

1. Summary

1.1. C#

1.1.1. 采用string.Format格式化日期

Demo

```
DateTime dt = new DateTime(2017,4,1,13,16,32,108);
string.Format("{0:y yy yyy yyyy}",dt); //17 17 2017 2017
string.Format("{0:M MM MMM MMMM}", dt); //4 04 四月 四月
string.Format("{0:d dd ddd dddd}", dt); //1 01 周六 星期六
string.Format("{0:t tt}", dt); //下 下午
string.Format("{0:H HH}", dt); //13 13
string.Format("{0:h hh}", dt); //1 01
string.Format("{0:m mm}", dt); //16 16
string.Format("{0:s ss}", dt); //32 32
string.Format("{0:F FF FFF FFFF FFFFF FFFFFFF FFFFFFFF}", dt); //1 1 108 108 108
108 108
string.Format("{0:f ff fff ffff fffff fffffff ffffffff}", dt); //1 10 108 1080 10800
108000 1080000
string.Format("{0:z zz zzz}", dt); //+8 +08 +08:00
string.Format("{0:yyyy/MM/dd HH:mm:ss.fff}",dt); //2017/04/01 13:16:32.108
string.Format("{0:yyyy/MM/dd dddd}", dt); //2017/04/01 星期六
string.Format("{0:yyyy/MM/dd dddd tt hh:mm}", dt); //2017/04/01 星期六 下午 01:16
string.Format("{0:yyyyMMdd}", dt); //20170401
string.Format("{0:yyyy-MM-dd HH:mm:ss.fff}", dt); //2017-04-01 13:16:32.108
除去string.Format()可以对日期进行格式化之外，*.ToString()也可以实现相同的效果：
DateTime dt = new DateTime(2017,4,1,13,16,32,108);
dt.ToString("y yy yyy yyyy");//17 17 2017 2017
```

1.1.2. 转换数据类型，失败可指定默认值

```
public static int ToInt32(object value, int defaultValue = 0)
{
    if (value == null)
    {
        return defaultValue;
    }
    int result;
    if (!int.TryParse(value.ToString(), out result))
    {
        result = defaultValue;
    }
    return result;
}
```

1.1.3. C# winform窗体间传值

1. 方式一： 使用公共静态变量传值

主窗体frmMain中代码

```
public partial class frmMain : Form
{
    //声明工位ID 为公共静态变量
    public static string terminalID = "";
    //给静态变量赋值
    terminalID = "q13bh01-bh12";
}
```

子窗体frmGroup中代码

```
private void frmGroup_Load(object sender, EventArgs e)
{
    this.txtTerminalID.Text= frmMain.terminalID.Trim();
    //可以再赋值给静态成员，方便其他窗体调用
    frmMain.terminalID = "q13bh01-bh11";
}
```

特点： 双向传值，实现简单 缺点： 静态变量在类加载的时候分配内存，存储于方法区，一般不会被销毁，在系统不够内存情况下会自动回收静态内存，这样就会引起访问全局静态错误。

2. 使用公共变量传值

主窗体frmMain中代码

```
public partial class frmMain : Form
{
    //声明工位ID 为公共变量
    public string terminalID = "";
    //给变量赋值
    terminalID = "q13bh01-bh12";
    //单击‘行为’按钮的时候会给窗体传值
    private void btnGroup_Click(object sender, EventArgs e)
    {
        frmGroup frmGro = new frmGroup();
        //变量传值 ， 注意顺序写在ShowDialog()方法之前
        frmGro .stationID = this.terminalID;
        frmGro .ShowDialog();
    }
}
```

子窗体frmGroup中代码

```
public partial class frmGroup : Form
{
    //定义公共属性
    public string stationID = "";
}
```

