1. vue 中的性能优化

Vue 应用运行时性能优化措施	引入生产环境的 Vue 文件
使用单文件组件预编译模板	提取组件的 CSS 到单独到文件
利用 Object.freeze()提升性能	扁平化 Store 数据结构
合理使用持久化 Store 数据	组件懒加载
Vue 应用加载性能优化措施	服务端渲染 / 预渲染
减少 http 请求，合理设置 HTTP 缓存	使用浏览器缓存
启用压缩	CSS Sprites
LazyLoad Images	尽量避免使用 eval 和 Function

2. Vue 的实例生命周期

（1） beforeCreate 初始化实例后 数据观测和事件配置之前调用
（2） created 实例创建完成后调用
（3） beforeMount 挂载开始前被用
（4） mounted el 被新建 vm.$el 替换并挂在到实例上之后调用
（5） beforeUpdate 数据更新时调用
（6） updated 数据更改导致的 DOM 重新渲染后调用
（7） beforeDestory 实例被销毁前调用
（8） destroyed 实例销毁后调用

3. Vue 的双向数据绑定的原理

VUE 实现双向数据绑定的原理就是利用了 Object.defineProperty() 这个方法重新定义了对象获取属性值(get)和设置属性值(set)的操作来实现的。
Vue3.0 将用原生 Proxy 替换 Object.defineProperty

4. 为什么要替换 Object.defineProperty？（Proxy 相比于 defineProperty 的优势）

在 Vue 中，Object.defineProperty 无法监控到数组下标的变化，导致直接通过数组的下标给数组设置值，不能实时响应。
Object.defineProperty 只能劫持对象的属性,因此我们需要对每个对象的每个属性进行遍历。

5. 什么是 Proxy？

Proxy 是 ES6 中新增的一个特性，翻译过来意思是"代理"，用在这里表示由它来“代理”某些操作。 Proxy 让我们能够以简洁易懂的方式控制外部对对象的访问。其功能非常类似于设计模式中的代理模式。
Proxy 可以理解成，在目标对象之前架设一层“拦截”，外界对该对象的访问，都必须先通过这层拦截，因此提供了一种机制，可以对外界的访问进行过滤和改写。
使用 Proxy 的核心优点是可以交由它来处理一些非核心逻辑（如：读取或设置对象的某些属性前记录日志；设置对象的某些属性值前，需要验证；某些属性的访问控制等）。 从而可以让对象只需关注于核心逻辑，达到关注点分离，降低对象复杂度等目的。

6.为什么避免 v-if 和 v-for 用在一起

当 Vue 处理指令时，v-for 比 v-if 具有更高的优先级，这意味着 v-if 将分别重复运行于每个 v-for 循环中。通过 v-if 移动到容器元素，不会再重复遍历列表中的每个值。取而代之的是，我们只检查它一次，且不会在 v-if 为否的时候运算 v-for。

7.组件的设计原则

(1)页面上每个独立的可视/可交互区域视为一个组件(比如页面的头部，尾部，可复用的区块)
(2)每个组件对应一个工程目录，组件所需要的各种资源在这个目录下就近维护(组件的就近维护思想体现了前端的工程化思想，为前端开发提供了很好的分治策略，在 vue.js 中，通过.vue 文件将组件依赖的模板，js，样式写在一个文件中)
(每个开发者清楚开发维护的功能单元，它的代码必然存在在对应的组件目录中，在该目录下，可以找到功能单元所有的内部逻辑)
(3)页面不过是组件的容器，组件可以嵌套自由组合成完整的页面

8.vue slot 是做什么的?

答案：可以插入的槽口，比如插座的插孔。

9. 对于 Vue 是一套渐进式框架的理解

每个框架都不可避免会有自己的一些特点，从而会对使用者有一定的要求，这些要求就是主张，主张有强有弱，它的强势程度会影响在业务开发中的使用方式。
1、使用 vue，你可以在原有大系统的上面，把一两个组件改用它实现，当 jQuery 用；
2、也可以整个用它全家桶开发，当 Angular 用；
3、还可以用它的视图，搭配你自己设计的整个下层用。你可以在底层数据逻辑的地方用 OO(Object - Oriented) 面向对象和设计模式的那套理念。 也可以函数式，都可以。
它只是个轻量视图而已，只做了自己该做的事，没有做不该做的事，仅此而已。

你不必一开始就用 Vue 所有的全家桶，根据场景，官方提供了方便的框架供你使用。
场景联想 场景 1： 维护一个老项目管理后台，日常就是提交各种表单了，这时候你可以把 vue 当成一个 js 库来使用，就用来收集 form 表单，和表单验证。
场景 2： 得到 boss 认可， 后面整个页面的 dom 用 Vue 来管理，抽组件，列表用 v-for 来循环，用数据驱动 DOM 的变化
场景 3: 越来越受大家信赖，领导又找你了，让你去做一个移动端 webapp，直接上了 vue 全家桶！
场景 1-3 从最初的只因多看你一眼而用了前端 js 库，一直到最后的大型项目解决方案。

10. vue.js 的两个核心是什么？

答案：数据驱动和组件化思想

11. 请问 v-if 和 v-show 有什么区别

v-show 指令是通过修改元素的 display 的 CSS 属性让其显示或者隐藏
v-if 指令是直接销毁和重建 DOM 达到让元素显示和隐藏的效果

12. vue 常用的修饰符

事件修饰符：.stop .prevent .capture .self .once .passive(不要和.prevent 一起用)
键盘修饰符：.enter .tab .space .up .down .left .right
系统修饰符：.ctrl .alt .shift .meta
鼠标按钮：.left .right .middle
修饰符：.lazy .number .trim

13. v-on 可以监听多个方法吗？

答案：肯定可以的。

```
<input
  type="text"
  :value="name"
  @input="onInput"
  @focus="onFocus"
  @blur="onBlur"
/>
```

14. vue 中 key 值的作用

需要使用 key 来给每个节点做一个唯一标识，Diff 算法就可以正确的识别此节点，找到正确的位置区插入新的节点 所以一句话，key 的作用主要是为了高效的更新虚拟 DOM

15. vue-cli 工程升级 vue 版本

在项目目录里运行 npm upgrade vue vue-template-compiler，不出意外的话，可以正常运行和 build。如果有任何问题，删除 node_modules 文件夹然后重新运行 npm i 即可。（简单的说就是升级 vue 和 vue-template-compiler 两个插件）

