目录

[基础总结 2](#_Toc535249926)

[概念 2](#_Toc535249927)

[生命周期 2](#_Toc535249928)

[constructor 3](#_Toc535249929)

[componentWillMount/DidMount 3](#_Toc535249930)

[componentWillReceiveProps 4](#_Toc535249931)

[shouldComponentUpdate 4](#_Toc535249932)

[componentWill/Did Update 4](#_Toc535249933)

[setState 4](#_Toc535249934)

[prop 5](#_Toc535249935)

[组件与标签 5](#_Toc535249936)

[事件/属性 5](#_Toc535249937)

[嵌套 5](#_Toc535249938)

[父子组件更新顺序 5](#_Toc535249939)

[setState 5](#_Toc535249940)

[state更新时间 6](#_Toc535249941)

[setTimeOut 6](#_Toc535249942)

[原生事件 6](#_Toc535249943)

[“异步”方式会导致生命周期更新吗？ 7](#_Toc535249944)

[代码 7](#_Toc535249945)

[事件 8](#_Toc535249946)

[事件绑定 8](#_Toc535249947)

[16.4 9](#_Toc535249948)

[getDerivedStateFromProps(nextProps, preState) 9](#_Toc535249949)

[getSnapshotBeforeUpdate(preProps,preState) 9](#_Toc535249950)

[diff 10](#_Toc535249951)

[virtualDOM 10](#_Toc535249952)

[对象 10](#_Toc535249953)

[key 10](#_Toc535249954)

[事件代理 11](#_Toc535249955)

[渲染 11](#_Toc535249956)

[setState 11](#_Toc535249957)

[setState 11](#_Toc535249958)

[其他 11](#_Toc535249959)

[className 11](#_Toc535249960)

# 基础总结

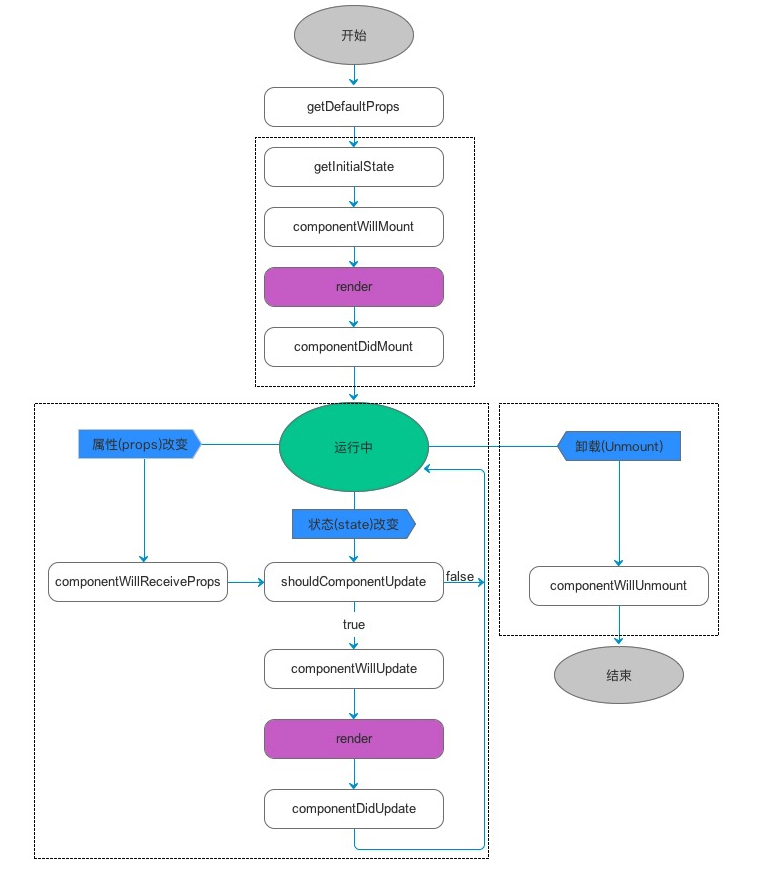
## 概念

派生状态： 依赖于prop值的state。prop赋值给state的过程不要放在生命周期里。（组件重新渲染时，会丢失保存的state。)

同时需要设置key，在tabs中可能导致tab组件共享 https://codesandbox.io/s/mz2lnkjkrx。或者提供重置的方法。

## 生命周期

注意组件第一次渲染和重新渲染，执行的是不同的生命周期。



### constructor

在WillMount之前。

### componentWillMount/DidMount

只第一次渲染时会执行。

为什么不在WillMount里请求数据？

因为请求是异步的，实际上异步执行完之前，就已经render了（同步代码继续向下执行），执行完之后通过setState进行重新渲染，和在DidMount里执行效果是一样的，还可能因为操作没渲染的dom出错。在componentWillMount里发起AJAX，不管多快得到结果也赶不上首次render。

### componentWillReceiveProps

第一次渲染时不会执行这个函数。

即使自身state更新导致的重新渲染，也会执行这个函数。

在这个函数里setState不会导致二次渲染，但是setState是有效的，或者说是将两次渲染合并到一起了。就算componentWillReceiveProps()中执行了this.setState，更新了state(在最后合并一起更新)，但在render前（如shouldComponentUpdate，componentWillUpdate），this.state依然指向更新前的state。

是以前唯一更新状态不导致重新渲染的方法。

### shouldComponentUpdate

### componentWill/Did Update

在should之前state就已经合并了。

不可在其中使用setState。

will (nextProps, preState)

Did(preProps, preState)(已经能拿到新的prop)

componentDidUpdate(prevProps, prevState, snapshot)

