JS_Even_JS RP 1.1k 发布于 前海拾贝
1 天前发布

# Vue常见面试题精讲【持续更新】

## 1. v-if和v-show指令有什么区别？

> v-if 是**条件渲染**指令，控制的是**组件是否创建(渲染)**，值为true则渲染该组件，值为false则不渲染该组件，对应Html元素则不会存在于浏览器的html文档中，即**打开浏览器调试工具找不到该组件对应的渲染结果**。
> v-show控制的是**组件是否可见**，并且是**通过css样式的display属性来控制组件的显示与隐藏**，所以其值无论是true还是false，对应Html元素都会存在于浏览器的html文档中，即**打开浏览器调试工具都能够找到该组件对应的渲染结果**，只不过值为false的时候，**会在该组件上添加style="display: none;"**；

## 2. v-for 与 v-if 的优先级?

可参考v-for与v-if的优先级

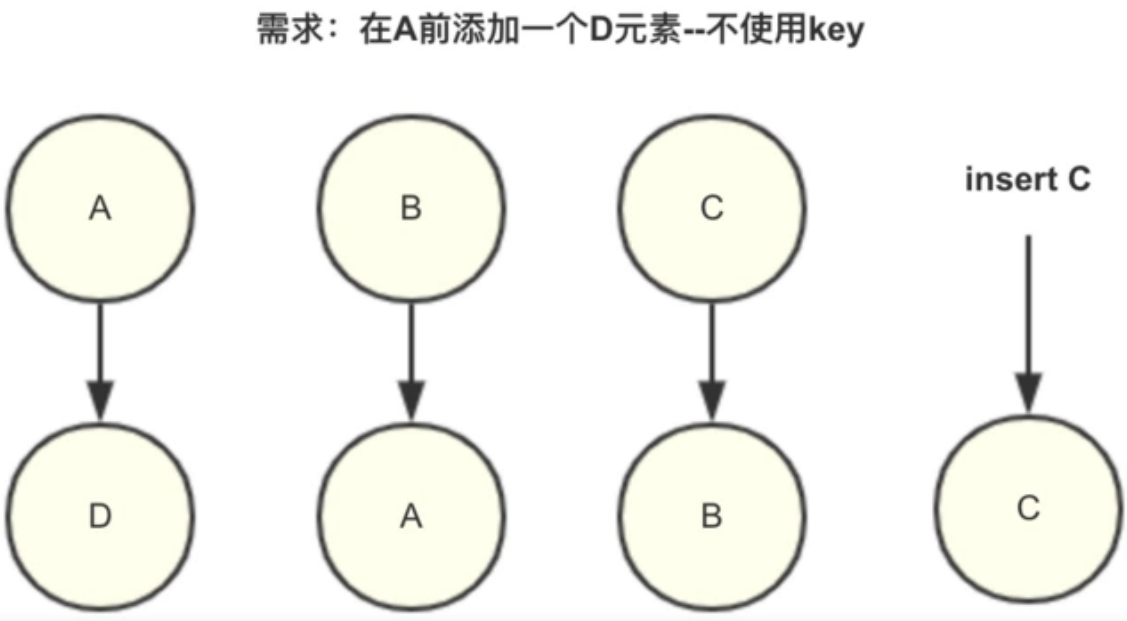
> **v-for比v-if优先级更高**，所以不建议v-for和v-if一起使用，如果v-for和v-if同时使用，那么数据发生变化时，v-for首先会进行遍历，然后通过v-if进行判断，这样**v-for和v-if都会同时执行一遍**，对性能和展现不友好。所以**vue建议用计算属性进行替代，返回过滤后的列表再进行遍历。**

