实战 3：动态渲染 .vue 文件的组件—— Display

你可能用过 jsfiddle (https://jsfiddle.net/) 或 jsbin (https://jsbin.com) 之类的网站，在里面你可以用 CDN 的形式引入 Vue.js，然后在线写示例，实时运行，比如下面这个例子：

https://jsfiddle.net/c87yh92v/ (https://jsfiddle.net/c87yh92v/)

不过，这类网站主要是一个 html，里面包含 js、css 部分，渲染侧是用 iframe 嵌入你编写的 html，并实时更新。在这些网站写示例，是不能直接写 .vue 文件的，因为没法进行编译。

再来看另一个网站 iView Run (https://run.iviewui.com/)（之前小节也有提到），它是能够在线编写一个标准的 .vue 文件，并及时渲染的，它也预置了 iView 环境，你可以使用 iView 组件库全部的组件。本小节，我们就来实现这样一个能够动态渲染 .vue 文件的 Display 组件，当然，用到的核心技术就是上一节的 extend 和 $mount。

接口设计

一个常规的 .vue 文件一般都会包含 3 个部分：

- <template>：组件的模板；
- <script>：组件的选项，不包含 el；
- <style>：CSS 样式。

回忆一下用 extend 来构造一个组件实例，它的选项 template 其实就是上面 <template> 的内容，其余选项对应的是 <script>，样式 <style> 事实上与 Vue.js 无关，我们可以先不管。这样的话，当拿到一个 .vue 的文件（整体其实是字符串），只需要把 <template>、<script>、<style> 使用正则分割，把对应的部分传递给 extend 创建的实例就可以。

Display 是一个功能型的组件，没有交互和事件，只需要一个 prop：code 将 .vue 的内容传递过来，其余工作都是在组件内完成的，这对使用者很友好。当然，你也可以设计成三个 props，分别对应 html、js、css，那分割的工作就要使用者来完成。出于使用者优先原则，苦活累活当然是在组件内完成了，因此推荐第一个方案。

实现

在 src/components 目录下创建 display 目录，并新建 display.vue 文件，基本结构如下：

```vue
<!-- display.vue -->
<template>
  <div ref="display"></div>
</template>
<script>
  export default {
    props: {
      code: {
        type: String,
        default: ''
      }
    },
    data () {
      return {
        html: '',
        js: '',
        css: ''
      }
    },
  }
</script>
```

父级传递 code 后，将其分割，并保存在 data 的 html、js、css 中，后续使用。

我们使用正则，基于 <> 和 </> 的特性进行分割：

```
// display.vue, 部分代码省略
export default {
  methods: {
    getSource (source, type) {
      const regex = new RegExp(`<${type}[^>]*>`);
      let openingTag = source.match(regex);

      if (!openingTag) return '';
      else openingTag = openingTag[0];

      return
source.slice(source.indexOf(openingTag) +
openingTag.length,
source.lastIndexOf(`</${type}>`));
    },
    splitCode () {
      const script = this.getSource(this.code,
'script').replace(/export default/, 'return ');
      const style = this.getSource(this.code,
'style');
      const template = '<div id="app">' +
this.getSource(this.code, 'template') + '</div>';

      this.js = script;
      this.css = style;
      this.html = template;
    },
  }
}
```

getSource 方法接收两个参数：

- source：.vue 文件代码，即 props: code；

- type：分割的部分，也就是 template、script、style。

分割后，返回的内容不再包含 <template> 等标签，直接是对应的内容，在 splitCode 方法中，把分割好的代码分别赋值给 data 中声明的 html、js、css。有两个细节需要注意：

1. .vue 的 <script> 部分一般都是以 export default 开始的，可以看到在 splitCode 方法中将它替换为了 return，这个在后文会做解释，当前只要注意，我们分割完的代码，仍然是字符串；
2. 在分割的 <template> 外层套了一个 ，这是为了容错，有时使用者传递的 code 可能会忘记在外层包一个节点，没有根节点的组件，是会报错的。

准备好这些基础工作后，就可以用 extend 渲染组件了，在这之前，我们先思考一个问题：上文说到，当前的 this.js 是字符串，而 extend 接收的选项可不是字符串，而是一个对象类型，那就要先把 this.js 转为一个对象。

不卖关子，来介绍 new Function 用法，先看个示例：

```
const sum = new Function('a', 'b', 'return a + b');

console.log(sum(2, 6)); // 8
```

new Function 的语法：

```
new Function ([arg1[, arg2[, ...argN]],] functionBody)
```

arg1, arg2, ... argN 是被函数使用的参数名称，**functionBody** 是一个含有包括函数定义的 JavaScript 语句的**字符串**。也就是说，示例中的字符串 return a + b 被当做语句执行了。

上文说到，this.js 中是将 *export default* 替换为 *return* 的，如果将 this.js 传入 new Function 里，那么 this.js 就执行了，这时因为有 return，返回的就是一个对象类型的 this.js 了。

如果你还不是很理解 new Function，可以到文末的扩展阅读进一步了解。除了 new Function，你熟悉的 eval 函数也可以使用，它与 new Function 功能类似。

