### 什么是vue

把一个 json对象的数据，显示到一个元素上去。

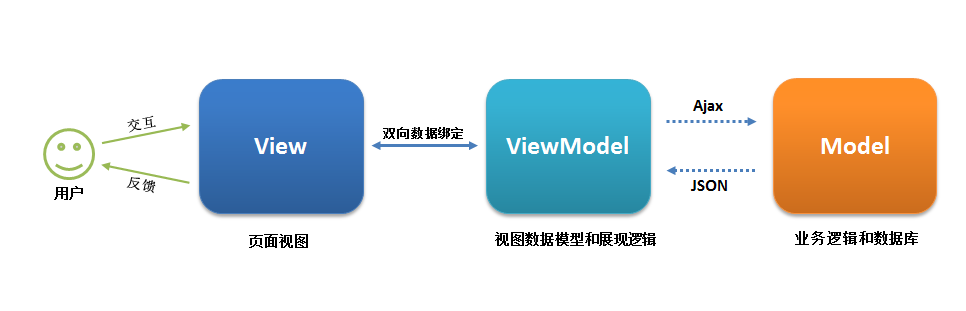
如果使用Vue, 那么仅仅需要提供数据，以及数据要绑定到的元素的id,就行了,不需要显式地操作HTML DOM。

MVVM:model层(储存数据的模型)，view层（显示数据），viewmodle层

model层：业务逻辑和实体模型(biz/bean)，泛指后端进行的各种业务逻辑处理和数据操控

view: 布局文件(XML)，典型的如HTML,CSS等模板文件。

ViewModel负责把Model的数据同步到View显示出来，还负责把View的修改同步回Model。并且实现了和view层的双向绑定。



# VUE基础

## Vue基本格式

new Vue({

el: '#div1',//el代表element，通过元素id绑定

data: {//定义元素属性

clickNumber:0

},

methods:{//作用在该元素上的方法

count: function(){

this.clickNumber++;

}

}

})

### 指令

指令是带有 v- 前缀的特殊属性。

指令用于在表达式的值改变时，将某些行为应用到 DOM 上。

## Vue.js 条件语句

### v-if

为true时显示该元素，否则隐藏

通过toggle函数切换show的值。 通过v-if 语句，当show 是true的时候，显示当前元素

<div v-if="show"> 默认这一条是看得见的</div>

new Vue({

el: '#div1',

data: {

show:true

},

methods:{

toggle: function(){

this.show=!this.show;

}

}

})

### v-else

当v-if为false时自动显示//在同一父类元素下才有用

### v-else-if

//在同一父类元素下才有用

<div v-if="number>98"> 神仙</div>

<div v-else-if="number>95"> 国家领导人</div>

## Vue.js 循环语句

首选准备一个数组，然后在构建Vue的时候把这个数组作为参数传递进去，最后在视图上，通过v-for 遍历这个数组。

data: {

heros:arrayname

}

<tr v-for="hero in heros">

<td>{{hero.name}}</td>

<td>{{hero.hp}}</td>

</tr>

遍历的时候也可以获取下标。

<tr v-for="hero,index in heros">

如果是纯数字遍历可直接

<div v-for="i in 10">

     {{ i }}

    </div>

## Vue.js 属性绑定

### v-bind

通过v-bind进行属性绑定

<a v-bind:href="page">http://12306.com</a>

new Vue({

el: '#div1',

data:{

page:'http://12306.com'

}

})

v-bind:href 简写成 :href

## Vue.js 计算属性

computed

把运算过程，都放在computed里去，只用显示运算结果就好了。

new Vue({

el: '#div1',

data: {

exchange:6.4,

rmb:0

},

computed:{

dollar:function() {

return this.rmb / this.exchange;

}

}

})

当然，method也能达到该效果。

区别：**computed 是有缓存的，只要rmb没有变化**，dollar 会直接返回以前计算出来的值，而**不会再次计算**。 这样如果是复杂计算，就会节约不少时间。

而methods每次都会调用重新计算。

## Vue.js 监听属性

watch也可实现随数据变动而变动的功能

new Vue({

el: '#div1',

data: {

exchange:6.4,

rmb:0,

dollar:0

},

watch:{

rmb:function(val) {

this.rmb = val;

this.dollar = this.rmb / this.exchange;

},

dollar:function(val) {

this.dollar = val;

this.rmb = this.dollar \* this.exchange;

},

}

})

## Vue.js 组件

组件即一个模板，只需修改一下参数便可有不同展示。类似于可多次创建的vue模板实例。

### 全局组件

Vue.component('componentName', {

template: '<div >MAXFEEL休闲男士手包真皮手拿包大容量信封包手抓包夹包软韩版潮</div>'

})

new Vue({

el: '#div1'

})

<div id="div1">

< componentName t></ componentName >

< componentName t></ componentName >

< componentName t></ componentName >

</div>

### 局部组件

在vue实例里添加compontents

components:{

'product':{

template:'<div >MAXFEEL休闲男士手包真皮手拿包大容量信封包手抓包夹包软韩版潮</div>'

