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。

说一下你对于MVVM 的理解

> MVVM 是 Model-View-ViewModel 的缩写

- Model 代表数据模型，也可以在 Model 中定义数据修改和操作的业务逻辑。
- View 代表 UI 组件，它负责将数据模型转化成 UI 展现出来。
- ViewModel 监听模型数据的改变和控制视图行为、处理用户交互，简单理解就是一个同步View 和 Model 的对象，连接 Model 和 View

> - 在 MVVM 架构下， View 和 Model 之间并没有直接的联系，而是通过 ViewModel 进行交互， Model 和 ViewModel 之间的交互是双向的， 因此 View 数据的变化会同步到Model中，而Model 数据的变化也会立即反应到 View 上。
> - ViewModel 通过双向数据绑定把 View 层和 Model 层连接了起来，而 View 和 Model 之间的同步工作完全是自动的，无需人为干涉，因此开发者只需关注业务逻辑，不需要手动操作DOM，不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理

请说下你对vue生命周期的理解

什么是vue生命周期？

- 答： Vue 实例从创建到销毁的过程，就是生命周期。从开始创建、初始化数据、编译模板、挂载Dom→渲染、更新→渲染、销毁等一系列过程，称之为 Vue 的生命周期。

vue生命周期的作用是什么？

- 答：它的生命周期中有多个事件钩子，让我们在控制整个Vue实例的过程时更容易形成好的逻辑。

DOM 渲染在哪个周期中就已经完成？

- 答： DOM 渲染在 mounted 中就已经完成了

vue生命周期总共有几个阶段？

> 答：总共分为8个阶段创建前/后，载入前/后，更新前/后，销毁前/后

- **beforeCreate**（创建前） ： Vue 实例的挂载元素 $el 和 数据对象data都为undefined，还未初始化。
- **created**（创建后） ：完成数据观测，属性和方法的运算，初始化事件， $el 还没有

- **beforeMount**（载入前）：Vue 实例的 $el 和 data 都初始化了，但还是挂载之前为虚拟的 dom节点，data.message 还未替换。
- **mounted**（载入后）：Vue 实例挂载完成，data.message 成功渲染。
- **beforeUpdate**（更新前）：当 data 发生变化时，在DOM重新渲染之前调用。
- **updated**（更新后）：在 DOM 重新渲染之后调用。
- **beforeDestroy**（销毁前）：在 Vue 实例销毁之前调用。实例仍然完全可用。
- **destroyed**（销毁后）：在实例销毁之后调用。调用后，所有的事件监听器会被移除，所有的子实例也会被销毁。

第一次页面加载会触发哪几个钩子？

- 答：会触发下面这几个 `beforeCreate`、`created`、`beforeMount`、`mounted`。

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。

Vue实现数据双向绑定的原理：Object.defineProperty()

- `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。

v-if 和 v-show 区别

1. 什么时候元素被渲染

   v-if 如果在初始条件为 false ，则什么也不做，每当条件为真时，都会重新渲染条件元素

   v-show 不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 `display` 的 `block` 或 `none` 进行切换

2. 使用场景选择

v-if 有更高的切换开销，

v-show 有更高的初始渲染开销。

因此，如果需要非常频繁地切换，则使用 v-show 较好；如果在运行后条件很少改变，则使用 v-if 较好。

v-for 与 v-if 的优先级

v-for的优先级比v-if高

计算属性 computed 和 methods 有什么区别

- computed：计算属性是基于它们的依赖属性进行缓存的，只有在它的相关依赖发生改变时才会重新求值。
- method：只要被触发重新渲染， method 调用总会执行该函数。

说出至少 4 种 vue 当中的指令和它的用法

v-if(判断是否隐藏)、v-for(把数据遍历出来)、v-bind(绑定属性)、v-model(实现双向绑定)

vue组件化的理解

组件是可复用的 Vue 实例, 如果网页中的某一个部分需要在多个场景中使用，那么我们可以将其抽出为一个组件进行复用。组件大大提高了代码的复用率。

Vue 组件 data 为什么必须是函数

- 在 `new Vue()` 中， `data` 是可以作为一个对象进行操作的，然而在 `component` 中， `data` 只能以函数的形式存在，不能直接将对象赋值给它。
- 当data选项是一个函数的时候，每个实例可以维护一份被返回对象的独立的拷贝，这样各个实例中的data不会相互影响，是独立的

