ConfigurationManager2015年8月11日21:19:57

1. 查找集合的第一个匹配项

用FirstOrDefault方法。可以用于数组，也可以用于list集合等。然后用lambda表达式来匹配

\_currentTarget = \_targets.FirstOrDefault(t => t.CultureName == cultureName);

其中\_targets是一个list集合，具有CultureName的属性。

C#后台执行 bat脚本代码

System.Diagnostics.Process proc = new System.Diagnostics.Process();

proc.StartInfo.FileName = batFilename;

proc.StartInfo.RedirectStandardOutput = true;

proc.StartInfo.UseShellExecute = false;

proc.StartInfo.CreateNoWindow = true;

proc.OutputDataReceived += new System.Diagnostics.DataReceivedEventHandler(proc\_OutputDataReceived);

proc.Start();

proc.BeginOutputReadLine();

proc.WaitForExit();

2015-8-12 08:51:13

1. 用.snk来加密dll文件，用是snk给dll签名后，别人就不能访问你的dll了。

使用vs自带的命令行工具，输入： sn -k c:\test.snk ，后面是路径

把生成的这个文件添加到自己的工程里面就可以了。

添加snk，在项目的属性中，选择签名项，下面有个选择强名称密钥文件，选择一个.snk文件即可。

或者在项目的Properities下的Assemblyinfo.cs文件里添加：

[assembly: AssemblyKeyFile("c:\\test.snk")]

1. 其实创建snk的强名密钥，不需要在命令行里面输入sn指令。

在visual studio 项目属性的签名里面选择密钥的时候，下拉列表里面有个新建，这里可以指定密码，如果没有密码，秘钥就是snk文件，如果有密码就是pfx文件。

2015年8月12日15:14:22

1. 使用共享程序集

默认的程序集不共享，比如一个工程写的库，被另一个工程引用，需要添加引用，并且将它的属性里面的“复制本地”设置为true；否则程序运行出错，提示找不到文件。

如果将那个库作为全局程序集放入全局程序集缓存（GAC）中，那么久不用复制那个库（dll）到自己的这个工程里面就嫩运行了。也就是把库dll放到了一个特殊的位置，任何程序都能访问。

使用共享程序集，需要给程序集加强名，程序集的源代码里面有个AssemblyInfo.cs

里面定义了程序集的GUID,保证了程序集的唯一性。加强名是为了保护该程序集。

可以使用sn命令来生产snk加密文件，也可以在工程的属性里面找到签名，在里面新建和设置签名文件。

然后就是将程序集加到GAC里面了，使用VS自带的命令提示符。用gacutil.exe的命令。

安装 gacutil /i myclass.dll

卸载 gacutil /u myclass.dll

1. 在类前面添加guid，用菜单栏工具自动生成guid会报错，提示guid不是特性类。出错的原因是没有添加正确的命名空间

正确应该添加using System.Runtime.InteropServices;

2015年8月14日09:21:12

1. 获取程序集的名字

Assembly \_sfgAssembly;

public InstallForm(Assembly sfgAssembly)

{

\_sfgAssembly = sfgAssembly;

InitializeComponent();

this.Text = string.Format("Install Single-file generator: {0}",

Path.GetFileNameWithoutExtension(sfgAssembly.Location));

}

2015年8月17日20:35:25

1. C#后台线程，设置线程的IsBackgrouond属性为true，则线程为后台线程，意思是如果主线程关闭了，那么后台线程自动关闭，如果不是后台线程，那么主线程关闭后台线程依然运行，需要手动关闭。

2015年8月29日18:58:09

1. 观察者模式

观察者模式实现了一个事件的发出者出触发了某个事件，那么观察者自动执行相应事件，不用主动去调用观察者的方法。观察的对象间是低耦合的，方便了维护和扩展。

通过委托来订阅事件，为了方便扩展，先定义了两个抽象的基类:抽象的观察目标，抽象的观察者。然后定义了具体的观察者和具体的目标。

在抽象的观察目标里定义委托和事件。然后写一个调用事件的方法，供子类调用。

然后在抽象的观察者基类里面注册前面定义的事件为本抽象类的一个抽象方法。然后继承自该观察者基类的子类，一旦重写了这个抽象方法，该方法就绑定到事件上面去了。

//观察目标基类

abstract class ModleBase

{

public delegate void SubEnentHanler();

public event SubEnentHanler subevent;

public void Notification()

{

if (this.subevent != null)

{

this.subevent();

}

}

}

//观察者基类

abstract class Observer

{

public Observer(ModleBase modlebase)

{

modlebase.subevent += new ModleBase.SubEnentHanler(Response);

}

public abstract void Response();

}

//具体观察目标

class Cat:ModleBase

{

public void Cry()

{

Console.WriteLine("猫叫");

this.Notification();

}

}

//具体观察者

class Mouse : Observer

{

//在构造函数里指定观察的对象

public Mouse(ModleBase mousebase)

: base(mousebase)

{

}

//重写父类的抽象方法，自动绑定事件

public override void Response()

{

Console.WriteLine("老鼠跑");

}

}

//主方法

static void Main(string[] args)

{

Cat cat = new Cat();

Mouse mouse = new Mouse(cat);

cat.Cry();

Console.ReadLine();

}

2015年8月31日09:52:36

1. C#的数组是引用传递，和java一样

将数组作为参数传递给一个方法，在方法内部修改了数组的值，该数组的值彻底被修改了。

//定义一个修改数组的方法

public static void SetArray(int [] arr)

{

arr[1] = 88;

}

//主方法

int[] testarr = { 12, 22, 33, 44 };

SetArray(testarr);

Console.WriteLine(testarr[1]);

结果输出88，说明数组的值被改变了，是引用传递。

定义了数组后必须在堆上分配内存。

数组的定义时必须制定大小，使用new关键字，没有小括号。

Int [] myarray = new int[4];可以在后面加大括号为数组的每个变量赋值。

数组的声明也可以不用new ，编译器自动识别。

Int[] myarry={1,2,3};

1. 由于数组是引用传递，因此将一个数组赋值给另一个数组，这两个数组将指向统一个内存空间。

int[] myArray = { 12, 13, 23, 45 };

int[] yourArray = myArray;

yourArray[1] = 99;

Console.WriteLine(myArray[1]);

输出99，

为了得到两个数组，内容相同，地址不同。需要用到数组的复制clone。

int[] hisArray = (int[])myArray.Clone();

hisArray[2] = 0;

Console.WriteLine(myArray[0]);

1. 数组排序

要实现数组排序，那么数组内的元素必须实现Icomparable借口，普通类型string，int等都实现了Icomparable。

自定义类的数组，就需要自己实现这个借口了。

使用Array的sort方法来排序

int[] testshuzu = { 43, 12, 66, 1, 53, 16 };

Array.Sort(testshuzu);

foreach (var item in testshuzu)

{

Console.WriteLine(item);

}

1. 方法发可变参数

指参数的个数是可变的，本质上是将参数转换为数组，使用关键字params

public static void fun(params int[] arg)

{

for (int i = 0; i < arg.Length; i++)

{

Console.WriteLine(arg[i]);

}

}

调用时fun(),fun(1,2,3)都可以。

在java里面是用…来代替

public static void fun(int ... arg)

