#声明数组

数组是多个变量值的集合，数组是`Array`对象的实例，所以可以像对象一样调用方法。

#创建数组

使用对象方式创建数组

```
console.log(new Array(1, '后盾人', 'hdcms')); //[1, "后盾人", "hdcms"]

```

使用字面量创建是推荐的简单作法

```
const array = ["hdcms", "houdunren"];

```

多维数组定义

```
const array = [["hdcms"], ["houdunren"]];
console.log(array[1][0]);

```

数组是引用类型可以使用`const`声明并修改它的值

```
const array = ["hdcms", "houdunren"];
array.push("houdunwang");
console.log(array);

```

使用原型的`length`属性可以获取数组元素数量

```
let hd = ["后盾人", "hdcms"];
console.log(hd.length); //2

```

数组可以设置任何值，下面是使用索引添加数组

```
let hd = ["后盾人"];
```

```
hd[1] = "hdcms";

```

下面直接设置 3 号数组，会将 1/2 索引的数组定义为空值

```
let hd = ["后盾人"];
hd[3] = "hdcms";
console.log(hd.length); //4

```

声明多个空元素的数组

```
let hd = new Array(3);
console.log(hd.length);
console.log(hd);

```

#Array.of

使用`Array.of` 与 `new Array` 不同是设置一个参数时不会创建空元素数组

```
let hd = Array.of(3);
console.log(hd); //[3]

hd = Array.of(1, 2, 3);
console.log(hd); //[1, 2, 3]

```

#类型检测

检测变量是否为数组类型

```
console.log(Array.isArray([1, "后盾人", "hdcms"])); //true
console.log(Array.isArray(9)); //false

```

#类型转换

可以将数组转换为字符串也可以将其他类型转换为数组。

#字符串

大部分数据类型都可以使用`.toString()`函数转换为字符串。

```
console.log(([1, 2, 3]).()); // 1,2,3

```

也可以使用函数`String`转换为字符串。

```
console.log(String([1, 2, 3]));

```

或使用`join`连接为字符串

```
console.log([1, 2, 3].join("-"));//1-2-3

```

#Array.from

使用`Array.from`可将类数组转换为数组，类数组指包含`length`属性或可迭代的对象。

- 第一个参数为要转换的数据，第二个参数为类似于

map 函数的回调方法

```
let str = '后盾人';
console.log(Array.from(str)); //["后", "盾", "人"]

```

为对象设置`length`属性后也可以转换为数组，但要下标为数值或数值字符串

```
let user = {
  0: '后盾人',
  '1': 18,
  length: 2
};
console.log(Array.from(user)); //["后盾人", 18]

```

DOM 元素转换为数组后来使用数组函数，第二个参数类似于`map`函数的方法，可对数组元素执行函数处理。

```
<body>
    <button message="后盾人">button</button>
    <button message="hdcms">button</button>
</body>

<script>
    let btns = document.querySelectorAll('button');
    console.log(btns); //包含length属性
    Array.from(btns, (item) => {
        item.style.background = 'red';
    });
</script>

```

#展开语法

使用展开语法将`NodeList`转换为数组操作

```
<style>
    .hide {
      display: none;
    }
</style>

<body>
  <div>hdcms</div>
  <div>houdunren</div>
</body>

<script>
  let divs = document.querySelectorAll("div");
  [...divs].map(function(div) {
    div.addEventListener("click", function() {
      this.classList.toggle("hide");
    });
```

```
  });
</script>

```

#展开语法

#数组合并

使用展开语法来合并数组相比 `concat` 要更简单，使用`...` 可将数组展开为多个值。

```
let a = [1, 2, 3];
let b = ['a', '后盾人', ...a];
console.log(b); //["a", "后盾人", 1, 2, 3]

```

#函数参数

使用展示语法可以替代 `arguments` 来接收任意数量的参数

```
function hd(...args) {
  console.log(args);
}
hd(1, 2, 3, "后盾人"); //[1, 2, 3, "后盾人"]

```

也可以用于接收部分参数

```
function hd(site, ...args) {
  console.log(site, args); //后盾人 (3) [1, 2, 3]
}
hd("后盾人", 1, 2, 3);

