

vue进阶篇

(陈华旺-chenhuawang@itany.com)

目录 [vue进阶篇]

- 1、生命周期
- 2、自定义指令
 - 2.1、全局指令定义
 - 2.2、局部指令定义
- 3、组件化
 - 3.1、构造器继承函数
 - options配置中的el属性
 - options配置中的data属性
 - 3.2、全局组件定义
 - 3.3、局部组件定义
 - 3.5、组件数据传递和共享
 - 3.5.1、父组件向子组件数据传递
 - 3.5.2、子组件向父组件数据传递
 - 3.5.3、非父子组件数据传递
 - 1、借助共有顶级组件
 - 2、中央事件总线 (Event Bus)
 - 3.5.4、组件的生命周期执行顺序
 - 3.5.5、单向数据流&组件双向数据共享
 - 1、单向数据流
 - 2、计算属性的双向共享操作
 - 3、v-bind指令的 `sync` 修饰符的双向共享操作
 - 4、JS引用类型的双向共享操作
 - 5、独立数据监视对象的双向共享操作
- 4、内置全局组件
 - 4.1、动态组件
 - 4.2、过渡组件
 - 4.3、组件缓存
 - 4.4、组件分发

1、生命周期

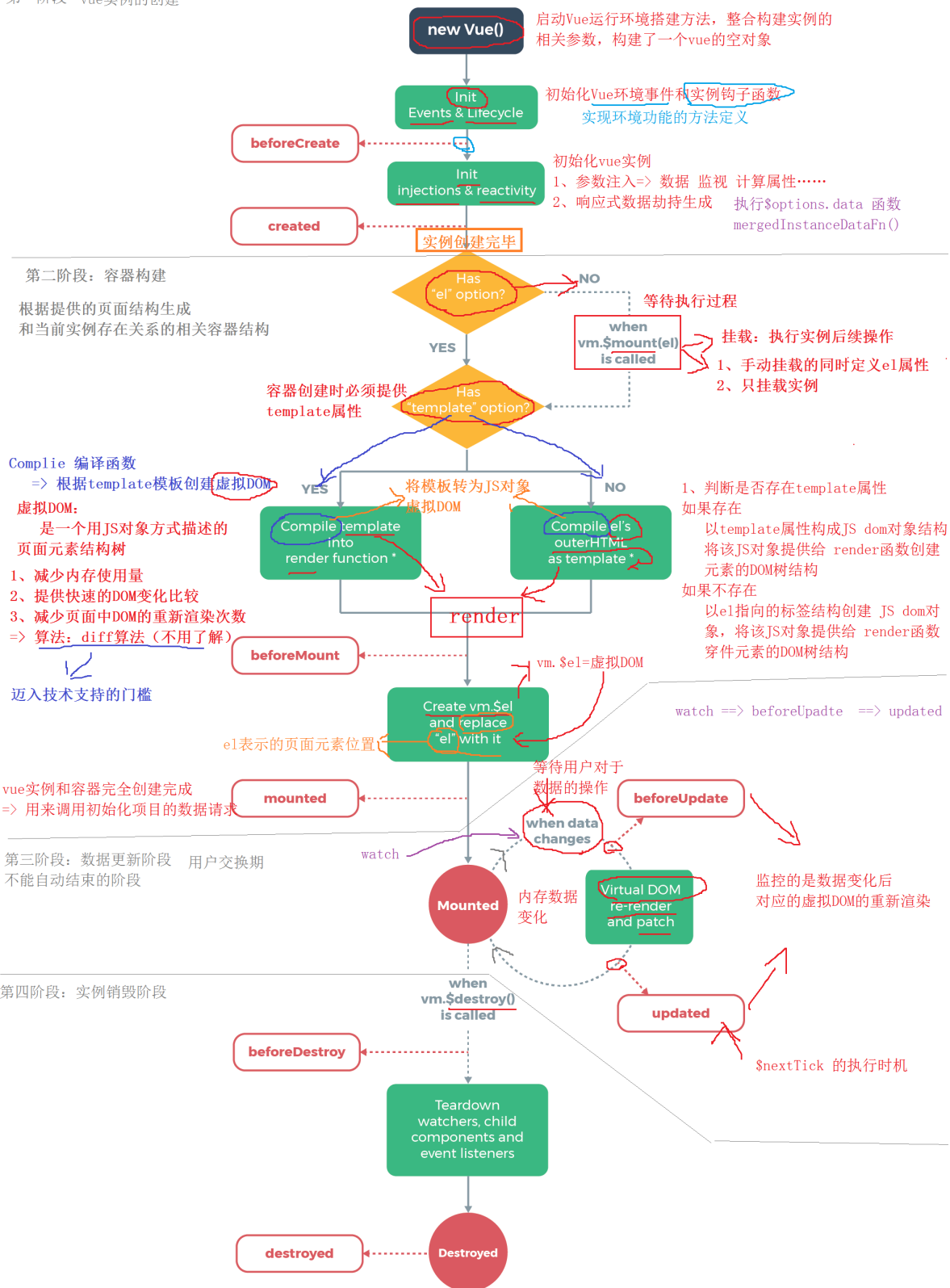
- 生命周期：**一段代码从 创建 到 销毁的 完成过程**
- 钩子函数: **Vue对外提供的在一些特点代码执行段上的回调函数**

```

1  new Vue({
2    // 在实例初始化之后，数据观测 和 事件配置之前被调用。
3    beforeCreate:function(){},
4    // 在实例创建完成后被立即调用
5    created:function(){},
6    // 在挂载开始之前被调用：相关的 render 函数首次被调用。
7    beforeMount:function(){},
8    // el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子
9    mounted:function(){},
10   // 数据更新时调用，发生在虚拟 DOM 打补丁之前。这里适合在更新之前访问现有的 DOM
11   beforeUpdate:function(){},
12   // 由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。
13   updated:function(){},
14   // 实例销毁之前调用。在这一步，实例仍然完全可用。
15   beforeDestroy:function(){},
16   // Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听
   器会被移除
17   destroyed:function(){},
18 })

```

- 生命周期流程



2、自定义指令

- 除了核心功能默认内置的指令，Vue 也允许注册自定义指令。
- 功能：**对页面展示效果功能的扩展==> 自定义指令 主要完成的是 页面DOM元素的操作**
 - MVVM 设计模式核心思想：简化开发者对于DOM的操作，vue基本实现了相关操作，基本上不需要完成DOM的操作
 - vue简化开发者对于DOM的操作，实际上就是 将DOM的操作封装成插值表达式或者指令**
- 自定义指令是vue提供给开发者 对DOM操作的接口（规范化的方法）

