一点 Duilib 编程总结

borlittle

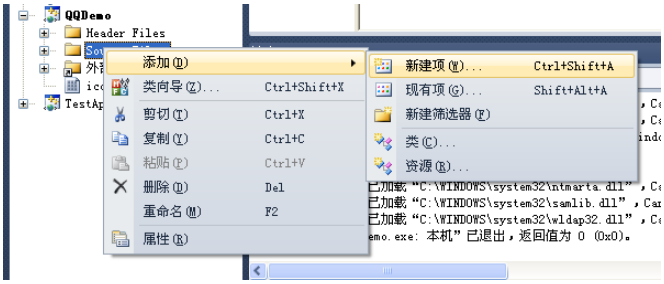
2011-9-1

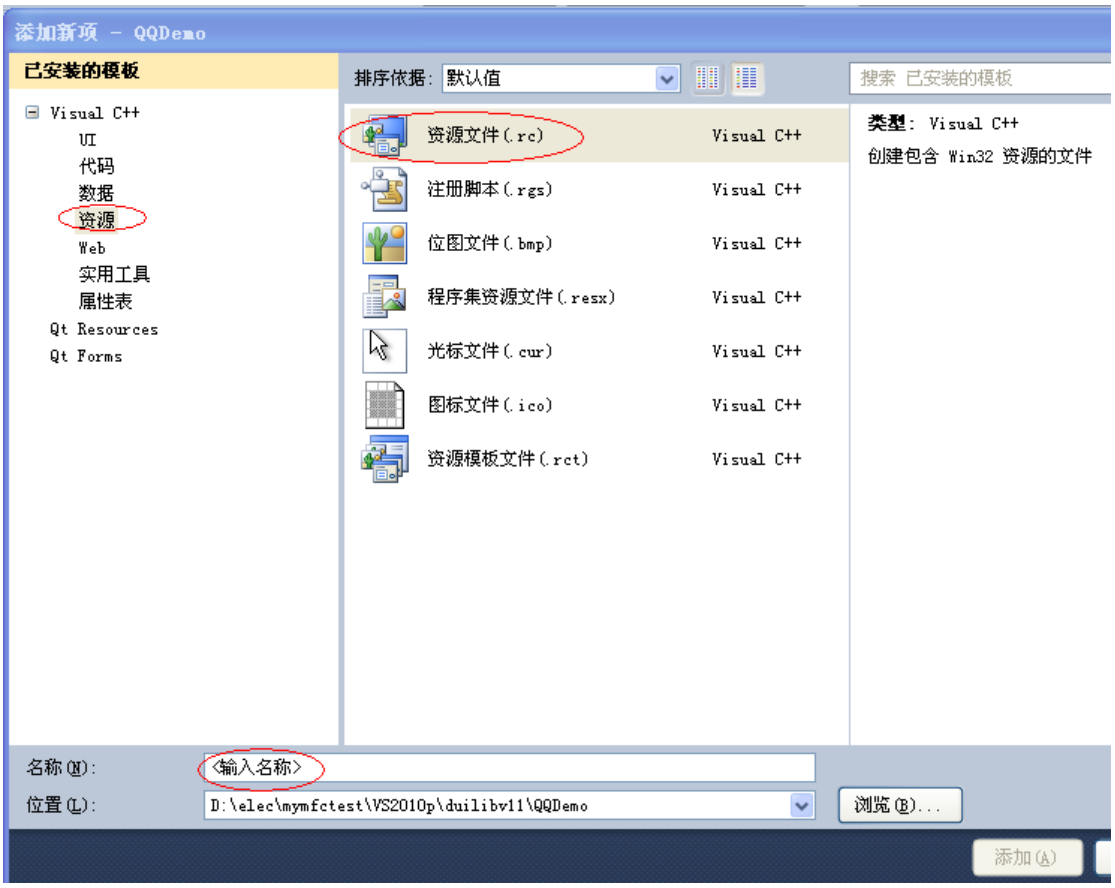
目录

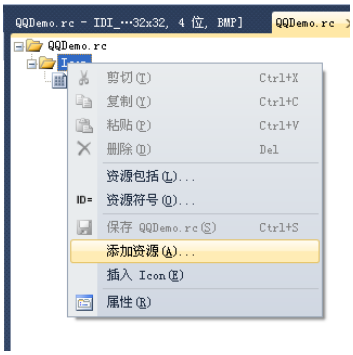
一直在寻找一个比较好的 UI 解决方案，最近一些时间看到了开源的 Duilib 库，觉得无论是其采用的 DirectUI 的设计思想还是实际的界面效果都让自己觉得非常喜欢，真有些相见恨晚的感觉。Duilib 团队做了很多的工作，极大的方便了将 Duilib 应用到程序的设计之中。之前在嵌入式 Linux 系统上，国内有开源的 MiniGUI 项目，而如今在 Windows 系统上也有了开源的 Duilib 项目，是国内不可多得的 UI 项目，希望能够坚持做下去。这里只是记录下一点自己的学习与大家分享，希望能够给其他像这样的初学者提供一些参考。

（1）更改程序默认的 WIN32 图标

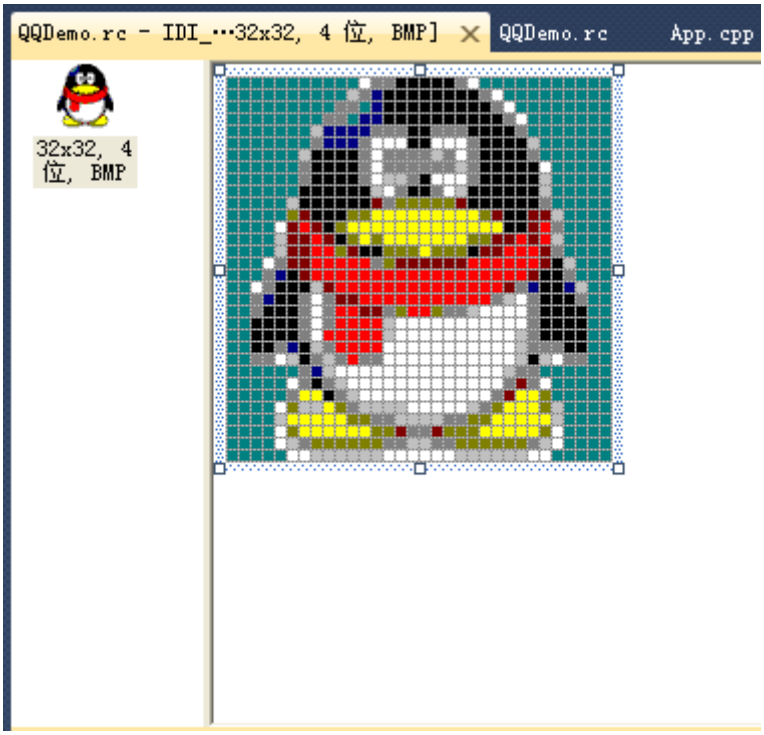
给程序添加资源，更改程序默认的 WIN32 图标




添加新项 - QQDemo
已安装的模板
Visual C++
UI
代码
数据
资源
Web
实用工具
属性表
Qt Resources
Qt Forms
排序依据: 默认值
搜索 已安装的模板
资源文件(.rc) Visual C++
注册脚本(.rgs) Visual C++
位图文件(.bmp) Visual C++
程序集资源文件(.resx) Visual C++
光标文件(.cur) Visual C++
图标文件(.ico) Visual C++
资源模板文件(.rct) Visual C++
类型: Visual C++
创建包含 Win32 资源的文件
名称(N): <输入名称>
位置(L): D:\elec\mymfctest\VS2010p\duilibv11\QQDemo
浏览(B)...
添加(A)


QQDemo.rc - IDI_...32x32, 4 位, BMP]
QQDemo.rc
QQDemo.rc
剪切(T) Ctrl+X
复制(Y) Ctrl+C
粘贴(P) Ctrl+V
删除(D) Del
资源包括(L)...
资源符号(Q)...
保存 QQDemo.rc(S) Ctrl+S
添加资源(A)...
插入 Icon(E)
属性(R)


添加资源
资源类型(T):
Accelerator
Bitmap
Cursor
Dialog
HTML
Icon
Menu
Ribbon
String Table
Toolbar
Version
新建(N)
导入(M)...
自定义(C)...
取消(A)
帮助(H)



程序编译之后就可以看到，生成的程序的图标已经从原来的 WIN32 默认图标改为了自定义的好看的图标了。

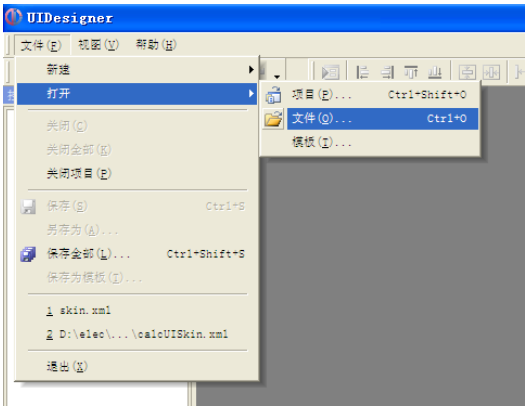
原来的 WIN32 程序默认图标

自定义的图标

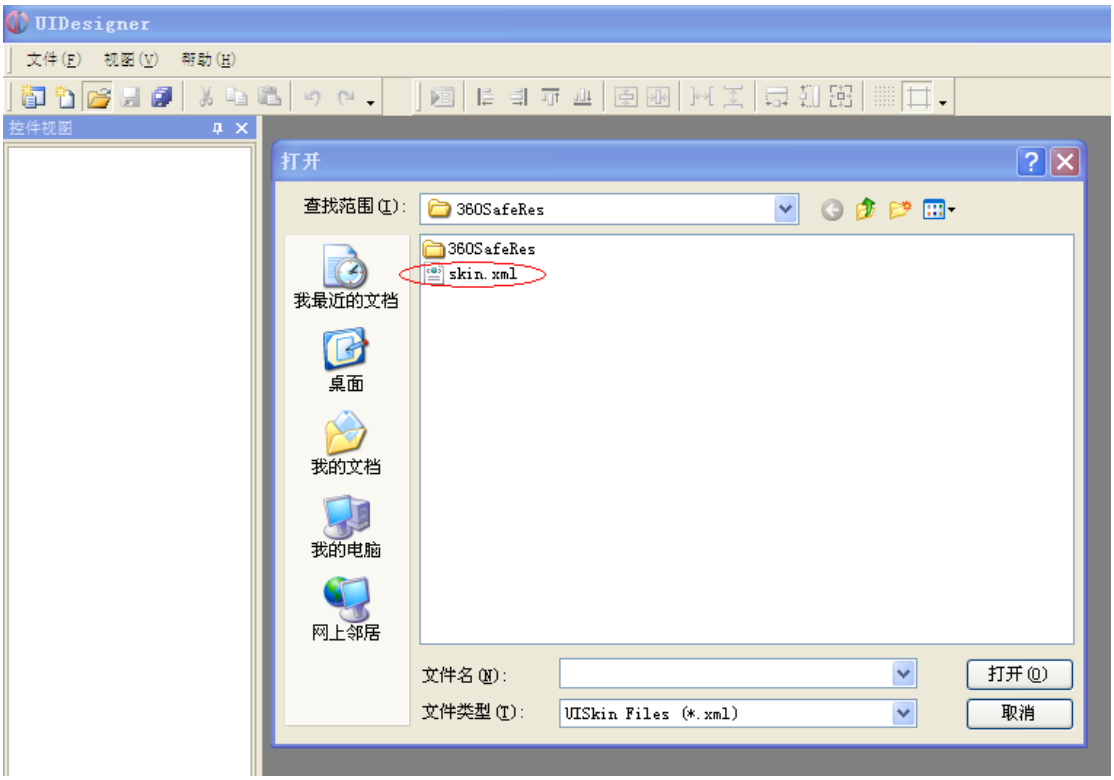
（2）使用 UIDesigner 设计皮肤

（1）首先可以用踏雪流云设计的 UIDesigner 打开一个已有的 DEMO 的皮肤文件 skin.xml（将 zip 皮肤解压到一个文件夹下），进行查看。这里以 360SafeRes 的皮肤文件作为观察：
示例程序运行后的界面如下图：

