第 8 章事件与听众

MATLAB 面向对象程序设计可以通过事件的发生与否来实现各个对象之间的通信联系，并完成一些指定的动作，即用事件的发生来触发指定函数的调用过程。这一点类似于 Visual C++中的“消息传递（message based），事件驱动（event driven）”的运行机制。本章将主要介绍事件和听众的概念，以及如何使用事件模型来实现基于事件驱动的具有交互响应功能的程序设计过程。

8.1 事件与听众的概念

8.1.1 事件的概念及事件模型

什么是事件（event）呢？事件的含义很广泛。一般来说，从程序上可探测到的任何动作，都能够产生相应的事件，并可以将该事件的发生告知其他对象，来实现对象之间的通信。这里事件是用来标识发生的某件事情的。

在 MATLAB 面向对象程序设计中，事件表示发生在类实例（对象）中的某些变化或行为，这些变化包括：

（1）对类（对象）中数据的修改；

（2）方法的执行；

（3）查询或设置一个属性成员值；

（4）析构一个对象。

在 MATLAB 中，事件驱动和事件响应过程构成了一个完整的事件模型。组织一个事件模型的一般步骤如下。

（1）命名事件：在一个 handle 型类中声明一个标识符来代表事件，这个类就是产生事件的类，简称事件类，相应的实例称为事件对象；

（2）创建听众：在一个类中声明事件后，就可以关联一个相应的听众，一般用 addlistener 函数来定义听众；

（3）定义一个方法来确定何时触发这个事件，并通过 notify 函数对听众进行广播来告之该事件的发生；

（4）定义回调函数：当听众接收到某事件发生的信息时，就会去执行一个与之关联的回调函数，因此还要定义一个回调函数。

在定义听众时，既可将该听众与产生事件的对象生命周期进行绑定，也可将其限制在听众对象的生存期和作用范围内。

图 8-1 给出了一个简单事件模型的示意图，其中时钟类 Mclock 对象的事件块中有 Alarm 事件，因此它是产生事件的对象。用户通过调用方法 AlarmSetChange 类触发事件 Alarm，然后通过 Notify 函数将事件广播给听众 Listener1、Listener2 和 Listener3。由于听众 Listener1 的成员 EvenName 的值与事件 Alarm 名称匹配，于是 MATLAB 就去调用听众 Listener1 中通过 FunctionHandle 注册的回调函数 Fun1。

