深入了解 JavaScript

第一部分

第 1 章：基础 JavaScript

1.1 背景

1.1.1

JavaScript 指的是一种编程语言

1.2 语法

1.2.2 语句和表达式

两大语法类别:语句和表达式

1.2.3 分号

分号用于结束语句,而不是结束块.有一种情况你会看到分号出现在块之后:函数表达式作为一个表达式时。如果这样的表达式出现在语句的最后，它后面就会跟上一个分号。

例：var f = function() { };

1.2.4 注释

单行注释:由两个斜杠//开始,行终止时结束

多行注释:限定在/*和*/之间

1.3 变量和赋值

变量在声明后使用

变量和赋值可以同时进行

标识符与变量名

标识符的第一个字符可以是任意 Unicode 字符、美元符（$），或者下划线（_）。

保留字不能作为变量名使用（包括函数名和参数名）

1.4 原始值和对象

原始值：布尔值、数值、字符串、null 和 undefined

其它的值都是对象

原始值得特点：（1）按值进行比较

（2）不可改变：其属性值不能改变、添加或移除

对象：简单对象、数组等

对象的特点：（1）按引用进行比较

（2）默认可变：对象可以很自由地被改变、添加、和移除

对值得分类：typeof 和 instanceof

Typeof 主要用于原始值

Instanceof 用于对象

操作数		结果
Undefined	————	‘undefined’
Null	————	object
布尔值	————	Boolean
数值	————	number
字符串	————	string
函数	————	function
所有其他数值	————	object

1.5 布尔值

原始布尔类型包含 true 和 false 两个值。以下运算会产生布尔值

（1） 二元逻辑运算符：&&（与）、‖（或）

(2) 前置逻辑运算符：！（非）

(3) 比较运算符

——相等运算符：===、！==、==、！=

——排序运算符（针对字符串及数字）：>,>=,<,<=

1.8 字符串（不可改变）

字符串可以直接通过字符串字面量来创建。这些字面量限定在单引号或双引号之内。反斜杠（/）用于转义字符及产生一些控制字符。

获得字符串的长度

— 通过字符串变量的 length 属性获得 — str.length

获得指定位置的字符

— 通过字符串变量的 charAt(n) 方法获得

— str.charAt(n)

1.8.1 字符串运算符

字符串可以通过加好（+）进行连接。

1.8.2 字符串方法

功能	字符串
长度	length
获取指定内容的位置	indexOf(value)
获取指定位置的内容	charAt(i) str[i]
子串获取（提取）	slice(start,end)
转换为数组	split(separator)
添加内容	+ 拼接符号
循环遍历	for
其他	toUpperCase() toLowerCase()

1.9 语句

条件语句（if……else，switch……case）

循环语句（for 循环，while 循环，do—while 循环）：break 可以跳离循环

Continue 会开始一个新的循环

1.10 函数

1.10.1 函数声明的提升特性

函数声明具有提升性——它们的实体会被移动到所在作用域的开始处。

Var 声明也具有提升的特性

1.10.2 特殊的变量 arguments

在 JS 中，函数的所有参数都可以被自由调用，它会通过 arguments 变量来使所有参数可用。Arguments 看起来像个数组，但却不具备数组的方法。

1.10.3 参数太多或太少

额外的参数会被忽略

丢失的参数会得到 undefined 这个值

1.10.4 可选参数

1.10.5 强制参数长度

1.10.6 将 arguments 转换为数组

1.11 异常捕获

最常见的捕获异常的方式如下所示（参考第 14 章）：

```
function getPerson(id) {
    if (id < 0) {
        throw new Error('ID must not be negative: '+id);
    }
    return { id: id }; // normally: retrieved from database
}

function getPersons(ids) {
    var result = [];
    ids.forEach(function (id) {
        try {
            var person = getPerson(id);


            result.push(person);
        } catch (exception) {
            console.log(exception);
        }
    });
    return result;
}
```

使用 try 语句包裹关键代码，如果 try 语句有异常会被抛出那么 catch 语句就会执行。使用之前的代码：

```
> getPersons([2, -5, 137])
[Error: ID must not be negative: -5]
[ { id: 2 }, { id: 137 } ]
```

1.12 严格模式

严格模式（参见 7.8“严格模式”）激活更多的警告以及使 JavaScript 变得更干净（非严格模式有时候被叫作“松散模式”）。要切换到严格模式，在 JavaScript 文件或者 <script>标签第一行输入：

```
'use strict';
```

你也可以在每一个函数中激活严格模式：

```
function functionInStrictMode() {
    'use strict';
}
```

