

# 万岁，浏览器原生支持ES6 export和import模块啦！

这篇文章发布于 2018年08月2日，星期四，01:09，归类于 JS API。阅读 12524 次, 今日 39 次 22 条评论

by zhangxinxu from <https://www.zhangxinxu.com/wordpress/?p=7876>

本文可全文转载，但需得到原作者书面许可，同时保留原作者和出处，摘要引流则随意。

## 一、前言

JS中直接 `import` 其他模块是个很棒的能力，ES6规范中就提供了这样的特性。然后，长久以来，都只有在Node.js中才能无阻使用，浏览器都没有原生支持。



Node.js对于我而言，就像是个在另外一个城市结交的好朋友，简单了解，能和睦相处即可，因此，Node.js支持 `import` 功能，就好像朋友升职赚了大钱一样，替他开心，不过也就只是替他开心，自己其实还是淡然的。但是，web浏览器就不一样了，这个可是我打算厮守一生的伴侣，因此，web浏览器原生支持 `import` 功能，那就好像自己的老婆升职赚了大钱一样，那比自己赚了大钱还开心，心中一百个“万岁”。

ES6在浏览器中的import功能分为静态import和动态import。

其中静态import出现更早，浏览器兼容性更好，支持浏览器包括：Safari 10.1+，Chrome 61+，Firefox 60+，Edge 16+。

IE	Edge *	Firefox	Chrome	Safari	iOS Safari *
			49		
			66		10.3
		60	67		11.2
11	17	61	68	11.1	11.4
	18	62	69	12	12
		63	70	TP	
			71		

动态import支持晚一些，兼容性要差一些，目前Chrome浏览器和Safari浏览器支持，不过相信很快其他浏览器也会跟进。

IE	Edge *	Firefox	Chrome	Safari	iOS Safari *
			49		
			66		10.3
		60	67		11.2
11	17	61	68	11.1	11.4
	18	62	69	12	12
		63	70	TP	
			71		

本文会对这两种模块导入都做介绍，因此，本文内容篇幅较长，且有一定深度，需要预留较多时间阅读。

## 二、静态import

我们先从最简单的案例说起，例如，我想想，demo比较方便演示的效果，啊，那就实现改变 `<p>` 元素的文字颜色。

主页面相关script代码如下：

```
<script type="module">
  // 导入firstBlood模块
  import { pColor } from './firstBlood.mjs';
  // 设置颜色为红色
  pColor('red');
</script>
```

然后firstBlood.mjs文件中代码为：

```
// export一个改变<p>元素颜色的方法
export function pColor (color) {
  const p = document.querySelector('p');
  p.style.color = color;
}
```

您可以狠狠地点击这里：[浏览器原生import实现文字变红demo](#)

可以看到 `<p>` 文字变红了：

有了案例，下面基础知识就更好消化与理解了。

- 对于需要引入模块的 `<script>` 元素，我们需要添加 `type="module"`，这个时候，浏览器会把这段内联script或者外链script认为是ECMAScript模块。
- 模块JS文件，业界或者官方约定俗成命名为 `.mjs` 文件格式，一来可以和普通JavaScript文件（`.js` 后缀）进行区分，一看就知道是模块文件；二来Node.js中ES6的模块化特性只支持 `.mjs` 后缀的脚本，可以和Node.js保持一致。当然，我们直接使用 `.js` 作为模块JS文件的后缀也是可以的。

在浏览器侧进行import模块引入，其对模块JS文件的mime type要求非常严格，务必和JS文件一致。这就导致，如果我们使用 `.mjs` 文件格式，则需要在服务器配置mime type类型，否则会报错：

Failed to load module script: The server responded with a non-JavaScript MIME type of "". Strict MIME type checking is enforced for module scripts per HTML spec.

