# React和vue对比：

相同点：

1.都支持服务器端渲染

2.都有虚拟DOM（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

3.都有支持native的方案，react的react native；vue的weex

4.集成了路由工具，打包工具和状态管理工具

不同：

1.react严格上只针对MVC的view层，vue则是MVVM模式。react是数据单项流动；vue是数据的双向绑定；

MVVM模式详解：model-view-viewmodel（模型（后端传递的数据）-视图（所看到的的页面）-视图模型（连接view和model的桥梁有两个方向：模型-->视图即将后端数据转化成看到的页面；实现方式是数据绑定；视图-->模型：将看到的页面转化成后端数据；实现方式：dom事件监听。两个方向都实现的是双向绑定））是一个软件架构设计模式。MVVM框架下视图和模型不直接通信，通过viewModel通信

model:模型，model中定义数据修改和操作的业务逻辑

view：视图，将数据模型转换成UI展示出来

viewmodel：view和model之间的桥梁。

MVVM架构下view和model不直接联系通过viewmodel交互。因为viewmodel通过双向数据绑定把view层和model层连接起来，view和model之间的同步工作是自动的，无需人工干涉也就是说开发者只需要关注业务逻辑而不需要手动操作DOM，复杂的数据状态维护由MVVM统一管理；

MVVM与jQuery区别：vue是数据驱动通过数据显示视图而不是节点操作；

MVC模式详解：model-view-controller模型视图控制器；MVC是单向通信，即view和model，必须通过controller承上启下。

2.virtual DOM不一样，vue会跟踪每个组件的依赖关系，不需要重新渲染整个组件树；

react每当组件发生变化时，全部组建重新渲染，所以react会需要重写shouldComponentUpdate这个生命周期函数方法减少不必要的DOM操作

3.组件写法不一样：react：jsx语法把HTML和css都写进js

vue是将HTML+css+js写入一个文件

4.state对象在react中不可变，需要使用setState方法更新状态

vue中state对象不是必须的，数据由data属性在vue对象中管理

**vue.cli怎样使用自定义组件？**

在src/components目录新建组件文件myVue.vue，script一定要export default{}-->在需要用的页面中导入import myVue from './components/myVue'(可以省略vue后缀)-->注入到需要用的页面中组件的components属性上面components:{myVue}-->在template视图中使用<myVue />

# 简述你知道的指令

v-once:只渲染一次

v-bind绑定HTML属性，动态更新HTML元素属性；因为HTML属性不能用{{}}；:简写

v-if（条件块中事件监听器和子组件适当的重建和销毁） v-show条件渲染

v-for列表渲染

v-on事件处理@简写

v-model可以获取input的值也可以改变input的值

v-model实现双向数据绑定：

输入框变化，data中数据也同步变化 view->model

data中数据变化，文本节点内容同步变化 model->view

# vue的computed和methods

计算属性（computed）：计算属性是基于它们的依赖进行缓存的。只有它们依赖的值发生改变时才会更新相关dom；依赖的数据没有发生改变，就可以直接返回缓存里的数据。

而methods：方法每调用一次就重新计算一次。所以做大量计算时最好使用计算属性。

watch：用于观察和响应vue实例上的数据变动。在表单中应用比较多；

**filter过滤器**

filter:template中{{value|FF}} data同级：filters:{FF:function(val){...}}

**directive:自定义指令：**注册全局自定义指令：Vue.directive('focus',{钩子函数});

bind：只调用一次，指令第一次绑定到元素时调用。在这里可以进行一次性的初始化设置。

inserted：被绑定元素插入父节点时调用 (仅保证父节点存在，但不一定已被插入文档中)。

update：所在组件的 VNode 更新时调用，但是可能发生在其子 VNode 更新之前。指令的值可能发生了改变，也可能没有。但是你可以通过比较更新前后的值来忽略不必要的模板更新 (详细的钩子函数参数见下)。

componentUpdated：指令所在组件的 VNode 及其子 VNode 全部更新后调用。

unbind：只调用一次，指令与元素解绑时调用。

# Vue的双向数据绑定原理

vue.js 是采用数据劫持结合发布者-订阅者模式的方式，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，触发相应的监听回调。

**具体步骤：**

**第一步：需要observe的数据对象进行递归遍历**，包括子属性对象的属性，都加上 setter和getter  
这样的话，给这个对象的某个值赋值，就会触发setter，那么就能监听到了数据变化

**第二步：compile解析模板指令**，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图

**第三步：Watcher订阅者是Observer和Compile之间通信的桥梁**，主要做的事情是:  
1、在自身实例化时往属性订阅器(dep)里面添加自己  
2、自身必须有一个update()方法  
3、待属性变动dep.notice()通知时，能调用自身的update()方法，并触发Compile中绑定的回调，则功成身退。

**第四步：MVVM作为数据绑定的入口，整合Observer、Compile和Watcher三者**，通过Observer来监听自己的model数据变化，通过Compile来解析编译模板指令，最终利用Watcher搭起Observer和Compile之间的通信桥梁，达到数据变化 -> 视图更新；视图交互变化(input) -> 数据model变更的双向绑定效果。

**vue组件之间参数的传递**

1.父--子：子组件通过props接收数据

2.子--父：$emit方法传递参数

子组件写 方法：比如点击按钮的函数中写this.$emit('getmsg',this.data00);第一个参数是自定义事件名称；

父组件写 v-on监听自定义方法：{{childData}}<child @getmsg="get0"></child> get0(data)中写this. childData=data;

v-model传递参数：区别是使用语法糖：this.$emit(input,this.data00);父级中直接写{{childData}}<child v-model=" childData "></child>

