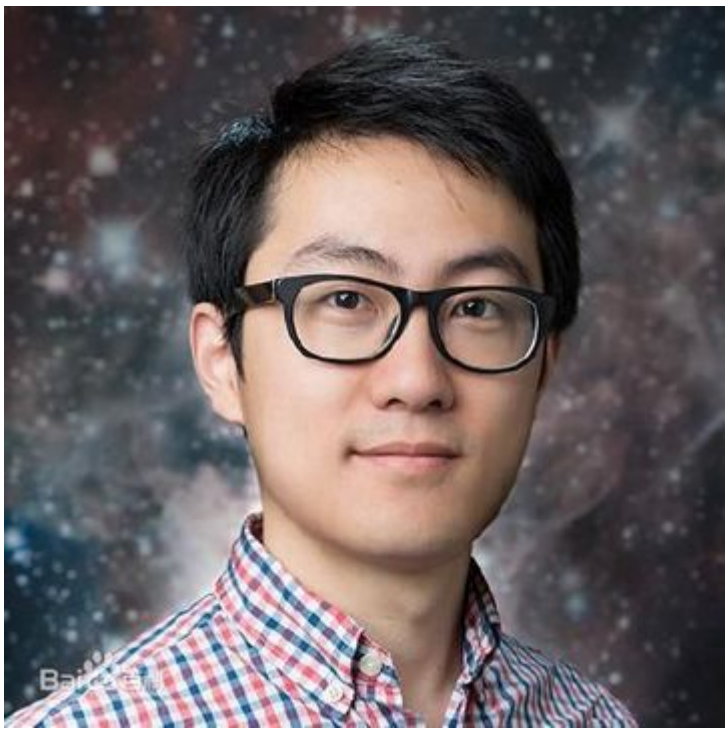
第一章 基础

第一节 Vue介绍

Vue是一套用于构建用户界面的**渐进式框架**，尤雨溪是Vue.js框架的作者，他认为，未来App的趋势是轻量化和细化，能解决问题的应用就是好应用。而在移动互联网时代大的背景下，个人开发者的机遇在门槛低，成本低，跨设备和多平台四个方面。



尤雨溪 毕业于上海复旦附中，在美国完成大学学业，本科毕业于Colgate University，后在Parsons设计学院获得Design & Technology艺术硕士学位，现任职于纽约Google Creative Lab。

2015年10月，正式发布1.0版

2016年5月，正式发布2.0版

浏览器兼容：支持html5 和ES6的浏览器（包括移动端浏览器）

什么是框架

框架（framework）

将那些与业务逻辑无关的重复代码进行封装。其实就是某种应用的半成品，是一组组件，供你选用完成你自己的系统。简单说就是使用别人搭好的舞台，你来做表演。而且，框架一般是成熟的，不断升级的软件。

优点：避免书写大量重复代码，提高开发效率

MVC

MVC是一种开发设计模式，注重对数据的处理

M（model）：模型（数据）

V（view）：视图（html）

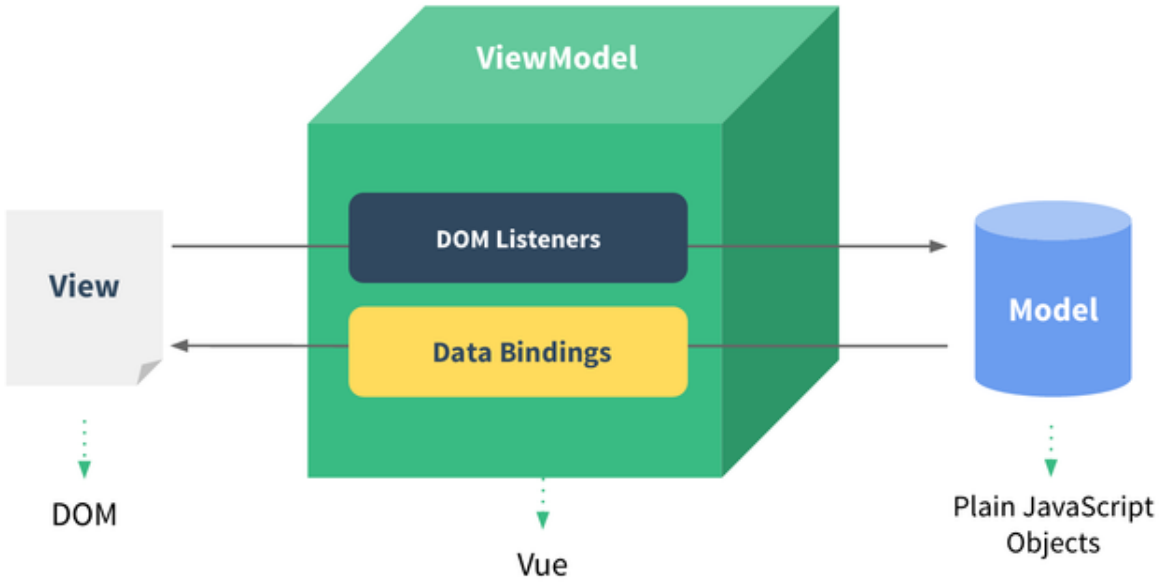
C（Controller）：控制器（js）

Controller将model数据内容显示到view页面视图上

MVC思想划分了MVC三个部分，将数据与视图分离开，而是通过controller控制器来建立联系，这样使开发者方便开发者将注意力集中在数据上，直接对数据进行操作，而无需像js那样先获得视图元素(操作DOM)才能对数据进行操作----即无需操作DOM。

