

普通文件和数据库存储的对比

数据库分类

资源

文件系统数据库

node.js如何连接mysql

Node.js ORM框架-Sequelize

概述

安装

基本使用

定义模型

数据类型

API详讲

添加一条或多条数据

查询所有数据

更新数据

删除数据

验证属性

模型扩展和实例扩展

根据条件查询

查询属性

查询条件

常用的符号运算符

范围运算符

组合

别名

分页

排序

如何让异步代码同步化

普通文件和数据库存储的对比

在大多数企业开发或Web开发中，都会涉及数据的存储和检索。存储数据有两种基本的方法：保存到普通文件中(File System)，或者保存到数据库(Database)中。

文件存储常见，并且简单，操作系统提供的完善的API，所以在早期项目中都会使用文件作存储载体。但是随着企业业务越来越复杂，网站访问量也越来越大时，对数据的并发性和检索速度有更高的要求。所以慢慢的也就引入使用数据库作为数据存储了。

- 文件系统用文件来保存数据，不宜共享；数据库系统用数据库统一存储数据
- 文件系统程序(代码)和数据有一定的联系；数据库系统中的程序和数据分离
- 不安全，因为文件系统没有锁的概念。数据库系统数据安全

数据库分类

- 非关系型数据库
 - mongodb
 - 书的文档对象

```
1  [  
2    {  
3      id: '1',  
4      name: '123'  
5    },  
6    {  
7      id: '1',  
8      name: '123'  
9    }  
10 ]
```

- - 作者的文档对象
- 关系型数据库
 - Mysql
 - 表和表之间的关系
 - Books(书籍)
 - id name author price
 - 1 javascript高级编程 小马哥xxx 88
 - 2 vue开发 小尤 99
 - author (作者)

资源

- MySQL相关
 - MySQL:[下载安装](#)
 - mac安装

- node驱动:[文档](#)
- Sequelize:[文档+v5API](#)
- mongodb相关
 - MongoDB:[下载](#)
 - node驱动:[文档](#)
 - mongoose:[文档](#)

文件系统数据库

```
1  const FILEPATH = './db/my.json'
2  const fs = require('fs');
3  function get(key) {
4      fs.readFile(FILEPATH, (err, data) => {
5          const json = JSON.parse(data);
6          console.log(json[key]);
7      })
8  }
9
10 function set(key, value) {
11     fs.readFile(FILEPATH, (err, data) => {
12
13         // 可能是空文件,则设置为空对象
14         const json = data.toString() ?
15             JSON.parse(data) : {};
16         json[key] = value; //设置值
17         // 重新写入文件
18         fs.writeFile(FILEPATH,
19             JSON.stringify(json), err => {
20             if (err) console.log(err);
21         });
22     });
23 }
```

```
18         console.log('写入成功');
19     })
20 })
21 }
22
23 // 命令行接口部分
24 const readline = require('readline');
25 //创建实例
26 const rl = readline.createInterface({
27     input: process.stdin,
28     output: process.stdout
29 });
30 //监听命令行的输入
31 rl.on('line', (input) => {
32     const [op, key, value] = input.split(' ');
33     if (op === 'get') {
34         get(key);
35     } else if (op === 'set') {
36         console.log(key, value);
37
38         set(key, value);
39     } else if (op === 'quit') {
40         rl.close();
41     } else {
42         console.log('没有该操作');
43
44     }
45
46 });
47 // 程序结束
48 rl.on('close', () => {
```

```
49     console.log('程序结束');
50     process.exit(0);
51 })
52
```

node.js如何连接mysql

```
1  // 1.安装 npm i mysql -S
2  // 2.导入模块
3  const mysql = require('mysql');
4
5  // 3.创建连接
6  const conn = mysql.createConnection({
7      host: 'localhost',
8      user: 'root', //用户
9      password: '', //密码
10     database: 'db1' //请确保数据库存在
11 });
12 // 4.连接
13 conn.connect(err => {
14     if (err) throw err;
15     console.log('连接成功');
16 });
17
18 // 创建表
19 const CREATE_SQL = `CREATE TABLE IF NOT EXISTS
test ( id INT NOT NULL PRIMARY KEY
auto_increment, name VARCHAR ( 30 ) )`;
```

```

20 const INSERT_SQL = `INSERT INTO test(name)
VALUES(?)`;
21 const SELECT_SQL = `SELECT * FROM test`;
22 //查询 conn.query()
23 conn.query(CREATE_SQL, (error)=> {
24     if (error) throw error;
25     conn.query(INSERT_SQL, 'hello', (err, result)
=> {
26         console.log(err);
27         if (err) throw err;
28         console.log(result);
29         conn.query(SELECT_SQL, (err, results)=>{
30             console.log(results);
31             // 关闭连接
32             conn.end(); //若query语句有嵌套,则end
需在此执行
33         })
34     })
35 });
36