2.1、全局指令定义

- 范围：可以在 vue 的 任意对象的容器中使用
- 用法：`v-指令名[:参数][.修饰符].....=取值` 定义在标签属性上
- 语法：`Vue.directive(id,definition)`
 - `id=name`: 定义指令名称
 - `definition`: 指令的处理函数, `function(el, binding, newNode,oldVnode){}`
 - `el`: 调用指令的当前DOM对象
 - `binding`: 是一个对象, 对象中包含了指令构成**参数、修饰符、取值**
 - `name`: **指令名称**
 - `rawName`: **指令调用时的表达式**
 - `expression`: **指令取值, = 右侧的取值表达式**
 - `arg`: **指令参数, : 后定义的参数名称**
 - `oldArg`: **是指令 : 后定义的,被修改后的旧的参数名称**
 - `value`: **指令 = 右侧 表达式的计算结果**
 - `oldValue`: **指令绑定变量发生变化后, 调用该方法时, 存储的 上次的结果值**
 - `modifiers`: **指令修饰符, . 后定义的修饰符名称**
 - **modifiers 为 Object{ key:value } 类型数据**
 - **key 为修饰符名称, value取值 为 true**
 - `true`作为 `value`表示当前指令被启用
 - 对象中没有修饰符 `key`, 表示指令不启用
 - 当没有修饰符时, 表示该对象为空对象
 - `newVnode`: 指令更新后的新虚拟DOM
 - `oldVnode`: 指令更新前的旧虚拟DOM
 - 定义：**全局指令定义必须在使用之前**

```
1 Vue.directive("lazy",function( el, binding, newNode,oldVnode ){
2
3 })
```

- 特性：
 - **自定指令的取值、参数、修饰符, 任意一个发生变化, 都会重新执行指令方法, 完成响应式特性**
 - 如果在操作过程中, 指令的值未发生变化, 指令方法不会重复执行

2.2、局部指令定义

- 范围：仅限于定义时关联的 vue 实例的容器中使用
- 定义：

```
1 new Vue({
2   directives:{ key[string]:value[Fuction] }
3 })
```

- `key (string)`: 指令名称, 定义完成后, 页面使用需 `v-` 前缀调用指令

- 如果名称为驼峰命名方式（例如 imgLazy），页面使用时需要转换为 连字符（v-img-lazy）
 - value (Function) : `function(el, binding, newVnode,oldVnode){}` 指令执行函数
 - 参数：参考全局指令
- 使用：<标签 v-自定义指令名[:参数][.修饰符.修饰符.....][=取值] ></标签>

3、组件化

- 组件是可复用的 Vue 实例，且带有一个名字，项目中把组件作为自定义元素来使用
- 组件构成：**将项目中可能被重复使用的页面结构，转换为vue的一个组件实例，以组件组合的方式构成整个项目结构**，或将特殊实现功能封装成组件进行特殊调用



3.1、构造器继承函数

- 构造器继承函数 `Vue.extend()`，通过以**基础 Vue 构造器为蓝本**，创建一个具有**特殊配置**的“子类”构造器。
- 被创建的新构造器可以用于构建Vue实例，但被构建的实例具有预先定义的相关特性
- 项目中可通过该功能构建具有特殊用途的Vue的实例
- 构造器继承函数语法：`Vue.extend(options)`
 - 返回一个新的具有Vue基本功能和特殊配置的新构造函数
 - options：该参数用于定义新构造器的相关特性，**该配置项等同于 `new Vue(options)` 的 options参数**



- 示例：

```

1 // 创建构造器
2 var Profile = Vue.extend({
3   template: '<p @click="printMsg()">这是一个特殊的Vue实例</p>',
4   methods: {
5     printMsg(){
6       console.log("自定义组件构造器创建的组件")
7     }
8   }
9 })
10 // 创建 Profile 实例，并挂载到一个元素上。
11 new Profile().$mount('#mount-point');
12 // 创建 Profile 实例，并执行空挂载
13 new Profile().$mount();

```

```

1 <head>
2   <script src="./js/vue.js"></script>
3 </head>
4 <body>
5   <div id="c1"></div>
6   <hr>
7   <div id="c2"></div>
8   <hr>
9   <div id="c3"></div>
10 </body>
11 <script>
12   // 构造器继承函数 Vue.extend(options)
13   // 完成配置继承
14   // options 配置项类似于 new Vue(options) 的 options
15   //           构造继承函数配置项目中，存在两个特殊配置区别于 new Vue 配置项
16   //           e1 不能定义与 extend继承函数
17   //           data
18   // extend 方法存在一个固定的返回值，返回值是带有相关配置的 新的vue实例构造函数
19   var NewVueFun = Vue.extend({
20     // e1:"c1",
21     template: "<h1 @click='printMsg()'>测试标签{{ msg }}</h1>",
22     // JS 的 堆栈
23     data:function(){
24       returns
25     },
26     methods: {
27       printMsg() {
28         console.log("printMsg方法");
29       },
30     },
31     beforeCreate() {
32       console.log("beforeCreate");
33     },
34     created() {
35       console.log("created");
36     },
37     mounted() {
38       console.log("mounted");
39     }
40   });
41   // console.log(NewVueFun);
42   // 通过extend继承返回的新构造器在创建对象时，不能接受参数

```

```

43 // 该对象创建生命周期会停止在 mount 挂载行为上 = 得到一个具有$options配置项的基础对
    象
44 var obj1 = new NewVueFun();
45 var obj2 = new NewVueFun();
46 var obj3 = new NewVueFun();
47 obj1.$mount("#c1");
48 obj2.$mount("#c2");
49 obj3.$mount("#c3");
50 </script>

```

options配置中的el属性

- **限制**：只在由 `new Vue()` 创建的实例中使用。
- **详细**：
 - 提供一个在页面上已存在的 DOM 元素作为 Vue 实例的挂载目标。可以是 CSS 选择器，也可以是一个 HTMLElement 实例。
 - 在实例挂载之后，元素可以用 `vm.$el` 访问。
 - 如果在实例化时存在这个选项，实例将立即进入编译过程，否则，需要显式调用 `vm.$mount()` 手动开启编译。

options配置中的data属性

- **类型**：Object | Function
- **限制**：只有在 `new Vue()` 中取值为 Object，**组件或构造器定义只接受 function**。
- **详细**：Vue 实例的数据对象。
 - Vue 将会递归将 data 的属性转换为 getter/setter，从而让 data 的属性能够响应数据变化。
 - 当一个**组件或构造器**被定义，`data` 必须声明为返回一个初始数据对象的函数
 - 因为组件可能被用来创建多个实例。如果 `data` 仍然是一个纯粹的对象，则所有的实例将**共享引用**同一个数据对象！
 - 通过提供 `data` 函数，每次创建一个新实例后，我们能够调用 `data` 函数，从而返回初始数据的一个全新副本数据对象。

40- 组件构造器继承函数.html 41- 继承函数中的data.html

```

6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <title>Document</title>
8   </head>
9   <script>
10    var Vue = function(options){
11      this.data = options.data;
12    }
13    var vm1 = new Vue({
14      data:{
15        msg:"数据"
16      }
17    });
18    var vm2 = new Vue({
19      data:{
20        msg:"数据"
21      }
22    });
23
24    var extend = function(options){
25      return function(){
26        this.data = options.data;
27      }
28    }
29    var Fun = extend({
30      data:{
31        msg:"数据"
32      }
33    });
34    // console.log(Fun);
35    var vm3 = new Fun();
36    var vm4 = new Fun();
37  </script>
38  <body>
39  </body>
40  </html>

