# 1.0 JavaScript 简介

## 1.1 什么是 JavaScript

- JavaScript 是因特网上最流行的脚本语言
- JavaScript 能够让网页动起来，网页中的 HTML，CSS 变得有思想，有灵魂
- 现在微信小程序的实现就是基于 JavaScript 来开发的
- 在互联网下岗位要求的变化，要求所有开发人员都要具备用户思维，考虑用户交互体验，而这些都是基于 JS 语言来实现的；同时 JS 也是一门基于面向对象的高级语言，能够进一步强化编程思维，所以作为世界上流行度最高的语言之一，作为让微软三次让步给予支持的唯一一门语言，JavaScript 的重要性不言而喻


## 1.2 案例演示

- 带图片旋转动画的手风琴

- 中国地图网点分布


## 1.3 学习 JavaScript 所具备条件

JavaScript web 开发人员必须学习的 3 门语言中的一门

- HTML 定义了网页的内容
- CSS 描述了网页的布局
- JavaScript 网页的行为

# 2.0 JavaScript 入门

## 2.1 了解 JavaScript 代码结构

- 首先在网页中使用<script></script>标签来表示一段 JavaScript 程序
- 同时<script>和</script>会告诉 JavaScript 在何处开始，在何处结束

```
<script>
alert("My First JavaScript");
</script>
```

- 注意：在早期老旧的浏览器中我们会在**<script>标签上加入 type="text/javascript"**。现在我们可以不必这样子做了。因为 JavaScript 是现在浏览器中**主流的脚本语言，也是默认的脚本语言**

## 2.2 第一个 JavaScript 程序

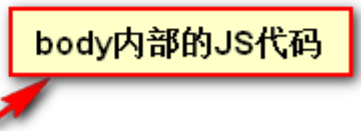

```
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <meta name="Generator" content="EditPlus®">
  <meta name="Author" content="">
  <meta name="Keywords" content="">
  <meta name="Description" content="">
  <title>Document</title>
  <script>
    alert("Hello World");
  </script>
 </head>
 <body>

 </body>
</html>
```

运行浏览器会弹出警告框，alert(...)浏览器内置的函数

## 2.3 JavaScript 在网页何处编写

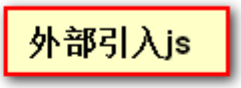
- 第 1 种：在<head></head>标签内部编写一段 JS 程序
- 第 2 种：在<body></body>标签内部编写一段 JS 程序

```html
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <meta name="Generator" content="EditPlus®">
  <meta name="Author" content="">
  <meta name="Keywords" content="">
  <meta name="Description" content="">
  <title>Document</title>
 </head>
 <body>
  <script>
    alert("Hello World");
  </script>
 </body>
</html>
```




- 第 3 种：把 JS 程序保存到外部文件中，其外部文件的扩展名.js，最后通过<script>的 src 属性引入.js 文件。**注意：外部脚本不能包含 <script> 标签**

```html
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <meta name="Generator" content="EditPlus®">
  <meta name="Author" content="">
  <meta name="Keywords" content="">
  <meta name="Description" content="">
  <title>Document</title>
  <script src="03.js"></script>
 </head>
 <body>

 </body>
</html>
```


## 2.4 JavaScript 规范

### 2.4.1 命名规范

- 驼峰式命名法由小(大)写字母开始，后续每个单词首字母都大写
  - **大驼峰式命名法**：首字母大写。StudentInfo、UserInfo、ProductInfo
  - **小驼峰式命名法**：首字母小写。studentInfo、userInfo、productInfo

**命名方法**：小驼峰式命名法。

**命名规范**：前缀应当是名词。(函数的名字前缀为动词，以此区分变量和函数)

**命名建议**：尽量在变量名字中体现所属类型，如:length、count等表示数字类型；而包含name、title表示为字符串类型。

**示例：**

变量命名

```
// 好的命名方式
var maxCount = 10;
var tableTitle = 'LoginTable';

// 不好的命名方式
var setCount = 10;
var getTitle = 'LoginTable';
```

