**浅析GDAL库C#版本支持中文路径问题**

发布者：[李小舌](http://www.cfanz.cn/index.php?c=uc/main&id=3802)

时间：2013-07-25 11:40:21

GDAL库对于C#的支持问题还是蛮多的，对于中文路径的支持就是其中之一（另一个就是通过OGR库获取图形的坐标信息）。

关于C#支持中文路径，看过我之前博客的应该都不陌生，如果使用的是我修改过的GDAL库，可以通过设置下面的代码即可让C#直接支持中文路径。如果使用官方的库，不用设置直接应该就可以支持中文路径。

    // 注册所有的驱动

    Ogr.RegisterAll();

    // 为了支持中文路径，请添加下面这句代码

    OSGeo.GDAL.Gdal.SetConfigOption("GDAL\_FILENAME\_IS\_UTF8","YES");

    // 为了支持shp属性表字段支持中文，请添加下面这句

   OSGeo.GDAL.Gdal.SetConfigOption("SHAPE\_ENCODING","");

         昨天，一位朋友说，他测试C#版本，发现中文路径有时候可以，有时候不可以，通过设置GDAL\_FILENAME\_IS\_UTF8也无济于事。

今天通过测试发现，**只要是文件名中的汉字个数是偶数，完全没有影响，读取和创建都正常，如果文件名中的汉字个数是奇数，肯定不能读取和创建。**

比如下面的文件名就是正常的：

    D:\\新建文件夹\\新建1.shp

    D:\\密云数据\\线分离的0.shp

而下面的肯定就是不行：

    D:\\新建文件夹\\新建的1.shp

    D:\\密云数据\\线分离0.shp

下面就通过C#程序调试GDAL库，找找原因。按照上篇博客中的跨语言调试的方式，在C#程序中的Open函数处设置断点，然后启动调试，程序在此处中断。

首先用一个GDAL库可以打开的正常路径进行测试，如下图所示。



接下来按F11键，进入swig封装的C#代码中，如下图所示。



在这里，我们发现了这样的代码。

  public static DataSourceOpen(string utf8\_path, int update) {

    IntPtr cPtr =OgrPINVOKE.Open(System.Text.Encoding.Default.GetString(System.Text.Encoding.UTF8.GetBytes(utf8\_path)),update);

    DataSource ret = (cPtr ==IntPtr.Zero) ? null : new DataSource(cPtr, true, ThisOwn\_true());

    if(OgrPINVOKE.SWIGPendingException.Pending) throwOgrPINVOKE.SWIGPendingException.Retrieve();

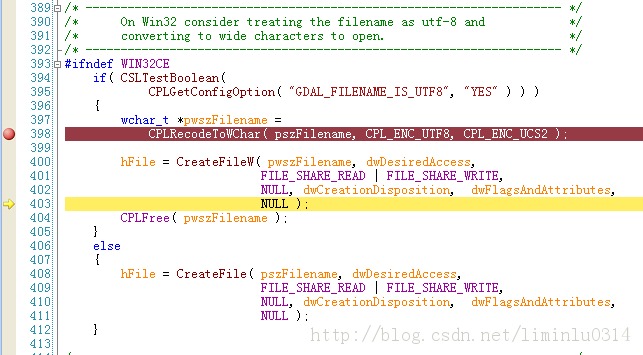
    return ret;

  }

           其中在调用OgrPINVOKE时，将路径进行了编码转换，核心代码如下：

System.Text.Encoding.Default.GetString(System.Text.Encoding.UTF8.GetBytes(utf8\_path))

          从代码可以看出，Swig首先将C#默认的字符串，使用UTF8的编码转换为默认的编码。上面的路径“D:\新建文件夹\新建1.shp”通过这句转换之后就变成了“D:\鏂板缓鏂囦欢澶筡鏂板缓1.shp”。而这个字符串传入GDAL库后，在文件gdal-1.10.0\port\cpl\_vsil\_win32.cpp中的函数 VSIVirtualHandle\*VSIWin32FilesystemHandler::Open( const char \*pszFilename,   const char \*pszAccess )中又进行了一次编码转换。如下图所示。

