# **Vue Render介绍和一些基本的实例**

为啥研究这个？在之前开发组件库的过程中，遇到了许多遗留的问题，包括数据模板渲染、组件按需加载、引入自定义组件插槽等等，所以为了修复和避免这些问题，学习一波更接近编译器的编写方式，看看如何通过这种完全编程方式来解决一波这些问题~当然这里只是一些最基本的使用和探索，因为官网例子太少了，只能一个个自己搭=。=

Vue 推荐在绝大多数情况下使用 template 来创建你的 HTML。然而在一些场景中，你真的需要 JavaScript 的完全编程的能力，这就是****render 函数****，它比 template 更接近编译器。（****从官网复制的，慌得一批，其实简单来说就是以函数的方式写HTML，可控性更强一些~****）

当然，官网已经给出了一个使用template来编写的不方便的demo，所以在这里就不反复提起了，初次使用或者有兴趣的大佬可以直接戳这个链接了解一下~[Vue Render](https://link.juejin.im/?target=https://cn.vuejs.org/v2/guide/render-function.html%23%E5%9F%BA%E7%A1%80" \o "Vue Render" \t "https://juejin.im/post/_blank)

### 本篇文章主要介绍了以下几点

了解基本概念的客官可以直接下拉到实例，实例已上传github

* 虚拟DOM
* CreateElement函数
* 【起步】最基本的实例
* 【进阶】包含属性配置较完整的实例
* 【进阶】CreateElement中slot属性的用法
* 【深入】CreateElement中scopedSlots的用法
* 【换口气儿】Render中的JSX配置和用法
* 【深入】函数式组件

### [Github传送门](https://link.juejin.im/?target=https://github.com/godaangel/vue-render-test.git" \t "https://juejin.im/post/_blank)

### 虚拟DOM

##### 1、****什么是DOM？****

DOM 就是浏览器解析 HTML 得来的一个树形逻辑对象。

##### 2、****什么是虚拟DOM？****

用 Object 来代表一个节点，这个节点叫做虚拟节点（ Virtual Node ）简写为 VNode，由 VNode 树组成虚拟DOM。

##### 3、****为什么要用虚拟DOM？****

Web 页面的大多数操作和逻辑的本质就是不停地修改 DOM 元素，但是 DOM 操作太慢了，过于频繁的 DOM 操作可能会导致整个页面掉帧、卡顿甚至失去响应。仔细想一想，很多 DOM 操作是可以打包（多个操作压成一个）和合并（一个连续更新操作只保留最终结果）的，同时 JS 引擎的计算速度要快得多，所以为什么不把 DOM 操作先通过JS计算完成后统一来一次大招操作DOM呢，于是就有了虚拟DOM的概念。当然，虚拟DOM操作的核心是Diff算法，也就是比较变化前后Vnode的不同，计算出最小的DOM操作来改变DOM，提高性能。

##### 4、****Vue里面的虚拟DOM怎么生成的呢？****

通过`createElement(tag, options, VNodes)`，下面就来介绍这个函数的基本概念。

### CreateElement 函数

简单来说CreateElement就是用来生成Vnode的函数

CreateElement 到底会返回什么呢？其实不是一个实际的 DOM 元素（返回的是Vnode）。它更准确的名字可能是 createNodeDescription，因为它所包含的信息会告诉 Vue 页面上需要渲染什么样的节点，及其子节点。

****【Tips】**** CreateElement****函数在惯例中通常也写作****h

##### 1、****CreateElement的参数如下所示****：（~~太懒了直接搬的官网~~）

// @returns {VNode}

createElement(

// {String | Object | Function}

// 一个 HTML 标签字符串，组件选项对象，或者 解析上述任何一种的一个 async 异步函数，必要参数。

'div',

// {Object}

// 一个包含模板相关属性的数据对象

// 这样，您可以在 template 中使用这些属性。可选参数。

{

// 详情见下方

},

// {String | Array}

// 子节点 (VNodes)，由 `createElement()` 构建而成，或使用字符串来生成“文本节点”。可选参数。

[

'先写一些文字',

createElement('h1', '一则头条'),

createElement(MyComponent, {

props: {

someProp: 'foobar'

