工作不好找， 这 35 道React 面试题可以助你一波

王大冶 2020-03-10 阅读 12 分钟

> 作者：Alex
> 译者：前端小智
> 来源：dev.to

> **点赞再看，养成习惯**
> **本文 `GitHub` https://github.com/qq44924588... 上已经收录，更多往期高赞文章的分类，也整理了很多我的文档，和教程资料。欢迎Star和完善，大家面试可以参照考点复习，希望我们一起有点东西。**

问题1：什么是虚拟DOM?

> 主题: React
> 难度: ⭐

虚拟 DOM (VDOM)是真实 DOM 在内存中的表示。UI 的表示形式保存在内存中，并与实际的 DOM 同步。这是一个发生在渲染函数被调用和元素在屏幕上显示之间的步骤，整个过程被称为**调和**。

问题2：类组件和函数组件之间的区别是啥?

> 主题: React
> 难度: ⭐ ⭐

- **类组件**可以使用其他特性，如状态 `state` 和生命周期钩子。
- 当组件只是接收 `props` 渲染到页面时，就是无状态组件，就属于函数组件，也被称为哑组件或展示组件。

函数组件和类组件当然是有区别的，而且函数组件的性能比类组件的性能要高，因为类组件使用的时候要实例化，而函数组件直接执行函数取返回结果即可。为了提高性能，尽量使用函数组件。

区别	函数组件	类组件
是否有 `this`	没有	有
是否有生命周期	没有	有
是否有状态 `state`	没有	有

问题 3：React 中 refs 干嘛用的?

> 主题: React
> 难度: ⭐⭐

`Refs` 提供了一种访问在`render`方法中创建的 DOM 节点或者 React 元素的方法。在典型的数据流中，`props` 是父子组件交互的唯一方式，想要修改子组件，需要使用新的`pros`重新渲染它。凡事有例外，某些情况下咱们需要在典型数据流外，强制修改子代，这个时候可以使用 `Refs`。

咱们可以在组件添加一个 `ref` 属性来使用，该属性的值是一个回调函数，接收作为其第一个参数的底层 DOM 元素或组件的挂载实例。

```
class UnControlledForm extends Component {
  handleSubmit = () => {
    console.log("Input Value: ", this.input.value)
  }
  render () {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type='text'
          ref={(input) => this.input = input} />
        <button type='submit'>Submit</button>
      </form>
    )
  }
}
```

请注意，`input` 元素有一个`ref`属性，它的值是一个函数。该函数接收输入的实际 DOM 元素，然后将其放在实例上，这样就可以在 `handleSubmit` 函数内部访问它。

经常被误解的只有在类组件中才能使用 `refs`，但是`refs`也可以通过利用 JS 中的闭包与函数组件一起使用。

```
function CustomForm ({handleSubmit}) {
  let inputElement
  return (
    <form onSubmit={() => handleSubmit(inputElement.value)}>
      <input
        type='text'
        ref={(input) => inputElement = input} />
      <button type='submit'>Submit</button>
    </form>
  )
}
```

问题 4：在 React 中如何处理事件

> 主题: React
> 难度: ⭐ ⭐

为了解决跨浏览器的兼容性问题，`SyntheticEvent` 实例将被传递给你的事件处理函数，`SyntheticEvent`是 React 跨浏览器的浏览器原生事件包装器，它还拥有和浏览器原生事件相同的接口，包括 `stopPropagation()` 和 `preventDefault()`。

比较有趣的是，React 实际上并不将事件附加到子节点本身。React 使用单个事件侦听器侦听顶层的所有事件。这对性能有好处，也意味着 React 在更新 DOM 时不需要跟踪事件监听器。

问题 5：state 和 props 区别是啥?

> 主题: React
> 难度: ⭐ ⭐

`props`和`state`是普通的 JS 对象。虽然它们都包含影响渲染输出的信息，但是它们在组件方面的功能是不同的。即

- `state` 是组件自己管理数据，控制自己的状态，可变；
- `props` 是外部传入的数据参数，不可变；
- 没有`state`的叫做无状态组件，有`state`的叫做有状态组件；
- 多用 `props`，少用 `state`，也就是多写无状态组件。

问题 6：如何创建 refs

> 主题: React
> 难度: ⭐ ⭐

Refs 是使用 `React.createRef()` 创建的，并通过 `ref` 属性附加到 React 元素。在构造组件时，通常将 `Refs` 分配给实例属性，以便可以在整个组件中引用它们。

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

或者这样用：

```
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 in
        <button type='submit'>Submit</button>
      </form>
    )
  }
}
```

问题 7：什么是高阶组件?

