
JScript

Table of Contents

1. JScript.....	13
1.1 JScript 用户指南.....	13
1.1.1 JScript 基础.....	13
1.1.1.1 什么是 JScript?	13
1.1.1.2 编写 JScript 代码.....	14
1.1.1.3 JScript 的变量.....	18
1.1.1.4 JScript 的数据类型.....	20
1.1.1.5 JScript 的运算符.....	24
1.1.1.6 控制程序的流程.....	27
1.1.1.7 JScript 函数.....	35
1.1.1.8 JScript 对象.....	39
1.1.1.8.1 创建自己的对象	42
1.1.1.8.2 内部对象.....	46
1.1.1.9 JScript 保留关键字.....	49
1.1.2 高级 JScript.....	50
1.1.2.1 创建高级对象	50
1.1.2.2 递归.....	53
1.1.2.3 变量范围	54
1.1.2.4 复制、传递和比较数据	57
1.1.2.5 使用数组	59
1.1.2.6 特殊字符	60
1.1.2.7 脚本问题解答	61
1.1.2.8 条件编译	62
1.1.2.9 条件编译变量	63
1.1.3 在浏览器中显示信息	63
1.1.3.1 使用消息框	66
1.2 语言参考	67
1.2.1 特性信息	89
1.2.2 Microsoft JScript 特性 - ECMA	90
1.2.3 Microsoft JScript 特性 - 非-ECMA.....	93
1.2.4 JScript 字母顺序的关键字列表.....	93
1.2.5 JScript 错误.....	101
1.2.5.1 JScript 运行时错误.....	101
1.2.5.2 JScript 语法错误.....	102
1.2.6 JScript 函数.....	103
1.2.6.1 GetObject 函数.....	104
1.2.6.2 ScriptEngine 函数.....	106
1.2.6.3 ScriptEngineBuildVersion 函数.....	107
1.2.6.4 ScriptEngineMajorVersion 函数	107
1.2.6.5 ScriptEngineMinorVersion 函数	108
1.2.7 JScript 方法.....	109

1.2.7.1 A-E.....	114
1.2.7.1.1 abs 方法.....	114
1.2.7.1.2 acos 方法.....	115
1.2.7.1.3 anchor 方法.....	115
1.2.7.1.4 apply 方法.....	116
1.2.7.1.5 asin 方法.....	116
1.2.7.1.6 atan 方法.....	117
1.2.7.1.7 atan2 方法.....	117
1.2.7.1.8 atEnd 方法.....	118
1.2.7.1.9 big 方法.....	119
1.2.7.1.10 blink 方法.....	120
1.2.7.1.11 bold 方法.....	121
1.2.7.1.12 call 方法.....	121
1.2.7.1.13 ceil 方法.....	122
1.2.7.1.14 charAt 方法.....	122
1.2.7.1.15 charCodeAt 方法.....	123
1.2.7.1.16 compile 方法.....	124
1.2.7.1.17 concat 方法 (Array).....	126
1.2.7.1.18 concat 方法 (String).....	127
1.2.7.1.19 cos 方法.....	128
1.2.7.1.20 decodeURI 方法.....	129
1.2.7.1.21 decodeURIComponent 方法.....	129
1.2.7.1.22 dimensions 方法.....	130
1.2.7.1.23 encodeURI 方法.....	132
1.2.7.1.24 encodeURIComponent 方法.....	132
1.2.7.1.25 escape 方法.....	133
1.2.7.1.26 eval 方法.....	133
1.2.7.1.27 exec 方法.....	134
1.2.7.1.28 exp 方法.....	136
1.2.7.2 F-I.....	136
1.2.7.2.1 fixed 方法.....	136
1.2.7.2.2 floor 方法.....	137
1.2.7.2.3 fontcolor 方法.....	137
1.2.7.2.4 fontsize 方法.....	138
1.2.7.2.5 fromCharCode 方法.....	139
1.2.7.2.6 getDate 方法.....	139
1.2.7.2.7 getDay 方法.....	140
1.2.7.2.8 getFullYear 方法.....	141
1.2.7.2.9 getHours 方法.....	142
1.2.7.2.10 getItem 方法.....	143
1.2.7.2.11 getMilliseconds 方法.....	146
1.2.7.2.12 getMinutes 方法.....	147
1.2.7.2.13 getMonth 方法.....	148
1.2.7.2.14 getSeconds 方法.....	149

1.2.7.2.15 getTime 方法	150
1.2.7.2.16 getTimezoneOffset 方法	151
1.2.7.2.17 getUTCDate 方法	152
1.2.7.2.18 getUTCDay 方法	153
1.2.7.2.19 getUTCFullYear 方法	154
1.2.7.2.20 getUTCHours 方法	155
1.2.7.2.21 getUTCMilliseconds 方法	156
1.2.7.2.22 getUTCMinutes 方法	157
1.2.7.2.23 getUTCMonth 方法	158
1.2.7.2.24 getUTCSeconds 方法	159
1.2.7.2.25 getVarDate 方法	160
1.2.7.2.26 getYear 方法	161
1.2.7.2.27 indexOf 方法	162
1.2.7.2.28 isFinite 方法	163
1.2.7.2.29 isNaN 方法	163
1.2.7.2.30 italics 方法	164
1.2.7.2.31 item 方法	164
1.2.7.3 J-R	166
1.2.7.3.1 join 方法	166
1.2.7.3.2 lastIndexOf 方法	167
1.2.7.3.3 lbound 方法	168
1.2.7.3.4 link 方法	170
1.2.7.3.5 localeCompare 方法	171
1.2.7.3.6 log 方法	172
1.2.7.3.7 match 方法	172
1.2.7.3.8 max 方法	174
1.2.7.3.9 min 方法	174
1.2.7.3.10 moveFirst 方法	175
1.2.7.3.11 moveNext 方法	177
1.2.7.3.12 parse 方法	178
1.2.7.3.13 parseFloat 方法	180
1.2.7.3.14 parseInt 方法	181
1.2.7.3.15 pop 方法	182
1.2.7.3.16 pow 方法	182
1.2.7.3.17 push 方法	183
1.2.7.3.18 random 方法	183
1.2.7.3.19 replace 方法	184
1.2.7.3.20 reverse 方法	186
1.2.7.3.21 round 方法	187
1.2.7.4 S	188
1.2.7.4.1 search 方法	188
1.2.7.4.2 setDate 方法	189
1.2.7.4.3 setFullYear 方法	190
1.2.7.4.4 setHours 方法	191

1.2.7.4.5 setMilliseconds 方法	193
1.2.7.4.6 setMinutes 方法	194
1.2.7.4.7 setMonth 方法	195
1.2.7.4.8 setSeconds 方法	197
1.2.7.4.9 setTime 方法	198
1.2.7.4.10 setUTCDate 方法	199
1.2.7.4.11 setUTCFullYear 方法	200
1.2.7.4.12 setUTCHours 方法	202
1.2.7.4.13 setUTCMilliseconds 方法	203
1.2.7.4.14 setUTCMinutes 方法	205
1.2.7.4.15 setUTCMonth 方法	206
1.2.7.4.16 setUTCSeconds 方法	207
1.2.7.4.17 setYear 方法	209
1.2.7.4.18 shift 方法	210
1.2.7.4.19 sin 方法	210
1.2.7.4.20 slice 方法 (Array)	210
1.2.7.4.21 slice 方法 (String)	211
1.2.7.4.22 small 方法	212
1.2.7.4.23 sort 方法	213
1.2.7.4.24 splice 方法	214
1.2.7.4.25 split 方法	215
1.2.7.4.26 sqrt 方法	216
1.2.7.4.27 strike 方法	216
1.2.7.4.28 sub 方法	217
1.2.7.4.29 substr 方法	218
1.2.7.4.30 substring 方法	219
1.2.7.4.31 sup 方法	220
1.2.7.5 T-Z	221
1.2.7.5.1 tan 方法	221
1.2.7.5.2 test 方法	221
1.2.7.5.3 toArray 方法	222
1.2.7.5.4 toDateString 方法	225
1.2.7.5.5 toExponential 方法	225
1.2.7.5.6 toFixed 方法	226
1.2.7.5.7 toGMTString 方法	227
1.2.7.5.8 toLocaleDateString 方法	227
1.2.7.5.9 toLocaleLowerCase 方法	228
1.2.7.5.10 toLocaleString 方法	228
1.2.7.5.11 toLocaleTimeString 方法	229
1.2.7.5.12 toLocaleUpperCase 方法	230
1.2.7.5.13 toLowerCase 方法	230
1.2.7.5.14 toPrecision 方法	231
1.2.7.5.15 toString 方法	231
1.2.7.5.16 toTimeString 方法	235

1.2.7.5.17 toUpperCase 方法	235
1.2.7.5.18 toUTCString 方法	236
1.2.7.5.19 ubound 方法	237
1.2.7.5.20 unescape 方法	240
1.2.7.5.21 unshift 方法	240
1.2.7.5.22 UTC 方法	241
1.2.7.5.23 valueOf 方法	243
1.2.8 JScript 对象	244
1.2.8.1 ActiveXObject 对象	244
1.2.8.2 Array 对象	246
1.2.8.3 Boolean 对象	248
1.2.8.4 Date 对象	249
1.2.8.5 Enumerator 对象	251
1.2.8.6 Error 对象	253
1.2.8.7 Function 对象	255
1.2.8.8 Global 对象	257
1.2.8.9 Math 对象	257
1.2.8.10 Number 对象	258
1.2.8.11 Object 对象	259
1.2.8.12 RegExp 对象	260
1.2.8.13 正则表达式对象	261
1.2.8.14 String 对象	263
1.2.8.15 VBArray 对象	265
1.2.9 JScript 运算符	267
1.2.9.1 通用信息	269
1.2.9.1.1 运算符优先级	269
1.2.9.1.2 运算符总结	270
1.2.9.2 加法赋值运算符 (+)=	271
1.2.9.3 加法运算符 (+)	272
1.2.9.4 赋值运算符 (=)	273
1.2.9.5 按位“与”赋值运算符 (&=)	273
1.2.9.6 按位“与”运算符 (&)	274
1.2.9.7 按位左移运算符 (<<)	275
1.2.9.8 按位“非”运算符 (~)	276
1.2.9.9 按位“或”赋值运算符 (=)	277
1.2.9.10 按位“或”运算符 ()	278
1.2.9.11 按位右移运算符 (>>)	279
1.2.9.12 按位“异或”赋值运算符 (^=)	279
1.2.9.13 按位“异或”运算符 (^)	280
1.2.9.14 逗号运算符 (,)	281
1.2.9.15 比较运算符	282
1.2.9.16 复合赋值运算符	283
1.2.9.17 条件（三目）运算符 (?:)	284
1.2.9.18 delete 运算符	285

1.2.9.19 除法赋值运算符 (/=)	285
1.2.9.20 除法运算符 (/)	286
1.2.9.21 in 运算符	287
1.2.9.22 递增 (++) 和递减 (--) 运算符	287
1.2.9.23 instanceof 运算符	289
1.2.9.24 左移赋值运算符 (<<=)	290
1.2.9.25 逻辑“与”运算符 (&&)	291
1.2.9.26 逻辑“非”运算符 (!)	292
1.2.9.27 逻辑“或”运算符 ()	293
1.2.9.28 取余赋值运算符 (%=)	294
1.2.9.29 取余运算符 (%)	295
1.2.9.30 乘法赋值运算符 (*=)	295
1.2.9.31 乘法运算符 (*)	296
1.2.9.32 new 运算符	296
1.2.9.33 右移赋值运算符 (>>=)	297
1.2.9.34 减法赋值运算符 (-=)	298
1.2.9.35 减法运算符 (-)	299
1.2.9.36 typeof 运算符	300
1.2.9.37 无符号右移运算符 (>>>)	300
1.2.9.38 无符号右移赋值操作 (>>>=)	301
1.2.9.39 void 运算符	302
1.2.10 JScript 属性	302
1.2.10.1 \$1..\$9 属性	304
1.2.10.2 arguments 属性	305
1.2.10.3 caller 属性	306
1.2.10.4 constructor 属性	307
1.2.10.5 description 属性	308
1.2.10.6 E 属性	309
1.2.10.7 global 属性	309
1.2.10.8 hasOwnProperty 方法	311
1.2.10.9 ignoreCase 属性	312
1.2.10.10 index 属性	314
1.2.10.11 Infinity 属性	316
1.2.10.12 input 属性(\$_)	316
1.2.10.13 isPrototypeOf 方法	317
1.2.10.14 lastIndex 属性	318
1.2.10.15 length 属性 (Array)	319
1.2.10.16 length 属性 (Function)	321
1.2.10.17 length 属性 (String)	322
1.2.10.18 LN10 属性	322
1.2.10.19 LN2 属性	323
1.2.10.20 LOG10E 属性	323
1.2.10.21 LOG2E 属性	323
1.2.10.22 MAX_VALUE 属性	324

1.2.10.23 message 属性	324
1.2.10.24 MIN_VALUE 属性	325
1.2.10.25 multiline 属性	325
1.2.10.26 name 属性	328
1.2.10.27 NaN 属性	329
1.2.10.28 NaN 属性 (Global)	330
1.2.10.29 NEGATIVE_INFINITY 属性	330
1.2.10.30 number 属性	330
1.2.10.31 PI 属性	332
1.2.10.32 POSITIVE_INFINITY 属性	332
1.2.10.33 propertyIsEnumerable 属性	332
1.2.10.34 prototype 属性	333
1.2.10.35 source 属性	334
1.2.10.36 SQRT1_2 属性	335
1.2.10.37 SQRT2 属性	336
1.2.10.38 undefined 属性	336
1.2.11 JScript 语句	337
1.2.11.1 @cc_on 语句	338
1.2.11.2 @if 语句	338
1.2.11.3 @set 语句	339
1.2.11.4 break 语句	341
1.2.11.5 Comment 语句	342
1.2.11.6 continue 语句	343
1.2.11.7 do...while 语句	344
1.2.11.8 for 语句	346
1.2.11.9 for...in 语句	347
1.2.11.10 function 语句	349
1.2.11.11 if...else 语句	350
1.2.11.12 Labeled 语句	351
1.2.11.13 return 语句	353
1.2.11.14 switch 语句	354
1.2.11.15 this 语句	356
1.2.11.16 throw 语句	357
1.2.11.17 try...catch...finally 语句	358
1.2.11.18 var 语句	361
1.2.11.19 while 语句	362
1.2.11.20 with 语句	363
1.3 FileSystemObject 用户指南	364
1.3.1 FileSystemObject 对象模型	364
1.3.2 FileSystemObject 和 Scripting 运行时库参考的介绍	364
1.3.3 FileSystemObject 对象	365
1.3.4 设计 FileSystemObject	366
1.3.5 处理驱动器和文件夹	370
1.3.6 处理文件	376

1.3.7 FileSystemObject 示例代码	386
1.4 Scripting 运行时库参考	414
1.4.1 脚本运行时方法	424
1.4.1.1 Add 方法 (Dictionary).....	426
1.4.1.2 Add 方法 (Folders).....	427
1.4.1.3 BuildPath 方法.....	428
1.4.1.4 Close 方法	429
1.4.1.5 Copy 方法	430
1.4.1.6 CopyFile 方法	431
1.4.1.7 CopyFolder 方法.....	433
1.4.1.8 CreateFolder 方法	434
1.4.1.9 CreateTextFile 方法	435
1.4.1.10 Delete 方法.....	436
1.4.1.11 DeleteFile 方法.....	437
1.4.1.12 DeleteFolder 方法	439
1.4.1.13 DriveExists 方法.....	440
1.4.1.14 Exists 方法	441
1.4.1.15 FileExists 方法.....	442
1.4.1.16 FolderExists 方法.....	443
1.4.1.17 GetAbsolutePathName 方法	444
1.4.1.18 GetBaseName 方法	445
1.4.1.19 GetDrive 方法.....	447
1.4.1.20 GetDriveName 方法	448
1.4.1.21 GetExtensionName 方法.....	449
1.4.1.22 GetFile 方法.....	450
1.4.1.23 GetFileName 方法	452
1.4.1.24 getFileVersion 方法	453
1.4.1.25 GetFolder 方法.....	454
1.4.1.26 GetParentFolderName 方法.....	455
1.4.1.27 GetSpecialFolder 方法.....	456
1.4.1.28 GetTempName 方法.....	458
1.4.1.29 Items 方法	459
1.4.1.30 Keys 方法.....	461
1.4.1.31 Move 方法.....	462
1.4.1.32 MoveFile 方法	462
1.4.1.33 MoveFolder 方法.....	464
1.4.1.34 OpenAsTextStream 方法	465
1.4.1.35 OpenTextFile 方法	467
1.4.1.36 Read 方法	469
1.4.1.37 ReadAll 方法.....	470
1.4.1.38 ReadLine 方法	471
1.4.1.39 Remove 方法.....	473
1.4.1.40 RemoveAll 方法.....	474
1.4.1.41 Skip 方法	474

1.4.1.42 SkipLine 方法.....	476
1.4.1.43 Write 方法.....	477
1.4.1.44 WriteBlankLines 方法.....	478
1.4.1.45 WriteLine 方法.....	480
1.4.2 脚本运行时对象.....	481
1.4.2.1 Dictionary 对象.....	481
1.4.2.2 Drive 对象.....	482
1.4.2.3 Drives 集合.....	483
1.4.2.4 File 对象.....	484
1.4.2.5 Files 集合.....	485
1.4.2.6 FileSystemObject 对象.....	486
1.4.2.7 Folder 对象.....	487
1.4.2.8 Folders 集合.....	488
1.4.2.9 TextStream 对象.....	490
1.4.3 脚本运行时属性.....	491
1.4.3.1 AtEndOfLine 属性.....	492
1.4.3.2 AtEndOfStream 属性.....	493
1.4.3.3 Attributes 属性.....	494
1.4.3.4 AvailableSpace 属性.....	496
1.4.3.5 Column 属性.....	497
1.4.3.6 CompareMode 属性.....	498
1.4.3.7 Count 属性.....	500
1.4.3.8 DateCreated 属性.....	501
1.4.3.9 DateLastAccessed 属性.....	502
1.4.3.10 DateLastModified 属性.....	503
1.4.3.11 Drive 属性.....	504
1.4.3.12 DriveLetter 属性.....	505
1.4.3.13 Drives 属性.....	506
1.4.3.14 DriveType 属性.....	507
1.4.3.15 Files 属性.....	509
1.4.3.16 FileSystem 属性.....	510
1.4.3.17 FreeSpace 属性.....	511
1.4.3.18 IsReady 属性.....	512
1.4.3.19 IsRootFolder 属性.....	513
1.4.3.20 Item 属性.....	515
1.4.3.21 Key 属性.....	516
1.4.3.22 Line 属性.....	517
1.4.3.23 Name 属性.....	518
1.4.3.24 ParentFolder 属性.....	519
1.4.3.25 Path 属性.....	520
1.4.3.26 RootFolder 属性.....	522
1.4.3.27 SerialNumber 属性.....	523
1.4.3.28 ShareName 属性.....	524
1.4.3.29 ShortName 属性.....	525

1.4.3.30 ShortPath 属性	526
1.4.3.31 Size 属性	527
1.4.3.32 SubFolders 属性	528
1.4.3.33 TotalSize 属性	529
1.4.3.34 Type 属性	530
1.4.3.35 VolumeName 属性	531
1.5 正则表达式简介	533
1.5.1 正则表达式	533
1.5.2 早期起源	534
1.5.3 使用正则表达式	534
1.5.4 正则表达式语法	535
1.5.5 建立正则表达式	537
1.5.6 优先权顺序	538
1.5.7 普通字符	538
1.5.8 特殊字符	539
1.5.9 非打印字符	539
1.5.10 字符匹配	540
1.5.11 限定符	542
1.5.12 定位符	544
1.5.13 选择和编组	545
1.5.14 后向引用	547
2. Addenda	549
2.1 版权所有	549
2.2 必须给数组长度指定一个有穷正整数	550
2.3 数组长度必须是一个有穷正整数	550
2.4 需要 Array 或 arguments 对象	551
2.5 需要布尔变量	551
2.6 不能赋值给函数结果	551
2.7 不能赋值给 'this'	552
2.8 在循环外不能有 'break'	553
2.9 在循环外不能有 'continue'	553
2.10 条件编译已关闭	554
2.11 需要 Date 对象	554
2.12 一条 'switch' 语句中只能有一个 'default'	555
2.13 需要 Enumerator 对象	555
2.14 异常抛出, 但无法抓住	556
2.15 缺少 '('	556
2.16 缺少 ')'	556
2.17 正则表达式中缺少 ')'	557
2.18 缺少 '{'	557
2.19 缺少 '}'	558
2.20 缺少 '/'	558
2.21 缺少 ':'	558
2.22 缺少 ';'	559

2.23 缺少 '@'	559
2.24 缺少 '}'	559
2.25 缺少 '='	560
2.26 缺少 'catch'	560
2.27 需要常量	560
2.28 缺少 '@end'	561
2.29 需要十六进制数	561
2.30 需要标识符	562
2.31 需要标识符、字符串或者数字	562
2.32 正则表达式中缺少 ')'	562
2.33 缺少 'while'	563
2.34 函数没有合法的 prototype 对象	563
2.35 需要 Function 对象	564
2.36 非法赋值	564
2.37 非法字符	564
2.38 字符集范围非法	565
2.39 需要 JScript 对象	565
2.40 未找到标签	566
2.41 标签定义重复	566
2.42 需要 Number 类型	566
2.43 需要 Object 类型	567
2.44 需要对象的成员	567
2.45 需要 Regular Expression 对象	567
2.46 函数外有 'return' 语句	568
2.47 需要 String 对象	568
2.48 语法错误	568
2.49 正则表达式语法错误	569
2.50 小数部分的位数越界	569
2.51 精度越界	569
2.52 待解码 URI 的编码非法	570
2.53 待编码的 URI 包含有非法字符	570
2.54 在 Throw 的同一行内必须后跟一个表达式	571
2.55 未定义标识符	571
2.56 未预期的限定符	572
2.57 注释未结束	572
2.58 字符串常量未结束	573
2.59 需要 VBArray	573
2.60 Microsoft Scripting 运行时库特性	574
2.61 JScript 语言参考	575
2.62 Arguments 对象	576
2.63 Script 运行时库参考	577
2.64 版本信息	577
2.65 0...n 属性	585
2.66 callee 属性	586

2.67 lastMatch 属性 (\$&).....	587
2.68 lastParen 属性 (\$+).....	589
2.69 leftContext 属性 (\$`).....	590
2.70 length 属性 (arguments)	592
2.71 rightContext 属性 (\$').....	593

1. JScript

[Jscript 用户指南](#)(See 1.1)

[在 Internet 浏览器中使用 JScript](#)(See 1.1.3)

[JScript 语言参考](#)(See 2.61)

[FlieSystemObject 基础](#)(See 1.3)

[Script 运行时参考](#)(See 2.63)

1.1 JScript 用户指南

[JScript 基础](#)(See 1.1.1)

[高级 JScript](#)(See 1.1.2)

1.1.1 Jscript 基础

[什么是 Jscript?](#) (See 1.1.1.1)

[编写 JScript 代码](#)(See 1.1.1.2)

[JScript 变量](#)(See 1.1.1.3)

[JScript 数据类型](#)(See 1.1.1.4)

[JScript 运算符](#)(See 1.1.1.5)

[运算符优先级](#)(See 1.2.9.1.1)

[控制程序流](#)(See 1.1.1.6)

[条件编译](#)(See 1.1.2.8)

[条件编译变量](#)(See 1.1.2.9)

[JScript 函数](#)(See 1.1.1.7)

[JScript 对象](#)(See 1.1.1.8)

[固有对象](#)(See 1.1.1.8.2)

[创建自己的对象](#)(See 1.1.1.8.1)

[JScript 保留字](#)(See 1.1.1.9)

1.1.1.1 什么是 JScript?

JScript 是 Microsoft 公司对 ECMA 262 语言规范 (ECMAScript 编辑器 3) 的一种实现。除了少数例外 (为了保持向后兼容), JScript 完全实现了 ECMA 标准。本概述的目的就是引导您学习使用 JScript。

使用 JScript

JScript 是一种解释型的、基于对象的脚本语言。尽管与 C++ 这样成熟的面向对

象的语言相比，JScript 的功能要弱一些，但对于它的预期用途而言，JScript 的功能已经足够大了。

JScript 不是其他语言的精简版（例如，它只是与 Java 有点模糊而间接的关系），也不是任何事物的简化。不过，它有其局限性。例如，您不能使用该语言来编写独立运行的应用程序，并且没有对读写文件的内置支持。此外，JScript 脚本只能在某个解释器或“宿主”上运行，如 Active Server Pages (ASP)、Internet 浏览器或者 Windows 脚本宿主。

JScript 是一种宽松类型的语言。宽松类型意味着您不必显式定义变量的数据类型。事实上 JScript 更进一步。您无法在 JScript 上明确地定义数据类型。此外，在大多数情况下，JScript 将根据需要自动进行转换。例如，如果将一个数值添加到由文本组成的某项（一个字符串），该数值将被转换为文本。

本用户指南的其余部分是 JScript 特性概述。有关该语言实现的全部细节，请参考[语言参考](#)(See 2.61)。

注意 下面大多数示例的代码比实际 Web 页中的代码应该更明确，并且不是太复杂。其目的是阐明相关概念，而不是提供最优的简短编码和风格。在任何情况下，如果六个月以后您还能毫不费力地阅读和理解所编写的代码，则说明这些代码写得不错。

1.1.1.2 编写 JScript 代码

与其他许多编程语言一样，Microsoft JScript 是用文本方式编写的，并被组织成为语句、由相关的语句集组成的块、以及注释。在一条语句内可以使用[变量](#)、比如字符串和数字（称为“文字”）的立即数、以及表达式。

语句

JScript 程序是语句的集合。一条 Jscript 语句相当于英语中的一个完整句。Jscript 语句将表达式组合起来，完成一个任务。

一条语句由一个或多个表达式、关键字或者运算符（符号）组成。典型地，一条语句写一行，尽管一条语句可以超过两行或更多行。两条或更多条语句也可以写在同一行上，语句之间用分号“;”隔开。通常，每一新行开始一条新语句。不过显式地终止语句是一个好方法。这是用分号 (;)来实现的，分号是 JScript 语句的终止字符。下面给出 Jscript 语句的两隔示例。

```
aBird = "Robin"; // 将文本"Robin"赋值给变量 aBird
```

```
var today = new Date(); // 将今天的日期赋值给变量
```

```
today
```

用大括号（{}）括起来的一组 JScript 语句称为一个语句块。分组到一个语句块中的语句通常可当作单条语句处理。这就是说在 JScript 期望有一条单个语句的大多数地方可以使用语句块。应该注意以 **for** 和 **while** 打头的循环语句是例外情况。注意，语句块中的原始语句以分号结束，但语句块本身并不以分号结束。

通常，在函数和条件语句中使用语句块。注意，Jscript 与 C++ 以及其他某些语言

不同，它不认为语句块是一个新的范围；只有函数创建新范围。在下面的示例中，第一条语句开始定义一个函数，该函数包含一个五条语句组成的语句块。语句块后的三条语句没有用大括号括起来；这些语句不是一个语句块，所以就不是函数定义的一部分。

```
function convert(inches)  {  
    feet = inches / 12;    // 这五条语句属于一个语句块。  
    miles = feet / 5280;  
    nauticalMiles = feet / 6080;  
    cm = inches * 2.54;  
    meters = inches / 39.37;  
}  
  
km = meters / 1000;    // 这三条语句不在语句块内。  
  
kradius = km;  
mradius = miles;
```

注释

单行的 JScript 注释以一对正斜杠(//)开始。下面给出一个单行注释的示例。

```
aGoodIdea    =    "Comment    your    code  
thoroughly.";    //    这是一个单行注释。
```

多行注释以一个正斜杠加一个星号的组合(/*)开始,并以其逆向顺序 (*/)结束。

```
/*
```

这是一个用来解释前面的代码语句的多行注释。

该语句将一个值赋给 aGoodIdea 变量。

用引号包含的这种值称为一个文字。

文字显式并直接包含信息；

而不是间接地引用信息。

(引号不属于该文字的内容。)

```
* /
```

注意 如果您试图将一个多行注释插入到另一个中，JScript 不能按正常的方式解释生成的多行注释。标明嵌入的多行注释结束的 `*/` 被认为是整个多行注释的结尾。这就意味着嵌入多行注释之后的文本不再被认为是注释；相应地，它将被解释为 JScript 代码，并会产生语法错误。

建议将所有的注释写为单行注释的语句块。这样您以后就能够将大段的代码与多行注释区分开。

`//` 这是另一种多行注释，写成一系列单行注释。

`//` 在执行完该语句后，可以使用 `aGoodIdea` 变量的名字来引用其内容，

`//` 如下一条语句那样，即采用连接操作将字符串文字添加到

`//` `aGoodIdea` 变量，以创建一个新的变量。

```
var extendedIdea = aGoodIdea + " You
never know when you'll have to figure
out what it does.";
```

赋值和相等

JScript 语句中使用等号 (`=`) 给变量赋值：等号是赋值运算符。`=` 运算符左边的操作项总是一个 Lvalue。Lvalue 可以是：

- 变量，
- 数组元素，
- 对象属性。

`=` 运算符右边的操作项总是一个 Rvalue。Rvalues 可以是任何类型的一个任意值，包括表达式的值。下面给出一个 JScript 赋值语句的示例。

```
anInteger = 3;
```

Jscript 编译器解释本语句的意义为：“将 3 赋给变量 `anInteger`”或“`anInteger` 的值为 3”。

确定您理解了 `=` 运算符（赋值）和 `==` 运算符（相等）的差异。在比较两个值是否相等时，应使用两个等于号 (`==`)。这些内容将在 [控制程序的流程](#) (See 1.1.1.6) 中详细介绍。

表达式

JScript 表达式是指 JScript 解释器能够计算生成值的 JScript “短语”。这个值可

以是任何有效的 JScript 类型 — 数字、字符串、对象，等等。最简单的表达式是文字。下面给出 JScript 文字表达式的一些示例。

```
3.9 // 数字文字
"Hello!" // 字符串文字
false // 布尔文字
null // 文字空值
{x:1, y:2} // 对象文字
[1,2,3] // 数组文字
function(x){return x*x;} // 函数文字
```

更多复杂的表达式中包含变量、函数、函数调用以及其他表达式。可以用运算符将表达式组合，创建复合表达式。运算符可以是：

```
+ // 加法
- // 减法
* // 乘法
/ // 除法
```

下面给出 JScript 复合表达式的一些示例。

```
var anExpression = 3 * (4 / 5) + 6;
var aSecondExpression = Math.PI * radius
* radius;
var          aThirdExpression          =
aSecondExpression + "%" + anExpression;
var  aFourthExpression  =  "("  +
aSecondExpression  +  ")"  %  "("  +
anExpression +  ")";
```

1.1.1.3 JScript 的变量

任何编程语言中，用一块数据量化一个概念。

How old am I?

在 Jscript 中，变量是给概念的名称；它代表了给出瞬间的值。当使用该变量时，实际是用的它所代表的的数据。给出示例：

```
NumberOfDaysLeft = EndDate - TodaysDate;
```

机械的理解是使用变量来存储、得到并操作脚本中出现的所有的不同值。创建有意义的变量名称；便于别人理解脚本。

变量声明

变量在脚本中的第一次出现是在声明中。变量在第一次用到时就设置于内存中，便于后来在脚本中引用。使用变量之前先进行声明。可以使用 **var** 关键字来进行变量声明。

```
var count; // 单个声明。
```

```
var count, amount, level; // 用单个 var 关键字声明的多个声明。
```

```
var count = 0, amount = 100; // 一条语句中的变量声明和初始化。
```

如果在 **var** 语句中没有初始化变量，变量自动取 JScript 值 **undefined**。尽管并不安全，但声明语句中忽略 **var** 关键字是合法的 JScript 语法。这时，JScript 解释器给予变量全局范围的可见度。当在过程级中声明一个变量时，它不能用于全局范围；这种情况下，变量声明必须用 **var** 关键字。

变量命名

变量名称是一个标识符。Jscript 中，用标识符来：

- 命名变量，
- 命名函数，
- 给出循环的标签。

JScript 是一种区分大小写的语言。因此变量名称 *myCounter* 和变量名称 *mYCounter* 是不一样的。变量的名称可以是任意长度。创建合法的变量名称应遵循如下规则：

- 第一个字符必须是一个 ASCII 字母（大小写均可），或一个下划线(_)。注意第一个字符不能是数字。
- 后续的字符必须是字母、数字或下划线。

- 变量名称一定不能是 [保留字](#)(See 1.1.1.9)。

下面给出合法变量名称的一些示例：

```
_pagecount
```

```
Part9
```

```
Number_Items
```

下面给出无效变量名称的一些示例：

```
99Balloons // 不能以数字开头。
```

```
Smith&Wesson // “与”符号（&）字符用于变量名称是无效的。
```

当要声明一个变量并进行初始化，但又不想指定任何特殊值，可以赋值为 JScript 值 [null](#)。下面给出示例。

```
var bestAge = null;
```

```
var muchTooOld = 3 * bestAge; //
```

`muchTooOld` 的值为 0。

如果声明了一个变量但没有对其赋值，该变量存在，其值为 Jscript 值 [undefined](#)。下面给出示例。

```
var currentCount;
```

```
var finalCount = 1 * currentCount; //
```

`finalCount` 的值为 NaN，因为 `currentCount` 为 `undefined`。

注意在 JScript 中 **null** 和 **undefined** 的主要区别是 **null** 的操作象数字 0，而 **undefined** 的操作象特殊值 NaN（不是一个数字）。对 **null** 值和 **undefined** 值作比较总是相等的。

可以不用 **var** 关键字声明变量，并赋值。这就是隐式声明。

```
noStringAtAll = ""; // 隐式声明变量
```

```
noStringAtAll。
```

不能使用未经过声明的变量。

```
var volume = length * width; // 错误 -
```

`length` 和 `width` 并不存在。

强制转换

表达式中操作项的数据类型相同时 JScript 解释器才能对其求值。如果表达式不经过强制转换就试图对两个不同的数据类型（如一个为数字，另一个为字符串）执行运算，将产生错误结果。但在 Jscript 中情况就不同了。

JScript 是一种自由类型的语言。它的变量没有预定类型（相对于强类型语言，如 C++）。相反，JScript 变量的类型相应于他们包含的值的类型。这种操作的好处是能将值作为另一类型处理。

在 Jscript 中，可以对不同类型的值执行运算，不必担心 JScript 解释器产生异常。相反，JScript 解释器自动将数据类型之一改变（强制转换）为另一种数据类型，然后执行运算。例如：

运算	结果
数值与字符串相加	将数值强制转换为字符串。
布尔值与字符串相加	将布尔值强制转换为字符串。
数值与布尔值相加	将布尔值强制转换为数值。

考虑下面的示例。

```
var x = 2000;           // 一个数字。

var y = "Hello";        // 一个字符串。

x = x + y;               // 将数字强制转换为字符串。

document.write(x);      // 输出 2000Hello。
```

要想显式地将字符串转换为整数，使用 [parseInt 方法](#)(See 1.2.7.3.14)。要想显式地将字符串转换为数字，使用 [parseFloat 方法](#)(See 1.2.7.3.13)。请注意，比较大小时字符串自动转换为相等的数字，但加法（连接）运算时保留为字符串。

1.1.1.4 JScript 的数据类型

Jscript 有三种主要数据类型、两种复合数据类型和两种特殊数据类型。主要（基本）数据类型是：

- 字符串
- 数值
- 布尔

复合（引用）数据类型是：

- 对象
- 数组

特殊数据类型是：

- Null
- Undefined

字符串数据类型

一个字符串值是排在一起的一串零或零以上的 Unicode 字符（字母、数字和标点符号）。字符串数据类型用来表示 JScript 中的文本。脚本中可以包含字符串文字，这些字符串文字放在一对匹配的的单引号或双引号中。字符串中可以包含双引号，该双引号两边需加单引号，也可以包含单引号，该单引号两边需加双引号。下面是字符串的示例：

```
"Happy am I; from care I'm free!"
```

```
'"Avast, ye lubbers!" roared the technician.'
```

```
"42"
```

```
'c'
```

请注意，JScript 中没有表示单个字符的类型（如 C++ 的 **char**）。要表示 Jscript 中的单个字符，应创建一个只包含一个字符的字符串。包含零个字符（""）的字符串是空（零长度）字符串。

数值数据类型

在 Jscript 中整数和浮点值没有差别；JScript 数值可以是其中任何一种（JScript 内部将所有的数值表示为浮点值）。

整型值

整型值可以是正整数，负整数和 0。可以用 10 进制，8 进制和 16 进制来表示。在 Jscript 中大多数字是用十进制表示的。加前缀“0”表示 8 进制的整型值，只能包含 0 到 7 的数字。前缀为“0”同时包含数字“8”或“9”的数被解释为十进制数。

加前缀“0x”（零和 x|X）表示 16 进制整型值。可以包含数字 0 到 9，以及字母 A 到 F（大写或小写）。使用字母 A 到 F 表示十进制 10 到 15 的单个数字。就是说 0xF 与 15 相等，同时 0x10 等于 16。

八进制和十六进制数可以为负，但不能有小数位，同时不能以科学计数法（指数）表示。

浮点值

浮点值为带小数部分的数。也可以用科学计数法来表示。这就是说，大写或小写“e”用来表示 10 的次方。Jscript 用数值表示的八字节 IEEE754 浮点标准。这意味着数字最大可以到 $\pm 1.7976931348623157 \times 10^{308}$ ，最小到 $\pm 5 \times 10^{-324}$ 。以“0”开始

且包含小数点的数字被解释为小数浮点数。

注意以“0x”或“00”开始并包含小数点的数将发生错误。以下是 Jscript 中数字的例子。

数字	描述	等价十进制数
.0001, 0.0001, 1e-4, 1.0e-4	四个相等的浮点数。	0.0001
3.45e2	浮点数。	345
42	整数。	42
0378	整数。虽然看起来是八进制数（以 0 开头），但是 8 不是有效的八进制数字，所以为十进制数。	378
0377	八进制整数。注意它虽然看起来比上面的数只小 1，但实际数值有很大不同。	255
0.0001	浮点数。虽然以零开头，但由于带有小数点所以不是八进制数。	0.0001
00.0001	错误。两个零开头表示为八进制，但八进制数不能带有小数部分。	N/A （编译错误）
0xff	十六进制整数。	255
0x37CF	十六进制整数。	14287
0x3e7	十六进制整数。注意 ‘e’ 并不被认为指数。	999
0x3.45e2	错误。十六进制数不能有小数部分。	N/A （编译错误）

另外，JScript 包含特殊值数字。它们是：

- NaN （不是数）。当对不适当的数据进行数学运算时使用，例如字符串或未定义值。
- 正无穷大。在 JScript 中如果一个正数太大的话使用它来表示。
- 负无穷大。在 JScript 中如果一个负数太大的话使用它来表示。
- 正 0 和负 0。Jscript 区分正 0 和负 0。

Boolean 数据类型

尽管字符串和数字类型可以有无数不同的值，boolean 数据类型却只有两个值。它们是文字 **true** 和 **false**。Boolean 值是一个真值，它表示一个状态的有效性（说明该状态为真或假）。

脚本中的比较通常得到一个 Boolean 结果。考虑下一行 Jscript 代码。

```
y = (x == 2000);
```

这里要比较变量 x 的值是否与数字 2000 相等。如果相等，比较的结果为 Boolean 值 **true**，并将其赋给变量 y。如果 x 与 2000 不等，则比较的结果为 boolean 值 **false**。

Boolean 值在结构控制中尤其有用。可以将直接创建 boolean 值的比较与使用该 boolean 值的语句相组合。考虑下面的 JScript 代码范例。

```

if (x == 2000)

    z = z + 1;

else

    x = x + 1;

```

当 **boolean** 值为 **true** 时，Jscript 中的 **if/else** 语句执行一个操作（这样， $z = z + 1$ ），而当 **boolean** 值为 **false** 时执行另一个操作（ $x = x + 1$ ）。

可以使用任意表达式作比较表达式。任何值为 0、**null**、未定义或空字符串的表达式被解释为 **false**。其他任意值的表达式解释为 **true**。例如，可以使用如下表达式：

```

if (x = y + z) // 这可能不是想要的结果 — 如下！

```

注意上面的代码并不检查 x 是否与 $y+z$ 相等，因为仅使用了一个等号（赋值）。相反的，上面的代码将 $y+z$ 赋给变量 x ，然后检查整个表达式的值是否为零。要检查 x 是否与 $y+z$ 相等，使用如下代码。

```

if (x == y + z) // 这与上面的代码不同！

```

有关比较的详细信息，请参见[控制程序的流程](#)(See 1.1.1.6)。

Null 数据类型

在 Jscript 中数据类型 **null** 只有一个值：**null**。关键字 **null** 不能用作函数或变量的名称。

包含 **null** 的变量包含“无值”或“无对象”。换句话说，该变量没有保存有效的数、字符串、**boolean**、数组或对象。可以通过给一个变量赋 **null** 值来清除变量的内容。

请注意，在 Jscript 中，**null** 与 0 不相等（与在 C 和 C++ 中不同）。同时应该指出的是，Jscript 中 **typeof** 运算符将报告 **null** 值为 **Object** 类型，而非类型 **null**。这点潜在的混淆是为了向下兼容。

Undefined 数据类型

如下情况使返回 **undefined** 值：

- 对象属性不存在，
- 声明了变量但从未赋值。

注意不能通过与 **undefined** 做比较来测试一个变量是否存在，虽然可以检查它的类型是否为“**undefined**”。在以下的代码范例中，假设程序员想测试是否已经声明变量 x ：

```

// 这种方法不起作用

```

```

if (x == undefined)

```

```
// 作某些操作
```

```
// 这个方法同样不起作用 — 必须检查
```

```
// 字符串 "undefined"
```

```
if (typeof(x) == undefined)
```

```
// 作某些操作
```

```
// 这个方法有效
```

```
if (typeof(x) == "undefined")
```

```
// 作某些操作
```

考虑将 `undefined` 值与 `null` 做比较。

```
someObject.prop == null;
```

如下情况时，比较的结果为 **true**，

- 如果属性 `someObject.prop` 包含 `null` 值，
- 如果属性 `someObject.prop` 不存在。
- 要检查一个对象属性是否存在，可以使用新的 `in` 运算符：

```
if ("prop" in someObject)
```

```
// someObject 有属性 'prop'
```

1.1.1.5 JScript 的运算符

JScript 具有全范围的运算符,包括算术、逻辑、位、赋值以及其他某些运算符。

计算	逻辑	位运算	赋值	杂项					
描述	符号	描述	符号	描述	符号	描述	符号	描述	符号
负 值 (See 1.2.9.35)	-	逻辑非 (See 1.2.9.26)	!	按位取反 (See 1.2.9.8)	~	赋 值 (See 1.2.9.4)	=	删 除 (See 1.2.9.18)	delete
递 增 (See	++	小 于 (See	<	按位左移 (See	<<	运算赋值 (See	oP=	typeof 运算符 (See	typeof

1.2.9.22)		1.2.9.15)		1.2.9.7)		1.2.9.16)		1.2.9.36)	
递减 (See 1.2.9.22)	--	大于 (See 1.2.9.15)	>	按位右移 (See 1.2.9.11)	>>			void (See 1.2.9.39)	void
乘法 (See 1.2.9.31)	*	小于等于 (See 1.2.9.15)	<=	无符号右移 (See 1.2.9.37)	>>>			instanceof (See 1.2.9.23)	instanceof
除法 (See 1.2.9.20)	/	大于等于 (See 1.2.9.15)	>=	按位与 (See 1.2.9.6)	&			new (See 1.2.9.32)	new
取模运算 (See 1.2.9.29)	%	等于 (See 1.2.9.15)	==	按位异或 (See 1.2.9.13)	^			in (See 1.2.9.21)	in
加法 (See 1.2.9.3)	+	不等于 (See 1.2.9.15)	!=	按位或 (See 1.2.9.10)					
减法 (See 1.2.9.35)	-	逻辑与 (See 1.2.9.25)	&&						
		逻辑或 (See 1.2.9.27)							
		条件(三元运算符) (See 1.2.9.17)	?:						
		逗号 (See 1.2.9.14)	,						
		严格相等 (See 1.2.9.15)	===						
		非严格相等 (See 1.2.9.15)	!==						

==（相等）与 ===（严格相等）的区别在于恒等运算符在比较前强制转换不同类型的值。例如，恒等对字符串 "1" 与数值 1 的比较结果将为 true。而严格相等不强制转换不同类型的值，因此它认为字符串 "1" 与数值 1 不相同。基本的字符串、数值和布尔值是按值比较的。如果它们的值相同，比较结果为相

等。对象（包括 **Array**、**Function**、**String**、**Number**、**Boolean**、**Error**、**Date** 以及 **RegExp** 对象）按引用比较。即使这些类型的两个变量具有相同的值，只有在它们正好为同一对象时比较结果才为 `true`。

例如：

```
// 具有相同值的两个基本字符串。
```

```
var string1 = "Hello";
```

```
var string2 = "Hello";
```

```
// 具有相同值的两个 String 对象。
```

```
var StringObject1 = new String(string1);
```

```
var StringObject2 = new String(string2);
```

```
// 比较结果为 true。
```

```
if (string1 == string2)
```

```
    // 执行某些命令（将要运行的）。
```

```
// 比较结果为 false。
```

```
if (StringObject1 == StringObject2)
```

```
    // 执行某些命令（不会运行）。
```

```
// 要比较 String 对象的值，
```

```
// 用 toString() 或者 valueOf() 方法。
```

```
if      (StringObject1.valueOf()      ==  
StringObject2)
```

```
    // 执行某些命令（将要运行的）。
```

1.1.1.6 控制程序的流程

Jscript 脚本中的语句一般是按照写的顺序来运行的。这种运行称为顺序运行，是程序流的默认方向。

与顺序运行不同，另一种运行将程序流转换到脚本的另外的部分。也就是，不按顺序运行下一条语句，而是运行另外的语句。

要使脚本可用，该控制的转换必须以逻辑方式执行。程序控制的转换是基于一个“决定”，这个“决定”结果是真或假（返回 Boolean 型 **true** 或 **false**）。创建一个表达式，然后测试其是否为真。主要有两种程序结构实现本功能。

第一种是选择结构。用来指明两种程序流方向，在程序中创建一个交叉点（像岔路）。在 Jscript 中有四种选择结构可用。

- 单一选择结构 (**if**)，
- 二路选择结构 (**if/else**)，
- 内联三元运算符 **?:**
- 多路选择结构 (**switch**)。

第二种类型的程序控制结构是循环结构。使用循环结构来指明当某些条件保持为真时要重复的动作。当控制语句的条件得到满足时（通常在某些迭代的特定数字后），控制跳过循环结构传递到下条语句。在 Jscript 中有四种循环结构可用。

- 在循环的开头测试表达式 (**while**)，
- 在循环的末尾测试表达式 (**do/while**)，
- 对对象的每个属性都进行操作 (**for/in**)，
- 由计数器控制的循环 (**for**)。

通过嵌套和堆栈选择、循环控制结构，可以创建相当复杂的脚本。

第三种形式的结构程序流由意外处理给出，本文档不作讨论。

使用条件语句

JScript 支持 **if** 和 [if...else](#)(See 1.2.11.11) 条件语句。在 **if** 语句中将测试一个条件，如果该条件满足测试，执行相关的 JScript 编码。在 **if...else** 语句中，如果条件不满足测试，则将执行不同的代码。最简单的 **if** 语句格式可以在一行中写完，不过更常见的是多行的 **if** 和 **if...else** 语句。

下述示例演示了使用 **if** 和 **if...else** 语句的各种可能的语法。第一个示例演示了最简单的布尔测试。当（且仅当）括号之间的项的值为（或者可被强制转换为）**true** 时，**if** 后续的语句或语句块才会被执行。

```
// smash() 函数是在该代码的其他地方定义的。
```

```
// 布尔测试，看 newShip 是否为 true。
```



```
if (newShip)
    smash(champagneBottle,bow);
```

// 在本示例中，除非两个条件都为真，否则该测试将不会被满足。

```
if (rind.color == "deep yellow " &&
    rind.texture == "large and small
    wrinkles")
{
    theResponse = ("Is it a Crenshaw
    melon?");
}
```

// 在本示例中，只要任何一个条件为真，则测试即会满足。

```
var theReaction = "";
if ((dayOfWeek == "Saturday" ||
    (dayOfWeek == "Sunday"))
{
    theReaction = ("I'm off to the
    beach!");
}
else
{
    theReaction = ("Hi ho, hi ho, it's
```

```
off to work I go!");
}
```

条件运算符

JScript 也支持隐式的条件格式。该格式在要测试的条件后使用一个问号（而不是在条件前的 **if** ）。它也指定两个可选项，一个在满足条件时使用，另一个在条件不满足时使用。这两个选择项之间必须用一个冒号隔开。

```
var hours = "";
```

```
// 下面的代码指定 hours 是包含 theHour 的内容，
```

```
// 还是包含 theHour - 12 的内容。
```

```
hours += (theHour >= 12) ? " PM" : " AM";
```

如果要一起测试多个条件，并且知道某个条件比其他条件更可能满足或不满足测试，可以使用称为“短路计算”的特性来加速脚本的运行速度。当 JScript 计算逻辑表达式时，只计算要得到结果所需的子表达式。

例如，如果有一个“与”表达式，如 `((x == 123) && (y == 42))`，Jscript 首先检查 `x` 是否为 123。如果不是，即使 `y` 等于 42，整个表达式的值也不可能为 **true**。因此，并不对 `y` 作测试，Jscript 返回 **false** 值。

类似地，如果多个条件中只要有一个为真（使用 `||` 运算符），则当任何一个条件满足该测试时测试则停止。如果要测试的条件包括函数调用或其他复合表达式，这种处理方式就有效。出于此种想法，写 OR 表达式时，先写最有可能为 **true** 的条件。写 AND 表达式时，先写最有可能为 **false** 的条件。

以这种方式设计脚本的好处的一个示例是：在下例中如果 `runfirst()` 返回 0 或 **false**，则不会运行 `runsecond()`。

```
if ((runfirst() == 0) || (runsecond() ==
0)) {
    // 若干代码。
}
```

使用循环

有多种方式来重复执行一条语句或语句块。通常重复执行被称为*循环*或*重复*。重复只是循环的一个运行。典型情况是用一个变量测试来进行控制，每执行一次循环变量的取值都会更改。JScript 支持四种循环：[for](#)(See 1.2.11.8) 循环、

[for...in](#)(See 1.2.11.9) 循环、 [while](#)(See 1.2.11.19) 循环、 [do...while](#)(See 1.2.11.7) 循环。

使用 for 循环

for 语句指定了一个计数器变量，一个测试条件，以及更新该计数器的操作。在每次循环的重复之前，都将测试该条件。如果测试成功，将运行循环中的代码。如果测试不成功，不运行循环中的代码，程序继续运行紧跟在循环后的第一行代码。在执行该循环后，计算机变量将在下一次循环之前被更新。

如果循环条件永不会满足，则不执行该循环。如果测试条件始终满足，则将导致无限循环。在有些情况下，前者可能是合乎需要的，而后者几乎没有用处，因此在编写循环条件时一定要特别注意。

```
/*
```

更新表达式（下例中的 `"icount++"`）将在循环结束时被执行，即在构成循环主体的语句块被执行后，在测试条件之前。

```
*/
```

```
var howFar = 10; // 将循环次数限制为 10。
```

```
var sum = new Array(howFar); // 创建一个称为
sum 并具有 10 个成员的数组，这 10 个成员从 0 到 9。
```

```
var theSum = 0;
```

```
sum[0] = 0;
```

```
for(var icount = 0; icount < howFar;
icount++) { // 在本例中将从 0 到 9 进行计数。
```

```
theSum += icount;
```

```
sum[icount] = theSum;
```

```
}
```

```
var newSum = 0;
```

```
for(var icount = 0; icount > howFar;
icount++) { // 该循环根本不会被执行，因为
icount 不大于 howFar。
newSum += icount;
}
```

```
var sum = 0;
for(var icount = 0; icount >= 0;
icount++) { // 这是一个无限循环。
sum += icount;
}
```

使用 for...in 循环

JScript 提供了一种特别的循环方式来遍历一个[对象](#)(See 1.1.1.8)的所有用户定义的属性或者一个数组的所有元素。**for...in** 循环中的循环计数器是一个字符串，而不是数字。它包含当前属性的名称或者当前数组元素的下标。

下面的代码范例应在 Internet 浏览器中运行，因为它使用 **alert** 方法，该方法不属于 Jscript。

```
// 创建具有某些属性的对象

var myObject = new Object();
myObject.name = "James";
myObject.age = "22";
myObject.phone = "555 1234";

// 枚举（循环）对象的所有属性
for (prop in myObject)
{
```

```
// 显示 "The property 'name' is James",  
等等。
```

```
    window.alert("The property '" + prop  
+ "'" is " + myObject[prop]);  
}
```

尽管 **for...in** 循环看起来像 VBScript 的 **For Each...Next** 循环，其实并不一样。JScript 的 **for...in** 循环重复 Jscript 对象所有的属性。VBScript 的 **For Each...Next** 循环重复集合中的所有项目。要循环 JScript 中的所有集合，需要用 **Enumerator** 对象。尽管某些对象（像 Internet 浏览器中的那些）支持 VBScript 的 **For Each...Next** 和 Jscript 的 **for...in** 循环，但多数对象并不都支持。

使用 while 循环

while 循环相似于 **for** 循环。其不同之处是 **while** 循环没有内置的计数器或更新表达式。如果希望控制语句或语句块的循环执行，需要不只是“运行该代码 n 次”，而是更复杂的规则，用 **while** 循环。下面的示例使用 Internet 浏览器对象模型和 **while** 循环来询问用户一个简单的问题。

```
var x = 0;  
  
while ((x != 42) && (x != null))  
{  
    x = window.prompt("What is my  
favourite number?", x);  
}
```

```
if (x == null)  
    window.alert("You gave up!");  
else  
    window.alert("Yep - it's the  
Ultimate Answer!");
```

注意 由于 **while** 循环没有显式的内置计数器变量，因此比其他类型的循环更容易产生无限循环。此外，由于不易发现循环条件是在何时何地更新的，很容易编写一个实际上从不更新条件的 **while** 循环。因此在编写 **while** 循环时应特别小心。

同上面所提到的，在 JScript 中还有 **do...while** 循环与 **while** 循环相似，不同之处在于它总是至少运行一次，因为是在循环的末尾检查条件，而不是在开头。例如，上面的循环可以被改写为：

```
var x = 0;

do

{

    x = window.prompt("What is my
favourite number?", x);
} while ((x != 42) && (x != null));

if (x == null)

    window.alert("You gave up!");

else

    window.alert("Yep - it's the
Ultimate Answer!");
```

使用 **break** 和 **continue** 语句

在 Microsoft Jscript 中当某些条件得到满足时，用 [break](#)(See 1.2.11.4) 语句来中断一个循环的运行。（请注意，也用 **break** 语句退出一个 **switch** 块。）。如果是一个 **for** 或者 **for...in** 循环，在更新计数器变量时使用 [continue](#)(See 1.2.11.6) 语句越过余下的代码块而直接跳到下一个循环中。

下面的例子基于前面的示例用 **break** 和 **continue** 语句控制循环。

```
var x = 0;

do

{
```

```
x = window.prompt("What is my  
favourite number?", x);  
  
// 判断用户是否选择取消？如果是，退出循环。  
if (x == null)  
    break;  
  
// 是否输入一个数？  
// 如果是则无需要求输入一个数。  
if (Number(x) == x)  
    continue;  
  
// 要求用户只输入数字。  
window.alert("Please only enter in  
numbers!");  
  
} while (x != 42)  
  
if (x == null)  
    window.alert("You gave up!");  
else  
    window.alert("Yep - it's the  
Ultimate Answer!");
```

1.1.1.7 JScript 函数

Microsoft Jscript 函数执行操作，也可以返回值。某些时候是计算或比较的结果。函数又被称为“全局方法”。

一个函数中包含有几个操作。这样可使得代码更合理化。可以写一组语句并给其命名，然后通过调用它并传递其需要的信息来运行整组语句。

给函数传递信息可以把信息放在函数名称后面的圆括号中。传递给函数的信息称作参数。某些函数根本不带任何参数，而其他函数带一个或者多个参数。在某些函数中，参数的个数取决于如何使用该函数。

Jscript 支持两种函数：一类是语言内部的函数，另一类是自己创建的。

特殊的内部函数

Jscript 语言包含很多内部函数。某些函数可以操作表达式和特殊字符，而其他函数将字符串转换为数值。一个有用的内部函数是 [eval\(\)](#) (See 1.2.7.1.26)。该函数可以对以字符串形式表示的任意有效的 Jscript 代码求值。**eval()** 函数有一个参数，该参数就是想要求值的代码。下面给出一个使用本函数的示例。

```
var anExpression = "6 * 9 % 7";

var total = eval(anExpression); // 将变量
total 赋值为 5。

var yetAnotherExpression = "6 * (9 % 7)";
total = eval(yetAnotherExpression) // 将
变量 total 赋值为 12。

// 将一个字符串赋给 totality （注意嵌套引用）

var totality = eval("'...surrounded by
acres of clams.'");
```

有关内部函数的详细信息请参考[语言参考](#)(See 2.61)。

创建自己的函数

在必要的时候，可以创建并使用自己的函数。一个函数的定义中包含了一个函数语句和一个 Jscript 语句块。

下面示例中的 **Checktriplet** 函数以三角形的边长为参数。通过查看三条边的长度是否可以组成一个毕达哥拉斯三元组（直角三角形斜边长度的平方等于其他两条边长的平方和）来计算该三角形是否为直角三角形。实际测试时 **checkTriplet** 函数要调用另两个函数中的一个函数。

注意在浮点数测试版本中极小数（“epsilon”）作为测试变量的使用。由于浮点运算的不确定性和舍入误差，除非问题中的三个值均已知为整数，直接测试这三个数是否组成毕达哥拉斯三元组是不可行的。因为直接的测试更为准确，本示例中的代码确定其是否可行，如果可行则使用它。

```
var epsilon = 0.000000000001; // 一些需要测试的极小数字。
```

```
// 测试整数的函数。
```

```
function integerCheck(a, b, c)
{
    // 测试。
    if ( (a*a) == ((b*b) + (c*c)) )
        return true;

    return false;
} // 整数检查函数的结尾。
```

```
// 测试浮点数的函数。
```

```
function floatCheck(a, b, c)
{
    // 得到测试数值。
    var delta = ((a*a) - ((b*b) + (c*c)))

    // 测试需要绝对值
    delta = Math.abs(delta);

    // 如果差小于 epsilon, 那么它相当接近。
```

```
    if (delta < epsilon)
        return true;

    return false;
} // 浮点检查函数的末尾。


// 三元检查。
function checkTriplet(a, b, c)
{
    // 创建临时变量，用于交换值
    var d = 0;

    // 先将最长的移动到位置“a”。

    // 需要的话交换 a 和 b
    if (b > a)
    {
        d = a;
        a = b;
        b = d;
    }

    // 需要的话交换 a 和 c
    if (c > a)
```

```
{  
    d = a;  
    a = c;  
    c = d;  
}  
  
// 测试全部的 3 个值，看其是否为整数？  
if (((a % 1) == 0) && ((b % 1) == 0)  
&& ((c % 1) == 0))  
{  
    // 如果成立，使用精确检查。  
    return integerCheck(a, b, c);  
}  
else  
{  
    // 如果不成立，取尽可能相近的。  
    return floatCheck(a, b, c);  
}  
} // 三元检查函数的末尾。  
  
// 下面的三个语句赋给范例值，用于测试。  
var sideA = 5;  
var sideB = 5;
```

```
var sideC = Math.sqrt(50.001);

// 调用函数。调用后, 'result' 中包含了结果。

var result = checkTriplet(sideA, sideB,
sideC);
```

1.1.1.8 Jscript 对象

Jscript 对象是属性和方法的集合。一个方法就是一个函数, 是对象的成员。属性是一个值或一组值 (以数组或对象的形式), 是对象的成员。Jscript 支持四种类型的对象: [内部对象](#)(See 1.1.1.8.2)、[生成的对象](#)(See 1.1.1.8.1)、宿主给出的对象 (如 Internet 浏览器中的 **window** 和 **document**) 以及 ActiveX 对象 (外部组件)。

作为数组的对象

在 Jscript 中, 对象和数组几乎是以相同的方式处理的。对象和数组均可以被赋予任意值, 实际上数组只是一种特殊的对象。数组和对象的区别在于数组有一个“奇妙的” **length** 属性, 而对象没有。这意味着可以给数组的一个元素赋予比其他元素更大的值。例如, **myArray[100] = "hello"** — 然后 **length** 属性将自动地被更新为 101 (新长度)。同样, 如果修改数组的 **length** 属性, 将删除不再是数组部分的元素。

Jscript 中所有的对象均支持“**expando**”属性或那些可以在运行时动态添加和删除的属性。这些属性可以有包含数字的任意名称。如果属性的名称是简单的标识符 <<参考标识符规则>>, 可以在对象名称的后面加句点, 例如:

```
var myObj = new Object();

// 添加两个 expando 属性, 'name' 和 'age'

myObj.name = "Fred";

myObj.age = 42;
```

如果属性名称不是一个简单的标识符, 或者在写脚本的时候不知道, 可以在方括号中使用任意表达式来索引属性。在 Jscript 中所有 **expando** 属性的名称在被添加到对象之前被转换为字符串。

```
var myObj = new Object();
```

```
// 添加两个无法写在 object.property 语  
// 法中的 expando 属性。  
// 第一个属性包含无效字符（空格），  
// 所以必须写在方括号里。  
myObj["not a valid identifier"] = "This  
is the property value";
```

```
// 第二个 expando 名称是一个数字，  
// 所以也必须写在方括号里。
```

```
myObj[100] = "100";
```

传统的作法是赋给数组元素以 0 开始的数字索引。这些数组元素与 **length** 属性相交互。然而，由于所有的数组也是对象，也支持 **expando** 属性。请注意，虽然如此，**expando** 属性并不以任何方式与 **length** 属性相交互。例如：

```
// 三个元素的数组
```

```
var myArray = new Array(3);
```

```
// 添加数据
```

```
myArray[0] = "Hello";
```

```
myArray[1] = 42;
```

```
myArray[2] = new Date(2000, 1, 1);
```

```
// 显示数组的长度 3
```

```
window.alert(myArray.length);
```

```
// 添加某些 expando 属性
```

```
myArray.expando = "JScript!";
```

```
myArray["another Expando"] = "Windows";
```

```
// 仍然显示 3, 因为两个 expando 属性
```

```
// 并不影响长度。
```

```
window.alert(myArray.length);
```

虽然 Jscript 并不直接支持多维数组,但是可以在数组元素中存储任意种类的数据——包含其他数组。所以通过在另一个数组的元素里存储其他数组可以得到多维数组的特性。例如,下面的代码为最大为 5 的数字建立了乘法表:

```
// 若是更大的表请改变本数
```

```
var iMaxNum = 5;
```

```
// 循环计数
```

```
var i, j;
```

```
// 新数组。由于数组从 0 开始计数,
```

```
// 而不是 1, 所以数组大小为 iMaxNum + 1。
```

```
var MultiplicationTable = new  
Array(iMaxNum + 1);
```

```
// 为每个主要的数做循环(表中的每一行)
```

```
for (i = 1; i <= iMaxNum; i++)
```

```
{
```

```
    // 生成表中的列
```

```
    MultiplicationTable[i] = new  
Array(iMaxNum + 1);
```

```
    // 将乘法的结果存在行中
```

```
    for (j = 1; j <= iMaxNum; j++)
    {
        MultiplicationTable[i][j] = i * j;
    }
}

window.alert(MultiplicationTable[3][4]);
// 显示 12

window.alert(MultiplicationTable[5][2]);
// 显示 10

window.alert(MultiplicationTable[1][4]);
// 显示 4
```

1.1.1.8.1 创建自己的对象

要创建自己的对象实例，必须首先为其定义一个构造函数。构造函数创建一个新对象，赋予对象属性，并在合适的时候赋予方法。例如，下面的示例为 `pasta` 对象定义了构造函数。注意 **this** 关键字的使用，它指向当前对象。

```
// pasta 是有四个参数的构造器。

function pasta(grain, width, shape,
hasEgg)
{
    // 是用什么粮食做的？

    this.grain = grain;

    // 多宽？（数值）
```

```
    this.width = width;

    // 横截面形状? (字符串)

    this.shape = shape;

    // 是否加蛋黄? (boolean)

    this.hasEgg = hasEgg;
}
```

定义了对象构造器后，用 **new** 运算符创建对象实例。

```
var spaghetti = new pasta("wheat", 0.2,
    "circle", true);
var linguine = new pasta("wheat", 0.3,
    "oval", true);
```

可以给对象实例添加属性以改变该实例，但是用相同的构造器生成的其他对象定义中并不包括这些属性，而且除非你特意添加这些属性那么在其他实例中并不显示出来。如果要将对象所有实例的附加属性显示出来，必须将它们添加到构造函数或构造器原型对象（原型在高级文档中讨论）中。

```
// spaghetti 的附加属性。

spaghetti.color = "pale straw";
spaghetti.drycook = 7;
spaghetti.freshcook = 0.5;

var chowFun = new pasta("rice", 3,
    "flat", false);

// chowFun 对象或其他现有的 pasta 对象

// 都没有添加到 spaghetti 对象
```



```
// 的三个新属性。
```

```
// 将属性 'foodgroup' 加到 pasta 原型对象
```

```
// 中，这样 pasta 对象的所有实例都可以有该属性，
```

```
// 包括那些已经生成的实例。
```

```
pasta.prototype.foodgroup =  
"carbohydrates"
```

```
// 现在 spaghetti.foodgroup 、  
chowFun.foodgroup, 等等
```

```
// 均包含值 "carbohydrates"。
```

在定义中包含方法

可以在对象的定义中包含方法（函数）。一种方法是在引用别处定义的函数的构造函数中添加一个属性。例如，下面的示例扩充上面定义的 `pasta` 构造函数以包含 **toString** 方法，该方法将在显示对象的值时被调用。

```
// pasta 是有四个参数的构造器。
```

```
// 第一部分与上面相同。
```

```
function pasta (grain, width, shape,  
hasEgg)  
{
```

```
    // 用什么粮食做的？
```

```
    this.grain = grain;
```

```
    // 多宽？（数值）
```

```
    this.width = width;
```

```
// 横截面形状? (字符串)

this.shape = shape;

// 是否加蛋黄? (boolean)

this.hasEgg = hasEgg;

// 这里添加 toString 方法 (如下定义)。

// 注意在函数的名称后没有加圆括号;

// 这不是一个函数调用, 而是

// 对函数自身的引用。

this.toString = pastaToString;
}

// 实际的用来显示 past 对象内容的函数。
function pastaToString()
{
    // 返回对象的属性。

    return "Grain: " + this.grain + "\n"
+
    "Width: " + this.width + "\n" +
    "Shape: " + this.shape + "\n" +
    "Egg?: " + Boolean(this.hasEgg);
}
```

```
var spaghetti = new pasta("wheat", 0.2,  
"circle", true);  
  
// 将调用 toString() 并显示 spaghetti 对象  
  
// 的属性（需要 Internet 浏览器）。  
  
window.alert(spaghetti);
```

1.1.1.8.2 内部对象

Microsoft Jscript 提供了 11 个内部（或“内置”）对象。它们是 **Array**、**Boolean**、**Date**、**Function**、**Global**、**Math**、**Number**、**Object**、**RegExp**、**Error** 以及 **String** 对象。每一个对象有相关的方法和属性，这在[语言参考](#)(See 2.61)中有详细的描述。本节中也描述了某些对象。

Array 对象

数组下标可以被认为是对象的属性，它是通过数字索引来引用的。注意添加到数组中的已命名的属性不能通过数字来索引；它们是与数组元素分离的。

使用 **new** 运算符和 **Array()** [构造器](#)(See 1.2.10.4) 生成一个新的数组，如下面的示例。

```
var theMonths = new Array(12);  
  
theMonths[0] = "Jan";  
theMonths[1] = "Feb";  
theMonths[2] = "Mar";  
theMonths[3] = "Apr";  
theMonths[4] = "May";  
theMonths[5] = "Jun";  
theMonths[6] = "Jul";  
theMonths[7] = "Aug";  
theMonths[8] = "Sep";
```

```

theMonths[9] = "Oct";
theMonths[10] = "Nov";
theMonths[11] = "Dec";

```

用关键字 **Array** 生成数组时，Jscript 包含了 **length** 属性，该属性记录了数组入口数。如果没有给该属性指定值，则设置长度为 0 且数组没有入口点。如果指定一个数值，则将长度设置为该数。如果指定了不止一个参数，则这些参数被用作数组的入口。另外，参数的数目被赋给 **length** 属性。如下面的示例与前一个示例是等价的。

```

var theMonths = new Array("Jan", "Feb",
"Mar", "Apr", "May", "Jun",
"Jul", "Aug", "Sep", "Oct", "Nov",
"Dec");

```

当向用关键字 **Array** 生成的数组中添加元素时，Jscript 自动改变属性 **length** 的值。Jscript 中的数组索引总是以 0 开始，而不是 1，所以属性 **length** 通常比数组的最大索引大 1。

String 对象

在 Jscript 中，可以将字符串（和数）当作对象来处理。[string 对象](#)(See 1.2.8.14) 有一些内置方法，可以和自己的字符串一起使用。其中一个是 [substring 方法](#)(See 1.2.7.4.30)，它返回字符串的一部分。该方法以两个数字作为参数。

```

aString = "0123456789";

var aChunk = aString.substring(4, 7);

// 将 aChunk 设为 "456"。

var aNotherChunk = aString.substring(7,
4); // 将 aNotherChunk 设为 "456"。

// 使用上面生成数组的示例：

```

```

firstLetter =
theMonths[5].substring(0,1); // 将变量
firstLetter 设为"J"。

```

String 对象的另一个属性是 **length** 属性。本属性包含了字符串中的字符数（空字符串为 0）。它是一个数值，可以直接在计算中使用。

```
var howLong = "Hello World".length // 设置变量 howLong 为 11。
```

Math 对象

Math 对象有许多预定义属性和方法。属性是特殊的数字。这些特殊的数字之一是 pi 值（近似 3.14159...）。这是 **Math.PI** 属性，如下例所示。

```
// 声明一个半径变量并赋数值。
```

```
var circleArea = Math.PI * radius * radius; // 注意 Math 和 PI 大写。
```

Math 对象的一个内置方法是乘幂方法（或 **pow**），使用该方法求得指定数的幂次。下面的例子同时使用了 pi 和乘幂。

```
// 本公式计算给定半径的球体的体积。
```

```
volume = (4/3) * (Math.PI*Math.pow(radius, 3));
```

Date 对象

Date 对象可以被用来表示任意的日期和时间，获取当前系统日期以及计算两个日期的间隔。它拥有一些预定义的属性和方法。通常，**Date** 对象给出星期；月份，天数和年份；以及以小时，分钟和秒表示的时间。该信息是基于 1970 年 1 月 1 日 00:00:00.000 GMT 开始的毫秒数，其中 GMT 是格林威治标准时间（首选术语是 UTC，或者“全球标准时间”，它引用的信号是由“世界时间标准”发布的）。Jscript 可以处理 250,000 B.C. 到 255,000 A.D 范围内的日期。

使用 **new** 运算符创建一个新的 **Date** 对象。下面的示例计算当年已过去的天数和今年剩下的天数。

```
/*
```

本示例使用前面定义的月份名称数组。

第一条语句以“Day Month Date 00:00:00 Year”格式

对 **thisIsToday** 变量赋值。

```
*/
```

```
var thisIsToday = new Date();
```

```

var today = new Date(); // 获取今天的日期。

// 提取年，月，日。
var thisYear = today.getFullYear();
var          thisMonth          =
theMonths[today.getMonth()];
var thisDay = thisMonth + " " +
today.getDate() + ", " + thisYear;

```

Number 对象

除了 **Math** 对象中可用的特殊数值属性（例如 **PI**）外，在 Microsoft Jscript 中，**Number** 对象有几个其他的数值属性。

属性	描述
MAX_VALUE	可能的最大数大约为 1.79E+308；可以是正数或负数。（具体数值随系统不同而有微小差别。）
MIN_VALUE	可能的最小数大约为 2.22E-308；可以是正数或负数。（具体数值随系统不同而有微小差别。）
NaN	特殊非数量值，“不是数值”。
POSITIVE_INFINITY	比最大正数（Number.MAX_VALUE）还要大的任意正值自动被转换为此值，代表正无穷大。
NEGATIVE_INFINITY	比最小的负数（负的 Number.MAX_VALUE）还小的任意值被自动转换为此值，代表负无穷。

Number.NaN 是一个特殊的属性，被定义为“不是数值”。例如被 0 除返回 **NaN**。试图解析一个无法被解析为数字的字符串同样返回 **Number.NaN**。把 **NaN** 与任何数值或本身作比较的结果都是不相等。不能通过与 **Number.NaN** 比较来测试 **NaN** 结果，而应该使用 **isNaN()** 函数。

1.1.1.9 JScript 保留关键字

Jscript 有一些保留字不能在标识符中使用。保留字对 Jscript 语言有特殊的含义，它们是语言语法的一部分。使用保留字在加载脚本的时候将产生编译错误。Jscript 还有一些留作将来使用的保留字。这些字不是现在的 Jscript 语言的一部分，然而它们是为将来的使用保留的。

保留词

break	delete	function	return	typeof
case	do	if	switch	var
catch	else	in	this	void
continue	false	instanceof	throw	while
debugger	finally	new	true	with
default	for	null	try	

为将来保留的词

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

当选择标识符时避免使用已经在内部 Jscript 对象或函数中使用的词，如 **String** 或 **parseInt**。

1.1.2 高级 JScript

[高级对象的创建](#)(See 1.1.2.1)

[递归](#)(See 1.1.2.2)

[变量范围](#)(See 1.1.2.3)

[复制、传递和比较数据](#)(See 1.1.2.4)

[使用数组](#)(See 1.1.2.5)

[特殊字符](#)(See 1.1.2.6)

[脚本疑难解答](#)(See 1.1.2.7)

1.1.2.1 创建高级对象

使用构造函数来创建对象

构造函数是一个函数，调用它来例示并初始化特殊类型的[对象](#)(See 1.1.1.8)。可以使用 **new** 关键字来调用一个构造函数。下面给出了使用构造函数的新示例。

```
var    myObject    =    new    Object();
```

// 创建没有属性的通用对象。

```
var myBirthday = new Date(1961, 5, 10);
```

// 创建一个 Date 对象。

```
var myCar = new Car(); //
```

创建一个用户定义的对象，并初始化其属性。

通过构造函数将一个参数作为特定的 **this** 关键字的值传递给新创建的空对象。然后构造函数负责为新对象执行适应的初始化（创建属性并给出其初始值）。完成后，构造函数返回它所构造的对象的一个参数。

编写构造函数

可以使用 **new** 运算符结合像 **Object()**、**Date()** 和 **Function()** 这样的预定义的构造函数来创建对象并对其初始化。面向对象的编程其强有力的特征是定义自定义构造函数以创建脚本中使用的自定义对象的能力。创建了自定义的构造函数，这样就可以创建具有已定义属性的对象。下面是自定义函数的示例（注意 **this** 关键字的使用）。

```
function Circle (xPoint, yPoint, radius)
{
    this.x = xPoint; // 圆心的 x 坐标。
    this.y = yPoint; // 圆心的 y 坐标。
    this.r = radius; // 圆的半径。
}
```

调用 **Circle** 构造函数时，给出圆心点的值和圆的半径（所有这些元素是完全定义一个独特的圆对象所必需的）。结束时 **Circle** 对象包含三个属性。下面是如何例示 **Circle** 对象。

```
var aCircle = new Circle(5, 11, 99);
```

使用原型来创建对象

在编写构造函数时，可以使用原型对象（它本身是所有构造函数的一个属性）的属性来创建继承属性和共享方法。原型属性和方法将按引用复制给类中的每个对象，因此它们都具有相同的值。可以在一个对象中更改原型属性的值，新的值将覆盖默认值，但仅在该实例中有效。属于这个类的其他对象不受此更改的影响。下面给出了使用自定义构造函数的示例，**Circle**（注意 **this** 关键字的使用）。

```
Circle.prototype.pi = Math.PI;
```

```
function ACirclesArea () {
```



```
return this.pi * this.r * this.r; //
```

计算圆面积的公式为 πr^2 。

```
}
```

```
Circle.prototype.area = ACirclesArea;
```

```
// 计算圆面积的函数现在是 Circle Prototype 对象的一个方法。
```

```
var a = ACircle.area(); // 此
```

为如何在 `Circle` 对象上调用面积函数。

使用这个原则，可以给预定义的构造函数（都具有原型对象）定义附加属性。例如，如果想要能够删除字符串的前后空格（与 VBScript 的 **Trim** 函数类似），可以给 **String** 原型对象创建自己的方法。

```
// 增加一个名为 trim 的函数作为
```

```
// String 构造函数的原型对象的一个方法。
```

```
String.prototype.trim = function()
```

```
{
```

```
    // 用正则表达式将前后空格
```

```
    // 用空字符串替代。
```

```
    return this.replace(/(^\\s*)|(\\s*$)/g,
    "");
```

```
}
```

```
// 有空格的字符串
```

```
var s = "    leading and trailing spaces
";
```

```
// 显示 "    leading and trailing spaces
```

```
(35) "
```

```
window.alert(s + " (" + s.length + ")");
```

```
// 删除前后空格
```

```
s = s.trim();
```

```
// 显示"leading and trailing spaces (27)"
```

```
window.alert(s + " (" + s.length + ")");
```

1.1.2.2 递归

递归是一种重要的编程技术。该方法用于让一个函数从其内部调用其自身。一个示例就是计算阶乘。0 的阶乘被特别地定义为 1。更大数的阶乘是通过计算 $1 * 2 * \dots$ 来求得的，每次增加 1，直至达到要计算其阶乘的那个数。

下面的段落是用文字定义的计算阶乘的一个函数。

“如果这个数小于零，则拒绝接收。如果不是一个整数，则将其向下舍入为相邻的整数。如果这个数为 0，则其阶乘为 1。如果这个数大于 0，则将其与相邻较小的数的阶乘相乘。”

要计算任何大于 0 的数的阶乘，至少需要计算一个其他数的阶乘。用来实现这个功能的函数就是已经位于其中的函数；该函数在执行当前的这个数之前，必须调用它本身来计算相邻的较小数的阶乘。这就是一个递归示例。

递归和迭代（循环）是密切相关的——能用递归处理的算法也都可以采用迭代，反之亦然。确定的算法通常可以用几种方法实现，您只需选择最自然贴切的方法，或者您觉得用起来最轻松的一种即可。

显然，这样有可能会出现一些问题。可以很容易地创建一个递归函数，但该函数不能得到一个确定的结果，并且不能达到一个终点。这样的递归将导致计算机执行一个“无限”循环。下面就是一个示例：在计算阶乘的文字描述中遗漏了第一条规则（对负数的处理），并试图计算任何负数的阶乘。这将导致失败，因为按顺序计算 -24 的阶乘时，首先不得不计算 -25 的阶乘；然而这样又不得不计算 -26 的阶乘；如此继续。很明显，这样永远也不会到达一个终止点。

因此在设计递归函数时应特别仔细。如果怀疑其中存在着无限递归的可能，则可以让该函数记录它调用自身的次数。如果该函数调用自身的次数太多，即使您已决定了它应调用多少次，就自动退出。

下面仍然是阶乘函数，这次是用 JScript 代码编写的。

```
// 计算阶乘的函数。如果传递了
```

```
// 无效的数值（例如小于零），
```

```
// 将返回 -1，表明发生了错误。若数值有效，

// 把数值转换为最相近的整数，并

// 返回阶乘。

function factorial(aNumber) {
aNumber = Math.floor(aNumber); // 如果这个数
不是一个整数，则向下舍入。
if (aNumber < 0) { // 如果这个数小于 0，拒绝接收。
    return -1;
}

    if (aNumber == 0) { // 如果为 0，则其阶乘为
1。

        return 1;
    }

        else return (aNumber *
factorial(aNumber - 1)); // 否则，递归直至完成。
}

```

1.1.2.3 变量范围

JScript 有两种变量范围：全局和局部。如果在任何函数定义之外声明了一个[变量](#)，则该变量为全局变量，且该变量的值在整个持续范围内都可以访问和修改。如果在函数定义内声明了一个变量，则该变量为局部变量。每次执行该函数时都会创建和破坏该变量；且它不能被该函数外的任何事物访问。

像 C++ 这样的语言也有“块范围”。在这里，任何一对“{}”都定义新的范围。

JScript 不支持块范围。

一个局部变量的名称可以与某个全局变量的名称相同，但这是完全不同和独立的两个变量。因此，更改一个变量的值不会影响另一个变量的值。在声明局部变量的函数内，只有该局部变量有意义。

```
var aCentaur = "a horse with rider,";
```

```
// aCentaur 的全局定义。
```

```
// JScript 代码，为简洁起见有省略。
```

```
function antiquities() // 在这个函数中声明了一个局部  
aCentaur 变量。
```

```
{
```

```
// JScript 代码，为简洁起见有省略。
```

```
var aCentaur = "A centaur is probably a  
mounted Scythian warrior";
```

```
// JScript 代码，为简洁起见有省略。
```

```
    aCentaur += ", misreported; that is, ";
```

```
// 添加到局部变量。
```

```
// JScript 代码，为简洁起见有省略。
```

```
} // 函数结束。
```

```
var nothinginparticular = antiquities();  
aCentaur += " as seen from a distance by  
a naive innocent.";
```

```
/*
```

```
在函数内，该变量的值为 "A centaur is probably a  
mounted Scythian warrior,  
misreported; that is, "; 在函数外，该变量的值为这句话的其
```

余部分：

```
"a horse with rider, as seen from a  
distance by a naive innocent."
```

```
*/
```

很重要的一点是注意变量是否是在其所属范围的开始处声明的。有时这会导致意想不到的情况。

```
tweak();
```

```
var aNumber = 100;
```

```
function tweak() {
```

```
var newThing = 0; // 显式声明 newThing 变量。
```

```
    // 本语句将未定义的变量赋给 newThing，因为已有名为  
aNumber 的局部变量。
```

```
    newThing = aNumber;
```

```
    // 下一条语句将值 42 赋给局部的 aNumber。aNumber =  
42;
```

```
if (false) {
```

```
    var aNumber; // 该语句永远不会执行。
```

```
    aNumber = 123; // 该语句永远不会执行。
```

```
    } // 条件语句结束。
```

```
} // 该函数定义结束。
```

当 JScript 运行函数时，首先查找所有的变量声明，

```
var someVariable;
```

并以未定义的初始值创建变量。如果变量被声明时有值，

```
var someVariable = "something";
```

那么该变量仍以未定义的值初始化，并且只有在运行了声明行时才被声明值取代，假如曾经被声明过。

JScript 在运行代码前处理变量声明，所以声明是位于一个条件块中还是其他某些结构中无关紧要。JScript 找到所有的变量后立即运行函数中的代码。如果变量是在函数中显式声明的 — 也就是说，如果它出现于赋值表达式的左边但没有用 `var` 声明 — 那么将把它创建为全局变量。

1.1.2.4 复制、传递和比较数据

在 JScript 中，对数据的处理取决于该数据的类型。

按值和按引用的比较

Numbers 和 Boolean 类型的值 (**true** 和 **false**) 是按值来复制、传递和比较的。当按值复制或传递时，将在计算机内存中分配一块空间并将原值复制到其中。然后，即使更改原来的值，也不会影响所复制的值（反过来也一样），因为这两个值是独立的实体。

对象、数组以及函数是按引用来复制、传递和比较的。当按地址复制或传递时，实际是创建一个指向原始项的指针，然后就像拷贝一样来使用该指针。如果随后更改原始项，则将同时更改原始项和复制项（反过来也一样）。实际上只有一个实体；“复本”并不是一个真正的复本，而只是该数据的又一个引用。

当按引用比较时，要想比较成功，两个变量必须参照完全相同的实体。例如，两个不同的 **Array** 对象即使包含相同的元素也将比较为不相等。要想比较成功，其中一个变量必须为另一个的参考。要想检查两个数组是否包含了相同的元素，比较 **toString()** 方法的结果。

最后，字符串是按引用复制和传递的，但是是按值来比较的。请注意，假如有两个 **String** 对象（用 `new String("something")` 创建的），按引用比较它们，但是，如果其中一个或者两者都是字符串值的话，按值比较它们。

注意 鉴于 [ASCII](#) 和 ANSI 字符集的构造方法，按序列顺序大写字母位于小写字母的前面。例如 "Zoo" 小于 "aardvark"。如果想执行不区分大小写的匹配，可以对两个字符串调用 `toUpperCase()` 或 `toLowerCase()`。

传递参数给函数

按值传递一个参数给函数就是制作该参数的一个独立复本，即一个只存在于该函数内的复本。即使按引用传递对象和数组时，如果直接在函数中用新值覆盖原先的值，在函数外并不反映新值。只有在对象的属性或者数组的元素改变时，在函数外才可以看出。

例如（使用 IE 对象模式）：

```
// 本代码段破坏（覆盖）其参数，所以
```

```
// 调用代码中反映不出变化。
```

```
function Clobber(param)
```

```
{
```

```
    // 破坏参数：在调用代码中
```

```
    // 看不到。
```

```
    param = new Object();
```

```
    param.message = "This will not work";
```

```
}
```

```
// 本段代码改变参数的属性，
```

```
// 在调用代码中可看到属性改变。
```

```
function Update(param)
```

```
{
```

```
    // 改变对象的属性；
```

```
    // 可从调用代码中看到改变。
```

```
    param.message = "I was changed";
```

```
}
```

```
// 创建一个对象，并赋给一个属性。
```

```
var obj = new Object();
```

```
obj.message = "This is the original";
```

```
// 调用 Clobber，并输出 obj.message。注意，它没有发生变化。
```

```
Clobber(obj);
```

```
window.alert(obj.message); // 仍然显示 "This  
is the original"。
```

```
// 调用 Update, 并输出 obj.message。注意, 它已经被改变了。  
Update(obj);  
window.alert(obj.message); // 显示 "I was  
changed"。
```

检验数据

当按值进行检验时, 是比较两个截然不同的项以查看它们是否相等。通常, 该比较是逐字节进行的。当按引用进行检验时, 是看这两项是否是指向同一个原始项的指针。如果是, 则比较结果是相等; 如果不是, 即使它们每个字节都包含完全一样的值, 比较结果也为不相等。

按引用复制和传递字符串能节约内存; 但是由于在字符串被创建后不能进行更改, 因此可以按值进行比较。这样可以检查两个字符串是否包含相同的内容, 即使它们是完全独立产生的。

1.1.2.5 使用数组

数组下标

JScript 中的数组是稀疏的。也就是说, 如果一个数组具有三个元素, 编号分别为 0、1 和 2, 您就可以创建元素 50, 而不必担心从 3 到 49 的参数。如果该数组有一个自动的 `length` 变量, (请参阅[内部对象](#)(See 1.1.1.8.2)了解有关数组长度的自动监控的说明), 该 `length` 变量被设为 51, 而不是 4。当然您可以创建各元素的编号之间没有间隙的数组, 不过没有必要这样做。

在 JScript 中, 对象和数组几乎相同。两个主要差别是对象没有自动长度属性, 而数组没有对象的[属性](#)和方法。

数组寻址

使用方括号 “[]” 来寻址数组。方括号中是一个数值或一个值为整数的[表达式](#)。下面的示例假定在脚本的其他地方已定义了 `entryNum` 变量, 且已赋值。

```
theListing = addressBook[entryNum];  
  
theFirstLine = theListing[1];
```

将对象作为关联数组

通常, 使用点运算符 “.” 访问对象的属性。例如,

```
myObject.aProperty
```

在这里, 属性名称是一个标识符。也可以用索引运算符 “[]” 访问对象的属性。在

这里，是把对象看作一个关联数组。关联数组是一种数据结构，它可以动态地将任意的数据的值与任意的字符串相关联。例如，

`myObject["aProperty"]` // 与上面相同。

尽管索引运算符更多地用于访问数组元素，当用于对象时，索引总是以字符串文字表示的属性名称。

注意访问对象属性的两种方法的重要差异。

运算符	属性名称作为	对属性名称的处理
点 “.”	标识符	不能作为数据处理
索引 “[]”	字符串文字	能被作为数据处理

在运行之前并不知道属性名称时，这个差异会有用（比如基于用户输入构造对象时）。要想从一个关联数组提取所有的属性，必须用 **for ... in** 循环。

1.1.2.6 特殊字符

JScript 提供了一些特殊字符，允许在字符串中包括一些无法直接键入的字符。每个字符都以反斜杠开始。反斜杠是一个转义字符，表示 JScript 解释器下面的字符为特殊字符。

转义序列	字符
<code>\b</code>	退格
<code>\f</code>	走纸换页
<code>\n</code>	换行
<code>\r</code>	回车
<code>\t</code>	横向跳格 (Ctrl-I)
<code>\'</code>	单引号
<code>\"</code>	双引号
<code>\\</code>	反斜杠

请注意，由于反斜杠本身用作转义符，因此不能直接在脚本中键入一个反斜杠。如果要产生一个反斜杠，必须一起键入两个反斜杠 (`\\`)。

```
document.write('The image path is
C:\\webstuff\\mypage\\gifs\\garden.gif.
');
```

```
document.write('The caption reads,
"After the snow of \'97. Grandma\'s
```

```
house is covered.'');
```

1.1.2.7 脚本问题解答

如果不够细致，任何编程语言都有一些可能发生错误的地方，而且每种语言都有其特殊之处。例如，对于 [null](#) 值：JScript 中这个值与 C 或 C++ 语言中的 **Null** 值所起的作用是不一样的。

下面提供了一些在编写 JScript 脚本时可能遇到的问题。

语法错误

由于编程语言中的语法比自然语言的语法要严格得多，因此在编写脚本时对细节应倍加关注。例如，如果您本意是将字符串作为某个参数，但是在键入时忘了使用引号引起来，就会产生问题。

脚本解释顺序

对 JScript 的解释是 Web 浏览器的 HTML 语法分析处理的一部分。因此，如果在文档的 <HEAD> 标识中放置了一个脚本，则将在检查所有的 <BODY> 标识之前加以解释。如果在 <BODY> 标识中将创建对象，但由于在分析处理 <HEAD> 标识时这些对象尚不存在，因而不能被脚本操作。

注意 本情况特定于 IE。ASP 和 WSH 具有不同的运行模式（其他宿主亦是）。

自动类型强制

JScript 是一种具有自动强制的自由类型语言。因此，尽管实际上不同类型的值是不相等的，但对下述示例中的表达式求值都将得到 **true**。

```
"100" == 100;
```

```
false == 0;
```

要核对类型与值都一致，用“严格相等”运算符（===）。下面两个表达式的值为 **false**：

```
"100" === 100;
```

```
false === 0;
```

运算符优先级

在对表达式求值时某个特定运算符的执行主要是根据 [运算符优先级](#) (See 1.2.9.1.1)，而不是表达式的位置。因此，在下面的示例中，乘法将先于减法执行，尽管在该表达式中第一个出现的运算符是减法。

```
theRadius      =      aPerimeterPoint      -
theCenterpoint * theCorrectionFactor;
```

对对象使用 for...in 循环

当使用 [for...in](#)(See 1.2.11.9) 循环对某个对象的属性进行遍历时，不必预先确定或管理将要指定给该循环计数器变量的对象字段的顺序。此外，在该语言的不同实现方案中该顺序可能会不一样。

with 关键字

[with](#)(See 1.2.11.20) 语句可以方便地用来引用某个特定对象中已有的属性，但是不能用来给对象添加属性。要给对象创建新的属性，必须明确地引用该对象。

this 关键字

尽管可以在对象的定义范围内使用 [this](#)(See 1.2.11.15) 关键字来引用该对象本身，但是当函数不是该对象的定义时，就不能象普通情况那样使用 **this** 或类似的关键字来引用当前的执行函数。如果该函数被指定为某个对象的方法，则可以在该函数内使用 **this** 关键字来引用该对象。

编写一个脚本，该脚本在 IE 中写脚本

当解释程序遇到</SCRIPT>标记时会终止当前脚本。要显示"</SCRIPT>" 本身，请将其改写为至少两个字符串，例如 "</SCR" 和 "IPT>"，这样就可以在输出语句中将其连接在一起。

IE 中的隐式窗口引用

由于同时可以打开多个窗口，任何隐式的窗口引用都被指向当前窗口。对于其他窗口必须使用显式引用。

1.1.2.8 条件编译

使用条件编译可以使用 Jscript 语言的新特性并且与不支持该特性的老版本兼容。用 **@cc_on** 语句、**@if** 或 **@set** 语句来激活条件编译。条件编译的某些典型用途包括使用 Jscript 中的新特性、在脚本中嵌入调试支持以及跟踪代码的运行。一般将条件编译代码放在注释中，所以不能理解条件编译的宿主（如 Netscape Navigator）就忽略了条件编译。下面是一个示例。

```
/*@cc_on @*/

/*@if (@_jscript_version >= 4)

    alert("JScript version 4 or better");

@else @*/
```

```

    alert("You need a more recent script
engine.");

/*@end @*/

```

本示例使用了特殊的注释分隔符，该分隔符只有在 **@cc_on** 语句激活条件编译时才使用。不支持条件编译的脚本引擎只能看到一个需要更新脚本引擎的信息。

1.1.2.9 条件编译变量

变量	描述
@_win32	在 Win32 系统上运行行为 true。
@_win16	在 Win16 系统上运行行为 true。
@_mac	在 Apple Macintosh 系统上运行行为 true。
@_alpha	在 DEC Alpha 处理器上运行行为 true。
@_x86	在 Intel 处理器上运行行为 true。
@_mc680x0	在 Motorola 680x0 处理器上运行行为 true。
@_PowerPC	在 Motorola PowerPC 处理器上运行行为 true。
@_jscript	永远为 true。
@_jscript_build	包含 Jscript 脚本引擎创建号。
@_jscript_version	包含以 major、minor 为格式的 Jscript 版本号。

1.1.3 在浏览器中显示信息

Microsoft JScript 提供了两种方式来在浏览器中直接显示数据。可以使用 **write()** 和 **writeln()**，这两个函数是 **document** 对象的方法。也可以在浏览器中以表格的方式显示信息，以及用 [警告、提示和确认](#) 消息框来显示信息。

使用 document.write() 和 document.writeln()

显示信息最常用的方式是 **document** 对象的 **write()** 方法。该方法用一个字符串作为其参数，并在浏览器中显示。该字符串可以是普通文本或 HTML。

字符串可以用单引号或双引号引起来。这样可以引用那些包含引号或撇号的内容。

```

document.write("Pi    is    approximately
equal to " + Math.PI);

document.write( );

```

注意 下面的简单函数可以避免在浏览器中显示信息时不得不键入“document.write”。该函数不能告知要显示的信息是否未定义，而是发布给命令“w()”，该命令将显示一个空行。

```
function w(m) { // 编写函数。

m = "" + m + ""; // 确保变量 m 是一个字符串。

if ("undefined" != m) { // 判别是否为空或其它未定义的
项。

    document.write(m);

}

document.write("<br>");

}

w('<IMG SRC="horse.gif">');

w();

w("This is an engraving of a horse.");

w();
```

writeln() 方法与 **write()** 方法几乎一样，差别仅在于前者将在所提供的任何字符串后添加一个换行符。在 HTML 中，这通常只会在后面产生一个空格；不过如果使用了 **<PRE>** 和 **<XMP>** 标识，这个换行符会被解释，且在浏览器中显示。在调用 **write()** 方法时，如果该文档不处于在调用 **write()** 方法时的打开和分析的过程中，该方法将打开并清除该文档，所以它可能是有危险的。该示例显示了一个每隔一分钟就显示时间的脚本，但是在第一次显示后由于它从过程中将自己清除，因此会导致失败。

<HTML>

<HEAD>

<SCRIPT LANGUAGE="JScript">

```
function singOut() {
var theMoment = new Date();
```

```
var theHour = theMoment.getHours();
var theMinute = theMoment.getMinutes();
var         theDisplacement         =
(theMoment.getTimezoneOffset() / 60);
theHour -= theDisplacement;
if (theHour > 23) {
theHour -= 24
}
document.write(theHour + " hours, " +
theMinute + " minutes, Coordinated
Universal Time.");
window.setTimeout("singOut();", 60000);
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
singOut();
</SCRIPT>
</BODY>
</HTML>
```

如果使用 `window` 对象的 **alert()** 方法而不是 **document.write()**, 则该脚本可以运行。

```
window.alert(theHour + " hours, " +  
theMinute + " minutes, Coordinated  
Universal Time.");  
  
window.setTimeout("singOut();", 60000);  
}
```

清除当前文档

document 对象的 **clear()** 方法将清空当前文档。该方法也将清除您的脚本（随文档的其他部分一起），因此要特别注意该方法的使用方式及在什么时候使用该方法。

```
document.clear();
```

1.1.3.1 使用消息框

使用警告、提示和确认

可以使用警告、确认和提示消息框来获得用户的输入。这些消息框是 **window** 对象的接口方法。由于 **window** 对象位于对象层次的顶层，因此实际应用中不必使用这些消息框的全名（例如 "window.alert()"），不过采用全名是一个好主意，这样有助于您记住这些消息框属于哪个对象。

警告消息框

alert 方法有一个参数，即希望对用户显示的文本字符串。该字符串不是 HTML 格式。该消息框提供了一个“确定”按钮让用户关闭该消息框，并且该消息框是模式对话框，也就是说，用户必须先关闭该消息框然后才能继续进行操作。

```
window.alert("欢迎！请按“确定”继续。");
```

确认消息框

使用确认消息框可向用户问一个“是-或-否”问题，并且用户可以选择单击“确定”按钮或者单击“取消”按钮。**confirm** 方法的返回值为 **true** 或 **false**。该消息框也是模式对话框：用户必须在响应该对话框（单击一个按钮）将其关闭后，才能进行下一步操作。

```
var truthBeTold = window.confirm("单击“确定”  
继续。单击“取消”停止。");
```

```
if (truthBeTold) {
```

```
window.alert("欢迎访问我们的 Web 页!");  
} else window.alert("再见啦!");
```

提示消息框

提示消息框提供了一个文本字段，用户可以在此字段输入一个答案来响应您的提示。该消息框有一个“确定”按钮和一个“取消”按钮。如果您提供了一个辅助字符串参数，则提示消息框将在文本字段显示该辅助字符串作为默认响应。否则，默认文本为 "<undefined>"。

与 **alert()** 和 **confirm()** 方法类似，**prompt** 方法也将显示一个模式消息框。用户在继续操作之前必须先关闭该消息框

```
var theResponse = window.prompt("欢迎?", "  
请在此输入您的姓名。");
```

1.2 语言参考

✦ 特性信息

✦ 字母顺序的关键字列表

✦ 错误

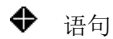
✦ 函数

✦ 方法

✦ 对象

✦ 运算符

✦ 属性



语句

欢迎使用 JScript 语言参考

这些方便实用的信息将帮助您了解 JScript 的各个部分。

在“字母顺序的关键字列表”中，可以找到按字母顺序列出的所有 JScript 语言的主题。如果只需要查看某个主题（例如对象），则有对该主题进行详细说明的章节可供查阅。

如何操作呢？单击左边任意一个标题，即可显示该标题所包含的项目列表。再从该列表中选择要查看的主题。在打开所选主题后，就可以方便地链接到相关章节。请尽情浏览 JScript 语言参考的各个部分，你会发现 JScript 语言有多么丰富。

特性

描述

[JScript 特性（非-ECMA）](#)

JScript 目前的 ECMA 特性列表。

[JScript 的特性（ECMA）](#) (See JScript 目前的非-ECMA 特性列表。

1.2.2)

[Microsoft Scripting 运行时特性](#) JScript 目前的 scripting 运行时特性列表。

(See 2.63)

语言元素

描述

[! 运算符](#) (See 1.2.9.26)

对表达式进行逻辑非运算。

[!= 运算符](#) (See 1.2.9.15)

比较两个表达式的值是否相等。

[!== 运算符](#) (See 1.2.9.15)

比较两个表达式的值，看其值是否相等或数据类型是否一致。

[0...n 属性](#) (See 2.65)

返回单个参数的实际值，该参数来自当前运行函数的 **参数** 属性返回的 **参数** 对象。

[\\$1...\\$9 属性](#) (See 1.2.10.1)

