2017-2-14

SOUI 界面库 v2.3.1.1

用户使用手册

本文档根据启程软件的博客整理而成，欢迎大家提出修改意见、使用过程中的心得

User
启程软件

修订历史			
日期	版本	作者	说明
2017/02/14	0.1	启程软件	文档初稿

目 录

1、概述

1.1 SOUI 是什么？

SOUI 是什么？SOUI 是启程软件开发的一个 C++ Direct UI 库。DirectUI 技术一般是指将所有的界面控件都绘制在一个窗口上，这些控件的逻辑和绘图方式都必须自己进行编写和封装，而不是使用 Windows 控件，所以这些控件都是无句柄的。虽然 Direct UI 不是什么新技术，但是要把 UI 做好，Direct UI 确实是目前为止最有效的解决方案。

DirectUI 技术需要解决的主要问题如下：

- 窗口的子类化，截获窗口的消息。
- 封装自己的控件，并将自己的控件绘制到该窗口上。
- 封装窗口的消息，并分发到自己的控件上，让自己的控件根据消息进行响应和绘制。
- 根据不同的行为发送自定义消息给窗口，以便程序进行调用。
- 一般窗口上控件的组织使用 XML 来描述。

通常 DirectUI 的界面库都采用 XML 配置文件+图片+控制脚本（Lua、Javascript 等）的开发方式，非常类似于 Web 程序的开发方式，当然这里面控制脚本也可以直接使用 C++代码来实现。这种开发方式可以大大提高开发效率，将程序员从繁琐的界面工作中解脱出来，并且通过美工的设计，可以使界面更美观。

SOUI 不是一个新项目，它是基于之前开发的 DuiEngine 界面库，再经过为期近半年的重构后完成的版本，除了 skia 渲染模块还有待进一步完善外，其它模块已经基本达到发布标准。（DuiEngine 是一个基于金山 Bkwin 重构的 UI 库，经过不断更新，已经相对稳定，并且已经在多个有大量用户的客户端产品中使用）

1.2 SOUI 相对于 DuiEngine 的改进

做 SOUI 有几个核心需求：

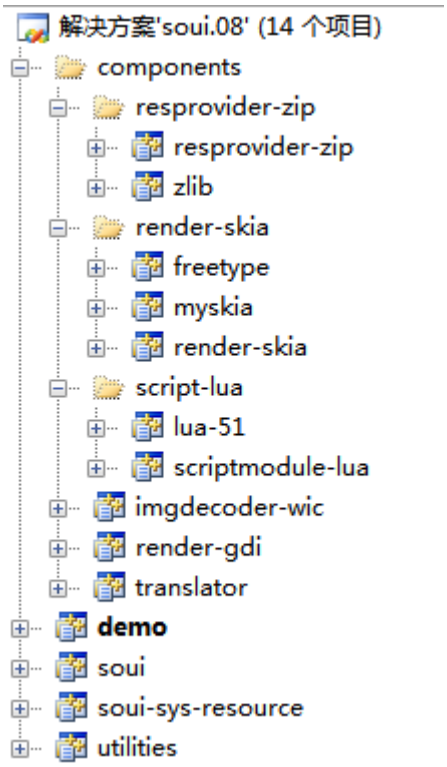
- 功能模块化：特别是渲染部分要让用户可以根据需要选择适合的模块，如 GDI，GDI+，SKIA；
- 性能优化,简化 XML 配置；
- 增加代码注释；
- 删除原有项目中为了版本兼容而遗留的垃圾代码;
- 优化项目管理;

同时新版本也增加了一些新的特性：

- 新增多语言翻译支持;
- 程序资源提供模块支持多份，以便为从脚本创建 UI 提供更好的支持，完善 LUA 脚本模块;

1.3 SOUI 模块结构

下图为 SOUI 模块截图：



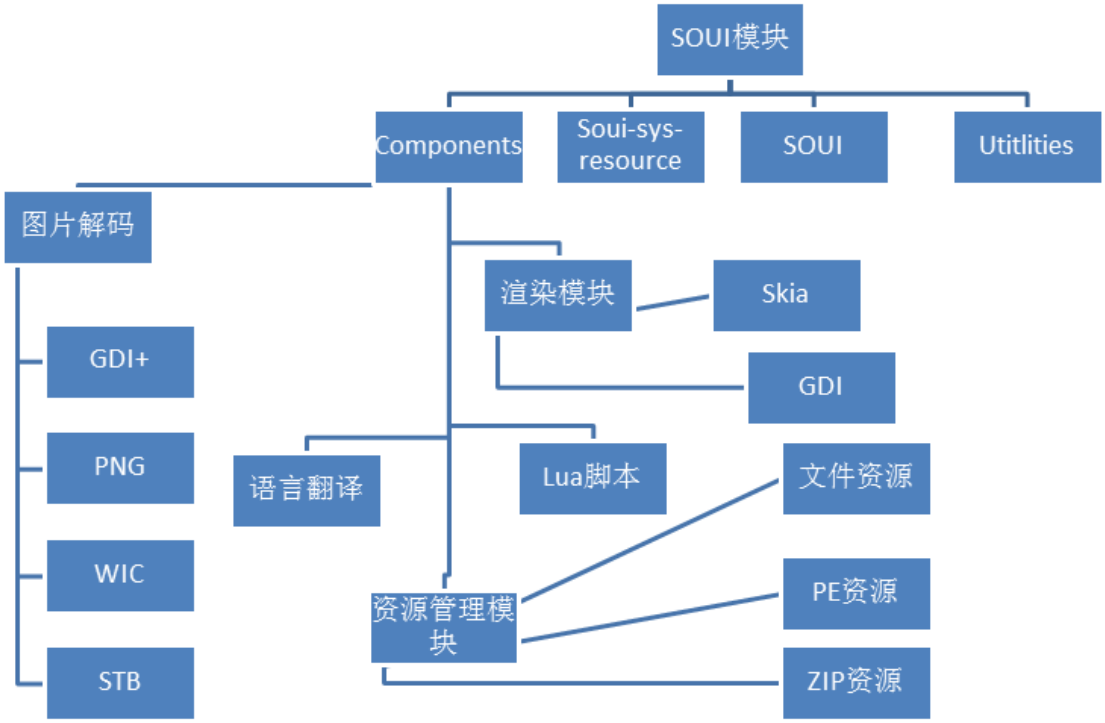
- soui 模块是整个项目的核心，除 utilities 模块外，其它模块都为 soui 模块服务。
- utilities 模块提供一些工具类，主要包含 pugixml，及一个 String 类。
- soui-sys-resource 模块是一个纯资源 DLL，提供一些内置控件必须的资源。
- demo 模块是 SOUI 界面库的功能演示程序。
- translator 实现一个从 XML 文件加载多语言翻译资料的类似 QT 的语言翻译模块。
- render-gdi 和 render-skia 分别实现两个基于 GDI 及 SKIA 的渲染模块，它们可以相互替换。GDI 的优点是体积小，但是对于 apha 通道支持能力有限；而 skia 的优点是速度快，全面支持 alpha 通道，但是程序体积会有所增加, DLL 编译后有 1M，压缩后有 600K。
- resprovider-zip 实现了一个从 ZIP 文件加载程序资源的模块。加上 soui 中内置的两个资源加载模块，SOUI 可以选择从文件中，从 EXE 资源中及从 ZIP 文件包中加载程序资源。

.◆ script-lua 是一个脚本支持模块，目前只实现了几个基本类的导出，要使用更多 SOUI 类型，还需要增加导出代码。

1.4 模块结构图及框架图

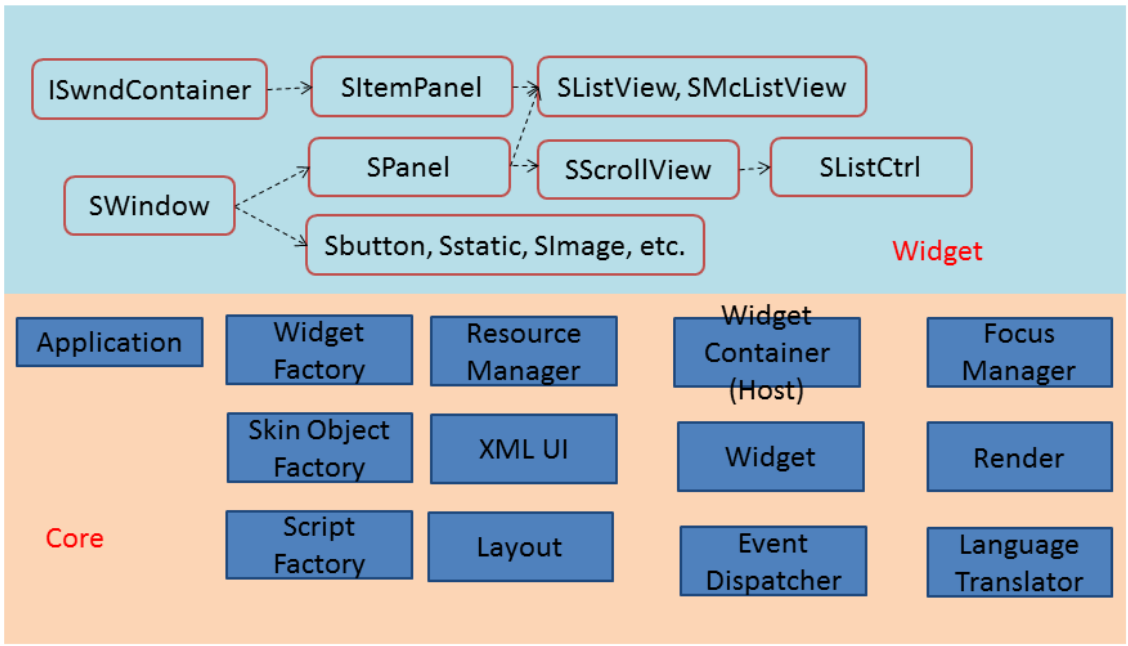
模块结构图：

SOUI模块结构图



SOUI 框架图：

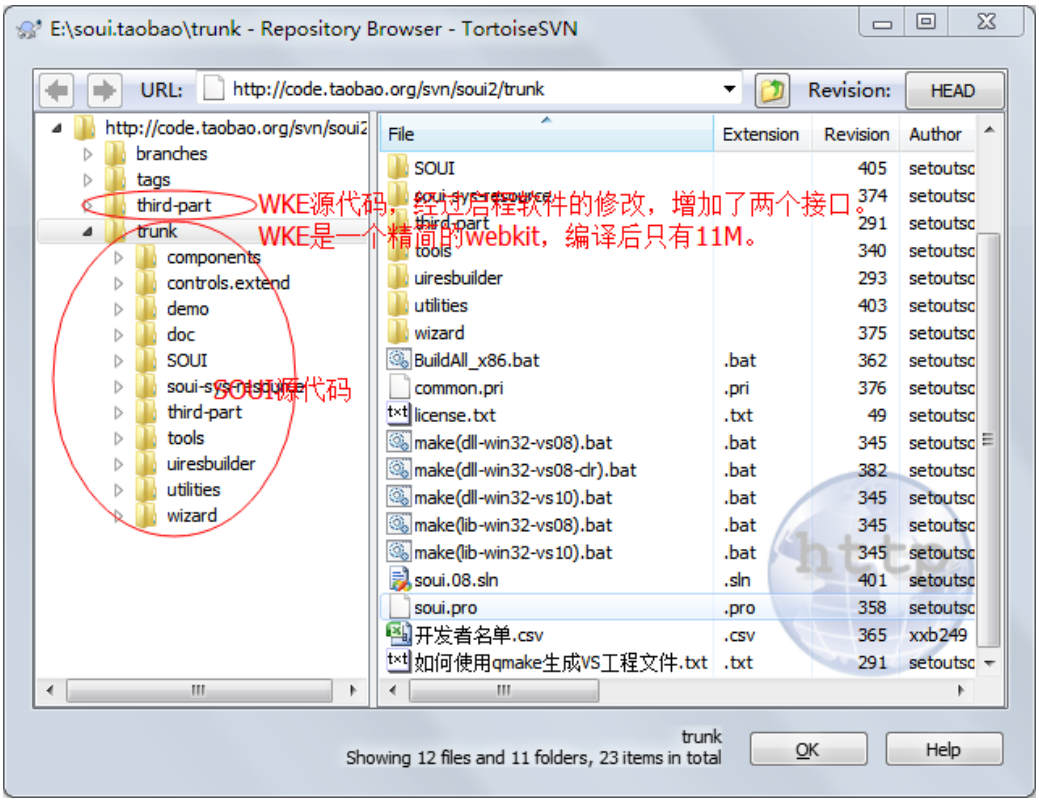
SOUI架构图

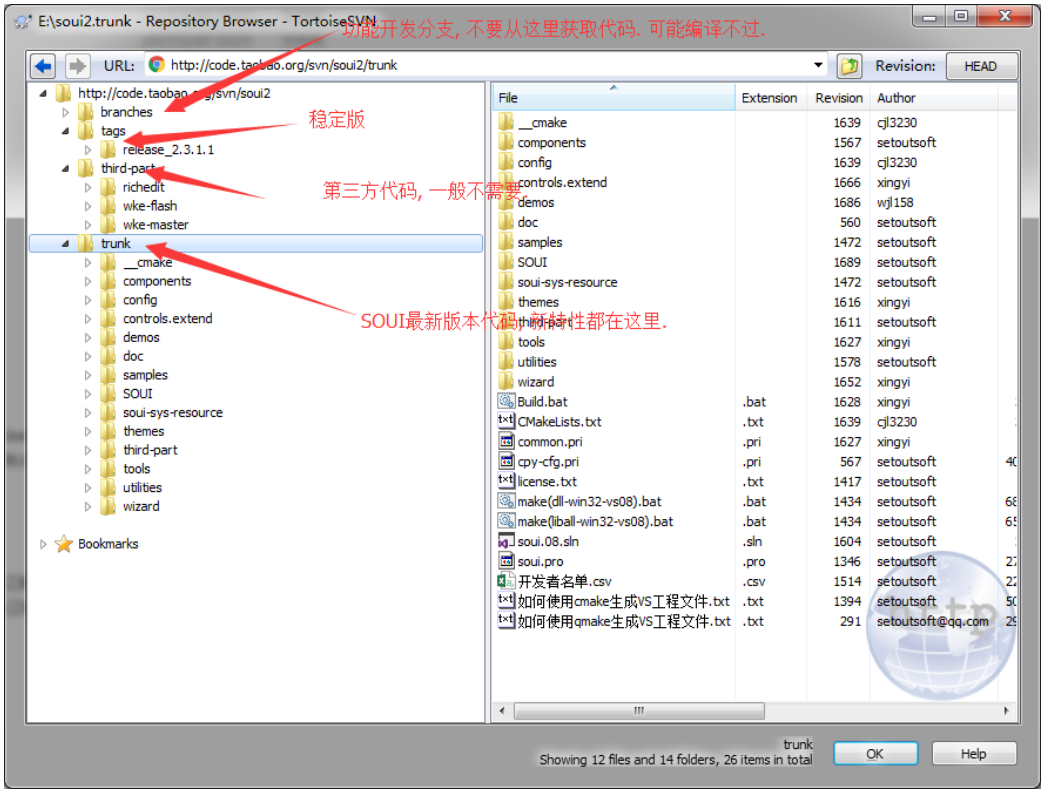


2、SOUI 的编译

2.1 获取 SOUI 的源代码

SOUI 是开源软件，其源码采用 svn 管理（http://code.taobao.org/svn/soui2 ）；也可以通过 git 客户端获取稳定版本：https://github.com/setoutsoft/soui。





这里主要包含两个目录：trunk 及 third-part。trunk 目录保存 SOUI 项目的全部代码，third-part 保存 soui 系统使用到的不方便放到 trunk 的第三方库，目前只有一个 WKE(一个精简的 webkit)的源代码。一般情况下只获取 trunk 的代码就行。

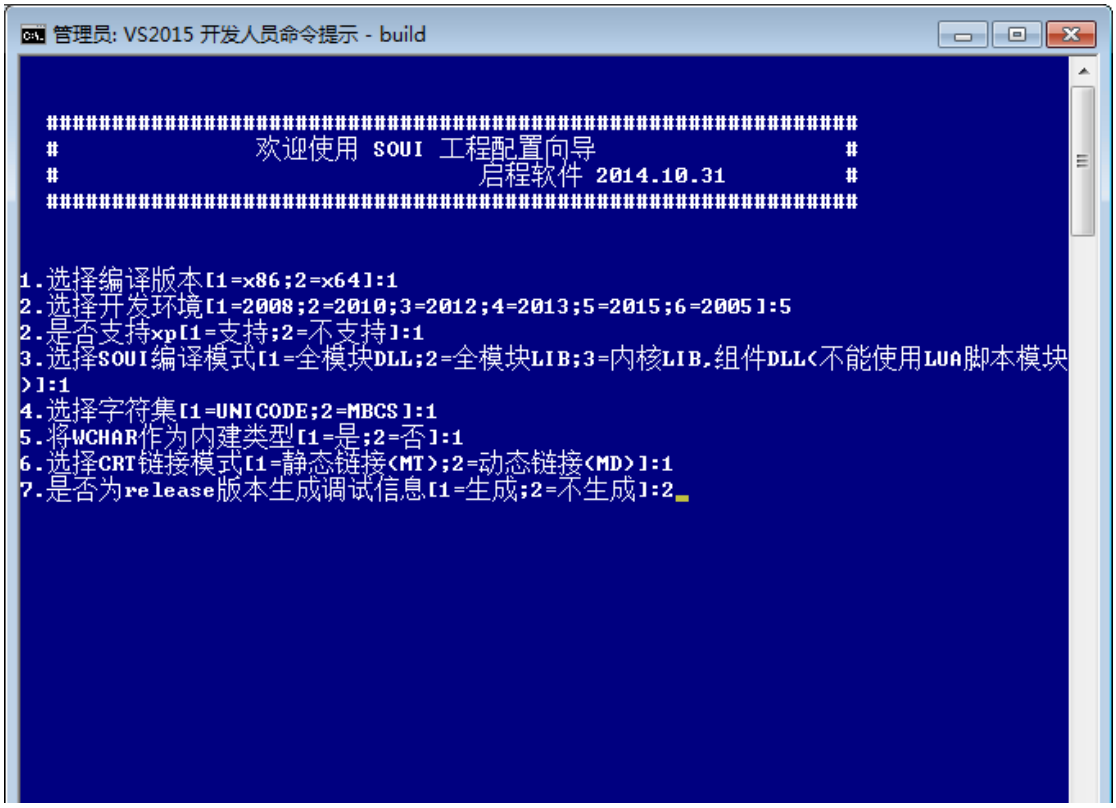
2.2 编译 SOUI 界面库

SOUI 项目采用 QT 的 qmake 管理项目文件，qmake 已经从 QT 中分离出来，不需要你的机器上安装 QT。（项目中也包含了网友提供的 cmake 编译脚本）

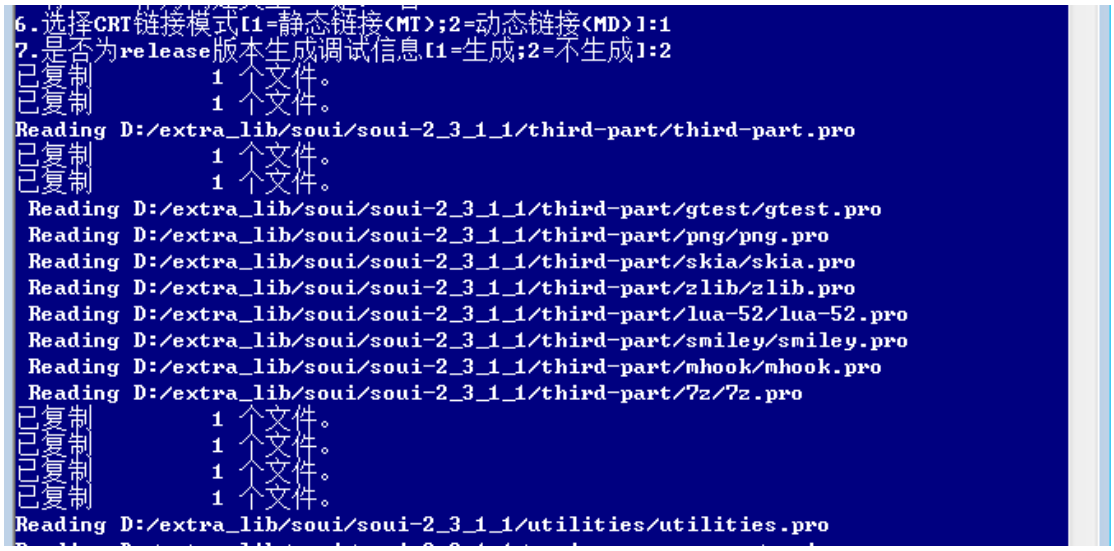
如果你在使用 vs2015，建议直接使用 build.bat 来编译 SOUI 库。

首先运行“vs2015 开发人员命令提示”，如果提示无法找到路径之类的提示，请用管理员权限运行，这种情况大部分是在远程桌面编译 SOUI 库时出现。

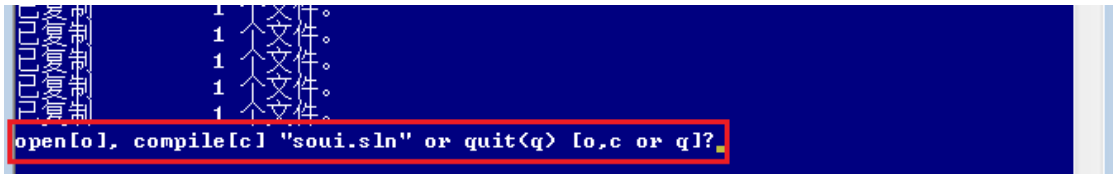
使用“cd /d ”命令，把当前路径，切换至 SOUI 项目文件夹，运行 build.bat 批处理。



根据自己的需要配置编译选项；



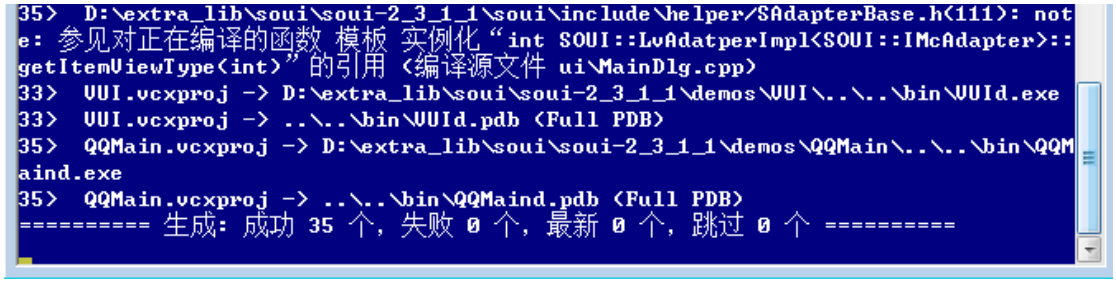
配置完成后，会询问是打开 sln 文件、直接编译、或者退出批处理。



可以直接选择直接编译，省得还用 vs2015 打开 sln 文件进行编译。

不知道 vs2015 编译比较严格，还是什么原因，在编译过程中可能会出现很多的 waring。

等待编译完成后，会提示成功 35 个，这说明已经成功编译 SOUI 界面库了（如下图）：



==========附、作者微博提供的编译教程==========

如果你的机器上安装了 VS2008，可以直接打开 trunk 的根目录下的 soui.08.sln 来编译，这个项目中各工程的编译依赖已经设置好，直接 F7 就可以全部完成编译。

如果你的机器安装的是其它版本（支持 vs2005-vs2015），可以采用 trunk 目录下的 make(*).bat 来生成对应版本的项目文件，项目文件生成成功后会在根目录生成一个 soui.sln，打开该 sln 即可。VS2010+的版本需要先生成 VS2010 的项目文件，再用 VS 打开并升级。要生成 vs2005，可以手动修改 make(*).bat 中的参数。

如果安装的是 vs2008 或者 vs2010 还可以使用 buildAll_x86.bat 来生成项目文件并使用命名行完成编译。

打开 make(dll-win32-vs08).bat 可以看到里面只有两行代码：

call "%VS90COMNTOOLS%..\..\VC\vcvarsall.bat" x86

tools\qmake -tp vc -r -spec .\tools\mkspecs\win32-msvc2008 "CONFIG += DLL_SOUI USING_MT CAN_DEBUG"

第一行通过 VS 的环境变量加载 VS 的 PATH 信息。

第二行调用 qmake 生成项目文件： -spec 后面的参数指定生成的项目文件 VS 版本(03,05,08,10)，CONFIG += ***用来控制如何生成项目文件。项目文件支持 4 个预定义参数：

DLL_SOUI:代表将 SOUI 模块编译生成一个 DLL，没有该参数则生成 LIB；

USING_MT:代表使用 MT 方式连接 CRT，否则采用 MD 方式；

CAN_DEBUG:为 release 版本生成调试符号；

USING_CLR:项目提供“公共语言运行时”支持；

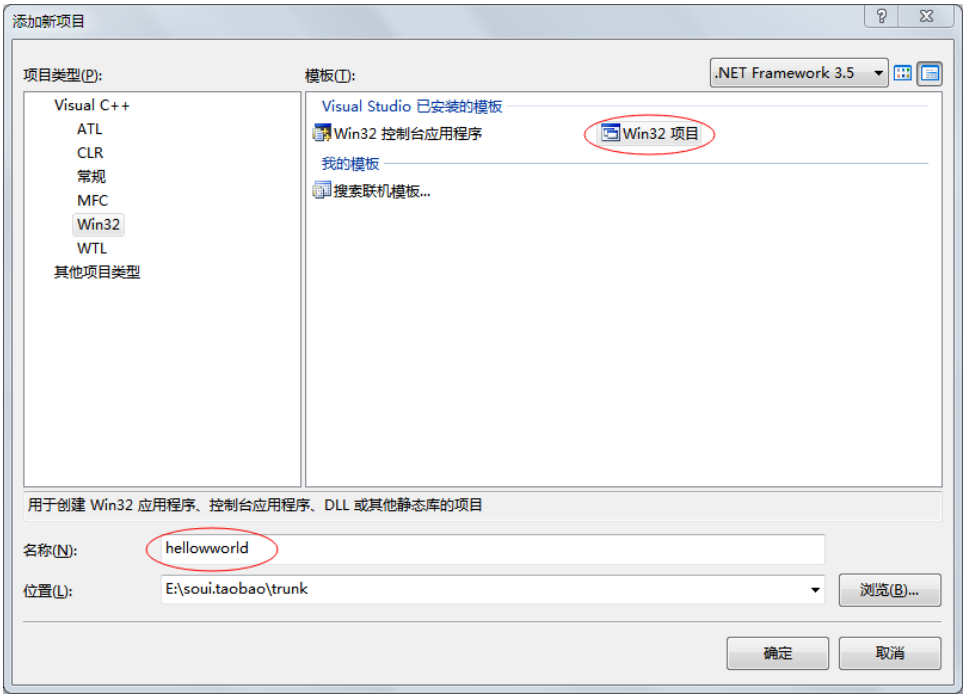
如果需要其它配置，可以手动修改 common.pri。

3、开始使用 SOUI

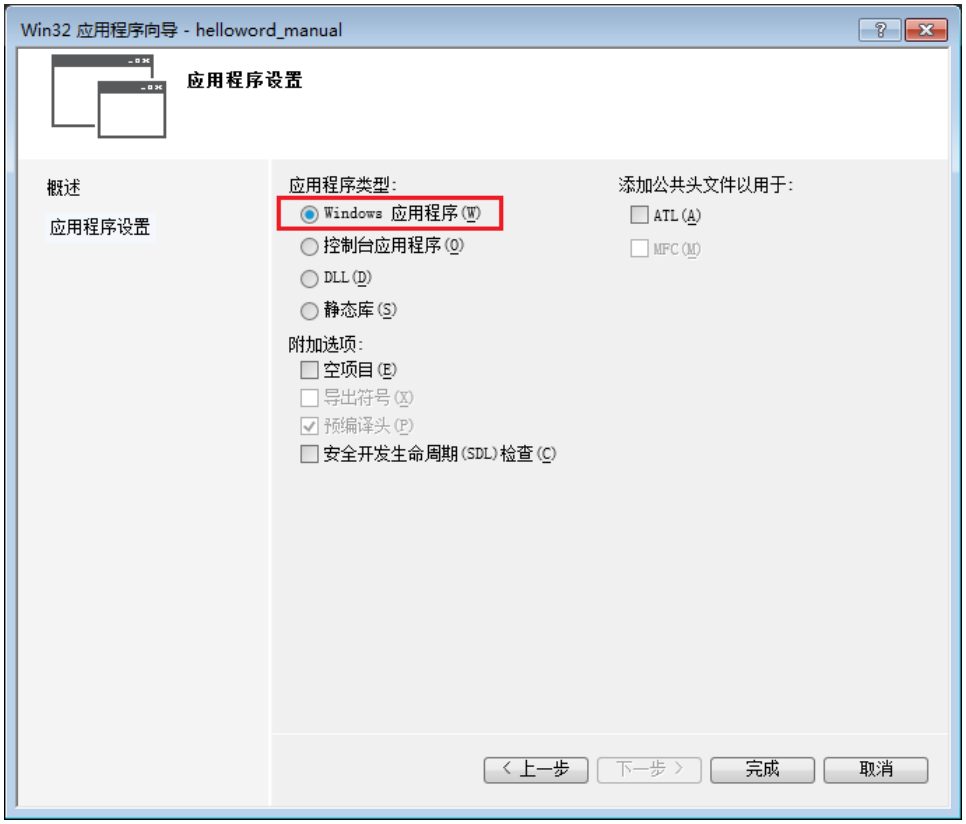
3.1 创建 SOUI 项目（手工创建）

3.1.1 项目环境配置

SOUI 项目本质是一个基于 Win32 窗口的应用程序。因此首先我们可以从 Win32 窗口应用程序向导创建一个简单的 Win32 项目。

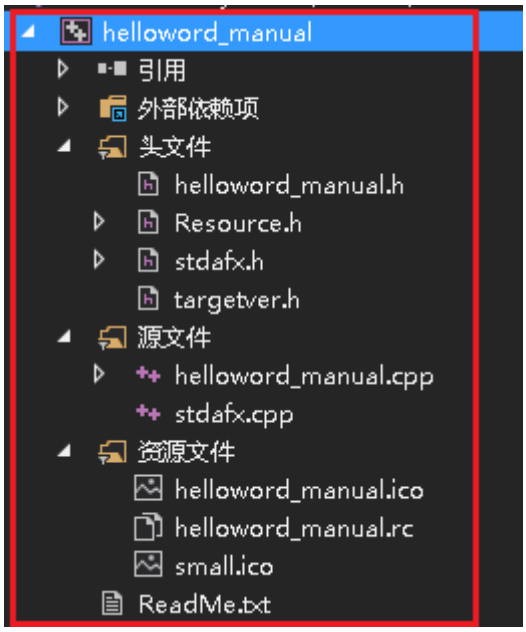


并在第 3 页选择“Window 应用程序”



选择“完成”后生成一个 Win32 应用程序骨架。

项目的文件结构如下图：

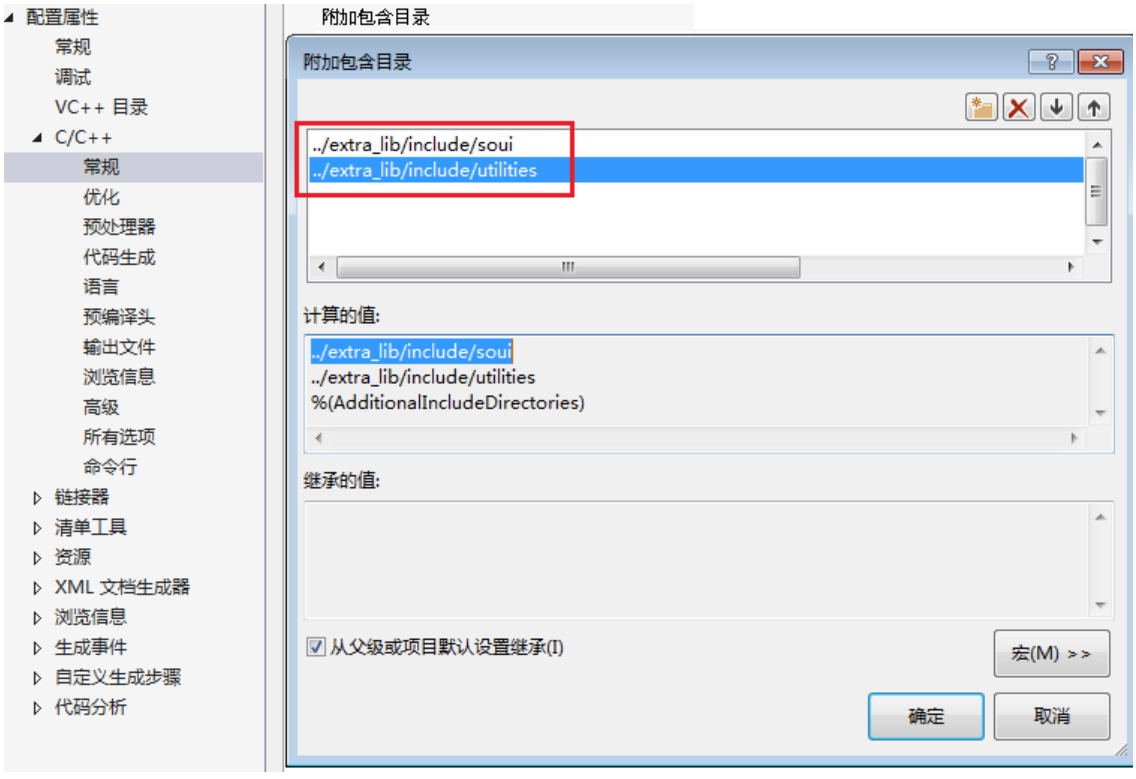


要使用 SOUI 开发程序，首先当然是要找到从 SVN 获取的 SOUI 项目代码，编译后把 include 和 lib 拷贝到项目文件夹。假定把头文件和库都拷贝至项目 extra_lib 文件夹下。

我们需要在 VS 的 include 目录中增加两个目录：

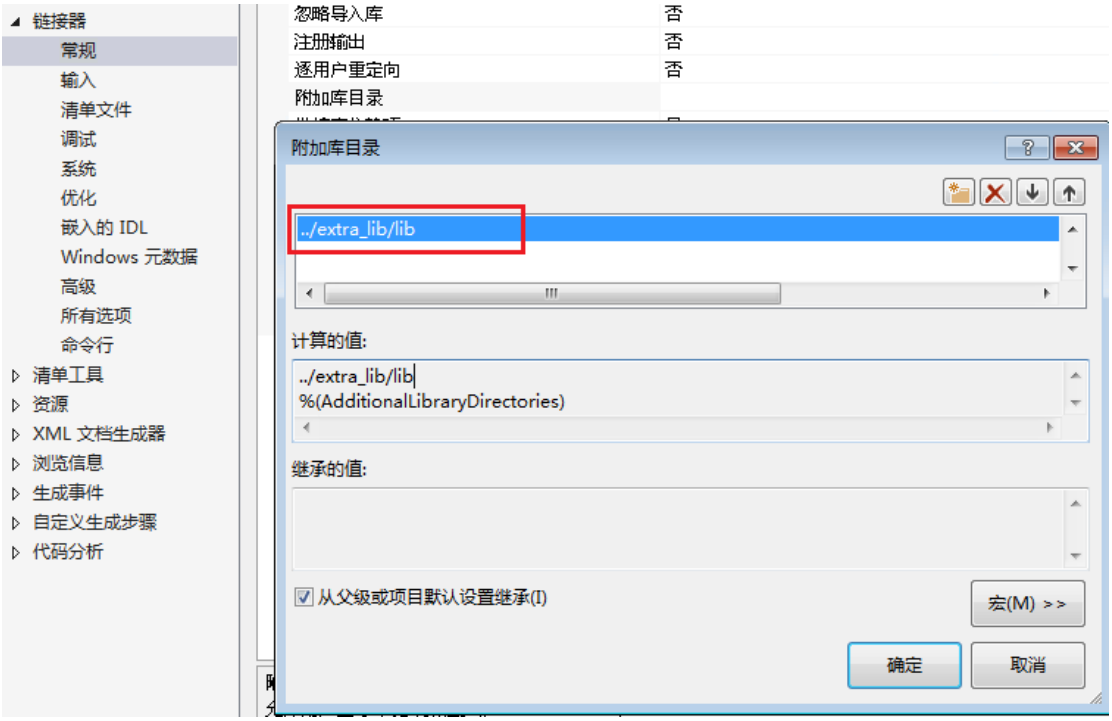
"..\extra_lib\include\soui"；"..\extra_lib\include\utilities"

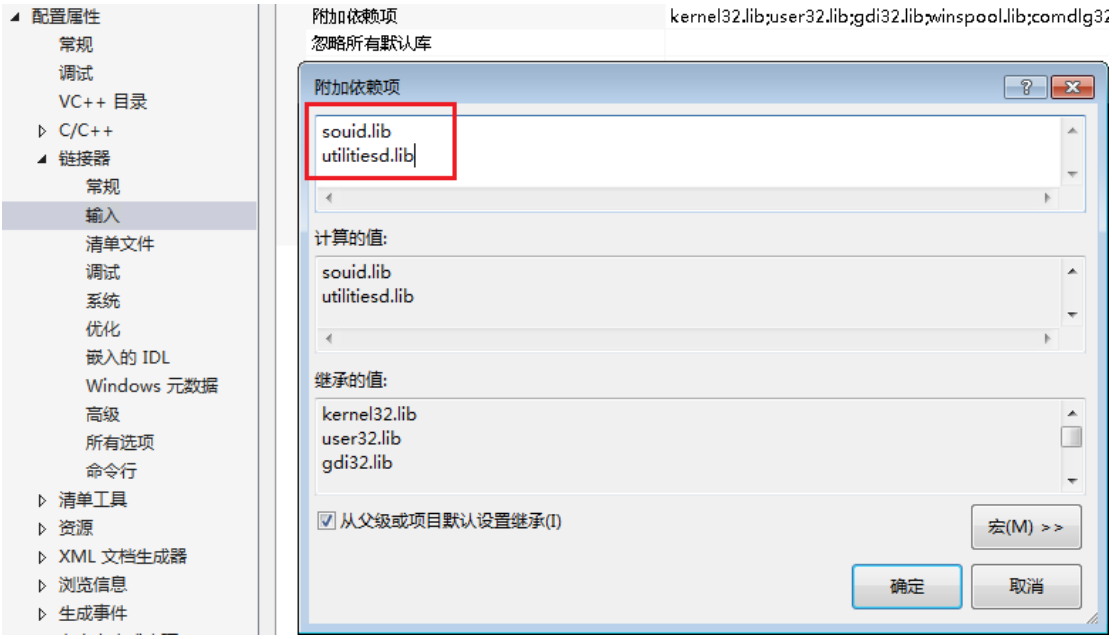
如图：



注：SOUI ver2.3.1.1 还需要增加一个包含目录：..\extra_lib\include\config

首先设置设置附加库目录为“../extra_lib/lib”，再附加依赖项添加两个库文件：souid.lib,utilitiesd.lib(以 debug 版本为例）。





