链滴

我还不知道这个叫什么

作者：devcui
原文链接：https://ld246.com/article/1629025993723
来源网站：链滴

ng-alain

转载请注明原出处，勘误请发送电子邮件

1.什么是ng-alain?

ng-alain是来自中国作者 卡色 开源出的一个企业级中后台前端/设计解决方案脚手架，秉承 Ant Design 的设计价值观，让 Angular 快速落地于企业生产实践中的一个高集成性框架。

其中提供了非常丰富的功能诸如

- Dynamic Form
- Antv/Echarts
- Acl
- Auth
- Theme
- Cache
- Mock
- Util (DeepCopy/TreeToArray等等)

假如你对Angular有一些使用基础，那么使用起来会相当的得心应手。

假如你对Angular只有一些略微了解，那么也可以通过ng-alain提供的CLI快速的建立起一个可运行Project用来快速上手。

2.ng-alain解析

- 生成一个ng-alain项目首先需要安装 AngularCli
- npm install @angular/cli@12.2.0
- 其次使用 ng new my-project --style less --routing创建名为my-project的纯Angular项目
- 然后使用 ng-alain的CLI生成项目
- ng add ng-alain
- 最后使用 npm run start 启动项目
- 或者直接前往 预览地址

3.ng-alain 项目

“没有比浏览 angular.json 配置文件更能快速了解一个angular项目的方法了” -詹姆斯高斯林

```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "ng-alain": {
```

```
"projectType": "application",
"root": "",
"sourceRoot": "src",
"prefix": "app",
"schematics": {
  "@schematics/angular:component": {
    "style": "less"
  },
  "@schematics/angular:application": {
    "strict": true
  }
},
"architect": {
  "build": {
    "builder": "@angular-devkit/build-angular:browser",
    "options": {
      "outputPath": "dist",
      "index": "src/index.html",
      "main": "src/main.ts",
      "tsConfig": "tsconfig.app.json",
      "polyfills": "src/polyfills.ts",
      "assets": ["src/assets", "src/favicon.ico"],
      "styles": ["src/styles.less"],
      "scripts": [],
      "allowedCommonJsDependencies": ["ajv", "ajv-formats"]
    },
    "configurations": {
      "production": {
        "fileReplacements": [
          {
            "replace": "src/environments/environment.ts",
            "with": "src/environments/environment.prod.ts"
          }
        ],
        "outputHashing": "all",
        "budgets": [
          {
            "type": "initial",
            "maximumWarning": "2mb",
            "maximumError": "6mb"
          },
          {
            "type": "anyComponentStyle",
            "maximumWarning": "6kb",
            "maximumError": "10kb"
          }
        ]
      },
      "development": {
        "buildOptimizer": false,
        "optimization": false,
        "vendorChunk": true,
        "extractLicenses": false,
        "sourceMap": true,
```

```
        "namedChunks": true
      }
    },
    "defaultConfiguration": "production"
  },
  "serve": {
    "builder": "@angular-devkit/build-angular:dev-server",
    "options": {
      "browserTarget": "ng-alain:build",
      "proxyConfig": "proxy.conf.json"
    },
    "configurations": {
      "production": {
        "browserTarget": "ng-alain:build:production"
      },
      "development": {
        "browserTarget": "ng-alain:build:development"
      }
    },
    "defaultConfiguration": "development"
  },
  "extract-i18n": {
    "builder": "@angular-devkit/build-angular:extract-i18n",
    "options": {
      "browserTarget": "ng-alain:build"
    }
  },
....
```

我们可以清晰的从angular.json中看出

- build部分
- projects 中仅仅含有一个类型为application的项目ng-alain
- builder编译器使用了默认的@angular-devkit/build-angular:browser，当然还有许多其他的编译，可以自行搜索使用。
- builder中的options指定了一些入口文件
- 引入了 CommanJS: ajv,ajv-formats，ajv是JSON Schema的一种验证器
- build中的configuration会使用env.prod.ts替换env.ts
- outputHashing打包文件带有哈希值
- serve部分
- serve 对应的命令为 ng serve，使用的builder为@angular-devkit/build-angular:dev-server，们可以在 @angular-devikt找到这个包，有一些配置参数和说明，刚刚build时使用的browser，还有pp-shell呦

