**ECMAScript6 入门**

**JavaScript的历史**

**诞生**

JavaScript诞生于1995年。起初它的主要目的是处理以前由服务器端负责的一些表单验证。时走在技术革新最前沿的Netscape（网景）公司，决定着手开发一种客户端语言，用来处理这种装简单的验证。当时就职于Netscape公司的布兰登·艾奇开始着手计划将1995年2月发布的LiveScript同时在浏览器和服务器中使用。

**Javascript于Java的关系**

为了赶在发布日期前完成LiveScript的开发，Netscape与Sun公司成立了一个开发联盟。而此时，Netscape为了搭上媒体热炒Java的顺风车，临时把LiveScript改名为JavaScript，所以从本质上来说JavaScript和Java没什么关系。

**Javascript的版本问题**

JavaScript 1.0获得了巨大的成功，Netscape随后在Netscape Navigator 3（网景浏览器）中发布了JavaScript 1.1。之后作为竞争对手的微软在自家的IE3中加入了名为JScript（名称不同是为了避免侵权）的JavaScript实现。而此时市面上意味着有3个不同的JavaScript版本，IE的JScript、网景的JavaScript和ScriptEase中的CEnvi。当时还没有标准规定JavaScript的语法和特性。随着版本的不同暴露的问题日益加剧，JavaScript的规范化最终被提上日程。

**JavaScript与ECMAScript 的关系**

1997年，以JavaScript1.1为蓝本的建议被提交给了欧洲计算机制造商协会（ECMA，European Computer Manufactures Association）该协会指定39号技术委员会负责将其进行标准化，TC39来此各大公司以及其他关注脚本语言发展的公司的程序员组成，经过数月的努力完成了ECMA-262——定义了一种名为ECMAScript的新脚本语言的标准。第二年，ISO/IEC（国标标准化组织和国际电工委员会）也采用了ECMAScript作为标准（即ISO/IEC-16262）。

**ECMAScript是什么？**

ECMAScript是Javascript语言的标准

ECMA European Computer Manufactures Association（欧洲计算机制造联合会），主要任务是研究信息和通讯技术方面的标准并发布有关技术报告

ECMAScript6：简称ES6，是JavaScript语言的下一代标准，也是目前正是发布的最新JavaScript标准，由于ES6是在2015年发布，所以ES6也成为ECMAScript2015

**ECMAScript的发展历史**

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，预计次年8月发布正式版本。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，发生了严重分歧。

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，由于会议的气氛，该版本的项目代号起名为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。然后是12个月的讨论期，听取各方反馈。

2015年6月17日，ECMAScript 6发布正式版本，即ECMAScript 2015。

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。

**运行ECMAScript6的途径**

* 浏览器（IE、Firefox、Chrome、Safari、Opera等）

浏览器对ES6的支持情况http://kangax.github.io/compat-table/es6/

* 使用Javascript引擎的系统（如Node.js）

Node.js是一个Javascript运行环境(runtime)。实际上它是对Google V8引擎进行了封装。V8引 擎执行Javascript的速度非常快，性能非常好。Google Chrome浏览器就是用的V8引擎

Node.js 下载地址<https://nodejs.org/zh-cn/>，目前支持到97%

* 使用Babel工具将ECMAScript6转换为ECMAScript5

Babel 是ES2015 语法转化器。这些转化器能让你现在就使用最新的 JavaScript 语法，而不用等待浏览器提供支持。官网地址：http://babeljs.io/