16. vue 事件中如何使用 event 对象？

v-on 指令（可以简写为 @）
1、使用不带圆括号的形式，event 对象将被自动当做实参传入；
2、使用带圆括号的形式，我们需要使用 $event 变量显式传入 event 对象。
解析：
一、event 对象
（一）事件的 event 对象
你说你是搞前端的，那么你肯定就知道事件，知道事件，你就肯定知道 event 对象吧？各种的库、框架多少都有针对 event 对象的处理。比如 jquery，通过它内部进行一定的封装，我们开发的时候，就无需关注 event 对象的部分兼容性问题。最典型的，如果我们要阻止默认事件，在 chrome 等浏览器中，我们可能要写一个：

```
event.preventDefault();
```

而在 IE 中，我们则需要写：

```
event.returnValue = false;
```

多亏了 jquery ，跨浏览器的实现，我们统一只需要写：

```
event.preventDefault();
```

兼容？jquery 内部帮我们搞定了。类似的还有比如阻止事件冒泡以以及事件绑定（addEventListener / attachEvent）等，简单到很多的后端都会使用 $('xxx').bind(...)，这不是我们今天的重点，我们往下看。
（二）vue 中的 event 对象
我们知道，相比于 jquery，vue 的事件绑定可以显得更加直观和便捷，我们只需要在模板上添加一个 v-on 指令（还可以简写为 @），即可完成类似于 $('xxx').bind 的效果，少了一个利用选择器查询元素的操作。我们知道，jquery 中，event 对象会被默认当做实参传入到处理函数中，如下

```
$("body").bind("click", function(event) {
```

```
  console.log(typeof event); // object
});
```
这里直接就获取到了 event 对象，那么问题来了，vue 中呢？
```
<div id="app">
    <button v-on:click="click">click me</button>
</div>
...
var app = new Vue({
    el: '#app',
    methods: {
        click(event) {
            console.log(typeof event);    // object
        }
    }
});
```
这里的实现方式看起来和 jquery 是一致的啊，但是实际上，vue 比 jquery 要要复杂得多，jquery 官方也明确的说，v-on 不简单是 addEventListener 的语法糖。在 jquery 中，我们传入到 bind 方法中的回调，只能是一个函数表类型的变量或者一个匿名函数，传递的时候，还不能执行它（在后面加上一堆圆括号），否则就变成了取这一个函数的返回值作为事件回调。而我们知道，vue 的 v-on 指令接受的值可以是函数执行的形式，比如 v-on:click="click(233)" 。这里我们可以传递任何需要传递的参数，甚至可以不传递参数：
```
<div id="app">
    <button v-on:click="click()">click me</button>
</div>
...
var app = new Vue({
    el: '#app',
    methods: {
        click(event) {
            console.log(typeof event);    // undefined
        }
    }
});
```
咦？我的 event 对象呢？怎么不见了？打印看看 arguments.length 也是 0，说明这时候确实没有实参被传入进来。T_T，那我们如果既需要传递参数，又需要用到 event 对象，这个该怎么办呢？

（三）$event

翻看 vue 文档，不难发现，其实我们可以通过将一个特殊变量 $event 传入到回调中解决这个问题：
```
<div id="app">
    <button v-on:click="click($event, 233)">click me</button>
</div>
...
var app = new Vue({
    el: '#app',
    methods: {
        click(event, val) {
            console.log(typeof event);    // object
        }
    }
});
```
好吧，这样看起来就正常了。 简单总结来说：

使用不带圆括号的形式，event 对象将被自动当做实参传入；

使用带圆括号的形式，我们需要使用 $event 变量显式传入 event 对象。

二、乌龙 前面都算是铺垫吧，现在真正的乌龙来了。 翻看小伙伴儿的代码，偶然看到了类似下面的代码：
```
<div id="app">
    <button v-on:click="click(233)">click me</button>
</div>
...
```

```
var app = new Vue({
    el: '#app',
    methods: {
        click(val) {
            console.log(typeof event);    // object
        }
    }
});
```

看到这一段代码，我的内心是崩溃的，丢进 chrome 里面一跑，尼玛还真可以，打印 arguments.length，也是正常的 1。尼玛！这是什么鬼？毁三观啊？ 既没有传入实参，也没有接收的形参，这个 event 对象的来源，要么是上级作用链，要么。。。是全局作用域。。。全局的，不禁想到了 window.event 。再次上 MDN 确认了一下，果然，window.event，ie 和 chrome 都在 window 对象上有这样一个属性：

17.$nextTick 的使用

1、什么是 Vue.nextTick()?

定义：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这个方法，获取更新后的 DOM。所以就衍生出了这个获取更新后的 DOM 的 Vue 方法。所以放在 Vue.nextTick()回调函数中的执行的应该是会对 DOM 进行操作的 js 代码；

理解：nextTick()，是将回调函数延迟在下一次 dom 更新数据后调用，简单的理解是：当数据更新了，在 dom 中渲染后，自动执行该函数，

```
<template>
  <div class="hello">
    <div>
      <button id="firstBtn" @click="testClick()" ref="aa">{{testMsg}}</button>
    </div>
  </div>
</template>

<script>
export default {
  name: 'HelloWorld',
  data () {
    return {
      testMsg:"原始值",
    }
  },
  methods:{
    testClick:function(){
      let that=this;
      that.testMsg="修改后的值";
      console.log(that.$refs.aa.innerText);   //that.$refs.aa 获取指定 DOM，输出：原始值
    }
  }
}
</script>
```

使用 this.$nextTick()

```
methods:{
    testClick:function(){
      let that=this;
      that.testMsg="修改后的值";
      that.$nextTick(function(){
        console.log(that.$refs.aa.innerText);  //输出：修改后的值
      });
    }
  }
```

注意：Vue 实现响应式并不是数据发生变化之后 DOM 立即变化，而是按一定的策略进行 DOM 的更新。$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，

2、什么时候需要用的 Vue.nextTick()？？

1、Vue 生命周期的 created()钩子函数进行的 DOM 操作一定要放在 Vue.nextTick()的回调函数中，原因是在 created()钩子函数执行的时候 DOM 其实并未进行任何渲染，而此时进行 DOM 操作无异于徒劳，所以此处一定要将 DOM 操作的 js 代码放进 Vue.nextTick()的回调函数中。与之对应的就是 mounted 钩子函数，因为该钩子函数执行时所有的 DOM 挂载已完成。

```
created(){
    let that=this;
    that.$nextTick(function(){  //不使用 this.$nextTick()方法会报错
        that.$refs.aa.innerHTML="created 中更改了按钮内容";  //写入到 DOM 元素
    });
}
```

