一种新型的编程模型——CAR 事件编程模型

李辉　陈榕
(清华大学深圳研究生院, 深圳, 518055)
E-mail: spanee@263.net

[摘要] “和欣”操作系统是一个新型的面向构件的操作系统,其上的的运行时环境我们称之为 CAR(Carefree Application Runtime).我们在 CAR 上开创了一种新型的事件编程模型.本文详细介绍了这种事件编程模型,并与传统的 Windows 消息模型和现有的其他几种事件模型作了比较,指出 CAR 事件编程模型在现代大型网络应用中的优势.

[关键词] Windows　CAR　构件　消息　事件　编程模型

A New Programming Model——the Events Model on CAR

Li Hui　Chen Rong
(Graduate School at ShenZhen, Tsinghua University, ShenZhen, 518055)

[Abstract] Elastos OS is a new Component-Oriented Operating System. We called the runtime environment of Elastos CAR(Carefree Application Runtime). In this paper, we proposed a new events programming model on CAR. As compared with the traditional Windows messages model and several popular events model, our CAR events model showed its advantages when applied in modern large scale network applications.

[key word] Windows, CAR, Component, Message, Event, Programming Model

1 引言

Windows 操作系统从诞生到现在已经十多年了,从 DOS 到 Windows 给人一种耳目一新的感觉.Windows 平台上的基于消息机制的编程模型曾经一度被认为是一种非常成功的模型.众所周知,Windows 的图形界面给用户带来了极大的方便.正是这种消息机制,极好的支持了图形界面应用程序的开发.它使得图形界面的开发非常模块化,条理清晰.正是有了这种消息机制,才诞生了许许多多的方便而又实用的应用程序.不可否认,在过去的十多年中,这是一种非常成功的编程模型.然而,在 Windows 诞生时,互联网远没有现在这样发达.现代大型网络应用需要在操作系统层面上给予更鲁棒,更高效的支持.以这样的需求来看,Windows 的消息机制就远远不能满足要求了.我们在这篇文章中介绍的,是一种新型的基于事件的编程模型,我们称之为 CAR 事件编程模型.

本文第 2 节简单介绍了一下传统的 Windows 消息编程模型;第 3 节提出了消息编程模型的缺点,由此引入第 4 节,对 CAR 事件编程模型的详细介绍;第 5 节论述了 CAR 事件编程模型的一些优点;第 6 节将其与现有的一些事件编程模型做了一个简单的比较;最后在第 7 节展望了 CAR 事件编程模型在未来的发展.

2 Windows 消息模型简介

写过 Windows 应用程序的人都知道,Windows 应用程序最基本的元素就是“窗口”(window).有些窗口很显而易见,比如应用程序的主窗口,一般都带有标题栏,菜单,工具栏,客户区域和状态栏等等.对话框也属于窗口的一种.还有一些窗口并不那么明显,比如按钮(button),单选框(radio box),复选框(check box),文本框(text-entry field)等等,它们被称作子窗口(child window)或控制窗口(control window).

Windows 以消息的方式向应用程序传递用户对窗

基金项目:国家高技术研究发展计划(863 计划)项目,项目编号:2001AA113400.

作者介绍:李辉,男,在读硕士研究生,清华大学深圳研究生院软件工程中心.

陈榕,男,清华大学信息技术研究院操作系统与中间件技术研究中心.

口的输入,比如鼠标点击,键盘上按键的按下与弹起等等.应用程序之间的通信也可以用传递消息的方式来进行.每个 Windows 应用程序可以有一个或多个窗口(有些特殊的应用程序没有窗口).Windows 为每个正在内存中运行的应用程序专门维护了一个消息队列(message queue),用户对该应用程序所属的所有窗口的输入,都被映射为相应的消息,存储在消息队列中.

