37-浏览器API（小实验）：动手整理全部API

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，已经一起学习了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，我们不可能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：

```
{
    let js = new Set();
    let objects = ["BigInt", "BigInt64Array", "BigUint64Array", "Infinity", "NaN", "undefined", "eval", "is
    objects.forEach(o => js.add(o));
    let names = Object.getOwnPropertyNames(window)
    names = names.filter(e => !js.has(e));
}
```

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。

```
    names = names.filter( e => {
        try {
            return !(window[e].prototype instanceof Node)
        } catch(err) {
            return true;
        }
    }).filter( e => e != "Node")
```

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：

```
 window,self,document,name,location,history,customElements,locationbar,menubar, personalbar,scrollbars,stat
```

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：

```
{
    let names = Object.getOwnPropertyNames(window)
    let js = new Set();
    let objects = ["BigInt", "BigInt64Array", "BigUint64Array", "Infinity", "NaN", "undefined", "eval", "is
    objects.forEach(o => js.add(o));
    names = names.filter(e => !js.has(e));

    names = names.filter( e => {
        try {
            return !(window[e].prototype instanceof Node)
        } catch(err) {
            return true;
        }
    }).filter( e => e != "Node")

    let windowprops = new Set();
    objects = ["window", "self", "document", "name", "location", "history", "customElements", "locationbar"
    objects.forEach(o => windowprops.add(o));
    names = names.filter(e => !windowprops.has(e));
}
```

我们还要过滤掉所有的事件，也就是on开头的属性。

```
names = names.filter( e => !e.match(/^on/))
```

webkit前缀的私有属性我们也过滤掉：

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

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：

```
    let interfaces = new Set();
    objects = ["ApplicationCache", "AudioTrack", "AudioTrackList", "BarProp", "BeforeUnloadEvent", "Broadca
    objects.forEach(o => interfaces.add(o));

    names = names.filter(e => !interfaces.has(e));
```

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：

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

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看。

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：

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

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：

```
names = filterOut(names, ["ReadableStream", "ReadableStreamDefaultReader", "ReadableStreamBYOBReader", "Rea
```

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：

```
names = filterOut(names, ["WebGLContextEvent","WebGLObject", "WebGLBuffer", "WebGLFramebuffer", "WebGLProgr
```

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/

Web Audio API中有大量的属性，这里我用代码做了过滤。得到了以下列表：

```
["AudioContext", "AudioNode", "AnalyserNode", "AudioBuffer", "AudioBufferSourceNode", "AudioDestinationNode
```

于是我们把它们也过滤掉：

```
names = filterOut(names, ["AudioContext", "AudioNode", "AnalyserNode", "AudioBuffer", "AudioBufferSourceNod
```

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，我们把它们过滤掉：

```
names = filterOut(names, ["TextDecoder", "TextEncoder", "TextDecoderStream", "TextEncoderStream"]);
```

我们继续来看下一个属性。

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。

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

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。

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

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。

精选留言：

- mfist 2019-04-18 06:42:24
  1. 通过老师的课，感觉慢慢会去翻标准了，之前学习没有见过的API，只是到MDN为止。
  2. 浏览器器中大多数的对象都原型继承自Object，是否可以根据原型继承关系 将window上面的api绘制成一颗树？有了这些继承关系 是否更容易理清这些全局属性呢。