* 常用的vue指令有哪些？你怎么理解指令？
* v-if 和 v-show 有什么区别？
* 文本插值有“{{}}一闪而过”问题，怎么处理？
* v-for 可以循环哪些数据类型？v-for列表渲染时为什么加key？
* v-model 有哪些修饰符？
* vue 中怎么阻止冒泡？怎么阻止默认事件？怎么监听键盘enter键？
* 工作中你封装过自定义指令吗？举一些例子
* 计算属性有什么作用？（两大作用）
* 计算属性能不能绑定在v-model上？（可以）
* 怎么理解计算属性的缓存功能？（有且仅有被关联的声明式变量变化时，计算属性才会重新计算）
* 说一下Vue的响应式原理？
* 什么是组件化？你怎么理解组件化？
* 你工作中有没有封装比较好的组件？(选做)
* 父子组件之间如何通信？（父传子、子传父）
* 什么是插槽？什么是具名插槽？

常用的vue指令有哪些？你怎么理解指令？

v-if：判断如果满足条件那么就显示内容，如果不满足条件就不显示内容

v-else：配合v-if使用，当v-if条件不成立，那么就会渲染v-else的元素或组件

v-else-if：可以配合 v-if使用，当我们需要对条件进行分级的时候使用，这样可以划分等级

v-show:后面跟布尔类型的数据，当满足条件就显示内容，当不满足条件就不显示内容，不显示内容区别于v-if，v-if是直接对内容进行销毁，而v-show是添加一个dispaly:none属性，对元素进行隐藏，当我们需要频繁对内容显示与隐藏进行切换的时候，我们需要使用v-show，这样可以节约性能

v-once:当元素和组件发生变化之后，只对元素渲染一次，之后修改数据，不会刷新带有v-once的元素和组件

v-for：后面Array|Object|Number|String类型的数据，有多少个数据就可以渲染多少个元素或组件

v-bind：设置自定义属性，:是v-bind的语法糖形式，可以直接在元素或组件上面使用v-bind来为元素或组件添加自定义属性 比如 name:"name" v-bind:name="name" 当在属性前面添加了v-bind那么后面的name就会变成一个变量形式的数据，如果不带v-bind的话值就是name

v-on:绑定自定义事件的指令，语法糖形式@,比如在元素或组件上设置点击事件就是 v-on:click="onClick" 语法糖形式： @click = "onClick"

v-html:当我们添加v-html，那么对文字解析的时候就会按照html对文本进行解析再渲染到页面上
比如我们设置内容为 content:`<sapn style="color:red;">哈哈哈</span>`
如果我们进行渲染数据的时候没有使用v-html
<h2>{{ content }}</h2> 那么就会原封不动的渲染页面
如果我们使用了 v-html 那么就会先解析之后再渲染 <h2 v-html>{{content}}</h2>
那么我们渲染到页面上的话就是红色的字体

v-pre:跳过渲染过程，直接将定义的数据渲染到页面上

```
<div v-pre>
    <!-- 跳过编译过程，提升速度 -->
    <h2>不会解析大括号里面的内容了，原始展示内容{{message}}</h2>
    <h2>{{counter}}</h2>
</div>
```

页面显示结果：

```
<h2>不会解析大括号里面的内容了，原始展示内容{{message}}</h2>
<h2>{{counter}}</h2>
```

v-cloak: cloak斗篷的含义，有时候我们在刷新页面的的时候会出现页面还没有加载，但是模板就已经出现的情况，当我们设置为 slow-3G的情况下更加常见，那么我们就可以设置 cloak属性，就可以在数据加载完之后再渲染页面

