箭头函数是匿名函数，ES5匿名函数的语法糖；但又增加了ES5所没有的一些优点，接下来我们一起来看一看箭头

```
//ES5
var tt = function tt() {
return 55 + 99;
 };

//ES6
var tt = () => 55 +99

//是不是一对比，写法的差异就看出来了
```

ES6 增加了箭头函数：

```
let func = value => value;
```

相当于：

```
let func = function (value) {
   return value;
};
```

如果需要给函数传入多个参数：

```
let func = (value, num) => value * num;
```

如果函数的代码块需要多条语句：

```
let func = (value, num) => {
   return value * num
};
```

如果需要直接返回一个对象：

```
let func = (value, num) => ({total: value * num});
```

与变量解构结合：

```
let func = ({value, num}) => ({total: value * num})

// 使用
var result = func({
   value: 10,
   num: 10
})

console.log(result); // {total: 100}
```

以上是箭头函数的用法，了解了箭头函数的用法之后呢，我们一起来进一步的看一看箭头函数，到底和我们的普通函数有什么区别，好处在哪?

1.没有 this

箭头函数没有 this，所以需要通过查找作用域链来确定 this 的值。

这就意味着如果箭头函数被非箭头函数包含，this 绑定的就是最近一层非箭头函数的 this。

模拟一个实际开发中的例子：

我们的需求是点击一个按钮，改变该按钮的背景色。

为了方便开发，我们抽离一个 Button 组件，当需要使用的时候，直接：

```
// 传入元素 id 值即可绑定该元素点击时改变背景色的事件

new Button("button")
```

HTML 代码如下：

```
<button id="button">点击变色</button>
```

JavaScript 代码如下：

```
function Button(id) {
   this.element = document.querySelector("#" + id);
   this.bindEvent();
}

Button.prototype.bindEvent = function() {
        this.element.addEventListener("click", this.setBgColor, false);
};

Button.prototype.setBgColor = function() {
   this.element.style.backgroundColor = '#1abc9c'
};

var button = new Button("button");
```

看着好像没有问题，结果却是报错 Uncaught TypeError: Cannot read property 'style' of undefined

这是因为当使用 addEventListener() 为一个元素注册事件的时候，事件函数里的 this 值是该元素的引用。

所以如果我们在 setBgColor 中 console.log(this)，this 指向的是按钮元素，那 this.element 就是 undefined，报错自然就理所当然了。

也许你会问，既然 this 都指向了按钮元素，那我们直接修改 setBgColor 函数为：

```
Button.prototype.setBgColor = function() {
   this.style.backgroundColor = '#1abc9c'
};
```

不就可以解决这个问题了?

确实可以这样做，但是在实际的开发中，我们可能会在 setBgColor 中还调用其他的函数，比如写成这种：

```
Button.prototype.setBgColor = function() {
   this.setElementColor();
   this.setOtherElementColor();
};
```

所以我们还是希望 setBgColor 中的 this 是指向实例对象的，这样就可以调用其他的函数。

利用 ES5，我们一般会这样做：

```
Button.prototype.bindEvent = function() {
   this.element.addEventListener("click", this.setBgColor.bind(this), false);
};
```

为避免 addEventListener 的影响，使用 bind 强制绑定 setBgColor() 的 this 为实例对象

使用 ES6，我们可以更好的解决这个问题：

```
Button.prototype.bindEvent = function() {
   this.element.addEventListener("click", event => this.setBgColor(event), false);
};
```

由于箭头函数没有 this，所以会向外层查找 this 的值，即 bindEvent 中的 this，此时 this 指向实例对象，所以可以正确的调用 this.setBgColor 方法， 而 this.setBgColor 中的 this 也会正确指向实例对象。

在这里再额外提一点，就是注意 bindEvent 和 setBgColor 在这里使用的是普通函数的形式，而非箭头函数，如果我们改成箭头函数，会导致函数里的 this 指向 window 对象 (非严格模式下)。

最后，因为箭头函数没有 this，所以也不能用 call()、apply()、bind() 这些方法改变 this 的指向，可以看一个例子：

```
var value = 1;
var result = (() => this.value).bind({value: 2})();
console.log(result); // 1
```

2. 没有 arguments

箭头函数没有自己的 arguments 对象，这不一定是件坏事，因为箭头函数可以访问外围函数的 arguments 对象：

```
function constant() {
   return () => arguments[0]
}

var result = constant(1);
console.log(result()); // 1
```

那如果我们就是要访问箭头函数的参数呢?

你可以通过命名参数或者 rest 参数的形式访问参数:

```
let nums = (...nums) => nums;
```

3. 不能通过 new 关键字调用

JavaScript 函数有两个内部方法：[[Call]] 和 [[Construct]]。

当通过 new 调用函数时，执行 [[Construct]] 方法，创建一个实例对象，然后再执行函数体，将 this 绑定到实例上。

当直接调用的时候，执行 [[Call]] 方法，直接执行函数体。

箭头函数并没有 [[Construct]] 方法，不能被用作构造函数，如果通过 new 的方式调用，会报错。

```
var Foo = () => {};
var foo = new Foo(); // TypeError: Foo is not a constructor
```

4. 没有原型

由于不能使用 new 调用箭头函数，所以也没有构建原型的需求，于是箭头函数也不存在 prototype 这个属性。

5. 没有 super

连原型都没有，自然也不能通过 super 来访问原型的属性，所以箭头函数也是没有 super 的，不过跟 this、arguments 一样，这些值由外围最近一层非箭头函数决定。

在出现了箭头函数之后是不是就可以舍弃原来的普通函数了呢？答案一定是否定的，一些情况下我们最好不要去进行箭头函数的操作，那都有哪些情况呢?