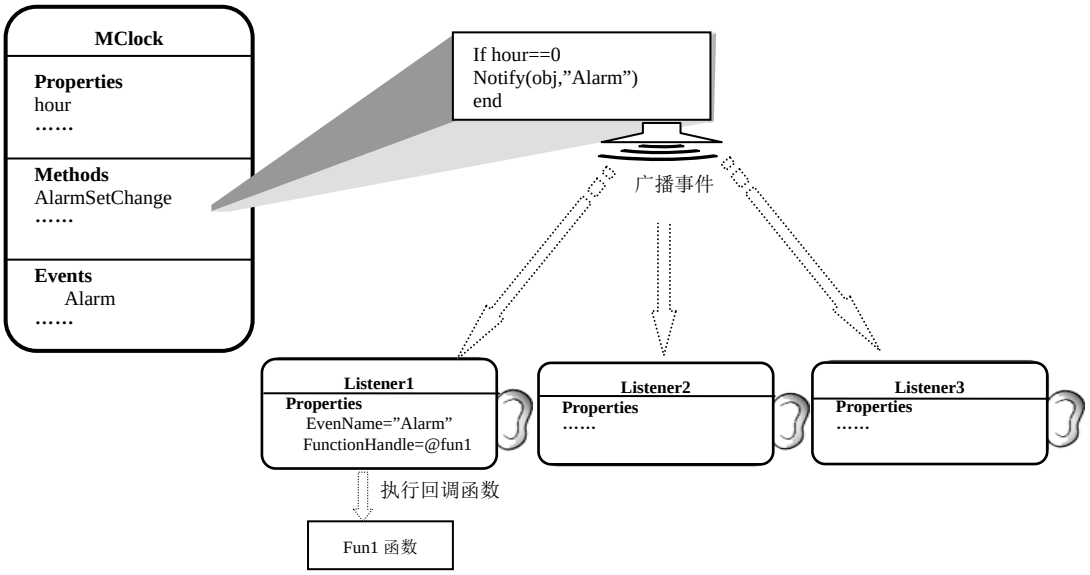

图 8-1　事件模型示意图

8.1.2　听众的概念

在上面的事件模型中，多次提到了听众，那么什么是听众（listener）呢？在 MATLAB 中，听众是用来接收事件发生消息的对象，是 event.listener 类型的对象。在 MATLAB 面向对象程序设计中，定义一个听众可以采用两种方法：一种方法是使用 handle 类中的 addlistener 函数；另外一种方法是直接调用 event.listener 类的构造函数。具体定义方法将在下节介绍。

一旦听众创建后，它会一直等待事件发生的消息，一旦关联的事件发生了，听众就会执行一个回调函数。这里 event.listener 类是 handle 的子类，表 8-1 给出了 event.listener 的全部新增特性。

表 8-1　event.listener 类属性成员表

属性成员	意义
Source	事件对象元胞数组
EventName	事件名
Callback	事件触发时执行的回调函数（这时需要将特性 Enabled 设置为 true）
Enabled	是否执行回调函数的开关（默认值是 true）
Recursive	回调函数是否递归调用（默认值是 false）

8.2　定义事件和听众

8.2.1　如何命名一个事件

在 MATLAB 面向对象程序设计中，一个事件要用一个名称来标识。在一个类的 events 块中声明一个标识符，便可定义该类中的一个事件的标识，我们把这样的标识也称为事件，其格式如下：

```
classdef 类名< handle
…
events
事件名
end
end
```

例如在下面的类 MClock 中定义了一个事件 Alarm，我们可以通过改变成员 Enable 的值

来触发这个事件。

```
classdefMClock< handle      %时钟类
properties
      Enable = false   %是否启动
end
events
      Alarm   %闹钟
end
end
```

像类中其他成员一样，在 events 块中定义事件时，也可以设置相应的特性值，以便控制事件的触发和听众的定义等，例如：

```
events (ListenAccess = 'private', NotifyAccess = 'private')
anEvent
anotherEvent
end
```

上面 events 块中的两个事件 anEvent、anotherEven 都具有相同的特性值，即 ListenAccess 和 NotifyAccess 都是私有的，这就表明这两个事件只有本类的方法才可访问，并且只有本类的方法才可触发它们的发生。如果要在同一个类中定义其他具有不同特性值的事件时，只要单独再开一个 events 块即可。表 8-2 给出了事件可以设置的所有特性名、类型及意义。

表 8-2　事件特性表

特性名称	类型	说明
Hidden	逻辑型，默认值为 false	隐藏性。当值为 true 时，该事件就不会出现在 events 函数的返回事件列表中。
ListenAccess	枚举，默认值为 public	收听访问控制特性。该属性用来确定在何处可以创建其听众。可以取的值有： （1）public——访问不受限。 （2）protected——只有本类和派生类的方法可以访问。 （3）private——仅本类方法可以访问。
	meta.class 类对象，或 meta.class 类对象元胞阵列（MATLAB2010 版本以下不支持）	仅在列表内的类被授权可收听该事件。 这里列表中的类是通过 meta.class 对象来指定的。
NotifyAccess	枚举，默认值为 public	广播访问控制特性。该属性用来确定在何处能够广播该事件。可以取的值有： （1）public——任意位置都可广播该事件。 （2）protected——只有本类和派生类的方法才可广播该事件。 （3）private——仅本类的方法才可广播该事件。
	meta.class 类对象，或 meta.class 类对象元胞阵列	仅在列表内的类被授权可广播该事件。 这里列表中的类是通过 meta.class 对象来指定的。

除了自己可以定义事件名称外，MATLAB 中还提供了 4 个预定义事件，这些事件主要涉及所在类中的属性成员值的改变和查询过程，它们分别是 PreSet、PostSet、PreGet 和 PostGet。在定义一个事件类时，不需要在事件块中列出上面的 4 个事件。当上面的事件发生时，相应的回调函数会接收到一个 event.PropertyEvent 对象。关于这 4 个预定义事件的使用我们将在本章第 3 节详细介绍。

