### 1. react和vue有哪些不同，说说你对这两个框架的看法

相同点

· 都支持服务器端渲染

· 都有Virtual DOM,组件化开发,通过props参数进行父子组件数据的传递,都实现webComponent规范

· 数据驱动视图

· 都有支持native的方案,React的React native,Vue的weex

不同点

· React严格上只针对MVC的view层,Vue则是MVVM模式

· virtual DOM不一样,vue会跟踪每一个组件的依赖关系,不需要重新渲染整个组件树.而对于React而言,每当应用的状态被改变时,全部组件都会重新渲染,所以react中会需要shouldComponentUpdate这个生命周期函数方法来进行控制

· 组件写法不一样, React推荐的做法是 JSX + inline style, 也就是把HTML和CSS全都写进JavaScript了,即'all in js'; Vue推荐的做法是webpack+vue-loader的单文件组件格式,即html,css,js写在同一个文件但是会区分开标签;

· 数据绑定: vue实现了数据的双向绑定,react数据流动是单向的

· state对象在react应用中不可变的,需要使用setState方法更新状态;在vue中,state对象不是必须的,数据由data属性在vue对象中管理

### 2. vue的双向绑定原理

1. Vue.js 采用的是 数据劫持+发布/订阅模式 的方式，通过 Object.defineProperty() 来劫持各个属性的 setter/getter， 在数据变动时发布消息给订阅者（Wacther）, 触发相应的监听回调
2. 简单实现

* <!DOCTYPE html>  
  <html lang="en">  
  <head>  
   <title>双向绑定最最最初级demo</title>  
   <meta charset="UTF-8">  
  </head>  
  <body>  
   <div id="app">  
   <input type="text" id="txt">  
   <p id="show-txt"></p>  
   </div>  
  </body>  
  <script>  
   var obj={}  
   Object.defineProperty(obj,'txt',{  
   get:function(){  
   return obj  
   },  
   set:function(newValue){  
   document.getElementById('txt').value = newValue  
   document.getElementById('show-txt').innerHTML = newValue  
   }  
   })  
   document.addEventListener('keyup',function(e){  
   obj.txt = e.target.value  
   })  
  </script>  
  </html>

### 2. vue的生命周期

1. 什么是vue的声明周期 Vue 实例从创建到销毁的过程，就是生命周期。也就是从开始创建、初始化数据、编译模板、挂载Dom→渲染、更新→渲染、卸载等一系列过程，我们称这是 Vue 的生命周期。
2. 组件第一次加载会触发哪些声明周期函数 第一次页面加载时会触发 beforeCreate, created, beforeMount, mounted 这几个钩子
3. DOM 渲染在 哪个周期中就已经完成 DOM 渲染在 mounted 中就已经完成了。
4. 声明周期各个函数的作用 生命周期钩子的一些使用方法： beforecreate : 可以在这加个loading事件，在加载实例时触发 created : 初始化完成时的事件写在这里，如在这结束loading事件，异步请求也适宜在这里调用 mounted : 挂载元素，获取到DOM节点 updated : 如果对数据统一处理，在这里写上相应函数 beforeDestroy : 可以做一个确认停止事件的确认框 nextTick : 更新数据后立即操作dom
5. 当keep组件激活时会调用哪个生命周期的函数 activated 该钩子在服务器端渲染期间不被调用。
6. 当keep-alive组件停用时调用。 deactivated 该钩子在服务端渲染期间不被调用。
7. keep的调用时机

activated调用时机：

第一次进入缓存路由/组件，在mounted后面，beforeRouteEnter守卫传给 next 的回调函数之前调用：

beforeMount=> 如果你是从别的路由/组件进来(组件销毁destroyed/或离开缓存deactivated)=>  
mounted=> activated 进入缓存组件 => 执行 beforeRouteEnter回调  
  
因为组件被缓存了，再次进入缓存路由/组件时，不会触发这些钩子：  
  
// beforeCreate created beforeMount mounted 都不会触发。  
所以之后的调用时机是：  
  
组件销毁destroyed/或离开缓存deactivated => activated 进入当前缓存组件   
=> 执行 beforeRouteEnter回调  
// 组件缓存或销毁，嵌套组件的销毁和缓存也在这里触发

1. 组件缓存和不缓存调用的生命周期

（1）不缓存： 进入的时候可以用created和mounted钩子，离开的时候用beforeDestory和destroyed钩子,beforeDestory可以访问this，destroyed不可以访问this。

