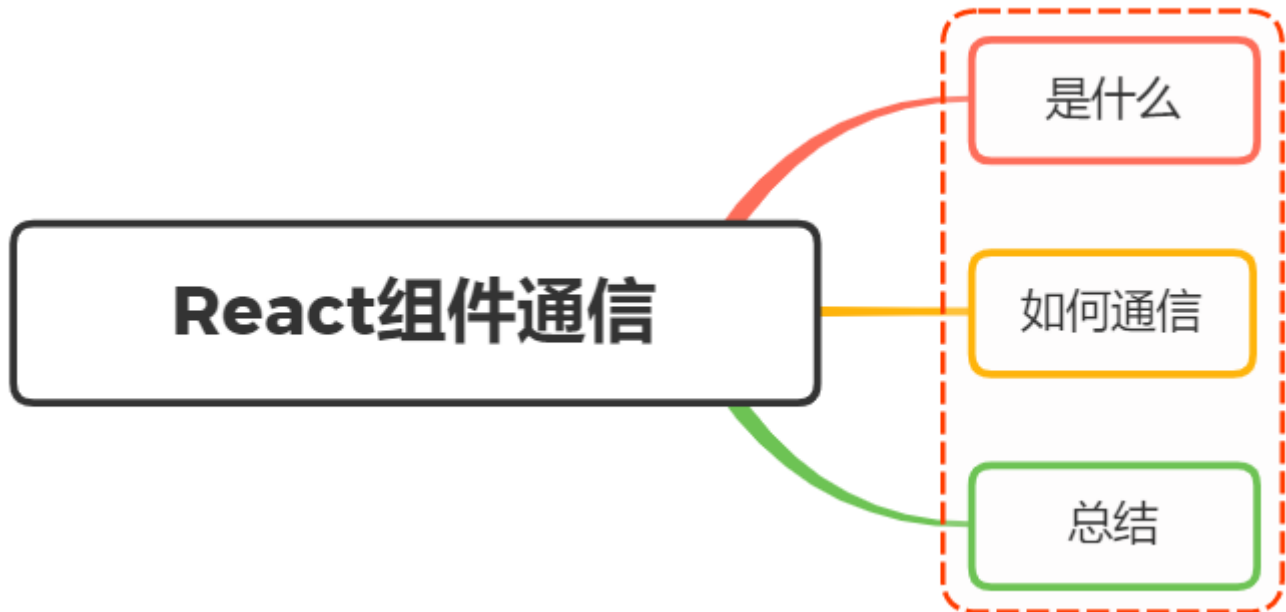


面试官：React中组件之间如何通信？



一、是什么

我们将组件间通信可以拆分为两个词：

- 组件
- 通信

回顾[Vue系列](#)的文章，组件是vue中最强大的功能之一，同样组件化是React的核心思想

相比vue，React的组件更加灵活和多样，按照不同的方式可以分成很多类型的组件

而通信指的是发送者通过某种媒体以某种格式来传递信息到受信者以达到某个目的，广义上，任何信息的交通都是通信

组件间通信即指组件通过某种方式来传递信息以达到某个目的

二、如何通信

组件传递的方式有很多种，根据传送者和接收者可以分为如下：

- 父组件向子组件传递
- 子组件向父组件传递
- 兄弟组件之间的通信
- 父组件向后代组件传递
- 非关系组件传递

父组件向子组件传递

由于React的数据流动为单向的，父组件向子组件传递是最常见的方式

父组件在调用子组件的时候，只需要在子组件标签内传递参数，子组件通过`props`属性就能接收父组件传递过来的参数

```
function EmailInput(props) {  
  return (  
    <label>  
      Email: <input value={props.email} />  
    </label>  
  );  
}  
  
const element = <EmailInput email="123124132@163.com" />;
```

子组件向父组件传递

子组件向父组件通信的基本思路是，父组件向子组件传一个函数，然后通过这个函数的回调，拿到子组件传过来的值

父组件对应代码如下：

```
class Parents extends Component {  
  constructor() {  
    super();  
    this.state = {  
      price: 0  
    };  
  }  
  
  getItemPrice(e) {  
    this.setState({  
      price: e  
    });  
  }  
  
  render() {  
    return (  
      <div>  
        <div>price: {this.state.price}</div>  
        /* 向子组件中传入一个函数 */  
        <Child getPrice={this.getItemPrice.bind(this)} />  
      </div>  
    );  
  }  
}
```

子组件对应代码如下：

```
class Child extends Component {
  clickGoods(e) {
    // 在此函数中传入值
    this.props.getPrice(e);
  }

  render() {
    return (
      <div>
        <button onClick={this.clickGoods.bind(this, 100)}>goods1</button>
        <button onClick={this.clickGoods.bind(this, 1000)}>goods2</button>
      </div>
    );
  }
}
```

兄弟组件之间的通信

如果是兄弟组件之间的传递，则父组件作为中间层来实现数据的互通，通过使用父组件传递

```
class Parent extends React.Component {
  constructor(props) {
    super(props)
    this.state = {count: 0}
  }
  setCount = () => {
    this.setState({count: this.state.count + 1})
  }
  render() {
    return (
      <div>
        <SiblingA
          count={this.state.count}
        />
        <SiblingB
          onClick={this.setCount}
        />
      </div>
    );
  }
}
```

父组件向后代组件传递

父组件向后代组件传递数据是一件最普通的事情，就像全局数据一样

使用`context`提供了组件之间通讯的一种方式，可以共享数据，其他数据都能读取对应的数据

通过使用`React.createContext`创建一个`context`

```
const PriceContext = React.createContext('price')
```

`context`创建成功后，其下存在`Provider`组件用于创建数据源，`Consumer`组件用于接收数据，使用实例如下：

`Provider`组件通过`value`属性用于给后代组件传递数据：

```
<PriceContext.Provider value={100}>  
</PriceContext.Provider>
```

如果想要获取`Provider`传递的数据，可以通过`Consumer`组件或者使用`contextType`属性接收，对应分别如下：

```
class MyClass extends React.Component {  
  static contextType = PriceContext;  
  render() {  
    let price = this.context;  
    /* 基于这个值进行渲染工作 */  
  }  
}
```

`Consumer`组件：

```
<PriceContext.Consumer>  
  { /*这里是一个函数*/ }  
  {  
    price => <div>price : {price}</div>  
  }  
</PriceContext.Consumer>
```

非关系组件传递

如果组件之间关系类型比较复杂的情况，建议将数据进行一个全局资源管理，从而实现通信，例如`redux`。关于`redux`的使用后续再详细介绍

三、总结

由于`React`是单向数据流，主要思想是组件不会改变接收的数据，只会监听数据的变化，当数据发生变化时它们会使用接收到的新值，而不是去修改已有的值

因此，可以看到通信过程中，数据的存储位置都是存放在上级位置中

参考文献

- <https://react.docschina.org/docs/context.html>