### 介绍一下react

React 是一个用于构建用户界面的 JAVASCRIPT 库。React主要用于构建UI，很多人认为 React 是 MVC 中的 V（视图）

React特点有：

1.声明式设计 −React采用声明范式，可以轻松描述应用。

2.高效 −React通过对DOM的模拟，最大限度地减少与DOM的交互。

3.灵活 −React可以与已知的库或框架很好地配合。

4.JSX − JSX 是 JavaScript 语法的扩展。React 开发不一定使用 JSX ，但我们建议使用它。

5.组件 − 通过 React 构建组件，使得代码更加容易得到复用，能够很好的应用在大项目的开发中。

6.单向响应的数据流 − React 实现了单向响应的数据流，从而减少了重复代码，这也是它为什么比传统数据绑定更简单。

### React单项数据流

在React中，数据是单向流动的，是从上向下的方向，即从父组件到子组件的方向。

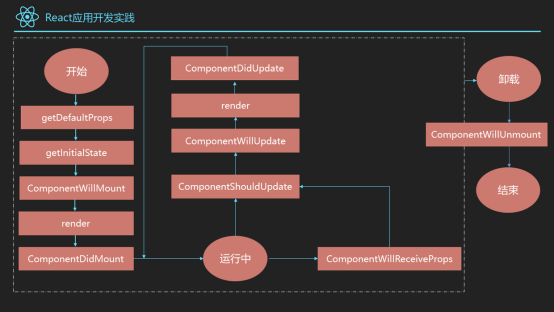
state和props是其中重要的概念，如果顶层组件初始化props，那么React会向下遍历整个组件树，重新渲染相关的子组件。其中state表示的是每个组件中内部的的状态，这些状态只在组件内部改变。  
把组件看成是一个函数，那么他接受props作为参数，内部由state作为函数的内部参数，返回一个虚拟dom的实现。

### react生命周期函数和react组件的生命周期

React的组件在第一次挂载的时候。首先获取父组件传递的props，接着获取初始的state值，接着经历挂载阶段的三个生命周期函数，也就是ComponentWillMount，render，ComponentDidMount，这三个函数分别代表**组件即将挂载，组件渲染，组件挂载**完毕三个阶段，在组件挂载完成后，组件的props和state的任意改变都会导致组件进入更新状态，在组件更新阶段。

1. 如果是props改变，则进入ComponentWillReceiveProps函数（组件将收到props），接着进入ComponentShouldUpdate（组件应该更新）进行判断是否需要更新。

2、如果是state改变则直接进入ComponentShouldUpdate判定，这个默认是true，当判定不需要更新的话，组件继续运行，需要更新的话则依次进入ComponentWillMount，render，ComponentDidMount三个函数，当组件卸载时，会首先进入生命周期函数ComponentWillUnmount,之后才进行卸载，如图



React的生命周期函数：

初始化阶段：getDefaultProps**获取实例的默认属性**，getInitialState**获取每个实例的初始化状态**，ComponentWillMount：**组件将被装载**，渲染到页面上，render：组件在这里**生成虚拟的DOM节点**，ComponentDidMount:组件真正**被装载之后**

运行中状态： componentWillReceiveProps:组件将要接收到属性的时候调用 ComponentshouldUpdate:组件接受到新属性或者新状态的时候（可以返回 false，接收数据后不更新，阻止 render 调用，后面的函数不会被继续执行了）。shouldComponentUpdate 这个方法用来判断是否需要调用 render 方法重新描绘 dom。因为 dom 的描绘非常消耗性能，如果我们能在 shouldComponentUpdate 方法中能够写出更优化的 dom diff 算法，可以极大的提高性能。 componentWillUpdate:组件即将更新不能修改属性和状态 render:组件重新描绘 componentDidUpdate:组件已经更新 销毁阶段： componentWillUnmount:组件即将销毁

