



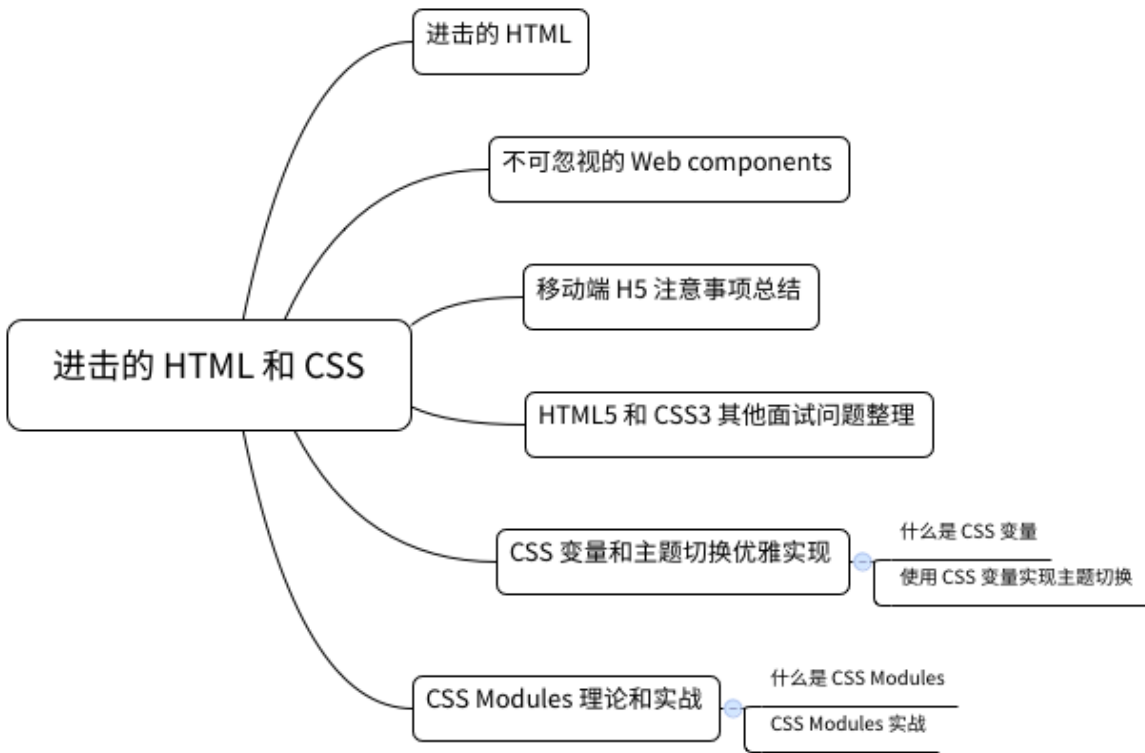
前端开发核心知识进阶：50 讲从夯实基础到突破瓶颈
来自 Lucas ... · 盐选专栏

[查看详情 >](#)

进击的 HTML 和 CSS

通过前面的课程介绍，我们认识到了 JavaScript 语言的飞速发展。其实 HTML 和 CSS 也不断「要求着」进步，本节课程我们就来了解一下发展中的 HTML 和 CSS。之所以叫做「进击的」，是因为确实有很多新的特性非常实用且具有变革精神。

相关知识点如下：



其中，除了罗列一些热点面试题目以外，我们将重点分析 CSS 变量和 CSS Modules，我认为这两个概念代表了未来的发展方向。当下来看，也有必要根据情况，融合到成熟的项目中，加以应用。因此这方面的内容除了基本理论，我都会给出实战案例和构建流程。

进击的 HTML

我们再来看看 HTML 的历史和规范常识。HTML 规范是 W3C 与 WHATWG 合作共同产出的，HTML5 因此也不例外。其中：

W3C 指 World Wide Web Consortium

WHATWG 指 Web Hypertext Application Technology Working Group

说好听了是「合作产出」，但其实更像是「HTML5 有两套规范」。但话说天下大势合久必分，分久必合，如今（就在前几天，2018.5.29）它们又表示将会开发单一版本的 HTML 规范。

那么 HTML5 给开发者提供了哪些便利呢？简单列举有：

用于绘画的 canvas 元素

用于媒介播放的 video 和 audio 元素

对本地离线存储更好的支持 (localStorage、sessionStorage)

新的语义化标签 (article、footer、header、nav、section...)

