1.JS免费课第二周第一天笔记

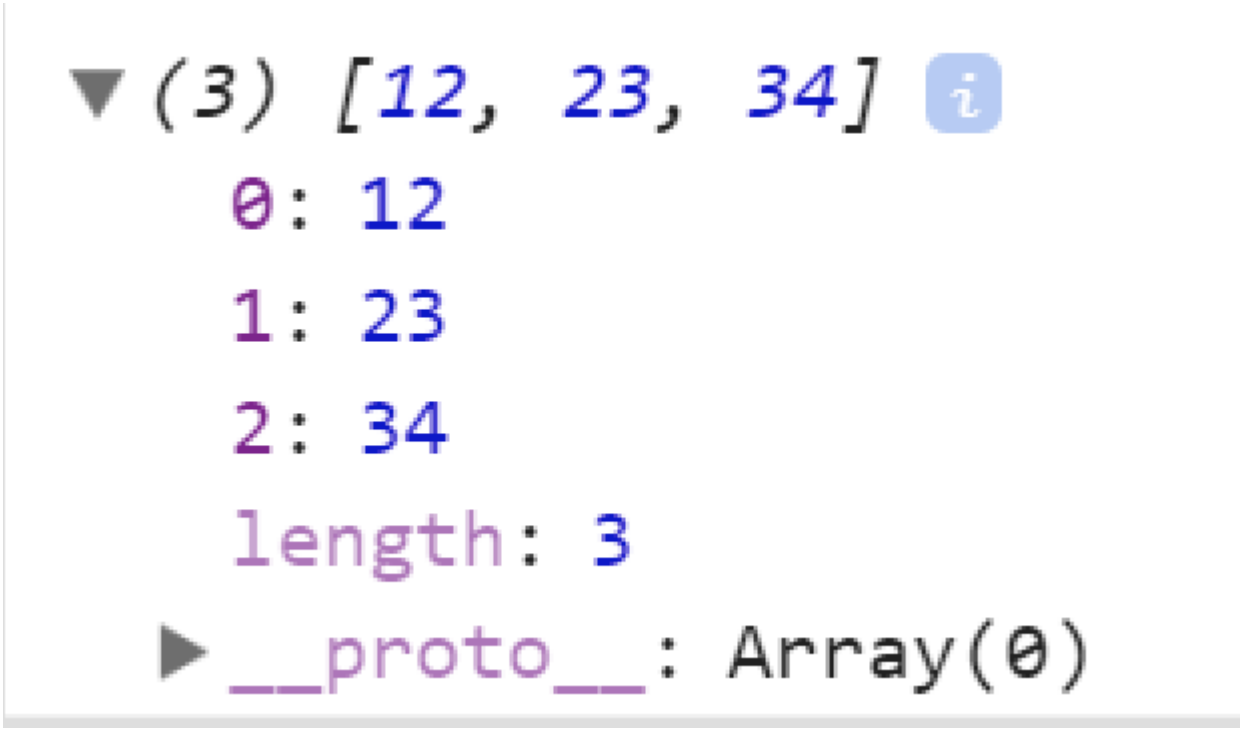
我的第一个笔记本

一、数组基础讲解

1.数组的基础结构

数组也是属于对象数据类型的 `typeof [] ->'object'`

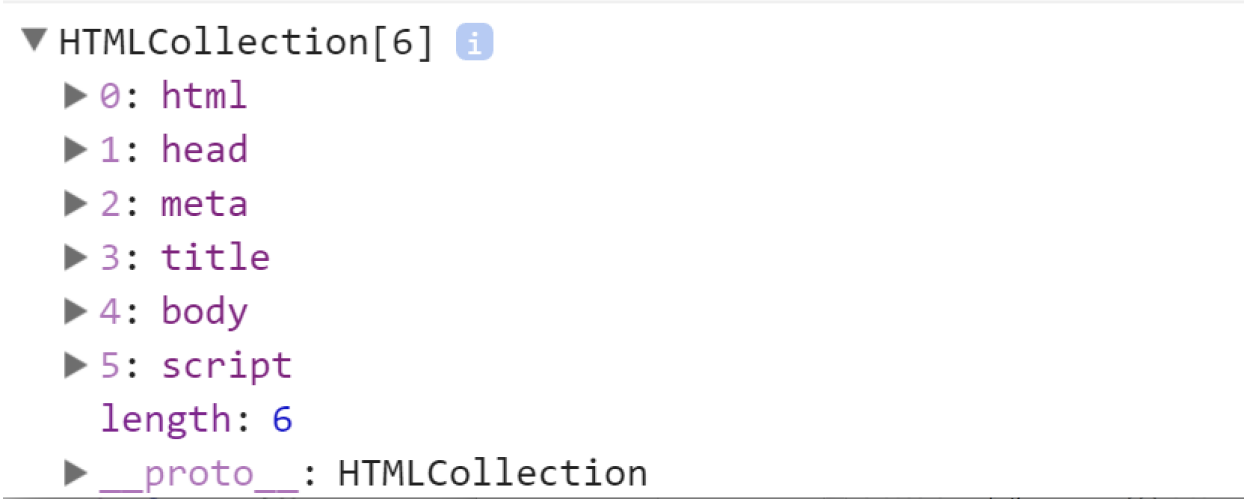
数组也有属性名，只不过属性名是数字，我们把数字属性名称之为它的索引：数组是以数字作为索引，索引从零开始，有一个length的属性，代表数组的长度；



类数组：类似于数组，但是不是数组
1、通过getElementsByTagName获取的元素集合是类数组
2、函数中的实参集合arguments也是类数组
…

元素集合getElementsByTagName

```
console.dir(document.getElementsByTagName('*'));
```



实参集合 arguments

```
function fn(){
    console.log(arguments);
}
fn(12,23,34)
```

```
▼ Arguments[3] i
    0: 12
    1: 23
    2: 34
  ▶ callee: fn()
    length: 3
  ▶ Symbol(Symbol.iterator): values()
  ▶ __proto__: Object
```

2.数组的遍历

> for 和 for in 循环都能遍历到数组私有的一些属性，而 for in 循环还可以把一些自定义的公共属性也能遍历到，for循环不可以；

```
var ary = [12,23,'珠峰']
Array.prototype.aa = 100;    //原型上写的属性和方法，是实例公有的属性和方法

//=>for循环操作
for (var i = 0; i < ary.length; i++) {
    console.log(ary[i]);
}

//=>for in循环操作
for (var key in ary) {
    console.log(ary[key]); //-> key:属性名(数组中的属性名是索引)
}
```

```
12
23
珠峰
12
23
珠峰
100
```

3.数组中的常用方法

`console.dir(Array.prototype)`：可以输出数组中有很多常用方法
1、方法的意义和作用
2、方法的形参
3、方法的返回值
4、通过此方法，原来的数组是否发生了改变

