01. 组件生命周期函数的对比

有哪些生命周期
在那些阶段执行
执行次数
能否调动this.setState

生命周期	调用次数	能否使用 setState()
static defaultProps = { }	其全局调用一次	否
this.state = { }	1	否
componentWillMount	1	是
render	>=1	否
componentDidMount	1	是
componentWillReceiveProps	>=0	是
shouldComponentUpdate	>=0	否
componentWillUpdate	>=0	否
componentDidUpdate	>=0	否
componentWillUnmount	1	否

02. 使用defaultProps设置组件的默认属性值

1、创建两个count组件 传递参数和不传递参数
2 、在封装一个组件的时候，组件内部，肯定有一些数据是必须的，哪怕用户没有传递一些相关的启动参数，这时候，组件内部 ，尽量 给自己提供一个默认值；
3、在 React 中，使用静态的 defaultProps 属性，来设置 组件的 默认属性值；

通过 static defaultProps = {}设置默认值，defaultProps 固定写法如图1

```
static defaultProps = {
  initcount: 0 // 如果外界没有传递 initcount，那么，自己初始化一个 数值，为0
}
```


03. 使用prop-types进行props属性值的类型校验

问题：传递进组件的值类型不确定会报错咋办？
1、校验，用propTypes 静态属性校验
2、安装一个包prop-type这个包

注意：prop-type职能单一，只能校验数据类型
参数类型错误会报警 如图2

用法总结：在需要接受参数的组件中 定义一个propType的静态属性 固定单词 在这个对象内部设置 数据与数据类型的映射 如图1

```
// 这是创建一个 静态的 propTypes 对象，在这个对象中，可以把 外界传递过来的属性，做类型校验；
// 注意， 如果要为 传递过来的属性做类型校验，必须安装 React 提供的 第三方包，叫做 prop-types ；
// prop-types 大概在 v.15.* 之前，并没有单独抽离出来，那时候，还和 react 包 在一起；
//后来， 从 v.15.* 之后，官方把类型校验的 模块，单独抽离为 一个包，就叫做 prop-types
static propTypes = {
  initcount: PropTypes.number // 使用 prop-types 包，来定义 initcount 为 number 类型
}
```

```
▶Warning: Failed prop type: Invalid prop `initcount` of type `string` supplied to
`Counter`, expected `number`.
    in Counter
```

2

04. 介绍componentWillMount函数

在组件即将挂载到页面上的时候执行，此时，组件尚未挂载到页面中呢
虚拟DOM是否创建好了呢？此时，内存中的虚拟DOM也没开始创建呢
ComponentWillMount执行时：获取不到dom
可以拿到props和state
定义一个自定义函数，在componentwillmount中也可以调用，等同于vue中的created
总结：主要用于数据和自定义方法的初始化

05. 组件创建阶段的render函数和componentDidMount函数说明

Render中dom也未渲染，也拿不到dom
Render函数执行完成后，虚拟dom创建完成，同样未渲染
Render-----》componentDidMount之间渲染dom，componentDidMount执行时虚拟daom挂载完成
相当于vue中的mounted

06. 使用原生的JS事件绑定机制实现count自增

1、原生方法事件函数中this.props有问题，改成箭头函数，props是只读的不能改变
2、要能可读可写只能用this.state,将this.props赋值给this.state 如右图
3、同时将页面中的props改成state，
4、在事件中调用this.setState修改state

注意：原生事件中的this props只能读不能写基数赋值给state 别忘了调用this.setState
查看vue文档

```
constructor(props) {
  super(props)

  // 初始化组件的私有状态，保存的是组件的私有
  this.state = {
    msg: 'ok',
    count: props.initcount // 基数，把 外界传
    这样，就把 count 值改成了可读可写的 state
  }
}
```

07. 使用React提供的事件绑定机制实现count自增

Jsx语法中this指向实例，this.increment只是函数的引用，不是调用，不能改变increment的this，
Increment的this在函数执行时会默认定义为undefind，所以在定义时用箭头函数定义从而绑定this 或者用bind

提问：调用传值函数
用法总结：1jsx中 on事件 绑定事件2、class中定义函数（用箭头函数定义）

08. 实现Counter是计数器偶数更新奇数不更新的需求

1. 在 shouldComponentUpdate 中要求必须返回一个布尔值
2. 在 shouldComponentUpdate 中，如果返回的值是 false，则 不会继续执行后续的生命周期函数，而是直接退回到了 运行中 的状态，此时有序 后续的 render 函数并没有被调用，因此，页面不会被更新，但是， 组件的 state 状态，却被修改了；

