数据持久化 - MySQL

回顾

- 模块系统
- 全局变量
- 核心API

课堂目标

- 掌握node.js中实现持久化的多种方法
- 掌握mysql下载、安装和配置
- 掌握node.js中原生mysql驱动模块的应用
- 掌握node.js中的ORM模块Sequelize的应用
- 实现商城案例中所需接口

资源

- MySQL相关：
  - MySQL：下载
  - node驱动：文档
  - Sequelize：文档、api
- mongodb相关：
  - MongoDB：下载
  - node驱动：文档
  - mongoose：文档
- redis相关：
  - redis：下载
  - node_redis：文档

node.js中实现持久化的多种方法

- 文件系统 fs
- 数据库
  - 关系型数据库-mysql
  - 文档型数据库-mongodb
  - 键值对数据库-redis

文件系统数据库

```
// 实现一个文件系统读写数据库
const fs = require("fs");

function get(key) {
  fs.readFile("./db.json", (err, data) => {
    const json = JSON.parse(data);
    console.log(json[key]);
  });
}
function set(key, value) {
  fs.readFile("./db.json", (err, data) => {
    // 可能是空文件，则设置为空对象
    const json = data ? JSON.parse(data) : {};
    json[key] = value; // 设置值
    // 重新写入文件
    fs.writeFile("./db.json", JSON.stringify(json), err => {
      if (err) {
        console.log(err);
      }
      console.log("写入成功！");
    });
  });
}

// 命令行接口部分
const readline = require("readline");
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

rl.on("line", function(input) {
  const [op, key, value] = input.split(" ");

  if (op === 'get') {
    get(key)
  } else if (op === 'set') {
    set(key, value)
  } else if(op === 'quit'){
```

```
    rl.close();
  }else {
    console.log('没有该操作');
  }
});

rl.on("close", function() {
  console.log("程序结束");
  process.exit(0);
});
```

MySQL安装、配置

mac

windows

node.js原生驱动

- 安装mysql模块：`npm i mysql --save`
- mysql模块基本使用

```
const mysql = require("mysql");
// 连接配置
const cfg = {
  host: "localhost",
  user: "root",
  password: "admin", // 修改为你的密码
  database: "kaikeba" // 请确保数据库存在
};
// 创建连接对象
const conn = mysql.createConnection(cfg);

// 连接
conn.connect(err => {
  if (err) {
    throw err;
  } else {
    console.log("连接成功! ");
  }
});

// 查询 conn.query()
// 创建表
const CREATE_SQL = `CREATE TABLE IF NOT EXISTS test (
                    id INT NOT NULL AUTO_INCREMENT,
                    message VARCHAR(45) NULL,
                    PRIMARY KEY (id))`;
```

```
const INSERT_SQL = `INSERT INTO test(message) VALUES(?)`;
const SELECT_SQL = `SELECT * FROM test`;
conn.query(CREATE_SQL, err => {
  if (err) {
    throw err;
  }
  // 插入数据
  conn.query(INSERT_SQL, "hello,world", (err, result) => {
    if (err) {
      throw err;
    }
    console.log(result);
    conn.query(SELECT_SQL, (err, results) => {
        console.log(results);
        conn.end(); // 若query语句有嵌套，则end需在此执行
    })
  });
});
```

Node.js ORM - Sequelize

- 概述：基于Promise的ORM(Object Relation Mapping)，支持多种数据库、事务、关联等
- 安装：`npm i sequelize mysql2 -S`
- 基本使用：

```
const Sequelize = require("sequelize");

// 建立连接
const sequelize = new Sequelize("kaikeba", "root", "admin", {
  host: "localhost",
  dialect: "mysql",
  operatorsAliases: false
});

// 定义模型
const Fruit = sequelize.define("Fruit", {
  name: { type: Sequelize.STRING(20), allowNull: false },
  price: { type: Sequelize.FLOAT, allowNull: false },
  stock: { type: Sequelize.INTEGER, defaultValue: 0 }
});

// 同步数据库，force: true则会删除已存在表
Fruit.sync()
  .then(() => {
    // 添加测试数据
    return Fruit.create({
      name: "香蕉",
      price: 3.5
    });
```

```
  })
  .then(() => {
    // 查询
    Fruit.findAll().then(fruits => {
      console.log(JSON.stringify(fruits));
    });
  });
```

- 强制同步：创建表之前先删除已存在的表

```
Fruit.sync({force: true})
```

- 避免自动生成时间戳字段

```
const Fruit = sequelize.define("Fruit", {}, {
  timestamps: false
});
```

- 指定表名：`freezeTableName: true` 或 `tableName:'xxx'`

> 设置前者则以modelName作为表名；设置后者则按其值作为表名。

- Getters & Setters：可用于定义伪属性或映射到数据库字段的保护属性

```
// 定义为属性的一部分
name: {
      type: Sequelize.STRING,
      allowNull: false,
      get() {
        const fname = this.getDataValue("name");
        const price = this.getDataValue("price");
        const stock = this.getDataValue("stock");
        return `${fname}(价格：￥${price} 库存：${stock}kg)`;
      }
}

// 定义为模型选项
{
    getterMethods:{
      amount(){
        return this.getDataValue("stock") + "kg";
      }
    },
    setterMethods:{
      amount(val){
        const idx = val.indexOf('kg');
        const v = val.slice(0, idx);
        this.setDataValue('stock', v);
      }
    }
  }
```

```
// 通过模型实例触发setterMethods
Fruit.findAll().then(fruits => {
    console.log(JSON.stringify(fruits));
    // 修改amount，触发setterMethods
    fruits[0].amount = '150kg';
    fruits[0].save();
});
```

