Javascript 基本概念

# 语法

区分大小写：ECMAScript 中的一切（变量、函数名和操作符）都区分大小。

标识符：指变量、函数、属性的名字，或者函数的参数。

* 第一个字符必须是一个字母、下划线（\_）或一个美元符号（$）；
* 其他字符可以是字母、下划线、美元符号或数字。
* 不能把关键字、保留字、true、false 和null 用作标识符。
* ECMAScript 标识符采用驼峰大小写格式。

注释：

* 单行注释://
* 多行注释/\* …\*/

严格模式：JavaScript 定义了一种不同的解析与执行模型。在严格模式下，ECMAScript 3 中的一些不确定的行为将得到处理，而且对某些不安全的操作也会抛出错误。持严格模式的浏览器包括IE10+、Firefox 4+、Safari 5.1+、Opera 12+和Chrome。

在整个脚本中启用严格模式："use strict";//是一个编译指示

指定函数在严格模式下执行：

function doSomething(){

"use strict";

//函数体 }

语句： ECMAScript 中的语句以一个分号结尾；如果省略分号，则由解析器确定语句的结尾：

var sum = a + b // 即使没有分号也是有效的语句——不推荐

var diff = a - b; // 有效的语句——推荐

# 关键字和保留字

## 关键字（带\*号上标的是第5 版新增的关键字）

|  |
| --- |
| break do instanceof typeof  case else new var  catch finally return void  continue for switch while  debugger\* function this with  default if throw  delete in try |

## 保留字

第三版保留字：

|  |
| --- |
| abstract enum int short  boolean export interface static  byte extends long super  char final native synchronized  class float package throws  const goto private transient  debugger implements protected volatile  double import public |

第五版非严格模式保留字：

|  |
| --- |
| class enum extends super  const export import |

第五版严格模式保留字：

|  |
| --- |
| implements package public  interface private static  let protected yield |

注意：（1）为了最大程度地保证兼容性，建议读者将第3 版定义的保留字外 加let 和yield 作为编程时的参考。

（2）在实现ECMAScript 3 的JavaScript 引擎中使用关键字作标识符，会导致“Identifier Expected”错误。而使用保留字作标识符可能会也可能不会导致相同的错误，具体取决于特定的引擎。

（3）ECMA-262 第5 版对eval 和arguments 还施加了限制。在严格模式下，这两个名字也不能作为标识符或属性名，否则会抛出错误。

# 变量

ECMAScript 的变量是松散类型的，所谓松散类型就是可以用来保存任何类型的数据。并且初始化变量并不会确定变量的类型，变量仅仅是一个用于保存值的占位符而已。

var message = "hi";

message = 100; // 有效，但不推荐

# 数据类型

ECMAScript 中有5 种简单数据类型（也称为基本数据类型）：Undefined、Null、Boolean、Number和String。还有1 种复杂数据类型——Object，Object 本质上是由一组无序的名值对组成的。

## typeof操作符

"undefined"——如果这个值未定义；

"boolean"——如果这个值是布尔值；

"string"——如果这个值是字符串；

"number"——如果这个值是数值；

"object"——如果这个值是对象或null；

"function"——如果这个值是函数。

注意：typeof 是一个操作符而不是函数，因此例子中的圆括号尽管可以使用，但不是必需的。例如：alert(typeof message);或者alert(typeof(message));都可以，但是尽量使用圆括号。

## Undefined类型

对未初始化和未声明的变量执行typeof 操作符都返回了undefined 值.

var message;等效于var message = undefined;另外，使用未申明的

变量是会报错的。（一般做法是：声明变量的时候初始化对象）

## Null类型(一个空对象指针)

只要意在保存对象的变量还没有真正保存对象，就应该明确地让该变量保存null 值：var car = null;

实际上，undefined 值是派生自null 值的：alert(null == undefined); //true

## Boolean类型

Boolean类型的字面量只有true和false，并且区分大小写。

转型函数Boolean()，可以对任何数据类型的值调用Boolean()函数。

* Boolean类型：true,false
* String类型：任何非空字符串未true，空字符串为false
* Number类型：任何非零字符串未true，0和NaN为false
* Object类型：任何对象为true，null为false
* Undefined类型：为false

## Number类型

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。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。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。

### 浮点数值（小数点前面没有整数的写法）

保存浮点数值需要的内存空间是保存整数值的两倍，因此ECMAScript 会不失时机地将浮点数值转换为整数值：浮点数值本身表示的就是一个整数（如1.0），那么该值也会被转换为整数

|  |
| --- |
| var floatNum1 = 1.1;  var floatNum2 = 0.1;  var floatNum3 = .1; // 有效，但不推荐 |

