vue介绍

- Vue.js是构建数据驱动的 web 界面的库,而不是一个全能框架—它只聚焦于 视图层 。
- 响应的数据绑定
  - 每当修改了数据，DOM 便相应地更新。这样我们应用中的逻辑就几乎都是直接修改数据了，不必与 DOM 更新搅在一起。这让我们的代码更容易 撰写、理解与维护 。
- 组件系统
  - 让我们可以用 独立可复用 的小组件来构建大型应用。
- 特性
  - 简洁 数据驱动 组件化 轻量快速 模块友好

vue的安装

- CDN地址

```
http://cdn.jsdelivr.net/vue/1.0.24/vue.min.js
```

- bower下载

```
bower install vue
```

- npm下载

```
npm install vue
```

vue的简单使用

- 引入vue.js
- 实现简单的Hello World

```
<div id="app">
    {{hello}}
</div>
```

```
new Vue({
  el:"#app",
  data:{
    hello:'hello world'
  }
});
```

实现双向数据绑定

- v-model

```
<div id="app">
    <input type="text" v-model="hello">
    {{hello}}
</div>
```

```
new Vue({el:"#app"});
```

绑定表达式

- 可以进行运算
  - {{}}
- 支持三元运算符
- 只绑定一次
  - {{*hello}}

```
new Vue({el:"#app",hello:{data:'hello'}});
```

- 实现绑定html
  - {{{hello}}}

```
new Vue({el:"#app",data:{hello:'<h1>hello world</h1>'}});
```

Vue的实例

- 一个 Vue 实例其实正是一个 MVVM 模式中所描述的 ViewModel
- 属性

```
var message = {hello:1};
var vm = new Vue({
  el:'#app',
  data:{
    message:message
  }
});
alert(vm.message === message);
```

> 当实例创建后给实例增加属性,不会导致视图的刷新

Vue通过$暴露实例上的属性和方法

- $el

```
vm.$el==document.getElementById('app')
```

- $data

```
vm.$data==message
```

- $watch

```
vm.$watch('message',function(newVal,oldVal){})
```

实例的生命周期

- Vue 实例在创建时有一系列初始化步骤 image
  - `created` 先实例化,在实例化后(检测el)
  - `vm.$mount('#app');` 手动挂载实例
  - `beforeCompile` 开始编译之前
  - `compiled` 编译完成后
  - `ready` 插入文档后
  - `vm.$destroy();` 手动销毁实例
  - `beforeDestroy` 将要销毁
  - `destroyed` 销毁实例

实例的生命周期

```
var vm = new Vue({
    data:{
      hello:123
    },
    created: function () {alert('实例创建完成');},
    beforeCompile: function () {alert('开始编译前')},
    compiled: function () {alert('编译完成')},
    ready: function () {alert('准备好了')},
    beforeDestroy: function () {alert('准备销毁')},
    destroyed: function () {alert("销毁")}
});
vm.$mount('#app');
vm.$destroy();
```

计算属性

- computed计算属性值

```
{{c}}
computed:{
  c: function () {
    return this.hello+345
  }
}
```

计算属性

- set和get方法

```
computed:{
    b:{
      set: function (v) {
        this.hello = v;
      },
      get: function () {
        return this.hello-100;
      }
    }
}
vm.b = 100;
```

> 当前this表示data的属性值

解决闪烁问题

- v-text

```
<div v-text="hello"></div>
```

- v-cloak

```
//引入css
[v-cloak] {display: none;}
```

```
<div v-cloak>{{hello}}</div>
```

v-if/v-show

- v-if 在不符合条件时,移除dom

```
<div v-if="false">{{hello}}</div>
      <div v-else>{{world}}</div>
```

- v-if `<template>`

```
<template  v-if="true">
  <div>{{hello}}</div>
  <div>{{hello}}</div>
  <div>{{hello}}</div>
</template>
<div v-else>{{he}}</div>
```