设置好项目后，默认情况下还需要把编译选项中的“代码生成”从 MDd 修改为 MTd，并把“将 wchar_t 视为内置类型”修改为“否”。（因为这是 SOUI 的默认编译配置）

到这里环境配置基本完成。

3.1.2 项目资源准备

到这里准备工作还没有完，我们需要为 SOUI 准备一套程序资源，并把它放到项目目录下。

最基本的资源至少应该包括：

uires.idx：定义资源索引

init.xml：定义全局 UI 的属性，包含字体，字符串表，skin，style，objattr，参见前篇介绍。

dlg_main.xml:主窗口而已 XML。

init.xml 和 dlg_main.xml 的文件名不限，但是 uires.idx 的文件名是固定的。

uires.idx：

```
<resource>
  <UIDEF>
    <file name="XML_INIT" path="xml\init.xml"  />
  </UIDEF>
  <LAYOUT>
    <file name="XML_MAINWND" path="xml\dlg_main.xml"  />
  </LAYOUT>
</resource>
```

init.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<UIDEF>
  <font face="宋体" size="15"/>
  <string>
    <title value=""/>
    <ver value="1.0"/>
  </string>
  <skin>
  </skin>
  <style>
    <class name="normalbtn" font="" colorText="#385e8b"
colorTextDisable="#91a7c0" textMode="25" cursor="hand" margin-x="0"/>
  </style>
  <objattr>
  </objattr>
</UIDEF>
```

dlg_main.xml:

```
<SOUI name="mainWindow" title="%title% ver:%ver%" width="600" height="400"
appWnd="1" margin="20,5,5,5"  resizable="1" translucent="1" >
  <root skin="_skin.sys.wnd.bkgnd">
    <caption pos="0,0,-0,30">
```

```
    <text pos="11,9">%title% ver:%ver%</text>
    <imgbtn name="btn_close" skin="_skin.sys.btn.close"    pos="-45,0"
tip="close" animate="1"/>
    <imgbtn name="btn_max" skin="_skin.sys.btn.maximize"  pos="-83,0"
animate="1" />
    <imgbtn name="btn_restore" skin="_skin.sys.btn.restore"  pos="-83,0"
show="0" animate="1" />
    <imgbtn name="btn_min" skin="_skin.sys.btn.minimize" pos="-121,0"
animate="1" />
  </caption>
  <window pos="5,30,-5,-5">
    <text pos="|0,|0" pos2type="center" colorText="#ff0000">Hellow World!
UI? Just so so!</text>
    <button class ="normalbtn" pos="|-50,[20,@100,@30"
name="btn_msgbox">show msg box</button>
  </window>
 </root>
</SOUI>
```

如果对 UI 布局不明白可以参考章节 4.1.2（即“第五篇”<需要更新>）。

3.1.3 开始编码【TODO：不适用 SOUI ver2.3.1.1，需更新】

要开始编写代码还需要修改一下 stdafx.h 如下：

```
// stdafx.h : 标准系统包含文件的包含文件，
// 或是经常使用但不常更改的
// 特定于项目的包含文件
//
#pragma once

#include "targetver.h"

#define _CRT_SECURE_NO_WARNINGS
#define    DLL_SOUI   //SOUI 是以 DLL 提供时需要定义这个宏

#include <souistd.h>
#include <core/SHostDialog.h>
#include <control/SMessageBox.h>
#include <control/souictrls.h>

using namespace SOUI;
```

功能就是引用几个 SOUI 头文件。

打开 helloworld.cpp，删除文件中除空 main 函数以外的全部代码，剩下的代码如下：

```
// hellowworld.cpp : 定义应用程序的入口点。
//

#include "stdafx.h"
#include "hellowworld.h"

int APIENTRY _tWinMain(HINSTANCE hInstance,
                       HINSTANCE hPrevInstance,
                       LPTSTR    lpCmdLine,
                       int       nCmdShow)
{
    return 0;
}
```

这是一个可以编译的空项目。下面，我们开始填充 Main 程序框架：

```
// helloworld.cpp : main source file
//

#include "stdafx.h"

#include <com-loader.hpp>

#ifdef _DEBUG
#define COM_IMGDECODER  _T("imgdecoder-wicd.dll")
#define COM_RENDER_GDI  _T("render-gdid.dll")
#define SYS_NAMED_RESOURCE _T("soui-sys-resourced.dll")
#else
#define COM_IMGDECODER  _T("imgdecoder-wic.dll")
#define COM_RENDER_GDI  _T("render-gdi.dll")
#define SYS_NAMED_RESOURCE _T("soui-sys-resource.dll")
#endif

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE /*hPrevInstance*/,
LPTSTR /*lpstrCmdLine*/, int /*nCmdShow*/)
{
    HRESULT hRes = OleInitialize(NULL);
    SASSERT(SUCCEEDED(hRes));

    int nRet = 0;

    SComLoader imgDecLoader;
    SComLoader renderLoader;
    SComLoader transLoader;
```

```
    //将程序的运行路径修改到项目所在目录所在的目录
    TCHAR szCurrentDir[MAX_PATH]={0};
    GetModuleFileName( NULL, szCurrentDir, sizeof(szCurrentDir) );
    LPTSTR lpInsertPos = _tcsrchr( szCurrentDir, _T('\\') );
    _tcscpy(lpInsertPos+1,_T(".."));
    SetCurrentDirectory(szCurrentDir);
    {
        CAutoRefPtr<SOUI::IImgDecoderFactory> pImgDecoderFactory;
        CAutoRefPtr<SOUI::IRenderFactory> pRenderFactory;

imgDecLoader.CreateInstance(COM_IMGDECODER,(IObjRef**)&pImgDecoderFactory);

renderLoader.CreateInstance(COM_RENDER_GDI,(IObjRef**)&pRenderFactory);
        pRenderFactory->SetImgDecoderFactory(pImgDecoderFactory);

        SApplication *theApp=new SApplication(pRenderFactory,hInstance);

        HMODULE hSysResource=LoadLibrary(SYS_NAMED_RESOURCE);
        if(hSysResource)
        {
            CAutoRefPtr<IResProvider> sysSesProvider;
            CreateResProvider(RES_PE,(IObjRef**)&sysSesProvider);
            sysSesProvider->Init((WPARAM)hSysResource,0);
            theApp->LoadSystemNamedResource(sysSesProvider);
        }

        CAutoRefPtr<IResProvider>   pResProvider;
        CreateResProvider(RES_FILE,(IObjRef**)&pResProvider);
        if(!pResProvider->Init((LPARAM)_T("uires"),0))
        {
            SASSERT(0);
            return 1;
        }
        theApp->AddResProvider(pResProvider);
         //2.x版本已经不需要下面这行。
        //theApp->Init(_T("XML_INIT"));
        {//在这里加入主窗口运行代码
        }
        delete theApp;
    }
    OleUninitialize();
    return nRet;
}
```

将 helloworld.cpp 修改如上。

这里主要功能是配置几个 SOUI 需要的组件。

接下来，为主窗口布局生成一个 C++类

和 MFC，WTL 等一样，SOUI 可以有两种方式显示窗口：模态与非模态。

这里我们采用非模态窗口来演示。

因此我们需要从 SHostWnd 派生出一个 C++对象：CMainWnd

```
// MainWnd.h : interface of the CMainWnd class
//
/////////////////////////////////////////////////////////////////////////////
/
#pragma once

class CMainWnd : public SHostWnd
{
public:
    CMainWnd()
        : SHostWnd(_T("LAYOUT:XML_MAINWND"))//这里定义主界面需要使用的布局文件
    {
        m_bLayoutInited=FALSE;
    }

    void OnClose()
    {
        PostMessage(WM_QUIT);
    }
    void OnMaximize()
    {
        SendMessage(WM_SYSCOMMAND,SC_MAXIMIZE);
    }
    void OnRestore()
    {
        SendMessage(WM_SYSCOMMAND,SC_RESTORE);
    }
    void OnMinimize()
    {
        SendMessage(WM_SYSCOMMAND,SC_MINIMIZE);
    }

    void OnSize(UINT nType, CSize size)
    {
        SetMsgHandled(FALSE);
```

```
        if(!m_bLayoutInited) return;
        if(nType==SIZE_MAXIMIZED)
        {
            FindChildByName(L"btn_restore")->SetVisible(TRUE);
            FindChildByName(L"btn_max")->SetVisible(FALSE);
        }else if(nType==SIZE_RESTORED)
        {
            FindChildByName(L"btn_restore")->SetVisible(FALSE);
            FindChildByName(L"btn_max")->SetVisible(TRUE);
        }
    }
    void OnBtnMsgBox()
    {
        SMessageBox(NULL,_T("this is a message
box"),_T("haha"),MB_OK|MB_ICONEXCLAMATION);
        SMessageBox(NULL,_T("this message box includes two
buttons"),_T("haha"),MB_YESNO|MB_ICONQUESTION);
        SMessageBox(NULL,_T("this message box includes three
buttons"),NULL,MB_ABORTRETRYIGNORE);
    }

    BOOL OnInitDialog( HWND hWnd, LPARAM lParam )
    {
        m_bLayoutInited=TRUE;

        return 0;
    }
protected:
    //按钮事件处理映射表
    EVENT_MAP_BEGIN()
        EVENT_NAME_COMMAND(L"btn_close",OnClose)
        EVENT_NAME_COMMAND(L"btn_min",OnMinimize)
        EVENT_NAME_COMMAND(L"btn_max",OnMaximize)
        EVENT_NAME_COMMAND(L"btn_restore",OnRestore)
        EVENT_NAME_COMMAND(L"btn_msgbox",OnBtnMsgBox)
    EVENT_MAP_END()

    //窗口消息处理映射表
    BEGIN_MSG_MAP_EX(CMainWnd)
        MSG_WM_INITDIALOG(OnInitDialog)
        MSG_WM_CLOSE(OnClose)
        MSG_WM_SIZE(OnSize)
```

```
        CHAIN_MSG_MAP(SHostWnd)//注意将没有处理的消息交给基类处理
        REFLECT_NOTIFICATIONS_EX()
    END_MSG_MAP()
private:
    BOOL          m_bLayoutInited;
};
```

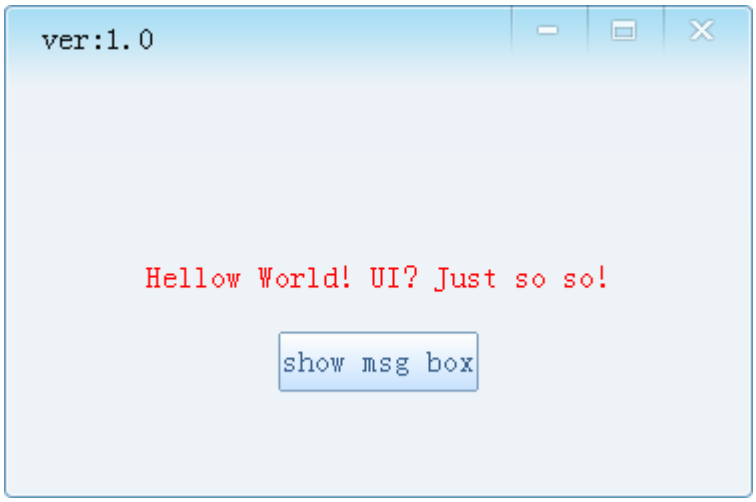
为了介绍方便，这个窗口类所有代码都在这个 mainwnd.h 文件里。

再进一步填充 main 函数

```
{
   //在这里加入主窗口运行代码
   CMainWnd wndMain;
   wndMain.Create(GetActiveWindow(),0,0,800,600);
   wndMain.SendMessage(WM_INITDIALOG);
   wndMain.CenterWindow(wndMain.m_hWnd);
   wndMain.ShowWindow(SW_SHOWNORMAL);
   nRet=theApp->Run(wndMain.m_hWnd);
   //程序结束
}
```

当然 main 前面还要有一行：#include "mainwnd.h"

大功造成！编译项目，运行结果如下：



3.2 √创建 SOUI 项目（通过向导创建）

开发一个使用 SOUI 的应用程序，如果从 Win32 项目类型开始，即便是对 SOUI 这样了如指掌，也至少要 10 分钟以上才能配置出一个使用 SOUI 的开发环境。

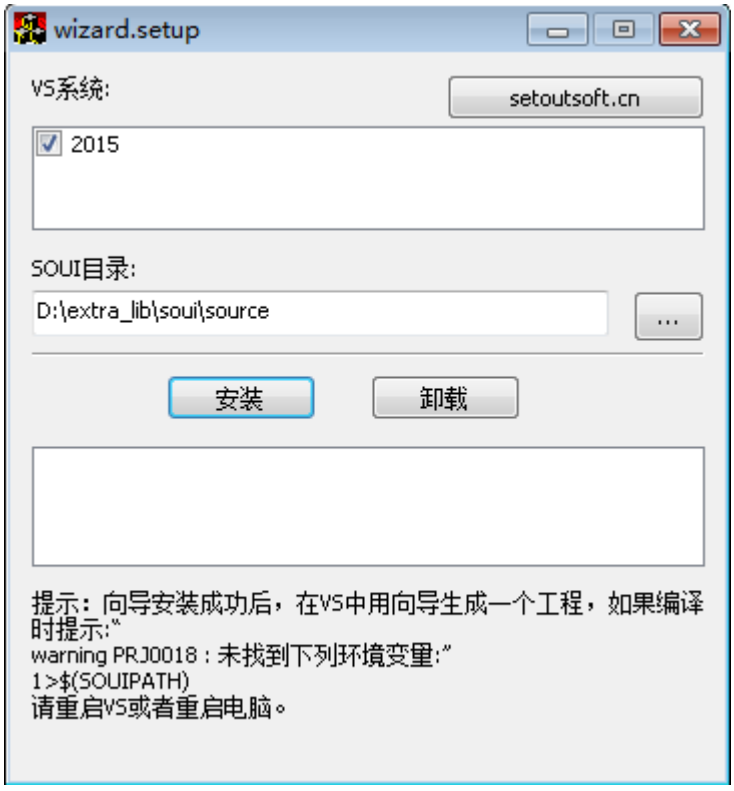
为了帮助程序员从简单的重复劳动解脱出来，也帮助初学都快速开始，我特别为 SOUI 制作了一个适用于 VS 各版本的应用程序向导。

使用向导可以通过简单的选择两个选项就自动完全项目配置，编译即可看到 UI 布局结果。

需要注意的是，使用那个版本的 SOUI，需要安装对应版本的向导。

3.2.1 SOUI 向导的安装

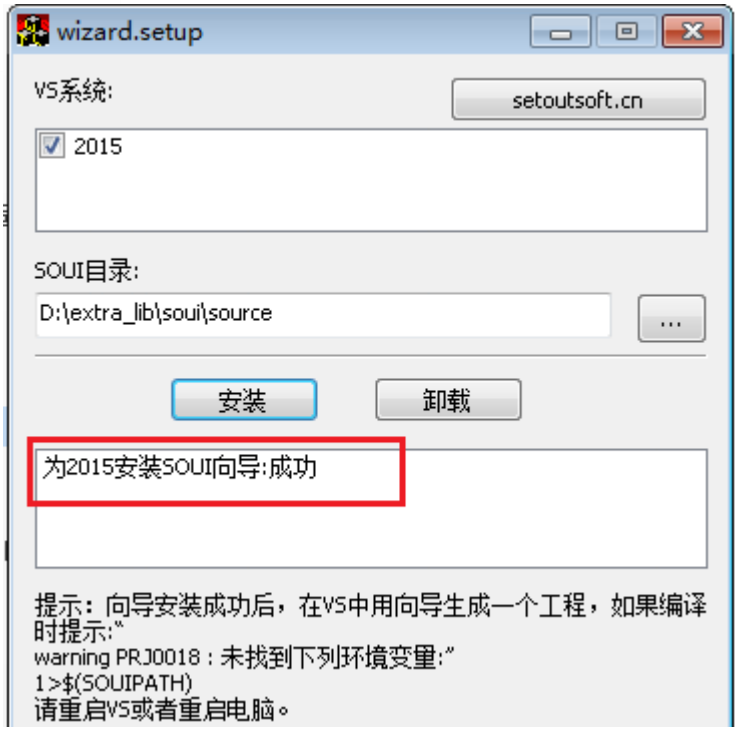
在 SOUI 项目下有一个 wizard 目录，运行目录下的 wizard.setup.exe 会显示下面界面：



如果系统安装了多个版本的 visual studio，这里将显示多个 vs；选择一个系统中安装的 VS 版本，点击安装就会向该版本中安装 SOUI 向导。

需要注意的是，关闭已打开的 vs，再进行安装 SOUI 向导，避免出现不必要的故障。

安装成功后显示：

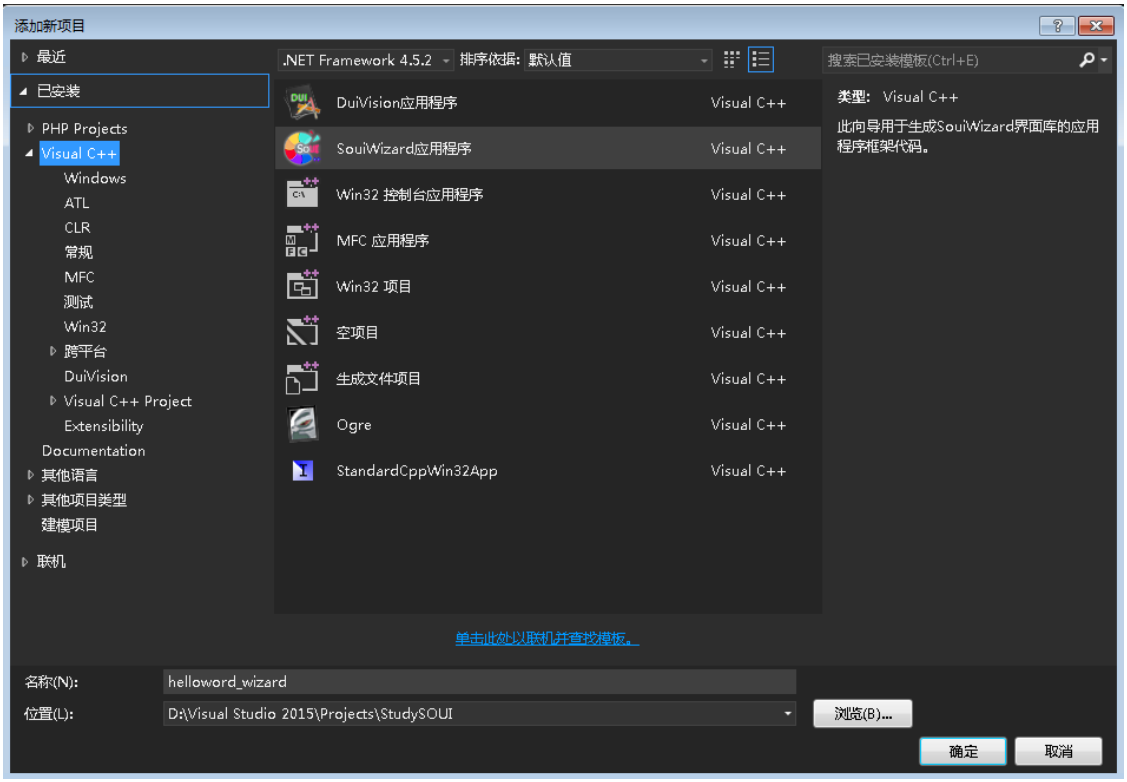


注：系统不同 vs 版本，wizard 会显示不同的 vs 版本；

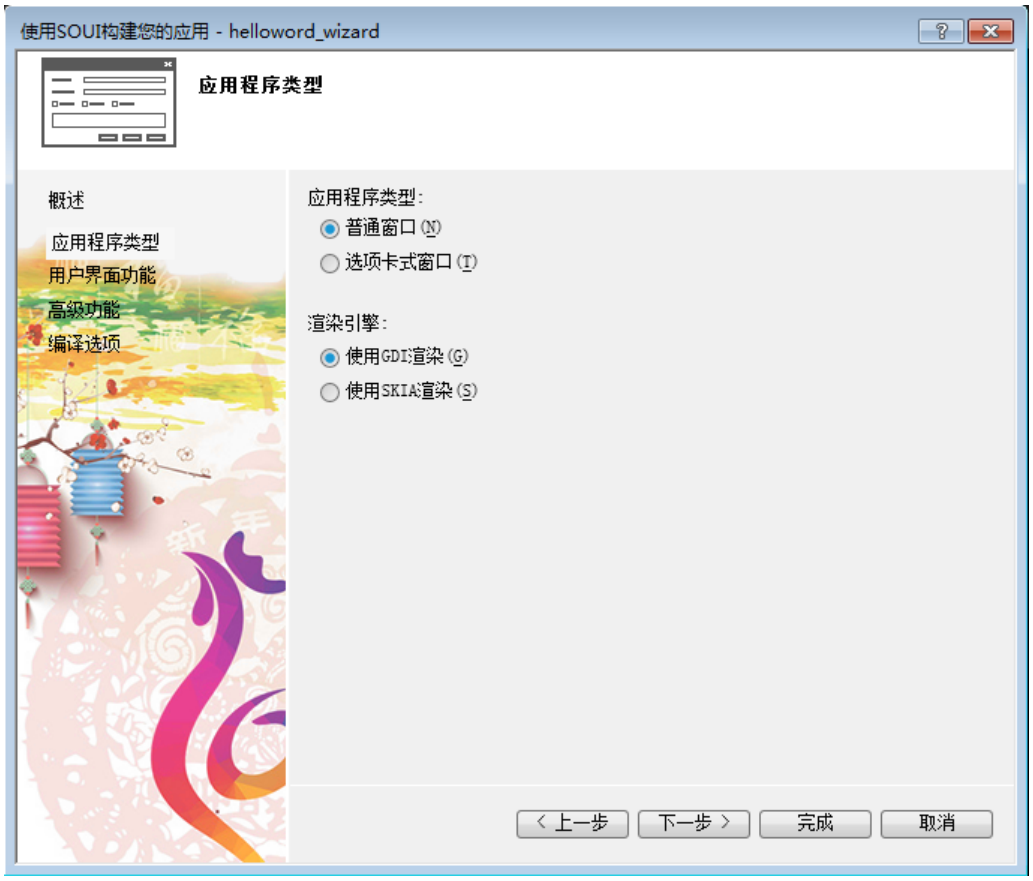
原来 DuiEngine 的向导安装程序运行后设置的环境变量经常要重启系统才生效，SOUI 向导已经经过优化，不需要重启系统。但已打开 VS 肯定是无新增加的环境变量的。

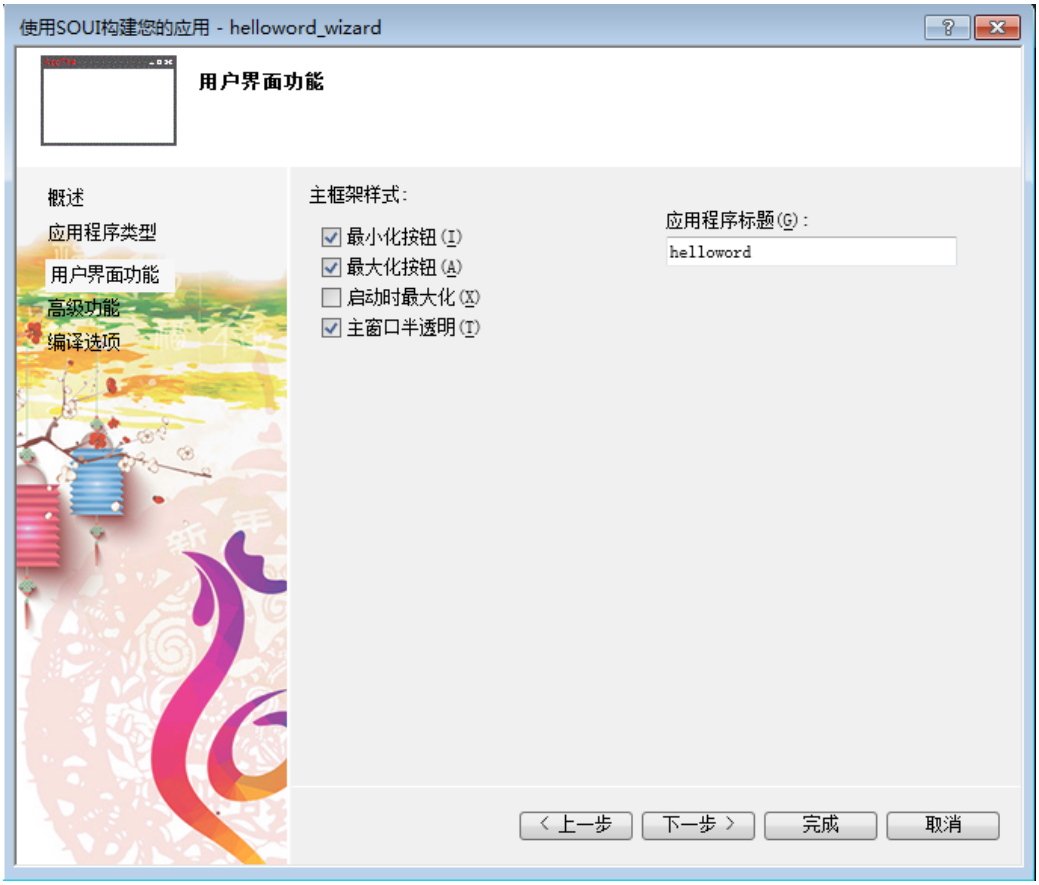
3.2.2 使用向导创建 SOUI 项目

打开 VS 的新建项目窗口，会在“Virtual C++”下找到 SOUI 应用程序向导。



输入项目名称后点击确定打开 SOUI 向导页（只有一页）





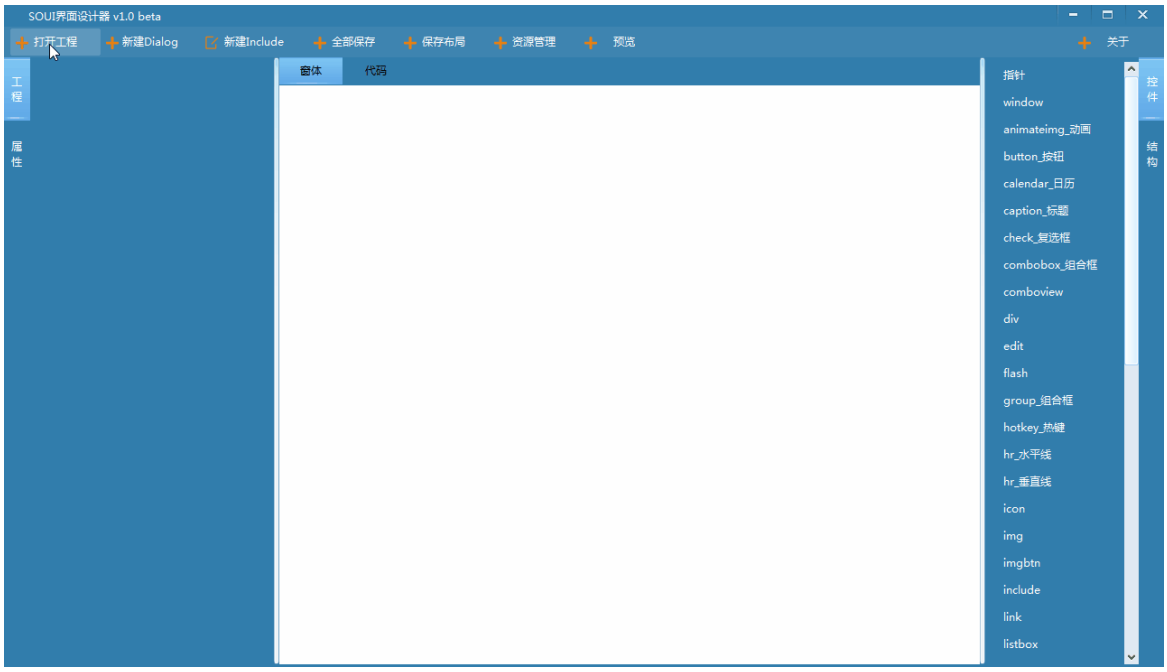
注：资源是否编译进 EXE 文件，可根据需要来选择！

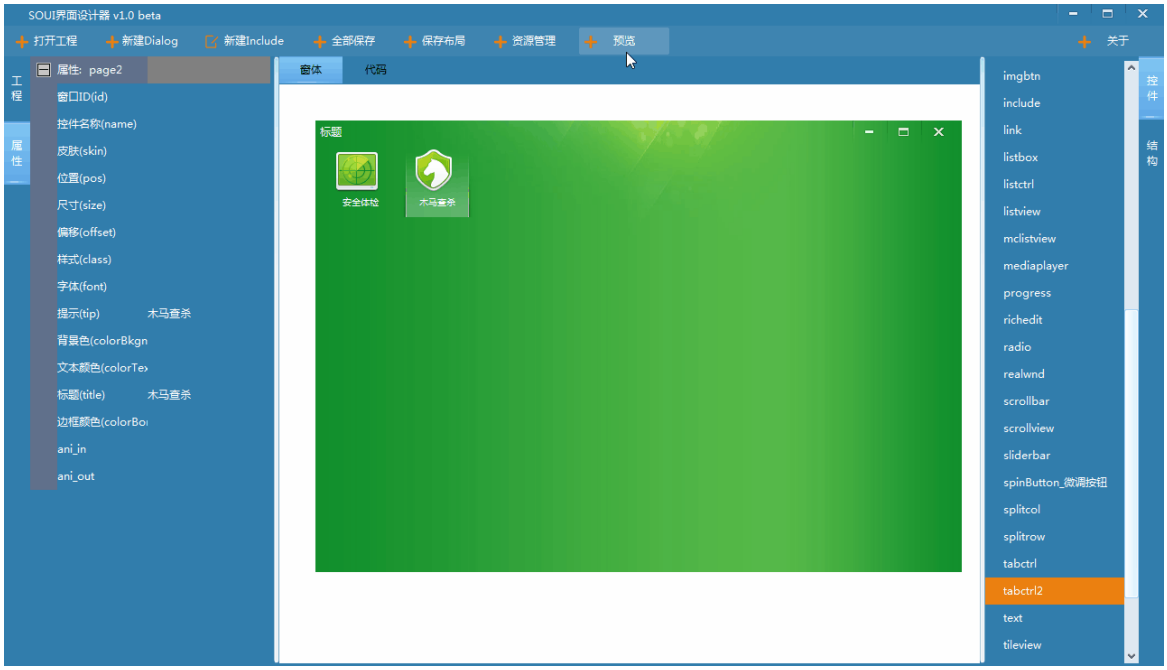
点击完成即可开始享受最简单高效的界面编程了。

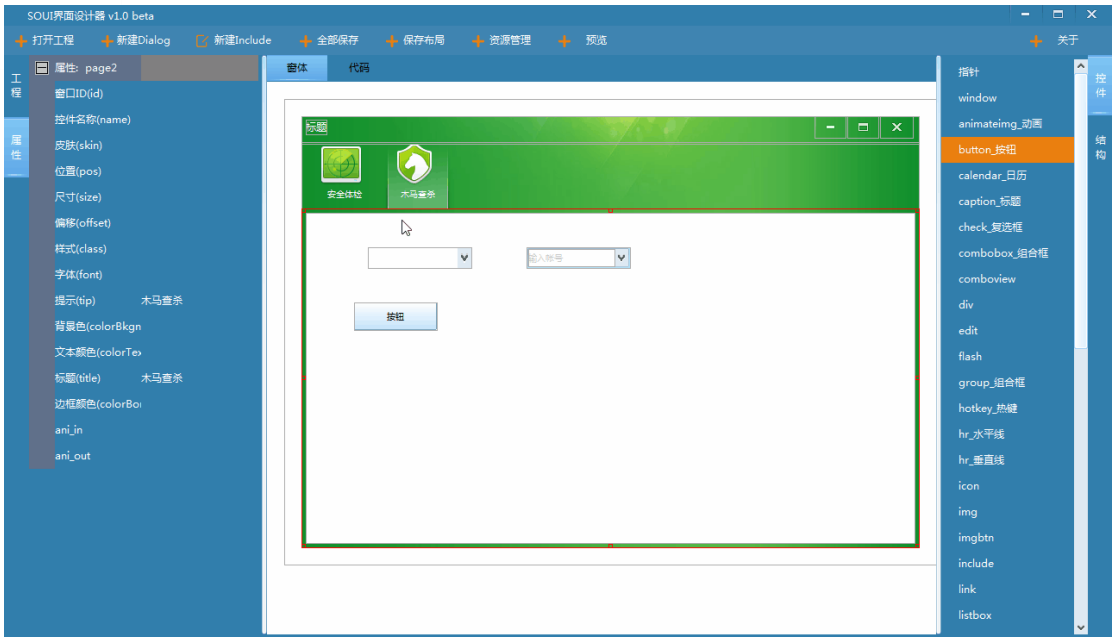
3.3 编辑器 SOUI Editor 使用教程

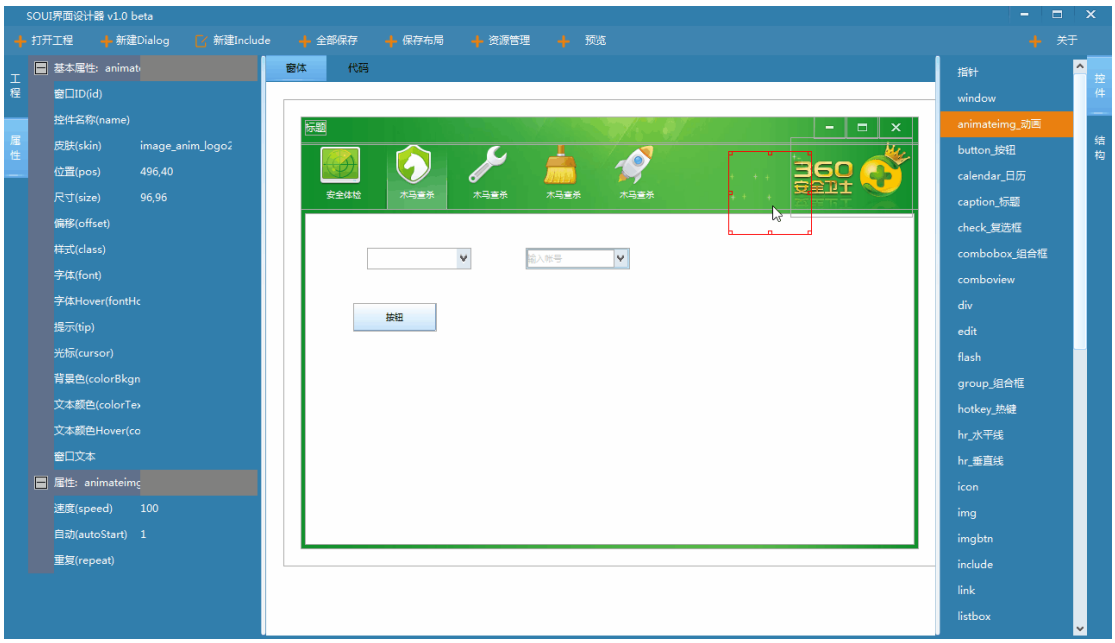
感谢网友"指尖"为 SOUI 开发的 UiEditor，目前该 UI 编辑器已经基本可用，源代码在 soui svn demos\uieditor。

下面是"指尖"提供的 GIF 动画教程。









大家快来膜拜"指尖", :)

3.4 使用 uiresImporter 生成 uires.idx 及 skin.xml

在 SOUI 中，使用 uires.idx 这个文件来记录程序中使用的所有资源文件。

此外绘制对象(ISkinObj)则一般放在 skin.xml 中描述。

要向一个界面中增加一个新的图片，在没有 uiresImporter 之前，首先我们需要把新的图片资源增复制到 uires 下的某个目录下，然后在 uires.idx 中加一条文件记录，然后在 skin.xml 中使用一个适当的 skin 类型(一般是 imglist,imgframe)来描述图片的显示方式，再在 UI 中引用该 skin 来绘制。

由于 SOUI 目前没有提供 UI 编辑器，所有的 XML 都需要手写，图片很多的时间文件导入是一个很麻烦又容易出错的工作。

根据前段时间一个网友制作的内部使用的 SOUI 辅助工具的思想，我开发了 uiresimporter 这个工具。uiresimporter.exe 位于 SOUI 的 tools 目录下，对应源代码在 tools\src\uiresimporter 里。

uiresimporter 的目标就是试图解决手动增加资源的麻烦。

和 uiresBuilder 一样，uiresimporter 也是一个命令行工具，它支持 5 个参数，见下面示例代码（demos\mclistview_demo\uiresimporter.bat）：

