# 上一节我们已经完成了用async+await的方式对数据的增删改查功能，并且添加了异常处理代码，但是还有一个问题，就是代码的可复用性差，我们这一节来解决这个问题

## 1.把MySQL-demo2-self-orm项目复制一份，改名：MySQL-demo2-self-orm-handleDB

|  |
| --- |
|  |

## 2.把01-04代码删除，因为我们已经有这些代码了，然后把05复制一份改名：06-mysql-app-orm-handleDB.js

|  |
| --- |
|  |

## 3.在db文件夹里面新建一个handleDB.js文件

|  |
| --- |
|  |

## 4.我们在handleDB.js文件里面写一个handleDB函数并且把它导出，此时我们先测试一下查询功能

|  |
| --- |
| //封装数据库操作到一个方法中，使他成为通用方法  let db = require('./nodejs-orm/index')  async function handleDB(res){  //    console.log("handleDB test...");      let stuModel = db.model('students')      try {          result =await new Promise((resolve,reject)=>{          stuModel.find('is\_deleted=0',(err,data)=>{          if(err){              reject(err)          }          resolve(data)          })       })      } catch (error) {          console.log(error);          res.send({errMsg:'数据库查询错误'})          return;      }    return result  }  module.exports = handleDB |

## 5.然后我们在06-mysql-app-orm-handleDB.js中调用这个代码

|  |
| --- |
|  |

### 测试一下，是可以工作的

|  |
| --- |
|  |

### 不过此时其实他是没有通用性的。我们需要对他进行封装，使他具备增删改查功能，你需要告诉他需要处理能够表格，执行的操作，数据，参数个数等等

## 6.完善handleDB函数的代码如下

|  |
| --- |
| //封装数据库操作到一个方法中，使他成为通用方法  let db = require('./nodejs-orm/index')  /\*\*   \*   \* @param {\*} res 服务器的响应对象   \* @param {\*} tableName 需要处理的表名   \* @param {\*} methodName 操作方法如find，insert，update，delete等等   \* @param {\*} n1 回调函数前面的第一个参数，是可选的   \* @param {\*} n2 回调函数前面的第二个参数，也是可选的   \* @returns  返回操作结果   \* 需要根据n1和n2来确定操作方式，更新是n1和n2都不为undefined，insert是只需要n1，n2可以为undefine(也就是不传给它)，如果n1和n2   \* 都为undefined，可以做查询所有数据的操作或者删除所有数据的操作   \*/  async function handleDB(res,tableName,methodName,n1,n2){  //    console.log("handleDB test...");      let Model = db.model(tableName)        try {          result =await new Promise((resolve,reject)=>{          if(!n1){ //连n1都没有传递，n2肯定也是undefined，此时做查询全部或者删除全部的操作               Model[methodName]('is\_deleted=0',(err,data)=>{              if(err){                  reject(err)              }              resolve(data)              })              return //做完一个查询后，需要返回          }          //如果代码能够执行到这里，说明有n1了，我们需要判断n2有没有          if(!n2){              Model[methodName](n1,(err,data)=>{                  if(err){                      reject(err)                  }                  resolve(data)              })              return          }          //如果代码能够执行到这里，说明有n1和n2了          Model[methodName](n1,n2,(err,data)=>{              if(err){                  reject(err)              }              resolve(data)          })         })      } catch (error) {          console.log(error);          res.send({errMsg:'数据库查询错误'})          return;      }    return result  }  module.exports = handleDB |

## 7.然后在06-mysql-app-orm-handleDB.js里面调用handleDB函数，代码变得非常简洁，我们把以前的代码都删除了，因为以前版本的程序有这些代码

|  |
| --- |
| 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 handleDB = require('./db/handleDB.js') //导入自己写的数据库通用操作函数  let port = 3001  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)  //获取学生列表的路由  app.get('/get\_data',(req,res)=>{      (async function get\_data(){          let result = await handleDB(res,'students','find','is\_deleted=0')          res.render('student-list',{students:result})      })() //立即调用函数  })  //显示学生详情信息的路由  app.get('/student\_detail/:id',(req,res)=>{      (async function(){          let id = req.params.id          let result = await handleDB(res,'students','find','id='+id)          res.render('detail',{students:result})      })()    })  //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)=>{      (async function(){          let student = req.body          handleDB(res,'students','insert',student)          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)=>{         //获取id      // let result      (async function(){          let id = req.params.id          let result = await handleDB(res,'students','find',"id="+id)          res.render('edit.html',{student:result[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)=>{     (async function(){        let student = req.body        await handleDB(res,'students','update','id='+student.id,student)        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)=>{      (async function(){          let id = req.params.id          await handleDB(res,'students','update','id='+id,{is\_deleted:1})          res.redirect('/get\_data')      })()  })  app.listen(port,()=>{       console.log('server is ready :http://localhost:'+port+'/');  }) |

