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
  }
}
```