**react**

1. 用于构建用户界面的JavaScript库(UI框架)
2. react.js 是 React 的核心库，react-dom.js 是提供与 DOM 相关的功能，Browser.js 的作用是将 JSX 语法转为 JavaScript 语法，这一步很消耗时间，实际上线的时候，应该将它放到服务器完成。
3. ReactDOM.render 是 React 的最基本方法，用于将模板转为 HTML 语言，并插入指定的 DOM 节点。。
4. npm install react

**语法**

1. <script> 标签的 type 属性为 text/babel。
2. 组件标签要大写开头，
3. 标签可以自闭合，可以嵌套，
4. 标签之间的东西都叫chldren，组件标签之间的children通过this.props.children访问
5. JSX中表达式用{}包起来，有结果返回。
6. class 替换为className。
7. react类如何用：
   1. 声明react的组件
   2. 继承React.Component的类就是一个组件至少有一个render方法。
8. npm config set registry <https://registry.npm.tabao.org/>
9. react 的属性- props
   1. 传递属性：<Content a="0" />
   2. 接收属性: this.props
10. 只有setState才会改变页面
11. state

组件有内部状态

1. this.forceUpdate()//最好不要使用
2. 父组件的render执行，子组件的render也会执行。
3. ev.nativeEvent可以访问浏览器dom元素的事件对象
4. ev.target 拿到浏览器dom元素
5. 在元素上面声明ref属性，可以通过this.refs.你想要的元素的ref值，拿到真实的DOM元素。（不推荐了）
6. **ref**={(e) =>{  
    this.element = e;  
   }}
7. [组件的生命周期](https://segmentfault.com/a/1190000004168886)：

**状态：实例化，存在期，销毁时**

**当组件在客户端被实例化时，**

* 1. 以下方法被调用
     1. getDefaultProps(遗传基因)
     2. getInitialState（一个很NB的细胞诞生了）
     3. componentWillMount（怀孕了）
     4. **render**（怀孕中）
     5. **componentDidMount**（落地了）
  2. getDefaultProps 对于每个组件实例来讲，这个方法只会调用一次，该组件类的所有后续应用，getDefaultPops 将不会再被调用，其返回的对象可以用于设置默认的 props(properties的缩写) 值。
  3. getInitialState 对于组件的每个实例来说，这个方法的调用有且只有一次，用来初始化每个实例的 state，在这个方法里，可以访问组件的 props。每一个React组件都有自己的 state，其与 props 的区别在于 state只存在组件的内部，props 在所有实例中共享。
  4. getInitialState 和 getDefaultPops 的调用是有区别的，getDefaultPops 是对于组件类来说只调用一次，后续该类的应用都不会被调用，而 getInitialState 是对于每个组件实例来讲都会调用，并且只调一次。
  5. 该方法在首次渲染之前调用，也是再 render 方法调用之前修改 state 的最后一次机会。
  6. render 该方法会创建一个虚拟DOM，用来表示组件的输出。对于一个组件来讲，render方法是唯一一个必需的方法。render方法需要满足下面几点：
     1. 只能通过 this.props 和 this.state 访问数据（不能修改）
     2. 可以返回 null,false 或者任何React组件
     3. 只能出现一个顶级组件，不能返回一组元素
     4. 不能改变组件的状态
     5. 不能修改DOM的输出
  7. componentDidMount 该方法被调用时，已经渲染出真实的 DOM，可以再该方法中通过 this.getDOMNode() 访问到真实的 DOM(推荐使用 ReactDOM.findDOMNode())。

