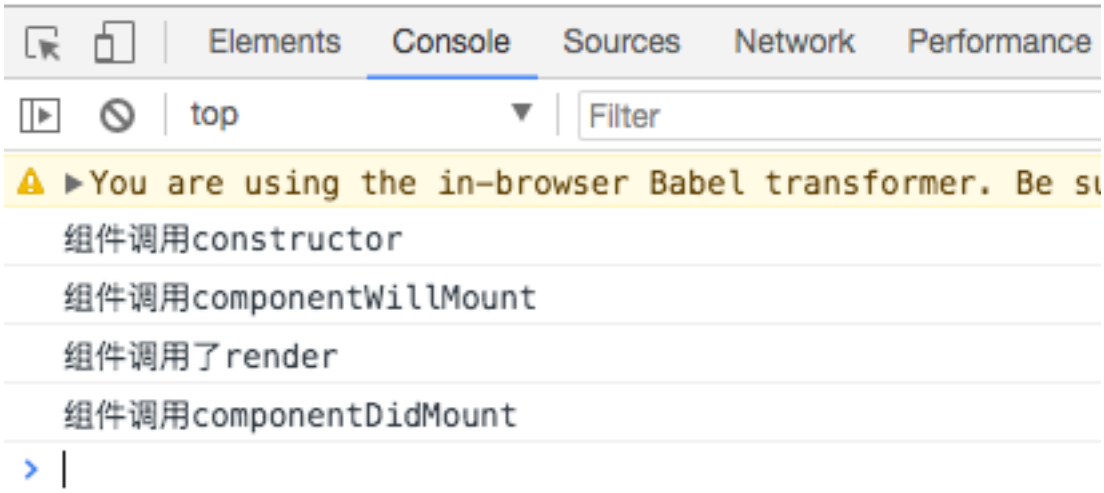
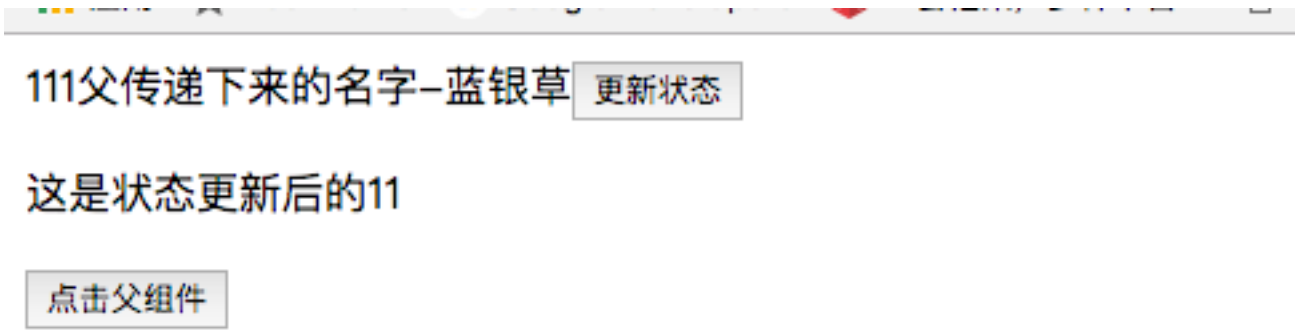
React 生命周期总结

