Web前端面试涨薪名企培养计划

Vue专题

讲师：村长

16年IT从业经验

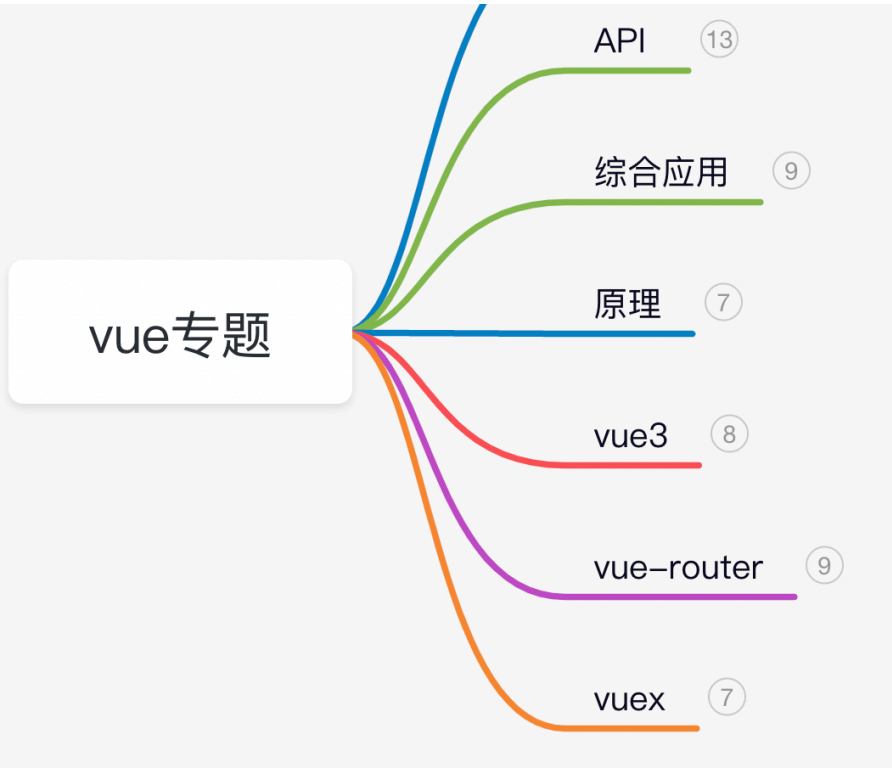
开课吧前端学科负责人


村长微信


村长B站

内容安排

https://www.processon.com/view/link/620c4de01efad406e72b891f



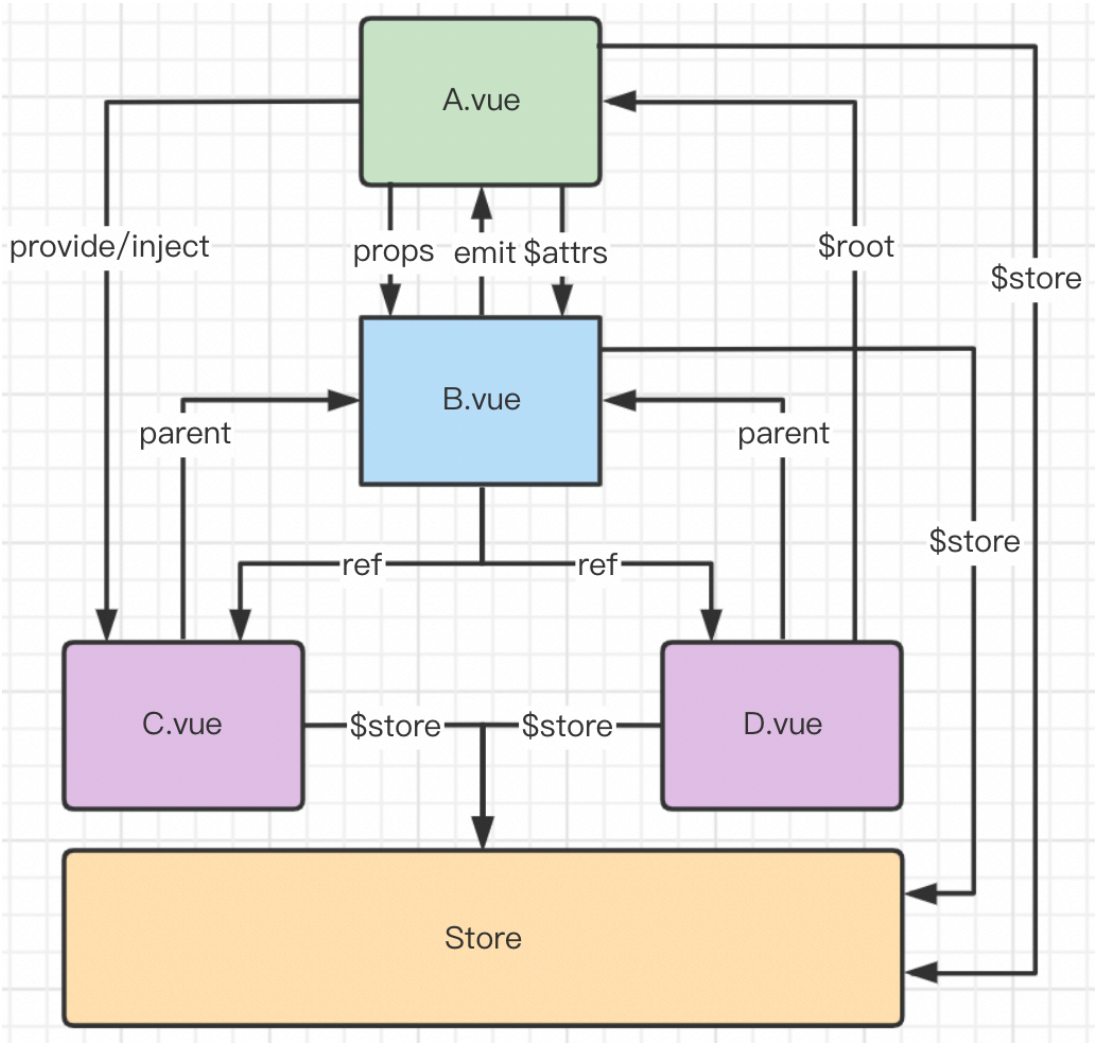
01-Vue组件之间通信方式有哪些

vue是组件化开发框架，所以对于vue应用来说组件间的数据通信非常重要。

此题主要考查大家vue基本功，对于vue基础api运用熟练度。

另外一些边界知识如provide/inject/$attrs则提现了面试者的知识广度。

组件传参的各种方式



思路分析：

1. 总述知道的所有方式
2. 按组件关系阐述使用场景

回答范例：

1. 组件通信常用方式有以下8种：

- props
- $emit/~~$on~~
- ~~$children~~/$parent
- $attrs/~~$listeners~~
- ref
- $root
- eventbus
- vuex

> 注意vue3中废弃的几个API

> https://v3-migration.vuejs.org/breaking-changes/children.html
>
> https://v3-migration.vuejs.org/breaking-changes/listeners-removed.html
>
