**深圳天源迪科信息技术股份有限公司**

DIC-TS-IT-NWMO-XZ-1.0 版 本：1.0

状态:WT

**Dva前端框架使用手册**

编制 : 李林、费庭均、纪卓玲

审核 :

日期 : 2017-03-16

**文档修订历史纪录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **编制** | **审核** |
| 2017-03-16 | 1.0 | 创建 |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

本文件属深圳天源迪科信息技术股份有限公司所有，

未经书面许可，不得以任何形式复印或传播。

目录

[1 dva介绍 4](#_Toc14093)

[2 环境配置 4](#_Toc22135)

[3 基本技能 4](#_Toc14325)

[3.1 ECMAScript 4](#_Toc16821)

[变量声明 const 和 let 4](#_Toc4471)

[箭头函数 5](#_Toc13342)

[模块的 Import 和 Export 5](#_Toc19350)

[ES6 对象和数组](#_Toc10068)

[3.2 React 8](#_Toc3155)

[Jsx语法 8](#_Toc6302)

[组件 9](#_Toc11953)

[React Component 有 3 种定义方式 9](#_Toc19994)

[PropTypes属性验证 10](#_Toc23330)

[组件的生命周期 10](#_Toc18609)

[3.3 React Router 11](#_Toc32447)

[基本用法 11](#_Toc27223)

[嵌套路由（容器组件加载子页面使用） 12](#_Toc8309)

[IndexRoute 组件 13](#_Toc27860)

[Link组件 13](#_Toc28476)

[3.4 Redux 13](#_Toc22118)

[设计思想 13](#_Toc9969)

[Store(数据仓库) 14](#_Toc28583)

[State（当前数据） 14](#_Toc16688)

[Action （触发事件） 14](#_Toc31108)

[store.dispatch() （触发器） 14](#_Toc31214)

[Reducer（接收器） 14](#_Toc11247)

[3.5 React-Redux 15](#_Toc11739)

[UI 组件 15](#_Toc2878)

[容器组件 15](#_Toc2258)

[connect() 15](#_Toc20812)

[mapStateToProps() 16](#_Toc27883)

[mapDispatchToProps() 17](#_Toc20697)

[4 安装 18](#_Toc32482)

[安装 dva-cli 18](#_Toc11454)

[创建应用 18](#_Toc27378)

[构建工具 18](#_Toc18928)

[5 使用说明 19](#_Toc19800)

[5.1 项目结构说明 19](#_Toc23143)

[5.2 主入口js说明 20](#_Toc17430)

[5.3 路由配置说明 20](#_Toc4191)

[5.4页面开发说明 21](#_Toc7093)

[5.4.1 定义主视图 21](#_Toc17555)

[5.4.2 初始化store及关联视图与model 21](#_Toc22283)

[5.4.3 定义model及交互事件 22](#_Toc28512)

[5.4.4 Ajax请求 22](#_Toc4696)

[5.4.5 项目运行 23](#_Toc5109)

[5.4.6 打包项目 23](#_Toc1086)

[6 ant design 组件加载 23](#_Toc30196)

[6.1 antd简介 23](#_Toc13136)

[6.2 Ant Design of React 24](#_Toc30003)

[6.3 支持环境 24](#_Toc27987)

[6.4 安装 24](#_Toc2181)

[6.5 文件配置（重要） 24](#_Toc13679)

[6.6 组件使用 24](#_Toc22438)

[6.6.1 组件引入 24](#_Toc18642)

[6.6.2 使用组件 25](#_Toc13769)

[7 可视化组件加载 25](#_Toc5535)

[7.1 Adtv g2 chart 25](#_Toc2395)

[7.1.1 基本步骤 26](#_Toc7709)

[7.1.2 在react中使用G2 28](#_Toc21225)

[7.2 EChart 30](#_Toc21113)

[7.2.1 基本步骤 30](#_Toc21597)

[7.2.2 在react中使用echarts 32](#_Toc32166)

# 1 dva介绍

[dva](https://github.com/dvajs/dva) 是一个基于 react 和 redux 的轻量应用框架，没有引入任何新概念，概念来自 elm，支持 side effects、热替换、动态加载、react-native、SSR 等，已在生产环境广泛应用。

dva 是 framework，不是 library，类似 emberjs，会很明确地告诉你每个部件应该怎么写，这对于团队而言，会更可控。另外，除了 react 和 react-dom 是 peerDependencies 以外，dva 封装了所有其他依赖。

# 2 环境配置

确保 Node 版本 6.5+

# 3 基本技能

Dva框架主要使用Es6、React、React-Router、Redux，以下会对dva框架中常用的基本技能使用进行讲解，更多内容需要使用者更深入的学习，参考http://www.ruanyifeng.com/blog/2016/09/react-technology-stack.html React技术栈教程。

