# BOM

浏览器对象模型

\* - 在BOM中为我们提供了一组对象，通过这些对象我们可以完成对浏览器的操作

\* - window

\* - 代表浏览器的窗口

\* - navigator

\* - 代表浏览器的信息，通过该对象可以来识别不同的浏览器

\* - history

\* - 代表浏览器的历史记录，通过该对象可以控制浏览器向前或向后翻页

\* - location

\* - 代表浏览器的地址栏的信息，通过该对象可以控制浏览器跳转页面

\* - screen

\* - 代表用户的屏幕信息

\* - 我们的BOM对象都是作为window对象的属性保存的，我们可以通过window对象来访问，由于是window属性，属于全局变量，前面加不加window都行

## navigator

\* - 通过该对象可以判断浏览器的信息

\* - 由于历史原因Navigator中的大部分属性都已经不能用来识别浏览器

\* - 但是该对象中还有一个属性叫做userAgent（用户代理），还可以用来识别不同的浏览器

\* userAgent实际上就是一个字符串，不同的浏览器会有不同的userAgent，所以我们可以通过该属性来判断浏览器的版本

火狐

Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0

chrome

Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36

IE8

Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)

IE9

Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)

IE10

Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)

利用以上各个浏览器的userAgent来判断浏览器版本

var ua=navigator.userAgent

if(/firefox/i.test(ua)){

alert("你是火狐~~~");

}else if(/chrome/i.test(ua)){

alert("你是chrome浏览器~~~");

}else if(/msie/i.test(ua)){

alert("你是IE");

}

IE11

Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; rv:11.0) like Gecko

在IE11的UserAgent中已经完全找不到微软的印记了

可以通过一些IE浏览器中独有的对象来判断浏览器是否是IE

\* 比如：ActiveXObject

\* - 这个对象是一个构造函数，它是IE中用于发送AJAX请求的对象，是IE中独有的

所以可以测试浏览器是否有ActiveXObject来判断是否是IE，

if(ActiveXObject){

alert("你是IE~~~");

}

由于其他浏览器没有这个对象，输出时会报错，前面需要加window来使其输出undefined而不报错（这个对象本身是全局变量，加不加window都没关系），undefined转换为布尔值是false

if(window.ActiveXObject){

alert("你是IE~~~");

}

但IE11为了不和其他浏览器有区别，判断时将window.ActiveXObject转换为false（IE其他版本都转换为true），还可以利用in 来判断

if("ActiveXObject" in window){

alert("你是IE~~~");

}

这样判断各个浏览器版本就完成了

if(/firefox/i.test(ua)){

alert("你是火狐~~~");

}else if(/chrome/i.test(ua)){

alert("你是chrome浏览器~~~");

}else if(/msie/i.test(ua)){

alert("你是IE");

}else if("ActiveXObject" in window){

alert("你是IE11"); //到这里IE版本只有IE 11了

}

## history

\* - 该对象代表用户的历史记录，通过它可以对历史记录进行操作

\* 由于隐私的原因该对象只能控制浏览器向前或向后翻页

\* - 它只可以访问当次（当前会话：从打开到关闭浏览器）的历史记录

属性：

length

\* - 获取当次访问的历史记录的数量

方法：

back()

\* - 可以回退到上一个页面

\* - 功能相当于浏览器的后退按钮

forward()

\* - 可以跳转到下一个页面

\* - 功能相当于浏览器的前进按钮

go()

\* - 可以跳转到指定的页面

\* - 需要一个整数作为参数，将会跳转指定数量的页面，正整数向前跳转，负整数向后跳转

## location

\* - location表示的是浏览器的地址栏的信息

\* - 如果直接输出location则会返回当前页面的地址

\* - 如果直接修改location的值，则会使浏览器跳转到指定的页面

location = "test01.html";

方法：

assign()

\* - 可以用来跳转到指定的页面

\* - 他需要一个路径作为参数，将会跳转到该路径

location.assign("test02.html");

