1

ECMAScript 6 简介

ECMAScript 6（以下简称 ES6）是 JavaScript 语言的下一代标准，正处在快速开发中，大部分已经完成，预计将于 2014 年底正式发布。Mozilla 将在这个标准的基础上，推出 JavaScript 2.0。

ES6 的目标，是使得 JavaScript 语言可以用来编写大型的复杂应用程序，成为企业级开发语言。

ECMAScript 和 JavaScript 的关系

ECMAScript 是 JavaScript 语言的国际标准，JavaScript 是 ECMAScript 的实现。

1996 年 11 月，JavaScript 的创造者 Netscape 公司，决定将 JavaScript 提交给国际标准化组织 ECMA，希望这种语言能够成为国际标准。次年，ECMA 发布 262 号标准文件（ECMA-262）的第一版，规定了浏览器脚本语言的标准，并将这种语言称为 ECMAScript。这个版本就是 ECMAScript 1.0 版。

之所以不叫 JavaScript，有两个原因。一是商标，Java 是 Sun 公司的商标，根据授权协议，只有 Netscape 公司可以合法地使用 JavaScript 这个名字，且 JavaScript 本身也已被 Netscape 公司注册为商标。二是想体现这门语言的制定者是 ECMA，而不是 Netscape，这样有利于保证这门语言的开放性和中立性。因此，ECMAScript 和 JavaScript 的关系是，前者是后者的规格，后者是前者的一种实现。不过，在日常场合，这两个词是可以互换的。

ECMAScript 的历史

1998 年 6 月，ECMAScript 2.0 版发布。

1999 年 12 月，ECMAScript 3.0 版发布，成为 JavaScript 的通行标准，得到了广泛支持。

2007 年 10 月，ECMAScript 4.0 版草案发布，对 3.0 版做了大幅升级，原计划次年 8 月发布正式版本。然而在草案发布后，由于 4.0 版的目标过于激进，各方对于是否通过这个标准，产生了严重分歧。以 Yahoo、Microsoft、Google 为首的大公司，反对 JavaScript 的大幅升级，主张小幅改动；而以 JavaScript 创造者 Brendan Eich 为首的 Mozilla 公司，则坚持当前的草案。

2008 年 7 月，由于对于下一个版本应该包括哪些功能，各方分歧太大，争论过于激进，ECMA 开会决定，中止 ECMAScript 4.0 的开发，将其中涉及现有功能改善的一小部分，发布为 ECMAScript 3.1，而将其他激进的设想扩大范围，放入以后的版本，鉴于会议的气氛，该版本的项目代号取名为 Harmony（和谐）。会后不久，ECMAScript 3.1 就改名为 ECMAScript 5。

2009 年 12 月，ECMAScript 5.0 版正式发布。Harmony 项目则一分为二，一些较为可行的设想定名为 JavaScript.next 继续开发，后来演变成 ECMAScript 6；一些不是很成熟的设想，则被视为 JavaScript.next.next，在更远的将来再考虑推出。

2011 年 6 月，ECMAscript 5.1 版发布，并且成为 ISO 国际标准（ISO/IEC 16262:2011）。

2013 年 3 月，ECMAScript 6 草案冻结，不再添加新功能。新的功能设想将被放到 ECMAScript 7。

2013 年 12 月，ECMAScript 6 草案发布。此后是 12 个月的讨论期，以听取各方反馈意见。

2015 年 6 月，ECMAScript 6 预计将发布正式版本。

ECMA 的第 39 号技术专家委员会（Technical Committee 39，简称 TC39）负责制订 ECMAScript 标准，成员包括 Microsoft、Mozilla、Google 等大公司。TC39 的总体考虑是，ES5 与 ES3 基本保持兼容，较大的语法修正和新功能加入，将由 JavaScript.next 完成。当前，JavaScript.next 指的是 ES6，而当第六版发布以后，将指 ES7。TC39 估计，ES5 会在 2013 年的年中成为 JavaScript 开发的主流标准，并在今后五年中一直保持这个位置。

部署进度

由于 ES6 还没有定案，有些语法规则还会变动，目前支持 ES6 的软件和开发环境还不多。关于各大浏览器的最新版本对 ES6 的支持，可以查看*http://kangax.github.io/es5-compat-table/es6/*。

Google 公司的 V8 引擎已经部署了 ES6 的部分特性。使用 Node.js 0.11 版，就可以体验这些特性。

Node.js 的 0.11 版还不是稳定版本，需要使用版本管理工具 nvm（*https://github.com/creationix/nvm*）切换。操作如下，下载 nvm 以后，进入项目目录，运行下面的命令：

```
source nvm.sh
nvm use 0.11
node --harmony
```

启动命令中的 `--harmony` 选项可以打开所有已经部署的 ES6 功能。使用下面的命令，可以查看所有与 ES6 有关的单个选项。

```
$ node --v8-options | grep harmony
  --harmony_typeof
  --harmony_scoping
  --harmony_modules
  --harmony_symbols
  --harmony_proxies
  --harmony_collections
  --harmony_observation
  --harmony_generators
  --harmony_iteration
  --harmony_numeric_literals
  --harmony_strings
  --harmony_arrays
  --harmony_maths
  --harmony
```

