# redux

尚硅谷前端研究院

## 1.0. 相关面试题

1. **redux结构图**
2. **redux的基本编码**
3. **说说你对redux的理解**
4. **区别UI组件与容器组件**
5. **何为高阶组件(higher order component)**
6. **详细说一下redux**

## 1.1. redux理解

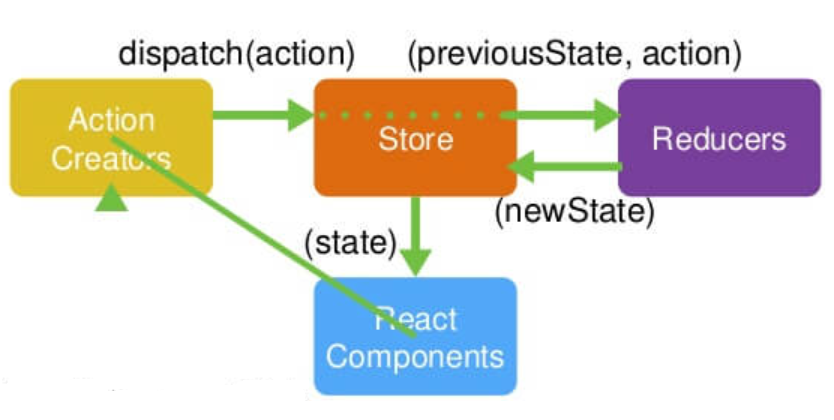
### 1.1.1. 学习文档

1. 英文文档: <https://redux.js.org/>
2. 中文文档: <http://www.redux.org.cn/>
3. Github: <https://github.com/reactjs/redux>

### 1.1.2. redux是什么?

1. redux是一个独立专门用于做状态管理的JS库(不是react插件库)
2. 它可以用在react, angular, vue等项目中, 但基本与react配合使用
3. 作用: 集中式管理(读/写)react应用中多个组件共享的状态

### 1.1.3. redux工作流程



### 1.1.4. 什么情况下需要使用redux

1. 总体原则: 能不用就不用, 如果不用比较吃力才考虑使用
2. 某个组件的状态，需要共享
3. 某个状态需要在任何地方都可以拿到
4. 一个组件需要改变全局状态
5. 一个组件需要改变另一个组件的状态

## 1.2. redux的核心API

### 1.2.1. createStore()

1. 作用:

创建包含指定reducer的store对象

1. 编码:

import {createStore} from 'redux'

import counter from './reducers/counter'

const store = createStore(counter)

### 1.2.2. store对象

1. 作用:

redux库最核心的管理对象

1. 它内部维护着:

state

reducer

1. 核心方法:

getState()

dispatch(action)

subscribe(listener)

1. 编码:

store.getState()

store.dispatch({type:'INCREMENT', number})

store.subscribe(render)

### 1.2.3. applyMiddleware()

1. 作用:

应用上基于redux的中间件(插件库)

1. 编码:

import {createStore, applyMiddleware} from 'redux'

import thunk from 'redux-thunk' // redux异步中间件

const store = createStore(

counter,

applyMiddleware(thunk) // 应用上异步中间件

)

### 1.2.4. combineReducers()

1. 作用:

合并多个reducer函数

1. 编码:

export default combineReducers({

user,

chatUser,

chat

})

## 1.3. redux的三个核心概念

### 1.3.1. action

1. 标识要执行行为的对象
2. 包含2个方面的属性
   1. type: 标识属性, 值为字符串, 唯一, 必要属性
   2. xxx: 数据属性, 值类型任意, 可选属性
3. 例子:

const action = {

type: 'INCREMENT',

data: 2

}

1. Action Creator(创建Action的工厂函数)

const increment = (number) => ({type: 'INCREMENT', data: number})

### 1.3.2. reducer

1. 根据老的state和action, 产生新的state的纯函数
2. 样例

export default function counter(state = 0, action) {

switch (action.type) {

case 'INCREMENT':

return state + action.data

case 'DECREMENT':

return state - action.data

default:

return state

}

}

1. 注意
   1. 返回一个新的状态
   2. 不要修改原来的状态

### 1.3.3. store

1. 将state,action与reducer联系在一起的对象
2. 如何得到此对象?

import {createStore} from 'redux'

import reducer from './reducers'

const store = createStore(reducer)

1. 此对象的功能?

getState(): 得到state

dispatch(action): 分发action, 触发reducer调用, 产生新的state

subscribe(listener): 注册监听, 当产生了新的state时, 自动调用

## 1.4. 使用redux编写应用

### 1.4.1. 效果



### 1.4.2. 下载依赖包

npm install --save redux

### 1.4.3. redux/action\_types.js

|  |
| --- |
| */\* action对象的type常量名称模块  \*/* **export const** INCREMENT = **'increment' export const** DECREMENT = **'decrement'** |

### 1.4.4. redux/actions\_creators.js

|  |
| --- |
| */\* action creator模块  \*/* **import** {INCREMENT, DECREMENT} **from './action-types'  export const** *increment* = number => ({**type**: INCREMENT, number}) **export const** *decrement* = number => ({**type**: DECREMENT, number}) |

