

## 28 | WebComponent：像搭积木一样构建Web应用

2019-10-08 李兵

《浏览器工作原理与实践》

课程介绍 >



讲述：李兵

时长 10:07 大小 8.12M



在 [上一篇](#) 文章中我们从技术演变的角度介绍了 PWA，这是一套集合了多种技术的理念，让浏览器渐进式适应设备端。今天我们要站在开发者和项目角度来聊聊 WebComponent，同样它也是一套技术的组合，能提供给开发者组件化开发的能力。

那什么是组件化呢？

其实组件化并没有一个明确的定义，不过这里我们可以使用 10 个字来形容什么是组件化，那就是：**对内高内聚，对外低耦合**。对内各个元素彼此紧密结合、相互依赖，对外和其他组件的联系最少且接口简单。

可以说，程序员对组件化开发有着天生的需求，因为一个稍微复杂点的项目，就涉及到多人协作开发的问题，每个人负责的组件需要尽可能独立完成自己的功能，其组件的内部状态不能影响到别人的组件，在需要和其他组件交互的地方得提前协商好接口。通过组件化可以降低整个系统的耦合度，同时也降低程序员之间沟通复杂度，让系统变得更加易于维护。



使用组件化能带来很多优势，所以很多语言天生就对组件化提供了很好的支持，比如 C/C++ 就可以很好地将功能封装成模块，无论是业务逻辑，还是基础功能，抑或是 UI，都能很好地将其组合在一起，实现组件内部的高度内聚、组件之间的低耦合。

大部分语言都能实现组件化，归根结底在于编程语言特性，大多数语言都有自己的函数级作用域、块级作用域和类，可以将内部的状态数据隐藏在作用域之下或者对象的内部，这样外部就无法访问了，然后通过约定好的接口和外部进行通信。

JavaScript 虽然有不少缺点，但是作为一门编程语言，它也能很好地实现组件化，毕竟有自己的函数级作用域和块级作用域，所以封装内部状态数据并提供接口给外部都是没有问题的。

既然 JavaScript 可以很好地实现组件化，那么我们所谈论的 WebComponent 到底又是什么呢？

## 阻碍前端组件化的因素

在前端虽然 HTML、CSS 和 JavaScript 是强大的开发语言，但是在大型项目中维护起来会比较困难，如果在页面中嵌入第三方内容时，还需要确保第三方的内容样式不会影响到当前内容，同样也要确保当前的 DOM 不会影响到第三方的内容。

所以要聊 WebComponent，得先看看 HTML 和 CSS 是如何阻碍前端组件化的，这里我们就通过下面这样一个简单的例子来分析下：

 复制代码

```
1 <style>
2 p {
3     background-color: brown;
4     color: cornsilk
5 }
6 </style>
7 <p>time.geekbang.org</p>
```

 复制代码

```
1 <style>
2 p {
3     background-color: red;
4     color: blue
5 }
6 </style>
```



上面这两段代码分别实现了自己 p 标签的属性，如果两个人分别负责开发这两段代码的话，那么在测试阶段可能没有什么问题，不过当最终项目整合的时候，其中内部的 CSS 属性会影响到其他外部的 p 标签的，之所以会这样，是因为 CSS 是影响全局的。

我们在 [《23 | 渲染流水线：CSS 如何影响首次加载时的白屏时间？》](#) 这篇文章中分析过，渲染引擎会将所有的 CSS 内容解析为 CSSOM，在生成布局树的时候，会在 CSSOM 中为布局树中的元素查找样式，所以有两个相同标签最终所显示出来的效果是一样的，渲染引擎是不能为它们分别单独设置样式的。

除了 CSS 的全局属性会阻碍组件化，DOM 也是阻碍组件化的一个因素，因为在页面中只有一个 DOM，任何地方都可以直接读取和修改 DOM。所以使用 JavaScript 来实现组件化是没有问题的，但是 JavaScript 一旦遇上 CSS 和 DOM，那么就相当难办了。