MVVM

MVVM是Model-View-ViewModel（视图模型）的简写。它本质上就是MVC 的改进版。MVVM 就是将其中的View的状态和行为抽象化，视图 UI 和业务逻辑分开。



MVVM模式和MVC模式一样，主要目的是分离视图（View）和模型（Model），有几大优点：

1. 低耦合。视图（View）可以独立于Model变化和修改，一个ViewModel可以绑定到不同的"View"上，当View变化的时候Model可以不变，当Model变化的时候View也可以不变。
2. 可重用性。你可以把一些视图逻辑放在一个ViewModel里面，让很多view重用这段视图逻辑。
3. 独立开发。开发人员可以专注于业务逻辑和数据的开发（ViewModel），设计人员可以专注于页面设计。
4. 可测试。界面素来是比较难于测试的，而现在测试可以针对ViewModel来写。

安装

Vue就是一个js文件，Vue官网（多语言）有下载说明

辅助工具：Vue Devtools（chrome调试工具）

第二节 hello World

```
<body>
    <!--View-->

    <div id="app">
```

```
        <h1>你好，{{ message }}</h1>
    </div>
</body>

<!--------------------------vue.js---------------------------------->
<script src="js/vue.js"></script>
<script>
    //Model--必须是对象
    var model = {
        name: 'Vue.js'
    }

    //创建一个viewModel
    let vm=new Vue({
        el: '#app',
        data:model //model
   })
</script>
```

> 实例化
>
> 实例化是指在**面向对象**的编程中，通常把用类创建对象的过程称为实例化。实例化一个对象就是为对象开辟内存空间。或者是不用声明，直接使用new 构造函数名，建立一个临时对象

实例配置项

上面代码，使用 let vm=new Vue({.....}) 创建了一个Vue实例，以对象方式传入配置项，每一个配置项代表一个功能模块，常用的配置项，包括：

配置项	说明
el	把 Vue 实例挂载到一个已有内容的元素上， 通过id绑定html元素（可以用 vm.$el 访问）
data	Vue 实例的数据对象
methods	事件处理器，事件所要触发的函数
computed	计算属性（当数据需要经过处理计算后才能得到结果的）
watch	监听器 检测指定的数据发生改变
components	组件容器
router	路由
hook函数（8个）	hook（钩子）函数，不同生命周期引发的动作
template	定义模板，可以是字符串，也可以是”#“选择器
props	用于接收来自父组件的数据，可以是数组或对象

实例属性/方法

vm的 属性 和 方法

属性	说明
$el	dom元素
$date	Vue的date配置项

方法	说明
vm.$emit()	子组件可以使用 $emit 触发父组件的自定义事件（组件章节详细说明）
vm.$set()	Vue.set的别名 设置对象的属性， 这个方法主要用于避开 Vue 不能检测属性被添加的限制
vm.$on()	监听当前实例上的自定义事件。事件可以由`vm.$emit`触发。回调函数会接收所有传入事件触发函数的额外参数。

全局API

API	说明
Vue.extend(options)	用基础 Vue 构造器，创建一个“子类”。 参数是一个包含组件选项的对象。
Vue.set(target, key, value)	设置对象的属性。如果对象是响应式的， 确保属性被创建后也是响应式的，同时触发视图更新。
Vue.component(id, [definition])	注册或获取全局组件。 注册还会自动使用给定的`id`设置组件的名称。
Vue.use(plugin)	安装 Vue.js 插件(插件通常会为 Vue 添加全局功能) Vue.js 官方提供的一些插件 (例如 `vue-router`) 在检测到 `Vue` 是可访问的全局变量时会自动调用Vue.use()。 在 CommonJS 的模块环境中，应该显式地调用Vue.use()

第三节 Vue常用指令

常用指令：v-xxx形式，是对标签属性功能的扩展

闪屏指令

指令	说明
v-cloak	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模板标签直到实例准备完毕。

数据渲染

指令	说明
{{}}	定界符 {{}}内部是表达式，是在data中数据，并且，支持各种逻辑运算和算术运算
v-text	内容为字符串
v-html	可以识别HTML标签元素

条件渲染

指令	说明
v-if	据其后表达式的bool值进行判断是否渲染该元素

```
<div id="example01">
    <p v-if="name.indexOf('san')>0">姓名:{{name}}</p>
    <p v-if="man">男</p>
    <p v-if="women">女</p>
    <p v-if="age>25">年龄:{{age}}</p>
</div>

<div v-if="city=='北京'">
    城市： {{city}}
</div>
<div v-else>
    城市 ：不在北京
</div>

var vm= new Vue({
    el:"#example01",
    data:{
        name:'zhangsan',
        age:30,
        man:true,
        women: false,
        city:'北京',
    }

})
```

