**8开发插件**

**在插件中使用$别名**

在编写jQuery插件时，必须假设jQuery库已经加载到了页面中。可是，我们不能假设$别名一定可用。第3章曾经讲过，$.noConflict()方法就是用于让渡这个快捷方式使用权的。考虑到这一点，我们自定义的插件就应该始终都使用jQuery这个名字来调用jQuery方法，或者也可以在内部定义一个$别名。

对于代码比较长的插件来说，很多开发人员都觉得不能使用$别名会导致代码难以理解。为了解决这个问题，我们可以在插件的作用域内定义这个快捷方式，方法就是定义一个函数并马上调用它。这种定义并立即调用函数的语法通常被称为**立即调用的函数表达式**

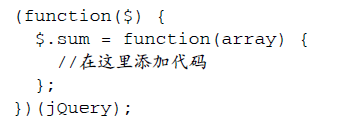
(function($) {

//在这里添加代码

})(jQuery);

jQuery内置的某些功能是通过全局函数提供的。所谓全局函数，实际上就是jQuery对象的方法，但从实践的角度上看，它们是位于jQuery命名空间内部的函数

要向jQuery的命名空间中添加一个函数，只需将这个新函数指定为jQuery对象的一个属性即可，参见代码清单8-1。

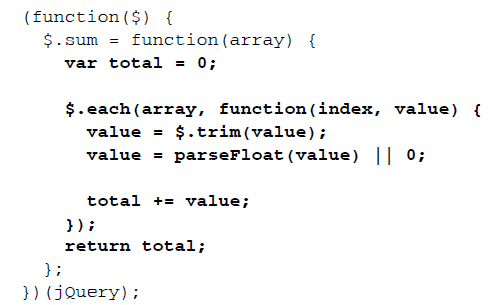


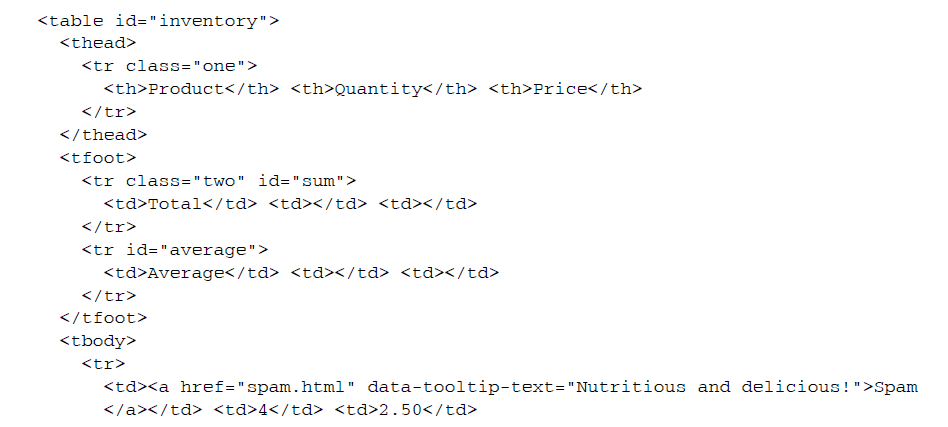
jQuery内置的某些功能是通过全局函数提供的。所谓全局函数，实际上就是jQuery对象的方法，但从实践的角度上看，它们是位于jQuery命名空间内部的函数