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：var floatNum = 3.125e7; // 等于31250000

浮点数值的最高精度是17 位小数，但在进行算术计算时其精确度远远不如整数。注意：永远不要测试某个特定的浮点数值：，0.1 加0.2的结果不是0.3

### 数值范围

ECMAScript 能够表示的最小数值保存为Number.MIN\_VALUE：5e-324；最大数值保存为Number.MAX\_VALUE：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（不能参与正常运算），可以通过isFinite()函数判断数值是不是在最小和最大的数值之间。

Number.NEGATIVE\_INFINITY 和Number.POSITIVE\_INFINITY 也可以得到负和正Infinity 的值。

### NaN，即非数值（Not a Number）是一个特殊的数值.

特点：任何涉及NaN 的操作（例如NaN/10）都会返回NaN；，NaN 与任何值都不相等，包括NaN 本身。

isNaN()函数：函数接受一个参数，该参数可以是任何类型。isNaN()在接收到一个值之后，会尝试将这个值转换为数值。而任何不能被转换为数值的值都会导致这个函数返回true。

isNaN()确实也适用于对象。在基于对象调用isNaN()函数时，会首先调用对象的valueOf()方法，然后确定该方法返回的值是否可以转换为数值。如果不能，则基于这个返回值再调用toString()方法，再测试返回值。

### 数值转换

Number()函数的规则：（使用于任何类型）

* 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。

parseInt()函数：Number()函数在转换字符串时比较复杂而且不够合理，因此在处理整数的时候更常用的是parseInt()函数：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。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。

注意：parseInt()也能够识别出各种整数格式（即前面讨论的十进制、八进制和十六进制数）：var num3 = parseInt("0xA"); // 10（十六进制数）。但是在使用parseInt()解析像八进制字面量ECMAScript 3 和5 存在分歧在ECMAScript 3 JavaScript 引擎中，"070"被当成八进制字面量，因此转换后的值是十进制的56。而在ECMAScript 5 JavaScript 引擎中，parseInt()已经不具有解析八进制值的能力，因此前导的零会被认为无效，从而将这个值当成"70"。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）: var num = parseInt("0xAF", 16);(其中ox可以省略)建议使用这种做法。

parseFloat()：从第一个字符（位置0）开始解析每个字符。而且也是一直解析到字符串末尾，或者解析到遇见一个无效的浮点数字字符为止。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。

var num1 = parseFloat("3.125e7"); //31250000

var num2 = parseFloat("0xA");

## String类型

由零或多个16 位Unicode 字符组成的字符序列，字符串可以由双引号（"）或单引号（'）表示：

|  |
| --- |
| var firstName = "Nicholas";  var lastName = 'Zakas'; |

### 字符字面量：

转义字符：

|  |
| --- |
| \n 换行 \t 制表 \b 空格 \r 回车 \f 进纸  \\ 斜杠 \' 单引号（'），在用单引号表示的字符串中使用。  \" 双引号（"），在用双引号表示的字符串中使用。  \xnn 以十六进制代码nn表示的一个字符（其中n为0～F）。例如，\x41表示"A"  \unnnn 以十六进制代码nnnn表示的一个Unicode字符（其中n为0～F）。例如，\u03a3表示希腊字符Σ  任何字符串的长度都可以通过访问其length 属性取得：如果字符串中包含双字节字符，那么length 属性可能不会精确地返回字符串中的字符数目。 |

字符串的特点：字符串是不可变的。

转换为字符串：

toString()函数：数值、布尔值、对象和字符串值（没错，每个字符串也都有一个toString()方法，该方法返回字符串的一个副本）都有toString()方法。但null 和undefined 值没有这个方法。在调用数值的toString()方法时，可以传递一个参数：输出数值的基数。默认情况下，toString()方法以十进制格式返回数值的字符串表示。

|  |
| --- |
| var num = 10;  alert(num.toString()); // "10"  alert(num.toString(2)); // "1010"  alert(num.toString(8)); // "12"  alert(num.toString(10)); // "10"  alert(num.toString(16)); // "a" |

String()函数：

* 如果值有toString( )方法，则调用该方法（没有参数）并返回相应的结果；
* 如果值是null，则返回"null"；
* 如果值是undefined，则返回"undefined"。