特点： 单向传值，只能主窗体给予子窗体传值，实现简单

3. 使用委托传值

```
namespace Siemens.Simatic.GUIClient.MESClient
{
    //1、声明一个委托
    public delegate void setTextValue(string textValue, bool flag);
    public partial class frmGroup : Form
    {
        //2、声明一个委托类型的事件
        public event setTextValue setFormTextValue;
        public string groupName = "";
        public bool flagBtnGroup = false;
        public frmGroup()
        {
            InitializeComponent();
        }
        //轮询‘行为’按钮（相当于按钮单击事件）
        private void tmrBtn_Tick(object sender, EventArgs e)
        {
            if (sender is ButtonX) {
                ButtonX butX = (ButtonX)sender; //判断触发事件的是不是Button
                groupName = butX.Text.Trim();
                flagBtnGroup = true;
                //3、准备要回传的数据。
                setFormTextValue(this.groupName.Replace(" ", ""),
                this.flagBtnGroup );
                this.Close();
                return;
            }
        }
    }
}
```

主窗体

```
private void btnGroup_Click(object sender, EventArgs e)
{
    frmGroup frmGro = new frmGroup();
    //4、初始化事件
    frmGro .setFormTextValue += new setTextValue(frmGro
```

```

_setFormTextValue);
    //变量传值 ， 注意顺序写在ShowDialog()方法之前
    frmGro .stationID = this.terminalID;
    frmGro .ShowDialog();
}
//5、事件具体实现
public void frmGro _setFormTextValue(string textValue,bool flag)
{
    this.newGroupName = textValue;
    this.flagBtnGroup = flag;
    if (!string.IsNullOrEmpty(newGroupName))
    {
        .....
    }
}
}

```

1.1.4. Winfrom文本框不能输入中文或特殊字符

看一下控件的ImeMode属性是否被设为Disable，如果是，设为NoControl或者On试下Enable IME for the control imeMode已经是NoControl了 改成On后是可以输入了 但默认是全角的TextBox.IMEMode 属性（访问）

语法

表达式。IMEMode

表达式_一个表示文本框对象的变量。

注解

IMEMode属性使用以下设置。

No Control 不设置“日文汉字转换模式”（默认值）。

On 打开“日文汉字转换模式”。

Off 关闭“日文汉字转换模式”。

Disable 禁用“日文汉字转换模式”。

Hiragana 设置全角平假名。

Full pitch Katakana 设置全角片假名。

Half pitch Katakana 设置半角片假名。

Full pitch Alpha/Num 设置全角字母/数字。

Half pitch Alpha/Num 设置半角字母/数字。

HangulFull 设置全角 Hangul。

Hangul 设置半角 Hangul。

当焦点切换到控件通过设置IMEMode属性，可以指定日文汉字转换模式。如果使用了无控制（默认值），切换到该控件之前焦点设置到组。对于任何其他设置，则使用该控件的日文汉字转换模式设置。例如，如果IMEMode属性设置为 Off，日文汉字转换模式已关闭，而如果IMEMode属性设置为 On，日文汉字转换模式开启。日文汉字转换模式会自动更改每次控件之间移动焦点。

请注意 如果设置为禁用，日文汉字转换模式设置不能更改。如果使用其他任何设置，可以更改日文汉字转换模式，但是当焦点改变时，这些设置都将丢失。如果您想要保存设置，控件失去焦点之前，设置IMEHold/HoldKanjiConversionMode属性。

1.1.5. C# winform 请求http

1. 通过WebRequest类创建一个HttpWebRequest的对象，该对象可以包含Http请求信息。
2. 设置HttpWebRequest对象，其实就是设置Http请求报文的信息内容。
3. 从HttpWebRequest对象中获取HttpWebResponse对象，该对象包含Http响应信息。
4. 从响应信息中获取响应头信息和响应主体信息。