```
rem 使用 uiresImporter 来自动导入资源到 uires.idx 及 values\skin.xml.
rem -p 中指定 uires 目录。
rem -s 中指定需要在 uires.idx 中自动更新的文件夹。存在多个目录时应该使用"a|b|c"这样的形式分割，并使用引号。
rem -i 参数中指定的图片支持自动生成 skin，自动生成 skin 只支持 imglist,imgframe 两种，不支持的图片放到其它目录，如示例中的滚动条皮肤。
rem -b yes 自动备份原有 XML。no 不备份。
rem -c yes 皮肤默认支持着色处理，no 默认禁止着色。
%SOUIPATH%\tools\uiresImporter.exe -p uires -s "layout|icon|imgx" -i image -b yes -c no
```

为了自动导入图片，我们需要为图片的文件名做点修改：uiresimporter 通过文件名后的以[]包含的内容来识别图片显示格式。

可以有 3 种格式：

1、对于 imglist，只需要在[]中指定一个子图数量即可，如 btn_login[3].png，这样 uiresimporter 自动生成一个名字为 btn_login 的 imglist 对象，这个对象有 3 种状态。（当不指定[x]时，也生成一个 imglist 对象，状态数量为 1。

2、对于 imgframe，有一种完成的方式和一种缩略形式：

2.1 完全形式：bg_login[1{2,40,2,10}].png。这代表图片只有一个状态，它的九宫切分为 left:2,top:40,right:2,bottom:10。

2.2 缩略形式：bg_login[1{2,5}].png。当九宫的上下及左右大小相同时，可以使用缩略形式来命名。

2016-5-2 号版本新增加以下可选参数：

{ec=0/1} 是否支持皮肤着色(enableColorize)

{fit=0/1} 自适应绘图标志

{tile=0/1} 平铺标志

{filter=0/1/2/3} 插值滤镜类型, 0=null, 1=low, 2= midium, 3=high

{vert=0/1} 子图垂直排列标志。

在 imgframe 中，上述新标志必须在 margin 标志之后，否则 margin 标志将不能识别。

注：如果是需要将资源编译到 EXE，导入文件后记得使用 uiresbuilder 来重新生成.rc2 文件。

4、SOUI 开发说明

现代的软件只要有 UI，基本上少不了资源。

资源是什么？资源就是在程序运行时提供固定的数据源的文件。

在 MFC 当道的时代，资源一般就是位图(Bitmap)，图标(Icon)，光标(Cursor)，对话框模板(Dialog)等资源。

在 SOUI 中，资源主要变成了 XML 布局和 PNG 图片文件。

4.1 xml 资源文件定义

基于 SOUI 界面库的程序，存在一个 UI 资源索引文件(uires.idx)，此文件是一个标准 xml 文件，其默认位置是 uires 文件所在路径下的 uires\uires.idx。如果使用了 PE 资源或者 zip 压缩文件，来保存所有资源文件，则此文件的位置不会改变。

此文件中可以定义程序的全局配置、XML 文件、字体、图片、文字等资源，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resource>
  <UIDEF>
    <file name="xml_init" path="xml\init.xml"  />
  </UIDEF>
  <ICON>
    <file name="LOGO"  path="image\img_logo.ico" />
  </ICON>
  <CURSOR>
    <file name="ANI_ARROW"  path="image\021.ani" />
    <file name="CUR_TST" path="image\camera_capture.cur"/>
  </CURSOR>
  <LAYOUT>
    <file name="maindlg" path="xml\dlg_main.xml"  />
    <file name="menu_test" path="xml\menu_test.xml"  />
    <file name="page_layout" path="xml\page_layout.xml"  />
    <file name="page_treebox" path="xml\page_treebox.xml"  />
    <file name="page_treectrl" path="xml\page_treectrl.xml"  />
    <file name="page_misc" path="xml\page_misc.xml"  />
    <file name="page_webkit" path="xml\page_webkit.xml"  />
    <file name="page_about" path="xml\page_about.xml"  />
  </LAYOUT>
  <IMGX>
    <file name="png_page_icons"   path="image\page_icons.png" />
    <file name="png_small_icons"  path="image\small_icons.png" />

    <file name="webbtn_back"  path="image\webbtn_back.png" />
    <file name="webbtn_forward"  path="image\webbtn_forward.png" />
    <file name="webbtn_refresh"  path="image\webbtn_refresh.png" />
```

```
    <file name="png_treeicon"           path="image\TreeIcon.png"/>
    <file name="png_menu_border"  path="image\menuborder.png" />
    <file name="png_vscroll"  path="image\vscrollbar.png" />
    <file name="png_tab_left"  path="image\tab_left.png" />
    <file name="png_tab_left_splitter"  path="image\tab_left_splitter.png" />
    <file name="png_tab_main"  path="image\tab_main.png" />
    <file name="btn_menu" path="image\btn_menu.png"  />
  </IMGX>
  <GIF>
    <file name="gif_horse"    path="image\horse.gif"/>
    <file name="gif_penguin"    path="image\penguin.gif"/>
  </GIF>
  <rtf>
    <file name="rtf_test"    path="rtf\RTF 测试.rtf"/>
  </rtf>
  <script>
    <file name="lua_test" path="lua\test.lua"/>
  </script>
  <translator>
    <file name="lang_cn" path="translation files\lang_cn.xml"/>
  </translator>
</resource>
```

uires.idx 文件是 SOUI 资源的入口，定义了程序中使用的各种资源，以"resource"为根节点。在 soui-demo 的 uires.idx 中，定义了 UIDEF，ICON，CURSOR，LAYOUT，IMGX，GIF，rtf，script，translator 这些资源类型。

其实你还可以定义任意的资源类型，只要类型长度不超过 30 个字符。

每一个资源类型下都是一些 file 元素，每个 file 包含两个属性：name 及 path。

name 是 UI 布局的 XML 文件引用该资源的标识，而 path 则是该资源真实的文件存储路径。

除了 UIDEF 中定义的 xml_init 及 layout 中定义的布局 XML 外，其它资源都比较简单，就是提供一种从 name 映射文件的方式。

关于 XML 定义的大概顺序是：

1：资源引入

uires.idx (这里定义界面要使用的一些资源)

格式：

<资源类型>

Name = 资源自定义名称， path=资源路径

</资源类型>

2：定义全局变量

init.xml (这里定义全局变量并用资源初始化这些全局变量)

格式：

<变量类型>

Name=变量名称， src=资源类型：资源别名， 控件属性...

</变量类型>

3: 使用全局变量

dlg_main.xml(这里开始引用上面的变量)

4.1.1 init.xml 资源文件

UIDEF 资源（布局 XML 再单独开篇介绍）

其中 uidef 中定义的 init.xml 用来定义 SOUI 中使用的全局 UI 定义。

这个文件一般应该在 main 函数中被 SApplication 对象使用，如：

```
//...
//定义一个唯一的 SApplication 对象，SApplication 管理整个应用程序的资源
SApplication *theApp=new SApplication(pRenderFactory,hInstance);
//...
//加载全局资源描述 XML
theApp->Init(_T("xml_init"));
```

这里直接使用 uidef 中定义的 name 属性来初始化系统。SApplication::Init 默认从 uidef 段中去查找 xml_init 资源，当然也可以定义在其它资源类型如 xml 中，但是需要在 init 中显示指定资源类型。

下面我们打开 xml\init.xml 看一下里面的内容：

```
<?xml version="1.0" encoding="utf-8"?>
<UIDEF>
  <font face="微软雅黑" size="18"/>
  <string>
    <ver value="1.0"/>
  </string>
  <skin>
    <imglist name="skin_page_icons"    src="imgx:png_page_icons" states="9"/>
    <imglist name="skin_small_icons" src="imgx:png_small_icons" states="12"/>
    <imglist name="skin_tree_icon"       src="imgx:png_treeicon" states="3"/>
    <imglist name="skin_menuicon" src="imgx:png_small_icons" states="12"/>
```

```
    <border name="skin_menuborder" src="imgx:png_menu_border" left="2"
top="2" border="2,2,2,2" key="FF00FF" alpha="100"/>
    <imglist name="skin_webbtn_back" src="imgx:webbtn_back" states="4"/>
    <imglist name="skin_webbtn_forward" src="imgx:webbtn_forward"
states="4"/>
    <imglist name="skin_webbtn_refresh" src="imgx:webbtn_refresh"
states="4"/>
    <imglist name="skin_tab_left" src="imgx:png_tab_left" states="3"/>
    <imglist name="skin_tab_left_splitter" src="imgx:png_tab_left_splitter"/>
    <imglist name="skin_tab_main" src="imgx:png_tab_main" states="3"/>
    <imglist name="skin_btn_menu" src="imgx:btn_menu" states="3"/>
    <gif name="gif_horse" src="gif:gif_horse"/>
    <gif name="gif_penguin" src="gif:gif_penguin"/>
  </skin>
  <style>
    <class name="cls_dlg_frame" skin="_skin.sys.wnd.bkgnd" font=""
colorText="#000000" margin-x="0"/>
    <class name="cls_btn_link"    cursor="hand" colorHover="#0A84D2" />
    <!--定义文字按钮的样式-->
    <class name="cls_btn_weblink" cursor="hand" colorText="#1e78d5"
colorHover="#1e78d5" font="italic:1" fontHover="underline:1,italic:1" />
    <class name="cls_txt_red"     font="face:宋体,bold:1"  colorText="#FF0000"
/>
    <!--定义白色粗体宋体-->
    <class name="cls_txt_black"   font="face:宋体,bold:1"  colorText="#000000"
/>
    <!--定义黑色粗体宋体-->
    <class name="cls_txt_white"   font="face:宋体,bold:1"  colorText="#FFFFFF"
/>
    <!--定义白色粗体宋体-->
    <class name="normalbtn" font="" colorText="#385e8b"
colorTextDisable="#91a7c0" textMode="25" cursor="hand" margin-x="0"/>
    <class name="toptext" textMode="20" />
    <class name="vcentertext" textMode="24"/>
    <class name="rightvcentertext" textMode="26"/>
    <class name="centertext" textMode="25"/>
    <class name="righttext" textMode="22"/>
    <class name="linkimage" cursor="hand"/>
    <class name="cls_edit" ncSkin="_skin.sys.border" margin-x="2" margin-
y="2" />
  </style>
  <objattr>
    <button class="normalbtn"/>
```

```
    <imgbtn class="linkimage"/>
    <tabctrl colorText="000000" align="top" tabWidth="70" tabHeight="38"
tabSpacing="0" tabPos="10" dotted="1"/>
    <edit transParent="1" margin-x="2" margin-y="2"/>
    <treectrl colorItemBkgnd="#FFFFFF" colorItemSelBkgnd="#000088"
colorItemText="#000000" colorItemSelText="#FFFFFF" indent="17"
itemMargin="4"/>
  </objattr>
</UIDEF>
```

这个 xml_init 必须是以 UIDEF 为唯一根节点。

在 UIDEF 下，可以定义 font，string，skins，style，objattr 五个子节点。

其中，font 定义 SOUI 中使用的默认字体，只有 face 和 size 两个属性。

string 是一个字符串表，定义一个"name-字符串"映射，在布局的 XML 文件中可以通过引用字符串的 name 来获得字符串。

skins 定义 SOUI 中使用的全局窗口元素绘制对象，每一个对象都对应一个 SOUI::ISkinObj 的派生类。

SOUI 系统默认实现了 SSkinImgList(imglist), SSkinImgFrame(imgframe), SSkinButton(button), SSkinGradation(gradation), SSkinScrollbar(scrollbar), SSkinMenuBorder(border)这六种绘图类型。SSkinImgList 为 SOUI 中的 C++类名，imglist 为在 skins 节点中的元素类型名。

下面分别介绍这几种绘图类型：

1、imglist

imglist 是一个图片序列对象，可以包含一组小图片，常见的如按钮需要使用的 4 种状态图。

imglist 包含 4 个属性：

```
SOUI_ATTRS_BEGIN()
    ATTR_CUSTOM(L"src", OnAttrImage)
      //skinObj 引用的图片文件定义在 uires.idx 中的 name 属性。
    ATTR_INT(L"tile", m_bTile, TRUE)
      //绘制是否平铺,0--位伸（默认），其它--平铺
    ATTR_INT(L"vertical", m_bVertical, TRUE)
      //子图是否垂直排列，0--水平排列(默认)，其它--垂直排列
    ATTR_INT(L"states",m_nStates,TRUE)
      //子图数量,默认为 1
SOUI_ATTRS_END()
```

假定上图的图片在 uires.idx 中的定义为：

```
<imgx>
    <file name='btn_next' file='image\btn.next.png'/>
</imgx>
```

要在 soui 中引用这个图片，需要在 init.xml 的 skins 结节中做如下声明：

```
<skins>
   <imglist name="skin_btn_next" src="imgx:btn_next" states="4" tile="0"
vertical="0"/>
</skins>
```

在上面的 skin 定义中，

name 属性告诉系统如何引用定义的 imglist

src 属性定义该 skin 需要使用哪一个图片资源，资源引用格式为 type:name，如上面使用的 imgx:btn_next，对于图片资源，通常情况下也可以不指定 type,系统会自动在常用的图片类型下查找，但不建议这样使用。

states 定义图中包含多少个子图。

title 定义图片在放大显示时时平铺还是拉伸，默认为拉伸。

vertical 属性定义图中的子图的排列方式。

在本例子中 tile 和 vertical 属性都可以不指定。

2、imgframe

imgframe 是一个提供九宫格显示的绘图对象，SSkinImgFrame 派生自 SSkinImgList，因此 imgframe 也拥有 imglist 的全部属性。

此外，imgframe 提供了几个新的属性：

```
SOUI_ATTRS_BEGIN()
       ATTR_INT(L"left", m_rcMargin.left, TRUE)
         //九宫格左边距
       ATTR_INT(L"top", m_rcMargin.top, TRUE)
         //九宫格上边距
       ATTR_INT(L"right", m_rcMargin.right, TRUE)
         //九宫格右边距
       ATTR_INT(L"bottom", m_rcMargin.bottom, TRUE)
         //九宫格下边距
       ATTR_INT(L"margin-x", m_rcMargin.left=m_rcMargin.right, TRUE)
         //九宫格左右边距
       ATTR_INT(L"margin-y", m_rcMargin.top=m_rcMargin.bottom, TRUE)
         //九宫格上下边距
 SOUI_ATTRS_END()
```

imgframe 的格式如上图，在 imgframe 中通过 left, top, right, bottom 来定义九宫格。

3、button

button 绘图对象是绘制按钮时使用的，它使用渐变实现绘制按钮的 4 种状态。

包含以下属性：

```
SOUI_ATTRS_BEGIN()
        ATTR_COLOR(L"colorBorder", m_crBorder, TRUE)                //边框颜色
        ATTR_COLOR(L"colorUp", m_crUp[ST_NORMAL], TRUE)             //正常状态渐变起始颜色
        ATTR_COLOR(L"colorDown", m_crDown[ST_NORMAL], TRUE)         //正常状态渐变终止颜色
        ATTR_COLOR(L"colorUpHover", m_crUp[ST_HOVER], TRUE)         //浮动状态渐变起始颜色
        ATTR_COLOR(L"colorDownHover", m_crDown[ST_HOVER], TRUE)     //浮动状态渐变终止颜色
        ATTR_COLOR(L"colorUpPush", m_crUp[ST_PUSHDOWN], TRUE)       //下压状态渐变起始颜色
        ATTR_COLOR(L"colorDownPush", m_crDown[ST_PUSHDOWN], TRUE)   //下压状态渐变终止颜色
        ATTR_COLOR(L"colorUpDisable", m_crUp[ST_DISABLE], TRUE)     //禁用状态渐变起始颜色
        ATTR_COLOR(L"colorDownDisable", m_crDown[ST_DISABLE], TRUE) //禁用状态渐变终止颜色
SOUI_ATTRS_END()
```

4、gradation

渐变绘图对象，提供 3 个属性：

```
    SOUI_ATTRS_BEGIN()
        ATTR_COLOR(L"colorFrom", m_crFrom, TRUE)    //渐变起始颜色
        ATTR_COLOR(L"colorTo", m_crTo, TRUE)        //渐变终止颜色
        ATTR_INT(L"vertical", m_bVert, TRUE)        //渐变方向,0--水平(默认), 1--垂直
    SOUI_ATTRS_END()
```

5、scrollbar

滚动条皮肤，虽然它派生自 imglist，实际上 imglist 中实现的属性在 scrollbar 中没有意义，只是为了省点代码。

```
SOUI_ATTRS_BEGIN()
    ATTR_INT(L"margin",m_nMargin,FALSE)
      //边缘不拉伸大小
    ATTR_INT(L"hasGripper",m_bHasGripper,FALSE)
      //滑块上是否有帮手(gripper)
    ATTR_INT(L"hasInactive",m_bHasInactive,FALSE)
      //是否有禁用态
SOUI_ATTRS_END()
```

一般的 scrollbar 皮肤资源如下：

如果没有帮手也没有禁用状态，图片应该是 8*3 的正方形网格。

有帮手则 X 增加一个网格，有禁用状态则 Y 增加一个网格。

6、border

给 menu 用的，以后再介绍。

style

在 style 节点中，定义 UI 布局中 SOUI 窗口对象的属性集合，它们是 SWindow 对象的属性，所有 SWindow 对象都可以通过 class 属性来引用 style 节点中定义的属性集合。

objattr

控件的默认属性。

SOUI 可以为每一类 UI 控件通过 objattr 来提供一种默认属性集合，以减少在 XML 布局中的重复定义。

4.1.2 布局(layout)资源文件

窗口布局的概念

每一个 UI 都是由大量的界面元素构成的，在 Windows 编程，这些界面元素的最小单位通常称之为控件。

布局就是这些控件在主界面上的大小及相对位置。

传统的布局一般使用一个 4 个绝对坐标来定义一个控件在主窗口的位置。对于窗口是固定大小的界面来说，这种方式是最简单有效的。

然而问题在于在 Windows 系统上编程，基本上很少有程序的窗口是固定大小的，用户希望它的窗口能够随时调整大小。调整大小后界面里的控件还能够按照一定的规则进行重排。

我自己最讨厌的就是在 WM_SIZE 里重排控件位置。

随着使用 XML 来描述控件布局方式的出现，这种窗口布局过程中的窗口重排才得到了根本的解决。

两种 XML 布局类型

目前流行的 UI 库基本都是采用 XML 来描述窗口布局，如 Android, WPF, QQ 等。

使用 XML 布局整体上可以划分为两种类型：锚点布局和流式布局。

所谓流式布局就是一个控件只描述控件的大小，而不关心位置，它的最终显示位置由布局器计算出来，如 Android 及 DuiLib 里实现的 VerticalLayout 及 HorizontalLayout 等。

锚点布局和流式布局不同在于，它具体的定义一个控件的 4 个点的坐标位置，但这些位置通常不是绝对位置，而是一个相对于父窗口不同锚点的位置，当父窗口大小改变时，子窗口也会根据锚点的位置变化自动调整。布局的人使用锚点布局时很清楚一个控件最终会显示在哪，不需要很强的想象能力。

两种布局都有它们的优点，完善的 UI 布局器通常会同时提供两种布局能力。

考虑到锚点布局更直观，而且功能上完全可以解决布局需求，为了简化设计，在 SOUI 中只提供锚点布局。

SOUI 布局范例及解析

下面 XML 为 soui-demo 的主界面布局文件。

```
<SOUI name="dlg_main" title="SOUI-DEMO version:%ver%" bigIcon="LOGO:32"
smallIcon="LOGO:16" width="600" height="400" appWnd="1" margin="5,5,5,5"
resizable="1" translucent="1" >
  <skin>
    <!--局部 skin 对象-->
    <gif name="gif_horse" src="gif:gif_horse"/>
    <gif name="gif_penguin" src="gif:gif_penguin"/>
  </skin>
  <style>
    <!--局部 style 对象-->
    <class name="cls_edit" ncSkin="_skin.sys.border" margin-x="2" margin-
y="2" />
  </style>
  <root class="cls_dlg_frame" cache="1">
    <caption pos="0,0,-0,30" show="1" font="adding:8">
      <icon pos="10,8" src="LOGO:16"/>
      <text class="cls_txt_red">SOUI-DEMO version:%ver%</text>
```

```
    <imgbtn id="1" skin="_skin.sys.btn.close"   pos="-45,0" tip="close"
animate="1"/>
    <imgbtn id="2" skin="_skin.sys.btn.maximize"  pos="-83,0" animate="1" />
    <imgbtn id="3" skin="_skin.sys.btn.restore"  pos="-83,0" show="0"
animate="1" />
    <imgbtn id="5" skin="_skin.sys.btn.minimize" pos="-121,0" animate="1"
/>
    <imgbtn name="btn_menu" skin="skin_btn_menu" pos="-151,2" animate="1"
/>
  </caption>
  <tabctrl name="tab_main" pos="5,30,-5,-5" show="1" curSel="0"
focusable="0" animateSteps="10" tabHeight="75" tabSkin="skin_tab_main" text-
y="50" iconSkin="skin_page_icons" icon-x="10">
    <page title="listctrl123">
      <listctrl name="lc_test" pos="10,0,-10,-10" itemHeight="20"
headerHeight="30" cache="1" cursor="CUR_TST">
        <header align="left" itemSwapEnable="1" fixWidth="0" sortHeader="1">
          <items>
            <item width="150">name</item>
            <item width="150">gender</item>
            <item width="150">age</item>
            <item width="150">score</item>
          </items>
        </header>
      </listctrl>
    </page>
    <page title="webkit">
      <include src="layout:page_webkit"/>
    </page>
    <page title="flash">
      <flash pos="0,0,-0,-0" name="ctrl_flash"
url="http://swf.sc.chinaz.com//Files//DownLoad//flash2//201401//flash2524.sw
f" delay="1"/>
    </page>
    <page title="gif">
      <gifplayer pos="10,10" skin="gif_horse" name="giftest"
cursor="ANI_ARROW"/>
      <button width="250" height="30" name="btnSelectGif">load gif
file</button>
      <gifplayer pos="10,150" skin="gif_penguin"/>
      <icon pos="10,300" src="LOGO:64"/>
    </page>
    <page title="layout">
```

```
      <include src="layout:page_layout"/>
    </page>
    <page title="treebox">
      <include src="layout:page_treebox"/>
    </page>
    <page title ="misc.">
      <include src="layout:page_misc"/>
    </page>
    <page title="treectrl">
      <include src="layout:page_treectrl"/>
    </page>
    <page title="about">
      <include src="layout:page_about"/>
    </page>
  </tabctrl>
 </root>
</SOUI>
```

可以看到在这个 XML 中，有一个根节点：SOUI。在根节点中，定义了主界面的真窗口的各种属性（属性的含义见后续篇）。

在根节点下有 3 个节点，分别是 skin, style 及 root。

skin, 和 style 和前一篇讲的 init.xml 的功能一样。不同在于在布局文件中定义的 skin 及 style 只在当前窗口的生命周期期间有效，类似于 C++函数中的局部变量，窗口关闭后这些对象会自动析构。我称之为局部 skin 及局部 style。

窗口中控件的布局信息定义在 root 节点中。

root 节点本身也是一个 SWindow 窗口对象，但是在这里必须是"root"才能识别，在这个节点中可以有 SWindow 的各种属性，但是和布局位置相关的属性自动无效，因为该窗口总是充满整个宿主窗口。

在 root 节点下可以按照不同的布局层次采用锚点布局方式布局各种系统内置控件及用户自定义控件。

在 demo 中，我首先在最上面布局一个 caption 控件，caption 控件里又有各种标题，按钮等子控件。

然后在下面布局一个 tabctrl 以及它的子控件。

在这个布局 XML 中有大量的控件属性定义。不同的控件有不同的属性，这里不详细展开，这里主要关注一下 page 节点下的 include 节点。

include 只有一个属性：src，src 定义如何去引用在另一个 XML 文件中定义的布局 XML，如“layout:page_layout”代表这里要引用在 layout 资源类型中定义的 name 为 page_layout 的 XML 文件（关于资源的定义参考第四篇）。

下面是 layout:page_layout 指向的 XML 文件的内容：

```
<include>
  <text pos="100,10" pos2type="center">center align1</text>
  <text pos="100,30" pos2type="center">center align align</text>
  <text pos="250,50" pos2type="rightTop">align right top</text>
  <text pos="250,70" pos2type="rightTop">align right top 2</text>
  <check pos="250,90" pos2type="rightTop">check right top</check>
  <check pos="250,110" pos2type="rightTop" font="adding:-5">check right
top1235</check>

  <text pos="250,130" class="cls_txt_red">text left top</text>
  <button pos="10,150,@150,@30">button 1 using @</button>
  <button pos="10,200" width="150" height="30">button 1 using width</button>

  <button name="btn_hidetst" pos="300,150,@100,@30" display="0" tip="click me
to hide me and see how the next image will move">hide test</button>
  <img skin="skin_page_icons" iconIndex="1" pos="[5,150,-10,-10" />
</include>
```

可以看到在这个文件中，有一个以"include"的根节点，在 include 节点下才是布局 XML。

这里的 include 代表该文件只能是被其它的有 include 元素的布局文件引用。

需要注意的是，在 include 的 XML 文件中不能定义局部 skin 及局部 style。

SOUI 的锚点布局

SOUI 布局全部采用相对坐标，由 pos,offset(pos2type), size, width,height 这几个个窗口属性配合指定。

size, width, height 属性

size, width, height 比较简单，是用来指定窗口的大小的，只有在 pos 属性指定的值个数不为 4 时生效。

size 是 2014 年底增加的布局属性，size="width,height"。

width, height 可以有 3 种值：full,-1,非负整数。

为 full 时，代表高度或者宽度和父窗口的客户区大小相等。

-1 代表根据窗口内容自动计算窗口大小。

非负整数直接指定窗口大小。

在图片控件中，控件是指定的皮肤默认大小。

在文本控件中，还可以指定一个 maxWidth 属性，控件是文本内容的大小，但宽度不超过 maxWidth。

pos 属性

pos 属性可以指定 4 个值，也可以指定 2 个值。指定 4 个值时，分别代表控件的 left,top,right,bottom,指定两个值时代表控件的 x,y，具体位置还依赖于另外 3 个参数。

指定 4 个值时，pos 目前支持 7 种标志：|,%,[,],{,},@

“|”代表参考父窗口的中心；如|-10 代表在父窗口的中心向左/上偏移 10 象素。

“%”代表在父窗口的百分比，可以是小数，负数。如：%40 代表在父窗口的 40%位置，%-40 则等价于(1-40%)。

“[”相对于前一兄弟窗口。用于 X 时，参考前一兄弟窗口的 right，用于 Y 时参考前一兄弟窗口的 bottom

“]”相对于后一兄弟窗口。用于 X 时，参考后一兄弟的 left,用于 Y 时参考后一兄弟的 top

“{”相对于前一兄弟窗口。用于 X 时，参考前一兄弟窗口的 left，用于 Y 时参考前一兄弟窗口的 top

“}”相对于后一兄弟窗口。用于 X 时，参考后一兄弟的 right,用于 Y 时参考后一兄弟的 bottom

“@”标志用来指定窗口的大小，只能出现在 pos 属性的第 3，4 个值中，用来标识窗口的宽度。当后面的值为负时，代表自动计算窗口的宽度或者高度（2015.3.3 新增加解释）。

注： “| “, "[" ,"]", "{", "}" 中指定的值都可以为正或者负，正时向右或者下偏移，负则向左或者上偏移。

当没有上述标志时，负号代表参考父窗口的右边或者下边缩进绝对值位置。如：pos="0,0,-0,-0"代表占满父窗口。而 pos="10,10,-10,-10"则代表在父窗口的基础上向内全部缩进 10 点。

@:指定窗口的 size。只能用于 x2,y2，用于 x2 时，指定窗口的 width，用于 y2 时指定窗口的 height。注：只能为正值，负号会自动忽略。

其中“{”和“}”是 SOUI 在 DUIENGINE 的基础上新增加的布局标志（SOUI 是在 DUIENGINE 的基础上全面重构而来）。

注意!!!由于系统运行向前及向后引用，理论上有可能出来循环引用，导致界面布局失败，因此在使用"["，"{"， “}” 和"]"这几个标志时需要特别注意。

当 pos 只指定了 x1,y1 时，通常需要和 offset(或者 pos2type),size(或者 width,height)配合使用。

offset 及 pos2type 属性

offset 属性包含两个值，用来代表窗口在通过其它布局属性完成后的偏移量：如 offset="-1,-1"，该 offset 表明窗口向左方及上方各平衡一个窗口大小的单位。

offset 及 pos2type 属性具体请参考**《第十六篇：SWindow 的布局属性 pos2type 及 offset》**

在 SOUI 的布局系统中，使用一个 pos 属性基本可以完整 90%以上的布局功能，建议用户在 demo 中修改各种布局属性来观察控件位置的变化以加深对 SOUI 布局系统的理解。

4.1.3 布局属性 pos2type 及 offset

当窗口大小需要根据内容来确定时，使用 XML 布局可能需要做一些特殊的处理。

例如：不管窗口多大，我需要将该窗口相对于父窗口居中在 XML 中应该怎么处理？

如果窗口大小是固定的（如, 100 *100)，这样 pos 属性可以定义为"|-50,|-50,|-50,|-50"即可。

当窗口大小不确定时，SOUI 中提供了 pos2type 及 offset 来协同处理。

其中 pos2type 是 offset 的子集。

下面先重点介绍 offset 属性

offset 属性是 SOUI 在通过 pos 属性完成坐标定位后再将坐标进行偏移的属性。和 pos 中一般使用象素为单位不同，offset 是以控件最后的大小为单位进行平移。

我们可以在 XML 中或者代码中使用 offset = "-0.5,-0.5"这样的形式来描述窗口的坐标平移属性。

属性中包含两个值，分别对应 X，Y 方向的平移相对于窗口大小的倍数，一般为[-1,0]的小数(float)，当然也可以超过这个范围。

我们先看一下代码中如何实现：

```
class SOUI_EXP SwndLayout
{
public:
    //...
    float fOffsetX,fOffsetY;  /**< 窗口坐标偏移量, x += fOffsetX *
    //...
};
```

```
int SwndLayout::CalcPosition(LPRECT lpRcContainer,CRect &rcWindow )
{
    int nRet=0;
   //...
    if(nRet==0)
    {//没有坐标等待计算了
        rcWindow.NormalizeRect();
        //处理窗口的偏移(offset)属性
        CSize sz = rcWindow.Size();
        CPoint ptOffset;
        ptOffset.x = (LONG)(sz.cx * fOffsetX);
        ptOffset.y = (LONG)(sz.cy * fOffsetY);
        rcWindow.OffsetRect(ptOffset);
    }
    return nRet;
}
```

SwndLayout::CalcPosition 是 SOUI 用来通过 pos 及 offset 属性计算窗口坐标的关键函数，为了突出重点，具体的坐标计算省略了，只列出平移处理部分的代码。

可以看出，在平移处理前，首先获得窗口的 Size,再将 Size 分别乘以 fOffsetX,fOffsetY 这两个平移系数获得在 x,y 两个方向上的平移量。

最后才是将矩形做平移处理。

下面我们再来看看 pos2type 属性：

pos2type 可以定义 9 个参考点：center, lefttop, leftmid, leftbottom,midtop,midbottom,righttop,rightmid,rightbottom。

下表显示对应原 pos2type 属性的 offset 属性：

pos2type	offset
center	-0.5,-0.5
lefttop	0,0
leftmid	0,-0.5
leftbottom	0,-1
midtop	-0.5,0
midbottom	-0.5,-1
righttop	-1,0
rightmid	-1,-0.5
rightbottom	-1,-1

从上表可以看出，原来的 pos2type 属性只能是 0.5 的倍数，新的 offset 属性没有该限制。

使用 pos2type 可能更为直观，但是 offset 属性则更灵活。如果两个属性同时使用，只有最后一个属性有效。

注意：offset 属性是 2014.11.20 才新增加的属性，pos2type 属性的命名是为了兼容 2014.11.20 前的版本。

SOUI 布局之相对于特定兄弟窗口

SOUI 中通过 pos 的标志如：[, {, },]，这 4 个标志可以相对于前一个及后一个兄弟窗口，但是有时候希望相对于不是前后窗口的兄弟窗口，比如一个通过一个中心窗口同时定义它的上下左右 4 个窗口，这个时候应该如何处理？

其实 SOUI 是支持相对于任意一个兄弟窗口的，但是定义方法有点复杂，所以在之前的博客文章中都没有介绍。

定义的方法是这样的：

首先被参考窗口（假定为窗口 A）必须要指定窗口的 ID 属性，有了 ID（假定 id=100），其它窗口才能引用它（这里指定 name 属性是不行的，系统只会通过 ID 去查询这个兄弟窗口）。

然后一个窗口（假定为窗口 B）要相对于窗口 A 布局，只需要在 pos 中指定为如：pos="sib.left@100:-20,sib.bottom@100:30,@100,@100"，坐标定义中的 sib.left,sib.bottom 用来指定这两个坐标是相对于被引用窗口的 left,bottom 的值，坐标中的 100:20,100:30 刚代表相对于 ID 为 100 的兄弟窗口的 left 向左偏移 20 像素及 bottom 向下偏移 30 像素。这里的负数是代表偏移方向，和没有 sib.xxx 时的负值意义不同。

下面看下 demo 中的示例 XML(demo/uires/xml/page_layout.xml)：

```
    <window skin="skin_page_icons" pos="[5,150,-10,-10" id="1236">
      <text pos="|0,|0" offset="-0.5,-0.5" font="adding:20"
colorText="#ff000066">alpha test</text>
      <text pos="5,5" id="100" visible="0">ref text</text>
      <button pos="sib.left@100:10,sib.bottom@100:10,@100,@25"
name="btn_hidetst" tip="click me to hide me and see how the next image will
move">ref id:100</button>
    </window>
```

PS：这个定义方法有点山寨，将就着用吧，关键是能解决问题 :)

4.1.4 在 SOUI 中使用线性布局

SOUI 2.5.1.1 开始支持线性布局(LinearLayout).

要在 SOUI 布局中使用线性布局，需要在布局容器窗口里指定布局类型为 vbox | hbox, (vbox 为垂直线性布局, hbox 为水平线性布局).

在指定布局类型后还可以为容器窗口指定 gravity 属性，用来指定子窗口的默认排列模式.

vbox 的 gravity 有:left(默认), center, right, hbox 有: top(默认), center, bottom.

在线性布局中的子窗口 pos 属性没有意义，一般直接指定 size="width,height", width/height 值: -1 代表 wrap_content, -2 代表 match_parent

可以使用 layout_gravity 可以更改当前窗口的排列模式.

使用 extend="left,top,right,bottom", extend_left, extend_top, extend_right, extend_bottom 来指定间距. (相当于 android 的 margin)

子窗口支持使用 weight 属性.

