第2章 JavaScript编程基础

在上一章中，我们回顾了JavaScript的发展历史，然后搭建了JavaScript的运行环境，并书写了我们的第一个JavaScript程序。在本章中，我们会更加深入一点，开始具体的学习JavaScript的基础语法。

本章中我们将学习如下的内容：

- JavaScript的基础语法
- 变量的声明和赋值
- 简单值和复杂值的区别
- 什么是作用域，以及全局作用域和局部作用域介绍
- JavaScript中的6种简单数据类型及其属性和方法
- 运算符与运算符的优先级

2-1 JavaScript基础语法

2-1-1 注释

关于注释的作用，这里就不用多说了。之所以书写注释，就是为了让阅读代码的人更加方便。拥有良好注释的代码是专业程序员的标志，它让阅读我们代码的任何人(包括我们自己)能更加容易的理解某一段代码的作用是什么。如果没有注释，往往在几周后，我们回过头来阅读自己写的代码时会发现，自己都看不懂。

在JavaScript中，注释可以分为单行注释和多行注释。
单行注释如下：

```
//这是一个单行注释
```

多行注释如下：

```
/*
 *
 * 这是一个较长的
 * 多行的注释
 *
 */
```

不过，虽然说写注释是一个好习惯，这也不是意味着每一句代码都要写注释。往往我们是针对某一个功能来书写一个注释说明，像下面这样的注释是毫无意义的

```
let i = 5;//将5赋值给i变量
```

2-1-2 语句

在JavaScript中，语句一般我们都会采用以分号结尾，每条语句独占一行的形式来书写代码。当然，如果一条语句后面不添加分号，也不会报错，但是在进行代码压缩的时候可能会带来一些问题。所以还是建议每条语句加上分号。

```
let i = 10;
console.log(i);//10
```

可以使用C语言的风格用一对大括号将多条语句组合到一个代码块里面。

```
{
    let i = 10;
    console.log(i);//10
}
```

2-1-3 标识符

所谓标识符，就是指用来标识某个实体的一个符号。再说通俗一点，就是自己起一个名字，这个名字可以用来作为变量名，函数名，对象名等。在JavaScript中，虽然说标识符是自己取名字，但是也是需要遵守一定的规则，其命名的规则大致可以分为2大类：**硬性要求**和**软性要求**

硬性要求

- 1.可以是由数字，字母，下划线和美元符号组成，不允许包含其他特殊符号
- 2.不能以数字开头
- 3.禁止使用JavaScript中的关键词和保留字来进行命名
- 4.严格区分大小写

软性要求

- 望文知意

命名的3种方法

1. 匈牙利命名法

匈牙利命名法是微软公司下面以为匈牙利籍的程序员所发明的命名法则，其特点是标识符的名字以一个或者多个小写字母开头，表示了该变量的数据类型。匈牙利命名法中特定字母所对应的含义如下表：

数据类型	对应前缀
Array数组	a
Boolean布尔	b
Float浮点	f
Function	fn
Interger(int)整型	i
Object对象	o
Regular Expression正则	re
String字符串	s

前缀之后的是一个单词或多个单词的组合，该单词表明变量的用途。
例如：a_array，o_object，i_userAge，b_isPassed

2. 驼峰命名法

驼峰命名法实际上分为两种，一种是大驼峰，另外一种是小驼峰。

- 大驼峰又被称之为帕斯卡命名法，就是每一个单词都是首字母大写
  例如：UserName
- 小驼峰和大驼峰的区别在于，小驼峰的第一个单词的首字母是小写，后面单词的首字母是大写
  例如：userName

3. 蛇形命名法

这种命名法常见于Linux内核，C++标准库，Boost以及Ruby，Rust等语言
蛇形命名法的特点在于单词与单词之间使用下划线进行分隔
例如：user_name，my_first_name

2-1-4 关键字和保留字

首先需要弄清楚关键字和保留字的区别是什么。
ECMA-262描述了一组具有特定用途的关键字。这些关键字可用于表示控制语句的开始或结束，或者用于执行特定操作等。按照规则，关键字是语言本身所保留的，不能用作标识符。

ECMA-262还描述了另外一组不能用作标识符的保留字。尽管保留字在这门语言中还没有任何特定的用途。但它们有可能在将来被用作关键字。

JavaScript中的关键字与保留字如下：

abstract、await、boolean、break、byte、case、catch、char、class、const、continue、debugger、default、delete、do、double、else、enum、export、extends、false、final、finally、float、for、function、goto、if、implements、import、in、instanceof、int、interface、let、long、native、new、null、package、private、protected、public、return、short、static、super、switch、synchronized、this、throw、throws、transient、true、try、typeof、var、volatile、void、while、with、yield

2-1-5 数据类型介绍

在JavaScript中，数据类型整体上来讲可以分为两大类：简单数据类型和复杂数据类型

- 简单数据类型
  简单数据类型一共有6种：
  string，symbol，number，boolean，undefined，null
  其中symbol类型是在ES6里面新添加的基本数据类型

- 复杂数据类型
  复杂数据类型就只有1种
  object
  包括JavaScript中的数组，正则等，其类型都是object类型

查看数据类型

在JavaScript中，我们可以通过`typeof`运算符来查看一个数据的数据类型

```
console.log(typeof 10);//number
console.log(typeof true);//boolean
console.log(typeof 'Hello');//string
console.log(typeof [1,2,3]);//object
```

2-2 变量

接下来我们需要探讨一下对于任何编程语言来讲都是非常重要的一个东西，**变量**。所谓变量，就是用于引用内存中存储的一个值。当然，我们在使用变量之前，还需要先做的一件事儿就是声明变量。

2-2-1 声明变量

在JavaScript中声明变量的方式有3种：`var`，`let`，`const`。其中`var`现在已经不推荐使用了，因为会有变量提升等问题。(后面我们会具体来探讨此问题)
`const`和`let`的区别在于，`const`所声明的变量如果是简单数据类型，那么是不能够再改变的。而`let`所声明的变量无论是简单数据类型还是复杂数据类型，在后面是可以改变的。示例如下：

const声明变量

```
const name = 'Bill';
name = 'Lucy';
//TypeError: Assignment to constant variable.
```

let声明变量

```
let name = 'Bill';
name = 'Lucy';
console.log(name);
```

2-2-2 变量赋值

关于变量的命名我们这里不在多说，只需要遵循标识符的命名规则即可。接下来要做的就是要给变量赋值。变量的赋值，大致可以分为两大类：**直接赋值**和**引用赋值**。要讲清楚什么是直接赋值什么是引用赋值，就先要从简单值和复杂值入手。
前面有提到过，在JavaScript里面，分为了**简单数据类型**以及**复杂数据类型**。其中简单数据类型所创建的值就是简单值，复杂数据类型所创建的值就为复杂值。

