

## 第三章 JavaScript常用对象

### 3.1、数组对象

#### 3.1.1、概述

数组也是对象的一种，数组是一种用于表达有顺序关系的值的集合的语言结构，也就是同类数据元素的有序集合。

数组的存储性能比普通对象要好，在开发中我们经常使用数组来存储一些数据。但是在JavaScript中支持数组可以是不同的元素，这跟JavaScript的弱类型有关，此处不用纠结，我们大多数时候都是相同类型元素的集合。数组内的各个值被称作元素，每一个元素都可以通过索引（下标）来快速读取，索引是从零开始的整数。

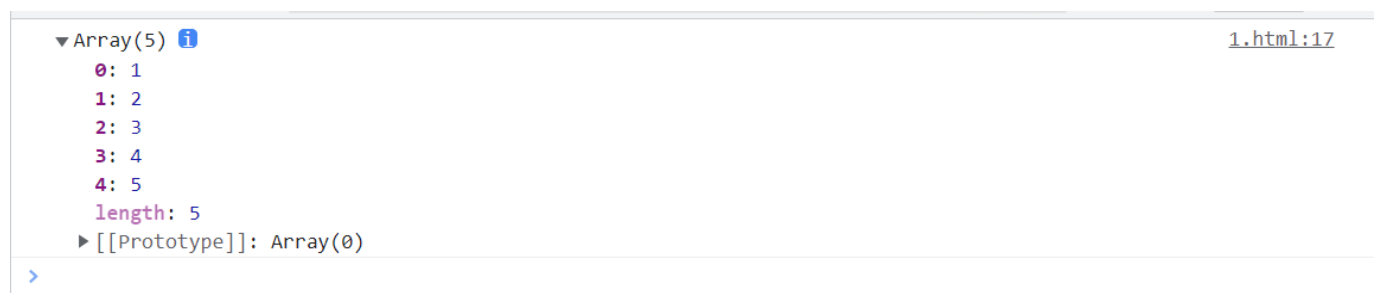
使用typeof检查一个数组对象时，会返回object。

#### 3.1.2、创建数组

##### 3.1.2.1、使用对象创建

- 同类型有序数组创建：

```
<script>
    var arr = new Array();
    arr[0] = 1;
    arr[1] = 2;
    arr[2] = 3;
    arr[3] = 4;
    arr[4] = 5;
    console.log(arr);
</script>
```



- 不同类型有序数组创建：

```
<script>
  var arr = new Array();
  arr[0] = 1;
  arr[1] = "2";
  arr[2] = 3;
  arr[3] = "4";
  arr[4] = 5;
  arr[5] = "6";
  console.log(arr)
</script>
```



### 3.1.2.2、使用字面量创建

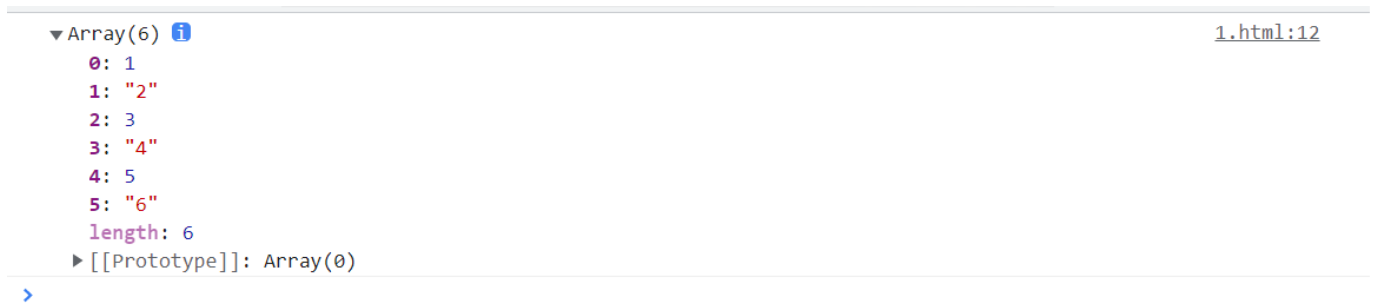
- 同类型有序数组创建:

```
<script>
  var arr = [1, 2, 3, 4, 5];
  console.log(arr)
</script>
```

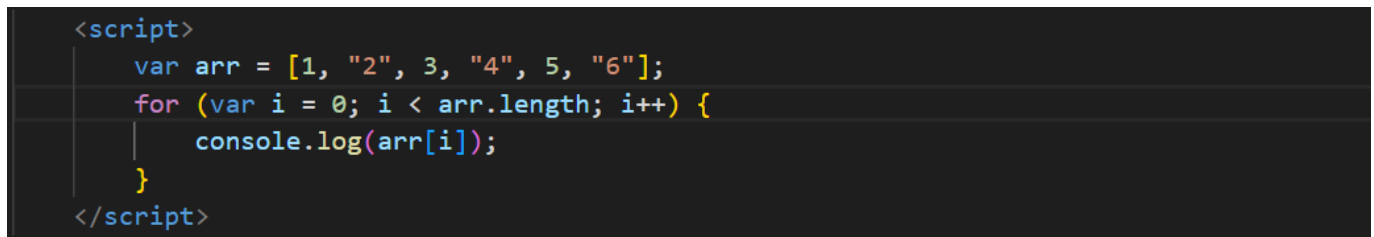


- 不同类型有序数组创建:

```
<script>
  var arr = [1, "2", 3, "4", 5, "6"];
  console.log(arr)
</script>
```

