网易微专业之《前端开发工程师》

学习笔记

开始时间：2015.12.28

《JavaScript 程序设计》

基础篇（三）

十一、JS 函数

函数是完成某个特定功能的一组语句。函数定义好后，是不能自动执行的，所以需调用它，只需直接在需要的位置写函数就 ok 了。

如:我们要完成多组数和的功能。

```
var sum;
sum = 3+2;
alert(sum);
sum=7+8 ;
alert(sum);
....  //不停重复两行代码
```

如果要实现 8 组数的和，就需要 16 行代码，实现的越多，代码行也就越多。所以我们可以把完成特定功能的代码块放到一个函数里，直接调用这个函数，就省去重复输入大量代码的麻烦。

使用函数完成:

```
function add2(a,b){
sum = a + b;
 alert(sum);
} //  只需写一次就可以
add2(3,2);
add2(7,8);
....  //只需调用函数就可以
```

函数，就是一个一系列 JavaScript 语句的集合，这是为了完成某一个会重复使用的特定功能。在需要该功能的时候，直接调用函数即可，而不必每次都编写一大堆重复的代码。并且在需要修改该功能的时候，也只要修改和维护这一个函数即可。

总之，将语句集合成函数，好处就是方便代码重用。并且，一个好的函数名，可以让人一眼就知道这个函数实现的是什么功能，方便维护。

函数的使用只需要 2 步：

（1）定义函数；

（2）调用函数；

函数语法：

```
function 函数名([形参列表])
{
    执行代码
}
函数名([实参列表]);    //调用函数
```

说明：

1.形参列表，**是可选项**，可以是多个参数，各个参数之间用”,”号分开。

2.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，函数中包含的代码不被执行。

3.形参，指定义的函数变量或常量；实参，指实际要传入函数中的变量或常量。

示例：

```
Function add(number0,number1){
          var sum =number0+number1;
          return sum;
}
var x = add(2,3); //函数调用
```

定义函数

```
function add(number0, number1) {
    var sum = number0 + number1;
    return sum;
}
```

1. 函数声明

```
var add = function(number0, number1) {
    var sum = number0 + number1;
    return sum;
}
```

2. 函数表达式

函数调用

下面这张图片就是一个函数调用的过程。

```
function add(number0, number1) {
    var sum = number0 + number1;
    return sum;
}

var x = add(2, 3);
```

number0 = 2, number1 = 3

sum = 5

x = 5

函数参数

1.实参数量少于形参数量：

```
function add(number0, number1) {
    var sum = number0 + number1;
    return sum;
}

var x = add(2);
```

number0 = 2, number1 = undefined

上面代码 x 变量的返回值为 NaN。

2.实参数量多于形参数量：

```
function add(number0, number1) {
    var sum = number0 + number1;
    return sum;
}

var x = add(2, 3, 4);
```

number0 = 2, number1 = 3

实际上在调用函数的时候，会有一个隐藏变量 arguments 属性，**无需明确指出参数名**，就能访问它们。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。
上面的图片类似于下面：

```
function add(number0, number1) {
    var sum = number0 + number1;
    return sum;
}

var x = add(2, 3, 4);
```

arguments
0: 2
1: 3
2: 4
length: 3

其代码 x 变量的返回值为 5。

示例：参数数量不定的求和

```
function add(){
        var length = arguments.length,sum=0,parameter;
        for (var i=0;i<length;i++){
        parameter = arguments[i];
        sum = sum + parameter;
```

```
        }
}
add(2,3);          //5
add(2,3,4);        //9
add(2,3,4,5);      //14
```

3.参数为原始类型：值传递。参数值不变

```
function increment(number) {
    number = number + 1;
    return number;
}

var a = 1;
var x = increment(a);    // 2
a;  // 1
```

4.参数为对象类型：引用传递。参数值会改变。

```
function increment(person) {
    person.age = person.age + 1;
    return person;
}

var jerry = {name: 'jerry', age: 1};
var x = increment(jerry); // {name: 'jerry', age: 2}
jerry;  // {name: 'jerry', age: 2}
```

函数作用域

函数创建了一个作用域，用来限制变量起作用的范围。

JavaScript 通过函数管理作用域。在函数内部声明的变量只在这个函数内部可用，而在函数外面不可用。另一方面，全局变量就是在任何函数外面声明的或是未声明直接简单使用的。您可以在不同的函数中使用名称相同的局部变量，因为只有声明过该变量的函数才能识别出该变量。

示例：

```
var zhoujielun = {
    name: "周杰伦",
    gender: 1
};
function class1() {
    var zhoujielun = {
        name: "周杰伦",
        gender: 0
    };
    zhoujielun.name = "周杰";
    zhoujielun.gender = 1;
}
class1();
zhoujielun;   // {name: "周杰伦", gender: 1}
```

```
var zhoujielun = {
    name: "周杰伦",
    gender: 1
};
function class1() {
    zhoujielun.name = "周杰";
    zhoujielun.gender = 1;
}
class1();
zhoujielun;   // {name: "周杰", gender: 0}
```

