= 是后面赋值给前面。前面不是确定值，后面是确定值。

Shift+R 是事件的快捷键

**Panel.SetZIndex(UIElement, Int32) 方法：**设置给定元素的ZIndex附加属性的值，

public static void SetZIndex (System.Windows.UIElement element, int value)

给定 element的 value 越大，element 在前台出现的可能性就越大。 同样，如果 element 的 value相对较低，则 element 可能会在背景中出现。 例如，value 为5的 element 将显示在其 value 为4的 element 上方，而该将显示在值为3的 element 上，依此类推。 允许使用负值，并继续使用此优先级模式。

**像素：每张图片都是由色点组成的，每个色点称为一个像素。**一张图片由30万个色点组成，这个图片的像素就是30W。**图像分辨率:图像分辨率是指每英寸图像内的像素点数。**图像分辨率是有单位的，叫 像素每英寸。分辨率越高，像素的点密度越高，图像越逼真。**屏幕分辨率：**屏幕分辨率是屏幕每行的像素点数\*每列的像素点数，每个屏幕有自己的分辨率。屏幕分辨率越高，所呈现的色彩越多，清晰度越高。

px就是表示pixel，像素，是屏幕上显示数据的最基本的点；

pt就是point，是印刷行业常用单位，等于1/72英寸。

**绝对长度单位。点（Point）。1in（英寸）= 2.54cm = 25.4 mm = 72pt（点） = 6pc（派卡）**

**///也是一种注释，但是这种注释主要有两种作用：**

**1、即是上面大家所说的这些注释能够生成一个XML文件。这样呢看起来会对程序的内容一目了然。**

**2、以这种方式对你的类，属性，方法等等加上注释之后，当你在其他地方使用以这种方式加上的类，属性，方法等等地时候，黄色的提示框里面会把你写的注释显示出来，是你更加明白你要使用的功能的作用。**

**return**

**表示从被调函数返回到主调函数继续执行，返回时可附带一个返回值，由return后面的参数指定。**

**中断功能，与break类似 结束该函数 执行下一函数 返回整个大函数**

**//π弧度＝180° 1弧度＝180°/ π （≈57.3°）**

**//arctan(-A) = -arctan(A)**

**//Atan = arctanx PI = 3.1415926**

**//角度= （弧度\*180）/PI**

# // C++/C#/C语言中cos,sin,asin,acos这些三角函数操作的是弧度,而非角度

**c# String.Format数字格式化输出**

**C 或 c货币 Console.Write("{0:C}", 2.5); //$2.50 Console.Write("{0:C}", -2.5); //($2.50)  
D 或 d十进制数 Console.Write("{0:D5}", 25); //00025  
E 或 e科学型 Console.Write("{0:E}", 250000); //2.500000E+005  
F 或 f固定点 Console.Write("{0:F2}", 25); //25.00 Console.Write("{0:F0}", 25); //25  
G 或 g常规 Console.Write("{0:G}", 2.5); //2.5  
N 或 n数字 Console.Write("{0:N}", 2500000); //2,500,000.00 n后面是几就保留几位小数  
X 或 x十六进制 Console.Write("{0:X}", 250); //FA Console.Write("{0:X}", 0xffff); //FFFF  
百分数格式应该用“p”这个参数。 格式 原始数据 结 果 "{0:P}" 0.40 40%**



**“字段”是包含在类中的对象的值，字段使类可以封装数据，字段的存储可以满足类设计中所需要描述。**

**class Animal  
    {  
        public string color;           //声明颜色字段  
        public bool haveFeather;          //声明是否含有羽毛字段  
        public int age;            //年龄字段  
    }  
上述代码中，对Animal类声明了另外两个字段，用来描述是否有羽毛和年龄。当需要访问该类的字段的时候，需要声明对象，并使用点“.”操作符实现，示例代码如下所示。  
            Animal bird = new Animal();         //创建对象  
            bird.haveFeather = true;         //鸟有羽毛  
            bird.color = "black";          //这是一只黑色的鸟**

