Vue 组件库实践和设计

现在前端的快速发展，已经让组件这个模式变的格外重要。对于市面上的组件库，虽然能满足大部分的项目，但是一些小型细节方面和使用方面，或者UI库存在的一些bug，会让人很头疼。

那我们应该如何面对解决这些问题。俗话说自己动手丰衣足食。有些组件不用刻意去造。应该考虑如何去打造一个快速，兼容性好，功能齐全的组件库。

1. 先到github上和一些大公司开源的组件库官网上去看看你所需组件库的demo例子，Prop和event暴露出来了哪些接口。

2. 货比三家。别人分别用了哪些模式设计，哪种模式最简便，更合理。

3. 一般成熟的UI库兼容性是经过大量测试和用户使用后没有问题后的结果。省去这一部分，进行样式的借鉴。

4. 不要太急于进行组件的大量造轮子。因为一个人的战斗是有限的。根据需要和场景项目进行一个个定位，积少成多。

GitChat

Crib-zk也是我个人目前针对自己项目的需求，额外进行的组件。虽然不能用于开源市场的使用，但是可以用于大家的学习。使用和学习是两种模式。会使用不代表你懂，一旦有些需求不在开源项目组件的范围之内。此时又不能清楚内部的原理，就会措手不及。接下来进行一步一步分析。

大纲：分析组件

1. alert 插件/组件

2. backtop 组件

3. sms-countDown组件（短信倒计时）

4. search 组件

5. infinitscroll 组件

6. actionSheet 组件

7. accordion 组件（手风琴）

注意 ： 看这篇文章最好结合我github上发布的组件，进行比对式观看。

一、alert组件/插件

如果在alert这种弹出式组件里，首先要加一些背景layout的背景层动画化，可以简称为dialog动化，进行一个包裹。

对于alert组件/插件的区别使用性是在那里？

一般来说，先会定义一个.vue文件的alert模板。

GitChat

```
<template>
    <div class="vux-alert">
        <x-dialog :value='alertShow' :isClose='false'>
            <div class="crib-confirm_hd" :style='[titleStyle]'>
                <strong class="crib-confirm_title">{{title}}
</strong>
            </div>
            <div class="crib-confirm_bd">
                <slot>
                    <div v-html="content"></div>
                </slot>
```

```
            </div>
            <div class="crib-confirm_ft">
                <a href="javascript:;" class="crib-confirm_btn
crib-confirm_btn_primary" @click="_onSubmit"
:style='[buttonStyle]'>{{buttonText}}</a>
            </div>
        </x-dialog>
    </div>
</template>
```

仔细看上面模板。

1. 我们发现唯一特别的是对content体中的定义了一个solt。这个slot就是组件模式和插件模式的区分。如果我们想对slot里面定义的是一个额外的展示模板或其它组件插入的话，此时只能用组件模式。
2. 如果只是我们对content这个数据进填充的话，插件模式也是最方便的。

props接口的暴露

1. value 显示消息
2. title (标题)
3. content 内容最好支持html格式
4. buttonText底部的按钮文案
5. titleStyle 标题样式
6. buttonStyle button样式

GitChat

event接口暴露

1. onsubmit 点击时向外暴露事件
2. onshow 显示时向外暴露的显示事件
3. onhide 显示时向暴露事件

```
data() {
        return {
            alertShow: this.value
        }
    },
    watch: {
        value(val) {
```

```
            this.alertShow = val
        }
    },
    methods: {
        _onSubmit() {
            this.alertShow = false
            this.$emit('update:value', false)
            this.$emit('on-submit')
        }
    }
```

对于value这个值来说，可以用.sync来进行简便的操作。不需要通过*emit*来进行通知。在声明组件的时候用on去进行监听事件，省去了开发者这一步的事。

插件模式

首先要把原本的alert的.vue的模板给引处进来，然后用Vue.extend继承一下。

```
$vm = new Alert({
            el: document.createElement('div')
        })
```

GitChat

我们自己要手动进行一个创建挂载点。

```
//此方法是用来把confirm上的prop属性合并到调用时的参数上
const mergeOptions = function($vm,options) {
    //声明一个默认的对象，就是comfirm上props属性的default的值
    const defaults = {}
    //循环confirm属性上的props值
    for (let i in $vm.$options.props){
       //不把value的值算上去，显示改变通过watch或者改变data代理的属性上去监听
        if(i !== 'value'){
           defaults[i] = $vm.$options.props[i].default
        }
    }
    //把confrim组件原本的值和插件传入的options合并
    const _options = Object.assign({},defaults,options)
    //把confirm组件生成的实列对象再次替换成合并的属性
    for(let i in _options) {
        $vm[i] = _options[i]
    }
}
```

