**1、响应式属性和方法**

每个 Vue 实例都会代理其 data 对象里所有的属性。

var data = { a: 1 }

var vm = new Vue({

data: data

})

vm.a === data.a // -> true

// 设置属性也会影响到原始数据

vm.a = 2

data.a // -> 2

// ... 反之亦然

data.a = 3

vm.a // -> 3

即：都可以使vm.xxx这种方式获取和修改实例属性。

注意：

注意只有这些被代理的属性是响应的。如果在实例创建之后添加新的属性到实例上，它不会触发视图更新。我们将在后面详细讨论响应系统。

那么这种情况的话，可以使用VueJs的vm.$xxx的方式来修改实例属性，这同样会触发视图更新，响应式的改变。

**2、生命周期钩子**

生命周期钩子包括created、beforeCompile、compiled、ready、beforeDestroy、destroyed

钩子的 this 指向调用它的 Vue 实例

我们可以在钩子的方法中使用this来指代当前实例。Vuejs中没有控制器的概念，组件的自定义逻辑可以分割在这些钩子里面。

**3、插值**

单词插值，只有在第一次渲染时候渲染值，之后不会随实例属性值的变化而改变，如：

<span>单词插值: {{ msg = <h3>hello</h3>}}</span>

插入原始HTML，这种方式内容以 HTML 字符串插入，数据绑定将被忽略，如：

<div>{{{ raw\_html }}}</div>

如果需要复用模板片断，应当使用 [partials](http://cn.vuejs.org/api/#partial)

2.0 Vue.partial 已经移除，取而代之的是更明确的组件之间的数据流–props。

注意：

在网站上动态渲染任意 HTML 是非常危险的，因为容易导致 XSS 攻击。记住，只对可信内容使用 HTML 插值，永不用于用户提交的内容。

**4、v-show 和 v-if的选择**

v-if 也是惰性的：如果在初始渲染时条件为假，则什么也不做——在条件第一次变为真时才开始局部编译（编译会被缓存起来）。

相比之下，v-show 简单得多——元素始终被编译并保留，只是简单地基于 CSS 切换。

**5、使用track-by，优化列表循环 (vue1.0)**

因为 v-for 默认通过数据对象的特征来决定对已有作用域和 DOM 元素的复用程度，这可能导致重新渲染整个列表。但是，如果每个对象都有一个唯一 ID 的属性，便可以使用 track-by 特性给 Vue.js 一个提示，Vue.js 因而能尽可能地复用已有实例。

例如，假定数据为：

{

items: [

{ \_uid: '88f869d', ... },

{ \_uid: '7496c10', ... }

]

}

然后可以这样给出提示：

<div v-for="item in items" track-by="\_uid">

<!-- content -->

</div>

然后在替换数组 items 时，如果 Vue.js 遇到一个包含 \_uid: ‘88f869d’ 的新对象，它知道它可以复用这个已有对象的作用域与 DOM 元素。

**6、prop数据绑定）**

prop 默认是单向绑定：当父组件的属性变化时，将传导给子组件，但是反过来不会。这是为了防止子组件无意修改了父组件的状态——这会让应用的数据流难以理解。

解决办法 : 使用 .sync 或.once 绑定修饰符显式地强制双向或单次绑定。

v-bind 的 .once和.sync 修饰符 移除

Props 现在只能单向传递。为了对父组件产生反向影响，子组件需要显式地传递一个事件而不是依赖于隐式地双向绑定。

<!-- 默认为单向绑定 -->

<child :msg="parentMsg"></child>

<!-- 双向绑定 -->

<child :msg.sync="parentMsg"></child>

<!-- 单次绑定 -->

<child :msg.once="parentMsg"></child>

**7、组件的父链**

子组件可以用 this.$parent 访问它的父组件。根实例的后代可以用 this.$root 访问它。父组件有一个数组 this.$children，包含它所有的子元素。

**8.Vuejs组件**

vuejs构建组件使用

Vue.component('componentName',{ /\*component\*/ })；

这里注意一点，组件要先注册再使用，也就是说：

Vue.component('mine',{

template:'#mineTpl',

props:['name','title','city','content']

});

var v=new Vue({

el:'#vueInstance',

data:{

name:'zhang',

title:'this is title',

city:'Beijing',

content:'these are some desc about Blog'

}

});

如果反过来会报错，因为反过来代表先使用了组件的，但是组件却没注册。

webpack报错后，使用webpack --display-error-details可以排错

**9.指令keep-alive**

在看demo的时候看到在vue-router写着keep-alive，keep-alive的含义：  
如果把切换出去的组件保留在内存中，可以保留它的状态或避免重新渲染。为此可以添加一个keep-alive指令

<component :is='curremtView' keep-alive></component>

**10.如何让css只在当前组件中起作用**

在每一个vue组件中都可以定义各自的css，[js](http://lib.csdn.net/base/javascript" \t "_blank" \o "JavaScript知识库)，如果希望组件内写的css只对当前组件起作用，只需要在style中写入scoped，即：