新的表单控件 (calendar、date、time、email、url、search...)

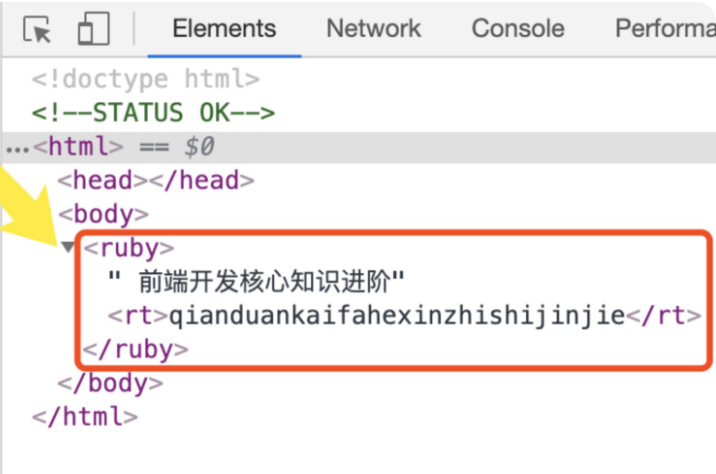
除了这些常规的之外，还有：

给汉字加拼音

qianduankaifahexinzhishijinjie

前端开发核心知识进阶

qianduankaifahexinzhishijinjie
前端开发核心知识进阶



```
<!doctype html>
<!--STATUS OK-->
...<html> == $0
  <head></head>
  <body>
    <ruby>
      " 前端开发核心知识进阶"
      <rt>qianduankaifahexinzhishijinjie</rt>
    </ruby>
  </body>
</html>
```


展开收起组件

简单几行代码：

► 详细信息

就可以实现：

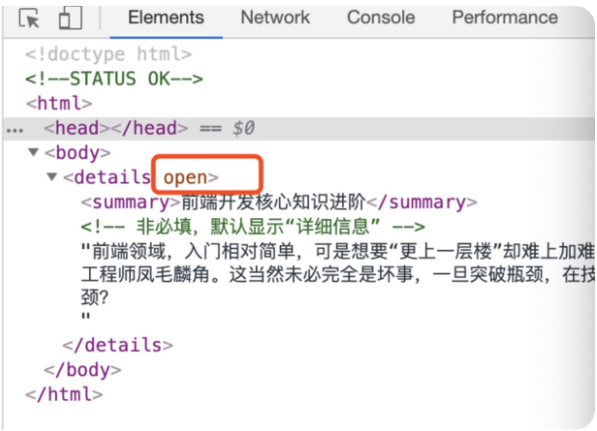
► 前端开发核心知识进阶



```
<!doctype html>
<!--STATUS OK-->
<html>
... <head></head> == $0
  <body>
    <details>
      <summary>前端开发核心知识进阶</summary>
      <!-- 非必填，默认显示“详细信息” -->
      "前端领域，入门相对简单，可是想要“更上一层楼”
      工程师凤毛麟角。这当然未必完全是坏事，一旦多
      颈？
    </details>
  </body>
</html>
```

上图为「收起」效果。

▼ 前端开发核心知识进阶
前端领域，入门相对简单，可是想要“更上一层楼”却难上加难，也就是我们常说的“职业天花板较低”，君不见——市场上高级/资深前端工程师凤毛麟角。这当然未必完全是坏事，一旦突破瓶颈，在技能上脱颖而出，便是更广阔的空间。那么，如何从夯实基础到突破瓶颈？



我并无意赘述 Web components 的基础概念，但是我认为，作为「更高阶」的前端工程师，要时刻保持技术视野和信息广度。在框架带来的「组件化」、「生命周期化」这些统治级别的概念下，对比并结合 Web components，我认为是可以深入研究的一个课题方向。我总结一下 Web components 的特殊点或者优点：

原生规范，无需框架

这条优点的后半句话是：「但是继承且具备了框架的优点」，在新的 Web components 规范中，我们会发现组件生命周期的概念、slot 概念、模版概念（类比 JSX 或者 Vue template），再结合本来就已经存在的组件化，shadow dom，扩展原生元素的能力，我认为 Web components 还是具备了较好的发展前景。