**存在期**

1. 以下方法被调用
   1. **componentWillReceiveProps**
   2. shouldComponentUpdate
   3. componentWillUpdate
   4. render
   5. componentDidUpdate
2. componentWillReceiveProps 组件的 props 属性可以通过父组件来更改，这时，componentWillReceiveProps 将来被调用。可以在这个方法里更新 state,以触发 render 方法重新渲染组件。
3. shouldComponentUpdate 如果你确定组件的 props 或者 state 的改变不需要重新渲染，可以通过在这个方法里通过返回 false 来阻止组件的重新渲染，返回 `false 则不会执行 render 以及后面的 componentWillUpdate，componentDidUpdate 方法。
4. componentWillUpdate 这个方法和 componentWillMount 类似，在组件接收到了新的 props 或者 state 即将进行重新渲染前，componentWillUpdate(object nextProps, object nextState) 会被调用，注意不要在此方法里再去更新 props 或者 state。
5. 这个方法和 componentDidMount 类似，在组件重新被渲染之后，componentDidUpdate(object prevProps, object prevState) 会被调用。可以在这里访问并修改 DOM。

**销毁时**

1. 以下方法被调用
   1. componentWillUnmount
2. componentWillUnmount 每当React使用完一个组件，这个组件必须从 DOM 中卸载后被销毁，此时 componentWillUnmout 会被执行，完成所有的清理和销毁工作，在 componentDidMount 中添加的任务都需要再该方法中撤销，如创建的定时器或事件监听器。
3. 组件的声明周期
   1. mounting阶段： 出生阶段，组件挂载阶段，只能被创建一次
   2. updating阶段：更新阶段
   3. unmounting阶段：卸载，死亡阶段

**link**

* 1. 最后会被渲染成a链接
  2. to的属性：跳到哪里
     1. 接收字符串： 一个URL地址
  3. replace： bool
     1. true: 跳转视图的时候，替换掉浏览器history stack里面当前的entry
     2. false 跳转视图的时候，往history stack里面push一个entry。

**route**

* 1. 接收一个组件变量，会自动往组件传入3个props，history,location,match
  2. 可以接收一个回调函数，会往回调函数传入一个对象作为参数，这个对象里面有： history,locaiton,match,回调函数需要你返回jsx结构
  3. render接收一个回调函数，和component的第二点一样children和render一样，不过，它无论如何都会匹配到。

1. history中的location是实时更新的。

**NavLink**

activeStyle,activeClassName, exact，其他和link一样

**Redirect**

重定向

**Switch**

一旦匹配到一个，就不再匹配。

**withRouter**

一个高阶组件（能够返回一个组件），接收一个组件（Aac）作为参数，导出一个组件。这时Aac组件的props，就有了那三个属性。

react 组件声明的两种方式

使用类声明，可以有内部状态，有生命周期

使用函数声明，没有内部状态，没有声明周期（不用this.state时使用）

**redux**

<https://div.io/topic/1309>

store

用来维护应用的状态

\*整个应用只有一个store

getState，得到整个应用的状态

dispatch 发布action

subscribe 监听action的发起

action发布之后执行回调函数

action

是一个对象，必须有一个type属性（相当于action的名字）

如果要改变数据（改变应用的状态），应该发起一个action

state

整个应用的状态

通过store.getState()拿到

\*整个应用只有一个state

reducer

一个函数返回整个应用的state。

决定了整个应用的状态。

接收两个参数: state和action

根据这两个参数 计算出应用的下一个state

createStore

用来创建store

**工作原理：**

首先得到一个store

如果想改变数据，发起一个action，store.dispatch(action)

发起action之后，reducer会计算出应用的下一个状态

监听器，在action发起之后，reducer计算完之后执行

如果想拿state,使用store.getState()