Nginx对于不识别后缀默认会给一个 `application/octet-stream` 的MIME type, 方便下载等处理, 但是, 不好意思, 在模块化引入这里, 这个MIME type无效, 需要足够精准才行, 为 `application/javascript`, 然后根据自己测试, IIS服务器中 `application/x-javascript` 也是可以的。

无论是Apache服务器还是Nginx, 都可以修改mime.types文件使 `.mjs` 的MIME type和 `.js` 文件一样。

除了 `export` 普通的 `function`, 我们还可以 `export const` 或者其他任何变量或者声明。也支持 `default` 命令。再看下面一个例子, `<p>` 文字变红, 以及垂直翻转, 演示 `const` 和 `default` 使用。

假设模块脚本文件名是doubleKill.mjs, 其代码如下:

```
// doubleKill.mjs
// const 和 default功能演示
export default () => {
  const p = document.querySelector('p');
  p.style.transform = 'scaleY(-1)';
};
export const pColor = (color) => {
  const p = document.querySelector('p');
  p.style.color = color;
}
```

import部分逻辑代码为:

```
<script type="module">
  // 导入doubleKill模块
  import * as module from './doubleKill.mjs';
  // 执行默认方法
  module.default();
  // 设置颜色为红色
  module.pColor('red');
</script>
```

就可以实现 `<p>` 元素文字变红同时垂直翻转的效果, 如下截图:



您可以狠狠地点击这里: [静态import模块const和default使用demo](#)

### 三、nomodule与向下兼容

模块脚本我们可以使用 `type="module"` 进行设定, 对于并不支持 `export` 和 `import` 的浏览器, 我们可以使用nomodule进行向下兼容。

```
<script type="module" src="module.mjs"></script>
<script nomodule src="fallback.js"></script>
```

对于支持ES6模块导入的浏览器, 自然也支持原生的 `nomodule` 属性, 此时 `fallback.js` 是忽略的;

但是，对于不支持的老浏览器，无视 `nomodule`，此时 `fallback.js` 就会执行，于是浏览器全兼顾。

理论就如上面分析得这么完美，然后实际上，还是存在问题的。

主要问题在低端浏览器 `.mjs` 资源会冗余加载，例如[这个测试demo](#)在IE11下的网络请求：



不过这并不是什么大问题，多一点请求和流量，功能这块可以不影响的。

## 四、静态import更多细节

### 1. 目前import不支持裸露的说明符

目前import不支持裸露的说明符，用白话讲就是import的地址前面不能是光秃秃的。例如下面这些就不支持：

```
// 目前不支持，以后可能支持
import {foo} from 'bar.mjs';
import {foo} from 'utils/bar.mjs';
```

下面这些则支持，可以是根路径的 `/`，同级路径 `./` 亦或者是父级 `../`，甚至完整的非相对地址也是可以的。

```
// 支持
import {foo} from 'https://www.zhangxinxu.com/utils/bar.mjs';
import {foo} from '/utils/bar.mjs';
import {foo} from './bar.mjs';
import {foo} from '../bar.mjs';
```

### 2. 默认Defer行为

传统 `<script>` 属性支持一个名为 `defer` 的属性值，可以让JS资源异步加载，同时保持顺序。例如：

```
<!-- 同步 -->
<script src="1.js"></script>

<!-- 异步但顺序保证 -->
<script defer src="2.js"></script>
<script defer src="3.js"></script>
```

加载顺序一定是 `1.js`，`2.js`，`3.js`。我们只要看 `2.js` 和 `3.js`，由于设置了 `defer`，这两个JS异步加载，因此，就算 `1.js` 放在最下面，也多半 `1.js` 先加载完。而多个 `<script>` 同时设置 `defer` 会从前往后依次加载执行。因此，一定是先加载完 `2.js` 然后是 `3.js`。

回到本文的ES6 module导入，对于 `type="module"` 的 `<script>` 元素，天然外挂 `defer` 特性，也就是天然异步，所有module脚本按顺序，因此，下面这段脚本执行顺序就好理解了：

```
<!-- 此script稍后执行 -->
<script type="module" src="1.mjs"></script>

<!-- 硬加载嘛 -->
<script src="2.js"></script>

<!-- 比第一个要晚一点 -->
```

```
<script defer src="3.js"></script>
```