## WebComponent 组件化开发

现在我们了解了 CSS 和 DOM 是阻碍组件化的两个因素，那要怎么解决呢？

WebComponent 给出了解决思路，它提供了对局部视图封装能力，可以让 DOM、CSSOM 和 JavaScript 运行在局部环境中，这样就使得局部的 CSS 和 DOM 不会影响到全局。

了解了这些，下面我们就结合具体代码来看看 WebComponent 是怎么实现组件化的。

前面我们说了，WebComponent 是一套技术的组合，具体涉及到了 Custom elements（自定义元素）、Shadow DOM（影子 DOM）和 HTML templates（HTML 模板），详细内容你可以参考 MDN 上的 [相关链接](#)。

下面我们就来演示下这 3 个技术是怎么实现数据封装的，如下面代码所示：

```
1 <!DOCTYPE html>
2 <html>
3
4
5 <body>
6   <!--
```

[复制代码](#)

- 一： 定义模板
- 二： 定义内部CSS样式
- 三： 定义JavaScript行为

```
-->
<template id="geekbang-t">
  <style>
    p {
      background-color: brown;
      color: cornsilk
    }

    div {
      width: 200px;
      background-color: bisque;
      border: 3px solid chocolate;
      border-radius: 10px;
    }
  </style>
  <div>
    <p>time.geekbang.org</p>
    <p>time1.geekbang.org</p>
  </div>
  <script>
    function foo() {
      console.log('inner log')
    }
  </script>
</template>
<script>
  class GeekBang extends HTMLElement {
    constructor() {
      super()
      // 获取组件模板
      const content = document.querySelector('#geekbang-t').content
      // 创建影子DOM节点
      const shadowDOM = this.attachShadow({ mode: 'open' })
      // 将模板添加到影子DOM上
      shadowDOM.appendChild(content.cloneNode(true))
    }
  }

  customElements.define('geek-bang', GeekBang)
</script>

<geek-bang></geek-bang>
<div>
  <p>time.geekbang.org</p>
  <p>time1.geekbang.org</p>
</div>
<geek-bang></geek-bang>
</body>
```



```
59
60 </html>
61
```

仔细观察上面这段代码，我们可以得出：要使用 WebComponent，通常要实现下面三个步骤。

**首先，使用 `template` 属性来创建模板。**利用 DOM 可以查找到模板的内容，但是模板元素是不会被渲染到页面上的，也就是说 DOM 树中的 `template` 节点不会出现在布局树中，所以我们可以使用 `template` 来自定义一些基础的元素结构，这些基础的元素结构是可以被重复使用的。一般模板定义好之后，我们还需要在模板的内部定义样式信息。

