**第6章 数值的扩展**

**6.1 二进制和八进制数值表示法**

ES6提供了二进制和八进制数值的新写法，分别用前缀0b（或0B）和0o（或0O）表示。

1. 0b111110111 === 503 // true

2. 0o767 === 503 // true

3.

从ES5开始，在严格模式中，八进制数值就不再允许使用前缀0表示，ES6进一步明确，要使用前缀0o表示。

1. // 非严格模式

2. (function () {

3.     console.log(0o11 === 011);

4. })(); // true

5.

6. // 严格模式

7. (function () {

8.     "use strict";

9.     console.log(0o11 === 011);

10. })(); // SyntaxError: Octal literals are not allowed in strict mode.

11.

如果要将使用0b和0x前缀的字符串数值转为十进制数值，要使用Number方法。

1. Number(0b1111);

2. Number(0o10);

3.

6.2 Number.isFinite()，Number.isNaN()

ES6在Number对象上新提供了Number.isFinite()和Number.isNaN()两个方法，用来检查Infinite和NaNcy这两个特殊值。

Number.isFinite()用来检查一个数值是否非无穷（infinity）。

1. Number.isFinite(15); // true

2. Number.isFinite(0.8); // true

3. Number.isFinite(NaN); // false

4. Number.isFinite(Infinity); // false

5. Number.isFinite(-Infinity); // false

6. Number.isFinite("foo"); // false

7. Number.isFinite("15"); // false

8. Number.isFinite(true); // false

9.

ES5可以通过下面的代码部署Number.isFinite方法。

1. (function (global) {

2.     var global\_isFinite = global.isFinite;

3.

4.     Object.defineProperty(Number, "isFinite", {

5.         value: function isFinite(value) {

6.             return typeof value === "number" && global\_isFinite(value);

7.         },

8.         configurable: true,

9.         enumerable: false,

10.         writable: true

11.     });

12. })(this);

13.

Number.isNaN用来检查一个值是否位NaN。

1. Number.isNaN(NaN); // true

2. Number.isNaN(15); //  false

3. Number.isNaN("15"); // false

4. Number.isNaN(true); // false

5. Number.isNaN(9/NaN); // true

6. Number.isNaN("true"/0); // true

7. Number.isNaN("true"/"true"); // true

8.

ES5可以通过下面的代码部署Number.isNaN方法。

1. (function (global) {

2.     var global\_isNaN = global.isNaN;

3.

4.     Object.defineProperty(Number, "isNaN", {

5.         value: function isNaN(value) {

6.             return typeof value === "number" && global\_isNaN(value);

7.         },

8.         configurable: true,

9.         enumerable: false,

10.         writable: true

11.     });

12. })(this);

13.

14.

这两个新方法与传统的全局方法isFinite()和isNaN()的区别在于，传统方法先调用Number()将非数值转为数值，再进行判断，而新方法只对数值有效，对于非数值一律返回false。

1. isFinite(25); // true

2. isFinite("25"); // true

3. Number.isFinite(25); // true

4. Number.isFinite("25"); // false

5.

6. isNaN(NaN); // ture

7. isNaN("NaN"); // true

8. Number.isNaN(NaN); // true

9. Number.isNaN("NaN"); // false

10.

6.3 Number.parseInt()，Number.parseFloat()

ES6将全局方法parseInt()和parseFloat()移植到了Number对象上，行为完全保持不变。

1. // ES5的写法

2. parseInt("12.34"); // 12

3. parseFloat("123.45#"); // 123.45

4.

5. // ES6的写法

6. Number.parseInt("12.34"); // 12

7. Number.parseFloat("123.45#"); // 123.45

8.

9.

这样做的目的，是逐步减少全局性方法，使语言逐步模块化。

1. Number.parseInt === parseInt // true

2. Number.parseFloat === parseFloat // true

3.