- 校验：可以通过校验功能验证模型字段格式、内容，校验会在`create`、`update`和`save`时自动运行

```
price: {
    validate: {
        isFloat: { msg: "价格字段请输入数字" },
        min: { args: [0], msg: "价格字段必须大于0" }
    }
},
stock: {
    validate: {
        isNumeric: { msg: "库存字段请输入数字" }
    }
}
```

- 模型扩展：可添加模型实例方法或类方法扩展模型

```
// 添加类级别方法
Fruit.classify = function(name) {
  const tropicFruits = ['香蕉', '芒果', '椰子']; // 热带水果
  return tropicFruits.includes(name) ? '热带水果':'其他水果';
};

// 添加实例级别方法
Fruit.prototype.totalPrice = function(count) {
  return (this.price * count).toFixed(2);
};

// 使用类方法
['香蕉','草莓'].forEach(f => console.log(f+'是'+Fruit.classify(f)));

// 使用实例方法
Fruit.findAll().then(fruits => {
    const [f1] = fruits;
    console.log(`买5kg${f1.name}需要￥${f1.totalPrice(5)}`);
});
```

- 数据查询

```
// 通过id查询
Fruit.findById(1).then(fruit => {
    // fruit是一个Fruit实例，若没有则为null
    console.log(fruit.get());
});
```

```
// 通过属性查询
Fruit.findOne({ where: { name: "香蕉" } }).then(fruit => {
    // fruit是首个匹配项，若没有则为null
    console.log(fruit.get());
});

// 获取数据和总条数
Fruit.findAndCountAll().then(result => {
    console.log(result.count);
    console.log(result.rows.length);
});

// 查询操作符
const Op = Sequelize.Op;
Fruit.findAll({
    // where: { price: { [Op.lt]:4 }, stock: { [Op.gte]: 100 } }
    where: { price: { [Op.lt]:4,[Op.gt]:2 }}
}).then(fruits => {
    console.log(fruits.length);
});

// 或语句
Fruit.findAll({
    // where: { [Op.or]:[{price: { [Op.lt]:4 }}, {stock: { [Op.gte]: 100 }}] }
    where: { price: { [Op.or]:[{[Op.gt]:3 }, {[Op.lt]:2 }]}}
}).then(fruits => {
    console.log(fruits[0].get());
});

// 分页
Fruit.findAll({
    offset: 0,
    limit: 2,
})

// 排序
Fruit.findAll({
    order: [['price', 'DESC']],
})

// 聚合
setTimeout(() => {
  Fruit.max("price").then(max => n
    console.log("max", max);
  });
  Fruit.sum("price").then(sum => {
    console.log("sum", sum);
  });
}, 500);
```

- 更新

```
Fruit.findById(1).then(fruit => {
    // 方式1
    fruit.price = 4;
    fruit.save().then(()=>console.log('update!!!!'));
});
// 方式2
Fruit.update({price:4}, {where:{id:1}}).then(r => {
    console.log(r);
    console.log('update!!!!')
})
```

- 删除

```
// 方式1
Fruit.findOne({ where: { id: 1 } }).then(r => r.destroy());

// 方式2
Fruit.destroy({ where: { id: 1 } }).then(r => console.log(r));
```

- 关联

> 视时间而定，充裕则讲，不够则在项目期带出

```
// 1:N关系
const Player = sequelize.define('player', {name: Sequelize.STRING});
const Team  = sequelize.define('team', {name: Sequelize.STRING});

// 会添加teamId到Player表作为外键
Player.belongsTo(Team); // 1端建立关系
Team.hasMany(Player); // N端建立关系

// 同步
sequelize.sync({force:true}).then(async ()=>{
  await Team.create({name: '火箭'});
  await Player.bulkCreate([{name: '哈登', teamId:1},{name: '保罗', teamId:1}]);

  // 1端关联查询
  const players = await Player.findAll({include:[Team]});
  console.log(JSON.stringify(players,null,'\t'));

  // N端关联查询
  const team = await Team.findOne({where:{name:'火箭'},include:[Player]});
  console.log(JSON.stringify(team,null,'\t'));
});


// 多对多关系
const Fruit = sequelize.define("fruit", { name: Sequelize.STRING });
const Category = sequelize.define("category", { name: Sequelize.STRING });
Fruit.FruitCategory = Fruit.belongsToMany(Category, {
  through: "FruitCategory"
});
```

```
// 插入测试数据
await Fruit.create(
    {
      name: "香蕉",
      categories: [{ id: 1, name: "热带" }, { id: 2, name: "温带" }]
    },
    {
      include: [Fruit.FruitCategory]
    }
);
// 多对多联合查询
const fruit = await Fruit.findOne({
    where: { name: "香蕉" }, // 通过through指定条件、字段等
    include: [{model:Category, through:{attributes:['id','name']} }]
});
```

- 事务