返回在模式匹配中找到的最近的九条记录。

[% 运算符](#) (See 1.2.9.29)

两个表达式的值相除，返回余数。

[%= 运算符](#) (See 1.2.9.28)

用变量的值除以表达式的值，余数赋给变量。

[& 运算符](#) (See 1.2.9.6)

对两个表达式执行按位“与”运算。

[&= 运算符](#) (See 1.2.9.5)

对变量和表达式执行按位“与”运算，结果赋给变量。

[&& 运算符](#) (See 1.2.9.25)

对两个表达式执行逻辑连接运算。

[* 运算符](#) (See 1.2.9.31)

将两个表达式的值相乘。

[*= 运算符](#) (See 1.2.9.30)

将变量与表达式的值相乘，结果赋给变量。

[+ 运算符](#) (See 1.2.9.3)

将两个数字表达式的值相加，或连接两个字符串。

[++ 运算符](#) (See 1.2.9.22)

变量值加 1。

[+= 运算符](#) (See 1.2.9.2)

将表达式的值加到变量中。

[, 运算符](#) (See 1.2.9.14)

使两个表达式按顺序执行。

[- 运算符](#) (See 1.2.9.35)

从一个表达式中减去另一个表达式的值，或对单个表达式取反。

[-- 运算符](#) (See 1.2.9.22)

变量值减 1。

-= 运算符 (See 1.2.9.35)	变量值减去表达式的值，结果赋给变量。
/ 运算符 (See 1.2.9.20)	两个表达式的值相除。
/*.*/* (多行注释语句) (See 使 JScript 语法分析器忽略多行注释。1.2.11.5)	
// (单行注释语句) (See 使 JScript 语法分析器忽略单行注释。1.2.11.5)	
/= 运算符 (See 1.2.9.19)	变量值除以表达式的值，结果赋给变量。
< 运算符 (See 1.2.9.15)	比较一个表达式的值是否小于另一个表达式。
<< 运算符 (See 1.2.9.7)	将表达式向左移位。
<<= 运算符 (See 1.2.9.24)	将变量的值左移由表达式指定的位数，结果赋给变量。
<= 运算符 (See 1.2.9.15)	比较一个表达式的值是否小于等于另一个表达式。
= 运算符 (See 1.2.9.4)	为变量赋值。
== 运算符 (See 1.2.9.15)	比较两个表达式是否相等。
=== 运算符 (See 1.2.9.15)	比较两个表达式，看其值是否相等或数据类型是否一致。
> 运算符 (See 1.2.9.15)	比较一个表达式的值是否大于另一个表达式。
>= 运算符 (See 1.2.9.15)	比较一个表达式的值是否大于等于另一个表达式。
>> 运算符 (See 1.2.9.11)	将表达式向左移位，符号位不变。
>>= 运算符 (See 1.2.9.33)	将变量的值右移由表达式指定的位数，符号位不变，结果赋给变量。
>>> 运算符 (See 1.2.9.37)	将表达式向左移位，包括符号位。
>>>= 运算符 (See 1.2.9.38)	将变量的值右移由表达式指定的位数，包括符号位，结果赋给变量。
?: 运算符 (See 1.2.9.17)	根据条件执行其中一个语句。
~ 运算符 (See 1.2.9.8)	对表达式执行按位“非”(取反)运算。
 运算符 (See 1.2.9.10)	对两个表达式执行按位“或”运算。
 = 运算符 (See 1.2.9.9)	对变量和表达式的值执行按位“或”运算，结果赋给变量。
 运算符 (See 1.2.9.27)	对两个表达式执行逻辑或运算。
^ 运算符 (See 1.2.9.13)	对两个表达式执行异或运算。
^= 运算符 (See 1.2.9.12)	对变量和表达式的值执行按位异或运算，结果赋给变量。
@cc on 语句 (See 1.2.11.1)	激活条件编译支持。
@if 语句 (See 1.2.11.2)	根据表达式的值，有条件地执行一组语句。
@set 语句 (See 1.2.11.3)	创建用于条件编译语句的变量。
abs 方法 (See 1.2.7.1.1)	返回一个数的绝对值。
acos 方法 (See 1.2.7.1.2)	返回一个数的反余弦。
ActiveXObject 对象 (See 启用并返回一个 Automation 对象的引用。1.2.8.1)	
加法运算符 (+) (See 1.2.9.3)	将两个数字表达式的值相加，或连接两个字符串。
anchor 方法 (See 1.2.7.1.3)	在对象的指定文本两端加上一个带 NAME 属性的 HTML 锚点。
apply 方法 (See 1.2.7.1.4)	应用对象的一个方法，用当前对象代替另一对象。

- [arguments 属性](#) (See 1.2.10.2) 返回一个包含传递给当前执行函数的每个参数的数组。
- [Array 对象](#) (See 1.2.8.2) 提供对创建任何数据类型的数组的支持。
- [asin 方法](#) (See 1.2.7.1.5) 返回一个数的反正弦。
- [赋值运算符 \(=\)](#) (See 1.2.9.4) 将一个值赋给变量。
- [atan 方法](#) (See 1.2.7.1.6) 返回一个数的反正切。
- [atan2 方法](#) (See 1.2.7.1.7) 返回从 X 轴到点 (y, x) 的角度 (以弧度为单位)。
- [atEnd 方法](#) (See 1.2.7.1.8) 返回一个指示枚举算子是否处于集合结束处的 Boolean 值。
- [big 方法](#) (See 1.2.7.1.9) 在 **String** 对象的文本两端加入 HTML 的<BIG>标识。
- [按位与运算符 \(&\)](#) (See 1.2.9.6) 对两个表达式执行按位与操作。
- [按位左移运算符 \(<<\)](#) (See 1.2.9.7) 将一个表达式的各位向左移。
- [按位取非运算符 \(~\)](#) (See 1.2.9.8) 对一个表达式执行按位取非 (求非) 操作。
- [按位或运算符 \(|\)](#) (See 1.2.9.10) 对两个表达式指定按位或操作。
- [按位右移运算符 \(>>\)](#) (See 1.2.9.11) 将一个表达式的各位向右移, 保持符号不变。
- [按位异或运算符 \(^\)](#) (See 1.2.9.13) 对两个表达式执行按位异或操作。
- [blink 方法](#) (See 1.2.7.1.10) 将 HTML 的 <BLINK> 标识添加到 **String** 对象中的文本两端。
- [bold 方法](#) (See 1.2.7.1.11) 将 HTML 的 标识添加到 **String** 对象中的文本两端。
- [Boolean 对象](#) (See 1.2.8.3) 创建一个新的 Boolean 值。
- [break 语句](#) (See 1.2.11.4) 终止当前循环, 或者如果与一个 *label* 语句关联, 则终止相关联的语句。
- [call 方法](#) (See 1.2.7.1.12) 应用对象的一个方法, 用当前对象代替另一对象。
- [callee 属性](#) (See 2.66) 返回正执行的函数对象, 它是指定的函数对象的文本正文。
- [caller 属性](#) (See 1.2.10.3) 返回调用当前函数的函数引用。
- [catch 语句](#) (See 1.2.11.17) 包含在 **try** 语句块中的代码发生错误时执行的语句。
- [ceil 方法](#) (See 1.2.7.1.13) 返回大于或等于其数值参数的最小整数。
- [charAt 方法](#) (See 1.2.7.1.14) 返回位于指定索引位置的字符。
- [charCodeAt 方法](#) (See 1.2.7.1.15) 返回指定字符的 Unicode 编码。
- [逗号运算符 \(,\)](#) (See 1.2.9.14) 使两个表达式连续执行。
- [/*.*/* \(多行注释语句\)](#) (See 1.2.11.5) 使多行注释部分被 JScript 语法分析器忽略。
- [注释语句 - 单行 \(//\)](#) (See 1.2.11.5) 使 JScript 语法分析器忽略单行注释。
- [比较运算符](#) (See 1.2.9.15) 返回一个显示比较结果的 Boolean 值。
- [compile 方法](#) (See 1.2.7.1.16) 将一个正则表达式编译为内部格式。
- [复合赋值运算符](#) 复合赋值运算符列表。

[concat 方法 \(Array\)](#) (See 返回一个由两个数组合并组成的新数组。

1. 2. 7. 1. 17)

[concat 方法 \(String\)](#) (See 返回一个包含给定的两个字符串连接的 **String** 对象。

1. 2. 7. 1. 18)

[条件 \(三元\) 运算符 \(?:\)](#) (See 根据条件执行两个表达式之一。

1. 2. 9. 17)

[constructor 属性](#) (See 1. 2. 10. 4) 指定创建对象的函数。

[continue 语句](#) (See 1. 2. 11. 6) 停止循环的当前迭代，并开始一次新的迭代。

[cos 方法](#) (See 1. 2. 7. 1. 19) 返回一个数的余弦。

[Date 对象](#) (See 1. 2. 8. 4) 提供日期和时间的基本存储和检索。

[decodeURI 方法](#) (See 1. 2. 7. 1. 20) 返回一个已编码的通用资源标识符 (URI) 的解码版。

[decodeURIComponent 方法](#) (See 返回一个已编码的通用资源标识符 (URI) 组件的解码版。

1. 2. 7. 1. 21)

[递减运算符 \(--\)](#) (See 1. 2. 9. 22) 将变量减一。

[delete 运算符](#) (See 1. 2. 9. 18) 删除对象的属性，或删除数组中的一个元素。

[description 属性](#) (See 1. 2. 10. 5) 返回或设置关于指定错误的描述字符串。

[Dictionary 对象](#) (See 1. 4. 2. 1) 存储数据键、项目对的对象。

[dimensions 方法](#) (See 返回 VArray 的维数。

1. 2. 7. 1. 22)

[除法运算符 \(/\)](#) (See 1. 2. 9. 20) 对两个表达式执行除法运算。

[do...while 语句](#) (See 1. 2. 11. 7) 先执行一次语句块，然后重复执行该循环，直至条件表达式的值为 **false**。

[E 属性](#) (See 1. 2. 10. 6) 返回 Euler 常数，即自然对数的底。

[encodeURIComponent 方法](#) (See 1. 2. 7. 1. 23) 将文本字符串编码为合法的通用资源标识符 (URI)。

[encodeURIComponent 方法](#) (See 将文本字符串编码为合法的通用资源标识符 (URI) 组件。

1. 2. 7. 1. 24)

[Enumerator 对象](#) (See 1. 2. 8. 5) 提供集合中的项的枚举。

[相等运算符 \(==\)](#) (See 1. 2. 9. 15) 比较两个表达式，看是否相等。

[Error 对象](#) (See 1. 2. 8. 6) 包含在运行 JScript 代码时发生错误信息的对象。

[escape 方法](#) (See 1. 2. 7. 1. 25) 对 **String** 对象编码，以便在所有计算机上都能阅读。

[eval 方法](#) (See 1. 2. 7. 1. 26) 对 JScript 代码求值然后执行。

[exec 方法](#) (See 1. 2. 7. 1. 27) 在指定字符串中执行一个匹配查找。

[exp 方法](#) (See 1. 2. 7. 1. 28) 返回 *e* (自然对数的底) 的幂。

[FileSystemObject 对象](#) (See 提供对计算机文件系统的访问。

1. 4. 2. 6)

[fixed 方法](#) (See 1. 2. 7. 2. 1) 将 HTML 的<TT> 标识添加到 **String** 对象中的文本两端。

[floor 方法](#) (See 1. 2. 7. 2. 2) 返回小于或等于其数值参数的最大整数。

- [fontcolor 方法](#) (See 1.2.7.2.3) 将 HTML 带 COLOR 属性的 标识添加到 **String** 对象中的文本两端。
- [fontsize 方法](#) (See 1.2.7.2.4) 将 HTML 带 SIZE 属性的 标识添加到 **String** 对象中的文本两端。
- [for 语句](#) (See 1.2.11.8) 只要指定的条件为 **true**，就一直执行语句块。
- [for...in 语句](#) (See 1.2.11.9) 对应于对象或数组中的每个元素执行一个或多个语句。
- [fromCharCode 方法](#) (See 1.2.7.2.5) 返回 Unicode 字符值的字符串。
- [Function 对象](#) (See 1.2.8.7) 创建一个新的函数。
- [function 语句](#) (See 1.2.11.10) 声明一个新的函数。
- [getDate 方法](#) (See 1.2.7.2.6) 使用当地时间返回 **Date** 对象的月份日期值。
- [getDay 方法](#) (See 1.2.7.2.7) 使用当地时间返回 **Date** 对象的星期几。
- [getFullYear 方法](#) (See 1.2.7.2.8) 使用当地时间返回 **Date** 对象的年份。
- [getHours 方法](#) (See 1.2.7.2.9) 使用当地时间返回 **Date** 对象的小时值。
- [getItem 方法](#) (See 1.2.7.2.10) 返回指定位置的项。
- [getMilliseconds 方法](#) (See 1.2.7.2.11) 使用当地时间返回 **Date** 对象的毫秒值。
- [getMinutes 方法](#) (See 1.2.7.2.12) 使用当地时间返回 **Date** 对象的分钟值。
- [getMonth 方法](#) (See 1.2.7.2.13) 使用当地时间返回 **Date** 对象的月份。
- [GetObject 函数](#) (See 1.2.6.1) 返回文件中的 Automation 对象的引用。
- [getSeconds 方法](#) (See 1.2.7.2.14) 使用当地时间返回 **Date** 对象的秒数。
- [getTime 方法](#) (See 1.2.7.2.15) 返回 **Date** 对象中的时间值。
- [getTimezoneOffset 方法](#) (See 1.2.7.2.16) 返回主机的时间和全球标准时间 (UTC) 之间的差 (以分钟为单位)。
- [getUTCDate 方法](#) (See 1.2.7.2.17) 使用全球标准时间 (UTC) 返回 **Date** 对象的日期值。
- [getUTCDay 方法](#) (See 1.2.7.2.18) 使用全球标准时间 (UTC) 返回 **Date** 对象的星期几的值。
- [getUTCFullYear 方法](#) (See 1.2.7.2.19) 使用全球标准时间 (UTC) 返回 **Date** 对象的年份。
- [getUTCHours 方法](#) (See 1.2.7.2.20) 使用全球标准时间 (UTC) 返回 **Date** 对象的小时数。
- [getUTCMilliseconds 方法](#) (See 1.2.7.2.21) 使用全球标准时间 (UTC) 返回 **Date** 对象的毫秒数。
- [getUTCMinutes 方法](#) (See 1.2.7.2.22) 使用全球标准时间 (UTC) 返回 **Date** 对象的分钟数。

[getUTCMonth 方法](#) (See 使用全球标准时间 (UTC) 返回 **Date** 对象的月份值。

1. 2. 7. 2. 23)

[getUTCSeconds 方法](#) (See 使用全球标准时间 (UTC) 返回 **Date** 对象的秒数。

1. 2. 7. 2. 24)

[getVarDate 方法](#) (See 返回 **Date** 对象中的 VT_DATE。

1. 2. 7. 2. 25)

[getFullYear 方法](#) (See 1. 2. 7. 2. 26) 返回 **Date** 对象中的年份。

[Global 对象](#) (See 1. 2. 8. 8) 是一个固有对象，目的是将全局方法集中在一个对象中。

[global 属性](#) (See 1. 2. 10. 7) 返回一个 Boolean 值，标记正则表达式使用的 global 标志状态 (g)。

[大于运算符 \(>\)](#) (See 1. 2. 9. 15) 比较两个表达式，看一个是否大于另一个。

[大于相等运算符 \(>=\)](#) (See 比较两个表达式，看一个是否大于等于另一个。

1. 2. 9. 15)

[hasOwnProperty 方法](#) (See 返回一个 Boolean 值，标记对象是否带有指定名称的属性。

1. 2. 10. 8)

[恒等运算符 \(===\)](#) (See 1. 2. 9. 15) 比较两个表达式，看是否值相等并具有相同的数据类型。

[if...else 语句](#) (See 1. 2. 11. 11) 根据表达式的值，有条件地执行一组语句。

[ignoreCase 属性](#) (See 1. 2. 10. 9) 返回一个 Boolean 值，标记正则表达式使用的 ignoreCase 标志状态 (i)。

[递增运算符 \(++\)](#) (See 1. 2. 9. 22) 给变量加一。

[index 属性](#) (See 1. 2. 10. 10) 返回在字符串中找到的第一个成功匹配的字符位置。

[indexOf 方法](#) (See 1. 2. 7. 2. 27) 返回在 **String** 对象中第一次出现子字符串的字符位置。

[不相等运算符 \(!=\)](#) (See 比较两个表达式，看是否不相等。

1. 2. 9. 15)

[Infinity 属性](#) (See 1. 2. 10. 11) 返回 **Number.POSITIVE_INFINITY** 的初始值。

[input 属性](#) (See 1. 2. 10. 12) 返回进行查找的字符串。

[instanceof 运算符](#) (See 返回一个 Boolean 值，表明某个对象是否为特定类的一个实例。

1. 2. 9. 23)

[isFinite 方法](#) (See 1. 2. 7. 2. 28) 返回一个 Boolean 值，表明某个给定的数是否有穷的。

[isNaN 方法](#) (See 1. 2. 7. 2. 29) 返回一个 Boolean 值，表明某个值是否为保留值 NaN (不是一个数)。

[isPrototypeOf 方法](#) (See 返回一个 Boolean 值，表明对象是否存在与另一对象的原型链中。

1. 2. 10. 13)

[italics 方法](#) (See 1. 2. 7. 2. 30) 将 HTML 的 <I> 标识添加到 **String** 对象中的文本两端。

[item 方法](#) (See 1. 2. 7. 2. 31) 返回集合中的当前项。

[join 方法](#) (See 1. 2. 7. 3. 1) 返回一个由数组中的所有元素连接在一起的 **String** 对象。

[Labeled 语句](#) (See 1. 2. 11. 12) 给语句提供一个标识符。

[lastIndex 属性](#) (See 1. 2. 10. 14) 返回在字符串中找到的最后一个成功匹配的字符位置。

[lastIndexOf 方法](#) (See 返回在 **String** 对象中最后出现子字符串的位置。

1. 2. 7. 3. 2)

- [lastMatch 属性 \(\\$\)](#) (See 2. 67) 从任何正则表达式搜索中返回最后匹配的字符。
- [lastParen 属性 \(\\$+\)](#) (See 2. 68) 从任意一个正则表达式搜索中返回最后的由括号括起的子匹配（若存在的话）。
- [lbound 方法](#) (See 1. 2. 7. 3. 3) 返回在 VArray 中指定维数所用的最小索引值。
- [leftContext 属性 \(\\$`\)](#) (See 2. 69) 返回由调用者传递给函数的实际参数个数。
- [length 属性 \(Arguments\)](#) (See 2. 70) 返回由调用者传递给函数的实际参数个数。
- [length 属性 \(Array\)](#) (See 1. 2. 10. 15) 返回比数组中所定义的最高元素大 1 的整数值。
- [length 属性 \(Function\)](#) (See 1. 2. 10. 16) 返回为函数所定义的参数个数。
- [length 属性 \(String\)](#) (See 1. 2. 10. 17) 返回 **String** 对象的长度。
- [小于运算符 \(<\)](#) (See 1. 2. 9. 15) 比较两个表达式，看是否一个小于另一个。
- [小于相等运算符 \(<=\)](#) (See 1. 2. 9. 15) 比较两个表达式，看是否一个小于等于另一个。
- [link 方法](#) (See 1. 2. 7. 3. 4) 将带 HREF 属性的 HTML 锚点添加到 **String** 对象中的文本两端。
- [LN2 属性](#) (See 1. 2. 10. 19) 返回 2 的自然对数。
- [LN10 属性](#) (See 1. 2. 10. 18) 返回 10 的自然对数。
- [localeCompare 方法](#) (See 1. 2. 7. 3. 5) 返回值表明在当前区域设置下，两个字符串是否相等。
- [log 方法](#) (See 1. 2. 7. 3. 6) 返回某个数的自然对数。
- [LOG2E 属性](#) (See 1. 2. 10. 21) 返回以 2 为底的 *e*（即 Euler 常数）的对数。
- [LOG10E 属性](#) (See 1. 2. 10. 20) 返回以 10 为底的 *e*（即 Euler 常数）的对数。
- [逻辑与运算符 \(&&\)](#) (See 1. 2. 9. 25) 对两个表达式执行逻辑与操作。
- [逻辑非运算符 \(!\)](#) (See 1. 2. 9. 26) 对表达式执行逻辑非操作。
- [逻辑或运算符 \(||\)](#) (See 1. 2. 9. 27) 对两个表达式执行逻辑或操作。
- [match 方法](#) (See 1. 2. 7. 3. 7) 使用给定的**正则表达式**对象对字符串进行查找，并将结果作为数组返回。
- [Math 对象](#) (See 1. 2. 8. 9) 一个固有对象，提供基本的数学函数和常数。
- [max 方法](#) (See 1. 2. 7. 3. 8) 返回给定的两个表达式中的较大者。
- [MAX VALUE 属性](#) (See 1. 2. 10. 22) 返回在 JScript 中能表示的最大值。
- [message 属性](#) (See 1. 2. 10. 23)
- [min 方法](#) (See 1. 2. 7. 3. 9) 返回给定的两个数中的较小者。
- [MIN VALUE 属性](#) (See 1. 2. 10. 24) 返回在 JScript 中能表示的最接近零的值。

- [取模运算符 \(%\)](#) (See 1.2.9.29) 对两个表达式执行除法运算，返回余数。
- [moveFirst 方法](#) (See 1.2.7.3.10) 将集合中的当前项设置为第一项。
- [moveNext 方法](#) (See 1.2.7.3.11) 将当前项设置为集合中的下一项。
- [multiline 属性](#) (See 1.2.10.25) 返回 Boolean 值，表明正则表达式使用的 multiline 标志 (m)。
- [乘法运算符 \(*\)](#) (See 1.2.9.31) 对两个表达式执行乘法操作。
- [name 属性](#) (See 1.4.3.23) 返回错误名称。
- [NaN 属性 \(Global\)](#) (See 1.2.10.28) 返回特殊值 NaN，表示某个表达式不是一个数。
- [NaN 属性 \(Number\)](#) (See 1.2.10.27) 返回特殊值 (NaN)，表示某个表达式不是一个数。
- [NEGATIVE_INFINITY 属性](#) (See 1.2.10.29) 返回比在 JScript 中能表示的最大的负数 (Number.MAX_VALUE) 更小的值。
- [new 运算符](#) (See 1.2.9.32) 创建一个新对象。
- [不恒等运算符 \(!=\)](#) (See 1.2.9.15) 比较两个表达式，看是否具有不相等的值或数据类型不同。
- [Number 对象](#) 表示数值数据类型和提供数值常数的对象。
- [number 属性](#) (See 1.2.10.30) 返回或设置与特定错误关联的数值。
- [Object 对象](#) (See 1.2.8.11) 提供所有的 JScript 对象的公共功能。
- [运算符优先级](#) 包含 JScript 运算符的执行优先级信息的列表。
- [parse 方法](#) (See 1.2.7.3.12) 对包含日期的字符串进行分析，并返回该日期与 1970 年 1 月 1 日零点之间相差的毫秒数。
- [parseFloat 方法](#) (See 1.2.7.3.13) 返回从字符串转换而来的浮点数。
- [parseInt 方法](#) (See 1.2.7.3.14) 返回从字符串转换而来的整数。
- [PI 属性](#) (See 1.2.10.31) 返回圆周与其直径的比值，约等于 3.141592653589793。
- [pop 方法](#) (See 1.2.7.3.15) 将数组中的最后一个元素删除，并返回其值。
- [POSITIVE_INFINITY 属性](#) (See 1.2.10.32) 返回比在 JScript 中能表示的最大的数 (-Number.MAX_VALUE) 更大的值。
- [pow 方法](#) (See 1.2.7.3.16) 返回一个指定幂次的底表达式的值。
- [propertyIsEnumerable 属性](#) (See 1.2.10.33) 返回一个 Boolean 值，表明指定的属性是否是对象的一部分或是否是可枚举的。
- [prototype 属性](#) (See 1.2.10.34) 返回对象类的原型引用。
- [push 方法](#) (See 1.2.7.3.17) 向数组中添加新的元素，返回新的数组长度。
- [random 方法](#) (See 1.2.7.3.18) 返回一个 0 和 1 之间的伪随机数。
- [RegExp 对象](#) (See 1.2.8.12) 存储有关正则表达式模式查找的信息。
- [正则表达式对象](#) (See 1.2.8.13) 包含一个正则表达式模式。
- [正则表达式语法](#) 在写正则表达式模式时可以使用的特殊字符和序列的列表。

- [replace 方法](#) (See 1. 2. 7. 3. 19) 返回根据正则表达式进行文字替换后的字符串的拷贝。
- [return 语句](#) (See 1. 2. 11. 13) 从当前函数退出并从该函数返回一个值。
- [reverse 方法](#) (See 1. 2. 7. 3. 20) 返回一个元素反序的 **Array** 对象。
- [rightContext 属性 \(\\$'\)](#) (See 2. 71) 返回被搜索字符串从最后匹配位置到结束之间的字符。
- [round 方法](#) (See 1. 2. 7. 3. 21) 将一个指定的数值表达式舍入到最近的整数值。
- [运行时错误](#) (See 1. 2. 5. 1) JScript 运行时错误列表
- [ScriptEngine 函数](#) (See 1. 2. 6. 2) 返回一个代表所使用的脚本语言的字符串。
- [ScriptEngineBuildVersion 函数](#) (See 1. 2. 6. 3) 返回所使用的脚本引擎的编译版本号。
- [ScriptEngineMajorVersion 函数](#) (See 1. 2. 6. 4) 返回所使用的脚本引擎的主版本号。
- [ScriptEngineMinorVersion 函数](#) (See 1. 2. 6. 5) 返回所使用的脚本引擎的次版本号。
- [search 方法](#) (See 1. 2. 7. 4. 1) 返回与正则表达式查找内容匹配的第一个子字符串的位置。
- [setDate 方法](#) (See 1. 2. 7. 4. 2) 使用当地时间设置 **Date** 对象的数值日期。
- [setFullYear 方法](#) (See 1. 2. 7. 4. 3) 使用当地时间设置 **Date** 对象的年份。
- [setHours 方法](#) (See 1. 2. 7. 4. 4) 使用当地时间设置 **Date** 对象的小时值。
- [setMilliseconds 方法](#) (See 1. 2. 7. 4. 5) 使用当地时间设置 **Date** 对象的毫秒值。
- [setMinutes 方法](#) (See 1. 2. 7. 4. 6) 使用当地时间设置 **Date** 对象的分钟值。
- [setMonth 方法](#) (See 1. 2. 7. 4. 7) 使用当地时间设置 **Date** 对象的月份。
- [setSeconds 方法](#) (See 1. 2. 7. 4. 8) 使用当地时间设置 **Date** 对象的秒值。
- [setTime 方法](#) (See 1. 2. 7. 4. 9) 设置 **Date** 对象的日期和时间。
- [setUTCDate 方法](#) (See 1. 2. 7. 4. 10) 使用全球标准时间 (UTC) 设置 **Date** 对象的数值日期。
- [setUTCFullYear 方法](#) (See 1. 2. 7. 4. 11) 使用全球标准时间 (UTC) 设置 **Date** 对象的年份。
- [setUTCHours 方法](#) (See 1. 2. 7. 4. 12) 使用全球标准时间 (UTC) 设置 **Date** 对象的小时值。
- [setUTCMilliseconds 方法](#) (See 1. 2. 7. 4. 13) 使用全球标准时间 (UTC) 设置 **Date** 对象的毫秒值。
- [setUTCMinutes 方法](#) (See 1. 2. 7. 4. 14) 使用全球标准时间 (UTC) 设置 **Date** 对象的分钟值。
- [setUTCMonth 方法](#) (See 1. 2. 7. 4. 15) 使用全球标准时间 (UTC) 设置 **Date** 对象的月份。
- [setUTCSeconds 方法](#) (See 1. 2. 7. 4. 15) 使用全球标准时间 (UTC) 设置 **Date** 对象的秒值。

1. 2. 7. 4. 16)

[setYear 方法](#) (See 1. 2. 7. 4. 17) 使用 **Date** 对象的年份。

[shift 方法](#) (See 1. 2. 7. 4. 18) 删除数组中的第一个元素并返回该值。

[sin 方法](#) (See 1. 2. 7. 4. 19) 返回一个数的正弦。

[slice 方法 \(Array\)](#) (See 返回数组的一个片段。

1. 2. 7. 4. 20)

[slice 方法 \(String\)](#) (See 返回字符串的一个片段。

1. 2. 7. 4. 21)

[small 方法](#) (See 1. 2. 7. 4. 22) 将 HTML 的<SMALL> 标识添加到 **String** 对象中的文本两端。

[sort 方法](#) (See 1. 2. 7. 4. 23) 返回一个元素被排序了的 **Array** 对象。

[source 属性](#) (See 1. 2. 10. 35) 返回正则表达式模式的文本的复制。

[splice 方法](#) (See 1. 2. 7. 4. 24) 从数组中删除一个元素，必要时在该位置插入一个新元素，返回被删除的元素。

[split 方法](#) (See 1. 2. 7. 4. 25) 将一个字符串分割为子字符串，然后将结果作为字符串数组返回。

[sqrt 方法](#) (See 1. 2. 7. 4. 26) 返回一个数的平方根。

[SQRT1 2 属性](#) (See 1. 2. 10. 36) 返回 0.5 的平方根，即 1 除以 2 的平方根。

[SQRT2 属性](#) (See 1. 2. 10. 37) 返回 2 的平方根。

[strike 方法](#) (See 1. 2. 7. 4. 27) 将 HTML 的<STRIKE> 标识添加到 **String** 对象中的文本两端。

[String 对象](#) (See 1. 2. 8. 14) 提供对文本字符串的操作和格式处理，判定在字符串中是否存在某个子字符串并确定其位置。

[sub 方法](#) (See 1. 2. 7. 4. 28) 将 HTML 的 <SUB> 标识放置到 **String** 对象中的文本两端。

[substr 方法](#) (See 1. 2. 7. 4. 29) 返回一个从指定位置开始并具有指定长度的子字符串。

[substring 方法](#) (See 1. 2. 7. 4. 30) 返回位于 **String** 对象中指定位置的子字符串。

[一元取相反数运算符 \(-\)](#) (See 从一个表达式的值中减去另一个表达式的值。

1. 2. 9. 35)

[sup 方法](#) (See 1. 2. 7. 4. 31) 将 HTML 的 <SUP> 标识放置到 **String** 对象中的文本两端。

[switch 语句](#) (See 1. 2. 11. 14) 当指定的表达式的值与某个标签匹配时，即执行相应的一个或多个语句。

[Syntax 错误](#) (See 1. 2. 5. 2) JScript 语法错误列表。

[tan 方法](#) (See 1. 2. 7. 5. 1) 返回一个数的正切。

[test 方法](#) (See 1. 2. 7. 5. 2) 返回一个 Boolean 值，表明在被查找的字符串中是否存在某个模式。

[this 语句](#) (See 1. 2. 11. 15) 对当前对象的引用。

[throw 语句](#) (See 1. 2. 11. 16) 产生一个可由 **try...catch** 语句处理的错误条件。

[toArray 方法](#) (See 1. 2. 7. 5. 3) 返回一个从 VArray 转换而来的标准 JScript 数组。

[toDateString 方法](#) (See 以字符串形式返回日期值。

1. 2. 7. 5. 4)

[toExponential 方法](#) (See 返回一个用指数形式表示的字符串。

1. 2. 7. 5. 5)

[toFixed 方法](#) (See 1. 2. 7. 5. 6) 返回一个字符串，表示一个用不动点记法表示的数字。

[toGMTString 方法](#) (See 返回一个转换为使用格林威治标准时间（GMT）的日期。

1. 2. 7. 5. 7)

[toLocaleDateString 方法](#) (See 返回一个与主机环境的当前区域设置相适应的字符串形式的日期。

1. 2. 7. 5. 8)

[toLocaleLowerCase 方法](#) (See 返回字符串，其中的所有字符都被转换成小写（考虑主机环境的当前设置）。

[toLocaleString 方法](#) (See 返回一个转换为使用当地时间的日期。

1. 2. 7. 5. 10)

[toLocaleTimeString 方法](#) (See 返回一个与主机环境的当前区域设置相适应的字符串形式的时间。

1. 2. 7. 5. 11)

[toLocaleUpperCase 方法](#) (See 返回字符串，其中的所有字符都被转换成大写（考虑主机环境的当前设置）。

[toLowerCase 方法](#) (See 返回一个字符串，该字符串中所有字母被转换为小写字母。

1. 2. 7. 5. 13)

[toString 方法](#) (See 1. 2. 7. 5. 15) 返回一个对象的字符串表示。

[toPrecision 方法](#) (See 返回一个字符串，该字符串包含用指定位数的指数或 fixed-point 形式表示的数字。

[getTimeString 方法](#) (See 返回以字符串形式表示的时间。

1. 2. 7. 5. 16)

[toUpperCase 方法](#) (See 返回一个字符串，该字符串中所有字母都被转换为大写字母。

1. 2. 7. 5. 17)

[toUTCString 方法](#) (See 返回一个转换为使用全球标准时间（UTC）的日期。

1. 2. 7. 5. 18)

[try 语句](#) (See 1. 2. 11. 17) 实现 JScript 的错误处理。

[typeof 运算符](#) (See 1. 2. 9. 36) 返回一个表示表达式的数据类型的字符串。

[ubound 方法](#) (See 1. 2. 7. 5. 19) 返回在 VBArray 的指定维中所使用的最大索引值。

[一元取负运算符 \(-\)](#) (See 表示一个数值表达式的相反数。

1. 2. 9. 35)

[undefined 属性](#) (See 1. 2. 10. 38) 返回 **undefined** 的初始值。

[unescape 方法](#) (See 1. 2. 7. 5. 20) 对用 **escape** 方法编码的 **String** 对象进行解码。

[unshift 方法](#) (See 1. 2. 7. 5. 21) 返回一个数组，在该数组头部插入了指定的元素。

[无符号右移运算符 \(>>>\)](#) (See 将表达式向右移位，包括符号位。

1. 2. 9. 37)

[UTC 方法](#) (See 1. 2. 7. 5. 22) 返回 1970 年 1 月 1 日零点的全球标准时间（UTC）（或 GMT）与指定日期之间的毫秒数。

[valueOf 方法](#) (See 1. 2. 7. 5. 23) 返回指定对象的原始值。

[var 语句](#) (See 1. 2. 11. 18) 声明一个变量。

VBAArray 对象 (See 1.2.8.15)	提供对 Visual Basic 安全数组的访问。
void 运算符 (See 1.2.9.39)	避免一个表达式返回值。
while 语句 (See 1.2.11.19)	执行语句直至给定的条件为 false 。
with 语句 (See 1.2.11.20)	确定一个语句的默认对象。

语言元素**描述**

GetObject 函数 (See 1.2.6.1)	返回文件中的 Automation 对象的引用。
ScriptEngine 函数 (See 1.2.6.2)	返回代表所使用的脚本语言的字符串。
ScriptEngineBuildVersion 函数 (See 1.2.6.3)	返回所使用的脚本引擎的编译版本号。
ScriptEngineMajorVersion 函数 (See 1.2.6.4)	返回所使用的脚本引擎的主版本号。
ScriptEngineMinorVersion 函数 (See 1.2.6.5)	返回所使用的脚本引擎的次版本号。

语言元素**描述**

abs 方法 (See 1.2.7.1.1)	返回一个数的绝对值。
acos 方法 (See 1.2.7.1.2)	返回一个数的反余弦。
anchor 方法 (See 1.2.7.1.3)	在对象的指定文本两端加上一个带 NAME 属性的 HTML 锚点。
apply 方法 (See 1.2.7.1.4)	应用对象的一个方法，用当前对象代替另一对象。
asin 方法 (See 1.2.7.1.5)	返回一个数的反正弦。
atan 方法 (See 1.2.7.1.6)	返回一个数的反正切。
atan2 方法 (See 1.2.7.1.7)	返回从 X 轴到点 (y, x) 的角度 (以弧度为单位)。
atEnd 方法 (See 1.2.7.1.8)	返回一个表明枚举算子是否处于集合结束处的 Boolean 值。
big 方法 (See 1.2.7.1.9)	在 String 对象的文本两端加入 HTML 的<BIG>标识。
blink 方法 (See 1.2.7.1.10)	将 HTML 的 <BLINK> 标识添加到 String 对象中的文本两端。
bold 方法 (See 1.2.7.1.11)	将 HTML 的 标识添加到 String 对象中的文本两端。
call 方法 (See 1.2.7.1.12)	应用对象的一个方法，用当前对象代替另一对象。
ceil 方法 (See 1.2.7.1.13)	返回大于或等于其数值参数的最小整数。
charAt 方法 (See 1.2.7.1.14)	返回位于指定索引位置的字符。
charCodeAt 方法 (See 1.2.7.1.15)	返回指定字符的 Unicode 编码。
compile 方法 (See 1.2.7.1.16)	将一个正则表达式编译为内部格式。
concat 方法 (Array) (See 1.2.7.1.17)	返回一个由两个数组合并组成的新数组。
concat 方法 (String) (See 1.2.7.1.18)	返回一个包含给定的两个字符串的连接的 String 对象。
cos 方法 (See 1.2.7.1.19)	返回一个数的余弦。
decodeURI 方法 (See 1.2.7.1.20)	返回一个已编码的通用资源标识符 (URI) 的解码版。

[decodeURIComponent 方法](#) (See 返回一个已编码的通用资源标识符 (URI) 的解码版。

1. 2. 7. 1. 21)

[dimensions 方法](#) (See 返回 VArray 的维数。

1. 2. 7. 1. 22)

[escape 方法](#) (See 1. 2. 7. 1. 25) 对 **String** 对象编码, 以便在所有计算机上都能阅读。

[eval 方法](#) (See 1. 2. 7. 1. 26) 对 JScript 代码求值然后执行之。

[exec 方法](#) (See 1. 2. 7. 1. 27) 在指定字符串中执行一个匹配查找。

[exp 方法](#) (See 1. 2. 7. 1. 28) 返回 *e* (自然对数的底) 的幂。

[fixed 方法](#) (See 1. 2. 7. 2. 1) 将 HTML 的<TT> 标识添加到 **String** 对象中的文本两端。

[floor 方法](#) (See 1. 2. 7. 2. 2) 返回小于或等于其数值参数的最大整数。

[fontcolor 方法](#) (See 1. 2. 7. 2. 3) 将 HTML 带 COLOR 属性的标识添加到 **String** 对象中的文本两端。

[fontsize 方法](#) (See 1. 2. 7. 2. 4) 将 HTML 带 SIZE 属性的标识添加到 **String** 对象中的文本两端。

[fromCharCode 方法](#) (See 返回 Unicode 字符值的字符串。

1. 2. 7. 2. 5)

[getDate 方法](#) (See 1. 2. 7. 2. 6) 使用当地时间返回 **Date** 对象的月份日期值。

[getDay 方法](#) (See 1. 2. 7. 2. 7) 使用当地时间返回 **Date** 对象的星期几。

[getFullYear 方法](#) (See 使用当地时间返回 **Date** 对象的年份。

1. 2. 7. 2. 8)

[getHours 方法](#) (See 1. 2. 7. 2. 9) 使用当地时间返回 **Date** 对象的小时值。

[getItem 方法](#) (See 1. 2. 7. 2. 10) 返回位于指定位置的项。

[getMilliseconds 方法](#) (See 使用当地时间返回 **Date** 对象的毫秒值。

1. 2. 7. 2. 11)

[getMinutes 方法](#) (See 使用当地时间返回 **Date** 对象的分钟值。

1. 2. 7. 2. 12)

[getMonth 方法](#) (See 1. 2. 7. 2. 13) 使用当地时间返回 **Date** 对象的月份。

[getSeconds 方法](#) (See 使用当地时间返回 **Date** 对象的秒数。

1. 2. 7. 2. 14)

[getTime 方法](#) (See 1. 2. 7. 2. 15) 返回 **Date** 对象中的时间。

[getTimezoneOffset 方法](#) (See 返回主机的时间和全球标准时间 (UTC) 之间的差 (以分钟为单位)。

1. 2. 7. 2. 16)

[getUTCDate 方法](#) (See 使用全球标准时间 (UTC) 返回 **Date** 对象的日期值。

1. 2. 7. 2. 17)

[getUTCDay 方法](#) (See 1. 2. 7. 2. 18) 使用全球标准时间 (UTC) 返回 **Date** 对象的星期几。

[getUTCFullYear 方法](#) (See 使用全球标准时间 (UTC) 返回 **Date** 对象的年份。

1. 2. 7. 2. 19)

[getUTCHours 方法](#) (See 使用全球标准时间 (UTC) 返回 **Date** 对象的小时数。

1. 2. 7. 2. 20)

[getUTCMilliseconds 方法](#) (See 使用全球标准时间 (UTC) 返回 **Date** 对象的毫秒数。

1. 2. 7. 2. 21)

[getUTCMinutes 方法](#) (See 使用全球标准时间 (UTC) 返回 **Date** 对象的分钟数。

1. 2. 7. 2. 22)

[getUTCMonth 方法](#) (See 使用全球标准时间 (UTC) 返回 **Date** 对象的月份值。

1. 2. 7. 2. 23)

[getUTCSeconds 方法](#) (See 使用全球标准时间 (UTC) 返回 **Date** 对象的秒数。

1. 2. 7. 2. 24)

[getVarDate 方法](#) (See 返回 **Date** 对象中的 VT_DATE。

1. 2. 7. 2. 25)

[getFullYear 方法](#) (See 1. 2. 7. 2. 26) 返回 **Date** 对象中的年份。

[hasOwnProperty 方法](#) (See 返回一个 Boolean 值, 表明对象是否具有指定的名称。

1. 2. 10. 8)

[indexOf 方法](#) (See 1. 2. 7. 2. 27) 返回在 **String** 对象中第一次出现子字符串的字符位置。

[isFinite 方法](#) (See 1. 2. 7. 2. 28) 返回一个 Boolean 值, 表明某个给定的数是否有穷的。

[isNaN 方法](#) (See 1. 2. 7. 2. 29) 返回一个 Boolean 值, 表明某个值是否为保留值 **NaN** (不是一个数)。

[isPrototypeOf 方法](#) (See 返回一个 Boolean 值, 表明对象是否存在于另一对象的原型链中。

1. 2. 10. 13)

[italics 方法](#) (See 1. 2. 7. 2. 30) 将 HTML 的 <I> 标识添加到 **String** 对象中的文本两端。

[item 方法](#) (See 1. 2. 7. 2. 31) 返回集合中的当前项。

[join 方法](#) (See 1. 2. 7. 3. 1) 返回一个由数组中的所有元素连接在一起的 **String** 对象。

[lastIndexOf 方法](#) (See 返回在 **String** 对象中子字符串最后出现的位置。

1. 2. 7. 3. 2)

[lbound 方法](#) (See 1. 2. 7. 3. 3) 返回在 VBAArray 中指定维数所用的最小索引值。

[link 方法](#) (See 1. 2. 7. 3. 4) 将带 HREF 属性的 HTML 锚点添加到 **String** 对象中的文本两端。

[localeCompare 方法](#) (See 返回一个值, 表明两个字符串在当前区域设置下是否相等。

1. 2. 7. 3. 5)

[log 方法](#) (See 1. 2. 7. 3. 6) 返回某个数的自然对数。

[match 方法](#) (See 1. 2. 7. 3. 7) 使用给定的**正则表达式**对象对字符串进行查找, 并将结果作为数组返回。

[max 方法](#) (See 1. 2. 7. 3. 8) 返回给定的两个表达式中的较大者。

[min 方法](#) (See 1. 2. 7. 3. 9) 返回给定的两个数中的较小者。

[moveFirst 方法](#) (See 1. 2. 7. 3. 10) 将集合中的当前项设置为第一项。

[moveNext 方法](#) (See 1. 2. 7. 3. 11) 将当前项设置为集合中的下一项。

[parse 方法](#) (See 1. 2. 7. 3. 12) 对包含日期的字符串进行分析, 并返回该日期与 1970 年 1 月 1 日零点之间相差的毫秒数。

[parseFloat 方法](#) (See 返回从字符串转换而来的浮点数。

1. 2. 7. 3. 13)

[parseInt 方法](#) (See 1. 2. 7. 3. 14) 返回从字符串转换而来的整数。

[pop 方法](#) (See 1. 2. 7. 3. 15) 删除数组中的最后一个元素并返回该值。

[pow 方法](#) (See 1. 2. 7. 3. 16) 返回一个指定幂次的底表达式的值。

[push 方法](#) (See 1. 2. 7. 3. 17) 向数组中添加新元素，返回数组的新长度。

[random 方法](#) (See 1. 2. 7. 3. 18) 返回一个 0 和 1 之间的伪随机数。

[replace 方法](#) (See 1. 2. 7. 3. 19) 返回根据正则表达式进行文字替换后的字符串的拷贝。

[reverse 方法](#) (See 1. 2. 7. 3. 20) 返回一个元素反序的 **Array** 对象。

[round 方法](#) (See 1. 2. 7. 3. 21) 将一个指定的数值表达式舍入到最近的整数并将其返回。

[search 方法](#) (See 1. 2. 7. 4. 1) 返回与正则表达式查找内容匹配的第一个子字符串的位置。

[setDate 方法](#) (See 1. 2. 7. 4. 2) 使用当地时间设置 **Date** 对象的数值日期。

[setFullYear 方法](#) (See 使用当地时间设置 **Date** 对象的年份。

1. 2. 7. 4. 3)

[setHours 方法](#) (See 1. 2. 7. 4. 4) 使用当地时间设置 **Date** 对象的小时值。

[setMilliseconds 方法](#) (See 使用当地时间设置 **Date** 对象的毫秒值。

1. 2. 7. 4. 5)

[setMinutes 方法](#) (See 1. 2. 7. 4. 6) 使用当地时间设置 **Date** 对象的分钟值。

[setMonth 方法](#) (See 1. 2. 7. 4. 7) 使用当地时间设置 **Date** 对象的月份。

[setSeconds 方法](#) (See 1. 2. 7. 4. 8) 使用当地时间设置 **Date** 对象的秒值。

[setTime 方法](#) (See 1. 2. 7. 4. 9) 设置 **Date** 对象的日期和时间。

[setUTCDate 方法](#) (See 使用全球标准时间 (UTC) 设置 **Date** 对象的数值日期。

1. 2. 7. 4. 10)

[setUTCFullYear 方法](#) (See 使用全球标准时间 (UTC) 设置 **Date** 对象的年份。

1. 2. 7. 4. 11)

[setUTCHours 方法](#) (See 使用全球标准时间 (UTC) 设置 **Date** 对象的小时值。

1. 2. 7. 4. 12)

[setUTCMilliseconds 方法](#) (See 使用全球标准时间 (UTC) 设置 **Date** 对象的毫秒值。

1. 2. 7. 4. 13)

[setUTCMinutes 方法](#) (See 使用全球标准时间 (UTC) 设置 **Date** 对象的分钟值。

1. 2. 7. 4. 14)

[setUTCMonth 方法](#) (See 使用全球标准时间 (UTC) 设置 **Date** 对象的月份。

1. 2. 7. 4. 15)

[setUTCSeconds 方法](#) (See 使用全球标准时间 (UTC) 设置 **Date** 对象的秒值。

1. 2. 7. 4. 16)

[setYear 方法](#) (See 1. 2. 7. 4. 17) 使用 **Date** 对象的年份。

[shift 方法](#) (See 1. 2. 7. 4. 18) 删除数组中的第一个元素并返回该值。

[sin 方法](#) (See 1. 2. 7. 4. 19) 返回一个数的正弦。

[slice 方法 \(Array\)](#) (See 返回数组的一个片段。

1. 2. 7. 4. 20)

[slice 方法 \(String\)](#) (See 返回字符串的一个片段。

1. 2. 7. 4. 21)

[small 方法](#) (See 1. 2. 7. 4. 22) 将 HTML 的<SMALL> 标识添加到 **String** 对象中的文本两端。

[sort 方法](#) (See 1. 2. 7. 4. 23) 返回一个元素被排序了的 **Array** 对象。

[splice 方法](#) (See 1. 2. 7. 4. 24) 从数组中删除元素，若必要，在相应位置处插入新元素，返回被删除的元素。

[split 方法](#) (See 1. 2. 7. 4. 25) 将一个字符串分割为子字符串，然后将结果作为字符串数组返回。

[sqrt 方法](#) (See 1. 2. 7. 4. 26) 返回一个数的平方根。

[strike 方法](#) (See 1. 2. 7. 4. 27) 将 HTML 的<STRIKE> 标识添加到 **String** 对象中的文本两端。

[sub 方法](#) (See 1. 2. 7. 4. 28) 将 HTML 的 <SUB> 标识放置到 **String** 对象中的文本两端。

[substr 方法](#) (See 1. 2. 7. 4. 29) 返回一个从指定位置开始并具有指定长度的子字符串。

[substring 方法](#) (See 1. 2. 7. 4. 30) 返回位于 **String** 对象中指定位置的子字符串。

[sup 方法](#) (See 1. 2. 7. 4. 31) 将 HTML 的 <SUP> 标识放置到 **String** 对象中的文本两端。

[tan 方法](#) (See 1. 2. 7. 5. 1) 返回一个数的正切。

[test 方法](#) (See 1. 2. 7. 5. 2) 返回一个 Boolean 值，表明在被查找的字符串中是否存在某个模式。

[toArray 方法](#) (See 1. 2. 7. 5. 3) 返回一个从 VArray 转换而来的标准 JScript 数组。

[toDateString 方法](#) (See 返回以字符串形式表示的日期。

1. 2. 7. 5. 4)

[toExponential 方法](#) (See 返回一个字符串，该字符串包含一个以指数形式表示的数字。

1. 2. 7. 5. 5)

[toFixed 方法](#) (See 1. 2. 7. 5. 6) 返回一个字符串，表明一个用 fixed-point 形式表示的数字。

[toGMTString 方法](#) (See 返回一个转换为使用格林威治标准时间 (GMT) 的字符串的日期。

1. 2. 7. 5. 7)

[toLocaleDateString 方法](#) (See 返回一个以字符串形式表示的日期，该日期与主机环境的当前区域设置相适应。

[toLocaleLowerCase 方法](#) (See 返回一个字符串，其中所有的字母字符都被转换成小写 (考虑主机环境的当前区域设置)。

[toLocaleTimeString 方法](#) (See 返回一个以字符串形式表示的时间，该时间与主机环境当前区域设置相适应。

[toLocaleString 方法](#) (See 返回一个转换为使用当地时间的字符串的日期。

1. 2. 7. 5. 10)

[toLocaleUppercase 方法](#) (See 返回一个字符串，其中所有的字母字符都被转换成大写 (考虑主机环境的当前区域设置)。

[toLowerCase 方法](#) (See 返回一个所有的字母字符都被转换为小写字母的字符串。

1. 2. 7. 5. 13)

[toPrecision 方法](#) (See 返回一个字符串，该字符串包含用指定位数的指数或 fixed-point 形式表示的数字。

1. 2. 7. 5. 14)

[toString 方法](#) (See 1.2.7.5.15) 返回一个对象的字符串表示。

[getTimeString 方法](#) (See 1.2.7.5.16) 返回一个以字符串形式表示的时间。

[toUpperCase 方法](#) (See 1.2.7.5.17) 返回一个所有的字母字符都被转换为大写字母的字符串。

[toUTCString 方法](#) (See 1.2.7.5.18) 返回一个转换为使用全球标准时间 (UTC) 的字符串的日期。

[ubound 方法](#) (See 1.2.7.5.19) 返回在 VArray 的指定维中所使用的最大索引值。

[unescape 方法](#) (See 1.2.7.5.20) 对用 **escape** 方法编码的 **String** 对象进行解码。

[unshift 方法](#) (See 1.2.7.5.21) 返回一个数组，在该数组头部插入了指定的元素。

[UTC 方法](#) (See 1.2.7.5.22) 返回 1970 年 1 月 1 日零点的全球标准时间 (UTC) (或 GMT) 与指定日期之间的毫秒数。

[valueOf 方法](#) (See 1.2.7.5.23) 返回指定对象的原始值。

语言元素

描述

[ActiveXObject 对象](#) (See 1.2.8.1) 启用并返回一个 Automation 对象的引用。

[Array 对象](#) (See 1.2.8.2) 提供对创建任何数据类型的数组的支持。

[Boolean 对象](#) (See 1.2.8.3) 创建一个新的 Boolean 值。

[Date 对象](#) (See 1.2.8.4) 提供日期和时间的基本存储和检索。

[Dictionary 对象](#) (See 1.4.2.1) 存储数据键、项对的对象。

[Enumerator 对象](#) (See 1.2.8.5) 提供集合中的项的枚举。

[Error 对象](#) (See 1.2.8.6) 包含在运行 JScript 代码时发生的错误的有关信息。

[FileSystemObject 对象](#) (See 1.4.2.6) 提供对计算机文件系统的访问。

[Function 对象](#) (See 1.2.8.7) 创建一个新的函数。

[Global 对象](#) (See 1.2.8.8) 是一个内部对象，目的是将全局方法集中在一个对象中。

[Math 对象](#) (See 1.2.8.9) 一个内部对象，提供基本的数学函数和常数。

[Number 对象](#) (See 1.2.8.10) 表示数值数据类型和提供数值常数的对象。

[Object 对象](#) (See 1.2.8.11) 提供所有的 JScript 对象的公共功能。

[RegExp 对象](#) (See 1.2.8.12) 存储有关正则表达式模式查找的信息。

[正则表达式对象](#) (See 1.2.8.13) 包含一个正则表达式模式。

[String 对象](#) (See 1.2.8.14) 提供对文本字符串的操作和格式处理，判定在字符串中是否存在某个子字符串及确定其位置。

[VArray 对象](#) (See 1.2.8.15) 提供对 Visual Basic 安全数组的访问。

语言元素

描述

[! 运算符](#) (See 1.2.9.26) 对表达式执行逻辑“非”运算。

[!= 运算符](#) (See 1.2.9.15) 比较两个表达式的值，看其是否相等。

!= 运算符 (See 1.2.9.15)	比较两个表达式的值，看其是否不相等或数据类型不一致。
% 运算符 (See 1.2.9.29)	对两个表达式执行除法运算，返回余数。
%= 运算符 (See 1.2.9.28)	对变量和表达式执行除法运算，余数赋给变量。
& 运算符 (See 1.2.9.6)	对两个表达式执行按位“与”运算。
&= 运算符 (See 1.2.9.5)	对变量和表达式执行按位“与”运算，结果赋给变量。
&& 运算符 (See 1.2.9.25)	对两个表达式执行逻辑连接运算。
* 运算符 (See 1.2.9.31)	对两个表达式执行乘法运算。
*= 运算符 (See 1.2.9.30)	对变量和表达式执行乘法运算，结果赋给变量。
+ 运算符 (See 1.2.9.3)	对两个数值表达式求和，或连接两个字符串。
++ 运算符 (See 1.2.9.22)	变量值加一。
+= 运算符 (See 1.2.9.2)	将变量和表达式的值相加，结果赋给变量。
, 运算符 (See 1.2.9.14)	使两个表达式按顺序执行。
- 运算符 (See 1.2.9.35)	从一个表达式的值减去另一个表达式的值或对一个表达式执行取负运算。
-- 运算符 (See 1.2.9.22)	变量值减一。
-= 运算符 (See 1.2.9.35)	从变量值中减表达式的值，结果赋给变量。
/ 运算符 (See 1.2.9.20)	对两个表达式执行除法运算。
/= 运算符 (See 1.2.9.19)	对变量和表达式执行除法运算，结果赋给变量。
< 运算符 (See 1.2.9.15)	比较一个表达式是否小于另一个表达式。
<< 运算符 (See 1.2.9.7)	将表达式向左移位。
<<= 运算符 (See 1.2.9.24)	将变量的值左移由表达式指定的位数，结果赋给变量。
<= 运算符 (See 1.2.9.15)	比较一个表达式的值是否小于等于另一个表达式的值。
= 运算符 (See 1.2.9.4)	为变量赋值。
== 运算符 (See 1.2.9.15)	比较两个表达式是否相等。
=== 运算符 (See 1.2.9.15)	比较两个表达式，看其值是否相等或数据类型是否一致。
> 运算符 (See 1.2.9.15)	比较一个表达式的值是否大于另一表达式。
>= 运算符 (See 1.2.9.15)	比较一个表达式的值是否大于等于另一表达式。
>> 运算符 (See 1.2.9.11)	表达式向右移位，符号位不变。
>>= 运算符 (See 1.2.9.33)	将变量的值左移由表达式指定的位数，符号位不变，结果赋给变量。
>>> 运算符 (See 1.2.9.37)	表达式向右移位，包括符号位。
>>>= 运算符 (See 1.2.9.38)	将变量的值左移由表达式指定的位数，包括符号位，结果赋给变量。
?: 运算符 (See 1.2.9.17)	根据条件执行其中一个语句。
~ 运算符 (See 1.2.9.8)	对表达式执行按位“非”（取反）运算。
 运算符 (See 1.2.9.10)	对两个表达式执行按位“或”运算。
 = 运算符 (See 1.2.9.9)	对变量和表达式的值执行按位“或”运算，结果赋给变量。
 运算符 (See 1.2.9.27)	对两个表达式执行逻辑或运算。

- [^ 运算符](#) (See 1. 2. 9. 13) 对两个表达式执行按位异或运算。
- [^= 运算符](#) (See 1. 2. 9. 12) 对变量和表达式的值执行按位异或运算，结果赋给变量。
- [加法运算符 \(+\)](#) (See 1. 2. 9. 3) 求两个数值表达式的和，或连接两个字符串。
- [赋值运算符 \(=\)](#) (See 1. 2. 9. 4) 将一个值赋给变量。
- [按位与运算符 \(&\)](#) (See 1. 2. 9. 6) 对两个表达式执行按位与操作。
- [按位左移运算符 \(<<\)](#) (See 1. 2. 9. 7) 将一个表达式的各位向左移。
- [按位取非运算符 \(~\)](#) (See 1. 2. 9. 8) 对一个表达式执行按位取非（求非）操作。
- [按位或运算符 \(|\)](#) (See 1. 2. 9. 10) 对两个表达式指定按位或操作。
- [按位右移运算符 \(>>\)](#) (See 1. 2. 9. 11) 将一个表达式的各位向右移，保持符号不变。
- [按位异或运算符 \(^\)](#) (See 1. 2. 9. 13) 对两个表达式执行按位异或操作。
- [逗号运算符 \(,\)](#) (See 1. 2. 9. 14) 使两个表达式连续执行。
- [比较运算符](#) (See 1. 2. 9. 15) 返回 Boolean 值，表示比较结果。
- [复合赋值运算符](#) 复合赋值运算符列表。
- [条件（三元）运算符 \(?:\)](#) (See 1. 2. 9. 17) 根据条件执行两个表达式之一。
- [递减运算符 \(--\)](#) (See 1. 2. 9. 22) 将变量减一。
- [delete 运算符](#) (See 1. 2. 9. 18) 删除对象的属性，或删除数组中的一个元素。
- [除法运算符 \(/\)](#) (See 1. 2. 9. 20) 对两个表达式执行除法运算。
- [相等运算符 \(==\)](#) (See 1. 2. 9. 15) 比较两个表达式，看是否相等。
- [大于运算符 \(>\)](#) (See 1. 2. 9. 15) 比较两个表达式，看一个是否大于另一个。
- [大于相等运算符 \(>=\)](#) (See 1. 2. 9. 15) 比较两个表达式，看一个是否大于等于另一个。
- [恒等运算符 \(===\)](#) (See 1. 2. 9. 15) 比较两个表达式，看是否值相等并具有相同的数据类型。
- [递增运算符 \(++\)](#) (See 1. 2. 9. 22) 给变量加一。
- [不相等运算符 \(!=\)](#) (See 1. 2. 9. 15) 比较两个表达式，看是否不相等。
- [instanceof 运算符](#) (See 1. 2. 9. 23) 返回一个 Boolean 值，表明某个对象是否为特定类的一个实例。
- [小于运算符 \(<\)](#) (See 1. 2. 9. 15) 比较两个表达式，看是否一个小于另一个。
- [小于相等运算符 \(<=\)](#) (See 1. 2. 9. 15) 比较两个表达式，看是否一个小于等于另一个。
- [逻辑与运算符 \(&&\)](#) (See 1. 2. 9. 25) 对两个表达式执行逻辑与操作。
- [逻辑非运算符 \(!\)](#) (See 1. 2. 9. 26) 对表达式执行逻辑非操作。

[逻辑或运算符 \(||\)](#) (See 对两个表达式执行逻辑或操作。

1. 2. 9. 27)

[取模运算符 \(%\)](#) (See 1. 2. 9. 29) 对两个表达式执行除法运算，返回余数。

[乘法运算符 \(*\)](#) (See 1. 2. 9. 31) 对两个表达式执行乘法操作。

[new 运算符](#) (See 1. 2. 9. 32) 创建一个新对象。

[不恒等运算符 \(!=\)](#) (See 比较两个表达式，看是否具有不相等的值或数据类型不同。

1. 2. 9. 15)

[运算符优先级](#) 包含 JScript 运算符的执行优先级信息的列表。

[减法运算符 \(-\)](#) (See 1. 2. 9. 35) 求两个表达式值的差。

[typeof 运算符](#) (See 1. 2. 9. 36) 返回一个表示表达式的数据类型的字符串。

[一元取相反数运算符 \(-\)](#) (See 表示一个数值表达式的相反数。

1. 2. 9. 35)

[无符号右移运算符 \(>>>\)](#) (See 表达式向右移位，包括符号位。

1. 2. 9. 37)

[void 运算符](#) (See 1. 2. 9. 39) 避免一个表达式返回值。

语言元素

描述

[0...n 属性](#) (See 2. 65) 返回单个参数的实际值，该参数来自当前运行函数的**参数**属性返回的**参数**对象。

[\\$1...\\$9 属性](#) (See 1. 2. 10. 1) 返回在模式匹配中找到的最近的九条记录

[arguments 属性](#) (See 1. 2. 10. 2) 返回一个包含传递给当前执行函数的每个参数的数组。

[caller 属性](#) (See 1. 2. 10. 3) 返回调用当前函数的函数引用。

[callee 属性](#) (See 2. 66) 返回正执行的**函数**对象，它是指定的**函数**对象的文本正文。

[constructor 属性](#) (See 1. 2. 10. 4) 指定创建对象的函数。

[description 属性](#) (See 1. 2. 10. 5) 返回或设置关于指定错误的描述字符串。

[E 属性](#) (See 1. 2. 10. 6) 返回 Euler 常数，即自然对数的底。

[global 属性](#) (See 1. 2. 10. 7) 返回一个 Boolean 值，表明正则表达式使用的 global 标志 (g) 状态。

[ignoreCase 属性](#) (See 1. 2. 10. 9) 返回一个 Boolean 值，表明正则表达式使用的 ignoreCase 标志 (i) 状态。

[index 属性](#) (See 1. 2. 10. 10) 返回在字符串中找到的第一个成功匹配的字符位置。

[Infinity 属性](#) (See 1. 2. 10. 11) 返回 **Number.POSITIVE_INFINITY** 的初始值。

[input 属性](#) (See 1. 2. 10. 12) 返回进行查找的字符串。

[lastIndex 属性](#) (See 1. 2. 10. 14) 返回在字符串中找到的最后一个成功匹配的字符位置。

[lastMatch 属性 \(\\$\)](#) (See 2. 67) 返回任意正则表达式搜索中最后匹配的字符。

[lastParen 属性 \(\\$+\)](#) (See 2. 68) 从任意一个正则表达式搜索中返回最后的由括号括起的子匹配（若存在的话）。

[leftContext 属性 \(\\$`\)](#) (See 返回由调用者传递给函数的实际参数个数。

2. 69)

- [length 属性 \(Arguments\)](#) (See 返回由调用者传递给函数的实际参数个数。
2. 70)
- [length 属性 \(Array\)](#) (See 返回比数组中所定义的最高元素大 1 的一个整数。
1. 2. 10. 15)
- [length 属性 \(Function\)](#) (See 返回为函数所定义的参数个数。
1. 2. 10. 16)
- [length 属性 \(String\)](#) (See 返回 **String** 对象的长度。
1. 2. 10. 17)
- [LN2 属性](#) (See 1. 2. 10. 19) 返回 2 的自然对数。
- [LN10 属性](#) (See 1. 2. 10. 18) 返回 10 的自然对数。
- [LOG2E 属性](#) (See 1. 2. 10. 21) 返回以 2 为底的 e (即 Euler 常数) 的对数。
- [LOG10E 属性](#) (See 1. 2. 10. 20) 返回以 10 为底的 e (即 Euler 常数) 的对数。
- [MAX_VALUE 属性](#) (See 1. 2. 10. 22) 返回在 JScript 中能表示的最大值。
- [message 属性](#) (See 1. 2. 10. 23) 返回错误消息串。
- [MIN_VALUE 属性](#) (See 1. 2. 10. 24) 返回在 JScript 中能表示的最接近零的值。
- [multiline 属性](#) (See 1. 2. 10. 25) 返回一个 Boolean 值, 表明正则表达式使用的 multiline 标志 (m) 状态。
- [name 属性](#) (See 1. 4. 3. 23) 返回错误名称。
- [NaN 属性 \(Global\)](#) (See 返回特殊值 **NaN**, 表示某个表达式不是一个数。
1. 2. 10. 28)
- [NaN 属性 \(Number\)](#) (See 返回特殊值 (**NaN**), 表示某个表达式不是一个数。
1. 2. 10. 27)
- [NEGATIVE_INFINITY 属性](#) (See 返回比在 JScript 中能表示的最大的负数 (-Number.MAX_VALUE) 更负
1. 2. 10. 29) 的值。
- [number 属性](#) (See 1. 2. 10. 30) 返回或设置与特定错误关联的数值。
- [PI 属性](#) (See 1. 2. 10. 31) 返回圆周与其直径的比值, 约等于 3. 141592653589793。
- [POSITIVE_INFINITY 属性](#) (See 返回比在 JScript 中能表示的最大的数 (Number.MAX_VALUE) 更大的
1. 2. 10. 32) 值。
- [propertyIsEnumerable 属性](#) (See 返回一个 Boolean 值, 表明指定的属性是否是对象的一部分或是否是
1. 2. 10. 33) 可枚举的。
- [prototype 属性](#) (See 1. 2. 10. 34) 返回对象类的原型引用。
- [source 属性](#) (See 1. 2. 10. 35) 返回正则表达式模式的文本的拷贝。
- [rightContext 属性 \(\\$'\)](#) (See 返回被搜索字符串从最后匹配位置到结束之间的字符。
2. 71)
- [SQRT1_2 属性](#) (See 1. 2. 10. 36) 返回 0.5 的平方根, 即 1 除以 2 的平方根。
- [SQRT2 属性](#) (See 1. 2. 10. 37) 返回 2 的平方根。
- [undefined 属性](#) (See 1. 2. 10. 38) 返回 **undefined** 的初始值。

语言元素

描述

break 语句 (See 1.2.11.4)	终止当前循环, 或者如果与一个 <i>label</i> 语句关联, 则终止相关联的语句。
catch 语句 (See 1.2.11.17)	包含在 try 语句块中的代码发生错误时执行的语句。
@cc on 语句 (See 1.2.11.1)	激活条件编译支持。
//(单行注释语句) (See 1.2.11.5)	使单行注释被 JScript 语法分析器忽略。
/*.*/(多行注释语句) (See 1.2.11.5)	使多行注释被 JScript 语法分析器忽略。
continue 语句 (See 1.2.11.6)	停止循环的当前迭代, 并开始一次新的迭代。
do...while 语句 (See 1.2.11.7)	先执行一次语句块, 然后重复执行该循环, 直至条件表达式的值为 false 。
for 语句 (See 1.2.11.8)	只要指定的条件为 true , 就一直执行语句块。
for...in 语句 (See 1.2.11.9)	对应于对象或数组中的每个元素执行一个或多个语句。
function 语句 (See 1.2.11.10)	声明一个新的函数。
@if 语句 (See 1.2.11.2)	根据表达式的值, 有条件地执行一组语句。
if...else 语句 (See 1.2.11.11)	根据表达式的值, 有条件地执行一组语句。
Labeled 语句 (See 1.2.11.12)	给语句提供一个标识符。
return 语句 (See 1.2.11.13)	从当前函数退出并从该函数返回一个值。
@set 语句 (See 1.2.11.3)	创建用于条件编译语句的变量。
switch 语句 (See 1.2.11.14)	当指定的表达式的值与某个标签匹配时, 即执行相应的一个或多个语句。
this 语句 (See 1.2.11.15)	对当前对象的引用。
throw 语句 (See 1.2.11.16)	产生一个可由 try...catch 语句处理的错误条件。
try 语句 (See 1.2.11.17)	实现 JScript 的错误处理。
var 语句 (See 1.2.11.18)	声明一个变量。
while 语句 (See 1.2.11.19)	执行语句直至给定的条件为 false 。
with 语句 (See 1.2.11.20)	确定一个语句的默认对象。
语言元素	描述
运行时错误 (See 1.2.5.1)	JScript 运行时错误列表
语法错误 (See 1.2.5.2)	JScript 语法错误列表

1.2.1 特性信息

描述	语言要素
JScript 按宿主应用的版本列表和按版本的特性列表。	版本信息 (See 2.64)
JScript 目前的 ECMA 特性列表。	JScript 特性 (ECMA) (See 1.2.2)
JScript 目前的非-ECMA 特性列表。	JScript 特性 (非-ECMA) (See 1.2.3)

1.2.2 Microsoft JScript 特性 - ECMA

种类	特性/关键字
数组处理	Array (See 1.2.8.2), concat (See 1.2.7.1.17), join (See 1.2.7.3.1), length (See 1.2.10.15), reverse (See 1.2.7.3.20), slice (See 1.2.7.4.20) sort (See 1.2.7.4.23)
赋值	赋值 (=) (See 1.2.9.4) 复合赋值(OP=) (See 1.2.9.16)
Boolean	Boolean (See 1.2.8.3)
注释	/*...*/ 或 // (See 1.2.11.5)_
常数/文字	NaN (See 1.2.10.28) null true, false Infinity (See 1.2.10.11) undefined
控制流	Break (See 1.2.11.4) continue (See 1.2.11.6) do...while (See 1.2.11.7) for (See 1.2.11.8) for...in (See 1.2.11.9) if...else (See 1.2.11.11) Labeled (See 1.2.11.12)_ return (See 1.2.11.13) switch (See 1.2.11.14) while (See 1.2.11.19)
日期和时间	Date (See 1.2.8.4) getDate (See 1.2.7.2.6), getDay (See 1.2.7.2.7), getFullYear (See 1.2.7.2.8), getHours (See 1.2.7.2.9), getMilliseconds (See 1.2.7.2.11), getMinutes (See 1.2.7.2.12), getMonth (See 1.2.7.2.13), getSeconds (See 1.2.7.2.14), getTime (See 1.2.7.2.15), getTimezoneOffset (See 1.2.7.2.16), getYear (See 1.2.7.2.26), getUTCDate (See 1.2.7.2.17), getUTCDay (See 1.2.7.2.18), getUTCFullYear (See 1.2.7.2.19), getUTCHours (See 1.2.7.2.20), getUTCMilliseconds (See 1.2.7.2.21), getUTCMinutes (See 1.2.7.2.22), getUTCMonth (See 1.2.7.2.23), getUTCSeconds (See 1.2.7.2.24), setDate (See 1.2.7.4.2), setFullYear (See 1.2.7.4.3), setHours (See 1.2.7.4.4), setMilliseconds (See 1.2.7.4.5), setMinutes (See 1.2.7.4.6), setMonth (See 1.2.7.4.7), setSeconds (See 1.2.7.4.8), setTime (See 1.2.7.4.9), setYear (See 1.2.7.4.17), setUTCDate (See 1.2.7.4.10), setUTCFullYear (See 1.2.7.4.11),

	setUTCHours (See 1.2.7.4.12), setUTCMilliseconds (See 1.2.7.4.13), setUTCMinutes (See 1.2.7.4.14), setUTCMonth (See 1.2.7.4.15), setUTCSeconds (See 1.2.7.4.16), toGMTString (See 1.2.7.5.7), toLocaleString (See 1.2.7.5.10), toUTCString (See 1.2.7.5.18), parse (See 1.2.7.3.12), UTC (See 1.2.7.5.22)
声明	Function (See 1.2.11.10) new (See 1.2.9.32) this (See 1.2.11.15) var (See 1.2.11.18) with (See 1.2.11.20)
错误处理	Error (See 1.2.8.6), description (See 1.2.10.5), number (See 1.2.10.30), throw (See 1.2.11.16), try...catch (See 1.2.11.17)
函数创建	Caller (See 1.2.10.3), Function (See 1.2.8.7) arguments (See 1.2.10.2), length (See 1.2.10.16)
Global 方法	Global (See 1.2.8.8) escape (See 1.2.7.1.25), unescape (See 1.2.7.5.20) eval (See 1.2.7.1.26) isFinite (See 1.2.7.2.28), isNaN (See 1.2.7.2.29) parseInt (See 1.2.7.3.14), parseFloat (See 1.2.7.3.13)
数学	Math (See 1.2.8.9) abs (See 1.2.7.1.1), acos (See 1.2.7.1.2), asin (See 1.2.7.1.5), atan (See 1.2.7.1.6), atan2 (See 1.2.7.1.7), ceil (See 1.2.7.1.13), cos (See 1.2.7.1.19), exp (See 1.2.7.1.28), floor (See 1.2.7.2.2), log (See 1.2.7.3.6), max (See 1.2.7.3.8), min (See 1.2.7.3.9), pow (See 1.2.7.3.16), random (See 1.2.7.3.18), round (See 1.2.7.3.21), sin (See 1.2.7.4.19), sqrt (See 1.2.7.4.26), tan (See 1.2.7.5.1), E (See 1.2.10.6), LN2 (See 1.2.10.19), LN10 (See 1.2.10.18), LOG2E (See 1.2.10.21), LOG10E (See 1.2.10.20), PI (See 1.2.10.31), SQRT1_2 (See 1.2.10.36), SQRT2 (See 1.2.10.37)
数字	Number (See 1.2.8.10) MAX_VALUE (See 1.2.10.22), MIN_VALUE (See 1.2.10.24) NaN (See 1.2.10.27) NEGATIVE_INFINITY (See 1.2.10.29), POSITIVE_INFINITY (See 1.2.10.32)
对象创建	Object (See 1.2.8.11) new (See 1.2.9.32) constructor (See 1.2.10.4), prototype (See 1.2.10.34), toString (See 1.2.7.5.15), valueOf (See 1.2.7.5.23)
运算符	加 (+) (See 1.2.9.3), 减 (-) (See 1.2.9.35) 算术取模 (%) (See 1.2.9.29) 乘 (*) (See 1.2.9.31), 除 (/) (See 1.2.9.20) 负 (-) (See 1.2.9.35) 相等 (==) (See 1.2.9.15), 不相等 (!=) (See 1.2.9.15) 小于 (<) (See 1.2.9.15), 小于等于 (<=) (See 1.2.9.15)

	大于 (>) (See 1.2.9.15) 大于等于 (>=) (See 1.2.9.15) 逻辑与 (&&) (See 1.2.9.25), 或 () (See 1.2.9.27), 非 (!) (See 1.2.9.26) 位与 (&) (See 1.2.9.6), 或 () (See 1.2.9.10), 非 (~) (See 1.2.9.8), 异或 (^) (See 1.2.9.13) 位左移 (<<) (See 1.2.9.7), 右移 (>>) (See 1.2.9.11) 无符号右移 (>>>) (See 1.2.9.37) 条件 (? :) (See 1.2.9.17) 逗号 (,) (See 1.2.9.14) delete (See 1.2.9.18), typeof (See 1.2.9.36), void (See 1.2.9.39) 递减 (--) (See 1.2.9.22), 递增 (++) (See 1.2.9.22), 恒等 (===) (See 1.2.9.15), 不恒等 (!==) (See 1.2.9.15)
对象	Array (See 1.2.8.2) Boolean (See 1.2.8.3) Date (See 1.2.8.4) Function (See 1.2.8.7) Global (See 1.2.8.8) Math (See 1.2.8.9) Number (See 1.2.8.10) Object (See 1.2.8.11) RegExp (See 1.2.8.12) String (See 1.2.8.14)
正则表达式和模式匹配	RegExp (See 1.2.8.12) index (See 1.2.10.10), input (See 1.2.10.12), lastIndex (See 1.2.10.14), \$1...\$9 (See 1.2.10.1), source (See 1.2.10.35), compile (See 1.2.7.1.16), exec (See 1.2.7.1.27), test (See 1.2.7.5.2) Regular Expression Syntax (See 1.5.4)
字符串	String (See 1.2.8.14) charAt (See 1.2.7.1.14), charCodeAt (See 1.2.7.1.15), fromCharCode (See 1.2.7.2.5) indexOf (See 1.2.7.2.27), lastIndexOf (See 1.2.7.3.2) split (See 1.2.7.4.25) toLowerCase (See 1.2.7.5.13), toUpperCase (See 1.2.7.5.17) length (See 1.2.10.17) concat (See 1.2.7.1.18), slice (See 1.2.7.4.21) match (See 1.2.7.3.7), replace (See 1.2.7.3.19), search (See 1.2.7.4.1) anchor (See 1.2.7.1.3), big (See 1.2.7.1.9), blink (See 1.2.7.1.10), bold (See 1.2.7.1.11), fixed (See 1.2.7.2.1), fontcolor (See 1.2.7.2.3), fontsize (See 1.2.7.2.4), italics (See 1.2.7.2.30), link (See 1.2.7.3.4), small (See 1.2.7.4.22), strike (See 1.2.7.4.27), sub (See 1.2.7.4.28), sup (See 1.2.7.4.31)

1.2.3 Microsoft JScript 特性 - 非-ECMA

种类	特性/关键字
数组处理	VBAArray (See 1.2.8.15) dimensions (See 1.2.7.1.22), getItem (See 1.2.7.2.10), lbound (See 1.2.7.3.3), toArray (See 1.2.7.5.3), ubound (See 1.2.7.5.19)
条件编译	@cc_on (See 1.2.11.1) @if 语句 (See 1.2.11.2) @set 语句 (See 1.2.11.3) 条件编译变量 (See 1.1.2.9)
日期和时间	getVarDate (See 1.2.7.2.25)
枚举	Enumerator (See 1.2.8.5) atEnd (See 1.2.7.1.8), item (See 1.2.7.2.31), moveFirst (See 1.2.7.3.10), moveNext (See 1.2.7.3.11)
对象	Enumerator (See 1.2.8.5) 正则表达式 (See 1.2.8.13) VBAArray (See 1.2.8.15) ActiveXObject (See 1.2.8.1) GetObject (See 1.2.6.1)
脚本引擎标识	ScriptEngine (See 1.2.6.2) ScriptEngineBuildVersion (See 1.2.6.3) ScriptEngineMajorVersion (See 1.2.6.4) ScriptEngineMinorVersion (See 1.2.6.5)

1.2.4 JScript 字母顺序的关键字列表

描述	语言要素
返回在模式匹配中找到的最近的九条记录。	\$1...\$9 属性 (See 1.2.10.1)
返回一个数的绝对值。	abs 方法 (See 1.2.7.1.1)
返回一个数的反余弦。	acos 方法 (See 1.2.7.1.2)
启用并返回一个 Automation 对象的引用。	ActiveXObject 对象 (See 1.2.8.1)
将两个数相加或连接两个字符串。	加法运算符 (+) (See 1.2.9.3)
在对象的指定文本两端加上一个带 NAME 属性的 HTML 锚点。	anchor 方法 (See 1.2.7.1.3)
返回一个包含传递给当前执行函数的每个参数的数组。	arguments 属性 (See 1.2.10.2)
提供对创建任何数据类型的数组的支持。	Array 对象 (See 1.2.8.2)
返回一个数的反正弦。	asin 方法 (See 1.2.7.1.5)

将一个值赋给变量。	赋值运算符 (=) (See 1.2.9.4)_
返回一个数的反正切。	atan 方法 (See 1.2.7.1.6)
返回从 x 轴到点 (y,x) 的角度(以弧度为单位)。	atan2 方法 (See 1.2.7.1.7)
返回一个指示枚举算子是否处于集合结束处的 Boolean 值。	atEnd 方法 (See 1.2.7.1.8)
在 String 对象的文本两端加入 HTML 的<BIG> 标识。	big 方法 (See 1.2.7.1.9)
对两个表达式执行按位与操作。	按位与运算符 (&) (See 1.2.9.6)_
将一个表达式的各位向左移。	按位左移运算符 (<<) (See 1.2.9.7)_
对一个表达式执行按位取非(求非)操作。	按位取非运算符 (~) (See 1.2.9.8)_
对两个表达式指定按位或操作。	按位或运算符 () (See 1.2.9.10)
将一个表达式的各位向右移, 保持符号不变。	按位右移运算符 (>>) (See 1.2.9.11)_
对两个表达式执行按位异或操作。	按位异或运算符 (^) (See 1.2.9.13)
将 HTML 的 <BLINK> 标识添加到 String 对象中的文本两端。	blink 方法 (See 1.2.7.1.10)
将 HTML 的 标识添加到 String 对象中的文本两端。	bold 方法 (See 1.2.7.1.11)
创建一个新的 Boolean 值。	Boolean 对象 (See 1.2.8.3)
终止当前循环, 或者如果与一个 label 语句关联, 则终止相关联的语句。	break 语句 (See 1.2.11.4)
返回调用当前函数的函数引用。	caller 属性 (See 1.2.10.3)
包含在 try 语句块中的代码发生错误时执行的语句。	catch 语句 (See 1.2.11.17)
激活条件编译支持。	@cc on 语句 (See 1.2.11.1)
返回大于或等于其数值参数的最小整数。	ceil 方法 (See 1.2.7.1.13)
返回位于指定索引位置的字符。	charAt 方法 (See 1.2.7.1.14)
返回指定字符的 Unicode 编码。	charCodeAt 方法 (See 1.2.7.1.15)
使两个表达式连续执行。	逗号运算符 (,) (See 1.2.9.14)
使单行注释部分被 JScript 语法分析器忽略。	// (单行注释语句) (See 1.2.11.5)
使多行注释部分被 JScript 语法分析器忽略。	/*..*/ (多行注释语句) (See 1.2.11.5)
返回一个显示比较结果的 Boolean 值。	比较运算符 (See 1.2.9.15)_
将一个正则表达式编译为内部格式。	compile 方法 (See 1.2.7.1.16)
复合赋值运算符列表。	复合赋值运算符 (See 1.2.9.16)
返回一个由两个数组合并组成的新数组。	concat 方法 (Array) (See 1.2.7.1.17)
返回一个包含给定的两个字符串连接的 String 对象。	concat 方法 (String) (See 1.2.7.1.18)
允许使用新的 JScript 语言特性, 但又不牺牲与不支持这些特性的浏览器的兼容性。	条件编译 (See 1.1.2.8)

条件编译中使用的预定义变量列表。	条件编译变量 (See 1.1.2.9)
根据条件执行两个表达式之一。	条件（三元）运算符（?:） (See 1.2.9.17)
指定创建对象的函数。	constructor 属性 (See 1.2.9.17)
停止循环的当前迭代，并开始一次新的迭代。	continue 语句 (See 1.2.11.6)
返回一个数的余弦。	cos 方法 (See 1.2.7.1.19)
提供日期和时间的基本存储和检索。	Date 对象 (See 1.2.8.4)
将变量减一。	递减运算符（--） (See 1.2.9.22)_
删除对象的属性，或删除数组中的一个元素。	delete 运算符 (See 1.2.9.18)
返回或设置关于指定错误的描述字符串。	description 属性 (See 1.2.10.5)
存储数据键、项目对的对象。	Dictionary 对象 (See 1.4.2.1)
返回 VBArray 的维数。	dimensions 方法 (See 1.2.7.1.22)
将两个数相除并返回一个数值结果。	除法运算符（/） (See 1.2.9.20)
先执行一次语句块，然后重复执行该循环，直至条件表达式的值为 false 。	do...while 语句 (See 1.2.11.7)
返回 Euler 常数，即自然对数的底。	E 属性 (See 1.2.10.6)
提供集合中的项的枚举。	Enumerator 对象 (See 1.2.8.5)
比较两个表达式，看是否相等。	相等运算符（==） (See 1.2.9.15)
包含在运行 JScript 代码时发生错误信息的对象。	Error 对象 (See 1.2.8.6)
对 String 对象编码，以便在所有计算机上都能阅读。	escape 方法 (See 1.2.7.1.25)
对 JScript 代码求值然后执行。	eval 方法 (See 1.2.7.1.26)
在指定字符串中执行一个匹配查找。	exec 方法 (See 1.2.7.1.27)
返回 e （自然对数的底）的幂。	exp 方法 (See 1.2.7.1.28)
提供对计算机文件系统的访问。	FileSystemObject 对象 (See 1.4.2.6)
将 HTML 的<TT> 标识添加到 String 对象中的文本两端。	fixed 方法 (See 1.2.7.2.1)
返回小于或等于其数值参数的最大整数。	floor 方法 (See 1.2.7.2.2)
将 HTML 带 COLOR 属性的 标识添加到 String 对象中的文本两端。	fontcolor 方法 (See 1.2.7.2.3)
将 HTML 带 SIZE 属性的 标识添加到 String 对象中的文本两端。	fontsize 方法 (See 1.2.7.2.4)
只要指定的条件为 true ，就一直执行语句块。	for 语句 (See 1.2.11.8)
对应于对象或数组中的每个元素执行一个或多个语句。	for...in 语句 (See 1.2.11.9)
返回 Unicode 字符值的字符串。	fromCharCode 方法 (See 1.2.7.2.5)
创建一个新的函数。	Function 对象 (See 1.2.8.7)
声明一个新的函数。	function 语句 (See 1.2.11.10)

使用当地时间返回 Date 对象的月份日期值。	getDate 方法 (See 1.2.7.2.6)
使用当地时间返回 Date 对象的星期几。	getDay 方法 (See 1.2.7.2.7)
使用当地时间返回 Date 对象的年份。	getFullYear 方法 (See 1.2.7.2.8)
使用当地时间返回 Date 对象的小时值。	getHours 方法 (See 1.2.7.2.9)
返回指定位置的项。	getItem 方法 (See 1.2.7.2.10)
使用当地时间返回 Date 对象的毫秒值。	getMilliseconds 方法 (See 1.2.7.2.11)
使用当地时间返回 Date 对象的分钟值。	getMinutes 方法 (See 1.2.7.2.11)
使用当地时间返回 Date 对象的月份。	getMonth 方法 (See 1.2.7.2.13)
返回文件中的 Automation 对象的引用。	GetObject 函数 (See 1.2.6.1)
使用当地时间返回 Date 对象的秒数。	getSeconds 方法 (See 1.2.7.2.14)
返回 Date 对象中的时间值。	getTime 方法 (See 1.2.7.2.15)
返回主机的时间和全球标准时间（UTC）之间的差（以分钟为单位）。	getTimezoneOffset 方法 (See 1.2.7.2.16)
使用全球标准时间（UTC）返回 Date 对象的日期值。	getUTCDate 方法 (See 1.2.7.2.17)
使用全球标准时间（UTC）返回 Date 对象的星期几的值。	getUTCDay 方法 (See 1.2.7.2.17)
使用全球标准时间（UTC）返回 Date 对象的年份。	getUTCFullYear 方法 (See 1.2.7.2.19)
使用全球标准时间（UTC）返回 Date 对象的小时数。	getUTCHours 方法 (See 1.2.7.2.20)
使用全球标准时间（UTC）返回 Date 对象的毫秒数。	getUTCMilliseconds 方法 (See 1.2.7.2.21)
使用全球标准时间（UTC）返回 Date 对象的分钟数。	getUTCMinutes 方法 (See 1.2.7.2.22)
使用全球标准时间（UTC）返回 Date 对象的月份值。	getUTCMonth 方法 (See 1.2.7.2.23)
使用全球标准时间（UTC）返回 Date 对象的秒数。	getUTCSeconds 方法 (See 1.2.7.2.24)
返回 Date 对象中的 VT_DATE。	getVarDate 方法 (See 1.2.7.2.25)
返回 Date 对象中的年份。	getYear 方法 (See 1.2.7.2.26)
是一个固有对象，目的是将全局方法集中在一个对象中。	Global 对象 (See 1.2.8.8)
比较两个表达式，看一个是否大于另一个。	大于运算符（>） (See 1.2.9.15)
比较两个表达式，看一个是否大于等于另一个。	大于等于运算符（>=） (See 1.2.9.15)
比较两个表达式，看是否值相等并具有相同的数据类型。	严格相等运算符（===） (See 1.2.9.15)
根据表达式的值，有条件地执行一组语句。	@if 语句 (See 1.2.11.2)
根据表达式的值，有条件地执行一组语句。	if...else 语句 (See 1.2.11.11)

给变量加一。	递增运算符 (++) (See 1.2.11.11)
返回在字符串中找到的第一个成功匹配的字符位置。	index 属性 (See 1.2.10.10)
返回在 String 对象中第一次出现子字符串的字符位置。	indexOf 方法 (See 1.2.7.2.27)
比较两个表达式，看是否不相等。	不等运算符 (!=) (See 1.2.9.15)
返回 Number.POSITIVE_INFINITY 的初始值。	Infinity 属性 (See 1.2.10.11)
返回进行查找的字符串。	input 属性 (See 1.2.10.12)
返回一个 Boolean 值，表明某个对象是否为特定类的一个实例。	instanceof 运算符 (See 1.2.9.23)
返回一个 Boolean 值，表明某个给定的数是否有穷的。	isFinite 方法 (See 1.2.7.2.28)
返回一个 Boolean 值，表明某个值是否为保留值 NaN （不是一个数）。	isNaN 方法 (See 1.2.7.2.29)
将 HTML 的 <I> 标识添加到 String 对象中的文本两端。	italics 方法 (See 1.2.7.2.30)
返回集合中的当前项。	item 方法 (See 1.2.7.2.31)
返回一个由数组中的所有元素连接在一起的 String 对象。	join 方法 (See 1.2.7.3.1)
给语句提供一个标识符。	Labeled 语句 (See 1.2.11.12)
返回在字符串中找到的最后一个成功匹配的字符位置。	lastIndex 属性 (See 1.2.10.14)_
返回在 String 对象中最后出现子字符串的位置。	lastIndexOf 方法 (See 1.2.7.3.2)
返回在 VBArray 中指定维数所用的最小索引值。	lbound 方法 (See 1.2.7.3.3)
返回比数组中所定义的最高元素大 1 的整数值。	length 属性 (Array) (See 1.2.10.15)
返回为函数所定义的参数个数。	length 属性 (Function) (See 1.2.10.16)
返回 String 对象的长度。	length 属性 (String) (See 1.2.10.17)
比较两个表达式，看是否一个小于另一个。	小于运算符 (<) (See 1.2.9.15)
比较两个表达式，看是否一个小于等于另一个。	小于等于运算符 (<=) (See 1.2.9.15)
将带 HREF 属性的 HTML 锚点添加到 String 对象中的文本两端。	link 方法 (See 1.2.7.3.4)
返回 2 的自然对数。	LN2 属性 (See 1.2.10.19)
返回 10 的自然对数。	LN10 属性 (See 1.2.10.18)
返回某个数的自然对数。	log 方法 (See 1.2.7.3.6)
返回以 2 为底的 e （即 Euler 常数）的对数。	LOG2E 属性 (See 1.2.10.21)
返回以 10 为底的 e （即 Euler 常数）的对数。	LOG10E 属性 (See 1.2.10.20)
对两个表达式执行逻辑与操作。	逻辑与运算符 (&&) (See 1.2.9.25)

对表达式执行逻辑非操作。	逻辑非运算符 (!) (See 1.2.9.26)
对两个表达式执行逻辑或操作。	逻辑或运算符 () (See 1.2.9.27)
使用给定的 正则表达式 对象对字符串进行查找，并将结果作为数组返回。	match 方法 (See 1.2.7.3.7)
一个固有对象，提供基本的数学函数和常数。	Math 对象 (See 1.2.8.9)
返回给定的两个表达式中的较大者。	max 方法 (See 1.2.7.3.8)
返回在 JScript 中能表示的最大值。	MAX_VALUE 属性 (See 1.2.10.22)
返回给定的两个数中的较小者。	min 方法 (See 1.2.7.3.9)
返回在 JScript 中能表示的最接近零的值。	MIN_VALUE 属性 (See 1.2.10.24)
将两个数相除，并返回余数。	取模运算符 (%) (See 1.2.9.29)
将集合中的当前项设置为第一项。	moveFirst 方法 (See 1.2.7.3.10)
将当前项设置为集合中的下一项。	moveNext 方法 (See 1.2.7.3.11)
将两个数相乘。	乘法运算符 (*) (See 1.2.9.31)
返回特殊值 NaN ，表示某个表达式不是一个数。	NaN 属性 (Global) (See 1.2.10.28)
返回特殊值 (NaN)，表示某个表达式不是一个数。	NaN 属性 (Number) (See 1.2.10.27)
返回比在 JScript 中能表示的最大的负数 (-Number.MAX_VALUE) 更小的值。	NEGATIVE_INFINITY 属性 (See 1.2.10.29)
创建一个新对象。	new 运算符 (See 1.2.9.32)
比较两个表达式，看是否具有不相等的值或数据类型不同。	非严格相等运算符 (!=) (See 1.2.9.15)
表示数值数据类型和提供数值常数的对象。	Number 对象 (See 1.2.8.10)
返回或设置与特定错误关联的数值。	number 属性 (See 1.2.10.30)
提供所有的 JScript 对象的公共功能。	Object 对象 (See 1.2.8.10)
包含 JScript 运算符的执行优先级信息的列表。	运算符优先级 (See 1.2.9.1.1)
对包含日期的字符串进行分析，并返回该日期与 1970 年 1 月 1 日零点之间相差的毫秒数。	parse 方法 (See 1.2.7.3.12)
返回从字符串转换而来的浮点数。	parseFloat 方法 (See 1.2.7.3.13)
返回从字符串转换而来的整数。	parseInt 方法 (See 1.2.7.3.14)
返回圆周与其直径的比值，约等于 3.141592653589793。	PI 属性 (See 1.2.10.31)
返回比在 JScript 中能表示的最大的数 (Number.MAX_VALUE) 更大的值。	POSITIVE_INFINITY 属性 (See 1.2.10.32)
返回一个指定幂次的底表达式的值。	pow 方法 (See 1.2.7.3.16)
返回对象类的原型引用。	prototype 属性 (See 1.2.10.34)
返回一个 0 和 1 之间的伪随机数。	random 方法 (See 1.2.7.3.18)
存储有关正则表达式模式查找的信息。	RegExp 对象 (See 1.2.8.12)
包含一个正则表达式模式。	正则表达式对象 (See 1.2.8.13)

在写正则表达式模式时可以使用的特殊字符和序列的列表。	正则表达式语法 (See 1.5.4)
返回根据正则表达式进行文字替换后的字符串的拷贝。	replace 方法 (See 1.2.7.3.19)
从当前函数退出并从该函数返回一个值。	return 语句 (See 1.2.11.13)
返回一个元素反序的 Array 对象。	reverse 方法 (See 1.2.7.3.20)
将一个指定的数值表达式舍入到最近的整数值。	round 方法 (See 1.2.7.3.21)
JScript 运行时错误列表。	运行时错误 (See 1.2.5.1)
返回一个代表所使用的脚本语言的字符串。	ScriptEngine 函数 (See 1.2.6.2)
返回所使用的脚本引擎的编译版本号。	ScriptEngineBuildVersion 函数 (See 1.2.6.2)
返回所使用的脚本引擎的主版本号。	ScriptEngineMajorVersion 函数 (See 1.2.6.4)
返回所使用的脚本引擎的次版本号。	ScriptEngineMinorVersion 函数 (See 1.2.6.5)
返回与正则表达式查找内容匹配的第一个子字符串的位置。	search 方法 (See 1.2.7.4.1)
创建用于条件编译语句的变量。	@set 语句 (See 1.2.11.3)
使用当地时间设置 Date 对象的数值日期。	setDate 方法 (See 1.2.7.4.2)
使用当地时间设置 Date 对象的年份。	setFullYear 方法 (See 1.2.7.4.3)
使用当地时间设置 Date 对象的小时值。	setHours 方法 (See 1.2.7.4.4)
使用当地时间设置 Date 对象的毫秒值。	setMilliseconds 方法 (See 1.2.7.4.5)
使用当地时间设置 Date 对象的分钟值。	setMinutes 方法 (See 1.2.7.4.6)
使用当地时间设置 Date 对象的月份。	setMonth 方法 (See 1.2.7.4.7)
使用当地时间设置 Date 对象的秒值。	setSeconds 方法 (See 1.2.7.4.8)
设置 Date 对象的日期和时间。	setTime 方法 (See 1.2.7.4.9)
使用全球标准时间 (UTC) 设置 Date 对象的数值日期。	setUTCDate 方法 (See 1.2.7.4.10)
使用全球标准时间 (UTC) 设置 Date 对象的年份。	setUTCFullYear 方法 (See 1.2.7.4.11)
使用全球标准时间 (UTC) 设置 Date 对象的小时值。	setUTCHours 方法 (See 1.2.7.4.12)
使用全球标准时间 (UTC) 设置 Date 对象的毫秒值。	setUTCMilliseconds 方法 (See 1.2.7.4.13)
使用全球标准时间 (UTC) 设置 Date 对象的分钟值。	setUTCMinutes 方法 (See 1.2.7.4.14)
使用全球标准时间 (UTC) 设置 Date 对象的月份。	setUTCMonth 方法 (See 1.2.7.4.15)
使用全球标准时间 (UTC) 设置 Date 对象的秒值。	setUTCSeconds 方法 (See 1.2.7.4.16)
使用 Date 对象的年份。	setYear 方法 (See 1.2.7.4.17)
返回一个数的正弦。	sin 方法 (See 1.2.7.4.19)

返回数组的一个片段。	slice 方法 (Array) (See 1.2.7.4.20)
返回字符串的一个片段。	slice 方法 (String) (See 1.2.7.4.21)
将 HTML 的<SMALL> 标识添加到 String 对象中的文本两端。	small 方法 (See 1.2.7.4.22)
返回一个元素被排序了的 Array 对象。	sort 方法 (See 1.2.7.4.23)
返回正则表达式模式的文本的复制。	source 属性 (See 1.2.10.35)
将一个字符串分割为子字符串，然后将结果作为字符串数组返回。	split 方法 (See 1.2.7.4.25)
返回一个数的平方根。	sqrt 方法 (See 1.2.7.4.26)
返回 0.5 的平方根，即 1 除以 2 的平方根。	SQRT1 2 属性 (See 1.2.10.36)
返回 2 的平方根。	SQRT2 属性 (See 1.2.10.37)
将 HTML 的<STRIKE> 标识添加到 String 对象中的文本两端。	strike 方法 (See 1.2.7.4.27)
提供对文本字符串的操作和格式处理，判定在字符串中是否存在某个子字符串并确定其位置。	String 对象 (See 1.2.8.14)
将 HTML 的 <SUB> 标识放置到 String 对象中的文本两端。	sub 方法 (See 1.2.7.4.28)
返回一个从指定位置开始并具有指定长度的子字符串。	substr 方法 (See 1.2.7.4.29)
返回位于 String 对象中指定位置的子字符串。	substring 方法 (See 1.2.7.4.30)
对两个表达式执行减法操作。	减法运算符 (-) (See 1.2.9.35)
将 HTML 的 <SUP> 标识放置到 String 对象中的文本两端。	sup 方法 (See 1.2.7.4.31)
当指定的表达式的值与某个标签匹配时，即执行相应的一个或多个语句。	switch 语句 (See 1.2.11.14)
JScript 运行时错误列表。	语法错误 (See 1.2.5.2)
返回一个数的正切。	tan 方法 (See 1.2.7.5.1)
返回一个 Boolean 值，表明在被查找的字符串中是否存在某个模式。	test 方法 (See 1.2.7.5.2)
对当前对象的引用。	this 语句 (See 1.2.11.15)
产生一个可由 try...catch 语句处理的错误条件。	throw 语句 (See 1.2.11.16)
返回一个从 VBArray 转换而来的标准 JScript 数组。	toArray 方法 (See 1.2.7.5.3)
返回一个转换为使用格林威治标准时间 (GMT) 的日期。	toGMTString 方法 (See 1.2.7.5.7)
返回一个转换为使用当地时间的日期。	toLocaleString 方法 (See 1.2.7.5.10)
返回一个字符串，该字符串中所有字母被转换为小写字母。	toLowerCase 方法 (See 1.2.7.5.13)
返回一个对象的字符串表示。	toString 方法 (See 1.2.7.5.15)