最终的加载执行顺序是：2.js，1.mjs，3.js。2.js 同步，解析这里就加载。1.mjs 虽然没有设置 defer，但默认 defer，因此和 3.js 其实是一样的，都是异步 defer 加载。由于 1.mjs 对于的 <script> 在 3.js 前面，因此，先 1.mjs 后 3.js。

相信不难理解。

### 3. 内联script同样defer特性

如下代码：

```
<script type="module">
  console.log("Inline module执行");
</script>

<script src="1.js"></script>

<script defer>
  console.log("Inline script执行");
</script>

<script defer src="2.js"></script>
```

最后的执行顺序是：1.js，Inline script，Inline module，2.js。

从[在线demo](#)控制台输出可以证明上面的结论。

原因在于，传统的内联 <script> 是没有 defer 这种概念的，从不异步，大家可以直接忽略，认为什么也没设置即可；而 type="module" 的 <script> 天然 defer。因此，先 1.js，Inline script；然后按照 defer 规则，从前往后依次是 Inline module，2.js。

### 4. 支持async

无论是内联的module <script> 还是外链的 <script>，都支持 async 这个异步标识属性。这个有别于传统的 <script>，也就是传统 <script> 仅外链JS才支持 async，内联JS直接忽略 async。

async 和 defer 都可以让JavaScript异步加载，区别在于 defer 保证执行顺序，而 async 谁先加载好谁先执行。这个特性表现在 type="module" 的 <script> 元素这里同样适用。

例如下面例子：

```
<!-- firstBlood模块一加载完就会执行 -->
<script async type="module">
  import { pColor } from './firstBlood.mjs';
  pColor('red');
</script>

<!-- doubleKill模块一加载完就会执行 -->
<script async type="module" src="./doubleKill.mjs"></script>
```

无论是 firstBlood.mjs 还是 doubleKill.mjs 都是异步加载，然后执行顺序不固定，有可能先 firstBlood.mjs，也有可能先 doubleKill.mjs，这样看哪个模块脚本先加载完毕。

## 5. 模块只会执行一次

传统的 `<script>` 如果引入的JS文件地址是一样的，则JS会执行多次。但是，对于 `type="module"` 的 `<script>` 元素，即使模块地址一模一样，也只会执行一次。例如：

```
<!-- 1.mjs只会执行一次 -->
<script type="module" src="1.mjs"></script>
<script type="module" src="1.mjs"></script>
<script type="module">
  import "./1.mjs";
</script>

<!-- 下面传统JS引入会执行2次 -->
<script src="2.js"></script>
<script src="2.js"></script>
```

我们看下[在线demo](#)控制台输出的结果，`2.js` 执行了2次，而 `1.mjs` 模块虽然3次引入，但只执行了一次。截图如下：



## 6. 总是CORS跨域

传统JS文件的加载，我们直接跨域也可以解析，例如，我们会使用一些大网站的CDN服务，例如，加载个百度提供的jQuery地址：

```
<script src="//apps.bdimg.com/libs/jquery/1.9.0/jquery.min.js"></script>
```

可以正常解析。但是，如果是module模式下 `import` 脚本资源，则不会执行，例如：

```
<script type="module" src="//apps.bdimg.com/.../jquery.min.js"></script>
<script>
window.addEventListener('DOMContentLoaded', function () {
  console.log(window.$);
});
</script>
```

我们使用Chrome浏览器跑一下[在线demo](#)，结果浏览器报CORS policy跨域相关错误，自然 `window.$` 是 `undefined`：



如何使支持跨域呢？

需要模块资源服务端配置 `Access-Control-Allow-Origin`，可以指定具体域名，或者直接使用 `*` 通配符，`Access-Control-Allow-Origin:*`。

