MongoDB 数据库

1. 项目架构 - 经典方案

2. 数据库的介绍

3. 数据库的分类

4. MongoDB 的安装

5. 数据库的基本操作

   5.1 新建数据库

   5.2 操作集合

   5.3 操作文档 CURD

   Create(增，创建) - Update(改，修改) - Retrieve(查) - Delete(删)

6. 通过NodeJS操作数据库

   6.1 删除数据

   6.2 更新数据（修改数据）

   6.3 查询数据

7. 操作数据库的GUI

8. 企业级封装

9. 案例

10. 总结

    10.1 Http 上网流程

    10.2 怎么理解 http 是无状态的?

    10.3 cookie 和 session 的区别?

    10.4 说说 HTTP 协议和 HTTPS 协议?

    10.5 网站的部署逻辑

1. 项目架构 – 经典方案

LAMP – WAMP

L 或者 W 指的都是系统，Windows 系统或者 Linux 系统

A 指的是 Apache 阿帕奇，用于创建服务器，进行代码托管

M 指的是 MySQL 数据库，一般作为数据核心

P 指的是 PHP，泛指后端语言，比如 Java、NodeJS...

2. 数据库的介绍

数据库作为项目的核心资源，比较重要，一般会有专门的技术人员维护

对于数据库的操作，还会设置管理权限，在操作的时候需要输入用户名和密码

所以对于数据库的存放，还有数据的安全在所有公司的研发部门来看是很重要的

简单的使用，无需设置用户名和密码，对于逻辑的掌握最重要；

前端发起请求，服务器端处理请求，服务器端操作数据库，数据库返回数据，服务器端接收并响应给客户端，客户端获取数据进行页面渲染；

如果要使用数据库，那么就得安装数据库，不安装就没得用。

数据库得有数据，**得启动**，没有数据，或者根本都没有启动，则还是不能用。

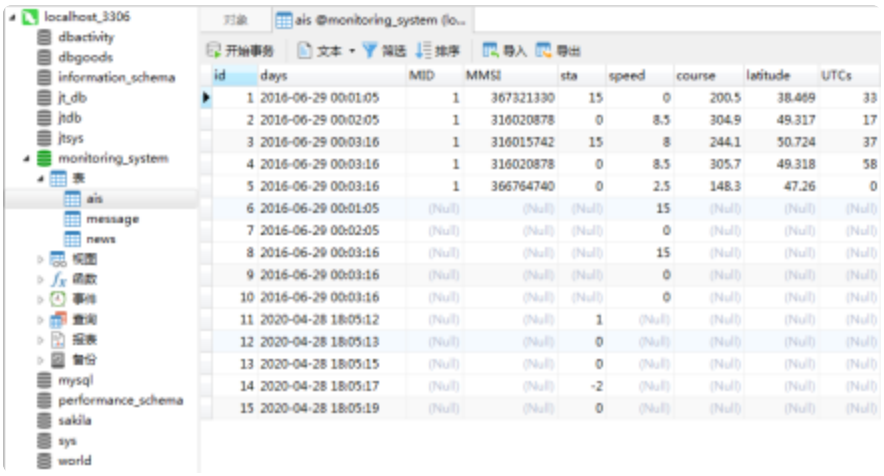
当数据库安装完毕，启动成功，数据添加结束，那么服务器端就可以开始想办法连接了，并且可以使用一些方式进行业务的增删查改的操作。