返回一个字符串，该字符串中所有字母都被转换为大写字母。	toUpperCase 方法 (See 1.2.7.5.17)
返回一个转换为使用全球标准时间（UTC）的日期。	toUTCString 方法 (See 1.2.7.5.18)
实现 JScript 的错误处理。	try 语句 (See 1.2.11.17)
返回一个表示表达式的数据类型的字符串。	typeof 运算符 (See 1.2.9.36)
返回在 VBArray 的指定维中所使用的最大索引值。	ubound 方法 (See 1.2.7.5.19)
表示一个数值表达式的相反数。	一元取相反数运算符 (-) (See 1.2.9.35)
对用 escape 方法编码的 String 对象进行解码。	unescape 方法 (See 1.2.7.5.20)
在表达式中对各位进行无符号右移。	无符号右移运算符 (>>>) (See 1.2.9.37)_
返回 1970 年 1 月 1 日零点的全球标准时间（UTC）（或 GMT）与指定日期之间的毫秒数。	UTC 方法 (See 1.2.7.5.22)
返回指定对象的原始值。	valueOf 方法 (See 1.2.7.5.23)
声明一个变量。	var 语句 (See 1.2.11.18)
提供对 Visual Basic 安全数组的访问。	VBArray 对象 (See 1.2.8.15)
避免一个表达式返回值。	void 运算符 (See 1.2.9.39)
执行语句直至给定的条件为 false 。	while 语句 (See 1.2.11.19)
确定一个语句的默认对象。	with 语句 (See 1.2.11.20)

1.2.5 JScript 错误

有关详细信息	参见
JScript 运行时错误列表	运行时错误 (See 1.2.5.1)
JScript 语法错误列表	语法错误 (See 1.2.5.2)

1.2.5.1 JScript 运行时错误

JScript 运行时错误是指当 JScript 脚本试图执行一个系统不能运行的动作时导致的错误。当正在运行脚本、计算变量表达式、或者正在动态分配内存时出现 JScript 运行时错误时。

错误号	描述
5029	数组长度必须为一有限正整数 (See 2.3)
5030	必须赋给数组长度一个有限正数 (See 2.2)
5028	需要 Array 或 arguments 对象 (See 2.4)
5010	需要 Boolean (See 2.5)

5003	不能给函数返回值赋值 (See 2.6)
5000	不能给 'this' 赋值 (See 2.7)
5006	需要 Date 对象 (See 2.11)
5015	需要 Enumerator 对象 (See 2.13)
5022	异常抛出，但无法抓住 (See 2.14)
5020	正则表达式中缺少 “)” (See 2.32)
5019	正则表达式中缺少 “[” (See 2.17)
5023	函数没有合法的 Prototype 对象 (See 2.34)
5002	需要 Function 对象 (See 2.35)
5008	非法赋值 (See 2.36)
5021	字符集范围无效 (See 2.38)
5014	需要 JScript 对象 (See 2.39)
5001	需要 Number 类型 (See 2.42)
5007	需要 Object 类型 (See 2.43)
5012	需要对象的成员 (See 2.44)
5016	需要正则表达式对象 (See 2.45)
5005	需要 String (See 2.47)
5017	正则表达式语法错误 (See 2.49)
5026	小数部分的位数越界 (See 2.50)
5027	精度越界 (See 2.51)
5025	待解码的 URI 编码非法 (See 2.52)
5024	待解码的 URI 包含有非法字符 (See 2.53)
5009	未定义标识符 (See 2.55)
5018	未预期的限定符 (See 2.56)
5013	需要 VArray (See 2.59)

请参阅

[JScript 语法错误](#)(See 1.2.5.2)

1.2.5.2 JScript 语法错误

JScript 语法错误是指当 JScript 语句违反了 JScript 脚本语言的一条或多条语法规则时导致的错误。JScript 语法错误发生在程序编译阶段，在开始运行该程序之前。

错误号	描述
1019	在循环外不能有 “break” (See 2.8)
1020	在循环外不能有 “continue” (See 2.9)

1030	条件编译已关闭 (See 2.10)
1027	一条 “switch” 语句中只能有一个 “default” (See 2.12)
1005	需要 “(” (See 2.15)
1006	需要 “)” (See 2.19)
1012	需要 “/” (See 2.20)
1003	需要 “:” (See 2.21)
1004	需要 “;” (See 2.22)
1032	需要 “@” (See 2.23)
1029	需要 “@end” (See 2.28)
1007	需要 “[” (See 2.16)
1008	需要 “{” (See 2.18)
1009	需要 “}” (See 2.24)
1011	需要 “=” (See 2.25)
1033	需要 “catch” (See 2.26)
1031	需要常数 (See 2.27)
1023	需要十六进制数 (See 2.29)
1010	需要标识符 (See 2.30)
1028	需要标识符、字符串或者数字 (See 2.31)
1024	需要 “while” (See 2.33)
1014	非法字符 (See 2.37)
1026	未找到标签 (See 2.40)
1025	标签定义重复 (See 2.41)
1018	函数外有 'return' 语句 (See 2.46)
1002	语法错误 (See 2.48)
1035	“Throw” 的后面必须跟有一个表达式，且在同一源代码行上 (See 2.54)
1016	注释未结束 (See 2.57)
1015	字符串常数未结束 (See 2.58)

请参阅

[JScript 运行时错误](#)(See 1.2.5.1)

1.2.6 JScript 函数

描述	语言要素
返回文件中的 Automation 对象的引用。	GetObject 函数 (See 1.2.6.1)
返回代表所使用的脚本语言的字符串。	ScriptEngine 函数 (See 1.2.6.2)
返回所使用的脚本引擎的编译版本号。	ScriptEngineBuildVersion 函数 (See 1.2.6.3)

返回所使用的脚本引擎的主版本号。	ScriptEngineMajorVersion 函数 (See 1.2.6.4)
返回所使用的脚本引擎的次版本号。	ScriptEngineMinorVersion 函数 (See 1.2.6.5)

1.2.6.1 GetObject 函数

从文件中返回对 [Automation 对象](#) 的一个引用。

GetObject(*[pathname]* [, *class*])

参数

pathname

可选项。完整的文件路径和名称，文件中包含了要检索的对象。如果忽略了 *pathname*，那么需要 *class*。

class

可选项。对象的[类](#)。

类参数采用的语法是 *appname.objecttype*，包括了以下部分：

appname

必选项。提供对象的应用程序名称。

objecttype

必选项。要创建的对象类的类型。

说明

使用 **GetObject** 函数可以从文件中访问一个 Automation 对象。可以将由 **GetObject** 返回的对象赋值给对象变量。例如：

```
var CADObject;
```

```
CADObject =
```

```
GetObject("C:\\CAD\\SCHEMA.CAD");
```

在执行这段代码时，将启动与指定的 *pathname* 相关的应用程序，所指定文件中的对象将被激活。如果 *pathname* 是长度为零的字符串 ("")，那么 **GetObject** 将返回指定类型的一个新实例。如果忽略了 *pathname* 参数，那么 **GetObject** 将返回指定类型的当前活动实例。如果不存在指定类型的对象，那么将出错。

一些应用程序允许激活文件的一部分。要实现此功能，可以在文件名的尾部添加一个感叹号 (!)，然后在感叹号后跟上一个字符串来指定要激活的文件部分。关于如何创建这个字符串的详细信息，请参阅创建该对象的应用程序的文档。例如，在一个绘画应用程序中，可能在文件中存储了绘画的很多层次。可以使用下面的代码激活名为 SCHEMA.CAD 的图画中的一层。

```
var LayerObject =
GetObject("C:\\CAD\\SCHEMA.CAD!Layer3");
```

如果没有指定对象的类，Automation 将根据所提供的文件名来决定要启动的应用程序和要激活的对象。但是，一些文件可能支持不止一种对象的类。例如，一个图画可能支持三种不同的对象类型：应用程序对象、绘画对象，以及工具栏对象，所有这些都是同一个文件中的部分。要指定文件中希望激活的对象，可以使用可选的 *class* 参数。例如：

```
var MyObject;

MyObject =
GetObject("C:\\DRAWINGS\\SAMPLE.DRW",
"FIGMENT.DRAWING");
```

在前面的例子中，FIGMENT 是绘画应用程序的名称，而 DRAWING 是它支持的一种对象类型。对象被激活后，可以使用定义的对象变量来引用它。在前面的例子中，可以通过对象变量 MyObject 来访问新对象的属性和方法。例如：

```
MyObject.Line(9, 90);

MyObject.InsertText(9, 100, "Hello,
world.");

MyObject.SaveAs("C:\\DRAWINGS\\SAMPLE.D
RW");
```

注意 如果当前已经有了对象的实例，或者想要由已经加载了的文件创建对象，那么可以使用 **GetObject** 函数。如果没有当前的实例，而且也不想由已经加载的文件来启动对象，那么可以使用 **ActiveXObject** 对象。

如果对象自身已经注册为单实例对象了，那么无论执行多少次 **ActiveXObject** 也只会创建对象的一个实例。对于单实例对象，**GetObject** 在用长度为零的字符串("")语法调用时总是返回相同的实例，而如果忽略了 *pathname* 参数就会出错。

要求

[版本 5](#)(See 2.64)

请参阅

[ActiveXObject 对象](#)(See 1.2.8.1)

1.2.6.2 ScriptEngine 函数

返回一个字符串，代表正在使用的脚本语言。

ScriptEngine()

说明

ScriptEngine 函数可以返回下列任意字符串：

字符串	描述
JScript	指明 Microsoft JScript 是当前的 Scripting 引擎。
VBA	指明 Microsoft Visual Basic for Applications 是当前的 Scripting 引擎。
VBScript	指明 Microsoft Visual Basic Scripting Edition 是当前的 Scripting 引擎。

示例

下面的例子说明了 **ScriptEngine** 函数的用法：

```
function GetScriptEngineInfo() {
    var s;

    s = ""; // 根据必要的信息创建字符串。

    s += ScriptEngine() + " Version ";
    s += ScriptEngineMajorVersion() + ".";
    s += ScriptEngineMinorVersion() + ".";
    s += ScriptEngineBuildVersion();

    return(s);
}
```

要求

[版本 5](#)(See 2.64)

请参阅

[ScriptEngineBuildVersion 函数](#)(See 1.2.6.3) | [ScriptEngineMajorVersion 函数](#)(See

1.2.6.4) | [ScriptEngineMinorVersion 函数](#)(See 1.2.6.5)

1.2.6.3 ScriptEngineBuildVersion 函数

返回所使用的 Scripting 引擎的生成版本号。

ScriptEngineBuildVersion()

说明

返回值直接依赖于所使用的 Scripting 语言的动态链接库 (DLL) 中包含的版本信息。

示例

下面的例子说明了 **ScriptEngineBuildVersion** 函数的用法：

```
function GetScriptEngineInfo() {  
    var s;  
    s = ""; // 根据必要的信息创建字符串。  
    s += ScriptEngine() + " Version ";  
    s += ScriptEngineMajorVersion() + ".";  
    s += ScriptEngineMinorVersion() + ".";  
    s += ScriptEngineBuildVersion();  
    return(s);  
}
```

要求

[版本 5](#)(See 2.64)

请参阅

[ScriptEngine 函数](#)(See 1.2.6.2) | [ScriptEngineMajorVersion 函数](#)(See 1.2.6.4) | [ScriptEngineMinorVersion 函数](#)(See 1.2.6.5)

1.2.6.4 ScriptEngineMajorVersion 函数

返回所使用的 Scripting 引擎的主要版本号。

ScriptEngineMajorVersion()

说明

返回值直接依赖于所使用的 Scripting 语言的动态链接库 (DLL) 中包含的版本信息。

示例

下面的例子说明了 **ScriptEngineMajorVersion** 函数的用法：

```
function GetScriptEngineInfo() {  
    var s;  
    s = ""; // 根据必要的信息创建字符串。  
    s += ScriptEngine() + " Version ";  
    s += ScriptEngineMajorVersion() + ".";  
    s += ScriptEngineMinorVersion() + ".";  
    s += ScriptEngineBuildVersion();  
    return(s);  
}
```

要求

[版本 5](#)(See 2.64)

请参阅

[ScriptEngine 函数](#)(See 1.2.6.2) | [ScriptEngineBuildVersion 函数](#)(See 1.2.6.3) | [ScriptEngineMinorVersion 函数](#)(See 1.2.6.5)

1.2.6.5 ScriptEngineMinorVersion 函数

返回所使用的 Scripting 引擎的次要版本号。

ScriptEngineMinorVersion()

说明

返回值直接依赖于所使用的 Scripting 语言的动态链接库 (DLL) 中包含的版本信息。

示例

下面的例子说明了 **ScriptEngineMinorVersion** 函数的用法：

```
function GetScriptEngineInfo() {
    var s;

    s = ""; // 根据必要的信息创建字符串。

    s += ScriptEngine() + " Version ";
    s += ScriptEngineMajorVersion() + ".";
    s += ScriptEngineMinorVersion() + ".";
    s += ScriptEngineBuildVersion();

    return(s);
}
```

要求

[版本 5](#)(See 2.64)

请参阅

[ScriptEngine 函数](#)(See 1.2.6.2) | [ScriptEngineBuildVersion 函数](#)(See 1.2.6.3) | [ScriptEngineMajorVersion 函数](#)(See 1.2.6.4)

1.2.7 JScript 方法

描述	语言要素
返回一个数的绝对值。	abs 方法 (See 1.2.7.1.1)
返回一个数的反余弦。	acos 方法 (See 1.2.7.1.2)
在对象的指定文本两端加上一个带 NAME 属性的 HTML 锚点。	anchor 方法 (See 1.2.7.1.3)
返回一个数的反正弦。	asin 方法 (See 1.2.7.1.5)
返回一个数的反正切。	atan 方法 (See 1.2.7.1.6)
返回从 x 轴到点 (y,x)的角度(以弧度为单位)。	atan2 方法 (See 1.2.7.1.7)
返回一个表明枚举算子是否处于集合结束处的 Boolean 值。	atEnd 方法 (See 1.2.7.1.8)
在 String 对象的文本两端加入 HTML 的<BIG>标识。	big 方法 (See 1.2.7.1.9)
将 HTML 的 <BLINK> 标识添加到 String 对象中的文本两端。	blink 方法 (See 1.2.7.1.10)

将 HTML 的 标识添加到 String 对象中的文本两端。	bold 方法 (See 1.2.7.1.11)
返回大于或等于其数值参数的最小整数。	ceil 方法 (See 1.2.7.1.13)
返回位于指定索引位置的字符。	charAt 方法 (See 1.2.7.1.14)
返回指定字符的 Unicode 编码。	charCodeAt 方法 (See 1.2.7.1.15)
将一个正则表达式编译为内部格式。	compile 方法 (See 1.2.7.1.16)
返回一个由两个数组合并组成的新数组。	concat 方法 (Array) (See 1.2.7.1.17)
返回一个包含给定的两个字符串的连接 String 对象。	concat 方法 (String) (See 1.2.7.1.18)
返回一个数的余弦。	cos 方法 (See 1.2.7.1.19)
返回 VBArray 的维数。	dimensions 方法 (See 1.2.7.1.22)
对 String 对象编码，以便在所有计算机上都能阅读。	escape 方法 (See 1.2.7.1.25)
对 JScript 代码求值然后执行之。	eval 方法 (See 1.2.7.1.26)
在指定字符串中执行一个匹配查找。	exec 方法 (See 1.2.7.1.27)
返回 e（自然对数的底）的幂。	exp 方法 (See 1.2.7.1.28)
将 HTML 的<TT> 标识添加到 String 对象中的文本两端。	fixed 方法 (See 1.2.7.2.1)
返回小于或等于其数值参数的最大整数。	floor 方法 (See 1.2.7.2.2)
将 HTML 带 COLOR 属性的标识添加到 String 对象中的文本两端。	fontcolor 方法 (See 1.2.7.2.3)
将 HTML 带 SIZE 属性的标识添加到 String 对象中的文本两端。	fontsize 方法 (See 1.2.7.2.4)
返回 Unicode 字符值的字符串。	fromCharCode 方法 (See 1.2.7.2.5)
使用当地时间返回 Date 对象的月份日期值。	getDate 方法 (See 1.2.7.2.6)
使用当地时间返回 Date 对象的星期几。	getDay 方法 (See 1.2.7.2.7)
使用当地时间返回 Date 对象的年份。	getFullYear 方法 (See 1.2.7.2.8)
使用当地时间返回 Date 对象的小时值。	getHours 方法 (See 1.2.7.2.9)
返回位于指定位置的项。	getItem 方法 (See 1.2.7.2.10)
使用当地时间返回 Date 对象的毫秒值。	getMilliseconds 方法 (See 1.2.7.2.11)
使用当地时间返回 Date 对象的分钟值。	getMinutes 方法 (See 1.2.7.2.12)
使用当地时间返回 Date 对象的月份。	getMonth 方法 (See 1.2.7.2.13)
使用当地时间返回 Date 对象的秒数。	getSeconds 方法 (See 1.2.7.2.14)
返回 Date 对象中的时间。	getTime 方法 (See 1.2.7.2.15)
返回主机的时间和全球标准时间（UTC）之间的差（以分钟为单位）。	getTimezoneOffset 方法 (See 1.2.7.2.15)
使用全球标准时间（UTC）返回 Date 对象的日期值。	getUTCDate 方法 (See 1.2.7.2.17)

使用全球标准时间 (UTC) 返回 Date 对象的星期几。	getUTCDay 方法 (See 1.2.7.2.18)
使用全球标准时间 (UTC) 返回 Date 对象的年份。	getUTCFullYear 方法 (See 1.2.7.2.19)
使用全球标准时间 (UTC) 返回 Date 对象的小时数。	getUTCHours 方法 (See 1.2.7.2.20)
使用全球标准时间 (UTC) 返回 Date 对象的毫秒数。	getUTCMilliseconds 方法 (See 1.2.7.2.21)
使用全球标准时间 (UTC) 返回 Date 对象的分钟数。	getUTCMinutes 方法 (See 1.2.7.2.22)
使用全球标准时间 (UTC) 返回 Date 对象的月份值。	getUTCMonth 方法 (See 1.2.7.2.23)
使用全球标准时间 (UTC) 返回 Date 对象的秒数。	getUTCSeconds 方法 (See 1.2.7.2.24)
返回 Date 对象中的 VT_DATE。	getVarDate 方法 (See 1.2.7.2.25)
返回 Date 对象中的年份。	getFullYear 方法 (See 1.2.7.2.26)
返回在 String 对象中第一次出现子字符串的字符位置。	indexOf 方法 (See 1.2.7.2.27)
返回一个 Boolean 值, 表明某个给定的数是否有穷的。	isFinite 方法 (See 1.2.7.2.28)
返回一个 Boolean 值, 表明某个值是否为保留值 NaN (不是一个数)。	isNaN 方法 (See 1.2.7.2.29)
将 HTML 的 <I> 标识添加到 String 对象中的文本两端。	italics 方法 (See 1.2.7.2.30)
返回集合中的当前项。	item 方法 (See 1.2.7.2.31)
返回一个由数组中的所有元素连接在一起的 String 对象。	join 方法 (See 1.2.7.3.1)
返回在 String 对象中子字符串最后出现的位置。	lastIndexOf 方法 (See 1.2.7.3.2)
返回在 VBArray 中指定维数所用的最小索引值。	lbound 方法 (See 1.2.7.3.3)
将带 HREF 属性的 HTML 锚点添加到 String 对象中的文本两端。	link 方法 (See 1.2.7.3.4)
返回某个数的自然对数。	log 方法 (See 1.2.7.3.6)
使用给定的正则表达式对象对字符串进行查找, 并将结果作为数组返回。	match 方法 (See 1.2.7.3.7)
返回给定的两个表达式中的较大者。	max 方法 (See 1.2.7.3.8)
返回给定的两个数中的较小者。	min 方法 (See 1.2.7.3.9)
将集合中的当前项设置为第一项。	moveFirst 方法 (See 1.2.7.3.10)
将当前项设置为集合中的下一项。	moveNext 方法 (See 1.2.7.3.11)
对包含日期的字符串进行分析, 并返回该日期与	parse 方法 (See 1.2.7.3.12)

1970 年 1 月 1 日零点之间相差的毫秒数。	
返回从字符串转换而来的浮点数。	parseFloat 方法 (See 1.2.7.3.13)
返回从字符串转换而来的整数。	parseInt 方法 (See 1.2.7.3.14)
返回一个指定幂次的底表达式的值。	pow 方法 (See 1.2.7.3.16)
返回一个 0 和 1 之间的伪随机数。	random 方法 (See 1.2.7.3.16)
返回根据正则表达式进行文字替换后的字符串的拷贝。	replace 方法 (See 1.2.7.3.19)
返回一个元素反序的 Array 对象。	reverse 方法 (See 1.2.7.3.20)
将一个指定的数值表达式舍入到最近的整数并将其返回。	round 方法 (See 1.2.7.3.21)
返回与正则表达式查找内容匹配的字符串的第一个子字符串的位置。	search 方法 (See 1.2.7.4.1)
使用当地时间设置 Date 对象的数值日期。	setDate 方法 (See 1.2.7.4.2)
使用当地时间设置 Date 对象的年份。	setFullYear 方法 (See 1.2.7.4.3)
使用当地时间设置 Date 对象的小时值。	setHours 方法 (See 1.2.7.4.4)
使用当地时间设置 Date 对象的毫秒值。	setMilliseconds 方法 (See 1.2.7.4.5)
使用当地时间设置 Date 对象的分钟值。	setMinutes 方法 (See 1.2.7.4.6)
使用当地时间设置 Date 对象的月份。	setMonth 方法 (See 1.2.7.4.7)
使用当地时间设置 Date 对象的秒值。	setSeconds 方法 (See 1.2.7.4.8)
设置 Date 对象的日期和时间。	setTime 方法 (See 1.2.7.4.9)
使用全球标准时间 (UTC) 设置 Date 对象的数值日期。	setUTCDate 方法 (See 1.2.7.4.10)
使用全球标准时间 (UTC) 设置 Date 对象的年份。	setUTCFullYear 方法 (See 1.2.7.4.11)
使用全球标准时间 (UTC) 设置 Date 对象的小时值。	setUTCHours 方法 (See 1.2.7.4.12)
使用全球标准时间 (UTC) 设置 Date 对象的毫秒值。	setUTCMilliseconds 方法 (See 1.2.7.4.13)
使用全球标准时间 (UTC) 设置 Date 对象的分钟值。	setUTCMinutes 方法 (See 1.2.7.4.14)
使用全球标准时间 (UTC) 设置 Date 对象的月份。	setUTCMonth 方法 (See 1.2.7.4.15)
使用全球标准时间 (UTC) 设置 Date 对象的秒值。	setUTCSeconds 方法 (See 1.2.7.4.16)
使用 Date 对象的年份。	setYear 方法 (See 1.2.7.4.17)
返回一个数的正弦。	sin 方法 (See 1.2.7.4.19)
返回数组的一个片段。	slice 方法 (Array) (See 1.2.7.4.20)
返回字符串的一个片段。	slice 方法 (String) (See 1.2.7.4.21)
将 HTML 的<SMALL> 标识添加到 String 对象	small 方法 (See 1.2.7.4.22)

中的文本两端。	
返回一个元素被排序了的 Array 对象。	sort 方法 (See 1.2.7.4.23)
将一个字符串分割为子字符串，然后将结果作为字符串数组返回。	split 方法 (See 1.2.7.4.25)
返回一个数的平方根。	sqrt 方法 (See 1.2.7.4.26)
将 HTML 的<STRIKE> 标识添加到 String 对象中的文本两端。	strike 方法 (See 1.2.7.4.27)
将 HTML 的 <SUB> 标识放置到 String 对象中的文本两端。	sub 方法 (See 1.2.7.4.28)
返回一个从指定位置开始并具有指定长度的子字符串。	substr 方法 (See 1.2.7.4.29)
返回位于 String 对象中指定位置的子字符串。	substring 方法 (See 1.2.7.4.30)
将 HTML 的 <SUP> 标识放置到 String 对象中的文本两端。	sup 方法 (See 1.2.7.4.31)
返回一个数的正切。	tan 方法 (See 1.2.7.5.1)
返回一个 Boolean 值，表明在被查找的字符串中是否存在某个模式。	test 方法 (See 1.2.7.5.2)
返回一个从 VBArray 转换而来的标准 JScript 数组。	toArray 方法 (See 1.2.7.5.3)
返回一个转换为使用格林威治标准时间（GMT）的字符串的日期。	toGMTString 方法 (See 1.2.7.5.7)
返回一个转换为使用当地时间的字符串的日期。	toLocaleString 方法 (See 1.2.7.5.10)
返回一个所有的字母字符都被转换为小写字母的字符串。	toLowerCase 方法 (See 1.2.7.5.13)
返回一个对象的字符串表示。	toString 方法 (See 1.2.7.5.15)
返回一个所有的字母字符都被转换为大写字母的字符串。	toUpperCase 方法 (See 1.2.7.5.17)
返回一个转换为使用全球标准时间（UTC）的字符串的日期。	toUTCString 方法 (See 1.2.7.5.18)
返回在 VBArray 的指定维中所使用的最大索引值。	ubound 方法 (See 1.2.7.5.19)
对用 escape 方法编码的 String 对象进行解码。	unescape 方法 (See 1.2.7.5.20)
返回 1970 年 1 月 1 日零点的全球标准时间（UTC）（或 GMT）与指定日期之间的毫秒数。	UTC 方法 (See 1.2.7.5.22)
返回指定对象的原始值。	valueOf 方法 (See 1.2.7.5.23)

1.2.7.1 A-E

1.2.7.1.1 abs 方法

返回数字的绝对值。

`Math.abs(number)`

必选项 *number* 参数是要需要计算绝对值的[数值表达式](#)。

说明

返回的值是 *number* 参数的绝对值。

示例

下面的例子演示了 **abs** 方法的使用。

```
function ComparePosNegVal(n)
{
    var s;

    var v1 = Math.abs(n);
    var v2 = Math.abs(-n);

    if (v1 == v2)
        s = n + " 和 "

        s += -n + " 的绝对值是相同的。";

    return(s);
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象方法](#)(See 1.2.8.9)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.1.2 acos 方法

返回数的反余弦值。

`Math.acos(number)`

必选项 *number* 参数是需要计算反余弦值的[数值表达式](#)。

说明

返回值为 *number* 参数的反余弦值。

要求

[版本 1](#)(See 2.64)

请参阅

[asin 方法](#)(See 1.2.7.1.5) | [atan 方法](#)(See 1.2.7.1.6) | [cos 方法](#)(See 1.2.7.1.19) | [sin 方法](#)(See 1.2.7.4.19) | [tan 方法](#)(See 1.2.7.5.1)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.1.3 anchor 方法

在对象中的指定文本两端放置一个有 NAME 属性的 HTML 锚点。

`strVariable.anchor(anchorString)`

参数

strVariable

必选项。任意的 **String** 对象或文字。

anchorString

必选项。想放在 HTML 锚点的 NAME 属性中的文本。

说明

调用 **anchor** 方法来在 **String** 对象外创建一个命名的锚点。下面示例说明了 **anchor** 方法是如何实现这个的:

```
var strVariable = "This is an anchor" ;

strVariable

strVariable.anchor("Anchor1") ;
```

执行完最后一条语句后 *strVariable* 的值为:

This is an anchor

不检查该标记是否已经被应用于该字符串。

要求

[版本 1](#)(See 2.64)

请参阅

[link 方法](#)(See 1.2.7.3.4) | [String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.1.4 apply 方法

应用某一对象的一个方法，用另一个对象替换当前对象。

apply ([*thisObj* [, *argArray*]])

参数

thisObj

可选项。将被用作当前对象的对象。

argArray

可选项。将被传递给该函数的参数数组。

说明

如果 *argArray* 不是一个有效的数组或者不是 **arguments** 对象，那么将导致一个 **TypeError**。

如果没有提供 *argArray* 和 *thisObj* 任何一个参数，那么 **Global** 对象将被用作 *thisObj*，并且无法被传递任何参数。

要求

[版本 5.5](#)(See 2.64)

请参阅

应用于: [Function 对象](#)(See 1.2.8.7)

1.2.7.1.5 asin 方法

返回数字的反正弦值。

`Math.asin(number)`

必选项 *number* 参数是需要计算反正弦的[数值表达式](#)。

说明

返回值为数字参数的反正弦值。

要求

[版本 1](#)(See 2.64)

请参阅

[acos 方法](#)(See 1.2.7.1.2) | [atan 方法](#)(See 1.2.7.1.6) | [cos 方法](#)(See 1.2.7.1.19) | [Math 对象方法](#)(See 1.2.8.9) | [sin 方法](#)(See 1.2.7.4.19) | [tan 方法](#)(See 1.2.7.5.1)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.1.6 atan 方法

返回数字的反正切值。

`Math.atan(number)`

必选项 *number* 参数是需要计算反正切的[数值表达式](#)。

说明

返回值为其数字参数的反正切值。

要求

[版本 1](#)(See 2.64)

请参阅

[acos 方法](#)(See 1.2.7.1.2) | [asin 方法](#)(See 1.2.7.1.5) | [atan2 方法](#)(See 1.2.7.1.7) | [cos 方法](#)(See 1.2.7.1.19) | [Math 对象方法](#)(See 1.2.8.9) | [sin 方法](#)(See 1.2.7.4.19) | [tan 方法](#)(See 1.2.7.5.1)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.1.7 atan2 方法

返回由 X 轴到 (y,x) 点的角度（以弧度为单位）。

`Math.atan2(y, x)`

参数

x

必选项。描述笛卡儿 *x* 坐标的[数值表达式](#)。

y

必选项。描述笛卡儿 *y* 坐标的数值表达式。

说明

返回值在 $-pi$ 和 pi 之间，表示提供的 (*y*,*x*) 点的角度。

要求

[版本 1](#)(See 2.64)

请参阅

[atan 方法](#)(See 1.2.7.1.6) | [tan 方法](#)(See 1.2.7.5.1)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.1.8 atEnd 方法

返回一个 Boolean 值，指明枚举算子是否位于集合的末尾。

myEnum.**atEnd**()

必选项 *myEnum* 参数是任意 **Enumerator** 对象。

说明

如果当前项是集合中的最后一个，或者集合为空，或者当前项没有定义,那么 **atEnd** 方法将返回 **true** 。否则返回 **false** 。

示例

在下面的代码中，使用了 **atEnd** 方法来决定是否到达了一个驱动器列表的末尾：

```
function ShowDriveList() {
    var fso, s, n, e, x;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    e = new Enumerator(fso.Drives);
    s = "";

    for (; !e.atEnd(); e.moveNext())
```

```

{
    x = e.item();
    s = s + x.DriveLetter;
    s += " - ";
    if (x.DriveType == 3)
        n = x.ShareName;
    else if (x.IsReady)
        n = x.VolumeName;
    else
        n = "[ 驱动器未就绪 ] ";
    s +=    n + "<br>";
}

return(s);
}

```

要求

[版本 2](#)(See 2.64)

请参阅

[item 方法](#)(See 1.2.7.2.31) | [moveFirst 方法](#)(See 1.2.7.3.10) | [moveNext 方法](#)(See 1.2.7.3.11)

应用于: [Enumerator 对象](#)(See 1.2.8.5)

1.2.7.1.9 big 方法

把 HTML <BIG> 标记放置在 **String** 对象中的文本两端。

strVariable.**big**()

必选项 *strVariable* 参数可以是任意的 **String** 对象或文字。

说明

下面示例说明了 **big** 方法是如何工作的:

```
var strVariable = "This is a string  
object";
```

```
strVariable = strVariable.big( );
```

在执行完最后一个语句后，*strVariable* 的值为：

```
<BIG>This is a string object</BIG>
```

不检查该标记是否已经被应用于该字符串了。

要求

[版本 1](#)(See 2.64)

请参阅

[small 方法](#)(See 1.2.7.4.22) | [String 对象方法](#)(See 1.2.8.14) | [String 对象属性](#)(See 1.2.8.14)

应用于： [String 对象](#)(See 1.2.8.14)

1.2.7.1.10 blink 方法

把 HTML <BLINK> 标记放置在 **String** 对象中的文本两端。

```
strVariable.blink( )
```

必选项 *strVariable* 参数是可以是任意的 **String** 对象或文字。

说明

下面的示例说明了 **blink** 方法是如何工作的：

```
var strVariable = "This is a string  
object";
```

```
strVariable = strVariable.blink( );
```

在最后一语句执行完后，*strVariable* 的值是：

```
<BLINK>This is a string object</BLINK>
```

不检查该标记是否已经被应用于该字符串了。

在 Microsoft Internet Explorer 中不支持 <BLINK> 标记。

要求

[版本 1](#)(See 2.64)

请参阅

[String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.1.11 bold 方法

把 HTML 标记放置在 **String** 对象中的文本两端。

`strVariable.bold()`

必选项 `strVariable` 参数可以是任意的 **String** 对象或文字。

说明

下面的示例说明了 **bold** 方法是如何工作的:

```
var strVariable = "This is a string
object";
```

```
strVariable = strVariable.bold( );
```

最后一条语句执行完后, `strVariable` 的值是:

```
<B>This is a string object</B>
```

不检查该标记是否已经被应用于该字符串了。

要求

[版本 1](#)(See 2.64)

请参阅

[italics 方法](#)(See 1.2.7.2.30) | [String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.1.12 call 方法

调用一个对象的一个方法, 以另一个对象替换当前对象。

`call([thisObj[,arg1[, arg2[, ..., .argN]]]])`

参数

`thisObj`

可选项。将被用作当前对象的对象。

`arg1, arg2, ..., argN`

可选项。将被传递方法参数序列。

说明

call 方法可以用来代替另一个对象调用一个方法。**call** 方法可将一个函数的对象上下文从初始的上下文改变为由 *thisObj* 指定的新对象。

如果没有提供 *thisObj* 参数，那么 **Global** 对象被用作 *thisObj*。

要求

[版本 5.5](#)(See 2.64)

请参阅

应用于: [Function 对象](#)(See 1.2.8.7)

1.2.7.1.13 ceil 方法

返回大于等于其数字参数的最小整数。

Math.ceil(*number*)

必选项 *number* 参数是[数值表达式](#)。

说明

返回值为大于等于其数字参数的最小整数。

要求

[版本 1](#)(See 2.64)

请参阅

[floor 方法](#)(See 1.2.7.2.2) | [Math 对象方法](#)(See 1.2.8.9)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.1.14 charAt 方法

返回指定索引位置处的字符。

strObj.**charAt**(*index*)

参数

strObj

必选项。任意 **String** 对象或文字。

index

必选项。想得到的字符的基于零的索引。有效值是 0 与字符串长度减 1 之间的值。

说明

charAt 方法返回一个字符值，该字符位于指定索引位置。字符串中的第一个字符的索引为 0，第二个的索引为 1，等等。超出有效范围的索引值返回空字符串。

示例

下面的示例说明了 **charAt** 方法的用法：

```
function charAtTest(n) {
    var str =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"; // 初始化变量。

    var s; //
    // 声名变量。
    s = str.charAt(n - 1);
    // 从索引为 n - 1 的位置处
    //
    // 获取正确的字符。
    return s; //
    // 返回字符。
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14)

应用于： [String 对象](#)(See 1.2.8.14)

1.2.7.1.15 charCodeAt 方法

返回一个整数，代表指定位置上字符的 Unicode 编码。

strObj.**charCodeAt**(*index*)

参数

strObj

必选项。任何 **String** 对象或文字。

index

必选项。将被处理的字符的从零开始计数的编号。有效值为 0 到字符串长度减 1 的数字。

说明

一个字符串中的第一个字符编号为 0，第二个字符编号为 1，依此类推。如果指定位置没有字符，将返回 **NaN**。

示例

以下示例阐明了 **charCodeAt** 方法的使用。

```
function charCodeAtTest(n) {

    var str = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    // 初始化变量。

    var n;                                     // 声明变量。

    n = str.charCodeAt(n - 1);

    // 获取位置 n 上字符的 Unicode 值。

    return(n);                                // 返回该值。
}
```

要求

[版本 5.5](#)(See 2.64)

请参阅

[fromCharCode 方法](#)(See 1.2.7.2.5) | [String 对象方法](#)(See 1.2.8.14)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.1.16 compile 方法

把正则表达式编译为内部格式，从而执行得更快。

rgExp.compile(pattern, [flags])

参数

rgexp

必选项。**正则表达式**对象的一个例子。可以是变量名或文字。

pattern

必选项。[字符串表达式](#)，它包含要被编译的正则表达式模式。

flags

可选项。可以组合使用，可用的标志有：

- **g** (全局搜索所有存在的 *pattern*)
- **i** (忽略事件)
- **m** (多行搜索)

说明

compile 方法把 *pattern* 转换为内部的格式，从而执行得更快。例如，这使得可以在循环中更有效地使用正则表达式。当重复使用相同的表达式，那么编译过的正则表达式使执行加速。然而，如果正则表达式改变了，这种编译毫无益处。

示例

下面的例子举例说明了 **compile** 方法的用法：

```
function CompileDemo() {
    var rs;

    var s = "AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPp"

    // 只为大写字母创建正则表达式。

    var r = new RegExp("[A-Z]", "g");
    var a1 = s.match(r) // 查找匹
    配。

    // 只为小写字母编译正则表达式。
```

```

    r.compile("[a-z]", "g");

    var a2 = s.match(r)                // 查找匹
配。

    return(a1 + "\n" + a2;
}

```

要求

[版本 3](#)(See 2.64)

请参阅

[正则表达式对象的方法](#)(See 1.2.8.13) | [正则表达式对象的属性](#)(See 1.2.8.13) | [正则表达式的语法](#)(See 1.5.4)

应用于: [正则表达式对象](#)(See 1.2.8.13)

1.2.7.1.17 concat 方法 (Array)

返回一个新数组，这个新数组是由两个或更多数组组合而成的。

```
array1.concat([item1[, item2[, . . . [, itemN]]]])
```

参数

array1

必选项。其他所有数组要进行连接的 **Array** 对象。

item1, . . . , itemN

可选项。要连接到 *array1* 末尾的其他项目。

说明

concat 方法返回一个 **Array** 对象，其中包含了 *array1* 和提供的任意其他项目的连接。

要加的项目 (*item1* ... *itemN*) 会按照从左到右的顺序添加到数组。如果某一项为数组，那么添加其内容到 *array1* 的末尾。如果该项目不是数组，就将其作为单个的数组元素添加到数组的末尾。

以下为从源数组复制元素到结果数组：

- 对于从正被连接到新数组的数组中复制的对象参数，复制后仍然指向相同的对象。不论新数组和源数组中哪一个有改变，都将引起另一个的改变。

- 对于连接到新数组的数值或字符串，只复制其值。一个数组中值有改变并不影响另一个数组中的值。

示例

下面这个例子说明了使用数组时 **concat** 方法的用法:

```
function ConcatArrayDemo() {  
    var a, b, c, d;  
    a = new Array(1, 2, 3);  
    b = "JScript";  
    c = new Array(42, "VBScript");  
    d = a.concat(b, c);  
    // 返回数组 [1, 2, 3, "JScript", 42,  
    "VBScript"]  
    return(d);  
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[concat 方法 \(String\)](#)(See 1.2.7.1.18) | [join 方法](#)(See 1.2.7.3.1) | [String 对象](#)(See 1.2.8.14)

应用于: [Array 对象](#)(See 1.2.8.2)

1.2.7.1.18 concat 方法 (String)

返回字符串值，该值包含了两个或更多个提供的字符串的连接。

string1.concat([string2[, string3[, . . . [, stringN]]])

参数

string1

必选项。要和所有其他指定的字符串进行连接的 **String** 对象或文字。

string2, . . . , *stringN*

可选项。要连接到 *string1* 尾部的 **String** 对象或文字。

说明

concat 方法的结果等于： $result = string1 + string2 + string3 + \cdots + stringN$ 。不论源字符串或结果字符串哪一个中的值改变了都不会影响另一个字符串中的值。如果有不是字符串的参数，在被连接到 *string1* 之前将先被转换为字符串。

示例

下面的示例说明了使用字符串时 **concat** 方法的用法：

```
function concatDemo()
{
    var str1 = "ABCDEFGHJKLM"
    var str2 = "NOPQRSTUVWXYZ";
    var s = str1.concat(str2);

    // 返回连接好的字符串。

    return(s);
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[加法运算符\(+\)](#)(See 1.2.9.3) | [Array 对象](#)(See 1.2.8.2) | [concat 方法 \(Array\)](#)(See 1.2.7.1.17) | [String 对象的方法](#)(See 1.2.8.14)

应用于：[String 对象](#)(See 1.2.8.14)

1.2.7.1.19 cos 方法

返回数的余弦值。

Math.cos(*number*)

必选项 *number* 参数是需要计算余弦的[数值表达式](#)。

说明

返回值为其数字参数的余弦值。

要求

[版本 1](#)(See 2.64)

请参阅

[acos 方法](#)(See 1.2.7.1.2) | [asin 方法](#)(See 1.2.7.1.5) | [atan 方法](#)(See 1.2.7.1.6) | [Math 对象方法](#)(See 1.2.8.9) | [sin 方法](#)(See 1.2.7.4.19) | [tan 方法](#)(See 1.2.7.5.1)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.1.20 decodeURI 方法

返回一个已编码的统一资源标识符 (URI) 的非编码形式。

decodeURI (*URIString*)

必要的 *URIString* 参数代表一个已编码 URI 的值。

说明

使用 **decodeURI** 方法代替已经过时的 **unescape** 方法。

decodeURI 方法返回一个字符串值。

如果 *URIString* 无效, 那么将产生一个 **URIError**。

要求

[版本 5.5](#)(See 2.64)

请参阅

[decodeURIComponent 方法](#)(See 1.2.7.1.21) | [encodeURIComponent 方法](#)(See 1.2.7.1.23)

应用于: [Global 对象](#)(See 1.2.8.8)

1.2.7.1.21 decodeURIComponent 方法

返回统一资源标识符 (URI) 的一个已编码组件的非编码形式。

decodeURIComponent (*encodedURIString*)

必选的 *encodedURIString* 参数代表一个已编码的 URI 组件。

说明

URIComponent 是一个完整的 URI 的一部分。

如果 *encodedURIString* 无效, 将产生一个 **URIError**。

要求

[版本 5.5](#)(See 2.64)

请参阅

[decodeURI 方法](#)(See 1.2.7.1.20) | [encodeURIComponent 方法](#)(See 1.2.7.1.23)

应用于: [Global 对象](#)(See 1.2.8.8)

1.2.7.1.22 dimensions 方法

返回 VBAArray 的维数。

`array.dimensions()`

必选项 *array* 参数为 **VBAArray** 对象。

说明

dimensions 方法提供了一个获取指定的 VBAArray 维数的方法。

下面的示例包括三个部分。第一部分是用来创建一个 Visual Basic 安全数组的 VBScript 代码。第二部分是 JScript 代码，确定该安全数组的维数和每一维的上界。这两部分都放在 HTML 页中的 <HEAD> 部分。第三部分是位于 <BODY> 部分的 JScript 代码，用来运行其他两个部分。

```
<HEAD>
```

```
<SCRIPT LANGUAGE="VBScript">
```

```
<!--
```

```
Function CreateVBAArray()
```

```
    Dim i, j, k
```

```
    Dim a(2, 2)
```

```
    k = 1
```

```
    For i = 0 To 2
```

```
        For j = 0 To 2
```

```
            a(j, i) = k
```

```
            k = k + 1
```

```
        Next
```

```
    Next
```

```
    CreateVBAArray = a
```

```
End Function
```

```
-->
```

```
</SCRIPT>
```

```
<SCRIPT LANGUAGE="JScript">
```

```
<!--
```

```
function VBAArrayTest(vba)
```

```
{
```

```
    var i, s;
```

```
    var a = new VBAArray(vba);
```

```
    for (i = 1; i <= a.dimensions(); i++)
```

```
    {
```

```
        s = "The upper bound of dimension  
";
```

```
        s += i + " is ";
```

```
        s += a.ubound(i) + ".<BR>";
```

```
    }
```

```
    return(s);
```

```
}
```

```
-->
```

```
</SCRIPT>
```

```
</HEAD>
```

```
<BODY>
```

```
<SCRIPT language="jscript">
```

```
    document.write(VBAArrayTest(CreateVBA
```



```
rray() ) ) ;
```

```
</SCRIPT>
```

```
</BODY>
```

要求

[版本 3](#)(See 2.64)

请参阅

[getItem 方法](#)(See 1.2.7.2.10) | [lbound 方法](#)(See 1.2.7.3.3) | [toArray 方法](#)(See 1.2.7.5.3) | [ubound 方法](#)(See 1.2.7.5.19)

应用于: [VBAArray 对象](#)(See 1.2.8.15)

1.2.7.1.23 encodeURIComponent 方法

将文本字符串编码为一个有效的统一资源标识符 (URI)。

encodeURIComponent (*URIString*)

必选的 *URIString* 参数代表一个已编码的 URI。

说明

encodeURIComponent 方法返回一个编码的 URI。如果您将编码结果传递给 **decodeURI**, 那么将返回初始的字符串。**encodeURIComponent** 方法不会对下列字符进行编码: ":", "/", ";", 和 "?". 请使用 **encodeURIComponent** 方法对这些字符进行编码。

要求

[版本 5.5](#)(See 2.64)

请参阅

[decodeURI 方法](#)(See 1.2.7.1.20) | [decodeURIComponent 方法](#)(See 1.2.7.1.21)

应用于: [Global 对象](#)(See 1.2.8.8)

1.2.7.1.24 encodeURIComponentComponent 方法

将文本字符串编码为一个统一资源标识符 (URI) 的一个有效组件。

encodeURIComponentComponent (*encodedURIString*)

必选的 *encodedURIString* 参数代表一个已编码的 URI 组件。

说明

encodeURIComponentComponent 方法返回一个已编码的 URI。如果您将编码结果传递给 **decodeURIComponent**, 那么将返回初始的字符串。因为 **encodeURIComponentComponent** 方法对所有的字符编码, 请注意, 如果该字符串代表一个路径, 例如

/folder1/folder2/default.html，其中的斜杠也将被编码。这样一来，当该编码结果被作为请求发送到 web 服务器时将无效的。如果字符串中包含不止一个 URI 组件，请使用 **encodeURIComponent** 方法进行编码。

要求

[版本 5.5](#)(See 2.64)

请参阅

[decodeURI 方法](#)(See 1.2.7.1.20) | [decodeURIComponent 方法](#)(See 1.2.7.1.21)

应用于：[Global 对象](#)(See 1.2.8.8)

1.2.7.1.25 escape 方法

对 **String** 对象编码以便它们能在所有计算机上可读，

escape (*charString*)

必选项 *charstring* 参数是要编码的任意 **String** 对象或文字。

说明

escape 方法返回一个包含了 *charstring* 内容的字符串值（Unicode 格式）。所有空格、标点、重音符号以及其他非 [ASCII](#) 字符都用 **%xx** 编码代替，其中 *xx* 等于表示该字符的十六进制数。例如，空格返回的是 **"%20"**。

字符值大于 255 的以 **%uxxxx** 格式存储。

注意 **escape** 方法不能够用来对统一资源标示码（URI）进行编码。对其编码应使用 **encodeURIComponent** 和 **encodeURIComponent** 方法。

要求

[版本 1](#)(See 2.64)

请参阅

[encodeURIComponent 方法](#)(See 1.2.7.1.23) | [encodeURIComponent 方法](#)(See 1.2.7.1.24) | [String 对象](#)(See 1.2.8.14) | [unescape 方法](#)(See 1.2.7.5.20)

应用于：[Global 对象](#)(See 1.2.8.8)

1.2.7.1.26 eval 方法

检查 JScript 代码并执行。

eval (*codeString*)

必选项 *codestring* 参数是包含有效 JScript 代码的字符串值。这个字符串将由 JScript 分析器进行分析和执行。

说明

eval 函数允许 JScript 源代码的动态执行。例如,下面的代码创建了一个包含 **Date** 对象的新变量 *mydate* :

```
eval("var mydate = new Date();");
```

传递给 **eval** 方法的代码执行时的上下文和调用 **eval** 方法的一样。

要求

[版本 1](#)(See 2.64)

请参阅

[String 对象](#)(See 1.2.8.14)

应用于: [Global 对象](#)(See 1.2.8.8)

1.2.7.1.27 exec 方法

用正则表达式模式在字符串中运行查找,并返回包含该查找结果的一个数组。

rgExp.**exec**(*str*)

参数

rgExp

必选项。包含正则表达式模式和可用标志的**正则表达式**对象。

str

必选项。要在其中执行查找的 **String** 对象或字符串文字。

说明

如果 **exec** 方法没有找到匹配,则它返回 **null**。如果它找到匹配,则 **exec** 方法返回一个数组,并且更新全局 **RegExp** 对象的属性,以反映匹配结果。数组的 0 元素包含了完整的匹配,而第 1 到 *n* 元素中包含的是匹配中出现的任意一个子匹配。这相当于没有设置全局标志 (g) 的 **match** 方法。

如果为正则表达式设置了全局标志, **exec** 从以 **lastIndex** 的值指示的位置开始查找。如果没有设置全局标志, **exec** 忽略 **lastIndex** 的值,从字符串的起始位置开始搜索。

exec 方法返回的数组有三个属性,分别是 **input**、**index** 和 **lastIndex**。**Input** 属性包含了整个被查找的字符串。**Index** 属性中包含了整个被查找字符串中被匹配的子字符串的位置。**LastIndex** 属性中包含了匹配中最后一个字符的下一个位置。

示例

下面的例子举例说明了 **exec** 方法的用法：

```
function RegExpTest() {
    var ver =
Number(ScriptEngineMajorVersion() + "."
+ ScriptEngineMinorVersion())
    if (ver >= 5.5) { // 测试
JScript 的版本。

        var src = "The rain in Spain falls
mainly in the plain.";
        var re = /\w+/g; // 创建正则
表达式模式。
        var arr;
        while ((arr = re.exec(src)) != null)
            document.write(arr.index + "-" +
arr.lastIndex + "\t" + arr);
    }
    else{
        alert("请使用 JScript 的更新版本");
    }
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[match 方法](#)(See 1.2.7.3.7) | [RegExp 对象](#)(See 1.2.8.12) | [正则表达式对象的方法](#)(See

1.2.8.13) | [正则表达式对象的属性](#)(See 1.2.8.13) | [正则表达式的语法](#)(See 1.5.4) | [search 方法](#)(See 1.2.7.4.1) | [test 方法](#)(See 1.2.7.5.2)
应用于: [正则表达式对象](#)(See 1.2.8.13)

1.2.7.1.28 exp 方法

返回 e （自然对数的底）的幂。

Math.exp(*number*)

必选项 *number* 参数是[数值表达式](#)，用来计算 e 的幂。

说明

返回值为 e^{number} 。常数 e 为自然对数的底，约等于 2.178，*number* 是给出的参数。

要求

[版本 1](#)(See 2.64)

请参阅

[E 属性](#)(See 1.2.10.6) | [Math 对象方法](#)(See 1.2.8.9)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.2 F-I

1.2.7.2.1 fixed 方法

把 HTML <TT> 标记放置在 **String** 对象中的文本两端。

strVariable.fixed()

必选项 *strVariable* 参数为任意的 **String** 对象或文字。

说明

下面的示例说明了 **fixed** 方法如何工作：

```
var strVariable = "This is a string  
object";
```

```
strVariable = strVariable.fixed( );
```

最后一条语句执行完后，*strVariable* 的值是：

```
<TT>This is a string object</TT>
```

不检查该标记是否已经被应用于该字符串了。

要求

[版本 1](#)(See 2.64)

请参阅

[String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.2.2 floor 方法

返回小于等于其数值参数的最大整数。

Math.floor(*number*)

必选项 *number* 参数是[数值表达式](#)。

说明

返回值为小于等于其数值参数的最大整数值。

要求

[版本 1](#)(See 2.64)

请参阅

[ceil 方法](#)(See 1.2.7.1.13) | [Math 对象方法](#)(See 1.2.8.9)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.2.3 fontcolor 方法

把带有 COLOR 属性的一个 HTML 标记放置在 **String** 对象中的文本两端。

strVariable.**fontcolor**(*colorVal*)

参数

strVariable

必选项。任意的 **String** 对象或文字。

colorVal

必选项。为包含颜色值的字符串值。可以是颜色的十六进制值，或预先定义好的颜色名字。

说明

下面的示例说明了 **fontcolor** 方法：

```
var strVariable = "This is a string";
```

```
strVariable =
```

```
strVariable.fontcolor("red");
```

最后一条语句执行完后，*strVariable* 的值是：

```
<FONT COLOR="RED">This is a string</FONT>
```

有效的预定义颜色名取决于 JScript 主机（浏览器、服务器，等等）。它们也可能由主机版本的不同而不同。详细信息请查阅主机的说明书。

不检查该标记是否已经被应用于该字符串了。

要求

[版本 1](#)(See 2.64)

请参阅

[fontsize 方法](#)(See 1.2.7.2.4) | [String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14)

应用于：[String 对象](#)(See 1.2.8.14)

1.2.7.2.4 fontsize 方法

把一个带有 SIZE 属性的 HTML 标记放置在 **String** 对象中的文本的两端。

```
strVariable.fontsize(intSize)
```

参数

strVariable

必选项。任意的 **String** 对象或文字。

intSize

必选项。用来指定文本大小的整数值。

说明

下面的示例说明了 **fontsize** 方法的用法：

```
var strVariable = "This is a string";
```

```
strVariable = strVariable.fontsize(-1);
```

最后一条语句执行完后，*strVariable* 的值是：

```
<FONT SIZE="-1">This is a string</FONT>
```

有效的整数值取决于 Microsoft JScript 主机。详细信息请参阅主机的说明书。

不检查该标记是否已经被应用于该字符串了。

要求

[版本 1](#)(See 2.64)

请参阅

[fontcolor 方法](#)(See 1.2.7.2.3) | [String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.2.5 fromCharCode 方法

从一些 Unicode 字符值中返回一个字符串。

String.fromCharCode(*code1*[, *code2*[, ...[, *codeN*]]])

参数

String

必选项。为 **String** 对象。

code1, . . . , *codeN*

可选项。是要转换为字符串的 Unicode 字符值序列。如果没有给出参数, 结果为空字符串。

说明

在调用 **fromCharCode** 前不必创建 **String** 对象。

在下面的例子中, *test* 包含字符串 "plain":

```
var test = String.fromCharCode(112, 108,  
97, 105, 110);
```

要求

[版本 3](#)(See 2.64)

请参阅

[charCodeAt 方法](#)(See 1.2.7.1.15) | [String 对象的方法](#)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.2.6 getDate 方法

返回 **Date** 对象中用[本地时间](#)表示的一个月中的日期值。

`dateObj.getDate()`

必选项 `dateObj` 参数为 **Date** 对象。

说明

要获取用[全球标准时间 \(UTC\)](#)表示的日期值，请使用 **getUTCDate** 方法。
返回值是一个处于 1 到 31 之间的整数，它代表了相应的 **Date** 对象中的日期值。

示例

下面这个例子说明了 **getDate** 方法的用法：

```
function DateDemo() {  
    var d, s = "今天日期是：";  
    d = new Date();  
    s += (d.getMonth() + 1) + "/";  
    s += d.getDate() + "/";  
    s += d.getFullYear();  
    return(s);  
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getUTCDate 方法](#)(See 1.2.7.2.17) | [setDate 方法](#)(See 1.2.7.4.2) | [setUTCDate 方法](#)(See 1.2.7.4.10)

应用于：[Date 对象](#)(See 1.2.8.4)

1.2.7.2.7 getDay 方法

返回 **Date** 对象中用[本地时间](#)表示的一周中的日期值。

`dateObj.getDay()`

必选项 `dateObj` 参数为 **Date** 对象。

说明

要获取用[全球标准时间 \(UTC\)](#)表示的一周中日期值，请使用 **getUTCDay** 方法。
getDay 方法所返回的值是一个处于 0 到 6 之间的整数，它代表了一周中的某一天，返回值与一周中日期的对应关系如下：

值	星期
0	星期天
1	星期一
2	星期二
3	星期三
4	星期四
5	星期五
6	星期六

下面这个例子说明了 **getDay** 方法的用法。

```
function DateDemo() {
    var d, day, x, s = "今天是： ";
    var x = new Array("星期日", "星期一", "星期二",
        "星期三", "星期四", "星期五", "星期六");
    d = new Date();
    day = d.getDay();
    return(s += x[day]);
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getUTCDay 方法](#)(See 1.2.7.2.18)

应用于： [Date 对象](#)(See 1.2.8.4)

1.2.7.2.8 getFullYear 方法

返回 **Date** 对象中用[本地时间](#)表示的年份值。

dateObj.getFullYear()

必选项 *dateObj* 参数为 **Date** 对象。

说明

要获取用[全球标准时间 \(UTC\)](#)表示的年份值，请使用 **getUTCFullYear** 方法。
getFullYear 方法以绝对数字的形式返回年份值。例如，1976 年的返回值就是 1976。这样可以避免出现 2000 年问题，从而不会将 2000 年 1 月 1 日以后的日期与 1900 年 1 月 1 日以后的日期混淆起来。

下面这个例子说明了 **getFullYear** 方法的用法。

```
function DateDemo() {
    var d, s = "今天 UTC 日期是： ";
    d = new Date();
    s += (d.getMonth() + 1) + "/";
    s += d.getDate() + "/";
    s += d.getFullYear();
    return(s);
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getUTCFullYear 方法](#)(See 1.2.7.2.19) | [setFullYear 方法](#)(See 1.2.7.4.3) | [setUTCFullYear 方法](#)(See 1.2.7.4.11)

应用于： [Date 对象](#)(See 1.2.8.4)

1.2.7.2.9 getHours 方法

返回 **Date** 对象中用[本地时间](#)表示的小时值。

dateObj.**getHours**()

必选项 *dateObj* 参数为 **Date** 对象。

说明

要获取用 [全球标准时间 \(UTC\)](#)表示的小时值，请使用 **getUTCHours** 方法。
getHours 方法返回一个处于 0 到 23 之间的整数，这个值表示从午夜开始计算的小时数。在下面两种情况下此方法的返回值是 0：时间在 1:00:00 am 之前，或者在创建 **Date** 对象的时候没有将时间保存在该对象中。而要确定究竟是哪种情

况, 唯一的方法就是进一步检查分钟和秒钟值是否也是 0。如果这两个值都是 0, 那就几乎可以确定是没有将时间值保存到 **Date** 对象中。

下面这个例子说明了 **getHours** 方法的用法。

```
function TimeDemo() {
    var d, s = "当前本地时间为: ";
    var c = ":";
    d = new Date();
    s += d.getHours() + c;
    s += d.getMinutes() + c;
    s += d.getSeconds() + c;
    s += d.getMilliseconds();
    return(s);
}
```

要求

[版本 1](#) (See 2.64)

请参阅

[Date 对象的方法](#) (See 1.2.8.4) | [getUTCHours 方法](#) (See 1.2.7.2.20) | [setHours 方法](#) (See 1.2.7.4.4) | [setUTCHours 方法](#) (See 1.2.7.4.12)

应用于: [Date 对象](#) (See 1.2.8.4)

1.2.7.2.10 getItem 方法

返回指定位置的项。

safeArray.**getItem**(*dimension1*[, *dimension2*, ...], *dimensionN*)

参数

safeArray

必选项。是一个 **VBAArray** 对象。

dimension1, ..., *dimensionN*

指定所需的 VBAArray 元素的确切位置。 n 等于 VBAArray 的维数。

示例

下面的示例包括三个部分。第一部分是用来创建一个 Visual Basic 安全数组的 VBScript 代码。第二部分是 JScript 代码，遍历该安全数组并打印每个元素的内容。这两部分都放在 HTML 页中的 <HEAD> 部分。第三部分是位于 <BODY> 部分的 JScript 代码，用来运行其他两个部分。

```
<HEAD>
```

```
<SCRIPT LANGUAGE="VBScript">
```

```
<!--
```

```
Function CreateVBAArray()
```

```
    Dim i, j, k
```

```
    Dim a(2, 2)
```

```
    k = 1
```

```
    For i = 0 To 2
```

```
        For j = 0 To 2
```

```
            a(i, j) = k
```

```
            document.writeln(k)
```

```
            k = k + 1
```

```
        Next
```

```
        document.writeln("<BR>")
```

```
    Next
```

```
    CreateVBAArray = a
```

```
End Function
```

```
-->
```

```
</SCRIPT>

<SCRIPT LANGUAGE="JScript">

<!--

function GetItemTest(vbarray)
{
    var i, j;
    var a = new VBArray(vbarray);
    for (i = 0; i <= 2; i++)
    {
        for (j =0; j <= 2; j++)
        {
            document.writeln(a.getItem(i,
j));
        }
    }
}-->

</SCRIPT>

</HEAD>

<ltBODY>

<SCRIPT LANGUAGE="JScript">

<!--

    GetItemTest(CreateVBArray());
```

```
-->  
  
</SCRIPT>  
  
</BODY>
```

要求

[版本 1](#)(See 2.64)

请参阅

[dimensions 方法](#)(See 1.2.7.1.22) | [lbound 方法](#)(See 1.2.7.3.3) | [toArray 方法](#)(See 1.2.7.5.3) | [ubound 方法](#)(See 1.2.7.5.19)
应用于: [VBAArray 对象](#)(See 1.2.8.15)

1.2.7.2.11 getMilliseconds 方法

返回 **Date** 对象中用[本地时间](#)表示的毫秒值。

dateObj.**getMilliseconds()**

必选项 *dateObj* 参数为 **Date** 对象。

说明

要获取用[全球标准时间 \(UTC\)](#)表示的毫秒值, 请使用 **getUTCMilliseconds** 方法。
所返回的毫秒值处于 0-999 之间。

示例

下面这个例子说明了 **getMilliseconds** 方法的用法:

```
function TimeDemo() {  
  
    var d, s = "当前本地时间为: ";  
  
    var c = ":";  
  
    d = new Date();  
  
    s += d.getHours() + c;  
  
    s += d.getMinutes() + c;  
  
    s += d.getSeconds() + c;  
  
    s += d.getMilliseconds();  
}
```

```

    return (s) ;

}

```

要求

[版本 3](#) (See 2.64)

请参阅

[Date 对象的方法](#) (See 1.2.8.4) | [getUTCMilliseconds 方法](#) (See 1.2.7.2.21) | [setMilliseconds 方法](#) (See 1.2.7.4.5) | [setUTCMilliseconds 方法](#) (See 1.2.7.4.13)
 应用于: [Date 对象的方法](#) (See 1.2.8.4) | [getUTCMilliseconds 方法](#) (See 1.2.7.2.21) | [setMilliseconds 方法](#) (See 1.2.7.4.5) | [setUTCMilliseconds 方法](#) (See 1.2.7.4.13)

1.2.7.2.12 getMinutes 方法

返回 **Date** 对象中用[本地时间](#)表示的分钟值。

dateObj.**getMinutes**()

必选项 *dateObj* 参数为 **Date** 对象。

说明

要获取用[全球标准时间 \(UTC\)](#)表示的分钟值, 请使用 **getUTCMinutes** 方法。
getMinutes 方法返回一个处于 0 到 59 之间的整数, 返回值就等于保存在 **Date** 对象中的分钟值。在下面两种情况下返回值为 0: 在时钟整点之后经过的时间少于一分钟, 或者是在创建 **Date** 对象的时候没有将时间值保存到该对象中。而要确定究竟是哪种情况, 唯一的方法是同时检查小时和秒钟值是否也是 0。如果这两个值都是 0, 那就几乎可以确定是没有将时间值保存到该 **Date** 对象中。

示例

下面这个例子说明了 **getMinutes** 方法的用法:

```

function TimeDemo () {

    var d, s = "当前本地时间为: ";

    var c = ":";

    d = new Date();

    s += d.getHours() + c;

    s += d.getMinutes() + c;

    s += d.getSeconds() + c;
}

```



```

    s += d.getMilliseconds();

    return(s);
}

```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getUTCMinutes 方法](#)(See 1.2.7.2.22) | [setMinutes 方法](#)(See 1.2.7.4.6) | [setUTCMinutes 方法](#)(See 1.2.7.4.14)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.2.13 getMonth 方法

返回 **Date** 对象中用[本地时间](#)表示的月份值。

dateObj.getMonth()

必选项 *dateObj* 参数为 **Date** 对象。

说明

要获取用[全球标准时间 \(UTC\)](#)表示的月份值, 请使用 **getUTCMonth** 方法。

getMonth 方法返回一个处于 0 到 11 之间的整数, 它代表 **Date** 对象中的月份值。这个整数并不等于按照惯例来表示月份的数字, 而是要比按惯例表示的值小 1。如果一个 **Date** 对象中保存的时间值是 "Jan 5, 1996 08:47:00", 那么 **getMonth** 方法就会返回 0。

示例

下面这个例子说明了 **getMonth** 方法的用法:

```

function DateDemo() {

    var d, s = "今天日期是: ";

    d = new Date();

    s += (d.getMonth() + 1) + "/";

    s += d.getDate() + "/";

    s += d.getFullYear();

    return(s);
}

```

```
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getUTCMonth 方法](#)(See 1.2.7.2.23) | [setMonth 方法](#)(See 1.2.7.4.7) | [setUTCMonth 方法](#)(See 1.2.7.4.15)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.2.14 getSeconds 方法

返回 **Date** 对象中用[本地时间](#)表示的秒钟值。

dateObj.**getSeconds**()

必选项 *dateObj* 参数 **Date** 对象。

说明

要获取用[全球标准时间 \(UTC\)](#)表示的秒钟值, 请使用 **getUTCSeconds** 方法。

getSeconds 方法返回一个处于 0 到 59 之间的整数, 它表示了相应的 **Date** 对象中的秒钟值。在下面两种情况下, 返回值为 0。第一种情况是在当前的一分钟中所经过的时间少于一秒。另外一种情况是在创建 **Date** 对象时没有将时间值保存到该对象中。而为了确定究竟属于哪种情况, 唯一的方法是同时检查小时和分钟值是否也都是 0。如果这两个值都是 0, 那就几乎可以确定是没有将时间值保存到 **Date** 对象中。

示例

下面这个例子说明了 **getSeconds** 方法的用法:

```
function TimeDemo() {
    var d, s = "当前本地时间为: ";
    var c = ":";
    d = new Date();
    s += d.getHours() + c;
    s += d.getMinutes() + c;
    s += d.getSeconds() + c;
    s += d.getMilliseconds();
}
```

```
    return (s) ;  
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getUTCSeconds 方法](#)(See 1.2.7.2.24) | [setSeconds 方法](#)(See 1.2.7.4.8) | [setUTCSeconds 方法](#)(See 1.2.7.4.16)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.2.15 getTime 方法

返回 **Date** 对象中的时间值。

dateObj.**getTime**()

必选项 *dateObj* 参数为 **Date** 对象。

说明

getTime 方法返回一个整数值, 这个整数代表了从 1970 年 1 月 1 日开始计算到 **Date** 对象中的时间之间的毫秒数。日期的范围大约是 1970 年 1 月 1 日午夜的前后各 285,616 年。负数代表 1970 年之前的日期。

在进行各种日期和时间换算的时候, 可以定义一些变量来表示一天、一个小时或一分钟中包含的毫秒数。这样做通常是很有帮助的。例如:

```
var MinMilli = 1000 * 60  
  
var HrMilli = MinMilli * 60  
  
var DyMilli = HrMilli * 24
```

示例

下面这个例子说明了 **getTime** 方法的用法:

```
function GetTimeTest() {  
    var d, s, t;  
  
    var MinMilli = 1000 * 60;  
  
    var HrMilli = MinMilli * 60;  
  
    var DyMilli = HrMilli * 24;
```

```

    d = new Date();

    t = d.getTime();

    s = "It's been "

    s += Math.round(t / DyMilli) + " days
since 1/1/70";

    return(s);

}

```

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [setTime 方法](#)(See 1.2.7.4.9)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.2.16 getTimezoneOffset 方法

返回用分钟表示的主计算机上的时间和[全球标准时间 \(UTC\)](#)之间的差别。

dateObj.getTimezoneOffset()

必选项 *dateObj* 参数为 **Date** 对象。

说明

getTimezoneOffset 方法返回一个整数值, 这个整数代表了当前计算机上的时间和 UTC 之间相差的分钟数。这些值和执行脚本的计算机相关。如果这个方法被一个服务器脚本调用, 那返回值和服务器相关。而如果这个方法被一个客户机脚本调用, 那么返回值就根据客户机上的时间来确定。

如果所在位置的时间落后于 UTC (比如, Pacific Daylight Time), 那么这个值就是正值, 而如果所在位置的时间超前于 UTC (比如说, Japan), 那么这个值就是负值。

例如, 假设在 12 月 1 日, 一台位于 Los Angeles 的客户机与一台位于 New York City 的服务器进行联络。如果在客户机上执行, **getTimezoneOffset** 方法将返回 480, 而如果在服务器上执行此方法将返回 300。

示例

下面这个例子说明了 **getTimezoneOffset** 方法的用法。

```
function TZDemo() {
```

```
var d, tz, s = " The current local  
time is ";  
d = new Date();  
tz = d.getTimezoneOffset();  
if (tz < 0)  
    s += tz / 60 + " hours before GMT";  
else if (tz == 0)  
    s += "GMT";  
else  
    s += tz / 60 + " hours after GMT";  
return(s);  
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.2.17 getUTCDate 方法

返回 **Date** 对象中用[全球标准时间 \(UTC\)](#)表示的日期。

dateObj.getUTCDate()

必选项 *dateObj* 参数为 **Date** 对象。

说明

要获取用[本地时间](#)表示的日期, 请使用 **getDate** 方法。

返回值是一个处于 1 到 31 之间的整数值, 这个整数代表了 **Date** 对象中的日期值。

示例

下面这个例子说明了 **getUTCDate** 方法的用法。

```
function UTCDateDemo() {
    var d, s = "今天 UTC 日期是： ";
    d = new Date();
    s += (d.getUTCMonth() + 1) + "/";
    s += d.getUTCDate() + "/";
    s += d.getUTCFullYear();
    return(s);
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getDate 方法](#)(See 1.2.7.2.6) | [setDate 方法](#)(See 1.2.7.4.2) | [setUTCDate 方法](#)(See 1.2.7.4.10)

应用于： [Date 对象](#)(See 1.2.8.4)

1.2.7.2.18 getUTCDay 方法

返回 **Date** 对象中用[全球标准时间 \(UTC\)](#)表示的一周中的日期值。

dateObj.**getUTCDay**()

必选项 *dateObj* 参数为 **Date** 对象。

说明

要获取用[本地时间](#)表示的一周中的日期值，请使用 **getDate** 方法。

getUTCDay 方法的返回值是一个处于 0 到 6 之间的整数，它代表了一周中的某一天，返回值与一周中日期的对应关系如下：

值	星期
0	星期天
1	星期一
2	星期二
3	星期三
4	星期四

5	星期五
6	星期六

示例

下面这个例子说明了 `getUTCDay` 方法的使用法。

```
function DateDemo() {
    var d, day, x, s = "今天是";
    var x = new Array("星期日", "星期一", "星期二",
        "星期三", "星期四", "星期五", "星期六");
    d = new Date();
    day = d.getUTCDay();
    return(s += x[day]);
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getDay 方法](#)(See 1.2.7.2.7)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.2.19 getUTCFullYear 方法

返回 **Date** 对象中用[全球标准时间 \(UTC\)](#)表示的年份值。

`dateObj.getUTCFullYear()`

必选项 `dateObj` 参数 **Date** 对象。

说明

要获取用 [本地时间](#)表示的年份值, 请使用 `getFullYear` 方法。

`getUTCFullYear` 方法以绝对数字的形式返回年份值。这样可以避免出现 2000 年

问题，从而不会将 2000 年 1 月 1 日以后的日期与 1900 年 1 月 1 日以后的日期混淆起来。

示例

下面这个例子说明了 **getUTCFullYear** 方法的用法。

```
function UTCDateDemo() {  
    var d, s = "今天 UTC 日期是：";  
    d = new Date();  
    s += (d.getUTCMonth() + 1) + "/";  
    s += d.getUTCDate() + "/";  
    s += d.getUTCFullYear();  
    return(s);  
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getFullYear 方法](#)(See 1.2.7.2.8) | [setFullYear 方法](#)(See 1.2.7.4.3) | [setUTCFullYear 方法](#)(See 1.2.7.4.11)

应用于：[Date 对象](#)(See 1.2.8.4)

1.2.7.2.20 getUTCHours 方法

返回 **Date** 对象中用[全球标准时间 \(UTC\)](#)表示的小时值。

dateObj.**getUTCHours**()

必选项 *dateObj* 参数为 **Date** 对象。

说明

要获取用[本地时间](#)表示的，从午夜开始已经过的小时数，请使用 **getHours** 方法。**getUTCHours** 方法返回一个处于 0 到 23 之间的整数，这个整数代表了从午夜开始所经过的小时数。在下面两种情况下返回值为 0：时间在 1:00:00 A.M. 之前或者在创建 **Date** 对象的时候没有将时间值保存到该对象中。而要确定究竟属于哪种情况，唯一的办法就是同时检查分钟和秒钟值是否都是 0。如果这两个值都是 0，那么就几乎可以确定是没有将时间值保存到 **Date** 对象中。

示例

下面这个例子说明了 **getUTCHours** 方法的用法。

```
function UTCTimeDemo () {  
    var d, s = "当前全球标准时间 (UTC) 是: ";  
    var c = ":";  
    d = new Date();  
    s += d.getUTCHours() + c;  
    s += d.getUTCMinutes() + c;  
    s += d.getUTCSeconds() + c;  
    s += d.getUTCMilliseconds();  
    return(s);  
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getHours 方法](#)(See 1.2.7.2.9) | [setHours 方法](#)(See 1.2.7.4.4) | [setUTCHours 方法](#)(See 1.2.7.4.12)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.2.21 getUTCMilliseconds 方法

返回 **Date** 对象中用[全球标准时间 \(UTC\)](#)表示的毫秒值。

dateObj.getUTCMilliseconds()

必选项 *dateObj* 参数为 **Date** 对象。

说明

要获取用[本地时间](#)表示的毫秒数, 请使用 **getMilliseconds** 方法。
返回的毫秒值的范围是 0-999。

示例

下面这个例子说明了 **getUTCMilliseconds** 方法的用法。

```
function UTCTimeDemo() {  
    var d, s = "当前全球标准时间 (UTC) 是: ";  
    var c = ":";  
    d = new Date();  
    s += d.getUTCHours() + c;  
    s += d.getUTCMinutes() + c;  
    s += d.getUTCSeconds() + c;  
    s += d.getUTCMilliseconds();  
    return(s);  
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getMilliseconds 方法](#)(See 1.2.7.2.11) | [setMilliseconds 方法](#)(See 1.2.7.4.5) | [setUTCMilliseconds 方法](#)(See 1.2.7.4.13)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.2.22 getUTCMinutes 方法

返回 **Date** 对象中用[全球标准时间 \(UTC\)](#)表示的分钟值。

dateObj.getUTCMinutes()

必选项 *dateObj* 参数为 **Date** 对象。

说明

要获得用[本地时间](#)表示的所保存的分钟数, 请使用 **getMinutes** 方法。

getUTCMinutes 方法返回一个处于 0 到 59 之间的整数, 这个整数就等于包含在 **Date** 对象中的分钟数的值。在下列两种情况下返回值为 0: 在整点过后的时间短于 1 分钟, 或者在创建 **Date** 对象的时候没有将时间值保存到对象之中。而要确定究竟属于哪种情况, 唯一的方法就是同时检查小时和秒钟值是否都是 0。如果这两个值都是 0, 那么几乎可以肯定是没有将时间保存到 **Date** 对象中。

示例

下面这个例子说明了 **getUTCMinutes** 方法的用法。

```
function UTCTimeDemo()
{
    var d, s = "当前全球标准时间 (UTC) 是: ";
    var c = ":";
    d = new Date();
    s += d.getUTCHours() + c;
    s += d.getUTCMinutes() + c;
    s += d.getUTCSeconds() + c;
    s += d.getUTCMilliseconds();
    return(s);
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getMinutes 方法](#)(See 1.2.7.2.12) | [setMinutes 方法](#)(See 1.2.7.4.6) | [setUTCMinutes 方法](#)(See 1.2.7.4.14)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.2.23 getUTCMonth 方法

返回 **Date** 对象中用[全球标准时间 \(UTC\)](#)表示的月份值。

dateObj.**getUTCMonth**()

必选项 *dateObj* 参数为 **Date** 对象。

说明

要获得用[本地时间](#)表示的月份, 请使用 **getMonth** 方法。

getUTCMonth 方法返回一个处于 0 到 11 之间的整数, 这个整数就表示 **Date** 对象中的月份值。所返回值并不等于按惯例来表示月份的数字。而是要比按惯例来表示月份的数字小 1。如果一个 **Date** 对象中保存的值是"Jan 5, 1996 08:47:00.0", 那么 **getUTCMonth** 将返回 0。

示例

下面这个例子说明了 **getUTCMonth** 方法的用法。

```
function UTCDateDemo() {  
    var d, s = "今天 UTC 日期是： ";  
    d = new Date();  
    s += (d.getUTCMonth() + 1) + "/";  
    s += d.getUTCDate() + "/";  
    s += d.getUTCFullYear();  
    return(s);  
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getMonth 方法](#)(See 1.2.7.2.13) | [setMonth 方法](#)(See 1.2.7.4.7) | [setUTCMonth 方法](#)(See 1.2.7.4.15)

应用于： [Date 对象](#)(See 1.2.8.4)

1.2.7.2.24 getUTCSeconds 方法

返回 **Date** 对象中用[全球标准时间 \(UTC\)](#)表示的秒钟值。

dateObj.getUTCSeconds()

必选项 *dateObj* 参数为 **Date** 对象。

说明

要获得用[本地时间](#)表示的秒钟数，请使用 **getSeconds** 方法。

getUTCSeconds 方法返回一个处于 0 到 59 之间的整数，这个整数表示相应的 **Date** 对象中的秒钟数。在下面两种情况下返回值为 0：在当前的一分钟中所经过的时间短于 1 秒钟，或者在创建 **Date** 对象的时候没有将时间保存到该对象中。而为了确定究竟属于哪种情况，唯一的方法就是同时检查分钟和小时值是否为 0。如果这两个值也都是 0，那么几乎可以确定是没有将时间保存到 **Date** 对象中。

示例

下面这个例子说明了 **getUTCSeconds** 方法的用法。

```
function UTCTimeDemo() {  
    var d, s = "当前 UTC 时间为: ";  
    var c = ":";  
    d = new Date();  
    s += d.getUTCHours() + c;  
    s += d.getUTCMinutes() + c;  
    s += d.getUTCSeconds() + c;  
    s += d.getUTCMilliseconds();  
    return(s);  
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getSeconds 方法](#)(See 1.2.7.2.14) | [setSeconds 方法](#)(See 1.2.7.4.8) | [setUTCSeconds 方法](#)(See 1.2.7.4.16)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.2.25 getVarDate 方法

返回 **Date** 对象中的 VT_DATE 值。

dateObj.**getVarDate**()

必选项 *dateObj* 参数为 **Date** 对象。

说明

当与 COM 对象、ActiveX(R) 对象或其他以 VT_DATE 格式接受和返回日期值的对象（例如 Visual Basic 和 VBScript）进行交互时，使用 **getVarDate** 方法。实际格式取决于地区设置，不随 JScript 变化。

要求

[版本 3](#)(See 2.64)

请参阅

[getDate 方法](#)(See 1.2.7.2.6) | [parse 方法](#)(See 1.2.7.3.12)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.2.26 getYear 方法

返回 **Date** 对象中的年份值。

`dateObj.getYear()`

必选项 `dateObj` 参数为 **Date** 对象。

说明

这个方法已经过时，之所以提供这个方法，是为了保持向后的兼容性。请改用 **getFullYear** 方法。

对于 1900-1999 这段时间而言，返回的年份值是一个两位数字的整数，它代表了所保存的年份与 1900 年之间的差距。而对于其它的年份，返回值是一个四位的整数。例如，1996 年的返回值是 96，而 1825 和 2025 年的返回值则相应地为 1825 和 2025。

注意 对于 JScript 1.0 版，**getYear** 返回的值始终为 **Date** 对象中的年份与 1900 年之间的差距。例如，1899 年的返回值是 -1，而 2000 年的返回值是 100。

示例

下面这个例子说明了 **getYear** 方法的用法。

```
function DateDemo() {  
    var d, s = "今天日期是：";  
    d = new Date();  
    s += (d.getMonth() + 1) + "/";  
    s += d.getDate() + "/";  
    s += d.getYear();  
    return(s);  
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getFullYear 方法](#)(See 1.2.7.2.8) | [getUTCFullYear 方法](#)(See 1.2.7.2.19) | [setFullYear 方法](#)(See 1.2.7.4.3) | [setUTCFullYear 方法](#)(See 1.2.7.4.11) | [setYear 方法](#)(See 1.2.7.4.17)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.2.27 indexOf 方法

返回 **String** 对象内第一次出现子字符串的字符位置。

strObj.**indexOf**(*subString*[, *startIndex*])

参数

strObj

必选项。**String** 对象或文字。

subString

必选项。要在 **String** 对象中查找的子字符串。

starIndex

可选项。该整数值指出在 **String** 对象内开始查找的索引。如果省略，则从字符串的开始处查找。

说明

indexOf 方法返回一个整数值，指出 **String** 对象内子字符串的开始位置。如果没有找到子字符串，则返回 -1。

如果 *startindex* 是负数，则 *startindex* 被当作零。如果它比最大的字符位置索引还大，则它被当作最大的可能索引。

从左向右执行查找。否则，该方法与 **lastIndexOf** 相同。

示例

下面的示例说明了 **indexOf** 方法的用法。

```
function IndexDemo(str2) {
    var str1 = "BABEBIBOBUBABEBIBOBU"
    var s = str1.indexOf(str2);
    return(s);
}
```

```
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[lastIndexOf 方法](#)(See 1.2.7.3.2) | [String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.2.28 isFinite 方法

返回一个 Boolean 值,指明所提供的数字是否是有限的。

isFinite(*number*)

必选项 *number* 参数为任意的数值。

说明

如果 *number* 不是 NaN、负无穷或正无穷,那么 **isFinite** 方法将返回 **true**。如果是这三种情况,函数返回 **false**。

要求

[版本 3](#)(See 2.64)

请参阅

[isNaN 方法](#)(See 1.2.7.2.29)

应用于: [Global 对象](#)(See 1.2.8.8)

1.2.7.2.29 isNaN 方法

返回一个 Boolean 值,指明提供的值是否是保留值 NaN (不是数字)。

isNaN(*numValue*)

必选项 *numvalue* 参数为要检查是否为 NaN 的值。

说明

如果值是 NaN,那么 **isNaN** 函数返回 **true**,否则返回 **false**。使用这个函数的典型情况是检查 **parseInt** 和 **parseFloat** 方法的返回值。还有一种办法,变量可以与它自身进行比较。如果比较的结果不等,那么它就是 NaN。这是因为 NaN 是唯一与自身不等的值。

要求

[版本 1](#)(See 2.64)

请参阅

[isFinite 方法](#)(See 1.2.7.2.28) | [NaN 属性 \(Global\)](#)(See 1.2.10.28) | [parseFloat 方法](#)(See 1.2.7.3.13) | [parseInt 方法](#)(See 1.2.7.3.14)
应用于: [Global 对象](#)(See 1.2.8.8)

1.2.7.2.30 italics 方法

把 HTML <I> 标记放置在 **String** 对象中的文本两端。

```
strVariable.italics( )  
"String Literal".italics( )
```

说明

下面的示例说明了 **italics** 方法是如何工作的:

```
var strVariable = "This is a string";  
  
strVariable = strVariable.italics( );
```

最后一条语句执行完后, *strVariable* 的值是:

```
<I>This is a string</I>
```

不检查该标记是否已经被应用于该字符串了。

要求

[版本 1](#)(See 2.64)

请参阅

[bold 方法](#)(See 1.2.7.1.11) | [String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14)
应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.2.31 item 方法

返回集合中的当前项。

```
enumObj.item()
```

必选项 *myEnum* 参数为任意 **Enumerator** 对象。

说明

item 方法返回当前项。 如果集合为空或者当前项没有定义, 那么将返回 **undefined** 。

示例

在下面的代码中, 使用了 **item** 方法返回 **Drives** 集合中的一个成员。

```
function ShowDriveList() {
    var fso, s, n, e, x;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    e = new Enumerator(fso.Drives);
    s = "";
    for (; !e.atEnd(); e.moveNext())
    {
        x = e.item();
        s = s + x.DriveLetter;
        s += " - ";
        if (x.DriveType == 3)
            n = x.ShareName;
        else if (x.IsReady)
            n = x.VolumeName;
        else
            n = "[ 驱动器未就绪 ] ";
        s += n + "<br>";
    }
    return(s);
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[atEnd 方法](#)(See 1.2.7.1.8) | [moveFirst 方法](#)(See 1.2.7.3.10) | [moveNext 方法](#)(See 1.2.7.3.11)

应用于: [Enumerator 对象](#)(See 1.2.8.5)

1.2.7.3 J-R

1.2.7.3.1 join 方法

返回字符串值，其中包含了连接到一起的数组的所有元素，元素由指定的分隔符分隔开来。

arrayObj.**join**(*separator*)

参数

arrayObj

必选项。**Array** 对象。

separator

必选项。是一个 **String** 对象，作为最终的 **String** 对象中对数组元素之间的分隔符。如果省略了这个参数，那么数组元素之间就用一个逗号来分隔。

说明

如果数组中有元素没有定义或者为 **null**，将其作为空字符串处理。

示例

下面这个例子说明了 **join** 方法的用法。

```
function JoinDemo() {  
    var a, b;  
    a = new Array(0,1,2,3,4);  
    b = a.join("-");  
}
```

```
    return (b) ;  
}
```

要求

[版本 2](#)(See 2.64)

请参阅

[Array 对象的方法](#)(See 1.2.8.2) | [String 对象](#)(See 1.2.8.14)

应用于: [Array 对象](#)(See 1.2.8.2)

1.2.7.3.2 lastIndexOf 方法

返回 **String** 对象中子字符串最后出现的位置。

strObj.**lastIndexOf**(*substring*[, *startindex*])

参数

strObj

必选项。**String** 对象或文字。

substring

必选项。要在 **String** 对象内查找的子字符串。

startindex

可选项。该整数值指出在 **String** 对象内进行查找的开始索引位置。如果省略，则查找从字符串的末尾开始。

说明

lastIndexOf 方法返回一个整数值，指出 **String** 对象内子字符串的开始位置。如果没有找到子字符串，则返回 -1。

如果 *startindex* 是负数，则 *startindex* 被当作零。如果它比最大字符位置索引还大，则它被当作最大的可能索引。

从右向左执行查找。否则，该方法和 **indexOf** 相同。

下面的示例说明了 **lastIndexOf** 方法的用法：

```
function lastIndexDemo(str2)  
{
```

```

var str1 = "BABEBIBOBUBABEBIBOBU"

var s = str1.lastIndexOf(str2);

return(s);

}

```

要求

[版本 1](#)(See 2.64)

请参阅

[indexOf 方法](#)(See 1.2.7.2.27) | [String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.3.3 lbound 方法

返回在 VBAArray 的指定维中所使用的最小索引值。

`safeArray.lbound(dimension)`

参数

`safeArray`

必选项。是一个 VBAArray 对象。

`dimension`

可选项。要获知其索引下界的 VBAArray 的维数。如果忽略，`lbound` 将该参数作为 1 进行处理。

说明

如果 VBAArray 为空，`lbound` 方法将返回 [undefined](#)。如果 `dimension` 大于 VBAArray 的维数或为负数，该方法将产生一个“下标越界”的错误。

示例

下面的示例包括三个部分。第一部分是用来创建一个 Visual Basic 安全数组的 VBScript 代码。第二部分是 JScript 代码，确定该安全数组的维数和每一维的下界。由于该安全数组是用 VBScript 而不是 Visual Basic 创建的，因此下界始终为 0。这两部分都放在 HTML 页中的 <HEAD> 部分。第三部分是位于 <BODY> 部分的 JScript 代码，用来运行其他两个部分。

```
<HEAD>

<SCRIPT LANGUAGE="VBScript">

<!--

Function CreateVBArray()
    Dim i, j, k
    Dim a(2, 2)
    k = 1
    For i = 0 To 2
        For j = 0 To 2
            a(j, i) = k
            k = k + 1
        Next
    Next
    CreateVBArray = a
End Function

-->

</SCRIPT>

<SCRIPT LANGUAGE="JScript">
<!--
function VBArrayTest(vba)
{
    var i, s;
    var a = new VBArray(vba);
```

```

    for (i = 1; i <= a.dimensions(); i++)
    {
        s = "The lower bound of dimension ";
        s += i + " is ";
        s += a.lbound(i) + ".<BR>";
        return(s);
    }
}
-->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT language="jscript">

    document.write(VBArrayTest(CreateVBArray() ) ) ;

</SCRIPT>
</BODY>

```

要求

[版本 3](#)(See 2.64)

请参阅

[dimensions 方法](#)(See 1.2.7.1.22) | [getItem 方法](#)(See 1.2.7.2.10) | [toArray 方法](#)(See 1.2.7.5.3) | [ubound 方法](#)(See 1.2.7.5.19)

应用于: [VBArray 对象](#)(See 1.2.8.15)

1.2.7.3.4 link 方法

把一个有 HREF 属性的 HTML 锚点放置在 **String** 对象中的文本两端。

strVariable.**link**(*linkstring*)

"String Literal".**link**(*linkstring*)

linkstring 参数是想要放置在 HTML 锚点的 HREF 属性中的文本。

说明

调用 **link** 方法来创建 **String** 对象外部的超链接。下面就是该方法如何实现这种

功能的一个例子：

```
var strVariable = "This is a hyperlink";
strVariable
strVariable.link("http://www.microsoft.com");
```

最后一条语句执行完后，*strVariable* 的值是：

```
<a href="http://www.microsoft.com">This
is a hyperlink</A>
```

不检查该标记是否已经被应用于该字符串了。

要求

[版本 1](#) (See 2.64)

请参阅

[anchor 方法](#) (See 1.2.7.1.3) | [String 对象的方法](#) (See 1.2.8.14) | [String 对象的属性](#) (See 1.2.8.14)

应用于： [String 对象](#) (See 1.2.8.14)

1.2.7.3.5 localeCompare 方法

返回一个值，指出在当前的区域设置中两个字符串是否相同。

stringVar.localeCompare(stringExp)

参数

stringVar

必选项。一个 **String** 对象后文字。

stringExp

必选项。将与 *stringVar* 进行比较的字符串。

说明

localeCompare 可以对 *stringVar* 和 *stringExp* 进行一个区分区域设置的字符串比较并返回 -1、0 或 +1，这取决于系统中缺省区域设置的排序。

如果 *stringVar* 排序在 *stringExp* 之前，那么 **localeCompare** 返回 -1；如果

stringVar 排序在 *stringExp* 之后，则返回 +1。如果返回值为 0，那就说明这两个字符串是相同的。

要求

[版本 5.5](#)(See 2.64)

请参阅

[toLocalString](#)(See 1.2.7.5.10)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.3.6 log 方法

返回数字的自然对数。

Math.log(*number*)

必选项 *number* 参数是要计算自然对数的数值表达式。

返回值

返回值为 *number* 的自然对数，底为 *e*。

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象方法](#)(See 1.2.8.9)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.3.7 match 方法

使用正则表达式模式对字符串执行查找，并将包含查找的结果作为数组返回。

stringObj.**match**(*rgExp*)

参数

stringObj

必选项。对其进行查找的 **String** 对象或字符串文字。

rgExp

必选项。为包含正则表达式模式和可用标志的**正则表达式**对象。也可以是包含正则表达式模式和可用标志的变量名或字符串文字。

说明

如果 **match** 方法没有找到匹配, 返回 **null**。如果找到匹配返回一个数组并且更新全局 **RegExp** 对象的属性以反映匹配结果。

match 方法返回的数组有三个属性: **input**、**index** 和 **lastIndex**。**Input** 属性包含整个的被查找字符串。**Index** 属性包含了在整个被查找字符串中匹配的子字符串的位置。**LastIndex** 属性包含了最后一次匹配中最后一个字符的下一个位置。

如果没有设置全局标志 (**g**), 数组的 0 元素包含整个匹配, 而第 1 到 **n** 元素包含了匹配中曾出现过的任一个子匹配。这相当于没有设置全局标志的 **exec** 方法。如果设置了全局标志, 元素 0 到 **n** 中包含所有匹配。

示例

下面的示例演示了 **match** 方法的用法:

```
function MatchDemo() {
    var r, re;                // 声明变量。

    var s = "The rain in Spain falls mainly
in the plain";

    re = /ain/i;              // 创建正则表达式模式。

    r = s.match(re);          // 尝试匹配搜索字符串。

    return(r);                // 返回第一次出现 "ain" 的地方。
}
```

本示例说明带 **g** 标志设置的 **match** 方法的用法。

```
function MatchDemo() {
    var r, re;                // 声明变量。

    var s = "The rain in Spain falls mainly
in the plain";

    re = /ain/ig;             // 创建正则表达式模式。

    r = s.match(re);          // 尝试去匹配搜索字符串。

    return(r);                // 返回的数组包含了所有 "ain"
                                // 出现的四个匹配。
}
```

下面几行代码演示了字符串文字的 **match** 方法的用法。

```
var r, re = "Spain";

r = "The rain in Spain".replace(re,
"Canada");
```

要求

[版本 3](#)(See 2.64)

请参阅

[exec 方法](#)(See 1.2.7.1.27) | [RegExp 对象](#)(See 1.2.8.12) | [replace 方法](#)(See 1.2.7.3.19) | [search 方法](#)(See 1.2.7.4.1) | [String 对象的方法](#)(See 1.2.8.14) | [test 方法](#)(See 1.2.7.5.2)
应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.3.8 max 方法

返回给出的零个或多个数值表达式中较大者。

Math.max([*number1*[, *number2*[. . . [,*numberN*]]]])

可选项 *number1*, *number2*, . . . , *numberN* 参数为需要进行比较的[数值表达式](#)。

说明

如果没有给出参数, 返回值等于 **NEGATIVE_INFINITY**。如果有参数为 **NaN**, 返回值仍然为 **NaN**。

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象方法](#)(See 1.2.8.9) | [min 方法](#)(See 1.2.7.3.9) | [NEGATIVE_INFINITY 属性](#)(See 1.2.10.29)
应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.3.9 min 方法

返回给出的零个或多个数值表达式中较小的值。

Math.min([*number1*[, *number2*[. . . [,*numberN*]]]])

可选项 *number1*, *number2*, . . . , *numberN* 参数为需要进行比较的[数值表达式](#)。

说明

如果没有给出参数, 返回值等于 **POSITIVE_INFINITY**。如果有参数为 **NaN**, 返回值也为 **NaN**。

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象方法](#)(See 1.2.8.9) | [max 方法](#)(See 1.2.7.3.8) | [POSITIVE INFINITY 属性](#)(See 1.2.10.32)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.3.10 moveFirst 方法

重新将集合中的当前项设置为第一项。

`enumObj.moveFirst( )`

必选项 `enumObj` 参数为任意的 **Enumerator** 对象。

说明

如果集合中没有项, 那么当前项将被设置为 [undefined](#) 。

示例

在下面的例子中, 使用了 **moveFirst** 方法从列表的开始处对 **Drives** 集合的成员进行计算:

```
function ShowFirstAvailableDrive() {
    var fso, s, e, x;                                // 声明变量。
    fso = new ActiveXObject("Scripting.FileSystemObject");
    e = new Enumerator(fso.Drives); // 创建 Enumerator 对象。
    e.moveFirst(); // 移到第一个驱动器。
    s = ""; // 初始化 s
    do
```

```

    {
        x = e.item(); // 测试驱动器的
存在与否。
        if (x.IsReady) // 是否已准备
好。
        {
            s = x.DriveLetter + ":"; // 给第一个驱
动器号赋值给 S。

            break;
        }
        else
            if (e.atEnd()) // 是否
在集合的结尾。
            {
                s = "没有可用的驱动器";

                break;
            }

        e.moveNext(); // 移到下一个驱
动器。
    }

    while (!e.atEnd()); // 执行循
环，直到集合的结尾。

    return(s); // 返回可用驱
动器的列表。
}

```

要求

[版本 3](#)(See 2.64)

请参阅

[atEnd 方法](#)(See 1.2.7.1.8) | [item 方法](#)(See 1.2.7.2.31) | [moveNext 方法](#)(See 1.2.7.3.11)
 应用于: [Enumerator 对象](#)(See 1.2.8.5)

1.2.7.3.11 moveNext 方法

将集合中的当前项移动到下一项。

enumObj.**moveNext**()

必选项 *myEnum* 参数是任意 **Enumerator** 对象。

说明

如果枚举算子位于集合的最后, 或者集合为空, 那么当前项将被设置为 [undefined](#) 。
 在下面的例子中, 使用了 **moveNext** 方法在 **Drives** 集合中向下一个驱动器移动:

```
function ShowDriveList() {
    var fso, s, n, e, x;                                //
    声明变量。
    fso                                =                new
    ActiveXObject("Scripting.FileSystemObject");
    e    =    new    Enumerator(fso.Drives);
    // 创建 Enumerator 对象。
    s = "";                                              // 初始化
    s。
    for (; !e.atEnd(); e.moveNext())
    {
        x = e.item();
        s = s + x.DriveLetter;                          //
        加驱动器号
        s += " - ";                                      // 加
        " - " 字符。
    }
}
```

```

        if (x.DriveType == 3)
            n = x.ShareName;                //
加共享名。

        else if (x.IsReady)
            n = x.VolumeName;                //
加卷名。

        else
            n = "[ 驱动器未就绪 ] ";        // 指明驱动器未就绪。

            s += n + "\n";

    }

    return (s);                            // 返回
驱动器状态。
}

```

要求

[版本 3](#)(See 2.64)

请参阅

[atEnd 方法](#)(See 1.2.7.1.8) | [item 方法](#)(See 1.2.7.2.31) | [moveFirst 方法](#)(See 1.2.7.3.10)

应用于: [Enumerator 对象](#)(See 1.2.8.5)

1.2.7.3.12 parse 方法

解析一个包含日期的字符串，并返回该日期与 1970 年 1 月 1 日午夜之间所间隔的毫秒数。

Date.parse(*dateVal*)

其中必选项 *dateVal* 是一个包含以诸如 "Jan 5, 1996 08:47:00" 的格式表示的日期的字符串，或者是一个从 ActiveX(R) 对象或其他对象中获取的 VT_DATE 值。

说明

parse 方法返回一个整数值，这个整数表示 *dateVal* 中所包含的日期与 1970 年 1 月 1 日午夜之间相间隔的毫秒数。

parse 方法是 **Date** 对象的一个静态方法。正因为它是一个静态方法，它是通过下面例子中所示的方法被调用的，而不是作为一个已创建 **Date** 对象的一个方法被调用。

```
var datestring = "November 1, 1997 10:15 AM";
```

```
Date.parse(datestring)
```

下面这些规则决定了 **parse** 方法能够成功地解析那些字符串：

- 短日期可以使用 “/” 或 “-” 作为日期分隔符，但是必须用月/日/年的格式来表示，例如 "7/20/96"。
- 以 "July 10 1995" 形式表示的长日期中的年、月、日可以按任何顺序排列，年份值可以用 2 位数字表示也可以用 4 位数字表示。如果使用 2 位数字来表示年份，那么该年份必须大于或等于 70。
- 括号中的任何文本都被视为注释。这些括号可以嵌套使用。
- 逗号和空格被视为分隔符。允许使用多个分隔符。
- 月和日的名称必须具有两个或两个以上的字符。如果两个字符所组成的名称不是独一无二的，那么该名称就被解析成最后一个符合条件的月或日。例如，"Ju" 被解释为七月而不是六月。
- 在所提供的日期中，如果所指定的星期几的值与按照该日期中剩余部分所确定的星期几的值不符合，那么该指定值就会被忽略。例如，尽管 1996 年 11 月 9 日实际上是星期五，"Tuesday November 9 1996" 也还是可以被接受并进行解析的。但是结果 **Date** 对象中包含的是 "Friday November 9 1996"。
- JScript 处理所有的标准时区，以及全球标准时间 (UTC) 和格林威治标准时间 (GMT)。
- 小时、分钟、和秒钟之间用冒号分隔，尽管不是这三项都需要指明。"10:"、"10:11"、和 "10:11:12" 都是有效的。
- 如果使用 24 小时计时的时钟，那么为中午 12 点之后的时间指定 "PM" 是错误的。例如 "23:15 PM" 就是错误的。
- 包含无效日期的字符串是错误的。例如，一个包含有两个年份或两个月份的字符串就是错误的。

