## React插槽

组建中写入内容，这些内容可以被识别和控制。React需要自己开发支持插槽功能。

原理：

组件中写入的HTML，可以传入到props中。

组件中的HTML内容直接全部插入

|  |
| --- |
| class ParentCom extends React.Component{        render(){          console.log(this.props)          return (              <div>                  <h1>组件插槽</h1>                  {this.props.children}              </div>          )      }  } |

组件中根据HTML内容的不同，插入的位置不同。

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  class ParentCom extends React.Component{        render(){          console.log(this.props)          return (              <div>                  <h1>组件插槽</h1>                  {this.props.children}                  <ChildCom>                      <h1 data-position="header">这是放置到头部的内容</h1>                      <h1 data-position="main">这是放置到主要的内容</h1>                      <h1 data-position="footer">这是放置到尾部的内容</h1>                  </ChildCom>              </div>          )      }  }  class ChildCom extends React.Component{      render(){          let headerCom,mainCom,footerCom;          this.props.children.forEach((item,index)=>{             if(item.props['data-position']==='header'){              headerCom = item             }else if(item.props['data-position']==='main'){                 mainCom = item             }else{                 footerCom = item             }          })          return (              <div>                  <div className="header">                      {headerCom}                  </div>                  <div className="main">                      {mainCom}                  </div>                  <div className="footer">                      {footerCom}                  </div>              </div>          )      }  }  class RootCom extends React.Component{      constructor(props){          super(props)          //console.log(props)          this.state = {              arr:[1,2,3]          }      }      render(){          return (              <ParentCom>                  <h2 data-name="a" data-index={this.state.arr[0]}>子组件1</h2>                  <h2 data-name="b" data-index={this.state.arr[1]}>子组件2</h2>                  <h2 data-name="c" data-index={this.state.arr[2]}>子组件3</h2>              </ParentCom>          )      }  }  ReactDOM.render(      <RootCom></RootCom>      ,      document.querySelector("#root")  ) |

## React 路由

根据不同的路径，显示不同的组件（内容）；React使用的库react-router-dom;

安装

|  |
| --- |
| Cnpm install react-router-dom --save |

ReactRouter三大组件：

Router：所有路由组件的根组件（底层组件），包裹路由规则的最外层容器。

属性：basename->设置跟此路由根路径，router可以在1个组件中写多个。

Route:路由规则匹配组件，显示当前规则对应的组件

Link:路由跳转的组件

注意：如果要精确匹配，那么可以在route上设置exact属性。

Router使用案例

|  |
| --- |
| import React from 'react';  //hash模式  //import {HashRouter as Router,Link,Route} from 'react-router-dom'  //history模式/后端匹配使用  import {BrowserRouter as Router,Link,Route} from 'react-router-dom'  function Home(){      return (          <div>              <h1>admini首页</h1>          </div>      )  }  function Me(){      return (          <div>              <h1>admin个人中心</h1>          </div>      )  }  function Product(){      return (          <div>              <h1>admin产品页面</h1>          </div>      )  }  class App extends React.Component{        render(){          return (              <div id="app">                  {/\* <div>所有页面普通内容</div> \*/}                  <Router>                      <Route path="/" exact component={()=>(<div>首页</div>)}></Route>                      <Route path="/me" component={()=>(<div>me</div>)}></Route>                      <Route path="/product" component={()=>(<div>product</div>)}></Route>                  </Router>                  <Router>                      {/\* <div className="nav">                          <Link to="/">Home</Link>                          <Link to="/product">Product</Link>                          <Link to="/me">个人中心</Link>                      </div> \*/}                      <Route path="/admin/" exact component={Home}></Route>                      <Route path="/admin/product" component={Product}></Route>                      <Route path="/admin/me" exact component={Me}></Route>                  </Router>              </div>          )      }  }  export default App |

Link组件可以设置to属性来进行页面的跳转，to属性可以直接写路径的字符串，也可以通过1个对象，进行路径的设置，如

|  |
| --- |
| render(){          let meObj = {              pathname:"/me",//跳转的路径              search:"?username=admin",//get请求参数              hash:"#abc",//设置的HASH值              state:{msg:'helloworld'}//传入组件的数据          };          return (              <div id="app">                    <Router>                      <div className="nav">                          <Link to="/">Home</Link>                          <Link to="/product">Product</Link>                          <Link to={ meObj }>个人中心</Link>                      </div>                      <Route path="/" exact component={Home}></Route>                      <Route path="/product" component={Product}></Route>                      <Route path="/me" exact component={Me}></Route>                  </Router>              </div>          )      } |