> https://v3-migration.vuejs.org/breaking-changes/events-api.html#overview

---

2. 根据组件之间关系讨论组件通信最为清晰有效

- 父子组件
  - `props` / `$emit` / `$parent` / `ref` / `$attrs`
- 兄弟组件
  - `$parent` / `$root` / `eventbus` / `vuex`
- 跨层级关系
  - `eventbus` / `vuex` / `provide` + `inject`

02-v-if和v-for哪个优先级更高?

分析:

此题考查常识，文档中曾有详细说明v2|v3；也是一个很好的实践题目，项目中经常会遇到，能够看出面试者api熟悉程度和应用能力。

思路分析:

1. 先给出结论
2. 为什么是这样的，说出细节
3. 哪些场景可能导致我们这样做，该怎么处理
4. 总结，拔高

回答范例:

1. 实践中**不应该把v-for和v-if放一起**
2. 在**vue2中**，**v-for的优先级是高于v-if**，把它们放在一起，输出的渲染函数中可以看出会先执行循环再判断条件，哪怕我们只渲染列表中一小部分元素，也得在每次重渲染的时候遍历整个列表，这会比较浪费；另外需要注意的是在**vue3中则完全相反，v-if的优先级高于v-for**，所以v-if执行时，它调用的变量还不存在，就会导致异常
3. 通常有两种情况下导致我们这样做:
   - 为了**过滤列表中的项目** (比如 `v-for="user in users" v-if="user.isActive"`)。此时定义一个计