组件在挂载的阶段会调用constructor, componentWillMount, render，挂在后会在调用componentDidMount





现在我们点击更新状态在看调用了哪些函数，

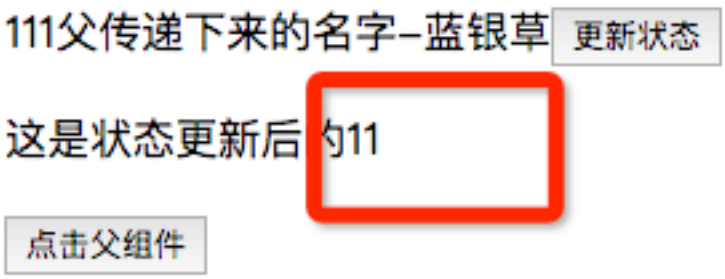
这是我们的更新状态的函数

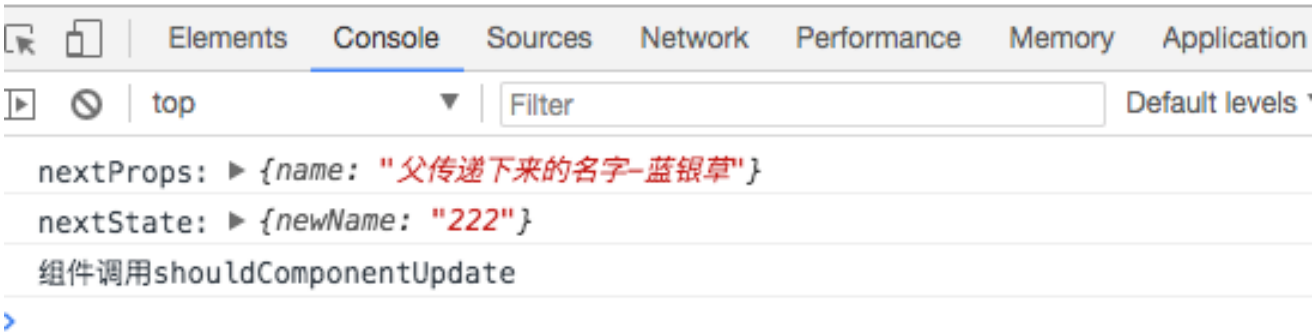
```
    handleClick() {
        this.setState({newName: '222'})
    }
    render() {
        console.log('组件调用了render')
        return (
            <div>
                111{this.props.name}
                <button onClick={this.handleClick}>更新状态</button>
                <p>这是状态更新后的{this.state.newName}</p>
            </div>
        )
    }
}
```

这是我们的shouldComponentUpdate的函数

```
    shouldComponentUpdate(nextProps, nextState) {
        console.log('nextProps:', nextProps)
        console.log('nextState:', nextState)
        console.log('组件调用shouldComponentUpdate')


    }
```

咦，结果是不是很奇怪啊，页面并没有更新啊，看看输出的值是什么和页面的状态





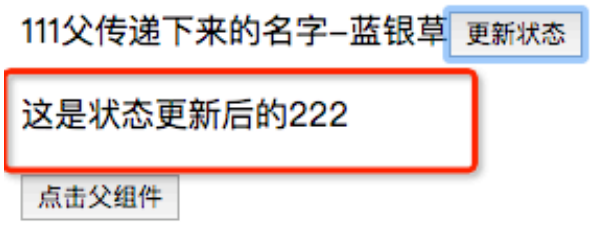
此时页面的值还是11啊，并且输出是调用了shouldComponentUpdate就停止了，

