社招前端二面react面试题集锦

在哪个生命周期中你会发出Ajax请求？为什么？

Ajax请求应该写在组件创建期的第五个阶段，即 componentDidMount生命周期方法中。原因如下。 在创建期的其他阶段，组件尚未渲染完成。而在存在期的5个阶段，又不能确保生命周期方法一定会执行（如通过 shouldComponentUpdate方法优化更新等）。在销毁期，组件即将被销毁，请求数据变得无意义。因此在这些阶段发出Ajax请求显然不是最好的选择。 在组件尚未挂载之前，Ajax请求将无法执行完毕，如果此时发出请求，将意味着在组件挂载之前更新状态（如执行 setState），这通常是不起作用的。 在 componentDidMount方法中，执行Ajax即可保证组件已经挂载，并且能够正常更新组件。

React- Router有几种形式？

有以下几种形式。 HashRouter，通过散列实现，路由要带#。 BrowerRouter，利用HTML5中history API实现，需要服务器端支持，兼容性不是很好。

redux有什么缺点

- 一个组件所需要的数据，必须由父组件传过来，而不能像 `flux` 中直接从 `store` 取。
- 当一个组件相关数据更新时，即使父组件不需要用到这个组件，父组件还是会重新 `render` ，可能会有效率影响，或者需要写复杂的 `shouldComponentUpdate` 进行判断。

在 Reducer文件里，对于返回的结果，要注意哪些问题？

在 Reducer文件里，对于返回的结果，必须要使用 Object.assign ( )来复制一份新的 state，否则页面不会跟着数据刷新。

javascript 复制代码

```
return Object.assign({}, state, {
  type: action.type,
  shouldNotPaint: true,
});
```

在 ReactNative中，如何解决 adb devices找不到连接设备的问题?

在使用 Genymotion时，首先需要在SDK的 platform-tools中加入环境变量，然后在Genymotion中单击 Setting，选择ADB选项卡，单击 Use custom Android SDK tools，浏览本地SDK的位置，单击OK按钮就可以了。启动虚拟机后，在cmd中输入 adb devices可以查看设备。

redux中间件

> 中间件提供第三方插件的模式，自定义拦截 `action -> reducer` 的过程。变为 `action -> middlewares -> reducer` 。这种机制可以让我们改变数据流，实现如异步 `action` ， `action` 过滤，日志输出，异常报告等功能

- `redux-logger` ： 提供日志输出
- `redux-thunk` ： 处理异步操作
- `redux-promise` ： 处理异步操作， `actionCreator` 的返回值是 `promise`

参考 前端进阶面试题详细解答

React怎么做数据的检查和变化

> `Model` 改变之后（可能是调用了 `setState` ），触发了 `virtual dom` 的更新，再用 `diff` 算法来 把`virtual DOM` 比较 `real DOM` ，看看是哪个 `dom` 节点更新了，再渲染 `real dom`

简述flux 思想

> `Flux` 的最大特点，就是数据的"单向流动"。

- 用户访问 `View`
- `View` 发出用户的 `Action`
- `Dispatcher` 收到 `Action` ，要求 `Store` 进行相应的更新
- `Store` 更新后，发出一个 `"change"` 事件
- `View` 收到 `"change"` 事件后，更新页面

React 中 refs 的作用是什么

- `Refs` 是 `React` 提供给我们的安全访问 `DOM` 元素或者某个组件实例的句柄
- 可以为元素添加 `ref` 属性然后在回调函数中接受该元素在 `DOM` 树中的句柄，该值会作为回调函数的第一个参数返回

shouldComponentUpdate有什么用？为什么它很重要?

组件状态数据或者属性数据发生更新的时候，组件会进入存在期，视图会渲染更新。在生命周期方法 should ComponentUpdate中，允许选择退出某些组件（和它们的子组件）的和解过程。 和解的最终目标是根据新的状态，以最有效的方式更新用户界面。如果我们知道用户界面的某一部分不会改变，那么没有理由让 React弄清楚它是否应该更新渲染。通过在shouldComponentUpdate方法中返回 false, React将让当前组件及其所有子组件保持与当前组件状态相同。

受控组件、非受控组件

- 受控组件就是改变受控于数据的变化，数据变了页面也变了。受控组件更合适，数据驱动是react核心
- 非受控组件不是通过数据控制页面内容

说说 React组件开发中关于作用域的常见问题。

在 EMAScript5语法规范中，关于作用域的常见问题如下。 （1）在map等方法的回调函数中，要绑定作用域this（通过bind方法）。 （2）父组件传递给子组件方法的作用域是父组件实例化对象，无法改变。 （3）组件事件回调函数方法的作用域是组件实例化对象（绑定父组件提供的方法就是父组件实例化对象），无法改变。 在 EMAScript6语法规范中，关于作用域的常见问题如下。 （1）当使用箭头函数作为map等方法的回调函数时，箭头函数的作用域是当前组件的实例化对象（即箭头函数的作用域是定义时的作用域），无须绑定作用域。 （2）事件回调函数要绑定组件作用域。 （3）父组件传递方法要绑定父组件作用域。 总之，在EMAScript6语法规范中，组件方法的作用域是可以改变的。

这段代码有什么问题?

javascript 复制代码

```
class App extends Component {
  constructor(props) {
```

```
    super(props);
    this.state = {
      username: "有课前端网",
      msg: " ",
    };
  }
  render() {
    return <div> {this.state.msg}</div>;
  }
  componentDidMount() {
    this.setState((oldState, props) => {
      return {
        msg: oldState.username + " - " + props.intro,
      };
    });
  }
}
```

render (< App intro=" 前端技术专业学习平台"> , ickt) 在页面中正常输出“有课前端网-前端技术专业学习平台”。但是这种写法很少使用，并不是常用的写法。React允许对 setState方法传递一个函数，它接收到先前的状态和属性数据并返回一个需要修改的状态对象，正如我们在上面所做的那样。它不但没有问题，而且如果根据以前的状态（ state）以及属性来修改当前状态，推荐使用这种写法。

在 React中元素（ element）和组件（ component）有什么区别?