## Object类型（是一组数据和功能的集合。）

Object 的每个实例都具有下列属性和方法：

* constructor：保存着用于创建当前对象的函数。对于前面的例子而言，构造函数（constructor）就是Object()。
* hasOwnProperty(propertyName)：用于检查给定的属性在当前对象实例中（而不是在实例的原型中）是否存在。其中，作为参数的属性名(propertyName)必须以字符串形式指定（如o.hasOwnProperty("name")）。
* isPrototypeOf(object)：用于检查传入的对象是否是传入对象的原型
* propertyIsEnumerable(propertyName)：用于检查给定的属性是否能够使用for-in 语句来枚举。与hasOwnProperty()方法一样，作为参数的属性名必须以字符串形式指定。
* toLocaleString()：返回对象的字符串表示，该字符串与执行环境的地区对应。toString()：返回对象的字符串表示。
* valueOf()：返回对象的字符串、数值或布尔值表示。通常与toString()方法的返回值相同。

注意：var o = new Object; // 有效，但不推荐省略圆括号

# 操作符

## 一元操作符

### 递增和递减操作符（++ , --）

执行前置递增和递减操作时，变量的值都是在语句被求值以前改变的。（在计算机科学领域，这种情况通常被称作副效应。

var a = 1; ++a;//a的值变成了2.

在应用于一个包含有效数字字符的字符串时，先将其转换为数字值，再执行加减1 的操作。字符串变量变成数值变量。

* 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。
* 在应用于布尔值false 时，先将其转换为0 再执行加减1 的操作。布尔值变量变成数值变量。
* 在应用于布尔值true 时，先将其转换为1 再执行加减1 的操作。布尔值变量变成数值变量。
* 在应用于浮点数值时，执行加减1 的操作。
* 在应用于对象时，先调用对象的valueOf()方法以取得一个可供操作的值。然后对该值应用前述规则。如果结果是NaN，则在调用toString()方法后再应用前述规则。对象变量变成数值变量。

### 一元加和减操作符

在对非数值应用一元操作符时，该操作符会像Number()转型函数一样对这个值执行转换。

## 位操作符

ECMAScript 中的所有数值都以IEEE-754 64 位格式存储，但位操作符并不直接操作64 位的值。而是先将64 位的值转换成32 位的整数，然后执行操作，最后再将结果转换回64 位。

* 按位非（NOT）：由一个波浪线（~）表示，执行按位非的结果就是返回数值的反码。
* 按位与（AND）：由一个和号字符（&）表示
* 按位或（OR）：由一个竖线符号（|）表示
* 按位异或操作符：由一个插入符号（^）表示，也有两个操作数。
* 左移操作符：由两个小于号（<<）表示，这个操作符会将数值的所有位左移动指定的位数。左移不会影响操作数的符号位。
* 有符号的右移操作符：由两个大于号（>>）表示，这个操作符会将数值向右移动，但保留符号位（即正负号标记）.
* 无符号右移操作符：由3 个大于号（>>>）表示，这个操作符会将数值的所有32 位都向右移动。

## 逻辑操作符

### 逻辑非（逻辑非操作符由一个叹号（！）表示，可以应用于任何值。）

* 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。

### 逻辑与（&&有短路原则）：同为true才为真

逻辑与操作可以应用于任何类型的操作数，而不仅仅是布尔值。在有一个操作数不是布尔值的情况下，逻辑与操作就不一定返回布尔值；此时，它遵循下列规则：

* 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。

### 逻辑或（||）：一个为真则为真，短路规则

与逻辑与操作相似，如果有一个操作数不是布尔值，逻辑或也不一定返回布尔值；此时，它遵循下列规则：

*  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；
*  如果两个操作数都是null，则返回null；
*  如果两个操作数都是NaN，则返回NaN；
*  如果两个操作数都是undefined，则返回undefined。

## 乘性操作符

### 乘法(\*)

在处理特殊值的情况下，乘法操作符遵循下列特殊的规则：

* 如果操作数都是数值，执行常规的乘法计算，即两个正数或两个负数相乘的结果还是正数，而如果只有一个操作数有符号，那么结果就是负数。如果乘积超过了ECMAScript 数值的表示范围，则返回Infinity 或-Infinity；
* 如果有一个操作数是NaN，则结果是NaN；
* 如果是Infinity 与0 相乘，则结果是NaN；
* 如果是Infinity 与非0 数值相乘，则结果是Infinity 或-Infinity，取决于有符号操作数的符号；
* 如果是Infinity 与Infinity 相乘，则结果是Infinity；
* 如果有一个操作数不是数值，则在后台调用Number()将其转换为数值，然后再应用上面的规则。