```

Diagram illustrating the relationship between Vue instances and the extend function:

- Vue Constructor:** Takes options and sets `this.data = options.data`. Instances `vm1` and `vm2` are created with `msg: "数据"`.
- extend Function:** Returns a function that sets `this.data = options.data`. An instance `Fun` is created with `msg: "数据"`.
- Instances:**
 - `vm1` and `vm2` are instances of `Vue`.
 - `vm3` and `vm4` are instances of `Fun`.
- Console Output:**
 - `vm1`: `{data: {msg: "数据"}, __proto__: Object}`
 - `vm2`: `{data: {msg: "数据"}, __proto__: Object}`
 - `vm1===vm2`: `false`
 - `vm1.data.msg = "aaaa"`: `"aaaa"`
 - `vm1`: `{data: {msg: "aaaa"}, __proto__: Object}`
 - `vm2`: `{data: {msg: "数据"}, __proto__: Object}`
 - `vm3`: `{data: {msg: "数据"}, __proto__: Object}`
 - `vm4`: `{data: {msg: "数据"}, __proto__: Object}`
 - `vm3.data.msg = "aaaa"`: `"aaaa"`
 - `vm3`: `{data: {msg: "aaaa"}, __proto__: Object}`
 - `vm4`: `{data: {msg: "aaaa"}, __proto__: Object}`
 - `vm3===vm4`: `false`
 - `vm5`: `{data: {msg: "数据"}, __proto__: Object}`
 - `vm6`: `{data: {msg: "数据"}, __proto__: Object}`
 - `vm5===vm6`: `false`
 - `vm5.data.msg = "aaaa"`: `"aaaa"`
 - `vm5`: `{data: {msg: "aaaa"}, __proto__: Object}`
 - `vm6`: `{data: {msg: "数据"}, __proto__: Object}`

Diagram illustrating the relationship between FunA and its instances:

- FunA Constructor:** Takes options and sets `this.data = options.data()`. Instance `FunA` is created with `msg: "数据"`.
- Instances:**
 - `vm5` and `vm6` are instances of `FunA`.
- Console Output:**
 - `vm5`: `{data: {msg: "数据"}, __proto__: Object}`
 - `vm6`: `{data: {msg: "数据"}, __proto__: Object}`
 - `vm5===vm6`: `false`
 - `vm5.data.msg = "aaaa"`: `"aaaa"`
 - `vm5`: `{data: {msg: "aaaa"}, __proto__: Object}`
 - `vm6`: `{data: {msg: "数据"}, __proto__: Object}`

● 实际应用-页面功能插件定义

```

1   <head>
2     <script src="./js/vue.js"></script>
3     <style>
4       .tip {
5         position: fixed;
6         top: 0px;
7         left: 50%;
8         transform: translate(-50%, -100%);
9         opacity: 0;
10        background-color: rgba(0, 0, 0, 0.6);
11        color: white;
12        padding: 5px 30px;
13        border-radius: 6px;
14        max-width: 60%;

```



```

15     white-space: pre-wrap;
16     word-break: break-word;
17     box-sizing: border-box;
18     animation-name: tipFade;
19     animation-iteration-count: 1;
20     /* animation-duration: 3s; */
21 }
22
23 @keyframes tipFade {
24     0% {
25         transform: translate(-50%, -100%);
26         opacity: 0;
27     }
28
29     16% {
30         transform: translate(-50%, 40px);
31         opacity: 1;
32     }
33
34     84% {
35         transform: translate(-50%, 40px);
36         opacity: 1;
37     }
38
39     100% {
40         transform: translate(-50%, -100%);
41         opacity: 0;
42     }
43 }
44 </style>
45 <script>
46     // 基于vue开发的插件功能-可以在任意的场景下使用
47     var Tip = Vue.extend({
48         template: "<div class='tip' :style='\"animation-
49 duration:\"+time+\"ms;\" ' >{{ msg }}</div>",
50         data() {
51             return {
52                 msg: "",
53                 time:3000
54             }
55         },
56         mounted() {
57             document.body.appendChild(this.$el);
58             console.log(this.time);
59             setTimeout(() => {
60                 document.body.removeChild(this.$el);
61                 this.$destroy();
62             }, this.time)
63         },
64     });
65
66     function tipBox(msg = "默认值",time=3000) {
67         let tip = new Tip();
68         tip.msg = msg;
69         tip.time = time;
70         tip.$mount();
71         // document.body.appendChild(tip.$el);
72         // setTimeout(() => {

```

```

72         //      document.body.removeChild(tip.$el);
73         //      tip.$destroy();
74         // }, 3000)
75     }
76 </script>
77 </head>
78
79 <body>
80     <input type="button" value="打开一个基于VUE语法开发的提示窗" onclick="tipBox('数
据加载成功')">
81     <input type="button" value="打开一个基于VUE语法开发的提示窗" onclick="tipBox('评
论添加成功',5000)">
82     <input type="button" value="打开一个基于VUE语法开发的提示窗" onclick="tipBox('操
作成功')">
83 </body>

```

3.2、全局组件定义

- 全局组件的定义，依赖于 Vue 构造函数提供的全局组件定义函数 `Vue.component()`
- 定义的全局组件，可以在项目的 任意位置进行使用
- **语法构成**：Vue.component(id, [definition])
- **参数**：
 - id (string)：定义组件唯一名称，在页面中作为组件调用的标签名
 - definition (Function | Object)：描述组件的相关配置特性
- **用法**：

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

```

1 // 注册组件，传入一个扩展过的构造器
2 Vue.component('my-component', Vue.extend({ /* ... */ }))
3
4 // 注册组件，传入一个选项对象（自动调用 Vue.extend）
5 Vue.component('my-component', { /* ... */ })

```

```

1 <!-- 以组件名称作为自定义标签名，在页面中调用启动组件生命周期 -->
2 <my-component></my-component>

```

```

1 <head>
2     <script src="./js/vue.js"></script>
3 </head>
4 <body>
5     <!-- <h1>aaaa</h1>
6     <H1>AAAA</H1> -->
7     <div id="app">
8         <p @click="print()">new Vue的 root组件:{{ msg }}</p>
9         <!-- <hr>
10        <compa></compa>
11        <hr>
12        <compa></compa>
13        <hr>
14        <compa></compa> -->
15
16        <hr>
17        <div-comp></div-comp>