1、数组的增加 `push(末尾)`、`unshift(开头)`

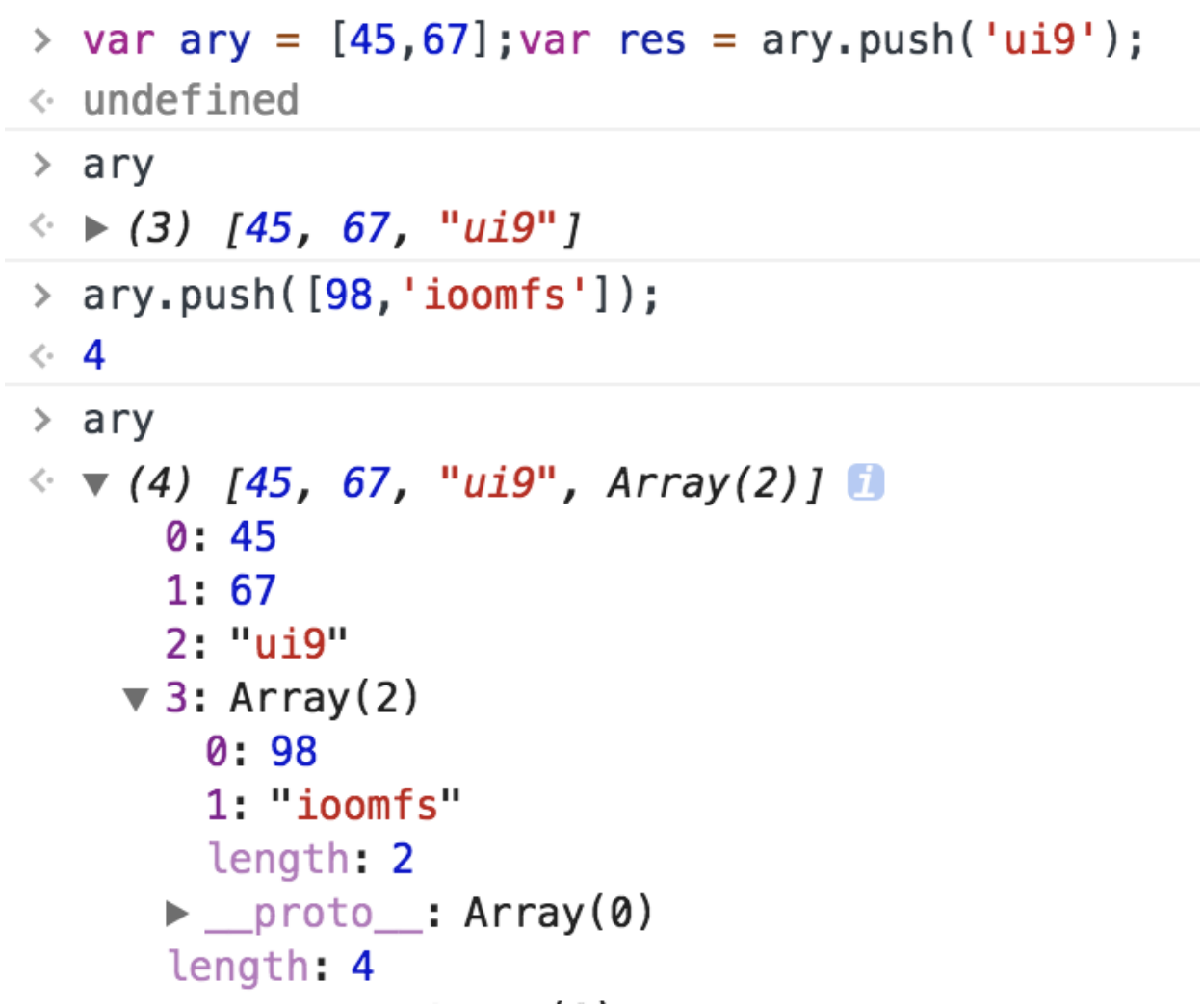
1、push：
含义：向数组的末尾追加新内容；
参数：一到多个，任何数据类型都可以，想要给数组末尾追加什么，直接传递到push方法中即可，传递多个用逗号隔开；
返回值：新增后数组的 长度；
是否改变：原有数组改变了；

2、unshift：
含义：向数组开头追加新内容；
参数：需要追加的内容，可以是多个任何数据类型的值；
返回值：新增后数组的长度；
是否改变：原有数组改变了；

3、把数组当做一个普通的对象，使用对象键值对的操作，利用索引给其设置新的属性；
`ary[ary.length]= xxx` 表示向数组的末尾追加了新的内容；

```
var ary = [12,23,45];
var res = ary.push(89,90);
console.log(ary);   //12,23,45,89,90
console.log(res);   //5

var ary = [12,23,45];
var res = ary.unshift(89,90);
console.log(ary);   //89,90,12,23,45
console.log(res);   //5
```



2、 数组的删除`pop(末尾)`、`shift(开头)`、`delete(指定项)`

1、pop:
`含义`： 删除数组最后一项;
`参数`： 无;
`返回值`： 被删除的那一项内容;
`是否改变`： 原有数组改变了;

2、shift:
`含义`： 删除数组第一项;
`参数`： 无;
`返回值`： 被删除的那一项内容;
`是否改变`： 原有数组改变了;

使用shift删除第一项之后，后面每一项的索引都要向前进一位（导致后面项的索引发生改变）

3、把数组当做普通的对象操作:
delete删除： `delete ary[索引]` 删除指定索引这一项 (当前项被删除后，原有数组其它项的索引不会改变；当前数组的length也不会改变);
`ary.length--`： 删除数组最后一项;

```
var ary = [12,23,45];
var res = ary.pop();
console.log(ary);   //12,23
console.log(res);   //45

var ary = [12,23,45];
var res = ary.shift();
console.log(ary);   //23,45
console.log(res);   //12
```

