**#### 一、调用setState的时候，发生了什么？**

当调用 setState 时，React会做的第一件事情是将传递给 setState 的对象合并到组件的当前状态。这将启动一个称为和解（reconciliation）的过程。和解（reconciliation）的最终目标是以最有效的方式，根据这个新的状态来更新UI。 为此，React将构建一个新的 React 元素树（您可以将其视为 UI 的对象表示）。

一旦有了这个树，为了弄清 UI 如何响应新的状态而改变，React 会将这个新树与上一个元素树相比较（ diff ）。

通过这样做， React 将会知道发生的确切变化，并且通过了解发生什么变化，只需在绝对必要的情况下进行更新即可最小化 UI 的占用空间。

**#### 二、在 React 当中 Element 和 Component 有何区别？**

简单地说，一个 React element 描述了你想在屏幕上看到什么。换个说法就是，一个 React element 是一些 UI 的对象表示。

一个 React Component 是一个函数或一个类，它可以接受输入并返回一个 React element t（通常是通过 JSX ，它被转化成一个 createElement 调用）。

**#### 三、什么时候在功能组件( Class Component )上使用类组件( Functional Component )？**

如果您的组件具有状态( state )或生命周期方法，请使用 Class 组件。否则，使用功能组件

**#### 四、什么是 React 的 refs ，为什么它们很重要？**

refs就像是一个逃生舱口，允许您直接访问DOM元素或组件实例。为了使用它们，您可以向组件添加一个 ref 属性，该属性的值是一个回调函数，它将接收底层的 DOM 元素或组件的已挂接实例，作为其第一个参数。

class UnControlledForm extends Component {

handleSubmit = () => {

console.log("Input Value: ", this.input.value)

}

render () {return (<form onSubmit={this.handleSubmit}>

<input

type='text'

ref={(input) => this.input = input} />

<button type='submit'>Submit</button>

</form>)

}

}

以上注意到我们的输入字段有一个 ref 属性，其值是一个函数。该函数接收我们然后放在实例上的实际的 DOM 元素，以便在 handleSubmit 函数内部访问它。经常误解的是，您需要使用类组件才能使用 ref ，但 ref 也可以通过利用 JavaScript 中的 闭包 与 功能组件( functional components )一起使用。

function CustomForm ({handleSubmit}) {

let inputElementreturn (<form onSubmit={() => handleSubmit(inputElement.value)}>

<input

type='text'

ref={(input) => inputElement = input} />

<button type='submit'>Submit</button>

</form>)

}

**#### 五、React 中的keys是什么，为什么它们很重要？**

keys是什么帮助 React 跟踪哪些项目已更改、添加或从列表中删除。

return (<ul>

{this.state.todoItems.map(({task, uid}) => {return <li key={uid}>{task}</li>})}</ul>)

}

每个 keys 在兄弟元素之间是独一无二的。我们已经谈过几次关于和解（reconciliation）的过程，而且这个和解过程（reconciliation）中的一部分正在执行一个新的元素树与最前一个的差异。keys 使处理列表时更加高效，因为 React 可以使用子元素上的 keys 快速知道元素是新的还是在比较树时才被移动。

而且 keys 不仅使这个过程更有效率，而且没有 keys ，React 不知道哪个本地状态对应于移动中的哪个项目。所以当你 map 的时候，不要忽略了 keys 。

**#### 六、受控组件( controlled component )与不受控制的组件( uncontrolled component )有什么区别？**

React 的很大一部分是这样的想法，即组件负责控制和管理自己的状态。

当我们将 native HTML 表单元素（ input, select, textarea 等）投入到组合中时会发生什么？我们是否应该使用 React 作为“单一的真理来源”，就像我们习惯使用React一样？ 或者我们是否允许表单数据存在 DOM 中，就像我们习惯使用HTML表单元素一样？ 这两个问题是受控（controlled） VS 不受控制（uncontrolled）组件的核心。

受控组件是React控制的组件，也是表单数据的唯一真理来源。

如下所示， username 不存在于 DOM 中，而是以我们的组件状态存在。每当我们想要更新 username 时，我们就像以前一样调用setState。

class ControlledForm extends Component {

state = {

username: ''

}

updateUsername = (e) => {this.setState({

username: e.target.value,

})

}

handleSubmit = () => {}

render () {return (<form onSubmit={this.handleSubmit}>

<input

type='text'

value={this.state.username}

onChange={this.updateUsername} />

<button type='submit'>Submit</button>

</form>)

}

}

不受控制( uncontrolled component )的组件是您的表单数据由 DOM 处理，而不是您的 React 组件。

我们使用 refs 来完成这个。

class UnControlledForm extends Component {

handleSubmit = () => {

console.log("Input Value: ", this.input.value)

}

render () {return (<form onSubmit={this.handleSubmit}>

<input

type='text'

ref={(input) => this.input = input} />

<button type='submit'>Submit</button>

</form>)

