# 前端开发流程：

## 一．环境

Git clone 项目到本地

Yarn install 安装包

Yarn start 运行开发环境

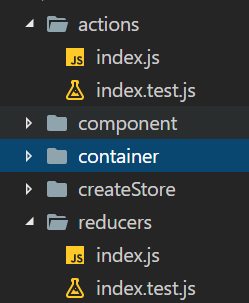
Yarn test 运行测试

关于单元测试，现阶段只对

Actions和 Reducer测试

我们一开始先只对Aciton 和 Reducer 进行测试，关于React组件，等我们之后开发到一定阶段后再考虑增进。

测试文件以\*.test.js 命名，存放在被测试文件目录



## 关于Action:

export const addTodo = (text) => ({

    type: 'ADD\_TODO',

    id: nextTodoId++,

    text,

})

需要单独进行 export导出

describe('测试Atcion 中 Addtodo 方法', () => {

it('should create an action to add a todo', () => {

const text = 'Finish docs'

const expectedAction = {

type: 'ADD\_TODO',

id: 0,

text:text

}

expect(actions.addTodo(text)).toEqual(expectedAction)

})

})

测试时进行引用。

Expect ( 函数调用 ) . toEqual(对象)

还可以toBe 等等匹配

详细可以看：

http://facebook.github.io/jest/docs/zh-Hans/using-matchers.html#content

## 关于Reducer

测试文件以\*.test.js 命名，存放在被测试文件目录

export const todos = (state = [], action) => {

    switch(action.type) {

        case 'ADD\_TODO':

         return [

                ...state,

                {

                 id: action.id,

                 text: action.text,

                 isCompleted: false

             }

            ];

        case 'TOGGLE\_TODO':

            return state.map(todo => {

                if(todo.id !== action.id) {

                    return todo;

                } else {

                    return {

                        ...todo,

                        isCompleted: !todo.isCompleted

                    }

                }

            });

        default:

            return state;

    }

}

也需要对函数单独的export导出

import { todos as todosReducer } from './index';

describe('测试Reducer 中 Addtodo 方法', () => {

it('should create an action to add a todo', () => {

const text = 'Finish docs'

const action = {

type: 'ADD\_TODO',

id: 0,

text: text

}

const initialState = [

{

text: 'Use Redux',

completed: false,

id: 0

}

]

const expectedAction =

[{ "id": undefined, "isCompleted": false, "text": "Run the tests" }]

expect(todosReducer([], {

type: 'ADD\_TODO',

text: 'Run the tests'

})).toEqual(expectedAction)

})

})

测试记得需要考虑测试

1.调用函数传进一个参数是否和预期相等

2.当内容为空，调用删除函数是否有正确的处理。

# 开发约定:

变量命名：

1普通变量命名: 变量 (包括函数参数) 和数据成员名一律小写(var todo ), 单词之间用下 划线连接(var todo\_list).

2类数据成员:不管是静态的还是非静态的, 类数据成员都可以和普通变量一样, 但要接下划线.类的成员变量以下划线结尾。

3常量命名：声明为const的变量。可大小写混合使用。（actions 中 const toggleTodo ，const addTodo ， const setVisibilityFilter）所有具有静态存储类型的变量 (例如静态变量或全局变量, 参见存储类型) 都应当以此方式命名.如果不采用这条规则, 就按照一般的变量命名规则.

函数命名：

函数名的每个单词首字母大写 (即 “驼峰变量名”） 没有下划线. 对于首字母缩写的单词, 更倾向于将它们视作一个单词进行首字母大写。（LoginEvent(), onClick()）

组件规范：

函数式声明组件：class TodoList extends React.Component { ...}

组件名称首字母必须大写，并且以组件的功能为其命名。

按组件的特点分为复用组件和容器组件。复用组件可在各个地方进行复用，由父组件向其传递数据，复用组件使用this.props.属性名进行获取。

（父组件：<ParentComponent options={data}/>,

子组件： <ChildComponent data={this.props.options}/>）

组件调用：要使用组件时，import 组件，在当前组件内进行调用。

import TodoList from ‘ compomnent/TodoList ’

class AddTodo extends Component {

constructor(props){

Super(props)

}

render() {

return (

<TodoList />

)

}

}

export default AddTodo

4、

class TestApp extends Component {

constructor(props) { // 如果不是纯组件，都写constructor

super(props);

this.state = {

name: 'real',

}

}

\_onClick(text) {

alert(1)

}

render() {

// console.log(this.props);

const { dispatch, getState } = this.props.store;

return (

<div>

<Button type="primary" onClick={(text) => { this.\_onClick(text) }}>Button</Button>

</div>

)

}

}

函数的调用，如上述的onClick={(text) => { this.\_onClick(text)}}，有两种方式：

①onClick = { this.\_onClick }

\_onClick = () => {} // 写函数时使用箭头函数

②onClick = { () => { this.\_onClick } } // 这里用箭头

\_onClick() {}