**命名方法**：小驼峰式命名法。

**命名规范**：前缀应当为动词。

**命名建议**：可使用常见动词约定

| 动词 | 含义 | 返回值 |
| --- | --- | --- |
| can | 判断是否可执行某个动作(权限) | 函数返回一个布尔值。true：可执行；false：不可执行 |
| has | 判断是否含有某个值 | 函数返回一个布尔值。true：含有此值；false：不含有此值 |
| is | 判断是否为某个值 | 函数返回一个布尔值。true：为某个值；false：不为某个值 |
| get | 获取某个值 | 函数返回一个非布尔值 |
| set | 设置某个值 | 无返回值、返回是否设置成功或者返回链式对象 |
| load | 加载某些数据 | 无返回值或者返回是否加载完成的结果 |

**示例：**

函数命名

```
// 是否可阅读
function canRead() {
    return true;
}

// 获取名称
function getName() {
    return this.name;
}
```

**命名方法**：名称全部大写。

**命名规范**：使用大写字母和下划线来组合命名，下划线用以分割单词。

**命名建议**：无。

示例：

常量命名

```
var MAX_COUNT = 10;
var URL = 'http://www.baidu.com';
```

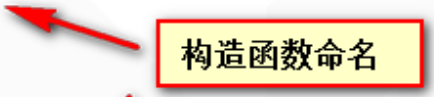
**介绍：**在JS中，构造函数也属于函数的一种，只不过采用new 运算符创建对象。

**命名方法：**大驼峰式命名法，首字母大写。

**命名规范：**前缀为名称。

**命名建议：**无。

**示例：**



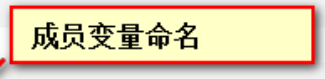

```
function Student(name) {
    this.name = name;
}

var st = new Student('tom');
```

**类的成员包含：**

① 公共属性和方法：跟变量和函数的命名一样。

② 私有属性和方法：前缀为_(下划线)，后面跟公共属性和方法一样的命名方式。

**示例：**




```
function Student(name) {
    var _name = name; // 私有成员

    // 公共方法
    this.getName = function () {
        return _name;
    }

    // 公共方式
    this.setName = function (value) {
        _name = value;
    }
}
var st = new Student('tom');
st.setName('jerry');
console.log(st.getName()); // => jerry: 输出_name私有变量的值
```

## 2.4.2 注释规范

**说明：**单行注释以两个斜线开始，以行尾结束。

**语法：**// 这是单行注释

**使用方式：**

① 单独一行：//(双斜线)与注释文字之间保留一个空格。

② 在代码后面添加注释：//(双斜线)与代码之间保留一个空格，并且//(双斜线)与注释文字之间保留一个空格。

③ 注释代码：//(双斜线)与代码之间保留一个空格。

**示例：**

单行注释规范

```
// 调用了一个函数；1)单独在一行
setTitle();

var maxCount = 10; // 设置最大量；2)在代码后面注释

// setName(); // 3)注释代码
```

**说明：**以/*开头，*/结尾

**语法：**/* 注释说明 */

**使用方法：**

① 若开始(/*)和结束(*/)都在一行，推荐采用单行注释。

② 若至少三行注释时，第一行为/*，最后行为*/，其他行以*开始，并且注释文字与*保留一个空格。

**示例：**

多行注释规范

```
/*
* 代码执行到这里后会调用setTitle()函数
* setTitle(): 设置title的值
*/
setTitle();
```

**说明：**函数(方法)注释也是多行注释的一种，但是包含了特殊的注释要求，参照 javadoc(百度百科)。

**语法：**

```
/**
* 函数说明
* @关键字
*/
```

函数注释规范

**常用注释关键字：**(只列出一部分，并不是全部)

| 注释名 | 语法 | 含义 | 示例 |
| --- | --- | --- | --- |
| @param | @param 参数名 {参数类型} 描述信息 | 描述参数的信息 | @param name {String} 传入名称 |
| @return | @return {返回类型} 描述信息 | 描述返回值的信息 | @return {Boolean} true:可执行;false:不可执行 |
| @author | @author 作者信息 [附属信息：如邮箱、日期] | 描述此函数作者的信息 | @author 张三 2015/07/21 |
| @version | @version XX.XX.XX | 描述此函数的版本号 | @version 1.0.3 |
| @example | @example 示例代码 | 演示函数的使用 | @example setTitle('测试') |