掌握react+redux的使用者可以跳过本章。

## 3.1 ECMAScript 6

### **变量声明 const 和 let**

不要用 var，而是用 const 和 let，分别表示常量和变量。不同于 var 的函数作用域，const 和 let 都是块级作用域，只在当前代码块有效。

const DELAY = 1000;

let count = 0;

count = count + 1;

#### **模板字符串**

模板字符串提供了另一种做字符串组合的方法。

const user = 'world';console.log(`hello ${user}`); // hello world

// 多行const content = ` Hello ${firstName}, Thanks for ordering ${qty} tickets to ${event}.`;

#### **默认参数**

无参数传入结果为默认参数，有参数则为当前参数

function logActivity(activity = 'skiing') {

console.log(activity);

}

logActivity(); // skiing

### **箭头函数**

函数的快捷写法，不需要通过 function 关键字创建函数，并且还可以省略 return 关键字。

同时，箭头函数还会继承当前上下文的 this 关键字。

比如：

[1, 2, 3].map(x => x + 1); // [2, 3, 4]

等同于：

[1, 2, 3].map((function(x) {

return x + 1;

}).bind(this));

### **模块的 Import 和 Export**

import 用于引入模块，export 用于导出模块。

比如：

// 引入全部import dva from 'dva';

// 引入部分import { connect } from 'dva';import { Link, Route } from 'dva/router';

// 引入全部并作为 github 对象import \* as github from './services/github';

// 导出默认export default App;// 部分导出，需 import { App } from './file'; 引入export class App extend Component {};

### **ES6 对象和数组**

#### **析构赋值**

析构赋值让我们从 Object 或 Array 里取部分数据存为变量。

// 对象const user = { name: 'guanguan', age: 2 };const { name, age } = user;console.log(`${name} : ${age}`); // guanguan : 2

// 数组const arr = [1, 2];const [foo, bar] = arr;console.log(foo); // 1

我们也可以析构传入的函数参数。

const add = (state, { payload }) => {

return state.concat(payload);

};

析构时还可以配 alias，让代码更具有语义。

const add = (state, { payload: todo }) => {

return state.concat(todo);

};

#### **对象字面量改进**

这是析构的反向操作，用于重新组织一个 Object 。

const name = 'duoduo';const age = 8;

const user = { name, age }; // { name: 'duoduo', age: 8 }

定义对象方法时，还可以省去 function 关键字。

app.model({

reducers: {

add() {} // 等同于 add: function() {}

},

effects: {

\*addRemote() {} // 等同于 addRemote: function\*() {}

},

});

#### **Spread Operator**

Spread Operator 即 3 个点 ...，有几种不同的使用方法。

可用于组装数组。

const todos = ['Learn dva'];

[...todos, 'Learn antd']; // ['Learn dva', 'Learn antd']

也可用于获取数组的部分项。

const arr = ['a', 'b', 'c'];const [first, ...rest] = arr;

rest; // ['b', 'c']

// With ignoreconst [first, , ...rest] = arr;

rest; // ['c']

还可收集函数参数为数组。

function directions(first, ...rest) {

console.log(rest);

}directions('a', 'b', 'c'); // ['b', 'c'];

代替 apply。

function foo(x, y, z) {}const args = [1,2,3];

// 下面两句效果相同foo.apply(null, args);foo(...args);

对于 Object 而言，用于组合成新的 Object 。(ES2017 stage-2 proposal)

const foo = {

a: 1,

b: 2,

};const bar = {

b: 3,

c: 2,

};const d = 4;

const ret = { ...foo, ...bar, d }; // { a:1, b:3, c:2, d:4 }

## 3.2 React

### **Jsx语法**

### HTML 语言直接写在 JavaScript 语言之中，不加任何引号，这就是 JSX 的语法，它允许 HTML 与 JavaScript 的混写。

var names = ['Alice', 'Emily', 'Kate'];

ReactDOM.render(

<div>

{

names.map(function (name) {

return <div>Hello, {name}!</div>

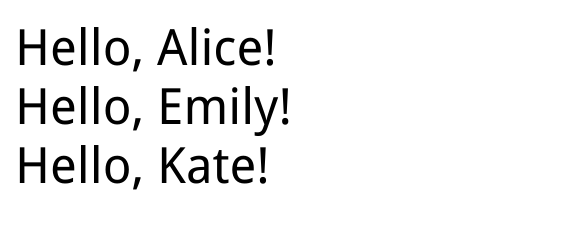
})

}

</div>,

document.getElementById('example'));

### 上面代码体现了 JSX 的基本语法规则：遇到 HTML 标签（以 < 开头），就用 HTML 规则解析；遇到代码块（以 { 开头），就用 JavaScript 规则解析。上面代码的运行结果如下。



### **组件**

React 允许将代码封装成组件（component），然后像插入普通 HTML 标签一样，在网页中插入这个组件。React.createClass 方法就用于生成一个组件类。

var HelloMessage = React.createClass({

render: function() {

return <h1>Hello {this.props.name}</h1>;