找到了入口文件index.html

```
<app-root></app-root>
  <div class="preloader">
```

```
    <div class="cs-loader">
      <div class="cs-loader-inner">
        <label> ●</label>
        <label> ●</label>
        <label> ●</label>
        <label> ●</label>
        <label> ●</label>
        <label> ●</label>
      </div>
    </div>
  </div>
```

比较重要的一部分

- <app-root> ，这里是对app.component.ts的直接引用
- 下面的 div中使用class和css做了一个加载动画，可以理解成遮罩层，利用最先渲染index.html，用css将整个webapp遮盖住
- index.html结束

找到了入口文件main.ts

```
preloaderFinished();
```

第九行调用了preloaderFinished，这个函数是在alain的另一个基础项目delon中以基础包的形式出的

```
export function preloaderFinished(): void {
  const body = document.querySelector('body')!;
  const preloader = document.querySelector('.preloader')!;

  body.style.overflow = 'hidden';

  function remove(): void {
    // preloader value null when running --hmr
    if (!preloader) return;
    preloader.addEventListener('transitionend', () => {
      preloader.className = 'preloader-hidden';
    });

    preloader.className += ' preloader-hidden-add preloader-hidden-add-active';
  }

  (window as NzSafeAny).appBootstrap = () => {
    setTimeout(() => {
      remove();
      body.style.overflow = '';
    }, 100);
  };
}
```

- 写了之前说的遮罩层如何删除

- 赋予window对象一个 appBootstrap函数，用来进行调用remove删除掉index.html中的遮罩层

```
if (environment.production) {
enableProdMode();
}
```

如果环境为prod模式那么开启Prod模式

```
platformBrowserDynamic()
  .bootstrapModule(AppModule, {
    defaultEncapsulation: ViewEncapsulation.Emulated,
    preserveWhitespaces: false
  })
  .then(res => {
    const win = window as NzSafeAny;
    if (win && win.appBootstrap) {
      win.appBootstrap();
    }
    return res;
  })
  .catch(err => console.error(err));
```

上面为main.ts中最后的内容

- 引导 AppModule启动
- 启动完毕后调用 win中存下的remove函数，删除遮罩层

```
@import '~@delon/theme/system/index';
@import '~@delon/abc/index';
@import '~@delon/chart/index';
@import '~@delon/theme/layout-default/style/index';
@import '~@delon/theme/layout-blank/style/index';

@import './styles/index';
@import './styles/theme';
```

在style.less中引入了一些less变量，以供全局css使用，这些less变量也是@delon/theme包中出现，后续可能会看到

至此，除了assets静态资源目录，其余从angular.json找到的入口的启动过程梳理完毕，分别

- 提供了模版
- 提供了静态资源
- 提供了JS启动模块
- 提供了全局CSS

3.1 项目配置

接下来看被bootstrap的AppModule

```
const LANG = {
  abbr: 'zh',
  ng: ngLang,
  zorro: zorroLang,
  date: dateLang,
  delon: delonLang
};
registerLocaleData(LANG.ng, LANG.abbr);
const LANG_PROVIDES = [
  { provide: LOCALE_ID, useValue: LANG.abbr },
  { provide: NZ_I18N, useValue: LANG.zorro },
  { provide: NZ_DATE_LOCALE, useValue: LANG.date },
  { provide: DELON_LOCALE, useValue: LANG.delon }
];
```

前面主要是本地化/i18n的一些配置，通过provide键值对的方式注入到包里，使得第三方库使用useValue提供的值

```
const I18NSERVICE_PROVIDES = [{ provide: ALAIN_I18N_TOKEN, useClass: I18NService, multi: false }];
```

当然除了使用useValue的方式，还可以使用useClass

```
export const ALAIN_I18N_TOKEN = new InjectionToken<AlainI18NService>('alainI18nToken',

  providedIn: 'root',
  factory: () => new AlainI18NServiceFake()
});
```