}

**在vue实例中的template会进行编译，并将编译后的虚拟dom(即在查看dom时没有该标签)取代该vue实例原来绑定的元素（即el绑定的元素）。**

### 参数

设置参数props(英语properties), 并且在组件里使用这个props，就可以传递不同内容给组件。

Vue.component('product', {

props:['name'],

template: '<div >{{name}}</div>'

})

new Vue({

el: '#div1'

})

<div id="div1">

<product name="请输入什么随便内容 "></product>

</div>

一个组件默认可以拥有任意数量的 prop，任何值都可以传递给任何 prop。在上述模板中，你会发现我们能够在组件实例中访问这个值，就像访问 data 中的值一样。作为一个自定义 attribute 传递

### 动态参数

通过v-bind关联组件内和组件外的值。

<product v-bind:name="outName"></product>

例：

<div id="div1">

组件外的值：<input v-model="outName" ><br>

<product v-bind:name="outName"></product>

</div>

<script>

Vue.component('product', {

props:['name'],

template: '<div class="product" >{{name}}</div>'

})

new Vue({

el: '#div1',

data:{

outName:'产品名称'

}

})

### 自定义事件

增加自定义事件和在一个Vue对象上增加 methods 是一样的做法

先来个methods

然后在组件里v-on:click="fuctionName"

注，这里是在组件上增加的，而不是在视图上增加的。

例：

Vue.component('product', {

props:['name','sale'],

template: '<div class="product" v-on:click="increaseSale">{{name}} 销量: {{sale}} </div>',

methods:{

increaseSale:function(){

this.sale++

}

}

})

new Vue({

el: '#div1'

})

### 单文件组件格式

<template><div></div></template>

<script>

export default{

//在vue组件实例中，data要写成函数形式

data(){

return{

key1:val1,

key2:val2

}}

}</script>

## Vue.js 自定义指令

### 自定义指令的方式：

1. 使用Vue.directive 来自定义

2. 第一个参数就是 指令名称 xart

3. el 表示当前的html dom对象

4. 在方法体内就可以通过 innerHTML style.color 等方式操控当前元素了

例：//将元素内字体变成粉红色的自定义方法

Vue.directive('指令名称', function (el) {

el.innerHTML = el.innerHTML + ' ( x-art ) '

el.style.color = 'pink'

})

### 带参数的自定义指令

Vue.directive('xart', function (el,arg1) {

//arg1.value即为v-xart=”XXX”的xxx,这里为blue。

})

<div v-xart="blue">

### 钩子函数

钩子函数即回调函数，或者事件响应函数。 指的是，一个指令在创建过程中，经历不同生命周期的时候，vue.js 框架调用的函数。即在特定事件在Vue里自动调用的函数。

事件常见的有如下几种：

bind：只调用一次，指令第一次绑定到元素时调用。在这里可以进行一次性的初始化设置。

update：所在组件的 VNode 更新时调用，但是可能发生在其子 VNode 更新之前。指令的值可能发生了改变，也可能没有。但是你可以通过比较更新前后的值来忽略不必要的模板更新 (详细的钩子函数参数见下)。

unbind：只调用一次，指令与元素解绑时调用。

卸载等一系列过程；也就是beforeCreate，created，beforeMount，mounted，beforeUpdate，updated，beforeDestroy，destroyed

## Vue.js 路由

# 2021年6月22日

### data

当一个 Vue 实例被创建时， data 对象中的数据将视为vue实例的 property。

例如：

var vm = new Vue({

data:{

newTodoText: '',

visitCount: 0,

hideCompletedTodos: false,

todos: [],

error: null

}

})

直接用vm.newTodoText获取该值

当实例创建好之后再加入新的property时将不会触发任何视图的更新。

使用Object.freeze()则会阻止修改现有的 property

vue实例的一些属性和方法都有前缀$，以便与用户定义的 property 区分开来

如$watch能监视a属性的前后值的变化

vm.$watch('a', function (newValue, oldValue) {

// 这个回调将在 `vm.a` 改变后调用

})

## 指令（接在元素内 v-形式）

v-once 绑定数据后不再更新

v-html 将内容解释为html代码

例：

<p>Using v-html directive: <span v-html="rawHtml"></span></p>

// rawHtml ：’’ 在vue实例的data中定义

v-bind 绑定元素属性

例：

<div v-bind:id="dynamicId"></div>

绑定数据时还可支持js**表达式**(不是js语句)

## Vue.js 事件处理

### v-on 监听事件

例：在按钮上增加 click 监听，调用 count 方法

<button v-on:click="count">点击</button>//每单击一次按钮调用一次方法。

**v-on可缩写为@，即v-on:click可写为@click**

需要在内联语句处理器中访问原始的DOM事件时 可以用特殊变量 $event 把它传入方法

**<button v-on:click="warn('message', $event)"></button>**

**methods: {**

**warn: function (message, event) {}**

### 事件修饰符

.stop

.prevent

.capture

.self

.once

.passive

.native

### 阻止传播. stop

click后面加一个 .stop， 那么冒泡到了这里就结束了，就不会冒到father上面去了

例<div id="me" v-on:click.stop="doc">

### 优先触发 .capture

在元素上增加一个.capture

<div id="father" v-on:click.capture="doc">

当它的子元素发生冒泡时优先让该元素捕捉事件，再从发生冒泡的地方依次冒泡。

### 只有自己能触发，子元素无法触发.self

<div id="father" v-on:click.self="doc">

冒泡时忽略该元素，除非从该元素开始冒泡。

### 只能提交一次 .once

<div id="father" v-on:click.once="doc">

该元素只会冒泡一次，之后会被忽略

### 阻止提交 .prevent

通过在 click 后面添加 .prevent 可以阻止页面刷新。

@click.prevent="jump"

