Vue 学习笔记

介绍

三大前端框架

Frame	Angular	React	Vue
作者	google 公司	facebook	尤雨溪
组织方式	MVC	模块化	模块化
数据绑定	双向绑定	单向绑定	双向绑定
模板能力	强大	自由	自由
自由度	较小	大	较大
路由	静态路由	动态路由	动态路由

官方文档学习

为什么要使用前端框架？

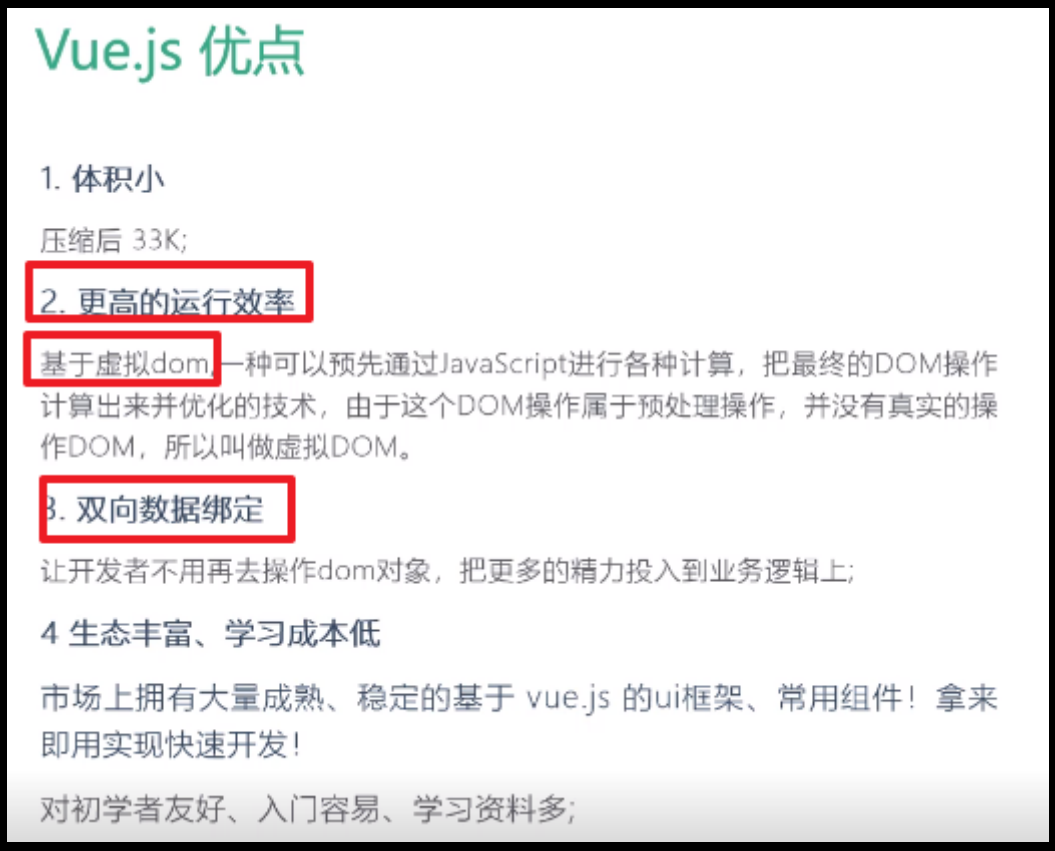
Angular、React、Vue
为了把**后端的业务逻辑移动到前端**实现
JS 文件会变得很多，使用框架可以简化

Vue 是什么

Vue 是一套用于构建用户界面的渐进式框架

Vue 特点

1. 渐进式框架：自底向上，由简单到复杂
2. 双向数据绑定
3. 不需要操作 DOM、状态机。状态改变会引起视图的更新
4. 可以构建大型应用，也可以嵌入到一个应用中
5. 环境构建方便
6. 单文件组件：组件化开发
7. 社区强大：插件、文档等



Vue 是一套用于构建用户界面的渐进式框架。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

声明式渲染

Vue.js 的核心是一个允许采用简洁的模板语法来声明式地将数据渲染进 DOM 的系统：{{ message }}
不再和 HTML 直接交互了。一个 Vue 应用会将其挂载到一个 DOM 元素上

v-bind attribute 被称为指令。指令带有前缀 v-，以表示它们是 Vue 提供的特殊 attribute。可能你已经猜到了，它们会在渲染的 DOM 上应用特殊的响应式行为。