### react和Vue的原理，区别，亮点，作用，

### **参考回答：**

vue的选择居于react与angular之间，框架自身的语法比**react多**一点，但是又比**angular少**一点。

react的简单在于，它的核心API其实非常少。所以我们会看到很多地方在说react其实是一个UI库，并不是一个完整的框架。他只是告诉我们如何创建组件以及组件之间如何进行数据传递。甚至于创建组件的方式正是使用ES6的class语法。因此开发中react的使用对于ES6的语法依赖非常高。因为react自身本来就没有多少强限制的语法。我们只需要掌握组件里的props，state，ref，生命周期，就好像没有过多额外的知识了。就连如果想要在jsx模板来遍历渲染，还得使用原生的map方法。

所以在我看来，react的最大特点就是简单并且与原生JavaScript非常接近。即给开发者带来的束缚非常少。一个功能的实现，如果你知道使用原生JavaScript如何实现，那么你就一定能够很轻松的知道使用react如何实现。

比如很多人在学习react的时候，会接触到一个倒计时的例子，这个例子使用修改组件中state的方式来实现。但是其实后来大家会慢慢知道，这种方式是非常错误的。因为state的每次修改，都会导致组件及其所有子组件的重新渲染。这是成本非常高的行为。当然，我还知道很多人，在调试react的时候，由于高频的重复渲染直接把浏览器都卡死的。这些问题都是尤雨溪所担心的限制过少带来的。

网上有的自以为牛逼的人，用着react/vue这样的框架，其实写着很烂的代码，恐怖的是他们还嘲讽这嘲讽那的。还遇到过一个人，口口声声说自己用了angular好多年，说angular真的好垃圾啊，性能好差啊，什么什么的各种黑，结果连track by都不会用。而react由于没有真正意义上的双向绑定。因此在处理一些复杂场景会非常麻烦，比如复杂的表单验证。

而相对而言，vue提供的能力则更多一点，这些便捷的能力会让初学者感觉到非常的幸福，因为很多效果只需要一些简单的代码既可以实现。我大概列举几条我个人认为非常棒的能力：

统一管理的计算属性

JavaScript的表达式非常便利，无论是vue还是react，表达式的能力是必不可少的。但正如vue官方文档所说，在模板中放入太多的逻辑会让模板过重且难以维护。而vue的组件中提供了一个计算属性computed来统一管理表达式。

<template>

<div id="example">

<p>Original message: "{{ message }}"</p>

<p>Computed reversed message: "{{ reversedMessage }}"</p>

</div>

</template>

<script>

export default {

name: 'example',

data () {

return {

message: 'Hello'}},

computed: {

reversedMessage: function() {

return this.message.split('').reverse().join('')

}}}

</script>

class的动态语法让我感觉非常爽

在实践中我们会发现非常多这样的场景，需要根据不同的状态来决定一个元素class的具体值。而如果仅仅只是简单的表达式或者条件判断在jsx模板中，例如下面这个样子就会让人感觉非常难受

<p className={active ? 'note active' : 'note'}></p>

当稍微复杂一点的逻辑还这样处理就是难受到忍不了了。而vue中支持的语法则非常轻松的搞定了这个问题。

// 可以放在任何你觉得舒服的位置

const pcls = {

active: active,

note: true}

<p class={pcls}></p>

这样我们继续添加更多的class名也不会造成额外的复杂度了。

当然，这仅仅只是一个工具方法就能搞定的问题，在使用react时，大家可以借助classnames来完成同样的功能。但vue是直接支持了。

双向绑定

由于react并不支持双向绑定，因此在复杂的表单验证时实现起来非常痛苦。而vue在以单向数据流为核心的同时，又没有完全抛弃双向绑定，这让在这样复杂的表单验证场景开发效率比react高出非常多。这也是vue省事儿的一个方面。