}

})

]

)复制代码

****【Tips】**** 文档中此处说VNodes子节点必须是唯一的，也就是说第三个参数的Array里不能出现相同指向的VNodes，实际验证以后，就算写重复的VNodes，也并不会报错，估计此处会有些坑，现在还没踩到，建议按照文档要求，保持子节点唯一。

##### 2、Vnode属性配置（第二个参数）：(~~太懒了也是直接搬，捂脸.jpg~~)

以下属性为简单介绍，具体用法和一些 \_备注解释 \_可以参考后面会讲到的【包含属性配置较完整的实例】

{

// 和`v-bind:class`一样的 API

// 接收一个字符串、对象或字符串和对象组成的数组

'class': {

foo: true,

bar: false

},

// 和`v-bind:style`一样的 API

// 接收一个字符串、对象或对象组成的数组

style: {

color: 'red',

fontSize: '14px'

},

// 正常的 HTML 特性

attrs: {

id: 'foo'

},

// 组件 props

props: {

myProp: 'bar'

},

// DOM 属性

domProps: {

innerHTML: 'baz'

},

// 事件监听器基于 `on`

// 所以不再支持如 `v-on:keyup.enter` 修饰器

// 需要手动匹配 keyCode。

on: {

click: **this**.clickHandler

},

// 仅对于组件，用于监听原生事件，而不是组件内部使用

// `vm.$emit` 触发的事件。

nativeOn: {

click: **this**.nativeClickHandler

},

// 自定义指令。注意，你无法对 `binding` 中的 `oldValue`

// 赋值，因为 Vue 已经自动为你进行了同步。

directives: [

{

name: 'my-custom-directive',

value: '2',

expression: '1 + 1',

arg: 'foo',

modifiers: {

bar: true

}

}

],

// 作用域插槽格式

// { name: props => VNode | Array<VNode> }

scopedSlots: {

default: props => createElement('span', props.text)

},

// 如果组件是其他组件的子组件，需为插槽指定名称

slot: 'name-of-slot',

// 其他特殊顶层属性

key: 'myKey',

ref: 'myRef'

}复制代码

### 【起步】最基本的实例

这是一个基础的Demo，包含了

简单的渲染用法

标签

props

slot

点击事件

以下示例Demo均采用单文件组件的方式，工程用vue-cli搭建的webpack-simple工程。

****组件****wii-first

<script>

export default {

name: 'wii-first',

data() {

return {

msg: 0

}

},

props: {

level: {

type: [Number, String],

required: true

}

},

render: function(createElement) {

this.$slots.subtitle = this.$slots.subtitle || []

// this.level = 1时, 等价于

// <h1 class="wii-first">

// 第一个组件, <slot></slot>

// <slot name="subtitle"></slot>，此处是data的值: {{msg}}

// <button @click="clickHandler">点我改变内部data值</button>

// </h1>

return createElement(

'h' + this.level, // tag name 标签名称

{

class: 'wii-first'

},

// this.$slots.default, // 子组件中的slot 单个传递

// this.$slots.subtitle,

[

'第一个组件, ',

...this.$slots.default, // 默认slots传递

...this.$slots.subtitle, // 具名slots传递

'，此处是data的值: ',

this.msg,

createElement('button', {

on: {

click: this.clickHandler

},

}, '点我改变内部data值')

]

)

},

methods: {

clickHandler() {

this.msg = Math.ceil(Math.random() \* 1000)

}

}

}

</script>复制代码

【Tips】：CreateElement的第三个参数在文档中规定组件树中的所有 VNode 必须是唯一的，也就是说在第三个参数中有两个指向相同的Vnode是无效的。但经过实践发现，实际上是可以渲染出来的，在此不推荐这么写哦，可能会掉到不可预料的大坑hiahiahia~