循环渲染

指令	说明
v-for	遍历数组和JSON

```
!--循环数值-->
<span v-for="n in num">{{ n + 4 }} </span>

<!--循环对象 可以有 3 个参数-->
<li v-for="value in obj">
    {{value}}
</li>
<li v-for="(value,key) in obj">
    {{value}}:{{key}}
</li>
<li v-for="(value,key,index) in obj">
   {{key}}:{{value}}:{{index}}
</li>

<!--循环数组 可以有 2 个参数-->
list = [1,7,15];
<span v-for="item in list">{{ item }} </span>

list = [1,7,15];
<span v-for="(item,index) in list">{{index}}:{{item}} </span>

data:{
    num:6,
    obj:{ "name":"zhangsan", "age":18 },
    list:[1,7,15]
}
```

数组的更新检查

数组方法	说明
push()、pop()、shift()、unshift() splice()、sort()、reverse()	可触发视图更新
filter(), concat() 和 slice()	不触发视图更新 它们可以返回新数组，用新数组替换旧数组，就可以刷新视图

受 JavaScript 的限制，**Vue 不能检测以下变动的数组**：

1. 当你利用索引直接设置一个项时，例如：`vm.items[i] = newValue`
2. 当你修改数组的长度时，例如：`vm.items.length = 0`

```
Vue.set(array, index, newValue)
```

对象的更新检查

受 JavaScript 的限制，**Vue 不能检测对象属性的添加或删除**：

```
Vue.set(object, key, value)
```

显示隐藏

指令	说明
v-show	控制dom元素隐藏/显示(display:none)

```
<p v-show="true">布尔值控制显示隐藏</p>
```

事件绑定

指令	说明
v-on:click	简写: @click="" click、mouseover、mouseout、mouseup、mousedown、 dblclick 、contextmenu、keydown、keyup等 **事件所要触发的函数，被定义在methods配置项中**

表单数据

指令	说明
v-model	双向数据绑定 v-model支持的input类型：text、password、search 和 <textarea> <select>标签

```
<!--type=text-->
<input type="text" name="username" v-model="username">

<!--type=radio-->
<input type="radio" name="sex" value="m" v-model="sex" >男
<input type="radio" name="sex" value="w" v-model="sex">女

<!--select-->
<select name="sel" v-model="city">
    <option value="1">北京</option>
    <option value="2">上海</option>
    <option value="3">广州</option>
</select>
```

```
<!--checkbox-->
<input type="checkbox" name="hobby[]" value="1" v-model="hobby"/>足球
<input type="checkbox" name="hobby[]" value="2" v-model="hobby"/>篮球
<input type="checkbox" name="hobby[]" value="3" v-model="hobby"/>游戏
<input type="checkbox" name="hobby[]" value="4" v-model="hobby"/>读书

data: {
    //表单的值类型：字符串
    username:"zhangsan",
    sex："m",
    city:"2",
    hobby:["1","3"]
},
```

属性绑定

指令	说明
v-bind	动态改变dom标签上的属性 v-bind :class=""　简写 :class=""

```
//对象分写 ac1 ac2 是类名
<p :class="{'ac1':bool1,'ac2':bool2}">火星时代</p>
data:{
    bool1:false,
    bool2:true
}

//行内样式
<div :style="{color:activeColor, fontSize:fontSize+'px'}">火星时代</div>
data: {
    activeColor:'red',
    fontSize: 30
}

//行内样式--data对象
<div :style="styleObj"></div>
data: {
    styleObj: {
        color:'red',
        fontSize:'13px'
    }
}
```

```
//数组表达式
<h2 class="ac" :class="bool?'red':''">abcd</h2>

<h3 :class="['ac', bool ? 'red':'']" @click="change">abcd</h3>

<h4 :class="[bool ? class1 : '', class2]" @click="change">abcd</h4>
```

```
data: {
    class1: 'ac1',
    class2: 'ac2',
    bool: false
}
methods: {
    change: function () {
        this.bool = !this.bool;
    }
}
```

```
//拼 字符串
<a v-for="(item,index) in list" :href="'doAction.php?id='+item.id">{{item.name}}</a>
```

第四节 计算属性和侦听器

computed

计算属性是基于它们的依赖进行缓存的。计算属性只有在它的相关依赖发生改变时才会重新求值。**计算属性具有缓存。**

watch

侦听器watch是侦听一个特定的值，当该值变化时执行特定的函数。

computed与watch的差别

computed	**watch**
依赖其它的属性计算所得出最后的值	监听一个属性值的变化，然后执行相对应的函数
读取数据→计算→缓存	监听的属性值变化时，执行回调
简单计算	watch回调里会传入监听属性的新、旧值，通过这两个值可以做一些特定的操作

1. computed适合**用已经声明的变量可以计算出来的另一个变量**用来渲染页面，例如：一个数据依赖于其他数据，那么把这个数据设计为computed的
2. watch适合监听某个变量来操作一些逻辑行为，例如：监听分页值变化，然后在回调函数中发起异步请求，获取后台数据。

第五节 事件对象

鼠标事件

click、contextmenu、dlclick、mouseover、mouseout、mouseup、mousedown

修饰符	说明
prevent	阻止默认行为 <a :href="links.hxsd" @click.prevent="cancel">hxsd
stop	阻止冒泡 <button @click.stop="go">发送</button>

键盘事件

keydown、keyup

修饰符	说明
up	up键
down	down键
left	left键
right	right键
enter	enter键
ctrl	ctrl键
delete	delete键
@keyup.keycode	<input type="text" @keyup.27="show()"> ESC键=27

$event

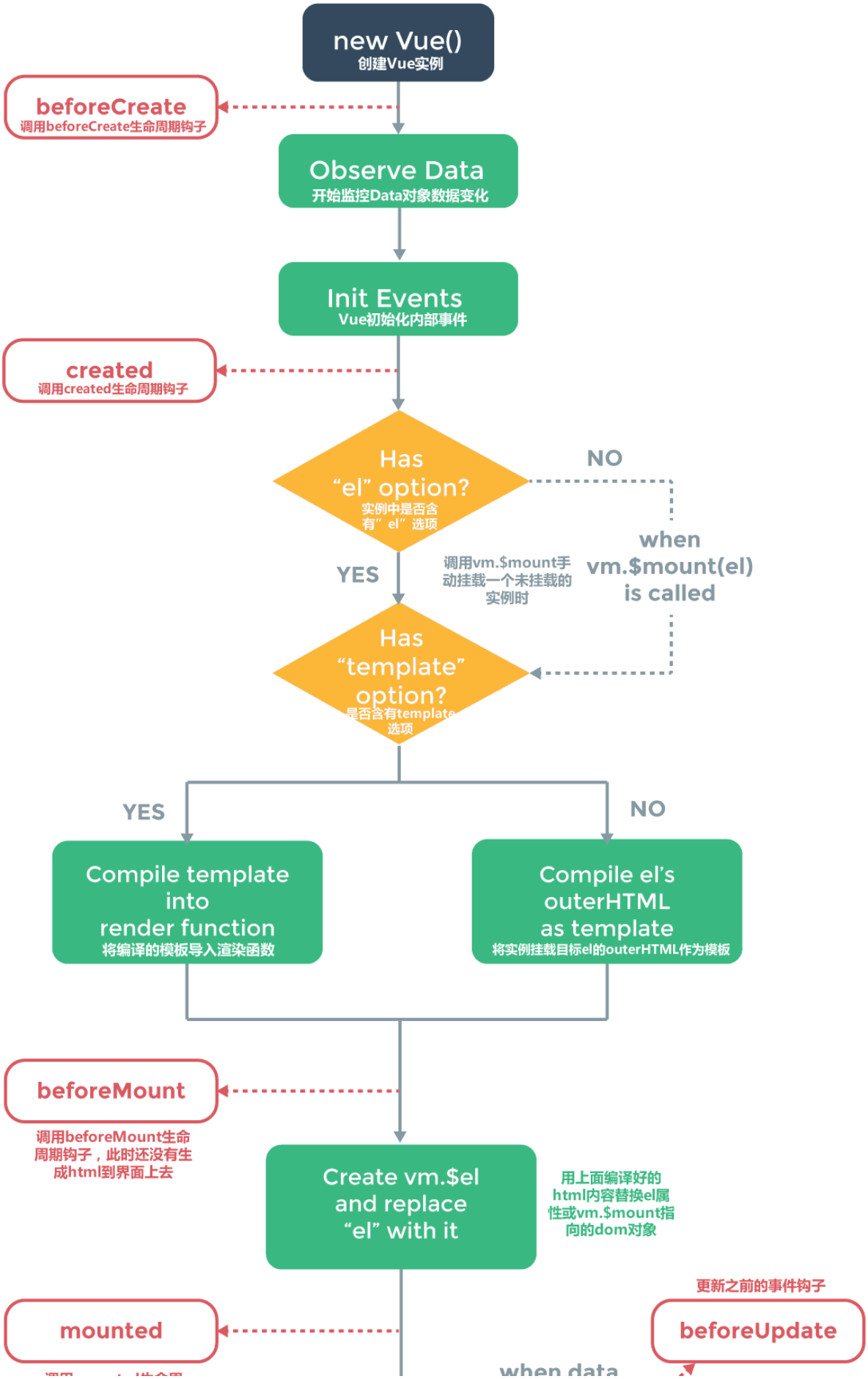
访问原生 DOM 事件对象，可以用特殊变量 $event 把它传入方法：

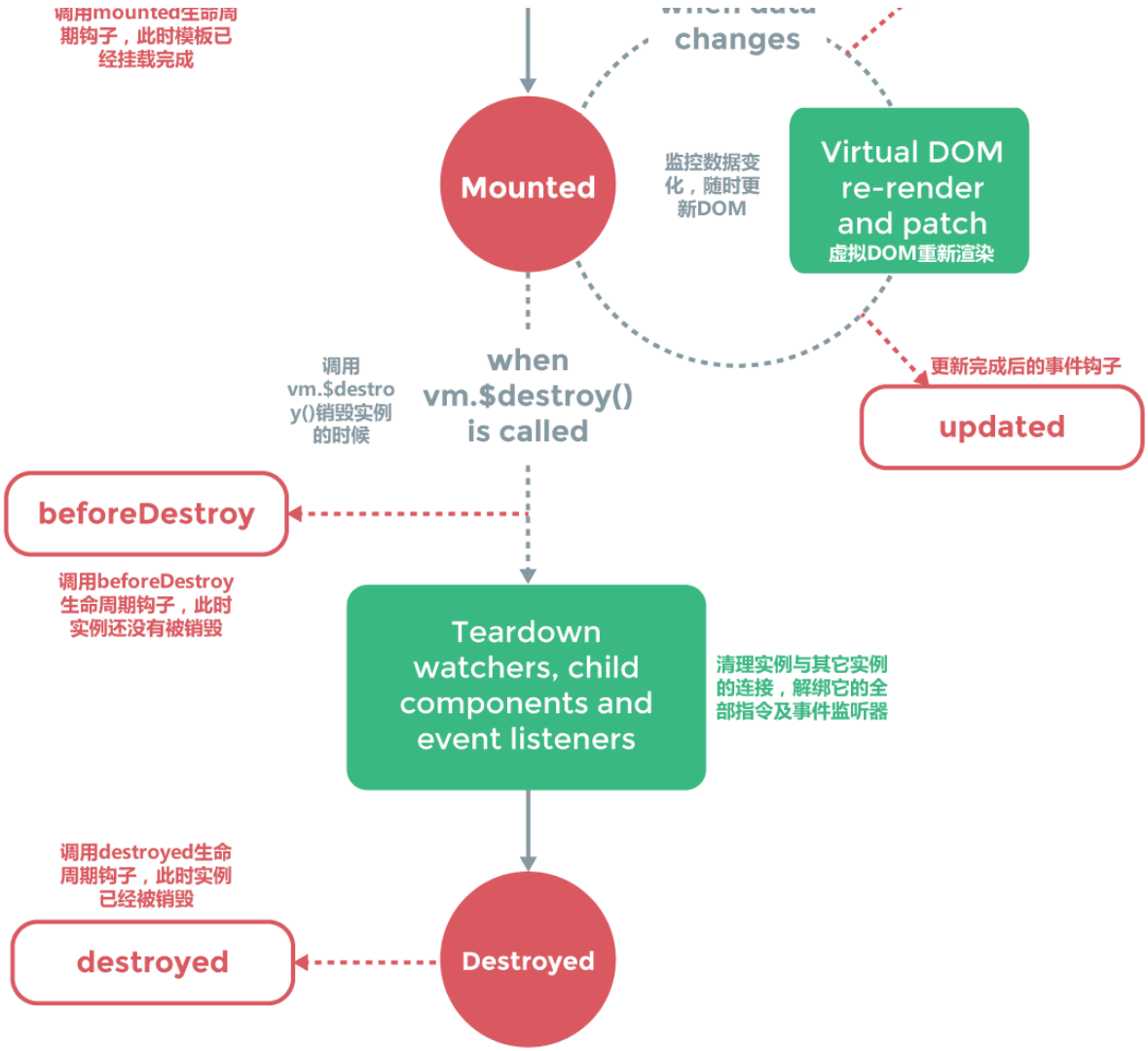
```
//同时传递参数 和 事件对象，$event放在最后
<button type="button" @click="btn_fn(10,$event)">click</button>

methods:{
    //事件函数如果不传参数，默认就有一个参数--ev
    btn_fn:function(i,ev){
        console.log(i,ev); //同时定义 参数 和 事件对象
    }
}
```

第六节 生命周期与钩子

生命周期

创建→挂载→更新→销毁


new Vue()
创建Vue实例
beforeCreate
调用beforeCreate生命周期钩子
Observe Data
开始监控Data对象数据变化
Init Events
Vue初始化内部事件
created
调用created生命周期钩子
Has "el" option?
实例中是否含有" el" 选项
NO
when vm.$mount(el) is called
调用vm.$mount手动挂载一个未挂载的实例时
YES
Has "template" option?
是否含有template选项
YES
NO
Compile template into render function
将编译的模板导入渲染函数
Compile el's outerHTML as template
将实例挂载目标el的outerHTML作为模板
beforeMount
调用beforeMount生命周期钩子，此时还没有生成html到界面上去
Create vm.$el and replace "el" with it
用上面编译好的html内容替换el属性或vm.$mount指向的dom对象
更新之前的事件钩子
beforeUpdate
mounted
when data



钩子函数

生命周期钩子	详细
beforeCreate	实例创建前
created	实例创建后
beforeMount	载入前
mounted	载入后
beforeUpdate	更新前
updated	更新后
beforeDestroy	实例销毁前
deactivated	实例销毁后

代码演示

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>生命周期</title>
    <script src="vue.js"></script>
</head>
<body>
    <div id="box">
      <p id="para">{{msg}}</p>
    </div>
</body>
</html>
<script>
    var vm = new Vue({
        el:'#box',
        data:{
            msg:'word'
        },
        beforeCreate:function(){
            console.log('beforeCreate---------------创建前');
            //vue实例本身自带一些属性：$el、$data
            console.log('el:'+this.$el);
            console.log('data:'+this.$data);
            console.log('msg:'+this.msg);
        },
        created:function(){
            console.log('created--------------------创建完成');
            console.log('el:'+this.$el);
            console.log('data:'+this.$data);
            console.log('msg:'+this.msg);
        },
        beforeMount:function(){
            console.log('beforeMount----------------挂载前');
            console.log('el:'+this.$el);    //占位
            console.log('data:'+this.$data);
            console.log('msg:'+this.msg);
            var para = document.getElementById('para').innerText;
            console.log('para:'+para);
        },
        mounted:function(){
            console.log('mounted--------------------挂载完成');
            console.log('el:'+this.$el);
            console.log('data:'+this.$data);
            console.log('msg:'+this.msg);
            var para = document.getElementById('para').innerText;
            console.log('para:'+para);
        },
        beforeUpdate:function(){
            console.log('beforeUpdate---------------更新前');
            console.log('el:'+this.$el);    //控制台更改msg数据
            console.log('data:'+this.$data);

            console.log('msg:'+this.msg);
```

```
            var para = document.getElementById('para').innerText;
            console.log('para:'+para);
        },
        updated:function(){
            console.log('updated--------------------更新完成');
            console.log('el:'+this.$el);
            console.log('data:'+this.$data);
            console.log('msg:'+this.msg);
            var para = document.getElementById('para').innerText;
            console.log('para:'+para);
        },
        beforeDestroy:function(){
            console.log('beforeDestroy--------------销毁前');
            console.log('el:'+this.$el);
            console.log('data:'+this.$data);
            console.log('msg:'+this.msg);
        },
        destroyed:function(){
            console.log('destroyed------------------销毁完成');
            console.log('el:'+this.$el);
            console.log('data:'+this.$data);
            console.log('msg:'+this.msg);
        }
    })
</script>
```

第三章 Vue组件

第一节 组件化开发基础

简介

将结构上(HTML)相近或相同的代码进行封装，成为一个高度可重复使用的部件称为组件，组件化开发具有高度可复用性，大大减少冗余代码，组件具有低耦合性，在大型项目中，便于团队协作开发

> 耦合性(Coupling)，也叫耦合度，是指模块之间的依赖关系，包括控制关系、调用关系、数据传递关系。模块间联系越多，其耦合性越强，同时表明其独立性越差。与之相反的是内聚度，是一个模块内部各成分彼此结合的紧密程度，软件设计中通常用耦合度和内聚度作为衡量模块独立程度的标准。划分模块的一个准则就是高内聚低耦合。

组件实例的作用域是孤立的，组件不能彼此直接使用对方的数据

使用组件

使用组件一个有4个步骤：

定义模板→定义组件→注册组件→引用组件

定义模板

```
<!--定义模板-->
<template id="temp">
    <div><!--需要一个容器-->
        <h3>{{title}}</h3>
    </div>
</template>
```

注意：

1. 将<template>模板保存到Vue实例挂载元素（el绑定id的元素）的外面
2. 因为 Vue 只有在浏览器解析、规范化模板**之后**才能获取其内容。尤其要注意，像 `<ul>`、`<ol>`、`<table>`、`<select>` 这样的元素里允许包含的元素有限制，而另一些像 `<option>` 这样的元素只能出现在某些特定元素的内部。

定义组件

```
//使用Vue.extend
//extend是构造一个组件的语法器（子类）. 你给它参数 他给你一个组件
let MyTemp=Vue.extend({
    template:'#temp', //模板id
    data: function(){ //必须为函数
        return { //必须有return，返回值为对象{}
            title:"hxsd"
        }
    }
});
```

注册组件

```
//方法1----使用Vue.component
Vue.component('myComponent',MyTemp);
let app=Vue({
    el:"#box",
});

//方法2----在配置项中注册组件（局部注册）
let app=Vue({
    el:"#box",
    components:{
        myTemp
    }
});
```

引用组件