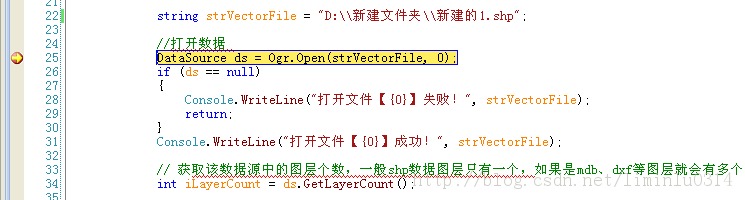


通过上图，可以发现，如果设置了GDAL\_FILENAME\_IS\_UTF8=YES时，系统先将编码从UTF8转为UCS2编码。通过这句之后，发现路径又编程了原来的，如下图：



这样GDAL库就可以正常打开该文件。下面再看一个GDAL不能打开的路径重复上面的步骤，下面只截取关键位置的截图。

首先是在打开时设置断点，文件路径为“**D:\新建文件夹\新建的1.shp**”



然后传入GDAL库中的路径通过转码变成了“**D:\鏂板缓鏂囦欢澶筡鏂板缓鐨?.shp**”。之后再通过GDAL库中的函数转为宽字节时称为了“**D:\新建文件夹\新建çš?.shp**”。如下图所示。



只要路径中出现了问号（?），这个路径肯定有问题，不管是不是乱码。所以这个路径肯定就打不开了。

通过上面的步骤，我们可以确定，C#的路径是好使的，而通过SWIG中的编码转换后就出现了问题，所以我们可以认为是编码转换出现的问题。

在SWIG封装的接口中，使用System.Text.Encoding.Default.GetString(System.Text.Encoding.UTF8.GetBytes(utf8\_path))进行转换，下面针对此代码片段写一个简单的测试代码进行验证。

staticvoid Main(string[] args)

{

       string strUtf8 = "D:\\新建文件夹\\新建的1.shp";

       byte[] byutf8 =System.Text.Encoding.UTF8.GetBytes(strUtf8);

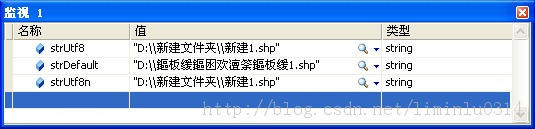
       string strDefault =System.Text.Encoding.Default.GetString(byutf8);

       byte[] byDefault =System.Text.Encoding.Default.GetBytes(strDefault);

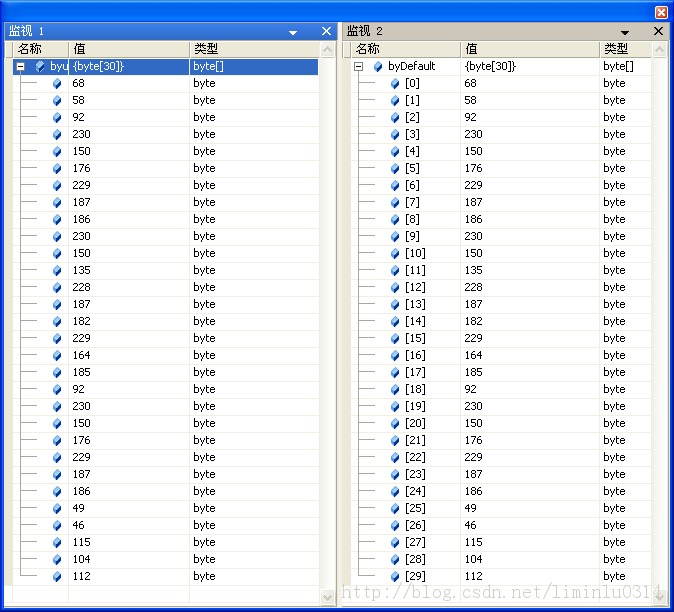
       string strUtf8n = System.Text.Encoding.UTF8.GetString(byDefault);

}

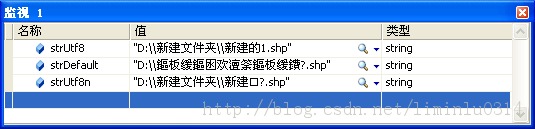
          首先看一个GDAL可以正常访问的路径，首先查看转换后再转回来，共三个字符串的对比，如下图，从图中可以看出，转换为Default再转为utf8之后，与原来的路径一样。所以GDAL库可以正常访问。