8.2.2　事件触发和广播

在一个类中定义事件后，接下来的工作是如何来触发事件和广播事件的发生。当定义了事件的类中某些成员满足一定条件时（属性成员值被更新等），我们就说这个事件触发了或

发生了。对于用户自定义的事件（即除 PreSet、PostSet、PreGet 和 PostGet 事件之外的事件），触发事件的条件是需要程序员根据需要来设置的，然后调用 handle 类中的方法 notify 来广播这个事件已经发生的消息。所有的听众对象将等待广播的消息，如果事件与听众注册的事件相匹配，将做出相应的动作。这里 notify 函数的原型如下：

```
notify(Hobj,'EventName')
```

或

```
notify(Hobj,'EventName',data)
```

上述 notify 函数的第一个参数 Hobj 表示有事件定义的对象句柄，这里也可以是对象句柄数组；第二个参数就是这个对象中的事件名称；第三个参数 data 是封装了有关事件数据的 event.EventData 对象。

例如，我们想通过 MClock 类中 Enable 值的改变来表示事件 Alarm 的发生，从而来通知所有的听众。这就需要该类中添加一个触发事件的方法来调用 notify 函数，具体方法如下：

```
classdefMClock< handle      %时钟类
properties
       Enable = false   %是否启动
end
events
       Alarm   %闹钟
end
methods
 ...
functionobj=MClock(IsEnable)
obj.Enable=IsEnable;
end
function AlarmSetChange(obj, IsEnable)     %触发事件的方法
ifIsEnable ~= obj.Enable
obj.Enable = IsEnable;
 notify(obj,'Alarm');                  %广播事件
end
end
end
end
```

MClock 类中通过方法 AlarmSetChange 来实现事件的触发，并调用 notify 函数来广播事件的发生。在这里，如果参数 IsEnable 与本类中的 Enable 值不同，就将触发事件 Alarm。

8.2.3 如何响应事件（听众的定义）

一旦某事件被触发，由谁来收听这个消息呢？上一小节中给出了触发事件的方法，但某事件的发生还需要产生相应的响应动作，这就需要定义听众，由听众来调用某一个方法，做出相应的动作来响应该事件。定义听众的一般方法是用 handle 类中的成员函数 addlistener 完成。addlistener 函数的原型如下：

```
lh = addlistener(Hsource,'EventName',callback)
```

或

```
lh= addlistener(Hsource,property,'EventName',callback)
```

这里的第一个函数用来为指定的事件创建一个听众，而第二个函数用来为 MATLAB 中预定义的事件创建一个听众。参数 Hsource 是发生事件的对象句柄，也可以是对象数组的句柄；参数 EventName 是 Hsource 所指对象中的事件名称；参数 callback 是函数句柄，即事件触发时将执行的函数。返回值 lh 是 event.listener 类型的听众对象句柄，用来表示一个听众。

例如，在 MClockFun 类中一个定义听众的代码如下：

```
classdefMClockFun< handle   %时钟功能类
properties
ListenerHandle  %听众句柄
end
methods
functionobj = MClockFun(MClock_obj)
hl =
addlistener(MClock_obj,'Alarm',@MClockFun.listenMyEvent);%创建听众
obj.ListenerHandle = hl;
end
end
methods(Static = true)
functionlistenMyEvent(obj,src,evnt)
disp('响应事件 Alarm 了');%用来检验该函数是否调用
end
end
end
```

这里 MClockFun 类的构造函数中定义了一个与 MClock 类中事件 Alarm 关联的听众，并将这个听众的句柄放置在 ListenerHandle 中，与此同时将该事件与回调函数 listenMyEvent 进行绑定。

现在可以来检验事件触发和事件响应的过程了。在 MATLAB 命令窗口中输入如下命令：

```
>>T=MClock;
>>TF=MClockFun(T);
>>T.AlarmSetChange(1);  %这里改变 Enable 的值
```

第一条命令表示定义一个 MClock 类对象 T，是一个发生事件的对象；第二条命令表示定义一个 MclockFun 类对象 TF，并将对象 T 通过构造函数传递给 TF；第三条命令是通过 T 的成员函数 AlarmSetChange 来触发事件 Alarm。这样 TF 中的听众 ListenerHandle 便会调用函数 listenMyEvent。