简单地说，在 React中元素（虚拟DOM）描述了你在屏幕上看到的DOM元素。 换个说法就是，在 React中元素是页面中DOM元素的对象表示方式。在 React中组件是一个函数或一个类，它可以接受输入并返回一个元素。 注意：工作中，为了提高开发效率，通常使用JSX语法表示 React元素（虚拟DOM）。在编译的时候，把它转化成一个 React. createElement调用方法。

说说你用react有什么坑点?

1. JSX做表达式判断时候，需要强转为boolean类型

> 如果不使用 `!!b` 进行强转数据类型，会在页面里面输出 `0` 。

```javascriptx
render() {
  const b = 0;
  return <div>
    {
      !!b && <div>这是一段文本</div>
    }
  </div>
}
```

2. 尽量不要在 `componentWillReviceProps` 里使用 setState，如果一定要使用，那么需要判断结束条件，不然会出现无限重渲染，导致页面崩溃

3. 给组件添加ref时候，尽量不要使用匿名函数，因为当组件更新的时候，匿名函数会被当做新的prop处理，让ref属性接受到新函数的时候，react内部会先清空ref，也就是会以null为回调参数先执行一次ref这个props，然后在以该组件的实例执行一次ref，所以用匿名函数做ref的时候，有的时候去ref赋值后的属性会取到null

4. 遍历子节点的时候，不要用 index 作为组件的 key 进行传入

shouldComponentUpdate 的作用

> `shouldComponentUpdate` 允许我们手动地判断是否要进行组件更新，根据组件的应用场景设置函数的合理返回值能够帮我们避免不必要的更新

key的作用

> 是给每一个 vnode 的唯一 id,可以依靠 key,更准确,更快的拿到 oldVnode 中对应的 vnode 节点

```html
<!-- 更新前 -->
<div>
  <p key="ka">ka</p>
  <h3 key="song">song</he>
</div>

<!-- 更新后 -->
<div>
```

```
  <h3 key="song">song</h3>
  <p key="ka">ka</p>
</div>
```

如果没有 key，React 会认为 div 的第一个子节点由 p 变成 h3，第二个子节点由 h3 变成 p，则会销毁这两个节点并重新构造。

但是当我们用 key 指明了节点前后对应关系后，React 知道 `key === "ka"` 的 p 更新后还在，所以可以复用该节点，只需要交换顺序。

key 是 React 用来追踪哪些列表元素被修改、被添加或者被移除的辅助标志。

在开发过程中，我们需要保证某个元素的 key 在其同级元素中具有唯一性。在 React diff 算法中，React 会借助元素的 Key 值来判断该元素是新近创建的还是被移动而来的元素，从而减少不必要的元素重新渲染。同时，React 还需要借助 key 来判断元素与本地状态的关联关系。

我现在有一个button，要用react在上面绑定点击事件，要怎么做?

javascriptx 复制代码

```
class Demo {
  render() {
    return <button onClick={(e) => {
      alert('我点击了按钮')
    }}>
      按钮
    </button>
  }
}
```

你觉得你这样设置点击事件会有什么问题吗?

> 由于 `onClick` 使用的是匿名函数，所有每次重渲染的时候，会把该 `onClick` 当做一个新的 `prop` 来处理，会将内部缓存的 `onClick` 事件进行重新赋值，所以相对直接使用函数来说，可能有一点的性能下降

修改

javascript 复制代码

```
class Demo {

  onClick = (e) => {
```

```
    alert('我点击了按钮')
  }

  render() {
    return <button onClick={this.onClick}>
      按钮
    </button>
  }
```

state 和 props 区别是啥?

- state 是组件自己管理数据，控制自己的状态，可变；
- props 是外部传入的数据参数，不可变；
- 没有state的叫做无状态组件，有state的叫做有状态组件；
- 多用 props，少用 state，也就是多写无状态组件。

React.createClass和extends Component的区别有哪些?

React.createClass和extends Component的bai区别主要在于：

（1）语法区别

- createClass本质上是一个工厂函数，extends的方式更加接近最新的ES6规范的class写法。两种方式在语法上的差别主要体现在方法的定义和静态属性的声明上。
- createClass方式的方法定义使用逗号，隔开，因为creatClass本质上是一个函数，传递给它的是一个Object；而class的方式定义方法时务必谨记不要使用逗号隔开，这是ES6 class的语法规范。

（2）propType 和 getDefaultProps

- React.createClass：通过proTypes对象和getDefaultProps()方法来设置和获取props.
- React.Component：通过设置两个属性propTypes和defaultProps

（3）状态的区别

- React.createClass：通过getInitialState()方法返回一个包含初始值的对象
- React.Component：通过constructor设置初始状态

（4）this区别

- React.createClass：会正确绑定this
- React.Component：由于使用了 ES6，这里会有些微不同，属性并不会自动绑定到 React 类的实例上。

(5) Mixins

- React.createClass：使用 React.createClass 的话，可以在创建组件时添加一个叫做 mixins 的属性，并将可供混合的类的集合以数组的形式赋给 mixins。
- 如果使用 ES6 的方式来创建组件，那么 `React mixins` 的特性将不能被使用了。