**React 面试题**

以下是面试官最有可能问到的 **50 个 React 面试题和答案**。为方便你学习，我对它们进行了分类：

* 基本知识
* React 组件
* React Redux
* React 路由

**基本知识**

**1. 区分Real DOM和Virtual DOM**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Real DOM** | **Virtual DOM** | | --- | --- | | 1. 更新缓慢。 | 1. 更新更快。 | | 2. 可以直接更新 HTML。 | 2. 无法直接更新 HTML。 | | 3. 如果元素更新，则创建新DOM。 | 3. 如果元素更新，则更新 JSX 。 | | 4. DOM操作代价很高。 | 4. DOM 操作非常简单。 | | 5. 消耗的内存较多。 | 5. 很少的内存消耗。 | |  |

**2. 什么是React？**

* React 是 Facebook 在 2011 年开发的前端 JavaScript 库。
* 它遵循基于组件的方法，有助于构建可重用的UI组件。
* 它用于开发复杂和交互式的 Web 和移动 UI。
* 尽管它仅在 2015 年开源，但有一个很大的支持社区。

**3. React有什么特点？**

React的主要功能如下：

1. 它使用**虚拟DOM**而不是真正的DOM。
2. 它可以进行**服务器端渲染**。
3. 它遵循**单向数据流**或数据绑定。

**4. 列出React的一些主要优点。**

React的一些主要优点是：

1. 它提高了应用的性能
2. 可以方便地在客户端和服务器端使用
3. 由于 JSX，代码的可读性很好
4. React 很容易与 Meteor，Angular 等其他框架集成
5. 使用React，编写UI测试用例变得非常容易

**5. React有哪些限制？**

React的限制如下：

1. React 只是一个库，而不是一个完整的框架
2. 它的库非常庞大，需要时间来理解
3. 新手程序员可能很难理解
4. 编码变得复杂，因为它使用内联模板和 JSX

**6. 什么是JSX？**

JSX 是J avaScript XML 的简写。是 React 使用的一种文件，它利用 JavaScript 的表现力和类似 HTML 的模板语法。这使得 HTML 文件非常容易理解。此文件能使应用非常可靠，并能够提高其性能。下面是JSX的一个例子：

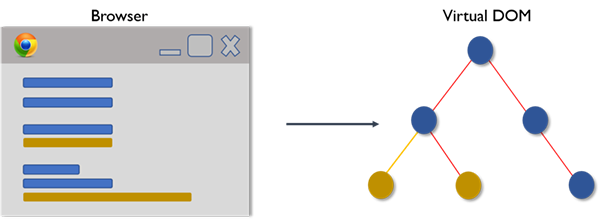
1. render(){
2. **return**(
3. <div>
4. <h1> Hello World **from** Edureka!!</h1>
5. </div>
6. );
7. }

**7. 你了解 Virtual DOM 吗？解释一下它的工作原理。**

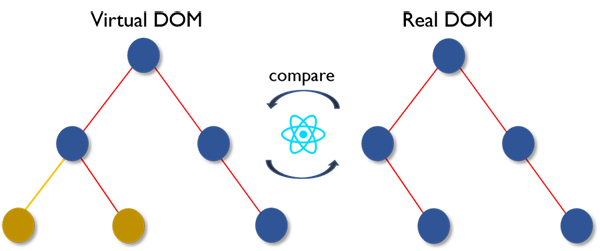
Virtual DOM 是一个轻量级的 JavaScript 对象，它最初只是 real DOM 的副本。它是一个节点树，它将元素、它们的属性和内容作为对象及其属性。 React 的渲染函数从 React 组件中创建一个节点树。然后它响应数据模型中的变化来更新该树，该变化是由用户或系统完成的各种动作引起的。

