Vue 第一天

1.Vue.js 的组件有三个步骤： 创建组件构造器(Vue.extend()方法），注册组件(Vue.component())和实例化组件。

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="container">
      <component1></component1>
    </div>
</body>
<script src="./vue.js"></script>
<script type="text/javascript">
  // 1. 创建一个组件构造器
  var component1 = Vue.extend({
    template: '<div>hello world</div>'
  });
  // 2. 注册组件，并指定组件的标签为<component1>
  Vue.component('component1', component1);

  // 3. 实例化组件
  new Vue({
    el: '#container'
  });
</script>
</html>
```

浏览器编译后 html 结构会变为

```
<div id="container">
  <div>hello world</div>
</div>
```

页面运行显示为

hello world

2.

2-1 Vue.extend() 是 Vue 构造器的扩展，调用 Vue.extend()创建的是一个组件构造器，该构造器有一个选项对象，选项对象的 template 属性用于定义组件要渲染的html。

2-2 Vue.component() 是注册组件，需要 2 个参数，第一个参数是自定义组件的标签，第二个参数是组件的构造器。

2-3 组件需要挂载到某个 Vue 的实例下，否则不生效。

如下实例：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="container1">
      <component1></component1>
    </div>
    <div id="container2">
      <component1></component1>
    </div>

    <div id="container3">
      <component1></component1>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    // 1. 创建一个组件构造器
    var component1 = Vue.extend({
      template: '<div>hello world</div>'
    });
    // 2. 注册组件，并指定组件的标签为<component1>
```

```
    Vue.component('component1', component1);

    // 3. 实例化组件 container1
    new Vue({
        el: '#container1'
    });
    // 3. 实例化组件 container2
    new Vue({
        el: '#container2'
    });
    // 不实例化 container3 因此第三个自定义标签是不会生效的
  </script>
</html>
```

最终代码被渲染成为如下：

```
<div id="container1"><div>hello world</div></div><div id="container2"><div>hello world</div></div>
```

3.

调用 Vue.component()注册组件时，组件的注册是全局的，如果想要使用组件的局部注册的话，可以用选项对象的 components 属性实现局部注册。

如下代码：中间就把第二步注册组件哪项移到实例化组件里面来了；如下代码：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="container1">
      <component1></component1>
    </div>
    <!-- 不能使用 component1 组件，因为它是 container1 里面局部注册的 -->
    <div id="container2">
      <component1></component1>
    </div>
  </body>
  <script src="./vue.js"></script>
```

```
  <script type="text/javascript">
    // 1. 创建一个组件构造器
    var component1 = Vue.extend({
        template: '<div>hello world</div>'
    });

    // 3. 实例化组件 container1
    new Vue({
        el: '#container1',
        components: {
            'component1': component1
        }
    });
    // 实例化 container2 是不生效的
    new Vue({
        el: '#container2'
    })
  </script>
</html>
```

实例化 container2 是不生效的，并且在浏览器控制台会报如下错误：



理解父组件和子组件。

在一个组件中包含另一个组件，那么另一个组件就是该组件的子组件。

如下代码：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
```

```
    <div id="container1">
        <parent-component></parent-component>
    </div>
</body>
<script src="./vue.js"></script>
<script type="text/javascript">
    // 1. 创建一个组件构造器
    var Child  =
    Vue.extend({

    });

    var Parent = Vue.extend({
        // 在组件内部使用<child-component>组件
        template: '<div>hello world <child-component></child-component></div>',
        components: {
            // 局部注册 Child 组件
            'child-component': Child
        }
    });
    // 全局注册 Parent 组件
    Vue.component('parent-component', Parent);

    // 实例化组件
    new Vue({
        el: '#container1'
    })
</script>
</html>
```

简单理解代码如下：

1. var Child = Vue.extend(...) 定义一个 Child 组件构造器。

2. var Parent = Vue.extend(...) 定义一个 Parent 组件构造器。

3. components: {'child-component': Child}, 将 Child 组件注册到 Parent 组件，并将 Child 组件的标签设置为

child-component;

4. template: 渲染 html 模板，找到 template 选项，然后使用 child-component 组件。

5. 注册 Parent 组件 Vue.component('parent-component', Parent);

6. 最后实例化组件，需要到 html 元素为 id='container1'里面去。

Child 组件是在 Parent 组件中注册的，只能在 Parent 组件中注册的。如下几种情况都不行的。

4-1 以子标签的形式在父组件中使用；如下代码：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="container1">
      <parent-component>
        <child-component></child-component>
      </parent-component>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    // 1. 创建一个组件构造器
    var Child = Vue.extend({
      template: '<div>hello world</div>'
    });

    var Parent = Vue.extend({
      // 在组件内部使用<child-component>组件
      template: '<div>hello world</div>',
      components: {
        // 局部注册 Child 组件
        'child-component': Child
      }
    });
    // 全局注册 Parent 组件
    Vue.component('parent-component', Parent);

