React

第0章 简介

React 起源于 Facebook 的内部项目，因为该公司对市场上所有 JavaScript MV* 框架，都不满意，就决定自己写一套，用来架设 Instagram（照片交友） 的网站。做出来以后，发现这套东西很好用，**就在2013年5月开源了。**

由于 React 的**设计思想极其独特**，属于革命性创新，性能出众，代码逻辑却非常简单。所以，越来越多的人开始关注和使用，认为它可能是将来 Web 开发的主流工具。

清楚两个概念：

- library（库）：小而巧的库，只提供了特定的API；
- Framework（框架）：大而全的是框架；框架提供了一整套的解决方案；

三大前端框架：

> AngularJS 2009年（谷歌）
>
> React.js 2013年5月（Facebook）
>
> Vue.js 2014年2月（尤雨溪 ）
>
> Angular2 2016年9月（谷歌）

Angular.js：出来**较早**的前端框架，学习曲线比较陡，NG1学起来比较麻烦，NG2 ~ NG5开始，进行了一系列的改革，也提供了组件化开发的概念；从NG2开始，也支持使用TS（TypeScript）进行编程；

Vue.js： 最火（关注的人比较多）的一门前端框架，它是中国人开发的，对我我们来说，文档要友好一些；

React.js： 最流行（用的人比较多）的一门框架，因为它的设计很优秀；

中文手册及教程： https://react.docschina.org/

React 谷歌调试工具： https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=zh-CN

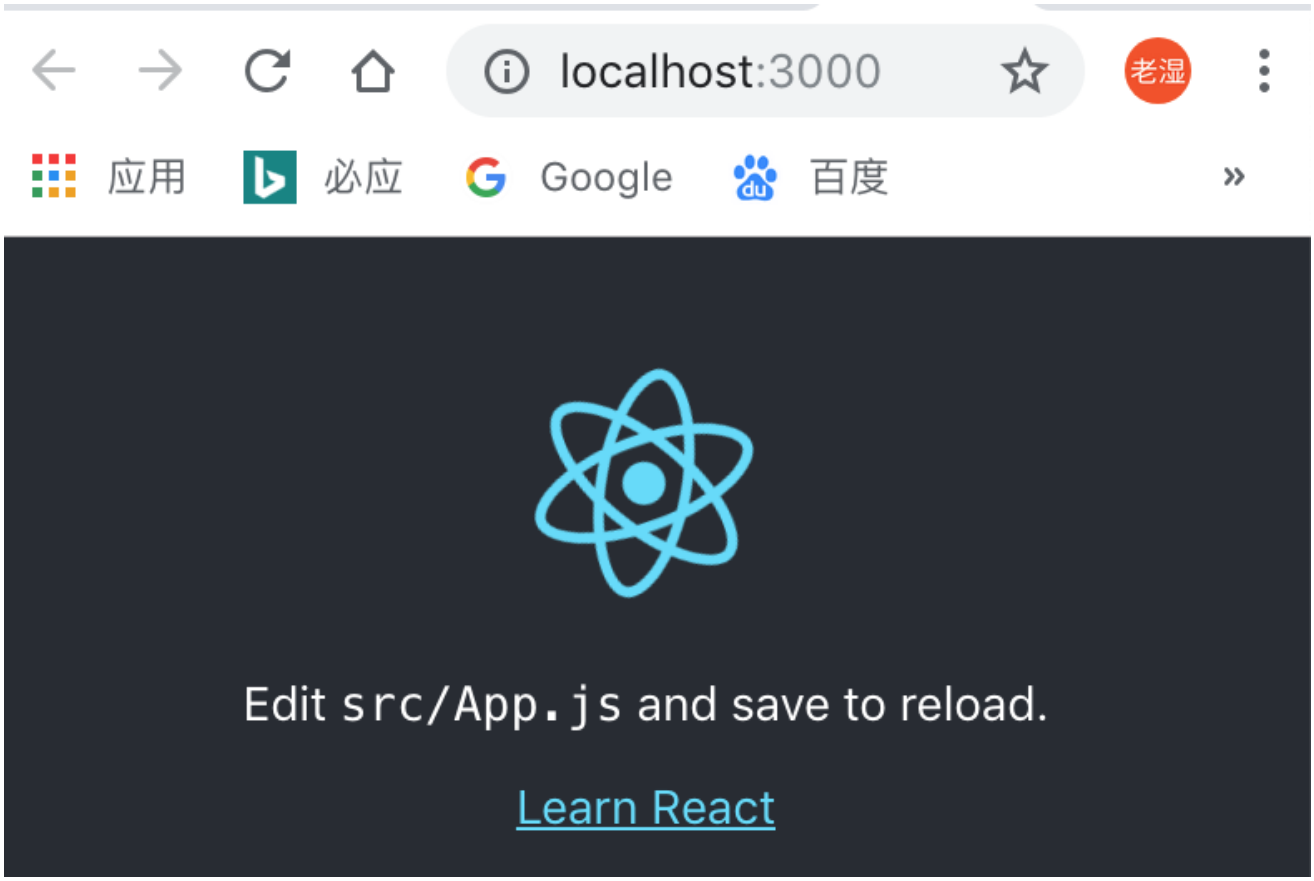
第1章 搭建项目

1：安装react命令行工具（官方脚手架工具）： `npm install create-react-app -g`

2：创建项目工程：`create-react-app myapp`

```
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd myapp
  npm start
```

```
→  myapp git:(master) npm start

> myapp@0.1.0 start /Applications/XAMPP/xamppfiles/
> react-scripts start

Compiled successfully!

You can now view myapp in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://192.168.1.100:3000/

Note that the development build is not optimized.
To create a production build, use npm run build.
```



修改 `/public/index.html`

```
<body>
  <div id="app"></div>
</body>
```

修改 `/src/index.js`

```
import React from 'react';
import ReactDOM from 'react-dom';

// 这是 创建虚拟DOM元素的 API     <h1 title="啊，五环" id="myh1">你比四环多一环</h1>
//  第一个参数:  字符串类型的参数，表示要创建的标签的名称
//  第二个参数: 对象类型的参数,  表示 创建的元素的属性节点
//  第三个参数:  子节点
var myh1 = React.createElement('h1', { title: '啊，五环', id: 'myh1' }, '你比四环多一环')

ReactDOM.render(myh1, document.getElementById('app'));

```

结果:

你比四环多一环

第2章 JSX 语法简介

https://react.docschina.org/docs/introducing-jsx.html

2.1 基本使用

按照上面的写法，每当写一段内容，就要调用 `React.createElement()` 方法；因此react引入了 jsx 语法：

我们来观察一下声明的这个变量：

```
1 const element = <h1>Hello, world!</h1>;
```

这种看起来可能有些奇怪的标签语法既不是字符串也不是 HTML。

它被称为 JSX， 一种 JavaScript 的语法扩展。 在 React 中使用 JSX 来描述用户界面。

JSX 乍看起来可能比较像是模版语言，但事实上它完全是在 JavaScript 内部实现的。

JSX 用来声明 React 当中的元素，我们接下来看看 JSX 的基本使用方法。

> 什么是JSX语法：就是符合 xml 规范的 JS 语法；（语法格式相对来说，要比HTML严谨很多）

修改 `/src/index.js`

```
import React from 'react';
import ReactDOM from 'react-dom';

// JSX 语法
var myh1 = <h1>你比四环多一环</h1>

ReactDOM.render(myh1, document.getElementById('app'));