}

}

虽然不受控制的组件通常更容易实现，因为您只需使用引用从DOM获取值，但是通常建议您通过不受控制的组件来支持受控组件。

主要原因是受控组件 支持即时字段验证 ，允许您有条件地禁用/启用按钮，强制输入格式

**#### 七、在哪个生命周期事件中你会发出 AJAX 请求，为什么？**

AJAX 请求应该在 componentDidMount 生命周期事件中。 有几个原因:

Fiber，是下一次实施React的和解算法，将有能力根据需要启动和停止渲染，以获得性能优势。其中一个取舍之一是 componentWillMount ，而在其他的生命周期事件中出发 AJAX 请求，将是具有 “非确定性的”。 这意味着 React 可以在需要时感觉到不同的时间开始调用 componentWillMount。这显然是AJAX请求的不好的方式。

-您不能保证在组件挂载之前，AJAX请求将无法 resolve。如果这样做，那意味着你会尝试在一个未挂载的组件上设置 StState，这不仅不会起作用，反而会对你大喊大叫。 在 componentDidMount 中执行 AJAX 将保证至少有一个要更新的组件。

**#### 八、shouldComponentUpdate 应该做什么，为什么它很重要？**

在生命周期方法 shouldComponentUpdate 中，允许我们选择退出某些组件（和他们的子组件）的 reconciliation 过程。

我们为什么要这样做？

如上所述，“和解（ reconciliation ）的最终目标是以最有效的方式，根据新的状态更新用户界面”。如果我们知道我们的用户界面（UI）的某一部分不会改变，那么没有理由让 React 很麻烦地试图去弄清楚它是否应该渲染。通过从 shouldComponentUpdate 返回 false，React 将假定当前组件及其所有子组件将保持与当前组件相同。

**#### 九、您如何告诉React 构建（build）生产模式，该做什么？**

通常，您将使用Webpack的 DefinePlugin 方法将 NODE\_ENV 设置为 production。这将剥离像 propType 验证和额外的警告。除此之外，还有一个好主意，可以减少你的代码，因为React使用 Uglify 的 dead-code 来消除开发代码和注释，这将大大减少你的包的大小。

为什么要使用 React.Children.map（props.children，（）=>） 而不是 props.children.map（（）=>）

因为不能保证props.children将是一个数组。

以此代码为例，

<Parent>

<h1>Welcome.</h1>

</Parent>

在父组件内部，如果我们尝试使用 props.children.map 映射孩子，则会抛出错误，因为 props.children 是一个对象，而不是一个数组。

如果有多个子元素，React 只会使props.children成为一个数组。就像下面这样：

<Parent>

<h1>Welcome.</h1>

<h2>props.children will now be an array</h2>

</Parent>

这就是为什么你喜欢 React.Children.map ，因为它的实现考虑到 props.children 可能是一个数组或一个对象。

**#### 十、概述下 React 中的事件处理逻辑**

为了解决跨浏览器兼容性问题，React 会将浏览器原生事件（Browser Native Event）封装为合成事件（SyntheticEvent）传入设置的事件处理器中。这里的合成事件提供了与原生事件相同的接口，不过它们屏蔽了底层浏览器的细节差异，保证了行为的一致性。另外有意思的是，React 并没有直接将事件附着到子元素上，而是以单一事件监听器的方式将所有的事件发送到顶层进行处理。这样 React 在更新 DOM 的时候就不需要考虑如何去处理附着在 DOM 上的事件监听器，最终达到优化性能的目的。

**#### 十一、createElement 与 cloneElement 的区别是什么？**

createElement 函数是 JSX 编译之后使用的创建 React Element 的函数，而 cloneElement 则是用于复制某个元素并传入新的 Props。

**#### 十二、传入 setState 函数的第二个参数的作用是什么？**

该函数会在setState函数调用完成并且组件开始重渲染的时候被调用，我们可以用该函数来监听渲染是否完成：

this.setState(

{ username: 'tylermcginnis33' },

() => console.log('setState has finished and the component has re-rendered.')

)