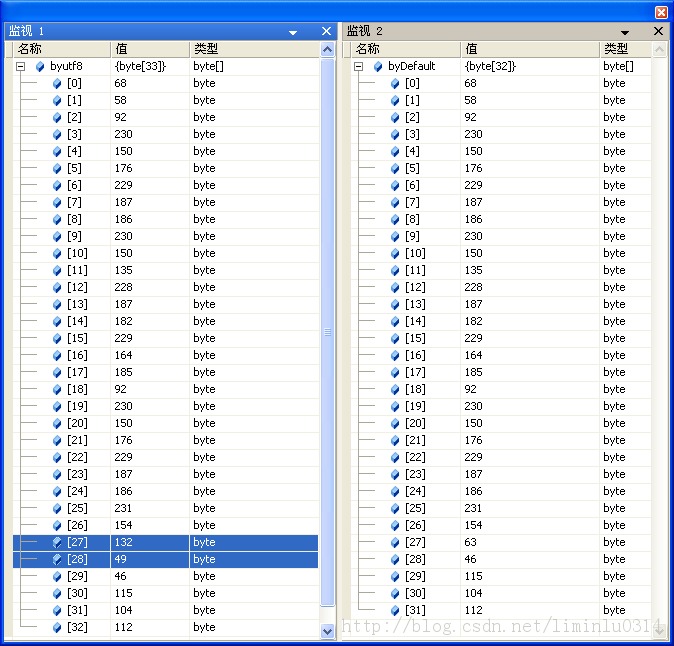
而转换前后获取的byte数组内容完全一致，如下图所示：



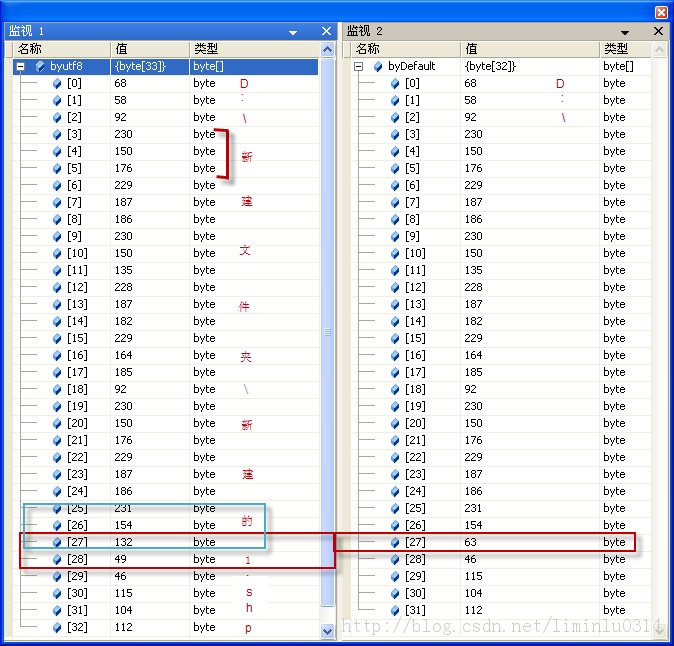
下面再使用一个GDAL不能访问的路径进行测试，查看转换后再转回来，共三个字符串的对比，如下图，从图中可以看出，转换为Default再转为utf8之后，与原来的路径发生了变化。



下面比较两次转换的byte数组，按理说内存中的byte数组应该是一样的，下面对比两个byte数组中的内容，如下图所示，从图中可以发现，数组转换前的27和28分别是132和49，而转换后，这两个字节变成了一个字节（63）。



从这里可以看出，可以认为问题就出在此处。对应ASCII码表，将上图中的值转为字符串，可以得到下面的图。英文字符占用一个byte，而汉字占用3个 byte。而在转码的时候应该是两个字节为一组进行转码处理，也就是说对于偶数个汉字，转成byte是3倍的偶数，结果肯定是偶数，所以按照两个字节转码刚好可以转完；而汉字为奇数个，转成byte是3倍的奇数，结果肯定是个奇数，按照两个字节转码，肯定会多出来一个，这多出来的一个系统可能不认识就用问号（?）来表示了。



所以，可以这么认为，汉字是偶数的就正常，奇数的就会出现问题，与GDAL表现的结果完全一致。上面的最后这一段的是我个人的分析，不代表微软内部就是这么实现的。或许这可能算作C#的一个bug？不知道微软有没有发现这个问题。

来源： <<http://www.cfanz.cn/index.php?c=article&a=read&id=103228>>