```

jsx 语法的本质： 并不是直接把 jsx 渲染到页面上，而是 内部先转换成了 createElement 形式，再渲染的；

2.2 在 jsx 中语法中的 js 表达式

在 jsx 语法中，要把 JS代码写到 `{ }` 中，所有标签必须要有结束

渲染数字 、渲染字符串、布尔值实现三元表达式:

```
import React from "react";
import ReactDOM from "react-dom";

var num = 12;
var str = "hellow";
var boo = false;

// JSX 语法
var myh1 = (
  <div>
    {/* 我是注释 */}
    {num}
    <hr />
    {str}
    <hr />
    {boo ? "条件为真" : "条件为假"}
    <hr />
  </div>
);

ReactDOM.render(myh1, document.getElementById("app"));

```

为属性绑定值:

```
const btnTitle = '啊,五环~~~'
……

{/* 如果在 元素的属性节点中,想使用 { } 渲染动态的属性, 则 { } 外面,一定不要加 引号进行包裹; */}
<button title={btnTitle} >按钮</button>
{/* 注意: class 是 ES6 中的JS关键字; 因此为了防止歧义, 应该把 class 替换为 className */}
<button title={btnTitle} className="mybtn">
```

渲染jsx元素:

```
const myh3 = <h3>这是h3标签</h3>
……
<hr />
    {myh3}
<hr />
```

渲染jsx元素数组:

```
// 总结:如果想要 把一些虚拟DOM的数组集合,渲染到页面中,直接 把 数组 使用 { } 来渲染即可;
const jsxArr = [<p key="1">1</p>, <p key="2">2</p>, <p key="3">3</p>]
……
<hr />
{jsxArr}
```

循环数组元素:

```
const nameList = ['张三', '李斯', '许三多']
```

方案1:

```
// 方案1: 在 jsx 外面,进行 循环,创建 jsx 元素的数组;
const newList = nameList.map((item, i) => {
  return <h6 key={i}>{item}</h6>
})
……
{newList}
```

方案2:

```
{/* 方案2: 在 jsx 里面,进行循环,创建 jsx 元素的数组; */}
<hr />
{nameList.map((item, i) => (
    <h6 key={i}>{item}</h6>
))}
```

总结:

在 JSX 语法中，要把 JS代码写到 `{ }` 中

在JSX创建DOM的时候，所有的节点，必须有唯一的根元素进行包裹；

在 JSX 语法中，标签必须 成对出现，如果是单标签，则必须自闭和！

为 jsx 中的元素添加class类名,需要使用 `className` 来替代 `class`

使用 `{ /* 这是注释 */ }` 在 jsx 中 写注释

第3章 组件 & Props

https://react.docschina.org/docs/components-and-props.html

3.1 构造函数定义组件

```
import React from "react";
import ReactDOM from "react-dom";

// 构造函数定义组件
// 这是 React 中创建组件最Easy的方式
// 在 使用 普通 function 构造函数, 创建 React 组件的时候,必须 要 return 一个 合法的 jsx 结构;
// 如果 什么东西都不想 return, 则可以 return null
// 注意:React 中创建的组件, 组件名称,必须是 大写;
function Hello () {
  // return null
  // 组件必须有返回值
    return <div>Hello 组件</div>
}

// 直接将组件的名字当做标签使用即可
ReactDOM.render(<Hello/>, document.getElementById("app"));
```

3.2 为组件传递数据

```
// 构造函数定义组件
// 构造函数会接受一个对象作为参数，对象中有组件传入的属性及值
function Hello(props) {
  console.log(props);
  return <div>Hello 组件</div>;
}

// 组件使用时的属性
ReactDOM.render(<Hello name="zs" age={20} />, document.getElementById("app"));

```

```jsx
// 构造函数定义组件
// 构造函数会接受一个对象作为参数，对象中有组件传入的属性及值
function Hello(props) {
  // 注意：组件中接收到外界传递过来的 props 数据，都是 只读的，无法 重新赋值!!!
  console.log(props);
  return <div>
    <h1>{props.name}</h1>
    <h1>{props.age}</h1>
  </div>;
}

var data = {
  name:'lisi',
  age:20
}
var my = <Hello name={data.name} age={data.age} />;

// 如果有多个数据需要传入组件,可以使用扩展运算符
var my = <Hello {...data} />;

// 组件使用时的属性
ReactDOM.render(my, document.getElementById("app"));

```

> 对象的扩展运算符（`...`）用于取出参数对象的所有可遍历属性，拷贝到当前对象之中。
>
> http://es6.ruanyifeng.com/#docs/object#%E6%89%A9%E5%B1%95%E8%BF%90%E7%AE%97%E7%AC%A6

3.3 将组件封装到单独的文件中

新建文件 `/src/hello.js`

```jsx
// 引入 react
import React from 'react';

// 导出组件
export default function Hello(props) {
  console.log(props);
  return <div>
    <h1>{props.name}</h1>
    <h1>{props.age}</h1>
  </div>;
}
```

`/src/index.js`

```jsx
import React from "react";
import ReactDOM from "react-dom";

```

```
// ES6 模块化语法导入
import Hello from "./hello"

var data = {
  name:'lisi',
  age:20,
  gender:'男'
}
var my = <Hello {...data} />;

ReactDOM.render(my, document.getElementById("app"));
```

3.4 JS 中的类

JS 中的 Class 语法及 Class 语法的继承：

http://es6.ruanyifeng.com/#docs/class

http://es6.ruanyifeng.com/#docs/class-extends

3.4.1 JS 中的类

在JS中，想要获取一个对象，有多种方式：

`var o1 = {}` `var o2 = new Object()`

自定义构造函数方式

```
function Point(x, y) {
  this.x = x;
  this.y = y;
  this.toString = function () {
    return this.x + ', ' + this.y;
  };
}

var p = new Point(1, 2);
console.log(p.toString());
```

但是上面这种使用构造函数获取对象的写法跟传统的面向对象语言（比如 C++ 和 Java）差异很大，很容易让新学习这门语言的程序员感到困惑。

ES6 提供了更接近传统语言的写法，引入了 Class（类）这个概念，作为对象的模板。

通过`class`关键字，可以定义类。

基本上，ES6 的`class`可以看作只是一个**语法糖**，它的绝大部分功能，ES5 都可以做到，新的`class`写法只是让对象原型的写法更加清晰、更像面向对象编程的语法而已。

上面的代码用 ES6 的`class`改写，就是下面这样。

```
//定义类
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  toString() {
    return  this.x + ', ' + this.y ;
  }
}
var p = new Point(1, 2);
console.log(p.toString());
```

上面代码定义了一个“类”，可以看到里面有一个`constructor`方法，这就是**构造方法**(后面还会讲到)，而`this`关键字则代表实例对象。也就是说，ES5 的构造函数`Point`，对应 ES6 的类`Point`。

`Point`类除了构造方法，还定义了一个`toString`方法。注意，定义“类”的方法的时候，前面不需要加上`function`这个关键字，直接把函数定义放进去了就可以了。另外，方法之间不需要逗号分隔，加了会报错。

ES6 的类，完全可以看作构造函数的另一种写法。

使用的时候，也是直接对类使用`new`命令，跟构造函数的用法完全一致。

类同样也有`prototype`属性，而属性的值依然是实例对象的原型对象；

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  toString() {
    return  this.x + ', ' + this.y ;
  }
}
Point.prototype.toValue = function(){
    console.log('123');
}
var p = new Point(1, 2);
p.toValue();
console.log(p.toString());
```

3.4.2 constructor 方法

`constructor`方法是类的默认方法，通过`new`命令生成对象实例时，自动调用该方法。一个类必须有`constructor`方法，如果没有显式定义，一个空的`constructor`方法会被默认添加。

```
class Point {
}

// 等同于
class Point {
  constructor() {}
}
```

类必须使用`new`进行实例化调用，否则会报错。

而类一旦实例化，`constructor()`方法就会被执行，就像 人一出生就会哭一样；

`constructor`方法默认返回实例对象（即`this`）；

但是返回值完全可以指定返回另外一个对象；