}});

ReactDOM.render(

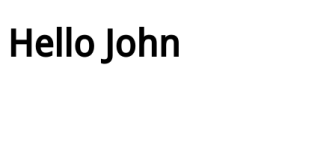
<HelloMessage name="John" />,

document.getElementById('example'));

上面代码中，变量 HelloMessage 就是一个组件类。模板插入 <HelloMessage /> 时，会自动生成 HelloMessage 的一个实例（下文的"组件"都指组件类的实例）。所有组件类都必须有自己的 render 方法，用于输出组件。

注意，组件类的第一个字母必须大写，否则会报错，比如HelloMessage不能写成helloMessage。另外，组件类只能包含一个顶层标签，否则也会报错。

组件的用法与原生的 HTML 标签完全一致，可以任意加入属性，比如 <HelloMessage name="John"> ，就是 HelloMessage 组件加入一个 name 属性，值为 John。组件的属性可以在组件类的 this.props 对象上获取，比如 name 属性就可以通过 this.props.name 读取。上面代码的运行结果如下。



添加组件属性，有一个地方需要注意，就是 class 属性需要写成className，for属性需要写成htmlFor，这是因为class和for是 JavaScript 的保留字。

### **React Component 有 3 种定义方式**

分别是 React.createClass, class 和 Stateless Functional Component。推荐尽量使用最后一种，保持简洁和无状态（主要针对使用Redux做状态管理时）。这是函数，不是 Object，没有 this 作用域，是 pure function。比如定义 App Component 。

function App(props) {

function handleClick() {

props.dispatch({ type: 'app/create' });

}

return <div onClick={handleClick}>${props.name}</div>

}

等同于：

class App extends React.Componnet {

handleClick() {

this.props.dispatch({ type: 'app/create' });

}

render() {

return <div onClick={this.handleClick.bind(this)}>${this.props.name}</div>

}

}

### **PropTypes属性验证**

组件的属性可以接受任意值，字符串、对象、函数等等都可以。有时，我们需要一种机制，验证别人使用组件时，提供的参数是否符合要求。

组件类的PropTypes属性，就是用来验证组件实例的属性是否符合要求

var MyTitle = React.createClass({

propTypes: {

title: React.PropTypes.string.isRequired,

},

render: function() {

return <h1> {this.props.title} </h1>;

}

})

var data = 123;

ReactDOM.render(

<MyTitle title={data} />,

document.body

);

以上代码会报错，data只能为字符串类型。

### **组件的生命周期**

组件的生命周期分成三个状态：

Mounting：已插入真实 DOM

Updating：正在被重新渲染

Unmounting：已移出真实 DOM

React 为每个状态都提供了两种处理函数，will 函数在进入状态之前调用，did 函数在进入状态之后调用，三种状态共计五种处理函数。

componentWillMount()

componentDidMount()

componentWillUpdate(object nextProps, object nextState)

componentDidUpdate(object prevProps, object prevState)

componentWillUnmount()

此外，React 还提供两种特殊状态的处理函数。

componentWillReceiveProps(object nextProps)：已加载组件收到新的参数时调用

shouldComponentUpdate(object nextProps, object nextState)：组件判断是否重新渲染时调用

具体使用请参考React官方文档

## 3.3 React Router

### **基本用法**

路由器Router就是React的一个组件。

import { Router } from 'react-router';

render(<Router/>, document.getElementById('app'));

Router组件本身只是一个容器，真正的路由要通过Route组件定义。

import { Router, Route, hashHistory } from 'react-router';

render((

<Router history={hashHistory}>

<Route path="/" component={App}/>

</Router>

), document.getElementById('app'));

上面代码中，用户访问根路由/（比如http://www.example.com/），组件APP就会加载到document.getElementById('app')。

你可能还注意到，Router组件有一个参数history，它的值hashHistory表示，路由的切换由URL的hash变化决定，即URL的#部分发生变化。举例来说，用户访问http://www.example.com/，实际会看到的是http://www.example.com/#/。

Route组件定义了URL路径与组件的对应关系。你可以同时使用多个Route组件。

<Router history={hashHistory}>

<Route path="/" component={App}/>

<Route path="/repos" component={Repos}/>

<Route path="/about" component={About}/>

</Router>

上面代码中，用户访问/repos（比如http://localhost:8080/#/repos）时，加载Repos组件；访问/about（http://localhost:8080/#/about）时，加载About组件。