# 3.0 JavaScript 语法基础

## 3.1 JavaScript 变量

### 3.1.1 什么是变量

- 变量是存储信息的容器，简单理解它可以帮助我们存储数字，字符，汉字等等

### 3.1.2 变量语法

- var 变量名 [= 值];
- 动手试试：定义一个变量，将变量中的值输出

```
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <meta name="Generator" content="EditPlus®">
  <meta name="Author" content="">
  <meta name="Keywords" content="">
  <meta name="Description" content="">
  <title>Document</title>
  <script>
    var a = 100; //a变量存储数字100
    var b = "Hello World"; //b变量存储字符串
    console.log("a = " + a);
    console.log("b = " + b); //将变量输出到浏览器控制台中
  </script>
 </head>
 <body>

 </body>
</html>
```

### 3.1.3 变量命名规范

- 变量必须以字母开头
- 变量也能以 $ 和 _ 符号开头（不过我们不推荐这么做）
- 变量名称对大小写敏感（y 和 Y 是不同的变量）

## 3.2 JavaScript 数据类型

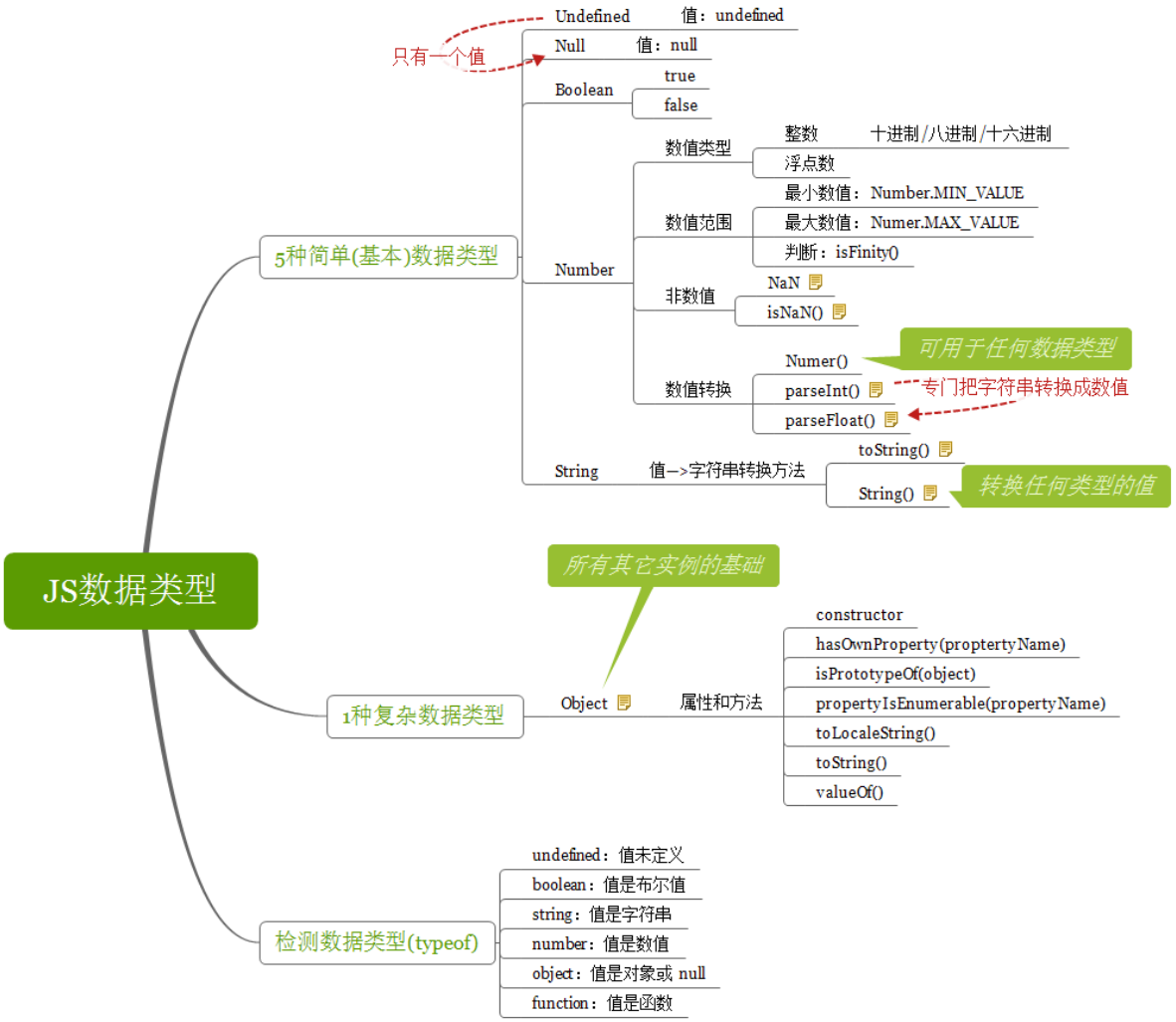
### 3.2.1 什么是数据类型

- JS 定义的变量在存储数据的时候是有一种标准，比如：var a = 100; 则变量 a 存储的值是数字类型；var b = “Hello World”; 则变量 b 存储的值是字符串类型