2、当项目中你想在改变 DOM 元素的数据后基于新的 dom 做点什么，对新 DOM 一系列的 js 操作都需要放进 Vue.nextTick()的回调函数中；通俗的理解是：更改数据后当你想立即使用 js 操作新的视图的时候需要使用它

```
<template>
  <div class="hello">
    <h3 id="h">{{testMsg}}</h3>
  </div>
</template>

<script>
export default {
  name: 'HelloWorld',
  data () {
    return {
      testMsg:"原始值",
    }
  },
  methods:{
    changeTxt:function(){
      let that=this;
      that.testMsg="修改后的文本值";  //vue 数据改变，改变 dom 结构
      let domTxt=document.getElementById('h').innerText;  //后续 js 对 dom 的操作
      console.log(domTxt);  //输出可以看到 vue 数据修改后 DOM 并没有立即更新，后续的 dom 都不是最新的
      if(domTxt==="原始值"){
        console.log("文本 data 被修改后 dom 内容没立即更新");
      }else {
        console.log("文本 data 被修改后 dom 内容被马上更新了");
      }
    },

  }
}
</script>
```

正确的用法是：vue 改变 dom 元素结构后使用 vue.$nextTick()方法来实现 dom 数据更新后延迟执行后续代码

```
    changeTxt:function(){
      let that=this;
      that.testMsg="修改后的文本值";  //修改 dom 结构

      that.$nextTick(function(){  //使用 vue.$nextTick()方法可以 dom 数据更新后延迟执行
        let domTxt=document.getElementById('h').innerText;
        console.log(domTxt);  //输出可以看到 vue 数据修改后并没有 DOM 没有立即更新，
        if(domTxt==="原始值"){
          console.log("文本 data 被修改后 dom 内容没立即更新");
```

```
        }else {
          console.log("文本 data 被修改后 dom 内容被马上更新了");
        }
      });
    }
```

3、在使用某个第三方插件时 ，希望在 vue 生成的某些 dom 动态发生变化时重新应用该插件，也会用到该方法，这时候就需要在 $nextTick 的回调函数中执行重新应用插件的方法。

Vue.nextTick(callback) 使用原理：

原因是，Vue 是异步执行 dom 更新的，一旦观察到数据变化，Vue 就会开启一个队列，然后把在同一个事件循环 (event loop) 当中观察到数据变化的 watcher 推送进这个队列。如果这个 watcher 被触发多次，只会被推送到队列一次。这种缓冲行为可以有效的去掉重复数据造成的不必要的计算和 DOm 操作。而在下一个事件循环时，Vue 会清空队列，并进行必要的 DOM 更新。 当你设置 vm.someData = 'new value'，DOM 并不会马上更新，而是在异步队列被清除，也就是下一个事件循环开始时执行更新时才会进行必要的 DOM 更新。如果此时你想要根据更新的 DOM 状态去做某些事情，就会出现问题。。为了在数据变化之后等待 Vue 完成更新 DOM ，可以在数据变化之后立即使用 Vue.nextTick(callback) 。这样回调函数在 DOM 更新完成后就会调用。

参与互动

18. Vue 组件中 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，这并非是 Vue 自身如此设计，而是跟 JavaScript 特性相关，我们来回顾下 JavaScript 的原型链

```
var Component = function() {};
Component.prototype.data = {
  message: "Love"
};
var component1 = new Component(),
  component2 = new Component();
component1.data.message = "Peace";
console.log(component2.data.message); // Peace
```

以上两个实例都引用同一个对象，当其中一个实例属性改变时，另一个实例属性也随之改变，只有当两个实例拥有自己的作用域时，才不会互相干扰 ！！！！

```
var Component = function() {
  this.data = this.data();
};
Component.prototype.data = function() {
  return {
    message: "Love"
  };
};
var component1 = new Component(),
  component2 = new Component();
component1.data.message = "Peace";
console.log(component2.data.message); // Love
```

20. vue 中子组件调用父组件的方法

答案：

- 第一种方法是直接在子组件中通过 this.$parent.event 来调用父组件的方法
- 第二种方法是在子组件里用$emit 向父组件触发一个事件，父组件监听这个事件就行了
- 第三种是父组件把方法传入子组件中，在子组件里直接调用这个方法

解析：

第一种方法是直接在子组件中通过 this.$parent.event 来调用父组件的方法

父组件

```
<template>
  <div>
    <child></child>
  </div>
</template>
<script>
```

```
  import child from '~/components/dam/child';
  export default {
    components: {
      child
    },
    methods: {
      fatherMethod() {
        console.log('测试');
      }
    }
  };
</script>
```
子组件
```
<template>
  <div>
    <button @click="childMethod()">点击</button>
  </div>
</template>
<script>
  export default {
    methods: {
      childMethod() {
        this.$parent.fatherMethod();
      }
    }
  };
</script>
```
第二种方法是在子组件里用$emit 向父组件触发一个事件，父组件监听这个事件就行了
父组件
```
<template>
  <div>
    <child @fatherMethod="fatherMethod"></child>
  </div>
</template>
<script>
  import child from "~/components/dam/child";
  export default {
    components: {
      child
    },
    methods: {
      fatherMethod() {
        console.log("测试");
      }
    }
  };
</script>
```
子组件
```
<template>
  <div>
    <button @click="childMethod()">点击</button>
  </div>
</template>
<script>
  export default {
    methods: {
```

```
      childMethod() {
        this.$emit("fatherMethod");
      }
    }
  };
</script>
```

第三种是父组件把方法传入子组件中，在子组件里直接调用这个方法

父组件

```
<template>
  <div>
    <child :fatherMethod="fatherMethod"></child>
  </div>
</template>
<script>
  import child from "~/components/dam/child";
  export default {
    components: {
      child
    },
    methods: {
      fatherMethod() {
        console.log("测试");
      }
    }
  };
</script>
```

子组件

```
<template>
  <div>
    <button @click="childMethod()">点击</button>
  </div>
</template>
<script>
  export default {
    props: {
      fatherMethod: {
        type: Function,
        default: null
      }
    },
    methods: {
      childMethod() {
        if (this.fatherMethod) {
          this.fatherMethod();
        }
      }
    }
  };
</script>
```