```

```

18     <hr>
19     <!-- <hr>
20     <div-comp></div-comp>
21     <hr>
22     <hr>
23     <div-comp></div-comp>
24     <hr> -->
25     <hr>
26     <hello-word></hello-word>
27 </div>
28 <!-- <hello-word></hello-word> -->
29 <template id="helloWord">
30     <div>
31         hellWord组件: {{ info }}
32         <hr>
33         <div-comp></div-comp>
34         <!-- <hello-word></hello-word> -->
35     </div>
36 </template>
37 </body>
38 <script>
39     // 全局组件定义 依赖 Vue构造器 提供的 component 方法完成构建
40     // Vue.component(id,definition)
41     //     id( tagName ) : 描述组件的唯一名称,同时也是vue容器中自定义的 新 标签名称
42     //     例 Vue.component("aaaa",.....) ==> 组件名称为aaaa ==> vue容器中可以使
43     //     用一个新标签 <aaaa>
44     //     如果组件名称使用的是驼峰方式定义的名称, 标签名称将转为连字符
45     //     例 Vue.component("helloWord",.....) ==> 组件名称为helloWord ==> vue容
46     //     器中可以使用一个新标签 <hello-word>
47     //     definition : 描述组件vue实例的相关配置和功能
48     //     definition: Function==>通过 Vue.extend 继承后返回的带有模板的vue
49     //     实例构造器函数
50     //     definition: Object对象 ==> 该对象实际上就是 Vue.extend(options)
51     //     的配置参数
52     //     Vue.component() 隐式调用 Vue.extend 方法
53     // Vue.component 完成的是自定义标签和实例构造函数的关联关系
54     //     ==> vue容器中 每次的 组件标签定义, 就相当于 组件构造函数的 new
55     // 1、构造器定义
56     var CompA = Vue.extend({
57         template: '<p @click=print()>compA:{{ msg }}</p>',
58         data() {
59             return {
60                 msg: "测试数据"
61             }
62         },
63         methods: {
64             print(){
65                 console.log("compA-print");
66             }
67         },
68         beforeCreate() {
69             console.log("beforeCreate");
70         },
71         mounted() {
72             console.log("compA-mounted");
73         },

```

```

72     });
73     // Vue.component("compa",CompA); // 定义一个全局组件 compA
74     Vue.component("divComp",CompA); // 定义一个全局组件 div
75
76     // 2、字面量构建
77     Vue.component("HelloWord",{
78         template:"#helloWord",
79         data(){
80             return {
81                 info:"hello-wrod 组件"
82             }
83         }
84     })
85     // 根组件==root组件
86     new Vue({
87         el:"#app",
88         data:{
89             msg:"root数据"
90         },
91         methods: {
92             print(){
93                 console.log("new Vue-print");
94             }
95         },
96     });
97 </script>

```

Tips：忽略自定义标签名

通过全局配置 `Vue.config.ignoredElements` 设置需要被忽略的自定义标签

- 类型： `Array<string | RegExp>`
- 默认值： `[]`
- 用法：

```

1  Vue.config.ignoredElements = [
2    'my-custom-web-component',
3    'another-web-component',
4    // 用一个 `RegExp` 忽略所有“ion-”开头的元素
5    // 仅在 2.5+ 支持
6    /^ion-/
7  ]

```

须使 Vue 忽略在 Vue 之外的自定义元素 (e.g. 使用了 Web Components APIs)。否则，它会假设你忘记注册全局组件或者拼错了组件名称，从而抛出一个关于 `Unknown custom element` 的警告。

- 特性：
 - **1、组件定义完成后，不能在页面独立使用和存在；**
 - **2、组件使用需要依赖一个当前页面或者项目中已经存在的vue实例容器；**

3.3、局部组件定义

- 局部组件构建**依赖于一个固定的 vue 实例**，通过**配置实例属性** `components:{}` 定义
- 局部组件仅限构建组件的 vue实例的容器范围中使用
- **语法构成**： `components:{ id:definition,..... }`

- **参数：**
 - id (string)：定义组件唯一名称，在页面中作为组件调用的标签名；
 - **当前全局组件名称和局部组件名称相同时，局部组件具有更高的优先级**
 - definition (Object | Function)：描述组件的相关配置特性

- **用法：**

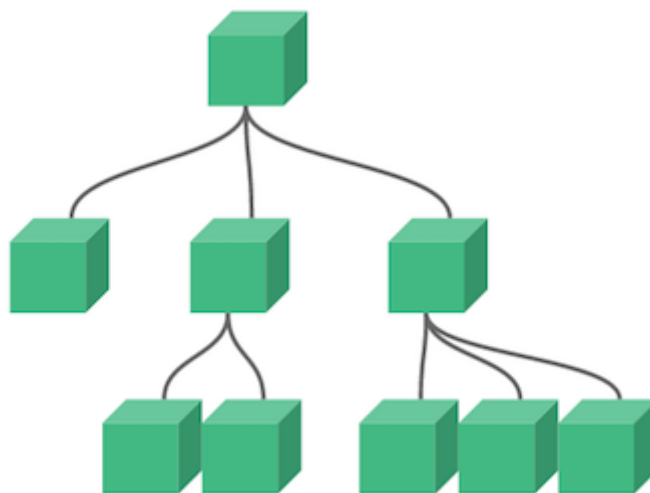
注册或获取全局组件。注册还会自动使用给定的 `id` 设置组件的名称

```
1 new Vue({
2   components:{
3     "component-a":Vue.extend({ /* ... */ }),
4     "component-b":{ /* ... */ }
5   }
6 })
```

```
1 <!-- 以组件名称作为自定义标签名，在页面中调用启动组件生命周期 -->
2 <component-a></component-a>
3 <component-b></component-b>
```

