

# 1.什么是 React?

[React](#) 是一个**开源前端 JavaScript 库**，用于构建用户界面，尤其是单页应用程序。它用于处理网页和移动应用程序的视图层。React 是由 Facebook 的软件工程师 Jordan Walke 创建的。在 2011 年 React 应用首次被部署到 Facebook 的信息流中，之后于 2012 年被应用到 Instagram 上。

## 2.React 的主要特点和优点是什么？

- 虚拟DOM，高效速度快
- 支持在客户端和服务端渲染。。
- 可复用/可组合的 UI 组件。
- JSX 使代码易于读写。
- 单向数据流：Flux是一个用于在JavaScript应用中创建单向数据层的[架构](#)，它随着React视图库的开发而被Facebook概念化。
- 跨浏览器兼容：虚拟DOM帮助我们解决了跨浏览器问题，它为我们提供了标准化的API，甚至在IE8中都是没问题的。

## 3.React 的局限性是什么？

1. React 只是一个视图库，而不是一个完整的框架。
2. 对于 Web 开发初学者来说，有一个学习曲线。
3. 将 React 集成到传统的 MVC 框架中需要一些额外的配置。
4. 代码复杂性随着内联模板和 JSX 的增加而增加。
5. 如果有太多的小组件可能增加项目的庞大和复杂。

## 4.与 Vue.js 相比，React 有哪些优势？

### //1.React和Vue不同点

#### 1.vue和React采用的模式不一样

React严格上只针对MVC的view层，数据流向是单项的；Vue则是MVVM模式，需要创建ViewModel，实现了数据的双向绑定

#### 2. 组件写法不一样

React推荐的做法是 `JSX + inline style`，也就是把HTML和CSS全都写进JavaScript了,即'`all in js`'; Vue推荐的做法是webpack+vue-loader的单文件组件格式，即html、css、js写在同一个文件。

#### 3. 路由和状态管理解决方案

React提供了一种称为Flux / Redux架构的创新解决方案，它代表单向数据流，是著名MVC架构的替代方案。Vue使用Vuex这个更高级架构，它集成到Vue中并提供无与伦比的体验。

#### 4. 建筑工具

React和Vue都有一个非常好的开发环境。只需很少或没有配置，您就可以创建应用程序，使您能够使用最新的实践和模板。在React中，有一个Create React App（CRA），在Vue中，它是vue-cli。

#### 5. 监听数据变化的原理不同

React组件中的state对象不可以直接修改，需要使用setState方法来更新状态，然后内部会触发forceUpdate方法更新页面(forceUpdate是否会执行会参照shouldComponentUpdate方法的返回值或者比较对象的引用地址是否一致)。Vue组件中的data可以直接修改，Vue是通过数据劫持(Object.defineProperty)+观察者模式来监听数据变化从而更新页面。

#### 6. 重新渲染和优化

当组件的状态发生变化时，React的机制会触发整个组件树的重新呈现。您可能需要使用额外的属性(shouldComponentUpdate)来避免不必要地重新渲染子组件。

但Vue提供了优化的重新渲染，Vue会跟踪每一个组件的依赖关系，不需要重新渲染整个组件树。

#### //看法

实际开发中，如果想要简单、快速、灵活的开发中小型SPA单页面应用程序，推荐使用Vue，因为它容易上手。而React适用于大规模应用程序和移动应用程序，它由Facebook团队提供支持，包含许多用例、解决方案、资源和项目。

## 5.什么是JSX?

JSX 是 ECMAScript 一个类似 XML 的语法扩展。基本上，它只是为 `React.createElement()` 函数提供语法糖，从而让我们在 JavaScript 中，使用类 HTML 模板的语法，进行页面描述。

在下面的示例中，`<h1>` 内的文本标签会作为 JavaScript 函数返回给渲染函数。

```
class App extends React.Component {
  render() {
    return(
      <div>
        <h1>{'Welcome to React world!'}</h1>
      </div>
    )
  }
}
```

以上示例 render 方法中的 JSX 将会被转换为以下内容：

```
React.createElement("div", null, React.createElement(
  "h1", null, 'Welcome to React world!'));
```

## 6.元素和组件有什么区别?

一个 *Element* 是一个简单的对象，它描述了你希望在屏幕上以 DOM 节点或其他组件的形式呈现的内容。*Elements* 在它们的属性中可以包含其他 *Elements*。创建一个 React 元素是很轻量的。一旦元素被创建后，它将不会被修改。

React Element 的对象表示如下：

```
const element = React.createElement(
  'div',
  {id: 'login-btn'},
  'Login'
)
```

上面的 `React.createElement()` 函数会返回一个对象。

```
{
  type: 'div',
  props: {
    children: 'Login',
    id: 'login-btn'
  }
}
```

最后使用 `ReactDOM.render()` 方法渲染到 DOM：

```
<div id='login-btn'>Login</div>
```

而一个组件可以用多种不同方式声明。它可以是一个含有 `render()` 方法的类。或者，在简单的情况中，它可以定义为函数。无论哪种情况，它都将 props 作为输入，并返回一个 JSX 树作为输出

## 7.如何在 React 中创建组件？

有两种可行的方法来创建一个组件：

1. **Function Components:** 这是创建组件最简单的方式。这些是纯 JavaScript 函数，接受 props 对象作为第一个参数并返回 React 元素：

```
function Greeting({ message }) {  
  return <h1>{`Hello, ${message}`}</h1>  
}
```

2. **Class Components:** 你还可以使用 ES6 类来定义组件。上面的函数组件若使用 ES6 的类可改写为：

```
class Greeting extends React.Component {  
  render() {  
    return <h1>{`Hello, ${this.props.message}`}</h1>  
  }  
}
```

通过以上任意方式创建的组件，可以这样使用：

```
<Greeting message="semlinker"/>
```

## 8.何时使用类组件和函数组件？

如果组件需要使用**状态或生命周期方法**，那么使用类组件，否则使用函数组件。

在React16.8之后，推出了hooks API，允许函数组件中使用hook来模拟类组件的状态和生命周期，所以React16.8之后，函数组件可以完成类组件的大部分功能。

## 9.什么是 Pure Components？

`React.PureComponent` 与 `React.Component` 完全相同，只是它为你处理了 `shouldComponentUpdate()` 方法。当属性或状态发生变化时，PureComponent 将对属性和状态进行浅比较(当组件的state引用地址没有改变的时候不会渲染页面)。另一方面，一般的组件不会将当前的属性和状态与新的属性和状态进行比较。因此，在默认情况下，每当调用 `shouldComponentUpdate` 时，默认返回 true，所以组件都将重新渲染。

## 10.React 的状态是什么？

State是组件的状态，用于组件保存、控制以及修改自己的状态。状态State是私有的，完全由组件控制，不可通过外部访问和修改，只能通过组件内部的 `this.setState` 来修改，修改 state 属性会导致组件的重新渲染。

```
class User extends React.Component {  
  constructor(props) {  
    super(props)
```

```
    this.state = {
      message: 'welcome to React world'
    }
  }

  render() {
    return (
      <div>
        <h1>{this.state.message}</h1>
      </div>
    )
  }
}
```

## 11.React 中的 props 是什么？

Props 是组件的输入。它们是单个值或包含一组值的对象，由于React是单向数据流，所以 props 基本上也就是从父级组件向子组件传递的数据。

## 12.状态和属性有什么区别？

state 和 props 都是普通的 JavaScript 对象。虽然它们都保存着影响渲染输出的信息，但它们在组件方面的功能不同。Props 以类似于函数参数的方式传递给组件，而状态则类似于在函数内声明变量并对它进行管理。

States vs Props

Conditions	States	Props
可从父组件接收初始值	是	是
可在父组件中改变其值	否	是
在组件内设置默认值	是	是
在组件内可改变	是	否
可作为子组件的初始值	是	是

## 13.我们为什么不能直接更新状态？

如果你尝试直接改变状态，那么组件将不会重新渲染。

```
//Wrong
this.state.message = 'Hello world'
```

正确方法应该是使用 `setState()` 方法。它调度组件状态对象的更新，内部会触发`forceUpdate`方法。当状态更改时，组件通将会重新渲染。

```
//Correct
this.setState({ message: 'Hello world' })
```

## 14.回调函数作为 `setState()` 参数的目的是什么？

setState()函数第一个参数用于修改state数据，第二个参数用于修改state数据的回调。

当 setState 完成和组件渲染后，回调函数将会被调用。由于 `setState()` 可能是异步的，回调函数用于任何后续的操作。

```
setState({ name: 'John' }, () => console.log('The name has updated and component re-rendered'))
```

## 15.什么时候组件的 props 属性默认为 true?

如果没有传递属性值，则默认为 true。此行为可用，以便与 HTML 的行为匹配。例如，下面的表达式是等价的：

```
<MyInput autocomplete />

<MyInput autocomplete={true} />
```

**注意：** 不建议使用此方法，因为它可能与 ES6 对象 shorthand 混淆（例如，`{name}`，它是`{name:name}`的缩写）

## 16.HTML 和 React 事件处理有什么区别?

1. 在 HTML 中事件名必须小写：

```
<button onclick='activateLasers() '>
```

而在 React 中它遵循 *camelCase* (驼峰) 惯例：

```
<button onClick={activateLasers}>
```

1. 在 HTML 中你可以返回 `false` 以阻止默认的行为：

```
<a href='#' onclick='console.log("The link was clicked."); return false;' />
```

而在 React 中你必须地明确地调用 `preventDefault()`：

```
function handleClick(event) {
  event.preventDefault()
  console.log('The link was clicked.')
}
```

## 17.如何在 JSX 回调中绑定方法或事件处理程序?

实现这一点有三种可能的方法：

1. **Binding in Constructor:** 在 JavaScript 类中，方法默认不被绑定。这也适用于定义为类方法的 React 事件处理程序。通常我们在构造函数中绑定它们。

```
class Component extends React.Component {
  constructor(props) {
    super(props)
    this.handleClick = this.handleClick.bind(this)
  }

  handleClick() {
    // ...
  }
}
```

## 2. 使用箭头函数声明函数:

```
handleClick = () => {
  console.log('this is:', this)
}
```

```
<button onClick={this.handleClick}>
  {'Click me'}
</button>
```

## 3. 在组件上绑定事件

```
<Component 事件={this.方法.bind(this)}></Component>
```

# 18. 如何将参数传递给事件处理程序或回调函数?

在迭代或循环期间，向事件处理程序传递额外的参数是很常见的。这可以通过箭头函数或绑定方法实现。让我们以网格中更新的用户详细信息为例：

```
<button onClick={(e) => this.updateUser(userId, e)}>Update User details</button>
<button onClick={this.updateUser.bind(this, userId)}>Update User
details</button>
```

在这两种方法中，合成参数 e 作为第二个参数传递。你需要在箭头函数中显式传递它，并使用 bind 方法自动转发它。

# 19. React 中的合成事件是什么?

React 基于浏览器的事件机制自身实现了一套事件机制，包括事件注册、事件的合成、事件冒泡、事件派发等，在 React 中这套事件机制被称之为合成事件。

与原生事件直接在元素上注册的方式不同的是，React 的合成事件不会直接绑定到目标 DOM 节点上，用事件委托机制，以队列的方式，从触发事件的组件向父组件回溯，直到 Root 节点。因此，React 组件上声明的事件最终绑定到了 Root 对象（React 17 之前是 Document）上。在 Root 节点，用一个统一的监听器去监听，这个监听器上保存着目标节点与事件对象的映射。当组件挂载或卸载时，只需在这个统一的事件监听器上插入或删除对应对象；当事件发生时（即 Root 上的事件处理函数被执行），在映射里按照冒泡或捕获的路径去组件中收集真正的事件处理函数，然后，由这个统一的事件监听器对所收集的事件逐一执行。

这样做的好处：

- 对事件进行归类，可以在事件产生的任务上包含不同的优先级

- 提供合成事件对象，抹平浏览器的兼容性差异
- 减少内存消耗，提升性能，不需要注册那么多的事件了，一种事件类型只在 Root 上注册一次
- 对开发者友好

## 20. 什么是内联条件表达式？

在 JS 中你可以使用 if 语句或三元表达式，来实现条件判断。除了这些方法之外，你还可以在 JSX 中嵌入任何表达式，方法是将它们用大括号括起来，然后再加上 JS 逻辑运算符 `&&`。

```
<h1>Hello!</h1>
{
  messages.length > 0 && !isLogin ?
    <h2>
      You have {messages.length} unread messages.
    </h2>
    :
    <h2>
      You don't have unread messages.
    </h2>
}
```