21. vue 中父组件调用子组件的方法

答案：使用$refs

解析：

父组件

```
<template>
  <div>
    <button @click="clickParent">点击</button>
```

```
    <child ref="mychild"></child>
  </div>
</template>

<script>
  import Child from "./child";
  export default {
    name: "parent",
    components: {
      child: Child
    },
    methods: {
      clickParent() {
        this.$refs.mychild.parentHandleclick("嘿嘿嘿"); // 划重点！！！！
      }
    }
  };
</script>
```

子组件

```
<template>
  <div>
    child
  </div>
</template>

<script>
  export default {
    name: "child",
    props: "someprops",
    methods: {
      parentHandleclick(e) {
        console.log(e);
      }
    }
  };
</script>
```

22. vue 中 keep-alive 组件的作用

答案：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。

用法也很简单：

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

props _ include - 字符串或正则表达，只有匹配的组件会被缓存 _ exclude - 字符串或正则表达式，任何匹配的组件都不会被缓存

```
// 组件 a
export default {
  name: "a",
  data() {
    return {};
  }
};
<keep-alive include="a">
  <component>
    <!-- name 为 a 的组件将被缓存！ -->
```

```
  </component> </keep-alive
>可以保留它的状态或避免重新渲染
<keep-alive exclude="a">
  <component>
    <!-- 除了 name 为 a 的组件都将被缓存！ -->
  </component> </keep-alive
>可以保留它的状态或避免重新渲染
```
但实际项目中,需要配合 vue-router 共同使用.
router-view 也是一个组件，如果直接被包在 keep-alive 里面，所有路径匹配到的视图组件都会被缓存：
```
<keep-alive>
  <router-view>
    <!-- 所有路径匹配到的视图组件都会被缓存！ -->
  </router-view>
</keep-alive>
```
如果只想 router-view 里面某个组件被缓存，怎么办？
增加 router.meta 属性
```
// routes 配置
export default [
  {
    path: "/",
    name: "home",
    component: Home,
    meta: {
      keepAlive: true // 需要被缓存
    }
  },
  {
    path: "/:id",
    name: "edit",
    component: Edit,
    meta: {
      keepAlive: false // 不需要被缓存
    }
  }
];
<keep-alive>
    <router-view v-if="$route.meta.keepAlive">
        <!-- 这里是会被缓存的视图组件，比如 Home！ -->
    </router-view>
</keep-alive>

<router-view v-if="!$route.meta.keepAlive">
    <!-- 这里是不被缓存的视图组件，比如 Edit！ -->
</router-view>
```
23. vue 中如何编写可复用的组件？
答案：总结组件的职能，什么需要外部控制（即 props 传啥），组件需要控制外部吗（$emit）,是否需要插槽（slot）
24. 什么是 vue 生命周期和生命周期钩子函数？
vue 的生命周期就是 vue 实例从创建到销毁的过程
25. vue 生命周期钩子函数有哪些？
26. vue 如何监听键盘事件中的按键？
27. vue 更新数组时触发视图更新的方法
1.Vue.set 可以设置对象或数组的值，通过 key 或数组索引，可以触发视图更新
数组修改
```
Vue.set(array, indexOfItem, newValue)
this.array.$set(indexOfItem, newValue)
```

对象修改
Vue.set(obj, keyOfItem, newValue)
this.obj.$set(keyOfItem, newValue)
2.Vue.delete 删除对象或数组中元素，通过 key 或数组索引，可以触发视图更新
数组修改
Vue.delete(array, indexOfItem)
this.array.$delete(indexOfItem)
对象修改
Vue.delete(obj, keyOfItem)
this.obj.$delete(keyOfItem)
3.数组对象直接修改属性，可以触发视图更新
this.array[0].show = true;
this.array.forEach(function(item){
    item.show = true;
});
4.splice 方法修改数组，可以触发视图更新
this.array.splice(indexOfItem, 1, newElement)
5.数组整体修改，可以触发视图更新
var tempArray = this.array;
tempArray[0].show = true;
this.array = tempArray;
6.用 Object.assign 或 lodash.assign 可以为对象添加响应式属性，可以触发视图更新
//Object.assign 的单层的覆盖前面的属性，不会递归的合并属性
this.obj = Object.assign({},this.obj,{a:1, b:2})

//assign 与 Object.assign 一样
this.obj = _.assign({},this.obj,{a:1, b:2})
//merge 会递归的合并属性
this.obj = _.merge({},this.obj,{a:1, b:2})
7.Vue 提供了如下的数组的变异方法，可以触发视图更新
push()　　pop()　　shift()　　unshift()
splice()　　sort()　　reverse()

28. vue 中对象更改检测的注意事项

29. 解决非工程化项目初始化页面闪动问题

30. v-for 产生的列表，实现 active 的切换

31. v-model 语法糖的组件中的使用

32. 十个常用的自定义过滤器

33. vue 等单页面应用及其优缺点

优点： 1、用户体验好、快，内容的改变不需要重新加载整个页面，避免了不必要的跳转和重复渲染。 2、前后端职责业分离（前端负责 view，后端负责 model），架构清晰 3、减轻服务器的压力
缺点： 1、SEO（搜索引擎优化）难度高 2、初次加载页面更耗时 3、前进、后退、地址栏等，需要程序进行管理，所以会大大提高页面的复杂性和逻辑的难度

34. 什么是 vue 的计算属性？

答案：先来看一下计算属性的定义： 当其依赖的属性的值发生变化的时，计算属性会重新计算。反之则使用缓存中的属性值。 计算属性和 vue 中的其它数据一样，都是响应式的，只不过它必须依赖某一个数据实现，并且只有它依赖的数据的值改变了，它才会更新。

36. vue 弹窗后如何禁止滚动条滚动？

37.vue 怎么实现页面的权限控制

答案：利用 vue-router 的 beforeEach 事件，可以在跳转页面前判断用户的权限（利用 cookie 或 token），是否能够进入此页面，如果不能则提示错误或重定向到其他页面，在后台管理系统中这种场景经常能遇到。

