# React 笔记 //我要学全栈

绑定元素，渲染组件：

```
ReactDOM.render(
    <h1>Hello, world!</h1>,
    document.getElementById('root')
 );
```

## JSX

JSX 允许在模板中插入数组，数组会自动展开所有成员：

无法使用if else,只能用三元

class需写成className

```
var myStyle = {
    fontSize: 100,
    color: '#FF0000'
};
ReactDOM.render(
    <h1 style = {myStyle}>aaa</h1>,
    document.getElementById('example')
);
```

## 组件

可以使用类或者函数的方式创建组件

```
function HelloMessage(props) {
    return <h1>Hello World!</h1>;
}
```

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello World!</h1>;
  }
}
```

> *原生 HTML 元素名以小写字母开头，而自定义的 React 类名以大写字母开头，比如 HelloMessage 不能写成 helloMessage。除此之外还需要注意组件类只能包含一个顶层标签，否则也会报错。*

**传递参数**：使用props来传递参数

```
function HelloMessage(props) {
    return <h1>Hello {props.name}!</h1>;
}

const element = <HelloMessage name="pmh"/>;

ReactDOM.render(
    element,
    document.getElementById('example')
);
```

### state:

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }
}
```

### 生命周期:

**挂载阶段**

1.constructor:构造方法，接受一个props属性对象

2.componentWillMount:组件被挂载到DOM前，只会调用一次。

3.render：唯一必要方法，根据组件的props和state返回一个react元素

4.componentDidMount：挂载之后调用，且只会调用一次，其中使用setState会使组件重新渲染。

**更新阶段**

1.componentWilReceiveProps(nextProps)

2.shoudComponentUpdate(nextProps,netState)

3.componentWillUpdate:在render前调用，作为组件更新前执行某些过的地方

4.componentDidUpdate(prevProps,prevState)：组件更新后调用，可以作为更新后调用DOM的地方

**卸载阶段:**

1.componnentWillUnMount:组件即将销毁，可以解绑某些事件

### 绑定事件

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // 这边绑定是必要的，这样 `this` 才能在回调函数中使用
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(prevState => ({
```

```
      isToggleOn: !prevState.isToggleOn
    }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}

ReactDOM.render(
  <Toggle />,
  document.getElementById('example')
);
```

```
<button onClick={(e) => this.handleClick(e)}>
```

### Ref

可以给控件绑定一个ref,这样就可以拿到该元素的实例

```
class MyComponent extends React.Component {
  handleClick() {
    // 使用原生的 DOM API 获取焦点
    this.refs.myInput.focus();
  }
  render() {
    //  当组件插入到 DOM 后，ref 属性添加一个组件的引用于到 this.refs
    return (
      <div>
        <input type="text" ref="myInput" />
        <input
          type="button"
          value="点我输入框获取焦点"
          onClick={this.handleClick.bind(this)}
        />
      </div>
    );
  }
}
```

## React Router

传递props

```
<Route exact path='/' component={Home}/> 不可以使用props
<Route exact path='/' render={() => <Home items={this.state.items} />} />
通过 render 属性来指定渲染函数，render 属性值是一个函数，当路由匹配的时候指定该函数进行渲染
```

NavLink

Switch

Redirect

### 路由传参

hooks:

- useHistory
- useLocation
- useParams
- useRouteMatch

## Redux

保存状态的容器。所有数据保存一个称为store的容器中。只能有一个,store本质上有一个状态树，保存了所有对象的状态。

只能使用单一数据源

只能使用纯函数来修改

可以通过本地或远程组件更改状态，需要分发一个action，将action代理给相关的reducer.

可以使用createStore()来引入redux,创建reducer的方法

- `type` - 一个简单的字符串常量，例如ADD, UPDATE, DELETE等。
- `payload` - 用于更新状态的数据。

创建redux存储区，只能使用reducer作为参数来构造。 存储在Redux存储区中的数据可以被直接访问，但只能通过提供的reducer进行更新。

可以使用store.getState()打印出当前的状态。

store.dispatch()来触发事件。

使用Provider类将React应用包装在Redux容器中。

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';

const App = <h1>Redux Shopping Cart</h1>;

ReactDOM.render(
  <Provider store={store}>
    { App }
  </Provider> ,
  document.getElementById('root')
);
```

useDispatch() 获取dispatch

useStore 获取store

useSelector 获取state

redux-thunk

- 参数是对象，直接调用 reducer 修改我们的 state
- 参数是函数，调用该函数，并且把 dispatch 和 getState 传递我们的函数，可以在函数中，进行异步操作