示例

下面这个例子说明了 **parse** 方法的用法：

```
function GetTimeTest(testdate) {
    var s, t; // 声明变量。
    var MinMilli = 1000 * 60; // 初始化变量。
    var HrMilli = MinMilli * 60;
```



```

    var DyMilli = HrMilli * 24;

    t = Date.parse(testdate);           // 解析
testdate.

    s = "There are "                      // 创建返回的
字符串。

    s += Math.round(Math.abs(t / DyMilli))
+ " days "

    s += "between " + testdate + " and
1/1/70";

    return(s);                           // 返回结果。
}

```

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.3.13 parseFloat 方法

返回由字符串转换得到的浮点数。

parseFloat(*numString*)

必选项 *numString* 参数是包含浮点数的字符串。

说明

parseFloat 方法返回与 *numString* 中保存的数相等的数字表示。如果 *numString* 的前缀不能解释为浮点数，则返回 **NaN**（而不是数字）。

parseFloat("abc") // 返回 NaN。

parseFloat("1.2abc") // 返回 1.2。

可以用 **isNaN** 方法检测 **NaN**。

要求

[版本 1](#)(See 2.64)

请参阅

[isNaN 方法](#)(See 1.2.7.2.29) | [parseInt 方法](#)(See 1.2.7.3.14) | [String 对象](#)(See 1.2.8.14)

应用于: [Global 对象](#)(See 1.2.8.8)

1.2.7.3.14 parseInt 方法

返回由字符串转换得到的整数。

parseInt(*numString*, [*radix*])

参数

numString

必选项。要转换为数字的字符串。

radix

可选项。在 2 和 36 之间的表示 *numString* 所保存数字的进制的值。如果没有提供, 则前缀为 '0x' 的字符串被当作十六进制, 前缀为 '0' 的字符串被当作八进制。所有其它字符串都被当作是十进制的。

说明

parseInt 方法返回与保存在 *numString* 中的数字值相等的整数。如果 *numString* 的前缀不能解释为整数, 则返回 **NaN** (而不是数字)。

parseInt("abc") // 返回 NaN。

parseInt("12abc") // 返回 12。

可以用 **isNaN** 方法检测 **NaN**。

要求

[版本 1](#)(See 2.64)

请参阅

[isNaN 方法](#)(See 1.2.7.2.29) | [parseFloat 方法](#)(See 1.2.7.3.13) | [String 对象](#)(See 1.2.8.14) | [valueOf 方法](#)(See 1.2.7.5.23)

应用于: [Global 对象](#)(See 1.2.8.8)

1.2.7.3.15 pop 方法

移除数组中的最后一个元素并返回该元素。

`arrayObj.pop()`

必选的 `arrayObj` 引用是一个 **Array** 对象。

说明

如果该数组为空，那么将返回 **undefined**。

要求

[版本 5.5](#)(See 2.64)

请参阅

[push 方法](#)(See 1.2.7.3.17)

应用于: [Array 对象](#)(See 1.2.8.2)

1.2.7.3.16 pow 方法

返回底表达式的指定次幂。

`Math.pow(base, exponent)`

参数

base

必选项。表达式底的值。

exponent

必选项。表达式的指数值。

示例

在下面的例子中，等同于 $base^{exponent}$ 的[数值表达式](#)，返回值为 1000。

```
Math.pow(10, 3);
```

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象方法](#)(See 1.2.8.9)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.3.17 push 方法

将新元素添加到一个数组中，并返回数组的新长度值。

```
arrayObj.push([item1 [item2 [. . . [itemN ]]])
```

参数

arrayObj

必选项。一个 **Array** 对象。

item, item2, . . . itemN

可选项。该 **Array** 的新元素。

说明

push 方法将以新元素出现的顺序添加这些元素。如果参数之一为数组，那么该数组将作为单个元素添加到数组中。如果要合并两个或多个数组中的元素，请使用 **concat** 方法。

要求

[版本 5.5](#)(See 2.64)

请参阅

[concat 方法](#)(See 1.2.7.1.17) | [pop 方法](#)(See 1.2.7.3.15)

应用于： [Array 对象](#)(See 1.2.8.2)

1.2.7.3.18 random 方法

返回介于 0 和 1 之间的伪随机数。

```
Math.random( )
```

说明

产生的伪随机数介于 0 和 1 之间（含 0，不含 1），也就是，返回值可能为 0，但总是小于 1。在第一次加载 JScript 时随机数发生器自动产生。

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象方法](#)(See 1.2.8.9)

应用于： [Math 对象](#)(See 1.2.8.9)

1.2.7.3.19 replace 方法

返回根据正则表达式进行文字替换后的字符串的复制。

`stringObj.replace(rgExp, replaceText)`

参数

`stringObj`

必选项。要执行该替换的 **String** 对象或字符串文字。该字符串不会被 **replace** 方法修改。

`rgExp`

必选项。为包含正则表达式模式或可用标志的**正则表达式**对象。也可以是 **String** 对象或文字。如果 *rgExp* 不是**正则表达式**对象，它将被转换为字符串，并进行精确的查找；不要尝试将字符串转化为正则表达式。

`replaceText`

必选项。是一个 **String** 对象或字符串文字，对于 *stringObj* 中每个匹配 *rgExp* 中的位置都用该对象所包含的文字加以替换。在 Jscript 5.5 或更新版本中，*replaceText* 参数也可以是返回替换文本的函数。

说明

replace 方法的结果是一个完成了指定替换的 *stringObj* 对象的复制。

下面任意的匹配变量都能用来识别最新的匹配以及找出匹配的字符串。在需要动态决定替换字符串的文本替换中可以使用匹配变量。

字符	含义
\$	\$ （JScript 5.5 或更新版本）
\$&	指定与整个模式匹配的 <i>stringObj</i> 的部分。（JScript 5.5 或更新版本）
\$`	指定由 \$& 描述的匹配之前的 <i>stringObj</i> 部分。（JScript 5.5 或更新版本）
\$'	指定由 \$& 描述的匹配之后的 <i>stringObj</i> 部分。（JScript 5.5 或更新版本）
\$n	捕获的第 <i>n</i> 个子匹配，此处 <i>n</i> 为从 1 到 9 的十进制一位数。（JScript 5.5 或更新版本）
\$nn	捕获的第 <i>nn</i> 个子匹配，此处 <i>nn</i> 为从 01 到 99 的十进制两位数。（JScript 5.5 或更新版本）

如果 *replaceText* 为函数，对于每一个匹配的子字符串，调用该函数时带有下面的 *m*+3 个参数，此处 *m* 是在 *rgExp* 中捕获的左括弧的个数。第一个参数是匹配的子字符串。接下来的 *m* 个参数是查找中捕获的全部结果。第 *m*+2 个参数是在 *stringObj* 中匹配出现的偏移量，而第 *m*+3 个参数为 *stringObj*。结果为将每一匹

配的子字符串替换为函数调用的相应返回值的字符串值。

Replace 方法更新全局 **RegExp** 对象的属性。

示例

下面的示例演示了 **replace** 方法将第一次出现的单词 "The" 替换为单词 "A" 的用法。

```
function ReplaceDemo() {
    var r, re;                                // 声明变量。

    var ss = "The man hit the ball with
the bat.\n";

    ss += "while the fielder caught the
ball with the glove.";

    re = /The/g;                              // 创建正则表达式模式。

    r = ss.replace(re, "A");                // 用 "A" 替换
"The"。

    return(r);                                // 返回替换后的字符串。
}
```

另外, **replace** 方法也可以替换模式中的子表达式。下面的范例演示了交换字符串中的每一对单词:

```
function ReplaceDemo() {
    var r, re;                                // 声明变量。

    var ss = "The rain in Spain falls
mainly in the plain.";

    re = /(\S+) (\s+) (\S+)/g;                // 创建
正则表达式模式。

    r = ss.replace(re, "$3$2$1");            // 交换每
一对单词。
```

```
return(r); // 返回结果字符串。
```

```
}
```

下面的示例（在 JScript 5.5 及更新版本中执行）执行的是从华氏到摄氏的转换，它演示了使用函数作为 *replaceText*。要想知道该函数是如何工作的，传递一个包含数值的字符串，数值后要紧跟 "F"（例如 "Water boils at 212"）。

```
function f2c(s) {
    var test = /(\d+(\.\d*)?)F\b/g; //
```

初始化模式。

```
    return(s.replace
        (test,
            function($0,$1,$2) {
                return((($1-32) * 5/9) + "C");
            }
        )
    );
}
```

```
document.write(f2c("Water freezes at
32F and boils at 212F."));
```

要求

[版本 1](#)(See 2.64)

请参阅

[exec 方法](#)(See 1.2.7.1.27) | [match 方法](#)(See 1.2.7.3.7) | [RegExp 对象](#)(See 1.2.8.12) | [search 方法](#)(See 1.2.7.4.1) | [String 对象的方法](#)(See 1.2.8.14) | [test 方法](#)(See 1.2.7.5.2)
应用于：[String 对象](#)(See 1.2.8.14)

1.2.7.3.20 reverse 方法

返回一个元素顺序被反转的 **Array** 对象。

`arrayObj.reverse( )`

必选项 `arrayObj` 参数为 **Array** 对象。

说明

reverse 方法将一个 **Array** 对象中的元素位置进行反转。在执行过程中，这个方法并不会创建一个新的 **Array** 对象。

如果数组是不连续的，**reverse** 方法将在数组中创建元素以便填充数组中的间隔。这样所创建的全部元素的值都是 [undefined](#)。

示例

下面这个例子说明了 **reverse** 方法的用法：

```
function ReverseDemo() {  
    var a, l;                                // 声明变量。  
    a = new Array(0,1,2,3,4);                // 创建数组并赋值。  
    l = a.reverse();                          // 反转数组的内容。  
    return(l);                                // 返回结果数组。  
}
```

要求

[版本 2](#)(See 2.64)

请参阅

[Array 对象的方法](#)(See 1.2.8.2)

应用于： [Array 对象](#)(See 1.2.8.2)

1.2.7.3.21 round 方法

返回与给出的[数值表达式](#)最接近的整数。

`Math.round(number)`

必选项 `number` 参数是要舍入到最接近整数的值。

说明

如果 `number` 的小数部分大于等于 0.5，返回值是大于 `number` 的最小整数。否则，**round** 返回小于等于 `number` 的最大整数。

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象方法](#)(See 1.2.8.9)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.4 S

1.2.7.4.1 search 方法

返回与正则表达式查找内容匹配的第一个子字符串的位置。

stringObj.**search**(*rgExp*)

参数

stringObj

必选项。要在其上进行搜索的 **String** 对象或字符串文字。

rgExp

必选项。包含正则表达式模式和可用标志的**正则表达式**对象。

说明

search 方法指明是否存在相应的匹配。如果找到一个匹配, **search** 方法将返回一个整数值, 指明这个匹配距离字符串开始的偏移位置。如果没有找到匹配, 则返回 -1。

示例

下面的示例演示了 **search** 方法的使用。

```
function SearchDemo() {
    var r, re;                                // 声明变量。
    var s = "The rain in Spain falls mainly
in the plain.";
    re = /falls/i;                            // 创建正则表达式模式。
    r = s.search(re);                          // 查找字符串。
    return(r);                                // 返回
```

Boolean 结果。

```
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[exec 方法](#)(See 1.2.7.1.27) | [match 方法](#)(See 1.2.7.3.7) | [正则表达式对象](#)(See 1.2.8.13) | [正则表达式语法](#)(See 1.5.4) | [replace 方法](#)(See 1.2.7.3.19) | [String 对象的方法](#)(See 1.2.8.14) | [test 方法](#)(See 1.2.7.5.2)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.4.2 setDate 方法

设置 **Date** 对象中用[本地时间](#)表示的数字日期。

```
dateObj.setDate(numDate)
```

参数

dateObj

必选项。任意 **Date** 对象。

numDate

必选项。是一个等于数字日期的数值。

说明

要设置用[全球标准时间 \(UTC\)](#)表示的日期值, 请使用 **setUTCDate** 方法。

如果 *numDate* 的值大于 **Date** 对象中所保存的月份的天数或者是负数。那么日期将被设置为由 *numDate* 减去所保存月份中天数而得到的日期。例如, 如果所保存的日期是 1996 年 1 月 5 日, 并且调用了方法 **setUTCDate(32)**, 那么日期将被改变为 1996 年 2 月 1 日。负数的处理方法与此相似。

示例

下面这个例子说明了 **setDate** 方法的使用。

```
function SetDateDemo(newdate) {
    var d, s;                                // 声明变量。
    d = new Date();                            // 创建 date 对
```

象。

```
d.setDate(newdate);           // 设置 date 为
newdate.

s = "Current setting is ";
s += d.toLocaleString();
return(s);                     // 返回新设的日期。
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getDate 方法](#)(See 1.2.7.2.6) | [getUTCDate 方法](#)(See 1.2.7.2.17) | [setUTCDate 方法](#)(See 1.2.7.4.10)
应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.4.3 setFullYear 方法

设置 **Date** 对象中用[本地时间](#)表示的年份值。

dateObj.**setFullYear**(*numYear*[, *numMonth*[, *numDate*]])

参数

dateObj

必选项。任意 **Date** 对象。

numYear

必选项。一个等于年份的数值。

numMonth

可选项。一个等于月份的数值。如果提供了 *numDate*, 那么此项也必须提供。

numDate

可选项。一个等于日期的数值。

说明

如果没有指定可选项，那么所有使用可选项的 **set** 方法都将使用从相应的 **get** 方法返回的数值作为可选项的数值。例如，如果 *numMonth* 参数是可选的，但是没有被指定，JScript 将使用从 **getMonth** 方法返回的值作为该参数的值。

此外，如果参数的值超出其范围或者是负数，其他被保存的值都将相应地得到修改。

要设置用[全球标准时间 \(UTC\)](#)表示的年份，请使用 **setUTCFullYear** 方法。

Date 对象中所支持的年份的范围大约是 1970 年的前后各 285,616 年。

示例

下面这个例子说明了 **setFullYear** 方法的用法。

```
function SetFullYearDemo(newyear) {
    var d, s;                                // 声明变量。
    d = new Date();                          // 创建 Date 对象。
    d.setFullYear(newyear);                 // 设置年份。
    s = "Current setting is ";
    s += d.toLocaleString();
    return(s);                               // 返回新的日期设置。
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getFullYear 方法](#)(See 1.2.7.2.8) | [getUTCFullYear 方法](#)(See 1.2.7.2.19) | [setUTCFullYear 方法](#)(See 1.2.7.4.11)

应用于： [Date 对象](#)(See 1.2.8.4)

1.2.7.4.4 setHours 方法

设置 **Date** 对象中用[本地时间](#)表示的小时值。

dateObj.**setHours**(*numHours*[, *numMin*[, *numSec*[, *numMilli*]])

参数

dateObj

必选项。任意 **Date** 对象。

numHours

必选项。一个等于小时值的数值。

numMin

可选项。一个等于分钟值的数值。如果使用了下面的参数，那么此参数也必须提供。

numSec

可选项。一个等于秒钟值的数值。如果使用了下面的参数，那么此参数也必须提供。

numMilli

可选项。一个等于毫秒值的数值。

说明

如果没有指定可选项，那么所有使用可选项的 **set** 方法都将使用从相应的 **get** 方法返回的数值作为可选项的数值。例如，如果 *numMinutes* 参数是可选的，但是没有被指定，JScript 将使用从 **getMinutes** 方法返回的值作为该参数的值。

要设置用[全球标准时间 \(UTC\)](#)表示的小时值，请使用 **setUTCHours** 方法。

此外，如果参数的值超出其范围或者是负数，其他被保存的值都将相应地得到修改。例如，如果所保存的日期是 "Jan 5, 1996 00:00:00"，且调用了 **setHours(30)** 方法，那么日期将被改变为 "Jan 6, 1996 06:00:00." 负数的处理方法与此相似。

示例

下面这个例子说明了 **setHours** 方法的用法。

```
function SetHoursDemo(nhr, nmin, nsec){
    var d, s;                                // 声明变量。
    d = new Date();                          // 创建 Date
    对象。
    d.setHours(nhr, nmin, nsec);           // 设置小时，
    分钟，秒。
    s = "Current setting is " +
    d.toLocaleString()
```

```

        return (s); // 返回新的日期设置。
    }

```

要求

[版本 3](#) (See 2.64)

请参阅

[Date 对象的方法](#) (See 1.2.8.4) | [getHours 方法](#) (See 1.2.7.2.9) | [getUTCHours 方法](#) (See 1.2.7.2.20) | [setUTCHours 方法](#) (See 1.2.7.4.12)

应用于: [Date 对象](#) (See 1.2.8.4)

1.2.7.4.5 setMilliseconds 方法

设置 **Date** 对象中用[本地时间](#)表示的毫秒值。

dateObj.**setMilliseconds** (*numMilli*)

参数

dateObj

必选项。任意 **Date** 对象。

numMilli

必选项。是一个等于毫秒值的数值。

说明

要设置用[全球标准时间 \(UTC\)](#)表示的毫秒值, 请使用 **setUTCMilliseconds** 方法。如果 *numMilli* 的值大于 999 或者是负数, 所保存的秒钟数 (以及分钟, 小时, 等等, 如果需要的话) 将增加适当的数量。

示例

下面这个例子说明了 **setMilliseconds** 方法的用法:

```

function SetMSecDemo (nmsec) {
    var d, s; // 声明变量。
    var sep = ":"; // 初始化分隔符。
    d = new Date(); // 创建 Date

```

对象。

```
d.setMilliseconds(nmsec);    // 设置毫秒数。

s = "Current setting is ";

s += d.toLocaleString() + sep +

d.getMilliseconds();

return(s);                    // 返回新的日期设置。

}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#) (See 1.2.8.4) | [getMilliseconds 方法](#) (See 1.2.7.2.11) | [getUTCMilliseconds 方法](#) (See 1.2.7.2.21) | [setUTCMilliseconds 方法](#) (See 1.2.7.4.13)
应用于: [Date 对象](#) (See 1.2.8.4)

1.2.7.4.6 setMinutes 方法

设置 **Date** 对象中用 [本地时间](#) 表示的分钟值。

dateObj.**setMinutes**(*numMinutes*[, *numSeconds*[, *numMilli*]])

参数

dateObj

必选项。任意 **Date** 对象。

numMinutes

必选项。一个等于分钟值的数值。

numSeconds

可选项。一个等于秒钟值的数值。如果使用了 *numMilli* 参数, 那么必须提供此参数。

numMilli

可选项。一个等于毫秒值的数值。

说明

如果没有指定可选项，那么所有使用可选项的 **set** 方法都将使用从相应的 **get** 方法返回的数值作为可选项的数值。例如，如果 *numSeconds* 参数是可选项的，但是没有指定，JScript 将使用从 **getSeconds** 方法返回的值作为该参数的值。

要设置用[全球标准时间 \(UTC\)](#)表示的分钟值，请使用 **setUTCMinutes** 方法。

如果参数的值超出其范围或者是负数，其他被保存的值都将相应地得到修改。

例如，如果所保存的日期是 "Jan 5, 1996 00:00:00"，且调用了 **setMinutes(90)** 方法，那么日期将被改变为 "Jan 5, 1996 01:30:00" 负数的处理方法与此相似。

示例

下面这个例子说明了 **setMinutes** 方法的使用。

```
function SetMinutesDemo(nmin, nsec) {
    var d, s;                                // 声明变量。
    d = new Date();                          // 创建 Date
    对象。
    d.setMinutes(nmin, nsec);               // 设置分钟。
    s = "Current setting is " +
    d.toLocaleString()
    return(s);                               // 返回新的设置。
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getMinutes 方法](#)(See 1.2.7.2.12) | [getUTCMinutes 方法](#)(See 1.2.7.2.22) | [setUTCMinutes 方法](#)(See 1.2.7.4.14)

应用于： [Date 对象](#)(See 1.2.8.4)

1.2.7.4.7 setMonth 方法

设置 **Date** 对象中用[本地时间](#)表示的月份值。

dateObj.setMonth(numMonth[, dateVal])

参数

dateObj

必选项。任意 **Date** 对象。

numMonth

必选项。一个等于月份值的数值。

dateVal

可选项。一个代表日期的数值。如果没有提供此参数，那么将使用通过调用 **getDate** 方法而得到的数值。

说明

要设置用[全球标准时间 \(UTC\)](#)表示的月份值，请使用 **setUTCMonth** 方法。如果 *numMonth* 的值大于 11 (月份 0 表示一月)或者是一个负数，那么所保存的年份将相应地得到改变。例如，如果所保存的日期是 "Jan 5, 1996" 并且调用了 **setMonth(14)** 方法，那么该日期就被改变为 "Mar 5, 1997."

示例

下面这个例子说明了 **setMonth** 方法的使用法。

```
function SetMonthDemo(newmonth) {
    var d, s;                                // 声明变量。
    d = new Date();                          // 创建 Date 对象
    d.setMonth(newmonth);                   // 设置月份。
    s = "Current setting is ";
    s += d.toLocaleString();
    return (s);                              // 返回新的设置。
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getMonth 方法](#)(See 1.2.7.2.13) | [getUTCMonth 方法](#)(See 1.2.7.2.23) | [setUTCMonth 方法](#)(See 1.2.7.4.15)

应用于： [Date 对象](#)(See 1.2.8.4)

1.2.7.4.8 setSeconds 方法

设置 **Date** 对象中用 [本地时间](#) 表示的秒钟值。

`dateObj.setSeconds(numSeconds[, numMilli])`

参数

dateObj

必选项。任意 **Date** 对象。

numSeconds

必选项。等于秒钟值的数值。

numMilli

可选项。等于毫秒值的数值。

说明

如果没有指定可选项，那么所有使用可选项的 **set** 方法都将使用从相应的 **get** 方法返回的数值作为可选项的数值。例如，如果 *numMilli* 参数是可选项的，但是没有指定，JScript 将使用从 **getMilliseconds** 方法返回的值作为该参数的值。

要设置用[全球标准时间 \(UTC\)](#)表示的秒钟值，请使用 **setUTCSeconds** 方法。

如果一个参数的值超出了其取值范围或者是一个负数，其他保存的值将相应地得到修改。例如如果所保存的日期是 "Jan 5, 1996 00:00:00" 而且调用了函数 **setSeconds(150)**，日期将被改变为 "Jan 5, 1996 00:02:30."

示例

下面这个例子说明了 **setSeconds** 方法的用法：

```
function SetSecondsDemo(nsec, nmsec) {
    var d, s;                                     // 声明变量。
    var sep = ":";
    d = new Date();                               // 创建 Date
    对象。
    d.setSeconds(nsec, nmsec);                  // 设置秒和毫秒。
    s = "Current setting is ";
```

```

    s += d.toLocaleString() + sep +
d.getMilliseconds();

    return(s); // 返回新的设置。
}

```

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getSeconds 方法](#)(See 1.2.7.2.14) | [getUTCSeconds 方法](#)(See 1.2.7.2.24) | [setUTCSeconds 方法](#)(See 1.2.7.4.16)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.4.9 setTime 方法

设置 **Date** 对象的日期和时间值。

dateObj.**setTime**(*milliseconds*)

参数

dateObj

必选项。任意 **Date** 对象。

milliseconds

必选项。是一个整数值，它代表从格林威治标准时间（GMT）的 1970 年 1 月 1 日午夜开始所经过的毫秒数。

说明

如果 *milliseconds* 是一个负值，那它就表示 1970 年之前的日期。可用的日期范围大约是 1970 年 1 月 1 日午夜的前后各 285,616 年。

使用 **setTime** 方法来设置日期和时间与时区无关。

示例

下面这个例子说明了 **setTime** 方法的用法。

```

function SetTimeTest(newtime) {

    var d, s; // 声明变量。

```

```

    d = new Date(); // 创建 Date 对
象。

    d.setTime(newtime); // 设置时间。

    s = "Current setting is ";

    s += d.toUTCString();

    return(s); // 返回新的设置。
}

```

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getTime 方法](#)(See 1.2.7.2.15)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.4.10 setUTCDate 方法

设置 **Date** 对象中用[全球标准时间 \(UTC\)](#)表示的数值日期。

dateObj.**setUTCDate**(*numDate*)

参数

dateObj

必选项。任意 **Date** 对象。

numDate

必选项。是一个与数值日期相等的数值。

说明

要设置用[本地时间](#)表示的日期, 请使用 **setDate** 方法。

如果 *numDate* 的值超出 **Date** 对象中所保存的月份中的天数或者该值是负数。那么日期将被设置为由 *numDate* 减去所保存月份中天数而得到的日期。例如, 如果所保存的日期是 1996 年 1 月 5 日, 并且调用了方法 **setUTCDate(32)**, 那么日期将被改变为 1996 年 2 月 1 日。负数的处理方法与此相似。

示例

下面这个例子说明了 **setUTCDate** 方法的用法。

```
function SetUTCDateDemo(newdate) {  
    var d, s;                                // 声明变量。  
    d = new Date();                            // 创建 Date 对  
象。  
    d.setUTCDate(newdate);                    // 设置 UTC 日  
期。  
    s = "Current setting is ";  
    s += d.toUTCString();  
    return(s);                                // 返回新的设置。  
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getDate 方法](#)(See 1.2.7.2.6) | [getUTCDate 方法](#)(See 1.2.7.2.17) | [setDate 方法](#)(See 1.2.7.4.2)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.4.11 setUTCFullYear 方法

设置 **Date** 对象中用[全球标准时间 \(UTC\)](#)表示的年份值。

dateObj.**setUTCFullYear**(*numYear*[, *numMonth*[, *numDate*]])

参数

dateObj

必选项。任意 **Date** 对象。

numYear

必选项。一个等于年份的数值。

numMonth

可选项。一个等于月份的数值。如果提供了 *numDate*，那么也必须提供此项。

numDate

可选项。一个等于日期的数值。

说明

如果没有指定可选项，那么所有使用可选项的 **set** 方法都将使用从相应的 **get** 方法返回的数值作为可选项的数值。例如，如果 *numMonth* 参数是可选项的，但是没有被指定，JScript 将使用从 **getUTCMonth** 方法返回的值作为该参数的值。此外，如果参数的值超出其范围或者是负数，其他被保存的值都将相应地得到修改。

要设置用[本地时间](#)表示的年份，请使用 **setFullYear** 方法。

Date 对象中所支持的年份范围大约是 1970 年前后各 285,616 年。

示例

下面这个例子说明了 **setUTCFullYear** 方法的用法。

```
function SetUTCFullYearDemo(newyear) {
    var d, s;                                // 声明变量。
    d = new Date();                          // 创建 Date 对
    象。
    d.setUTCFullYear(newyear);             // 设置 UTC
    完整年份。
    s = "Current setting is ";
    s += d.toUTCString();
    return(s);                               // 返回新设置。
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getFullYear 方法](#)(See 1.2.7.2.8) | [getUTCFullYear 方法](#)(See 1.2.7.2.19) | [setFullYear 方法](#)(See 1.2.7.4.3)

应用于： [Date 对象](#)(See 1.2.8.4)

1.2.7.4.12 setUTCHours 方法

设置 **Date** 对象中用[全球标准时间 \(UTC\)](#)表示的小时值。

`dateObj.setUTCHours(numHours[, numMin[, numSec[, numMilli]])`

参数

dateObj

必选项。任意 **Date** 对象。

numHours

必选项。等于小时值的数值。

numMin

可选项。等于分钟值的数值，如果使用了 *numSec* 或 *numMilli*，那么此参数也必须提供。

numSec

可选项。等于秒钟值的数值。如果使用了 *numMilli* 参数，那么此参数也必须提供。

numMilli

可选项。等于毫秒值的数值。

说明

如果没有指定可选项，那么所有使用可选项的 **set** 方法都将使用从相应的 **get** 方法返回的数值作为可选项的数值。例如，如果 *numMin* 参数是可选的，但是没有被指定，JScript 将使用从 **getUTCMinutes** 方法返回的值作为该参数的值。

要设置用[本地时间](#)表示的小时值，请使用 **setHours** 方法。

如果参数的值超出其范围或者是负数，其他被保存的值都将相应地得到修改。例如，如果所保存的日期是 "Jan 5, 1996 00:00:00"，且调用了 **setHours(30)** 方法，那么日期将被改变为 "Jan 6, 1996 06:00:00."

示例

下面这个例子说明了 **setUTCHours** 方法的用法。

```
function    SetUTCHoursDemo(nhr,    nmin,
nsec) {
```

```

    var d, s;                                     // 声明变
量。
    d = new Date();                               // 创建
Date 对象。
    d.setUTCHours(nh, nmin, nsec); // 设置
UTC 小时, 分钟, 秒。
    s = "Current setting is " +
d.toUTCString()
    return(s);                                     // 返回新设
置。
}

```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getHours 方法](#)(See 1.2.7.2.9) | [getUTCHours 方法](#)(See 1.2.7.2.20) | [setHours 方法](#)(See 1.2.7.4.4)
 应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.4.13 setUTCMilliseconds 方法

设置 **Date** 对象中用[全球标准时间 \(UTC\)](#)表示的毫秒值。

dateObj.**setUTCMilliseconds**(*numMilli*)

参数

dateObj

必选项。任意 **Date** 对象。

numMilli

必选项。是一个等于毫秒值的数值。

说明

要设置用 [本地时间](#) 表示的毫秒值，请使用 **setMilliseconds** 方法。
如果 *numMilli* 的值大于 999 或者是负数，那么秒钟数（以及分钟、小时等等，如果需要的话）将增加一个适当的数量。

示例

下面这个例子说明了 **setUTCMilliseconds** 方法的用法：

```
function SetUTCMSecDemo(nmsec) {  
    var d, s;                                     // 声明  
    变量。  
    var sep = ":";                                // 初始  
    化分隔符。  
    d = new Date();                               // 创建  
    Date 对象。  
    d.setUTCMilliseconds(nmsec);                 //  
    设置 UTC 毫秒。  
    s = "Current setting is ";  
    s += d.toUTCString() + sep +  
    d.getUTCMilliseconds();  
    return(s);                                    // 返  
    回新设置。  
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#) (See 1.2.8.4) | [getMilliseconds 方法](#) (See 1.2.7.2.11) |
[getUTCMilliseconds 方法](#) (See 1.2.7.2.21) | [setMilliseconds 方法](#) (See 1.2.7.4.5)

应用于： [Date 对象](#) (See 1.2.8.4)

1.2.7.4.14 setUTCMinutes 方法

设置 **Date** 对象中用[全球标准时间 \(UTC\)](#)表示的分钟值。

```
dateObj.setUTCMinutes(numMinutes[, numSeconds[, numMilli]])
```

参数

dateObj

必选项。任意 **Date** 对象。

numMinutes

必选项。等于分钟值的数值。

numSeconds

可选项。等于秒钟值的数值。如果使用了 *numMilli* 参数，那么此参数也必须提供。

numMilli

可选项。等于毫秒值的数值。

说明

如果没有指定可选项，那么所有使用可选项的 **set** 方法都将使用从相应的 **get** 方法返回的数值作为可选项的数值。例如，如果 *numSeconds* 参数是可选的，但是没有被指定，JScript 将使用从 **getUTCSeconds** 方法返回的值作为该参数的值。

要将分钟值修改为用[本地时间](#)表示的形式，请使用 **setMinutes** 方法。

如果参数的值超出其范围或者是负数，其他被保存的值都将相应地得到修改。

例如，如果所保存的日期是 "Jan 5, 1996 00:00:00"，且调用了 **setUTCMinutes(70)** 方法，那么日期将被改变为 "Jan 5, 1996 01:10:00.00."

示例

下面这个例子说明了 **setUTCMinutes** 方法的用法。

```
function SetUTCMinutesDemo(nmin, nsec){
    var d, s;                                // 声明变量。
    d = new Date();                          // 创建 Date
    对象。
    d.setUTCMinutes(nmin,nsec);              // 设置 UTC
```

分钟。

```
s = "Current setting is " +
d.toUTCString()

return(s); // 返回新设置。
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getMinutes 方法](#)(See 1.2.7.2.12) | [getUTCMinutes 方法](#)(See 1.2.7.2.22) | [setMinutes 方法](#)(See 1.2.7.4.6)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.4.15 setUTCMonth 方法

设置 **Date** 对象中用 [全球标准时间 \(UTC\)](#)表示的月份值。

dateObj.**setUTCMonth**(*numMonth*[, *dateVal*])

参数

dateObj

必选项。任意 **Date** 对象。

numMonth

必选项。一个等于月份的数值。

dateVal

可选项。代表日期的数值。如果没有提供此参数，那么将使用通过调用 **getUTCDate** 方法而得到的数值。

说明

要设置用 [本地时间](#)表示的月份值，请使用 **setMonth** 方法。

如果 *numMonth* 的值大于 11(月份 0 代表一月)或者是负数，那么所保存的年份也将相应地得到增加或减少。例如，如果所保存的日期是 "Jan 5, 1996 00:00:00.00"，而且调用了 **setUTCMonth(14)** 方法，那么日期将被改变为 "Mar 5, 1997 00:00:00.00."

示例

下面这个例子说明了 **setUTCMonth** 方法的使用。

```
function SetUTCMonthDemo(newmonth) {
    var d, s;                                // 声明变量。
    d = new Date();                          // 创建
Date 对象。
    d.setUTCMonth(newmonth);                // 设置
UTC 月份。
    s = "Current setting is ";
    s += d.toUTCString();
    return(s);                                // 返回新设
置。
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象方法](#)(See 1.2.8.4) | [getMonth 方法](#)(See 1.2.7.2.13) | [getUTCMonth 方法](#)(See 1.2.7.2.23) | [setMonth 方法](#)(See 1.2.7.4.7)
应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.4.16 setUTCSeconds 方法

设置 **Date** 对象中用[全球标准时间 \(UTC\)](#)表示的秒值。

dateObj.**setUTCSeconds**(*numSeconds*[, *numMilli*])

参数

dateObj

必选项。任意 **Date** 对象。

numSeconds

必选项。等于秒值的数值。

numMilli

可选项。等于毫秒值的数值。

说明

如果没有指定可选项，那么所有使用可选项的 **set** 方法都将使用从相应的 **get** 方法返回的数值作为可选项的数值。例如，如果 *numMilli* 参数是可选的，但是没有被指定，JScript 将使用从 **getUTCMilliseconds** 方法返回的值作为该参数的值。

要设置用[本地时间](#)表示的秒钟值，请使用 **setSeconds** 方法。

如果参数的值超出其范围或者是负数，其他被保存的值都将相应地得到修改。

例如，如果所保存的日期是 "Jan 5, 1996 00:00:00"，且调用了 **setSeconds(150)** 方法，那么日期将被改变为 "Jan 5, 1996 00:02:30.00."

示例

下面这个例子说明了 **setSeconds** 方法的使用。

```
function SetUTCSecondsDemo(nsec,
nmsec) {
    var d, s;                                // 声明变量。
    d = new Date();                            // 创建
Date 对象。
    d.setUTCSeconds(nsec, nmsec);             // 设置
UTC 秒和毫秒。
    s = "Current UTC milliseconds setting
is ";
    s += d.getUTCMilliseconds();             // 获取
新设置。
    return(s);                                // 返回新设置。
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getSeconds 方法](#)(See 1.2.7.2.14) | [getUTCSeconds 方法](#)(See 1.2.7.2.24) | [setSeconds 方法](#)(See 1.2.7.4.8)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.4.17 setYear 方法

设置 **Date** 对象中的年份值。

dateObj.**setYear**(*numYear*)

参数

dateObj

必选项。任意 **Date** 对象。

numYear

必选项。其数值等于年份减去 1900。

说明

这个方法已经过时，之所以仍然保留，只是为了保持向后兼容性。请改用 **setFullYear** 方法。

要将 **Date** 对象的年份设置为 1997，请调用 **setYear(97)**。而要将年份设置为 2010，请调用 **setYear(2010)**。最后，要将年份设置为 0-99 范围内的一年，请使用 **setFullYear** 方法。

注意 对于 JScript 1.0 版，**setYear** 设置的值始终为 *numYear* 参数的值加上 1900。例如，要将年份设置为 1899，*numYear* 参数的值是 -1，而要将年份设置为 2000，*numYear* 参数的值是 100。

要求

[版本 1](#) (See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [getFullYear 方法](#)(See 1.2.7.2.8) | [getUTCFullYear 方法](#)(See 1.2.7.2.19) | [getYear 方法](#)(See 1.2.7.2.26) | [setFullYear 方法](#)(See 1.2.7.4.3) | [setUTCFullYear 方法](#)(See 1.2.7.4.11)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.4.18 shift 方法

移除数组中的第一个元素并返回该元素。

`arrayObj.shift( )`

必选的 `arrayObj` 引用是一个 **Array** 对象。

说明

shift 方法可移除数组中的第一个元素并返回该元素。

要求

[版本 5.5](#)(See 2.64)

请参阅

[unshift 方法](#)(See 1.2.7.5.21)

应用于: [Array 对象](#)(See 1.2.8.2)

1.2.7.4.19 sin 方法

返回数字的正弦值。

`Math.sin(number)`

`number` 参数是需要计算正弦的[数值表达式](#)。

说明

返回值为数字参数的正弦值。

要求

[版本 1](#)(See 2.64)

请参阅

[acos 方法](#)(See 1.2.7.1.2) | [asin 方法](#)(See 1.2.7.1.5) | [atan 方法](#)(See 1.2.7.1.6) | [cos 方法](#)(See 1.2.7.1.19) | [Math 对象方法](#)(See 1.2.8.9) | [tan 方法](#)(See 1.2.7.5.1)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.4.20 slice 方法 (Array)

返回一个数组的一段。

`arrayObj.slice(start, [end])`

参数

`arrayObj`

必选项。一个 **Array** 对象。

start

必选项。*arrayObj* 中所指定的部分的开始元素是从零开始计算的下标。

end

可选项。*arrayObj* 中所指定的部分的结束元素是从零开始计算的下标。

说明

slice 方法返回一个 **Array** 对象，其中包含了 *arrayObj* 的指定部分。

slice 方法一直复制到 *end* 所指定的元素，但是不包括该元素。如果 *start* 为负，将它作为 *length + start* 处理，此处 *length* 为数组的长度。如果 *end* 为负，就将它作为 *length + end* 处理，此处 *length* 为数组的长度。如果省略 *end*，那么 **slice** 方法将一直复制到 *arrayObj* 的结尾。如果 *end* 出现在 *start* 之前，不复制任何元素到新数组中。

示例

在下面这个例子中，除了最后一个元素之外，*myArray* 中所有的元素都被复制到 *newArray* 中：

```
newArray = myArray.slice(0, -1)
```

要求

[版本 3](#)(See 2.64)

请参阅

[slice 方法 \(String\)](#)(See 1.2.7.4.21) | [String 对象](#)(See 1.2.8.14)

应用于：[Array 对象](#)(See 1.2.8.2)

1.2.7.4.21 slice 方法 (String)

返回字符串的片段。

```
stringObj.slice(start, [end])
```

参数

stringObj

必选项。是一个 **String** 对象或文字。

start

必选项。下标以 0 开始的 *stringObj* 指定部分起始索引。

end

可选项。下标以 0 起始的 *stringObj* 的指定部分结束索引。

说明

slice 方法返回一个包含 *stringObj* 的指定部分的 **String** 对象。

slice 方法一直复制到 *end* 所指定的元素，但是不包括该元素。如果 *start* 为负，将它作为 *length + start* 处理，此处 *length* 为数组的长度。如果 *end* 为负，就将它作为 *length + end* 处理，此处 *length* 为数组的长度。如果省略 *end*，那么 **slice** 方法将一直复制到 *arrayObj* 的结尾。如果 *end* 出现在 *start* 之前，不复制任何元素到新数组中。

示例

在下面的示例中，**slice** 方法的两种用法将返回相同的结果。第二个示例中的 -1 指向 *str1* 中的最后一个字符，并作为提取操作的结束位置。

```
str1.slice(0)
```

```
str2.slice(0, -1)
```

要求

[版本 3](#)(See 2.64)

请参阅

[Array 对象](#)(See 1.2.8.2) | [slice 方法 \(Array\)](#)(See 1.2.7.4.20) | [String 对象的方法](#)(See 1.2.8.14)

应用于：[String 对象](#)(See 1.2.8.14)

1.2.7.4.22 small 方法

将 HTML 的<SMALL> 标识添加到 **String** 对象中的文本两端。

```
strVariable.small( )
```

```
"String Literal".small( )
```

说明

下面的示例演示了 **small** 方法是如何使用的：

```
var strVariable = "This is a string";
```

```
strVariable = strVariable.small( );
```

在上一条语句之后 *strVariable* 的值为：

```
<SMALL>This is a string</SMALL>
```

不检查该标识是否已应用于该字符串了。

要求

[版本 1](#)(See 2.64)

请参阅

[big 方法](#)(See 1.2.7.1.9) | [String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.4.23 sort 方法

返回一个元素已经进行了排序的 **Array** 对象。

```
arrayobj.sort(sortfunction)
```

参数

arrayObj

必选项。任意 **Array** 对象。

sortFunction

可选项。是用来确定元素顺序的函数的名称。如果这个参数被省略，那么元素将按照 ASCII 字符顺序进行升序排列。

说明

sort 方法将 **Array** 对象进行适当的排序；在执行过程中并不会创建新的 **Array** 对象。

如果为 *sortfunction* 参数提供了一个函数，那么该函数必须返回下列值之一：

- 负值，如果所传递的第一个参数比第二个参数小。
- 零，如果两个参数相等。
- 正值，如果第一个参数比第二个参数大。

示例

下面这个例子说明了 **sort** 方法的用法：

```
function SortDemo() {
    var a, l;                                // 声明变量。
```

```

    a = new Array("X" , "y" , "d",  "Z",
    "v", "m", "r");

    l = a.sort(); // 排序数组。

    return(l); // 返回排序的数
组。
}

```

要求

[版本 2](#)(See 2.64)

请参阅

[Array 对象的方法](#)(See 1.2.8.2)

应用于: [Array 对象](#)(See 1.2.8.2)

1.2.7.4.24 splice 方法

从一个数组中移除一个或多个元素，如果必要，在所移除元素的位置上插入新元素，返回所移除的元素。

```
arrayObj.splice(start, deleteCount, [item1[, item2[, . . . [,itemN]]]])
```

参数

arrayObj

必选项。一个 **Array** 对象。

start

必选项。指定从数组中移除元素的开始位置，这个位置是从 0 开始计算的。

deleteCount

必选项。要移除的元素的个数。

item1, item2, . . . ,itemN

必选项。要在所移除元素的位置上插入的新元素。

说明

splice 方法可以移除从 *start* 位置开始的指定个数的元素并插入新元素,从而修改 *arrayObj*。返回值是一个由所移除的元素组成的新 **Array** 对象。

要求

[版本 5.5](#)(See 2.64)

请参阅

[slice 方法 \(Array\)](#)(See 1.2.7.4.20)

应用于: [Array 对象](#)(See 1.2.8.2)

1.2.7.4.25 split 方法

将一个字符串分割为子字符串,然后将结果作为字符串数组返回。

stringObj.**split**([*separator*[, *limit*]])

参数

stringObj

必选项。要被分解的 **String** 对象或文字。该对象不会被 **split** 方法修改。

separator

可选项。字符串或 **正则表达式** 对象,它标识了分隔字符串时使用的是一个还是多个字符。如果忽略该选项,返回包含整个字符串的单一元素数组。

limit

可选项。该值用来限制返回数组中的元素个数。

说明

split 方法的结果是一个字符串数组,在 *stringObj* 中每个出现 *separator* 的位置都要进行分解。*separator* 不作为任何数组元素的部分返回。

示例

下面的示例演示了 **split** 方法的用法。

```
function SplitDemo() {  
    var s, ss;  
    var s = "The rain in Spain falls  
mainly in the plain.";
```

// 在每个空格字符处进行分解。

```
ss = s.split(" ");
return(ss);
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[concat 方法](#)(See 1.2.7.1.18) | [RegExp 对象](#)(See 1.2.8.12) | [正则表达式对象](#)(See 1.2.8.13) | [正则表达式的语法](#)(See 1.5.4) | [String 对象的方法](#)(See 1.2.8.14)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.4.26 sqrt 方法

返回数字的平方根。

Math.sqrt(*number*)

必选项 *number* 参数是[数值表达式](#)。

说明

如果 *number* 为负数，则返回值为零。

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象方法](#)(See 1.2.8.9) | [SQRT1_2 属性](#)(See 1.2.10.36) | [SQRT2 属性](#)(See 1.2.10.37)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.4.27 strike 方法

将 HTML 的<STRIKE> 标识放置到 **String** 对象中的文本两端。

strVariable.**strike**()

"String Literal".**strike**()

说明

下面的示例演示了 **strike** 方法是如何使用的。

```
var strVariable = "This is a string
object";
```

```
strVariable = strVariable.strike( );
```

在上一条语句之后 *strVariable* 的值为：

```
<STRIKE>This is a string object</STRIKE>
```

不检查该标识是否已应用于该字符串了。

要求

[版本 1](#)(See 2.64)

请参阅

[String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14)

应用于： [String 对象](#)(See 1.2.8.14)

1.2.7.4.28 sub 方法

将 HTML 的 <SUB> 标识放置到 **String** 对象中的文本两端。

```
strVariable.sub( )
```

```
"String Literal".sub( )
```

说明

下面的示例演示了 **sub** 方法是如何使用的：

```
var strVariable = "This is a string
object";
```

```
strVariable = strVariable.sub( );
```

在执行上一条语句后 *strVariable* 的值为：

```
<SUB>This is a string object</SUB>
```

不检查该标识是否已应用于该字符串了。

要求

[版本 1](#)(See 2.64)

请参阅

[String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14) | [sup 方法](#)(See 1.2.7.4.31)

应用于： [String 对象](#)(See 1.2.8.14)

1.2.7.4.29 substr 方法

返回一个从指定位置开始的指定长度的子字符串。

`stringvar.substr(start [, length ])`

参数

`stringvar`

必选项。要提取子字符串的字符串文字或 **String** 对象。

`start`

必选项。所需的子字符串的起始位置。字符串中的第一个字符的索引为 0。

`length`

可选项。在返回的子字符串中应包括的字符个数。

说明

如果 `length` 为 0 或负数，将返回一个空字符串。如果没有指定该参数，则子字符串将延续到 `stringvar` 的最后。

示例

下面的示例演示了 **substr** 方法的用法。

```
function SubstrDemo() {
    var s, ss;                                // 声明变量。
    var s = "The rain in Spain falls mainly
in the plain.";
    ss = s.substr(12, 5); // 获取子字符串。
    return(ss);          // 返回 "Spain"。
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14) | [substring 方法](#)(See 1.2.7.4.30)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.4.30 substring 方法

返回位于 **String** 对象中指定位置的子字符串。

```
strVariable.substring(start, end)
```

```
"String Literal".substring(start, end)
```

参数

start

指明子字符串的起始位置，该索引从 0 开始起算。

end

指明子字符串的结束位置，该索引从 0 开始起算。

说明

substring 方法将返回一个包含从 *start* 到最后（不包含 *end*）的子字符串的字符串。

substring 方法使用 *start* 和 *end* 两者中的较小值作为子字符串的起始点。例如，*strvar.substring*(0, 3) 和 *strvar.substring*(3, 0) 将返回相同的子字符串。

如果 *start* 或 *end* 为 **NaN** 或者负数，那么将其替换为 0。

子字符串的长度等于 *start* 和 *end* 之差的绝对值。例如，在 *strvar.substring*(0, 3) 和 *strvar.substring*(3, 0) 返回的子字符串的长度是 3。

示例

下面的示例演示了 **substring** 方法的用法。

```
function SubstringDemo() {  
    var ss;                                // 声明变量。  
    var s = "The rain in Spain falls mainly  
in the plain..";  
    ss = s.substring(12, 17);           // 取子字符串。  
    return(ss);                           // 返回子字符串
```


串。

}

要求

[版本 1](#)(See 2.64)

请参阅

[String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14) | [substr 方法](#)(See 1.2.7.4.29)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.4.31 sup 方法

将 HTML 的 <SUP> 标识放置到 **String** 对象中的文本两端。

```
strVariable.sup( )
```

```
"String Literal".sup( )
```

说明

下面的示例演示了 **sup** 方法是如何使用的:

```
var strVariable = "This is a string  
object";
```

```
strVariable = strVariable.sup( );
```

在执行上一条语句后 *strVariable* 的值为:

```
<SUP>This is a string object</SUP>
```

不检查该标识是否已应用于该字符串了。

要求

[版本 1](#)(See 2.64)

请参阅

[String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14) | [sub 方法](#)(See 1.2.7.4.28)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.5 T-Z

1.2.7.5.1 tan 方法

返回数字的正切值。

Math.tan(*number*)

必选项 *number* 参数是要计算正切值的[数值表达式](#)。

说明

返回值为 *number* 的正切值。

要求

[版本 1](#)(See 2.64)

请参阅

[acos 方法](#)(See 1.2.7.1.2) | [asin 方法](#)(See 1.2.7.1.5) | [atan 方法](#)(See 1.2.7.1.6) | [atan2 方法](#)(See 1.2.7.1.7) | [cos 方法](#)(See 1.2.7.1.19) | [Math 对象方法](#)(See 1.2.8.9) | [sin 方法](#)(See 1.2.7.4.19)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.7.5.2 test 方法

返回一个 Boolean 值，它指出在被查找的字符串中是否存在模式。

rgexp.**test**(*str*)

参数

rgexp

必选项。包含正则表达式模式或可用标志的[正则表达式](#)对象。

str

必选项。要在其上测试查找的字符串。

说明

test 方法检查在字符串中是否存在一个模式，如果存在则返回 **true**，否则就返回 **false**。

全局 **RegExp** 对象的属性不由 **test** 方法来修改。

示例

下面的例子举例说明了 **test** 方法的用法：

```
function TestDemo(re, s) {
    var s1;                                // 声明变量。

    // 检查字符串是否存在正则表达式。

    if (re.test(s))                        // 测试是否存
在。
        s1 = " contains ";                // s 包含模
式。
    else
        s1 = " does not contain ";       // s
不包含模式。

    return("'" + s + "'" + s1 + "'" +
re.source + "'"); // 返回字符串。
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[RegExp 对象](#)(See 1.2.8.12) | [正则表达式对象](#)(See 1.2.8.13) | [正则表达式对象的方法](#)(See 1.2.8.13) | [正则表达式对象的属性](#)(See 1.2.8.13) | [正则表达式的语法](#)(See 1.5.4)

应用于： [正则表达式对象](#)(See 1.2.8.13)

1.2.7.5.3 toArray 方法

返回一个由 VBArray 转换而来的标准 JScript 数组。

safeArray.toArray()

必选项 *safeArray* 参数是一个 **VBArray** 对象。

说明

该转换将多维的 VBArray 翻译成一个一维的 JScript 数组。每个后续维被添加到前一维的结尾。例如，一个三维的且每一维有三个元素的 VBArray 将被转换为如下所示的 JScript 数组：

假定该 VBArray 包含：(1, 2, 3), (4, 5, 6), (7, 8, 9)。在转换后，JScript 数组将包含：

1, 2, 3, 4, 5, 6, 7, 8, 9。

现在没有将 JScript 数组转换为 VBArray 的方法。

示例

下面的示例包括三个部分。第一部分是用来创建一个 Visual Basic 安全数组的 VBScript 代码。第二部分是 JScript 代码，将这个 VB 安全数组转换为一个 JScript 数组。这两部分都放在 HTML 页中的 <HEAD> 部分。第三部分是位于 <BODY> 部分的 JScript 代码，用来运行其他两个部分。

```
<HEAD>
```

```
<SCRIPT LANGUAGE="VBScript">
```

```
<!--
```

```
Function CreateVBArray()
```

```
    Dim i, j, k
```

```
    Dim a(2, 2)
```

```
    k = 1
```

```
    For i = 0 To 2
```

```
        For j = 0 To 2
```

```
            a(j, i) = k
```

```
            document.writeln(k)
```

```
            k = k + 1
```

```
        Next
```

```
        document.writeln("<BR>")
```

```
    Next
```

```
    CreateVBArray = a
```

```
End Function
```

```
-->
```

```
</SCRIPT>
```

```
<SCRIPT LANGUAGE="JScript">
```

```
<!--
```

```
function VBArrayTest(vbarray)
```

```
{
```

```
    var a = new VBArray(vbarray);
```

```
    var b = a.toArray();
```

```
    var i;
```

```
    for (i = 0; i < 9; i++)
```

```
    {
```

```
        document.writeln(b[i]);
```

```
    }
```

```
}
```

```
-->
```

```
</SCRIPT>
```

```
</HEAD>
```

```
<BODY>
```

```
<SCRIPT LANGUAGE="JScript">
```

```
<!--
```

```
    VBArrayTest(CreateVBArray());
```

```
-->
```

</SCRIPT>

</BODY>

要求

[版本 3](#)(See 2.64)

请参阅

[dimensions 方法](#)(See 1.2.7.1.22) | [getItem 方法](#)(See 1.2.7.2.10) | [lbound 方法](#)(See 1.2.7.3.3) | [ubound 方法](#)(See 1.2.7.5.19)

应用于: [VBAArray 对象](#)(See 1.2.8.15)

1.2.7.5.4 toDateString 方法

以字符串值的形式返回一个日期。

objDate.toDateString()

必选的 *objDate* 引用是一个 **Date** 对象。

说明

toDateString 方法返回一个包含日期的字符串, 此日期是当前时区中, 以一种方便而易于阅读的格式表示的值。

要求

[版本 5.5](#)(See 2.64)

请参阅

[toTimeString 方法](#)(See 1.2.7.5.16) | [toLocaleDateString 方法](#)(See 1.2.7.5.8)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.5.5 toExponential 方法

返回一个字符串, 其中包含一个以指数记数法表示的数字。

numObj.toExponential([fractionDigits])

参数

numObj

必选项。一个 **Number** 对象。

fractionDigits

可选项。小数点后数字位数。必须在 0 - 20 之间，包括 0 和 20。

说明

toExponential 方法返回一个字符串，该字符串代表一个以指数记数法表示的数字。该字符串中小数点之前有一位有效数字，并可能在小数点后有 *fractionDigits* 位数字。

如果没有 *fractionDigits* 参数，**toExponential** 方法将返回足够多位数字以便特别地指定该数字。

要求

[版本 5.5](#)(See 2.64)

请参阅

[toFixed 方法](#)(See 1.2.7.5.6) | [toPrecision 方法](#)(See 1.2.7.5.14)

应用于： [Number 对象](#)(See 1.2.8.10)

1.2.7.5.6 toFixed 方法

返回一个字符串，代表一个以定点表示法表示的数字。

numObj.**toFixed**([*fractionDigits*])

参数

numObj

必选项。一个 **Number** 对象。

fractionDigits

可选项。小数点后的数字位数。其值必须在 0 - 20 之间，包括 0 和 20。

说明

toFixed 方法返回一个以定点表示法表示的数字的字符串形式。该字符串中小数点之前有一位有效数字，而且其后必须包含 *fractionDigits* 数字。

如果没有 *fractionDigits* 参数，或者该参数为 **undefined**，**toFixed** 方法假定该值为 0。

要求

[版本 5.5](#)(See 2.64)

请参阅

[toExponential 方法](#)(See 1.2.7.5.5) | [toPrecision 方法](#)(See 1.2.7.5.14)

应用于: [Number 对象](#)(See 1.2.8.10)

1.2.7.5.7 toGMTString 方法

返回一个日期, 该日期用格林威治标准时间 (GMT) 表示并已被转换为字符串。

dateObj.toGMTString()

说明

toGMTString 方法已经过时, 之所以仍然提供这个方法, 只是为了提供向后的兼容性。推荐改用 **toUTCString** 方法。

toGMTString 方法返回一个 **String** 对象, 此对象中包含了按照 GMT 惯例进行格式化的日期。返回值的格式如下: "05 Jan 1996 00:00:00 GMT"。

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [toUTCString 方法](#)(See 1.2.7.5.18)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.5.8 toLocaleDateString 方法

以字符串的形式返回与宿主环境的当前区域设置相对应的日期。

objDate.toLocaleDateString()

必选的 *objDate* 引用是一个 **Date** 对象。

说明

toLocaleDateString 方法返回一个包含日期的字符串值, 该日期是当前时区中, 以一种方便而易于阅读的格式表示的值。该日期的格式为宿主环境的当前区域设置中的缺省格式。因为返回值可能随计算机的不同而不同, 所以脚本编写过程中不能依赖此方法的返回值。**toLocalDateString** 方法应该仅仅用于格式化显示 - 而绝不要作为计算的一部分。

要求

[版本 5.5](#)(See 2.64)

请参阅

[toDateString 方法](#)(See 1.2.7.5.4) | [toLocaleTimeString 方法](#)(See 1.2.7.5.11)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.5.9 toLocaleLowerCase 方法

返回一个字符串，其中所有的字母字符都被转换为小写，同时考虑到宿主环境的当前区域设置。

`stringVar.toLocaleLowerCase( )`

必选的 `stringVar` 引用是一个 **String** 对象，值或文字。

说明

toLocaleLowerCase 方法转换字符串中的字符，同时适应宿主环境的当前区域设置。在大多数情况下，其结果与利用 **toLowerCase** 方法所得到的结果是一样的。然而，如果语言规则与常规的 Unicode 大小写映射方式冲突，那么结果就会不同。

要求

[版本 5.5](#)(See 2.64)

请参阅

[toLocaleUpperCase 方法](#)(See 1.2.7.5.12) | [toLowerCase 方法](#)(See 1.2.7.5.13)

应用于：[String 对象](#)(See 1.2.8.14)

1.2.7.5.10 toLocaleString 方法

返回一个日期，该日期使用当前[区域设置](#)并已被转换为字符串。

`dateObj.toLocaleString()`

必选项 `dateObj` 为任意的 **Date** 对象。

说明

toLocaleString 方法返回一个 **String** 对象，这个对象中包含了用当前区域设置的默认格式表示的日期。

- 对于公元 1601 和 1999 之间的时间，日期格式要按照用户的“控制面板”中“区域设置”来确定。
- F 对于此区间外的其他时间，使用 **toString** 方法的默认格式。

例如，同样是 1 月 5 日，在美国，**toLocaleString** 可能会返回 "01/05/96 00:00:00"，而在欧洲，返回值则可能是 "05/01/96 00:00:00"，因为欧洲的惯例是将日期放在月份前面。

注意 **toLocaleString** 只用来显示结果给用户；不要在脚本中用来做基本计算，因为返回的结果是随机器不同而不同的。

示例

下面这个例子说明了 **toLocaleString** 方法的用法。

```

var d, s;                                // 声明变量。

d = new Date();                          // 创建
Date 对象。

s = "Current setting is ";

s += d.toLocaleString();                 // 转换为
当前区域。

return(s);                               // 返回转换的日
期。
}

```

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4)

应用于: [Array 对象](#)(See 1.2.8.2) | [Date 对象](#)(See 1.2.8.4) | [Number 对象](#)(See 1.2.8.10) | [Object 对象](#)(See 1.2.8.11)

1.2.7.5.11 toLocaleTimeString 方法

以字符串的形式返回与宿主环境的当前区域设置相对应的时间。

objDate.toLocaleTimeString()

必选的 *objDate* 引用是一个 **Date** 对象。

说明

toLocaleTimeString 方法返回一个包含时间的字符串值，该日期是当前时区中，以一种方便而易于阅读的格式表示的值。该时间的格式为宿主环境的当前区域设置中的缺省格式。因为返回值可能随计算机的不同而不同，所以脚本编写过程中不能依赖此方法的返回值。**toLocaleTimeString** 方法应该仅仅用于格式化显示 - 而绝不要作为计算的一部分。

要求

[版本 5.5](#)(See 2.64)

请参阅

[ToTimeString 方法](#)(See 1.2.7.5.16) | [toLocaleDateString 方法](#)(See 1.2.7.5.8)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.5.12 toLocaleUpperCase 方法

返回一个字符串，其中所有的字母字符都被转换为大写，同时适应宿主环境的当前区域设置。

stringVar.toLocaleUpperCase()

必选的 *stringVar* 引用是一个 **String** 对象，值或文字。

说明

toLocaleUpperCase 方法转换字符串中的字符，同时适应宿主环境的当前区域设置。在大多数情况下，其结果与利用 **toUpperCase** 方法所得到的结果是一样的。然而，如果语言规则与常规的 Unicode 大小写映射方式冲突，那么结果就会不同。

要求

[版本 5.5](#)(See 2.64)

请参阅

[toLocaleLowerCase 方法](#)(See 1.2.7.5.9) | [toUpperCase 方法](#)(See 1.2.7.5.17)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.5.13 toLowerCase 方法

返回一个字符串，该字符串中的字母被转换为小写字母。

strVariable.toLowerCase()

"String Literal".toLowerCase()

说明

toLowerCase 方法对非字母字符不会产生影响。

下面的示例演示了 of the **toLowerCase** 方法的效果:

```
var strVariable = "This is a STRING
object";

strVariable                                     =

strVariable.toLowerCase( );
```

在执行上一条语句后 *strVariable* 的值为:

```
this is a string object
```

要求

[版本 1](#)(See 2.64)

请参阅

[String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14) | [toUpperCase 方法](#)(See 1.2.7.5.17)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.7.5.14 toPrecision 方法

返回一个字符串，其中包含一个以指数记数法或定点记数法表示的，具有指定数字位数的数字。

`numObj.toPrecision ([precision])`

参数

`numObj`

必选项。一个 **Number** 对象。

`precision`

可选项。有效数字的位数。必须在 1 - 21 之间，包括 1 和 21。

说明

对于以指数记数法表示的数字，将返回小数点后的 *precision* - 1 位数字。对于以定点记数法表示的数字，将返回 *precision* 位有效数字。

如果没有提供 `precision` 或者为 **undefined**，那么将改为调用 **toString** 方法。

要求

[版本 5.5](#)(See 2.64)

请参阅

[toFixed 方法](#)(See 1.2.7.5.6) | [toExponential 方法](#)(See 1.2.7.5.5)

应用于: [Number 对象](#)(See 1.2.8.10)

1.2.7.5.15 toString 方法

返回对象的字符串表示。

`objectname.toString ([radix])`

参数

`objectname`

必选项。要得到字符串表示的对象。

radix

可选项。指定将数字值转换为字符串时的进制。

说明

toString 方法是所有内建的 JScript 对象的成员。它的操作依赖于对象的类型：

对象	操作
Array	将 Array 的元素转换为字符串。结果字符串由逗号分隔，且连接起来。
Boolean	如果 Boolean 值是 true ，则返回 “true”。否则，返回 “false”。
Date	返回日期的文字表示法。
Error	返回一个包含相关错误消息的字符串。
Function	返回如下格式的字符串，其中 <i>functionname</i> 是被调用 toString 方法函数的名称： function functionname () { [native code] }
Number	返回数字的文字表示。
String	返回 String 对象的值。
默认	返回 “[object objectname]”，其中 objectname 是对象类型的名称。

示例

下面的例子演示了使用带有 radix 参数的 toString 方法。上面所示函数的返回值是一个 Radix 转换表。

```
function CreateRadixTable () {
    var      s,      s1,      s2,      s3,      x;

    // 声明变量。

    s      =      "Hex      Dec      Bin  \n";

    // 创建表头。

    for      (x      =      0;      x      <      16;      x++)

    // 根据所示值的
    {
        //
        数字建立
```

```
switch(x)
```

```
// 表尺寸。
```

```
{
```

```
//
```

```
设置栏目间空间。
```

```
case 0 :
```

```
    s1 = "    ";
```

```
    s2 = "    ";
```

```
    s3 = "    ";
```

```
    break;
```

```
case 1 :
```

```
    s1 = "    ";
```

```
    s2 = "    ";
```

```
    s3 = "    ";
```

```
    break;
```

```
case 2 :
```

```
    s3 = "    ";
```

```
    break;
```

```
case 3 :
```

```
    s3 = "    ";
```

```
    break;
```

```
case 4 :
```

```
    s3 = "    ";
```

```
    break;
```

```
case 5 :
    s3 = " ";
    break;
case 6 :
    s3 = " ";
    break;
case 7 :
    s3 = " ";
    break;
case 8 :
    s3 = "" ;
    break;
case 9 :
    s3 = "";
    break;
default:
    s1 = "      ";
    s2 = "";
    s3 = "      ";
}
```

//

转换为十六进制、十进制、二进制。

```
s += " " + x.toString(16) + s1 +
x.toString(10)
```

```

        s += s2 + s3 + x.toString(2) + "\n";

    }

    return(s);

// 返回整个 radix 表。
}

```

要求

[版本 2](#)(See 2.64)

请参阅

[function 语句](#)(See 1.2.11.10)

应用于: [Array 对象](#)(See 1.2.8.2) | [Boolean 对象](#)(See 1.2.8.3) | [Date 对象](#)(See 1.2.8.4) | [Error 对象](#)(See 1.2.8.6) | [Function 对象](#)(See 1.2.8.7) | [Number 对象](#)(See 1.2.8.10) | [Object 对象](#)(See 1.2.8.11) | [String 对象](#)(See 1.2.8.14)

1.2.7.5.16 toTimeString 方法

以字符串形式返回时间。

objDate.**toTimeString**()

必选的 *objDate* 引用是一个 **Date** 对象。

说明

toTimeString 方法返回一个字符串值, 该字符串包含以一种方便而易于阅读的格式表示的, 当前时区的时间。

要求

[版本 5.5](#)(See 2.64)

请参阅

[toDateString 方法](#)(See 1.2.7.5.4) | [toLocaleTimeString 方法](#)(See 1.2.7.5.11)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.5.17 toUpperCase 方法

返回一个字符串, 该字符串中的所有字母都被转化为大写字母。

strVariable.**toUpperCase**()

"String Literal".**toUpperCase**()

说明

toUpperCase 方法对非字母字符不会产生影响。

示例

下面的示例演示了 **toUpperCase** 方法的效果：

```
var strVariable = "This is a STRING
object";

strVariable

strVariable.toUpperCase( );
```

在执行上一条语句后 *strVariable* 的值为：

THIS IS A STRING OBJECT

要求

[版本 1](#)(See 2.64)

请参阅

[String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14) | [toLowerCase 方法](#)(See 1.2.7.5.13)

应用于：[String 对象](#)(See 1.2.8.14)

1.2.7.5.18 toUTCString 方法

返回一个已被转换为字符串的，用[全球标准时间 \(UTC\)](#)表示的日期。

dateObj.**toUTCString**()

必选项 *dateObj* 参数为任意 **Date** 对象。

说明

toUTCString 方法返回一个 **String** 对象，此对象中包含了使用 UTC 惯例以一种方便易读的形式进行格式化的日期。

示例

下面这个例子说明了 **toUTCString** 方法的用法。

```
function toUTCStrDemo() {

    var d, s;                                // 声明变量。

    d = new Date();                          // 创建 Date 对
```

象。

```
s = "Current setting is ";
s += d.toUTCString();           // 转换为 UTC
```

字符串。

```
return(s);                       // 返回 UTC 字符
串。
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [toGMTString 方法](#)(See 1.2.7.5.7)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.5.19 ubound 方法

返回在 VBAArray 的指定维中所使用的最大索引值。

safeArray.ubound(dimension)

参数

safeArray

必选项。是一个 VBAArray 对象。

dimension

可选项。要获知其索引上界的 VBAArray 的维数。如果忽略，ubound 将该参数作为 1 进行处理。

说明

如果 VBAArray 为空，ubound 方法将返回 [undefined](#)。如果 dim 大于 VBAArray 的维数或为负数，该方法将产生一个“下标越界”的错误。

示例

下面的示例包括三个部分。第一部分是用来创建一个 Visual Basic 安全数组的 VBScript 代码。第二部分是 JScript 代码，确定该安全数组的维数和每一维的上界。这两部分都放在 HTML 页中的 <HEAD> 部分。第三部分是位于 <BODY> 部分的 JScript 代码，用来运行其他两个部分。

```
<HEAD>

<SCRIPT LANGUAGE="VBScript">

<!--

Function CreateVBAArray()
    Dim i, j, k
    Dim a(2, 2)
    k = 1
    For i = 0 To 2
        For j = 0 To 2
            a(j, i) = k
            k = k + 1
        Next
    Next
    CreateVBAArray = a
End Function

-->

</SCRIPT>

<SCRIPT LANGUAGE="JScript">

<!--

function VBAArrayTest(vba)
{
    var i, s;
```

```
var a = new VBAArray(vba);
for (i = 1; i <= a.dimensions(); i++)
{
    s = "The upper bound of dimension ";
    s += i + " is ";
    s += a.ubound(i) + ".<BR>";
    return(s);
}
}
-->
</SCRIPT>
</HEAD>

<BODY>
<SCRIPT language="jscript">
    document.write(VBAArrayTest(CreateVBAr
ray())));
</SCRIPT>
</BODY>
```

要求

[版本 3](#)(See 2.64)

请参阅

[dimensions 方法](#)(See 1.2.7.1.22) | [getItem 方法](#)(See 1.2.7.2.10) | [lbound 方法](#)(See 1.2.7.3.3) | [toArray 方法](#)(See 1.2.7.5.3)

应用于: [VBAArray 对象](#)(See 1.2.8.15)

1.2.7.5.20 unescape 方法

解码用 **escape** 方法进行了编码的 **String** 对象。

unescape(*charstring*)

必选项 *charstring* 参数是要解码的 **String** 对象。

说明

unescape 方法返回一个包含 *charstring* 内容的字符串值。所有以 %xx 十六进制形式编码的字符都用 [ASCII](#) 字符集中等价的字符代替。

以 %uxxxx 格式 (Unicode 字符) 编码的字符用十六进制编码 xxxx 的 Unicode 字符代替。

注意 **unescape** 方法不能用于解码统一资源标识码 (URI)。解该码可使用 **decodeURI** 和 **decodeURIComponent** 方法。

要求

[版本 1](#)(See 2.64)

请参阅

[DecodeURI 方法](#)(See 1.2.7.1.20) | [decodeURIComponent 方法](#)(See 1.2.7.1.21) | [escape 方法](#)(See 1.2.7.1.25) | [String 对象](#)(See 1.2.8.14)

应用于: [Global 对象](#)(See 1.2.8.8)

1.2.7.5.21 unshift 方法

将指定的元素插入数组开始位置并返回该数组。

arrayObj.**unshift**([*item1* [, *item2* [, . . . [, *itemN*]]]])

参数

arrayObj

必选项。一个 **Array** 对象。

item1, *item2*, . . . , *itemN*

可选项。将插入到该 **Array** 开始部分的元素。

说明

unshift 方法将这些元素插入到一个数组的开始部分，所以这些元素将以参数序列中的次序出现在数组中。

要求

[版本 5.5](#)(See 2.64)

请参阅

[shift 方法](#)(See 1.2.7.4.18)

应用于: [Array 对象](#)(See 1.2.8.2)

1.2.7.5.22 UTC 方法

返回[全球标准时间 \(UTC\)](#) (或 GMT) 的 1970 年 1 月 1 日到所指定日期之间所间隔的毫秒数。

Date.UTC(*year*, *month*, *day*[, *hours*[, *minutes*[, *seconds*[,*ms*]]]])

参数

year

必选项。为了获得跨世纪日期处理的精确性，需要使用完整的年份表示法。如果 *year* 处于 0 到 99 之间，那么 *year* 就被假定为 1900 + *year*。

month

必选项。月份是一个处于 0 到 11 之间的整数（一月到十二月）。

day

必选项。日期是一个处于 1 到 31 之间的整数。

hours

可选项。如果提供了 *minutes*，那么此项也必须提供。0 到 23 之间的一个整数（午夜到 11pm），用来指定小时。

minutes

可选的，如果提供了 *seconds*，那么此项也必须提供。0 到 59 之间的一个整数，用来指定分钟值。

seconds

可选项。如果提供了 *milliseconds*，那么此项也必须提供。从 0 到 59 之间的一个整数，用来指定秒钟值。

ms

可选项。从 0 到 999 之间的一个整数，用来指定毫秒数。

说明

UTC 方法返回从 UTC 的 1970 年 1 月 1 日午夜到所指定日期之间的毫秒数。这个返回值可以用在 **setTime** 方法中和 **Date** 对象的[构造函数](#)中。如果一个参数的值超出其范围或者是一个负数，那么其他保存过的值将相应地得到改变。例如，如果使用者指定 150 秒，JScript 会将该数值重新定义为 2 分钟 30 秒。

UTC 方法和 **Date** 对象的接受日期值的构造函数之间的差别在于：**UTC** 方法假设 UTC，而 **Date** 对象的构造函数假定[本地时间](#)。

UTC 方法是一个静态方法。因此，**Date** 对象不一定要在创建完之后才可以使用。

注意 如果 *year* 处于 0 到 99 之间，年份为 $1900 + year$ 。

示例

下面这个例子说明了 **UTC** 方法的用法：

```
function DaysBetweenDateAndNow(yr, mo,
dy) {
    var d, r, t1, t2, t3;                // 声明
变量。
    var MinMilli = 1000 * 60             // 初始化
变量。
    var HrMilli = MinMilli * 60
    var DyMilli = HrMilli * 24
    t1 = Date.UTC(yr, mo - 1, dy)      // 获
取从 1/1/1970 开始的毫秒数。
    d = new Date();                      // 创建
Date 对象。
    t2 = d.getTime();                    // 获取当前
时间。
    if (t2 >= t1)
```

```

        t3 = t2 - t1;

    else

        t3 = t1 - t2;

    r = Math.round(t3 / DyMilli);

    return(r); // 返回差。
}

```

要求

[版本 1](#)(See 2.64)

请参阅

[Date 对象的方法](#)(See 1.2.8.4) | [setTime 方法](#)(See 1.2.7.4.9)

应用于: [Date 对象](#)(See 1.2.8.4)

1.2.7.5.23 valueOf 方法

返回指定对象的原始值。

object.**valueOf**()

必选项 *object* 参数是任意固有 JScript 对象。

说明

每个 JScript 固有对象的 **valueOf** 方法定义不同。

对象	返回值
Array	数组的元素被转换为字符串，这些字符串由逗号分隔，连接在一起。其操作与 Array.toString 和 Array.join 方法相同。
Boolean	Boolean 值。
Date	存储的时间是从 1970 年 1 月 1 日午夜开始计的毫秒数 UTC 。
Function	函数本身。
Number	数字值。
Object	对象本身。这是默认情况。
String	字符串值。

Math 和 **Error** 对象没有 **valueOf** 方法。

要求

[版本 2](#)(See 2.64)

请参阅

[toString 方法](#)(See 1.2.7.5.15)

应用于：[Array 对象](#)(See 1.2.8.2) | [Boolean 对象](#)(See 1.2.8.3) | [Date 对象](#)(See 1.2.8.4) | [Function 对象](#)(See 1.2.8.7) | [Number 对象](#)(See 1.2.8.10) | [Object 对象](#)(See 1.2.8.11) | [String 对象](#)(See 1.2.8.14)

1.2.8 JScript 对象

描述	语言要素
启用并返回一个 Automation 对象的引用。	ActiveXObject 对象 (See 1.2.8.1)
提供对创建任何数据类型的数组的支持。	Array 对象 (See 1.2.8.2)
创建一个新的 Boolean 值。	Boolean 对象 (See 1.2.8.3)
提供日期和时间的基本存储和检索。	Date 对象 (See 1.2.8.4)
存储数据键、项对的对象。	Dictionary 对象 (See 1.4.2.1)
提供集合中的项的枚举。	Enumerator 对象 (See 1.2.8.5)
包含在运行 JScript 代码时发生的错误的有关信息。	Error 对象 (See 1.2.8.6)
提供对计算机文件系统的访问。	FileSystemObject 对象 (See 1.4.2.6)
创建一个新的函数。	Function 对象 (See 1.2.8.7)
是一个内部对象，目的是将全局方法集中在一个对象中。	Global 对象 (See 1.2.8.8)
一个内部对象，提供基本的数学函数和常数。	Math 对象 (See 1.2.8.9)
表示数值数据类型和提供数值常数的对象。	Number 对象 (See 1.2.8.10)
提供所有的 JScript 对象的公共功能。	Object 对象 (See 1.2.8.11)
存储有关正则表达式模式查找的信息。	RegExp 对象 (See 1.2.8.12)
包含一个正则表达式模式。	正则表达式对象 (See 1.2.8.13)
提供对文本字符串的操作和格式处理，判定在字符串中是否存在某个子字符串及确定其位置。	String 对象 (See 1.2.8.14)
提供对 Visual Basic 安全数组的访问。	VBAArray 对象 (See 1.2.8.15)

1.2.8.1 ActiveXObject 对象

启用并返回 [Automation 对象](#)的引用。

```
newObj = new ActiveXObject(servername.typename[, location])
```

ActiveXObject 对象语法有这些部分：

参数

newObj

必选项。要赋值为 **ActiveXObject** 的变量名。

servername

必选项。提供该对象的应用程序的名称。

typename

必选项。要创建的对象类型或类。

location

可选项。创建该对象的网络服务器的名称。

说明

Automation 服务器至少提供一类对象。例如，字处理应用程序可能提供应用程序对象、文档对象和工具栏对象。

要创建 Automation 对象，将新的 **ActiveXObject** 赋给对象变量：

```
var ExcelSheet;
```

```
ExcelApp                                =                new  
ActiveXObject("Excel.Application");
```

```
ExcelSheet                              =                new  
ActiveXObject("Excel.Sheet");
```

本代码启动创建对象的应用程序（在这种情况下，Microsoft Excel 工作表）。一旦对象被创建，就可以用定义的对象变量在代码中引用它。 在下面的例子中，通过对象变量 `ExcelSheet` 访问新对象的属性和方法和其他 Excel 对象，包括 `Application` 对象和 `ActiveSheet.Cells` 集合。

```
// 使 Excel 通过 Application 对象可见。
```

```
ExcelSheet.Application.Visible = true;
```

```
// 将一些文本放置到表格的第一格中。
```

```
ExcelSheet.ActiveSheet.Cells(1,1).Value  
= "This is column A, row 1";
```

```
// 保存表格。
```

```
ExcelSheet.SaveAs("C:\\TEST.XLS");
```

```
// 用 Application 对象用 Quit 方法关闭 Excel。
```

```
ExcelSheet.Application.Quit();
```

只有当 Internet 安全性关闭时才能完成在远程服务器中创建对象。要在远程网络计算机创建对象，可以将该计算机的名称传递给 **ActiveXObject** 的 *servername* 参数。该名称与共享名的机器名部分相同。比如共享名为 "\\myserver\\public" 的网络，*servername* 是 "myserver"。另外，可以用 DNS 格式或 IP 地址指定 *servername*。

下面的代码返回在名为 "myserver" 的远程网络计算机上运行的 Excel 实例的版本号：

```
function GetAppVersion() {
    var          XLApp          =          new
ActiveXObject("Excel.Application",
    "MyServer");
    return(XLApp.Version);
}
```

如果指定的远程服务器不存在或找不到时将发生错误。

要求

[版本 1](#)(See 2.64)

请参阅

[GetObject 函数](#)(See 1.2.6.1)

1.2.8.2 Array 对象

提供对创建任何数据类型的数组的支持。

```
arrayObj = new Array()
```

```
arrayObj = new Array([size])
```

```
arrayObj = new Array([element0[, element1[, ...[, elementN]]]])
```

参数

arrayObj

必选项。要赋值为 **Array** 对象的变量名。

size

可选项。可选项数组的大小。由于数组的下标是从零开始，创建的元素的下标将从零到 *size* - 1。

element0, ..., elementN

可选项。要放到数组中的元素。这将创建具有 $n + 1$ 个元素的长度为 $n + 1$ 的数组。使用该语法时必须有一个以上元素。

说明

创建数组后，能够用 [] 符号访问数组单个元素，例如：

```
var my_array = new Array();  
for (i = 0; i < 10; i++)  
{  
    my_array[i] = i;  
}  
  
x = my_array[4];
```

由于 Microsoft JScript 中的数组的下标是从零开始的，前面例子中最后一条语句访问数组的第五个元素。该元素中保存的值是 4。

如果只向 **Array** 的构造函数传递了一个参数，而该参数是数字，则它必须是无符号 32 位整数（大约 40 亿）。该值成为数组的大小。如果该值为数值，但小于 0 或不为整数，发生运行时错误。

如果传递给 **Array** 构造函数的是单个值并且不是数值，设置 **length** 属性为 1，而且唯一的元素值成为单个的传入的参数。

请注意 JScript 数组为解析数组，也就是尽管可以分配多个元素给一个数组，但实际上只有包含数据的元素才存在。这减少了数组使用的内存数量。

属性

[constructor 属性](#) (See 1.2.10.4) | [length 属性](#) (See 1.2.10.15) | [prototype 属性](#) (See 1.2.10.34)

方法

[concat 方法](#) (See 1.2.7.1.17) | [join 方法](#) (See 1.2.7.3.1) | [pop 方法](#) (See 1.2.7.3.15) | [push 方法](#) (See 1.2.7.3.17) | [reverse 方法](#) (See 1.2.7.3.20) | [shift 方法](#) (See 1.2.7.4.18) | [slice 方法](#) (See 1.2.7.4.20) | [sort 方法](#) (See 1.2.7.4.23) | [splice 方法](#) (See 1.2.7.4.24) | [toLocaleString 方法](#) (See 1.2.7.5.10) | [toString 方法](#) (See 1.2.7.5.15) | [unshift 方法](#) (See 1.2.7.5.15)

1.2.7.5.21) | [valueOf 方法](#)(See 1.2.7.5.23)

要求

[版本 2](#)(See 2.64)

请参阅

[new 运算符](#)(See 1.2.9.32)

1.2.8.3 Boolean 对象

创建新的 Boolean 值。

语法

```
boolObj = new Boolean([boolValue])
```

参数

boolObj

必选项。要赋值为 Boolean 对象的变量名。

boolValue

可选项。是新对象的初始 Boolean 值。如果忽略 *Boolvalue*，或者其值为 **false**、0、null、NaN，或者空字符串，则该 Boolean 对象的初始值为 **false**。否则，初始值为 **true**。

说明

Boolean 对象是 Boolean 数据类型的[包装器](#)。每当 Boolean 数据类型转换为 **Boolean** 对象时，JScript 都隐含地使用 **Boolean** 对象。很少会显式地调用 **Boolean** 对象。

属性

[constructor 属性](#)(See 1.2.10.4) | [prototype 属性](#)(See 1.2.10.34)

方法

[toString 方法](#)(See 1.2.7.5.15) | [valueOf 方法](#)(See 1.2.7.5.23)

要求

[版本 2](#)(See 2.64)

请参阅

[new 运算符](#)(See 1.2.9.32) | [var 语句](#)(See 1.2.11.18)

1.2.8.4 Date 对象

启用基本存储器并取得日期和时间。

```
dateObj = new Date()
```

```
dateObj = new Date(dateVal)
```

```
dateObj = new Date(year, month, date[, hours[, minutes[, seconds[,ms]]]])
```

参数

dateObj

必选项。要赋值为 **Date** 对象的变量名。

dateVal

必选项。如果是数字值，*dateVal* 表示指定日期与 1970 年 1 月 1 日午夜间[全球标准时间](#)的毫秒数。如果是字符串，则 *dateVal* 按照 **parse** 方法中的规则进行解析。*dateVal* 参数也可以是从某些 ActiveX(R) 对象返回的 VT_DATE 值。

year

必选项。完整的年份，比如，1976（而不是 76）。

month

必选项。表示的月份，是从 0 到 11 之间的整数（1 月至 12 月）。

date

必选项。表示日期，是从 1 到 31 之间的整数。

hours

可选项。如果提供了 *minutes* 则必须给出。表示小时，是从 0 到 23 的整数（午夜到 11pm）。

minutes

可选项。如果提供了 *seconds* 则必须给出。表示分钟，是从 0 到 59 的整数。

seconds

可选项。如果提供了 *milliseconds* 则必须给出。表示秒钟，是从 0 到 59 的整数。

ms

可选项。表示毫秒，是从 0 到 999 的整数。

说明

Date 对象保存以毫秒为单位表示特定时间段。如果某个参数的值大于其范围或为负数，则存储的其他值将做相应的调整。例如，如果指定 150 秒，JScript 将该数字重新定义为 2 分 30 秒。

如果数字为 **NaN**，则表示该对象不代表特定的时间段。如果未向 **Date** 对象传递参数，它将被初始化为当前时间 (UTC)。在能够使用该对象前必须为其赋值。

Date 对象能够表示的日期范围约等于 1970 年 1 月 1 日前后各 285,616 年。

Date 对象具有两个不创建 **Date** 对象就可以调用的静态方法。它们是 **parse** 和 **UTC**。

错误

下面的示例演示了 **Date** 对象的用法。

```
function DateDemo() {
    var d, s = "Today's date is: ";

    // 声明变量。

    d = new Date();

    // 创建 Date 对象。

    s += (d.getMonth() + 1) + "/";

    // 获取月份。

    s += d.getDate() + "/";

    // 获取日。

    s += d.getFullYear();

    // 获取年份。

    return(s);
}
```

// 返回日期。

}

属性

[constructor 属性](#)(See 1.2.10.4) | [prototype 属性](#)(See 1.2.10.34)

方法

[getDate 方法](#)(See 1.2.7.2.6) | [getDay 方法](#)(See 1.2.7.2.7) | [getFullYear 方法](#)(See 1.2.7.2.8) | [getHours 方法](#)(See 1.2.7.2.9) | [getMilliseconds 方法](#)(See 1.2.7.2.11) | [getMinutes 方法](#)(See 1.2.7.2.12) | [getMonth 方法](#)(See 1.2.7.2.13) | [getSeconds 方法](#)(See 1.2.7.2.14) | [getTime 方法](#)(See 1.2.7.2.15) | [getTimezoneOffset 方法](#)(See 1.2.7.2.16) | [getUTCDate 方法](#)(See 1.2.7.2.17) | [getUTCDay 方法](#)(See 1.2.7.2.18) | [getUTCFullYear 方法](#)(See 1.2.7.2.19) | [getUTCHours 方法](#)(See 1.2.7.2.20) | [getUTCMilliSeconds 方法](#)(See 1.2.7.2.21) | [getUTCMinutes 方法](#)(See 1.2.7.2.22) | [getUTCMonth 方法](#)(See 1.2.7.2.23) | [getUTCSeconds 方法](#)(See 1.2.7.2.24) | [getVarDate 方法](#)(See 1.2.7.2.25) | [getYear 方法](#)(See 1.2.7.2.26) | [setDate 方法](#)(See 1.2.7.4.2) | [setFullYear 方法](#)(See 1.2.7.4.3) | [setHours 方法](#)(See 1.2.7.4.4) | [setMilliseconds 方法](#)(See 1.2.7.4.5) | [setMinutes 方法](#)(See 1.2.7.4.6) | [setMonth 方法](#)(See 1.2.7.4.7) | [setSeconds 方法](#)(See 1.2.7.4.8) | [setTime 方法](#)(See 1.2.7.4.9) | [setUTCDate 方法](#)(See 1.2.7.4.10) | [setUTCFullYear 方法](#)(See 1.2.7.4.11) | [setUTCHours 方法](#)(See 1.2.7.4.12) | [setUTCMilliSeconds 方法](#)(See 1.2.7.4.13) | [setUTCMinutes 方法](#)(See 1.2.7.4.14) | [setUTCMonth 方法](#)(See 1.2.7.4.15) | [setUTCSeconds 方法](#)(See 1.2.7.4.16) | [setYear 方法](#)(See 1.2.7.4.17) | [toGMTString 方法](#)(See 1.2.7.5.7) | [toLocaleString 方法](#)(See 1.2.7.5.10) | [toUTCString 方法](#)(See 1.2.7.5.18) | [toString 方法](#)(See 1.2.7.5.15) | [valueOf 方法](#)(See 1.2.7.5.23) | [parse 方法](#)(See 1.2.7.3.12) | [UTC 方法](#)(See 1.2.7.5.22)

要求

[版本 1](#)(See 2.64)

请参阅

[new 运算符](#)(See 1.2.9.32) | [var 语句](#)(See 1.2.11.18)

1.2.8.5 Enumerator 对象

启用枚举集中的项目。

```
enumObj = new Enumerator([collection])
```

参数

enumObj

必选项。要赋值为 **Enumerator** 对象的变量名。

collection

可选项。任意 **Collection** 对象。

说明

集合与数组的不同点在于集合的成员不能直接访问。不象处理数组时使用下标，这时只能将当前项目指针移动到集合的下一或前一元素。

Enumerator 对象提供了访问集合的任何成员的方法，其操作与 VBScript 中的 **For...Each** 语句相似。

示例

下面的代码显示了 **Enumerator** 对象的用法：

```
function ShowDriveList() {
    var    fso,    s,    n,    e,    x;

    // 声明变量。

    fso    =    new
ActiveXObject("Scripting.FileSystemObject");

    e    =    new    Enumerator(fso.Drives);
    // 在驱动器上创建 Enumerator。

    s = "";

    for    (;!e.atEnd();e.moveNext())

    // 枚举驱动器集合。
    {
        x = e.item();

        s = s + x.DriveLetter;

        s += " - ";

        if (x.DriveType == 3)                //
查看是否网络驱动。
```

```

        n = x.ShareName;                                //
获取共享名。
        else if (x.IsReady)                             //
查看驱动器是否已就绪。
        n = x.VolumeName;                                //
获取卷名。
    else
        n = "[ 驱动器未就绪 ] ";
        s += n + "<br>";
    }

    return(s);

// 返回活动驱动器列表。
}

```

属性

Enumerator 对象没有属性。

方法

[atEnd 方法](#)(See 1.2.7.1.8) | [item 方法](#)(See 1.2.7.2.31) | [moveFirst 方法](#)(See 1.2.7.3.10) | [moveNext 方法](#)(See 1.2.7.3.11)

要求

[版本 3](#)(See 2.64)

请参阅

[Drives 集合](#)(See 1.4.2.3) | [Files 集合](#)(See 1.4.2.5) | [Folders 集合](#)(See 1.4.2.8)

1.2.8.6 Error 对象

保存有关错误的信息。

```

var newErrorObj = new Error(

)

var newErrorObj = new Error(
    number

```

```
)
var newErrorObj = new Error(
    number,
    description
)
```

Error 对象的[构造函数](#)语法有以下部分：

参数

number

与错误相联的数字值。如果省略则为零。

description

描述错误的简短字符串。如果省略则为空字符串。

说明

每当产生[运行时错误](#)，就产生 **Error** 对象的一个实例以描述错误。该实例有两个固有属性保存错误的描述（**description** 属性）和错误号（**number** 属性）。

错误号是 32 位的值。高 16 位字是设备代码，而低字是实际的错误代码。

Error 对象也可以用如上所示的语法显式创建，或用 **throw** 语句抛掉。在两种情况下，都可以添加选择的任何属性，以拓展 **Error** 对象的能力。

典型地，在 **try...catch** 语句中创建的局部[变量](#)引用隐式创建的 **Error** 对象。因此，可以按选择的任何方法使用错误号和描述。

下面的例子演示了隐式创建 **Error** 对象的使用：

```
try
{
    x = y                                // 产生错误。
} catch (e) {                             // 创建局部变量
    e.
        response.write(e)                // 打印
        "[object Error]".
        response.write(e.number & 0xFFFF) //
打印 5009。
        response.write(e.description)    //
```

打印 "'y' is undefined".

```
}
```

方法

Error 对象没有方法。

属性

[description 属性](#)(See 1.2.10.5) | [number 属性](#)(See 1.2.10.30)

要求

[版本 5](#)(See 2.64)

请参阅

[new 运算符](#)(See 1.2.9.32) | [throw 语句](#)(See 1.2.11.16) | [try...catch 语句](#)(See 1.2.11.17) | [var 语句](#)(See 1.2.11.18)

1.2.8.7 Function 对象

创建新的函数。

语法 1

```
function functionName( [argname1 [, ...[, argnameN]] ] )  
{  
    body  
}
```

语法 2

```
functionName = new Function( [argname1, [... argnameN,]] body );
```

参数

functionName

必选项。最新创建函数的名称

argname1...*argnameN*

可选项。函数接收的参数列表。

body

可选项。包含调用该函数时执行的 JScript 代码块的字符串。

说明

函数 JScript 中的基本数据类型。语法 1 创建在必要时由 JScript 转换为 **Function** 对象的函数值。JScript 在调用函数时将用语法 2 创建的函数转换为 **Function** 对象。

语法 1 是 JScript 中创建函数的基本方法。语法 2 是用于显式创建函数对象的另一方法。

例如，要创建将传递给它的两个参数相加的函数，可以用两种方法中的任一种完成：

例子 1

```
function add(x, y)
{
    return (x + y);           // 执行加法并返回结果。
}
```

例子 2

```
var add = new Function("x", "y",
    "return (x+y)");
```

在两种情况下，都可以用如下代码行调用该函数：

```
add(2, 3);
```

注意 在调用函数时，请确保包含了括号和必需的参数。调用函数时不用括号导致返回函数的文本而不是函数执行的结果。

属性

[arguments 属性](#)(See 1.2.10.2) | [caller 属性](#)(See 1.2.10.3) | [constructor 属性](#)(See 1.2.10.4) | [prototype 属性](#)(See 1.2.10.34)

方法

[toString 方法](#)(See 1.2.7.5.15) | [valueOf 方法](#)(See 1.2.7.5.23)

要求

[版本 2](#)(See 2.64)

请参阅

[function 语句](#)(See 1.2.11.10) | [new 运算符](#)(See 1.2.9.32) | [var 语句](#)(See 1.2.11.18)

1.2.8.8 Global 对象

是一个[固有对象](#)，目的是把所有全局方法集中在一个对象中。

Global 对象没有语法。直接调用其方法。

说明

Global 对象从不直接使用，并且不能用 **new** 运算符创建。它在 Scripting 引擎被初始化时创建，并立即使其方法和属性可用。

属性

[Infinity 属性](#)(See 1.2.10.11) | [NaN 属性](#)(See 1.2.10.28)

方法

[escape 方法](#)(See 1.2.7.1.25) | [eval 方法](#)(See 1.2.7.1.26) | [isFinite 方法](#)(See 1.2.7.2.28) | [isNaN 方法](#)(See 1.2.7.2.29) | [parseFloat 方法](#)(See 1.2.7.3.13) | [parseInt 方法](#)(See 1.2.7.3.14) | [unescape 方法](#)(See 1.2.7.5.20)

要求

[版本 5](#)(See 2.64)

请参阅

[Object 对象](#)(See 1.2.8.11)

1.2.8.9 Math 对象

是一个固有对象，提供基本数学函数和常数。

Math.[*{property | method}*]

参数

property

必选项。**Math** 对象的一个属性名。

method

必选项。**Math.** 对象的一个方法名。

说明

Math 对象不能用 **new** 运算符创建，如果试图这样做则给出错误。该对象在装载脚本引擎时由该引擎创建。其所有方法和属性在脚本中总是可用。

属性

[E 属性](#)(See 1.2.10.6) | [LN2 属性](#)(See 1.2.10.19) | [LN10 属性](#)(See 1.2.10.18) | [LOG2E 属性](#)(See 1.2.10.21) | [LOG10E 属性](#)(See 1.2.10.20) | [PI 属性](#)(See 1.2.10.31) | [SQRT1 2 属性](#)(See 1.2.10.36) | [SQRT2 属性](#)(See 1.2.10.37)

方法

[abs 方法](#)(See 1.2.7.1.1) | [acos 方法](#)(See 1.2.7.1.2) | [asin 方法](#)(See 1.2.7.1.5) | [atan 方法](#)(See 1.2.7.1.6) | [atan2 方法](#)(See 1.2.7.1.7) | [ceil 方法](#)(See 1.2.7.1.13) | [cos 方法](#)(See 1.2.7.1.19) | [exp 方法](#)(See 1.2.7.1.28) | [floor 方法](#)(See 1.2.7.2.2) | [log 方法](#)(See 1.2.7.3.6) | [max 方法](#)(See 1.2.7.3.8) | [min 方法](#)(See 1.2.7.3.9) | [pow 方法](#)(See 1.2.7.3.16) | [random 方法](#)(See 1.2.7.3.18) | [round 方法](#)(See 1.2.7.3.21) | [sin 方法](#)(See 1.2.7.4.19) | [sqrt 方法](#)(See 1.2.7.4.26) | [tan 方法](#)(See 1.2.7.5.1)

要求

[版本 1](#)(See 2.64)

请参阅

[Number 对象](#)(See 1.2.8.10)

1.2.8.10 Number 对象

代表数值数据类型和提供数值常数的对象。

```
numObj = new Number (value)
```

参数

numObj

必选项。要赋值为 **Number** 对象的变量名。

value

必选项。Required. The numeric value of the **Number** object being created.

value 参数是对象的数字值。

说明

JScript 根据数字值的要求创建 **Number** 对象。 很少有必要显式创建 **Number** 对象。

Number 对象最主要的用途是将其属性集中到一个对象中，以及使数字能够通过 **toString** 方法转换为字符串。

属性

[MAX_VALUE 属性](#)(See 1.2.10.22) | [MIN_VALUE 属性](#)(See 1.2.10.24) | [NaN 属性](#)

(See 1.2.10.27) | [NEGATIVE INFINITY 属性](#)(See 1.2.10.29) | [POSITIVE INFINITY 属性](#)(See 1.2.10.32) | [constructor 属性](#)(See 1.2.10.4) | [prototype 属性](#)(See 1.2.10.34)

方法

[toLocaleString 方法](#)(See 1.2.7.5.10) | [toString 方法](#)(See 1.2.7.5.15) | [valueOf 方法](#)(See 1.2.7.5.23)

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象](#)(See 1.2.8.9) | [new 运算符](#)(See 1.2.9.32)

1.2.8.11 Object 对象

提供所有 JScript 对象通用的功能。

```
obj = new Object([value])
```

参数

obj

必选项。要赋值为 **Object** 对象的变量名。

value

可选项。任意一种 JScript [基本](#)数据类型。（Number、Boolean、或 String。）如果 *value* 为一个对象，返回不作改动的该对象。如果 *value* 为 **null**、**undefined**，或者没有给出，则产生没有内容的对象。

说明

Object 对象被包含在所有其它 JScript 对象中；在所有其它对象中它的方法和属性都是可用的。在用户定义的对象中可以重定义这些方法，并在适当的时候通过 JScript 调用。**toString** 方法是经常被重定义的 **Object** 方法的例子。

在本语言参考中，每个 **Object** 方法的说明包括默认的和与对象相关[固有 JScript 对象](#)的实现信息。

属性

[prototype 属性](#)(See 1.2.10.34) | [constructor 属性](#)(See 1.2.10.4)

方法

[toLocaleString 方法](#)(See 1.2.7.5.10) | [toString 方法](#)(See 1.2.7.5.15) | [valueOf 方法](#)(See 1.2.7.5.23)

要求

[版本 3](#)(See 2.64)

请参阅

[Function 对象](#)(See 1.2.8.7) | [Global 对象](#)(See 1.2.8.8)

1.2.8.12 RegExp 对象

保存有关正则表达式模式匹配信息的固有全局对象。

RegExp.*property*

必选项 *property* 参数是 **RegExp** 对象的任意一个属性。

说明

RegExp 对象不能直接创建，但始终可以使用。在成功的正则表达式查找完成之前，**RegExp** 对象各属性的初始值如下所示：

属性	简写	初始值
index		-1
lastIndex		-1
lastMatch	\$&	空字符串。
lastParen	\$+	空字符串。
leftContext		空字符串。
rightContext		空字符串。
\$1 - \$9	\$1 - \$9	空字符串。

在成功的正则表达式查找完成之前，其属性值为 [undefined](#)。

不要将全局 **RegExp** 对象与 **正则表达式** 对象混淆。尽管听起来像是一回事，但它们是截然不同的。全局 **RegExp** 对象的属性包含不断更新的关于每个匹配出现的信息，而**正则表达式**对象只包含出现**正则表达式**匹配的信息。

示例

下面的例子演示了全局 **RegExp** 对象的用法。

```
function matchDemo() {  
    var s;  
  
    var re = new RegExp("d(b+)(d)","ig");  
  
    var str = "cdbBdbbsbdbdz";  
  
    var arr = re.exec(str);
```

```

    s = "$1 contains: " + RegExp.$1 +
"\n";

    s += "$2 contains: " + RegExp.$2 +
"\n";

    s += "$3 contains: " + RegExp.$3;

    return(s);
}

```

属性

[\\$1...\\$9 属性](#)(See 1.2.10.1) | [index 属性](#)(See 1.2.10.10) | [input 属性](#)(See 1.2.10.12) | [lastIndex 属性](#)(See 1.2.10.14) | [lastMatch 属性](#)(See 2.67) | [lastParen 属性](#)(See 2.68) | [leftContext 属性](#)(See 2.69) | [rightContext 属性](#)(See 2.71)