3. 数据库的分类

独立安装的数据库展示方式，在cmd中运行：

```
mysql> desc tb_student;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int(11)     | NO   | PRI | NULL    |       |
| name    | varchar(20) | YES  |     | NULL    |       |
| classID | int(11)     | YES  | MUL | NULL    |       |
+---------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> alter table tb_student add age int(4) after name;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc tb_student;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int(11)     | NO   | PRI | NULL    |       |
| name    | varchar(20) | YES  |     | NULL    |       |
| age     | int(4)      | YES  |     | NULL    |       |
| classID | int(11)     | YES  | MUL | NULL    |       |
+---------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

在图形界面中展示的数据库



关系型数据库：

ORACLE®

Oracle（甲骨文）： 全球最大的应用软件供应商，软件收费，但是大公司都用，稳定性高，数据安全性也会很高；

MySQL： 全球通用比较高的数据库，免费，稳定，国内大部分企业都在使用；

SQL Server: Microsoft 公司 (微软)旗下的一款数据库应用

需要安装单独的软件，或者在其他项目中集成式运行（phpStudy， myWramp），操作它们需要学习专门的SQL语句， 还有图形界面的工具。

sql 语句也是比较麻烦一些

SQL语句的演示 SQL 复制代码

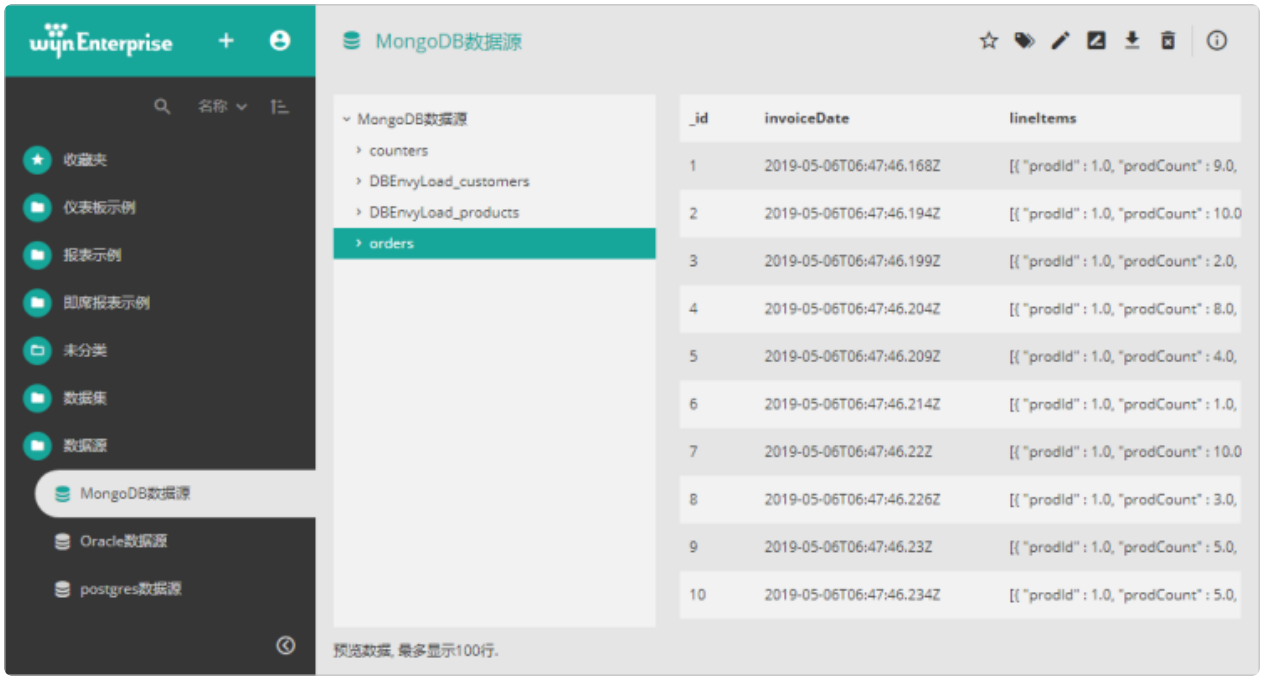
```sql
# 查询字段
SELECT * FROM users WHERE id = 2 AND age > 20 LIMIT 2, 10 DESC;
```

非关系型数据库

数据的特点：一条一条的数据，每条数据，都是一条json字段

非关系型数据库的模拟 JSON 复制代码

```
[
  {
    id: asdas12312dqsdq,
    {
      name: '张三',
      age: 20
    }
  },
  {
    id: asdas132342342d,
    {
      name: '李四',
      age: 21
    }
  }
]
```



MongoDB， 非关系型数据库，数据库的操作比较简单，都是一些方法继承，一般配合使用NodeJS 搭建的服务器使用。

4. MongoDB 的安装

建议使用压缩包的方式安装，官网直接下载可能有部分电脑不兼容，或者运行不动

在资源中找到压缩包，解压出来之后，放到 C:\Program Files 文件夹里面

改好名称： MongoDB （bin目录中，大量的exe后缀的文件）

添加到环境变量中： C:\Program Files\MongoDB\bin

此电脑 – 右键 – 属性 – 高级系统设置 – 环境变量 – 系统变量 – path属性 – 双击 – 添加

在C盘中，创建一个data目录，用于存放数据库文件

在任意地方，打开命名窗口 cmd 输入， **运行起来之后不要关了**

如果闪退了，建议使用 `$ mongod –dbpath C:/data`

上一个窗口不要关，再打开一个窗口运行一个命令 `$ mongo`

5. 数据库的基本操作

mongod –dbpath C:/data　　启动数据库

mongo　　开始命令操作

要启动数据库　　Bash　复制代码