那么简单值和复杂值具体有什么区别呢？下面我们一个一个来看：

1. 简单值(或原始值)

简单值是表示JS中可用的数据或信息的最底层形式或最简单形式。简单类型的值被称为简单值，是因为它们是**不可细化**的。
也就是说，数字是数字，字符是字符，布尔值是true或false，null和undefined就是null和undefined。这些值本身很简单，不能够再进行拆分。由于简单值的数据大小是固定的，所以简单值的数据是存储于内存中的栈区里面的。

```
let str = "Hello World";
let num = 10;
let bol = true;
let myNull = null;
let undef = undefined;
console.log(typeof str);//string
console.log(typeof num);//number
console.log(typeof bol);//boolean
console.log(typeof myNull);//object
console.log(typeof undef);//undefined
```

这里面就null和udnefined比较特殊，打印出来的数据类型分别是`object`和`undefined`。因为这两个数据类型没有对应的构造函数，当然什么是构造函数我们后面再说，现在我们只需要将null和undefined看作是特殊的操作符来使用即可。

2. 复杂值

在JavaScript中，对象就是一个复杂值。因为对象可以向下拆分，拆分成多个简单值或者复杂值。复杂值在内存中的大小是未知的，因为复杂值可以包含任何值，而不是一个特定的已知值。所以复杂值的数据都是存储于堆区里面。

```
//简单值
let str = "Hello World";
let num = 10;
let bol = true;
let myNull = null;
let undef = undefined;
//复杂值
let obj = {
    name : 'xiejie',
    age : 18,
    isPassed : true
};
let arr = [1,2,3,"Hello",true];
```

3. 访问方式

按值访问

简单值是作为不可细化的值进行存储和使用的，引用它们会转移其值

```
let str = "Hello";
let str2 = str;
str = null;
console.log(str,str2);//null "Hello"
```

引用访问

复杂值是通过引用进行存储和操作的，而不是实际的值。创建一个包含复杂对象的变量时，其值是内存中的一个引用地址。引用一个复杂对象时，使用它的名称(即变量或对象属性)通过内存中的引用地址获取该对象值

```
let obj = {};
let obj2 = obj;
obj.name = "xiejie";
console.log(obj.name);//xiejie
console.log(obj2.name);//xiejie
```

4. 比较方式

简单值采用值比较，而复杂值采用引用比较。复杂值只有在引用相同的对象(即有相同的地址)时才相等。即使是包含相同对象的两个变量也彼此不相等，因为它们并不指向同一个对象

示例1:

```
let a = 10;
let b = 10;
let c = new Number(10);
let d = c;
console.log(a === b);//true
console.log(a === c);//false
console.log(a == c);//true
d = 10;
console.log(d == c);//true
console.log(d === c);//false
```

示例2:

```
let obj = {name : 'xiejie'};
let obj2 = {name : 'xiejie'};
```

```
console.log(obj == obj2);//false
console.log(obj === obj2);//false
let obj3 = {name : 'xiejie'};
let obj4 = obj3;
console.log(obj3 == obj4);//true
console.log(obj3 === obj4);//ture
```

5. 动态属性

对于复杂值，可以为其添加属性和方法，也可以改变和删除其属性和方法；但简单值不可以添加属性和方法

```
let str = 'test';
str.abc = true;
console.log(str.abc);//undefined
let obj = {};
obj.abc = true;
console.log(obj.abc);//true
```

复杂值支持动态对象属性，因为我们可以定义对象，然后创建引用，再更新对象，并且所有指向该对象的变量都会获得更新。
一个新变量指向现有的复杂对象，并没有复制该对象。这就是复杂值有时被称为引用值的原因。
复杂值可以根据需求有任意多个引用，即使对象改变，它们也总是指向同一个对象

```
let obj = {name : 'xiejie'};
let obj2 = obj;
let obj3 = obj2;
obj.name = 'abc';
console.log(obj.name,obj2.name,obj3.name);
//abc abc abc
```

6. 变量赋值

到这里，我们已经知道了简单值和复杂值的区别。那么直接赋值，就是指将简单值赋值给变量，而引用赋值是指将一个复杂值引用赋值给变量，这个引用指向堆区实际存在的数据

直接赋值

```
let a = 3;
let b = a;
b = 5;
console.log(a);//3
```

引用赋值

```
let a = {value : 1};
let b = a;
b.value = 10;
console.log(a.value);//10
```

2-2-3 变量的初始化

给变量第一次赋值的过程，叫做变量的初始化。一般我们在声明变量的时候就会将变量给初始化

```
let a = 3;
```

当然我们也可以一次性初始化多个变量，将其写在一行里面。

```
let x = 3,y = 4,z = 5;
```

如果声明变量时没有赋予初值，那么默认值为`undefined`

```
let a;
console.log(a);//undefined
```

2-2-4 使用var声明变量

前面有提到过，在JavaScript中声明变量的方式有3种：`var`，`let`，`const`。其中`var`现在已经不推荐使用了。这是因为使用`var`来声明变量会伴随着一些问题。当然，这些问题也经常被看作是JavaScript的一些特点，例如重复声明和遗漏声明。

重复声明

如果是使用`var`关键字来声明的变量，那么是允许重复声明的。只不过这个时候会忽略此次声明。如果重新声明并且带有赋值，则相当于是重新赋值

重复声明不带有赋值操作，JS引擎会自动忽略后面的变量声明

```
var test = 3;
var test;
```

```
console.log(test);//3
```

重新声明时如果带有赋值操作，那么会进行一个数据的覆盖

```
var test = 3;
var test = 5;
console.log(test);//5
```

需要注释重复声明仅仅是使用`var`关键字时可以这样，如果是在严格模式中，或者使用`let`或者`const`的话是会报错的。

遗漏声明

如果试图读取一个没有声明的变量的值，JS会报错

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

JS允许遗漏声明，即直接对变量赋值而无需事先声明，赋值操作会自动声明该变量

```
{
    a = 5;
    console.log(a);//5
}
console.log(a);//5
```

2-2-5 作用域

所谓作用域，就是变量在程序中能够被访问到的区域。这里我们介绍一个全局作用域，一个局部作用域

1.全局作用域

这是JS引擎一进来就处于的运行环境。在全局作用域的中所声明变量称之为全局变量。全局变量的特点在于变量在任何地方都能被访问。

```
let a = 5;//这是一个全局变量
```