原生使用，无需编译

想想现有的一系列框架，不论是 Vue 还是 React，都需要进行编译。而 Web components 因为原生，会得到浏览器的天然支持，自然就可以免去编译构建过程。

真正的 CSS scope

Web components 实现了真正的 CSS scope，做到了样式隔离。这一点读者可以对比我们下面即将介绍的 CSS Modules。真正的 CSS scope 对于项目的可维护性至关重要。

进击的 HTML 和 CSS 带来了进击的 Web components 概念。通过这个案例，我更想建议读者：真正的高级工程师，不仅仅要理解 this、熟练掌握各种基础（当然这是前提），更要有技术嗅觉，对新的解决方案能够理解，并进行对比，面向「未来」编程。

移动端 H5 注意事项总结

HTML5 因为其强大先进的能力，毫无疑问带来了一场开发的变革。在国内，体现最明显的就是各种 H5 移动页面。

因为移动端的碎片化现象，以及技术落地的成熟度尚浅，造成了不少问题，那么移动端开发 H5 有哪些坑以及小技巧呢？

这里列举一些典型情况，目的在于梳理和整理，不再一一详解。具体信息社区上都可以找到，感兴趣的读者可以另行学习。

打电话发短信写邮件的小技巧

这些技巧都和 a 标签相关，其中打电话：

打电话给警察局

发短信：

发短信给警察局

写邮件依赖「mailto」：

发邮件给警察局

我们设置可以添加抄送：

发邮件给警察局，并抄送给我爸爸

除了抄送，也可以私密发送：

发邮件给警察局，并抄送给我爸爸，密送给我妈妈

群发也可以：

发邮件给警察局，以及 120 急救

既然都支持群发了，那么定义主题和内容也不在话下：

发邮件给警察局，并添加救命主题

包含内容用 body 体现：

发邮件给警察局，并添加救命主题和内容

内容也是支持插入图片和链接的，这里不再一一列举。

移动端 300 毫秒点击延迟以及点击穿透现象

这是由于历史原因造成的，一般解决手段为禁止混用 touch 和 click，或者增加一层「透明」蒙层，也可以通过延迟上层元素消失来实现。

点击元素禁止产生背景或边框

一般可以使用：

```
-webkit-tap-highlight-color: rgba(0,0,0,0);
```

属性进行禁用。

禁止长按链接与图片弹出菜单

一般可以使用：

```
-webkit-touch-callout: none;
```

禁止用户选中文字

```
-webkit-user-select:none;  
user-select: none;
```

取消 input 输入时，英文首字母的默认大写

iOS 有效。

语音和视频自动播放

自动播放是一个很麻烦的话题。不同浏览器内核支持自动播放的情况不一样，甚至 webkit 内核对于自动播放的策略也一直在调整当中。自动播放有时候也带着条件：比如设置静音等。

具体信息更新可以参考：[New video Policies for iOS](#)。

一般我们设置自动播放的回退策略是用户触摸屏幕时进行的播放：

```
// JS 绑定自动播放（操作 window 时，播放音乐）
$(window).on('touchstart', () => {
    video.play()
})

// 微信环境
document.addEventListener("WeixinJSBridgeReady", () => {
    video.play()
}, false)
```

视频全屏播放

为了使视频全屏播放，我们一般设置：

但是最终情况还是要受到浏览器引擎实现的影响。

开启硬件加速

在做动画时，为了达到更好的性能效果，我们往往会选用硬件加速。一般手段为：

```
transform: translate3d(0,0,0);
```

fixed 定位问题

这个问题主要体现在 iOS 端，比如软键盘弹出时，某些情况下，会影响 fixed 元素定位；配合使用 transform、translate 时，某些情况下，也会影响 fixed 元素定位。一般解决方案是模拟 fixed 定位，或者使用 iScroll 库。