    // 实例化组件
```

```
    new Vue({
        el: '#container1'
    })
  </script>
</html>
```

上面调用子组件的方式是无效的，因为在js 里面当父组件要需要的 html 模板 template 的内容的时候已经决定了

需要渲染什么，所以当 parent-component 运行的时候，在父组件使用自定义的子标签。

运行时会当做 html 的普通标签来渲染，但是它又不是普通的html 标签，因此会被忽略掉。

4-2. 在父组件标签外使用子组件。

```
<div id="container1">
  <parent-component></parent-component>
  <child-component></child-component>
</div>
```

js 代码还是上面的一样，运行完成后，在浏览器下会报错如下：



我们可以使用更简单的方式来注册组件。

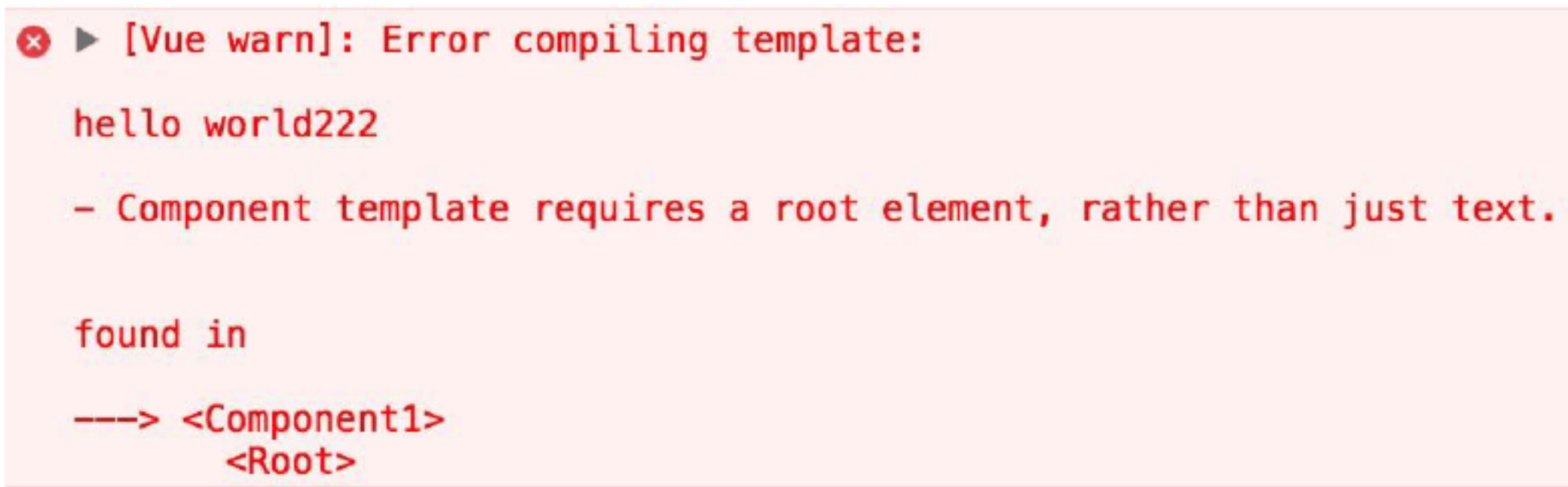
5-1 使用 Vue.component()直接创建和注册组件。如下代码：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
```

```
    <div id="container1">
        <component1></component1>
    </div>
</body>
<script src="./vue.js"></script>
<script type="text/javascript">
    // 全局注册
    Vue.component('component1
    ', {
        template: '<div>hello world222</div>'
    });
    // 实例化
    var vm1 = new Vue({
        el: '#container1'
    });
</script>
</html>
```

Vue.component()的第一个参数是标签名称，第二个参数是一个选项对象，使用选项对象的template 属性定义，使用该方式，在 Vue 源码中会调用 Vue.extend()方法。

注意： 在 template 元素中需要使用一个标签容器包围，比如我们可以把div 元素去掉的话，只放内容的话，会报错如下：



5-2 在选项对象的 components 属性中实现局部注册。

```
<!DOCTYPE html>
<html>
  <body>
    <head>
```

```
        <title>演示 Vue</title>
    </head>
    <div id="container1">
        <component1></component1>
    </div>

    <div id="container2">
        <component2></component2>
        <component3></component3>
    </div>
</body>
<script src="./vue.js"></script>
<script type="text/javascript">
    // 全局注册，my-component1 是标签名称
    Vue.component('component1',{
        template: '<div>This is the first component!</div>'
    })
    var vm1 = new Vue({
        el: '#container1'
    })

    // 实例化 局部注册
    var vm1 = new Vue({
        el: '#container2',
        components: {
            // 局部注册， component2 是标签名称
            'component2': {
                template: '<div>component2</div>'
            },
            // 局部注册，component3 是标签名称
            'component3': {
                template: '<div>component3</div>'
            }
        }
    });
</script>
</html>
```

6.

