🐮化身面试官出30+Vue面试题，超级干货（附答案）｜牛气冲天新年征文

了解过（用过）react或者angular吗，他们有什么区别？

▼ 答案

Vue 借鉴了 angular 的模板和数据绑定技术，又借鉴了 react 的组件化和虚拟 DOM 技术。

😶对 Vue 比较熟一些是吧~（这里只说 Vue 假设你就只熟练 Vue ）

那首先谈谈你对Vue的理解吧？

▼ 答案

官网介绍： cn.vuejs.org/index.html

关键点： 渐进式 JavaScript 框架、核心库加插件、动态创建用户界面（异步获取后台数据，数据展示在界面）

特点： MVVM 模式；代码简洁体积小，运行效率高，适合移动PC端开发；本身只关注 UI （和 react 相似），可以轻松引入 Vue 插件或其他的第三方库进行开发。

🌸思考一下自己所说的那些点，自己都非常清楚明白吗？

下面呢我就根据你对 vue 的理解，接着谈谈：

你刚刚说到了MVVM，能详细说说吗？

▼ 答案

全称： Model-View-ViewModel ， Model 表示数据模型层。 view 表示视图层， ViewModel 是 View 和 Model 层的桥梁，数据绑定到 viewModel 层并

自动渲染到页面中，视图变化通知 `viewModel` 层更新数据。

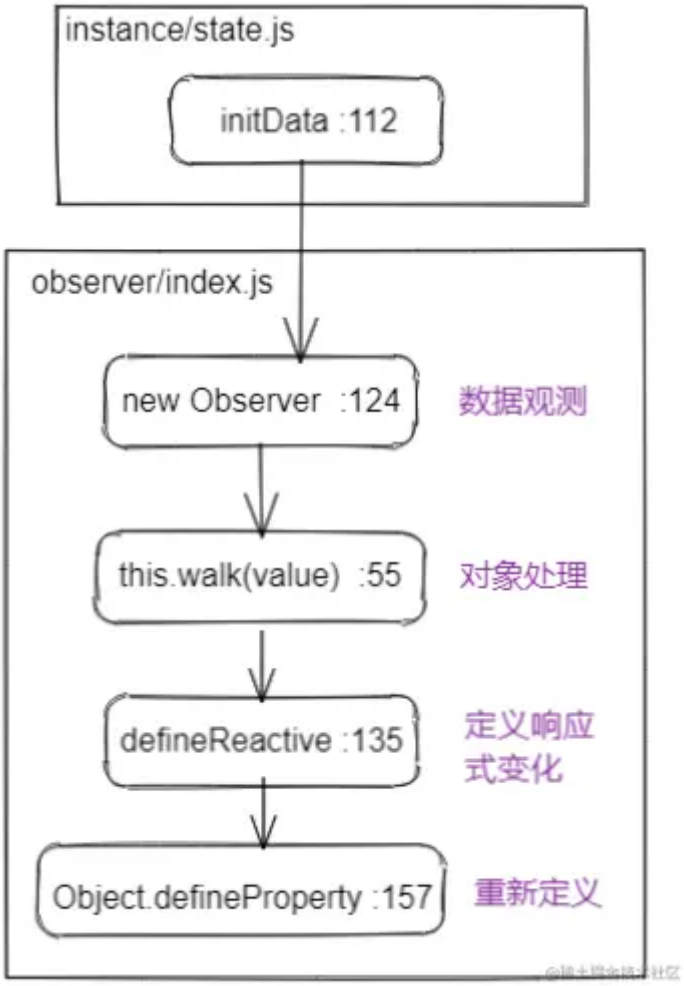
😶摸底差不多，问基础了，响应式数据得知道吧，问一问

vue是如何实现响应式数据的呢？（响应式数据原理）❗

▼ 答案

Vue2： `Object.defineProperty` 重新定义 `data` 中所有的属性，`Object.defineProperty` 可以使数据的获取与设置增加一个拦截的功能，拦截属性的获取，进行依赖收集。拦截属性的更新操作，进行通知。

具体的过程：首先Vue使用 `initData` 初始化用户传入的参数，然后使用 `new Observer` 对数据进行观测，如果数据是一个对象类型就会调用 `this.walk（value）` 对对象进行处理，内部使用 `defineeReactive` 循环对象属性定义响应式变化，核心就是使用 `Object.defineProperty` 重新定义数据。



🌸刚刚如果你说了对象的检测，然后又没说清楚数组的处理的话，我就会问下面这个问题

那vue中是如何检测数组变化的呢?

