1 介绍JSX（Introducing JSX）

1.1 JSX中的嵌入式表达式（Embedding Expressions in JSX）

JSX中的javascript代码用{ }包裹。

JSX分隔成多行代码去写的时候要，建议放在括号中，避免自动分号插入的陷阱，如：

const element = ( //括号

<h1>

Hello, {formatName(user)}!

</h1>

);

1.2 JSX也是表达式（JSX is an Expression Too）

编译后，JSX表达式成为常规的javascript对象。

意味着可以在if语句和for循环中使用JSX，可以将其分配给变量，将它作为函数的参数和返回值。

1.3 在JSX中指定属性（Specifying Attributes with JSX）

可以使用引号指定字符串字面量作为属性。

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

也可以使用花括号{ } 嵌入javascript表达式作为属性。

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

当使用嵌入式javascript表达式作为属性时，不能再使用“”。

1.4 Specifying Children with JSX

JSX标签可能包含children，如：

const element = (

<div>

<h1>Hello!</h1>

<h2>Good to see you here.</h2>

</div>

);

注意：React DOM使用camelCase属性命名约定而不是HTML属性名称。例如，类在JSX中变为className，tabindex变为tabIndex。

1.5 JSX防止注入攻击（JSX Prevents Injection Attacks）

默认情况下，React DOM会在渲染之前转义嵌入在JSX中的任何值。 因此，它确保您永远不会注入任何未明确写入应用程序的内容。 在呈现之前，所有内容都将转换为字符串。 这有助于防止XSS（跨站点脚本）攻击。

1.6 JSX表示对象（JSX Represents Objects）

Babel将JSX编译成React.createElement()调用：

const element = (

<h1 className="greeting">

Hello, world!

</h1>

);

等价于=>>

const element = React.createElement(

'h1',

{className: 'greeting'},

'Hello, world!'

);

React.createElement（）执行一些检查以帮助您编写无错误代码，但基本上它会创建一个如下所示的对象：

// Note: this structure is simplified

const element = {

type: 'h1',

props: {

className: 'greeting',

children: 'Hello, world'

}

};

2 渲染元素（Rendering Elements）

2.1 渲染元素（Rendering Elements）

与浏览器DOM元素不同，React元素是简单的对象，而且便于创建。 React DOM负责更新DOM以匹配React元素。

元素是由组件构成的。

2.2 将元素渲染到DOM中（Rendering an Element into the DOM）

React element, a root DOM node传递给ReactDOM.render()

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

ReactDOM.render(

element,

document.getElementById('root')

);

2.3 更新渲染元素（Updating the Rendered Element）

React Elements是不可变的。创建元素后，您无法更改其子项或属性。 一个元素就像一个电影中的一个帧：它代表了某个时间点的UI。

function tick() {

const element = (

<div>

<h1>Hello, world!</h1>

<h2>It is {new Date().toLocaleTimeString()}.</h2>

</div>

);

ReactDOM.render(

element,

document.getElementById('root')

);

}

setInterval(tick, 1000);

实际上，大多数React应用只会调用ReactDOM.render（）一次。

2.4 React仅更新有必要更新的（React Only Updates What’s Necessary）

React DOM将元素及其子元素与上一个元素进行比较，并且仅应用DOM将更新所需的DOM更新到所需状态。

在2.3的例子中，即使我们创建了一个描述整个UI树的元素，只有内容已经改变的文本节点才被React DOM更新。

3 组件和属性（Components and Props）

组件让您将UI拆分成独立的、可重复使用的部分，并单独考虑每个部分。

在概念上，组件就像JavaScript函数。 他们接受任意输入（称为“props”），并返回描述应该在屏幕上显示的React elements。

3.1 功能性的组件和类组件（Functional and Class Components）

用javascript方式定义组件：

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

用ES6 class定义组件：

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

上述两种定义组件的方式等价。

React组件的需求：

1. 接受一个单一的“props”对象参数；

② 返回一个React element。

3.2 渲染一个组件（Rendering a Component）

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

const element = <Welcome name="Sara" />;

ReactDOM.render(

element,

document.getElementById('root')

);

上述代码中，组件渲染过程：

① We call ReactDOM.render() with the <Welcome name="Sara" /> element.

② React calls the Welcome component with {name: 'Sara'} as the props.

③ Our Welcome component returns a <h1>Hello, Sara</h1> element as the result.

④ React DOM efficiently updates the DOM to match <h1>Hello, Sara</h1>.