在 Windows 应用程序中,每个窗口都伴随着一个窗口过程(window procedure),它负责对该窗口的所有消息进行相应的处理.如果一个 Windows 应用程序有多个窗口,相应的,也就有多个窗口过程,每个窗口过程负责处理相应窗口的消息,互不干扰.应用程序的主函数从消息队列中取得消息,然后将消息分发给相应的窗口过程,这是一个轮询的过程.窗口过程对不同的消息进行不同的处理.下面,我们以一个非常简单但是很典型的 Windows 应用程序中的代码片断来说明这一机制[1].

```
int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR lpCmdLine,
                     int nCmdShow)
{
    ……
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    ……
}
LRESULT CALLBACK WndProc(HWND hWnd,
                         UINT message,
                         WPARAM wParam,
                         LPARAM lParam)
{
    ……
    switch (message)
    {
    case WM_COMMAND:
        ……
        break;
    case WM_PAINT:
        ……
        break;
    case WM_DESTROY:
        ……
        break;
    }
    ……
}
```

以上代码片断中,WinMain 函数中的 while 循环被称为“消息循环”(message loop).WinMain 函数使用这段代码从消息队列中取得消息,并分发给窗口过程.我们可以很清楚的看到,所谓的消息,实际上是 Windows 内部定义的一种数据结构,Windows 用这种数据结构来记录用户对窗口进行了何种输入(鼠标点击,键盘活动等等),以及一些有关该输入的附加信息(鼠标当前的位置,键盘上的哪个键被按下等等).下面将消息循环中的函数逐一进行解释.

GetMessage——该函数负责从消息队列中读取消息,并将消息存储在 msg 中.

TranslateMessage——该函数将消息发回给 Windows,将一些键盘输入转化成用户程序可识别的数据接口.我们讨论的重点不在这里,故不作深入讨论.

DispatchMessage——该函数将消息发回给 Windows,由 Windows 对消息进行区分,然后发送给相应的窗口过程.

代码中,WndProc 函数就是所谓的窗口过程.可以看到,它前面用一个 CALLBACK 宏进行修饰,这表示这个函数是由操作系统来调用的.DispatchMessage 函数将消息发回给 Windows,并将控制权交给 Windows.这时 Windows 根据消息所属的窗口对消息进行区分,然后再分发给相应的窗口过程.本例中只有一个窗口过程,故 Windows 将消息发送给 WndProc,实际上就是对 WndProc 进行调用,这时,应用程序重新获得了控制权.WndProc 对不同的消息进行不同的处理,代码中,WM_COMMAND,WM_PAINT 这些宏是 Windows 预先定义好的消息类型,以方便应用程序对消息进行区分.当 WndProc 完成了对消息的处理后,便又将控制权交回给 Windows,然后,Windows 才让 DispatchMessage 函数返回.这样整个一个过程就是一轮消息循环.

从以上的说明中,我们可以很清楚的看到,Windows 的编程模型是一种基于消息机制的轮询模型.

3 Windows 消息编程模型的缺点

由于 Windows 编程模型是轮询模型,所以我们很自然地想到了效率问题.众所周知,轮询的效率是非常地下的,对系统资源占用很多.但是,这里有必要说明一下,Windows 编程模型中的轮询不同于一般意义的轮询,Windows 采用了一个很巧妙的手段来提高应用程序的效率.

当应用程序执行 GetMessage 函数时,如果消息队列为空,那么函数不会立即返回,而是阻塞.Windows 实际上是将当前线程挂起来实现阻塞的,这样,当前线程暂停执行,不会占用任何系统资源,使得有限的系统资源能够被其他线程有效利用.当有新消息进入消息队列时,Windows 将会唤醒被挂起的线程,这时 GetMessage 函数才会返回.

这样看来,似乎这样的轮询模型效率还是比较高的.但是如果遇到一些特殊的情况,效率问题就很明显的表现出来了.设想我们要编写一个应用程序,这个应用程序的窗口过程中要进行一些比较特殊的处理,需要消耗

很长时间,那么一次消息循环的周期便会很长.我们需要用户在按下 esc 键时停止当前的所有操作而立即退出应用程序,这时我们该如何设计应用程序呢?

