开发指南篇2：学会编写可复用性模块

在生活中，重复的机械劳动会消耗我们的时间和精力，提高生产成本，降低工作效率。同样，在代码世界中，编写重复的代码会导致代码的冗余，页面性能的下降以及后期维护成本的增加。由此可见将重复的事情复用起来是提高生产效率、降低维护成本的不二之选。

在Vue 项目中，每一个页面都可以看作是由大大小小的模块构成的，即便是一行代码、一个函数、一个组件都可以看作是一个个自由的模块。那么提高代码的复用性的关键便在于编写可复用的模块，也就是编写可复用的代码、函数和组件等。

一个简单的例子

```javascript
let person = [];

for (let i = 0; i < data.obj.items.length; i++) {
    person.push({
        name: data.obj.items[i].name,
        age: data.obj.items[i].age
    });
}
```

不知道上方代码给你的第一印象是什么？总之给我的印象是糟糕的，因为出现了重复性的代码片段 `data.obj.items` ，可能这样的代码在我们团队开发中随处可见，这也说明了重复编码现象其实无处不在。

面对自己编写的代码，我们应该保持一颗去重的心，发现重复的地方就相当于找到了可以复用的模块。在不复用的情况下，上述代码一旦需要修改变量 `items` 为 `lists` ，那么我们就得修改3处地方，不知不觉就增加了维护成本。而到时候往往修改你代码的人并不是你自己，所以对自己好点，对他人也会好点。复用后的代码如下：

```
for (let i = 0; i < values.length; i++) {
    person.push({
        name: values[i].name,
        age: values[i].age
    });
}
```

我们通过将data.obj.items的值赋值给变量values来实现了复用，此时修改 `items` 为 `lists` 的话我们只需修改一处地方即可，不管是维护成本还是代码可读性上，复用的优势都显而易见。

封装成一个函数

除了使用变量的赋值缓存使用来解决数据的重复读取外，我们在开发过程中重复性更多的也许是功能点的重复，比如：

```html
<tempalte>
    <div>
        <input type="text" v-model="str1">
        <input type="text" v-model="str2">
        <div>{{ str1.slice(1).toUpperCase() }}</div>
        <div>{{ str2.slice(1).toUpperCase() }}</div>
    </div>
</template>
```

上述代码的重复功能点在于截取输入框中第二个字符开始到最后的值并把它们转化成大写字母，像这样很简单的操作虽然重复使用也不会出现太大的问题，但是如果是代码量较多的操作呢？重复书写相同功能的代码是一种不经过大脑思考的行为，我们需要对其进行优化，这里我们可以把功能点封装成一个函数：

```javascript
export default {
    methods: {
        sliceUpperCase(val) {
            return val.slice(1).toUpperCase()
        }
    }
}
```

值和 v-bind 表达式中重复的功能点我们可以封装成过滤器比较合适：

```javascript
// 单文件组件注册过滤器
filters: {
    sliceUpperCase(val) {
        return val.slice(1).toUpperCase()
    }
}

// 全局注册过滤器
Vue.filter('sliceUpperCase', function (val) {
    return val.slice(1).toUpperCase()
})
```

然后在 html 中使用“管道”符进行过滤：

```html
<div>{{ str1 | toUpperCase }}</div>
<div>{{ str2 | toUpperCase }}</div>
```

这样我们就把重复的功能性代码封装成了函数，而不管是过滤器还是正常的方法封装，其本质都是函数的封装。

封装成一个组件

相比较于函数的封装，规模更大一点的便是组件的封装，组件包含了模板、脚本以及样式的代码，在实际开发中组件的使用频率也是非常大的，我们项目中的每一个页面其实都可以看作是一个父组件，其可以包含很多子组件，子组件通过接收父组件的值来渲染页面，父组件通过响应子组件的回调来触发事件。

封装一个组件主要包含两种方式，一种是最常见的整体封装，用户通过改变数据源来呈现不同的页面状态，代码结构不可定制化。例如：

```html
<div>
    <my-component data="我是父组件传入子组件的数据"></my-component>
</div>
```

```html
<div>
    <my-component data="我是父组件传入子组件的数据">
        <template slot="customize">
            <span>这是定制化的数据</span>
        </template>
    </my-component>
</div>
```

在 myComponent 组件中我们便可以接收对应的 slot：

```html
<div class="container">
    <span>{{ data }}</span>
    <slot name="customize"></slot>
<div>
```

