Chapter 1

语言基础

任何语言的核心所描述的都是这门语言在最基本的层面上如何工作，涉及语法、操作符、数据类型以及内置功能，在此基础之上才可以构建复杂的解决方案。

1.1 语法

ECMAScript 的语法很大程度上借鉴了 C 语言和其他类 C 语言，如 Java 和 Perl。

1.1.1 区分大小写

ECMAScript 中一切都区分大小写。无论是变量、函数名还是操作符，都区分大小写。

1.1.2 标识符

所谓标识符，就是变量、函数、属性或函数参数的名称。标识符可以由一或多个下列字符组成：

- 第一个字符必须是一个字母、下划线（_）或美元符号（$）；
- 剩下的其他字符可以是字母、下划线、美元符号或数字。

标识符中的字母可以是扩展 ASCII（Extended ASCII）中的字母，也可以是 Unicode 的字母字符。

按照惯例，ECMAScript 标识符使用驼峰大小写形式，即第一个单词的首字母小写，后面每个单词的首字母大写。

关键字、保留字、true、false 和 null 不能作为标识符。具体内容参考**??**。

1.1.3 注释

ECMAScript 采用 C 语言风格的注释，包括单行注释和块注释。单行注释以两个斜杠字符开头。块注释以一个斜杠和一个星号（/*）开头，以它们的反向组合（*/）结尾。

1.1.4 严格模式

严格模式是一种不同的 JavaScript 解析和执行模型，ECMAScript 3 的一些不规范写法在这种模式下会被处理，对于不安全的活动将抛出错误。要对整个脚本启用严格模式，在脚本开头加上这一行：

```
"use strict";
```

1.1.5 语句

1.2 关键字和保留字

ECMA-262 描述了一组保留的关键字，这些关键字有特殊用途。按照规定，保留的关键字不能用作标识符或属性名。ECMA-262 第 6 版规定的所有关键字如下：break do in typeof case else instanceof var catch export new void class extends return while const finally super with continue for switch yield debugger function this default if throw delete import try

规范中也描述了一组未来的保留字，同样不能用作标识符或属性名。虽然保留字在语言中没有特定用途，但它们是保留给将来做关键字用的。

以下是 ECMA-262 第 6 版为将来保留的所有词汇。

- 始终保留：enum
- 严格模式下保留：implements package public interface protected static let private
- 模块代码中保留：await

1.3 变量

ECMAScript 变量是松散类型的，意思是变量可以用于保存任何类型的数据。每个变量只不过是一个用于保存任意值的命名占位符。有 3 个关键字可以声明变量：var、const 和 let。其中，var 在 ECMAScript 的所有版本中都可以使用，而 const 和 let 只能在 ECMAScript 6 及更晚的版本中使用。

1.3.1 var关键字

要定义变量，可以使用 var 操作符（注意 var 是一个关键字），后跟变量名（即标识符）。

```
var msg;
```

不初始化的情况下，变量会保存一个特殊值 undefined，下一节讨论数据类型时会谈到。）ECMAScript 实现变量初始化，因此可以同时定义变量并设置它的值：

```
var msg = 'hi';
```

message 被定义为一个保存字符串值 hi 的变量。像这样初始化变量不会将它标识为字符串类型，只是一个简单的赋值而已。随后，不仅可以改变保存的值，也可以改变值的类型：

```
var message = "hi";
message = 100; // legal, but not recommended
```

var声明作用域

关键的问题在于，使用 var 操作符定义的变量会成为包含它的函数的局部变量。

```
function test(){
        var msg = 'hi'; // local variable
    }
test();
console.log(msg); // error
```

不过，在函数内定义变量时省略 var 操作符，可以创建一个全局变量：

```
function test(){
        msg = 'hi'; // global variable
    }
test();
console.log(msg); // 'hi'
```

虽然可以通过省略 var 操作符定义全局变量，但不推荐这么做。在局部作用域中定义的全局变量很难维护，也会造成困惑。这是因为不能一下子断定省略 var 是不是有意而为之。在严格模式下，如果像这样给未声明的变量赋值，则会导致抛出 ReferenceError。