要向jQuery的命名空间中添加一个函数，只需将这个新函数指定为jQuery对象的一个属性即可，参见代码清单8-1。

这个求和函数接受一个数组作为参数，然后把数组的值加在一起，最后返回结果。

**代码清单8-2**



为了测试这个新插件，我们创建了一个表格，其中包含库存的食品 

**代码清单8-3**

**$(document).ready(function() {**

**var $inventory = $('#inventory tbody');**

**var quantities = $inventory.find('td:nth-child(2)')**

**.map(function(index, qty) {**

**return $(qty).text();**

**}).get();**

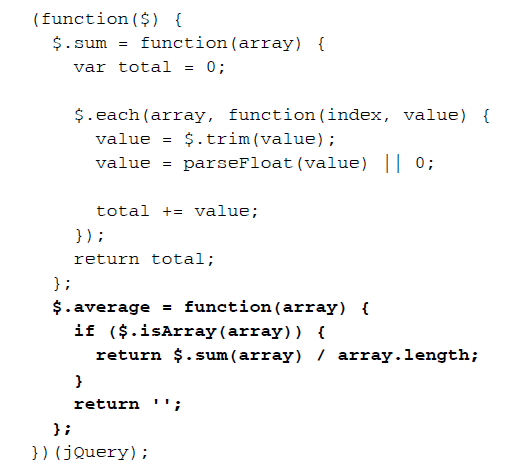
**var sum = $.sum(quantities);**

**$('#sum').find('td:nth-child(2)').text(sum);**

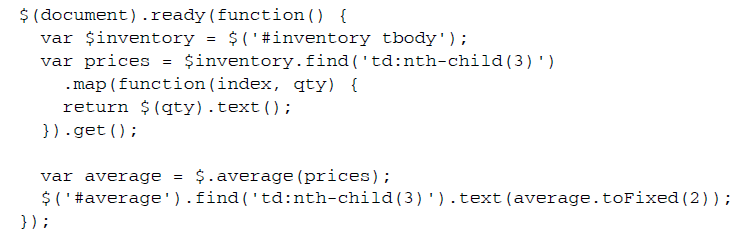
**});**

**代码清单8-4**

添加一个用于计算数值数组平均值的函数



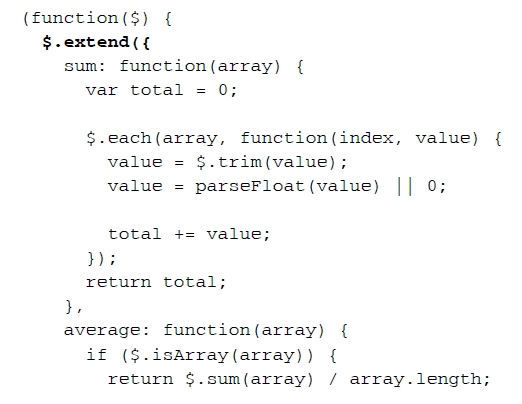
**代码清单8-5**



**扩展全局jQuery对象**

利用$.extend()函数，还可以通过另外一种语法来定义全局函数

**代码清单8-6**

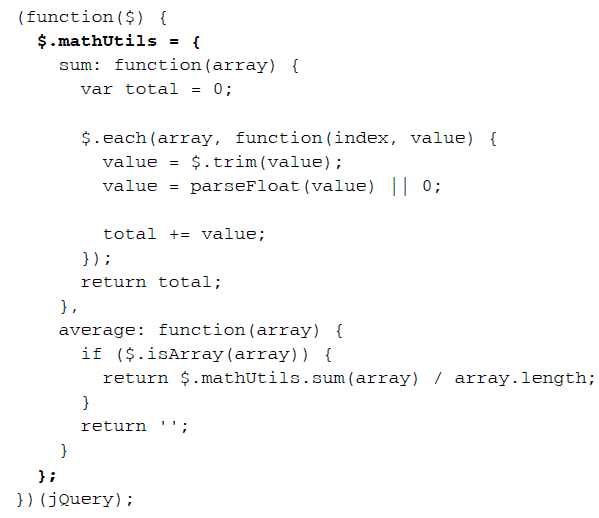


extend(dest,src1,src2,src3...);

扩展target对象，即将后面传入对象的属性添加到这个对象中

$.extend(src)  
省略dest参数，该方法就是将src合并到jquery的全局对象中去

为了避免冲突，最好的办法是把属于一个插件的全局函数都封装到一个对象中



这个模式的本质是为所有的全局函数又创建了一个命名空间，叫做jQuery.mathUtils。

虽然我们还称它们为全局函数，但实际上它们已经成了mathUtils对象的方法

，在调用它们时就必须得加上插件的名字

$.mathUtils.sum(sum);

$.mathUtils.average(average);

