**定义**

每次当控制器转到ECMAScript可执行代码的时候，即会进入到一个执行上下文。执行上下文(简称-EC)是ECMA-262标准里的一个抽象概念，用于同可执行代码(executable code)概念进行区分。

标准规范没有从技术实现的角度定义EC的准确类型和结构，这应该是具体实现ECMAScript引擎时要考虑的问题。

活动的执行上下文组在逻辑上组成一个堆栈。堆栈底部永远都是全局上下文(global context)，而顶部就是当前(活动的)执行上下文。堆栈在EC类型进入和退出上下文的时候被修改（推入或弹出）。

**可执行代码类型**

可执行代码的类型这个概念与执行上下文的抽象概念是有关系的。在某些时刻，可执行代码与执行上下文完全有可能是等价的。

例如，我们可以定义执行上下文堆栈是一个数组：

ECStack = [];

每次进入function (即使function被递归调用或作为构造函数) 的时候或者内置的eval函数工作的时候，这个堆栈都会被压入。

**全局代码**

这种类型的代码是在"程序"级处理的：例如加载外部的js文件或者本地<script></script>标签内的代码。全局代码不包括任何function体内的代码。

在初始化（程序启动）阶段，ECStack是这样的：

ECStack = [  
 globalContext  
];

**函数代码**

当进入funtion函数代码(所有类型的funtions)的时候，ECStack被压入新元素。需要注意的是，具体的函数代码不包括内部函数(inner functions)代码。如下所示，我们使函数自己调自己的方式递归一次：

(function foo(bar) {  
 if (bar) {  
 return;  
 }  
 foo(true);  
})();

那么，ECStack以如下方式被改变：

// 第一次foo的激活调用  
ECStack = [  
 <foo> functionContext  
 globalContext  
];  
   
// foo的递归激活调用  
ECStack = [  
 <foo> functionContext – recursively  
 <foo> functionContext  
 globalContext  
];

每次return的时候，都会退出当前执行上下文的，相应地ECStack就会弹出，栈指针会自动移动位置，这是一个典型的堆栈实现方式。一个抛出的异常如果没被截获的话也有可能从一个或多个执行上下文退出。相关代码执行完以后，ECStack只会包含全局上下文(global context)，一直到整个应用程序结束。

**Eval 代码**

eval 代码有点儿意思。它有一个概念： 调用上下文(calling context),例如，eval函数调用的时候产生的上下文。eval(变量或函数声明)活动会影响调用上下文(calling context)。

eval('var x = 10');  
   
(function foo() {  
 eval('var y = 20');  
})();  
   
alert(x); // 10  
alert(y); // "y" 提示没有声明

ECStack的变化过程：

ECStack = [  
 globalContext  
];  
   
// eval('var x = 10');  
ECStack.push(  
 evalContext,  
 callingContext: globalContext  
);  
   
// eval exited context  
ECStack.pop();  
   
// foo funciton call  
ECStack.push(<foo> functionContext);  
   
// eval('var y = 20');  
ECStack.push(  
 evalContext,  
 callingContext: <foo> functionContext  
);  
   
// return from eval  
ECStack.pop();  
   
// return from foo  
ECStack.pop();

也就是一个非常普通的逻辑调用堆栈。

在版本号1.7以上的SpiderMonkey(内置于Firefox,Thunderbird)的实现中，可以把调用上下文作为第二个参数传递给eval。那么，如果这个上下文存在，就有可能影响“私有”(有人喜欢这样叫它)变量。

function foo() {  
 var x = 1;  
 return function () { alert(x); };  
};  
   
var bar = foo();  
   
bar(); // 1  
   
eval('x = 2', bar); // 传入上下文，影响了内部的var x 变量  
   
bar(); // 2