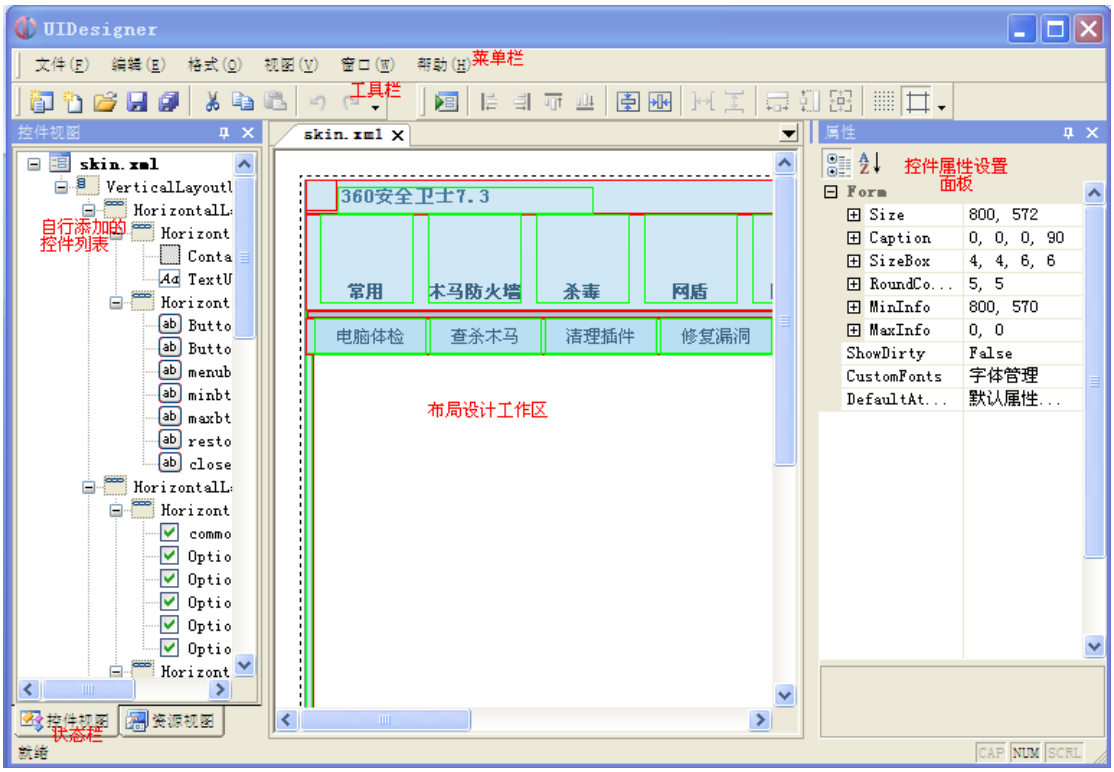
下面打开 UIDesigner：



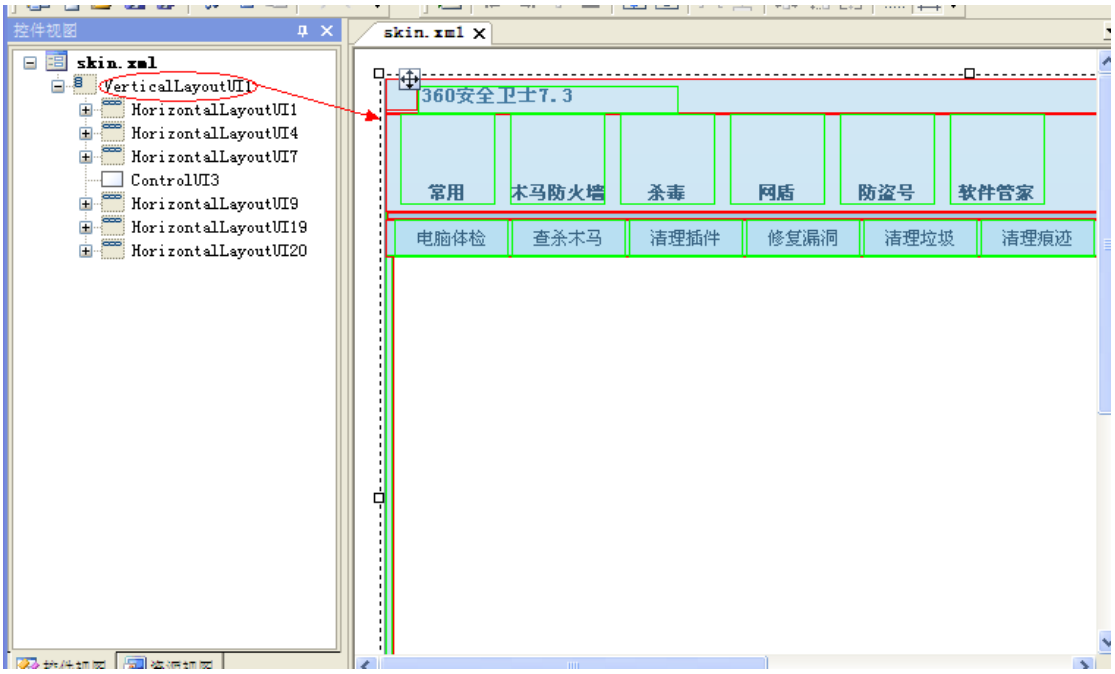
打开示例的皮肤文件：



打开后的界面如下：

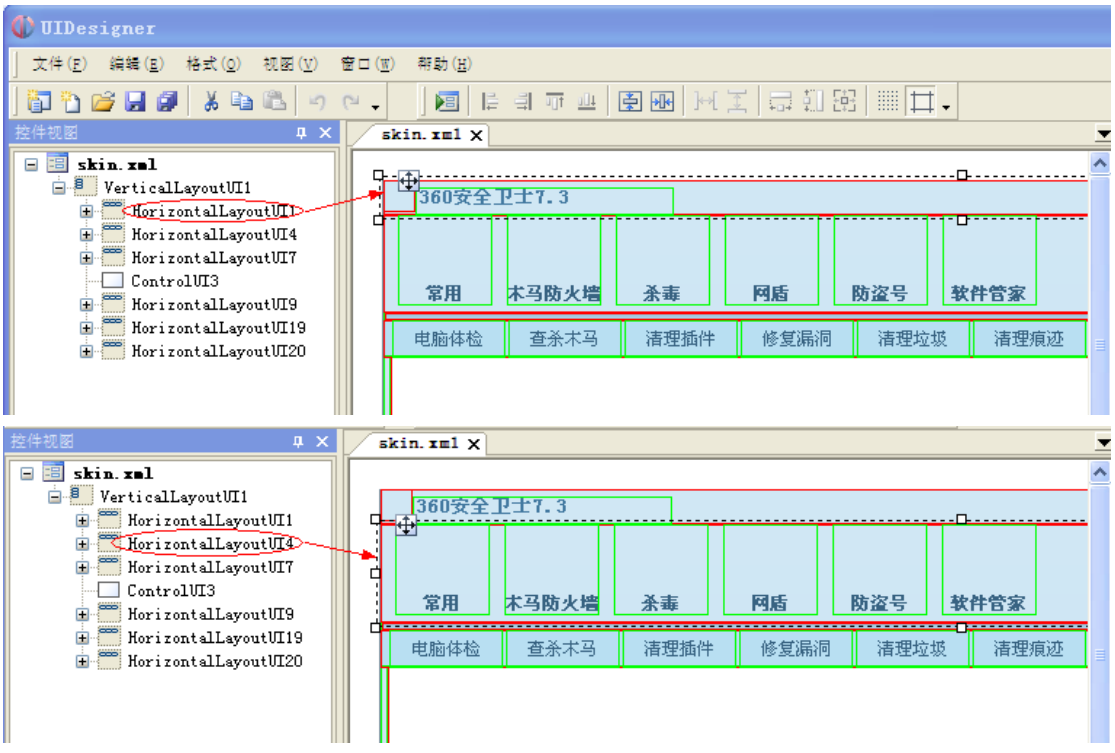


将左侧的控件视图的列表项收缩进行观察

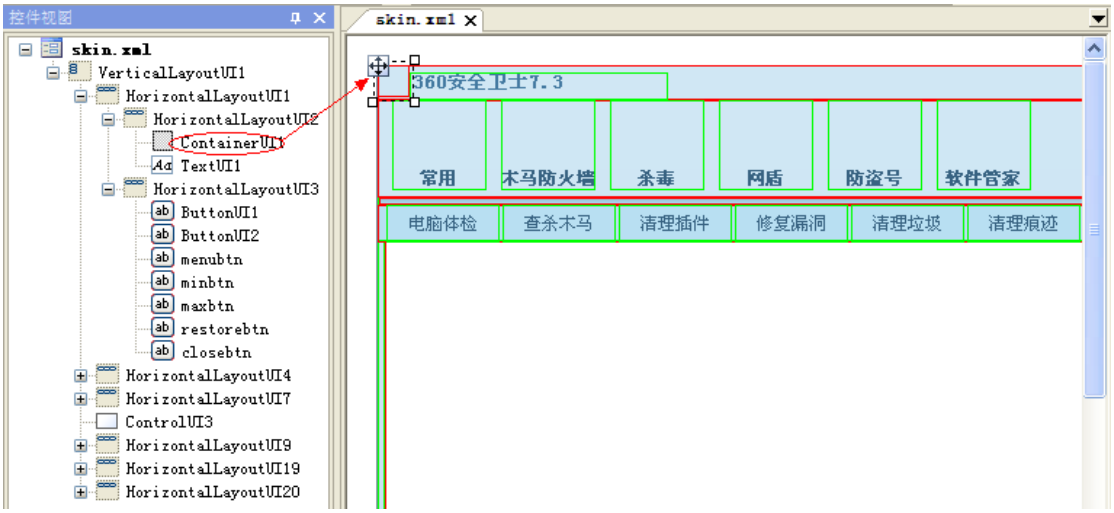


上图中，当鼠标单击根目录（为一个垂直布局控件）可以看到它对应图中工作区的最底层最大的那个虚线框，如红色箭头所指。

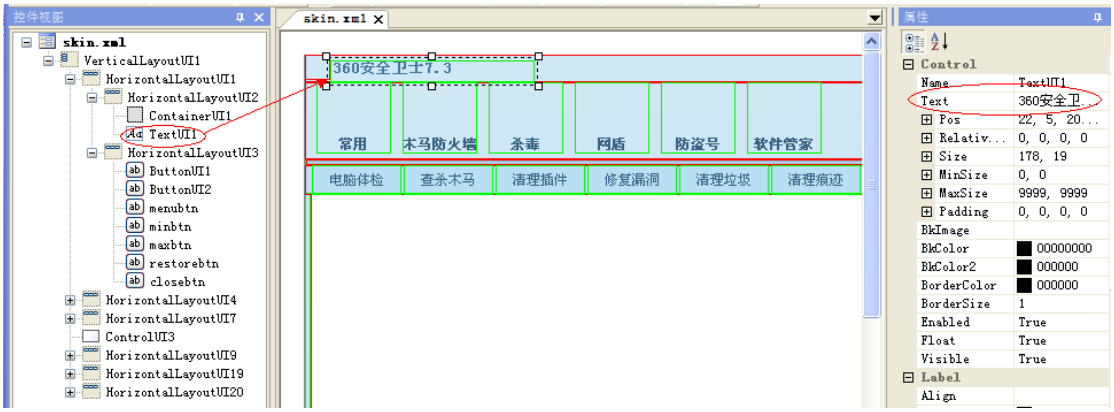
然后单击下面的子布局控件项也可看到对应的布局区（虚线框）



对应的各个布局控件包含的子控件也是类似的：

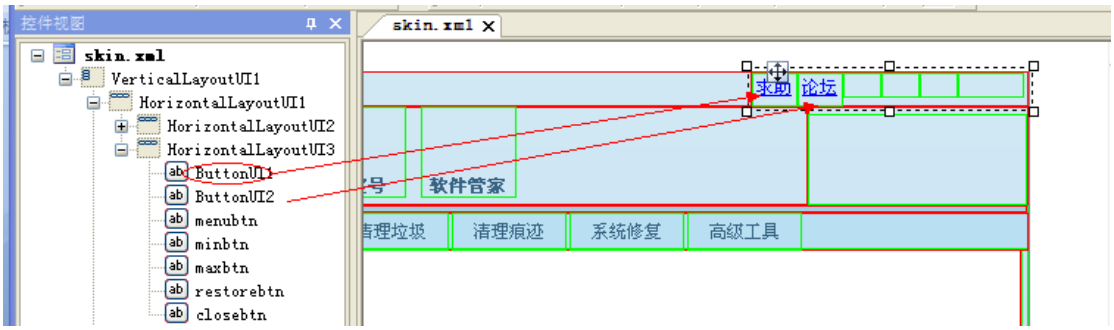


上图示为程序的图标。

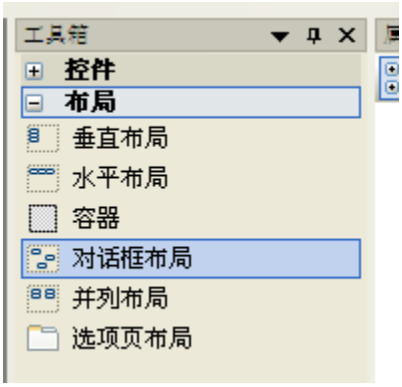


上图示为程序的标题。右侧的红圈处可以设置文字的内容，字体等属性。

其他控件也是类似的：



从工具箱中可以看到主要有控件和布局两大类工具:



从上面的分析就可以看到 UIDesigner 设计皮肤（程序界面）也和 VC 等 IDE 中设计是类似的，都是可视化，所见即所得的。而且做过网页的朋友应该会觉得这样设计程序的界面和用 Dreamweaver 等软件设计网页是非常像的，都是先设计好布局和层，然后再添加文字、图片、超链接等资源。

（2）经过上面的分析和观察，对于 UIDesigner 应该更为熟悉了，接下来就可以设计自己的程序的界面了。

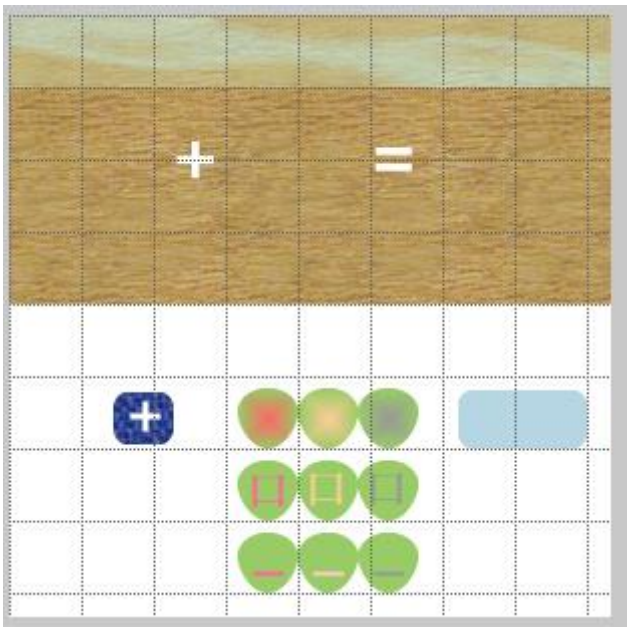
首先我们也应该像设计网页一样对自己的程序界面，形成自己的构思，最好画出草图和布局结构图。合理的划分布局和控件有利于简化程序的编写。在这一点上和设计网页等 UI 是相似的。

设计出草图之后我们接下来应该用图形图像设计软件，如 Photoshop 或者 Fireworks 等设计出界面的实际界面图像以及各个按钮控件等的多个状态的图像，然后利用这些图像设计软件带有的切片工具，将界面图像按各个控件部分切分出来。利用这些图像处理软件可以把我们切分好的图片直接全部导出到一个文件夹下，也可以选择导出对应的网页 html 或者 xml 文件，这样文件中会自动带有各个图片的大小和位置等信息，有利于我们在 UIDesigner 设置对应的参数。当然，如果以后 UIDesigner 能够直接带有图片编辑和切片功能那就会更方便了。

