使用 C#进行 CAN 总线编程

—— 基于 WINCE 平台 C#编程要点之三

英创 ARM9 系列嵌入式主板，如 EM9000、EM9260 均带有（或可选）CAN 总线接口，英创公司不仅提供了硬件平台支持，还提供了 CAN 总线通讯驱动程序。本文主要介绍在基于 Windows CE 平台的英创嵌入式主板下进行 C#（Microsoft Visual Stdio.Net 2005）CAN 总线应用程序开发时会常常用到的一些功能函数以及开发方法。

在英创嵌入式主板上进行 CAN 编程的思路是：

（1） 使用 Win32 的 CreateFile 方法(类似于传统操作串口的模式)来获得操作 CAN 总线端口的 Handle。

（2） 使用英创公司提供的 CAN 总线驱动程序动态链接库 CAN_API_DLL.dll 实现一系列的 CAN 通讯操作，包括：

CAN_StartChip、CAN_SetBaudRate、CAN_SetGlobalAcceptanceFilter、CAN_GetNextReceivedFrame、CAN_SendFrame、CAN_StartChip等方法。

（3） 使用 Win32 的 CloseHandle 方法关闭 CAN 操作的 Handle。

在使用C#编程操作CAN通讯之前，首先要明确：很多底层操作的函数（如CreateFile函数），Visual Studio 2005.NET的API库中并没有提供，这个时候，我们就要在C#开发中调用Win32的函数来进行相应的操作。一大批Win32底层操作的函数都存在于cordll.dll动态链接库中。

调用Win32的申明：

using System.Runtime.InteropServices;

要使用的两个Win32函数申明如下：

```
[DllImport("coredll.dll")]
public static extern uint CreateFile(
                                string FileName,    //file name
                                uint DesiredAccess, //access mode
                                uint ShareMode,     //share mode
                                uint SecurityAttributes,// Security Attributes
                                uint CreationDisposition, //how to create
                                uint FlagsAndAttributes, //file attributes
                                int hTemplateFile   //handle to template file
                                );

[DllImport("coredll.dll")]
static extern int CloseHandle(uint hDevice);
```

同样，英创公司提供的驱动程序动态链接库函数也需要进行申明如下：

```
        // 功能描述:读取CAN设备接收数据包。
        // 输入参数hDevice: 已创建CAN流式设备的句柄。
        // 输出参数pRxFrameBuffer: 用于读取CAN设备接收到的数据包。
        // 返回值= TRUE:  从CAN设备接收到数据，并将接收到的数据包读入pRxFrameBuffer。
        //        = FALSE: CAN设备没有接收数据。
        [DllImport("CAN_API_DLL.dll", EntryPoint =
"?CAN_GetNextReceivedFrame@@YAHPAXPATMessageFrame@@@Z")]
        public static extern bool CAN_GetNextReceivedFrame(uint hDevice, byte[] RxFrameBuffer );

        // 功能描述:通过CAN设备发送数据包。
        // 输入参数hDevice: 已创建CAN流式设备的句柄。。
        //          pTxFrameBuffer: 准备通过CAN设备发送的数据包。
        // 返回值= TRUE:  从CAN设备发送数据成功。
        //        = FALSE: 从CAN设备发送数据失败。
        [DllImport("CAN_API_DLL.dll", EntryPoint =
"?CAN_SendFrame@@YAHPAXPATMessageFrame@@@Z")]
        //public static extern bool CAN_SendFrame(uint hDevice, MessageFrame[] TxFrameBuffer);
        public static extern bool CAN_SendFrame(uint hDevice, byte[] TxFrameBuffer);

        // 功能描述:设置CAN设备通讯的波特率。
        // 输入参数hDevice: 已创建CAN流式设备的句柄。
        //             Index=CAN_TIMING_10K  : 10Kbps
        //                   CAN_TIMING_20K  : 20Kbps
        //                   CAN_TIMING_50K  : 50bps
        //                   CAN_TIMING_100K : 100bps
        //                   CAN_TIMING_125K : 125Kbps
        //                   CAN_TIMING_250K : 250Kbps
        //                   CAN_TIMING_500K : 500bps
        //                   CAN_TIMING_1000K: 1Mbps
        // 返回值= TRUE:  波特率设置成功。
        //        = FALSE: 波特率设置失败。
        [DllImport("CAN_API_DLL.dll", EntryPoint = "?CAN_SetBaudRate@@YAHPAXPAE@Z")]
        public static extern bool CAN_SetBaudRate(uint hDevice, byte[] index);

        // 功能描述:设置CAN设备通讯接收过滤器配置。
        // 输入参数hDevice: 已创建CAN流式设备的句柄。
        //             AcceptanceFilter: 根据通讯报文格式定义过滤器的配置，定义为个字节的过
滤器，其中前个字节用于定义过滤器的接收码，
        //                               后个字节用于定义过滤器的接收屏蔽码，最后一个字节用于定
义选择单/双滤波模式。
        //             size: 定义的过滤器的大小。
        // 返回值= TRUE:  配置设置成功。
        //        = FALSE: 配置设置失败。
        [DllImport("CAN_API_DLL.dll", EntryPoint =
```

```
"?CAN_SetGlobalAcceptanceFilter@@YAHPAXPAEE@Z")]
        public static extern bool CAN_SetGlobalAcceptanceFilter(uint hDevice, byte[]
AcceptanceFilter, byte size);

        // 功能描述:启动CAN设备端口。
        // 输入参数hDevice: 已创建CAN流式设备的句柄。
        // 返回值= NULL: 启动CAN设备端口失败。
        //       != NULL: 启动CAN设备端口返回的句柄。
        [DllImport("CAN_API_DLL.dll", EntryPoint = "?CAN_StartChip@@YAHPAX@Z")]
        public static extern bool CAN_StartChip(uint hDevice);

        // 功能描述:停止CAN设备端口。
        // 输入参数hDevice: 已创建CAN流式设备的句柄。
        // 返回值= TRUE:  停止CAN设备端口成功。
        //        = FALSE: 停止CAN设备端口失败。
        [DllImport("CAN_API_DLL.dll", EntryPoint = "?CAN_StopChip@@YAHPAX@Z")]
        public static extern bool CAN_StopChip(uint hDevice);
```