当然如果只是想判断 if，可以如下直接判断：

```
{
  isLogin && <span>Your have been login!</span>
}
```

在上面的代码中，不需要使用 `isLogin ? <span>Your have been login!</span> : null` 这样的形式\*\*

## 21. 什么是 "key" 属性，在元素数组中使用它们有什么好处？

`key` 是一个特殊的字符串属性，你在创建元素数组时需要包含它。`Keys` 帮助 React 识别哪些项已更改、添加或删除。

我们通常使用数据中的 IDs 作为 `keys`：

```
const todoItems = todos.map((todo) =>
  <li key={todo.id}>
    {todo.text}
  </li>
)
```

在渲染列表项时，如果你没有稳定的 IDs，你可能会使用 `index` 作为 `key`：

```
const todoItems = todos.map((todo, index) =>
  <li key={index}>
    {todo.text}
  </li>
)
```

**注意：**

1. 由于列表项的顺序可能发生改变，因此并不推荐使用 `indexes` 作为 `keys`。这可能会对性能产生负面影响，并可能导致组件状态出现问题。
2. 如果将列表项提取为单独的组件，则在列表组件上应用 `keys` 而不是 `li` 标签。
3. 如果在列表项中没有设置 `key` 属性，在控制台会显示警告消息。

## 22.refs 有什么用？

`ref` 用于返回对元素的引用。但在大多数情况下，应该避免使用它们。当你需要直接访问 DOM 元素或组件的实例时，它们可能非常有用。

## 23.如何创建 refs？

这里有两种方案

1. 可以使用 `React.createRef()` 来创建 `ref` 引用，也可以使用 `useRef()` 来创建 `ref` 引用

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props)
    this.myRef = React.createRef()
  }
  render() {
    return <div ref={this.myRef} />
  }
}
```

2. 你也可以使用 `ref` 回调函数的方案，而不用考虑 React 版本。例如，访问搜索栏组件中的 `input` 元素如下：

```
class SearchBar extends Component {
  constructor(props) {
    super(props);
    this.txtSearch = null;
    this.state = { term: '' };
    this.setInputSearchRef = e => {
      this.txtSearch = e;
    }
  }

  onInputChange(event) {
    this.setState({ term: this.txtSearch.value });
  }

  render() {
    return (
      <input
        value={this.state.term}
        onChange={this.onInputChange.bind(this)}
        ref={this.setInputSearchRef} />
    );
  }
}
```

**注意：** 你也可以使用内联引用回调，尽管这不是推荐的方法。



## 24. 为什么不建议使用内联引用回调或函数？

如果将 `ref` 回调定义为内联函数，则在更新期间它将会被调用两次。首先使用 `null` 值，然后再使用 `DOM` 元素。这是因为每次渲染的时候都会创建一个新的函数实例，因此 `React` 必须清除旧的 `ref` 并设置新的 `ref`。

```
class UserForm extends Component {
  handleSubmit = () => {
    console.log("Input value is: ", this.input.value)
  }

  render () {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type='text'
          ref={(input) => this.input = input} /> // Access DOM input in handle
submit
        <button type='submit'>Submit</button>
      </form>
    )
  }
}
```

但我们期望的是当组件挂载时，`ref` 回调只会被调用一次。一个快速修复的方法是使用 `ES7` 类属性语法定义函数。

```
class UserForm extends Component {
  handleSubmit = () => {
    console.log("Input value is: ", this.input.value)
  }

  setSearchInput = (input) => {
    this.input = input
  }

  render () {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type='text'
          ref={this.setSearchInput} /> // Access DOM input in handle submit
        <button type='submit'>Submit</button>
      </form>
    )
  }
}
```

## 25. 什么是 `forward refs`？

*Ref forwarding* 是一个特性，它允许一些组件获取接收到 `ref` 对象并将它进一步传递给子组件。

```
const ButtonElement = React.forwardRef((props, ref) => (  
  <button ref={ref} className="CustomButton">  
    {props.children}  
  </button>  
));  
  
// Create ref to the DOM button:  
const ref = React.createRef();  
<ButtonElement ref={ref}>{'Forward Ref'}</ButtonElement>
```

## 26.callback refs 和 findDOMNode() 哪一个的首选选项?

对于 React 组件来说，refs 会指向一个组件类的实例，所以可以调用该类定义的任何方法。如果需要访问该组件的真实 DOM，可以用 ReactDOM.findDOMNode 来找到 DOM 节点(类似于 document.getElementById)，但我们并不推荐这样做。因为这在大部分情况下都打破了封装性，而且通常都能用更清晰的办法在 React 中构建代码：

推荐的方案是：

```
class MyComponent extends Component {  
  componentDidMount() {  
    this.node.scrollToView()  
  }  
  
  render() {  
    return <div ref={node => this.node = node} />  
  }  
}
```

## 27.什么是 Virtual DOM?

Virtual DOM (VDOM) 是 Real DOM 的内存表示形式。UI 的展示形式被保存在内存中并与真实的 DOM 同步。这是在调用的渲染函数和在屏幕上显示元素之间发生的一个步骤。整个过程被称为 *reconciliation*。

Real DOM vs Virtual DOM

Real DOM	Virtual DOM
更新较慢	更新较快
可以直接更新 HTML	无法直接更新 HTML
如果元素更新，则创建新的 DOM	如果元素更新，则更新 JSX
DOM 操作非常昂贵	DOM 操作非常简单
较多的内存浪费	没有内存浪费

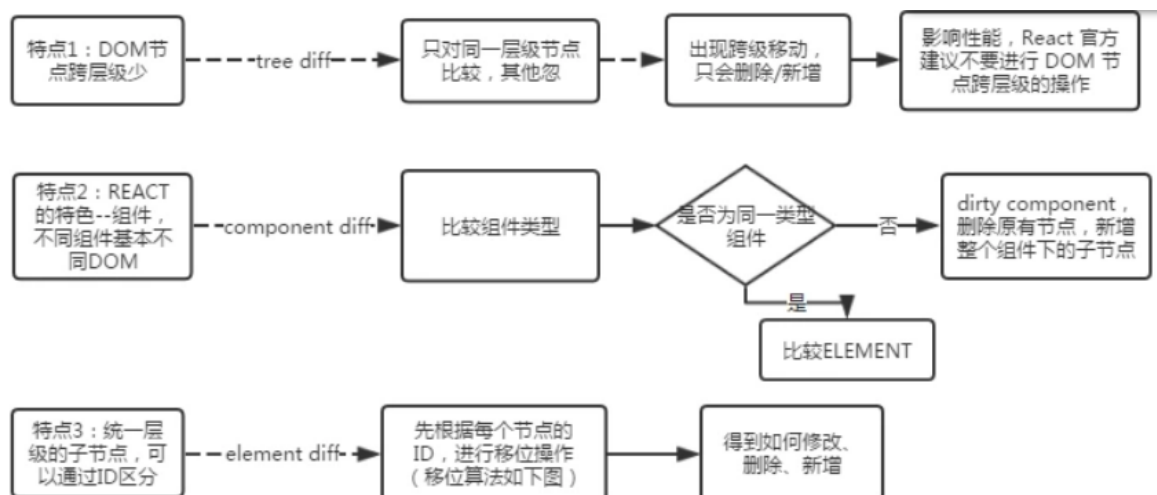
## 28.Virtual DOM 如何工作?

Virtual DOM 分为三个简单的步骤。

1. 每当任何底层数据发生更改时，整个 UI 都将以 Virtual DOM 的形式重新渲染。
2. 然后计算先前 Virtual DOM 对象和新的 Virtual DOM 对象之间的差异。
3. 一旦计算完成，真实的 DOM 将只更新实际更改的内容。

## 29.什么是差异算法?

React 需要使用算法来了解如何有效地更新 UI 以匹配最新的树。差异算法将生成将一棵树转换为另一棵树的最小操作次数。然而，算法具有  $O(n^3)$  的复杂度，其中  $n$  是树中元素的数量。在这种情况下，对于显示 1000 个元素将需要大约 10 亿个比较。这太昂贵了。相反，React 基于两个假设实现了一个复杂度为  $O(n)$  的算法



## 30.什么是调解?

当组件的 `props` 或 `state` 发生更改时，React 通过将新返回的元素与先前呈现的元素进行比较来确定是否需要实际的 DOM 更新。当它们不相等时，React 将更新 DOM。此过程称为 *reconciliation*。

## 31.当你调用setState的时候，发生了什么事?

当调用 `setState` 时，React 会做的第一件事情是将传递给 `setState` 的对象合并到组件的当前状态。这将启动一个称为和解（reconciliation）的过程。和解（reconciliation）的最终目标是以最有效的方式，根据这个新的状态来更新 UI。

为此，React 将构建一个新的 React 元素树（您可以将其视为 UI 的对象表示）。一旦有了这个树，为了弄清 UI 如何响应新的状态而改变，React 会将这个新树与上一个元素树相比较（diff）。

通过这样做，React 将会知道发生的确切变化，并且通过了解发生什么变化，只需在绝对必要的情况下进行更新即可最小化 UI 的占用空间。

## 32.shouldComponentUpdate 应该做什么

当状态改变之后，我们可以 `shouldComponentUpdate` 返回 `true` 或者 `false` 来指定页面是否需要更新

## 33.什么是 React Fiber?

Fiber 是 React v16 中新的 *reconciliation*(协调) 引擎，**优化了虚拟DOM的diff算法，同时也创建真实DOM和组件渲染拆分为无数的小分片任务**。React Fiber 的目标是提高对动画，布局，手势，暂停，中止或者重用任务的能力及为不同类型的更新分配优先级，及新的并发原语等领域的适用性。

## 34.什么是受控组件?

在随后的用户输入中，能够控制表单中输入元素的组件被称为受控组件，即每个状态更改都有一个相关的处理程序。

例如，我们使用下面的 `handleChange` 函数将输入框的值转换成大写：

```
handleChange(event) {  
  this.setState({value: event.target.value.toUpperCase()})  
}  
  
<input type="text" value={this.state.value} onChange={this.handleChange}/>
```

## 35. 什么是非受控组件?

非受控组件是在内部存储其自身状态的组件，当需要时，可以使用 `ref` 查询 DOM 并查找其当前值。这有点像传统的 HTML。

在下面的 `UserProfile` 组件中，我们通过 `ref` 引用 `name` 输入框：

```
class UserProfile extends React.Component {  
  constructor(props) {  
    super(props)  
    this.handleSubmit = this.handleSubmit.bind(this)  
    this.input = React.createRef()  
  }  
  
  handleSubmit(event) {  
    alert('A name was submitted: ' + this.input.current.value)  
    event.preventDefault()  
  }  
  
  render() {  
    return (  
      <form onSubmit={this.handleSubmit}>  
        <label>  
          {'Name: '}  
          <input type="text" ref={this.input} />  
        </label>  
        <input type="submit" value="Submit" />  
      </form>  
    );  
  }  
}
```

在大多数情况下，建议使用受控组件来实现表单。

## 36. `createElement` 和 `cloneElement` 有什么区别?

JSX 元素将被转换为 `React.createElement()` 函数来创建 React 元素，这些对象将用于表示 UI 对象。而 `cloneElement` 用于克隆元素并传递新的属性。

## 37. 在 React 中的提升状态是什么?

当多个组件需要共享相同的更改数据时，建议将共享状态提升到最接近的共同祖先。这意味着，如果两个子组件共享来自其父组件的相同数据，则将状态移动到父组件，而不是在两个子组件中维护局部状态。

可以通过高阶组件(属性继承)的方式来提升组件公共的状态。

## 38. 组件生命周期的不同阶段是什么?

组件生命周期有三个不同的生命周期阶段：

1. **Mounting:** 组件已准备好挂载到浏览器的 DOM 中. 此阶段包含来自 `constructor()`, `getDerivedStateFromProps()`, `render()`, 和 `componentDidMount()` 生命周期方法中的初始化过程。
2. **Updating:** 在此阶段, 组件以两种方式更新, 发送新的属性并使用 `setState()` 或 `forceUpdate()` 方法更新状态. 此阶段包含 `getDerivedStateFromProps()`, `shouldComponentUpdate()`, `render()`, `getSnapshotBeforeUpdate()` 和 `componentDidUpdate()` 生命周期方法。
3. **Unmounting:** 在这个最后阶段, 不需要组件, 它将从浏览器 DOM 中卸载。这个阶段包含 `componentWillUnmount()` 生命周期方法。

## 39.React 生命周期方法有哪些？

React 16.3+

- **getDerivedStateFromProps:** 在调用 `render()` 之前调用(props或者state改变的时候回调), 并在 每次渲染时调用。需要使用派生状态的情况是很罕见得。
- **componentDidMount:** 首次渲染后调用, 所有得 Ajax 请求、DOM 或状态更新、设置事件监听器都应该在此处发生。
- **shouldComponentUpdate:** 确定组件是否应该更新。默认情况下, 它返回 `true`。如果你确定在更新状态或属性后不需要渲染组件, 则可以返回 `false` 值。它是一个提高性能的好地方, 因为它允许你在组件接收新属性时阻止重新渲染。
- **getSnapshotBeforeUpdate:** 在最新的渲染输出提交给 DOM 前将会立即调用, 这对于从 DOM 捕获信息 (比如: 滚动位置) 很有用。
- **componentDidUpdate:** 它主要用于更新 DOM 以响应 prop 或 state 更改。如果 `shouldComponentUpdate()` 返回 `false`, 则不会触发。
- **componentWillUnmount** 当一个组件被从 DOM 中移除时, 该方法被调用, 取消网络请求或者移除与该组件相关的事件监听程序等应该在这里进行。

## 40.什么是上下文 (Context) ？

`Context` 通过组件树提供了一个传递数据的方法, 从而避免了在每一个层级手动的传递 `props`。比如, 需要在应用中许多组件需要访问登录用户信息、地区偏好、UI主题等。