Virtual DOM 工作过程有三个简单的步骤。

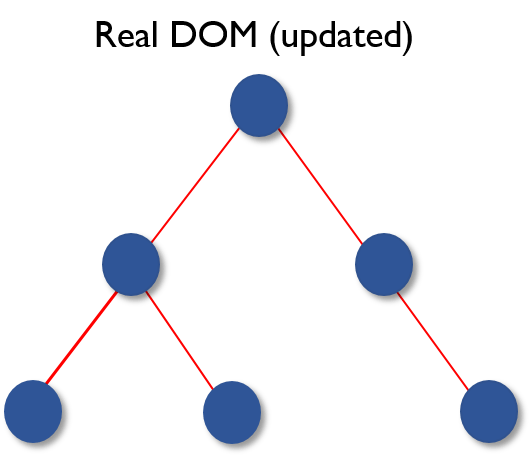
    1.每当底层数据发生改变时，整个 UI 都将在 Virtual DOM 描述中重新渲染。

[](https://s3.51cto.com/oss/201903/23/792297f720f2b45e8aaa79e82275b811.png)

    2.然后计算之前 DOM 表示与新表示的之间的差异。

[](https://s1.51cto.com/oss/201903/23/aaad68d656719e96ebacf366ef09092a.png)

    3.完成计算后，将只用实际更改的内容更新 real DOM。

[](https://s2.51cto.com/oss/201903/23/6a3532f3c8502e135c7733cba6f9658d.png)

**8. 为什么浏览器无法读取JSX？**

浏览器只能处理 JavaScript 对象，而不能读取常规 JavaScript 对象中的 JSX。所以为了使浏览器能够读取 JSX，首先，需要用像 Babel 这样的 JSX 转换器将 JSX 文件转换为 JavaScript 对象，然后再将其传给浏览器。

**9. 与 ES5 相比，React 的 ES6 语法有何不同？**

以下语法是 ES5 与 ES6 中的区别：

1.require 与 import

1. // ES5
2. var React = require('react');
4. // ES6
5. import React **from** 'react';

2.export 与 exports

1. // ES5
2. module.exports = Component;
4. // ES6
5. export **default** Component;

3.component 和 function

1. // ES5
2. var MyComponent = React.createClass({
3. render: **function**() {
4. **return**
5. <h3>Hello Edureka!</h3>;
6. }
7. });
9. // ES6
10. class MyComponent extends React.Component {
11. render() {
12. **return**
13. <h3>Hello Edureka!</h3>;
14. }
15. }

4.props

1. // ES5
2. var App = React.createClass({
3. propTypes: { **name**: React.PropTypes.string },
4. render: **function**() {
5. **return**
6. <h3>Hello, {this.props.**name**}!</h3>;
7. }
8. });
10. // ES6
11. class App extends React.Component {
12. render() {
13. **return**
14. <h3>Hello, {this.props.**name**}!</h3>;
15. }
16. }

5.state

1. // ES5
2. var App = React.createClass({
3. getInitialState: **function**() {
4. **return** { **name**: 'world' };
5. },
6. render: **function**() {
7. **return**
8. <h3>Hello, {this.state.**name**}!</h3>;
9. }
10. });
12. // ES6
13. class App extends React.Component {
14. constructor() {
15. super();
16. this.state = { **name**: 'world' };
17. }
18. render() {
19. **return**
20. <h3>Hello, {this.state.**name**}!</h3>;
21. }
22. }

**10. React与Angular有何不同？**

| **主题** | **React** | **Angular** |
| --- | --- | --- |
| 1. 体系结构 | 只有 MVC 中的 View | 完整的 MVC |
| 2. 渲染 | 可以进行服务器端渲染 | 客户端渲染 |
| 3. DOM | 使用 virtual DOM | 使用 real DOM |
| 4. 数据绑定 | 单向数据绑定 | 双向数据绑定 |
| 5. 调试 | 编译时调试 | 运行时调试 |
| 6. 作者 | Facebook | Google |

