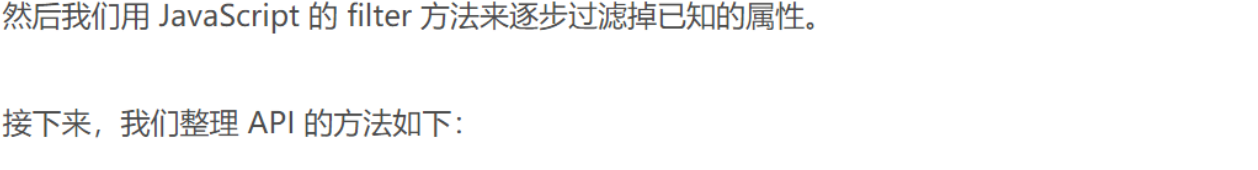


37 | 浏览器API (小实验)：动手整理全部API

winter 2019-04-18



你好，我是 winter。今天我们来讲讲浏览器 API。浏览器的 API 数目繁多，我们在之前的课程中，已经一起学习过其中几个比较有体系的部分：比如之前讲到过的 DOM 和 CSSOM 等等。但是，如果你留意过，会发现我们讲到的 API 仍然是标准中非常小的一部分。

这里，我们不再把课程变成一本厚厚的 API 参考手册，所以这一节课，我设计了一个实验，我们一起来给 API 分分类。

我们按照每个 API 所在的标准来分类。所以，我们用代码来反射浏览器环境中全局对象的属性，然后我们用 JavaScript 的 filter 方法来逐步过滤掉已知的属性。

接下来，我们整理 API 的方法如下：

- 从 Window 的属性中，找到 API 名称；
- 查阅 MDN 或者 Google，找到 API 所在的标准；
- 阅读标准，手工或者用代码整理出标准中包含的 API；
- 用代码在 Window 的属性中过滤掉标准中涉及的 API。

重复这个过程，我们可以找到所有的 API 对应的标准。首先我们先把前面已经讲过的 API 过滤掉。

##JavaScript 中规定的 API

大部分的 API 属于 Window 对象（或者说全局对象），我们可以用反射来看一看现行浏览器中已经实现的 API，我这里使用 Mac 下的 Chrome 72.0.3626.121 版本。

我们首先调用 Object.getOwnPropertyNames(window)。在我的环境中，可以看到，共有 821 个属性。

这里包含了 JavaScript 标准规定的属性，我们做一下过滤：

```
1 {
2   let js = new Set();
3   let objects = ["BigInt", "BigInt64Array", "BigUint64Array", "Infinity", "NaN", "und
4   objects.forEach(o => js.add(o));
5   let names = Object.getOwnPropertyNames(window)
6   names = names.filter(e => !js.has(e));
7 }
8
```

这一部分我们已经在 JavaScript 部分讲解过了（JavaScript 对象：你知道全部的对象分类吗），所以这里我就采用手工的方式过滤出来。

DOM 中的元素构造器

接下来我们看看已经讲过的 DOM 部分，DOM 部分包含了 document 属性和一系列的构造器，我们可以用 JavaScript 的 prototype 来过滤构造器。

```
1 names = names.filter( e => {
2   try {
3     return !(window[e].prototype instanceof Node)
4   } catch(err) {
5     return true;
6   }
7 }).filter( e => e != "Node")
8
```

这里我们把所有 Node 的子类都过滤掉，再把 Node 本身也过滤掉，这是非常大的一批了。

Window 对象上的属性

接下来我们要找到 Window 对象的定义，我们在下面链接中可以找到。

- <https://html.spec.whatwg.org/#window>

这里有一个 Window 接口，是使用 WebIDL 定义的，我们手工把其中的函数和属性整理出来，如下：

```
1
2 window,self,document,name,location,history,customElements,locationbar,menubar, persona
3
```

接下来，我们编写代码，把这些函数和属性，从浏览器 Window 对象的属性中去掉，JavaScript 代码如下：

```
1 {
2   let names = Object.getOwnPropertyNames(window)
3   let js = new Set();
4   let objects = ["BigInt", "BigInt64Array", "BigUint64Array", "Infinity", "NaN", "und
5   objects.forEach(o => js.add(o));
6   names = names.filter(e => !js.has(e));
7
8   names = names.filter( e => {
9     try {
10      return !(window[e].prototype instanceof Node)
11    } catch(err) {
12      return true;
13    }
14  }).filter( e => e != "Node")
15
16   let windowprops = new Set();
17   objects.forEach( "window", "self", "document", "name", "location", "history", "customElem
18   objects.forEach(o => windowprops.add(o));
19   names = names.filter(e => !windowprops.has(e));
20 }
21
```

webkit 前缀的私有属性我们也过滤掉：

```
1 names = names.filter( e => !e.match(/^webkit/))
2
```

除此之外，我们在 HTML 标准中还能找到所有的接口，这些我们也过滤掉：

```
1
2 let interfaces = new Set();
3 objects = ["ApplicationCache", "AudioTrack", "AudioTrackList", "BarProp", "BeforeUn
4 objects.forEach(o => interfaces.add(o));
5
6 names = names.filter(e => !interfaces.has(e));
7
8
```

这样过滤之后，我们已经过滤掉了所有的事件、Window 对象、JavaScript 全局对象和 DOM 相关的属性，但是，竟然还剩余了很多属性！你是不是很惊讶呢？好了，接下来我们才进入今天的正题。