```
//1. 创建上下文: const MyContext = React.createContext()

//2. 在根级组件通过Provider共享数据: <MyContext.Provider value={data}><Child/>
</MyContext.Provider>

//3. 在Child子组件通过Consumer来获取根级组件暴露的数据
<MyContext.Consumer>
  {(ctx)=><div>共享数据</div>}
</MyContext.Consumer>
```

## 41.children 属性是什么？

`Children` 是一个属性 (`this.props.children`), 它允许你将组件作为数据传递给其他组件, 就像你使用的任何其他组件一样。在组件的开始和结束标记之间放置的组件树将作为 `children` 属性传递给该组件。

React API 中有许多方法中提供了这个不透明数据结构的方法, 包括: `React.Children.map`、`React.Children.forEach`、`React.Children.count`、`React.Children.only`、`React.Children.toArray`。

```
const MyDiv = React.createClass({
  render: function() {
    return <div>{this.props.children}</div>
  }
})

ReactDOM.render(
  <MyDiv>
    <span>{'Hello'}</span>
    <span>{'world'}</span>
  </MyDiv>,
  node
)
```

## 42. 怎样在 React 中写注释?

React/JSX 中的注释类似于 JavaScript 的多行注释，但是是用大括号括起来。

**单行注释:**

```
<div>
  {/* 单行注释（在原生 JavaScript 中，单行注释用双斜杠（//）表示） */}
  {`welcome ${user}, let's play React`}
</div>
```

**多行注释:**

```
<div>
  {/* 多行注释超过
    一行 */}
  {`welcome ${user}, let's play React`}
</div>
```

## 43. 构造函数使用带 props 参数的目的是什么?

在调用 `super()` 方法之前，子类构造函数不能使用 `this` 引用。这同样适用于ES6子类。将 `props` 参数传递给 `super()` 的主要原因是为了在子构造函数中访问 `this.props`。

**带 props 参数:**

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props)

    console.log(this.props) // prints { name: 'John', age: 42 }
  }
}
```

## 44. 如何使用动态属性名(es6)设置 state ?

```
handleInputChange(event) {
  this.setState({ [event.target.id]: event.target.value })
}

// [event.target.id]为动态属性名。
```

## 45.为什么有组件名称要首字母大写?

如果使用 JSX 渲染组件，则该组件的名称必须以大写字母开头，否则 React 将会抛出无法识别标签的错误。这种约定是因为只有 HTML 元素和 SVG 标签可以以小写字母开头。

定义组件类的时候，你可以以小写字母开头，但在导入时应该使用大写字母。

```
class myComponent extends Component {
  render() {
    return <div />
  }
}

export default myComponent
```

当在另一个文件导入时，应该以大写字母开头：

```
import MyComponent from './MyComponent'
```

## 46.为什么 React 使用 `className` 而不是 `class` 属性?

`class` 是 JavaScript 中的关键字，而 JSX 是 JavaScript 的扩展。这就是为什么 React 使用 `className` 而不是 `class` 的主要原因。传递一个字符串作为 `className` 属性。

```
render() {
  return <span className={'menu navigation-menu'}>{'Menu'}</span>
}
```

在实际项目中，我们经常使用[classnames](#)来方便我们操作 `className`。

## 47.什么是 Fragments ?

它是 React 中的常见模式，用于组件返回多个元素。*Fragments* 可以让你聚合一个子元素列表，而无需向 DOM 添加额外节点。

```
render() {
  return (
    <React.Fragment>
      <ChildA />
      <ChildB />
      <ChildC />
    </React.Fragment>
  )
}
```

以下是简洁语法，但是在一些工具中还不支持：

```
render() {
  return (
    <>
      <ChildA />
      <ChildB />
      <ChildC />
    </>
  )
}
```

译注：React 16 以前，render 函数的返回必须有一个根节点，否则报错。

## 48.为什么使用 Fragments 比使用容器 div 更好？

1. 通过不创建额外的 DOM 节点，Fragments 更快并且使用更少的内存。这在非常大而深的节点树时很有好处。
2. 一些 CSS 机制如 *Flexbox* 和 *CSS Grid* 具有特殊的父子关系，如果在中间添加 div 将使得很难保持所需的结构。
3. 在 DOM 审查器中不会那么的杂乱。

## 49.什么是 Keyed Fragments ？

使用显式 `<React.Fragment>` 语法声明的片段可能具有 key 。一般用例是将集合映射到片段数组，如下所示

```
function Glossary(props) {
  return (
    <dl>
      {props.items.map(item => (
        // without the `key`, React will fire a key warning
        <React.Fragment key={item.id}>
          <dt>{item.term}</dt>
          <dd>{item.description}</dd>
        </React.Fragment>
      ))}
    </dl>
  );
}
```

**注意：**键是唯一可以传递给 Fragment 的属性。将来，可能会支持其他属性，例如事件处理程序。

## 50.什么是无状态组件？

如果行为独立于其状态，则它可以是无状态组件。你可以使用函数或类来创建无状态组件。但除非你需要在组件中使用生命周期钩子，否则你应该选择函数组件。无状态组件有很多好处：它们易于编写，理解和测试，速度更快，而且你可以完全避免使用 `this` 关键字。

## 51.什么是有状态组件？

如果组件的行为依赖于组件的 `state`，那么它可以被称为有状态组件。这些有状态组件总是类组件，并且具有在 `constructor` 中初始化的状态。



```
class App extends Component {
  constructor(props) {
    super(props)
    this.state = { count: 0 }
  }

  render() {
    // ...
  }
}
```

## 51.在 React 中如何校验 props 属性?

可以使用TS代替prop-types来做props类型检查。

只要给类组件/函数组件约定props的泛型即可。

## 52.在 React v16 中的错误边界是什么?

错误边界是在其子组件树中的任何位置捕获 JavaScript 错误、记录这些错误并显示回退 UI 而不是崩溃的组件树的组件。

如果一个类组件定义了一个名为 `componentDidCatch(error, info)` 或 `static getDerivedStateFromError()` 新的生命周期方法，则该类组件将成为错误边界：

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props)
    this.state = { hasError: false }
  }

  componentDidCatch(error, info) {
    // You can also log the error to an error reporting service
    logErrorToMyService(error, info)
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render will show the fallback UI.
    return { hasError: true };
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>{'Something went wrong.'}</h1>
    }
    return this.props.children
  }
}
```

之后，将其作为常规组件使用：

```
<ErrorBoundary>
  <Mywidget />
</ErrorBoundary>
```

## 53. `getDerivedStateFromError` 的目的是什么？

在子代组件抛出异常后会调用此生命周期方法。它以抛出的异常对象作为参数，并返回一个值用于更新状态。该生命周期方法的签名如下：

```
static getDerivedStateFromError(error)
```

让我们举一个包含上述生命周期方法的错误边界示例，来说明 `getDerivedStateFromError` 的目的：

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render will show the fallback UI.
    return { hasError: true };
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
  }
}
```

## 54. `react-dom` 包的用途是什么？

`react-dom` 包提供了特定的 DOM 方法，可以在应用程序的顶层使用。大多数的组件不需要使用此模块。该模块中提供的一些方法如下：

1. `render()`，react18中变成了 `createRoot`
2. `hydrate()`
3. `unmountComponentAtNode()`
4. `findDOMNode()`
5. `createPortal()`

## 55. `react-dom` 中 `render` 方法的目的是什么？

此方法用于将 React 元素渲染到所提供容器中的 DOM 结构中，并返回对组件的引用。如果 React 元素之前已被渲染到容器中，它将其执行更新，并且只在需要时改变 DOM 以反映最新的更改。

```
ReactDOM.render(element, container[, callback])
```

react18中变成了`createRoot`。

```
const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
root.render(<App/>)
```

## 56. React 和 ReactDOM 之间有什么区别？

`react` 包中包含 `React.createElement()`、`React.Component`、`React.Children`，以及与元素和组件类相关的其他帮助程序。你可以将这些视为构建组件所需的同构或通用帮助程序。`react-dom` 包中包含了 `ReactDOM.render()`，在 `react-dom/server` 包中有支持服务端渲染的 `ReactDOMServer.renderToString()` 和 `ReactDOMServer.renderToStaticMarkup()` 方法。

## 57. 为什么 ReactDOM 从 React 分离出来？

React 团队致力于将所有的与 DOM 相关的特性抽取到一个名为 ReactDOM 的独立库中。React v0.14 是第一个拆分后的版本。通过查看一些软件包，`react-native`、`react-art`、`react-canvas`，和 `react-three`，很明显，React 的优雅和本质与浏览器或 DOM 无关。为了构建更多 React 能应用的环境，React 团队计划将主要的 React 包拆分成两个：`react` 和 `react-dom`。这为编写可以在 React 和 React Native 的 Web 版本之间共享的组件铺平了道路。

## 58. 在 React 中如何使用 innerHTML？

`dangerouslySetInnerHTML` 属性是 React 用来替代在浏览器 DOM 中使用 `innerHTML`。与 `innerHTML` 一样，考虑到跨站脚本攻击（XSS），使用此属性也是有风险的。使用时，你只需传递以 `__html` 作为键，而 HTML 文本作为对应值的对象。

在本示例中 `MyComponent` 组件使用 `dangerouslySetInnerHTML` 属性来设置 HTML 标记：

```
function createMarkup() {
  return { __html: 'First &middot; Second' }
}

function MyComponent() {
  return <div dangerouslySetInnerHTML={createMarkup()} />
}
```

## 59. 如何在 React 中使用样式？

`style` 属性接受含有 camelCased（驼峰）属性的 JavaScript 对象，而不是 CSS 字符串。这与 DOM 样式中的 JavaScript 属性一致，效率更高，并且可以防止 XSS 安全漏洞。

```
const divStyle = {
  color: 'blue',
  backgroundImage: 'url(' + imgUrl + ')'
};

function HelloWorldComponent() {
  return <div style={divStyle}>Hello world!</div>
}
```

为了与在 JavaScript 中访问 DOM 节点上的属性保持一致，样式键采用了 camelcased（例如 `node.style.backgroundImage`）。

## 60.如果在构造函数中使用 `setState()` 会发生什么？

当你使用 `setState()` 时，除了设置状态对象之外，React 还会重新渲染组件及其所有的子组件。你会得到这样的错误：*Can only update a mounted or mounting component*。因此我们需要在构造函数中使用 `this.state` 初始化状态。

## 61.索引作为键的影响是什么？

Keys 应该是稳定的，可预测的和唯一的，这样 React 就能够跟踪元素。

在下面的代码片段中，每个元素的键将基于列表项的顺序，而不是绑定到即将展示的数据上。这将限制 React 能够实现的优化。

```
{todos.map((todo, index) =>
  <Todo
    {...todo}
    key={index}
  />
)}
```

假设 `todo.id` 对此列表是唯一且稳定的，如果将此数据作为唯一键，那么 React 将能够对元素进行重新排序，而无需重新创建它们。

```
{todos.map((todo) =>
  <Todo {...todo}
    key={todo.id} />
)}
```

## 62.如何 memoize（记忆）组件？

有可用于函数组件的 memoize 库。例如 `moize` 库可以将组件存储在另一个组件中。（类似Vue的 `keep-alive` 组件）

```
import moize from 'moize'
import Component from './components/Component' // this module exports a non-
memoized component

const MemoizedFoo = moize.react(Component)

const Consumer = () => {
  <div>
    {'I will memoize the following entry:'}
    <MemoizedFoo/>
  </div>
}
```