**React 组件**

**11. 你怎样理解“在React中，一切都是组件”这句话。**

组件是 React 应用 UI 的构建块。这些组件将整个 UI 分成小的独立并可重用的部分。每个组件彼此独立，而不会影响 UI 的其余部分。

**12. 怎样解释 React 中 render() 的目的。**

每个React组件强制要求必须有一个 **render()**。它返回一个 React 元素，是原生 DOM 组件的表示。如果需要渲染多个 HTML 元素，则必须将它们组合在一个封闭标记内，例如 <form>、<group>、<div> 等。此函数必须保持纯净，即必须每次调用时都返回相同的结果。

**13. 如何将两个或多个组件嵌入到一个组件中？**

可以通过以下方式将组件嵌入到一个组件中：

1. class MyComponent extends React.Component{
2. render(){
3. **return**(
4. <div>
5. <h1>Hello</h1>
6. <Header/>
7. </div>
8. );
9. }
10. }
11. class Header extends React.Component{
12. render(){
13. **return**
14. <h1>Header Component</h1>
15. };
16. }
17. ReactDOM.render(
18. <MyComponent/>, document.getElementById('content')
19. );

**14. 什么是 Props?**

Props 是 React 中属性的简写。它们是只读组件，必须保持纯，即不可变。它们总是在整个应用中从父组件传递到子组件。子组件永远不能将 prop 送回父组件。这有助于维护单向数据流，通常用于呈现动态生成的数据。

**15. React中的状态是什么？它是如何使用的？**

状态是 React 组件的核心，是数据的来源，必须尽可能简单。基本上状态是确定组件呈现和行为的对象。与props 不同，它们是可变的，并创建动态和交互式组件。可以通过 this.state() 访问它们。

**16. 区分状态和 props**

| **条件** | **State** | **Props** |
| --- | --- | --- |
| 1. 从父组件中接收初始值 | Yes | Yes |
| 2. 父组件可以改变值 | No | Yes |
| 3. 在组件中设置默认值 | Yes | Yes |
| 4. 在组件的内部变化 | Yes | No |
| 5. 设置子组件的初始值 | Yes | Yes |
| 6. 在子组件的内部更改 | No | Yes |

**17. 如何更新组件的状态？**

可以用 this.setState()更新组件的状态。

1. class MyComponent extends React.Component {
2. constructor() {
3. super();
4. this.state = {
5. **name**: 'Maxx',
6. id: '101'
7. }
8. }
9. render()
10. {
11. setTimeout(()=>{this.setState({**name**:'Jaeha', id:'222'})},2000)
12. **return** (
13. <div>
14. <h1>Hello {this.state.**name**}</h1>
15. <h2>Your Id **is** {this.state.id}</h2>
16. </div>
17. );
18. }
19. }
20. ReactDOM.render(
21. <MyComponent/>, document.getElementById('content')
22. );

**18. React 中的箭头函数是什么？怎么用？**

箭头函数（**=>**）是用于编写函数表达式的简短语法。这些函数允许正确绑定组件的上下文，因为在 ES6 中默认下不能使用自动绑定。使用高阶函数时，箭头函数非常有用。

1. //General way
2. render() {
3. **return**(
4. <MyInput onChange = {this.handleChange.bind(this) } />
5. );
6. }
7. //**With** Arrow **Function**
8. render() {
9. **return**(
10. <MyInput onChange = { (e)=>this.handleOnChange(e) } />
11. );
12. }

**19. 区分有状态和无状态组件。**

| **有状态组件** | **无状态组件** |
| --- | --- |
| 1. 在内存中存储有关组件状态变化的信息 | 1. 计算组件的内部的状态 |
| 2. 有权改变状态 | 2. 无权改变状态 |
| 3. 包含过去、现在和未来可能的状态变化情况 | 3. 不包含过去，现在和未来可能发生的状态变化情况 |
| 4. 接受无状态组件状态变化要求的通知，然后将 props 发送给他们。 | 4.从有状态组件接收 props 并将其视为回调函数。 |