怎么让 Chrome 支持小于 12px 的文字？

一般通过：

```
-webkit-text-size-adjust:none;
```

实现。

HTML5 和 CSS3 其他面试问题整理

关于 HTML5 和 CSS3 的面试问题都并不困难，往往都是属于「是否听说过」、「用过哪些」，这种知道即可的问题，不会太有「深度」。这里我们总结一下关于 HTML5 和 CSS3 新特性的问题，答案也比较容易找到，这里仅做梳理，不再进行展开。如果有疑问的读者，欢迎在读者群中讨论。

link 和 @import 的区别

CSS3 新增选择符有哪些

CSS 如何定义权重规则

如何使用纯 CSS 创建一个三角形

CSS3 如何写出一个旋转的立方体

localStorage 和 cookies 的区别是什么

如何实现浏览器内多个标签页之间的通信

渐进增强和优雅降级概念区别是什么

如何实现 CSS3 动画

这些内容比较基本，和进阶关系不大，我们不过多纠结。

CSS 变量和主题切换优雅实现

CSS 变量或者 CSS 自定义属性一直以来是一个值得关注的方向。我们前端没必要去「叫嚣」CSS + HTML 是否图灵完备，但是 CSS 变量时代确实已经到来。注意这里所说的不是 CSS 预处理器（类似 Less，Sass）中的变量，而是实实在在的原生支持特性。

什么是 CSS 变量

什么是 CSS 变量呢？我们直接来看实例，有代码：

```
body {  
  background: white;  
  color: #555;  
}
```

```
a, a:link {  
  color: #639A67;  
}  
a:hover {  
  color: #205D67;  
}
```

如果我们借助 CSS 变量，定义：

```
:root {  
  --bg: white;
```

```
--text-color: #555;
--link-color: #639A67;
--link-hover: #205D67;
}
```

之后，上述代码可以直接简化为：

```
body {
  background: var(--bg);
  color: var(--text-color);
}


a, a:link {
  color: var(--link-color);
}
a:hover {
  color: var(--link-hover);
}
```

这个很好理解，在任何语言中，变量是个好东西：它可以降低维护成本，甚至实现更好的性能。

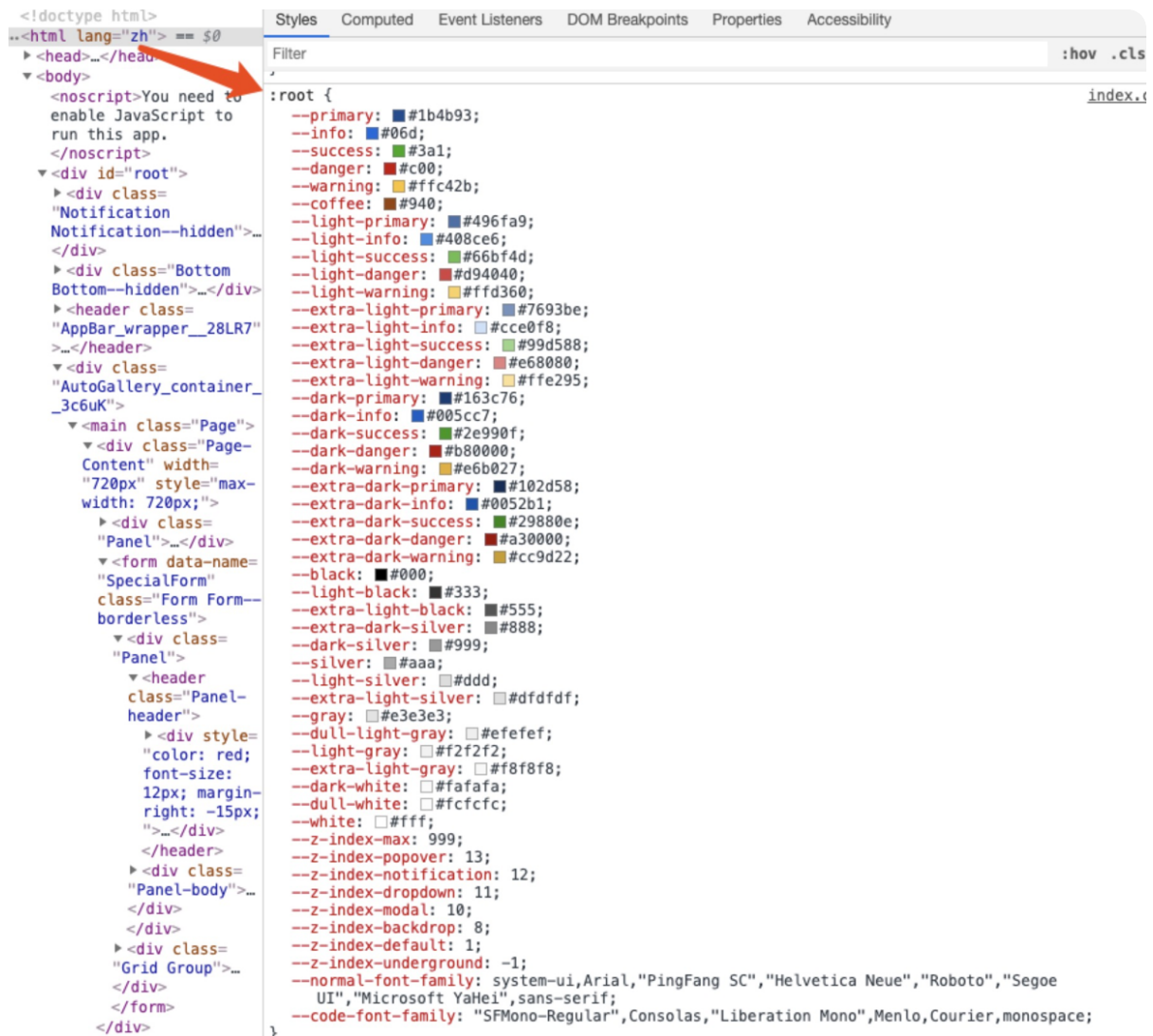
CSS 变量语法也很简单：我们使用`--变量名`的方式定义变量，使用`var(--变量名)`的方式消费变量。