```
var o1 = {
    f1:function(){
        console.log('f1');
    }
}
class Point{
    constructor (){
        return o1;
    }
    f2(){
        console.log('f2');
    }
}

var p = new Point();
p.f1(); // f1
p.f2(); //Uncaught TypeError: p.f2 is not a function
```

> `constructor`方法默认返回值尽量不要修改，一旦修改，我们获取的对象将脱离其原型；

3.4.3 变量提升

我们知道，在JS中，不管是全局变量还是局部变量都存在变量提升的特性，函数也有提升的特性，也可以先调用后声明，构造函数也一样；如下面的代码，完全没问题：

```
var p = new Point();
p.f2();

function Point(){
    this.f2 = function(){
        console.log('f2');
    }
}
```

但是，需要注意： **类不存在变量提升（hoist），这一点与 ES5 完全不同。**

```
new Man();
class Man{}

// Man is not defined
```

> 注意，class只是在原有面向对象的基础上新加的关键字而已，本质上依然没有改变JS依赖原型的面向对象方式;

3.4.4 类的继承

Class 可以通过`extends`关键字实现继承，这比 ES5 的通过修改原型链实现继承，要清晰和方便很多。

```
class Man {
  constructor(){
    this.names = "lisi";
  }
  getName() {
    console.log(this.names);
  }
}

class Run extends Man {
  getage(){
    console.log(this.names);
  }
}

var man = new Run();
man.getName();
man.getage();
```

ES5 的继承，实质是先创造子类的实例对象，然后再将父类的方法添加到上面（`Parent.call(this)`）。

```
function Man(){
  this.names = 'lisi';
  this.getn = function(){
    console.log(11);
```

```
  }
}
function Run(){
  // 实现继承
  Man.call(this);
  this.fun = function(){
    console.log(this.names);
  }
}
var r = new Run();
r.getn();
r.fun();
```

ES6 的继承机制完全不同，是先创造父类的实例对象,然后再用子类的构造函数修改。

因此，如果子类中 显式调用构造方法 `constructor(){}` ,必须要在子类构造方法中调用 `super()` 方法。如果不调用 `super` 方法，子类就得不到 `this` 对象。

```
class Run{
    p(){
        console.log('ppp');
    }
}
class Man extends Run{
    // 显式调用构造方法
    constructor(){}
}
var m = new Man();
m.p();
// Uncaught ReferenceError: Must call super constructor in derived class before accessing 'this' or returning from derived
```

上面代码中，`Man` 继承了父类 `Run`，但是子类并没有先实例化 `Run`，导致新建实例时报错。

```
class Run{
    p(){
        console.log('ppp');
    }
}
class Man extends Run{
    constructor(){
        // 调用父类实例
        super();
    }
}
var m = new Man();
m.p();
```

3.5 类定义组件

ComponentAPI： https://react.docschina.org/docs/react-api.html

```
// 如果要使用 class 定义组件，必须 让自己的组件，继承自 React.Component
class 组件名称 extends React.Component {
    // 在 组件内部，必须有 render 函数,作用：渲染当前组件对应的 虚拟DOM结构
    render(){
        // render 函数中，必须 返回合法的 JSX 虚拟DOM结构
        return <div>这是 class 创建的组件</div>
    }
}
```

修改 `hello.js` 代码：

ES6 模块化： http://es6.ruanyifeng.com/#docs/module#import-%E5%91%BD%E4%BB%A4

```
// import React from "react";
// class Hello extends React.Component {

// 按需导出
import React , { Component } from "react";
class Hello extends Component {
    render() {
        return (
            <div>
                <h1>哈哈哈</h1>
            </div>
        );
    }
}
export default Hello;
```

获取 props 数据

```
import React , { Component } from "react";
class Hello extends Component {
    render() {
        return (
            <div>
                {/* 直接使用this.props获取数据,props数据依然为只读的 */}
                <h1>{this.props.name}</h1>
                <h1>{this.props.age}</h1>
            </div>
        );
    }
}
```

3.5.1 两种创建组件方式的对比

1. 用**构造函数**创建出来的组件：无状态组件；
2. 用**class关键字**创建出来的组件：有状态组件；
3. 什么情况下使用有状态组件？什么情况下使用无状态组件？
    - 如果组件内，不需要有私有的数据，此时，使用构造函数创建无状态组件比较合适；
    - 如果组件内，需要有自己私有的数据，则，使用 class 关键字 创建有状态组件比较合适；

3.5.2 组件的私有的数据

```
import React, { Component } from "react";
class Hello extends Component {
  // 构造函数
  constructor(props){
    super(props);
    // 声明私有数据（类似vue中data的作用）
    this.state = {
      msg:'123'
    }
  }
  render() {
    console.log(this.state) // 打印查看
    return (
      <div>
        {/* 在组件中使用私有数据 */}
        <h1>{this.state.msg}</h1>
      </div>
    );
  }
}
export default Hello;
```

组件中的 `props` 和 `state` 之间的区别

- props 中存储的数据，都是外界传递到组件中的；
- props 中的数据，都是只读的；
- state 中的数据，都是组件内私有的；
- state 中的数据，都是可读可写的；
- props 在 有状态组件 和 无状态组件中，都有；
- state 只有在 有状态组件中才能使用；无状态组件中，没有 state；

第4章 评论列表案例

4.1 评论列表

/src/index.js

```
import React from "react";
import ReactDOM from "react-dom";

// ES6 模块化语法导入
import Cmtlist from "./Cmtlist"

ReactDOM.render(<Cmtlist/>, document.getElementById("app"));
```

/src/Cmtlist.js

```
import React, { Component } from "react";
class Cmtlist extends Component {
  constructor(props){
    super(props);
    // 私有数据
    this.state = {}
  }
  render(){
    return (
      // 组件内容
      <div>
        <h1>评论列表</h1>
      </div>
    );
  }
}
export default Cmtlist
```

/src/Cmtlist.js

```
import React, { Component } from "react";
class Cmtlist extends Component {
  constructor(props) {
    super(props);
    // 私有数据
    this.state = {
      CommentList: [
        { id: 1, user: "张三", content: "哈哈，沙发" },
        { id: 2, user: "李四", content: "哈哈，板凳" },
        { id: 3, user: "王五", content: "哈哈，凉席" },
        { id: 4, user: "赵六", content: "哈哈，砖头" },
        { id: 5, user: "田七", content: "哈哈，楼下山炮" }
      ]
    };
  }
  render() {
    return (
      // 组件内容
      <div>
        <h1>评论列表</h1>
        {this.state.CommentList.map((v)=>{
          return (
            // 循环必须有key属性，否则报错
            <div key={v.id}>
              <h3>评论人：{v.user}</h3>
              <p>评论内容：{v.content}</p>
              <hr></hr>
            </div>
          )
        })}
      </div>
    );
  }
}

export default Cmtlist;
```

4.2 组件拆分

拆分使用独立组件，/src/Cmtlist.js：

```
import React, { Component } from "react";

// 使用独立组件
function Cmtitem(props) {
  return (
    <div>
      <h3>评论人：{props.user}</h3>
```

```
      <p>评论内容：{props.content}</p>
      <hr />
    </div>
  );
}

class Cmtlist extends Component {
  constructor(props) {
    super(props);
    // 私有数据
    this.state = {
      CommentList: [
        { id: 1, user: "张三", content: "哈哈，沙发" },
        { id: 2, user: "李四", content: "哈哈，板凳" },
        { id: 3, user: "王五", content: "哈哈，凉席" },
        { id: 4, user: "赵六", content: "哈哈，砖头" },
        { id: 5, user: "田七", content: "哈哈，楼下山炮" }
      ]
    };
  }
  render() {
    return (
      // 组件内容
      <div>
        <h1>评论列表</h1>
        {this.state.CommentList.map(v => {
          // 循环一个独立的组件并将值使用属性出入
          // 每循环一次，将数据展开当做属性传入
          // 循环的组件必须有key值
          return <Cmtitem {...v} key={v.id}/>
        })}
      </div>
    );
  }
}

export default Cmtlist;

```

4.3 文件拆分

将组件单独封装到文件中：

`/src/index.js`

```
import React from "react";
import ReactDOM from "react-dom";

