5.JavaScript基础

2021年7月31日　　12:22

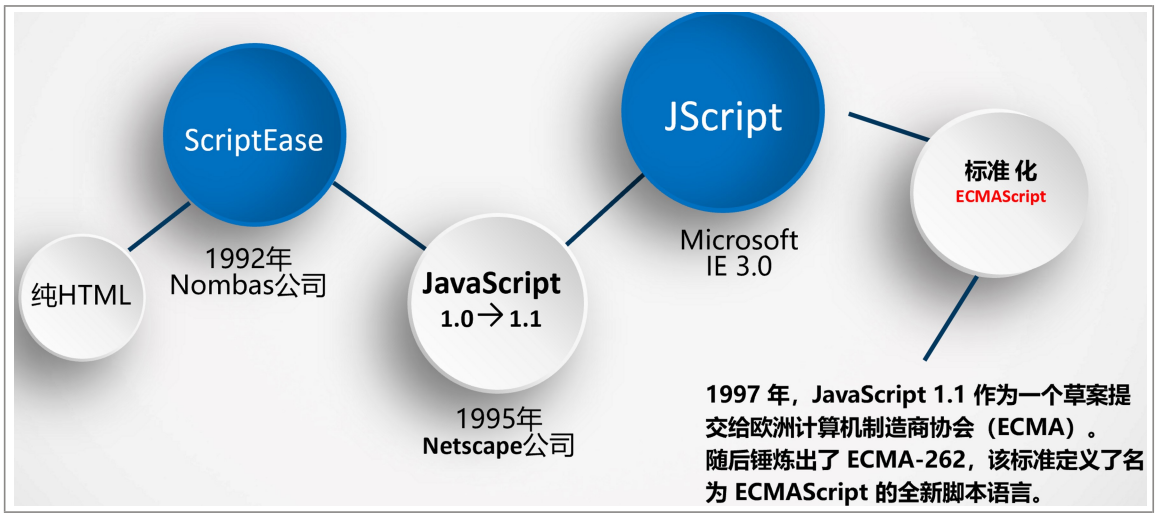
1. JavaScript简介

   JavaScript是一种脚本语言，可以直接嵌入HTML页面之中

   当用户在浏览器中预览该页面时，浏览器会解释并执行其中的JavaScript脚本

   随着HTML 5的出现，JavaScript的重要性更加凸显，在Canvas绘图、本地存储、离线应用和客户端通信等方面都需要结合JavaScript脚本来实现

   i. JS发展史



- 1995.11网景公司发布LiveScript
- 1995.12，Navigator 2版本发布，网景公司(Netspce)和Sun联合发表声明将LiveScript更名为JavaScript，此即为JavaScript1.0. 其功能强大，交互性强，是Web前端语言发展过程中的一个重要里程碑。
- Microsoft进入浏览器市场后，发布的IE3.0中搭载了JScript。此处之外，CEnvi中的ScriptEase也是一种脚本语言。 三种不同的脚本代码带了了很多兼容问题
- 1997年JavaScript1.1提交至ECMA，并由Netspace、sun、微软、Borland等组成委员会制定了ECMAScript程序语言规范书（ECMA-262标准），并被国际标准化组织（ISO）所采纳
- 现在的JavaScript由ECMAScript、DOM、BOM三者组成。JavaScript与JScript在ECMAScript方面是相同的，但在操作浏览器对象DOM、BOM上有各自独特的方法，这也是脚本设计时需要注意的为问题

ii. JS特显及规范

- JS是一种通用的、跨平台、基于对象和事件驱动并具有安全性的客户端脚本语言

  主要特点如下：
  - 解释性
  - 嵌套在HTML中
  - 弱数据类型

    - 跨平台
    - 基于对象
    - 基于事件驱动
- JS代码格式不够严谨，使用比较灵活。但过于随意又不易理解和维护，因此要遵循规范
    - 浏览器解析JavaScript时，会忽略运算符之间多余的空白字符。
    - 每条语句独占一行并以分号结束
    - 代码要有缩进，增强层次感

