【Tips】

1. Vue的属性目前有el、data、computed、created、mounted、beforeDestroy、methods、filters，这些属性有的是单数形式，有的是复数形式，目前看来可以这么区分：（1）el属性肯定是单数的，只能有一个（2）函数属性都是单数的，一个Vue对象内只需要一个同名函数（3）属性相关都是单数的，如data、computed（4）函数集合属性是复数的，如methods、filters。

一、基础篇

1. VUE.js的优点有哪些？精巧、渐进式，解耦能力强大，组件式开发能力强大，前端路由，状态管理，虚拟DOM。

2. Hello World

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>001</title>
</head>
<body>
    <!-- 标签里面的属性值，已经不再是简单的字符串，它也可能是个变量、表达式 -->
    <div id="app">
        <input type="text" v-model="name" placeholder="你的名字">
        <!-- 文本插值（Mustache语法），插值可以是一个变量也可以是一个表达式，但是不能是一句语句 -->
        <h1>你好，{{ name }}</h1>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            // el用于指定一个页面中已存在的DOM元素来挂载Vue 实例，它可以是HTMLElement ，也可以是 css 选择器
            // el: document.getElementByld('app') //或者是'#app'
            el: "#app",
            // 所有会用到的数据都 预先在 data 内声明，这样不至于将数据散落在业务逻辑中
            data: {
                name: ""
            }
        })
        // Vue实例本身也代理了 data对象里的所有属性，所以可以这样访问：
        // console.log(app.name);
    </script>
</body>
</html>
```

3. 生命周期 & 插值

• created 实例创建完成后调用，此阶段完成了数据的观测等，但尚未挂载， $el 还不可用。 需要初始化处理一些数据时会比较有用，后面章节将有介绍。

• mounted el 挂载到实例上后调用，一般我们的第一个业务逻辑会在这里开始 。

• beforeDestroy 实例销毁之前调用。主要解绑一些使用 addEventListener 监听的事件等。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>002</title>
</head>
<body>
    <div id="app">
        <!-- (1) 普通插值标签 -->
        <div>{{ date }}</div>
        <!-- (2) 表达式插值 -->
        <div>{{ number + 10 }}</div>
        <!-- (3) v-pre用于输出未编译的内容(pre可理解为编译之前的，这一块内容也不会被编译），v-pre标志不需编译，仅标志的形式 -->
        <div v-pre>{{ date }}</div>
        <!-- (4) v-html用于插入HTML文本，v-html作为接收html文本的标志，它的值为link变量，标志+变量的形式 -->
        <div v-html="link"></div>
        <!-- (5) v-if用于条件控制标签是否显示，标志+表达式的形式 -->
        <div v-if="number>1">你能看见我吗? </div>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: "#app",
            data: {
                date: new Date(),
                link: '<a href=”#”>这是一个连接</a>',
                number: 2
            },
            mounted: function () {
                var _this = this;
                this.timer = setInterval(function(){
                    _this.date = new Date();
                }, 1000);
            },
            beforeDestroy: function () {
                if (this.timer) {
                    clearInterval(this.timer);
                }
            }
```

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

4. 过滤器

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>003</title>
</head>
<body>
    <div id="app">
        <!-- 插值过滤，插值本身作为过滤器的第一个参数传入 -->
        <div>{{ date | formatDate }}</div>
        <!-- 链式过滤 -->
        <div>{{ scores | passedFilter }}</div>
        <div>{{ scores | passedFilter | excellentFilter }}</div>
        <!-- 接收参数，70作为函数的第二个参数传入 -->
        <div>{{ scores | scoreFilter(70) }}</div>

        <!-- 【注】过滤器应当用于处理简单的文本转换，如果要实现更为复杂的数据变换，应该使用计
算属性，下一章中会详细介绍它的用法， -->
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script>
        var padDate = function (value) {
            return value < 10 ? '0' + value : value;
        };
        var app = new Vue({
            el: "#app",
            data: {
                date: new Date(),
                scores: [45, 66, 77, 99]
            },
            mounted: function () {
                var _this = this;
                this.timer = setInterval(function () {
                    _this.date = new Date();
                }, 1000);
            },
            beforeDestroy: function () {
                if (this.timer) {
                    clearInterval(this.timer);
                }
            },
            // 过滤器集合
            filters: {
                // 单个过滤器函数，函数的第一个参数为插值本身
```

```
                formatDate: function (value) {
                    var date = new Date(value);
                    var year = date.getFullYear();
                    var month= padDate(date.getMonth() + 1) ;
                    var day= padDate(date .getDate());
                    var hours= padDate(date .getHours());
                    var minutes = padDate (date .getMinutes ());
                    var seconds = padDate (date .getSeconds ());

                    return year + '-' + month + '-' + day + ' ' + hours +
':' + minutes + ':' + seconds;
                },
                passedFilter: function (value) {
                    var result = [];
                    if (value instanceof Array) {
                        for (var i = 0; i < value.length; i++) {
                            var score = value[i];
                            if (score < 60) continue;
                            result.push(score);
                        }
                    }
                    return result;
                },
                excellentFilter: function (value) {
                    var result = [];
                    if (value instanceof Array) {
                        for (var i = 0; i < value.length; i++) {
                            var score = value[i];
                            if (score < 90) continue;
                            result.push(score);
                        }
                    }
                    return result;
                },
                // 带参数的过滤器函数
                scoreFilter: function (value, paramScore) {
                    var result = [];
                    if (value instanceof Array) {
                        for (var i = 0; i < value.length; i++) {
                            var score = value[i];
                            if (score < paramScore) continue;
                            result.push(score);
                        }
                    }
                    return result;
                }
            }
        });
    </script>
</body>
</html>
```

5. 指令

指令(Directives)是 Vue.js模板中最常用的一项功能，它带有前缀 v-， 在前文我们已经使用 过不少指令了，比如 v-if、 v-html、 v-pre等。指令的主要职责就是当其表达式的值改变时，相应地 将某些行为应用到 DOM 上。例如之前的v-if指令，在number>1时，DOM上的对应Element会显示，否则会隐藏。【数据驱动 DOM 是 Vue.js 的核心理念，所以不到万不得已时不要主动操作DOM，你只需要维护好数据， DOM的事 Vue会 帮你优雅的处理。】

这里不会介绍所有的指令，别着急，按涉及的点慢慢来介绍，这里先介绍v-bind与v-on。

v-bind 的基本用途是动态更新 HTML 元素上的属性，比如 id、class等。

v-on 用来绑定事件监听器。它可以监听原生的 DOM 事件，除了 click 外，还有 dblclick、 keyup, mousemove 等。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>004</title>
</head>
<body>
    <style>
        a {
            display: block;
        }
        img {
            width: 90px;
            height: 90px;
            display: block;
            margin: 20px;
        }
    </style>
    <div id="app">
        <div>
            <!-- v-bind，如果不加v-bind，那么href就是个常量，加了v-bind之后，href就对应Vue实例的url变量，会随着变量值的改变而改变 -->
            <a v-bind:href="url" target="blank">这是一个v-bind链接</a>
            <img v-bind:src="imgURL" v-if="seen">
            <!-- v-on -->
            <button v-on:click="toggleImg">{{ tips }}</button>

            <!-- 【语法糖】是指在不影响功能的情况下，添加某种方法实现同样的效果，从而方便程序开发。 -->
            <!-- img元素代码等同于 <img :src="imgURL" v-if="seen"> -->
            <!-- button元素代码等同于 <button @click="toggleImg">{{ tips }}</button> -->
        </div>
```

```
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script>
        var app = new Vue({
            el: "#app",
            data: {
                url: "http://www.4399.com",
                imgURL: "../image/a.jpg",
                seen: true,
                tips: "点击隐藏图片"
            },
            // methods属性，它里面的函数被Vue变量代理，可以通过app.toggleImg进行访问，跟data数据相同
            methods: {
                toggleImg: function () {
                    this.seen = !this.seen;
                    this.tips = this.seen ? "点击隐藏图片" : "点击显示图片";
                }
            }
        });
    </script>
</body>
</html>
```

