# 这里的所谓ORM是指培训机构自己封装的orm并不是网上那些orm第三方库。其实我们可以使用第三方库如prisma，mongoose，squelize等等

# 1.ORM介绍

|  |
| --- |
|  |

# 2.ORM的使用

|  |
| --- |
|  |

## 老师封装的orm文件

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

### 注意：使用自己的配置可能和老师的有点不一样，不过大体使用方法还是一致的。

# 我在网上找到一个比较像的库，和老师的可能不太一样，不过没有关系，可以自己学习研究，学习不一定要只会照抄，这个库的内容如下

|  |
| --- |
| const mysql = require('mysql');  //数据库配置  let options = {};  let tableSQL = '';  let isConnect = false;  function Model(name, option) {      this.name = name;      this.option = option;  };  /\*\*  \* @description: 查询数据  \* @param {} options：可选参数  \* @param {Function} callback :（err,results）=>{}  \*/  Model.prototype.find = function (options, callback) {      if (!isConnect) {          this.connect(err => {              isConnect = true;              var str = '';              if (!callback) {                  str = `select \* from ${this.name}`;                  callback = options;              } else if (options.constructor == Array) {                  str = `select ${options.join()} from ${this.name}`;              } else {                  str = `select \* from ${this.name} where ${options}`;              };              //console.log(str);              connection.query(str, (error, results, fields) => {                  // console.log(error.sqlState);                  if (error && error.sqlState == '42S02') {                      callback('表格不存在', []);                  } else {                      callback(error, results, fields);                  };              });              return this;          })      } else {          var str = '';          if (!callback) {              str = `select \* from ${this.name}`;              callback = options;          } else if (options.constructor == Array) {              str = `select ${options.join()} from ${this.name}`;          } else {              str = `select \* from ${this.name} where ${options}`;          };          //console.log(str);          connection.query(str, (error, results, fields) => {              // console.log(error.sqlState);              if (error && error.sqlState == '42S02') {                  callback('表格不存在', []);              } else {                  callback(error, results, fields);              };          });          return this;      }  };  /\*\*  \* @description: 分页查询  \* @param {Object} options :   { where:查询条件, number: 当前页数 , count : 每页数量 }  \* @return:  \*/  Model.prototype.limit = function (options, callback) {      var str = '';      if (!options.where) {          str = `select \* from ${this.name} limit ${(options.number - 1) \* options.count},${options.count}`;      } else {          str = str = `select \* from ${this.name} where ${options.where} limit ${(options.number - 1) \* options.count},${options.count}`;      };      console.log(str);      connection.query(str, (error, results, fields) => {          callback(error, results, fields);      });      return this;  };  /\*\*  \* @description: 插入数据  \* @param {Object} obj:对象或者数组  \* @param {Function} callback :（err,results）=>{}  \*/  Model.prototype.insert = function (obj, callback) {      if (!isConnect) {          this.connect(err => {              if (err) {                  throw err;              } else {                  connection.query(tableSQL, (error, results, fields) => {                      if (Array.isArray(obj)) {                          for (var i = 0; i < obj.length; i++) {                              this.insertObj(obj[i], callback)                          }                      } else {                          this.insertObj(obj, callback)                      }                  });              }          });      } else {          if (Array.isArray(obj)) {              for (var i = 0; i < obj.length; i++) {                  this.insertObj(obj[i], callback)              }          } else {              this.insertObj(obj, callback)          }      }  };  Model.prototype.insertObj = function (obj, callback) {      let keys = [];      let values = '';      for (var key in obj) {          keys.push(key);          values += `"${obj[key]}",`;      };      values = values.replace(/,$/, '');      let str = `INSERT INTO ${this.name} (${keys.join()}) VALUES (${values})`;      connection.query(str, (error, results, fields) => {          callback(error, results);      });  }  /\*\*  \* @description: 更新数据  \* @param {Object} option：可选参数 更新条件  \* @param {Object} obj： 修改后的数据  \* @param {Function} callback :（err,results）=>{}  \*/  Model.prototype.update = function (option, obj, callback) {      let str = '';      if (arguments.length == 2) {          callback = obj;          obj = option;          str = `UPDATE ${this.name} SET `;          for (var key in obj) {              str += `${key}='${obj[key]}', `;          };          str = str.replace(/(, )$/, '');      } else {          str = `UPDATE ${this.name} SET `;          for (var key in obj) {              str += `${key}='${obj[key]}', `;          };          str = str.replace(/(, )$/, '');          str += ` where ${option}`;      };      console.log(str);      connection.query(str, (error, results, fields) => {          callback(error, results, fields);      });      return this;  };  /\*\*  \* @description: 删除数据  \* @param {Object} option：可选参数 删除条件  \* @param {Function} callback :（err,results）=>{}  \*/  Model.prototype.delete = function (option, callback) {      var str = '';      if (!callback) {          str = `delete from ${this.name}`;          callback = option;      } else {          str = `delete from ${this.name} where ${option}`;      };      console.log(str);      connection.query(str, (error, results, fields) => {          callback(error, results, fields);      });      return this;  };  /\*\*  \* @description: 执行sql语句  \* @param {String} str : sql语句  \* @param {Function} callback :（err,results）=>{}  \*/  Model.prototype.sql = function (str, callback) {      connection.query(str, (error, results, fields) => {          callback(error, results, fields);      });      return this;  };  /\*\*  \* @description: 删除model表格 （慎用！）  \* @param {type}  \* @return:  \*/  Model.prototype.drop = function (callback) {      connection.query(`DROP TABLE ${this.name}`, (error, results, fields) => {          callback(error, results, fields);      });      return this;  };  //连接检测  Model.prototype.connect = function (callback) {      let p1 = new Promise((resolve, reject) => {          connection.connect((err) => {              if (err) {                  //console.log(err.stack);                  //console.log(err);//42000 数据库不存在  28000账号错误                  //console.log(err.sqlState);//42000 数据库不存在  28000账号错误                  reject(err);              } else {                  resolve();              }          });      });      p1.then(() => {          callback(null);      }, err => {          if (err.sqlState == 42000) {              createDatabase(callback);          } else if (err.sqlState == 28000) {              callback('数据库账号或密码错误');          } else {              callback(err);          }      });  };  //创建数据库  let createDatabase = function (callback) {      let p2 = new Promise((resolve, reject) => {          connection = mysql.createConnection({              host: options.host,//数据库地址              port: options.port,//端口号              user: options.user,//用户名，没有可不填              password: options.password,//密码，没有可不填          });          connection.connect((err) => {              //if (err) throw error;              if (err) {                  reject(err);              } else {                  resolve();              }          });      });      let p3 = new Promise((resolve, reject) => {          connection.query(`CREATE DATABASE ${options.database}`, (err, results, fields) => {              //if (error) throw error;              if (err) {                  reject(err);              } else {                  resolve();              }          });      });      let p4 = new Promise((resolve, reject) => {          connection.query(`use ${options.database}`, (err, results, fields) => {              if (err) {                  reject(err);              } else {                  resolve();              }          });      });      let pAll = Promise.all([p2, p3, p4]);      pAll.then(() => {          callback(null);      }).catch((err) => {          callback(err);      });  }    let ormTool = {      /\*\*      \* @description:连接数据库      \* @param {String} host: 主机名 默认localhost      \* @param {Number} port: 端口号 默认3306      \* @param {String} user: 用户名      \* @param {String} password: 密码      \* @param {String} database: 数据库名称 默认hm      \* @return:      \*/      connect: function ({ host = 'localhost', port = 3306, user = '', password = '', database = 'hm' }) {          databaseName = database;//全局存储当前数据库名称          options = {              host,//数据库地址              port,//端口号              user,//用户名，没有可不填              password,//密码，没有可不填              database//数据库名称          };          connection = mysql.createConnection(options);      },      /\*\*      \* @description:创建model (表格模型对象)      \* @param {String} name:表格名称      \* @param {Object} options:表格数据结构      \* @return: Model对象：负责数据库增删改查      \*/      model: function (name, options) {          let str = 'id int primary key auto\_increment, ';          for (var key in options) {              if (options[key] == Number) {                  str += `${key} numeric,`;              } else if (options[key] == Date) {                  str += `${key} timestamp,`;              } else {                  str += `${key} varchar(255),`;              }          };          str = str.replace(/,$/, '');          //console.log(`CREATE TABLE ${name} (${str})`);          //console.log(str);          tableSQL = `CREATE TABLE ${name} (${str})`;          return new Model(name, options);      }  };  module.exports = ormTool; |