2.局部作用域

在JavaScript中，一对大括号就可以产生一个局部作用域。局部作用域里面的变量称之为局部变量，既然是局部变量，那么就只能在这个局部的作用域里面能访问到，外部是访问不到的。

```
{
    let i = 10;
    console.log(i);//10
}
console.log(i);
//ReferenceError: i is not defined
```

顺带一提的是，在大括号中用`var`声明的变量不是局部变量，而是一个全局变量。这其实也是最早使用`var`来声明变量所遗留下来的一个问题。

```
{
    var i = 10;
    console.log(i);//10
}
console.log(i);//10
```

在局部作用域里面，如果变量名和全局作用域里面的变量名冲突，优先使用局部作用域里面的变量

```
let i = 10;
{
    let i = 100;
    console.log(i);//100
}
console.log(i);//10
```

如果在局部作用域里面声明变量时没有书写关键字，那么会声明一个全局变量

```
{
    i = 10;
}
console.log(i);//10
```

2-3 数据类型

在前面，我们有介绍过，在JavaScript中存在6种简单数据类型以及1种复杂数据类型。那么接下来，就让我们一起来看一下JavaScript中6种简单的数据类型

2-3-1 undefined类型

`undefined`类型就只有一个值，undefined。在使用变量但是没有为其赋值的时候，这个变量的值就是undefined。

还需要注意一点，就是没有申明的变量，使用时会报错，而不是undefined。但是打印其类型的时候，显示的类型却是undefined

```
let i;
console.log(typeof i);//undefined
console.log(typeof j);//undefined
console.log(i);//undefined
console.log(j);
//ReferenceError: j is not defined
```

2-3-2 null类型

`null`类型的值也是只有一个，就是null。`null`表示一个空的对象。从逻辑角度来看，null值表示一个空对象指针，这也正是用typeof操作符检测null值时会返回object的原因。
示例：

```
let i = null;
console.log(typeof i);//object
```

实际上，undefined值是从null值派生而来的，因此当我们对这两个数据类型进行相等测试时，会返回true
示例：

```
if(null == undefined)
{
    console.log('Yes');//Yes
}
```

2-3-3 布尔类型

所谓布尔类型，也被称之为`boolean`类型，就是真和假，这个类型的值只有两个，一个是true，另一个是false

```
let i = true;
console.log(i);//true
console.log(typeof i);//boolean
```

需要注意的是，这两个值与数字值不是一回事，因此true不一定等于1，而false也不一定等于0。还有一点就是Boolean类型的字面值true和false是区分大小写的。也就是说，True和False都不是布尔值。

虽然Boolean类型的字面值只有2个，但是ECMAScript中所有类型的值都可以转换为Boolean类型。可以使用`Boolean()`函数将其他类型转换为布尔值。

```
console.log(Boolean("Hello"));//true
console.log(Boolean(42));//true
console.log(Boolean(0));//false
```

最后需要注意的就是下面的9个值是对应布尔类型里面的假值

- "": 双引号的空字符串
- '': 单引号的空字符串
- ``: 空字符串模板
- 0: 数字0
- -0: JS中-0和0为不同的值
- NaN
- false
- null
- undefined

2-3-4 数字类型

数字类型又被称之为`number`类型。`number`类型的值可以分为整数和实数两大类

1. 整数

整数可以分为正整数和负整数，如下:

```
let a = 12;
let b = -7;
```

这个虽然没什么好说的，但是还是有一个注意点，那就是进制问题。二进制以0b开头，八进制以0开头，十六进制以0x，示例如下：

```
//二进制
let a = 0b101;//5
//八进制
let b = 017;//15
//十进制
let c = 21;//21
//十六进制
let d = 0xFD;//253
console.log(a,b,c,d);
```

需要注意的是，不管参与运算的变量的值是什么进制，计算结果仍然会为十进制。
在ES6中提供了八进制数值新的写法，使用0o作为前缀，如下：

```
let a = 017;
let b = 0o17;
console.log(a,b);//15 15
```

2. 实数

所谓实数，就是我们平常所常见的小数，或者称之为浮点数。
在JavaScript里面，表示浮点数的方式有两种：小数型和科学记数法型
示例如下：

```
let a = 3.14;
let b = 9.12e+2;
console.log(a,b);//3.14 912
```

3. 数值范围

由于内存限制，JavaScript并不能保存世界上所有的数值。在JS中能够表示的最小数值在绝大多数浏览器中为5e-324，而最大值为1.7976931348623157e+308。通过`Number.MIN_VALUE`和`Number.MAX_VALUE`我们可以查看到JS中支持的最小值和最大值。
示例：

```
console.log(Number.MIN_VALUE);//5e-324
console.log(Number.MAX_VALUE);//1.7976931348623157e+308
```

如果某次计算的结果超出了ECMAScript的数值范围，那么正数会被转化为infinity（正无穷），负数会被转换为-infinity（负无穷）
例如：
根据ECMAScript的规定大于等于2的1024次方的数为无穷大

```
let i = Math.pow(2,1024);
console.log(i);//Infinity
```

根据ECMAScript的规定小于等于2的1024次方的数为无穷小

```
let i = -Math.pow(2,1024);
console.log(i);//-Infinity
```

如果某次计算返回了infinity值，那么该值无法参与下一次计算，因为infinity不是能够参与计算的数值。要想确定一个数值是不是有穷的，可以使用isFinite()函数。
示例：如果是无穷大数，会返回false，否则返回true(其实这个函数就是用于判断一个变量是否为数值的)

```
let i = Math.pow(2,1024);
console.log(isFinite(i));//fasle
let j = 7;
console.log(isFinite(j));//true
```

4. NaN

英文全称为Not a Number，即非数值。这个数值用于表示本来要返回数值的操作数未返回数值的情况（这样就不会抛出错误了）。
NaN有两个特点：任何涉及NaN的操作都会返回NaN
示例：

```
let a = NaN + 10;
console.log(a);//NaN
```

第二个特点是NaN和任何值都不相等，包括它自己本身，示例如下：

```
console.log(NaN === NaN);//false
```

除此之外，ECMAScript还定义了一个isNaN()函数，来帮助我们判断一个参数是否不是数值。isNaN()函数在接收到一个值之后，会尝试将这个值转换为数值。
示例：

```
console.log(isNaN(NaN));//true
console.log(isNaN("123"));//false
console.log(isNaN(123));//false
console.log(isNaN("Hello"));//true
console.log(isNaN(true));//false
```

最后需要补充说明一下的是，NaN是属于`number`类型的，证明如下：

```
console.log(typeof NaN);//number
```

5. 数值转换

在JavaScript里面，有3个函数可以将非数值的数据转为数值，分别是`Number()`，`parseInt()`以及`parseFloat()`这3个方法，下面我们将依次对这几个方法进行介绍

Number():可以将非数值转为数值
使用`Number()`函数的时候，有下面几个规则需要注意：

- 如果是Boolean值，true和false将分别被转换为1和0
- 如果是数字，那么就是简单的传入和返回
- 如果是null值，那么返回0
- 如果是undefined，那么返回NaN

示例如下：

```
console.log(Number(true));//1
console.log(Number(false));//0
console.log(Number(10));//10
console.log(Number(null));//0
console.log(Number(undefined));//NaN
```

如果是字符串，那么又有如下的规则：

- 如果字符串只包含数字，则将其转为十进制，即”1”会变成1，”123”变成123，而”011”会变为11（注意这里不会被当成八进制来处理）
- 如果字符串中包含有效的十六进制格式，如”1.1”，则将其转换为对应的浮点数值
- 如果字符串中包含有效的十六进制格式，例如”0xf”，则会将其转换为相同大小的十进制整数。
- 如果字符串是空的，则将其转换为0
- 如果字符串包含上述格式之外的字符，则将其转换为NaN

示例：

```
console.log(Number("1"));//1
console.log(Number("012"));//12
console.log(Number("0o10"));//8
console.log(Number("0b111"));//7
console.log(Number("3.14"));//3.14
console.log(Number("0xf"));//15
console.log(Number(""));//0
console.log(Number("123Hello"));//NaN
```

parseInt():也是将一个非数值转为数值
说明：相比number()函数，parseInt()会更多的看是否有数字，有就会将其转换为数值。最简单的例子为number()函数转换”123Hello”时会转换为NaN，而parseInt()则会将其转换为123。
碰到空字符串时，number()函数会将其转换为0，而parseInt()则会将其转换为NaN
最后是在碰到小数时，会有一个取整的过程。例如”3.14”会被转换为”3”
示例：

```
console.log(parseInt("1"));//1
console.log(parseInt("012"));//12
console.log(Number("0o10"));//8
console.log(Number("0b111"));//7
console.log(parseInt("3.14"));//3
console.log(parseInt("0xf"));//15
console.log(parseInt(""));//NaN
console.log(parseInt("123Hello"));//123
```

我们的`parseInt()`从ES5开始还提供了第二个参数，那就是指定参数是多少进制，这样`parseInt()`函数就可以将指定的进制转换为十进制，如下：

```
console.log(parseInt("012"));//12
console.log(parseInt("012",8));//10
//如果是16进制数,后面添加上参数16后,字符串前面可以不写0x
```

```
console.log(parseInt("AF"));//NaN
console.log(parseInt("AF",16));//175
```

除此之外，我们的`parseInt()`函数的第2个参数还可以是任意值，八进制前面不用加前面的0，十六进制不用加前面的0x

```
console.log(parseInt("21",3));//7
console.log(parseInt("10",2));//2
console.log(parseInt("11",8));//9
console.log(parseInt("AF",16));//175
```

parseFloat():将非数值转为浮点数
说明：parseFloat()只解析十进制值，所以没有第二个参数，该函数会将带有小数点的字符串转换为小数。如果没有小数点的数会被转换为整数。同样的parseFloat()也是尽可能转换更多的数值，例如”123Hello”会被转换为123.
示例：

```
console.log(parseFloat("21"));//21
console.log(parseFloat("123Hello"));//123
console.log(parseFloat("0xA"));//0
console.log(parseFloat("3.14"));//3.14
console.log(parseFloat("22.34.5"));//22.34
console.log(parseFloat("0908.34"));//908.34
console.log(parseFloat("3.1415e2"));//314.15
```

ES6 将全局方法`parseInt()`和`parseFloat()`等方法，移植到`number`对象上面，行为完全保持不变，这么做的目的，是为了逐步减少全局性的方法，使得语言逐步模块化。

```
//ES5的写法
console.log(parseInt("12.34"));//12
console.log(parseFloat("12.34#"));//12.34
//ES6的写法
console.log(Number.parseInt("12.34"));//12
console.log(Number.parseFloat("12.34#"));//12.34
```

6. 算数运算

知道了`number`这种数据类型后，我们就可以使用这种类型来做最基本的算数运算了。
加减乘除和取模运算
需要注意的就是做除法运算时，能够得到小数

```
console.log(5 + 3.14);//8.14
console.log(6 - 11);//-5
console.log(7 * 3);//21
console.log(5 / 2);//2,5
console.log(10 % 3);//1
```

从ES6开始新增加了求幂运算，使用两个*号代表求幂。以此可以代替以前的`Math.pow()`方法

```
console.log(2 ** 3);//8
```

7. 递增和递减

递增和递减也是数值类型中经常会用到的操作。这里主要就是要注意前置和后置的区别。如果是前置，那么是先自增或自减，然后参与运算。如果是后置，则是先参与运算，然后再自增或者自减，示例如下：

前置示例：

```
let a = 2;
let b = 10;
let c = --a + b;
let d = a + b;
console.log(a,b,c,d);//1 10 11 11
```

后置示例：

```
let a = 2;
let b = 10;
let c = a-- + b;
let d = a + b;
console.log(a,b,c,d);//1 10 12 11
```

需要注意的是，我们的自增自减操作符不仅仅局限于数值，其他类型也可以，遵循下面的规则：

- 在应用于一个包含有效数字字符的字符串时，现将其转换为数字值，再执行加减1操作。字符串变量变为了数值变量。
- 在应用于一个不包含有效数字字符的字符串时，将变量的值设置为NaN，字符串变量变成数值变量。
- 遇布尔值false时，先将其转换为0再执行加减1操作，布尔值变量变成数值变量。
- 遇布尔值true时，先将其转换为1再执行加减1操作，布尔值变量变成数值变量。

- 在应用浮点数数值时，执行加减1操作。

示例：

```
let s1 = "123";
let s2 = "123Hello";
let s3 = "Hello";
let s4 = true;
let s5 = 3.14;
console.log(--s1);//122
console.log(--s2);//NaN
console.log(--s3);//NaN
console.log(--s4);//0
console.log(--s5);//2.14
```

8. 静态方法

这里是第一次听到静态方法这个说法。什么是静态方法呢?
所谓静态方法，主要是涉及到了面向对象里面的知识，这里简单理解，就是通过该类型能够直接调用的方法。下面介绍两个Number类型的静态方法