本站cdn.zhangxinxu.com域名有配置 `Access-Control-Allow-Origin`，所以，下面代码打印出来的值就不是 `undefined`。

```
<script type="module" src="//cdn.zhangxinxu.com/study/js/jquery-1.4.2.min.js"></script>
<script>
window.addEventListener('DOMContentLoaded', function () {
  console.log(window.$);
});
</script>
```

访问[在线demo](#)，打开控制台，可以看到输出如下内容：

### 7. 无凭证

如果请求来自同一个源（域名一样），大多数基于CORS的API将发送凭证（如cookie等），但 `fetch()` 和模块脚本是例外 – 除非您要求，否则它们不会发送凭证。

我们通过下面例子理解上面这句话的含义：

```
<!-- ① 获取资源会带上凭证（如cookie等） -->
<script src="1.js"></script>

<!-- ② 获取资源不带凭证 -->
<script type="module" src="1.mjs"></script>

<!-- ③ 获取资源带凭证 -->
<script type="module" crossorigin src="1.mjs?"></script>

<!-- ④ 获取资源不带凭证 -->
<script type="module" crossorigin src="//cdn.zhangxinxu.com/.../1.mjs"></script>

<!-- ⑤ 获取资源带凭证 -->
<script type="module" crossorigin="use-credentials" src="//cdn.zhangxinxu.com/.../1.mjs?">
</script>
```

这里出现了一个HTML属性 `crossorigin`，该属性在“[解决canvas图片跨域问题](#)”这篇文章有介绍，可以明确 `<script>` 以及 `<img>` 等可外链元素在获取资源时候，是否带上凭证。

`crossOrigin` 可以有下面两个值：

关键字
释义
anonymous
元素的跨域资源请求不需要凭证标志设置。
use-credentials
元素的跨域资源请求需要凭证标志设置，意味着该请求需要提供凭证。

其中，只要 `crossOrigin` 的属性值不是 `use-credentials`，全部都会解析为 `anonymous`。

回到本节案例。

- 1. 传统JS加载，都是默认带凭证的（对应注释①）。
- 2. module模块加载默认不带凭证（注释②）。
- 3. 如果我们设置 `crossOrigin` 为匿名 `anonymous`，又会带凭证（注释③）。
- 4. 如果import模块跨域，则设置 `crossOrigin` 为 `anonymous` 不带凭证（注释④）。
- 5. 如果import模块跨域，且明确设置 `crossOrigin` 为使用凭证 `use-credentials`，则带凭证（注释⑤）。

注意，如果跨域，需要同时服务器侧返回 `Access-Control-Allow-Credentials:true` 头信息。

然后，上面的凭证规则以后有可能会调整，欢迎大家及时反馈。

## 8. 天然严格模式

import的JS模块代码天然严格模式，如果里面有不太友好的代码会报错，例如：

## 四、动态import

静态import在首次加载时候会把全部模块资源都下载下来，但是，我们实际开发时候，有时候需要动态import（dynamic import），例如点击某个选项卡，才去加载某些新的模块，这个动态import特性浏览器也是支持的。

具体是使用一个长得像函数的 `import()`，注意，只是长得像函数，`import()` 实际上就是个单纯的语法，类似于 `super()`。这就意味着 `import()` 不会从 `Function.prototype` 获得继承，因此您无法 `call` 或 `apply` 它，并且 `const importAlias = import` 之类的东西不起作用，甚至 `import()` 都不是对象！

语法为：

```
import(moduleSpecifier);
```

`moduleSpecifier` 为模块说明符，其实就是模块地址，规则和静态 `import` 一样，不能是裸露的地址。

### 案例

静态 `import()` 那个红色翻转案例我们改造成动态 `import`，也就是把 `import xxxx from 'xxxx'` 改成 `import('xxxx')`，代码如下：

```
<script type="module">
  // 导入doubleKill模块
  import('./doubleKill.mjs').then((module) => {
    // 执行默认方法
    module.default();
    // 设置颜色为红色
    module.pColor('red');
  });
</script>
```

