

持久化之mongodb

资源

- mongodb相关：
 - MongoDB: [下载](#)
 - node驱动: [文档](#)
 - mongoose: [文档](#)
- redis相关：
 - redis: [下载](#)
 - node_redis: [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:

```
mongo
// 默认连接
```

- 测试:

```
// helloworld.js
// 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

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

// 条件查询
db.fruits.find({price: 5})
`1234`

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

// 查询
```

```
db.fruits.find()
```

[mongo命令行操作](#)

参考资料

菜鸟文档

<http://www.runoob.com/mongodb/mongodb-create-database.html>

官网

<https://docs.mongodb.com/manual/reference/method/>

mongodb原生驱动

<http://mongodb.github.io/node-mongodb-native/3.1/quick-start/quick-start/>

官网API

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

```
(async () => {
  const { MongoClient: MongoDB } = require('mongodb')

  // 创建客户端
  const client = new MongoDB(
    'mongodb://localhost:27017',
    {
      useNewUrlParser: true
    }
  )
  let ret
  // 创建连接
  ret = await client.connect()
  console.log('ret:', ret)

  const db = client.db('test')

  const fruits = db.collection('fruits')

  // 添加文档
  ret = await fruits.insertOne({
    name: '芒果',
    price: 20.1
  })
  console.log('插入成功', JSON.stringify(ret))
})
```

```

// 查询文档
ret = await fruits.findOne()
console.log('查询文档:', ret)

// 更新文档
// 更新的操作符 $set
ret = await fruits.updateOne({ name: '芒果' },
{ $set: { name: '苹果' } })
console.log('更新文档', JSON.stringify(ret.result))

// 删除文档
ret = await fruits.deleteOne({name: '苹果'})

await fruits.deleteMany()

client.close()

})()

```

- 案例：瓜果超市
 - 提取数据库配置,./models/conf.js

```

// models/conf.js
module.exports = {
  url: "mongodb://localhost:27017",
  dbName: 'test',
}

```

- 封装数据库连接, ./models/db.js

```

const conf = require("./conf");
const EventEmitter = require("events").EventEmitter;

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

class Mongodb {
  constructor(conf) {
    // 保存conf
    this.conf=conf;

    this.emmitter = new EventEmitter();
    // 连接
    this.client = new MongoClient(conf.url, { useNewUrlParser: true });
    this.client.connect(err => {
      if (err) throw err;
      console.log("连接成功");
      this.emmitter.emit("connect");
    });
  }
}

```

```

    });
  }

  col(colName, dbName = conf.dbName) {
    return this.client.db(dbName).collection(colName);
  }

  once(event, cb) {
    this.emitter.once(event, cb);
  }
}

// 2. 导出db
module.exports = new Mongodb(conf);

```

- eventEmmitter

```

// eventEmmitter.js
const EventEmitter = require('events').EventEmitter;
const event = new EventEmitter();
event.on('some_event', num => {
  console.log('some_event 事件触发:' + num);
});
let num = 0
setInterval(() => {
  event.emit('some_event', num ++ );
}, 1000);

```

- 添加测试数据, ./initData.js

```

const mongodb = require('./models/db')
mongodb.once('connect', async () => {
  const col = mongodb.col('fruits')
  // 删除已存在
  await col.deleteMany()
  const data = new Array(100).fill().map((v, i) => {
    return { name: "xxx" + i, price: i, category: Math.random() > 0.5
? '蔬菜' : '水果' }
  })

  // 插入
  await col.insertMany(data)
  console.log("插入测试数据成功")
})

```

- 前端页面调用 index.html

```
```html
<!DOCTYPE html>
<html lang="en">

<head>
 <meta charset="UTF-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 <meta http-equiv="X-UA-Compatible" content="ie=edge" />

 <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
 <script src="https://unpkg.com/element-ui/lib/index.js"></script>
 <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
 <link rel="stylesheet" href="https://unpkg.com/element-ui/lib/theme-
chalk/index.css" />
 <title>瓜果超市</title>
</head>

<body>
 <div id="app">
 <el-input placeholder="请输入内容" v-model="search" class="input-with-
select" @change="changeHandler">
 <el-button slot="append" icon="el-icon-search"></el-button>
 </el-input>
 <el-radio-group v-model="category" @change="getData">
 <el-radio-button v-for="v in categorys" :label="v" :key="v">
{{v}}</el-radio-button>
 </el-radio-group>
 <el-table :data="fruits" style="width: 100%">
 <el-table-column prop="name" label="名称" width="180">
 </el-table-column>
 <el-table-column prop="price" label="价格" width="180">
 </el-table-column>
 <el-table-column prop="category" label="种类">
 </el-table-column>
 </el-table>
 <el-pagination layout="prev, pager, next" @current-
change="currentChange" :total="total">
 </el-pagination>
 </div>
</script>
 var app = new Vue({
 el: "#app",
 data: {
 page: 1,
 total: 0,
 fruits: [],
 categorys: [],
 }
 })

```

