react 知识点一

react 知识点一

React 在创建可重用且有吸引力的 UI 组件的库，经常变化的数据的组件方面比较强大，侧重于将复杂页面分解为多个组件来构建可重用的用户界面。

React.js 和 Vue.js 的比较

	React	Vue
类型	JavaScript 库	JavaScript 库
跨平台开发	React Native 是一个成熟且广泛使用的本机渲染应用程序平台	Vue 的 Weex 仍在不断发展，旨在提供顺畅的开发体验
学习曲线	陡峭的学习曲线，需要深入的知识	简单的学习曲线，基于 HTML 的模板使其熟悉
可重用性	只有 CSS	最大的可重用性
性能	一样快	一样快
模型	虚拟 DOM（文档对象模型）	基于虚拟 DOM HTML 的模板
功能	可用作开发单页或移动应用程序的基础	Web 应用程序框架，能够为高级单页面应用程序提供支持
复杂性	复杂	简单
Bootstrap 应用程序	CRA（创建反应应用程序）	Vue 公司-CLI
显着特点		
与道具的单向数据绑定；有状态的组件；虚拟 DOM；生命周期方法；JSX（JavaScript XML）；超越 HTML 的架构	基于 HTML 的模板	反应 | 组件（将整个应用程序划分为小型，独立且通常可重复使用的组件）| 路由 | 集成

1. JSX

React 并没有采用将 html 标签与逻辑进行分离到不同文件这种人为地分离方式，而是通过将二者共同存放在称之为“组件”的松散耦合单元之中，来实现关注点分离。vue 是在一个 vue 中，但是严格区分不同部分的功能 这样做的好处：

1. JSX 和 UI 放在一起时，会在视觉上有辅助作用（更方便的看某块内容，而不是需要上下滑动查看）；它还可以使 React 显示更多有用的错误和警告消息。
2. JSX 可以防止注入攻击
3. JSX 可以表示对象
    1. Babel 会把 JSX 转译成一个名为 React.createElement() 函数调用。

例子：

jsx 复制代码

```jsx
// 基本应用
const name = "Josh Perez";
const element = <h1>Hello, {name}</h1>;

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

// 绑定更多属性
const _props = {
  name: "div",
  width: 20,
  color: "red",
};
const element = <div {..._props}></div>;

// 表达式返回
const getItem = (data) => {
  return (
    <div>
      <div>{name}</div>
    </div>
  );
};
```

2. 元素渲染

jsx 复制代码

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

挂载到节点上

jsx 复制代码

```jsx
<div id="root"></div>;

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

```jsx
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(element);
```

更新已渲染的元素(单向绑定)

> React 元素是不可变对象。一旦被创建，你就无法更改它的子元素或者属性。一个元素就像电影的单帧：它代表了某个特定时刻的 UI。 根据我们已有的知识，更新 UI 唯一的方式是创建一个全新的元素，并将其传入 root.render()。 在更新的时候只更新它需要更新的部分

jsx 复制代码

```jsx
// 通过不断调用*render()*函数来更新内容
const root = ReactDOM.createRoot(document.getElementById("root"));

function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  root.render(element);
}

setInterval(tick, 1000);
```

3. 组件 & Props

state 和 props 之间的区别是什么？ props（“properties” 的缩写）和 state 都是普通的 JavaScript 对象。它们都是用来保存信息的，这些信息可以控制组件的渲染输出，而它们的一个重要的不同点就是：props 是传递给组件的（类似于函数的形参），而 state 是在组件内被组件自己管理的（类似于在一个函数内声明的变量）。

函数组件与 class 组件

> props 能够让我们获取到 jsx 中标签里的属性

jsx 复制代码

```jsx
/// 函数组件
/// 该函数是一个有效的 *React* 组件，因为它接收唯一带有数据的 *“props”*（代表属性）对象与并返回一个 *React* 元素。
```

```jsx
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

/// class组件
/// 同时还可以使用 ES6 的 class 来定义组件：
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}

/// 上述两个组件在 React 里是等效的。

/// 结果
const element = <Welcome name="Sara" />;
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(element);
```

组合组件

> 组合组件可以让我们用多个组件来构建复杂的组件，从而更好的管理和维护业务

jsx 复制代码

```jsx
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}
```

提取组件

> 在写组件的时候会出现组件内容越来越大，逐渐增加维护成本，这个时候我们需要拆分成更细小的组件

```jsx
/// 提取前
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <img
          className="Avatar"
          src={props.author.avatarUrl}
          alt={props.author.name}
        />
        <div className="UserInfo-name">{props.author.name}</div>
      </div>
      <div className="Comment-text">{props.text}</div>
      <div className="Comment-date">{formatDate(props.date)}</div>
    </div>
  );
}