算属性 (比如 `activeUsers`)，让其返回过滤后的列表即可（比如 `users.filter(u=>u.isActive)`）。
    - 为了**避免渲染本应该被隐藏的列表** (比如 `v-for="user in users" v-if="shouldShowUsers"`)。此时把 `v-if` 移动至容器元素上 (比如 `ul`、`ol`)或者外面包一层`template`即可。
4. 文档中明确指出**永远不要把 `v-if` 和 `v-for` 同时用在同一个元素上**，显然这是一个重要的注意事项。
5. 源码里面关于代码生成的部分，能够清晰的看到是先处理v-if还是v-for，顺序上vue2和vue3正好相反，因此产生了一些症状的不同，但是不管怎样都是不能把它们写在一起的。

知其所以然：

做个测试，test.html
两者同级时，渲染函数如下：

```
ƒ anonymous(
) {
with(this){return _c('div',{attrs:{"id":"app"}},_l((items),function(item){return
(item.isActive)?_c('div',{key:item.id},[_v("\n      "+_s(item.name)+"\n
")]):_e()}),0)}
}
```

做个测试，test-v3.html

```
Uncaught TypeError: Cannot read properties of undefined (reading 'isActive')
    at Proxy.render (eval at compileToFunction (vue@3:15587:23), <anonymous>:11:13)
    at renderComponentRoot (vue@3:2415:46)
    at ReactiveEffect.componentUpdateFn [as fn] (vue@3:6473:59)
    at ReactiveEffect.run (vue@3:590:27)
```

源码中找答案

v2：https://github1s.com/vuejs/vue/blob/HEAD/src/compiler/codegen/index.js#L65-L66

v3：https://github1s.com/vuejs/core/blob/HEAD/packages/compiler-core/src/codegen.ts#L586-L587

03-简述 Vue 的生命周期以及每个阶段做的事

必问题目，考查vue基础知识。

思路

1. 给出概念
2. 列举生命周期各阶段
3. 阐述整体流程
4. 结合实践

5. 扩展：vue3变化

回答范例

1.每个Vue组件实例被创建后都会经过一系列初始化步骤，比如，它需要数据观测，模板编译，挂载实例到dom上，以及数据变化时更新dom。这个过程中会运行叫做生命周期钩子的函数，以便用户在特定阶段有机会添加他们自己的代码。

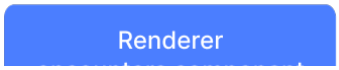
2.Vue生命周期总共可以分为8个阶段：**创建前后, 载入前后, 更新前后, 销毁前后**，以及一些特殊场景的生命周期。vue3中新增了三个用于调试和服务端渲染场景。

