# 复习

js中只有函数能够关住变量和函数的作用域。

局部变量，函数内部声明的变量，只能在函数内部使用。

函数的形式参数都是局部变量，只能在函数内部使用

全局变量作用：信号量，不同函数同时管理这个变量

函数天生存在闭包。函数能够记住定义时外部环境和内部语句

|  |
| --- |
| 1. function fun1() { 2. var a = 1; 3. function fun2() { 4. console.log(a); 5. } 6. // 返回fun2定义 7. return fun2; 8. } 9. // i = function fun2() {console.log(a)}; 10. var i = fun1(); 11. console.log(i); 12. i(); |

数组：有序数据集合 []

pop（）删除数组最后一项

push()添加在数组末尾

unshift(): 开头添加

shift(): 删除数组第一项

concat() 合并

slice(start,end): 截取不包含end

splice(index,howmany,elements):

删除：index，索引值 howmany不能是0 elements省略

插入：index,索引值 howmany是0， elements表示要插入数据

替换：index索引值 howmany 不能是0 elements表示替换的数据

sort():

省略参数：按照字符编码进行排序

参数：匿名函数（比较函数）

# 数组方法

## join()

将数组转为字符串方法

参数：省略（数据默认以逗号连接）

返回值：字符串

参数：连接字符

|  |
| --- |
| 1. // 参数：连接符 2. var str = arr.join("\*\_\*"); 3. console.log(str); |



# 字符串

## length属性

|  |
| --- |
| 字符串中包含字符总个数（字母，数字，汉字，标点，空格等）   1. str.length |

## charAt()

得到指定位置（索引值）的字符

参数：索引值

返回值：字符

|  |
| --- |
| 1. for (var i = 0 ; i < str.length; i ++) { 2. console.log(str.charAt(i)); 3. } |

## indexOf()

得到指定字符首次出现的位置。

参数：字符

返回值：索引值

|  |
| --- |
| 1. // 没有该字符，返回值-1 2. console.log(str.indexOf("阳光")); |

## concat()

用于字符串合并

参数：合并的字符串，字面量，变量等

返回值：合并后的新字符串

|  |
| --- |
| 1. var newStr = str.concat("希望天气暖和"); |

## slice()

字符串的截取方法

参数：start,end 从start开始截取到end，不包括end

返回值：截取都的新字符串

|  |
| --- |
| 1. // 参数都是正值 2. var str2 = "今天有微风，气温10°C"; 3. console.log(str2.slice(3,6)); 4. // 参数：负数 -1开始 5. console.log(str2.slice(-6,-4)); 6. // 参数：省略end表示截取到最后 7. console.log(str2.slice(2)); |

## substr()

用于字符串截取

参数：第一个参数index 第二个参数howmany 截取字符串个数

返回值：截取到的新字符串

|  |
| --- |
| 1. var str = "今天有微风，气温10°C有。"; 2. // substr(index,howmany) 3. console.log(str.substr(2,4)); 4. // index也可以使用负数 表示倒数 5. console.log(str.substr(-8,6)); 6. // 省略howmany表示截取到最后 7. console.log(str.substr(-8)); |

## substring()

用于字符串截取

参数：start,end （start，end不区分大小） 表示从小索引值截取到大索引值，不包括大索引值

不能使用负数

返回值：截取到新的字符串

|  |
| --- |
| 1. // substring() 从小索引值截取到大索引值，不包括大索引值 2. console.log(str.substring(2,5)); 3. console.log(str.substring(5,2)); |

## split()

将字符串转为数组方法

参数：切割字符

返回值：数组

|  |
| --- |
| 1. // split()字符串转为数组方法 2. var str2 = "天气 晴朗 微风 阴天 。" 3. var arr = str2.split(" "); |