// ES6 模块化语法导入
import Cmtlist from "./components/Cmtlist"

ReactDOM.render(<Cmtlist/>, document.getElementById("app"));
```

/src/components/Cmtlist.js

```
import React, { Component } from "react";

// 封装为独立的文件中
import Cmtitem from "./CmtItem"

class Cmtlist extends Component {
  constructor(props) {
    super(props);
    // 私有数据
    this.state = {
      CommentList: [
        { id: 1, user: "张三", content: "哈哈，沙发" },
        { id: 2, user: "李四", content: "哈哈，板凳" },
        { id: 3, user: "王五", content: "哈哈，凉席" },
        { id: 4, user: "赵六", content: "哈哈，砖头" },
        { id: 5, user: "田七", content: "哈哈，楼下山炮" }
      ]
    };
  }
  render() {
    return (
      // 组件内容
      <div>
        <h1>评论列表</h1>
        {this.state.CommentList.map(v => {
          // 循环一个独立的组件并将值使用属性出入
          // 每循环一次，将数据展开当做属性传入
          // 循环的组件必须有key值
          return <Cmtitem {...v} key={v.id}/>
        })}
      </div>
    );
  }
}

export default Cmtlist;
```

/src/components/CmtItem.js

```jsx
import React from 'react'
// 使用独立组件
export default function Cmtitem(props) {
  return (
    <div>
      <h3>评论人：{props.user}</h3>
      <p>评论内容：{props.content}</p>
      <hr />
    </div>
  );
}
```

4.4 CSS 样式

4.4.1 基本写法：

`/src/components/Cmtlist.js`

```jsx
render() {
    return (
        // 组件内容
        <div>
            {/* 外层{}表示jsx语法，内层{}表示CSS样式内容 */}
            <h1 style={{ color: "red" }}>评论列表</h1>
            {this.state.CommentList.map(v => {
                return <Cmtitem {...v} key={v.id} />;
            })}
        </div>
    );
}
}
```

4.4.2 封装样式表为对象

```jsx
import Cmtitem from "./CmtItem";
// 封装样式表
var h1_style = { color: "red" , 'font-size':'50px'};

class Cmtlist extends Component {
    ……
    ……

 <div>
    {/* 外层{}表示jsx语法，内层{}表示CSS样式内容 */}
    <h1 style={h1_style}>评论列表</h1>
    {this.state.CommentList.map(v => {
        return <Cmtitem {...v} key={v.id} />;
    })}
```

```
</div>
```

4.4.3 引入外部样式表

新建 `/src/components/style.css` 文件：

```
.clolrs{
  color: aqua;
}
```

```
// 引入外部样式表
import "./style.css"
……
……
<div>
    {/* class 属性使用classNmae替换 */}
    <h1 className="clolrs">评论列表</h1>
    {this.state.CommentList.map(v => {
        return <Cmtitem {...v} key={v.id} />;
    })}
</div>
```

4.4.4 美化评论

`/src/components/style.css` 文件

```
.title{
  font-size: 14px;
  padding: 0 5px;
}
.content{
  font-size: 12px;
  text-indent: 2em;
}

.box{
  border: 1px dashed #ccc;
  margin: 10px 0;
  padding: 0 5px;
  box-shadow: 0 0 5px #ccc;
}
```

`/src/components/Cmtlist.js`

```
import React, { Component } from "react";
import Cmtitem from "./CmtItem";

// 引入外部样式表
import "./style.css"
```

`/src/components/CmtItem.js`

```
import React from 'react'
// 使用独立组件
export default function Cmtitem(props) {
  return (
    <div className="box">
      <h3 className="title">评论人：{props.user}</h3>
      <p className="content">评论内容：{props.content}</p>
    </div>
  );
}
```

4.4.5 引入Bootstrap

安装 `npm i bootstrap`

`/src/index.js` 中加入

```
import React from "react";
import ReactDOM from "react-dom";

// 全局引入 bootstrap ，组件中都可以使用bootstrap提供的样式表
import 'bootstrap/dist/css/bootstrap.css'

import Cmtlist from "./components/Cmtlist"
ReactDOM.render(<Cmtlist/>, document.getElementById("app"));
```

第5章 事件

https://react.docschina.org/docs/handling-events.html

5.1 事件的绑定

- React事件绑定属性的命名采用驼峰式写法，而不是小写。

  `onClick`、`onMouseOver`、`onMouseEnter`、`onChange`

- 如果采用 JSX 的语法你需要传入一个函数作为事件处理函数，而不是一个字符串(DOM元素的写法)

  `<button onClick={ function(){alert(123)} }>按钮</button>`

```jsx
import React from "react";
import ReactDOM from "react-dom";

// 事件绑定
var myEve = <button onClick={function(){alert(1)}}>按钮</button>

ReactDOM.render(myEve, document.getElementById("app"));
```

函数组件中的事件绑定：

```jsx
import React from "react";
import ReactDOM from "react-dom";

function MyEve(){
  return (
    <div>
      {/* 为事件处理程序传入一个外部函数 */}
      <button onClick={ev}>按钮</button>
    </div>
  )
}
// 事件处理程序
var ev = function (){
  alert(123)
}

ReactDOM.render(<MyEve></MyEve>, document.getElementById("app"));
```

类组件中的事件绑定：

```jsx
import React from "react";
import ReactDOM from "react-dom";

class MyEve extends React.Component {
  render() {
    return (
      <div>
        {/* 为事件处理程序传入一个外部函数 */}
        <button onClick={ev}>按钮</button>
        {/* 为事件处理程序传入一个类内部方法 */}
        <button onClick={this.evs}>按钮</button>
      </div>
    );
  }
  // 类内部事件处理程序
  evs(){
    alert('12333');
  }
```

```
}

// 事件处理程序
var ev = function() {
  alert(123);
};

ReactDOM.render(<MyEve />, document.getElementById("app"));
```

> 注意：以上事件绑定中，事件处理程序中的this都是指向 undefined

```
// 类内部事件处理程序
evs(){
    console.log(this); //undefined
    alert('12333');
}
```

事件处理程序函数改为箭头函数：

```
import React from "react";
import ReactDOM from "react-dom";

class MyEve extends React.Component {
  constructor(){
    super();
    this.state = {
      age:330
    }
  }
  render() {
    return (
      <div>
        <h3>{this.state.age}</h3>
        {/* 为事件处理程序传入一个类内部方法 */}
        <button onClick={this.evs}>按钮</button>
      </div>
    );
  }
  // 类内部事件处理程序
  // evs(){
  //   console.log(this); //undefined
  //   alert('12333');
  // }

  // 将事件处理程序改为 箭头函数，
  // 此时函数中的thi会指向react组件
  evs=()=>{
    console.log(this.state.age);
    // alert('12333');
```

```
  }
}

ReactDOM.render(<MyEve />, document.getElementById("app"));
```

但是，我们在事件中修改state中的数据，却是失败的：

```
evs=()=>{
    // Do not mutate state directly. Use setState()
    this.state.age = '3333';
    console.log(this.state.age);
    // alert('12333');
}
```

react 提示我们使用 `setState()` 方法