其它属性

这些既不属于 Window 对象，又不属于 JavaScript 语言的 Global 对象的属性，它们究竟是什么呢？

我们可以一个一个来查看这些属性，来发现一些我们以前没有关注过的标准。

首先，我们要把过滤的代码做一下抽象，写成一个函数：

```
1 function filterOut(names, props) {
2   let set = new Set();
3   props.forEach(o => set.add(o));
4   return names.filter(e => !set.has(e));
5 }
6
```

每次执行完 filter 函数，都会剩下一些属性，接下来，我们找到剩下的属性来看一下。

ECMAScript 2018 Internationalization API

在我的浏览器环境中，第一个属性是：Intl。

查找这些属性来历的最佳文档是 MDN，当然，你也可以使用 Google。

总之，经过查阅，我发现，它属于 ECMA402 标准，这份标准是 JavaScript 的一个扩展，它包含了国际化相关的内容：

- <http://www.ecma-international.org/ecma-402/5.0/index.html#Title>

ECMA402 中，只有一个全局属性 Intl，我们也把它过滤掉：

```
1 names = names.filter(e => e != "Intl")
2
```

再来看看还有什么属性。

Streams 标准

接下来我看到的属性是：ByteLengthQueuingStrategy。

同样经过查阅，它来自 WHATWG 的 Streams 标准：

<https://streams.spec.whatwg.org/#blqs-class>

不过，跟 ECMA402 不同，Streams 标准中还有一些其它属性，这里我手工查阅了这份标准，并做了整理。

接下来，我们用代码把它们跟 ByteLengthQueuingStrategy 一起过滤掉：

```
1 names = filterOut(names, ["ReadableStream", "ReadableStreamDefaultReader", "ReadableStr
2
```

好了，过滤之后，又少了一些属性，我们继续往下看。

WebGL

接下来我看到的属性是：WebGLContextEvent。

显然，这个属性来自 WebGL 标准：

- <https://www.khronos.org/registry/webgl/specs/latest/1.0/#5.15>

我们在这份标准中找到了一些别的属性，我们把它一起过滤掉：

```
1 names = filterOut(names, ["WebGLContextEvent", "WebGLObject", "WebGLObject", "WebGLFrame
2
```

过滤掉 WebGL，我们继续往下看。

Web Audio API

下一个属性是 WaveShaperNode，这个属性名听起来就跟声音有关，这个属性来自 W3C 的 Web Audio API 标准。

我们来看一下标准：

- <https://www.w3.org/TR/webaudio/>

```
1 ["AudioContext", "AudioNode", "AnalyserNode", "AudioBuffer", "AudioBufferSourceNode", ".
2
```

于是我们把它们也过滤掉：

```
1
2 names = filterOut(names, ["AudioContext", "AudioNode", "AnalyserNode", "AudioBuffer", ".
3
```

我们继续看下一个属性。

Encoding 标准

在我的环境中，下一个属性是 TextDecoder，经过查阅得知，这个属性也来自一份 WHATWG 的标准，Encoding：

- <https://encoding.spec.whatwg.org/#dom-textencoder>

这份标准仅仅包含四个接口，我们把它它们过滤掉：

```
1 names = filterOut(names, ["TextDecoder", "TextEncoder", "TextDecoderStream", "TextEncod
2
```

我们继续看下一个属性。

Web Background Synchronization

下一个属性是 SyncManager，这个属性比较特殊，它并没有被标准化，但是我们仍然可以找到它的来源文档：

- <https://wicg.github.io/BackgroundSync/spec/#sync-manager-interface>

这个属性我们就不多说了，过滤掉就好了。

Web Cryptography API

我们继续看下去，下一个属性是 SubtleCrypto，这个属性来自 Web Cryptography API，也是 W3C 的标准。

- <https://www.w3.org/TR/WebCryptoAPI/>

这份标准中规定了三个 Class 和一个 Window 对象的扩展，给 Window 对象添加了一个属性 crypto。

```
1 names = filterOut(names, ["CryptoKey", "SubtleCrypto", "Crypto", "crypto"]);
2
```

我们继续来看。

Media Source Extensions

下一个属性是 SourceBufferList，它来自于：

- <https://www.w3.org/TR/media-source/>

这份标准中包含了三个接口，这份标准还扩展了一些接口，但是没有扩展 window。

```
1 names = filterOut(names, ["MediaSource", "SourceBuffer", "SourceBufferList"]);
2
```

我们继续看下一个属性。

The Screen Orientation API

下一个属性是 ScreenOrientation，它来自 W3C 的 The Screen Orientation API 标准：

- <https://www.w3.org/TR/screen-orientation/>

它里面只有 ScreenOrientation 一个接口，也是可以过滤掉的。

结语

到 Screen Orientation API，我这里看到还剩 300 余个属性没有处理，剩余部分，我想把它留给大家自己来完成。

我们可以看到，在整理 API 的过程中，我们可以找到各种不同组织的标准，比如：

- ECMA402 标准来自 ECMA；
- Encoding 标准来自 WHATWG；
- WebGL 标准来自 Khronos；
- Web Cryptography 标准来自 W3C；
- 还有些 API，根本没有被标准化。

浏览器环境的 API，正是这样复杂的环境。我们平时编程面对的环境也是这样的—一个环境。

所以，面对如此繁复的 API，我建议在系统掌握 DOM、CSSOM 的基础上，你可以仅仅做大概的浏览和记忆，根据实际工作需要，选择其中几个来深入学习。

做完这个实验，你对 Web API 的理解应该会有很大提升。

这一节课的问题就是完成所有的 API 到标准的归类，不同的浏览器环境应该略有不同，欢迎你把自己的结果留言一起讨论。