链滴

NestJs 搭建从零开始 (一)

作者：Gaoshengyue

原文链接：https://ld246.com/article/1625019657770

来源网站：链滴

前言

最近一年真是鸽了很久，比较忙。接触企业级框架SpringBoot的相关连续文章也没有再继续。趁着在还有部分时间，把之前写过demo示例的NestJs框架拿出来溜溜。

主旨在于如何从零搭建NestJs框架，在过程中可能会发现很多与SpringBoot类似的地方，其实可以说estJs就是仿照SpringBoot的相关架构来设计的，之前先开篇了SpringBoot的基本功能也是基于这个导的想法。

大的架构问题就不说太多了，直接用实际框架结构与代码来进行梳理。

首先先放整体的Demo地址: https://github.com/Gaoshengyue/NestJsLoanExample

官方文档地址:https://nestjs.bootcss.com/

正文

NestJs介绍

目前以个人见解，后端框架繁多的情况下，主流分类主要有以下几种:

1.适合用于大项目，约束性较强，市场应用较多，相当成熟的框架,比较有代表性的是:

Java:SpringBoot

Python:Django

2.适合互联网快餐，业务驱动型团队的首选，没有相对约束性的框架，可以自由构建体系，比较有代性的:

Python:FastApi,Flask

3.偏向于性能要求，技术驱动型团队的首选，大部分用于性能优化，比较有代表性的:

Go:Gin,Beego

还有其他的很多后台框架就不一一列举了。其实大体上框架理念在后端结构上都相差不大，差距在于自的语言特性差异，语法差异，和结构设计上。

NestJs的思想借鉴于SpringBoot，使用TypeScript来达到前后端开发语言统一的效果，同时又能支复杂的后端业务，又有自己的框架结构思想。介于1类与2类之间，在开放强约束限制的同时，又能达较为良好的可读性。

Demo结构，基本使用

```
├── app.controller.spec.ts
├── app.controller.ts
├── app.module.ts
├── app.service.ts
├── common
│   └── filters
│       └── exception.filters.ts
├── config
│   ├── env.ts
│   ├── env-utils.ts
│   ├── loanApplyNode.json
│   └── log4js.ts
├── customer
│   ├── customer.module.ts
│   ├── customer.service.ts
│   ├── dto
│   │   ├── customer.dto.ts
│   │   └── customerRegister.dto.ts
│   ├── manageCustomer.controller.ts
│   ├── userCustomer.controller.ts
│   └── vo
│       └── customer.vo.ts
├── customerAdmin
│   ├── customerDetail
│   │   ├── customerDetail.apiProperty.ts
│   │   ├── customerDetail.controller.ts
│   │   ├── customerDetailManager.controller.ts
│   │   ├── customerDetail.module.ts
│   │   └── customerDetail.service.ts
│   └── loanCustomer
│       ├── loanCustomer.apiProperty.ts
│       ├── loanCustomer.controller.ts
│       ├── loanCustomer.module.ts
│       └── loanCustomer.service.ts
├── entities
│   ├── AntiFraud.entity.ts
│   ├── BankCard.entity.ts
│   ├── Contract.entity.ts
│   ├── ContractOperation.entity.ts
```

```
│   ├── customer
│   │   ├── customerAudit.entity.ts
│   │   └── customer.entity.ts
│   ├── CustomerDetail.entity.ts
│   ├── EmergencyContact.entity.ts
│   ├── FAOperation.entity.ts
│   ├── FPOperation.entity.ts
│   ├── FunderAccountDetail.entity.ts
│   ├── FunderAccount.entity.ts
│   ├── Funder.entity.ts
│   ├── FunderProduct.entity.ts
│   ├── IDVerification.entity.ts
│   ├── JobInformation.entity.ts
│   ├── LAOperation.entity.ts
│   ├── LivenessDetection.entity.ts
│   ├── LoanApply.entity.ts
│   ├── LoanCustomer.entity.ts
│   ├── LoanRecordProcess.entity.ts
│   ├── RepaymentRecord.entity.ts
│   ├── RepaymentTask.entity.ts
│   ├── ReviewByFunder.entity.ts
│   ├── Risk.entity.ts
│   ├── RTOperation.entity.ts
│   ├── TelephoneInterview.entity.ts
│   └── Urge.entity.ts
├── funderAdmin
│   ├── funder
│   │   ├── funder.apiProperty.ts
│   │   ├── funder.controller.ts
│   │   ├── funderManager.controller.ts
│   │   ├── funder.module.ts
│   │   └── funder.service.ts
│   └── funderProduct
│       ├── funderProduct.apiProperty.ts
│       ├── funderProduct.controller.ts
│       ├── funderProductManager.controller.ts
│       ├── funderProduct.module.ts
│       └── funderProduct.service.ts
├── guard
│   └── customerAuth.guard.ts
├── loanAdmin
│   ├── loanApply
│   │   ├── loanApply.apiProperty.ts
│   │   ├── loanApply.controller.ts
│   │   ├── loanApplyManager.controller.ts
│   │   ├── loanApply.module.ts
│   │   └── loanApply.service.ts
│   └── loanRecordProcess
│       ├── loanRecordProcess.apiProperty.ts
│       ├── loanRecordProcess.controller.ts
│       ├── loanRecordProcessManager.controller.ts
│       ├── loanRecordProcess.module.ts
│       └── loanRecordProcess.service.ts
├── main.ts
```