```
evs=()=>{
    // Do not mutate state directly. Use setState()
    // this.state.age = '3333';
    // 在react中修改state的值，要使用react提供的 setState() 方法
    this.setState({age:'333'})
}
```

5.2 数据绑定

在 Vue 中，默认提供了`v-model`指令，可以很方便的实现 `数据的双向绑定`；

但是，在 React 中，默认只是`单向数据流`，也就是 只能把 state 上的数据绑定到 页面，无法把 页面中数据的变化，自动同步回 state；

如果需要把 页面上数据的变化，保存到 state，则需要程序员手动监听文本框的`onChange`事件，拿到最新的数据，手动调用`this.setState({ })`更改回去；

```
import React from "react";
import ReactDOM from "react-dom";

class MyEve extends React.Component {
  constructor(){
    super();
    this.state = {
      age:330
    }
  }
  render() {
    return (
      <div>
```

```
        <h3>{this.state.age}</h3>
        {/* 传入事件对象，利用事件对象获取更改后的数据 */}
        <input value={this.state.age} onChange={(e)=>this.ch(e)}/>
      </div>
    );
  }

  ch = (e)=>{
    // console.log(e.target.value);
    // 修改数据
    this.setState({age:e.target.value})
  }
}

ReactDOM.render(<MyEve />, document.getElementById("app"));
```

5.3 使用ref获取DOM元素

和 Vue 中差不多，vue 为页面上的元素提供了 `ref` 的属性，如果想要获取 元素引用，则需要使用`this.$refs.引用名称`

在 React 中，也有 `ref`，如果要获取元素的引用`this.refs.引用名称`

```
import React from "react";
import ReactDOM from "react-dom";

class MyEve extends React.Component {
  constructor(){
    super();
    this.state = {
      age:330
    }
  }
  render() {
    return (
      <div>
        <h3>{this.state.age}</h3>
        {/* 为元素添加ref 属性 */}
        <input ref="inp" onChange={this.ch}/>
      </div>
    );
  }

  ch = ()=>{
    //console.log(this);
    // 修改数据
    this.setState({age:this.refs.inp.value})
  }
}

ReactDOM.render(<MyEve />, document.getElementById("app"));
```

第6章 生命周期

6.1 生命周期及钩子函数

生命周期的概念：每个组件的实例，从 创建、到运行、直到销毁，在这个过程中，会出发一些列 事件，这些事件就叫做组件的生命周期函数；

React组件生命周期分为三部分：

- **组件创建阶段**：特点：一辈子只执行一次

> componentWillMount: render： componentDidMount:

- **组件运行阶段**：按需，根据 props 属性 或 state 状态的改变，有选择性的 执行 0 到多次

> componentWillReceiveProps: shouldComponentUpdate: componentWillUpdate: render: componentDidUpdate:

- **组件销毁阶段**：一辈子只执行一次

> componentWillUnmount:

vue中的生命周期图 React Native 中组件的生命周期

defaultProps

> 在组件创建之前，会先初始化默认的props属性，这是全局调用一次，严格地来说，这不是组件的生命周期的一部分。在组件被创建并加载候，首先调用 constructor 构造器中的 this.state = {}，来初始化组件的状态。

React的生命周期函数，总结成表格如下：

生命周期	调用次数	能否使用 setSate()
static defaultProps = {}	1(全局调用一次)	否
constructor	1	否
componentWillMount	1	是
render	>=1	否
componentDidMount	1	是
componentWillReceiveProps	>=0	是
shouldComponentUpdate	>=0	否
componentWillUpdate	>=0	否
componentDidUpdate	>=0	否
componentWillUnmount	1	否

组件生命周期的执行顺序：

1. **Mounting:**
    - constructor()
    - componentWillMount()
    - render()
    - componentDidMount()
2. **Updating:**
    - componentWillReceiveProps(nextProps)
    - shouldComponentUpdate(nextProps, nextState)
    - componentWillUpdate(nextProps, nextState)
    - render()
    - componentDidUpdate(prevProps, prevState)
3. **Unmounting:**
    - componentWillUnmount()

6.2 计数器的小案例 - 了解生命周期函数

shouldComponentUpdate

```
import React from "react";
import ReactDOM from "react-dom";

class MyEve extends React.Component {
  constructor(){
    super();
```

```
    this.state = {
      nu:0
    }
  }
  render(){
    return (
      <div>
        <button onClick={this.adds}>+1</button>
        <h4>{this.state.nu}</h4>
      </div>
    )
  }
  adds=()=>{
    this.setState({nu:this.state.nu+1})
  }
}
ReactDOM.render(<MyEve />, document.getElementById("app"));
```

```
class MyEve extends React.Component {
    constructor(){ ……
                  }
    // 组件
    render(){……
            }

    // 事件
    adds=()=>{……
              }

    // 组件是否需要被重新渲染 钩子函数
    // 结论：只要显示地，定义了 shouldComponentUpdate， 则必须 return 一个布尔值
    shouldComponentUpdate(){
        // 如果 在 shouldComponentUpdate 中返回了 true，则页面会立即进行重新 渲染，
        //从而让 页面 和 state 中地数据保持一致
        // 如果 在 shouldComponentUpdate 中返回了 false，则只会更新 state 状态中的数据，
        //但是 页面不会重新 渲染；

        // 值改变，页面改变
        // return true;

        // 值改变页面不改变
        return false;
    }
}
```

需求：值为奇数不修改，值为偶数修改页面

```
// 组件是否需要被重新渲染 钩子函数
shouldComponentUpdate(){
    if(this.state.nu %2 == 0){
        return true
    }else{
        return false;
    }
}
```

注意：在 shouldComponentUpdate 中，如果直接从 this.state 上获取状态值，获取到的，是上一次的旧的状态值； 如果要获取最新的状态值，需要从 shouldComponentUpdate 的形参列表中，来定义和接收

如下：

```
// 组件是否需要被重新渲染 钩子函数
shouldComponentUpdate(nextProps,nextState){
    // 注意：在 shouldCom,onentUpdate 中，如果直接从 this.state 上获取状态值，获取到的，
    // 是上一次的旧的状态值；
    // 如果要获取最新的状态值，需要从 shouldComponentUpdate 的形参列表中，来定义和接收
    if(nextState.nu %2 === 0){
        return true
    }else{
        return false;
    }
}
```

```
// 组件将要重新渲染
// 此时，浏览器中的页面，还是上一次的旧页面
componentWillUpdate(nextProps, nextState) {
    // console.log(nextState.count)
    // const h3 = document.getElementById('myh3')
    // console.log(h3.innerHTML)
}

// 组件已经完成了重新渲染，此时 ，浏览器中的页面是最新的！！！
componentDidUpdate(prevProps, prevState) {
    // 在 componentDidUpdate 中，如果想获取最新的 props 和 state 值，
    // 直接访问 this.props 或 this.state 即可；
    // 如果想 获取上一次的，旧的 props 和 state 值，需要通过形参 进行接收；
    console.log(this.state.count)
    // const h3 = document.getElementById('myh3')
    // console.log(h3.innerHTML)
}
```

componentWillReceiveProps

将父级组件的值使用props传入子组件，子组件的的state值直接使用父级的props数据，修改父级组件的数据更改props传入的值，在子级组件中设计狗仔函数

```jsx
import React from "react";
import ReactDOM from "react-dom";


// 计数器组件
class MyEve extends React.Component {
  constructor(props){
    super(props);
    this.state = {
      nu:props.nn
    }
  }
  // 组件
  render(){
    return (
      <div>
        <button onClick={this.adds}>+1</button>
        <h4>{this.state.nu}</h4>
      </div>
    )
  }

  // 事件
  adds=()=>{
    this.setState({nu:this.state.nu+1})
    // console.log(this.state.nu)
  }
  // props 改变的钩子函数
  componentWillReceiveProps(nextProps){
    // console.log(this.state.nu);
    // console.log(nextProps);
    this.setState({nu:nextProps.nn})
  }