Traceur 编译器

Google 公司的 Traceur（*https://github.com/google/traceur-compiler*）编译器，可以将 ES6 代码编译为 ES5 代码。

它有多种使用方式。

直接插入网页

Traceur 允许将 ES6 代码直接插入网页。

首先，必须在网页头部加载 Traceur 库文件。

```
<!-- 加载 Traceur 编译器 -->
<script src="http://google.github.io/traceur-compiler/bin/traceur
.js" type="text/javascript"></script>
<!-- 将 Traceur 编译器用于网页 -->
<script src="http://google.github.io/traceur-compiler/src/bootstr
ap.js" type="text/javascript"></script>
<!-- 打开实验选项，否则有些特性可能编译不成功 -->
<script>
    traceur.options.experimental = true;
</script>
```

接下来，就可以把 ES6 代码放入上面这些代码的下方。

```
<script type="module">
    class Calc {
        constructor(){
            console.log('Calc constructor');
        }
```

```
        add(a, b){
            return a + b;
        }
    }

    var c = new Calc();
    console.log(c.add(4,5));
</script>
```

正常情况下，上面的代码会在控制台打印出“9”。

注意，script 标签的 type 属性的值是 module，而不是 text/javascript。这是 Traceur 编译器用来识别 ES6 代码的标识，编译器会自动将所有标记了 type=module 的代码编译为 ES5 代码，然后交给浏览器执行。

如果 ES6 代码是一个外部文件，那么可以用 script 标签插入网页。

```
<script type="module" src="calc.js" >
</script>
```

在线转换

Traceur 提供一个在线编译器（*http://google.github.io/traceur-compiler/demo/repl.html*），可以在线将 ES6 代码转为 ES5 代码。转换后的代码，可以直接作为 ES5 代码插入网页运行。

上面的例子转为 ES5 代码运行，就是下面这个样子。

```
<script src="http://google.github.io/traceur-compiler/bin/traceur
```

```
.js" type="text/javascript"></script>
<script src="http://google.github.io/traceur-compiler/src/bootstr
ap.js" type="text/javascript"></script>
<script>
    traceur.options.experimental = true;
</script>
<script>
$traceurRuntime.ModuleStore.getAnonymousModule(function() {
    "use strict";

    var Calc = function Calc() {
        console.log('Calc constructor');
    };

    ($traceurRuntime.createClass)(Calc, {add: function(a, b) {
        return a + b;
    }}, {});

    var c = new Calc();
    console.log(c.add(4, 5));
    return {};
});
</script>
```

命令行转换

作为命令行工具使用时，Traceur 是一个 Node.js 的模块，首先需要用 npm 安装。

```
npm install -g traceur
```

安装成功后，就可以在命令行下使用 traceur 了。

traceur 直接运行 es6 脚本文件，会在标准输出中显示运行结果，以前面的 calc.js 为例。

```
$ traceur calc.js
Calc constructor
9
```

如果要将 ES6 脚本转为 ES5 代码，要采用下面的写法：

```
traceur --script calc.es6.js --out calc.es5.js
```

上面代码的 `--script` 选项用于指定输入文件，`--out` 选项用于指定输出文件。

为了防止有些特性编译不成功，最好加上 `--experimental` 选项。

```
traceur --script calc.es6.js --out calc.es5.js --experimental
```

命令行下转换得到的文件，可以放到浏览器中运行。

Node.js 环境的用法

Traceur 的 Node.js 用法如下（假定已安装 traceur 模块）。

```
var traceur = require('traceur');
var fs = require('fs');

// 将 ES6 脚本转为字符串
var contents = fs.readFileSync('es6-file.js').toString();

var result = traceur.compile(contents, {
```

```
  filename: 'es6-file.js',
  sourceMap: true,
  // 其他设置
  modules: 'commonjs'
});

if (result.error)
  throw result.error;

// result 对象的 js 属性就是转换后的 ES5 代码
fs.writeFileSync('out.js', result.js);
// sourceMap 属性对应 map 文件
fs.writeFileSync('out.js.map', result.sourceMap);
```

ECMAScript 7

2013 年 3 月，ES6 的草案封闭，不再接受新功能，新的功能将被加入 ES7。

ES7 可能包括的功能有：

1. **Object.observe**：对象与网页元素的双向绑定，只要其中之一发生变化，就会自动反映在另一方上。
2. **Multi-Threading**：多线程支持。目前，Intel 和 Mozilla 有一个共同的研究项目 RiverTrail，致力于让 JavaScript 多线程运行。预计这个项目的研究成果会被纳入 ECMAScript 标准。
3. **Traits**：它将是“类”功能（class）的一个替代。通过它，不同的对象可以分享同样的特性。

其他可能包括的功能还有：更精确的数值计算、改善的内存回收、增强的跨站点安全、类型化的更贴近硬件的低级别操作、国际化支持（Internationalization Support）、更多的数据结构，等等。