博学谷
www.boxuegu.com

第六章 函数的扩展

一样的在线教育，不一样的教学品质

目录 Contents

函数参数的默认值

小节导学

使用Ajax发送请求来获取对应的数据，为了方便操作，可将Ajax的操作封装到一个函数中，如下伪代码所示：

```
function ajaxAction(url, method, dataType) {
if (typeof url === 'undefined') throw Error('请求地址不能为空!!');
method = method ? method : 'GET';
 }
```

函数参数的默认值

采用**函数参数默认值**的方式实现：

```
function ajaxAction(url = new Error('请求地址不能为空!!'), method = 'GET', dataType = 'json') {
    console.log(url);
    console.log(method);
    console.log(dataType);
}
ajaxAction('/showUser');
```

说明：给ajaxAction函数的形参指定相应的默认值；那么，在调用该函数时，若传递了新的值，对应的形参为传递的值；若没有传递值，则形参为默认值。

优点：

1）方便简单，

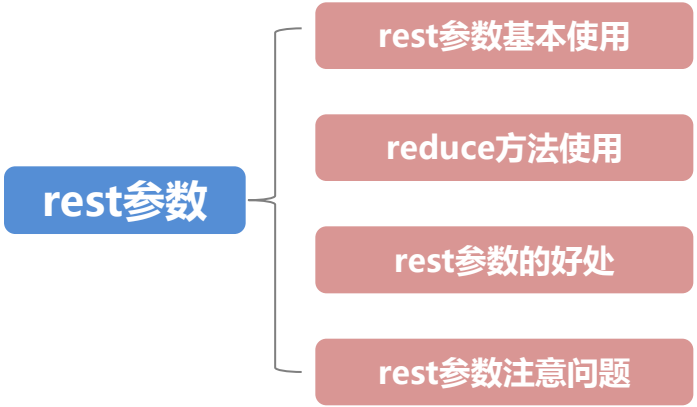
2）有利于代码的阅读；当看到这段代码后，就能明白哪些参数是可以省略的，不用查看函数体和文档。

目录 Contents

- 函数参数的默认值
- rest参数
- 扩展运算符
- 箭头函数

rest参数

小节导学



1.rest参数基本使用

rest参数：

形式："**...变量名**"，

应用场景：用于**获取函数中的多余参数**，因此，不需要使用arguments对象。

说明：rest参数搭配的变量是一个数组；

示例：

```
function add(...values) {
        console.log(values);
    }
    add(2, 3);
```

2.reduce方法基本使用

该方法作用：计算与汇总，可以把数组中的所有值计算出一个值。

```
function add(...values) {
        values.reduce(function(val, item, index, origin) {
            console.log('val=', val);
            console.log("item=", item);
            console.log("index=", index);
            console.log("origin=", origin);
        }, 0)
    }
    add(1, 2, 3)
```

Reduce方法的使用：

第一个参数：一个函数；第二个参数：一个初始值；

运行时：第一次执行时：第二个参数（也就是初始值）会赋值给函数中val参数；

输出时：val的值都是undefined；

原因：val参数后面保存的都是函数中返回的值，而上述代码中，函数中没有返回值，所以，val参数的值为undefined。

item参数：存储数组中的每一项；

index参数：存储对应的下标；

origin参数：存储原来的数组内容。

3.通过reduce方法完成求和运算

通过reduce完成加法运算：

```
function add(...values) {
            return values.reduce(function(val, item, index, origin) {
                return val + item;
            }, 0)
        }
        console.log(add(1, 2, 3));
```

4.通过reduce方法完成求平均值

通过reduce方法完成求平均值：

```
function add(...values) {
    return values.reduce(function(val, item, index, origin) {
        let sum = val + item;
        if (index === origin.length - 1) {
            return sum / origin.length;
        } else {
            return sum;
        }
    }, 0)
}

console.log(add(1, 2, 3));
```

5.reduceRight方法基本使用

特征：reduce是按照从左到右的顺序对数组中的数据进行计算。

reduceRight是从右向左进行计算。

使用reduceRight改造求平均值的案例：

```
function add(...values) {

        return values.reduceRight(function(val, item, index, origin) {

            let sum = val + item;

            if (index === 0) {
                return sum / origin.length;
            } else {
                return sum;
            }
        }, 0)
    }

    console.log(add(1, 2, 3));
```