**其次，我们需要创建一个 `GeekBang` 的类。**在该类的构造函数中要完成三件事：

1. 查找模板内容；
2. 创建影子 DOM；
3. 再将模板添加到影子 DOM 上。

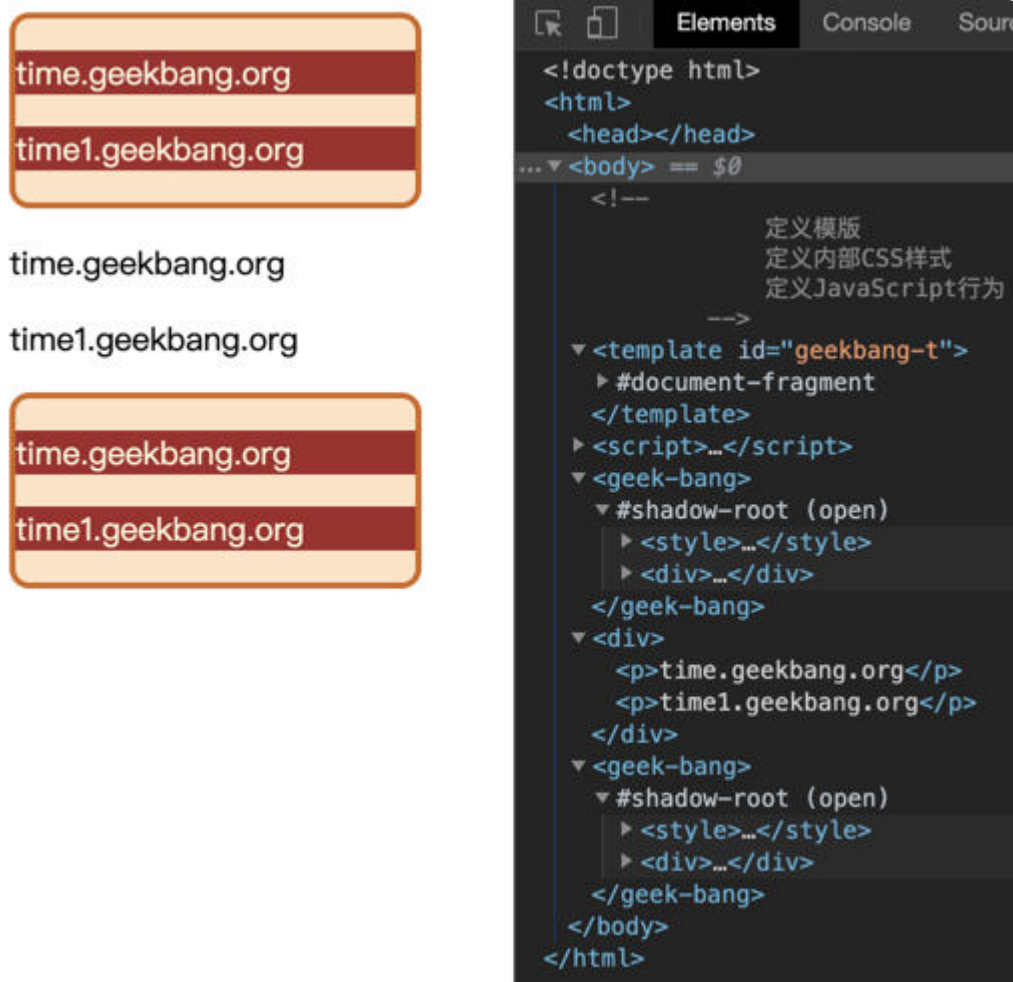
上面最难理解的是影子 DOM，其实影子 DOM 的作用是将模板中的内容与全局 DOM 和 CSS 进行隔离，这样我们就可以实现元素和样式的私有化了。你可以把影子 DOM 看成是一个作用域，其内部的样式和元素是不会影响到全局的样式和元素的，而在全局环境下，要访问影子 DOM 内部的样式或者元素也是需要通过约定好的接口的。

总之，通过影子 DOM，我们就实现了 CSS 和元素的封装，在创建好封装影子 DOM 的类之后，我们就可以使用 `customElements.define` 来自定义元素了（可参考上述代码定义元素的方式）。

**最后，就很简单了，可以像正常使用 HTML 元素一样使用该元素，**如上述代码中的 `<geek-bang></geek-bang>`。

上述代码最终渲染出来的页面，如下图所示：





使用影子 DOM 的输出效果

从图中我们可以看出，影子 DOM 内部的样式是不会影响到全局 CSSOM 的。另外，使用 DOM 接口也是无法直接查询到影子 DOM 内部元素的，比如你可以使用 `document.getElementsByTagName('div')` 来查找所有 `div` 元素，这时候你会发现影子 DOM 内部的元素都是无法查找的，因为要想查找影子 DOM 内部的元素需要专门的接口，所以通过这种方式又将影子内部的 DOM 和外部的 DOM 进行了隔离。

通过影子 DOM 可以隔离 CSS 和 DOM，不过需要注意一点，影子 DOM 的 JavaScript 脚本是不会被隔离的，比如在影子 DOM 定义的 JavaScript 函数依然可以被外部访问，这是因为 JavaScript 语言本身已经可以很好地实现组件化了。

## 浏览器如何实现影子 DOM

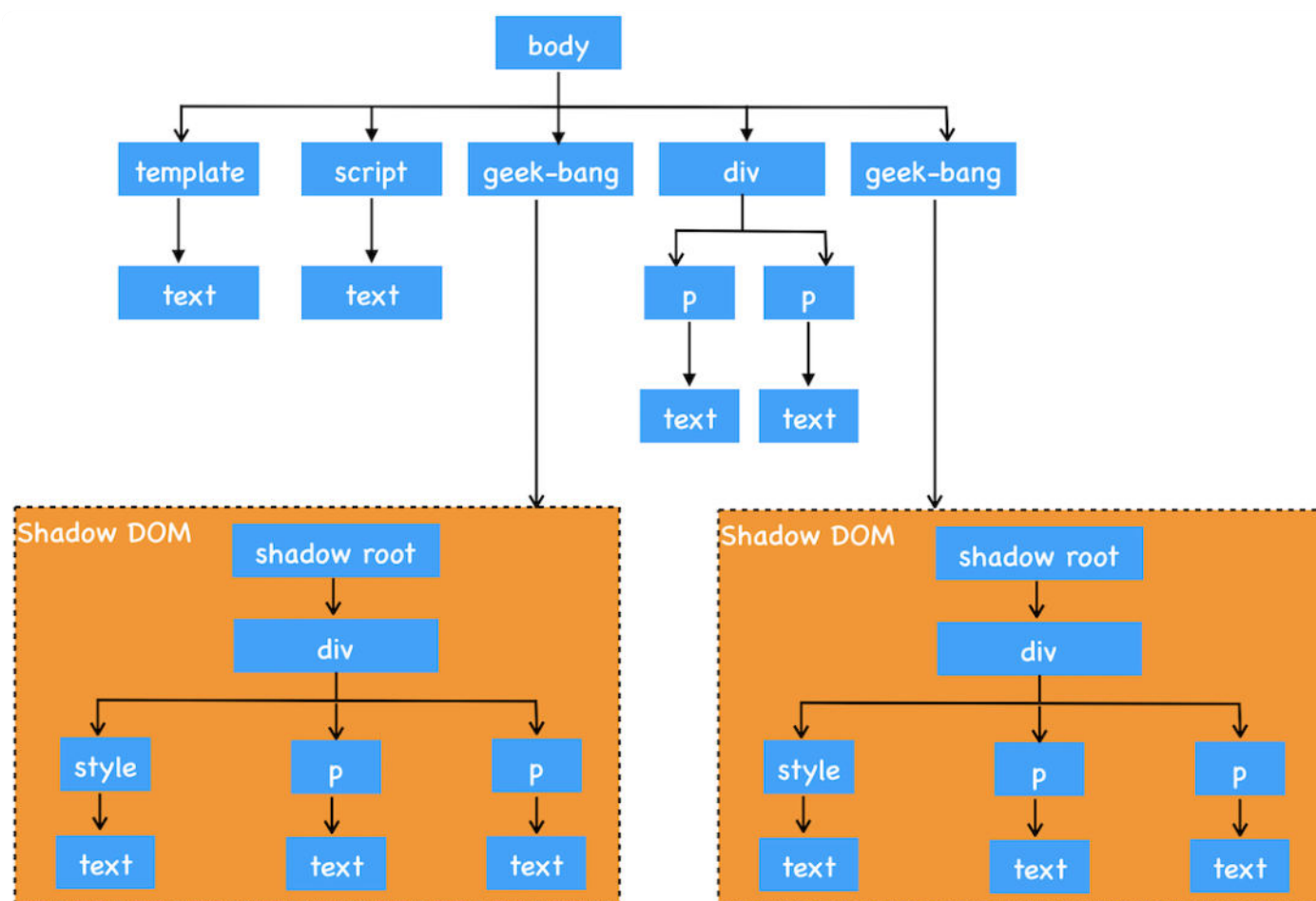
关于 WebComponent 的使用方式我们就介绍到这里。WebComponent 整体知识点不多，内容也不复杂，我认为核心就是影子 DOM。上面我们介绍影子 DOM 的作用主要有以下两点：