}


// props值组件
class Pro extends React.Component{
    constructor(){
      super();
      this.state = {
        nn:0
      }
    }

    render(){
      return (
        <div>
          <h3>{this.state.nn}</h3>
          <button onClick={this.changes}>修改</button>
          <hr ></hr>
          {/* 将state 数据 传入组件 props 值 */}
```

```
          <MyEve nn={this.state.nn}/>
        </div>
      )
    }
    // 事件修改值
    changes = ()=>{
      this.setState({nn:this.state.nn+1})
    }

}

ReactDOM.render(<Pro />, document.getElementById("app"));
```

6.3 子组件向父组件传值

```
import React from "react";
import ReactDOM from "react-dom";


// 计数器组件
class MyEve extends React.Component {
  constructor(props){
    super(props);
    this.state = {
      nu:props.nn
    }
  }
  // 组件
  render(){
    return (
      <div>
        <button onClick={this.adds}>+1</button>
        <h4>{this.state.nu}</h4>
      </div>
    )
  }

  // 事件
  adds=()=>{
    // 调用父组件传入的函数，将子组件的数据传入回调
    this.props.fun(this.state.nu+1);
    this.setState({nu:this.state.nu+1})
    // console.log(this.state.nu)
  }

}


// props值组件
class Pro extends React.Component{
    constructor(){
```

```
      super();
      this.state = {
        nn:0
      }
    }

    render(){
      return (
        <div>
          <h3>{this.state.nn}</h3>
          <button onClick={this.changes}>修改</button>
          <hr ></hr>
          {/* 将 函数 父级组件的方法 传入组件 props 值 */}
          <MyEve fun={this.ff} nn={this.state.nn}/>
        </div>
      )
    }
    // 子级组件回调
    ff = (data)=>{
      this.setState({nn:data})
    }
}

ReactDOM.render(<Pro />, document.getElementById("app"));
```

第7章 路由

[react-router官方文档]

7.1 基本使用

7.1.1 安装

运行 `npm i react-router-dom` 安装react路由依赖项

创建一个 `App.js` 根组件，并在根组件中，按需导入路由需要的三个组件

```
import React from "react";
import ReactDOM from "react-dom";

// 引入全局应用组件
import App from './App'

ReactDOM.render(<App />, document.getElementById("app"));
```

`/src/App.js`

HashRouter: 表示路由的包裹容器，这个组件，在项目中，只使用唯一的一次!

就在 App 根组件中，作为最外层的容器，报包裹住整个App中的UI结构

```jsx
import React from "react";
// 引入路由
import { HashRouter, Route, Link } from "react-router-dom";
class App extends React.Component {
  render() {
    return (
      // 全局应用组件路由
      // 此后所有内容 都在 HashRouter
      <HashRouter>
        {/* HashRouter 里面只能有一个根标签 */}
        <div>123</div>
      </HashRouter>
    );
  }
}
export default App;
```

7.1.2 路由跳转链接

Link: 表示路由的链接；就相当于 Vue 中的 `<router-link></router-link>`

Link 组件的属性节点上，有 to 属性，表示点击这个链接之后，会跳转到哪个路由地址

```jsx
render() {
    return (
        // 全局应用组件路由
        // 此后所有内容 都在 HashRouter
        <HashRouter>
            {/* HashRouter 里面只能有一个根标签 */}
            <div>
                <h1>组件路由</h1>
                {/* 添加路由跳转链接 */}
                <Link to="/home">首页</Link>  
                <Link to="/movie">电影</Link>  
                <Link to="/about">关于</Link>
            </div>
        </HashRouter>
    );
}
```

7.1.3 路由规则匹配

Route: 表示路由的匹配关系，可以把 每个 Route，都看成是每一个路由规则；

Route 的属性节点中，包含 path 属性和 component 属性；

其中，path 表示当前路由规则，要匹配的 hash 地址； component 表示当前路由规则对应要显示的组件

注意： Route 有两层身份： 1. 路由规则 2. 占位符

```
import React from "react";
// 引入路由
import { HashRouter, Route, Link } from "react-router-dom";

// 引入组件
import Home from './components/Home'
import Movie from './components/Movie'
import About from './components/About'

class App extends React.Component {
  render() {
    return (
      <HashRouter>
        <div>
          <h1>组件路由</h1>
          {/* 路由链接组件 */}
          <Link to="/home">首页</Link>  
          <Link to="/movie">电影</Link>  
          <Link to="/about">关于</Link>
          <hr></hr>
          {/* 新增一个重定向的路由规则 */}
          <Route path="/home" component={Home} ></Route>
          <Route path="/movie" component={Movie} ></Route>
          <Route path="/about" component={About} ></Route>
        </div>
      </HashRouter>
    );
  }
}
export default App;
```

7.2 嵌套路由

在 `/src/components/Home.js` 组件中，设置嵌套路由

```
import React from 'react'

// 引入路由
import { HashRouter, Route, Link } from "react-router-dom";

// 引入组件
import Ha from './Ha'
import He from './He'

export default class Home extends React.Component {
  constructor(props) {
    super(props)
    this.state = {}
  }

```

```
  render() {
    return (
      <div>
        <h3>Home组件</h3>
        {/* 设置路由跳转 */}
        <Link to="/home/ha">哈哈</Link>   
        <Link to="/home/he">呵呵</Link>
        <hr/>
        {/* 设置子路由匹配规则 */}
        <Route path="/home/ha" component={Ha} ></Route>
        <Route path="/home/he" component={He} ></Route>
      </div>
    )
  }
}

```

> 注意: 在 vue 中有专门的 集中式 路由匹配管理文件 `router/index.js` ，而在react 中是没有的；所有的路由都是分散在各个组件中的；

7.3 路由传参

在 `Link` 路由链接中携带路由参数：

```
<Link to="/movie/top50/3">电影</Link>
```

在 `Route` 路由规则中定义参数：

```
<Route path="/movie/:type/:id" component={Movie} />
```

在 `Route` 路由规则中， `component` 属性指定的组件中，获取路由参数：

```
render() {
    // 所有路由匹配到的参数，都存放到了 this.props.match 中
    console.log(this.props)
    return <div>
      <h3>Moive</h3>
        <h4>{this.props.match.params.type}---{this.props.match.params.id}</h4>
    </div>
}
```

7.4 编程式导航

1. 使用 `this.props.history.push('要跳转到的路径')` 跳转到指定页面

2. 使用 `this.props.history.go(n)` 向前向后跳转指定个数的历史记录
3. 使用 `this.props.history.goBack()` 后退一个历史记录
4. 使用 `this.props.history.goForward()` 前进一个历史记录

```
render() {
    return (
        <div>
            <h3>About组件</h3>
            {/* 点击按钮跳转 */}
            <button onClick={this.goMovie}>去电影</button>
        </div>
    )
}

goMovie = () => {
    // console.log(this.props)
    // 编程式导航跳转
    this.props.history.push('/movie/top666/555')
}
```

第8章 React UI组件--Ant Design

8.1 基本使用

官方地址： https://ant.design/index-cn

构建 react 项目 `create-react-app reactapp`

安装 Ant Design UI组件： `npm install antd`

基本使用：

```
import React, { Component } from 'react';

import Button from 'antd/lib/button';
import Icon from 'antd/lib/icon';
import Pagination from 'antd/lib/pagination';

// import './App.css';
import 'antd/dist/antd.css';

class App extends Component {
  render() {
    return (
      <div>
        <Button type="danger">Button</Button>
        <Icon type="twitter" />
        <Pagination defaultCurrent={1} total={50} />
      </div>
    );
```