### 1.4.5. redux/reducers.js

|  |
| --- |
| */\* 根据老的state和指定action, 处理返回一个新的state  \*/* **import** {INCREMENT, DECREMENT} **from '../constants/ActionTypes'  import** {INCREMENT, DECREMENT} **from './action-types'  export function** *counter*(state = 0, action) {  **console**.log(**'counter'**, state, action)  **switch** (action.**type**) {  **case** INCREMENT:  **return** state + action.**number  case** DECREMENT:  **return** state - action.**number  default**:  **return** state  } } |

### 1.4.6. components/app.jsx

|  |
| --- |
| */\* 应用组件  \*/* **import** React, {Component} **from 'react' import** PropTypes **from 'prop-types' import** \* **as** actions **from '../redux/actions'  export default class** App **extends** Component {   **static** *propTypes* = {  **store**: PropTypes.*object*.isRequired,  }   *increment* = () => {  **const** number = **this**.**refs**.numSelect.**value** \* 1  **this**.**props**.**store**.dispatch(actions.*increment*(number))  }   *decrement* = () => {  **const** number = **this**.**refs**.numSelect.**value** \* 1  **this**.**props**.**store**.dispatch(actions.*decrement*(number))  }   *incrementIfOdd* = () => {  **const** number = **this**.**refs**.numSelect.**value** \* 1   **let** count = **this**.**props**.**store**.getState()  **if** (count % 2 === 1) {  **this**.**props**.**store**.dispatch(actions.*increment*(number))  }  }   *incrementAsync* = () => {  **const** number = **this**.**refs**.numSelect.**value** \* 1  setTimeout(() => {  **this**.**props**.**store**.dispatch(actions.*increment*(number))  }, 1000)  }   render() {  **return** (  <**div**>  <**p**>  click {**this**.**props**.**store**.getState()} times {**' '**}  </**p**>  <**select ref="numSelect"**>  <**option value="1"**>1</**option**>  <**option value="2"**>2</**option**>  <**option value="3"**>3</**option**>  </**select**>{**' '**}  <**button onClick=**{**this**.*increment*}>+</**button**>  {**' '**}  <**button onClick=**{**this**.*decrement*}>-</**button**>  {**' '**}  <**button onClick=**{**this**.*incrementIfOdd*}>increment if odd</**button**>  {**' '**}  <**button onClick=**{**this**.*incrementAsync*}>increment async</**button**>  </**div**>  )  } } |

### 1.4.7. store.js

|  |
| --- |
| **import** React **from 'react' import** ReactDOM **from 'react-dom' import** {createStore} **from 'redux'  import** App **from './components/app' import** {*counter*} **from './redux/reducers'** *// 根据counter函数创建store对象* **const** store = createStore(*counter*)  *// 定义渲染根组件标签的函数* **const** *render* = () => {  ReactDOM.render(  <**App store=**{store}/>,  **document**.getElementById(**'root'**)  ) } *// 初始化渲染 render*()  *// 注册(订阅)监听, 一旦状态发生改变, 自动重新渲染* store.subscribe(*render*) |

### 1.4.8. 问题

1. redux与react组件的代码耦合度太高
2. 编码不够简洁

## 1.5. react-redux

### 1.5.1. 理解

1. 一个react插件库
2. 专门用来简化react应用中使用redux

### 1.5.2. React-Redux将所有组件分成两大类

1. UI组件
   1. 只负责 UI 的呈现，不带有任何业务逻辑
   2. 通过props接收数据(一般数据和函数)
   3. 不使用任何 Redux 的 API
   4. 一般保存在components文件夹下
2. 容器组件
   1. 负责管理数据和业务逻辑，不负责UI的呈现
   2. 使用 Redux 的 API
   3. 一般保存在containers文件夹下

### 1.5.3. 相关API

1. Provider

让所有组件都可以得到state数据

<Provider store={store}>  
    <App />  
  </Provider>

1. connect()

用于包装 UI 组件生成容器组件

import { connect } from 'react-redux'  
  connect(  
    mapStateToprops,  
    mapDispatchToProps  
  )(Counter)

1. mapStateToprops()

将外部的数据（即state对象）转换为UI组件的标签属性  
  const mapStateToprops = function (state) {  
   return {  
     value: state  
   }  
  }

1. mapDispatchToProps()