【注】v-model指令虽然是我们第一个代码示例中就用到的指令，但是现在才可以较好的解释它的作用。 v-model 指令在 <input> (<input> 标签有多种类型，如 button、select 等等)及 <textarea> 元素上进行双向数据绑定，它负责监听用户的输入事件以更新数据。

6. 计算属性

插值可以做简单的运算，但是它并不支持语句，复杂的多语句代码更加没法执行，所以在处理这一类数据时，计算属性就起了很好的作用。简单实例如下：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>005</title>
</head>
<body>
    <div id="app">
        <div>{{ reverseString }}</div>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script>
        var app = new Vue({
            el: "#app",
            data: {
                string: "Hello Vue!"
            },
```

```
            computed: {
                reverseString: function () {
                    return this.string.split('').reverse().join('');
                }
            }
        });
    </script>
</body>
</html>
```

在一个计算属性里可以完成各种复杂的逻辑，包括运算、函数调用等，只要最终返回一个结果就可以。除了上例简单的用法，计算属性还可以依赖多个 Vue 实例的数据，只要其中任一数据变化，计算属性就会重新执行，视图也会更新。例如，下面的示例展示的是在购物车内两个包裹的物品总价：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title></title>
</head>
<body>
    <div id= "app">
        总价: {{ prices }}
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script>
        var app = new Vue({
            el: "#app",
            data: {
                package1: [
                    {
                        name: 'iPhone7',
                        price: 6288,
                        count: 1
                    },
                    {
                        name: 'iPad',
                        price: 2688,
                        count: 2
                    }
                ],
                package2: [
                    {
                        name: 'iPhoneXs',
                        price: 10888,
                        count: 1
                    }
                ]
            },
            computed: {
                prices: function () {
```

```
                    var totalPrice = 0;
                    for (var i = 0; i < this.package1.length; i++) {
                        var item = this.package1[i];
                        totalPrice += item['price'] * item['count'];
                    }
                    for (var i = 0; i < this.package2.length; i++) {
                        var item = this.package2[i];
                        totalPrice += item['price'] * item['count'];
                    }
                    return totalPrice;
                }
            }
        });
    </script>
</body>
</html>
```

每一个计算属性都包含一个getter和一个 setter，下面的示例中演示了setter方法的使用:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>007-计算属性setter</title>
</head>
<body>
    <div id="app">
        <div>姓名: {{ fullname }}, 姓: {{ firstname }}, 名: {{ lastname }}</
div>
        <input v-model="fullname" type="text" placeholder="这里可以输入名字全
称">
    </div>
    <div id="anotherApp">
        {{ reversename }}
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: "#app",
            data: {
                firstname: '',
                lastname: ''
            },
            computed: {
                // 计算属性可以是一个函数变量，也可以是一个函数集合变量
                fullname: {
                    get: function () {
                        return this.firstname + ' ' + this.lastname;
                    },
                    set: function (newValue) {
                        var names = newValue.split(' ');
```

```
                        if (names.length > 1) {
                            this.firstname = names[0];
                            this.lastname = names[names.length - 1];
                        }
                    }
                }
            }
        });
        var anotherApp = new Vue({
            el: "#anotherApp",
            computed: {
                // 计算属性不仅可以依赖当前Vue实例的数据，还可以依赖其他Vue实例的数据
                reversename: function () {
                    return app.fullname.split('').reverse().join('');
                }
            }
        });
    </script>
</body>
</html>
```

【注】计算属性是基于它的依赖属性进行缓存的，一个计算属性所依赖的属性发生变化时，它才会重新取值。

7. v-bind:class、:class，v-bind:style、:style，对原生class和style的强化应用

7.1 v-bind:class、:class的基本使用

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>008-增强class</title>
    <style>
        #app > div {
            width: 100px;
            height: 100px;
            background-color: gray;
            margin: 20px;
        }
        #app > div.red {
            background-color: red;
        }
        #app > div.blue {
            background-color: blue;
        }
        #app > div.higher {
            height: 150px;
        }
        #app > div.wider {
            width: 150px;
        }
    </style>
</head>
```

```
<body>
    <div id="app">
        <!-- 这里的示例说明，标签属性值不仅可以是常量、单个变量、表达式，还可以是集合变量 -->
        <div v-bind:class="{'red':isRed, 'wider':isWider}"></div>

        <!-- v-bind:class, :class可以与原生class共存 -->
        <!-- 当:class中的键值对值为真时，该class值就被渲染到原生class中 -->
        <!-- 下面这个isBlue为false时，blue这个class值就不会被渲染进原生class中，下面这个div框就不会被渲染为蓝色 -->
        <div :class="{'blue':isBlue}" class="higher"></div>

        <!-- 当class值很多时，你可以用表达式，甚至计算属性 -->
        <div :class="computedClass"></div>

        <!-- :class的值还可以是一个数组 -->
        <div :class="['wider', higherTag, isRed ? 'red' : 'blue']"></div>

        <input @click="toggleState" type="button" value="点击切换状态">
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: "#app",
            data: {
                isRed: true,
                isWider: true,
                isBlue: false,
                higherTag: 'higher'
            },
            computed: {
                computedClass: function () {
                    return {'red':(this.isRed && this.isBlue), 'blue':!(this.isRed && this.isBlue), 'wider':true, 'higher':true};
                }
            },
            methods: {
                toggleState: function () {
                    this.isRed = !this.isRed;
                }
            }
        });
    </script>
</body>
</html>
```

7.2 在组件中使用v-bind:class、:class

组件定义：

```
Vue.component('my-component', {
```

```
    template: '<p class="article"></p>'
});
```

使用组件：

```
<div id="app">
    <my-component :class="{'acttive':isActive}"></my-component>
</div>
<script>
    var app = new Vue({
        el: "#app",
        data: {
            isActive: true
        }
    });
</script>
```

最终渲染结果：

```
<p class="article active">一些文本</p>
```

【注】这种用法仅适用于自定义组件的最外层是一个根元素，否则会无效。

7.3 v-bind:style、:style

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>009-增强style</title>
</head>
<body>
    <div id="app">
        <!-- 一些注意点跟增强class相同，不再赘述，这里只备注一下增强style特有的点 -->
        <!-- 1.常量如100px在增强style中要用引号包括起来，表示它是一个常量 -->
        <!-- 2.类似font-size这种style属性既可以横线分隔法命名，也可以用驼峰命名法命名，最终的渲染结果都是font-size -->
        <div style="width: 300px" :style="{'height': '100px', 'fontSize':fontSize+'px'}">Hello Vue!</div>
        <!-- 更推荐用属性或者计算属性，便于维护 -->
        <div :style="styles">我的style是通过属性得到的</div>
        <!-- 增强style还可以是一个集合 -->
        <div :style="[styleA, styleB]">我的style是通过多个计算属性形成的数组集合得到的</div>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: "#app",
            data: {
                fontSize: 66,
                styles: {
                        // 这里的red一定要写成字符串，不能不加引号，这是js中，不是
```

```
css里
                        color: 'red',
                        fontSize: 24 + 'px'
                    }
            },
            computed: {
                styleA: function () {
                    return {
                        color: 'blue'
                    }
                },
                styleB: function () {
                    return {
                        fontSize: 20 + 'px'
                    }
                }
            }
        });
    </script>
</body>
</html>
```

8. v-cloak，不需要值，配合display: none使用，解决js未加载完毕时页面显示{{ message }}这种类型的插值文本问题。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>010-内置指令</title>
    <style>
        [v-cloak] {
            display: none;
        }
    </style>
</head>
<body>
    <div id="app">
        <!-- js加载完之前，被标记为v-cloak，通过css设置为隐藏 -->
        <!-- 当js加载完之后，v-cloak标志会被移除，使其显示出来 -->
        <div v-cloak>{{ message }}</div>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: "#app",
            data: {
                message: "这是一段文本"
            }
        });
    </script>
</body>
```

```
</html>
```

【注】后面进阶篇将介绍 webpack和vue-router时，项目的HTML 结构只有一个空的div元素，剩余的内容都是由路由去挂载不同组件完成的，所以不再需要 v-cloak。

9. v-once，不需要值，作用是定义它的元素或组件只渲染一次，包括元素或组件的所有子节点。首次渲染后，不再随数据的变化重新渲染，将被视为静态内容。v-once在业务中也很少使用，当你需要进一步优化性能时，可能会用到。

10. v-if、v-else、v-else-if

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>012-v-if</title>
    <style>
        p, div, input {
            display: inline-block;
        }
    </style>
</head>
<body>
    <div id="app">
        <!-- 当不设置key值，在切换输入类型时，placeholder属性能够正常切换，但是如果用户已经输入文本内容，并不能被清空 -->
        <!-- 这是由于Vue复用之前渲染的input导致的，为每个文本框加上key值之后，就可以防止被复用 -->
        <div v-if="type === 'username'" key="username-input-key">
            <p>用户名: </p>
            <input type="text" placeholder="请输入用户名">
        </div>
        <div v-else-if="type === 'email'" key="email-input-key">
            <p>邮 箱: </p>
            <input type="text" placeholder="请输入邮箱">
        </div>
        <div v-else>
            <p>手 机: </p>
            <input type="text" placeholder="请输入手机号" key="mobile-input-key">
        </div>
        <input @click="toggleType" type="button" value="切换输入类型">
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: "#app",
            data: {
```

```
                type: 'username'
            },
            methods: {
                toggleType: function () {
                    if (this.type === 'username') {
                        this.type = 'email';
                    } else if (this.type === 'email') {
                        this.type = 'mobile';
                    } else {
                        this.type = 'username';
                    }
                }
            }
        });
    </script>
</body>
</html>
```

11. v-show，用法与v-if基本一致，只不过v-show是改变元素的css属性display。当v-show的值为true时，当前节点的display属性为显示属性；当v-show的值为false时，当前节点的display属性为none，展示未隐藏。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>011-v-show</title>
</head>
<body>
    <div id="app">
        <input @click="onClickButton(0)" type="button" value="路飞">
        <input @click="onClickButton(1)" type="button" value="索隆">
        <input @click="onClickButton(2)" type="button" value="山治">
        <div v-show="index === 0">{{ lufei }}</div>
        <div v-show="index === 1">{{ suolong }}</div>
        <div v-show="index === 2">{{ shanzhi }}</div>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: "#app",
            data: {
                lufei: "蒙奇·D·路飞，日本漫画《航海王》的主角，外号“草帽”路飞，草帽海贼团、草帽大船团船长，极恶的世代之一。橡胶果实能力者，悬赏金15亿贝里。梦想是找到传说中的One Piece，成为海贼王。",
                suolong: "罗罗诺亚·索隆。人称：海贼猎人。日本漫画《海贼王》及衍生作品中的角色。“草帽一伙”的战斗员 [1] ，是使用三把刀战斗的三刀流刀客。两年前集结香波地群岛的被称之为“极恶的世代”的十一超新星之一，在超新星中赏金排名第十位。",
                shanzhi: "山治，日本漫画《航海王》及衍生作品中的角色。草帽一伙厨师，金发，有着卷曲眉毛，永远遮住半边脸的家伙，香烟不离口，最爱女人，很花心但很有风度，海贼中的
```

```
绅士。",
                index: 0
            },
            methods: {
                onClickButton: function (param) {
                    this.index = param;
                }
            }
        });
    </script>
</body>
</html>
```

【注v-if和v-show的选择】v-if和v-show具有类似的功能，不过v-if才是真正的条件渲染，它会根据表达式适当地销毁或重建元素及绑定的事件或子组件。若表达式初始值为 false，则一开始元素/组件并不会渲染，只 有当条件第一次变为真时才开始编译。而 v-show 只是简单的css 属性切换，无论条件真与否，都会被编译。所以，v-if更适合条件不经常改变的场景，因为它切换开销相对较大，而 v-show适用于频繁切换条件。

12. v-for

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>013-v-for</title>
</head>
<body>
    <div id="app">
        <!-- v-for for integer -->
        <div v-for="number in 10">{{ number }}</div>
        <!-- v-for for array -->
        <div v-for="friend in friends">{{ friend }}</div>
        <!-- v-for for object, (value, key), not (key, value) -->
        <div v-for="(value, key) in me">{{ key }} : {{ value }}</div>

        <!-- todo list demo -->
        <input v-model="preTodo" type="text" placeholder="请输入一项待办事
项">
        <input @click="submitTodo" type="button" value="提交">
        <ul>
            <!-- v-for="todo in todos" || v-for="(todo, index) in todos"
-->
            <li v-for="(todo, index) in todos">{{ index }} - {{ todo }}</
li>
        </ul>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        var app = new Vue({
```

```
            el: "#app",
            data: {
                friends: ["Zhang", "Wang", "Lee", "Lou"],
                me: { name:"Andy", age:30, height:176.0 },
                preTodo: "",
                todos: []
            },
            methods: {
                submitTodo: function () {
                    if (this.preTodo.length > 0) {
                        this.todos.push(this.preTodo);
                        this.preTodo = "";
                    }
                }
            }
        });
    </script>
</body>
</html>
```

【注】数组更新时，请调用数组的方法进行更新，以下变动的数组中， Vue 是不能检测到的，也不会触发视图更新：

· 通过索引直接设置项，比如 app.todos[3] = "......"； ==> 应当这样 Vue.set(app.todos, 3, "......"); 或者更通用的方法 app.todos.splice(3, 1, "......");

· 修改数组长度，比如 app.todos.length= 1; ==> 应该这样 app.todos.splice(1);

【常用数组更新方法】

改变当前数组的方法：

· push()

· pop()

· shift() // 用于把数组的第一个元素从其中删除，并返回第一个元素的值。

· unshift() // 向数组的开头添加一个或更多元素,并返回新的长度。

· splice() // 从数组中添加/删除项目，然后返回被删除的项目。 arrayObject.splice(index,howmany,item1,.....,itemX)，index 整数，规定添加/删除项目的位置，使用负数可从数组结尾处规定位置。howmany 要删除的项目数量。如果设置为 0，则不会删除项目。item1, ..., itemX 可选。向数组添加的新项目。

· sort()

· reverse()

产生新数组的方法：

· filter()

· concat()

· slice()

13. 修饰符

@绑定的事件后加小圆点“.”，再跟一个后缀来使用修饰符。 Vue支持以下修饰符:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>0</title>
</head>
<body>
    <div id="app">
        <!-- 阻止单击事件冒泡 -->
        <a @click.stop="handle" href="http://www.4399.com"
target="blank">43999999999999</a>
        <!-- 提交事件不再重载页面 -->
        <form @submit.prevent="handle"></form>
        <!-- 修饰符可以串联 -->
        <a @click.stop.prevent="handle" href="http://www.4399.com"
target="blank">43999999999999</a>
        <!-- 只有修饰符 -->
        <form @submit.prevent></form>
        <!-- 添加事件侦听器时使用事件捕获模式 -->
        <div @click.capture="handle">这里可以点击</div>
        <!-- 只当事件在该元素本身(而不是子元素) 触发时触发回调 -->
        <div @click.self="handle">这里也可以点击</div>
        <!-- 只触发一次，组件同样适用 -->
        <div @click.once="handle">这里还可以点击</div>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: "#app",
            methods: {
                handle: function () {
                    alert('In handel.');
                }
            }
        });
    </script>
</body>
</html>
```

【键盘按键修饰符】

- .enter
- .tab
- .delete (捕获“删除”和“退格”键)
- .esc

- .space
- .up
- .down
- .left
- .right
- .ctrl
- .alt
- .shift
- .meta (Mac 下是 Command 键，Windows 下是窗口键)

14. 购物车实战（见代码）

15. 表单与v-model

v-model可与input、textarea、select进行数据双向绑定

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>015-v-model与表单</title>
</head>
<body>
    <div id="app">
        <div>
            <!-- v-model与value配合使用，实现互斥单选 -->
            <input type="radio" v-model="checkedUniversity"
value="Tsinghua">
            <label for="Tsinghua">清华</label>
            <br>
            <input type="radio" v-model="checkedUniversity"
value="Peking">
            <label for="Peking">北大</label>
            <br>
            <input type="radio" v-model="checkedUniversity"
value="Zhejiang">
            <label for="Zhejiang">浙大</label>
        </div>
        <div>
            <!-- v-model与value配合使用，实现复选 -->
            <input type="checkbox" v-model="foods" value="steak">
            <label for="steak">牛排</label>
            <br>
            <input type="checkbox" v-model="foods" value="soup">
            <label for="soup">汤</label>
            <br>
```

```
            <input type="checkbox" v-model="foods" value="vegetable">
            <label for="vegetable">蔬菜</label>
        </div>
        <div>
            <!-- v-model绑定属性，实现下拉菜单功能 -->
            <!-- <option>如果含有value属性，v-model会优先匹配value的值；如果没有，就会直接匹配<option>的text。 -->
            <!-- 比如选中第二项时，selected的值是28，而不是“二十八岁” -->
            <select v-model="selectedAge">
                <option>十八岁</option>
                <option value="28">二十八岁</option>
                <option>三十八岁</option>
            </select>
        </div>

        <!-- 修饰符 -->
        <!-- .lazy，lazyMessage并不是实时改变的，而是在失焦或按回车时才更新。 -->
        <div>
            <input type="text" v-model.lazy="lazyMessage">
            <p>{{ lazyMessage }}</p>
        </div>
        <!-- .trim，可以自动过滤输入的首尾空格（input框中首尾依然可以输入空格，但是trimMessage是去空格的） -->
        <div>
            <input type="text" v-model.trim="trimMessage">
            <p>{{ trimMessage }}</p>
        </div>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: "#app",
            data: {
                checkedUniversity: "Tsinghua",
                foods: [],
                selectedAge: 28,
                lazyMessage: "",
                trimMessage: ""
            }
        });
        setInterval(function () {
            console.log(app.checkedUniversity);
            console.log(app.foods);
            console.log(app.selectedAge);
        },3000);
    </script>
</body>
</html>
```

16. 组件

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>016-组件</title>
</head>
<body>
    <div id="app">
        <my-component-1></my-component-1>
        <my-component-2></my-component-2>
        <!-- 以下这种写法是不可行的，虽然div内容在页面上展示，但是调试可以看到组件内容被渲染到了table标签外层 -->
        <!-- html规定table标签内规定只允许是<tr>、<td>、<th>等这些表格元素 -->
        <table>
            <my-component-1></my-component-1>
        </table>
        <!-- 解决办法如下：使用特殊的is属性来挂载组件 -->
        <!-- tbody在渲染时，会被替换为组件的内容。常见的限制元素还有<ul>、<ol>、<select>  -->
        <table>
            <tbody is="my-component-1"></tbody>
        </table>
        <my-component-3></my-component-3>
        <my-component-3></my-component-3>
        <my-component-3></my-component-3>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        // 【1】全局注册组件
        Vue.component('my-component-1', {
            // template的DOM结构必须被一个元素包含，如果直接写成“这里是组件的内容”，不带“<div></div>”是无法渲染的
            template: "<div>组件内容</div>"
        });
        var app = new Vue({
            el: "#app",
            components: {
                // 【2】局部注册组件
                'my-component-2': {
                    template: "<div>这是一个局部注册的组件</div>"
                },
                'my-component-3': {
                    template: "<div><button @click='counter+='>{{ counter }}</button></div>",
                    // data必须是函数，然后将数据return出去
                    data: function () {
                        return {
                            counter: 0
```

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

17.props 【这里需要考虑明白一个问题：什么是父组件，什么是子组件？哪里是父组件的作用域，哪里是子组件的作用域？】

父组件的模板中包含子组件，父组件要正向地向子组件传递数据或参数，子组件接收到后根据参数的不同来渲染不同的内容或执行操作。这个正向传递数据的过程就是通过props来实现的。

【注】正向传递，由使用方给被使用方传递数据，类似iOS里面A页面present出B页面，A页面给B页面传递参数的过程，所以这个props推测是properties的简写形式。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>017-props</title>
</head>
<body>
    <div id="app">
        <!-- 参数property被绑定了父级superProperty属性 -->
        <my-component-1 :property="superProperty"></my-component-1>
        <!-- 使用v-bind绑定之后的属性为变量，是具有类型的，不使用v-bind绑定的属性为字符串常量 -->
        <my-component-2 name="Andy" :age="30" :propB="22"></my-component-2>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        Vue.component('my-component-1', {
            // 声明使用该组件需要传过来的参数
            props: ['property'],
            template: "<div>{{ property }}</div>"
        });
        Vue.component('my-component-2', {
            // props除了字符串数组，还可以是对象类型，而且更推荐这种写法，它可以进行数据类型验证
            props: {
                name: String,
                age: Number,
                sex: Boolean,
                propA: [String, Number], // 必须是字符串或数字类型
                propB: {            // 必须是数字类型，且必传
```

```
                    type: Number,
                    required: true
                },
                propC: {              // 必须是Boolean类型，缺省值为true
                    type: Boolean,
                    default: true
                },
                propD: {             // 当给数组或对象类型值提供缺省值时，必须通过函数返回
                    type: Object,
                    default: function () {
                        return {};
                    }
                },
                propE: {             // 可以自定义合法性验证函数
                    validator: function (value) {
                        return value > 10;
                    }
                }
            },
            template: "<div>name:{{name}}-age:{{computedAge}}-sex:{{sex}}-A:{{propA}}-B:{{propB}}-C:{{propC}}-D:{{propD}}-E:{{propE}}</div>",
            computed: {
                computedAge: function () {
                    console.log(typeof this.age);
                    return this.age + 3;
                }
            }
        });
        var app = new Vue({
            el: "#app",
            data: {
                superProperty: "这是我要传过去的参数"
            }
        });
    </script>
</body>
</html>
```

18. emit

子组件向父组件传参，反向传值。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>018-emit实现子组件向父组件传参-反向传值</title>
</head>
<body>
    <div id="app">
        <!-- 父组件用v-on:increase和v-on:reduce进行increse和reduce事件监听 -->
```

```
        <my-component-1 @increase="handelChange" @reduce="handelChange"></
my-component-1>
        <!-- v-model一般被用在input、textarea、select等地方 -->
        <!-- 用在组件中时，其双向绑定的属性为组件最外层标签的value值 -->
        <!-- 组件按钮点击，改变了最外层标签的value值，inputTotal的值通过v-model双向
绑定，也发生了改变 -->
        <my-component-2 v-model="inputTotal"></my-component-2>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        Vue.component('my-component-1', {
            template: "\
            <div>\
                <button @click='onIncrease'>+1</button>\
                <button @click='onReduce'>-1</button>\
            </div>\
            ",
            data: function () {
                return {
                    counter: 0
                }
            },
            methods: {
                onIncrease: function () {
                    this.counter++;
                    // $emit()触发事件
                    // $emit()方法的第一个参数是自定义事件的名称，后面的参数都是要传
递的数据，可以不填或填写多个。
                    this.$emit('increase', this.counter);
                },
                onReduce: function () {
                    this.counter--;
                    this.$emit('reduce', this.counter);
                }
            }
        });
        Vue.component('my-component-2', {
            template:  "\
            <div>\
                <button @click='onIncrease'>input +1</button>\
            </div>\
            ",
            data: function () {
                return {
                    counter: 0
                }
            },
            methods: {
                onIncrease: function () {
                    this.counter++;
                    // 触发input事件，改变的是最外层template外层div的value属性，
```

```
value值被改变为this.counter
                    this.$emit('input', this.counter);
                }
            }
        });
        var app = new Vue({
            el: '#app',
            data: {
                inputTotal: 0
            },
            methods: {
                handelChange: function (change) {
                    console.log(change);
                    console.log(this.inputTotal);
                }
            }
        });
    </script>
</body>
</html>
```

19. bus

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>019-bus</title>
</head>
<body>
    <div id="app">
        {{ message }}
        <my-component-1></my-component-1>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        // 空Vue实例，将会作为消息中间人
        var bus = new Vue();
        // 注册组件
        Vue.component('my-component-1', {
            template: '<button @click="handleEvent">点击传递事件</button>',
            methods: {
                handleEvent: function () {
                    // 通过消息中间人提交onMessage消息，并且传参
                    bus.$emit('onMessage', '来自组件my-component-1的内容');
                }
            }
        });
        // 主app对象
        var app = new Vue({
            el: "#app",
            data: {
```

```
                message: '初始文本内容'
            },
            mounted: function () {
                var _this = this;
                // 在app初始化时，通过消息中间人注册监听onMessage消息
                bus.$on('onMessage', function (msg) {
                    _this.message = msg;
                });
            }
        });
        // 【注】这时候我们发现Vue这个函数，或者更好理解的说，Vue这个类有很多作用，目前我们用到的作用有：
        // 1.通过选择器绑定一个节点，对该节点下的子节点、孙子节点的数据和节点属性等进行全局管理与事件处理
        // 2.通过类方法Vue.component()进行组件注册
        // 3.作为消息bus使用，常用方法bus.$emit()、bus.$on()
        // 总结来说，Vue这个函数是Vue.js的核心，功能都围绕这一个函数展开
    </script>
</body>
</html>
```

20. 父子链

在子组件中使用this.$parent.A可直接访问父组件的A属性；在父组件中使用this.$children访问它的所有子组件，也可以通过ref标签属性给子组件“取个名”，如<my-component ref="part1"></my-component>，这时候，在父组件中使用this.$refs.part1.B就可以直接访问part1子组件的B属性。

不过父子链形式的通信，耦合性很高，并不推荐使用，这里也不做代码示例了。

21. slot（插槽）

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>020-slot</title>
</head>
<body>
    <div id="app">
        <!-- 【1】slot的基本使用 -->
        <my-component-1>
            <p>内容1</p>
            <p>内容2</p>
        </my-component-1>
        <!-- 【2】slot根据名称能够对应插入 -->
        <my-component-2>
            <p slot="header">slot为header的内容</p>
```

```
            <p>没有名称的内容</p>
            <p slot="footer">slot为footer的内容</p>
        </my-component-2>
        <!-- 【3】通过scope声明作用域，通过作用域可以访问作用域下的变量 -->
        <my-component-3>
            <template scope="child">{{ child.childMessage }} --
{{ child.someData }}</template>
        </my-component-3>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        // 全局定义组件
        Vue.component('my-component-1', {
            // 一个slot就是一个“插槽”，父组件中的所有内容，作为一项元素插入一个插槽
            // 下例中有2个插槽，父组件中的内容1和内容2作为一个元素，会被插入两次
            template: '\
                <div>\
                    <slot>\
                        <p>如果父组件没有提供内容，这里是默认实现1</p>\
                    </slot>\
                    <slot>\
                        <p>如果父组件没有提供内容，这里是默认实现2</p>\
                    </slot>\
                </div>',
        });
        // 带有名称的插槽
        Vue.component('my-component-2', {
            template: '\
                <div>\
                    <slot name="header">\
                        <p>如果父组件没有提供内容，这里是默认实现-header</p>\
                    </slot>\
                    <slot>\
                        <p>如果父组件没有提供内容，这里是默认实现</p>\
                    </slot>\
                    <slot name="footer">\
                        <p>如果父组件没有提供内容，这里是默认实现-footer</p>\
                    </slot>\
                </div>',
            mounted: function () {
                // 【4】可以通过this.$slots访问到slot
                var header = this.$slots.header;
                console.log(header[0].elm.innerHTML);
            }
        });
        // 作用域插槽
        // 作用域插槽可以实现子组件给父组件传参
        Vue.component('my-component-3', {
            template: '\
```

```
                <div>\
                    <slot childMessage="这是子组件的消
息" :someData="someData">\
                        <p>如果父组件没有提供内容，这里是默认实现</p>\
                    </slot>\
                </div>\
                ',
            data: function () {
                return {
                    someData: "这是作用域插槽里的someData属性"
                }
            }
        });
        // 主要app对象
        var app = new Vue({
            el: "#app"
        });
    </script>
</body>
</html>
```

22. 递归组件
23. 动态组件
24. 异步组件

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>021-组件的高级用法</title>
</head>
<body>
    <div id="app">
        <h1>递归组件: </h1>
        <!-- 【1】递归组件 -->
        <recursive-component :count="1"></recursive-component>
        <h1>动态组件: </h1>
        <!-- 【2】动态组件 -->
        <!-- 关键点: 1.component泛型组件标签 2.is属性绑定组件名称 -->
        <component :is="currentComponent"></component>
        <button @click="handleClick('A')">切换到A组件</button>
        <button @click="handleClick('B')">切换到B组件</button>
        <button @click="handleClick('C')">切换到C组件</button>
        <h1>异步组件: </h1>
        <!-- 【3】异步组件 -->
        <asynchronize-component></asynchronize-component>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
```

```
    <script type="text/javascript">
        Vue.component('recursive-component', {
            name: 'recursive-component',
            props: {
                count: {
                    type: Number,
                    default: 1
                }
            },
            // template中通过添加自己标签，实现递归，一定要注意递归结束条件控制好
            template: '\
                <div class="child">这里是被递归的内容\
                    <recursive-component :count="count + 1" v-if="count <
3">\
                    </recursive-component>\
                </div>\
                '
        });
        Vue.component('asynchronize-component', function (resolve, reject)
{
            // 模拟异步获取组件内容
            window.setTimeout(function () {
                // 获取成功，调用resolve方法进行组件内容回调
                resolve({
                    template: '<div>我是异步渲染的组件</div>'
                });
                // 获取失败，调用reject方法进行原因回调
                //reject('reason');
            }, 3000);
        });
        var app = new Vue({
            el: "#app",
            components: {
                componentA: {
                    template: "<div>组件A</div>"
                },
                componentB: {
                    template: "<div>组件B</div>"
                },
                componentC: {
                    template: "<div>组件C</div>"
                }
            },
            data: {
                currentComponent: 'componentA'
            },
            methods: {
                handleClick: function (param) {
                    this.currentComponent = 'component' + param;
                }
            }
```

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

25. 内联模板 inline-template（不推荐使用）

26. $nextTick

假设一个节点在开始的时候是v-if隐藏的（隐藏的时候没有被创建），点击按钮后设置其显示。如果我们希望获取该节点的属性，在它隐藏的时候是获取不到的（因为它还没有被创建）。那么我们自然就想到，先修改v-if对应的Boolean属性，再获取我们需要的数据。但是这样依然是不可以的。

因为Vue本质上并不是实时更新DOM元素的，而是会开启一个任务队列，缓冲在同一事件循环中发生的所有数据改变。在缓冲时会去除重复数据，从而避免不必要的计算和 DOM 操作。然后，在下一个事件循环 tick中，Vue 刷新队列并执行实际已去重的工作。所以如果你用一个for循环来动态改变数据 100次，其实它只会应用最后一次改变，如果没有这种机制，DOM 就要重绘100次，这固然是一个很大的开销。

这时候我们就需要$nextTick，它标志着之前的DOM操作已经执行完毕。

【注】这有点类似于RunLoop机制

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>022-nextTick</title>
</head>
<body>
    <div id="app">
        <div id="testElement" v-if="isShow">我开始的时候是被隐藏的哦</div>
        <button @click="handleClick">点击显示被隐藏的内容</button>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        var app = new Vue({
            el: "#app",
            data: {
                isShow: false
            },
            methods: {
                handleClick: function () {
                    this.isShow = !this.isShow;
                    // 直接获取会报错 TypeError: Cannot read property
'innerHTML' of null
                    // var testElement =
document.getElementById("testElement");
                    // console.log(testElement.innerHTML);
```

```
                    this.$nextTick(function () {
                        var testElement =
document.getElementById("testElement");
                        console.log(testElement.innerHTML);
                    });
                }
            }
        });
    </script>
</body>
</html>
```

27. 手动挂载

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>023-手动挂载Vue实例</title>
</head>
<body>
    <div id="app">{{ message }}</div>
    <div id="delay-mounted"></div>
    <div id="delay-mounted-extend"></div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        // 创建Vue实例，并且初始化挂载到#app节点上
        var app = new Vue({
            el: "#app",
            data: {
                message: "我是普通挂载（创建Vue对象初始化挂载）"
            }
        });
        // 手动挂载
        var delayMounted = new Vue({
            template: "<div>{{ message }}</div>",
            data: function () {
                return {
                    message: "我是被延迟挂载的消息1"
                }
            }
        });
        delayMounted.$mount("#delay-mounted");
        // 手动挂载，extend，《Vue.js实战》示例
        // 【注，未经考证】个人觉得这是对Vue的子类化，通过extend方法，创建Vue的一个子
类（js中没有类的话，那就是继承链子元素）
        var MyCom = Vue.extend({
            template: '<div>Hello: {{ message }}</div>',
            data: function () {
                return {
                    message: "我是被延迟挂载的消息2"
```

```
                }
            }
        });
        var myVue = new MyCom();
        myVue.$mount("#delay-mounted-extend");
    </script>
</body>
</html>
```

28. 自定义指令

自定义指令的生命周期函数：

· bind: 只调用一次，指令第一次绑定到元素时调用，用这个钩子函数可以定义一个在绑定时执行一次的初始化动作。

· inserted: 被绑定元素插入父节点时调用（父节点存在即可调用，不必存在于document中）。

· update: 被绑定元素所在的模板更新时调用，而不论绑定值是否变化。通过比较更新前后的绑定值，可以忽略不必要的模板更新。

· componentUpdated: 被绑定元素所在模板完成一次更新周期时调用。

· unbind: 只调用一次，指令与元素解绑时调用 。

每个生命周期函数都有几个可用的参数：

· el：指令所绑定的元素，可以用来直接操作DOM。

· binding：一个对象，包含以下属性：

name – 指令名，不包括 v-前缀。

value – 指令的绑定值，例如v-my-directive="1+1"，value的值是2。

oldValue – 指令绑定的前一个值，仅在update和componentUpdated钩子中可用，无论值是否改变都可用。

expression – 绑定值的字符串形式，例如v-my-directive="1+1"，expression的值是'1+1'。

arg – 传给指令的参数，例如v-my-directive:foo，arg的值是foo。

modifiers – 一个包含修饰符的对象，例如v-my-directive.foo.bar，修饰符对象modifiers的值是{ foo: true, bar: true }。

· vnode：Vue 编译生成的虚拟节点，在进阶篇中介绍。

· oldVnode：上一个虚拟节点仅在update和componentUpdated 钩子中可用 。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>024-自定义指令</title>
</head>
<body>
```

```
    <div id="app">
        <input type="text" placeholder="请输入些什么..." v-my-directive-1>
        <my-component></my-component>
    </div>
    <script type="text/javascript" src="../js/vue2.6.6.min.js"></script>
    <script type="text/javascript">
        // 【1】目的：希望对DOM进行底层的操作
        // 【2】用法：跟注册组件很类似，分为全局注册和局部注册
        // （1）全局注册一个指令
        Vue.directive('my-directive-1', {
            // 指令配置
            inserted: function (element) {
                element.focus();
            }
        });
        Vue.component('my-component', {
            template: '<div><input type="text" placeholder="组件让你输入些什么..." v-my-directive-2></div>',
            directives: {
                // （2）局部注册一个指令，只能在当前作用下使用
                // 【注】javascript对象的key都是字符串，就算是给的其它类型的值，也会自动通过toString()转换成字符串。
                // 这里可以不显示的写双引号，但是js里面不支持变量名中包含'-'，可以用驼峰式命名
                'my-directive-2': {
                    // 指令配置
                    inserted: function (element) {
                        element.focus();
                    }
                }
            }
        });
        var app = new Vue({
            el: "#app"
        });
    </script>
</body>
</html>
```

二、进阶篇

1. Render函数

Render函数通过createElement参数来创建 Virtual Dom。（其实就是用js动态创建template，性能比template更高。）（如果使用了webpack做编译，template都会被预编译为render函数。）

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>001-Render函数</title>
</head>
<body>
    <div id="app">
        <my-component-1></my-component-1>
    </div>
    <script type="text/javascript" src="../js/
vue2.6.6.min.js"></script>
    <script type="text/javascript">
        // Render函数
        /*
            createElement(
                // String | Object | Function
                //一个 HTML 标签，组件选项，或一个函数
                //必须 Return 上述其中一个
                'div',
                // Object
                // 一个对应属性的数据对象，可选
                //您可以在 template 中使用
                {
                    //和 v-bind:class 一样的 API
                    'class' : {
                        foo: true,
                        bar: false
                    },
                    //和 v-bind:style 一样的 API
                    style : {
```

