# 面试题

## MVVM

相似问题：

### 什么是MVVM?

### 谈一谈你对vue中MVVM的理解

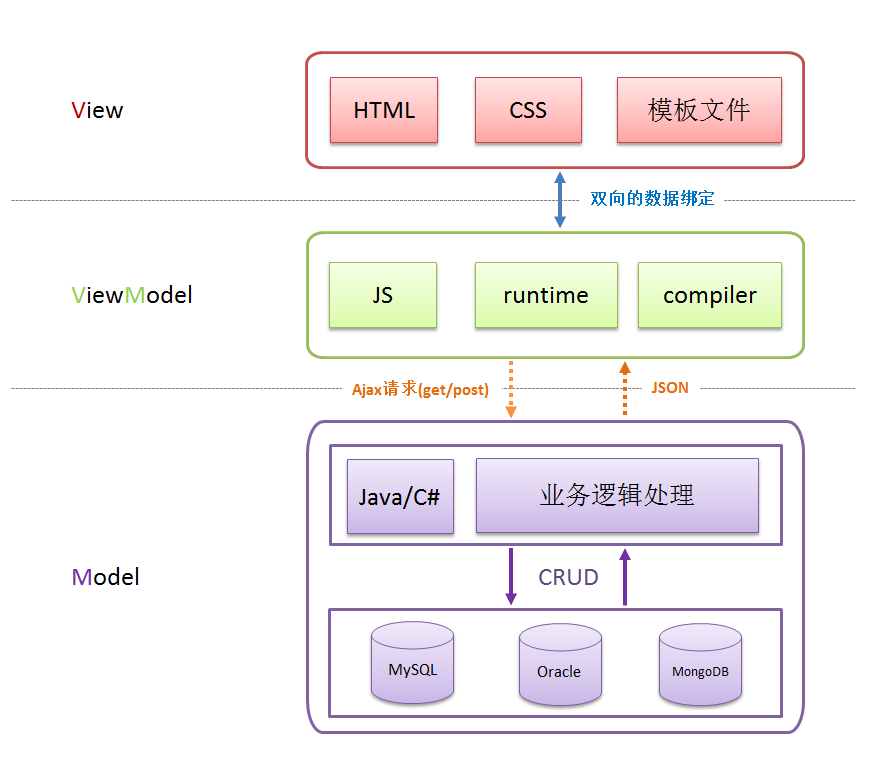
M就是：模型（Model）：数据模型；负责数据存储。泛指后端进行的各种业务逻辑处理和数据操控，主要围绕数据库系统展开。

V就是：View 视图，负责页面展示，也就是用户界面。主要由 HTML 和 CSS 来构建

VM就是：视图模型（View-Model）：负责业务逻辑处理（比如Ajax请求等），对数据进行加工后交给视图展示

通过vue类创建的对象叫Vue实例化对象，这个对象就是MVVM模式中的VM层，模型通过它可以将数据绑定到页面上，视图可以通过它将数据映射到模型上

ViewModel 是由前端开发人员组织生成和维护的视图数据层。在这一层，前端开发者对从后端获取的 Model 数据进行转换处理，做二次封装，以生成符合 View 层使用预期的视图数据模型。



### 为什么要使用 MVVM

相似问题：MVVM的优点？

低耦合。视图（View）可以独立于Model变化和修改，

可重用性。你可以把一些视图逻辑放在一个ViewModel里面，让很多view重用这段视图逻辑

前后端分离，开发人员可以专注于业务逻辑（ViewModel）和数据的开发，设计人员可以专注于页面设计

## 虚拟DOM(virtual DOM)

相似问题：

### 什么是虚拟dom？

### 什么是render渲染函数

### 虚拟dom实现的原理

除了字符串模板，还可以使用render渲染函数来创建HTML，它更接近编译器

**vue在使用render渲染函数在内存中生成与真实DOM相对应的数据结构,这个在内存中生成的结构称之为虚拟DOM.**

render函数语法格式：render(createElement){ return createElement(标签名,[数据对象],子元素) }

createElement()函数返回一个对象， 就是虚拟DOM => vnode

示例1：

<**div class="box"**>  
  
</**div**>  
  
<**script**>  
 **new** Vue({  
 **el**: **'.box'**,  
 **data**: {  
 **msg**: **'hello vue'** },  
 render: **function**(createElement) {  
 **return** createElement(**'h1'**, **'一则头条'**)  
 },  
 })  
</**script**>