条件与循环

控制切换一个元素是否显示也相当简单：v-if="seen"

v-for 指令可以绑定数组的数据来渲染一个项目列表

```
<li v-for="todo in todos">
    {{ todo.text }}
  </li>
```

处理用户输入

为了让用户和你的应用进行交互，我们可以用 v-on 指令添加一个事件监听器，通过它调用在 Vue 实例中定义的方法：

```
<button v-on:click="reverseMessage">反转消息</button>
```

Vue 还提供了 v-model 指令，它能轻松实现表单输入和应用状态之间的双向绑定。

组件化应用构建

组件系统是 Vue 的另一个重要概念，因为它是一种抽象，允许我们使用小型、独立和通常可复用的组件构建大型应用。仔细想想，几乎任意类型的应用界面都可以抽象为一个组件树

在 Vue 里，一个组件本质上是一个拥有预定义选项的一个 Vue 实例。在 Vue 中注册组件很简单：

```
// 定义名为 todo-item 的新组件
Vue.component('todo-item', {
  template: '<li>这是个待办项</li>'
})

var app = new Vue(...)
```

Vue 实例

每个 Vue 应用都是通过用 Vue 函数创建一个新的 Vue 实例开始的：

```
var vm = new Vue({
  // 选项
})
```

虽然没有完全遵循 MVVM 模型，但是 Vue 的设计也受到了它的启发。因此在文档中经常会使用 vm (ViewModel 的缩写) 这个变量名表示 Vue 实例。

只有当实例被创建时就已经存在于 data 中的 property 才是响应式的。

除了数据 property，Vue 实例还暴露了一些有用的实例 property 与方法。它们都有前缀 $，以便与用户定义的 property 区分开来。

```
// $watch 是一个实例方法
vm.$watch('a', function (newValue, oldValue) {
  // 这个回调将在 `vm.a` 改变后调用
```

```
})
```