### **嵌套路由（容器组件加载子页面使用）**

Route组件还可以嵌套。

<Router history={hashHistory}>

<Route path="/" component={App}>

<Route path="/repos" component={Repos}/>

<Route path="/about" component={About}/>

</Route>

</Router>

上面代码中，用户访问/repos时，会先加载App组件，然后在它的内部再加载Repos组件。

<App>

<Repos/>

</App>

App组件要写成下面的样子。

export default React.createClass({

render() {

return <div>

{this.props.children}

</div>

}

})

上面代码中，App组件的this.props.children属性就是子组件。

### **IndexRoute 组件**

显式指定根路由的子组件，即指定默认情况下加载的子组件。你可以把IndexRoute想象成某个路径的index.html。

<Router>

<Route path="/" component={App}>

<IndexRoute component={Home}/>

<Route path="accounts" component={Accounts}/>

<Route path="statements" component={Statements}/>

</Route>

</Router>

现在，用户访问/的时候，加载的组件结构如下。

<App>

<Home/>

</App>

### **Link组件**

Link组件用于取代<a>元素，生成一个链接，允许用户点击后跳转到另一个路由。它基本上就是<a>元素的React 版本，可以接收Router的状态。

render() {

return <div>

<ul role="nav">

<li><Link to="/about">About</Link></li>

<li><Link to="/repos">Repos</Link></li>

</ul>

</div>

}

## 3.4 Redux

### **设计思想**

Redux 的设计思想很简单，就两句话。

（1）Web 应用是一个状态机，视图与状态是一一对应的。

（2）所有的状态，保存在一个对象里面。

### **Store(数据仓库)**

Store 就是保存数据的地方，你可以把它看成一个容器。整个应用只能有一个 Store。

Redux 提供createStore这个函数，用来生成 Store。

### **State（当前数据）**

Store对象包含所有数据。如果想得到某个时点的数据，就要对 Store 生成快照。这种时点的数据集合，就叫做 State。

当前时刻的 State，可以通过store.getState()拿到。

import { createStore } from 'redux';

const store = createStore(fn);

const state = store.getState();

Redux 规定， 一个 State 对应一个 View。只要 State 相同，View 就相同。

### **Action （触发事件）**

State 的变化，会导致 View 的变化。但是，用户接触不到 State，只能接触到 View。所以，State 的变化必须是 View 导致的。Action 就是 View 发出的通知，表示 State 应该要发生变化了。

Action 是一个对象。其中的type属性是必须的，表示 Action 的名称。

const action = {

type: 'ADD\_TODO',

payload: 'Learn Redux'

};

上面代码中，Action 的名称是ADD\_TODO，它携带的信息是字符串Learn Redux。

可以这样理解，Action 描述当前发生的事情。改变 State 的唯一办法，就是使用 Action。它会运送数据到 Store。

### **store.dispatch() （触发器）**

store.dispatch()是 View 发出 Action 的唯一方法。

store.dispatch({

type: 'ADD\_TODO',

payload: 'Learn Redux'

});

上面代码中，store.dispatch接受一个 Action 对象作为参数，将它发送出去。

### **Reducer（接收器）**

Store 收到 Action 以后，必须给出一个新的 State，这样 View 才会发生变化。这种 State 的计算过程就叫做 Reducer。

Reducer 是一个函数，它接受 Action 和当前 State 作为参数，返回一个新的 State。

const reducer = (state = defaultState, action) => {

switch (action.type) {

case ''ADD\_TODO':

return state + action.payload;

default:

return state;

}

};

Reducer是纯函数，只承担计算 State 的功能

## 3.5 React-Redux

### **UI 组件**

React-Redux 将所有组件分成两大类：UI 组件（presentational component）和容器组件（container component）。

UI 组件有以下几个特征。

只负责 UI 的呈现，不带有任何业务逻辑

没有状态（即不使用this.state这个变量）

所有数据都由参数（this.props）提供

不使用任何 Redux 的 API

下面就是一个 UI 组件的例子。

const Title =

value => <h1>{value}</h1>;

因为不含有状态，UI 组件又称为"纯组件"，即它纯函数一样，纯粹由参数决定它的值。

### **容器组件**

容器组件的特征恰恰相反。

负责管理数据和业务逻辑，不负责 UI 的呈现

带有内部状态

使用 Redux 的 API

总之，只要记住一句话就可以了：UI 组件负责 UI 的呈现，容器组件负责管理数据和逻辑。

### **connect()**

React-Redux 提供connect方法，用于从 UI 组件生成容器组件。connect的意思，就是将这两种组件连起来。

import { connect } from 'react-redux'

const VisibleTodoList = connect(

mapStateToProps, /\*输入逻辑：外部的数据（即state对象）如何转换为 UI 组件的参数\*/

mapDispatchToProps/\*用户发出的动作如何变为 Action 对象，从 UI 组件传出去\*/

)(TodoList)

