腾讯前端常考react面试题（持续更新中）

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。

constructor

text 复制代码

```
答案是：在 constructor 函数里面，需要用到props的值的时候，就需要调用 super(props)
```

1. class语法糖默认会帮你定义一个constructor，所以当你不需要使用constructor的时候，是可以不用自己定义的
2. 当你自己定义一个constructor的时候，就一定要写super()，否则拿不到this
3. 当你在constructor里面想要使用props的值，就需要传入props这个参数给super，调用super(props)，否则只需要写super()

react 的渲染过程中，兄弟节点之间是怎么处理的？也就是key值不一样的时候

> 通常我们输出节点的时候都是map一个数组然后返回一个 `ReactNode` ，为了方便 `react` 内部进行优化，我们必须给每一个 `reactNode` 添加 `key` ，这个 `key prop` 在设计值处不是给开发者用的，而是给react用的，大概的作用就是给每一个 `reactNode` 添加一个身份标识，方便react进行识别，在重渲染过程中，如果key一样，若组件属性有所变化，则 `react` 只更新组件对应的属性；没有变化则不更新，如果key不一样，则react先销毁该组件，然后重新创建该组件

React组件的构造函数有什么作用？它是必须的吗？

构造函数主要用于两个目的：

- 通过将对象分配给this.state来初始化本地状态
- 将事件处理程序方法绑定到实例上

所以，当在React class中需要设置state的初始值或者绑定事件时，需要加上构造函数，官方Demo：

javascript 复制代码

```
class LikeButton extends React.Component {
  constructor() {
    super();
    this.state = {
      liked: false
    };
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    this.setState({liked: !this.state.liked});
  }
  render() {
    const text = this.state.liked ? 'liked' : 'haven\'t liked';
    return (
      <div onClick={this.handleClick}>
        You {text} this. Click to toggle.      </div>
    );
  }
}
ReactDOM.render(
  <LikeButton />,
  document.getElementById('example')
);
```

构造函数用来新建父类的this对象；子类必须在constructor方法中调用super方法；否则新建实例时会报错；因为子类没有自己的this对象，而是继承父类的this对象，然后对其进行加工。如果不调用super方法；子类就得不到this对象。

注意：

- constructor () 必须配上 super(), 如果要在constructor 内部使用 this.props 就要 传入 props , 否则不用
- JavaScript中的 bind 每次都会返回一个新的函数, 为了性能等考虑, 尽量在constructor中绑定事件

何为受控组件(controlled component)

在 HTML 中，类似 `<input>`，`<textarea>` 和 `<select>` 这样的表单元素会维护自身的状态，并基于用户的输入来更新。当用户提交表单时，前面提到的元素的值将随表单一起被发送。但在 React 中会有些不同，包含表单元素的组件将会在 state 中追踪输入的值，并且每次调用回调函数时，如 `onChange` 会更新 state，重新渲染组件。一个输入表单元素，它的值通过 React 的这种方式来控制，这样的元素就被称为"受控元素"。

React.Children.map和js的map有什么区别?

JavaScript中的map不会对为null或者undefined的数据进行处理，而React.Children.map中的map可以处理React.Children为null或者undefined的情况。

参考 前端进阶面试题详细解答

在React中组件的this.state和setState有什么区别?

this.state通常是用来初始化state的，this.setState是用来修改state值的。如果初始化了state之后再使用this.state，之前的state会被覆盖掉，如果使用this.setState，只会替换掉相应的state值。所以，如果想要修改state的值，就需要使用setState，而不能直接修改state，直接修改state之后页面是不会更新的。

什么是受控组件和非受控组件

- 受状态控制的组件，必须要有onChange方法，否则不能使用 受控组件可以赋予默认值（官方推荐使用 受控组件） 实现双向数据绑定

javascript 复制代码

```
class Input extends Component{
    constructor(){
        super();
        this.state = {val:'100'}
    }
    handleChange = (e) =>{ //e是事件源
        let val = e.target.value;
        this.setState({val});
    };
    render(){
        return (<div>
            <input type="text" value={this.state.val} onChange={this.handleChange}/>
            {this.state.val}
```

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

- 非受控也就意味着我可以不需要设置它的state属性，而通过ref来操作真实的DOM

javascript 复制代码

```
class Sum extends Component{
    constructor(){
        super();
        this.state =  {result:''}
    }
    //通过ref设置的属性 可以通过this.refs获取到对应的dom元素
    handleChange = () =>{
        let result = this.refs.a.value + this.b.value;
        this.setState({result});
    };
    render(){
        return (
            <div onChange={this.handleChange}>
                <input type="number" ref="a"/>
                {/*x代表的真实的dom,把元素挂载在了当前实例上*/}
                <input type="number" ref={(x)=>{
                    this.b = x;
                }}/>
                {this.state.result}
            </div>
        )
    }
}
```

对componentWillReceiveProps 的理解

该方法当 `props` 发生变化时执行，初始化 `render` 时不执行，在这个回调函数里面，你可以根据属性的变化，通过调用 `this.setState()` 来更新你的组件状态，旧的属性还是可以通过 `this.props` 来获取,这里调用更新状态是安全的，并不会触发额外的 `render` 调用。