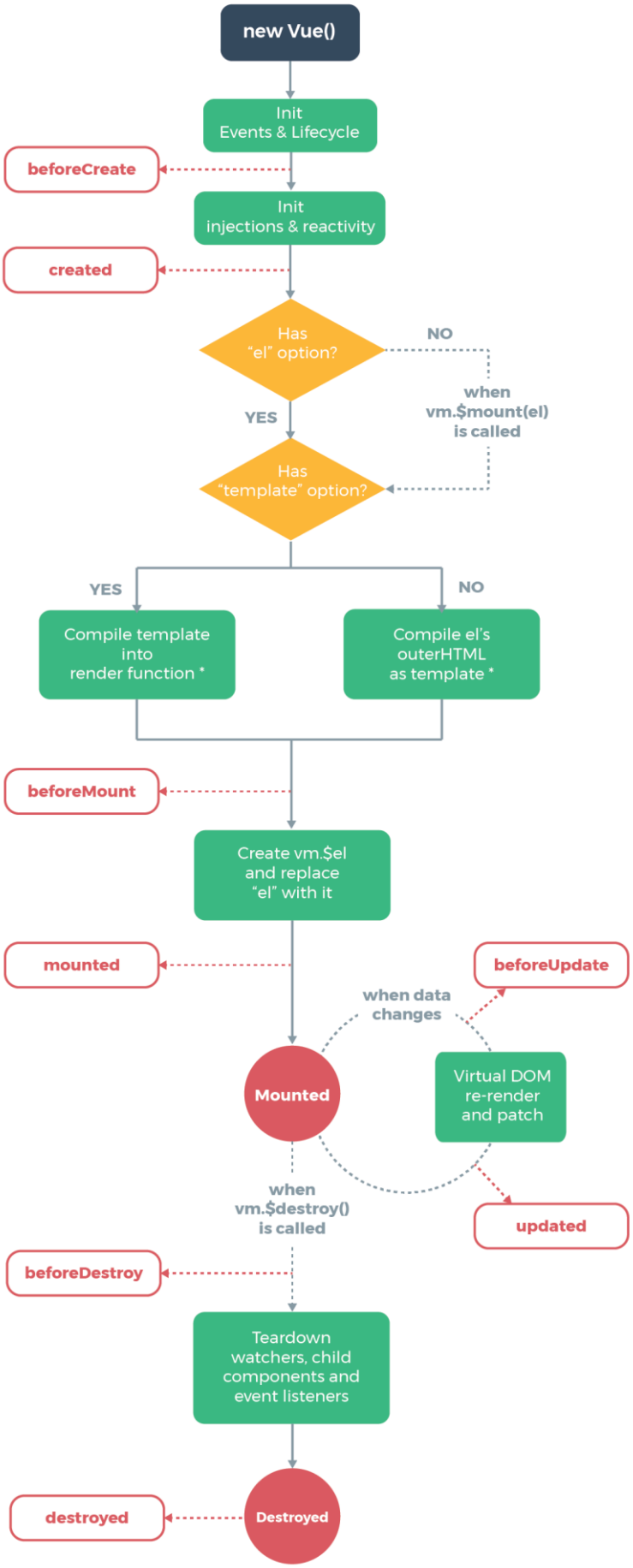
实例生命周期钩子

每个 Vue 实例在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到 DOM 并在数据变化时更新 DOM 等。同时在这个过程中也会运行一些叫做生命周期钩子的函数，这给了用户在不同阶段添加自己的代码的机会。

1. Created
2. Mounted
3. Updated
4. Destroyed
5.

不要在选项 property 或回调上使用箭头函数，比如 created: () => console.log(this.a) 或 vm.$watch('a', newValue => this.myMethod())。因为箭头函数并没有 this，this 会作为变量一直向上级词法作用域查找，直至找到为止，经常导致 Uncaught TypeError: Cannot read property of undefined 或 Uncaught TypeError: this.myMethod is not a function 之类的错误。

生命周期图示



* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

模板语法

Vue.js 使用了基于 HTML 的模板语法，允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据

数据绑定最常见的形式就是使用“Mustache”语法 (双大括号) 的文本插值

通过使用 v-once 指令，你也能执行一次性地插值，当数据改变时，插值处的内容不会更新。但请留心这会影响到该节点上的其它数据绑定：

```
<span v-once>这个将不会改变: {{ msg }}</span>
```

双大括号会将数据解释为普通文本，而非 HTML 代码。为了输出真正的 HTML，你需要使用 v-html 指令

```
<p>Using mustaches: {{ rawHtml }}</p>
<p>Using v-html directive: <span v-html="rawHtml"></span></p>
```

你的站点上动态渲染的任意 HTML 可能会非常危险，因为它很容易导致 XSS 攻击。请只对可信内容使用 HTML 插值，绝不要对用户提供的内容使用插值。

Mustache 语法不能作用在 HTML attribute 上，遇到这种情况应该使用 v-bind 指令：

```
<div v-bind:id="dynamicId"></div>
```

对于布尔 attribute (它们只要存在就意味着值为 true)，v-bind 工作起来略有不同，在这个例子中：

```
<button v-bind:disabled="isButtonDisabled">Button</button>
```

如果 isButtonDisabled 的值是 null、undefined 或 false，则 disabled attribute 甚至不会被包含在渲染出来的 <button> 元素中。

对于所有的数据绑定，Vue.js 都提供了完全的 JavaScript 表达式支持

```
{{ number + 1 }}
{{ ok ? 'YES' : 'NO' }}
{{ message.split('').reverse().join('') }}
<div v-bind:id="'list-' + id"></div>
```

有个限制就是，每个绑定都只能包含单个表达式

指令

指令 (Directives) 是带有 v- 前缀的特殊 attribute。指令 attribute 的值预期是单个 JavaScript 表达式 (v-for 是例外情况，稍后我们再讨论)。指令的职责是，当表达式的值改变时，将其产生的连带影响，响应式地作用于 DOM。

一些指令能够接收一个“参数”，在指令名称之后以冒号表示。例如，v-bind 指令可以用于响应式地更新 HTML attribute:

```
<a v-bind:href="url">...</a>
<a v-on:click="doSomething">...</a>
```

从 2.6.0 开始，可以用方括号括起来的 JavaScript 表达式作为一个指令的参数:

```
<a v-bind:[attributeName]="url"> ... </a>
```

修饰符 (modifier) 是以半角句号 . 指明的特殊后缀，用于指出一个指令应该以特殊方式绑定。例如，.prevent 修饰符告诉 v-on 指令对于触发的事件调用 event.preventDefault():

```
<form v-on:submit.prevent="onSubmit">...</form>
```

缩写

Vue 为 v-bind 和 v-on 这两个最常用的指令，提供了特定简写:

```
<!-- 完整语法 -->
<a v-bind:href="url">...</a>

<!-- 缩写 -->
<a :href="url">...</a>
```

```
<!-- 完整语法 -->
<a v-on:click="doSomething">...</a>

<!-- 缩写 -->
<a @click="doSomething">...</a>
```