也可以直接用@click.prevent后面不跟函数

@click.prevent

### 立即触发 .passive

例：<div v-on:scroll.passive="onScroll">...</div>

//滚动时立即触发而不是等onScroll完成

与.prevent一起使用时后者被忽略

### 按键修饰符

vue中可在事件监听时添加按键修饰符

//只有在 `key` 是 `Enter` 时调用 `vm.submit()`

**<input v-on:keyup.enter="submit">**

### 系统修饰键

可以用如下修饰符来实现仅在按下相应按键时才触发鼠标或键盘事件的监听器。

.ctrl

.alt

.shift

.meta

例：<!-- Ctrl + Click -->

**<div v-on:click.ctrl="doSomething">Do something</div>**

注意：在 Mac 系统键盘上，meta 对应 command 键 (⌘)。在 Windows 系统键盘 meta 对应 Windows 徽标键 (⊞)。

.exact 修饰符允许你控制由精确的系统修饰符组合触发的事件

例：//有且只有 Ctrl 被按下的时候才触发，若不加则同时按CTRL和shift时也能触发

**<button v-on:click.ctrl.exact="onCtrlClick">A</button>**

### 鼠标按钮修饰符

这些修饰符会限制处理函数仅响应特定的鼠标按钮。

.left

.right

.middle

## Vue.js 双向绑定

### v-model 双向绑定(即表单输入绑定)

把 Vue对象上的数据显示在视图上，还能把视图上的数据放到Vue对象上。

<input v-model="name" >

new Vue({

el: '#div1',

data:{

name:"teemo"

},

methods:{

doClick:function(){

alert(this.name);

}

}

})

**实际上V-model就是v-on和v-bind的语法糖**

<input type="text" v-model="message">

等同于

**<input type="text" :value="message" @input="message = $event.target.value">**

### v-model修饰符

#### .lazy

对于输入元素，默认的行为方式是一旦有数据变化，马上进行绑定。

但是加上.lazy之后，相当于监听change操作，只有在**失去焦点**的时候，才会进行数据绑定

<input v-model.lazy="input1" placeholder="输入数据">

#### .number

v-model默认是string类型，可以通过.number方式确保获取到的是数字类型

<input v-model.number="input2" type="number" placeholder="输入数据">

#### .trim

trim 去掉前后的空白

<input v-model.trim="input" placeholder="输入数据">

## 计算属性与侦听属性

### computed

computed: {

// 计算属性的 getter

functionname: function () {

}

}

1. 计算属性是基于它们的响应式依赖进行缓存的。只在相关响应式依赖发生改变时它们才会重新求值，适用于多对一或一对一。
2. 支持缓存
3. 不支持异步，当computed内有异步操作时无效，无法监听数据的变化
4. 在computed中的，属性都有一个get和一个set方法，当数据变化时，调用set方法。

### watch

watch:{

//每个语句侦听一个数据，数据改变时运行,可接收newval和oldval，默认接收newval，

dataname: function(newval,oldval){

}

}

1. 不支持缓存，数据变，直接会触发相应的操作；
2. watch支持异步；
3. 一对一或一对多适应

**当需要在数据变化时执行异步或开销较大的操作时，watch更合适**

## class与style绑定

#### 对象形式

<div v-bind:class="{ active: isActive }"></div>

可将class与style绑定实现动态切换

上面的语法表示 active 这个 class 存在与否将取决于数据property中的 isActive（bool值）

#### 引用形式

也可用<div v-bind:class="classObject"></div>的形式。然后在vue实例中定义classObject

如data: {

classObject: {

active: true,

'text-danger': false

}

}

#### 数组形式

<div v-bind:class="[activeClass, errorClass]"></div>

## 条件渲染

v-if 判断是否渲染该元素

Vue 会尽可能高效地渲染元素，通常会复用已有元素而不是从头开始渲染。

用 key 表达可复用的元素是完全独立的，不要复用它们。只需添加一个具有唯一值的 key attribute

例如：<input placeholder="Enter your username" key="username-input">

v-show 判断是否设置display为none（显示则为block）；带有 v-show 的元素始终会被渲染并保留在 DOM 中

## 列表渲染

在 v-for 块中，我们可以访问所有父作用域的 property。v-for 还支持一个可选的第二个参数，即当前项的索引。

例：<li v-for="(item, index) in items">

{{ parentMessage }} - {{ index }} - {{ item.message }}

</li>

var example2 = new Vue({

el: '#example-2',

data: {

parentMessage: 'Parent',

items: [

{ message: 'Foo' },

{ message: 'Bar' }

]

}

})