POST与GET的差异

1. GET是从服务器上获取数据，POST是向服务器传送数据。
2. GET是把参数数据队列加到提交表单的ACTION属性所指的URL中，值和表单内各个字段一一对应，在URL中可以看到。POST是通过HTTPPOST机制，将表单内各个字段与其内容放置在HTML HEADER内一起传送到ACTION属性所指的URL地址。用户看不到这个过程。
3. 对于GET方式，服务器端用Request.QueryString获取变量的值，对于POST方式，服务器端用Request.Form获取提交的数据。
4. GET传送的数据量较小，不能大于2KB（这主要是因为受URL长度限制）。POST传送的数据量较大，一般被默认为不受限制。但理论上，限制取决于服务器的处理能力。
5. GET安全性较低，POST安全性较高。因为GET在传输过程，数据被放在请求的URL中，而如今现有的很多服务器、代理服务器或者用户代理都会将请求URL记录到日志文件中，然后放在某个地方，这样就可能会有一些隐私的信息被第三方看到。另外，用户也可以在浏览器上直接看到提交的数据，一些系统内部消息将会一同显示在用户面前。POST的所有操作对用户来说都是不可见的。

get demo

```

/// <summary>
/// 获取访问网站时的IP
/// </summary>
private void GetrequestIP()
{
    try
    {
        //获取本机外网ip的url
        string getIpUrl = "http://www.ieasn.com/getip.php"; //网上获取ip地
        址的网站

        WebRequest wr = WebRequest.Create(getIpUrl);
        Stream s = wr.GetResponse().GetResponseStream();
    }
}

```

```
        StreamReader sr = new StreamReader(s, Encoding.UTF8);
        string all = sr.ReadToEnd(); //读取网站的数据
        GetRequestIP = all;

        sr.Close();
        s.Close();
    }
    catch (Exception e)
    {
        GetRequestIP = "获取客户端真实IP失败";
    }
}
```

1.1.6. 浏览器Request Header和Response Header的内容

1)请求(客户端->服务端[request])

GET(请求的方式) /hello.html(请求的目标资源) HTTP/1.1(请求采用的协议和版本号)

Accept: */*(客户端能接收的资源类型)

Accept-Language: en-us(客户端接收的语言类型)

Connection: Keep-Alive(维护客户端和服务端的连接关系)

Host: localhost:8080(连接的目标主机和端口号)

Referer: http://localhost/links.asp(告诉服务器我来自于哪里)

User-Agent: Mozilla/4.0(客户端版本号的名字)

Accept-Encoding: gzip, deflate(客户端能接收的压缩数据的类型)

If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT(缓存时间)

Cookie(客户端暂存服务端的信息)

Date: Tue, 11 Jul 2000 18:23:51 GMT(客户端请求服务端的时间)

2)响应(服务端->客户端[response])

HTTP/1.1(响应采用的协议和版本号) 200(状态码) OK(描述信息)

Location: http://www.baidu.com(服务端需要客户端访问的页面路径)

Server:apache tomcat(服务端的Web服务端名)

Content-Encoding: gzip(服务端能够发送压缩编码类型)

Content-Length: 80(服务端发送的压缩数据的长度)

Content-Language: zh-cn(服务端发送的语言类型)

Content-Type: text/html; charset=GB2312(服务端发送的类型及采用的编码方式)

Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT(服务端对该资源最后修改的时间)

Refresh: 1;url=http://www.ieasn.com/getip.php(服务端要求客户端1秒钟后, 刷新, 然后访问指定的页面路径)

Content-Disposition: attachment; filename=aaa.zip(服务端要求客户端以下载文件的方式打开该文件)

Transfer-Encoding: chunked(分块传递数据到客户端)

Set-Cookie:SS=Q0=5Lb_nQ; path=/search(服务端发送到客户端的暂存数据)

Expires: -1//3种(服务端禁止客户端缓存页面数据)

Cache-Control: no-cache(服务端禁止客户端缓存页面数据)

Pragma: no-cache(服务端禁止客户端缓存页面数据)

Connection: close(1.0)/(1.1)Keep-Alive(维护客户端和服务端的连接关系)

