

Class 组件详解

React 全解

版权声明

本内容版权属杭州饥人谷教育科技有限公司（简称饥人谷）所有。

任何媒体、网站或个人未经本网协议授权不得转载、链接、转贴，或以其他方式复制、发布和发表。

已获得饥人谷授权的媒体、网站或个人在使用时须注明「资料来源：饥人谷」。

对于违反者，饥人谷将依法追究 responsibility。

联系方式

如果你想要购买本课程

请微信联系 [xiedaimala02](#) 或 [xiedaimala03](#)

如果你发现有人盗用本课程

请微信联系 [xiedaimala02](#) 或 [xiedaimala03](#)

知识点

- 英语小课堂
- 创建方式
- props 和 state
- 生命周期

英语小课堂

英语	翻译	英语	中文
derived	导出的，衍生的，派生的	property	属性
render	渲染	state	状态
super class	超类、父类	mount	挂载

两种方式创建 Class 组件

- ES5 方式（过时）

```
import React from 'react'
```

```
const A = React.createClass({  
  render() {  
    return (  
      <div>hi</div>  
    )  
  }  
})
```

```
export default A
```

```
// 由于 ES5 不支持 class，才会有这种方式
```

两种方式创建 Class 组件

- ES6 方式

```
import React from 'react';
```

```
class B extends React.Component {  
  constructor(props) {  
    super(props);  
  }  
  render() {  
    return (  
      <div>hi</div>  
    )  
  }  
}
```

```
export default B;
```

// extends、constructor、super 强行记忆，别问为什么

哪种好

- ES6 的 class 方式好
 - ✓ 以后我们只用这种方式创建类组件
- 浏览器不支持 ES6 怎么办
 - ✓ 你公司怎么还在搞 IE 8，肯定是个破公司，跳槽吧
 - ✓ 用 webpack + babel 将 ES6 翻译成 ES5 即可

Props

外部数据

Props 外部数据

- 传入 props 给 B 组件

```
class Parent extends React.Component {  
  constructor(props){  
    super(props)  
    this.state = {name:'frank'}  
  }  
  onClick = ()=>{}  
  render(){  
    return <B name={this.state.name}  
              onClick={this.onClick}>hi</B>  
  }  
}
```

// 外部数据被包装为一个对象
// {name:'frank', onClick:..., children:'hi'}
// 此处的 onClick 是一个回调
// 不知道回调是什么的同学自己复习一下[系统班课程](#)

Props 外部数据

- 初始化

```
class B extends React.Component {  
  constructor(props) {  
    super(props);  
  }  
  render(){}  
}
```

要么不初始化，即不写 constructor

要么初始化，且必须写全套（不写 super 直接报错）

- 效果

✓ 这么做了之后，this.props 就是外部数据对象的地址了

Props 外部数据

- 读取

```
class B extends React.Component {  
  constructor(props) {  
    super(props);  
  }  
  render(){  
    return <div onClick={this.props.onClick}>  
      {this.props.name}  
      <div>  
        {this.props.children}  
      </div>  
    </div>  
  }  
}  
// 通过 this.props.xxx 读取
```

读 props 会了，写 props 呢？

不准写 props! ! !

写 Props

- 改 props 的值（一个地址）

- ✓ `this.props = { /* 另一个对象 */ }`
- ✓ 不要写这样的代码，没有意义
- ✓ 理由：既然是外部数据，就应该由外部更新

- 改 props 的属性

- ✓ `this.props.xxx = 'hi'`
- ✓ 不要写这样的代码，没有意义
- ✓ 理由：既然是外部数据，就不应该从内部改值

- 原则

- ✓ 应该由数据的**主人**对数据进行更改

相关钩子

- **componentWillReceiveProps 钩子**
 - ✓ 当组件接受新的 props 时，会触发此钩子
 - ✓ 啥是钩子？黑话，你就将其翻译成「特殊函数」即可
 - ✓ 示例，这个钩子的作用写在文档里了（我没用过）
- **该钩子已经被弃用**
 - ✓ 更名为 UNSAFE_componentWillReceiveProps
 - ✓ 总而言之，不要使用这个钩子
- **老师，你有病吗？**
 - ✓ 不推荐使用你还教？
 - ✓ 教是为了让你能看懂旧代码啊，这钩子以前是推荐用的

Props 的作用

- 接受外部数据

- ✓ 只能读不能写
- ✓ 外部数据由父组件传递

- 接受外部函数

- ✓ 在恰当的时机，调用该函数
- ✓ 该函数一般是父组件的函数

State & setState

内部数据

初始化 State

- 代码

```
class B extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      user: {name: 'frank', age: 18}  
    }  
  }  
  render() { /* ... */ }  
}
```

