Vue 的构造器——extend 与手动挂载——$mount

本节介绍两个 Vue.js 内置但却不常用的 API——extend 和 $mount，它们经常一起使用。不常用，是因为在业务开发中，基本没有它们的用武之地，但在独立组件开发时，在一些特定的场景它们是至关重要的。

使用场景

我们在写 Vue.js 时，不论是用 CDN 的方式还是在 Webpack 里用 npm 引入的 Vue.js，都会有一个根节点，并且创建一个根实例，比如：

```
<body>
  <div id="app"></div>
</body>
<script>
  const app = new Vue({
    el: '#app'
  });
</script>
```

Webpack 也类似，一般在入口文件 main.js 里，最后会创建一个实例：

```
import Vue from 'vue';
import App from './app.vue';

new Vue({
  el: '#app',
  render: h => h(App)
});
```

因为用 Webpack 基本都是前端路由的，它的 html 里一般都只有一个根节点 ，其余都是通过 JavaScript 完成，也就是许多的 Vue.js 组件（每个页面也是一个组件）。

有了初始化的实例，之后所有的页面，都由 vue-router 帮我们管理，组件也都是用 import 导入后局部注册（也有在 main.js 全局注册的），不管哪种方式，组件（或页面）的创建过程我们是无需关心的，只是写好 .vue 文件并导入即可。这样的组件使用方式，有几个特点：

1. 所有的内容，都是在 #app 节点内渲染的；
2. 组件的模板，是事先定义好的；
3. 由于组件的特性，注册的组件只能在当前位置渲染。

比如你要使用一个组件 <i-date-picker>，渲染时，这个自定义标签就会被替换为组件的内容，而且在哪写的自定义标签，就在哪里被替换。换句话说，常规的组件使用方式，只能在规定的地方渲染组件，这在一些特殊场景下就比较局限了，例如：

1. 组件的模板是通过调用接口从服务端获取的，需要动态渲染组件；
2. 实现类似原生 window.alert() 的提示框组件，它的位置是在 <body> 下，而非 ，并且不会通过常规的组件自定义标签的形式使用，而是像 JS 调用函数一样使用。

一般来说，在我们访问页面时，组件就已经渲染就位了，对于场景1，组件的渲染是异步的，甚至预先不知道模板是什么。对于场景2，其实并不陌生，在 jQuery 时代，通过操作 DOM，很容易就能实现，你可以沿用这种思路，只是这种做法不那么 Vue，既然使用 Vue.js 了，就应该用 Vue 的思路来解决问题。对于这两种场景，Vue.extend 和 vm.$mount 语法就派上用场了。

用法

上文我们说到，创建一个 Vue 实例时，都会有一个选项 el，来指定实例的根节点，如果不写 el 选项，那组件就处于未挂载状态。Vue.extend 的作用，就是基于 Vue 构造器，创建一个“子类”，它的参数跟 new Vue 的基本一样，但 data 要跟组件一样，是个函数，再配合 $mount ，就可以让组件渲染，并且挂载到任意指定的节点上，比如 body。

比如上文的场景，就可以这样写：

```
import Vue from 'vue';

const AlertComponent = Vue.extend({
  template: '<div>{{ message }}</div>',
  data () {
    return {
      message: 'Hello, Aresn'
    };
  },
});
```

这一步，我们创建了一个构造器，这个过程就可以解决异步获取 template 模板的问题，下面要手动渲染组件，并把它挂载到 body 下：

```
const component = new AlertComponent().$mount();
```

这一步，我们调用了 $mount 方法对组件进行了手动渲染，但它仅仅是被渲染好了，并没有挂载到节点上，也就显示不了组件。此时的 component 已经是一个标准的 Vue 组件实例，因此它的 $el 属性也可以被访问：

```
document.body.appendChild(component.$el);
```

当然，除了 body，你还可以挂载到其它节点上。

$mount 也有一些快捷的挂载方式，以下两种都是可以的：

```
// 在 $mount 里写参数来指定挂载的节点
new AlertComponent().$mount('#app');
// 不用 $mount，直接在创建实例时指定 el 选项
new AlertComponent({ el: '#app' });
```

实现同样的效果，除了用 extend 外，也可以直接创建 Vue 实例，并且用一个 Render 函数来渲染一个 .vue 文件：

```
import Vue from 'vue';
import Notification from './notification.vue';

const props = {};  // 这里可以传入一些组件的 props 选项

const Instance = new Vue({
  render (h) {
    return h(Notification, {
      props: props
    });
  }
});

const component = Instance.$mount();
document.body.appendChild(component.$el);
```

这样既可以使用 .vue 来写复杂的组件（毕竟在 template 里堆字符串很痛苦），还可以根据需要传入适当的 props。渲染后，如果想操作 Render 的 Notification 实例，也是很简单的：

```
const notification = Instance.$children[0];
```

因为 Instance 下只 Render 了 Notification 一个子组件，所以可以用 $children[0] 访问到。

> 如果你还不理解这样做的目的，没有关系，后面小节的两个实战你会感受到它的用武之地。

需要注意的是，我们是用 $mount 手动渲染的组件，如果要销毁，也要用 $destroy 来手动销毁实例，必要时，也可以用 removeChild 把节点从 DOM 中移除。

结语

这两个 API 并不难理解，只是不常使用罢了，因为多数情况下，我们只关注在业务层，并使用现成的组件库。

使用 Vue.js 也有二八原则，即 80% 的人看过 Vue.js 文档教程篇 (https://cn.vuejs.org/v2/guide/)，20% 的人看过 Vue.js 文档 API (https://cn.vuejs.org/v2/api/)。

下一节，我们来做点有趣的东西。

扩展阅读

- 聊聊 Vue.js 的 template 编译 (https://juejin.im/post/59da1c116fb9a00a4a4cf6dd)