1. 影子 DOM 中的元素对于整个网页是不可见的；



2. 影子 DOM 的 CSS 不会影响到整个网页的 CSSOM，影子 DOM 内部的 CSS 只对内部的元素起作用。

那么浏览器是如何实现影子 DOM 的呢？下面我们就来分析下，如下图：



影子 DOM 示意图

该图是上面那段示例代码对应的 DOM 结构图，从图中可以看出，我们使用了两次 `geek-bang` 属性，那么就会生成两个影子 DOM，并且每个影子 DOM 都有一个 `shadow root` 的根节点，我们可以将要展示的样式或者元素添加到影子 DOM 的根节点上，每个影子 DOM 你都可以看成是一个独立的 DOM，它有自己的样式、自己的属性，内部样式不会影响到外部样式，外部样式也不会影响到内部样式。

浏览器为了实现影子 DOM 的特性，在代码内部做了大量的条件判断，比如当通过 DOM 接口去查找元素时，渲染引擎会去判断 `geek-bang` 属性下面的 `shadow-root` 元素是否是影子 DOM，如果是影子 DOM，那么就直接跳过 `shadow-root` 元素的查询操作。所以这样通过 DOM API 就无法直接查询到影子 DOM 的内部元素了。



另外，当生成布局树的时候，渲染引擎也会判断 `geek-bang` 属性下面的 `shadow-root` 元素是否是影子 DOM，如果是，那么在影子 DOM 内部元素的节点选择 CSS 样式的时候，会直接使用影子 DOM 内部的 CSS 属性。所以这样最终渲染出来的效果就是影子 DOM 内部定义的风格。

## 总结

好了，今天就讲到这里，下面我来总结下本文的主要内容。

首先，我们介绍了组件化开发是程序员的刚需，所谓组件化就是功能模块要实现高内聚、低耦合的特性。不过由于 DOM 和 CSSOM 都是全局的，所以它们是影响了前端组件化的主要元素。基于这个原因，就出现 WebComponent，它包含自定义元素、影子 DOM 和 HTML 模板三种技术，使得开发者可以隔离 CSS 和 DOM。在此基础上，我们还重点介绍了影子 DOM 到底是怎么实现的。

关于 WebComponent 的未来如何，这里我们不好预测和评判，但是有一点可以肯定，WebComponent 也会采用渐进式迭代的方式向前推进，未来依然有很多坑需要去填。


## 思考时间

今天留给你的思考题是：你是怎么看待 WebComponents 和前端框架（React、Vue）之间的关系？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

分享给需要的人，Ta 订阅超级会员，你将得 50 元

Ta 单独购买本课程，你将得 20 元

 生成海报并分享

 赞 7

 提建议





## 学习推荐

# JVM + NIO + Spring

各大厂面试题及知识点详解

限时免费



## 精选留言 (25)

写留言



匡晨辉

2019-12-18

web component是通过浏览器引擎提供api接口进行操作，让后在dom，cssom生成过程中控制实现组件化的作用域/执行执行上下文的隔离； vue/react 是在没有浏览器引擎支持的情况下，通过采取一些取巧的手法（比如：js执行上下文的封装利用闭包；样式的封装利用文件hash值作为命名空间在css选择的时候多套一层选择条件（hash值），本质上还是全局的只是不同组件css选择的时候只能选择到组件相应的css样式，实现的隔离）



48



wubinsheng

2019-10-15

原来小程序用的是webComponent，控制台满屏的“#shadow-root”

共 1 条评论 >

37



Zkerhcy

2019-12-16

Vue, React是从开发者层面解决了组件化的问题, 提高了效率。WebComponent是从浏览器引擎实现层面解决了组件化的问题, 从社区来看, 前者的发展优势更明显



👍 23



**mfist**

2019-10-08

下面是我的理解, 请老师纠正。

在没有webcomponent的时候, 通过react和vue基于当前的前端特性去实现组件化, 他们之间是互相影响和借鉴的, 最终react和vue也会向webcomponent标准的方向演进。但是现在由于webcomponent的浏览器支持还不是太好, 所以现阶段它们还是会并存的



👍 11



**蓝配鸡**

2019-10-09

才疏学浅, 以下是个人的理解:

两者互相补充, 互不影响

react提供了陈述式的方法编写网页, 让用户不需要去关心dom改变之类的细节

webComponent则是提供了封装



👍 8



**redbuck**

2020-04-15

webComponent标准可以成为框架间的桥梁.