<style scoped></style>

**11.vuejs循环插入图片**

在写循环的时候，写入如下代码：

<div class="bio-slide" v-for="item in items">

<img src="{{item.image}}">

</div>

此时在控制台会出现警告  
[Vue Warn]: src="{{item.image}}": interpolation in "src" attribute will cause a 404 request. Use v-bind:src instead.这里意思是在“src”属性插值将导致404请求。使用v-bind：src代替。  
所以替换成如下：

<div class="bio-slide" v-for="item in items">

<img v-bind:src="item.image">

</div>

这里需要主要，v-bind在写的时候不能再用{{}}，根据官方的说法：

<a v-bind:href="url"></a>

这里 href 是参数，它告诉 v-bind 指令将元素的 href 特性跟表达式 url 的值绑定。可能你已注意到可以用特性插值href="{{url}}" 获得同样的结果：这样没错，并且实际上在内部特性插值会转为 v-bind 绑定。

**12.绑定value到Vue实例的一个动态属性上**

对于单选按钮，勾选框及选择框选项，v-model绑定的value通常是静态字符串（对于勾选框是逻辑值）：

<!-- `toggle` 为 true 或 false -->

<input type="checkbox" v-model="toggle">

但是有时候想绑定value到vue实例的一个动态属性上，这时可以用v-bind实现，并且这个属性的值可以不是字符串。例如绑定Checkbox的value到vue实例的一个动态属性：

<input

type="checkbox"

v-model="toggle"

v-bind:true-value="a"

v-bind:false-value="b">

<p>{{toggle}}</p>

这里绑定后，并不是说就可以点击后由true,false的切换变为a,b的切换，因为这里定义的动态a，b是scope上的a,b，并不能直接显示出来，此时

//当选中时

vm.toggle === vm.a

//当没选中时

vm.toggle === vm.b

所以此时需要在data中定义a,b，即：

new Vue({

el:'...',

data:{

a:'a',

b:'b'

}

});

**13.片段实例**

下面几种情况会让实例变成一个片断实例：

模板包含多个顶级元素。

模板只包含普通文本。

模板只包含其它组件（其它组件可能是一个片段实例）。

模板只包含一个元素指令，如<partial> 或 vue-router 的 <router-view>。

模板根节点有一个流程控制指令，如 v-if 或 v-for。

这些情况让实例有未知数量的顶级元素，它将把它的 DOM 内容当作片断。片断实例仍然会正确地渲染内容。不过，它没有一个根节点，它的$el 指向一个锚节点，即一个空的文本节点（在开发模式下是一个注释节点）。  
但是更重要的是，组件元素上的非流程控制指令，非 prop 特性和过渡将被忽略，因为没有根元素供绑定：

<!-- 不可以，因为没有根元素 -->

<example v-show="ok" transition="fade"></example>

<!-- props 可以 -->

<example :prop="someData"></example>

<!-- 流程控制可以，但是不能有过渡 -->

<example v-if="ok"></example>

片段实例也有用处，但是通常情况下组件有一个根节点比较好，它会保证组件元素上的指令和特性能正确的转换，同时性能也稍微好些。

**14.路由嵌套**

路由嵌套会将其他组件渲染到该组件内，而不是进行整个页面跳转router-view本身就是将组件渲染到该位置，想要进行页面跳转，就要将页面渲染到根组件，在起始配置路由时候写到：

var App = Vue.extend({ root });

router.start(App,'#app');

Vue.use(router)

这里首先将根组件注册进来，用于将路由中配置好的各个页面渲染出来，然后将根组件挂载到与#app匹配的元素上。

**15.实现多个根据不同条件显示不同文字的方法**

v-if,v-else可以实现条件选择，但是如果是多个连续的条件选择，则需要用到计算属性computed。例如实现当输入框中什么都没写的时候显示字符串‘empty’,否则显示输入框中的内容，代码如下：

<div id="test">

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

<h1>{{changeVaule}}</h1>

</div>

new Vue({

el:'#test',

data:{

changeVaule:'123'

},

computed :{

changeVaule:function(){

if(this.inputValue!==''){

return this.inputValue;

}else{

return 'empty';

}

}

}

});

**16.Vuejs在变化检测问题**

1.检测数组

由于[JavaScript](http://lib.csdn.net/base/javascript)的限制，vuejs不能检测到下面数组的变化：

直接索引设置元素，如vm.item[0]={};

修改数据的长度，如vm.item.length。

为了解决问题1，Vuejs扩展了观察数组，为它添加一个$set()方法：

// 与 `example1.items[0] = ...` 相同，但是能触发视图更新

example1.items.$set(0, { childMsg: 'Changed!'})

问题2，需要一个空数组替换items。

除了$set()，vuejs也为观察数组添加了$remove()方法，用于从目标数组中查找并删除元素，在内部调用了splice()。因此，不必：

var index = this.items.indexOf(item)

if (index !== -1) {

this.items.splice(index, 1)

