## ****Egg框架通过****[egg-sequelize](https://links.jianshu.com/go?to=https://link.zhihu.com/?target=https%3A//github.com/eggjs/egg-sequelize" \t "_blank) 插件连接mysql数据库保存并查询数据

## ****初始化项目****

通过 egg-init 初始化一个项目:

egg-init --type=simple --dir=sequelize-project

cd sequelize-project

npm i

安装并配置 [egg-sequelize](https://links.jianshu.com/go?to=https://link.zhihu.com/?target=https%3A//github.com/eggjs/egg-sequelize" \t "_blank) 插件（它会辅助我们将定义好的 Model 对象加载到 app 和 ctx 上）和 [mysql2](https://links.jianshu.com/go?to=https://link.zhihu.com/?target=https%3A//github.com/sidorares/node-mysql2" \t "_blank) 模块：

* 安装

npm install --save egg-sequelize mysql2

* 在 config/plugin.js 中引入 egg-sequelize 插件

exports.sequelize = {

enable: true,

package: 'egg-sequelize',};

* 在 config/config.default.js 中编写 sequelize 配置

config.sequelize = {

dialect: 'mysql',

host: '127.0.0.1',

port: 3306,

database: 'egg-sequelize-doc-default',

};

我们可以在不同的环境配置中配置不同的数据源地址，用于区分不同环境使用的数据库，例如我们可以新建一个 config/config.unittest.js 配置文件，写入如下配置，将单测时连接的数据库指向 egg-sequelize-doc-unittest。

exports.sequelize = {

dialect: 'mysql',

host: '127.0.0.1',

port: 3306,

database: 'egg-sequelize-doc-unittest',};

完成上面的配置之后，一个使用 sequelize 的项目就初始化完成了。[egg-sequelize](https://links.jianshu.com/go?to=https://link.zhihu.com/?target=https%3A//github.com/eggjs/egg-sequelize" \t "_blank) 和 [sequelize](https://links.jianshu.com/go?to=https://link.zhihu.com/?target=http%3A//docs.sequelizejs.com/" \t "_blank) 还支持更多的配置项，可以在他们的文档中找到。

## ****初始化数据库和 Migrations****

* 安装 sequelize-cli

npm install --save-dev sequelize-cli

在 egg 项目中，我们希望将所有数据库 Migrations 相关的内容都放在 database 目录下，所以我们在项目根目录下新建一个 .sequelizerc 配置文件：

'use strict';

const path = require('path');

module.exports = {

config: path.join(\_\_dirname, 'database/config.json'),

'migrations-path': path.join(\_\_dirname, 'database/migrations'),

'seeders-path': path.join(\_\_dirname, 'database/seeders'),

'models-path': path.join(\_\_dirname, 'app/model'),};

* 初始化 Migrations 配置文件和目录

npx sequelize init:config

npx sequelize init:migrations

npx sequelize init:seeders

执行完后会生成 database/config.json 文件和 database/migrations 目录，我们修改一下 database/config.json 中的内容，将其改成我们项目中使用的数据库配置：

{

"development": {

"username": "root",

"password": "123456",

"database": "egg-sequelize-doc-default",

"host": "127.0.0.1",

"dialect": "mysql"

},

"test": {

"username": "root",

"password": "123456",

"database": "egg-sequelize-doc-unittest",

"host": "127.0.0.1",

"dialect": "mysql"

}}

此时 sequelize-cli 和相关的配置也都初始化好了，我们可以开始编写项目的第一个 Migration 文件来创建我们的一个 users 表了。

npx sequelize migration:generate --name=init-users

执行完后会在 database/migrations 目录下生成一个 migration 文件(${timestamp}-init-users.js)，我们修改它来处理初始化 users表：

'use strict';

module.exports = {

// 在执行数据库升级时调用的函数，创建 users 表

up: async (queryInterface, Sequelize) => {

const { INTEGER, DATE, STRING } = Sequelize;

await queryInterface.createTable('users', {

id: {

type: INTEGER,

primaryKey: true,

autoIncrement: true

},

name: STRING(30),

email: STRING(100),

phone: STRING(20),

username: STRING(30),

password: STRING(100),

created\_at: DATE,

updated\_at: DATE

});

},

// 在执行数据库降级时调用的函数，删除 users 表

down: async queryInterface => {

await queryInterface.dropTable('users');

