面试官：按钮级别权限怎么控制

最近的面试中有一个面试官问我按钮级别的权限怎么控制，我说直接 `v-if` 啊，他说不够好，我说我们项目中按钮级别的权限控制情况不多，所以 `v-if` 就够了，他说不够通用，最后他对我的评价是做过很多东西，但是都不够深入，好吧，那今天我们就来深入深入。

因为我自己没有相关实践，所以接下来就从这个有 `16.2k` 星星的后台管理系统项目Vue vben admin 中看看它是如何做的。

获取权限码

要做权限控制，肯定需要一个 `code` ，无论是权限码还是角色码都可以，一般后端会一次性返回，然后全局存储起来就可以了， `Vue vben admin` 是在登录成功以后获取并保存到全局的 `store` 中：

```
import { defineStore } from 'pinia';
export const usePermissionStore = defineStore({
    state: () => ({
        // 权限代码列表
        permCodeList: [],
    }),
    getters: {
        // 获取
        getPermCodeList(){
                        return this.permCodeList;
            },
    },
    actions: {
        // 存储
        setPermCodeList(codeList) {
            this.permCodeList = codeList;
        },

        // 请求权限码
        async changePermissionCode() {
            const codeList = await getPermCode();
            this.setPermCodeList(codeList);
        }
    }
})
```

接下来它提供了三种按钮级别的权限控制方式，一一来看。

函数方式

使用示例如下：

```
<template>
  <a-button v-if="hasPermission(['20000', '2000010'])" color="error" class="mx-4">
    拥有[20000,2000010]code可见
  </a-button>
</template>

<script lang="ts">
  import { usePermission } from '/@/hooks/web/usePermission';

  export default defineComponent({
    setup() {
      const { hasPermission } = usePermission();
      return { hasPermission };
    },
  });
</script>
```

本质上就是通过 `v-if` ，只不过是通过一个统一的权限判断方法 `hasPermission` ：

```
export function usePermission() {
    function hasPermission(value, def = true) {
        // 默认视为有权限
        if (!value) {
            return def;
        }

        const allCodeList = permissionStore.getPermCodeList;
        if (!isArray(value)) {
            return allCodeList.includes(value);
        }
        // intersection是lodash提供的一个方法，用于返回一个所有给定数组都存在的元素组成的数组
        return (intersection(value, allCodeList)).length > 0;

        return true;
    }
}
```

很简单，从全局 `store` 中获取当前用户的权限码列表，然后判断其中是否存在当前按钮需要的权限码，如果有多个权限码，只要满足其中一个就可以。

组件方式

除了通过函数方式使用，也可以使用组件方式， `Vue vben admin` 提供了一个 `Authority` 组件，使用示例如下：

```
<template>
  <div>
    <Authority :value="RoleEnum.ADMIN">
      <a-button type="primary" block> 只有admin角色可见 </a-button>
    </Authority>
  </div>
</template>
<script>
  import { Authority } from '/@/components/Authority';
  import { defineComponent } from 'vue';
  export default defineComponent({
    components: { Authority },
  });
</script>
```

使用 `Authority` 包裹需要权限控制的按钮即可，该按钮需要的权限码通过 `value` 属性传入，接下来看看 `Authority` 组件的实现。

```
<script lang="ts">
  import { defineComponent } from 'vue';
  import { usePermission } from '/@/hooks/web/usePermission';
  import { getSlot } from '/@/utils/helper/tsxHelper';

  export default defineComponent({
    name: 'Authority',
    props: {
      value: {
        type: [Number, Array, String],
        default: '',
      },
    },
    setup(props, { slots }) {
      const { hasPermission } = usePermission();

      function renderAuth() {
```

```
        const { value } = props;
        if (!value) {
          return getSlot(slots);
        }
        return hasPermission(value) ? getSlot(slots) : null;
      }

      return () => {
        return renderAuth();
      };
    },
  });
</script>
```

同样还是使用 `hasPermission` 方法，如果当前用户存在按钮需要的权限码时就原封不动渲染 `Authority` 包裹的内容，否则就啥也不渲染。

指令方式

最后一种就是指令方式，使用示例如下：

```
<a-button v-auth="'1000'" type="primary" class="mx-4"> 拥有code ['1000']权限可见
</a-button>
```