虽然语法糖简化了组件注册，但是在template 选项中拼接了 html 元素，这导致 View 和 C 层的高耦合性。幸运的是 Vue.js 提供了 2 种方式将 javascript 中的 html 模板分离出来。

6-1 使用 script 标签, 如下代码：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="container1">
      <component1></component1>
    </div>
    <script type='text/x-template' id="myComponent">
      <div>hello world</div>
    </script>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    new Vue({
      el: '#container1',
      components: {
        'component1': {
          template: '#myComponent'
        }
      }
    })
  </script>
</html>
```

注意： 使用<script>标签时，type 指定为 text/x-template，是告诉浏览器这不是一段 js 脚本，浏览器在解析 html 文档时会忽略 script 标签内定义的内容。

6-2 使用<template>标签。不需要指定 type 属性。

如下代码：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="container1">
      <component1></component1>
    </div>
    <template id="myComponent">
      <div>hello world2222</div>
    </template>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    new Vue({
      el: '#container1',
      components: {
        'component1': {
          template: '#myComponent'
        }
      }
    })
  </script>
</html>
```

7.

父组件的数据如何传给子组件呢？可以使用props 把数据传给子组件。

代码如下：

```
<!DOCTYPE html>
<html>
  <body>
```

```
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="container1">
      <component1 v-bind:my-name="name" v-bind:my-age="age"></component1>
    </div>
    <template id="myComponent">
      <table>
        <tr>
          <th colspan="2">
            子组件数据
          </th>
        </tr>
        <tr>
          <td>myName</td>
          <td>{{ myName }}</td>
        </tr>
        <tr>
          <td>myAge</td>
          <td>{{ myAge }}</td>
        </tr>
      </table>
    </template>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    new Vue({
      el: '#container1',
      data: {
        name: 'longen',
        age: 30
      },
      components: {
        'component1': {
          template: '#myComponent',
          props: ['myName', 'myAge']
        }
      }
```

```
    })
  </script>
</html>
```

注意： 在子组件中定义 prop 时，使用了 camelCase 命名法。由于 HTML 特性不区分大小写，camelCase 的 prop 用于特性时，会转为短横线隔开的，比如上面的代码：

在 props 中定义的 myName,在用作特性时需要转换为 my-name

理解 prop 的单向绑定

既然父组件使用 props 把数据传给了子组件，那么如果子组件修改了数据，对父组件是否有影响呢？

看下面的代码如下：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title>演示 Vue</title>
      <style>
        * {
          margin: 0;
          padding: 0;
          box-sizing: border-box
        }
        html {
          font-size: 12px;
          font-family: Ubuntu, simHei, sans-serif;
          font-weight: 400
        }
        body {
          font-size: 1rem
        }

        table,
        td,
        th {
          border-collapse: collapse;
          border-spacing: 0
```

```
        }

        table {
            width: 100%;
            margin: 20px;
        }

        td,
        th {
            border: 1px solid #bcbcbc;
            padding: 5px 10px
        }

        th {
            background: #42b983;
            font-size: 1.2rem;
            font-weight: 400;
            color: #fff;
            cursor: pointer
        }

        tr:nth-of-type(odd) {
            background: #fff
        }

        tr:nth-of-type(even) {
            background: #eee
        }

        fieldset {
            border: 1px solid #BCBCBC;
            padding: 15px;
        }

        input {
            outline: none
        }
```

```
        input[type=text] {
            border: 1px solid #ccc;
            padding: .5rem .3rem;
        }

        input[type=text]:focus {
            border-color: #42b983;
        }

        button {
            outline: none;
            padding: 5px 8px;
            color: #fff;
            border: 1px solid #BCBCBC;
            border-radius: 3px;
            background-color: #009A61;
            cursor: pointer;
        }
        button:hover{
            opacity: 0.8;
        }

        #container1 {
            margin: 0 auto;
            max-width: 480px;
        }
    </style>
</head>
<div id="container1">
    <table>
        <tr>
            <th colspan="3">
                父组件数据
            </th>
        </tr>
        <tr>
            <td>name</td>
            <td>{{ name }}</td>
```