如果需要定义多个变量，可以在一条语句中用逗号分隔每个变量（及可选的初始化）：

```
var msg='hi',
found=false,
age=29;
```

因为 ECMAScript 是松散类型的，所以使用不同数据类型初始化的变量可以用一条语句来声明。插入换行和空格缩进并不是必需的，但这样有利于阅读理解。

在严格模式下，不能定义名为`eval`和`arguments`的变量，否则会导致语法错误。

var声明提升

`var`关键字声明的变量会自动提升到函数作用域顶部：

```
function foo() {
        console.log(age);
        var age = 26;
    }
foo(); // undefined
```

之所以不会报错，是因为 ECMAScript 运行时把它看成等价于如下代码：

```
function foo() {
        var age;
        console.log(age);
        age = 26;
    }
foo(); // undefined
```

这就是所谓的“提升”（hoist），也就是把所有变量声明都拉到函数作用域的顶部。此外，反复多次使用 var 声明同一个变量也没有问题。

1.3.2 let声明

let 跟 var 的作用差不多，但有着非常重要的区别。最明显的区别是，let 声明的范围是块作用域，而 var 声明的范围是函数作用域。

```
if (true) {
        var name = 'Matt';
        console.log(name);
    }
console.log(name);

if (true) {
        let age = 26;
        console.log(age);
    }
console.log(age); // ReferenceError: age is not defined
```

age 变量之所以不能在 if 块外部被引用，是因为它的作用域仅限于该块内部。块作用域是函数作用域的子集，因此适用于 var 的作用域限制同样也适用于 let。

let 也不允许同一个块作用域中出现冗余声明。

JavaScript 引擎会记录用于变量声明的标识符及其所在的块作用域，因此嵌套使用相同的标识符不会报错，而这是因为同一个块中没有重复声明。

对声明冗余报错不会因混用 let 和 var 而受影响。这两个关键字声明的并不是不同类型的变量，它们只是指出变量在相关作用域如何存在。

```
var name;
let name; // SyntaxError: Identifier 'name' has already been declared

let age;
var age; // SyntaxError: Identifier 'name' has already been declared
```

暂时性死区

全局声明

条件声明

for循环中的let声明

1.3.3 const声明

1.3.4 数据类型

typeof操作符

Undefined类型

Null类型

Boolean类型

Number类型

1.3.5 String类型

1.3.6 Symbol类型

1.3.7 Object类型

1.4 操作符

1.5 语句

1.5.1 if语句

1.5.2 do-while语句

1.5.3 while语句

1.5.4 for语句

1.5.5 for-in语句

1.5.6 for-of语句

1.5.7 标签语句

1.5.8 break和continue语句

break 和 continue 语句为执行循环代码提供了更严格的控制手段。其中，break 语句用于立即退出循环，强制执行循环后的下一条语句。而 continue 语句也用于立即退出循环，但会再次从循环顶部开始执

行。

1.5.9 with语句

1.5.10 switch语句

Chapter 2

集合引用类型

2.1 Object

2.2 Array

ECMAScript 数组也是一组有序的数据，数组中每个槽位可以存储任意类型的数据。这意味着可以创建一个数组，它的第一个元素是字符串，第二个元素是数值，第三个是对象。ECMAScript 数组也是动态大小的，会随着数据添加而自动增长。

2.2.1 创建数组

一种是使用 Array 构造函数。如果知道数组中元素的数量，那么可以给构造函数传入一个数值，然后 length 属性就会被自动创建并设置为这个值。也可以给 Array 构造函数传入要保存的元素。

```
let colors1 = new Array();
let colors2 = new Array(20);
let colors3 = new Array("red", "blue", "green");
```

创建数组时可以给构造函数传一个值。这时候就有点问题了，因为如果这个值是数值，则会创建一个长度为指定数值的数组；而如果这个值是其他类型的，则会创建一个只包含该特定值的数组。

```
let colors = new Array(3);
// create an array with three items
let names = new Array("Stephen CUI");
// create an array with one item, the string "Stephen CUI"
```

在使用 Array 构造函数时，也可以省略 new 操作符。

另一种创建数组的方式是使用数组字面量（array literal）表示法。数组字面量是在中括号中包含以逗号分隔的元素列表。