```

 category: [],
 search: ''
 },
 created() {
 this.getData()

 this.getCategory()
 },
 methods: {
 async currentChange(page) {
 this.page = page;
 await this.getData()
 },
 async changeHandler(val){
 console.log('search...',val)
 this.search = val
 await this.getData()
 },
 async getData() {
 const res = await axios.get(`/api/list?
page=${this.page}&category=${this.category}&keyword=${this.search}`)
 const data = res.data.data
 this.fruits = data.fruits
 this.total = data.pagination.total
 },
 async getCategory() {
 const res = await axios.get(`/api/category`)
 this.categorys = res.data.data
 console.log('category', this.categorys)
 }
 }
});
</script>
</body>

</html>
...

```

- 接口编写, index.js

```

const express = require("express")
const app = express()
const path = require("path")
const mongo = require("../models/db")
// const testdata = require("../initData")

app.get("/", (req, res) => {
 res.sendFile(path.resolve("../index.html"))
})

```

```

app.get("/api/list", async (req, res) => {
 // 分页查询
 const { page } = req.query
 try {
 const col = mongo.col("fruits")
 const total = await col.find().count()
 const fruits = await col
 .find()
 .skip((page - 1) * 5)
 .limit(5)
 .toArray()
 res.json({ ok: 1, data: { fruits, pagination: { total, page } } })
 } catch (error) {
 console.log(error)
 }
})

app.listen(3000)

```

- 增加类别搜索功能

```

app.get("/api/category", async (req, res) => {
 const col = mongo.col("fruits")
 const data = await col.distinct('category')
 res.json({ ok: 1, data })
})

app.get("/api/list", async (req, res) => {
 // 分页查询
 const { page, category, keyword } = req.query

 // 构造条件
 const condition = {}
 if (category) {
 condition.category = category
 }

 if (keyword) {
 condition.name = { $regex: new RegExp(keyword) }
 }

 // 增加
 const total = await col.find(condition).count()
 const fruits = await col
 .find(condition) // 增加
 .skip((page - 1) * 5)
 .limit(5)
 .toArray()

```

```
})
```

- [操作符](#)

<https://docs.mongodb.com/manual/reference/operator/query/>

操作符文档

- 查询操作符：提供多种方式定位数据库数据

```
// 比较$eq, $gt, $gte, $in等
await col.find({price:{$gt:10}}).toArray()

// 逻辑$and,$not,$nor,$or
// price>10 或 price<5
await col.find({$or: [{price:{$gt:10}}, {price:{$lt:5}}]})
// price不大于10且price不小于5
await col.find({$nor: [{price:{$gt:10}}, {price:{$lt:5}}]})

// 元素$exists, $type
await col.insertOne({ name: "芒果", price: 20.0, stack:true })
await col.find({stack:{$exists:true}})

// 模拟$regex, $text, $expr
await col.find({name:{$regex:/芒/}})
await col.createIndex({name:'text'}) // 验证文本搜索需首先对字段加索引
await col.find({$text:{$search:'芒果'}}) // 按词搜索，单独字查询不出结果

// 数组$all,$elemMatch,$size
col.insertOne({... , tags: ["热带", "甜"]}) // 插入带标签数据
// $all: 查询指定字段包含所有指定内容的文档
await col.find({ tags: {$all:['热带','甜']} })
// $elemMatch: 指定字段数组中至少有一个元素满足所有查询规则
col.insertOne({hisPrice: [20,25,30]}); // 数据准备
col.find({ hisPrice: { $elemMatch: { $gt: 24,$lt:26 } } }) // 历史价位
有没有出现在24~26之间

// 地理空间$geoIntersects,$geoWithin,$near,$nearSphere
// 创建stations集合
const stations = db.collection("stations");
// 添加测试数据，执行一次即可
await stations.insertMany([
 { name: "天安门东", loc: [116.407851, 39.91408] },
 { name: "天安门西", loc: [116.398056, 39.913723] },
 { name: "王府井", loc: [116.417809, 39.91435] }
]);
await stations.createIndex({ loc: "2dsphere" });
```



```

r = await stations.find({
 loc: {
 $nearSphere: {
 $geometry: {
 type: "Point",
 coordinates: [116.403847, 39.915526]
 },
 $maxDistance: 1000
 }
 }
}).toArray();
console.log("天安门附近地铁站", r);

```

- 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
// 更新多个字段
await fruitsColl.updateOne(
 { name: "芒果" },
 { $set: { price: 19.8, category: '热带水果' } },
);
// 更新内嵌字段
{ $set: { ..., area: {city: '三亚'} } }

// 数组相关: $,$[],$addToSet,$pull,$pop,$push,$pullAll
// $push用于新增
insertOne({tags: ['热带','甜']}) //添加tags数组字段
fruitsColl.updateMany({ name: "芒果" }, { $push: {tags: '上火'}})
// $pull,$pullAll用于删除符合条件项, $pop删除首项-1或尾项1
fruitsColl.updateMany({ name: "芒果" }, { $pop: {tags: 1}})
fruitsColl.updateMany({ name: "芒果" }, { $pop: {tags: 1}})
// $, $[]用于修改
fruitsColl.updateMany({ name: "芒果", tags: "甜" }, { $set: {"tags.$": "香甜"} })

// 修改器, 常结合数组操作符使用: $each,$position,$slice,$sort
$push: { tags: { $each: ["上火", "真香"], $slice: -3 } }

