JS引擎：又名js解释器，JavaScript是解释型语言，这就是说它无需编译，直接由Js引擎直接执行。

解析执行过程：编译和执行，执行过程又可分为语法分析、预解析、解释执行。

页面产生便创建了GO全局对象（Global Object）（也就是window对象）；

编译：将字符串代码编译为可执行码。

语法分析：对代码进行通篇检查，以排除一些低级错误，如果出现不合法会向外抛出一个语法错误（syntaxError），停止代码块的执行，然后继续查找并加载下一个代码块；如果语法正确，则进入到预编译阶段。

预解析(预编译)：变量收集和分号补全；

执行：执行代码。

预解析-变量收集：

1. 创建VO对象
2. 创建作用域链（Scope Chain）
3. 将函数形参（值赋予undefined）、变量声明（值赋予undefined）、函数声明（值赋予函数体）放入VO
4. 确定this的值

预解析造成了函数声明提前和变量声明提升，且函数声明优先级高于变量声明。如下：

*function* func(*age*) {

   console.log(*age*);

*var* *age* = 25;

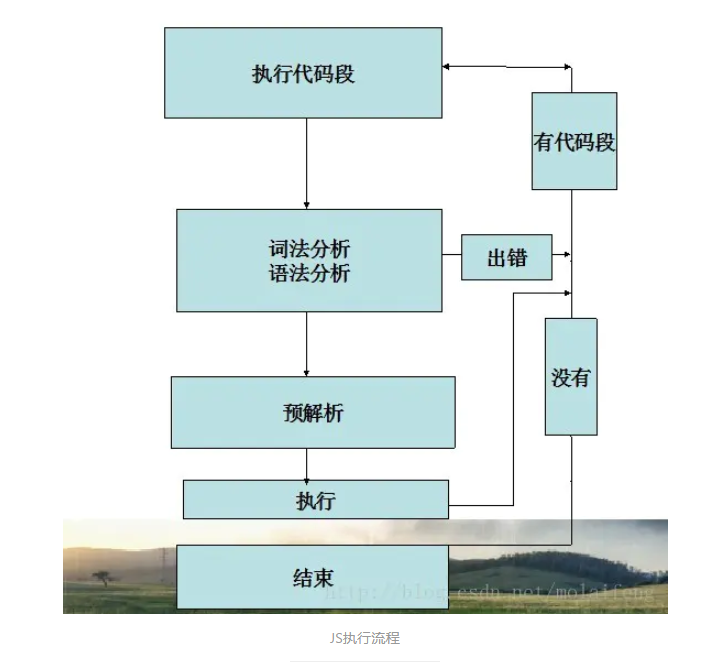
   console.log(*age*);

*function* age() {}

   console.log(*age*);

}

func(18);// fnc-25-25



执行上下文栈（Execution Context Stack）

函数多了，就有多个函数执行上下文，每次调用函数创建一个新的执行上下文，那如何管理创建的那么多执行上下文呢？

JavaScript 引擎创建了执行上下文栈来管理执行上下文。可以把执行上下文栈认为是一个存储函数调用的栈结构，遵循先进后出的原则。

JavaScript 执行在单线程上，所有的代码都是排队执行。

一开始浏览器执行全局的代码时，首先创建全局的执行上下文，压入执行栈的顶部。

每当进入一个函数的执行就会创建函数的执行上下文，并且把它压入执行栈的顶部。当前函数执行完成后，当前函数的执行上下文出栈，并等待垃圾回收。

浏览器的 JS 执行引擎总是访问栈顶的执行上下文。

全局上下文只有唯一的一个，它在浏览器关闭时出栈。

