数据库

1. 数据库的分类

```
(1) 关系型数据库（RDBS）
    代表有：MySQL、Oracle、DB2、SQL Server...
    特点：关系紧密，都是表

    优点：
    1、易于维护：都是使用表结构，格式一致；
    2、使用方便：SQL通用，可用于复杂查询；
    3、高级查询：可用于一个表以及多个表之间非常复杂的查询。
    缺点：
    1、读写性能比较差，尤其是海量数据的高效率读写；
    2、有固定的表结构，字段不可随意更改，灵活度稍欠；
    3、高并发读写需求，传统关系型数据库来说，硬盘I/O是一个很大的瓶颈。

(2) 非关系型数据库（NoSQL）
    代表有：MongoDB、Redis...
    特点：关系不紧密，有文档，有键值对

    优点：
    1、格式灵活：存储数据的格式可以是key,value形式。
    2、速度快：nosql可以内存作为载体，而关系型数据库只能使用硬盘；
    3、易用：nosql数据库部署简单。
    缺点：
    1、不支持SQL，学习和使用成本较高；
    2、不支持事务；      （事务：原子性、不可分割性）
    3、复杂查询时语句过于繁琐。
```

2. MongoDB

1. 首先搞清楚几个概念：

- 数据库 (database)：数据库是一个仓库，在仓库中可以存放集合。
- 集合 (collection)：集合类似于JS中的数组，在集合中可以存放文档。说白了，集合就是一组文档。
- 文档 (document)：文档是数据库中的最小单位，我们存储和操作的内容都是文档。类似于JS中的对象，在MongoDB中每一条数据都是一个文档。

2. MongoDB基本命令：

1. db : 查看当前在操作哪一个数据库
2. show dbs： 查看数据库列表（一共有几个数据库，备注：如果数据库为空，不出现在列表中）
3. use test :切换到test数据库，如果不存在，则创建一个test库
4. db.students.insert()： 向当前数据库的students集合中插入一个文档。
5. show collections： 展示当前数据库中所有的集合。

3. MongoDB原生CRUD（增删改查）命令总结

```
每种对应记住一个命令即可
-C creat（新增数据）：

    db.集合名.insert(文档对象)
    db.集合名.insertOne(文档对象)    ***
    db.集合名.insertMany([文档对象，文档对象])

-R read：

    db.集合名.find(查询条件[,投影])  ***
        举例:db.students.find({age:18}),查找年龄为18的所有信息
        举例:db.students.find({age:18,name:'jack'}),查找年龄为18且名字为jack的学生

    常用操作符：
        1. < , <= , > , >= , !==   对应为：  $lt $lte $gt $gte $ne
            举例：db.集合名.find({age:{$gte:20}}),年龄是大于等于20的
        2.逻辑或：使用$in 或 $or
            查找年龄为18或20的学生
            举例：db.students.find({age:{$in:[18,20]}})
            举例：db.students.find({$or:[{age:18},{age:20}]})
        3.逻辑非：$nin
        4.正则匹配：
            举例：db.students.find({name:/^T/})
        5.$where能写函数：
            db.students.find({$where:function(){
                return this.name === 'zhangsan' && this.age === 18
            }})

    投影：过滤掉不想要的数据，只保留想要展示的数据
        举例：db.students.find({},{_id:0,name:0}),过滤掉id和name
        举例：db.students.find({},{age:1}),只保留age

    补充：db.集合名.findOne(查询条件[,投影])，默认只要找到一个

-U update：

    db.集合名.update(查询条件,要更新的内容[,配置对象])

    //如下写法会将更新内容替换掉整个文档对象，但_id不受影响
        举例：db.students.update({name:'zhangsan'},{age:19})

    //使用$set修改指定内容，其他数据不变，不过只能匹配一个zhangsan
        举例：db.students.update({name:'zhangsan'},{$set:{age:19}})

    //修改多个文档对象，匹配多个zhangsan,把所有zhangsan的年龄都替换为19
        举例：db.students.update({name:'zhangsan'},{$set:{age:19}},{multi:true})
***

     补充：db.集合名.updateOne(查询条件,要更新的内容[,配置对象])
          db.集合名.updateMany(查询条件,要更新的内容[,配置对象])

-D delete

    db.集合名.remove(查询条件)
        //删除所有年龄小于等于19的学生
```

```
        举例：db.students.remove({age:{$lte:19}})
```

4. Mongoose的使用

Mongoose是一个**对象文档模型**（ODM）库，它对Node原生的MongoDB模块进行了进一步的优化封装，并提供了更多的功能。

为什么用mongoose?

