Windows 环境下数据交换及其在 C++下的实现

张兴滔

摘要:本文阐述了 *Windows* 环境下的两种数据交换方式:静态数据交换、动态数据交换的工作原理和特点以及在 *C++*语言中实现的具体步骤。

一、引 言

在当今计算机向着多媒体、多任务开放系统方向迅猛发展,软件技术朝着面向对象程序设计方法过渡的时候,Microsoft Windows 正是以其图形化的用户界面,支持多窗口多任务功能,突破了传统 DOS 环境下的 640K RAM 的禁,实现了多种数据交换方式等特点,正逐步成为最流行的操作系统。

在多任务环境下进行数据交换,也就是在多个进程之间相互传递(或接收)信息,这在 Windows 环境下的各个应用程序之间经常要使用到,如:实时数据采集处理交换、数据库管理、综合性文档管理,当某个文件修改后,通过 DDE 技术,与这个文件相关的所有图表,全部自动更新等等。这些都涉及到 Windows 环境下的数据交换

技术。数据交换技术是 Windows 的重要特色之一，它使得多窗口环境中的各个应用程序之间能够进行各种数据交换。在 DOS 环境下，程序之间的数据交换，是以外部设备磁盘为交换媒体，利用文件操作来完成数据的发送和接收。而 Windows 提供了几种方法供应用程序之间或者程序内部进行数据通讯，一般情况下，Windows 主要采用下列两种方式实现数据的发送和接收：以剪贴板为中间媒介的静态数据交换 SDE(Static Data Exchange)和相对而言的动态数据交换 DDE(Dynastic Data Exchange)。

二、Windows 环境下静态数据交换的原理及其在 C++上的实现

从上面的讲述可知，Windows 环境下提供了两种数据交换方式，下面就先谈谈静态数据交换及其在 C++上的具体实现方法。

1.Windows 环境下的剪贴板的具体含义及其数据交换的实现

(1)剪板(Clipboard)的具体含义。剪贴板是一种比较简单的静态数据传递系统，它传递的内容可以是文本字符、图形图象等各种数据类型，当一个应用程序把数据放入剪贴板，另一个应用程序需要时可以从剪贴板中获取。通常，这个剪贴板是整个系统可以共享的全局内存块。

(2)剪贴板的原理及其实现方法。剪贴板是 Windows 中最常用的和最基本的数据交换方式。各个应用程序可以在这个共享的数据内存块中存放用于交换的各种格式的数据或者是从这个数据内存块中读取某种类型的数据。

2.剪贴板的数据格式的定义以及数据交换操作的实现方法

(1)数据交换的标准数据格式的规定。正如前面所述，Windows 定义了多种标准数据格式用于剪贴板的数据传递，它们的标识符在 windows.h 头文件中已经定义了，其中常用的二种剪贴板标准格式如下：

①CF-TEXT(文本数据格式)

这是 Windows 默认的数据格式，它是以 NULL 结束的字符串，每行以回车和换行符结尾，送往剪贴板的数据存放在一个全局内存块中，利用在这个存储块中的把柄可以进行数据传递。

②CF__BITMAP(位图数据格式)

Windows 的各个应用程序可以通过位图把柄把位图传送到剪贴板，当应用程序把位图数据传送到剪贴板之后，其它的应用程序就可以使用它了。

(2)剪贴板中的数据交换的操作方法。剪贴板的主要操作是改变全局存储块的分配标记。当某个应用程序使用 GHND 标记(GMEM- MOVEABLE 和 GMEM-ZEROINIT 标记的组合)分配一个全局存储块时，这个存储块就被标记为属于这个应用程序。准确地说，属于这个应用程序的某个事例。通常，当这个程序事例结束后，Windows 将删除这个存储块。当应用程序用 SetClipboard Data 函数把全局内存块的把柄传递给剪贴板时，Windows 要把对这个存储块的所有权从应用程序变成它自己的。这就要求 Windows 调用 GlobalReAlloc 函数来修改这个全局存储块的存储分配标记：

GlobalReAlloc(hMem,OL,GMEM-MODIFY GMEM-DDESHARE);

在 SetCliphoardData 函数调用之后，全局存储块的把柄就不再属于分配它的应用程序。当程序结束时，不应该释放这个存储块，除非裁剪贴板板允许这个应用程序访问该存储块，否则产生存储块的应用程序就不能继续使用它。当应用程序调用 EmptyClipboard 函数后，就可以释放这个全局存储块。