3.3 集成组件（Composing Components）

组件可以在其输出中引用其他组件，因此在React应用程序中，通常都表示为组件。

利用组件集成概念，可以使用像Button这样的小组件自下而上，逐渐运行到视图层次结构的顶部。

注意：组件必须返回单个根节点。

3.4 提取组件（Extracting Components）

我们建议从组件自己的角度来命名道具，而不是使用组件的上下文。

一个很好的经验法则是，如果您的UI的一部分被使用了几次（按钮，面板，头像）或者足够复杂（App，FeedStory，Comment），那么作为可重用组件 。

3.5 Props是只读的（Props are Read-Only）

React is pretty flexible but it has a single strict rule:

All React components must act like **pure functions** with respect to their props.

**pure functions**: 不要尝试修改输入props；并且对于相同的输入，有相同的输出。

例如：

// pure functions

function sum(a, b) {

return a + b;

}

// impure functions

function withdraw(account, amount) {

account.total -= amount;

}

4 状态和生命周期（State and Lifecycle）

state与props是相似的，但它是私有（private）的，并且完全由组件控制。

state只适用于用ES6 class方式定义的组件。

4.1 函数式写法转为class式写法（Converting a Function to a Class）

function Clock(props){

return (

<div>

<h1>Hello, world!</h1>

<h2>It is {props.date.toLocaleTimeString()}.</h2>

</div>

);

}

转换为：

class Clock extends React.Component {

render() {

return (

<div>

<h1>Hello, world!</h1>

<h2>It is {this.props.date.toLocaleTimeString()}.</h2>

</div>

);

}

}

4.2 将局部state添加到class（Adding Local State to a Class）

class Clock extends React.Component {

**constructor(props) {**

**super(props);**

// 2) Add a class constructor that assigns the initial this.state:

**this.state = {date: new Date()};**

**}**

render() {

return (

<div>

<h1>Hello, world!</h1>

// 1) Replace this.props.date with this.state.date in the render() method:

<h2>It is {this.**state**.date.toLocaleTimeString()}.</h2>

</div>

);

}

}

ReactDOM.render(

// 3) Remove the date prop from the <Clock /> element:

<Clock />,

document.getElementById('root')

);

以上仍不能满足定时器每秒刷新一次的需求。

4.3 将生命周期方法添加到类中（Adding Lifecycle Methods to a Class）

在4.2的代码基础上：

class Clock extends React.Component {

constructor(props) {

super(props);

this.state = {date: new Date()};

}

// the Clock is rendered to the DOM for the first time，

// 在组件输出已经呈现给DOM之后，componentDidMount（）挂钩运行

**componentDidMount() {**

**this.timerID = setInterval(**

**() => this.tick(),**

**1000**

**);**

**}**

// the DOM produced by the Clock is removed

**componentWillUnmount() {**

**clearInterval(this.timerID);**

**}**

**tick() {**

**this.setState({**

**date: new Date()**

**});**

**}**

render() {

return (

<div>

<h1>Hello, world!</h1>

<h2>It is {this.state.date.toLocaleTimeString()}.</h2>

</div>

);

}

}

ReactDOM.render(

<Clock />,

document.getElementById('root')

);

如果需要存储未用于可视输出的内容，则可以使用this.state手动向类中添加其他字段。

4.4 正确使用state（Using State Correctly）

4.4.1 不要直接修改state（Do Not Modify State Directly）

// Wrong

this.state.comment = 'Hello';

// Correct

this.setState({comment: 'Hello'});

唯一可以使用this.state的地方，是在结构体中。

4.4.2 状态更新可能是异步的（State Updates May Be Asynchronous）

// Wrong

this.setState({

counter: this.state.counter + this.props.increment,

});

// Correct

// prevState 接收之前的状态 props更新应用时的prop作为第二个参数

this.setState((prevState, props) => ({

counter: prevState.counter + props.increment

}));

等价于：

// Correct

this.setState(function(prevState, props) {

return {

counter: prevState.counter + props.increment

};

});

4.4.3 （State Updates are Merged）

4.5 数据下流（The Data Flows Down）

组件的state可以向下 作为它的子组件的prop传递。

可以在有状态组件中使用无状态组件，反之亦然。

5 Handling Events

React elements和DOM事件处理是非常相似的，这儿有些语法不同：

① React事件使用驼峰命名，DOM使用小写；

② With JSX you pass a function as the event handler, rather than a string.

使用preventDefault 阻止默认行为。

当您使用ES6类定义组件时，常见的模式是将事件处理程序作为类上的方法。

// This binding is necessary to make `this` work in the callback

this.handleClick = this.handleClick.bind(this);

6 条件渲染（Conditional Rendering）

7 Lists and Keys

8 Forms

9 Lifting State Up

10 Composition vs Inheritance

11 Thinking In React