Vue.js 2.0从基础到组件

在这段时间里我不停的做着Vue的技术分享，虽然不是什么深层次的代码底能架构，如果底层架构真说出来，我就不会做Vue.js 2.0从基础到组件了，就会分享Vue从底层到还是到底层。我相信能来看我这次分享的朋友都是在工作开发层面上面临着一些问题。此次分享我们先抛开Vue - router和Vuex,很多朋友都觉得Vuex和router比较难，大错特错！

对Vue 2.0的介绍

我对Vue 2.0认知，能it前端框架的认知，在我们不算底层原理的情况下，什么才是精髓，那就是基础的方法和一些api介绍，随着现代数据量庞大，业务逻辑也变得更加复杂，随着数据情景的不同展示，jquery，angular1.0等一系列框架，已经满足不了开发的需求了，如何用数据驱动去管理数据，在我认知里，前后端联调，对接口，通过什么？那过json数据来传递着一切的信息。

我们操作dom来分析数据，那就是用屠龙刀去切菜，用数据驱动去改变数据，那才叫细功出好活。还有我们如何更好的通过组件来让一个复杂的页面划分为代码精简，易维护，可复用，阔展性强的组件集合！

如果是你如何去划分这个页面

1. 对头部进行一个组件的化分。（通过 `prop：{[img]}` 来进行头图的划分。）
2. input框的划分。
3. 图型验证码和图型input框。
4. 短信组件和短信input框。
5. 协议服务的弹窗。
6. button的划分。
7. 语音验证码的划分。

这样一算你们会发现一个小小的登陆就划分出6个组件，可能给你的感觉分的太细，那我也感觉分的太细，那我为什么要分的那么细呢，那就是增强可复用性，可阔展性。

那我何去解这个组件太过于细分的问题，我们可以合并那些东西，以我一眼看过去，唯一能合并的就是中间一套注册体系，我们把2,3,4,5,7,这几个细组件合并到login.Vue组件里，在这个层面上，我们只要暴露出四个输入框内容向外传递的数据，这样一个页面整

体就我们拆开来了，对于每个页面的代码量就减少了，对于维护，改bug是一个很大的帮助。

组件从基础开始

Vue的在项目中如何去做好一个体系问题，最主要的就是template里整体的组织，如何用好的组织体系方便的展现复杂的逻辑操作，我个人认为而不是通过new Vue去操控整体，反正new Vue里的一切选项是着template这个组织体系走的，如果是一个房子，template就是地基，new Vue里的选项就是水泥石头。

1. 模板语法

   能用javascript表达式则用表达式，我觉得表达式是给人感觉最清楚 的，能结合模板去正确使用表达式来解析那是最明了的。

   ```
   <p @click='show = false'></p>
   ```

   上面一眼就能让人明白，不用往下看就明白我所要改变数据是为了什么，就这一行模板语法用javascript表达式写让你能明白一切。

2. 修饰赋

   修饰符（Modifiers）是以半角句号 . 指明的特殊后缀，用于指出一个指令应该以特殊方式绑定。

   在修饰赋当中，我们如何灵活运用修饰符去减速少代码量，不要忘记对于组件事件要加.native。比如对组织事件冒泡和阻止默认事件都是很方便书写，直在template中书，例如 `@touchmove.prevent.stop` 。

虽然计算属性在大多数情况下更合适，但有时也需要一个自定义的 watcher 。这是为什么 Vue 提供一个更通用的方法通过 watch 选项，来响应数据的变化。当你想要在数据变化响应时，执行异步操作或开销较大的操作，这是很有用的。

基于这个官方的理解再总结我个人的整体理解。给出computed和watch的总结，记住这几点的总结，在做项目的时候想想这些总结，选择你的应用方法

- computed :
  监听多个数据或者一个数据来维护返回一个状态值 ，只要其中一个或多个数据发生了变化，则会从新计算整个函数体，从新返回状态值。

- watch :
  只有一个一个监听据，只要这个数据发生变化，就会在返回两个参数，第一个是当前的值，第二个是变化前的值，每当变化的时候，则会触发函数体的里的逻辑行为，来进逻辑后续操作。