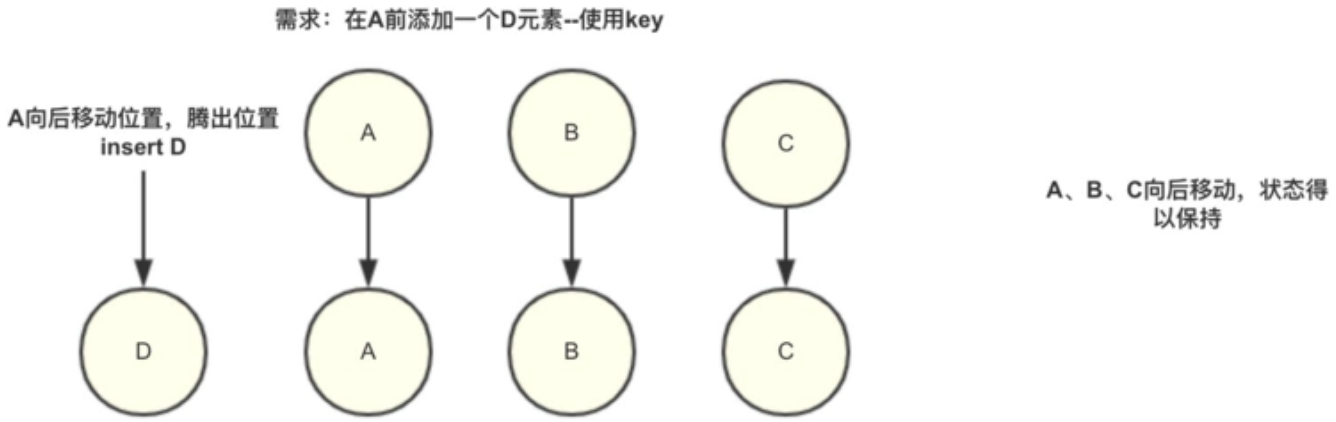
## 3. v-for循环中key有什么作用?

> key的作用主要就是为了**性能优化**，key让组件具有了唯一性，能让diff算法更快的找到需要更新的组件dom，在绑定key的时候，**最好是一个唯一的值**，**如item.id 而不能是简单的index**，如果不使用唯一key，那么在有状态组件中会出现渲染错误。因为它默认用**就地复用**策略，如果数据项的顺序被改变，那么**vue将不是移动DOM元素来匹配数据项的改变**，而是简单复用此处每个元素，不会重新排列元素的位置。如果是使用 key，**它会基于key重新排列元素顺序**，并且会移除 key 不存在的元素。
> 简单说就是，**不使用key就会原地复用，使用key就会对元素位置进行重新排列，能够关联数据状态。**

```
<template>
  <div class="home">
    <input type="text" v-model="name">
    <button @click="add">添加</button>
    <ul>
      <li v-for="(item, index) in list" :key="item.id"><!--注意，这里不能使用:key="index"-->
        <input type="checkbox">{{item.name}}
      </li>
    </ul>
  </div>
</template>
<script>
export default {
    data: () => {
        return {
            list: [
              { id: 0, name: 'A' },
              { id: 1, name: 'B' },
              { id: 2, name: 'C' }
            ]
        }
    },
     methods: {
```

> 如果v-for上不加key，那么当勾选A前面的复选框后，再点击添加按钮，D会添加到A的前面，由于原地复用原则，不会进行位置的移动，所以第一个位置的复选框是勾选状态会被继承到D上，即D会变成勾选状态而A将失去勾选状态，这个显然与原来状态不符；如果v-for上加上:key="item.id"，那么D添加到A前面之后，A、B、C都会向后移动，然后再将D插入到A的前面，所以插入D后，A仍然保持勾选状态。






## 4. vue路由传参数的三种方式？

① **query**: 直接输入url地址，url地址上带上查询参数，如: http://localhost:8080/home?foo=1 或者 通过路由对象$router调用push()方法进行传参this.$router.push({path:"/home",query:{"foo":"1"}});然后通过this.$route.query.foo进行获取传递过来的参数

② **params**: 通过路由对象$router调用push()方法进行传参this.$router.push({name:"home", params: {foo: 1}});然后通过this.$route.params.foo获取传递过来的参数

③ **动态路由传参**: 路由path位置为/home/:foo，然后输入url，如: http://localhost:8080/home/1 然后通过this.$route.params.foo获取传递过来的参数