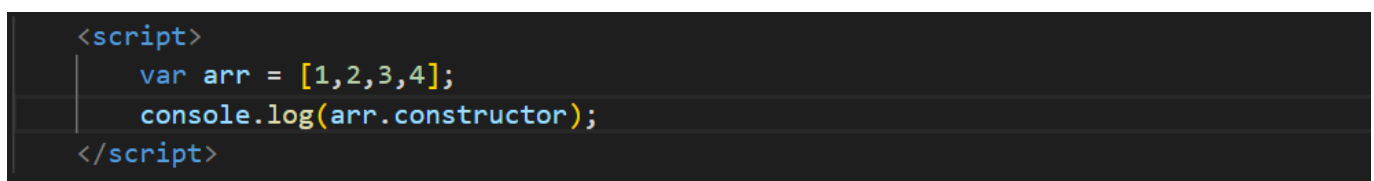


### 3.1.3、遍历数组



### 3.1.4、数组属性

- constructor属性演示：返回创建数组对象的原型函数



- length属性演示：设置或返回数组元素的个数

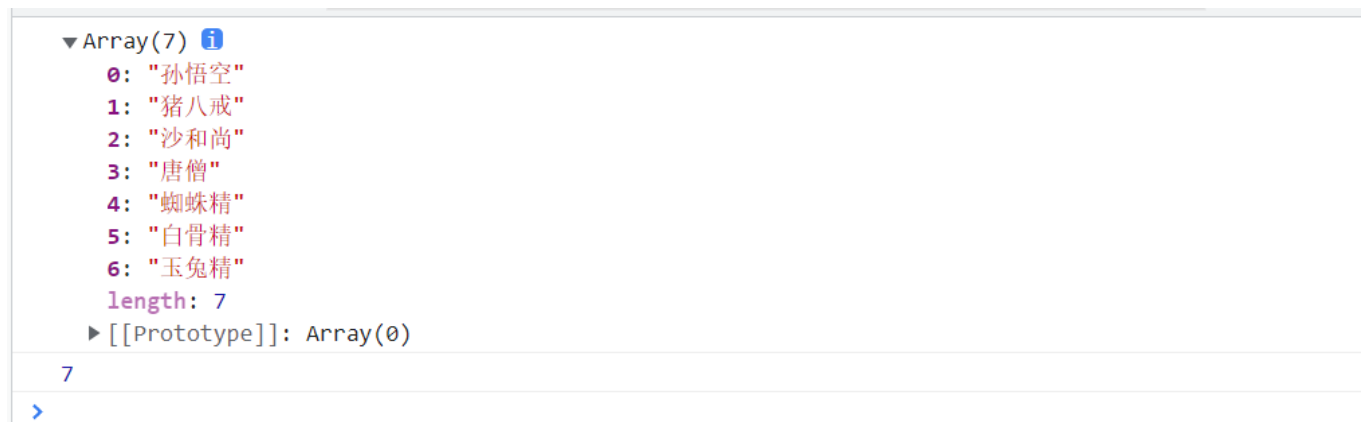
```
<script>
  var arr = [1,2,3,4];
  console.log(arr.length);
</script>
```



### 3.1.5、数组方法

- **push()方法演示：**该方法可以向数组的末尾添加一个或多个元素，并返回数组的新的长度

```
<script>
  var arr = ["孙悟空", "猪八戒", "沙和尚"];
  var result = arr.push("唐僧", "蜘蛛精", "白骨精", "玉兔精");
  console.log(arr);
  console.log(result);
</script>
```



- **pop()方法演示：**该方法可以删除数组的最后一个元素，并将被删除的元素作为返回值返回

```
<script>
  var arr = ["孙悟空", "猪八戒", "沙和尚"];
  var result = arr.pop();
  console.log(arr);
  console.log(result);
</script>
```

▼ Array(2) ⓘ

0: "孙悟空"

1: "猪八戒"

length: 2

▶ [[Prototype]]: Array(0)

沙和尚

- **unshift()方法演示：**该方法向数组开头添加一个或多个元素，并返回新的数组长度

```
<script>
    var arr = ["孙悟空", "猪八戒", "沙和尚"];
    var result = arr.unshift("牛魔王", "二郎神");
    console.log(arr);
    console.log(result);
</script>
```

▼ Array(5) ⓘ

0: "牛魔王"

1: "二郎神"

2: "孙悟空"

3: "猪八戒"

4: "沙和尚"

length: 5

▶ [[Prototype]]: Array(0)

5

>

- **shift()方法演示：**该方法可以删除数组的第一个元素，并将被删除的元素作为返回值返回

```
<script>
    var arr = ["孙悟空", "猪八戒", "沙和尚"];
    var result = arr.shift();
    console.log(arr);
    console.log(result);
</script>
```

▼ Array(2) ⓘ

0: "猪八戒"

1: "沙和尚"

length: 2

▶ [[Prototype]]: Array(0)

孙悟空

>

- **forEach()方法演示：**该方法可以用来遍历数组

forEach()方法需要一个函数作为参数，像这种函数，由我们创建但是不由我们调用的，我们称为回调函数。数组中有几个元素函数就会执行几次，每次执行时，浏览器会将遍历到的元素，以实

参的形式传递进来，我们可以来定义形参，来读取这些内容，浏览器会在回调函数中传递三个参数：

- 第一个参数：就是当前正在遍历的元素
- 第二个参数：就是当前正在遍历的元素的索引
- 第三个参数：就是正在遍历的数组

注意：这个方法只支持IE8以上的浏览器，IE8及以下的浏览器均不支持该方法，所以如果需要兼容IE8，则不要使用forEach()，还是使用for循环来遍历数组。

```
<script>
  var arr = ["孙悟空", "猪八戒", "沙和尚"];
  arr.forEach(function (value, index, obj) {
    console.log(value + " #### " + index + " #### " + obj);
  });
</script>
```

孙悟空 #### 0 #### 孙悟空,猪八戒,沙和尚

猪八戒 #### 1 #### 孙悟空,猪八戒,沙和尚

沙和尚 #### 2 #### 孙悟空,猪八戒,沙和尚

- slice()方法演示：该方法可以用来从数组提取指定元素，该方法不会改变元素数组，而是将截取到的元素封装到一个新数组中返回

参数：

- 第一个参数：截取开始的位置的索引，包含开始索引
- 第二个参数：截取结束的位置的索引，不包含结束索引，第二个参数可以省略不写，此时会截取从开始索引往后的所有元素

注意：索引可以传递一个负值，如果传递一个负值，则从后往前计算，-1代表倒数第一个，-2代表倒数第二个。

```
<script>
  var arr = ["孙悟空", "猪八戒", "沙和尚", "唐僧", "白骨精"];
  var result = arr.slice(1, 4);
  console.log(result);
  result = arr.slice(3);
  console.log(result);
  result = arr.slice(1, -2);
  console.log(result);
</script>
```

▼ Array(3) ⓘ	1
0: "猪八戒"	
1: "沙和尚"	
2: "唐僧"	
length: 3	
▶ [[Prototype]]: Array(0)	
▼ Array(2) ⓘ	1
0: "唐僧"	
1: "白骨精"	
length: 2	
▶ [[Prototype]]: Array(0)	
▼ Array(2) ⓘ	1
0: "猪八戒"	
1: "沙和尚"	
length: 2	
▶ [[Prototype]]: Array(0)	

- splice()方法演示：该方法可以用于删除数组中的指定元素，该方法会影响到原数组，会将指定元素从原数组中删除，并将被删除的元素作为返回值返回

参数：

- 第一个参数：表示开始位置的索引
- 第二个参数：表示要删除的元素数量
- 第三个参数及以后参数：可以传递一些新的元素，这些元素将会自动插入到开始位置索引前边

```

<script>
  var arr = ["孙悟空", "猪八戒", "沙和尚", "唐僧", "白骨精"];
  var result = arr.splice(3, 2);
  console.log(arr);
  console.log(result);
  result = arr.splice(1, 0, "牛魔王", "铁扇公主", "红孩儿");
  console.log(arr);
  console.log(result);
</script>