其实我觉得计算属性也好，computed,watch这几个都不是有多难，如果浅层面上看很容易理解，如果从深层面上看，很多小伙伴会存在什么问题，就是会滥用，混用，这些计算属性，虽然最后结果都能实现，就像条条大路通罗马，你走的是最远的路，想返回可能都难

我就举以上几个简单的例子，但是我想说的就是两点基础非常重要。

要如何结合三大牛B特性：

1. computed vs watch
2. v-for
3. v-if v-else-if v-else
4. template外层包裹模板

如果你看到这个需求，你花3分种时间如何去用以上三大特性做出两种解决方法你就是Vue精通者。

1. 用v-for进行数据循环，用template包v-if 进行标红和排名标识。
2. 用v-fot进行数据循环，用methods结合：class进行票红和排名标识。

我还是感觉第二个方便点。

如何去写组件，从什么地方开始写起，如果你对基础api已经了如指掌了，那你就可以开手动组件了，组件要学会利用几个组件的很大的特性，最大的特性我就不用说了，用props接数据，用$emit去触发事件 用v-on去接收自定义事件，有时候你会发现有时候通过父组件传递来的数据，我们在组件内部一直需要改动，那我们不得不用watch方法去复制一个复本，再进行操作，那我们有什么好办法，如更面临更多组件套组件的模式下，如果利用特性api去做最简便的写法，那就是 `$parent`、`$children`、`$root`、`$el`、`$refs` 这五大特性。

组件规范

- 基于模块开发
- Vue 组件命名
- 组件表达式简单化
- 组件 props 原子化
- 验证组件的 props
- 将 this 赋值给 component 变量
- 组件结构化
- 组件事件命名
- 避免 this.$parent
- 谨慎使用 this.$refs
- 使用组件名作为样式作用域空间
- 提供组件 API 文档
- 提供组件 demo
- 对组件文件进行代码校验

基于模块开发

始终基于模块的方式来构建你的 app，每一个子模块只做一件事情。

Vue.js 的设计初衷就是帮助开发者更好的开发界面模块。一个模块是应用程序中独立的一个部分。

怎么做？

每一个 Vue 组件(等同于模块)首先必须专注于解决一个单一的问题，独立的, 可复用的, 微小的 and 可测试的。

如果你的组件做了太多的事或是变得臃肿，请将其拆分成更小的组件并保持单一的原则。一般来说，尽量保证每一个文件的代码行数不要超过 100 行。也请保证组件可独立的运行。比较好的做法是增加一个单独的 demo 示例。

Vue 组件命名

组件的命名需遵从以下原则：

- 有意义的: 不过于具体，也不过于抽象
- 简短: 2 到 3 个单词
- 具有可读性: 以便于沟通交流

同时还需要注意：

- 必须符合自定义元素规范: 使用连字符分隔单词，切勿使用保留字。
- app- 前缀作为命名空间: 如果非常通用的话可使用一个单词来命名，这样可以方便于其它项目里复用。

为什么？因为组件是通过组件名来调用的。所以组件名必须简短、富有含义并且具有可读性。

如何做？

```
<!-- 推荐 -->
<app-header></app-header>
<user-list></user-list>
<range-slider></range-slider>

<!-- 避免 -->
<btn-group></btn-group> <!-- 虽然简短但是可读性差. 使用 `button-
group` 替代 -->
<ui-slider></ui-slider> <!-- ui 前缀太过于宽泛，在这里意义不明确 -->
<slider></slider> <!-- 与自定义元素规范不兼容 -->
```

组件表达式简单化

Vue.js 的表达式是 100% 的 Javascript 表达式。这使得其功能性很强大，但也带来潜在的复杂性。因此，你应该尽量保持表达式的简单化。

为什么？

- 复杂的行内表达式难以阅读。
- 行内表达式是不能够通用的，这可能会导致重复编码的问题。
- IDE 基本上不能识别行内表达式语法，所以使用行内表达式 IDE 不能提供自动补全和语法校验功能。