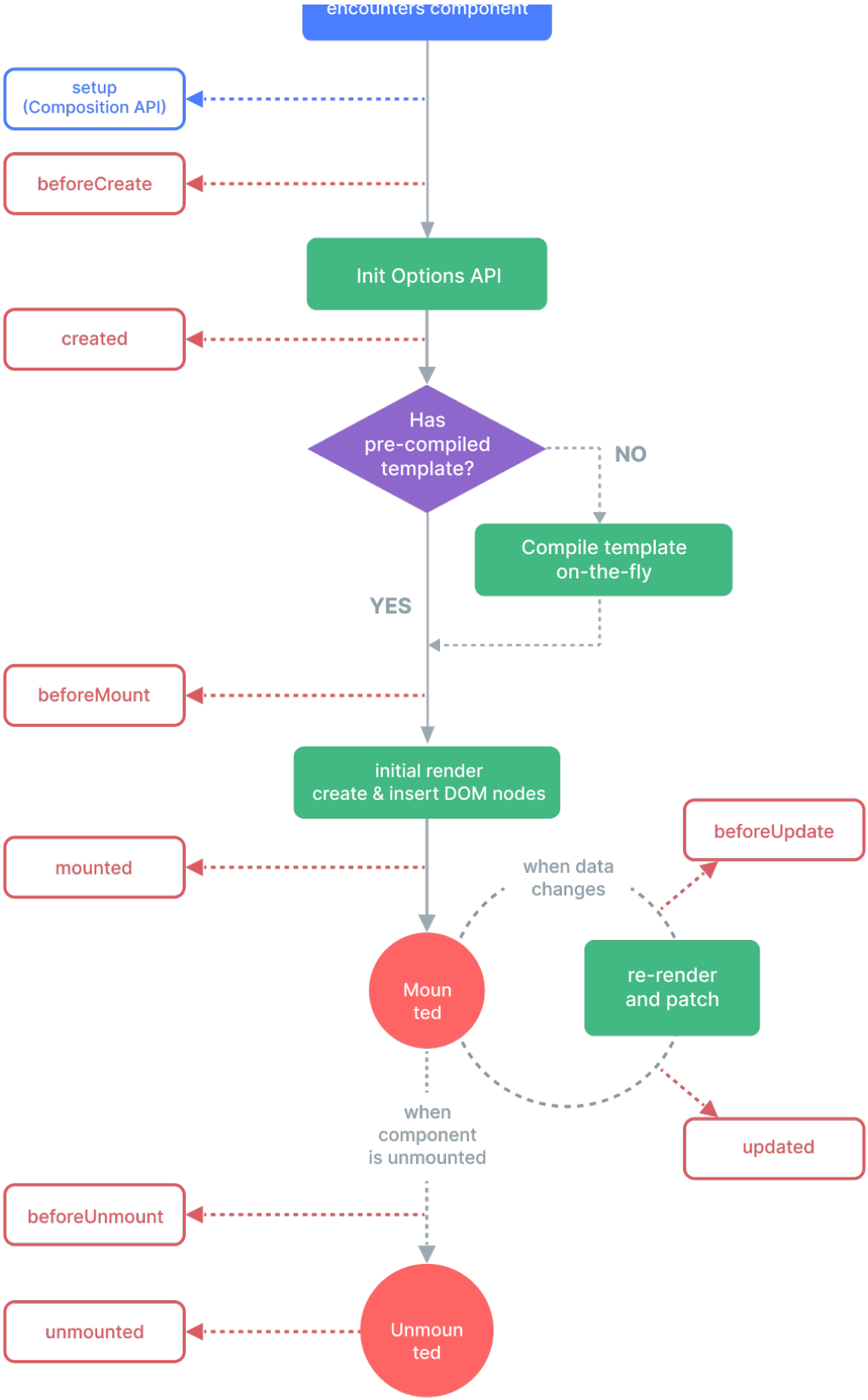
生命周期v2	生命周期v3	描述
beforeCreate	beforeCreate	组件实例被创建之初
created	created	组件实例已经完全创建
beforeMount	beforeMount	组件挂载之前
mounted	mounted	组件挂载到实例上去之后
beforeUpdate	beforeUpdate	组件数据发生变化，更新之前
updated	updated	数据数据更新之后
beforeDestroy	**beforeUnmounted**	组件实例销毁之前
destroyed	**unmounted**	组件实例销毁之后

生命周期v2	生命周期v3	描述
activated	activated	keep-alive 缓存的组件激活时
deactivated	deactivated	keep-alive 缓存的组件停用时调用
errorCaptured	errorCaptured	捕获一个来自子孙组件的错误时被调用
-	**renderTracked**	调试钩子，响应式依赖被收集时调用
-	**renderTriggered**	调试钩子，响应式依赖被触发时调用
-	**serverPrefetch**	ssr only，组件实例在服务器上被渲染前调用

---

3. `Vue` 生命周期流程图：




encounters component
setup
(Composition API)
beforeCreate
Init Options API
created
Has
pre-compiled
template?
NO
Compile template
on-the-fly
YES
beforeMount
initial render
create & insert DOM nodes
mounted
beforeUpdate
when data
changes
Moun
ted
re-render
and patch
when
component
is unmounted
updated
beforeUnmount
unmounted
Unmoun
ted

4.结合实践：

beforeCreate：通常用于插件开发中执行一些初始化任务

created：组件初始化完毕，可以访问各种数据，获取接口数据等

mounted：dom已创建，可用于获取访问数据和dom元素；访问子组件等。

beforeUpdate：此时`view`层还未更新，可用于获取更新前各种状态

updated：完成`view`层的更新，更新后，所有状态已是最新

beforeunmounted：实例被销毁前调用，可用于一些定时器或订阅的取消

unmounted：销毁一个实例。可清理它与其它实例的连接，解绑它的全部指令及事件监听器

可能的追问

1. setup和created谁先执行?
2. setup中为什么没有beforeCreate和created?