将分发action的函数转换为UI组件的标签属性

简洁语法可以直接指定为actions对象或包含多个action方法的对象

### 1.5.4. 使用react-redux

1. 下载依赖包

npm install --save react-redux

1. redux/action\_types.js

不变

1. redux/actions\_creators.js

不变

1. redux/reducers.js

不变

1. components/counter.jsx

|  |
| --- |
| */\* UI组件: 不包含任何redux API  \*/* **import** React **from 'react' import** PropTypes **from 'prop-types'  export default class** Counter **extends** React.Component {   **static** *propTypes* = {  **count**: PropTypes.*number*.isRequired,  **increment**: PropTypes.*func*.isRequired,  **decrement**: PropTypes.*func*.isRequired  }   *increment* = () => {  **const** number = **this**.**refs**.numSelect.**value** \* 1  **this**.**props**.**increment**(number)  }   *decrement* = () => {  **const** number = **this**.**refs**.numSelect.**value** \* 1  **this**.**props**.**decrement**(number)  }   *incrementIfOdd* = () => {  **const** number = **this**.**refs**.numSelect.**value** \* 1  **let** count = **this**.**props**.**count  if** (count % 2 === 1) {  **this**.**props**.**increment**(number)  }  }   *incrementAsync* = () => {  **const** number = **this**.**refs**.numSelect.**value** \* 1  setTimeout(() => {  **this**.**props**.**increment**(number)  }, 1000)  }   render() {  **return** (  <**div**>  <**p**>  click {**this**.**props**.**count**} times {**' '**}  </**p**>  <**select ref="numSelect"**>  <**option value="1"**>1</**option**>  <**option value="2"**>2</**option**>  <**option value="3"**>3</**option**>  </**select**>{**' '**}  <**button onClick=**{**this**.*increment*}>+</**button**>  {**' '**}  <**button onClick=**{**this**.*decrement*}>-</**button**>  {**' '**}  <**button onClick=**{**this**.*incrementIfOdd*}>increment if odd</**button**>  {**' '**}  <**button onClick=**{**this**.*incrementAsync*}>increment async</**button**>  </**div**>  )  } } |

1. containters/App.jsx

|  |
| --- |
| */\* 包含Counter组件的容器组件  \*/* **import** React **from 'react'** *// 引入连接函数* **import** {connect} **from 'react-redux'** *// 引入action函数* **import** {*increment*, *decrement*} **from '../redux/actions'  import** Counter **from '../components/counter'** *// 向外暴露连接App组件的包装组件* **export default** connect(  state => ({**count**: state}),  {*increment*, *decrement*} )(Counter) |

1. index.js

|  |
| --- |
| **import** React **from 'react' import** ReactDOM **from 'react-dom' import** {createStore} **from 'redux' import** {Provider} **from 'react-redux'  import** App **from './containers/app' import** {*counter*} **from './redux/reducers'** *// 根据counter函数创建store对象* **const** store = createStore(*counter*)  *// 定义渲染根组件标签的函数* ReactDOM.render(  (  <**Provider store=**{store}>  <**App** />  </**Provider**>  ),  **document**.getElementById(**'root'**) |

### 1.5.5. 问题

1. redux默认是不能进行异步处理的,
2. 应用中又需要在redux中执行异步任务(ajax, 定时器)

## 1.6. redux异步编程

### 1.6.1. 下载redux插件(异步中间件)

npm install --save redux-thunk

### 1.6.2. store.js

|  |
| --- |
| **import** {createStore, *applyMiddleware*} **from 'redux' import** thunk **from 'redux-thunk'** *// 根据reducer函数创建store对象* **const** store = createStore(  *reducer*,  *applyMiddleware(thunk)* *// 应用上异步中间件* ) |

### 1.6.3. redux/actions.js

|  |
| --- |
| *// 异步action creator(返回一个函数)* **export const** *incrementAsync* = number => {  **return** dispatch => {  setTimeout(() => {  dispatch(*increment*(number))  }, 1000)  } } |

### 1.6.4. components/counter.jsx

|  |
| --- |
| *incrementAsync* = () => {  **const** number = **this**.**refs**.numSelect.**value**\*1  **this**.**props**.**incrementAsync**(number) } |

### 1.6.5. containers/app.jsx

|  |
| --- |
| **import** {*increment*, *decrement*, *incrementAsync*} **from '../redux/actions'**  *// 向外暴露连接App组件的包装组件* **export default** connect(  state => ({**count**: state}),  {*increment*, *decrement*, *incrementAsync*} )(Counter) |

## 1.7. 使用上redux调试工具

### 1.7.1. 安装chrome浏览器插件



### 1.7.2. 下载工具依赖包

npm install --save-dev redux-devtools-extension

### 1.7.3. 编码

**Store.js**

|  |
| --- |
| **import** { **composeWithDevTools** } **from 'redux-devtools-extension'**  **const** store = createStore(  *reducer*,  **composeWithDevTools**(*applyMiddleware*(thunk)) ) |
|  |

## 1.8. 相关重要知识: 纯函数和高阶函数

### 1.8.1. 纯函数

1. 一类特别的函数: 只要是同样的输入(实参)，必定得到同样的输出(返回)
2. 必须遵守以下一些约束
   1. 不得改写参数数据
   2. 不会产生任何副作用，例如网络请求，输入和输出设备
   3. 不能调用Date.now()或者Math.random()等不纯的方法
3. redux的reducer函数必须是一个纯函数

### 1.8.2. 高阶函数

1. 理解: 一类特别的函数
   1. 情况1: 参数是函数
   2. 情况2: 返回是函数
2. 常见的高阶函数:
   1. 定时器设置函数
   2. 数组的forEach()/map()/filter()/reduce()/find()
   3. 函数对象的bind()
   4. Promise() / then()
   5. antd中的Form.create()()
   6. react-router-dom中的withRouter
   7. react-redux中的connect()
3. 作用:
   1. 能实现更加动态, 更加可扩展的功能