当应用程序调用 GetClipboardData 函数时，Windows 把全局存储块的把柄传递给这个应用程序，并允许它暂时方问这个存储块。然后，应用程序可以把数据拷贝到另一个全局磁储块或者局部存储块中。因此，剪贴板事实上是共享存储段的管理者，应用程序把全局存储块把柄传递给剪贴板，其它程序就可以访问这个存储块，裁剪板保留对这个存储块的所有权。

另外，在任何时刻只能有一个应用程序打开剪贴板。OpenClipboard 函数调用的目的就是防止一个应用程序在使用剪贴板过程中，剪贴板的内容发生变化。也就是说，打开剪贴板就能够获得对这个全局存储块的控制。OpenCliphoard 函数调用返回一个 BOOL 值表示剪贴板是否成功地打开，若另一个作用剪贴板的应用程序未将其关闭，则打开失败。在刚进行剪贴板程序设计的时候，设计人员可能要榆测该值，但是在这种非抢占式

多任务环境中,这个检查并不是十分严格的,如果每个应用程序在处理每务信息不都打开和关闭剪贴板,并且把控帛权给其它应用程序,那么就决不会出现不能打开剪贴板的问题。

注意:在剪贴板被打开后,不要使用 SendMessage 或者 PeekMessage 这两个函数;另外,当不能分配一个全局存储块来复制剪贴板的内容时,可能需要显示一信息框,若这个信息框不是系统形式的,那么在显示信息框的时候,用户可能切换到另一个应用程序,这样就可能出现问题,导致系统崩溃。因此,用户程序应该使用系统形式信息框,但必须在显示信息框之彰关闭板,如果在显示一个会话框时让剪贴板保持打开状态,也可能会碰到问题。

(3)裁剪板内的文本数据交换及其在 C++上的实现方法。假定我们要向剪贴板发送一个字符串,并且已经有了指向该字符串的指针,这时如果字符串存放在应用程序的局部数据段中,则为短指针;相反,如果存放在全局数据段中,则为长指针。假设被传递的字符串的长度是 Length。下为 C++语言中实现的具体步骤:

首先,分配大小为 wLength 的,可移动的全局存储块,其中也包括结尾 NULL 所占空间:

```
hGlobalMemory = GlobalAlloc(GHND,(DWORD)wLength+1);
```

如果分配不到足够的存储块,则 hGlobalMemory 的返回值为 NULL。若分配成功,则返回有效把柄,接着再将该存储块加锁,然后取得一个指向这个内存块的长指针:

```
lpGlobalMemory = GlobalLock(hGlobalMemory);
```

接下来就可以把字符串复制到这个全局存储块中去

```
for (n = 0;n < wLength;n++)
 * lpGlobalMemory++ = lpString++;
```

不必加上结尾的 NULL,因为 GlobalAlloc 的 GHND 标记使得在分配内存时把存储块清零.然后对这个全局内存块解锁:

```
GlobaUnLock(hGlobalMemory);
```

至此,有了一个以 NULL 结尾的文本存储块把柄。接着打开剪贴板,将其清零,以便将这个字符串送入剪贴板:

```
OpenClipboard(hWnd);
EmptyClipboard();
```

接下来就可以用 CF-TEXT 标记符把全局存储块的把柄发送给剪贴板,然后将其关闭:

```
SetClipboardData(CF-TEXT,hGlobalMemory);
CloseClipboard();
```

这样就结束了向剪贴板发送正文的过程。

从上面所述,可以看到在完成文本数据交换时,当数据准备好之后,把它们送入剪贴板需要四个函数调用:

```
OpenClipboard(hWnd);
EmptyClipboard();
SetClipboardData(wFormat,hHandle);
CloseClipboard();
```

而取出数据需要三个函数调用:

```
OpenClipboard(hWnd);
hHandle = GetClipboardData(wFormat);
    [...]
CloseClipboard();
```

在 GetClipboardData 函数和 CloseClipboard 函数调用之间,用户可以复制剪贴板中的数据或者以其它方式使用这些数据。在打开剪贴板把数据放入其中时,必须先调用 EmptyClipboard 函数,以便让 Windows 释放或删除剪贴板中的现存数据,这进用户不能向剪贴板的现存内容加入任何东西,因此在这个意义上,剪贴板在这一个时刻仅含有一种格式的数据。

最后谈一谈在进行静态数据传递过程中应该遵守的一些规则:

.在处理单条信息时,调用 OpenClipboard 和 CloseClipboard 这两个函数;在退出窗口函数时要关闭剪贴板;当剪贴打开时,不要把控制权让给其它程序(可能调用 SendMessage 或者 PeekMessage 这两个函数).