```

#节点转换

可以将 DOM 节点转为数组，下面例子不可以使用 filter 因为是节点列表

```
<body>
```

```
    <button message="后盾人">button</button>
    <button message="hdcms">button</button>
</body>

<script>
    let btns = document.querySelectorAll('button');
    btns.map((item) => {
        console.log(item); //TypeError: btns.filter is not a function
    })
</script>

```

使用展开语法后就可以使用数据方法

```
<body>
  <div>hdcms</div>
  <div>houdunren</div>
</body>

<script>
  let divs = document.querySelectorAll("div");
  [...divs].map(function(div) {
    div.addEventListener("click", function() {
      this.classList.toggle("hide");
    });
  });
</script>

```

学习后面章节后也可以使用原型处理

```
<body>
    <button message="后盾人">button</button>
    <button message="hdcms">button</button>
</body>

<script>
    let btns = document.querySelectorAll('button');
    Array.prototype.map.call(btns, (item) => {
```

```
        item.style.background = 'red';
    });
</script>

```

#解构赋值

解构是一种更简洁的赋值特性，可以理解为分解一个数据的结构

- 建设使用

var/let/const 声明

#基本使用

下面是基本使用语法

```
//数组使用
let [name, url] = ['后盾人', 'houdunren.com'];
console.log(name);

```

解构赋值数组

```
function hd() {
 return ['houdunren', 'hdcms'];
}
let [a, b] = hd();
console.log(a); //houdunren

```

剩余解构指用一个变量来接收剩余参数

```
let [a, ...b] = ['后盾人', 'houdunren', 'hdcms'];
console.log(b);

```

如果变量已经初始化过，就要使用`()`定义赋值表达式，严格模式会报错所以不建议使用。

```
let web = "后盾人";
[web, url] = ["hdcms.com", "houdunren.com"];
```

```
console.log(web);

```

字符串解构

```
"use strict";
const [...a] = "houdunren.com";
console.log(a); //Array(13)

```

#严格模式

非严格模式可以不使用声明指令，严格模式下必须使用声明。所以建议使用 let 等声明。

```
"use strict";

[web, url] = ["hdcms.com", "houdunren.com"];
console.log(web);

```

#简洁定义

只赋值部分变量

```
let [,url]=['后盾人','houdunren.com'];
console.log(url);//houdunren.com

```

使用展开语法获取多个值

```
let [name, ...arr] = ['后盾人', 'hdcms', 'houdunren.com'];
console.log(name, arr); //后盾人 (2) ["hdcms", "houdunren.com"]

```

#默认值

为变量设置默认值

```

```

```
let [name, site = 'hdcms'] = ['后盾人'];
console.log(site); //hdcms

```

#函数参数

数组参数的使用

```
function hd([a, b]) {
        console.log(a, b);
}
hd(['后盾人', 'hdcms']);

```

#管理元素

#基本使用

使用从 0 开始的索引来改变数组

```
let arr = [1, "后盾人", "hdcms"];
arr[1] = '后盾人教程';
console.log(arr); //[1, "后盾人教程", "hdcms"]

```

向数组追回元素

```
let arr = [1, "后盾人", "hdcms"];
arr[arr.length] = 'houdunren.com';
console.log(arr); //[1, "后盾人", "hdcms", "houdunren.com"]

```

#扩展语法

使用展示语法批量添加元素

```
let arr = ["后盾人", "hdcms"];
let hd = ["houdunren"];
```

```
hd.push(...arr);
console.log(hd); //["houdunren", "后盾人", "hdcms"]

```

#push

压入元素，直接改变元数组，返回值为数组元素数量

```
let arr = ["后盾人", "hdcms"];
console.log(arr.push('向军大叔', 'houdunren')); //4
console.log(arr); //["后盾人", "hdcms", "向军大叔", "houdunren"]

```

根据区间创建新数组

```
function rangeArray(begin, end) {
  const array = [];
  for (let i = begin; i <= end; i++) {
    array.push(i);
  }
  return array;
}
console.log(rangeArray(1, 6));

```

#pop

从末尾弹出元素，直接改变元数组，返回值为弹出的元素

```
let arr = ["后盾人", "hdcms"];
console.log(arr.pop()); //hdcms
console.log(arr); //["后盾人"]