{

for(int i=0;i<arg.length;i++)

{

System.out.println(arg[i]+",");

}

}

2015年8月31日15:49:25

1. 设计模式能降低系统中类与类之间的耦合度。比如工厂模式，使依赖类只需知道被依赖类所实现的接口或继承的抽象类，使依赖类与被依赖类之间的耦合度降低。

其实就是抽象出一个父类，有多个子类。以后想新增功能就只需要多添加一个子类就可以了，而不用修改父类和别的子类。使耦合降低。调用的时候定义父类变量，指向子类对象，指向不同的子类，结果就不同。实现多态。

一个例子是计算器，定义一个父类

public class Operation

{

public double NumberA{get;set;}

public double NumberB{get;set;}

public virtual double GetResult() {

double result = 0;

return result;

}

}

子类加减乘除都重写GetResult方法。调用时只需要定义这个父类，指向子类就可以了。以后想添加一个别的算法，比如开根号，只需要多写一个子类就可以了。

2015年9月1日09:33:19

1. C#和java的string的比较
2. 定义：

都可以直接定义用引号表示字符串。String s= “hello”,区别是C#的string是小写，java是大写。

Java可以用new来定义字符串。String str = new string(”hello”),而C#不能这样定义，C#的string的构造函数里不能传一个字符串，但可以传一个char型的数组。

1. C#每次给string变量赋值都会开辟新的内存空间，而java不会，java中，如果新定义的string变量内容和之前定义的string变量内容相同，则不开辟新的内存，指向和它内容相同的变量的空间。如果没有则用new开辟新的内存。
2. C#中==和equal 都是比较内容，而java中==是比较地址，equal才是比较内容。
3. Java和C#的string在赋值次数较多都会降低效率，因为都开辟了多的空间，解决办法是用新的数据类型，C#是stringbuilder，java是stringbuffer。

2015年9月2日09:47:57

1. 关于static

C#和java区别还是挺大的。在C#中一个字段定义为static，那么它只能通多类名来访问，对象是不能访问的。而在java中，类名和对象都可以访问到static类型的变量。

在java中，定义了一个含有static变量的类，实例化该类的多个对象，改变其中一个对象的static属性，则其他的对象也会改变。

**public** **class** Test902 {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Person p1= **new** Person();

Person p2 = **new** Person();

p1.getInfo();

p2.getInfo();

p1.*country*="B";

p1.getInfo();

p2.getInfo();

}

}

class Person

{

public static String country="A";

public void getInfo()

{

System.out.println(country);

}

}

输出：

A

A

B

B

这样的语法在C#里面是不允许的

Java中的static属性(即变量，字段)是所有对象共享的属性，保存在全局数据区中，所有的对象指向全局数据区中的一个内容，所以当一个对象修改之后，所有对象的内容将全部变化。

其实争取的方法应该是用类名来引用该static变量。这样和C#是一样的。类名引用的static变量修改后，每个对象间接输出该变量也会修改。

1. 统计创建了多少个对象，用static变量很好结解决了这个问题

Static变量是先于对象之前被创建，所有对象共同拥有

class Demo{ // 定义Person类

private static int count = 0 ; // 所有对象共享此属性

public Demo(){

count++ ; // 只要有对象产生就应该自增

System.out.println("产生了" + count + "个对象！") ;

}

};

public class StaticDemo06{

public static void main(String args[]){

new Demo() ; // 增加新对象

new Demo() ; // 增加新对象

new Demo() ; // 增加新对象

}

};

2015年9月7日20:57:10

1. 可空的值类型

形如：int? a;的代码

可空类型修饰符(？)：引用类型可以使用空引用表示一个不存在的值，而值类型通常不能表示为空，例如:string str=null;是正确的。int i=null；编译器将报错。为了使值类型也可为空，可空类型出现了，可空类型使用可空类型修饰符？来表示，表现形式为T?。例：int?表示是可空的整形，DateTime?表示为可空的时间。T?其实是System.Nullable<T>（泛型结构）的缩写形式，也就意味着当你用到T？时编译器在编译时会把T？编译成System.Nullable<T>的形式，例如：int?,编译后便是System.Nullable<int>的形式。

1. 空合并运算符(？？)：用于定义可空类型和引用类型的默认值。如果此运算符的左操作数不为 null，则此运算符将返回左操作数；否则返回右操作数。 例：a??b,当a为null时则返回b,a不为空时返回a本身。空合并运算符为右结合运算符，即操作时从右向左进行组合的。如，“a??b??c”的形式按“a??(b??c)”计算。

2015-9-9 17:30:56

1. 判断操作系统位数

方法一：

bool type;

type = Environment.Is64BitOperatingSystem;

Console.WriteLine(type);

方法二：

这个方法也是最直接的方法，但是有条件限制，例用IntPtr结构的size属性来查看系统的位宽  
命名空间是System;  
前题是程序需要采用any/CPU的方式进行编辑；  
正常情况下int的位宽是4位，即是32位操作系统。

if (IntPtr.Size == 8)

{

//64 bit

}

else if (IntPtr.Size == 4)

{

//32 bit

}

else

{

//...NotSupport

}

2015年9月20日18:18:06

1.vs2013 sdk安装时报错，提示Windows Program Compatibility mode is on. Turn it off and then try Setup again

意思是程序开启了兼容模式，关闭它然后重试。可是并没有开启上面兼容模式啊。查找资料才发现原来是程序exe的名字的问题。我的桌面之前已经下载了2012的sdk，名字是vssdk\_full.exe，然后下载的2013的sdk也是这个名字，操作系统自动在后面添加(1),变成了vssdk\_full(1).exe,这样就不行了。程序会默认检查exe的名字，它的名字必须是vssdk\_full.exe才行。改过来后安装正常。

2015年9月27日11:04:57

1. 接口的显示实现和隐式实现

先看例子：

隐式：

   interface IAnimal  
　　　　{  
　　　　　　　void Dog();  
　　　　}  
　　　　class Animal:IAnimal  
　　　　{  
　　　　　　public void Dog()  
　　　　　　{  
　　　　　　　　Console.WriteLine("dog...");  
　　　　　　}  
　　　　 }

显示：

interface IAnimal  
　　{  
　　　　void Dog();  
　　}  
　　class Animal:IAnimal  
　　{  
　　　　void IAnimal.Dog()  
　　　　{  
　　　　　　Console.WriteLine("dog...");  
　　　　}  
　　}

在显示实现时，方法名前多了个接口名。

隐式实现可以用接口和类来调用，而显示实现只能用接口调用。

如在显示中只能

　 IAnimal animal = new Animal();

　　　　animal.Dog();

而隐式中不仅能这样还能

Animal animal = new Animal();  
　　　　animal.Dog();

既然显示接口实现这么不给力，为什么还要存在了，凡事存在即合理。在实际项目中，有时某个类往往会继承多个接口，而接口中往往会有一些相同名称、参数与类型的值。通过显式接口实现可以为避免一些不必要的歧义（我在项目中碰到的不多，可能是因为项目太小的原因吧）。