```
            <td><input type='text' v-model = "name" /></td>
        </tr>
        <tr>
            <td>age</td>
            <td>{{ age }}</td>
            <td><input type="text" v-model="age" /></td>
        </tr>
    </table>
    <component1 v-bind:my-name="name" v-bind:my-age="age"></component1>
</div>
<template id="myComponent">
    <table>
        <tr>
            <th colspan="3">
                子组件数据
            </th>
        </tr>
        <tr>
            <td>myName</td>
            <td>{{ myName }}</td>
            <td><input type="text" v-model="myName" /></td>
        </tr>
        <tr>
            <td>myAge</td>
            <td>{{ myAge }}</td>
            <td><input type="text" v-model="myAge" /></td>
        </tr>
    </table>
</template>
</body>
<script src="./vue.js"></script>
<script type="text/javascript">
    new Vue({
        el: '#container1',
        data: {
            name: 'longen',
            age: 30
        },
```

```
        components: {
            'component1': {
                template: '#myComponent',
                props: ['myName', 'myAge']
            }
        }
    })
  </script>
</html>
```

Vue 第二天

每个 Vue 实例都会代理其 data 对象里所有的属性。

如下代码：

```
var data = {
  a: 1
};var vm = new Vue({
 data: data
});
console.log(vm);
console.log(vm.a === data.a); // true
// 设置属性也会影响到原始数据
vm.a = 2;
console.log(data.a); // 2
// 反之
data.a = 3;
console.log(vm.a); // 3//除了 data 属性，Vue 实例还暴露了一些有用的实例属性与方法。这些属性与方法都有前缀$，以便与代理的 data 属性区分。var data = { a: 1 }var vm = new Vue({
  el: '#container1',
  data: data
})
console.log(vm.$data === data) // true
console.log(vm.$el === document.getElementById('container1')) // true
data.a = 5;// $watch 是一个实例方法
vm.$watch('a', function (newVal, oldVal) {
```

```
  // 这个回调将在 `vm.a` 改变后调用  console.log(newVal);
  console.log(oldVal);
})
```

1-1. data必须是函数

通过 Vue 构造器传入的各种选项大多数都可以在组件里用。 data 是一个例外，它必须是函数。 如下代码 Vue 会停止，并在控制台会报错。

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title演示 Vue</title>
    </head>
    <div id="container1">

      <component1></component1>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    var data = { counter: 0 };
    // 全局注册
    Vue.component('component1', {
      template: '<span>{{ message }}</span>',
      data: {
        message: 'hello'
      }
    });
    new Vue({
      el: '#container1'
    })
  </script>
</html>
```

data 是函数解决该方案

代码如下：

```
<!DOCTYPE html>
<html>
  <body>
```

```
    <head>
      <title演示 Vue</title>
    </head>
    <div id="container1">
      <component1></component1>
      <component1></component1>
      <component1></component1>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    var data = { counter: 0 };
    // 全局注册
    Vue.component('component1', {
      template: '<button v-on:click="counter += 1">{{ counter }}</button>',
     // data 是一个函数，vue 不会报错，但是我们返回给每个组件的实列引用了同一个 data 对象
      data:function() {
       return data
      }
    });
    new Vue({
      el: '#container1'
    })
  </script>
</html>
```

由于这三个组件共享了同一个 data， 因此增加一个 counter 会影响所有组件！这不对。我们可以通过为每个组件返回全新的 data 对象来解决这个问题：

代码如下：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title演示 Vue</title>
    </head>
    <div id="container1">
      <component1></component1>
      <component1></component1>
      <component1></component1>
```

```
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    // 全局注册
    Vue.component('component1', {
      template: '<button v-on:click="counter += 1">{{ counter }}</button>',
     // data 是一个函数，vue 不会报错，但是我们返回给每个组件的实列引用了同一个 data 对象
      data:function() {
       return {
          counter: 0
        }
      }
    });
    new Vue({
      el: '#container1'
    })
  </script>
</html>
```

现在每个 counter 都有它自己内部的状态了.

2.

一般情况父子组件是这样的关系，组件A 在它的模板中使用了组件B，他们之间必然需要相互通信，父组件要给子组件传递数据，子组件需要将它内部发生的事情告知父组件，为了保证父子组件的解耦，可维护性及可重用性。在vue.js 中，父组件通过 props 向下传递数据给子组件，子组件通过events 给父组件发送消息。

2-1 使用 props 传递数据

不能在子组件的模板内直接引用父组件的数据，要让子组件使用父组件的数据，我们需要通过子组件的 props 选项。如下代码：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title演示 Vue</title>
    </head>
    <div id="container1">
```

```
      <child message="hello!"></child>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    // 全局注册
    Vue.component('child', {
      // 声明 props
      props: ['message'],
      template: '<span>{{ message }}</span>'
    });
    new Vue({
      el: '#container1'
    })
  </script>
</html>
```

结果在页面上会打印 hello。

注意： HTML 特性是不区分大小写的，所以当使用的不是字符串模板，camelCased(驼峰式) 命名的 prop 需要转换为相对应的 kebab-case(短横线隔开式)命名：