```bash
# 查看数据库
$ show dbs

# 或者是
$ show databases
```

一个项目一般管理一个数据库，一个数据库可以有很多的数据集合，一个集合可以有多条数据，内部的数据库一般不需要处理

5.1 新建数据库

Bash　复制代码

```bash
# 新建数据库 – 创建并切换到了这个数据库
$ use chenwei

# 显示当前的数据库名称
$ db

# 删除数据库，要切换到当前数据库
$ db.dropDatabase();
```

5.2 操作集合

操作集合都是使用 db 开头的

Bash | 复制代码

```bash
# 创建集合 - 字段都是自定义的
$ db.createCollection('heros')

> { ok: 1 }

# 展示当前数据库中所有的集合
$ show collections

# 删除集合
$ db.创建的集合名称.drop()

# 修改集合名称
$ db.创建的集合名称.renameCollection('kings')

```

5.3 操作文档　CURD

Create(增，创建) – Update(改，修改) – Retrieve(查) – Delete(删)

创建的数据，每一条都会**自动携带一个id**，每条数据的id都是唯一

创建数据　Bash | 复制代码

```bash
# 对于集合创建数据
$ db.创建的集合名称.insert({ a: 10, b: 100 })
```

查询 Bash 复制代码

```
# 不写条件，会显示所有的数据
$ db.创建的集合名称.find()
# 基础条件 - 符合条件的所有数据
$ db.kings.find({ a: 30 })
# 符合的第一条数据
$ db.kings.findOne({ a: 30 })


# 更新文档中的数据 - update 方法会替换新的文档
$ db.创建的集合名称.update(查询条件,  新的文档);
# 指定更新 - 更新的文档需要设置 $set  已有属性会覆盖，新属性会创建
$ db.kings.updateOne({ a: 40 }, { $set: { name: '张三' } })


# 删除符合条件的数据 - 符合多少就会删除多少
$ db.kings.remove({ a: 20 })
```

6. 通过NodeJS操作数据库

如果需要在node项目中操作数据库，那么需要下载一个单独的插件

```
$ npm i mongoose
```

这个插件中，涵盖了如何进行连接，及操作数据库的增删查改对应的方法。

连接注意事项 JavaScript 复制代码

```javascript
// 1. 引入模块
const mongoose = require('mongoose');

// 2. 连接数据库
/*
    协议:  mongodb://      http://   https://
    解析或者连接规则:  { useNewUrlParser: true },
    回调函数: 单独去处理回调

    默认端口:   27017         mysql :   3306
    ***** 完整的uri格式,   协议 + ip + 端口 + 数据库的名称
*/
const uri = 'mongodb://127.0.0.1:27017/chenwei';
mongoose.connect(uri, { useNewUrlParser: true });

// 3. 设置监听的回调事件
mongoose.connection.on('open', () => {
    console.log('连接成功');
})

mongoose.connection.on('error', () => {
    console.log('连接失败！！');
})
```

JavaScript | 复制代码

```
// 1. 引入模块
const mongoose = require('mongoose');

// 2. 连接数据库
const str = 'mongodb://127.0.0.1:27017/chenwei';
mongoose.connect(str, { useNewUrlParser: true });

// 3. 注册事件
mongoose.connection.on('open', () => {

    // 4. 创建文档结构对象
    // 可以一眼知道关于这个对象的字段有哪些描述(类似于表结构中的表头)
    // Schema  ['skimə]  关于一个集合的字段描述
    const Stars = mongoose.Schema({
        name: String,
        age: Number,
        hobby: Array,
        isMarry: Boolean,
        brith: Date
    })

    // 5. 创建真实的模型对象 - Model  一般代表数据的意思
    const StarsModel = mongoose.model('popstar', Stars);

    // 6. 插入数据
    StarsModel.insertMany(
        [
            { name: '迪丽热巴', age: 20, hobby: ['电影', '唱歌'], isMarry: false, brith: '1990-08-24' },
            { name: '古力娜扎', age: 22, hobby: ['旅游', '电影'], isMarry: true, brith: '1992-01-19' }
        ]
        , (err, data) => {
            if (err) throw err;

            console.log('插入数据成功: ', data);
            // 6.1 操作成功之后，记得关闭数据库的连接 - 选做
            mongoose.connection.close();
        })