38.$route 和$router 的区别

答案：$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

39.watch 的作用是什么

答案：watch 主要作用是监听某个数据值的变化。和计算属性相比除了没有缓存，作用是一样的。

借助 watch 还可以做一些特别的事情，例如监听页面路由，当页面跳转时，我们可以做相应的权限控制，拒绝没有权限的用户访问页面。

40. 计算属性的缓存和方法调用的区别

计算属性是基于数据的依赖缓存，数据发生变化，缓存才会发生变化，如果数据没有发生变化，调用计算属性直接调用的是存储的缓存值；

而方法每次调用都会重新计算；所以可以根据实际需要选择使用，如果需要计算大量数据，性能开销比较大，可以选用计算属性，如果不能使用缓存可以使用方法；

其实这两个区别还应加一个 watch，watch 是用来监测数据的变化，和计算属性相比，是 watch 没有缓存，但是一般想要在数据变化时响应时，或者执行异步操作时，可以选择 watch

41.vue 的双向绑定的原理，和 angular 的对比

42. vue 如何优化首屏加载速度？

43. vue 打包命令是什么？

答案：npm run build

44. vue 打包后会生成哪些文件？

45. 如何配置 vue 打包生成文件的路径？

46. vue 的服务器端渲染

47. vue 开发命令 npm run dev 输入后的执行过程

48. 什么是 Virtual DOM？

49. 响应式系统的基本原理

vue 响应式的原理，首先对象传入 vue 实例作为 data 对象时，首先被 vue 遍历所有属性，调用 Object.defineProperty 设置为 getter 和 setter，每个组件都有一个 watcher 对象，在组件渲染的过程中，把相关的数据都注册成依赖，当数据发生 setter 变化时，会通知 watcehr，从而更新相关联的组件

50. Vue.js 全局运行机制

51. 如何编译 template 模板？

52. diff 算法

53. 批量异步更新策略及 nextTick 原理？

54. Vue 中如何实现 proxy 代理？

webpack 自带的 devServer 中集成了 http-proxy-middleware。配置 devServer 的 proxy 选项即可

```
proxyTable: {
   '/api': {
    target: 'http://192.168.149.90:8080/', // 设置你调用的接口域名和端口号
    changeOrigin: true,   // 跨域
    pathRewrite: {
     '^/api': '/'
    }
   }
  }
```

55. vue 中如何实现 tab 切换功能？

56. vue 中如何利用 keep-alive 标签实现某个组件缓存功能？

57. vue 中实现切换页面时为左滑出效果

58. vue 中央事件总线的使用

59. vue 的渲染机制

60. vue 在什么情况下在数据发生改变的时候不会触发视图更新

v-for 遍历的数组，当数组内容使用的是 arr[0].xx =xx 更改数据，vue 无法监测到 vm.arr.length = newLength 也是无法检测的到的

61.vue 的优点是什么？

低耦合。视图（View）可以独立于 Model 变化和修改，一个 ViewModel 可以绑定到不同的"View"上，当 View 变化的时候 Model 可以不变，当 Model 变化的时候 View 也可以不变。

可重用性。你可以把一些视图逻辑放在一个 ViewModel 里面，让很多 view 重用这段视图逻辑。

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。

可测试。界面素来是比较难于测试的，而现在测试可以针对 ViewModel 来写。

62.vue 如何实现按需加载配合 webpack 设置

webpack 中提供了 require.ensure()来实现按需加载。以前引入路由是通过 import 这样的方式引入，改为 const 定义的方式进行引入。

不进行页面按需加载引入方式：import home from '../../common/home.vue'

进行页面按需加载的引入方式：const home = r => require.ensure([], () => r
(require('../../common/home.vue')))
在音乐 app 中使用的路由懒加载方式为：
```
const Recommend = (resolve) => {
  import('components/recommend/recommend').then((module) => {
    resolve(module)
  })
}

const Singer = (resolve) => {
  import('components/singer/singer').then((module) => {
    resolve(module)
  })
}
```
63.如何让 CSS 只在当前组件中起作用
答案：将当前组件的<style>修改为<style scoped>
64.指令 v-el 的作用是什么?
答案：提供一个在页面上已存在的 DOM 元素作为 Vue 实例的挂载目标.可以是 CSS 选择器，也可以是一个 HTMLElement 实例
65.vue-loader 是什么？使用它的用途有哪些？
vue-loader 是解析 .vue 文件的一个加载器，将 template/js/style 转换成 js 模块。
用途：js 可以写 es6、style 样式可以 scss 或 less；template 可以加 jade 等。
66.vue 和 angular 的优缺点以及适用场合?
67.你们 vue 项目是打包了一个 js 文件，一个 css 文件，还是有多个文件？
68.vue 遇到的坑，如何解决的？
69.vuex 工作原理详解
vuex 整体思想诞生于 flux,可其的实现方式完完全全的使用了 vue 自身的响应式设计，依赖监听、依赖收集都属于 vue 对对象 Property set get 方法的代理劫持。最后一句话结束 vuex 工作原理，vuex 中的 store 本质就是没有 template 的隐藏着的 vue 组件；
解析：vuex 的原理其实非常简单，它为什么能实现所有的组件共享同一份数据？ 因为 vuex 生成了一个 store 实例，并且把这个实例挂在了所有的组件上，所有的组件引用的都是同一个 store 实例。 store 实例上有数据，有方法，方法改变的都是 store 实例上的数据。由于其他组件引用的是同样的实例，所以一个组件改变了 store 上的数据， 导致另一个组件上的数据也会改变，就像是一个对象的引用。
70.vuex 是什么？怎么使用？哪种功能场景使用它？
vue 框架中状态管理。在 main.js 引入 store，注入。新建一个目录 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
main.js:
```
import store from './store'

new Vue({
el:'#app',
store
})
```
71.vuex 有哪几种属性？
有五种，分别是 State、 Getter、Mutation 、Action、 Module
vuex 的 State 特性
A、Vuex 就是一个仓库，仓库里面放了很多对象。其中 state 就是数据源存放地，对应于一般 Vue 对象里面的 data
B、state 里面存放的数据是响应式的，Vue 组件从 store 中读取数据，若是 store 中的数据发生改变，依赖这个数据的组件也会发生更新
C、它通过 mapState 把全局的 state 和 getters 映射到当前组件的 computed 计算属性中
- vuex 的 Getter 特性

A、getters 可以对 State 进行计算操作，它就是 Store 的计算属性
B、 虽然在组件内也可以做计算属性，但是 getters 可以在多组件之间复用
C、 如果一个状态只在一个组件内使用，是可以不用 getters

