React细节知识点 - React篇

1. setState() 是异步的

之前setState({})写法

```
//例子
handleBtnClick() {
  this.setState({
    list: [...this.state.list, this.state.inputValue],
    inputValue: ''
  });
}
```

现在setState()写法

```
//例子
handleBtnClick() {
  this.setState((prevState) => {
    return {                          --->return可以省略，写法下面这个
      list: [...prevState.list, prevState.inputValue],
      inputValue: ''
    }
  });
}

//如果想省去return的话，那么这么写(这种写法会自动return 对象)：在圆括号里再花括号
handleBtnClick() {
  this.setState((prevState) => ({       --->增加了个圆括号
    list: [...prevState.list, prevState.inputValue],
    inputValue: ''
  }));
}

//但是新的setState()写法，需要注意一个点
//例子
handleInputChange(e) {
  const value = e.target.value; //这个e.target.value需要现在外面赋给一个变量，再在
  this.setState(() => ({        //  this.setState里使用，如果直接在setState里
    inputValue: value           // inputValue: e.target.value 会报错
  }))
}
```

```
//查看官网后发现setState((state, props) => {})里有两个参数，一个是修改数据前的状态
prevState和当前的props
```

ES6中语法

```
this.setState(() => ({
  list: list    // 在es6中，如果是这样的，那么可以简写成
}))
简写：
this.setState(() => ({
  list
}))
```

——

2. 事件.bind(this)

1. **给标签绑定事件的时候写法**

```
//1. 如果没有bind(this),这时候打印this会显示undefined
//2. bind(this)后，这个事件就是绑定在当前这个组件上的
<button onClick={this.handleBtnClick.bind(this)}>新增</button>

//优化代码写法：
render() {
  //上面这里bind(this)了，下面用的时候就不用bind(this)了，会提高一些性能
  const { this.handleBtnClick } = this.handleBtnClick.bind(this);
  return (
    <div>
      <button onClick={this.handleBtnClick}>新增</button>
    </div>
  )
}
```

2. **父组件传方法给子组件的时候写法**

```
getTodoItem() {
  return this.state.list.map((item, index) => {
    return (
      <div>
        <TodoItem>
          deleteItem={this.handleItemDelete}  //这里一定要记得bind(this)，把方法
        </TodoItem>                           //绑在父组件上。因为在子组件里使用这
个
      </div>                                  //方法时这么写：this.handleItemDelete()
    )                     //如果this没绑定父组件的话，在子组件里this肯定是指向子组件的
  })                      //但是这个方法是在父组件里写的，子组件里没有，所以就会报错
}
```

3. **bind(this)写在constructor函数里面**

```
constructor(props) {
  super(props);
  this.state = {
    inputValue: '',
    list: []
  }
  this.handleInputChange = this.handleInputChange.bind(this);
  this.handleBtnClick = this.handleBtnClick.bind(this);
}
```

4. **总结：给标签绑定事件时候，一定要把事件绑定在当前组件上，也就是事件.bind(this)**

——

3. PropTypes 和 defaultProps

1. **子组件接收父组件传过来的值，我们希望用PropTypes强校验下**

```
//导入
import PropTypes from 'prop-types'

TodoItem.propTypes = {          //TodoItem是子组件名字
    //content是父组件传过来的值，要求content是string类型或者number类型都可以
  content: PropTypes.oneOfType([PropTypes.number, PropTypes.string])
  deleteItem: PropTypes.func, //deleteItem是一个函数
  index: PropTypes.number,    //index是数字类型
  test: PropTypes.string.isRequired  //父组件没有传这个值，但是我们要求isRequired，
所以需要设置defaultProps不然会报错
}
```

2. **defaultProps 设置默认值**

```
TodoItem.defaultProps = {
  test: 'hello world'
}
```

——

4. Props, state与render函数的关系

1. **当组件的state或者props发生改变的时候，render函数就会重新执行**

```
//在TodoList组件里加个Test子组件，这个Test子组件就是直接在输入框下面显示输入的内容
//每次在输入框输入一个字符，控制台就会输出render，test render，说明因为state里的
inputValue状态发生了变化，所以父组件重新渲染，因为子组件接收父组件传过来的props里的
inputValue,所以也会重新渲染子组件
Test.js:
import React, { Component } from 'react';

class Test extends Component {
  render() {
    console.log('test render');
    return <div>{ this.props.content }</div>
  }
}
export default Test;

父组件中:
```

```
...
console.log('render');
...
<Test content={this.state.inputValue} />
```