})
// 处理失败的事件
mongoose.connection.on('error', () => {

```

```
44  })
```

6.1 删除数据

使用模型自带的方法，操作删除数据的行为

JavaScript | 复制代码

```javascript
// 6. 删除数据
// StarsModel.deleteOne({ age: 20 }, (err, data) => {
//     if (err) throw err;
//     console.log('删除成功: ', data);
//     mongoose.connection.close();
// })

// 6.1 删除多项数据 - 参数1: 需要表述过滤的规则
// { age: { $gt: 20 } }   $gt  大于  $lt  小于
StarsModel.deleteMany({ age: { $gt: 20 } }, (err, data) => {
    if (err) throw err;
    console.log('删除成功: ', data);
    mongoose.connection.close();
})
```

6.2 更新数据（修改数据）

根据方法修改一条或者多条数据

JavaScript | 复制代码

```
// 6. 更新数据 - 参数1: 查询条件, 参数2: 新数据属性, 参数3: 回调函数
// 单条数据更新
// const filter = { name: '古力娜扎' };
// StarsModel.updateOne(filter, { isMarry: false }, (err, data) => {
//     if (err) throw err;
//     console.log('更新成功: ', data);
//     mongoose.connection.close();
// })

// 6.1 多条数据更新 - 参数1: 查询条件, 参数2: 新数据属性, 参数3: 回调函数
const filter = { age: { $lt: 30 } };
// 如果有新的字段是无法直接添加的 - 需要按照约定的结构来
// const updateData = { isMarry: false, isOk: 'ok' } // X  Ok 添加不了
const updateData = { isMarry: false }
StarsModel.updateMany(filter, updateData, (err, data) => {
    if (err) throw err;
    console.log('批量更新成功: ', data);
    mongoose.connection.close();
})
```

6.3 查询数据

基本查询和高级查询

基本查询 JavaScript 复制代码

```
// 6. 查询数据
// StarsModel.find({}, (err, data) => {
//     if (err) throw err;
//     console.log('查询成功: ', data);
//     mongoose.connection.close();
// })
// 6.1 根据条件查询 - 根据条件查询
// StarsModel.find({ age: { $gt: 20, $lt: 30 } }, (err, data) => {
//     if (err) throw err;
//     console.log('查询成功: ', data);
//     mongoose.connection.close();
// })
// 满足条件的第一条
// StarsModel.findOne({ age: { $gt: 20, $lt: 30 } }, (err, data) => {
//     if (err) throw err;
//     console.log('查询成功: ', data);
//     mongoose.connection.close();
// })
// 根据id获取数据
const id = "634cbecc2529c63a4cc96a5e";
StarsModel.findById(id, (err, data) => {
    // StarsModel.findById({ _id: id }, (err, data) => {
    if (err) throw err;
    console.log('查询成功: ', data);
    mongoose.connection.close();
})
```

高级查询 JavaScript 复制代码

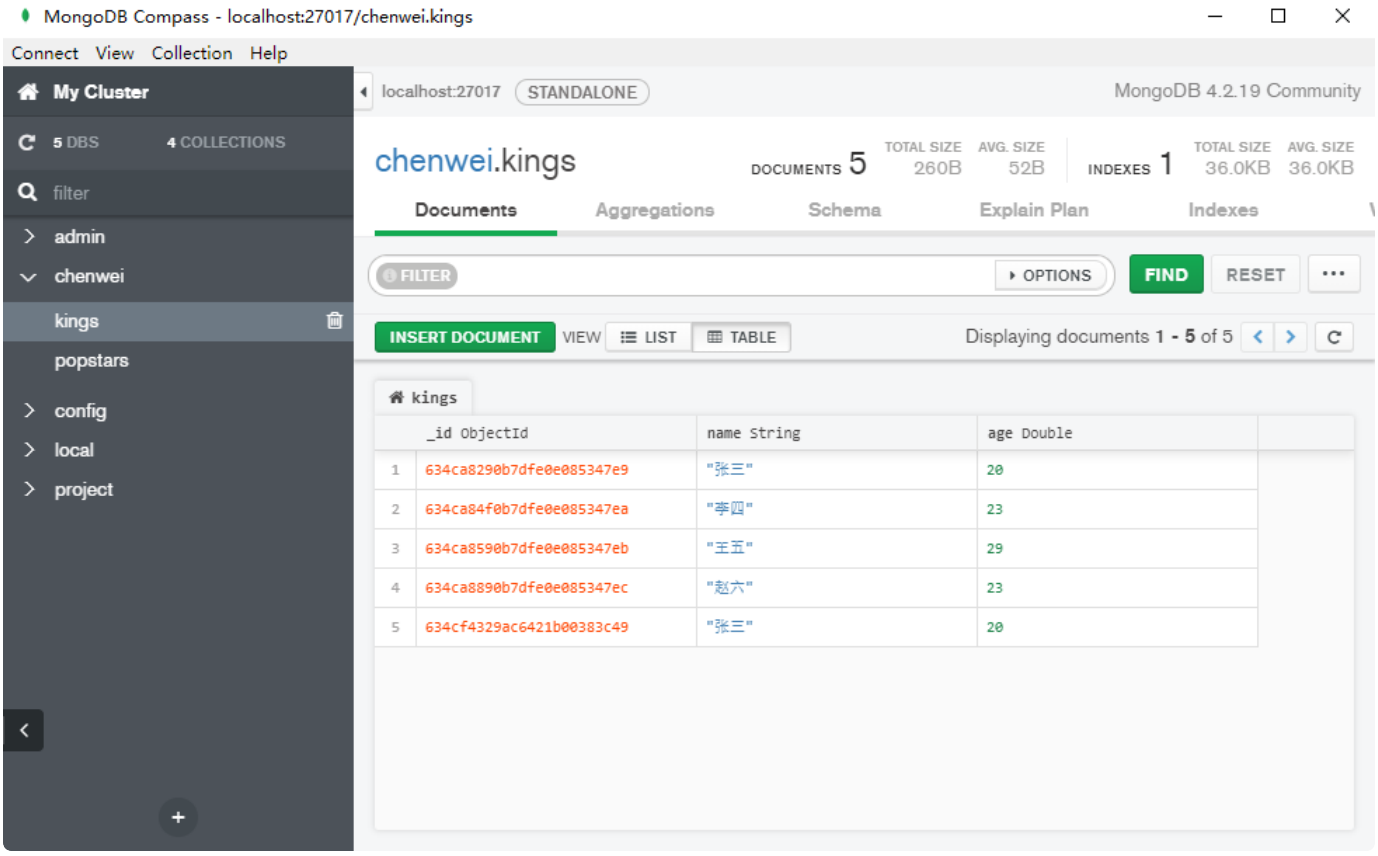
```
// 6. 查询数据
// 6.1 排序 - 链式方式操作
// sort 排序的规则，字段 1 表示以此字段做排序  1 从小到大(升序)，-1 从大到小(降序)
// exec 提取的方法 - 回调函数
// StarsModel.find().sort({ age: -1 }).exec((err, data) => {
//     if (err) throw err;
//     console.log('查询成功: ', data);
//     mongoose.connection.close();
// })