知其所以然

vue3中生命周期的派发时刻：

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/componentOptions.ts#L554-L555

vue2中声明周期的派发时刻：

https://github1s.com/vuejs/vue/blob/HEAD/src/core/instance/init.js#L55-L56

04-能说一说双向绑定使用和原理吗?

题目分析：

双向绑定是`vue`的特色之一，开发中必然会用到的知识点，然而此题还问了实现原理，升级为深度考查。

思路分析：

1. 给出双绑定义
2. 双绑带来的好处
3. 在哪使用双绑
4. 使用方式、使用细节、vue3变化
5. 原理实现描述

回答范例：

1. vue中双向绑定是一个指令`v-model`，可以绑定一个响应式数据到视图，同时视图中变化能改变该值。
2. `v-model`是语法糖，默认情况下相当于`:value`和`@input`。使用`v-model`可以减少大量繁琐的事件处理代码，提高开发效率。
3. 通常在表单项上使用`v-model`，还可以在自定义组件上使用，表示某个值的输入和输出控制。
4. 通过`<input v-model="xxx">`的方式将xxx的值绑定到表单元素value上；对于checkbox，可以使用`true-value`和false-value指定特殊的值，对于radio可以使用value指定特殊的值；对于select可以通过options元素的value设置特殊的值；还可以结合.lazy,.number,.trim对v-mode的行为做进一步限定；`v-model`用在自定义组件上时又会有很大不同，vue3中它类似于`sync`修饰符，最终展开的结果是modelValue属性和update:modelValue事件；vue3中我们甚至可以用参数形式指定多个不同的绑定，例如v-model:foo和v-model:bar，非常强大！
5. `v-model`是一个指令，它的神奇魔法实际上是vue的编译器完成的。我做过测试，包含`v-model`的模板，转换为渲染函数之后，实际上还是是value属性的绑定以及input事件监听，事件回调函数中会做相应变量更新操作。编译器根据表单元素的不同会展开不同的DOM属性和事件对，比如text类型的input和textarea会展开为value和input事件；checkbox和radio类型的input会展开为checked和change事件；select用value作为属性，用change作为事件。

可能的追问：

1. `v-model`和`sync`修饰符有什么区别
2. 自定义组件使用`v-model`如果想要改变事件名或者属性名应该怎么做

知其所以然：

测试代码，test.html

观察输出的渲染函数：

```
// <input type="text" v-model="foo">
_c('input', {
  directives: [{ name: "model", rawName: "v-model", value: (foo), expression: "foo" }],
  attrs: { "type": "text" },
  domProps: { "value": (foo) },
  on: {
    "input": function ($event) {
      if ($event.target.composing) return;
      foo = $event.target.value
    }
  }
})
```

```
// <input type="checkbox" v-model="bar">
_c('input', {
```

```
  directives: [{ name: "model", rawName: "v-model", value: (bar), expression: "bar" }],
  attrs: { "type": "checkbox" },
  domProps: {
    "checked": Array.isArray(bar) ? _i(bar, null) > -1 : (bar)
  },
  on: {
    "change": function ($event) {
      var $$a = bar, $$el = $event.target, $$c = $$el.checked ? (true) : (false);
      if (Array.isArray($$a)) {
        var $$v = null, $$i = _i($$a, $$v);
        if ($$el.checked) { $$i < 0 && (bar = $$a.concat([$$v])) }
        else {
          $$i > -1 && (bar = $$a.slice(0, $$i).concat($$a.slice($$i + 1))) }
      } else {
        bar = $$c
      }
    }
  }
})
```

```
// <select v-model="baz">
//     <option value="vue">vue</option>
//     <option value="react">react</option>
// </select>
_c('select', {
  directives: [{ name: "model", rawName: "v-model", value: (baz), expression: "baz" }],
  on: {
    "change": function ($event) {
      var $$selectedVal = Array.prototype.filter.call(
        $event.target.options,
        function (o) { return o.selected }
      ).map(
        function (o) {
          var val = "_value" in o ? o._value : o.value;
          return val
        }
      );
      baz = $event.target.multiple ? $$selectedVal : $$selectedVal[0]
    }
  }
}, [
  _c('option', { attrs: { "value": "vue" } }, [_v("vue")]), _v(" "),
  _c('option', { attrs: { "value": "react" } }, [_v("react")])
])
```

