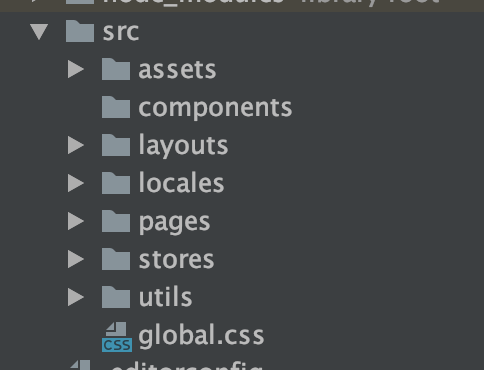
开发帮助

1. 目录约定



components: 通用组件

pages: 页面入口文件

stores: 业务数据文件

utils: 工具函数

global.less: 全局css文件，全局通用的css写在这里

1. 页面路由约定
2. 在pages下面创建文件夹(如Analysis)，里面增加一个文件\_layout.js， 这个是一级菜单。
3. 在Analysis目录下面创建页面文件，这个是二级菜单和页面内容。
4. \_layout.js固定写法:

import React from "react";

function Layout(props) {

return (

<>

{props.children}

</>

)

}

Layout.label = '分析台' //菜单文本

Layout.icon = 'AreaChartOutlined' //图标

Layout.order = 2 //排序

export default Layout;

1. 页面内容固定写法:

const Page = (props) => {

return (

<div></div>

)

}

Page.label = '分析台' //菜单文本

Page.module = 'dmcAnalysis' //页面权限模块定义

Page.order = 1 //菜单排序

export default Page;

1. 页面和业务逻辑
2. 业务逻辑,在stores目录下添加文件:

**import** {observable,action} **from** 'mobx';  
**import** \_ **from** 'underscore';  
**import** {loading} **from** 'mobx-loading';  
  
**class** Dishes {  
 @observable items = []; //数据  
 @observable items2 = [];  
  
 // 函数  
 @loading("dishes/getItems")  
 @action  
 getItems = **async** ()=>{  
 **this**.items = \_.shuffle(\_.range(10));  
 **return this**.items;  
 }  
  
 @loading("dishes/getItems2")  
 @action  
 getItems2 = **async** ()=>{  
 console.log('>>> run getItem2')  
 **this**.items2 = \_.shuffle(\_.range(10));  
 **return this**.items2;  
 }  
}  
  
**export default new** Dishes();

1. loading处理
2. 首先在业务逻辑函数前面加上@loading修饰符

**import** {loading} **from** 'mobx-loading';

@loading("dishes/getItems")  
@action  
getItems = **async** ()=>{  
 **this**.items = \_.shuffle(\_.range(10));  
 **return this**.items;  
}

注意： @loading注解一定要按"dishes/getItems"这种格式传值，否则生产环境会出问题。

1. 页面渲染函数里面调用useLoading获取对应action的loading状态

**import** {useLoading} **from** "@/utils";

**const** isLoading = useLoading('dishes/getItems')

1. 界面文本国际化写法

**import** { useI18n } **from** '@/utils';

const \_t = useI18n()

<Button>{\_t('测试')}</Button>

1. 组件写法规范
2. 顶级组件如果需要侦听mobx数据更新用observer包起来

**import** {observer} **from** 'mobx-react; //网页版

**import** {observer} **from** 'mobx-react-lite'; //手机版

**const** HomePage = observer(() =>{  
 return <View/>;  
});  
  
**export default** HomePage;

1. 子组件如果需要侦听mobx数据更新用observer包起来，或者用React.memo包起来(推荐这种方式)。

**const** SubView = React.memo(({id})=> {  
 **return** <Text>Hello world - {id}</Text>  
});

**export default** SubView;

1. 性能优化
2. mobx的类中的action函数，每给一个@observable的变量赋值一次，就会触发一次关联页面刷新，所以一个方案是不同的@observable的变量关联的页面部分划分小模块，或者将多个@observable变量合并成一个。
3. 尽量将页面划分成多个小模块，除非该页面只渲染一次，不会有任何操作刷新。基本原则就是把数据分组，每组数据对应的页面部分分出一个小组件。
4. 常用第三方库
5. React参考: <https://zh-hans.reactjs.org/docs/getting-started.html>
6. UI库： <https://ant-design.gitee.io/docs/react/introduce-cn>
7. 集合处理库: <http://underscorejs.org/>
8. 日期时间处理库: <http://momentjs.cn/docs/>
9. 状态管理: <https://cn.mobx.js.org/>
10. Umi: <https://umijs.org/zh-CN>
11. Ahooks: <https://ahooks.gitee.io/zh-CN>