看下面 demo 中的例子:

```
<page title="linear layout">
    <!--这里演示在 SOUI 中使用线性布局,在 window 中指定
layout="vbox,hbox,linearLayout"时窗口的子窗口布局变成自动布局模式-->
```

```
    <window layout="vbox" size="-1,-1" colorBkgnd="#cccccc"
gravity="center">
      <!--线性布局的自适应子窗口大小-->
      <text>vbox + gravity + wrapContent</text>
      <window size="100,30" colorBkgnd="#ff0000"/>
      <window size="200,30" extend="10,5,10,5" colorBkgnd="#ff0000"/>
      <window size="120,30" layout_gravity="right" colorBkgnd="#ff0000"/>
    </window>
    <window pos="0,[5,@-1,@200" layout="vbox" colorBkgnd="#cccccc">
      <!--线性布局的 weight 属性-->
      <text extend_bottom="10">vbox + gravity + weight</text>
      <window size="100,30" colorBkgnd="#ff0000"/>
      <window size="200,30" extend="10,5,10,5" colorBkgnd="#ff0000"
weight="1"/>
      <window size="120,30" layout_gravity="right" colorBkgnd="#ff0000"
weight="1"/>
      <button size="100,30" extend_top="10">button test</button>
    </window>
    <window pos="0,[5" layout="vbox" colorBkgnd="#cccccc" id="10000">
      <text extend_bottom="10" layout_gravity="center">hbox demo</text>
      <window size="-1,-1" layout="hbox" colorBkgnd="#888888">
        <!--线性布局之 hbox-->
        <button size="100,30">button1</button>
        <button size="100,30" extend_left="10">button2</button>
        <button size="100,30" extend_left="10">button3</button>
        <button size="100,30" extend_left="10">button4</button>
      </window>
    </window>
  </page>
```

4.2 系统资源管理

上一节，我们已经讲到在 SOUI 中所有资源文件通过一个 uires.idx 文件进行索引。

这里将介绍在程序中如何引用这些资源文件。

在 SOUI 系统中，资源文件通过一个统一的接口对象读取：

```
namespace SOUI
{
    enum BUILTIN_RESTYPE
    {
        RES_PE=0,
        RES_FILE,
    };
```

```
/**
* @struct    IResProvider
* @brief     ResProvider 对象
*
* Describe  实现各种资源的加载
*/
struct IResProvider : public IObjRef
{
    /**
     * Init
     * @brief    资源初始化函数
     * @param    WPARAM wParam --  param 1
     * @param    LPARAM lParam --  param 2
     * @return   BOOL -- true:succeed
     *
     * Describe  every Resprovider must implement this interface.
     */
    virtual BOOL Init(WPARAM wParam,LPARAM lParam) =0;

    /**
     * HasResource
     * @brief    查询一个资源是否存在
     * @param    LPCTSTR strType --  资源类型
     * @param    LPCTSTR pszResName --  资源名称
     * @return   BOOL -- true 存在，false 不存在
     * Describe
     */
    virtual BOOL HasResource(LPCTSTR strType,LPCTSTR pszResName)=0;

    /**
     * LoadIcon
     * @brief    从资源中加载 ICON
     * @param    LPCTSTR pszResName --  ICON 名称
     * @param    int cx --  ICON 宽度
     * @param    int cy --  ICON 高度
     * @return   HICON -- 成功返回 ICON 的句柄，失败返回 0
     * Describe
     */
    virtual HICON   LoadIcon(LPCTSTR pszResName,int cx=0,int cy=0)=0;

    /**
     * LoadBitmap
```

```
        * @brief    从资源中加载 HBITMAP
        * @param    LPCTSTR pszResName --  BITMAP 名称
        * @return   HBITMAP -- 成功返回 BITMAP 的句柄，失败返回 0
        * Describe
        */
       virtual HBITMAP    LoadBitmap(LPCTSTR pszResName)=0;

       /**
        * LoadCursor
        * @brief    从资源中加载光标
        * @param    LPCTSTR pszResName --  光标名
        * @return   HCURSOR -- 成功返回光标的句柄，失败返回 0
        * Describe  支持动画光标
        */
       virtual HCURSOR LoadCursor(LPCTSTR pszResName)=0;

       /**
        * LoadImage
        * @brief    从资源加载一个 IBitmap 对象
        * @param    LPCTSTR strType --  图片类型
        * @param    LPCTSTR pszResName --  图片名
        * @return   IBitmap * -- 成功返回一个 IBitmap 对象，失败返回 0
        * Describe  如果没有定义 strType，则根据 name 使用 FindImageType 自动查找匹配
的类型
        */
       virtual IBitmap * LoadImage(LPCTSTR strType,LPCTSTR pszResName)=0;

       /**
        * LoadImgX
        * @brief    从资源中创建一个 IImgX 对象
        * @param    LPCTSTR strType --  图片类型
        * @param    LPCTSTR pszResName --  图片名
        * @return   IImgX   * -- 成功返回一个 IImgX 对象，失败返回 0
        * Describe
        */
       virtual IImgX   * LoadImgX(LPCTSTR strType,LPCTSTR pszResName)=0;

       /**
        * GetRawBufferSize
        * @brief    获得资源数据大小
        * @param    LPCTSTR strType --  资源类型
        * @param    LPCTSTR pszResName --  资源名
```

```
        * @return   size_t -- 资源大小（byte），失败返回 0
        * Describe
        */
       virtual size_t GetRawBufferSize(LPCTSTR strType,LPCTSTR pszResName)=0;

       /**
        * GetRawBuffer
        * @brief    获得资源内存块
        * @param    LPCTSTR strType --  资源类型
        * @param    LPCTSTR pszResName --  资源名
        * @param    LPVOID pBuf --  输出内存块
        * @param    size_t size --  内存大小
        * @return   BOOL -- true 成功
        * Describe  应该先用 GetRawBufferSize 查询资源大小再分配足够空间
        */
       virtual BOOL GetRawBuffer(LPCTSTR strType,LPCTSTR pszResName,LPVOID
pBuf,size_t size)=0;

       /**
        * FindImageType
        * @brief    查询与指定名称匹配的资源类型
        * @param    LPCTSTR pszImgName --  资源名称
        * @return   LPCTSTR -- 资源类型，失败返回 NULL
        * Describe  没有指定图片类型时默认从这些类别中查找
        */
       virtual LPCTSTR FindImageType(LPCTSTR pszImgName) =0;
   };

   /**
   * Helper_FindImageType
   * @brief    查询与指定名称匹配的资源类型
   * @param    IResProvider * pResProvider --  当前的 ResProvider
   * @param    LPCTSTR pszImgName --  资源名称
   * @return   LPCTSTR -- 资源类型，失败返回 NULL
   * Describe  提供一个公共的辅助函数
   */
}//namespace SOUI
```

这个接口的实现类通过实现这些既定接口来完成图标(HICON)，光标(HCURSOR)，位图(HBITMAP)，一般图片(IBitmap)的解码，同时也提供原始数据(RawData)的读取。

在 SOUI 系统中内置了两种类型的资源加载(ResProvider）模块：SResProviderPE 和 SResProviderFiles，同时也通过外置组件的形式提供了从 ZIP 文件加载资源的功能。

这三种资源加载方式基本上涵盖了目前常见的资源加载方式。

为了能够从 PE 的资源数据段中加载资源，我们需要将 uires.idx 中索引的文件转换成 PE 资源可以识别的资源类型+资源名（不是资源 ID）的形式。

为了达到这个目的，我们只需要在 VS 的资源文件中（.rc）将 SOUI 的资源中定义的文件按照 uires.idx 定义的类型和名称加进去即可。

手工添加资源文件很难保证不写错。为此，我提供了一个工具（tools\uiresbuilder.exe)，这个工具接收一组命令参数，用来将 uires.idx 转换成一个 RC 编译器可以识别的.rc2 文件（命令行参见使用向导生成的工程）。要编译该.rc2 文件，需要在.rc 的资源包含中加上我们生成的.rc2 文件。

如果程序中的资源不从 PE 资源加载，则不需要编译 soui_res.rc2 文件，以减少程序体积。

SResProviderPE, SResProviderFiles 和 SResProviderZIP 分别从 PE 资源，文件夹及 ZIP 文件包中初始化资源：

```
#if (RES_TYPE == 0)//从文件加载
       CreateResProvider(RES_FILE,(IObjRef**)&pResProvider);
       if(!pResProvider->Init((LPARAM)_T("uires"),0))
       {
           SASSERT(0);
           return 1;
       }
#elif (RES_TYPE==1)//从 EXE 资源加载
       CreateResProvider(RES_PE,(IObjRef**)&pResProvider);
       pResProvider->Init((WPARAM)hInstance,0);
#elif (RES_TYPE==2)//从 ZIP 包加载
       bLoaded=pComMgr->CreateResProvider_ZIP((IObjRef**)&pResProvider);
       SASSERT_FMT(bLoaded,_T("load interface [%s]
failed!"),_T("resprovider_zip"));

       ZIPRES_PARAM param;
       param.ZipFile(pRenderFactory, _T("uires.zip"),"souizip");
       bLoaded = pResProvider->Init((WPARAM)&param,0);
       SASSERT(bLoaded);
#endif
```

资源加载成功后，调用 SApplication::AddResProvider(IResProvider *)接口将创建的资源加载器交给 SOUI 系统管理。

SApplication::AddResProvider 可以调用多次，便于加载不同的资源。

资源加载后不再需要了也可以使用 SApplication::RemoveResProvider(IResProvider *)来删除。

程序中要使用一个资源时，首先调用 SApplication::HasResource 来查询一个资源是否存在，然后再根据资源类型选择不同的接口加载资源。

SApplication 中管理着一个 IResProvider 列表，系统采用后进先查的算法处理资源重名，即最后调用 AddResProvider 加进来的资源加载器优先级最高。

4.3 应用程序中资源的组织

一个基于 SOUI 开发的应用程序通常资源分为两部分：

- 控件默认的系统资源，也可以理解为主题(theme)资源。
- 应用程序自定义的资源。

4.3.1 控件默认的系统资源

系统资源又分为 3 部分：

■ 控件默认的皮肤(Skin)：

SOUI 中很多控件都必须要定义一个皮肤(ISkinObj,称之为绘图对象更好解理)，例如绘制 checkbox，radiobox，combobox 等控件出出现的位图部分。如何在应用程序中使用了这些控件，则必须为它们定义这些皮肤资源。为了简化界面配置，我统一将这些资源进行命名，并打包到一起（参见 trunk\soui-sys-resource\theme_sys_res\sys_xml_skin.xml）。

下面是系统中使用定义的命名 ISkinObj:

```
const wchar_t * BUILDIN_SKIN_NAMES[]=
{
    L"_skin.sys.checkbox",
    L"_skin.sys.radio",
    L"_skin.sys.focuscheckbox",
    L"_skin.sys.focusradio",
    L"_skin.sys.btn.normal",
    L"_skin.sys.scrollbar",
    L"_skin.sys.border",
    L"_skin.sys.dropbtn",
    L"_skin.sys.tree.toggle",
    L"_skin.sys.tree.checkbox",
    L"_skin.sys.tab.page",
    L"_skin.sys.header",
    L"_skin.sys.split.vert",
    L"_skin.sys.split.horz",
    L"_skin.sys.prog.bkgnd",
    L"_skin.sys.prog.bar",
    L"_skin.sys.slider.thumb",
    L"_skin.sys.btn.close",
    L"_skin.sys.btn.minimize",
    L"_skin.sys.btn.maxmize",
    L"_skin.sys.btn.restore",
    L"_skin.sys.menu.check",
    L"_skin.sys.menu.sep",
    L"_skin.sys.menu.border",
    L"_skin.sys.menu.skin",
    L"_skin.sys.icons",
    L"_skin.sys.wnd.bkgnd"
};
```

如果程序中没有使用到特定控件，也可以不在系统资源中提供对应的 ISkinObj。

■ 系统使用的 MsgBox 布局模板：

MsgBox 是应用程序常用的功能。SOUI 通过提供一个 MSGBOX 的 XML 布局模板来实现。如果需要修改 MsgBox 的样式，只需要修改这个 XML。

```
<SOUI title="mesagebox" width="200" height="100" appwin="0"
frameSize="40,30,10,80" minSize="300,100" resize="0" translucent="1"
trCtx="messagebox">
  <style>
    <class name="normalbtn" skin="_skin.sys.btn.normal" font=""
colorText="#385e8b" colorTextDisable="#91a7c0" textMode="25" cursor="hand"
margin-x="0"/>
  </style>
  <root skin="_skin.sys.wnd.bkgnd">
    <caption id="101" pos="0,0,-0,29">
      <text pos="11,9" class="cls_txt_red" name="msgtitle" >title</text>
      <imgbtn id="2" skin="_skin.sys.btn.close"    pos="-45,0" tip="close"/>
    </caption>

    <window pos="5,30,-5,-50">
      <icon name="msgicon" pos="0,0,32,32" display="0"/>
      <text name="msgtext" pos="[0,0" colorText="#0000FF" multilines="1"
maxWidth="300"/>
    </window>
    <tabctrl name="btnSwitch" pos="0,-50,-0,-0" tabHeight="0">
      <page>
        <button pos="|-50,10,|50,-10" name="button1st"
class="normalbtn">button1</button>
      </page>
      <page>
        <button pos="|-100,10,|-10,-10" name="button1st"
class="normalbtn">button1</button>
        <button pos="|10,10,|100,-10" name="button2nd"
class="normalbtn">button2</button>
      </page>
      <page>
        <button pos="|-140,10,|-50,-10" name="button1st"
class="normalbtn">button1</button>
        <button pos="|-45,10,|45,-10" name="button2nd"
class="normalbtn">button2</button>
        <button pos="|50,10,|140,-10" name="button3rd"
class="normalbtn">button3</button>
      </page>
    </tabctrl>
```

```
  </root>
</SOUI>
```

上面是 SOUI 默认提供的模板。

在这个模板中，只提供了必须的命名对象。如果要修改这个模板，这些命名对象不能缺少，不过布局位置可以任意调整。

■ Edit 控件使用的右键菜单定义 XML：

edit 控件的菜单相对固定，因此我们也采用系统资源的形式提供一个预定义的右键菜单定义。

```
<editmenu trCtx="editmenu" iconSkin="_skin.sys.icons" itemHeight="26"
iconMargin="4" textMargin="8" >
  <item id="1" icon="3">cut</item>
  <item id="2" icon="4">copy</item>
  <item id="3" icon="5">paste</item>
  <item id="4" >delete</item>
  <sep/>
  <item id="5">select all</item>
</editmenu>
```

和前面两个不同，菜单资源每一个 item 必须包含一个 id，取值从 1-5,菜单项的位置可以任意。

4.3.1 应用程序自定义资源

很显然，系统资源提供的样式使得应用中每一个同类控件长得都一样。但是很多时候我们会希望两个功能相似的控件有不一样的长相，这就需要使用用户自定义资源。

用户自定义资源和系统资源一样，只不过它可以包含更多类型（任意类型），资源也可以任意命名（只要不和系统资源冲突）。

下面为 demo 中使用的自定义资源(uires.idx)：

```
<?xml version="1.0" encoding="utf-8"?>
<resource>
  <UIDEF>
    <file name="xml_init" path="xml\init.xml"  />
  </UIDEF>
  <ICON>
    <file name="LOGO"  path="image\img_logo.ico" />
  </ICON>
  <CURSOR>
    <file name="ANI_ARROW"  path="image\021.ani" />
    <file name="CUR_TST" path="image\camera_capture.cur"/>
  </CURSOR>
  <LAYOUT>
    <file name="maindlg" path="xml\dlg_main.xml"  />
```

```
    <file name="menu_test" path="xml\menu_test.xml" />
    <file name="page_layout" path="xml\page_layout.xml" />
    <file name="page_treebox" path="xml\page_treebox.xml" />
    <file name="page_treectrl" path="xml\page_treectrl.xml" />
    <file name="page_misc" path="xml\page_misc.xml" />
    <file name="page_webkit" path="xml\page_webkit.xml" />
    <file name="page_about" path="xml\page_about.xml" />
  </LAYOUT>
  <IMGX>
    <file name="png_page_icons"  path="image\page_icons.png" />
    <file name="png_small_icons"  path="image\small_icons.png" />

    <file name="webbtn_back"  path="image\webbtn_back.png" />
    <file name="webbtn_forward"  path="image\webbtn_forward.png" />
    <file name="webbtn_refresh"  path="image\webbtn_refresh.png" />

    <file name="png_treeicon"          path="image\TreeIcon.png"/>
    <file name="png_menu_border"  path="image\menuborder.png" />

    <file name="png_vscroll"  path="image\vscrollbar.png" />

    <file name="png_tab_left"  path="image\tab_left.png" />
    <file name="png_tab_left_splitter"  path="image\tab_left_splitter.png" />
    <file name="png_tab_main"  path="image\tab_main.png" />
    <file name="btn_menu" path="image\btn_menu.png" />
  </IMGX>
  <GIF>
    <file name="gif_horse"   path="image\horse.gif"/>
    <file name="gif_penguin"   path="image\penguin.gif"/>
  </GIF>
  <rtf>
    <file name="rtf_test"   path="rtf\RTF 测试.rtf"/>
  </rtf>
  <script>
    <file name="lua_test" path="lua\test.lua"/>
  </script>
  <translator>
    <file name="lang_cn" path="translation files\lang_cn.xml"/>
  </translator>
</resource>
```

不管是系统资源还是用户资源都由资源管理模块管理。

实际上这两种资源可以合并到一个资源包中交给系统管理。有心人可能注意到了项目中使用的系统资源使用的是一个资源 DLL，并且没有 uires.idx 文件。正是因为使用了资源 DLL 这一形式，才可以不提供 uires.idx 文件，因为 PE 资源本身已经有了分类命名（每一个资源都在一个类型下）。

DEMO 中使用系统资源和用户资源

```
        SApplication *theApp=new SApplication(pRenderFactory,hInstance);
        //定义一人个资源提供对象,SOUI 系统中实现了 3 种资源加载方式，分别是从文件加载，从
EXE 的资源加载及从 ZIP 压缩包加载
        CAutoRefPtr<IResProvider>   pResProvider;
#if (RES_TYPE == 0)//从文件加载
        CreateResProvider(RES_FILE,(IObjRef**)&pResProvider);
        if(!pResProvider->Init((LPARAM)_T("uires"),0))
        {
            SASSERT(0);
            return 1;
        }
#elif (RES_TYPE==1)//从 EXE 资源加载
        CreateResProvider(RES_PE,(IObjRef**)&pResProvider);
        pResProvider->Init((WPARAM)hInstance,0);
#elif (RES_TYPE==2)//从 ZIP 包加载
        bLoaded=pComMgr->CreateResProvider_ZIP((IObjRef**)&pResProvider);
        SASSERT_FMT(bLoaded,_T("load interface [%s]
failed!"),_T("resprovider_zip"));

        ZIPRES_PARAM param;
        param.ZipFile(pRenderFactory, _T("uires.zip"),"souizip");
        bLoaded = pResProvider->Init((WPARAM)&param,0);
        SASSERT(bLoaded);
#endif
        //将创建的 IResProvider 交给 SApplication 对象
        theApp->AddResProvider(pResProvider);

        //加载系统资源
        HMODULE hSysResource=LoadLibrary(SYS_NAMED_RESOURCE);
        if(hSysResource)
        {
            CAutoRefPtr<IResProvider> sysSesProvider;
            CreateResProvider(RES_PE,(IObjRef**)&sysSesProvider);
            sysSesProvider->Init((WPARAM)hSysResource,0);
            theApp->LoadSystemNamedResource(sysSesProvider);
        }
```

```
        //加载全局资源描述 XML
        theApp->Init(_T("xml_init"));
```

可以看到这里根据预定义宏：RES_TYPE 提供了 3 种参考资源加载形式来加载用户自定义资源。然后再采用 SResProviderPE 的资源加载器从系统资源 DLL 中加载系统资源。

加载系统资源一个关键步骤在于调用:

```
theApp->LoadSystemNamedResource(sysSesProvider);
```

这个函数从系统资源加载器中读取那些命名的系统资源。

所有资源加载完成后调用：

```
//加载全局资源描述 XML
theApp->Init(_T("xml_init"));
```

来初始化资源中定义的全局 Skin,Style,ObjAttr 对象。

大家可以想一想怎么样把这两种资源使用一个资源加载器来完成。

4.4 在 SOUI 中用九宫格拉伸方式显示一个图片资源

SOUI 的初学者刚开始可能难以搞清楚在 SOUI 中显示一个图片资源的流程，这里做一个简单的示范。

首先我们准备好一张图，以下图为例。

第一步，我们首先把这个图片文件复制到 demo 的 uires 目录下，新建一个目录 jpg,下面只有这一个文件 9.jpg

第二步，我们需要在 uires.idx 中引入该图片资源

```
<jpg>
  <file name="girl" path="jpg\9.jpg"/>
</jpg>
```

我们给这个资源命名为"girl"。

第三步，我们在全局或者窗口局部的 skin 结点中定义一个 imgframe 对象。这里定义在主窗口的局部 skin 中。

```
<skin>
  <!--局部 skin 对象-->
  <gif name="gif_horse" src="gif:gif_horse"/>
  <gif name="gif_penguin" src="gif:gif_penguin"/>
  <imgframe name="skin_girl" src="jpg:girl" margin-x="150" margin-y="150"/>
</skin>
```

注意上面代码中对 girl 的引用，我们保留 x 及 y 方向各 150 个点不拉伸。

第四步，在 UI 中定义一个 img 控件对象来显示该图片。

```
  <page title="jpg:girl">
    <img pos="0,0,-0,-0" skin="skin_girl"/>
  </page>    <page title="jpg:girl">
    <img pos="0,0,-0,-0" skin="skin_girl"/>
  </page>
```

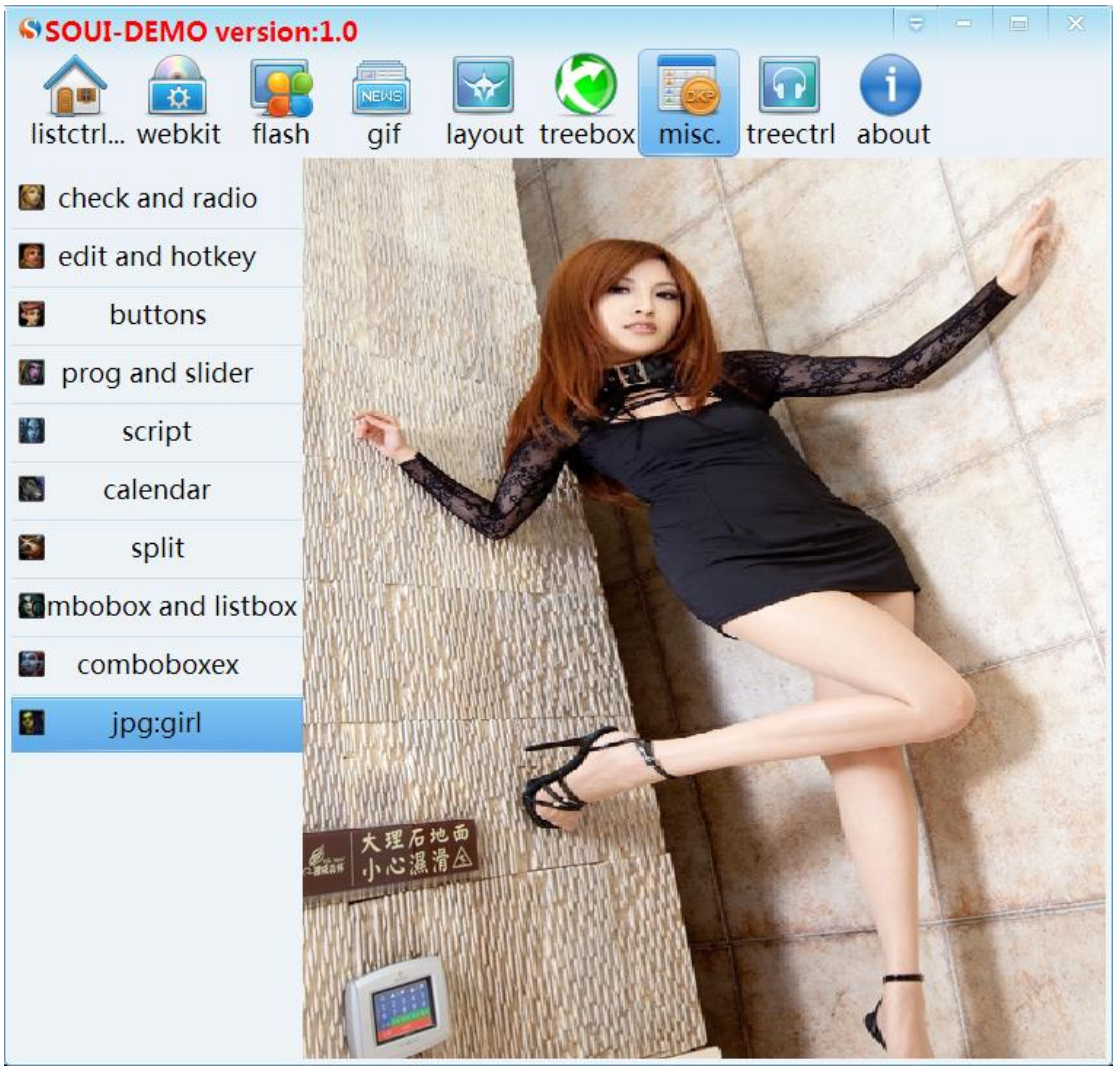
大功告成！

我们运行一下程序看看效果。

下面是缩小状态：

可以看到边缘的点和中间的点拉伸不一样。

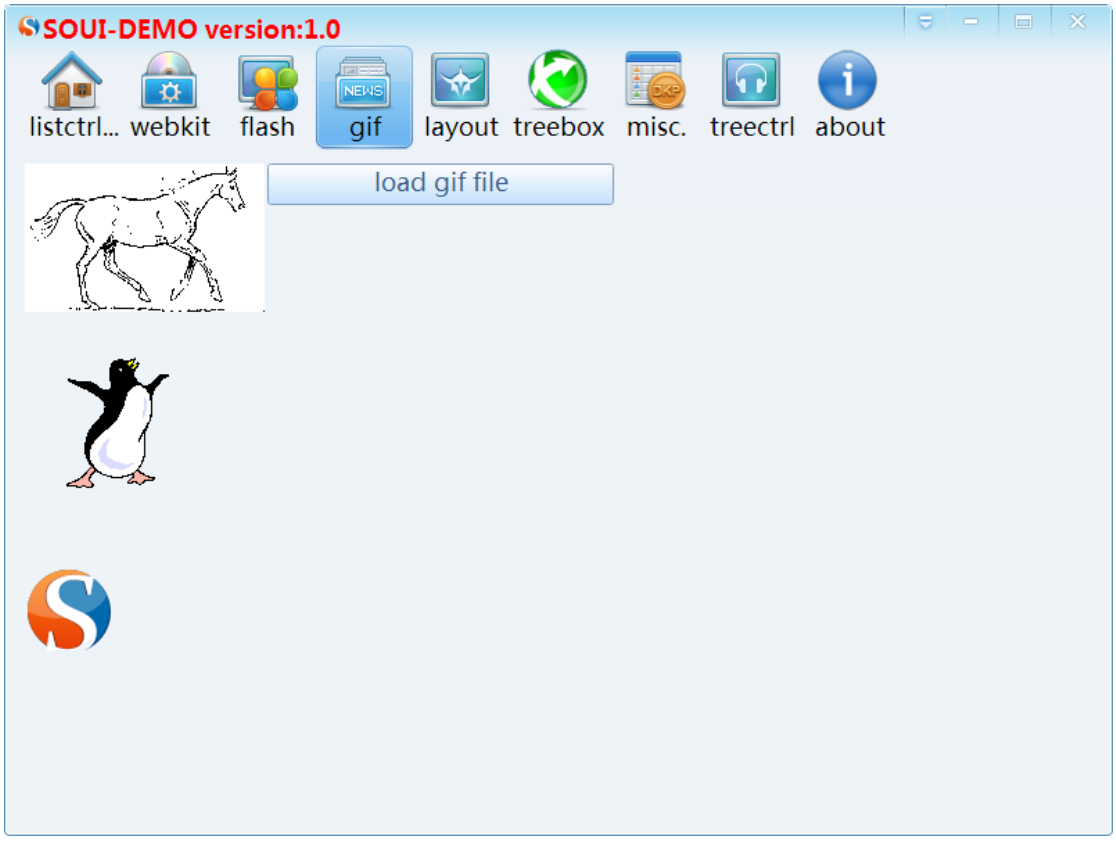
再看看放大一点的状态：



这样效果看上去好些了。

全部工作就是修改 XML 文件，不需要涉及一行 C++代码，即可完成一个图片的显示。

从文件中加载图片基本类似，可以参考 demo 中从文件中加载 GIF 动画的例子。



4.7 在 SOUI 中使用有窗口句柄的子窗口

无论一个 DirectUI 系统提供的 DUI 控件多么丰富，总会有些情况下用户需要在 DUI 窗口上放置有窗口句柄的子窗口。

为了和无窗口句柄的子窗口相区别，这里将有窗口句柄的子窗口称之为真窗口。

每一个使用 SOUI 创建的界面都是从 SHostWnd 派生出来的。SHostWnd 本身就是一个有窗口句柄的真窗口。

因此和一般的 win32 编程一样，用户可以简单的自己以 SHostWnd.m_hWnd 为父窗口创建各种真子窗口。然后和 win32 一样，响应 resize 等消息自己管理子窗口的位置及显示。

很显然，这样处理将不能有效的利用 SOUI 提供的强大的布局及子窗口管理功能。

为了能够更有效的管理真窗口，在 SOUI 系统中提供了一个控件：SRealWnd。

SRealWnd 派生自 SWindow，因此它能够实现和 SWindow 一样的布局功能，并被 SOUI 系统管理窗口的各种状态：如 size,visible 等。

要使用 SReaWnd 来管理子窗口，我们首先需要实现一个接口：IRealWndHandler

IRealWndHandler 的定义：

```
/**
* @struct    IRealWndHandler
* @brief
```

```
*
* Describe
*/
struct IRealWndHandler : public IObjRef
{
    /**
    * SRealWnd::OnRealWndCreate
    * @brief    窗口创建
    * @param    SRealWnd *pRealWnd -- 窗口指针
    *
    * Describe  窗口创建
    */
    virtual HWND OnRealWndCreate(SRealWnd *pRealWnd)=NULL;

    /**
    * SRealWnd::OnRealWndDestroy
    * @brief    销毁窗口
    * @param    SRealWnd *pRealWnd -- 窗口指针
    *
    * Describe  销毁窗口
    */
    virtual void OnRealWndDestroy(SRealWnd *pRealWnd)=NULL;

    /**
    * SRealWnd::OnRealWndInit
    * @brief    初始化窗口
    * @param    SRealWnd *pRealWnd -- 窗口指针
    * @return   BOOL -- FALSE:交由系统处理，TRUE:用户处理
    *
    * Describe  初始化窗口
    */
    virtual BOOL OnRealWndInit(SRealWnd *pRealWnd)=NULL;

    /**
    * SRealWnd::OnRealWndSize
    * @brief    调整窗口大小
    * @param    SRealWnd *pRealWnd -- 窗口指针
    * @return   BOOL -- FALSE: 交由 SOUI 处理; TRUE:用户管理窗口的移动
    *
    * Describe  调整窗口大小
    */
    virtual BOOL OnRealWndSize(SRealWnd *pRealWnd)=NULL;
};
```

可以看到这里一共有 4 个接口，其中 OnRealWndInit 是 OnRealWndSize 为真窗口初始化及位置调整的回调，一般可以不处理，其它 2 个接口则是管理真窗口的创建及销毁，因此必须有实现。

接口实现示例：

真窗口的具体使用方法可以参考 SOUI 代码中 samples 目录下的 mfc.demo。

这里把代码实现帖出来：

SouiRealWndHandler.h

```
#pragma once
#include <unknown/obj-ref-impl.hpp>
namespace SOUI
{
    class CSouiRealWndHandler :public
TObjRefImpl2<IRealWndHandler,CSouiRealWndHandler>
    {
    public:
        CSouiRealWndHandler(void);
        ~CSouiRealWndHandler(void);

        /**
         * SRealWnd::OnRealWndCreate
         * @brief    创建真窗口
         * @param    SRealWnd * pRealWnd --  窗口指针
         * @return   HWND -- 创建出来的真窗口句柄
         * Describe
         */
        virtual HWND OnRealWndCreate(SRealWnd *pRealWnd);

        /**
        * SRealWnd::OnRealWndDestroy
        * @brief    销毁窗口
        * @param    SRealWnd *pRealWnd -- 窗口指针
        *
        * Describe  销毁窗口
        */
        virtual void OnRealWndDestroy(SRealWnd *pRealWnd);

        /**
        * SRealWnd::OnRealWndInit
        * @brief    初始化窗口
        * @param    SRealWnd *pRealWnd -- 窗口指针
        *
```

```
        * Describe  初始化窗口
        */
        virtual BOOL OnRealWndInit(SRealWnd *pRealWnd);

        /**
        * SRealWnd::OnRealWndSize
        * @brief    调整窗口大小
        * @param    SRealWnd *pRealWnd -- 窗口指针
        * @return   BOOL -- TRUE:用户管理窗口的移动；FALSE：交由 SOUI 自己管理。
        * Describe  调整窗口大小，从 pRealWnd 中获得窗口位置。
        */
        virtual BOOL OnRealWndSize(SRealWnd *pRealWnd);
    };

}
```

SouiRealWndHandler.cpp:

```
#include "StdAfx.h"
#include "SouiRealWndHandler.h"

namespace SOUI
{
    CSouiRealWndHandler::CSouiRealWndHandler(void)
    {
    }

    CSouiRealWndHandler::~CSouiRealWndHandler(void)
    {
    }

    HWND CSouiRealWndHandler::OnRealWndCreate( SRealWnd *pRealWnd )
    {
        const SRealWndParam &param=pRealWnd->GetRealWndParam();
        if(param.m_strClassName==_T("button"))
        {//只实现了 button 的创建
            //分配一个 MFC CButton 对象
            CButton *pbtn=new CButton;
            //创建 CButton 窗口,注意使用 pRealWnd->GetContainer()->GetHostHwnd()作
为 CButton 的父窗口
            //把 pRealWnd->GetID()作为真窗口的 ID

pbtn->Create(param.m_strWindowName,WS_CHILD|WS_VISIBLE|BS_PUSHBUTTON,::CRect
```

```
(0,0,0,0),CWnd::FromHandle(pRealWnd->GetContainer()->GetHostHwnd()),pRealWnd
->GetID());
            //把 pbtn 的指针放到 SRealWnd 的 Data 中保存，以便在窗口 destroy 时释放 pbtn 对
象。
            pRealWnd->SetData(pbtn);
            //返回成功创建后的窗口句柄
            return pbtn->m_hWnd;
        }else
        {
            return 0;
        }
    }

    void CSouiRealWndHandler::OnRealWndDestroy( SRealWnd *pRealWnd )
    {
        const SRealWndParam &param=pRealWnd->GetRealWndParam();
        if(param.m_strClassName==_T("button"))
        {//销毁真窗口，释放窗口占用的内存
            CButton *pbtn=(CButton*) pRealWnd->GetData();
            if(pbtn)
            {
                pbtn->DestroyWindow();
                delete pbtn;
            }
        }
    }

    //不处理，返回 FALSE
    BOOL CSouiRealWndHandler::OnRealWndSize( SRealWnd *pRealWnd )
    {
        return FALSE;
    }

    //不处理，返回 FALSE
    BOOL CSouiRealWndHandler::OnRealWndInit( SRealWnd *pRealWnd )
    {
        return FALSE;
    }
}
```

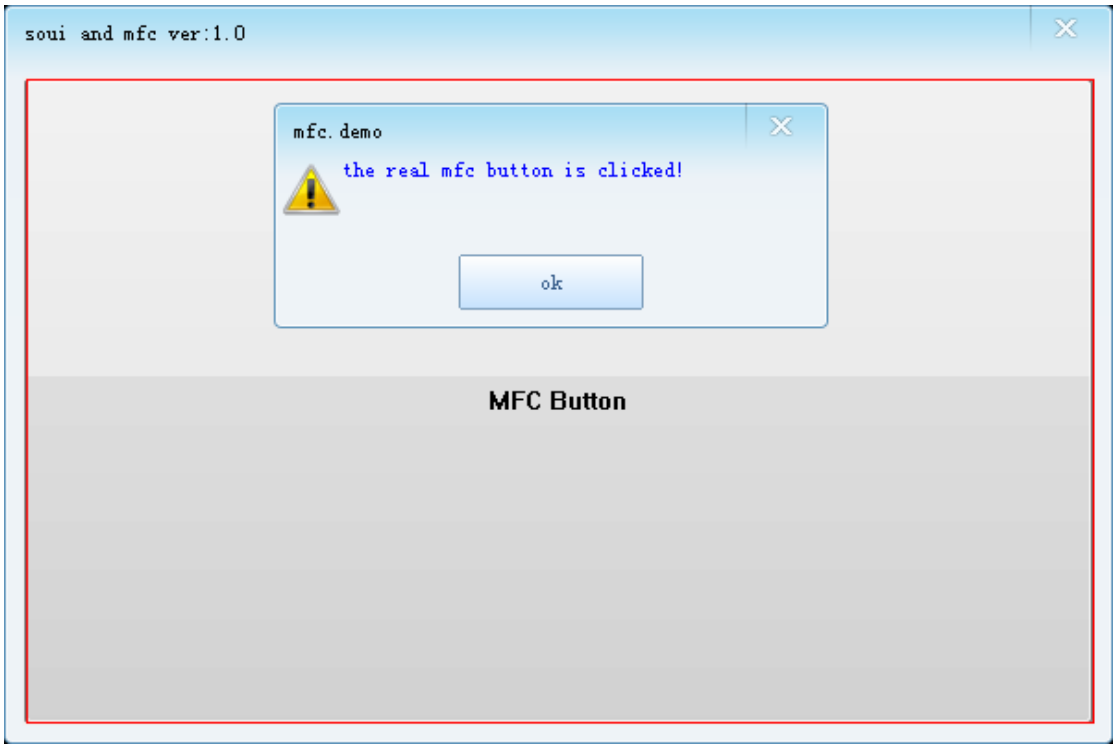
整体上代码很简单，配上注释，应该一看就懂。

XML 配置：

```
<SOUI title="DUI-DEMO" width="600" height="400" appwin="0" ncRect="5,5,5,5"
resize="1" translucent="0">
  <root skin="skin.bkframe" cache="1">
    <caption pos="0,0,-0,29">
      <text pos="11,9" >%title% ver:%ver%</text>
      <imgbtn id="1" name="btn_close" skin="skin.btnclose" pos="-45,0"
tip="close" animate="0"/>
    </caption>
    <window pos="0,29,-0,-0">
      <realwnd pos="10,10,-10,-10" name="mfcbtn" wndclass="button" id="100"
wndname="MFC Button"/>
    </window>
  </root>
</SOUI>
```

在 XML 中，我们使用了一个 realwnd 的标签，该标签有一个重要的属性：wndclass，IRealWndHandler 通过该属性来判断应该创建一个什么样的真窗口。

运行效果：



上面红框中的按钮即为使用 realwnd 标签创建的 MFC Button。

真窗口的消息响应：

由于真窗口是 SOUI 主窗口的子窗口，因此真窗口的消息可以在 SOUI 主窗口的消息映射表中处理（注意：这里不是 SOUI 控件的事件映射表）。

如：

```
#pragma once
```

```
class CRealWndDlg : public SOUI::SHostDialog
{
public:
    CRealWndDlg(void);
    ~CRealWndDlg(void);

    //响应MFC.button的按下消息，nID==100为在XML中指定的realwnd的id属性。
    void OnBtnClick( UINT uNotifyCode, int nID, HWND wndCtl )
    {
        if(uNotifyCode == BN_CLICKED && nID == 100)
        {
            SOUI::SMessageBox(m_hWnd,_T("the real mfc button is
clicked!"),_T("mfc.demo"),MB_OK|MB_ICONEXCLAMATION);
        }
    }

    //消息映射表
    BEGIN_MSG_MAP_EX(CMainDlg)
        MSG_WM_COMMAND(OnBtnClick)
        CHAIN_MSG_MAP(SOUI::SHostDialog)
        REFLECT_NOTIFICATIONS_EX()
    END_MSG_MAP()
};
```

结束语：

很显然，通过这种方式，也可以非常方便的创建出各种类型的其它窗口。

窗口创建出来后，系统就会自动管理窗口状态。

最后，要记住一条：有真窗口时，SOUI 主窗口不能设置 translucent="1"这一属性。因为任何子窗口在半透明窗口上都不能正常显示。这一条也适用于包含 IE 控件的窗口。

4.8 SOUI 中控件事件的响应

SOUI 中提供了大部分常用的 win32 标准控件的实现，如 pushbutton, checkbox, radiobox, edit, richedit, listbox, combobox, treectrl, listctrl (report), hotkeyctrl 等。大部分控件在接收用户输入后，会发生状态的改变，并以事件的形式传递给 UI 的所有者。在 SOUI 中提供了两种处理事件的方式：

4.8.1 在 SHostWnd 的派生类中重载

```
virtual BOOL SHostWnd::_HandleEvent(SOUI::EventArgs *pEvt){return FALSE;}
```