其运行结果为：

```
响应事件 Alarm 了
```

如果想终止一个听众的生命期，即删除该听众，可以用 delete 函数来实现，如：

```
>>delete(lh)
```

这时，将从 MATLAB 工作空间中移除该听众对象。

8.2.4 听众的几种创建方式

MATLAB 中有两种方式来创建听众：一种是利用 handle 类中的 addlistener 函数来创建听众；另一种是直接利用 event.listener 类（听众类）的构造函数来创建听众。

（1）用 addlistener 创建听众

使用这种方式时，该函数会同时将所创建的听众与某个产生事件的对象生命周期相绑定。关于用 addlistener 创建听众的过程在上一小节中说明了，这里再举例说明其使用方法。

例如：

```
L1 = addlistener(MClock_obj,'Alarm',@MClockFun.listenMyEvent)
```

将为 Mclock_obj 对象的事件 Alarm 创建一个听众，并且注册相应的回调函数 listenMyEvent。

又如：

```
L2=addlistener(MClock_obj, 'Enable','PostSet',@MClockFun.listenMyEvent)
```

将为 Mclock_obj 对象创建一个 PostSet 听众，并且注册相应的回调函数 listenMyEvent。

（2）用 event.listener 类的构造函数创建听众

一般情况下创建一个听众可以使用 addlistener 函数。我们也可以直接调用 event.listener 类的构造函数来创建听众。在这种方式下，所创建的听众不与产生事件的对象生命周期相绑定，而是与听众对象相关联。只要在听众对象的作用域范围内且没有被删除，则听众总是有效的。

event.listener 类是一个 handle 型类，其作用是来创建听众，并将创建的听众响应指定的事件，当事件触发时调用关联的回调函数。创建听众的格式为：

```
lh = event.listener(Hobj,'EventName',@CallbackFunction)
```

上面的命令为 Hobj 对象中的事件 EventName 创建了一个 event.listener 型听众 lh。如果 Hobj 是一个对象数组的句柄，则该听众会响应数组中任何一个对象的 EventName 事件。而 CallbackFunction 是关联的回调函数。这里的回调函数至少应接收两个输入参数，如：

```
functionCallbackFunction(source,eventData)
...
end
```

这里 source 应是产生事件的对象，eventData 是 event.EventData 类对象。

例 8-1 为 MClock 类的对象创建一个 event.listener 听众。

解：程序代码如下。

```
function main()
hobj=MClock;
hl = event.listener(hobj,'Alarm',@CallbackFunction);
hobj.AlarmSetChange(1);
end
functionCallbackFunction(source,eventData)
alarm=source.Enable
disp('响应事件 Alarm 了'); %用来检验该函数是否调用
end
```

其中 main 函数中的 hobj 是 MClock 对象，hl 是听众句柄，CallbackFunction 是与该听众关联的回调函数。

上面的代码运行结果是：

```
alarm =1
响应事件 Alarm 了
```

在上面的代码中，我们可以设置 Enabled 为 false 来关闭对回调函数的调用，例如：

```
lh.Enabled = false;
```

而使用如下命令时，就不会去调用回调函数 CallbackFunction：

```
hobj.AlarmSetChange(1);
```

注意：采用这种方法创建的听众其作用域范围与 addlistener 函数创建的听众有所不同，前者只有在创建该听众的作用域内有效。例如，看看下面的程序，可以比较出它们的不同。

```
function main()
clear all
clc
hobj=MClock(0);
fun(hobj);
hobj.AlarmSetChange(1);
end
function fun(hobj)
%以下听众对象作用域仅限于这个函数体
```

```
event.listener(hobj,'Alarm',@CallbackFunction);
%以下产生的听众对象作用域是全局的
%addlistener(hobj,'Alarm',@CallbackFunction);
end
functionCallbackFunction(source,eventData)
alarm=source.Enable
disp('响应事件 Alarm 了'); %用来检验该函数是否调用
end
```

在 fun 函数中当采用 event.listener 创建听众时，由于其作用域仅限于该函数体内，因此在 main 函数中无效；而改用 addlistener 创建的听众，则在 main 函数中有效，这是由于它与事件对象 hobj 的生命周期绑定了。

