跨平台应用系统的开发

The Development of the Cross -Platform Applications

李永强 杨林楠 何汉明（云南农业大学 650201）

摘 要：要开发出能够同时在Linux和Windows操作系统上运行的跨平台32位应用程序，可在一个现有的Windows应用程序基础上，对它进行修改；或者遵循编写与平台无关代码的方法来创建一个新的应用程序。本文介绍了如何将一个Kylix的Windows应用程序移植到Linux中的一般方法和步骤以及编写跨平台代码的一般原则。

关键词：平台无关 移植 CLX 编译

1 引言

Linux是当今最具竞争力的企业环境之一，由于其可靠性高、易于升级和低廉的价格，Linux系统被广泛应用于WEB应用服务器，并迅速成为Windows操作系统的竞争对手。这种情况下，市场对Linux应用软件的需求也空前增长。然而，目前大多数可视化集成开发环境是基于Windows系统的，基于linux操作系统的RAD软件则很少；而且，开发人员在开发同一套应用系统时，如果由于系统平台的不同又要重写一次源代码，这都将大大影响开发效率。如果能使用跨平台的技术方法来开发应用系统，使得系统在移植时只需做少量改动，甚至完全不用改动，这势必对软件开发产生深远影响。

2 Windows应用程序移植到Linux

对于基于linux开发的Kylix应用程序，只要没有使用Linux特定的API函数，经过重新编译，都可以在包含了CLX（跨平台通用组件库）的Windows平台上运行。如果已经编写了Kylix的Windows应用程序，或者说针对Windows平台的Kylix应用程序，则可以将其移植到Linux环境中。移植的难易程度取决于该应用程序的特性和复杂程度，以及该应用程序中有多少Windows相关性。

2.1 移植策略

跨平台移植。其着眼于处理一个支持跨平台的API，通常可以提供最快捷的技术，并且被移植的应用程序可以侧重于多个平台。在实际工作中，开发跨平台应用的主要工作量取决于现有的代码。如果现有代码在编写时没有考虑到平台的无关性，则会陷入平台无关的逻辑和平台相关的实现混合在一起的境地。

通常程序的逻辑就是以平台无关的术语来表示的。一些服务通常是抽象的，并隐藏在内部接口之后，使得在所有平台中看起来都是一样的，但实际上每个平台的实现都不一样。Kylix的运行库CLX就是这样的一个例子，对于Windows和Linux平台来说，两者的接口是非常相似的，尽管两者的实现有着很大的区别。

用户应该将跨平台的部分分离开，然后分别实现特定的服务。这种移植方法是最节省的解决办法，因为应用程序的代码大部分都是共享的，而且应用程序的结构可以得到提高，从而大大减少维护的费用。

2.2 移植应用程序

若将一s个应用程序移植到Linux平台上，并且只想让它在Linux平台上运行，则在移植时可能会选择完全除掉Windows特定的特性。但是，如果想让移植后的应用程序能同时在两种平台上运行，则需要对该程序代码进行修改，或者使用$IFDEF之类的指示符来表明某一段代码是Windows或者Linux平台所专有的。下面给出Windows应用程序移植到Linux的一般步骤。

（1）将Kylix的Windows应用程序的源文件和其他与项目相关的文件复制到Linux平台的计算机上。源文件应该包括单元文件(.pas)、项目文件(.dpr)和任何包文件(.dpk)，项目相关文件应包括窗体文件(.dfm)、资源文件(.res)和项目选项文件(.dof)。如果只想通过命令行(而不是IDE)来编译应用程序，则还需要配置文件(.cfg)。

（2）若要创建一个单一的应用程序源文件，并能够同时在Windows和Linux平台上使用，则需要将.dfm文件复制或者重命名为同名为.xfm文件，例如，将unit1.dfm重命名为unit1.xfm。然后在单元文件中将对.dfm文件的引用从{$R *.dfm}重命名(或者借助$IFDEF指示符)为{$R *.xfm}。这是因为，尽管.dfm文件也可以在Linux的Kylix中工作，但是它可能被修改，因而可能在Windows的Delphi中无法工作。