可以用 of 替代 in 作为分隔符

也可以用 v-for 来遍历一个对象的 property，遍历对象时可以提供第二个的参数为 property 名称 (也就是键名)，还可以用第三个参数作为索引。

### 维护状态

当 Vue 正在更新使用 v-for 渲染的元素列表时，它默认使用“就地更新”的策略。如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序，而是就地更新每个元素，并且确保它们在每个索引位置正确渲染。这个类似 Vue 1.x 的 track-by="$index"。

这个默认的模式是高效的，但是只适用于不依赖子组件状态或临时 DOM 状态 (例如：表单输入值) 的列表渲染输出。

为了给 Vue 一个提示，以便它能跟踪每个节点的身份，从而重用和重新排序现有元素，你需要为每项提供一个唯一 key attribute。

### 数组更新检测

变更方法

Vue 将被侦听的数组的变更方法进行了包裹，所以它们也将会触发视图更新。这些被包裹过的方法包括

push()

pop()

shift()

unshift()

splice()

sort()

reverse()

替换数组

变更方法，顾名思义，会变更调用了这些方法的原始数组。相比之下，也有非变更方法，例如 filter()、concat() 和 slice()。它们不会变更原始数组，而总是返回一个新数组。当使用非变更方法时，可以用新数组替换旧数组。

注意事项

由于 JavaScript 的限制，Vue 不能检测数组和对象的变化，所以是对方法进行包裹

### 显示过滤/排序后的结果

有时，我们想要显示一个数组经过过滤或排序后的版本，而不实际变更或重置原始数据。在这种情况下，可以创建一个计算属性，来返回过滤或排序后的数组。

在计算属性不适用的情况下 (例如，在嵌套 v-for 循环中) 你可以使用一个方法：

<ul v-for="set in sets">

<li v-for="n in even(set)">{{ n }}</li>

</ul>

//even即是methods中的过滤函数名

v-for 也可以接受整数。在这种情况下，它会把模板重复对应次数。

<div>

<span v-for="n in 10">{{ n }} </span>

</div>

//1 2 3 4 5 6 7 8 9 10

### v-for与v-if同时使用

当它们处于同一节点，v-for 的优先级比 v-if 更高，这意味着 v-if 将分别重复运行于每个 v-for 循环中。只渲染v-for中满足v-if的节点

例：

<li v-for="todo in todos" v-if="!todo.isComplete">

{{ todo }}

</li>

### v-if与v-show的用法与区别

vue-show本质就是标签display设置为none，控制隐藏

vue-if是动态的向DOM树内添加或者删除DOM元素

### 在组件上使用 v-for

**2.2.0+ 的版本里，当在组件上使用 v-for 时，key 现在是必须的。**

然而，任何数据都不会被自动传递到组件里，因为组件有自己独立的作用域。为了把迭代数据传递到组件里，我们要使用 v-bind绑定组件的prop

例：

<my-component

v-for="(item, index) in items"

v-bind:item="item"

v-bind:index="index"

v-bind:key="item.id"

></my-component>

## 组件

在组件中使用data必须以函数形式，因为每一个组件都是独立的实例。若按照之前的data形式，所有组件共享一个data。

例：

data: function () {

return {

count: 0

}

}

template必须有且只有一个根元素

props用于设置组件的自定义属性，类似于之前的data

父级组件可以像处理 native DOM 事件一样通过 v-on 监听子组件实例的任意事件：

<blog-post

...

v-on:**enlarge-text**="postFontSize += 0.1"

></blog-post>

**子组件可以使用 $emit 触发父组件的自定义事件**

<button v-on:click="$emit('**enlarge-text**')">

Enlarge text

</button>

$emit 的第二个参数可**给自定义事件传递值**

当在父级组件监听这个事件的时候，可以通过 $event 访问到被传递的这个值

## $refs&&ref

ref ，字符串形式的属性，用来注册元素或组件到父组件的$refs上

$refs 包含所有注册了ref的子元素或组件实例，并且在渲染完成后才创建

$refs相对document.getElementById的方法，会减少获取dom节点的消耗。

## Vue.js 过滤器

### 全局过滤器

直接用vue，不用实例化。

Vue.filter('capitalize', function (value) {

if (!value) return ''

value = value.toString()

return value.charAt(0).toUpperCase() + value.slice(1)

})

### 组件过滤器(局部)

定义在vue实例内部

new Vue({

el: '#div1',

data: {

data:''

},

filters:{

filterName1:function(value) {

//对value的操作

}

filterName2:function(value1,value2,value3) {

//对value1,value2,value3同时操作

return value1+value2+value3；

}

}

### 调用过滤器

**用法一：在双花括号中插值**

{{ data1,data2,data3 | filterName }}

{{data| filterName1|filterName2 }}

//此处视为filterName1返回的值被filterName2调用，最终结果由filterName2返回。

**用法二：在v-bind表达式中使用**

<div v-bind:data=”value|filterName”></div>

# 深入了解组件

## 组件注册

### 组件名

1.w3c规范：(字母全小写且必须包含一个连字符)

例：Vue.component('my-component-name', { /\* ... \*/ })

2.驼峰法：(首字母大写命名)