1.13 变量作用域和闭包

在 JavaScript 中，通过在变量前使用 var 语句声明变量：

```
> var x;
> x = 3;
> y = 4;
ReferenceError: y is not defined
```

你可以使用单个 var 语句声明和初始化多个变量：

```
var x = 1, y = 2, z = 3;
```

但是我推荐使用单独声明每一个变量（原因参考 26.4.1 “语法”）。因此，我会将之前的语句重写为：

```
var x = 1;
var y = 2;
var z = 3;
```

由于前置的缘故（参考 1.13.2 “变量的提升特性”），通常它的最佳实践是在一个函数的开始部分声明变量。

1.13.1 变量是函数作用域的

一个变量的作用域总是完整的函数。

1.13.2 变量的提升特性

所有变量声明都会被提升：声明会被移动到函数的开始处，而赋值则仍然会在原来的位置进行。

1.13.3 闭包

每个函数都和他周围的变量保持着连接，哪怕它离不开被创建时的作用域也是如此。

1.13.4 IIFE 模式：引用一个新的作用域

1.14 对象和构造函数

1.14.1 单一对象

和所有的值一样，对象也具有属性。你可以认为对象是一组属性的集合，事实也是如此，每个属性都是一个（键，值）对。键名都是字符串，而值可以是 JavaScript 的任意值。

在 JavaScript 中，可以直接通过对象字面量去创建普通对象：

```
'use strict';
var jane = {
    name: 'Jane',

    describe: function () {
        return 'Person named '+this.name;
    }
};
```

上述对象具有 name 和 describe 两个属性。你可以获取（get）以及设置（set）这些属性：

```
> jane.name  // get
'Jane'
> jane.name = 'John';  // set
> jane.newProperty = 'abc';  // property created automatically
```

以函数作为值的属性被称为方法，如 describe。它们使用 this 对调用它们的对象进行引用：

```
> jane.describe()  // call method
'Person named John'
> jane.name = 'Jane';
> jane.describe()
'Person named Jane'
```

使用 in 运算符检查属性是否存在：

```
> 'newProperty' in jane
true
> 'foo' in jane
false
```

如果读取一个不存在的属性，会得到 undefined。因此，之前的两个检查可以这样执行：

```
> jane.newProperty !== undefined
true
> jane.foo !== undefined
false
```

使用 delete 运算符移除属性：

```
> delete jane.newProperty
true
> 'newProperty' in jane
false
```

1.14.2 任意属性名

属性的键名可以是任何字符串。迄今为止，我们见到过对象字面量中的属性名和点运算符后的属性名。然而，只有当它们是标识符的时候才可以这样使用（参见 1.3.3 “标识符与变量名”）。如果想用其他的字符串作为属性名，则必须将它们用引号引起来，再通过对象字面量和方括号来获取或设置这个属性：

```
> var obj = { 'not an identifier': 123 };
> obj['not an identifier']
123
> obj['not an identifier'] = 456;
```

方括号可以用来动态计算属性键名：

```
> var obj = { hello: 'world' };
> var x = 'hello';

> obj[x]
'world'
> obj['hel'+'lo']
'world'
```

1.14.3 提取方法

如果对方法进行提取，则会失去与对象的连接。就这个函数而言，它不再是一个方法，this 的值也会是 undefined（在严格模式下）。

看如下示例，先回到之前的 jane 对象：

```
'use strict';
var jane = {
    name: 'Jane',

    describe: function () {
        return 'Person named '+this.name;
    }
};
```

我们要从 jane 对象中提取 describe 方法，将它赋值给变量 func，然后对它进行调用。你会发现，它不能正常运行：

```
> var func = jane.describe;
> func()
TypeError: Cannot read property 'name' of undefined
```

处理这个问题的解决方案可以使用 bind() 方法，所有函数都支持。它会创建一个 this 总是指向给定值的新函数：

```
> var func2 = jane.describe.bind(jane);
> func2()
'Person named Jane'
```

1.14.4 方法中的函数

所有函数都有其特殊的 this 变量。如果在方法中有嵌套函数，这可能会不太方便，因为在嵌套函数内部不能访问方法中的 this 变量。下面这个例子展示了调用 forEach 并结合一个函数来遍历数组：

```
var jane = {
    name: 'Jane',
    friends: [ 'Tarzan', 'Cheeta' ],
    logHiToFriends: function () {
        'use strict';
        this.friends.forEach(function (friend) {
            // `this` is undefined here
            console.log(this.name+' says hi to '+friend);
        });
    }
}
```

调用 logHiToFriends 会产生一个错误：

```
> jane.logHiToFriends()
TypeError: Cannot read property 'name' of undefined
```

