爱旅行项目—React技术分析

React作为目前最流行的前端框架之一，其受欢迎程度不容小觑。React 起源于 Facebook 的内部项目，因为该公司对市场上所有 JavaScript MVC 框架，都不满意，就决定自己写一套，用来架设 Instagram 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。既然 React 这么热门，看上去充满希望，当然应该好好学一下。从技术角度，可以满足好奇心，提高技术水平；从职业角度，有利于求职和晋升，有利于参与潜力大的项目。

通过React这门框架,我们可以学到许多其他前端框架所缺失的东西，也是其创新性所在的地方，比如虚拟DOM、数据操作UI等，同时在React的开发中我们还使用JSX语法、ES6语法来提高我们的开发效率。那我们为什么要使用这些技术进行开发？除了能提高我们个人能力外，还能开拓我们的思维，丰富我们的经验，提高我们的实战能力，帮助我们找到一个更好的工作。

## 前言

首先在学习这门框架前，我们需要了解一些技术，那我们需要了解什么技术？我们使用的技术需要满足什么条件？首先我们会使用到传统的技术CSS和原生JS。为了提高我们的开发效率和提高我们的代码质量，我们在此项目中使用到了React技术，针对于React组件化开发的思想，我们需要使用JSX来提高我们组件的产出效率，同时我们还使用ES6来简化语法提高我们的开发效率，同时开发的过程中我们需要使用第三方的插件antd组件，来减轻我们组件开发的压力。我们使用npm解决包依赖问题，使用webpack解决打包部署和本地调试工作，因此我们需要对以下知识有所了解：

1. CSS基础
2. 原生JS基础
3. JSX语法，作用：提高开发效率，简化代码逻辑；
4. ES6规范，作用：简化语法，提高开发效率；
5. npm包管理基础，参见《npm使用指南》
6. webpack构建项目基础，参见《webpack使用指南》

以上六个知识点也是目前学习其他前端框架所必须了解的前置任务。JS和CSS就不多说了，npm是目前最提倡也是占据主导地位的包管理工具，还在用bower或者其他包管理工具的同学可以考虑下了，使用npm主要是用于包管理，便于第三方文件的添加和删除，保证引用代码的整洁度，方便第三方包管理（例如：antd插件的使用）；而webpack作为新一代打包工具，已经在前端打包工具中独占鳌头，和Browserify相比也有很大优势，主要用于打包部署，本地环境调试时使用，结合npm使用，前端项目开发会更方便。对于JSX语法，它是一种JavaScript语法扩展，在React中可以方便地用来描述UI，提高代码的编写效率。至于ES6规范虽然现在主流浏览器还不兼容，但可以使用babel等转换器进行转换，ES6简洁的语法深受开发者的喜爱，你也会喜欢上他的。

根据React技术分析，并结合其他一些主流前端框架，我个人认为完成这个React项目开发，必须要学会使用：组件、路由、状态管理。那么接下来我就基于这三者来介绍React，在学习React框架之前，首先需要学一下前置知识：JSX语法、ES6语法规范。

## JSX语法

JSX在React中使用给我们带来了很大的便利，JSX的语法实际上十分简单也很容易掌握，在学习JSX语法之前大家想这样一个问题，如果没有JSX你要用多长时间完成一个相同的组件开发？以下我们主要从JSX定义到使用进行分析和讲解；

### 1.21 JSX是什么

JSX是一种像下面这样的语法：

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

它是一种JavaScript语法扩展，在React中可以方便地用来描述UI。本质上，JSX为我们提供了创建react元素方法（React.createElement(component, props, ...children)）的语法糖（syntactic sugar）。上面的代码实质上等价于：

var element = React.createElement(

"h1",

null,

"Hello, world!"

);

### 1.22 JSX代表JS对象

JSX本身也是一个表达式，在编译后，JSX表达式会变成普通的javascript对象。

你可以在if语句或for循环中使用JSX，你可以将它赋值给变量，你可以将它作为参数接收，你也可以在函数中返回JSX。

例如下面的代码：

function getGreeting(user) {

if (user) {

return <h1>Hello, {formatName(user)}!</h1>;

}

return <h1>Hello, Stranger.</h1>;

}

上面的代码在if语句中使用JSX，并将JSX作为函数返回值。实际上，这些JSX经过编译后都会变成JavaScript对象。

经过babel编译后会变成下面的js代码：

function test(user) {

if (user) {

return React.createElement(

"h1",

null,

"Hello, ",

formatStr(user),

"!"

);

}

return React.createElement(

"h1",

null,

"Hello, Stranger."