3、数组的修改 `splice`

splice： 数组中内置的方法， 可以实现数组的增加、修改、删除;
1) splice(n,m) 实现删除
含义： 从索引n开始删除m个， m不写是删除到数组的末尾;
返回值： 被删除的内容(以一个新数组保存);
是否改变： 原有数组改变了;

`splice(0)` 清空数组
`splice()` 一项都不删除,返回一个新的空数组
`splice(0,1)` 删除第一项
`splice(ary.length-1)` 删除最后一项

2) splice(n,m,x) 实现修改(替换)
含义： 在原有删除的基础上， 用x代替删除的内容;
返回值： 被删除的内容(以一个新数组保存);
是否改变： 原有数组改变了;

3) splice(n,0,x) 实现添加
含义： 在修改的基础上， 我们一项都不删除， 把x插入到索引n的前面;
返回值： 被删除的内容(以一个新数组保存);
是否改变： 原有数组改变了;

```
var ary = [12,23,45];
var res = ary.splice(1,2);
console.log(ary);   //[12]
console.log(res);   //[23,45]

var ary = [12,23,45,56,78,89];
var res = ary.splice(1,2,100);
console.log(ary);   //[12, 100, 56, 78, 89]
console.log(res);   //[23,45]

var ary = [12,23,45,56,78,89];
var res = ary.splice(1,0,100);
console.log(ary);   //[12, 100, 23, 45, 56, 78, 89]
console.log(res);   //[]
```

4、数组的查询 `slice`

slice:
含义： 数组的查询;
参数： slice(n,m) 从索引n开始找到索引为m处（不包含m）；
返回值： 把找到的部分以一个新数组返回;
是否改变： 原来的数组不变;

`slice(n)` 从索引n开始找到末尾;
`slice(0) / slice()` 数组克隆,克隆一份和原来数组一模一样的新数组;
`slice(-1)` slice支持负数索引，如果传递的索引为负数，浏览器解析的时候是按照 总长度+负数索引 来处理的;

```
var ary = [12,23,45,56,78,89];
var res = ary.slice(1,4);
console.log(ary);   //[12, 23, 45, 56, 78, 89]
console.log(res);   //[23, 45, 56]

var ary = [12,23,45,56,78,89];
var res = ary.slice(1,-4); //ary.length + (-4) => ary.slice(1,2)
console.log(ary);   //[12, 23, 45, 56, 78, 89]
console.log(res);   //[23]
```

5、数组的拼接 `concat`

concat:
含义： 将多个数组拼接在一起;
参数： 要拼接的内容（把内容放在原数组的后面），可以是一个数组，也可以是一些数据值;
返回值： 拼接后的新数组;
是否改变： 原来的数组不变;
concat() 什么都没有拼接，相当于把原有数组克隆一份一模一样的新数组出来

```
var ary = [12,23,45,56,78,89];
var res = ary.concat([900]);
console.log(ary);   //[12, 23, 45, 56, 78, 89]
console.log(res);   //12, 23, 45, 56, 78, 89,900]
```

6、 把数组转换为字符串 `toStirng`、`join`

1、toString:
含义：实现把数组转化为字符串, 转换后的字符串以逗号分隔每一项;
参数：无;
返回值：转换后的字符串;
是否改变：原来的数组不变;

2、join:
含义：把数组按照指定的分隔符转换为字符串，和字符串中的split相对应;
参数：指定的链接符，用引号引起来;
返回值：转换后的字符串;
是否改变：原来的数组不变;

eval 计算某个字符串，并执行

```
var ary = [12,23,45,56,78,89];
var res = ary.toString();
console.log(ary);   //[12, 23, 45, 56, 78, 89]
console.log(res);   //12,23,45,56,78,89

var ary = [12,23,45,56,78,89];
var res = ary.join('+');
console.log(ary);   //[12, 23, 45, 56, 78, 89]
console.log(res);   //12+23+45+56+78+89
console.log(eval(res));//303
//eval(string) 函数可计算某个字符串，并执行其中的的 JavaScript 代码。
```

```
> ary
◂ ▼ (5) [90, 8, 6, 4, Array(2)]
      0: 90
      1: 8
      2: 6
      3: 4
    ▶ 4: (2) [90, "zhufeng"]
      length: 5
    ▶ __proto__: Array(0)
> ary.toString()
◂ "90,8,6,4,90,zhufeng"
> ary.join(+)
✖ Uncaught SyntaxError: Unexpected token )        VM1093:1
> ary.join('-')
◂ "90-8-6-4-90,zhufeng"
> ary
◂ ▶ (5) [90, 8, 6, 4, Array(2)]
> eval(ary.join('-'))
✖ ▶Uncaught ReferenceError: zhufeng is not        VM1161:1
  defined
      at eval (eval at <anonymous> (:1:1),
  <anonymous>:1:1)
      at <anonymous>:1:1
> ary.pop()
◂ ▶ (2) [90, "zhufeng"]
> ary
◂ ▶ (4) [90, 8, 6, 4]
> eval(ary.join('-'))
◂ 72
```