**添加jQuery 对象方法**

，添加全局函数需要以新方法来扩展jQuery对象。添加实例方法也与此类似，但扩展的却是**jQuery.fn**对象：

**jQuery.fn.myMethod = function() {**

**alert('Nothing happens.');**

**};**

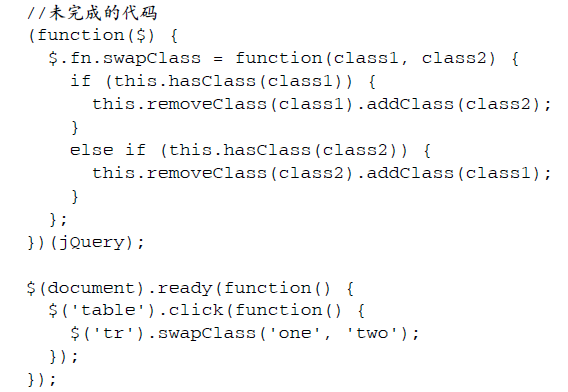
然后，就可以在使用任何选择符表达式之后调用这个新方法了：

**$('div').myMethod();**

**对象方法的上下文**

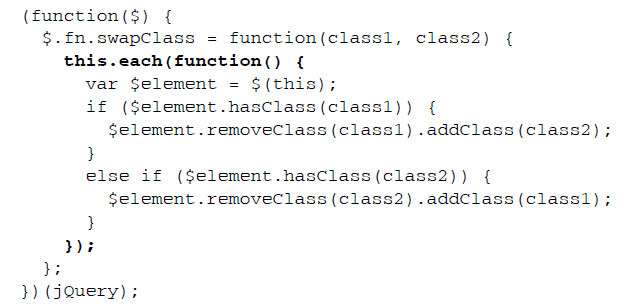
在任何插件方法内部，关键字this**引用的都是当前的jQuery对象。**因而，可以在this上面调用任何内置的jQuery方法，或者提取它包含的DOM节点并操作该节点。

**代码清单8-8**



**隐式迭代**

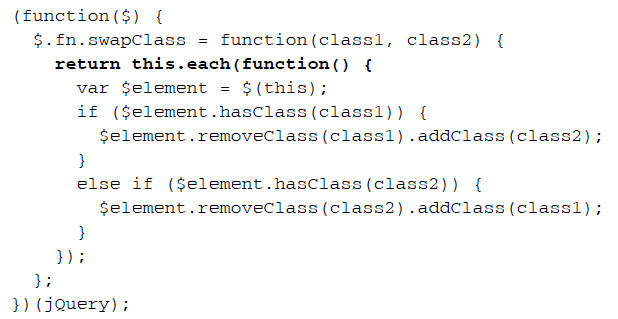
我们在此调用的.hasClass()只会检查匹配的第一个元素。换句话说，我们应该独立检查和操作每一个元素。**代码清单8-9**



**方法连缀**

除了隐式迭代之外，jQuery用户也应该能够正常使用连缀行为。因而，我们必须在所有插件方法中返回一个jQuery对象，除非相应的方法明显用于取得不同的信息。返回的jQuery对象通常就是this所引用的对象。

