react面试题合集

何为 redux

Redux 的基本思想是整个应用的 state 保持在一个单一的 store 中。store 就是一个简单的 javascript 对象，而改变应用 state 的唯一方式是在应用中触发 actions，然后为这些 actions 编写 reducers 来修改 state。整个 state 转化是在 reducers 中完成，并且不应该有任何副作用。

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

javascript 复制代码

```javascript
// ES5
var React = require('react');

// ES6
import React from 'react';
```

1. export 与 exports

javascript 复制代码

```javascript
// ES5
module.exports = Component;

// ES6
export default Component;
```

1. component 和 function

javascript 复制代码

```javascript
// ES5
var MyComponent = React.createClass({
    render: function() {
        return
            <h3>Hello Edureka!</h3>;
```

```
    }
});

// ES6
class MyComponent extends React.Component {
    render() {
        return
            <h3>Hello Edureka!</h3>;
    }
}
```

1. props

javascript 复制代码

```
// ES5
var App = React.createClass({
    propTypes: { name: React.PropTypes.string },
    render: function() {
        return
            <h3>Hello, {this.props.name}!</h3>;
    }
});

// ES6
class App extends React.Component {
    render() {
        return
            <h3>Hello, {this.props.name}!</h3>;
    }
}
```

1. state

javascript 复制代码

```
// ES5
var App = React.createClass({
    getInitialState: function() {
        return { name: 'world' };
    },
    render: function() {
        return
            <h3>Hello, {this.state.name}!</h3>;
    }
});

// ES6
class App extends React.Component {
```

```
    constructor() {
        super();
        this.state = { name: 'world' };
    }
    render() {
        return
            <h3>Hello, {this.state.name}!</h3>;
    }
}
```

react-router4的核心

- 路由变成了组件
- 分散到各个页面，不需要配置 比如 `<link> <route></route>`

在 React 中如何处理事件

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。

什么情况下使用异步组件

- 提高页面加载速度，使用 `reloadable` 把各个页面分别单独打包，按需加载

react中这两个生命周期会触发死循环

> `componentWillUpdate` 生命周期在 `shouldComponentUpdate` 返回true后被触发。在这两个生命周期只要视图更新就会触发，因此不能再这两个生命周期中使用setState。否则会导致死循环

参考 前端进阶面试题详细解答

React 如何区分 Class组件 和 Function组件

一般的方式是借助 typeof 和 Function.prototype.toString 来判断当前是不是 class，如下：

javascript 复制代码

```
function isClass(func) {
  return typeof func === 'function'
    && /^class\s/.test(Function.prototype.toString.call(func));
}
```

但是这个方式有它的局限性，因为如果用了 babel 等转换工具，将 class 写法全部转为 function 写法，上面的判断就会失效。

> React 区分 Class组件 和 Function组件的方式很巧妙，由于所有的类组件都要继承 React.Component，所以只要判断原型链上是否有 React.Component 就可以了：

javascript 复制代码

```
AComponent.prototype instanceof React.Component
```

为什么 JSX 中的组件名要以大写字母开头

因为 React 要知道当前渲染的是组件还是 HTML 元素

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

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。

Hooks可以取代 `render props` 和高阶组件吗?

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。

React中keys的作用是什么?

javascript 复制代码

```
render () {
  return (
    <ul>
      {this.state.todoItems.map(({item,i}) => {
        return <li key={i}>{item}</li>
      })}
    </ul>
  )
}
```

在React Diff算法中React会借助元素的Key值来判断该元素是新近创建的还是被移动而来的元素，从而减少不必要的元素重渲染。

##s# 如何避免在React重新绑定实例?

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

javascript 复制代码

```
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：

```javascript 复制代码
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

```javascript 复制代码
const SubmitButton = () => {
  const [isFormSubmitted, setIsFormSubmitted] = useState(false);
  return (
    <button
      onClick={() => {
        setIsFormSubmitted(true);
      }}
    >
      Submit
    </button>
  );
};
```

setState到底是异步还是同步?

先给出答案: 有时表现出异步，有时表现出同步。

- setState只在合成事件和钩子函数中是“异步”的，在原生事件和setTimeout中都是同步的；
- setState的“异步”并不是说内部由异步代码实现，其实本身执行的过程和代码都是同步的，只是合成事件和钩子函数的调用顺序在更新之前，导致在合成事件和钩子函数中没法立马拿到更新后的值，形成了所谓的“异步”，当然可以通过第二个参数setState(partialState, callback)中的callback拿到更新后的结果；
- setState的批量更新优化也是建立在“异步”（合成事件、钩子函数）之上的，在原生事件和setTimeout中不会批量更新，在“异步”中如果对同一个值进行多次 setState，setState的批

量更新策略会对其进行覆盖，取最后一次的执行，如果是同时setState多个不同的值，在更新时会对其进行合并批量更新。

React setState 笔试题，下面的代码输出什么

javascript 复制代码

```javascript
class Example extends React.Component {
  constructor() {
  super()
  this.state = {
    val: 0
  }
}
componentDidMount() {
  this.setState({ val: this.state.val + 1 })
  console.log(this.state.val)
  // 第 1 次 log
  this.setState({ val: this.state.val + 1 })
  console.log(this.state.val)
  // 第 2 次 log
  setTimeout(() => {
    this.setState({ val: this.state.val + 1 })
    console.log(this.state.val)
    // 第 3 次 log
    this.setState({ val: this.state.val + 1 })
    console.log(this.state.val)
    // 第 4 次 log
    }, 0)
  }
  render() {
    return null
  }
}

// 答：0, 0, 1, 2
```

什么是 React Fiber?

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。

一般可以用哪些值作为key

- 最好使用每一条数据中的唯一标识作为key，比如：手机号，id值，身份证号，学号等
- 也可以用数据的索引值（可能会出现一些问题）

useEffect(fn, []) 和 componentDidMount 有什么差异

> `useEffect` 会捕获 `props` 和 state。所以即便在回调函数里，你拿到的还是初始的 props 和 state。如果想得到"最新"的值，可以使用 ref。

讲讲什么是 JSX？

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。

javascript 复制代码

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

什么是 React的refs？为什么它们很重要

refs允许你直接访问DOM元素或组件实例。为了使用它们，可以向组件添加个ref属性。 如果该属性的值是一个回调函数，它将接受底层的DOM元素或组件的已挂载实例作为其第一个参数。可以在组件中存储它。

javascript 复制代码

```
export class App extends Component {
  showResult() {
    console.log(this.input.value);
  }
```

```
  render() {
    return (
      <div>
        <input type="text" ref={(input) => (this.input = input)} />
        <button onClick={this.showResult.bind(this)}>展示结果</button>
      </div>
    );
  }
}
```

如果该属性值是一个字符串， React将会在组件实例化对象的refs属性中，存储一个同名属性，该属性是对这个DOM元素的引用。可以通过原生的 DOM API操作它。

javascript 复制代码

```
export class App extends Component {
  showResult() {
    console.log(this.refs.username.value);
  }
  render() {
    return (
      <div>
        <input type="text" ref="username" />
        <button onClick={this.showResu1t.bind(this)}>展示结果</button>
      </div>
    );
  }
}
```

react性能优化方案

- 重写 `shouldComponentUpdate` 来避免不必要的dom操作
- 使用 `production` 版本的 `react.js`
- 使用 `key` 来帮助 `React` 识别列表中所有子组件的最小变化