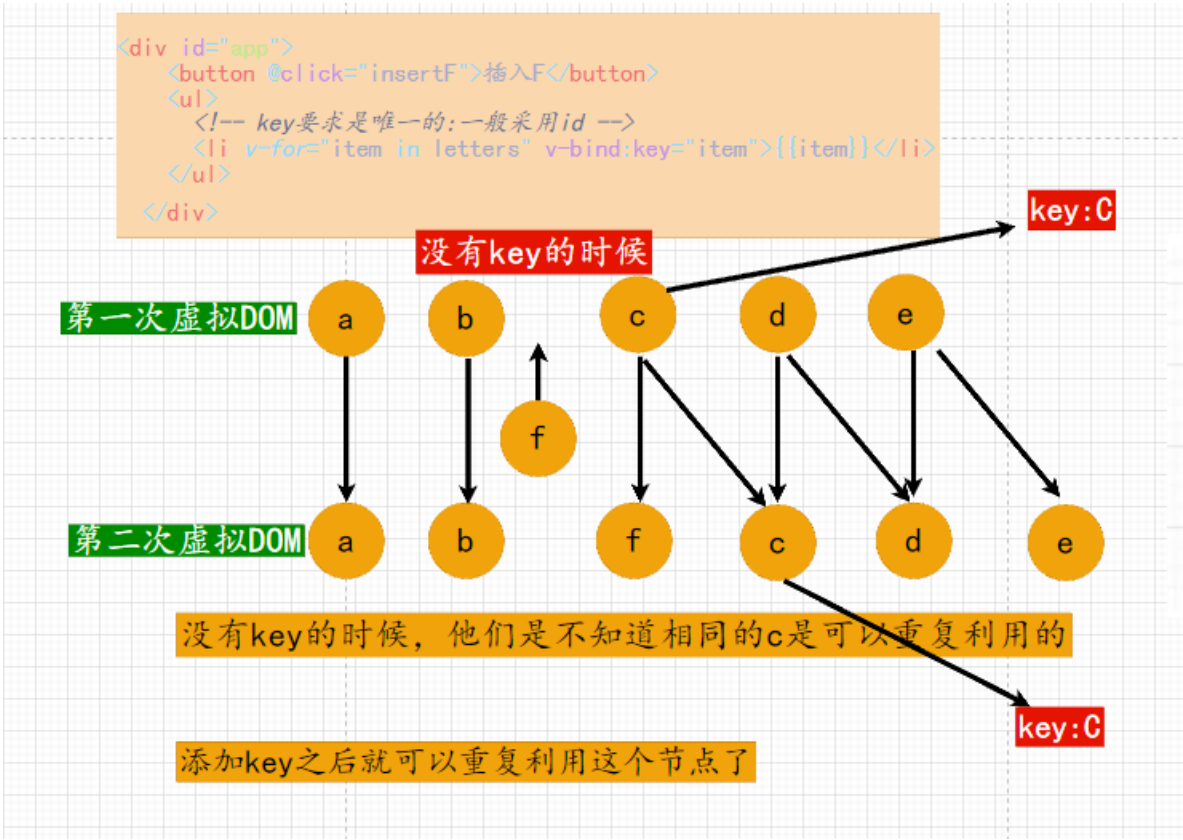
文本插值有“{{}}一闪而过”问题，怎么处理?

