1. 说一下 Vue 的双向绑定数据的原理

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

2. 解释单向数据流和双向数据绑定

单向数据流： 顾名思义，数据流是单向的。数据流动方向可以跟踪，流动单一，追查问题的时候可以更快捷。缺点就是写起来不太方便。要使 UI 发生变更就必须创建各种 action 来维护对应的 state

双向数据绑定：数据之间是相通的，将数据变更的操作隐藏在框架内部。优点是在表单交互较多的场景下，会简化大量与业务无关的代码。缺点就是无法追踪局部状态的变化，增加了出错时 debug 的难度

3. Vue 如何去除 url 中的

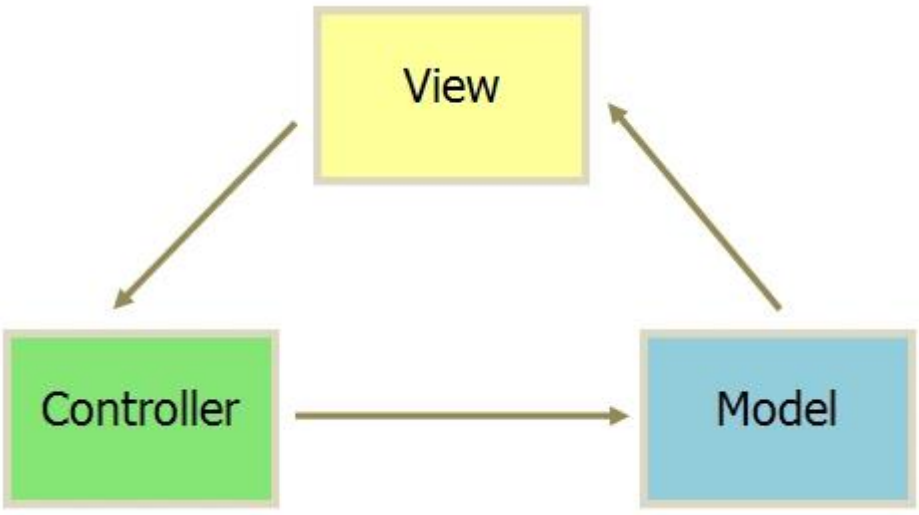
vue-router 默认使用 hash 模式，所以在路由加载的时候，项目中的 url 会自带 #。如果不想使用 #， 可以使用 vue-router 的另一种模式 history

```
new Router({
  mode: 'history',
  routes: [ ]
})
```

需要注意的是，当我们启用 history 模式的时候，由于我们的项目是一个单页面应用，所以在路由跳转的时候，就会出现访问不到静态资源而出现 404 的情况，这时候就需要服务端增加一个覆盖所有情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回同一个 index.html 页面