让我们来看看这个问题的两种解决方法。第一种，我们可以将 this 保存在不同的变量中：

```
logHiToFriends: function () {
    'use strict';
    var that = this;
    this.friends.forEach(function (friend) {
        console.log(that.name+' says hi to '+friend);
    });
}
```

第二种，利用 forEach 的第二个参数，它可以给 this 指定一个值：

```
logHiToFriends: function () {
    'use strict';
    this.friends.forEach(function (friend) {
        console.log(this.name+' says hi to '+friend);
    }, this);
}
```

函数表达式在 JavaScript 中通常被当作函数调用中的参数来使用。在这些函数表达式中引用 this 时要特别小心。

1.14.5 构造函数：对象工厂

到现在为止，JavaScript 对象字面量表现出的那种类似于其他语言中映射表/字典的印象，可能会使你觉得 JavaScript 对象仅仅是字符串到值的映射。然而，JavaScript 对象也支持真正的面向对象：继承。本节不会去完全解释 JavaScript 的继承是如何工作的，而会展示一种简单的模式让你快速上手。想了解更多详情，请查看第 17 章。

除了“真正的”函数和方法，函数在 JavaScript 中还扮演了另外一个角色：如果用 new 运算符来调用的话，它们将变成构造函数即对象工厂。构造函数就是这样简单地模拟了其他语言的类。按照惯例，构造函数的名称以大写字母开头。例如：

```
// Set up instance data
function Point(x, y) {
    this.x = x;
    this.y = y;
}
// Methods
Point.prototype.dist = function () {
    return Math.sqrt(this.x*this.x + this.y*this.y);
};
```

可以看到构造函数包含两部分。第一部分，Point 函数设置实例数据。第二部分，Point.prototype 属性包含一个带有方法的对象。第一部分里的实例数据是特定于每一个实例的，而之后的方法数据则是对所有实例共享的。

可以通过 new 运算符来使用 Point：

```
> var p = new Point(3, 5);
> p.x
3
> p.dist()
5.830951894845301
```

p 是 Point 的一个实例：

```
> p instanceof Point
true
```

1.15 数组

数组是一些有序的元素，可以通过证书索引从 0 开始被访问

1.15.1 数组方法

功能	字符串	数组
长度	length	length
获取指定内容的位置	indexOf(value)	indexOf(value)
获取指定位置的内容	charAt(i) str[i]	array[i]
子串获取（提取）	slice()	slice()
相互转换	split(separator)	join(separator)
添加内容	+ 拼接符号	push() shift() concat() splice()
删除内容		pop() unshift() splice()
循环遍历	for	for forEach()
其他	toUpperCase() toLowerCase()	sort() reverse()

1.16 正则表达式

JS 内置的支持正则表达式。它们使用斜线分割：`/^abc$/` `/[A-Za-z0-9]+/`

1.16.1 test（）方法：匹配吗

```
> /^a+b+$/.test('aaab')
true
> /^a+b+$/.test('aaa')
false
```

1.16.2 exec()方法：匹配以及捕获分组

```
> /a(b+)a/.exec('_abbba_aba_')
[ 'abbba', 'bbb' ]
```

返回的数组包含完整的匹配结果，它的索引从 0 开始，第一组被捕获的内容的索引是 1，以此类推。有一种方法（详见 19.6“RegExp.prototype.exec: 捕获分组”）可调用这个方法反复匹配所有内容。

1.16.3 replace（）方法：搜索和替换

```
> '<a> <bbb>'.replace(/<(.*?)>/g, '[$1]')
'[a] [bbb]'
```

Replace 的第一个参数必须是一个带着/g 标志的正则表达式；否则将只替换第一次出现的内容。还有一种方法可以使用一个函数来完成替换。

1.17 math

Math（参见第 21 章）是一个包含运算功能的对象。示例如下：

```
> Math.abs(-2)
2

> Math.pow(3, 2)  // 3 to the power of 2
9
> Math.max(2, -1, 5)
5

> Math.round(1.9)
2

> Math.PI  // pre-defined constant for n
3.141592653589793

> Math.cos(Math.PI)  // compute the cosine for 180°
-1
```

1.18 标准库的其他功能

JavaScript 的标准库相对简陋，但是有许多我们可以使用的：

Date（第 20 章）

一个日期构造器，主要功能是解析和创建日期字符串和访问组件的日期（年、小时等）。

JSON（第 22 章）

一个可以解析和生成 JSON 数据的对象。

console.* 系列方法（参考 23.5“Console API”）

这些浏览器特定的方法不是这个语言的一部分，但是有一些同样可以运行在 Node.js 中。