05-Vue中如何扩展一个组件

此题属于实践题，考察大家对vue常用api使用熟练度，答题时不仅要列出这些解决方案，同时最好说出他们异同。

答题思路：

1. 按照逻辑扩展和内容扩展来列举，
    - 逻辑扩展有：mixins、extends、composition api；
    - 内容扩展有slots；
2. 分别说出他们使用方法、场景差异和问题。
3. 作为扩展，还可以说说vue3中新引入的composition api带来的变化

回答范例：

1. 常见的组件扩展方法有：mixins，slots，extends等
2. 混入mixins是分发 Vue 组件中可复用功能的非常灵活的方式。混入对象可以包含任意组件选项。当组件使用混入对象时，所有混入对象的选项将被混入该组件本身的选项。

```
// 复用代码：它是一个配置对象，选项和组件里面一样
const mymixin = {
   methods: {
      dosomething(){}
   }
}
// 全局混入：将混入对象传入
Vue.mixin(mymixin)

// 局部混入：做数组项设置到mixins选项，仅作用于当前组件
const Comp = {
   mixins: [mymixin]
}
```

---

3. 插槽主要用于vue组件中的内容分发，也可以用于组件扩展。

   子组件Child

```
<div>
  <slot>这个内容会被父组件传递的内容替换</slot>
</div>
```

   父组件Parent

```
<div>
   <Child>来自老爹的内容</Child>
</div>
```

如果要精确分发到不同位置可以使用具名插槽，如果要使用子组件中的数据可以使用作用域插槽。

---

4. 组件选项中还有一个不太常用的选项extends，也可以起到扩展组件的目的

```
// 扩展对象
const myextends = {
   methods: {
      dosomething(){}
   }
}
// 组件扩展：做数组项设置到extends选项，仅作用于当前组件
// 跟混入的不同是它只能扩展单个对象
// 另外如果和混入发生冲突，该选项优先级较高，优先起作用
const Comp = {
   extends: myextends
}
```

---

5. 混入的数据和方法**不能明确判断来源**且可能和当前组件内变量**产生命名冲突**，vue3中引入的composition api，可以很好解决这些问题，利用独立出来的响应式模块可以很方便的编写独立逻辑并提供响应式的数据，然后在setup选项中组合使用，增强代码的可读性和维护性。例如：

```
// 复用逻辑1
function useXX() {}
// 复用逻辑2
function useYY() {}
// 逻辑组合
const Comp = {
   setup() {
      const {xx} = useXX()
      const {yy} = useYY()
      return {xx, yy}
   }
}
```

可能的追问

Vue.extend方法你用过吗？它能用来做组件扩展吗？

知其所以然

mixins原理：

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/apiCreateApp.ts#L232-L233

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/componentOptions.ts#L545

slots原理：

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/componentSlots.ts#L129-L130

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/renderer.ts#L1373-L1374

https://github1s.com/vuejs/core/blob/HEAD/packages/runtime-core/src/helpers/renderSlot.ts#L23-L24

06-子组件可以直接改变父组件的数据么，说明原因

分析

这是一个实践知识点，组件化开发过程中有个**单项数据流原则**，不在子组件中修改父组件是个常识问题。

参考文档：https://staging.vuejs.org/guide/components/props.html#one-way-data-flow

---

思路

1. 讲讲单项数据流原则，表明为何不能这么做
2. 举几个常见场景的例子说说解决方案
3. 结合实践讲讲如果需要修改父组件状态应该如何做

回答范例

1. 所有的 prop 都使得其父子之间形成了一个**单向下行绑定**：父级 prop 的更新会向下流动到子组件中，但是反过来则不行。这样会防止从子组件意外变更父级组件的状态，从而导致你的应用的数据流向难以理解。另外，每次父级组件发生变更时，子组件中所有的 prop 都将会刷新为最新的值。这意味着你**不**应该在一个子组件内部改变 prop。如果你这样做了，Vue 会在浏览器控制台中发出警告。

   ```
   const props = defineProps(['foo'])
   // ❌ 下面行为会被警告，props是只读的!
   props.foo = 'bar'
   ```

---

2. 实际开发过程中有两个场景会想要修改一个属性：
   - **这个 prop 用来传递一个初始值；这个子组件接下来希望将其作为一个本地的 prop 数据来使用。**在这种情况下，最好定义一个本地的 data，并将这个 prop 用作其初始值：