方法

RegExp 对象没有方法。

要求

[版本 3](#)(See 2.64)

请参阅

[正则表达式对象](#)(See 1.2.8.13) | [正则表达式语法](#)(See 1.5.4) | [String 对象](#)(See 1.2.8.14)

1.2.8.13 正则表达式对象

本对象包含正则表达式模式以及表明如何应用模式的标志。

语法 1

```
re = /pattern/[flags]
```

语法 2

```
re = new RegExp("pattern",["flags"])
```

参数

re

必选项。将要赋值为正则表达式模式的变量名。

Pattern

必选项。要使用的正则表达式模式。如果使用语法 1，用 “/” 字符分隔模式。如果用语法 2，用引号将模式引起来。

flags

可选项。如果使用语法 2 要用引号将 flag 引起来。标志可以组合使用，可用的有：

- **g** （全文查找出现的所有 *pattern*）
- **i** （忽略大小写）
- **m** （多行查找）

说明

不要将**正则表达式**对象跟全局 **RegExp** 对象混淆。尽管听起来像是一回事，但它们是截然不同的。**正则表达式**对象的属性只包含一个**正则表达式**的信息，而全局 **RegExp** 对象的属性包含了不断更新的每一个匹配出现的信息。

正则表达式对象保存用于查找字符串中的字符组合时的模式。创建**正则表达式**对象后，或者它被传递给字符串方法，或者字符串被传递给一个正则表达式方法。有关最近进行查找的信息被保存在全局 **RegExp** 对象中。

当预先知道查找字符串时用语法 1。当查找字符串经常变动或不知道时用语法 2，比如由用户输入得到的字符串。

在使用前 *pattern* 参数被编译为内部格式。对语法 1 来说，*pattern* 在该脚本被装载时被编译。对语法 2 来说，*pattern* 在使用前，或调用 **compile** 方法时被编译。

示例

下面的示例创建一个包含正则表达式模式及相关标志的对象(re)，向您演示**正则表达式**对象的用法。在本例中，作为结果的**正则表达式**对象又用于 **match** 方法中：

```
function MatchDemo() {
    var r, re; // 声明变量。

    var s = "The rain in Spain falls mainly
in the plain";

    re = new RegExp("Spain", "i"); // 创建正则表达式对象。

    r = s.match(re); // 在字符串 s
中查找匹配。
```

```
    return (r); // 返回匹配结果。
}
```

属性

[lastIndex 属性](#)(See 1.2.10.14) | [source 属性](#)(See 1.2.10.35)

方法

[compile 方法](#)(See 1.2.7.1.16) | [exec 方法](#)(See 1.2.7.1.27) | [test 方法](#)

要求

[版本 3](#)(See 2.64)

请参阅

[RegExp 对象](#)(See 1.2.8.12) | [正则表达式语法](#)(See 1.5.4) | [String 对象](#)(See 1.2.8.14)

1.2.8.14 String 对象

可用于处理或格式化文本字符串以及确定和定位字符串中的子字符串。

语法

```
newString = new String(["stringLiteral"])
```

参数

newString

必选项。要赋值为 **String** 对象的变量名。

stringLiteral

可选项。任意 Unicode 字符群。

说明

String 对象可用字符串文字显式创建。用这种方法创建的 **String** 对象（指以标准字符串形式）与用 **new** 运算符创建的 **String** 对象处理上不同。所有字符串文字共享公用的全局字符串对象。如果为字符串文字添加属性，则它对所有标准字符串对象都是可用的：

```
var alpha, beta;

alpha = "这是一个字符串";
```

```
beta = "这也是一个字符串";
```

```
alpha.test = 10;
```

在前一示例中，这时为 *beta* 和所有将来的字符串定义 *test*。然而，在下面的例子中，被添加属性的处理略有不同：

```
var gamma, delta;
```

```
gamma = new String("这是一个字符串");
```

```
delta = new String("这也是一个字符串");
```

```
gamma.test = 10;
```

在这种情况下，不为 *delta* 定义 *test*。每个用 **new String** 声明的 **String** 对象有其自己的一组成员。这是对 **String** 对象和字符串文字的处理不同的唯一情况。

属性

[constructor 属性](#)(See 1.2.10.4) | [length 属性](#)(See 1.2.10.17) | [prototype 属性](#)(See 1.2.10.34)

方法

[anchor 方法](#)(See 1.2.7.1.3) | [big 方法](#)(See 1.2.7.1.9) | [blink 方法](#)(See 1.2.7.1.10) | [bold 方法](#)(See 1.2.7.1.11) | [charAt 方法](#)(See 1.2.7.1.14) | [charCodeAt 方法](#)(See 1.2.7.1.15) | [concat 方法](#)(See 1.2.7.1.18) | [fixed 方法](#)(See 1.2.7.2.1) | [fontcolor 方法](#)(See 1.2.7.2.3) | [fontsize 方法](#)(See 1.2.7.2.4) | [fromCharCode 方法](#)(See 1.2.7.2.5) | [indexOf 方法](#)(See 1.2.7.2.27) | [italics 方法](#)(See 1.2.7.2.30) | [lastIndexOf 方法](#)(See 1.2.7.3.2) | [link 方法](#)(See 1.2.7.3.4) | [match 方法](#)(See 1.2.7.3.7) | [replace 方法](#)(See 1.2.7.3.19) | [search 方法](#)(See 1.2.7.4.1) | [slice 方法](#)(See 1.2.7.4.21) | [small 方法](#)(See 1.2.7.4.22) | [split 方法](#)(See 1.2.7.4.25) | [strike 方法](#)(See 1.2.7.4.27) | [sub 方法](#)(See 1.2.7.4.28) | [substr 方法](#)(See 1.2.7.4.29) | [substring 方法](#)(See 1.2.7.4.30) | [sup 方法](#)(See 1.2.7.4.31) | [toLowerCase 方法](#)(See 1.2.7.5.13) | [toUpperCase 方法](#)(See 1.2.7.5.17) | [toString 方法](#)(See 1.2.7.5.15) | [valueOf 方法](#)(See 1.2.7.5.23)

要求

[版本 1](#)(See 2.64)

请参阅

[new 运算符](#)(See 1.2.9.32)

1.2.8.15 VBAArray 对象

提供对 Visual Basic 安全数组的访问。

```
varName = new VBAArray(safeArray)
```

参数

varName

必选项。要赋值为 VBAArray 的变量名。

safeArray

必选项。VBAArray 值。

说明

VBAArrays 是只读的，并且不能被直接创建。在传递给 **VBAArray** 构造函数之前 *safeArray* 参数必须已获得 **VBAArray** 值。这只能通过从已有的 ActiveX 或其它对象获得值才能做到。

VBAArrays 可以有多维。每一维的下标可以不一样。**dimensions** 方法取得数组的维数；**lbound** 和 **ubound** 方法取得每个维的下标范围。

示例

下面的例子由三部分组成。第一部分是创建 Visual Basic 安全数组的 VBScript 代码。第二部分是將 VB 安全数组转换为 JScript 数组的 JScript 代码。这两部分都放到 HTML 页的 <HEAD> 节中。第三部分是放在 <BODY> 节中以运行其它两部分的 JScript 代码。

```
<HEAD>
```

```
<SCRIPT LANGUAGE="VBScript">
```

```
<!--
```

```
Function CreateVBAArray()
```

```
    Dim i, j, k
```

```
    Dim a(2, 2)
```

```
    k = 1
```

```
    For i = 0 To 2
```

```
        For j = 0 To 2
            a(j, i) = k
            document.writeln(k)
            k = k + 1
        Next
        document.writeln("vbCRLF")
    Next
    CreateVBAArray = a
End Function
-->
</SCRIPT>
```

```
<SCRIPT LANGUAGE="JScript">
<!--
function VBAArrayTest(vbarray) {
    var a = new VBAArray(vbarray);
    var b = a.toArray();
    var i;
    for (i = 0; i < 9; i++)
    {
        document.writeln(b[i]);
    }
}
```

```
}  
  
-->  
  
</SCRIPT>  
  
</HEAD>  
  
  
<BODY>  
  
<SCRIPT LANGUAGE="JScript">  
  
<!--  
  
    VBAArrayTest(CreateVBAArray());  
  
-->  
  
</SCRIPT>  
  
</BODY>
```

属性

VBAArray 对象没有属性。

方法

[dimensions 方法](#)(See 1.2.7.1.22) | [getItem 方法](#)(See 1.2.7.2.10) | [lbound 方法](#)(See 1.2.7.3.3) | [toArray 方法](#)(See 1.2.7.5.3) | [ubound 方法](#)(See 1.2.7.5.19)

要求

[版本 3](#)(See 2.64)

请参阅

[Array 对象](#)(See 1.2.8.2)

1.2.9 JScript 运算符

描述	语言要素
将两个数相加或连接两个字符串。	加法运算符 (+) (See 1.2.9.3)
将一个值赋给变量。	赋值运算符 (=) (See 1.2.9.4)_

对两个表达式执行按位与操作。	按位与运算符 (&) (See 1.2.9.6)_
将一个表达式的各位向左移。	按位左移运算符 (<<) (See 1.2.9.7)_
对一个表达式执行按位取非（求非）操作。	按位取非运算符 (~) (See 1.2.9.8)
对两个表达式指定按位或操作。	按位或运算符 () (See 1.2.9.10)
将一个表达式的各位向右移，保持符号不变。	按位右移运算符 (>>) (See 1.2.9.10)_
对两个表达式执行按位异或操作。	按位异或运算符 (^) (See 1.2.9.13)
使两个表达式连续执行。	逗号运算符 (,) (See 1.2.9.14)
返回 Boolean 值，表示比较结果。	比较运算符 (See 1.2.9.15)
复合赋值运算符列表。	复合赋值运算符 (See 1.2.9.16)
根据条件执行两个表达式之一。	条件（三元）运算符 (?:) (See 1.2.9.17)
将变量减一。	递减运算符 (--) (See 1.2.9.22)_
删除对象的属性，或删除数组中的一个元素。	delete 运算符 (See 1.2.9.18)
将两个数相除并返回一个数值结果。	除法运算符 (/) (See 1.2.9.20)
比较两个表达式，看是否相等。	相等运算符 (==) (See 1.2.9.20)
比较两个表达式，看一个是否大于另一个。	大于运算符 (>) (See 1.2.9.15)
比较两个表达式，看一个是否大于等于另一个。	大于等于运算符 (>=) (See 1.2.9.15)
比较两个表达式，看是否值相等并具有相同的数据类型。	严格相等运算符 (===) (See 1.2.9.15)
给变量加一。	递增运算符 (++) (See 1.2.9.22)
比较两个表达式，看是否不相等。	不等运算符 (!=) (See 1.2.9.15)
返回一个 Boolean 值，表明某个对象是否为特定类的一个实例。	instanceof 运算符 (See 1.2.9.23)
比较两个表达式，看是否一个小于另一个。	小于运算符 (<) (See 1.2.9.15)
比较两个表达式，看是否一个小于等于另一个。	小于等于运算符 (<=) (See 1.2.9.15)
对两个表达式执行逻辑与操作。	逻辑与运算符 (&&) (See 1.2.9.25)
对表达式执行逻辑非操作。	逻辑非运算符 (!) (See 1.2.9.26)
对两个表达式执行逻辑或操作。	逻辑或运算符 () (See 1.2.9.27)
将两个数相除，并返回余数。	取模运算符 (%) (See 1.2.9.29)
将两个数相乘。	乘法运算符 (*) (See 1.2.9.31)
创建一个新对象。	new 运算符 (See 1.2.9.32)
比较两个表达式，看是否具有不相等的值或数据类型不同。	非严格相等运算符 (!==) (See 1.2.9.15)
包含 JScript 运算符的执行优先级信息的列表。	运算符优先级 (See 1.2.9.1.1)
对两个表达式执行减法操作。	减法运算符 (-) (See 1.2.9.35)
返回一个表示表达式的数据类型的字符串。	typeof 运算符 (See 1.2.9.36)
表示一个数值表达式的相反数。	一元取相反数运算符 (-) (See 1.2.9.35)
在表达式中对各位进行无符号右移。	无符号右移运算符 (>>>) (See 1.2.9.37)
避免一个表达式返回值。	void 运算符 (See 1.2.9.39)

1.2.9.1 通用信息

1.2.9.1.1 运算符优先级

JScript 中的运算符优先级是一套规则。该规则在计算表达式时控制运算符执行的顺序。具有较高优先级的运算符先于较低优先级的运算符执行。例如，乘法的执行先于加法。

下表按从最高到最低的优先级列出 JScript 运算符。具有相同优先级的运算符按从左至右的顺序求值。

运算符	描述
. [] ()	字段访问、数组下标、函数调用以及表达式分组
++ -- ~ ! delete new typeof void	一元运算符、返回数据类型、对象创建、未定义值
* / %	乘法、除法、取模
+ - +	加法、减法、字符串连接
<< >> >>>	移位
< <= > >= instanceof	小于、小于等于、大于、大于等于、instanceof
== != === !==	等于、不等于、严格相等、非严格相等
&	按位与
^	按位异或
	按位或
&&	逻辑与
	逻辑或
?:	条件
= oP=	赋值、运算赋值
,	多重求值

圆括号可用来改变运算符优先级所决定的求值顺序。这意味着圆括号中的表达式应在其用于表达式的其余部分之前全部被求值。

例如：

```
z = 78 * (96 + 3 + 45)
```

在该表达式中有五个运算符：=, *, (), +, 以及另一个 +。根据运算符优先级的规则，它们将按下面的顺序求值：(), +, +, *, =。

1. 首先对圆括号内的表达式求值。圆括号中有两个加法运算符。因为两个加法运算符具有相同的优先级，从左到右求值。先将 96 和 3 相加，然后将其和与 45 相加，得到的结果为 144。
2. 然后是乘法运算。78 乘以 144，得到结果为 11232。

A 最后是赋值运算。将 11232 赋给 z。

1.2.9.1.2 运算符总结

计算运算符

运算符	符号
加法 (See 1.2.9.3)	+
递减 (See 1.2.9.22)	--
除法 (See 1.2.9.20)	/
递增 (See 1.2.9.22)	++
取模 (See 1.2.9.29)	%
乘法 (See 1.2.9.31)	*
减法 (See 1.2.9.35)	-
一元取相反数 (See 1.2.9.35)	--

逻辑运算符

运算符	符号
逗号 (See 1.2.9.14)	,
条件（三元） (See 1.2.9.17)	?:
相等 (See 1.2.9.15)	==
大于 (See 1.2.9.15)	>
大于等于 (See 1.2.9.15)	>=
严格相等 (See 1.2.9.15)	===
不等 (See 1.2.9.15)	!=
小于 (See 1.2.9.15)	<
小于等于 (See 1.2.9.15)	<=
逻辑与 (See 1.2.9.25)	&&
逻辑非 (See 1.2.9.26)	!
逻辑或 (See 1.2.9.27)	
非严格相等 (See 1.2.9.15)	!==

按位运算符

运算符	符号
按位与 (See 1.2.9.6)	&
按位左移 (See 1.2.9.7)	<<

按位取反 (See 1.2.9.8)	~
按位或 (See 1.2.9.10)	
按位右移 (See 1.2.9.11)	>>
按位异或 (See 1.2.9.13)	^
无符号右移 (See 1.2.9.37)	>>>

赋值运算符

[赋值运算符](#)(See 1.2.9.4)

[复合赋值运算符](#)(See 1.2.9.16)

其他运算符

[delete](#)(See 1.2.9.18)

[instanceof](#)(See 1.2.9.23)

[new](#)(See 1.2.9.32)

[typeof](#)(See 1.2.9.36)

[void](#)(See 1.2.9.39)

要求

[版本 1](#)(See 2.64)

1.2.9.2 加法赋值运算符 (+=)

将变量值与表达式值相加，并将和赋给该变量。

```
result += expression
```

参数

result

任何[变量](#)。

expression

任何[表达式](#)。

说明

使用本运算符与这样指定完全相同：

```
result = result + expression
```

表达式中加下划线的子类型决定 += 运算符的操作。

如果	那么

两个表达式都是数字或 Boolean 量	加
两个表达式都是字符串	连接
一个表达式是数字而另一个是字符串	连接

要求

[版本 1](#)(See 2.64)

请参阅

[+ 运算符](#)(See 1.2.9.3) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.3 加法运算符 (+)

将数字表达式的值加到另一数字表达式上，或连接两个字符串。

```
result = expression1 + expression2
```

参数

result

任何[变量](#)。

expression1

任何[表达式](#)。

expression2

任何表达式。

说明

表达式中加下划线的子类型决定 + 运算符的操作。

如果	那么
两个表达式都是数字或 Boolean 量	加
两个表达式都是字符串	连接
一个表达式是数字而另一个是字符串	连接

要求

[版本 1](#)(See 2.64)

请参阅

[+= 运算符](#)(See 1.2.9.2) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.4 赋值运算符 (=)

给变量赋值

```
result = expression
```

参数

result

任何[变量](#)。

expression

任何[数值表达式](#)。

说明

= 运算符和其他运算符一样,除了把值赋给变量外,使用它的表达式还有一个值。这就意味着可以象下面这样把赋值操作连起来写:

```
j = k = l = 0;
```

执行完该例子语句后, j、k、和 l 的值都等于零。

要求

[版本 1](#)(See 2.64)

请参阅

[运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.5 按位“与”赋值运算符 (&=)

对变量值与表达式值执行按位“与”,并将结果赋给该变量。

```
result &= expression
```

参数

result

任何[变量](#)。

expression

任何[表达式](#)。

说明

使用该运算符和使用下面的语句是等效的:

```
result = result & expression
```

&= 运算符查看 *result* 和 *expression* 的二进制表示法的值,并对其执行按位“与”操作。该操作的输出如下所示:

```
0101      (result)
```

```
1100      (expression)
```

```
-----
```

```
0100      (输出)
```

任何时候,只要两个表达式在某一位上都是 1,则结果的该位也是 1。否则,结果的该位是 0。

要求

[版本 1](#)(See 2.64)

请参阅

[& 运算符](#)(See 1.2.9.6) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.6 按位“与”运算符 (&)

对两个表达式执行按位“与”。

```
result = expression1 & expression2
```

参数

result

任何[变量](#)。

expression1

任何[表达式](#)。

expression2

任何表达式。

说明

& 运算符查看两个表达式的二进制表示法的值，并执行按位“与”操作。该操作的结果如下所示：

```
0101      (expression1)
```

```
1100      (expression2)
```

```
-----
```

```
0100      (result)
```

任何时候，只要两个表达式的某位都为 1，则结果的该位为 1。否则，结果的该位为 0。

要求

[版本 1](#)(See 2.64)

请参阅

[&= 运算符](#)(See 1.2.9.5) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.7 按位左移运算符 (<<)

左移表达式的位。

```
result = expression1 << expression2
```

参数

result

任何[变量](#)。

expression1

任何[表达式](#)。

expression2

任何表达式。

说明

<< 运算符把 *expression1* 的所有位向左移 *expression2* 指定的位数。例如：

```
var temp
```



```
temp = 14 << 2
```

变量 *temp* 的值为 56, 因为 14 (即二进制的 00001110) 向左移两位等于 56 (即二进制的 00111000)。

要求

[版本 1](#) (See 2.64)

请参阅

[<<= 运算符](#) (See 1.2.9.24) | [>> 运算符](#) (See 1.2.9.11) | [>>> 运算符](#) (See 1.2.9.37) | [运算符优先级](#) (See 1.2.9.1.1) | [运算符总结](#) (See 1.2.9.1.2)

1.2.9.8 按位“非”运算符 (~)

对一个表达式执行按位“非”（取反）。

```
result = ~ expression
```

参数

result

任何[变量](#)。

expression

任何[表达式](#)。

说明

所有一元运算符，如 ~ 运算符，以如下规则来求表达式的值：

- 如果将它用于 [undefined](#) 或 [null](#) 表达式，则产生一个运行时错误。
- 对象被转换成字符串。
- 如果可能，字符串被转换为数字。如果不能，则产生一个运行时错误。
- Boolean 值被当作数字（如果是 [false](#) 则为 0，如果是 [true](#) 则为 1）。

用该运算符来生成数字。

~ 运算符查看表达式的二进制表示法的值，并执行按位非操作。该操作的结果如下所示：

```
0101      (expression)
```

```
-----
```

1010 (result)

表达式中的任何一位为 1, 则在结果中相应位变为 0。表达式中的任何一位为 0, 则在结果中相应位变为 1。

要求

[版本 1](#)(See 2.64)

请参阅

[! 运算符](#)(See 1.2.9.26) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.9 按位“或”赋值运算符 (|=)

对变量值与表达式值执行按位“或”，并将结果赋给该变量。

```
result |= expression
```

参数

result

任何[变量](#)。

expression

任何[表达式](#)。

说明

使用该运算符和使用下面的语句是等效的：

```
result = result | expression
```

|= 运算符查看 *result* 和 *expression* 的二进制表示法的值，并执行按位“或”操作。该操作的结果如下所示：

0101 (result)

1100 (expression)

1101 (输出)

任何时候，只要两个表达式中的其中一个的某位是 1，则结果的该位是 1。否则，结果的该位是 0。

要求

[版本 1](#)(See 2.64)

请参阅

| [运算符](#)(See 1.2.9.10) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.10 按位“或”运算符 (|)

对两个表达式执行按位“或”

```
result = expression1 | expression2
```

参数

result

任何[变量](#)。

expression1

任何[表达式](#)。

expression2

任何表达式。

说明

| 运算符查看两个表达式的二进制表示法的值，并执行按位“或”操作。该操作的结果如下所示：

0101 (expression1)

1100 (expression2)

1101 (结果)

任何时候，只要任一表达式的一位为 1，则结果的该位为 1。否则，结果的该位为 0。

要求

[版本 1](#)(See 2.64)

请参阅

[|= 运算符](#)(See 1.2.9.9) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.11 按位右移运算符 (>>)

右移表达式的位，保持符号不变。

```
result = expression1 >> expression2
```

参数

result

任何[变量](#)。

expression1

任何[表达式](#)。

expression2

任何表达式。

说明

>> 运算符把 *expression1* 的所有位向右移 *expression2* 指定的位数。*expression1* 的符号位被用来填充右移后左边空出来的位。向右移出的位被丢弃。例如，下面的代码被求值后，*temp* 的值是 -4：-14 （即二进制的 11110010）右移两位等于 -4 （即二进制的 11111100）。

```
var temp
```

```
temp = -14 >> 2
```

要求

[版本 1](#)(See 2.64)

请参阅

[<< 运算符](#)(See 1.2.9.7) | [>= 运算符](#)(See 1.2.9.33) | [>>> 运算符](#)(See 1.2.9.37) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.12 按位“异或”赋值运算符 (^=)

对变量和表达式执行按位异或，并将结果赋给该变量。

```
result ^= expression
```

参数

result

任意[变量](#)。

expression

任意[表达式](#)。

说明

使用 ^= 运算符和使用下面的语句是等效的：

```
result = result ^ expression
```

^= 运算符查看两个表达式的二进制表示法的值，并执行按位异或。该操作的结果如下所示：

```
0101      (result)
1100      (expression)
-----
1001      (结果)
```

当且仅当只有一个表达式的某位为 1 时，结果的该位才为 1。否则，结果的该位为 0。

要求

[版本 1](#)(See 2.64)

请参阅

[^ 运算符](#)(See 1.2.9.13) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.13 按位“异或”运算符 (^)

对两个表达式执行按位异或。

```
result = expression1 ^ expression2
```

参数

result

任何[变量](#)。

expression1

任何[表达式](#)。

expression2

任何表达式。

说明

^ 运算符查看两个表达式的二进制表示法的值，并执行按位异或。该操作的结果如下所示：

0101 (expression1)

1100 (expression2)

1001 (结果)

当且仅当只有一个表达式的某位上为 1 时，结果的该位才为 1。否则结果的该位为 0。

要求

[版本 1](#)(See 2.64)

请参阅

[^= 运算符](#)(See 1.2.9.12) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.14 逗号运算符 (,)

顺序执行两个表达式。

expression1, expression2

参数

expression1

任何[表达式](#)。

expression2

任何表达式。

说明

，运算符使它两边的表达式以从左到右的顺序被执行，并获得右边表达式的值。，运算符最普通的用途是在 **for** 循环的递增表达式中使用。例如：

```
for (i = 0; i < 10; i++, j++)
{
    k = i + j;
}
```

每次通过循环的末端时，**for** 语句只允许单个表达式被执行。，运算符被用来允许多个表达式被当作单个表达式，从而规避该限制。

要求

[版本 1](#)(See 2.64)

请参阅

[for 语句](#)(See 1.2.11.8) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.15 比较运算符

返回表示比较结果的 Boolean 值。

expression1 **comparisonoperator** expression2

参数

expression1

任何[表达式](#)。

comparisonoperator

任何[比较运算符](#)。

expression2

任何表达式。

说明

比较字符串时，JScript 使用字符串表达式的 Unicode 字符值。

下面说明根据 *expression1* 和 *expression2* 的类型和值，不同组的运算符是如何作用的：

关系运算符（<、>、<=、>=）

- 试图将 *expression1* 和 *expression2* 都转换为数字。
- 如果两表达式均为字符串，则按字典序进行字符串比较。
- 如果其中一个表达式为 **NaN**，返回 **false**。
- 负零等于正零。
- 负无穷小于包括其本身在内的任何数。
- 正无穷大于包括其本身在内的任何数。

相等运算符（==、!=）

- 如果两表达式的类型不同，则试图将它们转换为字符串、数字或 **Boolean** 量。
- **NaN** 与包括其本身在内的任何值都不相等。
- 负零等于正零。
- **null** 与 **null** 和 **undefined** 相等。
- 相同的字符串、数值上相等的数字、相同的对象、相同的 **Boolean** 值或者（当类型不同时）能被强制转化为上述情况之一，均被认为是相等的。
- 其他比较均被认为是不相等的。

恒等运算符（===、!==）

除了不进行类型转换，并且类型必须相同以外，这些运算符与相等运算符的作用是一样的。

要求

[版本 1](#)(See 2.64)

请参阅

[运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.16 复合赋值运算符

运算符	符号
加法 (See 1.2.9.2)	+=
按位与 (See 1.2.9.5)	&=
按位或 (See 1.2.9.9)	=
按位异或 (See 1.2.9.12)	^=

除法 (See 1.2.9.19)	/=
左移 (See 1.2.9.24)	<<=
取模 (See 1.2.9.28)	%=
乘法 (See 1.2.9.30)	*=
右移 (See 1.2.9.33)	>>=
减法 (See 1.2.9.34)	-=
无符号右移 (See 1.2.9.38)	>>>=

要求

[版本信息](#)(See 2.64)

1.2.9.17 条件（三目）运算符 (?:)

根据条件执行两个语句中的其中一个。

`test ? 语句 1 : 语句 2`

参数

`test`

任何 [Boolean 表达式](#)。

语句 1

当 `test` 是 **true** 时执行的语句。可以是[复合语句](#)。

语句 2

当 `test` 是 **false** 时执行的语句。可以是复合语句。

说明

?: 运算符是 **if...else** 语句的快捷方式。它通常被用作较大表达式的一部分，而在此处使用 **if...else** 语句是不协调的。例如：

```
var now = new Date();
```

```
var      greeting      =      "Good"      +
```

```
((now.getHours() > 17) ? " evening." :
```

```
" day.");
```

在该例子中，如果是 6pm 以后，则创建一个包含 "Good evening." 的字符串。使用 **if...else** 语句的等效代码如下：

```
var now = new Date();  
  
var greeting = "Good";  
  
if (now.getHours() > 17)  
    greeting += " evening."  
else  
    greeting += " day.";
```

要求

[版本 1](#)(See 2.64)

请参阅

[if...else 语句](#)(See 1.2.11.11) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.18 delete 运算符

从对象中删除一个属性，或从数组中删除一个元素。

delete *expression*

expression 参数是一个有效的 JScript 表达式，通常是一个属性名或数组元素。

说明

如果 *expression* 的结果是一个对象，且在 *expression* 中指定的属性存在，而该对象又不允许它被删除，则返回 **false**。

在所有其他情况下，返回 **true**。

要求

[版本 3](#)(See 2.64)

请参阅

[运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.19 除法赋值运算符 (/=)

变量值除以表达式值，并将结果赋给该变量。

result /= *expression*

参数

result

任何数值[变量](#)。

expression

任何数值[表达式](#)。

说明

使用 /= 运算符和使用下面的语句是等效的：

```
result = result / expression
```

要求

[版本 1](#)(See 2.64)

请参阅

[/ 运算符](#)(See 1.2.9.20) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.20 除法运算符 (/)

将两个表达式的值相除。

```
result = number1 / number2
```

参数

result

任何数值[变量](#)。

number1

任何[数值表达式](#)。

number2

任何数值表达式。

要求

[版本 1](#) (See 2.64)

请参阅

[/= 运算符](#) (See 1.2.9.19) | [运算符优先级](#) (See 1.2.9.1.1) | [运算符总结](#) (See 1.2.9.1.2)

1.2.9.21 in 运算符

测试对象中是否存在该属性。

```
result = property in object
```

参数

result

必选项。任意[变量](#)。

property

必选项。相当于字符串[表达式](#)的一个表达式。

object

必选项。任意对象。

说明

in 操作检查对象中是否有名为 `property` 的属性。也可以检查对象的原型，以便知道该属性是否为原型链的一部分。

要求

[版本 1](#) (See 2.64)

请参阅

[运算符优先级](#) (See 1.2.9.1.1) | [运算符总结](#) (See 1.2.9.1.2)

1.2.9.22 递增 (++) 和递减 (--) 运算符

变量值递增一或递减一。

语法 1

```
result = ++variable
```

```
result = --variable
```

```
result = variable++
```

```
result = variable--
```

语法 2

```
++variable  
--variable  
variable++  
variable--
```

参数

result

任何[变量](#)。

variable

任何变量。

说明

递增和递减运算符，是修改存在变量中的值的快捷方式。包含其中一个这种运算符的表达式值，依赖于该运算符是在变量前面还是在变量后面：

```
var j, k;
```

```
k = 2;
```

```
j = ++k;
```

因为递增发生在表达式被求值前，所以值 3 赋给了 *j*。
和下面的例子对照：

```
var j, k;
```

```
k = 2;
```

```
j = k++;
```

在此，因为递增发生在表达式被求值后，所以值 2 赋给了 *j*。

要求

[版本 1](#)(See 2.64)

请参阅

[运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.23 instanceof 运算符

返回一个 Boolean 值，指出对象是否是特定[类](#)的一个实例。

```
result = object instanceof class
```

参数

result

必选项。任意[变量](#)。

object

必选项。任意对象[表达式](#)。

class

必选项。任意已定义的对象类。

说明

如果 *object* 是 *class* 的一个实例，则 **instanceof** 运算符返回 **true**。如果 *object* 不是指定类的一个实例，或者 *object* 是 **null**，则返回 **false**。

示例

下面的例子举例说明了 **instanceof** 运算符的用法。

```
function objTest(obj) {  
    var i, t, s = "";    // 创建变量。  
    t = new Array();    // 创建一个数组。  
    t["Date"] = Date;    // 填充数组。  
    t["Object"] = Object;  
    t["Array"] = Array;  
    for (i in t)  
    {  
        if (obj instanceof t[i])    // 检
```

查 obj 的类。

```

    {
        s += "obj is an instance of " + i
        + "\n";
    }
    else
    {
        s += "obj is not an instance
of " + i + "\n";
    }
}

return(s);    // 返回字符串。
}

```

```

var obj = new Date();
response.write(objTest(obj));

```

要求

[版本 5](#)(See 2.64)

请参阅

[运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.24 左移赋值运算符 (<<=)

变量值根据表达式值所规定的位数进行左移，并将结果赋给该变量

```
result <<= expression
```

参数

result

任何[变量](#)。

expression

任何[表达式](#)。

说明

使用 `<<=` 运算符与使用下面的语句是等效的：

```
result = result << expression
```

`<<=` 运算符把 *result* 的所有位向左移动 *expression* 所指定的位数。例如：

```
var temp
```

```
temp = 14
```

```
temp <<= 2
```

变量 *temp* 的值是 56，因为 14（即二进制的 00001110）左移两位等于 56（即二进制的 00111000）。在移动时用零来填充右边空出的位。

要求

[版本 1](#)(See 2.64)

请参阅

[<< 运算符](#)(See 1.2.9.7) | [>> 运算符](#)(See 1.2.9.11) | [>>> 运算符](#)(See 1.2.9.37) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.25 逻辑“与”运算符 (&&)

两个表达式执行逻辑联合。

```
result = expression1 && expression2
```

参数

result

任何[变量](#)。

expression1

任何[表达式](#)。

expression2

任何表达式。

说明

当且仅当两个表达式的值都等于 **True** 时，*result* 才是 **True**。如果任一表达式的值等于 **False**，则 *result* 为 **False**。

JScript 使用下面的规则，来把非 Boolean 值转换为 Boolean 值：

- 所有对象都被认为是 **true**。
- 字符串当且仅当为空时被认为是 **false**。
- **null** 和 [undefined](#) 被认为是 **false**。
- 数字当且仅当为零时是 **false**。

要求

[版本 1](#)(See 2.64)

请参阅

[运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.26 逻辑“非”运算符 (!)

对一个表达式执行逻辑非。

```
result = !expression
```

参数

result

任何[变量](#)。

expression

任何[表达式](#)。

说明

下面的表举例说明了 *result* 是如何决定的。

如果 expression 是	则 result 是
True	False
False	True

所有一元运算符，如 **!** 运算符，按照下面的规则来求表达式的值：

- 如果应用于 [undefined](#) 或 **null** 表达式，则会产生一个[运行时错误](#)。
- 对象被转换为字符串。
- 如果可能，字符串被转换为数值。否则会产生一个运行时错误。
- Boolean 值被当作数值（如果是 **false** 则为 0，如果是 **true** 则为 1）。

该运算符被用来产生数值。

对于 **!** 运算符，如果 *expression* 是非零的，则 *result* 是零。如果 *expression* 是零，则 *result* 是 1。

要求

[版本 1](#)(See 2.64)

请参阅

[~ 运算符](#)(See 1.2.9.8) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.27 逻辑“或”运算符 (||)

对两个表达式执行逻辑“或”。

```
result = expression1 || expression2
```

参数

result

任何[变量](#)。

expression1

任何[表达式](#)。

expression2

任何表达式。

说明

如果两个表达式中有一个或两个等于 **True**，则 *result* 是 **True**。下面的表举例说明了 *result* 是如何被决定的：

如果 expression1 是	且 expression2 是	则 result 是
True	True	True
True	False	True
False	True	True

False	False	False

JScript 使用下面的规则来把非 Boolean 值转换为 Boolean 值：

- 所有对象都被认为是 **true**。
- 字符串当且仅当为空时才被认为是 **false**。
- **null** 和 [undefined](#) 被认为是 **false**。
- 数字当且仅当为 0 时才是 **false**。

要求

[版本 1](#)(See 2.64)

请参阅

[运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.28 取余赋值运算符 (%=)

变量值除以表达式值，并将余数赋给变量。

```
result %= expression
```

参数

result

任何[变量](#)。

expression

任何[数值表达式](#)。

说明

使用 %= 运算符与使用下面的语句是等效的：

```
result = result % expression
```

要求

[版本 1](#)(See 2.64)

请参阅

[% 运算符](#)(See 1.2.9.29) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.29 取余运算符 (%)

一个表达式的值除以另一个表达式的值，返回余数。

```
result = number1 % number2
```

参数

result

任何[变量](#)。

number1

任何[数值表达式](#)。

number2

任何数值表达式。

说明

取余（或余数）运算符用 *number1* 除以 *number2*（把浮点数四舍五入为整数），然后只返回余数作为 *result*。例如，在下面的表达式中，A（即 *result*）等于 5。

```
A = 19 % 6.7
```

要求

[版本 1](#)(See 2.64)

请参阅

[%= 运算符](#)(See 1.2.9.28) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.30 乘法赋值运算符 (*=)

变量值乘以表达式值，并将结果赋给该变量。

```
result *= expression
```

参数

result

任何[变量](#)。

expression

任何[表达式](#)。

说明

使用 `*=` 运算符和使用下面的语句是等效的：

```
result = result * expression
```

要求

[版本 1](#)(See 2.64)

请参阅

[* 运算符](#)(See 1.2.9.31) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.31 乘法运算符 (*)

两个表达式的值相乘。

```
result = number1*number2
```

参数

result

任何[变量](#)。

number1

任何[表达式](#)。

number2

任何表达式。

要求

[版本 1](#)(See 2.64)

请参阅

[*= 运算符](#)(See 1.2.9.30) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.32 new 运算符

创建一个新对象。

```
new constructor[(arguments)]
```

参数

constructor

必选项。对象的[构造函数](#)。如果构造函数没有参数，则可以省略圆括号。

arguments

可选项。任意传递给新对象构造函数的参数。

说明

new 运算符执行下面的任务：

- 创建一个没有成员的对象。
- 为那个对象调用构造函数，传递一个指针给新创建的对象作为 **this** 指针。
- 然后构造函数根据传递给它的参数初始化该对象。

示例

下面这些是有效的 **new** 运算符的用法例子。

```
my_object = new Object;  
my_array = new Array();  
my_date = new Date("Jan 5 1996");
```

要求

版本 1

请参阅

[function 语句](#)(See 1.2.11.10)

1.2.9.33 右移赋值运算符 (>>=)

变量值右移表达式值所规定的位数，保持符号不变，并将结果赋给该变量。

```
result >>= expression
```

参数

result

任何[变量](#)。

expression

任何[表达式](#)。

说明

使用 `>>=` 运算符和使用下面的语句是等效的：

```
result = result >> expression
```

`>>=` 运算符把 *result* 的所有位向右移 *expression* 指定的位数。*result* 的符号位被用来填充右移后左边空出的位。从右边移出去的位被丢弃。例如，下面的代码被求值后，*temp* 的值是 **-4**：-14（即二进制的 11110010）右移两位等于 -4（即二进制的 11111100）。

```
var temp
```

```
temp = -14
```

```
temp >>= 2
```

要求

[版本 1](#)(See 2.64)

请参阅

[<< 运算符](#)(See 1.2.9.7) | [>> 运算符](#)(See 1.2.9.11) | [>>> 运算符](#)(See 1.2.9.37) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.34 减法赋值运算符 (-=)

从变量值中减去表达式值，并将结果赋给该变量。

```
result -= expression
```

参数

result

任何数值[变量](#)。

expression

任何[数值表达式](#)。

说明

使用 `-=` 运算符与使用下面的语句是等效的：

```
result = result - expression
```

要求

[版本 1](#) (See 2.64)

请参阅

- [运算符](#) (See 1.2.9.35) | [运算符优先级](#) (See 1.2.9.1.1) | [运算符总结](#) (See 1.2.9.1.2)

1.2.9.35 减法运算符 (-)

从一个表达式的值中减去另一个表达式的值，只有一个表达式时取其相反数。

语法 1

```
result = number1 - number2
```

语法 2

```
-number
```

参数

result

任何数值[变量](#)。

number

任何[数值表达式](#)。

number1

任何数值表达式。

number2

任何数值表达式。

说明

在语法 1 中，`-` 运算符是算术减法运算符，用来获得两个数值之间的差。在语法 2 中，`-` 运算符被用作一元取负运算符，用来指出一个表达式的负值。

对于语法 2，和所有一元运算符一样，表达式按照下面的规则来求值：

- 如果应用于 [undefined](#) 或 `null` 表达式，则会产生一个运行时错误。
- 对象被转换为字符串。
- 如果可能，则字符串被转换为数值。如果不能，则会产生一个运行时错误。
- Boolean 值被当作数值（如果是 `false` 则为 0，如果是 `true` 则为 1）。

该运算符被用来产生数值。在语法 2 中，如果生成的数值不是零，则 *result* 与生成的数值颠倒符号后是相等的。如果生成的数值是零，则 *result* 是零。

要求

[版本 1](#)(See 2.64)

请参阅

[-= 运算符](#)(See 1.2.9.34) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.36 typeof 运算符

返回一个用来表示表达式的数据类型的字符串。

typeof [() *expression*] ;

expression 参数是需要查找类型信息的任意[表达式](#)。

说明

typeof 运算符把类型信息当作字符串返回。**typeof** 返回值有六种可能： "number," "string," "boolean," "object," "function," 和 "undefined."

typeof 语法中的圆括号是可选项。

要求

[版本 1](#)(See 2.64)

请参阅

[运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.37 无符号右移运算符 (>>>)

右移表达式的位，不保留符号。

`result = expression1 >>> expression2`

参数

`result`

任何[变量](#)。

expression1

任何[表达式](#)。

expression2

任何表达式。

说明

>>> 运算符把 *expression1* 的各个位向右移 *expression2* 指定的位数。右移后左边空出的位用零来填充。移出右边的位被丢弃。例如：

```
var temp
```

```
temp = -14 >>> 2
```

变量 *temp* 的值为 -14 （即二进制的 11111111 11111111 11111111 11110010），向右移两位后等于 1073741820 （即二进制的 00111111 11111111 11111111 11111100）。

要求

[版本 1](#)(See 2.64)

请参阅

[>>>= 运算符](#)(See 1.2.9.38) | [<< 运算符](#)(See 1.2.9.7) | [>> 运算符](#)(See 1.2.9.11) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.38 无符号右移赋值操作 (>>>=)

对变量值根据表达式值所规定的位数进行无符号右移，并将结果赋给该变量。

```
result >>>= expression
```

参数

result

任何[变量](#)。

expression

任何[表达式](#)。

说明

使用 `>>>=` 运算符和使用下面的语句是等效的：

```
result = result >>> expression
```

`>>>=` 运算符把 *result* 的所有位向右移 *expression* 指定的位数。右移后左边空出的位用零来填充。向右移出的位被丢弃。例如：

```
var temp
```

```
temp = -14
```

```
temp >>>= 2
```

变量 *temp* 的值为 -14（即二进制的 11111111 11111111 11111111 11110010），向右移两位后等于 1073741820（即二进制的 00111111 11111111 11111111 11111100）。

要求

[版本 1](#)(See 2.64)

请参阅

[>>> 运算符](#)(See 1.2.9.37) | [<< 运算符](#)(See 1.2.9.7) | [>> 运算符](#)(See 1.2.9.11) | [运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)(See 1.2.9.1.2)

1.2.9.39 void 运算符

避免表达式返回值。

```
void expression
```

expression 参数是任意有效的 JScript [表达式](#)。

说明

void 运算符对表达式求值，并返回 [undefined](#)。在希望求表达式的值，但又不希望脚本的剩余部分看见这个结果时，该运算符最有用。

要求

[版本 2](#)(See 2.64)

请参阅

[运算符优先级](#)(See 1.2.9.1.1) | [运算符总结](#)

1.2.10 JScript 属性

描述	语言要素
返回在模式匹配中找到的最近的九条记录。	\$1...\$9 Properties (See 1.2.10.1)
返回一个包含传递给当前执行函数的每个参数	arguments 属性 (See 1.2.10.2)

的数组。	
返回调用当前函数的函数引用。	caller 属性 (See 1.2.10.4)
指定创建对象的函数。	constructor 属性 (See 1.2.10.4)
返回或设置关于指定错误的描述字符串。	description 属性 (See 1.2.10.5)
返回 Euler 常数，即自然对数的底。	E 属性 (See 1.2.10.6)
返回在字符串中找到的第一个成功匹配的字符位置。	index 属性 (See 1.2.10.10)
返回 Number.POSITIVE_INFINITY 的初始值。	Infinity 属性 (See 1.2.10.11)
返回进行查找的字符串。	input 属性 (See 1.2.10.12)
返回在字符串中找到的最后一个成功匹配的字符位置。	lastIndex 属性 (See 1.2.10.14)_
返回比数组中所定义的最高元素大 1 的一个整数。	length 属性 (Array) (See 1.2.10.15)
返回为函数所定义的参数个数。	length 属性 (Function) (See 1.2.10.16)
返回 String 对象的长度。	length 属性 (String) (See 1.2.10.17)
返回 2 的自然对数。	LN2 属性 (See 1.2.10.19)
返回 10 的自然对数。	LN10 属性 (See 1.2.10.18)
返回以 2 为底的 e（即 Euler 常数）的对数。	LOG2E 属性 (See 1.2.10.21)
返回以 10 为底的 e（即 Euler 常数）的对数。	LOG10E 属性 (See 1.2.10.20)
返回在 JScript 中能表示的最大值。	MAX_VALUE 属性 (See 1.2.10.22)
返回在 JScript 中能表示的最接近零的值。	MIN_VALUE 属性 (See 1.2.10.22)
返回特殊值 NaN ，表示某个表达式不是一个数。	NaN 属性 (Global) (See 1.2.10.28)
返回特殊值 (NaN)，表示某个表达式不是一个数。	NaN 属性 (Number) (See 1.2.10.27)
返回比在 JScript 中能表示的最大的负数（- Number.MAX_VALUE ）更负的值。	NEGATIVE_INFINITY 属性 (See 1.2.10.29)
返回或设置与特定错误关联的数值。	number 属性 (See 1.2.10.30)
返回圆周与其直径的比值，约等于 3.141592653589793。	PI 属性 (See 1.2.10.31)
返回比在 JScript 中能表示的最大的数（ Number.MAX_VALUE ）更大的值。	POSITIVE_INFINITY 属性 (See 1.2.10.32)
返回对象类的原型引用。	prototype 属性 (See 1.2.10.34)
返回正则表达式模式的文本的拷贝。	source 属性 (See 1.2.10.35)
返回 0.5 的平方根，即 1 除以 2 的平方根。	SQRT1_2 属性 (See 1.2.10.36)
返回 2 的平方根。	SQRT2 属性 (See 1.2.10.37)

1.2.10.1 \$1...\$9 属性

返回九个在模式匹配期间找到的、最近保存的部分。只读。

RegExp. \$*n*

参数

RegExp

总是全局 **RegExp** 对象。

n

1 至 9. 间的任意整数。

说明

无论何时产生一个成功的带插入语的匹配，**\$1...\$9** 属性的值都被修改，但是只有最近的九个可以被保存起来。

下面的例子举例说明了 **\$1...\$9** 属性的用法：

```
function matchDemo() {  
    var s;  
    var re = new RegExp("d(b+) (d)", "ig");  
    var str = "cdbBdbbsbdbdz";  
    var arr = re.exec(str);  
    s = "$1 contains: " + RegExp.$1 +  
    "\n";  
    s += "$2 contains: " + RegExp.$2 +  
    "\n";  
    s += "$3 contains: " + RegExp.$3;  
    return(s);  
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[RegExp 对象的属性](#)(See 1.2.8.12) | [正则表达式语法](#)(See 1.5.4)

应用于: [RegExp 对象](#)(See 1.2.8.12)

1.2.10.2 arguments 属性

为当前执行的 **function** 对象返回一个 **arguments** 对象。

function.arguments

function 参数是当前执行函数的名称，可以省略。

说明

通过 **arguments** 属性，函数可以处理可变数量的参数。 **arguments** 对象的 **length** 属性包含了传递给函数的参数的数目。对于 **arguments** 对象所包含的单个参数，其访问方法与数组中所包含的参数的访问方法相同。

示例

下面的例子说明了 **arguments** 属性的用法：

```
function ArgTest() {  
    var i, s, numargs = arguments.length;  
    s = numargs;  
    if (numargs < 2)  
        s += " argument was passed to  
ArgTest. It was ";  
    else  
        s += " arguments were passed to  
ArgTest. They were " ;  
    for (i = 0; i < numargs; i++)  
    {
```

```
        s += arguments[i] + " ";
    }

    return(s);
}
```

要求

[版本 2](#)(See 2.64)

请参阅

[function 语句](#)(See 1.2.11.10) | [length 属性 \(Array\)](#)(See 1.2.10.15)

应用于: [Function 对象](#)(See 1.2.8.7)

1.2.10.3 caller 属性

返回一个对函数的引用，该函数调用了当前函数。

functionName.**caller**

functionName 对象是所执行函数的名称。

说明

对于函数来说，**caller** 属性只有在函数执行时才有定义。如果函数是由 JScript 程序的顶层调用的，那么 **caller** 包含的就是 **null**。

如果在字符串上下文中使用 **caller** 属性，那么结果和 *functionName.toString* 一样，也就是说，显示的是函数的反编译文本。

下面的例子说明了 **caller** 属性的用法：

```
function CallLevel() {
    if (CallLevel.caller == null)
        return("CallLevel was called from
the top level.");
    else
        return("CallLevel was called by
another function.");
}
```

要求

[版本 2](#)(See 2.64)

请参阅

[function 语句](#)(See 1.2.11.10)

应用于: [Function 对象](#)(See 1.2.8.7)

1.2.10.4 constructor 属性

表示创建对象的函数。

object.**constructor**

必需的 *object* 是对象或函数的名称。

说明

constructor 属性是所有具有 *prototype* 的对象的成员。它们包括除 **Global** 和 **Math** 对象以外的所有 [JScript 固有对象](#)。**constructor** 属性保存了对构造特定对象实例的函数的引用。 例如:

```
x = new String("Hi");

if (x.constructor == String)

    // 进行处理（条件为真）。
```

或

```
function MyFunc {

    // 函数体。
}

y = new MyFunc;

if (y.constructor == MyFunc)

    // 进行处理（条件为真）。
```

要求

[版本 2](#)(See 2.64)

请参阅

[prototype 属性](#)(See 1.2.10.34)

应用于: [Array 对象](#)(See 1.2.8.2) | [Boolean 对象](#)(See 1.2.8.3) | [Date 对象](#)(See 1.2.8.4)

| [Function 对象](#)(See 1.2.8.7) | [Math 对象](#)(See 1.2.8.9) | [Number 对象](#)(See 1.2.8.10) | [Object 对象](#)(See 1.2.8.11) | [String 对象](#)(See 1.2.8.14)

1.2.10.5 description 属性

返回或设置与特定错误相联系的描述字符串。

`object.description [= stringExpression]`

description 属性的语法组成部分如下：

参数

`object`

必选项。**Error** 对象的任意实例。

`stringExpression`

可选项。包含错误描述的[字符串表达式](#)。

说明

description 属性包含与特定错误相联系的错误信息字符串。使用包含在这个中的值，来警告用户发生了一个不能或不想处理的错误。

下面的例子举例说明了 **description** 属性的使用：

```
try
    x = y    // 产生错误。
catch (var e) {    // 创建局部变量 e。
    document.write(e)    // 打印 "[object
Error]".
    document.write((e.number    &
0xFFFF))    // 打印 5009.
    document.write(e.description)    // 打印
"'y' is undefined".
}
```

要求

[版本 5](#)(See 2.64)

请参阅

[number 属性](#) (See 1.2.10.30)

应用于: [Error 对象](#)(See 1.2.8.6)

1.2.10.6 E 属性

返回 Euler 常数, 自然对数的底。**E** 属性约等于 2.718。

```
numVar = Math.E
```

要求

[版本 1](#)(See 2.64)

请参阅

[exp 方法](#)(See 1.2.7.1.28) | [Math 对象属性](#)(See 1.2.8.9)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.10.7 global 属性

返回 Boolean 值, 指出正则表达式使用的 global 标志 (g) 的状态。默认值为 **false**。
只读。

```
rgExp.global
```

必选项 *rgExp* 参数是正则表达式对象。

说明

如果正则表达式设置了 global 标志, 那么 **global** 属性返回 **true**, 否则返回 **false**。
使用 global 标志表明在被查找的字符串中搜索操作将查找所有符合的项, 而不仅仅是第一个。这也被称为全局匹配。

示例

以下示例演示了 **global** 属性的用法。如果传递 "g" 到下面所示的函数中, 那么所有的单词 "the" 将被 "a" 代替。请注意, 字符串首的 "The" 不会被替换。这是因为第一个字母是大写的, 因此, 不能与 "the" 中小写的 "t" 匹配。

本函数返回一个字符串以及一个表, 表中显示了与允许使用的正则表达式标志 (**g**、**i** 和 **m**) 相关的属性值。它还返回经过所有替换操作后的字符串。

```
function RegExpPropDemo(flag) {  
    if (flag.match(/[^gim]/))                // 检查标  
志的有效性。
```

```
        return("Flag specified is not
valid");

    var r, re, s                                // 声明变量。

    var ss = "The man hit the ball with
the bat.\n";

    ss += "while the fielder caught the
ball with the glove.";

    re = new RegExp("the",flag);                // 指定要
查找的样式。

    r = ss.replace(re, "a");                    // 用
"a" 替换 "the"。

    s = "Regular Expression property
values:\n\n"

    s += "global                ignoreCase
multiline\n"

    if (re.global)                            // 测试
global 标志。

        s += " True ";
    else

        s += "False ";

    if (re.ignoreCase)                          // 测试
ignoreCase 标志。

        s += " True ";
```

```

else
    s += "False ";
    if (re.multiline) // 测试
multiline 标志。
        s += "      True      ";
    else
        s += "      False  ";
    s += "\n\nThe resulting string
is:\n\n" + r;
    return(s); // 返回替换字符
串。
}

```

要求

[版本 5.5](#)(See 2.64)

请参阅

[ignoreCase 属性](#)(See 1.2.10.9) | [multiline 属性](#)(See 1.2.10.25) | [正则表达式语法](#)(See 1.5.4)

应用于: [RegExp 对象](#)(See 1.2.8.12)

1.2.10.8 hasOwnProperty 方法

返回一个布尔值，指出一个对象是否具有指定名称的属性。

object.**hasOwnProperty**(*proName*)

参数

object

必选项。一个对象的实例。

proName

必选项。一个属性名称的字符串值。

说明

如果 *object* 具有指定名称的属性，那么 **hasOwnProperty** 方法返回 **true**；反之则返回 **false**。此方法无法检查该对象的原型链中是否具有该属性；该属性必须是对象本身的一个成员。

示例

在下例中，所有的 **String** 对象共享一个公用 **split** 方法。下面的代码将输出 **false** 和 **true**。

```
var s = new String("JScript");  
  
print(s.hasOwnProperty("split"));  
  
print(String.prototype.hasOwnProperty("split"));
```

要求

[版本 5.5](#)(See 2.64)

请参阅

[in 运算符](#)(See 1.2.9.21)

应用于： [Object 对象](#)(See 1.2.8.11)

1.2.10.9 ignoreCase 属性

返回 Boolean 值，指出正则表达式使用的 **ignoreCase** 标志(**i**) 的状态。默认值为 **false**。只读。

rgExp.ignoreCase

必选项 *rgExp* 参数为 **RegExp** 对象。

说明

如果正则表达式设置了 **ignoreCase** 标志，那么 **ignoreCase** 属性返回 **true**，否则返回 **false**。

如果使用了 **ignoreCase** 标志，那就表明在被查找的字符串中匹配样式的时候查找操作将不区分大小写。

示例

以下示例演示了 **ignoreCase** 属性的用法。如果传递 "i" 到下面所示的函数中，那么所有的单词 "the" 将被 "a" 替换，包括最开始位置上的 "The"。这是因为设置了 **ignoreCase** 标志，搜索操作将不区分大小写。所以在进行匹配的时候 "T" 与

"t" 是等价的。

此函数返回一个字符串以及一个表，表中显示了与允许使用的正则表达式标志（**g**、**i** 和 **m**）相关的属性值。它还返回经过所有替换操作后的字符串。

```
function RegExpPropDemo(flag) {
    if (flag.match(/^[^gim]/))                // 检查标
志的有效性。
        return("Flag specified is not
valid");

    var r, re, s                                // 声明变量。
    var ss = "The man hit the ball with
the bat.\n";
    ss += "while the fielder caught the
ball with the glove.";
    re = new RegExp("the", flag);              // 指定要
查找的样式。
    r = ss.replace(re, "a");                   // 利用
"a" 替换 "the"。
    s = "Regular Expression property
values:\n\n"
    s += "global ignoreCase
multiline\n"
    if (re.global)                            // 测试
global 标志。
        s += " True ";
    else
```

```

        s += "False    ";
        if (re.ignoreCase)                                // 测试
ignoreCase 标志。

        s += " True  ";
    else
        s += "False  ";
        if (re.multiline)                                // 测试
multiline 标志。

        s += "      True      ";
    else
        s += "      False     ";
        s += "\n\nThe    resulting    string
is:\n\n" + r;

    return(s);                                           // 返回替换的字
字符串
}

```

要求

[版本 5.5](#)(See 2.64)

请参阅

[global 属性](#)(See 1.2.10.7) | [multiline 属性](#)(See 1.2.10.25) | [正则表达式语法](#)(See 1.5.4)
 应用于: [RegExp 对象](#)(See 1.2.8.12)

1.2.10.10 index 属性

返回字符位置，它是查找字符串中第一个成功匹配的开始位置。只读。

RegExp.index

该属性的相关对象总是全局 **RegExp** 对象。

说明

index 属性是基于零的。其初始值为 -1，不论何时产生一个成功匹配，它的值都将改变。

示例

下例阐明了 **index** 属性的用法。该函数重复了字符串查找，并将字符串中每一个字的 **index** 和 **lastIndex** 值都打印出来。

```
function RegExpTest() {  
    var ver =  
Number (ScriptEngineMajorVersion() + "."  
+ ScriptEngineMinorVersion())  
    if (ver >= 5.5) {  
        var src = "The rain in Spain falls  
mainly in the plain.";  
        var re = /\w+/g;  
        var arr;  
        while ((arr = re.exec(src)) != null)  
            print(arr.index + " - " +  
arr.lastIndex + "\t" + arr);  
    }  
    else {  
        alert("You need a newer version of  
JScript for this to work");  
    }  
}
```


要求

[版本 3](#)(See 2.64)

请参阅

[RegExp 对象的属性](#)(See 1.2.8.12) | [正则表达式语法](#)(See 1.5.4)

应用于: [RegExp 对象](#)(See 1.2.8.12)

1.2.10.11 Infinity 属性

返回 Number.POSITIVE_INFINITY 的初始值。

Infinity

说明

Infinity 属性是 **Global** 对象的成员, 在 Scripting 引擎初始化时变为可用。

要求

[版本 3](#)(See 2.64)

请参阅

[POSITIVE_INFINITY 属性](#)(See 1.2.10.32) | [NEGATIVE_INFINITY 属性](#)(See 1.2.10.29)

应用于: [Global 对象](#)(See 1.2.8.8)

1.2.10.12 input 属性(\$_)

返回执行规范表述查找的字符串。只读。

RegExp.input

与该属性有关的对象总是全局 **RegExp** 对象。

说明

任何时候改变了被查找的字符串, **input** 属性的值都会被修改。

下面的例子举例说明了 **input** 属性的用法:

```
function inputDemo() {  
  
    var s;  
  
    var re = new RegExp("d(b+) (d)", "ig");  
  
    var str = "cdbBdbsbdbdz";  
  
    var arr = re.exec(str);
```

```

    s = "The string used for the match
was " + RegExp.input;

    return(s);
}

```

要求

[版本 3](#)(See 2.64)

请参阅

[RegExp 对象的属性](#)(See 1.2.8.12) | [正则表达式语法](#)(See 1.5.4)

应用于: [RegExp 对象](#)(See 1.2.8.12)

1.2.10.13 isPrototypeOf 方法

返回一个布尔值，指出对象是否存在于另一个对象的原型链中。

object1.isPrototypeOf(object2)

参数

object1

必选项。一个对象的实例。

object2

必选项。另一个对象，将要检查其原型链。

说明

如果 *object2* 的原型链中包含 *object1*，那么 **isPrototypeOf** 方法返回 **true**。原型链可以用来在同一个对象类型的不同实例之间共享功能。如果 *object2* 不是一个对象或者 *object1* 没有出现在 *object2* 中的原型链中，**isPrototypeOf** 方法将返回 **false**。

示例

以下示例阐述了 **isPrototypeOf** 方法的用法。

```

function test() {

    var      re      =      new      RegExp();

```

```
// 初始化变量。

return

(RegExp.prototype.isPrototypeOf(re));

// 返回 true。
}
```

请参阅

应用于: [Object 对象](#)(See 1.2.8.11)

1.2.10.14 lastIndex 属性

返回字符位置，它是被查找字符串中下一次成功匹配的开始位置。

RegExp.lastIndex

与该属性有关的对象总是全局 **RegExp** 对象。

说明

lastIndex 属性是基于零的，也就是说，第一个字符的索引是零。其初始值为 -1，不论何时产生一个成功匹配，它的值都被修改。

lastIndex 属性被 **RegExp** 对象的 **exec** 和 **test** 方法，以及 **String** 对象的 **match**、**replace**、和 **split** 方法修改。

下面的规则应用于 **lastIndex** 的值：

- 如果还没有匹配，则 **lastIndex** 被设置为 -1。
- 如果 **lastIndex** 比字符串的长度大，则 **test** 和 **exec** 失败，并且 **lastIndex** 被设置为 -1。
- 如果 **lastIndex** 等于字符串的长度，且模式与空字符串匹配，则正则表达式匹配。否则，匹配失败并且 **lastIndex** 被重置为 -1。
- 否则，**lastIndex** 被设置为紧接最近的匹配的下一个位置。

示例

下例阐明了 **lastIndex** 属性的用法。该函数重复了查找字符串，并打印出字符串中每一个字的 **index** 和 **lastIndex** 值。

```
function RegExpTest() {

    var                                ver                                =

    Number(ScriptEngineMajorVersion() + ".")
```

```
+ ScriptEngineMinorVersion())
    if (ver >= 5.5){
        var src = "The rain in Spain falls
mainly in the plain.";
        var re = /\w+/g;
        var arr;
        while ((arr = re.exec(src)) != null)
            print(arr.index + "-" +
arr.lastIndex + "\t" + arr);
    }
    else{
        alert("You need a newer version of
JScript for this to work");
    }
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[RegExp 对象的属性](#)(See 1.2.8.12) | [正则表达式语法](#)(See 1.5.4)

应用于: [RegExp 对象](#)(See 1.2.8.12)

1.2.10.15 length 属性 (Array)

返回一个整数值，这个整数比数组中所定义的最高位元素的下标大 1。

numVar = *arrayObj*.**length**

参数

numVar

必选项。任意数值。

arrayObj

必选项。任意 **Array** 对象。

说明

因为一个数组中的元素并不一定是连续的，所以 **length** 属性也并不一定就等于数组中的元素个数。例如，在下面的数组定义中，`my_array.length` 中就包含 7，而不是 2：

```
var my_array = new Array( );  
my_array[0] = "Test";  
my_array[6] = "Another Test";
```

如果 **length** 属性被赋予了一个比原先值小的数值，那么数组就被截断，所有数组下标等于或者大于 **length** 属性的新值的元素都会被丢失。

如果 **length** 属性被赋予了一个比原先值大的数值，那么数组就被扩展，且所有新建元素都被赋值为 [undefined](#)。

下面这个例子说明了 **length** 属性的用法：

```
function LengthDemo() {  
    var a;  
    a = new Array(0,1,2,3,4);  
    return(a.length);  
}
```

要求

[版本 2](#)(See 2.64)

请参阅

[length 属性 \(Function\)](#)(See 1.2.10.16) | [length 属性 \(String\)](#)(See 1.2.10.17)

应用于： [Array 对象](#)(See 1.2.8.2)

1.2.10.16 length 属性 (Function)

返回函数定义的参数数目。

functionName.length

所必须的 *functionName* 参数是被讨论的函数的名称。

说明

函数的 **length** 属性是在创建函数实例时由 Scripting 引擎初始化的，值是函数定义中参数的数目。

如果调用函数时参数的数目和它的 **length** 属性值不同，那么这时发生的情况将依赖于函数本身。

下面的例子说明了 **length** 属性的用法：

```
function ArgTest(a, b) {  
    var i, s = "The ArgTest function  
expected ";  
    var numargs =  
ArgTest.arguments.length;  
    var expargs = ArgTest.length;  
    if (expargs < 2)  
        s += expargs + " argument. ";  
    else  
        s += expargs + " arguments. ";  
    if (numargs < 2)  
        s += numargs + " was passed.";  
    else  
        s += numargs + " were passed.";  
    return(s);  
}
```

要求

[版本 2](#)(See 2.64)

请参阅

[arguments 属性](#)(See 1.2.10.2) | [length 属性 \(Array\)](#)(See 1.2.10.15) | [length 属性 \(String\)](#)(See 1.2.10.17)

应用于: [Function 对象](#)(See 1.2.8.7)

1.2.10.17 length 属性 (String)

返回 **String** 对象的长度。

```
strVariable.length  
"String Literal".length
```

说明

length 属性包含一个整数,用来指出 **String** 对象中的字符数。**String** 对象中的最后一个字符的索引为 **length - 1**。

要求

[版本 1](#)(See 2.64)

请参阅

[length 属性 \(Array\)](#) (See 1.2.10.15) | [length 属性 \(Function\)](#) (See 1.2.10.16) | [String 对象的方法](#)(See 1.2.8.14) | [String 对象的属性](#)(See 1.2.8.14)

应用于: [String 对象](#)(See 1.2.8.14)

1.2.10.18 LN10 属性

返回 10 的自然对数。

```
numVar = Math.LN10
```

说明

LN10 属性约等于 2.302。

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象属性](#)(See 1.2.8.9)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.10.19 LN2 属性

返回 2 的自然对数。

```
numVar = Math.LN2
```

语法

LN2 属性约等于 0.693。

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象属性](#)(See 1.2.8.9)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.10.20 LOG10E 属性

返回以 10 为底 e （自然对数的底）的对数。

```
varName = Math.LOG10E
```

说明

LOG10E 属性，常数，约等于 0.434。

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象属性](#)(See 1.2.8.9)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.10.21 LOG2E 属性

返回以 2 为底 e （自然对数的底）的对数。

```
varName = Math.LOG2E
```

说明

LOG2E 属性，常数，约等于 1.442。

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象属性](#)(See 1.2.8.9)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.10.22 MAX_VALUE 属性

返回 JScript 能表达的最大的数。约等于 1.79E+308。

`number.MAX_VALUE`

`number` 参数是 **Number** 对象。

说明

在能够访问 **MAX_VALUE** 属性前不必创建 **Number** 对象。

要求

[版本 2](#)(See 2.64)

请参阅

[MIN_VALUE 属性](#) (See 1.2.10.24) | [NaN 属性](#) (See 1.2.10.27) | [NEGATIVE_INFINITY 属性](#) (See 1.2.10.29) | [POSITIVE_INFINITY 属性](#) (See 1.2.10.32) | [toString 方法](#) (See 1.2.7.5.15)

应用于: [Number 对象](#)(See 1.2.8.10)

1.2.10.23 message 属性

返回一个错误消息字符串。

`errorObj.message`

参数

`errorObj`

必选项。**Error** 对象。

说明

message 属性是一个字符串，其中包含显示给用户的出错消息。它包含与 **description** 属性相同的信息。

示例

以下示例将导致 **TypeError** 异常，并显示错误名称及其消息。

```
try {  
    // 'null' 不是有效的对象
```

```
    null.doSomething();  
}  
  
catch(e) {  
    print(e.name + ": " + e.message);  
    print(e.number + ": " + e.description);  
}
```

要求

[版本 5](#)(See 2.64)

请参阅

[description 属性](#)(See 1.2.10.5) | [name 属性](#)(See 1.4.3.23)

应用于: [Error 对象](#)(See 1.2.8.6)

1.2.10.24 MIN_VALUE 属性

返回 JScript 中能够表示的最接近零的数。约等于 2.22E-308。

`number.MIN_VALUE`

`number` 参数是 **Number** 对象。

说明

在能够访问 **MIN_VALUE** 属性前不必创建 **Number** 对象。

要求

[版本 2](#)(See 2.64)

请参阅

[MAX_VALUE 属性](#) (See 1.2.10.22) | [NaN 属性](#) (See 1.2.10.27) | [NEGATIVE_INFINITY 属性](#) (See 1.2.10.29) | [POSITIVE_INFINITY 属性](#) (See 1.2.10.32) | [toString 方法](#) (See 1.2.7.5.15)

应用于: [Number 对象](#)(See 1.2.8.10)

1.2.10.25 multiline 属性

返回 Boolean 值, 指出正则表达式使用的 `multiline` 标志(**m**)的状态。默认值为 **false**。只读。

`rgExp.multiline`

必选项 *rgExp* 参数为 **RegExp** 对象。

说明

如果正则表达式设置了 **multiline** 标志, 那么 **multiline** 属性返回 **true**, 否则返回 **false**。如果创建正则表达式对象时使用了 **m** 标志, 那么 **multiline** 属性就是 **true**。如果 **multiline** 为 **false**, 那么 "^" 匹配字符串的开始位置, 而 "\$" 匹配字符串的结束位置。如果 **multiline** 为 **true**, 那么 "^" 匹配字符串开始位置以及 "\n" 或 "\r" 之后的位置, 而 "\$" 匹配字符串结束位置以及 "\n" 或 "\r" 之前的位置。

示例

以下示例演示了 **multiline** 属性的特征。如果将 "m" 传递给下面所示的函数, 单词 "while" 将被替换为 "and"。这是因为设置了 **multiline** 标志且 "while" 出现在换行字符的下一行的开始位置。**multiline** 标志允许在多行的字符串中进行查找。本函数返回一个字符串以及一个表, 表中显示了允许使用的正则表达式标志 (**g**、**i** 和 **m**) 的状态。它还返回经过所有替换操作的字符串。

```
function RegExpPropDemo(flag) {

    if (flag.match(/[^gim]/))                // 检
查标志的有效性。

        return("Flag    specified    is    not
valid");

    var r, re, s                                // 声明变
量。

    var ss = "The man hit the ball with
the bat.";

    ss += "\nwhile the fielder caught the
ball with the glove.";

    re = new RegExp("^while",flag);           //
指定要查找的样式。

    r = ss.replace(re, "and");                // 用
"a" 替换 "the"。

    s  = "Regular    Expression    property
```

```

values:\n\n"

    s      +=      "global          ignoreCase
multiline\n"

    if (re.global)                                // 测试
global  标志。

        s += " True      ";
    else
        s += "False      ";
    if (re.ignoreCase)                            // 测试
ignoreCase 标志。

        s += " True  ";
    else
        s += "False  ";
    if (re.multiline)                            // 测试
multiline 标志。

        s += "      True      ";
    else
        s += "      False  ";
    s      +=      "\n\nThe      resulting      string
is:\n\n" + r;

    return(s);                                    // 返回替换
的字符串。
}

```

要求

[版本 5.5](#)(See 2.64)

请参阅

[global 属性](#)(See 1.2.10.7) | [ignoreCase 属性](#)(See 1.2.10.9) | [正则表达式语法](#)(See 1.5.4)

应用于: [RegExp 对象](#)(See 1.2.8.12)

1.2.10.26 name 属性

返回一个错误的名称。

`errorObj.name`

参数

`errorObj`

必选项。**Error** 对象。

说明

name 属性返回错误名称或异常类型。发生运行时错误时, 该错误的名称属性被设置为下列内在的异常类型之一:

异常类型	意义
ConversionError	当试图将一个对象转换为其不能转换的某种类型时, 产生本错误。
RangeError	当函数的某个给出的参数不在允许范围时, 发生本错误。例如, 当试图建立的 Array 对象的长度不是有效的正整数时就会发生本错误。
ReferenceError	当检测到无效的引用时, 发生本错误。例如, 如果所想要使用的引用为 null 时就会发生本错误。
RegExpError	当正则表达式产生编译错误时, 发生本错误。然而, 只要该正则表达式经过了编译, 就不会产生本错误。例如, 如果使用无效语法, 或标志不为 i 、 g 、 m , 或者同一标志出现多次的样式声明正则表达式时, 就会发生本错误。
SyntaxError	当对错误语法的源文本进行解析时, 发生本错误。例如, 调用 eval 函数时其参数不是有效的程序文本, 就会发生本错误。
TypeError	只要算子的实际类型与所期望的类型不符合, 就会发生本错误。例如, 如果进行函数调用的不是对象或者不支持该调用, 发生本错误。
URIError	当检测到非法的统一资源标识符 (URI) 时发生本错误。例如, 在被编码或解码的字符串中发现非法字符, 就会发生本错误。

示例

以下示例将导致 `TypeError` 异常，并显示该错误的名称及其消息。

```
try {  
    // 'null' 不是有效的对象  
    null.doSomething();  
}  
  
catch(e) {  
    print(e.name + ": " + e.message);  
    print(e.number + ": " + e.description);  
}
```

要求

[版本 5.5](#)(See 2.64)

请参阅

[description 属性](#) (See 1.2.10.5) | [message 属性](#) (See 1.2.10.23) | [number 属性](#) (See 1.2.10.30)

应用于: [Error 对象](#) (See 1.2.8.6)

1.2.10.27 NaN 属性

表示算术表达式返回非数字值的特殊值。

`number.NaN`

`number` 参数是 **Number** 对象。

说明

在能够访问 **NaN** 属性前不必创建 **Number** 对象。

NaN 不与任何值相等，包括其本身。要检测值是否为 **NaN**，请使用 **isNaN** 函数。

要求

[版本 2](#)(See 2.64)

请参阅

[isNaN 方法](#) (See 1.2.7.2.29) | [MAX_VALUE 属性](#) (See 1.2.10.22) | [MIN_VALUE 属性](#) (See 1.2.10.24) | [NEGATIVE_INFINITY 属性](#) (See 1.2.10.29) | [POSITIVE_INFINITY 属性](#) (See 1.2.10.32) | [toString 方法](#) (See 1.2.7.5.15)

应用于: [Number 对象](#) (See 1.2.8.10)

1.2.10.28 NaN 属性 (Global)

返回特殊值 **NaN**，表示表达式不是数字。

NaN

说明

NaN 属性（不是数字）是 **Global** 对象的成员，在 Scripting 引擎初始化时变为可用。

要求

[版本 3](#)(See 2.64)

请参阅

[isNaN 方法](#)(See 1.2.7.2.29)

应用于：[Global 对象](#)(See 1.2.8.8)

1.2.10.29 NEGATIVE_INFINITY 属性

返回比 JScript 能够表示的最小负数（**-Number.MAX_VALUE**）更小的值。

`number.NEGATIVE_INFINITY`

`number` 参数是 **Number** 对象。

说明

在能够访问 **NEGATIVE_INFINITY** 属性前不必创建 **Number** 对象。
JScript 将 **NEGATIVE_INFINITY** 值显示为 `-infinity`。该值数学上的作用与负无穷相同。

要求

[版本 2](#)(See 2.64)

请参阅

[MAX_VALUE 属性](#)(See 1.2.10.22) | [MIN_VALUE 属性](#)(See 1.2.10.24) | [NaN 属性](#)(See 1.2.10.27) | [POSITIVE_INFINITY 属性](#)(See 1.2.10.32) | [toString 方法](#)(See 1.2.7.5.15)

应用于：[Number 对象](#)(See 1.2.8.10)

1.2.10.30 number 属性

返回或设置与特定错误相联系的数字值。**Error** 对象的默认是 **number**。

`object.number [= errorNumber]`

参数

object

任意 **Error** 对象的实例。

errorNumber

表示一个错误的整数。

说明

错误号是一个 32 位的值。高 16 位字是设施代码，而低字才是真正的错误代码。下面的例子举例说明了 **number** 属性的用法：

```
try
    x = y    // 产生错误。
catch (var e) {    // 创建全局变量 e。
    document.write(e)    // 打印 "[object
Error]".
    document.write(e.number>>16    &
0x1FFF)    // 打印设施代码 10。
    document.write(e.number & 0xFFFF)    //
打印错误代码 5009。
    document.write(e.description)    // 打印
"'y' is undefined".
}
```

要求

[版本 5](#)(See 2.64)

请参阅

[description 属性](#)(See 1.2.10.5) | [message 属性](#)(See 1.2.10.23) | [name 属性](#)(See 1.4.3.23)

应用于： [Error 对象](#)(See 1.2.8.6)

1.2.10.31 PI 属性

返回圆的周长与其直径的比值，约等于 3.141592653589793。

numVar = **Math.PI**

语法

PI 属性，常数，约等于 3.14159。

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象属性](#)(See 1.2.8.9)

应用于： [Math 对象](#)(See 1.2.8.9)

1.2.10.32 POSITIVE_INFINITY 属性

返回比在 JScript 中能够表示的最大的数 (**Number.MAX_VALUE**) 更大的值。

Number.POSITIVE_INFINITY

number 参数是 **Number** 对象。

说明

在能够访问 **POSITIVE_INFINITY** 属性前不必创建 **Number** 对象。

JScript 将 **POSITIVE_INFINITY** 值显示为 *infinity*。该值数学上的作用与正无穷相同。

要求

[版本 2](#)(See 2.64)

请参阅

[MAX_VALUE 属性](#)(See 1.2.10.22) | [MIN_VALUE 属性](#)(See 1.2.10.24) | [NaN 属性](#)(See 1.2.10.27) | [NEGATIVE_INFINITY 属性](#)(See 1.2.10.29) | [toString 方法](#)(See 1.2.7.5.15)

应用于： [Number 对象](#)(See 1.2.8.10)

1.2.10.33 propertyIsEnumerable 属性

返回 **Boolean** 值，指出所指定的属性是否为一个对象的一部分以及该属性是否是可列举的。

object.propertyIsEnumerable (*propertyName*)

参数

object

必选项。一个对象。

proName

必选项。一个属性名称的字符串值。

说明

如果 *proName* 存在于 *object* 中且可以使用一个 **For...In** 循环穷举出来，那么 **propertyIsEnumerable** 属性返回 **true**。如果 *object* 不具有所指定的属性或者所指定的属性不是可列举的，那么 **propertyIsEnumerable** 属性返回 **false**。典型地，预定义的属性不是可列举的，而用户定义的属性总是可列举的。

propertyIsEnumerable 属性不考虑原型链中的对象。

示例

```
function testIsEnumerable() {  
    var a = new Array("apple", "banana",  
"cactus");  
  
    return (a.propertyIsEnumerable(1));  
}
```

要求

[版本 5.5](#)(See 2.64)

请参阅

应用于: [Object 对象](#)(See 1.2.8.11)

1.2.10.34 prototype 属性

返回对象类型原型的引用。

objectName.**prototype**

objectName 参数是对象的名称。

说明

用 **prototype** 属性提供对象的类的一组基本功能。对象的新实例“继承”赋予该对象原型的操作。

例如，要为 **Array** 对象添加返回数组中最大元素值的方法。要完成这一点，声

明该函数，将它加入 **Array.prototype**，并使用它。

```
function array_max( ){
    var i, max = this[0];
    for (i = 1; i < this.length; i++)
    {
        if (max < this[i])
            max = this[i];
    }
    return max;
}
```

```
Array.prototype.max = array_max;
var x = new Array(1, 2, 3, 4, 5, 6);
var y = x.max( );
```

该代码执行后，y 保存数组 x 中的最大值，或说 6。

所有 [JScript 固有对象](#) 都有只读的 **prototype** 属性。可以象该例中那样为原型添加功能，但该对象不能被赋予不同的原型。然而，[用户定义的对象](#) 可以被赋给新的原型。

本语言参考中每个内部对象的方法和属性列表指出哪些是对象原型的部分，哪些不是。

要求

[版本 2](#)(See 2.64)

请参阅

[constructor 属性](#)(See 1.2.10.4)

应用于：[Array 对象](#)(See 1.2.8.2) | [Boolean 对象](#)(See 1.2.8.3) | [Date 对象](#)(See 1.2.8.4) | [Function 对象](#)(See 1.2.8.7) | [Number 对象](#)(See 1.2.8.10) | [Object 对象](#)(See 1.2.8.11) | [String 对象](#)(See 1.2.8.14)

1.2.10.35 source 属性

返回正则表达式模式的文本的复本。只读。

rgExp.source

rgExp 参数是正则对象。它可以是变量名或文字。

下面的例子举例说明了 **source** 属性的用法：

```
function SourceDemo(re, s) {
    var s1;

    // 测试字符串中是否存在正则表达式。

    if (re.test(s))
        s1 = " contains ";
    else
        s1 = " does not contain ";

    // 获得正则表达式自己的文本。

    return(s + s1 + re.source);
}
```

要求

[版本 3](#)(See 2.64)

请参阅

[正则表达式对象的方法](#)(See 1.2.8.13) | [正则表达式对象的属性](#)(See 1.2.8.13) | [正则表达式的语法](#)(See 1.5.4)

应用于：[正则表达式对象](#)(See 1.2.8.13)

1.2.10.36 SQRT1_2 属性

返回 0.5 的平方根，或说 2 的平方根分之一。

numVar = **Math.SQRT1_2**

说明

SQRT1_2 属性，常数，约等于 0.707.

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象属性](#)(See 1.2.8.9) | [sqrt 方法](#)(See 1.2.7.4.26) | [SQRT2 属性](#)(See 1.2.10.37)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.10.37 SQRT2 属性

返回 2 的平方根。

```
numVar = Math.SQRT2
```

语法

SQRT2 属性, 常数, 约等于 1.414.

要求

[版本 1](#)(See 2.64)

请参阅

[Math 对象属性](#)(See 1.2.8.9) | [sqrt 方法](#)(See 1.2.7.4.26) | [SQRT1 2 属性](#)(See 1.2.10.36)

应用于: [Math 对象](#)(See 1.2.8.9)

1.2.10.38 undefined 属性

返回 **undefined** 的一个初始值。

```
undefined
```

说明

undefined 属性是 **Global** 对象的一个成员, 该属性在脚本引擎初始化后可用。如果已声明了一个变量但还没有初始化, 那么该变量的值就是 **undefined**。

如果还没有声明变量, 那么就不能将其与 **undefined** 进行比较, 但是可以将该变量的类型与字符串 "undefined" 进行比较。

当显式地测试变量或将变量设置为 **undefined** 时, **undefined** 属性是很有用的。

示例

```
var declared; // 声明变
```

量。

```
if (declared == undefined) // 测
```

试变量。

```
    document.write("declared has not been  
given a value.");
```

```
if (typeof(notDeclared) == "undefined")
    document.write("notDeclared has not
been defined.");
```

要求

[版本 5.5](#)(See 2.64)

请参阅

应用于: [Global Object](#)(See 1.2.8.8)

1.2.11 JScript 语句

描述	语言要素
终止当前循环，或者如果与一个 <i>label</i> 语句关联，则终止相关联的语句。	break 语句 (See 1.2.11.4)
包含在 try 语句块中的代码发生错误时执行的语句。	catch 语句 (See 1.2.11.17)
激活条件编译支持。	@cc on 语句 (See 1.2.11.1)
使单行注释被 JScript 语法分析器忽略。	// （单行注释语句） (See 1.2.11.1)
使多行注释被 JScript 语法分析器忽略。	/* */ （多行注释语句） (See 1.2.11.1)
停止循环的当前迭代，并开始一次新的迭代。	continue 语句 (See 1.2.11.6)
先执行一次语句块，然后重复执行该循环，直至条件表达式的值为 false 。	do...while 语句 (See 1.2.11.7)
只要指定的条件为 true ，就一直执行语句块。	for 语句 (See 1.2.11.8)
对应于对象或数组中的每个元素执行一个或多个语句。	for...in 语句 (See 1.2.11.9)
声明一个新的函数。	function 语句 (See 1.2.11.10)
根据表达式的值，有条件地执行一组语句。	@if 语句 (See 1.2.11.2)
根据表达式的值，有条件地执行一组语句。	if..else 语句 (See 1.2.11.11)
给语句提供一个标识符。	Labeled 语句 (See 1.2.11.12)
从当前函数退出并从该函数返回一个值。	return 语句 (See 1.2.11.13)
创建用于条件编译语句的变量。	@set 语句 (See 1.2.11.3)
当指定的表达式的值与某个标签匹配时，即执行相应的一个或多个语句。	switch 语句 (See 1.2.11.14)
对当前对象的引用。	this 语句 (See 1.2.11.15)
产生一个可由 try...catch 语句处理的错误条件。	throw 语句 (See 1.2.11.16)
实现 JScript 的错误处理。	try 语句 (See 1.2.11.17)

声明一个变量。	var 语句 (See 1.2.11.18)
执行语句直至给定的条件为 false 。	while 语句 (See 1.2.11.19)
确定一个语句的默认对象。	with 语句 (See 1.2.11.20)

1.2.11.1 @cc_on 语句

激活条件编译支持。

```
@cc_on
```

说明

@cc_on 语句激活 scripting 引擎中的条件编译。

极力推荐在注释中使用 **@cc_on** 语句，以使不支持条件编译的浏览器也接受此脚本为有效语法：

```
/*@cc_on*/
```

...

（脚本的剩余部分）

作为选择，在注释外的一个 **@if** 或 **@set** 语句也会激活条件编译。

要求

[版本 3](#)(See 2.64)

请参阅

[条件编译](#)(See 1.1.2.8) | [条件编译变量](#)(See 1.1.2.9) | [@if 语句](#)(See 1.2.11.2) | [@set 语句](#)(See 1.2.11.3)

1.2.11.2 @if 语句

根据表达式的值，有条件地执行一组语句。

```
@if (
    condition1
)
    text1
[@elif (
    condition2
)
    text2]
[@else
    text3]
@end
```

参数

`condition1, condition2`

可选项。可以强制转换为 [Boolean 表达式](#) 的表达式。

`text1`

可选项。`condition1` 是 **true** 时要解析的文本。

`text2`

可选项。`condition1` 是 **false** 且 `condition2` 是 **true** 时要解析的文本。

`text3`

可选项。`condition1` 和 `condition2` 都是 **false** 时要解析的文本。

说明

在书写 **@if** 语句时，不必将每个子句放到不同的行。可以使用多个 **@elif** 子句。但是，所有 **@elif** 子句必须在 **@else** 子句之前出现。通常使用 **@if** 语句来决定应当使用若干选项中的哪个选项来进行文本输出。例如：

```
alert(@if (@_win32) "using Windows NT or  
Windows 95" @else "using Windows 3.1"  
@end)
```

要求

[版本 3](#)(See 2.64)

请参阅

[条件编译](#)(See 1.1.2.8) | [条件编译变量](#)(See 1.1.2.9) | [@cc_on 语句](#)(See 1.2.11.1) | [@set 语句](#)(See 1.2.11.3)

1.2.11.3 @set 语句

用条件编译语句来创建变量。

```
@set @varname = term
```

参数

varname

必选项。有效的 JScript 变量名。必须总在前面放置一个 “@” 字符。

term

必选项。零或多个一元运算符，后跟一个常数、条件编译变量、或圆括号括起来的表达式。

说明

条件编译支持数字和 Boolean 变量。不支持字符串。用 **@set** 创建的变量通常在条件编译语句中使用，但也可以在 JScript 代码的任何地方使用。变量声明的例子看起来如下：

```
@set @myvar1 = 12
```

```
@set @myvar2 = (@myvar1 * 20)
```

```
@set @myvar3 = @_jscript_version
```

在园括号括起来的表达式中支持下面的运算符：

- !~
- * / %
- + -
- << >> >>>
- < <= > >=
- == != === !==
- & ^ |
- && ||

如果在变量被定义前使用了这个变量，则它的值是 **NaN**。**NaN** 可以用 **@if** 语句来检查：

```
@if (@newVar != @newVar)
```

...

这样做能成功是因为 **NaN** 是不等于它自己的唯一值。

要求

[版本 3](#)(See 2.64)

请参阅

[条件编译](#)(See 1.1.2.8) | [条件编译变量](#)(See 1.1.2.9) | [@cc on 语句](#)(See 1.2.11.1) | [@if 语句](#)(See 1.2.11.2)

1.2.11.4 break 语句

中断当前循环，或和 *label* 一起使用，中断相关联的语句。

break [*label*];

可选的 *label* 参数指定断点处语句的标签。

说明

通常在 **switch** 语句和 **while**、**for**、**for...in**、或 **do...while** 循环中使用 **break** 语句。最一般的是在 **switch** 语句中使用 *label* 参数，但它可在任何语句中使用，无论是简单语句还是[复合](#)语句。

执行 **break** 语句会退出当前循环或语句，并开始脚本执行紧接着的语句。

示例

下面的示例说明了 **break** 语句的用法。

```
function BreakTest(breakpoint) {  
    var i = 0;  
    while (i < 100)  
    {  
        if (i == breakpoint)  
            break;  
        i++;  
    }  
    return(i);  
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[continue 语句](#)(See 1.2.11.6) | [do...while 语句](#)(See 1.2.11.7) | [for 语句](#)(See 1.2.11.8) | [for...in 语句](#)(See 1.2.11.9) | [Labeled 语句](#)(See 1.2.11.12) | [while 语句](#)(See 1.2.11.19)

1.2.11.5 Comment 语句

使注解部分被 JScript 解析器忽略。

语法 1

单行注释:

```
// comment
```

语法 2

多行注释:

```
/*
```

```
comment
```

```
*/
```

comment 参数是想要包含到脚本中的注释。

语法 3

```
//@CondStatement
```

语法 4

```
/*@
```

```
condStatement
```

```
@*/
```

condStatement 参数是条件编译被激活后使用的条件编译代码。如果使用的是语法 3, 则在 "/" 和 "@" 字符间可以没有空格。

说明

用注释来使 JScript 解析器不读取脚本的某些部分。可以使用注释来在程序中包括解释性说明。

如果使用的是语法 1, 则解析器忽略在注释标记和行尾间的所有文本。如果使用的是语法 2, 则它忽略开始标记和结束标记之间的所有文本。

语法 3 和 4 用来支持条件编译, 以保证与不支持那个特性的浏览器兼容。这些浏览器把这些形式的注释分别当成语法 1 和 2 对待。

示例

下面示例说明了 **comment** 语句的最普通的用法:

```
function myfunction(arg1, arg2) {
```

```
    /* 这是多行注释，  
    可以根据需要跨越多行。 */  
  
    var r;  
  
    // 这是单行注释。  
  
    r = arg1 + arg2; // 求两个参数的和。  
  
    return(r);  
  
}
```

要求

[版本 1](#)(See 2.64)

1.2.11.6 continue 语句

停止循环的当前迭代，并开始新的迭代。

continue [*label*];

可选的 *label* 参数指定 **continue** 应用于哪条语句。

说明

只能在 **while**、**do...while**、**for**、或 **for...in** 循环内使用 **continue** 语句。执行 **continue** 语句会停止当前循环的迭代，并从循环的开始处继续程序流程。这样对不同的循环类型有如下影响：

- **while** 和 **do...while** 循环检查它们的条件，如果条件为 **true**，则再次执行循环。
- **for** 循环执行它们的递增表达式，如果测试表达式为 **true**，则再次执行循环。
- **for...in** 循环继续进行指定变量的下一个字段，并再次执行循环。

示例

下面示例说明了 **continue** 语句的用法：

```
function skip5() {  
    var s = "", i=0;  
    while (i < 10)  
    {
```

```

        i++;

        // 跳过 5

        if (i==5)
        {
            continue;
        }

        s += i;
    }

    return (s);
}

```

要求

[版本 1](#)(See 2.64)

请参阅

[break 语句](#)(See 1.2.11.4) | [do...while 语句](#)(See 1.2.11.7) | [for 语句](#)(See 1.2.11.8) | [for...in 语句](#)(See 1.2.11.9) | [Labeled 语句](#)(See 1.2.11.12) | [while 语句](#)(See 1.2.11.19)

1.2.11.7 do...while 语句

第一次执行一个语句块，然后重复循环的执行该语句块，直到条件表达式等于 **false**。

do

statement

while (expression) ;

参数

statement

可选项。*expression* 是 **true** 时要执行的语句。可以是 [复合语句](#)。

expression

可选项。一个可以强制转换为 Boolean **true** 或 **false** 的 [表达式](#)。如果 *expression* 是 **true**，则再执行一次循环。如果 *expression* 是 **false**，则结束循环。

说明

在循环的第一次重复执行完成前，不检查 *expression* 的值，保证至少执行循环一次。此后，循环每成功重复一次后都要检查表达式。

示例

下面的示例阐明了如何使用 **do...while** 语句来重复 **Drives** 集合。

```
function GetDriveList() {  
    var fso, s, n, e, x;  
    fso = new  
ActiveXObject("Scripting.FileSystemObject");  
    e = new Enumerator(fso.Drives);  
    s = "";  
    do  
    {  
        x = e.item();  
        s = s + x.DriveLetter;  
        s += " - ";  
        if (x.DriveType == 3)  
            n = x.ShareName;  
        else if (x.IsReady)  
            n = x.VolumeName;  
        else
```

```

        n = "[Drive not ready]";
        s += n + "<br>";
        e.moveToNext();
    }

    while (!e.atEnd());
    return(s);
}

```

要求

[版本 3](#) (See 2.64)

请参阅

[break 语句](#) (See 1.2.11.4) | [continue 语句](#) (See 1.2.11.6) | [for 语句](#) (See 1.2.11.8) | [for...in 语句](#) (See 1.2.11.9) | [while 语句](#) (See 1.2.11.19) | [Labeled 语句](#) (See 1.2.11.12)

1.2.11.8 for 语句

只要指定条件为 **true** 都执行语句块。

```

for (initialization; test; increment)
    statements

```

参数

initialization

必选项。一个 [表达式](#)。该表达式只在执行循环前被执行一次。

test

必选项。一个 [Boolean 表达式](#)。如果 *test* 是 **true**，则 *statement* 被执行。如果 *test* 是 **false**，则循环结束。

increment

必选项。一个表达式。在每次经过循环的最后执行该递增表达式。

statements

可选项。 *test* 是 **true** 时，要执行的一个或多个语句。可以是[复合语句](#)。

说明

循环要执行确定的次数，通常使用 **for** 循环。

示例

下面的例子示范了一个 **for** 循环。

*/** 在开始时 *i* 被设为 0，并在每次重复的最后被增加 1。

在循环重复前，

如果 *i* 不小于 10，则循环结束。 **/*

```
var myarray = new Array();  
for (i = 0; i < 10; i++) {  
    myarray[i] = i;  
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[for...in 语句](#)(See 1.2.11.9) | [while 语句](#)(See 1.2.11.19)

1.2.11.9 for...in 语句

对应于一个对象的每个，或一个数组的每个元素，执行一个或多个语句。

```
for (variable in [object | array])  
    statements
```

参数

variable

必选项。一个[变量](#)，它可以是 *object* 的任一属性或 *array* 的任一元素。

object, array

可选项。要在其上遍历的对象或数组。

statement

可选项。相对于 *object* 的每个属性或 *array* 的每个元素，都要被执行的一个或多个语句。可以是[复合语句](#)。

说明

在循环的每次迭代前，*variable* 被赋予 *object* 的下一个属性或 *array* 的下一个元素。然后可以在循环内的任一语句中使用它，就好像正在使用 *object* 的该属性或 *array* 的该元素一样。

当在一个对象上迭代时，没有办法决定或控制把对象的成员赋给 *variable* 的次序。在数组内将按元素的次序执行迭代，也就是，0、1、2、.....

示例

下面示例说明了 **for ... in** 语句的用法，它把一个对象用作一个联合数组：

```
function ForInDemo() {  
  
    // 创建某些变量。  
  
    var a, key, s = "";  
  
    // 初始化对象。  
  
    a = {"a" : "Athens" , "b" : "Belgrade",  
        "c" : "Cairo"}  
  
    // 迭代属性。  
  
    for (key in a) {  
        s += a[key] + "<br>";  
    }  
  
    return(s);  
}
```

要求

[版本 5](#)(See 2.64)

注意 使用 **enumerator** 对象来迭代集合的成员。

请参阅

[for 语句](#)(See 1.2.11.8) | [while 语句](#)(See 1.2.11.19)

1.2.11.10 function 语句

声明一个新的函数。

```
function functionname([arg1 [, arg2 [, ...[, argN]]]])  
{  
    statements  
}
```

参数

functionname

必选项。函数名。

arg1...argN

可选项。函数理解的参数列表、各个参数之间用逗号分开。

statements

可选项。一个或多个 JScript 语句。

说明

使用 **function** 语句来声明一个稍后要使用的函数。在脚本的其他地方调用该函数前，*statements* 中包含的代码不被执行。

示例

下面示例说明了 **function** 语句的用法。

```
function myfunction(arg1, arg2){  
    var r;  
    r = arg1 * arg2;  
    return(r);  
}
```

注意 在调用函数时，确保总是包括了圆括号以及任何需要的参数。不带圆括号调用函数将导致返回函数的文本，而不是函数的结果。

要求

[版本 1](#)(See 2.64)

请参阅

[new 运算符](#)(See 1.2.9.32)

1.2.11.11 if...else 语句

根据一个表达式的值，有条件地执行一组语句。

```
if (condition)  
    statement1  
[else  
    statement2]
```

参数

condition

必选项。一个 [Boolean 表达式](#)。如果 *condition* 是 [null](#) 或 [undefined](#)，则 *condition* 被当作 **false**。

statement1

可选项。*condition* 是 **true** 时要执行的语句。可以是[复合语句](#)。

statement2

可选项。*condition* 是 **false** 时要被执行的语句。可以是复合语句。

说明

把 *statement1* 和 *statement2* 括在大括号 ({}) 内通常是一个好习惯，这样就更清楚，并可以避免无意中造成错误。

示例

在下面示例中，您可能打算把 **else** 和第一个 **if** 语句匹配，但它实际上却是和第二个相匹配的。

```
if (x == 5)  
    if (y == 6)  
        z = 17;
```

```
else
```

```
    z = 20;
```

按如下方法改写代码可以消除任何含混不清：

```
if (x == 5)
```

```
{
```

```
    if (y == 6)
```

```
        z = 17;
```

```
}
```

```
else
```

```
    z = 20;
```

同样，如果希望添加一个语句到 *statement1*，同时不使用大括号，则可能会产生错误：

```
if (x == 5)
```

```
    z = 7;
```

```
    q = 42;
```

```
else
```

```
    z = 19;
```

在这种情况下，有一个语法错误，因为在 **if** 和 **else** 语句之间有多条语句。在 **if** 和 **else** 之间的语句需要大括号。

要求

[版本 1](#)(See 2.64)

请参阅

[条件运算符 \(?:\)](#)(See 1.2.9.17)

1.2.11.12 Labeled 语句

为语句提供一个标识符。

label :

statements

参数

label

必选项。在引用有标签的语句时使用的一个唯一的标识符。

statement

可选项。与 *label* 相关联的一个或多个语句。可以是[复合语句](#)。

说明

标签由 **break** 和 **continue** 语句使用，用来指示 **break** 和 **continue** 应用于哪个语句。

示例

在下面的语句中，**continue** 语句使用一个 **labeled** 语句来创建一个数组，在该数组中，每行的第三列包含一个未定义的值：

```
function labelDemo() {  
    var a = new Array();  
    var i, j, s = "", s1 = "";  
    Outer:  
        for (i = 0; i < 5; i++)  
        {  
            Inner:  
                for (j = 0; j < 5; j++)  
                {  
                    if (j == 2)  
                        continue Inner;  
                    else
```

```
        a[i,j] = j + 1;
    }

}

for (i = 0; i < 5; i++)
{
    s = ""
    for (j = 0; j < 5; j++)
    {
        s += a[i,j];
    }
    s1 += s + "\n";
}

return(s1)

}
```

要求

[版本 3](#)(See 2.64)

请参阅

[break 语句](#)(See 1.2.11.4) | [continue 语句](#)(See 1.2.11.6)

1.2.11.13 return 语句

从当前函数退出，并从那个函数返回一个值。

return [() [*expression*] []] ;

可选项 *expression* 参数是要从函数返回的值。如果省略，则该函数不返回值。

说明

用 **return** 语句来终止一个函数的执行，并返回 *expression* 的值。如果 *expression* 被省略，或在函数内没有 **return** 语句被执行，则把值 [undefined](#) 赋给调用当前函

数的[表达式](#)。

示例

下面示例说明了 **return** 语句的用法。

```
function myfunction(arg1, arg2) {  
    var r;  
    r = arg1 * arg2;  
    return(r);  
}
```

要求

[版本 1](#)(See 2.64)

请参阅

[function 语句](#)(See 1.2.11.10)

1.2.11.14 switch 语句

当指定的表达式的值与某个标签匹配时，即执行相应的一个或多个语句。

```
switch (expression) {  
    case label :  
        statementlist  
    case label :  
        statementlist  
    ...  
    default :  
        statementlist  
}
```

参数

expression

要求值的表达式。

label

根据 *expression* 来匹配的标识符。如果 *label* `===` *expression*，则立即从冒号后的 *statementlist* 处开始执行，直到遇到一个可选的 **break** 语句，或到达 **switch** 语句的最后。

statementlist

要被执行的一个或多个语句。

说明

使用 **default** 子句来提供一个语句，该语句只在没有任何一个标签值与 *expression* 相匹配时才被执行。它可以出现在 **switch** 代码块内的任何地方。

可以指定零或多个 *label* 块。如果没有 *label* 和 *expression* 的值匹配，并且没有提供 **default** 情况，则不执行任何语句。

通过 **switch** 语句执行流程如下：

- 求 *expression* 的值并依次序查看 *label*，直到找到一个匹配。
- 如果 *label* 的值等于 *expression* 的值，则执行它相应的 *statementlist*。
继续执行，直到遇到一个 **break** 语句，或者 **switch** 语句结束。这意味着如果没有使用一个 **break** 语句，则多个 *label* 块被执行。
- 如果没有 *label* 等于 *expression* 的值，则跳转到 **default** 情况。如果没有 **default** 情况，则跳转到最后一步。
- 继续执行紧接 **switch** 代码块末尾的语句。

示例

下面示例测试一个对象的类型：

```
function MyObject() {  
    ...  
  
    switch (object.constructor) {  
        case Date:  
            ...  
        case Number:  
            ...  
    }  
}
```



```
    case String:
        ...
    case MyObject:
        ...
    default:
        ...
}
```

要求

[版本 3](#) (See 2.64)

请参阅

[break 语句](#) (See 1.2.11.4) | [if...else 语句](#) (See 1.2.11.11)

1.2.11.15 this 语句

指当前对象。

this.property

必选的 property 参数指的是对象的属性。

说明

this 关键字通常在对象的 [构造函数](#) 中使用，用来引用对象。

示例

在下面示例中，**this** 指的是新创建的 Car 对象，并给三个属性赋值。

```
function Car(color, make, model) {
    this.color = color;
    this.make = make;
    this.model = model;
}
```

对于 JScript 的客户版本，如果在其他所有对象的上下文之外使用 **this**，则它指的是 **window** 对象。

要求

[版本 1](#) (See 2.64)

请参阅

[new 运算符](#) (See 1.2.9.32)

1.2.11.16 throw 语句

产生一个能被 **try...catch...finally** 语句处理的错误情形。

throw *exception*

必选的 *exception* 参数可以是任何[表达式](#)。

说明

下面的例子根据传递进来的值扔出一个错误，然后举例说明那个错误是如何在 **try...catch...finally** 语句的层次中被处理的：

```
function TryCatchDemo(x) {
    try {
        try {
            if (x == 0)        // 估参数的值。

                throw "x equals zero";    // 扔出一个错误。
        }
        else
            throw "x does not equal
zero";    // 扔出一个不同的错误。
    }
    catch(e) {    // 在这儿处理 "x = 0" 的错误。
        if (e == "x equals zero")    // 检查错
误能否在这儿被处理。
            return (e + " handled
locally.");    // 返回对象错误消息。
        else    // 不能在这儿处理这个错误。

```

```

        throw e;    // 重新抛出该错误给下一个

    }    // 错误处理程序。
}

catch(e) {    // 在此处理其他错误。

    return(e    +    "    handled    higher
up.");    // 返回错误消息。
}
}

document.write(TryCatchDemo(0));
document.write(TryCatchDemo(1));

```

要求

[版本 5](#)(See 2.64)

请参阅

[try...catch 语句](#)(See 1.2.11.17)

1.2.11.17 try...catch...finally 语句

为 JScript 实现错误处理。

```

try {
    tryStatements}
catch(exception) {
    catchStatements}
finally {
    finallyStatements}

```

参数

tryStatement

必选项。可能发生错误的语句。

exception

可选项。任何[变量](#)名。exception 的初始化值是扔出的错误的值。

catchStatement

可选项。处理在相关联的 *tryStatement* 中发生的错误的语句。

finallyStatements

可选项。在所有其他过程发生之后无条件执行的语句。

说明

try...catch...finally 语句提供了一种方法来处理可能发生在给定代码块中的某些或全部错误，同时仍保持代码的运行。如果发生了程序员没有处理的错误，JScript 只给用户提供它的普通错误消息，就好象没有错误处理一样。

tryStatements 参数包含可能发生错误的代码，而 *catchStatement* 则包含处理任何发生了的错误的代码。如果在 *tryStatements* 中发生了一个错误，则程序控制被传给 *catchStatements* 来处理。*exception* 的初始化值是发生在 *tryStatements* 中的错误的值。如果错误不发生，则不执行 *catchStatements*。

如果在与发生错误的 *tryStatements* 相关联的 *catchStatements* 中不能处理该错误，则使用 **throw** 语句来传播、或重新扔出这个错误给更高级的错误处理程序。

在执行完 *tryStatements* 中的语句，并在 *catchStatements* 的所有错误处理发生之后，可无条件执行 *finallyStatements* 中的语句。

请注意，即使在 **try** 或 **catch** 块中返回一个语句，或在 **catch** 块重新扔出一个错误，仍然会执行 *finallyStatements* 编码。一般将确保 *finallyStatements* 的运行，除非存在未处理的错误。（例如，在 **catch** 块中发生运行时错误。）。

示例

下面的例子阐明了 JScript 特例处理是如何进行的。

```
try {  
    print("Outer try running..");  
    try {  
        print("Nested try running...");  
        throw "an error";  
    }  
    catch(e) {  
        print("Nested catch caught " + e);  
        throw e + " re-thrown";  
    }  
}
```

```
    }  
    finally {  
        print("Nested        finally        is  
running...");  
    }  
}  
catch(e) {  
    print("Outer catch caught " + e);  
}  
finally {  
    print("Outer finally running");  
}  
//  Windows Script Host  作出该修改从而得出  
WScript.Echo(s)  
function print(s){  
    document.write(s);  
}
```

将得出以下结果:

Outer try running..

