# 方法一

function toThousands(num) {

var result = [ ], counter = 0;

num = (num || 0).toString().split('');

for (var i = num.length - 1; i >= 0; i--) {

counter++;

result.unshift(num[i]);

if (!(counter % 3) && i != 0) { result.unshift(','); }

}

return result.join('');

}

方法一的执行过程就是把数字转换成字符串后，打散为数组，再从末尾开始，逐个把数组中的元素插入到新数组（result）的开头。每插入一个元素，counter就计一次数（加1），当counter为3的倍数时，就插入一个逗号，但是要注意开头（i为0时）不需要逗号。最后通过调用新数组的join方法得出结果。

方法一比较清晰易懂，也在项目中用了一段时间。但是直觉告诉我，它的性能并不好。

# 方法二

function toThousands(num) {

var result = '', counter = 0;

num = (num || 0).toString();

for (var i = num.length - 1; i >= 0; i--) {

counter++;

result = num.charAt(i) + result;

if (!(counter % 3) && i != 0) { result = ',' + result; }

}

return result;

}

方法二是方法一的改良版，不把字符串打散为数组，始终对字符串操作。

# 方法三

function toThousands(num) {

var num = (num || 0).toString(), re = /\d{3}$/, result = '';

while ( re.test(num) ) {

result = RegExp.lastMatch + result;

if (num !== RegExp.lastMatch) {

result = ',' + result;

num = RegExp.leftContext;

} else {

num = '';

break;

}

}

if (num) { result = num + result; }

return result;

}

方法三是完全不同的算法，通过正则表达式循环匹配末尾的三个数字，每匹配一次，就把逗号和匹配到的内容插入到结果字符串的开头，然后把匹配目标（num）赋值为还没匹配的内容（RegExp.leftContext）。此外，还要注意：

1.如果数字的位数是3的倍数时，最后一次匹配到的内容肯定是三个数字，但是最前面的三个数字前不需要加逗号；

2.如果数字的位数不是3的倍数，那num变量最后肯定会剩下1到2个数字，循环过后，要把剩余的数字插入到结果字符串的开头。

虽然方法三减少了循环次数（一次循环处理三个字符），但由于用到了正则表达式，一定程度上增加了消耗。

# 方法四

function toThousands(num) {

var num = (num || 0).toString(), result = '';

while (num.length > 3) {

result = ',' + num.slice(-3) + result;

num = num.slice(0, num.length - 3);

}

if (num) { result = num + result; }

return result;

}

事实上，截取末尾三个字符的功能可以通过字符串类型的slice、substr或substring方法做到。这样就可以避免使用正则表达式。

# 方法五

function toThousands(num) {

var num = (num || 0).toString(), temp = num.length % 3;

switch (temp) {

case 1:

num = '00' + num;

break;

case 2:

num = '0' + num;

break;

}

return num.match(/\d{3}/g).join(',').replace(/^0+/, '');

}

先把数字的位数补足为3的倍数，通过正则表达式，将其切割成每三个数字一个分组，再通过join方法添加逗号，最后还要把补的0移除。

# 方法六

function toThousands(num) {

return (num || 0).toString().replace(/(\d)(?=(?:\d{3})+$)/g, '$1,');

}

一直觉得这个格式化是可以通过一条正则表达式替换做出来的，但是需要用到断言等写法，无奈自己对这部分不太熟。Google了一下，还真找到了这么一条正则表达式，这估计是代码最短的实现。

测试结果

执行5000次消耗的时间（ms）

数字 方法一 方法二 方法三 方法四 方法五 方法六

1 4 1 3 1 14 2

10 14 1 3 0 7 2

100 12 1 2 4 5 3

1000 13 2 3 2 9 5

10000 21 4 3 1 6 3

100000 21 3 2 1 5 6

方法一和方法二的强烈对比表明，字符串操作的效率比数组操作的效率要高得多；方法六的测试结果告诉我们，代码长短跟性能高低没有关系。方法四的综合性能是最好的（但为何num为100的时候，性能有所降低呢，这个实在不解），主要原因是：

1.对比方法一、二，每次操作3个字符而不是1个字符，减少循环次数；

2.对比方法三、五、六，没有使用正则表达式，减少了消耗。

最后，我选择了方法四作为最终的优化方案。各位读者如有更好的实现方法或改良意见，可以发表评论。