请求头部demo

```
private void GetResponseHeader(string RequestURL)
{
    string getstate = "";
    string responseHeaders = "";
    string responseHeaderCookies = "";
    try
    {
        HttpWebRequest request = (HttpWebRequest)
WebRequest.Create(RequestURL);
        HttpWebResponse response = (HttpWebResponse)
request.GetResponse();

        //协议版本号, 状态码, 状态描述
        getstate = (String.Format("{0,-20}HTTP/{1} {2:d} {3}", "(Status-
Line)", response.ProtocolVersion,
            response.StatusCode, response.StatusDescription));

        //Http response頭
```

```

        for (int i = 0; i < response.Headers.Keys.Count; i++)
        {
            responseHeaders +=
                (String.Format("{0,-20}{1}", response.Headers.Keys[i],
response.Headers.Get(i)) + "\r\n");
        }

        foreach (Cookie c in response.Cookies)
        {
            responseHeaderCookies = (c.ToString());
        }
    }
    catch (Exception ex)
    {
        responseHeaders = ex.Message;
    }
    finally
    {
        GetResponseHeaders = getstate + "\r\n" + responseHeaders + "\r\n"
+ responseHeaderCookies;
    }
}

```

1.1.7. 获取本地网络配置

只能获取本地IP，貌似无法获取远程访问IP

某些电脑中可能存在多个网卡，虚拟网卡，获取所需网络配置时，应适当注意

demo

```

/// <summary>
/// 从网卡获取ip设置信息
/// </summary>
private void GetIpInfo()
{
    string IPv4Mask = "";
    string IPAddress = "";
    string GatewayAddresses = "";
    string DnsAddresses = "";
    string GetAllDnsInfo = "";
    string IpInfos = "";
    try
    {
        NetworkInterface[] nics =
NetworkInterface.GetAllNetworkInterfaces();

        foreach (NetworkInterface adapter in nics)
        {
            bool Pd1 = (adapter.NetworkInterfaceType ==

```



```

NetworkInterfaceType.Ethernet); //判断是否是以太网连接
        if (Pd1)
        {
            IPInterfaceProperties ip = adapter.GetIPProperties(); //IP
配置信息
            if (ip.UnicastAddresses.Count > 0)
            {
                for (int i = 0; i < ip.UnicastAddresses.Count; i++)
                {
                    IPAddress =
ip.UnicastAddresses[i].Address.ToString(); //IP地址
                    IPv4Mask =
ip.UnicastAddresses[i].IPv4Mask.ToString(); //子网掩码
                    IpInfos += "IP地址:" + IPAddress + " | " + "子网掩
码:" + IPv4Mask + "\r\n";
                }
            }

            if (ip.GatewayAddresses.Count > 0)
            {
                for (int i = 0; i < ip.DnsAddresses.Count; i++)
                {
                    DnsAddresses +=ip.DnsAddresses[i].ToString()+" ";
//DNS
                }
                for (int i = 0; i < ip.GatewayAddresses.Count; i++)
                {
                    GatewayAddresses +=
ip.GatewayAddresses[i].Address.ToString(); //默认网关
                }

                GetAllDnsInfo = "默认网关:" + GatewayAddresses +
"\r\n" +
                    "DNS:" + DnsAddresses + "\r\n";
            }
        }
    }

    AllNetworkInfo = IpInfos + GetAllDnsInfo;
}
catch (Exception e)
{
    MessageBox.Show(e.Message);
}
}

```

1.1.8. 剪切板

» [clipboard类](#)