函数作为对象属性使用

```
var point = {
    x: 1,
    y: 1,
    move: function(stepX, stepY) {
        this.x += stepX;
        this.y += stepY;
    }
};

point.move(2, 1);
```

构造函数

JavaScript 中，当任意一个普通函数用于创建一类对象时，它就被称作构造函数，或构造器。一个函数要作为一个真正意义上的构造函数，需要满足下列条件：

1、 在函数内部对新对象（this）的属性进行设置，通常是添加属性和方法。

2、 构造函数可以包含返回语句（不推荐），但返回值必须是 this，或者其它非对象类型的值。

在 JavaScript 中，任何合法的函数都可以作为对象的构造函数，这既包括系统内置函数，也包括用户自己定义的函数。一旦函数被作为构造函数执行，它内部的 this 属性将引用函数本身。

通常来说，构造函数没有返回值，它们只是初始化由 this 指针传递进来的对象，并且什么也不返回。如果一个函数有返回值，被返回的对象就成了 new 表达式的值。从形式上看，一个函数被作为构造函数还是普通函数执行的唯一区别，是否用 new 运算符。

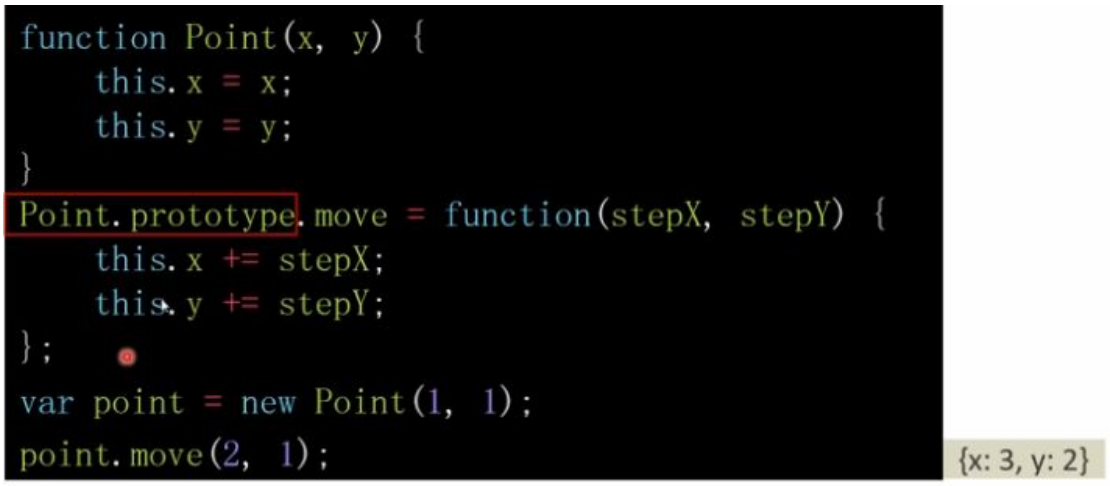
示例：

```
function Point(x, y) {
    this.x = x;
    this.y = y;
    this.move = function(stepX, stepY) {
        this.x += stepX;
        this.y += stepY;
    }
}
var point = new Point(1, 1);
var point2 = new Point(2, 2);
var point3 = new Point(3, 3);
// ...
```

```
{x: 1, y: 1, move: function(stepX, stepY){}}
{x: 2, y: 2, move: function(stepX, stepY){}}
{x: 2, y: 2, move: function(stepX, stepY){}}
```

函数原型

Js 所有的函数都有一个 prototype 属性，这个属性引用了一个对象，即原型对象，也简称原型。



资源：

1. 深入理解构造函数：http://www.2cto.com/kf/201402/281841.html
2. http://www.jb51.net/article/47871.htm

3. 深入理解 javascript 构造函数和原型对象：http://www.jb51.net/article/55539.htm

4. JS 构造函数：

http://www.tzwhx.com/NewShow/newBodyShow/%BB%F9%B4%A1%D6%AA%CA%B6_25420.html

十二、Date

创建日期

在 JavaScript 中，创建日期对象必须使用“new 语句”。使用关键字 new 新建日期对象时，常用的有 2 种：

方法一：

```
var 日期对象名 = new Date();
```

方法二：

```
var 日期对象名 = new Date(日期字符串);
```

Date 对象方法

日期对象 Date 的方法主要分为三大组：setXxx、getXxx 和 toXxx。

setXxx 用于设置时间和日期值；getXxx 用于获取时间和日期值；toXxxx 主要是将日期转换为指定格式。

表 1 用于获日期时间的 getXxx

方法	说明
getFullYear()	返回一个表示年份的 4 位数字
getYear()	返回年份
getMonth()	返回值是 0（一月）到 11（十二月）之间的一个整数
getDate()	返回值是 1~31 之间的一个整数
getDay()	返回星期，返回的是 0-6 的数字，0 表示星期天。如果要返回相对应“星期”，通过数组完成
getHours()	返回值是 0~23 之间的一个整数，来表示小时数
getMinutes()	返回值是 0~59 之间的一个整数，来表示分钟数
getSeconds()	返回值是 0~59 之间的一个整数，来表示秒数