上面代码中，TodoList是 UI 组件，VisibleTodoList就是由 React-Redux 通过connect方法自动生成的容器组件

### **mapStateToProps()**

mapStateToProps是一个函数。它的作用就是像它的名字那样，建立一个从（外部的）state对象到（UI 组件的）props对象的映射关系。

作为函数，mapStateToProps执行后应该返回一个对象，里面的每一个键值对就是一个映射。请看下面的例子。

const mapStateToProps = (state) => {

return {

todos: getVisibleTodos(state.todos, state.visibilityFilter)

}

}

上面代码中，mapStateToProps是一个函数，它接受state作为参数，返回一个对象。这个对象有一个todos属性，代表 UI 组件的同名参数，后面的getVisibleTodos也是一个函数，可以从state算出 todos 的值。

下面就是getVisibleTodos的一个例子，用来算出todos。

const getVisibleTodos = (todos, filter) => {

switch (filter) {

case 'SHOW\_ALL':

return todos

case 'SHOW\_COMPLETED':

return todos.filter(t => t.completed)

case 'SHOW\_ACTIVE':

return todos.filter(t => !t.completed)

default:

throw new Error('Unknown filter: ' + filter)

}

}

mapStateToProps会订阅 Store，每当state更新的时候，就会自动执行，重新计算 UI 组件的参数，从而触发 UI 组件的重新渲染。

### **mapDispatchToProps()**

mapDispatchToProps是connect函数的第二个参数，用来建立 UI 组件的参数到store.dispatch方法的映射。也就是说，它定义了哪些用户的操作应该当作 Action，传给 Store。它可以是一个函数，也可以是一个对象。

如果mapDispatchToProps是一个函数，会得到dispatch和ownProps（容器组件的props对象）两个参数。

const mapDispatchToProps = (

dispatch,

ownProps

) => {

return {

onClick: () => {

dispatch({

type: 'SET\_VISIBILITY\_FILTER',

filter: ownProps.filter

});

}

};

}

从上面代码可以看到，mapDispatchToProps作为函数，应该返回一个对象，该对象的每个键值对都是一个映射，定义了 UI 组件的参数怎样发出 Action。

如果mapDispatchToProps是一个对象，它的每个键名也是对应 UI 组件的同名参数，键值应该是一个函数，会被当作 Action creator ，返回的 Action 会由 Redux 自动发出。举例来说，上面的mapDispatchToProps写成对象就是下面这样。

const mapDispatchToProps = {

onClick: (filter) => {

type: 'SET\_VISIBILITY\_FILTER',

filter: filter

};

}

# 4 安装

### **安装 dva-cli**

通过 npm 安装 dva-cli 并确保版本是 0.7.0 或以上。

$ npm install dva-cli -g

$ dva -v

0.7.0

### **创建应用**

$ dva new youApp 创建项目包

$ cd youApp 进入项目包

### **构建工具**

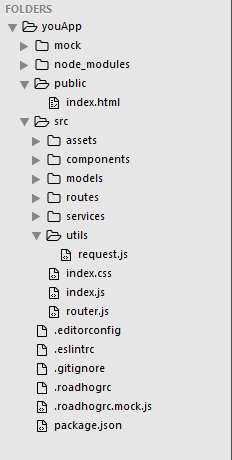
dva构建工具为roadhogrc，提供server和build功能

# 5 使用说明

通过 dva new 命令创建项目结构如下：

## 5.1 项目结构说明

入口页面



组件

数据模型js，定义数据交互规则

主页面视图，对应每个路由路径

Ajax请求js，定义每个视图所有的ajax请求规范

基于dva/fetch对ajax二次封装

主入口js，定义根元素，加载路由，加载model，加载插件，初始化数据

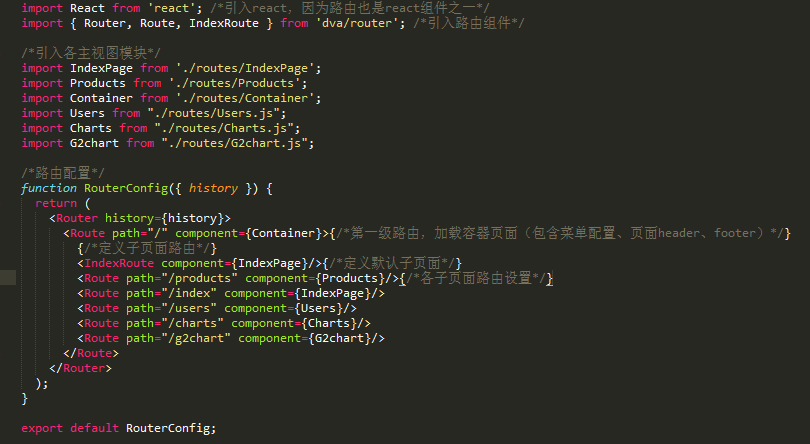
路由配置文件

Roadhogrc配置文件

## 5.2 主入口js说明



## 5.3 路由配置说明



## 5.4页面开发说明

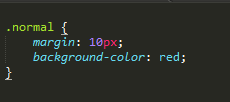
### 5.4.1 定义主视图

在routes中创建主视图js和css，开发规则：一个子页面对应一个routes主视图+一个models数据模型+一个services请求模块+一个或多个component组件视图。