在对象上定义函数

先来看下面这段代码

```
var obj = {
array: [1, 2, 3],
sum: () => {
console.log(this === window); // => true
 return this.array.reduce((result, item) => result + item);
}
};
// Throws "TypeError: Cannot read property 'reduce' of undefined"
obj.sum();
```

sum方法定义在obj对象上，当调用的时候我们发现抛出了一个TypeError，因为函数中的this是window对象，所以this.array也就是undefined。原因也很简单，相信只要了解过es6 箭头函数的都知道

箭头函数没有它自己的this值，箭头函数内的this值继承自外围作用域

解决方法也很简单，就是不用呗。这里可以用es6里函数表达式的简洁语法，在这种情况下，this值就取决于函数的调用方式了。

```
var obj = {
array: [1, 2, 3],
 sum() {
 console.log(this === obj); // => true
return this.array.reduce((result, item) => result + item);
}
};
obj.sum(); // => 6
```

通过object.method()语法调用的方法使用非箭头函数定义，这些函数需要从调用者的作用域中获取一个有意义的this值。

2.在原型上定义函数

在对象原型上定义函数也是遵循着一样的规则

```
function Person (pName) {
this.pName = pName;
}
Person.prototype.sayName = () => {
console.log(this === window); // => true
 return this.pName;
}
var person = new Person('wdg');
person.sayName(); // => undefined
```

使用function函数表达式

```
function Person (pName) {
this.pName = pName;
}
Person.prototype.sayName = function () {
console.log(this === person); // => true
return this.pName;
 }
var person = new Person('wdg');
person.sayName(); // => wdg
```

3.动态上下文中的回调函数

this是js中非常强大的特点，他让函数可以根据其调用方式动态的改变上下文，然后箭头函数直接在声明时就绑定了this对象，所以不再是动态的。

在客户端，在dom元素上绑定事件监听函数是非常普遍的行为，在dom事件被触发时，回调函数中的this指向该dom,可当我们使用箭头函数时:

```
var button = document.getElementById('myButton');
 button.addEventListener('click', () => {
console.log(this === window); // => true
 this.innerHTML = 'Clicked button';
});
```

因为这个回调的箭头函数是在全局上下文中被定义的，所以他的this是window。所以当this是由目标对象决定时，我们应该使用函数表达式:

```
var button = document.getElementById('myButton');
 button.addEventListener('click', function() {
console.log(this === button); // => true
this.innerHTML = 'Clicked button';
});
```

4.构造函数中

在构造函数中，this指向新创建的对象实例

```
this instanceOf MyFunction === true
```

需要注意的是，构造函数不能使用箭头函数，如果这样做会抛出异常

```
var Person = (name) => {
this.name = name;
 }
// Uncaught TypeError: Person is not a constructor
 var person = new Person('wdg');
```

理论上来说也是不能这么做的，因为箭头函数在创建时this对象就绑定了，更不会指向对象实例。

5.太简短的函数

箭头函数可以让语句写的非常的简洁，但是一个真实的项目，一般由多个开发者共同协作完成，就算由单人完成，后期也并不一定是同一个人维护，箭头函数有时候并不会让人很好的理解，比如

```
let multiply = (a, b) => b === undefined ? b => a * b : a * b;
let double = multiply(2);
 double(3); // => 6
multiply(2, 3); // =>6
```

这个函数的作用就是当只有一个参数a时，返回接受一个参数b返回a*b的函数，接收两个参数时直接返回乘积，这个函数可以很好的工作并且看起很简洁，但是从第一眼看去并不是很好理解。

为了让这个函数更好的让人理解，我们可以为这个箭头函数加一对花括号，并加上return语句，或者直接使用函数表达式:

```
function multiply(a, b) {
if (b === undefined) {
 return function (b) {
return a * b;
}
 }
return a * b;
 }
 let double = multiply(2);
double(3); // => 6
multiply(2, 3);  // => 6
```

总结

最后，关于箭头函数，引用 MDN 的介绍就是：箭头函数表达式的语法比函数表达式更短，并且不绑定自己的this，arguments，super。

毫无疑问，箭头函数带来了很多便利。恰当的使用箭头函数可以让我们避免使用早期的.bind()函数或者需要固定上下文的地方并且让代码更加简洁。

箭头函数也有一些不便利的地方。我们在需要动态上下文的地方不能使用箭头函数:定义需要动态上下文的函数，构造函数，需要this对象作为目标的回调函数以及用箭头函数难以理解的语句。在其他情况下，请尽情的使用箭头函数。