\* - 和直接修改location是一样的

\* - 修改location和使用assign()来跳转页面都会生成历史记录，可以使用回退按钮回退

replace()

\* - 使用replace()也可以跳转到其他的页面，同样需要一个地址作为参数 location.replace("test02.html");

\* 用法和assign()一样

\* - 不会生成历史记录，不可以回退

reload()

\* - 可以用来刷新当前网页，相当于浏览器的刷新按钮

location.reload();

\* - 在该方法中可以传递一个true，如果传了true则会强制清空缓存刷新网页 location.reload(true);

# 定时调用

\* - 定时调用指每间隔一段时间就调用指定的函数一次

setInterval()

\* - 用来开启一个定时调用

\* - 参数：

\* 回调函数，该函数将会每间隔一段时间被调用一次

\* 时间，函数调用相隔的时间（毫秒）

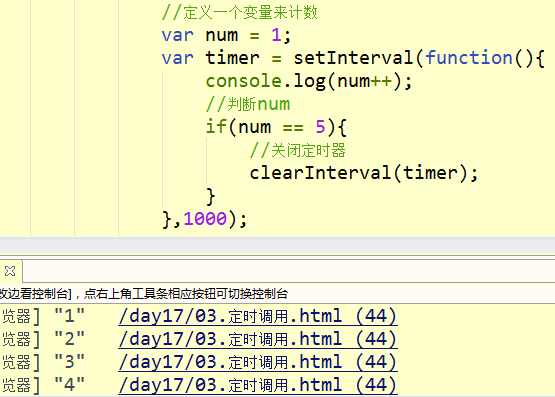
\* - 返回值：

\* 会返回一个number类型的值作为返回值，这个number就是定时器的标识

clearInterval()

\* - 可以用来关闭一个定时器

\* - 需要一个定时器的标识作为参数，需要用它来指定要关闭的定时器



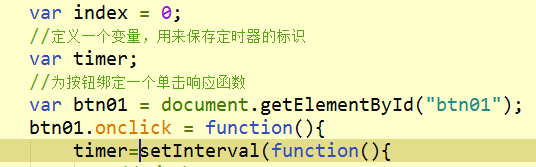
每隔1s向控制台输出一个num

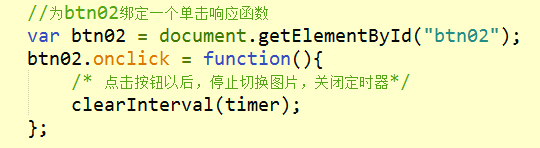
## 图片自动切换



点击按钮图片开始自动切换，还需要点击另一个按钮以后，停止切换图片，关闭定时器，而两个按钮的函数作用域没有关系，互不影响，如果在btn01中定义

var timer=setInterval()，在btn02中 不能访问到timer，需要将timer定义到全局中，只定义不赋值：var timer；这样在btn01中使timer=setInterval();在btn02中就可以访问到





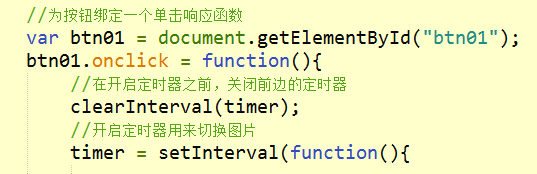
现在每点击一次按钮，就为图片开启一个定时器，点击几次就开启几次，就会导致同一个元素上会有多个定时器在执行动画效果，而且由于每次开启定时器都会覆盖timer，导致我们只能关闭最后一个定时器，其他的都不能关闭

注意：我们为一个元素开启定时器时，一般都需要将该元素上的其他的定时器关闭，避免互相干扰

clearInterval()

