react 16生命周期

新特性

1. React16新的生命周期弃用了componentWillMount、componentWillReceivePorps，componentWillUpdate
2. 新增了getDerivedStateFromProps、getSnapshotBeforeUpdate来代替弃用的三个钩子函数（componentWillMount、componentWillReceivePorps，componentWillUpdate）
3. React16并没有删除这三个钩子函数，但是不能和新增的钩子函数（getDerivedStateFromProps、getSnapshotBeforeUpdate）混用，React17将会删除componentWillMount、componentWillReceivePorps，componentWillUpdate

   过时的生命周期 UNSAFE_componentWillMount()、 UNSAFE_componentWillReceiveProps()、

   UNSAFE_componentWillUpdate()
   名称将继续使用至 React 17。可以使用 rename-unsafe-lifecycles codemod 自动更新你的组件。
4. 新增了对错误的处理（componentDidCatch）

hook使用规则

不能在一个 class 组件中使用 Hook

只在最顶层使用 Hook

　不要在循环，条件或嵌套函数中调用 Hook， 确保总是在你的 React 函数的最顶层调用他们。

只在 React 函数中调用 Hook

新生命周期

Mounting（加载阶段：涉及4个钩子函数）

constructor()

```
加载的时候调用一次，可以初始化state
```

static getDerivedStateFromProps(props, state)

```
组件每次被rerender的时候，包括在组件构建之后(虚拟dom之后，实际dom挂载之前)，每次获取新的props或state之后；每次接收新的props之后都会返回一个对象作为新的st
```

这个新的函数主要致力于确保在需要state和props的时候是同步的，并致力于替换componentWillReceiveProps函数。这个函数将会在组件更新被调用同时也包括更新，就在constructor之后，所以你不再需要用constructor来根据props初始化state了。强烈不建议做有副作用的操作。

render()

```
react最重要的步骤，创建虚拟dom，进行diff算法，更新dom树都在此进行
```

componentDidMount()

```
组件渲染之后调用，只调用一次
```

Updating（更新阶段：涉及5个钩子函数)

static getDerivedStateFromProps(props, state).

```
组件每次被rerender的时候，包括在组件构建之后(虚拟dom之后，实际dom挂载之前)，每次获取新的props或state之后；每次接收新的props之后都会返回一个对象作为新的st
```

shouldComponentUpdate(nextProps, nextState)

```
组件接收到新的props或者state时调用，return true就会更新dom（使用diff算法更新），return false能阻止更新（不调用render）
```

render()

```
react最重要的步骤，创建虚拟dom，进行diff算法，更新dom树都在此进行
```

getSnapshotBeforeUpdate(prevProps, prevState)

```
getSnapshotBeforeUpdate() 在最近一次渲染输出（提交到 DOM 节点）之前调用。它使得组件能在发生
更改之前从 DOM 中捕获一些信息（例如，滚动位置）。此生命周期的任何返回值将作为参数传递给
componentDidUpdate()。

此用法并不常见，但它可能出现在 UI 处理中，如需要以特殊方式处理滚动位置的聊天线程等。

应返回 snapshot 的值（或 null）。
```

componentDidUpdate(prevProps, prevState, prevContext)

```
组件加载时不调用，组件更新完成后调用
```

Unmounting（卸载阶段：涉及1个钩子函数）

```
组件渲染之后调用，只调用一次
```

Error Handling(错误处理)

componentDidCatch(error，info)

```
任何一处的javascript报错会触发

此生命周期在后代组件抛出错误后被调用。 它接收两个参数：
error —— 抛出的错误。
info —— 带有 componentStack key 的对象，其中包含有关组件引发错误的栈信息。
```

- errorString——error调用`.toString()`方法后的值；
- errorInfo——一个拥有`componentStack`组件调用栈的对象，能够追溯到error在哪里发生。
-

```
import React, { Component } from 'react'

export default class NewReactComponent extends Component {
    constructor(props) {
        super(props)
        // getDefaultProps：接收初始props
        // getInitialState：初始化state
    }
    state = {
```

```
    }
      // 组件每次被rerender的时候，包括在组件构建之后(虚拟dom之后，实际dom挂载之前)，
      // 每次获取新的props或state之后；;每次接收新的props之后都会返回一个对象作为新的state，
      // 返回null则说明不需要更新state
    static getDerivedStateFromProps(props, state) {

        return state
    }
    componentDidCatch(error, info) { // 获取到javascript错误

    }
    render() {
        return (
            <h2>New React.Component</h2>
        )
    }
    componentDidMount() { // 挂载后

    }
    shouldComponentUpdate(nextProps, nextState) { // 组件Props或者state改变时触发，true：更新，false：不更新
        return true
    }
    getSnapshotBeforeUpdate(prevProps, prevState) { // 组件更新前触发

    }
    componentDidUpdate() { // 组件更新后触发

    }
    componentWillUnmount() { // 组件卸载时触发

    }
}
```