　　显示接口实现与隐式接口实现的适应场景

当类实现一个接口时，通常使用隐式接口实现，这样可以方便的访问接口方法和类自身具有的方法和属性。

当类实现多个接口时，并且接口中包含相同的方法签名，此时使用显式接口实现。即使没有相同的方法签名，仍推荐使用显式接口，因为可以标识出哪个方法属于哪个接口。

隐式接口实现，类和接口都可访问接口中方法。显式接口实现，只能通过接口访问。

2． 一个类继承了Ienumerable或者Ienumerable<T>则它能被foreach遍历

遍历的是该类的一个对象。Foreach可以写在别的地方，也可以写在该类里面，这时遍历的就是this了。

在使用foreach遍历的时候，才会调用GetEnumerator方法。这个方法返回一个Ienumerator类型对象集合。可以自己写一个类来实现这个接口。自己开发GetEnumerator()、MoveNext()、Current和Reset()，但比较复杂，更简单的是委托到System.Array下因为System.Array类型和其他许多类型（如List）已经实现了IEnumerable和IEnumerator接口

意思就是在GetEnumerator方法里返回一个集合的（如数组，list）的GetEnumerator方法。

你的类容都写到这个集合里面了。实际上就是遍历的这个集合，表面上是遍历的对象。

看例子：一个车库类，实现了Ienumerable接口

class Garage :IEnumerable

{

Car[] cararry = new Car[]

{

new Car{name="Rusty",speed=30},

new Car{name="Clunker",speed=6},

new Car{name="Zippy",speed=80},

new Car{name="Fred",speed=45},

};

public IEnumerator GetEnumerator()

{

return cararry.GetEnumerator();

}

}

class Car

{

public string name;

public int speed;

}

主方法:  
static void Main(string[] args)

{

Garage ga = new Garage();

foreach(Car c in ga)

{

Console.WriteLine(c.name);

Console.WriteLine(c.speed);

Console.WriteLine();

}

Console.ReadLine();

}

也可以写成泛型枚举IEnumerable<T>

这样类型就任意了

在上面的例子中，Car类型的数组，可以写在Garage类的子类里。在抽象方法里调用。Foreach也不用再main方法里调用。可以在Garage里面调用.遍历的对象就是this了。

foreach(Car c in this)

总结：其实就是把一个集合传递给了一个继承了Ienumerable的类。然后在遍历这个类的对象。可以在写一个类继承前面这个类。将GetEnmuable的方法写成抽象方法。在子类里实现。再加上泛型T，更方便了扩展。

相同的一个问题：为什么要遍历对象？ 比如winform里面的ListBox列表框，可以用foreach遍历里面的内容，可以用add添加内容。遍历的就是一个对象。**为的就是将一个集合(元素个数未知)传递给这个类，在这个类里面对这个集合进行处理。不用管这个集合是什么类型的。**

**将集合写到类里面，在类外可对其添加元素，在类里面遍历对象，让类看起来更像一个容器。集合就包含在类里面了。使外部程序更加简洁。**

**要将一个未知数量的集合传递给父类，父类里面要遍历这个集合，就可以用到Ienumerable，**

class ListBox :IEnumerable

{

public List<string> boxlist = new List<string>();

public IEnumerator GetEnumerator()

{

return boxlist.GetEnumerator();

}（）

}

static void Main(string[] args)

{

ListBox lb = new ListBox();

lb.boxlist.Add("星期一");

lb.boxlist.Add("星期二");

lb.boxlist.Add("星期三");

lb.boxlist.Add("星期三");

lb.boxlist.Add("星期四");

lb.boxlist.Add("星期五");

foreach (string item in lb)

{

Console.WriteLine(item);

}

Console.ReadLine();

}

也可以写成在类里面遍历：

class ListBox :IEnumerable

{

public List<string> boxlist = new List<string>();

public IEnumerator GetEnumerator()

{

return boxlist.GetEnumerator();

}

public void printinfo()

{

foreach (string item in this)

{

Console.WriteLine(item);

}

}

}

主方法里：lb.printinfo();

当在一个父类中要调用子类传来的集合时，可以在父类里面遍历父类对象this，因此在父类里继承Ienumerable接口，接口方法写成抽象的，在子类里实现，并传递子类的集合。

在看一个例子：

abstract class Person<T> :IEnumerable<T>

{

public abstract IEnumerator<T> GetEnumerator();

System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

public abstract void printinfo(T elemet);

public void Test()

{

foreach (T item in this)

{

printinfo(item);

}

}

}

class Student : Person<Info>

{

List<Info> list = new List<Info>

{

new Info{age=10,name="zhangsan"},

new Info{age=20,name="lisi"},

new Info{age=30,name="wangwu"},

new Info{age=40,name="zhaoliu"}

};

public override void printinfo(Info elemet)

{

Console.WriteLine("姓名：{0}",elemet.name);

Console.WriteLine("年龄：{0}",elemet.age);

Console.WriteLine();

}

public override IEnumerator<Info> GetEnumerator()

{

return list.GetEnumerator();

}

}

class Info

{

public int age;

public string name;

}

主方法里：

Student st = new Student();

st.Test();

2015年9月29日10:04:39

1. 缙云山系统一退出就报错提示“vshost32.exe”已停止工作，找解决方法是说在vs里面将项目属性里面调试下的启用visual studio承载进程关闭。试了下还可以。
2. Devexpres中CheckEdit的使用

这个控件是前面带文字的复选框。添加方法是：在ribbon界面的某个group下右键选择add edit，然后选择check edit这个选项。就能添加一个带文字的复选框。可是在属性里看到的name为barEdititem2,下面还有个edit的属性，名字为：repositoryItemCheckEdit2，在这个控件是事件里找不到复选框勾选的事件，只有个itemclick的事件，这个事件的效果是点击文字的事件，根本不是复选框的勾选事件。

正确的做法是，打开ribbon的设计器，在里面设置。在ribbon菜单的空白处右键选择Run designer，打开Ribbon Control designer窗体。选择左边最下面的Repository下面的Repository editor,然后在右边就列出了复选框的真正名字repositoryItemFontEdit1，然后在右边可以设置属性和事件了。判断勾选与否的方法如下：

private void repositoryItemCheckEdit2\_CheckedChanged(object sender, EventArgs e)

{

if (((DevExpress.XtraEditors.CheckEdit)sender).Checked == true)

{

ribbonPage2.Visible = true;

}

else

{

ribbonPage2.Visible = false;

}

}

1. 在devexpress里面添加一个tab或者group都在designer里面设置，若要添加其他控件，如按钮等，是不能在这里添加的，要在主界面用右键添加。要删除控件只能用delete键，右键是没有这个选项的。

2015年9月29日20:46:33

1. 事件的逆向使用

这是我自己说法，不知是否科学。普通事件都是一个事件订阅多个方法，触发一个事件将执行多个方法。

然而可不可以多个相似事件都调用一个方法，在方法里判断事件来源。因为有些时候会遇到很多相似的事件，它们执行的方法都是相似的，为了减少代码的冗余，和类之间的耦合度。因此需要将事件反过来使用。

一个例子是，一个winform里面有很多按钮，如果为每个按钮都写一个单击事件，将会使代码冗余，出现重复工作。而且为了使主界面代码简洁，需要将界面和实际执行功能的代码分开。