2. JavaScript使用形式

i. 行内脚本：

- 在HTML中嵌入JavaScript脚本，例如：鼠标事件和超链接等。

```
<h4>校园评选活动</h4>
<dl>
    <dt>
      <img src="img/dress1.jpg" alt="" onclick="alert('你选择了1号选手')">
      <img src="img/dress2.jpg" alt="" onclick="alert('你选择了2号选手')">
      <img src="img/dress3.jpg" alt="" onclick="alert('你选择了3号选手')">
    </dt>
    <dd><a href="javascript:alert('请等待评选结果')">查看评选结果！</a></dd>
</dl>
```

ii. 内部脚本：

- 将这些JavaScript脚本提取出来统一放在<script></script>标签中
- <script>标签位于<head>或<body>标签内

```
<head>
    <script type="text/javascript">
    alert("head中的JavaScript");
    </script>
</head>
```

```
<body>
    <script type="text/javascript">
    alert("body中的JavaScript");
    </script>
</body>
```

iii. 外部JavaScript脚本

- 将JavaScript脚本以单独文件进行存放，实现JavaScript脚本与HTML代码彻底进行分离
- 在HTML页面中通过<script>标签将JS文件进行引入。

```
// test.js文件
alert("外部JavaScript脚本，导入成功。");

<script src="js/6-01useScriptWays.js" type="text/javascript"></script>
```

3. 基本语法

JS脚本语言借鉴了C和Java，有自身的数据类型，运算符、表达式及语法结构。

- 标识符（identifier）用来命名变量、函数或循环中的标签，命名规范如下：
    - 标识第一个字母必须是字母、下划线、或美元符号
    - 标识符区分字母的大小写，推荐使用小写形式或骆驼命名法
    - 标识符由数字、字母、下划线（_）、美元符号（$）构成
    - 标识符不能与JavaScript中的关键字相同
- 关键字

- 关键字（Reserved Words）是指JavaScript中预先定义的、有特别意义的标识符。
- 而保留关键字是指一些关键字在JavaScript中暂时未用到，可能会在后期扩展时使用。
- 关键字或保留关键字都不能用作标识符（包括变量名、函数名等）

abstract	arguments	boolean	break	byte	case	catch
char	class	const	• continue	debugger	default	delete
do	double	else	enum	eval	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	void	volatile	while	with	Yield

- 数据类型
  - 在JavaScript中，变量的类型可以改变，但某一时刻的类型是确定的。
  - 常见的数据类型有String、Boolean、Array、Number和Undefinded等类型。

数据类型	描 述
String	字符串是由双引号（"）或单引号（'）括起来的0~n个字符
Boolean	布尔类型包括true和false两个值
Null	表明某个变量的值为null
Undefined	当声明的变量未初始化时，默认值是undefined
Array	一系列变量或函数的集合，可以存放类型相同的数据，也可以存放类型不同的数据
Number	数值类型可以是32位的整数，也可以是64位的浮点数；而整数可以是十进制、八进制或十六进制等形式
Function	函数是一种特殊的对象数据类型，可以被存储在变量、数组或对象
Object	通过属性和方法定义的对象；常见的对象有String、Date、Math和Array等
String	字符串是由双引号（"）或单引号（'）括起来的0~n个字符

```
<script type="text/javascript">
    var a = '计算机科学与技术';
    var b = 2 == 3;
    var c = null;
    var x = undefined;
    var y = [1, 2, 3, '数组'];
    var y1 = new Array("1", "2", "3", "4");
    var z = 30;
    var plist = document.getElementsByTagName('p');
    plist[1].innerText = '字符串'+a+' 的数据类型为: ' + typeof a;
    plist[2].innerText = '逻辑结果'+b+' 的数据类型为: ' + typeof b;
    plist[3].innerText = c+' 的数据类型为: ' + typeof c;
    plist[4].innerText = x+' 的数据类型为: ' + typeof x;
    plist[5].innerText = '数组和实例化'+y+' 的数据类型为: ' + y1.constructor;
    plist[6].innerText = ''+数字+' 的数据类型为: ' + typeof z;
</script>
```

```
字符串计算机科学与技术 的数据类型为：string

逻辑结果false 的数据类型为：boolean

null 的数据类型为：object

undefined 的数据类型为：undefined

数组和实例化后1,2,3,数组 的数据类型为：function Array() { [native code] }
```

- 出此之外，也有一些教程将转移字符视为一种特殊的数据类型

特殊字符	含义	特殊字符	含义
\b	表示退格	\"	表示双引号本身
\n	表示换页	\'	表示单引号本身
\t	表示Tab符号	\\	表示反斜线
\r	表示回车符	\b	表示退格