**NodeJs安装步骤**

* nvm（NodeJs版本管理工具）安装
* nvm配置淘宝镜像（下载更快）

修改settings.txt

root: D:\dev\nvm

path: D:\dev\nodejs

node\_mirror: http://npm.taobao.org/mirrors/node/

npm\_mirror: https://npm.taobao.org/mirrors/npm/

* 通过nvm安装对应版本的nodejs集（node、npm）

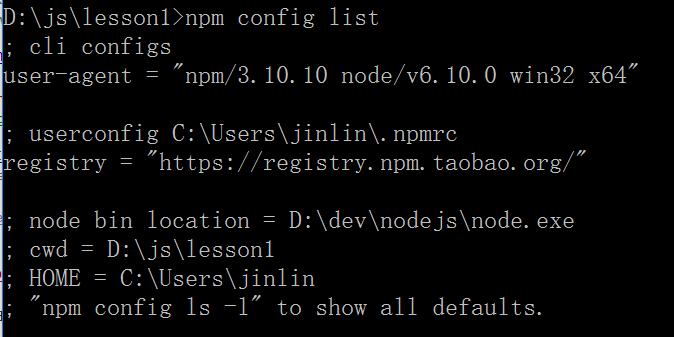
安装：nvm install 6.10.0

查看已经安装的node列表：nvm list

使用：nvm use 6.10.0

* npm设置淘宝镜像（下载更快）

npm config set registry <https://registry.npm.taobao.org>

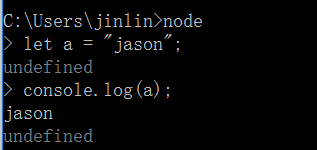


**NPM 是什么？**

npm是随同NodeJS一起安装的包管理工具，能解决NodeJS代码部署上的很多问题，常见的使用场景有以下几种：

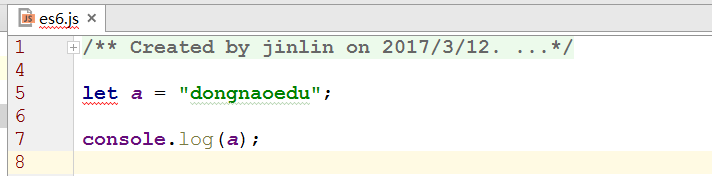
* 允许用户从NPM服务器下载别人编写的第三方包到本地使用。
* 允许用户从NPM服务器下载并安装别人编写的命令行程序到本地使用。
* 允许用户将自己编写的包或命令行程序上传到NPM服务器供别人使用。

