# 一、范围

定义了ES脚本语言的实现标准。

# 二、一致性

对于实现ECMAScript的脚本语言，必须支持这本规范里所有的类型，值，对象，属性，函数和程序语法，和语义上的描述。对标准的一致实现，会在unicode标准下解释字符，v3.0或者之后的版本，和ISO/IEC 10646-1采用UCS-2或UTF-16作为编码格式实现level 3。如果采用的ISO/IEC 10646-1的子集没有指定，默认成BMP。如果编码方式也没有指定，默认是UTF-16。(注：更多javascript和编码的知识可以参考

http://speakingjs.com/es5/ch24.html)

对于ECMAScript的实现允许包含除本指南外的其他类型，值，对象，属性，函数。也被允许包含额外的程序和正则表达式语法。特识别对于保留字的支持（节7.6.1.2）也是允许的。

# 三、标准参考

下面的参考文档是不可缺少的。对于老旧的参考，会有相应的版本说明，对于新一些的标准，最新版本的语言会有所体现。

ISO/IEC9899：1996，C语言，包含了第一版修订和勘误表1和2

ISO/IEC10646-1：1993，信息技术-通用多位元码字符集（UCS）和其修订版

# 四、总览

这节是关于ecmascript的非标准概括。

ECMAScript是一门在宿主环境内用于计算和操作可计算对象的面向对象编程语言。这里定义的ECMAScript并非一门自给自足的语言，确实，这里没写需要的输入是什么以及期望的输出是什么。相反，这些应该由计算环境给出，不仅包括一些对象，属性，还有跟具体环境紧密相关的宿主对象。上下文环境的描述和行为超出了本指南的范畴，但是他们确实提供了脚本语言可调用的接口。

脚本语言是一门可以操作，定制，自动化系统设备的编程语言。在这些系统中，有用的功能已经可以通过用户界面展现，脚本语言是为程序暴露功能的一种机制。所以，宿主环境为语言提供基础设施，包括对象、函数，从而完善脚本语言的能力。

ES是一门web脚本语言，它能够使浏览器中的网页动起来，并且可以把一部分服务端逻辑放到前端来做，形成一个CS架构的web应用。ES提供了核心的脚本功能，以至于其他的环境下这些功能依然适用，但不同环境下可能衍生出不同的其他功能。

## 4.1、web脚本

浏览器为ES提供了宿主环境，举个例子，有代表window的对象，menus的对象，有弹出窗pop-ups，对话框，文本区域，锚点，框架frames，历史记录列表history，cookie和input/output。此外，宿主环境提供了一种方式把要执行的脚本代码附加到特定的事件上，比如焦点的变化focus and blur，页面或图片的加载，卸载，发生错误和终止，选中区域，表单提交和其他的鼠标操作。出现在HTML中的脚本代码和用户界面元素，图片，文本混杂在一起。页面中的脚本负责对用户的动作给予回应，这样的app中没有main函数也是可以的，因为没有必要。

若服务器作为宿主环境那么会有另外一些计算型对象，比如代表请求的requests对象，代表客户端的clients对象和files对象；还会有共享和锁住数据的机制。把服务端脚本同客户端脚本一起使用，可以为定制化的web应用提供分布式的计算，可以放在服务端或者客户端。

每一个支持ES的浏览器或者服务器都提供了自己的宿主环境，和完整的ES执行环境。

## 4.2、语言概述

接下来是非正式的语言概述—并没有描述语言的方方面面。这些叙述并非标准的。

ECMAScript是基于对象的：基础特性和宿主的接口都是由对象的形式提供，ECMAScript程序就是一组通讯对象。一个ES对象是一些属性property的集合，每个属性都由若干特性attribute定义，标识这个属性如何使用---举个例子，当特性中的可写（writable）被设置成false的时候，任何想要改变这个属性的值的代码都会失败。属性也可以包含其他对象，原生值或者函数。原生值指的是下面内置的类型之一： Undefined，Null，Boolean，Number，String。即使是对象，也是内置对象类型Object的成员。函数是一个可调用（callable）的对象。若一个函数成为了一个对象的属性，则称之为方法（method） 。