（3）在Kylix中修改所有的uees语句（或者借助$IFDEF指示符来实现这一目的），以便它们引用了正确的单元。

（4）重新编写任何不需要Windows相关性的代码，使该代码更加与平台无关。可以使用运行库例程和常量来完成重写任务。

（5）为在Linux上一些不同的特性寻找等同的功能。使用$IFDEF指示符来限定一些Windows特定的信息。

例如，可在源文件中这样使用$IFDEF指示符来限定一些平台特定的代码。

```
[ $IFDEF MSWINDOWS ]
IniFile.LoadFromFile('c: \ x.txt');
[ $ENDIF ]
[ $IFDEF LINUX ]
IniFile.LoadFromFile('/home/name/x.txt');
[ $ENDIF ]
```

（6）在所有的项目文件中寻找对路径名的引用，并根据下面情况进行适当修改。

① Linux中的路径名使用左斜线/作为定界符（如/usr/lib），并且不同的文件可能位于Linux系统的不同路径中。使用PathDelim（位于SysUtils中）常量来确定与当前系统相适应的路径定界符，并为Linux中的任何文件确定正确的路径。

② 修改对驱动器（如C:\）字母的引用，可在路径字符串中判断第2个字符是否为冒号来寻找驱动器字母。使用DriveDelim常量（位于SysUtils中）来确定与当前系统相适应的驱动器表示术语。

③ 在使用了多个路径的地方，将路径分隔符由分号（；）变为冒号（：）。使用PathSep常量（位于SysUtils中）来确定与当前系统相适应的路径分隔符。

④ 由于在Linux中文件名是大小写相关的，因此要确保应用程序没有改变文件名的大小写。

（7）在Linux中编译该项目。检查任何错误信息，并进行相应修改，直到不出现任何错误。

3 编写可移植代码

在编写跨平台应用程序时，可以使用条件编译，使得在不同平台下进行不同的编译。使用条件编译，可以保留Windows代码，也允许其兼容Linux操作系统的区别。为了创建一个能够在Windows和Linux之间轻松移植的应用程序，要注意以下几点：

- 减少或者隔离对与平台相关（Win32或Linux）的API的调用，而使用CLX方法。
- 不要在应用程序中使用Windows消息机制（PostMessage，SendMessage）。
- 使用TmemIniFile来取代TregIniFile。
- 注意和保持文件和路径名的大小写。

在编写代码时要尽量使用与平台无关的运行库例程，并使用在System、SysUtils和其他运行库单元中的常量。例如，使用PathDelim常量来消除代码中的‘/’符号和‘\’符号在表示路径上的区别。

另一个示例涉及到在两个平台上使用多字节，尤其对于使用汉字的用户。Windows代码通常将多字节字符都看成是只有2个字节，而Linux中的多字节字符的编码可以支持多达6个字节。但是，使用SysUtils中的StrNextChar函数可以使得两个平台都能得到满足。例如，下面所示的Windows代码。

```
while p^<>#0 do
begin
  if p^in LeadBytes then
    inc(p);
    inc(p);
end;
```

可以被修改为下面的与平台无关的代码：

```
while p^<>#0 do
begin
  if p^in LeadBytes then
    p:=StrNextChar(p)
  else
    inc(p);
end;
```

上述代码就是可移植的了，并支持多于2个字节的多字节字符。如果不能够使用运行库函数时，则可以将用户的例程中与平台相关的代码隔离历来放在一个单独的子例程中。而且要尽量限制$IFDEF块的数目，以保持源代码的可读性和移植性。Linux中没有定义条件符号"WIN32"，而定义了条件符号"LINUX"，表明该源代码是准备用来在Linux中进行编译的。

（1）使用条件指示符。使用$IFDEF条件指示符是一种理智的方法来区分Windows平台和Linux平台上的特定代码。但是，因为$IFDEF会使得源代码难以理解和维护，因此一定要选择最适当的时候才使用$IFDEF。如果有其他办法的话，就不要使用$IFDEF。