**在NodeJs环境中运行ES6**

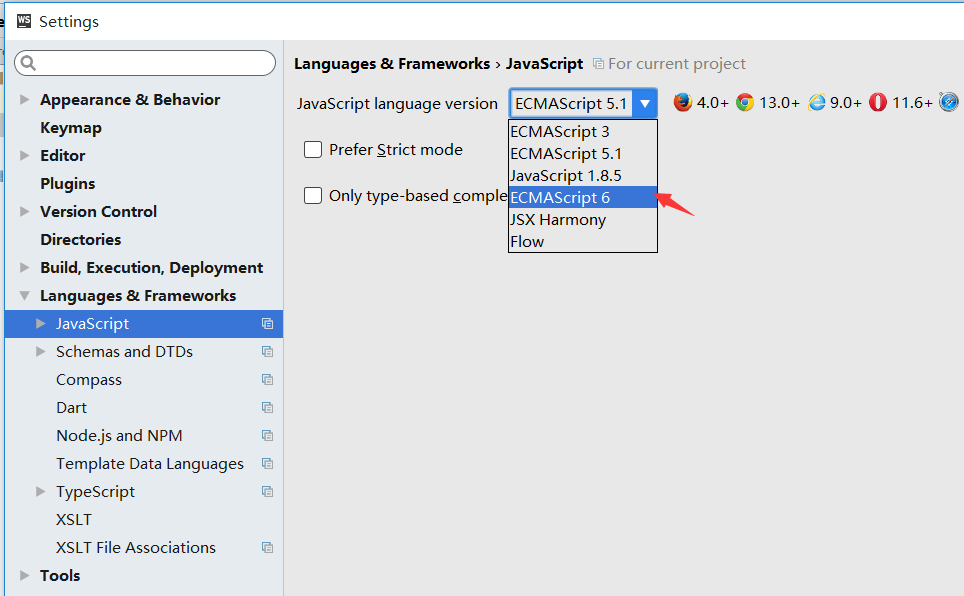


**在FireFox浏览器中运行ES6**

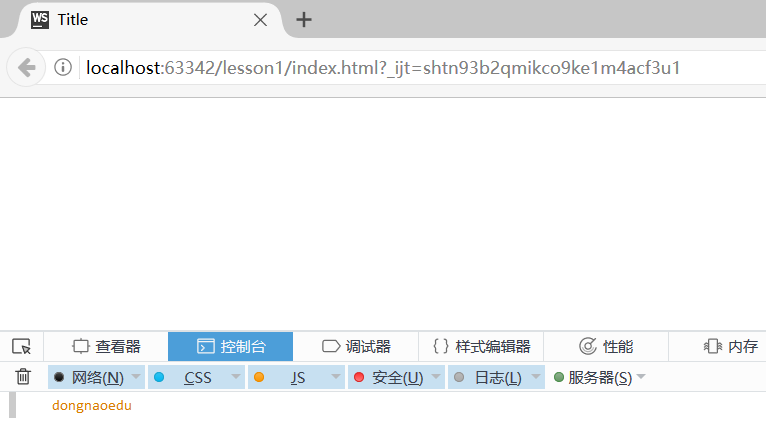
* 使用webstorm编写ES6代码



WebStorm配置支持ES6语法规则



* 运行结果

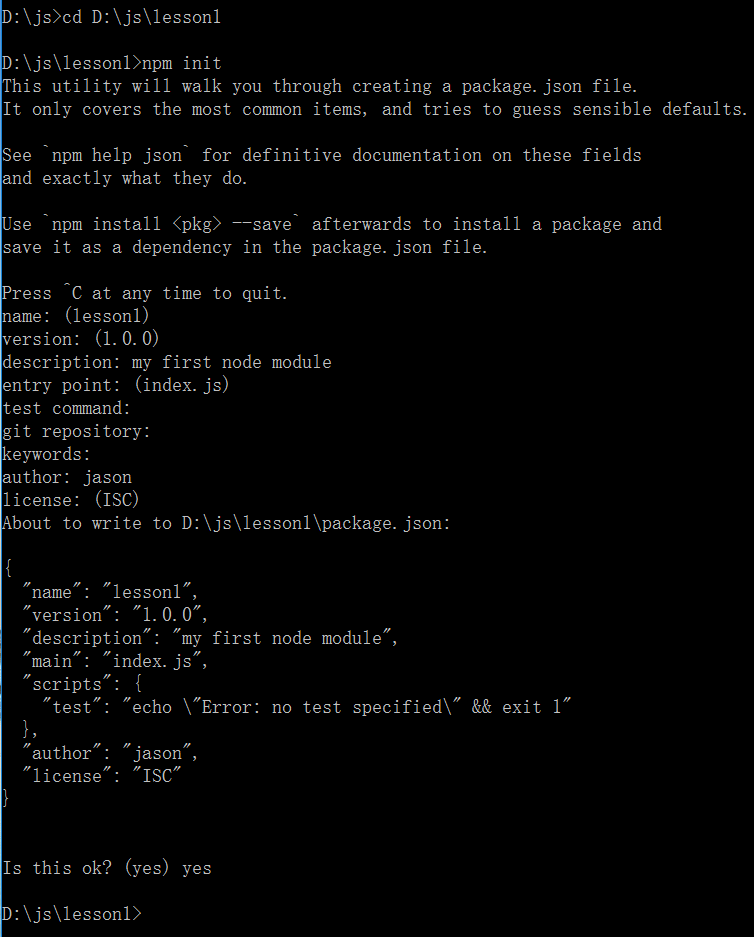


**使用Babel完成ES6到ES5的转换**

* babel全局安装

npm install -g babel-cli

* 使用npm创建一个nodejs工程，步骤如图：



* 在当前项目中安装转换插件（插件中的js代码，将用于转换）：

npm install babel-preset-es2015 –save

* 使用WebStorm打开第一步创建的nodejs工程，在工程中创建一个js文件，并编写ES6代码
* 手动将ES6转换成ES5

babel es6.js --out-file es5.js --presets es2015

**自动转换**

babel es6.js -w --out-file es5.js --presets es2015

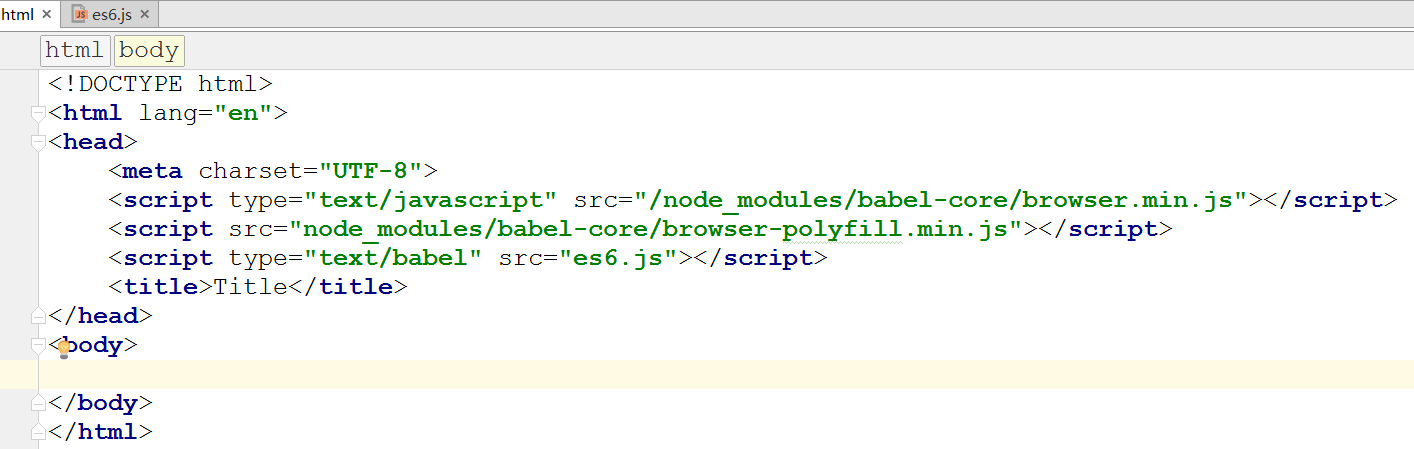