表 2 用于设置日期时间的 setXxx

方法	说明
setFullYear()	可以设置年、月、日
setMonth()	可以设置月、日
setDate()	可以设置日数
setDay()	可以设置星期
setHours()	可以设置时、分、秒、毫秒
setMinutes()	可以设置分、秒、毫秒
setSeconds()	可以设置秒、毫秒
setTime()	设置时间，单位毫秒数，计算从 1970 年 1 月 1 日零时到日期对象所指的日期的毫秒数。

表 3 将日期时间转换为字符串的 toXxx

方法	说明
toString()	将日期时间转换为普通字符串
toUTCString()	将日期时间转换为世界时间（UTC）格式的字符串
toLocaleString()	将日期时间转换为本地时间格式的字符串

案例：

```
var mydate=new Date();//假设当前时间 2014 年 3 月 6 日
document.write(mydate+"<br>");//输出当前时间
document.write(mydate.getFullYear()+"<br>");//输出当前年份
mydate.setFullYear(81); //设置年份
document.write(mydate+"<br>"); //输出年份被设定为 0081 年。
```

注意:不同浏览器， mydate.setFullYear(81)结果不同，年份被设定为 0081 或 81 两种情况。

结果:

```
Thu Mar 06 2014 10:57:47 GMT+0800
2014
Thu Mar 06 0081 10:57:47 GMT+0800
```

注意:

1.结果格式依次为：星期、月、日、年、时、分、秒、时区。这是默认格式，如需要转换成我们的本地时间格式，需要用到方法 **date.toLocaleString()**来设置。

2. 不同浏览器，时间格式有差异。

返回/设置时间方法

---

get/setTime() 返回/设置时间，单位毫秒数，计算从 1970 年 1 月 1 日零时到日期对象所指的日期的毫秒数。

如果将目前日期对象的时间推迟 1 小时，代码如下:

```
<script type="text/javascript">
  var mydate=new Date();
  document.write("当前时间: "+mydate+"<br>");
  mydate.setTime(mydate.getTime() + 60 * 60 * 1000);
  document.write("推迟一小时时间: " + mydate);</script>
```

结果:

当前时间：Thu Mar 6 11:46:27 UTC+0800 2014

推迟一小时时间：Thu Mar 6 12:46:27 UTC+0800 2014

注意:1. 一小时 60 分，一分 60 秒，一秒 1000 毫秒

2. 时间推迟 1 小时,就是: “x.setTime(x.getTime() + 60 * 60 * 1000);”

Date 格式化

```
<script>
function padding(number){
    return number < 10 ? '0' + number : '' + number;
}
function format(date) {
    return date.getFullYear() + '-' + padding(date.getMonth() + 1)
        + '-' + padding(date.getDate()) + ' '
        + padding(date.getHours()) + ':' + padding(date.getMinutes
            ()) + ':' + padding(date.getSeconds());
}
var date = new Date();
alert(date);
alert(format(date));
</script>
```

date.setXXX() ---设置日期

示例：

var date=new Date(2015,7,20,14,57,18) //自定义一个时间：2015-08-20 14:57:18；

date.setDate(35); // 结果会自动变成 2015-09-04 14:57:18

求天数

示例：

new Date(2001,2,0) → 2001-02-28 00:00:00

new Date(2001,3,0) → 2001-03-31 00:00:00

代码如下：

```
<script>
    function getDays(year, month){
        var date = new Date(year, month, 0);
        return date.getDate();
    }

    alert('2001年2月有' + getDays(2001, 2) + '天。');
    alert('2001年3月有' + getDays(2001, 3) + '天。');
</script>
```

Date ⟷ Number 相互转换

示例：

var date=new Date(2015,7,20,14,57,18) → 2015-08-20 14:57:18

date.getTime(): // 1440053838000 → 距 1970-1-1 00:00:00 的毫秒数

var date=new Date(1440053838000); → 2015-08-20 14:57:18

date.setTime(1440053838000); → 2015-08-20 14:57:18

十三、RegExp

正则表达式简介---描述字符串规则的表达式

正则表达式，全称“Regular Expression”，在代码中常简写为 regex、regexp 或 RE。

正则表达式，就是用某种模式去匹配一类字符串的公式。

学习正则表达式就是学习怎样定义一种“模式”的语法，说白了，就是学习各种匹配的规则，例如匹配数字要怎么要怎么写，匹配字符怎么写等等。

正则表达式的定义方法

在 JavaScript 中，正则表达式是由一个 RegExp 对象表示的，利用 RegExp 对象来完成有关正则表达式的操作和功能。

正则表达式的定义共有 2 种方式：

（1）显式定义；

显式定义必须是使用 new 关键词来定义。

语法：`var 变量名 = new RegExp(pattern,attrs);`

（2）隐式定义；

语法：`var 变量名 = /pattern/attrs`

- **pattern：**正则表达式规则。
- **attrs:** 正则表达式属性， 如 g 代表全局模式，i 表示不区分大小写。

举例：

```
var myregex = new ReExp("[0-9]");
var myregex = /[0-9]/;
```

regexObj.test(str)方法

测试正则表达式与指定字符串是否匹配。

该方法返回一个 boolean 值。也就是说，test()方法检查字符串 str 是否符合正则表达式模式 regexp，如果符合，则返回 true；如果不符合，则返回 false。