```

▼ Array(6) ⓘ 1.htm  
0: "孙悟空"  
1: "牛魔王"  
2: "铁扇公主"  
3: "红孩儿"  
4: "猪八戒"  
5: "沙和尚"  
length: 6  
▶ [[Prototype]]: Array(0)

▼ Array(2) ⓘ 1.htm  
0: "唐僧"  
1: "白骨精"  
length: 2  
▶ [[Prototype]]: Array(0)

▼ Array(6) ⓘ 1.htm  
0: "孙悟空"  
1: "牛魔王"  
2: "铁扇公主"  
3: "红孩儿"  
4: "猪八戒"  
5: "沙和尚"  
length: 6  
▶ [[Prototype]]: Array(0)

▼ Array(0) ⓘ 1.htm  
length: 0  
▶ [[Prototype]]: Array(0)

>

- **concat()方法演示：**该方法可以连接两个或多个数组，并将新的数组返回，该方法不会对原数组产生影响

```
<script>  
    var arr = ["孙悟空", "猪八戒", "沙和尚"];  
    var arr2 = ["白骨精", "玉兔精", "蜘蛛精"];  
    var arr3 = ["二郎神", "太上老君", "玉皇大帝"];  
    var result = arr.concat(arr2, arr3, "牛魔王", "铁扇公主");  
    console.log(result);  
</script>
```

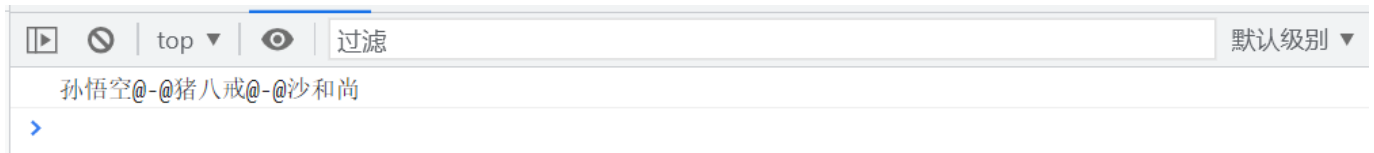
▼ Array(11) ⓘ  
0: "孙悟空"  
1: "猪八戒"  
2: "沙和尚"  
3: "白骨精"  
4: "玉兔精"  
5: "蜘蛛精"  
6: "二郎神"  
7: "太上老君"  
8: "玉皇大帝"  
9: "牛魔王"  
10: "铁扇公主"  
length: 11  
▶ [[Prototype]]: Array(0)

>



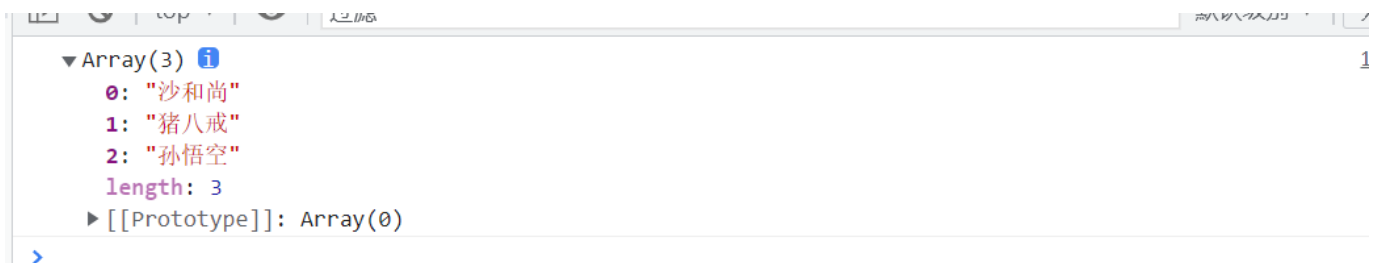
- **join()方法演示：**该方法可以将数组转换为一个字符串，该方法不会对原数组产生影响，而是将转换后的字符串作为结果返回，在join()中可以指定一个字符串作为参数，这个字符串将会成为数组中元素的连接符，如果不指定连接符，则默认使用，作为连接符

```
<script>
  var arr = ["孙悟空", "猪八戒", "沙和尚"];
  var result = arr.join("@-@");
  console.log(result);
</script>
```



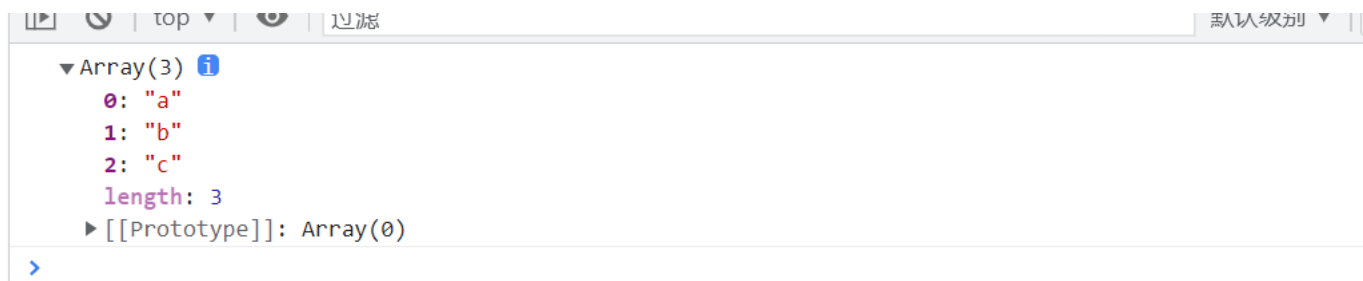
- **reverse()方法演示：**该方法用来反转数组（前边的去后边，后边的去前边），该方法会直接修改原数组

```
<script>
  var arr = ["孙悟空", "猪八戒", "沙和尚"];
  arr.reverse();
  console.log(arr);
</script>
```