怎么做？

如果你发现写了太多复杂并难以阅读的行内表达式，那么可以使用 method 或是 computed 属性来替代其功能。

```
<!-- 推荐 -->
<template>
    <h1>
        {{ `${year}-${month}` }}
    </h1>
</template>
<script type="text/javascript">
  export default {
    computed: {
      month() {
        return this.twoDigits((new Date()).getUTCMonth() + 1);
      },
      year() {
        return (new Date()).getUTCFullYear();
      }
    },
    methods: {
      twoDigits(num) {
```

```
        return ('0' + num).slice(-2);
      }
    },
  };
</script>

<!-- 避免 -->
<template>
    <h1>
        {{ `${(new Date()).getUTCFullYear()}-${('0' + ((new
Date()).getUTCMonth()+1)).slice(-2)}` }}
    </h1>
</template>
```

组件 props 原子化

虽然 Vue.js 支持传递复杂的 JavaScript 对象通过 props 属性，但是你应该尽可能的使用原始类型的数据。尽量只使用JavaScript 原始类型(字符串、数字、布尔值) 和 函数。尽量避免复杂的对象。

为什么？

- 使得组件 API 清晰直观
- 只使用原始类型和函数作为 props 使得组件的 API 更接近于 HTML(5) 原生元素。
- 其它开发者更好的理解每一个 prop 的含义、作用
- 传递过于复杂的对象使得我们不能够清楚的知道哪些属性或方法被自定义组件使用，这使得代码难以重构和维护。

怎么做？

组件的每一个属性单独使用一个 props，并且使用函数或是原始类型的值。

```
<!-- 推荐 -->
<range-slider
  :values="[10, 20]"
  min="0"
  max="100"
  step="5"
  :on-slide="updateInputs"
  :on-end="updateResults">
</range-slider>

<!-- 避免 -->
<range-slider :config="complexConfigObject"></range-slider>
```

验证组件的 props

在 Vue.js 中，组件的 props 即 API，一个稳定并可预测的 API 会使得你的组件更容易被其他开发者使用。

组件 props 通过自定义标签的属性来传递。属性的值可以是 Vue.js 字符串（:attr="value" 或 v-bind:attr="value"）或是不传。你需要保证组件的 props 能应对不同的情况。

为什么？

验证组件 props 可以保证你的组件永远是可用的(防御性编程)。即使其他开发者并未按照你预想的方法使用时也不会出错。

怎么做？

- 提供默认值
- 使用 type 属性校验类型
- 使用 props 之前先检查该 prop 是否存在

```
<template>
  <input type="range" v-model="value" :max="max" :min="min">
</template>
<script type="text/javascript">
  export default {
    props: {
      max: {
        type: Number, // 这里添加了数字类型的校验
        default() { return 10; },
      },
      min: {
        type: Number,
        default() { return 0; },
      },
      value: {
        type: Number,
        default() { return 4; },
      },
    },
  };
</script>
```


将 this 赋值给 component 变量(

在 Vue.js 组件上下文中，this指向了组件实例。因此当你切换到了不同的上下文时，要确保 this 指向一个可用的 component 变量。

换句话说，不要在编写这样的代码 const self = this; ，而是应该直接使用变量 component。

为什么？

将组件 this 赋值给变量 component可用让开发者清楚的知道任何一个被使用的地方，它代表的是组件实例。

怎么做？

```
<script type="text/javascript">
export default {
  methods: {
    hello() {
      return 'hello';
    },
    printHello() {
      console.log(this.hello());
    },
  },
};
</script>

<!-- 避免 -->
<script type="text/javascript">
export default {
  methods: {
    hello() {
      return 'hello';
    },
    printHello() {
      const self = this; // 没有必要
      console.log(self.hello());
    },
  },
};
</script>
```


组件结构化

按照一定的结构组织，使得组件便于理解。

为什么？

- 导出一个清晰、组织有序的组件，使得代码易于阅读和理解。同时也便于标准化。
- 按首字母排序属性，data, computed, watches 和 methods 使得属性便于查找。
- 合理组织，使得组件易于阅读。(name; extends; props, data and computed; components; watch and methods; lifecycle methods, 等)
- 使用 name 属性。借助于Vue devtools可以让你更方便的测试。
- 合理的 CSS 结构，如 BEM 或 oocss - 详情 ?; 。
- 使用单文件 .Vue 文件格式来组件代码。