);

}

### 1.23 在JSX中使用JavaScript表达式

在JSX中插入JavaScript表达式十分简单，直接在JSX中将JS表达式用大括号括起来即可。例如：

function formatName(user) {

return user.firstName + ' ' + user.lastName;

}

const user = {

firstName: 'Harper',

lastName: 'Perez'

};

const element = (

<h1>

Hello, {formatName(user)}!

</h1>

);

ReactDOM.render(

element,

document.getElementById('root')

);

上面的代码中用到了函数调用表达式fromatName(user)。

在JavaScript中，表达式就是一个短语，Javascript解释器会将其计算出一个结果，常量就是最简单的一类表达式。常用的表达式有：

1.变量名；

2.函数定义表达式；

3.属性访问表达式；

4.函数调用表达式；

5.算数表达式；

6.关系表达式；

7.逻辑表达式；

需要注意的是，if语句以及for循环不是JavaScript表达式，不能直接作为表达式写在{}中，但可以先将其赋值给一个变量（变量是一个JavaScript表达式）:

function NumberDescriber(props) {

let description;

if (props.number % 2 == 0) {

description = <strong>even</strong>;

} else {

description = <i>odd</i>;

}

return <div>{props.number} is an {description} number</div>;

}

### 1.24 JSX属性值

你可以使用引号将字符串字面量指定为属性值：

const element = <div tabIndex="0"></div>;

注意这里的”0”是一个字符串字面量。

或者你可以将一个JavaScript表达式嵌在一个大括号中作为属性值：

const element = <img src={user.avatarUrl}></img>;

这里用到的是JavaScript属性访问表达式，上面的代码将编译为：

const element = React.createElement("img", { src: user.avatarUrl });

### 1.25注意事项

1.使用JSX时要引入React库

前面已经解释过了，JSX是React.createElement方法的语法糖，因此在使用JSX的作用域中必须引入React库。

如果你使用了JS打包工具，你可以在文件的头部作如下引用：

import React from 'react';

2.注意引入JSX中用到的自定义组件

JSX中用到的组件可能并不会在JavaScript中直接引用到，但自定义组件本质上就是一个JS对象，你在JSX中使用的时候，需要首先将该组件引入到当前作用域：

import MyComponent from './MyComponent.js'

...

<Outer>

<MyComponent />

</Outer>

3.自定义组件首字母一定要大写

JSX中小写字母开头的element代表HTML固有组件如div，span，p，ul等。用户自定义组件首字母一定要大写如<Header>、<Picker>。

4.元素标签名不能使用表达式

下面的代码将产生错误：

const components = {

photo: PhotoStory,

video: VideoStory

};

function Story(props) {

// Wrong! JSX标签名不能使用表达式

return <components[props.storyType] story={props.story} />;

}

如果你需要使用一个表达式来决定元素标签，你应该先将该表达式的值赋给一个大写字母开头的变量：

const components = {

photo: PhotoStory,

video: VideoStory

};

function Story(props) {

// Correct! JSX type can be a capitalized variable.

const SpecificStory = components[props.storyType];

return <SpecificStory story={props.story} />;

}

5.设置style属性

在设置标签style属性的时候，要注意，我们是将一个描述style的对象以JavaScipt表达式的形式传入。因此应该有2层大括号：

<div style={{color:'red', margin:'10px auto'}}></div>

## 1.3 ES6规范

2015年6月份，在es5的基础上扩展了很多新的功能， 称为es6/es2015， ES7将在2017年6月份出来。我们要学习的仅仅只是es6中的部分常用新功能(类型规范、数组类型、解构类型、arrow箭头函数、map + set + weakmap + weakset 数据结构)，这些功能在使用的时候一定要慎重，因为他们中有一部分js代码在部分浏览器不能兼容，但是所有写在服务器端的代码基本上都支持ES6的写法。

### 1.3.1类型规范

对于常量或不修改的变量声明使用const，对于只在当前作用域下有效的变量，应使用let，全局变量使用var。将所有 const 变量放在一起，然后将所有 let 变量放在一起。

const foo = 1;

let foo1 = 2;

let bar = foo;

bar = 9;

foo1 = 3;

console.log(foo, bar); // => 1, 9

console.log(foo, bar, str); // => str is not defined

const和let使用时注意，let 和 const 都是块作用域的

{

let a = 1;

const b = 1;

}

console.log(a); // ReferenceError a is not defined

console.log(b); // ReferenceError b is not defined

### 1.32 数组类型

使用字面量语法创建数组

// 不推荐

const items = new Array();

// 推荐

const items = [];

