**Vue面试题**

**一、基本知识点**

1. **Vue.js介绍**

Vue.js是JavaScript MVVM（Model-View-ViewModel）库，十分简洁，Vue核心只关注视图层，相对AngularJS提供更加简洁、易于理解的API。Vue尽可能通过简单的API实现响应的数据绑定和组合的视图组件。

1. **Vue和MVVM模式**

MVVM模式即Model-View-ViewModel。

Vue是以数据为驱动的，Vue自身将DOM和数据进行绑定，一旦创建绑定，DOM和数据将保持同步，每当数据发生变化，DOM会跟着变化。

ViewModel是Vue的核心，它是Vue的一个实例。Vue实例时作用域某个HTML元素上的，这个HTML元素可以是body，也可以是某个id所指代的元素。

DOM Listeners和Data Bindings是实现双向绑定的关键。DOM Listeners监听页面所有View层DOM元素的变化，当发生变化，Model层的数据随之变化；Data Bindings监听Model层的数据，当数据发生变化，View层的DOM元素随之变化。

1. **Vue.js特点**

**简洁：**页面由HTML模板+Json数据+Vue实例组成  
**数据驱动：**自动计算属性和追踪依赖的模板表达式  
**组件化：**用可复用、解耦的组件来构造页面  
**轻量：**代码量小，不依赖其他库  
**快速：**精确有效批量DOM更新  
**模板友好：**可通过npm，bower等多种方式安装，很容易融入

1. **vue.js特性**

I: MVVM模式（数据变量（model）发生改变 视图（view）也改变， 视图（view）改变，数据变量（model）也发生改变） 使用MVVM模式有几大好处：

1. 低耦合。View可以独立于Model变化和修改，一个ViewModel可以绑定到不同的View上，当View变化的时候Model可以不变，当Model变化的时候View也可以不变。 2. 可重用性。可以把一些视图的逻辑放在ViewModel里面，让很多View重用这段视图逻辑。
2. 独立开发。开发人员可以专注与业务逻辑和数据的开发(ViewModel)。设计人员可以专注于界面(View)的设计。

4. 可测试性。可以针对ViewModel来对界面(View)进行测试 II: 组件化 III 指令系统 IIII: vue2.0开始支持虚拟dom vue1.0是操作的是真的dom元素而不是虚拟的 虚拟dom:可以提升页面的刷新速度

虚拟DOM有利也有弊。

A：大小 - 其中之一就是更多的功能意味着代码包中更多行的代码。幸运的是，Vue.js 2.0 依旧比 较小（当前版本 21.4kb），并 且也正在删除很多东西。

B: 内存 -同样，虚拟DOM需要将现有的DOM拷贝后保存在内存中，这是一个在DOM更新速度和内存使用中的权衡。

C: 并不适用所有情况 -如果虚拟DOM可以一次性进行批量的修改是非常好的。但是如果是单独的、稀少的更新呢？这样的任何 DOM更新都将会使虚拟DOM带来无意义的预计算

**二、常用指令**

**1.Vue.js常用指令**

**v-if指令,v-show指令,v-else指令,v-for指令,v-bind指令,v-on指令**

**2.v-show指令，v-if的区别**

条件渲染指令，与v-if不同的是，无论v-show的值为true或false，元素都会存在于HTML代码中；而只有当v-if的值为true，元素才会存在于HTML代码中。v-show指令只是设置了元素CSS的style值

**3.指令keep-alive**

在vue-router写着keep-alive的含义：

如果把切换出去的组件保留在内存中，可以保留它的状态或避免重新渲染。为此可以添加一个keep-alive指令。

**4.指令v-el的使用 (1,0 ，2.0已经弃用)**

有时候我们想就像使用jquery那样去访问一个元素，此时就可以使用v-el指令，去给这个元素注册一个索引，方便通过所属实例的$el访问这个元素。

注意 HTML不区分大小写，所以v-el:someEl将转换为全小写。可以用v-el:some-el然后设置this.$el.someEl。

示例

hello

world

this.$els.msg.textContent // ->"hello"

this.$els.otherMsg.textContent // ->"world"

this.$els.msg //->hello

**5.如何让css只在当前组件中起作用**

在每一个vue组件中都可以定义各自的css，js，如果希望组件内写的css只对当前组件起作用，只需要在style中写入scoped，即：<style scoped></style>

## 8.关于vuejs页面闪烁{{message}}

在vuejs指令中有v-cloak，这个指令保持在元素上直到关联实例结束编译。和CSS规则如[v-cloak]{display:none}一起用时，这个指令可以隐藏未编译的Mustache标签直到实例准备完毕。用法如下：