> 主题: React
> 难度: ⭐⭐

高阶组件(HOC)是接受一个组件并返回一个新组件的函数。基本上，这是一个模式，是从 React 的组合特性中衍生出来的，称其为**纯组件**，因为它们可以接受任何动态提供的子组件，但不会修改或复制输入组件中的任何行为。

```
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

HOC 可以用于以下许多用例

- 代码重用、逻辑和引导抽象
- 渲染劫持
- state 抽象和操作
- props 处理

问题 8：在构造函数调用 `super` 并将 `props` 作为参数传入的作用是啥?

> 主题: React
> 难度: ⭐ ⭐

在调用 `super()` 方法之前，子类构造函数无法使用`this`引用，ES6 子类也是如此。将 `props` 参数传递给 `super()` 调用的主要原因是在子构造函数中能够通过`this.props`来获取传入的 `props`。

传递 props

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    console.log(this.props);  // { name: 'sudheer',age: 30 }
  }
}
```

没传递 props

```
class MyComponent extends React.Component {
  constructor(props) {
    super();
    console.log(this.props); // undefined
    // 但是 Props 参数仍然可用
    console.log(props); // Prints { name: 'sudheer',age: 30 }
  }
```


上面示例揭示了一点。`props` 的行为只有在构造函数中是不同的，在构造函数之外也是一样的。

问题 9：什么是控制组件?

> 主题: React
> 难度: ⭐⭐⭐

在 HTML 中，表单元素如 `<input>`、`<textarea>`和`<select>`通常维护自己的状态，并根据用户输入进行更新。当用户提交表单时，来自上述元素的值将随表单一起发送。

而 React 的工作方式则不同。包含表单的组件将跟踪其状态中的输入值，并在每次回调函数(例如`onChange`)触发时重新渲染组件，因为状态被更新。以这种方式由 React 控制其值的输入表单元素称为**受控组件**。

问题 10：如何 React.createElement ?

> 主题: React
> 难度: ⭐⭐⭐

问题:

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
)
```

上述代码如何使用 `React.createElement` 来实现:

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

问题 11：讲讲什么是 JSX ?

> 主题: React
> 难度: ⭐⭐⭐

当 **Facebook** 第一次发布 React 时，他们还引入了一种新的 JS 方言 `JSX`，将原始 HTML 模板嵌入到 JS 代码中。JSX 代码本身不能被浏览器读取，必须使用`Babel`和`webpack`等工具将其转换为传统的JS。很多开发人员就能无意识使用 JSX，因为它已经与 React 结合在一直了。

```
class MyComponent extends React.Component {
  render() {
    let props = this.props;
    return (
      <div className="my-component">
      <a href={props.url}>{props.name}</a>
      </div>
    );
  }
}
```

问题 12：根据下面定义的代码，可以找出存在的两个问题吗？

> 主题: React
> 难度: ⭐⭐⭐

请看下面的代码：

```
class MyComponent extends React.Component {
  constructor(props) {
    // set the default internal state
    this.state = {
      clicks: 0
    };
  }

  componentDidMount() {
    this.refs.myComponentDiv.addEventListener('click', this.clickHandler);
  }

  componentWillUnmount() {
    this.refs.myComponentDiv.removeEventListener('click', this.clickHandler);
  }

  clickHandler() {
    this.setState({
      clicks: this.clicks + 1
    });
  }

  render() {
    let children = this.props.children;

    return (
      <div className="my-component" ref="myComponentDiv">
      <h2>My Component ({this.state.clicks} clicks})</h2>
      <h3>{this.props.headerText}</h3>
    {children}
    </div>
    );
  }
}
```

答案：

1.在构造函数没有将 `props` 传递给 `super`，它应该包括以下行

```
constructor(props) {
  super(props);
  // ...
}
```

2.事件监听器(通过`addEventListener()`分配时)的作用域不正确，因为 ES6 不提供自动绑定。因此，开发人员可以在构造函数中重新分配`clickHandler`来包含正确的绑定:

```
constructor(props) {
  super(props);
  this.clickHandler = this.clickHandler.bind(this);
  // ...
}
```

问题 13：为什么不直接更新 `state` 呢？

> 主题: React
> 难度: ⭐⭐⭐

如果试图直接更新 `state` ，则不会重新渲染组件。

```
// 错误
This.state.message = 'Hello world';
```

需要使用`setState()`方法来更新 `state`。它调度对组件`state`对象的更新。当`state`改变时，组件通过重新渲染来响应：