Number.isInteger():用来判断一个值是否为整数。需要注意的是，在JS内部，整数和浮点数是同样的存储方法，所以3和3.0被视为同一个值

```
console.log(Number.isInteger(25));//true
console.log(Number.isInteger(25.0));//true
console.log(Number.isInteger(25.1));//false
console.log(Number.isInteger("15"));//false
console.log(Number.isInteger(true));//false
```

Number.isFinite():这个方法我们前面在介绍数值范围的时候已经给大家介绍过了。如果一个值是字符串，布尔类型，Infinity，-Infinity，NaN等(总之就不是数字的时候)，则返回false，如果是数字的话就会返回true。并且不会进行自动类型转换。

```
console.log(Number.isFinite(true));//false
console.log(Number.isFinite(7));//true
console.log(Number.isFinite(NaN));//false
console.log(Number.isFinite(Infinity));//false
console.log(Number.isFinite("23"));//false
```

9. 实例方法

首先解释一下什么叫做实例方法。所谓实例方法，就是指必须要实例化对象，然后在对象上面调用。这也涉及到了后面面向对象的知识。现在只需要知道要先有一个数，然后在这个数上面调用相应的方法。

toFixed():toFixed()方法按照指定的小数位返回数值四舍五入后的字符串表示(常用于处理货币值)

> 注意：toFixed()里的参数只接受0-20，若不传参或参数为undefined则相当于参数是0

```
let num = 10.456;
console.log(num.toFixed(2));//10.46
console.log(num.toFixed());//10
console.log(num.toFixed(0));//10
console.log(num.toFixed(undefined));//10
console.log(num.toFixed(-1));//报错
//RangeError: toFixed() digits argument must be between 0 and 20
```

toExponential():返回数值四舍五入后的指数表示法(e表示法)的字符串表示，参数表示转换后的小数位数

> 注意：toExponential()方法里的参数只接受0-20，但与toFxied()不同的是，若不传参或参数为undefined，则保留尽可能多的有效数字；若参数是0表示没有小数部分

```
let num = 10.456;
console.log(num.toExponential(2));//1.05e+1
console.log(num.toExponential());//1.0456e+1
console.log(num.toExponential(0));//1e+1
console.log(num.toExponential(undefined));//1.0456e+1
console.log(num.toExponential(-1));//报错
//RangeError: toExponential() argument must be between 0 and 20
```

toPrecision():接收一个参数，即表示数值的所有数字的位数(不包括指数部分)，自动调用toFixed()或toExponential()

> 注意：toPrecision()里的参数只接受1-21，若不传参或参数为undefined则相当于调用toString()方法

```
let num = 10.456;
console.log(num.toPrecision(3));//10.5
console.log(num.toPrecision(2));//10
console.log(num.toPrecision(1));//1e+1
console.log(num.toPrecision());//10.456
console.log(num.toPrecision(undefined));//10.456
console.log(num.toPrecision(0));//报错
```

```
//RangeError: toPrecision() argument must be between 1 and 21
```

> 注意：`toFixed()`、`toExponential()`、`toPrecision()`这三个方法在小数位用于四舍五入时都不太可靠，跟浮点数不是精确储存有关

```
console.log((12.25).toPrecision(3));//12.3
console.log((12.25).toFixed(1));//12.3
console.log((12.25).toExponential(2));//1.23e+1
console.log((12.35).toPrecision(3));//12.3
console.log((12.35).toFixed(1));//12.3
console.log((12.35).toExponential(2));//1.23e+1
```

2-3-5 字符串类型

1. 字符串介绍

这是程序里面使用最为广泛的一种类型。在JavaScript里面，可以使用单引号，也可以使用双引号

```
let a = "abcd";
let b = 'abcd';
```

这里需要注意一个问题，就是字符串的内容本身包含单引号或者双引号的话，需要和字符串界限符区分开，如下：

```
let a = "Hello 'World',welcome";//正确
let b = 'Hello "World",welcome';//正确
let c = "Hello "World",welcome";//错误
```

当然，我们这里使用转义字符也是一个很好的选择，如下：

```
let a = "Hello 'World',welcome";//正确
let b = 'Hello "World",welcome';//正确
let c = "Hello \"World\",welcome";//正确
```

字符串这种数据类型非常的霸道，它和其他数据类型相加都会被转换为字符串类型，示例如下：

```
let a = "abcd";
let b = 13 + a;
```

```
let c = 3.14 + a;
let d = true + a;
let e = null + a;
let f = undefined + a;
console.log(typeof b);//string
console.log(typeof c);//string
console.log(typeof d);//string
console.log(typeof e);//string
console.log(typeof f);//string
```

所以如果我们要让一个非字符串的变量转换为字符串的话，只需要和一个空字符串相加就可以了。
当然，转换字符串事实上我们也有相应的函数来转换，最常见的就是toString()函数。

toString()
说明：该函数会将除了null和undefined以外的数据类型转换为字符串。
示例：

```
let a = 10,b = true,c = null,d;
console.log(typeof a.toString());//string
console.log(typeof b.toString());//string
console.log(typeof c.toString());//报错
console.log(typeof d.toString());
```

可以看到，程序报错，因为null和undefined并不能通过toString()函数来转换为相应的字符串。
还有一个知识点，就是toString()函数在转换数值的时候是可以带有参数的。可以将数值指定转换为几进制
示例：

```
let i = 10;
console.log(i.toString());//10
console.log(i.toString(2));//1010
console.log(i.toString(8));//12
console.log(i.toString(10));//10
console.log(i.toString(16));//a
```

这个时候，JavaScript还有一个函数是String()，这个函数就可以5种数据类型都转换为字符串。
示例：

```
let a = 10,b = true,c = null,d;
console.log(String(a),typeof String(a));//10 string
console.log(String(b),typeof String(b));//true string
```

```
console.log(String(c),typeof String(c));//null string
console.log(String(d),typeof String(d));//undefined string
```

2. 字符串属性

在字符串里面有一个 `length` 属性，可以访问到该字符串里面有多少个字符

```
console.log("Hello".length);//5
```

3. 字符串方法

访问特定字符

有两个访问字符串中特定字符的方法，分别是charAt()和charCodeAt()

charAt()：接收一个数字参数，找到对应该下标的字符是什么

```
let str = "Hello World";
console.log(str.charAt(1));//e
console.log(str.charAt('a'));//H 因为a被转为了数字0
```

charCodeAt()：接收一个数字参数，找到对应该下标的字符编码是什么

```
let str = "Hello World";
console.log(str.charCodeAt(1));//101
console.log(str.charCodeAt('a'));//72
```

fromCharCode()：这个方法基本上和charCodeAt()执行相反的操作，如下：

```
console.log(String.fromCharCode(104,101,108,108,111));//hello
```