```

#shift

从数组前面取出一个元素

```
let arr = ["后盾人", "hdcms"];
console.log(arr.shift()); //后盾人
console.log(arr); //["hdcms"]

```

#unshift

从数组前面添加元素

```
let arr = ["后盾人", "hdcms"];
console.log(arr.unshift('向军大叔', 'houdunren')); //4
console.log(arr); //["向军大叔", "houdunren", "后盾人", "hdcms"]

```

#fill

使用`fill` 填充数组元素

```
console.dir(Array(4).fill("后盾人")); //["后盾人", "后盾人", "后盾人", "后盾人"]

```

指定填充位置

```
console.log([1, 2, 3, 4].fill("后盾人", 1, 2)); //[1, "后盾人", 3, 4]

```

#slice

使用 `slice` 方法从数组中截取部分元素组合成新数组（并不会改变原数组），不传第二个参数时截取到数组的最后元素。

```
let arr = [0, 1, 2, 3, 4, 5, 6];
console.log(arr.slice(1, 3)); // [1,2]

```

不设置参数是为获取所有元素

```
let arr = [0, 1, 2, 3, 4, 5, 6];
console.log(arr.slice()); //[0, 1, 2, 3, 4, 5, 6]

```

#splice

使用 `splice` 方法可以添加、删除、替换数组中的元素，会对原数组进行改变，返回值为删除的元素。删除数组元素第一个参数为从哪开始删除，第二个参数为删除的数量。

```
let arr = [0, 1, 2, 3, 4, 5, 6];
console.log(arr.splice(1, 3)); //返回删除的元素 [1, 2, 3]
console.log(arr); //删除数据后的原数组 [0, 4, 5, 6]

```

通过修改`length`删除最后一个元素

```
let arr = ["后盾人", "hdcms"];
arr.length = arr.length - 1;
console.log(arr);

```

通过指定第三个参数来设置在删除位置添加的元素

```
let arr = [0, 1, 2, 3, 4, 5, 6];
console.log(arr.splice(1, 3, 'hdcms', '后盾人')); //[1, 2, 3]
console.log(arr); //[0, "hdcms", "后盾人", 4, 5, 6]

```

向末尾添加元素

```
let arr = [0, 1, 2, 3, 4, 5, 6];
console.log(arr.splice(arr.length, 0, 'hdcms', '后盾人')); //[]
console.log(arr); // [0, 1, 2, 3, 4, 5, 6, "hdcms", "后盾人"]

```

向数组前添加元素

```
let arr = [0, 1, 2, 3, 4, 5, 6];
```

```
console.log(arr.splice(0, 0, 'hdcms', '后盾人')); //[]
console.log(arr); //["hdcms", "后盾人", 0, 1, 2, 3, 4, 5, 6]

```

数组元素位置调整函数

```
function move(array, before, to) {
  if (before < 0 || to >= array.length) {
    console.error("指定位置错误");
    return;
  }
  const newArray = [...array];
  const elem = newArray.splice(before, 1);
  newArray.splice(to, 0, ...elem);
  return newArray;
}
const array = [1, 2, 3, 4];
console.table(move(array, 0, 3));

```

#清空数组

将数组值修改为`[]`可以清空数组，如果有多个引用时数组在内存中存在被其他变量引用。

```
let user = [{ name: "hdcms" }, { name: "后盾人" }];
let cms = user;
user = [];
console.log(user);
console.log(cms);

```

将数组`length`设置为 0 也可以清空数组

```
let user = [{ name: "hdcms" }, { name: "后盾人" }];
user.length = 0;
console.log(user);

```

使用`splice`方法删除所有数组元素

```
let user = [{ name: "hdcms" }, { name: "后盾人" }];
user.splice(0, user.length);
console.log(user);

```

使用`pop/shift`删除所有元素，来清空数组

```
let user = [{ name: "hdcms" }, { name: "后盾人" }];
while (user.pop()) {}
console.log(user);

```

#合并拆分

#join

使用`join`连接成字符串

```
let arr = [1, "后盾人", "hdcms"];
console.log(arr.join('-')); //1-后盾人-hdcms 使用join可以指定转换的连接方式