**#### 十三、react的优缺点**

优点：

可以通过函数式方法描述视图组件（好处：相同的输入会得到同样的渲染结果，不会有副作用；组件不会被实例化，整体渲染性能得到提升）集成虚拟DOM（性能好）

单向数据流（好处是更容易追踪数据变化排查问题

一切都是component：代码更加模块化，重用代码更容易，可维护性高

大量拥抱 es6 新特性

jsx

缺点：

jsx的一个问题是，渲染函数常常包含大量逻辑，最终看着更像是程序片段，而不是视觉呈现。后期如果发生需求更改，维护起来工作量将是巨大的

大而全，上手有难度

**#### 十四、React：组件的生命周期**

实例化

首次实例化

getDefaultProps

getInitialState

componentWillMount

render

componentDidMount

实例化完成后的更新

getInitialState

componentWillMount

render

componentDidMount

存在期

组件已存在时的状态改变

componentWillReceiveProps

shouldComponentUpdate

componentWillUpdate

render

componentDidUpdate

销毁&清理期

componentWillUnmount

说明

生命周期共提供了10个不同的API。

1.getDefaultProps

作用于组件类，只调用一次，返回对象用于设置默认的props，对于引用值，会在实例中共享。

2.getInitialState

作用于组件的实例，在实例创建时调用一次，用于初始化每个实例的state，此时可以访问this.props。

3.componentWillMount

在完成首次渲染之前调用，此时仍可以修改组件的state。

4.render

必选的方法，创建虚拟DOM，该方法具有特殊的规则：

只能通过this.props和this.state访问数据

可以返回null、false或任何React组件

只能出现一个顶级组件（不能返回数组）

不能改变组件的状态

不能修改DOM的输出

5.componentDidMount

真实的DOM被渲染出来后调用，在该方法中可通过this.getDOMNode()访问到真实的DOM元素。此时已可以使用其他类库来操作这个DOM。

在服务端中，该方法不会被调用。

6.componentWillReceiveProps

组件接收到新的props时调用，并将其作为参数nextProps使用，此时可以更改组件props及state。

componentWillReceiveProps: function(nextProps) {

if (nextProps.bool) {

this.setState({

bool: true

});

}

}

7.shouldComponentUpdate

组件是否应当渲染新的props或state，返回false表示跳过后续的生命周期方法，通常不需要使用以避免出现bug。在出现应用的瓶颈时，可通过该方法进行适当的优化。

在首次渲染期间或者调用了forceUpdate方法后，该方法不会被调用

8.componentWillUpdate

接收到新的props或者state后，进行渲染之前调用，此时不允许更新props或state。

9.componentDidUpdate

完成渲染新的props或者state后调用，此时可以访问到新的DOM元素。

10.componentWillUnmount

组件被移除之前被调用，可以用于做一些清理工作，在componentDidMount方法中添加的所有任务都需要在该方法中撤销，比如创建的定时器或添加的事件监听器。

**#### 十五、React组件通信**

1、父组件向子组件通信：使用 props

这是最简单也是最常用的一种通信方式：父组件通过向子组件传递 props，子组件得到 props 后进行相应的处理。

2、子组件向父组件通信：使用 props 回调

利用回调函数，可以实现子组件向父组件通信：父组件将一个函数作为 props 传递给子组件，子组件调用该回调函数，便可以向父组件通信。

3、跨级组件间通信：使用 context 对象

所谓跨级组件通信，就是父组件向子组件的子组件通信，向更深层的子组件通信。跨级组件通信可以采用下面两种方式：

中间组件层层传递 props

使用 context 对象

对于第一种方式，如果父组件结构较深，那么中间的每一层组件都要去传递 props，增加了复杂度，并且这些 props 并不是这些中间组件自己所需要的。不过这种方式也是可行的，当组件层次在三层以内可以采用这种方式，当组件嵌套过深时，采用这种方式就需要斟酌了。

使用 context 是另一种可行的方式，context 相当于一个全局变量，是一个大容器，我们可以把要通信的内容放在这个容器中，这样一来，不管嵌套有多深，都可以随意取用。

使用 context 也很简单，需要满足两个条件：

上级组件要声明自己支持 context，并提供一个函数来返回相应的 context 对象

子组件要声明自己需要使用 context

如果是父组件向子组件单向通信，可以使用变量，如果子组件想向父组件通信，同样可以由父组件提供一个回调函数，供子组件调用，回传参数。

在使用 context 时，有两点需要注意：

父组件需要声明自己支持 context，并提供 context 中属性的 PropTypes

子组件需要声明自己需要使用 context，并提供其需要使用的 context 属性的 PropTypes

父组件需提供一个 getChildContext 函数，以返回一个初始的 context 对象

4、非嵌套组件间通信：使用事件订阅

非嵌套组件，就是没有任何包含关系的组件，包括兄弟组件以及不在同一个父级中的非兄弟组件。对于非嵌套组件，可以采用下面两种方式：

利用二者共同父组件的 context 对象进行通信

使用自定义事件的方式

如果采用组件间共同的父级来进行中转，会增加子组件和父组件之间的耦合度，如果组件层次较深的话，找到二者公共的父组件不是一件容易的事

自定义事件是典型的发布/订阅模式，通过向事件对象上添加监听器和触发事件来实现组件间通信。

**#### 十六、react中prop和state的区别？**

需要理解的是，props是一个父组件传递给子组件的数据流，这个数据流可以一直传递到子孙组件。而state代表的是一个组件内部自身的状态（可以是父组件、子孙组件）。

**#### 十七、redux的原理？**

Redux 把一个应用程序中，所有应用模块之间需要共享访问的数据，都应该放在 State 对象中。这个应用模块可能是指 React Components，也可能是你自己访问 AJAX API 的代理模块，具体是什么并没有一定的限制。State 以 “树形” 的方式保存应用程序的不同部分的数据。这些数据可能来自于网络调用、本地数据库查询、甚至包括当前某个 UI 组件的临时执行状态（只要是需要被不同模块访问）