示例：

/13566668888/.test('1356666888'); // false

/13566668888/.test('13566668888'); // true

/13566668888/.test('x1356666888y'); // true

锚点---定位符

匹配一个位置

^	限定开始位置的字符
$	限定结尾位置的字符
\b	限定单词（字）边界的字符
\B	限定非单词（字）边界的字符

示例：

```
> /^http:/.test('http://www.163.com')
< true
> /^http:/.test('ahttp://www.163.com')
< false
> /^http:/.test('https://www.163.com')
< false
> /\.jpg$/.test('1.jpg')
< true
> /\.jpg$/.test('1.jpg abc')
< false
> /\.jpg$/.test('1.png')
< false
> /\bis\b/.test('this')
< false
> /\bis\b/.test('that is tom')
< true
```

字符类

匹配一类字符串的一个

[0-9]	匹配数字，等价于\d
[a-z]	匹配英文小写字母

[A-Z]	匹配英文大写字母
[0-9a-zA-Z]	匹配数字或英文字母
[^0-9]	匹配非数字的一个字符

```
> /[0-9]/.test('abc')
< false
> /[^0-9]/.test('abc')
< true
> /[a-z]/.test('abc')
< true
> /./.test('abcd')
< true
> /./.test('123')
< true
> /./.test('%%^^^&&^&')
< true
```

元字符

具有特殊意义的字符

\d	匹配数字，相当于[0-9]
\D	匹配非数字，相当于[^0-9]
\w	匹配字母或数字或汉字或下划线
\W	匹配任意不是字母、数字、汉字或下划线的字符,[^\w]
\s	匹配任意的空白符，如空格、换行符、制表符等
\S	匹配任意不是空白符的字符
.（点号）	匹配除了换行符以外的任意字符
[...]	匹配方括号中的所有字符
[^...]	匹配非方括号中的所有字符

```
> /\d/.test('123')
< true
> /\d/.test('1ab')
< true
> /\D/.test('1ab')
< true
> /\D/.test('112')
< false
```

量词---限定符

出现次数

+	重复 1 次或更多次
*	重复 0 次或更多次（任意次数）
?	重复 0 次或 1 次（最多 1 次）
{n}	重复 n 次
{n,}	重复 n 次或更多次（最少 n 次）
{n,m}	重复 n 到 m 次

```
> /\d*/.test('abc')
< true
> /\d+/.test('abc')
< false
> /\d+/.test('1abc')
< true
> /https?:/.test('http://www.163.com')
< true
> /https?:/.test('https://www.163.com')
< true
> /https?:/.test('httpss://www.163.com')
< false
```

转义符

如果我们要匹配的字符是元字符，我们就必须在该特殊字符前面加上反斜杠“\”将其进行转义。

例如要匹配字面意义的“\”，就需要使用“\\”表示。

需要转义的字符有：$、(、)、*、+、.、[、]、?、\、/、^、{、}、|。

表达式	说明
\r，\n	回车和换行
\\	匹配“\”本身
\^，\$，\.	分别匹配“^”、“$”和“.”

```
> /http:///
< /http:/
> /http:\/\//.test('http://www.163.com')
< true
> /@163.com$/
< /@163.com$/
> /@163\.com$/.test('abc@163.com')
< true
> /@163.com$/.test('abc@163.com')
< true
> /@163.com$/.test('abc@163acom')
< true
> /@163\.com$/.test('abc@163.com')
```

多选分支---|

当一个字符串的某一子串具有多种可能时，采用分支结构来匹配，“|”表示多个子表达式之间“或”的关系，“|”是以()限定范围的，如果在“|”的左右两侧没有()来限定范围，那么它的作用范围即为“|”左右两侧整体。

表达式	说明
|	多个子表达式之间取“或”的关系

例如，“abc|def1”匹配的是“abc”或“def1”，而不是“abc1”或“def1”。如果要匹配“abc1”或“def1”，应该使用分组符，即“(abcd|efgh)1”。

捕获---捕获组（Capture Group）

保存匹配到的字符串，日后在用。

1.捕获组

捕获组就是把正则表达式中子表达式匹配的内容，保存到内存中以数字编号或手动命名的组里，以供后面引用。

表达式	说明
(Expression)	普通捕获组，将子表达式 Expression 匹配的内容保存到以数字编号的组里
(?<name> Expression)	命名捕获组，将子表达式 Expression 匹配的内容保存到以 name 命名的组里

普通捕获组（在不产生歧义的情况下，简称捕获组）是以数字进行编号的，编号规则是以“(”