如下代码：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title演示 Vue</title>
    </head>
    <div id="container1">
      <!-- kebabease in HTML-->
      <child my-message="hello!"></child>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    // 全局注册
    Vue.component('child', {
      // 声明 props
      props: ['myMessage'],
```

```
      template: '<span>{{ myMessage }}</span>'
    });
    new Vue({
      el: '#container1'
    })
  </script>
</html>
```

2-2　理解动态

在模板中，要动态地绑定父组件的数据到子模板的props，使用 v-bind，每当父组件的数据变化时，该变化会传递给子组件。

```
<div id="container1">
  <input v-model='parentMsg' />
  <br />
  <!-- kebab-case in HTML-->
  <child v-bind:my-message="parentMsg"></child>
</div>
```

使用 v-bind 的缩写语法通常更简单：

```
<child :my-message="parentMsg"></child>
```

代码如下：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title演示 Vue</title>
    </head>
    <div id="container1">
      <input v-model='parentMsg' />
      <br />
      <!-- kebabease in HTML-->
      <child v-bind:my-message="parentMsg"></child>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    new Vue({
      el: '#container1',
```

```
      data: {
        parentMsg: 'Message'
      },
      components: {
        child: {
          props: ['myMessage'],
          template: '<span>{{myMessage}}</span>'
        }
      }
    })
  </script>
</html>
```

3.

父组件使用 props 传递数据给子组件，但是如果子组件需要把数据传回去的话，就需要自定义事件了；

3-1 使用 v-on 绑定自定义事件

每个 vue 实例都实现了事件接口，即：

1. 使用 监听事件

2.使用on(eventName)监听事件 2.使用emit(eventName) 触发事件

注意：on 和 emit 不是 addEventListener 和 dispatchEvent 的别名。且 父组件可以在使用组件的地方直接用 v-on 来监听子组件触发的事件。

不能用$on 侦听子组件抛出的事件，而必须在模板里直接用 v-on 绑定，就像以下的例子：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title演示 Vue</title>
    </head>
    <div id="container1">
      <p> {{ total }} </p>
      <button-counter v-on:increment="incrementTotal"></button-counter>
      <button-counter v-on:increment="incrementTotal"></button-counter>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    Vue.component('button-counter', {
```

```
    template: '<button v-on:click="increment">{{ counter }}</button>',
    data:function() {
     return {
        counter: 0
      }
    },
    methods: {
      increment:function() {
       this.counter += 1;
       this.$emit('increment');
      }
    },
  })
  new Vue({
    el: '#container1',
    data: {
      total: 0
    },
    methods: {
      incrementTotal:function() {
       this.total += 1;
      }
    }
  })
</script>
</html>
```

上面代码： 初始化时候 实例化设置 data: {total: 0}, 设置 total 为 0， 子组件 button-counter 默认为 0， 当点击子组件的时候调用 increment 方法，当前的 counter 自增 1， 然后在子组件触发 $emit('increment')事件，当使用 v-on:increment 会监听到事件后，会调用父组件的incrementTotal 方法，因此父组件也自增 1.

上面代码中 子组件已经和它外部完全解耦了。它所做的只是报告自己的内部事件，至于父组件是否关心则与它无关。

4.

自定义事件可以用来创建自定义的表单输入组件，使用v-modal 来进行数据双向绑定。比如如下代码：

```
<input v-modal="something" />
```

上面的代码是下面的语法糖；如下代码：

```
<input v-bind:value="something" v-on:input="something=$event.target.value" />
```

因此在创建组件中时，相当于下面的简写；如下代码：

```
<custom-input v-bind:value="something" v-on:input="something=arguments[0]"></custom-input>
```

所以要让组件的 v-model 生效，必须满足下面的条件：

1. 接受一个 value 属性。
2. 在有新的 value 时触发 input 事件。

如下测试代码：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title演示 Vue</title>
    </head>
    <div id="container1">
      <currency-input v-model="price"></currency-input>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script>

  </script>
  <script type="text/javascript">
    Vue.component('currency-input', {
      template: '\
        <span>\
          $\
          <input\
            ref="input"\
            v-bind:value="value"\
            v-on:input="updateValue($event.target.value)"\
          >\
        </span>\
      ',
      props: ['value'],
      methods: {
       // 不是直接更新值，而是使用此方法来对输入值进行格式化和位数限制
        updateValue:function (value) {
         var formattedValue = value
```

```
            // 删除两侧的空格符                .trim()
            // 保留 2 小数位
              .slice(0, value.indexOf('.') + 3)
          // 如果值不统一，手动覆盖以保持一致
          if (formattedValue !== value) {
            this.$refs.input.value = formattedValue
          }
          // 通过 input 事件发出数值
          this.$emit('input', Number(formattedValue))
        }
      }
    });
    new Vue({
      el: '#container1',
      data: {
        price: 0
      }
    })
  </script>
</html>
```

5.