}

只需：

this.items.$remove(item);

2.检测对象

受ES5的显示，Vuejs不能检测到对象属性的添加或删除。因为Vuejs在初始化时候将属性转化为getter/setter，所以属性必须在data对象才能让Vuejs转换它，才能让它是响应的，例如：

var data = { a: 1 }

var vm = new Vue({

data: data

})

// `vm.a` 和 `data.a` 现在是响应的

vm.b = 2

// `vm.b` 不是响应的

data.b = 2

// `data.b` 不是响应的

不过，有办法在实例创建之后添加属性并且让它是响应的。对于Vue实例，可以使用$set(key,value)实例方法：

vm.$set('b', 2)

// `vm.b` 和 `data.b` 现在是响应的

对于普通数据对象，可以使用全局方法Vue.set(object, key, value):

Vue.set(data, 'c', 3)

// `vm.c` 和 `data.c` 现在是响应的

有时你想向已有对象上添加一些属性，例如使用 Object.assign() 或 \_.extend() 添加属性。但是，添加到对象上的新属性不会触发更新。这时可以创建一个新的对象，包含原对象的属性和新的属性：

// 不使用 `Object.assign(this.someObject, { a: 1, b: 2 })`

this.someObject = Object.assign({}, this.someObject, { a: 1, b: 2 })

**17.关于vuejs页面闪烁{{message}}**

在vuejs指令中有v-cloak，这个指令保持在元素上直到关联实例结束编译。和CSS规则如[v-cloak]{display:none}一起用时，这个指令可以隐藏未编译的Mustache标签直到实例准备完毕。用法如下：

[v-cloak]{

display:none;

}

<div v-cloak>{{message}}</div>

这样<div>不会显示，直到编译结束

**18.关于在v-for循环时候v-model的使用**

有时候需要循环生成input，用v-model绑定后，利用vuejs操作它，此时我们可以在v-model中写一个数组selected[$index]，这样就可以给不同的input绑定不同的v-model，从而分别操作他们。

12.vuejs中过渡动画

在vuejs中，css定义动画：

.zoom-transition{

width:60%;

height:auto;

position: absolute;

left:50%;

top:50%;

transform: translate(-50%,-50%);

-webkit-transition: all .3s ease;

transition: all .3s ease;

}

.zoom-enter, .zoom-leave{

width:150px;

height:auto;

position: absolute;

left:20px;

top:20px;

transform: translate(0,0);

}

其中动画在定的时候要注意上下对应，上面有什么，下面有什么，都要变化的，如果有不变化的，应该抽离出去，作为公共css样式，在上面的css中，如果我只写 transform: translate(-50%,-50%);而不写下面的transform: translate(0,0);则会导致上面的transform: translate(-50%,-50%);被添加到下面，认为这个是不变的。

**19.指令v-el的使用 (vue1.0 , 2.0 用$ref 代替)**

有时候我们想就像使用[jQuery](http://lib.csdn.net/base/jquery" \t "_blank" \o "jQuery知识库)那样去访问一个元素，此时就可以使用v-el指令，去给这个元素注册一个索引，方便通过所属实例的$el访问这个元素。  
注意  
HTML不区分大小写，所以v-el:someEl将转换为全小写。可以用v-el:some-el然后设置this.$el.someEl。  
示例

<span v-el:msg>hello</span>

<span v-el:other-msg>world</span>

this.$els.msg.textContent // -> "hello"

this.$els.otherMsg.textContent // -> "world"

this.$els.msg //-><span>hello</span>

**20.关于vuejs中使用事件名**

在vuejs中，我们经常要绑定一些事件，有时候给DOM元素绑定，有时候给组件绑定。绑定事件在HTML中用v-on:click-"event",这时evet的名字不要出现大写，因为在1.x中不区分大小写，所以如果我们在HTML写v-on:click="myEvent"而在js中写myEvent就出错误，所以在vuejs的1.x绑定事件时候，要尽量避免使用大写字母。在2.0中没有该限制！

**21.v-if与v-show的区别**

v-if直接不渲染这个DOM元素，而v-show是会渲染DOM元素，只是使用display:none隐藏，打开开发者工具可以看到该DOM

16.关于transition全局钩子如何在组件中使用

Vue.transition是定义一个全局transition钩子的，如果想针对组件定义，则需要如下写法：

export default{

transition:{

'fade':{

enter() {},

leave() {}

}

}

}

这样fade这个过度钩子只会作用于组件内，如果同时有同名的全局钩子，则会优先使用组建定义的

**22.利用vue-router如何实现组件在渲染出来前执行某个事件**

export default{

data(){

return{

selected:0,

currentView:'view\_0'

}

},

methods:{

choose(index) {

this.selected=index;

this.currentView='view\_'+index;

}

},

route:{

data() {

/\*每次切换路由，在渲染出页面前都会执行\*/

}

}

}