6.4 Number.isInteger()

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

1. Number.isInteger(25); // true

2. Number.isInteger(25.0); // true

3. Number.isInteger(25.1); // false

4. Number.isInteger("14"); // false

5. Number.isInteger(true); // false

6.

ES5可以通过下面的代码部署Numer.isInteger方法。

1. (function (global) {

2.     var floor = Math.floor,

3.         isFinite = global.isFinite;

4.

5.     Object.defineProperty(Number, "isInteger", {

6.         value: function (value) {

7.             return typeof value === "number" && isFinite(value) &&

8.                 value > Number.MIN\_VALUE && value < Number.MAX\_VALUE &&

9.                 floor(value) === value;

10.         },

11.         configurable: true,

12.         enumerable: false,

13.         writable: true

14.     });

15. })(this);

16.

6.5 Number.EPSILON

ES6在Number对象上新增了一个极小的常量---Number.EPSILON。

1. Number.EPSILON

2. // 2.220446049250313e-16

3.

4. Number.EPSILON.toFixed(20);

5. // 0.00000000000000022204

6.

引入一个这么小的量，目的在于为浮点数计算设置一个误差范围。我们知道浮点数计算是不精确的。

1. 0.1 + 0.3

2. // 0.40000000000000002220

3.

4. 0.1 + 0.2 - 0.3

5. // 5.551115123125783e-17

5.

6. 5.551115123125783e-17.toFixed(20)

7. // 0.00000000000000005551

但如果这个误差能够小于Number.EPSILON，我们就可以认为得到了正确结果。

1. 5.551115123125783e-17 < Number.EPSILON

2. // true

因此，Number.EPSILON的实质是一个可以接收的误差范围。

下面的代码为浮点数运算部署了一个误差检查函数。

1. function withinErroMargin(left, right) {

2.     return Math.abs(left - right) < Number.EPSILON;

3. }

4.

5. withinErroMargin(0.1 + 0.2, 0.3);

6. // true

7. withinErroMargin(0.2 + 0.2, 0.3);

8. // false

9.

6.6 安全整数和Number.isSafeInteger()

JavaScript能够准确表示的整数范围在-2^{53}到2^{53}之间（不含两个端点），超过这个范围就无法精确表示。

1. Math.pow(2, 53); // 9007199254740992

2.

3. 9007199254740992 // 9007199254740992

4. 9007199254740993 // 9007199254740992

5.

6. Math.pow(2, 53) === Math.pow(2, 53) + 1

7. // true

8.

9.

上面的代码中，超出2的53次方之后，一个数就不精确了。

ES6引入了Number.MAX\_SAFE\_INTEGER和Number.MIN\_SAFE\_INTEGER两个常量，用来表示这个范围的上下限。

1. Number.MAX\_SAFE\_INTEGER === Math.pow(2, 53) - 1

2. // true

3.

4. Number.MAX\_SAFE\_INTEGER === 9007199254740991

5. // true

6.

7. Number.MIN\_SAFE\_INTEGER === -Number.MAX\_SAFE\_INTEGER

8. // true

9.

10. Number.MIN\_SAFE\_INTEGER === -9007199254740991

11. // true

12.

13.

Number.isSafeInteger()则用来判断一个整数是否落在这个范围内。

1. Number.isSafeInteger("a"); // false

2. Number.isSafeInteger(null); // false

3. Number.isSafeInteger(NaN); // false

4. Number.isSafeInteger(Infinity); // false

5. Number.isSafeInteger(-Infinity); // false

6.

7. Number.isSafeInteger(3); // true

8. Number.isSafeInteger(1.2); // false

9. Number.isSafeInteger(9007199254740990); // true

10. Number.isSafeInteger(9007199254740992); // false

11.

12. Number.isSafeInteger(Number.MIN\_SAFE\_INTEGER - 1); // false

13. Number.isSafeInteger(Number.MIN\_SAFE\_INTEGER); // true

14. Number.isSafeInteger(Number.MAX\_SAFE\_INTEGER); // true

15. Number.isSafeInteger(Number.MAX\_SAFE\_INTEGER + 1); // false

16.