例：Vue.component('MyComponentName', { /\* ... \*/ })

注：引用时<my-component-name> 和 <MyComponentName> 都是可接受的

### 局部组件

var ComponentA = { /\* ... \*/ }

var ComponentB = { /\* ... \*/ }

var ComponentC = { /\* ... \*/ }

new Vue({

el: '#app',

components: {

'component-a': ComponentA,

'component-b': ComponentB

}

})

对于 components 对象中的每个 property 来说，其 property 名就是自定义元素的名字，其 property 值就是这个组件的选项对象。

注意局部注册的组件在其子组件中不可用。

例如，如果你希望 ComponentA 在 ComponentB 中可用，则你需要这样写：

var ComponentA = { /\* ... \*/ }

var ComponentB = {

components: {

'component-a': ComponentA

},

// ...

}

## PROP

### 命名

HTML 中的 attribute 名是大小写不敏感的，浏览器会把所有大写字符解释为小写字符

当使用 DOM 中的模板时，camelCase (驼峰命名法) 的 prop 名需要使用其等价的 kebab-case (短横线分隔命名) 命名

Vue.component('blog-post', {

// 在 JavaScript 中是 camelCase 的

props: ['postTitle'],

template: '<h3>{{ postTitle }}</h3>'

})

<!-- 在 HTML 中是 kebab-case 的 -->

<blog-post post-title="hello!"></blog-post>

### prop类型

一般情况下props数组不能表现出prop类型，可以用对象形式列出prop。这些 property 的名称和值分别是 prop 各自的名称和类型，并不是固定了prop类型而是相当于提供了prop类型的说明与提示(当类型不对时控制台会发出警告)

例：props: {

title: String,

likes: Number,

isPublished: Boolean,

callback: Function,

contactsPromise: Promise // or any other constructor

}

无论传入的值是否动态，都需要v-bind告诉vue该值不是字符串而是js表达式

例：<blog-post v-bind:is-published="false"></blog-post>

prop 没有值的情况下，意味着传入为 `true`的布尔值

一次传入所有prop时可以直接使用不带参数的 v-bind(取代 v-bind:prop-name)

例：<blog-post v-bind="post"></blog-post>

### 单向数据流

所有的 prop 都使得其父子 prop 之间形成了一个单向下行绑定：父级 prop 的更新会向下流动到子组件中，但是反过来则不行。

### prop验证

props: {

// 必填字符串

propC: {

type: String,

**required: true**

},

// 带有默认值的数字

propD: {

type: Number,

default: 100

}

当 prop 验证失败的时候，(开发环境构建版本的) Vue 将会产生一个控制台的警告

### 非 Prop 的 Attribute

一个非 prop的attribute 是指传向一个组件，但是该组件并没有相应 prop 定义的 attribute。这些 attribute 会被添加到这个组件的根元素上。

禁用 Attribute 继承

不希望组件的根元素继承 attribute可在组件的选项中设置 inheritAttrs: false。

## 自定义事件

### 事件名

不同于组件和 prop，事件名不存在任何自动化的大小写转换。

### 自定义组件的v-model

组件上的 v-model 默认会利用名为 value 的 prop 和名为 input 的事件，可通过设置model选项改变默认。

Vue.component('base-checkbox', {

model: {

prop: 'checked',

event: 'change'

},

### 将原生事件绑定到组件

在一个组件的非原生元素上直接监听一个原生事件，使用 v-on 的 .native 修饰符：

<base-input v-on:focus.native="onFocus"></base-input>

**这是因为base-input不是原生DOM元素，不能直接绑定原生事件**

### .sync 修饰符

带有.sync 修饰符的 v-bind 不能和表达式一起使用

## 插槽(slot)

### 定义

slot就是可以让你在组件内添加内容的‘空间’,写在模板template中,**渲染时组件内的内容会被渲染进template中<slot>的位置。**

如果 <navigation-link> 的 template 中没有包含一个 <slot> 元素，则**该组件起始标签和结束标签之间的任何内容都会被抛弃。**

例：<navigation-link url="/profile">

Your Profile

</navigation-link>

//若component的template中无<slot></slot>，<navigation-link>标签内的所有内容（Your Profile）被抛弃，即等效于<navigation-link xxx></navigation-link>

### 编译作用域

**父级模板里的所有内容都是在父级作用域中编译的；子模板里的所有内容都是在子作用域中编译的。**

因此，插槽作用域和模板一样，不能访问组件的作用域

### 后备内容

即插槽的默认值，在模板中的slot标签内填写，它只会在组件标签之间没有提供内容的时候被渲染

### 具名插槽

将插槽指定名称，无则默认为“default”

在向具名插槽提供内容的时候，我们可以在一个 <template> 元素上使用 v-slot 指令，并以 v-slot 的参数的形式提供其名称**(v-slot可直接缩写为#)**