在停止一个定时器时，如果标识有效则会停止定时器，如果标识无效则什么也不做，不会报错

所以在开启定时器前，要将该元素上的其他定时器关闭



## 键盘控制div移动修改版

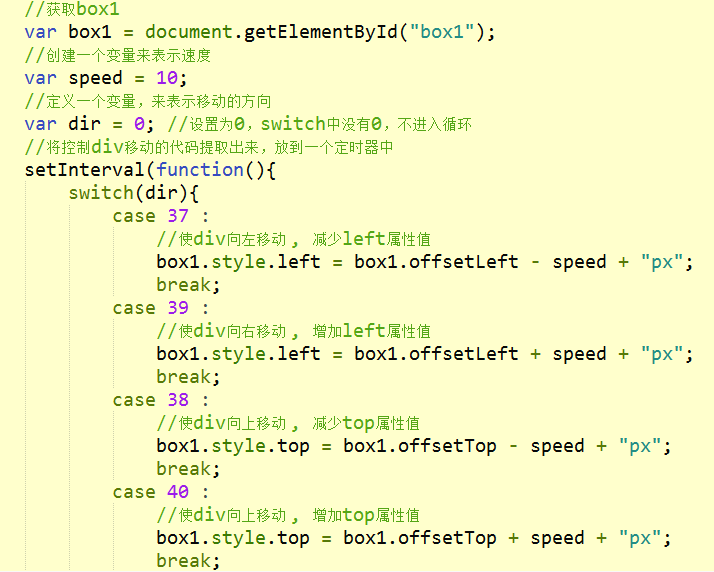
使div可以根据不同的按键向不同的方向移动

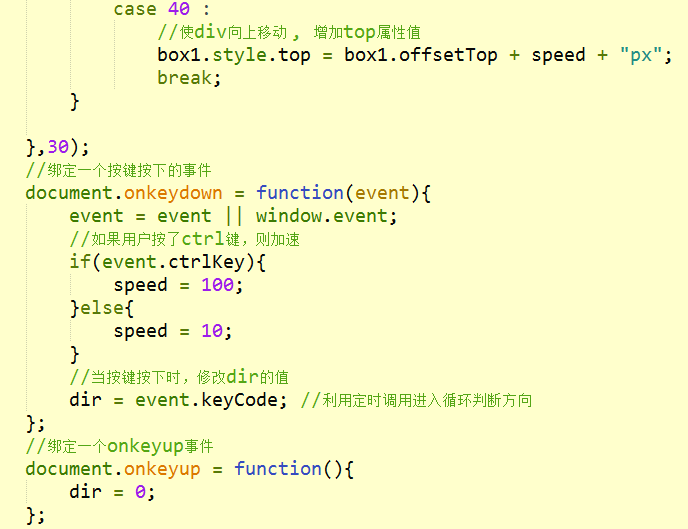
\* 控制元素移动，两个要素

\* 1.移动的方向

\* 2.移动的速度

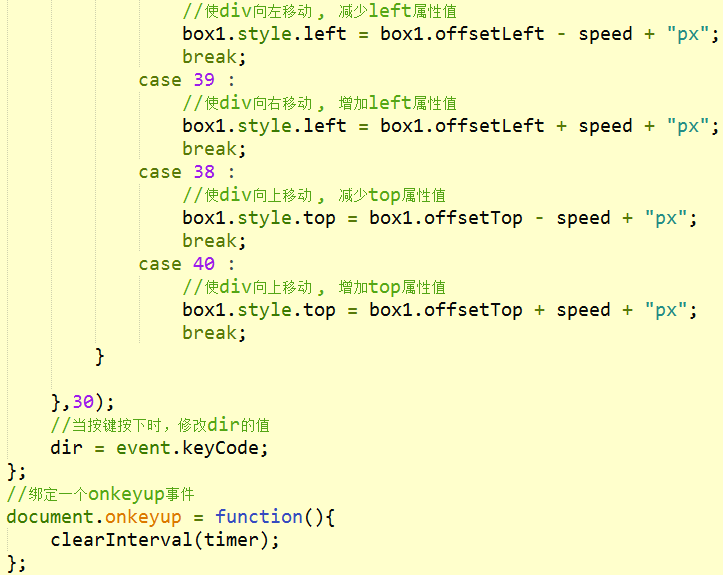
以前使用键盘控制移动时，在switch中既控制速度，又控制方向，就相当于骑自行车双手既控制速度，又控制方向，会出现问题，要把方向和速度分开处理