**直接引入vue.js和使用vue-cli区别：**

直接引入vue.js是将vue.js的代码引入到自己的代码中，里面直接有一个vue全局变量，可以通过new Vue来创建实例。

使用脚手架是模块化开发，用构建工具webpack载入模块，自动安装好基础的依赖库，能够更快的构建一个vue项目

# vue 的spa

**单页Web应用（single page web application，SPA）：**

就是只有一张Web页面的应用。单页应用程序 (SPA) 是加载单个HTML 页面并在用户与应用程序交互时动态更新该页面的Web应用程序。浏览器一开始会加载必需的HTML、CSS和JavaScript，所有的操作都在这张页面上完成，都由JavaScript来控制。因此，对单页应用来说模块化的开发和设计显得相当重要。

**单页Web应用的优点：**

1、提供了更加吸引人的用户体验：具有桌面应用的即时性、网站的可移植性和可访问性。

2、单页应用的内容的改变不需要重新加载整个页面，web应用更具响应性和更令人着迷。

3、单页应用没有页面之间的切换，就不会出现“白屏现象”,也不会出现假死并有“闪烁”现象

4、单页应用相对服务器压力小，服务器只用出数据就可以，不用管展示逻辑和页面合成，吞吐能力会提高几倍。

5、良好的前后端分离。后端不再负责模板渲染、输出页面工作，后端API通用化，即同一套后端程序代码，不用修改就可以用于Web界面、手机、平板等多种客户端。

**单页Web应用的缺点：**

1、首次加载耗时比较多。

2、SEO问题，不利于百度，360等搜索引擎收录。

3、容易造成Css命名冲突。

4、前进、后退、地址栏、书签等，都需要程序进行管理，页面的复杂度很高，需要一定的技能水平和开发成本高。

# Vue路由

路由器实例方法：

router实例提供了很多方法用来提供启动（start(App,el)）路由映射（Map(routerMap)）重定向（go(path)）路由切换全局钩子等功能。

在使用vue-router的应用中，路由对象会被注入到每个组件中，赋值为this.$route并且当路由切换时，路由对象会被更新。

vue-router响应路由参数变化：

使用侦听器：watch 来监听$route对象



**动态路由匹配：**

我们可以在实例VueRouter的路由路径中使用“动态路径参数”(dynamic segment)来达到这个效果。

routes: [

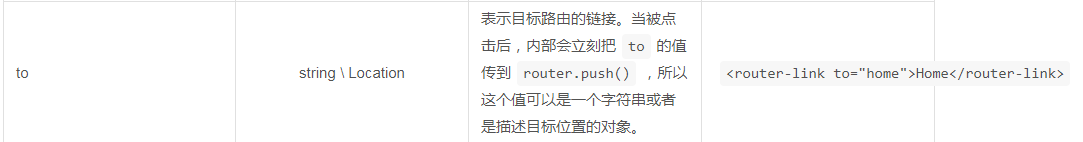
// 动态路径参数 以冒号开头

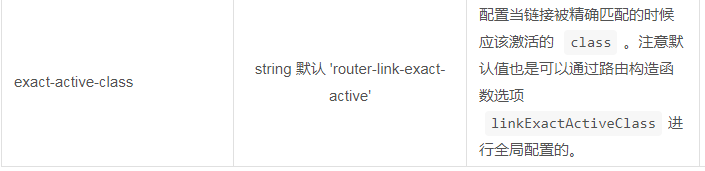
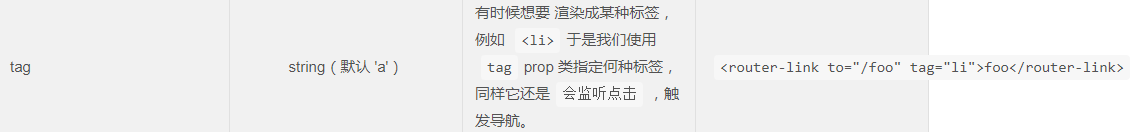
{ path: '/user/:id', component: User }

]

**嵌套路由：**

在需要被渲染的组件中嵌套<router-view>组件，在VueRouter的参数中使用children数组配置





**动态加载路由组件（按需加载即懒加载）**：当打包构建应用时，Javascript 包会变得非常大，影响页面加载。如果我们能把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件，这样就更加高效了。

使用AMD规范

比如用网易云的时候音乐播放和歌词呈现其实并不相关

const LRC = Vue.component("lrc",(resolve)=>require(["../components/LRC"],resolve))

# VUE的diff算法

真实DOM：每次dom中数据发生变化都会引起整颗dom树的重绘

虚拟DOM：当虚拟dom某个节点发生变化，将新的Vnode和old Vnode 对比如果有变化就修改真实DOM，同时oldVnode修改为Vnode。

diff算法的流程

调用patch函数对比Vnode和OldVnode是否值得比较，若不值得比较则直接替换return Vnode-值得比较则调用PatchNode函数比较：若Vnode有子节点但是O没有则将V的子节点转化为真实节点添加到el上；若Vnode没有子节点但是O有，删除el上的子节点；若两者均有文本节点但不同，将el上的文本节点替换为Vnode上的文本节点；若两者均有子节点-调用updateChildren函数：提取出两者的子节点并且标注首尾为 OldS OldE S E进行分别比较（四次），然后向中间靠拢，直到有开始索引大于结束索引表示其中一方结束遍历，则结束比较。