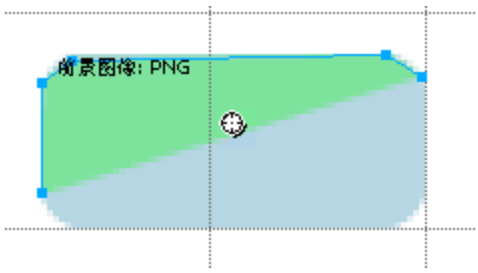
（3）简单的加法计算器实例

下面以一个小小的实例（一个简单的加法计算器）来进行演示：

首先利用 Fireworks 设计出效果图：

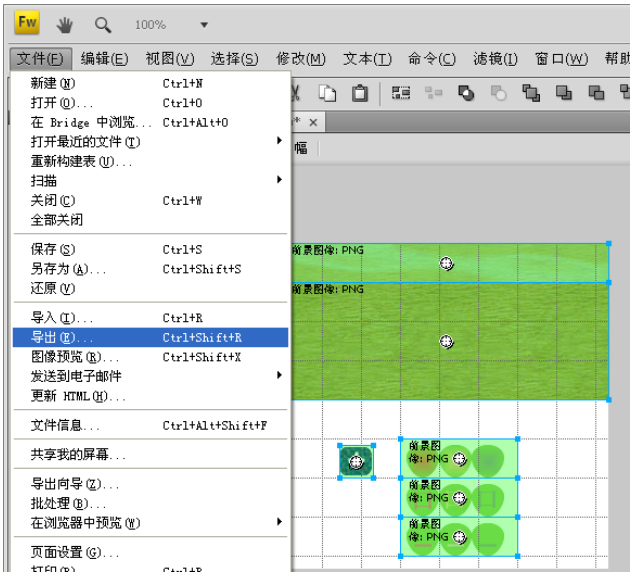


为了简单计，只设计程序图标，程序对话框底纹图像和几个按钮。

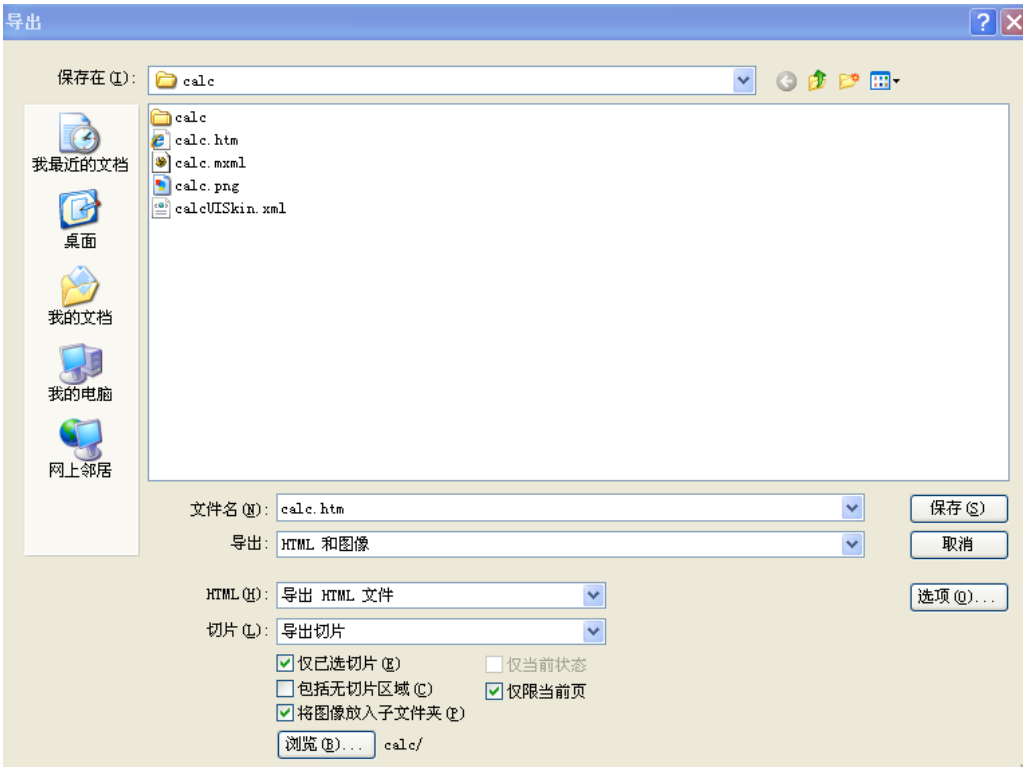


用多边形切片工具切出多边形按钮图形。

按下 shift 键、鼠标单击，选中要导出的切片，执行导出命令：



选择路径和输出选项：



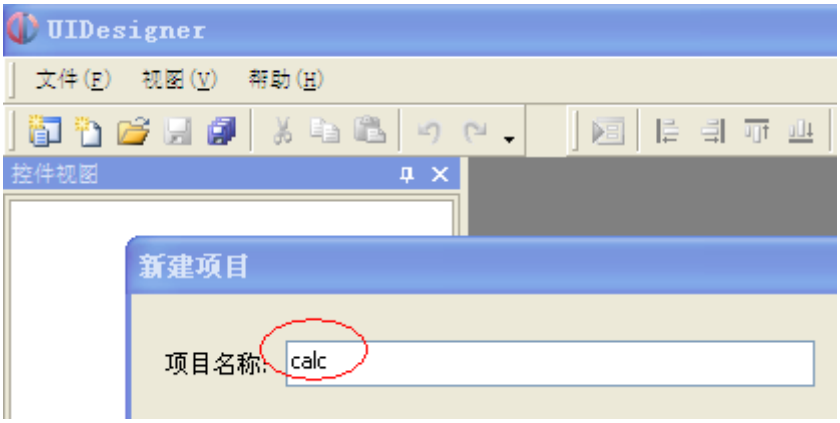
则在文件夹下出现：

如果之前就把各个按钮放到相应位置之后再切片导出，则打开导出的 html 文件可得到其相关的位置大小信息：

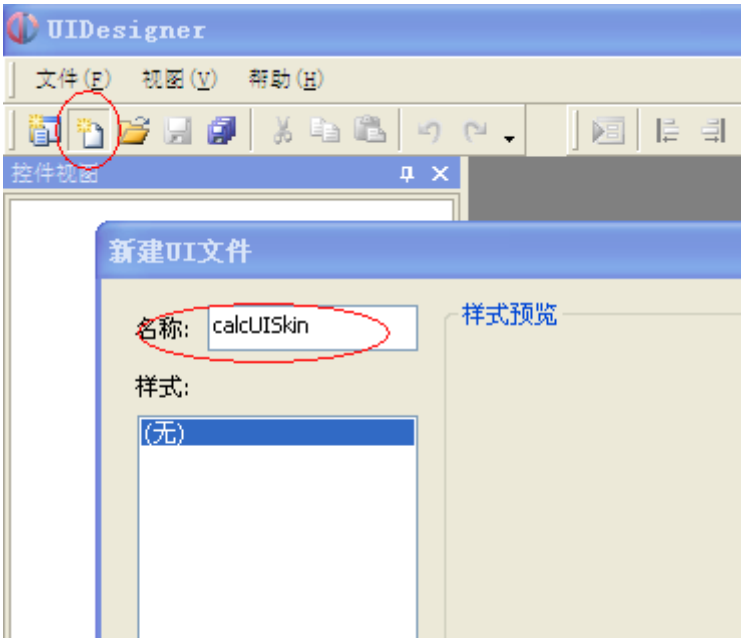
```
<tr>
 <td colspan="9"><img name="calc_r1_c1" src="calc/calc_r1_c1.png" width="300" height="36" border="0"
 <td><img src="calc/spacer.gif" width="1" height="36" border="0" alt="" /></td>
</tr>
<tr>
 <td colspan="9"><img name="calc_r2_c1" src="calc/calc_r2_c1.png" width="300" height="108" border="0"
 <td><img src="calc/spacer.gif" width="1" height="108" border="0" alt="" /></td>
</tr>
<tr>
 <td colspan="9" valign="top" bgcolor="#ffffff"><p style="margin:0px"></p></td>
 <td><img src="calc/spacer.gif" width="1" height="42" border="0" alt="" /></td>
</tr>
<tr>
 <td rowspan="2" colspan="3" valign="top" bgcolor="#ffffff"><p style="margin:0px"></p></td>
 <td rowspan="4" colspan="2"><img name="calc_r4_c4" src="calc/calc_r4_c4.png" width="90" height="30"
 <td colspan="4" valign="top" bgcolor="#ffffff"><p style="margin:0px"></p></td>
 <td><img src="calc/spacer.gif" width="1" height="1" border="0" alt="" /></td>
</tr>
<tr>
 <td rowspan="8" valign="top" bgcolor="#ffffff"><p style="margin:0px"></p></td>
 <td rowspan="4" colspan="2"><img name="calc_r5_c7" src="calc/calc_r5_c7.png" width="65" height="30"
 <td rowspan="8" valign="top" bgcolor="#ffffff"><p style="margin:0px"></p></td>
 <td><img src="calc/spacer.gif" width="1" height="1" border="0" alt="" /></td>
</tr>
<tr>
```