# 我的项目结构如下：

|  |
| --- |
|  |

# 下面是调用的测试代码：02-mysql-orm.js，使用的数据库是qd22，有2张表格：students和classes

|  |
| --- |
| let ormTool = require('./db/nodejs-orm/index')  // console.log(hm);  let orm\_config = {      host: 'localhost',//数据库地址      port:'3306',      user: 'root',//用户名，没有可不填      password: 'root',//密码，没有可不填      database: 'qd22'//数据库名称  }  ormTool.connect(orm\_config)  let stuModel = ormTool.model('students',{      id:Number,      name:String,      age:Number,      height:Number,      gender:String,      cls\_id:Number,      is\_deleted:Number  })  // console.log(stuModel);  //新增 ok  stuModel.insert({      id:0,      name:"Jerry",      age:16,  height:167.9,  gender:1,      cls\_id:1,      is\_deleted:0  },(err,result)=>{      if(err){          console.log(err);      }      console.log(result);      console.log('添加成功');  })  //新增多条数据  let stus = [      {id:0,name:"Jerrylee",age:16,height:167.9,gender:1,cls\_id:1,is\_deleted:0},      {id:0,name:"Tom",age:16,height:171.9,gender:1,cls\_id:2,is\_deleted:0},      {id:0,name:"Goofy",age:17,height:175.9,gender:1,cls\_id:1,is\_deleted:0},      {id:0,name:"Twitter",age:15,height:160.9,gender:2,cls\_id:2,is\_deleted:0},  ]  stuModel.insert(stus,(err,ret)=>{      console.log(ret);  })  //查找全部，ok  stuModel.find((err,res)=>{      console.log(res);  })  //查询指定字段需要用把字段放入[]中  stuModel.find(['name','age','gender','height'],(err,res)=>{          console.log(res.length);          console.log(res);  })  //聚合函数,  //--1.查询数据库一共又多少条记录  stuModel.find(['count(\*)'],(err,res)=>{          console.log(res.length);          console.log(res);  })  //--2.查询学生的最大年龄  stuModel.find(['max(age)'],(err,res)=>{          console.log(res.length);          console.log(res);  })  //--3.查询身高最矮的学生  stuModel.find(['min(height)'],(err,res)=>{          console.log(res.length);          console.log(res);  })  // 根据条件查询数据  // 'id=1' : 查询id为1的数据 (查询条件可以参考sql语句)  //例如 'age>10' : 查询age超过10的数据  //例如 'name="张三"' : 查询名字为张三的数据，注意字符串添加引号  stuModel.find('name="Mary"',(err,res)=>{ //ok          console.log(res);  })  //逻辑运算符，查询id比2大同时又比10小的学生 ok  stuModel.find('id>2 and id<10',(err,res)=>{          console.log(res);  })  //模糊查询,查找名字至少3个字的学生，注意字符串里面需要又双引号的 ok  stuModel.find('name like "\_\_\_%"',(err,res)=>{          console.log(res);  })  //查询年龄为13岁和17岁的所有学生，查询不连续记录  stuModel.find('age in (13,17)',(err,res)=>{      console.log(res);  })  //查询年龄在13岁和17岁之间的所有学生包括13岁17岁  stuModel.find('age between 13 and 17',(err,res)=>{      console.log(res.length);      console.log(res);  })  stuModel.find('age between 13 and 17',(err,res)=>{      console.log(res.length);      console.log(res);  })  //对于一些比较复杂的查询，可以直接输入sql语句来调用,调用sql方法如查询所有男生的平均身高ok  stuModel.sql('select avg(height) from students where gender=1',(err,res)=>{          console.log(res.length);          console.log(res);  })  //更新数据，update方法有3个参数，1条件，2新数据，3回调  //将id为1的学生改名为Marco  stuModel.update('id=1',{name:'Marco'},(err,res)=>{              console.log(res);  })  //删除数据,在生产环境中尽量不要删除，学习环境没有关系  //删除名字为Jerry的学生  stuModel.delete('name="Jerry"',(err,res)=>{                  console.log(res);  }) |

