WIN32 下动态对话框的生成

范明 （武汉大学软件工程国家重点试验室 430072）

摘要：动态对话框是 Windows 开发环境下一种强有力的技术。本文探讨了如何在 WIN16 及 WIN32 编程环境下实现动态对话框的细节和要点。

关键词：动态对话框 资源编辑器 DLGTEMPLATE DLGITEMTEMPLATE

1. 什么是动态对话框？

一般对话框是在编程时通过资源编辑器生成的，运行的时候对话框的外观和功能就已经固定了。而“动态对话框”从其名称上可以知道在运行过程中是能根据需要而改变其功能和外观的。不言而喻，这种“在飞的”对话框能大大提高用户界面的质量。

虽然被市面上大部分编程手册所忽略，但如果留心寻找，那么在许多成功的 Windows 商业软件产品中都不难发现动态对话框的存在。最为著名的例子也许是 Micorsoft 的中文 Word 5.0 的“工具”条中“选项...”对话框。在这个对话框中，左半部分是一个“用户自绘”的列表框（List box），包含有代表不同目的的选项组的图标，当用户选中某个图标之后，相对应的那组子控件就会出现在对话框的右半部分，乍一看这仿佛是在给用户变魔术。同把各组选项都做成分立的对话框的处理方法相比，这样处理“选项的输入”无疑方便清晰多了。

2. 生成动态对话框的简单办法

设想要实现这样一个动态对话框，它同三组子控件相关。要求根据运行时的需要在它们之中选择合适的一组作为对话框的当前子控件。如何实现呢？先介绍两个简单的办法。虽然它们很快捷方便，但是稍后将发现它们的不足之处。

第一种办法：利用资源编辑器，将所有的三组子控件都堆叠在这个对话框上，运行时根据需要利用自定义的函数在对话框上仅显示应该显示的子控件而隐藏其余的。由于子控件相互堆叠，所以这三组子控件实际上是共享同一块显示区域。用下面这个名为 ShowCategory 的函数可以实现这种形式的动态对话框：

```
void ShowCategory (HWND hwndDlg, int nFirstID,
int nLastID, BOOL fShow)
{
    int i;
    for (i = nFirstID; i < nLastID; i++)
        ShowWindow(GetDlgItem(hwndDlg, i), fShow)
;
};
```

很明显，要利用上面的函数就需要各个组的子控件的 ID 号连续。可以设想，当程序启动的时候，将会利用 ShowCategory 将所有的子控件的显示属性都初始化为“隐藏”，然后根据执行中对用户输入的处理结果把要显示的子控件的起始 ID 号和终止的 ID 号作为参数赋给函数 ShowCategory，修改这些子控件的显示属性为“显示”，同时将其余的子控件的显示属性置为“隐藏”，就达到了目的。

你一定会承认，这相当方便迅速，如果对话框拥有的子控件稍微多一点的话，将它们堆叠在一起很可能是一场恶梦，最终结果是你无法分辨资源编辑器中的对话框究竟包括了一些什么子控件。

第二种办法：也是利用资源编辑器。只是要说明一个事实，那就是利用资源编辑器定义对话框的时候子控件是能够放在对话框之外的。实际上一个对话框同任何窗口（Window）一样，从逻辑上说是一块大区域的可显示部分。如果一个子控件的坐标处于对话框的客户区范围以外，那么 Windows 在显示时会自动地将它剪贴掉。虽然 Microsoft 的 Windows SDK 中包含的对话框编辑器和 Visual C++ 的资源编辑器都不支持这种特性，但是 Borland 的 Resource Workshow 却是支持的。方法二就是在运行时根据需要将相应的那一组子控件“移动”到对话框的客户区中。尽管子控件仍然共享同一显示位置，但是相应的资源文件清晰整洁得多了。

至于具体实现，可以假设你运行 Borland 的资源编辑器 Resource Workshop，那么现在就能将你的对话框放在

编辑器工作区的正中间，上下左右都空出足够的空间。对于方法一中的实例，将三组子控件分别集中在对话框之外的上、下和左三块区域中。当然，这些子控件的 ID 编号仍然遵循同组连续，不同组不相同的规则。为了将相应的那组子控件移动到对话框的客户区中，你需要一个自定义的小函数。清楚了手段和目地之后，实现这样一个小函数是很简单的。

3. 简单方法的不足之处

从编程角度来看，方法一和方法二是生成动态对话框的快枪手。但从另外几个也许更重要的角度出发，方法一和方法二并不尽如人意。

首先是系统的资源耗费问题。在方法一和方法二中，尽管动态对话框可能在某时刻仅仅显示 10 个子控件，但实际上它可能包含的子控件在运行时都调入了内存，这要耗费大量的系统资源空间。所以一个稍微复杂一点的动态对话框（比方说，包含 60 个子控件而一次最多仅显示 10 个）就有可能导致系统的资源空间被耗尽。

其次这不是真正意义上的动态对话框，因为编译时就决定了对话框最多能有多少种外观和功能。这在某些要求很高的情况下，例如对结构不同的数据库动态生成编辑界面，仍是无法满足要求的。下面就讨论 Windows 下真正的动态对话框的生成。