为了更方便的处理事件，SOUI 提供了一组宏来构造这个事件处理函数，从而提供一种类似消息映射的事件处理形式。

如 demo 的 CMainDlg 中的实现：

```
//UI 控件的事件及响应函数映射表
EVENT_MAP_BEGIN()
    EVENT_ID_COMMAND(1, OnClose)
    EVENT_ID_COMMAND(2, OnMaximize)
    EVENT_ID_COMMAND(3, OnRestore)
    EVENT_ID_COMMAND(5, OnMinimize)
    EVENT_NAME_CONTEXTMENU(L"edit_1140",OnEditMenu)
    EVENT_NAME_COMMAND(L"btn_msgbox",OnBtnMsgBox)
    EVENT_NAME_COMMAND(L"btnSelectGif",OnBtnSelectGIF)
    EVENT_NAME_COMMAND(L"btn_menu",OnBtnMenu)
    EVENT_NAME_COMMAND(L"btn_webkit_go",OnBtnWebkitGo)
    EVENT_NAME_COMMAND(L"btn_webkit_back",OnBtnWebkitBackward)
    EVENT_NAME_COMMAND(L"btn_webkit_fore",OnBtnWebkitForeward)
    EVENT_NAME_COMMAND(L"btn_webkit_refresh",OnBtnWebkitRefresh)
    EVENT_NAME_COMMAND(L"btn_hidetst",OnBtnHideTest)
    EVENT_NAME_COMMAND(L"btn_insert_gif",OnBtnInsertGif2RE)
EVENT_MAP_END()
```

上面的 EVENT_MAP_BEGIN()和 EVENT_MAP_END()结合构造出一个_HandleEvent 函数的实现，具体可以自己展开这两个宏查看代码。

同时 SOUI 也提供了一组解析 SOUI::EventArgs *pEvt 的宏，如上例中的 EVENT_NAME_COMMAND, EVENT_ID_COMMAND 等。

帮助用户直接从控件的 name 或者 ID 属性映射到消息响应函数。

这种事件响应方式最大的好处是能够集中处理事件的分发，方便阅读代码，同时也和传统的 MFC，WTL 的编程风格类似，降低用户的学习成本。

4.8.2 采用事件订阅的方式响应控件事件

虽然事件映射表提供了一种简单有效的事件响应机制，由于事件映射表是一种编译期形成的静态的映射表，对于在运行期动态创建的控件的事件响应就无能为力了。

在 MFC 中，程序员通过要重载窗口类的 DefWindowProc 来处理运行期间动态创建的控件发来的消息。

这种方式灵活性够了，但是不够优雅，要在一个函数里做大量的 swich 分枝，导致这个处理函数很难维护。

设计模式里的观察者模式可以比较好的解决这个问题。

为些在 SOUI 中我提供了一种事件订阅的事件处理模式。

我们先看一下 demo 中怎样处理列表控件的表头点击来执行排序操作：

```
void CMainDlg::InitListCtrl()
{
```

```
    //找到列表控件
    SListCtrl *pList=FindChildByName2<SListCtrl>(L"lc_test");
    if(pList)
    {
        //列表控件的唯一子控件即为表头控件
        SWindow *pHeader=pList->GetWindow(GSW_FIRSTCHILD);
        //向表头控件订阅表明点击事件，并把它和 OnListHeaderClick 函数相连。

pHeader->GetEventSet()->subscribeEvent(EVT_HEADER_CLICK,Subscriber(&CMainDlg
::OnListHeaderClick,this));
        //省略列表初始化代码
    }
}
//表头点击事件处理函数
bool CMainDlg::OnListHeaderClick(EventArgs *pEvtBase)
{
    //事件对象强制转换
    EventHeaderClick *pEvt =(EventHeaderClick*)pEvtBase;
    SHeaderCtrl *pHeader=(SHeaderCtrl*)pEvt->sender;
    //从表头控件获得列表控件对象
    SListCtrl *pList= (SListCtrl*)pHeader->GetParent();
    //列表数据排序
    SHDITEM hditem;
    hditem.mask=SHDI_ORDER;
    pHeader->GetItem(pEvt->iItem,&hditem);
    pList->SortItems(funCmpare,&hditem.iOrder);
    return true;
}
```

通过事件订阅可以在运行时方便的将一个控件的事件关联到一个处理函数上，当然也可以随时取消订阅。

同时事件订阅也是在脚本中响应控件事件的唯一方式（关于在 SOUI 中使用 LUA 脚本将在后续讲解）。

4.9 SOUI 多语言翻译机制

为 UI 在不同地区显示不同的语言是产品国际化的一个重要要求。

在 SOUI 中实现了一套类似 QT 的多语言翻译机制：布局 XML 不需要调整，程序代码也不需要调整，只需要为不同地区的用户提供不同的语言翻译文件即可。

在 SOUI 中，我们实现了一个使用明文 XML 的语言翻译模块：translator.dll

为了使用多语言翻译，首先需要准备一个语言翻译的 XML 文件。demo 中使用的翻译文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
<language name="ch" guid="{0DAEDE3C-6B94-4a81-9A55-C304FDD69D98}">
  <context>
    <!--没有上下文的翻译-->
  </context>
  <context name="editmenu">
    <message>
      <source>copy</source>
      <translation>复制</translation>
    </message>
    <message>
      <source>cut</source>
      <translation>剪切</translation>
    </message>
    <message>
      <source>paste</source>
      <translation>粘贴</translation>
    </message>
  </context>
  <context name="messagebox">
    <message>
      <source>ok</source>
      <translation>确定</translation>
    </message>
    <message>
      <source>cancel</source>
      <translation>取消</translation>
    </message>
    <message>
      <source>retry</source>
      <translation>重试</translation>
    </message>
  </context>
  <context name="dlg_main">
    <message>
      <source>close</source>
      <translation>关闭窗口</translation>
    </message>
  </context>
```

```
</language>
```

可以看到该 XML 中有一个 language 的根节点，该节点有两个属性：name 和 guid，这两个属性都是用来标识该翻译文件的。

在 language 节点下，有多个 context 节点，每个 context 节点有一个 name 属性（可以为空），对应一个翻译上下文。

每一个 context 下有不同数量的 message 结点，每个 message 又有两个子节点：source 和 translation。

source 对应需要翻译的文字，而 translation 则对应翻译后的文字。

要使用这个语言翻译文件，首先需要从 translator.dll 中创建一个 SOUI::ITranslatorMgr 对象，并将该对象交给 SOUI::SApplication 管理。

再从 ItranslatorMgr 对象创建 SOUI::ITranslator 对象，并将 Itranslator 对象添加到 ItranslatorMgr 管理的翻译列表中。

最后还要为 ITranslator 对象加载翻译数据源（也就是前面提供的 XML 文件）。

下面是 demo 中使用和语言翻译相关的代码（见_tWinMain 函数）

```
        SApplication *theApp=new SApplication(pRenderFactory,hInstance);//SOUI
APP
        CAutoRefPtr<ITranslatorMgr> transMgr; //多语言翻译模块，由 translator.dll
提供
        transLoader.CreateInstance("translator.dll",(IObjRef**)&transMgr);//
        if(trans)
        {//加载语言翻译包
            theApp->SetTranslator(transMgr);
            pugi::xml_document xmlLang;
            if(theApp->LoadXmlDocment(xmlLang,_T("lang_cn"),
_T("translator")))
            {
                CAutoRefPtr<ITranslator> langCN;
                transMgr->CreateTranslator(&langCN);
                langCN->Load(&xmlLang.child(L"language"),1);
                //1=LD_XML
                transMgr->InstallTranslator(langCN);
            }
        }
```

我们先看一下 editmemu 的 XML：

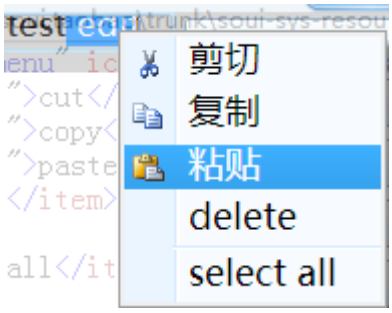
```
<editmenu trCtx="editmenu" iconSkin="_skin.sys.icons" itemHeight="26"
iconMargin="4" textMargin="8" >
  <item id="1" icon="3">cut</item>
  <item id="2" icon="4">copy</item>
  <item id="3" icon="5">paste</item>
```

```
  <item id="4" >delete</item>
  <sep/>
  <item id="5">select all</item>
</editmenu>
```

在这个 XML 中，根节点有一个属性 trCtx，代表翻译上下文，对应语言翻译文件中的 context 中的 name 属性。

在这个 menu 定义中，所有的菜单项的文字全是英文。其中前面 3 项：cut, copy and paste 在语言翻译文件中有对应的翻译项。

下图为程序运行时 edit 的右键菜单显示结果：



上面演示的是菜单资源的语言翻译，布局 XML 中的文字的翻译基本一样，只需要为布局的根结点定义一个翻译上下文(trCtx)（没有定义时则从没有指定 name 属性的 context 里查找翻译结果）。

可以参见 demo 的"close”按照的 tooltip 的翻译。

不管是菜单 XML 还是布局 XML，它们都是静态的，经过设计需要使用代码往 UI 添加新的文字，同样也需要翻译，这个时候如何处理？

其实看一下静态资源翻译的代码就知道，要实现语言翻译，需要为每一句待翻译的文字调用一个函数：

```
SApplication::getSingleton().GetTranslator()->tr(const SStringW &
strSrc,const SStringW & strCtx)
```

第一个参数是等翻译字符串，第二个参数是翻译上下文。

考虑到语句太长，系统提供了一个宏：

```
#define TR(p1,p2)
SApplication::getSingleton().GetTranslator()->tr(p1,p2)
```

这样用 TR 就可以实现文字翻译了。

4.10 自定义控件

4.10.1 开发自定义控件

在 SOUI 中已经提供了大部分常用的控件，但是内置控件不可能满足用户的所有要求，因此一个真实的应用少不得还要做一些自定义控件。

学习一个新东西，最简单的办法就是依葫芦画瓢。事实上在 SOUI 系统中内置控件和自定义控件的开发流程是完全一样的，因此只需要打开 SOUI 的源代码，随便找一个控件看一下就大体差不多了。

下面我以 controls.extend 目录下的的 SRadioBox2 控件为例对控件开发过程需要注意的地方做一点说明。

要开发一个控件，首先要确定的是应该从哪个控件来继承。选择一个合适的基类是正确开发自定义控件的前提。

之所以要开发一个 SRadioBox2 控件，我需要解决的问题很简单：SRadioBox 控件总是在左边显示一个圆圈，这个圆圈有时候不是我想要的。

因此我需要做的就是继承 SRadioBox 控件的行为，重写 WM_PAINT 的处理。

因此就有了下面的代码：

```
class SRadioBox2 : public SRadioBox
{
public:
  SRadioBox2(void);
  ~SRadioBox2(void);
}
```

需要注意的是，所有 SOUI 控件都是在 namespace SOUI 中，因此自定义控件也最好是放在 SOUI 这个 namespace 里。

有了上面的骨架，下面来逐步添加内容。

首先我们需要给自定义控件定义一个在 XML 中可以识别的标签。

只需要在类的最开始增加一行：

```
    class SRadioBox2 : public SRadioBox
    {
    SOUI_CLASS_NAME(SRadioBox2,L"radio2")
    public:
        SRadioBox2(void);
        ~SRadioBox2(void);
    }
```

SOUI_CLASS_NAME 告诉 XML 解析器，碰到 radio2 时自动创建 SRadioBox2 对象。

实际上这一行更重要的作用是用来做对象类型运行时识别（RTTI）,有了这个机制，在编译器关闭 C++的 RTTI 时仍然可以安全的进行类型转换。

然后我们需要处理控件的 WM_PAINT 消息。

为了处理这个消息，我们需要加入消息映射表及消息处理函数：

```
    class SRadioBox2 : public SRadioBox
    {
    SOUI_CLASS_NAME(SRadioBox2,L"radio2")
    public:
```

```
        SRadioBox2(void);
        ~SRadioBox2(void);
    protected:
        void OnPaint(IRenderTarget *pRT);

        SOUI_MSG_MAP_BEGIN()
            MSG_WM_PAINT_EX(OnPaint)
        SOUI_MSG_MAP_END()
    };
```

SOUI 控件的消息处理机制是和 WTL 中抄过来的，和 MFC 也很相似。只是对于部分消息，由于对于消息的参数的解释不一样，消息映射的宏会有一点变化。

如这里的 WM_PAINT 消息，在 SOUI 里 wparam 传递的是一个 IRenderTarget 指针，而传统的 Win32 传递的是一个 HDC。因此我们需要使用 MSG_WM_PAINT_EX 代替 WTL 中使用的 MSG_WM_PAINT。

大家可能会问有哪些消息映射宏 SOUI 和 WLT 不一样？

实际上对于一个有经验的程序员，他应该可以找到 MSG_WM_PAINT_EX 的宏定义，并且在定义附近就可以找到所有的 SOUI 和 WLT 不同的映射宏。

下面是到目前为止 SOUI 中所有和 WTL 不同的宏：

```
// BOOL OnEraseBkgnd(IRenderTarget * pRT)
#define MSG_WM_ERASEBKGND_EX(func) \
    if (uMsg == WM_ERASEBKGND) \
    { \
    SetMsgHandled(TRUE); \
    lResult = (LRESULT)func((IRenderTarget *)wParam); \
    if(IsMsgHandled()) \
    return TRUE; \
    }

// void OnPaint(IRenderTarget * pRT)
#define MSG_WM_PAINT_EX(func) \
    if (uMsg == WM_PAINT) \
    { \
    SetMsgHandled(TRUE); \
    func((IRenderTarget *)wParam); \
    lResult = 0; \
    if(IsMsgHandled()) \
    return TRUE; \
    }

// void OnNcPaint(IRenderTarget * pRT)
```

```
#define MSG_WM_NCPAINT_EX(func) \
    if (uMsg == WM_NCPAINT) \
{ \
    SetMsgHandled(TRUE); \
    func((IRenderTarget *)wParam); \
    lResult = 0; \
    if(IsMsgHandled()) \
    return TRUE; \
}

// void OnSetFont(IFont *pFont, BOOL bRedraw)
#define MSG_WM_SETFONT_EX(func) \
    if (uMsg == WM_SETFONT) \
    { \
    SetMsgHandled(TRUE); \
    func((IFont*)wParam, (BOOL)LOWORD(lParam)); \
    lResult = 0; \
    if(IsMsgHandled()) \
    return TRUE; \
    }

// void OnSetFocus()
#define MSG_WM_SETFOCUS_EX(func) \
    if (uMsg == WM_SETFOCUS) \
{ \
    SetMsgHandled(TRUE); \
    func(); \
    lResult = 0; \
    if(IsMsgHandled()) \
    return TRUE; \
}

// void OnKillFocus()
#define MSG_WM_KILLFOCUS_EX(func) \
    if (uMsg == WM_KILLFOCUS) \
{ \
    SetMsgHandled(TRUE); \
    func(); \
    lResult = 0; \
    if(IsMsgHandled()) \
    return TRUE; \
}
```

```
// void OnNcMouseHover(int nFlag,CPoint pt)
#define MSG_WM_NCMOUSEHOVER(func) \
    if(uMsg==WM_NCMOUSEHOVER)\
{\
    SetMsgHandled(TRUE); \
    func(wParam,CPoint(GET_X_LPARAM(lParam),GET_Y_LPARAM(lParam))); \
    lResult = 0; \
    if(IsMsgHandled()) \
    return TRUE; \
}

// void OnNcMouseLeave()
#define MSG_WM_NCMOUSELEAVE(func) \
    if(uMsg==WM_NCMOUSELEAVE)\
{\
    SetMsgHandled(TRUE); \
    func(); \
    lResult = 0; \
    if(IsMsgHandled()) \
    return TRUE; \
}

// void OnTimer(char cTimerID)
#define MSG_WM_TIMER_EX(func) \
    if (uMsg == WM_TIMER) \
{ \
    SetMsgHandled(TRUE); \
    func((char)wParam); \
    lResult = 0; \
    if(IsMsgHandled()) \
    return TRUE; \
}

#define WM_TIMER2    (WM_USER+5432)    //定义一个与 HWND 定时器兼容的 SOUI 定时器

#define MSG_WM_TIMER2(func) \
    if (uMsg == WM_TIMER2) \
{ \
    SetMsgHandled(TRUE); \
    func(wParam); \
    lResult = 0; \
    if(IsMsgHandled()) \
    return TRUE; \
```

```
}
```

通常情况下，自定义控件还需要处理一些自定义的属性，这时我需要还需要增加一个属性映射表（如 SGifPlayer）：

```
    SOUI_ATTRS_BEGIN()
        ATTR_CUSTOM(L"skin", OnAttrSkin)
          //为控件提供一个 skin 属性，用来接收 SSkinObj 对象的 name
    SOUI_ATTRS_END()
```

完成上面几步，一个自定义控件基本上就完成了。

可能还需要实现几个修改基类行为的虚函数。

4.10.2 绘图对象（ISkinObj）的扩展

尽管 SOUI 已经内置了大部分常用的控件，很显然内置控件很难满足各种应用的形式各异的需求。

因此只有提供足够的扩展性才能满足真实应用场景。

除了将系统尽可能的组件化外，SOUI 在控件自绘(SWindow)及绘图对象(ISkinObj)两个方面提供用户扩展。

- **绘图对象(ISkinObj)的扩展**

系统内置了如 SSkinImgList, SSkinImgFrame, SSkinScrollbar 等绘图对象，在窗口中通过引用这些绘图对象可以绘制出不同的预定义图形图象（如按钮，滚动条，九宫格等）。

实际上用户可以实现任意的绘图对象并把它们注册到系统里，以便在 XML 及代码中使用。

下面先看一下实现一个 ISkinObj 需要实现哪些接口：

```
    /**
    * @struct     ISkinObj
    * @brief      Skin 对象
    *
    * Describe
    */
    class SOUI_EXP ISkinObj : public SObject,public
TObjRefImpl2<IObjRef,ISkinObj>
    {
    public:
        ISkinObj()
        {
        }
        virtual ~ISkinObj()
        {
```

```
    }

    /**
     * Draw
     * @brief    将 this 绘制到 RenderTarget 上去
     * @param    IRenderTarget * pRT --  绘制用的 RenderTarget
     * @param    LPCRECT rcDraw --  绘制位置
     * @param    DWORD dwState --  绘制状态
     * @param    BYTE byAlpha --  透明度
     * @return   void
     * Describe
     */
    virtual void Draw(IRenderTarget *pRT, LPCRECT rcDraw, DWORD
dwState,BYTE byAlpha=0xFF)=0;

    /**
     * GetSkinSize
     * @brief    获得 Skin 的默认大小
     * @return   SIZE -- Skin 的默认大小
     * Describe  派生类应该根据 skin 的特点实现该接口
     */
    virtual SIZE GetSkinSize()
    {
        SIZE ret = {0, 0};
        return ret;
    }

    /**
     * IgnoreState
     * @brief    查询 skin 是否有状态信息
     * @return   BOOL -- true 有状态信息
     * Describe
     */
    virtual BOOL IgnoreState()
    {
        return TRUE;
    }

    /**
     * GetStates
     * @brief    获得 skin 对象包含的状态数量
     * @return   int -- 状态数量
```

```
        * Describe  默认为1
        */
       virtual int GetStates()
       {
           return 1;
       }
   };
```

ISkinObj 是一个派生自 SObject 及 TObjRefImpl2<IObjRef,ISkinObj>的类，提供了几个状态查询相关的接口，也提供了一个 Draw 接口来在 IRenderTarget 上绘制该绘制对象。

注意的是，这些接口中只有 Draw 接口是纯虚接口。

在它的基类中，SOjbect 使得 ISkinObj 可以方便的从 XML 配置文件中初始化，而 TObjRefImpl2<IObjRef,ISkinObj>则提供引用计数的实现。

内置的 ISkinObj 不支持显示 GIF 图片，以显示 GIF 图象为例来分析如何扩展绘图对象来支持 GIF 图片显示。

对象定义（trunk\controls.extend\gif\SSkinGif.h）

```
namespace SOUI
{
    class SGifFrame
    {
    public:
        CAutoRefPtr<IBitmap> pBmp;
        int                  nDelay;
    };

    /**
    * @class     SSkinGif
    * @brief     GIF 图片加载及显示对象
    *
    * Describe
    */
    class SSkinGif : public ISkinObj
    {
        SOUI_CLASS_NAME(SSkinGif, L"gif")
    public:
        SSkinGif():m_nFrames(0),m_iFrame(0),m_pFrames(NULL)
        {

        }
```

```
        //初始化 GDI+环境，由于这里需要使用 GDI+来解码 GIF 文件格式
        static BOOL Gdiplus_Startup();
        //退出 GDI+环境
        static void Gdiplus_Shutdown();

        virtual ~SSkinGif()
        {
            if(m_pFrames) delete [] m_pFrames;
        }

        /**
         * Draw
         * @brief    绘制指定帧的 GIF 图
         * @param    IRenderTarget * pRT --  绘制目标
         * @param    LPCRECT rcDraw --  绘制范围
         * @param    DWORD dwState --  绘制状态，这里被解释为帧号
         * @param    BYTE byAlpha --  透明度
         * @return   void
         * Describe
         */
        virtual void Draw(IRenderTarget *pRT, LPCRECT rcDraw, DWORD
dwState,BYTE byAlpha=0xFF);

        /**
         * GetStates
         * @brief    获得 GIF 帧数
         * @return   int -- 帧数
         * Describe
         */
        virtual int GetStates(){return m_nFrames;}

        /**
         * GetSkinSize
         * @brief    获得图片大小
         * @return   SIZE -- 图片大小
         * Describe
         */
        virtual SIZE GetSkinSize()
        {
            SIZE sz={0};
            if(m_nFrames>0 && m_pFrames)
            {
                sz=m_pFrames[0].pBmp->Size();
```

```
        }
        return sz;
    }

    /**
     * GetFrameDelay
     * @brief    获得指定帧的显示时间
     * @param    int iFrame --  帧号,为-1 时代表获得当前帧的延时
     * @return   long -- 延时时间(*10ms)
     * Describe
     */
    long GetFrameDelay(int iFrame=-1);

    /**
     * ActiveNextFrame
     * @brief    激活下一帧
     * @return   void
     * Describe
     */
    void ActiveNextFrame();

    /**
     * SelectActiveFrame
     * @brief    激活指定帧
     * @param    int iFrame --  帧号
     * @return   void
     * Describe
     */
    void SelectActiveFrame(int iFrame);

    /**
     * LoadFromFile
     * @brief    从文件加载 GIF
     * @param    LPCTSTR pszFileName --  文件名
     * @return   int -- GIF 帧数, 0-失败
     * Describe
     */
    int LoadFromFile(LPCTSTR pszFileName);

    /**
     * LoadFromMemory
     * @brief    从内存加载 GIF
```

```
        * @param    LPVOID pBits --  内存地址
        * @param    size_t szData --  内存数据长度
        * @return   int -- GIF 帧数，0-失败
        * Describe
        */
       int LoadFromMemory(LPVOID pBits,size_t szData);

       SOUI_ATTRS_BEGIN()
          ATTR_CUSTOM(L"src",OnAttrSrc)   //XML 文件中指定的图片资源
名,(type:name)
       SOUI_ATTRS_END()
   protected:
       LRESULT OnAttrSrc(const SStringW &strValue,BOOL bLoading);
       int LoadFromGdipImage(Gdiplus::Bitmap * pImg);
       int m_nFrames;
       int m_iFrame;

       SGifFrame * m_pFrames;
   };
}//end of name space SOUI
```

对象注册：

```
theApp->RegisterSkinFactory(TplSkinFactory<SSkinGif>());
//注册 SkinGif
```

对象的使用（trunk\demo\uires\xml\dlg_main.xml）：

```
  <skin>
    <!--局部 skin 对象-->
    <gif name="gif_horse" src="gif:gif_horse"/>
    <gif name="gif_penguin" src="gif:gif_penguin"/>
  </skin>
```

这里的 gif 标签与 SSkinGif 类中的宏 SOUI_CLASS_NAME(SSkinGif,L"gif")中的“gif”是匹配的。

到此，在布局 XML 及程序中都可以获得这个 SSkinGif 对象的指针了。

4.10.3 控件的扩展

控件的扩展和绘图对象的扩展套路类似，也是先从系统提供的基础类派生，再注册到系统，最后再 XML 或者代码中使用。

和绘图对象不同在于，控件是 UI，需要处理各种 UI 相关的消息以及向程序发出各种控件特有的事件。

同样我们还是以 GIF 显示的控件为例（trunk\controls.extend\gif\SGifPlayer.h）：

```
namespace SOUI
```

```
{   /**
    * @class     SGifPlayer
    * @brief     GIF 图片显示控件
    *
    * Describe
    */
    class SGifPlayer : public SWindow
    {
        SOUI_CLASS_NAME(SGifPlayer, L"gifplayer")   //定义 GIF 控件在 XM 加的标签
    public:
        SGifPlayer();
        ~SGifPlayer();
        /**
         * PlayGifFile
         * @brief    在控件中播放一个 GIF 图片文件
         * @param    LPCTSTR pszFileName --  文件名
         * @return   BOOL -- true:成功
         * Describe
         */
        BOOL PlayGifFile(LPCTSTR pszFileName);

    protected://SWindow 的虚函数
        virtual CSize GetDesiredSize(LPRECT pRcContainer);

    public://属性处理
        SOUI_ATTRS_BEGIN()
            ATTR_CUSTOM(L"skin", OnAttrGif) //为控件提供一个 skin 属性，用来接收
SSkinObj 对象的 name
        SOUI_ATTRS_END()
    protected:
        HRESULT OnAttrGif(const SStringW & strValue, BOOL bLoading);

    protected://消息处理，SOUI 控件的消息处理和 WTL，MFC 很相似，采用相似的映射表，相同或
者相似的消息映射宏
        /**
         * OnPaint
         * @brief    窗口绘制消息响应函数
         * @param    IRenderTarget * pRT --  绘制目标
         * @return   void
         * Describe  注意这里的参数是 IRenderTarget *,而不是 WTL 中使用的 HDC，同时消息
映射宏也变为 MSG_WM_PAINT_EX
         */
        void OnPaint(IRenderTarget *pRT);
```

```
        /**
         * OnTimer
         * @brief    SOUI 窗口的定时器处理函数
         * @param    char cTimerID --  定时器 ID，范围从 0-127。
         * @return   void
         * Describe  SOUI 控件的定时器是 Host 窗口定时器 ID 的分解，以方便所有的控件都通过
Host 获得定时器的分发。
         *          注意使用 MSG_WM_TIMER_EX 来映射该消息。定时器使用
SWindow::SetTimer 及 SWindow::KillTimer 来创建及释放。
         *          如果该定时器 ID 范围不能满足要求，可以使用 SWindow::SetTimer2 来创
建。
         */
        void OnTimer(char cTimerID);

        /**
         * OnShowWindow
         * @brief    处理窗口显示消息
         * @param    BOOL bShow --  true:显示
         * @param    UINT nStatus --  显示原因
         * @return   void
         * Describe  参考 MSDN 的 WM_SHOWWINDOW 消息
         */
        void OnShowWindow(BOOL bShow, UINT nStatus);

        //SOUI 控件消息映射表
        SOUI_MSG_MAP_BEGIN()
            MSG_WM_TIMER_EX(OnTimer)    //定时器消息
            MSG_WM_PAINT_EX(OnPaint)    //窗口绘制消息
            MSG_WM_SHOWWINDOW(OnShowWindow)//窗口显示状态消息
        SOUI_MSG_MAP_END()

    private:
        SSkinGif *m_pgif;
        int    m_iCurFrame;
    };
}
```

实现了该类以后，在 WinMain 中注册：

```
theApp->RegisterWndFactory(TplSWindowFactory<SGifPlayer>());
//注册 GIFPlayer
```

我们可以在布局 XML 中创建 GIF 控件控件并显示了(trunk\demo\uires\xml\dlg_main.xml ）：

```
        <gifplayer pos="10,10" skin="gif_horse" name="giftest"
cursor="ANI_ARROW"/>
        <button width="250" height="30" name="btnSelectGif">load gif
file</button>
        <gifplayer pos="10,150" skin="gif_penguin"/>
```

上面的 gifplayer 节点即表示从 XML 中创建两个 SGifPlayer 类对象。

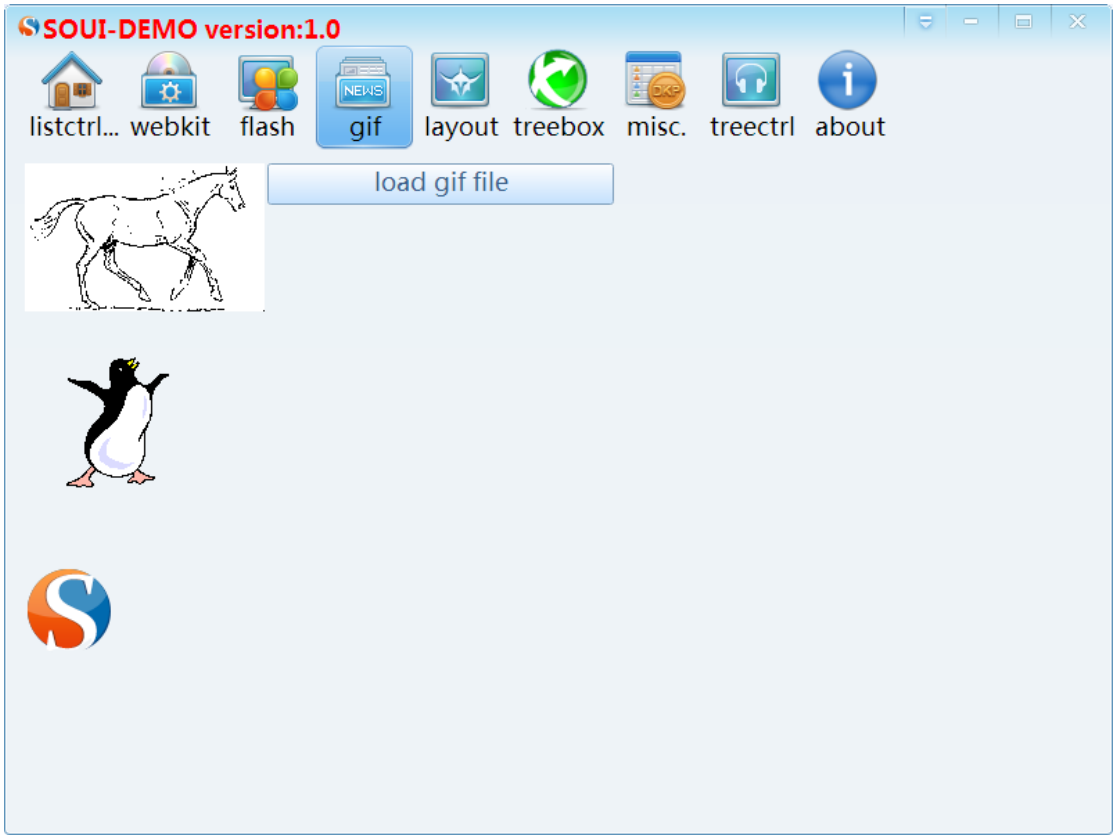
而 gifplayer 对象的 skin 属性则引用前面定义的 SSkinGif 对象。

备注：

实际上更多扩展技巧可以参考系统内置的控件的实现。内置控件与扩展控件唯一的区别就在于由谁实现将控件向系统注册。

内置控件在 SOUI 内核初始化的时候自动注册，而扩展控件则需要手动增加一行注册代码。

效果预览



4.11 在 SOUI 中使用定时器

定时器是 win32 编程中常用的制作动画效果的手段。在 Win32 编程中，可以使用::SetTimer 来创建定时器，定时器消息会被会发到调用 SetTimer 时指定的 HWND。

在 SOUI 中一般来说只有一个宿主窗口有 HWND，所有的 SWindow 都属于一个宿主窗口，如此一来直接使用::SetTimer 创建的定时器就难以直接分发到 SWindow 对象了。

在 SOUI 的控件中使用定时器

为了能够方便的在 SWindow 中使用定时器，在 SOUI 系统中，我们通过将定时器 ID（共 32 位）按位进行分解：

```
    class SOUI_EXP STimerID
    {
    public:
        DWORD    Swnd:24;        //窗口句柄,如果窗口句柄超过 24 位范围，则不能使用这种方式设置定时器
        DWORD    uTimerID:7;        //定时器 ID，一个窗口最多支持 128 个定时器。
        DWORD    bSwndTimer:1;    //区别通用定时器的标志，标志为 1 时，表示该定时器为 SWND 定时器
        STimerID(SWND hWnd,char id)
        {
            SASSERT(hWnd<0x00FFFFFF && id>=0);
            bSwndTimer=1;
            Swnd=hWnd;
            uTimerID=id;
        }
        STimerID(DWORD dwID)
        {
            memcpy(this,&dwID,sizeof(DWORD));
        }
        operator DWORD &() const
        {
            return *(DWORD*)this;
        }
    };
```

低 24 位用来存储 SWindow 的窗口 ID(swnd)。swnd 是一个 SWindow 创建序号，在一个应用中，通常很难产生超过 0xFFFFFF(16777215)个窗口对象，因此使用低 24 位来存储窗口 ID 在大部分情况下都是够用的了。

高 8 位中保留最高位设置为 1,用来区别直接使用::SetTimer 创建的定时器（不可以把最高位置 1）。

剩下 7 位用于 SWindow 中作为定时器 ID。因此在 SOUI 中，一个 SWindow 最多可以创建 0-127 个定时器。

创建定时器：

SWindow::SetTimer（0~127);

```
        /**
```

```
        * SWindow::SetTimer
        * @brief    利用窗口定时器来设置一个 ID 为 0-127 的 SWND 定时器
        * @param    char id --  定时器 ID
        * @param    UINT uElapse --  延时(MS)
        * @return   BOOL
        *
        * Describe  参考::SetTimer
        */
        BOOL SWindow::SetTimer(char id,UINT uElapse);
```

销毁定时器:

SWindow::KillTimer;

```
        /**
        * KillTimer
        * @brief    删除一个 SWND 定时器
        * @param    char id --  定时器 ID
        * @return   void
        *
        * Describe
        */
        void KillTimer(char id);
```

响应定时器消息：

在消息映射表中使用 MSG_WM_TIMER_EX。参见：SGifPlayer.h

```
        SOUI_MSG_MAP_BEGIN()
            MSG_WM_TIMER_EX(OnTimer)    //定时器消息
            MSG_WM_PAINT_EX(OnPaint)    //窗口绘制消息
            MSG_WM_SHOWWINDOW(OnShowWindow)//窗口显示状态消息
        SOUI_MSG_MAP_END()
```

如果在一个窗口中必须要创建使用 32 位的定时器 ID，在 SOUI 中可以使用另一个接口来实现:

```
        /**
        * SetTimer2
        * @brief    利用函数定时器来模拟一个兼容窗口定时器
        * @param    UINT_PTR id --  定时器 ID
        * @param    UINT uElapse --  延时(MS)
        * @return   BOOL
        *
        * Describe  由于 SetTimer 只支持 0-127 的定时器 ID，SetTimer2 提供设置其它
timerid
        *          能够使用 SetTimer 时尽量不用 SetTimer2，在 Kill 时效率会比较低
        */
        BOOL SetTimer2(UINT_PTR id,UINT uElapse);
```

```
        /**
        * KillTimer2
        * @brief    删除一个 SetTimer2 设置的定时器
        * @param    UINT_PTR id --  SetTimer2 设置的定时器 ID
        * @return   void
        *
        * Describe  需要枚举定时器列表
        */
        void KillTimer2(UINT_PTR id);
```

响应定时器:

使用 SWindow::SetTimer2 创建的定时器,最后会通过一个消息:WM_TIMER2 来分发到 SWindow。

```
#define WM_TIMER2    (WM_USER+5432)    //定义一个与 HWND 定时器兼容的 SOUI 定时器

#define MSG_WM_TIMER2(func) \
    if (uMsg == WM_TIMER2) \
{ \
    SetMsgHandled(TRUE); \
    func(wParam); \
    lResult = 0; \
    if(IsMsgHandled()) \
    return TRUE; \
}
```

在应用程序中使用定时器

前面两种定时器都是在控件开发的时候使用定时器的方法。而在应用层，还可以为宿主窗口直接使用::SetTimer 或者宿主窗口的基类：CSimpleWnd::SetTimer 来创建定时器**（注意最高位必须是 0）**。

在响应这类定时器时，一样可以在宿主窗口的消息映射表中使用 MSG_WM_TIMER 来响应定时器消息。但是需要注意的是，这个映射宏会截获所有分发给宿主窗口的定时器，如果不是自己创建的定时器，则需要继续交给基类处理。

可以调用：SetMsgHandled(FALSE); 或者：SHostWnd::OnTimer(UINT_PTR idEvent); 实现。

4.12 在 SOUI 中消息通讯

SOUI 是一套基于 Win32 SDK 的窗口开发的一套 DirectUI 框架。在 SOUI 中除了有真窗口使用窗口消息通讯机制外，还有 SOUI 控件之间的通讯，及控件的事件处理等。

- **真窗口消息通讯**

因此可以使用::SendMessage 这个 API 来与宿主窗口通讯。在任意一个地方只要获取到了 SOUI 的宿主窗口句柄就可以向该窗口发消息。

发消息以后可以在主界面的真窗口的消息映射表中响应各种自定义消息（如下）：

```
#define WM_MYMSG (WM_USER+100)
    LRESULT OnMyMsg(UINT uMsg,WPARAM wp,LPARAM lp,BOOL & bHandled)
    {
        return 0;
    }
    //HOST 消息及响应函数映射表
    BEGIN_MSG_MAP_EX(CMainDlg)
        MESSAGE_HANDLER(WM_MYMSG,OnMyMsg)
        MSG_WM_CREATE(OnCreate)
        MSG_WM_INITDIALOG(OnInitDialog)
        MSG_WM_DESTROY(OnDestory)
        MSG_WM_CLOSE(OnClose)
        MSG_WM_SIZE(OnSize)
        MSG_WM_COMMAND(OnCommand)
        MSG_WM_SHOWWINDOW(OnShowWindow)
        CHAIN_MSG_MAP(SHostWnd)
        REFLECT_NOTIFICATIONS_EX()
    END_MSG_MAP()
```

注意上面代码的红色部分。有 WTL 开发经验的朋友应该已经看出来了，SOUI 处理真窗口消息的形式和 WTL 完全一样。

- **SOUI 控件通讯**

我们知道，在 win32 编译中，要与一个控件（窗口）通讯能用 SendMessage(PostMessage)发送一个消息给目标窗口，目标窗口收到后进行处理。那么问题来了，如何向一个 SOUI 窗口发消息？

SOUI 的窗口类和 MFC 的窗口类很像，和 MFC 使用 SendMessage(PostMessage）发消息类似，在 SOUI 中也可以使用 SWindow::SSendMessage 来向目标窗口发送一个消息来通讯，但不支持 PostMessage，目标窗口在 SOUI 窗口的消息映射表中响应发送过来的消息。下面是一个内置控件 STabCtrl 的消息映射表：