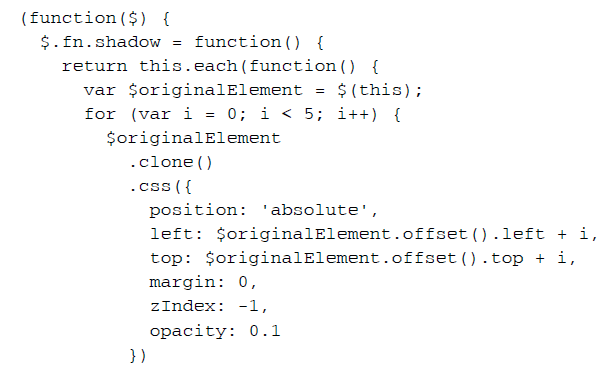
**代码清单8-10**

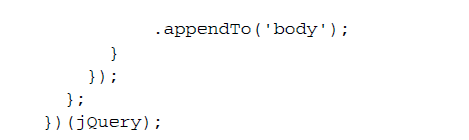


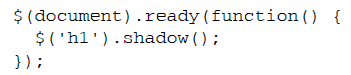
**提供灵活的方法参数**

一个为元素块加投影的插件方法

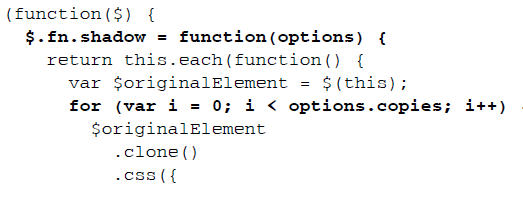
**代码清单8-11**

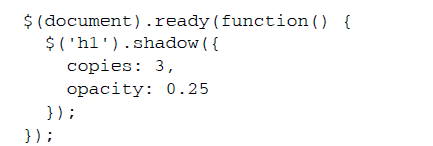






接下来，我们就赋予这个插件方法一些灵活性。这个方法的操作取决于一些用户可能想要修改值。可以把这些值提取出来作为参数，以便用户根据需要修改。

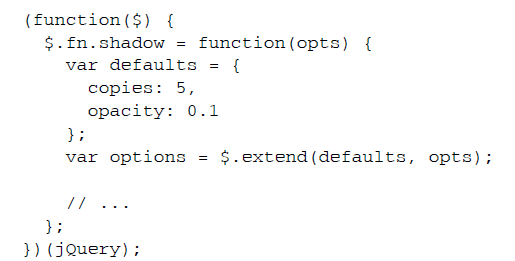




**默认参数值**

随着方法的参数逐渐增多，始终指定每个参数并不是必须的。此时，一组合理的默认值可以增强插件接口的易用性。

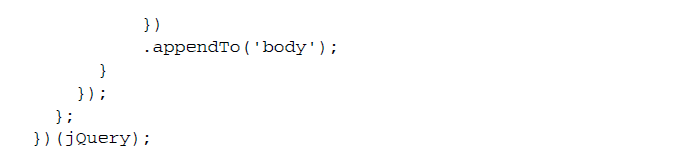
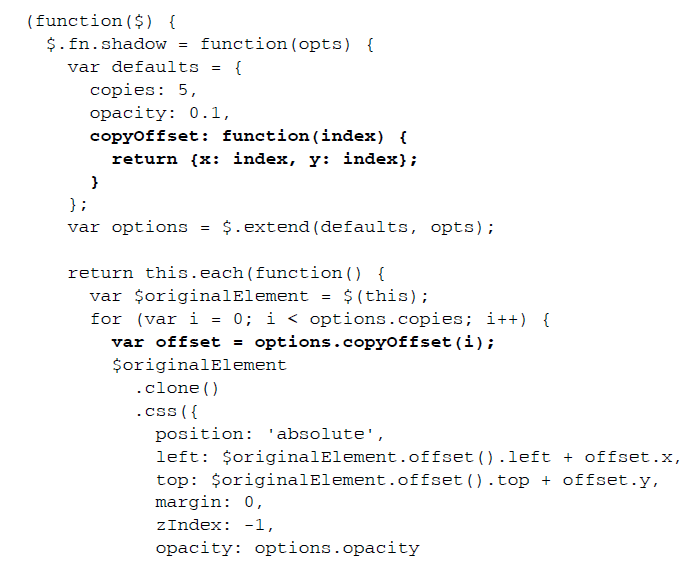
**代码清单8-13**



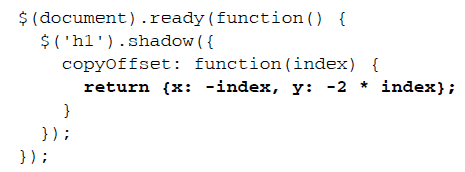
**回调函数**

要在方法中使用回调函数，需要接受一个函数对象作为参数，然后在方法中适当的位置上调用该函数。

可以扩展前面定义的文本投影方法，让用户能够自定义投影相对于文本的位置，**代码清单8-14**



用户可以在两个方向上指定负值偏移量：



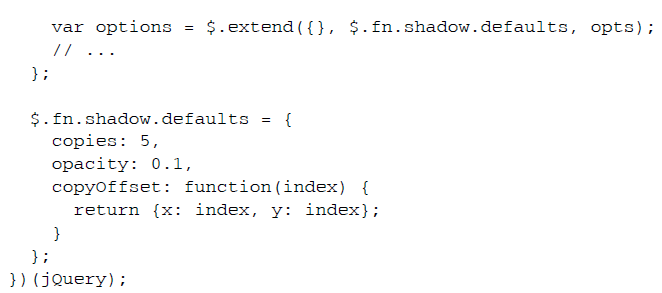
**可定制的默认值**

通过为方法参数设定合理的默认值，能够显著改善用户使用插件的体验。但是，到底什么默认值合理有时候也很难说。如果用户脚本会多次调用我们的插件，每次调用都要传递一组不同于默认值的参数，那么通过定制默认值就可以减少很多需要编写的代码量。