同时要把value显示操作的默认定义的属性除外，进行自己定义后覆盖默认属性，进行显示。

```
$watcher = $vm.$watch('alertShow', (val) => {
                    if (!val && options && options.onHide) {
                        options.onHide()
                    }
                })
```

同时对alertshow进行监听，当点击submit的时候会自己动触发事件，然后会改变alertshow的值，然后进行你所想要的操作。

二、backtop 组件

对于backtop组件的话，要理解几点属性。

1. scrollTop

2. offsetHeight

```
let offsetTop = this.scrollview == window ?
document.body.scrollTop : this.scrollview.scrollTop
            let offsetHeight = this.scrollview == window ?
document.body.offsetHeight : this.scrollview.offsetHeight
```

GitChat

scrollTop 是距离顶部的高度。

offsetHeight 元素的高度包括边框。

那如何去判断什么时候显示返回顶部按钮呢？

this.show = offsetTop >= offsetHeight / 2;

只要通过滚动的高度大于滚动元素的高度/2来进行一个适配是最好的。

对于如何进行那方面优化。

可以进行函数节流。节流是个什么？因为进行滚动监听的时候，scroll事件触发的太频繁了。这会影响到整个性能的问题。如果对于上下滚动也要频繁监听。用节点，不适用于防抖操作。

```
 throttle(func, wait) {
            var context, args;
            var previous = 0;

            return function () {
                var now = +new Date();
```

```
                context = this;
                args = arguments;
                if (now - previous > wait) {
                    func.apply(context, args);
                    previous = now;
                }
            }
        },
```

通过时间戳来进行对比，来进行函数节流。但是有一点需要注意，在节流的同时，不要节流的时节太长。因为mobile上面节流滚动的话，有一个自行滑动的时长。

```
const getScrollview = function (el) {
    //拿到当前节点
    let currentNode = el;
    //如果有节点，并且节点不等于html ,body 并且节点类型是元素节点
    while (currentNode && currentNode.tagName !== 'HTML' &&
currentNode.tagName !== 'BODY' && currentNode.nodeType === 1) {
        //拿到节点的overflowy的属性
        let overflowY =
document.defaultView.getComputedStyle(currentNode).overflowY;
        //如果此时属性是scroll或者atuo 就返回此节点
        if (overflowY === 'scroll' || overflowY === 'auto') {
            return currentNode;
        }
        //否则就继续向父节点上找
        currentNode = currentNode.parentNode;
    }
    //一但while语句为false的时候就直接返回window对像
    return window;
};


export {getScrollview}
```

GitChat

在外层要进行一个包裹，通overflow属性向来进行推测， 是全局滚动还是window 下的滚动，通过while来进行判断递归，来查找所对应的元素 。

三、sms-countDown 短信倒计时组件

对短信倒计时的认知

对于短信倒计时最重要的一点就是从父组件向sms组件通知倒计时开始的一个prop参数，用start替代。

GitChat

```
watch : {
      start (value) {
        if(value === true){
          this.countDown()
        }
      }
    }
```

同时对start在内部进行监听。一旦从外部传入开始的时候，则内部进行倒计时。

```
countDown () {
            this.myTime = 60;
            let time = setInterval(()=>{
                this.myTime --;
                if(this.myTime === 0){
                  this.$emit('update:start', false);
                  this.myTime = this.initText;
                  this.flag = true;
                  clearInterval(time)
                }
            },100)
        }
```

在这里同样要进行一个倒计时停止之后的向外通知。还是用.sync的双向绑定方法，用于简便操作。

在对于第一次倒计时和第二次倒计时的时候，也要对文案这方面进行一个设定。

```
firstCkText : {
            type : String,
            default : ''
          },
          secondCKText : {
            type : String,
            default : '重新获取'
          },
```

第一次点击和第二次点击的按钮，也是对主要的文案的一种设计，所以对文案的变化也是要很关注的。

四、search组件

GitChat

对于search组件通常能想到哪些对应的功能和想法呢？比如首次进来的时候，要进行自动获取Input的焦点。同时要向外面暴露是否要获取Input获点事的Prop ：autoFocus。同时也要注意，一定要在Mounted的时候才能拿 到dom元素。

```
mounted() {
      this.autoFocus && this.$refs.input.focus()
  }
```