- v-show 通过display CSS属性切换

```
<div v-show="false">{{hello}}</div>
<div v-else>{{world}}</div>
```

> v-show 不支持 `<template>` 语法

v-else

- v-else 元素必须立即跟在 v-if 或 v-show 元素的后面——否则它不能被识别;
- > 一般来说，v-if 有更高的切换消耗而 v-show 有更高的初始渲染消耗。因此，如果需要频繁切换 v-show 较好，如果在运行时条件不大可能改变 v-if 较好

v-for数据遍历

- 基于源数据将元素或模版块重复

```
<li v-for="data in datas ">
    {{data.name}}
</li>
```

- 遍历对象

```
<li v-for="l in list ">
  {{$key}}
  <!--$key当前键-->
</li>
```

- 嵌套循环

```
<li v-for="(index,value) in datas">
  <span v-for="va in value.name">
    {{index}}
    {{$index}}
    <!--$index当前索引-->
  </span>
</li>
```

v-for的track-by

- 如果没有唯一的键供追踪，可以使用:

```
track-by="$index"
```

v-bind

- 绑定图片属性

```
<img v-bind:src="imageSrc">
```

- 绑定图片属性

```
<a v-bind:href="aHref">
```

- 简写

```
<a :href="aHref">
```

> 不要使用{{aHref}}进行设置链接

v-on

- 绑定事件

```
<div v-on:click = 'dosome'>123</div>
<!--简写-->
<div @click = 'dosome'>123</div>
```

```
methods:{
  dosome: function (e) {
    console.log(e); //e是事件源
  }
}
```

- 绑定事件传递参数

```
<!--当传递参数时,手动调用$event参数-->
<div @click = 'dosome("1",$event)'>123</div>
```

```
methods:{
  dosome: function (a,e) {
    console.log(a,e); //e是事件源
  }
}
```

v-on中的修饰符

- 阻止事件冒泡.stop

```
<div @click="dosome">
  123
  <div>456</div>
</div>
```

- 阻止默认事件 .prevent

```
<a href="http://www.baidu.com" @click.prevent>123</a>
```

- 自己触发事件 .self

```
<div v-on:click.self="dosome">
  123
  <div>456</div>
</div>
```

v-on中的按键修饰符

- 按键修饰符号

```
<input v-on:keyup.13="dosome">
  或
<input type="text" @keyup.enter="dosome">
```

- enter tab delete esc space up down left right
- 自定义按键 Vue.directive('on').keyCodes.A = 65

绑定htmlClass

- 通过{{}}方式绑定class

```
<div class="{{hello}}">直接取data上对象的属性</div>
```

- 通过v-bind的方式(绑定单一class)

```
<div v-bind:class="hello">绑定属性</div>
```

- 绑定多个class名字

```
<div v-bind:class="{hello:true,world:false}">123</div>
```

- 直接绑定对象

```
<div v-bind:class="message">123</div>
   data:{
     message:{
       hello:true,
       world:true
     }
   }
```

绑定htmlClass

- 直接绑定数组

```
<div v-bind:class="[hello,world]">123</div>
  data:{
      hello:'hello',
      world:'world'
  }
```

- 三元运算符号

```
<div v-bind:class="[hello,isTrue?'hello1':'hellow2']">123</div>
```

- 对象和数组混用

```
<div v-bind:class="[hello,{world:isTrue}]">123</div>
```

> {{className}}和v-bind:class不要混用;
> class 和v-bind:class可以同时存在

绑定行内样式

- 直接绑定到

```
<div v-bind:style="{color:'red',background:'yellow'}">行内</div>
```

- 绑定对象

```
<div v-bind:style="className">行内</div>
 data:{
    className:{    color:'red'    }
 }
```

- 数组方式绑定多组对象

```
<div v-bind:style="[hello,world]">2个样式</div>
data:{
    hello:{    color:'red'     },
    world:{    'fontSize':'50px'    }
}
```

