JavaScript 基础之

JavaScript 作用域

猫眼电影前端组　王子龙

关键词： 编译原理　词法作用域　动态作用域　this　ES6

目录

- 简史
- 编译原理
- 作用域
- 异常
- this & 箭头函数

1. JavaScript 的诞生

1995 – Brendan Eich 读完了历史上所有在程序语言设计中曾经出现过的错误，自己又发明了一些更多的错误，然后用它们创造出了LiveScript。之后，为了紧跟Java语言的时髦潮流，它被重新命名为JavaScript。再然后，为了追随一种皮肤病的时髦潮流，这语言又被命名为ECMAScript。

—— 编程语言简史（其实是野史）

(注：Eczema，意为“湿疹”)

2. JavaScript 与编译原理

- 简化的编译过程
- 左值与右值
- 关于 Babel

2.1 简化的编译过程

源码

```
var foo = bar + 2;
```

↓

分词/词法分析
Tokenizing/Lexing

↓

词法单元

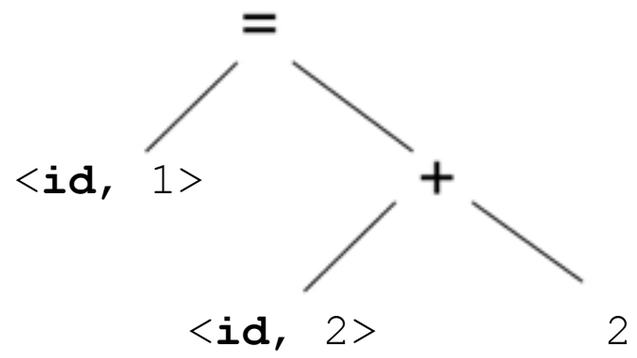
```
<var> <id, 1> <=> <id, 2> <+> <2> <;>
```

↓

解析/语法分析
Parsing

↓

语法树



↓

代码生成

↓

目标代码（例如可执行代码）

```
01010101010101010101...
```

1	foo	…
2	bar	…

符号表

注：这里只是一个简化的编译器的步骤示意，省略了
语义分析、中间代码生成、代码优化等步骤。

2.1.1 关于 Babel


```
let obj = {
  name: 'test',
  _map(arr) {
    return arr.map(n => {
      console.log(this.name);
      return n + 1;
    });
  }
}
```

→

```
'use strict';

var obj = {
  name: 'test',
  _map: function _map(arr) {
    var _this = this;

    return arr.map(function (n) {
      console.log(_this.name);
      return n + 1;
    });
  }
};
```

问题 1

下面的说法是否准确?

> 尽管通常将JavaScript 归类为“动态”或“解释执行”语言，但事实上它是一门编译语言……但与传统的编译语言不同，它不是提前编译的，编译结果也不能在分布式系统中进行移植。

问题 1 答案-Step 1

JavaScript at 20, by Brendan Eich：

So in **10 days** in May 1995, I wrote

A lexical scanner and parser for early JS

The parser emitted stack-machine bytecode

Which ran in a bytecode interpreter

`Function.prototype.toString` bytecode decompiler

The standard library was poor

`Array` was `Object` with `.length` property

`Date` hand-ported (h/t ksmith@netscape.com) from `java.util.Date`

V8 引擎，在运行前先将 JavaScript 编译为机器码，而非字节码或是解释执行它

问题 2 答案-Step 2

- 语言并不会定义其实现

类似的问题：

> JavaScript、PHP、Java，哪个运行最快？ → 由编译器、引擎等决定

> JavaScript 和 Java 哪个好？ → 有时由老板的工资来决定……

3. 作用域

> 在电脑程序设计中，作用域（scope，或译作有效范围）是名字（name）与实体（entity）的绑定（binding）保持有效的那部分计算机程序。
>
> —— 维基百科