},};

* 执行 migrate 进行数据库变更

# 运行迁移文件

npx sequelize db:migrate

# 如果有问题需要回滚，可以通过 `db:migrate:undo` 回退一个变更

npx sequelize db:migrate:undo

# 可以通过 `db:migrate:undo:all` 回退到初始状态

npx sequelize db:migrate:undo:all

## 数据填充

* 生成填充文件

npx sequelize seed:generate --name=userSeeder

执行完后会在 database/seeders目录下生成一个 migration 文件(${timestamp}-userSeeder.js)，我们修改它来填充 users表：

'use strict';

module.exports = {

up: async (queryInterface, Sequelize) => {

return queryInterface.bulkInsert('users', [{

name: '张三',

email: 'abc@qq.com',

phone: '18888888888',

username: 'zhangsan',

password: '123456',

created\_at: new Date(),

updated\_at: new Date()

}, {

name: '李四',

email: '123@qq.com',

phone: '18288888888',

username: 'lisi',

password: '123456',

created\_at: new Date(),

updated\_at: new Date()

}])

},

down: async (queryInterface, Sequelize) => {

return queryInterface.bulkDelete('users', null, {});

}};

* 运行填充文件

npx sequelize db:seed:all

* 撤销上一步填充

npx sequelize db:seed:undo

如果你想撤消特定的种子

npx sequelize-cli db:seed:undo --seed userSeeder

如果你想撤消所有的种子

npx sequelize-cli db:seed:undo:all

## 初始化模型

* 运行模型初始化命令

npx sequelize model:generate --name User --attributes name:string

执行完后会在 app/model 目录下生成一个User.js，并且会在database/migrations目录下生成一个：timestamp-create-user.js数据库迁移文件

## ****编写代码****

现在终于可以开始编写代码实现业务逻辑了，首先我们来在 app/model/ 目录下编写 user 这个 Model：

// app/model/user.js'use strict';

module.exports = app => {

const { STRING, INTEGER, DATE } = app.Sequelize;

const User = app.model.define('user', {

id: { type: INTEGER, primaryKey: true, autoIncrement: true },

name: STRING(30),

age: INTEGER,

created\_at: DATE,

updated\_at: DATE,

});

return User;};

这个 Model 就可以在 Controller 和 Service 中通过 app.model.User 或者 ctx.model.User 访问到了，例如我们编写 app/controller/users.js：

注意：访问 xxx.model.ModelName 中 ModelName 开头字母必须为大写

[Sequelize 中文API文档－1. 快速入门、Sequelize类](https://links.jianshu.com/go?to=https://link.zhihu.com/?target=https%3A//itbilu.com/nodejs/npm/VkYIaRPz-.html%23api-instance-define" \t "_blank) 查看 app.model.define('name', {attributes}, {options}) 配置项规则

// app/controller/users.jsconst Controller = require('egg').Controller;

function toInt(str) {

if (typeof str === 'number') return str;

if (!str) return str;

return parseInt(str, 10) || 0;}

class UserController extends Controller {

async index() {

const ctx = this.ctx;

const query = { limit: toInt(ctx.query.limit), offset: toInt(ctx.query.offset) };

ctx.body = await ctx.model.User.findAll(query);

}

async show() {

const ctx = this.ctx;

ctx.body = await ctx.model.User.findById(toInt(ctx.params.id));

}

async create() {

const ctx = this.ctx;

const { name, age } = ctx.request.body;

const user = await ctx.model.User.create({ name, age });

ctx.status = 201;

ctx.body = user;

}

async update() {

const ctx = this.ctx;

const id = toInt(ctx.params.id);

const user = await ctx.model.User.findById(id);

if (!user) {

ctx.status = 404;

return;

}

const { name, age } = ctx.request.body;

await user.update({ name, age });

ctx.body = user;

}

async destroy() {

const ctx = this.ctx;

const id = toInt(ctx.params.id);

const user = await ctx.model.User.findById(id);

if (!user) {

ctx.status = 404;

return;

}

await user.destroy();

ctx.status = 200;

}}

module.exports = UserController;

最后我们将这个 controller 挂载到路由上：

// app/router.js

module.exports = app => {

const { router, controller } = app;

router.resources('users', '/users', controller.users);};