# 实战：我们把MySQL-demo1项目复制一份，改名：MySQL-demo2-self-orm，然后用vscode打开这个项目

# 我们对项目做如下修改，

# 1.用我们下载的orm库替换原来的db.js文件

|  |
| --- |
|  |

# 2.新建一个config文件夹里面新建一共config.js把数据库的配置写在这里并且导出为一给对象

|  |
| --- |
|  |

# 3.把01-mysql-app.js复制一份改名03-mysql-app-orm.js，然后把数据库操作的代码改为使用orm库的代码

|  |
| --- |
| let path = require('path')  let express = require('express')  let db = require('./db/nodejs-orm/index') //导入orm模块  let orm\_config = require('./config/config') //导入数据库配置  let port = 3000  let app = express()  //app.use(express.static(\_\_dirname+'/public'))//静态服务器，根据需要配置  //配置模板引擎和视图路径  app.engine('html',require('express-art-template'))  app.set('view options',{      debug: process.env.NODE\_ENV!='production'  })  app.set('views',path.join(\_\_dirname,'views'))  app.set('view engine','html')  app.get('/',(req,res)=>{       res.render('index')  })  //连接数据库  db.connect(orm\_config)  //创建学生模型  let stuModel = db.model('students',{      id:Number,      name:String,      age:Number,      height:Number,      gender:String,      cls\_id:Number,      is\_deleted:Number  })  //获取学生列表的路由  app.get('/get\_data',(req,res)=>{      stuModel.find((err,data)=>{          res.render('student-list',{students:data})      })  })  //显示学生详情信息的路由  app.get('/student\_detail/:id',(req,res)=>{      let id = req.params.id      // let sql = 'select \* from students where id=' + id +";"      // //查询数据库，返回到浏览器        stuModel.find('id='+id,(err,data)=>{          res.render('detail',{students:data})      })    })  app.listen(port,()=>{       console.log('server is ready :http://localhost:'+port+'/');  }) |

# 4.views文件夹里面的html文件不用修改

## 效果

|  |
| --- |
|  |
|  |
|  |

## 和没有使用orm库的效果是一样的

# 我们来给这个案例增加一共添加新学生的功能

# 5.在views文件夹里面新建一共new.html文件，内容如下

|  |
| --- |
|  |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>添加学生</title>  </head>  <body>      <form action="/add\_stu" method="post">          <h1添加学生< /h1>              姓名:              <input type="text" name="name" value="">              <br><br>              年龄:              <input type="text" name="age" value="">              <br><br>              性别:              <input type="radio" name="gender" value="1" checked>男              <input type="radio" name="gender" value="2">女              <br><br>              身高:              <input type="text" name="height" value="">              <br><br>              班级id:              <input type="text" name="cls\_id">              <br><br>              <input type="submit" value="提交">              <input type="reset">      </form>  </body>  </html> |