从左到右出现的顺序，从 1 开始进行编号。通常情况下，编号为 0 的组表示整个表达式匹配的内容。
命名捕获组可以通过捕获组名，而不是序号对捕获内容进行引用，提供了更便捷的引用方式，不用关注捕获组的序号，也不用担心表达式部分变更会导致引用错误的捕获组。
示例：
/(.+)@(163|126|188)\.com$/ //匹配网易邮箱，捕获。

2.非捕获组
一些表达式中，不得不使用()，但又不需要保存()中子表达式匹配的内容，这时可以用非捕获组来抵消使用( )带来的副作用。

表达式	说明
(?:Expression)	进行子表达式 Expression 的匹配，并将匹配内容保存到最终的整个表达式的区配结果中，但 Expression 匹配的内容不单独保存到一个组内

示例：
/(.+)@(?:163|126|188)\.com$/ //匹配网易邮箱，但不捕获

3.反向引用
捕获组匹配的内容，可以在正则表达式的外部程序中进行引用，也可以在表达式中进行引用，表达式中引用的方式就是反向引用。
反向引用通常用来查找重复的子串，或是限定某一子串成对出现。

表达式	说明
\1，\2	对序号为 1 和 2 的捕获组的反向引用
\k<name>	对命名为 name 的捕获组的反向引用

举例：
“(a|b)\1”在匹配“abaa”时，匹配成功，匹配到的结果是“aa”。“(a|b)”在尝试匹配时，虽然既可以匹配“a”，也可以匹配“b”，但是在进行反向引用时，对应()中匹配的内容已经是固定的了。

4.环视（Look Around）
环视只进行子表达式的匹配，匹配内容不计入最终的匹配结果，是零宽度的。
环视按照方向划分有顺序和逆序两种，按照是否匹配有肯定和否定两种，组合起来就有四种环视。环视相当于对所在位置加了一个附加条件。

表达式	说明
(?<=Expression)	逆序肯定环视，表示所在位置左侧能够匹配 Expression
(?<!Expression)	逆序否定环视，表示所在位置左侧不能匹配 Expression
(?=Expression)	顺序肯定环视，表示所在位置右侧能够匹配 Expression
(?!Expression)	顺序否定环视，表示所在位置右侧不能匹配 Expression

举例：
“(?<=Windows)\d+”在匹配“Windows 2003”时，匹配成功，匹配结果为“2003”。我们知道“\d+”表示匹配一个以上的数字，而“(?<=Windows)”相当于一个附加条件，表示所在位置左侧必须为“Windows ”，它所匹配的内容并不计入匹配结果。同样的正则在匹配“Office 2003”时，匹配失败，因为这里任意一串数字子串的左侧都不是“Windows ”。

“(?!1)\d+”在匹配“123”时，匹配成功，匹配的结果为“23”。“\d+”匹配一个以上数字，但是附加条件“(?!1)”要求所在位置右侧不能是“1”，所以匹配成功的位置是“2”前面的位置。

正则表达式匹配的贪婪与非贪婪模式

1.什么是正则表达式的贪婪与非贪婪匹配

如：String str="abcaxc";

Patter p="ab*c";

贪婪匹配:正则表达式一般趋向于最大长度匹配，也就是所谓的贪婪匹配。如上面使用模式 p 匹配字符串 str，结果就是匹配到：abcaxc(ab*c)。

非贪婪匹配：就是匹配到结果就好，就少的匹配字符。如上面使用模式 p 匹配字符串 str，结果就是匹配到：abc(ab*c)。

2.编程中如何区分两种模式

默认是贪婪模式；在量词后面直接加上一个问号？就是非贪婪模式。

量词：{m,n}：m 到 n 个

*：任意多个

+：一个到多个

？：0 或一个

3.正则表达式量词分别是：贪婪的、惰性的、支配性的。

贪婪:? * + {n} {n,m} {n,}

惰性:?? *? +? {n}? {n,m}? {n,}?

支配:?+ *+ ++ {n}+ {n,m}+ {n,}+

分别的意思是:

零次或一次出现、零次或多次出现、一次或多次出现、恰好 N 次出现、至少 N 次最多 M 次出现、至少 N 次出现。

贪婪量词：先看整个字符串是否匹配，如果不匹配就把最后一个字符去掉在进行匹配，不匹配继续去掉最后一个字符，指导找到一个匹配或者不剩任何字符才停止。

惰性量词：先看第一个字符串是否匹配,如果第一个不匹配就在加入第二个字符串依此类推，指导找到一个匹配或者不剩任何字符才停止，贪婪量词与贪婪量词的方法正好相反.

浏览器对量词的支持还不完善，IE 和 OPERA 都不支持量词，MOZILLA 把支配量词看作是贪婪的例子：

```
var str = 'aabbazbbwwbbaa';
var arr =str.match(/.*bb/); //aabbazbbwwbb，贪婪的
var arr =str.match(/.*?bb/g); //aabb azbb wwbb 返回一个数组包含 3 个值，惰性的
```

str.match(regexp)

捕获匹配字符串


```
var url = 'http://blog.163.com/album?id=1#comment';
var reg = /(https?:)\/\/([^\/]+)(\/[^\?]*)?(\?[^#]*)?(#.*)?/;
var arr = url.match(reg);
var protocol = arr[1];
var host = arr[2];
var pathname = arr[3];
var search = arr[4];
var hash = arr[5];
```

str.replace(regexp/substr,replacement)

替换一个子串

```
var str = 'The price of tomato is 5, the price of apple is 10.';
str.replace(/(\d+)/g, '$1.00');
```

The price of tomatoes is 5.00, the price of apple is 10.00.