```

#split

`split` 方法用于将字符串分割成数组，类似`join`方法的反函数。

```
let price = "99,78,68";
console.log(price.split(",")); //["99", "78", "68"]

```

#concat

`concat`方法用于连接两个或多个数组，元素是值类型的是复制操作，如果是引用类型还是指向同一对象

```
let array = ["hdcms", "houdunren"];
let hd = [1, 2];
```

```
let cms = [3, 4];
console.log(array.concat(hd, cms)); //["hdcms", "houdunren", 1, 2, 3, 4]

```

也可以使用扩展语法实现连接

```
console.log([...array, ...hd, ...cms]);

```

#copyWithin

使用 `copyWithin` 从数组中复制一部分到同数组中的另外位置。
语法说明

```
array.copyWithin(target, start, end)

```

参数说明

参数	描述
target	必需。复制到指定目标索引位置。
start	可选。元素复制的起始位置。
end	可选。停止复制的索引位置 (默认为 array .length)。如果为负值，表示倒数。

```
const arr = [1, 2, 3, 4];
console.log(arr.copyWithin(2, 0, 2)); //[1, 2, 1, 2]

```

#查找元素

数组包含多种查找的函数，需要把这些函数掌握清楚，然后根据不同场景选择合适的函数。

#indexOf

使用 `indexOf` 从前向后查找元素出现的位置，如果找不到返回 `-1`。

```

```

```
let arr = [7, 3, 2, 8, 2, 6];
console.log(arr.indexOf(2)); // 2 从前面查找2出现的位置

```

如下面代码一下，使用 `indexOf` 查找字符串将找不到，因为`indexOf` 类似于`===`是严格类型约束。

```
let arr = [7, 3, 2, '8', 2, 6];
console.log(arr.indexOf(8)); // -1

```

第二个参数用于指定查找开始位置

```
let arr = [7, 3, 2, 8, 2, 6];
//从第二个元素开始向后查找
console.log(arr.indexOf(2, 3)); //4

```

#lastIndexOf

使用 `lastIndexOf` 从后向前查找元素出现的位置，如果找不到返回 `-1`。

```
let arr = [7, 3, 2, 8, 2, 6];
console.log(arr.lastIndexOf(2)); // 4 从后查找2出现的位置

```

第二个参数用于指定查找开始位置

```
let arr = [7, 3, 2, 8, 2, 6];
//从第五个元素向前查找
console.log(arr.lastIndexOf(2, 5));

//从最后一个字符向前查找
console.log(arr.lastIndexOf(2, -2));

```

#includes

使用 `includes` 查找字符串返回值是布尔类型更方便判断

```

```

```
let arr = [7, 3, 2, 6];
console.log(arr.includes(6)); //true

```

我们来实现一个自已经的`includes`函数，来加深对`includes`方法的了解

```

function includes(array, item) {
  for (const value of array)
    if (item === value) return true;
  return false;
}

console.log(includes([1, 2, 3, 4], 3)); //true

```

#find

find 方法找到后会把值返回出来

- 如果找不到返回值为

undefined

返回第一次找到的值，不继续查找

```

let arr = ["hdcms", "houdunren", "hdcms"];

let find = arr.find(function(item) {
  return item == "hdcms";
});

console.log(find); //hdcms

```

使用`includes`等不能查找引用类型，因为它们的内存地址是不相等的

```

const user = [{ name: "李四" }, { name: "张三" }, { name: "后盾人" }];
const find = user.includes({ name: "后盾人" });
console.log(find);

```

`find` 可以方便的查找引用类型

```
const user = [{ name: "李四" }, { name: "张三" }, { name: "后盾人" }];
const find = user.find(user => (user.name = "后盾人"));
console.log(find);

```

#findIndex

`findIndex` 与 `find` 的区别是返回索引值，参数也是：当前值，索引，操作数组。

- 查找不到时返回

–1

```
let arr = [7, 3, 2, '8', 2, 6];

console.log(arr.findIndex(function (v) {
 return v == 8;
})); //3

```

#find 原理

下面使用自定义函数

```
let arr = [1, 2, 3, 4, 5];
function find(array, callback) {
  for (const value of array) {
    if (callback(value) === true) return value;
  }
  return undefined;
}
let res = find(arr, function(item) {
  return item == 23;
});
console.log(res);