```
  }
}

export default App;

```

8.2 按需载入

我们现在已经把组件成功运行起来了，但是在实际开发过程中还有很多问题，例如上面的例子实际上加载了全部的 antd 组件的样式（对前端性能是个隐患）。

此时我们需要对 create-react-app 的默认配置进行自定义，这里我们使用 react-app-rewired （一个对 create-react-app 进行自定义配置的社区解决方案）。

引入 react-app-rewired 并修改 package.json 里的启动配置。

`npm i react-app-rewired`

```
/* package.json */
"scripts": {
-   "start": "react-scripts start",
+   "start": "react-app-rewired start",
-   "build": "react-scripts build",
+   "build": "react-app-rewired build",
-   "test": "react-scripts test",
+   "test": "react-app-rewired test",
}
```

然后在项目根目录创建一个 `config-overrides.js` 用于修改默认配置。

```
module.exports = function override(config, env) {
  // do stuff with the webpack config...
  return config;
};
```

使用 babel-plugin-import

babel-plugin-import 是一个用于按需加载组件代码和样式的 babel 插件（原理），现在我们尝试安装它并修改 `config-overrides.js` 文件。

`npm i babel-plugin-import`

```
+ const { injectBabelPlugin } = require('react-app-rewired');

module.exports = function override(config, env) {
    +   config = injectBabelPlugin(
    +     ['import', { libraryName: 'antd', libraryDirectory: 'es', style: 'css' }],
    +     config,
    +   );
     return config;
};
```

修改 `/src/App.js`

```
import React, { Component } from 'react';

//import Button from 'antd/lib/button';
//import Icon from 'antd/lib/icon';
//import Pagination from 'antd/lib/pagination';

import {Button,Icon,Pagination} from 'antd';
import 'antd/dist/antd.css';

class App extends Component {
  render() {
    return (
      <div>
        <Button type="danger">Button</Button>
        <Icon type="twitter" />
        <Pagination defaultCurrent={1} total={50} />
      </div>
    );
  }
}
export default App;
```

访问页面，antd 组件的 js 和 css 代码都会按需加载

第9章 开始项目

项目在线地址演示： http://www.liulongbin.top:3000/#/home

> API 请求地址：`http://www.liulongbin.top:3005`

1. 正在热映：`/api/v2/movie/in_theaters?start=0&count=1`
2. 即将上映：`/api/v2/movie/coming_soon?start=0&count=1`
3. Top250：`/api/v2/movie/top250?start=0&count=1`
4. 电影详情：`/api/v2/movie/subject/26861685`

9.1 项目首页

布局： https://ant.design/components/layout-cn/

`/src/global.css`

```css
body {
  margin: 0;
  padding: 0;
  font-family: sans-serif;
}

#root {
  height: 100%;
}

.logo {
  width: 120px;
  height: 31px;
  background: rgba(255, 255, 255, 0.2);
  margin: 16px 24px 16px 0;
  float: left;
  background: url('/logo.png');
  background-size: cover;
}

.ant-card-meta-title,
.ant-card {
  font-size: 13px !important;
}
```

`/src/index.js`

```js
import React from 'react';
import ReactDOM from 'react-dom';
import './global.css'
import App from './App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(<App />, document.getElementById('root'));

serviceWorker.unregister();
```

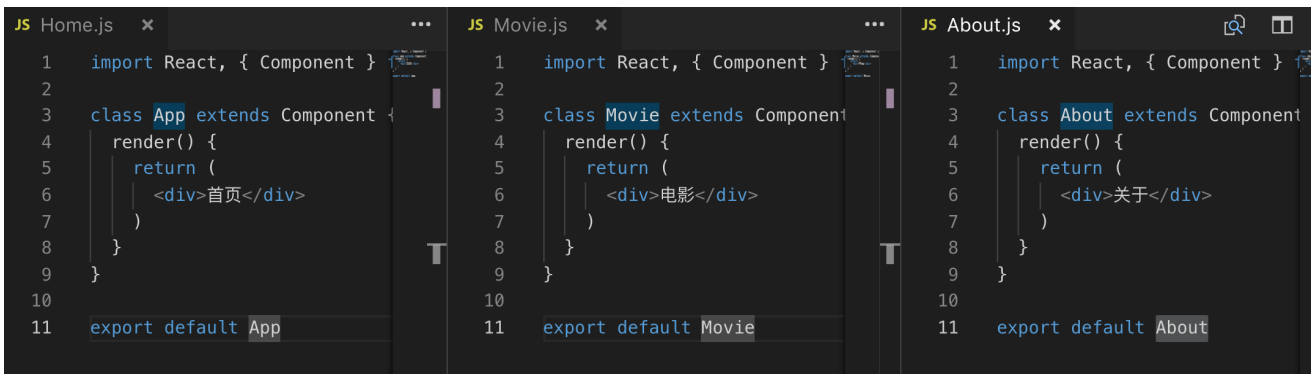
`/src/App.js`

```js
import React, { Component } from "react";
import { Layout, Menu } from "antd";
// 导入路由相关的组件
import { HashRouter } from "react-router-dom";
const { Header, Content, Footer } = Layout;

class App extends Component {
```

```
  render() {
    return (
      <HashRouter>
        <Layout className="layout" style={{ height: '100%' }}>
          <Header >
            <div className="logo" />
            <Menu
              theme="dark"
              mode="horizontal"
              defaultSelectedKeys={["2"]}
              style={{ lineHeight: "64px" }}
            >
              <Menu.Item key="1">首页</Menu.Item>
              <Menu.Item key="2">电影</Menu.Item>
              <Menu.Item key="3">关于</Menu.Item>
            </Menu>
          </Header>
          <Content style={{ padding: "0 50px" }}>

          </Content>
          <Footer style={{ textAlign: "center" }}>
          传智播客 ©2018 黑马程序员
          </Footer>
        </Layout>
      </HashRouter>
    );
  }
}

export default App;
```

9.2 导航路由切换

设置三个组件: `/src/components/Home.js`、 `/src/components/About.js`、 `/src/components/Movie.js`



设置路由匹配及跳转

`src/App.js`

```
import React, { Component } from "react";
import { Layout, Menu } from "antd";
```

```
// 导入路由相关的组件
import { HashRouter, Route, Link, Redirect } from "react-router-dom";
// 引入组件
import Home from "./components/Home";
import Movie from "./components/Movie";
import About from "./components/About";

const { Header, Content, Footer } = Layout;

class App extends Component {
  render() {
    return (
      <HashRouter>
        <Layout className="layout" style={{ height: "100%" }}>
          <Header>
            <div className="logo" />
            <Menu
              theme="dark"
              mode="horizontal"
              defaultSelectedKeys={["1"]}
              style={{ lineHeight: "64px" }}
            >
              <Menu.Item key="1">
                <Link to="/home">首页</Link>
              </Menu.Item>
              <Menu.Item key="2">
                <Link to="/movie">电影</Link>
              </Menu.Item>
              <Menu.Item key="3">
                <Link to="/about">关于</Link>
              </Menu.Item>
            </Menu>
          </Header>
          <Content style={{ padding: "0 50px" }}>
            <Route exact path="/" render={() => <Redirect to="/home" />} />
            <Route path="/home" component={Home} />
            <Route path="/movie" component={Movie} />
            <Route path="/about" component={About} />
          </Content>
          <Footer style={{ textAlign: "center" }}>
            传智播客 ©2018 黑马程序员
          </Footer>
        </Layout>
      </HashRouter>
    );
  }
}
export default App;
```

9.3 刷新选中

当地址栏为 movie 时，刷新页面后，却选中了首页：

原生 JS 提供了 `window.location.hash` 可以获取地址栏数据：