其实可以很简单的把每个button的click事件设置为同一个方法即可。但这样每个按钮就不好区分了，没有自己单独的单击事件了。

也可以写到另一个类里面，在另一个类里面处理事件的订阅。

public class ButtonClass :IEnumerable<Button>

{

public List<Button> btn\_list = new List<Button>();

public IEnumerator<Button> GetEnumerator()

{

return btn\_list.GetEnumerator();

}

System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

public void onCreate()

{

foreach (Button item in this)

{

item.Click += item\_Click;

}

}

void item\_Click(object sender, EventArgs e)

{

Button btn = sender as Button;

MessageBox.Show(btn.Name);

}

}

ButtonClass btdelegate;

public Form1()

{

InitializeComponent();

btdelegate = new ButtonClass();

btdelegate.btn\_list.Add(button1);

btdelegate.btn\_list.Add(button2);

btdelegate.btn\_list.Add(button3);

btdelegate.btn\_list.Add(button4);

btdelegate.btn\_list.Add(button5);

btdelegate.onCreate();

}

这段代码的好处是，主窗体界面非常简洁，没有click事件，仅有的是将按钮添加到Button\_Class里面，并在onCreate里面绑定每个按钮为同一个事件。完全做到了代码和界面分离。

还有一个好处是，还可以单独定义每个按钮的单击事件，这样每个按钮不仅有各自的单独事件，还有共同的单击事件。

第三个好处是，容易扩展，添加一个按钮后几乎不用修改别的的类，只需要在初始化的时候写一句添加按钮到list的语句就可以了，修改非常方便。

2015年9月30日23:07:43

1. 到此时，我的RibbonGenerator的custom tool终于弄成功了。能在vs2013下使用custom生成ribbon文件和日志文件了，但有个缺陷是工程路径不能有中文，否则生成失败。遇到了很多失败，好几个月了，总算弄好了。明天再记录心得体会把。也生成了vsix的插件文件了。

2015年10月1日22:07:24

1. 子类对象赋值给父类变量，然后调用子类重写父类的方法，还是调用的子类的方法，而不是父类的方法。即是间接调用重写过的方法，也是调用子类的方法。（重写的方法，在子类和父类中返回值，方法名，参数都一样）

class Program

{

static void Main(string[] args)

{

B b = new B();

A a = b;

a.fun1();

a.fun2();

Console.ReadLine();

}

}

class A

{

public virtual void fun1()

{

Console.WriteLine("A类的方法fun1");

}

public void fun2()

{

this.fun1();

}

}

class B : A

{

public override void fun1()

{

Console.WriteLine("B的方法fun1");

}

}

通过子类进行父类对象的实例化，实际上还是父类的变量，这个变量只能调用父类中已经存在的方法，只是复写过的会调用子类的方法，如果子类中有新的方法是父类中没有的，那么这个变量是不能调用的。比如在B类中有一个func3的方法，

A a = new B(); 那么a是不能调用func3的。

2015年10月14日21:39:43

1. 工厂模式降低了耦合度

实际上就是新建了一个工厂类，用它来创建子类，而不是在客户端创建子类，这样就降低了客户端和类的耦合，如果添加了类，只需要修改工厂，而不是客户端就可以了。

1. JAVA的接口中可以有常量，不能有变量，而C#中是不能定义一个变量或常量的。

在java中如果定义一个普通的变量是不会报错的，编译器会默认把它编译为常量，不能修改了等同于final static

子类不能重新接口中定义的常量。子类在复写的时候不用管这个变量，因为它是常量，所以不能重新赋值。但是可以拿来用。

这点和C#的区别很大。

2015年10月15日20:27:46

1. bool 取反

bool a = true;  
bool b = !a;

注意是 =！

If的判断里不等于是！=

1. DevExpress在Ribbon里面添加checkedit的用法。这里是在ribbon里面右键选择add editro –> checkedit，不是在工具箱里面选择checkedit，工具箱里面选择的checkedit和ribbon里面选择的有很大差别，前者添加后与ribbon无关，位置固定，不会因为ribbon的page的改变而改变位置和是否显示，他和普通winform里面添加一个控件是一样的 。它有checked属性，指明了勾选状态。而ribbon里面右键添加的checkedit，进去后是个BarEditItem,里面有个repositoryItemCheckItem，设置它的改变事件后，sender又变成了CheckEdit，需要将sender强制类型转换才能获取它是否勾选，很奇怪它是怎么就变换了类型的。

在找到这个ribbon上的checkedit的勾选设置的时候，废了很大的力气，根本找不到repositoryItemCheckItem有哪个属性可以设置是否勾选，因此无法设置它默认勾选。找了很久才发现这个repositoryItemCheckItem关联的BarEditItem下有个EditValue属性，这个属性是object类型的，可以设置为true或false，就表明是否勾选了。给这个属性设置值后，勾选框不在是未知状态了，要么勾选，要么不勾选。

找到一个怎么弄了，剩下的就很简单了。

为了简化程序，还是将所有的checkedit传递到一个集合里面了。这次不传repositoryItemCheckItem了，而是传BarEditItem，因为要设置EditValue属性。传递BarEditItem和RibbonPage。将传入的BarEditItem的Eidt字段转换为RepositoryItemCheckEdit，就可以访问caption属性了，就能和check的text对应了，就能找到那个BarEditItem对应哪个RibbonPage了。RepositoryItemCheckEdit rcheck = item.BarEdit.Edit as RepositoryItemCheckEdit;

class CheckEditor : IEnumerable<BarItem\_Page>

{

//存储复选框的集合

//List<RepositoryItemCheckEdit> checkEdits = new List<RepositoryItemCheckEdit>();

public List<BarItem\_Page> bar\_page\_list = new List<BarItem\_Page>();

public IEnumerator<BarItem\_Page> GetEnumerator()

{

return bar\_page\_list.GetEnumerator();

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

/// <summary>

/// 用于绑定复选框改变事件

/// </summary>

public void onCreate()

{

foreach (BarItem\_Page item in this)

{

item.BarEdit.Edit.EditValueChanged += Edit\_EditValueChanged;

item.BarEdit.EditValue = item.Ribbon\_page.Visible;

}

}

void Edit\_EditValueChanged(object sender, EventArgs e)

{

//MessageBox.Show(sender.ToString());

DevExpress.XtraEditors.CheckEdit dxck = (DevExpress.XtraEditors.CheckEdit)sender;

foreach (BarItem\_Page item in bar\_page\_list)

{

RepositoryItemCheckEdit rcheck = item.BarEdit.Edit as RepositoryItemCheckEdit;

if (rcheck.Caption == dxck.Text)

{

//item.Ribbon\_page.Visible = !item.Ribbon\_page.Visible;

if (dxck.Checked == true)

{

item.Ribbon\_page.Visible = true;

}

else

{

item.Ribbon\_page.Visible = false;

}

}

}

}

}

class BarItem\_Page

{

private BarEditItem barEdit;

private RibbonPage ribbon\_page;

public BarEditItem BarEdit

{

get { return barEdit; }

set { barEdit = value; }

}

public RibbonPage Ribbon\_page

{

get { return ribbon\_page; }

set { ribbon\_page = value; }

}

public BarItem\_Page(BarEditItem bar,RibbonPage page)

{

this.BarEdit = bar;

this.Ribbon\_page = page;

}

}

主窗体：

CheckEditor cked = new CheckEditor();

cked.bar\_page\_list.Add(new BarItem\_Page(D3DMod\_Check, DCT\_ribbonPage)); //三维对象

cked.bar\_page\_list.Add(new BarItem\_Page(barEditItem18, ribbonPage11)); //人员监测

cked.bar\_page\_list.Add(new BarItem\_Page(barEditItem19, ribbonPage13)); //跳跳鱼监测

cked.bar\_page\_list.Add(new BarItem\_Page(barEditItem12, Map\_ribbonPage)); //地图操作

cked.bar\_page\_list.Add(new BarItem\_Page(barEditItem13, ribbonPage10)); //编辑

cked.onCreate();

2015年10月16日10:08:19

1. 使用settings来保存配置，在winform程序中，在properties下有个Settings.Designer.cs文件，这个就是用来配置程序的，但是没有必要自己手动修改里面的内容。打开工程属性，在左边可以选择设置，右边有个表格，可以添加设置的名称，类型和值。比如添加一个值用于保存窗体位置，名字可以随便写，类型选择System.Drawing.Size，这样还没有完，是要在程序的代码里面调用这个值的。比如之前设置的窗体位置的值的名字为WindowLocation，那么调用的方法是Settings.Default.WindowLocation

在窗体加载和关闭的时候可以对它进行读取和设置，这样就能保存窗体的位置了。这个值不像用AppSetting那样保存到一个config的文件里面，而是直接写到程序集里面，只有一个exe，下次打开一样能记住窗体上次关闭时的位置。

可以加上null的判断

特别注意设置完值后还得保存配置

Settings.Default.Save();

把这句话加在窗体关闭时设置变量值的后面，就能保存窗体位置了。

if (Settings.Default.WindowLocation != null)

{

this.Location = Settings.Default.WindowLocation;

}

这种方法可以保存非常多类型的东西到程序里面，也不确定是不是把配置写入到exe里面了。因为一旦把exe移动到别的位置，配置就变成默认的了。

2015年10月17日22:47:44

1. 把事件封装为属性用 add 和remove

public event EventHandler AcceptsTabChanged

{

add

{

baseTextBox.AcceptsTabChanged += value;

}

remove

{

baseTextBox.AcceptsTabChanged -= value;

}

2015年11月16日22:28:49

1. C#中的public new void FunctionName的new

new关键字用于隐藏父类中同名的方法或字段属性，子类中有个方法和父类的方法同名，父类中的方法没有用abstract或者virtual修饰，不能用override重写，这样子类中的方法会提示警告，但不会报错，能正常运行。这时在访问修饰符后添加了new 就不会警告了，原理是隐藏了父类中的同名方法。字段也一样。

public new char PasswordChar

{

get { return passwordChar; }

set

{

passwordChar = value;

SetBasePasswordChar();

}

}

推断这种用途是修改父类的方法，使之实现新功能。

2016年3月2日16:26:49

1. Arrays.fill填充数组的方法

Arrays.fill（ a1, value ）;第一个参数的数组，第二个是值，把数组的每一个元素都赋值为value

## 2016年3月20日13:11:02

1. 最简单的文本写入方法

public static void write(string msg)

{

using (StreamWriter sw = new StreamWriter(@"E:\def\9.txt"))

{

sw.WriteLine(msg);

}

}

缺点是是后写入的数据会覆盖新写入的数据

1. 一个简单的写日志的类

public static class Log

{

private static string path1 = Assembly.GetExecutingAssembly().Location;

private static string path2 = Path.GetDirectoryName(path1);

public static void write\_ftp\_log(string msg)

{

string tempfolder = Path.Combine(path2, "log");

if (!Directory.Exists(tempfolder))

{

Directory.CreateDirectory(tempfolder);

}

string filepath = Path.Combine(tempfolder, "log.txt");

if (!File.Exists(filepath))

{

File.Create(filepath);

}

using (StreamWriter sw = new StreamWriter(filepath, true))

{

//sw.WriteLine();

sw.WriteLine(msg);

}

}

2016年3月21日20:13:11

SQL数据库相关操作

1. **数据库的连接**

(1)方法一，最常用的连接方法

String constr=”Server=10.17.200.131;Database=NEWDB;Integrated Security=false;User ID=sa;pwd=123456”;

SqlConnection conn = new SqlConnection(constr);

conn.open();

//执行命令等

conn.Close();

注意操作完后要断开连接，用close方法。连接的字符串可以写到应用程序的配置文件里面，在visusal studio里面就是app.config文件，编译好的文件就是执行程序exe.config文件，这个文件是xml格式的文件，用ConfigurationManager类来读取就可以了。

·public static string GetData(string key)

{ return ConfigurationManager.AppSettings[key];

}

servername = GetData("SERVERNAEM");

databasename = GetData("DATABASE");

user = GetData("USER");

password = GetData("PWD");

string constr = String.Format(@"Server= {0};Database={1};Integrated Security=false;User ID={2};pwd={3}", servername, databasename, user, password);

<appSettings>

<add key="SERVERNAEM" value="10.17.200.131"/>

<add key="DATABASE" value="NEWDB"/>

<add key="USER" value="sa"/>

<add key="PWD"/>

</appSettings>

（2）方法二，通过ConfigurationManager.ConnectionString获取数据库的配置信息,前面的方法将连接配置写到config的方法比较过时了。

<connectionStrings>

<add name=" ConnStr "

providerName="System.Data.SqlClient"

connectionString="Server=10.17.200.131;integrated security=SSPI;database=NEWDB"/>

</connectionStrings>

string connStr = ConfigurationManager.ConnectionStrings["ConnStr"].conncetionString;

前面是config里面的配置，后面是C#代码里面获取连接字符串

注意写法，红色部分是一个连接的名字，由这个名字找到连接字符串，因此这个名字ConnStr必须和config里面的name="ConStr"一致。

连接部分和前面的一样，用SqlConnection来连接。

(3) 方法三，用工厂方法

DbProviderFactory mdbProviderFactory = DbProviderFactories.GetFactory(ConfigurationManager.ConnectionStrings["ConStr"].ProviderName);

DbConnection conn = mdbProviderFactory.CreateConnection();

1. 数据库的断开连接

需要及时释放资源，必须及时断开数据库的连接，如果忘了断开连接，会导致各种问题。可以使用using语句来保证退出代码块后立即释放实现Idisopsable接口。

SqlConnection conn = new SqlConnection(constr);

using (conn)

{

conn.Open();

SqlCommand cmd = new SqlCommand(select, conn);

using (SqlDataReader aReader = cmd.ExecuteReader())

{

while (aReader.Read())

Console.WriteLine("'{0}' from {1}", aReader.GetString(0), aReader.GetString(1));

aReader.Close();

}

conn.Close();

}

1. 命令SqlCommand

3.1 把sql语句和SqlConnection对象作为的构造函数参数传递给sqlcommand，就定义好了一个命令，或者使用无参的构造方法，然后定义名参数。

1. //SqlCommand cmd = new SqlCommand(select, conn);

（2） SqlCommand cmd = new SqlCommand();

cmd.CommandText = select;

cmd.Connection = conn;

在此之前都要用conn.Open()打开连接。

3.2执行命令，执行语句有多种方式，取决于要从命令中返回什么数据

ExecuteNonQuery 执行命令，但不返回任何结果(书上是这么写的，但实际上应该是返回受影响的行数，一个int类型的数据)

