# 目录


- 改变this指向：
  - `show.call('call',10,20)`
  - `show.apply('apply',arr)`
  - `show.bind('bind')(10,20)`
- `let/const`关键字：不会声明提升
- 延时器：`setTimeout(函数，延迟时间);` `clearTimeout(返回值);`
- 箭头函数：`this`指向上一层的主人
  - `var 函数名 = () => {};`
- 解构：中括号解构；大括号解构
- ECMA6字符串：`` `${变量名/表达式/函数调用}` ``
- ECMA6新增数组的方法:
  - `Array.from()`：将伪数组转为真数组
  - `find()` / `findIndex()`：查找符合条件第一个元素 / 元素下标
  - `copyWithin(1,2,3)`：复制`2-3`位置元素,从`1`的位置开始覆盖
- `Object.assign( 目标对象,对象2 )`：合并对象
- `Set/Map`集合
- 数组/对象/集合/字符串 遍历的几种方式


## 1.认识this

- 每一个函数中都有一个内置的变量`this`，`this`指向当前函数的主人，函数的主人要根据上下文关系去进行判断。
  - `this`指向当前函数的主人。
- 常见的`this`,一般情况下是下面几种情况
  1. 全局函数，没有主人，指向window。

```
function show(){ alert(this); }
  show(); //[object Window]
```

  2. 对象的方法中的this指向当前对象。

```
var preson = {
  username : "钢铁侠",
  show: function(){
    alert( preson.username ); //钢铁侠
    alert( this.username ); //钢铁侠
  }
}
preson.show();
```

  3. 点击事件的函数`this`指向 点击的按钮对象(元素节点)

4. 定时器/延时器中的`this`，指向`window`对象
5. 事件监听器的`this`指向添加事件的节点。

## 2.强制改变this指向

- `call`
  - 格式：`函数名.call("call",10,20)`
    - 第一个参数：传入该函数`this`指向的对象，传入什么就强制指向什么
    - 第二个参数开始：将原函数的参数往后顺延一位。
  - `show.call("call",10,20);`：指向`"call"`字符串
- `apply`
  - 格式：`函数名.apply("apply",[20 , 40])`
    - 第一个参数：传入该函数`this`指向的对象，传入什么就强制指向什么
    - 第二个参数：数组，数组中按顺序放入原有的参数
  - `show.apply("apply",[20 , 40]);`

  - `apply`使用小技巧，因为`apply`第二个参数是数组，可以`Math`中传入数组

```
var arr = [10,20,30,40,50];
alert(Math.max.apply(null,arr)); //50
```

- 上述两种方法改变指向后立即执行

- `bind`：预设this指向，不会立即执行
  - `bind`只会改变指向，需要拿到改变指向后的函数，然后调用

```
show.bind("bind")(40,50);  // 第一种

// 第二种
var res = show.bind("bind");
res(40,50);
```

    - `bind`可以用在事件函数上，改变后不会立刻执行

```
oBtn.onclick = show.bind("bind");
addEventListener("onclick",show.bind("bind"),false);
```

- 关于函数带括号和不带括号的区别

```
function fun(){
  return 1;
}
var a = fun;    // fun 相当于函数地址
var b = fun();  // fun() 返回的是执行完毕的return值
```

## 3.ECMA6 (ECMA2015)新增语法

- ES6 规定暂时性死区和 `let`、`const` 语句**不出现变量提升**

- `let` 关键字
  - `let` 关键字声明变量 只要遇到大括号就形成作用域
    - `let` 关键字声明的变量，**所在作用域叫做块级作用域。**
    - `let` 不会声明提升，**在 `let` 声明前使用，会报错。**
  - 选项卡获取当前下标的问题：

```
// 以前要添加自定义属性才可以
// 现在用let就可以获取
var aBtns = document.getElementsByTagName("button");
for(let i = 0; i < aBtns.length; i++){
  aBtns[i].onclick = function(){
    alert(i);
  }
}
```

- `const` 关键字
  - `const` 声明变量的值，变量值只能在声明的时候确定，**后续是无法更改**的。
    - `const` 声明常量( 不能更改 `const` 常量 指针 );
    - `const` 不会声明提升，**在 `const` 声明前使用，会报错。**
  - 应用场景：改动频率几乎为0，又得声明变量方便使用，用 `const` 声明常量。

- 延时器
  - `setTimeout(函数，延迟时间);`：在延迟时间过后执行函数
  - `clearTimeout(返回值);`：关闭延时器。

## 4.箭头函数

- 尽量使用普通函数，写出人类容易理解的代码。
  - IE11 依然不兼容箭头函数
- 箭头函数：新潮的函数写法(只是给函数的写法换了种形式)