- 变量

  变量是程序存储数据的基本单位，用来保存程序中的数据。

  变量名是标识符中的一种，应遵循标识符的命名规范。

变量的定义

- var时定义变量的关键字，多个变量可以一起定义，变量名之间用逗号隔开
- 变量名可以在定义的同时进行赋值，JS中变量也可以不用赋值直接使用

变量的类型

- JavaScript中的变量是**弱数据类型**
- 在声明变量时不需要指明变量的数据类型，通过关键字var进行声明；
- 在变量的使用过程中，变量的类型可以动态改变，类型由所赋值的类型来确定。
- 通过typeof运算符或typeof()函数来获得变量的当前数据类型。

```
var name,age;
var type="student";
var major="软件工程";
school="QST软件学院";
```

```
var x=30;
alert(typeof x);  //弹出提示信息框
x="QST软件学院";
alert(typeof(x));
```

变量的作用域

- 全局变量：全局变量是指定义在函数之外的变量或者未定义直接使用的变量

```
<script type="text/javascript">
    var name="畅游时尚广场";
    //函数的定义
    function test(){
        name=name+" v1.0";
        address="明华镇高新区河东路8888号";
    }
    test();    //函数的调用
    alert("名称："+name+",地址："+address);
```

```
    alert(tel); //此处会报错
    </script>
```

- 局部变量：局部变量是指在函数内部声明变量，仅对当前函数体有效。

```
<script type="text/javascript">
    var name="此处为全局变量的信息"; //定义全局变量
    //函数的定义
    function test(){
        var name="此处为局部变量的信息"; //定义局部变量
        alert(name);  //弹出信息：“此处为局部变量的信息”
    }
    //调用函数
    test();
    alert(name);          //弹出信息：“此处为全局变量的信息”
</script>
```

- 注释

  良好的注释习惯是每一个优秀程序员必备的素质

  - 单行注释：使用“//”符号对单行信息进行注释
  - 多行注释：使用“/* */”符号对多行信息进行注释

4. 运算符

i. 赋值运算符：

运算符	描述	运算符	描述
x+=y,	即对应于x=x+y	x|=y,	即对应于x=x|y
x-=y,	即对应于x=x-y	x^=y,	即对应于x=x^y
x*=y,	即对应于x=x*y	x>>=y,	即对应于x=x>>y
x/=y,	即对应于x=x/y	x<<=y,	即对应于x=x<<y
x%=y,	即对应于x=x%y	x>>>=y	即对应于x=x>>>y
x&=y,	即对应于x=x&y		

- 包括：=、+=、-=、*=、/=、%=、&=、|=、^=、<<=、>>=、>>>=等
- *= 类似这种运算符 会提高运行效率

```
<script type="text/javascript">
    var goodName="护手霜";            //定义变量时进行赋值
    var prodcutAddress;
    prodcutAddress="明华镇";          //定义变量后，进行赋值
    var tmPrice=jdPrice=ddPrice=88; //多变量同时定义，并赋值
    var price=tPrice*0.8;           //将表达式的值赋给变量
</script>
```

ii. 算术运算符：

- 算术运算符用于执行基本的数学运算，包括：

  加（+）、减（-）、乘（*）、除（/）、取余（%）、自加（++）和自减（--）

  - **自加（++）与自减（--）**为单目运算符，++ / -- 在变量前，先执行符号功能(加 或 减)

        - 运算符可以出现在操作数的左侧，也可以出现在操作数的右侧
        - a++是先取a的值，然后对变量a加1
        - ++a则表示先对变量a加1后，再取a的值
    - 自减与自加原理基本一样
    - **+运算符中出现与字符串**运算时：任何数+字符串=字符串

iii. 比较运算符：<、>、<=、>=、==、!=、===（严格等于）、!==（严格不等于）

运算符	描 述
>	大于，左侧的值大于右侧的值时，则返回true；否则返回false
>=	大于等于，左侧的值大于等于右侧的值时，则返回true；否则返回false
<	小于，左侧的值小于右侧的值时，则返回true；否则返回false
<=	小于等于，左侧的值小于等于右侧的值时，则返回true；否则返回false
!=	不等于，左侧与右侧的值不相等时，则返回true；否则返回false
==	等于，左侧与右侧的值相等时，则返回true；否则返回false
!==	严格不等于，左侧与右侧的值不相等或数据类型不同时，返回true；否则返回false
===	严格等于，左侧与右侧的值相等，并且数据类型相同时，返回true；否则返回false

- 比较运算符用于判断两个变量（或常量）的大小，比较的结果是布尔类型
  在JavaScript中，比较运算符包括大于、大于等于、小于、小于等于、不等于、等于和严格等于
- **= =与= = =的区别在于**：
    - = = 支持自动类型转换，只要前后两个变量的值相同就返回true，而忽略数据类型的比较
    - = = = 需要两个变量的值相同并且数据类型一致时才返回true
- **== (!=)和 ===(!==)**，后者不经比较 值 还比较 数据类型 是否相等

iv. 逻辑运算符：&&、||、!