上文的ALAIN_I18N_TOKEN在@delon/theme中，接收一个AlainI18nService类型的class

I18nService继承自AlainI18nBaseService，而AlainI18nBaseService又实现了AlainI18NService接，所以类型匹配，这样我们就可以把自己的class传入第三方包中，第三方包注入class时调用的function就可以来自外部实际项目了

```
const GLOBAL_THIRD_MODULES: Array<Type<any>> = [BidiModule]
```

引入了一个左右布局切换的Module，在cdk里，cdk是@angular提供的一个工具包

```
const INTERCEPTOR_PROVIDES = [
  { provide: HTTP_INTERCEPTORS, useClass: SimpleInterceptor, multi: true },
  { provide: HTTP_INTERCEPTORS, useClass: DefaultInterceptor, multi: true }
];
```

再往下就是这两个拦截器

- Simple: 验证token
- Default: 处理错误信息

```
export function StartupServiceFactory(startupService: StartupService): () => Observable<void
 {
  return () => startupService.load();
}
const APPINIT_PROVIDES = [
  StartupService,
  {
    provide: APP_INITIALIZER,
    useFactory: StartupServiceFactory,
    deps: [StartupService],
    multi: true
  }
];
```

重点，这里我称为实际引导项目启动的一个函数，这里作为整个项目声明周期的初始化周期，使用了Angular内置的provide: APP_INITIALIZER 并返回了一个Promise函数load

```
@Injectable()
export class StartupService {
  constructor(
    iconSrv: NzIconService,
    private menuService: MenuService,
    @Inject(ALAIN_I18N_TOKEN) private i18n: I18NService,
    private settingService: SettingsService,
    private aclService: ACLService,
    private titleService: TitleService,
    private httpClient: HttpClient
  ) {
    iconSrv.addIcon(...ICONS_AUTO, ...ICONS);
  }

  load(): Observable<void> {
    const defaultLang = this.i18n.defaultLang;
    return zip(this.i18n.loadLangData(defaultLang), this.httpClient.get('assets/tmp/app-data.jso
')).pipe(
      // 接收其他拦截器后产生的异常消息
      catchError(res => {
        console.warn(`StartupService.load: Network request failed`, res);
        return [];
      }),
      map(([langData, appData]: [Record<string, string>, NzSafeAny]) => {
        // setting language data
        this.i18n.use(defaultLang, langData);
        // 应用信息：包括站点名、描述、年份
        this.settingService.setApp(appData.app);
        // 用户信息：包括姓名、头像、邮箱地址
        this.settingService.setUser(appData.user);
        // ACL：设置权限为全量
        this.aclService.setFull(true);
        // 初始化菜单
        this.menuService.add(appData.menu);
        // 设置页面标题的后缀
        this.titleService.default = '';
        this.titleService.suffix = appData.app.name;
```

```
      })
    );
  }
}
```

以上代码初始化了整体项目必备的数据

```
@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    HttpClientModule,
    GlobalConfigModule.forRoot(),
    CoreModule,
    SharedModule,
    LayoutModule,
    RoutesModule,
    STWidgetModule,
    NzNotificationModule,
    ...GLOBAL_THIRD_MODULES,
    ...FORM_MODULES
  ],
  providers: [...LANG_PROVIDES, ...INTERCEPTOR_PROVIDES, ...I18NSERVICE_PROVIDES, ...APP
NIT_PROVIDES],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

回到app.module中，imports一些模块，providers一些服务，

其中对于整体项目比较重要的模块有

- GlobalConfigModule.forRoot() 全局配置模块，主要配置alain中的一些组件功能/认证/等各方面内容
- LayoutModule 在用户端的一个布局模块
- SharedModule 共享模块
- RoutesModule 总体路由模块，比如/app/v1路径，要渲染哪个模块的那个组件，就是这个模块决的，在这个项目里，也称为业务模块
- CoreModule core模块，刚才提到的拦截器和i18n和startup就在这个模块中

```
const alainConfig: AlainConfig = {
  st: { modal: { size: 'lg' } },
  pageHeader: { homeI18n: 'home' },
  lodop: {
    license: `A59B099A586B3851E0F0D7FDBF37B603`,
    licenseA: `C94CEE276DB2187AE6B65D56B3FC2848`
  },
  auth: { login_url: '/passport/login' }
```

```
};

const alainModules: any[] = [AlainThemeModule.forRoot(), DelonACLModule.forRoot()];
const alainProvides = [{ provide: ALAIN_CONFIG, useValue: alainConfig }];

const ngZorroConfig: NzConfig = {};

const zorroProvides = [{ provide: NZ_CONFIG, useValue: ngZorroConfig }];

// #endregion

@NgModule({
  imports: [...alainModules, ...(environment.modules || [])]
})
export class GlobalConfigModule {
  constructor(@Optional() @SkipSelf() parentModule: GlobalConfigModule) {
    throwIfAlreadyLoaded(parentModule, 'GlobalConfigModule');
  }

  static forRoot(): ModuleWithProviders<GlobalConfigModule> {
    return {
      ngModule: GlobalConfigModule,
      providers: [...alainProvides, ...zorroProvides]
    };
  }
}
```

配置了一些基于AlainConfig接口的配置，配置了一些zorro的配置，然后再次通过providers传入到@elon包中

```
@NgModule({
  imports: [
    CommonModule,
    FormsModule,
    RouterModule,
    ReactiveFormsModule,
    AlainThemeModule.forChild(),
    DelonACLModule,
    DelonFormModule,
    ...SHARED_DELON_MODULES,
    ...SHARED_ZORRO_MODULES,
    // third libs
    ...THIRDMODULES
  ],
  declarations: [
    // your components
    ...COMPONENTS,
    ...DIRECTIVES
  ],
  exports: [
    CommonModule,
    FormsModule,
```

```
    ReactiveFormsModule,
    RouterModule,
    AlainThemeModule,
    DelonACLModule,
    DelonFormModule,
    ...SHARED_DELON_MODULES,
    ...SHARED_ZORRO_MODULES,
    // third libs
    ...THIRDMODULES,
    // your components
    ...COMPONENTS,
    ...DIRECTIVES
  ]
})
```

shared 模块存在的意义就是，统一管理每个routes模块所依赖的其他模块，新增业务模块时，只需引入SharedModule就可以将其中的一大坨模块直接引入到你建立的模块中，至于文件体积和过多引不用担心，打包时编译器会做优化的

```
export const SHARED_ZORRO_MODULES = [
  NzButtonModule,
  NzMessageModule,
  NzDropDownModule,
  NzGridModule,
  NzCheckboxModule,
  NzToolTipModule,
  NzPopoverModule,
  NzSelectModule,
  NzIconModule,
  NzBadgeModule,
  NzAlertModule,
  NzModalModule,
  NzTableModule,
  NzDrawerModule,
  NzTabsModule,
  NzInputModule,
  NzDatePickerModule,
  NzTimePickerModule,
  NzTagModule,
  NzInputNumberModule,
  NzBreadCrumbModule,
  NzListModule,
  NzSwitchModule,
  NzRadioModule,
  NzFormModule,
  NzAvatarModule,
  NzSpinModule,
  NzCardModule,
  NzDividerModule,
  NzProgressModule,
  NzPopconfirmModule,
  NzUploadModule
];
```

zorro模块，将zorro中所有的模块封装起来，为了之后全量引入，不再做按需引入

接下来是布局，在app.component中的模版只有一个router-outlet，这里直接为后续所有组件渲染出了位置接口，所有其他的组件会渲染到HTML中的这个位置