## 62.什么是 CRA 及其好处？

`create-react-app` CLI 工具允许你无需配置步骤，快速创建和运行 React 应用。

让我们使用 CRA 来创建 Todo 应用：

```
# Installation
$ npm install -g create-react-app

# Create new project
$ create-react-app todo-app
$ cd todo-app

# Build, test and run
$ npm run build
$ npm run test
$ npm start
```

它包含了构建 React 应用程序所需的一切：

1. React, JSX, ES6, 和 Flow 语法支持。
2. ES6 之外的语言附加功能，比如对象扩展运算符。
3. Autoprefixed CSS，因此你不在需要 -webkit- 或其他前缀。
4. 一个快速的交互式单元测试运行程序，内置了对覆盖率报告的支持。
5. 一个实时开发服务器，用于警告常见错误。
6. 一个构建脚本，用于打包用于生产中包含 hashes 和 sourcemaps 的 JS、CSS 和 Images 文件。

## 63.在 mounting 阶段生命周期方法的执行顺序是什么？

在创建组件的实例并将其插入到 DOM 中时，将按以下顺序调用生命周期方法。

1. `constructor()`
2. `static getDerivedStateFromProps()`
3. `render()`
4. `componentDidMount()`

## 64.在 React v16 中，哪些生命周期方法将被弃用？

以下生命周期方法将成为不安全的编码实践，并且在异步渲染方面会更有问题。

1. `componentWillMount()`
2. `componentWillReceiveProps()`
3. `componentWillUpdate()`

从 React v16.3 开始，这些方法使用 `UNSAFE_` 前缀作为别名，未加前缀的版本将在 React v17 中被移除。

## 65.生命周期方法 `getDerivedStateFromProps()` 的目的是什么？

新的静态 `getDerivedStateFromProps()` 生命周期方法在实例化组件之后以及重新渲染组件之前调用。它可以返回一个对象用于更新状态，或者返回 `null` 指示新的属性不需要任何状态更新。

他的主要作用是用来将父组件传递的 props 映射到组件的 state 中，函数返回值即是最新的 state 状态。

这个方法不常用，因为会导致组件的状态由多个源控制，逻辑不清晰的问题

```
class MyComponent extends React.Component {
  static getDerivedStateFromProps(props, state) {
    // ...
  }
}
```

## 66.生命周期方法 `getSnapshotBeforeUpdate()` 的目的是什么?

新的 `getSnapshotBeforeUpdate()` 生命周期方法在 DOM 更新之前被调用。此方法的返回值将作为第三个参数传递给 `componentDidUpdate()`。

他的主要作用是获取页面更新之前的dom信息。

```
class MyComponent extends React.Component {
  getSnapshotBeforeUpdate(prevProps, prevState) {
    // ...
  }
}
```

## 67.推荐的组件命名方法是什么?

建议通过引用命名组件，而不是使用 `displayName`。

使用 `displayName` 命名组件:

```
export default React.createClass({
  displayName: 'TodoApp',
  // ...
})
```

推荐的方式:

```
export default class TodoApp extends React.Component {
  // ...
}
```

## 68.在组件类中方法的推荐顺序是什么?

- **组件创建阶段:**

`static` 开头的 只会执行一次

`constructor` 构造器 只会执行一次

`getDerivedStateFromProps`: 当子组件接收到新的props会执行，作用：将传递的props映射到state里面 会执行多次

`render` 构建虚拟dom，但是此时虚拟dom还没有渲染到页面 会执行多次

`componentDidMount`: 组建的虚拟dom已经挂载到页面 只会执行一次

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

`getDerivedStateFromProps`: 组件将要接收到新的props属性

shouldComponentUpdate: 组件是否需要被更新，返回值是true或者false。此时可以获取最新的props和state数据

render: 重新更新渲染组件的虚拟dom

getSnapshotBeforeUpdate: 在最近一次渲染提交到 DOM 节点之前调用。它使得组件能在发生更改之前从 DOM 中捕获一些信息（例如，滚动位置），返回值将作为参数传递给componentDidUpdate

componentDidUpdate: 组件完成了更新，此时页面已经是最新的了

- **组件销毁阶段**：只执行一次

componentWillUnmount: 组件将要被销毁，此时组件还可以被使用

## 69.为什么我们需要将函数传递给 setState() 方法?

这背后的原因是 `setState()` 可能是一个异步操作。出于性能原因，React 会对状态更改进行批处理，因此在调用 `setState()` 方法之后，状态可能不会立即更改。这意味着当你调用 `setState()` 方法时，你不应该依赖当前状态，因为你不能确定当前状态应该是什么。这个问题的解决方案是将一个函数传递给 `setState()`，该函数会以上一个状态作为参数。通过这样做，你可以避免由于 `setState()` 的异步性质而导致用户在访问时获取旧状态值的问题。

假设初始计数值为零。在连续三次增加操作之后，该值将只增加一个。

```
// assuming this.state.count === 0
this.setState({ count: this.state.count + 1 })
this.setState({ count: this.state.count + 1 })
this.setState({ count: this.state.count + 1 })
// this.state.count === 1, not 3
```

如果将函数传递给 `setState()`，则 count 将正确递增。

```
this.setState((prevState, props) => ({
  count: prevState.count + props.increment
}))
// this.state.count === 3 as expected
```

## 70.你认为状态更新(state)是如何合并的?

当你在组件中调用 `setState()` 方法时，React 会将提供的对象合并到当前状态。例如，让我们以一个使用帖子和评论详细信息的作为状态变量的 Facebook 用户为例：

```
constructor(props) {
  super(props);
  this.state = {
    posts: [],
    comments: []
  };
}
```

现在，你可以独立调用 `setState()` 方法，单独更新状态变量：

```
componentDidMount() {
  fetchPosts().then(response => {
```

```

    this.setState({
      posts: response.posts
    });
  });

  fetchComments().then(response => {
    this.setState({
      comments: response.comments
    });
  });
}

```

如上面的代码段所示，`this.setState({comments})` 只会更新 `comments` 变量，而不会修改或替换 `posts` 变量。

## 71.在 React 中什么是严格模式？

`React.StrictMode` 是一个有用的组件，用于突出显示应用程序中的潜在问题。就像 `<Fragment>`，`<StrictMode>` 一样，它们不会渲染任何额外的 DOM 元素。它为其后代激活额外的检查和警告。这些检查仅适用于开发模式。

```

import React from 'react'

function ExampleApplication() {
  return (
    <div>
      <Header />
      <React.StrictMode>
        <div>
          <ComponentOne />
          <ComponentTwo />
        </div>
      </React.StrictMode>
      <Footer />
    </div>
  )
}

```

在上面的示例中，*strict mode* 检查仅应用于 `<ComponentOne>` 和 `<ComponentTwo>` 组件。

## 72.React 中支持哪些指针事件？

*Pointer Events* 提供了处理所有输入事件的统一方法。在过去，我们有一个鼠标和相应的事件监听器来处理它们，但现在我们有许多与鼠标无关的设备，比如带触摸屏的手机或笔。我们需要记住，这些事件只能在支持 *Pointer Events* 规范的浏览器中工作。

目前以下事件类型在 *React DOM* 中是可用的：

1. `onPointerDown`
2. `onPointerMove`
3. `onPointerUp`
4. `onPointerCancel`
5. `onGotPointerCapture`
6. `onLostPointerCapture`



7. `onPointerEnter`
8. `onPointerLeave`
9. `onPointerOver`
10. `onPointerOut`

## 73.在 React v16 中是否支持自定义 DOM 属性?

是的, 在过去 React 会忽略未知的 DOM 属性。如果你编写的 JSX 属性 React 无法识别, 那么 React 将跳过它。例如:

```
<div mycustomattribute={'something'} />
```

在 React 15 中将在 DOM 中渲染一个空的 div:

```
<div />
```

在 React 16 中, 任何未知的属性都将会在 DOM 显示:

```
<div mycustomattribute='something' />
```

这对于应用特定于浏览器的非标准属性, 尝试新的 DOM APIs 与集成第三方库来说非常有用。

## 74.是否可以在不调用 `setState` 方法的情况下, 强制组件重新渲染?

默认情况下, 当组件的状态或属性改变时, 组件将重新渲染。如果你的 `render()` 方法依赖于其他数据, 你可以通过调用 `forceUpdate()` 来告诉 React, 当前组件需要重新渲染。

```
component.forceUpdate(callback)
```

建议避免使用 `forceUpdate()`, 并且只在 `render()` 方法中读取 `this.props` 和 `this.state`。

## 75.在 JSX 中如何进行循环?

你只需使用带有 ES6 箭头函数语法的 `Array.prototype.map` 即可。例如, `items` 对象数组将会被映射成一个组件数组:

```
<tbody>
  {items.map(item => <SomeComponent key={item.id} name={item.name} />)}
</tbody>
```

你不能使用 `for` 循环进行迭代:

```
<tbody>
  for (let i = 0; i < items.length; i++) {
    <SomeComponent key={items[i].id} name={items[i].name} />
  }
</tbody>
```

这是因为 JSX 不能在 JSX 表达式中使用语句。

## 76.如何在 attribute 引号中访问 props 属性?

1. React (或 JSX) 不支持属性值内的变量插值。下面的形式将不起作用：

```
<img className='image' src='images/{this.props.image}' />
```

2. 但你可以将 JS 表达式作为属性值放在大括号内。所以下面的表达式是有效的：

```
<img className='image' src={'images/' + this.props.image} />
```

3. 使用模板字符串也是可以的：

```
<img className='image' src={`images/${this.props.image}`} />
```

## 77. 如何有条件地应用样式类？

你不应该在引号内使用大括号，因为它将被计算为字符串。

```
<div className="btn-panel {this.props.visible ? 'show' : 'hidden'}">
```

相反，你需要将大括号移到外部（不要忘记在类名之间添加空格）：

```
<div className={'btn-panel ' + (this.props.visible ? 'show' : 'hidden')}>
```

模板字符串也可以工作：

```
<div className={`btn-panel ${this.props.visible ? 'show' : 'hidden'}`}>
```

## 78. 如何使用 React label 元素？

如果你尝试使用标准的 `for` 属性将 `<label>` 元素绑定到文本输入框，那么在控制台将会打印缺少 HTML 属性的警告消息。

```
<label for={'user'}>{'User'}</label>
<input type={'text'} id={'user'} />
```

因为 `for` 是 JavaScript 的保留字，请使用 `htmlFor` 来替代。

```
<label htmlFor={'user'}>{'User'}</label>
<input type={'text'} id={'user'} />
```

## 79. 如何合并多个内联的样式对象？

在 React 中，你可以使用扩展运算符：

```
<button style={{...styles.panel.button, ...styles.panel.submitButton}}>
  {'Submit'}</button>
```

如果你使用的是 React Native，则可以使用数组表示法：

```
<button style={ [styles.panel.button, styles.panel.submitButton]}>{'Submit'}</button>
```

## 80.如何在调整浏览器大小时重新渲染视图?

你可以在 `componentDidMount()` 中监听 `resize` 事件, 然后更新尺寸 (`width` 和 `height`)。你应该在 `componentWillUnmount()` 方法中移除监听。

```
class WindowDimensions extends React.Component {
  componentDidMount() {
    window.addEventListener('resize', this.updateDimensions)
  }

  componentWillUnmount() {
    window.removeEventListener('resize', this.updateDimensions)
  }

  updateDimensions() {
    this.setState({width: $(window).width(), height: $(window).height()})
  }

  render() {
    return <span>{this.state.width} x {this.state.height}</span>
  }
}
```

## 81.setState() 和 replaceState() 方法之间有什么区别?

当你使用 `setState()` 时, 当前和先前的状态将被合并。`replaceState()` 会抛出当前状态, 并仅用你提供的内容替换它。通常使用 `setState()`, 除非你出于某种原因确实需要删除所有以前的键。你还可以在 `setState()` 中将状态设置为 `false/null`, 而不是使用 `replaceState()`。

## 82.如何使用 setState 防止不必要的更新?

你可以把状态的当前值与已有的值进行比较, 并决定是否重新渲染页面。如果没有更改, 你需要返回 `null` 以阻止渲染, 否则返回最新的状态值。例如, 用户配置信息组件将按以下方式实现条件渲染:

```
getUserProfile = user => {
  const latestAddress = user.address;
  this.setState(state => {
    if (state.address === latestAddress) {
      return null;
    } else {
      return { title: latestAddress };
    }
  });
};
```