想在Node平台下，更加简单、高效、安全、稳定的操作mongoDB。

```
//引入mongoose
let mongoose = require('mongoose')

//1.连接数据库
mongoose.connect('mongodb://localhost:27017/demo',{
  useNewUrlParser: true, //使用一个新的URL解析器，用于解决一些安全性问题。
  useUnifiedTopology: true //使用一个统一的新的拓扑结构
})

//2.绑定数据库连接的监听
mongoose.connection.on('open',function (err) {
  if(err){
    console.log('数据库连接失败',err)
  }else{
    console.log('数据库连接成功')
    //3.操作数据库
    console.log('操作数据库')
  }
})
```

5. Mongoose的CRUD

```
-Create

        模型对象.create(文档对象，回调函数)
 -Read

        模型对象.find(查询条件[,投影],回调函数)不管有没有数据，都返回一个数组
        模型对象.findOne(查询条件[,投影],回调函数)找到了返回一个对象，没找到返回null
 -Update

       模型对象.updateOne(查询条件,要更新的内容[,配置对象],回调函数)
       模型对象.updateMany(查询条件,要更新的内容[,配置对象],回调函数)
       备注：存在update方法，但是即将废弃，查询条件匹配到多个时，依然只修改一个，强烈建议用
updateOne或updateMany
 -Delete

        模型对象.deleteOne(查询条件,回调函数)
        模型对象.deleteMany(查询条件,回调函数)
        备注：没有delete方法，会报错!

备注：  以上所有方法，如果没有指定回调函数，则返回值是一个Promise的实例
```

6. Mongoose的模块化

1. db/db.js

```
/*
* 该模块主要用于连接数据库，且判断数据库的连接状态
* */
let mongoose = require('mongoose')
mongoose.set('useCreateIndex',true) //使用一个新的索引创建器

const DB_NAME = 'demo'
const PORT = 27017
const IP = 'localhost'

function connectMongo(success, failed) {
  //1.连接数据库
  mongoose.connect(`mongodb://${IP}:${PORT}/${DB_NAME}`,{
    useNewUrlParser: true, //使用一个新的URL解析器，用于解决一些安全性问题。
    useUnifiedTopology: true //使用一个统一的新的拓扑结构。
  })

  //2.绑定数据库连接的监听
  mongoose.connection.on('open',function (err) {
    if(err){
      console.log('数据库连接失败',err)
      failed('connect failed')
    }else{
      console.log('数据库连接成功')
      success()
    }
  })
}

module.exports = connectMongo
```

2. model/studentModel.js

3.

```
//创建模型对象
//把数据库想象成你家的别墅
let mongoose = require('mongoose')

//1.请来一个帮你看门的保安 ------ 引入模式对象
let Schema = mongoose.Schema

//2.制定进入你家的规则 --------  创建约束对象
let studentsRule = new Schema({
  stu_id:{
    type:String, //限制学号必须为：字符串
    required:true,
    unique:true
  },
  name:{
    type:String, //限制姓名必须为：字符串
    required:true, //限制姓名为必填项
  },
  age:{
    type:Number, //限制年龄必须为：字符串
```

```
    required:true, //限制年龄为必填项
  },
  sex:{
    type:String, //限制性别必须为：字符串
    required:true, //限制性别为必填项
  },
  hobby:[String], //限制爱好只能为数组，数组中的每一项必须为字符串
  info:Schema.Types.Mixed, //接收所有类型
  date:{
    type:Date,
    default:Date.now()
  },
  enable_flag:{
    type:String,
    default:'Y'
  }
})

//3.告诉保安你的规则 ------- 创建模型对象
//let studentsModel = mongoose.model('students',studentsRule)
//module.exports = studentsModel
module.exports = mongoose.model('students',studentsRule) //用于生成某个集合所对应的模型对象
```

4. app.js

```
//使用模型对象进行CRUD
//1.引入mongoose库
let mongoose = require('mongoose')
//2.引入数据库连接模块
let db = require('./db/db')
//3.引入学生模型
let stuModel = require('./model/studentModel')

//判断数据的连接状态，若成功，CRUD
//判断数据的连接状态，若失败，报告错误

db(function(err){
  if (err) console.log(err)
  else{
    //真正进行CRUD
    stuModel.create({
      teac_id:'001',
      name:'光头强',
      age:'24',
      sex:'男',
      hobby:['女','打代码','打篮球'], //限制爱好只能为数组，数组中的每一项必须为字符串
      info:'一个风一样的男子', //接收所有类型
    },function(err,data){
      if (!err) console.log(data)
      else console.log(err)
    })
  }
})
```