▼ 答案

数组就是使用 `object.defineProperty` 重新定义数组的每一项，那能引起数组变化的方法我们都是知道的， `pop` 、 `push` 、 `shift` 、 `unshift` 、 `splice` 、 `sort` 、 `reverse` 这七种，只要这些方法执行改了数组内容，我就更新内容就好了，是不是很好理解。

1. 是用来函数劫持的方式，重写了数组方法，具体呢就是更改了数组的原型，更改成自己的，用户调数组的一些方法的时候，走的就是自己的方法，然后通知视图去更新。

2. 数组里每一项可能是对象，那么我就是会对数组的每一项进行观测，（且只有数组里的对象才能进行观测，观测过的也不会进行观测）

vue3：改用 `proxy` ，可直接监听对象数组的变化。

那你说说Vue的事件绑定原理吧

▼ 答案

- 原生 `DOM` 的绑定：Vue在创建真实DOM时会调用 `createElm` ，默认会调用 `invokeCreateHooks` 。会遍历当前平台下相对的属性处理代码，其中就有 `updateDOMListeners` 方法，内部会传入 `add（）` 方法

- 组件绑定事件，原生事件，自定义事件；组件绑定之间是通过Vue中自定义的 `$on` 方法实现的。 （可以理解为：组件的 `nativeOnOn` 等价于 普通元素on 组件的on会单独处理）

v-model中的实现原理及如何自定义v-model ❗

▼ 答案

`v-model` 可以看成是 `value+input` 方法的语法糖（组件）。原生的 `v-model`，会根据标签的不同生成不同的事件与属性。解析一个指令来。

自定义：自己写 `model` 属性，里面放上 `prop` 和 `event`

👍还行哟~知道响应式数据和数据绑定问完了，接着问问渲染呗：

为什么Vue采用异步渲染呢?

▼ 答案

`Vue` 是组件级更新，如果不采用异步更新，那么每次更新数据都会对当前组件进行重新渲染，所以为了性能， `Vue` 会在本轮数据更新后，在异步更新视图。核心思想 `nextTick` 。

`dep.notify()` 通知 watcher进行更新， `subs[i].update` 依次调用 watcher 的 `update` ， `queueWatcher` 将watcher 去重放入队列， nextTick（ `flushSchedulerQueue` ）在下一tick中刷新watcher队列（异步）。

🌸接着追问，要是你nextTick都能讲得很清楚的话那基本你是明白了。

了解nextTick吗?

▼ 答案

异步方法，异步渲染最后一步，与JS事件循环联系紧密。主要使用了宏任务微任务（ `setTimeout` 、 `promise` 那些），定义了一个异步方法，多次调用 `nextTick` 会将方法存入队列，通过异步方法清空当前队列。

可以的可以的，先问你个生命周期，我再想想怎么难住你😄

说说Vue的生命周期吧❗

▼ 答案

什么时候被调用?

- beforeCreate ：实例初始化之后，数据观测之前调用

- created：实例创建万之后调用。实例完成：数据观测、属性和方法的运算、`watch/event` 事件回调。无 `$el` .
- beforeMount：在挂载之前调用，相关 `render` 函数首次被调用
- mounted：了被新创建的 `vm.$el` 替换，并挂载到实例上去之后调用改钩子。
- beforeUpdate：数据更新前调用，发生在虚拟DOM重新渲染和打补丁，在这之后会调用改钩子。
- updated：由于数据更改导致的虚拟DOM重新渲染和打补丁，在这之后会调用改钩子。
- beforeDestroy：实例销毁前调用，实例仍然可用。
- destroyed：实例销毁之后调用，调用后，Vue实例指示的所有东西都会解绑，所有事件监听器和所有子实例都会被移除

每个生命周期内部可以做什么?

- created：实例已经创建完成，因为他是最早触发的，所以可以进行一些数据、资源的请求。
- mounted：实例已经挂载完成，可以进行一些DOM操作。
- beforeUpdate：可以在这个钩子中进一步的更改状态，不会触发重渲染。
- updated：可以执行依赖于DOM的操作，但是要避免更改状态，可能会导致更新无线循环。
- destroyed：可以执行一些优化操作，清空计时器，解除绑定事件。

ajax放在哪个生命周期?：一般放在 `mounted` 中，保证逻辑统一性，因为生命周期是同步执行的，`ajax` 是异步执行的。单数服务端渲染 `ssr` 同一放在 `created` 中，因为服务端渲染不支持 `mounted` 方法。 **什么时候使用beforeDestroy?**：当前页面使用 `$on` ，需要解绑事件。清楚定时器。解除事件绑定，`scroll mousemove` 。