组件内部可以用vue/react或随便什么技术实现, 只要最终实现约定接口即可.

这样的话, 就可以引入用react开发的A组件, 同时引入用vue开发的B组件, 而他们都在一个Angular项目中. 就像一个原生html标签一样被使用.

所以这也可以是微前端的一种实现方式

共 2 条评论 >

👍 7



**张峰**

2019-10-08

shadow dom 中的style使用rem, r是相对的html的font-size 这点很坑

共 1 条评论 >

👍 2





张峰

2019-10-08

web-component之于vue/react，类似于ES6789之于coffeeScript/typeScript，后者只是前者的临时替补，omi和angular都已经支持web-component



1



君自兰芳

2020-11-12

“在影子 DOM 定义的 JavaScript 函数依然可以被外部访问”

有个疑问，在影子 DOM 定义的变量或函数是属于全局作用域吗？

共 1 条评论 >



neohope

2020-07-16

虚拟DOM解决的是效率问题，防止频繁的DOM操作，导致浏览器不断的刷新，将多次刷新操作，变成一次刷新操作。

影子DOM解决的是作用域隔离的问题，特别是在大规模项目上，可以规避全局设置的相互影响。

其实这两种方案，都是在现行标准下的解决方案，问题确实是解决了，但算不上优雅。还是期待能在语言规范和浏览器底层进行解决，这样才能真正去取代本地应用和各种小程序。



Roy

2020-04-29

webcomponent很好的实现与第三方应用的组合使用。



HoSalt

2020-04-12

WebComponents 如何传递数据以及如何重置样式？



blueBean

2020-03-13

老师请问应该如何拿到shadowDom呢？只能用选择器选中自定义标签，shadowDom下的元素全都选不到，没办法在它的子元素下操作dom，网上也没找到解决办法..老师知道怎么拿吗？



共 1 条评论 >



大前端洞见

2020-03-08

webcomponent 组件目前是可以在 angular react 框架中渲染使用的。



狂躁小胖

2020-03-07

Webcomponent、React以及Vue都实现了DOM的组件化，webcomponent 是W3C的亲儿子，通过shadow dom 技术实现dom以及css的隔离；React以及Vue则不是正规军，但是也同样达到了dom组件化的目的，然后结合已有的html特性实现样式的隔离比如scoped。



Jy

2020-03-02

WebComponent使用后，搜索引擎的SEO支持如何？



匡晨辉

2019-12-18

在回答课后题的过程中我又想到一个问题：文中讲述了WebComponent对cssom， dom的隔离，没有谈到对WebComponent中的js作用域的隔离，老师能具体讲讲js的作用域在web component的实现中是怎么实现隔离的呢？

共 1 条评论 >



coder

2019-12-14

有个疑问，“影子 DOM 中的元素对于整个网页是不可见的”，那么“再将模板添加到影子 DOM 上”，不还是看不见吗？也就是虽然我们组件化了，但是这个组件我们看不见？这样看不见的组件有什么使用意义吗？



-\_-||

2019-12-13

vue/react的出现一个目的就是解决前端组件化，WebComponents也是在组件化的思想下产生的。有点像“开局就送vip，一刀999”的感觉，这样后期框架在基于WebComponents演进的过程中，会有更好的体验，自然也要求WebComponents不断的完善。





陈十二

2019-12-13

React Vue 的出现是为了解决包括 WebComponent 想解决的问题在内的很多 web 开发的痛点，在单纯组件化方面 他们采取了各自的解决方案，我觉得 webcomponent 应该考虑借鉴流行度高的框架的实现。