由于窗口过程的处理时间较长,而用户需要对 esc 键进行及时的反应,所以,用户按下 esc 键这个消息不能放在窗口过程中处理.我们需要另外启动一个线程,在这个线程中,对消息推列中已存在的消息进行查询,如果查询到 esc 键按下的消息,就进行一些处理以结束应用程序.下面的代码片断展示了这一过程.

```
// in a new thread
while(PeekMessage(&msg, hwnd, 0, 0, PM_REMOVE))
{
    if (msg.message == WM_KEYDOWN
       && msg.wParam == VK_ESCAPE)
    {
        //Terminate the program
    }
}
```

PeekMessage 与 GetMessage 的不同之处在于:PeekMessage 是立即返回,不管有没有得到消息;而 GetMessage 是阻塞,直到得到消息才返回.由此可以看出,以上这段代码就是彻底的轮询了.为了检测用户是否按下 esc 键而使用这样的轮询,显然代价太大.

当然,对于这一问题,我们可以在 Windows 编程模型下提出更好的解决方法.比如把窗口过程中需要花费时间较长的操作放在一个新的线程中进行,这样就可以缩短消息循环的周期,而使应用程序对用户输入能够进行及时的反应.但是,这需要应用程序开发者对 Windows 的内部实现机制非常了解,适合于经验丰富的开发人员,而对于缺乏经验的开发人员来说,这无疑是一个很严重的问题.对于现代操作系统来说,应该使应用程序开发人员能够专注于应用本身的流程和逻辑,而不应该让开发人员花费过多的精力去关注与操作系统相关的程序结构设计.显然,Windows 的编程模型不能满足这一需求.

Windows 编程模型的这一缺点同时也体现在网络应用上.随着互联网的发展,大型的网络应用越来越多,并且现在的网络应用大多是 server/client 模式的.有时一个 server 需要和成百上千个 client 同时进行通信,如图 1 所示:

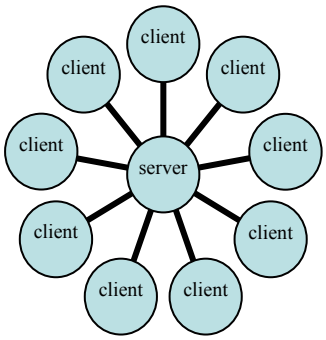

图 1 大型网络应用的 Server/Client 结构

在这样的体系结构下,server 需要处理所以 client 的请求,并且作为 client 之间通信的中转站.很显然,server 的负担是相当重的.在这种情况下,如果使用 Windows 的消息编程模型,引起的后果是显而易见的:

1. 并不是所有的消息都需要处理,应用程序实际上只需要针对其中的一部份消息进行处理,所以,发送过多无用的消息将会浪费网络带宽.

2. 如果每个 client 都对消息进行轮询,那么无疑会很大程度的增加 server 的负担,严重影响效率,这在一些效率要求很高的网络应用上(例如大型网络游戏)是决不能容忍的.

当然,也可以通过其他办法解决上述问题,比如用一些专用芯片来处理特殊应用.显然,这样做将会大大提高成本.对于现代操作系统来说,应该为应用开发人员提供最大程度的方便,所以,这些问题都应该在操作系统层面来解决.这样看来,Windows 的编程模型已经不能适应日新月异的应用需求,我们需要对他进行彻底的改进.

4 CAR 事件编程模型

CAR 的全称是 Carefree Application Runtime,是“和欣”上的运行时环境. “和欣”是科泰世纪有限公司独立开发的具有自主知识产权的操作系统.它是完全面向构件技术的操作系统,其主要功能模块完全基于构件技术,可以灵活拆卸,应用系统可以按照需要剪裁组装,或在运行时动态加载必要构件.CAR 编程模型完全体现了面向网络服务的构件化编程思想.关于 CAR 的其他信息,请读者参阅相关文献[2],本文不再详细论述.下面主要描述一下在 CAR 上开发应用程序时所使用的事件编程模型.

