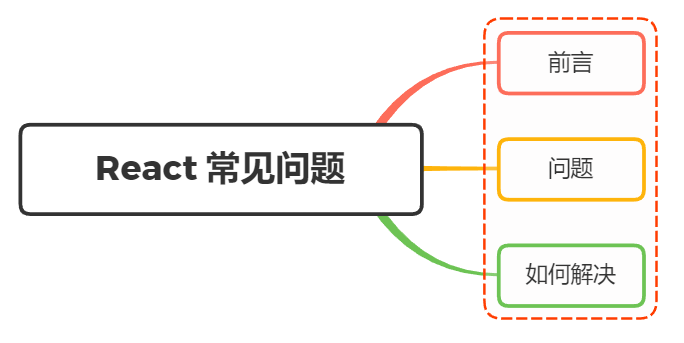
# 面试官：说说你在使用React 过程中遇到的常见问题？如何解决?



## 一、前言

在使用react开发项目过程中，每个人或多或少都会遇到一些"奇怪"的问题，本质上都是我们对其理解的不够透彻

react 系列，33个工作日，33次凌晨还在亮起的台灯，到今天就圆满画上句号了，比心

在系列中我们列出了很多比较经典的考题，工作中遇到的问题也往往就藏中其中，只是以不同的表现形式存在罢了

今天的题解不算题解，准确来说是对整个系列的一次贯穿，总结

目录:

* react 有什么特性
* 生命周期有哪些不同阶段？每个阶段对应的方法是？
* state 和 props有什么区别？
* super()和super(props)有什么区别？
* setState执行机制？
* React的事件机制？
* 事件绑定的方式有哪些？
* 构建组件的方式有哪些？区别？
* 组件之间如何通信？
* key有什么作用？
* refs 的理解？应用场景？
* Hooks的理解？解决了什么问题？
* 如何引入css？
* redux工作原理？
* redux中间件有哪些？
* react-router组件有哪些？
* render触发时机？
* 如何减少render？
* JSX转化DOM过程？
* 性能优化手段有哪些
* 如何做服务端渲染？

### react 有什么特性

主要的特性分为：

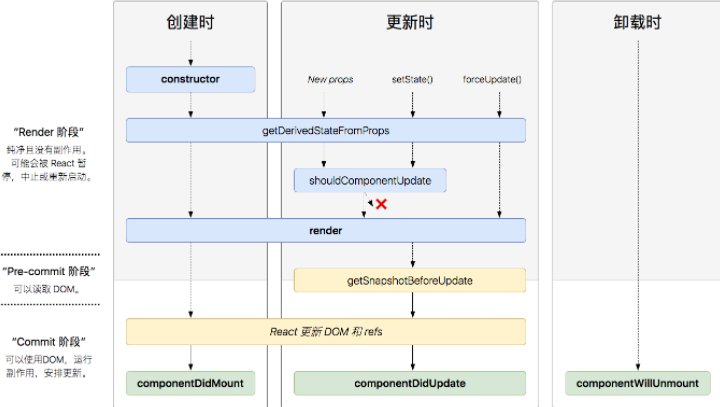
* JSX语法
* 单向数据绑定
* 虚拟DOM
* 声明式编程
* Component