```
    SOUI_MSG_MAP_BEGIN()
        MSG_WM_PAINT_EX(OnPaint)
        MSG_WM_DESTROY(OnDestroy)
        MSG_WM_LBUTTONDOWN(OnLButtonDown)
        MSG_WM_MOUSEMOVE(OnMouseMove)
        MSG_WM_MOUSELEAVE(OnMouseLeave)
        MSG_WM_KEYDOWN(OnKeyDown)
    SOUI_MSG_MAP_END()
```

和真窗口的映射表使用 WTL 的映射宏不一样，SOUI 窗口的映射宏使用 SOUI_MSG_MAP_BEGIN 和 SOUI_MSG_MAP_END 来构造消息处理函数，但是映射表中的消息映射项基本和 WTL 的映射形式是一样的（注意个别消息是经过重定义的，典型的如 WM_PAINT 消息，在 SOUI 中需要使用 MSG_WM_PAINT_EX 来处理）。

- **SOUI 的事件机制**

此外 SOUI 中控件要发出事件交给应用层处理使用的是一套事件机制。

每一个事件有对应一个 EventArg 类，事件在控件中使用 FireEvent 启动事件路由，应用程序可以在事件响应映射表中对各种事件统一处理，也可以使用 subscribeEvent 来直接订阅特定 SOUI 窗口的一个事件，直接将事件与事件处理函数关联。这一部分请参考前面相关章节。

4.13 使用窗口的 cache 属性加速 SOUI 的渲染

内容渲染速度是决定一个 UI 成败的关键。无论 UI 做得多华丽，没有速度都没有意义。

在 MFC，WTL 等开发框架下，每个控件都是一个窗口，窗口只需要画前景，背景。因为窗口之间的内容不需要做混合，一个子窗口的一次刷新只涉及该窗口本身，和其它窗口无关，因此这样效率很高。但是美中不足在于，窗口之间内容是孤立的，要想不同窗口之间的内容更协调，对美术有很高的要求，同时也基本只能适应个别窗口大小相对固定的场景。

要使 UI 更加漂亮，最简单有效的方法就是采用 alpha 混合的方式将各层的窗口相互混合，这样窗口之间没有了边界的感觉，从而使得 UI 更像一个整体，这也是现代 UI 的主要实现方式。

想像一下，如果一个最上层的子窗口内容发生了变化，UI 系统如何把窗口内容绘制出来呢？

首先需要通知 UI 系统的渲染对象说我的窗口显示区域内容发生了变化，请求重绘。

UI 系统拿到渲染对象后，对渲染范围设定剪裁区（clip），防止刷新那些不需要重绘的区域。

UI 系统从最顶层的窗口开始依次绘制每一层的和剪裁区重叠的窗口的内容（不同窗口之间绘制内容的混合由绘制方法定义，例如图片可能是利用 alpha 通道）。

为了防止绘制过程中出现闪烁，一般上面绘制都是在一个内存绘图设备（如内存 DC）上进行。在全部绘制完成后，将绘制内容再呈现到屏幕。

那么问题来了：一般子窗口会比父窗口小，子窗口请求重绘时，要执行父窗口的绘制函数，如果父窗口没有根据当前的剪裁范围来确定自己的绘制范围，那么尽管有剪裁区限制了不会绘制超出边界，在实际执行过程中，超出剪裁区的绘制行为也可能极大的影响 UI 的性能。

如何解决这个问题呢？

解决的方法可以有很多，比如在每个窗口的绘制方法都实现只绘制在剪裁区的那部分（实际上实现起来非常麻烦，附加的剪裁区计算也可能成为新的绘制瓶颈）；又比如每个窗口都保留背景窗口的内容，当窗口需要刷新时，直接直接把背景内容拿过来在上面画。

第二种方法看起来效率是最高的，但是实际上可能并不高：虽然在绘制自己的内容时速度快了，问题是自己更新以后，还需要同时更新它的上层窗口的缓存，可能导致内存开销大，效率也不高。

在 SOUI 中，我们采用了第三种方法：给窗口加一个 cache 属性。

当 cache=0 时，窗口的绘制和前面提到的流程一样，只是如果窗口比较大，而需要一时半会的区域比较小时，速度慢一点。

如果 cache=1，则窗口绘制的内容会被保存到一张缓存位图上，当其它窗口刷新并导致该窗口重绘时，直接从缓存中复制出来就可以了。

假定一个窗口使用一张小图片进行九宫格拉伸绘制出来，绘制整个窗口由于拉伸中的插值等一系列的计算，绘制可能需要消耗大量的 CPU 时间，而有了缓存以后，只在第一次绘制的时候使用图片绘制方法进行绘制，其它时间都是直接从缓存复制，效率能够极大的提高。

很显然，cache 属性在提高渲染效率的同时，也需要开辟一块缓存来保存绘制内容，也就是常见的空间换时间的方法。因此如果绘制本身速度就足够快，那么完全可以不使用缓存（比如给一个窗口填充一个纯色的矩形）。

总结：

cache 可以提高绘制速度，但是是以更高的内存占用为代价的，应该只在需要的地方使用（例如顶层背景窗口）。

4.14 在 SOUI 中实现 PreTranslateMessage

在 MFC 中，通常可以通过重载 CWnd::PreTranslateMessage 这样一个虚函数来实现对一些窗口消息的预处理。多用于 tooltip 的显示控制。

在 SOUI 中也实现了类似的机制。

要在 SOUI 中实现 PreTranslateMessage，我们首先需要实现一个接口：

```
struct IMessageFilter
{
    virtual BOOL PreTranslateMessage(MSG* pMsg) = 0;
};
```

可以看出，实现这个接口和在 MFC 中重载 PreTranslateMessage 是相同的道理。

和 MFC 中只需要重载这个接口不同，在 SOUI 中，除了需要实现 IMessageFilter 外，还需要向当前的 MessageLoop 注册该 IMessageFilter。

```
class SOUI_EXP SMessageLoop
{
public:
    SArray<IMessageFilter*> m_aMsgFilter;

    // Message filter operations
    BOOL AddMessageFilter(IMessageFilter* pMessageFilter);

    BOOL RemoveMessageFilter(IMessageFilter* pMessageFilter);
    //...
};
```

上面是 SMessageLoop 两个和 IMessageFilter 相关的方法。

SMessageLoop::AddMessageFilter 向当前的 message loop 注册一个 IMessageFilter；

SMessageLoop::RemoveMessageFilter 则向当前的 message loop 注销一个 IMessageFilter

剩下的问题就是如何获得当前的 MessageLoop 了。

在 SHostWnd 或者 SHostDialog 中可以调用 SHostWnd::GetMsgLoop()方法获得。

在 SWindow 中，则可以调用 SWindow::GetContainer()->GetMsgLoop()获得。

使用示例可以参考 SDropDownWnd 的实现。

```
class SOUI_EXP SDropDownWnd : public SHostWnd, public IMessageFilter
{
//...
};
```

4.15 提高 SOUI 应用程序渲染性能的三种武器

SOUI 是一套 100%开源的基于 DirectUI 的客户端开发框架。

基于 DirectUI 设计的 UI 虽然 UI 呈现的效果可以很炫，但是相对于传统的 win32 应用程序中每个控件一个窗口句柄的形式，渲染效率是一个很重要的问题。

在 SOUI 系统中提供了三种武器可以用来提高渲染效率：

第一种武器：选择更高效的渲染引擎

渲染引擎提供文字，几何图形，图像的在缓存上的绘制功能。在 SOUI 系统中，渲染引擎是一个独立的模块，它不依赖于 SOUI 系统中的其它模块。

在 SOUI 系统中已经内置了基于 skia 及 GDI 两种框架的渲染模块(skia 即在 google 的 chrome 及 android 中使用的渲染引擎）。直观的比较采用两种不同渲染引擎 demo 中动画的流畅度就可以知道基于 skia 的渲染引擎速度要比基于 GDI 的快不少。（GDI 慢的原

因可能是因为在 GDI 原生不支持 alpha 通道，而在实现的过程中采用 alphablend 模拟 alpha 时导致性能损失）

如果用户觉得内置的渲染引擎还是不足以满足自己的需求，还可以选择自己实现新的渲染引擎，如基于 Direct2D, cairo, agg 等渲染引擎。

第二种武器：绘制缓存

一个窗口中的呈现的内容很多时候都是固定的，特别是当窗口大小不变的时候。窗口中呈现的内容可能是经过复杂计算获得的（如图像的九宫格切分，拉伸等），如果每次刷新都重新计算显示渲染效率可能下降，特别是当窗口还比较大的时候。

为了解决这个问题，在 SOUI 系统中，我们为 SWindow 提供了一个 cache 属性，cache=“1”时，在窗口中绘制的内容会被自动缓存，下次刷新时，自动从缓存中提取数据，从而大大加速绘制过程。

第三种武器：非背景混合技术

DirectUI 炫酷的效果是依赖于各窗口之间的相互混合实现的（alphablend)，但是有些时候一个窗口它可能有自己固定的背景，或者前景完全覆盖窗口而不需要背景。如果这个窗口刷新非常频繁，那么每次刷新都先通知各级父窗口刷新获得背景再做混合将是很大的性能损失（如视频播放窗口）。

为了解决这个问题，在 SOUI 系统中(ver:1.3.0.1)为 SWindow 实现了一个新的属性：bkgndBlend，该属性默认为“1”，代表刷新时使用背景混合。如果该属性为“0”，则该窗口刷新的时候直接刷新自己，而不请求父窗口刷新背景，最终提高 UI 的渲染效率。

4.16 在 SOUI 中使用分层窗口

从 Windows 2K 开始，MS 为 UI 开发引入了分层窗口这一窗口风格。使用分层窗口，应用程序的主窗口可以是半透明，也可以是逐点半透明（即每一个像素点的透明度可以不同）。

可以说，正是因为有了分层窗口，在 Windows 上开发的应用程序的 UI 才真正炫起来。

在 UI 的主窗口上加一个分层窗口的风格对于一个稍有点 UI 开发经验的程序员来说是非常简单的，本篇要说的是在 SOUI 的窗口系统中实现 SOUI 的分层窗口。

正如使用系统的窗口已经可以实现很漂亮的 UI，我们还是会需要 DirectUI 这样的 UI 开发技术；有了系统的分层窗口，我们还是会需要在 DirectUI Window 之间的分层窗口技术。

一个 DirectUI 系统的分层窗口有什么用呢？

分层窗口和一般窗口的关键区别在在于：一个分层窗口是一个相对独立的渲染层，它能将它的子窗口渲染到这个窗口上，当所有分层窗口都渲染完成后，再按照分层窗口的 zorder

顺序通过不同的 alpha 混合技术进行混合；而一般的 DirectUI 系统只有一个渲染层，所有窗口按照 zorder 顺序依次渲染到这个渲染层上，缺少了不同渲染层的概念。

一个简单的例子：业务可能希望一个功能面板能够整体半透明，该功能面版可能包含很多子窗口。

如果没有分层窗口特性，在 DirectUI 中实现类似的需求很很复杂（如分别指定每一个子窗口的半透明）而且可能导致性能下降、内存消耗的提高等问题。

如果有了分层窗口，实现类似的需求就非常简单了：只需要为该功能面板指定一个透明度就行了。在渲染的时候，功能面板的子窗口会以正常的渲染方式渲染到分层窗口的缓存上，全部渲染完成后，再整体和上一个渲染层混合，从而得到期望的效果。

在 SOUI 里如何使用分层窗口：

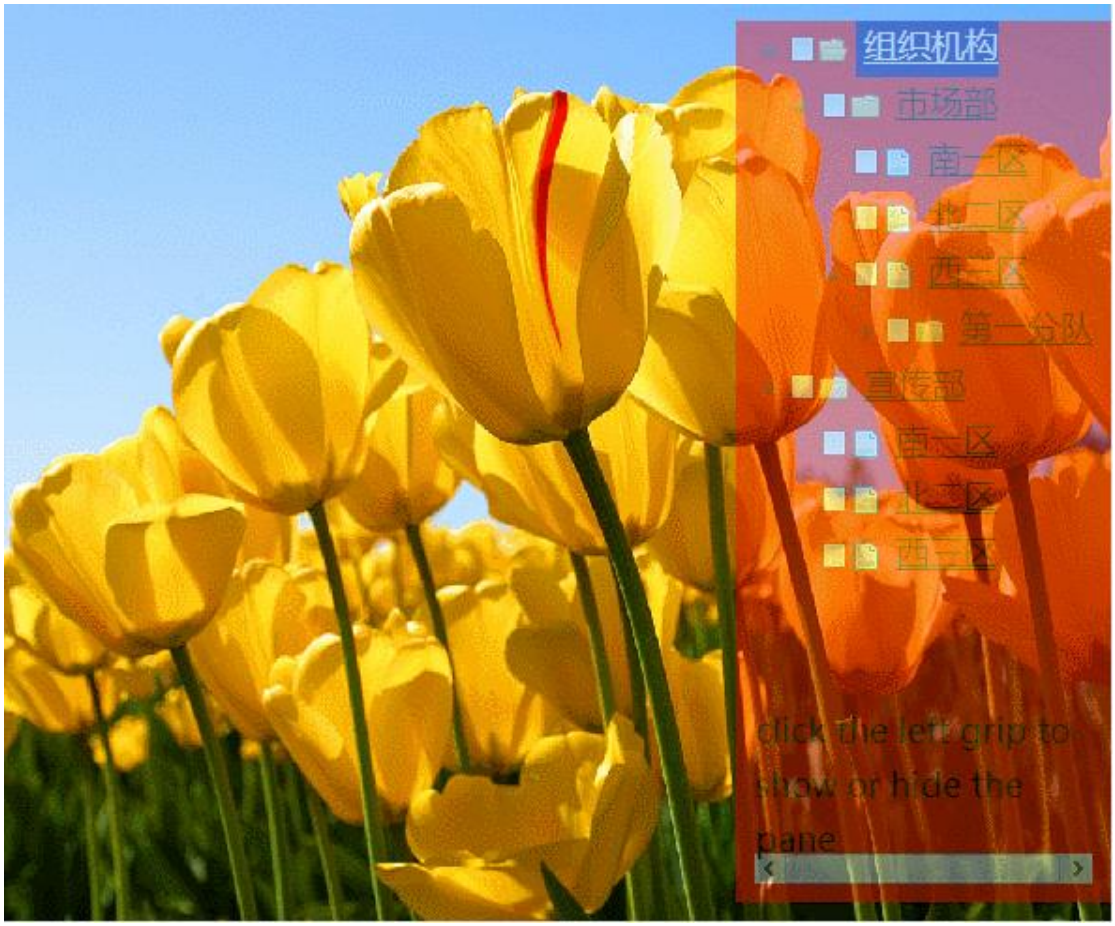
SOUI 采用 XML 定义 UI，要定义分层窗口只需要一个 layeredWindow="1"的属性即可。

XML 定义：

```
     <window pos="0,0,-0,-0" clipClient="1" skin="skin_bkgnd" >
      <flywnd pos="-210,10,-0,-10" posEnd="-10,10,@210,-10" alpha="100"
layeredWindow="1">
        <toggle pos="0,|-15,@10,@30" skin="_skin.sys.tree.toggle"
name="switch" cursor="hand" tip="click me to show the animator that show or
hide the pane"></toggle>
        <window pos="10,0,-0,-0" colorBkgnd="#ff0000">
          <treectrl pos="10,0,-10,-10"  name="mytree2" itemHeight="30"
iconSkin="skin_tree_icon" checkBox="1" font="underline:1">
            <item text="组织机构" img="0" selImg="1"  expand="0">
              <item text="市场部" img="0" selImg="1">
                <item text="南一区" img="2"/>
                <item text="北二区" img="2"/>
                <item text="西三区" img="2">
                  <item text="第一分队" img="0" selImg="1" expand="0">
                    <item text="张三组" img="2"/>
                    <item text="李四组" img="2"/>
                    <item text="王五组" img="2"/>
                  </item>
                </item>
              </item>
            </item>
            <item text="宣传部" img="0" selImg="1" expand="0">
              <item text="南一区" img="2"/>
              <item text="北二区" img="2"/>
              <item text="西三区" img="2"/>
            </item>
          </treectrl>
```

```
          <text pos="10,-100,-0,@100" multilines="1">click the left grip to
\nshow or hide the \npane</text>
        </window>
      </flywnd>
    </window>
```

显示效果：



4.17 在 SOUI 中的控件注册机制

Win32 编程中，用户需要一个新控件时，需要向系统注册一个新的控件类型。注册以后，调用::CreateWindow 时才能根据标识控件类型的字符串创建出一个新的控件窗口对象。为了能够从 XML 描述的字符串中创建出需要的控件对象，和 Win32 类似，在 SOUI 中要创建一个新的控件也同样需要向 SOUI 系统注册新的控件类。

从 demo.cpp 的 main 中我们可以看到类似如下的控件注册控件的代码：

```
        //向 SApplication 系统中注册由外部扩展的控件及 SkinObj 类
        SWkeLoader wkeLoader;
        if(wkeLoader.Init(_T("wke.dll")))
        {
```

```
            theApp->RegisterWndFactory(TplSWindowFactory<SWkeWebkit>());//注册
WKE 浏览器
        }
        theApp->RegisterWndFactory(TplSWindowFactory<SGifPlayer>());//注册
GIFPlayer
        theApp->RegisterSkinFactory(TplSkinFactory<SSkinGif>());//注册 SkinGif
        theApp->RegisterSkinFactory(TplSkinFactory<SSkinAPNG>());//注册
SSkinAPNG
        theApp->RegisterSkinFactory(TplSkinFactory<SSkinVScrollbar>());//注册纵
向滚动条皮肤
        theApp->RegisterWndFactory(TplSWindowFactory<SIPAddressCtrl>());//注册
IP 控件
        theApp->RegisterWndFactory(TplSWindowFactory<SPropertyGrid>());//注册属
性表控件
        theApp->RegisterWndFactory(TplSWindowFactory<SChromeTabCtrl>());//注册
ChromeTabCtrl
        theApp->RegisterWndFactory(TplSWindowFactory<SIECtrl>());//注册 IECtrl
        theApp->RegisterWndFactory(TplSWindowFactory<SChatEdit>());//注册
ChatEdit
        theApp->RegisterWndFactory(TplSWindowFactory<SScrollText>());//注册
SScrollText
        if(SUCCEEDED(CUiAnimation::Init()))
        {

theApp->RegisterWndFactory(TplSWindowFactory<SUiAnimationWnd>());//注册动画控
件
        }
        theApp->RegisterWndFactory(TplSWindowFactory<SFlyWnd>());//注册飞行动画
控件
        theApp->RegisterWndFactory(TplSWindowFactory<SFadeFrame>());//注册渐显
隐动画控件
        theApp->RegisterWndFactory(TplSWindowFactory<SRadioBox2>());//注册渐显
隐动画控件
```

上面代码中即有新皮肤对象的注册也有窗口控件的注册，这种注册控件方式看起来有点怪异，但使用起来也还算是简单，只需要一行代码（实际上是从著名的游戏 GUI CEGUI 借鉴过来的）。

以注册窗口控件为例，下面我来解释一下为什么会有这样一种控件注册方式。

先看看 TplSWindowFactory 模板类的实现：

```
    class SWindowFactory
    {
    public:
```

```
    virtual ~SWindowFactory() {}
    virtual SWindow* NewWindow() = 0;
    virtual LPCWSTR SWindowBaseName()=0;

    virtual const SStringW & getWindowType()=0;

    virtual SWindowFactory* Clone() const =0;
};

template <typename T>
class TplSWindowFactory : public SWindowFactory
{
public:
    //! Default constructor.
    TplSWindowFactory():m_strTypeName(T::GetClassName())
    {
    }
    LPCWSTR SWindowBaseName(){return T::BaseClassName();}

    // Implement WindowFactory interface
    virtual SWindow* NewWindow()
    {
        return new T;
    }

    virtual const SStringW & getWindowType()
    {
        return m_strTypeName;
    }

    virtual SWindowFactory* Clone() const
    {
        return new TplSWindowFactory();
    }
protected:
    SStringW m_strTypeName;
};
```

TplSWindowFactory 从 SWindowFactory 派生，自动为模板参数中指定的类型生成一个 SWindowFactory 对象。

SWindowFactory 有什么用呢？SWindowFactory 最核心的用途就在于它的方法：

SWindow * SWindowFactory::NewWindow();

该方法说来很简单，不过是根据不同的字符串从堆上分配一个 SWindow 对象，但是在 C++中我们需要满足一条基本原则：内存在哪个模块中分配（从哪个模块的堆上分配）就必须被哪个模块释放。

在 SOUI 中，通常情况下控件都是由 SOUI.DLL 这个模块来分配内存的，也就是说 SWindow * SWindowFactory::NewWindow()只会在 SOUI 模块中调用。

那么生成的对象在哪里释放呢？

这就需要看一下 SWindow 或者 SSkinObjBase 这两个类的实现了。

```
class SOUI_EXP SWindow : public SObject
    , public SMsgHandleState
    , public TObjRefImpl2<IObjRef,SWindow>
{
    SOUI_CLASS_NAME(SWindow, L"window")
    //....
};
class SOUI_EXP SSkinObjBase : public TObjRefImpl<ISkinObj>
{
   //......
};
```

SWindow 和 SSkinObjBase 实际上都派生自 TObjRefImpl 这个模板类。下面看一下这个类的实现:

```
template<class T>
class TObjRefImpl :  public T
{
public:
    TObjRefImpl():m_cRef(1)
    {
    }
    virtual ~TObjRefImpl(){
    }
    //!添加引用
    /*!
    */
    virtual void AddRef()
    {
        InterlockedIncrement(&m_cRef);
    }
    //!释放引用
    /*!
    */
    virtual void Release()
```

```
    {
        InterlockedDecrement(&m_cRef);
        if(m_cRef==0)
        {
            OnFinalRelease();
        }
    }
    //!释放对象
    /*!
    */
    virtual void OnFinalRelease()
    {
        delete this;
    }
protected:
    volatile LONG m_cRef;
};

template<class T,class T2>
class TObjRefImpl2 :  public TObjRefImpl<T>
{
public:
    virtual void OnFinalRelease()
    {
        delete static_cast<T2*>(this);
    }
};
```

TObjRefImpl 一个重要的虚函数是 void OnFinalRelease(){delete this;}

由于 SWindow 及 SSkinObjBase 是在 SOUI 模块中实现的，因此派生自这两个类的新的控件类及皮肤类最后都将在 SOUI 模块中被释放，从而保证了对象内存的分配、释放在一个模块。

这也就是说不管是 SOUI 模块内实现的控件还是在应用层扩展的控件一般情况下都应该向 SOUI 系统注册并经由 SOUI 内核来实现对象的分配与释放。

那么问题来了，是不是所有新控件都必须向系统注册呢？

当然不是的，注意 OnFinalRelease 是一个虚函数，只需要在新的控件类中增加一个 void OnFinalRelease(){delete this;}

就可以把控件内存的释放转移到应用层的模块了。如此你就可以在自己的模块中直接使用 new 来创建新控件了。(可以参考属性表控件中部分子控件的实现)

4.18 在 SOUI 中使用代码向窗口中插入子窗口

使用 SOUI 开发客户端 UI 程序，通常也推荐使用 XML 代码来创建窗口，这样创建的窗口使用方便，当窗口大小改变时，内部的子窗口也更容易协同变化。

但是最近不断有网友咨询如何使用代码来创建 SOUI 子窗口，特此在这里统一解答。

要回答这个问题，首先要了解 SOUI 窗口创建及布局的流程。

先从 swnd.cpp 里抄一段创建子窗口的代码：

```
BOOL SWindow::CreateChildren(pugi::xml_node xmlNode)
    {
        TestMainThread();
        for (pugi::xml_node xmlChild=xmlNode.first_child(); xmlChild;
xmlChild=xmlChild.next_sibling())
        {
            if(xmlChild.type() != pugi::node_element) continue;

            if(_wcsicmp(xmlChild.name(),KLabelInclude)==0)
            {//在窗口布局中支持 include 标签
                SStringW strSrc = S_CW2T(xmlChild.attribute(L"src").value());
                pugi::xml_document xmlDoc;
                SStringTList strLst;

                if(2 == ParseResID(strSrc,strLst))
                {
                    LOADXML(xmlDoc,strLst[1],strLst[0]);
                }else
                {
                    LOADXML(xmlDoc,strLst[0],RT_LAYOUT);
                }
                if(xmlDoc)
                {
                    CreateChildren(xmlDoc.child(KLabelInclude));
                }else
                {
                    SASSERT(FALSE);
                }
            }else if(!xmlChild.get_userdata())//通过 userdata 来标记一个节点是否可以
忽略
            {
                SWindow *pChild =
SApplication::getSingleton().CreateWindowByName(xmlChild.name());
                if(pChild)
                {
```

```
                InsertChild(pChild);
                pChild->InitFromXml(xmlChild);
            }
        }
    }
    return TRUE;
}
```

这个函数的功能是为 this 从 XML 中创建它的子窗口，主要注意代码中红色部分。

其中第 30 行是从 SOUI 的窗口类厂根据 XML 结点名 new 出一个窗口对象。

第 33 行把创建出来的窗口插入到 this 的子窗口链表里去。

而第 34 行的功能是从子窗口对应的 XML 结点初始化子窗口属性。

很多网友以为只要完成上面步骤就应该可以显示窗口了，但结果确大失所望。

SOUI 一个重要特点就是能够自动布局，这个过程的秘密就在于 SWindow::OnRelayout 方法。

```
void SWindow::OnRelayout(const CRect &rcOld, const CRect & rcNew)
{
    SWindow *pParent= GetParent();
    if(pParent)
    {
        pParent->InvalidateRect(rcOld);
        pParent->InvalidateRect(rcNew);
    }else
    {
        InvalidateRect(m_rcWindow);
    }
    SSendMessage(WM_NCCALCSIZE);//计算非客户区大小
    UpdateChildrenPosition();   //更新子窗口位置
    CRect rcClient;
    GetClientRect(&rcClient);

SSendMessage(WM_SIZE,0,MAKELPARAM(rcClient.Width(),rcClient.Height()));
}
```

当 this 窗口位置改变后都会进入 OnRelayout 方法。

注意函数第 15 行：UpdateChildrenPosition()；这个调用的功能就是将 this 的所有子控件按照控件中定义的布局属性来自动布局。

要实现窗口的布局，除了调用父窗口的 UpdateChildrenPosition()方法外，当然也可以使用 SWindow::Move 方法直接修改窗口位置。

看到这里大家应该已经明白是什么原因了。

当然上述方法是 SOUI 中使用的窗口创建及布局方法，具体到应用程序中使用代码创建控件还有一个地方需要注意。

关键的问题是在 SOUI 系统中编译默认使用 MT（静态链接）方式来链接 CRT。

MT 方式编译时使用 CRT 分配内存是内存是属性调用的模块（DLL）的，内存的释放也因此必须在该模块内执行。

如果用户直接使用 new 来分配一个窗口对象，并把它插入到 SOUI 窗口链表中，在窗口释放的时机是在 SWindow::OnFinalRelease()中（实际是基类 TObjRefImpl2<IObjRef,SWindow>的方法)。

SWindow 的代码段是编译在 soui.dll 中，因此默认执行内存释放的代码是在 soui.dll 中，从而导致程序崩溃。

要解决这个问题有两种方法：

对于系统控件，用户应该使用 SApplication::getSingleton().CreateWindowByName(xmlChild.name());来创建窗口对象。

而对于用户自己派生实现的扩展窗口类并没有向 SOUI 的窗口类厂注册时，只能使用 new 方法来创建窗口对象。注意 SWindow 的基类：TObjRefImpl2<IObjRef,SWindow>

```
template<class T>
class TObjRefImpl : public T
{
public:
    TObjRefImpl():m_cRef(1)
    {
    }
    virtual ~TObjRefImpl(){
    }
    //!添加引用
    /*!
    */
    virtual long AddRef()
    {
        return InterlockedIncrement(&m_cRef);
    }
    //!释放引用
    /*!
    */
    virtual long Release()
    {
        long lRet = InterlockedDecrement(&m_cRef);
        if(lRet==0)
```

```
        {
            OnFinalRelease();
        }
        return lRet;
    }
    //!释放对象
    /*!
    */
    virtual void OnFinalRelease()
    {
        delete this;
    }
protected:
    volatile LONG m_cRef;
};

template<class T,class T2>
class TObjRefImpl2 :  public TObjRefImpl<T>
{
public:
    virtual void OnFinalRelease()
    {
        delete static_cast<T2*>(this);
    }
};
```

注意代码中的 OnFinalRelease，它是一个虚方法。因此对于使用 new 创建的窗口对象，只需要在窗口类中抄一段代码如下即可：

```
class myctrl : public SWindow
{
SOUI_CLASS_NAME(myctrl,L"myctrl")
public:
//...
virtual void OnFinalRelease() {delete this;}
//...
};
```

4.19 在 SOUI 中使用 LUA 脚本开发界面

像写网页一样做客户端界面可能是很多客户端开发的理想。

做好一个可以实现和用户交互的动态网页应该包含两个部分：使用 html 做网页的布局，使用脚本如 vbscript,javascript 做用户交互的逻辑。当需求变化时，只需要在服务端把相关代码调整一下，用户即可看到新的内容（界面）。

传统的客户端程序开发流程和网页开发可能完全不同。

首先是界面的布局，在老式的界面布局过程中，程序员先在界面上放好各种控件，然后需要自己通过相应的代码来维护界面在不同状态下控件的显示状态及位置。当界面中元素很多时，单纯布局的代码可能就会非常的复杂。

界面做好了，程序员需要增加代码响应界面中 UI 元素对应的逻辑。如逻辑比较固定，一旦做好就不需要变化时，这样的实现的程序效率可能更高。然而做客户端的人可能都经历过修改界面的问题。简单的控件坐标调整还好办，如果程序的界面风格甚至整个程序的运行逻辑都要变化时，使用传统的开发方式实现的客户端程序更新起来会非常困难，有时甚至变得不可能实现。

近年来，在各种新兴的界面开发库中使用 XML 来描述布局已经得到了广泛的应用。使用 XML 描述布局的基本出发点应该和使用 HTML 描述网页布局类似，关键是解决界面元素的批量创建及元素间位置的自动布局及 UI 大小变化后的 UI 元素自适应。然而到目前为止在 UI 库中使用脚本来实现逻辑控制的还不多见，据我所知，在流行的 UI 库中只有 qt, bolt 这两个项目支持。无论是 QT 还是 Bolt，这两个都是重量级的 UI 库，而且 QT 中使用脚本（Bolt 不了解）的方式也和网页开发想去甚远。

尽管在 DuiEngine 时代的代码库中就包含了一个 LUA 脚本模块，但那个时候的脚本模块也仅限于简单的 LUA 脚本和 C++控件代码之间简单的相互调用，还没有形成一个清晰的应用模式。

经过 2015 年春节期间对脚本模块的重构，终于在 SOUI 中实现了和网页开发基本一样的界面开发流程。

先看一个 SOUI DEMO 中 100%使用 LUA 脚本实现的小游戏效果：

下面我们再看看实现上述效果用到的代码：

在 SOUI 中，脚本模块和其它如渲染模块等一样是使用类似插件形式实现的，当然首先需要我们在程序的入口加载 LUA 脚本模块：

```
int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE /*hPrevInstance*/,
LPTSTR /*lpstrCmdLine*/, int /*nCmdShow*/)
{
    //必须要调用 OleInitialize 来初始化运行环境
    HRESULT hRes = OleInitialize(NULL);
    SASSERT(SUCCEEDED(hRes));
//    LoadLibrary(L"E:\\soui.taobao\\richedit\\Debug\\riched20.dll");

    int nRet = 0;

    SComMgr *pComMgr = new SComMgr;
    {
    //...
        //定义一个唯一的 SApplication 对象，SApplication 管理整个应用程序的资源
        SApplication *theApp=new SApplication(pRenderFactory,hInstance);

#ifdef DLL_CORE
        //加载 LUA 脚本模块，注意，脚本模块只有在 SOUI 内核是以 DLL 方式编译时才能使用。
        bLoaded=pComMgr->CreateScrpit_Lua((IObjRef**)&pScriptLua);
        SASSERT_FMT(bLoaded,_T("load interface [%s]
failed!"),_T("scirpt_lua"));
        theApp->SetScriptFactory(pScriptLua);
#endif//DLL_CORE

        //加载全局资源描述 XML
        theApp->Init(_T("xml_init"));

        {
            //创建并显示使用 SOUI 布局应用程序窗口,为了保存窗口对象的析构先于其它对象，把它
们缩进一层。
            CMainDlg dlgMain;
            dlgMain.Create(GetActiveWindow(),0,0,800,600);
            dlgMain.GetNative()->SendMessage(WM_INITDIALOG);
            dlgMain.CenterWindow();
            dlgMain.ShowWindow(SW_SHOWNORMAL);
            nRet=theApp->Run(dlgMain.m_hWnd);
        }
        //应用程序退出
        delete theApp;
```

```
    //...
    }
exit:
    delete pComMgr;
    OleUninitialize();

    return nRet;
}
```

其次我们需要在 XML 布局中的 SOUI 节点下增加一个 script 的子节点：

```
<SOUI trCtx="dlg_main" title="SOUI-DEMO version:%ver%" bigIcon="LOGO:32"
smallIcon="LOGO:16" width="600" height="400" appWnd="1" margin="5,5,5,5"
resizable="1" translucent="1" alpha="255">
  <skin>
    <!--局部 skin 对象-->
    <gif name="gif_penguin" src="gif:gif_penguin"/>
    <apng name="apng_haha" src="apng:apng_haha"/>
  </skin>
  <style>
    <!--局部 style 对象-->
    <class name="cls_edit" ncSkin="_skin.sys.border" margin-x="2" margin-
y="2" />
  </style>
  <script src="lua:lua_test">
    <!--当没有指定 src 属性时从 cdata 段中加载脚本-->
    <![CDATA[
    function on_init(args)
        SMessageBox(0,T "execute script function: on_init", T "msgbox", 1);
    end
    function on_exit(args)
        SMessageBox(0,T "execute script function: on_exit", T "msgbox", 1);
    end
    function onEvtTest2(args)
        SMessageBox(0,T "onEvtTest2", T "msgbox", 1);
        return 1;
    end
    function onEvtTstClick(args)
        local txt3=SStringW(L"append",-1);
        local sender=toSWindow(args.sender);
        sender:GetParent():CreateChildrenFromString(L"<button
pos=\"0,0,150,30\" on_command=\"onEvtTest2\">lua btn 中文</button>");
        sender:SetVisible(0,1);
        return 1;
```

```
    end
    ]]>
  </script>
  <root class="cls_dlg_frame" cache="1" on_init="on_init" on_exit="on_exit">
    <caption pos="0,0,-0,30" show="1" font="adding:8">
      <icon pos="10,8" src="LOGO:16"/>
      <text class="cls_txt_red">SOUI-DEMO version:%ver%</text>
      <imgbtn id="1" skin="_skin.sys.btn.close"    pos="-45,0" tip="close"
animate="1"/>
      <imgbtn id="2" skin="_skin.sys.btn.maximize"  pos="-83,0" animate="1" />
      <imgbtn id="3" skin="_skin.sys.btn.restore"  pos="-83,0" show="0"
animate="1" />
      <imgbtn id="5" skin="_skin.sys.btn.minimize" pos="-121,0" animate="1"
/>
      <imgbtn name="btn_menu" skin="skin_btn_menu" pos="-151,2" animate="1"
/>
    </caption>
    <other/>
  </root>
</SOUI>
```

在 script 节点中，可以使用 src 属性来指定脚本所在的资源文件，也可以将脚本直接使用 cdata 写在 script 节点中，但是 src 中指定的脚本优先。

注意，上述 XML 中我们还在 root 结点中增加了两个新的属性：on_init, on_exit，这两个属性告诉程序在界面初始化完成及界面销毁前需要执行的两个脚本函数。

实现了上面两步，在 SOUI 中使用脚本的准备工作已经就绪。

为了实现上图的跑马机效果，首先我们需要在一个界面中使用 XML 实现基本的界面元素布局：

```
<?xml version="1.0" encoding="utf-8"?>
<include>
  <window size="full,full" name="game_wnd" on_size="on_canvas_size" id="300">
    <text pos="0,0,-0,@30" colorBkgnd="#cccccc" colorText="#ff0000"
align="center">SOUI + LUA 跑马机</text>
    <window pos="0,[0,-0,-50">
      <window pos="0,0,-64,-0" name="game_canvas" clipClient="1"
colorBkgnd="#ffffff">
        <!--比赛场地-->
        <gifplayer name="player_1" float="1" skin="gif_horse">
          <text pos="0,%1" colorText="rgb(255,0,0)" font="size:20">1</text>
        </gifplayer>
        <gifplayer name="player_2" float="1" skin="gif_horse">
          <text pos="0,0" colorText="rgb(255,0,0)" font="size:20">2</text>
```

