VUE

1.vue中v-for中的key是什么作用?若没有key,会发生什么?

当 Vue.js 用 v-for 正在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。**key 的作用主要是为了高效的更新虚拟DOM。**

没有key,控制台会出现警告

2. v-if和v-show的区别

v-if和 v-show指令都可以控制一个元素的显示和隐藏

如果v-if和v-show的值同时为假时，v-if和v-show绑定的元素都不会在页面中显示，

对于v-show而言，当取值为假时，生成了这个元素，但却将这个元素通过 display:none 的方式隐藏了这个元素，所以在页面显示的时候就不显示

对于v-if而言，当取值为假时，不会生成这个元素，自然而然的也就不会在页面中显示

频繁的需要隐藏和出现就用v-show

3.vue兄弟组件,父子组件是怎么通信的?

1. 父传子： 先在**父组件中绑定变量**``，parent是定义在父组件中的变量/值； 再在**子组件中添加props属性接收**父组件传递过来的变量`props:['msg']`； 最后就可以在子组件中使用``来表示父组件中parent变量中的值了。
2. 子传父： 先在子组件中**绑定事件**`@change="sendChild"`，触发的时候在`setChild`事件中**用**`$emit()`**触发父组件中的函数**，并将子组件中的变量作为参数传递；

```
methods：{
    sendChild:function(){
        this.$emit('transparent',this.msg)
    }
}
```

在父组件中绑定事件``，当子组件触发这个事件的时候，就可以**调用getChild方法获取到传递过来的参数**；

```
methods:{
    getChild(msg){
        this.user=msg;
    }
}
```

1.兄弟组件互相传值，**通过Vuex状态管理传值**： 先通过npm加载vuex，创建store.js文件

```
//store.js
   import Vue from 'vue'
   import Vuex from 'vuex'
   Vue.use(Vuex);
   const state={name:'Alice'};
   const mutations={
       newName(state,message){
           state.name=message
       }
   }
   export default new Vuex.Store({state,mutations})
```

2.兄弟组件互相传值，引入bus.js文件，发布者订阅者模式：

```
import Bus from './bus.js'
//一个子组件触发
methods:{
  Bus.$emit('触发的方法名', 需要传递的值);
}
//一个子组件监听
   mounted:{
       bus.$on("方法名",(传递的值)=>{ })
   }
```

3.兄弟组件互相传值`$root`

```
//一个子组件触发
this.$root.$emit('触发的方法名', 需要传递的值);
//一个子组件监听
this.$root.$off("方法名");//每次进入先关闭一下
this.$root.$on("方法名",(传递的值)=>{ })
```

4. vue 生命周期有哪些?分别有什么作用?什么时候用created和mouted?

beforeCreate（初始化界面前）created（初始化界面后）

beforeMount（渲染dom前）mounted（渲染dom后）

beforeUpdate（更新数据前）updated（更新数据后）

beforeDestroy（卸载组件前）destroyed（卸载组件后）

`beforeCreate`：初始化了部分参数，如果有相同的参数，做了参数合并，执行`beforeCreate`；

`created`：初始化了`Inject`、`Provide`、`props`、`methods`、`data`、`computed`和`watch`，执行`created`；

`beforeMount`：检查是否存在`el`属性，存在的话进行渲染`dom`操作，执行`beforeMount`；

`mounted`：实例化`watcher`，渲染`dom`，执行`mounted`；

`beforeUpdate`：在渲染`dom`后，执行了`mounted`钩子后，在数据更新的时候，执行`beforeUpdate`；

`updated`：检查当前的`watcher`列表中，是否存在当前要更新数据的`watcher`，如果存在就执行`updated`；

`beforeDestroy`：检查是否已经被卸载，如果已经被卸载，就直接`return`出去，否则执行`beforeDestroy`；

`destroyed`：把所有有关自己痕迹的地方，都给删除掉；

created和mounted之间的区别：

- created和mounted中**ajax请求**的区别：created的时候视图未出现，请求较多的情况下，会出现**白屏**；
- created 是在模板渲染成html前调用，即通常初始化某些属性值，然后再渲染成视图，比如初始化、获取屏幕高度调整、赋值等等；