### 除法(/)

除法操作符对特殊的值也有特殊的处理规则。这些规则如下：

* 如果操作数都是数值，执行常规的除法计算，即两个正数或两个负数相除的结果还是正数，而如果只有一个操作数有符号，那么结果就是负数。如果商超过了ECMAScript 数值的表示范围，则返回Infinity 或-Infinity；
* 如果有一个操作数是NaN，则结果是NaN；
* 如果是Infinity 被Infinity 除，则结果是NaN；
* 如果是零被零除，则结果是NaN；
* 如果是非零的有限数被零除，则结果是Infinity 或-Infinity，取决于有符号操作数的符号；
* 如果是Infinity 被任何非零数值除，则结果是Infinity 或-Infinity，取决于有符号操作数的符号；

### 求模(%)

求模操作符会遵循下列特殊规则来处理特殊的值：

*  如果操作数都是数值，执行常规的除法计算，返回除得的余数；
*  如果被除数是无穷大值而除数是有限大的数值，则结果是NaN；
*  如果被除数是有限大的数值而除数是零，则结果是NaN；
*  如果是Infinity 被Infinity 除，则结果是NaN；
*  如果被除数是有限大的数值而除数是无穷大的数值，则结果是被除数；
*  如果被除数是零，则结果是零；
*  如果有一个操作数不是数值，则在后台调用Number()将其转换为数值，然后再应用上面的规则。

## 加性操作符

### 加法(+)

如果两个操作符都是数值，执行常规的加法计算，然后根据下列规则返回结果：

* 如果有一个操作数是NaN，则结果是NaN；
* 如果是Infinity 加Infinity，则结果是Infinity；
* 如果是-Infinity 加-Infinity，则结果是-Infinity；
* 如果是Infinity 加-Infinity，则结果是NaN；
* 如果是+0 加+0，则结果是+0；
* 如果是0 加0，则结果是0；
* 如果是+0 加0，则结果是+0。
* 如果有一个操作数是字符串，那么就要应用如下规则：如果两个操作数都是字符串，则将第二个操作数与第一个操作数拼接起来；如果只有一个操作数是字符串，则将另一个操作数转换为字符串，然后再将两个字符串拼接起来。
* 如果有一个操作数是对象、数值或布尔值，则调用它们的toString()方法取得相应的字符串值，然后再应用前面关于字符串的规则。
* 对于undefined 和null，则分别调用String()函数并取得字符串"undefined"和"null"。

### 减法(-)

与加法操作符类似，ECMAScript 中的减法操作符在处理各种数据类型转换时，同样需要遵循一些特殊规则，如下所示：

* 如果两个操作符都是数值，则执行常规的算术减法操作并返回结果；
* 如果有一个操作数是NaN，则结果是NaN；
* 如果是Infinity 减Infinity，则结果是NaN；
* 如果是-Infinity 减-Infinity，则结果是NaN；
* 如果是Infinity 减-Infinity，则结果是Infinity；
* 如果是-Infinity 减Infinity，则结果是-Infinity；
* 如果是+0 减+0，则结果是+0；
* 如果是+0 减0，则结果是0；
* 如果是0 减0，则结果是+0；
* 如果有一个操作数是字符串、布尔值、null 或undefined，则先在后台调用Number()函数将其转换为数值，然后再根据前面的规则执行减法计算。如果转换的结果是NaN，则减法的结果就是NaN；
* 如果有一个操作数是对象，则调用对象的valueOf()方法以取得表示该对象的数值。如果得到的值是NaN，则减法的结果就是NaN。如果对象没有valueOf()方法，则调用其toString()方法并将得到的字符串转换为数值。

## 关系操作符

<,>,<=,>=

以下就是相应的规则：

* 如果两个操作数都是数值，则执行数值比较。
* 如果两个操作数都是字符串，则比较两个字符串对应的字符编码值。
* 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。
* 如果一个操作数是布尔值，则先将其转换为数值，然后再执行比较。

## 相等操作符

相等和不相等——先转换再比较，全等和不全等——仅比较而不转换。

### 相等和不相等(==,!=)

在转换不同的数据类型时，相等和不相等操作符遵循下列基本规则：

* 如果有一个操作数是布尔值，则在比较相等性之前先将其转换为数值——false 转换为0，而true 转换为1；
* 如果一个操作数是字符串，另一个操作数是数值，在比较相等性之前先将字符串转换为数值；
* 如果一个操作数是对象，另一个操作数不是，则调用对象的valueOf()方法，用得到的基本类型值按照前面的规则进行比较；

两个操作符在进行比较时则要遵循下列规则。

* null 和undefined 是相等的。
* 要比较相等性之前，不能将null 和undefined 转换成其他任何值。
* 如果有一个操作数是NaN，则相等操作符返回false，而不相等操作符返回true。重要提示：即使两个操作数都是NaN，相等操作符也返回false；因为按照规则，NaN 不等于NaN。
* 如果两个操作数都是对象，则比较它们是不是同一个对象。如果两个操作数都指向同一个对象，则相等操作符返回true；否则，返回false。