做好了界面的图片素材，接下来就可以在 UIDesigner 中构建程序的皮肤了。

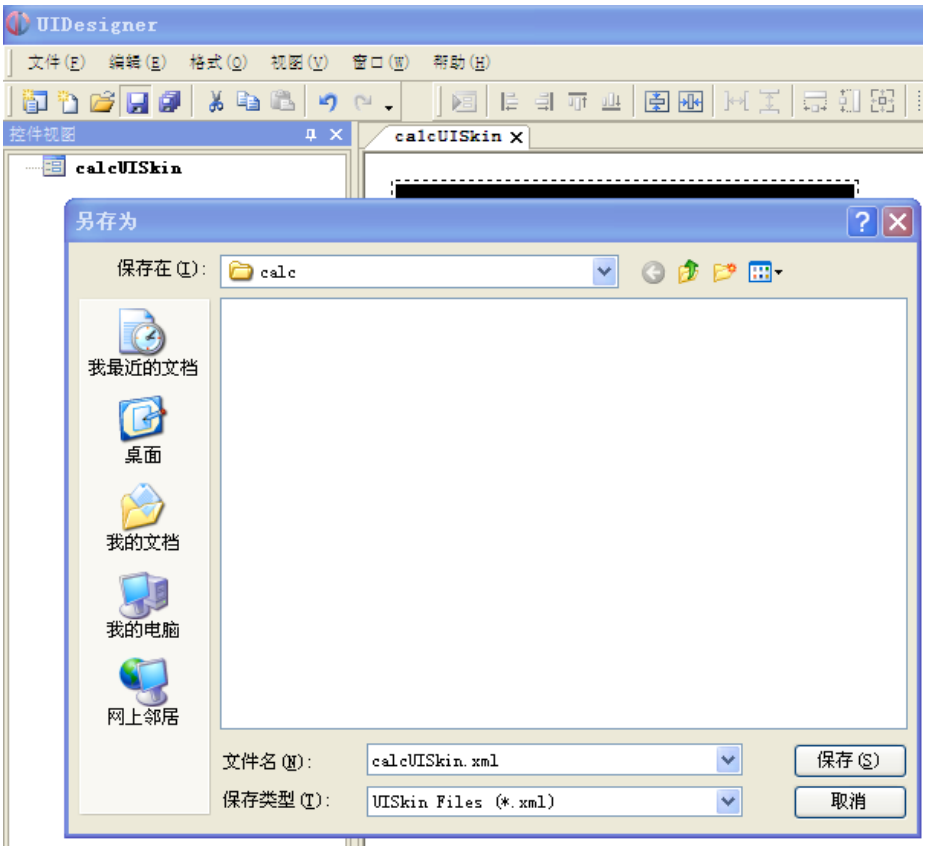
新建一个皮肤工程



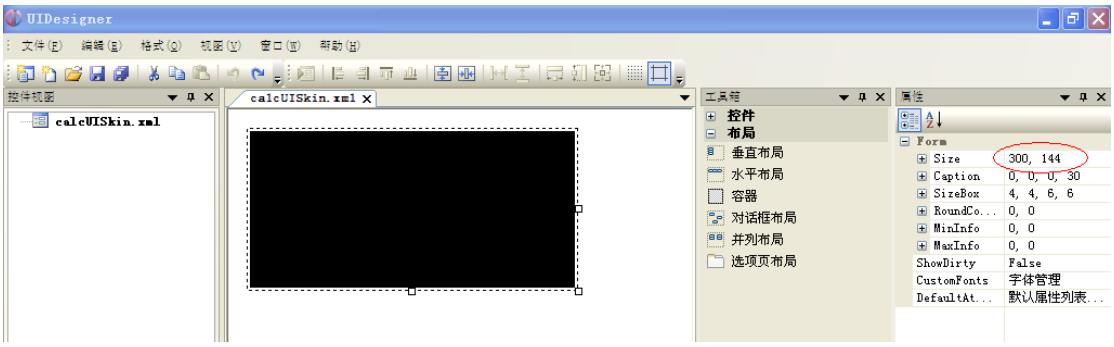
新建一个皮肤 xml 文件



保存 xml 文件

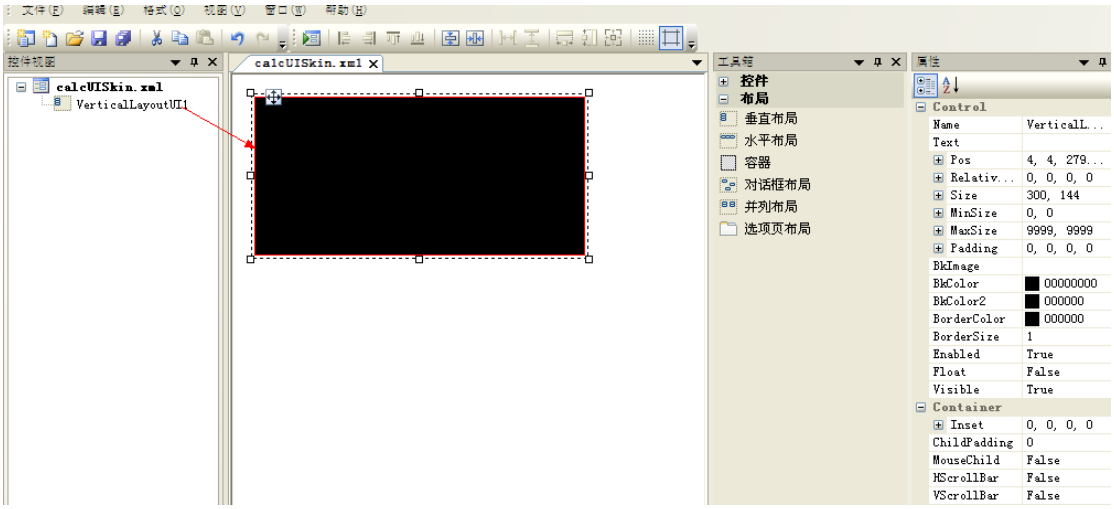


接下来将进入工作界面：

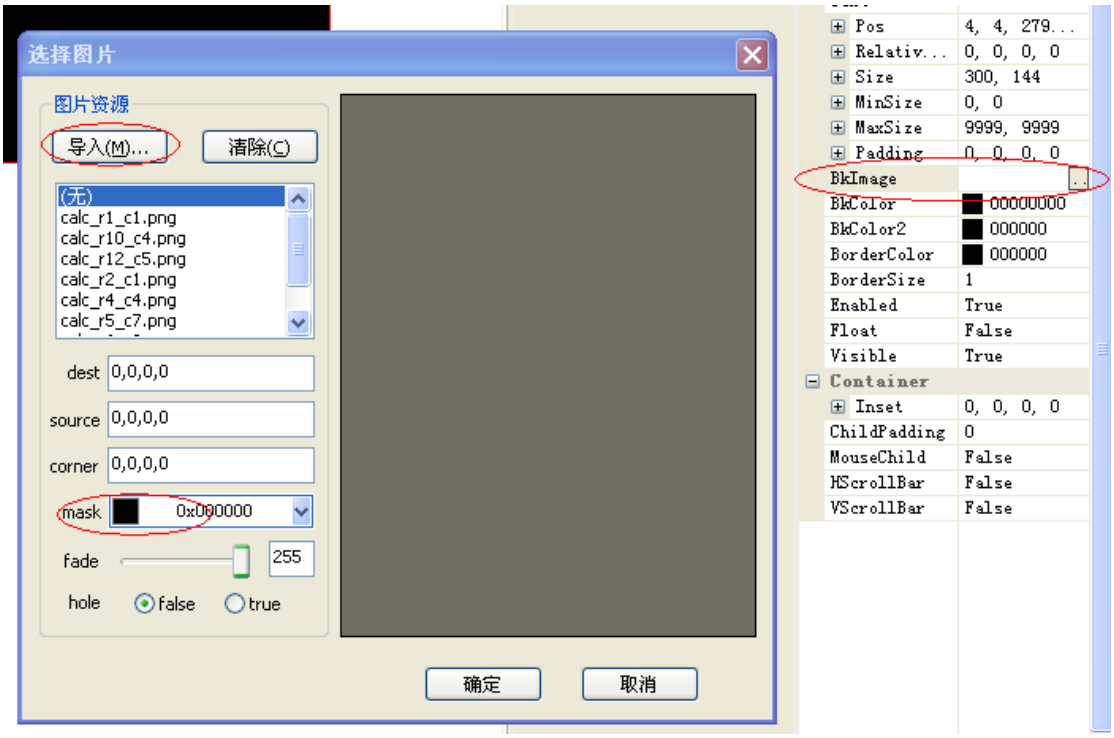


首先将布局区的黑色矩形调整大小为我们的对话框大小。

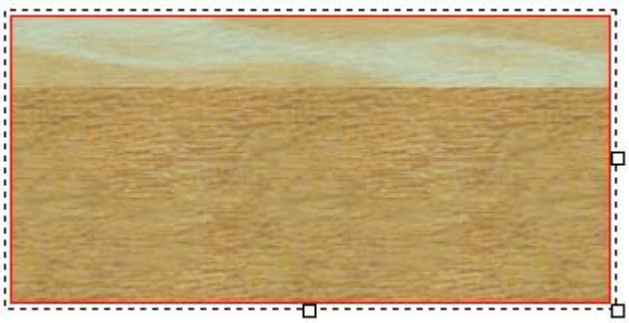
然后从工具栏中选中“布局”中的垂直布局，到工作区窗口按住鼠标右键拖拽一下，则将出现红色边框的一个矩形，同时左侧的控件资源视图中会自动为其命名



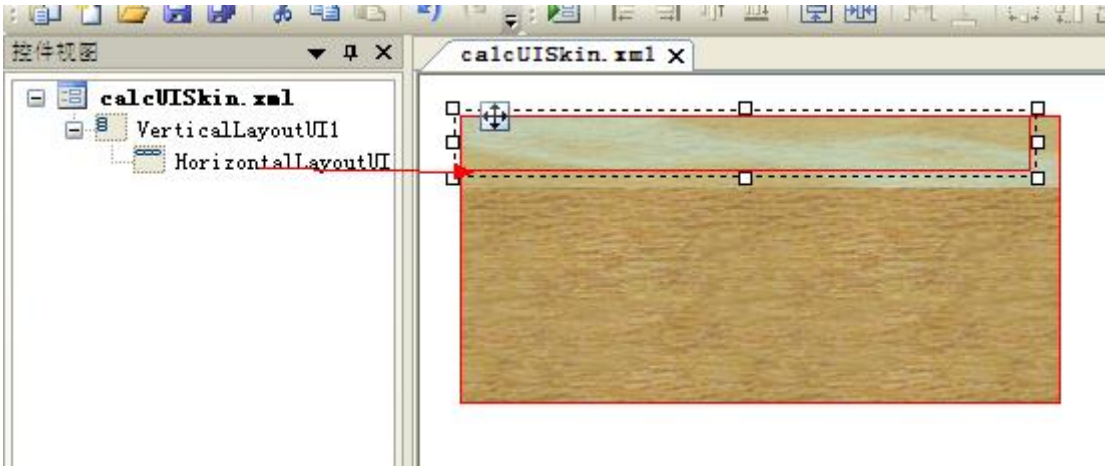
接下来要调整这个布局控件的属性，这些属性设置之后将由 Duilib 为其提供对应的功能、特性支持，因而需要仔细设置。这里设置其大小和窗口大小一样，并为其导入背景图片：



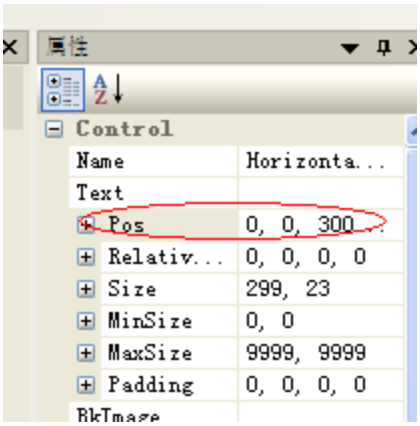
在选择图片对话框中点击“导入”按钮，可以打开文件浏览对话框，通过缩略图可以方便选择图。如果设置上图中的“mask”的颜色并将“hole”选为“true”，我们可以实现透明和异形窗口（即背景图片中与之颜色一样的将会被除去，就像镂空雕刻一样。



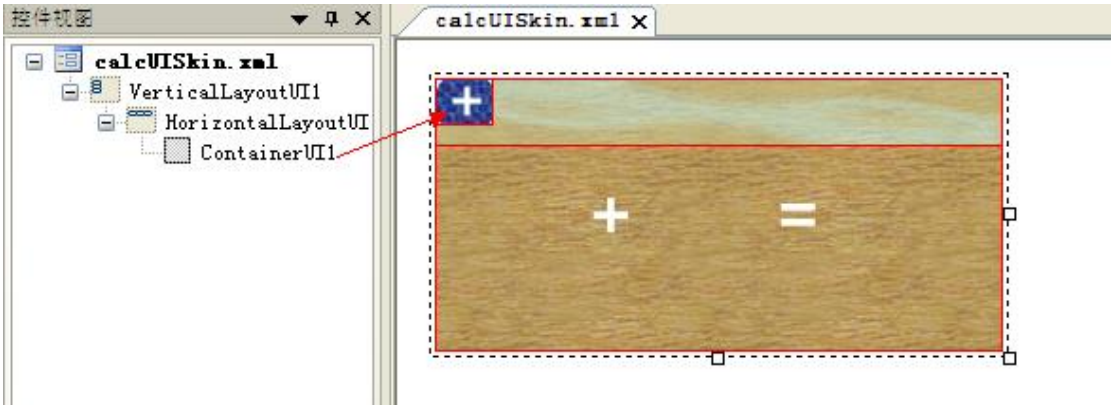
然后我们继续添加一个水平布局放在顶端，作为 calc 的标题栏。



在垂直布局框里边拖动出一个长条矩形，调节其大小和位置，精细的调节可以用右侧的属性栏。



再添加一个水平布局用于显示图标和程序名称，在其中添加一个容器用于显示程序的图标：



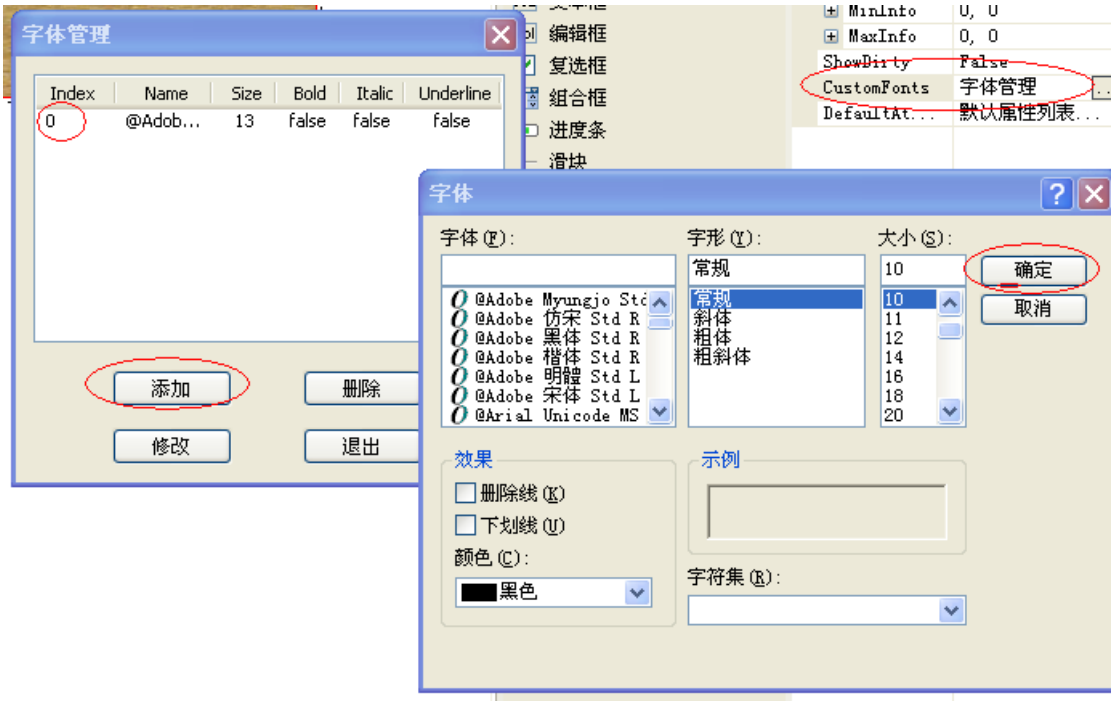
添加一个文字控件，用于显示程序名字：

设置对应的属性值：

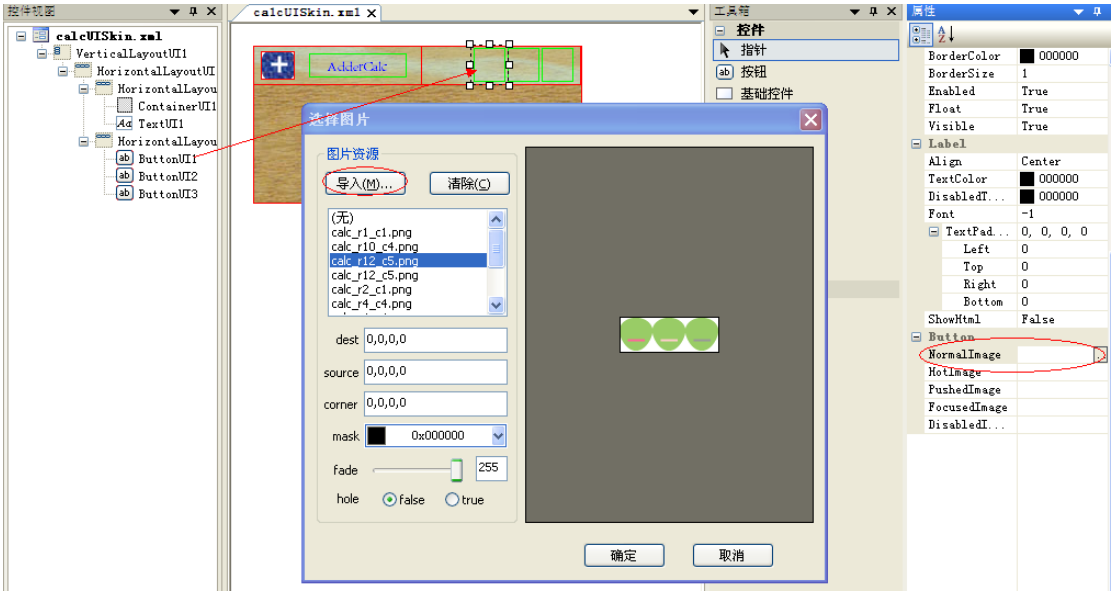
注：相关的属性值的含义可以参考 Duilib 团队提供的文档“属性列表.xml”，其中均给出了设置示例，如下图：