示例2：

**' <style>** .bg {  
 **background**: #abcdef;  
}  
</style>  
<div **id**=**"dome"**>  
 <div>{{message}}</div>  
</div>  
<script>  
**new** Vue({  
 **el**: **"#dome"**,  
 **data**: {  
 **message**: **"hello world"** },  
 render(createElement) {  
 **return** createElement(  
 **"ul"**,  
 {  
 **class**: { **bg**: **true** },  
 **style**: { **color**: **'#fff'**, **textDecoration**:**"underline"**},  
 **on**: {  
 click: **function**(){  
 *alert*(1)  
 }  
 }  
 },  
 [  
 createElement(**"li"**, **'苹果'**),  
 createElement(**"li"**, **'香蕉'**),  
 createElement(**"li"**, **this**.**message**),  
 ]  
 );  
 }  
});  
</script>'

### 虚拟dom的优点？

大量的操作DOM会影响性能，在更新数据后会重新渲染页面，这样造成在没有改变数据的地方也重新渲染DOM节点，这样就造成了很大程度上的资源浪费.

当数据发生变化时,能够智能地计算出重新渲染组件的最小代价并应用到DOM操作上.

## 

## 数据双向绑定

相似问题

### 谈一谈vue的数据双向绑定？

### vue双向数据绑定实现的原理？

vue实现数据双向绑定主要是：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，触发相应监听回调。

**当把一个普通 Javascript 对象传给 Vue 实例来作为它的 data 选项时，Vue 将遍历它的属性，用 Object.defineProperty() 将它们转为 getter/setter。用户看不到 getter/setter，但是在内部它们让 Vue 追踪依赖，在属性被访问和修改时通知变化。**

vue的数据双向绑定 将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的数据变化，通过Compile来解析编译模板指令（vue中是用来解析 {{}}），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变更双向绑定效果。

### Object.defineProperty()

Object.defineProperty() 方法用于在一个对象上定义一个新属性，或者修改一个对象的现有属性，并返回此对象。注意：只能在Object 构造器对象上调用此方法，而不能在任意一个 Object 类型的实例上调用。

Object.defineProperty(obj, prop, descriptor)  
参数  
obj：目标对象，就是要在这个对象上面定义属性  
prop：要定义或修改的属性的名称或 Symbol 。  
descriptor：要定义或修改的属性描述符。是一个对象

通过赋值操作添加的普通属性是可枚举的，（for...in 或 Object.keys 方法），既可以改变这些属性的值，也可以删除这些属性。

Object.defineProperty()方法可以通过对属性的配置，实现更精确的控制

关于descriptor：对象里目前存在的属性描述符有两种主要形式：数据描述符和存取描述符。  
**数据描述符**是一个具有值的属性，该值可以是可写的，也可以是不可写的。  
**存取描述符是**由 getter 函数和 setter 函数所描述的属性。  
一个描述符只能是这两者其中之一；不能同时是两者。

数据描述符和存取描述符都是对象。它们共享以下可选键值（默认值是指在使用 Object.defineProperty() 定义属性时的默认值）：

**configurable**是否可以删除目标属性：当且仅当该属性的 configurable 键值为 true 时，该属性的描述符才能够被改变，同时该属性也能从对应的**对象上被删除。**（使用delete删除）  
默认为 false。

**enumerable**当且仅当该属性的 enumerable 键值为 true 时，该属性才会出现在**对象的枚举属性中。**  
默认为 false。

-------------------------------

数据描述符还具有以下可选键值：  
value  
该属性对应的值。可以是任何有效的 JavaScript 值（数值，对象，函数等）。  
默认为 undefined。

writable  
该属性的值是否可以被重写：当且仅当该属性的 writable 键值为 true 时，属性的值，也就是上面的 value，才能被赋值运算符改变。  
默认为 false。

示例：

<**script**>  
 **let *obj*** = {  
 **a**:1  
 }  
 ***Object***.defineProperty(***obj***,**'b'**,{  
 **configurable**:**true**,  
 **value**: 2,  
 **writable**:**true**,  
 **enumerable**:**true** })  
  
 *// 添加的b属性的描述符writeable为true才可以修改属性b的值* ***obj***.**b** = 3  
 ***console***.log(***obj***.**b**);  
  
 *// 添加的b属性的描述符configurable:true,才可以删除属性b的值* **delete *obj***.**b**;  
 ***console***.log(***obj***);  
  
 *// 添加的b属性的描述符enumerable为true才可以在枚举中出现b属性* **for** (**const *key* in *obj***) {  
 ***console***.log(***key***);  
 }  
</**script**>