8.2.5 回调函数定义和调用

创建完听众后就可以定义听众的回调函数了，听众的回调函数可以是一个对象中的普通方法或静态方法，也可以是一个普通的函数（全局函数）。根据回调函数在不同位置处的定义，其函数参数形式有所不同。一般回调函数的定义格式如下。

（1）作为一个类的普通方法

```
methods
functionCallbackFunction (obj,src,evnt)
    %obj：本类实例
    %src：产生事件的对象
    %evnt：事件数据
    ...
end
end
```

这时，回调函数的第一个参数 obj 就必须是本类对象；第二个参数 src 是产生事件的对象；第三个参数 evnt 是事件数据。

（2）作为一个类的静态方法

此时，本类对象参数 obj 可以缺省，形式如下：

```
methods(Static = true)
functionCallbackFunction (src,evnt)
    %src：产生事件的对象
    %evnt：事件数据
      ...
end
end
```

（3）作为普通函数

此时，对象 obj 可以缺省，形式如下：

```
functionCallbackFunction (src,evnt)
    %src：产生事件的对象
    %evnt：事件数据
    ...
end
```

注意： 无论什么情况下，回调函数至少应有两个输入参数。

有了回调函数，接下来就是回调函数的调用问题了。根据回调函数是类的普通方法、静态方法和普通函数的不同情况，其调用格式也有差别，具体如下。

（1）如果回调函数是一个普通函数，调用格式为：

```
hlistener = addlistener(eventSourceObj,'MyEvent', @CallbackFunction);
```

（2）如果回调函数是一个对象中的方法，则调用格式为：

```
hlistener = addlistener(eventSourceObj,'MyEvent', @obj.CallbackFunction);
```

（3）如果回调函数是一个类中的静态方法，则调用格式为：

```
hlistener = addlistener(eventSourceObj,'MyEvent', @ClassName.CallbackFunction);
```

另外，如果函数形式比较简单，也可用无名函数的形式调用，如：

```
hlistener = addlistener(eventSourceObj,'MyEvent', @(x,y) x.^2+y.^2);
```

当用 event.listener 类的构造函数来定义听众对象时，回调函数的调用方法与 addlistener 一致。

8.2.6 一个例子

前面介绍了 MATLAB 中事件模型的基本原理和设计方法。下面通过一个完整的实例来实现事件模型的具体设计过程。

例 8-2 建立一个图形用户界面，要求：

（1）在界面上用 subplot 命令绘制两个坐标轴，在每个坐标轴上各绘制一个曲面；

（2）利用函数的变化和函数区间的变化来触发件，通过事件来及时更新函数图形的绘制；

（3）通过上下文菜单来设置各个坐标轴是否收听事件。

分析：在本例中涉及到两个事件来对应函数变化情况和自变量区间变化情况。函数的变化情况可以在 events 块中定义一个普通的事件 UpdateGraphics 来描述，而函数区间的变化情况可以采用 PostSet 事件来设置。这样，就需要定义两个听众来响应这两个事件。利用听众的 Enabled 特性来设置是否收听这些事件。

解：为解决本例问题，需要设计两个类，一个是函数运算类 MfunEval，在这个类中定义一个函数句柄 hFun 和区间变量 Lm，并在其中 events 块中定义一个 UpdateGraph 事件，另一个是坐标轴类 Maxes，在其中定义两个听众对象，并与相应的回调函数相关联实现绘图更新。

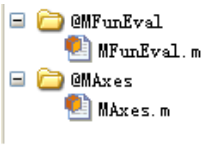

图 8-2 文件目录结构

本实例文件目录结构如图 8-2 所示。

文件 MFunEval.m 中的代码为：

```
classdefMFunEval< handle %函数计算类
properties
hFun%函数句柄
end
properties (SetObservable = true)
     Lm = [];  %区间
end
properties (Dependent = true)
     Data    %保存网格化数据
end
events
UpdateGraph  %更新图形
end
methods
     function obj = MFunEval(fcn_handle,limits)     %构造函数
obj.hFun = fcn_handle;
obj.Lm = limits;
end
functionfofxy = set.hFun(obj,func)
obj.hFun = func;
      notify(obj,'UpdateGraph');                    %广播事件
end
     function data = get.Data(obj)                  %获取网格化数据
      [x,y] = MFunEval.grid(obj.Lm);
matrix = obj.hFun(x,y);
```

```
data.X = x;
data.Y = y;
data.Matrix = matrix;
end
end
methods (Static)
      function [x,y] = grid(lim)                    %网格化
inc = (lim(2)-lim(1))/20;
       [x,y] = meshgrid(lim(1):inc:lim(2));
end
end
end
```