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 可能有错漏，欢迎补充。 wangchyz(wangchyz@gmail.com) -->
<Controls>
  <Window parent="">
    <Attribute name="size" default="0,0" type="SIZE" comment="窗口的初始化大小,如(800,600)"/>
    <Attribute name="sizebox" default="0,0,0,0" type="RECT" comment="窗口可拖动改变窗口大小的边距,如(4,4,6
    <Attribute name="caption" default="0,0,0,0" type="RECT" comment="窗口可拖动的标题栏大小的边距,最后一个
    <Attribute name="roundcorner" default="0,0" type="SIZE" comment="窗口圆角大小,如(4,4)"/>
    <Attribute name="mininfo" default="0,0" type="SIZE" comment="窗口最小大小,如(320,240)"/>
    <Attribute name="maxinfo" default="0,0" type="SIZE" comment="窗口最大大小,如(1600,1200)"/>
    <Attribute name="alpha" default="255" type="BYTE" comment="窗口的alpha值(0-255),如(100)"/>
```

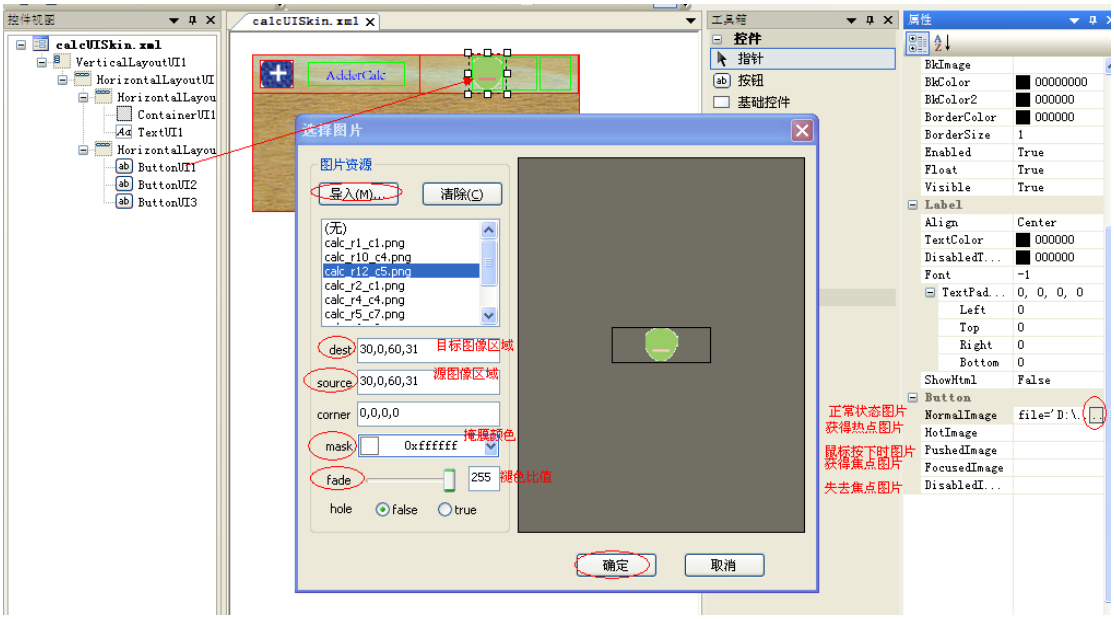
如果字体不怎么好看，可以自行添加字体：



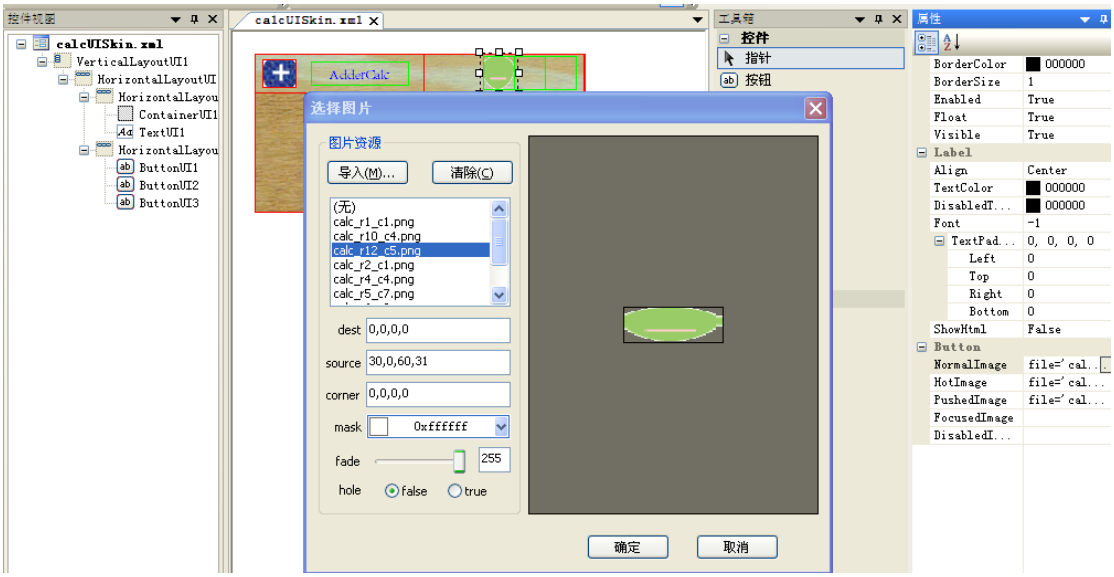
再添加一个水平布局，并在其中添加三个按钮用于显示和控制窗口状态：



为每个按钮设置其不同状态时的图片，即从一整张图片中裁出的一部分。继续为其添加其余状态的图片：

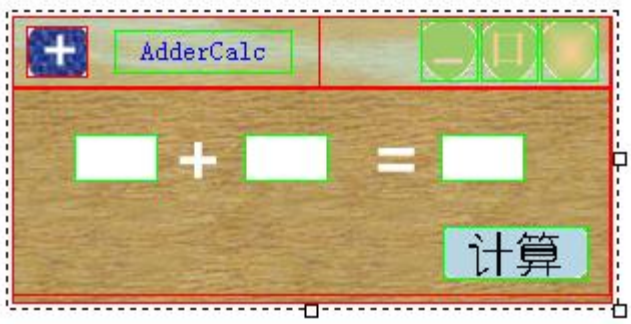


设置之后如下：