```

上个代码可以看出 `conn.query()` 方法的多次嵌套使用,已经造成了回调地域的问题,我们可以尝试将我们整个代码进行封装

```
1 //db/mysql.config.js
2 module.exports = {
3   host: 'localhost',
4   user: 'root',
5   password: '',
6   database: 'db1'
7 }
```

```
1 //db/mysql.js
2 const mysql = require('mysql');
3 const databaseConfig =
  require('./mysql.config');
4 module.exports = {
5   query: function (sql, params) {
6     return new Promise((resolve, reject) =>
7     {
8       // 3.创建连接
9       const conn =
10      mysql.createConnection(databaseConfig);
11      // 创建连接
12      conn.connect(err => {
13        if (err) reject(err);
14        console.log('连接成功');
15      });
16      // 查询 格式化的方式插入字段值
17      sql = mysql.format(sql, params);
18      conn.query(sql, (err, results,
19      fields) => {
20        if (err) throw reject(err);
21      });
22    });
23   }
24 }
```



```

18          // 如果没有错 将查询出来的数据返回
    给回调函数
19          resolve(results);
20          //停止链接数据库，必须在查询语句
    后，要不然一调用这个方法，就直接停止链接，数据操作就
    会失败
21          conn.end();
22      })
23  })
24
25  }
26  }

```

调用示例:

```

1  const CREATE_SQL = `CREATE TABLE IF NOT EXISTS
    test ( id INT NOT NULL PRIMARY KEY
    auto_increment, name VARCHAR ( 30 ) )`;
2  const INSERT_SQL = `INSERT INTO test(name)
    VALUES(?)`;
3  const SELECT_SQL = `SELECT * FROM test`;
4  const DELETE_SQL = 'DELETE FROM test WHERE id=1'
5  const db = require('./db/mysql.js');
6  async function asyncQuery() {
7      const re1 = await db.query(CREATE_SQL);
8      const re2 = await
    db.query(INSERT_SQL, 'HELLO');
9      const re3 = await db.query(SELECT_SQL);
10 }
11 asyncQuery();

```

Node.js ORM框架-[Sequelize](#)

[中文文档](#)

- 1 对象关系映射 (Object Relational Mapping, 简称ORM) 是通过使用描述对象和数据库之间映射的元数据, 将面向对象语言程序中的对象自动持久化到关系数据库中

概述

Sequelize.js是一款基于Promise的针对nodejs的ORM框架。具体就是突出一个支持广泛, 配置和查询方法统一。它支持的数据库包括:PostgreSQL、MySQL、MariaDB、SQLite 和 MSSQL。

为什么选择它?

使用nodejs连接过数据库的人肯定对数据库不陌生了。如果是直接链接, 需要自己建立并管理连接, 还需要手动编写sql语句。简单的项目倒无所谓, 可是一旦项目设计的东西比较复杂, 表比较多的时候整个sql的编写就非常的消耗精力。

在Java、c#等语言中已经有轻量的数据库框架或者解决方案了。在nodejs中我推荐Sequelize。它是一个很成熟的框架, 在速度和性能上也非常有优势。而其中最关键的地方就在于, 日常开发只需要管理对象的创建、查询方法的调用等即可, 极少需要编写sql语句。这一个好处就是省去了复杂的sql语句维护, 同时也避免了因sql而引起的不必要的bug。

安装

```
1 npm i sequelize mysql2 -S
```

基本使用

```
1 const Sequelize = require('sequelize');
2 // 建立连接
3 const sequelize = new Sequelize('db1 'root', '',
4   {
5     host: 'localhost',
6     dialect: 'mysql'
7   })
```