更多 CSS 变量的基础内容可以访问：[使用 CSS 变量](#)。

值得一提的是，CSS 变量的兼容性「出乎意料」的好：

		Desktop						Mobile						
		Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Android webview	Chrome for Android	Edge Mobile	Firefox for Android	Opera for Android	Safari on iOS	Samsung Internet
var()		49	15	31	No	36	Yes	50	49	15	31	36	Yes	?

我就在自己的项目中大范围使用了 CSS 变量，在 html 根节点下，定义 :root：



除了简单应用变量，我们能玩出哪些更高级的用法呢？

使用 CSS 变量实现主题切换

一键切换主题，以往实现方式较为复杂。借助 CSS 变量，一切变得容易起来。

仍然以开头：

```
:root {  
  --bg: white;  
  --text-color: #555;  
  --link-color: #639A67;  
  --link-hover: #205D67;  
}
```

为例，我们再定义一个 .pink-theme 对应粉色主题：

```
.pink-theme {  
  --bg: hotpink;  
  --text-color: white;  
  --link-color: #B793E6;  
  --link-hover: #3532A7;  
}
```

这样一来，在切换主题时，就变得和 toggle class 一样简单。

```
const toggleBtn = document.querySelector('.toggle-theme')  
  
toggleBtn.addEventListener('click', e => {  
  e.preventDefault()  
  
  if (document.body.classList.contains('pink-theme')) {  
    // 当前主题为粉色主题，需要移除 pink-theme class  
    document.body.classList.remove('pink-theme')  
  
    toggle.innerText = '切换正常主题色'  
  } else {  
    document.body.classList.add('pink-theme')  
    toggle.innerText = '切换为粉色少女主题'  
  }  
})
```

同时，我们可以将「进击的 CSS」和「进击的 HTML」相结合，利用 localStorage 实现主题的保存：

```
const toggleBtn = document.querySelector('.toggle-theme')  
  
if (localStorage.getItem('pinkTheme')) {  
  document.body.classList.add('pink-theme')  
  toggle.innerText = '切换为粉色少女主题'  
}
```

```
toggleBtn.addEventListener('click', e => {
  e.preventDefault()

  if (document.body.classList.contains('pink-theme')) {
    // 当前主题为粉色主题，需要移除 pink-theme class
    document.body.classList.remove('pink-theme')

    toggle.innerText = '切换正常主题色'
    localStorage.removeItem('pinkTheme')
  } else {
    document.body.classList.add('pink-theme')
    toggle.innerText = '切换为粉色少女主题'
    localStorage.setItem('pinkTheme', true)
  }
})
```

非常的简单直观，我认为这将会成为 CSS 发展的一个不可避免的趋势。

CSS Modules 理论和实战

我做面试官时，对 CSS 的考察除了基础布局和经验以外，非常喜欢问 CSS 工程相关的题目，比如：

如何维护大型项目的 z-index

比如，

如何维护 CSS 选择器和样式之间的冲突

这个环节我们就来谈谈 CSS Modules，看看这个方案是否能让「CSS 冲突成为历史」。

什么是 CSS Modules

CSS Modules 是指：

项目中所有 class 名称默认都是局部起作用的。

其实，CSS Modules 并不是一个官方规范，更不是浏览器的机制。它依赖我们的项目构建过程，因此实现往往需要借助 Webpack。借助 Webpack 或者其他构建工具的帮助，可以将 class 的名字唯一化，从而实现局部作用。

