## Map的第二个参数与原生js的bind函数

#### Map函数的基本用法

map()方法返回一个新数组，数组中的元素为原始数组元素调用函数处理后的值。

Map()方法按照原始数组元素顺序依次处理元素。

需要注意的有：

Map()方法不会改变原始数组，会返回一个新数组。

Map()方法并不会对空数组进行检测。

基本的语法为：

array.map(function(*currentValue*,*index*,*array*){},thisDirect);

有两个参数，第一个是一个回调函数（必选），这个函数的参数为：当前的值、当前值的索引值，当前数组本身。第二个参数为前面的回调function的中的this指向，这是个可选的参数。

当然第一个回调函数可以是任意一个函数指针（或者是一个匿名函数），只不过在map中会自动为这个函数带入上面的三个参数。

下面两个典型的使用方式：

第一种：

var new\_arr = [1,2,3,4].map(function(currentValue){

return *arguments*[0] / 2; // 为了体现出第一个参数这样写的

});

*console*.log(new\_arr); // (4) [0.5, 1, 1.5, 2]

第二种：

var new\_arr = [4,9,16,25].map(Math.sqrt); // 注意是函数指针，不要犯低级错误

*console*.log(new\_arr); // (4) [2, 3, 4, 5]

因为经过查阅资料和我的鄙陋项目实践，对第二个参数（回调函数中this的指向）的使用和说明的实例很少，因此顺便探究一下，不过在这之前有必要说一下bind()方法。

#### 原生Js中bind()函数

先来看一个易犯错误：

var altwrite = document.write;

altwrite("hello");//报错，因为altwrite方法是在window上进行调用的，而window上是没有该方法的

最开始看到这里我甚至都没有注意到我的错误。因为我并不知道在document.write这个函数中也可能存在着this。

将它转换成另一个明显的例子:

var a = 200;

var obj = {

a:100,

sayA:function(){

*console*.log(*this*.a);

}

}

obj.sayA(); // 100

var win = obj.sayA;

win(); // 200 相当于window.win();

这回懂了吧。

var win = obj.sayA; 其实这句话只是引用了函数的指针

#### 那么如何才能改变this的指向，比如让在全局作用域中的win()也能达到和obj.sayA()同样的效果呢?

在这里只需要Bind方法来改变A的指向即可。

var a = 200;

var obj = {

a:100,

sayA:function(){

*console*.log(*this*.a);

}

}

obj.sayA(); // 100

var win = obj.sayA.bind(obj); // 通过bind函数改变了this的指向

win(); // 100 即使是在全局作用域中也依旧能保证this指向obj

如果需要改变this的指向，还有很多原生方法，比如call、apply。这将会在之后的博文中讨论，在此不多赘述。

#### 三、Map()方法的第二个参数，回调函数中this的指向

再后头说说这个第二个参数。

array.map(function() {}, this) 的作用实际上和 array.map(function() {}.bind(this)) 是一样的。map的第二个参数就是给第一个参数bind一个对象，这样在第一个参数里面就可以用this代替第二个参数。