怎么做——组件结构化

```
<template lang="html">
    <div class="Ranger__Wrapper">
        <!-- ... -->
    </div>
</template>

<script type="text/javascript">
  export default {
        // 不要忘记了 name 属性
    name: 'RangeSlider',
    // 组合其它组件
    extends: {},
    // 组件属性、变量
    props: {
            bar: {}, // 按字母顺序
            foo: {},
            fooBar: {},
        },
    // 变量
    data() {},
    computed: {},
    // 使用其它组件
    components: {},
    // 方法
    watch: {},
    methods: {},
    // 生命周期函数
    beforeCreate() {},
    mounted() {},
};
</script>

<style scoped>
  .Ranger__Wrapper { /* ... */ }
</style>
```


组件事件命名

Vue.js 提供的处理函数和表达式都是绑定在 ViewModel 上的，组件的每一个事件都应该按照一个好的命名规范来，这样可以避免不少的开发问题，具体可见如下 *为什么*。

为什么？

- 开发者可以随意给事件命名，即使是原生事件的名字，这样会带来迷惑性。
- 过于宽松的事件命名可能与DOM模板不兼容。

怎么做？

- 事件命名也连字符命名

- 一个事件的名字对应组件外的一组意义操作，如： `upload-success`，`upload-error` 以及 `dropzone-upload-success`，`dropzone-upload-error` （如果需要前缀的话）。
- 事件命名应该以动词（如 `client-api-load` ）或是形容词（如 `drive-upload-success` ）结尾。（出处）

避免this.$parent

Vue.js 支持组件嵌套，并且子组件可访问父组件的上下文。访问组件之外的上下文违反了基于模块开发的第一原则。因此你应该尽量避免使用 `this.$parent` 。

为什么？

- 组件必须相互保持独立，Vue 组件也是。如果组件需要访问其父层的上下文就违反了该原则。
- 如果一个组件需要访问其父组件的上下文，那么该组件将不能再其它上下文中复用。

怎么做？

- 通过 props 将值传递给子组件
- 通过 props 传递回调函数给子组件来达到调用父组件方法的目的
- 通过在子组件触发事件来通知父组件

谨慎使用 this.$refs

Vue.js 支持通过 ref 属性来访问其它组件和 HTML 元素。并通过 `this.$refs` 可以得到组件或 HTML 元素的上下文。在大多数情况下，通过 `this.$refs` 来访问其它组件的上下文是可以避免的。在使用的的时候你需要注意避免调用了不恰当的组件 API，所以应该尽量避免使用 `this.$refs` 。

为什么？

- 组件必须是保持独立的，如果一个组件的 API 不能够提供所需的功能，那么这个组件在设计、实现上是有问题的。
- 组件的属性和事件必须足够的给大多数的组件使用。

怎么做？

- 提供良好的组件 API。
- 总是关注于组件本身的目的。
- 拒绝定制代码。如果你在一个通用的组件内部编写特定需求的代码，那么代表这个组件的 API 不够通用，或者你可能需要一个新的组件来应对该需求。
- 检查所有的 props 是否有缺失的，如果有提一个 issue 或是完善这个组件。
- 检查所有的事件。子组件向父组件通信一般是通过事件来实现的，但是大多数的开发者更多的关注于 props 从忽视了这点。
- Props向下传递，事件向上传递！以此为目标升级你的组件，提供良好的 API 和 独立性。