- vuex 的 Mutation 特性

Action 类似于 mutation，不同在于：Action 提交的是 mutation，而不是直接变更状态；Action 可以包含任意异步操作。

72.不用 Vuex 会带来什么问题？

可维护性会下降，想修改数据要维护三个地方；

可读性会下降，因为一个组件里的数据，根本就看不出来是从哪来的；

增加耦合，大量的上传派发，会让耦合性大大增加，本来 Vue 用 Component 就是为了减少耦合，现在这么用，和组件化的初衷相背。

73.vue-router 如何响应 路由参数 的变化？

74.完整的 vue-router 导航解析流程

75.vue-router 有哪几种导航钩子（ 导航守卫 ）？

答案：三种

第一种是全局导航钩子：router.beforeEach(to,from,next)，作用：跳转前进行判断拦截。 第二种：组件内的钩子； 第三种：单独路由独享组件

76.vue-router 的几种实例方法以及参数传递

77.怎么定义 vue-router 的动态路由？怎么获取传过来的动态参数？

答案：在 router 目录下的 index.js 文件中，对 path 属性加上/:id。 使用 router 对象的 params.id

78.vue-router 如何定义嵌套路由？

79.<router-link></router-link>组件及其属性

80.vue-router 实现路由懒加载（ 动态加载路由 ）

81.vue-router 路由的两种模式

答案：hash history

82.history 路由模式与后台的配合

83.vue 路由实现原理?或 vue-router 原理?

说简单点，vue-router 的原理就是通过对 URL 地址变化的监听，继而对不同的组件进行渲染。 每当 URL 地址改变时，就对相应的组件进行渲染。原理是很简单，实现方式可能有点复杂，主要有 hash 模式和 history 模式。 如果想了解得详细点，建议百度或者阅读源码。

84.什么是 MVVM?

答案：1.拆分说明（M，V，VM 都是干啥的） 2.之间联系（Model 和 ViewModel 的双向数据绑定）

MVVM 是 Model-View-ViewModel 的缩写。MVVM 是一种设计思想。Model 层代表数据模型，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 是一个同步 View 和 Model 的对象（桥梁）。

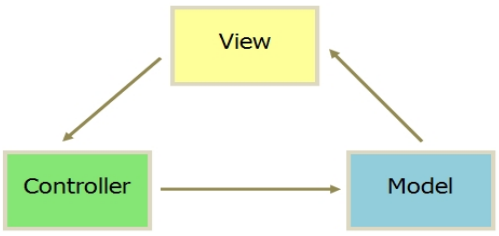
在 MVVM 架构下，View 和 Model 之间并没有直接的联系，而是通过 ViewModel 进行交互，Model 和 ViewModel 之间的交互是双向的， 因此 View 数据的变化会同步到 Model 中，而 Model 数据的变化也会立即反应到 View 上。

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。

85.MVC、MVP 与 MVVM 模式

一、MVC

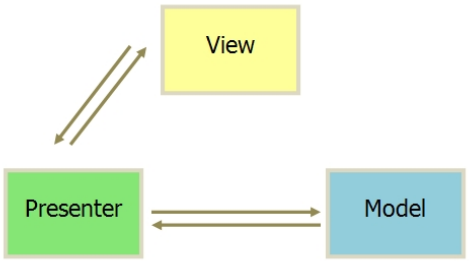
通信方式如下



1. 视图（View）：用户界面。 传送指令到 Controller
2. 控制器（Controller）：业务逻辑 完成业务逻辑后，要求 Model 改变状态
3. 模型（Model）：数据保存 将新的数据发送到 View，用户得到反馈

二、MVP

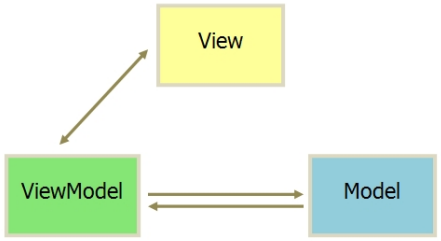
通信方式如下



1. 各部分之间的通信，都是双向的。
2. View 与 Model 不发生联系，都通过 Presenter 传递。
3. View 非常薄，不部署任何业务逻辑，称为"被动视图"（Passive View），即没有任何主动性，而 Presenter 非常厚，所有逻辑都部署在那里。

五、MVVM

MVVM 模式将 Presenter 改名为 ViewModel，基本上与 MVP 模式完全一致。通信方式如下



唯一的区别是，它采用双向绑定（data-binding）：View 的变动，自动反映在 ViewModel，反之亦然。

86.常见的实现 MVVM 几种方式

87.解释下 Object.defineProperty()方法

答案：这是 js 中一个非常重要的方法，ES6 中某些方法的实现依赖于它，VUE 通过它实现双向绑定，此方法会直接在一个对象上定义一个新属性，或者修改一个已经存在的属性， 并返回这个对象

语法

```
Object.defineProperty(object, attribute, descriptor)
```

- 这三个参数都是必输项
- 第一个参数为 目标对象
- 第二个参数为 需要定义的属性或者方法
- 第三个参数为 目标属性所拥有的特性

descriptor

前两个参数都很明确，重点是第三个参数 descriptor， 它有以下取值

- value: 属性的值
- writable: 属性的值是否可被重写（默认为 false）
- configurable: 总开关，是否可配置，若为 false，则其他都为 false（默认为 false）
- enumerable: 属性是否可被枚举（默认为 false）
- get: 获取该属性的值时调用
- set: 重写该属性的值时调用

一个例子

```
var a = {};
Object.defineProperty(a, "b", {
  value: 123
});
console.log(a.b); //123
a.b = 456;
console.log(a.b); //123
a.c = 110;
for (item in a) {
```

```
  console.log(item, a[item]); //c 110
}
```
因为 writable 和 enumerable 默认值为 false, 所以对 a.b 赋值无效，也无法遍历它

configurable

总开关，是否可配置，设置为 false 后，就不能再设置了，否则报错， 例子
```
var a = {};
Object.defineProperty(a, "b", {
  configurable: false
});
Object.defineProperty(a, "b", {
  configurable: true
});
//error: Uncaught TypeError: Cannot redefine property: b
```
writable

是否可重写
```
var a = {};
Object.defineProperty(a, "b", {
  value: 123,
  writable: false
});
console.log(a.b); // 打印 123
a.b = 25; // 没有错误抛出（在严格模式下会抛出，即使之前已经有相同的值）
console.log(a.b); // 打印 123， 赋值不起作用。
```
enumerable