```
<!--在Vue实例中使用组件-->
<div id='box'>
    <!--组件名如果用驼峰定义，改为短横线命名-->
    <my-temp></my-temp>
</div>
```

注意事项

模板：使用**id**绑定

命名：组件名如果用**驼峰**定义，html中**引用组件时**，改为**短横线**命名

局部组件（子组件）

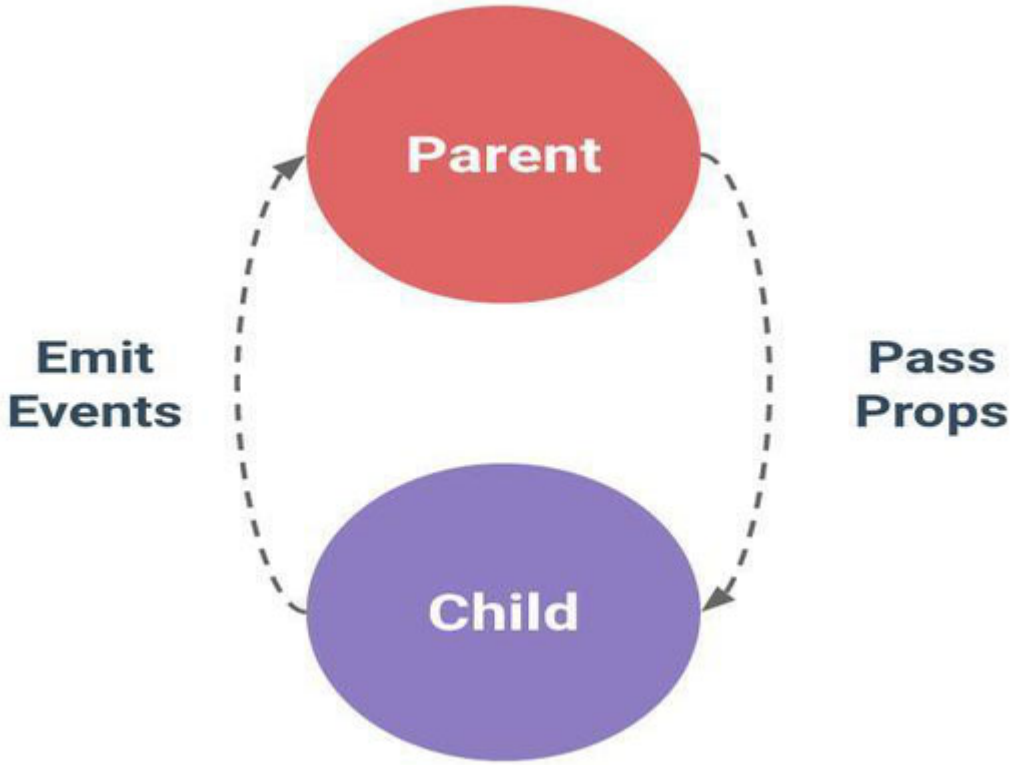
```
new Vue({
    el:'#box',
    data:{},
    methods:{
        change:function(){
            alert('hxsd')
        }
    },
    components:{
        myTest:{
            template:'#tmp',
            data:function(){
                return {msg:'hxsd'}
            }
        }
    }
})
```

第二节 组件间的通信

组件之间是相互独立的，不能共享数据。

父子通讯

父子组件之间的数据操作，是通过**props**属性和**$emit()**方法来实现的。



props

当子组件需要读取父组件的数据（或者说，父组件要将数据传入子组件），子组件通过props自定义属性，来接收父组件的数据（或者说，父组件通过 props将数据下发到子组件），这是一个单向传递。

- 声明位置：props 属性**在子组件中声明**
- 属性值：**在组件模板中绑定**

$emit事件触发

子组件向父组件传值（或者说，子组件要修改父组件的数据），使用$emit（"自定义事件"）**自定义事件不能使用驼峰命名**

声明位置：在子组件上声明**自定义事件**

```
//定义组件
let pagetmp=Vue.extend({
    template:"#page",//指定模板
    props:["ap","pi"],   //ap:接收总页数  pi：接收分页索引
    methods:{
        flip:function(i){
        //触发父级事件
        this.$emit("zijichange",i);
        }
    }
});
```

非父子通信

可以实例化一个vue实例，相当于一个第三方

```
let bus = new Vue(); //创建一个新实例
```

```
//组件A
<div @click="run"></div>
methods: {
    run:function() {
        bu.$emit('名称',传递的数据); //触发事件
    }
}
```

```
//组件B
<div></div>
created() {
  bus.$on('名称',function(arg){
      console.log ( agr ) ;//arg是接受传递来的数据
  });
}
```

第三节 slot

为提高组件的灵活性，在定义组件时，不确定组件内容，在组件内预留一些slot，引用组件的时候传入内容

```
<!--引用组件-->
<my-computer>
    <h2>我的diy电脑</h2>
    <p slot="CPU">intel i7</p>
    <p slot="memery">16G</p>
    <p slot="motherboard">华硕主板</p>
    <p slot="ssd">240G 固态硬盘</p>
</my-computer>