需求：已知数组中的每一项都是数字，想实现数组求和，我们如何实现？

```
//1、循环实现
var total = null;
for(var i = 0;i < ary.length;i++){
    total += ary[i];
}

//2、利用join
//=> evel:把字符串变为JS表达式执行
var total = eval(ary.join('+'));
```

7、数组中的排列和排序 `reverse` 、 `sort`

1、reverse:
含义：把数组中的每一项倒过来排列;
参数：无;
返回值：排序后的数组;
是否改变：原有数组改变;

2、sort:
含义：实现数组的排序(从小到大);
参数：无或者回调函数;
返回值：排序后的数组;
是否改变：原有数组改变;

sort()不传递参数的情况下：可以给10以内的数字进行升序排列，但是超过10的就无法处理了（多位数只识别第一位）,此时可以用回调函数处理，如下

```
ary.sort(function(a,b){
    return a-b; //->升序,如果a>b，两者交换位置，大的值始终在最后
    return b-a; //->降序
});
```

```
var ary = [1,3,44,5,0,7,2];
var res = ary.sort();
console.log(ary);   //[0, 1, 2, 3, 44, 5, 7]
console.log(res);   //[0, 1, 2, 3, 44, 5, 7]
var res2 = ary.sort(function (a,b){
    return a-b;
})
console.log(res2);  //[0, 1, 2, 3, 5, 7, 44]
```

```
> ary
◂ ▶ (4) [90, 8, 6, 4]
> ary.reverse()
◂ ▶ (4) [4, 6, 8, 90]
> ary
◂ ▶ (4) [4, 6, 8, 90]
> ary.sort()
◂ ▶ (4) [4, 6, 8, 90]
> ary.push(5,23)
◂ 6
> ary
◂ ▶ (6) [4, 6, 8, 90, 5, 23]
> ary.sort()
◂ ▶ (6) [23, 4, 5, 6, 8, 90]
> ary.sort(function(a,b){return a-b;})
◂ ▶ (6) [4, 5, 6, 8, 23, 90]
> ary.sort(function(a,b){return b-a;})
◂ ▶ (6) [90, 23, 8, 6, 5, 4]
> ary
◂ ▶ (6) [90, 23, 8, 6, 5, 4]
```

8、验证数组中是否包含某一项

1、indexOf / lastIndexOf

`含义`： 获取当前项在数组中第一次出现/最火一次位置的索引；

`参数`： 要检测的值

`返回值`： 被检测值在数组中的索引

`原有数组是否改变`： 否

`数组中`的这两个方法在IE6~8下不兼容

`字符串中`的这两个方法兼容所有的浏览器

`如果当前数组中并没有这一项，返回的索引是-1；我们根据这一点可以验证数组中是否包含这一项；`

```
> ary
<· ▶ (9) [90, 23, 8, 6, 5, 4, 4, 23, 8]
> ary.indexOf(7)
<· -1
> ary.indexOf(23)
<· 1
> ary.lastIndexOf(23)
<· 7
```