定义模型

在使用之前一定要先创建模型对象。就是数据库中表的名称、使用到的字段、字段类型等。

这里有一个推荐的开发方式。先在nodejs中将对象创建出来，然后调用Sequelize的同步方法，将数据库自动创建出来。这样就避免了既要写代码建表，又要手工创建数据库中的表的操作。只需要单独考虑代码中的对象类型等属性就好了。

如果数据库中已经建好了表，并且不能删除，这个时候就不能自动创建了，因为创建的时候会删除掉旧的数据。

```
1 const Sequelize = require('sequelize');
2 // 建立连接
```

```
3  const sequelize = new Sequelize('db1', 'root',
  '', {
4    host: 'localhost',
5    dialect: 'mysql'
6  })
7  // 定义模型
8  const Books = sequelize.define('books'/*自定义表
  名 */, {
9    id: {
10      type: Sequelize.INTEGER, //定义类型
11      primaryKey: true, //主键
12      autoIncrement: true,
13      comment: '自增id'
14    },
15    name: {
16      type: Sequelize.STRING,
17      allowNull: false, //不允许为null
18    },
19    price: {
20      type: Sequelize.FLOAT,
21      allowNull: false,
22    },
23    count: {
24      type: Sequelize.INTEGER,
25      defaultValue: 0
26    }
27  })
28  //模型同步 你的模型定义自动创建表(或根据需要进行修
  改),你可以使用sync方法 同步:没有就新建,有就不变
29  // students.sync();
30  Books.sync({
```

```

31     force: true
32 })
33
34 /*
35 一次同步所有模型
36 你可以调用sequelize.sync()来自动同步所有模型,而不是
    为每个模型调用sync().
37
38 */

```

| id | username | age | createdAt | updatedAt |
|----|----------|-----|--------------------|--------------------|
| 1 | 张三 | 30 | 2019-09-05 00:50:3 | 2019-09-05 00:50:3 |
| 2 | 李四 | 16 | 2019-09-05 00:50:3 | 2019-09-05 00:50:3 |

如果不要createdAt和updatedAt字段

```

1  const Books = sequelize.define('books'/*自定义表名
    */, {...}, {
2    // `timestamps` 字段指定是否将创建 `createdAt` 和
    `updatedAt` 字段.
3    // 该值默认为 true, 但是当前设定为 false
4    timestamps: false,

    freezeTableName:true,//表名冻结
5  })

```

数据类型

```
1 Sequelize.STRING           //字符串,长度默认
   255,VARCHAR(255)
2 Sequelize.STRING(1234)    //设定长度的字符串,VARCHAR(1234)
3 Sequelize.STRING.BINARY   //定义类型VARCHAR
   BINARY
4 Sequelize.TEXT            //长字符串,文本 TEXT
5 Sequelize.TEXT('tiny')    //小文本字符串,TINYTEXT
6
7 Sequelize.INTEGER         //int数字,int
8 Sequelize.BIGINT          //更大的数字,BIGINT
9 Sequelize.BIGINT(11)      //设定长度的数字,BIGINT(11)
10
11 Sequelize.FLOAT          //浮点类型,FLOAT
12 Sequelize.FLOAT(11)       //设定长度的浮点,FLOAT(11)
13 Sequelize.FLOAT(11, 12)   //设定长度和小数位数的浮
   点,FLOAT(11,12)
14
15 Sequelize.DOUBLE         // DOUBLE
16 Sequelize.DOUBLE(11)     // DOUBLE(11)
17 Sequelize.DOUBLE(11, 12) // DOUBLE(11,12)
18
19 //准确的小数点 数据类型
20 Sequelize.DECIMAL        // DECIMAL
21 Sequelize.DECIMAL(10, 2) // DECIMAL(10,2)
22
23
24 Sequelize.DATE           // 日期类型,DATETIME for mysql
   / sqlite, TIMESTAMP WITH TIME ZONE for postgres
```

```

25 Sequelize.DATE(6) // mysql 5.6.4+支持,分秒精度为6
   位
26 Sequelize.DATEONLY // 仅日期部分
27
28 Sequelize.BOOLEAN // int类型,长度为1,TINYINT(1)
29 Sequelize.ENUM('value 1', 'value 2') // 枚举类型
30 Sequelize.JSON // JSON column. PostgreSQL,
   SQLite and MySQL only.