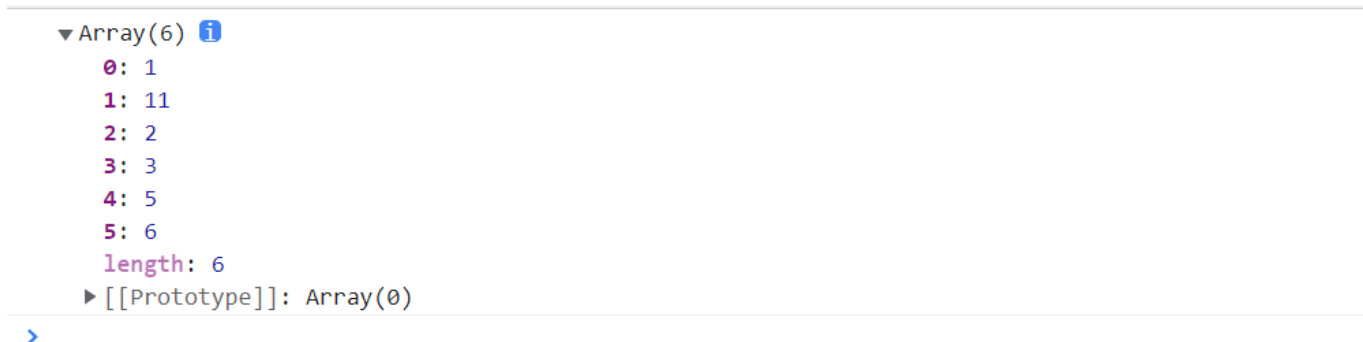
- **sort()方法演示：**该方法可以用来对数组中的元素进行排序，也会影响原数组，默认会按照Unicode编码进行排序

```
<script>
  var arr = ["b", "c", "a"];
  arr.sort();
  console.log(arr);
</script>
```



注意：即使对于纯数字的数组，使用`sort()`排序时，也会按照Unicode编码来排序，所以对数字进排序时，可能会得到错误的结果。

```
<script>
  var arr = [1, 3, 2, 11, 5, 6];
  arr.sort();
  console.log(arr);
</script>
```



我们可以自己来指定排序的规则，我们可以在`sort()`添加一个回调函数，来指定排序规则，回调函数中需要定义两个形参，浏览器将会分别使用数组中的元素作为实参去调用回调函数，使用哪个元素调用不确定，但是肯定的是在数组中a一定在b前边，浏览器会根据回调函数的返回值来决定元素的顺序，如下：

- 如果返回一个大于0的值，则元素会交换位置
- 如果返回一个小于0的值，则元素位置不变
- 如果返回一个等于0的值，则认为两个元素相等，也不交换位置 经过上边的规则，我们可以总结下：
- 如果需要升序排列，则返回  $a-b$
- 如果需要降序排列，则返回  $b-a$

```
<script>
  var arr = [1, 3, 2, 11, 5, 6];
  arr.sort(function (a, b) {
    return a - b;
  });
  console.log(arr);
</script>
```

```
▼ Array(6) ⓘ
  0: 1
  1: 2
  2: 3
  3: 5
  4: 6
  5: 11
  length: 6
  ► [[Prototype]]: Array(0)
```

## 3.2、函数对象

### 3.2.1、call()和apply()

call()和apply()这两个方法都是函数对象的方法，需要通过函数对象来调用，当对函数调用call()和apply()都会调用函数执行，在调用call()和apply()可以将一个对象指定为第一个参数，此时这个对象将会成为函数执行时的this，call()方法可以将实参在对象之后依次传递，apply()方法需要将实参封装到一个数组中统一传递，如下演示：

**call()方法演示：**

```

<script>
    function fun(a, b) {
        console.log("a = " + a);
        console.log("b = " + b);
        console.log("fun = " + this);
    }

    var obj = {
        name: "obj",
        sayName: function () {
            console.log(this.name);
        }
    };

    fun(2, 3);
    console.log("=====");
    fun.call(obj, 2, 3);
</script>

```

a = 2	<a href="#">1.1</a>
b = 3	<a href="#">1.1</a>
fun = [object Window]	<a href="#">1.1</a>
=====	<a href="#">1.1</a>
a = 2	<a href="#">1.1</a>
b = 3	<a href="#">1.1</a>
fun = [object Object]	<a href="#">1.1</a>

注意：默认fun()函数调用，this指向的是window对象，你可以使用call()调用函数，在调用的时候传入一个对象，这个对象就是this所指向的对象，也就是说，可以自己指定this的指向，然后从第二个参数开始，实参将会依次传递

**apply()方法演示：**

```

<script>
    function fun(a, b) {
        console.log("a = " + a);
        console.log("b = " + b);
        console.log("fun = " + this);
    }

    var obj = {
        name: "obj",
        sayName: function () {
            console.log(this.name);
        }
    };

    fun(2, 3);
    console.log("=====");
    fun.apply(obj, [2, 3]);
</script>

```

```

a = 2
b = 3
fun = [object Window]
=====
a = 2
b = 3
fun = [object Object]
>

```

注意：默认fun()函数调用，this指向的是window对象，你可以使用apply()调用函数，在调用的时候传入一个对象，这个对象就是this所指向的对象，也就是说，可以自己指定this的指向，然后从第二个参数开始，需要制定一个实参数组进行参数传递

### 3.2.2、this指向

- 以函数形式调用时，this永远都是window
- 以方法的形式调用时，this是调用方法的对象
- 以构造函数的形式调用时，this是新创建的那个对象
- 使用call和apply调用时，this是传入的那个指定对象

### 3.2.3、arguments参数

在调用函数时，浏览器每次都会传递进两个隐含的参数：

- 函数的上下文对象：this

- 封装实参的对象：arguments

arguments是一个类数组对象，它也可以通过索引来操作数据，也可以获取长度，在调用函数时，我们所传递的实参都会在arguments中保存，比如：arguments.length 可以用来获取实参的长度，我们即使不定义形参，也可以通过arguments来使用实参，只不过比较麻烦，例如：

arguments[0]：表示第一个实参 arguments[1]：表示第二个实参 ... 它里边有一个属性叫做 callee，这个属性对应一个函数对象，就是当前正在指向的函数的对象。

**arguments对象演示：**

```
<script>
function fun(a, b) {
    // 通过下标获取第一个参数
    console.log(arguments[0]);
    // 通过下标获取第二个参数
    console.log(arguments[1]);
    // 获取实参的个数
    console.log(arguments.length);
    // 看看它的函数对象
    console.log(arguments.callee);
    console.log(arguments.callee == fun);
}

fun("Hello", "World");
</script>
```

Hello

World

2

```
f fun(a, b) {
    // 通过下标获取第一个参数
    console.log(arguments[0]);
    // 通过下标获取第二个参数
    console.log(arguments[1]);
    // 获取实参的个数
    console.log(arguments.length);
    // 看看它的函数对象
    console.log(arguments.cal...
```

true

>

### 3.3、Date对象

在JavaScript中使用Date对象来表示一个时间，如果直接使用构造函数创建一个Date对象，则会封装为当前代码执行的时间。

案例演示：

```

<script>
    var date = new Date();
    console.log(date);

    console.log(date.getFullYear()); // 获取当前日期对象的年份(四位数字年份)
    console.log(date.getMonth()); // 获取当前日期对象的月份(0 ~ 11)
    console.log(date.getDate()); // 获取当前日期对象的日数(1 ~ 31)
    console.log(date.getHours()); // 获取当前日期对象的小时(0 ~ 23)
    console.log(date.getMinutes()); // 获取当前日期对象的分钟(0 ~ 59)
    console.log(date.getSeconds()); // 获取当前日期对象的秒钟(0 ~ 59)
    console.log(date.getMilliseconds()); // 获取当前日期对象的毫秒(0 ~ 999)

</script>