     ```
     const props = defineProps(['initialCounter'])
     const counter = ref(props.initialCounter)
     ```

   - **这个 prop 以一种原始的值传入且需要进行转换。**在这种情况下，最好使用这个 prop 的值来定义一个计算属性：

```
const props = defineProps(['size'])
// prop变化，计算属性自动更新
const normalizedSize = computed(() => props.size.trim().toLowerCase())
```

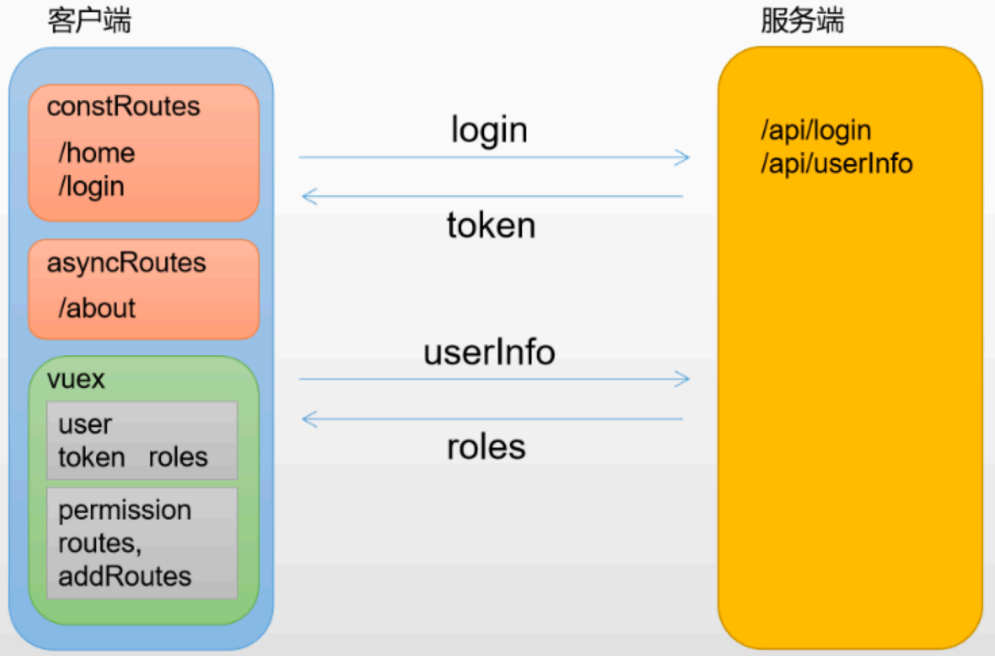
3. 实践中如果确实想要改变父组件属性应该emit一个事件让父组件去做这个变更。注意虽然我们不能直接修改一个传入的对象或者数组类型的prop，但是我们还是能够直接改内嵌的对象或属性。

07-Vue要做权限管理该怎么做？控制到按钮级别的权限怎么做？

分析

综合实践题目，实际开发中经常需要面临权限管理的需求，考查实际应用能力。

权限管理一般需求是两个：页面权限和按钮权限，从这两个方面论述即可。



思路

1. 权限管理需求分析：页面和按钮权限
2. 权限管理的实现方案：分后端方案和前端方案阐述
3. 说说各自的优缺点

回答范例

1. 权限管理一般需求是**页面权限**和**按钮权限**的管理
2. 具体实现的时候分后端和前端两种方案：

前端方案会**把所有路由信息在前端配置**，通过路由守卫要求用户登录，用户**登录后根据角色过滤出路由表**。比如我会配置一个`asyncRoutes`数组，需要认证的页面在其路由的`meta`中添加一个`roles`字段，等获取用户角色之后取两者的交集，若结果不为空则说明可以访问。此过滤过程结束，剩下的路由就是该用户能访问的页面，**最后通过`router.addRoutes(accessRoutes)`方式动态添加路由**即可。

后端方案会**把所有页面路由信息存在数据库**中，用户登录的时候根据其角色**查询得到其能访问的所有页面路由信息**返回给前端，前端**再通过`addRoutes`动态添加路由**信息

按钮权限的控制通常会**实现一个指令**，例如`v-permission`，**将按钮要求角色通过值传给v-permission指令**，在指令的`moutned`钩子中可以**判断当前用户角色和按钮是否存在交集**，有则保留按钮，无则移除按钮。