**属性是类中可以像类的字段一样访问的方法。属性可以为字段提供保护，避免字段在用户创建的对象不知情的情况下下被更改。属性机制非常灵活，提供了读取、编写或计算私有字段的值，可以像公共数据成员一样使用属性。  
在C#中，它们被称为“访问器”，为C#应用程序中类的成员的访问提供安全性保障。当一个字段的权限为私有（private）时，不能通过对象的“.”操作来访问，但是可以通过“访问器”来访问，示例代码如下所示。  
    public class Animal  
    {  
        private int \_age;             //定义私有变量  
        public int Age { get { return \_age; } set { \_age = value; } }     //赋值属性  
    }  
上述代码中为Animal类声明了一个属性Age，在主程序中，同样可以通过“.”操作符来访问属性，示例代码如下所示。  
            Animal bird = new Animal();          //创建对象  
            bird.Age = 1;            //Age访问了\_age  
在Visual Studio 2008中，属性的声明被简化，不再需要冗长的声明，示例代码如下所示。  
    public class Animal             //创建类  
    {  
        public int Age { get; set; }           //简便的属性编写  
    }**

**方法用来执行类的操作，方法是一段小的代码块。**

**在C#中，方法接收输入的数据参数，并通过参数执行函数体，返回所需的函数值，方法的语法如下所示。  
        私有级别 返回类型 方法名称(参数1，参数2)  
        {  
            方法代码块。  
        }  
方法在类中声明。对方法的声明，需要指定访问级别、返回值、方法名称以及任何必要的参数。参数在方法名称后的括号中，多个参数用逗号分割，空括号表示无参数，示例代码如下所示。  
        public string output()           //一个无参数传递的方法  
        {  
            return "没有任何参数";          //返回字符串值  
        }  
        public string out\_put(string output)        //一个有参数传递的方法  
        {  
            return output;           //返回参数的值  
        }**

**上述代码中，创建了两个方法，一个是无参数传递方法output和一个参数传递的方法out\_put，在主函数中可以调用该方法，调用代码如下所示。  
            Animal bird = new Animal();         //创建对象  
            bird.output();           //使用无参数的方法  
            string str = "我是一只鸟";         //创建字符串用于参数传递  
            bird.output(str);           //使用有参数的方法**

**.NET 2.0对串口的支持  
.NET 2.0提供了对串口通信功能的支持，有关类可以在命名空间System.IO.Ports下找到，这其中最为重要的是SerialPort类。  
通过创建一个新的SerialPort 对象，我们就可以在.NET程序中控制串口通讯的全过程。**

**3。使用 SerialPort 设置串口属性  
进行串口通讯时，需要设置一些相关参数，可以通过设置SerialPort 类的属性来进行。串口属性主要包括  
.PortName 串口名称，COM1, COM2等。  
.BaudRate 波特率，也就是串口通讯的速度，进行串口通讯的双方其波特率需要相同，如果用PC连接其他非PC系统，一般地，波特率由非PC系统决定。  
.Parity 奇偶校验。可以选取枚举Parity中的值  
.DataBits 数据位  
.StopBits 停止位，可以选取枚举StopBits中的值  
.Handshake 握手方式，也就是数据流控制方式，可以选取枚举Handshake中的值**

**4。打开与关闭串口  
  
  
在创建一个SerialPort 对象，设置串口属性后，可以通过 Open()方法打开串口。数据读写完成后，可以通过Close()方法关闭串口。  
根据经验，对于有些系统，在打开串口后，还需要将RtsEnable设置为True，这样才能读写数据，否则不能正常读写数据。**

**5。读写行数据  
双方通讯时，一般都需要定义通讯协议，即使最简单的通过串口发送文本聊天的程序。  
通常是在当一方按下回车时，将其所数据的文本连同换行符发给另一方。在这个通讯事例中，协议桢是通过换行符界定的，每一桢数据都被换行符隔开，这样就很容易识别出通讯双发发送的信息。  
在以上的例子中，可以用WriteLine()来发送数据，用ReadLine()来读取数据。WriteLine发送完数据后，会将换行符作为数据也发送给对方。ReadLine()读取数据时，直至遇到一个换行符，然后返回一个字符串代表一行信息。换行符可以通过SerialPort 的属性NewLine来设置。一般地，Windows将CrLn作为换行符，而在Linux下，换行符则只用一个Ln表示。  
ReadLine()方法是阻塞的，直至遇到一个换行符后返回。在读取数据时，如果一直没有遇到换行符，那么在等待ReadTimeout时间后，抛出一个TimeoutException。默认情况下，ReadTimeout为InfiniteTimeout。这样，ReadLine一直处于阻塞状态，直至有新一行数据到达。  
WriteLine()方法也是阻塞的，如果另一方不能及时接收数据，就会引起TimeoutException异常。  
由于ReadLine()和WriteLine()方法都是阻塞式的，在程序使用SerialPort 进行串口通讯时，一般应该把读写操作交由其他线程处理，避免因为阻塞而导致程序不响应。  
  
6。读写字节或字符数据  
对于字节或字符数据，用Read()方法来读数据，该方法需要一个字节或字符数组作为参数来保存读取的数据，结果返回实际读取的字节或字符数。写数据使用Write()方法，该方法可以将字节数组、字符数据或字符串发送给另一方。  
如果通讯双方交换的数据位字节流数据，要构建一个使用的串口通讯程序，那么双方应该定义数据桢格式。通常数据桢由桢头和桢尾来界定。  
发送数据比较简单，只需要将构造好的数据用Write()方法发送出去即可。  
接收数据则比较复杂，通讯是以字节流的形式到达的，通过调用一次Read()方法并不能确保所读取的数据就是完整一桢。因此需要将每次读取的数据整合在一起，对整合后的数据进行分析，按照定义的桢格式，通过桢头和桢尾，将桢信息从字节流中抽取出来，这样才能获取有意义的信息。  
除了利用Read()方法来读数据，还可以使用ReadExisting()方法来读取数据。该方法读取当前所能读到的数据，以字符串的形式返回。**

**Try catch finally语句**

**将可能引发异常的代码节放在 Try 块中，而将错误信息打印出来或者处理异常的代码放在 Catch 块中，finally则表示不管前面发声什么 一定会执行的代码。**

**波特率（BaudRate），模拟线路信号的速率，也称调制速率，以波形每秒的振荡数来衡量。波特率是指数据信号对载波的调制速率，它用单位时间内载波调制状态改变的次数来表示。  
  在信息传输通道中，携带数据信息的信号单元叫码元，每秒钟通过信道传输的码元数称为码元传输速率，简称波特率。波特率是传输通道频宽的指标。**

**COM 端口是信息传输通道之一。速度较慢。**

**Form\_Load:装载窗口事件, 是窗体启动时触发的事件**

**C#窗体包含一个Load事件，该事件在窗口加载完毕后执行响应**

**delegate void MyDel(int value);    //声明委托类型**

**和类一样，委托是用户自定义的类型，但是类是数据和方法的集合，而委托是持有一个或多个方法。**

**delegate可以看做是一个包含有序方法列表的对象，这些方法具有相同的签名和返回类型。**

**委托保存的方法可以来自任何类或结构**

**调用列表中的方法可以是实例方法也可以是静态方法**

**调用委托时，会执行调用列表中的所有方法**

**FormClosing事件**

**在窗体关闭时，FormClosing事件发生。此事件会得到处理。从而释放与窗体相关的所有资源。**

**如果取消此事件，则窗体仍然保持打开状态。**

**当窗体显示为模式对话框时，单击“关闭”会隐藏窗体并将DialogResult属性设为Cancel。**

**通过在些事件中设置DialogResult属性可以在用户单击右上角关闭按钮时重写DialogResult的值。**

**FormClosed事件**

**在用户或Application类的Close方法或Exit方法关闭窗体后，会发生FormClosed事件。**

**可以使用此事件释放窗体的一些资源。还可以使用此事件保存输入窗体中的一些信息或者更新父窗体。**

**object sender:表示触发事件的控件对象  
EventArgs e：表示事件数据的类的基类**

**sender我们很容易猜到肯定是例子中触发单击事件的那个button控件，它类型是object。C#里的继承关系是所有类的基类，万物基于object（滑稽），所以控件同样也是继承与object的，所以将sender进行类型转换就能直接得到触发事件的控件对象。**