**20. React组件生命周期的阶段是什么？**

React 组件的生命周期有三个不同的阶段：

1. 初始渲染阶段：这是组件即将开始其生命之旅并进入 DOM 的阶段。
2. 更新阶段：一旦组件被添加到 DOM，它只有在 prop 或状态发生变化时才可能更新和重新渲染。这些只发生在这个阶段。
3. 卸载阶段：这是组件生命周期的\*\*\*阶段，组件被销毁并从 DOM 中删除。

**21. 详细解释 React 组件的生命周期方法。**

一些最重要的生命周期方法是：

1. componentWillMount**()** – 在渲染之前执行，在客户端和服务器端都会执行。
2. componentDidMount**()** – 仅在\*\*\*次渲染后在客户端执行。
3. componentWillReceiveProps**()** – 当从父类接收到 props 并且在调用另一个渲染器之前调用。
4. shouldComponentUpdate**()** – 根据特定条件返回 true 或 false。如果你希望更新组件，请返回**true** 否则返回 **false**。默认情况下，它返回 false。
5. componentWillUpdate**()** – 在 DOM 中进行渲染之前调用。
6. componentDidUpdate**()** – 在渲染发生后立即调用。
7. componentWillUnmount**()** – 从 DOM 卸载组件后调用。用于清理内存空间。

**22. React中的事件是什么？**

在 React 中，事件是对鼠标悬停、鼠标单击、按键等特定操作的触发反应。处理这些事件类似于处理 DOM 元素中的事件。但是有一些语法差异，如：

1. 用驼峰命名法对事件命名而不是仅使用小写字母。
2. 事件作为函数而不是字符串传递。

事件参数重包含一组特定于事件的属性。每个事件类型都包含自己的属性和行为，只能通过其事件处理程序访问。

**23. 如何在React中创建一个事件？**

1. class Display extends React.Component({
2. show(evt) {
3. // code
4. },
5. render() {
6. // Render the div **with** an onClick prop (value **is** a **function**)
7. **return** (
8. <div onClick={this.show}>Click Me!</div>
9. );
10. }
11. });

**24. React中的合成事件是什么？**

合成事件是围绕浏览器原生事件充当跨浏览器包装器的对象。它们将不同浏览器的行为合并为一个 API。这样做是为了确保事件在不同浏览器中显示一致的属性。

**25. 你对 React 的 refs 有什么了解？**

Refs 是 React 中引用的简写。它是一个有助于存储对特定的 React 元素或组件的引用的属性，它将由组件渲染配置函数返回。用于对 render() 返回的特定元素或组件的引用。当需要进行 DOM 测量或向组件添加方法时，它们会派上用场。

1. class ReferenceDemo extends React.Component{
2. display() {
3. const **name** = this.inputDemo.value;
4. document.getElementById('disp').innerHTML = **name**;
5. }
6. render() {
7. **return**(
8. <div>
9. **Name**: <input type="text" ref={input => this.inputDemo = input} />
10. <button **name**="Click" onClick={this.display}>Click</button>
11. <h2>Hello <span id="disp"></span> !!!</h2>
12. </div>
13. );
14. }
15. }

**26. 列出一些应该使用 Refs 的情况。**

以下是应该使用 refs 的情况：

* 需要管理焦点、选择文本或媒体播放时
* 触发式动画
* 与第三方 DOM 库集成

**27. 如何模块化 React 中的代码？**

可以使用 export 和 import 属性来模块化代码。它们有助于在不同的文件中单独编写组件。

1. //ChildComponent.jsx
2. export **default** class ChildComponent extends React.Component {
3. render() {
4. **return**(
5. <div>
6. <h1>This **is** a child component</h1>
7. </div>
8. );
9. }
10. }
12. //ParentComponent.jsx
13. import ChildComponent **from** './childcomponent.js';
14. class ParentComponent extends React.Component {
15. render() {
16. **return**(
17. <div>
18. <App />
19. </div>
20. );
21. }
22. }

