Vue 面试专题

简述MVVM

什么是MVVM?

视图模型双向绑定，是Model-View-ViewModel的缩写，也就是把MVC中的Controller演变成ViewModel。Model层代表数据模型，View代表UI组件，ViewModel是View和Model层的桥梁，数据会绑定到viewModel层并自动将数据渲染到页面中，视图变化的时候会通知viewModel层更新数据。以前是操作DOM结构更新视图，现在是数据驱动视图。

MVVM的优点:

1.低耦合。视图（View）可以独立于Model变化和修改，一个Model可以绑定到不同的View上，当View变化的时候Model可以不变化，当Model变化的时候View也可以不变;
2.可重用性。你可以把一些视图逻辑放在一个Model里面，让很多View重用这段视图逻辑。
3.独立开发。开发人员可以专注于业务逻辑和数据的开发(ViewModel)，设计人员可以专注于页面设计。
4.可测试。

Vue底层实现原理

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和getter，在数据变动时发布消息给订阅者，触发相应的监听回调
Vue是一个典型的MVVM框架，模型（Model）只是普通的javascript对象，修改它则试图（View）会自动更新。这种设计让状态管理变得非常简单而直观

Observer（数据监听器）：Observer的核心是通过Object.defineProprtty()来监听数据的变动，这个函数内部可以定义setter和getter，每当数据发生变化，就会触发setter。这时候Observer就要通知订阅者，订阅者就是Watcher

Watcher（订阅者）：Watcher订阅者作为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

Compile（指令解析器）：Compile主要做的事情是解析模板指令，将模板中变量替换成数据，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，添加鉴定数据的订阅者，一旦数据有变动，收到通知，更新试图

传送门：☞ 20分钟吃透Diff算法核心原理

谈谈对vue生命周期的理解?

每个Vue实例在创建时都会经过一系列的初始化过程，vue的生命周期钩子，就是说在达到某一阶段或条件时去触发的函数，目的就是为了完成一些动作或者事件

- create阶段：vue实例被创建
  beforeCreate: 最初调用触发，创建前，此时data和methods中的数据都还没有初始化，data和events都不能用

created： 创建完毕，data中有值，未挂载，data和events已经初始化好，data已经具有响应式；在这里可以发送请求

- mount阶段： vue实例被挂载到真实DOM节点
  beforeMount：在模版编译之后，渲染之前触发，可以发起服务端请求，去数据，ssr中不可用，基本用不上这个hook
  mounted: 在渲染之后触发，此时可以操作DOM，并能访问组件中的DOM以及$ref,SSR中不可用
- update阶段：当vue实例里面的data数据变化时，触发组件的重新渲染
  beforeUpdate :更新前，在数据变化后，模版改变前触发，切勿使用它监听数据变化
  updated：更新后，在数据改变后，模版改变后触发，常用于重渲染案后的打点，性能检测或触发vue组件中非vue组件的更新
- destroy阶段：vue实例被销毁
  beforeDestroy：实例被销毁前，组件卸载前触发，此时可以手动销毁一些方法，可以在此时清理事件、计时器或者取消订阅操作
  destroyed:卸载完毕后触发，销毁后，可以做最后的打点或事件触发操作

组件生命周期

生命周期（父子组件） 父组件beforeCreate --> 父组件created --> 父组件beforeMount --> 子组件beforeCreate --> 子组件created --> 子组件beforeMount --> 子组件 mounted --> 父组件mounted -->父组件beforeUpdate -->子组件beforeDestroy--> 子组件destroyed --> 父组件updated

加载渲染过程 父beforeCreate->父created->父beforeMount->子beforeCreate->子created->子beforeMount->子mounted->父mounted

挂载阶段 父created->子created->子mounted->父mounted

父组件更新阶段 父beforeUpdate->父updated

子组件更新阶段 父beforeUpdate->子beforeUpdate->子updated->父updated

销毁阶段 父beforeDestroy->子beforeDestroy->子destroyed->父destroyed

computed与watch

通俗来讲，既能用 computed 实现又可以用 watch 监听来实现的功能，推荐用 computed， 重点在于 computed 的缓存功能 computed 计算属性是用来声明式的描述一个值依赖了其它的值，当所依赖的值或者变量 改变时，计算属性也会跟着改变； watch 监听的是已经在 data 中定义的变量，当该变量变化时，会触发 watch 中的方法。

watch 属性监听 是一个对象，键是需要观察的属性，值是对应回调函数，主要用来监听某些特定数据的变化，从而进行某些具体的业务逻辑操作,监听属性的变化，需要在数据变化时执行异步或开销较大的操作时使用

computed 计算属性 属性的结果会被缓存，当computed中的函数所依赖的属性没有发生改变的时候，那么调用当前函数的时候结果会从缓存中读取。除非依赖的响应式属性变化时才会重新计算，主要当做属性来使用 computed中的函数必须用return返回最终的结果 computed更高效，优先使用。data 不改变，computed 不更新。

使用场景 computed：当一个属性受多个属性影响的时候使用，例：购物车商品结算功能 watch：当一条数据影响多条数据的时候使用，例：搜索数据

组件中的data为什么是一个函数?