实现如下：

```
import { usePermission } from '/@/hooks/web/usePermission';

function isAuth(el, binding) {
  const { hasPermission } = usePermission();

  const value = binding.value;
  if (!value) return;
  if (!hasPermission(value)) {
    el.parentNode?.removeChild(el);
  }
}

const mounted = (el, binding) => {
  isAuth(el, binding);
};

const authDirective = {
```

```
  // 在绑定元素的父组件
  // 及他自己的所有子节点都挂载完成后调用
  mounted,
};

// 注册全局指令
export function setupPermissionDirective(app) {
  app.directive('auth', authDirective);
}
```

只定义了一个 `mounted` 钩子，也就是在绑定元素挂载后调用，依旧是使用 `hasPermission` 方法，判断当前用户是否存在通过指令插入的按钮需要的权限码，如果不存在，直接移除绑定的元素。

很明显， `Vue vben admin` 的实现有两个问题，一是不能动态更改按钮的权限，二是动态更改当前用户的权限也不会生效。

解决第一个问题很简单，因为上述只有删除元素的逻辑，没有加回来的逻辑，那么增加一个 `updated` 钩子：

```
app.directive("auth", {
    mounted: (el, binding) => {
        const value = binding.value
        if (!value) return
        if (!hasPermission(value)) {
            // 挂载的时候没有权限把元素删除
            removeEl(el)
        }
    },
    updated(el, binding) {
        // 按钮权限码没有变化，不做处理
        if (binding.value === binding.oldValue) return
        // 判断用户本次和上次权限状态是否一样，一样也不用做处理
        let oldHasPermission = hasPermission(binding.oldValue)
        let newHasPermission = hasPermission(binding.value)
        if (oldHasPermission === newHasPermission) return
        // 如果变成有权限，那么把元素添加回来
        if (newHasPermission) {
            addEl(el)
        } else {
        // 如果变成没有权限，则把元素删除
            removeEl(el)
        }
    },
})

const hasPermission = (value) => {
```

```
    return [1, 2, 3].includes(value)
}

const removeEl = (el) => {
    // 在绑定元素上存储父级元素
    el._parentNode = el.parentNode
    // 在绑定元素上存储一个注释节点
    el._placeholderNode = document.createComment("auth")
    // 使用注释节点来占位
    el.parentNode?.replaceChild(el._placeholderNode, el)
}

const addEl = (el) => {
    // 替换掉给自己占位的注释节点
    el._parentNode?.replaceChild(el, el._placeholderNode)
}
```

主要就是要把父节点保存起来，不然想再添加回去的时候获取不到原来的父节点，另外删除的时候创建一个注释节点给自己占位，这样下次想要回去能知道自己原来在哪。

第二个问题的原因是修改了用户权限数据，但是不会触发按钮的重新渲染，那么我们就需要想办法能让它触发，这个可以使用 `watchEffect` 方法，我们可以在 `updated` 钩子里通过这个方法将用户权限数据和按钮的更新方法关联起来，这样当用户权限数据改变了，可以自动触发按钮的重新渲染：

```
import { createApp, reactive, watchEffect } from "vue"
const codeList = reactive([1, 2, 3])

const hasPermission = (value) => {
    return codeList.includes(value)
}

app.directive("auth", {
    updated(el, binding) {
        let update = () => {
            let valueNotChange = binding.value === binding.oldValue
            let oldHasPermission = hasPermission(binding.oldValue)
            let newHasPermission = hasPermission(binding.value)
            let permissionNotChange = oldHasPermission === newHasPermission
            if (valueNotChange && permissionNotChange) return
            if (newHasPermission) {
                addEl(el)
            } else {
                removeEl(el)
            }
        };
```

```
        if (el._watchEffect) {
            update()
        } else {
            el._watchEffect = watchEffect(() => {
                update()
            })
        }
    },
})
```

将 `updated` 钩子里更新的逻辑提取成一个 `update` 方法，然后第一次更新在 `watchEffect` 中执行，这样用户权限的响应式数据就可以和 `update` 方法关联起来，后续用户权限数据改变了，可以自动触发 `update` 方法的重新运行。