**28. 如何在 React 中创建表单**

React 表单类似于 HTML 表单。但是在 React 中，状态包含在组件的 state 属性中，并且只能通过 setState() 更新。因此元素不能直接更新它们的状态，它们的提交是由 JavaScript 函数处理的。此函数可以完全访问用户输入到表单的数据。

1. handleSubmit(event) {
2. alert('A name was submitted: ' + this.state.value);
3. event.preventDefault();
4. }
6. render() {
7. **return** (
8. <form onSubmit={this.handleSubmit}>
9. <label>
10. **Name**:
11. <input type="text" value={this.state.value} onChange={this.handleSubmit} />
12. </label>
13. <input type="submit" value="Submit" />
14. </form>
15. );
16. }

**29. 你对受控组件和非受控组件了解多少？**

| **受控组件** | **非受控组件** |
| --- | --- |
| 1. 没有维持自己的状态 | 1. 保持着自己的状态 |
| 2.数据由父组件控制 | 2.数据由 DOM 控制 |
| 3. 通过 props 获取当前值，然后通过回调通知更改 | 3. Refs 用于获取其当前值 |

**30. 什么是高阶组件（HOC）？**

高阶组件是重用组件逻辑的高级方法，是一种源于 React 的组件模式。 HOC 是自定义组件，在它之内包含另一个组件。它们可以接受子组件提供的任何动态，但不会修改或复制其输入组件中的任何行为。你可以认为 HOC 是“纯（Pure）”组件。

**31. 你能用HOC做什么？**

HOC可用于许多任务，例如：

* 代码重用，逻辑和引导抽象
* 渲染劫持
* 状态抽象和控制
* Props 控制

**32. 什么是纯组件？**

纯（Pure） 组件是可以编写的最简单、最快的组件。它们可以替换任何只有 **render()** 的组件。这些组件增强了代码的简单性和应用的性能。

**33. React 中 key 的重要性是什么？**

key 用于识别唯一的 Virtual DOM 元素及其驱动 UI 的相应数据。它们通过回收 DOM 中当前所有的元素来帮助 React 优化渲染。这些 key 必须是唯一的数字或字符串，React 只是重新排序元素而不是重新渲染它们。这可以提高应用程序的性能。

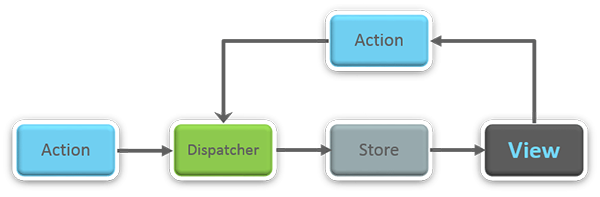
**React Redux**

**34. MVC框架的主要问题是什么？**

以下是MVC框架的一些主要问题：

* 对 DOM 操作的代价非常高
* 程序运行缓慢且效率低下
* 内存浪费严重
* 由于循环依赖性，组件模型需要围绕 models 和 views 进行创建

**35. 解释一下 Flux**

[](https://s3.51cto.com/oss/201903/23/8b485ec6f7e1abfebd6b1baed6d20378.png)

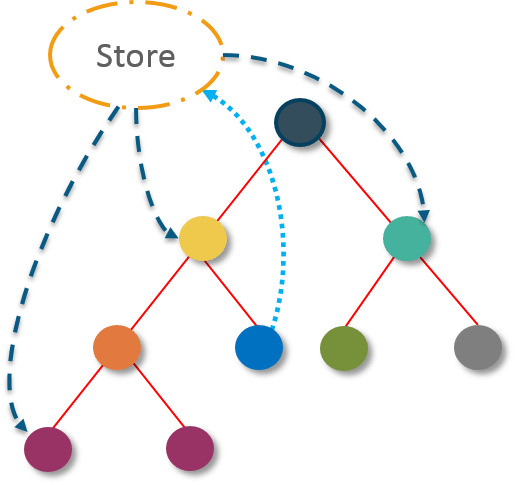
Flux 是一种强制单向数据流的架构模式。它控制派生数据，并使用具有所有数据权限的中心 store 实现多个组件之间的通信。整个应用中的数据更新必须只能在此处进行。 Flux 为应用提供稳定性并减少运行时的错误。