```

API详讲

下面所有演示都是以插入单条数据为例

添加一条或多条数据

```

1 Books.sync({
2   force: true
3 }).then(() => {
4   // 单条数据使用create({})
5   return Books.create({
6     name: '你不知道JavaScript',
7     price: 12.9,
8     count: 10
9   })
10
11   // 插入多条数据使用bulkCreate([{},{ }])
12   // return Books.bulkCreate([
13     // {
14     //   name: '你不知道JavaScript',
15     //   price: -12.9,
16     //   count: 10

```

```
17     //     },
18     //     {
19     //         name: 'Vue实战开发',
20     //         price: -20.9
21     //     },
22     // ])
23 })
24 //返回Promise
```

查询所有数据

```
1 // 查询所有数据
2 Books.findAll().then(books => {
3     console.log(JSON.stringify(books));
4 })
```

为某字段添加新特性

比如可以给name字段添加get()方法


```

1 name: {
2     type: Sequelize.STRING,
3     allowNull: false, //不允许为null
4     get() {
5         const name =
6         this.getDataValue('name');
7         const price =
8         this.getDataValue('price');
9         return `${name}的这本书价格为
10        (${price})`
11     },
12 },
13 //[{ "name": "你不知道JavaScript的这本书价格为
14    (12.9)", "id": 1, "price": 12.9, "count": 10 },
15    { "name": "Vue实战开发的这本书价格为
16    (20.9)", "id": 2, "price": 20.9, "count": 0 }]

```

也可以给实例添加setter和getter方法

```


1 const Books = sequelize.define('books', {},
2 {
3     getterMethods: {
4         amount() {
5             return getDataValue('count') + '本';
6         }
7     },
8     setterMethods: {
9         amount(val) {
10             const idx = val.indexOf('本');
11             const v = val.slice(0, idx);
12             this.setDataValue('count', v);

```

```
13     }  
14   }  
15 })
```

更新数据

```
1 Books.findAll().then(books => {  
2   // 更新  
3   books[0].amount = '30本';  
4   // 更新数据后一定要save 不然不起作用  
5   books[0].save();  
6 })  
7 //更新数据  
8 Books.update({price:5},where:{id:1})
```



| id | name | price | count |
|----|----------------|-------|-------|
| 1 | 你不知道JavaScript | 12.9 | 30 |
| 2 | Vue实战开发 | 20.9 | 0 |

删除数据

```
1 Books.destroy({where:{id:1}})
```

验证属性

```
1 price: {
```

```
2         type: Sequelize.FLOAT,
3         allowNull: false,
4         validate: {
5             isFloat: {
6                 msg: '价格字段必须输入数字'
7             },
8             min: {
9                 args: [[0]],
10                msg: "价格必须大于0"
11            },
12            max: 100
13        }
14    },
```

模型扩展和实例扩展

```
1 //模型扩展
2 Books.classify = function (name) {
3     const books = ['你不知道的JavaScript', 'Vue实战
4     开发'];
5     return books.includes(name) ? '新华出版
6     社': '其它出版社';
7 }
8
9
10 // 实例扩展
11 Books.prototype.totalPrice = function(count) {
12     return this.price * count
13 }
```

```
14 |
15 //测试
16 books[0].totalPrice(50); //计算总价格
```

根据条件查询

```
1 // 查询一条数据
2 // SELECT `id`, `name`, `price`, `count` FROM
  `books` AS `books` WHERE `books`.`id` = 1;
3 Books.findOne({where:
  {id:1}}).then(book=>console.log(book.get()))
```

查询属性

```
1 Books.findAll({
2   attributes:['id','name'],
3 })
4 //select id,name...
5 //可以使用嵌套数组重命名属性
6 Books.findAll({
7   attributes:['id',['name','myName']],
8 })
9 //select id,name as myName ...
10
11 //sequelize.fn用来进行聚合
12 Books.findAll({
13   attributes: [[sequelize.fn('COUNT',
14     sequelize.col('name')), 'no_name']]
14 })
15 //select count(name) as no_name ...
```

```
16 //如何不添加attributes属性，是获取所有的属性，我们可以
    删除选定的几个属性，获得剩余的属性
17 Books.findAll({
18     attributes: {exclude: 'name'}
19 })
20 //select id,price,count ....
```