****引入方式****

<template>

<div id="app">

<wii-first level="1">我是标题 <span slot="subtitle">我是subtitle</span></wii-first>

</div>

</template>

<script>

import WiiFirst from './components/first/index.vue'

export default {

name: 'app',

components: {

WiiFirst

},

data() {

return {

}

}

}

</script>复制代码

### 【进阶】包含属性配置较完整的实例

这个Demo主要展示了createElement属性用法，包含

* class，style，attrs，on等
* 包含了点击事件以及click.stop的转换示例

不包含

* v-for v-if v-model的实现，这些在官网都有很详细的说明了 [传送门](https://link.juejin.im/?target=https://cn.vuejs.org/v2/guide/render-function.html%23v-if-%E5%92%8C-v-for" \t "https://juejin.im/post/_blank)

****组件****wii-second

export default {

name: 'wii-second',

data() {

return {

myProp: '我是data的值, 只是为了证明props不是走这儿'

}

},

props: {

},

render: function(createElement) {

// 等价于

// <div id="second" class="wii-second blue-color" style="color: green;" @click="clickHandler">

// 我是第二个组件测试, 点我触发组件内部click和外部定义的@click.native事件。

// <div>{{myProp}}</div>

// <button @click="buttonClick">触发emit</button>

// </div>

return createElement(

'div', {

//【class】和`v-bind:class`一样的 API

// 接收一个字符串、对象或字符串和对象组成的数组

// class: 'wii-second',

// class: {

// 'wii-second': true,

// 'grey-color': true

// },

class: [{

'wii-second': true

}, 'blue-color'],

//【style】和`v-bind:style`一样的 API

// 接收一个字符串、对象或对象组成的数组

style: {

color: 'green'

},

//【attrs】正常的 HTML 特性, id、title、align等，不支持class，原因是上面的class优先级最高[仅猜测]

// 等同于DOM的 Attribute

attrs: {

id: 'second',

title: '测试'

},

// 【props】组件 props，如果createElement定义的第一个参数是组件，则生效，此处定义的数据将被传到组件内部

props: {

myProp: 'bar'

},

// DOM 属性 如 value, innerHTML, innerText等, 是这个DOM元素作为对象, 其附加的内容

// 等同于DOM的 Property

// domProps: {

// innerHTML: 'baz'

// },

// 事件监听器基于 `on`, 用于组件内部的事件监听

on: {

click: this.clickHandler

},

// 仅对于组件，同props，等同@click.native，用于监听组件内部原生事件，而不是组件内部使用 `vm.$emit` 触发的事件。

// nativeOn: {

// click: this.nativeClickHandler

// },

// 如果组件是其他组件的子组件，需为插槽指定名称，见 wii-third 组件

// slot: 'testslot',

// 其他特殊顶层属性

// key: 'myKey',

// ref: 'myRef'

}, [

`我是第二个组件测试, 点我触发组件内部click和外部定义的@click.native事件。`,

createElement('div', `${this.myProp}`),

createElement('button', {

on: {

click: this.buttonClick

}

}, '触发emit')

]

)

},

methods: {

clickHandler() {

console.log('我点击了第二个组件，这是组件内部触发的事件')

},

buttonClick(e) {

e.stopPropagation() // 阻止事件冒泡 等价于 click.stop

console.log('我点击了第二个组件的button，将会通过emit触发外部的自定义事件')

this.$emit('on-click-button', e)

}

}

}复制代码

****引入方式****

<template>

<div id="app">

<wii-second @click.native="nativeClick" @on-click-button="clickButton"></wii-second>

</div>

</template>

<script>

import WiiSecond from './components/second/index.vue'

export default {

name: 'app',

components: {

WiiSecond

},

data() {

return {

}

},

methods: {

nativeClick() {

console.log('这是组件外部click.native触发的事件，第二个组件被点击了')

},

clickButton() {

console.log('这是组件外部触发的【emit】事件，第二个组件被点击了')