### 全等和不全等(===,!==)：不会进行类型转换

记住：null == undefined 会返回true，因为它们是类似的值；但null === undefined 会返回false，因为它们是不同类型的值。

## 条件操作符

variable = boolean\_expression ? true\_value : false\_value;

## 赋值操作符

简单的赋值操作符由等于号（=）表示，其作用是把右侧的值赋给左侧的变量

*  乘/赋值（\*=）；
*  除/赋值（/=）；
*  模/赋值（%=）；
*  加/赋值（+=）；
*  减/赋值（=）；
*  左移/赋值（<<=）；
*  有符号右移/赋值（>>=）；
* 无符号右移/赋值（>>>=）。

## 逗号操作符

var num = (5, 1, 4, 8, 0); // num 的值为0

# 控制语句

## if语句

|  |
| --- |
| 其中的condition（条件）可以是任意表达式；而且对这个表达式求值的结果不一定是布尔值。ECMAScript 会自动调用Boolean()转换函数将这个表达式的结果转换为一个布尔值。  if (condition){  statement1  } else {  statement2  } |

## do-while语句

|  |
| --- |
| do {  statement  } while (expression); |

## while语句

while(expression) {statement}

## for语句

|  |
| --- |
| //1-2-4-3的执行顺序，另外循环中定义的变量能够在循环外访问  for (initialization; expression; post-loop-expression){  statement  } |

## for-in语句

for-in 语句是一种精准的迭代语句，可以用来枚举对象的属性。以下是for-in 语句的语法： for (property in expression) { statement }//循环的属性名是不可测的

注意：如果表示要迭代的对象的变量值为null 或undefined，for-in 语句会抛出错误。ECMAScript 5 更正了这一行为；对这种情况不再抛出错误，而只是不执行循环体。为了保证最大限度的兼容性，建议在使用for-in 循环之前，先检测确认该对象的值不是null 或undefined。

## label语句（结合break和continue使用）

## break和continue语句（和java中的相同）

* break：立即退出循环，强制继续执行循环后面的语句
* continue：退出循环的本次执行，继续执行循环。
* label 和break：强制退出label包含的多重循环
* label和continue：退出内部循环，执行label所在的外部循环。

## with语句

with 语句的作用是将代码的作用域设置到一个特定的对象中:

with (expression) statement;

例子：

var qs = location.search.substring(1);

var hostName = location.hostname;

var url = location.href;

等效于：

with(location){

var qs = search.substring(1);

var hostName = hostname;

var url = href;

}//严格模式下不允许使用with语句

## switch语句

|  |
| --- |
| //可以在switch 语句中使用任何数据类型（在很多其他语言中只能使用数值），无论是字符串，还是对象都没有问题。其次，每个case 的值不一定是常量，可以是变量，甚至是表达式。switch 语句在比较值时使用的是全等操作符。  switch (expression) {  case value: statement break;  case value: statement break;  case value: statement break;  case value: statement break;  default: statement  } |

# 函数

基本语法：

|  |
| --- |
| function functionName(arg0, arg1,...,argN) {  statements  } |

严格模式对函数有一些限制：

* 不能把函数命名为eval 或arguments；
* 不能把参数命名为eval 或arguments；
* 不能出现两个命名参数同名的情况。

注意事项：return后面的代码不会执行。return 语句也可以不带有任何返回值。在这种情况下，函数在停止执行后将返回undefined值。

## 理解参数

ECMAScript 中的参数在内部是用一个数组来表示.参数可以是任意类型也可以是任意的个数。，在函数体内可以通过arguments 对象来访问这个参数数组，从而获取传递给函数的每一个参数**（第一个元素是arguments[0]**）。

例如：function sayHi() {

alert("Hello " + arguments[0] + "," + arguments[1]);

}//ECMAScript 函数的重要特点：命名的参数只提供便利，但不是必需的。

另外，通过访问arguments 对象的length 属性可以获知有多少个参数传递了函数，arguments 对象的长度是由调用时传入的参数个数决定。argument的值永远与对应命名参数的值保持同步，但它们读取的内存空间不同。最后一点：没有传递值的命名参数将自动被赋予undefined 值。这就跟定义了变量但又没有初始化一样。

## 没有重载

ECMAScirpt函数没有签名，因为其参数是由包含零或多个值的数组来表示的。当定义了两种相同名字的函数，默认去后面的方式。我们可以利用if判断参数的个数来实现重载。