// 6.2 筛选部分属性 - 字段写1代表需要展示，0不展示
// const filter = { name: 1, age: 1, hobby: 1, isMarry: 1, _id: 0 };
// StarsModel.find().select(filter).exec((err, data) => {
//     if (err) throw err;
//     console.log('查询成功: ', data);
//     mongoose.connection.close();
// })

// 6.3 截取数据
// 设置起始位置
// StarsModel.find().skip(3).exec((err, data) => {
//     if (err) throw err;
//     console.log('查询成功: ', data);
//     mongoose.connection.close();
// })

// 设置获取的数量
// StarsModel.find().limit(2).exec((err, data) => {
//     if (err) throw err;
//     console.log('查询成功: ', data);
//     mongoose.connection.close();
// })

// 设置获取的区间 - 起始1，截取2，筛选3，排序4
// 逻辑关系: 3筛选 - 4排序 - 1起始 - 2截取
const fil = { name: 1, age: 1, _id: 0 };
// StarsModel
//     .find()
//     .sort({ age: -1 })
//     .select(fil)
//     .skip(2)
//     .limit(2)
//     .exec((err, data) => {
//         if (err) throw err;
//         console.log('查询成功: ', data);
```

```
//         mongoose.connection.close();
//     })

// 分页逻辑   pageNum 当前第几页    pageSize  每页展示的数据 8
// skip((pageNum - 1) * pageSize).limit(pageSize)

// 模糊查询
const str = '娜'; // req.body.name;
const reg = new RegExp(str, 'ig');
// i 忽略大小写，  g 开启全局匹配
StarsModel.find({ name: reg }).select(fil).exec((err, data) => {
    if (err) throw err;
    console.log('查询成功：', data);
    mongoose.connection.close();
})
```

7. 操作数据库的GUI

GUI是可视化界面操作数据库的软件，不能替代数据库，只是为了方便操作和查看数据库

GUI需要安装，也需要单独把数据库在本地启动起来，项目上线不需要GUI



8. 企业级封装

在企业项目中，经常需要对于数据库进行一些操作，所以没有办法直接每次写那些东西

封装和抽离模块的意思，就是为了更加方便的维护和使用

初级封装 – db.js 连接对象 JavaScript 复制代码