——

5. 深入了解虚拟DOM

1. **JSX --> JS对象 --> DOM**

```
return <div><span>item</span></div>
上面这段代码，react会帮我们变成js对象：
上面和下面 是等价的
return React.createElement('div', {}, React.createElement('span', {},
'item'))
```

——

6. React中获取dom节点：e 或者 ref

1. **e**

```
e.target.value
```

2. **ref**

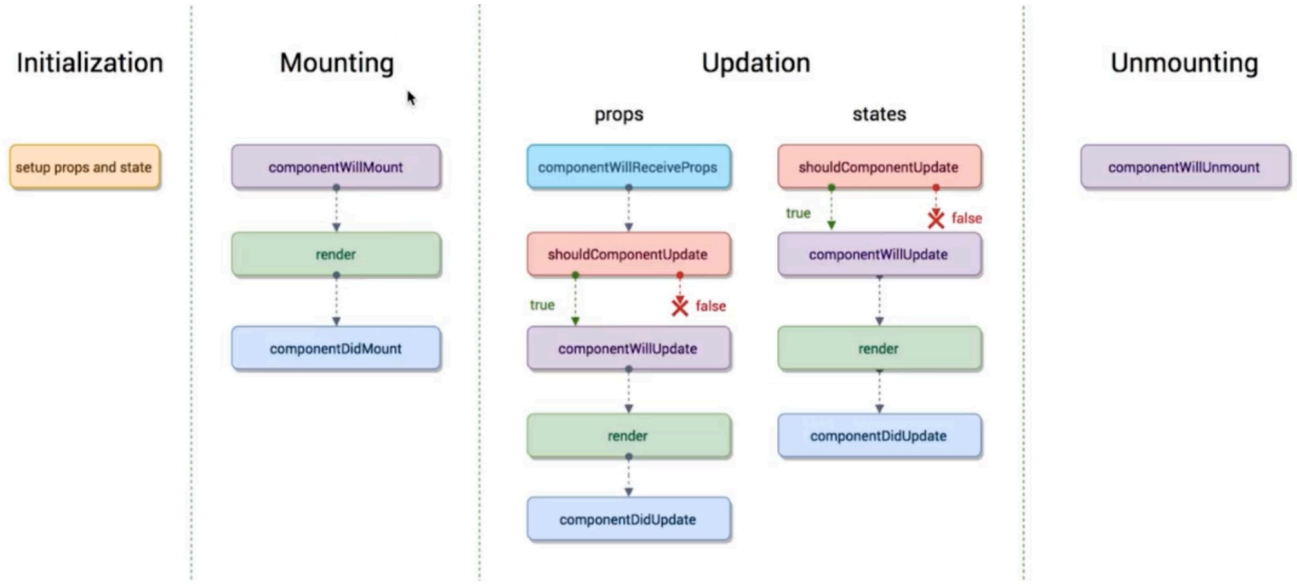
```
//这样就能获取到input这个dom节点
<input value={this.state.inputValue} ref={(input) => {this.input = input}}
onChange={this.handleInputChange.bind(this)}/>

//之前使用e.target.value写这个onChange函数
handleInputChange() {
  const value = e.target.value;
  this.setState(() => ({
    inputValue: value
  }));
}
```

```
//使用ref的写法
handleInputChange() {
  const value = this.input.value;     //input这个Dom节点的value属性
  this.setState(() => ({
    inputValue: value
  }));
}
```

---

7. React的生命周期函数 & Ajax异步请求



1. **生命周期函数指在某一个时刻组件会自动调用执行的函数**
2. **Mounting部分的componentWillMount和componentDidMount函数只在第一个挂载到页面时候执行，之后重新渲染页面只会执行render这个周期函数；而Updation部分的生命周期函数只要state或者props发生了变化生命周期函数就会被执行**
3. **componentWillReceivedProps比较特殊，需要满足下面两个条件才会执行**
    1. 一个组件要从父组件接收参数
    2. 只要父组件的render函数被执行了，子组件的这个生命周期函数就会被执行
    3. 第二点另种表述方式：如果这个组件第一次存在于父组件中，不会执行，如果这个组件之前已经存在于父组件中，才会执行
4. **所有生命周期函数除了render函数都可以不存在，但是render这个生命周期函数必须存在**
5. **为什么其他生命周期函数都可以不存在呢？这是因为我们的组件一开头TodoList extends Component，在Component已经内置了除了render以外的所有生命周期函数**