- 自动添加前缀

filter内置过滤器

- capitalize

```
{{ msg | capitalize }}
```

- uppercase/lowercase

```
{{ msg | uppercase }}
{{ msg | lowercase }}
```

filter内置过滤器

- currency

```
{{ msg | currency }}
```

```
{{ amount | currency '£' 0 }}
```

filter内置过滤器

- pluralize

```
{{5}}{{5 | pluralize 'item'}}
{{23}}{{23 | pluralize 'st' 'nd' 'rd' 'th'}}
```

filter内置过滤器

- json

```
{{obj | json 4}}
```

filter内置过滤器

- debounce

```
<input @keyup="onKeyup | debounce 500">
```

filter内置过滤器

- limitBy

```
<!--显示五条 从第几个开始-->
<div v-for="item in items | limitBy 5 10"></div>
```

filter内置过滤器

- 在所有数据中过滤

```
<div v-for="item in items | filterBy 'hello'">
```

- 限定搜索范围(提高性能)

```
<div v-for="user in users | filterBy 'Jack' in 'name' 'phone'">
```

filter内置过滤器

- 在所有数据中根据指定字段排序

```
<button @click="order = order * -1">排序反</button>
<div v-for="item in items | orderBy 'name' -1">
```

自定义过滤器

- 全局方法注册

```
Vue.filter()
```

- 定义过滤方法

```
Vue.filter('reverse', function (value,begin,end) {
  return value+begin+end;
});
```

自定义过滤器

- 将数据写回model

```
Vue.filter('myFilter', {
  read: function (val) {
    return val.slice(3);
  },
  write: function (val,oldVal) {
    return val.slice(3);
  }
})
```

> 如果不是字符串则在当前作用域上查找相关字段

表单控件元素checkbox

- 获取checkbox值

```
<input type="checkbox" id="check" v-model="check">
<label for="check">{{check}} </label>
```

- 获取一组checkbox的值

```
<input type="checkbox" id="play" value="play" v-model="checks">
<input type="checkbox" id="join" value="join" v-model="checks">
<input type="checkbox" id="think" value="think"  v-model="checks">
{{checks}}
```

表单控件元素radio

- 获取radio的值

```
<input type="radio" value="first" v-model="radio">
<input type="radio" value="second" v-model="radio">
{{radio}}
```

表单控件元素select

- 下拉菜单的获取值

```
<select v-model="selected">
  <option value="4" selected>A</option>
  <option value="5">b</option>
  <option value="6">c</option>
</select>
{{selected}}
```

- 下拉菜单的多选

```
<select v-model="selects" multiple>
  <option value="4" selected>A</option>
  <option value="5">b</option>
  <option value="6">c</option>
</select>
{{selects}}
```

- 动态获取下拉菜单数据

```
<select v-model="name">
  <!--绑定value 到 Vue 实例的一个动态属性-->
  <option v-for="a in ary" :value="a.value">{{a.name}}</option>
```

表单元素参数特性

- lazy 将input改变为change

```
<input type="text" v-model="data" lazy>
{{data}}
```

- 延时数据改变时间

```
<input type="text" v-model="data" debounce="500">
```

> 在使用 debounce 时应当用 vm.$watch();

组件

- 使用方法Vue.component(tag, constructor)

```
var com = Vue.extend({
  template:'<div>hello world</div>'
})
Vue.component('my',com);
var vm = new Vue({
  el:"#app"
});
```

> 要先注册组件再创造实例

components

- 多个组件

```
var Child = Vue.extend({
  template:'****'
});
var Parent = Vue.extend({
  template: '...<my-component></my-component>',
  components: {
    //<my-component> 只能用在父组件模板内
    'my-component': Child
  }
});
Vue.component('parent',Parent);
```

- 语法糖

```
Vue.component('parent',{
  template:'<div>Hello</div><child></child>',
  components: {
    'child': {
      template:'<div>world</div>'
    }
  }
```

