如何使应用程序支持多国语言

曾 妍 张俊彩 (中国矿业大学北京校区机电系硕99-3班 100083)

摘 要	在视窗2000中，多国语言用户界面（MUI，Multilanguage User Interface）支持是其全球化的最有用，最典型的特征。多语言用户界面支持允许用户在操作系统里任意设置视窗2000支持的本地化的语言环境。本文主要介绍如何使读者在视窗2000下开发的应用程序，充分利用视窗2000的这一特性来使读者自己开发的应用程序也支持多国语言。

关键词	多国语言 MUI DLL UNICODE

1 引言

1.1 应用程序要支持多国语言

读者可能会有疑问，为什么应用程序要支持多国语言呢？原因是很显然地，现在的世界变的越来越小，国际化、全球化是一种必然的趋势。为了增强产品的竞争力，打入国际市场，获得更高的产品利益回报，我们必须而且只有这么做，否则只能自取灭亡。除了这个很明显地全球化的思想影响外，还有其他两个因素促使我们开发支持多国语言的软件产品：

(1) 只发布一个二进制软件产品。如果我们的产品支持多国语言，那对于不同的的语言版本，我们只需要发布一个统一的二进制软件产品。这样我们不必再为每一个不同的语言版本开发一套软件，节省了大量的人力、物力和财力、也不需再去维护不同的语言版本了。实际上视窗2000、Microsoft Office 2000和Microsoft Internet Explorer 5.0都只有一个唯一的内核二进制代码（gdi32.dll），它们不必为不同的国家而发布不同的内核。也就是说微软的这些产品本身是支持多国语言的。例如，对于日本版，中文版或英文版的Windows 2000，如果将来系统升级需要服务的补丁，那对于三个不同的语言版本，只需要发布一个统一的补丁内核程序就可以了，不再需要去区分不同的语言版本，可以看出支持多语言还可以便于系统的升级。

(2) 避免成为自己的竞争对手。如何来理解这句话呢？我们可以做个比喻：假如您正在为一个产品开发两个不同的版本，一个是中文版的，另一个是日文版的。现在假设中文版的已经发布了，您以为日文版的很快就可以发布了（你认为不会因为语言的不同而耽误很多的时间），所以您给客户的承诺日期很短，可是结果并不象您想象的那样简单，后果是发布日期延期了很多，您的承诺没有兑现，客户有一种被欺骗的感觉，不会再信任你了。原因很简单，就是因为你自己而使客户离开了你。但如果您从一开始就使您的产品支持多语言，那所有这些问题都可以避免，您的客户群体也不会减小了。

1.2 支持UNICODE

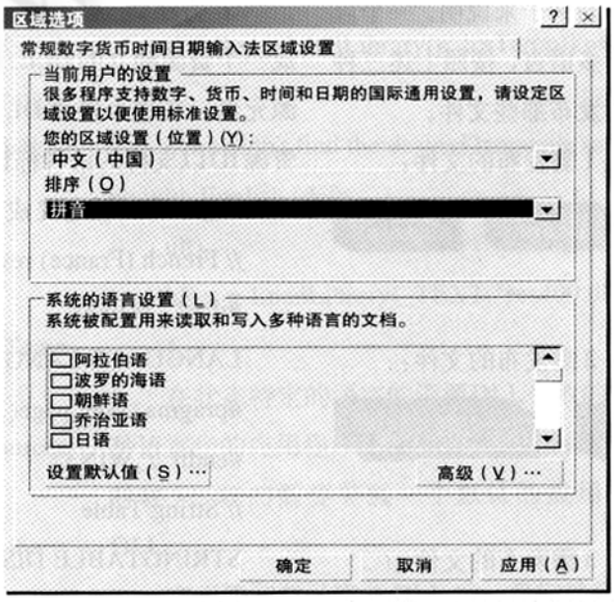

图 1 系统本地化语言配置界面

有一点读者必须认识到，不管您采用什么方法来发布单一的、支持本地化的软件产品，您的产品都必须支持UNICODE。实现UNICODE支持，可以使您不必为您的应用程序运行的操作系统支持哪种语言而担心。UNICODE采用16位字符编码，能够代表全球的所有语言（以前的8位字符编码，比如ANSI编码只能支持约220种不同的国别语言）。在视窗2000中用户可以通过控制面板中的区域选项设置本地化的语言环境，如图1。