**36. 什么是Redux？**

Redux 是当今最热门的前端开发库之一。它是 JavaScript 程序的可预测状态容器，用于整个应用的状态管理。使用 Redux 开发的应用易于测试，可以在不同环境中运行，并显示一致的行为。

**37. Redux遵循的三个原则是什么？**

1. 单一事实来源：整个应用的状态存储在单个 store 中的对象/状态树里。单一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去触发一个动作。动作是描述变化的普通 JS 对象。就像 state 是数据的最小表示一样，该操作是对数据更改的最小表示。
3. 使用纯函数进行更改：为了指定状态树如何通过操作进行转换，你需要纯函数。纯函数是那些返回值仅取决于其参数值的函数。

[](https://s5.51cto.com/oss/201903/23/4c6f8db1c9b4e97ceb585ff7953e4f86.png)

**38. 你对“单一事实来源”有什么理解？**

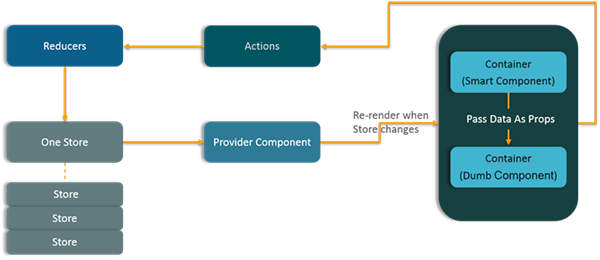
Redux 使用 “Store” 将程序的整个状态存储在同一个地方。因此所有组件的状态都存储在 Store 中，并且它们从 Store 本身接收更新。单一状态树可以更容易地跟踪随时间的变化，并调试或检查程序。

**39. 列出 Redux 的组件。**

Redux 由以下组件组成：

1. **Action** – 这是一个用来描述发生了什么事情的对象。
2. **Reducer** – 这是一个确定状态将如何变化的地方。
3. **Store** – 整个程序的状态/对象树保存在Store中。
4. **View** – 只显示 Store 提供的数据。

**40. 数据如何通过 Redux 流动？**

[](https://s1.51cto.com/oss/201903/23/95c15717d16278068de27a9ad8ccdfa5.png)

**41. 如何在 Redux 中定义 Action？**

React 中的 Action 必须具有 type 属性，该属性指示正在执行的 ACTION 的类型。必须将它们定义为字符串常量，并且还可以向其添加更多的属性。在 Redux 中，action 被名为 Action Creators 的函数所创建。以下是 Action 和Action Creator 的示例：

1. **function** addTodo(text) {
2. **return** {
3. type: ADD\_TODO,
4. text
5. }
6. }

**42. 解释 Reducer 的作用。**

Reducers 是纯函数，它规定应用程序的状态怎样因响应 ACTION 而改变。Reducers 通过接受先前的状态和 action 来工作，然后它返回一个新的状态。它根据操作的类型确定需要执行哪种更新，然后返回新的值。如果不需要完成任务，它会返回原来的状态。

**43. Store 在 Redux 中的意义是什么？**

Store 是一个 JavaScript 对象，它可以保存程序的状态，并提供一些方法来访问状态、调度操作和注册侦听器。应用程序的整个状态/对象树保存在单一存储中。因此，Redux 非常简单且是可预测的。我们可以将中间件传递到 store 来处理数据，并记录改变存储状态的各种操作。所有操作都通过 reducer 返回一个新状态。