### 3.2.2 数据类型分类

- 字符串
- 数字
- 布尔
- 数组
- 对象
- Null
- Undefined

## 3.2.3 数据类型图谱




## 3.2.4 undefined 类型

- undefined 中文翻译过来可以理解为“未定义”，**在使用 var 操作符定义一个变量，但是并未给该变量赋值，也就是没有初始化**，此时该变量的值就是 undefined。undefined 类型只有一个值，那就是 undefined

```
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <meta name="Generator" content="EditPlus®">
  <meta name="Author" content="">
  <meta name="Keywords" content="">
  <meta name="Description" content="">
  <title>Document</title>
  <script>
    var a1; // 变量没有赋值初始化
    console.log("a1 = " + a1); //输出结果undefined
  </script>
 </head>
 <body>

 </body>
</html>
```

### 3.2.5 Null 类型

- Null 类型与 Undefined 类型一样，也是只有一个值 null

```
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <meta name="Generator" content="EditPlus®">
  <meta name="Author" content="">
  <meta name="Keywords" content="">
  <meta name="Description" content="">
  <title>Document</title>
  <script>
    var a1; //未初始化，值为undefined
    var a2 = null; //初始化，且值为null
    console.log("a1 = " + a1);
    console.log("a2 = " + a2);
  </script>
 </head>
 <body>

 </body>
</html>
```

## 3.2.6 布尔类型

- Boolean 类型只有两个值，分别是 true 和 false，注意在 javascript 中是严格区分大小写的，**写法不能改变，如果写成 TRUE 或者 True 都不再表示布尔类型**

```
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <meta name="Generator" content="EditPlus®">
  <meta name="Author" content="">
  <meta name="Keywords" content="">
  <meta name="Description" content="">
  <title>Document</title>
  <script>
    var a1 = true;
    var a2 = false;
    console.log("a1 = " + a1);
    console.log("a2 = " + a2);
  </script>
 </head>
 <body>

 </body>
</html>
```

## 3.2.7 Number 类型

- Number 类型来表示**整数和浮点数**，其中字面量格式有三种，分别是十进制整数、八进制整数以及十六进制整数
- 如果某次计算的值超过了 javascript 数值范围，如果是正数，**那么将表示为 Infinity**
- 如果是负数，那么表示为-Infinity
- 如果出现了 Infinity 就不能再进行下一次计算
- 如果要确定一个数值是否处于 javascript 的数值范围期间，可以使用 **isFinite()**函数。在范围期间，返回 true，否则返回 false
- 在 javascript 中有一个特殊的数值 NaN，表示一个本来要返回数值的操作数未返回数值的情况，在 ECMAScript 中，**任何数值除以非数值都会返回 NaN，ECMAScript 中使用 isNaN()函数确定传入的任何类型的参数是否“不是数值”**
- Javascript 中使用三个函数可以将非数值转换数值，三个函数分别是
  - Number()
  - parseInt()
  - parseFloat()

```html
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <meta name="Generator" content="EditPlus®">
  <meta name="Author" content="">
  <meta name="Keywords" content="">
  <meta name="Description" content="">
  <title>Document</title>
  <script>
    var a = 100;
    var b = 0;
    var c = "HelloWorld";
    var d = "100.05";
    console.log("a / b = " + a / b); //返回Infinity
    console.log("isFinite = " + (isFinite(a / b))); //判断数值是否在范围内
    console.log("isNaN = " + (isNaN(c))); //判断是否是数字类型
    console.log("Number = " + Number(d)); //转换为数字类型
    console.log("parseInt = " + parseInt(d)); //转换为整型类型
    console.log("parseFloat = " + parseFloat(d)); //转换为浮点类型
  </script>
 </head>
 <body>

 </body>
</html>
```