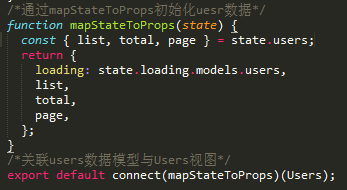
视图js说明及组件使用



视图css

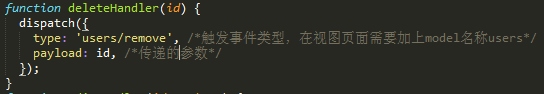


### 5.4.2 初始化store及关联视图与model

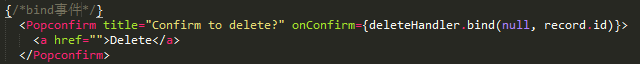


### 5.4.3 定义model及交互事件

定义触发事件



绑定事件



在models中创建视图对应model，定义数据model及调用规则

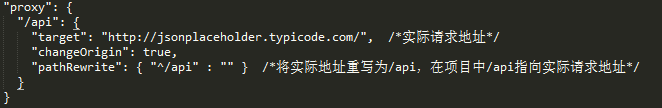


### 5.4.4 Ajax请求

在services中创建与视图对应的文件



在前后端分离式开发中，可以在roadhogrc中配置远程请求代理，在不更改接口地址的情况下发起ajax请求



代理设置后，将本地服务的请求映射成了一个远程请求。

项目打包时，代理地址不会引入，而是会重当前服务地址发起请求，这样只要一开始约束好接口规范，前端项目包就可以直接放入服务中运行。

### 5.4.5 项目运行

$ npm start

启动服务，打开浏览器

### 5.4.6 打包项目

$ npm run build

打包完成的工程在根目录dist下面，将整个dist重命名放到服务中即可启动项目

# 6 ant design 组件加载

官网：https://ant.design/index-cn

## 6.1 antd简介

Ant Design 是一个致力于提升『用户』和『设计者』使用体验的中台设计语言。它模糊了产品经理、交互设计师、视觉设计师、前端工程师、开发工程师等角色边界，将进行 UE 设计和 UI 设计人员统称为『设计者』，利用统一的规范进行设计赋能，全面提高中台产品体验和研发效率。

## 6.2 Ant Design of React

是 Ant Design 的 React 实现，开发和服务于企业级后台产品。

## 6.3 支持环境

现代浏览器和 IE9 及以上。

支持服务端渲染。

[Electron](http://electron.atom.io/)

## 6.4 安装

推荐使用npm进行安装 如果你的网络环境不佳，推荐使用 [cnpm](https://github.com/cnpm/cnpm)

通过 npm 安装 antd 和 babel-plugin-import 。 babel-plugin-import 是用来按需加载 antd 的脚本和样式的

npm install antd babel-plugin-import --save

## 6.5 文件配置（重要）

Antd安装完成后，编辑src文件夹下的.roadhogrc文件 在development和production对象内的 extraBabelPlugins对象内部添加

["import", { "libraryName": "antd", "style": "css" }] （共两处） 如下：

"extraBabelPlugins": [

"transform-runtime",

["import", { "libraryName": "antd", "style": "css" }]//用于使babel-plugin-import 插件 ] //生效

## 6.6 组件使用

Antd安装及文件配置完成后，就可以在页面内使用antd提供的组件了。（注意：在自定义组件时，一个组件内部只能有一个根节点，所以如果在单独的组件文件内部使用其他组件时，不管是否只使用一个组件，都建议使用div标签包围）

### 6.6.1 组件引入

使用import引入antd组件

import { Table, Popconfirm, Button } from 'antd';

只需要引入需要使用的组件即可，具体组件类型及名称请详见官网：<https://ant.design/docs/react/introduce-cn>

### 6.6.2 使用组件

使用组件时，只需要将引入的组件当成标签使用即可（react使用JSX语法，可以直接在js文件中使用html标签，首字母大写的标签会默认为组件，首字母小写的会当成普通html标签）

<div>

<Button type="primary">Primary</Button>

</div>

type：定义按钮的类型，不同类型有不同的样式

在antd中，不同的组件拥有不同的属性和方法，具体属性及使用方法请详见官网

# 7 可视化组件加载

## 7.1 Adtv g2 chart

官网：<https://antv.alipay.com/index.html>

var data = [

{genre:"成都",sold:300},

{genre:"武汉",sold:250},

{genre:"北京",sold:500},

{genre:"深圳",sold:400},

{genre:"上海",sold:450},

{genre:"拉萨",sold:100},

]

const Pie=createG2(chart=>{

chart.source(data,{

genre:{alias:"城市"},

sold:{alias:"数量"}

});

chart.interval().position("genre\*sold").color('genre');

chart.render();

