函数

每个函数都是Function类型的实例
不推荐用
`let sum = new Function("num1", "num2", "return num1 + num2");`
方式来定义函数。因为会被解释2次
第一次是将它当作常规ECMAScript 代码，第二次是解释传给构造函数的字符串

箭头函数

ES6新增 =>

- 1个参数 括号可有可无
- 没有参数/多个参数，需要括号
- 有大括号则是函数体，需要return，没return默认返回undefined
- 没有大括号则是一个表达式/赋值操作，会自动返回值

注意点:

- 箭头函数没有this指向，this指向最近的作用域
- 箭头函数没有prototype，不能当构造函数
- 数不能使用 arguments、super 和 new.target

```
// 以下两种写法都有效
let double = (x) => { return 2 * x; };
let triple = x => { return 3 * x; };
// 没有参数需要括号
let getRandom = () => { return Math.random(); };
// 多个参数需要括号
let sum = (a, b) => { return a + b; };
// 无效的写法：
let multiply = a, b => { return a * b; };
```

函数名

没啥好说的，函数就是对象，函数名就是对象名
函数可以有多个函数名，就跟对象可以有多个指针指向自己

```
function foo() {
    console.log('I am foo')
}

let anotherFoo = foo
foo();  // I am foo
anotherFoo();   // I am foo
```

所有函数都有name属性，只读
函数没有名称 name为空字符串
函数如果是Function构造函数创建的，标识成"anonymous"

```
function foo() {}
let bar = function() {};
let baz = () => {};
console.log(foo.name); // foo
console.log(bar.name); // bar
console.log(baz.name); // baz
console.log((() => {}).name); //（空字符串）
console.log((new Function()).name); // anonymous
```

如果函数是获取函数、设置函数，或者用bind()实例化，标识符前面会加上前缀get、set、bound

```
function foo() {}
console.log(foo.bind(null).name); // bound foo
let dog = {
 years: 1,
 get age() {
 return this.years;
 },
 set age(newAge) {
 this.years = newAge;
 }
}
let propertyDescriptor = Object.getOwnPropertyDescriptor(dog, 'age');
console.log(propertyDescriptor.get.name); // get age
console.log(propertyDescriptor.set.name); // set age
```

理解参数

ES函数的参数在内部表现为一个数组。
函数并不关心数组包含什么。
使用function 定义的函数（不包含箭头函数），arguments对象，取得传进来的每一个参数值
不能重载

```
function doAdd() {
 if (arguments.length === 1) {
 console.log(arguments[0] + 10);
 } else if (arguments.length === 2) {
 console.log(arguments[0] + arguments[1]);
 }
}
doAdd(10); // 20
doAdd(30, 20); // 50
```

箭头函数中的参数

箭头函数没有 arguments
但是可以通过包装函数实现

```
function foo() {
    let bar = () => {
        console.log(arguments[0]); // 5
    };
    bar();
}
foo(5);
```

没有重载

ES没有重载，如果定义了两个重名函数，后面的会把前面的覆盖。

默认参数值

ES5.1 及以前，实现默认参数的方法

```
function makeKing(name) {
 name = (typeof name !== 'undefined') ? name : 'Henry';
 return `King ${name} VIII`;
}
```

ES6以后可以显式定义默认参数了

```
function makeKing(name = 'Henry') {
 return `King ${name} VIII`;
}
```

传undefined 等于没传参

```
function makeKing(name = 'Henry', numerals = 'VIII') {
 return `King ${name} ${numerals}`;
}
console.log(makeKing()); // 'King Henry VIII'
console.log(makeKing('Louis')); // 'King Louis VIII'
console.log(makeKing(undefined, 'VI')); // 'King Henry VI'
```

arguments对象不反应参数的默认值，只反映传参

默认参数作用域与暂时性死区

给参数定义默认值，实际上跟使用let一样

- 前面定义的参数不能引用后面定义的
- 参数不能引用定义在函数体内的变量

```
function makeKing(name = 'Henry', numerals = name) {
 return `King ${name} ${numerals}`;
```

```
}
console.log(makeKing()); // King Henry Henry

// 调用时不传第一个参数会报错
function makeKing(name = numerals, numerals = 'VIII') {
 return `King ${name} ${numerals}`;
}

// 调用时不传第二个参数会报错
function makeKing(name = 'Henry', numerals = defaultNumeral) {
 let defaultNumeral = 'VIII';
 return `King ${name} ${numerals}`;
}
```

参数扩展与收集

ES6新增 扩展操作符

扩展参数

可以把可迭代对象拆分，单独传入

```
let values = [1, 2, 3, 4];
function getSum() {
    let sum = 0;
    for (let i = 0; i < arguments.length; ++i) {
        sum += arguments[i];
    }
    return sum;
}

console.log(getSum.apply(null, values)); // 10
console.log(getSum(...values)); // 10
```

收集参数

可以把不同长度的独立参数组合成一个数组。

- 只能作为最后一个参数
- 箭头函数不支持arguments对象，但是支持收集参数

```
function getSum(...values) {
 // 顺序累加 values 中的所有值
 // 初始值的总和为 0
 return values.reduce((x, y) => x + y, 0);
}
console.log(getSum(1,2,3)); // 6
```