查询条件

```
1  const Op = Sequelize.Op;
2  Books.findAll({
3      where:{
4          id:1,
5          count:10
6      }
7  })
8  //select * from books where id = 1 and count = 10;
9  Books.findAll({
10     where:{
11         [Op.or]:[{price:12.5},{price:10.5}]
12     }
13 })
14 //select * from books where price = 12.5 or price =10.5;
15 Books.findAll({
16     where:{
17         price:{
18             [Op.or]:[12.5,10.5]
19         }
20     }
```

```

21  })
22  //select * from books where price = 12.5 or
    price = 10.5;
23  Books.findAll({
24      where:{
25          price:{
26              [Op.gt]:10 //价格大于10
27          }
28      }
29  })
30  //select * from books where price > 10;
31

```

常用的符号运算符

```

1  const Op = Sequelize.Op
2
3  [Op.and]: {a: 5}           // AND (a = 5)
4  [Op.or]: [{a: 5}, {a: 6}] // (a = 5 OR a = 6)
5  [Op.gt]: 6,                // > 6
6  [Op.gte]: 6,               // >= 6
7  [Op.lt]: 10,               // < 10
8  [Op.lte]: 10,              // <= 10
9  [Op.ne]: 20,               // != 20
10 [Op.eq]: 3,                // = 3
11 [Op.is]: null               // IS NULL
12 [Op.not]: true,             // IS NOT TRUE
13 [Op.between]: [6, 10],      // BETWEEN 6 AND 10
14 [Op.notBetween]: [11, 15], // NOT BETWEEN 11 AND
    15
15 [Op.in]: [1, 2],           // IN [1, 2]

```

```

16 [Op.notIn]: [1, 2],           // NOT IN [1, 2]
17 [Op.like]: '%hat',          // LIKE '%hat'
18 [Op.notLike]: '%hat'        // NOT LIKE '%hat'
19 [Op.iLike]: '%hat'          // ILIKE '%hat' (case
    insensitive) (PG only)
20 [Op.notILike]: '%hat'       // NOT ILIKE '%hat'
    (PG only)
21 [Op.startsWith]: 'hat'      // LIKE 'hat%'
22 [Op.endsWith]: 'hat'        // LIKE '%hat'
23 [Op.substring]: 'hat'       // LIKE '%hat%'
24 [Op.regexp]: '^h|a|t]'      // REGEXP/~
    '^h|a|t]' (MySQL/PG only)
25 [Op.notRegexp]: '^h|a|t]'   // NOT REGEXP/!~
    '^h|a|t]' (MySQL/PG only)
26 [Op.iRegexp]: '^h|a|t]'     // ~* '^h|a|t]' (PG
    only)
27 [Op.notIRegexp]: '^h|a|t]' // !~* '^h|a|t]'
    (PG only)
28 [Op.like]: { [Op.any]: ['cat', 'hat']}
29                               // LIKE ANY
    ARRAY['cat', 'hat'] - also works for iLike and
    notLike
30 [Op.overlap]: [1, 2]        // && [1, 2] (PG
    array overlap operator)
31 [Op.contains]: [1, 2]       // @> [1, 2] (PG
    array contains operator)
32 [Op.contained]: [1, 2]      // <@ [1, 2] (PG
    array contained by operator)
33 [Op.any]: [2,3]             // ANY ARRAY[2,
    3]::INTEGER (PG only)
34

```

```
35 [Op.col]: 'user.organization_id' // =  
    "user"."organization_id", with dialect specific  
    column identifiers, PG in this example  
36 [Op.gt]: { [Op.all]: literal('SELECT 1') }  
37           // > ALL (SELECT 1)
```

范围运算符

```
1 [Op.contains]: 2           // @> '2'::integer (PG  
    range contains element operator)  
2 [Op.contains]: [1, 2]      // @> [1, 2) (PG range  
    contains range operator)  
3 [Op.contained]: [1, 2]     // <@ [1, 2) (PG range  
    is contained by operator)  
4 [Op.overlap]: [1, 2]       // && [1, 2) (PG range  
    overlap (have points in common) operator)  
5 [Op.adjacent]: [1, 2]     // -|- [1, 2) (PG  
    range is adjacent to operator)  
6 [Op.strictLeft]: [1, 2]    // << [1, 2) (PG range  
    strictly left of operator)  
7 [Op.strictRight]: [1, 2]   // >> [1, 2) (PG range  
    strictly right of operator)  
8 [Op.noExtendRight]: [1, 2] // &< [1, 2) (PG range  
    does not extend to the right of operator)  
9 [Op.noExtendLeft]: [1, 2]  // &> [1, 2) (PG range  
    does not extend to the left of operator)
```