或者





# 延时调用

\* 延时调用指函数不马上执行，而是过一段时间以后再执行

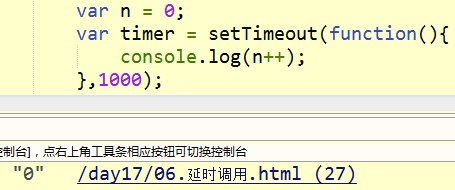
\* 使用setTimeout()来设置延时调用， setTimeout()和setInterval()用法一样，也是需要两个参数：

\* 1.回调函数

\* 2.毫秒数

\* 延时调用中的函数，只会执行一次；而定时调用会执行多次

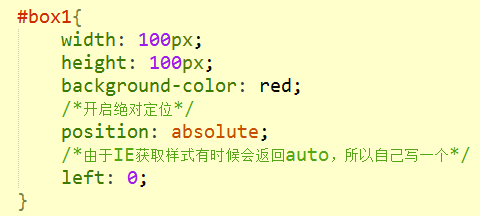
\* 定时调用和延时调用本质上没有区别，我们可以通过setTimeout()实现setInterval()的功能，也可以通过setInterval()实现setTimeout()的功能

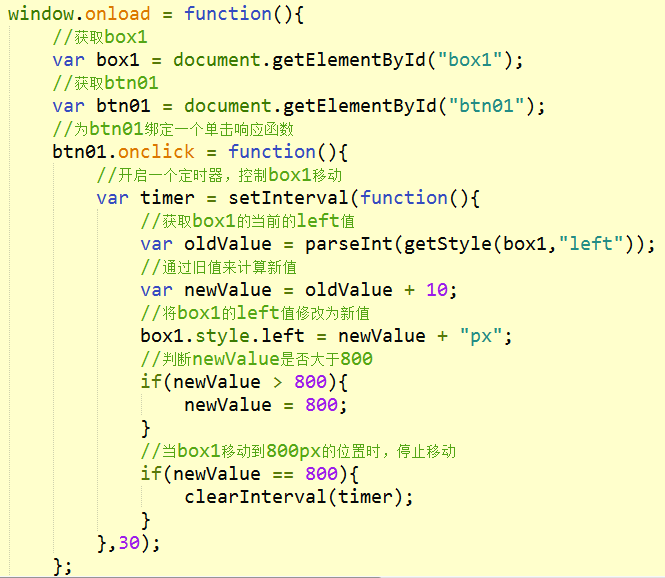


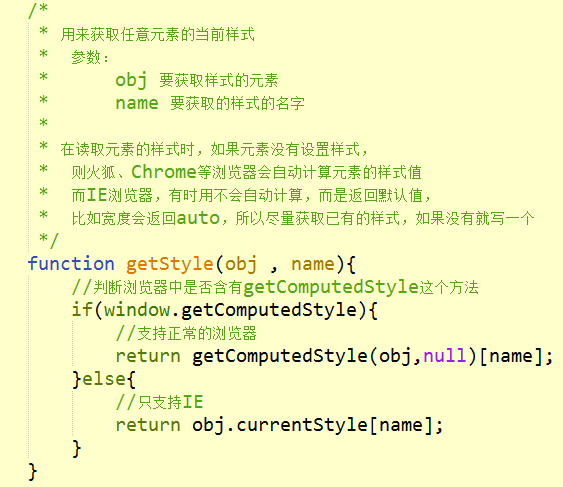
1s后执行，而且只执行一次

# 定时器的应用

## 点击按钮以后，使box1向右移动

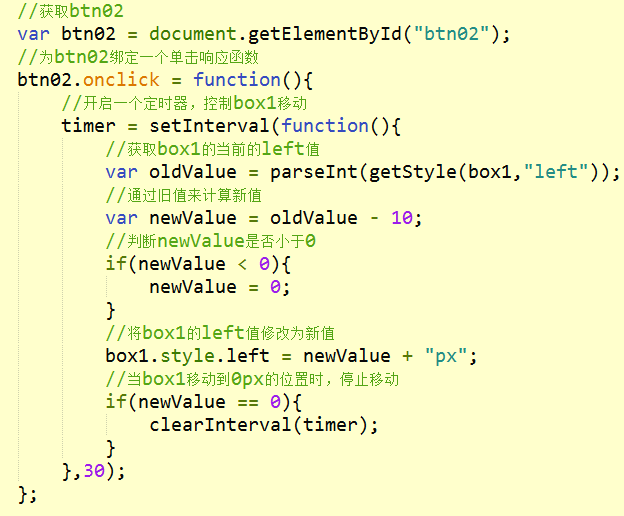




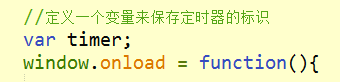


## 点击按钮以后，使box1向左移动

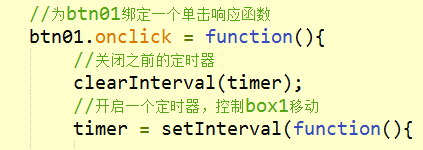
这个功能和使box1向右移动是类似的

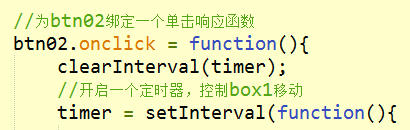


但当前给同一个元素设置了两个定时器，当点击向左移动时，元素向左移动，而此时点击向右移动时，由于向左右移动都是10px，元素会在当前位置做左右移动的运动，不会偏离当前位置，此时需要开启一个定时器时，要关闭另一个定时器，而且又是两个局部作用域，所以还需要在全局定义一个timer