})

### 7.1.1 基本步骤

#### 1）、引入G2

可以直接引入文件



也可以使用npm安装



安装之后直接使用require或者import引入即可

#### 2）、创建容器

<div id=”c1”></div>

#### 3）、创建chart对象

var G2=require（“g2”）;

Var G2=new G2.chart（{

Id：”c1” ,

Width: 600,

Height:300

}）

Id:用于指定图表的容器 也可以用container:getElementById(“c1”)的方法指定

Width：指定canvas的高度

forceFit：用于指定宽度是否自适应 有false和true两个值，默认为false，设置之后width 不生效

Height：指定canvas的宽度

#### 4）、载入源数据

var data = [

{genre:"成都",sold:300},

{genre:"武汉",sold:250},

{genre:"北京",sold:500},

{genre:"深圳",sold:400}

]

Chart.source(data,{

genre:{alias:"城市"},//列定义，定义列属性显示的别名（横坐标值为genre对应的 //值，加上此属性后会将横坐标额外标记为城市），下同

sold:{alias:"数量"}}

)

Source:载入数据

方法名：chart.source(data,{colDef，namesArr});

Data：数据源数据，格式为JSON形式的数组

colDef：可选，用于数据字段的列定义，如设置数据的类型，显示别名，时间 类型的展示格式等，不同的数字类型的配置项不同

namesArr：作为附加字段，用于补全数据源包含的数据字段，常用于数据源含 有不同数据字段的记录。

更多参数及作用请详见官网https://antv.alipay.com/g2/api/chart.html

#### 5）、创建图表语法

chart.interval().position("genre\*sold").color('genre');

语法：chart.<geomType>().<attrType>(“dims”,[callback]);

geomType():规定以何种方式展现图表，如：点，线，面等

interval()：使用矩形或弧形，用面积展现图表）

具体属性和用途请详见https://antv.alipay.com/g2/api/geom.html#adjusttypes

attrType():图形属性类型，对应视觉通道

Position():位置属性的映射，用于确定由数据中的哪几个字段来确定数据在平 面坐标系的位置,即确定 x 轴和 y 轴的数据字段。

Color():确定不同类型的数据图形颜色以哪个数据区分

#### 6）、渲染图表

chart.render():

### 7.1.2 在react中使用G2

在react中使用g2与一般方法基本类似

#### 1）、安装g2-react

首先使用npm安装g2与g2-react

npm install g2 --save

Npm install g2-react --save

#### 2）、引入组件

然后在文件中引入

import createG2 from "g2-react";

import {Stat} from "g2";

#### 3）、创建chart对象

const Pie=createG2(chart=>{

chart.source(data,{

genre:{alias:"城市"},

sold:{alias:"数量"}

});

chart.interval().position("genre\*sold").color('genre');

chart.render();

})

#### 4）、使用Pie组件

然后直接将创建的Pie直接当成组件使用即可

<Pie

forceFit= {true}

height={500}

plotCfg={plotCfg}

/>

属性可以在组件使用时直接定义，也可以在定义chart对象是定义

更多方法和属性请详见官网

## 7.2 EChart

官网：<http://echarts.baidu.com/>

### 7.2.1 基本步骤

#### 1）、引入echart

1、在官方网站下载页面下载EChart的完整版本

下载地址：<http://echarts.baidu.com/download.html>

然后直接在文件中引入

<script src=”echart.js”></script>

1. 在github上下载最新的release版本解压出来的文件夹里的 dist 目录里可以找到最新 版本的 echarts 库。
2. 通过 npm 获取 echarts

npm install echarts --save

1. cdn 引入，你可以在 [cdnjs](https://cdnjs.com/libraries/echarts" \t "http://echarts.baidu.com/_blank)，[npmcdn](https://npmcdn.com/echarts@latest/dist/" \t "http://echarts.baidu.com/_blank) 或者国内的 [bootcdn](http://www.bootcdn.cn/echarts/" \t "http://echarts.baidu.com/_blank)上找到 ECharts的最新版本。

#### 2）、创建容器

为ECharts创建一个具备宽高的容器

<div id="main" style="width: 600px;height:400px;"></div>

#### 3）、实例echarts

以下步骤写在script标签中

基于创建好的容器，初始化echarts的实例

var myChart = echarts.init(document.getElementById('main'));