### 作用域插槽

只有组件能访问user时，为了让 user 在父级的插槽内容中可用，我们可以将 user 作为 <slot> 元素的一个 attribute 绑定上去：

例：<slot v-bind:user="user">

{{ user.lastName }}

</slot>

## 动态组件&异步组件

### 动态组件

<component v-bind:is="currentTabComponent"></component>

**通过改变data中currentTabComponent的值来达到切换组件的动态组件效果**

### 在动态组件上使用 keep-alive

切换组件时，旧组件会被覆盖销毁掉，使用<keep-alive> 元素将其动态组件包裹起来可缓存该组件，防止切换回来时重复渲染

例：<keep-alive>

<component v-bind:is="currentTabComponent"></component>

</keep-alive>

这样在该组件和其他组件切换时不会销毁该组件

### 异步组件

在工厂函数中设置异步解析组件。

在定义该组件的时候什么都不做

Vue 只有在这个组件需要被渲染的时候才会触发该工厂函数，且会把结果缓存起来供未来重渲染。

通常结合webpack的代码分割功能进行

例：Vue.component('async-example', function (resolve, reject) {

setTimeout(function () {

// 向 `resolve` 回调传递组件定义

resolve({

template: '<div>I am async!</div>'

})

}, 1000)

})

## 处理边界情况

### 访问元素&组件

访问根实例 $root

访问父实例 $parent

访问子实例 先为子实例或组件赋予一个ref的attribute引用，如<base-input ref="usernameInput"></base-input>，再在父类中使用$refs.refName

依赖注入 由provide和inject组成

provide 选项允许我们指定我们想要提供给后代组件的数据/方法。

inject 选项来接收指定的我们想要添加在这个后代实例上的 property

### 循环引用

**递归组件** 组件可以在模板中根据name来引用自己，

全局注册时，组件名称即默认为该组件的name

name: 'stack-overflow',

template: '<div><stack-overflow></stack-overflow></div>'

该引用会导致无限循环，因此递归还需加递归条件

## 进入/离开 & 列表过渡

### 单元素/组件的过渡

Vue 提供了 transition 的封装组件，在下列情形中，可以给任何元素和组件添加进入/离开过渡，transition 根据name值起效，默认名为v

例：<transition name="fade">

<p v-if="show">hello</p>

</transition>

在css中根据transition的6个状态设置效果

.fade-enter-active, .fade-leave-active {

transition: opacity .5s;

}

//也可设置css中效果为动画

.fade-enter-active {

animation: bounce-in .5s;

}

@keyframes bounce-in {

。。。

}



# 可复用行&组合

## 混入

### 基础

混入 (mixin) 提供了一种非常灵活的方式，来分发 Vue 组件中的可复用功能。

相当于把vue组件中的可复用部分单独拿出来，打包成一个混入，再在vue组件中引用该混入。

例：// 定义一个混入对象

var myMixin = {

methods: {

}

}

**// 定义一个使用混入对象的组件**

**var Component = Vue.extend({**

**mixins: [myMixin]**

})

//创建组件实例

var component = new Component() // => "hello from mixin!"

选项合并

当混入的内容和组件内容重叠时，会发生合并，先调用混入的内容再调用组件内容，若内容冲突则以组件为准。

例：var mixin = {

data: function () {

return {

message: 'hello',

foo: 'abc'

}

}

}

new Vue({

mixins: [mixin],

data: function () {

return {

message: 'goodbye',

bar: 'def'

}

},

created: function () {

console.log(this.$data)

**// => { message: "goodbye", foo: "abc", bar: "def" }**

}

})

### 全局混入

混入也可以进行全局注册。使用时格外小心！一旦使用全局混入，它将自动影响每一个之后创建的 Vue 实例。使用恰当时，这可以用来为自定义选项注入处理逻辑。

Vue.mixin({

、、、

})

//所有之后的vue实例都会混入该内容

## 自定义指令directive

### 基础

**全局指令**

// 注册一个全局自定义指令 `v-focus`

Vue.directive('focus', {

// 当被绑定的元素插入到 DOM 中时……

inserted: function (el) {

// 聚焦元素

el.focus()

}

})

**局部指令** 在组件中添加directives 的选项

directives: {

focus: {

// 指令的定义

inserted: function (el) {

el.focus()

}

}

}

然后就可以在模板中任何元素上使用新的 v-focus指令

### 钩子函数

**bind**：只调用一次，指令第一次绑定到元素时调用。在这里可以进行一次性的初始化设置。

**inserted**：被绑定元素插入父节点时调用 (仅保证父节点存在，但不一定已被插入文档中)。

**update：**所在组件的 **VNode 更新时调用，但是可能发生在其子 VNode 更新之前**。指令的值可能发生了改变，也可能没有。但是你可以通过比较更新前后的值来忽略不必要的模板更新 (详细的钩子函数参数见下)。

**componentUpdated：**指令所在组件的 VNode 及其子 VNode 全部更新后调用。

**unbind：**只调用一次，指令与元素解绑时调用。

### 钩子函数参数

**el：指令所绑定的元素，可以用来直接操作 DOM。**

**binding：一个对象，包含以下 property：**

name：指令名，不包括 v- 前缀。

value：指令的绑定值，例如：v-my-directive="1 + 1" 中，绑定值为 2。

oldValue：指令绑定的前一个值，仅在 update 和 componentUpdated 钩子中可用。无论值是否改变都可用。

expression：字符串形式的指令表达式。例如 v-my-directive="1 + 1" 中，表达式为 "1 + 1"。

arg：传给指令的参数，可选。例如 v-my-directive:foo 中，参数为 "foo"。

modifiers：一个包含修饰符的对象。例如：v-my-directive.foo.bar 中，修饰符对象为 { foo: true, bar: true }。

