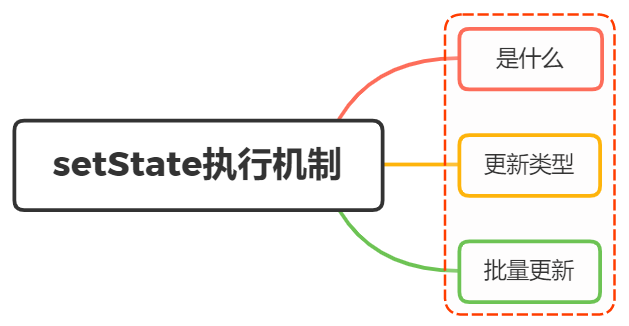
# 面试官：说说 React中的setState执行机制



## 一、是什么

一个组件的显示形态可以由数据状态和外部参数所决定，而数据状态就是state

当需要修改里面的值的状态需要通过调用setState来改变，从而达到更新组件内部数据的作用

如下例子：

import React, { Component } from 'react'  
  
export default class App extends Component {  
 constructor(props) {  
 super(props);  
  
 this.state = {  
 message: "Hello World"  
 }  
 }  
  
 render() {  
 return (  
 <div>  
 <h2>{this.state.message}</h2>  
 <button onClick={e => this.changeText()}>面试官系列</button>  
 </div>  
 )  
 }  
  
 changeText() {  
 this.setState({  
 message: "JS每日一题"  
 })  
 }  
}

通过点击按钮触发onclick事件，执行this.setState方法更新state状态，然后重新执行render函数，从而导致页面的视图更新

如果直接修改state的状态，如下：

changeText() {  
 this.state.message = "你好啊,李银河";  
}

我们会发现页面并不会有任何反应，但是state的状态是已经发生了改变

这是因为React并不像vue2中调用Object.defineProperty数据响应式或者Vue3调用Proxy监听数据的变化

必须通过setState方法来告知react组件state已经发生了改变

关于state方法的定义是从React.Component中继承，定义的源码如下：

Component.prototype.setState = function(partialState, callback) {  
 invariant(  
 typeof partialState === 'object' ||  
 typeof partialState === 'function' ||  
 partialState == null,  
 'setState(...): takes an object of state variables to update or a ' +  
 'function which returns an object of state variables.',  
 );  
 this.updater.enqueueSetState(this, partialState, callback, 'setState');  
};

从上面可以看到setState第一个参数可以是一个对象，或者是一个函数，而第二个参数是一个回调函数，用于可以实时的获取到更新之后的数据

## 二、更新类型

在使用setState更新数据的时候，setState的更新类型分成：

* 异步更新
* 同步更新

### 异步更新

先举出一个例子：

changeText() {  
 this.setState({  
 message: "你好啊"  
 })  
 console.log(this.state.message); // Hello World  
}

从上面可以看到，最终打印结果为Hello world，并不能在执行完setState之后立马拿到最新的state的结果

如果想要立刻获取更新后的值，在第二个参数的回调中更新后会执行

changeText() {  
 this.setState({  
 message: "你好啊"  
 }, () => {  
 console.log(this.state.message); // 你好啊  
 });  
}

### 同步更新

同样先给出一个在setTimeout中更新的例子：

changeText() {  
 setTimeout(() => {  
 this.setState({  
 message: "你好啊  
 });  
 console.log(this.state.message); // 你好啊  
 }, 0);  
}

上面的例子中，可以看到更新是同步

再来举一个原生DOM事件的例子：

componentDidMount() {  
 const btnEl = document.getElementById("btn");  
 btnEl.addEventListener('click', () => {  
 this.setState({  
 message: "你好啊,李银河"  
 });  
 console.log(this.state.message); // 你好啊,李银河  
 })  
}

### 小结

* 在组件生命周期或React合成事件中，setState是异步
* 在setTimeout或者原生dom事件中，setState是同步

### 三、批量更新

同样先给出一个例子：

handleClick = () => {  
 this.setState({  
 count: this.state.count + 1,  
 })  
 console.log(this.state.count) // 1  
  
 this.setState({  
 count: this.state.count + 1,  
 })  
 console.log(this.state.count) // 1  
  
 this.setState({  
 count: this.state.count + 1,  
 })  
 console.log(this.state.count) // 1  
}

点击按钮触发事件，打印的都是 1，页面显示 count 的值为 2

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，取最后一次的执行结果

上述的例子，实际等价于如下：

Object.assign(  
 previousState,  
 {index: state.count+ 1},  
 {index: state.count+ 1},  
 ...  
)

由于后面的数据会覆盖前面的更改，所以最终只加了一次

如果是下一个state依赖前一个state的话，推荐给setState一个参数传入一个function，如下：

onClick = () => {  
 this.setState((prevState, props) => {  
 return {count: prevState.count + 1};  
 });  
 this.setState((prevState, props) => {  
 return {count: prevState.count + 1};  
 });  
}

而在setTimeout或者原生dom事件中，由于是同步的操作，所以并不会进行覆盖现象

## 参考文献

* https://juejin.cn/post/6844903667426918408
* https://juejin.cn/post/6844903636749778958
* https://segmentfault.com/a/1190000039077904