## 5. v-on 常用修饰符？

> **.stop** 该修饰符将**阻止事件向上冒泡。**同理于调用 event.stopPropagation() 方法，即如果当前元素添加了.stop修饰符，那么当点击该元素时候，click事件不会冒泡到其父元素上，即**父元素不会触发click事件。**
> **.prevent** 该修饰符会**阻止当前事件的默认行为。**同理于调用 event.preventDefault() 方法，即如果<a href="http://www.baidu.com" **@click.stop="show"**>连接，点击后默认会跳转到百度，但是添加上.stop修饰符之后，就**不会跳转到百度了，而是执行show()方法了。**
> **.self** 该指令**只有当事件是从事件绑定的元素本身触发时才触发回调**，即**冒泡事件到达该元素上时，并不会触发事件**，但是**其不影响事件继续向上冒泡，其父元素仍然会触发冒泡事件**
> **.native** 就是**给自定义组件的根元素添加一个原生事件**，所以其通常用在自定义组件上，**如果给普通的HTML元素添加.native修饰符，那么该HTML元素将无法监听到该事件了。**
> **.capture** 就是**让事件监听变成捕获**，默认为冒泡，**通常用于修饰父元素**，如果给父元素添加

> click事件。
> **.once** 该修饰符表示绑定的事件**只会被触发一次**。

## 6. 什么是动态组件?

> 简单的说，动态组件就是**将几个组件放在一个挂载点下**，这个挂载点就是**<component>标签**，其需要**绑定is属性**，属性值为**父组件中的变量，变量对应的值为要挂载的组件的组件名**，然后根据父组件里某个变量来动态显示哪个，也可以都不显示，如:

```
<template>
  <div class="home">
      <component :is="currentComponent"></component>
  </div>
</template>
<script>
import Tab0 from "@/components/Tab0.vue";
import Tab1 from "@/components/Tab1.vue";
export default {
    data: () => {
        return {
            currentIndex: 0 // 通过改变currentIndex改变要挂载的组件名
        }
    },
    components: {
        "tab-0": Tab0,
        "tab-1": Tab1
    }
    currentComponent() { // 动态计算要挂载的组件的组件名
      return `tab-${this.currentIndex}`; // "tab-0" 、"tab-1"
    }
}
</script>
```

> **可以将动态组件放到<keep-alive>组件内对动态组件进行缓存**，这样动态组件进行切换的时候，就不会每次都重新创建了。

```
<template>
  <div class="home">
     <keep-alive>
          <component :is="currentComponent"></component>
     </keep-alive>
  </div>
</template>
```

## 7. 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**>

//foo.vue

```
<style scoped>
<!--其中的css样式只在foo.vue组件起作用-->
</style>
```

## 8. active-class是哪个组件的属性？

> vue-router模块的**router-link**组件

## 9. Vue子组件调用父组件的方法?

> 子组件需要调用父组件的方法，那么可以通过**this.$parent**获取到父组件实例，然后就可以调用父组件上的方法了。还有一种方法就是，**子组件向父组件emit一个事件**，**父组件在子组件上监听到该事件**后，就可以调用父组件上的方法了。

## 10. vue中的 ref 是什么？

> **ref 被用来给元素或子组件注册引用信息**，引用信息将会注册在父组件的 $refs 对象上，即**类似于给组件或者DOM元素上添加一个标识id，然后通过这个标识id拿到对应的DOM元素或组件实例**，如果在普通的 DOM 元素上使用，引用指向的就是 DOM 元素；如果用在子组件上，引用就指向组件实例。

## 11. $route和$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。
> $router是"**路由实例**"对象包括了路由的跳转方法，钩子函数等。

> **全局定义**：调用Vue的component()方法创建，**Vue.component**(组件名, {template: 模板字符串})
> **局部定义**：在**创建Vue实例时传递的options对象中的components对象中进行定义**，components: {组件名: {template: 模板字符串}}
> **单文件组件**：在**.vue文件**中定义，包含template，script，style三部分。

## 13. Vue-cli的src文件夹中有哪些文件？

> assets文件夹是放**静态资源**；
> components是放**组件**；
> router是定义**路由相关的配置**;
> view**视图**；
> app.vue是一个**应用主组件**；
> main.js是**入口文件**

## 14. 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。

## 15. vue等单页面应用及其优缺点？

> 优点：Vue 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件，核心是一个响应的数据绑定系统。MVVM、数据驱动、组件化、轻量、简洁、高效、快速、模块友好。
> 缺点：不支持低版本的浏览器，最低只支持到IE9；不利于SEO的优化（如果要支持SEO，建议通过服务端来进行渲染组件）；第一次加载首页耗时相对长一些；不可以使用浏览器的导航按钮需要自行实