[v-cloak]{

display:none;}

<div v-cloak>{{message}}</div>

这样<div>不会显示，直到编译结束

## 9.Vuejs在变化检测问题

#### 1.检测数组

#### 由于javascript的限制，vuejs不能检测到下面数组的变化：

直接索引设置元素，如vm.item[0]={};

修改数据的长度，如vm.item.length。

为了解决

问题1，Vuejs扩展了观察数组，为它添加一个$set()方法：

// 与 `example1.items[0] = ...` 相同，但是能触发视图更新

example1.items.$set(0, { childMsg: 'Changed!'})

问题2，需要一个空数组替换items。

除了$set()，vuejs也为观察数组添加了$remove()方法，用于从目标数组中查找并删除元素，在内部调用了splice()。因此，不必：

var index = this.items.indexOf(item)if (index !== -1) {

this.items.splice(index, 1)

}

只需：

this.items.$remove(item);

#### 2.检测对象

受ES5的显示，Vuejs不能检测到对象属性的添加或删除。因为Vuejs在初始化时候将属性转化为getter/setter，所以属性必须在data对象才能让Vuejs转换它，才能让它是响应的，例如：

var data = { a: 1 }var vm = new Vue({

data: data

})// `vm.a` 和 `data.a` 现在是响应的

vm.b = 2// `vm.b` 不是响应的

data.b = 2// `data.b` 不是响应的

不过，有办法在实例创建之后添加属性并且让它是响应的。对于Vue实例，可以使用$set(key,value)实例方法：

vm.$set('b', 2)// `vm.b` 和 `data.b` 现在是响应的

对于普通数据对象，可以使用全局方法Vue.set(object, key, value):

Vue.set(data, 'c', 3)// `vm.c` 和 `data.c` 现在是响应的

有时你想向已有对象上添加一些属性，例如使用 Object.assign() 或 \_.extend() 添加属性。但是，添加到对象上的新属性不会触发更新。这时可以创建一个新的对象，包含原对象的属性和新的属性：

// 不使用 `Object.assign(this.someObject, { a: 1, b: 2 })`this.someObject = Object.assign({}, this.someObject, { a: 1, b: 2 })

## 11.片段实例

下面几种情况会让实例变成一个片断实例：

模板包含多个顶级元素。

模板只包含普通文本。

模板只包含其它组件（其它组件可能是一个片段实例）。

模板只包含一个元素指令，如<partial> 或 vue-router 的 <router-view>。

模板根节点有一个流程控制指令，如 v-if 或 v-for。

这些情况让实例有未知数量的顶级元素，它将把它的 DOM 内容当作片断。片断实例仍然会正确地渲染内容。不过，它没有一个根节点，它的$el 指向一个锚节点，即一个空的文本节点（在开发模式下是一个注释节点）。  
但是更重要的是，组件元素上的非流程控制指令，非 prop 特性和过渡将被忽略，因为没有根元素供绑定：

<!-- 不可以，因为没有根元素 --><example v-show="ok" transition="fade"></example>

<!-- props 可以 --><example :prop="someData"></example>

<!-- 流程控制可以，但是不能有过渡 --><example v-if="ok"></example>

片段实例也有用处，但是通常情况下组件有一个根节点比较好，它会保证组件元素上的指令和特性能正确的转换，同时性能也稍微好些。

## 12.vuejs循环插入图片

在写循环的时候，写入如下代码：

<div class="bio-slide" v-for="item in items">

<img src="{{item.image}}"></div>

此时在控制台会出现警告  
[Vue Warn]: src="{{item.image}}": interpolation in "src" attribute will cause a 404 request. Use v-bind:src instead.这里意思是在“src”属性插值将导致404请求。使用v-bind：src代替。  
所以替换成如下：

<div class="bio-slide" v-for="item in items">

<img v-bind:src="item.image"></div>

这里需要主要，v-bind在写的时候不能再用{{}}，根据官方的说法：

<a v-bind:href="url"></a>

这里 href 是参数，它告诉 v-bind 指令将元素的 href 特性跟表达式 url 的值绑定。可能你已注意到可以用特性插值href="{{url}}" 获得同样的结果：这样没错，并且实际上在内部特性插值会转为 v-bind 绑定。

**15.vue 1.0 和2.0生命周期的对比。**

1. **路由**
2. **介绍一下vue路由**

vue-router是Vue.js官方的路由插件，它和vue.js是深度集成的，适合用于构建单页面应用。vue的单页面应用是基于路由和组件的，路由用于设定访问路径，并将路径和组件映射起来。传统的页面应用，是用一些超链接来实现页面切换和跳转的。在vue-router单页面应用中，则是路径之间的切换，也就是组件的切换。

**2.路由嵌套**

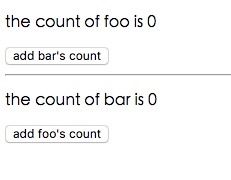
路由嵌套会将其他组件渲染到该组件内，而不是进行整个页面跳转。router-view本身就是将组件渲染到该位置，想要进行页面跳转，就要将页面渲染到根组件。