1.1.9. 控件拖拽

1. 通过DragEnter事件获得被拖入窗口的“信息”(可以是若干文件，一些文字等等)，在DragDrop事件中对“信息”进行解析。
2. 接受拖放控件的AllowDrop属性必须设置成true;
3. 必须在DragEnter事件中设置好要接受拖放的效果，默认为无效果。(所以单独写DragDrop事件是不会具有拖拽功能的)

demo

```
private void textBox1_DragEnter(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(DataFormats.FileDrop))
    {
        e.Effect = DragDropEffects.Link;
        this.textBox1.Cursor = System.Windows.Forms.Cursors.Arrow; //指定鼠标形状
(更好看)
    }
    else
    {
        e.Effect = DragDropEffects.None;
    }
}
private void textBox1_DragDrop(object sender, DragEventArgs e)
{
    //GetValue(0) 为第1个文件全路径
    //DataFormats 数据的格式，下有多个静态属性都为string型，除FileDrop格式外还有
    Bitmap,Text,WaveAudio等格式
    string path =
    ((System.Array)e.Data.GetData(DataFormats.FileDrop)).GetValue(0).ToString();
    textBox1.Text = path;
    this.textBox1.Cursor = System.Windows.Forms.Cursors.IBeam; //还原鼠标形状
}
```

1.1.10. 实现进程间通讯

用到一个windows api 32函数

```
[DllImport("User32.dll", EntryPoint = "SendMessage")]
private static extern int SendMessage(IntPtr wnd,int msg,IntPtr wP,IntPtr lP);
```

要有此函数，需要添加命名空间 `using System.Runtime.InteropServices;`

此方法各个参数表示的意义

wnd: 接收消息的窗口的句柄。如果此参数为HWND_BROADCAST, 则消息将被发送到系统中所有顶层窗口, 包括无效或不可见的非自身拥有的窗口、被覆盖的窗口和弹出式窗口, 但消息不被发送到子窗口。

msg: 指定被发送的消息类型。

wP: 消息内容。

lp: 指定附加的消息指定信息。

用api参考手册查看SendMessage用法时, 参考手册则提示

SendMessage与PostMessage之间的区别: SendMessage和PostMessage, 这两个函数虽然功能非常相似, 都是负责向指定的窗口发送消息 但是SendMessage() 函数发出消息后一直等到接收方的消息响应函数处理完之后才能返回, 并能够得到返回值 在此期间发送方程序将被阻塞, SendMessage() 后面的语句不能被继续执行, 即是说此方法是同步的。

而PostMessage() 函数在发出消息后马上返回, 其后语句能够被立即执行, 但是无法获取接收方的消息处理返回值, 即是说此方法是异步的。

1.1.11. 对于按键的操作

采用keycode

```
if (e.Event.WindowsKeyCode == 17 && e.Event.WindowsKeyCode == 67)
{
    MessageBox.Show("你按下了ctrl + C");
}
```

1.1.12. 习惯用trycatch捕获异常

DEMO :

```
try
{
    FUNCTION
}
catch (Exception e)
{
    messagebox.show(e.message);
}
finally
{
    FUNCTION
}
```

1.1.13. 转义特殊字符

```
public static string TransferStr(this string str)
{
    return str.Replace("\\", "\\").Replace("'", "\\'"); //旧→新
}
```

可多次替换

1.1.14. C# 可空类型（Nullable）

C# 可空类型（Nullable）C# 单问号 ? 与 双问号 ??

?: 单问号用于对 int,double,bool 等无法直接赋值为 null 的数据类型进行 null 的赋值，意思是这个数据类型是 Nullable 类型的。

`int? i = 3` 等同于 `Nullable<int> i = new Nullable<int>(3);`

`int i;` //默认值0 `int? ii;` //默认值null

?: 双问号 可用于判断一个变量在为 null 时返回一个指定的值。

C# 可空类型（Nullable）C# 提供了一个特殊的数据类型，nullable 类型（可空类型），可空类型可以表示其基础值类型正常范围内的值，再加上一个 null 值。

例如，`Nullable< Int32 >`，读作“可空的 Int32”，可以被赋值为 -2,147,483,648 到 2,147,483,647 之间的任意值，也可以被赋值为 null 值。类似的，`Nullable< bool >` 变量可以被赋值为 true 或 false 或 null。