属性特性 enumerable 定义了对象的属性是否可以在 for...in 循环和 Object.keys() 中被枚举
```
var a = {};
Object.defineProperty(a, "b", {
  value: 3445,
  enumerable: true
});
console.log(Object.keys(a)); // 打印["b"]
```
enumerable 改为 false
```
var a = {};
Object.defineProperty(a, "b", {
  value: 3445,
  enumerable: false //注意咯这里改了
});
console.log(Object.keys(a)); // 打印[]
```
set 和 get

如果设置了 set 或 get, 就不能设置 writable 和 value 中的任何一个，否则报错
```
var a = {};
Object.defineProperty(a, "abc", {
  value: 123,
  get: function() {
    return value;
  }
});
//Uncaught TypeError: Invalid property descriptor. Cannot both specify accessors and a value or
writable attribute, #<Object> at Function.defineProperty
```
对目标对象的目标属性 赋值和取值 时， 分别触发 set 和 get 方法
```
var a = {};
var b = 1;
Object.defineProperty(a, "b", {
  set: function(newValue) {
    b = 99;
```

```
    console.log("你要赋值给我,我的新值是" + newValue);
  },
  get: function() {
    console.log("你取我的值");
    return 2; //注意这里，我硬编码返回 2
  }
});
a.b = 1; //打印 你要赋值给我,我的新值是 1
console.log(b); //打印 99
console.log(a.b); //打印 你取我的值
//打印 2    注意这里，和我的硬编码相同的
```

上面的代码中，给 a.b 赋值，b 的值也跟着改变了。原因是给 a.b 赋值，自动调用了 set 方法，在 set 方法中改变了 b 的值。vue 双向绑定的原理就是这个。

88.实现一个自己的 MVVM（原理剖析）

89.递归组件的使用

答案：组件是可以在自己的模板中调用自身的，不过他们只能通过 name 选项来做这件事

90.Obj.keys()与 Obj.defineProperty

91.发布-订阅模式

92.实现 MVVM 的思路分析

93.mvvm 和 mvc 区别？它和其它框架（jquery）的区别是什么？哪些场景适合？

mvc 和 mvvm 其实区别并不大。都是一种设计思想。主要就是 mvc 中 Controller 演变成 mvvm 中的 viewModel。mvvm 主要解决了 mvc 中大量的 DOM 操作使页面渲染性能降低，加载速度变慢，影响用户体验。

区别：vue 数据驱动，通过数据来显示视图层而不是节点操作。

场景：数据操作比较多的场景，更加便捷

94. 构建的 vue-cli 工程都到了哪些技术，它们的作用分别是什么？

1、vue.js：vue-cli 工程的核心，主要特点是 双向数据绑定 和 组件系统。

2、vue-router：vue 官方推荐使用的路由框架。

3、vuex：专为 Vue.js 应用项目开发的状态管理器，主要用于维护 vue 组件间共用的一些 变量 和 方法。

4、axios（ 或者 fetch 、ajax ）：用于发起 GET 、或 POST 等 http 请求，基于 Promise 设计。

5、vux 等：一个专为 vue 设计的移动端 UI 组件库。

6、创建一个 emit.js 文件，用于 vue 事件机制的管理。

7、webpack：模块加载和 vue-cli 工程打包器。

95. vue-cli 工程常用的 npm 命令有哪些？

答案：npm install、npm run dev、npm run build --report 等

解析：

下载 node_modules 资源包的命令：npm install

启动 vue-cli 开发环境的 npm 命令：npm run dev

vue-cli 生成 生产环境部署资源 的 npm 命令：npm run build

用于查看 vue-cli 生产环境部署资源文件大小的 npm 命令：npm run build --report，此命令必答

命令效果：在浏览器上自动弹出一个 展示 vue-cli 工程打包后 app.js、manifest.js、vendor.js 文件里面所包含代码的页面。可以具此优化 vue-cli 生产环境部署的静态资源，提升 页面 的加载速度。

96. 请说出 vue-cli 工程中每个文件夹和文件的用处

vue-cli 目录解析：

build 文件夹：用于存放 webpack 相关配置和脚本。开发中仅 偶尔使用 到此文件夹下 webpack.base.conf.js 用于配置 less、sass 等 css 预编译库，或者配置一下 UI 库。

config 文件夹：主要存放配置文件，用于区分开发环境、线上环境的不同。 常用到此文件夹下 config.js 配置开发环境的 端口号、是否开启热加载 或者 设置生产环境的静态资源相对路径、是否开启 gzip 压缩、npm run build 命令打包生成静态资源的名称和路径等。

dist 文件夹：默认 npm run build 命令打包生成的静态资源文件，用于生产部署。

node_modules：存放 npm 命令下载的开发环境和生产环境的依赖包。

src: 存放项目源码及需要引用的资源文件。

src 下 assets：存放项目中需要用到的资源文件，css、js、images 等。

src 下 componets：存放 vue 开发中一些公共组件：header.vue、footer.vue 等。

src 下 emit：自己配置的 vue 集中式事件管理机制。

src 下 router：vue-router vue 路由的配置文件。

src 下 service：自己配置的 vue 请求后台接口方法。

src 下 page：存在 vue 页面组件的文件夹。
src 下 util：存放 vue 开发过程中一些公共的.js 方法。
src 下 vuex：存放 vuex 为 vue 专门开发的状态管理器。
src 下 app.vue：使用标签<route-view></router-view>渲染整个工程的.vue 组件。
src 下 main.js：vue-cli 工程的入口文件。
index.html：设置项目的一些 meta 头信息和提供 用于挂载 vue 节点。
package.json：用于 node_modules 资源部 和 启动、打包项目的 npm 命令管理。