最后效果和静态import一样：

您可以狠狠地点击这里：[ES6动态import模块基本使用demo](#)

由于 `import()` 返回一个 `promise`，所以，我们可以使用 `async/await` 来代替 `then` 这种回调形式。

```
<script type="module">
(async () => {
  // 导入doubleKill模块
  const module = await import('./doubleKill.mjs');
  // 执行默认方法
  module.default();
  // 设置颜色为红色
  module.pColor('red');
})
```



```
}O;  
</script>
```

您可以狠狠地点击这里：[async/await下的动态import演示demo](#)

## 五、交互中的动态import

不像静态 `import` 只能用在 `<script type="module">` 一样，动态 `import()` 也可以用在普通的 `script`，我们来看一个更接近真实开发的案例——选项卡内容动态加载。

首先，页面HTML代码如下：

```
<nav>  
  <a href="javascript:" class="active" data-module="mm1">美女1</a>  
  <a href="javascript:" data-module="mm2">美女2</a>  
  <a href="javascript:" data-module="mm3">美女3</a>  
</nav>  
<main></main>
```

需求如下，点击不同的美女选项卡的时候，去加载对应的模块，模块有个方法可以改变 `<main>` 元素内容。

则，我们的交互JS和动态 `import()` JS如下：

```
<script>  
  const main = document.querySelector('main');  
  const links = document.querySelectorAll('nav > a');  
  for (const link of links) {  
    link.addEventListener('click', async (event) => {  
      const module = await import(`./${link.dataset.module}.mjs`);  
      // 模块暴露名为`loadPageInto`的方法，内容是写入一段HTML  
      module.loadPageInto(main);  
    });  
  }  
</script>
```

结果，当我们点击其他选项卡的时候，`<main>` 元素中的美女图片就会发生变化，例如默认是这个：



点击“美女2”选项卡按钮，此时浏览器会动态加载 `mm2.mjs` 这个模块，然后执行这个模块中暴露的 `loadPageInfo` 方法，从而改变呈现内容。



您可以狠狠地点击这里：[选项卡模块动态import demo](#)

## 六、结语

这篇文章写了一个月，从7月30号写到8月2号，是不是跨了一个月？😄

最近看自己很多年前写的技术文章，不太正经，插科打诨的东西比较多，甚至有时候会花一般篇幅讲一个不知所云的故事。后来，有人说啰嗦，于是自己文风尝试简洁，持续了差不多2年，最近发现这样不行，完全就成了干巴巴的技术科普，很无聊，很没劲，没有辨识度，缺少有趣的灵魂，时间流逝，很容易湮没在茫茫多的技术洪流中，所以呢，决定，还是回到过去，本站就是个个人网站，所谓个人网站，不

就是用来展示自己的特质的嘛，精神无限自由的自留地，不必因为某些言论而局促自己。

哎呀呀呀，这世上很多事情都是这样，实践了一圈下来，发现，还是最初的决策是最准确的。

#### 参考文章

- [ECMAScript modules in browsers](#)
- [dynamic-import](#)
- [Using JavaScript modules on the web](#)

感谢阅读，行文仓促，如果文中有表述不准的地方，欢迎指正。

《CSS世界》签名版独家发售，包邮，可指定寄语，点击显示购买码

(本篇完) // 想要打赏? 点击[这里](#)。有话要说? 点击[这里](#)。



« [隐私相关-了解HTML5 Do Not Track API](#)

[原来浏览器原生支持JS Base64编码解码](#) »

