React 小知识点

声明式编程&&命令式编程

- 声明式编程

```ini 复制代码
const doubleWithDec = numbers.map(number => number * 2);
```

- 命令式编程

```ini 复制代码
const doubleWithImp = [];
    for(let i=0; i<numbers.length; i++) {
    const numberdouble = numbers[i] * 2;
    doubleWithImp.push(numberdouble)
}
```

> 声明式编程的编写方式描述了应该做什么，而命令式编程描述了如何做。

函数式编程

函数式编程是声明式编程的一部分。 函数是数据 ，你可以像保存变量一样在应用程序中保存、检索和传递这些函数。其具有以下特性：

- 不可变性(Immutability)
- 纯函数(Pure Functions)
- 数据转换(Data Transformations)
- 高阶函数 (Higher-Order Functions)
- 递归
- 组合

React

React 是js的一个UI库，能够高效、快速的构建用户界面。使用虚拟dom操作dom,遵循数据从高阶组件到低阶组件单向流动

虚拟dom

浏览器遵循HTML指令来构造文档对象模型(DOM)。当浏览器加载HTML并呈现用户界面时，HTML文档中的所有元素都变成DOM元素。eg:

```css
<div>
    <div>
        <h1>This is heading</h1>
        <p>this is paragraph</p>
        <div>
            <p>This is just a paragraon</p>
        </div>
    </div>
</div>
```

每当DOM发生更改时，浏览器都需要重新计算CSS、进行布局并重新绘制web页面,这是个非常消耗计算的过程.而虚拟dom运用两个虚拟dom树，通过比较两个虚拟DOM 差异，并将这些变化更新到实际DOM，从而大大减少计算

JSX

- JSX是一种JavaScript的语法扩展（eXtension），也在很多地方称之为JavaScript XML，因为看起来就是一段XML语法；
- 它用于描述我们的UI界面，并且其完成可以和JavaScript融合在一起使用
- 它生成React元素，这些元素将在DOM中呈现。React建议在组件使用JSX

```javascript
import React from 'react';
export const Header = () => {
    const heading = 'TODO App'
    return(
        <div style={{backgroundColor:'orange'}}>
            <h1>{heading}</h1>
        </div>
    )
}
```

函数组件

函数组件(无状态组件)是一个纯函数，它可接受接受参数，并返回react元素。eg: 上面的 `Header` 组件

类组件

由于hooks的出现，类组件已经慢慢被淘汰的。这里不讨论。

React如何引用css

- 外部样式表

```javascript 复制代码
import React from 'react';
import './App.css';//外部样式表
import { Header } from './header/header';

function App() {
  return (
    <div className="App">
      <Header />
    </div>
  );
}

export default App;
```

- 内联样式

```javascript 复制代码
import React from 'react';
export const Header = () => {
    const heading = 'TODO App'
    return(
        //内联样式
        <div style={{backgroundColor:'orange'}}>
            <h1>{heading}</h1>
        </div>
    )
}
```

- 定义样式对象并使用它 styled

```javascript 复制代码
//可以在组件中定义一个`style`对象并使用它
import React from 'react';

const footerStyle = {
    width: '100%',
    backgroundColor: 'green',
    padding: '50px',
    font: '30px',
    color: 'white',
    fontWeight: 'bold'
}

export const Footer = () => {
    return(
        <div style={footerStyle}>
            All Rights Reserved 2019
        </div>
    )
}
```

React-Redux

- Redux 是 React的一个状态管理库，它基于flux。
- Redux简化了React中的单向数据流。 Redux将状态管理完全从React中抽象出来 原理
- redux是react全家桶的一员，它试图为 React 应用提供「可预测化的状态管理」机制。
- Redux是将整个应用状态存储到到一个地方，称为store
- 里面保存一棵状态树(state tree)
- 组件可以派发(dispatch)行为(action)给reduer，reduer处理好数据后返回新的state给store
- store不是直接通知其它组件
- 其它组件可以通过订阅store中的状态(state)来刷新自己的视图

React状态管理

React组件列表渲染、条件渲染与DOM的 Diffing 算法

key 的使用

有的人发现了写列表渲染的时候，我都会给每个标签加上一个 `key` 属性，这是为什么呢？ 其实 `key` 的作用是给当前的标签添加一个**唯一的标识**，用于给**React**进行 **diffing算法**计算的时候使用

```ini 复制代码
export const Content = () => {
    const listData = [
        { name:'lee', age: 23, id: 277 },
        { name:'fun', age: 24, id: 898 },
        { name:'kao', age: 25, id: 554 },
    ];
    return <div style={footerStyle}>
       {listData.map(item => {
           return <li key={item.id}>{item.name}---{item.age}</li>
       })}

    </div>);
}
```

3. diffing 算法

当状态发生改变时，react会根据【新的状态】生成【新的虚拟DOM】
然后将新旧虚拟DOM进行 diff比较，比较规则如下：

1. 旧虚拟DOM 中**找到了**与 新虚拟DOM 相同的 key
    - 若虚拟DOM中的内容没变，则直接使用之前的真实DOM
    - 若虚拟DOM中的内容变了，则生成新的真实DOM并进行替换
2. 旧虚拟DOM 中**未找到**与 新虚拟DOM 相同的 key，则根据数据创建新的真实DOM，然后渲染到页面

用index作为key可能引发的问题

- 对数据进行 逆序添加、逆序删除 邓破坏顺序的操作 会产生没有必要的真实DOM更新，影响效率
- 对结构中包含输入类的DOM 会产生错误的 DOM更新，同时界面渲染有问题
- 若仅用于展示数据，那用 index作为 key则没有问题

react-router-dom(v6说法)

react-router-dom 是目前最新的说法之前是 react-router 全局路由有常用两种路由模式可选：HashRouter 和 BrowserRouter

- HashRouter：URL中采用的是hash(#)部分去创建路由，类似www.example.com/#/
- BrowserRouter：URL采用真实的URL资源 后续有文章会详细讲HashRouter的原理和实现，这里我们采用BrowserRouter来创建路由

useNavigate

`useNavigate` 是替代原有V5中的 `useHistory` 的新hooks，其用法和 `useHistory` 类似，整体使用起来更轻量

javascript 复制代码

```
//js写法
let navigate = useNavigate();
function handleClick() {
  navigate("/home");
}
//组件写法
function App() {
   return <Navigate to="/home" replace state={state} />;
}
//替代原有的go goBack和goForward
<button onClick={() => navigate(-2)}>
  Go 2 pages back
</button>
<button onClick={() => navigate(-1)}>Go back</button>
<button onClick={() => navigate(1)}>
  Go forward
</button>
<button onClick={() => navigate(2)}>
  Go 2 pages forward
</button>
```