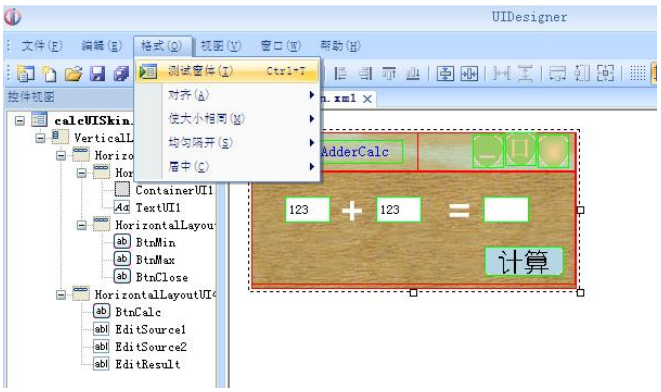


由于只绘制了三种状态的颜色，因而分配为正常状态、获得热点状态、鼠标按下状态。

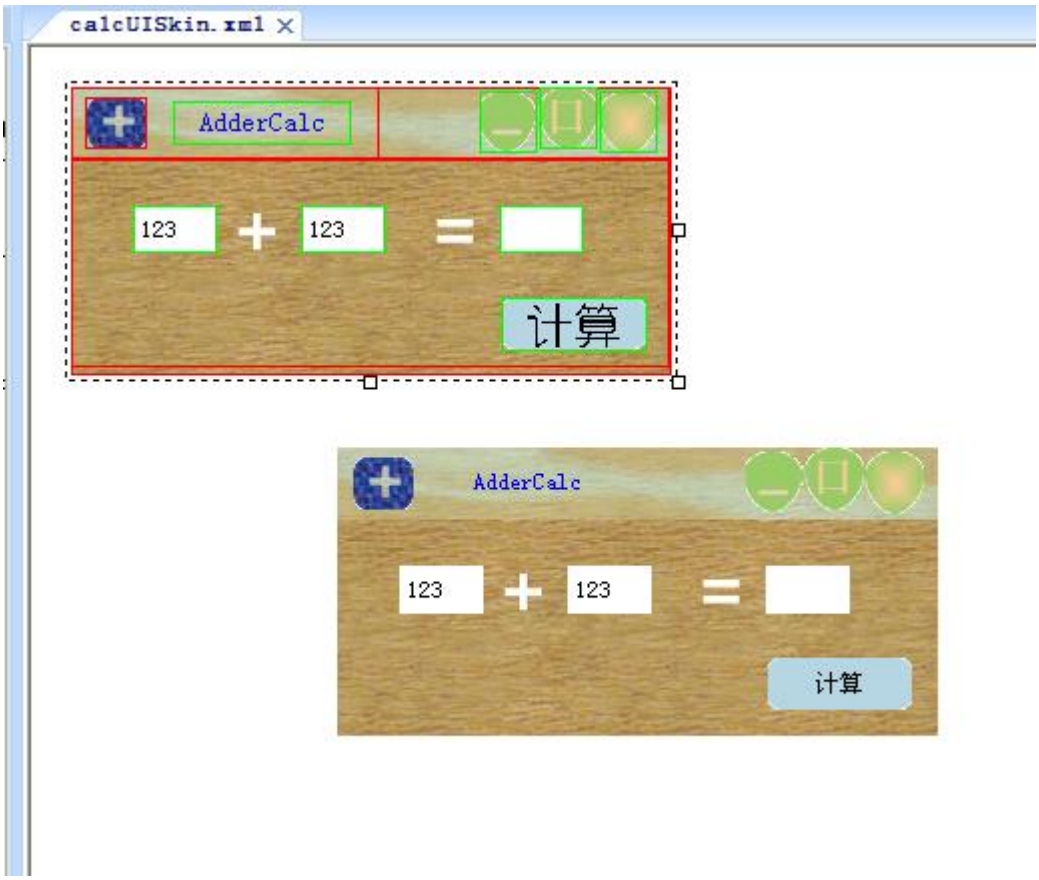
接下来再添加一个水平布局，并在其中添加一个按钮，三个编辑框。



设计完毕我们可以测试一下实际运行时的界面（这个功能非常的方便，设计得很好）选择“格式”菜单下的“测试窗体”命令。



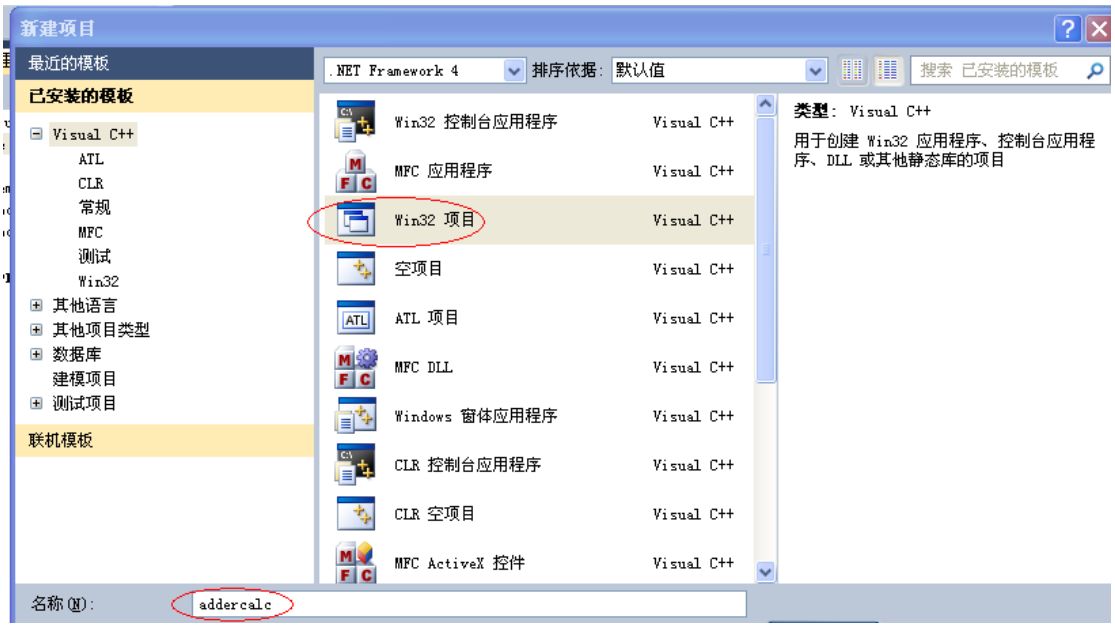
可以看到运行时的界面如下图中第二排那个窗口：



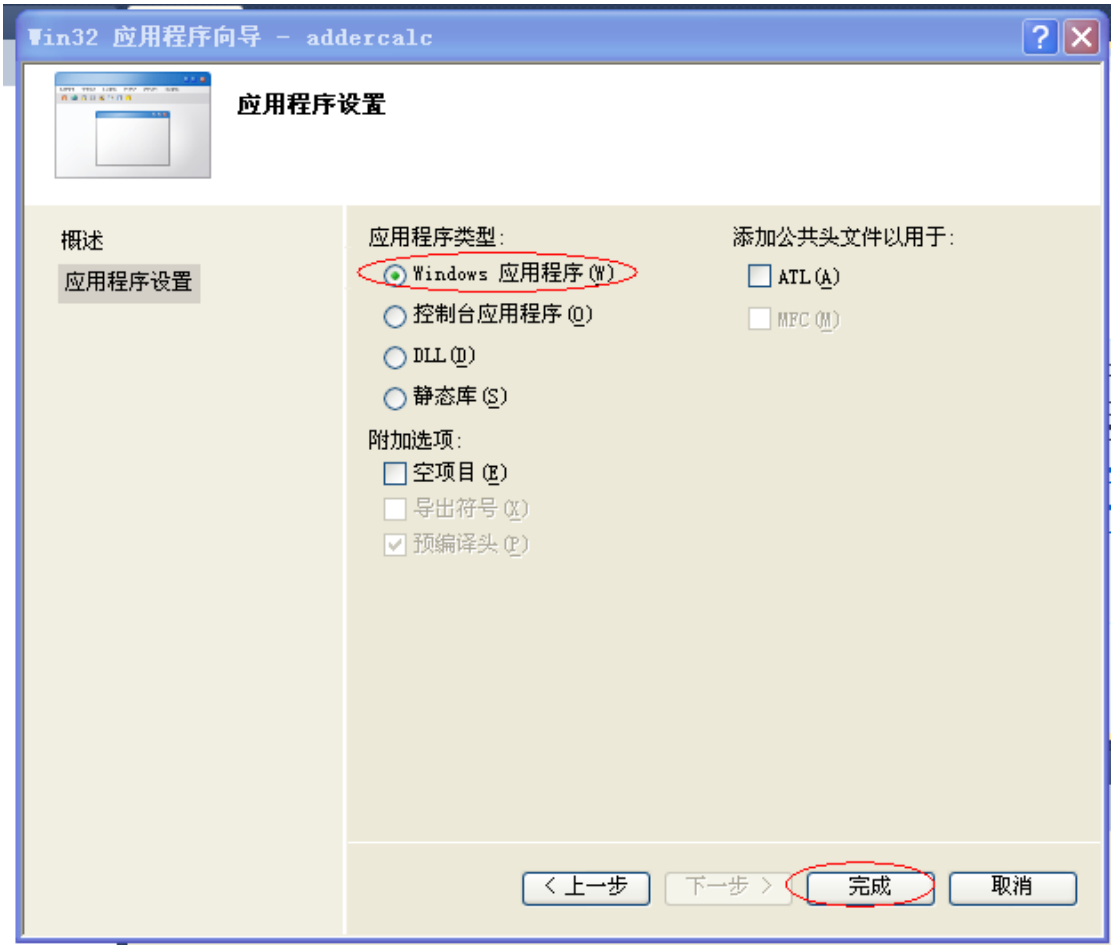
鼠标选中测试窗口，按下“ESC”键可以退出测试状态，可以继续修改窗口的界面。

设计完程序的界面，接下来就编写程序，实现计算器的计算功能，这里以 VS2010 平台为例，在其他平台操作应该是类似的。

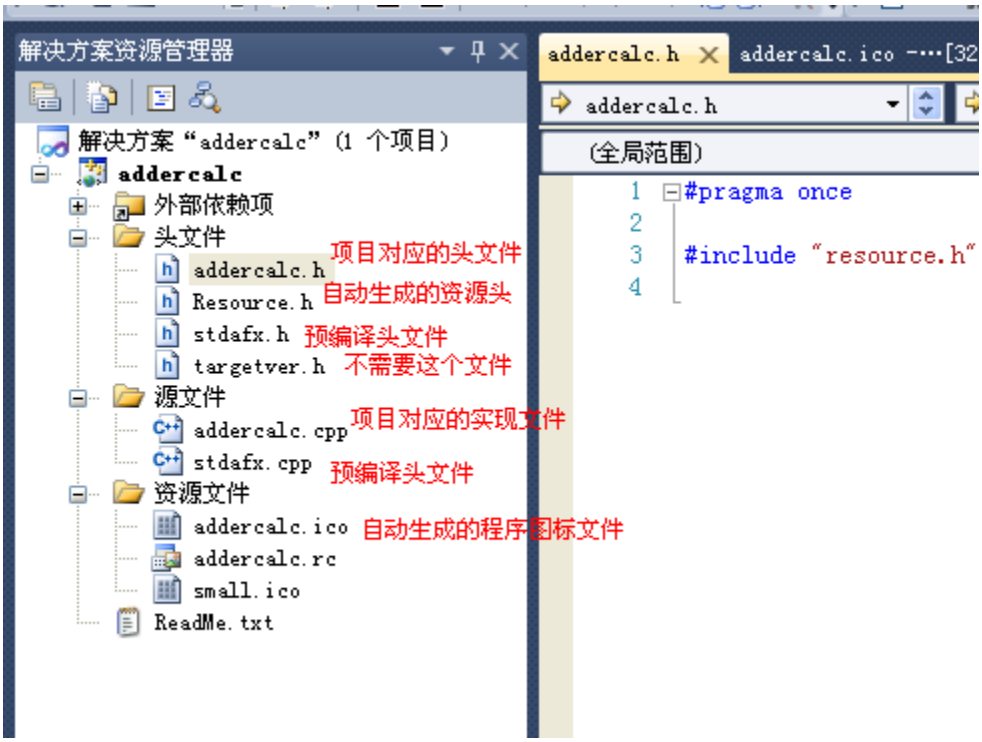
打开 VS2010，新建一个解决方案和一个 WIN32 项目：



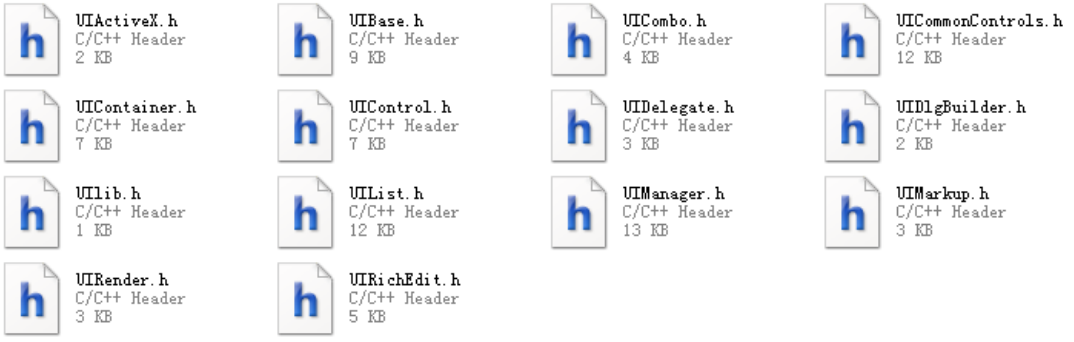
输入项目名称和路径：



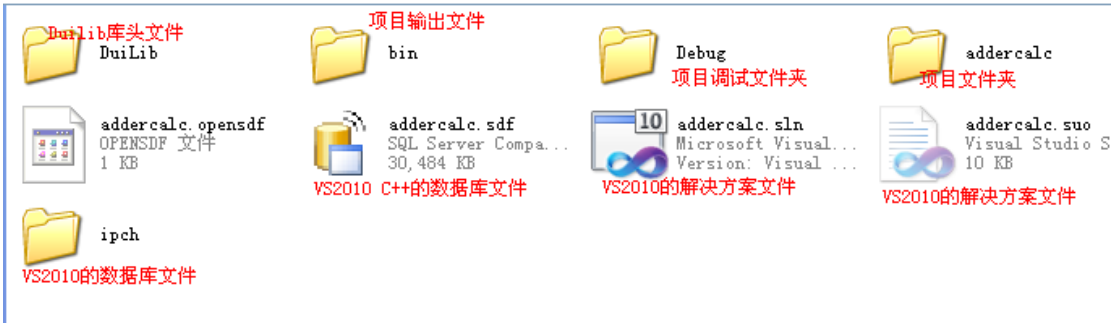
IDE 会默认生成若干个文件，如下图：



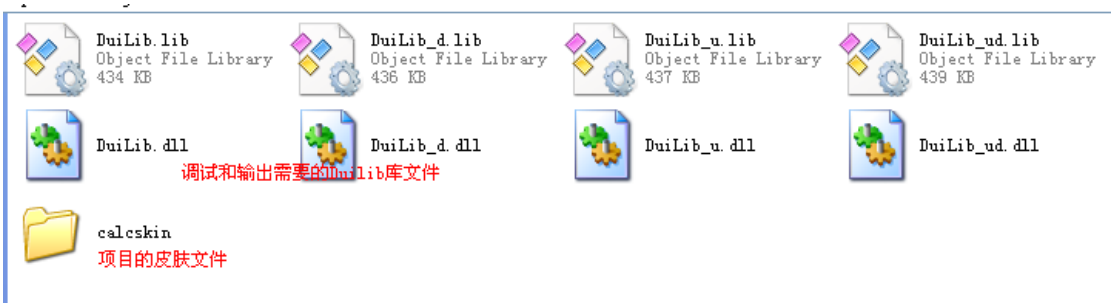
接下来需要把 Duilib 库引入引入到项目中，首先要 Duilib 库源码编译并生成动态链接库 dll 文件和对应的 lib 库文件用于实现隐式调用 Duilib 库中的函数。将 Duilib 库导出的四个版本的文件都复制到工程目录下的 bin 文件夹之中，再将 Duilib 库中的所有头文件拷到工程目录下。



项目文件夹下结构图如下：



编译前 bin 下的文件：

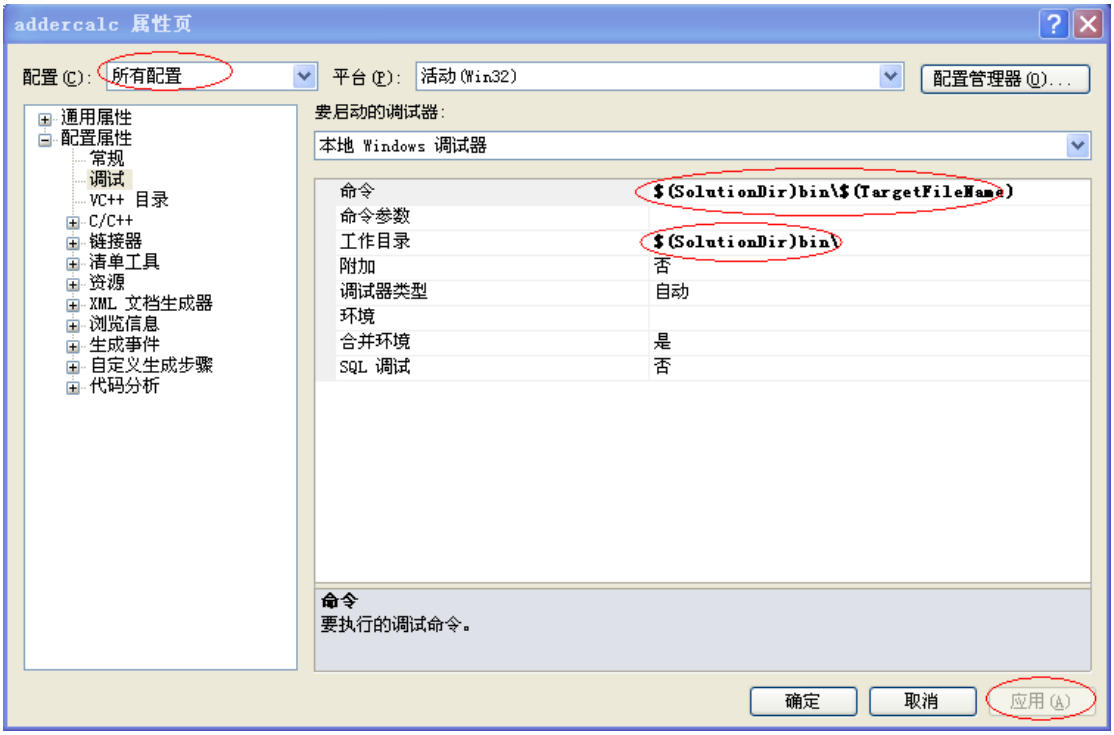


修改预编译头文件 StdAfx.h 添加对库文件的引用：

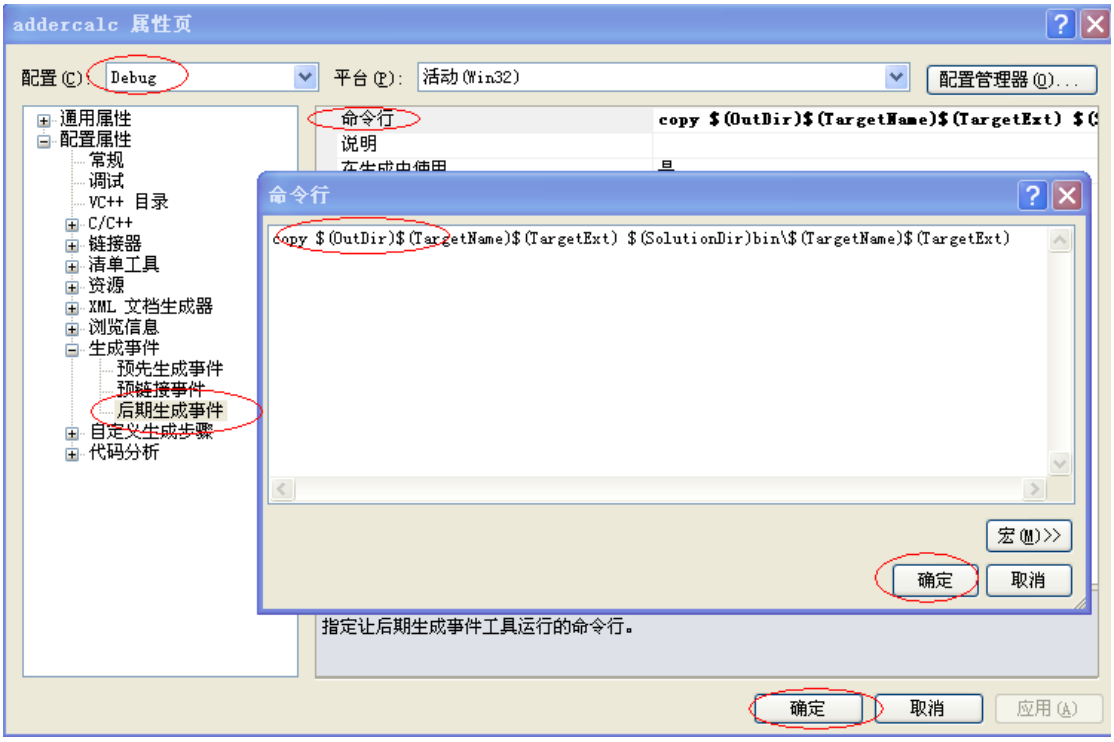
```
#include "..\Duilib\UIlib.h"