由于“和欣”是全面面向构件的操作系统,因此,用户在 CAR 上开发应用程序时,实际上就是编写能够实现各种功能的构件,并对这些构件加以调用.在以往的构件编程模型中,用户总是处于主动状态,而构件总是处以被动状态.对于一个全面的交互过程来说,这样的单向通信显然不能满足要求,有时候构件也需要主动同用户进行通信.比如编写一个简单的窗口程序,我们可以使用一个 Button 构件,调用 Button 构件中所提供的接口

将一个按钮显示在窗口上.那么当按钮被按下时,用户程序如何知道这一动作呢？这就需要 Button 构件主动与用户程序进行通信,这就引出了 CAR 编程模型中的“事件”(event)这一概念.

在传统的构件编程模型中(比如微软的COM)[3],用户可以为构件定义一组接口,用户程序对构件的调用完全通过接口来实现.而在 CAR 中,构件中可以有两种接口,一种与传统的接口类似,我们称之为普通接口(interface),也称为入接口;另一种接口我们称之为事件接口(events),也称为出接口.构件就是通过事件接口来与用户程序主动进行通信.

如果一个 CAR 对象支持一个或多个事件接口,那么这样的对象称为可连接对象(或可连接构件,connectable object),有时也称为源对象(或源构件,source).事件接口中每个成员函数代表一个事件(event).当特定事情发生时,如定时消息或用户鼠标操作发生时,构件对象产生一个事件,用户程序可以捕获并处理这些事件.构件中事件接口的成员函数并不由构件实现,而是由用户程序来实现.用户程序实现这些事件处理函数,并通过注册把函数指针告诉构件对象,对象在条件成熟时激发事件,回调事件处理函数.

下面,我们通过一个简单的例子来说明这一机制.该例子中对于描述这一机制用处不大的代码我们都省略掉了,只列出一些精简的代码,帮助读者清晰的理解这一事件机制.我们编写一个简单的构件 demo,构件中实现两个接口,一个普通接口,一个事件接口.首先要编写 demo.car 文件,这是 CAR 中的接口描述文件,类似于微软 COM 中的.idl 文件.

```
[
    version(1.0),
    uuid(d0b275bc-320b-41b4-a7b2-6ca76c41317c),
    urn(http://www.koretide.com/ezcom/demo.dll)
]
component demo
{
    [ uuid(b87bbdb5-f09d-4b68-ae38-96849e42d328) ]
    interface IDemoInterface        //普通接口
    {
        HRESULT Foo();
    }
    [ uuid(f867b977-56cc-4a8d-86be-d9ba2357b921) ]
    events IDemoEvents        //事件接口
    {
        HRESULT ICallYou();
    }
    [ uuid(47e3d31e-85da-4efa-ae8c-f00340a69cf0) ]
    class CDemo
    {
        interface IDemoInterface;
        events IDemoEvents;
    }
}
```

示例中,构件 demo 使用 interface 关键字定义了一个普通接口 IDemoInterface,使用 events 关键字定义了一个事件接口 IDemoEvents,为简洁起见,事件接口只声明了一个方法 ICallYou,并且没有任何参数.在实际应用中,我们可以为事件接口声明任意多个方法,每个方法都可以使用任意参数列表.类 CDemo 实现了 IDemoInterface 和 IDemoEvents 两个接口.