这里我们通过定义 slot 标签的 name 值为 customize 来接收父组件在使用该组件时在 template 标签上定义的 slot="customize" 中的代码，不同父组件可以定制不同的 slot 代码来实现差异化的插槽。最终渲染出来的代码如下：

```html
<div>
    <div class="container">
        <span>我是父组件传入子组件的数据</span>
        <span>这是定制化的数据</span>
    </div>
</div>
```

这样我们就完成了一个小型组件的封装，将共用代码封装到组件中去，页面需要引入的时候直接使用 import 并进行相应注册即可，当然你也可以进行全局的引入：

```javascript
import myComponent from '../myComponent.vue'

// 全局
Vue.component('my-component', myComponent)
```

在某些情况下，我们封装的内容可能不需要使用者对其内部代码结构进行了解，其只需要熟悉我们提供出来的相应方法和 api 即可，这需要我们更系统性的将公用部分逻辑封装成插件，来为项目添加全局功能，比如常见的 loading 功能、弹框功能等。

Vue 提供给了我们一个 install 方法来编写插件，使用该方法中的第一个 Vue 构造器参数可以为项目添加全局方法、资源、选项等。比如我们可以给组件添加一个简单的全局调用方法来实现插件的编写：

javascript

```
/* toast.js */
import ToastComponent from './toast.vue' // 引入组件

let $vm

export default {
    install(Vue, options) {

        // 判断实例是否存在
        if (!$vm) {
            const ToastPlugin = Vue.extend(ToastComponent); // 创建一个“扩展实例构造器”

            // 创建 $vm 实例
            $vm = new ToastPlugin({
                el: document.createElement('div')  // 声明挂载元素
            });

            document.body.appendChild($vm.$el); // 把 toast 组件的 DOM 添加到 body 里
        }

        // 给 toast 设置自定义文案和时间
        let toast = (text, duration) => {
            $vm.text = text;
            $vm.duration = duration;

            // 在指定 duration 之后让 toast 消失
            setTimeout(() => {
                $vm.isShow = false;
            }, $vm.duration);
        }

        // 判断 Vue.$toast 是否存在
        if (!Vue.$toast) {
```

```
        Vue.prototype.$toast = Vue.$toast; // 全局添加 $toast 事件
    }
}
```

成功编写完插件的 JS 脚本后，我们在入口文件中需要通过 Vue.use() 来注册一下该插件：

```javascript
import Toast from '@/widgets/toast/toast.js'

Vue.use(Toast); // 注册 Toast
```

最后我们在需要调用它的地方直接传入配置项使用即可，比如：

```javascript
this.$toast('Hello World', 2000);
```

当然你也可以不使用 install 方法来编写插件，直接采用导出一个封装好的实例方法并将其挂载到 Vue 的原型链上来实现相同的功能。

更详细的编写插件和实例的方法可以参考我之前写的一篇文章：Vue 插件编写与实战

结语

本文讲解了编写可复用性模块的常见方法，通过出现了重复代码 -> 封装成一个变量 -> 封装成一个函数 -> 封装成一个组件 -> 封装成一个插件，一步步将重复代码进行分析和复用。而

思考 & 作业

- 在 Vue 中如何添加全局自定义指令?
- 在 vue 路由切换时如何全局隐藏某个插件？比如文中的 toast
- 如何实现一个表单验证插件？需要运用到哪些知识?

留言

评论将在后台进行审核，审核通过后对所有人可见

雪飞花1991 前端开发工程师 @ 光环云

能不能详细讲讲组件这层的封装使用，相当困惑，文中提到 slot 插槽，我没怎么使用，只是 ui框架中使用罢了，感觉放弃了一个美好的花园

▲ 0　收起评论　3月前

劳卜 前端工程师 @ TC

复用性高的组件，建议把业务逻辑抽离出来，可以增强组件的适用性和可维护性，具体的组件封装可以参考这篇文章：
https://www.cnblogs.com/landeanfen/p/6518679.html

3月前

墨月落羽舞苍风 前端小白

确实插槽这个功能没有用到过很多,有的时候甚至想不起来这个功能模块

3月前

雪飞花1991 前端开发工程师 @ 光环云

回复 劳卜： 好呢，谢谢

3月前

彼岸花开1477528254153 前端开发工程师

回复 彼岸花开1477528254153： 可以了 在插件里面将isShow重置下就行了，打扰了

2月前

评论审核通过后显示

评论