> x 的一个声明的作用域（scope）是指程序的一个区域，在其中对 x 的使用都指向这个声明。
>
> —— 《编译原理（第2版）》

3.1 词法作用域 vs 动态作用域

- **词法作用域（Lexical Scoping）**的重要特征：作用域的确定发生在代码的书写阶段（关注函数的定义地点）

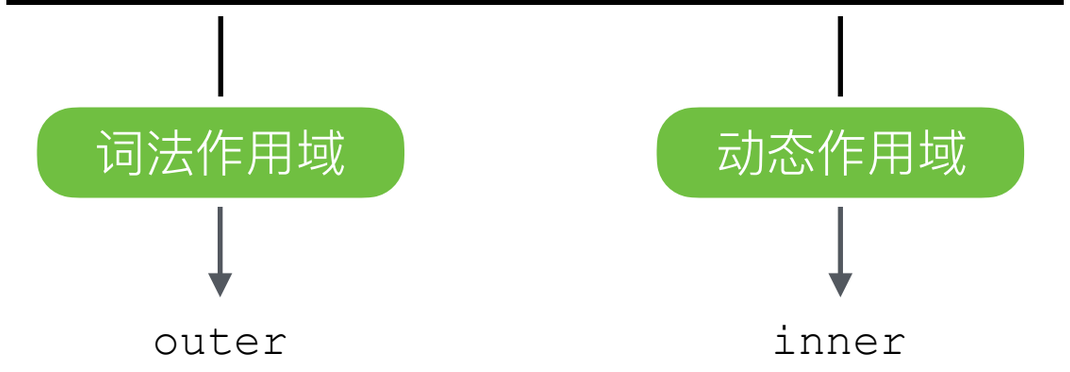
- **动态作用域（Dynamic Scoping）**的重要特征：作用域链是基于运行时的调用栈来确定的（关注函数的调用地点）

- 词法作用域（lexical scope） === 静态作用域（static scope）

- 大部分编程语言是基于词法作用域的：C/C++，Java，JavaScript，Python

- 少部分语言是基于动态作用域的：Pascal、Emacs Lisp

```
function foo() {
  console.log(a);
}

function bar() {
  var a = 'inner';
  foo();
}

var a = 'outer';

bar();
```



标识符

- 标识符（token）：变量、属性、函数、函数参数的名字。

- 这段代码运行正常：

```
var foo = {
    enum: 'haha'
};

console.log(foo.enum);
```

- 但使用保留字命名变量会报错：

```
> var enum = 'haha';
Uncaught SyntaxError: Unexpected reserved word
```

词法作用域只会查找一级标识符

对象属性名的查找，由对象属性访问规则来负责

LHS查找与RHS查找

```
var b = 0;
var a = b + 1;
```

术语左值和右值分别表示可以出现在赋值表达式左部和右部的值。也就是说，右值是我们通常所说的“值”，而左值是存储位置。
——《编译原理（第2版）》

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。

常见的赋值操作有：

① =

② 对函数参数的赋值

问题 2

请指出下面一段代码中的所有 LHS 查询和 RHS 查询

```
function foo(a) {
  var b = a;
  return a + b;
}

var c = foo(2);
```

问题 2：答案

请指出下面一段代码中的所有 LHS 查询和 RHS 查询

```
function foo(a) {
  var b = a;
  return a + b;
}

var c = foo(2);
```

LHS 查询共3处：
① c = …
② a = 2
③ b = …

RHS 查询共4处：
① 查找 foo 标识符的值（是个函数）
② 查找形参 a 的值
③ 在 return 语句中查找 a 的值
④ 在 return 语句中查找 b 的值

问题 2：进一步

请指出下面一段代码中的所有 LHS 查询和 RHS 查询

```
var c = 0;
var a = b = c;
```

```
var a = b = c;
```

这个语句有一次右值查找，两次左值查找。

因为赋值语句是从右向左执行，赋值表达式本身是有返回值的，返回值等于表达式的右值。

上面这句就相当于这样：

```
var a = (global.b = c);
```

3.2 JavaScript 的作用域层级

程序语言的不同静态作用域层级：

① 函数作用域 ⟶ ES3

② 块作用域 ⟶ ES3，ES6

③ 类作用域 ⟶ ES6

JavaScript 中的块作用域

ES3:

```
try{
  throw 2;
} catch(a) {
  // 正常输出
  console.log(a);
}

// 报错：ReferenceError
console.log(a);
```

ES6:

```
{
  let a = 2;

  // 正常输出
  console.log(a);
}

// 报错：ReferenceError
console.log(a);
```

问题 3

开放性问题

JavaScript 为何要支持 this ？

this 关键字几乎是面向对象语言的标配

> A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self")
>
> — wikipedia

4. 异常

- 解析期异常
- 运行时异常 ⟶ 注意，有 bug 出没!