```

Mon Feb 26 2024 10:07:08 GMT+0800 (中国标准时间)

2024

1

26

10

7

8

577

>

### 3.4、Math对象

Math和它的对象不同，它不是一个构造函数，它属于一个工具类不用创建对象，它里边封装了数学运算相关的属性和方法。

案例演示：

```

<script>
  /*固定值*/
  console.log("PI = " + Math.PI);
  console.log("E = " + Math.E);
  console.log("=====");
  /*正数*/
  console.log(Math.abs(1));           //可以用来计算一个数的绝对值
  console.log(Math.ceil(1.1));       //可以对一个数进行向上取整，小数位只要有值就自动进1
  console.log(Math.floor(1.99));     //可以对一个数进行向下取整，小数部分会被舍掉
  console.log(Math.round(1.4));      //可以对一个数进行四舍五入取整
  console.log("=====");
  /*负数*/
  console.log(Math.abs(-1));          //可以用来计算一个数的绝对值
  console.log(Math.ceil(-1.1));       //可以对一个数进行向上取整，小数部分会被舍掉
  console.log(Math.floor(-1.99));     //可以对一个数进行向下取整，小数位只要有值就自动进1
  console.log(Math.round(-1.4));      //可以对一个数进行四舍五入取整
  console.log("=====");
  /*随机数*/
  //Math.random(): 可以用来生成一个0-1之间的随机数
  //生成一个0-x之间的随机数: Math.round(Math.random()*x)
  //生成一个x-y之间的随机数: Math.round(Math.random()*(y-x)+x)
  console.log(Math.round(Math.random() * 10));           //生成一个0-10之间的随机数
  console.log(Math.round(Math.random() * (10 - 1) + 1)); //生成一个1-10之间的随机数
  console.log("=====");
  /*数学运算*/
  console.log(Math.pow(12, 3)); //Math.pow(x,y): 返回x的y次幂
  console.log(Math.sqrt(4));   //Math.sqrt(x) : 返回x的平方根
</script>

```

PI = 3.141592653589793

E = 2.718281828459045

=====

1

2

1

1

=====

1

-1

-2

-1

=====

3

2

=====

1728

2

>

## 3.5、String对象

### 3.5.1、概述



在JS中为我们提供了三个包装类，通过这三个包装类可以将基本数据类型的数据转换为对象

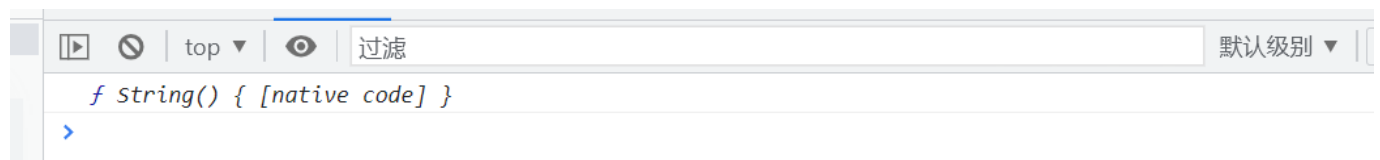
- String(): 可以将基本数据类型字符串转换为String对象
- Number(): 可以将基本数据类型的数字转换为Number对象
- Boolean(): 可以将基本数据类型的布尔值转换为Boolean对象

但是注意：我们在实际应用中不会使用基本数据类型的对象，如果使用基本数据类型的对象，在做一些比较时可能会带来一些不可预期的结果，在这一章节中，我们重点介绍String()对象的属性和方法。

### 3.5.2、字符串属性

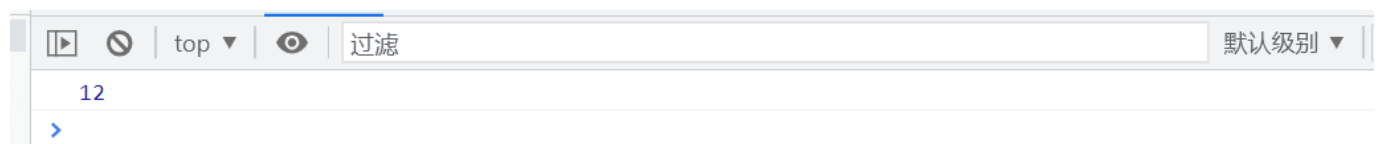
constructor属性演示：返回创建字符串对象的原型函数

```
<script>
  var str = "Hello,World!";
  console.log(str.constructor);
</script>
```



length属性演示：可以用来获取字符串的长度

```
<script>
  var str = "Hello,World!";
  console.log(str.length);
</script>
```



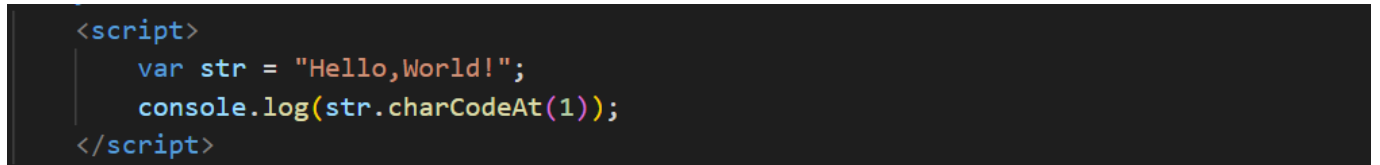
### 3.5.3、字符串方法

charAt()方法演示：该方法可以根据索引获取指定位置的字符

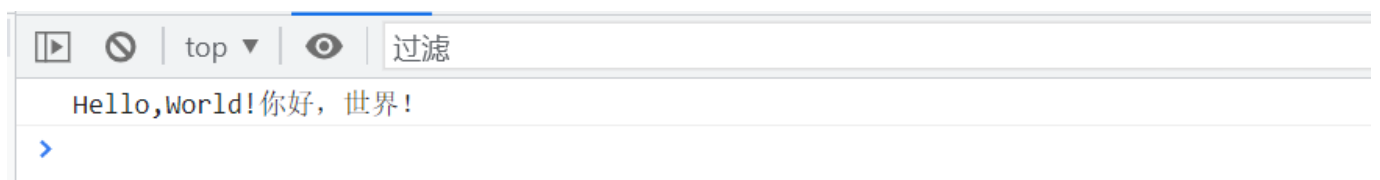
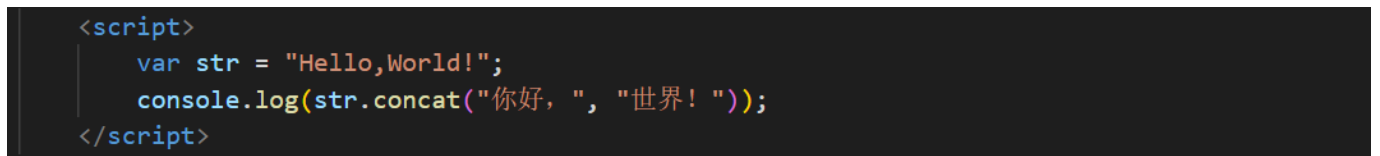
```
<script>
  var str = "Hello,World!";
  console.log(str.charAt(1));
</script>
```



