Number类型

最基本的数值字面量格式是十进制整数，十进制整数可以像下面这样直接在代码中输入：

```
var intNum = 55; // 整数
```

除了以十进制表示外，整数还可以通过八进制（以 8 为基数）或十六进制（以 16 为基数）的字面值

来表示。其中，八进制字面值的第一位必须是零（0），然后是八进制数字序列（0～7）。如果字面值中的

数值超出了范围，那么前导零将被忽略，后面的数值将被当作十进制数值解析。请看下面的例子：

```
var octalNum1 = 070; // 八进制的 56
var octalNum2 = 079; // 无效的八进制数值——解析为 79
var octalNum3 = 08; // 无效的八进制数值——解析为 8
```

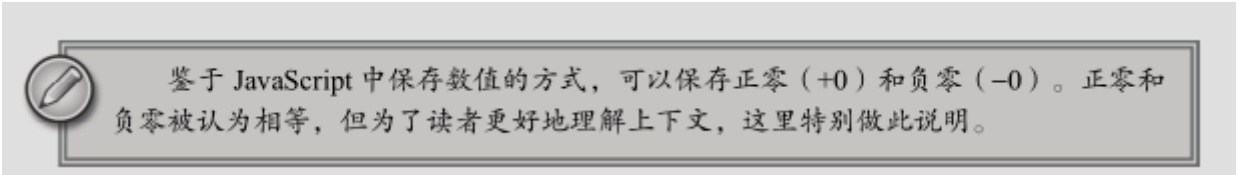
八进制字面量在严格模式下是无效的，会导致支持的 JavaScript 引擎抛出错误。

十六进制字面值的前两位必须是 0x，后跟任何十六进制数字（0～9 及 A～F）。其中，字母 A～F

可以大写，也可以小写。如下面的例子所示：

```
var hexNum1 = 0xA; // 十六进制的 10
var hexNum2 = 0x1f; // 十六进制的 31
```

在进行算术计算时，所有以八进制和十六进制表示的数值最终都将被转换成十进制数值。



对于那些极大或极小的数值，可以用 e 表示法（即科学计数法）表示的浮点数值表示。用 e 表示法表示的数值等于 e 前面的数值乘以 10 的指数次幂。ECMAScript 中 e 表示法的格式也是如此，即前面是一个数值（可以是整数也可以是浮点数），中间是一个大写或小写的字母 E，后面是 10 的幂中的指数，该幂值将用来与前面的数相乘。下面是一个使用 e 表示法表示数值的例子：

```
var floatNum = 3.125e7;    // 等于 31250000
```

也可以使用 e 表示法表示极小的数值，如 0.00000000000000003，这个数值可以使用更简洁的 3e□17

表示。在默认情况下，ECMASctipt 会将那些小数点后面带有 6 个零以上的浮点数值转换为以 e 表示法

表示的数值（例如，0.0000003 会被转换成 3e□7）。

浮点数值的最高精度是 17 位小数，

关于浮点数值计算会产生舍入误差的问题，有一点需要明确：这是使用基于 IEEE754 数值的浮点计算的通病，ECMAScript 并非独此一家；其他使用相同数值格式的语言也存在这个问题。

2. 数值范围

由于内存的限制，ECMAScript 并不能保存世界上所有的数值。ECMAScript 能够表示的最小数值保存在 Number.MIN_VALUE 中——在大多数浏览器中，这个值是 5e-324；能够表示的最大数值保存在 Number.MAX_VALUE 中——在大多数浏览器中，这个值是 1.7976931348623157e+308。如果某次计算的结果得到了一个超出 JavaScript 数值范围的值，那么这个数值将被自动转换成特殊的 Infinity 值。具体来说，如果这个数值是负数，则会被转换成-Infinity（负无穷），如果这个数值是正数，则会被转换成 Infinity（正无穷）。

如上所述，如果某次计算返回了正或负的 Infinity 值，那么该值将无法继续参与下一次的计算，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（换句话说，是不是位于最小和最大的数值之间），可以使用 isFinite()函数。这个函数在参数位于最小与最大数值之间时会返回 true，如下面的例子所示：

```
var result = Number.MAX_VALUE + Number.MAX_VALUE;
alert(isFinite(result));   //false
```

3. NaN

NaN ，即非数值（Not a Number）是一个特殊的数值，这个数值用于表示一个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 本身有两个非同寻常的特点。首先，任何涉及 NaN 的操作（例如 NaN/10）都会返回 NaN，这个特点在多步计算中有可能导致问题。其次，NaN 与任何值都不相等，包括 NaN 本身。例如，下面的代码会返回 false：

```
alert(NaN == NaN);     //false
```

