从 0 开始使用 Sequelize 进行基础的数据库操作

作者：limanting
原文链接：https://ld246.com/article/1628423512557
来源网站：链滴

官方文档：https://www.sequelize.com.cn/

使用node.js做后端是一个小项目中的任务，这篇文章是我不断踩坑后总结的内容。

如果官方文档有点看不懂，可以按照这篇文章做，从0开始搭建并使用Sequelize进行简单的数据操作。

开始

Sequelize帮助我们便捷的操作数据库，在使用它之前，我们需要下好数据库，这里我是用mysql作例子。

下载 mysql　brew install mysql

启动 mysql　mysql.serve start

如果需要进入mysql 进行一些设置，可以mysql -p

关闭 mysql mysql.serve stop

链接数据库设置

配置公共的数据表设置

可以作为一个单独的文件，在描述模型的时候引用该公共配置的实例。

```
// db.js
const { Sequelize,Model } = require('sequelize');
const { dbName, host, port, user, password } = require('../config').db; // 引入配置好的数据库配


const sequelize = new Sequelize(dbName, user , password, {
  dialect: 'mysql', // 数据库的配置
  host, // 数据库的配置
  port, // 数据库的配置
  logging: false, // 或console.log, 是否输出sql语句
  timezone: '+08:00',
  define:{
    //create_time  update_time delete_time
    timestamps: true,
    paranoid: true,
    createdAt: 'creationTime',
    updatedAt: 'updated_time',
    deletedAt: 'deleted_time',
    underscored: true,
    freezeTableName: true,
  }
});

sequelize.sync({
  // 每次初始化表是否删除原表
  force: false,
  // 检查数据库中表的当前状态,进行必要的更改使其与模型匹配
```

```
  alter: true
});

module.exports = {
  sequelize
}
```

配置数据库信息

在工作时，我们会有多个环境，比如测试环境、生产环境、本地环境。为了方便的使用配置，我们将个环境的配置内容写在不同的文件中，在程序执行时根据运行环境读取对应的配置。

```
//
const localConf = require('./local');
const devConf = require('./development');
const prodConf = require('./production');

let conf = localConf;

if (process.env.NODE_ENV === 'production') {
  conf = prodConf;
}

if (process.env.NODE_ENV === 'development') {
  conf = devConf;
}

module.exports = conf;
```

配置信息举例。可以写成下面的形式，也可以直接是一个json文件。

```
module.exports = {
  env: 'local',
  db: {
    dbName: 'iconfont',
    host: '127.0.0.1',
    port: 3306,
    user: 'root',
    password: '12345678',
  },
  auth: {
    authKey: 'eUnE#9~P',
    authPass: '*3mExsa}asd@#afSx;iPqwEqasvho;',
  },
  oss: {
    cdn: "https://conan-test.oss-cn-beijing.aliyuncs.com",
    endpoint: "oss-cn-beijing-internal.aliyuncs.com",
    accessKeyId: "LTAIoRierLFA55aQ",
    accessKeySecret: "6T7jyq6UuE3wLZ69JwORltpySgEykB",
    domain: "https://oss-cn-beijing.aliyuncs.com",
    bucket: "conan-test",
    direct: 'conan-iconfont'
  }
};
```

设计数据库表

设计数据库表的时候，需要明确好实体和关系。实体进作为描述实体，关系分为一对多关系和多对多系，画出ER图。一对多关系，将多写在一的表里面；多对多关系，一定要建一个多对多关系表。

实体表删除是软删除，即没有真的删除，而是填充删除时间。对于关系表，则需要配置硬删除，即删是真的从表中删除，不然在操作数据库的时候会有约束限制造成操作失败。

建立实体表，并建立一对多关系

假设：一个图集内包含多个图标，一个图标有一个唯一对应的图集。图标是一，图集是多，我们需要图标的主键写在图集内。

图标数据表

```
const { sequelize } = require('../../core/db');// 引入公共数据表配置实例
const { Sequelize, Model } = require('sequelize');

class IconModel extends Model {}
// 模型定义
IconModel.init({
  id: {
    type: Sequelize.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  name: Sequelize.STRING, // 图标名称
  svg: Sequelize.STRING(10240),  // 图标内容<svg ..../>
  isPrivate: Sequelize.BOOLEAN  // 是否私密的标记
}, {
  sequelize,
  tableName: 'icon'
});

module.exports = { IconModel };
```