1. **vue组件**
2. **vue非父子组件如何进行通讯？**

组件是 vue 的核心部分，而组件之间通信方式是必不可少的。 父子之间的通信方式很简单，父组件通过 props 向子组件传值，而子组件通过自定义事件把数据传递回父组件，那么非父子关系组件怎么进行通信？ Vue2.x 废弃了 broadcast 和 dispatch 之后，可以通过 vuex ，还有 event bus 来解决。这里不讲 vuex ，讲起来是另外一个话题，就讲一下怎么在非父子组件之间通过 event bus 进行通信。

### demo

首先我们要实现的效果是



上下分别是 foo组件和 bar 组件，它们之间是非父子关系，分别点击各自的 button ，另一个组件的 count 对应增加。

代码。。。。。

以上就实现了一个简易的非父子组件之间的通信方式。通过 event bus ，在一个组件创建时的钩子函数中监听某个事件，而在需要与其进行通信的组件中触发这个函数，同时交换数据。

当然，event bus 只适于某些不复杂的场景，在需要频繁进行组件通信的情况下，还是应该尽量使用 Vuex ，不仅使用上更加简单，同时数据流的流向也会相对清晰。

**五．Vue和其他两个框架的区别**

**1.Vue.js 和 AngularJS 之间的区别是什么?**

下面是一些选择 Vue 而不是 Angular 的可能原因；

Vue.js 是一个更加灵活开放的解决方案。它允许你以希望的方式组织你的应用程序，而不是任何时候都必须遵循 Angular 制定的规则。它仅仅是一个视图层，所以你可以将它嵌入一个现有页面而不一定要做成一个庞大的单页应用。在结合其他库方面它给了你更大的的空间，但相应，你也需要做更多的架构决策。例如，Vue.js 核心默认不包含路由和 ajax 功能，并且通常假定你在用应用中使用了一个外部的模块构建系统。这可能是最重要的区别。 在 API 和内部设计方面，Vue.js 比 Angular 简单得多, 因此你可以快速地掌握它的全部特性并投入开发。

Vue.js 拥有更好的性能，因为它不使用脏检查。当 watcher 越来越多时, Angular 会变得越来越慢，因为作用域内的每一次数据变更，所有的 watcher 都需要被重新求值。Vue 则根本没有这个问题，因为它采用的是基于依赖追踪的观察系统，所以所有的数据变更触发都是独立的，除非它们之间有明确的依赖关系。

Vue.js 中指令和组件的概念区分得更为清晰。指令只负责封装 DOM 操作，而组件代表一个自给自足的独立单元 —— 它拥有自己的视图和数据逻辑。在 Angular 中它们两者间有不少概念上的混淆。

**2.Vue.js 和 React.js 有什么区别?**

React.js 和 Vue.js 确实有一些相似——它们都提供数据驱动、可组合搭建的视图组件。然而，它们的内部实现是完全不同的。React 是基于 Virtual DOM——一种在内存中描述 DOM 树状态的数据结构。React 中的数据通常被看作是不可变的，而 DOM 操作则是通过 Virtual DOM 的 diff 来计算的。与之相比，Vue.js 中的数据默认是可变的，而数据的变更会直接出发对应的 DOM 更新。相比于 Virtual DOM，Vue.js 使用实际的 DOM 作为模板，并且保持对真实节点的引用来进行数据绑定。

Virtual DOM 提供了一个函数式的描述视图的方法，这很 cool。因为它不使用数据观察机制，每次更新都会重新渲染整个应用，因此从定义上保证了视图通与数据的同步。它也开辟了 JavaScript 同构应用的可能性。

实话实说，我自己对 React 的设计理念也是十分欣赏的。但 React 有一个问题就是组件的逻辑和视图结合得非常紧密。对于部分开发者来说，他们可能觉得这是个优点，但对那些像我一样兼顾设计和开发的人来说，还是更偏好模板，因为模板能让我们更好地在视觉上思考设计和 CSS。JSX 和 JavaScript 逻辑的混合干扰了我将代码映射到设计的思维过程。相反，Vue.js 通过在模板中加入一个轻量级的 DSL (指令系统)，换来一个依旧直观的模板，且能够将逻辑封装进指令和过滤器中。

React 的另一个问题是：由于 DOM 更新完全交由 Virtual DOM 管理，当你真的想要自己控制 DOM 是就有点棘手了（虽然理论上你可以，但这样做时你本质上在对抗 React 的设计思想）。对于需要复杂时间控制的动画来说这就变成了一项很讨人厌的限制。在这方面，Vue.js 允许更多的灵活性，并且有不少用 Vue.js 构建的富交互实例。