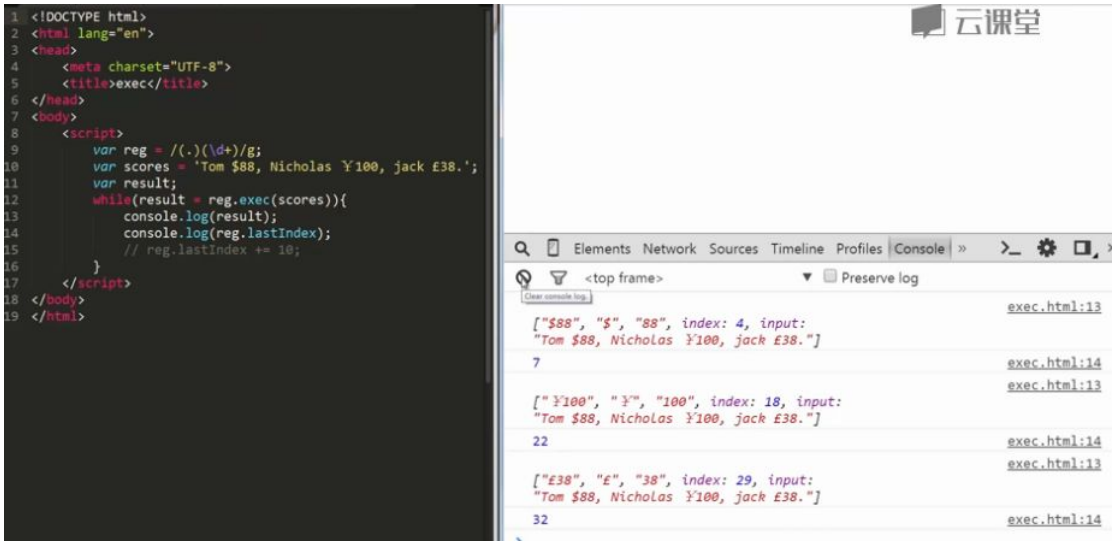
```
<script>
    var container = document.getElementById('container
    ');

    var html = '<label>网址:</label><input
        placeholder="以http://起始">';
    html = html.replace(/[<>]/g, function(m0){
        switch(m0){
            case '<':
                return '&lt;';
            case '>':
                return '&gt;';
        }
    });
    console.log(html);
    container.innerHTML = html;
</script>
```

regexpObj.exec(str)

更强大的检索

- 更详尽的结果：index
- 过程的状态：lastIndex



正则表达式优先级

在正则表达式中，同样存在优先级顺序。正则表达式存在元字符、限定符、转义字符、|等运算符。在匹配过程中，正则表达式都是先规定了这些运算或表达式的优先级。

正则表达式也可以像数学表达式一样来求值。也就是说，正则表达式可以从左到右，并按照一个给定的优先级来求值。

优先级顺序

运算符或表达式	说明
\	转义符
()、(?:)、(?=)、[]	圆括号或方括号
*、+、?、{n}、{n,}、{n,m}	限定符
^、$、\b、\B	位置和顺序
|	选择符，“或”运算

上面优先级是从高到低排列。

参考资源：

http://blog.csdn.net/lxcnn/article/details/4268033

十四、JSON

JSON 简介

JSON，全称“JavaScript Object Notation（JavaScript 对象表示）”，起源于 JavaScript 的对象和数组。JSON，说白了就是 JavaScript 用来处理数据的一种格式，是一种数据交换格式。

JSON，大部分都是**用来处理 JavaScript 和 web 服务器端之间的数据交换**，把后台 web 服务器的数据传递到前台，然后使用 JavaScript 进行处理，例如 ajax 等。

JSON 支持的语言非常多，包括 JavaScript、C#、PHP、Java 等等，这是由于 JSON 独立于语言的轻量级的数据交换格式，这个特点有点类似于 SQL 语言。

JSON 结构共有 2 种：

- **（1）对象结构；**
- **（2）数组结构；**

JSON，简单来说就是 JavaScript 中的对象或数组，所以这两种结构就是对象和数组。通过这两种结构就可以表示各种复杂的结构。

JSON 对象结构

对象结构是使用大括号“{}”括起来的，大括号内是由 0 个或多个用英文逗号分隔的“关键字:值”对（key:value）构成的。

语法：

```
var jsonObj =
{
    "键名 1":值 1,
    "键名 2":值 2,
    ……
```

```
6     "键名 n":值 n
7}
```