一般想知道input里面的value值是否改动的时候，通常都会用keydown或者是keyup事件。但是这里不需要，可以时时把value的值给暴露出去，让外层父组件可以去进行watch监听来进行进所需要的事件操作。

```
watch: {
        inputValue (val) {
           if(val == ''){
               this.value = ""
           }
        },
        value: {
            handler(val, oldvalue) {
                //当值改变的时候，触发事件
                this.$emit('update:inputValue', val)
                this.$emit('change-val')
```

```
            },
            immediate: true

        }
    },
```

同时这里用到了.sync ，在页面一加载的时候，立马执行了。immediate使得value这个值立马值行了监听。

五、infinite-scroll 组件（无限滚动组件)

无限滚动最关键的三个地方。第一滚动动底部触发事件;第二如果有二次加载则显示loading;第三如何没有二次加载则结束文案。

```
import { getScrollview } from '../../libs/getScrollview.js';
```

这个不用说，继续寻找需要滚动范围的元素 。

```
data() {
        return {
            isLoading: false, //是否正在加载
            isDone: false,  //是否加载完毕
        }
    },
```

GitChat

data里面进行之前说的两种模式的状态进行定义。往下看，这一处定义之后对后面有什么好处。

```
this.$on('Load', () => {
                this.isLoading = false;
            });
            this.$on('loadDone', () => {
                    this.isLoading = false;
                    this.isDone = true;
            });
```

需要监听两个事件：

1. 二次加载load事件。一旦进行二次加载的时候，马上进行isloading等于false 防止重复加载。
2. 通过loadDone对是否监听完毕。如果加载完毕的话，同样的关闭isloading 对isDone进行true的设置。

isloading和isDone分别对应的那个html 的template部分。

```
<div class="list-donetip" v-show='!isLoading&& isDone'>
            <slot name='doneTip'>没有更多数据了</slot>
        </div>

        <div class="list-loading" v-show='isLoading'>
            <slot name='loadingTip'>加载中...</slot>
        </div>
```

当isloading 为true的时候显示“加载中…”当isloading 为false的时候，isDone为true的时候才显示 “没有更多数据”，这也是一个标准的无限滚动。

什么时候对isloading和isDone设置为true?

```
 scrollHandler() {
            if (this.isLoading || this.isDone) return;
            let baseHeight = this.scrollview == window ?
document.body.offsetHeight : this.scrollview.offsetHeight
            let moreHeight = this.scrollview == window ?
document.body.scrollHeight : this.scrollview.scrollHeight;
            let scrollTop = this.scrollview == window ?
document.body.scrollTop : this.scrollview.scrollTop
            if (baseHeight + scrollTop + this.distance >
moreHeight) {
                this.isLoading = true;

                this.onInfinite()
            }

            if (!this.scrollview) {
                console.warn('Can\'t find the scrollview!');
                return;
            }
        },
```

当滚动到底部的时候，对isloading进行为true的设置。外部组件可以调用onInfinite，进行ajax请求等操作。

在外部如何调用呢？

```
this.$refs.infinite.$emit('loadDone')
```

对组件进行 ref的设置，然后进行触发loadDone或者load。

比对饿么的组件，它使用的是指令的模式，内部实现还是太复杂。

GitChat

这里就用到了函数防抖，同上面不用函数节流，用防抖。防抖跟节流的有什么区别？防抖比较更节省性能。如果我们在设置的时间内一直滑动，则不会进行加载，只有滑动到指定的地方，则可以进行检测,通过定时器来实现。

```
debounce  (func, wait) {
            var timeout;
            return function () {
                var context = this;
                var args = arguments;

                clearTimeout(timeout)
                timeout = setTimeout(function () {
                    func.apply(context, args)
                }, wait);
            }
        },
```

六、actionSheet 组件

GitChat

GitChat

actionSheet 这里亮点就是巧用了。call方法来改变了this的指向。这个有什么好处？往下面看。

prpps ： model 我通过Model这个数据进行递，把文案的改变，点击后所执行的方法一并封装到Model数据里来进行操作。

如果在父组件引入这个组件的时候，看下面代码。

```
    model: [
                    { name: this.name, method(name, index) {
    location.href = 'tel:110' } }
                ],
```