## 3.2.8 String 类型

- String 类型由零或者多个 16 位 Unicode 字符组成，是一个字符序列，即字符串
- String 类型的变量在赋值时候，**需要用双引号或者单引号**
- 字符串的长度可以**通过 length 属性取得，注意是 length 属性，而不是 length()方法**，两者不能混淆
- String 数据类型中包含一些特殊的字符字面量，也称之为转义序列

```html
<script>
  var a = "HelloWorld"; //字符串类型
  var b = 'HelloWorld'; //也可以使用单引号方式进行赋值
  console.log("a = " + a);
  console.log("b = " + b);
  console.log("字符串的长度 = " + a.length);
  var c = "\""; //通过反斜线的方式就可以将双引号变成普通的字符
  console.log("c = " + c);
</script>
```

| 字面量 | 含义 |
| --- | --- |
| \n | 换行 |
| \t | 制表 |
| \b | 退格 |
| \r | 回车 |
| \f | 进纸 |
| \\ | 斜杠 |
| \' | 单引号，再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 |

## 3.3 运算符

### 3.3.1 运算符分类

- 算术运算符
- 比较运算符
- 逻辑运算符
- 赋值运算符
- 字符串 + 运算符
- 条件运算符
- typeof 运算符

### 3.3.2 算术运算符

- 给定 y=5，下面的表格解释了这些算术运算符：

| 运算符 | 描述 | 例子 | 结果 |
| --- | --- | --- | --- |
| + | 加 | x=y+2 | x=7 |
| - | 减 | x=y-2 | x=3 |
| * | 乘 | x=y*2 | x=10 |
| / | 除 | x=y/2 | x=2.5 |
| % | 求余数 (保留整数) | x=y%2 | x=1 |
| ++ | 累加 | x=++y | x=6 |
| -- | 递减 | x=--y | x=4 |

```
<script>
    //运算符我们重点给大家介绍
    //求余 %
    //累加 ++
    //1. 如果 ++ 运算符在变量前面，则先做加法，然后得结果
    //2. 如果 ++ 运算符在变量后面，则先使用，然后在做加法运算
    //递减 --
    //除法操作 / ，在JS中除法操作是含有小数部分

    var a = 10;
    var b = 3;
    console.log("a % b = " + a % b);

    var c = ++a; //原来的变量a做一次加1操作，同时变量a值已经变成11
    console.log("c = " + c);
    console.log("a = " + a);

    var d = a++;
    console.log("d = " + d);
    console.log("a = " + a);

    var e = a--;
    console.log("e  = " + e);
    console.log("a = " + a);

    var f = a / b;
    console.log("f = " + f);
</script>
```

## 3.3.3 赋值运算符

- 给定 y=5，下面的表格解释了这些算术运算符：

| 运算符 | 例子 | 等价于 | 结果 |
| --- | --- | --- | --- |
| = | x=y | | x=5 |
| += | x+=y | x=x+y | x=15 |
| -= | x-=y | x=x-y | x=5 |
| *= | x*=y | x=x*y | x=50 |
| /= | x/=y | x=x/y | x=2 |
| %= | x%=y | x=x%y | x=0 |

```
<script>
    var x = 2;
    var y = 5;

    // +=, 相当于 x += y; ==> x = x + y
    console.log("x += y : " + (x += y));

    // -=, 相当于 x -= y; ==> x = x - y
    console.log("x -= y : " + (x -= y));

    // /=, 相当于 x /= y; ==> x = x / y;
    console.log("x /= y : "+ (x /= y));
</script>
```

## 3.3.4 比较运算符

- 比较运算符又称为关系运算符
- 比较运算符的计算结果只有两种值：true 或 false

| 运算符 | 例子 | 结果 |
| --- | --- | --- |
| > | 1 > 2 | false |
| < | 1 < 2 | true |
| >= | 1 >= 2 | false |
| <= | 1 <= 2 | true |
| != | 1 != 2 | true |
| == | 1 == 2 | false |
| === | 1 === 2 | false |

- 注意：== 和 === 之间是有区别的
  - == 比较的两个数的值是否相当
  - === 不仅比较两个数的值，还要比较两个数的类型。所谓的全相等的才能返回 true

```
<script>
    var a = 10;
    var b = 9; // Number类型
    var c = 9; // Number类型
    var d = "9"; //字符串类型

    // 大于 >
    console.log("a > b = " + (a > b));

    // 大于等于 >=
    console.log("a >= b " + (a >= b));

    // 两个等于号 ==
    console.log("b == c " + (b == c));

    // 三个等于号 ===
    // 比较值和数据类型完全相等的时候才返回true
    console.log("b === c " + (b === c));

    console.log("b === d " + (b === d)); // false, 因为b是Number类型, d是字符串类型
    console.log("b == d " + (b == d)); //两个等号它是比较值
</script>
```