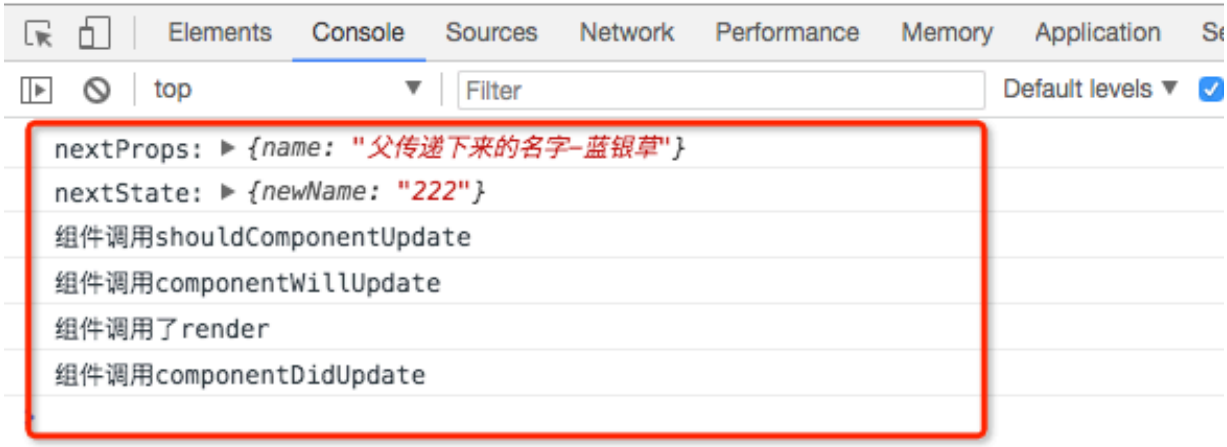
没有再一次渲染了啊。

我们不妨改一下shouldComponentUpdate下，我们让它返回true吧

```
shouldComponentUpdate(nextProps, nextState) {
    console.log('nextProps:', nextProps)
    console.log('nextState:', nextState)
    console.log('组件调用shouldComponentUpdate')
    return true

}
```

此时我们再一次点击会怎么样？





状态是不是更新了，也调用了相关的函数了，调用顺序如输出的是一样的。

调用的是

shouldComponentUpdate,componentWillUpdate,render,componentDidUpdate这是在

组件内部改变的状态，

总结：如果我们不在shoudComponentUpdate里面返回true是不会更新组件的，

所以这个生命周期的函数可以作为性能优化的一个方面

那我们在组件外部改变props在看看组件调用的顺序

组件的调用顺序是

componentWillReceiveProps->shouldComponentUpdate->componentWillUpdate->render->componentDidUpdate

```
}
class Parent extends React.Component {
    constructor() {
        super()
        this.state = {
            name: '父传递下来的名字-蓝银草'
        }
    }
    handleClick() {
        this.setState({name: '父更新后的名字-蓝银皇'})
    }
    render() {
        return (
            <div>
                <Ele name={this.state.name} />
                <button onClick={this.handleClick.bind(this)}>点击父组件</button>
            </div>
        )
    }
}
ReactDOM.render(<Parent />, document.getElementById('app'));
```

代码截图可以看到前一个状态是的值与后一个的值，如果setState后的值是一样的会怎么样呢？

```
class Parent extends React.Component {
    constructor() {
        super()
        this.state = {
            name: '父传递下来的名字-蓝银草'
        }
    }
    handleClick() {
        this.setState({name: '父传递下来的名字-蓝银草'})
    }
    render() {
        return (
            <div>
                <Ele name={this.state.name} />
                <button onClick={this.handleClick.bind(this)}>点击父组件</button>
            </div>
        )
    }
}
```

```
preProps: ▶ {name: "父传递下来的名字-蓝银草"}
nextProps: ▶ {name: "父传递下来的名字-蓝银草"}
组件调用componentWillReceiveProps
组件调用shouldComponentUpdate
组件调用componentWillUpdate
组件调用了render
组件调用componentDidUpdate
>
```