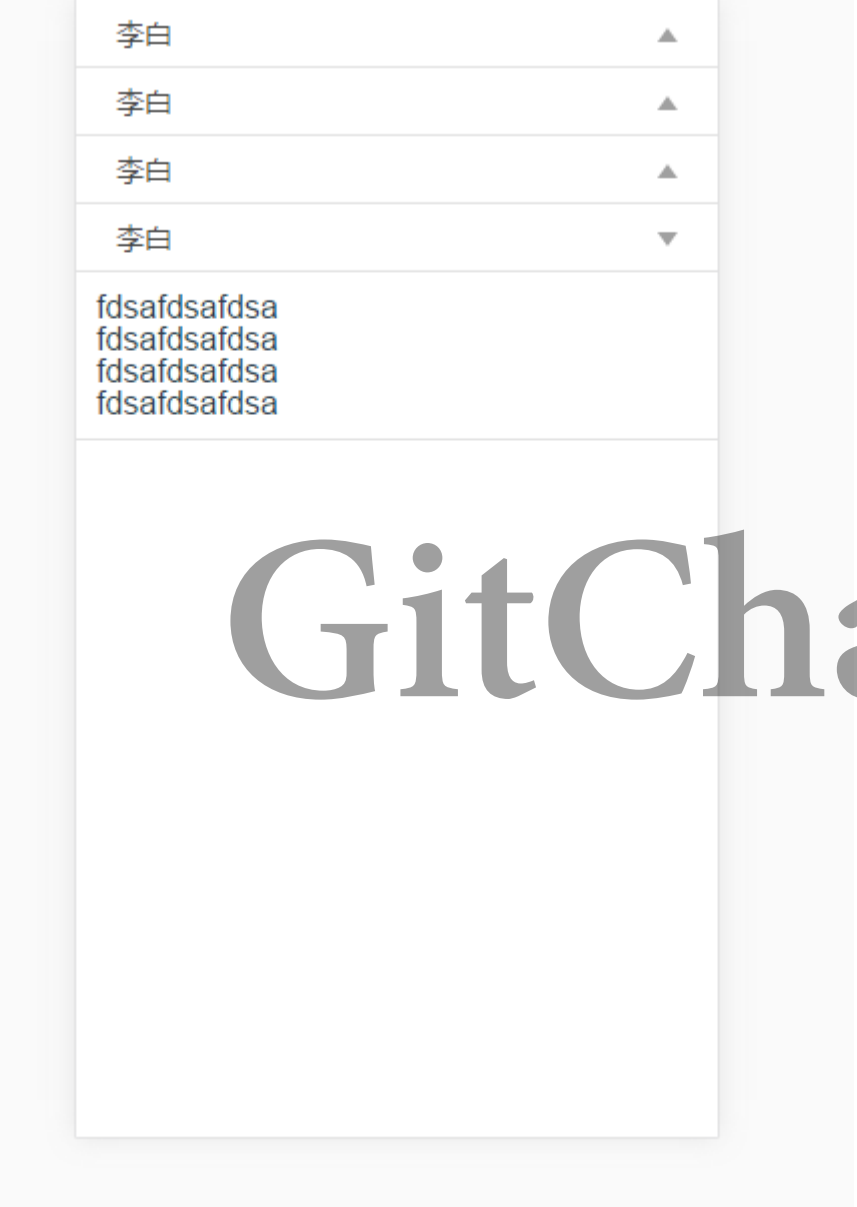
如果进行this.指向的话，指向的是父组件。这里就不能直接在data里面声明了。如果是异步的话，只有在ajax请求的异步里进行声明，把值传入，是如何做的呢？

```
    ff (item,index,method) {
                this.$emit('update:show', false)
```

```
                method.call(this.$parent,item,index)
            }
```

在这里通过.call来把this.的指向到父组件，就能成功的方便的调用了。

七、accordion 手风琴组件



GitChat

对accordion组件要进行定义两个组件合并成一个组件的模式。

1. 一个最外层的包裹组件。
2. 第二个是每一个item的组件。

每说 accordion-item里面的组件，通过

```
this.height = (this.show ? this.$refs.content.offsetHeight: 0) +
'px';
```

如果需要显示的话，让每一个item的元素来计算高度，展现出来。

通过_uid来进行 每个item的识别。

```
methods : {
          open(uid) {
              this.$children.forEach (item => {
                  console.log(item._uid)
                  if(item._uid == uid){
                      item.show = !item.show
                  }else{
                      if(!this.repeat){
                          item.show = false
                          item.height = 0;
                      }
                  }
              })
          }
      }
```

GitChat

能够收起的是那个item组件，则向收起的那个item组件进行一个传递。本质上通过index找到子组件对应的项也可以实现。因为_uid是唯一的。这一步也是省了一些简便的操作。

在这里把一些突出的组件，来开阔我们的思想，来进行一其它组件的封装 ，也可以基于这些组件对自己所需要的项目根据不同的需求来封装。

最后，尤大说了一句话，我最喜欢的就是看别人代码。记住这句话，你的组件能写的又快又好。