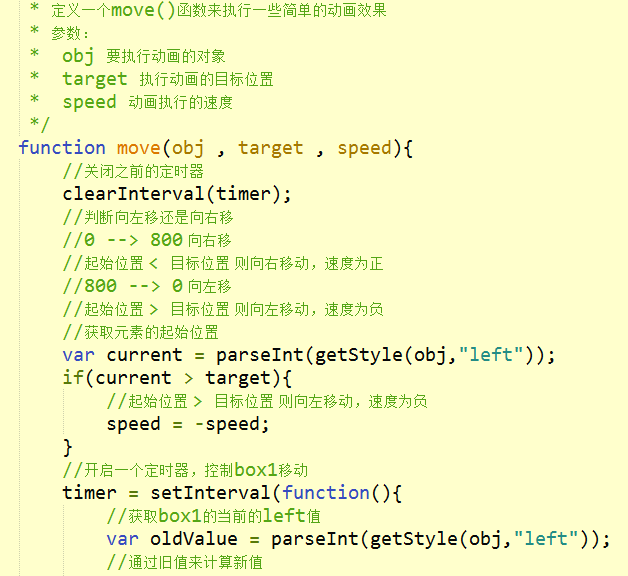
而且在每一个开启定时器前面都要关闭另一个定时器

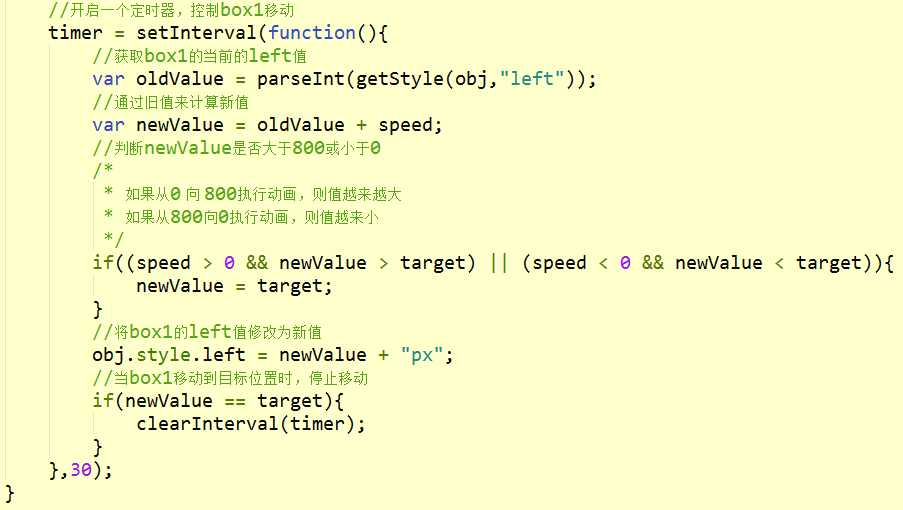




而且这两个功能类似，所以可以定义一个函数来表示移动

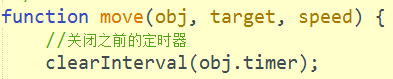
## move函数





这个函数对同一个元素可以实现移动，点击向左就向左移动，点击向右移动后，向左停止，开始向右移动，但如果给不同元素设置后，移动另一个元素后，前一个元素会停止移动，因为每次开启一个定时器前都要停止一个定时器，而且这些定时器的标识都一样（在全局作用域中声明，都可以访问），所以需要给不同元素设置不同的定时器标识（不在全局作用域中声明）