注意，验证运算结果是否落在安全整数的范围内时，不要只验证运算结果，而要同时验证参与运算的每个值。

1. Number.isSafeInteger(9007199254740993);

2. // false

3.

4. Number.isSafeInteger(990);

5. // true

6.

7. Number.isSafeInteger(9007199254740993 - 990);

8. // true

9.

10. 9007199254740993 - 990

11. // 返回结果9007199254740002

12. // 正确答案因该是9007199254740003

13.

14.

上面的代码中，9007199254740993不是一个安全整数，但是Number.isSafeInteger会返回结果，显示计算结果是安全的。这是因为，这个数超出了精度范围，导致在计算机内部以9007199254740992的形式存储。

1. 9007199254740993 === 9007199254740992

2. // true

3.

所以，如果只验证运算结果是否为安全整数，则很可能得到错误结果。下面的函数可以同时验证两个运算数和运算结果。

1. function trusty(left, right, result) {

2.   if (

3.     Number.isSafeInteger(left) &&

4.     Number.isSafeInteger(right) &&

5.     Number.isSafeInteger(result)

6.   ) {

7.     return result;

8.   }

9.

10.   throw new RangeError("Operation cannot be trusted!")

11. }

12.

13. trusty(9007199254740993 , 990, 9007199254740993 - 990);

14. // RangeError: Operation cannot be trusted!

15.

16. trusty(1, 2, 3);

17. // 3

18.

19.

6.7 Math对象的扩展

ES6在Math对象上新增了17个与数学相关的方法。所有这些方法都是静态方法，只能在Math对象上调用。

**Math.trunc()**

Math.truct方法用于去除一个数的小数部分，返回整数部分。

1. Math.trunc(4.1); // 4

2. Math.trunc(4.9); // 4

3. Math.trunc(-4.1); // -4

4. Math.trunc(-4.9); // -4

5. Math.trunc(-0.1234); // 0

6.

对于非数值，Math.trunc内部使用Number方法将其先转为数值。

1. Math.trunc("123,456");

2. // 123

3.

对于空值和无法截取整数的值，返回NaN。

1. Math.trunc(NaN); // NaN

2. Math.trunc("foo"); // NaN

3. Math.trunc(); // NaN

4.

对于没有部署这个方法的环境，可以用下面的代码模拟。

1. Math.trunc = Math.trunc || function (x) {

2.   return x < 0 ? Math.ceil(x) : Math.floor(x);

3. };

4.

**Math.sign()**

Math.sign方法用于判断一个数到底是正数、负数，还是零。

其返回值有5中情况。

* 参数为正数，返回+1。
* 参数为负数，返回-1。
* 参数为0，返回0。
* 参数为-0，返回-0。
* 其他值，返回NaN。

1. Math.sign(-5); // -1

2. Math.sign(5); // 1

3. Math.sign(0); // 0

4. Math.sign(-0); // -0

5. Math.sign(NaN); // NaN

6. Math.sign("foo"); // NaN

7. Math.sign(); // NaN

8.

对于没有部署这个方法的环境，可以用下面的代码模拟。

1. Math.sign = Math.sign || function (x) {

2.   x = +x; // convert to a number

3.   if (x === 0 || isNaN(x)) {

4.     return x;

5.   }

6.   return x > 0 ? 1 : -1;

7. };

8.

**Math.cbrt()**

Math.cbrt方法用于计算一个数的立方根。

1. Math.cbrt(-1); // -1

2. Math.cbrt(0); // 0

3. Math.cbrt(1); // -1

4. Math.cbrt(2); // 1.2599210498948732

5.

对于非数值，Math.cbrt方法内部也是先使用Number方法将其转为数值。

1. Math.cbrt("8"); // 2

2. Math.cbrt(NaN); // NaN

3.

