### 什么是vue

把一个 json对象的数据，显示到一个元素上去。

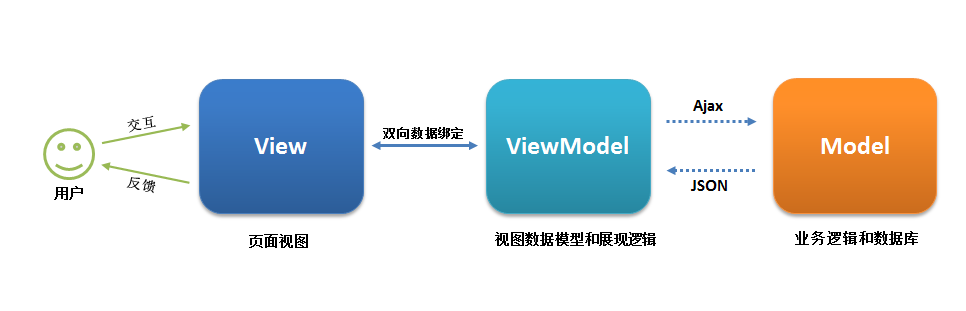
如果使用Vue, 那么仅仅需要提供数据，以及数据要绑定到的元素的id,就行了,不需要显式地操作HTML DOM。

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 监听事件

### 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**

### 事件修饰符

.stop

.prevent

.capture

.self

.once

### 阻止冒泡. 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

## 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 双向绑定

### v-model 双向绑定

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);

}

}

})

为什么alert的是name,弹出的却是value值？

因为是双向绑定，当input里面的值发生变化的时候，就会自动把变化后的值，绑定到Vue对象上去了

只用于表单元素

### 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，才会进行数据绑定了

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

#### .number

有时候，拿到了数据需要进行数学运算， 为了保证运算结果，必须先把类型转换为number类型，而v-model默认是string类型，所以就可以通过.number方式确保获取到的是数字类型了。

<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">

### 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。

## 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 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>

## 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'

})

## 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。

## Vue.js 路由

# 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.definePropert。

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

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)

### 生命周期

### 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