applyMiddleware 中间件

redux-thunk

激活这个中间件后，dispatch可以接收更多的数据类型了。

action creators

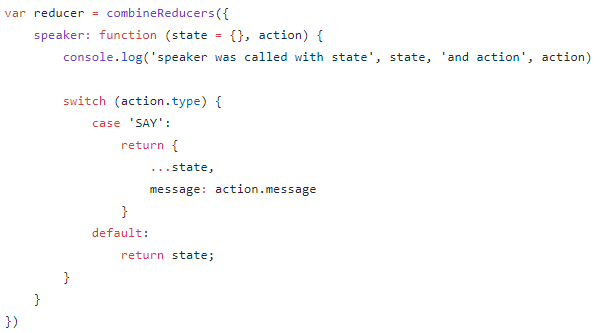
它就是返回action的一个函数，返回一个

但是如果有中间件参与，它可以返回其他数据类型，比如redux-thunk，就可以返回一个函数

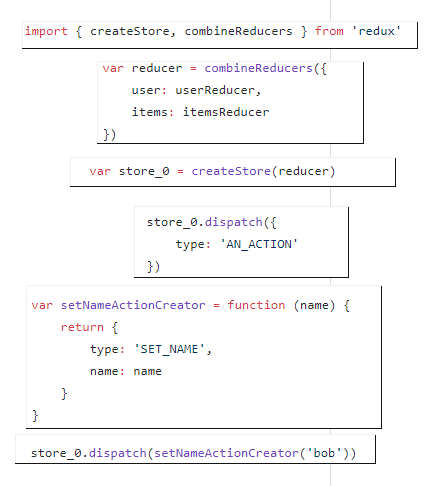
1. 快速新建react项目
2. npm i -g create-react-app yarn
3. create-react-app reactX
4. 在create-react-app中默认开启这个语法<https://doc.react-china.org/docs/handling-events.html>

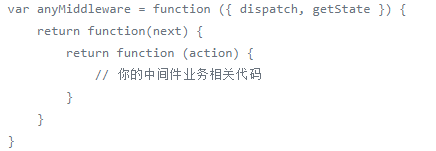
## Redux

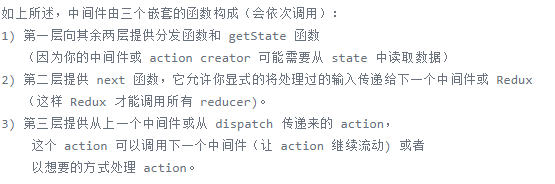
1. reducer定义和combineReducers使用



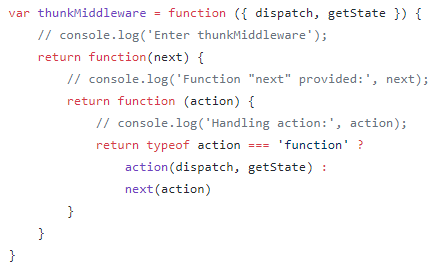
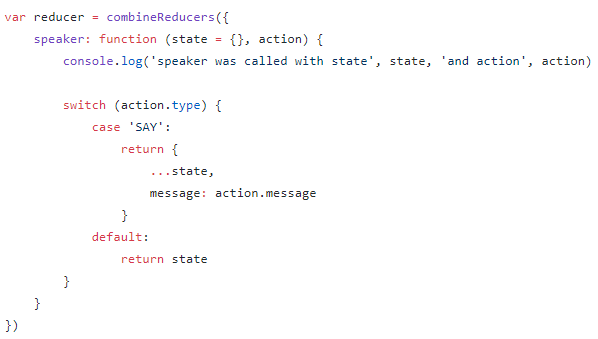
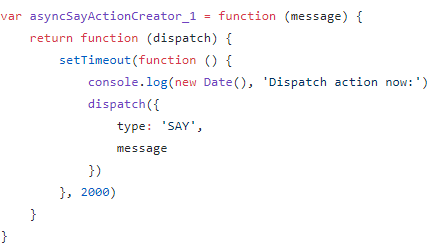
1. ActionCreator -> Action -> dispatcher -> reducer

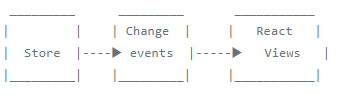


1. 中间件业务代码



1. action ---> dispatcher ---> middleware 1 ---> middleware 2 ---> reducers

1. 监视 Redux store 更新
2. 
3. Action Creator => action => store.dispatch(action) => reducer(state, action) => 原 state state = nextState