```
// 1. 引入模块
const mongoose = require('mongoose');

// 2. 创建连接
mongoose.connect('mongodb://127.0.0.1:27017/chenwei');

// 3. 监听事件
// mongoose.connection.on('open', () => {
//     console.log('成功');
// })
// mongoose.connection.on('error', () => {
//     console.log('连接失败，请检查错误原因!');
// })

// 4. 导出模块
module.exports = {
    conn(success, error) {
        // 在函数中，注册这俩事件
        // 4.1 open 事件，处理成功
        mongoose.connection.on('open', () => {
            // if (success) {
            //     success();
            // }
            // success && success();
            success?.();
        })
        // 4.2 error 事件，处理失败
        mongoose.connection.on('error', () => {
            error?.();
        })
    }
}
```

初级封装 – model.js 数据模型 JavaScript 复制代码

```javascript
// 1. 引入模块
const mogoose = require('mongoose');

// 2. 声明文档的结构
const BookModels = mogoose.Schema({
    name: String,
    price: Number,
    count: Number,
    Tags: String,
    update: Date
})

// 创建数据模型并且导出
// 这里的字符叫book是最终数据库中集合的名称
// 如果集合不加s，在生成的时候会自动加
module.exports = mogoose.model('book', BookModels);
```

初级封装 – index.js 最终使用 JavaScript 复制代码

```
// 1. 引入模块，会调用模块的内容
const mogoose = require('mongoose');
const db = require('./db/index');
const books = require('./models/books');

// 2. 调用封装的方法
// 处理函数给这边使用，不需要那么一大堆函数
// 参数1：成功的回调
// 参数2：失败的回调
db.conn(
    // 成功
    () => {
        // console.log('连接成功! ');

        // 2.1 操作books集合
        books.create({
            name: '《三国演义》',
            price: 68,
            count: 200,
            tags: '文学，四大名著',
            update: '1995-09-10'
        }, (err, data) => {
            if (err) throw err;

            console.log(data);
            // 关闭数据库的连接
            mogoose.connection.close();
        })

    },
    // 失败
    () => {
        console.log('error, please check your net.');
    }
);
```

在项目中，可能还会需要更高级的封装

更好的封装 – db.js JavaScript 复制代码

```
// 1. 引入模块
const mongoose = require('mongoose');

// 2. 导出连接方式
module.exports = (success, error) => {
    // 2. 创建连接服务
    // 协议，ip，端口，数据库名称
    const protocal = 'mongodb://';
    const ip = '127.0.0.1';
    const port = '27017'
    const database = 'chenwei';

    // 创建连接
    mongoose
        .connect(`${protocal}${ip}:${port}/${database}`)
        .then(() => {
            // 处理成功的回调
            success?.();
        })
        .catch(() => {
            // 捕获错误的回调
            // error?.();

            console.log('数据库连接失败，请检查网络.');
        })
}

```

更好的封装 – models 数据模型 JavaScript 复制代码

```javascript
// 1. 引入模块
const mogoose = require('mongoose');

// 2. 声明文档的结构
const BookModels = mogoose.Schema({
    name: String,
    price: Number,
    count: Number,
    Tags: String,
    update: Date
})

// 创建数据模型并且导出
// 这里的字符叫book是最终数据库中集合的名称
// 如果集合不加s，在生成的时候会自动加
module.exports = mogoose.model('book', BookModels);
```

更好的封装 – routes 请求中完成 JavaScript 复制代码

```javascript
// 1.1 引入数据库的模块
const db = require('./db/www');
const books = require('./models/books')
const mongoose = require('mongoose');


// 3. 书写路由模块
app.get('/', (req, res) => {
    // 3.1 接收模型对象进行调用方法
    db(() => {
        // 3.2 使用books 操作
        books.find().exec((err, data) => {
            if (err) throw err;

            // 3.3 返回给前端数据
            mongoose.connection.close();
            res.send({
                status: 200,
                msg: '查询成功',
                data
            })
        })
    })
})
```

9. 案例

```
$ express 项目名称
```

```
$ cd 项目
```

```
$ npm i
```

```
$ npm i mongoose
```

整理项目

封装代码 JavaScript 复制代码

```
// 1. 引入模块
const mongoose = require('mongoose');