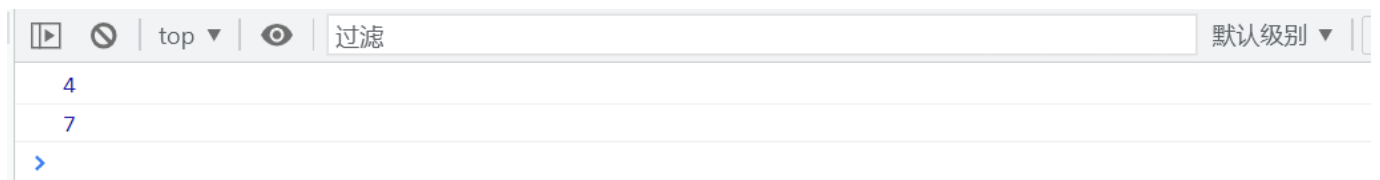
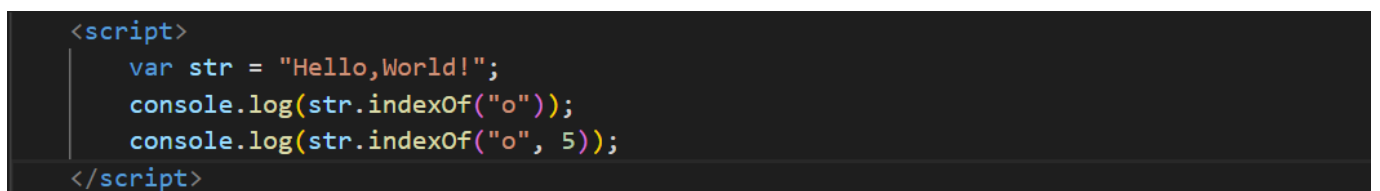
**charCodeAt()方法演示：该方法获取指定位置字符的字符编码（Unicode编码）**



**concat()方法演示：该方法可以用来连接两个或多个字符串**



**indexOf()方法演示：该方法可以检索一个字符串中是否含有指定内容，如果字符串中含有该内容，则会返回其第一次出现的索引，如果没有找到指定的内容，则返回-1，可以指定一个第二个参数，指定开始查找的位置**



**lastIndexOf()方法演示：该方法的用法和indexOf()一样，不同的是indexOf是从前往后找，而lastIndexOf是从后往前找，也可以指定开始查找的位置**

```
<script>
    var str = "Hello,World!";
    console.log(str.lastIndexOf("o"));
    console.log(str.lastIndexOf("o", 5));
</script>
```



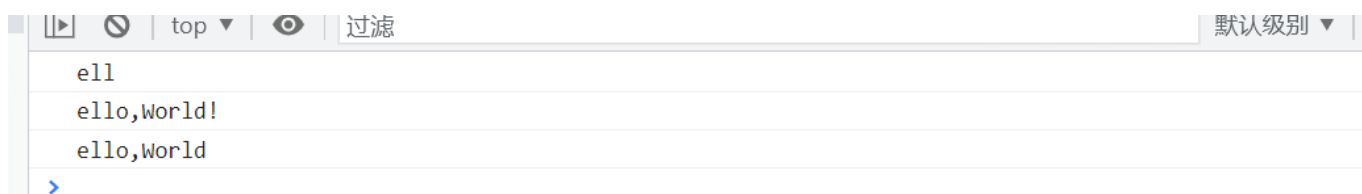
**slice()方法演示：**可以从字符串中截取指定的内容，不会影响原字符串，而是将截取到内容返回

参数：

- 第一个参数：开始位置的索引（包括开始位置）
- 第二个参数：结束位置的索引（不包括结束位置），如果省略第二个参数，则会截取到后边所有的

注意：也可以传递一个负数作为参数，负数的话将会从后边计算

```
<script>
    var str = "Hello,World!";
    var result = str.slice(1, 4);
    console.log(result);
    result = str.slice(1);
    console.log(result);
    result = str.slice(1, -1);
    console.log(result);
</script>
```



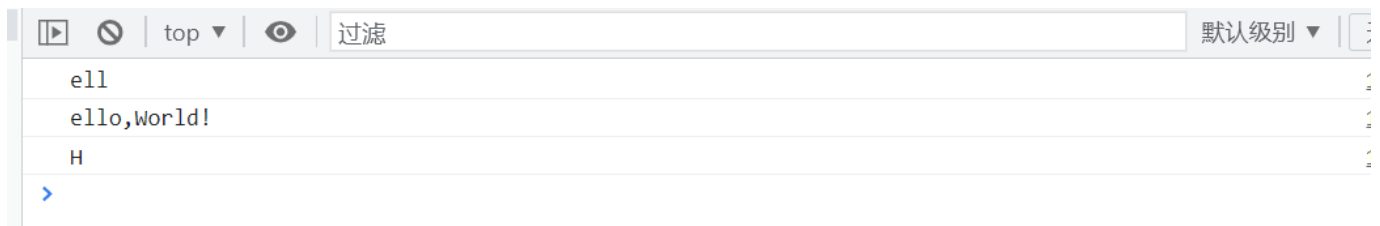
**substring()方法演示：**可以用来截取一个字符串，它和slice()类似

参数：

- 第一个参数：开始截取位置的索引（包括开始位置）
- 第二个参数：结束位置的索引（不包括结束位置），如果省略第二个参数，则会截取到后边所有的

注意：不同的是这个方法不能接受负值作为参数，如果传递了一个负值，则默认使用 0，而且它还自动调整参数的位置，如果第二个参数小于第一个，则自动交换

```
<script>
    var str = "Hello,World!";
    var result = str.substr(1, 4);
    console.log(result);
    result = str.substr(1);
    console.log(result);
    result = str.substr(1, -1);
    console.log(result);
</script>
```