**在页面中实时转换ES6到ES5**

* 在当前项目安装babel-core插件（在网页上实时转换ES6到ES5）

npm install babel-core@5 –save

* HTML页面引入babel-core实时转换js代码

/node\_modules/babel-core/browser.min.js



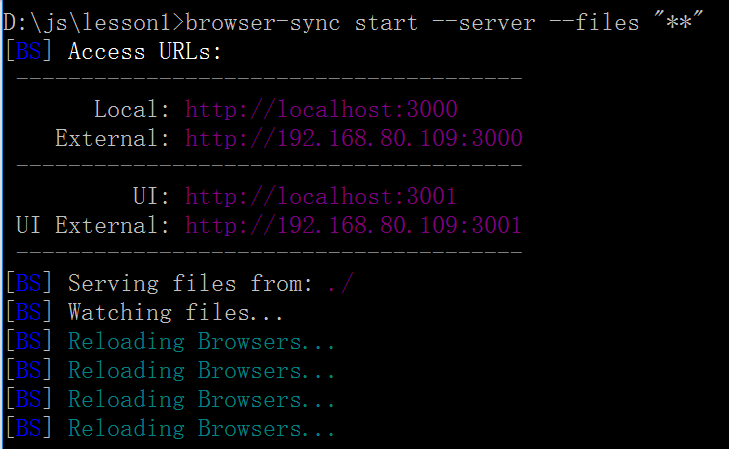
**使用Browsersync实时刷新页面和**

* Browsersync能让浏览器实时、快速响应您的文件更改（html、js、css、sass、less等）并自动刷新页面。
* 安装Browsersync

npm install -g browser-sync

* 在当前项目目录下启动Browsersync，开始监听

browser-sync start --server --files "\*\*"



**本地安装**

* 将安装包放在 ./node\_modules 下（运行 npm 命令时所在的目录），如果没有 node\_modules 目录，会在当前执行 npm 命令的目录下生成 node\_modules 目录。
* 可以通过 require() 来引入本地安装的包。

**全局安装**

* 将安装包放在 /usr/local 下或者你 node 的安装目录。
* 可以直接在命令行里使用。