9、遍历数组的一些方法

> 以下方法在IE6~8下都不兼容
> 1、forEach(回调函数)
> 含义：遍历数组中的每一项;
> forEach() 方法用于调用数组的每个元素，并将元素传递给回调函数。

```
ary.forEach(function(value,index){
    //=>数组中有多少项，当前回调函数执行多少次,每一次传递进来的value就是当前遍历数组这一项的值，index就是遍历这一项的索引
});
```

```
> var ary = [23,5,4,9,1,0,7];
  var res = ary.forEach(function(value, index, ary){
      return ary[index] = value*10;
  });
  console.log(ary);
  console.log(res);
▶ (7) [230, 50, 40, 90, 10, 0, 70]                    VM61:5
undefined                                             VM61:6
<· undefined
> var ary = [23,5,4,9,1,0,7];
  var res = ary.map(function(value, index, ary){
      return ary[index] = value*10;
  });
  console.log(ary);
  console.log(res);
▶ (7) [230, 50, 40, 90, 10, 0, 70]                    VM75:5
▶ (7) [230, 50, 40, 90, 10, 0, 70]                    VM75:6
```

```
> sum = 0
< 0
> ary
< ▶ (9) [90, 23, 8, 6, 5, 4, 4, 23, 8]
> ary.forEach(function(value,index){sum +=
  value;console.log(sum);})
  90                                          VM2104:1
  113                                         VM2104:1
  121                                         VM2104:1
  127                                         VM2104:1
  132                                         VM2104:1
  136                                         VM2104:1
  140                                         VM2104:1
  163                                         VM2104:1
  171                                         VM2104:1
< undefined
> sum = 0;
< 0
> var res = ary.forEach(function(value,index){sum +=
  value;})
< undefined
> res
< undefined
```

2、map

含义：遍历数组中的每一项，在forEach的基础上，可以修改每一项的值；

map() 方法返回一个新数组，数组中的元素为原始数组元素调用函数处理后的值。

map() 方法按照原始数组元素顺序依次处理元素。

注意： map() 不会对空数组进行检测。

注意： map() 不会改变原始数组。

```
ary.map(function(value,index){
    //=>数组中有多少项，当前回调函数执行多少次；每一次传递进来的value就是当前遍历数组这一项的值，index就是遍历这一项的索引
    return xxx; //=>return后面返回的结果就是把当前遍历的这一项修改为xxx
});
```

filter
find
reduce
every
...

4.实战案例：数组去重

1、方案一：双循环遍历法

遍历数组中的每一项，拿每一项和它后面的项依次比较，如果相同了，则把相同的这一项在原来数组中删除即可

`数组塌陷问题`：我们使用splice删除数组中的某一项后,删除这一项后面的每一项索引都要向前进一位(在原有索引上减一)，此时如果我们j++，循环操作的值累加了，我们通过最新j获取的元素不是紧挨删除这一项的元素，而是跳过一项获取的元素；

```
var ary = [1, 2, 2, 2, 3, 2, 1, 2, 3, 2, 2, 3, 4, 3, 2, 2, 3];

for (var i = 0; i < ary.length - 1; i++) {
    var cur = ary[i];
    for (var j = i + 1; j < ary.length; j++) {
        if (cur === ary[j]) {
            ary.splice(j, 1);
            j--;//=>先让j--，然后再j++，相当于没加没减，此时j还是原有索引，再获取的时候就是删除这一项后面紧挨着的这一项，解决数组塌陷问题
        }
    }
}
```

2、方案二：indexOf处理

利用indexOf来验证当前数组中是否包含某一项，包含把当前项删除掉（不兼容IE6~8）
`ary.splice(i, 1)删除` 使用splice会导致后面的索引向前进一位,如果后面有很多项,消耗的性能很大;
`解决思路:` 我们把最后一项拿过来替换当前要删除的这一项,然后再把最后一项删除;

```
var ary = [1, 2, 2, 2, 3, 2, 1, 2, 3, 2, 2, 3, 4, 3, 2, 2, 3];
for (var i = 0; i < ary.length; i++) {
    var cur = ary[i];//=>当前项
    var curNextAry = ary.slice(i + 1);//=>把当前项后面的那些值以一个新数组返回，我们需要比较的就是后面的这些项对应的新数组
    if (curNextAry.indexOf(cur) > -1) {
        //=>后面项组成的数组中包含当前这一项（当前这一项是重复的），我们把当前这一项删除掉即可
       // ary.splice(i, 1);  //=> 消耗性能
        ary[i] = ary[ary.length - 1];
        ary.length--;
        i--;
    }
}
console.log(ary);
```

3、方案三：对象键值对处理

> 遍历数组中的每一项，把每一项作为新对象的属性名和属性值存储起来，例如：当前项1，对象中存储的 {1:1}
>
> 在每一次向对象中存储之前，首先看一下原有对象中是否包含了这个属性（`typeof obj[xxx]==='undefined'` 说明当前对象中没有xxx这个属性），如果已经存在这个属性说明数组中的当前项是重复的（1-在原有数组中删除这一项 2-不再向对象中存储这个结果），如果不存在（把当前项作为对象的属性名和属性值存储进去即可）