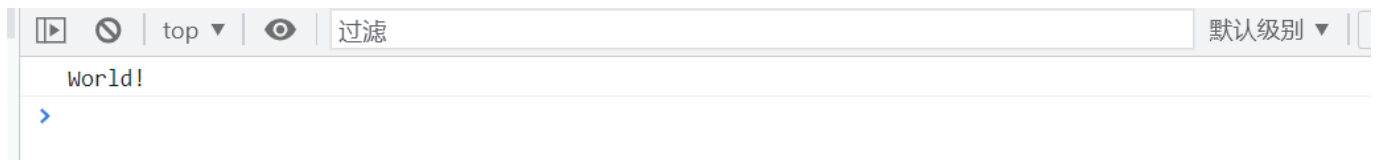


**substr()方法演示：该方法用来截取字符串**

参数：

- 第一个参数：截取开始位置的索引
- 第二个参数：截取的长度

```
<script>
    var str = "Hello,World!";
    var result = str.substr(6, 6);
    console.log(result);
</script>
```



**split()方法演示：该方法可以将一个字符串拆分为一个数组，需要一个字符串作为参数，将会根据该字符串去拆分数组**

```
<script>
    var str = "Hello,World!";
    var result = str.split(",");
    console.log(result);
</script>
```

```
> result
< (2) ['Hello', 'World!']
```

**toUpperCase()方法演示：将一个字符串转换为大写并返回**

```
<script>
  var str = "Hello,World!";
  var result = str.toUpperCase();
  console.log(result);
</script>
```

```
HELLO,WORLD!
```

**toLowerCase()方法演示：将一个字符串转换为小写并返回**

```
<script>
  var str = "Hello,World!";
  var result = str.toLowerCase();
  console.log(result);
</script>
```

```
hello,world!
```

## 3.6、RegExp对象

### 3.6.1、概述

正则表达式用于定义一些字符串的规则，计算机可以根据正则表达式，来检查一个字符串是否符合规则，获取将字符串中符合规则的内容提取出来。

使用typeof检查正则对象，会返回object。

### 3.6.2、创建正则对象

#### 3.6.2.1、使用对象创建

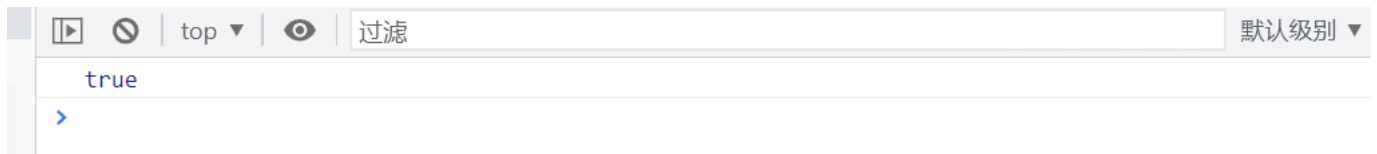
语法格式：

```
<script>
    var 变量名 = new RegExp("正则表达式", "匹配模式");
</script>
```

匹配模式：

- i: 忽略大小写
  - g: 全局匹配模式
  - ig: 忽略大小写且全局匹配模式
- 案例演示：**

```
<script>
    // 这个正则表达式可以来检查一个字符串中是否含有ab
    var reg = new RegExp("ab", "i");
    var str = "Abc";
    var result = reg.test(str);
    console.log(result);
</script>
```



### 3.6.2.2、使用字面量创建

语法格式：

```
<script>
    var 变量名 = /正则表达式/匹配模式;
</script>
```

匹配模式：

- i: 忽略大小写
- g: 全局匹配模式
- m: 执行多行匹配




注意：可以为一个正则表达式设置多个匹配模式，且顺序无所谓

案例演示：

```

<script>
    // 这个正则表达式可以来检查一个字符串中是否含有a
    var reg = /a/i;
    var str = "Abc";
    var result = reg.test(str);
    console.log(result);
</script>

```


 
top ▼ 
 
过滤 
默认级别 ▼

```

true
>

```

### 3.6.3、正则进阶

**需求信息：**创建一个正则表达式，检查一个字符串中是否有a或b

**语法格式：**使用 | 表示或者的意思

```

<script>
    // 这个正则表达式可以来检查一个字符串中是否含有a或b
    var reg = /a|b/i;
    var str1 = "Ack";
    var result1 = reg.test(str1);
    console.log(result1);
    var str2='glue';
    var result2=reg.test(str2);
    console.log(result2);
</script>

```

```

true
false
>

```

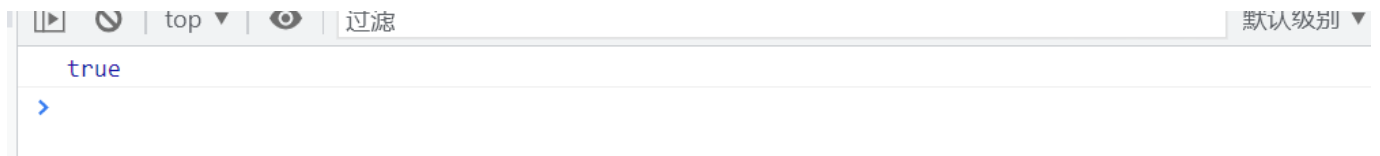
**需求信息：**创建一个正则表达式，检查一个字符串中是否有字母

**语法格式：**[ ] 里的内容也是或的关系

```

<script>
    // 这个正则表达式可以来检查一个字符串中是否含有字母
    var reg = /[A-z]/;
    var str = "Abc";
    var result = reg.test(str);
    console.log(result);
</script>

```



常见组合：

- [a-z]：任意小写字母
- [A-Z]：任意大写字母
- [A-z]：任意字母
- [0-9]：任意数字

**需求信息：创建一个正则表达式，检查一个字符串中是否含有 abc 或 adc 或 aec**

```
<script>
// 这个正则表达式可以来检查一个字符串中是否含有abc或adc或aec
var reg = /a[bde]c/;
var str = "abc123";
var result = reg.test(str);
console.log(result);
</script>
```

判断除了某些字符序列是否还有其他字符，使用[^字符序列]

常见组合：

- [^a-z]：除了任意小写字母
- [^A-Z]：除了任意大写字母
- [^A-z]：除了任意字母
- [^0-9]：除了任意数字

**需求信息：创建一个正则表达式，检查一个字符串中是否除了数字还有其它字母**

```
<script>
// 这个正则表达式可以来检查一个字符串中是否除了数字还有其它字母
var reg = /^[^0-9]/;
var str = "0123456789";
var result = reg.test(str);
console.log(result);
</script>
```