说明：

jsonObj 指的是 json 对象。对象结构是以“{”开始，到“}”结束。其中“键名”和“值”之间用英文冒号构成对，两个“键名:值”之间用英文逗号分隔。

注意，这里的键名是字符串，但是值可以是数值、字符串、对象、数组或逻辑 true 和 false。

1、从 JSON 中读数据

对于 JSON 对象结构，读取 JSON 非常简单，获取 JSON 中的数据共有 2 种方式。

语法：

```
jsonObj.key
jsonObj["key"]
```

说明：

jsonObj 指的是 JSON 对象，key 指的是键名。读取 JSON 数据使用的是“.”操作符，这个跟 JavaScript 对象读取属性值是差不多的。

2、向 JSON 写数据

对于 JSON 对象结构，要往 JSON 中增加一条数据，也是使用“.”操作符。

语法：

```
jsonObj.key = 值;
jsonObj["key"] = 值;
```

说明：

jsonObj 指的是 JSON 对象，key 指的是键名。

3、修改 JSON 中的数据

对于 JSON 对象结构，要修改 JSON 中的数据，也是使用“.”操作符。

语法：

```
1 jsonObj.key = 新值;
```

4、删除 JSON 中的数据

对于 JSON 对象结构，我们使用 delete 关键字来删除 JSON 中的数据。

语法：

```
1delete  jsonObj.key;
```

说明：

删除 JSON 中数据非常简单，只需要使用 delete 关键字即可。

5、遍历 JSON 对象

对于 JSON 对象结构，可以使用 for…in…循环来遍历 JSON 对象中的数据。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <script type="text/javascript">
        var obj =
        {
            "name":"helicopter",
```

```
            "age":23,
            "gender":"男",
        }
        for(var c in obj)
        {
            if(c=="name")
            {
                document.write("姓名是："+obj[c]);
            }
        }
    </script>
</head>
<body>
</body>
</html>
```

在浏览器预览效果如下：
姓名是：helicopter

分析：
由于变量 c 是字符串，因此不能使用 obj.c 来获取 JSON 数据，而必须使用 obj[c]来获取 JSON 数据。

JSON 数组结构

JSON 数组结构是用中括号“[]”括起来，中括号内部由 0 个或多个以英文逗号“,”分隔的值列表组成。

语法：

```
var arr =
[
    {
        "键名 1":值 1,
        "键名 2":值 2
    },
    {
        "键名 3":值 3,
        "键名 4":值 4
    },
    ……
```

说明：
arr 指的是 json 数组。数组结构是以“[”开始，到“]”结束，这一点跟 JSON 对象不同。不过在 JSON 数组结构中，每一对“{}”相当于一个 JSON 对象，大家看看像不像？而且语法都非常类似。
注意，这里的键名是字符串，但是值可以是数值、字符串、对象、数组或逻辑 true 和 false。JSON 数组结构也是非常简单的，只需要通过数组下标来获取哪一个数组元素（一个“{}”的内容就类似一个数组元素），然后再配合使用“.”操作符就可以获取相应数组元素的内

部数据。
对于获取、写入、修改、删除、遍历 JSON 数组结构中的数据，跟 JSON 对象结构的数据操作类似。

普通字符串、JSON 字符串和 JSON 对象

- 普通字符串，用""/''号括起来的字符串。
- JSON 对象，指的是符合 JSON 格式要求的 JavaScript 对象。
- JSON 字符串，指的是符合"JSON 格式"的字符串。

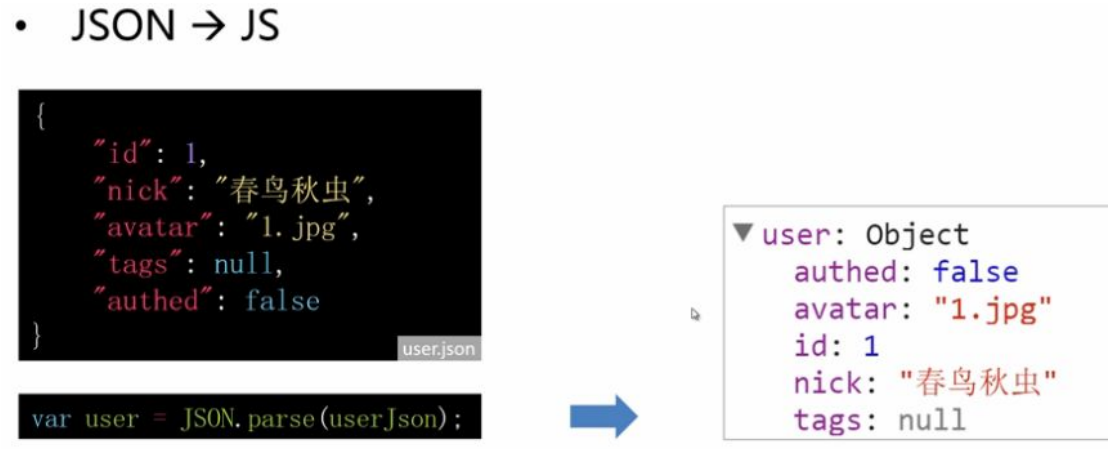
示例：