- 逻辑运算符用于对布尔类型的变量（或常量）进行操作；
    - 与（&&）：两个操作数同时为true时，结果为true；否则为false
    - 或（||）：两个操作数中同时为false，结果为false；否则为true
    - 非（！）：只有一个操作数，操作数为true，结果为false；否则结果为true
- 禁止 使用联立不等式 10<x<20 需要使用逻辑运算符 &&

v. 三目判断式：条件表达式？A：B
    - `expression ? value1 : value2;`

```
<script type="text/javascript">
  document.write(11>'11' ? "数字11大于字符'11'" : "数字11不大于字符'11'");
</script>
```

vi. 运算符优先级

运算符从高到低	Java运算符
分隔符	. [] () {} , ;
单目	++ -- ! ~ + -
强转运算符	(typeof) (void) (delete)

算数运算符	% * / + -
位运算符	<< >> >>>
关系运算符	< <= > >= == != ===
逻辑运算符	& ^ | && ||
三目运算符	? :
赋值运算符	= += -= *= /= %= <<= >>= >>>= &= ^= |=

5. 流程控制
   i. 分支结构

   分支结构是指根据条件表达式的成立与否，决定是否执行流程的相应分支结构

   JavaScript中的分支结构有以下两种：if条件语句和switch多分支语句

- if 条件语句

if(condition1){
  statement1; //语句执行块
}
[else if(condition2){
  statement2;
}]
...
[else{
  statement3;
}]
	注意：
	○ 当condition值设为 **0、null、false、undefined或NaN时**，不执行相应的程序分支部分
	○ 当condition值为**true、非空字符串（包括"false"字符串）或非null对象时**，执行相应的程序分支部分

```
<script type="text/javascript">
    var authority=1;
    if(authority==1){
      document.write("您的权限是<b>管理员</b>，欢迎使用漫步时尚广场后台管理系统");
    }else if(authority==2){
      document.write("您的权限是<b>店主</b>，欢迎使用漫步时尚广场后台管理系统");
    }else if(authority==3){
      document.write("您的权限是<b>普通会员</b>，欢迎光临漫步时尚广场");
    }else{
      document.write("<b>游客</b>你好，欢迎光临漫步时尚广场");
    }
</script>
```

- switch语句

  switch语句是由控制表达式和case标签共同构成。其中，控制表达式的数据类型可以是字符串、整型、对象类型等任意类型。

```
switch(expression){
    case value1:        statement1;        break;
    case value2:        statement2;        break;
    ...
    default:        statement;
}
```

```
<script type="text/javascript">
    var authority1 = "管理员"
    document.writeln("<h3>" + authority1 + "的权限如下</h3>");
    switch (authority1) {
```

```
        case "管理员":
            document.writeln("<p>用户管理<br />商品管理<br />"
             +"商品类型管理<br />系统管理<br />"
             +"个人管理<br />退出系统</p>"); break;
        case "店长":
            document.writeln("<p>商品管理<br />商品类型管理<br />"
             +"个人管理<br />退出系统</p>"); break;
        case "普通会员":
            document.writeln("<p>个人管理<br />退出系统</p>"); break;
        default: document.writeln("漫步时代广场"); break;
    }
</script>
```

ii. 循环结构

用于反复执行某段程序代码，直到满足某一条件为止

- while循环

```
while(expression){
     statement;
}
```

while循环又称前测试循环：
是指在执行循环代码之前判断条件是否满足；
当条件满足时执行循环体部分，否则结束循环，

```
<script type="text/javascript">
    document.write("<table border='1' rules='all'>");
    document.write("<tr>");
    document.write("<th>ID</th><th>商品编号</th><th>价格</th>")
    document.write("</tr>");
    var i = 1;
    while (i <= 4) {
        document.write('<tr>');
        document.write('<td>' + i + '</td>');
        document.write('<td>FZ00' + i + '</td>');
        document.write('<td>' + (Math.random() * 100).toFixed(2) + '</td>')
        document.write('</tr>');
        i++;
    }
    document.write("</table>");
</script>
```