**1.using指令**

**using+命名空间，这种方法基本学习过C#的都用过，好处在于，写代码的时候不需要指定详细的命名空间。**

**using System;**

**2.using语句**

**是用来简化资源释放的，在一定的范围内有效，除了这个范围时，自动调用IDisposable释放掉，当然并不是所有的类都适用，只有实现了IDisposable接口的类才可以使用。**

**using (MySqlConnection conn = new MySqlConnection(connectionString)) //using语句，定义一个范围，在范围结束时处理对象。**

**{**

**PrepareCommand(cmd, conn, null, cmdType, cmdText, commandParameters);**

**int val = cmd.ExecuteNonQuery();**

**cmd.Parameters.Clear();**

**return val;**

**}//在这个大括号中，代码执行完毕之后，就会跳出这个大括号，而出了大括号之后，就等于出了using的这域。那么，using的这个域就会在内存中被释放，从而获得空闲空间。**

**因为stream占用缓存较大而且通常是使用独占方式，如果不及时释放会浪费资源，并造成其他线程无法访问文件。**

**DataGridViewColumnHeadersHeightSizeMode**

**成员名称 说明   
  AutoSize 列标题的高度根据所有列标题单元格的内容进行调整。     
  DisableResizing 用户不能使用鼠标调整列标题的高度。    
  EnableResizing 用户可以使用鼠标调整列标题的高度。**

**DataGridViewRowHeadersWidthSizeMode:**

**成员名称 说明   
  AutoSizeToAllHeaders 为适应所有行标头单元格的内容大小而进行的行标头宽度调整。     
  AutoSizeToDisplayedHeaders 为适应当前显示的行中所有行标头的内容大小而进行的行标头宽度调整。     
  AutoSizeToFirstHeader 为适应第一个行标头的内容大小而进行的行标头宽度调整。    
  DisableResizing 用户不能使用鼠标调整列标头的宽度。    
  EnableResizing 用户可以使用鼠标调整列标头的宽度。**

**Add：是将指定的对象添加到集合或者容器中  
AddRange:向集合或者容器中的末尾添加数据数组。  
一般来说，在群体操作时，都是用AddRange替代Add。因为AddRange可以将要加入的东西一次性加入，而Add一次只能添加一个，显然使用AddRange可以加快速度。**

**DataBindXY(IEnumerable, String, IEnumerable, String)**

**将集合中数据点的 X 值和 Y 值绑定到指定数据源的指定列的数据。**

**参数**

**xValue IEnumerable**

**将为数据点提供 X 值的数据源。**

**xField String**

**将为数据点提供 X 值的列的名称。**

**yValue IEnumerable**

**对象的 Y 值的逗号分隔列表。**

**yFields String**

**一个逗号分隔列表，其中列出了将应用数据点的 Y 值的列名称。 请注意，若要在列名称中嵌入逗号，可以使用双逗号。**

**注解**

**数据将集合中数据点的 X 值和 Y 值绑定到指定数据源的第一列。**

**可以通过将yFields参数设置为多个列名来绑定到多个 Y 值。 用逗号分隔列名称。**

**如果数据源未提供 Y 值，或者如果提供的值的数目不正确，则将引发异常。**

**如果不想显示同时使用 X 和 Y 值的散点图类型图形，请改用其中一DataBindY种方法。**

**KeyPress 和KeyDown 、KeyPress之间的区别**

1. **KeyPress主要用来捕获数字(注意：包括Shift+数字的符号)、字母（注意：包括大小写）、小键盘等除了F1-12、SHIFT、Alt、Ctrl、Insert、Home、PgUp、Delete、End、PgDn、ScrollLock、Pause、NumLock、{菜单键}、{开始键}和方向键外的ANSI字；而KeyDown 和KeyUp 通常可以捕获键盘除了PrScrn所有按键(这里不讨论特殊键盘的特殊键）   
   2、KeyPress 只能捕获单个字符；而KeyDown 和KeyUp 可以捕获组合键。   
   3、KeyPress 可以捕获单个字符的大小写；而KeyDown和KeyUp 对于单个字符捕获的KeyValue 都是一个值，也就是不能判断单个字符的大小写。   
   4、KeyPress 不区分小键盘和主键盘的数字字符；KeyDown 和KeyUp 区分小键盘和主键盘的数字字符。   
   5、其中PrScrn 按键KeyPress、KeyDown和KeyUp 都不能捕获。   
   注：3和4说明KeyDown捕获的是键码，键盘上每个不同的按键都有不同的键码，而大小写字母都是在一个键上，所有大小写字母的键码是相同的；主键盘和小键盘上的数字是不同的键，有不同的键码，所以用KeyDown是不同的值；KeyPres捕获的可以认为是ASCII码，所以区分大小写，不区分主键盘和小键盘的数字，毕竟无论小键盘还是主键盘都是0-9这些数字，它们的ASCII码是一致的。**