```

下面添加原型方法实现

```
Array.prototype.findValue = function(callback) {
  for (const value of this) {
    if (callback(value) === true) return value;
  }
  return undefined;
};

let re = arr.findValue(function(item) {
  return item == 2;
});
console.log(re);

```

#数组排序

#reverse

反转数组顺序

```
let arr = [1, 4, 2, 9];
console.log(arr.reverse()); //[9, 2, 4, 1]

```

#sort

`sort`每次使用两个值进行比较 `Array.sort((a,b)=>a-b`

- 返回负数 a 排在 b 前面，从小到大
- 返回正数 b 排在 a 前面
- 返回 0 时不动

默认从小于大排序数组元素

```
let arr = [1, 4, 2, 9];
console.log(arr.sort()); //[1, 2, 4, 9]

```

使用排序函数从大到小排序，参数一与参数二比较，返回正数为降序负数为升序

```
let arr = [1, 4, 2, 9];

console.log(arr.sort(function (v1, v2) {
 return v2 - v1;
})); //[9, 4, 2, 1]

```

下面是按课程点击数由高到低排序

```
let lessons = [
  { title: "媒体查询响应式布局", click: 78 },
  { title: "FLEX 弹性盒模型", click: 12 },
  { title: "MYSQL多表查询随意操作", click: 99 }
];

let sortLessons = lessons.sort((v1, v2) => v2.click - v1.click);
console.log(sortLessons);

```

#排序原理

```
let arr = [1, 5, 3, 9, 7];
function sort(array, callback) {
  for (const n in array) {
    for (const m in array) {
      if (callback(array[n], array[m]) < 0) {
        let temp = array[n];
        array[n] = array[m];
        array[m] = temp;
      }
    }
  }
  return array;
}
arr = sort(arr, function(a, b) {
  return a - b;
```

```
});
console.table(arr);

```

#循环遍历

#for

根据数组长度结合`for`循环来遍历数组

```
let lessons = [
 {title: '媒体查询响应式布局',category: 'css'},
  {title: 'FLEX 弹性盒模型',category: 'css'},
 {title: 'MYSQL多表查询随意操作',category: 'mysql'}
];

for (let i = 0; i < lessons.length; i++) {
  lessons[i] = `后盾人: ${lessons[i].title}`;
}
console.log(lessons);

```

#forEach

`forEach`使函数作用在每个数组元素上，但是没有返回值。

下面例子是截取标签的五个字符。

```
let lessons = [
 {title: '媒体查询响应式布局',category: 'css'},
  {title: 'FLEX 弹性盒模型',category: 'css'},
 {title: 'MYSQL多表查询随意操作',category: 'mysql'}
];

lessons.forEach((item, index, array) => {
    item.title = item.title.substr(0, 5);
});
console.log(lessons);

```

#for/in

遍历时的 key 值为数组的索引

```
let lessons = [
 {title: '媒体查询响应式布局',category: 'css'},
  {title: 'FLEX 弹性盒模型',category: 'css'},
 {title: 'MYSQL多表查询随意操作',category: 'mysql'}
];

for (const key in lessons) {
    console.log(`标题: ${lessons[key].title}`);
}

```

#for/of

与 `for/in` 不同的是 `for/of` 每次循环取其中的值而不是索引。

```
let lessons = [
 {title: '媒体查询响应式布局',category: 'css'},
  {title: 'FLEX 弹性盒模型',category: 'css'},
 {title: 'MYSQL多表查询随意操作',category: 'mysql'}
];

for (const item of lessons) {
  console.log(`
    标题: ${item.title}
    栏目: ${item.category == "css" ? "前端" : "数据库"}
  `);
}

```

使用数组的迭代对象遍历获取索引与值（有关迭代器知识后面章节会讲到）

```
const hd = ['houdunren', 'hdcms'];
const iterator = hd.entries();
console.log(iterator.next()); //value:{0:0,1:'houdunren'}
console.log(iterator.next()); //value:{0:1,1:'hdcms'}
```

```
5
```