原文链接：NestJs 搭建从零开始 (一)

```
├── middleware
│   ├── CheckToken.middleware.ts
│   ├── logger.middleware.spec.ts
│   └── logger.middleware.ts
├── minioService
│   └── minio.service.ts
├── public
│   └── test.jpg
├── redisDB
│   └── cache.service.ts
├── repaymentAdmin
│   ├── repaymentRecord
│   │   ├── repaymentRecord.apiProperty.ts
│   │   ├── repaymentRecord.controller.ts
│   │   ├── repaymentRecord.Managercontroller.ts
│   │   ├── repaymentRecord.module.ts
│   │   └── repaymentRecord.service.ts
│   └── repaymentTask
│       ├── repaymentTask.apiProperty.ts
│       ├── repaymentTask.Managercontroller.ts
│       ├── repaymentTask.module.ts
│       └── repaymentTask.service.ts
├── support
│   ├── apiResult.ts
│   ├── index.ts
│   ├── pageParams.dto.ts
│   ├── ResponseErrorEvent.ts
│   ├── responsePage.ts
│   └── support.module.ts
└── utils
    ├── log4js.ts
    └── numberUtil.ts
```

基本结构如上，main示例如下:

```
import {NestFactory} from '@nestjs/core';
import {AppModule} from './app.module';
import {DocumentBuilder, SwaggerModule} from "@nestjs/swagger";
import {logger} from "./middleware/logger.middleware";
import {CheckTokenMiddleware} from "./middleware/CheckToken.middleware";
import { HttpExceptionFilter } from './common/filters/exception.filters';
async function bootstrap() {
    const app = await NestFactory.create(AppModule);

    // 监听所有的请求路由，并打印日志
    app.use(logger);
    // 监听所有的请求路由，处理token
    // app.use(new CheckTokenMiddleware().use);
    // 错误过滤器
    app.useGlobalFilters(new HttpExceptionFilter())
    app.enableCors();
    const options = new DocumentBuilder()
        .setTitle("APItest")
```

```
        .setDescription("The test docs")
        .setVersion("1.0")
        .addTag("app")
        .addTag("admin")
        .build();
    const document=SwaggerModule.createDocument(app,options)
    SwaggerModule.setup("api",app,document)
    // await app.setGlobalPrefix("public");
    await app.listen(3001);
}

bootstrap();
```