**Thread线程**

**定义线程 Thread threadA = new Thread（new ThreadStart（methodA）） 或**

**Thread threadA = new Thread（methodA）**

**启动线程 threadA.Start();**

**=> Lambda表达式 （参数） => {语句表达式} 把参数传递给语句表达式 也可无参数**

**你完全可以把delegate理解成C中的函数指针，它允许你传递一个类A的方法m给另一个类B的对象，使得类B的对象能够调用这个方法m，说白了就是可以把方法当作参数传递。不过delegate和函数指针还是有点区别的，delegate有许多函数指针不具备的优点。首先，函数指针只能指向静态函数，而delegate既可以引用静态函数，又可以引用非静态成员函数。在引 用非静态成员函数时，delegate不但保存了对此函数入口指针的引用，而且还保存了调用此函数的类实例的引用。其次，与函数指针相 比，delegate是面向对象、类型安全、可靠的受控（managed）对象。也就是说，runtime能够保证delegate指向一个有效的方法， 你无须担心delegate会指向无效地址或者越界地址。**

**Invoke用法**

**在用.NET Framework框架的WinForm构建GUI程序界面时，如果要在控件的事件响应函数中改变控件的状态，例如：某个按钮上的文本原先叫“打开”，单击之后按钮上的文本显示“关闭”，初学者往往会想当然地这么写：**

**void ButtonOnClick(object sender,EventArgs e)**

**{    button.Text="关闭";}**

**这样的写法运行程序之后，可能会触发异常，异常信息大致是“不能从不是创建该控件的线程调用它”。注意这里是“可能”，并不一定会触发该种异常。造成这种异常的原因在于，控件是在主线程中创建的（比如this.Controls.Add(...);)，进入控件的事件响应函数时，是在控件所在的线程，并不是主线程。在控件的事件响应函数中改变控件的状态，可能与主线程发生线程冲突。如果主线程正在重绘控件外观，此时在别的线程改变控件外观，就会造成画面混乱。不过这样的情况并不总会发生，如果主线程此时在重绘别的控件，就可能逃过一劫，这样的写法可以正常通过，没有触发异常。**

**正确的写法是在控件响应函数中调用控件的Invoke方法（其实如果大家以前用过C++ Builder的话，也会找到类似Invoke那样的激活到主线程的函数）。Invoke方法会顺着控件树向上搜索，直到找到创建控件的那个线程（通常是主线程），然后进入那个线程改变控件的外观，确保不发生线程冲突。正确写法的示例如下：**

**void ButtonOnClick(object sender,EventArgs e)**

**{**

**button.Invoke(new EventHandler(delegate**

**{**

**button.Text="关闭";**

**}));**

**}**

**Invoke方法需要创建一个委托。你可以事先写好函数和与之对应的委托。不过，若想直观地在Invoke方法调用的时候就看到具体的函数，而不是到别处搜寻的话，上面的示例代码是不错的选择。这样的写法有一个烦人的地方：对不同的控件写法不同。对于TextBox，要TextBoxObject.Invoke，对于Label，又要LabelObject.Invoke。有没有统一一点的写法呢？**

**主窗口类本身也有Invoke方法。如果你不想对不同的控件写法不一样，可以全部用this.Invoke：**

**void ButtonOnClick(object sender,EventArgs e)**

**{**

**this.Invoke(new EventHandler(delegate**

**{**

**button.Text="关闭";**

**}));**

**}**

**在C# 3.0及以后的版本中有了Lamda表达式，像上面这种匿名委托有了更简洁的写法。.NET Framework 3.5及以后版本更能用Action封装方法。例如以下写法可以看上去非常简洁：**

**void ButtonOnClick(object sender,EventArgs e)**

**{**

**this.Invoke(new Action(()=>**

**{**

**button.Text="关闭";**

**}));**

**}**

**以上写法往往充斥着WinForm构建的程序。在微软新一代的界面开发技术WPF中，由于界面呈现和业务逻辑原生态地分开在两个线程中，所以控件的事件响应函数就不必Invoke了。但是，如果手动开辟一个新线程，那么在这个新线程中改变控件的外观，则还是要Invoke的。**