如果你不知道数组的长度，使用 push

const someStack = [];

// 推荐

someStack.push('abracadabra');

使用 ... 来拷贝数组，不要使用 Array.from、Array.of等数组的新的内置API，Array新api用于适合的场景

// 不推荐

const len = items.length;

const itemsCopy = [];

let i;

for (i = 0; i < len; i++) {

itemsCopy[i] = items[i];

}

// 推荐

const itemsCopy = [...items];

### 1.33解构Destructuring

对象解构元素与顺序无关，对象指定默认值时仅对恒等于undefined ( !== null ) 的情况生效

若函数形参为对象时，使用对象解构赋值

// 可以（不推荐）

function someFun(opt) {

let opt1 = opt.opt1;

let opt2 = opt.opt2;

console.log(op1);

}

// 推荐

function someFun(opt) {

let { opt1, opt2 } = opt;

console.log(`$(opt1) 加上 $(opt2)`);

}

或

function someFun({ opt1, opt2 }) {

console.log(opt1);

}

若函数有多个返回值时，使用对象解构，不使用数组解构，避免添加顺序的问题

// 不推荐

function anotherFun() {

const one = 1, two = 2, three = 3;

return [one, two, three];

}

const [one, three, two] = anotherFun(); // 顺序乱了

// one = 1, two = 3, three = 2

// 推荐

function anotherFun() {

const one = 1, two = 2, three = 3;

return { one, two, three };

}

const { one, three, two } = anotherFun(); // 不用管顺序

// one = 1, two = 2, three = 3

已声明的变量不能用于解构赋值（语法错误）

// 语法错误

let a;

{ a } = { b: 123};

数组解构时数组元素与顺序相关

例如交换数组两个元素的值

let x = 1;

let y = 2;

// 不推荐

let temp;

temp = x;

x = y;

y = temp;

//推荐

[x, y] = [y, x]; // 交换变量

将数组成员赋值给变量时，使用数组解构

const arr = [1, 2, 3, 4, 5];

// 不推荐

const one = arr[0];

const two = arr[1];

// 推荐

const [one, two] = arr;

函数有多个返回值时使用对象解构，而不是数组解构。

这样你就可以随时添加新的返回值或任意改变返回值的顺序，而不会导致调用失败。

function processInput(input) {

return [left, right, top, bottom];

}

const [left, \_\_, top] = processInput(input);

// 推荐

function processInput(input) {

return { left, right, top, bottom };

}

const { left, right } = processInput(input);

解构赋值在取JSON数据的时候有很高的处理效率，可以好好掌握一下；

### 1.34 arrow箭头函数

当必须使用函数表达式时（例如传递一个匿名函数时），请使用箭头函数

箭头函数提供了更简洁的语法，并且箭头函数中 this 对象的指向是不变的，this 对象绑定定义时所在的对象，这通常是我们想要的。如果该函数的逻辑非常复杂，请将该函数提取为一个函数声明。

// 一般写法

"use strict";

var fn = function fn(v) {

return console.log(v);

};

//推荐

var fn= (v=>console.log(v));

箭头函数总是用括号包裹参数，省略括号只适用于单个参数，并且还降低了程序的可读性

// 不推荐

[1, 2, 3].forEach(x => x \* x);

// 推荐

[1, 2, 3].forEach((x) => x \* x);

立即执行的匿名函数

// 函数表达式

// immediately-invoked function expression (IIFE)

(() => {

console.log('Welcome to World.');

})();

### 1.35 map + set + weakmap + weakset 数据结构

新加的集合类型，提供了更加方便的获取属性值的方法，可以检查某个属性是属于原型链上还是当前对象的，并用获取对象的set和get方法，但是，推荐使用weakmap和weakset，而不是map和set，除非必须使用。普通集合会阻止垃圾回收器对这些作为属性键存在的对象的回收，有造成内存泄漏的危险

// 不推荐, Maps

var wm = new Map();

wm.set(key, { extra: 42 });

wm.size === 1

// 不推荐, Sets

var ws = new Set();

ws.add({ data: 42 });

//推荐, Weak Maps

var wm = new WeakMap();

wm.set(key, { extra: 42 });

wm.size === undefined

//推荐, Weak Sets

var ws = new WeakSet();

ws.add({ data: 42 });//因为添加到ws的这个临时对象没有其他变量引用它，所以ws不会保存它的值，也就是说这次添加其实没有意思

// 不推荐

let object = {},

object.hasOwnProperty(key)

// 推荐

let object = new WeakSet();

object.has(key) === true;