ID	商品编号	价格
1	FZ001	57.31
2	FZ002	63.74
3	FZ003	91.93
4	FZ004	83.92

- do …while循环

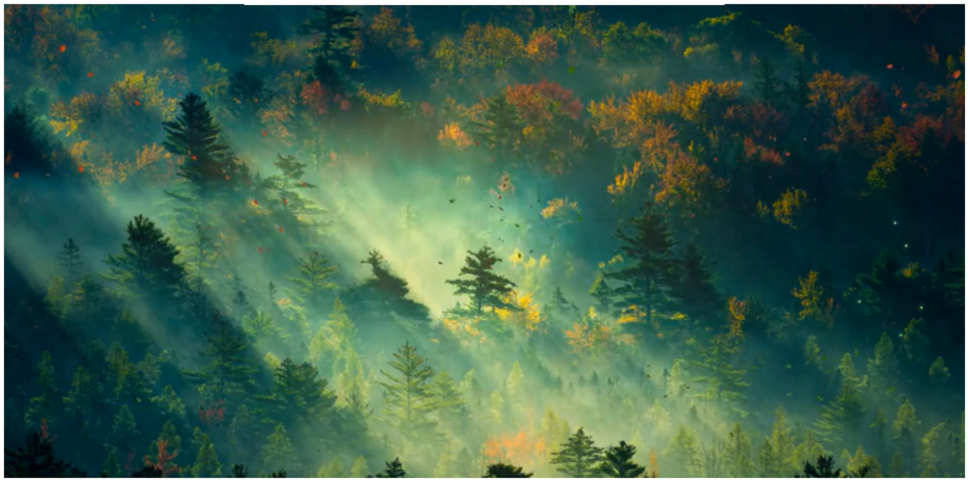
do while循环又称后测试循环，

```
do{
      statement;
}while(expression);
```

与while循环有一定的区别：
while循环是先判断循环条件，符合条件后进行循环
do while是先执行循环体，然后判断条件是满足进入

	下一次循环的条件

```
<script type="text/javascript">
    var imgpre = 'picList';
    var imgrear = '.png';
    var i = 1;
    do {
      var j = 1;
      do {
        var imgname = imgpre + "_r" + i + "_c" + j + imgrear;
        document.write('<img src="img/flowControl/' + imgname + '"/>');
        j++;
      } while (j <= 4);
      // document.write('<br>');
      i++;
    } while (i <= 3);
</script>
```



- for 循环

  for循环也是一种前测试循环，一般用于循环次数已知的情况

```
for(initialization; expression; post-loop-expression){
     statement;
}
```

```
<script type="text/javascript">
    for (let index = 0; index < 7; index++) {
        // 次数注意条件为  j<=index, 实现 下三角
        for (let j = 1; j <= index; j++) {
            document.write("T T T ");
        }
        document.write('<img src="img/flowControl/w' + (index % 9 + 1) + '.png" />
<br/>')
    }
    for (let i = 0; i < 7; i++) {
        for (let k = 6; k > i; k--) {
            document.write('T T T ');
        }
        document.write("<img src='img/flowControl/w" + (i % 9 + 1) + ".png' /><br />");
    }
</script>
```

