28-WebComponent：像搭积木一样构建Web应用

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：

```
<style>
p {
      background-color: brown;
      color: cornsilk
   }
</style>
<p>time.geekbang.org</p>
```

www.ixuexi.cf 分享站 用户标记 vip_user_001

```
<style>
p {
      background-color: red;
      color: blue
   }
<p>time.geekbang</p>
```

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：

```
<!DOCTYPE html>
<html>


<body>
    <!--
            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>


</html>
```

详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；

www.ixuexi.cf 分享站 用户标记 vip_user_001

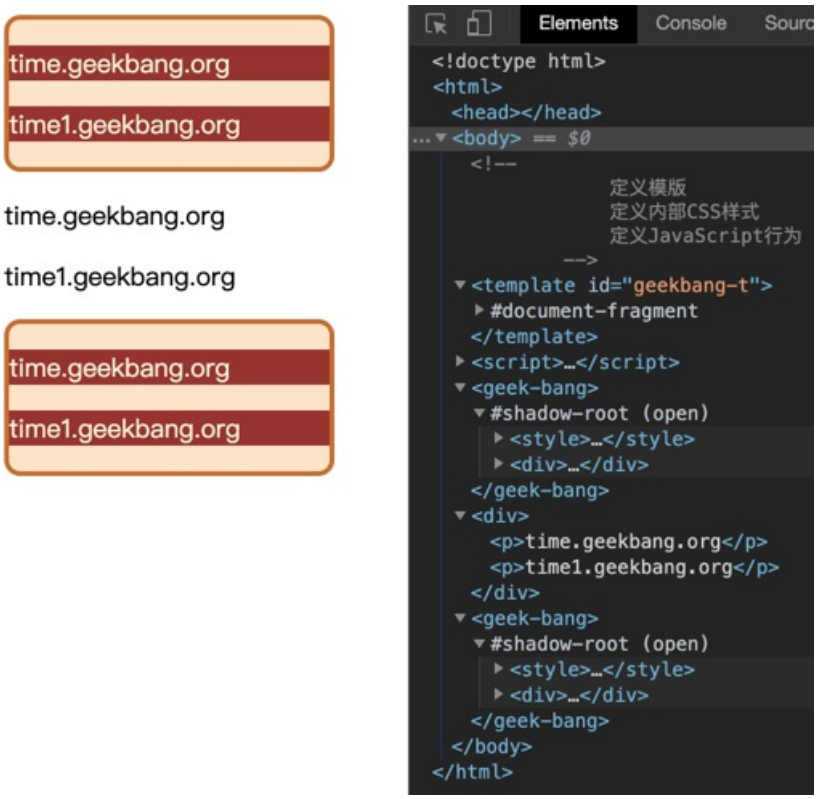
3. 再将模板添加到影子DOM上。

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。

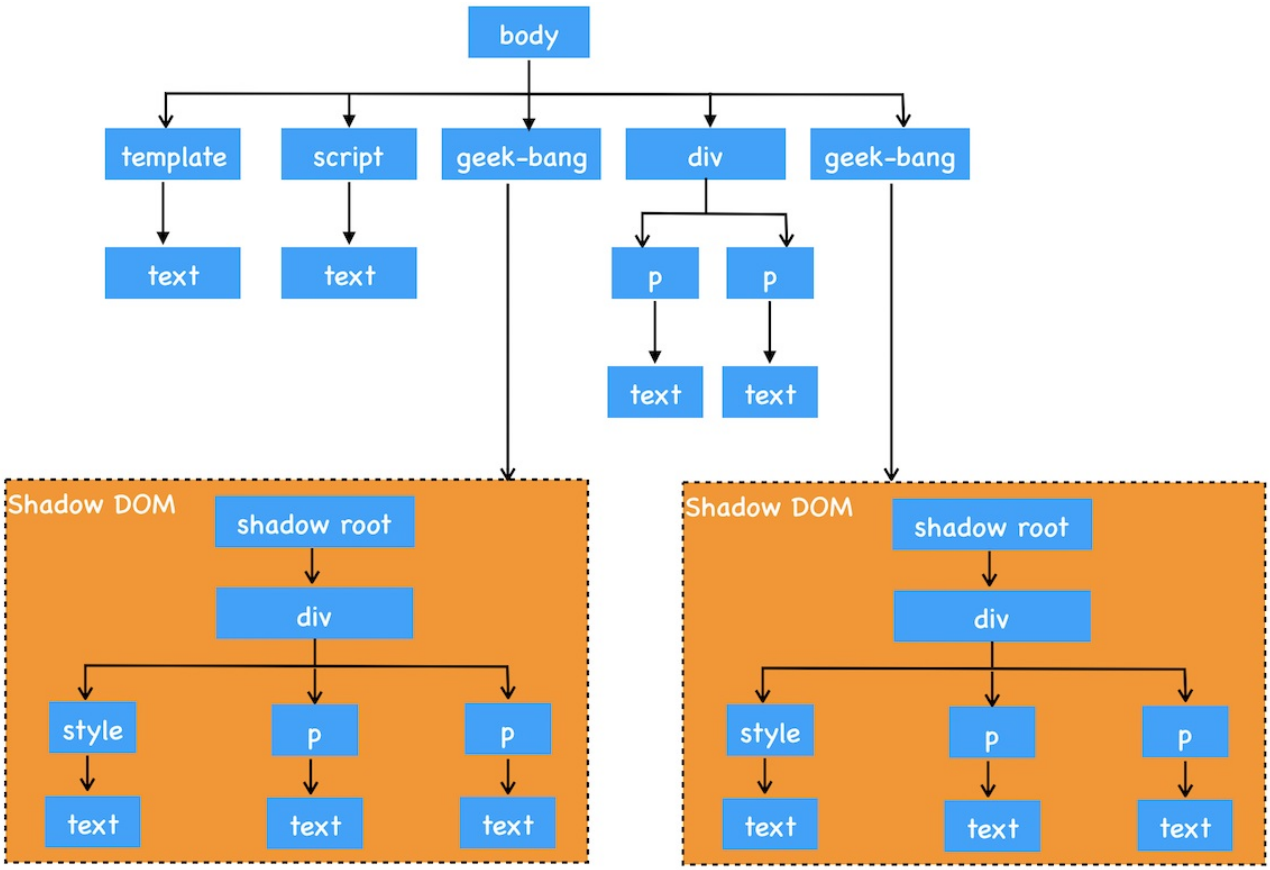
浏览器如何实现影子DOM

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内部定义的样式。

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

www.ixuexi.cf 分享站 用户标记 vip_user_001

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）之间的关系的?

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。