模版is特性

```
<tr is="parent"></tr>
```

组件传递数据

```
Vue.component('parent',{
    template:'<div>Hello</div>{{msg}}',
    props:['msg']
});
```

> 父子组件不能通用属性 属性采用-线,props采用驼峰命名

动态绑定Props

- 动态绑定

```
:msg-com="hello"
```

- 注意

```
<!-- 传递了一个字符串 "1" -->
<comp some-prop="1"></comp>

<!-- 传递实际的数字  -->
<comp :some-prop="1"></comp>
```

props验证

- 设置双向绑定

```
<parent :msg.sync="hello"></parent>
```

- 验证规则

```
msg:{
  type:[String, Number], //String Number Object  Function Boolean Array
  default: function () {
    return {name:1}
  },
  twoWay:true,
  validator: function (v) {
    return v==1; //不成立报错
  },
  coerce: function (val) {
    return val; //在赋值前进行操作
  }
}
```

父子组件的通信事件

- `$broadcast` 向下传导给所有后代
- `$dispatch` 沿父链冒泡(包括自己)
- `$emit` 在本身触发

父子组件的通信事件

```
var parent = Vue.extend({
    template:'#parent',
    methods:{
      parent: function () { this.$broadcast('broad','爸爸被点击了');}
    },
    events:{
      emit: function (data) {alert(data);}
    },
});
var child = Vue.extend({
    template:'#child',
    events:{
      broad: function (data) {alert(data);},
      self: function (data) {alert(data);}
    },
    methods:{
      child: function () {
        this.$dispatch('emit','儿子被点击了');
        this.$emit('self','自己被点击了');
      }
    }
});
Vue.component('parent',parent);
```

子组件索引

- 通过v-ref拿到当前组件

```
<div id="parent">
  <user-profile v-ref:p></user-profile>
</div>
var parent = new Vue({el:'#parent'});
console.log(parent.$refs.p);
```

嵌套内容

- 嵌入组件中的内容

```
<user-profile>
  <div>123</div>
  <div>456</div>
</user-profile>
template:"<div>789</div><slot></slot>"
```

- 嵌套内容指定名字

```
<user-profile>
  <div slot="name">123</div>
  <div slot="age">456</div>
  <div slot>890</div>
</user-profile>
template:"<div>789</div><slot name='age'></slot><slot name='name'></slot><slot></s
```

动态引用模版

- 自带的component元素

```
<component :is="current"></component>
```

- 切换components

```
var parent = new Vue({
  el: '#parent',
  data:{
    current:'home'
  },
  components:{
    home:{
      template:'<div>12</div>'
    },
    age:{
      template:'<div>34</div>'
    }
  }
});
```

数据响应

- 通过实例设置数据并响应

```
setTimeout(function () {
  parent.$set('bug',100);
},2000)
```

- 通过全局对象设置

```
var data = {}
var parent = new Vue({
  el: '#parent',
  data: data
});
//parent.bug = 100; //无法响应
Vue.set(data,'bug',500)
```

异步设置数据

- 下一事件环

```
parent.bug = 100;
parent.$el.innerHTML==='100';//无法立即获得数据
Vue.nextTick(function () {
  console.log( parent.$el.innerHTML==='100');
})
```

computed深入用法

- 默认缓存机制

```
var data = {}
var parent = new Vue({
  el: '#parent',
  data:{
    bug:'',
    timer:''
   },
  computed:{
    mess:{
      cache: false, //默认为true
      get: function () {
        return new Date();
      }
    }
  }
});
setInterval(function () {
  parent.$set('timer',parent.mess);
},1000);
```

directive指令

- 创建指令

```
Vue.directive(id, definition)
```

- definition里的参数

```
Vue.directive('my-directive', {
  bind: function () {
  },
  update: function (newValue, oldValue) {
  },
  unbind: function () {
  }
})
```

- 默认可写一个参数

```
Vue.directive('my-directive',function (newValue, oldValue) {
  //默认为update
});
```