.不要把加锁的存储块把柄传递给剪贴板。

.调用 SetClipboardData 函数后,不要再继续使用这个全局存储块,它不再属于用户的应用程序,因此,那个把柄对于用户来说,应该认为是无效的了。如果还要用到那些数据,,再复制它,并从剪贴板中将其读出。在调用 SetClipboardData 和 ColseClipboard 两个函数之间还可以多次方问这个存储块,但是必须通过 SetClipboardData 函数调用返回的全局把柄,而且在调用了 ColseClipboard 函数之后,应该对这个把柄解锁。

一般说来,向剪贴板传送各类数据需要经过以下几

个步骤:

.安排一个全局内存块,将数据复制到这个区域中;

.打开剪贴板;

.将剪贴板清零;

.将存放数据的全局内存块的把柄,以与其对应的数据格式传给剪贴板;

.关闭剪贴板。

上面讲述了静态数据交换剪贴板的原理和特点以及实现的具体方法,下面我们就讲一讲 Windows 环境下数据交换的另一种方式动态数据交换。

三、Windows 环境动态数据交换的原理及其在 C++上的实现

动态数据交换与剪贴板不同,它允许多个应用程序 DDE 按照 Windows 提供的通讯规则和格式,通过系统的信息队列交换数据或命令,Windows 提供标准通讯规程,使得两个应用程序(或两个事例)之间进行有务不率的数据交换或命令传递。这种传递是以信息为基础,经过发信者和收信者之间的应答对话,实现数据的持久或者自动的交换.DDE 是应用程序通过共享内存进行进程之间通讯的一种方式,它是不需要用户干预的最好的数据交换方法,通常,两个应用程序之间自动建立数据交换链,根据这个链,这两个应用程序之间可以进行自动的数据传递。

1.动态数据交换的具体含义

DDE 是建立在 Windows 内部的信息处理系统上,两个应用程序通过相互之间传递信息进行 DDE"会话",也就是两个应用程序之间进行数据交换的过程,这两个程序分别称为"服务器"Server 和"客户"Client。DDE 服务器是一个存取那些可能对其它应用程序有用的数据的程序;DDE 客户则是一个从服务器获得数据的程序。一次 DDE 会话由客户程序作初始化,客户发送信息给所有当前正在运行的应用程序,这条信息表明客户所需数据种类,则具有该类数据的 DDE 服务器就响应这条信息,由此完成会话。一个单独的应用程序既可以是另一个应用程序的客户,也可以是其它程序的服务器但要求两次不同的 DDE 会话。

2.DDE 的原理及其组织结构

两个应用程序之间的数据交换或者称为"会话",它是 DDE 的一个基本概念。一次会话是由两个应用程序参与:一个客户和一个服务器其中客户负责初始化与服务器的会话以及控制会话流;而服务器则响应客户的各种请求。在两个应用程序之间能够进行数据交换或者称为"会话"之前,必须根据 DDE 规程在两个应用程序之间建立客户与服务器的关系。提语"会话"的应用程序称为服务器而对服务器的会话语求作出响应的则是客户。一个服务器可同时与多个客户,一个客户也可以向多个服务器请求数据,而且一个应用程序能够充当客户和服务器并进行多路"会话",也就是一个应用程序能够参与几个"会话",它不仅可以是一个"会话"中的服务器而且也可以其它"会话"中的客户。当不需要服务器的数据或服务时,客户将终止"会话"。显然,"会话"至少由两个应用程序组成。

服务器与客户之间的"会话"是由应用名和"会话"的"话题"唯一地确定。DDE 的"会话"分为三个层次进行:在开始"会话"时,服务器与客户进行初始化,对通讯标志,即应用名和"话题"进行统一。应用名通常是客户名,而"话题"是一个广义的数据组合题。这个组合题中包含许多具体的数据项,它提供 DDE 中两个应用程序交换的真正内容,它可以由客户发往服务器,也可以是由服务器发往客户。数据项也可以是多种格式,其中包括用于剪贴的数据格式。