}

}

}

</script>复制代码

#### 事件 & 按键修饰符

上面例子中用到了e.stopPropagation这个方法，等价于 template 模板写法的click.stop，其他的事件和按键修饰符也有对应的方法，对应情况如下。

****事件修饰符对应的前缀****

| **template事件修饰符** | **render写法前缀** |
| --- | --- |
| .passive | & |
| .capture | ! |
| .once | ~ |
| .capture.once 或 .once.capture | ~! |

例如

on: {

'!click': **this**.doThisInCapturingMode,

'~keyup': **this**.doThisOnce,

'~!mouseover': **this**.doThisOnceInCapturingMode

}复制代码

****其他事件修饰符，对应的事件处理函数中使用事件方法****

| **template事件修饰符** | **对应的事件方法** |
| --- | --- |
| .stop | event.stopPropagation() |
| .prevent | event.preventDefault() |
| .self | if (event.target !== event.currentTarget) return |
| Keys: .enter, .13 | if (event.keyCode !== 13) return (对于其他的键盘事件修饰符，将13换成其他的键盘code就行) |
| Modifiers Keys: .ctrl, .alt, .shift, .meta | if (!event.ctrlKey) return (将 ctrlKey 换成 altKey, shiftKey, 或者 metaKey, respectively) |

例如

on: {

keyup: **function** (event) {

// 如果触发事件的元素不是事件绑定的元素

// 则返回

**if** (event.target !== event.currentTarget) **return**

// 如果按下去的不是 enter 键或者

// 没有同时按下 shift 键

// 则返回

**if** (!event.shiftKey || event.keyCode !== 13) **return**

// 阻止 事件冒泡

event.stopPropagation()

// 阻止该元素默认的 keyup 事件

event.preventDefault()

// ...

}

}复制代码

### 【进阶】CreateElement中slot属性的用法

这个Demo主要展示render中createElement的配置slot属性用法。

因为此处一直很疑惑什么情况下能够用到这个slot属性，所以就试了一下，仅供参考，具体使用场景需要根据业务逻辑来定

****组件****wii-third

<script>

export default {

name: 'wii-third',

data() {

return {}

},

components: {

WiiTestSlot: {

name: 'wii-test-slot',

render(createElement) {

this.$slots.testslot = this.$slots.testslot || []

// 等价于

// <div>

// 第三个组件，测试在组件中定义slot, <slot name="testslot"></slot>

// </div>

return createElement(

'div', [

'第三个组件，测试在组件中定义slot, ',

...this.$slots.testslot

]

)

}

},

WiiTestSlotIn: {

name: 'wii-test-slot-in',

render(createElement) {

// 等价于

// <span>我是组件中的slot内容</span>

return createElement(

'span', [

'我是组件中的slot内容'

]

)

}

}

},

props: {

},

render: function(createElement) {

// 等价于

// <div style="margin-top: 15px;">

// <wii-test-slot>

// <wii-test-slot-in slot="testslot"></wii-test-slot-in>

// </wii-test-slot>

// </div>

return createElement(

'div', {

style: {

marginTop: '15px'

}

}, [

createElement(

'wii-test-slot',

//这么写不会被渲染到节点中去

// createElement(

// 'wii-test-slot-in',

// {

// slot: 'testslot'

// }

// ),

[

// createElement再放createElement需要放入数组里面，建议所有的组件的内容都放到数组里面，统一格式，防止出错

createElement(

'wii-test-slot-in', {

slot: 'testslot'

}

)

]

)

]

)

},

methods: {

}

}

</script>复制代码

【Tips】：如果createElement里面的第三个参数传递的是createElement生成的VNode对象，将不会被渲染到节点中，需要放到数组中才能生效，此处猜测是因为VNode对象不会被直接识别，因为文档要求是String或者Array。