用echarts.init方法创建一个echarts对象，并将其添加到id为main的容器中

#### 4）、指定图表的配置项和数据

更多属性及方法请详见官网：http://echarts.baidu.com/echarts2/doc/doc.html#Tooltip

var option = {

title: {

text: 'ECharts 入门示例'

},

tooltip: {},

legend: {

data:['销量']

},

xAxis: {

data: ["衬衫","羊毛衫","雪纺衫","裤子","高跟鞋","袜子"]

},

yAxis: {},

series: [{

name: '销量',

type: 'bar',

data: [5, 20, 36, 10, 10, 20]

}]

};

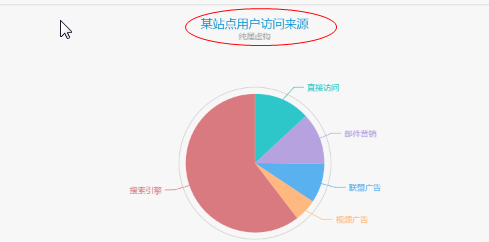
title：用于指定图表的标题内容及样式（如图红色标记），如：

Text：主标题文本

Link：主标题文本超链接

Left：主标题水平位置

更多属性见官网



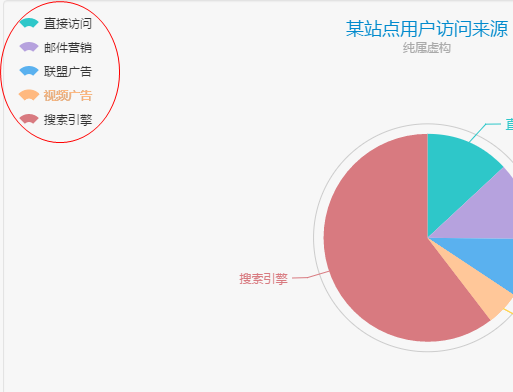
Tooltip：鼠标移入时显示的信息

Legend：图例，用来指示不同颜色的区域对应的数据

data：需要显示的数据

Left：放置的水平位置

更多属性见官网



Xaxis：横坐标轴 用于指定坐标轴的样式

坐标轴共有三种样式 类目型、数值型、时间型。不同类型的坐标轴需要选用不同 的属性，个别属性只能在特点类型的坐标轴下才会生效

yAxis：纵坐标轴，用途同上

Axis：坐标轴，用途同上 详情见http://echarts.baidu.com/echarts2/doc/doc.html#Axis

Series：驱动图表生成的数据内容的数组（也就是需要生成到图表中的数据，为一个数 组）。

根据不同是图表类型（柱状图或饼状图等），属性有所不同，某些属性只能在某些 类型的图表中使用，选用时请注意适用类型

Name：整个图表的标识，鼠标移入时会显示（还可用tooltip配置其他需要显示的 内容）

Type：指定图表的类型

bar：柱状图

pei：饼状图

更多详情请见官网

data：图表的数据，为一个数组，分别对应每一条数据

除此之外，在ECharts中还能配置编辑菜单，辅助线，图表类型切换，保存图片等其他 功能，详情请见官网

#### 5）、渲染数据

myChart.setOption(option);

setOption方法将我们定义的数据渲染到创建的echarts对象中

### 7.2.2 在react中使用echarts

在react中使用echarts时，图表参数配置基本不变

#### 1）、安装echarts-for-react

在react中，安装到项目的不再是echarts，而是echarts-for-react

npm install echarts-for-react --save

#### 2）、引入 echarts-for-react组件

在组件中使用charts时，需要先引入echarts-for-react模块，可以用require或者import

Var ReactEcharts = require(“echarts-for-react”) ;

Import ReactEcharts from “echarts-for-react”;

#### 3）、使用ReactEcharts组件

引入ReactEcharts（引入组件时定义的组件名）组件后，就可以直接使用react的语法 直接将ReactEcharts当成标签使用。

<div>

<ReactEcharts option={getOtion()} height={300} onEvents={onEvents} />

</div>

option属性：用于配置图表的数据参数，传入的为一个函数对象，如下：

可以用width和height配置图表的容器大小

其他属性及方法请参考

https://github.com/hustcc/echarts-for-react

<http://echarts.baidu.com/api.html#events>

function getOtion(){

return {

color: ['#3398DB'],

tooltip : {

trigger: 'axis',

axisPointer : { // 坐标轴指示器，坐标轴触发有效

type : 'shadow' // 默认为直线，可选为：'line' | 'shadow'

}

},

grid: {

left: '3%',

right: '4%',

bottom: '3%',

containLabel: true

},

xAxis : [

{

type : 'category',

data : ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'],

axisTick: {

alignWithLabel: true

}

}

],

yAxis : [

{

type : 'value'

}

],

series : [

{

name:'直接访问',

type:'bar',

barWidth: '60%',

data:[10, 52, 200, 334, 390, 330, 220]

}

]

};

}