```
      </gifplayer>
      <gifplayer name="player_3" float="1" skin="gif_horse">
        <text pos="0,0" colorText="rgb(255,0,0)" font="size:20">3</text>
      </gifplayer>
      <gifplayer name="player_4" float="1" skin="gif_horse">
        <text pos="0,0" colorText="rgb(255,0,0)" font="size:20">4</text>
      </gifplayer>
      <gifplayer name="flag_win" float="1" skin="gif_win" show="0"
id="400"/>
      <text pos="|0,0">赔率:</text>
      <text pos="[0,0,@20,[0" colorText="#ff0000" name="txt_rate">4</text>
      <hr pos="-1,0,-0,-0" mode="vertical" colorLine="#ff0000"/>
    </window>
    <window pos="[0,0,-0,-0">
      <window pos="0,%12.5,@64,@64" offset="0,-0.5" skin="img_coin" id="1"
tip="下注 1 号马" on_command="on_bet">0</window>
      <window pos="0,%37.5,@64,@64" offset="0,-0.5" skin="img_coin" id="2"
tip="下注 2 号马" on_command="on_bet">0</window>
      <window pos="0,%62.5,@64,@64" offset="0,-0.5" skin="img_coin" id="3"
tip="下注 3 号马" on_command="on_bet">0</window>
      <window pos="0,%87.5,@64,@64" offset="0,-0.5" skin="img_coin" id="4"
tip="下注 4 号马" on_command="on_bet">0</window>
    </window>
  </window>
  <window pos="10,[5,-0,-0" name="game_toolbar">
    <button name="btn_run" pos="0,|0,@100,@30" offset="0,-0.5" tip="run the
game" on_command="on_run">run</button>
    <text pos="]-5,0,@-1,-0" offset="-1,0">现有金币:</text>
    <text pos="-64,0,@64,-0" name="txt_coins" align="center">100</text>
  </window>
 </window>
</include>
```

在上述 XML 中，我们使用 gifplayer 控件定义了 4 匹马，但是我们并没有使用 pos 属性定义它的位置，因为它的位置是在变化的，我们需要使用脚本在控制。注意上述 XML 中为几个按钮控件指定的 on_command 属性。on_command 属性是窗口的点击事件属性，指定后可以执行脚本中对应的函数。

事实上在 SOUI 中每一个事件都定义了一个在 XML 中可以映射到脚本脚本函数的属性，这里简单介绍一下这个属性在哪里实现的：

```
  class SOUI_EXP EventCmd : public TplEventArgs<EventCmd>
  {
    SOUI_CLASS_NAME(EventCmd,L"on_command")
```

```
    public:
        EventCmd(SObject *pSender):TplEventArgs<EventCmd>(pSender){}
        enum{EventID=EVT_CMD};
    };
```

通过上面代码，可以发现，这个属性实际上就是这个事件类在 SOUI 中的字符串名字。

下面我们看看实现这个跑马机真正使用的 LUA 脚本代码：

```
win = nil;
tid = 0;
gamewnd = nil;
gamecanvas = nil;
players = {};
flag_win = nil;

coins_all = 100;    --现有资金
coins_bet = {0,0,0,0} --下注金额
bet_rate = 4;       --赔率
prog_max    = 200;    --最大步数
prog_all = {0,0,0,0} --马匹进度

function on_init(args)
    --初始化全局对象
    win = toHostWnd(args.sender);
    gamewnd = win:GetRoot():FindChildByNameA("game_wnd",-1);
    gamecanvas = gamewnd:FindChildByNameA("game_canvas",-1);
    flag_win = gamewnd:FindChildByNameA("flag_win",-1);
    players = {
                gamecanvas:FindChildByNameA("player_1",-1),
                gamecanvas:FindChildByNameA("player_2",-1),
                gamecanvas:FindChildByNameA("player_3",-1),
                gamecanvas:FindChildByNameA("player_4",-1)
                };
    --布局
    on_canvas_size(nil);

    math.randomseed(os.time());
    --SMessageBox(0,T "execute script function: on_init", T "msgbox", 1);
end

function on_exit(args)
    --SMessageBox(0,T "execute script function: on_exit", T "msgbox", 1);
```

```
end

function on_timer(args)
    if(gamewnd ~= nil) then

        local rcCanvas = gamecanvas:GetWindowRect2();
        local heiCanvas = rcCanvas:Height();
        local widCanvas = rcCanvas:Width();

        local rcPlayer =  players[1]:GetWindowRect2();
        local wid = rcPlayer:Width();
        local hei = rcPlayer:Height();

        local win_id = 0;
        for i = 1,4 do
            local prog = prog_all[i];
            if(prog<prog_max) then
                prog = prog + math.random(0,5);
                prog_all[i] = prog;
                local rc = players[i]:GetWindowRect2();
                rc.left = rcCanvas.left + (widCanvas-wid)*prog/prog_max;
                players[i]:Move2(rc.left,rc.top,-1,-1);
            else
                win_id = i;
                local rc = players[i]:GetWindowRect2();
                rc.left = rcCanvas.left + (widCanvas-wid);
                players[i]:Move2(rc.left,rc.top,-1,-1);
            end
        end
        if win_id ~= 0 then
            gamewnd:FindChildByNameA("btn_run",-1):FireCommand();
            coins_all = coins_all + coins_bet[win_id] * 4;
            gamewnd:FindChildByNameA("txt_coins",-
1):SetWindowText(T(coins_all));

            coins_bet = {0,0,0,0};

            local rcPlayer = players[win_id]:GetWindowRect2();
            local szFlag = flag_win:GetDesiredSize(rcPlayer);
            rcPlayer.right = rcPlayer.left + szFlag.cx;
            rcPlayer.bottom = rcPlayer.top + szFlag.cy;
            rcPlayer:OffsetRect(-szFlag.cx,-szFlag.cy/3);
```

```
            flag_win:Move(rcPlayer);
            flag_win:SetVisible(1,1);
            flag_win:SetUserData(win_id);

            for i= 1,4 do
                gamewnd:FindChildByID(i,-1):SetWindowText(T("0"));
            end
        end
    end
end

function on_bet(args)
    if tid ~= 0 then
        return 1;
    end
    local btn = toSWindow(args.sender);
    if coins_all >= 10 then
        id = btn:GetID();
        coins_bet[id] = coins_bet[id] + 10;
        coins_all = coins_all -10;
        btn:SetWindowText(T(coins_bet[id]));
        gamewnd:FindChildByNameA("txt_coins",-1):SetWindowText(T(coins_all));
    end
    return 1;
end
function on_canvas_size(args)
    if win == nil then
        return 0;
    end
    local rcCanvas =  gamecanvas:GetWindowRect2();
    local heiCanvas = rcCanvas:Height();
    local widCanvas = rcCanvas:Width();
    local szPlayer = players[1]:GetDesiredSize(rcCanvas);
    local wid = szPlayer.cx;
    local hei = szPlayer.cy;
    local rcPlayer = CRect(0,0,wid,hei);
    local interval = (heiCanvas - hei*4)/5;
    rcPlayer:OffsetRect(rcCanvas.left,rcCanvas.top+interval);
    for i = 1, 4 do
        local rc = rcPlayer;
        rc.left = rcCanvas.left + (widCanvas-wid)*prog_all[i]/prog_max;
        rc.right = rc.left+wid;
        players[i]:Move(rc);
```

```
        rcPlayer:OffsetRect(0,interval+hei);
    end
    local win_id = flag_win:GetUserData();
    if win_id ~= 0 then
        local rcPlayer = players[win_id]:GetWindowRect2();
        local szFlag = flag_win:GetDesiredSize(rcPlayer);
        flag_win:Move2(rcPlayer.left-szFlag.cx,rcPlayer.top-szFlag.cy/3,-1,-
1);
    end
    return 1;
end
function on_run(args)
    local btn = toSWindow(args.sender);
    if tid == 0 then
        prog_all = {0,0,0,0};
        on_canvas_size(nil);
        tid = win:setInterval("on_timer",200);
        btn:SetWindowText(T"stop");
        flag_win:SetVisible(0,1);
    else
        win:clearTimer(tid);
        btn:SetWindowText(T"run");
        tid = 0;
    end
    return 1;
end
function on_btn_select_cbx(args)
    local btn = toSWindow(args.sender);
    local cbxwnd = btn:GetWindow(2);--get previous sibling
    local cbx = toComboboxBase(cbxwnd);
    cbx:SetCurSel(-1);
end
```

在 on_init 中，我们获得必须的几个 UI 控件，并保存到 LUA 的全局变量中。

在 on_canvas_size 中，我们根据 UI 大小，自动调整 4 匹马的位置。

然后在几个按钮的事件响应函数中响应用户操作。

至此一个简单的跑马机效果就完成了。

也许有人觉得这个示例太简单，但请想象一下，采用类似的方法，以后 UI 及逻辑都可以在服务器控件，客户端就像网页一样每天都可以从服务器更新界面会是什么场景，那时客户端可真就成了客户了。

后记：在 SOUI 中实现和网页开发类似的脚本功能算是 SOUI 2015 年非常重要一次更新，本来早就想写一篇这样的博客来介绍，无奈由于原来使用的 lua_tinker 0.5c 版本导出 C++类到 LUA 还有这样那样的问题，一直没有找到合适的解决方案。

虽然也有比较成熟的方法如，luabind, tolua++等，但是总觉得太庞大，不适合 SOUI 这个非常轻量级的 UI 库。

这期间也尝试了如 luawrapper, fflua，以及网上找到的别人修改的各版本 lua_tinker，但都不理想。

使用其它导出方法就不说了，我也没有深入。使用 lua_tinker 主要的问题就是导出的子类不能正常调用基类的方法，这一点很头痛，虽然用变通的方法可以获得调用基类方法的能力，但是看起来有点背叛了 OOP 的思想。

直到今天才搞明白不能调用基类成员最主要的问题来自 C++对象的多继承，明白了这个问题才能规避问题，才敢把这个 LUA 模块拿出来讲解，也顺便把 SOUI 中使用的 LUA 内核和 5.1 升级到了 5.2.3。

4.20 导出 SOUI 对象到 LUA 脚本

LUA 是一种体积小，速度快的脚本语言。脚本语言虽然性能上和 C++这样的 Naitive 语言相比差一点，但是开发速度快，可以方便的更新代码等，近年来受到了越来越多开发者的重视。

在 SOUI 框架中，我把脚本模块参考 CEGUI 抽象出一个独立的脚本接口，方便实现各种脚本语言的对接。

下面简单介绍一下在 SOUI 中实现的 LUA 脚本模块的实现。

在客户端程序中使用脚本语言一个基本的需求就是 C++代码和脚本代码的相互调用，即 C++代码可以调用脚本代码，脚本代码也要能够方便的调用 C++代码。

LUA 脚本原生提供了访问 C 函数的方法，只需要简单的调用几行代码就可以方便的把 C 函数注册到 LUA 函数空间中，但是并没有原生提供访问 C++对象的能力。

但是 LUA 中实现的 metatable 能够很好的模拟 C++的 OOP 能力，这也为导出 C++对象到 LUA 提供了可能。

目前已经有很多方法可以将 C++对象导出到 LUA，比如 luabind,tolua++, fflua 及本文中用到的 lua_tinker。

luabind 据说体积比较大，tolua++好像已经没人维护了，目前只支持 lua 5.1.4，fflua 是国内一个大神的作品，使用简单，只是我使用中碰到一点问题，最后还是选择了 lua_tinker。

lua_tinker 是一个韩国大神的作品，虽然作者本人没有维护了，但是代码相对比较简单易懂，国内有不少高手都对它进行了扩展。

在 SOUI 中使用的是官方的 0.5c 版本上结合网友修改的版本，实现对 lua 5.2.3 的支持。

言归正传，下面说说如何使用 lua_tinker 导出 SOUI 对外到 LUA。

要使用 LUA，首先当然要有一份 LUA 内核代码，这里用的 lua 5.2.3。

为了在 SOUI 中使用 LUA，我们还需要使用 LUA 内核实现一个 SOUI::IScriptModuler 接口：

```
namespace SOUI
{
    class SWindow;
/*!
\brief
    Abstract interface required for all scripting support modules to be used
with
    the SOUI system.
*/
struct IScriptModule : public IObjRef
{
    /**
     * GetScriptEngine
     * @brief    获得脚本引擎的指针
     * @return   void * -- 脚本引擎的指针
     * Describe
     */
    virtual void * GetScriptEngine () = 0;

/************************************************************************
       Abstract interface

************************************************************************/
    /*!
    \brief
       Execute a script file.
    \param pszScriptFile
       String object holding the filename of the script file that is to be
executed
    */
    virtual void    executeScriptFile(LPCSTR pszScriptFile)  = 0;
    /*!
    \brief
       Execute a script buffer.
    \param buff
       buffer of the script that is to be executed
```

```
    \param sz
       size of buffer
    */
    virtual    void    executeScriptBuffer(const char* buff, size_t sz)  = 0;
    /*!
    \brief
       Execute script code contained in the given String object.

    \param str
       String object holding the valid script code that should be executed.
    \return
       Nothing.
    */
    virtual void executeString(LPCSTR str) = 0;
    /*!
    \brief
       Execute a scripted global 'event handler' function.  The function
should take some kind of EventArgs like parameter
       that the concrete implementation of this function can create from the
passed EventArgs based object.
    \param handler_name
       String object holding the name of the scripted handler function.
    \param EventArgs *pEvt
       SWindow based object that should be passed, by any appropriate means,
to the scripted function.
    \return
       - true if the event was handled.
       - false if the event was not handled.
    */
    virtual    bool    executeScriptedEventHandler(LPCSTR handler_name,
EventArgs *pEvt)=0;
    /*!
    \brief
       Return identification string for the ScriptModule.  If the internal id
string has not been
       set by the ScriptModule creator, a generic string of "Unknown
scripting module" will be returned.
    \return
       String object holding a string that identifies the ScriptModule in
use.
    */
    virtual LPCSTR getIdentifierString() const = 0;
    /*!
```

```
    \brief
          Subscribes or unsubscribe the named Event to a scripted function
    \param target
          The target EventSet for the subscription.
    \param uEvent
          Event ID to subscribe to.
    \param subscriber_name
          String object containing the name of the script function that is
to be subscribed to the Event.
    \return
    */
    virtual bool subscribeEvent(SWindow* target, UINT uEvent, LPCSTR
subscriber_name) = 0;
    /**
     * unsubscribeEvent
     * @brief    取消事件订阅
     * @param    SWindow * target --  目标窗口
     * @param    UINT uEvent --  目标事件
     * @param    LPCSTR subscriber_name --  脚本函数名
     * @return   bool -- true 操作成功
     * Describe
     */
    virtual bool unsubscribeEvent(SWindow* target, UINT uEvent, LPCSTR
subscriber_name ) = 0;
};
struct IScriptFactory : public IObjRef
{
    virtual HRESULT CreateScriptModule(IScriptModule ** ppScriptModule) = 0;
};
}
```

实现上述接口后，SOUI 就可以用这个接口和脚本交互。

导出 SOUI 对象通常应该在 IScriptModule 的实现类的构造中执行。

使用 lua_tinker 导出 C++对象非常简单，下面看一下 scriptmodule-lua 是如何导出 SOUI 中使用的几个 C++对象的：

```
//导出基本结构体类型
UINT rgb(int r,int g,int b)
{
    return RGBA(r,g,b,255);
}

UINT rgba(int r,int g, int b, int a)
{
```

```
    return RGBA(r,g,b,a);
}

BOOL ExpLua_Basic(lua_State *L)
{
    try{
        lua_tinker::def(L,"RGB",rgb);
        lua_tinker::def(L,"RGBA",rgba);

        //POINT
        lua_tinker::class_add<POINT>(L,"POINT");
        lua_tinker::class_mem<POINT>(L, "x", &POINT::x);
        lua_tinker::class_mem<POINT>(L, "y", &POINT::y);
        //RECT
        lua_tinker::class_add<RECT>(L,"RECT");
        lua_tinker::class_mem<RECT>(L, "left", &RECT::left);
        lua_tinker::class_mem<RECT>(L, "top", &RECT::top);
        lua_tinker::class_mem<RECT>(L, "right", &RECT::right);
        lua_tinker::class_mem<RECT>(L, "bottom", &RECT::bottom);
        //SIZE
        lua_tinker::class_add<SIZE>(L,"SIZE");
        lua_tinker::class_mem<SIZE>(L, "cx", &SIZE::cx);
        lua_tinker::class_mem<SIZE>(L, "cy", &SIZE::cy);

        //CPoint
        lua_tinker::class_add<CPoint>(L,"CPoint");
        lua_tinker::class_inh<CPoint,POINT>(L);

lua_tinker::class_con<CPoint>(L,lua_tinker::constructor<CPoint,LONG,LONG>);
        //CRect
        lua_tinker::class_add<CRect>(L,"CRect");
        lua_tinker::class_inh<CRect,RECT>(L);

lua_tinker::class_con<CRect>(L,lua_tinker::constructor<CRect,LONG,LONG,LONG,
LONG>);
        lua_tinker::class_def<CRect>(L,"Width",&CRect::Width);
        lua_tinker::class_def<CRect>(L,"Height",&CRect::Height);
        lua_tinker::class_def<CRect>(L,"Size",&CRect::Size);
        lua_tinker::class_def<CRect>(L,"IsRectEmpty",&CRect::IsRectEmpty);
        lua_tinker::class_def<CRect>(L,"IsRectNull",&CRect::IsRectNull);
        lua_tinker::class_def<CRect>(L,"PtInRect",&CRect::PtInRect);
        lua_tinker::class_def<CRect>(L,"SetRectEmpty",&CRect::SetRectEmpty);
```

```
        lua_tinker::class_def<CRect>(L,"OffsetRect",(void
(CRect::*)(int,int))&CRect::OffsetRect);
        //CSize
        lua_tinker::class_add<CSize>(L,"CSize");
        lua_tinker::class_inh<CSize,SIZE>(L);

lua_tinker::class_con<CSize>(L,lua_tinker::constructor<CSize,LONG,LONG>);

        return TRUE;
    }catch(...)
    {
        return FALSE;
    }
}
```

```
#include <core/swnd.h>

//定义一个从 SObject 转换成 SWindow 的方法
SWindow * toSWindow(SObject * pObj)
{
    return sobj_cast<SWindow>(pObj);
}

BOOL ExpLua_Window(lua_State *L)
{
    try{
        lua_tinker::def(L,"toSWindow",toSWindow);
        lua_tinker::class_add<SWindow>(L,"SWindow");
        lua_tinker::class_inh<SWindow,SObject>(L);
        lua_tinker::class_con<SWindow>(L,lua_tinker::constructor<SWindow>);

lua_tinker::class_def<SWindow>(L,"GetContainer",&SWindow::GetContainer);
        lua_tinker::class_def<SWindow>(L,"GetRoot",&SWindow::GetRoot);

lua_tinker::class_def<SWindow>(L,"GetTopLevelParent",&SWindow::GetTopLevelPa
rent);
        lua_tinker::class_def<SWindow>(L,"GetParent",&SWindow::GetParent);

lua_tinker::class_def<SWindow>(L,"DestroyChild",&SWindow::DestroyChild);

lua_tinker::class_def<SWindow>(L,"GetChildrenCount",&SWindow::GetChildrenCou
nt);
```

```
lua_tinker::class_def<SWindow>(L,"FindChildByID",&SWindow::FindChildByID);
	lua_tinker::class_def<SWindow>(L,"FindChildByNameA",(SWindow*
(SWindow::*)(LPCSTR,int))&SWindow::FindChildByName);
	lua_tinker::class_def<SWindow>(L,"FindChildByNameW",(SWindow*
(SWindow::*)(LPCWSTR,int ))&SWindow::FindChildByName);
	 lua_tinker::class_def<SWindow>(L,"CreateChildrenFromString",(SWindow*
(SWindow::*)(LPCWSTR))&SWindow::CreateChildren);

lua_tinker::class_def<SWindow>(L,"GetTextAlign",&SWindow::GetTextAlign);
	lua_tinker::class_def<SWindow>(L,"GetWindowRect",(void
(SWindow::*)(LPRECT))&SWindow::GetWindowRect);
	lua_tinker::class_def<SWindow>(L,"GetWindowRect2",(CRect
(SWindow::*)())&SWindow::GetWindowRect);
	lua_tinker::class_def<SWindow>(L,"GetClientRect",(void
(SWindow::*)(LPRECT))&SWindow::GetClientRect);
	lua_tinker::class_def<SWindow>(L,"GetClientRect2",(CRect
(SWindow::*)())&SWindow::GetClientRect);

lua_tinker::class_def<SWindow>(L,"GetWindowText",&SWindow::GetWindowText);

lua_tinker::class_def<SWindow>(L,"SetWindowText",&SWindow::SetWindowText);

lua_tinker::class_def<SWindow>(L,"SendSwndMessage",&SWindow::SSendMessage);
	lua_tinker::class_def<SWindow>(L,"GetID",&SWindow::GetID);
	lua_tinker::class_def<SWindow>(L,"SetID",&SWindow::SetID);
	lua_tinker::class_def<SWindow>(L,"GetUserData",&SWindow::GetUserData);
	lua_tinker::class_def<SWindow>(L,"SetUserData",&SWindow::SetUserData);
	lua_tinker::class_def<SWindow>(L,"GetName",&SWindow::GetName);
	lua_tinker::class_def<SWindow>(L,"GetSwnd",&SWindow::GetSwnd);
	lua_tinker::class_def<SWindow>(L,"InsertChild",&SWindow::InsertChild);
	lua_tinker::class_def<SWindow>(L,"RemoveChild",&SWindow::RemoveChild);
	lua_tinker::class_def<SWindow>(L,"IsChecked",&SWindow::IsChecked);
	lua_tinker::class_def<SWindow>(L,"IsDisabled",&SWindow::IsDisabled);
	lua_tinker::class_def<SWindow>(L,"IsVisible",&SWindow::IsVisible);
	lua_tinker::class_def<SWindow>(L,"SetVisible",&SWindow::SetVisible);

lua_tinker::class_def<SWindow>(L,"EnableWindow",&SWindow::EnableWindow);
	lua_tinker::class_def<SWindow>(L,"SetCheck",&SWindow::SetCheck);
	lua_tinker::class_def<SWindow>(L,"SetOwner",&SWindow::SetOwner);
	lua_tinker::class_def<SWindow>(L,"GetOwner",&SWindow::GetOwner);
	lua_tinker::class_def<SWindow>(L,"Invalidate",&SWindow::Invalidate);
```

```
        lua_tinker::class_def<SWindow>(L,"InvalidateRect",(void
(SWindow::*)(LPCRECT))&SWindow::InvalidateRect);

lua_tinker::class_def<SWindow>(L,"AnimateWindow",&SWindow::AnimateWindow);

lua_tinker::class_def<SWindow>(L,"GetScriptModule",&SWindow::GetScriptModule
);
        lua_tinker::class_def<SWindow>(L,"Move2",(void
(SWindow::*)(int,int,int,int))&SWindow::Move);
        lua_tinker::class_def<SWindow>(L,"Move",(void
(SWindow::*)(LPCRECT))&SWindow::Move);
        lua_tinker::class_def<SWindow>(L,"FireCommand",&SWindow::FireCommand);

lua_tinker::class_def<SWindow>(L,"GetDesiredSize",&SWindow::GetDesiredSize);
        lua_tinker::class_def<SWindow>(L,"GetWindow",&SWindow::GetWindow);

        return TRUE;
    }catch(...)
    {
        return FALSE;
    }
}
```

还是很简单吧？！

这里有两点需要注意：

前面的代码里一般是导出全局函数，成员函数及成员变量，但是类的静态成员函数是不能用上面的方法导出的，下面看一下静态函数如何处理：

```
BOOL ExpLua_App(lua_State *L)
{
    try{
        lua_tinker::class_add<SApplication>(L,"SApplication");

lua_tinker::class_def<SApplication>(L,"AddResProvider",&SApplication::AddRes
Provider);

lua_tinker::class_def<SApplication>(L,"RemoveResProvider",&SApplication::Rem
oveResProvider);
        lua_tinker::class_def<SApplication>(L,"Init",&SApplication::Init);

lua_tinker::class_def<SApplication>(L,"GetInstance",&SApplication::GetInstan
ce);
```

```
lua_tinker::class_def<SApplication>(L,"CreateScriptModule",&SApplication::CreateScriptModule);

lua_tinker::class_def<SApplication>(L,"SetScriptModule",&SApplication::SetScriptFactory);

lua_tinker::class_def<SApplication>(L,"GetTranslator",&SApplication::GetTranslator);

lua_tinker::class_def<SApplication>(L,"SetTranslator",&SApplication::SetTranslator);
        lua_tinker::def(L,"theApp",&SApplication::getSingletonPtr);
        return TRUE;
    }catch(...)
    {
        return FALSE;
    }
}
```

注意上面导出 SApplication::getSingletonPtr 使用的方法，实际使用的是和导出全局函数一样的方法，因此在脚本中调用的时候也只能和全局函数一样调用，这一点和 C++调用静态函数是不同的。

第二个需要注意的地方就是，使用 lua_tinker 导出的 C++类如果是多继承的，那么只能导出一个基类，而且这个基类必须是第一个基类。

例如 SWindow 类，它从多个基类继承而来，但只能使用 lua_tinker::class_inh 来声明第一个基类 SObject，如果把 SWindow 的继承顺序调整一下，在 LUA 脚本里获得 SWindow 对象后也访问不了 SObject 的方法，这一点需要特别注意。

注：上面这个问题是我搜索好长时间才发现的，但也没有完全解决问题，本来想在导出 SHostWnd 时声明继承自 SWindow，尽管把 SWindow 放到了继承的第一位，但是在 LUA 脚本中用 SHostWnd 对象访问 SWindow 方法仍然失败，不知道什么原因，有兴趣的朋友可以研究一下。

上面介绍了如何导出 C++对象到 LUA 空间，下面介绍一下在 LUA 脚本中如何使用这些 C++对象：

所有的 C++对象导出到 LUA 后都将对应一个 metatable，可以使用"."来访问 table 中的成员变量（映射了 C++对象的成员变量），也可以使用 “：” 来访问 table 中的成员函数（映射了 C++对象函数），全局函数则直接使用函数名调用。

例如上面导出的 CRect 对象，在 LUA 脚本中使用如下：

```
function test(arg)
```

```
    local rc = CRect(0,0,100,100);
    local wid = rc:Width(); --访问成员函数 Width()
    local x1 = rc.left;--访问基类对象 RECT 的成员变量 left
end
```

更多操作请参考 SOUI 的 demo

4.21 在 SOUI 中做事件分发处理

不同的 SOUI 控件可以产生不同的事件。SOUI 系统中提供了两种事件处理方式：事件订阅 + 事件处理映射表(参见第八篇：SOUI 中控件事件的响应)

事件订阅由于直接将事件及事件处理函数连接，不存在事件分发的问题，这里主要介绍使用事件映射表时的事件分发。

在回答这个问题前，首先了解一下什么是事件分发。

在大型项目中，程序逻辑可能非常复杂，如果将所有 UI 中控件的事件处理集中在一个消息/事件映射表里，代码的可维护性会变得非常差。解决这个问题常见的方法就是将事件进行分类（如根据来源分类），不同类别的事件采用一个独立的事件处理对象来处理，这就是事件分发的核心。

目前流行的 UI 通常采用 Tab 控件来组织 UI，不同的功能放到不同的 Tab 页中，不同的 Tab 页可能互不相干的功能模块，对于类似这样的情形很自然的会想到采用事件分发机制来实现模块之间逻辑的解耦（如下图中 SoTool 采用的 UI）。

在上面的 UI 中，虽然整个 UI 被 TAB 分成了 6 个页面，但是 6 个页面都存在于同一个宿主窗口中。

一般情况下，如果 UI 相对比较简单，我们推荐直接在宿主窗口的事件处理映射表中统一处理控件事件。

但是当出现如上图这样复杂的界面时，最好是将不同功能页的事件处理在不同的对象中分别处理。

在 MFC 中，一个类要处理消息，这个类通常派生自 CCmdTarget（可能记错了，太久不用 MFC 了），主窗口收到的消息会自动路由到这个消息处理对象中。

在 WTL 中，WTL 提供了一组消息映射宏：CHAIN_MSG_MAP，CHAIN_MSG_MAP_MEMBER 等以便将消息分发到同样实现了消息映射表的任意 C++对象。

SOUI 的事件分发采用了 WTL 消息分发类似的机制，同样采用事件映射宏的方式来构造事件映射表，下面是 SOUI 中几个主要的和事件分发相关的宏：

```
#define EVENT_MAP_BEGIN()                           \
protected:                                         \
    virtual BOOL _HandleEvent(SOUI::EventArgs *pEvt)\
    {                                              \
        UINT     uCode = pEvt->GetID();            \

#define EVENT_MAP_DECLEAR()                         \
protected:                                         \
    virtual BOOL _HandleEvent(SOUI::EventArgs *pEvt);\

#define EVENT_MAP_BEGIN2(classname)                 \
    BOOL classname::_HandleEvent(SOUI::EventArgs *pEvt)\
    {                                              \
        UINT     uCode = pEvt->GetID();            \

#define EVENT_MAP_END()                             \
        return __super::_HandleEvent(pEvt);         \
    }                                              \

#define EVENT_MAP_BREAK()                           \
    return FALSE;                                  \
    }                                              \

#define CHAIN_EVENT_MAP(ChainClass)                 \
        if(ChainClass::_HandleEvent(pEvt))          \
            return TRUE;                           \
```

```
#define CHAIN_EVENT_MAP_MEMBER(theChainMember)      \
    {                                               \
    if(theChainMember._HandleEvent(pEvt))           \
        return TRUE;                                \
    }

#define EVENT_CHECK_SENDER_ROOT(pRoot)              \
    {                                               \
    SWindow *pWnd = sobj_cast<SWindow>(pEvt->sender);\
    if(!pWnd->IsDescendant(pRoot))                  \
        return FALSE;                               \
    }

// void OnEvent(EventArgs *pEvt)
#define EVENT_HANDLER(cd, func)                     \
    if(cd == uCode)                                 \
    {                                               \
        func(pEvt); return TRUE;                    \
    }
```

下面是 SoTool 中的 MainDlg 中的事件处理：

```
    //soui 消息
    EVENT_MAP_BEGIN()
        EVENT_NAME_COMMAND(L"btn_close", OnClose)
        EVENT_NAME_COMMAND(L"btn_min", OnMinimize)
        EVENT_NAME_COMMAND(L"btn_max", OnMaximize)
        EVENT_NAME_COMMAND(L"btn_restore", OnRestore)
        CHAIN_EVENT_MAP_MEMBER(m_imgMergerHandler)
        CHAIN_EVENT_MAP_MEMBER(m_codeLineCounter)
        CHAIN_EVENT_MAP_MEMBER(m_2UnicodeHandler)
        CHAIN_EVENT_MAP_MEMBER(m_folderScanHandler)
        CHAIN_EVENT_MAP_MEMBER(m_calcMd5Handler)
    EVENT_MAP_END()
```

上面代码中，EVENT_MAP_BEGIN()和 EVENT_MAP_END()这两个宏构造出一个空的事件处理函数，该函数自动将未处理的事件交给基类的事件处理函数处理。

如果基类中没有事件处理函数，显然这个事件映射表编译不能通过，此时 SOUI 提供了另一个 EVENT_MAP_BREAK()来代替。

上面的事件分发表中，我使用 CHAIN_EVENT_MAP_MEMBER 宏将来自不同页面的控件事件传递到不同的事件处理对象中。

下面代码是 m_imgMergerHandler 对象头文件。

```
class CImageMergerHandler : public IFileDropHandler
{
```

```
friend class CMainDlg;
public:
    CImageMergerHandler(void);
    ~CImageMergerHandler(void);

    void OnInit(SWindow *pRoot);
    void AddFile(LPCWSTR pszFileName);
protected:
    virtual void OnFileDropdown(HDROP hDrop);
    void OnSave();
    void OnClear();
    void OnModeHorz();
    void OnModeVert();

    EVENT_MAP_BEGIN()
        EVENT_CHECK_SENDER_ROOT(m_pPageRoot)
        EVENT_NAME_COMMAND(L"btn_save", OnSave)
        EVENT_NAME_COMMAND(L"btn_clear", OnClear)
        EVENT_NAME_COMMAND(L"radio_horz", OnModeHorz)
        EVENT_NAME_COMMAND(L"radio_vert", OnModeVert)
    EVENT_MAP_BREAK()

    SWindow *m_pPageRoot;
    SImgCanvas *m_pImgCanvas;
};
```

可以看到这里的事件映射表使用了 EVENT_MAP_BREAK 来结束。

在 SOUI 中推荐使用控件的 name 属性来标识一个控件（name 属性是一个 wchar*的字符串，使用 name 虽然在事件分发时采用字符串比较，较基于整数 id 属性的比较效率低一点，好处在于代码的可读性好），不同的页面中的控件如果出现相同的 name 该如何识别呢？

在 SOUI 中使用了一点小技巧：在事件处理对象中实现一个 oninit 函数，该函数在 maindlg 中处理 WM_INITDIALOG 时被调用，在 oninit 中保存了一个页面根节点的指针：SWindow *m_pPageRoot;

在事件映射表的开始，我们采用 EVENT_CHECK_SENDER_ROOT(m_pPageRoot)这个宏来识别那些来自本页面的事件。如果事件是来自其它页面则不处理。

4.22 在 SOUI 中使用异步通知

异步通知是客户端开发中常见的需求，比如在一个网络处理线程中要通知 UI 线程更新等等。

通常在 Windows 编程中，为了方便，我们一般会向 UI 线程的窗口句柄 Post/Send 一个窗口消息从而达到将非 UI 线程的事件切换到 UI 线程处理的目的。

在 SOUI 引入通知中心以前要在 SOUI 中处理非 UI 线程事件我也推荐用上面的方法。

使用窗口消息至少有以下两个不足：

1、需要在线程中持有一个窗口句柄。

2、发出的消息只能在该窗口句柄的消息处理函数里处理。

SNotifyCenter

最新的 SOUI 引入了一个新的单例对象：SNotifyCenter，专门用来处理这类问题。

新的 SNotifyCenter 解决了窗口消息存在的上面的两个问题：

1、通过使用全局单例，SNotifyCenter 可以在代码任意位置获取它的指针（前提当然是要在它的生命周期内）；

2、使用 SNotifyCenter 产生的通知采用 SOUI 的事件系统来派发，通过结合 SOUI 的事件订阅系统，用户可以在任意位置处理发出的事件。

在介绍如何使用 SNotifyCenter 前，先看一下 NotifyCenter.h 的代码：

```
#pragma once

#include <core/SSingleton.h>

namespace SOUI
{
    template<class T>
    class TAutoEventMapReg
    {
        typedef TAutoEventMapReg<T> _thisClass;
    public:
        TAutoEventMapReg()
        {

SNotifyCenter::getSingleton().RegisterEventMap(Subscriber(&_thisClass::OnEve
nt,this));
        }
        ~TAutoEventMapReg()
        {

SNotifyCenter::getSingleton().UnregisterEventMap(Subscriber(&_thisClass::OnE
vent,this));
        }
    protected:
        bool OnEvent(EventArgs *e){
```

```
            T * pThis = static_cast<T*>(this);
            return !!pThis->_HandleEvent(e);
        }
    };
    class SOUI_EXP SNotifyCenter : public SSingleton<SNotifyCenter>
                    , public SEventSet
    {
    public:
        SNotifyCenter(void);
        ~SNotifyCenter(void);
        /**
        * FireEventSync
        * @brief    触发一个同步通知事件
        * @param    EventArgs *e -- 事件对象
        * @return
        *
        * Describe  只能在UI线程中调用
        */
        void FireEventSync(EventArgs *e);
        /**
        * FireEventAsync
        * @brief    触发一个异步通知事件
        * @param    EventArgs *e -- 事件对象
        * @return
        *
        * Describe  可以在非UI线程中调用，EventArgs *e必须是从堆上分配的内存，调用后使用Release释放引用计数
        */
        void FireEventAsync(EventArgs *e);
        /**
        * RegisterEventMap
        * @brief    注册一个处理通知的对象
        * @param    const ISlotFunctor &slot -- 事件处理对象
        * @return
        *
        * Describe
        */
        bool RegisterEventMap(const ISlotFunctor &slot);
        /**
        * RegisterEventMap
        * @brief    注销一个处理通知的对象
        * @param    const ISlotFunctor &slot -- 事件处理对象
        * @return
```

```
        *
        * Describe
        */
        bool UnregisterEventMap(const ISlotFunctor & slot);
    protected:
        void OnFireEvent(EventArgs *e);
        void ExecutePendingEvents();
        static VOID CALLBACK OnTimer( HWND hwnd,
            UINT uMsg,
            UINT_PTR idEvent,
            DWORD dwTime
            );
        SCriticalSection    m_cs;           //线程同步对象
        SList<EventArgs*>    *m_evtPending;//挂起的等待执行的事件
        DWORD               m_dwMainTrdID;//主线程 ID
        UINT_PTR           m_timerID;    //定时器 ID，用来执行异步事件
        SList<ISlotFunctor*>    m_evtHandlerMap;
    };
}
```

在这个文件中提供了两个类，一个就是 SNotifyCenter，另一个是 TAutoEventMapReg<T>。

可以看到 SNotifyCenter 中除构造外只有 4 个 public 方法:

FireEventSync, FireEventAsync 用来触发事件。

RegisterEventMap，UnregisterEventMap 则用来提供事件处理订阅。

如何使用 SNotifyCenter?

1、在 main 中实例化 SNotifyCenter。（不明白可以参考 demo)

2、定义需要通过通知中心分发的事件类型，首先定义事件，然后向通知中心注册，参见下面代码：

```
void CMainDlg::OnBtnStartNotifyThread()
 {
    if(IsRunning()) return;
    SNotifyCenter::getSingleton().addEvent(EVENTID(EventThreadStart));
    SNotifyCenter::getSingleton().addEvent(EVENTID(EventThreadStop));
    SNotifyCenter::getSingleton().addEvent(EVENTID(EventThread));

    EventThreadStart evt(this);
    SNotifyCenter::getSingleton().FireEventSync(&evt);
    BeginThread();
 }
```

```
void CMainDlg::OnBtnStopNotifyThread()
{
    if(!IsRunning()) return;

    EndThread();
    EventThreadStop evt(this);
    SNotifyCenter::getSingleton().FireEventSync(&evt);