要支持默认值的可定制，需要把它们从方法定义中移出，然后放到外部代码可以访问的地方

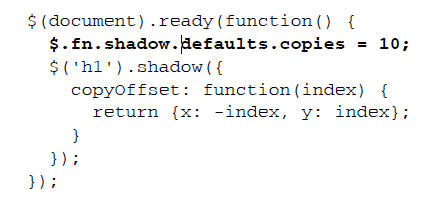
**如代码清单8-15所示**。





默认值被放在了投影插件的命名空间里，可以通过$.fn.shadow.defaults直接引用。由于现在所有对.shadow()的调用都要重用defaults对象，因此不能让$.extend()修改它。我们就在此将一个空对象（{}）作为$.extend()的第一个参数，让这个新对象成为被修改的目标

于是，使用我们插件的代码就可以修改默认值了，修改之后的值可以被所有后续对.shadow()的调用共享。而且，在调用方法时仍然可以传递选项。



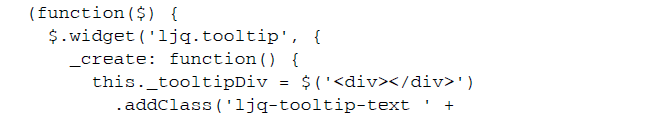
**使用jQuery UI 部件工厂创建插件**

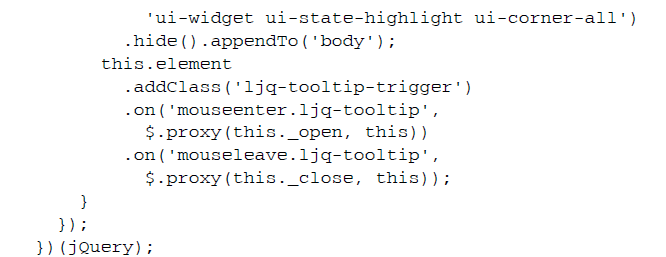
我们自己要编写的插件会**创建新的用户界面元素，**通常最好以扩展jQuery UI库的方式来实现。这些部件对JavaScript开发人员而言，有一组非常统一的API，因而学习起来非常简单。

jQueryUI库的核心包含了一个工厂方法，叫$.widget()，使用这个方法可以确保我们的代码达到所有jQuery UI部件用户认可的API标准。

**创建部件**

在下面的例子中，我们要编写一个插件为元素添加自定义的提示条。为了创建这个提示条，需要为页面中的每个元素创建一个<div>容器，然后在鼠标悬停在元素上时，把这个容器放在相应元素的旁边。**代码清单8-16**

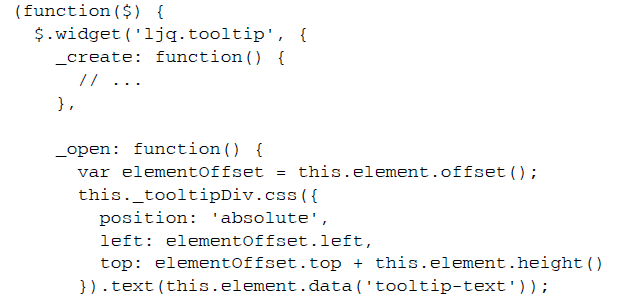


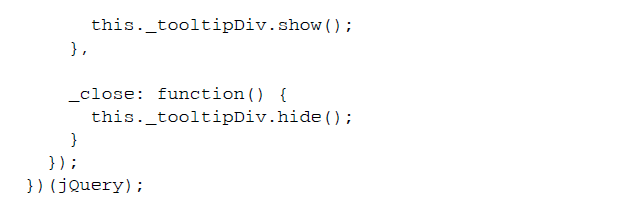


每次调用$.widget()都会通过部件工厂创建一个jQuery UI插件。这个函数接受部件的名称和一个包含部件属性的对象作为参数。部件名称必须带命名空间，在这里使用ljq作为命名空间，使用tooltip作为插件名称。这样，在jQuery项目中就可以通过.tooltip()调用这个插件了。