## 16. 路由的跳转方式？

> ① <router-link to='/home'> router-link标签会渲染为<a>标签，点击该a标签即可跳转到/home路由;
> ② 另一种是通过js跳转，即通过路由对象this.$router的push()方法进行跳转， 比如router.push('/home')

## 17. 计算属性（computed）、方法（methods）和侦听属性（watch）的区别与使用场景?

> methods VS 计算属性
> 我们可以将同一函数定义为一个 method 而不是一个计算属性。对于最终的结果，两种方式确实是相同的。
> 然而，不同的是计算属性是基于它们的依赖进行缓存的。计算属性只有在它的相关依赖发生改变时才会重新求值。这就意味着只要 message 还没有发生改变，多次访问 reversedMessage 计算属性会立即返回之前的计算结果，而不必再次执行函数。
> 相比而言，只要发生重新渲染，method 调用总会执行该函数。总之，重新计算开销很大的话请选计算属性，不希望有缓存的请选methods。
> watch VS 计算属性
> 当你在模板内使用了复杂逻辑的表达式时，你应当使用计算属性。
> 侦听属性是一个对象，键是需要观察的表达式，值是对应回调函数。值也可以是方法名，或者包含选项的对象。
> 当你有一些数据需要随着其它数据变动而变动时，或者当需要在数据变化时执行异步或开销较大的操作时，你可以使用 watch。

## 18. axios是什么？怎么使用？

> axios是**基于Promise的**，用于浏览器和nodeJS的http客户端，主要作用就是**向后台发送请求**。其存在许多优点:

- **支持Promise**
- **支持并发请求**
- **提供拦截器**
- **浏览器支持防止csrf(跨站请求伪造)**

## 19. axios、fetch、ajax(jquery)的区别？

> axios和fetch是基于Promise的，ajax是基于callback的形式。fetch脱离了xhr，是新的语法，默认是不传cookie的，**监听不到请求进度**。

## 20. vuex是什么？哪种功能场景使用它？

> vuex是一个专门为vue构建的状态机管理机制，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**解决组件间数据共享的问题**，其实就是**采用类似全局对象的形式来管理所有组件的公共数据**，其强调的是集中式管理，主要是为了便于维护、组件解耦，适合大型项目，有多个视图组件共同依赖一个状态的情况下使用，比如商城系统、外卖系统。
> vuex的核心: **state**、**getters**、**mutations**、**actions**、**modules**。

## 21. 说出4个vue当中常用的指令及其用法？

- v-if: 这是一个**条件渲染**指令，代表存在和销毁，用于控制组件的创建与否;
- v-bind: 这是一个**绑定**指令，用于绑定属性，可简写为冒号;
- v-on: 这是一个**监听**指令，用于监听事件，可简写为@;
- v-for: 这是一个**循环**指令，用于遍历数组;

## 22. 导航钩子有哪些？它们有哪些参数？

> 导航主要有三类钩子: **全局级路由钩子**、**路由级路由钩子**、**组件级路由钩子**。主要参数有to(目标路由对象)、from(当前路由对象)、next(是一个函数，用于控制是否放行，即是否能通过当前守卫)。

- **全局级路由钩子**: beforeEach和afterEach，**每次路由跳转全局路由钩子都会执行**，beforeEach(to, from, next)钩子有三个参数，但是**afterEach是已经跳转结束了**，所以**其没有next参数**，afterEach(to, from)，**全局路由钩子由router对象调用。**
- **路由级路由钩子**: 路由级钩子只有一个即**beforeEnter**，其是**在配置路由表的时候配置**，也是有to、from、next三个参数，**只有进入该路由的时候才会执行**，如果是**动态路由之间的切换**，那么则不会触发beforeEnter钩子，**因为是同一个路由，只是参数不一样。**
- **组件级路由钩子**: beforeRouteEnter(路由进入该组件的时候执行)、beforeRouteUpdate(动态路由切换时候执行)、beforeRouteLeave(路由离开当前组件的时候执行)，需要注意的是，**路由钩子是优先**

