## 1.模块化功能实现与编程规范

### 1.1 功能目标

1. 模块之间没有耦合；

   因此模块与模块之间不得直接引用。如：模块A中的JS文件不得直接`require`或`import`模块B的JS文件

2. 删除某个模块，其他模块也能正常运作，不会导致崩溃。

### 1.2 功能实现

为实现上述功能目标，将采取以下方法实现：

1. 单个模块声明所有可跳转页面，和声明所有模块接口。如下

   `ModuleA`包含页面：`ScreenA1`和`ScreenA2`，提供的接口：`getDataFromModuleA(key)`，我们可在`ModuleA`文件夹下新建`index.js`来进行声明，文件结构如下：

   ```
   └── ModuleA
       ├── ScreenA1.js
       ├── ScreenA2.js
       └── index.js
   ```

   `index.js`代码如下：

   ```
   /*
    * @providesModule ModuleA
    */

   function getDataFromModuleA(key){
       return ...;
   }

   module.exports = {
       ScreenA1: require('./ScreenA1');
       ScreenA2: require('./ScreenA2');
       getDataFromModuleA,
   };
   ```

#### # 1.2.1 模块间页面跳转

在项目中，每个页面都被赋予了`navigator`属性，我们利用它来做页面跳转，其中`screen`属性指定某个页面，`props`代表传递的参数。

- 模块内跳转：

  例：`ModuleA`的`ScreenA1`跳转到`ScreenA2`

  同一个模块内，页面可直接引用；`screen`一定是`require`之后的对象，不可以直接是`ScreenA2`字符串，新规定。

```
this.props.navigator.push({
    screen:require('./ScreenA2'),
    props:{
        //需要传递的属性
    }
});
```

- 模块间跳转：

  例：`ModuleA`的`ScreenA1`跳转到`ModuleB`的`ScreenB1`

  模块间不可以直接引用；`screen`一定是`模块名 + . + 页面名`字符串，因此这里`screen`为`ModuleB.ScreenB1`。

```
this.props.navigator.push({
    screen:'ModuleB.ScreenB1',
    props:{
        //需要传递的属性
    }
});
```

### 2. 模块目录规范

遵循以下目录结构

- redux模块目录结构

```
moduleName/                    //模块目录
    ├── actions/               //存放action源码
    ├── constants/             //存放常量源码
    ├── containers/            //存放UI源码
    |   ├── component/         //存放UI相关的自定义组件源码
    |   └── img/               //存放UI相关的图片资源
    ├── reducers/              //存放reducer源码
    |   └── index.js           //用于声明整个目录下的reducer
    └──  index.js               //用于声明模块页面与接口，以及reducer
```

- 非redux模块目录结构

```
moduleName/                    //模块目录
    ├── constants/             //存放常量源码
    ├── containers/            //存放UI源码
    |   ├── component/         //存放UI相关的自定义组件源码
    |   └── img/               //存放UI相关的图片资源
    └──  index.js               //用于声明模块页面与接口，方便注册
```

二者区别：`redux`含有`actions`和`reducers`文件夹。

### 3. 模块声明规范

### Q&A

---

Q：为什么要在模块下面建立`index.js`？

A：因为模块化之后，模块对外声明`页面`与`接口`，以及`redux`在`createStore`时所需要的`reducer`，这些声明需要写在这个文件。

---

Q：为什么一定要叫`index.js`？

A：事实上，可以起其它名字，例如，你可以起名为`${moduleName}.js`，最主要的是js开头的注释内容需要包含`providesModule ${moduleName}`。建议你使用`index.js`名字，因为有这样的共识，不需要记忆模块名称，打开模块目录就能快速找到模块声明。另外js有个特点，通过`require`模块目录路径，实际上就是`require`了这个目录下的`index.js`文件。

---

Q：到底需要声明些什么东西?

A：1，模块名称；2.模块主页与模块对外页面；3.模块对外接口；4.redux所需的reducer。

---