4. 生成在 WIN32 API 下的真正意义的动态对话框

对于这个问题，首先要说明的是内存中的对话框模板。实际上每个对话框资源在调入内存时都要生成一个对话框模板。根据这个模板，系统生成屏幕上的对话框。一般情况下这是自动实现的，但你可以通过函数 LoadResource 和函数 DialogBoxIndirect 来“手动”实现这一过程。因此，内存中的对话框模板是真正的动态对话框的核心。

内存中的对话框模板由一个 DLGTEMPLATE 数据结构，相关数据以及一到多个对话框包含的子控件的 DLGITEMTEMPLATE 数据结构组成，每个子控件都有一个 DLGITEMTEMPLATE 结构相对应。

DLGTEMPLATE 的结构定义如下：

```
typedef struct {
  DWORD style ;
  DWORD dwExtendedStyle ;
  WORD cdit ;
  WORD x ;
  WORD y ;
  WORD cx ;
  WORD cy ;
} DLGTEMPLATE ;
```

其中 style 字段定义对话框的类型，dwExtendedStyle 系统不用，可由用户来定义自己的对话框类型；x 和 y 两个字段定义对话框在屏幕上的位置；cx 和 cy 两个字段定义对话框的大小；cdit 字段定义对话框中的子控件个数，也就是本结构后继的 DLGITEMTEMPLATE 结构的个数。

在 DLGTEMPLATE 结构之后是一系列变长信息。

第一个 16 位值定义对话框是否具有菜单。如果这个值是 0x0000 就没有菜单，如果是 0xFFFF 那么后继的那个 16 位值就是菜单的资源 ID。在既不是 0x0000 又不是 0xFFFF 的情况下，系统就认为这是一个以空结尾的菜单资源的名称字符串。

在菜单定义之后的 16 位值定义对话框所属的窗口类。当这个值是 0x0000 时，Windows 就使用预定义的窗口类作为对话框的窗口类；如果是 0xFFFF，那么其后的那个 16 位值就是一个已注册的窗口类的原子值；如果都不是，Windows 就认为这是一个以空结尾的已注册类的类名称。

窗口类型定义之后是对话框的标题字串，这是一个以空结尾的字串。如果对话框没有设置 WS-CAPTION 类型，那么这个标题将不被显示。

如果对话框设置了 DS-SETFONT 类型，那么标题字串之后是对话框中用到的字号定义，这个 16 位的值定义了显示的字大小；紧接着是字型的定义，这是一个以空结尾的字型名称字串。如果对话框不包含 DS-SETFONT 类型，那么以上的内容就不是对话框内存模板的一部分。

这些变长信息之后就是对话框包含的子控件的定义结构 DLGITEMTEMPLATE 了，其数目根据对话框上实际拥有的子控件的数目变化。DLGITEMTEMPLATE 结构的定义如下：

```
typedef struct {
  DWORD Style ;
  DWORD dwExtendedStyle ;
  WORD x ;
  WORD y ;
  WORD cx ;
  WORD cy ;
  WORD id ;
} DLGITEMTEMPLATE ;
```

结构中的 Style 字段定义了子控件的类型，可能拥有的值是一个或多个窗口类型标志如 BS-PUSHBUTTON

及 WS-VISIBLE 的“或”组合。dwExtendedStyle 的功能同 DLGTEMPLATE 结构中的功能相同，用于定义用户自己的子控件类型；x 和 y 指出这个子控件在对话框客户区中的位置，cx 和 cy 则指出它的大小。id 字段就是它的标识。

每个 DLGITEMTEMPLATE 之后都紧接着一些变长的信息，这些信息定义了这个子控件的一些重要特征。

第一个 16 位的值定义子控件所属的窗口类，如果这个值是 0xFFFF 的话，其后紧接着的那个 16 位值就是一个已注册的窗口类的原子值；否则 Windows 就认为这是一个以空结尾的字符串，是一个已注册类的名称。

窗口类定义之后是子控件中显示信息的定义。由于子控件上不仅可显示字符，更可以显示图标和位图，所以这可能是一个资源的 ID 或一个字符串。如果窗口类定义之后的第一个 16 位值是 0xFFFF 的话，那么它后面的那个 16 位值就是子控件中显示的资源的 ID，否则 Windows 就认为这一部分是一个以空结尾的字符串，将把它直接显示在子控件上。

显示信息定义之后是这个子控件的用户扩展信息，这一部分根据子控件是否具有用户扩展属性而存在。

通过上面的介绍，如何利用程序生成动态对话框就比较清楚了。有两种手段可供选择。第一种手段就是完全通过程序定义和操纵数据来实现；第二种可先用资源编辑器定义一个模板对话框，然后在运行时将它装入内存并利用程序动态修改来达到目的。

上面所描述的技术可以很顺利地应用于 WIN16 编程之下，对于 WIN32 有一些特别的地方需要交待。首先在 WIN32 下，以上 DLGTEMPLATE 以及 DLGITEMTEMPLATE 结构中所有的字符串都是 UNICODE 形式的。同一般的字符串最主要的区别在于 UNICODE 格式的字符串中每个字符占据一个字。由于 UNICODE 有一套完整的函数可供调用，因此建议对它们进行操作时最好使用 UNICODE 的函数而不是字符串的函数。另外就是结构对齐的问题。在 WIN32 下，系统要求在 DLGTEMPLATE 结构后接的每一个 DLGITEMTEMPLATE 结构都处于四字节(LONG)对齐的地址上。

(来稿时间：1996 年 10 月)