组件仍然和以前以前进行了更新操作,现在我们在shouldComponentUpdate里面做

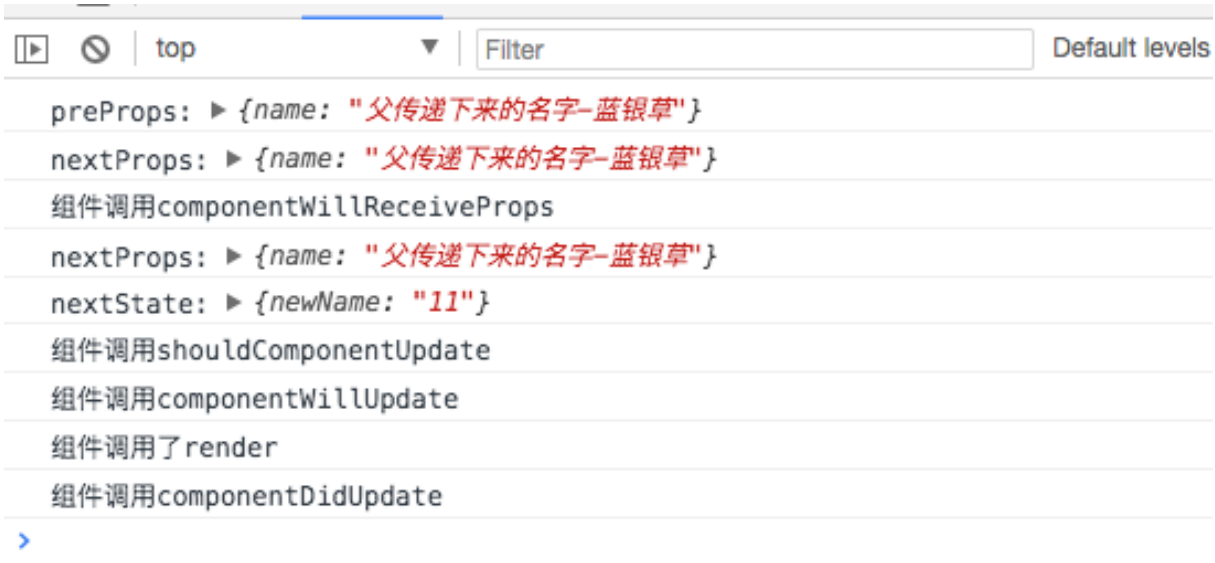
一些判断

先示范一个**错误**的比较方式

```
    }
    shouldComponentUpdate(nextProps, nextState) {
        console.log('nextProps:', nextProps)
        console.log('nextState:', nextState)
        console.log('组件调用shouldComponentUpdate')
        if(this.props === nextProps) {
            return false
        } else {
            return true
        }

    }
```