借助这些特性，react整体使用起来更加简单高效，组件式开发提高了代码的复用率

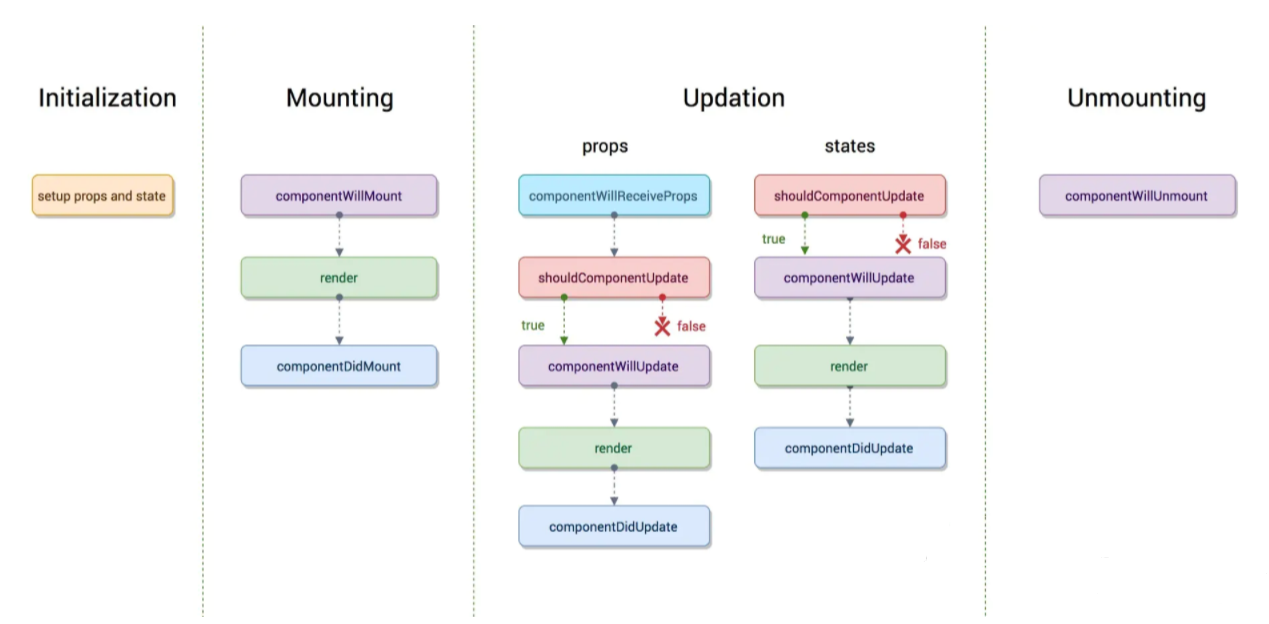
### 生命周期有哪些不同阶段？每个阶段对应的方法是？

主要分成了新的生命周期和旧的生命周期：

* 新版生命周期整体流程如下图所示：



旧的生命周期流程图如下：



### state 和 props有什么区别？

两者相同点：

* 两者都是 JavaScript 对象
* 两者都是用于保存信息
* props 和 state 都能触发渲染更新

区别：

* props 是外部传递给组件的，而 state 是在组件内被组件自己管理的，一般在 constructor 中初始化
* props 在组件内部是不可修改的，但 state 在组件内部可以进行修改
* state 是多变的、可以修改

### super()和super(props)有什么区别？

在React中，类组件基于ES6，所以在constructor中必须使用super

在调用super过程，无论是否传入props，React内部都会将porps赋值给组件实例porps属性中

如果只调用了super()，那么this.props在super()和构造函数结束之间仍是undefined

### setState执行机制？

在react类组件的状态需要通过setState进行更改，在不同场景下对应不同的执行顺序：

* 在组件生命周期或React合成事件中，setState是异步
* 在setTimeout或者原生dom事件中，setState是同步

当我们批量更改state的值的时候，react内部会将其进行覆盖，只取最后一次的执行结果

当需要下一个state依赖当前state的时候，则可以在setState中传递一个回调函数进行下次更新

### React的事件机制？

React基于浏览器的事件机制自身实现了一套事件机制，包括事件注册、事件的合成、事件冒泡、事件派发等

组件注册的事件最终会绑定在document这个 DOM上，而不是 React组件对应的 DOM，从而节省内存开销

自身实现了一套事件冒泡机制，阻止不同时间段的冒泡行为，需要对应使用不同的方法

### 事件绑定的方式有哪些？

react常见的绑定方式有如下：

* render方法中使用bind
* render方法中使用箭头函数
* constructor中bind
* 定义阶段使用箭头函数绑定

前两种方式在每次组件render的时候都会生成新的方法实例，性能问题欠缺

### 构建组件的方式有哪些？区别？

组件的创建主要分成了三种方式：

* 函数式创建
* 继承 React.Component 创建
* 通过 React.createClass 方法创建

如今一般都是前两种方式，对于一些无状态的组件创建，建议使用函数式创建的方式，再比如hooks的机制下，函数式组件能做类组件对应的事情，所以建议都使用函数式的方式来创建组件

### 组件之间如何通信？

组件间通信可以通过props、传递回调函数、context、redux等形式进行组件之间通讯

### key有什么作用？

使用key是react性能优化的手段，在一系列数据最前面插入元素，如果没有key的值，则所有的元素都需要进行更换，而有key的情况只需要将最新元素插入到前面，不涉及删除操作