这么说可能比较抽象，我们来看一个例子：

```
This is a test
```

对应的样式表为：

```
.test {  
  color: red;  
}
```

再经过编译构建之后，对应的 HTML 和 CSS 分别为：

```
This is a test  
  
._style_test_309571057 {  
  color: red;  
}
```

其中 class 名是动态生成的，全项目唯一的。因此通过命名规范的唯一性，达到了避免样式冲突的目的。

仔细想来，这样的解决方案似乎有一个问题：如何实现样式复用？因为生成了全局唯一的 class 名，那么我们如何像传统方式那样实现样式复用呢？

从原理上想，全局唯一的 class 是在构建过程中，如果能给在构建过程进行标识，表示该 class 将被复用，就可以解决问题了。这样的方式，就依靠 composes 关键字实现。我们来看案例：

样式表 style.css 文件中：

```
.common {  
  color: red;  
}  
  
.test {  
  composes: common;  
  font-size: 18px;  
}
```

注意我们使用了 composes 关键字，在 .test 中关联了 .common 样式。

对于 HTML 文件：

```
import style from "./style.css";
```

```
  This is a test
```

进行编译构建后：

```
  This is a test
```

我们看 div 的 class 被加进了 _style__common_404840，这样就实现了复用样式。

明白了道理，我们该如何应用 CSS Modules 呢？

CSS Modules 实战

实战应用 CSS Modules，我将会选取 Webpack 构建一个项目，一步一步进行分析讲解。因为主题并不是「如何配置 Webpack」，因此一些 Webpack 基础不再赘述，同时为了简化问题，我们不进行其他 Webpack（比如 dev server）配置。

Step 1: 创建项目

```
npm init --y
```

此时生成 package.json 如下：

```
{
  "name": "css-modules",
  "version": "1.0.0",
  "description": "README.md",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Step 2: 创建必要文件

```
mkdir src
touch index.html
```

在 ./src 文件夹中，创建：index.js：

```
import bluestyle from './style.css';
import greenstyle from './app.css';

let html = `
```

I should be displayed in blue.

I should be displayed in green.

```
`;  
document.write(html);
```

以及 style.css:

```
.my_css_selector {  
  color: blue;  
}
```

和 app.css:

```
.my_css_selector {  
  color: green;  
}
```

在这两个样式文件中，我们使用了相同的 class 名。

Step 3: 安装依赖

接下来我们按照 webpack、webpack-cli、babel 全家桶（babel-core、babel-loader、babel-preset-env）和相应的 loaders：css-loader、style-loader 以及 extract-text-webpack-plugin 插件。

这些依赖项具体是做什么的这里不再赘述，有不了解的读者可以自行 Google 学习。另外，强烈建议安装版本遵循：

```
"babel-core": "^6.26.3",  
"babel-loader": "^7.1.4",
```

```
"babel-preset-env": "^1.6.1",  
"css-loader": "^0.28.11",  
"extract-text-webpack-plugin": "^4.0.0-beta.0",  
"style-loader": "^0.21.0",  
"webpack": "^4.1.0",  
"webpack-cli": "^3.1.1"
```

否则会出现类似 webpack 版本和 extract-text-webpack-plugin 不兼容等依赖版本问题。

正常流程下来，我们 package.json 如下：

```
{  
  "name": "css-modules",  
  "version": "1.0.0",  
  "description": "README.md",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "devDependencies": {  
    "babel-core": "^6.26.3",  
    "babel-loader": "^7.1.4",  
    "babel-preset-env": "^1.6.1",  
    "css-loader": "^0.28.11",  
    "extract-text-webpack-plugin": "^4.0.0-beta.0",  
    "style-loader": "^0.21.0",  
    "webpack": "^4.1.0",  
    "webpack-cli": "^3.1.1"  
  }  
}
```

Step 4: 编写 webpack 配置

创建 webpack 配置文件：