可以点击图1中的设置默认值按钮直接设置成本地的语言环境（用户在安装操作系统时指定的语言环境）。

2 实现多语言用户界面

通常有三种方法可以实现应用程序多国语言支持用户界面：

(1) 一种语言一个应用程序；

(2) 一个独立的应用程序内核，一个单一的、包含多语言资源的资源DLL；

(3) 一个独立的应用程序内核，每个语种形成一个独立的语言资源DLL。

下面举一个支持英文、日文、德文的应用程序的例子来说明，一个应用程序要支持多语言，按照上述三种方法各自需要发布那些文件。

采用方法1需发布的文件：

采用方法2需发布的文件：

采用方法3需发布的文件：

表 1　三种方法的比较

	优点	缺点
方法一	* 最容易实现	* 每种语言需要一套源代码 * 资源改变需要重新编译整个工程 * 不同语言界面之间的切换需要启动不同的应用程序实例 * 浪费磁盘空间：每种语言需一个内核应用程序
方法二	* 比较容易实现 * 可以平滑的在不同的语言环境中切换 * 版本更新不需要重新编译资源文件	* 添加对新语种的支持很困难 * 安装支持的语种子集困难 * 不需要的语种也在一个资源DLL里，浪费内存和磁盘空间 * 在动态装载语言界面元素（如LoadMenu,LoadString等）时不够直白 * 在Windows 9x环境下不能改变线程的本地化
方法三	* 可以平滑的在不同的语言环境中切换 * 版本更新不需要重新编译资源文件 * 可以完全控制安装支持的语种 * 很容易添加对新语种的支持 * 特定语种的更新不影响其他的语言 * 不需要担心用户，操作系统以及线程的本地化 * 在Windows 9x,NT,2000下均可以实现	* 需要同步维护所有支持语种的资源DLL

下面比较一下三种方法的优缺点：

① 方法1：是最传统的方法，它对不同的语言都需要一套应用程序，很浪费也不好维护。目前这种方法已经基本上被淘汰了。

② 方法2：思想精髓是把应用程序的资源部分从源代码中分离出来，生成一个资源独立的DLL，这个资源DLL包含了所有应用程序支持的语种的资源元素。在这个资源DLL的RC（Resource Complier）文件中，对于同一个资源ID以支持的语种标识符为标记形成多份拷贝。例如下面的例子，字符串ID IDS_MYEX被定义成同时支持法语和英语：

```
// French (France) resources
#ifdef _WIN32
LANGUAGE LANG_FRENCH, SUBLANG_FRENCH
#pragma code_page(1252)
#endif //_WIN32
// String Table
STRINGTABLE DISCARDABLE
BEGIN
IDS_MYEX "Cette phrase est en fran?ais...(France)"
```

```
END
#endif // French (France) resources
// English (U.S.) resources
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif _win32
// String Table
STRINGTABLE DISCARDABLE
BEGIN
IDS_MYEX "This is an English string...(USA)"
END
#endif // English (U.S.) resources
```

为了在程序运行的时候获得资源，可以用LoadLibrary API来装载这个资源DLL。可以调用EnumResourceLanguages API来列举选定的资源DLL支持的所有语言，同时调用FindResourceEx API来获得特定的类型、名称、语言所对应的资源。然后显示不同语言界面就是从资源DLL里找到对应的资源并显示的事情了。如果在不同的语言环境间切换只需重新装载新的语言界面元素，然后重新刷新客户区就可以了。