---------------------------

存取描述符还具有以下可选键值  
get 属性，是一个函数，是一种获得属性值的方法  
如果没有 getter，则为 undefined。当访问该属性时，会调用此函数。执行时不传入任何参数，但是会传入 this 对象（由于继承关系，这里的this并不一定是定义该属性的对象）。该函数的返回值会被用作属性的值。  
默认为 undefined。

set 属性，是一个函数，是一种设置属性值的方法  
如果没有 setter，则为 undefined。当属性值被修改时，会调用此函数。该方法接受一个参数（也就是被赋予的新值），会传入赋值时的 this 对象。  
默认为 undefined。

示例：

<**script**>  
  
 *// 在对象中添加一个设置了存取描述符属性的示例* **var *bValue*** = 38;  
 ***Object***.defineProperty(o, **"b"**, {  
 *// 使用了方法名称缩写（ES2015 特性）  
 // 下面两个缩写等价于：  
 // get : function() { return bValue; },  
 // set : function(newValue) { bValue = newValue; },* get() {  
 ***console***.log(**'有人访问数据了'**);  
 **return *bValue***;  
 },  
 set(newValue) {  
 ***console***.log(**'设置了新值'**);  
 ***bValue*** = newValue;  
 },  
 **enumerable**: **true**,  
 **configurable**: **true** });  
  
 o.**b**; *// 38  
 // 对象 o 拥有了属性 b，值为 38  
 // 现在，除非重新定义 o.b，o.b 的值总是与 bValue 相同  
  
 // 数据描述符和存取描述符不能混合使用* ***Object***.defineProperty(o, **"conflict"**, {  
 **value**: 0x9f91102,  
 get() { **return** 0xdeadbeef; }  
 });  
 *// 抛出错误 TypeError: value appears only in data descriptors, get appears only in accessor descriptors*</**script**>

说明：  
拥有布尔值的键 configurable、enumerable 和 writable 的默认值都是 false。  
属性值和函数的键 value、get 和 set 字段的默认值为 undefined。

实现vue的双向数据绑定：



## v- if和v- show的区别

v-show指令通过为元素设置CSS的style属性控制隐藏和显示  
v-if是DOM节点操作，当条件成立的时候会将元素加上，不成立的时候，就会移除dom节点。

v-if 频繁切换消耗高，如果在运行时条件不大可能改变 使用v‐if 较好  
v-show 指令初始渲染消耗高，适合频繁切换

## vue八个生命周期及相应生命周期都做了什么

<https://blog.csdn.net/qq_33712668/article/details/90374687>

## scoped的作用

实现组件的私有化，不对全局造成样式污染，表示当前style属性只属于当前模块

## 父子组件件传参

可以使用props选项来声明需要从父级接收的数据，props的值可以是两种：字符串数组，或对象。

子组件里面的元素通过 $emit()来触发自定义事件，父组件可以直接在子组件的标签上使用 v-on 来监听子组件触发的自定义事件

兄弟组件传值

https://baijiahao.baidu.com/s?id=1666564542300874209&wfr=spider&for=pc

## data为什么推荐用函数方式return

之前在Vue的根实例中定义数据对象时， data是一个对象：data：{count: 0}

而在注册组件时， data选项是一个函数(也必须是一个函数)

这是因为组件是可复用的Vue实例，如果还是允许使用先前根实例的数据定义方式，那么所有复用的组件实例都将共享同一份数据，显然，这很容易导致混乱。采用函数定义方式，每复用一次组件， data函数就执行一次， 从而返回一个新的数据对象，那么每个组件实例都将拥有自己的一份返回对象的独立拷贝，

## 说一下你用过的几个UI(框架)库的区别

bootstrap: 响应式（PC,平板，移动三合一）；缺点：性能，代码冗余

layui： 相比较bootstrap来说，组件更丰富，兼容比较旧的浏览器（IE）

Elementui: 基于 Vue 的桌面端组件库，快速开发

Vant：移动端UI库，分为vue版和小程序版

## JS框架和库

js框架：angular, vue.js, react

js库：jQuery , prototype.js

## 用户登陆的问题和授权的问题

## vue的数据监听对数组和对象起作用吗(对哪些数组方法起作用，为什么？）

## 计算属性和方法的区别

计算属性是基于它们的响应式依赖进行缓存的。只在相关响应式依赖发生改变时它们才会重新求值。

相比之下，每当触发重新渲染时，方法将总会再次执行函数。

如果你不希望有缓存，请用方法来替代。

## 计算属性和过滤器的区别



## 计算属性和侦听器的区别

共同点，都是根据依赖变化从而发生变化

不同点：   
计算属性中，返回的是一个data中没有的值（新值）并且必须包含return，  
侦听器中，变化的值是data中存在的值，并且不包含return ；在侦听器中可以执行异步操作，并控制操作的频率，这些都是计算属性无法做到的