1.一个组件被复用多次的话，也就会创建多个实例。本质上，这些实例用的都是同一个构造函数。 2.如果data是对象的话，对象属于引用类型，会影响到所有的实例。所以为了保证组件不同的实例之间data不冲突，data必须是一个函数。

为什么v-for和v-if不建议用在一起

1.当 v-for 和 v-if 处于同一个节点时，v-for 的优先级比 v-if 更高，这意味着 v-if 将分别重复运行于每个 v-for 循环中。如果要遍历的数组很大，而真正要展示的数据很少时，这将造成很大的性能浪费 2.这种场景建议使用computed，先对数据进行过滤

注意：3.x 版本中 `v-if` 总是优先于 `v-for` 生效。由于语法上存在歧义，建议避免在同一元素上同时使用两者。比起在模板层面管理相关逻辑，更好的办法是通过创建计算属性筛选出列表，并以此创建可见元素。

解惑传送门 ☞ # v-if 与 v-for 的优先级对比非兼容

React/Vue 项目中 key 的作用

- key的作用是为了在diff算法执行时更快的找到对应的节点，提高diff速度，更高效的更新虚拟DOM;

  vue和react都是采用diff算法来对比新旧虚拟节点，从而更新节点。在vue的diff函数中，会根据新节点的key去对比旧节点数组中的key，从而找到相应旧节点。如果没找到就认为是一个新增节点。而如果没有key，那么就会采用遍历查找的方式去找到对应的旧节点。一种一个map映射，另一种是遍历查找。相比而言。map映射的速度更快。

- 为了在数据变化时强制更新组件，以避免“就地复用”带来的副作用。

  当 Vue.js 用 `v-for` 更新已渲染过的元素列表时，它默认用“就地复用”策略。如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序，而是简单复用此处每个元素，并且确保它在特定索引下显示已被渲染过的每个元素。重复的key会造成渲染错误。

数组扁平化转换

在说到模版编译的时候，有可能会提到数组的转换，一般就用递归处理 将 [1,2,3,[4,5]] 转换成

```
{
    children:[
        {
            value:1
        },
        {
            value:2
        },
        {
            value:3
        },
        {
            children:[
                {
                    value:4
```

```
                },
                {
                    value:5
                }
            ]
        },
    ]
}
```

```
// 测试数组
var arr =[1,2, 3, [4,5]];
// 转换函数
function convert(arr){
    //准备一个接收结果数组
    var result = [];
    // 遍历传入的 arr 的每一项
    for(let i=0;i<arr.length;i++){
        //如果遍历到的数字是number，直接放进入
        if(typeof arr[i] == 'number'){
            result.push({
                value:arr[i]
            });
        } else if(Array.isArray(arr[i])){
            //如果遍历到这个项目是数组，那么就递归
            result.push({
                children: convert(arr[i])
            });
        }
    }
    return result;
}

var o = convert(arr);
console.log(o);
```

vue组件的通信方式

- `props`/`$emit` 父子组件通信

  父->子`props`，子->父 `$on`、`$emit` 获取父子组件实例 `parent`、`children` `Ref` 获取实例的方式调用组件的属性或者方法 父->子孙 `Provide`、`inject` 官方不推荐使用，但是写组件库时很常用

- `$emit`/`$on` 自定义事件 兄弟组件通信

  `Event Bus` 实现跨组件通信 `Vue.prototype.$bus = new Vue()` 自定义事件

- vuex 跨级组件通信

  Vuex、`$attrs`、`$listeners` `Provide`、`inject`

$emit 后面的两个参数是什么

1、父组件可以使用 props 把数据传给子组件。 2、子组件可以使用 $emit,让父组件监听到自定义事件 。

`vm.$emit( event, arg );`//触发当前实例上的事件，要传递的参数 `vm.$on( event, fn );`//监听event事件后运行 fn；

子组件

```
<template>
  <div class="train-city">
    <h3>父组件传给子组件的toCity:{{sendData}}</h3>
    <br/><button @click='select(`大连`)'>点击此处将‘大连’发射给父组件</button>
  </div>
</template>
<script>
  export default {
    name:'trainCity',
    props:['sendData'], // 用来接收父组件传给子组件的数据
    methods:{
      select(val) {
        let data = {
          cityName: val
        };
        this.$emit('showCityName',data);//select事件触发后，自动触发showCityName事件
      }
    }
  }
</script>
```