```
// 正确做法
This.setState({message: ‘Hello World’});
```

问题 14：React 组件生命周期有哪些不同阶段?

> 主题: React
> 难度: ⭐⭐⭐

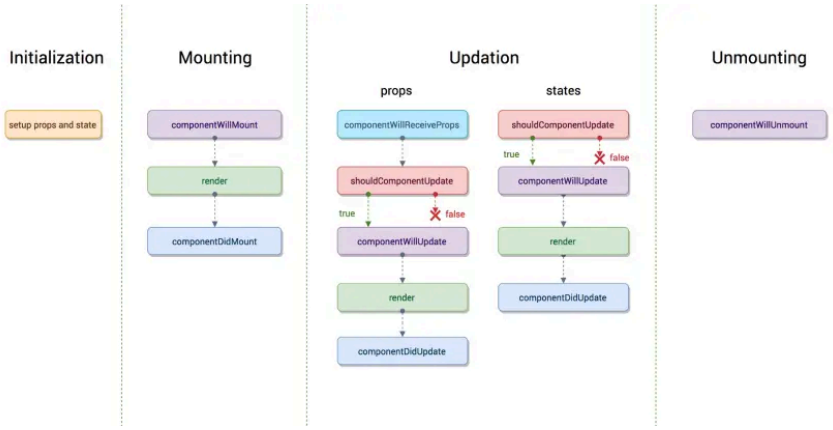
在组件生命周期中有四个不同的阶段：

1. **Initialization**：在这个阶段，组件准备设置初始化状态和默认属性。
2. **Mounting**：react 组件已经准备好挂载到浏览器 DOM 中。这个阶段包括`componentWillMount`和`componentDidMount`生命周期方法。

3. **Updating**：在这个阶段，组件以两种方式更新，发送新的 props 和 state 状态。此阶段包括`shouldComponentUpdate`、`componentWillUpdate`和`componentDidUpdate`生命周期方法。
4. **Unmounting**：在这个阶段，组件已经不再被需要了，它从浏览器 DOM 中卸载下来。这个阶段包含 `componentWillUnmount` 生命周期方法。

除以上四个常用生命周期外，还有一个错误处理的阶段：

Error Handling：在这个阶段，不论在渲染的过程中，还是在生命周期方法中或是在任何子组件的构造函数中发生错误，该组件都会被调用。这个阶段包含了 `componentDidCatch` 生命周期方法。



问题 15：React 的生命周期方法有哪些?

> 主题: React
> 难度: ⭐⭐⭐

- `componentWillMount`:在渲染之前执行，用于根组件中的 App 级配置。
- `componentDidMount`：在第一次渲染之后执行，可以在这里做AJAX请求，DOM 的操作或状态更新以及设置事件监听器。
- `componentWillReceiveProps`：在初始化`render`的时候不会执行，它会在组件接受到新的状态(Props)时被触发，一般用于父组件状态更新时子组件的重新渲染
- `shouldComponentUpdate`：确定是否更新组件。默认情况下，它返回`true`。如果确定在 `state` 或 `props` 更新后组件不需要在重新渲染，则可以返回`false`，这是一个提高性能的方法。
- `componentWillUpdate`：在`shouldComponentUpdate`返回 `true` 确定要更新组件之前件之前执行。
- `componentDidUpdate`：它主要用于更新DOM以响应`props`或`state`更改。
- `componentWillUnmount`：它用于取消任何的网络请求，或删除与组件关联的所有事件监听器。

问题 16：这三个点(...)在 React 干嘛用的?

> 主题: React
> 难度: ⭐⭐⭐

`...` 在React（使用JSX）代码中做什么？它叫什么？

```
<Modal {...this.props} title='Modal heading'  animation={false}/>
```

这个叫扩展操作符号或者展开操作符，例如，如果`this.props`包含`a: 1`和`b: 2`，则

```
<Modal {...this.props} title='Modal heading' animation={false}>
```

等价于下面内容：

```
<Modal a={this.props.a} b={this.props.b} title='Modal heading' an
```

扩展符号不仅适用于该用例，而且对于创建具有现有对象的大多数（或全部）属性的新对象非常方便，在更新`state` 咱们就经常这么做：