在处理数据库和其他包含可能未赋值的元素的数据类型时，将 null 赋值给数值类型或布尔型的功能特别有用。例如，数据库中的布尔型字段可以存储值 true 或 false，或者，该字段也可以未定义。声明一个 nullable 类型（可空类型）的语法如下：

```
< data_type> ? <variable_name> = null;
```

Null 合并运算符（??）Null 合并运算符用于定义可空类型和引用类型的默认值。Null 合并运算符为类型转换定义了一个预设值，以防可空类型的值为 Null。Null 合并运算符把操作数类型隐式转换为另一个可空（或不可空）的值类型的操作数的类型。如果第一个操作数的值为 null，则运算符返回第二个操作数的值，否则返回第一个操作数的值。下面的实例演示了这点：

```
using System;namespace CalculatorApplication{
class NullablesAtShow
{
static void Main(string[] args)
{
double? num1 = null;
double? num2 = 3.14157;
double num3;
num3 = num1 ?? 5.34; // num1 如果为空值则返回 5.34
Console.WriteLine("num3 的值: {0}", num3);
num3 = num2 ?? 5.34;
Console.WriteLine("num3 的值: {0}", num3);
Console.ReadLine();
}
```

```
}  
}}
```

1.1.15. 全局异常捕获

1)简单粗暴：在Program.cs使用Try...Catch...

现在我们改造Program.cs里的main()函数，改成以下内容：

```
/// <summary>  
/// 应用程序的主入口点。  
/// </summary>  
[STAThread]  
static void Main()  
{  
    try  
    {  
        Application.EnableVisualStyles();  
        Application.SetCompatibleTextRenderingDefault(false);  
        Application.Run(new FrmMain());  
    }  
    catch (Exception ex)  
    {  
        MessageBox.Show(string.Format("捕获到未处理异常: {0}\r\n异常信息: {1}\r\n异常堆栈: {2}", ex.GetType(), ex.Message, ex.StackTrace));  
    }  
}
```

2)事件监听：Application.ThreadException

Application类，这个负责控制整个Windows 程序的运行。

在ThreadException事件上，微软对它的注释是：“在发生未捕获线程异常时发生。”

用try..catch...包住所有代码，当程序异常时将代码将会进入catch块里，处理完成后程序就退出了。

然而在这边我们用Application.ThreadException事件监听并处理后，程序并不会因为异常而退出。可以说是可挽回的异常。

```
//处理未捕获的异常  
Application.SetUnhandledExceptionMode(UnhandledExceptionMode.CatchException);  
//处理UI线程异常  
Application.ThreadException += Application_ThreadException;  
//处理非UI线程异常  
AppDomain.CurrentDomain.UnhandledException +=CurrentDomain_UnhandledException;
```

```
private static void Application_ThreadException(object sender,
ThreadExceptionEventArgs e)
{
    var ex = e.Exception;
    if (ex != null)
    {
        MessageBox.Show(ex.Message, "TITLE", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    }
}

private static void CurrentDomain_UnhandledException(object sender,
UnhandledExceptionEventArgs e)
{
    var ex = e.ExceptionObject as Exception;
    if (ex != null)
    {
        MessageBox.Show(ex.Message, "TITLE", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    }
}
```

1.1.16. 对注册表的操作

参考 [registrykey类](#)

在对应用软件的注册表进行操作时，注意**32**和**64**的注册表路径

为了支持 32 位和 64 位 COM 注册和程序状态的共存，WOW64 为 32 位程序提供了一种备用注册表视图。32 位程序会看到与真正的 64 位 HKEY_LOCAL_MACHINE\Software 树完全分开的 32 位

HKEY_LOCAL_MACHINE\Software 树（HKEY_LOCAL_MACHINE\Software\WOW6432Node）。这样可以隔 HKEY_CLASSES_ROOT，因为此树的每个计算机部分驻留在以下注册表项内：HKEY_LOCAL_MACHINE\Software 为了通过 COM 和其他机制实现 64 位/32 位程序互操作性，WOW64 使用了一个“注册表反射器”来在 64 位注册表视图和 32 位注册表视图之间镜像某些注册表项和项值。该反射器是“智能”的，因为它只反射 COM 激活数据。