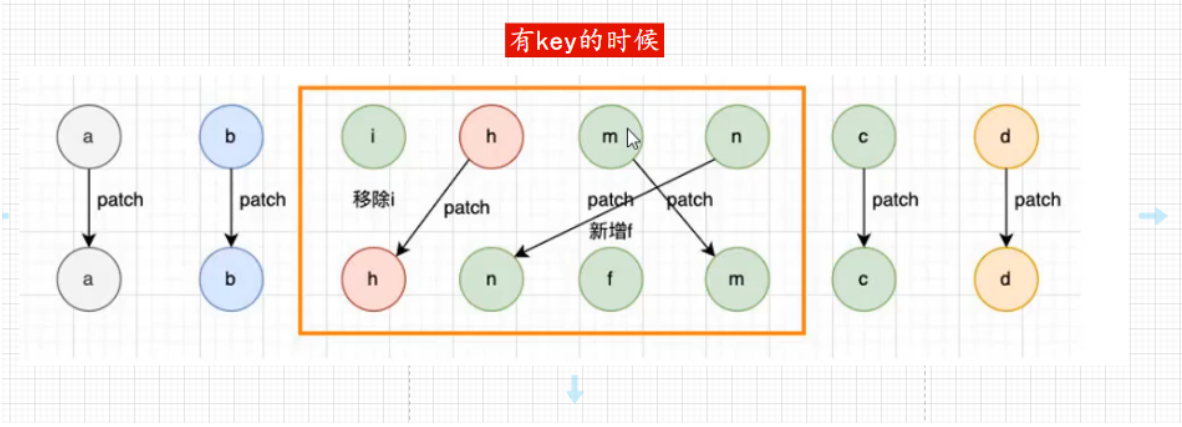
在文本插值元素上面添加v-cloak属性：
浏览器渲染流程
1. 浏览器先解析html元素，当发现有script标签的时候，那么就会去下载对应的JS文件，当下载完之后去重新渲染页面中的内容，但是当JS文件下载速度太慢的话，那么页面中的数据来不及下载JS就会提前渲染，特别是我们设置NetWork为slow-3G的情况下更为明显，那么我们就可以添加一个v-cloak属性，但是v-cloak属性需要配合Css使用，那么我们就可以在页面中设置如下

```
<head>
 <style>
    [v-cloak]{
        display:none
    }
    </style>
    </head>
<body>
  <div>
    <h2 v-cloak>{{message}}</h2>
  </div>
</body>
```

v-for 可以循环哪些数据类型？v-for列表渲染时为什么加key?

String|Array|Number|Object
在虚拟DOM的diff算法中，如果对v-for添加了key属性，那么重新渲染页面的时候，就会对前后的key进行比对，如果前后的key一致，那么就不用重新渲染元素，而是对旧元素进行重复使用，节省性能





v-model 有哪些修饰符?

```
<div id="app">
        <!-- 1. lazy:绑定change事件 -->
        <input type="text" v-model.lazy="message">
        <h2>message:{{message}}</h2>
        <!-- 2.number：自动将内容转换成数字 -->
        <input type="text" v-model.number="counter">
        <h2>counter:{{counter}}-{{typeof counter}}</h2>
        <!-- 3.另外一种情况就是type对应的就是number类型的数字,没必要加number-->
        <input type="number" v-model="counter2">
        <h2>counter:{{counter2}}-{{typeof counter2}}</h2>
        <!-- 4.trim修饰符 -->
        <input type="text" v-model.trim="name">
        <h2>counter:{{name}}-{{typeof name}}</h2>
        <!-- 同时使用多个修饰符 -->
        <input type="text" v-model.lazy.trim="name">
        <h2>counter:{{name}}-{{typeof name}}</h2>
    </div>
```

vue 中怎么阻止冒泡？怎么阻止默认事件？怎么监听键盘enter键?

事件绑定好后，将@click属性修改为@click.stop即可阻止事件冒泡；

```
<div class="parent-name" @click.stop="clickname"></div>

<!-- 事件触发之后是具有冒泡的特性的，同时我们也可以对冒泡进行捕获 -->
<!-- v-on 支持修饰符,修饰符相当于对事件进行了一些特殊处理 -->
<!-- .stop 调用 event.stopPropagation() 阻止冒泡-->
<!-- .prevent 调用 event.preventDefault() 阻止默认事件,比如a元素超链接默认会跳到另一个链接里面-->
<!-- .capture 添加事件监听器时使用 capture 模式 让我们的事件变成一个捕获的事件-->
```

计算属性有什么作用

相较于methods，不管依赖的数据变不变，methods都会重新计算，但是依赖数据不变的时候computed从缓存中获取，不会重新计算
常用的是getter方法，获取数据，也可以使用set方法改变数据
依赖于数据，数据更新，处理结果自动更新

