

## ILS V1.0前端框架设计

李淳

2020年5月14日

## 文档背景

为了提升协作质量，降低后续的维护成本，针对框架的一些设计进行  
本文档未记录一些常规的约定信息。

## 基本架设原则及优势

1. 快速开发能力：对于重复度较高的页面，能够快速产出；
2. 提高下限：对于零基础的同学也足够友好，不因对React熟悉程度不同而写出水平过于参差的代码；
3. 延伸上限：面向未来，能够很好的去扩展、延伸，对于超复杂代码、逻辑能够有独立解耦的吸纳方案，对于超大规模的未来有很好的扩展方案，不因规模变大而尾大不掉；
4. 可维护性强，高效协作：有完备的开发规范，有限制的自由；

## 核心选型

ITEM	NAME	VERSION	DESCRIPTION
基础框架	React	^16.9.0	
强类型	TypeScript	^3.8.3	
静态代码检查	Airbnb		
UI组件库	Ant-design	^4.2.0	
状态管理	Redux		
国际化	React-intl		
样式	Styled-component		
不可变数据结构	Immer		
流程控制中间件	Redux-saga		
MORE			

## 选型说明及考量

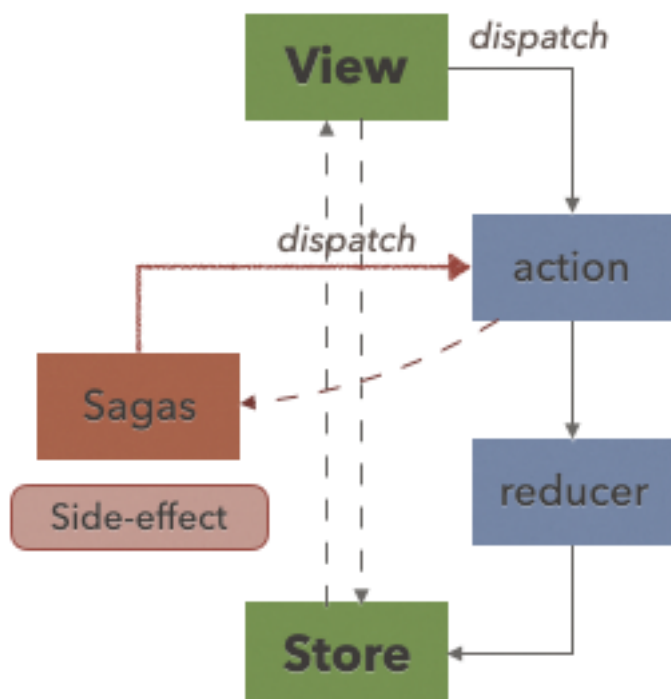
### REDUX

项目采用Redux进行全量的数据管理，虽然开发上会带来一定的负担，但是能够更清晰的维护数据流，让项目高可控，代码更易测试，有利于协作型项目写出更健壮的代码；

### REDUX-SAGA

Redux的副作用管理使用的是Redux-saga，当前已经编写了asyncSaga用以对异步请求做统一的处理，多数情况下我们是不需要单独写saga来进行一些短流程的处理，asyncSaga能够处理多数场景，并提供了一步回调。

引入saga，是由于saga针对复杂业务场景能够非常清晰的控制和处理，当你的开发遇到复杂的流程性处理时，请先想到saga。




---

## REACT-INTL

需求要求实现国际化版本，因此选用了常规的国际化选型React-intl，但是任何国际化方案都会为开发带来一定负担，国际化会要求页面中所有的文字都抽取成变量，如下图所示。具体的国际化方案会在后面详述。

```
return (
  <Modal
    title={intl.formatMessage(messages.default.userModalTitle)}
    visible={modalData.visible}
    onOk={handleOk}
    onCancel={handleCancel}
  >
    <Form form={form}>
      <Form.Item
        name="chinesename"
        label={intl.formatMessage(messages.default.chineseName)}
        rules={[
          { required: true },
        ]}
        initialValue={initialFormData.chineseName}
      >
```



---

## 类型控制

全局类型定义在 `ils/types/global.d.ts`，主要包含一些状态、权限、分页、弹窗等通用的类型定义，如果有业务相关的通用类型定义可在 `ils/types/business.d.ts` 中定义。