vnode：Vue 编译生成的虚拟节点。

oldVnode：上一个虚拟节点，仅在 update 和 componentUpdated 钩子中可用。

**动态指令参数：**指令的参数可以是动态的。例如，在 v-mydirective:[argument]="value" 中，argument 参数可以根据组件实例数据进行更新！

# 组间通信

## prop&$emit

适用于父子组件传值

## localStorage 只能字符串形式的键值对

## Vuex

mapState、 mutationState

## EventBus事件总线

用于无任何关系的组件通信

### 初始化

首先需要创建事件总线并将其导出，以便其它模块可以使用或者监听它。

1. 方法一：import Vue from 'vue'

export const EventBus = new Vue()

1. 方法二：Vue.prototype.$EventBus = new Vue()//该方法是一个全局的事件总线，可以像其他组件间通信一样使用。

### 使用

方法一：

// 发送消息

EventBus.$emit(channel: string, callback(payload1,…))

// 监听接收消息

EventBus.$on(channel: string, callback(payload1,…))

方法二：

this.$EventBus.$on(channel: string, callback(payload1,…))//监听接收消息

this.$EventBus.$emit (channel: string, callback(payload1,…))//发送消息

this.$ EventBus.$off(channel: string)//移除监听

例：在模板中按方法一使用eventbus

<!-- A.vue -->

<template>

<button @click="sendMsg()">-</button>

</template>

<script>

import { EventBus } from "../event-bus.js";

export default {

methods: {

sendMsg() {

EventBus.$emit("aMsg", '来自A页面的消息');

}

}

};

</script>

<script>

import {

EventBus

} from "../event-bus.js";

export default {

data(){

return {

msg: ''

}

},

mounted() {

EventBus.$on("aMsg", (msg) => {

// A发送来的消息

this.msg = msg;

});

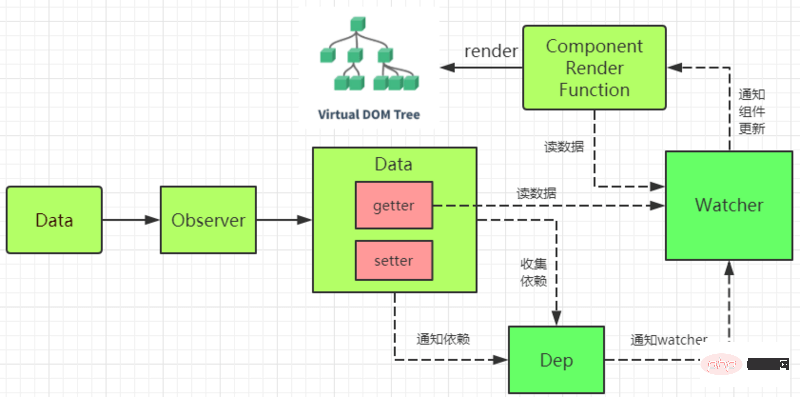
}

};

</script>

# 深入式响应原理

## 响应式原理



Observer : 它的作用是给对象的属性添加 getter 和 setter，用于依赖收集和派发更新

Dep : 用于收集当前响应式对象的依赖关系,每个响应式对象包括子对象都拥有一个 Dep 实例（里面 subs 是 Watcher 实例数组）,当数据有变更时,会通过 dep.notify()通知各个 watcher。

Watcher : 观察者对象 , 实例分为渲染 watcher (render watcher),计算属性 watcher (computed watcher),侦听器 watcher（user watcher）三种

### 双向绑定原理

vue数据双向绑定是通过数据劫持结合发布者-订阅者模式的方式来实现的。

我们已经知道实现数据的双向绑定，首先要对数据进行劫持监听，所以我们需要设置一个监听器Observer，用来监听所有属性。如果属性发生变化了，就需要告诉订阅者Watcher看是否需要更新。因为订阅者是有很多个，所以我们需要有一个消息订阅器Dep来专门收集这些订阅者，然后在监听器Observer和订阅者Watcher之间进行统一管理的。接着，我们还需要有一个指令解析器Compile，对每个节点元素进行扫描和解析，将相关指令（如v-model，v-on）对应初始化成一个订阅者Watcher，并替换模板数据或者绑定相应的函数，此时当订阅者Watcher接收到相应属性的变化，就会执行对应的更新函数，从而更新视图。

### Watcher 和 Dep 的关系

watcher 中实例化了 dep 并向 dep.subs 中添加了订阅者,dep 通过 notify 遍历了 dep.subs 通知每个 watcher 更新。

## vue更新dom

Vue在更新DOM时是异步执行的，只要侦听到数据变化，**Vue将开启一个队列，并缓冲在同一事件循环中发生的所有数据变更**，如果同一个watcher被多次触发，只会被推入到队列中一次。这种在缓冲时去除重复数据对于避免不必要的计算和DOM操作是非常重要的。