97. config 文件夹 下 index.js 的对于工程 开发环境 和 生产环境 的配置
答案：
build 对象下 对于 生产环境 的配置：
index：配置打包后入口.html 文件的名称以及文件夹名称
assetsRoot：配置打包后生成的文件名称和路径
assetsPublicPath：配置 打包后 .html 引用静态资源的路径，一般要设置成 "./"
productionGzip：是否开发 gzip 压缩，以提升加载速度

dev 对象下 对于 开发环境 的配置：
port：设置端口号
autoOpenBrowser：启动工程时，自动打开浏览器
proxyTable：vue 设置的代理，用以解决 跨域 问题
98. 请你详细介绍一些 package.json 里面的配置
scripts：npm run xxx 命令调用 node 执行的 .js 文件
dependencies：生产环境依赖包的名称和版本号，即这些 依赖包 都会打包进 生产环境的 JS 文件里面
devDependencies：开发环境依赖包的名称和版本号，即这些 依赖包 只用于 代码开发 的时候，不会打包进 生产环境 js 文件 里面。
99. vue-cli 中常用到的加载器
1.安装 sass:
2.安装 axios:
3.安装 mock:
4.安装 lib-flexible: --实现移动端自适应
5.安装 sass-resourses-loader
100.vue.cli 中怎样使用自定义的组件？有遇到过哪些问题吗？
第一步：在 components 目录新建你的组件文件（如：indexPage.vue），script 一定要 export default {}
第二步：在需要用的页面（组件）中导入：import indexPage from '@/components/indexPage.vue'
第三步：注入到 vue 的子组件的 components 属性上面,components:{indexPage}
第四步：在 template 视图 view 中使用
遇到的问题： 例如有 indexPage 命名，使用的时候则 index-page
101. vue-cli 提供的几种脚手架模板
102. vue-cli 开发环境使用全局常量
103. vue-cli 生产环境使用全局常量
104. vue-cli 中自定义指令的使用
105. vue 是如何对数组方法进行变异的？例如 push、pop、splice 等方法
106. vue 组件之间的通信种类
1. 父组件向子组件通信
2. 子组件向父组件通信
3. 隔代组件间通信
4. 兄弟组件间通信
107. vue 是如何对数组方法进行变异的？例如 push、pop、splice 等方法
108. 谈一谈 nextTick 的原理
109. Vue 中的 computed 是如何实现的
110. vue 如何优化首页的加载速度？vue 首页白屏是什么问题引起的？如何解决呢？
vue 如何优化首页的加载速度？
- 路由懒加载
- ui 框架按需加载
- gzip 压缩

vue 首页白屏是什么问题引起的？

- 第一种，打包后文件引用路径不对，导致找不到文件报错白屏

解决办法：修改一下 config 下面的 index.js 中 bulid 模块导出的路径。因为 index.html 里边的内容都是通过 script 标签引入的，而你的路径不对，打开肯定是空白的。先看一下默认的路径。

- 第二种，由于把路由模式 mode 设置影响

解决方法：路由里边 router/index.js 路由配置里边默认模式是 hash，如果你改成了 history 模式的话，打开也会是一片空白。所以改为 hash 或者直接把模式配置删除，让它默认的就行 。如果非要使用 history 模式的话，需要你在服务端加一个覆盖所有的情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回一个 index.html，这个页面就是你 app 依赖页面。

所以只要删除 mode 或者把 mode 改成 hash 就 OK 了。

- 第三种，项目中使用了 es6 的语法，一些浏览器不支持 es6，造成编译错误不能解析而造成白屏

解决方法：

安装 npm install --save-dev babel-preset-es2015

安装 npm install --save-dev babel-preset-stage-3

在项目根目录创建一个.babelrc 文件 里面内容 最基本配置是：

```
{
    // 此项指明，转码的规则
    "presets": [
        // env 项是借助插件 babel-preset-env，下面这个配置说的是 babel 对 es6,es7,es8 进行转码，并且设置 amd,commonjs 这样的模块化文件，不进行转码
        ["env", {
            "modules": false
        }],
        // 下面这个是不同阶段出现的 es 语法，包含不同的转码插件
        "stage-2"
    ],
    // 下面这个选项是引用插件来处理代码的转换，transform-runtime 用来处理全局函数和优化 babel 编译
    "plugins": ["transform-runtime"],
    // 下面指的是在生成的文件中，不产生注释
    "comments": false,
    // 下面这段是在特定的环境中所执行的转码规则，当环境变量是下面的 test 就会覆盖上面的设置
    "env": {
        // test 是提前设置的环境变量，如果没有设置 BABEL_ENV 则使用 NODE_ENV，如果都没有设置默认就是 development
        "test": {
            "presets": ["env", "stage-2"],
            // instanbul 是一个用来测试转码后代码的工具
            "plugins": ["istanbul"]
        }
    }
}
```

111.Vue 的父组件和子组件生命周期钩子执行顺序是什么

答案：

- 加载渲染过程
  - 父 beforeCreate->父 created->父 beforeMount->子 beforeCreate->子 created->子 beforeMount->子 mounted->父 mounted
- 子组件更新过程
  - 父 beforeUpdate->子 beforeUpdate->子 updated->父 updated
- 父组件更新过程
  - 父 beforeUpdate->父 updated
- 销毁过程
  - 父 beforeDestroy->子 beforeDestroy->子 destroyed->父 destroyed

112.在 Vue 中，子组件为何不可以修改父组件传递的 Prop，如果修改了，Vue 是如何监控到属性的修改并给出警告的。

113.实现通信方式

答案：

方式 1: props

1) 通过一般属性实现父向子通信
2) 通过函数属性实现子向父通信
3) 缺点：隔代组件和兄弟组件间通信比较麻烦

方式 2：vue 自定义事件
1) vue 内置实现，可以代替函数类型的 props
   a. 绑定监听：<MyComp @eventName="callback"
   b. 触发(分发)事件：this.$emit("eventName", data)
2) 缺点：只适合于子向父通信

方式 3：消息订阅与发布
1) 需要引入消息订阅与发布的实现库，如：pubsub-js
   a. 订阅消息：PubSub.subscribe('msg', (msg, data)=>{})
   b. 发布消息：PubSub.publish('msg' , data)
2) 优点：此方式可用于任意关系组件间通信

方式 4：vuex
1) 是什么：vuex 是 vue 官方提供的集中式管理 vue 多组件共享状态数据的 vue 插件
2) 优点：对组件间关系没有限制，且相比于 pubsub 库管理更集中，更方便

方式 5：slot
1) 是什么：专门用来实现父向子传递带数据的标签
   a. 子组件
   b. 父组件
2) 注意：通信的标签模板是在父组件中解析好后再传递给子组件的

114. 说说 Vue 的 MVVM 实现原理

答案：

理解

1) Vue 作为 MVVM 模式的实现库的 2 种技术
a. 模板解析
b. 数据绑定

2) 模板解析：实现初始化显示
a. 解析大括号表达式
b. 解析指令

3) 数据绑定：实现更新显示
a. 通过数据劫持实现

原理结构图