此外每个页面模块也都有自己的类型定义文件，在 `pages/*PageName*/types.ts`，类型定义应对照与后端的接口协议进行详细定义，比如对于供应商管理，我们通常需要定义：

供应商实体： `ISupplier`

检索参数： `ISearchCondition`

供应商列表Row： `ITableItem`

一些枚举类型等等

通常来讲，这些接口定义之间存在关联和包含的关系，可自行维护。

## 路由及权限管理

路由控制是根据Menu结构自动进行路由的生成，例如系统管理下的用户管理，路由即为 `system/userManage`，这个映射的核心逻辑在 `ils/app/components/Route` 中进行维护的。

### 新增路由

1. 在Menu配置文件 `ils/app/configs/menu.conf` 中按照结构新增菜单项；
2. 在 `ils/app/pages/` 目录下新增与路由末端同名文件夹（首字母大写）即可，此时新增路由会自动拉取此文件夹下的 `index` 文件；
3. 部分子页面在Menu中不需要展示（也就是不能通过菜单直达页面），但是需要生成对应的路由，此时在 `menu.conf` 中将其配置为不可见即可，此时路由仍然会按照层级关系构建，但菜单不外露此路由；

### 页面权限

---

---

页面级的用户权限也在menu.conf中进行维护，因此在页面开发时不需要关注用户页面级权限控制。由于具体的权限控制模式还未与RD协定，后期会在此做一补充。

## 页面内功能权限

页面内功能权限控制会在ils/app/controllers/AuthControl中对外暴露方法，供业务方直接使用，登录用户权限的获取会统一进行处理，不需要也不应页面独立处理。

## 面包屑

面包屑与Menu的active状态是自动同步的，除去带自定义信息的面包屑需要单独处理，其他理应自动匹配。

## 国际化

上面有提到，国际化方案使用的是react-intl，国际化必定要将文字抽取成message词条，因此message的具体维护方式很重要。在过往经验中发现对于一些枚举值的处理是国际化中非常重要的一个环节，比如说用户状态有离职、在职两种，如果独立维护一份id值与变量名的枚举，再维护一份语言版本的词条会将有效信息割裂开来，非常不利于后期维护，因此这里将国际化词条与枚举值进行了统一的维护。

```
// 这里是枚举值及下拉选择的一般方式
export const accountStatusMap = defineMessages({
  1: {
    id: `${scope}.accountStatusMap.1`,
    defaultMessage: '在职',
  },
  2: {
    id: `${scope}.accountStatusMap.2`,
    defaultMessage: '离职',
  },
});
```

```

<Form.Item
  name="role"
  label={intl.formatMessage(messages.default.role)}
  rules={[{required: true}]}
  initialValue={initialFormData.role}
>
  <Select allowClear>
    {Object.keys(messages.roleMap).map(key => (
      <Option value={key} key={key}>
        {intl.formatMessage((messages.roleMap as any)[key]))}
      </Option>
    ))}
  </Select>
</Form.Item>

```

上面为一具体使用实例其中 **intl** 通过HOC高阶组件进行注入，在职离职的下拉选择器直接使用国际化的 message自动进行生成。

此外，messages的维护也分了global、pageMessage两部分，对于一些通用的词条请维护在ils/app/global/messages.ts中，各个页面的messages对于枚举的部分也都独立导出。

## 菜单与面包屑文字的国际化

为了快捷开发，菜单与面包屑自动进行了国际化处理，但是需要将菜单对应的名称写入message文件，示例如下：

## 样式方案

对于中后台系统，我们的原则是尽量少的去写样式，对于不同情况有以下几个原则：

1. 框架级全局样式维护在ils/app/theme文件夹中，这里一般是与antd相关的主题样式及全局的样式覆盖。
2. 各个页面的独立样式推荐使用**styled-component**完成，或者对于复杂样式使用附加less文件的方式，在JSX中禁止超过三项的style写法。

```

<Form.Item>
  <Button onClick={handleSearch} style={{width: 30, height: 30, flex: 1, ...}}>{intl.formatMessage(globalMessages.search)}</Button>
  <Button onClick={handleOpenModal}>{intl.formatMessage(globalMessages.create)}</Button>
</Form.Item>