对于没有部署这个方法的环境，可以用下面的代码模拟。

1. Math.cbrt = Math.cbrt || function (x) {

2.   var y = Math.pow(Math.abs(x), 1/3);

3.   return x < 0 ? -y : y;

4. };

5.

**Math.clz32()**

JavaScript的整数使用32位二进制形式表示，Math.clz32方法返回一个数的32位无符号整数形式有多少个前导0。

1. Math.clz32(0); // 32

2. Math.clz32(1); // 31

3. Math.clz32(1000); // 22

4. Math.clz32(0b01000000000000000000000000000000); // 1

5. Math.clz32(0b00100000000000000000000000000000); // 2

6.

上面的代码中，0的二进制形式全为0，所以有32个前导0；1的二进制形式是0b1，只占一位，所以32位中有31个前导0；1000的二进制形式是0b1111101000，一共有10位，所以32位中有22个前导0。

clz32这个函数名字就来自“count leading zero bits in 32-bits binary representations of a number”（计算32位整数的前导零）的缩写。

左移运算符（<<）与Math.clz32方法直接相关。

1. Math.clz32(0); // 32

2. Math.clz32(1); // 31

3. Math.clz32(1 << 1); // 30

4. Math.clz32(1 << 2); // 29

5. Math.clz32(1 << 29); // 2

6.

对于小数，Math.clz32方法只考虑整数部分。

1. Math.clz32(3.2); // 30

2. Math.clz32(3.9); // 30

3.

对于空值或其他类型的值，Math.clz32方法会将其先转为数值，然后再计算。

1. Math.clz32(); // 32

2. Math.clz32(NaN); // 32

3. Math.clz32(Infinity); // 32

4. Math.clz32(null); // 32

5. Math.clz32('foo'); // 32

6. Math.clz32([]); // 32

7. Math.clz32({}); // 32

8. Math.clz32(true); // 31

9.

**Math.imul()**

Math.imul方法返回两个数以32位带符号整数形式相乘的结果，返回的也是一个32位的带符号整数。

1. Math.imul(2, 4); // 8

2. Math.imul(2, -4); // -8

3. Math.imul(-2, -2); // -4

4.

如果只考虑最后32位，大多数情况下，Math.imul(a, b)与a \* b的结果是相同的，即该方法等同于(a \* b) | 0的效果（超过32位的部分溢出）。之所以需要部署这个方法，是因为JavaScript有精度限制，超过2的53次方的值无法精确表示。这就是说，对于那些很大的数的乘法，低位数值往往都是不精确的，Math.imul方法可以返回正确的低位数值。

1. (0x7fffffff \* 0x7fffffff) | 0 // 0

2.

上面这个乘法算式的返回结果为0。但由于这两个二进制数的最低为都是1，所以这个结果肯定是不正确的，因为根据二进制乘法，计算结果的二进制最低位应该也是1。错误的原因就是它们的乘机超过了2的53次方，JavaScript无法保存额外的精度，就把低位的值都变成了0。

Math.imul方法可返回正确的值---1。

1. Math.imul(0x7fffffff, 0x7fffffff); // 1

2.

**Math.fround()**

Math.fround方法返回一个数的单精度浮点数形式。

1. Math.fround(0); // 0

2. Math.fround(1); // 1

3. Math.fround(1.337); // 1.3370000123977661

4. Math.fround(1.5); // 1.5

5. Math.fround(NaN); // NaN

6.

对于整数来说，Math.fround方法的返回结果不会有任何不同，区别主要在于那些无法用64个二进制位精确表示的小数。这时，Math.fround会返回最接近这个小数的单精度浮点数。

对于没有部署这个方法的环境，可以用下面的代码模拟。

1. Math.fround = Math.fround || function (x) {

2.   return new Float32Array([x])[0];

3. };

4.