在 CAR 开发环境下编译.car 文件,开发环境会自动为用户生成应用代码框架(CDemo.h 与 CDemo.cpp 文件),用户只需要在代码框架中填写自己需要实现的代码即可.类 CDemo 的实现代码如下:

```
class CDemo : public _CDemo
{
public:
    STDMETHODIMP Foo();
    STDMETHODIMP OnAddICallYouHandler();
    STDMETHODIMP OnRemoveICallYouHandler();
};
```

其中灰色阴影部分是我们手动添加的代码,其余部分均由 CAR 开发环境自动生成.在 CAR 中,对于每一个事件,系统均提供了两个重载方法:OnAddXXXHandler 方法和 OnRemoveXXXHandler 方法,其中 XXX 是事件名.OnAddXXXHandler 方法只有当 XXX 事件在客户端第一次注册时调用,用户可以用它来实现一些初始化操作的代码;OnRemoveXXXHandler 方法只有当 XXX 事件在客户端最后一次注销时调用,用户可以用它来实现一些资源回收操作的代码.当用户没有对这两个方法进行重载时,则执行空操作.

由该例可以看出,构件并不需要实现事件接口当中的方法.前面已经讲过,事件方法是由调用构件的用户程序来实现的.这样做将构件需要实现的功能与用户程序需要实现的功能分开,各司其职,使代码更加模块化,清晰明了.

对于构件程序来说,并不知道事件方法是什么,它只需要激发事件.事件的激发过程和传递过程对于构件程序来说是完全透明的,对于用户程序来说也是透明的,这完全由 CAR 内部实现.用户只需要在运行时将事件处理函数注册到构件中就可以了.构件在适当的条件下可以激发事件.对于 XXX 事件,构件程序只需要调用 OnXXX 方法就可以激发事件.这种结构极大的方便了应用程序开发人员.对于上例,我们可以在 Foo 方法中激发 ICallYou 事件,下面代码是对 Foo 方法的简单实现.

```
HRESULT CDemo::Foo()
{
    int nRandNum = rand();
    if (nRandNum > 500)
    OnICallYou();
}
```

这段代码非常简单,产生一个随机数,当数值大于

500 时,就激发 ICallYou 事件.CDemo 类中的其他方法的实现在此省略.到此为止,demo 构件的实现工作就全部完成了.在 CAR 开发环境下将上述代码进行编译,可以得到 demo.dll 文件.这是一个包含自描述信息的可连接构件.我们想要使用这个构件,还需要编写段用户程序来调用它.在用户程序中,还要实现事件方法.该例中事件方法就是 ICallYou.下面是用户程序的代码.

```
#import <demo.dll>
#include <stdio.h>
HRESULT ICallYou(CDemoRef& cDmRef)
{
    printf("I was called.\n");
    return S_OK
}
int __cdecl main()
{
    CDemoRef cDmRef;
    cDmRef.Instantiate();
    if (!cDmRef.IsValid())
    {
        assert(0 && "Can't create CDemo.");
        return 1;
    }
    cDmRef.AddICallYouHandler(
        CDemoICallYouHandler(&ICallYou));
    cDmRef.Foo();
    cGnRef.RemoveICallYouHandler(
        CDemoICallYouHandler(&ICallYou));
    return 0;
}
```

从示例中可以看出,ICallYou 事件的处理函数是 ICallYou(CDemoRef& cDmRef).cDmRef 是 CDemo 类的智能指针,关于智能指针的概念,请读者参考 CAR 的其他文献[2],为节省篇幅,在此不再详述.cDmRef 参数可以指明事件的来源.比如程序中有好几个 CDemo 类的实例,cDmRef 可以指明事件是在哪个实例中被激发的.

在 CAR 中,提供了两个注册和注销事件处理函数的方法:AddXXXHandler 方法用于注册 XXX 事件的处理函数;RemoveXXXHandler 方法用户注销 XXX 事件的处理函数.注册和注销都是在运行时进行的,当用户需要某个事件处理函数来响应事件是,就可将其注册;反之,不需要对某一事件进行响应时,就可将已注册的该事件的处理函数注销.

在该例中,AddICallYouHandler 函数将 ICallYou 函数注册为 ICallYou 事件的处理函数,由于是第一次注册 ICallYou 事件,所以系统会调用 OnAddICallYouHandler 函数进行初始化.当调用 RemoveICallYouHandler 函数时,ICallYou 函数就被注销了,这时系统会调用 OnRemoveICallYouHandler 以进行一些资源回收工作.