```
window.location.hash.split('/')[1]
"movie"
```

修改代码：

```
class App extends Component {
  constructor (props){
    super(props);
    this.state = {
      // 获取地址栏数据，随时更改
      locations:[window.location.hash.split('/')[1] || 'home']
    }
  }
  render() {
    return (
      <HashRouter>
        <Layout className="layout" style={{ height: "100%" }}>
          <Header>
            <div className="logo" />
            <Menu
              theme="dark"
              mode="horizontal"
              // 使用this.state 中的数据替换原有属性值
              defaultSelectedKeys={this.state.locations}
              style={{ lineHeight: "64px" }}
            >
            {/* 将key 值换为具体的路由名称 */}
              <Menu.Item key="home">
                <Link to="/home">首页</Link>
              </Menu.Item>
              <Menu.Item key="movie">
                <Link to="/movie">电影</Link>
              </Menu.Item>
              <Menu.Item key="about">
                <Link to="/about">关于</Link>
              </Menu.Item>
            </Menu>
          </Header>
          <Content style={{ padding: "0 50px" }}>……
          </Content>
          <Footer style={{ textAlign: "center" }}>
            传智播客 ©2018 黑马程序员
          </Footer>
        </Layout>
      </HashRouter>
    );
  }
```

```
}

export default App;
```

9.4 电影页布局

https://ant.design/components/layout-cn/#%E7%BB%84%E4%BB%B6%E6%A6%82%E8%BF%B0

```
import React, { Component } from "react";
import { Layout, Menu } from "antd";

const {Content, Sider } = Layout;

class Movie extends Component {
  render() {
    return (
      <Layout>
        <Sider width={200} style={{ background: "#fff" }}>
          <Menu
            mode="inline"
            defaultSelectedKeys={["1"]}
            defaultOpenKeys={["sub1"]}
            style={{ borderRight: 0 }}
          >
            <Menu.Item key="1">
              <span>正在热映</span>
            </Menu.Item>
            <Menu.Item key="2">
              <span>即将上映</span>
            </Menu.Item>
            <Menu.Item key="3">
              <span>Top 250</span>
            </Menu.Item>
          </Menu>
        </Sider>
        <Layout style={{ padding: "0 24px 24px" }}>
          <Content
            style={{
              background: "#fff",
              padding: 24,
              margin: 0,
              minHeight: 880
            }}
          >
            Content
          </Content>
        </Layout>
      </Layout>
    );
  }
}
```

```
44
45 export default Movie;
```

正在热映、即将上映、Top 250三个都使用相同的布局，当用户点击时，只需要获取不同的数据即可：

9.5 电影列表页

`/src/components/Movie.js`

```
1  import MovieList from "./movieList";
2  ……
3  <Layout>
4      <Sider width={200} style={{ background: "#fff" }}>
5          <Menu
6              mode="inline"
7              defaultSelectedKeys={["1"]}
8              defaultOpenKeys={["sub1"]}
9              style={{ borderRight: 0 }}
10             >
11             <Menu.Item key="1">
12                 <Link to="/movie/in_theaters/1">
13                     <span>正在热映</span>
14                 </Link>
15             </Menu.Item>
16             <Menu.Item key="2">
17                 <Link to="/movie/coming_soon/1">
18                     <span>即将上映</span>
19                 </Link>
20             </Menu.Item>
21             <Menu.Item key="3">
22                 <Link to="/movie/top250/1">
23                     <span>Top 250</span>
24                 </Link>
25             </Menu.Item>
26         </Menu>
27     </Sider>
28     <Layout style={{ padding: "0 24px 24px" }}>
29         {/* 根据传入值得到不同的数据结果并展示 */}
30         <Route path="/movie/:type/id" component={MovieList} />
31     </Layout>
32 </Layout>
```

`/src/components/movieList.js`

```
1 import React, { Component } from "react";
2 import Axios from "axios";
3 import { Card } from "antd";
```

```
const { Meta } = Card;

class MovieList extends Component {
  constructor(props) {
    super(props);
    this.state = {
      type: "", //存放电影类型
      page: 1, // 存放页面
      movielist: [] // 存放电影数据
    };
  }
  render() {
    return (
      // 对卡片布局
      <div
        style={{
          display: "flex",
          flexWrap: "wrap",
          justifyContent: "space-between"
        }}
      >
        {/* 循环电影数据列表 */}
        {this.state.movielist.map(item => {
          return (
            <Card
              hoverable
              style={{ width: 160, margin: "5px 0" }}
              cover={<img alt="example" src={item.images.large} />}
            >
              <Meta
                title={item.title}
                description={"豆瓣评分: " + item.rating.average + "分"}
              />
            </Card>
          );
        })}
      </div>
    );
  }

  // shouldComponentUpdate(pro, ste) {
  //   console.log(ste);
  // }

  // 利用钩子函数，监听路由传参变化
  componentWillReceiveProps(nextProps) {
    this.setState({
      type: nextProps.match.params.type,
      page: nextProps.match.params.id
    });
    // 发送axios 请求，获取电影数据
    Axios.get(
      "http://www.liulongbin.top:3005/api/v2/movie/" +
```

```
        nextProps.match.params.type +
        "?start=0&count=" +
        nextProps.match.params.id
    ).then(backdata => {
      console.log(backdata);
      this.setState({
        movielist: backdata.data.subjects
      });
    });
  }
}
export default MovieList;

```

9.6 电影详情页

添加点击事件：

`/src/components/movieList.js`

```
{this.state.movielist.map(item => {
    return (
        // 添加点击事件
        <Card
            onClick={() => this.goDetail(item.id)}
            key={item.id}
            hoverable
            style={{ width: 160, margin: "5px 0" }}
            cover={<img alt="example" src={item.images.large} />}
            >
            <Meta
                title={item.title}
                description={"豆瓣评分：" + item.rating.average + "分"}
                />
        </Card>
    );
})}

        ……
    ……
……

  goDetail(id) {
    // console.log(id);
    this.props.history.push("/movie/detail/" + id);
  }
```

设置路由监听

/src/components/Movie.js

```
import MovieDetail from "./movieDetail";
……

<Layout style={{ padding: "0 24px 24px" }}>
    <Route path="/movie/detail/:id" component={MovieDetail} />
    {/* 根据传入值得到不同的数据结果并展示 */}
    <Route path="/movie/:type/:id" component={MovieList} />
</Layout>
```

/src/components/movieDetail.js

```
import React from "react";
import Axios from "axios";

export default class MovieDetail extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      // 电影详情
      minfo: {},
      img:''
    };
  }

  render() {
    return (
      <div>
        <div>
          <div style={{ textAlign: 'center' }}>

            <h1>{this.state.minfo.title}</h1>
            <img src={this.state.img} alt="" />
          </div>
          <p style={{ lineHeight: '30px', textIndent: '2em' }}>
{this.state.minfo.summary}</p>
        </div>
      </div>
    );
  }

  componentWillMount() {
    Axios.get(
      "http://www.liulongbin.top:3005/api/v2/movie/subject/" +
        this.props.match.params.id
    ).then(backdata => {
      console.log(backdata);
      this.setState({
        minfo: backdata.data,
        img:backdata.data.images.large
```

```
      });
    });
  }
}
```

而此时，我们发现电影详情页和列表页会同时出现：

`/src/components/Movie.js`

```
<Layout style={{ padding: "0 24px 24px" }}>
    <Switch>
        {/* 电影列表的 路由规则是逐条匹配的，只要匹配成功就会加载相应的组件 */}
        {/* 通过 Switch 实现路由的单条匹配 */}
        {/* 如果前面的路由匹配成功，则放弃后面路由规则的匹配 */}
        {/* 根据传入值得到不同的数据结果并展示 */}
        <Route path="/movie/detail/:id" component={MovieDetail} />
        <Route path="/movie/:type/:id" component={MovieList} />
    </Switch>
</Layout>
```