字符串操作方法

concat()：用于将一个或者多个字符串拼接起来，返回拼接得到的新字符串

需要注意的就是原字符串不会产生改变，拼接后的字符串以返回值的方式返回

```
let str = "Hello";
let newStr = str.concat(" World!!!");
console.log(str);//Hello
console.log(newStr);//Hello World!!!
```

还有一点，那就是虽然concat()可以用于拼接字符串，但是还是使用+号拼接的方式比较简便，使用的情况更多一些

slice()：和数组的slice()方法相似，接收一或者两个参数，截取字符串

```
let str = "Hello World";
let str2 = str.slice(2);
let str3 = str.slice(2,7);//不包括7
console.log(str);//Hello World
console.log(str2);//llo World
console.log(str3);//llo W
```

substr()：在字符串中抽取从开始下标开始的指定数目的字符，效果和slice()方法基本一样。但是还是有一定的区别，区别就在于第二个参数，如下：

```
let str = "Hello World";
let str1 = str.slice(2);
let str2 = str.substr(2);
console.log(str1);//llo World
console.log(str2);//llo World

str1 = str.slice(2,7);//结束位置为7,不包含7
str2 = str.substr(2,7);//要返回的字符个数
console.log(str1);//llo W
console.log(str2);//llo Wor
```

substring()：用于提取字符串中介于两个指定下标之间的字符。和前面的方法也是非常的相似，但是还是有不同的地方

```
let str = "Hello World";
let str1 = str.slice(2);
let str2 = str.substr(2);
let str3 = str.substring(2);
console.log(str1);//llo World
console.log(str2);//llo World
console.log(str3);//llo World

str1 = str.slice(2,7);//结束位置为7,不包含7
str2 = str.substr(2,7);//要返回的字符个数
str3 = str.substring(2,7);//结束位置为7,不包含7
console.log(str1);//llo W
console.log(str2);//llo Wor
console.log(str3);//llo W
```

普通的传入2个参数的时候，好像substring()方法和slice()方法就是一样的方法，那么区别在什么地方呢？区别就在于参数为负数的时候，如下：

```
let str = "Hello World";
let str1 = str.slice(-3);
let str2 = str.substr(-3);
let str3 = str.substring(-3);
//slice 将所有的负值和字符串的长度相加
//substr 负的第一个参数和字符串长度相加 负的第二个参数转换为0
//substring 把所有的负值转换为0 substring()会将较小的数作为开始 较大的数作为结束
console.log(str1);//rld
console.log(str2);//rld
console.log(str3);//Hello World

str1 = str.slice(2,-3);//等价于slice(2,8)
str2 = str.substr(2,-3);//等价于substr(2,0)
str3 = str.substring(2,-3);//等价于substring(2,0) 等价于substring(0,2)
console.log(str1);//llo Wo
console.log(str2);//空
console.log(str3);//He
```

字符串位置方法

indexOf()和lastIndexOf()：这两个方法都是从一个字符串中搜索给定的子字符串，然后返回子字符串的位置，没有找到就是返回-1，两个方法的区别在于一个是从前面开始找，一个是从后面开始找

```
let str = "Hello World";
console.log(str.indexOf('l'));//2
console.log(str.lastIndexOf('l'));//9
```

查看是否包含字符

includes()：如果包含返回true，否则返回false

```
let str = "Hello World";
console.log(str.includes("l"));//true
console.log(str.includes("M"));//false
```

检测开始字符和结束字符

startsWith() 和 endsWith()

```
let str = "Hello World";
```

```
console.log(str.startsWith("H"));//true
console.log(str.endsWith("d"));//true
console.log(str.endsWith("z"));//false
```

去除字符串空白方法

有一个去除字符串两端空白的方法trim()，这个在表单里面经常会用

trim()：这个方法会创建一个字符串副本，删除前置以及后缀的所有空格。

```
let str = "    Hello World    ";
let newStr = str.trim();
console.log(str);//    Hello World
console.log(newStr);//Hello World
```

除了trim()以外，还有trimLeft()和trimRight()方法，分别用于删除字符串开头和末尾的空格。

重复字符串

repeat()：里面传入要重复的次数即可

```
let str = "Hello";
console.log(str.repeat(3));
//HelloHelloHello
```

字符串大小写转换方法

这里涉及到4个方法，分别是toLowerCase()和toLocaleLowerCase()，还有就是toUpperCase()和toLocaleUpperCase()。

用得较多的一般就是toLowerCase()和toUpperCase()

```
let str = "HELLO";
console.log(str.toLowerCase());//hello
str = "hello";
console.log(str.toUpperCase());//HELLO
```

4. 字符串模板

字符串模板

在ES6中新增了模板字面量是增强版的字符串，它用反引号（`）标识

```
let str = `Hello World`;
console.log(str);//Hello World
console.log(typeof str);//string
```

```
console.log(str.length);//11
```

以上代码中，使用模板字面量语法创建一个字符串，并赋值给message变量，这时变量的值与一个普通的字符串无异
如果想在字符串中包含反引号，只需使用反斜杠（\）转义即可

ES6 的模板字面量使多行字符串更易创建，因为它不需要特殊的语法，只需在想要的位置直接换行即可，此处的换行会同步出现在结果中

```
let str = `Hello
World`;
console.log(str);
//Hello
//World
console.log(typeof str);//string
console.log(str.length);//12
```

在反引号之内的所有空白符都是字符串的一部分，因此需要特别留意缩进

模板字面量看上去仅仅是普通JS字符串的升级版，但二者之间真正的区别在于模板字面量的变量占位符。变量占位符允许将任何有效的JS表达式嵌入到模板字面量中，并将其结果输出为字符串的一部分
变量占位符由起始的 ${ 与结束的 } 来界定，之间允许放入任意的 JS 表达式。最简单的变量占位符允许将本地变量直接嵌入到结果字符串中

```
let name = "xiejie";
console.log(`Hello,${name}`);
//Hello,xiejie
```

占位符 ${name} 会访问本地变量 name ，并将其值插入到 message 字符串中。 message变量会立即保留该占位符的结果

既然占位符是JS表达式，那么可替换的就不仅仅是简单的变量名。可以轻易嵌入运算符、函数调用等

```
let count = 10,price = 0.25;
console.log(`${count} items cost $${(count*price).toFixed(2)}`);
//10 items cost $2.50
```

