持久化之mongodb

回顾

- 数据持久化之关系型数据库mysql应用
- 原生mysql驱动node-mysql应用
- ORM模块Sequelize的应用

课堂目标

- 掌握mongodb基本使用
- 理解文档型数据库设计理念
- 掌握原生模块node-mongodb-native应用
- 掌握ODM模块mongoose应用

资源

- mongodb相关：
  - MongoDB：下载
  - node驱动：文档
  - mongoose：文档
- 可视化工具：Robo3T

mongodb安装、配置

- 下载安装
- 配置环境变量
- 创建dbpath文件夹
- 启动：`mongod --dbpath=/data`
- 测试：

```
// 查询所有数据库
show dbs

// 切换/创建数据库,当创建一个集合(table)的时候会自动创建当前数据库
use test

// 得到当前db的所有聚集集合
db.getCollectionNames()

// 查询
db.fruits.find()

// 插入一条数据
db.fruits.save({name:'苹果',price:5})

// 条件查询
db.fruits.find({price:5})
db.fruits.find({price: {$lte: 10}})
```

mongo命令行操作

mongodb原生驱动

- 安装mysql模块：`npm install mongodb --save`
- 连接mongodb

```
// 客户端
const MongoClient = require("mongodb").MongoClient;

// 连接URL
const url = "mongodb://localhost:27017";

// 数据库名
const dbName = "test";

(async function() {
  // 0.创建客户端
  const client = new MongoClient(url, { useNewUrlParser: true });
  try {
    // 1.连接数据库(异步)
    await client.connect();
    console.log("连接成功");
  } catch (error) {
    console.error(error);
  }

  client.close();
})();
```

- 插入文档

```
    // 2.获取数据库
    const db = client.db(dbName);

    // 3.获取集合
    const fruitsColl = db.collection("fruits");

    // 4.插入文档（异步）
    let r;
    r = await fruitsColl.insertOne({ name: "芒果", price: 20.0 });
    console.log("插入成功", r.result);
```

- 更新文档

```
    // 5.更新文档（异步）
    r = await fruitsColl.updateOne({ name: "芒果" }, { $set: { price: 19.8 } });
    console.log("更新成功", r.result);
```

- 查询文档

```
    // 6.查询文档
    r = await fruitsColl.find().toArray();
    console.log('查询结果', r);
```

- 删除文档

```
    // 7.删除文档
    r = await fruitsColl.deleteMany({price:20});
    console.log('删除成功', r.result);
```

- 操作符
  - 查询操作符：提供多种方式定位数据库数据
    - 比较$eq，$gt，$gte，$in
    - 逻辑$and,$not,$nor,$or
    - 模拟$regex，$text，$expr
    - 元素$exists，$type
    - 数组$all,$elemMatch,$size
    - 地理空间$geoIntersects,$geoWithin,$near,$nearSphere
  - 更新操作符：可以修改数据库数据或添加附加数据
    - 字段相关：$set,$unset,$setOnInsert,$rename,$inc,$min,$max,$mul
    - 数组相关：$,$[],$addToSet,$pull,$pop,$push,$pullAll
    - 修改器，常结合数组操作符使用：$each,$position,$slice,$sort
  - 聚合操作符：使用aggregate方法，使文档顺序通过管道阶段从而得到最终结果
    - 聚合管道阶段：$group,$count,$sort,$skip,$limit,$project等
    - 聚合管道操作符：$add,$avg,$sum等

ODM - Mongoose

- 安装：`npm install mongoose -S`
- 基本使用：

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

// 1.连接
mongoose.connect("mongodb://localhost:27017/test", { useNewUrlParser: true });

const conn = mongoose.connection;
conn.on("error", () => console.error("连接数据库失败"));
conn.once("open", async () => {
  // 2.定义一个Schema - Table
  const Schema = mongoose.Schema({
    category: String,
    name: String
  });

  // 3.编译一个Model，它对应数据库中复数、小写的Collection
  const Model = mongoose.model("fruit", Schema);
  try {
    // 4.创建，create返回Promise
    let r = await Model.create({
      category: "温带水果",
      name: "苹果",
      price: 5
    });
    console.log("插入数据:", r);

    // 5.查询，find返回Query，它实现了then和catch，可以当Promise使用
    // 如果需要返回Promise，调用其exec()
    r = await Model.find({ name: "苹果" });
    console.log("查询结果:", r);

    // 6.更新，updateOne返回Query
    r = await Model.updateOne({ name: "苹果" }, { $set: { name: '芒果' } });
    console.log("更新结果：", r);

    // 7.删除，deleteOne返回Query
    r = await Model.deleteOne({ name: "苹果" });
    console.log("删除结果：", r);
  } catch (error) {
    console.log(error);
  }
});
```

- Schema
  - 字段定义

```
const blogSchema = mongoose.Schema({
```

```
  title: { type: String, required: [true, '标题为必填项'] }, // 定义校验规则
  author: String,
  body: String,
  comments: [{ body: String, date: Date }], // 定义对象数组
  date: { type: Date, default: Date.now }, // 指定默认值
  hidden: Boolean,
  meta: {
    // 定义对象
    votes: Number,
    favs: Number
  }
});
// 定义多个索引
blogSchema.index({ title:1, author: 1, date: -1 });
const BlogModel = mongoose.model("blog", blogSchema);
const blog = new BlogModel({
  title: "nodejs持久化",
  author: "jerry",
  body: "...."
});
const r = await blog.save();
console.log("新增blog", r);
```

- 定义实例方法：抽象出常用方法便于复用

```
// 定义实例方法
blogSchema.methods.findByAuthor = function () {
    return this.model('blog').find({ author: this.author }).exec();
}

// 获得模型实例
const BlogModel = mongoose.model("blog", blogSchema);
const blog = new BlogModel({...});

// 调用实例方法
r = await blog.findByAuthor();
console.log('findByAuthor', r);
```

- 静态方法

```
blogSchema.statics.findByAuthor = function(author) {
  return this.model("blog")
    .find({ author })
    .exec();
};

r=await BlogModel.findByAuthor('jerry')
console.log("findByAuthor", r);
```

- 虚拟属性

```
blogSchema.virtual("commentsCount").get(function() {
  return this.comments.length;
});
let r = await blog.findOne({author:'jerry'});
console.log("blog留言数: ", r.commentsCount);
```

购物车相关接口实现