读写 State

- 读用 `this.state`

- ✓ `this.state.xxx.yyy.zzz`

- 写用 `this.setState(???, fn)`

- ✓ `this.setState(newState, fn)`

- ✓ 注意 `setState` 不会立刻改变 `this.state`, 会在当前代码运行完后, 再去更新 `this.state`, 从而触发 UI 更新

- ✓ `this.setState((state, props) => newState, fn)`

- ✓ 这种方式的 `state` 反而更易于理解

- ✓ `fn` 会在写入成功后执行

- 写时会 `shallow merge`

- ✓ `setState` 会自动将新 `state` 与旧 `state` 进行一级合并

不守规矩

- 修改 `this.state` 的属性值

- ✓ `this.state.n += 1`

- ✓ `this.setState(this.state)`

- ✓ 不推荐用，但能用（因为 React 没法阻止你这么写）

生命周期

Lifecycle

生命周期

- 类比如下代码

- ✓ `let div = document.createElement('div')`
- ✓ `// 这是 div 的 create / construct 过程`
- ✓ `div.textContent = 'hi'`
- ✓ `// 这是初始化 state`
- ✓ `document.body.appendChild(div)`
- ✓ `// 这是 div 的 mount 过程`
- ✓ `div.textContent = 'hi2'`
- ✓ `// 这是 div 的 update 过程`
- ✓ `div.remove()`
- ✓ `// 这是 div 的 unmount 过程`

- 同理

- ✓ React 组件也有这些过程，我们称之为生命周期

生命周期

- 函数列表

- ✓ `constructor()`
- ✓ `static getDerivedStateFromProps()`
- ✓ `shouldComponentUpdate()`
- ✓ `render()`
- ✓ `getSnapshotBeforeUpdate()`
- ✓ `componentDidMount()`
- ✓ `componentDidUpdate()`
- ✓ `componentWillUnmount()`
- ✓ `static getDerivedStateFromError()`
- ✓ `componentDidCatch()`

生命周期 - 必会

- 函数列表

- ✓ `constructor()` - 在这里初始化 state
- ✓ `shouldComponentUpdate()` - return false 阻止更新
- ✓ `render()` - 创建虚拟 DOM
- ✓ `componentDidMount()` - 组件已出现在页面
- ✓ `componentDidUpdate()` - 组件已更新
- ✓ `componentWillUnmount()` - 组件将死

constructor

- 用途

- ✓ 初始化 props
- ✓ 初始化 state, 但此时不能调用 setState
- ✓ 用来写 bind this

```
constructor(){  
    /* 其他代码略*/  
    this.onClick = this. onClick.bind(this)  
}
```

可以用新语法代替

```
onClick = ()=> {}  
constructor(){ /* 其他代码略*/ }
```

- ✓ 可不写

shouldComponentUpdate

- 用途

- ✓ 返回 true 表示不阻止 UI 更新
- ✓ 返回 false 表示阻止 UI 更新
- ✓ 示例见下页

- 面试常问

- ✓ 问：shouldComponentUpdate 有什么用？
- ✓ 答：它允许我们手动判断是否要进行组件更新，我们可以根据应用场景灵活地设置返回值，以避免不必要的更新

示例代码

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      n: 1
    };
  }
  onClick = () => {
    this.setState(state => ({ n: state.n + 1 }));
    this.setState(state => ({ n: state.n - 1 }));
  };
  shouldComponentUpdate(newProps, newState){
    if(newState.n === this.state.n){
      return false
    }else{
      return true
    }
  }
  render() {
    console.log('render 了一次')
    return (
      <div>
        {this.state.n}
        <button onClick={this.onClick}>+1-1</button>
      </div>
    );
  }
}
```

启发:

其实可以将
newState 和 this.state 的
每个属性都对比一下
如果全都相等，就不更新
如果有一个不等，就更新

再启发:

为什么 React 不内置此功能?
React 确实内置了，
这个功能叫做
React.PureComponent，
可以代替 React.Component

质疑:

为什么要用新的对象?
为什么我们不直接在
this.state 身上改呢?

这触及到编程哲学了，以后聊

render

- 用途

- ✓ 展示视图