3.5、组件数据传递和共享

- 页面组件关系结构，**类似于HTML的DOM树结构，存在父子关系**，因此组件间的数据传递共享存在**父=>子、子=>父和非父子关系组件数据传递**



3.5.1、父组件向子组件数据传递

- 技术实现：**属性绑定，数据拦截**
- 详细描述：**以属性绑定的方式将传递的数据绑定在子组件标签上，在子组件对象中以属性 `props` 方式进行 绑定属性的拦截**

```
1 <body>
2   <div id="app">
3     <p>父组件</p>
4     <p>msg:{{ msg }}</p>
5     <input type="text" v-model="msg">
6     <p>num:{{ num }}</p>
7     <hr>
8     <!--
9       技术：属性绑定+数据拦截
```

```

10
11         属性绑定：采用VUE的 v-bind 指令 为 子组件标签绑定一个自定义属性，取值为父组
    件变量
12         vue 会将子组件标签上绑定的属性，直接传递给子组件的根标签
13         vue 的子组件 会直接将非DOM属性的变量，装载进子组件的 实例属性
    $attrs
14         属性绑定和$attrs，本身具有响应特性
15         $attrs 会保留父组件传递的数据类型
16         ==> 在组件生命周期的编译过程直接获取的父组件的相关数据赋值给
    $attrs (JS环境)
17         ==> 在构成页面结构时，将$attrs 的属性数据写入到 页面标签上
18
19         -->
20         <comp-a v-bind:aaaa=" msg " :num=" num "></comp-a>
21         <hr>
22         <!--
23         数据拦截：在子组件中以 props 属性定义需要被拦截的，子组件标签上的自定义属性
24         props中的拦截变量具有 数据仓库的 取值行为功能
25
26         -->
27         <comp-b :msg=" msg " :num=" num "></comp-b>
28     </div>
29
30     <template id="compA">
31         <div itany="网博">
32             <p>子组件</p>
33             <p>获取父组件的数据MSG:{{ getMsg }}</p>
34             <p>获取父组件的数据num:{{ num }}</p>
35         </div>
36     </template>
37     <template id="compB">
38         <div itany="网博">
39             <p>子组件</p>
40             <p>获取父组件的数据MSG:{{ msg }}</p>
41             <p>获取父组件的数据num:{{ num }}</p>
42         </div>
43     </template>
44 </body>
45 <script>
46     new Vue({
47         el: "#app",
48         data: {
49             msg: "父组件的MSG数据",
50             num: 100
51         },
52         components: {
53             CompA: {
54                 template: "#compA",
55                 computed: {
56                     getMsg() {
57                         return this.$attrs.aaaa;
58                     },
59                     num() {
60                         return this.$attrs.num;
61                     }
62                 },
63                 beforeMount() {
64                     console.log(this.$el);

```

```

65     },
66   },
67   CompB:{
68     template: "#compB",
69     // 认为是一个数据仓库->拦截仓库-拦截子组件标签上绑定的自定义属性
70     props:["msg","num"]
71   }
72 }
73 })
74 </script>

```

- 属性绑定使用vue `v-bind` 语法，数据拦截使用 `props` 属性
 - `props` : 拦截父组件传递的数据变量，取值 **Array|Object**
 - 取值 Array:数组中使用字符串描述，子组件标签上绑定的对应属性，获取属性值
 - 取值Object:使用props属性的**数据校验**功能，完成更加严格的数据获取
 - 数据校验：**针对vue框架的使用者，为开发者提示数据传递的正确与否**

```

1  var CompA = Vue.extend({
2    template: "#compA",
3    // props:["msg"]
4    // props的对象定义
5    //   对象的key: 需要拦截的子组件标签上绑定的自定义属性名
6    //   对象的value :
7    //       1、取JS数据类型的包装器 => 限制该属性的值为特点数据类型
8    //       2、取值Object => key:功能限制关键字    value 功能描述
9    //       key:功能限制关键字 => vue 语法定义完成的固定校验描述关
   键字
10   //       -type: 类型限制
11   //       -default: 当前组件没有绑定对应属性或该属性是
   undefined,
12   //       提供默认值支持
13   //       + 非Object|Array类型，可以直接赋值默认值
14   //       + Object|Array类型，需要通过方法 返回 默认值
15   //       -required:限制属性的传递状态 默认值false
16   //       + false可传递也可不传递
17   //       + true属性必须绑定必须传递
18   //       -validator: 取值为Function,提供开发者自定义校验规
   则
19   props: {
20     // key:value
21     msg: String,
22     num: Number,
23     info: {
24       type: String,
25       default: "默认值", // 当前组件没有绑定info属性，提供默认值支持
26     },
27     user: {
28       type: Object,
29       // default:{
30       //   name:"默认名字"
31       // }
32       default: function () {
33         return {
34           name: "默认名字"
35         }
36       }
37     }
38   }
39 })

```

```

36     }
37   },
38   arg:{
39     required:true,
40     // 当传递的属性值为 undefined default 生效
41     default:"默认值"
42   },
43   number:{
44     type:Number,
45     validator(value){
46       // value为属性绑定的数据
47       // return value>=0&&value<=100;
48       // 返回校验状态 , true 表示校验通过, false 校验不通过
49       var flag = value>=0&&value<=100;
50       if(!flag){
51         // alert("值只能在 0~100"); // 可以给用户提示
52       }
53       return true;
54     }
55   }
56 }
57 });

```

Tips：子组件属性绑定特性的特殊使用

- 样式传递：通过属性绑定的传递特性，不进行数据拦截，完成样式有父组件范围传递到子组件根标签上

3.5.2、子组件向父组件数据传递

- 技术实现：**事件绑定，事件触发**
- 详细描述：**在子组件标签上以事件绑定的方式调用父组件方法，在子组件对应的执行区通过 \$emit 实例属性进行自定义事件触发，并传递参数**

```

1  <body>
2    <AAAAA onclick="show()">HTML的自定义标签</AAAAA>
3    <hr>
4    <div id="app">
5      <p>父组件</p>
6      <p>子组件MSG:{{ msg }}</p>
7    <hr>
8    <!--
9      技术：事件绑定，事件触发
10     事件绑定：使用vue的事件绑定语法v-on，为子组件标签绑定处理函数
11              v-on=>为指定标签添加对应的事件监听
12              v-on在组件标签上进行事件绑定时，在子组件创建实例时，将该事件直
13 接作为
14              当前子组件的一个事件环境（运行环境），定义在当前组件的事件库中
15              ( _events )
16              vm._events = { key在组件标签上绑定的事件名: value绑定的
17 事件数组 }
18              ==> 如果绑定的事件用于数据传递，子组件标签上绑定的事件不能定义
19              ( )
20              为子组件事件触发提供 参数传递的自定义构成方式
21 事件触发:使用的是vue 提供的 实例方法 $emit()
22              vm.$emit() 只用于触发对应vue实例，_events 中定义事件
23              vm.$emit(eventName,data) eventName 实例_events中自定义事件名

```


data 为触发事件是要传递的参数

```
20
21      -->
22      <input type="button" value="普通标签" @click=" loadMsg() ">
23      <comp-a v-on:click="loadMsg"></comp-a>
24      <!--
25          _events: {
26              click:[function invoker(){
27                  loadMsg();
28                  loadMsg2();
29              }]
30          }
31      -->
32  </div>
33
34  <template id="compA">
35      <div itany="网博">
36          <p>子组件</p>
37          <p>子组件MSG: {{ msg }}</p>
38          <input type="button" value="发送数据" @click=" sendMsg() ">
39      </div>
40  </template>
41 </body>
42 <script>
43     var CompA = Vue.extend({
44         template: "#compA",
45         data(){
46             return {
47                 msg:"子组件的msg变量"
48             }
49         },
50         // beforeCreate() {
51         //     console.log(this);
52         //     console.log(this._events.click[0]());
53         // },
54         methods:{
55             sendMsg(){
56                 console.log("子组件sendMsg==>");
57                 console.log(this.msg);
58                 // this._events.click[0]( this.msg );
59                 this.$emit("click",this.msg);
60                 console.log("<==子组件sendMsg");
61             }
62         }
63     })
64
65     new Vue({
66         el: "#app",
67         data:{
68             msg:""
69         },
70         components: {
71             CompA
72         },
73         methods: {
74             loadMsg( arg ){
75                 console.log("父组件loadMsg==>");
76                 console.log(arg);
77                 this.msg = arg;
```

```

78         console.log("<==父组件的loadMsg");
79     },
80     loadMsg2(){
81         console.log("父组件的loadMsg2");
82     }
83 },
84 })
85 </script>