Nested try running...

Nested catch caught an error

Nested finally is running...

Outer catch caught an error re-thrown

Outer finally running

要求

[版本 5](#)(See 2.64)

请参阅

[throw 语句](#)(See 1.2.11.16)

1.2.11.18 var 语句

声明[变量](#)。

```
var variable1 [ = value1 ] [, variable2 [ = value2], ...]
```

参数

variable, variable2

被声明的变量的名字。

value, value2

赋给变量的初始化值。

说明

使用 **var** 语句来声明变量。这些变量可以在声明时或声明后在脚本中被赋值。下面是声明的例子：

示例

下例阐明了 **var** 语句的用法。

```
var index;
```

```
var name = "Thomas Jefferson";
```

```
var answer = 42, counter, numpages = 10;
```

要求

[版本 1](#)(See 2.64)

请参阅

[function 语句](#)(See 1.2.11.10) | [new 运算符](#)(See 1.2.9.32)

1.2.11.19 while 语句

执行一个语句，直到指定的条件为 **false**。

```
while (expression)  
    statements
```

参数

expression

必选项。[Boolean 表达式](#)，在循环的每次迭代前被检查。如果 *expression* 是 **true**，则执行循环。如果 *expression* 是 **false**，则结束循环。

statements

可选项。*expression* 是 **true** 时要执行的一个或多个语句。

说明

while 语句在循环第一次被执行前检查 *expression*。如果 *expression* 此次是 **false**，则该循环一次都不被执行。

示例

下面示例说明了 **while** 语句的用法：

```
function BreakTest(breakpoint) {  
    var i = 0;  
    while (i < 100)  
    {  
        if (i == breakpoint)  
            break;  
        i++;  
    }  
    return(i);  
}
```

要求

[版本 1](#) (See 2.64)

请参阅

[break 语句](#) (See 1.2.11.4) | [continue 语句](#) (See 1.2.11.6) | [do...while 语句](#) (See 1.2.11.7) | [for 语句](#) (See 1.2.11.8) | [for...in 语句](#) (See 1.2.11.9)

1.2.11.20 with 语句

为语句设定默认对象。

```
with (object)
    statements
```

参数

object

新的默认对象。

statements

一个或多个语句，*object* 是该语句的默认对象。

说明

with 语句通常用来缩短特定情形下必须写的代码量。在下面的例子中，请注意 **Math** 的重复使用：

```
x      =      Math.cos (3      *      Math.PI)      +
Math.sin (Math.LN10)
```

```
y = Math.tan (14 * Math.E)
```

当使用 **with** 语句时，代码变得更短且更易读：

```
with (Math) {
    x = cos (3 * PI) + sin (LN10)
    y = tan (14 * E)
}
```


要求

[版本 1](#)(See 2.64)

请参阅

[this 语句](#)(See 1.2.11.15)

1.3 FileSystemObject 用户指南

[FileSystemObject 对象模式](#)(See 1.3.1)

[FileSystemObject 和 Scripting 运行时库参考的介绍](#)(See 1.3.2)

[FileSystemObject 对象](#)(See 1.3.3)

[设计 FileSystemObject](#)(See 1.3.4)

[处理驱动器和文件夹](#)(See 1.3.5)

[处理文件](#)(See 1.3.6)

[FileSystemObject 示例代码](#)(See 1.3.7)

1.3.1 FileSystemObject 对象模型

在为 Active Server Pages、Windows Scripting Host、或其他可以使用 scripting 的应用程序写脚本时，在 Web 服务器上添加、移动、改变、创建或删除文件夹（目录）和文件通常是很重要的。获得有关信息和操作连接在 Web 服务器上的驱动器也可能是必要的。

Scripting 允许用 **FileSystemObject** (FSO) 对象模式来处理驱动器、文件夹和文件，在下面的章节中将解释该对象模式：

- [FileSystemObject 和 Scripting 运行时库参考的介绍](#)(See 1.3.2)
- [FileSystemObject 对象](#)(See 1.3.3)
- [设计 FileSystemObject](#)(See 1.3.4)
- [处理驱动器和文件夹](#)(See 1.3.5)
- [处理文件](#)(See 1.3.6)
- [FileSystemObject 示例代码](#)(See 1.3.7)

1.3.2 FileSystemObject 和 Scripting 运行时库参考的介绍

FileSystemObject (FSO) 对象模式，允许对大量的属性、方法和事件，使用较熟悉的 `object.method` 语法，来处理文件夹和文件。

使用这个基于对象的工具和：

- HTML 来创建 Web 页
- Windows Scripting Host 来为 Microsoft Windows 创建批文件
- Script Control 来对用其他语言开发的应用程序提供编辑脚本的能力

因为在客户端使用 FSO 而引起重要的安全性问题，提供潜在地不受欢迎的对客户本地文件系统的访问，假定本文档使用 FSO 对象模式，来创建由服务器端的 Internet Web 页执行的脚本。因为使用了服务器端，Internet Explorer 默认安全设置不允许客户端使用 **FileSystemObject** 对象。覆盖那些默认值可能会引起在本地计算机上不受欢迎的对其文件系统的访问，从而导致文件系统完整性的全部破坏，同时引起数据遗失或更糟的情况。

FSO 对象模式使服务器端的应用程序能创建、改变、移动和删除文件夹，或探测特定的文件夹是否存在，若存在，还可以找出有关文件夹的信息，如名称、被创建或最后一次修改的日期，等等。

FSO 对象模式还使文件处理变得很容易。在处理文件时，主要的目标是以易于访问的格式把数据存储在有有效的空间和资源中。这就要求能够创建文件，插入和改变数据，以及输出（读取）数据。因为把数据存储数据库中，如 Access 或 SQL 服务器，会给应用程序增加很大的开销，所以把数据存储成二进制或文本文件中可能是最有效的解决方案。可能不希望有该开销，或者数据访问要求可能不需要与功能完备的数据库相关联的所有额外功能。

包含在 Scripting 类型库 (Scrpt.dll) 中的 FSO 对象模式，支持通过 **TextStream** 对象来创建和操作文本文件。虽然还不支持二进制文件的创建或操作，但计划将来要支持二进制文件。

1.3.3 FileSystemObject 对象

对象/集合	描述
FileSystemObject	主对象。包含用来创建、删除和获得有关信息，以及通常用来操作驱动器、文件夹和文件的方法和属性。和该对象相关联的许多方法，与其他 FSO 对象中的方法完全相似；它们是为了方便才被提供的。
Drive	对象。包含用来收集信息的方法和属性，这些信息是关于连接在系统上的驱动器的，如驱动器的共享名和它有多少可用空间。请注意，"drive" 并非必须是硬盘，也可以是 CD-ROM 驱动器，RAM 磁盘等等。并非必须把驱动器实物地连接到系统上；它也可以通过网络在逻辑上被连接起来。
Drives	集合。提供驱动器的列表，这些驱动器实物地或在逻辑上与系统相连接。 Drives 集合包括所有驱动器，与类型无关。要可移动的媒体驱动器在该集合中显现，不必把媒体插入到驱动器中。
File	对象。包含用来创建、删除或移动文件的方法和属性。也用来向系统询问文件名、路径和多种其他属性。
Files	集合。提供包含在文件夹内的所有文件的列表。
Folder	对象。包含用来创建、删除或移动文件夹的方法和属性。也用来向系统询

	问文件夹名、路径和多种其他属性。
Folders	集合。提供在 Folder 内的所有文件夹的列表。
TextStream	对象。用来读写文本文件。

1.3.4 设计 FileSystemObject

要用 **FileSystemObject** (FSO) 对象模式来编程，则：

- 使用 `CreateObject` 方法来创建 **FileSystemObject** 对象。
- 在新创建的对象上使用适当的方法。
- 访问对象的属性。

FSO 对象模式包含在 `Scripting` 类型库中，该库位于 `Scrrun.dll` 文件中。因而，要使用 FSO 对象模式，必须把 `Scrrun.dll` 放在 Web 服务器的适当系统目录中。

创建 FileSystemObject 对象

首先，使用 **CreateObject** 对象来创建 **FileSystemObject** 对象，在 VBScript 中，使用下面的代码来创建 **FileSystemObject** 的一个实例：

```
Dim fso

Set                               fso                               =
CreateObject("Scripting.FileSystemObject")
```

示例代码示范了如何创建 **FileSystemObject** 的一个实例。在 JScript 中，使用下面的代码来做同样的事情：

```
var fso;

fso                               =                               new
ActiveXObject("Scripting.FileSystemObject");
```

在这两个示例中，**Scripting** 是类型库的名字，而 **FileSystemObject** 则是想要创建的对象的名字。可以只创建 **FileSystemObject** 对象的一个实例，而不管试图创建另一个实例的次数。

使用适当的方法

其次，使用 **FileSystemObject** 对象的适当方法。例如，要创建一个新的对象，则

使用 **CreateTextFile** 或 **CreateFolder**（FSO 对象模式不支持驱动器的创建或删除）。

要删除对象，则使用 **FileSystemObject** 对象的 **DeleteFile** 和 **DeleteFolder** 方法，或 **File** 和 **Folder** 对象的 **Delete** 方法。也可以使用适当的方法，来复制和移动文件与文件夹。

注意 **FileSystemObject** 对象模式中的某些功能是多余的。例如，可以用 **FileSystemObject** 对象的 **CopyFile** 方法，也可以用 **File** 对象的 **Copy** 方法来复制文件。这两种方法功能是相同的；两种方法都能使编程灵活。

访问现有驱动器、文件和文件夹

要访问现有驱动器、文件或文件夹，则使用 **FileSystemObject** 对象中的适当的 "get" 方法：

- `GetDrive`
- `GetFolder`
- `GetFile`

在 VBScript 中要访问现有文件：

```
Dim fso, f1

Set fso = CreateObject("Scripting.FileSystemObject")

Set f1 = fso.GetFile("c:\test.txt")

在 JScript 中要做同样的事情，则使用下面的代码：

var fso, f1;

fso = new ActiveXObject("Scripting.FileSystemObject");

f1 = fso.GetFile("c:\\test.txt");
```

不要对新创建的对象使用 "get" 方法，因为 "create" 函数已经返回那个对象的一个句柄。例如，如果使用 **CreateFolder** 方法创建了一个新的文件夹，则不要使用 **GetFolder** 方法来访问它的属性，如 **Name**、**Path**、**Size** 等等。只需设一个变量给

CreateFolder 函数，来获得新创建文件夹的句柄名，然后访问它的属性、方法和事件。要在 VBScript 中这样做，则使用下面的代码：

```
Sub CreateFolder

    Dim fso, fldr

    Set fso =
CreateObject("Scripting.FileSystemObject")

    Set fldr =
fso.CreateFolder("C:\MyTest")

    Response.Write "Created folder: " &
fldr.Name

End Sub
```

在 JScript 中给 **CreateFolder** 函数设置一个变量，则使用下面的语法：

```
function CreateFolder()

{

    var fso, fldr;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    fldr =
fso.CreateFolder("C:\\MyTest");

    Response.Write("Created folder: " +
fldr.Name);

}
```

访问对象的属性

一旦有了对象的句柄，就可以访问它的属性。例如，要获得特定文件夹的名字，首先创建该对象的一个实例，然后用适当的方法获得它的句柄（在这个例子中是 **GetFolder** 方法，因为该文件夹已经存在了）。

在 VBScript 中，用该代码来获得 **GetFolder** 方法的一个句柄：

```
Set fldr = fso.GetFolder("c:\")
```

在 JScript 中要做同样的事情，则使用下面的代码：

```
var fldr = fso.GetFolder("c:\\");
```

现在，已经有了 **Folder** 对象的句柄，就可以检查它的 **Name** 属性了。在 VBScript 中使用下面的代码来检查：

```
Response.Write "Folder name is: " &
fldr.Name
```

在 JScript 中要检查 **Name** 属性，则使用下面的语法：

```
Response.Write("Folder name is: " +
fldr.Name);
```

要找出最后一次修改文件的时间，则使用下面的 VBScript 语法：

```
Dim fso, fl
```

```
Set fso =
CreateObject("Scripting.FileSystemObject")
```

' 获得要查询的文件对象。

```
Set fl = fso.GetFile("c:\detlog.txt")
```

' 打印信息。

```
Response.Write "File last modified: " &
fl.DateLastModified
```

要用 JScript 找出上一次修改的文件，则使用下面的代码：

```
var fso, fl;
```

```
fso                                =                                new
ActiveXObject("Scripting.FileSystemObject");

// 获得要查询的文件对象。

f1 = fso.GetFile("c:\\detlog.txt");

// 打印信息。

Response.Write("File last modified: " +
f1.DateLastModified);
```

1.3.5 处理驱动器和文件夹

使用 **FileSystemObject** (FSO) 对象模式，可以有计划地处理驱动器和文件夹，就像在 Windows 资源管理器中交互式地处理它们一样。可以复制和移动文件夹，获取有关驱动器和文件夹的信息，等等。

获取有关驱动器的信息

可以用 **Drive** 对象来获得有关各种驱动器的信息，这些驱动器是实物地或通过网络连接到的系统上的。它的属性可以用来获得下面的信息内容：

- 驱动器的总容量，以字节为单位（**TotalSize** 属性）
- 驱动器的可用空间是多少，以字节为单位（**AvailableSpace** 或 **FreeSpace** 属性）
- 哪个号被赋给了该驱动器（**DriveLetter** 属性）
- 驱动器的类型是什么，如可移动的、固定的、网络的、CD-ROM 或 RAM 磁盘（**DriveType** 属性）
- 驱动器的序列号（**SerialNumber** 属性）
- 驱动器使用的文件系统类型，如 FAT、FAT32、NTFS 等等（**FileSystem** 属性）
- 驱动器是否可以使用（**IsReady** 属性）
- 共享和/或卷的名字（**ShareName** 和 **VolumeName** 属性）
- 驱动器的路径或根文件夹（**Path** 和 **RootFolder** 属性）

请考察 [示例代码](#) (See 1.3.7)，来领会如何在 **FileSystemObject** 中使用这些属性。

Drive 对象的用法示例

使用 **Drive** 对象来收集有关驱动器的信息。在下面的代码中,没有对实际的 **Drive** 对象的引用;相反,使用 **GetDrive** 方法来获得现有 **Drive** 对象的引用(在这个例子中就是 **drv**)。

下面示例示范了如何在 VBScript 中使用 **Drive** 对象:

```
Sub ShowDriveInfo(drvPath)

    Dim fso, drv, s

    Set fso =
CreateObject("Scripting.FileSystemObject")

    Set drv =
fso.GetDrive(fso.GetDriveName(drvPath))

    s = "Drive " & UCase(drvPath) & " -
"

    s = s & drv.VolumeName & "<br>"
    s = s & "Total Space: " &
FormatNumber(drv.TotalSize / 1024, 0)
    s = s & " Kb" & "<br>"
    s = s & "Free Space: " &
FormatNumber(drv.FreeSpace / 1024, 0)
    s = s & " Kb" & "<br>"

    Response.Write s

End Sub
```

下面的代码说明在 JScript 中实现同样的功能:

```
function ShowDriveInfo1(drvPath)
{
```



```

var fso, drv, s = "";

fso = new
ActiveXObject("Scripting.FileSystemObject");

drv =
fso.GetDrive(fso.GetDriveName(drvPath));

s += "Drive " +
drvPath.toUpperCase() + " - ";

s += drv.VolumeName + "<br>";

s += "Total Space: " + drv.TotalSize
/ 1024;

s += " Kb" + "<br>";

s += "Free Space: " + drv.FreeSpace
/ 1024;

s += " Kb" + "<br>";

Response.Write(s);
}

```

处理文件夹

在下面的表中，描述了普通的文件夹任务和执行它们的方法。

任务	方法
创建文件夹。	FileSystemObject.CreateFolder
删除文件夹。	Folder.Delete 或 FileSystemObject.DeleteFolder
移动文件夹。	Folder.Move 或 FileSystemObject.MoveFolder
复制文件夹。	Folder.Copy 或 FileSystemObject.CopyFolder
检索文件夹的名字。	Folder.Name
如果文件夹在驱动器上存在，则找出	FileSystemObject.FolderExists

它。	
获得现有 Folder 对象的实例。	FileSystemObject.GetFolder
找出文件夹的父文件夹名。	FileSystemObject.GetParentFolderName
找出系统文件夹的路径。	FileSystemObject.GetSpecialFolder

请考察[示例代码](#)(See 1.3.7)，来看看在 **FileSystemObject** 中使用了多少种这些的方法和属性。

下面的示例示范了如何在 VBScript 中使用 **Folder** 和 **FileSystemObject** 对象，来操作文件夹和获得有关它们的信息：

```
Sub ShowFolderInfo()

    Dim fso, fldr, s

    ' 获得 FileSystemObject 的实例。

    Set                fso                =
CreateObject("Scripting.FileSystemObject")

    ' 获得 Drive 对象。

    Set fldr = fso.GetFolder("c:")

    ' 打印父文件夹名字。

    Response.Write "Parent folder name is:
" & fldr & "<br>"

    ' 打印驱动器名字。

    Response.Write "Contained on drive " &
fldr.Drive & "<br>"

    ' 打印根文件名。

    If fldr.IsRootFolder = True Then

        Response.Write "This is the root
```

```

folder." & "<br><br>"

Else

    Response.Write "This folder isn't a
root folder." & "<br><br>"

End If

' 用 FileSystemObject 对象创建新的文件夹。
fso.CreateFolder ("C:\Bogus")

Response.Write "Created folder
C:\Bogus" & "<br>"

' 打印文件夹的基本名字。
Response.Write "Basename = " &
fso.GetBaseName("c:\bogus") & "<br>"

' 删除新创建的文件夹。
fso.DeleteFolder ("C:\Bogus")

Response.Write "Deleted folder
C:\Bogus" & "<br>"

End Sub

```

下面的示例显示如何在 JScript 中使用 **Folder** 和 **FileSystemObject** 对象：

```

function ShowFolderInfo()
{
    var fso, fldr, s = "";
    // 获得 FileSystemObject 的实例。
    fso = new

```

```
ActiveXObject("Scripting.FileSystemObject");

// 获得 Drive 对象。
fldr = fso.GetFolder("c:");

// 打印父文件夹名。
Response.Write("Parent folder name is:
" + fldr + "<br>");

// 打印驱动器名字。
Response.Write("Contained on drive " +
fldr.Drive + "<br>");

// 打印根文件名。
if (fldr.IsRootFolder)
    Response.Write("This is the root
folder.");
else
    Response.Write("This folder isn't a
root folder.");

Response.Write("<br><br>");

// 用 FileSystemObject 对象创建新的文件夹。
fso.CreateFolder ("C:\\Bogus");

Response.Write("Created          folder
C:\\Bogus" + "<br>");
```

```
// 打印文件夹的基本名。

Response.Write("Basename      =      "      +
fso.GetBaseName("c:\\bogus") + "<br>");

// 删除新创建的文件夹。

fso.DeleteFolder ("C:\\Bogus");

Response.Write("Deleted          folder
C:\\Bogus" + "<br>");
}
```

1.3.6 处理文件

有两种主要的文件处理类型：

- 创建、添加或删除数据，以及读取文件
- 移动、复制和删除文件

创建文件

创建空文本文件（有时被叫做“文本流”）有三种方法。

第一种方法是用 **CreateTextFile** 方法。下面的示例示范了在 VBScript 中如何用这种方法来创建文本文件：

```
Dim fso, fl

Set                                fso                                =
CreateObject("Scripting.FileSystemObjec
t")

Set                                fl                                =
fso.CreateTextFile("c:\testfile.txt",
True)
```

要在 JScript 中用这种方法，则使用下面的代码：

```
var fso, fl;  
  
fso = new  
ActiveXObject("Scripting.FileSystemObject");  
  
fl =  
fso.CreateTextFile("c:\\testfile.txt",  
true);
```

创建文本文件的第二种方法是，使用 **FileSystemObject** 对象的 **OpenTextFile** 方法，并设置 **ForWriting** 标志。在 VBScript 中，代码就像下面的示例一样：

```
Dim fso, ts  
  
Const ForWriting = 2  
  
Set fso = CreateObject("Scripting.  
FileSystemObject")  
  
Set ts = fso.OpenTextFile("c:\\test.txt",  
ForWriting, True)
```

要在 JScript 中使用这种方法来创建文本文件，则使用下面的代码：

```
var fso, ts;  
  
var ForWriting= 2;  
  
fso = new  
ActiveXObject("Scripting.FileSystemObject");  
  
ts = fso.OpenTextFile("c:\\test.txt",  
ForWriting, true);
```

创建文本文件的第三种方法是，使用 **OpenAsTextStream** 方法，并设置

ForWriting 标志。要使用这种方法，在 VBScript 中使用下面的代码：

```
Dim fso, f1, ts

Const ForWriting = 2

Set fso = CreateObject("Scripting.FileSystemObject")

fso.CreateTextFile ("c:\test1.txt")

Set f1 = fso.GetFile("c:\test1.txt")

Set ts = f1.OpenAsTextStream(ForWriting, True)
```

在 JScript 中，则使用下面示例中的代码：

```
var fso, f1, ts;

var ForWriting = 2;

fso = new ActiveXObject("Scripting.FileSystemObject");

fso.CreateTextFile ("c:\\test1.txt");

f1 = fso.GetFile("c:\\test1.txt");

ts = f1.OpenAsTextStream(ForWriting, true);
```

添加数据到文件中

一旦创建了文本文件，使用下面的三个步骤向文件添加数据：

打开文本文件。

写入数据。

关闭文件。

要打开现有的文件，则使用 `FileSystemObject` 对象的 `OpenTextFile` 方法或 `File` 对象的 `OpenAsTextStream` 方法。

要写数据到打开的文本文件，则根据下表所述任务使用 `TextStream` 对象的 `Write`、`WriteLine` 或 `WriteBlankLines` 方法。

任务	方法
向打开的文本文件写数据，不用后续一个换行字符。	<code>Write</code>
向打开的文本文件写数据，后续一个换行字符。	<code>WriteLine</code>
向打开的文本文件写一个或多个空白行。	<code>WriteBlankLines</code>

要关闭一个打开的文件，则使用 `TextStream` 对象的 `Close` 方法。

注意 换行字符包含一个或几个字符（取决于操作系统），以把光标移动到下一行的开始位置（回车/换行）。注意某些字符串末尾可能已经有这个非打印字符了。下面的 `VBScript` 例子示范了如何打开文件，和同时使用三种写方法来向文件添加数据，然后关闭文件：

```
Sub CreateFile()

    Dim fso, tf

    Set fso =
CreateObject("Scripting.FileSystemObject")

    Set tf =
fso.CreateTextFile("c:\testfile.txt",
True)

    ' 写一行，并且带有换行字符。

    tf.WriteLine("Testing 1, 2, 3.")

    ' 向文件写三个换行字符。

    tf.WriteBlankLines(3)

    ' 写一行。

    tf.Write ("This is a test.")

    tf.Close
```


End Sub

这个示例示范了在 JScript 中如何使用这三个方法：

```
function CreateFile()  
{  
    var fso, tf;  
    fso = new  
ActiveXObject("Scripting.FileSystemObject");  
    tf =  
fso.CreateTextFile("c:\\testfile.txt",  
true);  
    // 写一行，并且带有换行字符。  
    tf.WriteLine("Testing 1, 2, 3.") ;  
    // 向文件写三个换行字符。  
    tf.WriteLine(3) ;  
    // 写一行。  
    tf.Write ("This is a test.");  
    tf.Close();  
}
```

读取文件

要从文本文件读取数据，则使用 **TextStream** 对象的 **Read**、**ReadLine** 或 **ReadAll** 方法。下表描述了不同的任务应使用哪种方法。

任务	方法
从文件读取指定数量的字符。	Read
读取一整行（一直到但不包括换行字符）。	ReadLine

读取文本文件的整个内容。	ReadAll

如果使用 **Read** 或 **ReadLine** 方法，并且想跳过数据的特殊部分，则使用 **Skip** 或 **SkipLine** 方法。**read** 方法的结果文本存在一个字符串中，该字符串可以显示在一个控件中，也可以用字符串函数（如 **Left**、**Right** 和 **Mid**）来分析，连接等等。

下面的 VBScript 示例示范了如何打开文件，和如何写数据到文件中并从文件读取数据：

```
Sub ReadFiles
```

```
    Dim fso, fl, ts, s
```

```
    Const ForReading = 1
```

```
    Set                                fso                                =
```

```
CreateObject("Scripting.FileSystemObject")
```

```
    Set                                fl                                =
```

```
fso.CreateTextFile("c:\testfile.txt",
True)
```

```
    ' 写一行。
```

```
    Response.Write "Writing file <br>"
```

```
    fl.WriteLine "Hello World"
```

```
    fl.WriteLineBlankLines(1)
```

```
    fl.Close
```

```
    ' 读取文件的内容。
```

```
    Response.Write "Reading file <br>"
```

```
    Set                                ts                                =
```

```
fso.OpenTextFile("c:\testfile.txt",
```

```
ForReading)
    s = ts.ReadLine
    Response.Write "File contents = '" & s
& "'"
    ts.Close
End Sub
```

下面的代码示范了在 JScript 中做同样的事：

```
function ReadFiles()
{
    var fso, f1, ts, s;
    var ForReading = 1;
    fso = new
ActiveXObject("Scripting.FileSystemObject");
    f1 =
fso.CreateTextFile("c:\\testfile.txt",
true);
    // 写一行。
    Response.Write("Writing file <br>");
    f1.WriteLine("Hello World");
    f1.WriteLineBlankLines(1);
    f1.Close();
    // 读取文件的内容。
```

```

Response.Write("Reading file <br>");

ts                                     =
fso.OpenTextFile("c:\\testfile.txt",
ForReading);

s = ts.ReadLine();

Response.Write("File contents = '" + s
+ "'");

ts.Close();

}

```

移动、复制和删除文件

FSO 对象模式各有两种方法移动、复制和删除文件，如下表所述。

任务	方法
移动文件	File.Move 或 FileSystemObject.MoveFile
复制文件	File.Copy 或 FileSystemObject.CopyFile
删除文件	File.Delete 或 FileSystemObject.DeleteFile

下面的 VBScript 示例，在驱动器 C 的根目录中创建一个文本文件，向其中写一些信息，然后把它移动到 \tmp 目录中，并在 \temp 中做一个备份，最后把它们从两个目录中删掉。

要运行下面的示例，需要先在驱动器 C 的根目录中创建 \tmp 和 \temp 目录：

```

Sub ManipFiles

    Dim fso, f1, f2, s

    Set                                     fso                                     =
CreateObject("Scripting.FileSystemObject")

    Set                                     f1                                     =
fso.CreateTextFile("c:\testfile.txt",

```

```
True)

    Response.Write "Writing file <br>"

    ' 写一行。

f1.Write ("This is a test.")

    ' 关闭文件。

f1.Close

    Response.Write "Moving file to c:\tmp
<br>"

    ' 获取 C 的根目录 (C:\) 中的文件的句柄。

Set f2 = fso.GetFile("c:\testfile.txt")

    ' 把文件移动到 \tmp 目录。

f2.Move ("c:\tmp\testfile.txt")

    Response.Write "Copying file to
c:\temp <br>"

    ' 把文件复制到 \temp 目录。

f2.Copy ("c:\temp\testfile.txt")

    Response.Write "Deleting files <br>"

    ' 获得文件当前位置的句柄。

Set                f2                =
fso.GetFile("c:\tmp\testfile.txt")

    Set                f3                =
fso.GetFile("c:\temp\testfile.txt")
```

```
' 删除文件。
```

```
f2.Delete
```

```
f3.Delete
```

```
Response.Write "All done!"
```

```
End Sub
```

下面的代码示范了在 JScript 中做同样的事：

```
function ManipFiles()
```

```
{
```

```
var fso, f1, f2, s;
```

```
    fso                                =                new  
ActiveXObject("Scripting.FileSystemObject");
```

```
    f1                                =  
fso.CreateTextFile("c:\\testfile.txt",  
true);
```

```
    Response.Write("Writing file <br>");
```

```
    // 写一行。
```

```
f1.Write("This is a test.");
```

```
    // 关闭文件。
```

```
f1.Close();
```

```
    Response.Write("Moving      file      to  
c:\\tmp <br>");
```

```
    // 获取 C 的根目录 (C:\) 中的文件的句柄。
```

```

f2 = fso.GetFile("c:\\testfile.txt");

// 把文件移动到 \tmp 目录。

f2.Move ("c:\\tmp\\testfile.txt");

Response.Write("Copying file to
c:\\temp <br>");

// 把文件复制到 \temp 目录。

f2.Copy ("c:\\temp\\testfile.txt");

Response.Write("Deleting files
<br>");

// 获得文件当前位置的句柄。

f2 =
fso.GetFile("c:\\tmp\\testfile.txt");

f3 =
fso.GetFile("c:\\temp\\testfile.txt");

// 删除文件。

f2.Delete();

f3.Delete();

Response.Write("All done!");
}

```

1.3.7 FileSystemObject 示例代码

在本节描述的示例代码，提供真实的例子来示范在 **FileSystemObject** 对象模式中可用的许多功能。该代码显示了如何一起使用对象模式的所有功能，以及如何在您自己的代码中有效地使用这些功能。

请注意，由于该代码是极一般的，所以要使该代码能够真正在您的机器上运行，可

.....

.....

1) 下面的代码有许多字符串操作，用"&"运算符来把短字符串连接在一起。由于字符串连接是费时的，所以这是一种低效率的写代码方法。无论如何，它是一种非常好维护的写代码方法，并且在这儿使用了这种方法，因为该程序执行大量的磁盘操作，而磁盘操作比连接字符串所需的内存操作要慢得多。

记住这是示范代码，而不是产品代码。

DriveTypeCDROM 误拼

3) 为了使代码更可读, 该代码中没有错误处理。虽然采取了防范措施, 来保证代码在普通情况下没有错误, 但文件系统是不可预知的。在产品代码中, 使用

388

[illegible]

▼ ▼

[illegible]

Dim TabStop

Dim NewLine

```
Const TestDrive = "C"
```

```
Const TestFilePath = "C:\Test"
```

▼ ▼

[illegible]

```
Const DriveTypeRemovable = 1
```

```
Const DriveTypeFixed = 2
```

```
Const DriveTypeNetwork = 3
```

```
Const DriveTypeCDROM = 4
```

```
Const DriveTypeRAMDisk = 5
```

.....

.....

' 由 File.Attributes 返回的常数

.....

.....

Const FileAttrNormal = 0

Const FileAttrReadOnly = 1

Const FileAttrHidden = 2

Const FileAttrSystem = 4

Const FileAttrVolume = 8

Const FileAttrDirectory = 16

Const FileAttrArchive = 32

Const FileAttrAlias = 64

Const FileAttrCompressed = 128

.....

.....

' 用来打开文件的常数

.....

.....

Const OpenFileForReading = 1

Const OpenFileForWriting = 2

```
Const OpenFileForAppending = 8
```

```
.....
```

```
.....
```

```
' ShowDriveType
```

```
' 目的:
```

```
' 生成一个字符串, 来描述给定 Drive 对象的驱动器类型。
```

```
' 示范下面的内容
```

```
' - Drive.DriveType
```

```
.....
```

```
.....
```

```
Function ShowDriveType(Drive)
```

```
    Dim S
```

```
    Select Case Drive.DriveType
```

```
    Case DriveTypeRemovable
```

```
        S = "Removable"
```

```
    Case DriveTypeFixed
```

```
        S = "Fixed"
```

```
    Case DriveTypeNetwork
```

```
        S = "Network"
```

```
    Case DriveTypeCDROM
```

```
        S = "CD-ROM"
```

```

Case DriveTypeRAMDisk
    S = "RAM Disk"

Case Else
    S = "Unknown"

End Select

ShowDriveType = S

End Function

.....

' ShowFileAttr

' 目的:

' 生成一个字符串, 来描述文件或文件夹的属性。

' 示范下面的内容

' - File.Attributes

' - Folder.Attributes

.....

.....

Function ShowFileAttr(File) ' File 可以是文
件或文件夹

Dim S

```

```
Dim Attr
```

```
Attr = File.Attributes
```

```
If Attr = 0 Then
```

```
    ShowFileAttr = "Normal"
```

```
    Exit Function
```

```
End If
```

```
    If Attr And FileAttrDirectory    Then S  
= S & "Directory "
```

```
    If Attr And FileAttrReadOnly    Then S  
= S & "Read-Only "
```

```
    If Attr And FileAttrHidden      Then S  
= S & "Hidden "
```

```
    If Attr And FileAttrSystem      Then S  
= S & "System "
```

```
    If Attr And FileAttrVolume      Then S  
= S & "Volume "
```

```
    If Attr And FileAttrArchive     Then S  
= S & "Archive "
```

```
    If Attr And FileAttrAlias       Then S  
= S & "Alias "
```

```
    If Attr And FileAttrCompressed Then S
```

= S & "Compressed "

$$\text{ShowFileAttr} = S$$

End Function

|||||

|||||

' GenerateDriveInformation

目的:

生成一个字符串，来描述可用驱动器的当前状态。

示范下面的内容

```
' - FileSystemObject.Drives
```

```
' - Iterating the Drives collection
```

' - Drives.Count

```
' - Drive.AvailableSpace
```

```
' - Drive.DriveLetter
```

```
' - Drive.DriveType
```

```
' - Drive.FileSystem
```

```
' - Drive.FreeSpace
```

```
' - Drive.IsReady
```

' - Drive.Path

' - Drive.SerialNumber

```
' - Drive.ShareName
```

395


```
S = S & TabStop & "Path"
S = S & TabStop & "Type"
S = S & TabStop & "Ready?"
S = S & TabStop & "Name"
S = S & TabStop & "System"
S = S & TabStop & "Space"
S = S & TabStop & "Space"
S = S & TabStop & "Space"
S = S & TabStop & "Number" & NewLine

' 分隔行。
S = S & String(105, "-") & NewLine

For Each Drive In Drives
    S = S & Drive.DriveLetter
    S = S & TabStop & Drive.Path
    S = S & TabStop & ShowDriveType(Drive)
    S = S & TabStop & Drive.IsReady

    If Drive.IsReady Then
        If DriveTypeNetwork =
Drive.DriveType Then
            S = S & TabStop & Drive.ShareName
        Else
```

.....

.....

- ▼ 目的:
- ▼ 生成一个字符串, 来描述文件的当前状态。
- ▼ 示范下面的内容

```
' - File.Path
' - File.Name
' - File.Type
' - File.DateCreated
' - File.DateLastAccessed
' - File.DateLastModified
' - File.Size
.....
.....
```

```
Function GenerateFileInformation(File)
```

```
    Dim S
```

```
    S = NewLine & "Path:" & TabStop &
File.Path
```

```
    S = S & NewLine & "Name:" & TabStop &
File.Name
```

```
    S = S & NewLine & "Type:" & TabStop &
File.Type
```

```
    S = S & NewLine & "Attribs:" & TabStop
& ShowFileAttr(File)
```

```
    S = S & NewLine & "Created:" & TabStop
& File.DateCreated
```

```
S = S & NewLine & "Accessed:" & TabStop
& File.DateLastAccessed
```

```
S = S & NewLine & "Modified:" & TabStop
& File.DateLastModified
```

```
S = S & NewLine & "Size" & TabStop &
File.Size & NewLine
```

```
GenerateFileInformation = S
```

```
End Function
```

```
.....
```

```
.....
```

' **GenerateFolderInformation**

' 目的:

' 生成一个字符串, 来描述文件夹的当前状态。

' 示范下面的内容

' - Folder.Path

' - Folder.Name

' - Folder.DateCreated

' - Folder.DateLastAccessed

' - Folder.DateLastModified

' - Folder.Size

```
.....
```

|||||

Function

GenerateFolderInformation(Folder)

Dim S

```
S = "Path:" & TabStop & Folder.Path
```

```
S = S & NewLine & "Name:" & TabStop &
Folder.Name
```

```
S = S & NewLine & "Attrs:" & TabStop
& ShowFileAttr(Folder)
```

```
S = S & NewLine & "Created:" & TabStop
& Folder.DateCreated
```

```
S = S & NewLine & "Accessed:" & TabStop
& Folder.DateLastAccessed
```

```
S = S & NewLine & "Modified:" & TabStop
& Folder.DateLastModified
```

```
S = S & NewLine & "Size:" & TabStop &
Folder.Size & NewLine
```

```
GenerateFolderInformation = S
```

End Function

[illegible]

' GenerateAllFolderInformation

目的:

生成一个字符串，来描述一个文件夹和所有文件及子文件夹的当前状态。

示范下面的内容

' - Folder.Path

- ' - Folder.SubFolders

' - Folders.Count

[illegible][illegible]

Function

GenerateAllFolderInformation(Folder)

 $\text{Dim } S$

Dim SubFolders

Dim SubFolder

Dim Files

Dim File

```
S = "Folder:" & TabStop & Folder.Path
```

& NewLine & NewLine

```
Set Files = Folder.Files
```

```
If 1 = Files.Count Then
    S = S & "There is 1 file" & NewLine
Else
    S = S & "There are " & Files.Count &
" files" & NewLine
End If

If Files.Count <> 0 Then
    For Each File In Files
        S = S &
GenerateFileInformation(File)
    Next
End If

Set SubFolders = Folder.SubFolders

If 1 = SubFolders.Count Then
    S = S & NewLine & "There is 1 sub
folder" & NewLine & NewLine
Else
    S = S & NewLine & "There are " &
SubFolders.Count & " sub folders" &
NewLine & NewLine
```

.....

.....

- 目的:
- 生成一个字符串, 来描述 `C:\Test` 文件夹和所有文件及子文件夹的当前状态。
- 示范下面的内容


```
' - FileSystemObject.DriveExists
' - FileSystemObject.FolderExists
' - FileSystemObject.GetFolder
```

```
.....
.....
```

```
Function GenerateTestInformation(FSO)
```

```
    Dim TestFolder
```

```
    Dim S
```

```
    If Not FSO.DriveExists(TestDrive) Then
Exit Function
```

```
    If Not FSO.FolderExists(TestFilePath)
Then Exit Function
```

```
    Set                TestFolder                =
FSO.GetFolder(TestFilePath)
```

```
    GenerateTestInformation                =
GenerateAllFolderInformation(TestFolder)
```

```
End Function
```

```
.....
.....
```

' DeleteTestDirectory

目的:

清理 test 目录。

示范下面的内容

```
' - FileSystemObject.GetFolder
```

```
' - FileSystemObject.DeleteFile
```

```
' - FileSystemObject.DeleteFolder
```

- ' - Folder.Delete

' - File.Delete

[illegible]

▼ ▼

Sub DeleteTestDirectory(FSO)

Dim TestFolder

Dim SubFolder

Dim File

有两种方法可用来删除文件：

```
FSO.DeleteFile(TestFilePath &
"\Beatles\OctopusGarden.txt")
```

```
Set File = FSO.GetFile(TestFilePath &
"\Beatles\BathroomWindow.txt")
```

```
File.Delete
```

```
' 有两种方法可用来删除文件夹:
```

```
FSO.DeleteFolder (TestFilePath      &
"\Beatles")
```

```
FSO.DeleteFile (TestFilePath      &
"\ReadMe.txt")
```

```
Set      TestFolder      =
FSO.GetFolder (TestFilePath)
TestFolder.Delete
```

```
End Sub
```

```
.....
.....
```

' **CreateLyrics**

```
' 目的:
```

```
' 在文件夹中创建两个文本文件。
```

```
' 示范下面的内容
```

```
' - FileSystemObject.CreateTextFile
```

```
' - TextStream.WriteLine
```

```
' - TextStream.Write
```

```
' - TextStream.WriteLine
```

```
' - TextStream.Close
```

Dim TextStream

```
TextStream.Write("Octopus' Garden ")'
```

```
TextStream.WriteLine (" (by Ringo  
carr) ")
```

```
TextStream.WriteLine(1)
```

```

    TextStream.WriteLine("He'd let us in,  

    knows where we've been -- in his  

    octopus' garden in the shade.")

```

```
TextStream.WriteLineLines(2)
```

407

End Sub

[illegible]

▼ ▼

' GetLyrics

目的:

' 显示 lyrics 文件的内容。

```
' 示范下面的内容

'  - FileSystemObject.OpenTextFile
'  - FileSystemObject.GetFile
'  - TextStream.ReadAll
'  - TextStream.Close
'  - File.OpenAsTextStream
'  - TextStream.AtEndOfStream
'  - TextStream.ReadLine

' .....
' .....
```

```
Function GetLyrics(FSO)
```

```
    Dim TextStream
```

```
    Dim S
```

```
    Dim File
```

```
    ' 有多种方法可用来打开一个文本文件，和多种方法来从文件读取数据。
```

```
    ' 这儿用了两种方法来打开文件和读取文件：
```

```
    Set                TextStream                =
FSO.OpenTextFile(TestFilePath                &
"\Beatles\OctopusGarden.txt",
OpenFileForReading)
```

.....

.....

- ' 目的:
- ' 创建一个目录分层结构来示范 `FileSystemObject`。

```
' 以这样的次序来创建分层结构:
' C:\Test
' C:\Test\ReadMe.txt
' C:\Test\Beatles
' C:\Test\Beatles\OctopusGarden.txt
' C:\Test\Beatles\BathroomWindow.txt
' 示范下面的内容
' - FileSystemObject.DriveExists
' - FileSystemObject.FolderExists
' - FileSystemObject.CreateFolder
' - FileSystemObject.CreateTextFile
' - Folders.Add
' - Folder.CreateTextFile
' - TextStream.WriteLine
' - TextStream.Close
' . . . . .
' . . . . .
```

```
Function BuildTestDirectory(FSO)
```

```
    Dim TestFolder
```

```
    Dim SubFolders
```

```
    Dim SubFolder
```



```
Dim TextStream
```

```
' 排除 (a) 驱动器不存在, 或 (b) 要创建的目录已经存在的情况。
```

```
If Not FSO.DriveExists(TestDrive) Then
```

```
    BuildTestDirectory = False
```

```
    Exit Function
```

```
End If
```

```
If FSO.FolderExists(TestFilePath) Then
```

```
    BuildTestDirectory = False
```

```
    Exit Function
```

```
End If
```

```
Set TestFolder =
```

```
FSO.CreateFolder(TestFilePath)
```

```
Set TextStream =
```

```
FSO.CreateTextFile(TestFilePath &
```

```
"\ReadMe.txt")
```

```
TextStream.WriteLine("My song lyrics  
collection")
```

```
TextStream.Close
```

```
Set SubFolders = TestFolder.SubFolders
```

```
Set SubFolder =
```

```
SubFolders.Add("Beatles")

CreateLyrics SubFolder

BuildTestDirectory = True

End Function

'-----
'-----
' 主程序
' 首先，它创建一个 test 目录，以及一些子文件夹和文件。
' 然后，它转储有关可用磁盘驱动器和 test 目录的某些信息，
' 最后，清除 test 目录及其所有内容。
'-----
'-----

Sub Main

Dim FSO

' 设立全局变量。

TabStop = Chr(9)

NewLine = Chr(10)

Set FSO =
CreateObject("Scripting.FileSystemObjec
```

```
t")
```

```

    If Not BuildTestDirectory(FSO) Then
        Print "Test directory already exists
or cannot be created.  Cannot continue."
        Exit Sub
    End If

    Print GenerateDriveInformation(FSO) &
NewLine & NewLine
    Print GenerateTestInformation(FSO) &
NewLine & NewLine
    Print GetLyrics(FSO) & NewLine &
NewLine
    DeleteTestDirectory(FSO)

End Sub

```

1.4 Scripting 运行时库参考

◆ 特性信息

◆ 字母顺序的关键字列表

◆ 方法

欢迎使用 Scripting 运行时库参考

通过对信息进行分组，可以使您方便地研究 Scripting 运行时库的各个主题。

可以在“字母顺序的关键字列表”中找到按照字母顺序排列的 Scripting 运行时库的所有主题。如果只需要查看某一个主题（例如对象），则有对该主题进行详细说明的章节可供查阅。

如何操作呢？单击左边任意一个标题，即可显示该标题

对象

属性

所包含的项目列表。再从该列表中选择想要查看的主题。打开所选主题之后，就能够很容易地连接到其他相关章节。

请尽情浏览脚本运行时库参考的各个部分，您会发现脚本运行时库参考有多么丰富

特性	描述
Microsoft Scripting 运行时特性 (See 2.60)	目前 Microsoft Scripting 运行时库中的特性列表
语言元素	描述
Add 方法 (See 1.4.1.1)	向 Dictionary 对象中添加主键和项目对。
Add 方法 (See 1.4.1.2)	向 Folders 集合中添加新的 Folder 。
AtEndOfLine 属性 (See 1.4.3.1)	如果在 TextStream 文件中文件指针紧接在 end-of-line 标志之前则返回 True ；如果不是则返回 False 。
AtEndOfStream 属性 (See 1.4.3.2)	如果在 TextStream 文件中文件指针到达文件尾则返回 True ；如果不是则返回 False 。
Attributes 属性 (See 1.4.3.3)	设置或返回文件或文件夹的属性。
AvailableSpace 属性 (See 1.4.3.4)	向用户返回指定驱动器或网络共享上的可用空间大小。
BuildPath 方法 (See 1.4.1.3)	将一个名称追加到已有的路径后。
Close 方法 (See 1.4.1.4)	关闭打开的 TextStream 文件。
Column 属性 (See 1.4.3.5)	返回 TextStream 文件中当前字符所在的列号。
CompareMode 属性 (See 1.4.3.6)	设置和返回在 Dictionary 对象中对字符串键进行比较的比较方式。
Copy 方法 (See 1.4.1.5)	将指定文件或文件夹从一个位置复制到另一个位置。
CopyFile 方法 (See 1.4.1.6)	将一个或多个文件从一个位置复制到另一个位置。
CopyFolder 方法 (See 1.4.1.7)	将文件夹从一个位置完全复制到另一个位置。
Count 属性 (See 1.4.3.7)	返回集合或 Dictionary 对象中的项目数。

[CreateFolder 方法](#) 创建文件夹。

(See 1.4.1.8)

[CreateTextFile 方法](#) 创建指定的文件名并返回可用于读写该文件的

(See 1.4.1.9) **TextStream** 对象。

[DateCreated 属性](#) 返回指定文件或文件夹创建的日期和时间。

(See 1.4.3.8)

[DateLastAccessed 属性](#) 返回指定文件或文件夹最后一次访问的日期和
[性](#) (See 1.4.3.9) 时间。

[DateLastModified 属性](#) 返回指定文件或文件夹最后一次修改的日期和
[性](#) (See 1.4.3.10) 时间。

[Delete 方法](#) (See 删除指定的文件或文件夹。

1.4.1.10)

[DeleteFile 方法](#) (See 删除指定文件。

1.4.1.11)

[DeleteFolder 方法](#) 删除指定文件夹及其内容。

(See 1.4.1.12)

[Dictionary 对象](#) (See 存储数据键和项目对的对象。

1.4.2.1)

[Drive 对象](#) (See 提供了对特定磁盘驱动器或网络共享的属性的
1.4.2.2) 访问。

[Drive 属性](#) (See 返回指定文件或文件夹所在驱动器的驱动器
1.4.3.11) 号。

[DriveExists 方法](#) 如果指定驱动器存在则返回 **True** ； 否则返回
(See 1.4.1.13) **False** 。

[DriveLetter 属性](#) 返回本地物理驱动器或网络共享的驱动器号。

(See 1.4.3.12)

[Drives 集合](#) (See 包含所有可用驱动器的只读集合。

1.4.2.3)

[Drives 属性](#) (See 返回一个 **Drives** 集合，由所有本地机器上可
1.4.3.13) 用的 **Drive** 对象组成 。

[DriveType 属性](#) (See 返回说明指定驱动器类型的值。

1.4.3.14)

[Exists 方法](#) (See 如果 **Dictionary** 对象中存在指定键则返回
1.4.1.14) **True** ， 否则返回 **False** 。

[File 对象](#) (See 提供了对文件所有属性的访问。

1.4.2.4)

[FileExists 方法](#) (See 如果指定文件存在则返回 **True** ； 否则返回
1.4.1.15) **False** 。

[Files 集合](#) (See 文件夹中所有 **File** 对象的集合。

1.4.2.5)

[Files 属 性](#) (See 返回一个 **Files** 集合，由所有包含在指定文件夹中的 **File** 对象组成，包括那些设置了隐藏和系统文件属性的文件。
1. 4. 3. 15)

[FileSystemObject 对 象](#) (See 提供了对计算机文件系统的访问。
(See 1. 4. 2. 6)

[FileSystem 属 性](#) (See 返回指定驱动器所使用的文件系统类型。
1. 4. 3. 16)

[Folder 对 象](#) (See 提供了对文件夹所有属性的访问。
1. 4. 2. 7)

[Folders 集 合](#) (See 包含在一个 **Folder** 对象中的所有 **Folder** 对象的集合。
1. 4. 2. 8)

[FolderExists 方 法](#) 如果指定文件夹存在则返回 **True** ；否则返回
(See 1. 4. 1. 16) **False** 。

[FreeSpace 属 性](#) (See 向用户返回指定驱动器或网络共享上可用空间的大小。
1. 4. 3. 17)

[GetAbsolutePathName 方 法](#) 根据所提供的路径说明返回完整明确的路径。
(See 1. 4. 1. 17)

[GetBaseName 方 法](#) 根据提供的路径说明返回一个字符串，包含文件或文件夹的基本名称（不包括任何文件扩展名）。
(See 1. 4. 1. 18)

[GetDrive 方 法](#) (See 返回相应于指定路径中驱动器的 **Drive** 对象。
1. 4. 1. 19)

[GetDriveName 方 法](#) 根据指定路径返回包含驱动器名称的字符串。
(See 1. 4. 1. 20)

[GetExtensionName 方 法](#) 返回一个字符串，包含路径中最后成分的扩展名。
(See 1. 4. 1. 21)

[GetFile 方 法](#) (See 返回相应于指定路径中文件的 **File** 对象。
1. 4. 1. 22)

[GetFileName 方 法](#) 返回指定路径中的最后文件名或文件夹名，该路径不能是驱动器说明。
(See 1. 4. 1. 23)

[GetFileVersion 方 法](#) 返回指定文件的版本号码。
(See 1. 4. 1. 24)

[GetFolder 方 法](#) (See 返回相应于指定路径中文件夹的 **Folder** 对象。
1. 4. 1. 25)

[GetParentFolderName 方 法](#) 返回一个字符串，包含指定路径中最后文件或文件夹的父文件夹名。
(See 1. 4. 1. 26)

[GetSpecialFolder 方 法](#) 返回指定的特殊文件夹。
(See 1. 4. 1. 27)

[GetTempName 方 法](#) 返回一个随机生成的临时文件或文件夹名称。
(See 1. 4. 1. 28)

[IsReady 属性](#) (See 如果指定驱动器已就绪则返回 **True** ; 否则返回 **False** 。
1. 4. 3. 18)

[IsRootFolder 属性](#) 如果指定文件夹是根文件夹则返回 **True** ; 否则返回 **False** 。
(See 1. 4. 3. 19)

[Item 属性](#) (See 根据 **Dictionary** 对象中指定的 *key* 设置或返回 *item* 。
1. 4. 3. 20)

[Items 方法](#) (See 返回包含 **Dictionary** 对象中所有项目的数组。
1. 4. 1. 29)

[Key 属性](#) (See 在 **Dictionary** 对象中设置 *key* 。
1. 4. 3. 21)

[Keys 方法](#) (See 返回包含了 **Dictionary** 对象中全部已有键的数组。
1. 4. 1. 30)

[Line 属性](#) (See 返回 **TextStream** 文件中的当前行号。
1. 4. 3. 22)

[Move 方法](#) (See 将指定文件或文件夹从一个位置移动到另一个位置。
1. 4. 1. 31)

[MoveFile 方法](#) (See 从一个位置向另一个位置移动一个或多个文件。
1. 4. 1. 32)

[MoveFolder 方法](#) (See 从一个位置向另一个位置移动一个或多个文件夹。
1. 4. 1. 33)

[Name 属性](#) (See 设置或返回指定文件或文件夹的名称。
1. 4. 3. 23)

[OpenAsTextStream 方法](#) 打开指定文件并返回可用于读、写或追加该文件的 **TextStream** 对象。
(See 1. 4. 1. 34)

[OpenTextFile 方法](#) 打开指定文件并返回可用于读、写或追加该文件的 **TextStream** 对象。
(See 1. 4. 1. 35)

[ParentFolder 属性](#) 返回指定文件或文件夹的父文件夹对象。
(See 1. 4. 3. 24)

[Path 属性](#) (See 返回指定文件、文件夹或驱动器的路径。
1. 4. 3. 25)

[Read 方法](#) (See 从 **TextStream** 文件中读取指定数量的字符并返回结果字符串。
1. 4. 1. 36)

[ReadAll 方法](#) (See 读取整个 **TextStream** 文件并返回结果字符串。
1. 4. 1. 37)

[ReadLine 方法](#) (See 从 **TextStream** 文件中读取整行（在 *newline* 字符前，但不包括它）并返回结果字符串。
1. 4. 1. 38)

[Remove 方法](#) (See 从 **Dictionary** 对象中删除一对键和项目。
1. 4. 1. 39)

[RemoveAll 方法](#) (See 删除 **Dictionary** 对象中的所有键和项目对。
1. 4. 1. 40)

[RootFolder 属性](#) (See 返回表示指定驱动器上根文件夹的 **Folder** 对象。
1. 4. 3. 26)

[SerialNumber 属性](#) 返回用于唯一标识驱动器卷的数字序列号。
(See 1. 4. 3. 27)

[ShareName 属性](#) (See 返回指定驱动器的网络共享名。
1. 4. 3. 28)

[ShortName 属性](#) (See 返回一些程序使用的短名称, 这些短名称由需要以前的 8.3 命名规范的程序使用。
1. 4. 3. 29)

[ShortPath 属性](#) (See 返回一些程序使用的短路径, 这些短路径名由需要以前的 8.3 命名规范的程序使用。
1. 4. 3. 30)

[Size 属性](#) (See 以字节为单位返回指定文件或文件夹的大小。
1. 4. 3. 31)

[Skip 方法](#) (See 在读取 **TextStream** 文件时掠过指定数目的字符。
1. 4. 1. 41)

[SkipLine 方法](#) (See 在读取 **TextStream** 文件时掠过下一行。
1. 4. 1. 42)

[SubFolders 属性](#) (See 返回由指定文件夹中包含的所有文件夹组成的 **Folders** 集合, 包括那些设置了隐藏和系统文件属性的文件夹。
1. 4. 3. 32)

[TextStream 对象](#) (See 便于对文件的顺序访问。
1. 4. 2. 9)

[TotalSize 属性](#) (See 以字节为单位返回驱动器或网络共享的总共空间大小。
1. 4. 3. 33)

[Type 属性](#) (See 返回关于文件或文件夹类型的信息。
1. 4. 3. 34)

[VolumeName 属性](#) (See 设置或返回指定驱动器的卷号。
1. 4. 3. 35)

[Write 方法](#) (See 向 **TextStream** 文件中写入指定字符串。
1. 4. 1. 43)

[WriteBlankLines 方法](#) 向 **TextStream** 文件中写入指定数量的
newline 字符。
(See 1. 4. 1. 44)

[WriteLine 方法](#) (See 向 **TextStream** 文件中写入指定字符串和
newline 字符。
1. 4. 1. 45)

语言元素 描述

[Add 方法](#) 向 **Dictionary** 对象中添加一对键和项目。
([Dictionary](#)) (See
1. 4. 1. 1)

[Add 方法](#) 向 **Folders** 集合中添加新的 **Folder**。
([Folders](#)) (See
1. 4. 1. 2)

[BuildPath 方法](#) (See 在已有路径后追加一个名称。

1. 4. 1. 3)

[Close 方法](#) (See 关闭打开的 **TextStream** 文件。

1. 4. 1. 4)

[Copy 方法](#) (See 将指定文件或文件夹从一个位置复制到另一个位置。

1. 4. 1. 5)

[CopyFile 方法](#) (See 从一个位置向另一个位置复制一个或多个文件。

1. 4. 1. 6)

[CopyFolder 方法](#) (See 从一个位置向另一个位置复制整个文件夹。

1. 4. 1. 7)

[CreateFolder 方法](#) 创建文件夹。

(See 1. 4. 1. 8)

[CreateTextFile 方法](#) 创建指定的文件名并返回可用于读写该文件的
(See 1. 4. 1. 9) **TextStream** 对象。

[Delete 方法](#) (See 删除指定文件或文件夹。

1. 4. 1. 10)

[DeleteFile 方法](#) (See 删除指定文件。

1. 4. 1. 11)

[DeleteFolder 方法](#) 删除指定文件夹及其内容。

(See 1. 4. 1. 12)

[DriveExists 方法](#) (See 如果指定驱动器存在则返回 **True** ; 如果不存在
1. 4. 1. 13) 则返回 **False** 。

[Exists 方法](#) (See 如果 **Dictionary** 对象中存在指定键则返回
1. 4. 1. 14) **True** , 否则返回 **False** 。

[FileExists 方法](#) (See 如果指定文件存在则返回 **True** ; 否则返回
1. 4. 1. 15) **False** 。

[FolderExists 方法](#) 如果指定文件夹存在则返回 **True** ; 否则返回
(See 1. 4. 1. 16) **False** 。

[GetAbsolutePathName 方法](#) 根据所提供的路径说明返回完整明确的路径。
[方法](#) (See 1. 4. 1. 17)

[GetBaseName 方法](#) (See 根据提供的路径说明返回一个字符串, 包含文件
1. 4. 1. 18) 或文件夹的基本名称 (不包括任何文件扩展名)。

[GetDrive 方法](#) (See 返回相应于指定路径中驱动器的 **Drive** 对象。
1. 4. 1. 19)

[GetDriveName 方法](#) 根据指定路径返回包含驱动器名称的字符串。
(See 1. 4. 1. 20)

[GetExtensionName 方法](#) 返回一个字符串, 包含路径中最后成分的名称。
[法](#) (See 1. 4. 1. 21)

[GetFile 方法](#) (See 返回相应于指定路径中文件的 **File** 对象。
1. 4. 1. 22)

[GetFileName 方法](#) (See 返回指定路径中的最后文件名或文件夹名, 该路
1. 4. 1. 23) 径不能是驱动器说明。

[GetFileVersion 方法](#) 返回指定文件的版本号码。

(See 1. 4. 1. 24)

[GetFolder 方法](#) (See 返回相应于指定路径中文件夹的 **Folder** 对象。

1. 4. 1. 25)

[GetParentFolderName 方法](#) (See 返回一个字符串, 包含指定路径中最后文件或文件夹的父文件夹名。

[GetSpecialFolder 方法](#) (See 返回指定的特殊文件夹。

1. 4. 1. 27)

[GetTempName 方法](#) (See 返回一个随机生成的临时文件或文件夹名称。

1. 4. 1. 28)

[Items 方法](#) (See 返回包含 **Dictionary** 对象中所有项目的数组。

1. 4. 1. 29)

[Keys 方法](#) (See 返回包含了 **Dictionary** 对象中全部已有键的数组。

[Move 方法](#) (See 将指定文件或文件夹从一个位置移动到另一个位置。

[MoveFile 方法](#) (See 从一个位置向另一个位置移动一个或多个文件。

1. 4. 1. 32)

[MoveFolder 方法](#) (See 从一个位置向另一个位置移动一个或多个文件夹。

[OpenAsTextStream 方法](#) (See 打开指定文件并返回可用于读、写或追加该文件的 **TextStream** 对象。

[OpenTextFile 方法](#) (See 打开指定文件并返回可用于读、写或追加该文件的 **TextStream** 对象。

[Read 方法](#) (See 从 **TextStream** 文件中读取指定数量的字符并返回结果字符串。

[ReadAll 方法](#) (See 读取整个 **TextStream** 文件并返回结果字符串。

1. 4. 1. 37)

[ReadLine 方法](#) (See 从 **TextStream** 文件中读取整行 (在 `newline` 字符前, 但不包括它) 并返回结果字符串。

[Remove 方法](#) (See 从 **Dictionary** 对象中删除一个键和项目对。

1. 4. 1. 39)

[RemoveAll 方法](#) (See 删除 **Dictionary** 对象中的所有键和项目对。

1. 4. 1. 40)

[Skip 方法](#) (See 在读取 **TextStream** 文件时掠过指定数目的字符。

[SkipLine 方法](#) (See 在读取 **TextStream** 文件时掠过下一行。

1. 4. 1. 42)

[Write 方法](#) (See 向 **TextStream** 文件中写入指定字符串。

1. 4. 1. 43)

[WriteBlankLines 方法](#) (See 向 **TextStream** 文件中写入指定数量的 `newline` 字符。

[WriteLine 方法](#) (See 向 **TextStream** 文件中写入指定字符串和 `newline` 字符。

语言元素	描述

[Dictionary 对象](#) (See 存储数据键和项目对的对象。

1. 4. 2. 1)

[Drive 对象](#) (See 提供了对特定磁盘驱动器或网络共享的属性的访问。

1. 4. 2. 3)

[Drives 集合](#) (See 包含所有可用驱动器的只读集合。

1. 4. 2. 3)

[File 对象](#) (See 提供了对文件所有属性的访问。

1. 4. 2. 4)

[Files 集合](#) (See 文件夹中所有 **File** 对象的集合。

1. 4. 2. 5)

[FileSystemObject 对象](#) 提供了对计算机文件系统的访问。

(See 1. 4. 2. 6)

[Folder 对象](#) (See 提供了对文件夹所有属性的访问。

1. 4. 2. 7)

[Folders 集合](#) (See 包含在一个 **Folder** 对象中的所有 **Folder** 对象的集合。

1. 4. 2. 8)

[TextStream 对象](#) (See 便于对文件的顺序访问。

1. 4. 2. 9)

语言元素	描述

[AtEndOfLine 属性](#) 如果在 **TextStream** 文件中文件指针紧接在 end-of-line 标志之前则返回 **True** ; 否则返回 **False**。

(See 1. 4. 3. 1)

[AtEndOfStream 属性](#) 如果在 **TextStream** 文件中文件指针到达文件尾则返回 **True** ; 否则返回 **False** 。

(See 1. 4. 3. 2)

[Attributes 属性](#) (See 设置或返回文件或文件夹的属性。

1. 4. 3. 3)

[AvailableSpace 属性](#) 向用户返回指定驱动器或网络共享上的可用空间大小。

(See 1. 4. 3. 4)

[Column 属性](#) (See 返回 **TextStream** 文件中当前字符所在的列号。

1. 4. 3. 5)

[CompareMode 属性](#) 设置和返回在 **Dictionary** 对象中对字符串键进行比较的比较方式。

(See 1. 4. 3. 6)

[Count 属性](#) (See 返回集合或 **Dictionary** 对象中的项目数。

1. 4. 3. 7)

[DateCreated 属性](#) 返回指定文件或文件夹创建的日期和时间。

(See 1. 4. 3. 8)

[DateLastAccessed 属性](#) 返回指定文件或文件夹最后一次访问的日期和时间。

(See 1. 4. 3. 9)

[DateLastModified 属性](#) 返回指定文件或文件夹最后一次修改的日期和时间。(See 1.4.3.10)

[Drive 属性](#) (See 返回指定文件或文件夹所在驱动器的驱动器号。1.4.3.11)

[DriveLetter 属性](#) 返回本地物理驱动器或网络共享的驱动器号。(See 1.4.3.12)

[Drives 属性](#) (See 返回一个 **Drives** 集合，由所有本地机器上可用的 **Drive** 对象组成。1.4.3.13)

[DriveType 属性](#) (See 返回说明指定驱动器类型的值。1.4.3.14)

[Files 属性](#) (See 返回一个 **Files** 集合，由所有包含在指定文件夹中的 **File** 对象组成，包括那些设置了隐藏和系统文件属性的文件。1.4.3.15)

[FileSystem 属性](#) (See 返回指定驱动器所使用的文件系统类型。1.4.3.16)

[FreeSpace 属性](#) (See 向用户返回指定驱动器或网络共享上可用空间的大小。1.4.3.17)

[IsReady 属性](#) (See 如果指定驱动器已就绪则返回 **True**；否则返回 **False**。1.4.3.18)

[IsRootFolder 属性](#) 如果指定文件夹是根文件夹则返回 **True**；否则返回 **False**。(See 1.4.3.19)

[Item 属性](#) (See 根据 **Dictionary** 对象中指定的 *key* 设置或返回 *item*。1.4.3.20)

[Key 属性](#) (See 在 **Dictionary** 对象中设置 *key*。1.4.3.21)

[Line 属性](#) (See 返回 **TextStream** 文件中的当前行号。1.4.3.22)

[Name 属性](#) (See 设置或返回指定文件或文件夹的名称。1.4.3.23)

[ParentFolder 属性](#) 返回指定文件或文件夹的父文件夹对象。(See 1.4.3.24)

[Path 属性](#) (See 返回指定文件、文件夹或驱动器的路径。1.4.3.25)

[RootFolder 属性](#) (See 返回表示指定驱动器上根文件夹的 **Folder** 对象。1.4.3.26)

[SerialNumber 属性](#) 返回用于唯一标识驱动器卷的数字序列号。(See 1.4.3.27)

[ShareName 属性](#) (See 返回指定驱动器的网络共享名。1.4.3.28)

[ShortName 属性](#) (See 返回一些程序使用的短名称，这些短名由需要

1. 4. 3. 29) 以前的 8.3 命名规范的程序使用。

[ShortPath 属性](#) (See 返回一些程序使用的短路径, 这些短路径由需要以前的 8.3 文件名规范的程序使用。

[Size 属性](#) (See 以字节为单位返回指定文件或文件夹的大小。
1. 4. 3. 31)

[SubFolders 属性](#) (See 返回由指定文件夹中包含的所有文件夹组成的
1. 4. 3. 32) **Folders** 集合, 包括那些设置了隐藏和系统文件属性的文件夹。

[TotalSize 属性](#) (See 以字节为单位返回驱动器或网络共享的总共空间大小。
1. 4. 3. 33)

[Type 属性](#) (See 返回关于文件或文件夹类型的信息。
1. 4. 3. 34)

[VolumeName 属性](#) (See 设置或返回指定驱动器的卷号。
1. 4. 3. 35)

1.4.1 脚本运行时方法

描述	语言要素
向 Dictionary 对象增加一个键值和项目对。	Add 方法 (Dictionary) (See 1.4.1.1)
向 Folders 集合增加一个新 Folder。	Add 方法 (Folders) (See 1.4.1.2)
向一个已有的路径附加名称。	BuildPath 方法 (See 1.4.1.3)
关闭一个打开的 TextStream 文件。	Close 方法 (See 1.4.1.4)
从某位置复制指定的文件或文件夹到另一位置。	Copy 方法 (See 1.4.1.5)
从某位置复制一个或多个文件到另一位置。	CopyFile 方法 (See 1.4.1.6)
从某一位置复制一个文件夹到另一位置。	CopyFolder 方法 (See 1.4.1.7)
创建一个文件夹。	CreateFolder 方法 (See 1.4.1.8)
创建一个指定的文件名称, 并返回能用来读写该文件的 TextStream 对象。	CreateTextFile 方法 (See 1.4.1.9)
删除一个指定的文件或文件夹。	Delete 方法 (See 1.4.1.10)
删除一个指定的文件。	DeleteFile 方法 (See 1.4.1.11)
删除一个指定的文件夹及其内容。	DeleteFolder 方法 (See 1.4.1.12)
如果指定的驱动器存在, 返回 True; 如果不存在, 返回 False。	DriveExists 方法 (See 1.4.1.13)
如果在 Dictionary 对象中存在指定的键值, 返回 True, 如果不存在, 返回 False。	Exists 方法 (See 1.4.1.14)
如果指定的文件存在返回 True; 否则返回 False。	FileExists 方法 (See 1.4.1.15)
如果指定的文件夹存在返回 True; 否则返回	FolderExists 方法 (See 1.4.1.16)

False。	
根据给出的路径规范中返回一个完整明确的路径。	GetAbsolutePathName 方法 (See 1.4.1.17)
对于给出的路径规范，返回一个包含其中文件（没有文件扩展名）或文件夹基本名称的字符串。	GetBaseName 方法 (See 1.4.1.18)
返回一个与指定路径中的驱动器相应的 Drive 对象。	GetDrive 方法 (See 1.4.1.19)
返回一个字符串，该字符串包含指定路径的驱动器的名称。	GetDriveName 方法 (See 1.4.1.20)
返回一个字符串，该字符串包含一个路径中末尾组件的扩展名。	GetExtensionName 方法 (See 1.4.1.21)
返回一个与指定路径中文件相应的 File 对象。	GetFile 方法 (See 1.4.1.22)
返回指定路径末尾的文件名称或文件夹，该指定路径不是驱动器规范的一部分。	GetFileName 方法 (See 1.4.1.22)
返回指定文件的版本号。	GetFileVersion 方法 (See 1.4.1.24)
返回一个与指定路径中的文件夹相应的 Folder 对象。	GetFolder 方法 (See 1.4.1.25)
返回一个字符串，该字符串包含指定路径中末尾文件或文件夹的上一级文件夹的名称。	GetParentFolderName 方法 (See 1.4.1.26)
返回指定的特殊文件夹。	GetSpecialFolder 方法 (See 1.4.1.27)
返回一个随机产生的临时文件或文件夹的名称。	GetTempName 方法 (See 1.4.1.28)
返回一个数组，该数组包含 Dictionary 对象中的所有项。	Items 方法 (See 1.4.1.29)
返回一个数组，该数组包含 Dictionary 对象中现有的所有键值。	Keys 方法 (See 1.4.1.30)
将指定文件或文件夹从某位置移动到另一位置。	Move 方法 (See 1.4.1.31)
将一个或多个文件从某位置移动到另一位置。	MoveFile 方法 (See 1.4.1.32)
将一个或多个文件夹从某位置移动到另一位置。	MoveFolder 方法 (See 1.4.1.33)
打开指定的文件，并返回一个能用于读写或追加到该文件的 TextStream 对象。	OpenAsTextStream 方法 (See 1.4.1.34)
打开指定的文件并返回一个能用于读写或追加到该文件的 TextStream 对象。	OpenTextFile 方法 (See 1.4.1.35)
从 TextStream 文件读取指定数量的字符，并返回结果字符串。	Read 方法 (See 1.4.1.36)
读取一个完整的 TextStream 文件并返回结果字符串。	ReadAll 方法 (See 1.4.1.37)
从一个 TextStream 文件读一整行（读到但不包含换行字符），并返回结果字符串。	ReadLine 方法 (See 1.4.1.38)
从一个 Dictionary 对象删除一个键值、项目对。	Remove 方法 (See 1.4.1.39)

从一个 Dictionary 对象删除所有的键值、项目对。	RemoveAll 方法 (See 1.4.1.40)
当读取一个 TextStream 文件时跳过指定数量的字符。	Skip 方法 (See 1.4.1.41)
读 TextStream 文件时跳过下一行。	SkipLine 方法 (See 1.4.1.42)
将指定的字符串写入 TextStream 文件。	Write 方法 (See 1.4.1.43)
将指定数量的换行字符写入 TextStream 文件。	WriteBlankLines 方法 (See 1.4.1.44)
将指定的字符串和换行字符写入 TextStream 文件。	WriteLine 方法 (See 1.4.1.45)

1.4.1.1 Add 方法 (Dictionary)

向 **Dictionary** 对象中添加一个主键条目对。

`object.Add (key, item)`

参数

`object`

必选项。总是一个 **Dictionary** 对象的名称。

`key`

必选项。与被添加的 *item* 相关联的 *key*。

`item`

必选项。与被添加的 *key* 相关联的 *item*。

说明

如果 *key* 已经存在，那么将导致一个错误。
下面这个例子说明了 **Add** 方法的用法。

```
var d;

d = new
ActiveXObject("Scripting.Dictionary");

d.Add("a", "Athens");
```

```
d.Add("b", "Belgrade");
```

```
d.Add("c", "Cairo");
```

请参阅

[Add 方法 \(Folders\)](#)(See 1.4.1.2) | [Exists 方法](#)(See 1.4.1.14) | [Items 方法](#)(See 1.4.1.29) | [Keys 方法](#)(See 1.4.1.30) | [Remove 方法](#)(See 1.4.1.39) | [RemoveAll 方法](#)(See 1.4.1.40)应用于: [Dictionary 对象](#)(See 1.4.2.1)

1.4.1.2 Add 方法 (Folders)

向 **Folders** 集合中添加新的 **Folder** 。

```
object.Add (folderName)
```

参数

object

必选项。 应为 **Folders** 集合的名称。

folderName

必选项。 要添加的新 **Folder** 的名称。

说明

下面的例子说明了使用 **Add** 方法创建新文件夹的方法。

```
function AddNewFolder(path, folderName)
{
    var fso, f, fc, nf;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFolder(path);
    fc = f.SubFolders;
```



```

    if (folderName != "" )
        nf = fc.Add(folderName);
    else
        nf = fc.Add("New Folder");
}

```

如果 *folderName* 已经存在了，则会出错。

请参阅

[Add 方法 \(Dictionary\)](#)(See 1.4.1.1)应用于: [Folders 集合](#)(See 1.4.2.8)

1.4.1.3 BuildPath 方法

在已存在路径后追加名称。

object.**BuildPath**(*path*, *name*)

参数

object

必选项。 应为 `FileSystemObject` 的名称。

path

必选项。 将被追加 *name* 的并存在的路径。 路径可以是绝对的或相对的，同时不需要指定已有文件夹。

name

必选项。 被追加到已存在的 *path* 后的名称。

说明

如果需要，**BuildPath** 方法会在已有路径和名称之间另外再插入一个路径分隔符。下面的例子说明了 **BuildPath** 方法的用法。

```

function GetBuildPath(path)
{
    var fso, newpath;

```

```

        fso                                =                                new
ActiveXObject("Scripting.FileSystemObject");

        newpath        =        fso.BuildPath(path,
        "New    Folder");

        return(newpath);
    }

```

请参阅

[GetAbsolutePathName 方法](#) (See 1.4.1.17) | [GetBaseName 方法](#) (See 1.4.1.18) | [GetDriveName 方法](#) (See 1.4.1.20) | [GetExtensionName 方法](#) (See 1.4.1.21) | [GetFileName 方法](#) (See 1.4.1.23) | [GetParentFolderName 方法](#) (See 1.4.1.26) | [GetTempName 方法](#) (See 1.4.1.28) 应用于: [FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.4 Close 方法

关闭打开的 **TextStream** 文件。

```
object.Close( );
```

object 通常为 **TextStream** 对象的名字。

说明

下面的例子说明了 **Close** 方法的用法。

```

var fso;

fso                                =                                new
ActiveXObject("Scripting.FileSystemObject");

a                                =

fso.CreateTextFile("c:\\testfile.txt",
true);

```

```
a.WriteLine("This is a test.");
a.Close();
```

1.4.1.5 Copy 方法

将指定文件或文件夹从一个位置复制到另一位置。

```
object.Copy( destination[, overwrite] );
```

参数

object

必选项。 应为 **File** 或 **Folder** 对象的名称。

destination

必选项。 复制文件或文件夹的目的位置。 不允许通配字符。

overwrite

可选项。 Boolean 值，如果要覆盖已有文件或文件夹，则为 **True**（默认）；否则，则为 **False**。

说明

Copy 方法对单个 **File** 或 **Folder** 所产生的结果和使用 **FileSystemObject.CopyFile** 或 **FileSystemObject.CopyFolder** 所执行的操作结果一样，其中，后者把由 *object* 所引用的文件或文件夹作为参数传递。但是请注意，后两种替换方法能够复制多个文件或文件夹。

示例

下面的例子说明了 **Copy** 方法的用法。

```
var fso, f;

fso = new
ActiveXObject("Scripting.FileSystemObject");

f =
```

```
fso.CreateTextFile("c:\\testfile.txt",
true);
f.WriteLine("This is a test.");
f.Close();
f = fso.GetFile("c:\\testfile.txt");
f.Copy("c:\\windows\\desktop\\test2.txt
");
```

请参阅

[CopyFile 方法](#)(See 1.4.1.6) | [CopyFolder 方法](#)(See 1.4.1.7) | [Delete 方法](#)(See 1.4.1.10) | [Move 方法](#)(See 1.4.1.31) | [OpenAsTextStream 方法](#)(See 1.4.1.34) 应用于: [File 对象](#)(See 1.4.2.4) | [Folder 对象](#)(See 1.4.2.7)

1.4.1.6 CopyFile 方法

从一个位置向另一个位置复制一个或多个文件。

object.CopyFile (source, destination[, overwrite])

参数

object

必选项。 *object* 应为 **FileSystemObject** 的名称。

source

必选项。 指定文件字符串，可以包含通配字符来复制一个或多个文件。

destination

必选项。 目的字符串，文件将从 *source* 复制到这里。 不允许通配字符。

overwrite

可选项。 Boolean 值，指明是否覆盖已有文件。 如果为 **true**，则文件将被覆盖；如果为 **false**，则文件不会被覆盖。 默认的是 **true**。 注意，如

果 *destination* 设置了只读属性，那么无论 *overwrite* 的值是什么，**CopyFile** 都会失败。

说明

通配字符只能用在 *source* 参数中最后的路径成分中。 举例来说，可以使用：

```
fso = new  
ActiveXObject("Scripting.FileSystemObject");  
fso.CopyFile  
("c:\\mydocuments\\letters\\*.doc",  
"c:\\tempfolder\\")
```

但不能够使用：

```
fso = new  
ActiveXObject("Scripting.FileSystemObject");  
fso.CopyFile  
("c:\\mydocuments\\*\\R1???97.xls",  
"c:\\tempfolder")
```

如果 *source* 包含了通配字符或者 *destination* 以路径分隔符 (\) 结束，那么将假定 *destination* 是一个已有的文件夹，满足匹配的文件将复制到这个文件夹中。 否则，将假定 *destination* 是要创建的文件名。 无论哪种情况，如果复制的是单个文件，将会有三种可能。

- 如果 *destination* 不存在，那么 *source* 将被复制。 这是通常情况。
- 如果 *destination* 是已有文件，当 *overwrite* 为 **false** 时将会产生一个错误。 否则将会试图用 *source* 覆盖已有文件。
- 如果 *destination* 是目录，将会出错。

如果使用通配字符的 *source* 不能匹配任何文件，将会出错。 **CopyFile** 方法遇到第一个错误时终止。 出错后不会试图回滚或撤消出错前所做的修改。

请参阅

[Copy 方法](#) (See 1.4.1.5) | [CopyFolder 方法](#) (See 1.4.1.7) | [CreateTextFile 方法](#) (See 1.4.1.9) | [DeleteFile 方法](#) (See 1.4.1.11) | [MoveFile 方法](#) (See 1.4.1.32) 应用于：
[FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.7 CopyFolder 方法

将文件夹连同子文件夹从一个位置复制到另一个位置。

```
object.CopyFolder ( source, destination[, overwrite] );
```

参数

object

必选项。 应为 FileSystemObject 的名称。

source

必选项。 指定文件夹字符串，可以包含通配字符来复制一个或多个文件夹，

destination

必选项。 目的位置字符串， 文件夹和子文件夹将从 *source* 复制到这里。
不允许通配字符。

overwrite

可选项。 Boolean 值，指明是否覆盖已有文件夹。 如果为 **true**，则文件将被覆盖；如果为 **false**，则文件不会被覆盖。 默认的是 **true**。

说明

通配字符只能用在 *source* 参数中最后的路径成分中。 举例来说，可以使用：

```
fso = new
ActiveXObject("Scripting.FileSystemObject");
fso.CopyFolder
("c:\\mydocuments\\letters\\*",
"c:\\tempfolder\\")
```

但不能使用：

```
fso = new
ActiveXObject("Scripting.FileSystemObject");

fso.CopyFolder ("c:\\mydocuments\\*\\*",
"c:\\tempfolder\\")
```

如果 *source* 包含了通配字符或者 *destination* 以路径分隔符 (\) 结束，那么将假定 *destination* 是一个已有的文件夹，满足匹配的文件夹和子文件夹将复制到这个文件夹中。否则，将假定 *destination* 是要创建的文件夹名。无论哪种情况，如果复制的是单个文件夹，将会有四种可能。

- 如果 *destination* 不存在，将会复制 *source* 文件夹及其所有子文件夹。这是通常情况。
- 如果 *destination* 是已有的文件夹，将产生一个错误。
- 如果 *destination* 是一个目录，那么将试图复制文件夹及其所有内容。如果 *source* 中包含的一个文件在 *destination* 已经有了，那么当 *overwrite* 为 **false** 时将会出错。否则会试图用该文件覆盖已有文件。
- 如果 *destination* 是一个只读目录，同时 *overwrite* 为 **false**，那么在试图将已有的只读文件复制到目录中时将会出错。

如果使用了通配字符的 *source* 不能匹配任何文件夹，那么将产生错误。

CopyFolder 方法在遇到第一个错误时终止。出错后不试图回滚出错前所做的修改。

请参阅

[CopyFile 方法](#)(See 1.4.1.6) | [Copy 方法](#)(See 1.4.1.5) | [CreateFolder 方法](#)(See 1.4.1.8) | [DeleteFolder 方法](#)(See 1.4.1.12) | [MoveFolder 方法](#)(See 1.4.1.33) 应用于：
[FileSystemObject 对象](#)(See 1.4.2.6)

1.4.1.8 CreateFolder 方法

创建文件夹。

```
object.CreateFolder(foldername)
```

参数

object

必选项。应为 FileSystemObject 的名称。

foldername

必选项。 [字符串表达式](#)，标识所要创建的文件夹。

说明

如果指定的文件夹已经存在则出错。

下面的代码说明了如何使用 **CreateFolder** 方法来创建一个文件夹。

```
var fso = new
ActiveXObject("Scripting.FileSystemObject");

var a = fso.CreateFolder("c:\\new
folder");
```

请参阅

[CopyFolder 方法](#)(See 1.4.1.7) | [DeleteFolder 方法](#)(See 1.4.1.12) | [MoveFolder 方法](#)(See 1.4.1.33)应用于: [FileSystemObject 对象](#)(See 1.4.2.6)

1.4.1.9 CreateTextFile 方法

创建指定的文件名并返回一个 **TextStream** 对象，可以使用这个对象对文件进行读写。

```
object.CreateTextFile(filename[, overwrite[, unicode]])
```

参数

object

必选项。 应为 FileSystemObject 或 Folder 对象的名称。

filename

必选项。 指明所要创建文件的[字符串表达式](#)。

overwrite

可选项。 Boolean 值，指明能否覆盖已有文件。 如果文件可以覆盖，则值为 **true**，否则为 **false**。 如果忽略，则已有文件不能被覆盖。

unicode

可选项。 Boolean 值，指明文件是否以 Unicode 或 ASCII 文件方式创建。如果文件作为 Unicode 文件创建，则值为 **true**，如果作为 ASCII 文件创建，则为 **false**。如果忽略，则假定为 ASCII 文件。

说明

下面的代码说明了如何使用 **CreateTextFile** 方法来创建和打开一个文本文件。

```
var fso = new ActiveXObject("Scripting.FileSystemObject");

var a = fso.CreateTextFile("c:\\testfile.txt", true);

a.WriteLine("This is a test.");

a.Close();
```

如果 *overwrite* 参数是 **false**，或者没有提供这个参数，那么对于已有的 *filename* 将产生一个错误。

请参阅

[CreateFolder 方法](#) (See 1.4.1.8) | [OpenAsTextStream 方法](#) (See 1.4.1.34) | [OpenTextFile 方法](#) (See 1.4.1.35) 应用于: [FileSystemObject 对象](#) (See 1.4.2.6) | [Folder 对象](#) (See 1.4.2.7)

1.4.1.10 Delete 方法

删除指定的文件或文件夹。

```
object.Delete( force );
```

参数

object

必选项。 应为 **File** 或 **Folder** 对象的名称。

force

可选项。 Boolean 值，如果要删除设置了只读属性的文件或文件夹则为 **True** ； 否则为 **False** （默认）。

说明

如果指定的文件或文件夹不存在，那么会产生一个错误。

Delete 方法对于单个 **File** 或 **Folder** 产生的结果和使用 **FileSystemObject.DeleteFile** 或 **FileSystemObject.DeleteFolder** 所执行的操作结果一样。

Delete 方法对于包含内容和不包含内容的文件夹不做区分。删除指定的文件夹时不考虑是否包含了内容。

下面的例子说明了 **Delete** 方法的用法。

```
var fso, f;

fso = new
ActiveXObject("Scripting.FileSystemObject");

f =
fso.CreateTextFile("c:\\testfile.txt",
true);

f.WriteLine("This is a test.");

f.Close();

f = fso.GetFile("c:\\testfile.txt");

f.Delete();
```

请参阅

[Copy 方法](#) (See 1.4.1.5) | [DeleteFile 方法](#) (See 1.4.1.11) | [DeleteFolder 方法](#) (See 1.4.1.12) | [Move 方法](#) (See 1.4.1.31) | [OpenAsTextStream 方法](#) (See 1.4.1.34) 应用于：
[File 对象](#) (See 1.4.2.4) | [Folder 对象](#) (See 1.4.2.7)

1.4.1.11 DeleteFile 方法

删除指定文件。

```
object.DeleteFile ( filespec[, force] );
```

参数

object

必选项。 应为 FileSystemObject 的名称。

filespec

必选项。 要删除的文件的名称。 *filespec* 可以在最后的路径成分中包含通配字符。

force

可选项。 Boolean 值，如果要删除设置了只读属性的文件，则为 **true** ； 如果不删除则为 **false** （默认）。

说明

如果找不到匹配的文件则出错。 **DeleteFile** 方法在遇到第一个错误时终止。 出错后不试图回滚或撤消出错前做的修改。

下面的例子说明了 **DeleteFile** 方法的使用法。

```
function DeleteFile(filespec)
{
    var fso;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    fso.DeleteFile(filespec);
}
```

请参阅

[CopyFile 方法](#)(See 1.4.1.6) | [CreateTextFile 方法](#)(See 1.4.1.9) | [Delete 方法](#)(See 1.4.1.10) | [DeleteFolder 方法](#)(See 1.4.1.12) | [MoveFile 方法](#)(See 1.4.1.32)应用于：
[FileSystemObject 对象](#)(See 1.4.2.6)

1.4.1.12 DeleteFolder 方法

删除指定的文件夹及其内容。

```
object.DeleteFolder ( folderspec[, force] );
```

参数

object

必选项。 应为 FileSystemObject 的名称。

folderspec

必选项。 要删除的文件夹的名称。 *folderspec* 可以在最后的路径成分中包含通配字符。

force

可选项。 Boolean 值，如果要删除设置了只读属性的文件夹，则为 **true** ；否则为 **false** （默认）。

说明

DeleteFolder 方法不区分文件夹是否包含了内容。 删除指定的文件夹时不考虑其中是否有内容。

如果找不到匹配的文件夹则出错。 **DeleteFolder** 方法在遇到第一个错误时终止。出错后不试图回滚或撤消出错前做的修改。

下面的例子说明了 **DeleteFolder** 方法的用法。

```
function DeleteFolder(folderspec)
{
    var fso;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    fso.DeleteFolder(folderspec);
}
```

请参阅

[CopyFolder 方法](#) (See 1.4.1.7) | [CreateFolder 方法](#) (See 1.4.1.8) | [Delete 方法](#) (See 1.4.1.10) | [DeleteFile 方法](#) (See 1.4.1.11) | [MoveFolder 方法](#) (See 1.4.1.33) 应用于：
[FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.13 DriveExists 方法

如果指定的驱动器存在则返回 **True** ； 如果不存在则返回 **False** 。

`object.DriveExists(drivespec)`

参数

object

必选项。 应为 FileSystemObject 的名称。

drivespec

必选项。 驱动器号或完整的路径说明。

说明

对于可移动媒体的驱动器， 即使没有媒体 **DriveExists** 方法也返回 **true** 。 可以使用 **Drive** 对象的 **IsReady** 属性来决定驱动器是否就绪。

下面的例子说明了 **DriveExists** 方法的用法。

```
function ReportDriveStatus(drv)
{
    var fso, s = "";

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    if (fso.DriveExists(drv))
        s += "Drive " + drv + " exists.";
    else
        s += "Drive " + drv + " doesn't
```

```

exist.";

    return(s);
}

```

请参阅

[Drive Object](#)(See 1.4.2.2) | [Drives Collection](#)(See 1.4.2.3) | [FileExists 方法](#)(See 1.4.1.15) | [FolderExists 方法](#)(See 1.4.1.16) | [GetDrive 方法](#)(See 1.4.1.19) | [GetDriveName 方法](#)(See 1.4.1.20) | [IsReady 属性](#)(See 1.4.3.18)应用于: [FileSystemObject 对象](#)(See 1.4.2.6)

1.4.1.14 Exists 方法

如果 **Dictionary** 对象中存在所指定的主键则返回 **true**，否则返回 **false**。

object.Exists(key)

参数

object

必选项。总是一个 **Dictionary** 对象的名称。

key

必选项。需要在 **Dictionary** 对象中搜索的 *key* 值。

下面这个例子说明了 **Exists** 方法的用法。

```

function keyExists(k)
{
    var fso, s = "";
    d = new
ActiveXObject("Scripting.Dictionary");
    d.Add("a", "Athens");
    d.Add("b", "Belgrade");
    d.Add("c", "Cairo");
}

```

```

    if (d.Exists(k))
        s += "Specified key exists.";
    else
        s += "Specified key doesn't
exist.";
    return(s);
}

```

请参阅

[Add 方法 \(Dictionary\)](#)(See 1.4.1.1) | [Items 方法](#)(See 1.4.1.29) | [Keys 方法](#)(See 1.4.1.30) | [Remove 方法](#)(See 1.4.1.39) | [RemoveAll 方法](#)(See 1.4.1.40)应用于：
[Dictionary 对象](#)(See 1.4.2.1)

1.4.1.15 FileExists 方法

如果指定文件存在返回 **True** ;否则返回 **False** 。

object.FileExists(filespec)

参数

object

必选项。 应为 FileSystemObject 的名称。

filespec

必选项。 需要判定是否存在的文件的名称。 如果文件可能不在当前文件夹中, 那么必须提供完整的路径说明 (可以是绝对的或相对的)。

下面的例子说明了 **FileExists** 方法的用法。

```

function ReportFileStatus(filespec)
{
    var fso, s = filespec;

    fso = new

```

```

ActiveXObject("Scripting.FileSystemObject");

    if (fso.FileExists(filespec))
        s += " exists.";
    else
        s += " doesn't exist.";
    return(s);
}

```

请参阅

[DriveExists 方法](#)(See 1.4.1.13) | [FolderExists 方法](#)(See 1.4.1.16) | [GetFile 方法](#)(See 1.4.1.22) | [GetFileName 方法](#)(See 1.4.1.23)应用于: [FileSystemObject 对象](#)(See 1.4.2.6)

1.4.1.16 FolderExists 方法

如果指定的文件夹存在则返回 **True** ; 否则返回 **False** 。

object.FolderExists(folderspec)

参数

object

必选项。 应为 FileSystemObject 的名称。

folderspec

必选项。 需要判定是否存在的文件夹的名称。 如果文件夹可能不在当前文件夹中, 那么必须提供完整的路径说明(可以是绝对的或相对的)。

下面的例子说明了 **FileExists** 方法的使用。

```

function ReportFolderStatus(flldr)
{
    var fso, s = flldr;

```



```

        fso                                =                                new
ActiveXObject("Scripting.FileSystemObject");

        if (fso.FolderExists(fldr))
            s += " exists.";
        else
            s += " doesn't exist.";

        return(s);
    }

```

请参阅

[DriveExists 方法](#)(See 1.4.1.13) | [FileExists 方法](#)(See 1.4.1.15) | [GetFolder 方法](#)(See 1.4.1.25) | [GetParentFolderName 方法](#)(See 1.4.1.26) 应用于: [FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.17 GetAbsolutePathName 方法

根据提供的路径说明返回明确完整的路径。

`object.GetAbsolutePathName(pathspec)`

参数

`object`

必需 应为 `FileSystemObject` 的名称。

`pathspec`

必选项。 要变为明确完整路径的路径说明

说明

如果路径提供了从指定驱动器的根开始的完整的引用,那么它就是明确和完整的。如果路径指定的是映射驱动器的根文件夹,那么完整的路径将只能由一个路径分隔符 (\) 结束。

假设当前目录是 `c:\mydocuments\reports`,下表说明了 **GetAbsolutePathName** 方

法的操作。

pathspec	返回的路径
"c:"	"c:\mydocuments\reports"
"c:.."	"c:\mydocuments"
"c:\\"	"c:\"
"c:*..*\may97"	"c:\mydocuments\reports*..*\may97"
"region1"	"c:\mydocuments\reports\region1"
"c:\\..\\..\\mydocuments"	"c:\mydocuments"

下面的例子说明了 `GetAbsolutePathName` 方法的使用。

```
function ShowAbsolutePath(path)
{
    var fso, s = "";

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    s += fso.GetAbsolutePathName(path);

    return(s);
}
```

请参阅

[GetBaseName 方法](#)(See 1.4.1.18) | [GetDrive 方法](#)(See 1.4.1.19) | [GetDriveName 方法](#)(See 1.4.1.20) | [GetExtensionName 方法](#)(See 1.4.1.21) | [GetFile 方法](#)(See 1.4.1.22) | [GetFileName 方法](#)(See 1.4.1.23) | [GetFileVersion 方法](#)(See 1.4.1.24) | [GetFolder 方法](#)(See 1.4.1.25) | [GetParentFolderName 方法](#)(See 1.4.1.26) | [GetSpecialFolder 方法](#)(See 1.4.1.27) | [GetTempName 方法](#)(See 1.4.1.28)应用于: [FileSystemObject 对象](#)(See 1.4.2.6)

1.4.1.18 GetBaseName 方法

返回字符串，该字符串包含路径中最后成分中的基本名称，不包含文件扩展名。

`object.GetBaseName(path)`

参数

object

必选项。 应为 FileSystemObject 的名称。

path

必选项。 为将返回其最后成分中的基本名称指定路径。

说明

如果没有成分和 *path* 参数匹配, 那么 **GetBaseName** 方法将返回长度为零的字符串 ("")。

注意 **GetBaseName** 方法只作用于所提供的 *path* 字符串。 它不试图解析路径, 也不检查指定路径是否存在。

下面的例子说明了 **GetBaseName** 方法的用法。

```
function ShowBaseName(filespec)
{
    var fso, s = "";

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    s += fso.GetBaseName(filespec);

    return(s);
}
```

请参阅

[GetAbsolutePathName 方法](#) (See 1.4.1.17) | [GetDrive 方法](#) (See 1.4.1.19) | [GetDriveName 方法](#) (See 1.4.1.20) | [GetExtensionName 方法](#) (See 1.4.1.21) | [GetFile 方法](#) (See 1.4.1.22) | [GetFileName 方法](#) (See 1.4.1.23) | [GetFileVersion 方法](#) (See 1.4.1.24) | [GetFolder 方法](#) (See 1.4.1.25) | [GetParentFolderName 方法](#) (See 1.4.1.26) | [GetSpecialFolder 方法](#) (See 1.4.1.27) | [GetTempName 方法](#) (See 1.4.1.28) 应用于: [FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.19 GetDrive 方法

返回相应于指定路径中驱动器的 **Drive** 对象。

```
object.GetDrive ( drivespec );
```

参数

object

必选项。 应为 FileSystemObject 的名称。

drivespec

必选项。 *drivespec* 参数可以是驱动器号 (c) 、带冒号的驱动器号 (c:) 、带冒号和路径分隔符的驱动器号 (c:\) ，或者任意网络共享的说明 (\\computer2\share1) 。