## nextTrick(callback())

定义：在下次 DOM 更新循环结束之后执行延迟回调。在修改数据之后立即使用这个方法，获取更新后的 DOM。**简单来说就是当数据更新时，在DOM中渲染完成后，执行回调函数。**

## diff算法

### 虚拟DOM

虚拟dom用对象属性来描述节点，将整个dom树抽象成一个对象，该对象至少包含tag,props,children三个属性。

原生DOM操作每有一个DOM节点改变都会遍历所有DOM节点来重排，改变多个节点会多次重复遍历，浪费性能；虚拟DOM将多个节点打包到一个JS对象中，一次遍历改变所有节点。

在第一次渲染时直接采用虚拟dom生成dom树，之后则采用对比新旧虚拟dom，只更改不同的部分到真实dom上

### diff算法

DOM 发生变化的时候，通过 diff 算法比对 JavaScript 原生对象，计算出需要变更的 DOM，然后只对变化的 DOM 进行操作，而不是更新整个视图。

### 时间复杂度O(n3)到O(n)

字符串的最小编辑距离为O(nm),可视为O(n2),树可视为一种递归的数据结构，遍历的时间复杂度也为O(n),因此树的最小距离编辑算法时间复杂度为O(n3)；

为了提高效率，vue和react做了假设

**检测VDOM的变化只发生在同一层**

**检测VDOM的变化依赖于用户指定的key**

**这样就不用去比较不同层级节点并且用key指明了变化。**

//在dom树的修改中，新树A的每个节点会遍历旧树B，为O(n)，当节点a在B中没有时//会插入一个b，b节点之后的向后移为m，这里取极值为n，树中共有n给节点，所以是O(n3)

## Vue中的Key

标识出每一个节点，能高效的更新虚拟dom。在虚拟DOM中通过key判断是否是同一节点来避免非必要的元素更新。

## Vue2.0和3.0的区别

### 完全重写vitrual dom，修改了diff算法

### 基于es6的proxy（也是双向绑定原理）

重构响应式系统，使用Proxy替换Object.defineProperty。

VUE2.0使用的是Object.defineProperty 的 getter 和 setter来响应元素

Proxy 对象用于创建一个对象的代理，从而实现基本操作的拦截和自定义（如属性查找、赋值、枚举、函数调用等）。

**可直接监听数组类型的数据变化**

监听的目标为对象本身，不需要像Object.defineProperty一样遍历每个属性，有一定的性能提升

可拦截apply、ownKeys、has等13种方法，而Object.defineProperty不行，设置defineReactive函数可使vue2.0也能监听数组类型的变化，但性能代价较大。

直接实现对象属性的新增/删除

### 默认进行懒观察（lazy observation）。

修改属性时，只有依赖该属性的watcher才会重新运行

### 独特的响应式值ref

import { ref } from 'vue'

const count = ref(0)

### 新增组件 API

VUE3.0增加了const app = Vue.createApp({})，相较于之前2.0的new vue{}形式

对typescript完全支持

## 生命周期钩子

vue实例有一个完整的生命周期，也就是从开始创建、初始化数据、编译模板、挂载Dom、渲染→更新→渲染、销毁等一系列过程，我们称这是Vue的生命周期。通俗说就是Vue实例从创建到销毁的过程，就是生命周期。

即

beforecreate，vue实例初始化之后，数据观测和event/watcher事件配置之前被调用

created，实例创建之后，挂载阶段之前，$el属性还不可见

beforemounte，挂载阶段之前。相关渲染函数被首次调用

**mounted，el被新创建的vue.$el替换，挂载成功**

**注意：mounted 不会保证所有的子组件也都一起被挂载。如果你希望等到整个视图都渲染完毕，可以在 mounted 内部使用 vm.$nextTick：**

beforeupdate，数据更新时调用

update，组件dom更新完毕

beforeunmounte，

unmounted，

beforeDestroy，

destroyed

每一个组件或者实例都会经历一个完整的生命周期，总共分为三个阶段：初始化、运行中、销毁。

# 项目总结

## 实用方法

### 直接调用模板中数据或方法时需加this

### 钩子函数都采用函数形式

### 动态绑定元素属性如style等

### v-for 必须设置每个元素的key了

### 获取通过事件调用该方法的当前dom元素

通过$event.srcElement获取当前元素

例：<div @click="onSelect($event)">

onSelect($event){ var curNode=$event.target;

}

## 理论细节

### <keep-alive>

<keep-alive>是Vue提供的一个抽象组件，用来对组件进行缓存，从而节省性能，由于是一个抽象组件，其所有节点都是虚拟节点，所以在页面渲染完毕后不会被渲染成一个DOM元素。第一次搭载时的生命周期也是创建，初始化，编译，搭载，渲染，更新

后面则通过active和deactive

### 父子组件生命周期

加载渲染：

1. 父组件 beforecreate=>父组件created=>父组件beforemount=>子组件beforecreate=>子组件created=>子组件beforemoubt=>子组件mounted=>父组件mounted

更新过程

父组件 beforeupdate=>子组件beforeupdate=>子组件updated=>父组件updated