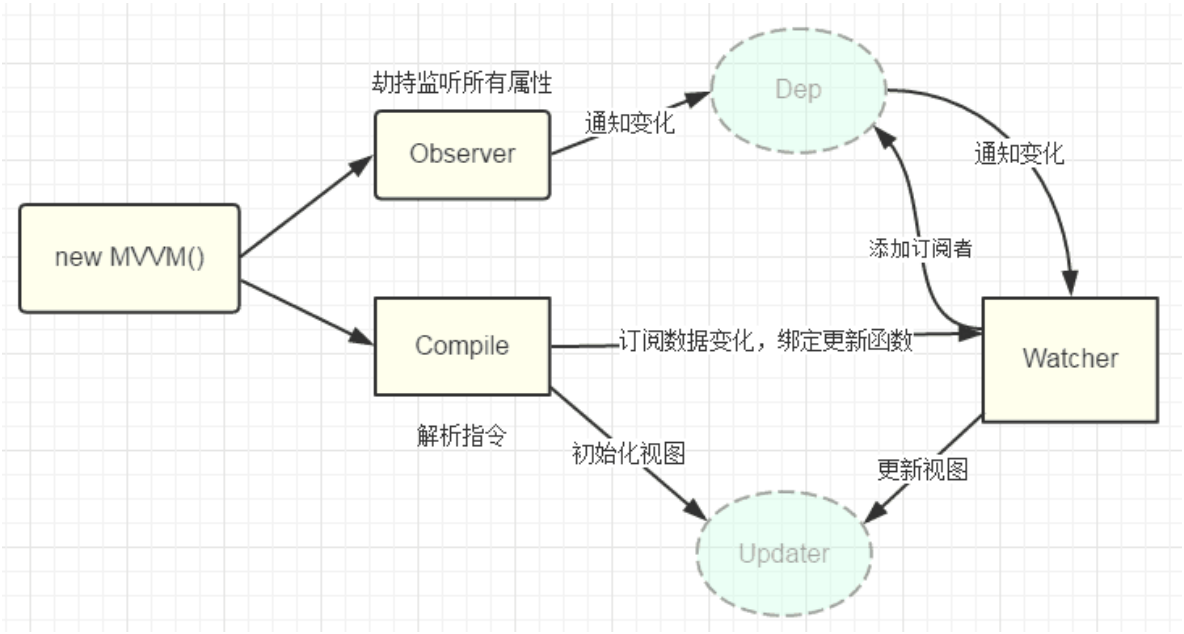
  而mounted是在模板渲染成html后调用，通常是初始化页面完成后，再对html的dom节点进行一些需要的操作；

5. vue中的data为什么是函数?

为了保证组件的独立性 和 可 复用性，data 是一个函数，组件实例化的时候这个函数将会被调用，返回一个对象，计算机会给这个对象分配一个内存地址，你实例化几次，就分配几个内存地址，他们的地址都不一样，所以每个组件中的数据不会相互干扰，改变其中一个组件的状态，其它组件不变。

6. vue双向绑定的原理? vue 数据劫持

mvvm 双向绑定，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，触发相应的监听回调。



具体步骤：

1. 需要 observe 的数据对象进行递归遍历，包括子属性对象的属性，都加上 setter 和 getter
   这样的话，给这个对象的某个值赋值，就会触发 setter，那么就能监听到了数据变化
2. compile 解析模板指令，将模板中的变量替换成数据，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，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图
3. Watcher 订阅者是 Observer 和 Compile 之间通信的桥梁，主要做的事情是:
   - 在自身实例化时往属性订阅器(dep)里面添加自己
   - 自身必须有一个 update() 方法
   - 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，则功成身退。

4. 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 数据变化，通过 Compile 来解析编译模板指令，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 变更的双向绑定效果。

7. keep-alive是什么?有什么作用?

keep-alive用来缓存组件,避免多次加载相应的组件,减少性能消耗，简单一点来说就是从页面1链接到其他页面后回退到页面1不用在重新执行页面1的代码，只会从缓存中加载之前已经缓存的页面1，这样可以减少加载时间及性能消耗，提高用户体验性。

通过设置了keep-alive，可以简单理解为从页面1跳转到页面2后，然后后退到页面1，只会加载缓存中之前已经渲染好的页面1，而不会再次重新加载页面1，及不会再触发页面一种的created等类似的钩子函数，除非自己重新刷新该页面1。