（2）缓存了组件： 缓存了组件之后，再次进入组件不会触发beforeCreate、created 、beforeMount、 mounted，如果你想每次进入组件都做一些事情的话，你可以放在activated进入缓存组件的钩子中。 同理：离开缓存组件的时候，beforeDestroy和destroyed并不会触发，可以使用deactivated离开缓存组件的钩子来代替

1. 完整的触发顺序

* 触发钩子的完整顺序：  
    
  将路由导航、keep-alive、和组件生命周期钩子结合起来的，触发顺序，假设是从a组件离开，第一次进入b组件：  
    
  1. beforeRouteLeave:路由组件的组件离开路由前钩子，可取消路由离开。  
    
  2. beforeEach: 路由全局前置守卫，可用于登录验证、全局路由loading等。  
    
  3. beforeEnter: 路由独享守卫  
    
  4. beforeRouteEnter: 路由组件的组件进入路由前钩子。  
    
  5. beforeResolve:路由全局解析守卫  
    
  6. afterEach:路由全局后置钩子  
    
  7. beforeCreate:组件生命周期，不能访问this。  
    
  8. created:组件生命周期，可以访问this，不能访问dom。  
    
  9. beforeMount:组件生命周期  
    
  10. deactivated: 离开缓存组件a，或者触发a的beforeDestroy和destroyed组件销毁钩子。  
    
  11. mounted:访问/操作dom。  
    
  12. activated:进入缓存组件，进入a的嵌套子组件(如果有的话)。  
    
  13. 执行beforeRouteEnter回调函数next。

### 3. vue的computed属性

1. computed用来监控自己定义的变量，该变量不在data里面声明，直接在computed里面定义，然后就可以在页面上进行双向数据绑定展示出结果或者用作其他处理
2. computed比较适合对多个变量或者对象进行处理后返回一个结果值，也就是数多个变量中的某一个值发生了变化则我们监控的这个值也就会发生变化，举例：购物车里面的商品列表和总金额之间的关系，只要商品列表里面的商品数量发生变化，或减少或增多或删除商品，总金额都应该发生变化。这里的这个总金额使用computed属性来进行计算是最好的选择
3. computed和watch的区别 watch主要用于监控vue实例的变化，它监控的变量当然必须在data里面声明才可以，它可以监控一个变量，也可以是一个对象

* data(){  
  　　　return {  
  　　　　　　　　　　example0:"",  
  　　　　　　　　　　example1:"",  
  　　　　　　　　　　example2:{  
  　　　　　　　　　　　　inner0:1, 　　　　　　　　　  
   　　　innner1:2 　　　　　　　　　  
   　}  
  　　　　　　}  
  　　　　},  
  watch:{  
  　example0(newVal,oldVal){//监控单个变量  
   ……  
   }，  
  example2(newVal,oldVal){//监控对象  
   ……  
   }，  
  }
* watch一般用于监控路由、input输入框的值特殊处理等等，它比较适合的场景是一个数据影响多个数据

计算属性可用于快速计算视图（View）中显示的属性。这些计算将被缓存，并且只在需要时更新

### 4. vuex

1. 什么时候需要使用vuex : 当项目业务复杂 页面之间很多关联数据需要交互的时候
2. 本地存储和vuex的区别
   1. vuex是状态管理，是为了解决跨组件之间数据共享问题的，一个组件的数据变化会映射到使用这个数据的其他组件当中。如果刷新页面，之前存储的vuex数据全部都会被初始化掉
   2. localStorage是H5提供的一个更简单的数据存储方式，之前是用cookie存放数据，但是cookie的数据量太小，所以就用localStorage，它可以有5M的限制，不受刷新页面的控制，长久保存
   3. 所以，在用vue进行项目开发的时候，什么时候用到vuex呢？

* 当应用遇到多个组件共享状态时候，即：多个视图依赖于同一个状态，不同视图的行为需要变更同一状态。
* vuex的官网也说了，对于页面之间的传参对于多层嵌套组件将会很繁琐，而且对于兄弟组件之间的状态传递无能为力。所以就将这些组件的共享状态抽取出来，以一个全局单例模式管理，即vuex
* 1.【响应式】vuex的状态存储是响应式的，当Vue组件从store中读取状态的时候，若store中的状态发生变化，那么相应的组件也会得到高效更新。  
    
  2.【不能直接改变store】不能直接改变store的变化，改变store中状态的唯一途径是commit mutation。方便于跟踪每一个状态的变化
  1. 但是localstorage改变了不会自动动态更新视图需要手动获取更新