```

• 注意事项：

- 1、用于数据传递的子组件事件绑定，不要使用 **标签定义的已知事件**，使用自定义事件名称，在绑定自定义事件时，事件名 **不要使用驼峰方式**
- 2、\$emit 完成子组件到父组件的数据传递，**不具有响应式功能**，**子组件数据变化只会影响子组件本身，不会主动触发\$emit更改父组件**
 - `$emit(eventName,...data)` : eventName 自定义事件名, ...data 传递的数据值
- 3、项目开发时，存在需要将**父组件方法绑定到子组件跟标签**的行为，使用事件绑定修饰符 `.native`

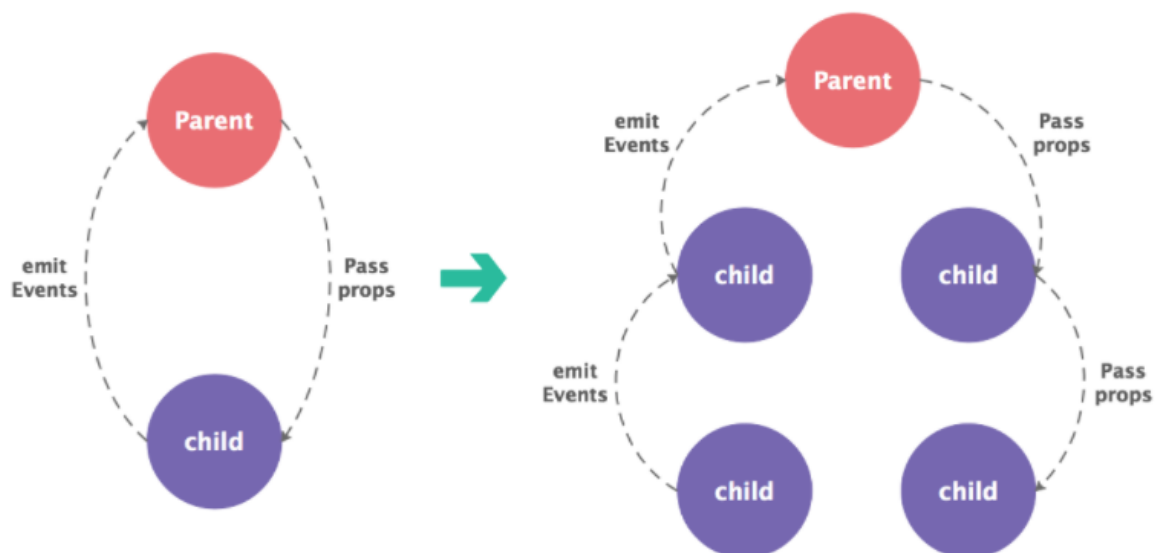
子组件中如何触发父组件方法？

- 如果是子组件的跟标签需要触发，在子组件上绑定常规事件，以.native进行修饰
- 如果不是跟标签，子组件标签上绑定自定义事件，子组件中使用 \$emit触发即可

3.5.3、非父子组件数据传递

1、借助共有顶级组件

- Vue组件构成结构无论多复杂，都必须基于 `new Vue()` 开始项目构成，因此**在同一个构成结构中，组件间必然存在一个共同的上层组件**
- 借助上层组件，使用**父=>子、子=>父**数据技术，进行层层数据传递



2、中央事件总线 (Event Bus)

- 基于在组件中可访问的**vue实例对象作为跳板实例，以方法定义和方法执行的环境变化传递数据**

Tips:Vue构成的组件化项目中，new Vue() 创建的组件一般作为整个项目的启动组件和容器组件，所以该组件叫做 Root组件

- 中央事件总线的实现**需要辅助的三个实例方法和属性**
 - `vm.$root` : 该属性提供在任意组件中可**直接获取 Root组件(new Vue())**
 - `vm.$on()` : 在实例的自定义事件库 `_events` 中装载自定义事件