下面是开发跨平台应用程序时使用$IFDEF的一些原则。

① 除非不得已，否则不要使用$IFDEF。一个源文件中的$IFDEF仅仅在源代码被编译时才被评估。和C/C++不一样，Kylix在编译一个项目时不需要头文件。对于大多数Kylix项目来说，完全重新编译所有的源代码是不可取的。

② 不要在包文件(.dpk)中使用$IFDEF。组件编写者在进行跨平台开发时，要创建两个设计时刻包(分别针对两个平台)，而不是一个使用了$IFDEF的包。

③ 通常，可以使用$IFDEF MSWINDOWS来测定任何Windows平台(包括WIN32)。要保留对$IFDEF WIN32的使用，以便区分特定的Windows平台，如32位和64位的Windows。

④ 除非不得已，不要使用反向的测试指示符，如$IFNDEF。例如，$IFNDEF LINUX并不等于$IF DEF MSWINDOWS。

⑤ 避免$IFNDEF/$ELSE的组合，可以使用肯定的测试指示符(如$IFDEF)来取代它。

⑥ 在与平台无关的$IFDEF指示符中要避免使用$ELSE，要为LINUX特定的代码和MSWINDOWS特定的代码使用单独的$IFDEF块，而不要使用$IFDEF LINUX/$ELSE或者$IFDEF MSWINDOWS/$ELSE之类的指示符语句。

例如，下面的代码在Windows中能够正常运行，但是在Linux中会产生致命的错误。

```
{$IFDEF  WIN32}
{32-bit Windows code}
{$ESLE}
{16-bit Windows code}  //!! By mistake,
```

下转第62页 >>

Linux could fall into this code.

{$ENDIF}

⑦ 对于复杂的测试要使用$IF语法。在一个$IF指示符语句中使用一个布尔表达式来取代嵌套的$IFDEF语句。

(2) 终止条件指示。使用$IFEND指示符可以终止$IF和$ELSEIF条件指示符。$IF只能用$IFEND来终止。$ENDIF只能用来终止旧式的指示符(如$IFDEF、$IFNDEF、$IFOPT等)。当在$IFDEF/$ENDIF中嵌套使用$IF时，不要使用$ELSE指示符，应使用{$ELSEIF True}来取代{$ELSE}。

$ELSEIF是由$ELSE和$IF组合起来的。$ELSEIF指示符允许用户编写多部分条件块，其中只有一个条件块会被执行。如：

```
{$IFDEF doit}
do_doit
{$ELSEIF RTL Version>=14}
goforit
{$ELSEIF somestring='yes'}
beep
{$ELSE}
last chance
{$IFEND}
```

在上面的4个条件中，只有一个会被执行。如果前3个条件都不满足，则$ELSE语句将被执行。$ELSEIF必须使用$INEND来终止，在$ELSE后面不能出现$ELSEIF。各条件是按照正常的$IF…$ELSE顺序来判断的。如果doit没有定义，RTLVersion的值为15，并且somestring='yes'，则只有"goforit"块会被执行，而"beep"块不会被执行，尽管也满足条件。

如果没有使用$ENDIF来终止$IFDEF，则编译器会报告错误信息"Missing ENDIF"。

(3) 发出信息。$MESSAGE编译器指示符可以允许源代码发出提示、警告和错误信息。使用方法为：

```
{$MESSAGE HINT|WARN|ERROR|FATAL 'test string'}
```

信息类型是可选的。如果没有确定信息类型，则默认为HINT。文本字符串是必须的，它代表要发出的信息，且要放在单引号中。

4 结束语

本文对Windows和Linux系统平台间应用系统的移植进行了讨论，说明了不同平台间移植应用程序的一般方法和步骤，同时介绍了开发跨平台应用系统的原则。依据此方法，本单位的一些Windows平台上的应用系统已经顺利移植到Linux平台上，避免了同一系统的重复开发，提高了工作效率。

参考文献

1 俞俊平、余安平，《Kylix编程》，电子工业出版社，2002。

2 Gene?Henriksen著，《Windows NT and UNIX Integration》，电子工业出版社，1999。

3 裴植、肖薇，《红旗Linux开发及网络应用》，人民邮电出版社，2001。

4 李维，《Delphi5.x分布式多层应用系统篇》，机械工业出版社，2000。