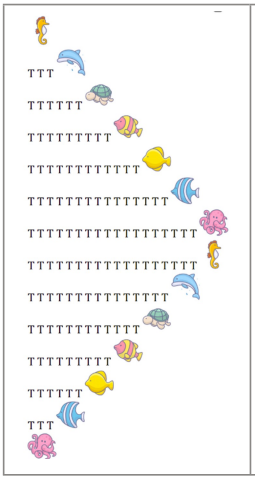
- for in 循环

  是JavaScript提供的一种特殊循环：可以对字符串，数组,对象集合,对象属性等进行遍历

for (property in object){	for in 循环多用于对集合的遍历

statement; }	由于部分浏览器for in数组遍历的支持不够好，可能出知错误。建议尽量不使用forin循环对数组进行遍历。

```
<script type="text/javascript">
    var name ="我从小生活在一个纷争不断的家庭，父母给我的生活造成了很多烦恼"
    document.write('<p>');
    for (const i in name) {
        if (Object.hasOwnProperty.call(name, i)) {
            var size = (i % 7);
            document.write("<font size='" + size + "'>"
              + name.substr(i, 1) + "</font>");
        }
    }
    var family = ['丈夫', '妻子', '父亲', '母亲', '儿子', '女儿', '妹妹'];
    document.write("</p><h4>家庭成员主要包括：</h4><p>");
    for (const t in family) {
        if (Object.hasOwnProperty.call(family, t)) {
            document.write(family[t] + "  ");
        }
    }
    document.write('</p><h4>管理员信息如下：</h4>')
    var person = {
        name: 'CloudWhales',
        authority: '管理员',
        address: '青岛',
        tel: '15824355399'
    };
    document.write("<table><thead><th>信息项</th>"
         +"<th>信息内容</th></thead><tbody>");
    for (const property in person) {
        if (Object.hasOwnProperty.call(person, property)) {
            document.write("<tr><td>" + property + ":</td><td>"
              + person[property] + "</td></tr>");
        }
    }
    document.write("</tbody></table>");
</script>
```



7-5-4 for in 循环

我从小生活在一个纷争不断的家庭，父母给我的生活造成了很多烦恼。

家庭成员主要包括：

丈夫 妻子 父亲 母亲 儿子 女儿 妹妹

管理员信息如下：

信息项	信息内容
name:	CloudWhales
authority:	管理员
address:	青岛
tel:	15824355399

iii. 转移语句

使用转移语句来控制程序的执行方向，包括break语句、continue语句和return语句。

- break语句

在switch结构中，遇到break语句时，就会跳出switch分支结构

在循环结构中，遇到break语句时，立即退出循环，不再执行循环体中的任何代码

- continue语句

  当程序执行过程中遇到continue时，仅仅退出当前本次循环，

  然后判断是否满足继续下一次循环的条件

- return语句

  在执行函数体时，遇到return语句便会退出当前函数，不再执行该函数return语句之后的代码

```
<script>
    for (let index = 1; index < 9; index++) {
        document.write("<img src='img/w" + index + ".png'/>")
    }
    document.write("<br/>")
    for (let i = 1; i < 9; i++) {
        if (i % 3 == 0) {
            document.write("<img src='img/w.png'>")
            continue;
        }
        if (i % 8 == 0) {
            break;
        }
        document.write("<img src='img/w" + i + ".png' />")
    }
    document.write("<br/><br/>")
    function count() {
        var sum = 0;
        for (let i = 100; i < 1000; i++) {
            sum += i;
            if (sum < 10000) {
                document.write('已统计到: ' + i+'        ');
                continue;
            }
            if (i > 200) {
                break;
            }
            return sum;
        }
        document.write('不会被执行的代码');
    }
    document.write('<br/>统计结果为:' + count());
</script>
```

iv. 补充说明：with 语句

- with语句虽然能够精简代码，但其运行效率相对缓慢，一般情况下要避免是同with语句

6. 函数

i. 预定义函数

- 函数是一组延迟动作集的定义，可以通过事件触发或在其他脚本中进行调用
- 在JavaScript中，函数可分为预定义函数和用户自定义函数。
- 预定义函数是指在JavaScript引擎中预先定义的内建函数，用户无需定义便可直接使用。

内建函数	说明
parseInt()	将输入值转换成整数类型输出，第二参数指定转换的进制
parseFloat()	将输入值转换成十进制数，只有一个参数
isNaN()	测试是否为一个数字(Is not a number).返回值为Boolean