针对 NaN 的这两个特点，ECMAScript 定义了 isNaN() 函数。这个函数接受一个参数，该参数可以

是任何类型，而函数会帮我们确定这个参数是否“不是数值”。 isNaN() 在接收到一个值之后，会尝试

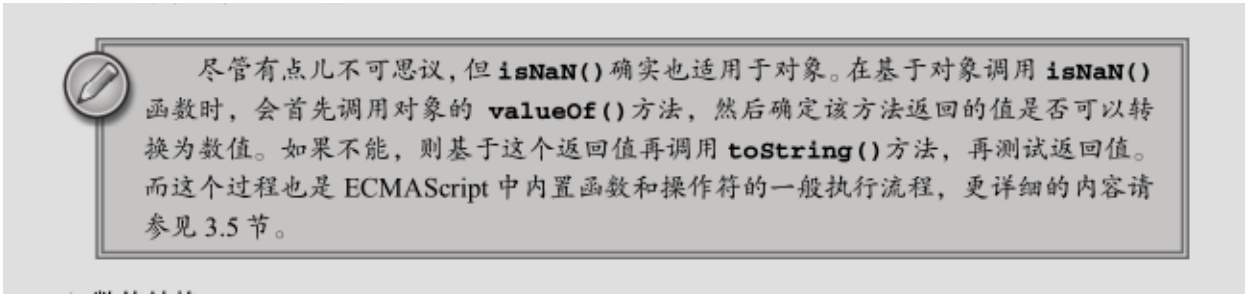
将这个值转换为数值。某些不是数值的值会直接转换为数值，例如字符串 ″10″ 或 Boolean 值。而任何不能被转换为数值的值都会导致这个函数返回 true 。请看下面的例子：

```
alert(isNaN(NaN)); //true
alert(isNaN(10)); //false（10 是一个数值）
```

alert(isNaN("10")); //false（可以被转换成数值 10）

alert(isNaN("blue")); //true（不能转换成数值）

alert(isNaN(true)); //false（可以被转换成数值 1）



数值转换

有 3 个函数可以把非数值转换为数值： Number() 、 parseInt() 和 parseFloat() 。第一个函数，

即转型函数 Number() 可以用于任何数据类型，而另两个函数则专门用于把字符串转换成数值。这 3个

函数对于同样的输入会有返回不同的结果。

Number() 函数的转换规则如下。

□ 如果是 Boolean 值， true 和 false 将分别被转换为 1 和 0。

□ 如果是数字值，只是简单的传入和返回。

□ 如果是 null 值，返回 0。

□ 如果是 undefined ，返回 NaN 。

□ 如果是字符串，遵循下列规则：

□ 如果字符串中只包含数字（包括前面带正号或负号的情况），则将其转换为十进制数值，即 "1"

会变成 1， "123" 会变成 123，而 "011" 会变成 11（注意：前导的零被忽略了）；

□ 如果字符串中包含有效的浮点格式，如 "1.1" ，则将其转换为对应的浮点数值（同样，也会忽

略前导零）；

□ 如果字符串中包含有效的十六进制格式，例如 "0xf" ，则将其转换为相同大小的十进制整

数值；

□ 如果字符串是空的（不包含任何字符），则将其转换为 0；

□ 如果字符串中包含除上述格式之外的字符，则将其转换为 NaN 。

□ 如果是对象，则调用对象的 valueOf() 方法，然后依照前面的规则转换返回的值。如果转换

的结果是 NaN ，则调用对象的 toString() 方法，然后再次依照前面的规则转换返回的字符
串值。

```
var num1 = Number("Hello world!");      //NaN
var num2 = Number("");                  //0
var num3 = Number("000011");            //11
var num4 = Number(true);                //1
```

parseInt() 函数在转换字符串时，更多的是看其是否符合数值模式。它会忽略字符串前面的空格，直至找到第一个非空格字符。如果第一个字符不是数字字符或者负号，parseInt()
就会返回 NaN ；也就是说，用 parseInt() 转换空字符串会返回 NaN （ Number() 对空字符返回 0）。如
果第一个字符是数字字符， parseInt() 会继续解析第二个字符，直到解析完所有后续字符或者遇到了
一个非数字字符。例如， "1234blue" 会被转换为 1234，因为 "blue" 会被完全忽略。类似地， "22.5"
会被转换为 22，因为小数点并不是有效的数字字符。

```
var num1 = parseInt("1234blue");    // 1234
var num2 = parseInt("");            // NaN
var num3 = parseInt("0xA");         // 10 (十六进制数)
var num4 = parseInt(22.5);          // 22
var num5 = parseInt("070");         // 56 (八进制数)
var num6 = parseInt("70");          // 70 (十进制数)
var num7 = parseInt("0xf");         // 15 (十六进制数)
```

为了消除在使用 parseInt()函数时可能导致的上述困惑，可以为这个函数提供第二个参数：转换时使用的基数（即多少进制）。如果知道要解析的值是十六进制格式的字符串，那么指定基数 16 作为第二个参数，可以保证得到正确的结果，例如：

```
var num = parseInt("0xAF", 16);     //175
```

实际上，如果指定了 16 作为第二个参数，字符串可以不带前面的"0x"，如下所示：

```
var num1 = parseInt("AF", 16);      //175
var num2 = parseInt("AF");          //NaN
```

不指定基数意味着让 parseInt()决定如何解析输入的字符串，因此为了避免错误的解析，我们建议无论在什么情况下都明确指定基数。

多数情况下，我们要解析的都是十进制数值，因此始终将 10 作为第二个参数是非常必要的。

parseFloat() 除了第一个小数点有效之外，与 parseInt() 的第二个区别在于它始终都会忽略前导

的零。 parseFloat() 可以识别前面讨论过的所有浮点数值格式，也包括十进制整数格式。但十六进制格式的字符串则始终会被转换成 0。由于 parseFloat() 只解析十进制值，因此它没有用第二个参数指定基数的用法。最后还要注意一点：如果字符串包含的是一个可解析为整数的数（没有小数点，或者小数点后都是零）， parseFloat() 会返回整数。以下是使用 parseFloat() 转换数值的几个典型示例。

```
var num1 = parseFloat("1234blue"); //1234 （整数）
var num2 = parseFloat("0xA"); //0
var num3 = parseFloat("22.5"); //22.5
var num4 = parseFloat("22.34.5"); //22.34
var num5 = parseFloat("0908.5"); //908.5
var num6 = parseFloat("3.125e7"); //31250000
```