3.DDE 的初始化过程

在 DDE 发生前,一定有某一个应用程序打算与其它应用程序进行数据交换。因此这个称为"会话"过程中的服务器通常利用 SendMessage 函数。将第一个参数设置为-1,向 Windows 整个系统"广播"一个"消息",即 WM-DDE-INITIATE,并附上"广播者",也就是服务器的名字(服务器应用程序的把柄",以及将进行"会话"的可能"话题",即是交换的数据格式或类型:

SendMessage(-1,WM-DDE-INITIATE, hWndClient, MAKELONG(Application atom,Topic));

如果系统中某个应用程序收到这个信息后,认为能够就要求的话题进行会话,则作出响应:

SendMessage(hWndClient, WM-DDE-INITIATE ,hWndClient, MAKELONG(Application atom, Topic));

客户向服务 hWndClient 发出 WM-DDE-ACK 信息,来表示"知道"请求就 Application 的 Topic 进行会

话。这里使用宏 MAKELONG,安的两个 WORD 参数将返回在 IParam 和 wParam 中。

当服务器收到客户的肯定回答后,知道它将与应用程序 hWndServer 进行通信。然后便可以向应用程序 hWndServer 正式提出进行交换的具体内容:

SendMessage(hWndClient,WM−DDE−REQUEST,hWndClient,MAKELONG(CF−TEXT, Topic));

这里 Topic 可以是一个 OxOOOOH 到 OxFFFFH 之间的任何一个数,通常它是一个原子号(atom)。由于两个应用程序间的数据交换是基于 Windows 的信息队列系统,参数 IParam 和 wParam 每次只能分别传递一个字的数据。因此对于诸如字任串这样的数据,Windows 提供了一个全局原子表,通过下列函数调用,就可以将字符串唯一地加进这个全局原子表,以便供整个系统中的应用程序使用:

atomApp = GlobalAddAtom("So manyy characters!");

atomTopic = GlobalAddAtom(szTopic);

基中 szTopic 是一个指向以 NUSS 结尾的字符串的指针(LPSTR)。

服务器与客户初始化之后,便可以进行数据交换。这种交换过程如下所示:

服务器	客户
WM−DDE−INITIATE	
	WM−DDE−ACK
WM−DDE−DATA	
	WM−DDE−ACK
	WM−DDE−DATA
WM−DDE−ACK	

由上面可以看出 DDE 初始化结束之后,两个应用程序之间便可以进行数据交换。而 Windows 提供了一种通讯机构,使得两个应用程序之间可以建立一个或者多个数据交换"链",而且可以是"永久"性的。每当两者之间的数据项发生变化,将自动地(无需用户干预地)通知对方这种变化。这种"链"的作用持续到 DDE 会话结束,或者"链"本身终止。

DDE 有两种类型的永久数据交换"链":

."湿链"—每当数据发生变化时,客户只通知服务器这种变化,而不立即斜变更后的数据发往服务器,直到服务器请求。

."热链"—每当数据发生变化时,客户不仅通知服务器这种变化,而且立即将变更后的数据发往服务器。

当服务器需要建立上述永久性数据交换"链"的时候,必须先向客户发送 WM−DDE−ADVISE 信息:

SendMessage(hWndClient,WM−DDE−REQUEST, hWndClient, MAKELONG(hGlobalRam , atomItem));

当这种"链"建立好以后,每当客户数据发生变化时,应该向服务器发送新数据。

4.DDE 在 C++上的实现步骤

进行数据动态交换,主要依靠两个函数来传递信息:

SendMessage(hReceiver,Message,hSender,Data);

以及

PostMessage(hReceiver,Message,hSender,Data);

这两个函数的功能十分相似,但有一个主要区别:SendMessage 函数在向 hReceiver 发送信息 Message 和数据 Data 后,必须等到 hReceiver 处理完毕后,将控制权返回给 hSender,函数 SendMessage 的操作才算结束;但是函数 PostMessage 则不然,当它向 hReceiver 发送 Message 和 Data 后,立即继续执行下一条语句,不管 hReceiver 是否接受到 Message,也不管 hReceiver 是否处理。

四、动态数据交换的具体实现

笔者编写了一个具体的 C++语言实例来实现动态数据交换,这个 DDE 的任务是:Client 要求从 Server 得到一组数据。在这两个应用程序分别起动后,选择 Client 的清单"DDE",Client 和 Server 的数据交换就开始。运行的结果是,在 Client 的一个信息框内显示字符串提示。应该注意的是,在这组动态数据交换过程中,最终由 Server 传递给 Client 的是一个指向字符串的指针,而不是字符串本身。源程序清单本刊略,有兴趣者可与本文作者联系。