父组件

```
<template>
    <div>
        <div>父组件的toCity{{toCity}}</div>
        <train-city @showCityName="updateCity" :sendData="toCity"></train-city>
    </div>
<template>
<script>
  export default {
    name:'index',
    components: {},
    data () {
      return {
        toCity:"北京"
      }
    },
    methods:{
      updateCity(data){//触发子组件城市选择-选择城市的事件
        this.toCity = data.cityName;//改变了父组件的值
```

```
          console.log('toCity:'+this.toCity)
        }
      }
    }
  </script>
```

nextTick的实现

1. nextTick是Vue提供的一个全局API,是在下次DOM更新循环结束之后执行延迟回调，在修改数据之后使用$nextTick，则可以在回调中获取更新后的DOM；
2. Vue在更新DOM时是异步执行的。只要侦听到数据变化，Vue将开启1个队列，并缓冲在同一事件循环中发生的所有数据变更。如果同一个watcher被多次触发，只会被推入到队列中-次。这种在缓冲时去除重复数据对于避免不必要的计算和DOM操作是非常重要的。nextTick方法会在队列中加入一个回调函数，确保该函数在前面的dom操作完成后才调用；
3. 比如，我在干什么的时候就会使用nextTick，传一个回调函数进去，在里面执行dom操作即可；
4. 我也有简单了解nextTick实现，它会在callbacks里面加入我们传入的函数，然后用timerFunc异步方式调用它们，首选的异步方式会是Promise。这让我明白了为什么可以在nextTick中看到dom操作结果。

nextTick的实现原理是什么?

在下次 DOM 更新循环结束之后执行延迟回调，在修改数据之后立即使用 nextTick 来获取更新后的 DOM。nextTick主要使用了宏任务和微任务。 根据执行环境分别尝试采用Promise、MutationObserver、setImmediate，如果以上都不行则采用setTimeout定义了一个异步方法，多次调用nextTick会将方法存入队列中，通过这个异步方法清空当前队列。

使用过插槽么？用的是具名插槽还是匿名插槽或作用域插槽

vue中的插槽是一个非常好用的东西slot说白了就是一个占位的 在vue当中插槽包含三种一种是默认插槽（匿名）一种是具名插槽还有一种就是作用域插槽 匿名插槽就是没有名字的只要默认的都填到这里具名插槽指的是具有名字的

keep-alive的实现

keep-alive是Vue.js的一个内置组件。它能够不活动的组件实例保存在内存中，而不是直接将其销毁，它是一个抽象组件，不会被渲染到真实DOM中，也不会出现在父组件链中。

作用：实现组件缓存，保持这些组件的状态，以避免反复渲染导致的性能问题。 需要缓存组件 频繁切换，不需要重复渲染

场景：tabs标签页 后台导航，vue性能优化

原理：Vue.js内部将DOM节点抽象成了一个个的VNode节点，keep-alive组件的缓存也是基于VNode节点的而不是直接存储DOM结构。它将满足条件（pruneCache与pruneCache）的组件在cache对象中缓存起来，在需要重新渲染的时候再将vnode节点从cache对象中取出并渲染。

keep-alive 的属性

它提供了include与exclude两个属性，允许组件有条件地进行缓存。

include定义缓存白名单，keep-alive会缓存命中的组件；exclude定义缓存黑名单，被命中的组件将不会被缓存；max定义缓存组件上限，超出上限使用LRU的策略置换缓存数据。

在动态组件中的应用

```
<keep-alive :include="whiteList" :exclude="blackList" :max="amount">
     <component :is="currentComponent"></component>
</keep-alive>
```

在vue-router中的应用

```
<keep-alive :include="whiteList" :exclude="blackList" :max="amount">
    <router-view></router-view>
</keep-alive>
```

vue 中完整示例

```
<keep-alive>
    <coma v-if="test"></coma>
    <comb v-else="test"></comb>
</keep-alive>
<button @click="test=handleClick">请点击</button>

export default {
    data () {
        return {
            test: true
        }
    },
    methods: {
        handleClick () {
            this.test = !this.test;
        }
    }
}
```

参考: keep-alive 官网

keep-alive实现原理

Vue keep-alive的实现原理

mixin

mixin 项目变得复杂的时候，多个组件间有重复的逻辑就会用到mixin 多个组件有相同的逻辑，抽离出来 mixin 并不是完美的解决方案，会有一些问题 vue3提出的Composition API旨在解决这些问题【追求完美是要消耗一定的成本的，如开发成本】 场景：PC端新闻列表和详情页一样的右侧栏目，可以使用mixin进行混合 劣势：1.