```

- 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
// 分页查询
r = await fruitsColl
 .aggregate([{$sort: { price: -1 } }, { $skip: 0 }, { $limit: 2
}])
 .toArray();
// 投射:只选择name,price并排除_id
fruitsColl.aggregate([..., {$project:
{name:1,price:1,_id:0}}]).toArray();

// 聚合管道操作符: $add,$avg,$sum等
// 按name字段分组, 统计组内price总和
fruitsColl.aggregate([{$group: {_id: "$name", total:
{$sum: "$price"}}}]).toArray();
```

## ODM - [Mongoose](#)

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

```
// mongoose.js
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);

```

可选字段类型:

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- ObjectId
- Array

避免创建索引警告：

```
mongoose.connect("mongodb://localhost:27017/test", {
 useCreateIndex: true
})
```

- 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;
});
r = await BlogModel.findOne({author: 'jerry'});
console.log("blog留言数: ", r.commentsCount);
```

## 购物车相关接口实现

- mongoose.js

```
// mongoose.js
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"));
```

- 用户模型, ./models/user.js

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

const schema = mongoose.Schema({
 name: String,
 password: String,
 cart: []
});

schema.statics.getCart = function(_id) {
 return this.model("user")
 .findById(_id)
 .exec();
};

schema.statics.setCart = function(_id, cart) {
 return this.model("user")
 .findByIdAndUpdate(_id, { $set: { cart } })
 .exec();
};

const model = mongoose.model("user", schema);

// 测试数据
model.updateOne(
 { _id: "5c1a2dce951e9160f0d8573b" },
```

```

 { name: "jerry", cart: [{ pname: "iPhone", price: 666, count: 1 }] },
 { upsert: true },
 (err, r) => {
 console.log('测试数据');
 console.log(err, r);
 }
);

 module.exports = model;

```

- API编写, ./index.js

```

// mongoose.js
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"));

```

```

// models/user.js
const mongoose = require("mongoose");

const schema = mongoose.Schema({
 name: String,
 password: String,
 cart: []
});

schema.statics.getCart = function(_id) {
 return this.model("user")
 .findById(_id)
 .exec();
};

schema.statics.setCart = function(_id, cart) {
 return this.model("user")
 .findByIdAndUpdate(_id, { $set: { cart } })
 .exec();
};

const model = mongoose.model("user", schema);

// 测试数据
model.updateOne(
 { _id: "5c1a2dce951e9160f0d8573b" },
 { name: "jerry", cart: [{ pname: "iPhone", price: 666, count: 1 }] },

```

```

 { upsert: true },
 (err, r) => {
 console.log('测试数据');
 console.log(err, r);
 }
);

module.exports = model;

```

```

// index.js
const express=require('express')
const app=new express();
const bodyParser = require('body-parser');
const path = require('path')

// 数据库相关
require('./mongoose')
const UserModel = require('./models/user')

// mock session
const session = {sid:{userId:'5c1a2dce951e9160f0d8573b'}}

app.use(bodyParser.json());
app.get("/", (req, res) => {
 res.sendFile(path.resolve("./index.html"))
})
// 查询购物车数据
app.get('/api/cart', async (req,res)=>{
 const data = await UserModel.getCart(session.sid.userId)
 res.send({ok:1, data})
})

// 设置购物车数据
app.post('/api/cart', async (req,res)=>{
 await UserModel.setCart(session.sid.userId, req.body.cart)
 res.send({ok:1})
})

app.listen(3000);

```

index.html

```

<!DOCTYPE html>
<html lang="en">

<head>

```

```

<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta http-equiv="X-UA-Compatible" content="ie=edge" />
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script src="https://unpkg.com/element-ui/lib/index.js"></script>
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<link rel="stylesheet" href="https://unpkg.com/element-ui/lib/theme-
chalk/index.css" />
<title>瓜果超市</title>
</head>
<body>
 <div id="app">
 <el-button @click='getCart'>getCart</el-button>
 <el-button @click='setCart'>setCart</el-button>
 </div>
 <script>
 var app = new Vue({
 el: "#app",
 methods: {
 async getCart(page) {
 const ret = await axios.get('/api/cart')
 console.log('ret:', ret.data.data)
 },
 async setCart() {
 const ret = await axios.post(
 '/api/cart', {
 cart:[
 {
 name:'菠萝',
 count:1
 }
]
 }
)
 }
 }
 });
 </script>
</body>

</html>

```