setTimeout()是js中的一类重要函数，将一段代码延迟一定时间并异步执行。但是这个函数经常不听话。在实践中，可能经常有人碰到类似下面的这种情况：

for (var i = 1; i <= 2; i++) {

setTimeout(function() { alert(i) }, 100);

}

 （setTimeout同时开始执行，然后同时等100ms，这时候for早完事了，所以…by wt）

我们期望的结果是，先隔100毫秒弹出1，再隔100毫秒弹出2。但是跑起来后，alert的两次内容都是数字3，而且紧挨着输出，并不是自己所期望的先1后2。有一种很基础的面试题是，如何合理改动代码，使它返回期望的结果？

其实很简单。在stackoverflow上早有大神解释了，可以参考[这个链接](http://stackoverflow.com/questions/5226285/settimeout-in-for-loop-does-not-print-consecutive-values" \t "_blank)

https://stackoverflow.com/questions/5226285/settimeout-in-for-loop-does-not-print-consecutive-values

答案翻译成中文如下（并做了部分修改方便理解）：

---------------------------------------------------------------------------------

你要为每个定时器处理函数创建不同的“i”变量副本。比如这样：

function doSetTimeout(i) {

setTimeout(function() { alert(i); }, 100);

}

for (var i = 1; i <= 2; ++i)

doSetTimeout(i);

 （这个也是setTimeout同时启动，但是每一个都有不一样的i by wt）

如果你不做这样的事情 （这种方法在实际上也会有其他变种），每个定时器处理函数就会共享同一作用域里的同一变量"i"。当循环完成时的"i"是多少？是3！这里通过定义一个函数来实现中介的作用，从而创建了变量的值的副本。由于setTimeout()是在该副本的上下文中调用的，所以它每次都有自己的私有的"i"以供使用。

但是随着时间的推移，这些代码的效果显得有些混乱是显而易见的事实，因为设立一些时间间隔相同的连续的setTimeout()将导致所有延时处理程序同时被调用。了解设置timer（对setTimeout()的调用）几乎不消耗时间是很重要的。也就是说，告诉系统“请在1000毫秒后调用此函数”将会被立即返回，因为在timer队列中安装延时请求的过程非常快。

因此，如果有一串连续的延时请求（比如我答案中的代码），而且每一个时间延迟值是相同的，那么一旦经过足够的时间，所有延时处理程序将一个接一个快速连续调用。

如果你需要的是在固定时间间隔调用的处理程序，你可以使用setInterval()，行为非常类似setTimeout()，但每经过一定时间就运行一次。或者你可以调用以“时间值乘以迭代计数器”为时间间隔的setTimeout。也就是说，修改我刚才的示例代码：

function doScaledTimeout(i) {

setTimeout(function() {

alert(i);

}, i \* 5000);

}

（100毫秒超时，效果不会很明显，所以我设置的数字高达5000）

“i”值乘以基础延迟值，所以循环5次将导致分别延迟5秒，10秒，15秒，20秒，和25秒。