6.模拟实现reduce和reduceRight函数

模拟实现 reduce：

```
Array.prototype.reduce1 = function(reduceFunc, initValue) {
            for (let i = 0; i < this.length; i++) {
                initValue = reduceFunc(initValue, this[i]);
            }
            return initValue;
        }
        let arr = [1, 2, 3];
        let result = arr.reduce1(function(val, item) {
            return val + item;
        }, 0)
        console.log('result=', result);
```

6.模拟实现reduce和reduceRight函数

模拟实现reduceRight：

```
Array.prototype.reduceRight1 = function(reduceFunc, initValue) {
            for (let i = this.length - 1; i >= 0; i--) {
                initValue = reduceFunc(initValue, this[i], i, this);
            //这里如果需要索引和源数组，那么可以进行传递
            }
            return initValue;
        }
        let arr = [1, 2, 3];
        let result = arr.reduceRight1(function(val, item) {
            return val + item;
        }, 0)
        console.log('result=', result);
```

注意：循环条件的设定。

7.rest参数的优点

使用rest参数代替了arguments；那么，**rest参数与arguments 相比有什么样优势**？

使用arguments对数据进行排序：

```
function sortFunc() {

        return Array.prototype.slice.call(arguments).sort()
    }

    console.log(sortFunc(23, 12, 67));
```

使用 rest对数据进行排序：

```
function sortFunc(...values) {
        return values.sort()
    }
    console.log(sortFunc(23, 12, 67));
```

8.rest参数注意问题

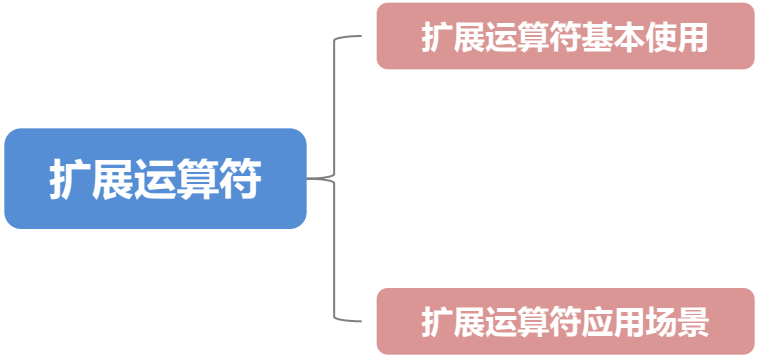
- rest参数之后不能再有其他的参数，也就是说**rest参数只能是最后一个参数，否则会报错。**

```
function test(a, ...b, c) {
            console.log(a);
            console.log(b);
            console.log(c);
        }
        test(1, 23, 2, 5);
```

目录 Contents

扩展运算符

小节导学



1.扩展运算符基本使用

扩展运算符的**表现形式**：**三个点（...）**,可将一个**数组转换为用逗号分隔的序列**。

案例：

将两个数组合并为一个数组；

传统做法：

```
let arr1 = [1, 2, 3];
let arr2 = [4, 5, 6];
let arr3 = [].concat(arr1, arr2);
console.log(arr3);
```

使用扩展运算符:

```
let arr1 = [1, 2, 3];
let arr2 = [4, 5, 6];
let arr3 = [...arr1, ...arr2];
console.log(arr3);
```

2.扩展运算符应用场景

◆ 代替数组中的apply方法

用Math.max来计算数组中的最大值（ES5的写法）：

```
let arr = [12, 23, 11, 56];

console.log(Math.max.apply(null, arr));
```