### 3.6.4、正则方法

这些正则方法其实都是字符串的方法，但是它的参数需要传递正则表达式，在这里，我就先称为正则方法。

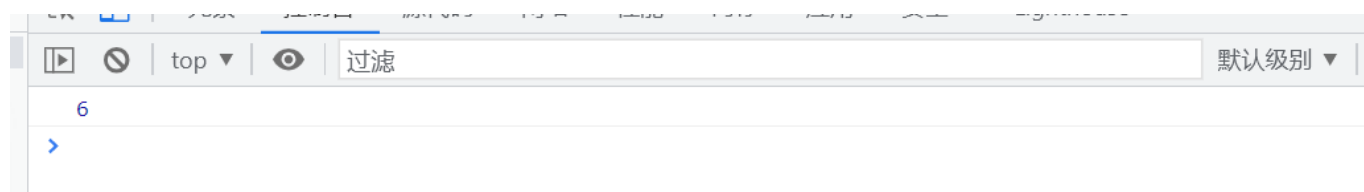
**split()方法演示：**该方法可以将一个字符串拆分为一个数组，方法中可以传递一个正则表达式作为参数，这样方法将会根据正则表达式去拆分字符串，这个方法即使不指定全局匹配，也会全都拆分

```
<script>
  var str = "1a2b3c4d5e6f7";
  var result = str.split(/[A-z]/);
  console.log(result);
</script>
```

```
> result
< (7) ['1', '2', '3', '4', '5', '6', '7']
```

**search()方法演示：**该方法可以搜索字符串中是否含有指定内容，如果搜索到指定内容，则会返回第一次出现的索引，如果没有搜索到返回-1，它可以接受一个正则表达式作为参数，然后会根据正则表达式去检索字符串，search()只会查找第一个，即使设置全局匹配也没用

```
<script>
  var str = "hello abc hello aec afc";
  var result = str.search(/a[bef]c/);
  console.log(result);
</script>
```



**match()方法演示：**该方法可以根据正则表达式，从一个字符串中将符合条件的内容提取出来，默认情况下我们的match()只会找到第一个符合要求的内容，找到以后就停止检索，我们可以设置正则表达式为全局匹配模式，这样就会匹配到所有的内容，可以为一个正则表达式设置多个匹配模式，且顺序无所谓，match()会将匹配到的内容封装到一个数组中返回，即使只查询到一个结果

```
<script>
  var str = "1a2a3a4a5e6f7A8B9C";
  var result = str.match(/[a-z]/ig);
  console.log(result);
</script>
```

```
> result
< (9) ['a', 'a', 'a', 'a', 'e', 'f', 'A', 'B', 'C']
```

**replace()方法演示：**该方法可以将字符串中指定内容替换为新的内容，默认只会替换第一个，但是可以设置全局匹配替换全部

参数：

- 第一个参数：被替换的内容，可以接受一个正则表达式作为参数
- 第二个参数：新的内容

```
<script>
  var str = "1a2a3a4a5e6f7A8B9C";
  var result = str.replace(/[a-z]/gi, "--");
  console.log(result);
</script>
```

```
1--2--3--4--5--6--7--8--9--
```

### 3.6.5、正则量词

通过量词可以设置一个内容出现的次数，量词只对它前边的一个内容起作用，如果有多个内容可以使用 () 括起来，常见量词如下：

- {n}：正好出现n次
- {m,}：出现m次及以上
- {m,n}：出现m-n次
- +：至少一个，相当于{1,}
- \*：0个或多个，相当于{0,}
- ?：0个或1个，相当于{0,1}

```

<script>
  var str = "abbc";

  reg = /(ab){3}/;
  console.log(reg.test(str));
  console.log("=====");
  reg = /b{3}/;
  console.log(reg.test(str));
  console.log("=====");
  reg = /ab{1,3}c/;
  console.log(reg.test(str));
  console.log("=====");
  reg = /ab{3,}c/;
  console.log(reg.test(str));
  console.log("=====");
  reg = /ab+c/;
  console.log(reg.test(str));
  console.log("=====");
  reg = /ab*c/;
  console.log(reg.test(str));
  console.log("=====");
  reg = /ab?c/;
  console.log(reg.test(str));
  console.log("=====");
</script>

```

### 3.6.6、正则高阶

如果我们要检查或者说判断是否以某个字符或者字符序列开头或者结尾就会使用^和\$。

- ^：表示开头，注意它在[^字符序列]表达的意思不一样
- \$：表示结尾

**需求描述：检查一个字符串中是否以a开头**

```

<script>
  var str = "abcabca";
  var reg = /^a/;
  console.log(reg.test(str));
</script>

```

**需求描述：检查一个字符串中是否以a结尾**

```
<script>
  var str = "abcbca";
  var reg = /a$/;
  console.log(reg.test(str));
</script>
```

那如果我们想要检查一个字符串中是否含有.和\就会使用转义字符

- \. : 表示.
- \\ : 表示\

注意：使用构造函数时，由于它的参数是一个字符串，而\是字符串中转义字符，如果要使用则需要使用\\来代替

- \w : 任意字母、数字、\_，相当于[A-z0-9\_]
- \W : 除了字母、数字、\_，相当于[^A-z0-9\_]
- \d : 任意的数字，相当于[0-9]
- \D : 除了任意的数字，相当于[^0-9]
- \s : 空格
- \S : 除了空格
- \b : 单词边界
- \B : 除了单词边界

**需求描述：**创建一个正则表达式，去除掉字符串中的前后的空格

```
<script>
  var str = "  hello child  "
  var reg = /^\\s*|\\s*$ /g;
  console.log(str);
  str = str.replace(reg, "");
  console.log(str);
</script>
```

hello child  
hello child

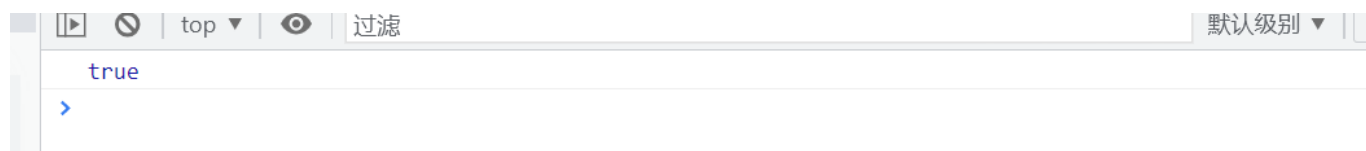
>

**需求描述：**创建一个正则表达式，检查一个字符串中是否含有单词child

```
<script>
  var str = "hello child"
  var reg = /\bchild\b/;
  console.log(reg.test(str));
</script>
```



```
<script>
  var str = "hello child"
  var reg = /child/;
  console.log(reg.test(str));
</script>
```



### 3.6.7、正则案例

#### 3.6.7.1、检查手机号

```
<script>
  var phoneStr = "15131494600";
  var phoneReg = /^1[3-9][0-9]{9}$/;
  console.log(phoneReg.test(phoneStr));
</script>
```

#### 3.6.7.2、检查邮箱号

```
<script>
  var emailStr = "abc.def@163.com";
  var emailReg = /\w{3,}(\.\w+)*@[A-z0-9]+(\.[A-z]{2,5}){1,2}$/;
  console.log(emailReg.test(emailStr));
</script>
```