- 当遇到 props 和 events 难以实现的功能时，通过 `this.$refs` 来实现。
- 当需要操作 DOM 无法通过指令来做的时候可使用 `this..$ref` 而不是 `JQuery`，`document.getElement*` ，`document.queryElement`。

```
 <!-- 推荐，并未使用 this.$refs -->
 <range :max="max"
  :min="min"
  @current-value="currentValue"
  :step="1"></range>
<!-- 使用 this.$refs 的适用情况-->
<modal ref="basicModal">
  <h4>Basic Modal</h4>
  <button class="primary"
@click="$refs.basicModal.close()">Close</button>
</modal>
<button @click="$refs.basicModal.open()">Open modal</button>

<!-- Modal component -->
<template>
  <div v-show="active">
    <!-- ... -->
  </div>
</template>

<script>
  export default {
    // ...
    data() {
        return {
            active: false,
        };
    },
    methods: {
      open() {
        this.active = true;
      },
      hide() {
        this.active = false;
      },
    },
    // ...
  };
</script>
<!-- 如果可通过 emited 来做则避免通过 this.$refs 直接访问 -->
<template>
  <range :max="max"
    :min="min"
    ref="range"
    :step="1"></range>
</template>
```

```
<script>
  export default {
    // ...
    methods: {
      getRangeCurrentValue() {
        return this.$refs.range.currentValue;
      },
    },
    // ...
  };
</script>
```

使用组件名作为样式作用域空间

Vue.js 的组件是自定义元素，这非常适合用来作为样式的根作用域空间。可以将组件名作为 css 类的命名空间。

为什么？

- 给样式加上作用域空间可以避免组件样式影响外部的样式
- 保持模块名、目录名、样式根作用域名一样，可以很好的将其关联起来，便于开发者理解。

怎么做？

使用组件名作为样式命名的前缀，可基于 BEM 或 OOCSS 范式。同时给style标签加上 scoped 属性。加上 scoped 属性编译后会给组件的 class 自动加上唯一的前缀从而避免样式的冲突。

```
<style scoped>
    /* 推荐 */
    .MyExample { }
    .MyExample li { }
    .MyExample__item { }

    /* 避免 */
    .My-Example { } /* not scoped to component or module name,
not BEM compliant */
</style>
```

提供组件 API 文档

使用 Vue.js 组件的过程中会创建 Vue 组件实例，这个实例是通过自定义属性配置的。为了便于其他开发者使用该组件，对于这些自定义属性即组件API应该在 README.md 文件中进行说明。

为什么？

- 良好的文档可以让开发者比较容易的对组件有一个整体的认识，而不用去阅读组件的源码，也更方便开发者使用。
- 组件配置属性即组件的 API，对于组件的用户来说他们更感兴趣的是 API 而不是实现原理。
- 正式的文档会告诉开发者组件 API 变更以及向后的兼容性情况。
- README.md 是标准的我们应该首先阅读的文档文件。代码托管网站 (github/bitbucket/gitlab 等) 会默认在仓库中展示该文件作为仓库的介绍。

怎么做？

在模块目录中添加 README.md 文件：

```
range-slider/
├─── range-slider.Vue
├─── range-slider.less
└─── README.md
```

在 README 文件中说明模块的功能以及使用场景。对于 Vue 组件来说，比较有用的描述是组件的自定义属性即 API 的描述介绍。

如何提高自己的写组件能力

1. 第一，多写，多练，从简单的开始写，然后做总结，不停的换写法，写到自己的很满意为止，我可以这么说，就光上面那个login.Vue组件，我写了五次，最后一次才让我满意，如何分分合合，这个也是对组件层面上一个考验

2. 看他人源码，比方说mintui，iview他们组件如何去组件的，通过看开源组件库的代码你会从中很多收益，你会发现有些不常的特性，是如次的牛B，如何之方便，你就能在组件层面上有更大的提高，渐渐的你的价值也会体现，无论谁的Vue ui框架库再好，也没有自己写一个好，因为功能是千变万化的，你掌握了组件编写能力，你就主动了，不是被动了，发现别人的ui逻辑和样式不适合自己，怎么办，轮子麻，自行轮子拿 过来改改加加厚那就是摩托车轮子，单车变摩托麻。

我觉得吧！无论学什么，学好基础才是最重要，我的学习方式是从浅入深，而不是深入浅出，只有基础打的好，才会有更好的解决方案，Vue一切的组件功能都围绕着基础。