## 3.3.5 逻辑运算符

- 逻辑运算符是对两个布尔值类型做运算，且计算后的结果也是一个布尔值

| 运算符 | 例子 | 结果 |
| --- | --- | --- |
| && | true && false | false |
| \|\| | true \|\| false | true |
| ! | !true | false |

```
<script>
    var a = true;
    var b = false;
    var c = true;
    var d = false;

    // &&, 要求布尔值全部为true, 则结果为true
    console.log("a && b = " + (a && b));
    console.log("a && c = " + (a && c));

    // ||, 要求布尔值全部为false, 则结果为false
    console.log("a || b = " + (a || b));
    console.log("b || d = " + (b || d));

    // !, 求布尔值相反的值
    console.log("!a = " + (!a));

    // 先用比较运算 + 逻辑运算
    var e = 10;
    var f = 5;
    var h = 12;
    console.log("e > f && e > h = " + (e > f && e > h));
</script>
```

## 3.3.6 字符串 + 运算符

- 字符串 + 运算符用于把文本值或字符串变量加起来（连接起来）

```
txt1="What a very";
txt2="nice day";
txt3=txt1+txt2;
```

在以上语句执行后，变量 txt3 包含的值是 "What a verynice day"。

```
<script>
  //字符串+，则这个+就是一个连接作用
  var a = 10;
  var b = "Hello";
  var c = 5;
  var d = true;
  var e = false;

  console.log(a + b);

  console.log(a + b + c);

  console.log(b + a + c); //字符串+和算术运算+级别是同等

  console.log(b + a / c); //除法运算符的优先级高于字符串+

  console.log(b + a % c); //求余数运算符优先级高于字符串+

  console.log(b + ++a); //递增运算符优先级高于字符串+

  console.log(b + (a > c)); //比较运算符和字符串+的运算，需要注意运算符优先级

  console.log(b + (d && e));  //逻辑运算符和字符串+的运算，需要注意运算符优先级
</script>
```

## 3.3.7 条件运算符

条件运算符是 ECMAScript 中功能最多的运算符，它的形式与 Java 中的相同。

```
variable = boolean_expression ? true_value : false_value;
```

该表达式主要是根据 *boolean_expression* 的计算结果有条件地为变量赋值。如果 *Boolean_expression* 为 true，就把 *true_value* 赋给变量；如果它是 false，就把 *false_value* 赋给变量。

例如：

```
var iMax = (iNum1 > iNum2) ? iNum1 : iNum2;
```

在这里例子中，iMax 将被赋予数字中的最大值。表达式声明如果 iNum1 大于 iNum2，则把 iNum1 赋予 iMax。但如果表达式为 false（即 iNum2 大于或等于 iNum1），则把 iNum2 赋予 iMax。

```
<script>
    var a = 10;
    var b = 8;
    var d = 18;
    var c = a > b ? a : b;
    console.log("c = " + c);

    //求出三个数字中最大的数是多少
    c = a > b ? (a > d ? a : d) : (b > d ? b : d);
    console.log("c = " + c);

    //求出三个数字中最小的数是多少
    c = a < b ? (a < d ? a : d) : (b < d ? b : d);
    console.log("c = " + c);
</script>
```

### 3.3.8 typeof 运算符

- typeof 主要计算某个变量的数据类型是什么
- 语法：typeof 变量

```
<script>
    var a1 ;
    console.log(a1);//undefined
    console.log(typeof a1);//undefined
    console.log(typeof a2);//undefined
    console.log(a2);//Uncaught ReferenceError: a2 is not defined
</script>
```

```
<script>
    //变量的数据类型
    //Number类型, undefined类型, String类型, Boolean类型
    var a = 10;
    var b = "HelloWorld";
    var c = true;
    var d;
    console.log("typeof(a) = " + typeof(a));
    console.log("typeof(b) = " + typeof(b));
    console.log("typeof(c) = " + typeof(c));
    console.log("typeof(d) = " + typeof(d));
</script>
```