行顺序: **beforeRouteLeave --> beforeEach --> beforeEnter --> beforeRouteEnter --> afterEach --> beforeCreate --> created --> mounted，beforeRouteUpdate只有动态路由切换的时候才会执行，即/user/1切换到/user/2才会执行。**

## 23. v-model是什么？

> v-model主要用于数据的双向绑定，其内部主要完成了两个操作: **通过v-bind绑定value属性值**和**监听input事件，并更新数据**，如:

```
<template>
    <!-- <input v-model="msg"/>{{msg}} -->
    <input v-bind:value="msg" @input="msg=$event.target.value"/>{{msg}} <!--二者是等价的-->
</template>
<script>
    export default {
        data: () => {
            return {
                msg: "hello"
            }
        }
    }
</script>
```

## 24. 什么是路由懒加载？其原理是什么？

> 所谓路由懒加载，即在项目打包的时候，项目中通常会有多个路由，如果将所有路由对应的组件都打包到一个文件中，那么最终打包的文件就会变得非常大，会影响页面的加载性能，如果我们能把不同路由对应的组件**分割**成不同的代码块，然后**当路由被访问的时候才异步加载出对应组件**，这样就会变得更加高效。
> 所以其原理就是利用了**webpack的代码分割(按需加载)**机制和**vue的异步组件**功能，代码被分割后就会变成一个单独的文件，所以**路由被访问的时候需要向服务器发起请求加载该组件**，这是一个**异步的过程**，所以需要使用到vue的异步组件机制。

**异步组件？**

> 异步组件，就是在注册组件的时候，**传入一个函数**，然后这个函数返回一个Promise对象，**resolve的值为这个组件对象**，如:

```
export default new Router({
    routes: [
        {
          path: '/about',
          name: 'about',
          component: () => { // 注册为一个异步组件
              const About = require("./views/About.vue");
              return Promise.resolve(About);
          }
        }
    ]
});
```

或者**在注册异步组件的时候传入resolve和reject**，如:

```
export default new Router({
    routes: [
        {
          path: '/about',
          name: 'about',
          component: (resolve, reject) => { // 注册为一个异步组件
              const About = require("./views/About.vue");
              resolve(About);
          }
        }
    ]
});
```

**webpack提供的import()函数会返回一个Promise对象，并且会对引入的组件进行代码分割**，所以可以通过import()同时实现代码分割和组件异步加载，即路由懒加载，如:

```
export default new Router({
    routes: [
        {
          path: '/about',
          name: 'about',
          component: () => import('./views/About.vue') // 等价于注册异步组件并返回一个Promise对象
        }
    ]
});
```

## 25. 用过插槽吗？用过哪些类型的插槽？

> 插槽其实就是**组件中提供的占位符**，所以**插槽占位符在组件中使用**，插槽有三种: **匿名插槽、具名插槽、作用域插槽。**

- **匿名插槽**：即没有名字的插槽，即**<slot></slot>**，使用组件的时候会将组件中的innerHTML插入到<slot></slot>位置上，相当于动态向组件内部传递数据。

```
// About.vue组件
<template>
  <div class="hello">
    <slot></slot>
  </div>
</template>

// 使用About组件
<About>
    <h1>hello world</h1><!--该内容会插入到上面<slot></slot>位置上，即替换掉<slot></slot>-->
</About>
```

- **具名插槽**: 即有名字的插槽，**需要在<slot>标签上添加一个name属性，指定<slot>的名称**，即<slot name="header"></slot>，同时使用组件的时候需要给其中的innerHTML添加slot属性，属性值为<slot>的name属性值，如:

```
// About.vue组件
<template>
  <div class="hello">
    <slot name="header"></slot> <!--定义slot的名称为header-->
  </div>
</template>

// 使用About组件
<About>
    <h1 slot="header">header</h1> <!--该内容会被插入到名称为header的slot上-->
</About>
```

- **作用域插槽**: 作用域插槽可以理解为**组件向外输出数据**，我们可以在组件的<slot>标签上添加上一些属性，然后其中的属性值可以传到组件外使用，会将slot标签上的所以属性合并到一个对象对外输出，组件外通过slot-scope指定一个变量名来接收这个对象，如:

```
// About.vue组件
<template>
  <div class="hello">
    <h1 slot="footer" slot-scope="innerData">{{innerData.msg}} {{innerData.foo}}</h1><!--指定
  </div>
```

```
// 使用About组件
<About>
    <slot name="footer" msg="haha" foo="foo"></slot><!--会将其中的属性合并成一个对象对外输出-->
</About>
```