<slot>标签中的任何内容都被视为 **备用内容**。备用内容在子组件的作用域内编译，并且只有在宿主元素为空，且没有插入的内容时才显示备用内容。

如果<slot>标签中有内容的话，就显示该内容。

比如 **mycomponent** 组件有如下代码：

```
<div class="content">
  <h2>this is a component</h2>
  <slot>如果没有分发内容，则显示 slot 中的内容</slot>
  <p>asdsadsdad</p>
</div>
```

父组件有如下代码：

```
<div id="container1">
  <my-component>
    <h1>Hello Vue.js</h1>
  </my-component>
```

```
  <my-component></my-component>
</div>
```

渲染后的结果为：

```
<div id="container1">
  <div class="content">
    <h2>this is a component</h2>
    <h1>Hello Vue.js</h1>
    <p>asdsadsdad</p>
  </div>
  <div class="content">
    <h2>this is a component</h2>     如果没有分发内容，则显示 slot 中的内容
    <p>asdsadsdad</p>
  </div>
</div>
```

所有测试实例代码如下：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title演示 Vue</title>
    </head>

    <div id="container1">
      <my-component>
        <h1>Hello Vue.js</h1>
      </my-component>
      <my-component></my-component>
    </div>

    <template id="myComponent">
      <div class="content">
        <h2t>his is a component</h2>
        <slot如果没有分发内容，则显示 slot 中的内容</slot>
        <p>asdsadsdad</p>
      </div>
    </template>
```

```
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    Vue.component('my-component', {
      template: '#myComponent'
    })
    new Vue({
      el: '#container1'
    })
  </script>
</html>
```

6.

<slot> 元素可以用一个特殊的属性 name 来配置如何分发内容。多个 slot 可以有不同的名字。具名 slot 将匹配内容片段中有对应 slot 特性的元素。

如果没有默认的 slot ，这些找不到匹配的内容片段将被抛弃。

比如：假如有一个 my-component 组件，它的模板为：

```
<template id="myComponent">
  <div class='content'>
    <header>
      <slot name='header'></slot>
    </header>
    <main>
      <slot></slot>
    </main>
    <footer>
      <slot name='footer'></slot>
    </footer>
  </div>
</template>
```

父组件的模板如下：

```
<div id="container1">
  <h1 slot="header">这里可能是一个页面标题</h1>
  <p>主要内容的一个段落</p>
  <p>另一个主要段落</p>
  <p slot='footer'>这里是底部信息</p>
</div>
```

页面渲染的结果如下：

```
<div id="container1">
  <h1>这里可能是一个页面标题</h1>
  <p>主要内容的一个段落</p>
  <p>另一个主要段落</p>
  <p>这里是底部信息</p>
</div>
```

所有的代码如下：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title演示 Vue</title>
    </head>

    <div id="container1">
      <h1 slot="header"这里可能是一个页面标题</h1>
      <p主要内容的一个段落</p>
      <p另一个主要段落</p>
      <p slot='footer'这里是底部信息</p>
    </div>

    <template id="myComponent">
      <div class='content'>
        <header>
          <slot name='header'></slot>
        </header>
        <main>
          <slot></slot>
        </main>
        <footer>
          <slot name='footer'></slot>
        </footer>
      </div>
    </template>

  </body>
```

```
  <script src="./vue.js"></script>
  <script type="text/javascript">
    Vue.component('my-component', {
      template: '#myComponent'
    })
    new Vue({
      el: '#container1'
    })
  </script>
</html>
```

在 slot 分发中，无论是单分发还是具名分发，都是父组件替换子组件的数据，或者没有替换，用子组件默认的数据。 但是通过设置作用域槽，就可以改变这种状况，让子组件可以在父组件进行分发时获取自己的数据，至于是什么数据，由子组件决定，这样就能解耦了。

作用域槽通过 slot 的一个自定义的属性，官方给出的DEMO 是 text，但也可以是其他，值为暴露的数据。 这个自定义属性已经存放在子组件的 prop 对象里了。等待着被父组件获取。

怎么获取呢？ 在父组件的模板里，使用一个 Vue 自带的特殊组件<template> ，并在该组件上使用 scope 属性，值是一个临时的变量，存着的是由子组件传过来的prop 对象，获得由子传过来的prop 对象。这时候，父组件就可以访问子组件在自定义属性上暴露的数据了。

如下代码：

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <title演示 Vue</title>
    </head>
    <div id="container1">
      <parent-component></parent-component>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    // 子组件
    Vue.component('child-component', {
      template: '<ul><slot name="child-ul" v-for="item in rets"
v-bind:text="item.name"></slot></ul>',
```