Vue组件间的参数传递

父组件与子组件传递数据

- 父组件传给子组件：子组件通过 `props` 方法接受父组件传递的数据
- 子组件传给父组件： `$emit` 方法传递参数

非父子组件间的数据传递，兄弟组件传值

可通过 PubSubJS 库来实现非父子组件之间的通信 ，使用 PubSubJS 的消息发布与订阅模式，来进行数据的传递。

路由之间跳转

声明式（标签跳转）

```
<router-link :to="index">
```

编程式（ js跳转）

```
router.push('index')
```

Vue 路由的实现：hash模式 和 history 模式

hash 模式

在浏览器中符号 # ，# 以及#后面的字符称之为 hash ，用 window.location.hash 读取。特点： hash 虽然在 URL 中，但不被包括在 HTTP 请求中；用来指导浏览器动作，对服务端安全无用， hash 不会重加载页面。

history 模式

history 采用 HTML5 的新特性；且提供了两个新方法： pushState() ， replaceState() 可以对浏览器历史记录栈进行修改，以及 popState 事件的监听到状态变更.

Vue 路由的钩子函数

> 首页可以控制导航跳转， beforeEach ， afterEach 等，一般用于页面 title 的修改。一些需要登录才能调整页面的重定向功能。

- beforeEach 主要有3个参数 to ， from ， next 。
- to ： route 即将进入的目标路由对象。
- from ： route 当前导航正要离开的路由。
- next ： function 一定要调用该方法 resolve 这个钩子。执行效果依赖 next 方法的调用参数。可以控制网页的跳转

$route 和 $router 的区别

- $route 是“路由信息对象”，包括 path ， params ， hash ， query ， fullPath ， matched ， name 等路由信息参数。
- 而 $router 是“路由实例”对象包括了路由的跳转方法，钩子函数等

vuex 是什么？怎么使用？哪种功能场景使用它？

- 只用来读取的状态集中放在 store 中； 改变状态的方式是提交 mutations，这是个同步的事物； 异步逻辑应该封装在 action 中。
- 在 main.js 引入 store，注入。新建了一个目录 store， ... export
- **场景有**：单页应用中，组件之间的状态、音乐播放、登录状态、加入购物车

- state ： Vuex 使用单一状态树,即每个应用将仅仅包含一个 store 实例，但单一状态树和模块化并不冲突。存放的数据状态，不可以直接修改里面的数据。
- mutations ： mutations 定义的方法动态修改 Vuex 的 store 中的状态或数据
- getters ：类似 vue 的计算属性，主要用来过滤一些数据。
- action ： actions 可以理解为通过将 mutations 里面处里数据的方法变成可异步的处理数据的方法，简单的说就是异步操作数据。 view 层通过 store.dispath 来分发 action

> modules：项目特别复杂的时候，可以让每一个模块拥有自己的 state、 mutation、 action、 getters，使得结构非常清晰，方便管理

如何让CSS只在当前组件中起作用?

将当前组件的 <style> 修改为 <style scoped>

<keep-alive></keep-alive> 的作用是什么?

- <keep-alive></keep-alive> 包裹动态组件时，会缓存不活动的组件实例,主要用于保留组件状态或避免重新渲染

> 比如有一个列表和一个详情，那么用户就会经常执行打开详情=>返回列表=>打开详情...这样的话列表和详情都是一个频率很高的页面，那么就可以对列表组件使用 <keep-alive></keep-alive> 进行缓存，这样用户每次返回列表的时候，都能从缓存中快速渲染，而不是重新渲染

在Vue中使用插件的步骤

- 采用 ES6 的 import ... from ... 语法或 CommonJS 的 require() 方法引入插件
- 使用全局方法 Vue.use(plugin) 使用插件,可以传入一个选项对象 Vue.use(MyPlugin, { someOption: true })

NextTick 作用

nextTick 可以让我们在下次 DOM 更新循环结束之后执行延迟回调，用于获得更新后的 DOM

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
- 初次加载耗时多