其中增加了错误过滤器，Token验证中间件不过并没有挂载，开启了默认使用swagger，监听3001等。

在 const app = await NestFactory.create(AppModule); 创建时会调用app.mudule.ts读取配置。置文件如下:

```
import {MiddlewareConsumer, Module, NestModule, RequestMethod} from '@nestjs/commo
';
import {ConfigModule} from 'nestjs-config';
import * as path from 'path';
import {AppController} from './app.controller';
import {AppService} from './app.service';
import {TypeOrmModule} from '@nestjs/typeorm';
import {Connection} from 'typeorm';
import {LoanCustomerModule} from './customerAdmin/loanCustomer/loanCustomer.module';
import './config/env';
import {envBoolean, envNumber, envString} from './config/env-utils';
import {RedisModule} from "nestjs-redis";
import {FunderModule} from "./funderAdmin/funder/funder.module";
import {FunderProductModule} from "./funderAdmin/funderProduct/funderProduct.module";
import {LoanApplyModule} from "./loanAdmin/loanApply/loanApply.module";
import {CacheService} from "./redisDB/cache.service";
import {LoanRecordProcessModule} from "./loanAdmin/loanRecordProcess/loanRecordProce
s.module";
import {ServeStaticModule} from "@nestjs/serve-static";

import {CheckTokenMiddleware} from "./middleware/CheckToken.middleware";
import {CustomerDetailModule} from "./customerAdmin/customerDetail/customerDetail.mo
ule";
import {RepaymentRecordModule} from "./repaymentAdmin/repaymentRecord/repaymentRe
ord.module";
import {RepaymentTaskModule} from "./repaymentAdmin/repaymentTask/repaymentTask.m
dule";
import {SupportModule} from "./support/support.module";

import {CustomerModule} from "./customer/customer.module";

@Module({
  imports: [
    ConfigModule.load(path.resolve(__dirname, 'config', '**/!(*.d).{ts,js}')),
    TypeOrmModule.forRoot({       //数据库连接配置
```

```
      type: 'mysql',
      host: envString('DB_HOST', 'localhost'),
      port: envNumber('DB_PORT', 3306),
      username: envString('DB_USERNAME', 'root'),
      password: envString('DB_PASSWORD', 'root'),
      database: envString('DB_DATABASE', 'ts_test'),
      entities: [__dirname + '/**/*.entity{.ts,.js}'],    //装载全部实体
      synchronize: envBoolean('DB_SYNCHRONIZE', true),              //自动迁移同步数据库表结构

    }),

    RedisModule.register({
        port: envNumber("REDIS_PORT",6379),
        host: envString("REDIS_HOST","127.0.0.1"),
        password: envString("REDIS_PASSWORD",""),
        db: envNumber("REDIS_DB",0)
    }),

    // ServeStaticModule.forRoot({
    //   rootPath:path.join(__dirname,"..","public"),
    //   exclude:["/public*"],
    // }),
    CustomerModule,

    SupportModule,
    LoanCustomerModule,
    FunderModule,
    FunderProductModule,
    LoanApplyModule,
    LoanRecordProcessModule,
    CustomerDetailModule,
    RepaymentRecordModule,
    RepaymentTaskModule,
  ],
  controllers: [AppController],
  providers: [AppService,CacheService],
})


export class AppModule implements NestModule{
  constructor(private readonly connection: Connection) {
  }

  configure(consumer: MiddlewareConsumer) {

    // token 中间件拦截
    // consumer.apply(CheckTokenMiddleware).forRoutes({path:"*",method:RequestMethod.AL
})

  }
}
```

这里要注意的是。我们所有的基层组件Module，都要在这里引入，并加入到框架启动的扫描。Dem

里配置了Mysql，Redis，还有业务相关的CustomerModule等等。因为各个业务的Controller为了便示例，都做在各业务单独的Module里面，所以在Controllers里面只有AppController用于测试。CcheService等基层服务也要一同引入。如果是在业务的Module里面进行引入，这里可以忽略。

接下来其实已经走过了router的过程。app.Controller已经可以直接引入路由导航了。

```
import {Body, Controller, Get, Post, Req} from '@nestjs/common';
import {json, Request} from 'express';
import {AppService} from './app.service';
import {Redirect} from "@nestjs/common";
import {createConnection} from "typeorm";

@Controller("/")
export class AppController {
    constructor(private readonly appService: AppService) {
    }

    @Post("uploadLoanProcessJson")
    async uploadJson(@Body() jsonData): Promise<any> {
        this.appService.upload(jsonData)
        return jsonData

    }

    @Get("")
    async Index(): Promise<string> {

        return "<a href=\"weixin://dl/business/?ticket=te1e2ba9afbaa76e79a6185e74ef8b1ab\"
手机浏览器打开测试<a/>"
    }


}
```

在这里做了简单的示例。与SpringBoot的结构类似。NestJs也采用了注解的形式，习惯使用Python同学们可以理解为装饰器。

在 constructor(private readonly appService: AppService) { } 中引用了所需服务。依赖注入。调方式直接使用this.service。这里的的service执行业务逻辑。直接到了app.service

```
import { Injectable } from '@nestjs/common';
import {CacheService} from "./redisDB/cache.service";
import {Repository} from "typeorm";
import {LoanRecordProcess} from "./entities/LoanRecordProcess.entity";

@Injectable()
export class AppService {
  constructor(private readonly cache: CacheService,
){}

  async upload(jsonData):Promise<number>{

    this.cache.set("loanApllyProcess:config",JSON.stringify(jsonData));
```

```
    return 1
  }

  getHello(): string {
    return 'Hello World!';
  }

  getTwo():string{
    return "here is two";
  }
}
```

@Injectable()的作用为可以注入实例化，在这里也通过依赖注入了CacheService，可以调用其他servie封装的函数。

以上是NestJs的基本使用。