# React 生命周期

## constructor()

React借用class类的constructor充当初始化钩子。

React几乎没做什么手脚，但是因为我们只允许通过特定的途径给组件传递参数，所以constructor的参数实际上是被React规定好的。

React规定constructor有三个参数，分别是props、context和updater。

* props是属性，它是不可变的。
* context是全局上下文。
* updater是包含一些更新方法的对象，this.setState最终调用的是this.updater.enqueueSetState方法，this.forceUpdate最终调用的是this.updater.enqueueForceUpdate方法，所以这些API更多是React内部使用，暴露出来是以备开发者不时之需。

在React中，因为所有class组件都要继承自Component类或者PureComponent类，因此和原生class写法一样，要在constructor里首先调用super方法，才能获得this.

constructor生命周期钩子的最佳实践是在这里初始化this.state。

你也可以使用属性初始化器来代替，如下：

import React, { Component } from 'react';  
  
class App extends Component {  
 state = {  
 name: 'biu',  
 };  
}  
  
export default App;

## componentWillMount()

这是组件挂载到DOM之前的生命周期钩子。

很多人会有一个误区：这个钩子是请求数据然后将数据插入元素一同挂载的最佳时机。

其实componentWillMount和挂载是同步执行的，意味着执行完这个钩子，立即挂载。而向服务器请求数据是异步执行的。所以无论请求怎么快，都要排在同步任务之后再处理，这是辈分问题。

也就是说，永远不可能在这里将数据插入元素一同挂载。

## static getDerivedStateFromProps(props, state)

这是React v16.3.0发布的API。

首先，这是一个静态方法生命周期钩子。

也就是说，定义的时候得在方法前加一个static关键字，或者直接挂载到class类上。

简要区分一下实例方法和静态方法：

* 实例方法，挂载在this上或者挂载在prototype上，class类不能直接访问该方法，使用new关键字实例化之后，实例可以访问该方法。
* 静态方法，直接挂载在class类上，或者使用新的关键字static，实例无法直接访问该方法。

import React, { Component } from 'react';  
  
class App extends Component {  
 state = {  
 name: 11  
 }  
 render() {  
 // this.state = {name: 11, age: 15}  
 return (  
 <div>React</div>  
 );  
 }  
  
 static getDerivedStateFromProps(props, state) {  
 return {  
 age: 15  
 }  
 }  
}  
  
export default App;

这个生命周期钩子的使命是根据父组件传来的props按需更新自己的state，这种state叫做衍生state。返回的对象就是要增量更新的state。 它被设计成静态方法的目的是保持该方法的纯粹，它就是用来定义衍生state的，除此之外不应该在里面执行任何操作。 这个生命周期钩子也经历了一些波折，原本它是被设计成初始化、父组件更新和接收到props才会触发，现在只要渲染就会触发，也就是初始化和更新阶段都会触发。

## render()

作为一个组件，最核心的功能就是把元素挂载到DOM上，所以render生命周期钩子是一定会用到的。

render生命周期钩子怎么接收模板呢？当然是你return给它。

但是不推荐在return之前写过多的逻辑，如果逻辑过多，可以封装成一个函数。

render() {  
 // 这里可以写一些逻辑  
 return (  
 <div>  
 <input type="text" />  
 <button>click</button>  
 </div>  
 );  
}

注意，千万不要在render生命周期钩子里调用this.setState，因为this.setState会引发render，这下就没完没了了。

## componentDidMount()

这是组件挂载到DOM之后的生命周期钩子。

这可能是除了render之外最重要的生命周期钩子，因为这时候组件的各方面都准备就绪，天地任你闯。

## componentWillReceiveProps(nextProps)

componentWillReceiveProps生命周期钩子只有一个参数，更新后的props。

该声明周期函数可能在两种情况下被触发：

* 组件接收到了新的属性。
* 组件没有收到新的属性，但是由于父组件重新渲染导致当前组件也被重新渲染。

初始化时并不会触发该生命周期钩子。

## shouldComponentUpdate(nextProps, nextState)

这个生命周期钩子是一个开关，判断是否需要更新，主要用来优化性能。

有一个例外，如果开发者调用this.forceUpdate强制更新，React组件会无视这个钩子。

shouldComponentUpdate生命周期钩子默认返回true。也就是说，默认情况下，只要组件触发了更新，组件就一定会更新。React把判断的控制权给了开发者。

不过周到的React还提供了一个PureComponent基类，它与Component基类的区别是PureComponent自动实现了一个shouldComponentUpdate生命周期钩子。

对于组件来说，只有状态发生改变，才需要重新渲染。所以shouldComponentUpdate生命周期钩子暴露了两个参数，开发者可以通过比较this.props和nextProps、this.state和nextState来判断状态到底有没有发生改变，再相应的返回true或false。

## componentWillUpdate(nextProps, nextState)

shouldComponentUpdate生命周期钩子返回true，或者调用this.forceUpdate之后，会立即执行该生命周期钩子。 要特别注意，componentWillUpdate生命周期钩子每次更新前都会执行，所以在这里调用this.setState非常危险，有可能会没完没了

## getSnapshotBeforeUpdate(prevProps, prevState)

顾名思义，保存状态快照用的。

它会在组件即将挂载时调用，注意，是即将挂载。它甚至调用的比render还晚，由此可见render并没有完成挂载操作，而是进行构建抽象UI的工作。getSnapshotBeforeUpdate执行完就会立即调用componentDidUpdate生命周期钩子。

它是做什么用的呢？有一些状态，比如网页滚动位置，我不需要它持久化，只需要在组件更新以后能够恢复原来的位置即可。

getSnapshotBeforeUpdate生命周期钩子返回的值会被componentDidUpdate的第三个参数接收，我们可以利用这个通道保存一些不需要持久化的状态，用完即可舍弃。

## componentDidUpdate(nextProps, nextState, snapshot)

这是组件更新之后触发的生命周期钩子。

搭配getSnapshotBeforeUpdate生命周期钩子使用的时候，第三个参数是getSnapshotBeforeUpdate的返回值。 同样的，componentDidUpdate生命周期钩子每次更新后都会执行，所以在这里调用this.setState也非常危险，有可能会没完没了。

## componentWillUnmount()

这是组件卸载之前的生命周期钩子。

React的最佳实践是，组件中用到的事件监听器、订阅器、定时器都要在这里销毁。

当然我说的事件监听器指的是这种：

render(  
 return (  
 <button onClick={this.handle}>click</button>  
 );  
)

## componentDidCatch(error, info)

这是React v16.3.0发布的API。

它主要用来捕获错误并进行相应处理，所以它的用法也比较特殊。

定制一个只有componentDidCatch生命周期钩子的ErrorBoundary组件，它只做一件事：如果捕获到错误，则显示错误提示，如果没有捕获到错误，则显示子组件。

将需要捕获错误的组件作为ErrorBoundary的子组件渲染，一旦子组件抛出错误，整个应用依然不会崩溃，而是被ErrorBoundary捕获。

## 生命周期

这么多生命周期钩子，实际上总结起来只有三个过程：

* 挂载
* 更新
* 卸载

挂载和卸载只会执行一次，更新会执行多次。

一个完整的React组件生命周期会依次调用如下钩子

1. 挂载
   * constructor
   * componentWillMount
   * render
   * componentDidMount
2. 更新
   * componentWillReceiveProps
   * shouldComponentUpdate
   * componentWillUpdate
   * render
   * componentDidUpdate
3. 卸载
   * componentWillUnmount