ES还定义了一系列内置对象。这些对象包括全局对象，Object对象，Function对象，Array对象，String对象，Boolean对象，Number对象，Math对象，Date对象，RegExp对象，JSON对象，和一些Error对象，包括EvalError，RangeError，ReferenceError，SyntaxError，TypeError和URIError。

ES也定义了一些内置的操作符（operators）。ES操作符号包含多种一元操作符，乘法操作符，加法操作符，位移操作符，关系操作符，等值操作符，二进制位运算操作符，二进制逻辑运算操作符，赋值操作符和逗号操作符。

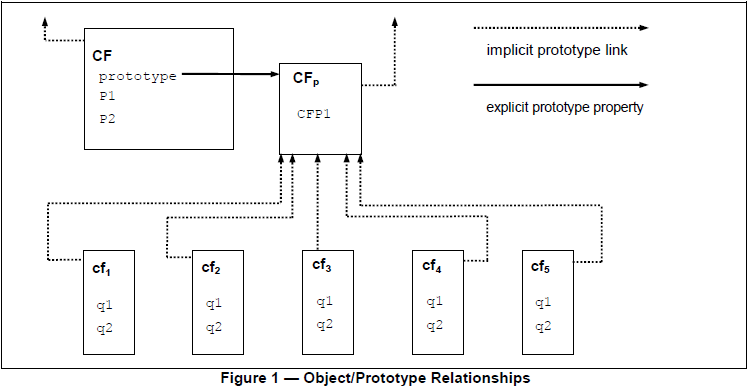
ES的语法基本和Java保持一致。ES的语法非常松散使得它可以成为易用的脚本语言。举个例子，变量并不要求有一个明确声明的类型，定义过的函数也不一定在调用之前一定要在字面上做声明（注：比如及时执行的匿名函数）

### 4.2.1 对象

ECMAScript没有像C++，Smalltalk，或者Java那样的类的概念。相反对象的创建可以用对象字面量或者构造函数。构造函数创建对象时执行的代码会把新对象的一些属性附上值，当然也可以不作任何事情。每个构造函数都是一个函数，并且有一个prototype属性，原型属性用于实现原型继承和在各个实例间共享一些状态和方法。用构造函数创建对象需要使用new操作符；举个例子，new Date(2009,11)就创建了一个新的Date对象。调用构造函数时没有用new操作符，返回什么对象取决于该函数的实现。举个例子，Date()会生成一个字符串表示当前的日期和时间，并非一个对象。

每个由构造函数生成的对象有一个隐式的引用，叫做对象的原型对象，此对象引用的是它构造函数的原型属性，即prototype对象。另外，这个原型对象也许（为什么说也许呢）会有它自己的原型对象，以此类推这就是所谓的原型链。当引用一个对象的某个属性时，首先检查该对象是否含有这个属性，若没有则会沿着原型链向上查找该属性，直到找到或没找到为止。

在基于类的面向对象语言里，状态state保存在实例里，方法保存在类Class里，继承只会作用于结构structure和行为behaviour。在ES里，状态和方法都保存在对象里，于是状态，结构和行为都会被继承。所有没有某个属性的对象，而恰巧他们的原型对象有这个属性时，它们就会共享这个属性，只要属性变了则这些对象都会变。图1表明了这个原则：



CF是一个都早函数（同时也是一个对象）， 五个对象都通过new CF()创建: cf1, cf2, cf3, cf4, and cf5。这些对象都包含q1和q2两个属性。虚线表明是一个隐含的原型继承关系；所以， cf3的原型对象是CFp。而构造函数CF，自身有P1和P2两个属性，着两个属性对于CFp, cf1, cf2, cf3, cf4, or cf5都是不可见的。CFp中的名为CFP1的属性同时被cf1, cf2, cf3, cf4, and cf5（除了CF）共享。注意CF和CFp之间没有隐式的原型链。

不像基于类的语言，属性可以动态的插入到对象上。也就是说构造函数不必在新生成的对象上创建许多属性，比如每次构造函数执行都会走相同的代码。 在上图中我们可以cf1, cf2, cf3, cf4, cf5添加共享属性，只需要把该属性赋值到CFp对象上即可。

### 4.2.2 严格模式