计算属性和侦听器

模板内的表达式非常便利，但是设计它们的初衷是用于简单运算的。在模板中放入太多的逻辑会让模板过重且难以维护

所以，对于任何复杂逻辑，你都应当使用计算属性。

```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>
var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hello'
  },
  computed: {
    // 计算属性的 getter
    reversedMessage: function () {
      // `this` 指向 vm 实例
      return this.message.split('').reverse().join('')
    }
  }
})
```

这里我们声明了一个计算属性 reversedMessage。我们提供的函数将用作 property vm.reversedMessage 的 getter 函数：

```
console.log(vm.reversedMessage) // => 'olleH'
```

…

Class 与 Style 绑定

操作元素的 class 列表和内联样式是数据绑定的一个常见需求。因为它们都是 attribute，所以我们可以用 v-bind 处理它们：只需要通过表达式计算出字符串结果即可。不过，字符串拼接麻烦且易错。因此，在将 v-bind 用于 class 和 style 时，Vue.js 做了专门的增强。表达式结果的类型除了字符串之外，还可以是对象或数组。

绑定 HTML Class

我们可以传给 v-bind:class 一个对象，以动态地切换 class：

```
<div v-bind:class="{ active: isActive }"></div>
```

上面的语法表示 active 这个 class 存在与否将取决于数据 property isActive 的 truthiness。

v-bind:class 指令也可以与普通的 class attribute 共存

对象语法

我们可以传给 v-bind:class 一个对象，以动态地切换 class:

```
<div v-bind:class="{ active: isActive }"></div>
```

我们可以把一个**数组**传给 v-bind:class，以应用一个 class 列表:

```
<div v-bind:class="[activeClass, errorClass]"></div>
```

绑定内联样式

v-bind:style 的对象语法十分直观——看着非常像 CSS，但其实是一个 JavaScript 对象。CSS property 名可以用驼峰式 (camelCase) 或短横线分隔 (kebab-case，记得用引号括起来) 来命名:

```
<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
```

条件渲染

v-if 指令用于条件性地渲染一块内容。这块内容只会在指令的表达式返回 truthy 值的时候被渲染

因为 v-if 是一个指令，所以必须将它添加到一个元素上。但是如果想切换多个元素呢？此时可以把一个 <template> 元素当做不可见的包裹元素，并在上面使用 v-if。最终的渲染结果将不包含 <template> 元素。

```
<template v-if="ok">
  <h1>Title</h1>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</template>
```

你可以使用 v-else 指令来表示 v-if 的“else 块”:

```
<div v-if="Math.random() > 0.5">
  Now you see me
</div>
<div v-else>
  Now you don't
</div>
```

v-else-if，顾名思义，充当 v-if 的“else-if 块”

用 key 管理可复用的元素

Vue 为你提供了一种方式来表达“这两个元素是完全独立的，不要复用它们”。只需添加一个具有唯一值的 key attribute 即可：

```
<template v-if="loginType === 'username'">
  <label>Username</label>
  <input placeholder="Enter your username" key="username-input">
</template>
<template v-else>
  <label>Email</label>
  <input placeholder="Enter your email address" key="email-input">
</template>
```

另一个用于根据条件展示元素的选项是 v-show 指令。用法大致一样：
不同的是带有 v-show 的元素始终会被渲染并保留在 DOM 中。v-show 只是简单地切换元素的 CSS property display。
注意，**v-show 不支持 <template> 元素，也不支持 v-else**。

v-if 是“真正”的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。
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——直到条件第一次变为真时，才会开始渲染条件块。

列表渲染

用 v-for 指令基于一个数组来渲染一个列表。v-for 指令需要使用 item in items 形式的特殊语法，其中 items 是源数据数组，而 item 则是被迭代的数组元素的别名。

可以提供第二个的参数为 property 名称（也就是键名）：

```
<div v-for="(value, name) in object">
  {{ name }}: {{ value }}
</div>
```

还可以用第三个参数作为索引：

```
<div v-for="(value, name, index) in object">
  {{ index }}. {{ name }}: {{ value }}
```

```
</div>
```