在这个函数的上下文中，**this**引用的是当前部件实例，可以通过它为部件添加任何想要的属性。另外，部件实例本身也有一些预定义的属性可以为我们提供便利；特别地，**this.element**中保存着一个jQuery对象，这个对象指向最初选择的元素。

**代码清单8-17**





**.\_open()**和**.\_close()**为自定义函数，前面下划线表示私有

在打开（open）提示条时，使用CSS定位将它放到合适的位置然后显示它；而在关闭（close）提示条时，隐藏它即可。

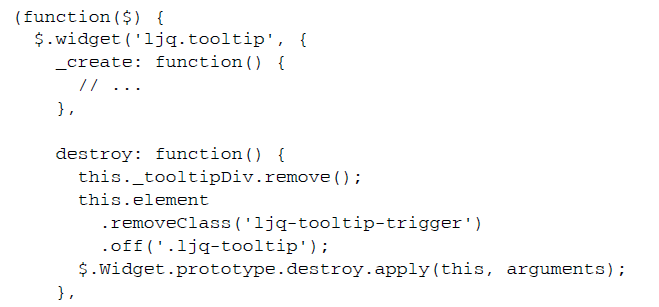
新的HTML5标准允许在普通的元素标签里，嵌入类似data-\*的属性，来实现一些简单数据的存取。它的数量不受限制，并且也能由javascript动态修改，也支持CSS选择器进行样式设置。这使得data属性特别灵活，也非常强大。有了这样的属性我们能够更加有序直观的进行数据预设或存储。

我们用到了**.data()**方法，这个方法可以用来取得和设置与任何元素相关的数据。利用了这个方法读取HTML5数据属性的能力

**销毁部件**

我们知道，部件工厂可以创建新的jQuery方法。在我们的例子中这个方法就是.tooltip()，不传递任何参数调用它，可以为一组元素应用提示条部件。不过，除了单纯的应用提示条，这个方法还可以做其他很多事情。这时候，需要给这个方法传入一个字符串参数，以便调用适当的**子方法**。

其中一个内置的子方法是destroy。调用.tooltip('destroy')就可以从页面中删除这个提示条部件。然后部件工厂会为我们完成大部分工作，但在通过\_create修改了文档（比如这里创建了用于保存提示条文本的<div>）的情况下，还要负责将其清理掉，参见**代码清单8-18**。

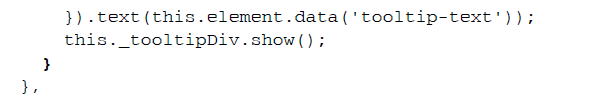
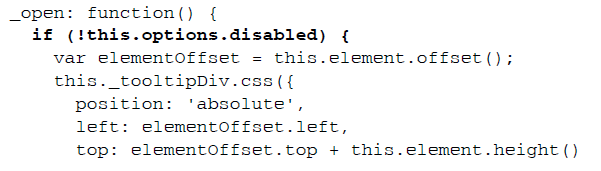


**$.Widget.prototype.destroy.apply(this, arguments);**

需要注意的是，destroy方法在jquery.ui.widget.js中是有默认实现的，而\_create没有实现。因此如果用自己的方法覆盖destroy，需要调用调用默认的实现的destroy前面并没有加下划线，这是因为它是一个可以通过.tooltip('destroy')调用的公有子方法。

**启用和禁用部件**

除了被完全销毁，也可以临时禁用然后再在将来重新启用部件。内置的enable和disable子方法可以帮我们实现部件的启用和禁用，方法是将**this.options.disabled**的值设置为true或false。要支持这两个子方法，我们要做的就是在对部件进行任何操作前先检查这个值，参见**代码清单8-19**

