对于JS来说，大部分情况下编译发生在代码执行前的几微秒（甚至更短）的时间内。

引擎：从头到尾负责整个JS程序的编译及执行过程。编译器：负责语法分析及代码生成等‘脏活累活’。

作用域：负责收集并维护由所有声明的标识符（变量）组成的一系列查询，并实施一套非常严格的规则，确定当前执行的代码对这些标识符的访问权限。

变量的赋值会执行两个操作：1.编译器在当前作用域中声明一个变量2.运行时引擎在作用域中查找该变量，如果可以找对就进行赋值。

变量出现在赋值操作左侧时进行LHS查询,出现在右侧时进行RHS查询。

严格模式禁止自动或隐式创建全局变量。

词法化的过程会对源代码中的字符进行检查，如果是有状态的解析过程，还会赋予单词语义。

词法作用域是由你在写代码时将变量和块作用域写在哪里来决定的，因此词法分析器处理代码会保持作用域不变。

在多层嵌套作用域中可以定义同名标识符，这叫做‘遮蔽效应’。

eval()函数参数可以为字符串，并将其中的内容视为好像在书写时就存在于程序中这个位置的代码。

with通常被当作重复引用同一个对象中的多个属性的快捷方式，可以不需要重复引用对象本身。

eval()、with()会在运行时修改或创建新的作用域，来欺骗其他在书写时定义的词法作用域。

对函数的认知：从所写的代码中挑选一个任意的片段，然后用函数声明对它进行包装，实际上就是把这些代码‘隐藏’了。

函数表达式可以匿名，函数声明不可以匿名。

匿名函数的几个缺点：1.匿名函数在追踪栈中不会显示出有意义的函数名，对调试造成影响。

2.函数引用自身时只能使用已经过期的arguments.callee引用。

3.匿名函数省略了对于代码可读性/可理解性很重要的函数名，不便于对函数理解。

IIFE 可以解决undefined标识符的默认值被错误覆盖导致的异常。

将一个参数命名为undefined，但是在对应的位置不传入任何值，这样就可以保证在代码块中undefined标识符的值为undefined。

IIFE还可以将代码顺序调整。实现：将需要运行的函数放在第二位，在IIFE执行之后被当作参数传递进去。

（function IIFE(def){ def(window);}）

(function def(global)

{ var a=3; console.log(a);//3console.log(global.a);//2

});

块作用域将代码从在函数中隐藏信息扩展为在块中隐藏信息。

JS的ES3规定try/catch的catch分句会创建一个块作用域，其中声明的变量仅在catch内部有效。

let关键字可以将变量绑定到所在的任意作用域中。const可以创建块作用域变量，但是值是不可变的。

函数声明会被提升，但是函数表达式却不会被提升。

函数声明在变量声明前。

闭包是基于词法作用域书写代码时所产生的自然结果。

延迟函数的回调会在循环结束时才执行。

for (let i=1; i<=5; i++) {

setTimeout( function timer() {

console.log( i );

}, i\*1000 );

}

模块模式需要具备两个必要条件。

1. 必须有外部的封闭函数，该函数必须至少被调用一次（每次调用都会创建一个新的模块实例）。

2. 封闭函数必须返回至少一个内部函数，这样内部函数才能在私有作用域中形成闭包，并且可以访问或者修改私有的状态。

ES6 中为模块增加了一级语法支持。但通过模块系统进行加载时，ES6 会将文件当作独立的模块来处理。每个模块都可以导入其他模块或特定的 API 成员，

同样也可以导出自己的API 成员。

基于函数的模块并不是一个能被稳定识别的模式（编译器无法识别），它们的 API 语义只有在运行时才会被考虑进来。因此可以在运行时修改一个模块的 API。

import 可以将一个模块中的一个或多个 API 导入到当前作用域中，并分别绑定在一个变量上。

module 会将整个模块的 API 导入并绑定到一个变量上。 export 会将当前模块的一个标识符（变量、函数）导出为公共 API。

这些操作可以在模块定义中根据需要使用任意多次。

模块文件中的内容会被当作好像包含在作用域闭包中一样来处理，就和前面介绍的函数闭包模块一样。