显示过滤/排序后的结果

有时，我们想要显示一个数组经过过滤或排序后的版本，而不实际变更或重置原始数据。在这种情况下，可以创建一个计算属性，来返回过滤或排序后的数组。

```
<li v-for="n in evenNumbers">{{ n }}</li>
data: {
  numbers: [ 1, 2, 3, 4, 5 ]
},
computed: {
  evenNumbers: function () {
    return this.numbers.filter(function (number) {
      return number % 2 === 0
    })
  }
}
```

在计算属性不适用的情况下,使用功能普通函数

v-for 也可以接受整数。在这种情况下，它会把模板重复对应次数。

```
<div>
  <span v-for="n in 10">{{ n }} </span>
</div>
```

事件处理

用 v-on 指令监听 DOM 事件，并在触发时运行一些 JavaScript 代码
v-on 还可以接收一个需要调用的方法名称。

除了直接绑定到一个方法，也可以在内联 JavaScript 语句中调用方法:

```
<button v-on:click="say('what')">Say what</button>
```

事件修饰符

在事件处理程序中调用 event.preventDefault() 或 event.stopPropagation() 是非常常见的需求。尽管我们可以在方法中轻松实现这点，但更好的方式是：方法只有纯粹的数据逻辑，而不是去处理 DOM 事件细节。

为了解决这个问题，Vue.js 为 v-on 提供了事件修饰符。之前提过，修饰符是由点开头的指令后缀来表示的。

1. .stop
2. .prevent
3. .capture
4. .self
5. .once
6. .passive

```
<!-- 阻止单击事件继续传播 -->
<a v-on:click.stop="doThis"></a>

<!-- 提交事件不再重载页面 -->
<form v-on:submit.prevent="onSubmit"></form>

<!-- 修饰符可以串联 -->
<a v-on:click.stop.prevent="doThat"></a>

<!-- 只有修饰符 -->
<form v-on:submit.prevent></form>

<!-- 添加事件监听器时使用事件捕获模式 -->
<!-- 即内部元素触发的事件先在此处理，然后才交由内部元素进行处理 -->
<div v-on:click.capture="doThis">...</div>

<!-- 只当在 event.target 是当前元素自身时触发处理函数 -->
<!-- 即事件不是从内部元素触发的 -->
<div v-on:click.self="doThat">...</div>
```

按键修饰符

在监听键盘事件时，我们经常需要检查详细的按键。Vue 允许为 v-on 在监听键盘事件时添加按键修饰符：

```
<!-- 只有在 `key` 是 `Enter` 时调用 `vm.submit()` -->
<input v-on:keyup.enter="submit">
```

你可以直接将 KeyboardEvent.key 暴露的任意有效按键名转换为 kebab-case 来作为修饰符。

```
<input v-on:keyup.page-down="onPageDown">
```

按键码

keyCode 的事件用法已经被废弃了并可能不会被最新的浏览器支持。

使用 keyCode attribute 也是允许的：

```
<input v-on:keyup.13="submit">
```

为了在必要的情况下支持旧浏览器，Vue 提供了绝大多数常用的按键码的别名：

1. .enter
2. .tab
3. .delete (捕获“删除”和“退格”键)
4. .esc
5. .space
6. .up
7. .down
8. .left
9. .right

系统修饰键

可以用如下修饰符来实现仅在按下相应按键时才触发鼠标或键盘事件的监听器。

1. .ctrl
2. .alt
3. .shift
4. .meta

注意：在 Mac 系统键盘上，meta 对应 command 键 (⌘)。在 Windows 系统键盘 meta 对应 Windows 徽标键 (⊞)。在 Sun 操作系统键盘上，meta 对应实心宝石键 (◆)。在其他特定键盘上，尤其在 MIT 和 Lisp 机器的键盘、以及其后继产品，比如 Knight 键盘、space-cadet 键盘，meta 被标记为“META”。在 Symbolics 键盘上，meta 被标记为“META”或者“Meta”。

鼠标按钮修饰符

1. **.left**
2. **.right**
3. **.middle**

这些修饰符会限制处理函数仅响应特定的鼠标按钮。

表单输入绑定

你可以用 v-model 指令在表单 <input>、<textarea> 及 <select> 元素上创建双向数据绑定。它会根据控件类型自动选取正确的方法来更新元素。尽管有些神奇，但 **v-model 本质上不过是语法糖**。它负责监听用户的输入事件以更新数据，并对一些极端场景进行一些特殊处理。

v-model 会忽略所有表单元素的 value、checked、selected attribute 的初始值而总是将 Vue 实例的数据作为数据来源。你应该通过 JavaScript 在组件的 data 选项中声明初始值。

v-model 在内部为不同的输入元素使用不同的 property 并抛出不同的事件：

1. text 和 textarea 元素使用 value property 和 input 事件；
2. checkbox 和 radio 使用 checked property 和 change 事件；
3. select 字段将 value 作为 prop 并将 change 作为事件。