Link的replace属性：点击链接后，将新地址替换成历史访问记录的原地址。

动态路由实现：

|  |
| --- |
| import React from 'react';  //hash模式  //import {HashRouter as Router,Link,Route} from 'react-router-dom'  //history模式/后端匹配使用  import {BrowserRouter as Router,Link,Route} from 'react-router-dom'  function Home(){      return (          <div>              <h1>admini首页</h1>          </div>      )  }  function Me(props){      console.log(props)      return (          <div>              <h1>admin个人中心</h1>          </div>      )  }  function Product(){      return (          <div>              <h1>admin产品页面</h1>          </div>      )  }  function News(props){      console.log(props)      return (          <div>              新闻页，新闻id：{props.match.params.id}          </div>      )  }  class App extends React.Component{        render(){          let meObj = {              pathname:"/me",//跳转的路径              search:"?username=admin",//get请求参数              hash:"#abc",//设置的HASH值              state:{msg:'helloworld'}//传入组件的数据          };          return (              <div id="app">                    <Router>                      <div className="nav">                          <Link to="/">Home</Link>                          <Link to="/product">Product</Link>                          <Link to={ meObj }   replace>个人中心</Link>                          <Link to="/news/4568789">新闻页</Link>                      </div>                      <Route path="/" exact component={Home}></Route>                      <Route path="/product" component={Product}></Route>                      <Route path="/me" exact component={Me}></Route>                      <Route path="/news/:id" component={News}></Route>                  </Router>              </div>          )      }  }  export default App |

## 重定向组件

如果访问某个组件时，如果有重定向组件，那么就会修改页面路径，使得页面内容显示为所定向路径的内容

用例：

|  |
| --- |
| function LoginInfo(props){      //props.loginState = 'success';      //props.loginState = "fail"      console.log(props)      if(props.location.state.loginState === 'success'){          return <Redirect to="/admin"></Redirect>      }else{          return <Redirect to="/login"></Redirect>      }  } |

## Switch组件

让switch组件内容的route只匹配1个，只要匹配到了，剩余的路由规则将不再匹配

|  |
| --- |
| class App extends React.Component{      render(){          return (              <div>                  <Router>                      <Switch>                          <Route path="/" exact  component={()=>(<h1>首页</h1>)}></Route>                          <Route path="/form" exact  component={FormCom}></Route>                          <Route path="/login" exact  component={()=>(<h1>登录页</h1>)}></Route>                          <Route path="/logininfo" exact component={LoginInfo}></Route>                          <Route path="/admin" exact component={()=>(<h1>admin页,登录成功</h1>)}></Route>                          <Route path="/abc" exact component={()=>(<h1>abc1页,登录成功</h1>)}></Route>                          <Route path="/abc" exact component={()=>(<h1>abc2页,登录成功</h1>)}></Route>                      </Switch>                  </Router>              </div>          )      }  } |

## Redux

解决React数据管理（状态管理），用于中大型，数据比较庞大，组件之间数据交互多的情况下使用。如果你不知道是否需要使用Redux，那么你就不需要用它！

\* 解决组件的数据通信。

\* 解决数据和交互较多的应用

Redux只是一种状态管理的解决方案！

Store:数据仓库，保存数据的地方。

State:state是1个对象，数据仓库里的所有数据都放到1个state里。

Action:1个动作，触发数据改变的方法。

Dispatch:将动作触发成方法

Reducer:是1个函数，通过获取动作，改变数据，生成1个新state。从而改变页面

安装

|  |
| --- |
| Cnpm install redux --save |

初始化数据

|  |
| --- |
| //创建仓库  const store = createStore(reducer)  //用于通过动作，创建新的state  //reduce有2个作用，1初始化数据，第二个就是通过获取动作，改变数据  const reducer = function(state={num:0},action){      console.log(action)      switch(action.type){          case "add":              state.num++;              break;          case 'decrement':              state.num--;              break;          default:              break;      }      return {...state}//相当于对象的COPY  } |

获取数据

|  |
| --- |
| let state = store.getState() |

修改数据（通过动作修改数据）

|  |
| --- |
| //通过仓库的方法dispatch进行修改数据  store.dispatch({type:"add",content:{id:1,msg:"helloworld"}}) |

修改视图（监听数据的变化，重新渲染内容）

|  |
| --- |
| store.subscribe(()=>{      ReactDOM.render(<Counter></Counter>,document.querySelector("#root"))  }) |

## React-redux

安装

|  |
| --- |
| cnpm install react-redux --save |

概念：

Provider组件：自动的将store里的state和组件进行关联。

MapStatetoProps：这个函数用于将store的state映射到组件的里props

mapdispatchToProps:将store中的dispatch映射到组件的props里，实现了方法的共享。

Connect方法：将组件和数据（方法）进行连接

使用：

初始化数据，实例化store

|  |
| --- |
| function reducer(state={num:0},action){      switch(action.type){          case "add":              state.num++;              break;          default:              break;      }      return {...state}  }  const store = createStore(reducer) |

数据的获取，数据的修改

要state映射到到组件的props中，将修改数据的方法映射到组件的props中

|  |
| --- |
|  |