```
      data:function() {
       return {
          rets: [
            {name:我是苹果'},
            {name:我是香蕉'},
            {name:我是橘子'}
          ]
        }
      }
    });
   // 父组件
    Vue.component('parent-component', {
      template: '<child-component><template scope="props"
slot="child-ul"><li>{{props.text}}</li></template></child-component>'
    })
    new Vue({
      el: '#container1'
    })
  </script>
</html>
```

页面渲染后的代码如下：

```
<div id="container1">
  <ul>
    <li>我是苹果</li>
    <li>我是香蕉</li>
    <li>我是橘子</li>
  </ul></div>
```

8.

通过使用保留的<component>元素，动态地绑定到它的 is 特性，我们可以让多个组件使用同一个挂载点，并动态的切换。

keep-alive: 如果把切换出去的组件留在内存中，可以保留它的状态或避免重新渲染，为此我们可以添加一个 keep-alive 指令参数。

如下实现的 tab 切换代码：

```
<!DOCTYPE html>
<html>
  <body>
```

```
    <head>
      <title演示 Vue</title>
    </head>

    <h3动态组件</h3>
    <template id="tab-01">
      <div>his is tab01</div>
    </template>
    <template id='tab-02'>
      <div>his is tab02</div>
    </template>
    <template id="tab-03">
      <div>his is tab03</div>
    </template>

    <div id="container1">
      <!--导航栏 -->
      <ul>
        <li>
          <a href="javascript:void(0)" @click="toggleTabs(tab01Text);">{{ tab01Text }}</a>
        </li>
        <li>
          <a href="javascript:void(0)" @click="toggleTabs(tab02Text);">{{ tab02Text }}</a>
        </li>
        <li>
          <a href="javascript:void(0)" @click="toggleTabs(tab03Text);">{{ tab03Text }}</a>
        </li>
      </ul>
      <!--点击导航后要切换的内容 -->
      <div class='content' style='height: 200px'>
        <!--如果把切换出去的组件保留在内存中，可以保留它的状态或避免重新渲染。为此可以添加一个 keep-alive 指令参
数 -->
        <keep-alive>
           <component :is="currentView"></component>
        </keep-alive>
      </div>
    </div>
  </body>
```

```
  <script src="./vue.js"></script>
  <script type="text/javascript">
     var tab01 = Vue.extend({
        template: '#tab-01'
      });
     var tab02 = Vue.extend({
        template: '#tab-02'
      });
     var tab03 = Vue.extend({
         template: '#tab-03'
      });
     // 新建 vue 实例
     var newVue = new Vue({
        el: '#container1',
        data: {
          tab01Text: "tab01",// 菜单一
          tab02Text: "tab02",// 菜单二
          tab03Text: "tab03",// 菜单三
          currentView: "tab01"// 默认选中的导航栏        },
       // 局部注册组件        components: {
          tab01: tab01,
          tab02: tab02,
          tab03: tab03,
        },
        methods: {
         // 绑定 tab 的切换事件
          toggleTabs:function(tabText) {
          this.currentView = tabText;
          }
        }
       })
  </script>
</html>
```

Vue 第三天

1.

在模板内可以对一些属性的计算。如下代码：

```
<!DOCTYPE html><html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="app">
      <p>正序："{{ message }}"</p>
      <p>倒序："{{ reversedMessage }}"</p>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    var vm = new Vue({
      el: '#app',
      data: {
        message: 'Hello'
      },
      computed: {
        reversedMessage:function() {
         return this.message.split('').reverse().join('')
        }
      }
    })
  </script>
</html>
```

输出结果为：

正序："Hello"
倒序："olleH"

如上我们提供的函数将用作属性 vm.reversedMessage 的 getter 。如下代码：

```
console.log(vm.reversedMessage) // -> 'olleH'
vm.message = 'Goodbye'
console.log(vm.reversedMessage) // -> 'eybdooG'
```

我们可以通过调用表达式中的 method 来达到同样的效果：代码如下：

```
<!DOCTYPE html><html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="app">
      <p>Reversed message: "{{ reversedMessage() }}"</p>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    var vm = new Vue({
      el: '#app',
      data: {
        message: 'Hello'
      },
      methods: {
        reversedMessage:function () {
         return this.message.split('').reverse().join('')
        }
      }
    });
    console.log(vm.reversedMessage())// -> 'olleH'
  </script>
</html>
```

那么他们的区别是?

计算属性是基于他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。也就是说只要 message 值没有发生变化，多次访问 reversedMessage 计算属性会立即返回之前的结果，不必再执行函数。

但是 method 方法只要发生重新渲染，总是会执行该函数。

1. 绑定 Html class --- 对象语法

我们可以传给 v-bind:class 一个对象，以动态的切换 class。如下代码：