使用好处： 在这个生命周期中，可以在子组件的render函数执行前获取新的props，从而更新子组件自己的state。 可以将数据请求放在这里进行执行，需要传的参数则从componentWillReceiveProps(nextProps)中获取。而不必将所有的请求都放在父组件中。于是该请求只会在该组件渲染时才会发出，从而减轻请求负担。

componentWillReceiveProps在初始化render的时候不会执行，它会在Component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。

redux是如何更新值得

用户发起操作之后，dispatch发送action ，根据type，触发对于的reducer，reducer 就是一个纯函数，接收旧的 state 和 action，返回新的 state。通过 subscribe(listener)监听器，派发更新。

在React中遍历的方法有哪些?

（1）遍历数组：map && forEach

javascript 复制代码

```
import React from 'react';

class App extends React.Component {
  render() {
    let arr = ['a', 'b', 'c', 'd'];
    return (
      <ul>
        {
          arr.map((item, index) => {
            return <li key={index}>{item}</li>
          })
        }
      </ul>
    )
  }
}

class App extends React.Component {
  render() {
    let arr = ['a', 'b', 'c', 'd'];
    return (
      <ul>
        {
          arr.forEach((item, index) => {
            return <li key={index}>{item}</li>
          })
        }
      </ul>
    )
  }
}
```

(2) 遍历对象：map && for in

javascript 复制代码

```
class App extends React.Component {
  render() {
    let obj = {
      a: 1,
      b: 2,
      c: 3
    }
    return (
      <ul>
        {
          (() => {
            let domArr = [];
            for(const key in obj) {
              if(obj.hasOwnProperty(key)) {
                const value = obj[key]
                domArr.push(<li key={key}>{value}</li>)
              }
            }
            return domArr;
          })()
        }
      </ul>
    )
  }
}

// Object.entries() 把对象转换成数组
class App extends React.Component {
  render() {
    let obj = {
      a: 1,
      b: 2,
      c: 3
    }
    return (
      <ul>
        {
          Object.entries(obj).map(([key, value], index) => {   // item是一个数组，把item解构，写法是[k
            return <li key={key}>{value}</li>
          })
        }
      </ul>
```

```
    )
  }
}
```

什么是状态提升

使用 react 经常会遇到几个组件需要共用状态数据的情况。这种情况下，我们最好将这部分共享的状态提升至他们最近的父组件当中进行管理。我们来看一下具体如何操作吧。

javascript 复制代码

```javascript
import React from 'react'
class Child_1 extends React.Component{
    constructor(props){
        super(props)
    }
    render(){
        return (
            <div>
                <h1>{this.props.value+2}</h1>
            </div>
        )
    }
}
class Child_2 extends React.Component{
    constructor(props){
        super(props)
    }
    render(){
        return (
            <div>
                <h1>{this.props.value+1}</h1>
            </div>
        )
    }
}
class Three extends React.Component {
    constructor(props){
        super(props)
        this.state = {
            txt:"牛逼"
        }
        this.handleChange = this.handleChange.bind(this)
    }
    handleChange(e){
```

```
            this.setState({
                txt:e.target.value
            })
        }
        render(){
           return (
                <div>
                    <input type="text" value={this.state.txt} onChange={this.handleChange}/>
                    <p>{this.state.txt}</p>
                    <Child_1 value={this.state.txt}/>
                    <Child_2 value={this.state.txt}/>
                </div>
           )
        }
}
export default Three
```

展示组件(Presentational component)和容器组件(Container component)之间有何不同

展示组件关心组件看起来是什么。展示专门通过 props 接受数据和回调，并且几乎不会有自身的状态，但当展示组件拥有自身的状态时，通常也只关心 UI 状态而不是数据的状态。

容器组件则更关心组件是如何运作的。容器组件会为展示组件或者其它容器组件提供数据和行为(behavior)，它们会调用 `Flux actions` ，并将其作为回调提供给展示组件。容器组件经常是有状态的，因为它们是(其它组件的)数据源。

React Hook 的使用限制有哪些?

React Hooks 的限制主要有两条：

- 不要在循环、条件或嵌套函数中调用 Hook；
- 在 React 的函数组件中调用 Hook。

那为什么会有这样的限制呢？Hooks 的设计初衷是为了改进 React 组件的开发模式。在旧有的开发模式下遇到了三个问题。

- 组件之间难以复用状态逻辑。过去常见的解决方案是高阶组件、render props 及状态管理框架。
- 复杂的组件变得难以理解。生命周期函数与业务逻辑耦合太深，导致关联部分难以拆分。

- 人和机器都很容易混淆类。常见的有 this 的问题，但在 React 团队中还有类难以优化的问题，希望在编译优化层面做出一些改进。

这三个问题在一定程度上阻碍了 React 的后续发展，所以为了解决这三个问题，Hooks **基于函数组件**开始设计。然而第三个问题决定了 Hooks 只支持函数组件。