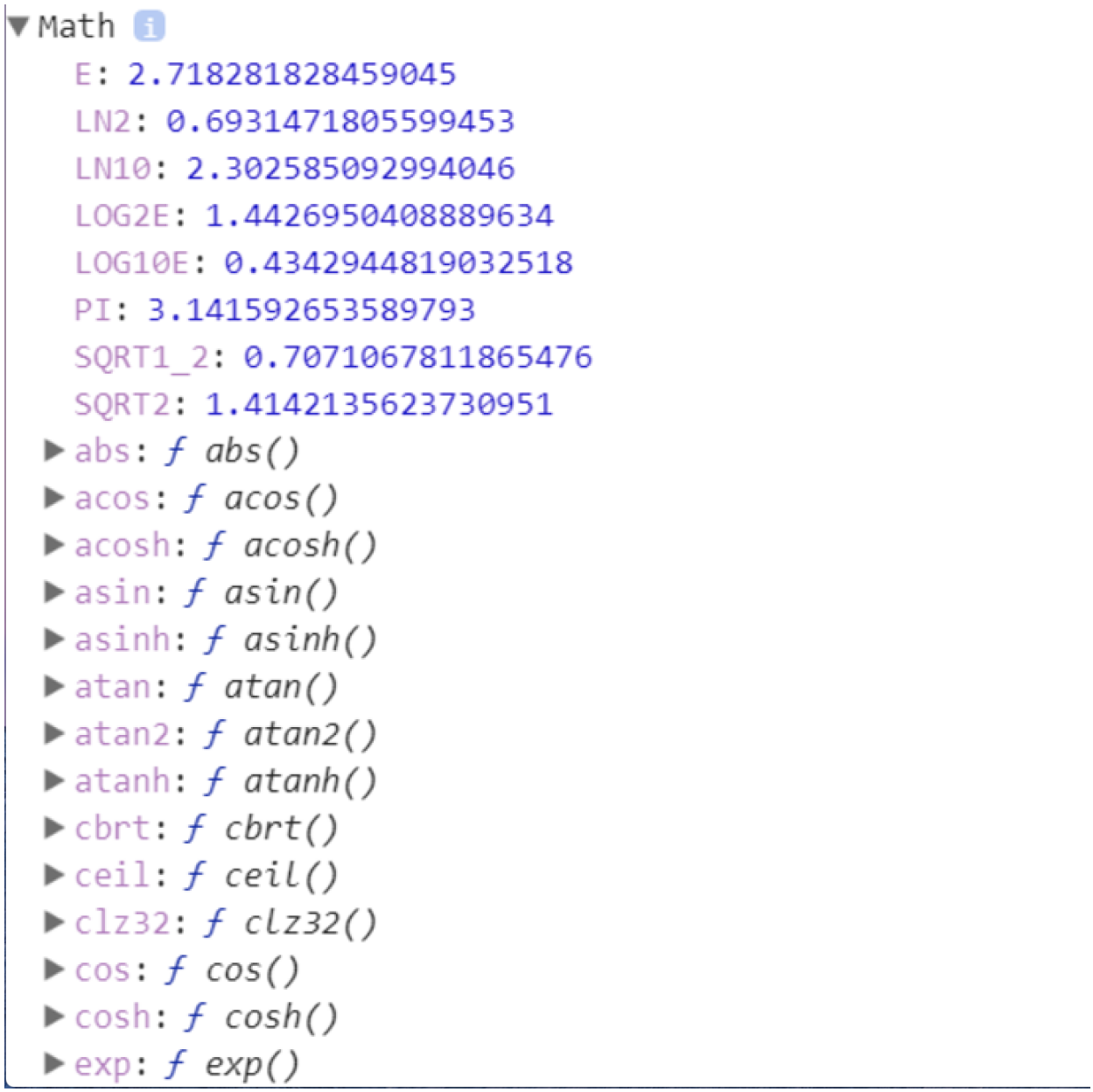
```
var ary = [1, 2, 2, 2, 3, 2, 1, 2, 3, 2, 2, 3, 4, 3, 2, 2, 3];
var obj = {};
for (var i = 0; i < ary.length; i++) {
    var cur = ary[i];
    if (typeof obj[cur] !== 'undefined') {
        ary[i] = ary[ary.length - 1];
        ary.length--;
        i--;
        continue;
    }
    obj[cur] = cur;
}
```