说明：也可使用Math.max.apply来实现；

使用扩展运算符实现：

```
let arr = [12, 23, 11, 56];

console.log(Math.max(...arr));
```

说明：由于JavaScript不提供求数组中最大值的函数，只能将数组转换成一个参数的列表，再进行相应的求值。

2.扩展运算符应用场景

◆ 用于函数调用

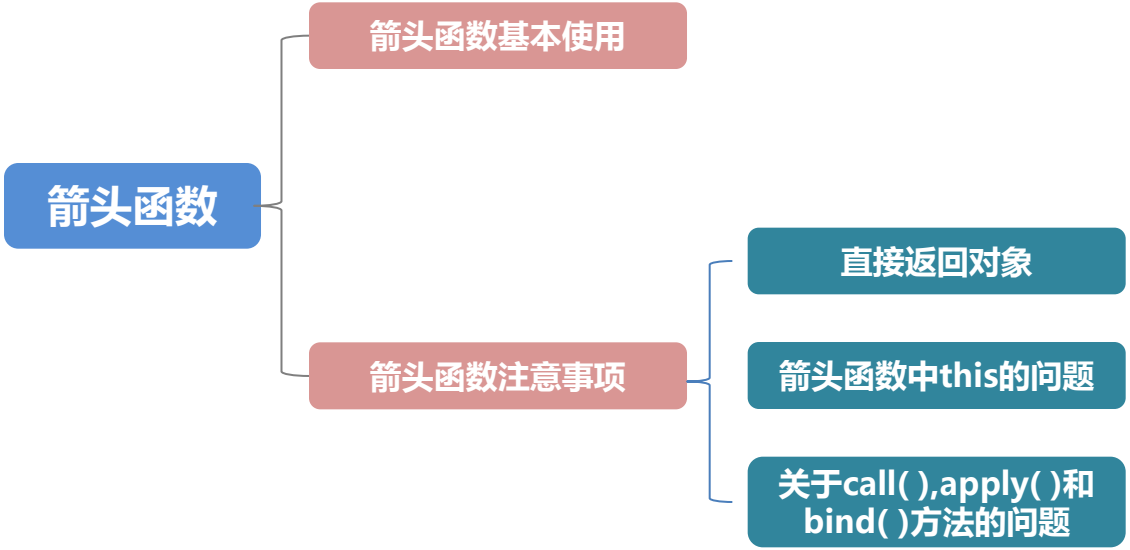
在函数调用时，需要进行参数的传递，在某些情况下，**通过扩展运算符，更有利于参数的传递。**

```
function test(num1, num2) {

            return num1 + num2;

           }
       let array = [23, 56];

       console.log(test(...array));
```

目录 Contents

箭头函数

小节导学



1.箭头函数基本使用

使用传统的方式定义一个函数：

```
let f = function(x, y) {
    return x + y;
}
console.log(f(3, 6));
```

使用箭头函数：

```
let f = (x, y) => {
       return x + y
   };
   console.log(f(9, 8));
```

1.箭头函数基本使用

如果参数只有一个，可以省略小括号:

```
let f = num => {
            return num / 2;
        }
        console.log(f(6));
```

如果没有参数，只需要写一对小括号即可：

```
let f = () => {
            return 9 / 3;
        }
        console.log(f());
```

函数体中只有一条语句，此时可省略大括号：

```
let f = (x, y) => x + y;

console.log(f(3, 6));
```

2.箭头函数的注意事项

- 直接返回对象
- 箭头函数中this的问题
- 关于call( ),apply( )和bind( )方法的问题

2.箭头函数的注意事项

◆ 直接返回对象

箭头函数**直接返回**一个对象:

```
let f = (id, name) => ({
      id: id,
      userName: name
  });
  console.log(f(1, 'zs'));
```

采用如下写法，可达到相同的结果:

```
let f = (id, name) => {
        return {
            id: id,
            userName: name
        }
    };
    console.log(f(1, 'zs'));
```

2.箭头函数的注意事项

◆ 箭头函数中this的问题

箭头函数中没有this，若在箭头函数中使用this，实际上使用的是外层代码块的this。

箭头函数不会创建自己的this，它只会从自己**作用域链的上一层继承this**。

通俗理解：

找出定义箭头函数的上下文（即包含箭头函数最近的函数或者是对象），那么上下文所处的父上下文即为this。

```
let person = {
     userName: 'zhangsan',
     getUserName() {
         return () => {
             console.log(this.userName);
         }
     }
 }
 person.getUserName()();
```

2.箭头函数的注意事项

- **关于call( ),apply( )和bind( )方法的问题**

由于箭头函数没有自己的this，因此，**不能使用 call( )、apply( )、bind( )等方法来改变this的指向**。

```
let adder = {
            base: 1,
            add: function(a) {
                let f = v => v + this.base;
                let b = {
                    base: 2
                };
                return f.call(b, a);
            }
        };
        console.log(adder.add(1))
```

输出结果为：2

一样的在线教育，不一样的教学品质