- o `vm.$emit()`: 通过实例触发指定实例 `_events` 中的自定义事件

```

1 <body>
2   <div id="app">
3     <p>父组件</p>
4     <hr>
5     <comp-a></comp-a>
6     <hr>
7     <comp-b></comp-b>
8   </div>
9   <template id="a">
10    <div>
11      <p>组件A</p>
12      <p>info: {{ info }}</p>
13      <input type="button" value="打印eventBus" @click="printEventBus()">
14    </div>
15  </template>
16  <template id="b">
17    <div>
18      <p>组件B</p>
19      <p>info: {{ info }}</p>
20
21      <input type="button" value="打印eventBus" @click="printEventBus()">
22    </div>
23  </template>
24 </body>
25 <script>
26   // 1、中央事件总线，必须依赖一个vue实例
27   var eventBus = new Vue();
28   // 2、数据共享对该实例相关操作方式，让该实例成为了事件总线
29   // 用一个可以被项目中所有组件直接方法的vue实例中的 _events 事件库完成事件共享
30   // console.log(eventBus)
31   new Vue({
32     el: "#app",
33     // =====

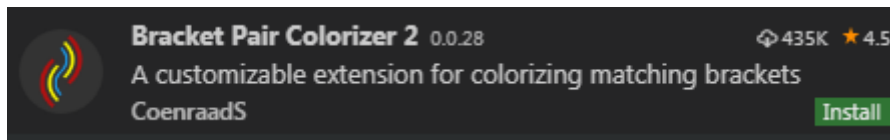
```

```

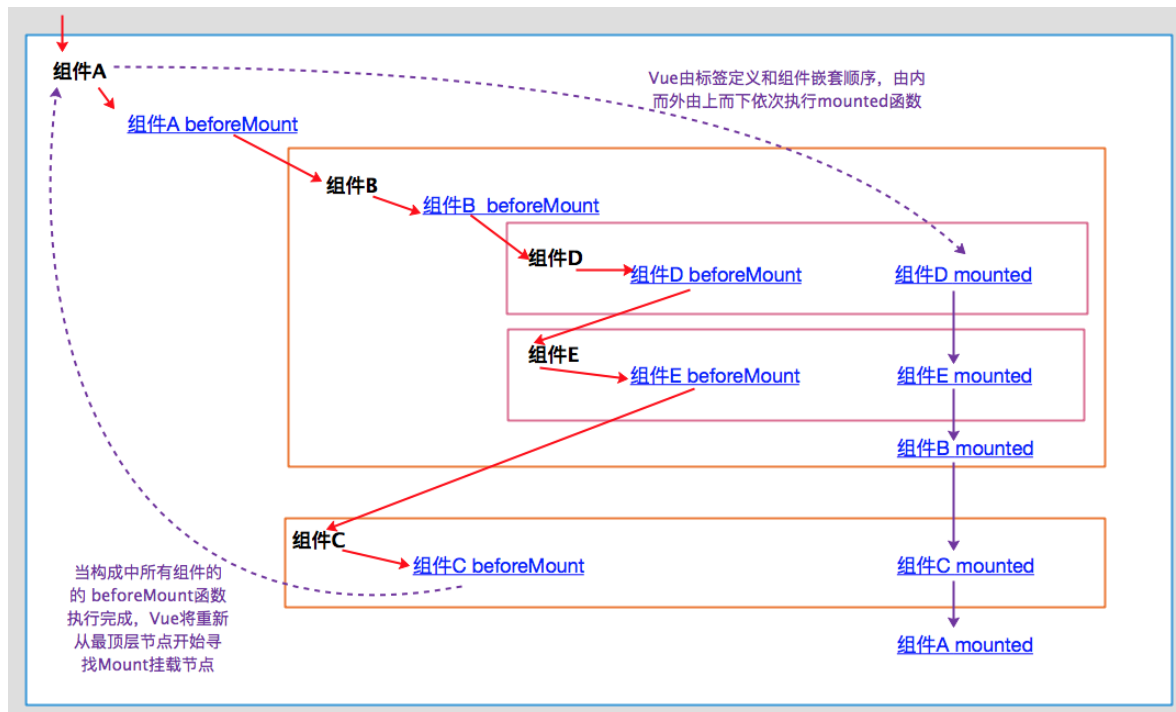
35     components: {
36         // 组件A==>
37         CompA: {
38             template: "#a",
39             data(){
40                 return {
41                     info:"组件A的变量"
42                 }
43             },
44             methods: {
45                 printEventBus(){
46                     console.log(this);
47                     // console.log(eventBus);
48                     // eventBus.$emit("update-info",this.info)
49                     console.log(this.$root);
50                     this.$root.$emit("update-info",this.info);
51                 }
52             },
53         },
54     },
55     // <==组件A
56     // 组件B==>
57     CompB: {
58         template: "#b",
59         data(){
60             return {
61                 info:""
62             }
63         },
64         methods: {
65             printEventBus(){
66                 console.log(this);
67                 console.log(eventBus);
68             }
69         },
70         mounted() {
71             // vm.$on(eventName,callback)
72             // eventName 自定义事件的事件名
73             // callback 该事件触发的回调函数
74             // eventBus.$on("update-info",(arg)=>{
75             //     this.info = arg;
76             //     console.log("组件B",arg);
77             // })
78
79             // vm.$root 在任意组件中，都是直接获取项目的 根实例
80             console.log(this.$root);
81             this.$root.$on("update-info",(arg)=>{
82                 this.info = arg;
83                 console.log("组件B",arg);
84             })
85         },
86     },
87     // <==组件B
88 }
89 })
90 </script>

```

3.5.4、组件的生命周期执行顺序



- 当项目由多个组件，进行嵌套方式完成页面构成，每个组件在项目运行时都会**独立执行生命周期**
- 因项目由多个组件组成，因此组件生命周期执行具有**先后关系**



- **总结：**
 1. 嵌套组件 beforeMount 之前包含beforeMount 生命周期，**由页面定义顺序由上而下，由外而内进行执行**
 2. 当项目中所有的 beforeMount 生命周期函数执行完成，vue将以**元素定义的顺序有上而下，有内而外依次执行mounted 函数**
 3. 由上述两个结论可知，**在组件化构成的vue结构中，所有的组件都beforeMount都将优先于mounted执行**

3.5.5、单向数据流&组件双向数据共享

1、单向数据流

- 单向数据流实际上只是一种框架语法限制的描述
- Vue组件间所有的父子prop之间形成了一个**单向下行绑定**
 - 父级 prop 的更新会**主动**向下流动到子组件中，但是反过来则不行。
 - 防止从子组件意外改变父级组件的状态，从而**导致你的应用的数据流向难以理解**。
- 有上述框架限制也进一步的描述了 `props` 拦截的变量为只读变量的特性

2、计算属性的双向共享操作

- 技术原理：通过计算属性整合**父=>子、子=>父**的数据传递特性，同时对一个变量进行数据操作，从而实现组件间的双向数据共享

3、v-bind指令的 `sync` 修饰符的双向共享操作

- 技术原理：`v-bind` 指令内置的 `.sync` 修饰符是一个**语法糖**代码，该修饰符实际是 **Vue核心语法已经整合后父=>子、子=>父 技术的封装调用方式**

4、JS引用类型的双向共享操作

- 技术原理：采用JS内存中变量存储时，引用类型堆栈分离构建的地址指向特性，实现数据的双向共享
- **注意：谨慎使用==>违背单向数据流要求**

5、独立数据监视对象的双向共享操作

- 技术原理：使用**Vue2.6.0**新增的独立数据监控对象 `Vue.observable(object)`，实现独立数据管理
 - 因数据为独立对象，所以项目中任意组件都可直接使用
 - `Vue.observable` 管理的数据变量，默认执行了 `Object.defineProperty` 完成了数据劫持和响应式功能，因此同样具有响应数据的特性
- 应用场景：该技术提供项目的数据统一管理维护接口方法