```
this.setState(prevState => {
    return {foo: {...prevState.foo, a: "updated"}};
});
```

问题 17：使用 React Hooks 好处是啥?

> 主题: React
> 难度: ⭐⭐⭐

首先，Hooks 通常支持提取和重用跨多个组件通用的有状态逻辑，而无需承担高阶组件或渲染 `props` 的负担。`Hooks` 可以轻松地操作函数组件的状态，而不需要将它们转换为类组件。

Hooks 在类中不起作用，通过使用它们，咱们可以完全避免使用生命周期方法，例如 `componentDidMount`、`componentDidUpdate`、`componentWillUnmount`。相反，使用像`useEffect`这样的内置钩子。

问题 18：什么是 React Hooks?

> 主题: React
> 难度: ⭐ ⭐ ⭐

Hooks是 React 16.8 中的新添加内容。它们允许在不编写类的情况下使用`state`和其他 React 特性。使用 Hooks，可以从组件中提取有状态逻辑，这样就可以独立地测试和重用它。Hooks 允许咱们在不改变组件层次结构的情况下重用有状态逻辑，这样在许多组件之间或与社区共享 Hooks 变得很容易。

问题 19：React 中的 `useState()` 是什么?

> 主题: React
> 难度: ⭐ ⭐ ⭐

下面说明`useState(0)`的用途：

```
...
const [count, setCounter] = useState(0);
const [moreStuff, setMoreStuff] = useState(...);
...

const setCount = () => {
    setCounter(count + 1);
    setMoreStuff(...);
    ...
};
```

`useState` 是一个内置的 React Hook。`useState(0)` 返回一个元组，其中第一个参数`count`是计数器的当前状态，`setCounter` 提供更新计数器状态的方法。

咱们可以在任何地方使用`setCounter`方法更新计数状态-在这种情况下，咱们在`setCount`函数内部使用它可以做更多的事情，使用 Hooks，能够使咱们的代码保持更多功能，还可以避免过多使用基于类的组件。

问题 20：React 中的StrictMode(严格模式)是什么？？

> 主题: React
> 难度: ⭐ ⭐ ⭐

React 的`StrictMode`是一种辅助组件，可以帮助咱们编写更好的 react 组件，可以使用`<StrictMode />`包装一组组件，并且可以帮咱们以下检查：

- 验证内部组件是否遵循某些推荐做法，如果没有，会在控制台给出警告。

- 验证是否使用的已经废弃的方法，如果有，会在控制台给出警告。
- 通过识别潜在的风险预防一些副作用。

问题 21：为什么类方法需要绑定到类实例?

> 主题: React
> 难度: ⭐ ⭐ ⭐

在 JS 中，`this` 值会根据当前上下文变化。在 React 类组件方法中，开发人员通常希望 `this` 引用组件的当前实例，因此有必要将这些方法绑定到实例。通常这是在构造函数中完成的:

```
class SubmitButton extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      isFormSubmitted: false
    };
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleSubmit() {
    this.setState({
      isFormSubmitted: true
    });
  }

  render() {
    return (
      <button onClick={this.handleSubmit}>Submit</button>
    )
  }
}
```

问题 22：什么是 prop drilling，如何避免?

> 主题: React
> 难度: ⭐ ⭐ ⭐

在构建 React 应用程序时，在多层嵌套组件来使用另一个嵌套组件提供的数据。最简单的方法是将一个 `prop` 从每个组件一层层的传递下去，从源组件传递到深层嵌套组件，这叫做**prop drilling**。

`prop drilling`的主要缺点是原本不需要数据的组件变得不必要地复杂，并且难以维护。

为了避免`prop drilling`，一种常用的方法是使用**React Context**。通过定义提供数据的`Provider`组件，并允许嵌套的组件通过`Consumer`组件或`useContext` Hook 使用上下文数据。

问题 23：描述 Flux 与 MVC?

> 主题: React
> 难度: ⭐ ⭐ ⭐

传统的 MVC 模式在分离数据(Model)、UI(View和逻辑(Controller)方面工作得很好，但是 MVC 架构经常遇到两个主要问题:

数据流不够清晰:跨视图发生的级联更新常常会导致混乱的事件网络，难于调试。

缺乏数据完整性:模型数据可以在任何地方发生突变，从而在整个UI中产生不可预测的结果。

使用 Flux 模式的复杂用户界面不再遭受级联更新，任何给定的React 组件都能够根据 `store` 提供的数据重建其状态。Flux 模式还通过限制对共享数据的直接访问来加强数据完整性。

问题 24：受控组件和非受控组件区别是啥?

> 主题: React
> 难度: ⭐ ⭐ ⭐

- **受控组件**是 React 控制中的组件，并且是表单数据真实的唯一来源。
- 非受控组件是由 DOM 处理表单数据的地方，而不是在 React 组件中。

尽管非受控组件通常更易于实现，因为只需使用`refs`即可从 DOM 中获取值，但通常建议优先选择受控制的组件，而不是非受控制的组件。

这样做的主要原因是受控组件支持即时字段验证，允许有条件地禁用/启用按钮，强制输入格式。

问题 25：这段代码有什么问题吗?

> 主题: React
> 难度: ⭐ ⭐ ⭐ ⭐

这段代码有什么问题:

```
this.setState((prevState, props) => {
  return {
    streak: prevState.streak + props.count
  }
})
```

答案：

没有什么问题。这种方式很少被使用，咱们可以将一个函数传递给 `setState`，该函数接收上一个 `state` 的值和当前的`props`，并返回一个新的状态，如果咱们需要根据以前的状态重新设置状态，推荐使用这种方式。

问题 26：什么是 React Context?

> 主题: React
> 难度: ⭐⭐⭐⭐

`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` 属性。

问题 27：什么是 React Fiber?

> 主题: React
> 难度: ⭐⭐⭐⭐

Fiber 是 React 16 中新的协调引擎或重新实现核心算法。它的主要目标是支持虚拟DOM的增量渲染。**React Fiber** 的目标是提高其在动画、布局、手势、暂停、中止或重用等方面的适用性，并为不同类型的更新分配优先级，以及新的并发原语。

React Fiber 的目标是增强其在动画、布局和手势等领域的适用性。它的主要特性是增量渲染:能够将渲染工作分割成块，并将其分散到多个帧中。

问题 28：如何在 ReactJS 的 Props上应用验证?

> 主题: React
> 难度: ⭐⭐⭐⭐

当应用程序在开发模式下运行时，React 将自动检查咱们在组件上设置的所有 `props`，以确保它们具有正确的数据类型。对于不正确的类型，开发模式下会在控制台中生成警告消息，而在生产模式中由于性能影响而禁用它。强制的 `props` 用 `isRequired`定义的。

下面是一组预定义的 prop 类型:

- React.PropTypes.string
- React.PropTypes.number
- React.PropTypes.func
- React.PropTypes.node
- React.PropTypes.bool

例如，咱们为用户组件定义了如下的`propTypes`

```
import PropTypes from 'prop-types';

class User extends React.Component {
  render() {
    return (
      <h1>Welcome, {this.props.name}</h1>
      <h2>Age, {this.props.age}
    );
  }
}

User.propTypes = {
  name: PropTypes.string.isRequired,
  age: PropTypes.number.isRequired
};
```

问题 29：在 React 中使用构造函数和 getInitialState 有什么区别?

> 主题: React
> 难度: ⭐ ⭐ ⭐ ⭐

构造函数和`getInitialState`之间的区别就是`ES6`和`ES5`本身的区别。在使用`ES6`类时，应该在构造函数中初始化`state`，并在使用`React.createClass`时定义`getInitialState`方法。

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = { /* initial state */ };
  }
}
```

等价于:

```
var MyComponent = React.createClass({
  getInitialState() {
    return { /* initial state */ };
```

```
  },
});
```

问题 30：如何有条件地向 React 组件添加属性?

> 主题: React
> 难度: ⭐⭐⭐⭐

对于某些属性，React 非常聪明，如果传递给它的值是虚值，可以省略该属性。例如：

```
var InputComponent = React.createClass({
    render: function() {
      var required = true;
      var disabled = false;

      return (
        <input type="text" disabled={disabled} required={required
      );
    }
});
```

渲染结果：

```
<input type="text" required>
```

另一种可能的方法是：

```
var condition = true;

var component = (
  <div
    value="foo"
    { ...( condition && { disabled: true } ) } />
);
```

问题 31：Hooks会取代 `render props` 和高阶组件吗?

> 主题: React
> 难度: ⭐⭐⭐⭐

通常，`render props`和高阶组件仅渲染一个子组件。React团队认为，Hooks 是服务此用例的更简单方法。

这两种模式仍然有一席之地(例如，一个虚拟的 `scroller` 组件可能有一个 `renderItem prop`，或者一个可视化的容器组件可能有它自己的 DOM 结构)。但在大多数情况下，Hooks 就足够了，可以帮助减少树中的嵌套。

问题 32：如何避免组件的重新渲染?