```

```
const Circle = (props: IProps) => {
  const { rotate, delay } = props;
  const CirclePrimitive = styled.div`
    width: 100%;
    height: 100%;
    position: absolute;
    left: 0;
    top: 0;
    ${rotate}
    &&`
    -webkit-transform: rotate(${rotate}deg);
    -ms-transform: rotate(${rotate}deg);
    transform: rotate(${rotate}deg);
  ` &:before {
    content: '';
    display: block;
    margin: 0 auto;
    width: 15%;
    height: 15%;
    background-color: #999;
    border-radius: 100%;
    animation: ${circleFadeDelay} 1.2s infinite ease-in-out both;
    ${delay}
    &&`
    -webkit-animation-delay: ${delay}s;
    animation-delay: ${delay}s;
  `;
}
```

styled-component提供的能力已经非常足够，且能够与props直接进行关联，非常推荐。

3. 可开发一些包裹器组件，来完成样式的统一，比如说Table、Form等。

## 公共状态数据

当前公共状态数据主要包含五部分：

1. 登录用户信息及用户功能权限、数据权限；
2. 当前语言，国际化相关；
3. 当前App状态，Loading、Error状态的维护；



---

4. layout样式控制，比如说左侧栏收起；

5. history等路由相关信息；

因此对于上述五类数据的获取，请使用global/selectors.ts中提供的selector完成数据获取，务必不要再行独立获取。

## 公共组件抽取

针对当前需求，以下公共组件需要进行抽取：

菜单

面包屑

区域选择组件

用户数据权限级联选择组件

---

---

## APP 文件结构

```
|-----components    // 公共组件文件夹
|-----configs      // 配置文件夹
|       |--base.conf   // 项目公共信息配置，比如说默认分页
|       |--auth.conf   // 功能权限对应规则
|       |--i18n.conf   // 国际化配置
|       |--menu.conf   // 菜单及路由配置 *非常重要
|       |--proxy.conf  // 开发代理配置
|-----containers    // 公共数据容器文件夹，可以理解为带数据的组件
|-----hooks         // 公共hooks文件夹
|-----pages         // 页面，实际上也都是containers，为了便于理解，与containers进行了拆分
|       |--PageName    // 大驼峰
|           |--index.tsx // 入口文件
|           |--modules  // 可以理解为通过connect store带数据的组件
|           |--components // UI组件，不带数据组件，如若无复用性，可简单的归为modules
|           |--constants.ts // 常量定义
|           |--actions.ts
|           |--reducer.ts
|           |--selectors.ts
|           |--services.ts
|           |--saga.ts
|           |--types.ts  // 类型定义
|-----static        // 图片、文件等静态资源文件夹
|-----theme         // 主题样式文件夹
|-----translations  // 国际化词条及翻译文件，前期不用关注
|-----global        // 公共状态
|       |--selectors.ts // 公共状态selector，所有的公共状态通过此获取
|       |--actions.ts   // 公共状态修改action
|       |--globalReducer.ts // 公共状态reducer
|       |--asyncSaga.ts // 统一异步处理saga
```

---

---

```
|-----utils      // 公共工具、方法、数据
    |--messages.ts  // 公共国际化词条
    |--constants.ts // 公共常量
    |--renders.ts   // 公共渲染器
    |--validators.ts // 公共校验器
    |--utils.ts     // 公共的方法
```

---

# 开发方案优化

## 1. 热更新

快捷热更新；

## 2. 代理及Mock能力

在后端开发没有ready时，能够通过代理到EasyMock来进行开发，在联调阶段也能够提供代理到RD环境进行联调的能力，且有方案针对需要登录的case进行联调。

EasyMock地址：<http://easy-mock.sftcwl.com/project/5ebc25690e1f322cd339b684>

## 3. 自动服务重启

针对DevServer的某些修改，能够自动进行服务重启。

## 4. PROD部署方案

提供命令完成prod模式下的代码部署。

## 5. 部分编译能力

项目达到一定体量后，全量编译速度会受到影响，提供部分编译能力以进行快速热更新响应。

## 6. Commit 标准化钩子

## 7. 自动化lint检查

等等等

---

---

## 后续配套建设

### 开发脚手架

新项目使用此脚手架进行快速生成。

### 快速开发工具

迭代项目可使用工具进行快速且标准的页面生成、组件生成等。

### 分支协作及Git使用规范

### 中后台标准交互规范

### 公共组件、容器、hooks抽象规范

### 代码提交规范

---

---

## 特殊说明与折中

\*以上门类未能涵盖的框架特性，可在此做一说明\*