```
<!DOCTYPE html><html>
  <body>
```

```
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="app">
      <div class="static" v-bind:class = "{ active: isActive, 'text-danger': hasError }"></div>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    var vm = new Vue({
      el: '#app',
      data: {
        isActive:true,
        hasError:true
      }
    });
  </script></html>
```

代码被渲染为如下：

```
<div id="app"><div class="static active text-danger"></div></div>
```

当 isActive 或者 hasError 变化时，class 列表将相应地更新。例如，如果hasError 的值为 false ， class 列表将变为 "static active "。

也可以直接绑定数据里的一个对象：如下代码

```
<!DOCTYPE html><html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="app">
      <div class="static" v-bind:class="classObject"></div>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    var vm = new Vue({
      el: '#app',
      data: {
        classObject: {
```

```
            active:true,
            'text-danger':true
          }
        }
    });
  </script></html>
```

渲染效果和上面一样。

也可以在这里绑定返回对象的计算属性。如下代码：

```
<!DOCTYPE html><html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="app">
      <div class="static" v-bind:class="classObject"></div>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    var vm = new Vue({
      el: '#app',
      data: {
        isActive:true,
        error:null
      },
      computed: {
        classObject:function () {
          return {
            activethis.isActive && !this.error,
            'text-danger'this.error && this.error.type === 'fatal',
          }
        }
      }
    });
  </script></html>
```

渲染后的代码如下：

```
<div id="app"><div class="static active"></div></div>
```

2. 数组语法

我们可以把一个数组传给 v-bind:class ， 以应用一个 class 列表：如下代码：

```
<!DOCTYPE html><html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="app">
      <div class="static" v-bind:class="[activeClass, errorClass]"></div>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">
    var vm = new Vue({
      el: '#app',
      data: {
        activeClass: 'active',
        errorClass: 'text-danger'
      }
    });
  </script></html>
```

被渲染为如下代码：

```
<div id="app"><div class="static active text-danger"></div></div>
```

3. 使用在组件上

例如，如果你声明了这个组件：

```
Vue.component('my-component', {
  template: '<p class="foo bar">Hi</p>'
})var vm = new Vue({
  el: '#app'
});
```

使用组件的时候 添加一些 class，如下代码：

```
<div id="app">
  <my-component class='baz boo'></my-component></div>
```

所有代码如下：

```
<!DOCTYPE html><html>
```

```
<body>
  <head>
    <title>演示 Vue</title>
  </head>
  <div id="app">
    <my-component class='baz boo'></my-component>
  </div>
</body>
<script src="./vue.js"></script>
<script type="text/javascript">

  Vue.component('my-component', {
    template: '<p class="foo bar">Hi</p>'
  })
  var vm = new Vue({
    el: '#app'
  });
</script></html>
```

HTML 最终被渲染为：

```
<div id="app"><p class="foo bar baz boo">Hi</p></div>
```

同样的适用于绑定 HTML class :

```
<!DOCTYPE html><html>
<body>
  <head>
    <title>演示 Vue</title>
  </head>
  <div id="app">
    <my-component v-bind:class="{ active: isActive }"></my-component>
  </div>
</body>
<script src="./vue.js"></script>
<script type="text/javascript">

  Vue.component('my-component', {
    template: '<p class="foo bar">Hi</p>'
  })
  var vm = new Vue({
```

```
      el: '#app',
      data: {
        isActive:true
      }
    });
  </script></html>
```

当 isActive 为 true 的时候，HTML 最终被渲染成为如下代码：

```
<div id="app"><p class="foo bar active">Hi</p></div>
```

4. 级联内联样式

如下级联样式代码：

```
<!DOCTYPE html><html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="app">
      <div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }">11112222</div>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">

    Vue.component('my-component', {
      template: '<p class="foo bar">Hi</p>'
    })
    var vm = new Vue({
      el: '#app',
      data: {
        activeColor: 'red',
        fontSize: 30
      }
    });
  </script></html>
```

页面被渲染为如下代码：

```
<div id="app"><div style="color: red; font-size: 30px;">11112222</div></div>
```

也可以绑定到一个样式对象通常更好，让模板更清晰：如下代码：

```
<!DOCTYPE html><html>
  <body>
    <head>
      <title>演示 Vue</title>
    </head>
    <div id="app">
      <div v-bind:style="styleObject">11112222</div>
    </div>
  </body>
  <script src="./vue.js"></script>
  <script type="text/javascript">

    Vue.component('my-component', {
      template: '<p class="foo bar">Hi</p>'
    })
    var vm = new Vue({
      el: '#app',
      data: {
        styleObject: {
          color: 'red',
          fontSize: '22px'
        }
      }
    });
  </script></html>
```

代码被渲染成为如下：

```
<div id="app"><div style="color: red; font-size: 22px;">11112222</div></div>
```

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：
https://d.book118.com/097122015015006134