## 83.react如何监听状态变化?

当状态更改时将调用以下生命周期方法。你可以将提供的状态和属性值与当前状态和属性值进行比较, 以确定是否发生了有意义的改变。

React16之后：在最新版本的React中可以使用新出的`getDerivedStateFromProps`进行props和state的监听

## 84.在 React 状态中删除数组元素的推荐方法是什么？

更好的方法是使用 `Array.prototype.filter()` 方法。

```
removeItem(index) {  
  this.setState({  
    data: this.state.data.filter((item, i) => i !== index)  
  })  
}
```

## 85.在 React 中是否可以不在页面上渲染 HTML 内容？

可以使用最新的版本 ( $\geq 16.2$ )，以下是可能的选项：

```
render() {  
  return false  
}  
  
render() {  
  return null  
}  
  
render() {  
  return []  
}  
  
render() {  
  return <React.Fragment></React.Fragment>  
}  
  
render() {  
  return <></>  
}
```

返回 `undefined` 是无效的。

## 86.如何用 React 漂亮地显示 JSON？

我们可以使用 `<pre>` 标签，以便保留 `JSON.stringify()` 的格式：

```
const data = { name: 'John', age: 42 }

class User extends React.Component {
  render() {
    return (
      <pre>
        {JSON.stringify(data, null, 2)}
      </pre>
    )
  }
}

React.render(<User />, document.getElementById('container'))
```

## 87.为什么你不能更新 React 中的 props?

React 的哲学是 props 应该是 *immutable* 和 *top-down*。这意味着父级可以向子级发送任何属性值，但子级不能修改接收到的属性。

## 88.如何在页面加载时聚焦一个输入元素?

你可以为 `input` 元素创建一个 `ref`，然后在 `componentDidMount()` 方法中使用它：

```
class App extends React.Component{
  componentDidMount() {
    this.nameInput.focus()
  }

  render() {
    return (
      <div>
        <input
          defaultValue={'won\'t focus'}
        />
        <input
          ref={(input) => this.nameInput = input}
          defaultValue={'will focus'}
        />
      </div>
    )
  }
}

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

## 89.更新状态中的对象有哪些可能的方法?

1. 用一个对象调用 `setState()` 来与状态合并：
  - 使用 `Object.assign()` 创建对象的副本：

```
const user = Object.assign({}, this.state.user, { age: 42 })
this.setState({ user })
```

- 使用扩展运算符：

```
const user = { ...this.state.user, age: 42 }
this.setState({ user })
```

2. 使用一个函数调用 `setState()`：

```
this.setState(prevState => ({
  user: {
    ...prevState.user,
    age: 42
  }
}))
```

## 90. 为什么函数比对象更适合于 `setState()`？

出于性能考虑，React 可能将多个 `setState()` 调用合并成单个更新。这是因为我们可以异步更新 `this.props` 和 `this.state`，所以不应该依赖它们的值来计算下一个状态。

以下的 counter 示例将无法按预期更新：

```
// Wrong
this.setState({
  counter: this.state.counter + this.props.increment,
})
this.setState({
  counter: this.state.counter + this.props.increment,
})
```

首选方法是使用函数而不是对象调用 `setState()`。该函数将前一个状态作为第一个参数，当前时刻的 props 作为第二个参数。

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

## 91. 我们如何在浏览器中找到当前正在运行的 React 版本？

你可以使用 `React.version` 来获取版本：

```
const REACT_VERSION = React.version

ReactDOM.render(
  <div>{`React version: ${REACT_VERSION}`}</div>,
  document.getElementById('app')
)
```

## 92. 在 `create-react-app` 项目中导入 polyfills 的方法有哪些？

一个 polyfill 是一段代码（或者插件），提供了那些开发者们希望浏览器原生提供支持的功能。我们通常的做法是先检查当前浏览器是否支持某个 API，如果不支持的话就加载对应的 polyfill，然后新旧浏览器就都可以使用这个 API 了。

## 1. 从 core-js 中手动导入:

创建一个名为 `polyfills.js` 文件, 并在根目录下的 `index.js` 文件中导入它。运行 `npm install core-js` 或 `yarn add core-js` 并导入你所需的功能特性:

```
import 'core-js/fn/array/find'
import 'core-js/fn/array/includes'
import 'core-js/fn/number/is-nan'
```

## 2. 使用 Polyfill 服务:

通过将以下内容添加到 `index.html` 中来获取自定义的特定于浏览器的 polyfill:

```
<script src='https://cdn.polyfill.io/v2/polyfill.min.js?
features=default,Array.prototype.includes'></script>
```

在上面的脚本中, 我们必须显式地请求 `Array.prototype.includes` 特性, 因为它没有被包含在默认的特性集中。

# 93. 如何在 create-react-app 中使用 https 而不是 http?

你只需要使用 `HTTPS=true` 配置。你可以编辑 `package.json` 中的 `scripts` 部分:

```
"scripts": {
  "start": "set HTTPS=true && react-scripts start"
}
```

或直接运行 `set HTTPS=true && npm start`

# 94. 如何避免在 create-react-app 中使用相对路径导入?

//方式1. 在项目的根目录中创建一个名为 `.env` 的文件, 并写入导入路径:

`NODE_PATH=src/app`

然后重新启动开发服务器。现在, 你应该能够在没有相对路径的情况下导入 `src/app` 内的任何内容。

//方式2. 使用 `react-app-rewired` 和 `customize-cra`

1) `yarn add react-app-rewired customize-cra`

2) 在项目根目录下创建一个 `config-overrides.js`

```
const path = require("path");
const {
  override,
  addWebpackAlias
} = require("customize-cra");

module.exports = override(
  addWebpackAlias({
    ["@"]: path.resolve(__dirname, "src")
  }),
);
```

3) 修改 `package.json`

```
"start": "react-app-rewired start",
"build": "react-app-rewired build",
"test": "react-app-rewired test --env=jsdom"
```

## 95.如何每秒更新一个组件?

你需要使用 `setInterval()` 来触发更改,但也需要在组件卸载时清除计时器,以防止错误和内存泄漏。

```
componentDidMount() {
  this.interval = setInterval(() => this.setState({ time: Date.now() }), 1000)
}

componentWillUnmount() {
  clearInterval(this.interval)
}
```

## 96.如何将 vendor prefixes 应用于 React 中的内联样式?

默认情况下, *Autoprefixer* 插件已经在 `create-react-app` 中被支持,但是 React 不会对内联样式应用 *vendor prefixes*, 你需要手动添加 *vendor prefixes*。

```
<div style={{
  transform: 'rotate(90deg)',
  webkitTransform: 'rotate(90deg)', // note the capital 'w' here
  msTransform: 'rotate(90deg)'      // 'ms' is the only lowercase vendor prefix
}}>
```

## 97.如何使用 React 和 ES6 导入和导出组件?

导出组件时,你应该使用默认导出:

```
import React from 'react'
import User from 'user'

export default class MyProfile extends React.Component {
  render(){
    return (
      <User type="customer">
        //...
      </User>
    )
  }
}
```

## 98.在 React 中如何定义常量?

你可以使用 ES7 的 `const` 来定义常量。

```
const user = {
  firstName: "陈亮",
  secondName: '哈哈'
}
```

## 99.在 React 中如何以编程方式触发点击事件?



你可以使用 `ref` 属性通过回调函数获取对底层的 `HTMLInputElement` 对象的引用，并将该引用存储为类属性，之后你就可以利用该引用在事件回调函数中，使用 `HTMLElement.click` 方法触发一个点击事件。这可以分为两个步骤：

1. 在 `render` 方法创建一个 `ref`：

```
<input ref={input => this.inputElement = input} />
```

2. 在事件处理器中触发点击事件

```
this.inputElement.click()
```

## 100.在 React 中是否可以使用 `async/await`?

如果要在 React 中使用 `async/await`，则需要 `Babel` 和 [transform-async-to-generator](#) 插件。

## 101.最流行的动画软件包是什么?

`React Transition Group` 和 `React Motion` 或者 `React spring` 是 React 生态系统中流行的动画包。

## 102.模块化样式文件有什么好处?

开启样式的模块化作用域，避免样式的冲突问题

## 103.什么是 React 流行的特定 linter?

ESLint 是一个流行的 JavaScript linter，可以帮我们检查 js 代码是否符合规范。在 React 中最常见的一个是为 `eslint-plugin-react` npm 包。默认情况下，它将使用规则检查许多最佳实践，检查内容从迭代器中的键到一组完整的 `prop` 类型。

## 104.如何发起 AJAX 调用以及应该在哪些组件生命周期方法中进行 AJAX 调用?

你可以使用 AJAX 库，如 `Axios`，`jQuery AJAX` 和浏览器内置的 `fetch` API。你应该在 `componentDidMount()` 生命周期方法中获取数据。这样当获取到数据的时候，你就可以使用 `setState()` 方法来更新你的组件。

例如，从 API 中获取员工列表并设置本地状态：

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      employees: [],
      error: null
    }
  }

  componentDidMount() {
    fetch('https://api.example.com/items')
      .then(res => res.json())
      .then(
        (result) => {
          this.setState({
```

```

        employees: result.employees
      })
    },
    (error) => {
      this.setState({ error })
    }
  )
}

render() {
  const { error, employees } = this.state
  if (error) {
    return <div>Error: {error.message}</div>;
  } else {
    return (
      <ul>
        {employees.map(item => (
          <li key={employee.name}>
            {employee.name}-{employees.experience}
          </li>
        ))}
      </ul>
    )
  }
}
}

```

## 105.在 React Router v6 中的 `<Router>` 组件是什么?

React Router v4 提供了以下三种类型的 `<Router>` 组件:

1. `<BrowserRouter>`
2. `<HashRouter>`
3. `<MemoryRouter>`

以上组件将创建 *browser*, *hash* 和 *memory* 的 history 实例。React Router v4 通过 `router` 对象中的上下文使与您的路由器关联的 `history` 实例的属性和方法可用。

## 106.history 中的 `push()` 和 `replace()` 方法的目的是什么?

一个 history 实例有两种导航方法:

1. `push()`
2. `replace()`

如果您将 history 视为一个访问位置的数组, 则 `push()` 将向数组添加一个新位置, `replace()` 将用新的位置替换数组中的当前位置。

## 107.如何使用在 React Router v6 中以编程的方式进行导航?

可以使用 `useNavigate` 这个 hook 来实现程式化导航

## 108.如何在 React Router v6 中获取查询字符串参数?

可以使用 `useSearchParams` 这个 hook, 要获取动态路由参数可以使用 `useParams` 这个 hook

## 109.如何实现默认页面或 404 页面?

使用 `<Route path="/" element={<NotFound />} />`

```
<Route path="home" element={<Home></Home>}></Route>
<Route path="about" element={<About></About>}></Route>
{/**动态路由参数 */}
<Route path="movie/:name/:id" element={<Movie></Movie>}></Route>
<Route path="/" element={<NotFound />} />
```

## 110.登录后如何执行自动重定向？

`react-router` 包在 React Router 中提供了 `<Navigate>` 组件。渲染 `<Navigate>` 将导航到新位置。与服务器端重定向一样，新位置将覆盖历史堆栈中的当前位置。

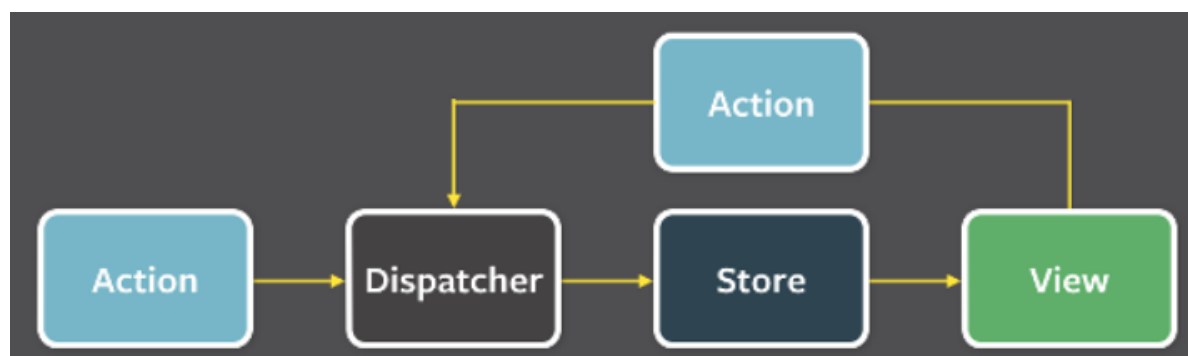
```
import React, { Component } from 'react'
import { Navigate } from 'react-router-dom'

export default class LoginComponent extends Component {
  render() {
    if (this.state.isLoggedIn === true) {
      return <Navigate to="/home" replace={true} />
    } else {
      return <div>{'Login Please'}</div>
    }
  }
}
```