****引入方式****

<template>

<div id="app">

<wii-third></wii-third>

</div>

</template>

<script>

import WiiThird from './components/third/index.vue'

export default {

name: 'app',

components: {

WiiThird

},

data() {

return {}

}

}

</script>复制代码

### 【深入】CreateElement中scopedSlots的用法

这个Demo主要展示scopedSlots的用法，包括定义和使用。scopedSlots的template用法和解释参考[vue-slot-scope](https://link.juejin.im/?target=https://cn.vuejs.org/v2/guide/components-slots.html%23%E8%A7%A3%E6%9E%84-slot-scope" \t "https://juejin.im/post/_blank)。

****组件****wii-forth

<script>

export default {

name: 'wii-forth',

data() {

return {}

},

components: {

WiiScoped: {

name: 'wii-scoped',

props: {

message: String

},

render(createElement) {

// 等价于 <div><slot :text="message"></slot></div>

return createElement(

'div', [

this.$scopedSlots.default({

text: this.message

})

]

)

}

}

},

render: function(createElement) {

// 等价于

// <div style="margin-top: 15px;">

// <wii-scoped message="测试scopedSlots，我是传入的message">

// <span slot-scope="props">{{props.text}}</span>

// </wii-scoped>

// </div>

return createElement(

'div', {

style: {

marginTop: '15px'

}

}, [

createElement('wii-scoped', {

props: {

message: '测试scopedSlots，我是传入的message'

},

// 传递scopedSlots，通过props(自定义名称)取值

scopedSlots: {

default: function(props) {

return createElement('span', props.text)

}

}

})

]

)

}

}

</script>复制代码

****引入方法****

<template>

<div id="app">

<wii-forth></wii-forth>

</div>

</template>

<script>

import WiiForth from './components/forth/index.vue'

export default {

name: 'app',

components: {

WiiForth

},

data() {

return {}

}

}

</script>复制代码

### 【换口气儿】Render中的JSX配置和用法

写了这么多createElement，眼睛都花了，有的写起来也挺麻烦的。我们试试来换个口味，试试JSX的写法。

工欲善其事，必先利其器。

* 首先我们安装vue写JSX必要的依赖：

npm install babel-plugin-syntax-jsx babel-plugin-transform-vue-jsx babel-helper-vue-jsx-merge-props babel-preset-env --save-dev复制代码

安装完成以后，在.babelrc文件配置"plugins": ["transform-vue-jsx"]。

将webpack配置文件中的js解析部分改成test: /\.jsx?$/表示对jsx的代码块进行解析。

****这个示例做了如下功能：****

1. 通过props传入颜色，并在子组件配置
2. 点击改变颜色
3. 点击事件通过native获取

****父组件****wii-jsx

<script type="text/jsx">

**import** WiiJsxItem **from** './item.vue'

**export** **default** {

name: 'wii-jsx',

components: {

WiiJsxItem

},

data() {

**return** {

color: 'red'

}

},

props: {

},

render: **function** (h) {

**return** (

<div class="wii-jsx">

<wii-jsx-item color={this.color} nativeOnClick={this.clickHandler}>

<span>我是wii-jsx-item组件的slot, color通过变量传入: {this.color}</span>

</wii-jsx-item>

</div>

)

},

methods: {

clickHandler() {

this.color = this.color == 'red' ? 'blue' : 'red'

console.log(`点击了wii-jsx-item，通过native触发，改变了颜色为【${this.color}】`)

}

}

}</script>复制代码

****子组件****wii-jsx-item

该子组件在父组件中被引入，并用JSX的写法渲染。

**export** **default** {

name: 'wii-jsx-item',

data() {

**return** {}

},

props: {

color: String

},

render: **function**(createElement) {

// 等价于 <div class="wii-jsx-item"><slot></slot></div>

**return** createElement(

'div', {

class: 'wii-jsx-item',

style: {

color: **this**.color

}

},

**this**.$slots.default

)

},

methods: {

}

}复制代码

****引入方式****

<template>

<div id="app">

<wii-jsx></wii-jsx>

</div></template>

<script>**import** WiiJsx **from** './components/jsx/index.vue'