using namespace Duilib;

#ifdef _DEBUG
#    ifdef _UNICODE
#        pragma comment(lib, "..\\bin\\Duilib_ud.lib")
#    else
#        pragma comment(lib, "..\\bin\\Duilib_d.lib")
#    endif
#else
#    ifdef _UNICODE
#        pragma comment(lib, "..\\bin\\Duilib_u.lib")
#    else
#        pragma comment(lib, "..\\bin\\Duilib.lib")
#    endif
#endif
```

设置项目的属性：

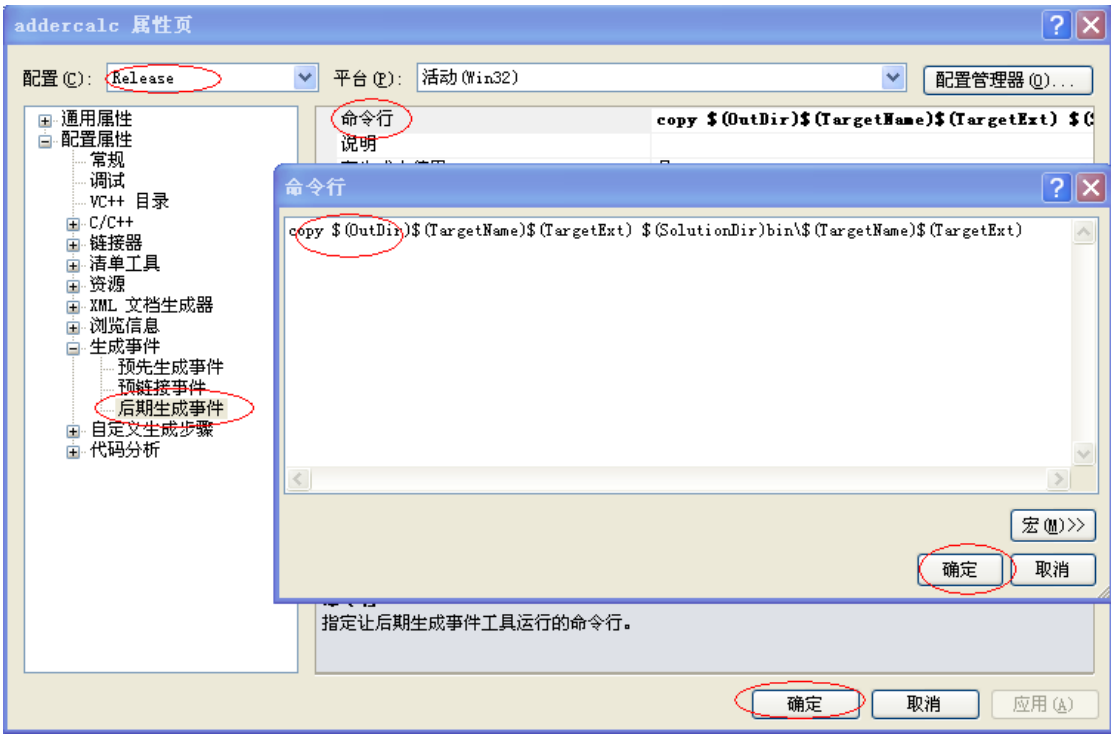
设置调试目录



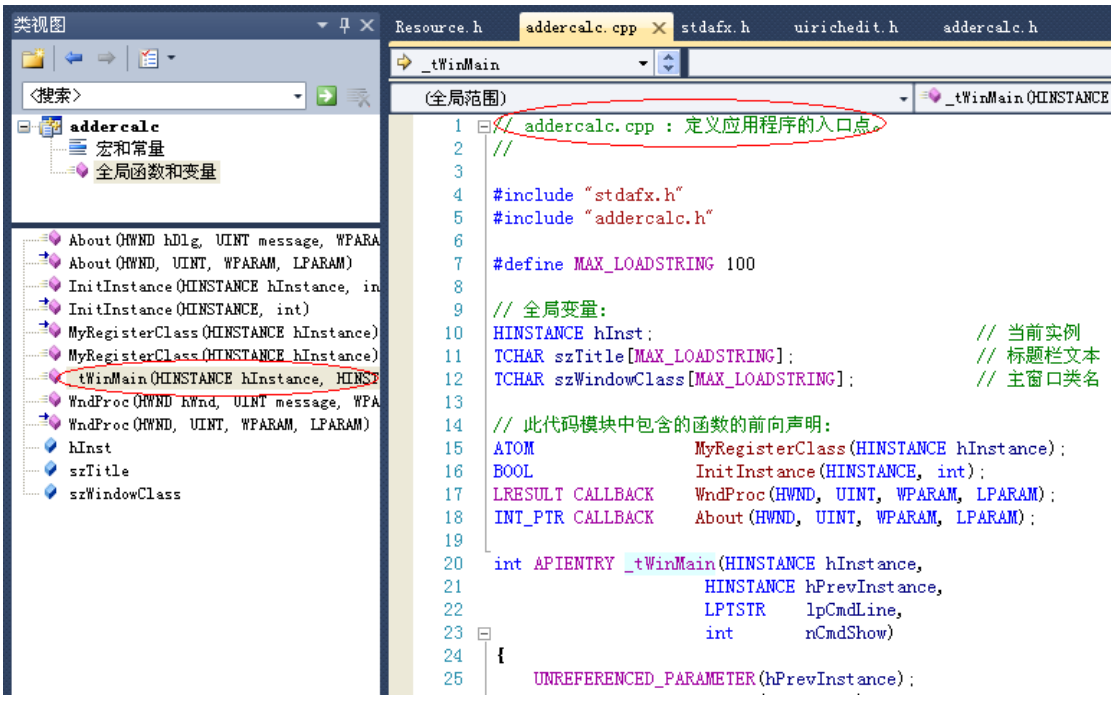
再设置调试版本属性



设置发行版本属性

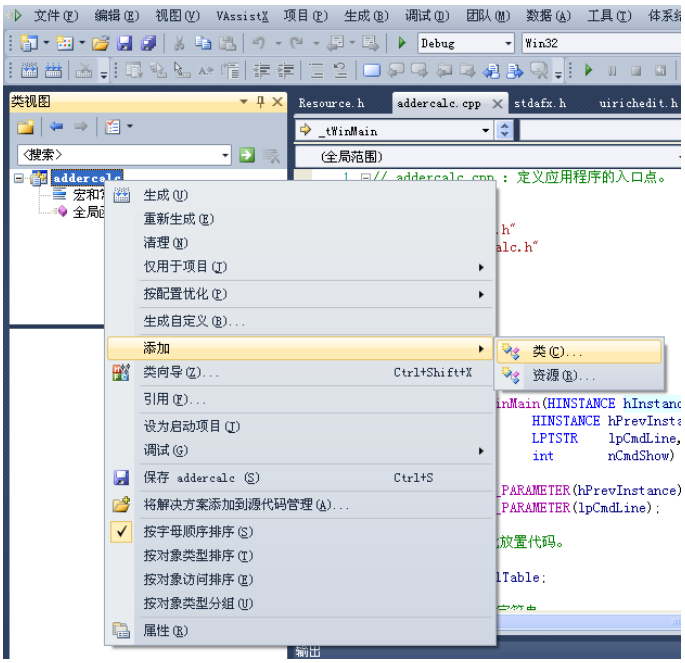


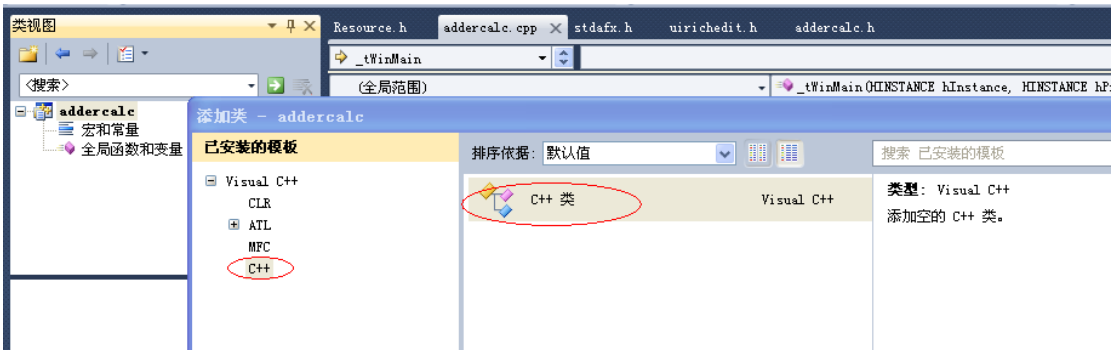
接下来修改工程实现对应的 cpp 文件。IDE 自动生成的 Win32 程序实现的 cpp 文件中有如下一些函数：



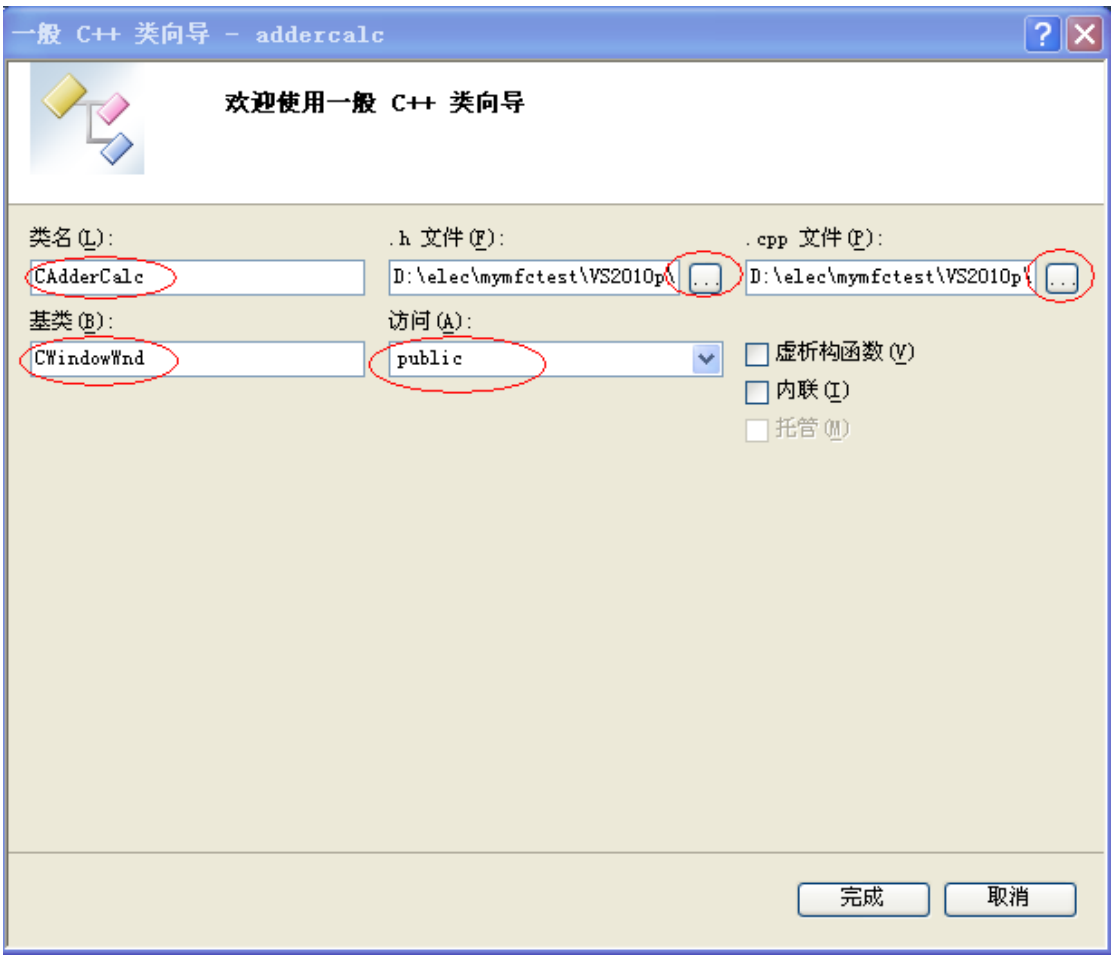
除了红色圆圈的主函数保留修改外，其他的都要删除，即接下来需要我们自己调用 Duilib 库的函数和类来写出程序的构架。

添加我们自己的程序框架类：





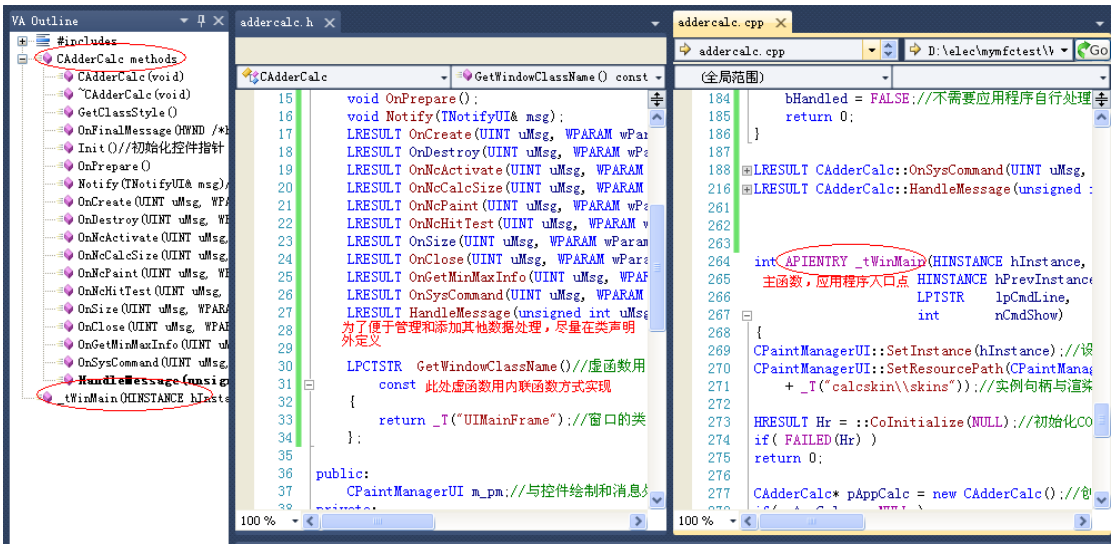
双击右侧红圈中的列表，进入添加页面：



再在文件中加上通知类：