2-3-6 symbol类型

ES5中包含5种原始类型：字符串、数字、布尔值、null和undefined。ES6引入了第6种原始类型:symbol

ES5的对象属性名都是字符串，很容易造成属性名冲突。比如，使用了一个他人提供的对象，想为这个对象添加新的方法，新方法的名字就有可能与现有方法产生冲突。如果有一种机制，保证每个属性的名字都是独一无二的，这样就从根本上防止了属性名冲突。这就是ES6引入symbol的原因。

我们在讲解面向对象的时候再具体来讲解symbol类型。

2-3-7 类型转换

型转换可以分为两种，隐性转换和强制转换

1. 隐性转换

当不同数据类型进行相互运算的时候，
当对非布尔类型的数据求布尔值的时候

预期为数字的时候：
算术运算的时候，我们的结果和运算的数都是数字，数据会转换为数字进行计算(- * / %)

类型	转换前	转换后
number	4	4
string	‘1’	1
string	‘abc’	NaN
string	‘’	0
boolean	true	1
boolean	false	0
undefined	undefined	NaN
null	null	0

预期为字符串的时候：
转为字符串 使用`+`号时候，会自动转换为字符串

预期为布尔的时候：

转换为布尔值，undefined，null，''”，0，NaN转为false，其余转为true

2. 强制转换

转换为数值`number()`， `parseInt()`， `parseFloat()`转换函数
小技巧：

- 转换字符串: a=""+数据
- 转换布尔: !数据类型
- 转换数值: 数据类型*或/1

2-4 运算符

任何编程语言基本上都离不开运算符。在JavaScript中也是支持众多的运算符。除开最常用的算数运算符之外，常见的还有逻辑运算符，位运算符和比较运算符等。接下来，就让我们一起来看一下JavaScript中这几种常见的运算符。

2-4-1 逻辑运算符

1. 非

所谓非，就是取反，非真即假，非假即真

```
let i = true;
console.log(!i);//false
```

非运算符不仅仅只能用于布尔值，其他数据类型也是可以的，如下：

- 如果操作数是一个对象，返回false
- 如果操作数是一个空字符串，返回true
- 如果操作数是一个非空字符串，返回false
- 如果操作数是数值0，返回true
- 如果操作数是任意非0数值(包括Infinity)，返回false
- 如果操作数是null，返回true
- 如果操作数是NaN，返回true
- 如果操作数是undefined，返回true

```
console.log(!false);//true
console.log(!"blue");//false
console.log(!0);//true
console.log(!NaN);//true
console.log(!"");//true
console.log(!12);//false
```

可使用双否定`!!`来判定一个值是真值还是假值，如下：

```
console.log(!!'');//false
console.log(!!NaN);//false
console.log(!!'Hello');//true
```

2. 与

作用于两到多个值，并且只有所有的操作数都是真值时，才为true

```
console.log(false && true);//false
console.log(true && true);//true
```

JavaScript里面的与不一定返回真或假，如果所有操作数都为真，那么返回值是最后一个真值，如果至少有一个操作数为假，那么返回值就是第一个假值

```
console.log(3 && 5);//5
console.log("Hello" && 20);//20
console.log("" && "shoe");//""
```

与存在短路操作。即第一个操作符为假的话，就不会再对第二个操作数进行判断了。
练习：

```
let a = true;
let b = a && c;//因为a是true,所以会判断第2个数
console.log(b);
//ReferenceError: c is not defined
```

```
let a = false;
let b = a && c;//因为a是false,所以不会判断第2个数
console.log(b);//false
```

其次还存在其他规则：

- 如果第一个操作数是对象，则返回第二个操作数
- 如果第二个操作数是对象，则只有在第一个操作数的求值结果为true的情况下才会返回该对象
- 如果两个操作数都是对象，则返回第二个操作数
- 如果有一个操作数是null，则返回null
- 如果有一个操作数是NaN，则返回NaN
- 如果有一个操作数是undefined，则返回undefined

```
console.log(3 && 5);//5
console.log(NaN && NaN);//NaN
console.log(null && null);//null
```

```
console.log(undefined && undefined);//undefined
```

3. 或

这里重点要讲一下JavaScript里面的或。
作用于两到多个值，但是只要有一个操作数为真，就返回真

```
console.log(false || true);//true
console.log(true || false);//true
```

如果有一个操作数为真，返回的值是第一个真值
如果所有的操作数都是假，返回的值就是最后一个假值

```
console.log(false || true);//true
console.log("Hello" || "");//Hello
console.log("Hello" || "str");//Hello
console.log(NaN || "");//""
console.log(0 || "Hello World");//Hello World
```

具体的规则遵循如下规则：

- 如果第一个操作数是对象，则返回第一个操作数
- 如果第一个操作数的求值结果为false，则返回第二个操作数
- 如果两个操作数都是对象，则返回第一个操作数
- 如果两个数都是null，则返回null
- 如果两个数都是NaN，则返回NaN
- 如果两个数都是undefined，则返回undefined

```
console.log(3 || 5);//3
console.log(NaN || NaN);//NaN
console.log(null || null);//null
console.log(undefined || undefined);//undefined
```

或运算符也存在短路现象，如果第一个操作数为真的话，就不会再进行第二个数的判断。

```
let a = false;
let b = a || c;//因为a是false,所以会判断第2个数
console.log(b);
//ReferenceError: c is not defined
```

```
let a = true;
let b = a || c;//因为a是false,所以会判断第2个数
console.log(b);//true
```

2-4-2 位运算符

按位运算符是将操作数换算成32位的二进制整数，然后按每一位来进行运输。例如：

- 5 的32位为 00000000000000000000000000000101
- 100 的32位为 00000000000000000000000001100100
- 15 的32位为 00000000000000000000000000001111

1. 按位非

按位非运算符 `~` 会把数字转为32位二进制整数，然后反转每一位
所有的1变为0，所有的0变为1
5 的32位为 00000000000000000000000000000101
~5：11111111111111111111111111111010 转换出来就为 -6
按位非，实质上是对操作数求负，然后减去1