3. 纯前端方案的优点是实现简单，不需要额外权限管理页面，但是维护起来问题比较大，有新的页面和角色需求就要修改前端代码重新打包部署；服务端方案就不存在这个问题，通过专门的角色和权限管理页面，配置页面和按钮权限信息到数据库，应用每次登陆时获取的都是最新的路由信息，可谓一劳永逸！

知其所以然

路由守卫

https://github1s.com/PanJiaChen/vue-element-admin/blob/HEAD/src/permission.js#L13-L14

路由生成

https://github1s.com/PanJiaChen/vue-element-admin/blob/HEAD/src/store/modules/permission.js#L50-L51

动态追加路由

https://github1s.com/PanJiaChen/vue-element-admin/blob/HEAD/src/permission.js#L43-L44

可能的追问

1. 类似`Tabs`这类组件能不能使用`v-permission`指令实现按钮权限控制?

```
<el-tabs>
  <el-tab-pane label="用户管理" name="first">用户管理</el-tab-pane>
  <el-tab-pane label="角色管理" name="third">角色管理</el-tab-pane>
</el-tabs>
```

---

2. 服务端返回的路由信息如何添加到路由器中?

```
// 前端组件名和组件映射表
const map = {
  //xx: require('@/views/xx.vue').default // 同步的方式
  xx: () => import('@/views/xx.vue') // 异步的方式
}
// 服务端返回的asyncRoutes
const asyncRoutes = [
  { path: '/xx', component: 'xx',... }
]
```

```
// 遍历asyncRoutes，将component替换为map[component]
function mapComponent(asyncRoutes) {
  asyncRoutes.forEach(route => {
    route.component = map[route.component];
    if(route.children) {
      route.children.map(child => mapComponent(child))
    }
  })
}
mapComponent(asyncRoutes)
```

---

08 - 说一说你对vue响应式理解?

分析

这是一道必问题目，但能回答到位的比较少。如果只是看看一些网文，通常没什么底气，经不住面试官推敲，但像我们这样即看过源码还造过轮子的，回答这个问题就会比较有底气啦。

答题思路：

1. 啥是响应式?
2. 为什么vue需要响应式?
3. 它能给我们带来什么好处?
4. vue的响应式是怎么实现的？有哪些优缺点?
5. vue3中的响应式的新变化

回答范例：

1. 所谓数据响应式就是**能够使数据变化可以被检测并对这种变化做出响应的机制**。
2. MVVM框架中要解决的一个核心问题是连接数据层和视图层，通过**数据驱动**应用，数据变化，视图更新，要做到这点的就需要对数据做响应式处理，这样一旦数据发生变化就可以立即做出更新处理。
3. 以vue为例说明，通过数据响应式加上虚拟DOM和patch算法，开发人员只需要操作数据，关心业务，完全不用接触繁琐的DOM操作，从而大大提升开发效率，降低开发难度。
4. vue2中的数据响应式会根据数据类型来做不同处理，如果是**对象则采用Object.defineProperty()**的方式定义数据拦截，当数据被访问或发生变化时，我们感知并作出响应；如果是**数组则通过覆盖数组对象原型的7个变更方法**，使这些方法可以额外的做更新通知，从而作出响应。这种机制很好的解决了数据响应化的问题，但在实际使用中也存在一些缺点：比如初始化时的递归遍历会造成性能损失；新增或删除属性时需要用户使用Vue.set/delete这样特殊的api才能生效；对于es6中新产生的Map、Set这些数据结构不支持等问题。
5. 为了解决这些问题，vue3重新编写了这一部分的实现：利用ES6的Proxy代理要响应化的数据，它有很多好处，编程体验是一致的，不需要使用特殊api，初始化性能和内存消耗都得到了大幅改善；另外由于响应化的实现代码抽取为独立的reactivity包，使得我们可以更灵活的使用它，第三方的扩展开发起来更加灵活了。

知其所以然

vue2响应式：

https://github1s.com/vuejs/vue/blob/HEAD/src/core/observer/index.js#L135-L136

vue3响应式：

https://github1s.com/vuejs/core/blob/HEAD/packages/reactivity/src/reactive.ts#L89-L90

https://github1s.com/vuejs/core/blob/HEAD/packages/reactivity/src/ref.ts#L67-L68