说一下Vue的响应式原理

当Vue组件被创建时，在生命周期的第1个阶段，Vue使用Object.defineProperty()对data中的数据进行深度递归遍历劫持，添加get和set钩子。在生命周期的第二个阶段，指令第一次与声明式变量touch时，开始进行依赖收集，就是收集Watcher，Watcher就是一个更新函数。然后进行第次的页面初始化（首次渲染），当数据发生变化时，Vue再次通过Wathcer更新视图，这就是Vue的响应式原理。

什么是组件化，说一下对组件化的理解

vue组件系统提供了一种抽象，让我们可以使用独立可复用的组件来构建大型应用，任意类型的应用界 面都可以抽象为一个组件树。组件化能提高开发效率，方便重复使用，简化调试步骤，提升项目可维护 性，便于多人协同开发。

```
// 组件通信常用方式
1. props

// child
props: { msg: String }
// parent
<HelloWorld msg="Welcome to Your Vue.js App"/>

2. 自定义事件（子传父）

// child this.$emit('add', good)
// parent
<Cart @add="cartAdd($event)"></Cart>

3. 事件总线
// 任意两个组件之间传值常用事件总线 或 vuex的方式。

// Bus:事件派发、监听和回调管理 class Bus {
constructor(){ this.callbacks = {}
  }
  $on(name, fn){
this.callbacks[name] = this.callbacks[name] || []
this.callbacks[name].push(fn) }
$emit(name, args){ if(this.callbacks[name]){
this.callbacks[name].forEach(cb => cb(args)) }
```

```
} }
// main.js
Vue.prototype.$bus = new Bus()
// child1
this.$bus.$on('foo', handle) // child2 this.$bus.$emit('foo')

4. vuex
// 创建唯一的全局数据管理者store，通过它管理数据并通知组件状态变更。

5. root
// 兄弟组件之间通信可通过共同祖辈搭桥，$parent或$root

// brother1
this.$parent.$on('foo', handle)
// brother2
this.$parent.$emit('foo')

6. $children（vue3已经移除）
// 父组件可以通过$children访问子组件实现父子通信。
// parent
 this.$children[0].xx = 'xxx'

7. listeners

// 包含了父作用域中不作为 prop 被识别 (且获取) 的特性绑定 ( class 和 style 除外)。当一个组件没有 声明任何 prop 时，这里会包含所有父作用域的绑定 ( class 和 style 除外)，并且可以通过 v- bind="$attrs" 传入内部组件——在创建高级别的组件时非常有用。

// child:并未在props中声明foo
<p>{{$attrs.foo}}</p>
// parent
<HelloWorld foo="foo"/>

8. refs
// 获取子节点引用

// parent
<HelloWorld ref="hw"/>
mounted() {
  this.$refs.hw.xx = 'xxx'
}

9. provide/inject
// 能够实现祖先和后代之间传值

// ancestor
provide() {
  return {foo: 'foo'}
}
// descendant
inject: ['foo']

10.插槽
// 插槽语法是Vue 实现的内容分发 API，用于复合组件开发。该技术在通用组件库开发中有大量应用

1. 匿名插槽

// comp1
```

```
<div>
  <slot></slot>
</div>
// parent
<comp>hello</comp>
```

2. 具名插槽
将内容分发到子组件指定位置

```
// comp2
<div>
<slot></slot>
<slot name="content"></slot>
</div>
// parent
<Comp2>
<!-- 默认插槽用default做参数 -->
<template v-slot:default>具名插槽</template> <!-- 具名插槽用插槽名做参数 -->
<template v-slot:content>内容...</template>
</Comp2>
```

3. 作用域插槽
分发内容要用到子组件中的数据

```
// comp3
<div>
<slot :foo="foo"></slot>
</div>
// parent
<Comp3>
<!-- 把v-slot的值指定为作用域上下文对象 --> <template v-slot:default="slotProps"> 来自子组件数据:{{slotProps.foo}} </template>
</Comp3>
```