2. 按位与

按位或运算符 `&` 会把两个数字转为32位二进制整数，并对两个数的每一位执行按位与运算

第一个数字	第二个数字	结果
1	1	1
1	0	0
0	1	0
0	0	0

具体示例：

```
console.log(12 & 10);//8
```

12的32位二进制表示为：1100
10的32位二进制表示为：1010
结果为：1000

3. 按位或

按位或运算符`|`会把两个数字转为32位二进制整数，并对两个数的每一位执行按位或运算

第一个数字	第二个数字	结果
1	1	1
1	0	1
0	1	1
0	0	0

具体示例：

```
console.log(12 | 10);//14
```

12的32位二进制表示为：1100
10的32位二进制表示为：1010
结果为：1110

4. 按位异或

按位或运算符`^`会把两个数字转为32位二进制整数，并对两个数的每一位执行按位异或运算，两位不同返回1，两位相同返回0

第一个数字	第二个数字	结果
1	1	0
1	0	1
0	1	1
0	0	0

具体示例：

```
console.log(12 ^ 10);//6
```

12的32位二进制表示为：1100
10的32位二进制表示为：1010

结果为：0110

按位异或如果是非整数值，如果两个操作数中只有一个为真，就返回1，如果两个操作数都是真，或者都是假，就返回0

```
console.log(true ^ "Hello");//1
console.log(false ^ "Hello");//0
console.log(true ^ true);//0
console.log("Hello" ^ "Hello");//0
console.log(false ^ false);//0
console.log(true ^ false);//1
```

注意这里的Hello被转换为了NaN

5. 按位移位

按位移位运算符 `<<` 和 `>>` 会将所有位向左或者向右移动指定的数量，实际上就是高效率地将数字乘以或者除以2的指定数的次方
`<<`：乘以2的指定数次方

```
console.log(2<<2);//8
```

2乘以2的2次方
00000010转换为00001000

`>>`：除以2的指定数次方

```
console.log(16>>1);//8
```

16除以2的1次方
00010000转换为00001000

2-4-3 比较运算符

1. 关系运算符

小于，大于，小于等于，大于等于
数的比较就不同说了，如下：

```
console.log(5 > 3);//true
```

```
console.log(3 > 5);//false
```

主要说一下字符串的比较，如下：

```
console.log("c" > "b");//true
console.log("c" > "box");//true
```

这里的比较主要是按照ASCII码来进行比较的。
如果是一个字符串和一个数字进行比较，那么会将字符串先转换为数字，如果不能转换为数字，则转换为NaN

```
console.log("5" > 3);//true,因为"5"转为了5
//任何一个数与NaN进行关系比较,返回的都是false
console.log("Hello" > 3);//false,因为"Hello"转为了NaN
```

完整的特殊规则如下：

- 如果两个数都是数值，则执行数值比较
- 如果两个数都是字符串，则比较两个字符串对应的字符编码
- 如果一个操作数是数值，则将另一个操作数转换为一个数值，然后执行数值的比较
- 如果一个操作数是对象，则调用这个对象的`valueOf()`方法，用得到的结果按照前面的规则执行比较。如果对象没有`valueOf()`方法，则调用`toString()`方法，并用得到的结果根据前面的规则执行比较。
- 如果一个数是布尔值，则先将其转换为数值，然后再进行比较

还需要注意，任何数和NaN进行比较返回的都是false

```
console.log(10 > NaN);//false
console.log(10 < NaN);//false
```

2. 相等和不相等

`==`表示相等，`!=`表示不相等，数据类型不同的数据进行相等比较的话会自动转换数据类型，还有一些其他的转换规则如下：

- null和undefined是相等的
- 如果有一个操作数是NaN，则返回false，NaN和NaN比较也是false
- 如果是数字的字符串和数字进行比较，会先将字符串转换为数字
- 布尔值里面true转换为1，false转换为0

下表列出了一些特殊的情况

表达式	值
null == undefined	true
"NaN" == NaN	false
5 == NaN	false
NaN == NaN	false
NaN != NaN	true
false == 0	true
true == 1	true
true == 2	false
undefined == 0	false
null == 0	false
"5" == 5	true

3. 全等和不全等

全等是`===`，不全等是`!==`这个就是必须数据类型和数值都相等，如下：

```
console.log("5" == 5);//true
console.log("5" === 5);//false
```

2-4-4 运算符优先级

JavaScript中的运算符优先级是一套规则。该规则在计算表达式时控制运算符执行的顺序。具有较高优先级的运算符先于较低优先级的运算符执行。
例如，乘法的执行先于加法。

下表按从最高到最低的优先级列出JavaScript运算符。具有相同优先级的运算符按从左至右的顺序求值。

运算符	描述
	字段访问，数组下标，函数调用以及表达

`.` `[]` `()`	式分组
`++` `--` `-` `~` `!` `delete` `new` `typeof` `void`	一元运算符，返回类型，对象创建，未定义值
`*` `/` `%`	乘法，除法，取模
`+` `-` `+`	加法，减法，字符串拼接
`<<` `>>` `>>>`	移位
`<` `<=` `>` `>=` `instanceof`	小于，小于等于，大于，大于等于，instanceof
`==` `!=` `===` `!==`	等于，不等于，全等，不全等
`&`	按位与
`^`	按位异或
`|`	按位或
`&&`	逻辑与
`||`	逻辑或
`?:`	三目运算符
`=`	赋值
`,`	多重赋值

总结

1. 拥有良好注释的代码是专业程序员的标志。如果没有注释，往往在几周后自己也很难明白自己写的代码的含义。
2. 在JavaScript中，每一条代码以分号结束。
3. 标识符就是我们自己给变量。函数或者对象起的一个名字。定义标识符时需要遵循一定的硬性要求和软性要求。
4. 常见的命名法则有匈牙利命名法，驼峰命名法和蛇形命名法。
5. 在JavaScript中存在一组关键字和保留字，不能用作自定义标识符。
6. 在JavaScript中数据类型整体上可以分为简单数据类型和复杂数据类型。
7. 使用typeof运算符可以查看一个数据的数据类型。
8. 在JavaScript中声明变量可以使用let，const和var关键字来进行声明。

9. 简单值和复杂值在访问方式，比较方式以及内存中存储空间的区别。
10. 使用var关键字声明变量存在一些特殊的特性，例如重复声明以及遗漏声明
11. 所谓作用域就是指变量在程序中能够被访问到的区域。常见的有全局作用域和局部作用域。
12. JavaScript中存在6种基本数据类型，分别为undefined，null，boolean，number，string和symbol。
13. 每种数据类型都有其对应的属性和方法。掌握这些属性和方法可以让我们在编写代码时事半功倍。
14. 不同的数据类型之间可以相互进行转换，其中分为隐性转换和强制转换
15. JavaScript中有众多的运算符，除开最常用的算数运算符以外，常见的还有逻辑运算符，位运算符和比较运算符。不同的运算符之间有一套优先级关系。