当我们这样写的时候 实际上永远是返回true的 因为两个对象是不会相等的。可

以看一下结果

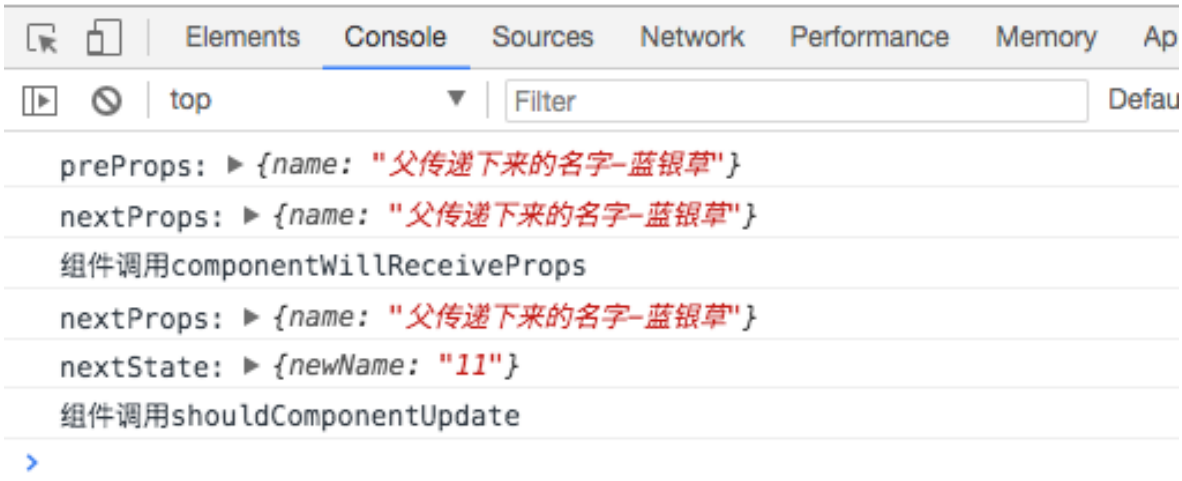


还是一样的更新了，这样该怎么办？

我们可以这样改一下

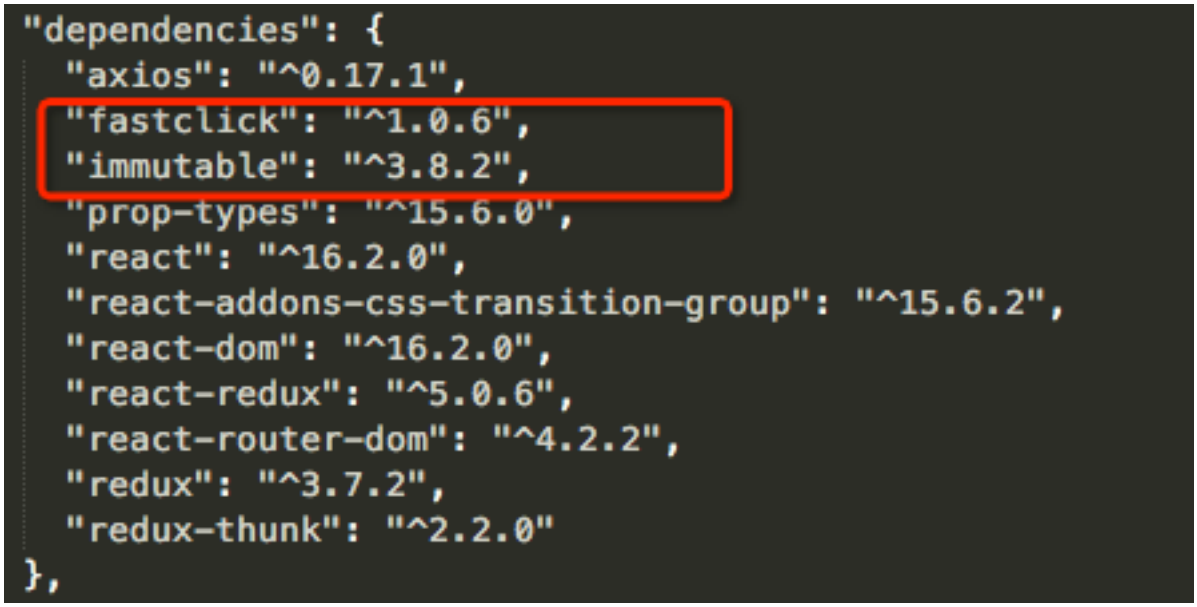
```
shouldComponentUpdate(nextProps, nextState) {
    console.log('nextProps:', nextProps)
    console.log('nextState:', nextState)
    console.log('组件调用shouldComponentUpdate')
    if(this.props.name === nextProps.name) {
        return false
    } else {
        return true
    }

}
```

我们既然知道是name属性可以直接写上去　在看下结果



此时调用了shouldComponentUpdate后不再调用其他的钩子函数了，就不在渲染了。

但是我们不知道里面有多少属性，或者可能还是一个引用。这个函数我们正常用的不会很多，考虑到后续的性能优化可以去使用。但是这个问题还是要解决吧。我们可以用immutable,我们在页面导入这个，**记得加上依赖**



import { is, fromJS } from 'immutable';

然后我们在shouldComponentUpdate里面这样写

```
shouldComponentUpdate(nextProps, nextState) {
  return !is(fromJS(this.props), fromJS(nextProps)) || !is(fromJS(this.state), fromJS(nextState))
}
```