这样就可以使用解构特性与 `for/of` 遍历并获取索引与值了

```
const hd = ["hdcms", "houdunren"];

for (const [key, value] of hd.entries()) {
  console.log(key, value); //这样就可以遍历了
}

```

取数组中的最大值

```
function arrayMax(array) {
  let max = array[0];
  for (const elem of array) {
    max = max > elem ? max : elem;
  }
  return max;
}

console.log(arrayMax([1, 3, 2, 9]));

```

#迭代器方法

数组中可以使用多种迭代器方法，迭代器后面章节会详解。

#keys

通过迭代对象获取索引

```
const hd = ["houdunren", "hdcms"];
const keys = hd.keys();
console.log(keys.next());
console.log(keys.next());

```

获取数组所有键

```

```

```
"use strict";
const arr = ["a", "b", "c", "后盾人"];

for (const key of arr.keys()) {
  console.log(key);
}

```

使用 while 遍历

```
let arr = ["hdcms", "houdunren"];
while (({ value, done } = values.keys()) && done === false) {
        console.log(value);
}

```

#values

通过迭代对象获取值

```
const hd = ["houdunren", "hdcms"];
const values = hd.values();
console.log(values.next());
console.log(values.next());
console.log(values.next());

```

获取数组的所有值

```
"use strict";
const arr = ["a", "b", "c", "后盾人"];

for (const value of arr.values()) {
  console.log(value);
}

```

#entries

返回数组所有键值对，下面使用解构语法循环

```
const arr = ["a", "b", "c", "后盾人"];
for (const [key, value] of arr.entries()) {
  console.log(key, value);
}

```

解构获取内容（对象章节会详细讲解）

```
const hd = ["houdunren", "hdcms"];
const iterator = hd.entries();

let {done,value: [k, v]} = iterator.next();

console.log(v);

```

#扩展方法

#every

`every` 用于递归的检测元素，要所有元素操作都要返回真结果才为真。

查看班级中同学的 JS 成绩是否都及格

```
const user = [
  { name: "李四", js: 89 },
  { name: "马六", js: 55 },
  { name: "张三", js: 78 }
];
const resust = user.every(user => user.js >= 60);
console.log(resust);

```

标题的关键词检查

```
let words = ['后盾', '北京', '培训'];
let title = '后盾人不断分享技术教程';

let state = words.every(function (item, index, array) {
  return title.indexOf(item) >= 0;
});

if (state == false) console.log('标题必须包含所有关键词');

```

#some

使用 `some` 函数可以递归的检测元素，如果有一个返回 true，表达式结果就是真。第一个参数为元素，第二个参数为索引，第三个参数为原数组。

下面是使用 `some` 检测规则关键词的示例，如果匹配到一个词就提示违规。

```
let words = ['后盾', '北京', '武汉'];
let title = '后盾人不断分享技术教程'

let state = words.some(function (item, index, array) {
 return title.indexOf(item) >= 0;
});

if (state) console.log('标题含有违规关键词');

```

#filter

使用 `filter` 可以过滤数据中元素，下面是获取所有在 CSS 栏目的课程。

```
let lessons = [
  {title: '媒体查询响应式布局',category: 'css'},
  {title: 'FLEX 弹性盒模型',category: 'css'},
  {title: 'MYSQL多表查询随意操作',category: 'mysql'}
];

let cssLessons = lessons.filter(function (item, index, array) {
```

```
  if (item.category.toLowerCase() == 'css') {
    return true;
  }
});

console.log(cssLessons);

```

我们来写一个过滤元素的方法来加深些技术

```
function except(array, excepts) {
  const newArray = [];
  for (const elem of array)
    if (!excepts.includes(elem)) newArray.push(elem);
  return newArray;
}

const array = [1, 2, 3, 4];
console.log(except(array, [2, 3])); //[1,4]

```

#map

使用 `map` 映射可以在数组的所有元素上应用函数，用于映射出新的值。
获取数组所有标题组合的新数组

```
let lessons = [
  {title: '媒体查询响应式布局',category: 'css'},
  {title: 'FLEX 弹性盒模型',category: 'css'},
  {title: 'MYSQL多表查询随意操作',category: 'mysql'}
];

console.log(lessons.map(item => item.title));