### 最后，把views/student-list.html修改一下

|  |
| --- |
| <!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>          <span>学生列表 </span> &nbsp;&nbsp;&nbsp;          <h3>              <b><a href="/new">添加学生</a>&nbsp;&nbsp;&nbsp;</b>              <b><a href="/">返回首页</a></b>          </h3>      </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>  </body>  </html> |

### 效果是一样的

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

## 我们发现老师有一个扩展方法我没有，所以我自己给nodejs-orm库添加了一个findExtended方法，nodejs-orm/index.js的最新内容如下

|  |
| --- |
| 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;      }  };  //find方法扩展可以这么调用：stuModel.findExtended({where:'age<18',arr:["name","age",'gender']},(err,data)=>{...})  Model.prototype.findExtended = function (options, callback) {      if (!isConnect) {          this.connect(err => {              isConnect = true;              var str = '';              if (!callback) {                  str = `select \* from ${this.name}`;                  callback = options;              } else if (options.where && options.arr) {                  str = `select ${options.arr.join()} from ${this.name} where ${options.where}`;              } 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; |

# handleDB还有一个问题，就是出错信息是显示问题，我们需要给他传递一个错误信息，我们修改一下handleDB.js

|  |
| --- |
| //封装数据库操作到一个方法中，使他成为通用方法  let db = require('./nodejs-orm/index')  /\*\*   \*   \* @param {\*} res 服务器的响应对象   \* @param {\*} tableName 需要处理的表名   \* @param {\*} methodName 操作方法如find，insert，update，delete等等   \* @param {\*} errMsg 错误信息必须放在n1和n2前面，否则报错因为n1和n2必须放最后   \* @param {\*} n1 回调函数前面的第一个参数，是可选的   \* @param {\*} n2 回调函数前面的第二个参数，也是可选的   \* @returns  返回操作结果   \* 需要根据n1和n2来确定操作方式，更新是n1和n2都不为undefined，insert是只需要n1，n2可以为undefine(也就是不传给它)，如果n1和n2   \* 都为undefined，可以做查询所有数据的操作或者删除所有数据的操作   \*/  async function handleDB(res,tableName,methodName,errMsg,n1,n2){  //    console.log("handleDB test...");      let Model = db.model(tableName)        try {          result =await new Promise((resolve,reject)=>{          if(!n1){ //连n1都没有传递，n2肯定也是undefined，此时做查询全部或者删除全部的操作               Model[methodName]('is\_deleted=0',(err,data)=>{              if(err){                  reject(err)              }              resolve(data)              })              return //做完一个查询后，需要返回          }          //如果代码能够执行到这里，说明有n1了，我们需要判断n2有没有          if(!n2){              Model[methodName](n1,(err,data)=>{                  if(err){                      reject(err)                  }                  resolve(data)              })              return          }          //如果代码能够执行到这里，说明有n1和n2了          Model[methodName](n1,n2,(err,data)=>{              if(err){                  reject(err)              }              resolve(data)          })         })      } catch (error) {          console.log(error);          res.send({errMsg:errMsg})          return;      }    return result  }  module.exports = handleDB |

## 然后需要修改一下06-mysql-app-orm-handleDB.js，在调用handleDB的地方传递错误信息

|  |
| --- |
| 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 handleDB = require('./db/handleDB.js') //导入自己写的数据库通用操作函数  let port = 3001  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)=>{      (async function get\_data(){          let result = await handleDB(res,'students','find','查询数据错误','is\_deleted=0')          res.render('student-list',{students:result})      })() //立即调用函数  })  //显示学生详情信息的路由  app.get('/student\_detail/:id',(req,res)=>{      (async function(){          let id = req.params.id          let result = await handleDB(res,'students','find','查询数据错误','id='+id)          res.render('detail',{students:result})      })()    })  //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)=>{      (async function(){          let student = req.body          await handleDB(res,'students','insert',"插入数据错误",student,)          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)=>{        (async function(){          //获取id          let id = req.params.id          let result = await handleDB(res,'students','find','查询数据错误',"id="+id)          res.render('edit.html',{student:result[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)=>{     (async function(){        let student = req.body        await handleDB(res,'students','update','更新数据错误','id='+student.id,student)        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)=>{      (async function(){          let id = req.params.id          await handleDB(res,'students','update','删除数据错误','id='+id,{is\_deleted:1})          res.redirect('/get\_data')      })()  })  app.listen(port,()=>{       console.log('server is ready :http://localhost:'+port+'/');  }) |

### 注意：有些参数是必须放在最后的，然后handleDB函数中的n1和n2，如果你改变了位置，就会报错！！！

## 其他文件的内容查看：[上两节内容](node框架Express+mysql笔记06-使用ORM操作mysql使用async+await.docx)

# 这一节学习结束