## 111.什么是 Flux？

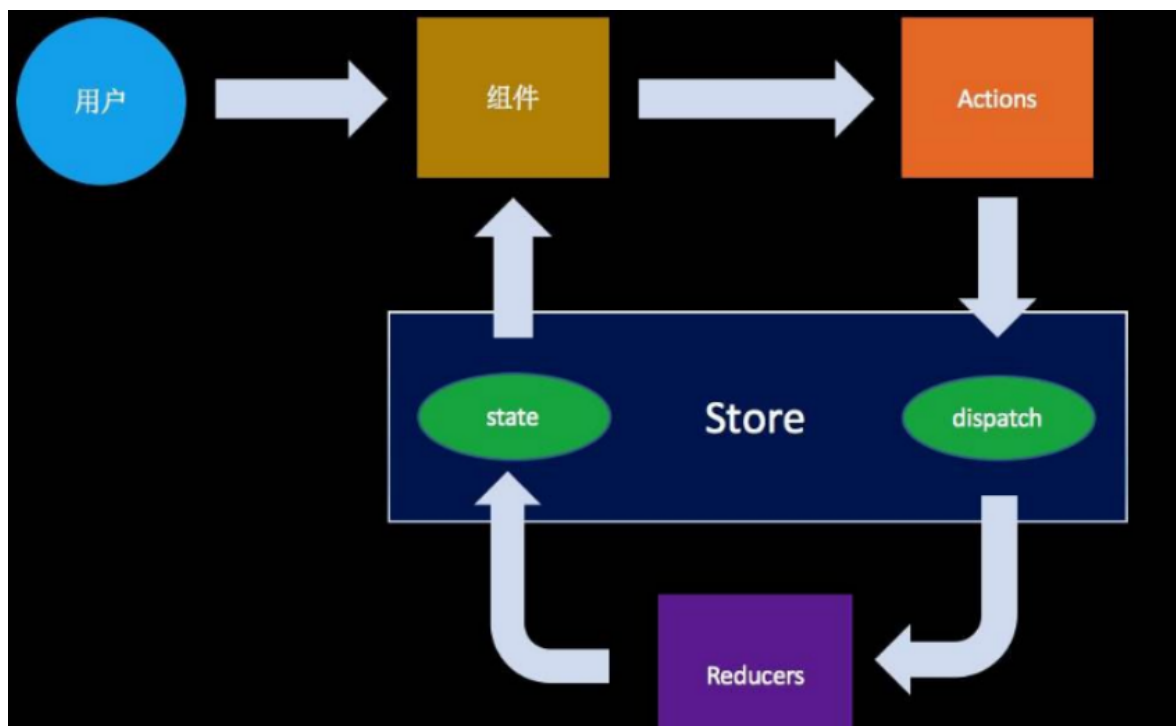
*Flux* 是应用程序设计范例，用于替代更传统的 MVC 模式。它不是一个框架或库，而是一种新的体系结构，它补充了 React 和单向数据流的概念。在使用 React 时，Facebook 会在内部使用此模式。

在 dispatcher, stores 和视图组件具有如下不同的输入和输出：



## 112.什么是 Redux？

*Redux* 是基于 *Flux* 设计模式的 JavaScript 应用程序的可预测状态容器。Redux 可以与 React 一起使用，也可以与任何其他视图库一起使用。它很小（约2kB）并且没有依赖性。



## 113.Redux 的核心原则是什么？？

Redux 遵循三个基本原则：

1. **单一数据来源：** 整个应用程序的状态存储在单个对象树中。单状态树可以更容易地跟踪随时间的变化并调试或检查应用程序。
2. **状态是只读的：** 改变状态的唯一方法是发出一个动作，一个描述发生的事情的对象。这可以确保视图和网络请求都不会直接写入状态。
3. **使用纯函数进行更改：** 要指定状态树如何通过操作进行转换，您可以编写reducers。Reducers 只是纯函数，它将先前的状态和操作作为参数，并返回下一个状态。

## 114.Flux 和 Redux 之间有什么区别？

以下是 Flux 和 Redux 之间的主要区别

Flux	Redux
状态是可变的	状态是不可变的
Store 包含状态和更改逻辑	存储和更改逻辑是分开的
存在多个 Store	仅存在一个 Store
所有的 Store 都是断开连接的	带有分层 reducers 的 Store
它有一个单独的 dispatcher	没有 dispatcher 的概念
React 组件监测 Store	容器组件使用连接函数

## 115.如何在组件外部访问 Redux 存储的对象？

使用 redux toolkit，创建slice

```
//使用RTK来构建store
import {configureStore, createSlice} from "@reduxjs/toolkit";
```

```

// createSlice 创建reducer的切片
// 它需要一个配置对象作为参数，通过对象的不同的属性来指定它的配置
const stuSlice = createSlice({
  name:'stu', // 用来自动生成action中的type
  initialState:{
    name:'孙悟空',
    age:18,
    gender:'男',
    address:'花果山'
  }, // state的初始值
  reducers:{ // 指定state的各种操作，直接在对象中添加方法
    setName(state, action){
      // 可以通过不同的方法来指定对state的不同操作
      // 两个参数: state 这个state的是一个代理对象，可以直接修改
      state.name = action.payload;
    },
    setAge(state, action){
      state.age = action.payload;
    }
  }
});

// 切片对象会自动的帮助我们生成action
// actions中存储的是slice自动生成action创建器（函数），调用函数后会自动创建action对象
// action对象的结构 {type:name/函数名, payload:函数的参数}
export const {setName, setAge} = stuSlice.actions;

// configureStore 用来创建store对象，需要一个配置对象作为参数
const store = configureStore({
  reducer:{
    student:stuSlice.reducer
  }
});

export default store;

```

index.jsx中通过Provide暴露store

```

import ReactDOM from "react-dom/client";
import App from "./App";
import {Provider} from "react-redux";
import store from './store';

const root = ReactDOM.createRoot(document.getElementById('root')!);
root.render(
  <Provider store={store}>
    <App/>
  </Provider>
);

```

组件中通过useDispatch, useSelector来获取dispatch和state

```

import React from "react";
import { useDispatch, useSelector } from "react-redux";
import { setName, setAge } from './store';

```

```

const App = () => {
  // useSelector() 用来加载state中的数据
  const student = useSelector((state: any) => state.student);
  // 通过useDispatch()来获取派发器对象
  const dispatch = useDispatch();
  // 获取action的构建器

  const setNameHandler = () => {
    dispatch(setName("沙和尚"));
  };

  const setAgeHandler = () => {
    dispatch(setAge(33));
  };

  return (
    <div>
      <p>
        {student.name} ---
        {student.age} ---
        {student.gender} ---
        {student.address}
      </p>
      <button onClick={setNameHandler}>修改name</button>
      <button onClick={setAgeHandler}>修改age</button>
    </div>
  );
};

export default App;

```

## 116.Redux 中的 Action 是什么？

*Actions*是纯 JavaScript 对象或信息的有效负载，可将数据从您的应用程序发送到您的 Store。它们是 Store 唯一的数据来源。Action 必须具有指示正在执行的操作类型的 type 属性。

例如，表示添加新待办事项的示例操作：

```

{
  type: ADD_TODO,
  text: 'Add todo item'
}

```

## 117.如何在 Redux 中发起 AJAX 请求？

在redux tool kit中可以使用 **createAsyncThunk**方法可以创建一个**异步的action**，这个方法被执行的时候会有三个（ pending(进行中) fulfilled(成功) rejected(失败)）状态。

```

// studentslice.js
// 它需要一个配置对象作为参数，通过对象的不同的属性来指定它的配置
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";

export const getStuDate = createAsyncThunk('stu/getStu',
  async () => {
    let result = await new Promise((resolve, reject) => {
      setTimeout(() => {

```

```

        resolve({
          name: "猪八戒",
          age: 108,
          gender: '男',
          address: '天安门'
        })
      }, 2000);
    })
    return result
  }
);

const stuSlice = createSlice({
  name: 'stu', // 用来自动生成action中的type
  initialState: {
    .....
  }, // state的初始值
  reducers: { // 指定state的各种操作，直接在对象中添加方法
    .....
  },
  // extraReducers 字段让 slice 处理在别处定义的 actions,
  // 包括由 createAsyncThunk 或其他slice生成的actions。
  extraReducers(builder) {
    builder
      .addCase(getStuDate.pending, (state) => {
        console.log("🚀 ~ 进行中!")
      })
      .addCase(getStuDate.fulfilled, (state, { payload }) => {
        console.log("🚀 ~ fulfilled", payload);
        state.name = payload.name
        state.address = payload.address
        state.age = payload.age
        state.gender = payload.gender
      })
      .addCase(getStuDate.rejected, (state, err) => {
        console.log("🚀 ~ rejected", err)
      });
  },
});

```

然后在组件中通过dispatch触发createAsyncThunk创建的异步action

```

import { getStuDate } from "../store/stuSlice";

<button onClick={() => dispatch(getStuDate() as any)}>
  获取学生数据--异步
</button>

```

## 118.我需要将所有状态保存到 Redux 中吗？我应该使用 react 的内部状态吗？

这取决于开发者的决定。一般情况下，组件UI的状态控制的数据我们可以放在组件内部声明。而如果有多个组件要共享的数据、该数据有多个派生数据、该数据需要缓存等情况我们可以放在redux中

## 119.什么是 redux-saga？

`redux-saga` 是一个库，旨在解决 React/Redux 项目中异步问题（数据获取等异步操作和访问浏览器缓存等可能产生副作用的动作）更容易，更好。

这个包在 NPM 上有发布：

```
$ npm install --save redux-saga
```

## 120.在 `redux-saga` 中 `call()` 和 `put()` 之间有什么区别？

`put`：用于触发 `action`

```
yield put({ type: 'todos/add', payload: 'Learn Dva' });
```

`call`：用于调用异步逻辑，支持 `Promise`。

```
const result = yield call(fetch, '/todos');
```

这个 `call` 与 JS 的 `call` 用法大概一致，这个 `call` 的第一个参数是你调用的函数，第二个参数开始是你传递的参数，可一一传递。

```
effects: {
  // 访问接口获取数据 并且保存数据
  *fetchUserList({ payload: { page = 1 } }, { call, put }) {
    const { data } = yield call(queryUserList, { page });
    yield put({
      type: 'save',
      payload: {
        list: data.list,
        total: parseInt(data.total, 10),
        page: parseInt(data.page, 10)
      }
    });
  },
},
```

## 121.什么是 `Redux Thunk`？

`redux-thunk` 是一个 `redux` 的中间件，用来处理 `redux` 中的复杂逻辑，比如异步请求；

`redux-thunk` 中间件可以让 `action` 创建函数先不返回一个 `action` 对象，而是返回一个函数；

## 122. `redux-saga` 和 `redux-thunk` 之间有什么区别？

`Redux Thunk` 和 `Redux Saga` 都负责处理副作用。在大多数场景中，`Thunk` 使用 `Promises` 来处理它们，而 `Saga` 使用 `Generators`。`Thunk` 易于使用，因为许多开发人员都熟悉 `Promise`，`Sagas/Generators` 功能更强大，但您需要学习它们。但是这两个中间件可以共存，所以你可以从 `Thunks` 开始，并在需要时引入 `Sagas`。

## 122.如何向 `Redux` 添加多个中间件？

你可以使用 `applyMiddleware()`。

例如，你可以添加 `redux-thunk` 和 `logger` 作为参数传递给 `applyMiddleware()`：



```
import { createStore, applyMiddleware } from 'redux'
const createStoreWithMiddleware = applyMiddleware(ReduxThunk, logger)
(createStore)
```

## 123.在 React 中如何使用 Font Awesome 图标?

接下来的步骤将在 React 中引入 Font Awesome:

1. 安装 `font-awesome`:

```
$ npm install --save font-awesome
```

1. 在 `index.js` 文件中导入 `font-awesome`:

```
import 'font-awesome/css/font-awesome.min.css'
```

1. 在 `className` 中添加 Font Awesome 类:

```
render() {
  return <div><i className='fa fa-spinner' /></div>
}
```

## 124.什么是 React 开发者工具?

*React Developer Tools* 允许您检查组件层次结构, 包括组件属性和状态。它既可以作为浏览器扩展 (用于 Chrome 和 Firefox), 也可以作为独立的应用程序 (用于其他环境, 包括 Safari、IE 和 React Native)。