图集数据表

```
const { Sequelize, Model } = require('sequelize');
const { sequelize } = require('../../core/db');
const { IconModel } = require('./icon'); //引入图标数据表模型

class AtlasModel extends Model {}

AtlasModel.init({
  id: {
    type: Sequelize.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  name: Sequelize.STRING, // 图集名称
  atlasPrefix: Sequelize.STRING, // 图标库前缀
  fontOssUrl: Sequelize.STRING, // css文件cdn地址
```

```
  isPrivate: Sequelize.BOOLEAN, // 是否私密 私密图集：true  公开图集：false
}, {
  sequelize,
  tableName: 'atlas'
});
// 定义关系
IconModel.belongsTo(AtlasModel); // 图标 属于 图集 ,会在数据库的图集表中存一个icon_model_i
 ,在代码中我们只需要调用函数，不需要关注这个细节，具体使用方法会在下面说清楚
AtlasModel.hasMany(IconModel); // 图集 有很多 图标

module.exports = { AtlasModel };
```

建立多对多关系

假设：一个项目内包含多个图标，每个图标也可以被不同的项目引用。多对多关系需要用一个单独的描述。

（假如已经存在图标表IconModel、项目表ProjectModel）

多对多关系表

```
const { sequelize } = require("../../core/db");
const { Sequelize, Model } = require("sequelize");
const { IconModel } = require("./icon");
const { ProjectModel } = require("./project");

class IconProjectModel extends Model {}

IconProjectModel.init(
  {
    id: {
      type: Sequelize.INTEGER,
      primaryKey: true,
      autoIncrement: true,
    },
  },
  {
    sequelize,
    paranoid: false, // 设置删除为硬删除。关系表必须设置这个，否则修改关联关系是会出错。（因数据库的索引约束）
    tableName: "icon_project",
  }
);
// 定义图标和项目的关系---
// 图标 对应 多个项目
IconModel.belongsToMany(ProjectModel, {
  through: IconProjectModel,
  as: "projects",// 对于icon模型来说，对应的项目的名字叫projects
});
// 项目 对应 多个图标
ProjectModel.belongsToMany(IconModel, {
  through: IconProjectModel,
  as: "icons",// 对于项目模型来说，对应的项目的名字叫icons
});
```

```
module.exports = { IconProjectModel };
```

与关系表有关的操作

基本的操作比较简单，这里不做赘述。较为不好懂的是关于关系表存取数据的操作。这里举几个例子让大家快速理解如何使用。

相关文档：https://www.sequelize.com.cn/core-concepts/assocs#%E6%B7%BB%E5%8A%A0%E%88%B0%E5%AE%9E%E4%BE%8B%E7%9A%84%E7%89%B9%E6%AE%8A%E6%96%B9%E6%3%95

例一：通过项目id，找到项目所有的图标。

```
const project = await ProjectModel.findByPK(87);
const icons = await project.getIcons(); // 这样就取出来了所有的图标
const icons = await ProjectModel.getIcons(project); // 另一种写法

// 如果不想让返回的icons里面携带与icon相关的其他关系内容，可以输入参数
    const icons = await project.getIcons({
      joinTableAttributes: [], // 不携带关系表内容
      attributes: { exclude: ["deleted_time", "AtlasModelId","creationTime","updated_time"] }, //
不携带删除时间和图集外键字段
    });
```

例二： 更新项目中的图标

```
const newIcons = await IconModel.findAll({
 where:{
  name:{[Op.like]:'%icon1'}
 }
)// 找出所有名称中含有icon1的图标
const project = await ProjectModel.findByPK(87); // 需要修改的项目

await project.setIcons(newIcons) // 重置关联关系
await project.addIcons(newIcons) // 增加关联关系
await project.removeIcons(newIcons) // 删除关联关系
```