在 CAR 中,一个事件可以注册多个处理函数.在实际应用中,这多个处理函数可以对事件进行不同的处理.事件处理函数的调用过程是由 CAR 内部实现的,对用户完全透明.CAR 会根据运行时注册事件处理函数的先后顺序来调用多个事件处理函数.这种机制使得用户对程序有更大的可控制性,比如在多个处理函数有先后逻辑关系的情况下,用户完全可以对此进行控制,使程序严格遵守应用逻辑,而不需要再增加其他的代码.

上面的代码简单说明了 CAR 的事件编程模型,还有许多细节问题,比如事件处理函数的参数传递问题,返回值问题等等.由于篇幅原因,就不在此详述了.使用这样一个简单例子主要是为了让读者从大的框架上对 CAR 事件编程模型有一个整体的认识,这样,我们就可以将 CAR 编程模型与其他编程模型做一些比较,突出其特点.

5 CAR 编程模型的特点

通过上一小节的论述,我们可以对 CAR 事件编程模型作一个总结.它的整体流程是这样的:

在构件中声明事件接口 —— 用户程序来实现接口处理函数 —— 用户在运行时动态注册事件处理函数 —— 构件对象在特定情况下激发事件 —— CAR 调用事件处理函数 —— 用户程序在运行时动态注销事件处理函数

我们可以将 CAR 事件编程模型与 Windows 消息编程模型作一个比较:

1. 在 Windows 消息编程模型中,应用程序主函数对消息进行轮询,将得到的消息发送到消息处理函数;而在 CAR 事件编程模型中,不需要对事件进行轮询,在某些情况下大大减轻了系统负担.

2. 在 Windows 消息编程模型中,消息是由用户程序主动去获取,消息的获取,传送都需要用户程序来实现;CAR 事件编程模型中,事件由构件激发,事件激发过程以及事件处理函数的调用过程都是由 CAR 内部完成的,对用户完全透明,用户可以完全不必关心这些代码的实现过程,给用户提供了极大的方便.

3. 消息模型中,用户程序需要取得所有的消息,不管这些消息是否需要处理,都需要将他们发送到消息处理函数,这在一定程度上浪费了系统资源;事件模型中,用户可以根据需要注册或注销事件处理函数,不需要处理的事件可以完全不必理会.这显然比消息模型更高效.

4. 消息模型中,消息处理函数在编译时就已经确定了,运行时无法更改;而在事件模型中,事件处理函数实在运行时注册和注销的.编译时和运行时是两个完全不同的状态,无疑,运行是比编译时遇有更大的灵活性.CAR 消息编程模型的这一特点,给应用程序设计带来了极大的灵活性,使应用设计人员能够更加游刃有余.

5. 从总体框架来看,用户程序在消息模型处于主动地位,而在事件模型中处于被动地位.如果把消息或

事件的发出者比作“我”,而把消息或事件的处理者比作“你”,那么在消息模型中,是“你”问“我”有没有事,如果没事, “你”就等待,如果有事“我”就让你去办事;而在事件模型中,是如果“我”有事,我会去找“你”办事.很显然,事件模型更加符合日常的逻辑,对于应用开发人员来说更加容易理解.

经过上述对 CAR 事件编程模型特点的论述,我们可以将这一模型扩展到现代大型网络应用上.我们可以回顾一下图 1 所示的网络应用的体系结构:一个 server 与多个 client 相连,这一数目可能达到成百上千,甚至上万.server 同时需要和所有的 client 进行通信,并且还要帮助 client 之间进行通信(作为 client 之间通信的中转站).显然,server 的负担是相当重的.

在这样的应用中,如果使用 Windows 消息编程模型,很明显,将会造成有用的与无用的消息“满天飞”的现象,严重时将会造成网络阻塞.如果使用 CAR 事件编程模型,我们只需要注册需要使用的事件.那些不需要使用的事件由于没有被注册,所以不会在事件发出者和事件处理者之间传递.这样,可以大大的节省网络带宽.

