'Matthew';
// parent.children[0].save()不会操作，虽然它触发了中间件，但实际上没有保存文档
// 需要执行保存父文档的操作
parent.save(callback);
```

Section 06 【子文档】

▸▸▸ 操作子文档

- 每个子文档都有一个_id。Mongoose文档数组具有特殊的id()方法，用于查询文档数组以寻找具有指定_id的文档：

```
var doc = parent.children.id(_id);
```

- 可将子文档添加到数组中。

```
var Parent = mongoose.model('Parent');
var parent = new Parent;
// 添加一个评论
parent.children.push({ name: 'Liesl' });
var subdoc = parent.children[0];
console.log(subdoc) // { _id: '501d86090d371bab2c0341c5', name: 'Liesl' }
subdoc.isNew; // true
parent.save(function (err) {
  if (err) return handleError(err)
  console.log('成功!');
});
```

Section 06 【子文档】

操作子文档

- 使用Mongoose数组的create()方法创建子文档，不会将子文档添加到数组中：

```
var newdoc = parent.children.create({ name: 'Aaron' });
```

- 使用对象数组创建模式，Mongoose会自动将对象转换为模式。

```
var parentSchema = new Schema({
  children: [{ name: 'string' }]
});
// 以上代码的作用等同于以下代码
var parentSchema = new Schema({
  children: [new Schema({ name: 'string' })]
});
```

Section 07

【Mongoose对Promise的支持】

内置的Promise

- Mongoose异步操作，如save()方法和查询，都会返回thenable（带有then()方法的对象）。

```
var promise = user1.save();
promise.then(function (doc) {
  console.log(doc);
});
```

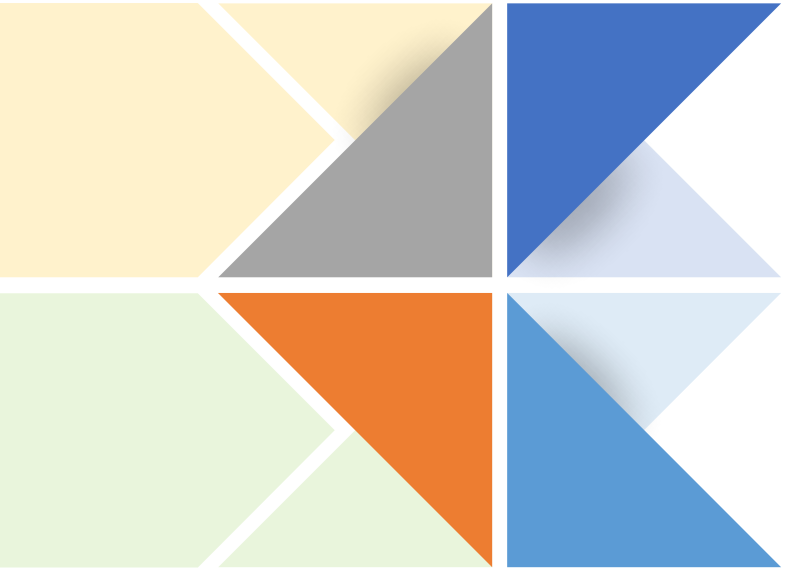
Query对象并非Promise

- 需要完全的Promise对象，则应当使用exec()方法。