## 125.我可以导入一个 SVG 文件作为 React 组件么?

你可以直接将 SVG 作为组件导入, 而不是将其作为文件加载。此功能仅在 `react-scripts@2.0.0` 及更高版本中可用。

```
import { ReactComponent as Logo } from './logo.svg'

const App = () => (
  <div>
    { /* Logo is an actual react component */ }
    <Logo />
  </div>
)
```

## 126.在 React 中 registerServiceWorker 的用途是什么?

默认情况下, React 会为你创建一个没有任何配置的服务 worker。Service worker 是一个 Web API, 它帮助你缓存资源和其他文件, 以便当用户离线或在弱网络时, 他/她仍然可以在屏幕上看到结果, 因此, 它可以帮助你建立更好的用户体验, 这是你目前应该了解的关于 Service worker 的内容。

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import registerServiceWorker from './registerServiceWorker';

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

## 127.React lazy 函数是什么?

使用 `React.lazy` 函数允许你将动态导入的组件作为常规组件进行渲染。当组件开始渲染时，它会自动加载包含 `OtherComponent` 的包。它必须返回一个 `Promise`，该 `Promise` 解析后为一个带有默认导出 `React` 组件的模块。

```
const OtherComponent = React.lazy(() => import('./OtherComponent'));

function MyComponent() {
  return (
    <div>
      <OtherComponent />
    </div>
  );
}
```

## 128.什么是 suspense 组件?

如果父组件在渲染时包含 `dynamic import` 的模块尚未加载完成，在此加载过程中，你必须使用一个 `loading` 指示器显示后备内容。这可以使用 **Suspense** 组件来实现。例如，下面的代码使用 `Suspense` 组件：

```
const OtherComponent = React.lazy(() => import('./OtherComponent'));

function MyComponent() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <OtherComponent />
      </Suspense>
    </div>
  );
}
```

正如上面的代码中所展示的，懒加载的组件被包装在 `Suspense` 组件中。

## 129.如何在 React 16 版本中渲染数组、字符串和数值?

**Arrays:** 与旧版本不同的是，在 `React 16` 中你不需要确保 `render` 方法必须返回单个元素。通过返回数组，你可以返回多个没有包装元素的同级元素。例如，让我们看看下面的开发人员列表：

```
const ReactJSDevs = () => {
  return [
    <li key="1">John</li>,
    <li key="2">Jackie</li>,
    <li key="3">Jordan</li>
  ];
}
```

你还可以将此数组项合并到另一个数组组件中：

```
const JSDevs = () => {
  return (
    <ul>
      <li>Brad</li>
      <li>Brodge</li>
      <ReactJSDevs/>
      <li>Brandon</li>
    </ul>
  );
}
```

**Strings and Numbers:** 在 render 方法中，你也可以返回字符串和数值类型：

```
// String
render() {
  return 'Welcome to ReactJS questions';
}
// Number
render() {
  return 2018;
}
```

## 130. 什么是 hooks?

Hooks 是一个新的草案，它允许你在不编写类的情况下使用状态和其他 React 特性。让我们来看一个 useState 钩子示例：

```
import { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

## 131. Hooks 需要遵循什么规则?

为了使用 hooks，你需要遵守两个规则：

1. 仅在顶层的 React 函数调用 hooks。也就是说，你不能在循环、条件或内嵌函数中调用 hooks。这将确保每次组件渲染时都以相同的顺序调用 hooks，并且它会在多个 useState 和 useEffect 调用之间保留 hooks 的状态。
2. 仅在 React 函数中调用 hooks。例如，你不能在常规的 JavaScript 函数中调用 hooks。

## 132.如何确保hooks遵循正确的使用规则？

React 团队发布了一个名为 **eslint-plugin-react-hooks** 的 ESLint 插件，它实施了这两个规则。您可以使用以下命令将此插件添加到项目中，

```
npm install eslint-plugin-react-hooks@next
```

并在您的 ESLint 配置文件中应用以下配置：

```
// Your ESLint configuration
{
  "plugins": [
    // ...
    "react-hooks"
  ],
  "rules": {
    // ...
    "react-hooks/rules-of-hooks": "error"
  }
}
```

**注意：** 此插件在 Create React App 已经默认配置。

## 133.是否必须为 React 组件定义构造函数？

这不是强制的。也就是说，如果你不需要初始化状态且不需要绑定方法，则你不需要为 React 组件实现一个构造函数。

## 134.什么是默认属性？

defaultProps 被定义为组件类上的属性，用于设置组件类默认的属性值。它只适用于 undefined 的属性，而不适用于 null 属性。例如，让我们为按钮组件创建默认的 color 属性：

```
class MyButton extends React.Component {
  // ...
}

MyButton.defaultProps = {
  color: 'red'
};
```

如果未设置 props.color，则会使用默认值 `red`。也就是说，每当你试图访问 color 属性时，它都使用默认值。

```
render() {
  return <MyButton /> ; // props.color will be set to red
}
```

**注意：**如果你提供的是 null 值，它会仍然保留 null 值。

## 135.为什么不能在 componentWillMount 中调用 setState() 方法？

不应在 componentWillMount() 中调用 setState()，因为一旦卸载了组件实例，就永远不会再次装载它。

## 136.支持 React 应用程序的浏览器有哪一些？

React 支持所有流行的浏览器，包括 Internet Explorer 9 和更高版本，但旧版本的浏览器（如 IE 9 和 IE 10）需要一些 polyfill。

IE8 本来不在支持的计划中，如果你发现你的app能用，那是运气好。因为 React 依赖了最基本的 ES5 实现，而这个却是 IE8 不具备的。所以你只能通过[es5-shim and es5-sham](#)这类库，先把缺的 ES5 方法补气了，才有可能在 IE8 下运行

## 137.React 支持所有的 HTML 属性么？

从 React 16 开始，完全支持标准或自定义 DOM 属性。由于 React 组件通常同时使用自定义和与 DOM 相关的属性，因此 React 与 DOM API 一样都使用 camelCase 约定。让我们对标准 HTML 属性采取一些措施：

```
<div tabIndex="-1" />      // Just like node.tabIndex DOM API
<div className="Button" /> // Just like node.className DOM API
<input readOnly={true} />  // Just like node.readOnly DOM API
```

除了特殊情况外，这些属性的工作方式与相应的 HTML 属性类似。它还支持所有 SVG 属性。

## 138.如何将事件处理程序传递给组件？

可以将事件处理程序和其他函数作为属性传递给子组件。它可以在子组件中使用，如下所示：

```
//父组件
<Child func={this.handleClick}></Child>

//子组件Child
<button onClick={() => this.props.func()}>点我</button>
```

## 139.在渲染方法中使用箭头函数好么？

是的，你可以用。它通常是向回调函数传递参数的最简单方法。但在使用时需要优化性能。

```
class Foo extends Component {
  handleClick() {
    console.log('Click happened');
  }
  render() {
    return <button onClick={() => this.handleClick()}>Click Me</button>;
  }
}
```

**注意：**组件每次渲染时，在 render 方法中的箭头函数都会创建一个新的函数，这可能会影响性能。

## 138.如何防止函数被多次调用？

如果你使用一个事件处理程序，如 `onClick` 或 `onScroll` 并希望防止回调被过快地触发，那么你可以限制回调的执行速度。

这可以通过以下可能的方式实现：

1. **Throttling**: 基于时间的频率进行更改。例如，它可以使用 `lodash` 的 `_throttle` 函数。
2. **Debouncing**: 在一段时间不活动后发布更改。例如，可以使用 `lodash` 的 `_debounce` 函数。
3. **RequestAnimationFrame throttling**: 基于 `requestAnimationFrame` 的更改。例如，可以使用 `raf-schd`。

注意：`_debounce`，`_throttle` 和 `raf-schd` 都提供了一个 `cancel` 方法来取消延迟回调。所以需要调用 `componentWillUnmount`，或者对代码进行检查来保证在延迟函数有效期内组件始终挂载。

## 139.JSX 如何防止注入攻击？

React DOM 会在渲染 JSX 中嵌入的任何值之前对其进行转义。因此，它确保你永远不能注入任何未在应用程序中显式写入的内容。

```
const name = response.potentiallyMaliciousInput;  
const element = <h1>{name}</h1>;
```

这样可以防止应用程序中的XSS（跨站点脚本）攻击。

## 140.如何更新已渲染的元素？

通过将新创建的元素传递给 ReactDOM 的 `render` 方法，可以实现 UI 更新。例如，让我们举一个滴答时钟的例子，它通过多次调用 `render` 方法来更新时间：

```
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);
```

## 141.如何防止组件渲染？

你可以基于特定的条件通过返回 `null` 值来阻止组件的渲染。这样它就可以有条件地渲染组件。

```
function Greeting(props) {
  if (!props.loggedIn) {
    return null;
  }

  return (
    <div className="greeting">
      welcome, {props.name}
    </div>
  );
}
```

```
class User extends React.Component {
  constructor(props) {
    super(props);
    this.state = {loggedIn: false, name: 'John'};
  }

  render() {
    return (
      <div>
        //Prevent component render if it is not loggedIn
        <Greeting loggedIn={this.state.loggedIn} />
        <UserDetails name={this.state.name}>
      </div>
    );
  }
}
```

在上面的示例中，greeting 组件通过应用条件并返回空值跳过其渲染部分。

## 142.keys 是否需要全局唯一？

数组中使用的键在其同级中应该是唯一的，但它们不需要是全局唯一的。也就是说，你可以在两个不同的数组中使用相同的键。例如，下面的 book 组件在不同的组件中使用相同的数组：

```
function Book(props) {
  const index = (
    <ul>
      {props.pages.map((page) =>
        <li key={page.id}>
          {page.title}
        </li>
      )}
    </ul>
  );
  const content = props.pages.map((page) =>
    <div key={page.id}>
      <h3>{page.title}</h3>
      <p>{page.content}</p>
      <p>{page.pageNumber}</p>
    </div>
  );
  return (
    <div>
      {index}
      <hr />
    </div>
  );
}
```

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

## 143.用于表单处理的流行选择是什么？

Formik 是一个用于 React 的表单库，它提供验证、跟踪访问字段和处理表单提交等解决方案。具体来说，你可以按以下方式对它们进行分类：

1. 获取表单状态输入和输出的值。
2. 表单验证和错误消息。
3. 处理表单提交。

它用于创建一个具有最小 API 的可伸缩、性能良好的表单助手，以解决令人讨厌的问题。

## 144.什么是高阶组件（HOC）？

高阶组件既不会修改原组件，也不会使用继承复制原组件的行为。相反，高阶组件是通过将原组件包裹（wrapping）在容器组件（container component）里面的方式来组合（composes）使用原组件。高阶组件就是一个没有副作用的纯函数。

高阶组件的创建方式有两种：

1. 属性代理(props proxy)。属性组件通过被包裹的 React 组件来操作 props。

```
const MyContainer = (WrappedComponent) => {
  return class extends Component {
    render() {
      return (
        <WrappedComponent
          {...props}
        />
      )
    }
  }
}

export default MyContainer;
```

2. 反向继承(inheritance inversion)。高阶组件继承于被包裹的 React 组件。

```
function iihoc(WrappedComponent) {
  return class Enhancer extends WrappedComponent {
    render() {
      return super.render()
    }
  }
}
```

[https://blog.csdn.net/weixin\\_34150830/article/details/88019752](https://blog.csdn.net/weixin_34150830/article/details/88019752)

## 145.HOC 有哪些限制？



除了它的好处之外，高阶组件还有一些注意事项。以下列出的几个注意事项：

1. **不要在渲染方法中使用HOC：** 建议不要将 HOC 应用于组件的 render 方法中的组件。

```
render() {  
  // A new version of EnhancedComponent is created on every render  
  // EnhancedComponent1 !== EnhancedComponent2  
  const EnhancedComponent = enhance(MyComponent);  
  // That causes the entire subtree to unmount/remount each time!  
  return <EnhancedComponent />;  
}
```

上述代码通过重新装载，将导致该组件及其所有子组件状态丢失，会影响到性能。正确的做法应该是在组件定义之外应用 HOC，以便仅生成一次生成的组件

2. **静态方法必须复制：** 将 HOC 应用于组件时，新组件不具有原始组件的任何静态方法

```
// Define a static method  
wrappedComponent.staticMethod = function() { /*...*/}  
// Now apply a HOC  
const EnhancedComponent = enhance(wrappedComponent);  
  
// The enhanced component has no static method  
typeof EnhancedComponent.staticMethod === 'undefined' // true
```

您可以通过在返回之前将方法复制到输入组件上来解决此问题