一般都会将定时器的标识作为执行动画对象的属性保存，这样可以确保只有当前对象能访问到定时器

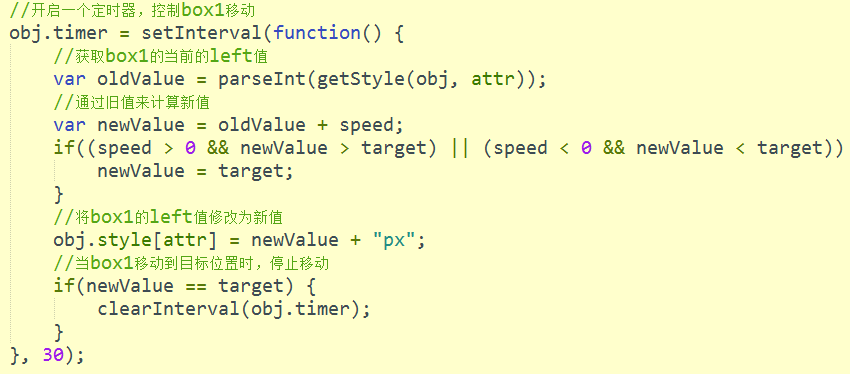


将所有timer都改为obj.timer；

但现在只能向左后向右移动，所以还需要一个参数，来控制样式

attr 执行动画时要修改的属性，可以是左右上下、宽度高度…



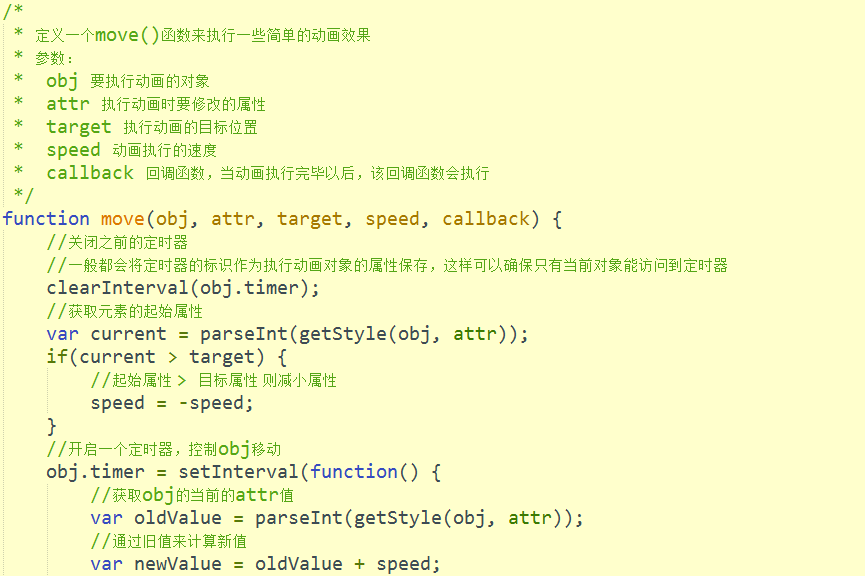


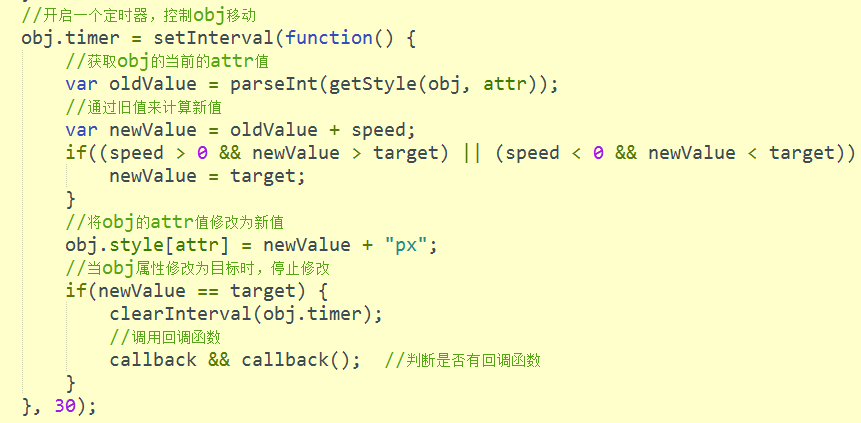