6. **在输入框里输入字符，但是还没点击新增，我们发现父组件render了，因为父组件里的inputValue改变了，但是我们也发现子组件也render了，但是我们每点击新增，其实传给子组件的props中InputValue值没变，这里就性能浪费了。如何解决呢?**
    1. **解决方法：在子组件的shouldComponentUpdate生命周期函数里做个判断，这样就可以避免子组件做无谓的render操作了** (父组件里是以content = {this.state.inputValue}传过来值的)

```
shouldComponentUpdate(nextProps, nextState) {
  if (nextProps.content !== this.props.content) {
    return true;
  } else {
    return false;
  }
}
```

7. **一般把ajax异步请求放在componentDidMount生命周期函数里**
    1. react没有提供发送ajax请求的库，我们需要自己安装第三方库，yarn add axios

```
import axios from 'axios';

//ajax请求数据，获取到数据后放入state里
componentDidMount() {
  axios.get('/api/todolist')          --->与后端约定好的地址
  .then((res) => {
    this.setState(() => {
      return {
        list: res.data    //或者 list: [...res.data]
      }
    });
  })
  .catch(() => {alert('失败')})
}
```

---

8. React-transition-group实现动画

1. 当我们没有使用这个第三方库时，我们需要自己给标签写className

```
<div className={this.state.show ? 'show' : 'hide'}>Hello World!</div>
```

当我们使用了这个第三方库之后，我们不需要自己给标签写className，不用负责样式的添加删除；这个库会自动帮我们做className的添加和删除

```
//1.安装这个第三方库
yarn add react-transition-group
//2.导入这个库,其实CSSTransition就是一个动画组件
import { CSSTransition } from 'react-transition-group';
//3.使用
<CSSTransition       //需要通过this.state.show来感知什么时候入场、出场(入场出场状态)
  in={this.state.show}
  timeout={1000}     //时间
  classNames='fade'  //后面是用fade-enter，用谁开头 这边className就写谁
  unmountOnExit      //完成出场动画后，去除挂载
  onEntered={(el)=>{ el.style.color='blue'}}  //onEnter钩子函数(当入场动画完成后)
  appear={true}      //第一次展示也要动画效果的话 在css文件中增加fade-appear,
  />                 //fade-appear-active, fade-appear-active
  <div>Hello World!s</div>
</CSSTransition>
```

2. fade-enter、fade-enter-active、fade-enter-done

```
.fade-enter .fade-appear{         //在入场动画执行的第一个时刻
  opacity: 0;
}

.fade-enter-active .fade-appear-active{ //入场动画执行的第一个时刻到入场动画执行完前一时刻
  opacity: 1;
  transition: opacity 1s ease-in;
}

.fade-enter-done {     //整个入场动画执行完后
  opacity: 1;
  color: blue;        --> 也可以通过js的形式实现
}                     --> onEntered={(el)=>{ el.style.color='blue'}}

.fade-exit {           //出场动画执行的第一个时刻
  opacity: 1;
}

.fade-exit-active {    //出场动画执行的一个时刻后到执行完之前一个时刻
  opacity: 0;
  transition: opacity 1s ease-in;
}
```

```
.fade-exit-done {       //出场动画执行完后
  opacity: 0;
}
```

3. **多个dom节点都要css动画的时候**

```
//使用TransitionGroup
import { CSSTransition, TransitionGroup } from 'react-transition-group';
//最外层使用<TransitionGroup>，每个dom也还是要<CSSTransition>
<TransitionGroup>
{
  this.state.list.map((item, index) => {
    return (
      <CSSTransition
        //in={this.state.show}      --->这个不需要了
        timeout={1000}
        classNames='fade'
        unmountOnExit
        onEntered={(el) => {el.style.color='blue'}}
        appear={true}
        key={index}                 --->key值写在这里
        >
        <div>{item}</div>
      </CSSTransition>
    )
  })
}
</TransitionGroup>
```