## setState

详见setState

## prop

每当父组件重新render导致的重传props，子组件将直接跟着重新渲染，无论props是否有变化。

## 组件与标签

### 事件/属性

className之类属性用在组件上时，像普通属性一样传递。

### 嵌套

## 父子组件更新顺序

### 第一次渲染

所有代码都是同步的情况：

F.render

C.render

C.componentDidMount

F.componentDidMount

### 更新

# setState

setState(obj | fun, cb)

默认只要调用了就会重新渲染，如果使用了PureComponent,会进行浅层比较，如果是同一个引用，就不会重新渲染。

一般不要在setState里用state和props，因为值是不可控的。应该用传函数的方式（preState, props） => {}。

## state更新时间

setState有普通模式和batch模式，一般是batch模式。

batch模式就是进行了正常更新过程，进行了状态合并的过程。（怎么判断：调用setstate时里面有个标识，正常是true，进入）

### setTimeOut

在setTimeout里调用时，使用普通模式。在原生事件中setState后，能同步拿到更新后的state值。。

代码块执行到 setTimeout 的时候，浏览器把它丢到列队（事件循环）里，并没有去执行，而是先执行的 finally 代码块，等 finally 执行完了， isBatchingUpdates 又变为了 false ，导致最后去执行队列里的 setState 时候， requestWork 走的是和原生事件一样的 expirationTime === Sync if分支，所以表现就会和原生事件一样，可以同步拿到最新的state值。

大概明白了，react更新时把isBatchUpdates 状态修改了，所有的setState放到queue中，统一render flush。。。

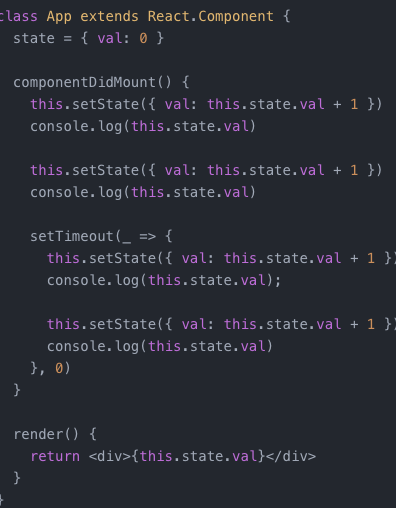
而在setTimeout这个isBatchingUpdates没法在生命周期里设为true，直接就同步更新了。

### 原生事件

回调函数里拿到的是最新的state。

### “异步”方式会导致生命周期更新吗？

### 代码



输出: 0 0 2 3

为什么不是: 0 1 2 3 ; 0 0 2 2

# 事件

react为了解决跨平台，兼容性问题，自己封装了一套事件机制，代理了原生的事件，像在jsx中常见的onClick、onChange这些都是合成事件。

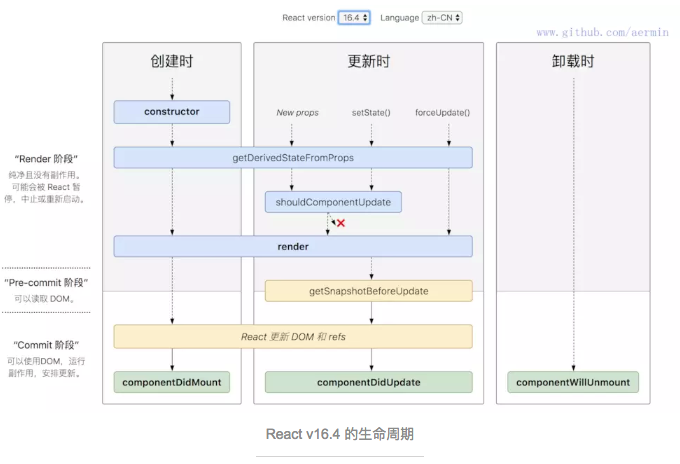
原生事件是指非react合成事件，原生自带的事件监听 addEventListener ，或者也可以用原生js、jq直接 document.querySelector().onclick 这种绑定事件的形式都属于原生事件。

在原生事件中setState后，能同步拿到更新后的state值。

## 事件绑定

有匿名函数和普通函数两种。匿名函数在每次重新渲染时会当成新的属性来处理（在传递时用.bind也是类似）。普通函数会缓存起来，提高性能。

# 16.4



除了shouldComponentUpdate，其他在render函数之前的所有函数（componentWillMount，componentWillReceiveProps，componentWillUpdate）都被getDerivedStateFromProps替代。

### getDerivedStateFromProps(nextProps, preState)

static getDerivedStateFromProps(props, state)

返回一个对象来更新state，或者返回null来表示新的props不需要任何state更新。

这个新的生命周期应该覆盖传统componentWillMount，componentWillReceiveProps的所有用例。

### getSnapshotBeforeUpdate(preProps,preState)

普通方法。

返回值将作为第三个参数传递给componentDidUpdate。 （这个生命周期不是经常需要的，但可以用于在恢复期间手动保存滚动位置的情况。）

与componentDidUpdate一起，这个新的生命周期将覆盖旧版componentWillUpdate的所有用例。

# diff

## virtualDOM

render 执行的结果得到的不是真正的 DOM 节点,而是js对象,我们称之为 virtual DOM.

### 对象



## key

没有key，diff时通过顺序去对比，有key,通过key去对比。

## 事件代理

## 渲染

### setState

参考上面。

# 其他

## className

不能给自定义组件添加className,style

可以将className作为属性传递，在子组件中使用。