```
var query = User.findOne({name: "laoli"});
// query不是完全的Promise，但它有一个.then()方法
query.then(function (doc) {
  // 使用doc
});
// .exec()方法给出完全的Promise
var promise = query.exec();
promise.then(function (doc) {
  // 使用doc
});
```

内容导航

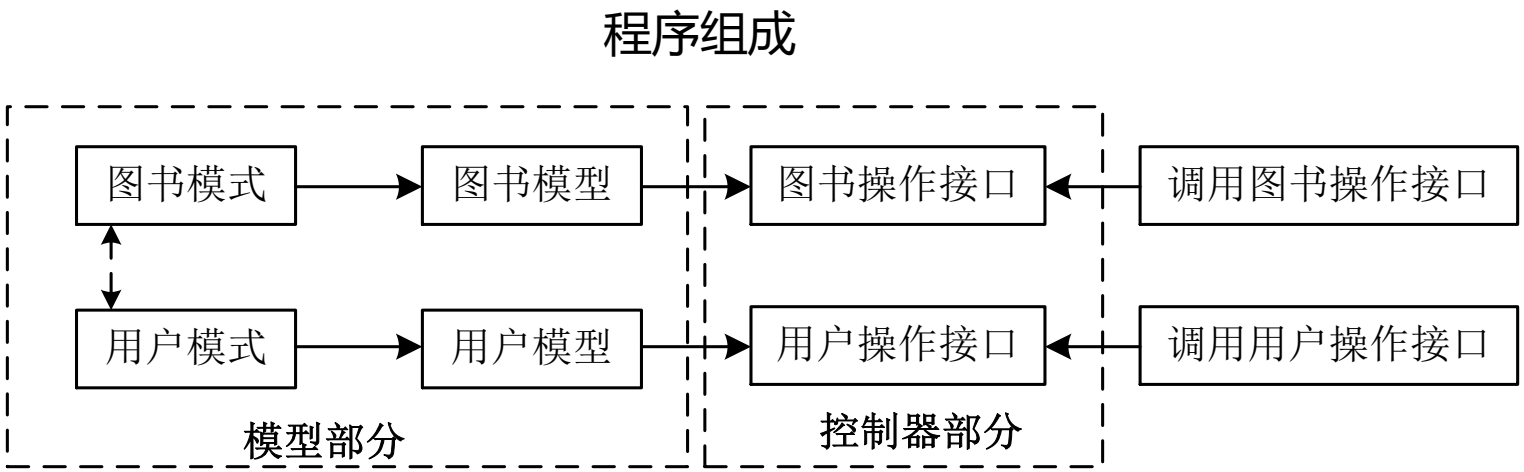
Contents



Section 00

【学习目标】

（1）了解Mongoose的填充功能；
（2）综合运用所学的Mongoose知识；
（3）利用Mongoose封装MongoDB数据库操作接口。



Section 01

【Mongoose的填充功能】

▸▸▸ 理解填充技术

- Mongoose提供填充（Population）功能来实现类似关系数据库中的“连接查询”功能。
- 填充是使用其他集合中的文档自动替换文档中指定路径的过程。
- 可以填充单个文档、多个文档、单个普通对象、多个普通对象，或者从查询返回的所有对象。
- 填充的实现涉及模式定义、文档保存和查询填充等阶段。

Section 01

【Mongoose的填充功能】

▸▸▸ 模式定义时使用ref选项定义引用字段

- 填充用到的字段可称为引用字段，使用ref选项来引用另一个集合的文档。

```
var personSchema = Schema({   //Person（人员）模式
  _id: Schema.Types.ObjectId,
  name: String,
  age: Number
});
var storySchema = Schema({   //Story（小说）模式
  author: { type: Schema.Types.ObjectId, ref: 'Person' },
  title: String,
  fans: [{ type: Schema.Types.ObjectId, ref: 'Person' }]
});
```

Section 01

【Mongoose的填充功能】

▸▸▸ 保存引用字段

➢ 将引用字段保存到其他文档的方法与正常属性保存相同，只是需要指定_id值。

```
const author = new Person({
  ......
});
author.save(function (err) {
  const story1 = new Story({
    title: 'Casino Royale',
    author: author._id    // 设置来自Person实例的_id
  });
  story1.save(function (err) {
    ......
  });
});
```

Section 01

【Mongoose的填充功能】

▸▸▸ 查询时使用填充

- 查询时使用Model.populate()方法即可实现填充：

  Model.populate(docs, [options], [callback])

- 可以仅指定要填充的文档：

```
Story.findOne({ title: 'Casino Royale' }).populate('author')
.exec(function (err, story) {  ..... });
```

- 只想返回填充的文档的某些字段

```
populate('author', 'name'). // 仅返回 Person 的name字段
```

Section 02

【模式和模型定义】

- bookinfo集合的模型定义

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
var BookSchema = new Schema({
    isbn:String,
    name:String,
    author: String,
    press: String,
    price: Number,
    pubdate: Date,
    user: { type: Schema.ObjectId, ref: 'User' }
});
//定义模型时加上MongoDB集合名参数
module.exports = mongoose.model('Book',BookSchema,'bookinfo');
```