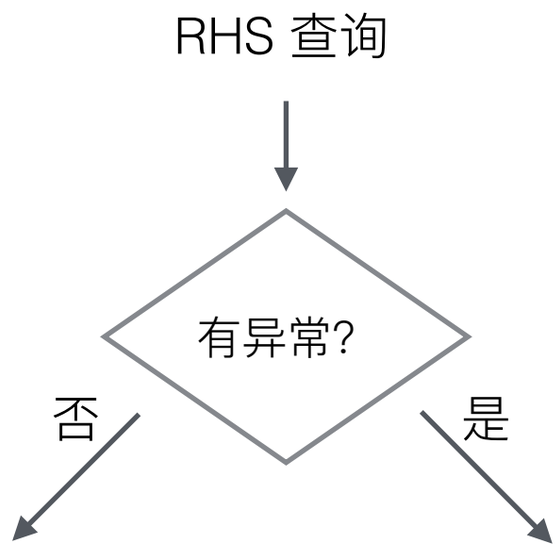
解析异常（静态）


```
<script>
console.log(1);
  function(){
      return ee}
  }
console.log(2);
</script>
```

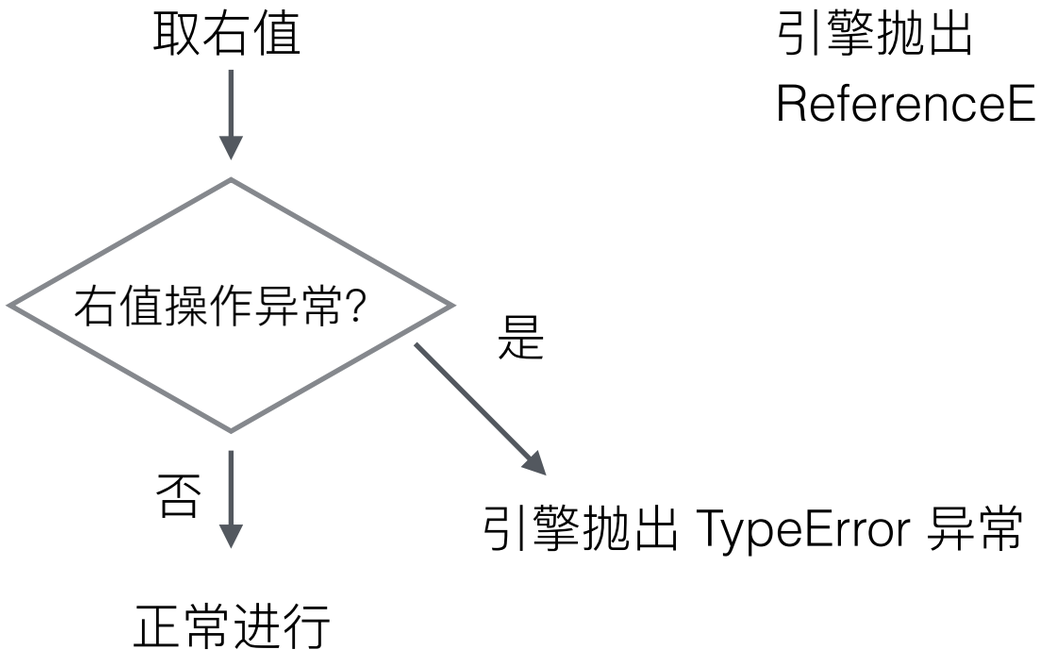
两个 console.log 都不会执行，立即抛出异常：
Uncaught **SyntaxError**: Unexpected token (

左值与右值：右值的异常

```
function foo(a) {
  console.log(a + b);
  b = a;
};
foo(2);
// Uncaught ReferenceError: b is not defined
```



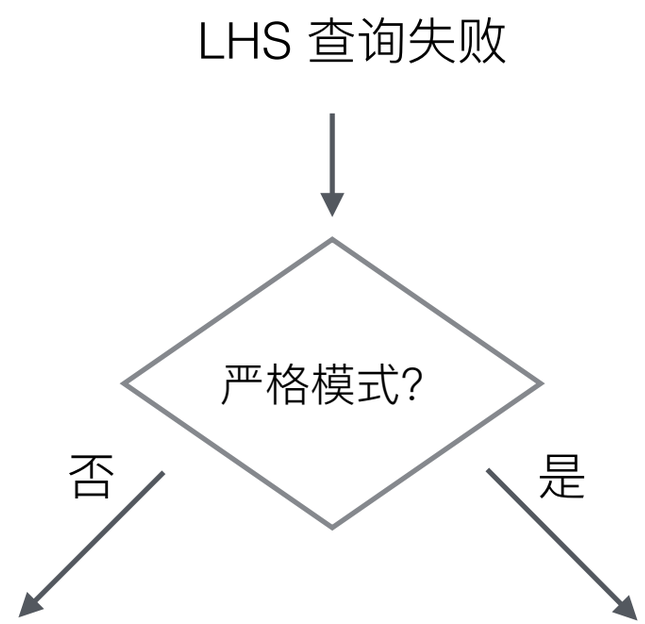

```
var b;
var a = b;
console.log(a.haha);

// Uncaught TypeError: Cannot read
property 'haha' of undefined
```