```
const routes: Routes = [
  {
    path: '',
    component: LayoutBasicComponent,
    canActivate: [SimpleGuard],
    canActivateChild: [SimpleGuard],
    data: {},
    children: [
      { path: '', redirectTo: 'dashboard', pathMatch: 'full' },
      { path: 'dashboard', loadChildren: () => import('./dashboard/dashboard.module').then(m
> m.DashboardModule) },
      {
        path: 'widgets',
        loadChildren: () => import('./widgets/widgets.module').then(m => m.WidgetsModule)
      },
      { path: 'style', loadChildren: () => import('./style/style.module').then(m => m.StyleModule)
},
      { path: 'delon', loadChildren: () => import('./delon/delon.module').then(m => m.DelonMo
ule) },
      { path: 'extras', loadChildren: () => import('./extras/extras.module').then(m => m.ExtrasM
dule) },
      { path: 'pro', loadChildren: () => import('./pro/pro.module').then(m => m.ProModule) }
    ]
  },
  // Blak Layout 空白布局
  {
    path: 'data-v',
    component: LayoutBlankComponent,
    children: [{ path: '', loadChildren: () => import('./data-v/data-v.module').then(m => m.Data
Module) }]
  },
  // passport
  { path: '', loadChildren: () => import('./passport/passport.module').then(m => m.PassportMo
ule) },
  { path: 'exception', loadChildren: () => import('./exception/exception.module').then(m => m.
xceptionModule) },
  { path: '**', redirectTo: 'exception/404' }
];
```

然后看路由表，最外层的父路由中，指定了渲染的component: LayoutBasicComponent，这个组件是定义在LayoutModule中的

```
....
<ng-template #asideUserTpl>
      <div nz-dropdown nzTrigger="click" [nzDropdownMenu]="userMenu" class="alain-defa
lt__aside-user">
        <nz-avatar class="alain-default__aside-user-avatar" [nzSrc]="user.avatar"></nz-avatar
```

```
        <div class="alain-default__aside-user-info">
          <strong>{{ user.name }}</strong>
          <p class="mb0">{{ user.email }}</p>
        </div>
      </div>
      <nz-dropdown-menu #userMenu="nzDropdownMenu">
        <ul nz-menu>
          <li nz-menu-item routerLink="/pro/account/center">{{ 'menu.account.center' | i18n }}
/li>
          <li nz-menu-item routerLink="/pro/account/settings">{{ 'menu.account.settings' | i18
 }}</li>
        </ul>
      </nz-dropdown-menu>
    </ng-template>


    <ng-template #contentTpl>
      <router-outlet></router-outlet>
    </ng-template>
```

.....

拿出一段LayoutBasicComponent的HTML，其中又有一个<router-outlet>

这里是为子路由对应的组件放出的渲染位置

假设我的访问路径为:/dashboard/

- 首先会匹配到路由表中: ''空路由，将LayoutBasicComponent 渲染在app.component中的route-outlet上，又由于，index.html在删除preloader-div后只剩下app-root，所以LayoutBasicCompoent占了整张页面
- 其次匹配 dashboard路由，这里dashboard指向的组件为dashboard/dashboard.module模块的DashboardV1Component，所以DashboardV1Component就会渲染在 已经渲染完LayoutBasicomponent组件中的router-outlet位置
- 这样通过路由和子路由加上 router-outlet完成了整体布局
- PS: loadChildren: () => import('./dashboard/dashboard.module').then(m => m.DashboarModule)这种加载方法被称为懒加载，在DashboardModule中还有一个子路由表我这里没贴出来，兴趣可以自己去看一下

至此，大体上过了个差不多，你可以看到虽然alain是一个项目，但它又不仅仅是一个项目，下面我们析一下alain项目的组成

4.alain 项目组成

直接看package.json吧，就不对照模块了，刚才涉及到的代码有很多都来自以下这些npm包

"ng-zorro-antd": "^12.0.1", 阿里ant组件库

"@delon/abc": "^12.1.0"，delon包，也是最核心的包

```
"@delon/acl": "^12.1.0",
"@delon/auth": "^12.1.0",
"@delon/cache": "^12.1.0",
"@delon/chart": "^12.1.0",
"@delon/form": "^12.1.0",
"@delon/mock": "^12.1.0",
"@delon/theme": "^12.1.0",
"@delon/util": "^12.1.0",

"ngx-tinymce": "^12.0.0", 两个编辑器包
 "ngx-ueditor": "^12.0.0",
 "screenfull": "^5.1.0",

 "ajv": "^8.6.2" jsonSchematicValidate包

"ng-alain": "^12.1.0"，脚手架包

  "ng-alain-plugin-theme": "^12.0.0"，主题生成插件
  "ng-alain-sts": "^0.0.1"，构建 Swagger API 转换为列表、编辑页的命令行工具
```

5.delon

delon的项目地址位于: https://github.com/ng-alain/delon

按照惯例先看angular.json