Section 02

【模式和模型定义】

- user集合的模型定义

```
const mongoose = require('mongoose');
var UserSchema = new mongoose.Schema({
    name: String,
    pwd: String,
    age: {type:Number,  max: 60},
    date:{type: Date,default: Date.now },
    book: { type: Schema.ObjectId, ref: 'Book' }
});
module.exports = mongoose.model('User',UserSchema);
```

Section 03

【编写数据库操作接口】

```
const co = require('co');  //导入co模块
const Book = require("./book_model.js");
const User = require("./user_model.js");
//添加一个文档，两个参数可以是对象形式表示的Book和User文档内容
exports.add = co.wrap(function*(book,user) {
    //如果提供有用户信息，则将其_id值保存到Book的引用字段user
    if (user!=undefined ) {
        var u = yield User.create(user);
        book['user'] = u._id;
    }
    return  yield Book.create(book);
});
//查询文档，参数是对象形式表示的查询条件
exports.find = co.wrap(function*(cond) {
    //使用User文档的name和age字段来填充Book文档的user字段
    return yield Book.find(cond).populate('user','name  age').exec();
});
```

Section 03

【编写数据库操作接口】

```
/** 查询结果分页,第1个参数是对象形式表示的查询条件
第2个参数为每页大小，第3个参数为当前页 */
exports.findPaged = co.wrap(function*(cond,pageSize,currentPage){
    var skipnum = (currentPage - 1) * pageSize;   //跳过文档数
 return yield Book.find(cond).skip(skipnum).limit(pageSize).populate('user'). exec();
});
//更改文档，两个参数分别是对象形式表示的修改条件和修改的文档内容
exports.update = co.wrap(function*(cond, upd) {
    return  yield Book.updateOne(cond, upd).exec();
});
//删除文档，参数是对象形式表示的删除条件
exports.del = co.wrap(function*(cond) {
    return  yield Book.deleteOne(cond).exec();;
});
```

Section 04

【调用数据库操作接口】

```
const mongoose = require('mongoose');
const co = require('co');
const bookcrud = require('./book_controller.js');
const uri = 'mongodb://127.0.0.1:27017/testmgdb';
const options = {  autoIndex: false,   useNewUrlParser: true, useUnifiedTopology: true, poolSize: 10 };
co(function * () {   //声明一个co-generator函数
    //建立连接
    yield mongoose.connect(uri, options);
    //添加文档
    const book = { "isbn":"9787115474582","name":"Docker实践","author":"尹恩•米尔","press":"人民邮电出版社","price":79.00,"pubdate":"2018-02-01"};
    const user = {name: 'laoli', pwd: '673456',age: 30 };
    var newDoc = yield bookcrud.add(book,user).then(function (value) {
        return value;
    }) ;
```

Section 04

【调用数据库操作接口】

```
//查询刚添加的文档
    yield bookcrud.find({_id:newDoc._id}).then(function (value) {
        console.log(value);
    }) ;
    //查询所有文档并分页显示
    yield bookcrud.findPaged({},2,2).then(function (value) {
        console.log('每页2个文档，第2页图书信息：'+value);
    }) ;
    //更改文档
     yield bookcrud.update({_id:newDoc._id},{ $set: { price: 57.80 }}).then(function (value) {
        console.log(value);
    }) ;
    //删除文档
    yield bookcrud.del({_id:newDoc._id},{ $set: { price: 57.80 }}).then(function (value) {
        console.log(value);
    }) ;
    //关闭连接
    yield mongoose.connection.close();
});
```

本章小结

Summary

本章的主要内容是编写Node.js程序操作NoSQL数据库MongoDB。MongoDB是一个对象数据库，适合存储网站内容、缓存等大尺寸、低价值的数据。掌握使用原生驱动连接和操作MongoDB数据库的方法是有必要的，这有利于熟悉MongoDB基本操作。为提高开发效率，一般会选用更加灵活简单的Mongoose，它是封装了MongoDB操作的一个对象模型库。掌握MongoDB之后，其他NoSQL数据库操作也会触类旁通。

Thank you