其次,如果使用 Windows 消息编程模型,在应用程序结构设计得不是很好的情况下,对消息的轮询会使 server 增加很大的负担,不能有效的利用系统资源;而 CAR 事件编程模型使用的是“我”去找“你”办事的模式,所以不会对系统宝贵的资源造成浪费,同时减轻了 server 的负担.

最后,CAR 是完全面向构件的编程模型,而现代大型网络应用往往包含非常巨大的代码量.在这种情况下,构件技术以其高度的模块化与封装性,很明显的显示出它的优势,给大型应用程序的开发和维护带来极大的便利.由此看来,CAR 编程模型比 Windows 编程模型更能适应现代应用的需求.

6 CAR 事件编程模型与其它事件编程模型的比较

微软在其 COM 技术中也引入了事件编程模型,并在.NET 中将其进一步完善.对于微软的事件编程模型,就不在此详述了,有兴趣的读者可以查阅相关资料.这里只将微软的事件编程模型与 CAR 事件编程模型做一比较.

在 COM 中,微软也提出了可连接对象技术,通过这一技术可以实现 COM 对象对客户程序的调用.用户需要自己实现 IConnectionPoint 接口以达到这一目的,为此,用户必须自己实现客户程序与构件对象的连接,事件的激发,接收器的编写等.显然,这给应用程序开发人员带来了极大的负担.而在 CAR 事件编程模型中,这些细节对用户来说是完全透明的.应用程序开发人员只需要关注应用本身的逻辑,完全可以把这些细节工作交给 CAR 环境来完成.由此看来,CAR 事件编程模型会加速应用程序的开发进程,从而节省了应用程序的开发成本.

在.NET Framework 中[4],微软创造了 delegate 关键字,并且提出了 Delegates & Events 模型.在这个模型下,可以实现完全基于事件的编程.可是需要说明的是,.NET Framework 中编译好的代码是一种中间代码(Intermediate Language code),而不是可以在硬件上运行的二进制代码.这种代码在运行时需要先经过 JIT(Just In Time) Compiler 进行编译,将其编译成 native code,然后再执行.也就是说,.NET Framework 的 PE 文件是一边编译一边执行的.显然这种方式会使运行效率受到影响.而在 CAR 环境中可以直接将 C++代码编译成 native code,便以好的程序是直接可在硬件上执行的二进制代码.所以,CAR 在运行效率上要远远高于.NET framework.

对于其他的一些事件编程模型,在此就不一一比较了,相信读者也应该对 CAR 编程模型有一个完整认识了.

7 结论与展望

通过以上的论述,我们可以看到,CAR 事件编程模型具有模块化,高效,便利等许多特点.而这些特点恰恰是在现代大型应用中非常重要的.另一方面,Windows 是一个很成功的操作系统,基于 Windows 消息模型的成千上万的应用程序也在很好的为人们服务.但是,CAR 事件编程模型的这些特点使它能在大型网络应用中能够发挥得更为出色.另外,在一些对于效率要求比较高的应用中,CAR 事件模型也会比 Windows 消息模型更容易达到要求.从软件工程的角度来看,CAR 模型的模块化可以使代码更加容易维护.因此,总体看来,CAR 事件编程模型应该能够给应用开发人员带来一片新天地.

参 考 文 献

1 Charles Petzold. Programming Windows, Fifth Edition [M]. Redmond: Microsoft Press, 1998.

2 科泰世纪有限公司. 《和欣 1.1》资料大全[EB/OL]. .http://www.koretide.com.cn/download.php?DownLoadID=3, 2004 年 3 月 31 日.

3 潘爱民. COM 原理与应用 [M]. 北京: 清华大学出版社, 1999

4 Thuan Thai, Hoang Q. Lam. .NET Framework Essentials, First Edition [M]. Cambridge Mss.: O’Reilly, June 2001.