## toUpperCase()

转为大写字母

参数：省略

返回值：新字符串

## toLowerCase()

转为小写字母

参数：省略

返回值：新字符串

案例：

将“everything is good in its season”，转为每个单词的首字母大写。

即“Everything Is Good In Its Season”

|  |
| --- |
| 1. /\* 2. 将“everything is good in its season”，转为每个单词的首字母大写。 3. 即“Everything Is Good In Its Season” 4. 1 “everything is good in its season” 的到每一个单词（数组中每一项） 5. 2 数组中每一项重新赋值 = 第一个字母大写 + 每一个单词除了第一个字母后面所有的字符 6. 3 将数组转为字符串 7. \*/ 8. var str = "everything is good in its season"; 9. // 1的到每一个单词 字符串转为数组方法 10. var arr = str.split(" "); 11. // 2数组每一项重新赋值 12. for (var i = 0; i <= arr.length - 1; i ++) { 13. // console.log(arr[i]); 14. arr[i] = arr[i].charAt(0).toUpperCase() + arr[i].slice(1); 15. } 16. // 3将数组转为字符串 17. str = arr.join(" "); 18. console.log(str); |

# 正则**表达式**

## 3.1 概述

正则表达式（regular expression）。简写RegExp。

创建正则：使用正则字面量。/表达式/

正则的数据类型是引用数据类型object。

正则表达式：是被用来匹配字符串中的字符组合的模式。是一种最简单的验证字符串方法。常用来做表单验证

## 3.2 字符串的方法

配合正则表达式使用的方法有字符串方法和正则表达式的方法。

### 3.2.1 split()

split()方法根据匹配字符串切割父字符串，返回切割后的数组。

|  |
| --- |
| 1. "ajbbbbbdsbbfkbjfbbbbbbbbbbbbbbbbbbbbk".split(/b+/) |



### 3.2.2 match()

使用正则表达式与字符串相比较，返回一个包含匹配结果的数组。

参数：可以是字符串

|  |
| --- |
| 1. "axyzbccc".match("xyz") |



match匹配时，不书写g只会输出匹配到的第一个字符串。

参数：还可以是正则表达式

match还可以进行全局匹配；返回所有匹配到的字符串组成的数组。

|  |
| --- |
| 1. "abccbbbbczbbbbbbbz".match(/b+/g) |



### 3.2.3 search

search() 对正则表达式或指定字符串进行搜索，返回第一个出现的匹配项的索引值。

参数：字符串

|  |
| --- |
| 1. // search:搜索 查找首次出现字符串所在的索引值 2. // 参数：字符串 3. console.log("abccbbbbczbbbbbbbz".search("b")); 4. // 参数：正则 search没有全局搜索 只能将首次位置返回 5. console.log("abccbbbbczbbbbbbbz".search(/b+/)); 6. // 没有要搜索字符串 返回值-1 7. console.log("abccbbbbczbbbbbbbz".search(/xyz/)); |

### 3.2.4 replace

替换：用新字符串替换匹配字符串。

参数：第一个参数匹配字符串（字符串，正则） 第二个参数：要替换的字符串

返回值：替换后新字符串

|  |
| --- |
| 1. // replace(a,b)替换 用b替换a a形式可以是字符串 正则 2. console.log("www.baidu.com".replace("baidu","icketang")); 3. console.log("www.baidu.com".replace(/baidu/,"icketang")); |



## 3.3 正则方法

### 3.3.1 exec()

查找，将匹配的字符串用数组方式返回。

exec()没有全局匹配。

参数：父字符串

返回值：数组

|  |
| --- |
| 1. console.log(/\d{3}/.exec("010-12abc789")); |

|  |
| --- |
| 1. // 没有匹配结果 2. console.log(/a{3}/.exec("010-12abc789")); |



### 3.3.2 test()

test(): 检测字符串中是否含有正则匹配字符串

参数：检测父字符串

返回值：布尔值 含有true 不含有false

|  |
| --- |
| 1. console.log(/a{2}/.test("abchdsjdaa")); 2. console.log(/a{3}/.test("abchdsjdaa")); |

# 四、正则表达式的术语和操作符

正则表达式的组成：由普通字符和特殊字符（元字符）组成。

普通字符包括：字母，或者是数字等

特殊字符：() {} [] \ | ^ $ ? + .