还可以传一个回调函数

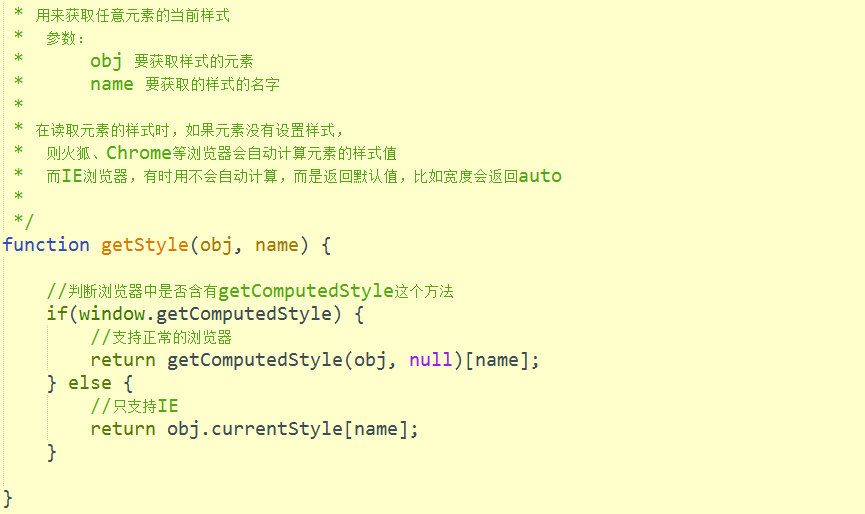
callback 回调函数，当动画执行完毕以后，该回调函数会执行

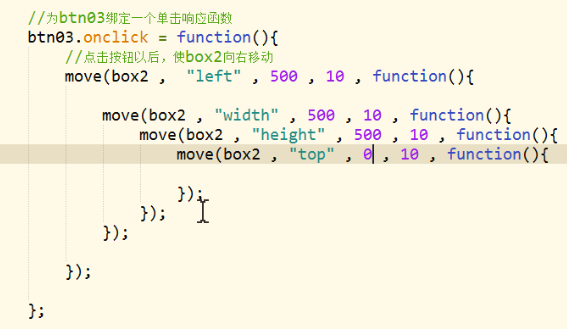
这样整个动画效果函数就完成了

## 动画效果函数









利用回调函数，可以完成一组动画效果：先向右移动到500px出，然后宽度改为500px，再高度改为500px，再向上移动到0px处