## **前面的话**

　　javascript拥有一套设计良好的规则来存储变量，并且之后可以方便地找到这些变量，这套规则被称为作用域。作用域貌似简单，实则复杂，由于作用域与[this机制](http://www.cnblogs.com/xiaohuochai/p/5735901.html" \t "http://www.cnblogs.com/xiaohuochai/p/_blank)非常容易混淆，使得理解作用域的原理更为重要。本文是深入理解javascript作用域系列的第一篇——内部原理

　　内部原理分成编译、执行、查询、嵌套和异常五个部分进行介绍，最后以一个实例过程对原理进行完整说明

### **编译**

　　以var a = 2;为例，说明javascript的内部编译过程，主要包括以下三步：

【1】分词(tokenizing)

　　把由字符组成的字符串分解成有意义的代码块，这些代码块被称为词法单元(token)

　　var a = 2;被分解成为下面这些词法单元：var、a、=、2、;。这些词法单元组成了一个词法单元流数组

[IMG_256](http://www.cnblogs.com/xiaohuochai/p/javascript:void(0);)

// 词法分析后的结果[

"var" : "keyword",

"a" : "identifier",

"=" : "assignment",

"2" : "integer",

";" : "eos" (end of statement)

]

[IMG_257](http://www.cnblogs.com/xiaohuochai/p/javascript:void(0);)

【2】解析(parsing)

　　把词法单元流数组转换成一个由元素逐级嵌套所组成的代表程序语法结构的树，这个树被称为“抽象语法树” (Abstract Syntax Tree, AST)

　　var a = 2;的抽象语法树中有一个叫VariableDeclaration的顶级节点，接下来是一个叫Identifier(它的值是a)的子节点，以及一个叫AssignmentExpression的子节点，且该节点有一个叫Numericliteral(它的值是2)的子节点

[IMG_258](http://www.cnblogs.com/xiaohuochai/p/javascript:void(0);)

{

operation: "=",

left: {

keyword: "var",

right: "a"

}

right: "2"

}

[IMG_259](http://www.cnblogs.com/xiaohuochai/p/javascript:void(0);)

【3】代码生成

　　将AST转换为可执行代码的过程被称为代码生成

　　var a=2;的抽象语法树转为一组机器指令，用来创建一个叫作a的变量(包括分配内存等)，并将值2储存在a中

　　实际上，javascript引擎的编译过程要复杂得多，包括大量优化操作，上面的三个步骤是编译过程的基本概述

　　任何代码片段在执行前都要进行编译，大部分情况下编译发生在代码执行前的几微秒。javascript编译器首先会对var a=2;这段程序进行编译，然后做好执行它的准备，并且通常马上就会执行它

### **执行**

　　简而言之，编译过程就是编译器把程序分解成词法单元(token)，然后把词法单元解析成语法树(AST)，再把语法树变成机器指令等待执行的过程

　　实际上，代码进行编译，还要执行。下面仍然以var a = 2;为例，深入说明编译和执行过程

【1】编译

　　1、编译器查找作用域是否已经有一个名称为a的变量存在于同一个作用域的集合中。如果是，编译器会忽略该声明，继续进行编译；否则它会要求作用域在当前作用域的集合中声明一个新的变量，并命名为a

　　2、编译器将var a = 2;这个代码片段编译成用于执行的机器指令

　　[注意]依据编译器的编译原理，javascript中的重复声明是合法的

//test在作用域中首次出现，所以声明新变量，并将20赋值给testvar test = 20;//test在作用域中已经存在，直接使用，将20的赋值替换成30var test = 30;

【2】执行

　　1、引擎运行时会首先查询作用域，在当前的作用域集合中是否存在一个叫作a的变量。如果是，引擎就会使用这个变量；如果否，引擎会继续查找该变量

　　2、如果引擎最终找到了变量a，就会将2赋值给它。否则引擎会抛出一个异常

### **查询**

　　在引擎执行的第一步操作中，对变量a进行了查询，这种查询叫做LHS查询。实际上，引擎查询共分为两种：LHS查询和RHS查询

　　从字面意思去理解，当变量出现在赋值操作的左侧时进行LHS查询，出现在右侧时进行RHS查询

　　更准确地讲，RHS查询与简单地查找某个变量的值没什么区别，而LHS查询则是试图找到变量的容器本身，从而可以对其赋值

function foo(a){

console.log(a);//2}

foo( 2 );

　　这段代码中，总共包括4个查询，分别是：

　　1、foo(...)对foo进行了RHS引用

　　2、函数传参a = 2对a进行了LHS引用

　　3、console.log(...)对console对象进行了RHS引用，并检查其是否有一个log的方法

　　4、console.log(a)对a进行了RHS引用，并把得到的值传给了console.log(...)

### **嵌套**

　　在当前作用域中无法找到某个变量时，引擎就会在外层嵌套的作用域中继续查找，直到找到该变量，或抵达最外层的作用域（也就是全局作用域）为止

function foo(a){

console.log( a + b ) ;

}var b = 2;

foo(2);// 4

　　在代码片段中，作用域foo()函数嵌套在全局作用域中。引擎首先在foo()函数的作用域中查找变量b，并尝试对其进行RHS引用，没有找到；接着，引擎在全局作用域中查找b，成功找到后，对其进行RHS引用，将2赋值给b

### **异常**

　　为什么区分LHS和RHS是一件重要的事情？因为在变量还没有声明（在任何作用域中都无法找到变量）的情况下，这两种查询的行为不一样

**RHS**

【1】如果RHS查询失败，引擎会抛出ReferenceError(引用错误)异常

//对b进行RHS查询时，无法找到该变量。也就是说，这是一个“未声明”的变量function foo(a){

a = b;

}

foo();//ReferenceError: b is not defined

【2】如果RHS查询找到了一个变量，但尝试对变量的值进行不合理操作，比如对一个非函数类型值进行函数调用，或者引用null或undefined中的属性，引擎会抛出另外一种类型异常：TypeError(类型错误)异常

function foo(){

var b = 0;

b();

}

foo();//TypeError: b is not a function

**LHS**

【1】当引擎执行LHS查询时，如果无法找到变量，全局作用域会创建一个具有该名称的变量，并将其返还给引擎

function foo(){

a = 1;

}

foo();

console.log(a);//1

【2】如果在严格模式中LHS查询失败时，并不会创建并返回一个全局变量，引擎会抛出同RHS查询失败时类似的ReferenceError异常

[IMG_260](http://www.cnblogs.com/xiaohuochai/p/javascript:void(0);)

function foo(){

'use strict';

a = 1;

}

foo();

console.log(a);//ReferenceError: a is not defined

[IMG_261](http://www.cnblogs.com/xiaohuochai/p/javascript:void(0);)

### **原理**

function foo(a){

console.log(a);

}

foo(2);

　　以上面这个代码片段来说明作用域的内部原理，分为以下几步：

【1】引擎需要为foo(...)函数进行RHS引用，在全局作用域中查找foo。成功找到并执行

【2】引擎需要进行foo函数的传参a=2，为a进行LHS引用，在foo函数作用域中查找a。成功找到，并把2赋值给a

【3】引擎需要执行console.log(...)，为console对象进行RHS引用，在foo函数作用域中查找console对象。由于console是个内置对象，被成功找到

【4】引擎在console对象中查找log(...)方法，成功找到

【5】引擎需要执行console.log(a)，对a进行RHS引用，在foo函数作用域中查找a，成功找到并执行

【6】于是，引擎把a的值，也就是2传到console.log(...)中

【7】最终，控制台输出2