# 6.在03-mysql-app-orm.js中添加新增页面的渲染路由/new和提交数据的处理路由/add\_stu

|  |
| --- |
|  |

# 7.此时新增是没有效果的，因为服务器压根拿不到数据，我们添加引入body-parser的插件和注册bodyParser中间件的代码

|  |
| --- |
|  |

## 测试一下，拿到数据了

|  |
| --- |
|  |

# 8.修改一下/add\_stu路由，把数据存入数据库，然后返回列表

|  |
| --- |
| let path = require('path')  let express = require('express')  let bodyParser = require('body-parser') //导入bodyParser中间件  let db = require('./db/nodejs-orm/index') //导入orm模块  let orm\_config = require('./config/config') //导入数据库配置  let port = 3000  let app = express()  //注册bodyParser中间件  app.use(bodyParser.urlencoded({extended:false}))  //app.use(express.static(\_\_dirname+'/public'))//静态服务器，根据需要配置  //配置模板引擎和视图路径  app.engine('html',require('express-art-template'))  app.set('view options',{      debug: process.env.NODE\_ENV!='production'  })  app.set('views',path.join(\_\_dirname,'views'))  app.set('view engine','html')  app.get('/',(req,res)=>{       res.render('index')  })  //连接数据库  db.connect(orm\_config)  //创建学生模型  let stuModel = db.model('students')  //获取学生列表的路由  app.get('/get\_data',(req,res)=>{      //查询所有      stuModel.find((err,data)=>{          res.render('student-list',{students:data})      })  //    //分页查询  //    stuModel.limit({number:2,count:5},(err,data)=>{  //      res.render('student-list',{students:data})  //    })  })  //显示学生详情信息的路由  app.get('/student\_detail/:id',(req,res)=>{      let id = req.params.id      // let sql = 'select \* from students where id=' + id +";"      // //查询数据库，返回到浏览器        stuModel.find('id='+id,(err,data)=>{          res.render('detail',{students:data})      })    })  //渲染新增学生页面  app.get('/new',(req,res)=>{      res.render('new')  })  //实现新增学生功能的路由  app.post('/add\_stu',(req,res)=>{      // console.log(req.body);      //获取表单数据      let stu = req.body      stuModel.insert(stu,(err,data)=>{         if(err){          console.log(err);          return         }         res.redirect('/get\_data')      })    })  app.listen(port,()=>{       console.log('server is ready :http://localhost:'+port+'/');  }) |

## 新增路由的核心代码

|  |
| --- |
|  |

## 测试一下

|  |
| --- |
|  |
|  |

# 我们把学生列表改进一下，给每一条学生记录添加修改和删除功能

# 9.打开student-list.html文件，添加修改和删除两个链接，随便给a标签添加一点点样式

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>学生列表</title>      <style>          a {              text-decoration: none;              font-weight: 540;          }          a:link {              color: rgb(24, 22, 22);              font-size: 18px;          }          a:visited {              color: deepskyblue;          }          a:hover {              color: lime;          }          a:active {              color: skyblue;          }      </style>  </head>  <body>      <h2>学生列表</h2>      <hr>      <table>          {{each students}}          <tr>              <td>学生姓名：</td>              <td>{{$value.name}}</td>              <td>详情：</td>              <td><a href="/student\_detail/{{$value.id}}">学生详情</a></td>              <td><a href="/edit\_student/{{$value.id}}">修改</a></td>              <td><a href="/delete\_student/{{$value.id}}">删除</a></td>          </tr>          {{/each}}      </table>      <a href="/new">添加学生</a>  </body>  </html> |

# 10.在views文件夹下面新建一个edit.html文件，内容如下，它需要我们传递一个student对象给他渲染数据

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>编辑学生信息</title>  </head>  <body>      <form action="/edit\_student" method="post">          <h1添加学生< /h1>              <input type="hidden" name="id" value={{student.id}}> <!-- 这个很重要否则post路由无法更新数据 -->              姓名:              <input type="text" name="name" value={{student.name}}>              <br><br>              年龄:              <input type="text" name="age" value={{student.age}}>              <br><br>              性别:              <input type="text" name="gender" value={{student.gender}}>              <br><br>              身高:              <input type="text" name="height" value={{student.height}}>              <br><br>              班级id:              <input type="text" name="cls\_id" value={{student.cls\_id}}>              <br><br>              <input type="submit" value="提交">              <input type="reset">      </form>  </body>  </html> |