这样就可以来判断了。

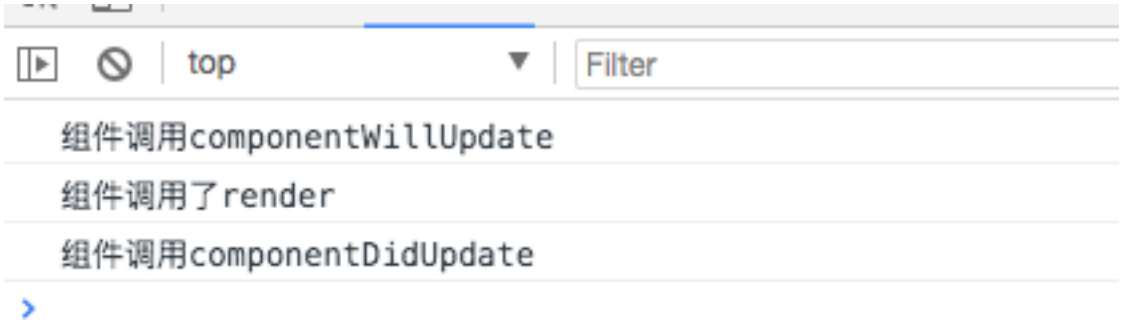
就像之前说的那样，如果你不用到shouldComponentUpdate函数的话，也就不需

要关心这些了

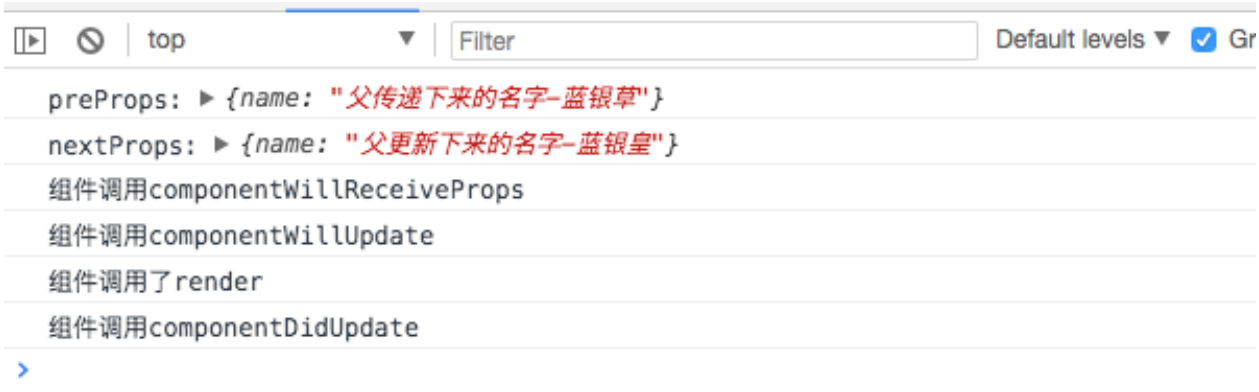
此时我们去掉了这个函数，看下结果

```
class Ele extends React.Component {
    constructor(props) {
        super(props)
        console.log('组件调用constructor')
        this.state = {
            newName: '11'
        }
        this.handleClick = this.handleClick.bind(this)
    }
    componentWillMount() {
        console.log('组件调用componentWillMount')
    }
    componentDidMount() {
        console.log('组件调用componentDidMount')
    }
    componentWillReceiveProps( nextProps) {
        console.log('preProps:', this.props)
        console.log('nextProps:', nextProps)
        console.log('组件调用componentWillReceiveProps')
    }

    componentWillUpdate() {
        console.log('组件调用componentWillUpdate')
    }
    componentDidUpdate() {
        console.log('组件调用componentDidUpdate')
    }
    componentWillUnmount() {
        console.log('组件调用componentWillUnmount')
    }
    handleClick() {
        this.setState({newName: '222'})
    }
    render() {
        console.log('组件调用了render')
        return (
            <div>
                111{this.props.name}
                <button onClick={this.handleClick}>更新状态</button>
                <p>这是状态更新后的{this.state.newName}</p>
            </div>
        )
    }
}
```

首先这是更新了状态



这是我们更新了父组件的props



所以我们平时对shouldComponentUpdate函数用的不多的话，我们也不要去写这个函数。也没有必要，如果真想做一些优化的话 这个是可以考虑的一个方面。