**当一个控件的InvokeRequired属性值为真时，说明有一个创建它以外的线程想访问它，此时它将会在内部调用new MethodInvoker(LoadGlobalImage)来完成下面的步骤，这个做法保证了控件的安全，你可以这样理解，有人想找你借钱，他可以直接在你的钱包中拿，这样太不安全，因此必须让别人先要告诉你，你再从自己的钱包把钱拿出来借给别人，这样就安全了。**

# OpenFileDialog的FileName和FileNames

# 打开对话框的这两个方法均返回选定文件的全路径，包括目录及文件名和扩展名。（例E:\TEMP\OGRLayer\Intersection.shp）

**FileName可以单选和多选，返回值为string；若单选则返回此文件名，若多选则返回选定的第一个文件名；**

**FileNames可以单选和多选，返回值为string[]；若单选则返回此文件名，若多选则返回选定的所有文件名。**

**null和空的区别**

**1.声明的string类型的变量和属性以及字段在未赋值的情况下均为null，这个null不仅仅表示为无字符，更表示为空的引用。**

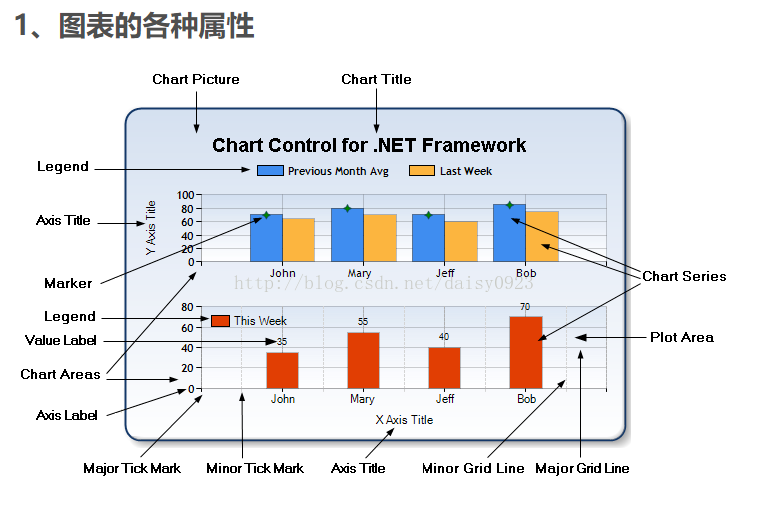
**比如：string userName；此时判断userName是否为空应该是if（userName==null）而不是if（userName==“”），并且使用userName.Equals(null)或userName.Equals(“”）均为错的，因为此时userName是个空引用，根本无法调用Equals()方法。但是如果声明为string userName=“”，则上述方法均成立。但是此时userName并不是null而是空值。**

**2.所有控件的Text属性的在未赋值的情况下是空值（""）, 而不是null。**

**string str = null 没有创建内存空间，str中存放的是空引用指针；**

**string str = "" 创建了内存空间，str中存放的是指向堆中的指针。**

**Clear只清数据，不清表头 //**





**static都是在类初始化的时候加载的，而非静态的变量都是在对象初始化的时候加载。**

**输入输出函数**

**//输出:System.Console.WriteLine()**

**Console.WriteLine("Hello World!!!");//完成后换行**

**Console.Write("hello world");//完成后不换行**

**//输入**

**string str = Console.ReadLine();//输入一行**

**Console.WriteLine(str);**

**Console.WriteLine("--------------");**

**char ch =(char)Console.Read();//输入一个字符**

**Console.WriteLine(ch);**

**Console.WriteLine("---------------");**

**int n = Convert.ToInt32(Console.ReadLine());//输入数字**

**Console.WriteLine(n);**

**Console.ReadKey();**