### 5. vuex中常见的属性

1. vuex有哪几种属性 有五种，分别是 State、 Getter、Mutation 、Action、 Module
2. vuex的State特性是？
   1. Vuex就是一个仓库，仓库里面放了很多对象。其中state就是数据源存放地，对应于与一般Vue对象里面的data
   2. state里面存放的数据是响应式的，Vue组件从store中读取数据，若是store中的数据发生改变，依赖这个数据的组件也会发生更新
   3. 它通过mapState把全局的 state 和 getters 映射到当前组件的 computed 计算属性中
3. vuex的Getter特性是？
   1. getters 可以对State进行计算操作，它就是Store的计算属性
   2. 虽然在组件内也可以做计算属性，但是getters 可以在多组件之间复用
   3. 如果一个状态只在一个组件内使用，是可以不用getters
4. vuex的Mutation特性是？
   1. Action 类似于 mutation，不同在于：
   2. Action 提交的是 mutation，而不是直接变更状态。
   3. Action 可以包含任意异步操作
5. Vue.js中ajax请求代码应该写在组件的methods中还是vuex的actions中？
   1. 如果请求来的数据是不是要被其他组件公用，仅仅在请求的组件内使用，就不需要放入vuex 的state里。
   2. 如果被其他地方复用，这个很大几率上是需要的，如果需要，请将请求放入action里，方便复用，并包装成promise返回，在调用处用async await处理返回的数据。如果不要复用这个请求，那么直接写在vue文件里很方便。
6. vuex中如何更新状态
   1. 通过store.commit('状态名',状态值);
7. vuex中如果获取更新后的状态
   1. 通过vuex的store.getters.状态名

### 6. vue的导航守卫

1. 导航守卫的概念
   1. 什么时候需要用到导航守卫 简单来说就是当登录系统的角色权限不一样的时候通过导航守卫来控制当前角色能够看到哪些页面(权限控制、登录验证等)
2. 导航守卫的常见钩子函数(生命周期函数) 全局守卫 vue-router全局有三个守卫：

* router.beforeEach 全局前置守卫 进入路由之前 router.beforeResolve 全局解析守卫(2.5.0+) 在beforeRouteEnter调用之后调用 router.afterEach 全局后置钩子 进入路由之后 使用方法：
* // main.js 入口文件  
   import router from './router'; // 引入路由  
   router.beforeEach((to, from, next) => {   
   next();  
   });  
   router.beforeResolve((to, from, next) => {  
   next();  
   });  
   router.afterEach((to, from) => {  
   console.log('afterEach 全局后置钩子');  
   });
* 路由组件内的守卫 beforeRouteEnter 进入路由前 beforeRouteUpdate (2.2) 路由复用同一个组件时 beforeRouteLeave 离开当前路由时 文档中的介绍：
* beforeRouteEnter (to, from, next) {  
   // 在路由独享守卫后调用 不！能！获取组件实例 `this`，组件实例还没被创建  
   },  
   beforeRouteUpdate (to, from, next) {  
   // 在当前路由改变，但是该组件被复用时调用 可以访问组件实例 `this`  
   // 举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2 之间跳转的时候，  
   // 由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用。  
   },  
   beforeRouteLeave (to, from, next) {  
   // 导航离开该组件的对应路由时调用，可以访问组件实例 `this`  
   }
* to,from,next 这三个参数的概念 to和from是将要进入和将要离开的路由对象,路由对象指的是平时通过this.$route获取到的路由对象。 next:Function 这个参数是个函数，且必须调用，否则不能进入路由(页面空白)。 next() 进入该路由。 next(false): 取消进入路由，url地址重置为from路由地址(也就是将要离开的路由地址)。 next 跳转新路由，当前的导航被中断，重新开始一个新的导航。 我们可以这样跳转：next('path地址')或者next({path:''})或者next({name:''}) 且允许设置诸如 replace: true、name: 'home' 之类的选项 以及你用在router-link或router.push的对象选项。

1. 导航守卫的流程

完整的路由导航解析流程(不包括其他生命周期)：

触发进入其他路由。 调用要离开路由的组件守卫beforeRouteLeave 调用局前置守卫：beforeEach 在重用的组件里调用 beforeRouteUpdate 调用路由独享守卫 beforeEnter。 解析异步路由组件。 在将要进入的路由组件中调用beforeRouteEnter 调用全局解析守卫 beforeResolve 导航被确认。 调用全局后置钩子的 afterEach 钩子。 触发DOM更新(mounted)。 执行beforeRouteEnter 守卫中传给 next 的回调函数