**export** **default** {

name: 'app',

components: {

WiiJsx

},

data() {

**return** {}

}

}复制代码

JSX的主要转换还是依靠我们之前安装的[babel插件](https://link.juejin.im/?target=https://github.com/vuejs/babel-plugin-transform-vue-jsx" \t "https://juejin.im/post/_blank)，而JSX的事件以及属性的用法见babel插件的[使用说明](https://link.juejin.im/?target=https://github.com/vuejs/babel-plugin-transform-vue-jsx%23usage" \t "https://juejin.im/post/_blank)，这里面包含了vue里面事件和属性对应的用法说明。

### 【深入】函数式组件

下面来进行最后一个模块的介绍，函数式组件functional，这个东西的用法就见仁见智了，这里也没啥好的方案，只是给出了一些示例，各位大佬如果有一些具体的使用到的地方，阔以指点一下哇~thx~(~~害羞.jpg~~)。

官方文档的定义是functional组件需要的一切都是通过上下文传递，包括：

* props：提供所有 prop 的对象
* children: VNode 子节点的数组
* slots: 返回所有插槽的对象的函数
* data：传递给组件的数据对象，作为 createElement 的第二个参数传入组件
* parent：对父组件的引用
* listeners: (2.3.0+) 一个包含了所有在父组件上注册的事件侦听器的对象。这只是一个指向 data.on 的别名。
* injections: (2.3.0+) 如果使用了 inject 选项，则该对象包含了应当被注入的属性。

\_在添加 \_functional: true之后，组件的render函数会增加第二个参数context（第一个是createElement），数据和节点通过context传递。

****Tips：****

在 2.3.0 之前的版本中，如果一个函数式组件想要接受 props，则props选项是必须的。在 2.3.0 或以上的版本中，你可以省略props选项，所有组件上的属性都会被自动解析为 props。

****我个人的理解是：****

Functional相当于一个纯函数一样，内部不存储用于在界面上展示的数据，传入什么，展示什么，传入的是相同的数据，展示的必然是相同的。无实例，无状态，没有this上下文，均通过context来控制。

****优点：****

因为函数式组件只是一个函数，所以渲染开销低很多。

****使用场景：****

1. 仅仅作为接收参数用的组件（不做任何管理和监听状态）
2. 目前看大多数使用场景是用来包装动画组件，因为动画组件不需要状态管理这些
3. 程序化地在多个组件中选择一个（官方）
4. 在将 children, props, data 传递给子组件之前操作它们（官方）

接下来就通过两个组件来看看如何使用的吧，这里也仅仅只是示例而已，使用的场景仍在探索中，具体的使用场景还需要在开发过程中根据需求复杂的和性能要求来酌情选择~

#### 函数式组件一 wii-functional用在动画的functional

这个Demo的作用是在输入框中输入字符，对数据列表进行筛选，筛选时加入显示和消失的动画。

****组件主体****

<script>

import Velocity from 'velocity-animate' // 这是一个动画库

export default {

name: 'wii-functional',

functional: true, //表明是函数式组件

render: function(createElement, context) {

// context是在functional: true时的参数

let data = {

props: {

tag: 'ul',

css: false

},

on: {

// 进入前事件

beforeEnter: function(el) {

el.style.opacity = 0

el.style.height = 0

},

// 进入事件

enter: function(el, done) {

let delay = el.dataset.index \* 150

setTimeout(function() {

Velocity(el, {

opacity: 1,

height: '1.6em'

}, {

complete: done

})

}, delay)

},

// 离开事件

leave: function(el, done) {

let delay = el.dataset.index \* 150

setTimeout(function() {

Velocity(el, {

opacity: 0,

height: 0

}, {

complete: done

})

}, delay)

}

}

}

return createElement('transition-group', data, context.children)

}

}

</script>复制代码

上面这个组件相当于创建了一个ul-li标签组成的vue动画，通过functional方式包裹到组件外部，可以作为通用的动画。

****引入方式****

<template>

<div id="app">

<input v-model="query"/>

<wii-functional>

<li v-for="(item, index) in computedList"