isFinite()	检查输入是不是一个既非infinity 也非NaN 的数字
escape()	将字符转换为Unicode码
unescape()	解码由escape函数编码的字符
eval()	将输入字符串(表达式)当作JavaScript代码来执行
alert()	显示一个提醒对话框,包括一个OK按钮
confirm()	显示一个确认对话框，包括OK和Cancel按钮
prompt()	显示一个输入对话框，提示等待用户输入

```
<h4>6-12-1 对话框函数</h4>
请输入收入水平：<input type="text" name="" id="" value="">
<hr>
<input type="button" value="提交" onclick="predefined()">
<h4>6-12-2 数字转换函数</h4>
<script src="./js/6-12functionPredefined.js"></script>
<h4>6-12-3 字符串转数字函数</h4>
<script type="text/javascript">
    document.write("parseInt('88.9')的执行结果：" 
        + parseInt('88.9') + "<br/>");
    document.write("parseInt('8T9')的执行结果：" 
        + parseInt('8T9') + "<br/>");
    document.write("parseInt('B89')的执行结果：" 
        + parseInt('B89') + "<br/>");
    document.write("parseInt('0x10')的执行结果：" 
        + parseInt('0x10') + "<br/>");
    document.write("parseInt('12',16)的执行结果：" 
        + parseInt('12', 16) + "<br/>");
    var str = "3000+499*2";
    document.write("表达式" + str + "的结果是：" + eval(str));

    document.write(+"023");
    document.write("<br/>" + ("0xFF" - 0));
    document.write("<br/>" + "1.23" * 1);
</script>
```

```
function predefined() {
    do {
        // var money = prompt('请输入你的存款余额');
        var money = document.getElementsByTagName('input')[0].value;
        // var money = parseFloat(money);
        if (isNaN(money)) {
            alert('数据输入不合法,请重新输入！');
            break;
        }
        switch (parseInt(money / 10000)) {
            case 0:
                if (money < 0) {
                    alert('难以解决温饱问题')
                }
                else {
                    alert('贫下中农,达到低保生活');
                }
                break;
```

```
            case 1: case 2: case 3:  case 4:
            case 5: case 6: case 7:  case 8:
            case 9:
                alert('富农，衣食无忧坚守一线');
                break;
            default:
                alert('容易获得');
                break;
        }
    } while (false);
}
```

ii. 自定义函数

在自定义函数时既不需要声明函数的参数类型，也不需要声明函数的返回类型

JavaScript目前支持的自定义方式有命名函数、匿名函数、对象函数和自调用函数

- **命名函数：**

函数是由函数定义和函数调用两部分组成。在使用函数时，应先定义函数，然后再进行调用。

  - 完成函数的定义后，函数并不会自动执行，只有通过事件或脚本调用时才会执行。
  - 在同一个<script></script>标签中，函数的调用可以在函数定义之前，也可以在函数定义之后。
  - 在不同的<script></script>标签中时，函数的定义必须在函数的调用之前，否则调用无效。

```
function funcName (param) {
    statements;
    [return expression]
}
```

```
function count(data){
    var sum=0;
    for (var i in data) {
        // if (Object.hasOwnProperty.call(data, data[i])) {
            sum+= parseInt(data[i]);
        // }
    }
    return sum;
}
```

- **匿名函数：**

匿名函数，是网页前端设计者经常使用的一种函数形式，通过表达式的形式来定义一个函数。

  - 由于没有函数名字，所以需要使用变量对匿名函数进行接收，方便后面函数的调用。
  - 匿名函数的定义格式与命名函数基本相同，只是没有提供函数的名称，且在函数结束位置以分号结束。

  - 命名函数对初学者来说，上手容，但可读性相对较差。
  - 匿名函数使用相对更加方便、可读性更好，当前比较流行的」avaScript框架
  - （如Prototype.js、jQuery等）基本上都是采用匿名函数的方式来定义函数的。

```
function (param) {
    statements;
    [return expression]
}
```

```
var anonymous=function(name,i){
  // alert('欢迎'+name+'来到漫步时代广场');
  document.getElementsByTagName('p')[i].innerText=
```

```
      '欢迎'+name+'来到漫步时代广场';
}
var test=anonymous;
anonymous('admin',0);
test('admin',1);
```

- **对象函数：**

  JavaScript还提供了Function类，用于定义函数

  Function是用来定义函数的关键字，首字母必须大写

```
var funcName=new Function([param],statements)
```

```
var showinfo=new Function(
    'name','age','authority','address',
    "alert('数据处理中...'); "+
    "return ('姓名:'+name+',年龄:'+age
      +',权限:'+authority+',地址:'+address);"
);
// alert(showinfo('chuanming','27','管理员','上海'));
document.getElementsByTagName('p')[2].innerText
=showinfo('chuanming','27','管理员','上海');
```

- **自调函数**

  函数本身不会自动执行，只有调用时才会被执行。

  在JavaScript中，提供了一种自调用函数，将函数的定义与调用一并实现

```
(function ([param]) {
    statements;
    [return expression]
})([params]);
```

```
var userName='chuanming';
(function(userData){
    document.getElementsByTagName('p')[3].innerText='欢迎: '
      +userData+'来到漫步时代广场!';
    // alert('欢迎: '+userData+'来到漫步时代广场!');
})(userName);
```