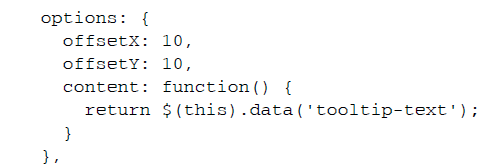


有了这个额外的检查之后，提示条就会在调用.tooltip('disable')之后暂停显示，而在

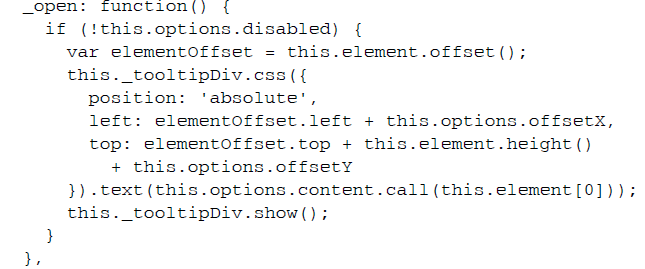
调用.tooltip('enable')之后恢复显示。

**接受部件选项**

考虑让部件可以定制。在前面编写.shadow()插件时，我们已经体验到为部件提供一组定制的默认设置，然后再让用户指定的选项覆盖默认设置是一种很友好的机制。在这个过程中，几乎所有工作都是由部件工厂执行的，而我们所要做的就是提供一个**options**属性，如**代码清单8-20**所示。



在我们的代码中，唯一需要用到options属性的就是.\_open()方法



语法：**call([thisObj[,arg1[, arg2[, [,.argN]]]]])**

参数 thisObj 可选项。将被用作当前对象的对象。 arg1, arg2, , argN 可选项。将被传递方法参数序列。

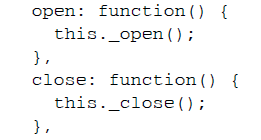
说明

call 方法可以用来代替另一个对象调用一个方法。call 方法可将一个函数的对象上下文从初始的上下文改变为由 thisObj 指定的新对象。

现在，不用传递参数也还是可以向页面中添加部件（比如，直接调用.tooltip()），但得到的都是默认行为。不过，提供选项则将覆盖默认行为，例如.tooltip({offsetX: -10,offsetY: 25})。部件工厂甚至可以让我们在部件实例化之后再修改选项，例如：.tooltip('option', 'offsetX', 20)。下次再访问这些选项时，就会取得新设置的值。

**添加子方法**

内置的子方法确实很方便，但有时候我们可能想为自己插件的用户提供更多“挂钩”。前面已经介绍过如何在部件中创建私有函数，实际上创建公有函数（也就是子方法）也一样，唯一的区别在于部件的属性名不以下划线开头。知道了这一点，要创建手工打开和关闭提示条的子方法就非常简单了，参见**代码清单8-22。**

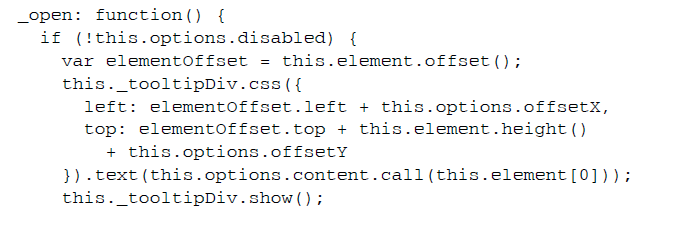


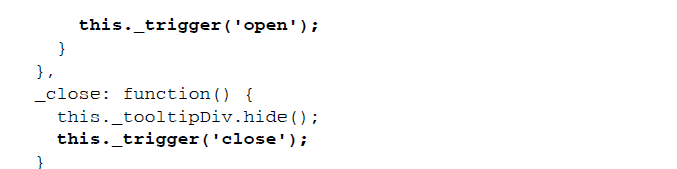
通过添加调用私有函数的子方法，现在就可以使用.tooltip('open')来打开提示条，使用.tooltip('close')来关闭提示条了。即使在子方法中什么也不返回，部件工厂也会替我们做很多工作，从而确保连缀语法可以正常工作。

**触发部件事件**

真正的好插件不仅自己扩展jQuery，而且还能为其他代码提供机制来扩展它。

**代码清单8-23**





在一个函数中调用**this.\_trigger()**可以让代码监听新的自定义事件。事件名字会加上部件名作为前缀，因而不必担心它会与其他事件冲突。因为这里在提示条的\_open函数中调用了this.\_trigger('open')，那么每次打开提示条的时候都会分派tooltipopen事件。而在这个元素上调用.on('tooltipopen')可以监听这个事件。