```
return (<div>...</div>)
```

- ✓ 只能有一个根元素
- ✓ 如果有两个根元素，就要用 `<React.Fragment>` 包起
- ✓ `<React.Fragment />` 可以缩写成 `<></>`

- 技巧

- ✓ render 里面可以写 `if...else`
- ✓ render 里面可以写 `?:` 表达式
- ✓ render 里面不能直接写 `for` 循环，需要用数组
- ✓ render 里面可以写 `array.map` (循环)

componentDidMount()

- 用途

- ✓ 在元素插入页面后执行代码，这些代码依赖 DOM
- ✓ 比如你想获取 div 的高度，就最好在这里写
- ✓ 此处可以发起加载数据的 AJAX 请求（官方推荐）
- ✓ 首次渲染会执行此钩子

- 参数

- ✓ 看文档

componentDidUpdate()

- 用途

- ✓ 在视图更新后执行代码
- ✓ 此处也可以发起 AJAX 请求，用于更新数据（看文档）
- ✓ 首次渲染不会执行此钩子
- ✓ 在此处 setState 可能会引起无限循环，除非放在 if 里
- ✓ 若 shouldCom...Update 返回 false，则不触发此钩子

- 参数

- ✓ 看文档

componentWillUnmount

- 用途

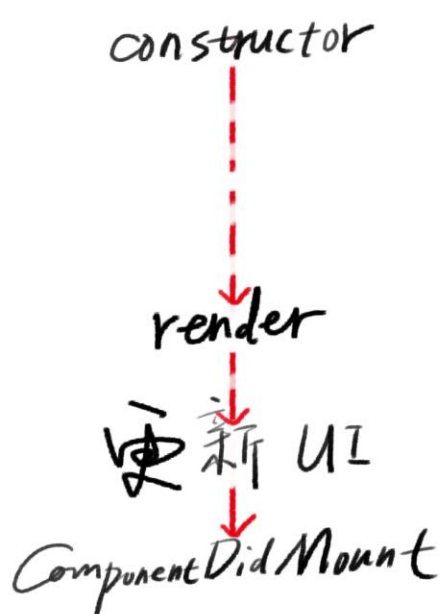
- ✓ 组件将要被移出页面然后被销毁时执行代码
- ✓ unmount 过的组件不会再次 mount

- 举例

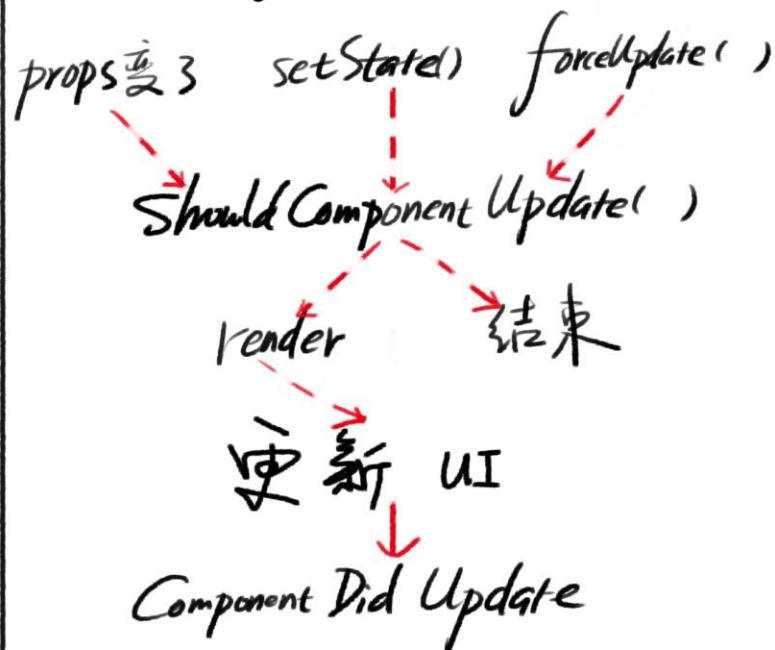
- ✓ 如果你在 c..DidMount 里面监听了 window scroll
- ✓ 那么你就要在 c..WillUnmount 里面取消监听
- ✓ 如果你在 c..DidMount 里面创建了 Timer
- ✓ 那么你就要在 c..WillUnmount 里面取消 Timer
- ✓ 如果你在 c..DidMount 里面创建了 AJAX 请求
- ✓ 那么你就要在 c..WillUnmount 里面取消请求
- ✓ 否则你就是菜逼
- ✓ 原则：谁污染谁治理

分阶段看钩子执行顺序

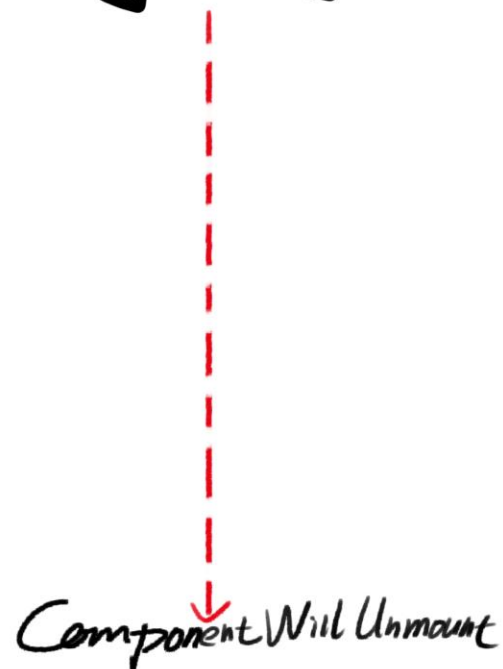
首次渲染



再次渲染



销毁



生命周期回顾

- 函数列表

- ✓ `constructor()` - 在这里初始化 state
- ✓ `shouldComponentUpdate()` - return false 阻止更新*
- ✓ `render()` - 创建虚拟 DOM
- ✓ `componentDidMount()` - 组件已出现在页面
- ✓ `componentDidUpdate()` - 组件已更新
- ✓ `componentWillUnmount()` - 组件将死

*不要忘了 return true

再见

希望你背下常用生命周期的拼写和用途