说明

对于网络共享，需要进行检查以确保共享存在。

如果 *drivespec* 和可接受的形式不一致或是不存在，那么将产生错误。

要在普通的路径字符串上调用 **GetDrive** 方法，可以使用以下序列来获取适于作为 *drivespec* 的字符串。

```
DriveSpec =
```

```
GetDriveName (GetAbsolutePathName (Path))
```

下面的例子说明了 **GetDrive** 方法的用法。

```
function ShowFreeSpace (drvPath)
{
    var fso, d, s = "";

    fso = new
ActiveXObject ("Scripting.FileSystemObject");

    d =
fso.GetDrive (fso.GetDriveName (drvPath));
```

```

        s = "Drive " + drvPath.toUpperCase( )
+ " - ";

        s += d.VolumeName + "<br>";

        s += "Free Space: " +
d.FreeSpace/1024 + " Kbytes";

        return(s);
}

```

请参阅

[GetAbsolutePathName 方法](#) (See 1.4.1.17) | [GetBaseName 方法](#) (See 1.4.1.18) | [GetDriveName 方法](#) (See 1.4.1.20) | [GetExtensionName 方法](#) (See 1.4.1.21) | [GetFile 方法](#) (See 1.4.1.22) | [GetFileName 方法](#) (See 1.4.1.23) | [GetFileVersion 方法](#) (See 1.4.1.24) | [GetFolder 方法](#) (See 1.4.1.25) | [GetParentFolderName 方法](#) (See 1.4.1.26) | [GetSpecialFolder 方法](#) (See 1.4.1.27) | [GetTempName 方法](#) (See 1.4.1.28) 应用于：
[FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.20 GetDriveName 方法

根据指定路径返回包含驱动器名称的字符串。

object.GetDriveName(*path*)

参数

object

必选项。 应为 FileSystemObject 的名称。

path

必选项。 路径说明，将根据其中成分返回驱动器名称。

说明

如果无法确定驱动器，**GetDriveName** 方法将返回长度为零的字符串 ("")。

注意 **GetDriveName** 只作用于所提供的 *path* 字符串。 不会试图解析路径，也不会检查指定的路径是否存在。

下面的例子说明了 **GetDriveName** 方法的使用。

```
function GetDriveLetter(path)
{
    var fso, s = "";

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    s +=
fso.GetDrive(fso.GetDriveName(fso.GetAbsolutePathName(path)));

    return(s);
}
```

请参阅

[GetAbsolutePathName 方法](#) (See 1.4.1.17) | [GetBaseName 方法](#) (See 1.4.1.18) | [GetDrive 方法](#) (See 1.4.1.19) | [GetExtensionName 方法](#) (See 1.4.1.21) | [GetFile 方法](#) (See 1.4.1.22) | [GetFileName 方法](#) (See 1.4.1.23) | [GetFileVersion 方法](#) (See 1.4.1.24) | [GetFolder 方法](#) (See 1.4.1.25) | [GetParentFolderName 方法](#) (See 1.4.1.26) | [GetSpecialFolder 方法](#) (See 1.4.1.27) | [GetTempName 方法](#) (See 1.4.1.28) 应用于：[FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.21 GetExtensionName 方法

返回包含路径中最后成分扩展名的字符串。

object.GetExtensionName(*path*)

参数

object

必选项。 应为 FileSystemObject 的名称。

path

必选项。 路径说明，将根据其中的成分返回其扩展名。

说明

对于网络驱动器，根目录 (\) 将被认为是一个成分。

如果没有和 *path* 参数匹配的成分，那么 **GetExtensionName** 方法将返回长度为零的字符串 ("")。

下面的例子说明了 **GetExtensionName** 方法的使用法。

```
function ShowExtensionName(filespec)
{
    var fso, s = "";

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    s += fso.GetExtensionName(filespec);

    return(s);
}
```

请参阅

[GetAbsolutePathName 方法](#) (See 1.4.1.17) | [GetBaseName 方法](#) (See 1.4.1.18) | [GetDrive 方法](#) (See 1.4.1.19) | [GetDriveName 方法](#) (See 1.4.1.20) | [GetFile 方法](#) (See 1.4.1.22) | [GetFileName 方法](#) (See 1.4.1.23) | [GetFileVersion 方法](#) (See 1.4.1.24) | [GetFolder 方法](#) (See 1.4.1.25) | [GetParentFolderName 方法](#) (See 1.4.1.26) | [GetSpecialFolder 方法](#) (See 1.4.1.27) | [GetTempName 方法](#) (See 1.4.1.28) 应用于：
[FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.22 GetFile 方法

根据指定的路径中的文件返回相应的 **File** 对象。

`object.GetFile(filespec)`

参数

object

必选项。 应为 `FileSystemObject` 的名称。

filespec

必选项。 *filespec* 是指定文件的路径（绝对和或相对的）。

说明

如果指定文件不存在则出错。

下面的例子说明了 **GetFile** 方法的用法。

```
function ShowFileAccessInfo(filespec)
{
    var fso, f, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFile(filespec);
    s = f.Path.toUpperCase() + "<br>";
    s += "Created: " + f.DateCreated +
"<br>";
    s += "Last Accessed: " +
f.DateLastAccessed + "<br>";
    s += "Last Modified: " +
f.DateLastModified

    return(s);
}
```

请参阅

[GetAbsolutePathName 方法](#) (See 1.4.1.17) | [GetBaseName 方法](#) (See 1.4.1.18) | [GetDrive 方法](#) (See 1.4.1.19) | [GetDriveName 方法](#) (See 1.4.1.20) | [GetExtensionName 方法](#) (See 1.4.1.21) | [GetFileName 方法](#) (See 1.4.1.23) | [GetFileVersion 方法](#) (See 1.4.1.24) | [GetFolder 方法](#) (See 1.4.1.25) | [GetParentFolderName 方法](#) (See 1.4.1.26) |

[GetSpecialFolder 方法](#) (See 1.4.1.27) | [GetTempName 方法](#) (See 1.4.1.28) 应用于：
[FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.23 GetFileName 方法

返回指定路径的最后成分，但指定的路径不能只是驱动器说明。

object.**GetFileName** (*pathspec*)

参数

object

必选项。 应为 `FileSystemObject` 的名称。

pathspec

必选项。 指定文件的路径（绝对的或相对的）。

说明

如果 *pathspec* 没有以文件名成分结束，那么 **GetFileName** 方法将返回长度为零的字符串 ("")。

注意 **GetFileName** 方法只作用于所提供的路径字符串。 不会试图解析路径，也不会检查路径存在与否。

下面的例子说明了 **GetFileName** 方法的用法。

```
function ShowFileName(filespec)
{
    var fso, s = "";

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    s += fso.GetFileName(filespec);

    return(s);
}
```

请参阅

[GetAbsolutePathName 方法](#) (See 1.4.1.17) | [GetBaseName 方法](#) (See 1.4.1.18) | [GetDrive 方法](#) (See 1.4.1.19) | [GetDriveName 方法](#) (See 1.4.1.20) | [GetExtensionName 方法](#) (See 1.4.1.21) | [GetFile 方法](#) (See 1.4.1.22) | [GetFileVersion 方法](#) (See 1.4.1.24) | [GetFolder 方法](#) (See 1.4.1.25) | [GetParentFolderName 方法](#) (See 1.4.1.26) | [GetSpecialFolder 方法](#) (See 1.4.1.27) | [GetTempName 方法](#) (See 1.4.1.28) 应用于：
[FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.24 getFileVersion 方法

返回指定文件的版本号。

Object.**GetFileVersion**(*pathspec*)

参数

object

必选项。总是一个 **FileSystemObject** 的名字。

pathspec

必选项。指向特定文件的（绝对或相对）路径。

说明

如果 *pathspec* 并不以所命名的文件结束，或者文件并不包含版本信息，**GetFileVersion** 方法将返回长度为 0 的字符串("")。

注意 **GetFileVersion** 方法只处理所提供的路径字符串。它并不会尝试对路径进行解析，也不会检查指定的路径是否存在。

下面的例子说明了 **GetFileVersion** 方法的用法。

```
function ShowFileVersion(pathspect)
{
    var fso, s = "";
    fso = new
ActiveXObject("Scripting.FileSystemObje
ct");
```

```

s += fso.GetFileVersion(pathspec);

if (s == "")

    s = "无可用的版本信息。";

return(s);
}

```

要求

[版本 1](#) (See 2.64)

1.4.1.25 GetFolder 方法

根据指定路径中的文件夹返回相应的 **Folder** 对象。

`object.GetFolder(folderspec)`

参数

`object`

必选项。 应为 `FileSystemObject` 的名称。

`folderspec`

必选项。 *folderspec* 是指定文件夹的路径（绝对的或相对的）。

说明

如果指定的文件夹不存在则出错。

下面的例子说明了 **GetFolder** 方法的用法。

```

function ShowFolderList(folderspec)
{
    var fso, f, fc, s;

    fso = new
ActiveXObject("Scripting.FileSystemObje

```

```

ct");

    f = fso.GetFolder(folderspec);
    fc = new Enumerator(f.SubFolders);
    s = "";
    for (; !fc.atEnd(); fc.moveNext())
    {
        s += fc.item();
        s += "<br>";
    }

    return(s);
}

```

请参阅

[GetAbsolutePathName 方法](#) (See 1.4.1.17) | [GetBaseName 方法](#) (See 1.4.1.18) | [GetDrive 方法](#) (See 1.4.1.19) | [GetDriveName 方法](#) (See 1.4.1.20) | [GetExtensionName 方法](#) (See 1.4.1.21) | [GetFile 方法](#) (See 1.4.1.22) | [GetFileName 方法](#) (See 1.4.1.23) | [GetFileVersion 方法](#) (See 1.4.1.24) | [GetParentFolderName 方法](#) (See 1.4.1.26) | [GetSpecialFolder 方法](#) (See 1.4.1.27) | [GetTempName 方法](#) (See 1.4.1.28) 应用于：
[FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.26 GetParentFolderName 方法

根据指定路径中的最后成分返回包含其父文件夹名称的字符串。

`object.GetParentFolderName(path)`

参数

`object`

必选项。 应为 `FileSystemObject` 的名称。

`path`

必选项。 路径说明，将根据其中成分返回其父文件夹的名称。

说明

如果 *path* 参数中指定的成分没有父文件夹，那么 **GetParentFolderName** 方法将返回长度为零的字符串 ("")。

注意 **GetParentFolderName** 方法只作用于所提供的 *path* 字符串。不会试图解析路径，也不会检查指定的路径存在与否。

下面的例子说明了 **GetParentFolderName** 方法的使用法。

```
function ShowParentFolderName(filespec)
{
    var fso, s = "";

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    s +=
fso.GetParentFolderName(filespec);

    return(s);
}
```

请参阅

[GetAbsolutePathName 方法](#) (See 1.4.1.17) | [GetBaseName 方法](#) (See 1.4.1.18) | [GetDrive 方法](#) (See 1.4.1.19) | [GetDriveName 方法](#) (See 1.4.1.20) | [GetExtensionName 方法](#) (See 1.4.1.21) | [GetFile 方法](#) (See 1.4.1.22) | [GetFileName 方法](#) (See 1.4.1.23) | [GetFileVersion 方法](#) (See 1.4.1.24) | [GetFolder 方法](#) (See 1.4.1.25) | [GetSpecialFolder 方法](#) (See 1.4.1.27) | [GetTempName 方法](#) (See 1.4.1.28) 应用于: [FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.27 GetSpecialFolder 方法

返回指定的特殊文件夹对象。

object.**GetSpecialFolder**(*folderspec*)

参数

object

必选项。 应为 FileSystemObject 的名称。

folderspec

必选项。 要返回的特殊文件夹的名称。 可以为下面设置中显示的任意常数。

设置

folderspec 参数可以取以下的任意值：

常数	值	描述
WindowsFolder	0	Windows 文件夹，包含了由 Windows 操作系统安装的文件。
SystemFolder	1	包含库、字体，以及设备驱动程序的 System 文件夹
TemporaryFolder	2	用于存储临时文件的 Temp 文件夹。 这个路径可以在 TMP 环境变量中找到。

下面的例子说明了 **GetSpecialFolder** 方法的用法。

```
var fso, tempfile;

fso = new
ActiveXObject("Scripting.FileSystemObject");
```

```
function CreateTempFile()
{
    var tfolder, tfile, tname, fname,
    TemporaryFolder = 2;

    tfolder =

fso.GetSpecialFolder(TemporaryFolder);

    tname = fso.GetTempName();
```

```

        tfile
    }
    tfolder.CreateTextFile(tname);
    return(tfile);
}
tempfile = CreateTempFile();
tempfile.writeline("Hello World");
tempfile.close();

```

请参阅

[GetAbsolutePathName 方法](#) (See 1.4.1.17) | [GetBaseName 方法](#) (See 1.4.1.18) | [GetDrive 方法](#) (See 1.4.1.19) | [GetDriveName 方法](#) (See 1.4.1.20) | [GetExtensionName 方法](#) (See 1.4.1.21) | [GetFile 方法](#) (See 1.4.1.22) | [GetFileName 方法](#) (See 1.4.1.23) | [GetFileVersion 方法](#) (See 1.4.1.24) | [GetFolder 方法](#) (See 1.4.1.25) | [GetParentFolderName 方法](#) (See 1.4.1.26) | [GetTempName 方法](#) (See 1.4.1.28) 应用于: [FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.28 GetTempName 方法

返回一个随机产生的临时文件或文件夹名，有助于执行那些需要临时文件或文件夹的操作。

```
object.GetTempName ( );
```

可选的 *object* 应为 **FileSystemObject** 的名称。

说明

GetTempName 方法并不创建文件。 它只提供一个临时文件名，可以通过 **CreateTextFile** 来创建文件。

下面的例子说明了 **GetTempName** 方法的用法。

```

var fso, tempfile;

fso = new
ActiveXObject("Scripting.FileSystemObject");

```

```

function CreateTempFile()
{
    var  tfolder,  tfile,  tname,  fname,
TemporaryFolder = 2;

    tfolder                                     =
fso.GetSpecialFolder(TemporaryFolder);
    tname = fso.GetTempName();
    tfile                                     =
tfolder.CreateTextFile(tname);
    return(tfile);
}
tempfile = CreateTempFile();
tempfile.writeline("Hello World");
tempfile.close();

```

请参阅

[GetAbsolutePathName 方法](#) (See 1.4.1.17) | [GetBaseName 方法](#) (See 1.4.1.18) | [GetDrive 方法](#) (See 1.4.1.19) | [GetDriveName 方法](#) (See 1.4.1.20) | [GetExtensionName 方法](#) (See 1.4.1.21) | [GetFile 方法](#) (See 1.4.1.22) | [GetFileName 方法](#) (See 1.4.1.23) | [GetFileVersion 方法](#) (See 1.4.1.24) | [GetFolder 方法](#) (See 1.4.1.25) | [GetParentFolderName 方法](#) (See 1.4.1.26) | [GetSpecialFolder 方法](#) (See 1.4.1.27) 应用于: [FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.29 Items 方法

返回一个数组，其中包含了一个 **Dictionary** 对象中的所有条目。

`object.Items( )`

其中 *object* 总是一个 **Dictionary** 对象的名称。

说明

下面这段代码说明了 **Items** 方法的用法：


```

function ItemsDemo()
{
    var a, d, i, s;                                // 创建一
些变量。
    d = new
ActiveXObject("Scripting.Dictionary");
    d.Add ("a",
"Athens");                                         // 添加一些主键和条目。
    d.Add ("b", "Belgrade");
    d.Add ("c", "Cairo");
    a = (new
VBAArray(d.Items())) .toArray();                // 获取条目。
    s = "";
    for (i in a)                                    // 遍历该
dictionary。
    {
        s += a[i] + "<br>";
    }
    return(s);                                     // 返回
结果。
}

```

请参阅

[Add 方法 \(Dictionary\)](#)(See 1.4.1.1) | [Exists 方法](#)(See 1.4.1.14) | [Keys 方法](#)(See 1.4.1.30) | [Remove 方法](#)(See 1.4.1.39) | [RemoveAll 方法](#)(See 1.4.1.40)应用于:
[Dictionary 对象](#)(See 1.4.2.1)

1.4.1.30 Keys 方法

返回一个数组，其中包含了一个 **Dictionary** 对象中的全部现有的主键。

`object.Keys()`

其中 *object* 总是一个 **Dictionary** 对象的名称。

说明

下面这段代码说明了 **Keys** 方法的用法：

```
function KeysDemo()
{
    var a, d, i, s;                                // 创建一
    些变量。
    d = new
    ActiveXObject("Scripting.Dictionary");
    d.Add("a", "Athens");                          // 添加一些主键和条目。
    d.Add("b", "Belgrade");
    d.Add("c", "Cairo");
    a = (new
    VBArray(d.Keys())).toArray();                  // 获得主键。
    s = "";
    for (i in a)                                    // 遍历
    dictionary.
    {
        s += a[i] + " - " + d(a[i]) + "<br>";
    }
    return(s);                                      // 返回结
```

果。

}

请参阅

[Add 方法 \(Dictionary\)](#)(See 1.4.1.1) | [Exists 方法](#)(See 1.4.1.14) | [Items 方法](#)(See 1.4.1.29) | [Remove 方法](#)(See 1.4.1.39) | [RemoveAll 方法](#)(See 1.4.1.40)应用于：
[Dictionary 对象](#)(See 1.4.2.1)

1.4.1.31 Move 方法

将指定文件或文件夹从一个位置移动到另一个位置。

```
object.Move ( destination );
```

参数

object

必选项。 应为 **File** 或 **Folder** 对象的名称。

destination

必选项。 移动文件或文件夹的目的位置。 不允许通配字符。

说明

Move 方法对于单个 **File** 或 **Folder** 产生的结果和使用 **FileSystemObject.MoveFile** 或 **FileSystemObject.MoveFolder** 所执行的操作结果一样。但是请注意，后两种替换方法都能够移动多个文件或文件夹。

请参阅

[Copy 方法](#)(See 1.4.1.5) | [Delete 方法](#)(See 1.4.1.10) | [MoveFile 方法](#)(See 1.4.1.32) | [MoveFolder 方法](#)(See 1.4.1.33) | [OpenAsTextStream 方法](#)(See 1.4.1.34)应用于：
[File 对象](#)(See 1.4.2.4) | [Folder 对象](#)(See 1.4.2.7)

1.4.1.32 MoveFile 方法

从一个位置向另一个位置移动一个或多个文件。

```
object.MoveFile ( source, destination );
```

参数

object

必选项。 应为 `FileSystemObject` 的名称。

`source`

必选项。 所要移动的文件的路径。 `source` 参数的字符串只能在路径的最后成分中包含通配字符。

`destination`

必选项。 文件要移到的路径。 `destination` 参数不能包含通配字符。

说明

如果 `source` 包含了通配字符，或者 `destination` 以路径分隔符 (\) 结束，那么将假定 `destination` 指定了一个已有的文件夹，匹配的文件将移入其中。 否则，将假定 `destination` 为需要创建的目的文件名。 无论哪种情况，如果移动的是单个文件，将有三种可能：

- 如果 `destination` 不存在，那么那么文件将被移动。 这是普通情况。
- 如果 `destination` 是已经存在了的文件，那么将出错。
- 如果 `destination` 是目录，那么也将出错。

如果 `source` 中使用的通配字符无法匹配任何文件，那么也将出错。 **MoveFile** 方法在遇到第一个错误时终止。 出错后不会试图回滚任何出错前所做的修改。

重要 只有在操作系统支持时，才能通过这种方法在卷之间移动文件。

下面的例子说明了 **MoveFile** 方法的用法：

```
function MoveFile2Desktop(filespec)
{
    var fso;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    fso.MoveFile(filespec,
"c:\\windows\\desktop\\");
}
```

请参阅

[CopyFile 方法](#) (See 1.4.1.6) | [DeleteFile 方法](#) (See 1.4.1.11) | [GetFile 方法](#) (See 1.4.1.22) | [GetFileName 方法](#) (See 1.4.1.23) | [Move 方法](#) (See 1.4.1.31) | [MoveFolder 方法](#) (See 1.4.1.33) | [OpenTextFile 方法](#) (See 1.4.1.35) 应用于: [FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.33 MoveFolder 方法

从一个位置向另一个位置移动一个或多个文件夹。

```
object.MoveFolder ( source, destination );
```

参数

object

必选项。 应为 FileSystemObject 的名称。

source

必选项。 要移动的文件夹的路径。 *source* 参数的字符串只能在路径的最后成分中包含通配字符。

destination

必选项。 文件夹要移入的路径。 *destination* 参数不能包含通配字符。

说明

如果 *source* 包含了通配字符, 或者 *destination* 以路径分隔符 (\) 结束, 那么将假定 *destination* 指定的是已经存在了的文件夹, 匹配的文件将移入其中。 否则将假定 *destination* 是需要创建的目的文件夹的名称。 无论哪种情况, 如果移动的是单个文件夹, 将会有三种可能:

- 如果 *destination* 不存在, 那么文件夹将被移动。 这是普通情况。
- 如果 *destination* 是已有的文件, 那么将出错。
- 如果 *destination* 是目录, 那么也将出错。

如果 *source* 中使用的通配字符无法匹配任何文件夹, 那么也将出错。**MoveFolder** 方法在遇到第一个错误时终止。 出错后不会试图回滚出错前所做的修改。

重要 只有在操作系统支持时, 才能通过这个方法在卷之间移动文件夹。

下面的例子说明了 **MoveFolder** 方法的用法：

```
function MoveFldr2Desktop(fldrspec)
{
    var fso;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    fso.MoveFolder(fldrspec,
"c:\\windows\\desktop\\");
}
```

请参阅

[CopyFile 方法](#) (See 1.4.1.6) | [DeleteFile 方法](#) (See 1.4.1.11) | [GetFile 方法](#) (See 1.4.1.22) | [GetFileName 方法](#) (See 1.4.1.23) | [Move 方法](#) (See 1.4.1.31) | [MoveFile 方法](#) (See 1.4.1.32) | [OpenTextFile 方法](#) (See 1.4.1.35) 应用于：[FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.34 OpenAsTextStream 方法

打开指定的文件并返回一个 **TextStream** 对象，可以通过这个对象对文件进行读、写或追加。

```
object.OpenAsTextStream([iomode, [format]])
```

参数

object

必选项。 应为 **File** 对象的名称。

iomode

可选项。 指明输入/输出的模式。 可以是三个常数之一：**ForReading**、**ForWriting** 或 **ForAppending**。

format

可选项。使用三态值中的一个来指明打开文件的格式。如果忽略，文件将以 ASCII 格式打开。

设置

iomode 参数可以是下列设置中的任一种：

常数	值	描述
ForReading	1	以只读方式打开文件。不能写这个文件。
ForWriting	2	以写方式打开文件。如果存在同名的文件，那么它以前的内容将被覆盖。
ForAppending	8	打开文件并从文件末尾开始写。

format 参数可以是下列设置中的任一种：

常数	值	描述
TristateUseDefault	-2	使用系统默认值打开文件。
TristateTrue	-1	以 Unicode 方式打开文件。
TristateFalse	0	以 ASCII 方式打开文件。

说明

OpenAsTextStream 方法提供的功能和 FileSystemObject 的 OpenTextFile 方法一样。另外，OpenAsTextStream 方法可以用来写文件。

下面的代码说明了 **OpenAsTextStream** 方法的用法：

```
function TextStreamTest( )
{
    var fso, f, ts, s;

    var ForReading = 1, ForWriting = 2,
    ForAppending = 8;

    var TristateUseDefault = -2,
    TristateTrue = -1, TristateFalse = 0;

    fso = new
    ActiveXObject("Scripting.FileSystemObject");
```

```

        fso.CreateTextFile( "test1.txt" );

// 创建文件。

f = fso.GetFile("test1.txt");

ts = f.OpenAsTextStream(ForWriting,
TristateUseDefault);

ts.Write( "Hello World" );

ts.Close( );

ts = f.OpenAsTextStream(ForReading,
TristateUseDefault);

s = ts.ReadLine( );

ts.Close( );

return(s);
}

```

请参阅

[Copy 方法](#)(See 1.4.1.5) | [CreateTextFile 方法](#)(See 1.4.1.9) | [Delete 方法](#)(See 1.4.1.10) | [Move 方法](#)(See 1.4.1.31) | [OpenTextFile 方法](#)(See 1.4.1.35)应用于: [File 对象](#)(See 1.4.2.4)

1.4.1.35 OpenTextFile 方法

打开指定的文件并返回一个 **TextStream** 对象, 可以通过这个对象对文件进行读、写或追加。

object.OpenTextFile(filename[, iomode[, create[, format]])

参数

object

必选项。 *object* 应为 **FileSystemObject** 的名称。

filename

必选项。 指明要打开文件的[字符串表达式](#)。

iomode

可选项。 可以是三个常数之一： `ForReading` 、 `ForWriting` 或 `ForAppending` 。

create

可选项。 Boolean 值，指明当指定的 *filename* 不存在时是否创建新文件。 如果创建新文件则值为 **True** ， 如果不创建则为 **False** 。 如果忽略， 则不创建新文件。

format

可选项。 使用三态值中的一个来指明打开文件的格式。 如果忽略， 那么文件将以 ASCII 格式打开。

设置

iomode 参数可以是下列设置中的任一种：

常数	值	描述
<code>ForReading</code>	1	以只读方式打开文件。 不能写这个文件。
<code>ForWriting</code>	2	以写方式打开文件
<code>ForAppending</code>	8	打开文件并从文件末尾开始写。

format 参数可以是下列设置中的任一种：

值	描述
<code>TristateTrue</code>	以 Unicode 格式打开文件。
<code>TristateFalse</code>	以 ASCII 格式打开文件。
<code>TristateUseDefault</code>	使用系统默认值打开文件。

说明

下面的代码说明了如何使用 **OpenTextFile** 方法打开文件并追加文本：

```
var fs, a, ForAppending;
```

```
ForAppending = 8;
```

```
fs = new
```

```
ActiveXObject("Scripting.FileSystemObjec
```

```

ct");
a = fs.OpenTextFile("c:\\testfile.txt",
ForAppending, false);
...
a.Close();

```

请参阅

[CreateTextFile 方法](#) (See 1.4.1.9) | [OpenAsTextStream 方法](#) (See 1.4.1.34) 应用于：
[FileSystemObject 对象](#) (See 1.4.2.6)

1.4.1.36 Read 方法

从 **TextStream** 文件中读取指定数量的字符，并返回由此得到的字符串。

`object.Read(characters)`

参数

object

必选项。总是 **TextStream** 对象的名称。

characters

必选项。要从该文件读取的字符个数。

下面的示例演示类如何使用 **Read** 方法来从文件中读取六个字符 **Header**，并返回获得的字符串：

```

function GetHeader()
{
    var fso, f;
    var ForReading = 1, ForWriting = 2;
    fso = new
ActiveXObject("Scripting.FileSystemObj

```

```

ct");

    f                                     =
fso.OpenTextFile("c:\\testfile.txt",
ForWriting, true);

    f.Write("Header");

    f.Write("1234567890987654321");

    f.Close();

    f                                     =
fso.OpenTextFile("c:\\testfile.txt",
ForReading);

    return(f.Read(6));
}

```

请参阅

[ReadAll 方法](#)(See 1.4.1.37) | [ReadLine 方法](#)(See 1.4.1.38) | [Skip 方法](#)(See 1.4.1.41) | [SkipLine 方法](#)(See 1.4.1.42)应用于: [TextStream 对象](#)(See 1.4.2.9)

1.4.1.37 ReadAll 方法

读取 **TextStream** 文件的全部内容并返回由此得到的字符串。

object.ReadAll();

object 总是 **TextStream** 对象的名称。

说明

对大文件而言, 使用 **ReadAll** 方法将会耗费内存资源。可以使用其他技术来输入一个文件, 如逐行读取文件。

下面的示例演示了 **ReadAll** 方法的用法:

```

function GetEverything()
{
    var fso, f;

```

```

var ForReading = 1, ForWriting = 2;

fso = new
ActiveXObject("Scripting.FileSystemObject");

f =
fso.OpenTextFile("c:\\testfile.txt",
ForWriting, true);

f.Write("Header");
f.Write("1234567890987654321");
f.Close();

f =
fso.OpenTextFile("c:\\testfile.txt",
ForReading);

return (f.ReadAll());
}

```

请参阅

[Read 方法](#)(See 1.4.1.36) | [ReadLine 方法](#)(See 1.4.1.38) | [Skip 方法](#)(See 1.4.1.41) | [SkipLine 方法](#)(See 1.4.1.42)

应用于: [TextStream 对象](#)(See 1.4.2.9)

1.4.1.38 ReadLine 方法

从 **TextStream** 文件中读取一整行（一直到换行符，但不包括换行符），并返回由此得到的字符串。

object.ReadLine()

object 参数总是 **TextStream** 对象的名称。

说明

下面的示例演示了 **Line** 属性的用法：

```
function GetLine()
{
    var fso, f, r;
    var ForReading = 1, ForWriting = 2;
    fso = new
ActiveXObject("Scripting.FileSystemObject");
    f =
fso.OpenTextFile("c:\\testfile.txt",
ForWriting, true);
    f.WriteLine("Hello world!");
    f.WriteLine("JScript is fun");
    f.Close();
    f =
fso.OpenTextFile("c:\\testfile.txt",
ForReading);
    r = f.ReadLine();
    return(r);
}
```

请参阅

[Read 方法](#)(See 1.4.1.36) | [ReadAll 方法](#)(See 1.4.1.37) | [Skip 方法](#)(See 1.4.1.41) | [SkipLine 方法](#)(See 1.4.1.42)应用于: [TextStream 对象](#)(See 1.4.2.9)

1.4.1.39 Remove 方法

从一个 **Dictionary** 对象中删除一个主键，条目对。

`object.Remove(key)`

参数

`object`

必选项。总是一个 **Dictionary** 对象的名称。

`key`

必选项。 `key` 与要从 **Dictionary** 对象中删除的主键，条目对相关联。

说明

如果所指定的主键，条目对不存在，那么将导致一个错误。

下面这段代码说明了 **Remove** 方法的用法：

```
var a, d, i, s;                                // 创建一些变量。

d = new
ActiveXObject("Scripting.Dictionary");

d.Add ("a", "Athens");                          // 添加一些主键条目
对。

d.Add ("b", "Belgrade");

d.Add ("c", "Cairo");

...

d.Remove ("b");                                // 删除第二对主键条目
对。
```

请参阅

[Add 方法 \(Dictionary\)](#)(See 1.4.1.1) | [Exists 方法](#)(See 1.4.1.14) | [Items 方法](#)(See 1.4.1.29) | [Keys 方法](#)(See 1.4.1.30) | [RemoveAll 方法](#)(See 1.4.1.40)应用于：
[Dictionary 对象](#)(See 1.4.2.1)

1.4.1.40 RemoveAll 方法

RemoveAll 方法从一个 **Dictionary** 对象中删除所有的主键，条目对。

`object.RemoveAll( )`

其中 *object* 总是一个 **Dictionary** 对象的名称。

说明

下面这段代码说明了 **RemoveAll** 方法的用法：

```
var a, d, i;                                // 创建一些变量。

d                                     =                               new
ActiveXObject("Scripting.Dictionary");

d.Add ("a", "Athens");                      // 添加一些主键和条
目。
d.Add ("b", "Belgrade");
d.Add ("c", "Cairo");
...

d.RemoveAll( );      // 清除 dictionary。
```

请参阅

[Add 方法 \(Dictionary\)](#)(See 1.4.1.1) | [Exists 方法](#)(See 1.4.1.14) | [Items 方法](#)(See 1.4.1.29) | [Keys 方法](#)(See 1.4.1.30) | [Remove 方法](#)(See 1.4.1.39)应用于：[Dictionary 对象](#)(See 1.4.2.1)

1.4.1.41 Skip 方法

在读取 **TextStream** 文件时跳过指定个数的字符。

`object.Skip(characters)`

参数

object

必选项。总是某个 **TextStream** 对象的名称。

characters

必选项。在读取文件时要跳过的字符个数。

说明

被跳过的字符即被放弃。

下面的示例演示了 **Skip** 方法的用法：

```
function SkipDemo()  
{  
    var fso, f, r;  
    var ForReading = 1, ForWriting = 2;  
    fso = new  
ActiveXObject("Scripting.FileSystemObject")  
    f =  
fso.OpenTextFile("c:\\testfile.txt",  
ForWriting, true);  
    f.WriteLine("Hello world!");  
    f.WriteLine("JScript is fun");  
    f.Close();  
    f =  
fso.OpenTextFile("c:\\testfile.txt",  
ForReading);  
    f.Skip(6);  
    r = f.ReadLine();  
    return(r);  
}
```


请参阅

[Close 方法](#)(See 1.4.1.4) | [Read 方法](#)(See 1.4.1.36) | [ReadAll 方法](#)(See 1.4.1.37) | [ReadLine 方法](#)(See 1.4.1.38) | [SkipLine 方法](#)(See 1.4.1.42) | [Write 方法](#)(See 1.4.1.43) | [WriteLine 方法](#)(See 1.4.1.45) | [WriteBlankLines 方法](#)(See 1.4.1.44) 应用于：
[TextStream 对象](#)(See 1.4.2.9)

1.4.1.42 SkipLine 方法

在读取 **TextStream** 文件时跳过下一行。

`object.SkipLine( )`

object 总是一个 **TextStream** 对象的名称。

说明

下面的示例演示了 **SkipLine** 方法的用法：

```
function SkipLineDemo()
{
    var fso, f, r

    var ForReading = 1, ForWriting = 2;
    fso = new
ActiveXObject("Scripting.FileSystemObject")
    f =
fso.OpenTextFile("c:\\testfile.txt",
ForWriting, true)
    f.WriteLine("Hello world!");
    f.WriteLine("JScript is fun");
    f.Close();
    f =
```

```
fso.OpenTextFile("c:\\testfile.txt",
ForReading);

    f.SkipLine();

    r = f.ReadLine();

    return(r);
}
```

请参阅

[Read 方法](#)(See 1.4.1.36) | [ReadAll 方法](#)(See 1.4.1.37) | [ReadLine 方法](#)(See 1.4.1.38) | [Skip 方法](#)(See 1.4.1.41)应用于: [TextStream 对象](#)(See 1.4.2.9)

1.4.1.43 Write 方法

将给定的字符串写入到一个 **TextStream** 文件。

`object.Write(string)`

参数

object

必选项。总是一个 **TextStream** 对象的名称。

string

必选项。要写入文件的文本。

说明

给定的字符串在写入该文件时不会在字符串之间插入空格或字符。可以使用 **WriteLine** 方法来写入一个换行符或以换行符结束的字符串。

下面的示例演示了 **Write** 方法的用法:

```
function WriteDemo()

{

    var fso, f, r

    var ForReading = 1, ForWriting = 2;
```

```

        fso                                =                                new
ActiveXObject("Scripting.FileSystemObject")

        f                                =
fso.OpenTextFile("c:\\testfile.txt",
ForWriting, true)

        f.Write("Hello world!");

        f.Close();

        f                                =
fso.OpenTextFile("c:\\testfile.txt",
ForReading);

        r = f.ReadLine();

        return(r);
}

```

请参阅

[WriteBlankLines 方法](#) (See 1.4.1.44) | [WriteLine 方法](#) (See 1.4.1.45) 应用于：
[TextStream 对象](#) (See 1.4.2.9)

1.4.1.44 WriteBlankLines 方法

将指定数量的换行符写入到一个 **TextStream** 文件。

`object.WriteBlankLines(lines)`

参数

object

必选项。总是一个 **TextStream** 对象的名称。

lines

必选项。要写入该文件的换行符的个数。

说明

下面的示例演示了 **WriteBlankLines** 方法的使用法：

```
function WriteBlanksDemo()  
{  
    var fso, f, r;  
    var ForReading = 1, ForWriting = 2;  
    fso = new  
ActiveXObject("Scripting.FileSystemObject");  
    f =  
fso.OpenTextFile("c:\\testfile.txt",  
ForWriting, true);  
    f.Write("Hello world!");  
    f.WriteBlankLines(2);  
    f.Write("JScript is fun!");  
    f.Close();  
    f =  
fso.OpenTextFile("c:\\testfile.txt",  
ForReading);  
    r = f.ReadAll();  
    return(r);  
}
```

请参阅

[Write 方法](#)(See 1.4.1.43) | [WriteLine 方法](#)(See 1.4.1.45)应用于: [TextStream 对象](#)(See 1.4.2.9)

1.4.1.45 WriteLine 方法

向 **TextStream** 文件中写入给定的字符串和一个换行符。

`object.WriteLine([string])`

参数

object

必选项。总是一个 **TextStream** 对象的名称。

string

可选项。要写入该文件的文本。如果忽略该参数，则向该文件写入一个换行符。

说明

下面的示例演示了 **WriteLine** 方法的用法：

```
var fso, f;

fso = new
ActiveXObject("Scripting.FileSystemObject");

f =
fso.CreateTextFile("c:\\testfile.txt",
true);

f.WriteLine("This is a test.");

f.Close();
```

请参阅

[Write 方法](#)(See 1.4.1.43) | [WriteBlankLines 方法](#)(See 1.4.1.44)应用于: [TextStream 对象](#)(See 1.4.2.9)

1.4.2 脚本运行时对象

描述	语言要素
存储数据键值、项目对的对象。	Dictionary 对象 (See 1.4.2.1)
提供对特定磁盘驱动器或者网络共享的属性的访问。	Drive 对象 (See 1.4.2.2)
所有可用驱动器的只读集合。	Drives 集合 (See 1.4.2.3)
提供对一个文件的所有属性的访问。	File 对象 (See 1.4.2.4)
一个文件夹中所有 File 对象的集合。	Files 集合 (See 1.4.2.5)
提供对一个计算机的文件系统的访问。	FileSystemObject 对象 (See 1.4.2.6)
提供对一个文件夹的所有属性的访问。	Folder 对象 (See 1.4.2.7)
包含在一个 Folder 对象中的所有 Folder 对象的集合。	Folders 集合 (See 1.4.2.8)
方便对文件的顺序访问。	TextStream 对象 (See 1.4.2.9)

1.4.2.1 Dictionary 对象

存储数据键和项目对的对象。

```
y = new ActiveXObject("Scripting.Dictionary")
```

说明

Dictionary 对象等价于 PERL 联合数组。项目可以是数据的任何形式，并存储在数组中。每个项目都与一个具有唯一性的键相联。该键用于取得单个项目，并且通常是整数或字符串，但也可以是除数组以外的任何类型。

下面代码演示了如何创建 **Dictionary** 对象：

```
var y = new
ActiveXObject("Scripting.Dictionary");
y.add ("a", "test");
if (y.Exists("a"))
    document.write("true");
...
```

方法

[Add 方法 \(Dictionary\)](#)(See 1.4.1.1) | [Exists 方法](#)(See 1.4.1.14) | [Items 方法](#)(See 1.4.1.29) | [Keys 方法](#)(See 1.4.1.30) | [Remove 方法](#)(See 1.4.1.39) | [RemoveAll 方法](#)(See 1.4.1.40)

属性

[Count 属性](#)(See 1.4.3.7) | [Item 属性](#)(See 1.4.3.20) | [Key 属性](#)(See 1.4.3.21)

请参阅

[FileSystemObject 对象](#)(See 1.4.2.6) | [TextStream 对象](#)(See 1.4.2.9)

1.4.2.2 Drive 对象

提供对特定磁盘驱动器或网络共享属性的访问。

说明

下面的代码演示了如何用 **Drive** 对象访问驱动器属性：

```
function ShowFreeSpace (drvPath)
{
    var fso, d, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    d =
fso.GetDrive(fso.GetDriveName(drvPath));

    s = "Drive " + drvPath + " - ";
    s += d.VolumeName + "<br>";
    s += "Free Space: " +
d.FreeSpace/1024 + " Kbytes";

    return(s);
}
```

```
}
```

方法

Drive 对象没有方法。

属性

[AvailableSpace 属性](#)(See 1.4.3.4) | [DriveLetter 属性](#)(See 1.4.3.12) | [DriveType 属性](#)(See 1.4.3.14) | [FileSystem 属性](#)(See 1.4.3.16) | [FreeSpace 属性](#)(See 1.4.3.17) | [IsReady 属性](#)(See 1.4.3.18) | [Path 属性](#)(See 1.4.3.25) | [RootFolder 属性](#)(See 1.4.3.26) | [SerialNumber 属性](#)(See 1.4.3.27) | [ShareName 属性](#)(See 1.4.3.28) | [TotalSize 属性](#)(See 1.4.3.33) | [VolumeName 属性](#)(See 1.4.3.35)

请参阅

[Drives 集合](#)(See 1.4.2.3) | [File 对象](#)(See 1.4.2.4) | [Files 集合](#)(See 1.4.2.5) | [Folder 对象](#)(See 1.4.2.7) | [Folders 集合](#)(See 1.4.2.8) | [GetDrive 方法](#)(See 1.4.1.19)

1.4.2.3 Drives 集合

所有可用驱动器的只读集合。

说明

可移动媒体的驱动器不需要插入媒体，就可以出现在 **Drives** 集合中。
下面这个例子说明了如何使用 **Drives** 属性来获取 **Drives** 集合以及如何使用 **Enumerator** 对象来遍历该集合：

```
function ShowDriveList()
{
    var fso, s, n, e, x;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    e = new Enumerator(fso.Drives);
    s = "";

    for (; !e.atEnd(); e.moveNext())
    {
```



```
x = e.item();  
s = s + x.DriveLetter;  
s += " - ";  
if (x.DriveType == 3)  
    n = x.ShareName;  
else if (x.IsReady)  
    n = x.VolumeName;  
else  
    n = "[Drive not ready]";  
s +=    n + "<br>";  
}  
return(s);  
}
```

属性

[Count 属性](#)(See 1.4.3.7) | [Item 属性](#)(See 1.4.3.20)

请参阅

[Drive 对象](#)(See 1.4.2.2) | [Drives 属性](#)(See 1.4.3.13) | [File 对象](#)(See 1.4.2.4) | [Files 集合](#)(See 1.4.2.5) | [Folder 对象](#)(See 1.4.2.7) | [Folders 集合](#)(See 1.4.2.8)

1.4.2.4 File 对象

提供对文件所有属性的访问。

说明

下面的代码演示了如何获得 **File** 对象以及如何查看它的一个属性。

```
function ShowFileInfo(filespec)  
{
```

```

var fso, f, s;

fso = new
ActiveXObject("Scripting.FileSystemObject");

f = fso.GetFile(filespec);
s = f.DateCreated;

return(s);
}

```

方法

[Copy 方法](#) (See 1.4.1.5) | [Delete 方法](#) (See 1.4.1.10) | [Move 方法](#) (See 1.4.1.31) | [OpenAsTextStream 方法](#) (See 1.4.1.34)

属性

[Attributes 属性](#) (See 1.4.3.3) | [DateCreated 属性](#) (See 1.4.3.8) | [DateLastAccessed 属性](#) (See 1.4.3.9) | [DateLastModified 属性](#) (See 1.4.3.10) | [Drive 属性](#) (See 1.4.3.11) | [Name 属性](#) (See 1.4.3.23) | [ParentFolder 属性](#) (See 1.4.3.24) | [Path 属性](#) (See 1.4.3.25) | [ShortName 属性](#) (See 1.4.3.29) | [ShortPath 属性](#) (See 1.4.3.30) | [Size 属性](#) (See 1.4.3.31) | [Type 属性](#) (See 1.4.3.34)

请参阅

[Drive 对象](#) (See 1.4.2.2) | [Drives 集合](#) (See 1.4.2.3) | [Files 集合](#) (See 1.4.2.5) | [Folder 对象](#) (See 1.4.2.7) | [Folders 集合](#) (See 1.4.2.8)

1.4.2.5 Files 集合

一个文件夹中所有 **File** 对象的集合。

说明

下面这个例子说明了如何获得一个 **Files** 集合以及如何使用 **Enumerator** 对象和 **for** 语句来遍历该集合：

```

function ShowFolderFileList(folderspec)
{
    var fso, f, f1, fc, s;

```

```

        fso                                =                                new
ActiveXObject("Scripting.FileSystemObject");

        f = fso.GetFolder(folderspec);
        fc = new Enumerator(f.files);
        s = "";
        for (; !fc.atEnd(); fc.moveNext())
        {
            s += fc.item();
            s += "<br>";
        }
        return(s);
    }

```

方法

Files 集合没有方法。

属性

[Count 属性](#)(See 1.4.3.7) | [Item 属性](#)(See 1.4.3.20)

请参阅

[Drive 对象](#)(See 1.4.2.2) | [Drives 集合](#)(See 1.4.2.3) | [File 对象](#)(See 1.4.2.4) | [Folder 对象](#)(See 1.4.2.7) | [Folders 集合](#)(See 1.4.2.8)

1.4.2.6 FileSystemObject 对象

提供对计算机文件系统的访问。

```
y = new ActiveXObject("Scripting.FileSystemObject")
```

说明

下面的代码演示了如何用 **FileSystemObject** 返回可用于读或写的 **TextStream** 对象：

```

var fso = new
ActiveXObject("Scripting.FileSystemObject");

var a =
fso.CreateTextFile("c:\\testfile.txt",
true);

a.WriteLine("This is a test.");

a.Close();

```

在该例子代码中，**ActiveXObject** 对象被赋给 **FileSystemObject** (fso)。随后 **CreateTextFile** 方法创建文件 **TextStream** 对象 (a)，并用 **WriteLine** 方法将一行文本写入创建的文本文件。**Close** 方法刷新缓冲区并关闭该文件。

方法

[BuildPath 方法](#)(See 1.4.1.3) | [CopyFile 方法](#)(See 1.4.1.6) | [CopyFolder 方法](#)(See 1.4.1.7) | [CreateFolder 方法](#)(See 1.4.1.8) | [CreateTextFile 方法](#)(See 1.4.1.9) | [DeleteFile 方法](#)(See 1.4.1.11) | [DeleteFolder 方法](#)(See 1.4.1.12) | [DriveExists 方法](#)(See 1.4.1.13) | [FileExists 方法](#)(See 1.4.1.15) | [FolderExists 方法](#)(See 1.4.1.16) | [GetAbsolutePathName 方法](#)(See 1.4.1.17) | [GetBaseName 方法](#)(See 1.4.1.18) | [GetDrive 方法](#)(See 1.4.1.19) | [GetDriveName 方法](#)(See 1.4.1.20) | [GetExtensionName 方法](#)(See 1.4.1.21) | [GetFile 方法](#)(See 1.4.1.22) | [GetFileName 方法](#)(See 1.4.1.23) | [GetFolder 方法](#)(See 1.4.1.25) | [GetParentFolderName 方法](#)(See 1.4.1.26) | [GetSpecialFolder 方法](#)(See 1.4.1.27) | [GetTempName 方法](#)(See 1.4.1.28) | [MoveFile 方法](#)(See 1.4.1.32) | [MoveFolder 方法](#)(See 1.4.1.33) | [OpenTextFile 方法](#)(See 1.4.1.35)

属性

[Drives 属性](#)(See 1.4.3.13)

请参阅

[Dictionary 对象](#)(See 1.4.2.1) | [Drive 对象](#)(See 1.4.2.2) | [Drives 集合](#)(See 1.4.2.3) | [File 对象](#)(See 1.4.2.4) | [Files 集合](#)(See 1.4.2.5) | [Folder 对象](#)(See 1.4.2.7) | [Folders 集合](#)(See 1.4.2.8) | [TextStream 对象](#)(See 1.4.2.9)

1.4.2.7 Folder 对象

提供对文件夹的所有属性的访问。

说明

下面的代码演示了如何获得 **Folder** 对象并返回它的一个属性：

```
function ShowFolderInfo(folderspec)
{
    var fso, folder, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    folder = fso.GetFolder(folderspec);
    s = folder.DateCreated;

    return(s);
}
```

方法

[Copy 方法](#)(See 1.4.1.5) | [Delete 方法](#)(See 1.4.1.10) | [Move 方法](#)(See 1.4.1.31) | [OpenAsTextStream 方法](#)(See 1.4.1.34)

属性

[Attributes 属性](#)(See 1.4.3.3) | [DateCreated 属性](#)(See 1.4.3.8) | [DateLastAccessed 属性](#)(See 1.4.3.9) | [DateLastModified 属性](#)(See 1.4.3.10) | [Drive 属性](#)(See 1.4.3.11) | [Files 属性](#)(See 1.4.3.15) | [IsRootFolder 属性](#)(See 1.4.3.19) | [Name 属性](#)(See 1.4.3.23) | [ParentFolder 属性](#)(See 1.4.3.24) | [Path 属性](#)(See 1.4.3.25) | [ShortName 属性](#)(See 1.4.3.29) | [ShortPath 属性](#)(See 1.4.3.30) | [Size 属性](#)(See 1.4.3.31) | [SubFolders 属性](#)(See 1.4.3.32) | [Type 属性](#)(See 1.4.3.34)

请参阅

[Drive 对象](#)(See 1.4.2.2) | [Drives 集合](#)(See 1.4.2.3) | [File 对象](#)(See 1.4.2.4) | [Files 集合](#)(See 1.4.2.5) | [Folders 集合](#)(See 1.4.2.8)

1.4.2.8 Folders 集合

一个 **Folder** 对象中所包含的所有 **Folder** 对象的集合。

说明

下面这个例子说明了如何获得一个 **Folders** 集合以及如何使用 **Enumerator** 对象和 **for** 语句来遍历该集合：

```
function ShowFolderList(folderspec)
{
    var fso, f, fc, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFolder(folderspec);
    fc = new Enumerator(f.SubFolders);
    s = "";
    for (; !fc.atEnd(); fc.moveNext())
    {
        s += fc.item();
        s += "<br>";
    }

    return(s);
}
```

方法

[Add 方法 \(Folders\)](#)(See 1.4.1.2)

属性

[Count 属性](#)(See 1.4.3.7) | [Item 属性](#)(See 1.4.3.20)

请参阅

[Drive 对象](#)(See 1.4.2.2) | [Drives 集合](#)(See 1.4.2.3) | [File 对象](#)(See 1.4.2.4) | [Files 集合](#)(See 1.4.2.5) | [Folder 对象](#)(See 1.4.2.7) | [SubFolders 属性](#)(See 1.4.3.32)

1.4.2.9 TextStream 对象

方便对文件的顺序访问。

TextStream.{*property* | *method*()}

property 和 *method* 参数可以是与 **TextStream** 对象相连的任何属性和方法。请注意在实际使用时，**TextStream** 被从 **FileSystemObject** 返回的代表 **TextStream** 对象的变量占位符代替。

说明

在下面的代码中，a 是由 **FileSystemObject** 的 **CreateTextFile** 方法返回的 **TextStream** 对象：

```
var fso = new
ActiveXObject("Scripting.FileSystemObject");

var a =
fso.CreateTextFile("c:\\testfile.txt",
true);

a.WriteLine("这是一个测试。");

a.Close();
```

WriteLine 和 **Close** 是 **TextStream** 对象的两个方法。

方法

[Close 方法](#)(See 1.4.1.4) | [Read 方法](#)(See 1.4.1.36) | [ReadAll 方法](#)(See 1.4.1.37) | [ReadLine 方法](#)(See 1.4.1.38) | [Skip 方法](#)(See 1.4.1.41) | [SkipLine 方法](#)(See 1.4.1.42) | [Write 方法](#)(See 1.4.1.43) | [WriteBlankLines 方法](#)(See 1.4.1.44) | [WriteLine 方法](#)(See 1.4.1.45)

属性

[AtEndOfLine 属性](#)(See 1.4.3.1) | [AtEndOfStream 属性](#)(See 1.4.3.2) | [Column 属性](#)(See 1.4.3.5) | [Line 属性](#)(See 1.4.3.22)

请参阅

[Dictionary 对象](#)(See 1.4.2.1) | [FileSystemObject 对象](#)(See 1.4.2.6)

1.4.3 脚本运行时属性

描述	语言要素
在 TextStream 文件中，如果文件指针越过了行尾标志，返回 True ；否则返回 False 。	AtEndOfLine 属性 (See 1.4.3.1)
如果文件指针在 TextStream 文件的末尾，返回 True ；否则返回 False 。	AtEndOfStream 属性 (See 1.4.3.2)
设置或返回文件或者文件夹的属性。	Attributes 属性 (See 1.4.3.3)
给用户返回指定驱动器或网络共享上可用空间的总数。	AvailableSpace 属性 (See 1.4.3.4)
返回 TextStream 文件中当前字符位置的列号。	Column 属性 (See 1.4.3.5)
为 Dictionary 对象中的比较字符串键值设置并返回比较方式。	CompareMode 属性 (See 1.4.3.6)
返回集合或者 Dictionary 对象中项目的数量。	Count 属性 (See 1.4.3.7)
返回指定的文件或文件夹被创建的日期和时间。	DateCreated 属性 (See 1.4.3.8)
返回指定的文件或文件夹上一次被访问的日期和时间。	DateLastAccessed 属性 (See 1.4.3.9)
返回指定的文件或文件夹上次被修改的日期和时间。	DateLastModified 属性 (See 1.4.3.10)
返回指定文件或文件夹所在驱动器的驱动器号。	Drive 属性 (See 1.4.3.11)
返回本地驱动器或网络共享的驱动器号。	DriveLetter 属性 (See 1.4.3.12)
返回一个 Drives 集合，该集合包含了本地机上所有可用 Drive 对象。	Drives 属性 (See 1.4.3.13)
返回一个值，该值表示指定驱动器的类型。	DriveType 属性 (See 1.4.3.14)
返回一个 Files 集合，该集合包含了指定文件夹中包含的所有 File 对象（包括那些具有隐含和系统文件属性设置的对象）。	Files 属性 (See 1.4.3.15)
返回正用于指定驱动器的文件系统的类型。	FileSystem 属性 (See 1.4.3.16)
向用户返回指定的驱动器或网络共享上总的可用空间。	FreeSpace 属性 (See 1.4.3.17)
如果指定的驱动器已就绪，返回 True ；否则返回 False 。	IsReady 属性 (See 1.4.3.18)
如果指定的文件夹是根文件夹，返回 True ；否则返回 False 。	IsRootFolder 属性 (See 1.4.3.19)
为 Dictionary 对象中的指定的键值设置或返回一个项目。	Item 属性 (See 1.4.3.20)
在 Dictionary 对象中设置一个键值。	Key 属性 (See 1.4.3.21)
返回 TextStream 文件中当前的行号。	Line 属性 (See 1.4.3.22)
设置或返回指定文件或者文件夹的名称。	Name 属性 (See 1.4.3.23)

返回指定文件或文件夹上一级文件夹的 folder 对象。	ParentFolder 属性 (See 1.4.3.24)
返回指定文件、文件夹或者驱动器的路径。	Path 属性 (See 1.4.3.25)
返回一个 Folder 对象，该对象代表指定驱动器的根文件夹。	RootFolder 属性 (See 1.4.3.26)
返回用于唯一确定一个磁盘卷的十进制序号。	SerialNumber 属性 (See 1.4.3.27)
返回指定驱动器的网络共享名称。	ShareName 属性 (See 1.4.3.28)
返回程序所使用的需要以前的 8.3 文件命名约定的短名称。	ShortName 属性 (See 1.4.3.29)
返回程序所使用的需要以前的 8.3 文件命名约定的短路径。	ShortPath 属性 (See 1.4.3.30)
按字节返回指定文件或文件夹的大小。	Size 属性 (See 1.4.3.31)
返回一个 Folders 集合，该集合包含了指定文件夹中包含的所有文件夹（包括那些具有隐含和系统文件属性设置的文件夹）。	SubFolders 属性 (See 1.4.3.32)
按字节返回驱动器或网络共享的总空间。	TotalSize 属性 (See 1.4.3.33)
返回与文件或文件夹类型有关的信息。	Type 属性 (See 1.4.3.34)
设置或者返回指定驱动器的卷名。	VolumeName 属性 (See 1.4.3.35)

1.4.3.1 AtEndOfLine 属性

如果文件指针正好位于 **TextStream** 文件中的行尾符之前，则返回 **true**，否则返回 **false**。只读。

object.AtEndOfLine

object 总是 **TextStream** 对象的名称。

说明

AtEndOfLine 属性只能应用于打开用来读取的 **TextStream** 文件；否则将产生错误。

下面的代码演示了 **AtEndOfLine** 属性的用法：

```
function GetALine(filespec)
{
    var fso, a, s, ForReading;
    ForReading = 1, s = "";
    fso = new
```

```
ActiveXObject("Scripting.FileSystemObject");

a = fso.OpenTextFile(filespec,
ForReading, false);

while (!a.AtEndOfLine)
{
    s += a.Read(1);
}

a.Close();

return(s);
}
```

请参阅

[AtEndOfStream 属性](#) (See 1.4.3.2)

应用于: [TextStream 对象](#) (See 1.4.2.9)

1.4.3.2 AtEndOfStream 属性

如果文件指针正好位于 **TextStream** 文件中的结尾, 则返回 **true**, 否则返回 **false**。
只读。

object.AtEndOfStream

object 总是 **TextStream** 对象的名称。

说明

AtEndOfStream 属性只能应用于打开用来读取的 **TextStream** 文件; 否则将产生错误。

下面的代码演示了 **AtEndOfStream** 属性的用法:

```
function GetALine(filespec)
{
    var fso, f, s, ForReading;
```

```
    ForReading = 1, s = "";

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.OpenTextFile(filespec,
ForReading, false);

    while (!f.AtEndOfStream)
        s += f.ReadLine();

    f.Close();

    return(s);
}
```

请参阅

[AtEndOfLine 属性](#)(See 1.4.3.1)

应用于: [TextStream 对象](#)(See 1.4.2.9)

1.4.3.3 Attributes 属性

设置或返回文件或文件夹的属性。 根据不同属性为读/写或只读。

`object.Attributes` [= *newattributes*]

参数

object

必选项。 应为 **File** 或 **Folder** 对象的名称。

newattributes

可选项。 如果提供了这个部分, 那么 *newattributes* 将成为指定的 *object* 的新属性值。

设置

newattributes 参数可以是下列各值或者这些值的任意逻辑组合:

常数	值	描述
Normal	0	普通文件。 不设置属性。
ReadOnly	1	只读文件。 属性为读/写。
Hidden	2	隐藏文件。 属性为读/写。
System	4	系统文件。 属性为读/写。
Volume	8	磁盘驱动器卷标。 属性为只读。
Directory	16	文件夹或目录。 属性为只读。
Archive	32	文件在上次备份后已经修改。 属性为读/写。
Alias	64	链接或者快捷方式。 属性为只读。
Compressed	128	压缩文件。 属性为只读。

说明

下面的代码通过一个文件说明了 **Attributes** 属性的用法:

```
function ToggleArchiveBit(filespec)
{
    var fso, f, r, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFile(filespec)
    if (f.attributes && 32)
    {
        f.attributes = f.attributes - 32;
        s = "Archive bit is cleared.";
    }
    else
    {
```

```

        f.attributes = f.attributes + 32;

        s = "Archive bit is set.";

    }

    return(s);
}

```

请参阅

[DateCreated 属性](#) (See 1.4.3.8) | [DateLastAccessed 属性](#) (See 1.4.3.9) | [DateLastModified 属性](#) (See 1.4.3.10) | [Drive 属性](#) (See 1.4.3.11) | [Files 属性](#) (See 1.4.3.15) | [IsRootFolder 属性](#) (See 1.4.3.19) | [Name 属性](#) (See 1.4.3.23) | [ParentFolder 属性](#) (See 1.4.3.24) | [Path 属性](#) (See 1.4.3.25) | [ShortName 属性](#) (See 1.4.3.29) | [ShortPath 属性](#) (See 1.4.3.30) | [Size 属性](#) (See 1.4.3.31) | [SubFolders 属性](#) (See 1.4.3.32) | [Type 属性](#) (See 1.4.3.34)

应用于: [File 对象](#) (See 1.4.2.4) | [Folder 对象](#) (See 1.4.2.7)

1.4.3.4 AvailableSpace 属性

返回在所指定的驱动器或网络共享上可用的空间的大小。

`object.AvailableSpace`

其中 *object* 总是一个 **Drive** 对象。

说明

在典型情况下, 由 **AvailableSpace** 属性所返回的值与 **FreeSpace** 属性所返回的值是一样的。但是对于支持 *quotas* 的两个计算机系统之间返回值可能会有所不同。下面这段代码说明了 **AvailableSpace** 属性的用法:

```

function ShowAvailableSpace(drvPath)
{
    var fso, d, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    d =

```

```
fso.GetDrive(fso.GetDriveName(drvPath));
    s = "Drive " + drvPath.toUpperCase()
+ " - ";
    s += d.VolumeName + "<br>";
    s += "Available Space: " +
d.AvailableSpace/1024 + " Kbytes";
    return(s);
}
```

请参阅

[DriveLetter 属性](#)(See 1.4.3.12) | [DriveType 属性](#)(See 1.4.3.14) | [FileSystem 属性](#)(See 1.4.3.16) | [FreeSpace 属性](#)(See 1.4.3.17) | [IsReady 属性](#)(See 1.4.3.18) | [Path 属性](#)(See 1.4.3.25) | [RootFolder 属性](#)(See 1.4.3.26) | [SerialNumber 属性](#)(See 1.4.3.27) | [ShareName 属性](#)(See 1.4.3.28) | [TotalSize 属性](#)(See 1.4.3.33) | [VolumeName 属性](#)(See 1.4.3.35)

应用于: [Drive 对象](#)(See 1.4.2.2)

1.4.3.5 Column 属性

为只读属性，返回当前字符在 **TextStream** 文件中的列号。

object.Column

object 总是 **TextStream** 对象的名称。

说明

如果在已换行符之后，但在其他任何字符之前，**Column** 将等于 1。

下面的示例演示了 **Column** 属性的用法：

```
function GetColumn()
{
    var fso, f, m;
    var ForReading = 1, ForWriting = 2;
    fso = new
```

```

ActiveXObject("Scripting.FileSystemObject");

    f                                     =
fso.OpenTextFile("c:\\testfile.txt",
ForWriting, true);

    f.Write("Hello World!");

    f.Close();

    f                                     =
fso.OpenTextFile("c:\\testfile.txt",
ForReading);

    m = f.ReadLine();

    return(f.Column);
}

```

请参阅

[Line 属性](#) (See 1.4.3.22)

应用于: [TextStream 对象](#) (See 1.4.2.9)

1.4.3.6 CompareMode 属性

设置或者返回在 **Dictionary** 对象中进行字符串主键比较时所使用的比较模式。

object.CompareMode[= compare]

参数

object

必选项。总是一个 **Dictionary** 对象的名称。

compare

可选项。如果提供了此项，*compare* 就是一个代表比较模式的值。可以使用的值是 0（二进制）、1（文本）、2（数据库）。大于 2 的值可以用来表示使用特殊的区域设置 ID（LCID）而进行的比较。

说明

如果试图改变一个已经包含有数据的 **Dictionary** 对象的比较模式，那么将导致一个错误。

下面这个例子说明了 **CompareMode** 属性的用法：

```
function TestCompareMode(key)
{
    // 创建一些变量。

    var a, d;

    var BinaryCompare = 0, TextCompare = 1;

    d = new
ActiveXObject("Scripting.Dictionary");

    // 将比较模式设置为文本模式。

    d.CompareMode = TextCompare;

    // 添加一些主键和条目。

    d.Add("a", "Athens");
    d.Add("b", "Belgrade");
    d.Add("c", "Cairo");

    return(d.Item(key));
}
```

请参阅

[Key 属性](#)(See 1.4.3.21)

应用于：[Dictionary 对象](#)(See 1.4.2.1)

1.4.3.7 Count 属性

返回一个集合或 **Dictionary** 对象中的条目数。只读属性。

object.Count

其中 *object* 总是“应用于”列表中一个条目的名称。

说明

下面这段代码说明了 **Count** 属性的用法:

```
function CountDemo()
{
    var a, d, i, s;                                // 创建一
些变量。
    d = new
ActiveXObject("Scripting.Dictionary");

    d.Add ("a", "Athens");                          // 添
加一些主键和条目。
    d.Add ("b", "Belgrade");
    d.Add ("c", "Cairo");
    a = (new VBArray(d.Keys()));                    // 获
取主键。
    s = "";
    for (i = 0; i < d.Count; i++)                  // 遍历
dictionary。
    {
        s += a.getItem(i) + " - " +
d(a.getItem(i)) + "<br>";
    }
}
```

```
return(s); // 返
```

回结果。

```
}
```

请参阅

[CompareMode 属性](#)(See 1.4.3.6) | [Item 属性](#)(See 1.4.3.20) | [Key 属性](#)(See 1.4.3.21)
应用于: [Dictionary 对象](#)(See 1.4.2.1) | [Drives 集合](#)(See 1.4.2.3) | [Files 集合](#)(See 1.4.2.5) | [Folders 集合](#)(See 1.4.2.8)

1.4.3.8 DateCreated 属性

返回指定文件或文件夹的创建日期和时间。 只读。

`object.DateCreated`

`object` 应为 **File** 或 **Folder** 对象。

说明

下面的代码通过一个文件说明了 **DateCreated** 属性的用法:

```
function ShowFileInfo(filespec)
{
    var fso, f, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFile(filespec);
    s = "Created: " + f.DateCreated;
    return(s);
}
```

请参阅

[Attributes 属性](#)(See 1.4.3.3) | [DateLastAccessed 属性](#)(See 1.4.3.9) | [DateLastModified 属性](#)(See 1.4.3.10) | [Drive 属性](#)(See 1.4.3.11) | [Files 属性](#)(See 1.4.3.15) | [IsRootFolder 属性](#)(See 1.4.3.19) | [Name 属性](#)(See 1.4.3.23) | [ParentFolder 属性](#)(See 1.4.3.24) | [Path 属性](#)(See 1.4.3.25) | [ShortName 属性](#)(See 1.4.3.29) | [ShortPath 属性](#)(See 1.4.3.30) | [Size 属性](#)(See 1.4.3.31) | [SubFolders 属性](#)(See 1.4.3.32) | [Type 属性](#)(See 1.4.3.34)

应用于: [File 对象](#)(See 1.4.2.4) | [Folder 对象](#)(See 1.4.2.7)

1.4.3.9 DateLastAccessed 属性

返回最后访问指定文件或文件夹的日期和时间。 只读。

`object.DateLastAccessed`

`object` 应为 **File** 或 **Folder** 对象。

说明

下面的代码通过一个文件说明了 **DateLastAccessed** 属性的用法:

```
function ShowFileAccessInfo(filespec)
{
    var fso, f, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFile(filespec);
    s = filespec.toUpperCase() + "<br>";
    s += "创建时间: " + f.DateCreated + "<br>";
    s += "最近一次访问时间: " + f.DateLastAccessed
+ "<br>";

    s += "最近一次修改时间: " + f.DateLastModified;
    return(s);
}
```

重要 这个方法的操作依赖于底层的操作系统。 如果操作系统不支持提供时间信息, 那么将不会返回任何信息。

请参阅

[Attributes 属性](#)(See 1.4.3.3) | [DateCreated 属性](#)(See 1.4.3.8) | [DateLastModified 属性](#)

(See 1.4.3.10) | [Drive 属性](#)(See 1.4.3.11) | [Files 属性](#)(See 1.4.3.15) | [IsRootFolder 属性](#)(See 1.4.3.19) | [Name 属性](#)(See 1.4.3.23) | [ParentFolder 属性](#)(See 1.4.3.24) | [Path 属性](#)(See 1.4.3.25) | [ShortName 属性](#)(See 1.4.3.29) | [ShortPath 属性](#)(See 1.4.3.30) | [Size 属性](#)(See 1.4.3.31) | [SubFolders 属性](#)(See 1.4.3.32) | [Type 属性](#)(See 1.4.3.34)
 应用于: [File 对象](#)(See 1.4.2.4) | [Folder 对象](#)(See 1.4.2.7)

1.4.3.10 DateLastModified 属性

返回最后修改指定文件或文件夹的日期和时间。 只读。

`object.DateLastModified`

`object` 应为 **File** 或 **Folder** 对象。

说明

下面的代码通过一个文件说明了 **DateLastModified** 属性的用法:

```
function ShowFileAccessInfo(filespec)
{
    var fso, f, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFile(filespec);
    s = filespec.toUpperCase() + "<br>";
    s += "创建时间: " + f.DateCreated + "<br>";
    s += "最近一次访问时间: " + f.DateLastAccessed
+ "<br>";
    s += "最近一次修改时间: " + f.DateLastModified;
    return(s);
}
```

请参阅

[Attributes 属性](#)(See 1.4.3.3) | [DateCreated 属性](#)(See 1.4.3.8) | [DateLastAccessed 属性](#)

(See 1.4.3.9) | [Drive 属性](#)(See 1.4.3.11) | [Files 属性](#)(See 1.4.3.15) | [IsRootFolder 属性](#)(See 1.4.3.19) | [Name 属性](#)(See 1.4.3.23) | [ParentFolder 属性](#)(See 1.4.3.24) | [Path 属性](#)(See 1.4.3.25) | [ShortName 属性](#)(See 1.4.3.29) | [ShortPath 属性](#)(See 1.4.3.30) | [Size 属性](#)(See 1.4.3.31) | [SubFolders 属性](#)(See 1.4.3.32) | [Type 属性](#)(See 1.4.3.34)
 应用于： [File 对象](#)(See 1.4.2.4) | [Folder 对象](#)(See 1.4.2.7)

1.4.3.11 Drive 属性

返回指定文件或文件夹所在驱动器的驱动器号。 只读。

object.**Drive**

object 应为 **File** 或 **Folder** 对象。

说明

下面的代码说明了 **Drive** 属性的用法：

```
function ShowFileAccessInfo(filespec)
{
    var fso, f, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFile(filespec);
    s = f.Name + " on Drive " + f.Drive
+ "<br>";

    s += " 创建时间: " + f.DateCreated + "<br>";
    s += " 最近一次访问时间: " + f.DateLastAccessed
+ "<br>";

    s += " 最近一次修改时间: " + f.DateLastModified;
    return(s);
}
```

请参阅

[Attributes 属性](#)(See 1.4.3.3) | [DateCreated 属性](#)(See 1.4.3.8) | [DateLastAccessed 属性](#)(See 1.4.3.9) | [DateLastModified 属性](#)(See 1.4.3.10) | [Files 属性](#)(See 1.4.3.15) | [IsRootFolder 属性](#)(See 1.4.3.19) | [Name 属性](#)(See 1.4.3.23) | [ParentFolder 属性](#)(See 1.4.3.24) | [Path 属性](#)(See 1.4.3.25) | [ShortName 属性](#)(See 1.4.3.29) | [ShortPath 属性](#)(See 1.4.3.30) | [Size 属性](#)(See 1.4.3.31) | [SubFolders 属性](#)(See 1.4.3.32) | [Type 属性](#)(See 1.4.3.34)

应用于: [File 对象](#)(See 1.4.2.4) | [Folder 对象](#)(See 1.4.2.7)

1.4.3.12 DriveLetter 属性

返回一个物理上的本地驱动器或者网络共享的驱动器号。只读属性。

`object.DriveLetter`

其中 *object* 总是一个 **Drive** 对象。

说明

如果所指定的驱动器没有与一个驱动器号关联起来, 例如, 一个没有映射到驱动器号的网络共享, 那么 **DriveLetter** 属性将返回一个长度为 0 的字符串 ("")。

下面这段代码说明了 **DriveLetter** 属性的用法:

```
function ShowDriveLetter(drvPath)
{
    var fso, d, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    d =
fso.GetDrive(fso.GetDriveName(drvPath));

    s = "Drive " +
d.DriveLetter.toUpperCase() + ": - ";

    s += d.VolumeName + "<br>";

    s += "Available Space: " +
```

```
d.AvailableSpace/1024 + " Kbytes";

    return(s);
}
```

请参阅

[AvailableSpace 属性](#)(See 1.4.3.4) | [DriveType 属性](#)(See 1.4.3.14) | [FileSystem 属性](#)(See 1.4.3.16) | [FreeSpace 属性](#)(See 1.4.3.17) | [IsReady 属性](#)(See 1.4.3.18) | [Path 属性](#)(See 1.4.3.25) | [RootFolder 属性](#)(See 1.4.3.26) | [SerialNumber 属性](#)(See 1.4.3.27) | [ShareName 属性](#)(See 1.4.3.28) | [TotalSize 属性](#)(See 1.4.3.33) | [VolumeName 属性](#)(See 1.4.3.35)

应用于: [Drive 对象](#)(See 1.4.2.2)

1.4.3.13 Drives 属性

返回一个 **Drives** 集合，包含了本地机器上所有可用的 **Drive** 对象。

object.**Drives**

object 应为 FileSystemObject 。

说明

可移动媒体的驱动器不需要插入媒体就可以出现在 **Drives** 集合中。

可以通过 **Enumerator** 对象和 **for** 语句来逐个引用 **Drives** 集合中的成员：

```
function ShowDriveList()

{

    var fso, s, n, e, x;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    e = new Enumerator(fso.Drives);
    s = "";
    for (; !e.atEnd(); e.moveNext())
    {
```

```

        x = e.item();
        s = s + x.DriveLetter;
        s += " - ";
        if (x.DriveType == 3)
            n = x.ShareName;
        else if (x.IsReady)
            n = x.VolumeName;
        else
            n = "[ 驱动器未就绪 ] ";
        s +=    n + "<br>";
    }
    return(s);
}

```

请参阅

[Drives 集合](#)(See 1.4.2.3) | [Files 属性](#)(See 1.4.3.15) | [SubFolders 属性](#)(See 1.4.3.32)

应用于: [FileSystemObject 对象](#)(See 1.4.2.6)

1.4.3.14 DriveType 属性

返回一个值，表示所指定驱动器的类型。

object.DriveType

其中 *object* 总是一个 **Drive** 对象。

说明

下面这段代码说明了 **DriveType** 属性的用法：

```

function ShowDriveType(drvpath)
{
    var fso, d, s, t;

```



```

        fso                                =                                new
ActiveXObject("Scripting.FileSystemObject");

        d = fso.GetDrive(drspath);
        switch (d.DriveType)
        {
            case 0: t = "Unknown"; break;
            case 1: t = "Removable"; break;
            case 2: t = "Fixed"; break;
            case 3: t = "Network"; break;
            case 4: t = "CD-ROM"; break;
            case 5: t = "RAM Disk"; break;
        }

        s = "Drive " + d.DriveLetter + ": -
" + t;

        return(s);
    }

```

请参阅

[AvailableSpace 属性](#)(See 1.4.3.4) | [DriveLetter 属性](#)(See 1.4.3.12) | [FileSystem 属性](#)(See 1.4.3.16) | [FreeSpace 属性](#)(See 1.4.3.17) | [IsReady 属性](#)(See 1.4.3.18) | [Path 属性](#)(See 1.4.3.25) | [RootFolder 属性](#)(See 1.4.3.26) | [SerialNumber 属性](#)(See 1.4.3.27) | [ShareName 属性](#)(See 1.4.3.28) | [TotalSize 属性](#)(See 1.4.3.33) | [VolumeName 属性](#)(See 1.4.3.35)

应用于: [Drive 对象](#)(See 1.4.2.2)

1.4.3.15 Files 属性

返回一个 **Files** 集合，由指定文件夹中包含的所有 **File** 对象组成，包括设置了隐藏和系统文件属性的文件。

object.**Files**

object 应为 **Folder** 对象。

说明

下面的代码说明了 **Files** 属性的用法：

```
function ShowFolderFileList(folderspec)
{
    var fso, f, fc, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFolder(folderspec);
    fc = new Enumerator(f.files);
    s = "";
    for (; !fc.atEnd(); fc.moveNext())
    {
        s += fc.item();
        s += "<br>";
    }

    return(s);
}
```

请参阅

[Attributes 属性](#)(See 1.4.3.3) | [DateCreated 属性](#)(See 1.4.3.8) | [DateLastAccessed 属性](#)

(See 1.4.3.9) | [DateLastModified 属性](#)(See 1.4.3.10) | [Drive 属性](#)(See 1.4.3.11) | [IsRootFolder 属性](#)(See 1.4.3.19) | [Name 属性](#)(See 1.4.3.23) | [ParentFolder 属性](#)(See 1.4.3.24) | [Path 属性](#)(See 1.4.3.25) | [ShortName 属性](#)(See 1.4.3.29) | [ShortPath 属性](#)(See 1.4.3.30) | [Size 属性](#)(See 1.4.3.31) | [SubFolders 属性](#)(See 1.4.3.32) | [Type 属性](#)(See 1.4.3.34)

应用于: [Folder 对象](#)(See 1.4.2.7)

1.4.3.16 FileSystem 属性

返回指定驱动器所使用的文件系统的类型。

`object.FileSystem`

其中 *object* 总是一个 **Drive** 对象。

说明

可能的返回类型包括 FAT、NTFS、和 CDFS。

下面这段代码说明了 **FileSystem** 属性的用法:

```
function ShowFileSystemType(drvPath)
{
    var fso, d, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    d = fso.GetDrive(drvPath);
    s = d.FileSystem;

    return(s);
}
```

请参阅

[AvailableSpace 属性](#)(See 1.4.3.4) | [DriveLetter 属性](#)(See 1.4.3.12) | [DriveType 属性](#)(See 1.4.3.14) | [FreeSpace 属性](#)(See 1.4.3.17) | [IsReady 属性](#)(See 1.4.3.18) | [Path 属性](#)(See 1.4.3.25) | [RootFolder 属性](#)(See 1.4.3.26) | [SerialNumber 属性](#)(See 1.4.3.27) | [ShareName 属性](#)(See 1.4.3.28) | [TotalSize 属性](#)(See 1.4.3.33) | [VolumeName 属性](#)(See 1.4.3.35)

应用于: [Drive 对象](#)(See 1.4.2.2)

1.4.3.17 FreeSpace 属性

向用户返回指定驱动器或网络共享上的可用空间的大小。只读。

`object.FreeSpace`

`object` 应为 **Drive** 对象。

说明

典型情况中，由 **FreeSpace** 属性返回的值和由 **AvailableSpace** 属性返回的值是相同的。对于支持 `quotas` 的计算机系统来说两者有可能不同。

下面的代码说明了 **FreeSpace** 属性的用法：

```
function ShowFreeSpace(drvPath)
{
    var fso, d, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    d =
fso.GetDrive(fso.GetDriveName(drvPath));

    s = "Drive " + drvPath.toUpperCase()
+ " - ";

    s += d.VolumeName + "<br>";

    s += "Free Space: " +
d.FreeSpace/1024 + " Kbytes";

    return(s);
}
```

请参阅

[AvailableSpace 属性](#)(See 1.4.3.4) | [DriveLetter 属性](#)(See 1.4.3.12) | [DriveType 属性](#)(See 1.4.3.14) | [FileSystem 属性](#)(See 1.4.3.16) | [IsReady 属性](#)(See 1.4.3.18) | [Path 属](#)

[性](#)(See 1.4.3.25) | [RootFolder 属性](#)(See 1.4.3.26) | [SerialNumber 属性](#)(See 1.4.3.27) | [ShareName 属性](#)(See 1.4.3.28) | [TotalSize 属性](#)(See 1.4.3.33) | [VolumeName 属性](#)(See 1.4.3.35)

应用于: [Drive 对象](#)(See 1.4.2.2)

1.4.3.18 IsReady 属性

如果指定驱动器已就绪则返回 **True** ; 否则 **False**。

object.**IsReady**

object 应为 **Drive** 对象。

说明

对于可移动媒体的驱动器和 CD-ROM 驱动器来说, **IsReady** 只有在插入了适当的媒体并已准备好访问时才返回 **True** 。

下面的代码说明了 **IsReady** 属性的用法:

```
function ShowDriveInfo(drvpath)
{
    var fso, d, s, t;

    fso = new
ActiveXObject("Scripting.FileSystemObject")

    d = fso.GetDrive(drvpath)
    switch (d.DriveType)
    {
        case 0: t = "Unknown"; break;
        case 1: t = "Removable"; break;
        case 2: t = "Fixed"; break;
        case 3: t = "Network"; break;
        case 4: t = "CD-ROM"; break;
```

```

        case 5: t = "RAM Disk"; break;
    }

    s = "Drive " + d.DriveLetter + ": -
" + t;

    if (d.IsReady)
        s += "<br>" + "驱动器就绪。";
    else
        s += "<br>" + "驱动器未准备好。";

    return(s);
}

```

请参阅

[AvailableSpace 属性](#)(See 1.4.3.4) | [DriveLetter 属性](#)(See 1.4.3.12) | [DriveType 属性](#)(See 1.4.3.14) | [FileSystem 属性](#)(See 1.4.3.16) | [FreeSpace 属性](#)(See 1.4.3.17) | [Path 属性](#)(See 1.4.3.25) | [RootFolder 属性](#)(See 1.4.3.26) | [SerialNumber 属性](#)(See 1.4.3.27) | [ShareName 属性](#)(See 1.4.3.28) | [TotalSize 属性](#)(See 1.4.3.33) | [VolumeName 属性](#)(See 1.4.3.35)

应用于: [Drive 对象](#)(See 1.4.2.2)

1.4.3.19 IsRootFolder 属性

如果指定的文件夹是根文件夹则返回 **True** ; 否则返回 **False** 。

object.IsRootFolder

object 应为 **Folder** 对象。

说明

下面的代码说明了 **IsRootFolder** 属性的用法:

```

function DisplayLevelDepth(pathspec)
{
    var fso, f, n, s = "";

    fso = new

```

```
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFolder(pathspec);
    n = 0;
    if (f.IsRootFolder)
        s = "The specified folder is the
root folder."
    else
    {
        do
        {
            f = f.ParentFolder;
            n++;
        }
        while (!f.IsRootFolder)
        s = "The specified folder is
nested " + n + " levels deep."
    }
    return(s);
}
```

请参阅

[Attributes 属性](#)(See 1.4.3.3) | [DateCreated 属性](#)(See 1.4.3.8) | [DateLastAccessed 属性](#)(See 1.4.3.9) | [DateLastModified 属性](#)(See 1.4.3.10) | [Drive 属性](#)(See 1.4.3.11) | [Files 属性](#)(See 1.4.3.15) | [Name 属性](#)(See 1.4.3.23) | [ParentFolder 属性](#)(See 1.4.3.24) | [Path](#)

[属性](#)(See 1.4.3.25) | [ShortName 属性](#)(See 1.4.3.29) | [ShortPath 属性](#)(See 1.4.3.30) | [Size 属性](#)(See 1.4.3.31) | [SubFolders 属性](#)(See 1.4.3.32) | [Type 属性](#)(See 1.4.3.34)
 应用于: [Folder 对象](#)(See 1.4.2.7)

1.4.3.20 Item 属性

在一个 **Dictionary** 对象中设置或者返回所指定 *key* 的 *item*。对于集合则根据所指定的 *key* 返回一个 *item*。读/写。

`object.Item(key) [= newitem]`

Item 属性包括下面这些部分:

部分	描述
object	必选项。总是一个集合的名称或 Dictionary 对象的名称。
key	必选项。与要被查找或添加的 <i>item</i> 相关联的 <i>key</i> 。
newitem	可选项。仅适用于 Dictionary 对象; 对于集合不可使用此项。如果提供了此项, <i>newitem</i> 就是与所指定的 <i>key</i> 相关联的新值。

说明

如果在改变一个 *key* 的时候没有找到该 *item*, 那么将利用所指定的 *newitem* 创建一个新的 *key*。如果在试图返回一个已有条目时候没有找到 *key*, 那么将创建一个新的 *key* 且其相关的条目被设置为空。

下面这个例子说明了 **Item** 属性的用法。

```
function DicTest(keyword)
{
    var a, d;

    d = new
ActiveXObject("Scripting.Dictionary");

    d.Add("a", "Athens");
    d.Add("b", "Belgrade");
    d.Add("c", "Cairo");

    a = d.Item(keyword);

    return(a);
}
```


请参阅

[CompareMode 属性](#)(See 1.4.3.6) | [Count 属性](#)(See 1.4.3.7) | [Key 属性](#)(See 1.4.3.21)
 应用于: [Dictionary 对象](#)(See 1.4.2.1) | [Drives 集合](#)(See 1.4.2.3) | [Files 集合](#)(See 1.4.2.5) | [Folders 集合](#)(See 1.4.2.8)

1.4.3.21 Key 属性

在 **Dictionary** 对象中设置一个 *key*。

`object.Key(key) = newkey`

Key 属性包括下面这些部分:

部分	描述
object	必选项。总是一个 Dictionary 对象的名称。
key	必选项。被改变的 <i>key</i> 值。
newkey	必选项。替换所指定的 <i>key</i> 的新值。

说明

如果在改变一个 *key* 时没有发现该 *key*, 那么将创建一个新的 *key* 并且其相关联的 *item* 被设置为空。

下面这段代码说明了 **Key** 属性的用法:

```
var d;

d = new
ActiveXObject("Scripting.Dictionary");

function AddStuff()
{
    var a;

    d.Add("a", "Athens");
    d.Add("b", "Belgrade");
    d.Add("c", "Cairo");
}
```

```
function ChangeKey(oldkey, newkey)
{
    var s;

    d.Key("c") = "Ca";

    s = "Key " + oldkey + " changed to "
+ newkey;

    return(s);
}
```

请参阅

[CompareMode 属性](#)(See 1.4.3.6) | [Count 属性](#)(See 1.4.3.7) | [Item 属性](#)(See 1.4.3.20)
 应用于: [Dictionary 对象](#)(See 1.4.2.1)

1.4.3.22 Line 属性

只读属性, 返回 **TextStream** 文件中当前的行号。

`object.Line`

object 总是 **TextStream** 对象的名称。

说明

在文件初始打开并写入任何字符之前, **Line** 等于 1。

下面的示例演示了 **Line** 属性的用法:

```
function GetLine()
{
    var fso, f, r

    var ForReading = 1, ForWriting = 2;

    fso = new
ActiveXObject("Scripting.FileSystemObject")
```

```

        f
    =
fso.OpenTextFile("c:\\textfile.txt",
ForWriting, true)
    f.WriteLine("Hello world!");
    f.WriteLine("JScript is fun");
    f.Close();
    f
    =
fso.OpenTextFile("c:\\textfile.txt",
ForReading);
    r =    f.ReadAll();
    return(f.Line);
}

```

请参阅

[Column 属性](#)(See 1.4.3.5)

应用于: [TextStream 对象](#)(See 1.4.2.9)

1.4.3.23 Name 属性

设置或返回指定文件或文件夹的名称。 读/写。

`object.Name [= newname]`

Name 属性包括了以下部分:

部分	描述
<code>object</code>	必选项。 应为 File 或 Folder 对象的名称。
<code>newname</code>	可选项。 如果提供了这个部分, <i>newname</i> 将成为指定的 <i>object</i> 的新名称。

说明

下面的代码说明了 **Name** 属性的用法:

```
function ShowFileAccessInfo(filespec)
```

```

{
    var fso, f, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFile(filespec);

    s = f.Name + " on Drive " + f.Drive
+ "<br>";

    s += "创建时间: " + f.DateCreated + "<br>";

    s += "最近一次访问时间: " + f.DateLastAccessed
+ "<br>";

    s += "最近一次修改时间: " + f.DateLastModified;

    return(s);
}

```

请参阅

[Attributes 属性](#)(See 1.4.3.3) | [DateCreated 属性](#)(See 1.4.3.8) | [DateLastAccessed 属性](#)(See 1.4.3.9) | [DateLastModified 属性](#)(See 1.4.3.10) | [Drive 属性](#)(See 1.4.3.11) | [Files 属性](#)(See 1.4.3.15) | [IsRootFolder 属性](#)(See 1.4.3.19) | [ParentFolder 属性](#)(See 1.4.3.24) | [Path 属性](#)(See 1.4.3.25) | [ShortName 属性](#)(See 1.4.3.29) | [ShortPath 属性](#)(See 1.4.3.30) | [Size 属性](#)(See 1.4.3.31) | [SubFolders 属性](#)(See 1.4.3.32) | [Type 属性](#)(See 1.4.3.34)

应用于: [File 对象](#)(See 1.4.2.4) | [Folder 对象](#)(See 1.4.2.7)

1.4.3.24 ParentFolder 属性

返回指定文件或文件夹的父文件夹对象。 只读。

`object.ParentFolder`

`object` 应为 **File** 或 **Folder** 对象。

说明

下面的代码通过一个文件说明了 **ParentFolder** 属性的用法:

```
function ShowFileAccessInfo(filespec)
{
    var fso, f, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFile(filespec);
    s = f.Name + " in " + f.ParentFolder
+ "<br>";

    s += "创建: " + f.DateCreated + "<br>";
    s += "最近一次访问时间: " + f.DateLastAccessed
+ "<br>";

    s += "最近一次修改时间: " + f.DateLastModified;

    return(s);
}
```

请参阅

[Attributes 属性](#)(See 1.4.3.3) | [DateCreated 属性](#)(See 1.4.3.8) | [DateLastAccessed 属性](#)(See 1.4.3.9) | [DateLastModified 属性](#)(See 1.4.3.10) | [Drive 属性](#)(See 1.4.3.11) | [Files 属性](#)(See 1.4.3.15) | [IsRootFolder 属性](#)(See 1.4.3.19) | [Name 属性](#)(See 1.4.3.23) | [Path 属性](#)(See 1.4.3.25) | [ShortName 属性](#)(See 1.4.3.29) | [ShortPath 属性](#)(See 1.4.3.30) | [Size 属性](#)(See 1.4.3.31) | [SubFolders 属性](#)(See 1.4.3.32) | [Type 属性](#)(See 1.4.3.34)
 应用于: [File 对象](#)(See 1.4.2.4) | [Folder 对象](#)(See 1.4.2.7)

1.4.3.25 Path 属性

返回指定文件、文件夹或驱动器的路径。

object.**Path**

object 总是为 **File**、**Folder** 或 **Drive** 对象。

说明

驱动器字母后不包括根驱动器。例如，C 驱动器的路径是 C:，而不是 C:\。下面的代码通过 **File** 对象说明了 **Path** 属性的用法：

```
function ShowFileAccessInfo(filespec)
{
    var fso, d, f, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFile(filespec);
    s = f.Path.toUpperCase() + "<br>";
    s += "创建时间: " + f.DateCreated + "<br>";
    s += "最近一次访问时间: " + f.DateLastAccessed
+ "<br>";

    s += "最近一次修改时间: " +
f.DateLastModified

    return(s);
}
```

请参阅

[Attributes 属性](#) (See 1.4.3.3) | [AvailableSpace 属性](#) (See 1.4.3.4) | [DateCreated 属性](#) (See 1.4.3.8) | [DateLastAccessed 属性](#) (See 1.4.3.9) | [DateLastModified 属性](#) (See 1.4.3.10) | [Drive 属性](#) (See 1.4.3.11) | [DriveLetter 属性](#) (See 1.4.3.12) | [DriveType 属性](#) (See 1.4.3.14) | [Files 属性](#) (See 1.4.3.15) | [FileSystem 属性](#) (See 1.4.3.16) | [FreeSpace 属性](#) (See 1.4.3.17) | [IsReady 属性](#) (See 1.4.3.18) | [IsRootFolder 属性](#) (See 1.4.3.19) | [Name 属性](#) (See 1.4.3.23) | [ParentFolder 属性](#) (See 1.4.3.24) | [RootFolder 属性](#) (See 1.4.3.26) | [SerialNumber 属性](#) (See 1.4.3.27) | [ShareName 属性](#) (See 1.4.3.28) | [ShortName 属性](#) (See 1.4.3.29) | [ShortPath 属性](#) (See 1.4.3.30) | [Size 属性](#) (See 1.4.3.31) | [SubFolders 属性](#) (See 1.4.3.32) | [TotalSize 属性](#) (See 1.4.3.33) | [Type 属性](#) (See

1.4.3.34) | [VolumeName 属性](#)(See 1.4.3.35)

应用于: [Drive 对象](#)(See 1.4.2.2) | [File 对象](#)(See 1.4.2.4) | [Folder 对象](#)(See 1.4.2.7)

1.4.3.26 RootFolder 属性

返回一个 **Folder** 对象，表示指定驱动器的根文件夹。 只读。

object.RootFolder

object 总是为 **Drive** 对象。

说明

可以通过返回的 **Folder** 对象来访问驱动器上的所有文件和文件夹。

下面的例子说明了 **RootFolder** 属性的用法：

```
function GetRootFolder(drv)
{
    var fso,d;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    if (fso.DriveExists(drv))
    {
        d = fso.GetDrive(drv);
        return(d.RootFolder);
    }
    else
        return(false);
}
```

请参阅

[AvailableSpace 属性](#)(See 1.4.3.4) | [DriveLetter 属性](#)(See 1.4.3.12) | [DriveType 属性](#)(See 1.4.3.14) | [FileSystem 属性](#)(See 1.4.3.16) | [FreeSpace 属性](#)(See 1.4.3.17) | [IsReady 属性](#)(See 1.4.3.18) | [Path 属性](#)(See 1.4.3.25) | [SerialNumber 属性](#)(See

1.4.3.27) | [ShareName 属性](#) (See 1.4.3.28) | [TotalSize 属性](#) (See 1.4.3.33) | [VolumeName 属性](#) (See 1.4.3.35)

应用于: [Drive 对象](#) (See 1.4.2.2)

1.4.3.27 SerialNumber 属性

返回连续十进制数字, 用于唯一标识磁盘卷。

`object.SerialNumber`

`object` 为 **Drive** 对象。

说明

可以使 **SerialNumber** 属性来确保在带有可移动媒体的驱动器中插入正确的磁盘。下列代码阐明了 **SerialNumber** 属性的用法:

```
function ShowDriveInfo (drvpath) {
    var fso, d, s, t;

    fso = new
ActiveXObject ("Scripting.FileSystemObject");

    d =
fso.GetDrive (fso.GetDriveName (fso.GetAbsolutePathName (drvpath)));

    switch (d.DriveType)
    {
        case 0: t = "Unknown"; break;
        case 1: t = "Removable"; break;
        case 2: t = "Fixed"; break;
        case 3: t = "Network"; break;
        case 4: t = "CD-ROM"; break;
```



```

        case 5: t = "RAM Disk"; break;
    }

    s = "Drive " + d.DriveLetter + ": - "
+ t;

    s += "<br>" + "SN: " + d.SerialNumber;
    return(s);
}

```

要求

[版本 1](#) (See 2.64)

1.4.3.28 ShareName 属性

返回指定驱动器的网络共享名。

object.ShareName

object 总是为 **Drive** 对象。

说明

如果 *object* 不是网络驱动器，那么 **ShareName** 属性将返回长度为零的字符串 ("")。

下面的代码说明了 **ShareName** 属性的用法：

```

function ShowDriveInfo(drvpath)
{
    var fso, d, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    d =
fso.GetDrive(fso.GetDriveName(fso.GetAb

```

```

solutePathName (drvpath) ) ) ;

    s = "Drive " + d.DriveLetter + ": -
" + d.ShareName;

    return (s) ;
}

```

请参阅

[AvailableSpace 属性](#) (See 1.4.3.4) | [DriveLetter 属性](#) (See 1.4.3.12) | [DriveType 属性](#) (See 1.4.3.14) | [FileSystem 属性](#) (See 1.4.3.16) | [FreeSpace 属性](#) (See 1.4.3.17) | [IsReady 属性](#) (See 1.4.3.18) | [Path 属性](#) (See 1.4.3.25) | [RootFolder 属性](#) (See 1.4.3.26) | [SerialNumber 属性](#) (See 1.4.3.27) | [TotalSize 属性](#) (See 1.4.3.33) | [VolumeName 属性](#) (See 1.4.3.35)

应用于: [Drive 对象](#) (See 1.4.2.2)

1.4.3.29 ShortName 属性

返回短名称, 这些短名称由需要以前的 8.3 命名规范的程序使用。

object.ShortName

object 应为 **File** 或 **Folder** 对象。

说明

下面的代码通过一个 **File** 对象说明了 **ShortName** 属性的用法:

```

function ShowShortName (filespec)
{
    var fso, f, s;

    fso = new
ActiveXObject ("Scripting.FileSystemObje
ct");

    f = fso.GetFile (filespec);

    s = "The short name for " + "" +
f.Name;

```

```

    s += "\"" + "<br>";

    s += "is: " + "\"" + f.ShortName + "\"";

    return(s);
}

```

请参阅

[Attributes 属性](#)(See 1.4.3.3) | [DateCreated 属性](#)(See 1.4.3.8) | [DateLastAccessed 属性](#)(See 1.4.3.9) | [DateLastModified 属性](#)(See 1.4.3.10) | [Drive 属性](#)(See 1.4.3.11) | [Files 属性](#)(See 1.4.3.15) | [IsRootFolder 属性](#)(See 1.4.3.19) | [Name 属性](#)(See 1.4.3.23) | [ParentFolder 属性](#)(See 1.4.3.24) | [Path 属性](#)(See 1.4.3.25) | [ShortPath 属性](#)(See 1.4.3.30) | [Size 属性](#)(See 1.4.3.31) | [SubFolders 属性](#)(See 1.4.3.32) | [Type 属性](#)(See 1.4.3.34)

应用于: [File 对象](#)(See 1.4.2.4) | [Folder 对象](#)(See 1.4.2.7)

1.4.3.30 ShortPath 属性

返回短路径名，这些短文件名由需要以前的 8.3 文件命名规范的程序使用。

object.ShortPath

object 应为 **File** 或 **Folder** 对象。

说明

下面的代码通过一个 **File** 对象说明了 **ShortName** 属性的用法：

```

function ShowShortPath(filespec)
{
    var fso, f, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFile(filespec);

    s = "The short path for " + "\"" +
f.Name;

```

```

    s += "" + "<br>";

    s += "is: " + "" + f.ShortPath + "";

    return(s);
}

```

请参阅

[Attributes 属性](#)(See 1.4.3.3) | [DateCreated 属性](#)(See 1.4.3.8) | [DateLastAccessed 属性](#)(See 1.4.3.9) | [DateLastModified 属性](#)(See 1.4.3.10) | [Drive 属性](#)(See 1.4.3.11) | [Files 属性](#)(See 1.4.3.15) | [IsRootFolder 属性](#)(See 1.4.3.19) | [Name 属性](#)(See 1.4.3.23) | [ParentFolder 属性](#)(See 1.4.3.24) | [Path 属性](#)(See 1.4.3.25) | [ShortName 属性](#)(See 1.4.3.29) | [Size 属性](#)(See 1.4.3.31) | [SubFolders 属性](#)(See 1.4.3.32) | [Type 属性](#)(See 1.4.3.34)

应用于: [File 对象](#)(See 1.4.2.4) | [Folder 对象](#)(See 1.4.2.7)

1.4.3.31 Size 属性

对于文件，以字节为单位返回指定文件的大小。对于文件夹，以字节为单位返回文件夹中包含的所有文件和子文件夹的大小。

object.**Size**

object 应为 **File** 或 **Folder** 对象。

说明

下面的代码通过一个 **Folder** 对象说明了 **Size** 属性的用法：

```

function ShowFolderSize(filespec)
{
    var fso, f, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFolder(filespec);

    s = f.Name + " uses " + f.size + "
bytes.";
}

```

```
    return (s) ;
}
```

请参阅

[Attributes 属性](#) (See 1.4.3.3) | [DateCreated 属性](#) (See 1.4.3.8) | [DateLastAccessed 属性](#) (See 1.4.3.9) | [DateLastModified 属性](#) (See 1.4.3.10) | [Drive 属性](#) (See 1.4.3.11) | [Files 属性](#) (See 1.4.3.15) | [IsRootFolder 属性](#) (See 1.4.3.19) | [Name 属性](#) (See 1.4.3.23) | [ParentFolder 属性](#) (See 1.4.3.24) | [Path 属性](#) (See 1.4.3.25) | [ShortName 属性](#) (See 1.4.3.29) | [ShortPath 属性](#) (See 1.4.3.30) | [SubFolders 属性](#) (See 1.4.3.32) | [Type 属性](#) (See 1.4.3.34)

应用于: [File 对象](#) (See 1.4.2.4) | [Folder 对象](#) (See 1.4.2.7)

1.4.3.32 SubFolders 属性

返回一个 **Folders** 集合，由指定文件夹中包含的所有文件夹组成，包括设置了隐藏和系统文件属性的文件夹。

object.SubFolders

object 应为 **Folder** 对象。

说明

下面的代码说明了 **SubFolders** 属性的用法：

```
function ShowFolderList(folderspec)
{
    var fso, f, fc, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    f = fso.GetFolder(folderspec);
    fc = new Enumerator(f.SubFolders);
    s = "";
    for (; !fc.atEnd(); fc.moveNext())
    {
```

```

        s += fc.item();

        s += "<br>";

    }

    return(s);
}