修饰符

我们在写事件处理逻辑时，常常需要e.preventDefault等操作。vue提供的修饰符功能可以帮助我们省去这些代码，极为方便。用多了就会发现，真TM好用。

<!-- 阻止单击事件冒泡 -->

<a v-on:click.stop="doThis"></a>

<!-- 提交事件不再重载页面 -->

<form v-on:submit.prevent="onSubmit"></form>

<!-- 修饰符可以串联  -->

<a v-on:click.stop.prevent="doThat"></a>

<!-- 只有修饰符 -->

<form v-on:submit.prevent></form>

<!-- 添加事件侦听器时使用事件捕获模式 -->

<div v-on:click.capture="doThis">...</div>

<!-- 只当事件在该元素本身（而不是子元素）触发时触发回调 -->

<div v-on:click.self="doThat">...</div>

当然，还有按键修饰符等，可以去官网进一步查看学习。

vue提供的方便可爱的语法糖还有很多，就不细说，大家可以在官网上一一体验。正如文章开头所说，vue会有一些语法限制，而这些语法限制在某种程度上来说降低了我们的开发成本，提高了开发效率。这大概也就是很多人认为vue更加简单易学的原因所在吧。

就从学习难易程度上来说，react之所以上手更加困难，主要的原因并不在于react本身，而在于围绕react的丰富的生态圈。正是由于react本身足够简单，所以我们需要掌握的react组件就更多。比如react-router，react-redux等。而且很多好用的，功能特别棒的组件在我们涉猎不广的时候都不知道。例如我在学习ant-design源码的时候，常常会惊讶于发现原来这里有一个组件可以这样用，真的好棒！而我在学习vue的时候又会惊讶的发现，原来这么棒的组件vue直接都已经支持了！

所以后来我才发现，原来vue与react既然如此相似。

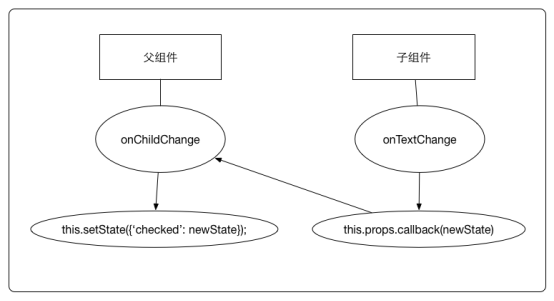
我仍然更加偏好于react。但仅仅只是因为react的语法更加接近于ES6而已。

### reactJs的组件交流

React组件之间的交流方式可以分为以下三种

1、父组件向子组件传值:主要是利用props来进行交流

2、子组件向父组件传值：子组件通过控制自己的state然后告诉父组件的点击状态。然后在父组件中展示出来，如图：



3、没有任何嵌套关系的组件之间传值：如果组件之间没有任何关系，组件嵌套层次比较深（个人认为 2 层以上已经算深了），或者你为了一些组件能够订阅、写入一些信号，不想让组件之间插入一个组件，让两个组件处于独立的关系。对于事件系统，这里有 2 个基本操作步骤：**订阅（subscribe）/监听（listen）**一个事件通知，并**发送（send）/触发（trigger）/发布（publish）/发送（dispatch）**一个事件通知那些想要的组件。

### 有了解过react的虚拟DOM吗，虚拟DOM是怎么对比的呢

当然是使用的diff算法，diff算法有三种优化形式：

tree diff：将**新旧两颗DOM树按照层级遍历**，只**对同级的DOM节点进行比较**，即同一父节点下的所有子节点，当发现节点已经不存在，则该节点及其子节点会被完全删除，不会进一步比较

component diff：**不同组件之间的对比**，如果组件类型相同，暂不更新，若组件类型不同，删除旧的组件，再创建一个新的组件，插入到删除组件的位置

element diff:在类型相同的组件内，再**继续对比组件内部的元素**。

### 怎么获取真正的dom

ReactDOM.findDOMNode()或this.refs

### react的生命周期函数

初始化

1、getDefaultProps()

设置默认的props，也可以用dufaultProps设置组件的默认属性.