ExecuteReader，返回一个类型化的IdataReader，（SqlDataReader）

ExecuteScalar，执行命令，返回结果集中的第一行第一列的值

* + 1. ExecuteNonQuery 这个方法一般用于UPDATA、INSERT、DELETE语句
    2. ExecuteReader返回一个类型化的datareader对象，可用于遍历返回的记录，和SqlDataAdapter里面的DataTable类似

string select = "SELECT ContactName,Age FROM [NEWDB].[dbo].[Customers]";

SqlConnection conn = new SqlConnection(GetDatabaseConnection());

conn.Open();

SqlCommand cmd = new SqlCommand(select, conn);

SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

Console.WriteLine("Contact: {0,-20} Company: {1}",

reader[0], reader[1]);

}

SqlDataAdapter的方法

public DataTable getDataTable(string sql) //获取dataTable

{

SqlDataAdapter sda = new SqlDataAdapter(sql, conn);

DataTable dt = new DataTable();

sda.Fill(dt);

return dt;

}

* + 1. ExecuteScalar

string select = "SELECT COUNT(\*) FROM Customers";

SqlConnection conn = new SqlConnection(GetDatabaseConnection());

conn.Open();

SqlCommand cmd = new SqlCommand(select, conn);

object o = cmd.ExecuteScalar();

Console.WriteLine(o);

* + 1. ExecuteXmlReader

static void ExecuteXmlReader()

{

string select = "SELECT ContactName,Age " +

"FROM Customers FOR XML AUTO";

SqlConnection conn = new SqlConnection(GetDatabaseConnection());

conn.Open();

SqlCommand cmd = new SqlCommand(select, conn);

XmlReader xr = cmd.ExecuteXmlReader();

xr.Read();

string data;

do

{

data = xr.ReadOuterXml();

if (!string.IsNullOrEmpty(data))

Console.WriteLine(data);

} while (!string.IsNullOrEmpty(data));

conn.Close();

}

2016年3月25日17:23:37

1. C#使用锯齿数组，必须要多次实例化，外层实例化后，内层每行都要实例化

byte[][] all\_data = new byte[9][];

all\_data[0]= new byte[8];

all\_data[1]= new byte[]7;

…

如果每行不单独实例化，则会报错。

1. 普通二维数组的声明方法是

**int**[,] arrray1=**new** **int**[2,5]{{9,9,9,9,0},{0,0,9,0,0}};

和c语言不一样，C语言类似C#的锯齿数组

2016年3月25日19:37:25

1. 格式化时间

DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");

2016年3月26日11:30:42

1. 让supersocket的服务安装后自动启动

在SocketServiceInstaller.cs文件里面添加

protected override void OnAfterInstall(System.Collections.IDictionary savedState)

{

System.ServiceProcess.ServiceController sc = new System.ServiceProcess.ServiceController(serviceInstaller.ServiceName);

if (sc != null)

{

sc.Start();

}

}

2016年3月28日17:05:30

1. Configuration保存数据到configuration里面

public static void SaveData(string key, string value)

{

Configuration config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);

config.AppSettings.Settings.Remove(key);

config.AppSettings.Settings.Add(key, value);

config.Save(ConfigurationSaveMode.Full);

ConfigurationManager.RefreshSection("appsetting\_\_\_111");

}

2016年3月29日10:49:50

1. 问号的作用？

1.可空类型修饰符(？)：引用类型可以使用空引用表示一个不存在的值，而值类型通常不能表示为空，例如:string str=null;是正确的。int i=null；编译器将报错。为了使值类型也可为空，可空类型出现了，可空类型使用可空类型修饰符？来表示，表现形式为T?。例：int?表示是可空的整形，DateTime?表示为可空的时间。T?其实是System.Nullable<T>（泛型结构）的缩写形式，也就意味着当你用到T？时编译器在编译时会把T？编译成System.Nullable<T>的形式，例如：int?,编译后便是System.Nullable<int>的形式。

public DateTime? Day { get; set; }

2、三元（运算符）表达式(？:)：如果这个你也不了解的话(我很不相信)，那你就当我也不了解吧，不再赘述。

      3、空合并运算符(？？)：用于定义可空类型和引用类型的默认值。如果此运算符的左操作数不为 null，则此运算符将返回左操作数；否则返回右操作数。 例：a??b,当a为null时则返回b,a不为空时返回a本身。空合并运算符为右结合运算符，即操作时从右向左进行组合的。如，“a??b??c”的形式按“a??(b??c)”计算。

2016年3月31日22:23:44

1. 将坝光的数据移植到缙云山系统要修改的问题
2. 首先注释掉鹰眼查看的代码，原来的鹰眼加载的是县界或者乡界，现在把关的系统没有这个数据，再加载就会出问题。
3. 坐标的问题

在二维还好，三维单独也行，但是联动就出问题了，根本连不到一起，问题就出现在坐标转换问题。Citymaker默认的就是wgs84的坐标，缙云山的是xian80的坐标，因此，程序里面有个坐标转换的过程，但是用在坝光系统就多余了，两个都是wgs84的坐标，不用转换，转换了反而出问题了。

修改方法：找到WetGIS.DCTEngine.Commands下的LinkDCTApp类，修改第78行如下

//DSA.Instance.CoordSys.Reproject(from, to, ref x, ref y);

DSA.Instance.CoordSys.Reproject(to, to, ref x, ref y);

1. 水位模拟的修改

找到WetGIS.DCTEngine.Dialogs下的waterLevelControlFrm窗体，修改里面的zoomTrackBarControl1控件的属性，展开property，修改里面的最大值和最小值，最大值设置为20，最小值设置为0即可。程序里面是将这个控件的值除以10，然后作为水深的。

2016年7月5日12:27:44

1. C#获取程序集的资源文件

程序集简单说就是编译过的exe或者dll，程序集的类名称是Assembly，因此首先要获得Assembly的对象。

获取主程序的程序集用： Assembly asmbly = Assembly.GetExecutingAssembly();

如果程序集是dll，在别的项目里面，可以用load方法加载

 通过程序集名称返回Assembly对象  
        Assembly ass = Assembly.Load("ClassLibrary831");  
    通过DLL文件名称返回Assembly对象  
        Assembly ass = Assembly.LoadFrom("ClassLibrary831.dll");

如果知道一个对象，可以用它的getType方法来获取Type类型，该方法在object类中，然后调用Assembly属性，即可以获得程序集。

获得了程序集后，就可以获得里面的资源了，比如有一个xml的文件，编译好后，只有exe或者dll，看不到xml，就可以用GetManifestResourceStream来获取。前提是资源文件的生成方式一定要是嵌入的资源，在文件的属性编辑里面可以设置。

Stream stream2 = asmbly2.GetManifestResourceStream("ResourceClass." + filename2);

该方法得到数据流，然后就可以读出数据了。

string str = "";

using (var stream3 = asmbly2.GetManifestResourceStream("ResourceClass." + filename2))

{

var buffer = new byte[stream.Length];

stream.Read(buffer, 0, buffer.Length);

str = System.Text.Encoding.Default.GetString(buffer);