总结：ture时state变了，页面也变了，false时state变了 页面未变
完成一个 偶数更新，奇数不变的案例

注意：经过打印分析，得到，此时页面上的 DOM 节点，都是旧的，应该慎重操作，因为你可能操作的是旧DOM

怎么解决呢 shouldComponentUpdate有两个参数，第一个参数为最新props，第二个参数为最新的state 代码如图：

```
▶Warning: Counter.shouldComponentUpdate(): Returned undefined instead of a boolean
value. Make sure to return true or false.
```

```
shouldComponentUpdate(nextProps, nextState) {
  // 经过打印测试发现，在 shouldComponentUpdate 中，通过 this.state.count 拿到的值，是上一次的旧数据，并不是当前最新的；
  // console.log(this.state.count + ' ---- ' + nextState.count);
  // return this.state.count % 2 === 0 ? true : false
  // return nextState.count % 2 === 0 ? true : false
  return true
}
```

Webapp第三天课程

09. 学习componentWillUpdate和ref获取元素

此时dom未渲染，谨慎操作，用ref获取dom元素 类似vue的this.$refs.xxx

疑问：render为什么不会覆盖componentWillUpdate中的dom修改呢，因为在生命周期中所有的dom操作都是异步执行，都在虚拟dom与真实dom计算渲染之后执行

分别在componwillupdate中直接使用this.refs修改dom，最终结果为render，这也解释通了，render的虚拟dom未覆盖componentWillUpdate中修改的dom

10. componentDidUpdate函数的说明

Render中直接调用this.refs.h3会报错
开始第一次执行时页面中并没有dom
解决方案：加一个判断，短路运算如图1
此时页面中的dom也是旧的

ComponentDidUpdate 页面更新完成，此时修改dom操作为最终结果。

```
g(this.refs.h3 && this.refs.h3.innerHTML);
```

1

11. 为什么在render中不能调用this.setState

测试在render中调用this.setState,瞬间到1000，栈溢出
分析如下：不要在 render 中使用 this.setState,因为 会陷入死循环

根据声明周期来理解

```
▶Uncaught Error: Maximum update depth exceeded. This can happen when a component    bundle.js:1442
repeatedly calls setState inside componentWillUpdate or componentDidUpdate. React limits the number
of nested updates to prevent infinite loops.
```

```
render() {
  return <div>
    <h1>这是父组件</h1>
    <input type="button" value="点击修改父组件的 MSG" onClick={this.changeMsg} />
    <hr />
    <Son pmsg={this.state.msg}></Son>
  </div>
}
```

对比vue的父子组件传值

1

```
class Son extends React.Component {
  constructor(props) {
    super(props)

    this.state = {}
  }

  render() {
    return <div>
      <h3>这是子组件 --- {this.props.pmsg}</h3>
    </div>
  }
}
```

2

12. 介绍componentWillReceiveProps和钩子函数的参数列表

新建一个testreciveProps组件
父组件向子组件传值，
1、当子组件第一次被渲染到页面上的时候，不会触发这个 函数；
2、只有当 父组件中，通过 某些 事件，重新修改了 传递给 子组件的 props 数据之后，才会触发 componentWillReceiveProps

注意： 在 componentWillReceiveProps 被触发的时候，如果我们使用 this.props 来获取属性值，这个属性值，不是最新的，是上一次的旧属性值
如果想要获取最新的属性值，需要通过 componentWillReceiveProps 的参数列表来获取

哪些生命周期有参数，都分别有哪些参数 如图3

```
componentWillReceiveProps(nextProps) {

  console.log(this.props.pmsg + ' ---- ' + nextProps.pmsg);
}
```

- Mounting :
  - constructor()
  - componentWillMount()
  - render()
  - componentDidMount()
- Updating :
  - componentWillReceiveProps(nextProps)
  - shouldComponentUpdate(nextProps, nextState)
  - componentWillUpdate(nextProps, nextState)
  - render()
  - componentDidUpdate(prevProps, prevState)
- Unmounting :
  - componentWillUnmount()