```

请参阅

[Attributes 属性](#)(See 1.4.3.3) | [DateCreated 属性](#)(See 1.4.3.8) | [DateLastAccessed 属性](#)(See 1.4.3.9) | [DateLastModified 属性](#)(See 1.4.3.10) | [Drive 属性](#)(See 1.4.3.11) | [Files 属性](#)(See 1.4.3.15) | [IsRootFolder 属性](#)(See 1.4.3.19) | [Name 属性](#)(See 1.4.3.23) | [ParentFolder 属性](#)(See 1.4.3.24) | [Path 属性](#)(See 1.4.3.25) | [ShortName 属性](#)(See 1.4.3.29) | [ShortPath 属性](#)(See 1.4.3.30) | [Size 属性](#)(See 1.4.3.31) | [Type 属性](#)(See 1.4.3.34)

应用于: [Folder 对象](#)(See 1.4.2.7)

1.4.3.33 TotalSize 属性

以字节为单位返回驱动器或网络共享的所有空间大小。

object.TotalSize

object 总是为 **Drive** 对象。

说明

下面的代码说明了 **TotalSize** 属性的用法:

```

function SpaceReport(drvPath)
{
    var fso, d, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    d =
fso.GetDrive(fso.GetDriveName(drvPath));

```

```

    s = "Drive " + drvPath + " - ";
    s += d.VolumeName + "<br>";
    s += "Total Space: " +
d.TotalSize/1024 + " Kbytes <br>";
    s += "Free Space: " +
d.FreeSpace/1024 + " Kbytes";
    return(s);
}

```

请参阅

[AvailableSpace 属性](#)(See 1.4.3.4) | [DriveLetter 属性](#)(See 1.4.3.12) | [DriveType 属性](#)(See 1.4.3.14) | [FileSystem 属性](#)(See 1.4.3.16) | [FreeSpace 属性](#)(See 1.4.3.17) | [IsReady 属性](#)(See 1.4.3.18) | [Path 属性](#)(See 1.4.3.25) | [RootFolder 属性](#)(See 1.4.3.26) | [SerialNumber 属性](#)(See 1.4.3.27) | [ShareName 属性](#)(See 1.4.3.28) | [VolumeName 属性](#)(See 1.4.3.35)

应用于: [Drive 对象](#)(See 1.4.2.2)

1.4.3.34 Type 属性

返回关于文件或文件夹类型的信息。例如,对于以 .TXT 结尾的文件将返回“文本文档”。

object.**Type**

object 应为 **File** 或 **Folder** 对象。

说明

下面的代码说明了利用 **Type** 属性返回文件夹属性的方法。在这个例子中,可以尝试一下向过程提供回收站或其他特殊文件夹的路径。

```

function ShowFileType(filespec)
{
    var fso, f, s;

    fso = new
ActiveXObject("Scripting.FileSystemObje

```

```

ct");

    if (fso.FolderExists(filespec))
        f = fso.GetFolder(filespec);
    else if (fso.FileExists(filespec))
        f = fso.GetFile(filespec);
    else
        s = "File or Folder does not
exist.";

    s = f.Name + " is a " + f.Type;

    return(s);
}

```

请参阅

[Attributes 属性](#)(See 1.4.3.3) | [DateCreated 属性](#)(See 1.4.3.8) | [DateLastAccessed 属性](#)(See 1.4.3.9) | [DateLastModified 属性](#)(See 1.4.3.10) | [Drive 属性](#)(See 1.4.3.11) | [Files 属性](#)(See 1.4.3.15) | [IsRootFolder 属性](#)(See 1.4.3.19) | [Name 属性](#)(See 1.4.3.23) | [ParentFolder 属性](#)(See 1.4.3.24) | [Path 属性](#)(See 1.4.3.25) | [ShortName 属性](#)(See 1.4.3.29) | [ShortPath 属性](#)(See 1.4.3.30) | [Size 属性](#)(See 1.4.3.31) | [SubFolders 属性](#)(See 1.4.3.32)

应用于: [File 对象](#)(See 1.4.2.4) | [Folder 对象](#)(See 1.4.2.7)

1.4.3.35 VolumeName 属性

设置或返回指定驱动器的卷名。 读/写。

object.VolumeName [= *newname*]

参数

object

必选项。 总是为 **Drive** 对象的名称。

newname

可选项。 如果提供了这个部分,那么 *newname* 将成为指定的 *object* 的新名称。

说明

下面的代码说明了 **VolumeName** 属性的用法:

```
function SpaceReport(drvPath)
{
    var fso, d, s;

    fso = new
ActiveXObject("Scripting.FileSystemObject");

    d =
fso.GetDrive(fso.GetDriveName(drvPath));

    s = "Drive " + drvPath + " - ";
    s += d.VolumeName + "<br>";
    s += "Total Space: " +
d.TotalSize/1024 + " Kbytes <br>";
    s += "Free Space: " +
d.FreeSpace/1024 + " Kbytes";

    return(s);
}
```

请参阅

[AvailableSpace 属性](#)(See 1.4.3.4) | [DriveLetter 属性](#)(See 1.4.3.12) | [DriveType 属性](#)(See 1.4.3.14) | [FileSystem 属性](#)(See 1.4.3.16) | [FreeSpace 属性](#)(See 1.4.3.17) | [IsReady 属性](#)(See 1.4.3.18) | [Path 属性](#)(See 1.4.3.25) | [RootFolder 属性](#)(See 1.4.3.26) | [SerialNumber 属性](#)(See 1.4.3.27) | [ShareName 属性](#)(See 1.4.3.28) | [TotalSize 属性](#)(See 1.4.3.33)

应用于: [Drive 对象](#)(See 1.4.2.2)

1.5 正则表达式简介

这些页包含的信息其目的是提供一个关于正则表达式的通用介绍。

尽管试图让每个主题的内容都比较独立,但这些主题所包含的大部分信息都依赖于对前面所介绍的特性或概念的理解。因此,建议您顺序地仔细阅读这些主题,以便最全面地了解这些材料。

“正则表达式简介”包括下述各个主题:

[正则表达式](#)(See 1.5.1)

[早期起源](#)(See 1.5.2)

[使用正则表达式](#)(See 1.5.3)

[正则表达式语法](#)(See 1.5.4)

[建立正则表达式](#)(See 1.5.5)

[优先权顺序](#)(See 1.5.6)

[普通字符](#)(See 1.5.7)

[特殊字符](#)(See 1.5.8)

[非打印字符](#)(See 1.5.9)

[字符匹配](#)(See 1.5.10)

[限定符](#)(See 1.5.11)

[定位符](#)(See 1.5.12)

[选择与编组](#)(See 1.5.13)

[后向引用](#)(See 1.5.14)

1.5.1 正则表达式

如果原来没有使用过正则表达式,那么可能对这个术语和概念会不太熟悉。不过,它们并不是您想象的那么新奇。

请回想一下在硬盘上是如何查找文件的。您肯定会使用 ? 和 * 字符来帮助查找您正寻找的文件。? 字符匹配文件名中的单个字符,而 * 则匹配一个或多个字符。一个如 'data?.dat' 的模式可以找到下述文件:

data1.dat

data2.dat

datax.dat

dataN.dat

如果使用 * 字符代替 ? 字符,则将扩大找到的文件数量。'data*.dat' 可以匹配下述所有文件名:

data.dat

data1.dat

data2.dat

data12.dat

datax.dat

dataXYZ.dat

尽管这种搜索文件的方法肯定很有用，但也十分有限。`?` 和 `*` 通配符的有限能力可以使你对正则表达式能做什么有一个概念，不过正则表达式的功能更强大，也更灵活。

1.5.2 早期起源

正则表达式的“祖先”可以一直上溯至对人类神经系统如何工作的早期研究。Warren McCulloch 和 Walter Pitts 这两位神经生理学家研究出一种数学方式来描述这些神经网络。

1956 年，一位叫 Stephen Kleene 的美国数学家在 McCulloch 和 Pitts 早期工作的基础上，发表了一篇标题为“神经网络事件的表示法”的论文，引入了正则表达式的概念。正则表达式就是用来描述他称为“正则集的代数”的表达式，因此采用“正则表达式”这个术语。

随后，发现可以将这一工作应用于使用 Ken Thompson 的计算搜索算法的一些早期研究，Ken Thompson 是 Unix 的主要发明人。正则表达式的第一个实用应用程序就是 Unix 中的 *qed* 编辑器。

如他们所说，剩下的就是众所周知的历史了。从那时起直至现在正则表达式都是基于文本的编辑器和搜索工具中的一个重要部分。

1.5.3 使用正则表达式

在典型的搜索和替换操作中，必须提供要查找的确切文字。这种技术对于静态文本中的简单搜索和替换任务可能足够了，但是由于它缺乏灵活性，因此在搜索动态文本时就有困难了，甚至是不可能的。

使用正则表达式，就可以：

- 测试字符串的某个模式。例如，可以对一个输入字符串进行测试，看该字符串是否存在一个电话号码模式或一个信用卡号码模式。这称为数据有效性验证。
- 替换文本。可以在文档中使用一个正则表达式来标识特定文字，然后可以全部将其删除，或者替换为别的文字。
- 根据模式匹配从字符串中提取一个子字符串。可以用来在文本或输入字段中查找特定文字。

例如，如果需要搜索整个 web 站点来删除某些过时的材料并替换某些 HTML 格式化标记，则可以使用正则表达式对每个文件进行测试，看在该文件中是否存在所要查找的材料或 HTML 格式化标记。用这个方法，就可以将受影响的文件范围缩小到包含要删除或更改的材料的那些文件。然后可以使用正则表达式来删除过时的材料，最后，可以再次使用正则表达式来查找并替换那些需要替换的标记。另一个说明正则表达式非常有用的示例是一种其字符串处理能力还不为人所知的语言。VBScript 是 Visual Basic 的一个子集，具有丰富的字符串处理功能。与 C 类似的 Jscript 则没有这一能力。正则表达式给 JScript 的字符串处理能力带来了明显改善。不过，可能还是在 VBScript 中使用正则表达式的效率更高，它允许在单个表达式中执行多个字符串操作。

1.5.4 正则表达式语法

一个正则表达式就是由普通字符（例如字符 a 到 z）以及特殊字符（称为元字符）组成的文字模式。该模式描述在查找文字主体时待匹配的一个或多个字符串。正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。

这里有一些可能会遇到的正则表达式示例：

JScript	VBScript	匹配
/^[\t]*\$/	"^[\t]*\$"	匹配一个空白行。
/\d{2}-\d{5}/	"\d{2}-\d{5}"	验证一个 ID 号码是否由一个 2 位数字，一个连字符以及一个 5 位数字组成。
/<(.*?)>.*<\/1>/	"<(.*?)>.*<\/1>"	匹配一个 HTML 标记。

下表是元字符及其在正则表达式上下文中的行为的一个完整列表：

字符	描述
\	将下一个字符标记为一个特殊字符、或一个原义字符、或一个后向引用、或一个八进制转义符。例如，'n' 匹配字符 "n"。'\n' 匹配一个换行符。序列 '\\' 匹配 "\" 而 "\" 则匹配 "("。
^	匹配输入字符串的开始位置。如果设置了 RegExp 对象的 Multiline 属性，^ 也匹配 '\n' 或 '\r' 之后的位置。
\$	匹配输入字符串的结束位置。如果设置了 RegExp 对象的 Multiline 属性，\$ 也匹配 '\n' 或 '\r' 之前的位置。
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。* 等价于 {0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。
?	匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do" 或 "does" 中的 "do" 。? 等价于 {0,1}。
{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。

<code>{n,}</code>	n 是一个非负整数。至少匹配 n 次。例如, ' <code>o{2,}</code> ' 不能匹配 "Bob" 中的 'o', 但能匹配 "foooooo" 中的所有 o。' <code>o{1,}</code> ' 等价于 ' <code>o+</code> '。' <code>o{0,}</code> ' 则等价于 ' <code>o*</code> '。
<code>{n,m}</code>	m 和 n 均为非负整数, 其中 $n \leq m$ 。最少匹配 n 次且最多匹配 m 次。刘, " <code>o{1,3}</code> " 将匹配 "foooooo" 中的前三个 o。' <code>o{0,1}</code> ' 等价于 ' <code>o?</code> '。请注意在逗号和两个数之间不能有空格。
<code>?</code>	当该字符紧跟在任何一个其他限制符 (<code>*</code> , <code>+</code> , <code>?</code> , <code>{n}</code> , <code>{n,}</code> , <code>{n,m}</code>) 后面时, 匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串, 而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如, 对于字符串 "oooo", ' <code>o+?</code> ' 将匹配单个 "o", 而 ' <code>o+</code> ' 将匹配所有 'o'。
<code>.</code>	匹配除 <code>"\n"</code> 之外的任何单个字符。要匹配包括 <code>"\n"</code> 在内的任何字符, 请使用象 <code>"[\n]"</code> 的模式。
<code>(pattern)</code>	匹配 <i>pattern</i> 并获取这一匹配。所获取的匹配可以从产生的 <code>Matches</code> 集合得到, 在 VBScript 中使用 <code>SubMatches</code> 集合, 在 JScript 中则使用 <code>\$0...\$9</code> 属性。要匹配圆括号字符, 请使用 <code>"\"</code> 或 <code>"\"</code> 。
<code>(?:pattern)</code>	匹配 <i>pattern</i> 但不获取匹配结果, 也就是说这是一个非获取匹配, 不进行存储供以后使用。这在使用 "或" 字符 (<code> </code>) 来组合一个模式的各个部分是很有用。例如, ' <code>industr(?:y ies)</code> ' 就是一个比 ' <code>industry industries</code> ' 更简略的表达式。
<code>(?=pattern)</code>	正向预查, 在任何匹配 <i>pattern</i> 的字符串开始处匹配查找字符串。这是一个非获取匹配, 也就是说, 该匹配不需要获取供以后使用。例如, ' <code>Windows(=95 98 NT 2000)</code> ' 能匹配 "Windows 2000" 中的 "Windows", 但不能匹配 "Windows 3.1" 中的 "Windows"。预查不消耗字符, 也就是说, 在一个匹配发生后, 在最后一次匹配之后立即开始下一次匹配的搜索, 而不是从包含预查的字符之后开始。
<code>(?!pattern)</code>	负向预查, 在任何不匹配 <i>Negative lookahead matches the search string at any point where a string not matching pattern</i> 的字符串开始处匹配查找字符串。这是一个非获取匹配, 也就是说, 该匹配不需要获取供以后使用。例如 ' <code>Windows(?!95 98 NT 2000)</code> ' 能匹配 "Windows 3.1" 中的 "Windows", 但不能匹配 "Windows 2000" 中的 "Windows"。预查不消耗字符, 也就是说, 在一个匹配发生后, 在最后一次匹配之后立即开始下一次匹配的搜索, 而不是从包含预查的字符之后开始
<code>x y</code>	匹配 x 或 y 。例如, ' <code>z food</code> ' 能匹配 "z" 或 "food"。' <code>(z f)ood</code> ' 则匹配 "zood" 或 "food"。
<code>[xyz]</code>	字符集合。匹配所包含的任意一个字符。例如, ' <code>[abc]</code> ' 可以匹配 "plain" 中的 'a'。
<code>[^xyz]</code>	负值字符集合。匹配未包含的任意字符。例如, ' <code>[^abc]</code> ' 可以匹配 "plain" 中的 'p'。
<code>[a-z]</code>	字符范围。匹配指定范围内的任意字符。例如, ' <code>[a-z]</code> ' 可以匹配 'a' 到 'z' 范围内的任意小写字母字符。
<code>[^a-z]</code>	负值字符范围。匹配任何不在指定范围内的任意字符。例如, ' <code>[^a-z]</code> ' 可以匹配任何不在 'a' 到 'z' 范围内的任意字符。
<code>\b</code>	匹配一个单词边界, 也就是指单词和空格间的位置。例如, ' <code>er\b</code> ' 可以匹配 "never" 中的 'er', 但不能匹配 "verb" 中的 'er'。
<code>\B</code>	匹配非单词边界。' <code>er\B</code> ' 能匹配 "verb" 中的 'er', 但不能匹配 "never" 中的 'er'。

<code>\cx</code>	匹配由 <i>x</i> 指明的控制字符。例如, <code>\cM</code> 匹配一个 Control-M 或回车符。 <i>x</i> 的值必须为 A-Z 或 a-z 之一。否则, 将 <i>c</i> 视为一个原义的 'c' 字符。
<code>\d</code>	匹配一个数字字符。等价于 <code>[0-9]</code> 。
<code>\D</code>	匹配一个非数字字符。等价于 <code>[^0-9]</code> 。
<code>\f</code>	匹配一个换页符。等价于 <code>\x0c</code> 和 <code>\cL</code> 。
<code>\n</code>	匹配一个换行符。等价于 <code>\x0a</code> 和 <code>\cJ</code> 。
<code>\r</code>	匹配一个回车符。等价于 <code>\x0d</code> 和 <code>\cM</code> 。
<code>\s</code>	匹配任何空白字符, 包括空格、制表符、换页符等等。等价于 <code>[\f\n\r\t\v]</code> 。
<code>\S</code>	匹配任何非空白字符。等价于 <code>[^\f\n\r\t\v]</code> 。
<code>\t</code>	匹配一个制表符。等价于 <code>\x09</code> 和 <code>\cI</code> 。
<code>\v</code>	匹配一个垂直制表符。等价于 <code>\x0b</code> 和 <code>\cK</code> 。
<code>\w</code>	匹配包括下划线的任何单词字符。等价于 <code>'[A-Za-z0-9_]'</code> 。
<code>\W</code>	匹配任何非单词字符。等价于 <code>'[^A-Za-z0-9_]'</code> 。
<code>\xn</code>	匹配 <i>n</i> , 其中 <i>n</i> 为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如, <code>'\x41'</code> 匹配 "A"。 <code>'\x041'</code> 则等价于 <code>'\x04' & "1"</code> 。正则表达式中可以使用 ASCII 编码。.
<code>\num</code>	匹配 <i>num</i> , 其中 <i>num</i> 是一个正整数。对所获取的匹配的引用。例如, <code>'(.)\1'</code> 匹配两个连续的相同字符。
<code>\n</code>	标识一个八进制转义值或一个后向引用。如果 <code>\n</code> 之前至少 <i>n</i> 个获取的子表达式, 则 <i>n</i> 为后向引用。否则, 如果 <i>n</i> 为八进制数字 (0-7), 则 <i>n</i> 为一个八进制转义值。
<code>\nm</code>	标识一个八进制转义值或一个后向引用。如果 <code>\nm</code> 之前至少有 <i>is preceded by at least nm</i> 个获取得子表达式, 则 <i>nm</i> 为后向引用。如果 <code>\nm</code> 之前至少有 <i>n</i> 个获取, 则 <i>n</i> 为一个后跟文字 <i>m</i> 的后向引用。如果前面的条件都不满足, 若 <i>n</i> 和 <i>m</i> 均为八进制数字 (0-7), 则 <code>\nm</code> 将匹配八进制转义值 <i>nm</i> 。
<code>\nml</code>	如果 <i>n</i> 为八进制数字 (0-3), 且 <i>m</i> 和 <i>l</i> 均为八进制数字 (0-7), 则匹配八进制转义值 <i>nml</i> 。
<code>\un</code>	匹配 <i>n</i> , 其中 <i>n</i> 是一个用四个十六进制数字表示的 Unicode 字符。例如, <code>\u00A9</code> 匹配版权符号 (?)。

1.5.5 建立正则表达式

构造正则表达式的方法和创建数学表达式的方法一样。也就是用多种元字符与操作符将小的表达式结合在一起来创建更大的表达式。

可以通过在一对分隔符之间放入表达式模式的各种组件来构造一个正则表达式。

对 JScript 而言, 分隔符为一对正斜杠 (/) 字符。例如:

```
/expression/
```

对 VBScript 而言, 则采用一对引号 (") 来确定正则表达式的边界。例如:

"expression"

在上面所示的两个示例中，正则表达式模式 (*expression*) 均存储在 **RegExp** 对象的 **Pattern** 属性中。

正则表达式的组件可以是单个的字符、字符集合、字符范围、字符间的选择或者所有这些组件的任意组合。

1.5.6 优先权顺序

在构造正则表达式之后，就可以象数学表达式一样来求值，也就是说，可以从左至右并按照一个优先权顺序来求值。

下表从最高优先级到最低优先级列出各种正则表达式操作符的优先权顺序：

操作符	描述
\	转义符
(), (?:), (?:=), []	圆括号和方括号
*, +, ?, {n}, {n,}, {n,m}	限定符
^, \$, \anymetacharacter	位置和顺序
	“或”操作

1.5.7 普通字符

普通字符由所有那些未显式指定为元字符的打印和非打印字符组成。这包括所有的大写和小写字母字符，所有数字，所有标点符号以及一些符号。

最简单的正则表达式是一个单独的普通字符，可以匹配所搜索字符串中的该字符本身。例如，单字符模式 'A' 可以匹配所搜索字符串中任何位置出现的字母 'A'。这里有一些单字符正则表达式模式的示例：

```
/a/
```

```
/7/
```

```
/M/
```

等价的 VBScript 单字符正则表达式为：

```
"a"
```

```
"7"
```

```
"M"
```

可以将多个单字符组合在一起得到一个较大的表达式。例如，下面的 JScript 正则表达式不是别的，就是通过组合单字符表达式 'a'、'7'以及 'M' 所创建出来的一个

表达式。

/a7M/

等价的 VBScript 表达式为：

"a7M"

请注意这里没有连接操作符。所需要做的就是将一个字符放在了另一个字符后面。

1.5.8 特殊字符

有不少元字符在试图对其进行匹配时需要进行特殊的处理。要匹配这些特殊字符，必须首先将这些字符转义，也就是在前面使用一个反斜杠 (\)。下表给出了这些特殊字符及其含义：

特殊字符	说明
\$	匹配输入字符串的结尾位置。如果设置了 RegExp 对象的 Multiline 属性，则 \$ 也匹配 '\n' 或 '\r'。要匹配 \$ 字符本身，请使用 \\$。
()	标记一个子表达式的开始和结束位置。子表达式可以获取供以后使用。要匹配这些字符，请使用 \() 和 \()。
*	匹配前面的子表达式零次或多次。要匹配 * 字符，请使用 *。
+	匹配前面的子表达式一次或多次。要匹配 + 字符，请使用 \+。
.	匹配除换行符 \n 之外的任何单字符。要匹配 .，请使用 \。
[	标记一个中括号表达式的开始。要匹配 [，请使用 \[。
?	匹配前面的子表达式零次或一次，或指明一个非贪婪限定符。要匹配 ? 字符，请使用 \?。
\	将下一个字符标记为或特殊字符、或原义字符、或后向引用、或八进制转义符。例如，'\n' 匹配字符 '\n'。'\n' 匹配换行符。序列 '\\ 匹配 "\"，而 \" 则匹配 "("。
^	匹配输入字符串的开始位置，除非在方括号表达式中使用，此时它表示不接受该字符集合。要匹配 ^ 字符本身，请使用 \^。
{	标记限定符表达式的开始。要匹配 {，请使用 \{。
	指明两项之间的一个选择。要匹配 ，请使用 \ 。

1.5.9 非打印字符

有不少很有用的非打印字符，偶尔必须使用。下表显示了用来表示这些非打印字符的转义序列：

字符	含义
\cx	匹配由 x 指明的控制字符。例如，\cM 匹配一个 Control-M 或回车符。x 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的 'c' 字符。
\f	匹配一个换页符。等价于 \x0c 和 \cL。

<code>\n</code>	匹配一个换行符。等价于 <code>\x0a</code> 和 <code>\cJ</code> 。
<code>\r</code>	匹配一个回车符。等价于 <code>\x0d</code> 和 <code>\cM</code> 。
<code>\s</code>	匹配任何空白字符，包括空格、制表符、换页符等等。等价于 <code>[\f\n\r\t\v]</code> 。
<code>\S</code>	匹配任何非空白字符。等价于 <code>[^\f\n\r\t\v]</code> 。
<code>\t</code>	匹配一个制表符。等价于 <code>\x09</code> 和 <code>\cI</code> 。
<code>\v</code>	匹配一个垂直制表符。等价于 <code>\x0b</code> 和 <code>\cK</code> 。

1.5.10 字符匹配

句点 (.) 匹配一个字符串中任何单个的打印或非打印字符，除了换行符 (`\n`) 之外。下面的 JScript 正则表达式可以匹配 'aac'、'abc'、'acc'、'adc' 如此等等，同样也可以匹配 'a1c'、'a2c'、'a-c' 以及 'a#c'：

```
/a.c/
```

等价的 VBScript 正则表达式为：

```
"a.c"
```

如果试图匹配一个包含文件名的字符串，其中句点 (.) 是输入字符串的一部分，则可以在正则表达式中的句点前面加上一个反斜杠 (\) 字符来实现这一要求。举例来说，下面的 JScript 正则表达式就能匹配 'filename.ext'：

```
/filename\.ext/
```

对 VBScript 而言，等价的表达式如下所示：

```
"filename\.ext"
```

这些表达式仍然是相当有限的。它们只允许匹配 *任何* 单字符。很多情况下，对从列表中匹配特殊字符十分有用。例如，如果输入文字中包含用数字表示为 Chapter 1, Chapter 2 诸如此类的章节标题，你可能需要找到这些章节标题。

括号表达式

可以在一个方括号 ([和]) 中放入一个或多个单字符，来创建一个待匹配的列表。如果字符被放入括号中括起来，则该列表称为 *括号表达式*。括号内和其他任何地方一样，普通字符代表其本身，也就是说，它们匹配输入文字中出现的一处自己。大多数特殊字符在位于括号表达式中时都将失去其含义。这里有一些例外：

- ']' 字符如果不是第一项，则将结束一个列表。要在列表中匹配 ']' 字符，请将其放在第一项，紧跟在开始的 '[' 后面。
- '\' 仍然作为转义符。要匹配 '\' 字符，请使用 '\\'

括号表达式中所包含的字符只匹配该括号表达式在正则表达式中所处位置的一个单字符。下面的 JScript 正则表达式可以匹配 'Chapter 1'、'Chapter 2'、'Chapter 3'、'Chapter 4' 以及 'Chapter 5'：

```
/Chapter [12345]/
```

在 VBScript 中要匹配同样的章节标题，请使用下面的表达式：

```
"Chapter [12345]"
```

请注意单词 'Chapter' 及后面的空格与括号内的字符的位置关系是固定的。因此，括号表达式只用来指定满足紧跟在单词 'Chapter' 和一个空格之后的单字符位置的字符集合。这里是第九个字符位置。

如果希望使用范围而不是字符本身来表示待匹配的字符，则可以使用连字符将该范围的开始和结束字符分开。每个字符的字符值将决定其在一个范围内的相对顺序。下面的 JScript 正则表达式包含了一个等价于上面所示的括号列表的范围表达式。

```
/Chapter [1-5]/
```

VBScript 中相同功能的表达式如下所示：

```
"Chapter [1-5]"
```

如果以这种方式指定范围，则开始和结束值都包括在该范围内。有一点特别需要注意的是，在 Unicode 排序中起始值一定要在结束值之前。

如果想在括号表达式中包括连字符，则必须使用下述方法之一：

- 使用反斜杠将其转义：

```
[\-]
```

- 将连字符放在括号列表的开始和结束位置。下面的表达式能匹配所有的小写字母和连字符：

```
[-a-z]
[a-z-]
```

- 创建一个范围，其中开始字符的值小于连字符，而结束字符的值等于或大于连字符。下面两个正则表达式都满足这一要求：

```
[!--]
[!~]
```

同样，通过在列表开始处放置一个插入符(^)，就可以查找所有不在列表或范围中的字符。如果该插入符出现在列表的其他位置，则匹配其本身，没有任何特殊含义。下面的 JScript 正则表达式匹配章节号大于 5 的章节标题：

```
/Chapter [^12345]/
```

对 VBScript 则使用：

```
"Chapter [^12345]"
```

在上面所示的示例中，表达式将匹配第九个位置处除 1, 2, 3, 4, or 5 之外的任何数字字符。因此，'Chapter 7' 为一个匹配，同样 'Chapter 9' 也是如此。

上面的表达式可以使用连字符 (-) 表示。对 JScript 为：

```
/Chapter [^1-5]/
```

或者，对 VBScript 为：

```
"Chapter [^1-5]"
```

括号表达式的典型用法是指定对任何大写或小写字母字符或任何数字的匹配。下面的 JScript 表达式给出了这一匹配：

```
/[A-Za-z0-9]/
```

等价的 VBScript 表达式为：

```
"[A-Za-z0-9]"
```

1.5.11 限定符

有时候不知道要匹配多少字符。为了能适应这种不确定性，正则表达式支持限定符的概念。这些限定符可以指定正则表达式的一个给定组件必须要出现多少次才能满足匹配。

下表给出了各种限定符及其含义的说明：

字符	描述
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。 * 等价于 {0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。 + 等价于 {1,}。
?	匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do" 或 "does" 中的 "do" 。 ? 等价于 {0,1}。
{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
{n,}	n 是一个非负整数。至少匹配 n 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "foooooo" 中的所有 o。'o{1,}' 等价于 'o+'。'o{0,}' 则等价于 'o*'。
{n,m}	m 和 n 均为非负整数，其中 n <= m。最少匹配 n 次且最多匹配 m 次。刘，"o{1,3}" 将匹配 "foooooo" 中的前三个 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格。

对一个很大的输入文档而言，章节数很轻易就超过九章，因此需要有一种方法来处理两位数或者三位数的章节号。限定符就提供了这个功能。下面的 JScript 正则表达式可以匹配具有任何位数的章节标题：

```
/Chapter [1-9][0-9]*/
```

下面的 VBScript 正则表达式执行同样的匹配：

```
"Chapter [1-9][0-9]*"
```

请注意限定符出现在范围表达式之后。因此，它将应用于所包含的整个范围表达式，在本例中，只指定了从 0 到 9 的数字。

这里没有使用 '+' 限定符，因为第二位或后续位置上并不一定需要一个数字。同样也没有使用 '?' 字符，因为这将把章节数限制为只有两位数字。在 'Chapter' 和空格字符之后至少要匹配一个数字。

如果已知章节数限制只有 99 章，则可以使用下面的 JScript 表达式来指定至少有一位数字，但不超过两个数字。

```
/Chapter [0-9]{1,2}/
```

对 VBScript 可以使用下述正则表达式：

```
"Chapter [0-9]{1,2}"
```

上述表达式的缺点是如果有一个章节号大于 99，它仍只会匹配前两位数字。另一个缺点是某些人可以创建一个 Chapter0，而且仍能匹配。一个更好的用来匹配两位数的 JScript 表达式如下：

```
/Chapter [1-9][0-9]?/
```

或者

```
/Chapter [1-9][0-9]{0,1}/
```

对 VBScript 而言，下述表达式与上面等价：

```
"Chapter [1-9][0-9]?"
```

或者

```
"Chapter [1-9][0-9]{0,1}"
```

'*'、'+' 和 '?' 限定符都称之为 *贪婪的*，也就是说，他们尽可能多地匹配文字。有时这根本就不是所希望发生的情况。有时则正好希望最小匹配。

例如，你可能要搜索一个 HTML 文档来查找一处包含在 H1 标记中的章节标题。在文档中该文字可能具有如下形式：

```
<H1>Chapter 1 - Introduction to Regular
Expressions</H1>
```

下面的表达式匹配从开始的小于号 (<) 到 H1 标记结束处的大于号之间的所有内容。

```
/<.*>/
```

VBScript 的正则表达式为：

```
"<.*>"
```

如果所要匹配的就是开始的 H1 标记，则下述非贪婪地表达式就只匹配 <H1>。

```
/<.*?>/
```

或者

```
"<.*?>"
```

通过在 '*'、'+' 或 '?' 限定符后放置 '?', 该表达式就从贪婪匹配转为了非贪婪或最小匹配。

1.5.12 定位符

到现在为止, 所看到的示例都只考虑查找任何地方出现的章节标题。出现的任何一个字符串 'Chapter' 后跟一个空格和一个数字可能是一个真正的章节标题, 也可能是对其他章节的交叉引用。由于真正的章节标题总是出现在一行的开始, 因此需要设计一个方法只查找标题而不查找交叉引用。

定位符提供了这个功能。定位符可以将一个正则表达式固定在一行的开始或结束。也可以创建只在单词内或只在单词的开始或结尾处出现的正则表达式。下表包含了正则表达式及其含义的列表:

字符	描述
^	匹配输入字符串的开始位置。如果设置了 RegExp 对象的 Multiline 属性, ^ 也匹配 '\n' 或 '\r' 之后的位置。
\$	匹配输入字符串的结束位置。如果设置了 RegExp 对象的 Multiline 属性, \$ 也匹配 '\n' 或 '\r' 之前的位置。
\b	匹配一个单词边界, 也就是指单词和空格间的位置。
\B	匹配非单词边界。

不能对定位符使用限定符。因为在一个换行符或者单词边界的前面或后面不会有连续多个位置, 因此诸如 '^*' 的表达式是不允许的。

要匹配一行文字开始位置的字符, 请在正则表达式的开始处使用 '^' 字符。不要把 '^' 的这个语法与其在括号表达式中的语法弄混。它们的语法根本不同。

要匹配一行文字结束位置的字符, 请在正则表达式的结束处使用 '\$' 字符。

要在查找章节标题时使用定位符, 下面的 JScript 正则表达式将匹配位于一行的开始处最多有两个数字的章节标题:

```
/^Chapter [1-9][0-9]{0,1}/
```

VBScript 中相同功能的正则表达式如下:

```
"^Chapter [1-9][0-9]{0,1}"
```

一个真正的章节标题不仅出现在一行的开始, 而且这一行中也仅有这一个内容, 因此, 它必然也位于一行的结束。下面的表达式确保所指定的匹配只匹配章节而不会匹配交叉引用。它是通过创建一个只匹配一行文字的开始和结束位置的正则表达式来实现的。

```
/^Chapter [1-9][0-9]{0,1}$/
```

对 VBScript 则使用:

```
"^Chapter [1-9][0-9]{0,1}$"
```

匹配单词边界有少许不同，但却给正则表达式增加了一个非常重要的功能。单词边界就是单词和空格之间的位置。非单词边界就是其他任何位置。下面的 JScript 表达式将匹配单词 'Chapter' 的前三个字符，因为它们出现在单词边界后：

```
/\bCha/
```

对 VBScript 为：

```
"\bCha"
```

这里 '\b' 操作符的位置很关键。如果它位于要匹配的字符串的开始，则将查找位于单词开头处的匹配；如果它位于字符串的末尾，则查找位于单词结束处的匹配。例如，下面的表达式将匹配单词 'Chapter' 中的 'ter'，因为它出现在单词边界之前：

```
/ter\b/
```

以及

```
"ter\b"
```

下面的表达式将匹配 'apt'，因为它位于 'Chapter' 中间，但不会匹配 'aptitude' 中的 'apt'：

```
/\Bapt/
```

以及

```
"\Bapt"
```

这是因为在单词 'Chapter' 中 'apt' 出现在非单词边界位置，而在单词 'aptitude' 中位于单词边界位置。非单词边界操作符的位置不重要，因为匹配与一个单词的开头或结尾无关。

1.5.13 选择和编组

选择允许使用 '|' 字符来在两个或多个候选项中进行选择。通过扩展章节标题的正则表达式，可以将其扩充为不仅仅适用于章节标题的表达式。不过，这可没有想象的那么直接。在使用选择时，将匹配 '|' 字符每边最可能的表达式。你可能认为下面的 JScript 和 VBScript 表达式将匹配位于一行的开始和结束位置且后跟一个或两个数字的 'Chapter' 或 'Section'：

```
/^Chapter|Section [1-9][0-9]{0,1}$/
```

```
"^Chapter|Section [1-9][0-9]{0,1}$"
```

不幸的是，真正的情况是上面所示的正则表达式要么匹配位于一行开始处的单词 'Chapter'，要么匹配一行结束处的后跟任何数字的 'Section'。如果输入字符串为 'Chapter 22'，上面的表达式将只匹配单词 'Chapter'。如果输入字符串为 'Section 22'，

则该表达式将匹配 'Section 22'。但这种结果不是我们此处的目的，因此必须有一种办法来使正则表达式对于所要做的更易于响应，而且确实也有这种方法。

可以使用圆括号来限制选择的范围，也就是说明该选择只适用于这两个单词 'Chapter' 和 'Section'。不过，圆括号同样也是难处理的，因为它们也用来创建子表达式，有些内容将在后面关于子表达式的部分介绍。通过采用上面所示的正则表达式并在适当位置添加圆括号，就可以使该正则表达式既可以匹配 'Chapter 1'，也可以匹配 'Section 3'。

下面的正则表达式使用圆括号将 'Chapter' 和 'Section' 组成一组，所以该表达式才能正确工作。对 JScript 为：

```
/^(Chapter|Section) [1-9][0-9]{0,1}$/
```

对 VBScript 为：

```
"^(Chapter|Section) [1-9][0-9]{0,1}$"
```

这些表达式工作正确，只是产生了一个有趣的副产品。在 'Chapter|Section' 两边放置圆括号建立了适当的编组，但也导致两个待匹配单词之一都被捕获供今后使用。由于在上面所示的表达式中只有一组圆括号，因此只能有一个捕获的 *submatch*。可以使用 VBScript 的 **Submatches** 集合或者 JScript 中 **RegExp** 对象的 **\$1-\$9** 属性来引用这个子匹配。

有时捕获一个子匹配是所希望的，有时则是不希望的。在说明所示的示例中，真正想做的就是使用圆括号对单词 'Chapter' 或 'Section' 之间的选择编组。并不希望在后面再引用该匹配。实际上，除非真的是需要捕获子匹配，否则请不要使用。由于不需要花时间和内存来存储那些子匹配，这种正则表达式的效率将更高。

可以在正则表达式模式圆括号内部的前面使用 '?' 来防止存储该匹配供今后使用。对上面所示正则表达式的下述修改提供了免除子匹配存储的相同功能。对 JScript：

```
/^(?:Chapter|Section) [1-9][0-9]{0,1}$/
```

对 VBScript：

```
"^(?:Chapter|Section) [1-9][0-9]{0,1}$"
```

除了 '?' 元字符，还有两个非捕获元字符用于称之为 *预查* 的匹配。一个为正向预查，用 **?=** 表示，在任何开始匹配圆括号内的正则表达式模式的位置来匹配搜索字符串。一个为负向预查，用 **?!** 表示，在任何开始不匹配该正则表达式模式的位置来匹配搜索字符串。

例如，假定有一个包含引用有 Windows 3.1、Windows 95、Windows 98 以及 Windows NT 的文档。进一步假设需要更新该文档，方法是查找所有对 Windows 95、Windows 98 以及 Windows NT 的引用，并将这些引用更改为 Windows 2000。可以使用下面的 JScript 正则表达式，这是一个正向预查，来匹配 Windows 95、Windows 98 以及 Windows NT：

```
/Windows(?=95 |98 |NT )/
```

在 VBScript 要进行同样的匹配可以使用下述表达式：

```
"Windows(?=95 |98 |NT )" 
```

找到一个匹配后，紧接匹配到的文字（而不包括预查中使用的字符）就开始对下一次匹配的搜索。例如，如果上面所示的表达式匹配到 'Windows 98'，则将从 'Windows' 而不是 '98' 之后继续查找。

1.5.14 后向引用

正则表达式一个最重要的特性就是将匹配成功的模式的某部分进行存储供以后使用这一能力。请回想一下，对一个正则表达式模式或部分模式两边添加圆括号将导致这部分表达式存储到一个临时缓冲区中。可以使用非捕获元字符 '?:'、'?='，或 '?!' 来忽略对这部分正则表达式的保存。

所捕获的每个子匹配都按照在正则表达式模式中从左至右所遇到的内容存储。存储子匹配的缓冲区编号从 1 开始，连续编号直至最大 99 个子表达式。每个缓冲区都可以使用 '\n' 访问，其中 *n* 为一个标识特定缓冲区的一位或两位十进制数。后向引用一个最简单，最有用的应用是提供了确定文字中连续出现两个相同单词的位置的能力。请看下面的句子：

Is is the cost of of gasoline going up
up?

根据所写内容，上面的句子明显存在单词多次重复的问题。如果能有一种方法无需查找每个单词的重复现象就能修改该句子就好了。下面的 JScript 正则表达式使用一个子表达式就可以实现这一功能。

```
/\b([a-z]+) \1\b/gi
```

等价的 VBScript 表达式为：

```
"\b([a-z]+) \1\b"
```

在这个示例中，子表达式就是圆括号之间的每一项。所捕获的表达式包括一个或多个字母字符，即由 '[a-z]+' 所指定的。该正则表达式的第二部分是对前面所捕获的子匹配的引用，也就是由附加表达式所匹配的第二次出现的单词。'\1' 用来指定第一个子匹配。单词边界元字符确保只检测单独的单词。如果不这样，则诸如 "is issued" 或 "this is" 这样的短语都会被该表达式不正确地识别。

在 JScript 表达式中，正则表达式后面的全局标志 ('g') 表示该表达式将用来在输入字符串中查找尽可能多的匹配。大小写敏感性由表达式结束处的大小写敏感性标记 ('i') 指定。多行标记指定可能出现在换行符的两端的潜在匹配。对 VBScript 而言，在表达式中不能设置各种标记，但必须使用 **RegExp** 对象的属性来显式设置。

使用上面所示的正则表达式，下面的 JScript 代码可以使用子匹配信息，在一个文字字符串中将连续出现两次的相同单词替换为一个相同的单词：

```
var ss = "Is is the cost of of gasoline  
going up up?.\n";
```



```
var re = /\b([a-z]+) \1\b/gim;           //
```

创建正则表达式样式。

```
var rv = ss.replace(re, "$1");           //
```

用一个单词替代两个单词。

最接近的等价 VBScript 代码如下：

```
Dim ss, re, rv

ss = "Is is the cost of of gasoline going
up up?." & vbNewLine

Set re = New RegExp
re.Pattern = "\b([a-z]+) \1\b"
re.Global = True
re.IgnoreCase = True
re.Multiline = True

rv = re.Replace(ss, "$1")
```

请注意在 VBScript 代码中，全局、大小写敏感性以及多行标记都是使用 **RegExp** 对象的适当属性来设置的。

在 **replace** 方法中使用 **\$1** 来引用所保存的第一个子匹配。如果有多个子匹配，则可以用 **\$2**, **\$3** 等继续引用。

后向引用的另一个用途是将一个通用资源指示符 (URI) 分解为组件部分。假定希望将下述的 URI 分解为协议 (ftp, http, etc)，域名地址以及页面/路径：

```
http://msdn.microsoft.com:80/scripting/
default.htm
```

下面的正则表达式可以提供这个功能。对 JScript，为：

```
/(\w+):\/\/([^\:]+)(:\d*)?([^\# ]*)/
```

对 VBScript 为：

```
"(\w+):\/\/([^\:]+)(:\d*)?([^\# ]*)"
```

第一个附加子表达式是用来捕获该 web 地址的协议部分。该子表达式匹配位于一

个冒号和两个正斜杠之前的任何单词。第二个附加子表达式捕获该地址的域名地址。该子表达式匹配不包括 '\、\' 或 ':' 字符的任何字符序列。第三个附加子表达式捕获网站端口号码，如果指定了该端口号。该子表达式匹配后跟一个冒号的零或多个数字。最后，第四个附加子表达式捕获由该 web 地址指定的路径以及\或者页面信息。该子表达式匹配一个和多个除'#' 或空格之外的字符。将该正则表达式应用于上面所示的 URI 后，子匹配包含下述内容：

RegExp. \$1 包含 "http"

RegExp. \$2 包含 "msdn.microsoft.com"

RegExp. \$3 包含 ":80"

RegExp. \$4 包含 "/scripting/default.htm"

2. Addenda

2.1 版权所有

Microsoft® JScript®

本文档中的信息，包括 URL 和其他 Internet Web 站点参考，如有变更，恕不另行通知。除非特别说明，其中提及的示例公司、机构、产品、姓名和事件均属虚构。如与真实公司、机构、产品、姓名和事件雷同，纯属巧合。遵守任何适用的版权法是用户的责任。在不限版权所辖权利的前提下，未经 Microsoft Corporation 的明确书面许可，无论出于何种目的，均不得以任何形式或借助任何手段（电子、机械、影印或其他手段）复制或传播文档中的任何部分，或将其存储于或引入检索系统。

Microsoft 可能拥有对本文档内容的专利、专利申请、商标、版权或其他知识产权。除了任何 Microsoft 授权许可协议所提供的明确书面许可，拥有本文档并不赋予您任何有关这些专利、专利申请、商标、版权或其他知识产权的许可。

© 1991-2000 Microsoft Corporation。保留所有权利。

Microsoft、MS、MS-DOS、ActiveX、JScript、Microsoft Press、MSDN、Visual Basic、Windows、Windows NT、Win32、和 Win32s 均系 Microsoft Corporation 在美国或其他国家（或地区）的注册商标或商标。

其中提到的其他真实公司及产品名称可能是其各自所有者的商标。

2.2 必须给数组长度指定一个有穷正整数

在对现有 **Array** 对象设置其 **length** 属性时，所指定的数组长度不是正整数，也不是零。如果给某个 **Array** 对象的 **length** 属性所赋的值时为负数或不是数值 (**NaN**) 时，就会产生这个错误。请注意 JScript 会自动将小数数值转换为整数。

要改正此错误

将一个正整数赋给 **length** 属性。数组大小没有上界限制，但不能超过整数的最大值（大约为 40 亿）。下面的示例演示了给 **Array** 对象的 **length** 属性赋值的正确方法。

```
var my_array = new Array();  
my_array.length = 99;
```

请参阅

[使用数组](#)(See 1.1.2.5) | [JScript 语言参考](#)(See 2.61)

2.3 数组长度必须是一个有穷正整数

在调用 **Array** 构造函数时使用的参数不是自然数（所谓自然数就是零加上正整数集合）。

要改正此错误

在创建一个新的 **Array** 对象时只能使用正整数。如果要创建一个只有一个元素且不是整数的数组，请执行下面两步操作。首先，创建一个只有一个元素的数组，然后将这个值赋给第一个元素 (**array[0]**)。下面的示例将产生这个错误。

```
var piArray = new Array(3.14159);
```

下面的示例演示了指定一个只有单个数值元素的数组的正确方法。

```
var piArray = new Array(1);  
piArray [0] = 3.14159;
```

数组大小没有上界限制，但不能超过整数的最大值（大约为 40 亿）。

请参阅

[使用数组](#)(See 1.1.2.5) | [JScript 语言参考](#)(See 2.61)

2.4 需要 Array 或 arguments 对象

没有将数组作为参数。这个错误只在 **Function.prototype.apply** 方法中发生。如果使用这个函数，则该函数的第二个参数要么是 **Array** 对象，要么是 **Arguments** 对象。

要改正此错误

- 指定一个 **Array** 或 **Arguments** 对象作为第二个参数。

请参阅

[使用数组](#)(See 1.1.2.5) | [apply 方法](#)(See 1.2.7.1.4) | [JScript 函数](#)(See 1.1.1.7) | [JScript 语言参考](#)(See 2.61)

2.5 需要布尔变量

试图用一个不是 **Boolean** 类型的对象来调用 **Boolean.prototype.toString** 或者 **Boolean.prototype.valueOf** 方法。这种调用类型的对象必须是 **Boolean** 类型。例如：

```
var o = new Object;

o.f = Boolean.prototype.toString;

o.f();
```

要改正此错误

- 只使用 **Boolean** 类型的对象调用 **Boolean.prototype.toString** 或 **Boolean.prototype.valueOf** 方法。

请参阅

[Boolean 对象](#)(See 1.2.8.3) | [JScript 数据类型](#)(See 1.1.1.4) | [控制程序流](#)(See 1.1.1.6) | [拷贝、传递和比较数据](#)(See 1.1.2.4)

2.6 不能赋值给函数结果

试图将一个值赋给函数结果。可以将函数结果赋给某个变量，但其本身不能用作

变量。如果希望将一个新值赋给该函数本身，则不要圆括号（函数调用操作符）。下面的示例演示了产生这一错误的情形。

```
myFunction() = 42; // 试图强将 42 赋给该函数调用的结果。
```

要改正此错误

- 不要使用函数调用的值作为赋值对象。可以将函数调用的结果赋给某个变量。

```
myVar = myFunction(42);
```

- 或者，可以将函数本身（不是其返回数值）赋给一个变量。

```
myFunction = new Function("return 42;");
```

请参阅

[Function 对象](#)(See 1.2.8.7) | [编写 JScript 代码](#)(See 1.1.1.2) | [JScript 函数](#)(See 1.1.1.7) | [JScript 方法](#)(See 1.2.7)

2.7 不能赋值给 'this'

试图给 **this** 赋值，**this** 是 JScript 关键字，表示：

- 当前正在执行某个方法的对象，
- 如果没有当前方法（或该方法不属于任何其它对象），则是指全局对象。

所谓方法就是由对象调用的 JScript 函数。在一个方法内，**this** 关键字代表调用该方法的对象的引用（这种情况下，就是指使用 **new** 操作符调用类构造函数所创建的对象）。

在一个方法内可以使用 **this** 来引用当前对象，但不能对其赋予新的值。

要改正此错误

- 不要试图对 **this** 赋值。要访问某个实例化对象的属性或方法，请使用点操作符（例如 `circle.radius`）。

注意 不能将用户创建的变量命名为 **this**；这是一个 JScript 保留字。

请参阅

[This](#)(See 1.2.11.15) | [脚本查错](#)(See 1.1.2.7) | [JScript 语言参考](#)(See 2.61)

2.8 在循环外不能有 'break'

试图在循环外使用 **break** 关键词。**break** 关键词用来结束一个循环或 **switch** 语句。它必须位于循环体内或 **switch** 语句内。

例外

break 关键字后面可以带一个标签。

```
break labelname;
```

只有在嵌套式循环或 **switch** 语句中并且需要跳出的循环不是最内层循环时，才需要使用这种带标签形式的 **break** 关键词。

要改正此错误

- 请确保 **break** 关键词位于封闭的循环或 **switch** 语句内。

请参阅

[Break](#)(See 1.2.11.4) | [控制程序流](#)(See 1.1.1.6) | [脚本查错](#)(See 1.1.2.7) | [JScript 语言参考](#)(See 2.61)

2.9 在循环外不能有 'continue'

试图在循环外使用 **continue** 语句。只有在下述语句内才可以使用 **continue** 语句：

- **do-while** 循环，
- **while** 循环，
- **for** 循环，
- **for/in** 循环。

要改正此错误

请确保 **continue** 语句位于下述语句的内部：

- **do-while** 循环，
- **while** 循环，
- **for** 循环，
- **for/in** 循环。

请参阅

[Continue](#)(See 1.2.11.6) | [控制程序流](#)(See 1.1.1.6) | [脚本查错](#)(See 1.1.2.7) | [JScript 语言参考](#)(See 2.61)

2.10 条件编译已关闭

在没有先启用条件编译的情况下试图使用条件编译变量。打开条件编译将告诉 JScript 编译器要解释那些以 @ 开头作为条件编译变量的标识符。使用下面的语句作为条件代码的起始语句就可以打开条件编译：

```
/*@cc_on @*/
```

要改正此错误

- 在条件代码的起始处添加下面的语句：
- `/*@cc_on @*/`

请参阅

[条件编译](#)(See 1.1.2.8) | [条件编译变量](#)(See 1.1.2.9) | [@cc_on](#)(See 1.2.11.1) | [@if 语句](#)(See 1.2.11.2) | [@set 语句](#)(See 1.2.11.3)

2.11 需要 Date 对象

试图对一个不是 **Date** 类型的对象调用 **Date.prototype.toString** 或 **Date.prototype.valueOf** 方法。这种调用类型的对象必须是 **Date** 类型。例如：

```
var o = new Object;  
  
o.f = Date.prototype.toString;  
  
o.f();
```

要改正此错误

- 只能对 **Date** 类型的对象调用 **Date.prototype.toString** 或者 **Date.prototype.valueOf** 方法。

请参阅

[Date 对象](#)(See 1.2.8.4) | [getDate 方法](#)(See 1.2.7.2.6) | [内部对象](#)(See 1.1.1.8.2)

2.12 一条 'switch' 语句中只能有一个 'default'

在一个 switch 语句中试图使用多个 **default** 语句。Default 分支总是位于 switch 语句的最后一个分支语句（也就是匹配失败的情形）。

要改正此错误

- 从 switch 语句中删除任何多余的 **default** 分支语句（在 switch 语句中使用了多个 default 语句）。

请参阅

[Switch](#)(See 1.2.11.14) | [控制程序流](#)(See 1.1.1.6) | [JScript 保留的关键字](#)(See 1.1.1.9)

2.13 需要 Enumerator 对象

试图对一个非 **Enumerator** 类型的对象调用 **Enumerator.prototype.atEnd**、**Enumerator.prototype.item**、**Enumerator.prototype.moveFirst** 或者 **Enumerator.prototype.moveNext** 方法。这种调用类型的对象必须是 **Enumerator** 类型。下面是违反这个规则的代码示例：

```
var o = new Object;  
  
o.f = Enumerator.prototype.atEnd;  
  
o.f();
```

要改正此错误

- 只能对 **Enumerator** 类型的对象调用其 **Enumerator.prototype.atEnd**、**Enumerator.prototype.item**、**Enumerator.prototype.moveFirst** 或者 **Enumerator.prototype.moveNext** 方法。要查找某个对象是否为 **Enumerator** 对象，请使用：

```
if(x instanceof Enumerator)
```

请参阅

[Enumerator 对象](#)(See 1.2.8.5) | [Files 集合](#)(See 1.4.2.5) | [Folders 集合](#)(See 1.4.2.8) | [Drives 集合](#)(See 1.4.2.3)

2.14 异常抛出，但无法抓住

在代码中包括了一条 **throw** 语句，但没有将其包含在某个 **try** 语句块内，或者没有相关的 **catch** 语句块来捕获错误。异常是在 **try** 语句块中使用 **throw** 语句被抛出，并且应该由 **try** 语句块外的 **catch** 语句抓住。

要改正此错误

- 用 **try** 语句块将可能产生异常的代码封住，并确保外面有一个相应的 **catch** 语句块。
- 确认 **catch** 语句中为正确的异常格式。
- 如果异常被再次抛出，确认存在相应的 **catch** 语句。

请参阅

[Error 对象](#)(See 1.2.8.6) | [throw 语句](#)(See 1.2.11.16) | [try...catch 语句](#)(See 1.2.11.17)

2.15 缺少 '('

试图用一组圆括号来括住一个表达式，但缺少开始圆括号。有些表达式必须用一组开始圆括号和结束圆括号括住。

```
for (initialize; test; increment) {  
statement;  
}
```

要改正此错误

- 给求值表达式添加左圆括号。

请参阅

[JScript 语言参考](#)(See 2.61)

2.16 缺少 ']'

引用数组元素时缺少右中括号。引用数组元素的任何表达式都必须包括左中括号和右中括号。

要改正此错误

- 给引用该数组无素的表达式添加右中括号。

请参阅

[使用数组](#)(See 1.1.2.5)| [Array 对象](#)(See 1.2.8.2)

2.17 正则表达式中缺少 ']'

试图创建一个用于正则表达式匹配的字符类，但是没有右中括号。单个的文字字符组合可以用括号括起来表示一个字符类。一个字符类可以匹配其中包含的任一字符。例如，/[abc]/ 可以匹配字母 “a”，“b”，或 “c” 的任何一个。

要改正此错误

- 给正则表达式加上右中括号。

注意 如果想匹配单中括号，可以使用反斜杠将其转义 - \[- 这样就不会被 JScript 解释为特殊字符。)

请参阅

[Regular Expression 对象](#)(See 1.2.8.13)| [Regular Expression 语法](#)(See 1.5.4)

2.18 缺少 '{'

没有用左大括号来标记函数体的开始位置。对于组成函数体的代码，即使只有一行，也必须包含在左大括号和右大括号中。请注意函数体对大括号的使用要比循环中的使用更严格。

要改正此错误

- 添加左大括号，标记函数体的开始。

请参阅

[Function 对象](#)(See 1.2.8.7)| [JScript 语言参考](#)(See 2.61)

2.19 缺少 ')'

试图用一组圆括号来括住一个表达式，但缺少结束圆括号。有些表达式必须用一组开始圆括号和结束圆括号括住。

```
for (initialize; test; increment) {  
    statement;  
}
```

要改正此错误

- 给求值表达式添加右圆括号。

请参阅

[JScript 语言参考](#)(See 2.61)

2.20 缺少 '/'

试图创建一个正则表达式文字，但没有包括其中的斜杠 (/)。正如字符串文字表示为一对单引号内的字符一样，正则表达式文字则表示为一对斜杠 (/) 内的字符。

要改正此错误

- 插入一个结束斜杠，标志正则表达式的结束位置。

请参阅

[Regular Expression 对象](#)(See 1.2.8.13) | [Regular Expression 语法](#)(See 1.5.4)

2.21 缺少 ':'

试图创建一个使用三目条件运算符的表达式，但是在第二操作数和第三操作数之间没有包括冒号。三目（三个操作数）条件操作符在第二个（为真）和第三个（为假）操作数之间一定要有一个冒号。

要改正此错误

- 在第二个和第三个操作数之间插入一个冒号。

请参阅

[条件（三目）操作符 \(?:\)](#)(See 1.2.9.17) | [JScript 的操作符](#)(See 1.1.1.5) | [JScript 语言参考](#)(See 2.61)

2.22 缺少 ';'

试图在一行内写多条语句，但没有用分号将各条语句加以分隔。分号是用来结束语句的。可以在一行内写多条语句，不过每条语句后都必须加分号以便分开。

要改正此错误

- 在每条语句后写一个分号。
- 确保函数调用正确使用圆括号。
- 确保没有忘记 **for** 循环头部中的分号。

请参阅

[编写 JScript 代码](#)(See 1.1.1.2) | [JScript 语言参考](#)(See 2.61)

2.23 缺少 '@'

试图使用 `@set` 语句创建一个用于条件编译语句的变量，但在变量名称前没有 `@` 符号。

要改正此错误

- 立即在变量名之前添加一个 `@` 符号。例如：

```
@set @myvar = 1
```

请参阅

[@set 语句](#)(See 1.2.11.3) | [条件编译](#)(See 1.1.2.8) | [条件编译变量](#)(See 1.1.2.9)

2.24 缺少 '}'

没有用右大括号来标记函数体、循环、代码块或者对象初始化软件的结束位置。这种错误的一个示例是：**for** 循环只有标记循环体起点的左括号。

要改正此错误

- 添加右大括号，标记函数、循环、语句块、或者对象初始化的结束位置。

请参阅

[Function 对象](#)(See 1.2.8.7) | [控制程序流](#)(See 1.1.1.6) | [JScript 语言参考](#)(See 2.61)

2.25 缺少 '='

试图创建一个供条件编译语句使用的变量，但在变量和要赋给该变量的值之间缺少一个等号。

要改正此错误

- 添加一个等号。例如：

```
@set @myvar1 = 1
```

请参阅

[条件编译](#)(See 1.1.2.8) | [条件编译变量](#)(See 1.1.2.9)

2.26 缺少 'catch'

使用了处理异常的 **try** 语句块，但没有相应的 **catch** 语句。异常处理机制需要将那些可能失败的代码，连同当异常发生后不会执行的代码一起，包在一个 **try** 语句块中。发生异常时通过使用 **throw** 语句将异常从 **try** 语句块中抛出，并被 **try** 语句块外的一个或多个 **catch** 语句捕获。

要改正此错误

- 添加相应的 **catch** 语句块。
- 试试使用 **finally** 语句块而不使用 **catch** 语句块。

请参阅

[try...catch 语句](#)(See 1.2.11.17) | [Error 对象](#)(See 1.2.8.6)

2.27 需要常量

试图在条件编译测试语句中使用一个（非条件编译）变量。条件编译测试语句的值必须为一个常数。

要改正此错误

- 用文字替代该变量。
- 用条件编译变量替代该变量。

请参阅

[条件编译](#)(See 1.1.2.8) | [条件编译变量](#)(See 1.1.2.9)

2.28 缺少 '@end'

试图创建一个条件编译的代码块，但在最后没有 **@end** 语句。通过把 JScript 语句放在 **@if/@end** 块内，可以实现条件编译。

要改正此错误

- 添加相应的 **@end** 语句。

请参阅

[条件编译](#)(See 1.1.2.8) | [条件编译变量](#)(See 1.1.2.9)

2.29 需要十六进制数

创建了一个不正确的 Unicode 转义序列。Unicode 转义序列以 **\u** 开头，后跟整整四位十六进制数（不多，也不少）。Unicode 十六进制数不仅包含数字 **0-9**，大写字母 **A-F**，还有小写字母 **a-f**。下面的示例演示了正确形成的 Unicode 转义序列。

```
z = "\u1A5F";
```

要改正此错误

- 请确保 Unicode 十六进制数是以 **\u** 开头的，只包含数字 **0-9**，大写字母 **A-F**、小写字母 **a-f**；并组成四个数字。

注意 如果希望在字符串中使用文字文本 **t \u**，请使用两个短斜杠 **_(\\u)**
- 一个用来将第一个反斜杠转义。

请参阅

[JScript 语言参考](#)(See 2.61) | [JScript 数据类型](#)(See 1.1.1.4)

2.30 需要标识符

在需要标识符的上下文中，使用了不是标识符的内容。标识符可以是：

- 变量，
- 属性，
- 数组，
- 函数名称。

要改正此错误

- 更改该表达式，使标识符出现在等号的左边。

请参阅

[JScript 语言参考](#)(See 2.61) | [使用数组](#)(See 1.1.2.5)

2.31 需要标识符、字符串或者数字

使用了不正确的文字语法来声明对象文字。对象文字的属性必须是标识符、字符串或一个数。对象文字(也称为“对象初始化程序”)由用逗号分割的属性:值对组成，所有值对都用括号括起来。例如：

```
var point = {x:1.2, y:-3.4};
```

要改正此错误

- 请确保使用了正确的文字语法。

请参阅

[逗号操作符 \(,\)](#)(See 1.2.9.14)

2.32 正则表达式中缺少 ')'

试图创建一个正则表达式捕获、断言和组合，但不包括结束圆括号。正则表达式中的圆括号有多个用途。首先，它们主要用来抓捕子表达式，指定断言、或将模式组合，从而通过 *、+、? 等等使这些项被看作一个单一的单位。

要改正此错误

- 加上最右边的结束圆括号。

注意 如果想匹配单括号，可以使用反斜杠将其转义 - \`(` - 这样就不会被 JScript 解释为特殊字符。)

请参阅

[Regular Expression 对象](#)(See 1.2.8.13) | [Regular Expression 语法](#)(See 1.5.4)

2.33 缺少 'while'

在 `do ... while` 循环中没有包含 `while` 条件。一条 `do` 语句必须有相应的 `while` 测试。

要改正此错误

- 在循环体结束处加上 `while` 测试语句。

请参阅

[while 语句](#)(See 1.2.11.19) | [控制程序流](#)(See 1.1.1.6) | [JScript 语言参考](#)(See 2.61)

2.34 函数没有合法的 `prototype` 对象

试图使用 `instanceof` 来确定一个对象是否是从某个特定函数类派生而来,但重新定义的该对象的 `prototype` 属性要么是空的,要么是外部对象类型(都不是有效的 JScript 对象)。外部对象可能来自主机对象模型(例如, Internet Explorer 的产品和 window 产品, 或一个外部的 COM 对象)。

要改正此错误

- 确保函数的 `prototype` 属性引用了合法的 JScript。

请参阅

[Function 对象](#)(See 1.2.8.7) | [prototype 属性](#)(See 1.2.10.34)

2.35 需要 Function 对象

试图对一个不是 **Function** 对象调用 **Function prototype** 方法，或者在函数调用上下文中使用了对象。例如，下面的代码将产生该错误，因为 **foo** 不是一个函数。

```
var foo = new Object(); // 创建一个称为 "foo"
                          的新对象。
```

```
var x = foo(); // 试图将 foo 作为一个函数加以调用。
```

要改正此错误

- 只对 **Function** 对象调用 **Function prototype** 方法。
- 确保只使用函数调用操作符() 来调用函数。

请参阅

[Function 对象](#)(See 1.2.8.7) | [prototype 属性](#)(See 1.2.10.34) | [JScript 对象](#)(See 1.2.8)

2.36 非法赋值

试图对一个只读的标识符赋值。不能对只读的标识符赋值。例如，主机定义对象和外部 COM 对象。

要改正此错误

- 不要赋值给只读标识符。

请参阅

[赋值操作符 \(=\)](#)(See 1.2.9.4)

2.37 非法字符

JScript 编译器认为由字符（或多个字符）组成的标识符非法。合法字符应遵守下述规则：

- 第一个字符必须为 ASCII 字母（大写或小写）或下划线（_）。
- 后续字符可以是 ASCII 字母、数字和下划线。

- 标识符名称不能为保留字。

要改正此错误

- 避免使用不属于 JScript 语言定义部分的字符。

请参阅

[JScript 变量](#)(See 1.1.1.3) | [特殊字符](#)(See 1.1.2.6) | [JScript 数据类型](#)(See 1.1.1.4)

2.38 字符集范围非法

试图创建的正则表达式含有无效的字符集范围。字符集的范围必须只能是单个字符的变化，诸如 `a-z` 或 `0-9`；不能在字符集中包括诸如 `\w` 的字符类。字符集范围的第一个字符也必须位于范围中第二个字符的前面。例如：

```
var good = /[a-z]/; // 有效的字符范围 - a 在 z 之前。
```

```
var notGood = /[z-a]/; // 无效的字符范围 - z 不在 a 之前。
```

要改正此错误

- 只使用单字符来组合正则表达式字符集，并确保他们具有正确的顺序。

请参阅

[Regular Expression 对象](#)(See 1.2.8.13) | [Regular Expression 语法](#)(See 1.5.4)

2.39 需要 JScript 对象

试图将一个非-JScript 对象传给需要 JScript 对象的内置函数。各种内置函数都需要在 JScript 中定义的对象（与主机定义的对象和控件之类的外部组件相对）。

要改正此错误

- 请确保正在作为参数传递的对象具有正确的类型。

请参阅

[JScript 对象](#)(See 1.1.1.8) | [使用数组](#)(See 1.1.2.5)

2.40 未找到标签

所引用的标签不存在。标签可用来标记代码块，不过必须在特定范围内，而且必须唯一。

要改正此错误

- 确认没有拼错标签名称。
- 确保所有标签所引用的都是在当前作用域内定义过的标签（包括在作用域之前所做的定义）。

请参阅

[带标签的语句](#)(See 1.2.11.12) | [switch 语句](#)(See 1.2.11.14) | [break 语句](#)(See 1.2.11.4) | [continue 语句](#)(See 1.2.11.6)

2.41 标签定义重复

创建新标签时给定的名称与现有标签名称重复。标签可以用来标记语句块，但在一个指定范围内必须是唯一的。

要改正此错误

- 请确保程序中所使用的所有标签在各自的作用域内都是唯一的。

请参阅

[Labeled 语句](#)(See 1.2.11.12) | [switch 语句](#)(See 1.2.11.14) | [break 语句](#)(See 1.2.11.4) | [continue 语句](#)(See 1.2.11.6)

2.42 需要 Number 类型

试图对一个不是 **Number** 类型的对象调用 **Number.prototype.toString** 或 **Number.prototype.valueOf** 方法。这种调用类型的对象必须是 **Number** 类型。

要改正此错误

只调用 **Number** 的 **Number.prototype.toString** 或 **Number.prototype.valueOf** 方法。

请参阅

[Number 对象](#)(See 1.2.8.10) | [number 属性](#)(See 1.2.10.30)

2.43 需要 Object 类型

试图对一个不是 **Object** 类型的对象调用 **Object.prototype.toString** 或 **Object.prototype.valueOf** 方法。这种调用类型的对象必须是 **Object** 类型。

要改正此错误

只调用 **Object** 的 **Object.prototype.toString** 或 **Object.prototype.valueOf** 方法。

请参阅

[Object 对象](#)(See 1.2.8.11) | [JScript 对象](#)(See 1.1.1.8)

2.44 需要对象的成员

试图对某个对象的数据成员执行某个操作，但该成员不存在。

要改正此错误

- 请确保所输入的数据成员名字是正确的。

请参阅

[Object 对象](#)(See 1.2.8.11) | [JScript 对象](#)(See 1.1.1.8) | [JScript 语言参考](#)(See 2.61)

2.45 需要 Regular Expression 对象

试图对 **RegExp** 类型对象之外的其他对象调用 **RegExp.prototype.toString** 或 **RegExp.prototype.valueOf** 方法。这种调用的类型必须是 **RegExp** 对象。

要改正此错误

- 在 **RegExp** 对象类型 中，只能调用 **RegExp.prototype.toString** 或 **RegExp.prototype.valueOf** 方法

请参阅

[Regular Expression 对象](#)(See 1.2.8.13) | [Regular Expression 语法](#)(See 1.5.4)

2.46 函数外有 'return' 语句

在代码的全部范围内使用。**return** 语句只能出现在函数体内。

使用 **()** 操作符调用函数是一个表达式。所有的表达式都有值；**return** 语句用来指定由函数返回的值。通常的形式是：

```
return [ expression ];
```

当执行到 **return** 语句时，首先对该表达式求值并将其返回作为该函数的值。如果没有表达式，则返回 **undefined**。

当 **return** 语句执行完毕后，该函数的执行就终止了，即使还有函数体中仍有其它语句。这个规则的一个例外是如果该 **return** 语句发生在某个 **try** 块内，且有相应的 **finally** 块，那么 在该函数返回前将执行 **finally** 块中的代码。

如果函数是由于到达函数体结束处而不是因为执行 **return** 语句返回，则返回值为 **undefined**（表示该函数的结果无法作为大表达式的部分。

要改正此错误

- 从代码的主要部分中删除 **return** 语句。（全部范围）。

请参阅

[return 语句](#)(See 1.2.11.13) | [Function 对象](#)(See 1.2.8.7) | [caller 属性](#)(See 1.2.10.3)

2.47 需要 String 对象

试图对 **String** 对象类型之外的其他类型对象调用 **String.prototype.toString** 或 **String.prototype.valueOf** 方法。这种调用的类型必须是 **String** 类型。

要改正此错误

- 在 **String** 类型对象中只能调用 **String.prototype.toString** 或 **String.prototype.valueOf** 方法

请参阅

[String 对象](#)(See 1.2.8.14) | [toString 方法](#)(See 1.2.7.5.15)

2.48 语法错误

所创建的语句违反了一条或多条 JScript 的语法规则。

要改正此错误

- 根据所给定的行号检查程序的语法。
- 查找不匹配的圆括号或中括号。

请参阅

[Error 对象](#)(See 1.2.8.6) | [JScript 语言参考](#)(See 2.61)

2.49 正则表达式语法错误

查找字符串的结构违反了一条或多条关于 JScript 正则表达式的语法规则。

要改正此错误

- 确保正则表达式查找字符串的结构与 JScript 正则字符串的语法一致。

请参阅

[Regular Expression 对象](#)(See 1.2.8.13) | [Regular Expression 对象的属性](#)(See 1.2.8.13) | [Regular Expression 对象的方法](#)(See 1.2.8.13) | [Regular Expression 语法](#)(See 1.5.4) | [compile 方法](#)(See 1.2.7.1.16)

2.50 小数部分的位数越界

试图将一个非法参数传递给 **Number.prototype.toExponential()**。函数 **toExponential()** 的参数为 0 到 20（含 20）之间。

要改正此错误

- 请确保的 **toExponential()** 不是太大也不是太小。

请参阅

[toExponential 方法](#)(See 1.2.7.5.5)

2.51 精度越界

试图将一个非法参数传递给 **Number.prototype.toPrecision**。给 **toPrecision** 的参数必须为 1 到 21（含 21）之间。

要改正此错误

- 请确保 **toPrecision** 不是太大也不是太小。

请参阅

[toPrecision 方法](#)(See 1.2.7.5.14)

2.52 待解码 URI 的编码非法

试图对一个不正确 URI（通用资源标识符）进行解码。URI 具有特殊的语法；绝大多数非字母数字字符在用于 URI 必须加以编码。可以使用 **encodeURIComponent** 和 **encodeURIComponent** 方法从一个标准的 JScript 字符串创建一个 URI。一个完整的 URI 是由一系列部件和分隔符组成的。其通用格式为：

<Scheme>:<first>/<second>;<third>?<fourth>

尖括号内的名称代表部件，而 “:”， “/”， “;” 和 “?” 都是作为分隔符的保留字。

要改正此错误

- 请确保您只需要对合法的 URI 进行解码。你不能对标准的 JScript 字符串解码，因为它们可能含有非法字符。

请参阅

[decodeURI 方法](#)(See 1.2.7.1.20) | [decodeURIComponent 方法](#)(See 1.2.7.1.21)

2.53 待编码的 URI 包含有非法字符

试图将字符串编码得到 URI（通用资源标识符），但是包含非法字符。尽管字符串中的绝大多数字符都是合法的，可以转换为 URI，但有些 Unicode 字符序列是非法的。

要改正此错误

请确保待编码的字符串只包含合法的 Unicode 序列。一个完整的 URI 是由一系列部件和分隔符组成的。其通用格式为：

<Scheme>:<first>/<second>;<third>?<fourth>

尖括号内的名称代表部件，而 “:”， “/”， “;” 和 “?” 都是作为分隔符的保留字。

请参阅

[encodeURIComponent 方法](#)(See 1.2.7.1.23) | [encodeURIComponent 方法](#)(See 1.2.7.1.24)

2.54 在 Throw 的同一行内必须后跟一个表达式

使用了 **throw** 关键字，但在同一行内没有紧接一个表达式。一个 **throw** 语句由两部分组成：**throw** 关键字，后面紧跟待抛出的表达式。例如；

```
if (denominator == 0) {  
  
throw new DivideByZeroException();  
}
```

不能将这两个组件分成两行。

要改正此错误

- 请确保 **throw** 关键字和待抛出的表达式在同一行。

请参阅

[Error 对象](#)(See 1.2.8.6) | [throw 语句](#)(See 1.2.11.16) | [try...catch 语句](#)(See 1.2.11.17)

2.55 未定义标识符

JScript 编译器不认识试图使用的标识符。当出现下述情况时，返回未定义值：

- 不存在的变量
- 变量已声明，但从未对其赋值
- 对象属性不存在。

要改正此错误

- 用 **var** 语句声明一个变量（如 **var x**）。

请参阅

[JScript 变量](#)(See 1.1.1.3) | [变量的范围](#)(See 1.1.2.3) | [JScript 语言参考](#)(See 2.61)

2.56 未预期的限定符

在组织正则表达式的查找模式时，创建了一个带有非法循环因子的模式元素。例如，模式

```
/^+ /
```

是非法的，因为元素 `^`（输入开始）没有循环因子。下表列出了没有循环因子的元素：

元素	描述
<code>^</code>	输入开始
<code>\$</code>	输入结束
<code>\b</code>	词的边界
<code>\B</code>	非单词的边界
<code>*</code>	零个或多个循环
<code>+</code>	零个或多个循环
<code>?</code>	零个或多个循环
<code>{n}</code>	n 循环
<code>{n,}</code>	n 或多个循环
<code>{n,m}</code>	从 n 到 m 循环,包括 m

要改正此错误

- 请确保查找模式元素仅包括合法的循环因子。

请参阅

[Regular Expression 对象](#)(See 1.2.8.13) | [Regular Expression 语法](#)(See 1.5.4)

2.57 注释未结束

多行注释块没有结束。多行注释块以一个 `"/**` 组合开始，并以 `*/` 组合结束。下面是示例：

```
/* 这是注释
```

这是同一注释的另一部分。 `*/`

要改正此错误

- 请确保以 “*/” 结束多行注释。

请参阅

[注释语句](#)(See 1.2.11.5)

2.58 字符串常量未结束

字符串常数后面没有用结束的单引号。字符串常数必须用单引号引起来。

注意 可以使用单引号或双引号作为匹配符，双引号可以包含在单引号所引起来的字符串中，单引号也可以包含在双引号引起来的字符串中。

要改正此错误

- 给字符串的结束处添加单引号。

请参阅

[String 对象](#)(See 1.2.8.14) | [toString 方法](#)(See 1.2.7.5.15)

2.59 需要 VBAArray

所提供的对象不是所需要的 Visual Basic 中的 safeArray。

```
new VBAArray(safeArray);
```

VBAArrays 是只读的，且不能直接被创建。safeArray 参数为一个 VBAArray 值，并且在被传递给 VBAArray 构造函数之前必须获得一个 VBAArray 值。这个值只能通过检索现有的 ActiveX 或其他对象得到。

要改正此错误

- 请确保只将 VBAArray 对象传递给 VBAArray 构造函数。

请参阅

[VBAArray 对象](#)(See 1.2.8.15) | [使用数组](#)(See 1.1.2.5)

2.60 Microsoft Scripting 运行时库特性

分类	特性/关键字
集合	Drives (See 1.4.2.3) Files (See 1.4.2.5) Folders (See 1.4.2.8)
数据存储	Dictionary (See 1.4.2.1)
目录	Add (See 1.4.1.1) Exists (See 1.4.1.14) Items (See 1.4.1.29), Keys (See 1.4.1.30) Remove (See 1.4.1.39), RemoveAll (See 1.4.1.40) Count (See 1.4.3.7) Item (See 1.4.3.20), Key (See 1.4.3.21)
文件系统	Drive (See 1.4.2.2) File (See 1.4.2.4) FileSystemObject (See 1.4.2.6) Folder (See 1.4.2.7) TextStream (See 1.4.2.9)
FileSystemObject	BuildPath (See 1.4.1.3) CopyFile (See 1.4.1.6), CopyFolder (See 1.4.1.7) CreateFolder (See 1.4.1.8), CreateTextFile (See 1.4.1.9) DeleteFile (See 1.4.1.11), DeleteFolder (See 1.4.1.12) DriveExists (See 1.4.1.13), FileExists (See 1.4.1.15), FolderExists (See 1.4.1.16) GetAbsolutePathName (See 1.4.1.17), GetBaseName (See 1.4.1.18) GetDrive (See 1.4.1.19), GetDriveName (See 1.4.1.20) GetFile (See 1.4.1.22), GetExtensionName (See 1.4.1.21) GetFileName (See 1.4.1.23) GetFolder (See 1.4.1.25), GetParentFolderName (See 1.4.1.26) GetSpecialFolder (See 1.4.1.27) GetTempName (See 1.4.1.28) MoveFile (See 1.4.1.32), MoveFolder (See 1.4.1.33) OpenTextFile (See 1.4.1.35) Drives (See 1.4.3.13)
驱动器	AvailableSpace (See 1.4.3.4) Count (See 1.4.3.7) DriveLetter (See 1.4.3.12) DriveType (See 1.4.3.14) FileSystem (See 1.4.3.16) FreeSpace (See 1.4.3.17) IsReady (See 1.4.3.18)

	Item (See 1.4.3.20) RootFolder (See 1.4.3.26) SerialNumber (See 1.4.3.27) ShareName (See 1.4.3.28) TotalSize (See 1.4.3.33) VolumeName (See 1.4.3.35)
文件 文件夹	Add (See 1.4.1.2) Attributes (See 1.4.3.3) Copy (See 1.4.1.5), Delete (See 1.4.1.10), Move (See 1.4.1.31) Count (See 1.4.3.7) OpenAsTextStream (See 1.4.1.34) DateCreated (See 1.4.3.8), DateLastAccessed (See 1.4.3.9), DateLastModified (See 1.4.3.10) Drive (See 1.4.3.11) Item (See 1.4.3.20) ParentFolder (See 1.4.3.24) Name (See 1.4.3.23), Path (See 1.4.3.25) ShortName (See 1.4.3.29), ShortPath (See 1.4.3.30) Size (See 1.4.3.31)
文本流	Close (See 1.4.1.4) Read (See 1.4.1.36), ReadAll (See 1.4.1.37), ReadLine (See 1.4.1.38) Skip (See 1.4.1.41), SkipLine (See 1.4.1.42) Write (See 1.4.1.43), WriteBlankLines (See 1.4.1.44), WriteLine (See 1.4.1.45) AtEndOfLine (See 1.4.3.1), AtEndOfStream (See 1.4.3.2) Column (See 1.4.3.5), Line (See 1.4.3.22)

2.61 JScript 语言参考

[特性信息](#)(See 1.2.1)

[字母顺序的关键字列表](#)(See 1.2.4)

[错误](#)(See 1.2.5)

[函数](#)(See 1.2.6)

[方法](#)(See 1.2.7)

[对象](#)(See 1.2.8)

[运算符](#)(See 1.2.9)

[属性](#)(See 1.2.10)

[语句](#)(See 1.2.11)

2.62 Arguments 对象

该对象代表正在执行的函数和调用它的函数的参数。

`[function.]arguments[n]`

参数

function

可选项。当前正在执行的 **Function** 对象的名字。

n

必选项。要传递给 **Function** 对象的从 0 开始的参数值索引。

说明

不能显式创建 **arguments** 对象。**arguments** 对象只有函数开始时才可用。函数的 **arguments** 对象并不是一个数组，访问单个参数的方式与访问数组元素的方式相同。索引 *n* 实际上是 **arguments** 对象的 **0...n** 属性的其中一个参数。

示例

下面的示例演示了 **arguments** 对象的用法。

```
function ArgTest(a, b) {
    var i, s = "The ArgTest function
expected ";

    var numargs = arguments.length;    //
    获取被传递参数的数值。

    var expargs = ArgTest.length;      //
    获取期望参数的数值。

    if (expargs < 2)
        s += expargs + " argument. ";
    else
        s += expargs + " arguments. ";
}
```

```

if (numargs < 2)
    s += numargs + " was passed.";
else
    s += numargs + " were passed.";
s += "\n\n"
for (i =0 ; i < numargs; i++){          //
获取参数内容。
    s += "  Arg " + i + " = " + arguments[i]
+ "\n";
}

return(s);                               // 返回
参数列表。
}

```

要求

[版本 1](#)(See 2.64)

请参阅

[0...n 属性](#)(See 2.65) | [callee 属性](#)(See 2.66) | [length 属性](#)(See 2.70)

2.63 Script 运行时库参考

[方法](#)(See 1.4.1)

[对象](#)(See 1.4.2)

[属性](#)(See 1.4.3)

2.64 版本信息

下表列出了宿主应用实现的 Microsoft JScript 的版本信息。

语言要素	1.0	2.0	3.0	4.0	5.0	5.1	5.5
Microsoft Internet Explorer 3.0	x						

Microsoft Internet Information Server 1.0	x					
Microsoft Internet Explorer 4.0		x				
Microsoft Internet Information Server 4.0		x				
Microsoft Windows Scripting Host 1.0		x				
Microsoft Visual Studio 6.0			x			
Microsoft Internet Explorer 5.0				x		
Microsoft Internet Information Services 5.0					x	
Microsoft Windows 2000					x	
Microsoft Internet Explorer 5.5						x

下表列出了 JScript 语言特性第一次介绍时的版本信息。

语言特性	1.0	2.0	3.0	4.0	5.0	5.5
0...n 属性						x
\$1...\$9 属性			x			
abs 方法	x					
acos 方法	x					
ActiveXObject 对象			x			
加法运算符 (+)	x					
anchor 方法	x					
apply 方法						x
arguments 属性		x				
Array 对象		x				
asin 方法	x					
赋值运算符 (=)	x					
atan 方法	x					
atan2 方法	x					
atEnd 方法			x			
big 方法	x					
按位与运算符 (&)	x					
按位左移运算符 (<<)	x					
按位取非运算符 (~)	x					
按位或运算符 ()	x					
按位右移运算符 (>>)	x					
按位异或运算符 (^)	x					
blink 方法	x					
bold 方法	x					
Boolean 对象		x				

break 语句	x					
call 方法						x
callee 属性						x
caller 属性		x				
catch 语句					x	
@cc_on 语句			x			
ceil 方法	x					
charAt 方法	x					
charCodeAt 方法						x
逗号运算符 (,)	x					
// (单行注释语句)	x					
/*..*/ (多行注释语句)	x					
比较运算符	x					
compile 方法			x			
concat 方法 (Array)			x			
concat 方法 (String)			x			
条件编译			x			
条件编译变量			x			
条件 (三元) 运算符 (?:)	x					
constructor 属性		x				
continue 语句	x					
cos 方法	x					
数据类型转换			x			
Date 对象	x					
decodeURI 方法						x
decodeURIComponent						x
递减运算符 (--)	x					
delete 运算符			x			
description 属性					x	
dimensions 方法			x			
除法运算符 (/)	x					
do...while 语句			x			
E 属性	x					
encodeURI 方法						x
encodeURI 组件						x
Enumerator 对象			x			
相等运算符 (==)	x					

Error 对象				x	
escape 方法	x				
eval 方法	x				
exec 方法			x		
exp 方法	x				
fixed 方法	x				
floor 方法	x				
fontcolor 方法	x				
fontsize 方法	x				
for 语句	x				
for...in 语句				x	
fromCharCode 方法			x		
Function 对象		x			
function 语句	x				
getDate 方法	x				
getDay 方法	x				
getFullYear 方法			x		
getHours 方法	x				
getItem 方法			x		
getMilliseconds 方法			x		
getMinutes 方法	x				
getMonth 方法	x				
GetObject 函数			x		
getSeconds 方法	x				
getTime 方法	x				
getTimezoneOffset 方法	x				
getUTCDate 方法			x		
getUTCDay 方法			x		
getUTCFullYear 方法			x		
getUTCHours 方法			x		
getUTCMilliseconds 方法			x		
getUTCMinutes 方法			x		
getUTCMonth 方法			x		
getUTCSeconds 方法			x		
getVarDate 方法			x		
getYear 方法	x				
Global 对象			x		

global 属性						x
大于运算符 (>)	x					
大于等于运算符 (>=)	x					
hasOwnProperty 方法						x
严格相等运算符 (===)	x					
@if 语句			x			
if...else 语句	x					
ignoreCase 属性						x
递增运算符 (++)	x					
index 属性			x			
indexOf 方法	x					
不等运算符 (!=)	x					
Infinity 属性			x			
input 属性			x			
instanceof 运算符					x	
isFinite 方法			x			
isNaN 方法	x					
isPrototypeOf 方法						x
italics 方法	x					
item 方法			x			
join 方法		x				
Labeled 语句			x			
lastIndex 属性			x			
lastIndexOf 方法	x					
lastMatch 属性						x
lastParen 属性						x
lbound 方法			x			
leftContext 属性						x
length 属性 (Arguments)						x
length 属性 (Array)		x				
length 属性 (Function)		x				
length 属性 (String)	x					
小于运算符 (<)	x					
小于等于运算符 (<=)	x					
link 方法	x					
LN2 属性	x					
LN10 属性	x					

localeCompare 方法						x
log 方法	x					
LOG2E 属性	x					
LOG10E 属性	x					
逻辑与运算符 (&&)	x					
逻辑非运算符 (!)	x					
逻辑或运算符 ()	x					
match 方法			x			
Math 对象	x					
max 方法	x					
MAX_VALUE 属性		x				
message 属性						x
min 方法	x					
MIN_VALUE 属性		x				
取模运算符 (%)	x					
moveFirst 方法			x			
moveNext 方法			x			
multiline 属性						x
乘法运算符 (*)	x					
name 属性						x
NaN 属性 (Global)			x			
NaN 属性 (Number)		x				
NEGATIVE_INFINITY 属性		x				
new 运算符	x					
非严格相等运算符 (!=)	x					
Number 对象		x				
number 属性					x	
Object 对象			x			
运算符优先级	x					
parse 方法	x					
parseFloat 方法	x					
parseInt 方法	x					
PI 属性	x					
pop 方法						x
POSITIVE_INFINITY 属性		x				
pow 方法	x					
prototype 属性		x				

propertyIsEnumerable 属性						x
push 方法						x
random 方法	x					
RegExp 对象			x			
正则表达式对象			x			
正则表达式语法			x			
replace 方法	x					
return 语句	x					
reverse 方法		x				
rightContext 属性						x
round 方法	x					
ScriptEngine 函数		x				
ScriptEngineBuildVersion 函数		x				
ScriptEngineMajorVersion 函数		x				
ScriptEngineMinorVersion 函数		x				
search 方法			x			
@set 语句			x			
setDate 方法	x					
setFullYear 方法			x			
setHours 方法	x					
setMilliseconds 方法			x			
setMinutes 方法	x					
setMonth 方法	x					
setSeconds 方法	x					
setTime 方法	x					
setUTCDate 方法			x			
setUTCFullYear 方法			x			
setUTCHours 方法			x			
setUTCMilliseconds 方法			x			
setUTCMinutes 方法			x			
setUTCMonth 方法			x			
setUTCSeconds 方法			x			
setYear 方法	x					
shift 方法						x
sin 方法	x					
slice 方法 (Array)			x			

slice 方法 (String)			x			
small 方法	x					
sort 方法		x				
source 属性			x			
splice 方法						x
split 方法			x			
sqrt 方法	x					
SQRT1_2 属性	x					
SQRT2 属性	x					
strike 方法	x					
String 对象	x					
sub 方法	x					
substr 方法			x			
substring 方法	x					
减法运算符 (-)	x					
sup 方法	x					
switch 语句			x			
tan 方法	x					
test 方法			x			
this 语句	x					
throw 语句					x	
toArray 方法			x			
toDateString 方法						x
toExponential 方法						x
toFixed 方法						x
toGMTString 方法	x					
toLocaleDateString 方法						x
toLocaleLowerCase 方法						x
toLocaleString 方法	x					
toLocaleTimeString 方法						x
toLocaleUpperCase 方法						x
toLowerCase 方法	x					
toPrecision 方法						x
toString 方法		x				
toTimeString 方法						x
toUpperCase 方法	x					
toUTCString 方法			x			

try 语句					x	
typeof 运算符	x					
ubound 方法			x			
一元取相反数运算符 (-)	x					
undefined 属性						x
unescape 方法	x					
unshift 方法						x
无符号右移运算符 (>>>)	x					
UTC 方法	x					
valueOf 方法		x				
var 语句	x					
VBAArray 对象			x			
void 运算符		x				
while 语句	x					
with 语句	x					

2.65 0...n 属性

返回一个 **arguments** 对象中的各个参数的实际值，相应的值是由一个正在执行的函数的 **arguments** 属性返回的。

`[function.]arguments[[0|1|2|...|n]]`

参数

function

可选项。当前正在执行的 **Function** 对象的名称。

0, 1, 2, ..., *n*

必选项。0 到 *n* 范围内的非负整数，其中 0 代表第一个参数而 *n* 代表最后一个参数。最后参数 *n* 的值为 **arguments.length-1**。

说明

0...n 属性所返回的值就是传递给正在执行的函数的实际值。尽管实际上并不是一个参数数组，您还是可以按照与访问数组元素的方法相同的方式访问组成 **arguments** 对象的各个参数。

示例

下面的例子演示了 **arguments** 对象的 **0...n** 属性的用法。要完全理解以下例子，请向该函数传递一个或多个参数：

```
function ArgTest() {
    var s = "";
    s += "The individual arguments are: ";
    for (n=0; n< arguments.length; n++) {
        s += ArgTest.arguments[n];
        s += " ";
    }
    return(s);
}

print(ArgTest(1, 2, "hello", new
Date()));
```

要求

[版本 5.5](#)(See 2.64)

请参阅

应用于：[arguments 对象](#)(See 2.62) | [Function 对象](#)(See 1.2.8.7)

2.66 callee 属性

返回正被执行的 **Function** 对象，也就是所指定的 **Function** 对象的正文。

[function.]arguments.callee

可选项 *function* 参数是当前正在执行的 **Function** 对象的名称。

说明

callee 属性是 **arguments** 对象的一个成员，仅当相关函数正在执行时才可用。
callee 属性的初始值就是正被执行的 **Function** 对象。这允许匿名的递归函数。

示例

```
function factorial(n) {
    if (n <= 0)
        return 1;
    else
        return n * arguments.callee(n - 1)
}
print(factorial(3));
```

要求

[版本 5.5](#)(See 2.64)

请参阅

应用于: [arguments 对象](#)(See 2.62) | [Function 对象](#)(See 1.2.8.7)

2.67 lastMatch 属性 (\$&)

返回任何正则表达式搜索过程中的最后匹配的字符。只读。

RegExp.lastMatch

与此属性相关联的对象总是全局 **RegExp** 对象。

说明

lastMatch 属性的初始值是一个空字符串。只要找到了匹配的字符，**lastMatch** 属性的值就会相应改变。

示例

下面的例子演示了 **lastMatch** 属性的用法:

```
function matchDemo() {
    var s; // 声明
    // 变量。
    var re = new RegExp("d(b+) (d)", "ig");
    // 正则表达式样式。
    var str = "cdbBdbsbdbdz";
```



```

// 要查找的字符串。

var      arr      =      re.exec(str);

// 进行查找。

s = "$1 returns: " + RegExp.$1 + "\n";
s += "$2 returns: " + RegExp.$2 + "\n";
s += "$3 returns: " + RegExp.$3 + "\n";
s += "input returns : " + RegExp.input
+ "\n";

s += "lastMatch returns: " +
RegExp.lastMatch + "\n";

s += "leftContext returns: " +
RegExp.leftContext + "\n";

s += "rightContext returns: " +
RegExp.rightContext + "\n";

s += "lastParen returns: " +
RegExp.lastParen + "\n";

return(s); // 返
回结果。
}

```

```
document.write(matchDemo());
```

要求

[版本 5.5](#)(See 2.64)

请参阅

[\\$1...\\$9 属性](#)(See 1.2.10.1) | [index 属性](#)(See 1.2.10.10) | [input 属性](#)(See 1.2.10.12) |

[lastIndex 属性](#)(See 1.2.10.14) | [lastParen 属性](#)(See 2.68) | [leftContext 属性](#)(See 2.69) | [rightContext 属性](#)(See 2.71)
 应用于: [RegExp 对象](#)(See 1.2.8.12)

2.68 lastParen 属性 (\$+)

如果有的话, 返回任何正则表达式查找过程中最后括的子匹配。只读。

RegExp.lastParen

与此属性相关的对象总是全局 **RegExp** 对象。

说明

属性的初始值 **lastParen** 属性是空字符串。只要成功找到匹配字符, **lastParen** 属性的值就会相应改变。

示例

下面的例子演示了 **lastParen** 属性的用法:

```
function matchDemo() {

    var s;                                     // 声明
    变量。

    var re = new RegExp("d(b+) (d)", "ig");

    // 正则表达式样式。

    var str = "cdbBdbdbdbdz";

    // 将被查找的字符串。

    var arr = re.exec(str);

    // 进行查找。

    s = "$1 returns: " + RegExp.$1 + "\n";
    s += "$2 returns: " + RegExp.$2 + "\n";
    s += "$3 returns: " + RegExp.$3 + "\n";
    s += "input returns : " + RegExp.input
    + "\n";
```

```

    s += "lastMatch returns: " +
RegExp.lastMatch + "\n";

    s += "leftContext returns: " +
RegExp.leftContext + "\n";

    s += "rightContext returns: " +
RegExp.rightContext + "\n";

    s += "lastParen returns: " +
RegExp.lastParen + "\n";

    return (s); // 返
回结果。
}

```

```
document.write(matchDemo());
```

要求

[版本 5.5](#)(See 2.64)

请参阅

[\\$1...\\$9 属性](#)(See 1.2.10.1) | [index 属性](#)(See 1.2.10.10) | [input 属性](#)(See 1.2.10.12) | [lastIndex 属性](#)(See 1.2.10.14) | [lastMatch 属性](#)(See 2.67) | [leftContext 属性](#)(See 2.69) | [rightContext 属性](#)(See 2.71)
 应用于: [RegExp 对象](#)(See 1.2.8.12)

2.69 leftContext 属性 (\$`)

返回被查找的字符串中从字符串开始位置到最后匹配之前的位置之间的字符。只读。

RegExp.leftContext

与此属性相关联的对象总是全局 **RegExp** 对象。

说明

leftContext 属性的初始值是空字符串。只要找到了匹配的字符, **leftContext** 属性的值就会相应改变。

示例

以下示例阐述了 **leftContext** 属性的用法：

```
function matchDemo() {

    var s;                                     // 声明
    变量。

    var re = new RegExp("d(b+) (d)", "ig");

    // 正则表达式样式。

    var      str      =      "cdbBdbbsbdbdz";

    // 将被查找的字符串。

    var      arr      =      re.exec(str);

    // 进行查找。

    s = "$1 returns: " + RegExp.$1 + "\n";
    s += "$2 returns: " + RegExp.$2 + "\n";
    s += "$3 returns: " + RegExp.$3 + "\n";
    s += "input returns : " + RegExp.input
+ "\n";

    s  +=  "lastMatch  returns:  "  +
RegExp.lastMatch + "\n";

    s  +=  "leftContext  returns:  "  +
RegExp.leftContext + "\n";

    s  +=  "rightContext  returns:  "  +
RegExp.rightContext + "\n";

    s  +=  "lastParen  returns:  "  +
```

```

RegExp.lastParen + "\n";

    return (s);                                     // 返回结果。
}

```

```
document.write(matchDemo());
```

要求

[版本 5.5](#)(See 2.64)

请参阅

[\\$1...\\$9 属性](#)(See 1.2.10.1) | [index 属性](#)(See 1.2.10.10) | [input 属性](#)(See 1.2.10.12) | [lastIndex 属性](#)(See 1.2.10.14) | [lastMatch 属性](#)(See 2.67) | [lastParen 属性](#)(See 2.68) | [rightContext 属性](#)(See 2.71)

应用于: [RegExp 对象](#)(See 1.2.8.12)

2.70 length 属性 (arguments)

返回调用程序传递给函数的实际参数数目。

[*function*.]**arguments.length**

可选项 *function* 参数是当前正在执行的 **Function** 对象的名称。

说明

当 **Function** 对象开始执行时, 脚本引擎将 **arguments** 对象的 **length** 属性初始化为传递给该函数的实际参数数目。

示例

下面的例子演示了 **arguments** 对象的 **length** 属性的用法。要完全理解示例, 请向该函数传递更多的参数:

```

function ArgTest(a, b) {

    var i, s = "The ArgTest function
expected ";

    var numargs = arguments.length;
    var expargs = ArgTest.length;

    if (expargs < 2)

```

```

        s += expargs + " argument. ";
    else
        s += expargs + " arguments. ";
    if (numargs < 2)
        s += numargs + " was passed.";
    else
        s += numargs + " were passed.";
    return (s);
}

```

要求

[版本 5.5](#)(See 2.64)

请参阅

[arguments 属性](#)(See 1.2.10.2) | [length 属性 \(Array\)](#)(See 1.2.10.15) | [length 属性 \(String\)](#)(See 1.2.10.17)

应用于: [arguments 对象](#)(See 2.62) | [Function 对象](#)(See 1.2.8.7)

2.71 rightContext 属性 (\$')

返回被搜索的字符串中从最后一个匹配位置开始到字符串结尾之间的字符。只读。

RegExp.rightContext

与此属性相关联的对象总是全局的 **RegExp** 对象。

说明

rightContext 属性的初始值是空字符串。只要成功找到了匹配的字符，**rightContext** 属性的值就会相应改变。

示例

以下示例阐述了 **rightContext** 属性的用法：

```

function matchDemo() {
    var s;                                     // 声明

```

变量。

```

    var re = new RegExp("d(b+) (d)", "ig");

    // 正则表达式样式。

    var      str      =      "cdbBdbbsbdbdz";

    // 将被查找的字符串。

    var      arr      =      re.exec(str);

    // 进行查找。

    s = "$1 returns: " + RegExp.$1 + "\n";
    s += "$2 returns: " + RegExp.$2 + "\n";
    s += "$3 returns: " + RegExp.$3 + "\n";
    s += "input returns : " + RegExp.input
+ "\n";

    s  +=  "lastMatch  returns:  "  +
RegExp.lastMatch + "\n";

    s  +=  "leftContext  returns:  "  +
RegExp.leftContext + "\n";

    s  +=  "rightContext  returns:  "  +
RegExp.rightContext + "\n";

    s  +=  "lastParen  returns:  "  +
RegExp.lastParen + "\n";

    return(s);                                     // 返
回结果。
}

document.write(matchDemo());

```

要求

[版本 5.5](#)(See 2.64)

请参阅

[\\$1...\\$9 属性](#)(See 1.2.10.1) | [index 属性](#)(See 1.2.10.10) | [input 属性](#)(See 1.2.10.12) | [lastIndex 属性](#)(See 1.2.10.14) | [lastMatch 属性](#)(See 2.67) | [lastParen 属性](#)(See 2.68) | [leftContext 属性](#)(See 2.69)

应用于: [RegExp 对象](#)(See 1.2.8.12)