**Math.hypot()**

Math.hypot方法返回所有参数的平方和的平方根。

1. Math.hypot(3, 4); // 5

2. Math.hypot(3, 4, 5); // 7.0710678118654755

3. Math.hypot(); // 0

4. Math.hypot(NaN); // NaN

5. Math.hypot(3, 4, 'foo'); // NaN

6. Math.hypot(3, 4, '5'); // 7.0710678118654755

7. Math.hypot(-3); // 3

8.

第一行代码中，3的平方加上4的平方，等于5的平方，所以结果为5。

如果参数不是数值，Math.hypot方法会将其转为数值。只要有一个参数无法转为数值，就会返回NaN。

**对数方法**

ES6新增了4个对数相关的方法。

* Math.expm1()

Math.expm1(x)返回e^x-1，即Math.exp(x) – 1。

1. Math.expm1(-1); // -0.6321205588285577

2. Math.expm1(0); // 0

3. Math.expm1(1); // 1.718281828459045

4.

对于没有部署这个方法的环境，可以用下面的代码模拟。

1. Math.expm1 = Math.expm1 || function (x) {

2.   return Math.exp(x) - 1;

3. };

4.

* Math.log1p()

Math.log1p(x)方法返回\ln(1+x)，即Math.log(1+x)。如果x小于-1，则返回NaN。

1. Math.log1p(1); // 0.6931471805599453

2. Math.log1p(0); // 0

3. Math.log1p(-1); // -Infinity

4. Math.log1p(-2); // NaN

5.

对于没有部署这个方法的环境，可以用下面的代码模拟。

1. Math.log1p = Math.log1p || function (x) {

2.   return Math.log(1 + x);

3. };

4.

* Math.log10()

Math.log10(x)返回以10为底的x的对数。如果x小于0，则返回NaN。

1. Math.log10(2); // 0.3010299956639812

2. Math.log10(1); // 0

3. Math.log10(0); // -Infinity

4. Math.log10(-2); // NaN

5. Math.log10(100000); // 5

6.

对于没有部署这个方法的环境，可以用下面的代码模拟。

1. Math.log10 = Math.log10 || function (x) {

2.   return Math.log(x) / Math.LN10;

3. };

4.

* Math.log2()

Math.log2(x)返回以2为底的x的对数。如果x小于0，则返回NaN。

1. Math.log2(3); // 1.584962500721156

2. Math.log2(2); // 1

3. Math.log2(1); // 0

4. Math.log2(0); // -Infinity

5. Math.log2(-2); // NaN

6. Math.log2(1024); // 10

7. Math.log2(1 << 29); // 29

8.

对于没有部署这个方法的环境，可以用下面的代码模拟。

1. Math.log2 = Math.log2 || function (x) {

2.   return Math.log(x) / Math.LN2;

3. };

4.

**三角函数方法**

ES6新增了6个三角函数方法。

* Math.sinh(x): 返回x的双曲正弦（hyperbolic sine）
* Math.cosh(x): 返回x的双曲余弦（hyperbolic cosine）
* Math.tanh(x): 返回x的双曲正切（hyperbolic tangent）
* Math.asinh(x): 返回x的反双曲正弦（inverse hyperbolic sine）
* Math.acosh(x): 返回x的反双曲余弦（inverse hyperbolic cosine）
* Math.atanh(x): 返回x的反双曲正切（inverse hyperbolic tangent）

6.8 指数运算符

ES7新增了一个指数运算符（\*\*），目前Bebel转码器已经支持。

1. 2 \*\* 2 // 4

2. 2 \*\* 3 // 8

3.

指数运算符可以与等号结合，形成一个新的赋值运算符（\*\*=）。

1. let a = 2;

2. a \*\*= 2;

3. // 等同于 a = a \* a

4.

5. let b  = 3;

6. b \*\*= 3;

7. // 等同于 b = b \* b \* b

8.