4、在数组原型上扩展方法 - myUnique

```
Array.prototype.myUnique = function myUnique() {
    var obj = {};
    for (var i = 0; i < this.length; i++) {
        var item = this[i];
        if (typeof obj[item] !== 'undefined') {
            this[i] = this[this.length - 1];
            this.length--;
            i--;
            continue;
        }
        obj[item] = item;
    }
    obj = null;
    return this;
};
var ary = [12, 23, 13, 23, 12, 12, 12, 13, 23, 2, 31, 14];
console.log(ary.myUnique().sort(function (a, b) {
    return a - b;
}));
```

二、Math中常用的方法

> 数学函数：它属于对象数据类型 `typeof Math ->'object'`
> Math对象中提供了很多操作数字的方法
> `console.dir(Math)`


Math
E: 2.718281828459045
LN2: 0.6931471805599453
LN10: 2.302585092994046
LOG2E: 1.4426950408889634
LOG10E: 0.4342944819032518
PI: 3.141592653589793
SQRT1_2: 0.7071067811865476
SQRT2: 1.4142135623730951
abs: ƒ abs()
acos: ƒ acos()
acosh: ƒ acosh()
asin: ƒ asin()
asinh: ƒ asinh()
atan: ƒ atan()
atan2: ƒ atan2()
atanh: ƒ atanh()
cbrt: ƒ cbrt()
ceil: ƒ ceil()
clz32: ƒ clz32()
cos: ƒ cos()
cosh: ƒ cosh()
exp: ƒ exp()

```
▶ expm1: ƒ expm1()
▶ floor: ƒ floor()
▶ fround: ƒ fround()
▶ hypot: ƒ hypot()
▶ imul: ƒ imul()
▶ log: ƒ log()
▶ log1p: ƒ log1p()
▶ log2: ƒ log2()
▶ log10: ƒ log10()
▶ max: ƒ max()
▶ min: ƒ min()
▶ pow: ƒ pow()
▶ random: ƒ random()
▶ round: ƒ round()
▶ sign: ƒ sign()
▶ sin: ƒ sin()
▶ sinh: ƒ sinh()
▶ sqrt: ƒ sqrt()
▶ tan: ƒ tan()
▶ tanh: ƒ tanh()
▶ trunc: ƒ trunc()
  Symbol(Symbol.toStringTag): "
▶ __proto__: Object
```

1.Math.abs 取绝对值

```
Math.abs(12) ->12
Math.abs(-12) ->12
```

2.Math.ceil 向上取整

```
Math.ceil(12) ->12
Math.ceil(12.1) ->13
Math.ceil(12.9) ->13
Math.ceil(-12.1) ->-12
Math.ceil(-12.9) ->-12
```

3. Math.floor 向下取整

```
Math.floor(12) ->12
Math.floor(12.1) ->12
Math.floor(12.9) ->12
Math.floor(-12.1) ->-13
Math.floor(-12.9) ->-13
```

4.Math.round 四舍五入

```
Math.round(12.3) ->12
Math.round(12.5) ->13 正数中5包含在向上
Math.round(-12.3) ->-12
Math.round(-12.5) ->-12 负数中5包含在向下
Math.round(-12.51) ->-13
```

5.Math.random 获取(0,1)之间的随机小数

```
for(var i=0;i<100;i++){
    console.log(Math.random());
}
//=>需求：获取[0,10]之间的随机整数
Math.round(Math.random()*10)

//=>需求：获取[3,15]之间的随机整数
Math.round(Math.random()*12+3)

//=>获取[n,m]之间的随机整数
Math.round(Math.random()*(m-n)+n)
```

6.Math.max 获取一组值中的最大值

```
Math.max(12,23,14,24,34,25,13,15,16,27,13,12); ->34
```

7. Math.min 获取一组值中的最小值

```
Math.min(12,23,14,24,34,25,13,15,16,27,13,12); ->12
```

8.Math.PI 获取圆周率（π）

```
Math.PI ->3.141592653589793
```

9.Math.pow 获取一个值的多少次幂

```
Math.pow(10,2) ->100
```

10.Math.sqrt 开平方

```
Math.sqrt(100) ->10
```