## React

在看完JSX语法和ES6语法之后，我们再来学习React技术会事半功倍，首先我们先从组件下手学习，大家想一下，如果没有JSX你的组件会写多长时间？没有ES6你是如何解析数组、设定this对象的作用域？JSX跟ES6给你带来什么？大家需要带着这些问题看一下关于React组件、路由、状态的技术知识，这样才能深入体会JSX跟ES6的优势；

### 1.41 React 组件

React推出后，出于不同的原因先后出现两种定义react组件的方式，殊途同归；具体的两种方式：

1、es5原生方式React.createClass定义的组件

2、es6形式的extends React.Component定义的组件

下面就来介绍以下是定义组件的两种方式：

**React.createClass定义的组件**

`React.createClass`是react刚开始推荐的创建组件的方式，这是ES5的原生的JavaScript来实现的React组件，其形式如下：

var InputControlES5 = React.createClass({

propTypes: {//定义传入props中的属性各种类型

initialValue: React.PropTypes.string

},

defaultProps: { //组件默认的props对象

initialValue: ''

},

// 设置 initial state

getInitialState: function() {//组件相关的状态对象

return {

text: this.props.initialValue || 'placeholder'

};

},

handleChange: function(event) {

this.setState({

text: event.target.value

});

},

render: function() {

return (

<div>

Type something:

<input onChange={this.handleChange} value={this.state.text} />

</div>

);

}

});

InputControlES6.propTypes = {

initialValue: React.PropTypes.string

};

InputControlES6.defaultProps = {

initialValue: ''

};

React.createClass和后面要描述的React.Component都是创建有状态的组件，这些组件是要

被实例化的，并且可以访问组件的生命周期方法。但是随着React的发展，React.createClass形式自身的问题暴露出来：

React.createClass会自绑定函数方法（不像React.Component只绑定需要关心的函数）导致不必要的性能开销，增加代码过时的可能性。

React.createClass的mixins不够自然、直观；React.Component形式非常适合高阶组件（Higher Order Components--HOC）,它以更直观的形式展示了比mixins更强大的功能，并且HOC是纯净的JavaScript，不用担心他们会被废弃。HOC可以参考无状态组件(Stateless Component) 与高阶组件。

**extends React.Component定义的组件**

React.Component是以ES6的形式来创建react的组件的，是React目前极为推荐的创建有状态组件的方式，最终会取代React.createClass形式；相对于 React.createClass可以更好实现代码复用。将上面React.createClass的

形式改为React.Component形式如下：

class InputControlES6 extends React.Component {

constructor(props) {

super(props);

// 设置 initial state

this.state = {

text: props.initialValue || 'placeholder'

};

// ES6 类中函数必须手动绑定

this.handleChange = this.handleChange.bind(this);

}

handleChange(event) {

this.setState({

text: event.target.value

});

}

render() {

return (

<div>

Type something:

<input onChange={this.handleChange}

value={this.state.text} />

</div>

);

}

}

InputControlES6.propTypes = {

initialValue: React.PropTypes.string

};

InputControlES6.defaultProps = {

initialValue: ''

};

### 1.42 React 路由

路由库React-Router。它是官方维护的，事实上也是唯一可选的路由库。它通过管理 URL，实现组件的切换和状态的变化，开发复杂的应用几乎肯定会用到。

**基本用法**

React Router 安装命令如下。

$ npm install -S react-router

使用时，路由器Router就是React的一个组件。

import { Router } from 'react-router';

render(<Router/>, document.getElementById('app'));

Router组件本身只是一个容器，真正的路由要通过Route组件定义。

import { Router, Route, hashHistory } from 'react-router';

render((

<Router history={hashHistory}>

<Route path="/" component={App}/>

</Router>

), document.getElementById('app'));

上面代码中，用户访问根路由/，组件APP就会加载到document.getElementById('app')。

你可能还注意到，Router组件有一个参数history，它的值hashHistory表示，路由的切换由URL的hash变化决定，即URL的#部分发生变化。举例来说，用户访问http://www.example.com/，实际会看到的是http://www.example.com/#/。

Route组件定义了URL路径与组件的对应关系。你可以同时使用多个Route组件。

<Router history={hashHistory}>

<Route path="/" component={App}/>

<Route path="/repos" component={Repos}/>

<Route path="/about" component={About}/>

</Router>

上面代码中，用户访问/repos（比如http://localhost:8080/#/repos）时，加载Repos组件；访问/about（http://localhost:8080/#/about）时，加载About组件。