#### 猜你喜欢

- [Service Worker实现浏览器端页面渲染或CSS,JS编译](#)
- [博闻强识：了解CSS中的@ AT规则](#)
- [JS一般般的网页重构可以使用Node.js做些什么](#)
- [CSS的样式合并与模块化](#)
- [页面重构“鑫三无准则”之“无宽度”准则](#)
- [高富帅seajs使用示例及spm合并压缩工具露脸](#)
- [基于HTML5 drag/drop模块拖动插入排序删除完整实例](#)
- [ES6 JavaScript Promise的感性认知](#)
- [HTMLUnknownElement与HTML5自定义元素的故事](#)
- [简单了解ES6/ES2015 Symbol\(\) 方法](#)
- [HTML5 datalist在实际项目中应用的可行性研究](#)

分享到: 1

标签: [@import](#), [async](#), [await](#), [ES6](#), [export](#), [module](#), [模块化](#), [浏览器](#)

#### 发表评论 (目前22条评论)

<input type="text"/>	名称 (必须)
<input type="text"/>	邮件地址(不会被公开) (必须)
<input type="text"/>	网站

提交评论

1. zento说道:

2019年01月2日 22:25

学习了

[回复](#)



2. yyyyyy说道:

2018年10月19日 10:30

哥，你的特色就是无处不在的美女图片和各种美女图片。。

[回复](#)



老牛说道:

2018年10月24日 11:03

google前端工程师之前写过script module的支持情况。看到这篇文章介绍了基于peg.js可以自己实现这种module级别的http code loader， 传送门：<http://www.fed123.com/dsl-ast-peg/>

[回复](#)



3. xgqfrms说道:

2018年09月4日 10:23

# markdown title testing

点个赞，汇总的比较详细。

虽然开发中，文中提到的好多坑已经踩过了。

<http://2ality.com/2017/01/import-operator.html#async-functions-and-import>

可以看下，这篇文章也不错!

[回复](#)



xgqfrms说道:

2018年09月4日 11:00

# css trick ???

<https://github.com/xgqfrms/FEIQA/issues/31#issuecomment-418226565>

> -1， 费解

```
transform: scale(-1);
transform: scaleX(-1);
transform: scaleY(-1);
```

# good

> 语义化，好理解

```
transform: rotate(180deg);
transform: rotateX(180deg);
transform: rotateY(180deg);
```

[回复](#)



4. **lzx**说道:

2018年08月15日 11:17

今天刚发现import返回的是promise，百度了一下结果很少有讲到这个点的，看到你这篇文章大致明白了，感谢大神。  
我个人觉得，风格幽默是好事，比如偶尔抖个包袱什么的，但是大量穿插一些无关的内容，读起来确实会分心，感觉你早期的文章就有点这个情况，不过这篇看得挺舒服的，当然像你说的，不必因为别人的评论而局促自己。

[回复](#)



5. **jason**说道:

2018年08月10日 14:50

大佬，什么时候给个webpack跟vue的课程呢

[回复](#)



6. **帅哥胡**说道:

2018年08月9日 10:04

vue 一直用的就是这

[回复](#)



7. **鹿啦啦**说道:

2018年08月8日 16:25

太久没有来逛大神的博客了，受益匪浅

[回复](#)



8. **张允**说道:

2018年08月6日 16:19

赞

[回复](#)



9. **shoyuf**说道:

2018年08月6日 14:31

caniuse 支持 embed 加载，求老大别再截图啦

[回复](#)



10. **lza**说道:

2018年08月5日 19:23

赞

[回复](#)



11. **DeathGhost**说道:

2018年08月3日 21:46

这个好，Angular中用过...

[回复](#)



12. **Colin**说道:

2018年08月3日 14:48

赞坑!!!!!!