<!--模板-->
<template id="computer">
    <div>
        <slot></slot><!---<h2>我的diy电脑</h2> 没有匹配的slot name 放入这里----->
        <slot name="CPU"></slot>
        <slot name="memery"></slot>
        <slot name="motherboard"></slot>
        <slot name="ssd"></slot>
    </div>
</template>
```

第四章 路由

第一节 路由概述

什么是路由

路由---是网络技术中的一个名词，意为到达目的地所选择的路线。

当前，**SPA**（single page web application）单页应用程序越来越流行；SPA借用路由这一个概念，通过改变url地址的#hash值，来显示不同内容（模块），SPA的好处是：

- 良好的交互体验：用户不需要重新刷新页面，页面显示流畅。
- 前后端工作分离模式：服务器只用出数据就可以，不用管展示逻辑和页面合成，减轻服务器压力。

hash（哈希值）

url中出现的 " # " 符号，代表网页中的一个位置（锚点）。其右面的字符，就是该位置的标识符。浏览器读取这个URL后，会自动将该位置（锚点）滚动至可视区域。

#是用来指导浏览器动作的，对服务器端完全无用 ，所以，也不会被发送到服务器端。

hash值会被浏览器的历史记录（window.history）保存。

第二节 实现路由

引入vue-router.js（在vue.js后），创建路由对象：

```
//创建路由对象
var rt=new VueRouter({
    routes:[
        {
            path: '/',   //默认页面
            redirect:"/home"  //默认模块
        },

        {
            path:"/home",
            component:home
        },
        {
            path:"/news",//有二级路由，不能使用name
            component:news,

            //一级路由下面还可以有二级路由，也就是页面当中，还有小的模块切换。
            children:[
                {
                    path:"/",
                    redirect:'sub1'
                },
                ......
})
```

routes的主要配置项

配置项	说明
path	路径 / 是#符号后的位置
name	命名路径
component	单个模板 值：模板（字符串）
components	多个模板 值：{名称：模板，名称：模板}
redirect	重定向 定义默认页
children	子路由

注册、使用路由

```
let app = new Vue({
    el: "#app",
    router:rt,
})
```

router-link 组件

router-link 组件支持用户在具有路由功能的应用中（点击）导航。 通过`to`属性指定目标地址，默认渲染成带有正确链接的`<a>`标签，可以通过配置`tag`属性生成别的标签.。

当目标路由成功激活时，链接元素自动设置一个表示激活的 CSS 类名：**router-link-active**。

```
<!--路由切换-->
<ul>
    <li><router-link to='/home'>主页</router-link></li>
    <li><router-link to='/news'>新闻</router-link></li>
</ul>

<!--路由视图区-->
<router-view></router-view>
```

$router属性与方法

new VueRouter( ) 将得到一个路由的实例，该实例是一个全局的对象（$router），该对象的常用属性和方法：

属性

属性	说明
app	应用router的vue对象
mode	路由使用的模式 "hash" | "history" | "abstract"
currentRoute	当前路由对应的路由信息对象 **$route** 注意：$router 与 $tourte

方法

方法	说明
push	导航到不同的 URL

$route对象的属性

$tourte是当前路由的信息对象

属性	说明
$route.path	对应当前路由的路径，总是解析为绝对路径，如 `"/foo/bar"`。
$route.params	一个 key/value 对象，包含了动态片段和全匹配片段，如果没有路由参数，就是一个空对象。
$route.query	一个 key/value 对象，表示 URL 查询参数

路由组件之间的通信

官方推荐的event bus 解决方案的缺陷在于： 在数据传递过程中，两个组件必须都已经被渲染过。

路由的组件是分别渲染，所以无法传递数据。

vuex

Vuex 是一个专为 Vue.js 应用程序开发的**状态管理模式。**它采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。Vuex 也集成到 Vue 的官方调试工具 devtools extension，提供了诸如零配置的 time-travel 调试、状态快照导入导出等高级调试功能。

vuex适用于：

- 多个视图依赖于同一状态。
- 来自不同视图的行为需要变更同一状态。

应用vuex

引入vuex.js文件

```
const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment (state) {
      state.count++
    }
  }
})
```

可以通过 `store.state` 来获取状态对象，以及通过 `store.commit` 方法触发状态变更：

```
store.commit('increment');
console.log(store.state.count) // -> 1
```

第三节 过渡动效

Vue 在插入、更新或者移除 DOM 时，提供多种不同方式的应用过渡效果。

包括以下工具：

- 在 CSS 过渡和动画中自动应用 class
- 可以配合使用第三方 CSS 动画库，如 Animate.css
- 在过渡钩子函数中使用 JavaScript 直接操作 DOM
- 可以配合使用第三方 JavaScript 动画库，如 Velocity.js

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

- 条件渲染 (使用 `v-if`)
- 条件展示 (使用 `v-show`)
- 动态组件
- 组件根节点

```
/*css*/
.fade-enter-active, .fade-leave-active {  transition: opacity .5s}
.fade-enter, .fade-leave-to { opacity: 0}
```

```
<!--html-->
<div id="demo">
  <button v-on:click="show = !show">
    Toggle
  </button>
  <transition name="fade">
    <p v-if="show">hello</p>
  </transition>
</div>
```

```
//js
new Vue({
  el: '#demo',
  data: {
    show: true
  }
})
```