```
var str = "绿叶学习网 json 教程"; //普通字符串
var jsonObj = {"name":"helicopter","age":23,"gender":"男"}; //JSON 对象
var jsonStr = '{"name": "helicopter", "age":23, "gender": "男"}'; //JSON 字符串
```

JSON 是一个包含了函数 parse 和 stringify 的简单对象。parse 函数用来解析一个 JSON 文本（一个 JSON 格式的字符串）到一个 ECMAScript 值（例如 JSON 对象被解析为 ECMAScript 对象，JSON 数组被解析为 ECMAScript 数组，其它类型以此类推）；stringify 则相反，它是将一个 ECMAScript 值解析为一个 JSON 格式的字符串，比如将一个 ECMAScript 对象解析为一个 JSON 对象的字符串。

JSON.parse(text[,reviver]) --- JSON→JS

在 JavaScript 中，将 JSON 字符串转换为 JSON 对象非常有用。一般，在 Web 服务器后台向前台传输数据的过程中，往往都是用字符串形式来传输 JSON 数据。如果我们在前台想要获取 JSON 数据，就必须将 JSON 字符串转换为 JSON 对象才能操作。
现在大多数浏览器（IE8 及以上，Chrome 和 Firefox 差不多全部）自带原生 JSON 对象，提供 JSON.parse()方法来将 JSON 字符串转换为 JSON 对象。



JSON.parse（）方法在 IE6/7 下不支持，解决方案如下：

```
if(!window.JSON){
        window.JSON = {
            parse:function(sJSON) {
                return eval( '( '+sJSON+' )' );
            }
        };
```

```
}
```

eval()函数

在 JavaScript 中，eval()函数可以把一个字符串当做一个 JavaScript 表达式一样去执行它。
语法：eval(string)
string 表示一个字符串，是 eval()函数必选参数。eval()函数是有返回值的，如果参数字符串是一个表达式，就会返回表达式的值。如果参数字符串不是表达式，也就是没有值，那么就会返回“undefined”。

JSON.stringify(value[,replacer[,space]]) --- JS→JSON

在 JSON 中，可以使用 JSON.stringify()方法将 JSON 对象转换为 JSON 字符串。

JSON.stringify（）方法在 IE6/7 下不支持，解决方案如下：

方法一：直接引用 json2.js 文件。
步骤：

1. 从网上下载 json2.js 文件。下载地址
2. 在页面中引用该脚本：<script type="text/javascript" src="js/json2.js"></script>

方法二：使用如下代码：（以下代码未做验证）

```
if(!window.JSON){
    window.JSON = {
        stringify: function(obj){
            var result = "";
            for(var key in obj){
                if(typeof obj[key] == "string"){
                    // 如果属性值是 String 类型，属性值需要加上双引号
                    result += "\"" + key + "\":\"" + obj[key] + "\",";
                }else if(obj[key] instanceof RegExp){
                    // 如果属性是正则表达式，属性值只保留一对空大括号{}
                    result += "\"" + key + "\":{},";
                }else if(typeof obj[key] == "undefined" || obj[key] instanceof Function){
```

```
                    // 如果属性值是 undefined, 该属性被忽略。忽略方法。
                }else if(obj[key] instanceof Array){
                    // 如果属性值是数组
                    result += "\"" + key + "\":[";
                    var arr = obj[key];
                    for(var item in arr){
                        if(typeof arr[item] == "string"){
                            // 如果数组项是 String 类型，需要加上双引号
                            result += "\"" + arr[item] + "\",";
                        }else if(arr[item] instanceof RegExp){
                            // 如果属数组项是正则表达式，只保留一对空大括号{}
                            result += "{},";
                        }else if(typeof arr[item] == "undefined" || arr[item] instanceof Fun
ction){
                            // 如果数组项是 undefined, 则显示 null。如果是函数，则显示 null?。
                            result += null +",";
                        }else if(arr[item] instanceof Object){
                            //如果数组项是对象(非正则，非函数，非 null)，调用本函数处理
                            result += this.stringify(arr[item]) +",";
                        }else{
                            result += arr[item] + ",";
                        }
                    }
                    result = result.slice(0,-1)+"],"

                }else if(obj[key] instanceof Object){
                    // 如果属性值是对象(非 null，非函数，非正则)，调用本函数处理
                    result += "\"" + key + "\":" + this.stringify(obj[key]) + ",";
                }else{
                    result += "\"" + key + "\":" + obj[key] + ",";
                }
            }
            // 去除最后一个逗号,两边加{}
            return "{" + result.slice(0,-1) + "}";
        }
    };
}
```

参考资源：

1.JSON 介绍：

http://json.org/json-zh.html

2.IE6/7 不支持 JSON 分析：

http://www.cnblogs.com/duanhuajian/p/3392856.html

3.JS 中 typeof 与 instanceof 的区别

http://blog.sina.com.cn/s/blog_532751d90100iv1r.html

（上面的 JSON.parse（）方法兼容 IE6/7 方案中有用到 instanceof）