拓展：Vue.js 容易忽略的 API 详解

前面的 15 小节已经覆盖了 Vue.js 组件的绝大部分内容，但还是有一些 API 容易忽略。本节则对 Vue.js 的一些重要且易忽略的 API 进行详细介绍。

nextTick

nextTick 是 Vue.js 提供的一个函数，并非浏览器内置。nextTick 函数接收一个回调函数 cb，在下一个 DOM 更新循环之后执行。比如下面的示例：

```
<template>
  <div>
    <p v-if="show" ref="node">内容</p>
    <button @click="handleShow">显示</button>
  </div>
</template>
<script>
  export default {
    data () {
      return {
        show: false
      }
    },
    methods: {
      handleShow () {
        this.show = true;
        console.log(this.$refs.node);  // undefined
        this.$nextTick(() => {
          console.log(this.$refs.node);  // <p>内容</p>
        });
      }
    }
  }
</script>
```

当 show 被置为 true 时，这时 p 节点还未被渲染，因此打印出的是 undefined，而在 nextTick 的回调里，p 已经渲染好了，这时能正确打印出节点。

nextTick 的源码在 https://github.com/vuejs/vue/blob/dev/src/core/util/next-tick.js (https://github.com/vuejs/vue/blob/dev/src/core/util/next-tick.js)，可以看到，Vue.js 使用了 Promise、setTimeout 和 setImmediate 三种方法来实现 nextTick，在不同环境会使用不同的方法。

v-model 语法糖

v-model 常用于表单元素上进行数据的双向绑定，比如 <input>。除了原生的元素，它还能在自定义组件中使用。

v-model 是一个语法糖，可以拆解为 props: value 和 events: input。就是说组件必须提供一个名为 value 的 prop，以及名为 input 的自定义事件，满足这两个条件，使用者就能在自定义组件上使用 v-model。比如下面的示例，实现了一个数字选择器：

```
<template>
  <div>
    <button @click="increase(-1)">减 1</button>
    <span style="color: red;padding: 6px">{{
currentValue }}</span>
    <button @click="increase(1)">加 1</button>
  </div>
</template>
<script>
  export default {
    name: 'InputNumber',
    props: {
      value: {
        type: Number
      }
```

```
    },
    data () {
      return {
        currentValue: this.value
      }
    },
    watch: {
      value (val) {
        this.currentValue = val;
      }
    },
    methods: {
      increase (val) {
        this.currentValue += val;
        this.$emit('input', this.currentValue);
      }
    }
  }
</script>
```

props 一般不能在组件内修改，它是通过父级修改的，因此实现 v-model 一般都会有一个 currentValue 的内部 data，初始时从 value 获取一次值，当 value 修改时，也通过 watch 监听到及时更新；组件不会修改 value 的值，而是修改 currentValue，同时将修改的值通过自定义事件 input 派发给父组件，父组件接收到后，由父组件修改 value。所以，上面的数字选择器组件可以有下面两种使用方式：

```
<template>
  <InputNumber v-model="value" />
</template>
<script>
  import InputNumber from '../components/input-number/input-number.vue';

  export default {
    components: { InputNumber },
    data () {
      return {
        value: 1
      }
    }
  }
</script>
```

或：

```
<template>
  <InputNumber :value="value"
@input="handleChange" />
</template>
<script>
  import InputNumber from '../components/input-
number/input-number.vue';

  export default {
    components: { InputNumber },
    data () {
      return {
        value: 1
      }
    },
    methods: {
      handleChange (val) {
        this.value = val;
      }
    }
  }
</script>
```

如果你不想用 value 和 input 这两个名字，从 Vue.js 2.2.0 版本开始，提供了一个 model 的选项，可以指定它们的名字，所以数字选择器组件也可以这样写：

```
<template>
  <div>
    <button @click="increase(-1)">减 1</button>
    <span style="color: red;padding: 6px">{{
currentValue }}</span>
    <button @click="increase(1)">加 1</button>
```

```
  </div>
</template>
<script>
  export default {
    name: 'InputNumber',
    props: {
      number: {
        type: Number
      }
    },
    model: {
      prop: 'number',
      event: 'change'
    },
    data () {
      return {
        currentValue: this.number
      }
    },
    watch: {
      value (val) {
        this.currentValue = val;
      }
    },
    methods: {
      increase (val) {
        this.currentValue += val;
        this.$emit('number', this.currentValue);
      }
    }
  }
</script>
```

在 model 选项里，就可以指定 prop 和 event 的名字了，而不一定非要用 value 和 input，因为这两个名字在一些原生表单元素里，有其它用处。

.sync 修饰符

如果你使用过 Vue.js 1.x，一定对 .sync 不陌生。在 1.x 里，可以使用 .sync 双向绑定数据，也就是父组件或子组件都能修改这个数据，是双向响应的。在 Vue.js 2.x 里废弃了这种用法，目的是尽可能将父子组件解耦，避免子组件无意中修改了父组件的状态。

不过在 Vue.js 2.3.0 版本，又增加了 .sync 修饰符，但它的用法与 1.x 的不完全相同。2.x 的 .sync 不是真正的双向绑定，而是一个语法糖，修改数据还是在父组件完成的，并非在子组件。

仍然是数字选择器的示例，这次不用 v-model，而是用 .sync，可以这样改写：

```
<template>
  <div>
    <button @click="increase(-1)">减 1</button>
    <span style="color: red;padding: 6px">{{
value }}</span>
    <button @click="increase(1)">加 1</button>
  </div>
</template>
<script>
  export default {
    name: 'InputNumber',
    props: {
      value: {
        type: Number
      }
    },
    methods: {
      increase (val) {
        this.$emit('update:value', this.value +
val);
      }
    }
  }
</script>
```