正则中匹配特殊字符必须使用转义字符\

## 4.1 精确匹配

精确匹配：正则表达式只含有普通字符

/abc/: 匹配的字符串：只含有a,b,c三个字符 顺序只能是abc

|  |
| --- |
| 1. "abcjacb".match(/abc/g) |

## 4.2 预定义特殊字符

\n:换行

\t：制表符

|  |
| --- |
| 1. console.log("ah 2. c".match(/\n/g)); |



|  |
| --- |
| 1. console.log(`ah 2. c`.match(/\n/g)); |



## 4.3 字符集

[]。把所有匹配的可能性都书写在中括号内，用于匹配一个字符。

简单类：直接书写所有的可能性。

范围类：表示一类可能性。[0-9],[a-z],[A-Z]

组合类：表示不同类型的范围组合。[0-9a-z],[a-zA-Z];

负向类：表示不包含后面书写匹配字符的可能性。书写在[]左中括号的后面

## 4.4 修饰符

g：表示全局匹配。表示在匹配到一个满足条件的字符串并没有停止而是将所有的匹配进行查找。

语法：//g。

i:表示不区分大小写（对大小写不敏感）。

语法：//i。

修饰符可以同时使用。

## 4.5 边界

^：开头，千万不能书写在中括号内。表示以^之后的字符串做为开头的匹配。

$:结尾。表示必须以$前面的字符作为字符串的结尾。

|  |
| --- |
| 1. var str = "abcicjabcegkgagc"; 2. console.log(str.match(/agc$/g)); |

\b:单词边界，用于查找位于单词的开头或结尾的匹配。

\B： 非单词边界，用于查找不处在单词的开头或结尾的匹配。

## 4.6 量词

量词：一般用于处理一排紧密相连的同类的字符。

语法：{}

{n} 硬性量词。表示出现n次

{n,m} 软性量词。表示至少出现n次不能超过m次。

{n**,**} 软性量词。表示至少出现n次。

？ 。表示出现0或者1次。

+ 。表示出现1次或者多次。{1,}

\* 。表示出现0次或者多次。{0,}

## 4.7 分组

表示重复出现多个连续字符。

语法：（）

## 4.8预定义类

js提前给我们定义好的，一些特殊字符。表示一类字符，是一些特殊字符集的简写。

. [^\n\r] 表示除了换行和回车之外的任意字符。

\d [0-9] 表示数字字符。

\D [^0-9] 表示非数字字符。

\s [ \t\n\x0B\f\r] 表示空白字符。

\S [^ \t\n\x0B\f\r] 表示非空白字符

\w [a-zA-Z\_0-9] 表示单词字符(所有的字母/数字/下划线)。

\W [^a-zA-Z\_0-9] 表示非单词字符

## 4.9 或匹配

字符1 | 字符2 可以是字符1或者是字符2

|  |
| --- |
| 1. "abcbc".match(/(ab)|(bc)/g) |

## 4.10 分组反向引用

分组反向引用：正则表达式匹配的字符串组再次使用。

每一个分组对应一个唯一的编码，按照自然数编码从1开始，1,2,3,4

1在正则表达式里面再次使用 \编码，再次使用哪一个匹配组书写对应的编码

|  |
| --- |
| 1. console.log("abcxyzxyzxyz1212abcabc12xyzabc".match(/(\d{2})\1(abc)\2\1/g)); |



2还可以在正则表达式外面使用 $编码

|  |
| --- |
| 1. console.log("abc12".replace(/([a-z]{3})(12)/,"$2")); |

replace()第二个参数还可以书写为match函数，

形式参数第一个参数必须是match，其他参数依次书写编码，每一个小括号的编码必须书写

但是形参可以不使用

|  |
| --- |
| 1. var str2 = "456\*678"; 2. str2 = str2.replace(/(\d{3})\\*(\d{3})/,function (match,$1,$2) { 3. return $2 + "+" + $1; 4. }); 5. console.log(str2); |

## 4.11中文

匹配中文：[\u4e00-\u9fa5]

是一个固定用法，中文只能在正则表达式里这样表示。

|  |
| --- |
| 1. console.log("中文\*匹配".match(/[\u4e00-\u9fa5]+/g)); |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |