

基础知识

函数是将复用的代码块封装起来的模块，在 JS 中函数还有其他语言所不具有的特性，接下来我们会详细掌握使用技巧。

#声明定义

在 JS 中函数也是对象函数是 `Function` 类的创建的实例，下面的例子可以方便理解函数是对象。

```
1 let hd = new Function("title", "console.log(title)");
2 hd('后盾人');
3
```

标准语法是使用函数声明来定义函数

```
1 function hd(num) {
2   return ++num;
3 }
4 console.log(hd(3));
5
```

对象字面量属性函数简写

```
1 let user = {
2   name: null,
3   getName: function (name) {
4     return this.name;
5   },
6   //简写形式
7   setName(value) {
8     this.name = value;
9   }
10 }
11 user.setName('后盾人');
12 console.log(user.getName()); // 后盾人
13
```

全局函数会声明在 window 对象中，这不正确建议使用后面章节的模块处理

```
1 console.log(window.screenX); //2200
2
```

当我们定义了 `screenX` 函数后就覆盖了 `window.screenX` 方法

```
1 function screenX() {
2   return "后盾人";
3 }
4 console.log(screenX()); //后盾人
5
```

使用 `let/const` 时不会压入 `window`

```
1 let hd = function() {
2   console.log("后盾人");
3 };
4 window.hd(); //window.hd is not a function
5
```

#匿名函数

函数是对象所以可以通过赋值来指向到函数对象的指针，当然指针也可以传递给其他变量，注意后面要以 `;` 结束。下面使用函数表达式将 `匿名函数` 赋值给变量

```
1 let hd = function(num) {
2   return ++num;
3 };
4
5 console.log(hd instanceof Object); //true
6
7 let cms = hd;
8 console.log(cms(3));
9
```

标准声明的函数优先级更高，解析器会优先提取函数并放在代码树顶端，所以标准声明函数位置没有限制，所以下面的代码可以正常执行。

```
1 console.log(hd(3));
```

```
2 function hd(num) {
3   return ++num;
4 };
5
```

标准声明优先级高于赋值声明

```
1 console.log(hd(3)); //4
2
3 function hd(num) {
4   return ++num;
5 }
6
7 var hd = function() {
8   return "hd";
9 };
10
```

程序中使用匿名函数的情况非常普遍

```
1 function sum(...args) {
2   return args.reduce((a, b) => a + b);
3 }
4 console.log(sum(1, 2, 3));
5
```

#立即执行

立即执行函数指函数定义时立即执行

- 可以用来定义私有作用域防止污染全局作用域

```
1 "use strict";
2 (function () {
3   var web = 'houdunren';
4 })();
5 console.log(web); //web is not defined
6
```

使用 `let/const` 有块作用域特性，所以使用以下方式也可以产生私有作用域

```
1 {  
2   let web = 'houdunren';  
3 }  
4 console.log(web);  
5
```

#函数提升

函数也会提升到前面，优先级行于`var`变量提高

```
1 console.log(hd()); //后盾人  
2 function hd() {  
3   return '后盾人';  
4 }  
5
```

变量函数定义不会被提升

```
1 console.log(hd()); //后盾人  
2  
3 function hd() {  
4   return '后盾人';  
5 }  
6 var hd = function () {  
7   return 'hdcms.com';  
8 }  
9
```

#形参实参

形参是在函数声明时设置的参数，实参指在调用函数时传递的值。

- 形参数量大于实参时，没有传参的形参值为 `undefined`
- 实参数量大于形参时，多于的实参将忽略并不会报错

```
1 // n1,n2 为形参  
2 function sum(n1, n2) {  
3   return n1+n2;  
4 }
```

```
4 }
5 // 参数 2,3 为实参
6 console.log(sum(2, 3)); //5
7
```

当没传递参数时值为 undefined

```
1 function sum(n1, n2) {
2   return n1 + n2;
3 }
4 console.log(sum(2)); //NaN
5
```

#默认参数

下面通过计算年平均销售额来体验以往默认参数的处理方式

```
1 //total:总价 year:年数
2 function avg(total, year) {
3   year = year || 1;
4   return Math.round(total / year);
5 }
6 console.log(avg(2000, 3));
7
```

使用新版本默认参数方式如下

```
1 function avg(total, year = 1) {
2   return Math.round(total / year);
3 }
4 console.log(avg(2000, 3));
5
```

下面通过排序来体验新版默认参数的处理方式，下例中当不传递 type 参数时使用默认值 asc。

```
1 function sortArray(arr, type = 'asc') {
2   return arr.sort((a, b) => type == 'asc' ? a - b : b - a);
3 }
4 console.log(sortArray([1, 3, 2, 6], 'desc'));
```

默认参数要放在最后面

```

1 //total:价格,discount:折扣,dis:折后折
2 function sum(total, discount = 0, dis = 0) {
3   return total * (1 - discount) * (1 - dis);
4 }
5 console.log(sum(2000, undefined, 0.3));
6

```

#函数参数

函数可以做为参数传递，这也是大多数语言都支持的语法规则。

```

1 <body>
2   <button>订阅</button>
3 </body>
4 <script>
5   document.querySelector('button').addEventListener('click', function () {
6     alert('感谢订阅');
7   })
8 </script>
9

```

函数可以做为参数传递

```

1 function filterFun(item) {
2   return item <= 3;
3 }
4 let hd = [1, 2, 3, 4, 5].filter(filterFun);
5 console.log(hd); //[1,2,3]
6

```

#arguments

arguments 是函数获得到所有参数集合，下面是使用 arguments 求和的例子

```
1 function sum() {
2   return [...arguments].reduce((total, num) => {
3     return (total += num);
4   }, 0);
5 }
6 console.log(sum(2, 3, 4, 2, 6)); //17
7
```

更建议使用展示语法

```
1 function sum(...args) {
2   return args.reduce((a, b) => a + b);
3 }
4 console.log(sum(2, 3, 4, 2, 6)); //17
5
```

#箭头函数

箭头函数是函数声明的简写形式，在使用递归调用、构造函数、事件处理器时不建议使用箭头函数。无参数时使用空扩号即可

```
1 let sum = () => {
2   return 1 + 3;
3 }
4 console.log(sum()); //4
5
```

函数体为单一表达式时不需要 `return` 返回处理，系统会自动返回表达式计算结果。

```
1 let sum = () => 1 + 3;
2 console.log(sum()); //4
3
```

多参数传递与普通声明函数一样使用逗号分隔

```
1 let hd = [1, 8, 3, 5].filter((item, index) => {
2   return item <= 3;
3
```

```
3 });  
4 console.log(hd);  
5
```

只有一个参数时可以省略括号

```
1 let hd = [1, 8, 3, 5].filter(item => item <= 3);  
2 console.log(hd);  
3
```

有关箭头函数的作用域知识会在后面章节讨论

#递归调用

递归指函数内部调用自身的方式。

- 主要用于数量不确定的循环操作
- 要有退出时机否则会陷入死循环

下面通过阶乘来体验递归调用

```
1 function factorial(num = 3) {  
2   return num == 1 ? num : num * factorial(--num);  
3 }  
4 console.log(factorial(5)); //120  
5
```

累加计算方法

```
1 function sum(...num) {  
2   return num.length == 0 ? 0 : num.pop() + sum(...num);  
3 }  
4 console.log(sum(1, 2, 3, 4, 5, 7, 9)); //31  
5
```

递归打印倒三角

```
1 *****  
2 *****  
3 *****  
4 ***
```



```

5  **
6  *
7
8  function star(row = 5) {
9    if (row == 0) return "";
10   document.write("".repeat(row) + "<br/>");
11   star(--row);
12 }
13

```

使用递归修改课程点击数

```

1  let lessons = [
2    {
3      title: "媒体查询响应式布局",
4      click: 89
5    },
6    {
7      title: "FLEX 弹性盒模型",
8      click: 45
9    },
10   {
11     title: "GRID 栅格系统",
12     click: 19
13   },
14   {
15     title: "盒子模型详解",
16     click: 29
17   }
18 ];
19 function change(lessons, num, i = 0) {
20   if (i == lessons.length) {
21     return lessons;
22   }
23   lessons[i].click += num;
24   return change(lessons, num, ++i);
25 }
26 console.table(change(lessons, 100));
27

```

#回调函数

在某个时刻被其他函数调用的函数称为回调函数，比如处理键盘、鼠标事件的函数。

```
1 <button id='hd'>button</button>
2 <script>
3     document.getElementById('hd').addEventListener('click', () => alert('通过回调函数调用'));
4 </script>
5
```

使用回调函数递增计算

```
1 let hd = ([1, 2, 3]).map(item => item + 10);
2 console.log(hd)
3
```

#展开语法

展示语法或称点语法体现的就是收/放特性，做为值时是放，做为接收变量时是收。

```
1 let hd = [1, 2, 3];
2 let [a, b, c] = [...hd];
3 console.log(a); //1
4 console.log(b); //2
5 console.log(c); //3
6 [...hd] = [1, 2, 3];
7 console.log(hd); //[1, 2, 3]
8
```

使用展示语法可以替代 `arguments` 来接收任意数量的参数

```
1 function hd(...args) {
2     console.log(args);
3 }
4 hd(1, 2, 3, "后盾人"); //[1, 2, 3, "后盾人"]
5
```

也可以用于接收部分参数

```

1 function hd(site, ...args) {
2   console.log(site, args); //后盾人 (3) [1, 2, 3]
3 }
4 hd("后盾人", 1, 2, 3);
5

```

使用 `...` 可以接受传入的多个参数合并为数组，下面是使用点语法进行求合计算。

```

1 function sum(...params) {
2   console.log(params);
3   return params.reduce((pre, cur) => pre + cur);
4 }
5 console.log(sum(1, 3, 2, 4));
6

```

多个参数时 `...参数` 必须放后面，下面计算购物车商品折扣

```

1 function sum(discount = 0, ...prices) {
2   let total = prices.reduce((pre, cur) => pre + cur);
3   return total * (1 - discount);
4 }
5 console.log(sum(0.1, 100, 300, 299));
6

```

#标签函数

使用函数来解析标签字符串，第一个参数是字符串值的数组，其余的参数为标签变量。

```

1 function hd(str, ...values) {
2   console.log(str); //["站点", "-", "", raw: Array(3)]
3   console.log(values); //["后盾人", "houdunren.com"]
4 }
5 let name = '后盾人', url = 'houdunren.com';
6 hd `站点${name}-${url}`;
7

```

#this

调用函数时 `this` 会隐式传递给函数指函数调用时的关联对象，也称之为函数的上下文。

#函数调用

全局环境下 `this` 就是 window 对象的引用

```
1 <script>
2   console.log(this == window); //true
3 </script>
4
```

使用严格模式时在全局函数内 `this` 为 `undefined`

```
1 var hd = '后盾人';
2 function get() {
3   "use strict"
4   return this.hd;
5 }
6 console.log(get());
7 //严格模式将产生错误 Cannot read property 'name' of undefined
8
```

#方法调用

函数为对象的方法时 `this` 指向该对象

可以使用多种方式创建对象，下面是使用构造函数创建对象

构造函数

函数当被 `new` 时即为构造函数，一般构造函数中包含属性与方法。函数中的上下文指向到实例对象。

- 构造函数主要用来生成对象，里面的 `this` 默认就是指当前对象

```
1 function User() {
2   this.name = "后盾人";
3   this.say = function() {
4     console.log(this); //User {name: "后盾人", say: f}
5     return this.name;
6   };
7 }
8 let hd = new User();
9 console.log(hd.say()); //后盾人
```

对象字面量

- 下例中的 `hd` 函数不属于对象方法所以指向

`window`

- `show` 属于对象方法指向

`obj`对象

```

1 let obj = {
2   site: "后盾人",
3   show() {
4     console.log(this.site); //后盾人
5     console.log(`this in show method: ${this}`); //this in show method: [object
      Object]
6     function hd() {
7       console.log(typeof this.site); //undefined
8       console.log(`this in hd function: ${this}`); //this in hd function:
          [object Window]
9     }
10    hd();
11  }
12 };
13 obj.show();
14
```

在方法中使用函数时有些函数可以改变 `this` 如 `forEach`，当然也可以使用后面介绍的 `apply/call/bind`

```

1 let Lesson = {
2   site: "后盾人",
3   lists: ["js", "css", "mysql"],
4   show() {
5     return this.lists.map(function(title) {
6       return `${this.site}-${title}`;
7     }, this);
8   }
9 };
10 console.log(Lesson.show());
11
```

也可以在父作用域中定义引用 `this` 的变量

```

1 let Lesson = {
2   site: "后盾人",
3   lists: ["js", "css", "mysql"],
4   show() {
5     const self = this;
6     return this.lists.map(function(title) {
7       return `${self.site}-${title}`;
8     });
9   }
10 };
11 console.log(Lesson.show());
12

```

#箭头函数

箭头函数没有 `this`，也可以理解为箭头函数中的 `this` 会继承定义函数时的上下文，可以理解为和外层函数指向同一个 `this`。

- 如果想使用函数定义时的上下文中的 `this`，那就使用箭头函数

下例中的匿名函数的执行环境为全局所以 `this` 指向 `window`。

```

1 var name = 'hdcms';
2 var obj = {
3   name: '后盾人',
4   getName: function () {
5     return function () {
6       return this.name;
7     }
8   }
9 }
10 console.log(obj.getName())(); //返回window.name的值hdcms
11

```

以往解决办法会匿名函数调用处理定义变量，然后在匿名函数中使用。

```

1 var name = 'hdcms';
2 var obj = {
3   name: '后盾人',

```

```

4   getName: function () {
5       var self = this;
6           return () => {
7               return this.name;
8           }
9   }
10 }
11 console.log(obj.getName()); //返回window.name的值hdcms
12

```

使用箭头函数后 `this` 为定义该函数的上下文，也可以理解为定义时父作用域中的 `this`

```

1   var name = 'hdcms';
2   var obj = {
3       name: '后盾人',
4       getName: function () {
5           return () => {
6               return this.name;
7           }
8       }
9   }
10 console.log(obj.getName()); //后盾人
11

```

事件中使用箭头函数结果不是我们想要的

- 事件函数可理解为对象

onclick设置值，所以函数声明时

this为当前对象

- 但使用箭头函数时

this为声明函数上下文

下面体验使用普通事件函数时 `this` 指向元素对象

使用普通函数时 `this` 为当前 DOM 对象

```

1   <body>
2       <button desc="hdcms">button</button>
3   </body>
4   <script>
5       let Dom = {
6           site: "后盾人",

```

```

7     bind() {
8         const button = document.querySelector("button");
9         button.addEventListener("click", function() {
10             alert(this.getAttribute("desc"));
11         });
12     }
13 };
14 Dom.bind();
15 </script>
16

```

下面是使用箭头函数时 this 指向上下文对象

```

1 <body>
2   <button desc="hdcms">button</button>
3 </body>
4 <script>
5   let Dom = {
6     site: "后盾人",
7     bind() {
8       const button = document.querySelector("button");
9       button.addEventListener("click", event => {
10         alert(this.site + event.target.innerHTML);
11       });
12     }
13   };
14   Dom.bind();
15 </script>
16

```

使用 `handleEvent` 绑定事件处理器时，`this` 指向当前对象而不是 DOM 元素。

```

1 <body>
2   <button desc="hdcms">button</button>
3 </body>
4 <script>
5   let Dom = {
6     site: "后盾人",
7     handleEvent: function(event) {

```



```
8     console.log(this);
9 },
10 bind() {
11     const button = document.querySelector("button");
12     button.addEventListener("click", this);
13 }
14 };
15 Dom.bind();
16 </script>
17
```

#apply/call/bind

改变 this 指针，也可以理解为对象借用方法，就现像生活中向邻居借东西一样的事情。

#原理分析

构造函数中的 `this` 默认是一个空对象，然后构造函数处理后把这个空对象变得有值。

```
1 function User(name) {
2     this.name = name;
3 }
4 let hd = new User("后盾人");
5
```

可以改变构造函数中的空对象，即让构造函数 this 指向到另一个对象。

```
1 function User(name) {
2     this.name = name;
3 }
4
5 let hdcms = {};
6 User.call(hdcms, "HDCMS");
7 console.log(hdcms.name); //HDCMS
8
```

#apply/call

call 与 apply 用于显示的设置函数的上下文，两个方法作用一样都是将对象绑定到 this，只是在传递参数上有所不同。

- apply 用数组传参
- call 需要分别传参
- 与 bind 不同 call/apply 会立即执行函数

语法使用介绍

```
1 function show(title) {
2     alert(`${title+this.name}`);
3 }
4 let lisi = {
5     name: '李四'
6 };
7 let wangwu = {
8     name: '王五'
9 };
10 show.call(lisi, '后盾人');
11 show.apply(wangwu, ['HDCMS']);
12
```

使用 `call` 设置函数上下文

```
1 <body>
2     <button message="后盾人">button</button>
3     <button message="hdcms">button</button>
4 </body>
5 <script>
6     function show() {
7         alert(this.getAttribute('message'));
8     }
9     let bts = document.getElementsByTagName('button');
10    for (let i = 0; i < bts.length; i++) {
11        bts[i].addEventListener('click', () => show.call(bts[i]));
12    }
13 </script>
14
```

找数组中的数值最大值

```
1 let arr = [1, 3, 2, 8];
```

```
2 console.log(Math.max(arr)); //NaN
3 console.log(Math.max.apply(Math, arr)); //8
4 console.log(Math.max(...arr)); //8
5
```