用例：

```
<template>
  <InputNumber :value.sync="value" />
</template>
<script>
  import InputNumber from '../components/input-number/input-number.vue';

  export default {
    components: { InputNumber },
    data () {
      return {
        value: 1
      }
    }
  }
</script>
```

看起来要比 v-model 的实现简单多，实现的效果是一样的。v-model 在一个组件中只能有一个，但 .sync 可以设置很多个。.sync 虽好，但也有限制，比如：

- **不能**和表达式一起使用（如 v-bind:title.sync="doc.title + '!'" 是无效的）；
- **不能**用在字面量对象上（如 v-bind.sync="{ title: doc.title }" 是无法正常工作的）。

$set

在上一节已经介绍过 $set，有两种情况会用到它：

1. 由于 JavaScript 的限制，Vue 不能检测以下变动的数组：
    1. 当利用索引直接设置一个项时，例

如：this.items[index] = value;
2. 当修改数组的长度时，例如：vm.items.length = newLength。

2. 由于 JavaScript 的限制，**Vue 不能检测对象属性的添加或删除**。

举例来看，就是：

```
// 数组
export default {
  data () {
    return {
      items: ['a', 'b', 'c']
    }
  },
  methods: {
    handler () {
      this.items[1] = 'x';  // 不是响应性的
    }
  }
}
```

使用 $set：

```
// 数组
export default {
  data () {
    return {
      items: ['a', 'b', 'c']
    }
  },
  methods: {
    handler () {
      this.$set(this.items, 1, 'x');  // 是响应性的
    }
  }
}
```

以对象为例：

```
// 对象
export default {
  data () {
    return {
      item: {
        a: 1
      }
    }
  },
  methods: {
    handler () {
      this.item.b = 2;  // 不是响应性的
    }
  }
}
```

使用 $set：

```
// 对象
export default {
  data () {
    return {
      item: {
        a: 1
      }
    }
  },
  methods: {
    handler () {
      this.$set(this.item, 'b', 2);  // 是响应性的
    }
  }
}
```

另外，数组的以下方法，都是**可以**触发视图更新的，也就是响应性的：

push()、pop()、shift()、unshift()、splice()、sort()、

还有一种小技巧，就是先 copy 一个数组，然后通过 index 修改后，再把原数组整个替换，比如：

```
handler () {
  const data = [...this.items];
  data[1] = 'x';
  this.items = data;
}
```

计算属性的 set

计算属性（computed）很简单，而且也会大量使用，但大多数时候，我们只是用它默认的 get 方法，也就是平时的常规写法，通过 computed 获取一个依赖其它状态的数据。比如：

```
computed: {
  fullName () {
    return `${this.firstName} ${this.lastName}`;
  }
}
```

这里的 fullName 事实上可以写为一个 Object，而非 Function，只是 Function 形式是我们默认使用它的 get 方法，当写为 Object 时，还能使用它的 set 方法：

```
computed: {
  fullName: {
    get () {
      return `${this.firstName}
${this.lastName}`;
    },
    set (val) {
      const names = val.split(' ');
      this.firstName = names[0];
      this.lastName = names[names.length - 1];
    }
  }
}
```

计算属性大多时候只是读取用，使用了 set 后，就可以写入了，比如上面的示例，如果执行 this.fullName = 'Aresn Liang'，computed 的 set 就会调用，firstName 和 lastName 会被赋值为 Aresn 和 Liang。

剩余值得注意的 API

还有一些 API，可能不常用，也比较简单，只需知道就好，本册不详细展开介绍，可以通过指引到 Vue.js 文档查看。

delimiters (https://cn.vuejs.org/v2/api/#delimiters)

改变纯文本插入分隔符，Vue.js 默认的是 {{ }}，如果你使用其它一些后端模板，比如 Python 的 Tornado 框架，那 Vue.js 和 Tornado 的 {{ }} 就冲突了，这时用它可以修改为指定的分隔符。

v-once (https://cn.vuejs.org/v2/api/#v-once)

只渲染元素和组件**一次**。随后的重新渲染，元素/组件及其所有的子节点将被视为静态内容并跳过。这可以用于优化更新性能。

vm.$isServer (https://cn.vuejs.org/v2/api/#vm-isServer)

当前 Vue 实例是否运行于服务器，如果你的组件要兼容 SSR，它会很有用。

inheritAttrs (https://cn.vuejs.org/v2/api/#inheritAttrs)

一些原生的 html 特性，比如 `id`，即使没有定义 props，也会被集成到组件根节点上，设置 inheritAttrs 为 false 可以关闭此特性。

errorHandler (https://cn.vuejs.org/v2/api/#errorHandler)

使用 errorHandler 可以进行异常信息的获取。

watch (https://cn.vuejs.org/v2/api/#watch)

监听状态的变化，用的也很多了，但它和 computed 一样，也有 Object 的写法，这样能配置更多的选项，比如：

- handler 执行的函数
- deep 是否深度
- immediate 是否立即执行

完整的配置可以阅读文档。

comments (https://cn.vuejs.org/v2/api/#comments)

开启会保留 html 注释。

transition (https://cn.vuejs.org/v2/api/#transition)

内置的组件，可做过渡效果，比如 CSS 的高度从 0 到 auto（使用纯 CSS 是无法实现动画的）。

结语

彻底掌握一门语言（框架），不需要阅读它所有的源码，但至少要阅读它所有的 API (https://cn.vuejs.org/v2/api/)。