## 小技巧：必须用一个隐藏字段把学生的id传递给后端，否则后端无法处理数据

# 11.在03-mysql-app-orm.js文件中添加对修改路由的响应，先添加get路由/edit\_student/:id，用来获取学生的id，然后查询出学生信息，再用获取到的学生信息来渲染edit.html模板

|  |
| --- |
|  |

# 12.接下来我们需要一个/edit\_student的post路由来在数据库更新学生信息，更新成功后跳转到学生列表页面

|  |
| --- |
|  |

# 13.我们来做删除功能，其实在实际开发中删除只是给记录打上一个删除标记并不会真正删除数据，我们来添加删除路由/delete\_student/:id

|  |
| --- |
|  |

## 测试一下，页面没有变化，我们用命令行查看数据库，发现小牛牛的is\_deleted被修改为1

|  |
| --- |
|  |

# 14.按照规则，is\_deleted为1，的学生不能被显示，所有我们来修改一下/get\_data路由的处理函数，使他只显示is\_deleted为0的学生信息

|  |
| --- |
|  |

### 此时进入/get\_data,发现小牛牛的信息没有显示了

|  |
| --- |
|  |

# 此次演练结束，03-mysql-app-orm.js的完整代码如下

|  |
| --- |
| let path = require('path')  let express = require('express')  let bodyParser = require('body-parser') //导入bodyParser中间件  let db = require('./db/nodejs-orm/index') //导入orm模块  let orm\_config = require('./config/config') //导入数据库配置  let port = 3000  let app = express()  //注册bodyParser中间件  app.use(bodyParser.urlencoded({extended:false}))  //app.use(express.static(\_\_dirname+'/public'))//静态服务器，根据需要配置  //配置模板引擎和视图路径  app.engine('html',require('express-art-template'))  app.set('view options',{      debug: process.env.NODE\_ENV!='production'  })  app.set('views',path.join(\_\_dirname,'views'))  app.set('view engine','html')  app.get('/',(req,res)=>{       res.render('index')  })  //连接数据库  db.connect(orm\_config)  //创建学生模型  let stuModel = db.model('students')  //获取学生列表的路由  app.get('/get\_data',(req,res)=>{      //查询所有,其实就是查询所有is\_deleted=0的学生is\_deleted=1就相当于删除了,不能做任何处理      stuModel.find('is\_deleted=0',(err,data)=>{          res.render('student-list',{students:data})      })  })  //显示学生详情信息的路由  app.get('/student\_detail/:id',(req,res)=>{      let id = req.params.id      // let sql = 'select \* from students where id=' + id +";"      // //查询数据库，返回到浏览器        stuModel.find('id='+id,(err,data)=>{          res.render('detail',{students:data})      })    })  //渲染新增学生页面  app.get('/new',(req,res)=>{      res.render('new')  })  //实现新增学生功能的路由  app.post('/add\_stu',(req,res)=>{      // console.log(req.body);      //获取表单数据      let stu = req.body      stuModel.insert(stu,(err,data)=>{         if(err){          console.log(err);          return         }         res.redirect('/get\_data')      })    })  //根据动态路由:id获取到学生id然后查询数据渲染edit页面  app.get('/edit\_student/:id',(req,res)=>{      let id = req.params.id //获取id      //根据id查询学生学习      stuModel.find("id="+id,(err,data)=>{          if(err){              console.log(err);              return;          }          res.render('edit.html',{student:data[0]}) //虽然数据是一个数组但是它只有一个元素,可以用下标0来获取      })    })  //将修改后的学生信息更新到数据库  app.post('/edit\_student',(req,res)=>{      let student = req.body //获取表单数据      //注意学生的id必须使用一个hidden字段传递过来      //更新学生信息      stuModel.update('id='+student.id,student,(err,data)=>{          if(err){              console.log(err);              return;          }          res.redirect('/get\_data')      })      })  //删除学生信息,其实就是给这个学生信息的is\_delete设置为1  app.get('/delete\_student/:id',(req,res)=>{      let id = req.params.id //获取id      //根据id查询学生学习      stuModel.update('id='+id,{is\_deleted:1},(err,data)=>{          if(err){              console.log(err);              return;          }          res.redirect('/get\_data')      })    })  app.listen(port,()=>{       console.log('server is ready :http://localhost:'+port+'/');  }) |

## 注意：这里有一个分页显示的功能，暂时没有实现，以后需要实现

、