反射的注册表项 WOW64 注册表反射器可能会在反射过程中修改注册表项的内容和项值，目的是为了调整路径名等。因此，32 位的内容与 64 位的内容可能会有所不同。下面的注册表项会影射：

```
HKEY_LOCAL_MACHINE\Software\Classes
HKEY_LOCAL_MACHINE\Software\COM3
HKEY_LOCAL_MACHINE\Software\Ole
HKEY_LOCAL_MACHINE\Software\EventSystem
HKEY_LOCAL_MACHINE\Software\RPC
```

注：WoW（Windows on Windows）模式 WOW64 是 Windows-32-on-Windows-64

1.1.17. Win控制台转dll

调用方法

- (1) 更加节省内存并减少页面交换;
- (2) DLL文件与EXE文件独立, 只要输出接口不变(即名称、参数、返回值类型和调用约定不变), 更换DLL文件不会对EXE文件造成任何影响, 因而极大地提高了可维护性和可扩展性;
- (3) 不同编程语言编写的程序只要按照函数调用约定就可以调用同一个DLL函数;
- (4) 适用于大规模的软件开发, 使开发过程独立、耦合度小, 便于不同开发者和开发组织之间进行开发和测试。
- (5) 节约磁盘空间: 当应用程序使用动态链接时, 多个应用程序可以共享磁盘上单个DLL副本。相比之下, 当应用程序使用静态链接库时, 每个应用程序要将库代码作为独立的副本链接到可执行镜像中

1.1.18. 自定义URL Protocol 协议 (http访问客户端)

参考地址:

<https://blog.csdn.net/chinahaerbin/article/details/8783024>

<http://www.cnblogs.com/zjneter/archive/2008/01/08/1030066.html>

<https://www.cnblogs.com/wang726zq/archive/2012/12/11/UrlProtocol.html>

git参考: <https://github.com/Hawkeyes0/UrlProtocolRegister.git>


运行regedit.exe

HKEY_CLASSES_ROOT是应用程序运行时必需的信息,HKEY_LOCAL_MACHINE\SOFTWARE\Classes是一样的, 但是在HKEY_CLASSES_ROOT窗编辑相对来说显得更容易和有条理。

HKEY_LOCAL_MACHINE是一个显示控制系统和软件的处理键。HKLM键保存着计算机的系统信息。它包括网络和硬件上所有的软件设置。(比如文件的位置, 注册和未注册的状态, 版本号等等) 这些设置和用户无关, 因为这些设置是针对使用这个系统的所有用户的。

HKEY_CLASSES_ROOT\HANRUI\

hanrui://xxxx 注释 协议头 协议参数 Tencent实例

12333.gif

1.1.19. Graphics 类

参考地址

demo:获取分辨率

```
float dpiX, dpiY,x,y;
Graphics graphics = this.CreateGraphics();
dpiX = graphics.DpiX;
dpiY = graphics.DpiY;
x = dpiX / 96;
```

```
y = dpiY / 96;
```

1.1.20. c#打开指定文件、程序

ex1

```
Process pr = new Process();//声明一个进程类对象
pr.StartInfo.FileName = "E://Program Files//Tencent//QQ//QQ.exe";//指定运行的程序，
我的QQ的物理路径。
pr.Start();//运行QQ
```

EX2

```
System.Diagnostics.Process.Start("explorer.exe", dir);
```

EX3

```
//创建启动对象
System.Diagnostics.ProcessStartInfo startInfo = new
System.Diagnostics.ProcessStartInfo();
//设置运行文件
startInfo.FileName = System.Windows.Forms.Application.ExecutablePath;
//设置启动参数
startInfo.Arguments = String.Join(" ", Args);
//设置启动动作,确保以管理员身份运行
startInfo.Verb = "runas";
```