- 正常函数

```
function 函数名(形参){
  // 函数体;
}
```

- 箭头函数

```
(形参) => {函数体};
```

  - 没有函数名，可以通过 `var 函数名 = () => {};` 来设置函数名;
  - 适当的省略函数中的 `function` 和 `return` 关键字。
- 当函数体只有一个 `return` 时 使用箭头函数是很方便的

```
// 计算一个数加10后的值
var add2 = (x) => x + 10;
```

- 当函数体只有一个`return`时，不用给函数体加`{}`

## 5.箭头函数 和 ECMA5数组方法结合

- `arr.filter( item => item > 20);`
- `arr.map(item => item * 1.3);`
  - 函数体只有一个`return`的情况下，不用加`{}`

- 使用箭头函数需要注意的部分
  1. 箭头函数，不能用`new`
  2. 箭头函数**返回值**如果**是一个对象**，一定要**加`()`**; (会优先计算小括号的内容)

```
function show(){
    return{
        username: "xxx";
    }
}
var show = () => ({
    username: "xxx";
})
```

     - 关于函数返回对象的应用 (可以给对象赋值属性，方法)

```
var show2 = () => ({
    username : "xxx"
})
var parson2 = new Object();
parson2 = show2(); // 会覆盖对象原来的属性和方法。
  // 指向了一个新对象
```

  3. 箭头函数的`this`，**指向的是上一层函数的主人。**

```
var person = {
    username: "钢铁侠",
    show: () =>{
        alert(person.username);
        alert(this);    //[object Window]
    }
}
person.show();  // this指向window
```

## 6.解构

- 中括号解构

```
var [x , y , z] = [10 ,20 ,30];
alert(x + "," + y + "," + z);   //10,20,30

var [x, [a, b], y] = [10, [20 ,30], 40];
//x 10,  a 20,  b 30,  y 40

var [x, [a, b], y] = [10, [20], 40];
//少了b的值，b也是可以正常声明的，undefined
```

- 大括号解构
  - 赋值不是对象中的属性，是一个变量

```
var {name, age, sex} = {
  age: 18,
  name: "钢铁侠",
  sex: "男",
}
alert(name);  //钢铁侠
alert(age);   //18
alert(sex);   //男
```

- 中括号解构和大括号解构的区别
  1. 中括号解构要一一对应，大括号解构不用一一对应。
  2. 中括号解构可以多层嵌套
  - 大括号解构 就是 复制了一份 变量的引用而已 改变左侧变量 也会直接改变引用的值

- 解构对传统的颠覆(好处)
  1. 交换两个数的值

```
var [x , y] = [10 , 20];
[x , y] = [y , x];      //20 , 10
```

  2. 函数可以返回多个值 (类似于返回数组，但比返回数组更方便)

```
function show(){
    return ["结果1" ,"结果2" ,"结果3"]
}
var [a , b , c] = show();
```

  3. 函数定义参数 和 传入参数的顺序改变 （参数可以带默认值）
     - 形参写成大括号解构

```
function showSelf( {name="xxx" , age , sex} ){
  alert("我叫" + name +", 今年" + age + "岁，是一位" + sex + "性");
}
showSelf({
  name : "小明", sex : "男", age : 18
});
```

    - 参数可以带默认值：`function showSelf( {name , age , sex = "男"}`
        - 当不给`sex`传参时，默认是`男`
4. 快速取出数组中的某一个元素

```
var arr = [10 ,20 ,30 ,40 ,50];
var {0:first, 3:last} = arr;
alert(first + "," + last);  // 10  40
```

## 7.ECMA6字符串

- 传统字符串：所有单引号(`''`)，双引号(`""`)括起来的都叫做字符串
    - IE 11不兼容 ECMA6字符串

- ECMA6字符串：反引号 ``` ``
    1. ECMA6字符串，想怎么写就怎么写，换行，代码缩进，都能在字符串中体现出来
    2. ECMA6字符串 可以将变量嵌入字符串 `` `${变量名/表达式/函数调用}` ``

```
alert(`我叫${name},最大值${Math.max(20,30)},调用show函数${show()}`);
```

- 字符串中可以使用`\n`换行符 `\r`回车字符 来进行换行

```
var str = 'hello \n world';   //传统字符串
var str2 = `hello \n world`;  //ECMA6字符串
```

## 8.ECMA6新增数组的方法

- `Array.from()` ：将伪数组转为真数组
    - 格式： `var arr = Array.from(伪数组);`
    - 返回值：转换的真数组
        - 伪数组：
        - `node.getElementByClassName()`
        - `node.getElementByTagName()`
        - `document.getElementByName()`
        - `node.querySelectorAll()`
        - `node.childNodes`
        - `node.children`
    - 伪数组概念：1、拥有length属性。2、不具有数组所具有的方法