**44. Redux与Flux有何不同？**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Flux** | **Redux** | | --- | --- | | 1. Store 包含状态和更改逻辑 | 1. Store 和更改逻辑是分开的 | | 2. 有多个 Store | 2. 只有一个 Store | | 3. 所有 Store 都互不影响且是平级的 | 3. 带有分层 reducer 的单一 Store | | 4. 有单一调度器 | 4. 没有调度器的概念 | | 5. React 组件订阅 store | 5. 容器组件是有联系的 | | 6. 状态是可变的 | 6. 状态是不可改变的 | |  |

**45. Redux 有哪些优点？**

Redux 的优点如下：

* **结果的可预测性 -** 由于总是存在一个真实来源，即 store ，因此不存在如何将当前状态与动作和应用的其他部分同步的问题。
* **可维护性 -** 代码变得更容易维护，具有可预测的结果和严格的结构。
* **服务器端渲染 -** 你只需将服务器上创建的 store 传到客户端即可。这对初始渲染非常有用，并且可以优化应用性能，从而提供更好的用户体验。
* **开发人员工具 -** 从操作到状态更改，开发人员可以实时跟踪应用中发生的所有事情。
* **社区和生态系统 -** Redux 背后有一个巨大的社区，这使得它更加迷人。一个由才华横溢的人组成的大型社区为库的改进做出了贡献，并开发了各种应用。
* **易于测试 -** Redux 的代码主要是小巧、纯粹和独立的功能。这使代码可测试且独立。
* **组织 -** Redux 准确地说明了代码的组织方式，这使得代码在团队使用时更加一致和简单。

**React 路由**

**46. 什么是React 路由？**

React 路由是一个构建在 React 之上的强大的路由库，它有助于向应用程序添加新的屏幕和流。这使 URL 与网页上显示的数据保持同步。它负责维护标准化的结构和行为，并用于开发单页 Web 应用。 React 路由有一个简单的API。

**47. 为什么React Router v4中使用 switch 关键字 ？**

虽然 <div>用于封装 Router 中的多个路由，当你想要仅显示要在多个定义的路线中呈现的单个路线时，可以使用 “switch” 关键字。使用时，<switch> 标记会按顺序将已定义的 URL 与已定义的路由进行匹配。找到\*\*\*个匹配项后，它将渲染指定的路径。从而绕过其它路线。

**48. 为什么需要 React 中的路由？**

Router 用于定义多个路由，当用户定义特定的 URL 时，如果此 URL 与 Router 内定义的任何 “路由” 的路径匹配，则用户将重定向到该特定路由。所以基本上我们需要在自己的应用中添加一个 Router 库，允许创建多个路由，每个路由都会向我们提供一个独特的视图

1. <switch>
2. <route exact path=’/’ component={Home}/>
3. <route path=’/posts/:id’ component={Newpost}/>
4. <route path=’/posts’   component={Post}/>
5. </switch>

**49. 列出 React Router 的优点。**

几个优点是：

1. 就像 React 基于组件一样，在 React Router v4 中，API 是 'All About Components'。可以将 Router 可视化为单个根组件（<BrowserRouter>），其中我们将特定的子路由（<route>）包起来。
2. 无需手动设置历史值：在 React Router v4 中，我们要做的就是将路由包装在 <BrowserRouter> 组件中。
3. 包是分开的：共有三个包，分别用于 Web、Native 和 Core。这使我们应用更加紧凑。基于类似的编码风格很容易进行切换。

**50. React Router与常规路由有何不同？**

| **主题** | **常规路由** | **React 路由** |
| --- | --- | --- |
| **参与的页面** | 每个视图对应一个新文件 | 只涉及单个HTML页面 |
| **URL 更改** | HTTP 请求被发送到服务器并且接收相应的 HTML 页面 | 仅更改历史记录属性 |
| **体验** | 用户实际在每个视图的不同页面切换 | 用户认为自己正在不同的页面间切换 |