在使用key的时候应保证：

* key 应该是唯一的
* key不要使用随机值（随机数在下一次 render 时，会重新生成一个数字）
* 避免使用 index 作为 key

### refs 的理解？应用场景？

Refs允许我们访问 DOM节点或在 render方法中创建的 React元素

下面的场景使用refs非常有用：

* 对Dom元素的焦点控制、内容选择、控制
* 对Dom元素的内容设置及媒体播放
* 对Dom元素的操作和对组件实例的操作
* 集成第三方 DOM 库

### Hooks的理解？解决了什么问题？

Hook 是 React 16.8 的新增特性。它可以让你在不编写 class 的情况下使用 state 以及其他的 React 特性

解决问题如下：

* 难以重用和共享组件中的与状态相关的逻辑
* 逻辑复杂的组件难以开发与维护，当我们的组件需要处理多个互不相关的 local state 时，每个生命周期函数中可能会包含着各种互不相关的逻辑在里面
* 类组件中的this增加学习成本，类组件在基于现有工具的优化上存在些许问题
* 由于业务变动，函数组件不得不改为类组件等等

### 如何引入css？

常见的CSS引入方式有以下：

* 在组件内直接使用
* 组件中引入 .css 文件
* 组件中引入 .module.css 文件
* CSS in JS

组件内直接使用css会导致大量的代码，而文件中直接引入css文件是全局作用域，发生层叠

引入.module.css文件能够解决局部作用域问题，但是不方便动态修改样式，需要使用内联的方式进行样式的编写

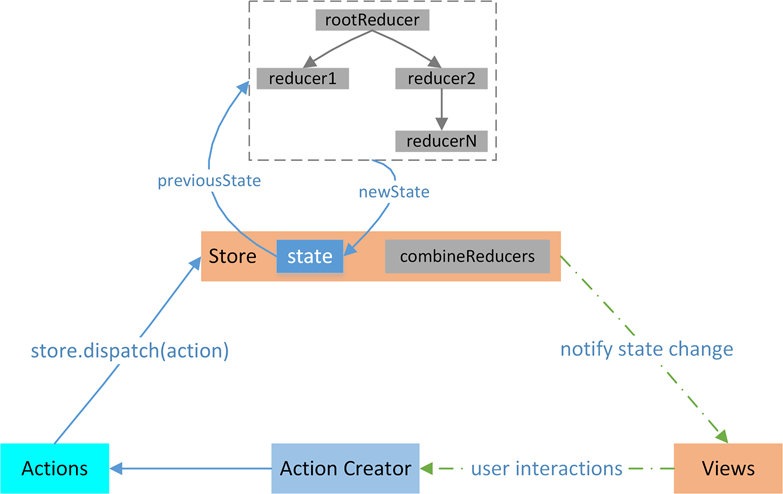
css in js这种方法，可以满足大部分场景的应用，可以类似于预处理器一样样式嵌套、定义、修改状态等

### redux工作原理？

redux要求我们把数据都放在 store公共存储空间

一个组件改变了 store 里的数据内容，其他组件就能感知到 store的变化，再来取数据，从而间接的实现了这些数据传递的功能

工作流程图如下所示：



### redux中间件有哪些？

市面上有很多优秀的redux中间件，如：

* redux-thunk：用于异步操作
* redux-logger：用于日志记录

### react-router组件有哪些？

常见的组件有：

* BrowserRouter、HashRouter
* Route
* Link、NavLink
* switch
* redirect

### render触发时机？

在React 中，类组件只要执行了 setState 方法，就一定会触发 render 函数执行

函数组件useState 会判断当前值有无发生改变确定是否执行render方法，一旦父组件发生渲染，子组件也会渲染

### 如何减少render？

父组件渲染导致子组件渲染，子组件并没有发生任何改变，这时候就可以从避免无谓的渲染，具体实现的方式有如下：

* shouldComponentUpdate
* PureComponent
* React.memo

### JSX转化DOM过程？

jsx首先会转化成React.createElement这种形式，React.createElement作用是生成一个虚拟Dom对象，然后会通过ReactDOM.render进行渲染成真实DOM

### 性能优化手段有哪些

除了减少render的渲染之外，还可以通过以下手段进行优化：

除此之外， 常见性能优化常见的手段有如下：

* 避免使用内联函数
* 使用 React Fragments 避免额外标记
* 使用 Immutable
* 懒加载组件
* 事件绑定方式
* 服务端渲染

### 如何做服务端渲染？

node server 接收客户端请求，得到当前的请求url 路径，然后在已有的路由表内查找到对应的组件，拿到需要请求的数据，将数据作为 props、context或者store 形式传入组件

然后基于 react 内置的服务端渲染方法 renderToString()把组件渲染为 html字符串在把最终的 html进行输出前需要将数据注入到浏览器端

浏览器开始进行渲染和节点对比，然后执行完成组件内事件绑定和一些交互，浏览器重用了服务端输出的 html 节点，整个流程结束