左值与右值：左值的异常

非严格模式：

```
(function() {
  var a = 'haha';
  b = a;
})();
```



全局作用域：“变量 b 之前并不存在，但是我很热心地帮你创建了一个。”

严格模式的代码：

```
(function() {
  'use strict';
  var a = 'haha';
  b = a;
})();
```

引擎抛出 ReferenceError 异常

5. this 的动态作用域特性

this 永远指向当前的上下文对象

this 是个关键字，被自动定义在所有函数的作用域中

函数的四种调用方式：

- 函数调用模式
- 构造器调用模式
- 方法调用模式
- apply 调用模式

this的指向：关注调用位置

```
function foo() {
  console.log(this.a);
}

var obj2 = {
  a: 2,
  foo: foo
}

var obj1 = {
  a: 1,
  obj2: obj2
}

obj1.obj2.foo();  // 2

var a = 'global';
var b = obj1.obj2.foo;
b();
```

函数是对象　　值传递

JS 的对象可以有方法，但这只是简化的说法，稍微详细一些的说法是:

对象的属性可以是引用属性，其指向了某个函数对象，该函数对象调用时隐式地将自己的 this 绑定为该对象，于是可以用该函数访问、操作对象的数据，于是看起来就是个方法。

箭头函数 & this 词法化

- 以前，我们手动词法化 this：

```
var self = this;
var that = this;
var _this = this;
```

- 现在，如果源码为 ES6 或 ES6+，那么更好的方案是箭头函数：

```
// The classic mistake...
Car.prototype.start = function () {
  setTimeout(function () {
    this.startDriving();    // Wrong this!
  }, 1000);
};

// Much better than using ES5 .bind!
Car.prototype.start = function () {
  setTimeout(() => this.startDriving(), 1000);
};
```

http://brendaneich.github.io/ModernWeb.tw-2015/#57

Runtime Semantics: Evaluation

…

2. Let scope be the LexicalEnvironment of the running execution context.

…

Note: An ArrowFunction does not define local bindings for **`arguments, super, this`**, or **`new.target`**. Any reference to arguments, super, this, or new.target within an ArrowFunction must resolve to a binding in a lexically enclosing environment.

http://www.ecma-international.org/ecma-262/6.0/#sec-arrow-function-definitions

Thanks

今后的分享主题

- 如何写出好代码
    - 以《代码大全》为基础，结合自己的经验、思考，提炼出适合前端工程师的编码指导（但不是 airbnb 那样的规范）
- JavaScript 函数式编程基础
- JavaScript 正则表达式 10 例
- 致我们久违的 CSS
    - 你是否觉得，很久没有写 CSS 了?

参考资料

- 《你不知道的JavaScript（上卷）》，[美] Kyle Simpson 著，赵望野 梁杰 译.
- 《ES6标准入门（第2版）》，阮一峰，2016
- 《编译原理（第3版）》, Alfred V. Aho, etc.
- 《操作系统》
- 《JavaScript 语言精粹与编程实践（第2版）》，周爱民，2009
- 《JavaScript 语言精粹（修订版）》, Douglas Crockford
- 《静态作用域和动态作用域》, http://www.cnblogs.com/lienhua34/archive/2012/03/10/2388872.html
- this | MDN, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this
- Object-oriented programming | wikipedia, https://en.wikipedia.org/wiki/Object-oriented_programming
- 浅谈JavaScript中的错误, http://www.html-js.com/article/On-the-error-in-JavaScript
- JavaScript at 20, by Brendan Eich, http://brendaneich.github.io/ModernWeb.tw-2015/#1
- Java 是编译型语言还是解释型语言？ | 知乎, https://www.zhihu.com/question/19608553
- 虚拟机随谈（一）：解释器，树遍历解释器，基于栈与基于寄存器，大杂烩, http://rednaxelafx.iteye.com/blog/492667
- V8 (JavaScript引擎) | 维基百科), https://zh.wikipedia.org/wiki/V8_(JavaScript%E5%BC%95%E6%93%8E