عنوان: **استخراج متن از فایل‌های PDF توسط iTextSharp**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۰۱ ۱۲:۵۳
آدرس: www.dotnettips.info
گروه‌ها: iTextSharp, PDF

پیشنیاز

نحوه ذخیره شدن متن در فایل‌های PDF

حتما نیاز است پیشنیاز فوق را یکبار مطالعه کنید تا علت خروجی‌های متفاوتی را که در ادامه ملاحظه خواهید نمود، بهتر مشخص شوند. همچنین فایل PDF ایی که مورد بررسی قرار خواهد گرفت، همان فایلی است که توسط متد writePdf ذکر شده در پیشنیاز تهیه شده است.

دو کلاس متفاوت برای استخراج متن از فایل‌های PDF در iTextSharp وجود دارند:

الف) SimpleTextExtractionStrategy

```
using System.Diagnostics;
using System.IO;
using iTextSharp.text;
using iTextSharp.text.pdf;
using iTextSharp.text.pdf.parser;

namespace TestReaders
{
    class Program
    {
        private static void readPdf1()
        {
            var reader = new PdfReader("test.pdf");
            int intPageNum = reader.NumberOfPages;
            for (int i = 1; i <= intPageNum; i++)
            {
               var text = PdfTextExtractor.GetTextFromPage(reader, i, new
SimpleTextExtractionStrategy());
                File.WriteAllText("page-" + i + "-text.txt", text);
            }
            reader.Close();
        }

        static void Main(string[] args)
        {
            readPdf1();
        }
    }
}
```

مثال فوق، متن موجود در تمام صفحات یک فایل PDF را در فایل‌های txt جداگانه‌ای ثبت می‌کند. برای نمونه اگر از PDF پیشنیاز یاد شده استفاده کنیم، خروجی آن به نحو زیر خواهد بود:

```
Test
ld Wor llo He
Hello People
```

علت آن نیز پیشتر بررسی گردید. متن، در این فایل ویژه در مختصات خاصی ترسیم شده است. حاصل از دیدگاه خواننده نهایی بسیار خوانا است؛ اما خروجی hello world متنی جالبی از آن استخراج نمی‌شود. SimpleTextExtractionStrategy دقیقا بر اساس همان عملگرهای Tj و همچنین منابع صفحه، عبارات را یافته و سر هم می‌کند.

ب) LocationTextExtractionStrategy

همان مثال قبل را درنظر بگیرید، اینبار به شکل زیر:

```
private static void readPdf2()
        {
            var reader = new PdfReader("test.pdf");
            int intPageNum = reader.NumberOfPages;
            for (int i = 1; i <= intPageNum; i++)
            {
                var text = PdfTextExtractor.GetTextFromPage(reader, i, new
LocationTextExtractionStrategy());
                File.WriteAllText("page-" + i + "-text.txt", text);
            }
            reader.Close();
        }
```

کلاس LocationTextExtractionStrategy هوشمندتر عمل کرده و بر اساس عملگرهای هندسی یک فایل PDF، سعی می‌کند جملات و حروف را کنار هم قرار دهد و در نهایت خروجی متنی بهتری را تولید کند. برای نمونه اینبار خروجی متنی حاصل به صورت زیر خواهد بود:

```
Test
Hello World
Hello People
```

این خروجی با آنچه که در صفحه نمایش داده می‌شود تطابق دارد.

استخراج متون فارسی از فایل‌های PDF توسط iTextSharp

روش‌های فوق با PDFهای فارسی هم کار می‌کنند اما خروجی حاصل آن مفهوم نیست و نیاز به پردازش ثانوی دارد. ابتدا مثال زیر را درنظر بگیرید:

```
static void writePdf2()
        {
            using (var document = new Document(PageSize.A4))
            {
                var writer = PdfWriter.GetInstance(document, new FileStream("test.pdf",
FileMode.Create));
                document.Open();

                FontFactory.Register("c:\\windows\\fonts\\tahoma.ttf");
                var tahoma = FontFactory.GetFont("tahoma", BaseFont.IDENTITY_H);

                ColumnText.ShowTextAligned(
                            canvas: writer.DirectContent,
                            alignment: Element.ALIGN_CENTER,
                            phrase: new Phrase("تست می‌شود", tahoma),
                            x: 100,
                            y: 100,
                            rotation: 0,
                            runDirection: PdfWriter.RUN_DIRECTION_RTL,
                            arabicOptions: 0);
            }

            Process.Start("test.pdf");
        }
```

از متد فوق، برای تولید یک فایل PDF که متنی فارسی را نمایش می‌دهد استفاده خواهیم کرد. اگر متد readPdf2 را که به همراه LocationTextExtractionStrategy تعریف شده است، بر روی فایل حاصل فراخوانی کنیم، خروجی آن به صورت زیر خواهد بود:

```
ïº©ï»®ïº·ï»²ï»£ ïº–ïº´ïº–
```