Tips：可借助Vue的全局混入技术（`Vue.mixin`），统一为组件设定监控对象的操作方法

4、内置全局组件

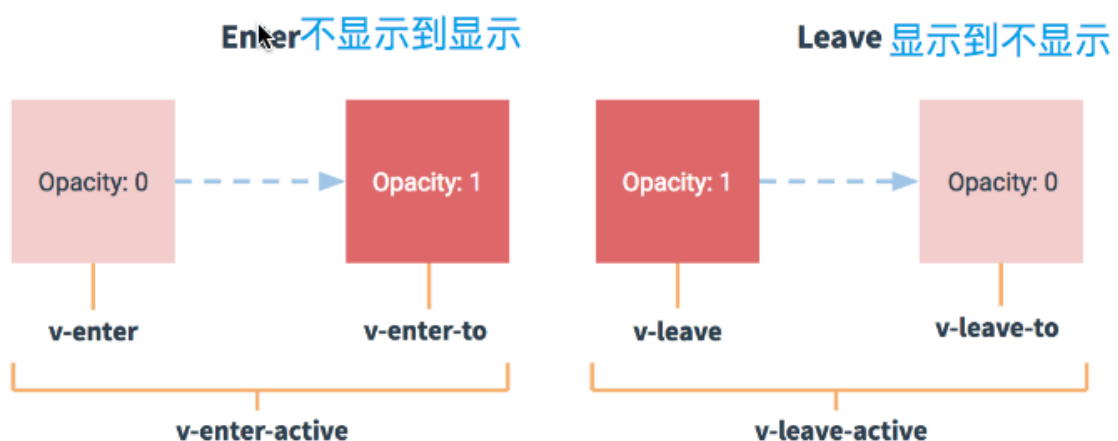
- Vue提供全局内置组件，这些组件主要完成的都是功能封装

4.1、动态组件

- 组件名：component
- 作用：vue核心语法中内置的组件，在页面中描述一个组件的占位，根据需求动态切换需要展示组件

4.2、过渡组件

- 组件名：transition
- 组件名：transition-group
- 作用：用于为vue项目中 组件、路由、页面 **提供通过vue语法控制切换**过程中的 css 动画



```
1 <head>
2   <style>
3     .start{
4       opacity: 0;
5     }
6     .active{
7       transition: all 3s ease;
8     }
9   .end{
```

```

10         opacity: 1;
11     }
12     /* h1{
13         position: absolute;
14     } */
15 </style>
16 </head>
17 <body>
18     <div id="app">
19         <p>父组件</p>
20         <hr>
21         <input type="button" value="切换标签" @click=" flag = !flag ">
22         <!--
23             过渡效果组件，通过包裹需要添加动画的元素，在vue语法控制和切换时，完成对应
24             CSS动画的装载
25             开发者需要通过自行定义css动画样式，提供装载动画
26
27             元素进入
28                 enter-class - string 描述元素进入前的样式
29                 enter-active-class - string 描述元素进入过程样式（css动画）
30                 enter-to-class - string 描述元素进入后的样式
31
32             元素离开
33                 leave-class - string 描述元素离开前的样式
34                 leave-active-class - string 描述元素离开过程样式（css动画）
35                 leave-to-class - string 描述元素离开后的样式
36         -->
37         <!--
38             transition 只为一个根标签提供动画效果
39         -->
40         <transition
41             enter-class="start"
42             enter-to-class="end"
43             enter-active-class="active"
44             leave-class="end"
45             leave-to-class="start"
46             leave-active-class="active"
47         >
48             <h1 v-if="flag" style="background-color: burlywood;">标签1</h1>
49         </transition>
50         <hr>
51         <!--
52             transition-group 对内部多个根元素进行动态控制操作
53             被动画控制的根元素 必须提供 唯一的 key 属性
54         -->
55         <transition-group
56             enter-class="start"
57             enter-to-class="end"
58             enter-active-class="active"
59             leave-class="end"
60             leave-to-class="start"
61             leave-active-class="active"
62         >
63             <h1 v-if="flag" style="background-color: burlywood;" :key=" 'item1'
64             ">标签1</h1>
65             <h1 v-if="!flag" style="background-color: cornflowerblue;"
66             key="item2">标签2</h1>
67         </transition-group>

```

```

65     <hr>
66     <transition-group
67         enter-class="start"
68         enter-to-class="end"
69         enter-active-class="active"
70         leave-class="end"
71         leave-to-class="start"
72         leave-active-class="active"
73     >
74         <p v-for="(item, index) in arr" :key="item">元素: {{ item }}</p>
75     </transition-group>
76     <input type="button" value="添加元素" @click=" arr.push(Math.random()) ">
77     <hr>
78     <transition
79         enter-class="start"
80         enter-to-class="end"
81         enter-active-class="active"
82         leave-class="end"
83         leave-to-class="start"
84         leave-active-class="active"
85     >
86         <component :is="page"></component>
87     </transition>
88 </div>
89
90 <template id="a">
91     <div style="background-color: burlywood;">
92         <p>组件A</p>
93     </div>
94 </template>
95
96 <template id="b">
97     <div style="background-color: cornflowerblue;">
98         <p>组件B</p>
99     </div>
100 </template>
101 </body>
102 <script>
103     new Vue({
104         el: "#app",
105         data:{
106             page:"CompA",
107             flag:true,
108             arr:[1,2,3]
109         },
110         components: {
111             CompA: {
112                 template: "#a",
113             },
114             CompB: {
115                 template: "#b",
116             }
117         }
118     });
119 </script>

```

4.3、组件缓存

- 组件名：keep-alive
- 作用：让已经访问的组件在内存保持存在，后续执行组件切换时，不会重新创建组件
- 属性
 - `include` - 字符串或正则表达式。只有名称匹配的**组件会被缓存**。
 - `exclude` - 字符串或正则表达式。任何名称匹配的**组件都不会被缓存**。
 - `max` - 数字。最多可以缓存多少组件实例。
- 设计生命周期函数：`activated` `deactivated`

4.4、组件分发

- 组件名：slot：
- 作用：针对于组件定义时的内部模板结构的操作，**主要在组件开发时，如果模板存在一部分页面代码无法确定，可以通过slot 定义页面占位，当组件调用时，可以通过标签中定义 新的内容，完成 slot 的内容补充**
- 标签属性：name - string 定义命名插槽
- 涉及属性：slot 属性（Vue2.6.0将使用指令 `v-slot` 替代该属性）

```

1  <body>
2    <div id="app">
3      <p>父组件</p>
4      <p>msg:{{ msg }}</p>
5      <hr>
6      <comp-a>
7        <template v-slot:default>
8          <!-- 被分发的标签依然属于 定义的容器实例 -->
9          1111111=={{ msg }}
10         </template>
11         <!-- <p slot="slotb">标签b</p> -->
12
13         <!--
14           被分发的标签需要定义在一个固定的 template 模板标签中
15           在模板标签上以 指令 v-slot:分发名称 => 描述分发位置
16         -->
17         <!--
18           命名空间分发=>让开发者定义的分发标签，可以直接获取组件中的数据变量
19           标签对应的分发占位符，需要提供属性绑定操作
20         -->
21         <template v-slot:slotb=" spaceData ">
22           <p>标签b={{ spaceData.msg }}</p>
23         </template>
24       </comp-a>
25       <hr>
26       <comp-a>
27         2222222222222222
28         <h5 slot="slotc">标签c</h5>
29       </comp-a>
30     </div>
31     <template id="a">
32       <div style="background-color: burlywood;">
33         <p>组件A</p>
34         <p>msg:{{ msg }}</p>
35         <hr>
36       <!--

```

38 组件分发： 将开发者在调用组件时，定义的自定义结构，写入到组件固定的模板位
置

39 <slot> 标签为一个组件模板结构中的占位标签，接收组件调用时开发者的自定义
代码

40 1、模式一：默认分发，不区分slot占位，不区分开发者定义页面，统一为所
有slot装载所有开发者标签

41 2、模式二：具名分发,通过为slot指定名称，让开发者页面结构选择对应标签
进行装载

```
42                    -->
43                    <slot></slot>
44                    <hr>
45                    <slot name="slotb" :msg="msg"></slot>
46                    <hr>
47                    <slot name="slotc"></slot>
48                    <hr>
49                    </div>
50                    </template>
51 </body>
52 <script>
53        new Vue({
54          el: "#app",
55          data: {
56            page: "CompA",
57            msg: "root-msg"
58          },
59          components: {
60            CompA: {
61              template: "#a",
62              data() {
63                return {
64                  msg: "compA-msg"
65                }
66              }
67            }
68          }
69        });
70 </script>
```