/// 提取后
function Avatar(props) {
  return (
    <img className="Avatar" src={props.user.avatarUrl} alt={props.user.name} />
  );
}

function UserInfo(props) {
  return (
    <div className="UserInfo">
      <Avatar user={props.user} />
      <div className="UserInfo-name">{props.user.name}</div>
    </div>
  );
}

function Comment(props) {
  return (
    <div className="Comment">
      <UserInfo user={props.author} />
      <div className="Comment-text">{props.text}</div>
      <div className="Comment-date">{formatDate(props.date)}</div>
    </div>
  );
}
```

4. State & 生命周期

使用 state 更新 UI

state 详解

jsx 复制代码

```jsx
class Clock extends React.Component {
  /// 添加一个 class 构造函数，然后在该函数中为 this.state 赋初值
  /// 将 props 传递到父类的构造函数中
  /// Class 组件应该始终使用 props 参数来调用父类的构造函数（建议不能写其他的属性）
  constructor(props) {
    super(props);
    this.state = { date: new Date() };
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

正确地使用 State

> state 更新过程不一定是同步的，出于性能考虑，React 可能会把多个 setState() 调用合并成一个调用。 因为 this.props 和 this.state 可能会异步更新，所以你不要依赖他们的值来更新下一个状态。 要解决这个问题，可以让 setState() 接收一个函数而不是一个对象。这个函数用上一个 state 作为第一个参数，将此次更新被应用时的 props 做为第二个参数

jsx 复制代码

```jsx
// Wrong
this.state.comment = 'Hello';
而是应该使用 setState():

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

jsx 复制代码

```jsx
/// 此代码可能会无法更新计数器：
// Wrong
this.setState({
```

```
  counter: this.state.counter + this.props.increment,
});

// Correct
this.setState((state, props) => ({
  counter: state.counter + props.increment,
}));

///上面使用了箭头函数，不过使用普通的函数也同样可以：

// Correct
this.setState(function (state, props) {
  return {
    counter: state.counter + props.increment,
  };
});
```

将生命周期方法添加到 Class 中

生命周期详解

jsx 复制代码

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

  componentDidMount() {
    /// do something.
  }

  componentWillUnmount() {
    /// do something.
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

5.数据是向下流动的

不管是父组件或是子组件都无法知道某个组件是有状态的还是无状态的，并且它们也并不关心它是函数组件还是 class 组件。

这就是为什么称 state 为局部的或是封装的的原因。除了拥有并设置了它的组件，其他组件都无法访问。

组件可以选择把它的 state 作为 props 向下传递到它的子组件中：

jsx 复制代码

```jsx
<FormattedDate date={this.state.date} />;
function FormattedDate(props) {
  return <h2>It is {props.date.toLocaleTimeString()}.</h2>;
}
```

6.事件处理

> React 元素的事件处理和 DOM 元素的很相似，但是有一点语法上的不同：
>
> > React 事件的命名采用小驼峰式（camelCase），而不是纯小写。 使用 JSX 语法时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。

例如：

html 复制代码

```html
/// html
<button onclick="activateLasers()">Activate Lasers</button>
/// react
<button onClick="{activateLasers}">Activate Lasers</button>
```

jsx 复制代码

```jsx
/// 在 React 中另一个不同点是你不能通过返回 false 的方式阻止默认行为。你必须显式的使用 preventDefault。例如
///html
<form onsubmit="console.log('You clicked submit.'); return false">
  <button type="submit">Submit</button>
</form>;
/// react
function Form() {
  function handleSubmit(e) {
    e.preventDefault();
```

```
    console.log("You clicked submit.");
  }

  return (
    <form onSubmit={handleSubmit}>
      <button type="submit">Submit</button>
    </form>
  );
}
/// 在这里，e 是一个合成事件。React 根据 W3C 规范来定义这些合成事件，所以你不需要担心跨浏览器的兼容性问题。R
```

如果想了解更多，请查看 SyntheticEvent 参考指南。

向事件处理程序传递参数

jsx 复制代码

```
<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>
<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>
///上述两种方式是等价的，分别通过箭头函数和 Function.prototype.bind 来实现。

///在这两种情况下，React 的事件对象 e 会被作为第二个参数传递。如果通过箭头函数的方式，事件对象必须显式的进行
```

7.条件渲染

> 通过 if 来进行条件判断

jsx 复制代码

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}

function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}
```

```jsx
ReactDOM.render(
  // Try changing to isLoggedIn={true}:
  <Greeting isLoggedIn={false} />,
  document.getElementById("root")
);
```

与运算符 &&

jsx 复制代码

```jsx
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Hello!</h1>
      {unreadMessages.length > 0 && (
        <h2>You have {unreadMessages.length} unread messages.</h2>
      )}
    </div>
  );
}

const messages = ["React", "Re: React", "Re:Re: React"];
ReactDOM.render(
  <Mailbox unreadMessages={messages} />,
  document.getElementById("root")
);
```

三目运算符

jsx 复制代码

```jsx
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      The user is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.
    </div>
  );
}

render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      {isLoggedIn
        ? <LogoutButton onClick={this.handleLogoutClick} />
        : <LoginButton onClick={this.handleLoginClick} />
```

```
      }
    </div>
  );
}
```