// 2. 导出连接方式
module.exports = (success, error) => {
    // 2. 创建连接服务
    // 协议，ip，端口，数据库名称
    const protocal = 'mongodb://';
    const ip = '127.0.0.1';
    const port = '27017'
    const database = 'my_books';

    // 创建连接
    mongoose
        // .connect(`${protocal}${ip}:${port}/${database}`)
        .connect(`mongodb://127.0.0.1:27017/my_books`)
        .then(() => {
            // 处理成功的回调
            success?.();
        })
        .catch(() => {
            // 捕获错误的回调
            error?.();
        })
}

```

创建数据模块

根据要导入的json处理 JavaScript 复制代码

```javascript
// 1. 引入模块
const mogoose = require('mongoose');

// 2. 声明文档的结构
const BookModels = mogoose.Schema({
    name: String,
    author: String,
    price: Number,
    cbs: String,
    img: String,
    intro: String
})

// 创建数据模型并且导出
// 这里的字符叫book是最终数据库中集合的名称
// 如果集合不加s，在生成的时候会自动加
module.exports = mogoose.model('book', BookModels);
```

测试接口，只能测试一次，不然会创建重复的数据，数据创建之后，记得注释掉

JavaScript 复制代码

```
const fs = require('fs');
const path = require('path');

// 1. 数据库的那些模块
const db = require('../db/db');
const books = require('../model/books');
const mongoose = require('mongoose');

/* 首页 */
router.get('/', function (req, res, next) {
  // 1. 测试 - 添加数据
  db(() => {
    // 1.1 使用添加数据的方式创建内容
    fs.readFile(path.join(__dirname, '../public/db.json'), 'utf8', (err, data) => {
      if (err) throw err;

      // 1.2 读取了数据之后，需要创建
      // console.log(typeof data);  // 是字符串类型的
      books.create(JSON.parse(data), (err, data) => {
        if (err) throw err

        res.send('数据写入成功！')
      })
    })
  }, () => {
    console.log('数据库连接失败，请检查网络.');
  })
});
```

响应前端页面

```
$ npm i express-art-template
```

注册一下 JavaScript 复制代码

```
app.engine('htm', require('express-art-template'))
app.set('view engine', 'htm');
app.set('views', './public')
```

先写接口 JavaScript 复制代码

```
// 2. 客户端请求所有的数据
router.get('/data', (req, res) => {
  const { num } = req.query;
  count = 0;

  // 2.1 使用模型对象访问数据
  db(() => {
    // 2.2 find方法获取所有的数据
    if (num) count = num;
    books.find().skip(count).limit(5).exec((err, data) => {
      if (err) throw err;

      // 渲染指定页面，并返回数据
      res.render('index', { data, msg: '请求成功！' })
    })
  })
})
```

HTML | 复制代码

```html
<head>
    <style>
        p {
            // 多行省略
            display: -webkit-box;
            -webkit-line-clamp: 3;
            -webkit-box-orient: vertical;
            overflow: hidden;
            text-overflow: ellipsis;
        }
    </style>
</head>

<body>
    <h3>{{ msg }}</h3>
    <div>
        <ul>
            {{ each data val i }}
            <li>
                <img src="/static{{ val.img }}" alt="">
                <p>名称：{{ val.name }}</p>
                <p>作者：{{ val.author }}</p>
                <p>出版社：{{ val.cbs }}</p>
                <p>简介：{{ val.intro }}</p>
            </li>
            {{ /each }}
        </ul>
    </div>

    <a href="/data?num=5">下一页</a>
</body>
```

案例总结：

1， 所有的文件，都是存储在服务器上，不是存储在数据库中，数据库只能存储服务器中文件存储的地址，也就是说，数据库中只能出现描述的数据字符，而不应该出现具体的文件内容

2，如果需要做**局部刷新**，目前无法完成，因为每一次的请求或者跳转，都会带来页面的整体刷新，意味着每次页面的内容都会被重置。

10. 总结

10.1 Http 上网流程

http是基于TCP/IP之上的应用层协议 LaTeX 复制代码

```
1, url 地址会被运营商的DNS服务器解析域名为对应的ip
2, 根据ip地址和端口，找到互联网对应的那台服务器
3, 客户端跟服务器进行三次握手，建立稳定的通信
4, 客户端使用http协议发起请求，服务器端根据响应规则返回内容
5, 浏览器接收响应体，在本地进行渲染，分别使用排版引擎和js解析引擎
6, 部分会进入缓存，响应结束之后，客户端和服务器端会断开连接(四次挥手)
```

10.2 怎么理解 http 是无状态的?

一次请求和一次响应是一次会话 Plain Text 复制代码