知道了这些，下面的内容就容易理解了：

```
<!-- display.vue, 部分代码省略 -->
<template>
  <div ref="display"></div>
</template>
<script>
  import Vue from 'vue';

  export default {
    data () {
      return {
        component: null
      }
    },
    methods: {
      renderCode () {
        this.splitCode();

        if (this.html !== '' && this.js !== '') {
          const parseStrToFunc = new
Function(this.js)();

          parseStrToFunc.template =  this.html;
          const Component = Vue.extend(
```

```
parseStrToFunc );
          this.component = new
Component().$mount();


this.$refs.display.appendChild(this.component.$el
);
        }
      }
    },
    mounted () {
      this.renderCode();
    }
  }
</script>
```

extend 构造的实例通过 $mount 渲染后，挂载到了组件唯一的一个节点 上。

现在 html 和 js 都有了，还剩下 css。加载 css 没有什么奇技淫巧，就是创建一个 <style> 标签，然后把 css 写进去，再插入到页面的 <head> 中，这样 css 就被浏览器解析了。为了便于后面在 this.code 变化或组件销毁时移除动态创建的 <style> 标签，我们给每个 style 标签加一个随机 id 用于标识。

在 src/utils 目录下新建 random_str.js 文件，并写入以下内容：

```
// 生成随机字符串
export default function (len = 32) {
  const $chars =
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUV
WXYZ1234567890';
  const maxPos = $chars.length;
  let str = '';
  for (let i = 0; i < len; i++) {
    str += $chars.charAt(Math.floor(Math.random()
* maxPos));
  }
  return str;
}
```

不难理解，这个方法是从指定的 a-zA-Z0-9 中随机生成 32 位的字符串。

补全 renderCode 方法：

```
// display.vue, 部分代码省略
import randomStr from
'../../utils/random_str.js';

export default {
  data () {
    return {
      id: randomStr()
    }
  },
  methods: {
    renderCode () {
      if (this.html !== '' && this.js !== '') {
        // ...
        if (this.css !== '') {
          const style =
document.createElement('style');
          style.type = 'text/css';
          style.id = this.id;
          style.innerHTML = this.css;
          document.getElementsByTagName('head')
[0].appendChild(style);
        }
      }
    }
  }
}
```

当 Display 组件销毁时，也要手动销毁 extend 创建的实例以及上面的 css：

```
// display.vue, 部分代码省略
export default {
  methods: {
    destroyCode () {
      const $target =
document.getElementById(this.id);
      if ($target)
$target.parentNode.removeChild($target);

      if (this.component) {

this.$refs.display.removeChild(this.component.$el
);
        this.component.$destroy();
        this.component = null;
      }
    }
  },
  beforeDestroy () {
    this.destroyCode();
  }
}
```

当 this.code 更新时，整个过程要重新来一次，所以要对 code 进行 watch 监听：

```
// display.vue, 部分代码省略
export default {
  watch: {
    code () {
      this.destroyCode();
      this.renderCode();
    }
  }
}
```

以上就是 Display 组件的所有内容。

使用

新建一条路由，并在 src/views 下新建页面 display.vue 来使用 Display 组件：

```vue
<!-- src/views/display.vue -->
<template>
  <div>
    <h3>动态渲染 .vue 文件的组件-- Display</h3>

    <i-display :code="code"></i-display>
  </div>
</template>
<script>
  import iDisplay from
'../components/display/display.vue';
  import defaultCode from './default-code.js';

  export default {
    components: { iDisplay },
    data () {
      return {
        code: defaultCode
      }
    }
  }
</script>
```

```
// src/views/default-code.js
const code =
`<template>
    <div>
        <input v-model="message">
        {{ message }}
    </div>
</template>
<script>
    export default {
        data () {
            return {
                message: ''
            }
        }
    }
</script>`;

export default code;
```

如果使用的是 Vue CLI 3 默认的配置，直接运行时，会抛出下面的错误：

> [Vue warn]: You are using the runtime-only build of Vue where the template compiler is not available. Either pre-compile the templates into render functions, or use the compiler-included build.

这涉及到另一个知识点，就是 Vue.js 的版本。在使用 Vue.js 2 时，有独立构建（standalone）和运行时构建（runtime-only）两种版本可供选择，详细的介绍请阅读文末扩展阅读 2。

Vue CLI 3 默认使用了 vue.runtime.js，它不允许编译 template 模板，因为我们在 Vue.extend 构造实例时，用了 template 选项，所以会报错。解决方案有两种，一是手动将 template 改写为 Render 函数，但这成本太高；另一种是对 Vue CLI 3 创建的工程做简单的配置。我们使用后者。

在项目根目录，新建文件 vue.config.js：

```
module.exports = {
  runtimeCompiler: true
};
```

它的作用是，是否使用包含运行时编译器的 Vue 构建版本。设置为 true 后就可以在 Vue 组件中使用 template 选项了，但是应用额外增加 10kb 左右（还好吧）。

加了这个配置，报错就消失了，组件也能正常显示。

以上就是 Display 组件所有的内容，如果你感兴趣，可以把它进一步封装，做成 iView Run 这样的产品。

结语

这个小小的 Display 组件，能做的事还有很多，比如要写一套 Vue 组件库的文档，传统方法是在开发环境写一个个的 .vue 文件，然后编译打包、上传资源、上线，如果要修改，哪怕一个标点符号，都要重新编译打包、上传资源、上线。有了 Display 组件，只需要提供一个服务来在线修改文档的 .vue，就能实时更新，不用打包、上传、上线。

还有一点很重要的是，可以看到，在 iView Run 里，默认是直接可以写 iView 组件库的全部组件，并没有额外引入，这是因为 Display 所在的工程，已经将 iView 安装在了全局，Vue.extend 在构造实例

时，已经可以使用全局安装的插件了，如果你还全局安装了其它插件，比如 axios，都是可以直接使用的。

扩展阅读

- new Function (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Func
- Vue.js 2.0 独立构建和运行时构建的区别 (https://jingsam.github.io/2016/10/23/standalone-vs-runtime-only-build-in-vuejs2.html)