# 窗体传值

将一个值从一个窗体传递到另一个窗体，用来调用方法。

主要有2种方式：

1. 使用属性和构造方法
2. 使用事件委托

## 属性通过构造方法传值

### 案例：窗体2修改窗体1的背景色

点击窗体1中按钮，弹出窗体2。点击窗体2中的按钮会修改窗体1的背景色。

实现步骤：

1. 点击按钮时，创建并显示窗体2。在构造方法中把Form的实例窗体1传递给窗体2。

|  |
| --- |
| private void button1\_Click(object sender, EventArgs e)  {  Form2 form2 = new Form2(this);  form2.Show();  } |

1. 在窗体2的代码中重写构造方法

|  |
| --- |
| Random random = new Random();  Form1 form1;  public Form2(Form1 form1)  : this()  {  this.form1 = form1;  } |

1. 点击按钮时，改变窗体1的背景

|  |
| --- |
| private void button1\_Click(object sender, EventArgs e)  {  form1.BackColor = Color.FromArgb(random.Next(256), random.Next(256), random.Next(256));  } |

### 案例2：登录成功弹出主窗口

在Main方法中，把登录窗口以对话框的形式显示，当点击登陆按钮时，设置DialogResult属性。这时登录窗口会隐藏，并根据DialogResult值来判断登录是否成功，如果登录成功，则弹出主窗口。

在构建主窗口时，登录窗口中用户名作为属性值传递给主窗口，并在主窗口中显示出登录用户名。

1. 在Main窗口中，实例化登录窗口，并根据DialogResult值来实例化主窗口

|  |
| --- |
| static void Main()  {  Application.EnableVisualStyles();  Application.SetCompatibleTextRenderingDefault(false);  Form1 form1 = new Form1();  //每一个窗体都有一个DialogResult属性， 使用ShowDialog方法显示窗体的时候，如果给这个属性赋值，那么窗体会隐藏起来。  DialogResult dr = form1.ShowDialog();//显示登陆界面  if (dr == DialogResult.OK)  { //运行主界面。构造函数的参数为用户名信息  Application.Run(new MainForm(form1.UserUid));  }  else  {  MessageBox.Show(dr.ToString());  }  } |

1. 登录界面中，判断是否登录成功，设置不同的DialogResult。并设置属性保存用户名

|  |
| --- |
| public partial class Form1 : Form  {  public Form1()  {  InitializeComponent();  }  //属性用来保存用户名。  public string UserUid { get { return txtUid.Text.Trim(); } }  private void button1\_Click(object sender, EventArgs e)  {  string uid = txtUid.Text.Trim();  string pws = txtPwd.Text;  if (uid == "admin" && pws == "888")  {//设置DialogResult。这时登录界面会隐藏  this.DialogResult = DialogResult.OK;  }  else  {  this.DialogResult = DialogResult.No;  }  }  } |

1. 构造方法重载，并设置标签为登录的用户名

|  |
| --- |
| public partial class MainForm : Form  {  public MainForm()  {  InitializeComponent();  }  //构造方法重载  public MainForm(string uid)//属性传值  : this()  { //设置标签显示为用户名  this.label1.Text = uid;  }  } |

案例1，案例2，演示了如何通过构造方法实现属性值的传递。

## 事件实现值传递

这里用事件实现1.1.1中的案例：窗体2修改窗体1的背景色。

1. 新建一个cs文件，里面定义委托

|  |
| --- |
| namespace \_03\_事件实现传值  { //参数为Color类型  public delegate void EchangeColor(System.Drawing.Color color);  } |

1. 窗体2中定义事件变量，并定义触发事件

|  |
| --- |
| //事件变量  public event EchangeColor changeColor;  //点击按钮触发事件  private void button1\_Click(object sender, EventArgs e)  {  if(changeColor !=null)  {  Random r = new Random();  Color c = Color.FromArgb(r.Next(256), r.Next(256), r.Next(256));  //这个方法不是方法体本身，需要调到其他地方执行具体的方法。当方法执行结束再次回到这个地方，继续往下执行。这个就是典型的回调函数。  changeColor(c);  }  } |

1. 当点击窗体1中的按钮时，弹出窗体2，并为窗体2中的事件变量赋值。

|  |
| --- |
| private void button1\_Click(object sender, EventArgs e)  {  Form2 f2 = new Form2();  //为事件变量赋值  f2.changeColor += c => this.BackColor = c;  f2.Show();  } |

## 泛型EventHandler事件处理实现值传递

在父类中进行事件的注册调用

|  |
| --- |
| //1.定义事件  public event EventHandler<string> MEvent;  private void button1\_Click(object sender, EventArgs e)  {  //触发事件  MEvent(this, this.txtMsg.Text);  }  private void Form1\_Load(object sender, EventArgs e)  {  Form2 f2 = new Form2();  f2.Show();  //注册事件  MEvent += f2.SetText;  } |

在子类中实现方法

|  |
| --- |
| public void SetText(object sender, string str)  {  this.label1.Text = str;  } |

## 静态类传值

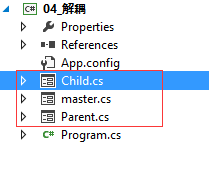
## 接口与中间类传值（解耦）

1. 当主窗体向子窗体传值时，如果不知道有多少个子窗体接受值，可以在父窗体中设计一个子窗体的集合。把要接受值的子窗体加入集合中。

2. 因为子窗体有相同的方法来处理传入的值。可以设计一个接口，子窗体都继承这个接口。第一步中创建集合的类型为这个接口类型。

3. 设计一个中间类，在加载时创建父窗体与子窗体。把子窗体加入父窗体中的接口集合中。

### 创建三个窗体



当master加载时创建Child与Parent

|  |
| --- |
| private void master\_Load(object sender, EventArgs e)  {  Child cd = new Child();  Parent p = new Parent();  cd.Show();  p.Show();  } |

### 创建接口

|  |
| --- |
| public interface ISetText  {  void SetText(string str);  } |

### 子窗口实现接口

|  |
| --- |
| public partial class Child : Form, ISetText  {  public Child()  {  InitializeComponent();  }  public void SetText(string str)  {  this.lbl\_GetMsg.Text = str;  }  } |

### 父窗口创建集合并在点击按钮时变量集合调用方法

|  |
| --- |
| public List<ISetText> list;  private void button1\_Click(object sender, EventArgs e)  {  if (list == null)  {  return;  }  foreach (var item in list)  {  item.SetText(this.txtMsg.Text);  }  } |

### 在中间类master中创建父类与子类，并添加子类到集合

|  |
| --- |
| private void master\_Load(object sender, EventArgs e)  {  Child cd = new Child();  Parent p = new Parent();  p.list = new List<ISetText>();  p.list.Add(cd);  cd.Show();  p.Show();  } |

这样父类和子类就实现了低耦合。删除子类不会影响父类。