}

2016年7月7日13:22:52

1.Ribbon开发中汉字自动换行的问题

每个按钮下面都有文字说明，参考程序是用的英文，而我要用的是中文。也许是中文要比英文宽的原因，当我给设置为两个汉字的时候就自动换行，一上一下，在splitbutton中还有个展开的三角符号，导致上下都不对称了，非常不美观。我看到系统的画图板里面的splitbutton的汉字都是在一行的。

2017年9月15日09:31:40

1. 快捷打开visual studio，在运行(win+R)里面输入devenv
2. C#的占位符

string address = "天朝 帝都";

string name = "张三";

int age = 20;

Console.WriteLine("我的地址是{0}，我的名字是{1}，我的年龄是{2}", address, name, age);

1. 变量交换

只给出两个int类型的变量，不用第三方变量交换两个变量的值。

int numberOne = 10;

int numberTwo = 20;

numberOne = numberOne - numberTwo;

numberTwo = numberOne + numberTwo;

numberOne = numberTwo - numberOne;

主要是存两个量之间的差值。

1. Console.ReadLine();的返回值类型是string
2. 转义字符

a. char c = '\"';//此时\ 不表示字符 而表示 转义的意思

char类型本身只能表示一个字符。

b. Console.WriteLine("\b今天天气好晴\b朗又是刮\b风又是下雨\b");

输出：今天天气好朗又是风又是下雨

\b转义字符表示退格键

1. double 类型

double a = 10.0 / 3;

Console.WriteLine(a);

输出：3.33333333333333

可以在占位符里面指定输出小数位数

Console.WriteLine("{0:0.000}", a);

输出3位小数

1. 小数类型float ,double,decimal

三个都可以表示小数类型，但是精度和位数不一样

float 单精度浮点 32bit，  
double 双精度浮点64bit，  
decimal是高精度 128bit，浮点型。  
float double 是 基本类型（primitive type），decimal不是。  
float [有效数字](https://www.baidu.com/s?wd=%E6%9C%89%E6%95%88%E6%95%B0%E5%AD%97&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dhnW7bn1nzuAR1uWK9rHck0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWDkrHbdP10zPHfvP101PHfzPs)7位，范围 ±1.5 × 10E−45 to ±3.4 × 10E38  
double [有效数字](https://www.baidu.com/s?wd=%E6%9C%89%E6%95%88%E6%95%B0%E5%AD%97&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dhnW7bn1nzuAR1uWK9rHck0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWDkrHbdP10zPHfvP101PHfzPs)15/16 位，范围 ±5.0 × 10 E−324 to ±1.7 × 10E308  
decimal [有效数字](https://www.baidu.com/s?wd=%E6%9C%89%E6%95%88%E6%95%B0%E5%AD%97&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dhnW7bn1nzuAR1uWK9rHck0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWDkrHbdP10zPHfvP101PHfzPs) 28/29 位，范围 ±1.0 × 10E−28 to ±7.9 × 10E28

如果希望实数被视为 **decimal** 类型，请使用后缀 m 或 M，

要定义一个decimal类型，必须在后面加上m或M,否则提升错误。

定义一个double类型，要在后面加d或者D，但不加好像也行。

定义一个float类型，要在后面加f或者F。

1. **C#中的char类型和c语音中的char类型不一样**

C#中的char占两个字节，使用Unicode编码，而c语言中的char只占一个字节，因此在C#中char可以表示一个汉字。

char sex = '男';

Console.WriteLine(sex);

上面是可以的，

char sex = '男女'; 这样是不可以的

这个特点C#和java是一样的，java也可以这样表示

char s='A';

char s1='男';

System.out.println(s);

System.out.println(s1);

Char类型可以直接赋值一个整数，应该表示在ascii中的编码

char[] chars = new char[4];

chars[0] = 'X'; // Character literal

chars[1] = '\x0058'; // Hexadecimal

chars[2] = (char)88; // Cast from integral type

chars[3] = '\u0058'; // Unicode

输出：X X X X

1. 类型不兼容的转换

比如将string转换位int，使用Convert来转换

string bb = "123";

int cc = Convert.ToInt32(bb);

注意string中的内容必须是数字，不能是其他的字符。

String转char：

string nn = "男";

char mm = Convert.ToChar(nn);

使用Console.ReadLine()输入的是字符串，比如输入的是135，要把它当数字来使用就必须要用Convert来转换。

1. 随机数

Random r = new Random();

int rNumber = r.Next(1, 7);

1. 枚举，结构

public enum QQState

{

OnLine=1,

OffLine,

Leave,

QMe,

Busy

}

C#有结构体和枚举，结构体是一次性声明多个不同类型的变量，而枚举是一次性声明多个相同类型的变量。

public struct AddressBook

{

//字段、属性、方法、事件

}

一次性声明多个相同类型的变量也可以使用数组。

**数组的4种声明方法：**

int[] nums = new int[3]；

int[] nums = {5,3,8}；

in[]nums=new int[3]{1,2,3}；

int[]nums=new int[]{2,3,5};

1. Console.WriteLine("我\0爱\0你\0爱\0我\0嘛？");

输出：我 爱 你 爱 我 嘛？

\0表示空格

#### 2017年9月16日22:10:03

1. Out和ref参数

两者都是按地址传递，使用后都会改变原来参数的值

ref 关键字使参数按引用传递。其效果是，当控制权传递回调用方法时，在方法中对参数的任何更改都将反映在该变量中。若要使用 ref 参数，则方法定义和调用方法都必须显式使用 ref 关键字。

class RefExample

{

    static void Method(ref int i)

    {

        i = 44;

    }

    static void Main()

    {

        int val = 0;

        Method(ref val);

        // val is now 44

    }

}

定义和调用都要添加ref参数。

不同之处在于 ref 要求变量必须在传递之前进行初始化。若要使用 out 参数，方法定义和调用方法都必须显式使用out 关键字。

**区别**：a. 使用out，在函数退出前必须给变量赋值，若用ref就不必。比如

public static void fucx(ref int a)

{

a ++;

}

这样是允许的，若是换成out则报错。

b. 使用ref，变量必须在调用前赋值，而out不必。比如

int x=80;

int y;

x可以用做ref的变量，y则不能，y只能用out。

总结：ref和out都是地址传递，调用后会修改原值，并且都要在调用的时候加上ref或out参数。区别是，ref要在调用前赋值，out要在调用玩之前赋值。

out适合用在需要retrun多个返回值的地方，而ref则用在需要被调用的方法修改调用者的引用的时候。

2017年9月18日11:01:51

1. 总结一个界面和业务代码分离的技巧，这是以前的管理系统使用的方式，发现arcgis也用了这种模式，这个经验应该是总结过的，但是比价可能掉了。

利用接口来实现界面和代码的分离。通常情况下，button的点击事件都是写在form代码里面的onclick事件里面，如果有很多按钮或别的工具，若都写在一个文件里面，肯定不方便。于是建立一个ICommand接口：

public interface ICommand

{

void Create(object item);

string Caption

{

get;

}

string Hint

{

get;

}

}

他有一个方法和和两个属性，通过oncreate方法来绑定按钮也业务代码。Caption和hint是标题和提示。然后写一个抽象类Command继承这个接口

public abstract class Command : ICommand

{

/// <summary>

/// 初始化

/// </summary>

/// <param name="item"></param>

public abstract void Create(object item);