برای تبدیل آن به یونیکد خواهیم داشت:

```
private static void readPdf2()
```

```
        {
            var reader = new PdfReader("test.pdf");
            int intPageNum = reader.NumberOfPages;
            for (int i = 1; i <= intPageNum; i++)
            {
                var text = PdfTextExtractor.GetTextFromPage(reader, i, new
LocationTextExtractionStrategy());
                text = Encoding.UTF8.GetString(Encoding.UTF8.GetBytes(text));
                File.WriteAllText("page-" + i + "-text.txt", text, Encoding.UTF8);
            }
            reader.Close();
        }
```

اکنون خروجی ثبت شده در فایل متنی حاصل به صورت زیر است:

```
دوشيم تست
```

دقیقا به همان نحوی است که iTextSharp و اکثر تولید کننده‌های PDF فارسی از آن استفاده می‌کنند و اصطلاحا چرخاندن حروف یا تولید Glyph mirrors صورت می‌گیرد. روش‌های زیادی برای چرخاندن حروف وجود دارند. در ادامه از روشی استفاده خواهیم کرد که خود ویندوز در کارهای داخلی‌اش از آن استفاده می‌کند:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Runtime.InteropServices;
using System.Security;

namespace TestReaders
{
    [SuppressUnmanagedCodeSecurity]
    class GdiMethods
    {
        [DllImport("GDI32.dll")]
        public static extern bool DeleteObject(IntPtr hgdiobj);

        [DllImport("gdi32.dll", CharSet = CharSet.Auto, SetLastError = true)]
        public static extern uint GetCharacterPlacement(IntPtr hdc, string lpString, int nCount, int
nMaxExtent, [In, Out] ref GcpResults lpResults, uint dwFlags);

        [DllImport("GDI32.dll")]
        public static extern IntPtr SelectObject(IntPtr hdc, IntPtr hgdiobj);
    }

    [StructLayout(LayoutKind.Sequential)]
    struct GcpResults
    {
        public uint lStructSize;
        [MarshalAs(UnmanagedType.LPTStr)]
        public string lpOutString;
        public IntPtr lpOrder;
        public IntPtr lpDx;
        public IntPtr lpCaretPos;
        public IntPtr lpClass;
        public IntPtr lpGlyphs;
        public uint nGlyphs;
        public int nMaxFit;
    }

    public class UnicodeCharacterPlacement
    {
        const int GcpReorder = 0x0002;
        GCHandle _caretPosHandle;
        GCHandle _classHandle;
        GCHandle _dxHandle;
        GCHandle _glyphsHandle;
        GCHandle _orderHandle;

        public Font Font { set; get; }

        public string Apply(string lines)
        {
            if (string.IsNullOrWhiteSpace(lines))
                return string.Empty;
```

```
        return Apply(lines.Split('\n')).Aggregate((s1, s2) => s1 + s2);
    }

    public IEnumerable<string> Apply(IEnumerable<string> lines)
    {
        if (Font == null)
            throw new ArgumentNullException("Font is null.");

        if (!hasUnicodeText(lines))
            return lines;

        var graphics = Graphics.FromHwnd(IntPtr.Zero);
        var hdc = graphics.GetHdc();
        try
        {
            var font = (Font)Font.Clone();
            var hFont = font.ToHfont();
            var fontObject = GdiMethods.SelectObject(hdc, hFont);
            try
            {
                var results = new List<string>();
                foreach (var line in lines)
                    results.Add(modifyCharactersPlacement(line, hdc));
                return results;
            }
            finally
            {
                GdiMethods.DeleteObject(fontObject);
                GdiMethods.DeleteObject(hFont);
                font.Dispose();
            }
        }
        finally
        {
            graphics.ReleaseHdc(hdc);
            graphics.Dispose();
        }
    }

    void freeResources()
    {
        _orderHandle.Free();
        _dxHandle.Free();
        _caretPosHandle.Free();
        _classHandle.Free();
        _glyphsHandle.Free();
    }

    static bool hasUnicodeText(IEnumerable<string> lines)
    {
        return lines.Any(line => line.Any(chr => chr >= '\u00FF'));
    }

    void initializeResources(int textLength)
    {
        _orderHandle = GCHandle.Alloc(new int[textLength], GCHandleType.Pinned);
        _dxHandle = GCHandle.Alloc(new int[textLength], GCHandleType.Pinned);
        _caretPosHandle = GCHandle.Alloc(new int[textLength], GCHandleType.Pinned);
        _classHandle = GCHandle.Alloc(new byte[textLength], GCHandleType.Pinned);
        _glyphsHandle = GCHandle.Alloc(new short[textLength], GCHandleType.Pinned);
    }