[回复](#)

13. **Tim**说道:

2018年08月2日 15:33

这个必须马上立即迅速的阅读并收藏一下。

[回复](#)



14. **鸡肋**说道:

2018年08月2日 14:59

食之无味,弃之可惜

[回复](#)



15. **Arther**说道:

2018年08月2日 12:32

node.js并不是支持import，是经过babel转译

[回复](#)



16. **Tony**说道:

2018年08月2日 11:33

在控制台

直接运行

```
await import('./doubleKill.mjs');
```

会抛异常,

直接运行

```
var im = async src=>import(src); await im('./doubleKill.mjs');
```

也抛异常,

分两次运行

```
var im = async src=>import(src);
```

和

```
await im('./doubleKill.mjs');
```

却不会呢

[回复](#)



**lzx**说道:

2018年08月15日 11:07

await不能直接使用，必须在async函数中使用

[回复](#)



17. **zc**说道:

2018年08月2日 11:09

踩个沙发

[回复](#)



18. **郭二蛋**说道:

2018年08月2日 10:14

学到了很多，谢谢旭叔

[回复](#)



## 19. Dont说道:

2018年08月2日 09:44

倒不觉得啰嗦，篇幅也不算长，适合我这种有点阅读障碍的人看，看完也get到了知识点。棒

[回复](#)



### 最新文章

- » [常见的CSS图形绘制合集](#)
- » [粉丝群第1期CSS小测点评与答疑](#)
- » [分享三个纯CSS实现26个英文字母的案例](#)
- » [小tips: 纯CSS实现打字动画效果](#)
- » [CSS/CSS3 box-decoration-break属性简介](#)
- » [CSS :placeholder-shown伪类实现Material Design占位符交互效果](#)
- » [从天猫某活动视频不必要的3次请求说起](#)
- » [CSS vector-effect与SVG stroke描边缩放](#)
- » [CSS ::backdrop伪元素是干嘛用的?](#)
- » [周知: CSS -webkit-伪元素选择器不再导致整行无效](#)

### 今日热门

- » [常见的CSS图形绘制合集](#) (179)
- » [粉丝群第1期CSS小测点评与答疑](#) (113)
- » [未来必热: SVG Sprite技术介绍](#) (111)
- » [HTML5终极备忘大全 \(图片版+文字版\)](#) (85)
- » [让所有浏览器支持HTML5 video视频标签](#) (83)
- » [Selectivizr-让IE6~8支持CSS3伪类和属性选择器](#) (80)
- » [CSS3下的147个颜色名称及对应颜色值](#) (78)
- » [小tips: 纯CSS实现打字动画效果](#) (73)
- » [写给自己看的display: flex布局教程](#) (70)
- » [分享三个纯CSS实现26个英文字母的案例](#) (70)

### 今年热议

- » [《CSS世界》女主角诚寻靠谱一起奋斗之人](#) (76)
- » [不借助Echarts等图形框架原生JS快速实现折线图效果](#) (64)
- » [看, for..in和for..of在那里吵架!](#) (60)
- » [是时候好好安利下LuLu UI框架了!](#) (47)
- » [原来浏览器原生支持JS Base64编码解码](#) (35)
- » [妙法攻略: 渐变虚框及边框滚动动画的纯CSS实现](#) (33)
- » [炫酷H5中序列图片视频化播放的高性能实现](#) (31)
- » [CSS scroll-behavior和JS scrollIntoView让页面滚动平滑](#) (30)
- » [windows系统下批量删除OS X系统.DS\\_Store文件](#) (26)
- » [写给自己看的display: flex布局教程](#) (26)

### 猜你喜欢

- [Service Worker实现浏览器端页面渲染或CSS,JS编译](#)
- [博闻强识: 了解CSS中的@ AT规则](#)
- [JS一般般的网页重构可以使用Node.js做些什么](#)

- CSS的样式合并与模块化
- 页面重构“鑫三无准则”之“无宽度”准则
- 高富帅seajs使用示例及spm合并压缩工具露脸
- 基于HTML5 drag/drop模块拖动插入排序删除完整实例
- ES6 JavaScript Promise的感性认知
- HTMLUnknownElement与HTML5自定义元素的故事
- 简单了解ES6/ES2015 Symbol() 方法
- HTML5 datalist在实际项目中应用的可行性研究