- `find()` ：查找符合条件第一个元素
    - 功能：在数组中查找符合条件的元素。只要找到第一个符合条件的元素，就会终止遍历
    - 返回值：找到的元素。

```
arr.find( (item,index,arr) => item > 30);
arr.find(function(item,index,arr){ return item > 30; })
```

- `findIndex()` ：查找符合条件第一个元素的下标
  - 功能：在数组中查找符合条件的第一个元素的下标。
  - 返回值：找到元素的下标
  - 格式：同上

- `copyWithin(1,2,3)`
  - 第一个参数：从哪个下标开始
  - 第二个参数和第三个参数：坐标的范围 [start,end)
  - 功能：复制第二到第三范围的参数,从第一参数位置开始是向后覆盖

## 9.合并对象（Object.assign( 目标对象,对象2 ) )

- `Object.assign( 目标对象,对象2 )`
  - 功能：将所有传入的对象都合并到 目标对象中，其他对象不会被改变。
    - 复制对象： `var newobj = Object.assign({},对象);`
  - 如果对象中有重名的属性和方法，会以后填写的对象为准。
  - `Object.assign` 和 赋值运算符 `=` 一样，属于浅拷贝

## 10.ECMA6集合

- 集合：1.不重复; 2.无序
- `Set` 集合
  - Set集合默认去重，前提是两个添加的元素严格相等， `5` 和 `"5"` 不相等，两个 `new string` 的字符串不相等。
  - **声明** `Set` 集合： `let set = new Set();`
    - `Set.size` ：获取当前 `Set` 对象的长度
    - `Set.add("value");` ：赋值，键和值都是 `value`
    - `Set.delete("value")` ：删除
    - `Set.has("value")` ：检测 `value` 是否存在 `Set` 中，返回布尔值
    - `Set.clear()` ：清除 `Set` 所有元素
  - 集合中可以添加多个相同字符串，原因如下

```
set.add(new String("world"));
set.add(new String("world"));
```

    - 通过 `String()` 声明的是基本数据类型字符串
    - 通过 `new String()` 声明的是对象类型字符串

  - `Set` 集合使用场景
    - set-数组变集合 （可以将数组中的元素去重）

```
let arr = [10,20,30,10,20];
let set = new Set(arr);     //10,20,30
```

    - set-集合变数组

```
set.add('a');
var arr = [...set];  //a
```

    - set-数组去重

```
let arr = [10,20,30,10,20];
arr = [...new Set(arr)];
```

- Map集合
    - 赋值：`map.set("键/key"，值/value)`
        - 键和值可以不一样；键要有唯一性，会覆盖。
    - 取值：`map.get("键/key")`
    - `Map.size`：获取当前`Map`对象的长度
    - `Map.delete(key)`：删除
    - `Map.has(key)`：检测`key`是否存在`Map`中，返回布尔值
    - `Map.clear()`：清除`Map`所有元素

    - 集合遍历：`for...of`和`Set`集合一样

```
for( let [key,value] of map) //配合解构使用
```

    - Map集合应用场景
        - map-数组变集合：**不行**
        - map-集合变数组

```
map.set('a',1)
var arr = [...map];  //[a,1]
// map的每一组数据使用扩展运算符拆分后都是一个数组
```

- 集合的遍历
    - `for...of`（不会改变原来集合的内容）

```
// 通过 集合.keys() 取键
for(let item of set.keys() ){
    console.log(item);
}
```

```
// 通过 集合.values() 取值
for(let item of set.values() ){
    console.log(item);
}
```

```
// 通过 集合.entries 取键和值
for(let item of set.entries() ){
    console.log(item);
}
// 集合.entries()遍历，返回的是一个个数组
```

- 通过`forEach`遍历

```
set.forEach((key,value) => {
  console.log(key,value);
})
```

## 11.数组/对象/集合/字符串 遍历的几种方式

- `Set/Map`集合
  for...of (keys/values/entries) for(var item of set.keys()){ }
  forEach(不常用)
- 数组
  1. `for`循环
  2. `for...in` （数组/对象/字符串）
  3. `forEach` （数组/集合）
  4. `for...of`
- 对象
  `for...in`唯一遍历方法
  - `for...in`遍历对象时，`i`代表的是对象的属性名，并且`i`是字符串类型，要用`对象[i]`来取值
  - `for...in`遍历数组和字符串时,遍历的是对象的下标
- 字符串
  1. `for`循环
  2. `for...in`
- `for...in`和`for...of`的区别
  - `for...in`遍历的是下标，会遍历原型链的属性和方法，需要迭代器
  - `for...of`遍历的是元素，不会遍历原型链的属性和方法