2、getInitialState()

在使用es6的class语法时是没有这个钩子函数的，可以直接在constructor中定义this.state。此时可以访问this.props

3、componentWillMount()

组件初始化时只调用，以后组件更新不调用，整个生命周期只调用一次，此时可以修改state。

4、 render()

react最重要的步骤，创建虚拟dom，**进行diff算法，更新dom树都在此进行**。此时就不能更改state了。

5、componentDidMount()

组件渲染之后调用，**只调用一次**。

更新

6、componentWillReceiveProps(nextProps)

组件初始化时不调用，**组件接受新的props**时调用。

7、shouldComponentUpdate(nextProps, nextState)

react性能优化非常重要的一环。组件接受新的state或者props时调用，我们可以设置在此对比前后两个props和state是否相同，如果相同则返回false阻止更新，因为相同的属性状态一定会生成相同的dom树，这样就不需要创造新的dom树和旧的dom树进行diff算法对比，节省大量性能，尤其是在dom结构复杂的时候

8、componentWillUpdata(nextProps, nextState)

组件初始化时不调用，只有在组件将要更新时才调用，此时可以修改state

9、render()

组件渲染

10、componentDidUpdate()

组件初始化时不调用，组件更新完成后调用，此时可以获取dom节点。

卸载

11、componentWillUnmount()

组件将要卸载时调用，一些事件监听和定时器需要在此时清除。

### setState之后的流程

在代码中调用setState函数之后，React 会将传入的**参数对象与组件当前的状态合并**，然后**触发所谓的调和过程（Reconciliation）**。 经过调和过程，React 会以相对高效的方式**根据新的状态构建 React 元素树**并且着手**重新渲染整个UI界面**。 在 React 得到元素树之后，React 会自动计算出新的树与老树的节点差异，然后根据差异对界面进行最小化重渲染。在差异计算算法中，React 能够相对精确地知道哪些位置发生了改变以及应该如何改变，这就保证了**按需更新，而不是全部重新渲染**。

### react高阶组件知道吗？

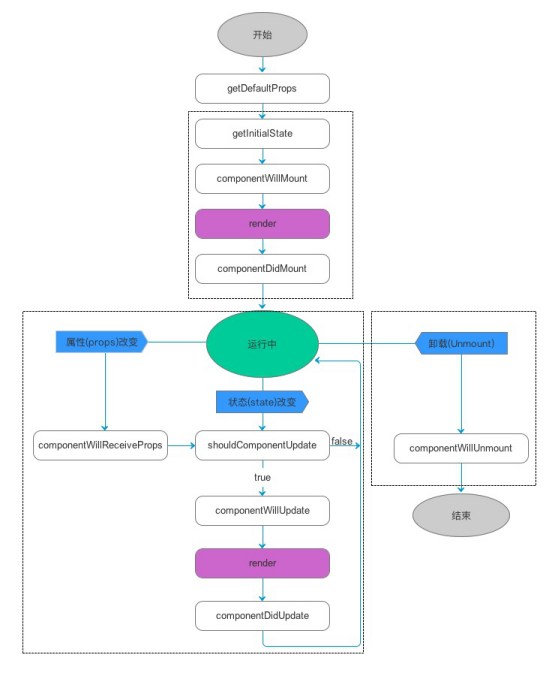
高阶组件**接收React组件作为参数**，并且**返回一个新的React组件**。高阶组件本质上也是一个函数，并不是一个组件。

### React的jsx，函数式编程

### React的生命周期

参考回答：

React 生命周期分为三种状态 1. 初始化 2.更新 3.销毁



### react

### 说说自己理解的react

React是用于构建用户界面的JavaScript库。React可以创建交互式UI。为应用程序中的每个状态建立的视图，并且React将在数据更改时进行更新，呈现正确的组件。另外，我们也可以构建管理自己状态的封装组件，然后将它们组合成复杂的UI。因为组件用JS编写而不是模板，所以可以通过应用传递数据，并使状态与DOM分离。

### react的组件是通过什么去判断是否刷新的

通过state是否改变

### React Native

### 其他移动APP开发框架（PhoneGap，AppCan，HTML5+，Framework7）