一、启动CAN接口

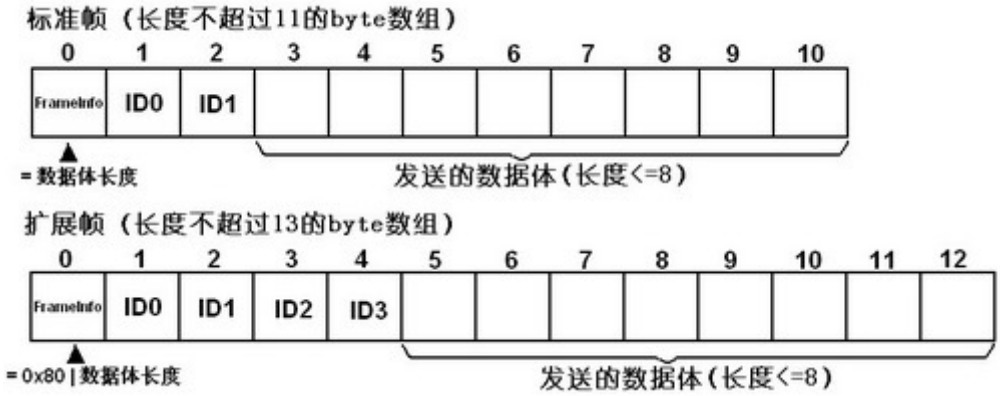
启动CAN接口一般只需要按顺序进行如下四个函数的调用：

```
CANhandle = CreateFile(strPort, GENERIC_READ|GENERIC_WRITE, 0, 0, OPEN_EXISTING,
0, 0);  //注意：strPort是一个字符串，定义CAN端口的名称，如“CAN1:”或“CAN2:”
CAN_StartChip(CANhandle);      //如下三个函数的定义请见上文注释
CAN_SetBaudRate(CANhandle, BaudrateSerialno);
CAN_SetGlobalAcceptanceFilter(CANhandle, ACCFilter, 9);
```

注意：ACCFilter是一个长度为9的字符串数组，请在调用之前安装相关设置进行初始化。

二、数据发送

启动CAN接口后，就可进行CAN通讯了。CAN的数据帧分标准帧和扩展帧两种，其区别如下图所示：



根据如上格式将发送的帧打包成byte数组后，即可通过如下函数进行发送：

```
bResult = CAN_SendFrame(CANhandle, TxBuf); //bResult是bool变量判断发送成功与否，TxBuff就是打包后的帧byte数组。
```

三、数据接收

可以通过在程序里设置一个Timer或直接启动一个线程来监视是否有CAN数据帧发送进来，并进行接收。在Timer启动的事件函数或线程函数中使用如下函数接收数据：

```
bResult = CAN_GetNextReceivedFrame(CANhandle, RxBuf); //bResult是bool变量判断收到数据与否，TxBuff就是收到后的帧byte数组。
```

需要注意的是函数CAN_GetNextReceivedFrame每执行一次，只是读取了一帧CAN数据报文，而收到的数据可能往往有不止一帧，客户应当在应用程序中将最新的数据全部读出，这个时候需要反复调用该函数，直到该函数的返回值为false。

建议在接收帧的时候将数据先放到一个自定义的buffer中，然后在应用程序的其它部分进行处理，可以定义两个全局变量：一个int类型存储一次接收到帧的数量（如int mycount），一个较长的byte数组（或二维数组）作为自定义buffer存储接收到的帧（如byte[,] myBuffer = new byte [20,13]。接收部分的示例代码如下：

```
byte[] RxBuf = new byte[13];
byte dlen;
bool bresult = true;
mycount = 0;
while (bresult)
{
    bresult = CAN_GetNextReceivedFrame(CANhandle, RxBuf);
    if (bresult == false) return;
    for (int j = 0; j < 13; j++)
    {
        myBuffer[mycount, j] = RxBuf[j];
    }
    Mycount ++;
 }
```

收到数据后，可以通过判断FrameInfo位的高位确定是标准帧还是扩展帧，并获得数据体的长度，进而将数据分离出来进行处理。

四、关闭CAN接口

关闭CAN接口一般只需要按顺序进行如下二个函数的调用即可实现：

```
CAN_StopChip(CANhandle);
CloseHandle(CANhandle);
```