4. 对 MVC、MVVM 的理解

MVC

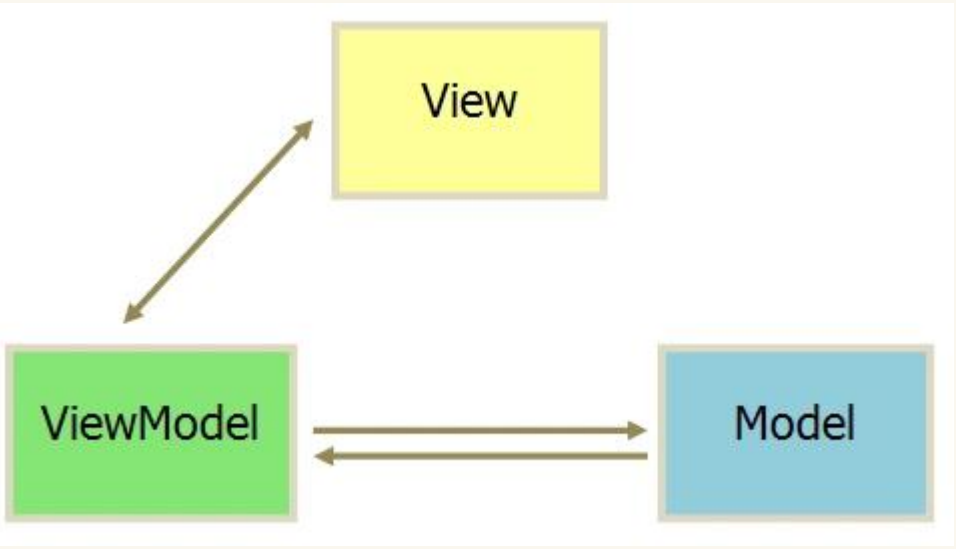


特点：

1. View 传送指令到 Controller
2. Controller 完成业务逻辑后，要求 Model 改变状态
3. Model 将新的数据发送到 View，用户得到反馈

所有通信都是单向的

MVVM



特点：

1. 各部分之间的通信，都是双向的
2. 采用双向绑定：View 的变动，自动反映在 ViewModel，反之亦然

具体请移步 这里

5. 介绍虚拟 DOM

参考这里

6. vue 生命周期的理解

> vue 实例有一个完整的生命周期，生命周期也就是指一个实例从开始创建到销毁的这个过程

beforeCreated() 在实例创建之间执行，数据未加载状态

created() 在实例创建、数据加载后，能初始化数据，dom 渲染之前执行

beforeMount() 虚拟 dom 已创建完成，在数据渲染前最后一次更改数据

mounted() 页面、数据渲染完成，真实 dom 挂载完成

beforeUpadate() 重新渲染之前触发

updated() 数据已经更改完成，dom 也重新 render 完成,更改数据会陷入死循环

beforeDestory() 和 destoryed() 前者是销毁前执行（实例仍然完全可用），后者则是销毁后执行

7. 组件通信

> 父组件向子组件通信

子组件通过 props 属性，绑定父组件数据，实现双方通信

子组件向父组件通信

将父组件的事件在子组件中通过 $emit 触发

非父子组件、兄弟组件之间的数据传递

```
/*新建一个 Vue 实例作为中央事件总线*/let event = new Vue();
/*监听事件*/
event.$on('eventName', (val) => {
    //.....do something
});
/*触发事件*/
event.$emit('eventName', 'this is a message.')
```

Vuex 数据管理

8. vue-router 路由实现

路由就是用来跟后端服务器进行交互的一种方式，通过不同的路径，来请求不同的资源，请求不同的页面是路由的其中一种功能

参考 这里

9. v-if 和 v-show 区别

使用了 v-if 的时候，如果值为 false ，那么页面将不会有这个 html 标签生成。

v-show 则是不管值为 true 还是 false ， html 元素都会存在，只是 CSS 中的 display 显示或隐藏

10. $route 和$router 的区别

$router 为 VueRouter 实例，想要导航到不同 URL，则使用 $router.push 方法

$route 为当前 router 跳转对象里面可以获取 name 、 path 、 query 、 params 等

11. NextTick 是做什么的

$nextTick 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

具体可参考官方文档 深入响应式原理

12. Vue 组件 data 为什么必须是函数

因为 js 本身的特性带来的，如果 data 是一个对象，那么由于对象本身属于引用类型，当我们修改其中的一个属性时，会影响到所有 Vue 实例的数据。如果

将 data 作为一个函数返回一个对象，那么每一个实例的 data 属性都是独立的，不会相互影响了

13. 计算属性 computed 和事件 methods 有什么区别

我们可以将同一函数定义为一个 method 或者一个计算属性。对于最终的结果，两种方式是相同的

不同点：

computed: 计算属性是基于它们的依赖进行缓存的，只有在它的相关依赖发生改变时才会重新求值

对于 method，只要发生重新渲染，method 调用总会执行该函数

14. 对比 jQuery ，Vue 有什么不同

jQuery 专注视图层，通过操作 DOM 去实现页面的一些逻辑渲染； Vue 专注于数据层，通过数据的双向绑定，最终表现在 DOM 层面，减少了 DOM 操作

Vue 使用了组件化思想，使得项目子集职责清晰，提高了开发效率，方便重复利用，便于协同开发

15. Vue 中怎么自定义指令

全局注册

```
// 注册一个全局自定义指令 `v-focus`
Vue.directive('focus', {
  // 当被绑定的元素插入到 DOM 中时……
  inserted: function (el) {
    // 聚焦元素
    el.focus()
  }
})
```

局部注册

```
directives: {
  focus: {
    // 指令的定义
    inserted: function (el) {
      el.focus()
    }
  }
}
```

参考 官方文档-自定义指令

16. Vue 中怎么自定义过滤器

可以用全局方法 Vue.filter() 注册一个自定义过滤器，它接收两个参数：过滤器 ID 和过滤器函数。过滤器函数以值为参数，返回转换后的值

```
Vue.filter('reverse', function (value) {
  return value.split('').reverse().join('')
})
<!-- 'abc' => 'cba' --><span v-text="message | reverse"></span>
```

过滤器也同样接受全局注册和局部注册

17. 对 keep-alive 的了解

keep-alive 是 Vue 内置的一个组件，可以使被包含的组件保留状态，或避免重新渲染

```
<keep-alive>
  <component>
    <!-- 该组件将被缓存！ -->
  </component></keep-alive>
```

可以使用 API 提供的 props，实现组件的动态缓存

具体参考 官方 API

18. Vue 中 key 的作用

> key 的特殊属性主要用在 Vue 的虚拟 DOM 算法，在新旧 nodes 对比时辨识 VNodes。如果不使用 key，Vue 会使用一种最大限度减少动态元素并且尽可能的尝试修复/再利用相同类型元素的算法。使用 key，它会基于 key 的变化重新排列元素顺序，并且会移除 key 不存在的元素。
>
> 有相同父元素的子元素必须有独特的 key。重复的 key 会造成渲染错误

具体参考 官方 API

19. Vue 的核心是什么

> 数据驱动 组件系统

20. vue 等单页面应用的优缺点

> 优点：

- 良好的交互体验
- 良好的前后端工作分离模式
- 减轻服务器压力

> 缺点：

- SEO 难度较高
- 前进、后退管理
- 初次加载耗时多