文件 MAxes.m 中的代码为：

```
classdefMAxes< handle       %坐标轴类
properties
FunObject  = []             %函数计算类对象
hLUpdateGraph = []      %图形更新听众句柄
hLLm = []               %区间 Lm 的 PostSet 事件听众句柄
hEnableCm = []          %上下文菜单“Listen”句柄
hDisableCm = []         %上下文菜单“Don't Listen”句柄
hAxes = [];             %坐标轴句柄
hSurface = []           %曲面对象句柄
end
methods
functionobj= MAxes(funobj)
obj.FunObject = funobj;
obj.CreateLisn;  %创建听众
end
     function CreateLisn(obj)  %创建两个听众
obj.hLUpdateGraph = addlistener(obj.FunObject,'UpdateGraph',...
@(src,evnt)listenUpdateGraph(obj,src,evnt));
obj.hLLm = addlistener(obj.FunObject,'Lm','PostSet',...
@(src,evnt)listenLm(obj,src,evnt));
end
     %听众 1 的回调函数
     function listenUpdateGraph(obj,src,evnt)  %更新图形
ifishandle(obj.hSurface)
obj.updateSurfaceData
end
end
functionupdateSurfaceData(obj)
data = obj.FunObject.Data;
set(obj.hSurface,...
            'XData',data.X,...
            'YData',data.Y,...
            'ZData',data.Matrix);
end
     %听众 2 的回调函数
functionlistenLm(obj,src,evnt)
ifishandle(obj.hAxes)
lims(obj);
ifishandle(obj.hSurface)
obj.updateSurfaceData
end
end
end
functionlims(obj)
lmts = obj.FunObject.Lm;
set(obj.hAxes,'XLim',lmts);
set(obj.hAxes,'Ylim',lmts);
end
     function delete(obj)  %析构函数
ifishandle(obj.hAxes)
delete(obj.hAxes);
else
return
```

```
end
end
end
methods (Static = true)
      function CreateViews(funobj) %绘制曲面
hFigure = figure('Name','利用事件模型更新图', 'Toolbar','none');
for k=1:2
axesobj(k)= MAxes(funobj); %定义坐标轴对象
axh=subplot(1,2,k);
axesobj(k).hAxes=axh;
hcm(k)=uicontextmenu; %上下文菜单
set(axesobj(k).hAxes,'Parent',hFigure,...
            'FontSize',8,...
            'UIContextMenu',hcm(k));
axesobj(k).hEnableCm = uimenu(hcm(k),...
             'Label','Listen',...
              'Checked','on',...
'Callback',...
@(src,evnt)enableLisn(axesobj,src,evnt));
axesobj(k).hDisableCm = uimenu(hcm(k),...
             'Label','Don''tListen',...
              'Checked','off',...
'Callback',...
@(src,evnt)disableLisn(axesobj(k),src,evnt));
view(axesobj(k).hAxes,60,30)
axesobj(k).lims;
surfLight(axesobj(k),axesobj(k).hAxes); %绘制曲面
end
end
end
end
```

以下是三个全局函数：

```
functionsurfLight(obj,axh)
obj.hSurface = surface(obj.FunObject.Data.X,...
obj.FunObject.Data.Y,...
obj.FunObject.Data.Matrix,...
        'FaceColor',[.8 .8 0],'EdgeColor',[.3 .3 .2],...
        'FaceLighting','phong',...
        'FaceAlpha',.3,...
        'HitTest','off',...
        'Parent',axh);
lims(obj)
camlight left; material shiny; grid off
colormap copper
end
functionenableLisn(obj,src,evnt)
obj.hLUpdateGraph.Enabled = true;
obj.hLLm.Enabled = true;
set(obj.hEnableCm,'Checked','on')
set(obj.hDisableCm,'Checked','off')
end
functiondisableLisn(obj,src,evnt)
obj.hLUpdateGraph.Enabled = false;
obj.hLLm.Enabled = false;
set(obj.hEnableCm,'Checked','off')
set(obj.hDisableCm,'Checked','on')
end
```