> 主题: React
> 难度: ⭐ ⭐ ⭐ ⭐

React 中最常见的问题之一是组件不必要地重新渲染。React 提供了两个方法，在这些情况下非常有用：

- `React.memo()`:这可以防止不必要地重新渲染函数组件
- `PureComponent`:这可以防止不必要地重新渲染类组件

这两种方法都依赖于对传递给组件的`props`的浅比较，如果 `props` 没有改变，那么组件将不会重新渲染。虽然这两种工具都非常有用，但是浅比较会带来额外的性能损失，因此如果使用不当，这两种方法都会对性能产生负面影响。

通过使用 **React Profiler**，可以在使用这些方法前后对性能进行测量，从而确保通过进行给定的更改来实际改进性能。

问题 33：什么是纯函数?

> 主题: React
> 难度: ⭐ ⭐ ⭐ ⭐ ⭐

纯函数是不依赖并且不会在其作用域之外修改变量状态的函数。本质上，纯函数始终在给定相同参数的情况下返回相同结果。

问题 34：当调用`setState`时，React `render` 是如何工作的?

> 主题: React
> 难度: ⭐ ⭐ ⭐ ⭐ ⭐

咱们可以将"`render`"分为两个步骤：

1. 虚拟 DOM 渲染:当`render`方法被调用时，它返回一个新的组件的虚拟 DOM 结构。当调用`setState()`时，`render`会被再次调用，因为默认情况下`shouldComponentUpdate`总是返回`true`，所以默认情况下 React 是没有优化的。
2. 原生 DOM 渲染:React 只会在虚拟DOM中修改真实DOM节点，而且修改的次数非常少——这是很棒的React特性，它优化了真实DOM的

变化，使React变得更快。

问题 35：如何避免在React重新绑定实例?

> 主题: React
> 难度: ⭐⭐⭐⭐⭐

有几种常用方法可以避免在 React 中绑定方法：

1.将事件处理程序定义为内联箭头函数

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

  render() {
    return (
      <button onClick={() => {
        this.setState({ isFormSubmitted: true });
      }}>Submit</button>
    )
  }
}
```

2.使用箭头函数来定义方法：

```
class SubmitButton extends React.Component {
  state = {
    isFormSubmitted: false
  }

  handleSubmit = () => {
    this.setState({
      isFormSubmitted: true
    });
  }

  render() {
    return (
      <button onClick={this.handleSubmit}>Submit</button>
    )
  }
}
```

3.使用带有 Hooks 的函数组件

```
const SubmitButton = () => {
  const [isFormSubmitted, setIsFormSubmitted] = useState(false);

  return (
    <button onClick={() => {
        setIsFormSubmitted(true);
    }}>Submit</button>
  )
};
```

代码部署后可能存在的BUG没法实时知道，事后为了解决这些BUG，花了大量的时间进行log 调试，这边顺便给大家推荐一个好用的BUG监控工具 Fundebug。

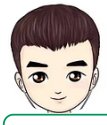
原文：https://www.gangboard.com/blo...

交流

文章每周持续更新，可以微信搜索「 大迁世界 」第一时间阅读和催更（比博客早一到两篇哟），本文 GitHub https://github.com/qq449245884/xiaozhi 已经收录，整理了很多我的文档，欢迎Star和完善，大家面试可以参照考点复习，另外关注公众号，后台回复**福利**，即可看到福利，你懂的。

前端 javascript html node.js

阅读 7k • 更新于 2020-03-11



王大冶
67.8k 声望 • 104.9k 粉丝 •

« 上一篇
上次24个实用 ES6 方法受到好评，... 这 10 个技巧让你成为一个更好的...
下一篇 »

引用和评论

推荐阅读

我们忘记了前端基础知识
王大冶 • 阅读 6

使用JavaScript完成二叉树的一些基本操作
tonychen • 赞 53 • 阅读 5.9k • 评论 5

深入理解React Diff算法
nero • 赞 36 • 阅读 15.8k • 评论 4

前端开发方法集
寒青 • 赞 23 • 阅读 2.6k

React中的高优先级任务插队机制
nero • 赞 32 • 阅读 15.1k • 评论 14

解读生产环境为何避免使用console
小皇帝James • 赞 18 • 阅读 14.5k • 评论 3

关于小程序如何做到强制更新
南玖 • 赞 13 • 阅读 828 • 评论 1

1 条评论 得票 最新

撰写评论 ...

提交评论

评论支持部分 Markdown 语法：**粗体** _斜体_ [链接](http://example.com) `代码` - 列表 > 引用。你还可以使用 @ 来通知其他用户。

all2005：撸起来

• 回复 • 2020-03-12