父子组件生命周期调用顺序（简单）

▼ 答案

渲染顺序：先父后子，完成顺序：先子后父

更新顺序：父更新导致子更新，子更新完成后父

销毁顺序：先父后子，完成顺序：先子后父

Vue组件通信 ❗

▼ 答案

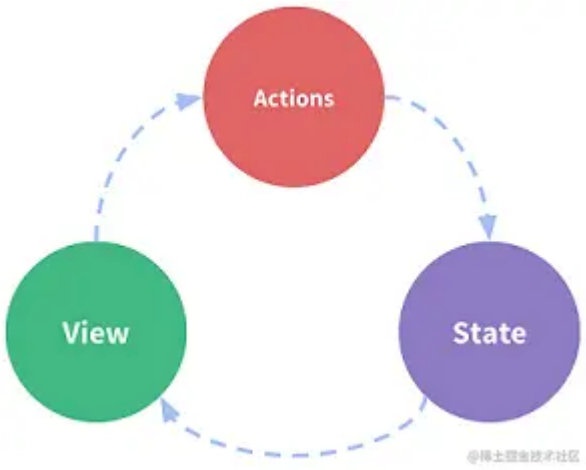
- 父子间通信:父亲提供数据通过属性 `props` 传给儿子；儿子通过 `$on` 绑父亲的事件，再通过 `$emit` 触发自己的事件（发布订阅）
- 利用父子关系 `$parent` 、 `$children` ， 获取父子组件实例的方法。
- 父组件提供数据，子组件注入。 `provide` 、 `inject` ，插件用得多。
- `ref` 获取组件实例，调用组件的属性、方法
- 跨组件通信 `Event Bus` （Vue.prototype.$bus = new Vue）其实基于on与$emit
- `vuex` 状态管理实现通信

Vuex 工作原理

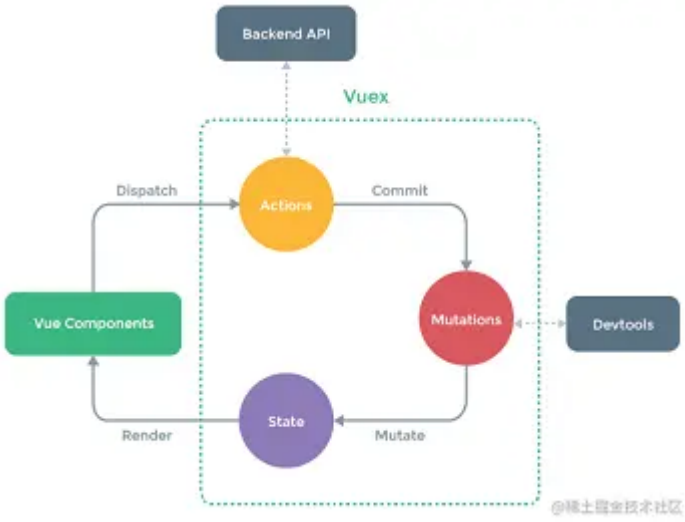
▼ 答案

官网：vuex.vuejs.org/zh/

- Vuex 是一个专为 Vue.js 应用程序开发的**状态管理模式。**
- 状态自管理应用包含以下几个部分：
  - state，驱动应用的数据源；
  - view，以声明方式将 state 映射到视图；
  - actions，响应在 view 上的用户输入导致的状态变化。下图单向数据流示意图：



- vuex，多组件共享状态，因-单向数据流简洁性很容易被破坏：
  - 多个视图依赖于同一状态。
  - 来自不同视图的行为需要变更同一状态。



问虚拟 `DOM` 吧，看你能不能讲清楚从真实 `DOM` 到虚拟 `DOM` ，再和我说说 `diff`

如何从真实DOM到虚拟DOM

▼ 答案

涉及到Vue中的模板编译原理，主要过程：

1. 将模板转换成 `ast` 树， `ast` 用对象来描述真实的JS语法（将真实DOM转换成虚拟DOM）
2. 优化树
3. 将 `ast` 树生成代码

用VNode来描述一个DOM结构

▼ 答案

虚拟节点就是用一个对象来描述一个真实的DOM元素。首先将 `template` （真实DOM）先转成 `ast` ， `ast` 树通过 `codegen` 生成 `render` 函数， `render` 函数里的 `_c` 方法将它转为虚拟dom

diff算法

▼ 答案

时间复杂度： 个树的完全 `diff` 算法是一个时间复杂度为 `O(n*3)` ，vue进行优化转化成 `O(n)` 。

理解：

- 最小量更新， `key` 很重要。这个可以是这个节点的唯一标识，告诉 `diff` 算法，在更改前后它们是同一个DOM节点
    - 扩展 `v-for` 为什么要有 `key` ，没有 `key` 会暴力复用，举例子的话随便说一个比如移动节点或者增加节点（修改DOM），加 `key` 只会移动减少操作DOM。
- 只有是同一个虚拟节点才会进行精细化比较，否则就是暴力删除旧的，插入新的。
- 只进行同层比较，不会进行跨层比较。

diff算法的优化策略：四种命中查找，四个指针

1. 旧前与新前（先比开头，后插入和删除节点的这种情况）
2. 旧后与新后（比结尾，前插入或删除的情况）
3. 旧前与新后（头与尾比，此种发生了，涉及移动节点，那么新前指向的节点，移动到旧后之后）