```
touch webpack.config.js
```

并编写：

```
var ExtractTextPlugin = require('extract-text-webpack-plugin');

module.exports = {
  entry: './src',
  output: {
    path: __dirname + '/build',
    filename: 'bundle.js'
  },
  module: {
    rules: [
      {
        test: /\.js/,
        loader: 'babel-loader',
        include: __dirname + '/src'
      },
      {
        test: /\.css/,
        loader: ExtractTextPlugin.extract("css-loader?modules&importLoaders=1&localIdentName=[name]__[local]__[hash:base64:5]")
      }
    ]
  },
  plugins: [
    new ExtractTextPlugin("styles.css")
  ]
}
```

我们使用了 `extract-text-webpack-plugin` 插件，并定义入口为 `./src` 目录，产出为 `__dirname + '/build'` 目录。对后缀名为 `css` 的文件使用 `css-loader` 解析，产出为 `styles.css` 文件并在 `index.html` 中使用。

注意我们看对于 `css-loader`，设置了 `modules` 参数，进行了 `css modules` 处理。

Step 4: 编写 npm script 并运行

还差一步，我们将 `package.json` 中的 `script` 命令改为：

```
"scripts": {  
  "start": "webpack --mode development"  
},
```

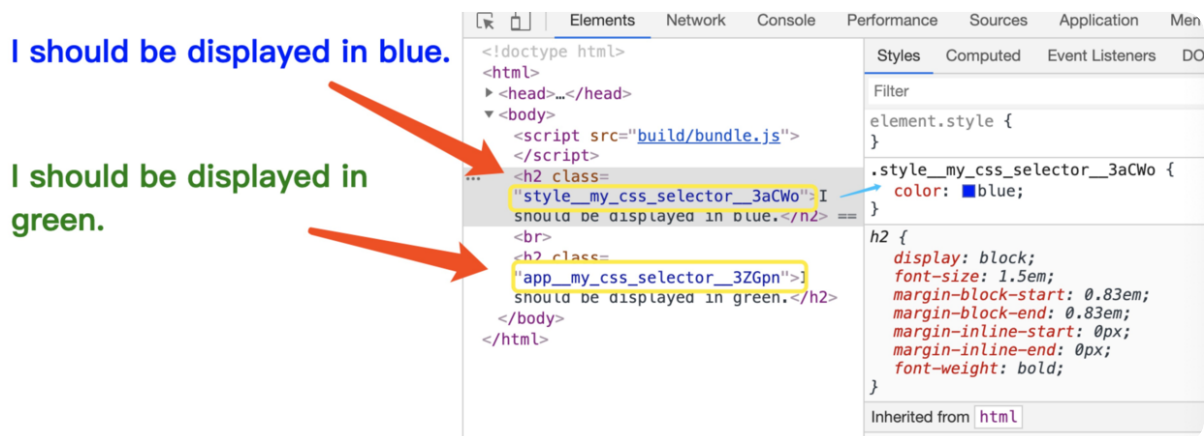
便是运行 `webpack`，此时 `package.json` 内容为：

```
{  
  "name": "css-modules",  
  "version": "1.0.0",  
  "description": "README.md",  
  "main": "index.js",  
  "scripts": {  
    "start": "webpack --mode development"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "devDependencies": {  
    "babel-core": "^6.26.3",  
    "babel-loader": "^7.1.4",  
    "babel-preset-env": "^1.6.1",  
    "css-loader": "^0.28.11",  
    "extract-text-webpack-plugin": "^4.0.0-beta.0",  
    "style-loader": "^0.21.0",  
    "webpack": "^4.1.0",
```

```
"webpack-cli": "^3.1.1"
}
}
```

运行 npm start，得到产出，打开页面会发现：

如图，已经在编译过程中完成了 css module 处理。



总结

本节课程我们既有「大面儿」上的梳理，也有关键点的「实战」深入。有趣实用的标签和属性、移动端 H5 注意事项总结、HTML5 和 CSS3 面试题梳理，这三块内容旨在将碎片化的知识点以「记事本」式的排列；Web components 更多给大家带来对新技术的思考和总结；CSS 变量、CSS Module 是我认为最有发展潜力、最有实用价值、最能马上落地实现的解决方案。

HTML 和 CSS 向来被忽视，但是涉及到项目组织和构建，涉及到新技术的调研和决断，我们切不可含糊。

点击查看下一节 >

响应式布局和 Bootstrap 实现