```
"delon": {
    "root": "packages",
    "projectType": "library",
    "prefix": "",
    "architect": {
      "lint": {
        "builder": "@angular-eslint/builder:lint",
        "options": {
          "fix": true,
          "lintFilePatterns": [
            "packages/**/*.ts",
            "packages/**/*.html"
          ]
        }
      },
      "test": {
        "builder": "@angular-devkit/build-angular:karma",
        "options": {
          "main": "packages/test.ts",
          "karmaConfig": "packages/karma.conf.js",
          "polyfills": "packages/polyfills.ts",
          "tsConfig": "packages/tsconfig.spec.json",
          "scripts": ["node_modules/@antv/g2/dist/g2.min.js", "node_modules/@antv/data-set/
ist/data-set.js"],
          "codeCoverageExclude": ["schematics/**", "packages/testing/**"]
        }
      }
    }
```

```
  }
```

可以看到这里delon的跟路径为packages，应用类行为package，并不是刚才的application，说明并不是一个应用，而是一个最终为npm 包形式的项目，所以自然没有serve启动项的配置

但是我们找到了入口root配置项配置的是packages

5.1 delon/packages

5.2 delon build

5.3 delon schematics

5.4 sts

5.5 plugin-theme

plugin-theme主要是用来生成主题配置的一个插件

plugin-theme 的项目在 https://github.com/ng-alain/plugin-theme.git

```
const cli = meow({
  help: `
  Usage
    ng-alain-plugin-theme
  Example
    ng-alain-plugin-theme -t=themeCss -c=ng-alain.json
  Options
    -t, --type    Can be set 'themeCss', 'colorLess'
    -c, --config  A filepath of NG-ALAIN config script
    -d, --debug   Debug mode
  `,
  flags: {
    type: {
      type: 'string',
      default: 'themeCss',
      alias: 't',
    },
    config: {
      type: 'string',
      default: 'ng-alain.json',
      alias: 'c',
    },
    debug: {
      type: 'boolean',
      default: false,
      alias: 'd',
    },
  },
});
```

这里使用了基于nodejs的meow包，构建了一个CLI工具，所谓CLI即为Command Line Interface，实就是命令行，这里给出了三个配置项

- -t 文件类型,less/css
- -c 自定义主题config，详情见ng-alalin项目内的ng-alain.json，也可以在 主题切换中找到使用式
- -d 是否开启debug模式

```
let config: { theme: Config; colorLess: Config; [key: string]: Config };
```

传入JSON的格式

```
[key:string]:Config
```

意思是接口的任何属性，都要是Config格式

```
try {
  const configFile = resolve(process.cwd(), cli.flags.config);
  if (existsSync(configFile)) {
    config = getJSON(configFile);
  } else {
    console.error(`The config file '${cli.flags.config}' will not found`);
    process.exit(1);
  }
} catch (err) {
  console.error('Invalid config file', err);
  process.exit(1);
}
```

读取cli传入的文件路径 然后转换成JSON，赋值给config

```
if (cli.flags.type === 'themeCss') {
  buildThemeCSS(config.theme);
} else if (cli.flags.type === 'colorLess') {
  genColorLess(config.colorLess);
} else {
  throw new Error(`Invalid type, can be set themeCss or colorLess value`);
}
```

假如type是css那么走buildThemeCSS函数

假如type是less那么走genColorLess函数

```
export async function genColorLess(config: ColorLessConfig): Promise<void> {
  config = fixConfig(config);

  if (existsSync(config.outputFilePath!)) {
    unlinkSync(config.outputFilePath!);
  }
```

```
  await generateTheme(config);
}
```

genColorLess简单的处理config传入的ng-alalin.json转换成的对象，然后调用了generateTheme来成主题，最后输出到项目的目录中，具体通过json来生成theme的函数规则较为复杂，概括一下就是拿son传入的配置，对文件内容自动进行@import，替换等操作，最后生成目标文件less，写入原项目中

5.6 scripts

5.7 site

5.8 changelog

5.9 publish

5.10 release