组合

举的额外的例子，大家可以参考一下api，看书写方式，最后再看生成的sql语句


```
1  const Op = Sequelize.Op;
2
3  {
4    count: {
5      [Op.or]: {
6        [Op.lt]: 100,
7        [Op.eq]: null
8      }
9    }
10 }
11 // count < 100 OR count IS NULL
12
13 {
14   createdAt: {
15     [Op.lt]: new Date(),
16     [Op.gt]: new Date(new Date() - 24 * 60 * 60
17 * 1000)
18   }
19 }
20 // createdAt < [timestamp] AND createdAt >
21 // [timestamp]
22 {
23   [Op.or]: [
24     {
25       title: {
26         [Op.like]: 'Boat%'
27       }
28     },
29     {
30       description: {
```

```

30         [Op.like]: '%boat%'
31     }
32 }
33 ]
34 }
35 // title LIKE 'Boat%' OR description LIKE
    '%boat%'

```

别名

Sequelize允许将特定字符串设置为运算符的别名。使用v5，这将为您提供弃用警告。

```

1  const Op = Sequelize.Op;
2  const operatorsAliases = {
3      $gt: Op.gt
4  }
5  const sequelize = new Sequelize(db, user, pass,
6      { operatorsAliases });
7  Books.findAll({
8      where:{
9          price:{
10             $gt:10 //价格大于10
11         }
12     }
13 })

```

所有的别名签名

```

1  const Op = Sequelize.Op;

```

```
2  const operatorsAliases = {
3    $eq: Op.eq,
4    $ne: Op.ne,
5    $gte: Op.gte,
6    $gt: Op.gt,
7    $lte: Op.lte,
8    $lt: Op.lt,
9    $not: Op.not,
10   $in: Op.in,
11   $notIn: Op.notIn,
12   $is: Op.is,
13   $like: Op.like,
14   $notLike: Op.notLike,
15   $iLike: Op.iLike,
16   $notILike: Op.notILike,
17   $regexp: Op.regexp,
18   $notRegexp: Op.notRegexp,
19   $iRegexp: Op.iRegexp,
20   $notIRegexp: Op.notIRegexp,
21   $between: Op.between,
22   $notBetween: Op.notBetween,
23   $overlap: Op.overlap,
24   $contains: Op.contains,
25   $contained: Op.contained,
26   $adjacent: Op.adjacent,
27   $strictLeft: Op.strictLeft,
28   $strictRight: Op.strictRight,
29   $noExtendRight: Op.noExtendRight,
30   $noExtendLeft: Op.noExtendLeft,
31   $and: Op.and,
32   $or: Op.or,
```

```
33     $any: Op.any,  
34     $all: Op.all,  
35     $values: Op.values,  
36     $col: Op.col  
37 };  
38 const sequelize = new Sequelize(db, user, pass,  
    { operatorsAliases });
```

分页

```
1 //取前2条  
2 Books.findAll({ limit: 2 })  
3  
4 // 从第8条数据开始向后取所有  
5 Books.findAll({ offset: 8 })  
6  
7 // 从第5条数据开始向后取5条  
8 Books.findAll({ offset: 5, limit: 5 })
```

排序

```
1 Books.findAll({  
2     order:[  
3         'id'//根据id正序  
4         ['id','desc'] //根据id倒序  
5     ]  
6 })
```

如何让异步代码同步化

为了严格控制语句的执行，我们可以使用async+await来严格控制变量的输出

```
1  ...then(async () => {
2      let books = await Books.findAll();
3      books = await Books.findAll({
4          where: {
5              price: {
6                  // 价格大于10块钱的
7                  [Op.gt]: 10
8              }
9          },
10         order: [
11             // "id", //根据id正序
12             ["id", 'desc'] //根据id倒序
13         ],
14         limit: 2, //返回个数
15         // attributes: [[sequelize.fn('COUNT',
16         // sequelize.col('name')), 'no_name']], //返回的字段
17         attributes: { exclude: ['id'] }, //删除选
18         定的字段
19     })
20     await Books.update({ price: 5 }, { where: {
21         id: 1 } })
22     await Books.destroy({ where: { id: 1 } })
23     console.log(JSON.stringify(books));
24 })
```