    string modifyCharactersPlacement(string text, IntPtr hdc)
    {
        var textLength = text.Length;
        initializeResources(textLength);
        try
        {
            var gcpResult = new GcpResults
            {
                lStructSize = (uint)Marshal.SizeOf(typeof(GcpResults)),
                lpOutString = new String('\0', textLength),
                lpOrder = _orderHandle.AddrOfPinnedObject(),
                lpDx = _dxHandle.AddrOfPinnedObject(),
                lpCaretPos = _caretPosHandle.AddrOfPinnedObject(),
                lpClass = _classHandle.AddrOfPinnedObject(),
                lpGlyphs = _glyphsHandle.AddrOfPinnedObject(),
                nGlyphs = (uint)textLength,
                nMaxFit = 0
            };
```

```
                var result = GdiMethods.GetCharacterPlacement(hdc, text, textLength, 0, ref gcpResult,
GcpReorder);
                return result != 0 ? gcpResult.lpOutString : text;
            }
            finally
            {
                freeResources();
            }
        }
    }
}
```

از کلاس فوق در هر برنامه‌ای که راست به چپ را به نحو صحیحی پشتیبانی نمی‌کند، می‌توان استفاده کرد؛ خصوصا برنامه‌های گرافیکی.

در اینجا برای اصلاح متد readPdf2 خواهیم داشت:

```
private static void readPdf2()
        {
            var reader = new PdfReader("test.pdf");
            int intPageNum = reader.NumberOfPages;
            for (int i = 1; i <= intPageNum; i++)
            {
                var text = PdfTextExtractor.GetTextFromPage(reader, i, new
LocationTextExtractionStrategy());
                text = Encoding.UTF8.GetString(Encoding.UTF8.GetBytes(text));
                text = new UnicodeCharacterPlacement
                {
                    Font = new System.Drawing.Font("Tahoma", 12)
                }.Apply(text);
                File.WriteAllText("page-" + i + "-text.txt", text, Encoding.UTF8);
            }
            reader.Close();
        }
```

اگر خروجی متد اصلاح شده فوق را بررسی کنیم، دقیقا به «تست می‌شود» خواهیم رسید.

سؤال: آیا این روش با تمام PDFهای فارسی کار می‌کند؟

پاسخ: خیر! همانطور که در پیشنیاز مطلب جاری عنوان شد، در یک حالت خاص، PDF writer می‌تواند شماره Glyphها را کاملا عوض کرده و در فایل PDF نهایی ثبت کند. خروجی حاصل در برنامه Adobe reader خوانا است، چون نمایش را بر اساس اطلاعات هندسی Glyphها انجام می‌دهد؛ اما خروجی متنی آن به نوعی obfuscated است چون مثلا حرف A آن به کاراکتر مرسوم دیگری نگاشت شده است.

نظرات خوانندگان

نویسنده: ابراهیم بیاگوی
تاریخ: ۱۳۹۱/۱۰/۰۲ ۱۴:۳۱

بسیار عالی هست، بسیار بسیار عالی. فقط یک سوال، راه حل بدون P/Invoke هم در حال حاضر سراغ دارید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۰۲ ۱۴:۳۸

در مورد پیاده سازی «UnicodeCharacterPlacement»؟
بله. یک نمونه جاوا اسکریپتی هست که به سادگی قابل تبدیل به سی شارپ خالص است (mirror Glyphs را پیاده سازی کرده):
bidi.js

نویسنده: ابراهیم بیاگوی
تاریخ: ۱۳۹۱/۱۰/۰۲ ۱۴:۴۲

عالی. ممنون

نویسنده: هیمن روحانی
تاریخ: ۱۳۹۲/۱۱/۰۱ ۱۵:۳۸

برای استخراج متن و عکس و رندر کردن می‌تونید از این کتابخانه PdfLib.Net با چند خط کد، متن کامل فارسی رو بدون مشکل استخراج کنید. البته حجم این کتابخونه یکم زیاده چون کار اصلیش رندر کردن PDF.

```
PDFLibNet.PDFWrapper wrapper = new PDFLibNet.PDFWrapper();
            wrapper.LoadPDF(pdfPath);
            string page1Text = wrapper.Pages[1].Text;
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۰۱ ۱۶:۶

از MuPDF هم استفاده می‌کنه (کد خالص دات نتی نیست و استفاده ازش در برنامه‌های وب مشکل ساز هست؛ نیاز به فول تراست دارد و همچنین 32 بیتی و 64 بیتی آن باید بر اساس نوع سرور لحاظ شود).