在命令窗口中输入如下命令后，会出现如图 8-3 所示的执行效果。

```
>>funobj = MFunEval(@(x,y) 3*cos(x).*cos(y).*exp(-sqrt(x.^2+y.^2)/6),[-5 5]);
>>MAxes.CreateViews(funobj);
```

提示：第一条命令创建了一个 MfunEval 对象，构造函数中第一个参数是一个二元函数，用无名函数方式给出；第二个参数是区间，即函数计算区域是[–5,5]×[–5,5]。第二条命令是

调用了坐标轴类 MAxes 中的静态方法 CreateViews。

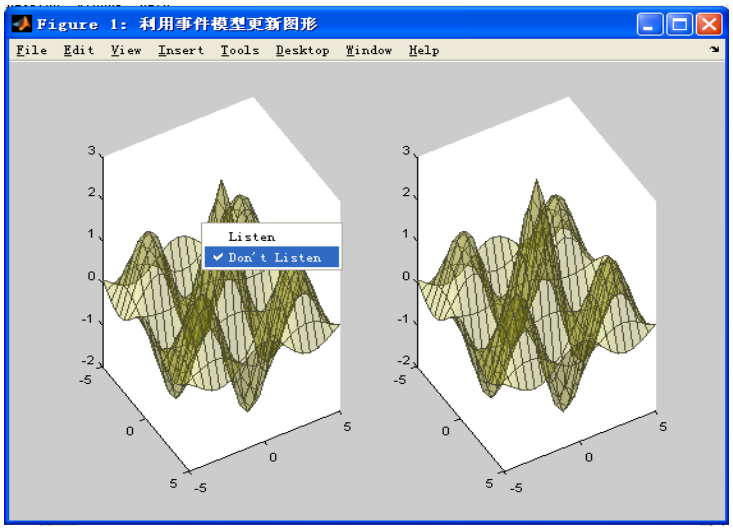

图 8-3　利用事件模型更新图形

将鼠标移到在上面图形界面的左面坐标轴上点击右键，在弹出的上下文快捷菜单中选择“Don’t Listen”，在命令窗口再输入如下命令：

```
>>funobj.hFun=@(x,y) (x.^2+y.^2)
```

此对象 funobj 中的事件 UpdateGraph 被触发，由于左侧坐标轴对象听众的特性值 Enabled 为 false，所有它不去执行回调函数 listenUpdateGraph；而右侧听众的特性值 Enabled 为 true，所有会去执行回调函数 listenUpdateGraph。此时坐标轴上图形更新变化情况如图 8-4 所示。

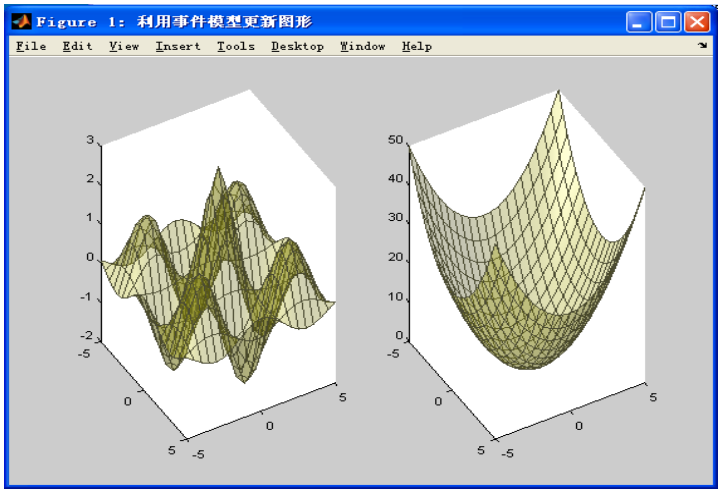

图 8-4　利用事件模型更新图形

同样输入：

```
>>funobj.Lm=[-10,10];
```

会触发 PostSet 事件的发生，可改变函数绘图区间，并有相同的更新效果。