复选框

单个复选框，绑定到布尔值：

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

多个复选框，绑定到同一个数组：

```
<input type="checkbox" id="jack" value="Jack" v-model="checkedNames">
<label for="jack">Jack</label>
<input type="checkbox" id="john" value="John" v-model="checkedNames">
<label for="john">John</label>
<input type="checkbox" id="mike" value="Mike" v-model="checkedNames">
<label for="mike">Mike</label>
<br>
<span>Checked names: {{ checkedNames }}</span>
```

单选按钮

```
<div id="example-4">
  <input type="radio" id="one" value="One" v-model="picked">
  <label for="one">One</label>
  <br>
  <input type="radio" id="two" value="Two" v-model="picked">
  <label for="two">Two</label>
  <br>
  <span>Picked: {{ picked }}</span>
</div>
```

修饰符

组件基础

组件是可复用的 Vue 实例，且带有一个名字

```
// 定义一个名为 button-counter 的新组件
Vue.component('button-counter', {
  data: function () {
    return {
      count: 0
    }
  },
  template: '<button v-on:click="count++">You clicked me {{ count }}
times.</button>'
})

<div id="components-demo">
  <button-counter></button-counter>
</div>
```

data 必须是一个函数

通过 Prop 向子组件传递数据

Prop 是你可以**在组件上注册的一些自定义 attribute**。
当一个值传递给一个 prop attribute 的时候，它就变成了那个组件实例的一个 property。

```
Vue.component('blog-post', {
  props: ['title'],
  template: '<h3>{{ title }}</h3>'
})
```

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。

一个 prop 被注册之后，你就可以像这样把数据作为一个自定义 attribute 传递进来：

```
<blog-post title="My journey with Vue"></blog-post>
<blog-post title="Blogging with Vue"></blog-post>
```

```
<blog-post title="Why Vue is so fun"></blog-post>
```

组件注册

https://cn.vuejs.org/v2/guide/components-registration.html

路由

对于大多数单页面应用，都推荐使用官方支持的 vue-router 库

其他

router-view

在 app.vue 这个页面里面存在一个<router-view>的 tag 标签。通过它，我们点击链接，即可显示出 Vue 的 HelloWorld.vue 页面

路由，其实就是指向的意思，当我点击页面上的 home 按钮时，页面中就要显示 home 的内容，如果点击页面上的 about 按钮，页面中就要显示 about 的内容

路由中有三个基本的概念 route，routes，router。

1，route，它是一条路由，由这个英文单词也可以看出来，它是单数， Home 按钮 => home 内容， 这是一条 route，about 按钮 => about 内容， 这是另一条路由。
2，routes 是一组路由，把上面的每一条路由组合起来，形成一个数组。[{home 按钮 =>home 内容 }， { about 按钮 => about 内容}]
3，router 是一个机制，相当于一个管理者，它来管理路由。因为 routes 只是定义了一组路由，它放在哪里是静止的，当真正来了请求，怎么办？ 就是当用户点击 home 按钮的时候，怎么办？这时 router 就起作用了，它到 routes 中去查找，去找到对应的 home 内容，所以页面中就显示了 home 内容。

4, 客户端中的路由，实际上就是 dom 元素的显示和隐藏。当页面中显示 home 内容的时候，about 中的内容全部隐藏，反之也是一样。客户端路由有两种实现方式：基于 hash 和基于 html5 history api.

vue-router 中的路由也是基于上面的内容来实现的

```
export default new Router({
    routes:[
        {
            path:'/result',
            name:'result', //使用 params 的传参方式，path 和 name 都必须呀在这里声明
            component:Result
        },
```

```
new Vue({
  router,
  render: h => h(App),
}).$mount('#app');
```

Vue 开发积累

Body 全屏

在 inde.html 中<head>添加样式：

```
    <style>
      /*
      body 占满全屏
      */
      html, body{
        height: 100%;
        width: 100%;
        border:hidden;
        overflow:hidden;
        margin: 0;
        padding: 0;
      }
```

```
    </style>
```