4. 旧后与新前（尾与头比，此种发生了，涉及移动节点，那么新前指向的节点，移动到旧前之前）

--- 问完上面这些如果都能很清楚的话，基本O了 ---

以下的这些简单的概念，你肯定也是没有问题的啦😉

Computed watch 和 method

▼ 答案

computed：默认 `computed` 也是一个 `watcher` 具备缓存，只有当依赖的数据变化时才会计算，当数据没有变化时，它会读取缓存数据。如果一个数据依赖于其他数据，使用 `computed`

watch：每次都需要执行函数。 `watch` 更适用于数据变化时的异步操作。如果需要在某个数据变化时做一些事情，使用watch。

method：只要把方法用到模板上了,每次一变化就会重新渲染视图，性能开销大

v-if 和 v-show 区别

▼ 答案

- `v-if` 如果条件不成立不会渲染当前指令所在节点的DOM元素
- `v-show` 只是切换当前DOM的显示与隐藏

v-for和v-if为什么不能连用

▼ 答案

`v-for` 会比 `v-if` 的优先级更高，连用的话会把 `v-if` 的每个元素都添加一下，造成性能问题。

v-html 会导致哪些问题（简单）

▼ 答案

- `XSS` 攻击
- `v-html` 会替换标签内部的元素

描述组件渲染和更新过程

▼ 答案

渲染组件时，会通过 `vue.extend()` 方法构建子组件的构造函数，并进行实例化。最终手动调用 `$mount()` 进行挂载。更新组件时会进行 `patchVnode` 流程，核心就是 `diff` 算法。

组件中的data为什么是函数

▼ 答案

避免组件中的数据互相影响。同一个组件被复用多次会创建多个实例，如果 `data` 是一个对象的话，这些实例用的是同一个构造函数。为了保证组件的数据独立，要求每个组件都必须通过 `data` 函数返回一个对象作为组件的状态。

为什么要使用异步组件?

▼ 答案

1. 节省打包出的结果，异步组件分开打包，采用jsonp的方式进行加载，有效解决文件过大的问题。
2. 核心就是包组件定义变成一个函数，依赖 `import（）` 语法，可以实现文件的分割加载。 详细的看官方文档：cn.vuejs.org/v2/guide/co...

action 与 mutation 的区别

▼ 答案

- `mutation` 是同步更新， `$watch` 严格模式下会报错
- `action` 是异步操作，可以获取数据后调用 `mutation` 提交最终数据

插槽与作用域插槽的区别

插槽

▼ 答案

- 创建组件虚拟节点时，会将组件儿子的虚拟节点保存起来。当初始化组件时，通过插槽属性将儿子进行分类 `{a:[vnode],b[vnode]}`
- 渲染组件时会拿对应的 `slot` 属性的节点进行替换操作。（插槽的作用域为父组件）

作用域插槽

▼ 答案

- 作用域插槽在解析的时候不会作为组件的孩子节点。会解析成函数，当子组件渲染时，会调用此函数进行渲染。
- 普通插槽渲染的作用域是父组件，作用域插槽的渲染作用域是当前子组件。

vue中相同逻辑如何抽离

▼ 答案

其实就是考察 `vue.mixin` 用法，给组件每个生命周期，函数都混入一些公共逻辑。

谈谈对keep-alive的了解

▼ 答案

`keep-alive` 可以实现组件的缓存，当组件切换时不会对当前组件进行卸载。常用的2个属性 `include/exclude` ，2个生命周期 `activated` ， `deactivated`

Vue性能优化

▼ 答案

编码优化:

- 事件代理
- `keep-alive`
- 拆分组件
- `key` 保证唯一性
- 路由懒加载、异步组件
- 防抖节流

Vue加载性能优化

- 第三方模块按需导入（ `babel-plugin-component` ）
- 图片懒加载

用户体验

- `app-skeleton` 骨架屏
- `shellap` p壳
- `pwa`

SEO优化

- 预渲染