13. React中绑定this并传参的前两种方式

定义bindthis组件，事件绑定的函数如何传递参数，并绑定this
普通方法，this指向的是undefined，不能再函数中使用this
如何解决：1、改成箭头函数
2、绑定是调用bind，第一个参数指定this bind和call和apply的区别
Call、apply会立即调用，bind只会修改this不会修改
注：bind第一个参数修改this指向，bind第二个及之后的参数可以设置函数参数

Bind不会修改原函数，会返回一个新的函数
3、在构造函数中使用this.changeMsg = this.changeMsg.bind(this);
疑问：第三种方式怎么传递参数呢？不能写死啊

14. 绑定this并传参的第三种方式

课程中传递的参数是死的，绑定时用一个箭头函数来返回
不能写 立即调用的方法 不然的话会立即执行

15. React中把页面上的数据通过onChange同步到state中

```
    {/* 在 Vue 中，有 v-model 指令来实现双向数据绑定，但是，在 React 中， 根本没有指令的概念，因此React默认也不支持 双向数据绑定 */}
    {/* React 只支持，把数据从 state 上传输到 页面，但是，无法自动实现数据从 页面 传输到 state 中 进行保存，也就是，React 不支持数据的自动逆向传输， 只是实现了数据的单向绑定 */}
    {/* 注意：如果为 表单元素，提供了 value 属性绑定，那么，必须同时为 表单元素 绑定 readOnly，或者提供要给 onChange 事件 */}
    {/* 如果提供了readOnly，表示这个元素只读的不能被修改 */}
    {/* 如果提供了onChange 表示，这个元素的值可以被修改，但是，要自己定义修改的逻辑 */}

// 如果想让 文本框在触发 onChange 的时候，同时把文本框最新的值，保存到 state 中，那么，我们需要手动调用 this.setState

    // 获取文本框中 最新文本的3种方式：
    //  1. 使用 document.getElementById 来拿
    //  2. 使用 ref 来拿
    // console.log(this.refs.txt.value);
    //  3. 使用 事件对象的 参数 e 来拿   e.target 就表示触发 这个事件的 事件源对象，得到的是一个原生的JS DOM 对象
    // console.log(e.target.value);

单项数据流 dom改变 =》触发事件=》this.setState=>修改数据=》渲染dom
```

```
▶Warning: Failed prop type: You provided a `value` prop to a form field    bundle.js:1398
without an `onChange` handler. This will render a read-only field. If the field should
be mutable use `defaultValue`. Otherwise, set either `onChange` or `readOnly`.
    in input (created by BindThis)
    in div (created by BindThis)
    in BindThis
    in div
```

16. 评论列表案例

this.props.reload() 与this.$emit("父组件绑定的名称")
传递前者用props后者用v-on

```
{/* 发表评论的组件 */}
{/* 相对于 Vue 中，把 父组件传递给子组件的 普通属性 和 方法属性，区别对待， 普通属性用 props 接收， 方法 使用 this.$emit('方法名') */}
{/* react 中，只要是传递给 子组件的数据，不管是 普通的类型，还是方法，都可以使用 this.props 来调用 */}
<CMTBox reload={this.loadCmts}></CMTBox>
```


17. 扩展：父组件使用Context特性为子孙组件传递数据

多层组件传值比较繁琐，那怎么办呢？借用context
案例 定义三层组件

演示逐层传递
1、在父组件中定义一个方法和一个静态属性
2、子组件先校验
3、在自组件通过this.context来访问

记忆小窍门

```
//   getChildContextTypes
// 1. 在 父组件中，定义一个 function，这个function 有个固定的名称，叫做 getChildContext ，
// 内部，必须 返回一个 对象，这个对象，就是要共享给 所有子孙自建的  数据
getChildContext() {
  return {
    color: this.state.color
  }
}

// 2. 使用 属性校验，规定一下传递给子组件的 数据类型，
// 需要定义 一个 静态的（static） childContextTypes（固定名称，不要改）
static childContextTypes = {
  color: ReactTypes.string // 规定了 传递给子组件的 数据类型
}
```

```
// 3. 上来之后，先来个属性校验，去校验一下父组件传递过来的 参数类型
static contextTypes = {
  color: ReactTypes.string
  // 这里，如果子组件，想要使用 父组件通过 context 共享的数据，
  // 那么在使用之前，一定要先 做一下数据类型校验
}

render() {
  return <div>
    <h5 style={{ color: this.context.color }}>这是 孙子组件  ---  {this.context.color} </h5>

  </div>
}
```