/// <summary>

/// 标签

/// </summary>

public abstract string Caption

{

get;

}

/// <summary>

/// 提示

/// </summary>

public abstract string Hint

{

get;

}

}

然后真正的业务代码就继承这个抽象类就可以了。

比如在主窗体里面绑定按钮只需要两句话即可

//刷新视图

RefreshViewApp pRefreshView = new RefreshViewApp();

pRefreshView.Create(RefreshView\_btn);

然后RefreshViewApp类继承commnad类，实现oncreate方法。

private BarButtonItem \_ButtonItem = null;

public override void Create(object buttonItemHook)

{

\_ButtonItem = buttonItemHook as BarButtonItem;

\_ButtonItem.ItemClick += new ItemClickEventHandler(\_ButtonItem\_ItemClick);

\_ButtonItem.Caption = Caption;

\_ButtonItem.Hint = Hint;

}

其中\_ButtonItem.ItemClick += new ItemClickEventHandler(\_ButtonItem\_ItemClick);

是绑定点击事件。

2018年9月12日09:41:19

1. 使用WindowsFormsApplicationBase来设计winform 的单例和启动窗体。

这是微软提供的一个非常好的winfrom启动初始化框架。

1. 使用它需要导入Microsoft.VisualBasic.dll
2. 然后创建一个自定义类，继承自WindowsFormsApplicationBase
3. 在构造方法厘米设置单例属性为true

base.IsSingleInstance = true;

1. 复写里面的OnCreateMainForm()、OnStartupNextInstance()等方法
2. 调用方式，在program.cs的main方法里面修改。

先给main方法添加参数，原来是没有参数的，增加为string[] args

然后new出前面自定义类的对象，然后调用run方法。

SingleInstanceApplication sg = new SingleInstanceApplication();

sg.Run(args);

1. protected override void OnCreateSplashScreen()

{

base.OnCreateSplashScreen();

this.MinimumSplashScreenDisplayTime = 50;

this.SplashScreen = new SplashForm();

}

这个方法是创建闪屏的方法，并且，这个方法会让你的主窗体初始化完成后才结束闪屏。不管你的演示时间设置的有多小。

如果设置的大于主窗体加载时间，则延时时间为该设置值。

而且，当主程序已经运行后，再次启动，也不会显示闪屏了。

1. 除此之外，这个类还可以创建系统托盘。

2018年9月13日21:35:33

1. 一个比较坑的地方

最近写自定义控件，其实是在别人的基础上修改的。做的是一个checkbox，但是发现字体模糊，暗淡。

无论是怎么修改字体，颜色都是这样。

找了好几发现是在onPaint下面多了这么一句话：

g.TextRenderingHint = TextRenderingHint.AntiAlias;

删掉这句代码就好了。

2018年9月14日22:07:41

1. Usb FM项目的主界面都做的差不了，现在需要移植usb hid功能，这个也是有开源的代码，而且参考例子都写得很简单明了。可是还是遇到坑了。

直接在原来的项目里面添加这个UsbLibrary项目，编译正常，运行主界面都正常，就是连不上设备，检查了好多地方，都没发现问题。

然后新建了一个简单的winform项目来测试，同样导入这个库，一样不行。可是人家的代码就没问题。

接下来我就在人家的项目上新建了一个窗体，用它来测试，只有一个按钮和标签，点击按钮连接设备，标签显示连接状态。

没想到这样就行了。

可是也不能在人家的项目上修改啊。

然后我决定重新建一个项目，然后在分别移植material和usb，没想到这样就可以了。

所以说到现在我都不知道为什么之前连不上，只是现在可以了。

2018年11月4日23:59:50

1. winform中textbox在enable=false下的颜色设置。

实际上在textbox的enable属性设置为false后，颜色是没法修改的，只能是灰黑色，不管之前设置的是什么颜色。这样似乎不太好。

不过在csdn上看到一个解决办法，使用了c++的方式来修改。

原帖链接： <https://bbs.csdn.net/topics/350241687>

[System.Runtime.InteropServices.DllImport("user32.dll ")]

public static extern int SetWindowLong(IntPtr hWnd, int nIndex, int wndproc);

[System.Runtime.InteropServices.DllImport("user32.dll ")]

public static extern int GetWindowLong(IntPtr hWnd, int nIndex);

public const int GWL\_STYLE = -16;

public const int WS\_DISABLED = 0x8000000;

public static void SetControlEnabled(Control c, bool enabled)

{

if (enabled)

{ SetWindowLong(c.Handle, GWL\_STYLE, (~WS\_DISABLED) & GetWindowLong(c.Handle, GWL\_STYLE)); }

else

{ SetWindowLong(c.Handle, GWL\_STYLE, WS\_DISABLED + GetWindowLong(c.Handle, GWL\_STYLE)); }

}

private void button2\_Click(object sender, System.EventArgs e)

{

SetControlEnabled(this.button1, false);

this.button1.ForeColor = Color.White;

}

把user32.dll引入后，定义一个SetControlEnabled方法，然后就可以直接使用这个方法来设置控件的enabl属性了，参数就是控件对象，和是否enable。

2018年11月5日22:48:23

1. 真的很奇怪

文本框设置Focus()后确实是获得焦点了，并且立即输出它的Focused属性，输出也是true。

接下来就是奇怪的地方了，点击窗体空白处，在winform的mousedown事件里面输出文本框的focused属性，得到false。也就是失去焦点了。可是在键盘上还是能输入数据，并且光标也在啊。

这个现象只出现在了我的MyUSBFM项目里面。

在我写的简单的testLostFocus项目里面却没有出现。

1. 写了一下午和一晚上的文本框失去焦点的方法，最终用了一个很简单的技巧实现了。好不容易。

winform里面的textbox只能获得焦点，不能通过代码直接让其失去焦点。textbox的Focuse属性是一个只读的属性，不能设置值。而且方法里面关于焦点的只有Focus()是获得焦点的，但没有LostFocus()方法。也就是不能失去焦点。

现有的方法一是点击窗体上的别的控件，如果文本框一开始是获得焦点的话，这时会触发它的LostFocus事件。我并不需要点击别的控件啊，我的需求是不仅点击别的控件会失去焦点，点击窗体空白处也会失去焦点。

另一种就是使用user32.dll来发送windows消息。

private const int WM\_KILLFOCUS = 0x0008;

[DllImport("User32.dll", EntryPoint = "SendMessage")]

private static extern int SendMessage(IntPtr hWnd, int Msg, IntPtr wParam, string lParam);

在窗体上点击空白处的调用方法是：

PostMessage(hWnd, WM\_KILLFOCUS, 0, 0);

这种方法确实触发了textbox的LostFocus事件，但是根本没有改变它的Focused属性，它依然是获得焦点的。依然能编辑，光标依然闪烁。

这种方法根本不行。

后来偶然看到了一个非常巧妙的方法：将焦点转移到别的控件上，来触发textbox的lostfocus事件。

比如，在窗体上有个label。在窗体的空白处点击事件里写 ：label.Focus() 这样就能让文本框完全失去焦点了。

最好不用讲焦点转移到按钮上，转移到按钮上后如果按enter键会触发它的click事件。