8. 多页应用与单页应用的区别？以及优缺点

多页应用：

- 定义：每一次页面的跳转，后端都会返回一个新的html文件。

优点：1. **首屏事件快**（只请求 html 文件就可以展示页面了，只经历了一个http请求）；2. SEO（**搜索引擎优化**，可以识别 html 内的内容）效果好。

缺点：页面切换**慢**（每次跳转都要发送一个http请求）

单页应用：

- 定义：每一次页面的跳转，都是使用`JS`渲染。页面跳转不使用 a 标签，而是使用，不请求html文件，通过`JS`感知到`URL`的变化，可以用`JS`动态得把当前页面清除掉，再把下一个页面挂载到当前页面上。

优点：页面切换**快。**不需要http请求。

缺点：首屏时间稍慢（除了请求一个html文件，还要请求一个 JS 文件）；SEO差（不认识JS文件中的内容）

9. vue的computed 和 watch的区别以及应用场景?mouted里存放什么?

computed 计算属性	**watch 观察的动作**	**methods**
1. 数据会被缓存，只要依赖不发生改变，即使页面重新渲染，该方法也不会被调用 2.computed中的函数必须用return返回	1. 直接监测一个值的变化，监测值不发生变化，该方法就不会调用； 2. watch只会监听数据的值是否发生改变，而不会去监听数据的地址是否发生改变。也就是说，watch想要监听引用类型数据的变化，需要进行深度监听。 3.watch中的函数有两个参数，前者是newVal，后者是oldVal。	每次页面发生渲染，都会被重新调用
在computed中不要对data中的属性进行赋值操作。如果对data中的属性进行赋值操作了，就是data中的属性发生改变，从而触发computed中的函数，形成死循环了。	监听复杂数据类型需用深度监听（在被监听对象中使用`handler`）；特殊情况下，watch无法监听到数组的变化，特殊情况就是说更改数组中的数据时，数组已经更改，但是视图没有更新。**更改数组必须要用splice()或者$set。**	
使用场景：**当一个值受多个属性影响的时候————购物车最终结算**	使用场景：**当一条数据的更改影响到多条数据的时候————搜索框**	

物车商品结算 computed 计算属性	watch 观察的动作	methods

computed和watch区别：

1、 功能上：computed是计算属性，也就是依赖其它的属性计算所得出最后的值，是用于定义**基于数据之上的数据。**watch是监听一个值的变化，然后执行对应的回调，是在**某个数据变化时做一些事情。**

2、 **是否调用缓存**：computed中的函数所依赖的属性没有发生变化，那么调用当前的函数的时候会从缓存中读取，而watch在每次监听的值发生变化的时候都会执行回调。

3、 **是否调用return**：computed中的函数必须要用return返回，watch中的函数不是必须要用return。

4、 如果一个值依赖多个属性（多对一），用`computed`肯定是更加方便的。如果一个值变化后会引起一系列操作，或者一个值变化会引起一系列值的变化（一对多），用`watch`更加方便一些。

10. vuex的作用?

vuex专为 Vue.js 应用程序开发的**状态管理模式。它采用集中式存储管理应用的所有组件的状态**，并以相应的规则保证状态以一种可预测的方式发生变化。

主要用于管理vue中的数据，可以兄弟组件互相传值；

11. vue中插槽的作用?

插槽就是Vue实现的一套**内容分发的API**，将``元素作为承载分发内容的出口，没有插槽的情况下在组件标签内些一些内容是不起任何作用的。

插槽内可以是任意内容。在`你好`内放置一些内容，输出内容还是在组件中的内容，直接在父组件的`标签中定义的内容不会被渲染。**在子组件template中加入**`元素占位，便能渲染父组件`**标签下的内容。**

具名插槽，当需要多个插槽时，可以使用``的特性：name。这个特性可以用来定义额外的插槽。

插槽默认内容，插槽可以提供一个默认内容，如果如果父组件没有为这个插槽提供了内容，会显示默认的内容。如果父组件为这个插槽提供了内容，则默认的内容会被替换掉。

作用域插槽，作用域插槽就是父组件在**调用子组件的时候给子组件传了一个插槽**，这个插槽为作用域插槽，该插槽必须放在template标签里面，同时声明从子组件接收的数据放在一个自定义属性内，并定义该数据的渲染方式。