```
#pragma once

#include "resource.h"
#include "stdafx.h"

class CAdderCalc :
    public CWindowWnd, public INotifyUI//应用程序窗口类
{

public:
    CAdderCalc(void);
    ~CAdderCalc(void);
};

```

接下来向类中添加相关的函数以处理消息（该部分可以参考《DUILIB 帮助文档》中程序和说明）程序大体框架内容主要借鉴 Duilib 团队给出的几个 Demo 程序，大家可以参考一下，可以学到其他控件的应用和更多的功能范例。



首先为了实现输入数据并且能够控制程序中的按钮操作，我们要先声明几个控件对象指针变量：

```
//定义按钮控件指针
    CButtonUI* m_pBtnClose;//关闭
    CButtonUI* m_pBtnMax;//最大化
    CButtonUI* m_pBtnMin;//最小化
    CButtonUI* m_pBtnCalc;//计算按钮
    CEditUI* m_pEditSource1;//加数
    CEditUI* m_pEditSource2;//被加数
    CEditUI* m_pEditResult;//加数和
```

下面重点给出几个成员函数：

1.Init()函数

我们需要在这个函数里边初始化用户与程序交互需要使用到的控件的指针（也就是在程序类中声明的那些控件对象指针）比如按钮，编辑框等所有我们需要在程序中访问和设置处理的控件。这个非常重要，我一开始就是忘了初始化加数对应的编辑框指针，后面之间访问而造成错误，无法对其进行操作。该函数将在程序窗口创建时被调用。

```
//初始化按钮控件对应的指针
    m_pBtnClose = static_cast<CButtonUI*>(m_pm.FindControl(_T("BtnClose")));
//找到从 xml 文件中的命名加载的对应按钮
    m_pBtnMax = static_cast<CButtonUI*>(m_pm.FindControl(_T("BtnMax")));
    m_pBtnMin = static_cast<CButtonUI*>(m_pm.FindControl(_T("BtnMin")));
    m_pBtnCalc=static_cast<CButtonUI*>(m_pm.FindControl(_T("BtnCalc")));
    //初始化编辑框控件对应的指针
    m_pEditSource1=static_cast<CEditUI*>(m_pm.FindControl(_T("EditSource1")));
    m_pEditSource2=static_cast<CEditUI*>(m_pm.FindControl(_T("EditSource2")));
    m_pEditResult=static_cast<CEditUI*>(m_pm.FindControl(_T("EditResult")));
```

2. OnCreate()函数

主要就是把皮肤文件名"calcUISkin.xml"添加上

```
LONG styleValue = ::GetWindowLong(*this, GWL_STYLE);
    styleValue &= ~WS_CAPTION;
    ::SetWindowLong(*this, GWL_STYLE, styleValue | WS_CLIPSIBLINGS | WS_CLIPCHILDREN);
```

```
//主窗口类与窗口句柄关联
    m_pm.Init(m_hWnd);
    CDialogBuilder builder;
    CControlUI* pRoot = builder.Create(_T("calcUISkin.xml"), (UINT)0, NULL, &m_pm);
//加载 XML 并动态创建界面无素，与布局界面元素，核心函数单独分析
    //注意：CDialogBuilder 并不是一个对话框类
    ASSERT(pRoot && "Failed to parse XML");
    if (NULL==pRoot)
//如果找不到皮肤文件则退出
    {
        MessageBox(NULL,TEXT("Cant not Find the skin!"),NULL,MB_ICONHAND);
        return 0;
    }
    m_pm.AttachDialog(pRoot);
//附加控件数据到 HASH 表中
    m_pm.AddNotifier(this);
//增加通知处理
    Init();
//应用程序其他初始化
    return 0;
```

3. Notify()函数

这个函数非常重要，它主要用于用户自定义的需要自行处理的消息以响应用户的输入，比如按钮的按下消息，控件获得焦点的消息等。在这个例子中，当输入加数按下“计算”按钮之后，就响应该“Click”消息，将编辑框中的数据取出进行求和。

```
if( msg.sType == _T("windowinit") )
        OnPrepare();
    else if( msg.sType == _T("click") )
//鼠标键按下按钮控件消息
    {
        if( msg.pSender == m_pBtnClose )
//关闭按钮按下，程序退出
        {
            PostQuitMessage(0);
            return;
        }
        else if( msg.pSender == m_pBtnMin )
//最小化按钮按下，发送系统命令消息将窗口最小化
        {
            SendMessage(WM_SYSCOMMAND, SC_MINIMIZE, 0);
            return;
        }
        else if( msg.pSender == m_pBtnMax)
 //最大化按钮按下，发送系统命令消息将窗口最大化
```

```
        {
            SendMessage(WM_SYSCOMMAND, SC_MAXIMIZE, 0);
            return;
        }
        else if (msg.pSender == m_pBtnCalc)//计算按钮按下
        {
            //从编辑控件中获取两个加数
            //判断输入的 CStdString 的内容是否为空，实际上还应该对输入数据类型做限制的
            if ((m_pEditSource1->GetText()).IsEmpty()||(m_pEditSource2->GetText()).IsEmpty())
            {
                MessageBox(NULL,TEXT("Please input two interger number first!"),NULL,MB_ICONHAND);
                return;
            }
            //将输入字符串转化为整型数
#   ifdef _UNICODE
            int add1=_wtof((m_pEditSource1->GetText()).GetData());
            int add2=_wtof((m_pEditSource2->GetText()).GetData());
#else
            int add1=atof((m_pEditSource1->GetText()).GetData());
            int add2=atof((m_pEditSource2->GetText()).GetData());
#   endif
            int add12=add1+add2;
            //将计算和整型数转化为字符串,并显示出来
            TCHAR temp1[2*sizeof((m_pEditSource1->GetText()).GetData())];
#   ifdef _UNICODE
            m_pEditResult->SetText(_itow(add12,temp1,10));
#else
            m_pEditResult->SetText(_itoa(add12,temp1,10));
#   endif
        }
    }
```

注：关于 GetText()函数，编辑框 CEditUI 类中的该函数是其基类 CControlUI 中的成员函数，其返回值为 CStdString 类型，实际上内容为 CStdString 类中的保护成员 m_pstr（LPCSTR 类型）之中，访问该成员需要用 CStdString 类的成员函数。

```
protected:
CStdString m_sText;

  // 文本相关
  virtual CStdString GetText() const;
  virtual void SetText(LPCTSTR pstrText);

CStdString CControlUI::GetText() const
{
    return m_sText;
}
-----------------------------------------
class UILIB_API CStdString
{
protected:
    LPTSTR m_pstr;
}
LPCTSTR CStdString::GetData()
{
    return m_pstr;
}
------------------ CComboUI-------------------------
CStdString CComboUI::GetText() const
//与 CEditUI 中的同名函数有所不同
{
    if( m_iCurSel < 0 ) return _T("");
    CControlUI* pControl = static_cast<CControlUI*>(m_items[m_iCurSel]);
    return pControl->GetText();
}
```

4. HandleMessage()函数

在该函数中主要完成窗口各种消息的分派处理，可以在各分支选项中调用对应的函数来完成相应的操作，算是回调函数，填充好其中的内容之后，将由类库或者系统调用。

```
LRESULT lRes = 0;
    BOOL bHandled = TRUE;
//需要应用程序自行处理
    switch( uMsg )
////应用需要处理的消息
    {
    case WM_CREATE:
//创建窗口
        lRes = OnCreate(uMsg, wParam, lParam, bHandled);
        break;
    case WM_CLOSE:
//关闭窗口
        lRes = OnClose(uMsg, wParam, lParam, bHandled);
```

```
            break;
        case WM_DESTROY:
//销毁窗口
            lRes = OnDestroy(uMsg, wParam, lParam, bHandled);
            break;
        case WM_NCACTIVATE:
//激活非客户区
            lRes = OnNcActivate(uMsg, wParam, lParam, bHandled);
            break;
        case WM_NCCALCSIZE:
//重新计算客户区尺寸
            lRes = OnNcCalcSize(uMsg, wParam, lParam, bHandled);
            break;
        case WM_NCPAINT:
//重绘非客户区
            lRes = OnNcPaint(uMsg, wParam, lParam, bHandled);
            break;
        case WM_NCHITTEST:
//枚举鼠标所在区域
            lRes = OnNcHitTest(uMsg, wParam, lParam, bHandled);
            break;
        case WM_SIZE:
//改变窗口尺寸
            lRes = OnSize(uMsg, wParam, lParam, bHandled);
            break;
        case WM_GETMINMAXINFO:
//获得显示窗口的最大最小尺寸信息
            lRes = OnGetMinMaxInfo(uMsg, wParam, lParam, bHandled);
            break;
        case WM_SYSCOMMAND:
 //处理系统命令，包括菜单命令，最大最小化按钮按下
            lRes = OnSysCommand(uMsg, wParam, lParam, bHandled);
            break;
        default:
//不需要应用程序自行处理
            bHandled = FALSE;
        }
        if( bHandled )
            return lRes;
        if( m_pm.MessageHandler(uMsg, wParam, lParam, lRes) )
  //DUILIB 库帮处理的消息及相关的处理函数
            return lRes;
        return CWindowWnd::HandleMessage(uMsg, wParam, lParam);
//应用层和 DUILIB 都不处理的消息交由系统默认处理
```

5.程序入口点 Winmain()函数

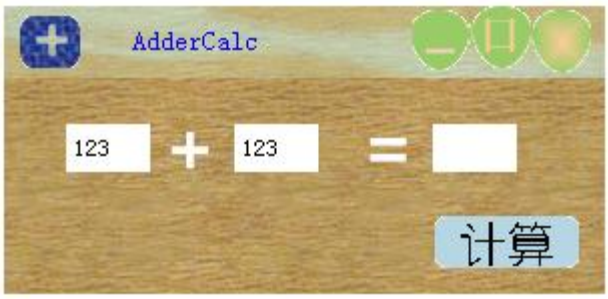
程序在这里完成必要的初始化和创建窗口，然后调用函数进入消息循环。

```
int APIENTRY _tWinMain(HINSTANCE hInstance,
                       HINSTANCE hPrevInstance,
                       LPTSTR    lpCmdLine,
                       int       nCmdShow)
{
CPaintManagerUI::SetInstance(hInstance);
//设置程序实例
CPaintManagerUI::SetResourcePath(CPaintManagerUI::GetInstancePath()
    + _T("calcskin\\skins"));
//实例句柄与渲染类关联,获得皮肤文件目录（加载皮肤文件在 OnCreate 之中）
HRESULT Hr = ::CoInitialize(NULL);
//初始化 COM 库，为加载 COM 库提供支持
if( FAILED(Hr) )
return 0;

CAdderCalc* pAppCalc = new CAdderCalc();
//创建应用程序窗口类对象
if( pAppCalc == NULL )
return 0;
pAppCalc->Create(NULL, _T("AdderCalc"), UI_WNDSTYLE_DIALOG, 0);
pAppCalc->CenterWindow();
//将窗口放到桌面中央
pAppCalc->ShowWindow(true);
//显示窗口
CPaintManagerUI::MessageLoop();
//进入消息循环
::CoUninitialize();
//退出程序并释放 COM 库
return 0;
```

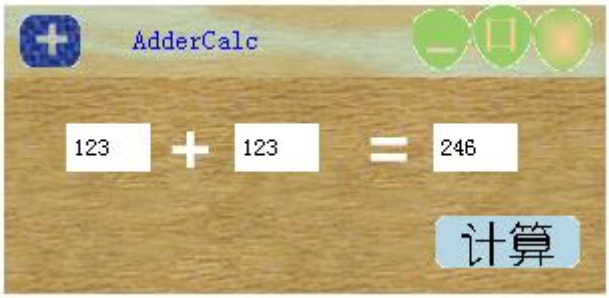
程序的其他部分见源码。

完成代码输入调试完毕之后就可以调试了。

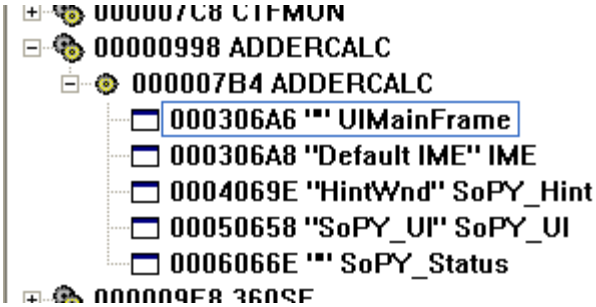
运行时的界面如下图：



点击“计算”按钮之后得到：



对于该程序，我们使用 SPY++工具来观察可看到如下图所示：



如图蓝色框所示，的确主要只有一个窗口，实现了 DirectUI 的想法，而不像传统的 WindowsForm 程序，是由很多的控件子窗口构成的，这样的应用程序简化了消息循环的复杂度和冗余度，执行效率更高。

暂时就只支持整数加法运算，以后再扩展功能吧，界面临时制作的，比较丑，让大家见笑了，不过既然决定分享了就欢迎大家拍砖(borlittle@qq.com)。个人感觉利用 Duilib 库还是比较方便的，尤其团队开发了那么多方便的控件和工具，可以让大家随心地打造自己的程序界面。相信随着越来越多的人加入到 Duilib 库的开发和使用过程中来，以后会拥有更多的扩展，更为强大的功能。

感谢 Bjarke Viksoe、Duilib 团队的工作，感谢 Duilib 官方交流群 3 中码农、看我的头像等朋友的帮助。