```
function enhance(wrappedComponent) {  
  class Enhance extends React.Component { /*...*/}  
  // Must know exactly which method(s) to copy :(  
  Enhance.staticMethod = wrappedComponent.staticMethod;  
  return Enhance;  
}
```

3. **Refs 不会被往下传递** 对于HOC，您需要将所有属性传递给包装组件，但这对于 refs 不起作用。这是因为 ref 并不是一个类似于 key 的属性。在这种情况下，您需要使用 React.forwardRef API。

## 146.在 HOCs 中 forward ref 的目的是什么？

因为 ref 不是一个属性，所以 Refs 不会被传递。就像 **key** 一样，React 会以不同的方式处理它。如果你将 ref 添加到 HOC，则该 ref 将引用最外层的容器组件，而不是包装的组件。在这种情况下，你可以使用 Forward Ref API。例如，你可以使用 React.forwardRef API 显式地将 refs 转发的内部的 FancyButton 组件。

以下的 HOC 会记录所有的 props 变化：

```
function logProps(Component) {  
  class LogProps extends React.Component {  
    componentDidUpdate(prevProps) {  
      console.log('old props:', prevProps);  
      console.log('new props:', this.props);  
    }  
  }  
}
```

```

render() {
  const {forwardedRef, ...rest} = this.props;

  // Assign the custom prop "forwardedRef" as a ref
  return <Component ref={forwardedRef} {...rest} />;
}

return React.forwardRef((props, ref) => {
  return <LogProps {...props} forwardedRef={ref} />;
});
}

```

让我们使用这个 HOC 来记录所有传递到我们“fancy button”组件的属性：

```

class FancyButton extends React.Component {
  focus() {
    // ...
  }

  // ...
}

export default logProps(FancyButton);

```

现在让我们创建一个 ref 并将其传递给 FancyButton 组件。在这种情况下，你可以聚焦到 button 元素上。

```

import FancyButton from './FancyButton';

const ref = React.createRef();
ref.current.focus();

<FancyButton
  label="Click Me"
  handleClick={handleClick}
  ref={ref}
/>;

```

## 147.ref 参数对于所有函数或类组件是否可用？

对于类组件而言，我们可以使用 ref 来引用类组件，获取类组件实例

对于函数组件而言，当在函数组件上使用 ref 的时候，只有在使用 `React.forwardRef` 定义组件时，才存在第二个 ref 参数。我们无法通过 ref 引用函数组件对象，因为函数组件没有对象，只能通过 ref 转发的方式将父组件的 ref 转发给子组件。

## 148.如何设置非受控组件的默认值？

在 React 中，表单元素的属性值将覆盖其 DOM 中的值。对于非受控组件，你可能希望能够指定其初始值，但不会控制后续的更新。要处理这种情形，你可以指定一个 **defaultValue** 属性来取代 **value** 属性。

```

render() {
  return (
    <form onSubmit={this.handleSubmit}>

```

```

    <label>
      User Name:
      <input
        defaultValue="John"
        type="text"
        ref={this.input} />
    </label>
    <input type="submit" value="Submit" />
  </form>
);
}

```

这同样适用于 `select` 和 `textArea` 输入框。但对于 `checkbox` 和 `radio` 控件，需要使用 `defaultChecked`。

## 149.你最喜欢的 React 技术栈是什么？

主要使用 `redux` 和 `redux saga` 进行状态管理和具有副作用的异步操作，使用 `react-router` 进行路由管理，使用 `styled-components` 库开发 React 组件，使用 `axios` 调用 REST api，使用 `react-spring` 做动画，使用 `formik+yup` 做表单，以及其他支持的技术栈，如 `webpack`、`reseselect`、`esnext`、`babel` 等。

还使用 `umi` 来构建项目，内置了很多插件和插件集，开箱即用，比如 `useRequest`

## 150.如何为 React 应用程序添加 bootstrap？

Bootstrap 可以通过三种可能的方式添加到 React 应用程序中：

1. 使用 Bootstrap CDN: 这是添加 bootstrap 最简单的方式。在 `head` 标签中添加 bootstrap 相应的 CSS 和 JS 资源。
2. 把 Bootstrap 作为依赖项：如果你使用的是构建工具或模块绑定器（如 `Webpack`），那么这是向 React 应用程序添加 bootstrap 的首选选项。

```
npm install bootstrap
```

3. 使用 React Bootstrap 包: 在这种情况下，你可以将 Bootstrap 添加到我们的 React 应用程序中，方法是使用一个以 React 组件形式对 Bootstrap 组件进行包装后包。下面的包在此类别中很流行：

1. `react-bootstrap`
2. `reactstrap`

## 151.是否建议在 React 中使用 CSS In JS 技术？

React 对如何定义样式没有任何意见，但如果你是初学者，那么好的起点是像往常一样在单独的 `*.css` 文件中定义样式，并使用类名引用它们。此功能不是 React 的一部分，而是来自第三方库。但是如果你想尝试不同的方法（JS 中的 CSS），那么 `styled-components` 库是一个不错的选择。

## 152.我需要用 hooks 重写所有类组件吗？

不需要。但你可以在某些组件（或新组件）中尝试使用 `hooks`，而无需重写任何已存在的代码。因为在 `ReactJS` 中目前没有移除 `classes` 的计划。

## 153.如何使用 React Hooks 获取数据？

名为 `useEffect` 的 effect hook 可用于使用 `axios` 从 API 中获取数据，并使用 `useState` 钩子提供的更新函数设置组件本地状态中的数据。让我们举一个例子，它从 API 中获取 react 文章列表。

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

function App() {
  const [data, setData] = useState({ hits: [] });

  useEffect(async () => {
    const result = await axios(
      'http://hn.algolia.com/api/v1/search?query=react',
    );

    setData(result.data);
  }, []);

  return (
    <ul>
      {data.hits.map(item => (
        <li key={item.objectID}>
          <a href={item.url}>{item.title}</a>
        </li>
      ))}
    </ul>
  );
}

export default App;
```

记住，我们为 effect hook 提供了一个空数组作为第二个参数，以避免在组件更新时再次激活它，它只会在组件挂载时被执行。比如，示例中仅在组件挂载时获取数据。

## 154.Hooks 是否涵盖了类的所有用例？

Hooks 并没有涵盖类的所有用例，但是有计划很快添加它们。目前，还没有与不常见的 `getSnapshotBeforeUpdate` 和 `componentDidCatch` 生命周期等效的钩子。

## 155.在 React 中什么是 Portal ？

*Portal* 提供了一种很好的将子节点渲染到父组件以外的 DOM 节点的方式。

```
ReactDOM.createPortal(child, container)
```

第一个参数是任何可渲染的 React 子节点，例如元素，字符串或片段。第二个参数是 DOM 元素。

## 156.portals 的典型使用场景是什么？

当父组件拥有 `overflow: hidden` 或含有影响堆叠上下文的属性（z-index、position、opacity 等样式），且需要脱离它的容器进行展示时，React portal 就非常有用。例如，对话框、全局消息通知、悬停卡和工具提示。

## 157.useEffect和useLayoutEffect有什么区别

#### useEffect:

基本上90%的情况下,都应该用这个,这个是在render结束后,你的callback函数执行,但是不会block browser painting,算是某种异步的方式吧,但是class的componentDidMount 和 componentDidUpdate是同步的,在render结束后就运行,useEffect在大部分场景下都比class的方式性能更好。

#### useLayoutEffect:

这个是用在处理DOM的时候,当你的useEffect里面的操作需要处理DOM,并且会改变页面的样式,就需要用这个,否则可能会出现出现闪屏问题, useLayoutEffect里面的callback函数会在DOM更新完成后立即执行,但是会在浏览器进行任何绘制之前运行完成,阻塞了浏览器的绘制。

## 158.使用webpack打包React项目,怎么减小生成的js大小?

- 1: 分离出业务代码和第三方库
- 2: 分离css
- 3: 清除打包后的文件中的注释, 和copyright信息
- 4: 引入的React切换到产品版本

<https://www.meiwen.com.cn/subject/wjdnattx.html>

## 159.你最不喜欢React的哪一个特性 (说一个就好)

React它不是一个框架, 它只是MVC (模型 - 视图 - 控制器) 中的view

—所以如果是大型项目想要一套完整的框架的话, 基本都需要加上ReactRouter和Flux才能写大型应用。

React-Native还没有提供1.0正式版, 坑多, 打包出来的项目体积大, 性能较原生App有显著差异。

## 160.immutable的原理是什么?

Immutable 实现的原理是 Persistent Data Structure (持久化数据结构), 也就是使用旧数据创建新数据时, 要保证旧数据同时可用且不变。同时为了避免 deepCopy 把所有节点都复制一遍带来的性能损耗, Immutable 使用了 Structural Sharing (结构共享), 即如果对象树中一个节点发生变化, 只修改这个节点和受它影响的父节点, 其它节点则进行共享。Immutable 则提供了简洁高效的判断数据是否变化的方法, 只需 === 和 is 比较就能知道是否需要执行 render(), 而这个操作几乎 0 成本, 所以可以极大提高性能

## 161.react如何提高组件的渲染效率呢?

- 子组件执行 shouldComponentUpdate 方法, 自行决定是否更新
- 给列表中的组件添加key属性
- 使用PureComponent进行浅比较
- 使用memo高阶组件来提高函数组件的渲染效率
- 使用useMemo和useCallback来缓存变量和函数
- 使用useRef替代useState, 绕过capture value特性

## 162.写出React动态改变class切换组件样式

可以通过一个变量来控制样式的切换, 当点击该变量的时候改变变量值为true或者false即可:

```
<p className={this.state.display?"active":"active1"}>你是我的唯一</p>
```

## 163.create-react-app创建新运用怎么解决卡的问题?

切换npm和yarn的源

## 164.在使用React过程中你都踩过哪些坑？

普通函数点击事件中的this是undefined

label标签要使用htmlFor

img标签指定src属性的时候要require() 图片

指定class要用className

组件名字首字母要大写

create-react-app脚手架中配置less要安装react-app-rewired和customize-cra和 babel-plugin-import，并且在项目根目录下创建config-overrides.js，还需要修改package.json中的script命令才行

配置react-native环境的时候，package server使用8081端口，而8081被java.exe占用的问题

。。。

## 165.React-router怎么获取历史对象和URL的参数？

使用useLocation、useParams和useSearchParams这些hook

## 166.React-Router的实现原理是什么？

底层原理：借助了h5 API中的pushState、replaceState以及location.hash location.replace来实现的

1. 页面跳转实现

BrowserHistory: pushState、replaceState;

HashHistory: location.hash、location.replace

2. 浏览器回退

BrowserHistory: popstate;

HashHistory: hashchang;

## 167.React-Router怎样在路由变化时重新渲染同一个组件？

可以使用useEffect(()=>{},location)来监听路由地址变化，每当路由地址变化重新发送请求获取最新state，然后修改state重新渲染组件

1. [高阶组件Hoc](#)

2. [redux+react-redux最佳实践](#)

3. [hook](#)

4. [vue和react比较中](#)

5. [vue和react比较上](#)

6. [react 8种优化方式](#)

7. [react优化方向](#)

- 8.[react-router](#)
- 9.[react开发34个技巧](#)
- 10.[hook](#)
- 11.[react优化](#)
- 12.[react面试题](#)
13. <https://mp.weixin.qq.com/s/kEEExKE-Chr40z-mCc6AZ7Q>
14. <https://mp.weixin.qq.com/s/hvdaknxlCmbFEti0lrj8Ag>
15. <https://mp.weixin.qq.com/s/a3l7ZyAHMISPicjmexW-Yw>
16. [https://mp.weixin.qq.com/s/myWdeq1Lq\\_b3vNgApcVB0A](https://mp.weixin.qq.com/s/myWdeq1Lq_b3vNgApcVB0A)
- 17.[https://mp.weixin.qq.com/s/XU9LCZ\\_CWrAzjiD04tMA-Q](https://mp.weixin.qq.com/s/XU9LCZ_CWrAzjiD04tMA-Q)
- 18.[https://mp.weixin.qq.com/s/sonaMf34\\_ZZHKbaGZqz8yg](https://mp.weixin.qq.com/s/sonaMf34_ZZHKbaGZqz8yg)

- 1.[webpack](#)
- 2.[轻量化部署](#)
- 3.[面试总结](#)
- 4.[vue12道高频原理面试题](#)
- 5.[vue钩子](#)
- 6.[keepAlive条件缓存](#)
- 7.[vue性能优化](#)
- 8.[js](#)
- 9.[算法1](#)
- 10.[算法2](#)
- 11.[Set和Map](#)
- 12.[Iterator](#)
13. 闭包(<https://mp.weixin.qq.com/s/fWS0zkeQgQrtxp6563WjCw>)