```
                    color: 'red',
                },
                //正常的 HTML特性
                attrs: {
                    id : 'foo'
                },
                //组件 props
                props: {
                    myProp: 'bar'
                },
                // DOM属性
                domProps: {
                    innerHTML : 'baz'
                },
                //自定义事件监听器”on”
                //不支持如 v-on :keyup.enter 的修饰器
                //需要手动匹配 keyCode
                on: {
                    click : this.clickHandler
                },
                //仅对于组件，用于监听原生事件
                //而不是组件使用 vm. $emit 触发的自定义事
件
                nativeOn: {
                    click: this.nativeClickHandler
                },
                //自定义指令
                directives: [
                    {
                        name : 'my-custom -
directive',
                        value : '2',
                        expression : '1 + 1'
                        arg: 'foo',
                        modifiers : {
                            bar: true
                        }
```

```
                    }
                ],
                // 作用域 slot
                // { name: props => VNode |
Array<VNode> }
                scopedSlots: {
                    default: props => h ('span',
props .text)
                }
            }
            // String | Array
            //子节点( VNodes ), 可选
            [
                createElement('hl', 'hello world'),
                createElement(MyComponent, {
                    props : {
                        someProp: 'foo'
                    }
                }),
                'bar'
            ]
        );
    */
    Vue.component('my-component-1', {
        render: function (createElement) {
            var _this = this;
            return createElement('input', {
                attrs: {
                    placeholder: '占位符'
                },
                style: {
                    width: '200px'
                },
                on: {
                    keyup: function (event) {
                        if (event.keyCode === 13) {
                            _this.test();
                        }
                    }
```

```
                    }
                });
            },
            methods: {
                test: function () {
                    alert(event.target.value);
                }
            }
        });
        var app = new Vue({
            el: "#app"
        });
    </script>
</body>
</html>
```

【注】Render函数的练习使用，见代码DemoSortableTable和DemoMessageList两块。

2. webpack

【webpack的作用】

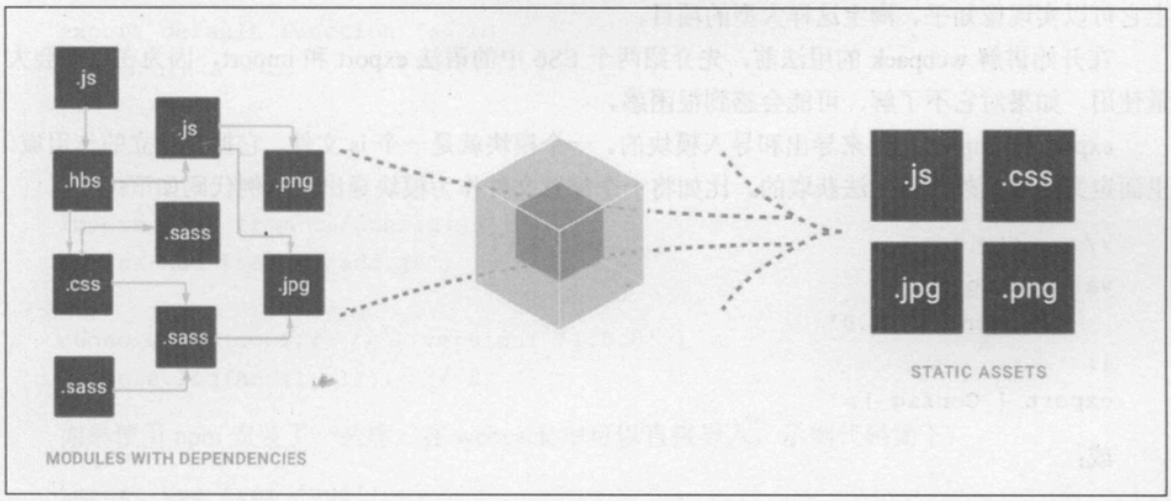

图 10-1　webpack 模块化示意图

【export和import是用来导出和导入模块的】
一个模块就是一个 js文件，它拥有独立的作用域， 里面定义的变量外部是无法获取的。比如将一个配置文件作为模块导出，示例代码如下：

```
// config .js
var Config = {
    version : '1. 0. 0'
}
export { Config };
// config.js
export var Config = {
    version : '1. 0. 0'
}
// add.js
export function add(a , b) {
    return a + b;
}
```

模块导出后，在需要使用模块的文件使用 import 再导入，就可以在这个文件内使用这些模块了。示例代码如下:

```
// main.js
import { Config } from './config.js'
import { add } from './add.js';
console.log(Config); // {version:'1.0.0'}
console .log(add(1, 1)); // 2
```

以上几个示例中，导入的模块名称都是在 export 的文件中设置的，也就是说用户必须预先知道这个名称叫什么，比如 Config、add。而有的时候，用户不想去了解名称是什么，只是把模块的功能拿来使用，或者想自定义名称，这时可以使用 export default来输出默认的

模块。示例代码如下:

```
// config.js
export default {
    version : '1. 0. 0'
}
// add.js
export default function (a, b) {
    return a + b;
}
// main.js
// import 自定义名称 from 'js路径'
import conf from './config.js';
import Add from './add.js';
console.log(Config); // {version:'1.0.0'}
console .log(add(1, 1)); // 2
```

【注】webpack示例代码见文件夹webpackdemo

【注】webpack只不过是一个js配置文件，入口(Entry)、 出口(Output)、加载器(Loaders)和插件(Plugins)是它的核心概念

3. 单文件组件与vue-loader

顾名思义，.vue单文件组件就是一个后缀名为.vue的文件，在webpack中使用vue-loader就可以对.vue格式的文件进行处理。

一个.vue文件一般包含3部分，<template>、<script>和<style>。

4. 前端路由与vue-router

介绍了通过 is 特性来实现动态组件的方法。vue-router 的实现原理与之类似，路由不同的页面事实上就是动态加载不同的组件。

5. 状态管理与vuex

使用bus（中央事件总线）的方法，来触发和接收事件，起到通信的作用。Vuex所 解决的问题与bus类似，它作为Vue的一个插件来使用，可以更好地管理和维护整个项目的组件状态。