```

为所有标题添加上 `后盾人`

```
let lessons = [
  {title: '媒体查询响应式布局',category: 'css'},
```

```
  {title: 'FLEX 弹性盒模型',category: 'css'},
  {title: 'MYSQL多表查询随意操作',category: 'mysql'}
];

lessons = lessons.map(function (item, index, array) {
    item.title = `[后盾人] ${item['title']}`;
    return item;
});
console.log(lessons);

```

#reduce

使用 `reduce` 与 `reduceRight` 函数可以迭代数组的所有元素，`reduce` 从前开始 `reduceRight` 从后面开始。下面通过函数计算课程点击数的和。

第一个参数是执行函数，第二个参数为初始值

- 传入第二个参数时将所有元素循环一遍
- 不传第二个参数时从第二个元素开始循环

函数参数说明如下

参数	说明
prev	上次调用回调函数返回的结果
cur	当前的元素值
index	当前的索引
array	原数组

统计元素出现的次数

```
function countArrayELem(array, elem) {
  return array.reduce((total, cur) => (total += cur == elem ? 1 : 0), 0);
}

let numbers = [1, 2, 3, 1, 5];
console.log(countArrayELem(numbers, 1)); //2

```

取数组中的最大值

```

```

```
function arrayMax(array) {
  return array.reduce(
        (max, elem) => (max > elem ? max : elem), array[0]
  );
}

console.log(arrayMax([1, 3, 2, 9]));

```

取价格最高的商品

```
let cart = [
  { name: "iphone", price: 12000 },
  { name: "imac", price: 25000 },
  { name: "ipad", price: 3600 }
];

function maxPrice(array) {
  return array.reduce(
    (goods, elem) => (goods.price > elem.price ? goods : elem),
    array[0]
  );
}
console.log(maxPrice(cart));

```

计算购物车中的商品总价

```
let cart = [
  { name: "iphone", price: 12000 },
  { name: "imac", price: 25000 },
  { name: "ipad", price: 3600 }
];

const total = cart.reduce(
 (total, goods) => total += goods.price, 0
);
console.log(total); //40600

```

获取价格超过 1 万的商品名称

```
let goods = [
  { name: "iphone", price: 12000 },
  { name: "imac", price: 25000 },
  { name: "ipad", price: 3600 }
];

function getNameByPrice(array, price) {
  return array.reduce((goods, elem) => {
    if (elem.price > price) {
      goods.push(elem);
    }
    return goods;
  }, []).map(elem => elem.name);
}
console.table(getNameByPrice(goods, 10000));

```

使用 `reduce` 实现数组去重

```
let arr = [1, 2, 6, 2, 1];
let filterArr = arr.reduce((pre, cur, index, array) => {
  if (pre.includes(cur) === false) {
      pre = [...pre, cur];
  }
  return pre;
}, [])
console.log(filterArr); // [1,2,6]

```

#动画案例


```
<style>
  body {
    width: 100vw;
    height: 100vh;
    display: flex;
    justify-content: center;
    align-items: center;
    background: #2c3e50;
  }

  * {
    padding: 0;
    margin: 0;
  }
  div {
    color: #9b59b6;
    font-size: 5em;
    font-weight: bold;
    text-transform: uppercase;
    cursor: pointer;
  }
```

```
  div > span {
    position: relative;
    display: inline-block;
  }
  .changeColor {
    animation-name: changeColor;
    animation-duration: 1s;
    animation-direction: alternate;
    animation-iteration-count: 2;
    animation-timing-function: linear;
  }
  @keyframes changeColor {
    50% {
      color: #f1c40f;
      transform: scale(1.5);
    }
    to {
      color: #9b59b6;
      transform: scale(0.5);
    }
  }
</style>

<body>
  <div>houdunren.com</div>
</body>

<script>
  let div = document.querySelector("div");
  [...div.textContent].reduce((pre, cur, index) => {
    pre == index && (div.innerHTML = "");
    let span = document.createElement("span");
    span.textContent = cur;
    div.appendChild(span);
    span.addEventListener("mouseover", function() {
      this.classList.add("changeColor");
    });
    span.addEventListener("animationend", function() {
      this.classList.remove("changeColor");
    });
```

```
  }, 0);
</script>
```