```
1, http是短连接，一次请求一次响应，多次请求，多次响应
   比如 websocket 就是长链接，可以不需要请求直接响应，也可以多次请求不需要响应
2, http无状态的意思，就是下一次会话，并不会知晓或者记录上一次会话的内容
3,
  解决方案:
      服务器端，在客户端第一次发起请求之后，返回一个 cookie 的字段
      存储在客户端，下一次发起请求，会自动在请求头中携带这个 cookie 的字段

      或者说，在客户端第一次发起请求之后，创建一个session的内容
      把session对应的id以加密的方式使用cookie响应回去，存储在客户端
      下一次发起请求，会自动在请求头中携带这个加密的字段
```

10.3 cookie 和 session 的区别?

```latex
1, cookie 能存储的内容比较小，浏览器大部分只能允许一次最多22次，最大50次
   单次存储最大的字符大小为 4 kb, 只能存储字符
   客户端自行存储，自行携带，如果内容比较多，比较消耗网络传输的能力
   cookie 是明文传输，安全性不高，存储的内容会被程序员直接看到
   cookie 可以设置有效期的时长，过期之后会自动销毁


2, session 存储的位置在服务器端，能存储的内容较多，没有限制
   返回给客户端的是一个加密之后的id，根据id可以解密对应的内容
   session 返回的id还是基于 cookie 响应的，不能单独响应

3, 不管是 cookie 还是 session，对于 http 协议来说，都不怎么安全
   https 因为在发起请求的时候就会做一些加密的工作，所以相对安全

   如果是跨服务器存储内容，那么 cookie 或者 session 都不可靠
   现在流行的项目，是以 校验 token 的方式，也是一堆加密的字符
   但是需要自己手动登录之后存储，再后期请求之后手动加入到请求头

   jwt 生成token
```

10.4 说说 HTTP 协议和 HTTPS 协议?

HTTP优先 Plain Text 复制代码

```
1, http 是基于 TCP/IP 之上的一个应用层协议
2, IP 主要负责找到服务器地址, 属于是通信协议，规定了ip的地址，4个号段，0-255
3，TCP 属于是传输协议，根据ip找到的服务器，建立3次握手，响应结束之后4次挥手
4, http 的主要作用是 请求和响应

5, 每一次请求会产生请求报文  常见的请求方式 GET  POST  PUT  DELETE
   // 传统的页面只有表单才可以发送post请求，浏览器大部分都是 GET 请求
   请求行  请求方式   url地址   get 的请求参数
   请求头
         content-type:   不同的请求内容，需要使用不同的请求头主体
             application/x-www-form-urlencoded
             application/json
             multipart/form-data
   请求体  get 请求没有请求体

6, 每一次响应会产生响应报文
   响应状态行
      响应状态码  响应状态的短语  OK
        1xx 发送中  2xx 成功   3xx 重定向   4xx  客户端问题  5xx 服务器问题
   响应头
        Set-Cookies ...
   响应体
       主要接收的内容，进行渲染， MIME 类型  text/html  text/css  image/jpg


7, https 会更安全，在简历通信的时候，握手次数更多，请求内容会SSL加密
   需要在建立网站的时候申请对应的证书，然后需要消耗更多的网络
   加密过程会更加繁琐，一般来说大型网站才需要使用
   访问的过程会稍微慢一些，所以需要提升前端的加载性能
```

10.5 网站的部署逻辑

一个网站，通常需要有 系统，服务器，数据库，前后端的项目 组成

LaTeX | 复制代码

```
1, 系统基本上都是使用 windows 居多，大佬使用 Linux，有钱的公司都用 Mac
2, 服务器，可以是后端的集成软件创建，也可以是nodejs通过 express 快速创建
3, 数据库有关系型数据库和非关系型数据库
    mysql     =>   sql 的查询语句
    mongodb   =>   mongoose 模块，有对应的 CRUD 的操作方法

4, 远程的服务器，需要部署这些内容，
       数据库有数据，数据库要启动
       服务器要有后端项目，服务器要启动
       前端的项目要托管，访问的方式以地址的方式

5, 当客户端发起请求，服务器端的路由就会响应，查看req 请求参数
    根据请求参数，调用查找数据库的操作方法
    提取相关的内容，通过 res 返回给客户端渲染

6. 期待，能通过异步刷新的方式，完成页面的其他数据更新。
```