关联

有四种类型的

1. BelongsTo
2. HasOne
3. HasMany
4. BelongsToMany

一对多

一对多（或多对一）：一个出版社可以出版多本书。看图说话。 关联方式：foreign key

| 书book | | fk | | |
|-------|-------|----------|--|--------------|
| id | name | press_id | | |
| 1 | 九阳神功 | 1 | | |
| 2 | 九阴真经 | 2 | | 出版社press |
| 3 | 九阴白骨爪 | 2 | | id |
| 4 | 孤独九剑 | 3 | | name |
| 5 | 降龙十八掌 | 2 | | 1 北京工业地雷出版社 |
| 6 | 葵花宝典 | 3 | | 2 人民音乐不好听出版社 |
| | | | | 3 沙河地铁出版社 |

```
1 const Sequelize = require('sequelize');
2 // 建立连接
3 const conn = new Sequelize('db1', 'root', '', {
4   host: 'localhost',
5   dialect: 'mysql'
6 })
7 const Book = conn.define('book', {
8   name: {
9     type: Sequelize.STRING,
10    allowNull: false
11  }
12 }, {
13   timestamps: false,
```

```
14         freezTableName: true, // 表名冻结
15     })
16     const Press = conn.define('press', {
17         name: {
18             type: Sequelize.STRING,
19             allowNull: false
20         }
21     }, {
22         timestamps: false,
23         freezTableName: true, // 表名冻结
24     })
25     // 1对多 成立
26     Press.hasMany(Book); // Will add pressId to Book
27     // model 则会添加
28     Book.belongsTo(Press); // Will also add userId
29     // to Task model
30     // 同步
31     conn.sync({ force: true }).then(async () => {
32         // 先创建主表
33         await Press.bulkCreate([
34             { name: '北京工业地雷出版社' },
35             { name: "朝阳大悦城出版社" },
36             { name: '沙河地铁出版' },
37         ])
38         // 再创建从表
39         await Book.bulkCreate([
40             { name: '九阳神功', pressId: 1 },
41             { name: '九阴真经', pressId: 2 },
42             { name: '九阴白骨爪', pressId: 2 },
43             { name: '孤独九剑', pressId: 3 },
44             { name: '降龙十八掌', pressId: 2 },
```

```

43         { name: '葵花宝典', pressId: 3 },
44     ])
45
46
47     const presses = await
Press.findAll({include:[Book]});
48     console.log(JSON.stringify(presses,null,2));
49
50     const books = await Book.findOne({where:
{name:'葵花宝典'},include:[Press]});
51     console.log(JSON.stringify(books, null, 2));
52
53
54 })

```

```

1  //1.第一条查询的记录
2  [
3      {
4          "id": 1,
5          "name": "北京工业地雷出版社",
6          "books": [
7              {
8                  "id": 1,
9                  "name": "九阳神功",
10                 "pressId": 1
11             }
12         ]
13     },
14     {
15         "id": 2,
16         "name": "朝阳大悦城出版社",

```



```
17     "books": [  
18         {  
19             "id": 2,  
20             "name": "九阴真经",  
21             "pressId": 2  
22         },  
23         {  
24             "id": 3,  
25             "name": "九阴白骨爪",  
26             "pressId": 2  
27         },  
28         {  
29             "id": 5,  
30             "name": "降龙十八掌",  
31             "pressId": 2  
32         }  
33     ]  
34 },  
35 {  
36     "id": 3,  
37     "name": "沙河地铁出版",  
38     "books": [  
39         {  
40             "id": 4,  
41             "name": "孤独九剑",  
42             "pressId": 3  
43         },  
44         {  
45             "id": 6,  
46             "name": "葵花宝典",  
47             "pressId": 3
```

```
48     }
49   ]
50 }
51 ]
52 //2.第二条查询的记录
53 {
54   "id": 6,
55   "name": "葵花宝典",
56   "pressId": 3,
57   "press": {
58     "id": 3,
59     "name": "沙河地铁出版"
60   }
61 }
62
63
```