1.1.21. 以管理员身份运行

1、直接在program.cs中添加

```
static class Program
{
    /// <summary>
    /// 应用程序的主入口点。
    /// </summary>
    [STAThread]
    static void Main(String[] Args)
    {
        if (Args.Length != 0 && Args[0].ToString() == "123")
        {
            //获得当前登录的Windows用户标示
            System.Security.Principal.WindowsIdentity identity =
```



```
        System.Security.Principal.WindowsIdentity.GetCurrent();
        System.Security.Principal.WindowsPrincipal principal =
            new System.Security.Principal.WindowsPrincipal(identity);
        //判断当前登录用户是否为管理员
        if
(principal.IsInRole(System.Security.Principal.WindowsBuiltInRole.Administrator))
        {
            //如果是管理员，则直接运行
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            MainRepairTool.RepairRegister();
            //MessageBox.Show("注册表修复成功!", "title",
            MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
            Application.Exit();
        }
        else
        {
            try
            {
                //创建启动对象
                System.Diagnostics.ProcessStartInfo startInfo = new
System.Diagnostics.ProcessStartInfo();
                //设置运行文件
                startInfo.FileName = "aaa.exe";
                //设置启动参数
                startInfo.Arguments = "213";
                //设置启动动作,确保以管理员身份运行
                startInfo.Verb = "runas";
                //如果不是管理员，则启动UAC
                System.Diagnostics.Process.Start(startInfo);
            }
            catch (Exception e)
            {
            }
            finally
            {
                Application.Exit();
            }
        }
    }
    else
    {
        Application.Exit();
    }
}
```

2、调用app.mainfest

- 一： 在Visual Studio 中--解决方案资源管理器--右键项目名称--属性，找到“安全性”选项，
- 二： 勾选“启用ClickOnce安全设置”，

三：这时，在项目下面会多出一个“app.manifest”的文件，选中它，并找到代码段

```
<requestedExecutionLevel level="asInvoker" uiAccess="false" />, 将其改为:  
<requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
```

四：改正后，不要急于重新编译生成，再次打开“属性--安全性”界面，将“启用ClickOnce安全设置”前面的勾去掉后再编译运行。不然程序会报错无法运行。

1.1.22 控件重绘

1.2. 基于NanUI

1.2.1. NanUI中如何调用devtools

可以直接采用获取按键方式调用调试窗口，为避免普通用户使用，可以判断是否在调试状态

```
/// 获取键盘按下的key值  
private void KeyboardHandler_OnKeyEvent(object sender, CfxOnKeyEventEventArgs e)  
{  
    {  
        if (e.Event.WindowsKeyCode == 123) //F12  
        {  
            Chromium.ShowDevTools();  
        }  
        if (e.Event.WindowsKeyCode == 116) //F5  
        {  
        }  
    }  
}
```

1. 值操作

2. `SetReturnValue`

`CfrV8Value.CreateString`

1.2.2. 前后端交互

win调用前端

```
Chromium.ExecuteJavascript("Comprefresh()");
```

前端调用Winform

```
GlobalObject.AddFunction("normalFrm").Execute += NormalFrm;  
private void NormalFrm(object sender, CfrV8HandlerExecuteEventArgs e)  
{  
    WindowState = FormWindowState.Normal;  
}
```

1.3. 其他

1.3.1. 正则表达式

参考<http://www.runoob.com/regexp/regexp-tutorial.html>

举例：如果我们想要找到字符串The dog chased the cat中单词 the，我们可以使用下面的正则表达式: `/the/gi`
我们可以把这个正则表达式分成几段：

`/`是这个正则表达式的头部

`the` 是我们想要匹配的模式

`/` 是这个正则表达式的尾部

`g`代表着 `global`(全局)，意味着返回所有的匹配而不仅仅是第一个。

`i` 代表着忽略大小写，意思是当我们寻找匹配的字符串的时候忽略掉字母的大小写。

1.3.2. Keycode对照表

Image.png