指令实例属性

- `el`: 指令绑定的元素。(当前指令元素)
- `vm`: 拥有该指令的上下文 ViewModel。(当前作用域)
- `expression`: 指令的表达式，不包括参数和过滤器。(当前表达式)
- `arg`: 指令的参数。
- `name`: 指令的名字，不包含前缀。(指令名字)
- `modifiers`: 一个对象，包含指令的修饰符。(指令参数的属性)
- `descriptor`: 一个对象，包含指令的解析结果。(描述当前指令)

```
<div v-my-directive:hello.a.e="aa"></div>
```

元素指令

- elementDirective

```
Vue.elementDirective('my-directive', {
  bind: function () {
  // 操作 this.el...
  }
})
```

指令中的高级参数

- 设置指令中的属性

```
<div v-my-directive a="hello"></div>
Vue.directive('my-directive',{
  params:['a'],
  bind: function () {
    console.log(this.params.a);
  }
});
```

- 动态设置指令中的属性

```
Vue.directive('my-directive',{
  params:['a'],
  paramWatchers:{ //监听动态变化
    a: function (old,o) {
      console.log(old,o);
    }
  },
  bind: function () {
  }
});
```

deep深度监听

- 对象内部属性监听

```
<div v-my-directive="obj"></div>
Vue.directive('my-directive',{
  params:['a'],
  deep:true,
  update: function (obj) {
    console.log(obj.name);
  }
});
```

通过指令写回数据

- 指令和数据交互

```
<input type="text" v-model="bug" v-my-directive="bug">
Vue.directive('my-directive',{
  twoWay:true,
  bind: function () {
    this.handle= function () {
      this.set(this.el.value);
    }.bind(this);
    this.el.addEventListener('input',this.handle,false)
  }
});
```

接受内联语句

- 在指令中运算结果

```
<input type="text" v-model="bug" v-my-directive="bug=(bug==124?0:1)">
Vue.directive('my-directive',{
  twoWay:true,
  acceptStatement:true,
  update: function (fn ) {
    console.log(fn());
  }
});
```

mixins

- 抽取共有逻辑(自动合并)

```
var mixins = {
  created: function () {
    this.hello();
  },
  methods:{
    hello: function () {
      console.log('from mixinHello');
    }
  }
}
var com = Vue.component('parent',{
  mixins:[mixins],
  created: function () {
    console.log('from my');
  },
  methods:{
    hello: function () {
      console.log('from myHello');
    }
  }
});
new com;
```

注册全局mixins

- 全局注册混合 (慎用)

```
Vue.mixin({
  created: function () {
    var v = this.$options.dd;
    console.log(v);
  }
})
```

过渡效果

- v-if/v-show

```
.expand-transition {
  height: 40px;
  background: yellow;
  transition:all 2s;
}
.expand-enter, .expand-leave {
  background: red;
}
```

```
<input type="checkbox" v-model="flag">
<div v-if="flag" transition="expand"></div>
```

- 动态绑定

```
<div v-if="flag" :transition="expand"></div>
```

过渡效果

- 绑定状态

```
Vue.transition('fade', {
  beforeEnter: function (el) {
    el.innerHTML='进入之前';
  },
  enter: function (el) {
    setTimeout(function () {
      el.innerHTML='进入中';
    },500);
  },
  afterEnter: function (el) {  el.innerHTML='进入后';  },
  beforeLeave: function (el) {  el.innerHTML='离开之前';  },
  leave: function (el) {
    setTimeout(function () {  el.innerHTML='离开中'},500);
  },
  afterLeave: function (el) {  el.innerHTML='离开后';  },
});
```

v-for中使用过渡

- 让v-for带有transition效果

```
.stagger-transition {
  transition: all .5s ease;
  overflow: hidden;
  margin: 0;
  height: 20px;
}
.stagger-enter, .stagger-leave {
  opacity: 0;
  height: 0;
}
```

```
transition="stagger" stagger="100"
```