那为什么不要在循环、条件或嵌套函数中调用 Hook 呢？因为 Hooks 的设计是基于数组实现。在调用时按顺序加入数组中，如果使用循环、条件或嵌套函数很有可能导致数组取值错位，执行错误的 Hook。当然，实质上 React 的源码里不是数组，是链表。

这些限制会在编码上造成一定程度的心智负担，新手可能会写错，为了避免这样的情况，可以引入 ESLint 的 Hooks 检查插件进行预防。

使用 React 有何优点

- 只需查看 `render` 函数就会很容易知道一个组件是如何被渲染的
- JSX 的引入，使得组件的代码更加可读，也更容易看懂组件的布局，或者组件之间是如何互相引用的
- 支持服务端渲染，这可以改进 SEO 和性能
- 易于测试
- React 只关注 View 层，所以可以和其它任何框架(如Backbone.js, Angular.js)一起使用

React key 是干嘛用的 为什么要加？key 主要是解决哪一类问题的

Keys 是 React 用于追踪哪些列表中元素被修改、被添加或者被移除的辅助标识。在开发过程中，我们需要保证某个元素的 key 在其同级元素中具有唯一性。

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此外，React 还需要借助 Key 值来判断元素与本地状态的关联关系。

注意事项：

- key值一定要和具体的元素一一对应；
- 尽量不要用数组的index去作为key；
- 不要在render的时候用随机数或者其他操作给元素加上不稳定的key，这样造成的性能开销比不加key的情况下更糟糕。

Component, Element, Instance 之间有什么区别和联系?

- **元素：** 一个元素 `element` 是一个普通对象(plain object)，描述了对于一个DOM节点或者其他组件 `component` ，你想让它在屏幕上呈现成什么样子。元素 `element` 可以在它的属性 `props` 中包含其他元素(译注:用于形成元素树)。创建一个React元素 `element` 成本很低。元素 `element` 创建之后是不可变的。
- **组件：** 一个组件 `component` 可以通过多种方式声明。可以是带有一个 `render()` 方法的类，简单点也可以定义为一个函数。这两种情况下，它都把属性 `props` 作为输入，把返回的一棵元素树作为输出。
- **实例：** 一个实例 `instance` 是你在所写的组件类 `component class` 中使用关键字 `this` 所指向的东西(译注:组件实例)。它用来存储本地状态和响应生命周期事件很有用。

函数式组件(`Functional component`)根本没有实例 `instance` 。类组件(`Class component`)有实例 `instance` ，但是永远也不需要直接创建一个组件的实例，因为React帮我们做了这些。

同时引用这三个库react.js、react-dom.js和babel.js它们都有什么作用?

- react：包含react所必须的核心代码
- react-dom：react渲染在不同平台所需要的核心代码
- babel：将jsx转换成React代码的工具

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：

javascript 复制代码

```
class MyComponent extends React.Component{
    render(){
        return(
            <div>
                <h1>Hello</h1>
                <Header/>
            </div>
        );
    }
}
class Header extends React.Component{
    render(){
        return
            <h1>Header Component</h1>
   };
}
```

```
ReactDOM.render(
    <MyComponent/>, document.getElementById('content')
);
```

React中refs的作用是什么？有哪些应用场景？

Refs 提供了一种方式，用于访问在 render 方法中创建的 React 元素或 DOM 节点。Refs 应该谨慎使用，如下场景使用 Refs 比较适合：

- 处理焦点、文本选择或者媒体的控制
- 触发必要的动画
- 集成第三方 DOM 库

Refs 是使用 `React.createRef()` 方法创建的，他通过 `ref` 属性附加到 React 元素上。要在整个组件中使用 Refs，需要将 `ref` 在构造函数中分配给其实例属性：

javascript 复制代码

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

由于函数组件没有实例，因此不能在函数组件上直接使用 `ref`：

javascript 复制代码

```
function MyFunctionalComponent() {
  return <input />;
}
class Parent extends React.Component {
  constructor(props) {
    super(props);
    this.textInput = React.createRef();
  }
  render() {
    // 这将不会工作！
    return (
      <MyFunctionalComponent ref={this.textInput} />
    );
```

```
  }
}
```

但可以通过闭合的帮助在函数组件内部进行使用 Refs：

javascript 复制代码

```
function CustomTextInput(props) {
  // 这里必须声明 textInput，这样 ref 回调才可以引用它
  let textInput = null;
  function handleClick() {
    textInput.focus();
  }
  return (
    <div>
      <input
        type="text"
        ref={(input) => { textInput = input; }} />      <input
        type="button"
        value="Focus the text input"
        onClick={handleClick}
      />
    </div>
  );
}
```

注意：

- 不应该过度的使用 Refs
- `ref` 的返回值取决于节点的类型：
  - 当 `ref` 属性被用于一个普通的 HTML 元素时，`React.createRef()` 将接收底层 DOM 元素作为他的 `current` 属性以创建 `ref` 。
  - 当 `ref` 属性被用于一个自定义的类组件时，`ref` 对象将接收该组件已挂载的实例作为他的 `current` 。
- 当在父组件中需要访问子组件中的 `ref` 时可使用传递 Refs 或回调 Refs。