变量来源不明确，不利于阅读 2.多mixin可能会造成命名冲突 3.mixin和组件可能出现多对多的关系，使得项目复杂度变高

vue 如何实现模拟 v-model 指令

可以使用 vue 自定义指令 Vue.directive() 模拟

具体参考：vue自定义指令模拟v-model指令

如何实现 v-model,说下思路

Vue Router 相关

Vuex的理解及使用场景

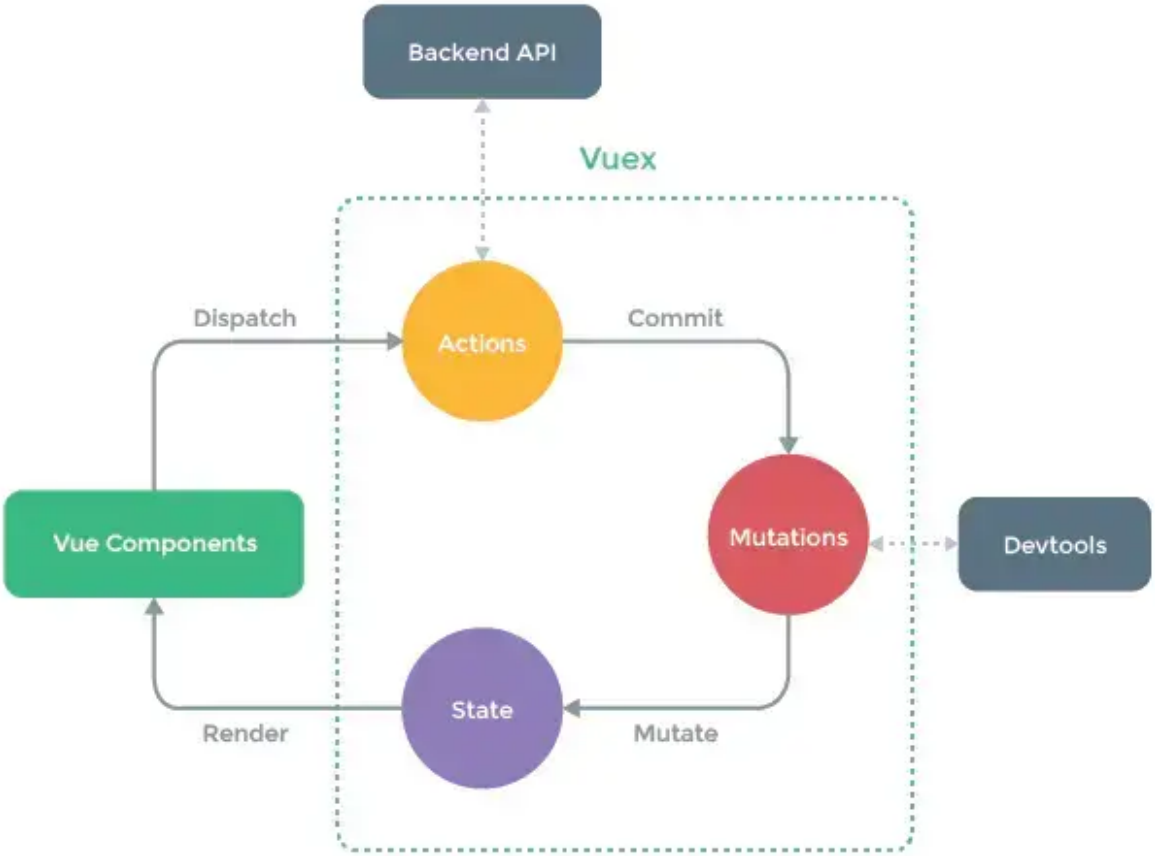
Vuex 是一个专为 Vue 应用程序开发的状态管理模式。每一个 Vuex 应用的核心就是 store（仓库）。

1. 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相应地得到高效更新 2. 改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation， 这样使得我们可以方便地跟踪每一个状态的变化 Vuex主要包括以下几个核心模块：

1. State：定义了应用的状态数据
2. Getter：在 store 中定义“getter”（可以认为是 store 的计算属性），

就像计算属性一样，getter 的返回值会根据它的依赖被缓存起来， 且只有当它的依赖值发生了改变才会被重新计算 3. Mutation：是唯一更改 store 中状态的方法，且必须是同步函数 4. Action：用于提交 mutation，而不是直接变更状态，可以包含任意异步操作 5. Module：允许将单一的 Store 拆分为多个 store 且同时保存在单一的状态树中


Backend API
Vuex
Dispatch
Actions
Commit
Vue Components
Mutations
Devtools
State
Render
Mutate