有一点很重要Windows的资源装载API函数一直默认装载现在操作系统所处的语言环境的资源。调用GetThreadLocale API可以查询当前线程所处的语言环境，也可以调用SetThreadLocale API来设置某线程的语言环境。所有的资源装载API函数（比如，LoadIcon,LoadString,LoadCursor等等）都是基于当前线程的语言环境来获取资源元素的。在上面的资源文件的例子中，如果用户的语言环境不是法语，那您就不可能通过API函数LoadString获得法语的IDS_MYEX。下面是一段有关这些API函数应用的代码：

```
// if our thread locale is English to start with...
g_hInst = LoadLibrary(_TEXT("intl_res.dll"));
LoadString(g_hInst, IDS_ENUMSTRTEST, g_szTemp, MAX_STR);
// g_szTemp would then point to the English resources
// changing our thread locale to French.
// Always make sure that French is in fact one of the
// valid languages returned by EnumResourceLanguages
()
// Save a copy of the current thread-locale to set back later
Lcid = GetThreadLocale();
SetThreadLocale(MAKELCID(0x040c, SORT_DEFAULT));
LoadString(g_hInst, IDS_ENUMSTRTEST, g_szTemp, MAX_STR);
// g_szTemp would then point to the French resources
```

③ 方法3：实际上是方法2的扩展。方法2是把所有的语言资源统一封装在一个资源DLL中，而方法3是把不同的语言从这个资源DLL中分出来，对于每种语言生成一个资源DLL。这种方法也是Microsoft Office 2000、Microsoft Internet Explorer 5.x和Microsoft Windows 2000采用的方法。方法3也同样需要用API函数LoadLibrary来装载资源DLL,不同的是：这次必须装载确定的语言的资源DLL，通常情况下我们假定用户要装载的资源DLL的语言就是他所用的操作系统的当前语言环境。然后，如果用户选择了一种新的语言，那当前的资源被释放，新选定语言的资源DLL再被装载。很显然这个过程需要应用程序用新的语言资源重新生成窗体，初始化对话框。

由于语言检测的API函数返回的都是对应语言的langID，所以为了编程及调用的方便，我们建议用语言的langID来命名对应的资源DLL。比如对于英文的资源DLL res_eng.dll就可以命名为res409.dll。因为409就是代表英文的langID。

下面一段代码，演示了在Windows 2000下如何检测用户的语言环境以便装载正确的资源DLL：

```
wLangId = GetUserDefaultUILanguage();
_stprintf(g_tcsTemp, _TEXT("res%x.dll"), wLangId);
if((hRes = LoadLibrary(g_tcsTemp)) == NULL)
{
// we didn't find the desired language satellite DLL, lets go with English (defa
ult).
hRes = LoadLibrary(_TEXT("res409.dll"));
}
```

在状态特定的语言的资源DLL失败的时候（比如，某种语言您的应用程序还没有支持，或着操作系统不支持的时候），我们需要装载一个默认的或用户设定的资源DLL。

在正确装载了资源DLL后，我们就获得这个资源

（下转第74页）

(上接第 70 页)

DLL的句柄（hRes），然后须在应用程序初始化的时候把这个句柄设置给你的应用程序，这是通过 API 函数 AfxSetResourceHandle 实现的。

还有一种方法是综合这三种方法，由于这样做也没有什么特别的好处，所以也不被推荐使用。

3 结论

写一个统一的应用程序内核，是您做国际化软件产品的第一步，这样做会大大减少你的产品的开发和维护开销。实现一个多语言用户界面的支持会使你的产品得到很高的国际化客户满意度。写一个支持UNICODE的应用程序，然后针对不同的语言生成对应的资源 DLL 将使你的产品达到意想不到的效益。■

参考文献

1 *Microsoft MSDN 2000 10版 Using Multiple Languages in Windows 98。*