阻止组件渲染

在极少数情况下，你可能希望能隐藏组件，即使它已经被其他组件渲染。若要完成此操作，你可以让 render 方法直接返回 null，而不进行任何渲染。

8.列表 & Key

例子：编写一个动态的列表

jsx 复制代码

```jsx
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) => <li>{number}</li>);
<ul>{listItems}</ul>;

/// 组合一下
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) => <li>{number}</li>);
  return <ul>{listItems}</ul>;
}

const numbers = [1, 2, 3, 4, 5];
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<NumberList numbers={numbers} />);
```

注意：⚠ 当我们运行这段代码，将会看到一个警告 a key should be provided for list items 意思是当你创建一个元素时，必须包括一个特殊的 key 属性

改造如下：

jsx 复制代码

```jsx
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) => (
    <li key={number.toString()}>{number}</li>
  ));
  return <ul>{listItems}</ul>;
}
```

注意：⚠ 如果列表项目的顺序可能会变化，我们不建议使用索引来用作 key 值，因为这样做会导致性能变差，还可能引起组件状态的问题。Robin Pokorny 的深度解析使用索引作为 key 的负面影响 深入解析为什么 key 是必须的

key 值在兄弟节点之间必须唯一

key 会传递信息给 React，但不会传递给你的组件。如果你的组件中需要使用 key 属性的值，请用其他属性名显式传递这个值：

9.表单

> 在 React 里，HTML 表单元素的工作方式和其他的 DOM 元素有些不同，这是因为表单元素通常会保持一些内部的 state。例如这个纯 HTML 表单只接受一个名称：

jsx 复制代码

```jsx
<form>
  <label>
    名字:
    <input type="text" name="name" />
  </label>
  <input type="submit" value="提交" />
</form>
```

受控组件

perl 复制代码

表单元素（如<input>、 <textarea> 和 <select>）通常自己维护 state，而在 React 中，可变状态（mutable state
我们可以把两者结合起来，使 React 的 state 成为“唯一数据源”。渲染表单的 React 组件还控制着用户输入过程中表单

jsx 复制代码

```jsx
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = { value: "" };

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

  handleChange(event) {
    this.setState({ value: event.target.value });
```

```
  }

  handleSubmit(event) {
    alert("提交的名字: " + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          名字:
          <input
            type="text"
            value={this.state.value}
            onChange={this.handleChange}
          />
        </label>
        <input type="submit" value="提交" />
      </form>
    );
  }
}
```

10.状态提升

通常，多个组件需要反映相同的变化数据，这时我们建议将共享状态提升到最近的共同父组件中去。让我们看看它是如何运作的。

jsx 复制代码

```
class Calculator extends React.Component {
  constructor(props) {
    super(props);
    this.handleCelsiusChange = this.handleCelsiusChange.bind(this);
    this.handleFahrenheitChange = this.handleFahrenheitChange.bind(this);
    this.state = { temperature: "", scale: "c" };
  }

  handleCelsiusChange(temperature) {
    this.setState({ scale: "c", temperature });
  }

  handleFahrenheitChange(temperature) {
    this.setState({ scale: "f", temperature });
  }
```

```
  render() {
    const scale = this.state.scale;
    const temperature = this.state.temperature;
    const celsius =
      scale === "f" ? tryConvert(temperature, toCelsius) : temperature;
    const fahrenheit =
      scale === "c" ? tryConvert(temperature, toFahrenheit) : temperature;

    return (
      <div>
        <TemperatureInput
          scale="c"
          temperature={celsius}
          onTemperatureChange={this.handleCelsiusChange}
        />
        <TemperatureInput
          scale="f"
          temperature={fahrenheit}
          onTemperatureChange={this.handleFahrenheitChange}
        />
        <BoilingVerdict celsius={parseFloat(celsius)} />
      </div>
    );
  }
}
```

11.组合 vs 继承

> React 有十分强大的组合模式。我们推荐使用组合而非继承来实现组件间的代码重用。

包含关系

有些组件无法提前知晓它们子组件的具体内容。在 Sidebar（侧边栏）和 Dialog（对话框）等展现通用容器（box）的组件中特别容易遇到这种情况。

jsx 复制代码

```
function FancyBorder(props) {
  return (
    <div className={"FancyBorder FancyBorder-" + props.color}>
      {props.children}
    </div>
  );
}
```

或者通过 props 传入

jsx 复制代码

```jsx
function SplitPane(props) {
  return (
    <div className="SplitPane">
      <div className="SplitPane-left">{props.left}</div>
      <div className="SplitPane-right">{props.right}</div>
    </div>
  );
}

function App() {
  return <SplitPane left={<Contacts />} right={<Chat />} />;
}
```

继承 特例关系

比如 messageDialog -> Dialog 可以使用 extend 关键字, 来继承 Dialog 里的方法和属性 可以通过 props 传入特殊的内容组件或者特殊参数