**嵌套路由**

Route组件还可以嵌套。

<Router history={hashHistory}>

<Route path="/" component={App}>

<Route path="/repos" component={Repos}/>

<Route path="/about" component={About}/>

</Route>

</Router>

上面代码中，用户访问/repos时，会先加载App组件，然后在它的内部再加载Repos组件。

<App>

<Repos/>

</App>

App组件要写成下面的样子。

export default React.createClass({

render() {

return <div>

{this.props.children}

</div>

}

})

上面代码中，App组件的this.props.children属性就是子组件。

子路由也可以不写在Router组件里面，单独传入Router组件的routes属性。

let routes = <Route path="/" component={App}>

<Route path="/repos" component={Repos}/>

<Route path="/about" component={About}/>

</Route>;

<Router routes={routes} history={browserHistory}/>

**Path属性**

Route组件的path属性指定路由的匹配规则。这个属性是可以省略的，这样的话，不管路径是否匹配，总是会加载指定组件。

请看下面的例子。

<Route path="inbox" component={Inbox}>

<Route path="messages/:id" component={Message} />

</Route>

上面代码中，当用户访问/inbox/messages/:id时，会加载下面的组件。

<Inbox>

<Message/>

</Inbox>

如果省略外层Route的path参数，写成下面的样子。

<Route component={Inbox}>

<Route path="inbox/messages/:id" component={Message} />

</Route>

现在用户访问/inbox/messages/:id时，组件加载还是原来的样子。

<Inbox>

<Message/>

</Inbox>

上述讲到的三点路由知识，是最基本的，也会是最常用的几点，如果大家有兴趣的话，可以自己去React官方网站上自行学习；

这是官方的实例库（https://github.com/reactjs/react-router-tutorial/tree/master/lessons）

### 1.43 React 状态

React 把组件看成是一个状态机（State Machines）。通过与用户的交互，实现不同状态，然后渲染 UI，让用户界面和数据保持一致。

React 里，只需更新组件的 state，然后根据新的 state 重新渲染用户界面（不要操作 DOM）。

以下实例中创建了 LikeButton 组件，getInitialState 方法用于定义初始状态，也就是一个对象，这个对象可以通过 this.state 属性读取。当用户点击组件，导致状态变化，this.setState 方法就修改状态值，每次修改以后，自动调用 this.render 方法，再次渲染组件。

案例核心代码如下:

var LikeButton = React.createClass({

getInitialState: function() {

return {liked: false};

},

handleClick: function(event) {

this.setState({liked: !this.state.liked});

},

render: function() {

var text = this.state.liked ? '喜欢' : '不喜欢';

return (

<p onClick={this.handleClick}>

你<b>{text}</b>我。点我切换状态。

</p>

);

}

});

ReactDOM.render(

<LikeButton />,

document.getElementById('example')

);

**state工作原理**

通过调用setState(data, callback)方法，改变状态，就会触发React更新UI。大部分情况下，我们不需要提供callback函数。React会自动的帮我们更新UI。

**什么样的组件该有state**

大部分的组件应该从props属性中获取数据并渲染。但有的时候组件得到相应用户输入，同服务器交互，这些情况下会用到state。

React的官方说法是：尽可能的保持你的组件无状态化。为了实现这个目标，得保持你的状态同业务逻辑分离，并减少冗余信息，尽可能保持组件的单一职责。

React官方推荐的一种模式就是：构建几个无状态的组件用来渲染数据，在这些之上构建一个有状态的组件同用户和服务交互，数据通过props传递给无状态的组件。

**state应该包含什么样的数据**

UI交互会导致改变的数据。

**state不应包含什么样的数据**

计算过的数据

组件

从props复制的数据

state应保含最原始的数据，比如说时间，格式化应该交给展现层去做。组件应在render方法里控制。

**补充：**

React的作者认为，组件应该同关注分离，而不是同模板和展现逻辑分离。结构化标记和生成结构化标记的代码是紧密关联的，此外，展现逻辑一般都很复杂，使用模板语言会使展现变得笨重。

React解决这个问题的方式就是：直接通过JavaScript代码生成HTML和组件树，这样的话，你就可以使用JavaScript丰富的表达力去构建UI。为了使这个过程变得更简单，React创建了类似HTML的语法去构建节点树，也就是JSX了。

JSX语法是可选的，也就是说你也可以不使用，直接写JavaScript代码。看个对比例子：

JSX语法

React.render(

<div className="c-list">content</div>,

document.getElementById('example')

);

JavaScript

React.render(

React.createElement('div', {className: 'c-list'}, 'content'),

document.getElementById('example')

);

这样简单的例子，我们都能感觉到JSX更加语义化，更别说复杂的组件了。**所以强烈建议使用JSX**。