:key="item.msg"

:data-index="index">

{{item.msg}}

</li>

</wii-functional>

</div></template>

<script>**import** WiiFunctional **from** './components/functional/index.vue'

**export** **default** {

name: 'app',

components: {

WiiFunctional

},

data() {

**return** {

// 关键字

query: '',

// 数据列表

list: [{

msg: 'Bruce Lee'

}, {

msg: 'Jackie Chan'

}, {

msg: 'Chuck Norris'

}, {

msg: 'Jet Li'

}, {

msg: 'Kung Furry'

}, {

msg: 'Chain Zhang'

}, {

msg: 'Iris Zhao'

}, ]

}

},

computed:{

computedList: **function**() {

**var** vm = **this**

// 过滤出符合条件的查询结果

**return** **this**.list.filter(**function**(item) {

**return** item.msg.toLowerCase().indexOf(vm.query.toLowerCase()) !== -1

})

}

},

watch: {

computedList(newVal, oldVal) {

console.log(newVal)

}

}

}</script>复制代码

#### 函数式组件二 wii-choose-comp用在组件切换的functional

在这个示例中，通过props来切换加载不同的组件，并且在props传递给子组件之前操作它，组件内部定义了click.native事件来展示示例。如果对一批组件进行同样的操作，则可以用这个functional，类似于加工厂。

当然如果组件需要不同的点击事件或者表现方式也可以在各个组件内部单独写逻辑或者监听~因为wii-choose-comp这个外壳本质不过就是个函数而已~

****组件主体**wii-choose-comp**

<script>

export default {

name: 'wii-choose-comp',

functional: true,

props: { // 2.3.0版本以上也可以不写props，会将组件属性默认绑定成props，为了统一标准还是写上

componentName: String // 组件名

},

render: function(createElement, context) {

// 给组件加上class

context.data.class = [context.props.componentName]

// 在props传给子组件之前操作它

context.data.props = {

compName: context.props.componentName

}

context.data.nativeOn = {

click() {

alert('我是functional里面统一的点击事件')

}

}

return createElement(context.props.componentName, context.data, context.children)

}

}

</script>复制代码

****切换组件1**wii-comp-one**

<script>

export default {

name: 'wii-comp-one',

props: {

compName: String

},

render: function(createElement) {

return createElement('div', [

'我是第一个comp, 我有点击效果, ',

`我的名字叫${this.compName}, `,

...this.$slots.default

])

}

}

</script>复制代码

****切换组件2**wii-comp-two**

<script>

export default {

name: 'wii-comp-two',

props: {

compName: String

},

render: function(createElement) {

return createElement('div', [

'我是第二个comp, 点我试试呗, ',

`我的名字叫${this.compName}, `,

...this.$slots.default

])

}

}

</script>复制代码

****引入方式****

<template>

<div id="app">

<button @click="changeComponent">点击切换组件</button>

<wii-choose-comp :component-name="componentName">

<span>我是{{componentName}}的slot</span>

</wii-choose-comp>

</div>

</template>

<script>

import WiiChooseComp from './components/functional/chooseComp.vue'

import WiiCompOne from './components/functional/comp1.vue'

import WiiCompTwo from './components/functional/comp2.vue'

export default {

name: 'app',

components: {

WiiChooseComp,

WiiCompOne,

WiiCompTwo

},

data() {

return {

componentName: 'wii-comp-one'

}

},

methods: {

changeComponent() {

this.componentName = this.componentName == 'wii-comp-one' ? 'wii-comp-two' : 'wii-comp-one'

}

}

}

</script>复制代码

****【Tips】**** 需要将待切换的组件全部引入到外层。（不造有没有更好的办法？）

### 总结

以上就是最近对Vue Render的一个探索，因为对于公共组件库开发来说，需要考虑的问题有很多，所以灵活性要求也更高，如果用Vue Render这种更接近编译的方式来编写组件库，可能会让逻辑更清晰，虽然不停的创建元素的写法是挺恶心的哈哈哈哈~~