实现构造函数属性继承

```
1 "use strict";
2 function Request() {
3   this.get = function(params = {}) {
4     //组合请求参数
5     let option = Object.keys(params)
6       .map(i => i + "=" + params[i])
7       .join("&");
8
9     return `获取数据 API:${this.url}?${option}`;
10  };
11 }
12 //文章控制器
13 function Article() {
14   this.url = "article/index";
15   Request.apply(this, []);
16 }
17 let hd = new Article();
18 console.log(
19   hd.get({
20     row: 10,
21     start: 3
22   })
23 );
24 //课程控制器
25 function Lesson() {
26   this.url = "lesson/index";
27   Request.call(this);
28 }
29 let js = new Lesson();
30 console.log(
31   js.get({
32     row: 20
```

```
33     })
34   );
35
```

制作显示隐藏面板



```
1  <style>
2    * {
3      padding: 0;
4      margin: 0;
5    }
6
7    body {
8      display: flex;
9      justify-content: center;
10     align-items: center;
11     width: 100vw;
12     height: 100vh;
13   }
14
15   dl {
16     width: 400px;
17     display: flex;
18     flex-direction: column;
```

```
19     }
20
21     dt {
22         background: #e67e22;
23         border-bottom: solid 2px #333;
24         height: 50px;
25         display: flex;
26         justify-content: center;
27         align-items: center;
28         cursor: pointer;
29     }
30
31     dd {
32         height: 200px;
33         background: #bdc3c7;
34         font-size: 5em;
35         text-align: center;
36         line-height: 200px;
37     }
38 </style>
39
40 <body>
41     <dl>
42         <dt>后盾人</dt>
43         <dd>1</dd>
44         <dt>hdcms</dt>
45         <dd hidden="hidden">2</dd>
46     </dl>
47 </body>
48 <script>
49     function panel(i) {
50         let dds = document.querySelectorAll("dd");
51         dds.forEach(item => item.setAttribute("hidden", "hidden"));
52         dds[i].removeAttribute("hidden");
53     }
54     document.querySelectorAll("dt").forEach((dt, i) => {
55         dt.addEventListener("click", () => panel.call(null, i));
56     });
57 </script>
58
```

#bind

bind()是将函数绑定到某个对象，比如 a.bind(hd) 可以理解为将 a 函数绑定到 hd 对象上即 hd.a()。

- 与 call/apply 不同 bind 不会立即执行
- bind 是复制函数形为会返回新函数

bind 是复制函数行为

```
1 let a = function() {};  
2 let b = a;  
3 console.log(a === b); //true  
4 //bind是新复制函数  
5 let c = a.bind();  
6 console.log(a == c); //false  
7
```

绑定参数注意事项

```
1 function hd(a, b) {  
2     return this.f + a + b;  
3 }  
4  
5 //使用bind会生成新函数  
6 let newFunc = hd.bind({ f: 1 }, 3);  
7  
8 //1+3+2 参数2赋值给b即 a=3,b=2  
9 console.log(newFunc(2));  
10
```

在事件中使用bind

```
1 <body>  
2   <button>后盾人</button>  
3 </body>  
4 <script>  
5   document.querySelector("button").addEventListener(  
6     "click",  
7     function(event) {  
8       console.log(event.target.innerHTML + this.url);  
9     }  
10  );
```

```
9     }.bind({ url: "houdunren.com" })
10   );
11 </script>
12
```

动态改变元素背景颜色，当然下面的例子也可以使用箭头函数处理



houdunren.com

```
1 <style>
2   * {
3     padding: 0;
4     margin: 0;
5   }
6
7   body {
8     width: 100vw;
9     height: 100vh;
10    font-size: 3em;
11    padding: 30px;
12    transition: 2s;
13    display: flex;
14    justify-content: center;
15    align-items: center;
16    background: #34495e;
17    color: #34495e;
18  }
```

```
19 </style>
20 <body>
21   houdunren.com
22 </body>
23 <script>
24   function Color(elem) {
25     this.elem = elem;
26     this.colors = ["#74b9ff", "#ffeaa7", "#fab1a0", "#fd79a8"];
27     this.run = function() {
28       setInterval(
29         function() {
30           let pos = Math.floor(Math.random() * this.colors.length);
31           this.elem.style.background = this.colors[pos];
32         }.bind(this),
33         1000
34       );
35     };
36   }
37   let obj = new Color(document.body);
38   obj.run();
39 </script>
```