## 26. 什么是vue-loader？

> vue-loader就是**.vue组件的加载器**，**可以将.vue组件转换为javascript模块**，及**动态渲染一些数据**，同时vue-loader还对.vue组件中的三个标签都进行了相应的优化。**<template>标签中可以使用src属性引入一个组件**，引入的组件可以直接使用当前组件中的数据，**<script>标签中可以直接使用ES6语法**，**<style>标签可以默认使用sass并且支持scoped作用域选择**。如:

// foo.vue

```
<template>
    <h1>{{msg}}</h1><!--可以直接使用hello.vue中的数据-->
</template>
```

// hello.vue

```
<template src="./foo.vue"> <!--直接通过src引入foo.vue-->

</template>

<script>
export default {
  name: 'HelloWorld',
  data: () => {
    return {
      msg: "to foo.vue"
    }
  }
}
</script>
<style scoped>

</style>
```

## 27. Vue中keep-alive 的作用以及用法？

> keep-alive是vue中一个内置的组件，主要用于**缓存组件**，其会在组件created的时候，将需要缓存的组件放到缓存中，然后再render的时候再根据name进行取出。<keep-alive>主要配合路由进行使用，在配置路由的时候添加上meta元数据对象，里面添加上keepAlive属性，表示是否缓存该组件，然后将<router-view>放到<keep-alive>中，router-view通过v-if指令，从路由配置上的meta对象中取出keepAlive的值进行判断是否需要缓存，如:

```
<template>
  <div id="app">
      <keep-alive>
        <router-view v-if="$route.meta.keepAlive"/> <!--这个是需要缓存的-->
      </keep-alive>
        <router-view v-if="!$route.meta.keepAlive"></router-view><!--这个是不需要缓存的-->
  </div>
</template>


// 路由配置


export default new Router({
    routes: [
        {
          path: '/',
          name: 'home',
          component: Home,
          meta: {
            keepAlive: true
          }
        },
        {
          path: '/about',
          name: 'about',
          component: About,
          meta: {
            keepAlive: false
          }
        }
    ]
});
```

> **组件缓存后就不会执行组件的beforeCreate、created和beforeMount、mounted钩子了**，所以其提供了actived和deactived钩子，**actived钩子主要用于承担原来created钩子中获取数据的任务。**

## 28. Vue 组件中 data 为什么必须是函数？

> 因为**组件是可以多次复用的**，也就是说**会有多个组件实例同时存在**，同时由于**对象是引用数据类型**，如果所有组件实例都共用一个data数据对象，那么一个组件对data数据对象进行修改，那么**其他组件实例也会受到影响**，所以需要使用函数返回data对象的独立拷贝，**使得每个组件实例都会拥有自己的data数据对象**，相互之间独立，不会互相受影响，便于组件维护。

## 29. Vue组件复用时，vue-router如何响应路由参数的变化？

> 当使用路由参数时，例如从 /user/lucy 导航到 /user/lily，**原来的组件实例会被复用**。因为两个路由都渲染同个组件，比起销毁再创建，复用则显得更加高效。不过，这也意味着**组件的生命周期钩子不会再被调用**，复用组件时，想对路由参数的变化作出响应的话，有两种方式:

- 监听$route对象数据变化

```
export default {
  watch: {
    '$route': (to, from) =>{
      console.log("route change");// 对路由变化作出响应...

    }
  }
}
```

- 通过beforeRouteUpdate路由钩子

```
export default {
  beforeRouteUpdate(to, from ,next) {
    console.log("beforeRouteUpdate hook run.");
    next();
  }
}
```

## 30. 简述Vue的响应式原理?

> 当一个Vue实例创建时，vue会遍历data选项的属性，用 Object.defineProperty 将它们转为getter/setter并且在内部追踪相关依赖，在属性被访问和修改时通知变化。
> 每个组件实例都有相应的 watcher 程序实例，它会在组件渲染的过程中把属性记录为依赖，之后当依赖项的 setter 被调用时，会通知 watcher 重新计算，从而致使它关联的组件得以更新。