    SNotifyCenter::getSingleton().removeEvent(EventThreadStart::EventID);
    SNotifyCenter::getSingleton().removeEvent(EventThreadStop::EventID);
    SNotifyCenter::getSingleton().removeEvent(EventThread::EventID);
}
```

3、使需要处理通知中心分发的事件的对象从 TAutoEventMapReg 继承，实现事件的自动订阅（方便在事件映射表中统一处理事件），这一步是可选的，你也可以直接使用 SOUI 提供的事件订阅机制向通知中心订阅特定事件。

4、在事件映射表里处理事件（没有第 3 步时，则同样没有这一步）。

具体用法参见 SOUI 的 demo。

4.23 在 SOUI2.0 中像 android 一样使用资源

SOUI2.0 之前，在 SOUI 中使用资源通常是直接使用这个资源的 name（一个字符串）来引用。使用字符串的好处在于字符串能够表达这个资源的意义，因此使用字符串也是现代 UI 引擎常用的方式。

尽管直接使用字符串有意义明确的优点，它同样也有缺点：

1、字符串写错了，编译器不知道。这可能导致一些很难发现的 BUG。

2、控件查询，比较时基于字符串，相对来说性能会差一点（好在现在 CPU 够强，这点性能损失通常可以忽略）。

做过 Android 开发的朋友可能知道，在 Android 中要引用一个资源如图片、字符串、颜色等可以使用 R.id.xxx, R.string.xxx,R.color.xxx 这样的形式来引用。

Android 内部全部自动转换成 ID，整数比较显然比字符串比较快，这里不作讨论。

这种方式一个好处在于 Android 的自动补全功能能够帮助你快速的输入你需要的资源，除了加快了编码速度，还大大减少了输入错误。

SOUI2.0 把 Android 的这种资源引用方式引入了进来。

关键在于 uiresbuilder。原来 SOUI 中的 uiresbuilder 只提供将资源转换成.rc2 功能，方便将资源编译到 EXE/DLL 中。

2.0 版本新增加 name 提取，id 生成，字符串表 ID 生成，颜色表 ID 生成功能。它们会输出到一个 C++头文件（由命令行参数指定）。

要使用该功能首先要保证所有的布局 XML 所在的资源类型为"Layout"，然后在 uiresbuilder 的命名行中加入： -h “输出文件名” idtable。-h 后面紧跟输出文件名，idtable 指示需要给没有指定 ID 的控件自动生成 ID，该功能默认关闭。

生成成功后，你的“输出文件”的内容可能是下面的样子：

```
//stamp:0ae7b68801b8deb8
/*<-----------------------------------------------------------------------------
------------------------>*/
/*该文件由 uiresbuilder 生成，请不要手动修改*/
/*<-----------------------------------------------------------------------------
------------------------>*/
#pragma once
#include <res.mgr/snamedvalue.h>
namespace SOUI
{
    const SNamedID::NAMEDVALUE namedXmlID[]={
        {L"btnSelectGif",65540},
        {L"btn_display",65541},
        {L"btn_hidetst",65542},
        {L"btn_lrc",65543},
        {L"btn_menu",65536},
        {L"ctrl_flash",65538},
        {L"gif_test",1000},
        {L"giftest",65539},
        {L"tab_main",65537}       };
    class _R{
    public:
        class _name{
        public:
        _name(){
            btnSelectGif = namedXmlID[0].strName;
            btn_display = namedXmlID[1].strName;
            btn_hidetst = namedXmlID[2].strName;
            btn_lrc = namedXmlID[3].strName;
            btn_menu = namedXmlID[4].strName;
            ctrl_flash = namedXmlID[5].strName;
            gif_test = namedXmlID[6].strName;
            giftest = namedXmlID[7].strName;
            tab_main = namedXmlID[8].strName;
        }
         const wchar_t * btnSelectGif;
         const wchar_t * btn_display;
         const wchar_t * btn_hidetst;
```

```
        const wchar_t * btn_lrc;
        const wchar_t * btn_menu;
        const wchar_t * ctrl_flash;
        const wchar_t * gif_test;
        const wchar_t * giftest;
        const wchar_t * tab_main;
    }name;

    class _id{
    public:
    const static int btnSelectGif    =    65540;
    const static int btn_display    =    65541;
    const static int btn_hidetst    =    65542;
    const static int btn_lrc    =    65543;
    const static int btn_menu    =    65536;
    const static int ctrl_flash    =    65538;
    const static int gif_test    =    1000;
    const static int giftest    =    65539;
    const static int tab_main    =    65537;
    }id;

    class _string{
    public:
    const static int mccol_1    =    0;
    const static int mccol_2    =    1;
    const static int mccol_3    =    2;
    const static int mccol_4    =    3;
    const static int mccol_5    =    4;
    const static int mccol_6    =    5;
    const static int title    =    6;
    const static int ver    =    7;
    }string;

    class _color{
    public:
    const static int blue    =    0;
    const static int gray    =    1;
    const static int green    =    2;
    const static int red    =    3;
    const static int white    =    4;
    }color;

};
```

```
    const _R R;
}
```

第一行保留的是一个时间戳，如果资源中布局相关的资源没有变化，则不再生成。

首先会自动生成一个 name, id 映射表：SNamedID::NAMEDVALUE，这是一个结构体数组，保留每一个控件的名字及 ID（自动生成的及 XML 中定义的，自动生成的 ID 自动从 65536 开始，因此自己定义时应该小于这个值）。

接下来定义了一个类 class _R。_R 中有 4 个子类：_name, _id, _string, _color，每个类有一个实例，对应的名字为:name, id, string, color。

最后定义一个_R 的实例 R。

到这里你应该已经知道在 SOUI 中 R 这个对象有哪几个成员了。

那么在代码中如何使用 R 这个对象呢？

如何使用 name 对象：

观察 R 这个对象，你可能已经发现，在代码直接使用 R.name.btnSelectGif 就等价于在代码中输入 L“btnSelectGif”，这样的好处在于你在输入 R.name.btn 后 VS 或者 VA 可能就给你补全后面的 SelectGif，既提高了编码效率，又保证了不会出错。（对象 name 修改以后也可以使用 VA 的变量重命名功能自动批量修改）。

如何使用 ID 对象：

前面提到使用字符串来查找窗口对象相对来说较 ID 比较会慢一点，那么如何使用 ID 对象呢？要使用 ID 对象，有一个要求：由于自动生成的 ID 并没有修改到原有的 XML 中，直接从 XML 中初始化布局时是没有 ID 属性的。为此 SOUI2.0 的 SApplication 对象增加了一个方法：InitXmlNamedID，参见 demo(注意调用位置）：

```
        //如果需要在代码中使用 R::id::namedid 这种方式来使用控件必须要这一行代码：2016 年 2 月 2 日，R::id::namedXmlID 是由 uiresbuilder 增加-h .\res\resource.h idtable 这 3 个参数后生成的。
        theApp->InitXmlNamedID(namedXmlID,ARRAYSIZE(namedXmlID),TRUE);
```

在布局创建前给 App 对象初始化一个自动生成的 Name 转 ID 表。

控件创建并初始化 name 属性时，自动从该表中查询 ID。

如此，在代码中可以直接使用 R.id.btnSelectGif 来查找对应的控件了。

如何使用 string, color 对象：

在布局 XML 中使用使用 string, color 对象和 android 一样：采用@string/str-name, @color/color-name 来分别引用在 string，color 中定义的对应的字符串或者颜色值。

这里重点讲一下在代码中使用这两个对象：

```
        //演示 R.color.xxx,R.string.xxx 在代码中的使用。
        COLORREF crRed = GETCOLOR(R.color.red);
        SStringW strVer = GETSTRING(R.string.ver);
```

上面是 demo：winmain 中一个使用示例。

R.color.red， R.string.ver 是自动生成的两个整数，GETCOLOR, GETSTRING 这两个宏会自动从资源中的字符串表及颜色表中获取对应的 ID 指定的值。

4.24 两个 SOUI 新控件 ---- SListView 和 SComboView（借用 Andorid 的设计）

SOUI 原来实现的 SListBoxEx 的效率一直是我对 SOUI 不太满意的地方，包括后来网友实现的 SListCtrlEx。

这类控件为每一个列表项创建一个 SWindow 来容纳数据，当数据量比较大（10000+）时，一方面内存消耗会很严重；另一方面列表数据初始化也需要大量的时间。

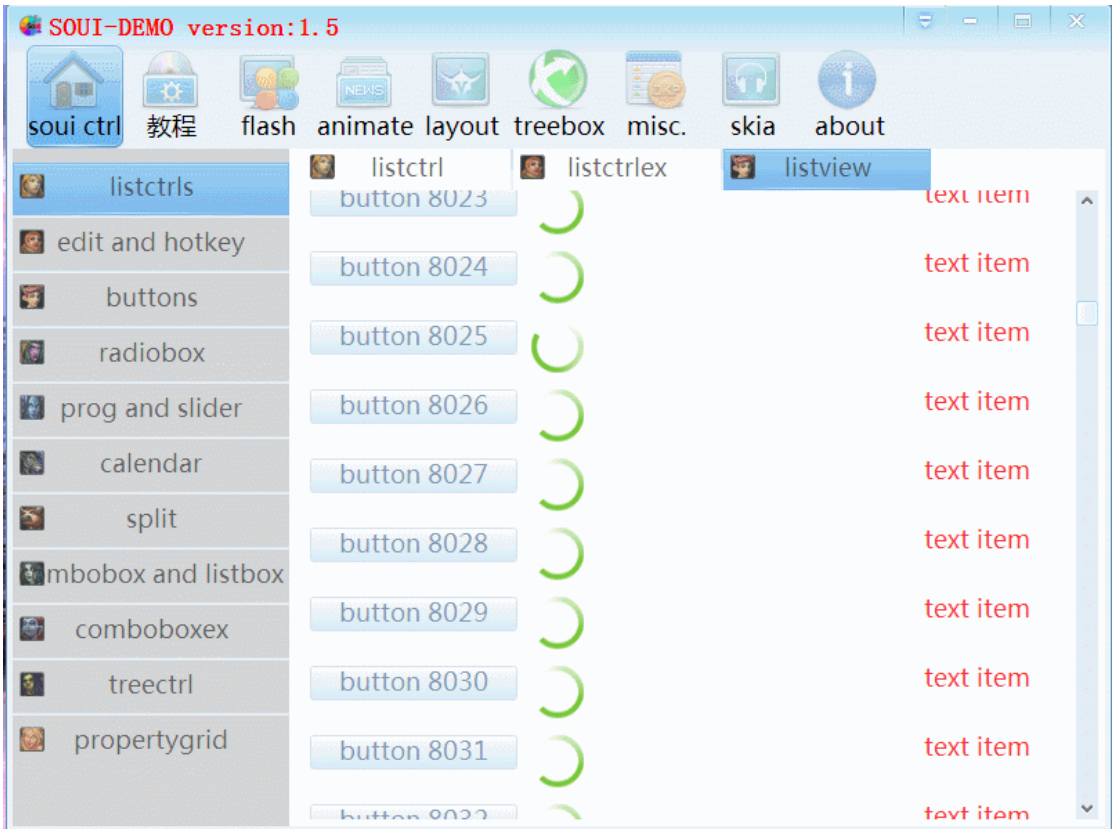
今年开始转型做 Android 开发。大家都知道 Android 开发 APP 和 PC 上开发 APP 相比要简单很多，其中我个人体会最深的就是 Android 的 ListView 控件。

在 Android 中，ListView 中列表项的显示采用控件+适配器(Adapter)的模式，也就是所谓的 MVC 模式。一个表项在需要显示的时候才会把数据加载到 View 里去，当这个表项被隐藏起来以后，容纳该表项的容器(View)则自动被加入到 ListView 中保存的一个容器回收列表中。需要显示一个新表项时首先去回收站里查找是否存在指定类型的容器，存在则自动复用。

基本思想如上，当然实际实现还使用了很多技巧。通过上述机制，可以有效解决 ListView 显示大量数据的问题。

本来也一直想重写 SOUI 的 ListBox, 这段时间正好项目需要，抽出时间模仿了一个，效果不错。

先看看效果：



)

第一张图是一个加载 50000 行的 SListView 控件，第二个图是演示使用 SComboView 来做用户登陆界面。

要在 SOUI 中使用 SListView，我们首先需要自己实现数据填充的 Adapter:

```
class CTestAdapter : public SAdapterBase
{
```

```
protected:
    SListView * m_pOwenr;
public:
    CTestAdapter(SListView *pOwner):m_pOwenr(pOwner)
    {
    }
    virtual int getCount()
    {
        return 50000;
    }
    virtual void getView(int position, SWindow * pItem)
    {
        if(pItem->GetChildrenCount()==0)
        {
            pItem->InitFromXml(m_pOwenr->GetTemplate());
        }
        SAnimateImgWnd *pAni =
pItem->FindChildByName2<SAnimateImgWnd>(L"ani_test");

        SButton *pBtn = pItem->FindChildByName2<SButton>(L"btn_test");
        pBtn->SetWindowText(SStringW().Format(L"button %d",position));
        pBtn->SetUserData(position);

pBtn->GetEventSet()->subscribeEvent(EVT_CMD,Subscriber(&CTestAdapter::OnButt
onClick,this));
    }
    bool OnButtonClick(EventArgs *pEvt)
    {
        SButton *pBtn = sobj_cast<SButton>(pEvt->sender);
        int iItem = pBtn->GetUserData();
        SMessageBox(NULL,SStringT().Format(_T("button of %d item was
clicked"),iItem),_T("haha"),MB_OK);
        return true;
    }
};
```

这里最关键的就是实现 IAdapter 中的 getView 方法。

和 Android 的 ListView 不同，SOUI 中使用条件 pItem->GetChildrenCount()==0 来判断一个容器是否是被复用。

当 pItem->GetChildrenCount()==0 时代表该容器还没有被初始化，需要自己从 XML 模板中初始化容器。XML 模板可以自己自己定义的任意符合 SOUI 布局语法的数据。

容器初始化完成后就可以向里面填充数据，也可以向控件连接响应函数了（subscribeEvent）。

在 UI 创建完成后需要在代码中把这个 Adapter 交给 SListView：

```
LRESULT CMainDlg::OnInitDialog( HWND hWnd, LPARAM lParam )
{
    //....

    SListView *pLstView = FindChildByName2<SListView>("lv_test");
    if(pLstView)
    {
        CTestAdapter *pAdapter = new CTestAdapter(pLstView);
        pLstView->SetAdapter(pAdapter);
        pAdapter->Release();
    }

    return 0;
}
```

第二个界面是演示 SComboView 的。SComboView 的用户和 SListView 基本一样，具体看代码。

4.25 在 SOUI 中控件属性查询方法

SOUI 项目的 SVN 根目录下有一个 doc 目录，下面有一份控件属性表。包含了大部分控件的大部分属性，不过也不一定完全准确。最保险的办法还是查源代码。

SOUI 对象包含控件及 ISkinObj 等从 SObject 派生的对象都可以使用 XML 配置属性。要知道如何查 SOUI 对象属性，首先要看一下 SOUI 解释属性的流程。

```
BOOL SObject::InitFromXml( pugi::xml_node xmlNode )
    {
        if(!xmlNode) return FALSE;
#ifdef _DEBUG
        {
            pugi::xml_writer_buff writer;

xmlNode.print(writer,L"\t",pugi::format_default,pugi::encoding_utf16);
            m_strXml=SStringW(writer.buffer(),writer.size());
        }
#endif
        //设置当前对象的属性
        //优先处理"class"属性
        pugi::xml_attribute attrClass=xmlNode.attribute(L"class");
        if(attrClass)
```

```
        {
            attrClass.set_userdata(1);      //预处理过的属性，给属性增加一个 userdata
            SetAttribute(attrClass.name(), attrClass.value(), TRUE);
        }
        for (pugi::xml_attribute attr = xmlNode.first_attribute(); attr; attr
= attr.next_attribute())
        {
            if(attr.get_userdata()) continue;   //忽略已经被预处理的属性
            SetAttribute(attr.name(), attr.value(), TRUE);
        }
        if(attrClass)
        {
            attrClass.set_userdata(0);
        }
        //调用初始化完成接口
        OnInitFinished(xmlNode);
        return TRUE;
    }
```

在 InitFromXml 方法中，首先获取 class 属性交给对象解释，然后再枚举其它属性。

获取到一个属性以后调用 SOjbect::SetAttribute 方法来处理属性。

```
class SObject{
public
        virtual HRESULT SetAttribute(const SStringW &  strAttribName, const
SStringW &  strValue, BOOL bLoading)
        {
            return DefAttributeProc(strAttribName,strValue,bLoading);
        }
};
```

SetAttribute 是一个虚函数，所有派生类都可以通过重载该方法来实现不同属性的处理。

在 SOUI 中提供了一组宏来处理 SObject 的属性，下面看一下 SWindow 类的实现：

```
        SOUI_ATTRS_BEGIN()
            ATTR_CUSTOM(L"id",OnAttrID)
            ATTR_STRINGW(L"name",m_strName,FALSE)
            ATTR_CUSTOM(L"skin", OnAttrSkin)        //直接获得皮肤对象
            ATTR_SKIN(L"ncskin", m_pNcSkin, TRUE)   //直接获得皮肤对象
            ATTR_CUSTOM(L"class", OnAttrClass)      //获得 style
            ATTR_INT(L"data", m_uData, 0 )
            ATTR_CUSTOM(L"enable", OnAttrEnable)
            ATTR_CUSTOM(L"visible", OnAttrVisible)
            ATTR_CUSTOM(L"show", OnAttrVisible)
            ATTR_CUSTOM(L"display", OnAttrDisplay)
            ATTR_CUSTOM(L"pos", OnAttrPos)
```

```
        ATTR_CUSTOM(L"offset", OnAttrOffset)
        ATTR_CUSTOM(L"pos2type", OnAttrPos2type)
        ATTR_CUSTOM(L"cache", OnAttrCache)
        ATTR_CUSTOM(L"alpha",OnAttrAlpha)
        ATTR_CUSTOM(L"layeredWindow",OnAttrLayeredWindow)
        ATTR_CUSTOM(L"trackMouseEvent",OnAttrTrackMouseEvent)
        ATTR_I18NSTRT(L"tip", m_strToolTipText, FALSE)  //使用语言包翻译
        ATTR_INT(L"msgTransparent", m_bMsgTransparent, FALSE)
        ATTR_INT(L"maxWidth",m_nMaxWidth,FALSE)
        ATTR_INT(L"clipClient",m_bClipClient,FALSE)
        ATTR_INT(L"focusable",m_bFocusable,FALSE)
        ATTR_INT(L"float",m_bFloat,FALSE)
        ATTR_CHAIN(m_style)                    //支持对 style 中的属性定制
    SOUI_ATTRS_END()
```

SOUI_ATTRS_BEGIN + SOUI_ATTRS_END 宏展开构成一个空的 SetAttribute 函数。

ATTR_XXX 则对不同类型的属性提供一种固定的处理方式，全部属性形成一个属性映射表。

因此要查对象属性，首先就是查对象代码中对应的属性映射表，同时也别忘记去查对象的基类的属性映射表。

有些对象（如 Richedit)的属性可能在这个属性映射表里找不到，那又是为什么呢？

注意 SetAttribute 最后调用了 DefAttributeProc，使用属性映射表构造的属性处理函数也同样会调用 DefAttributeProc。

在 DefAttributeProc 中可以批量的按照自己的方式处理属性。下面看一下 Richedit::DefAttributeProc 的实现：

```
HRESULT SRichEdit::DefAttributeProc(const SStringW & strAttribName,const
SStringW & strValue, BOOL bLoading)
{
    HRESULT hRet=S_FALSE;
    DWORD dwBit=0,dwMask=0;
    //hscrollbar
    if(strAttribName.CompareNoCase(L"hscrollBar")==0)
    {
        if(strValue==L"0")
            m_dwStyle&=~WS_HSCROLL;
        else
            m_dwStyle|=WS_HSCROLL;
        dwBit|=TXTBIT_SCROLLBARCHANGE;
        dwMask|=TXTBIT_SCROLLBARCHANGE;
    }
    //vscrollbar
```

```
else if(strAttribName.CompareNoCase(L"vscrollBar")==0)
{
    if(strValue==L"0")
        m_dwStyle&=~WS_VSCROLL;
    else
        m_dwStyle|=WS_VSCROLL;
    dwBit|=TXTBIT_SCROLLBARCHANGE;
    dwMask|=TXTBIT_SCROLLBARCHANGE;
}
//auto hscroll
else if(strAttribName.CompareNoCase(L"autoHscroll")==0)
{
    if(strValue==L"0")
        m_dwStyle&=~ES_AUTOHSCROLL;
    else
        m_dwStyle|=ES_AUTOHSCROLL;
    dwBit|=TXTBIT_SCROLLBARCHANGE;
    dwMask|=TXTBIT_SCROLLBARCHANGE;
}
//auto hscroll
else if(strAttribName.CompareNoCase(L"autoVscroll")==0)
{
    if(strValue==L"0")
        m_dwStyle&=~ES_AUTOVSCROLL;
    else
        m_dwStyle|=ES_AUTOVSCROLL;
    dwBit|=TXTBIT_SCROLLBARCHANGE;
    dwMask|=TXTBIT_SCROLLBARCHANGE;
}
//multilines
else if(strAttribName.CompareNoCase(L"multiLines")==0)
{
    if(strValue==L"0")
        m_dwStyle&=~ES_MULTILINE;
    else
        m_dwStyle|=ES_MULTILINE,dwBit|=TXTBIT_MULTILINE;
    dwMask|=TXTBIT_MULTILINE;
}
//readonly
else if(strAttribName.CompareNoCase(L"readOnly")==0)
{
    if(strValue==L"0")
        m_dwStyle&=~ES_READONLY;
```

```
        else
            m_dwStyle|=ES_READONLY,dwBit|=TXTBIT_READONLY;
        dwMask|=TXTBIT_READONLY;
        if(!bLoading)
        {//update dragdrop
            OnEnableDragDrop(!(m_dwStyle&ES_READONLY) && m_fEnableDragDrop);
        }
    }
    //want return
    else if(strAttribName.CompareNoCase(L"wantReturn")==0)
    {
        if(strValue==L"0")
            m_dwStyle&=~ES_WANTRETURN;
        else
            m_dwStyle|=ES_WANTRETURN;
    }
    //password
    else if(strAttribName.CompareNoCase(L"password")==0)
    {
        if(strValue==L"0")
            m_dwStyle&=~ES_PASSWORD;
        else
            m_dwStyle|=ES_PASSWORD,dwBit|=TXTBIT_USEPASSWORD;
        dwMask|=TXTBIT_USEPASSWORD;
    }
    //number
    else if(strAttribName.CompareNoCase(L"number")==0)
    {
        if(strValue==L"0")
            m_dwStyle&=~ES_NUMBER;
        else
            m_dwStyle|=ES_NUMBER;
    }
    //password char
    else if(strAttribName.CompareNoCase(L"passwordChar")==0)
    {
        SStringT strValueT=S_CW2T(strValue);
        m_chPasswordChar=strValueT[0];
    }
    //enabledragdrop
    else if(strAttribName.CompareNoCase(L"enableDragdrop")==0)
    {
        if(strValue==L"0")
```

```
        {
            m_fEnableDragDrop=FALSE;
        }else
        {
            m_fEnableDragDrop=TRUE;
        }
        if(!bLoading)
        {
            OnEnableDragDrop( !(m_dwStyle&ES_READONLY) & m_fEnableDragDrop);
        }
    }
    //auto Sel
    else if(strAttribName.CompareNoCase(L"autoSel")==0)
    {
        if(strValue==L"0")
        {
            m_fAutoSel=FALSE;
        }else
        {
            m_fAutoSel=TRUE;
        }
    }
    else
    {
        hRet=__super::DefAttributeProc(strAttribName,strValue,bLoading);
    }
    if(!bLoading)
    {
        m_pTxtHost->GetTextService()->OnTxPropertyBitsChange(dwMask,dwBit);
        hRet=TRUE;
    }
    return hRet;
}
```

可以看到在这个方法中对很多属性做了处理。

总结：

先查目标对象的属性映射表，再找基类的属性映射表，最后查对象的 DefAttributeProc。

4.26 √使用 SOUI 的 SMCListView 控件

列表控件是客户端应用最常用的控件之一。列表控件通常只负责显示数据，最多通知一下 APP 列表行的选中状态变化。

现在的 UI 经常要求程序猿在列表控件里不光显示内容，还要能和用户交互，显示动画等等，传统的列表控件对于这样的需求基本是无能为力了。

Android 开发中很多界面都直接采用 ListView 实现，ListView 中每一个 Item 中都可以容纳其它控件，这样的设计使得在表项中的交互和在主面板上交互一样简单。

虽然在列表项中容纳其它控件并不是什么新的思想，考虑到列表中的数据量是不确定的，如果给每一个表项的分配一个容器窗口，系统的内存占用及效率都成问题。

还好 Andriod 开源，简单看一下 Android 里 ListView 控件的源代码就可以发现，Android 实现的 ListView 关键在于容器窗口的重用。

借鉴 Andriod ListView 控件的思想，在 SOUI 中也实现了对应的 ListView 控件。但是 ListView 只有一列，显示显示复杂内容问题不大，但是不能调整列宽等，和一个 ListCtrl 的功能还是有些差距。

多列列表和单列表最大的区别在于多了一个表头，核心的东西并没有区别，经过近 3 天的编码调试，终于完成了这个革命性的控件（至少我认为是 Windows UI 上革命性的）。

先看下效果：

SMCListView 控件的使用：

XML 配置：

要使用这个列表控件，首先应该在 XML 中定义该控件的位置及属性，参考下面摘自 demo 的代码：

```
<mclistview name="mclv_test" colorBkgnd="#ffffff" pos="10,10,-10,-10"
headerHeight="30">
          <header align="center"  sortSkin="skin_lcex_header_arrow"
itemSkin="skin_lcex_header" itemSwapEnable="1" fixWidth="0"
font="underline:0,adding:-3" sortHeader="1" colorBkgnd="#ffffff" >
            <items>
              <item width="480">软件名称</item>
              <item width="95">软件评分</item>
              <item width="100">大小</item>
              <item width="100">安装时间</item>
              <item width="100">使用频率</item>
              <item width="100">操作</item>
            </items>
          </header>
          <template itemHeight="80" colorHover="#cccccc"
colorSelected="#0000ff" id="30000">
            <window name="col1">
              <img name="img_icon" skin="skin_icon6" pos="10,8,@64,@64"/>
              <text name="txt_name" pos="[5,16" font="bold:1,adding:-1">火狐浏
览器</text>
              <text name="txt_desc" pos="{0,36,-10,-10" font="bold:1,adding:-
4" dotted="1">速度最快的浏览器</text>
              <text name="txt_index" pos="|0,|0" offset="-0.5,-0.5"
font="adding:10" colorText="#ff000088">10</text>
            </window>
            <window name="col2">
              <ratingbar name="rating_score" starSkin="skin_star1"
starNum="5" value="3.5" pos="10,16"  />
              <text name="txt_score" pos="15,36,50,-16" font="adding:-
5"  >8.5 分</text>
              <link pos="[5,36,@30,-16" cursor="hand" colorText="#1e78d5"
href="www.163.com" font="adding:-5" >投票</link>
            </window>
            <window name="col3">
              <text name="txt_size" pos="0,26,-0,-26" font="adding:-4"
align="center" >85.92M</text>
            </window>
            <window  name="col4">
```

```
            <text name="txt_installtime" pos="0,26,-0,-26" font="adding:-4"
align="center" >2015-01-09</text>
          </window>
          <window name="col5">
            <text name="txt_usetime" pos="0,26,-0,-26" font="adding:-4"
align="center" >今天</text>
            <animateimg pos="|0,|0" offset="-0.5,-0.5" skin="skin_busy"
name="ani_test" tip="animateimg is used here" msgTransparent="0" />
          </window>
          <window name="col6">
            <imgbtn animate="1"  pos="|-35,|-14" font="adding:-3"
align="center" skin="skin_install" name="btn_uninstall">卸载</imgbtn>
          </window>
        </template>

      </mclistview>
```

mclistview 有一个属性 headerHeight，该属性定义表头的显示高度。

节点下有一个 header 控件，用来定义表头控件的样式，都很简单，自己看 XML。

最关键的在于下面的 template（模板）节点，该 XML 节点用来定义如何显示列表项。

模板内样式的定义其实并没有特别的规定，因为最后如何解析这个模板是由 APP 决定的，但推荐使用上面的样式：template 节点下为每一列定义一个 window 节点，只需要指定一个 name 属性（当然也可以指定其它的窗口属性，布局属性无效）。在该 window 节点下可以定义任意的其它控件。

代码编写：

和 listview 控件一样，mclistview 也需要 XML 和代码配合才能正确显示数据。

要使用 mclistview，首先需要实现一个数据适配器（IMcAdapter，继承自 SListView 中实现的 IAdapter)，还是先看 demo 中的实现：

```
class CTestMcAdapterFix : public SMcAdapterBase
{
public:
struct SOFTINFO
{
    wchar_t * pszSkinName;
    wchar_t * pszName;
    wchar_t * pszDesc;
    float     fScore;
    DWORD     dwSize;
    wchar_t * pszInstallTime;
    wchar_t * pszUseTime;
};
```

```
static SOFTINFO s_info[];

public:
    CTestMcAdapterFix()
    {
    }
    virtual int getCount()
    {
        return 12340;
    }
    SStringT getSizeText(DWORD dwSize)
    {
        int num1=dwSize/(1<<20);
        dwSize -= num1 *(1<<20);
        int num2 = dwSize*100/(1<<20);
        return SStringT().Format(_T("%d.%02dM"),num1,num2);
    }

    virtual void getView(int position, SWindow * pItem,pugi::xml_node
xmlTemplate)
    {
        if(pItem->GetChildrenCount()==0)
        {
            pItem->InitFromXml(xmlTemplate);
        }
        int dataSize = 7;
        SOFTINFO *psi = s_info+position%dataSize;

pItem->FindChildByName(L"img_icon")->SetAttribute(L"skin",psi->pszSkinName);

pItem->FindChildByName(L"txt_name")->SetWindowText(S_CW2T(psi->pszName));

pItem->FindChildByName(L"txt_desc")->SetWindowText(S_CW2T(psi->pszDesc));

pItem->FindChildByName(L"txt_score")->SetWindowText(SStringT().Format(_T("%1
.2f 分"),psi->fScore));

pItem->FindChildByName(L"txt_installtime")->SetWindowText(S_CW2T(psi->pszIns
tallTime));

pItem->FindChildByName(L"txt_usetime")->SetWindowText(S_CW2T(psi->pszUseTime
));
```

```
pItem->FindChildByName(L"txt_size")->SetWindowText(getSizeText(psi->dwSize))
;

pItem->FindChildByName2<SRatingBar>(L"rating_score")->SetValue(psi->fScore/2
);

pItem->FindChildByName(L"txt_index")->SetWindowText(SStringT().Format(_T("
第%d行"),position));

       SButton *pBtnUninstall =
pItem->FindChildByName2<SButton>(L"btn_uninstall");
       pBtnUninstall->SetUserData(position);

pBtnUninstall->GetEventSet()->subscribeEvent(EVT_CMD,Subscriber(&CTestMcAdap
terFix::OnButtonClick,this));
    }
    bool OnButtonClick(EventArgs *pEvt)
    {
        SButton *pBtn = sobj_cast<SButton>(pEvt->sender);
        int iItem = pBtn->GetUserData();
        SMessageBox(NULL,SStringT().Format(_T("button of %d item was
clicked"),iItem),_T("uninstall"),MB_OK);
        return true;
    }
    SStringW GetColumnName(int iCol) const{
        return SStringW().Format(L"col%d",iCol+1);
    }
    struct SORTCTX
    {
        int iCol;
        SHDSORTFLAG stFlag;
    };
    bool OnSort(int iCol,SHDSORTFLAG * stFlags,int nCols)
    {
        if(iCol==5) //最后一列"操作"不支持排序
            return false;
        SHDSORTFLAG stFlag = stFlags[iCol];
        switch(stFlag)
        {
            case ST_NULL:stFlag = ST_UP;break;
            case ST_DOWN:stFlag = ST_UP;break;
            case ST_UP:stFlag = ST_DOWN;break;
```

```
    }
    for(int i=0;i<nCols;i++)
    {
        stFlags[i]=ST_NULL;
    }
    stFlags[iCol]=stFlag;
    SORTCTX ctx={iCol,stFlag};
    qsort_s(s_info,7,sizeof(SOFTINFO),SortCmp,&ctx);
    return true;
}
static int __cdecl SortCmp(void *context,const void * p1,const void * p2)
{
    SORTCTX *pctx = (SORTCTX*)context;
    const SOFTINFO *pSI1=(const SOFTINFO*)p1;
    const SOFTINFO *pSI2=(const SOFTINFO*)p2;
    int nRet =0;
    switch(pctx->iCol)
    {
        case 0://name
            nRet = wcscmp(pSI1->pszName,pSI2->pszName);
            break;
        case 1://score
            {
                float fCmp = (pSI1->fScore - pSI2->fScore);
                if(fabs(fCmp)<0.0000001) nRet = 0;
                else if(fCmp>0.0f) nRet = 1;
                else nRet = -1;
            }
            break;
        case 2://size
            nRet = (int)(pSI1->dwSize - pSI2->dwSize);
            break;
        case 3://install time
            nRet = wcscmp(pSI1->pszInstallTime,pSI2->pszInstallTime);
            break;
        case 4://user time
            nRet = wcscmp(pSI1->pszUseTime,pSI2->pszUseTime);
            break;
    }
    if(pctx->stFlag == ST_UP)
        nRet = -nRet;
    return nRet;
}
```

```
};

CTestMcAdapterFix::SOFTINFO CTestMcAdapterFix::s_info[] =
{
    {
        L"skin_icon1",
        L"鲁大师",
        L"鲁大师是一款专业的硬件检测，驱动安装工具",
        5.4f,
        15*(1<<20),
        L"2015-8-5",
        L"今天"
    },
    {
        L"skin_icon2",
        L"PhotoShop",
        L"强大的图片处理工具",
        9.0f,
        150*(1<<20),
        L"2015-8-5",
        L"今天"
    },
    {
        L"skin_icon3",
        L"QQ7.0",
        L"腾讯公司出品的即时聊天工具",
        8.0f,
        40*(1<<20),
        L"2015-8-5",
        L"今天"
    },
    {
        L"skin_icon4",
        L"Visual Studio 2008",
        L"Microsoft 公司的程序开发套件",
        9.0f,
        40*(1<<20),
        L"2015-8-5",
        L"今天"
    },
    {
        L"skin_icon5",
        L"YY8",
```

```
        L"YY 语音",
        9.0f,
        20*(1<<20),
        L"2015-8-5",
        L"今天"
    },
    {
        L"skin_icon6",
        L"火狐浏览器",
        L"速度最快的浏览器",
        8.5f,
        35*(1<<20),
        L"2015-8-5",
        L"今天"
    },
    {
        L"skin_icon7",
        L"迅雷",
        L"迅雷下载软件",
        7.3f,
        17*(1<<20),
        L"2015-8-5",
        L"今天"
    }
};
```

注意 CTestMcAdapterFix::getView 虚函数，上面提到的 template 会通过该函数的参数 pugi::xml_node xmlTemplate 传递过来。

在 getView 中，首先需要判断表项容器的子窗口是不是已经被初始化过，如果没有就执行 InitFromXml 如下：

```
        if(pItem->GetChildrenCount()==0)
        {
            pItem->InitFromXml(xmlTemplate);
        }
```

在子窗口初始化完成后，还需要从数据表中获取对应项的数据填充到控件中显示：

```
        int dataSize = 7;
        SOFTINFO *psi = s_info+position%dataSize;

pItem->FindChildByName(L"img_icon")->SetAttribute(L"skin",psi->pszSkinName);

pItem->FindChildByName(L"txt_name")->SetWindowText(S_CW2T(psi->pszName));
```

```
pItem->FindChildByName(L"txt_desc")->SetWindowText(S_CW2T(psi->pszDesc));

pItem->FindChildByName(L"txt_score")->SetWindowText(SStringT().Format(_T("%1
.2f 分"),psi->fScore));

pItem->FindChildByName(L"txt_installtime")->SetWindowText(S_CW2T(psi->pszIns
tallTime));

pItem->FindChildByName(L"txt_usetime")->SetWindowText(S_CW2T(psi->pszUseTime
));

pItem->FindChildByName(L"txt_size")->SetWindowText(getSizeText(psi->dwSize))
;

pItem->FindChildByName2<SRatingBar>(L"rating_score")->SetValue(psi->fScore/2
);

pItem->FindChildByName(L"txt_index")->SetWindowText(SStringT().Format(_T("
第%d行"),position));
```

和主面板上的控件响应不同，要响应表项中控件的事件，没有事件映射表可以使用，可能在 IMcAdapter 的实现中使用控件的 GetEventSet()->subscribeEvent 方法来响应：

```
      SButton *pBtnUninstall =
pItem->FindChildByName2<SButton>(L"btn_uninstall");
      pBtnUninstall->SetUserData(position);

pBtnUninstall->GetEventSet()->subscribeEvent(EVT_CMD,Subscriber(&CTestMcAdap
terFix::OnButtonClick,this));
```

除了 getView 这个方法外，相对于 IAdapter，IMcAdapter 还需要实现另外两个非常重要的方法：

```
   interface IMcAdapter : public IAdapter
   {
      //获取列名
      virtual SStringW GetColumnName(int iCol) const PURE;
      //排序接口
      // int iCol:排序列
      // SHDSORTFLAG * stFlags [in, out]:当前列排序标志
      // int nCols:总列数,stFlags 数组长度
      virtual bool OnSort(int iCol,SHDSORTFLAG * stFlags,int nCols) PURE;
   };
```

实现 GetColumnName 方法来获取每一列对应的子窗口名称，SMcListView 通过它来确实 template 中的子窗口哪一个应该显示在什么位置，返回在 template 中定义的子节点的 name 即可。

实现 OnSort 来处理表头点击事件，以确实如何对数据排序。

至此，这个超级列表控件的使用就完成了。

4.27 在... ...

5、控件说明（SOUI）

根据“属性列表.xls”文档整理，并剔除了与代码不符的空间和属性；

个人精力有限，欢迎大家批评指正。

5.1 SWindow 类

类名：SWindow

控件名：window

基类：SObject、SMsgHandleState、TObjRefImpl2<IObjRef,SWindow>

说明：大多数界面控件本质上都是窗口，该类是大部分界面控件的基类；绝大部分控件都拥有该类的属性和方法。

属性：

属性名	类型	说明
id	数字	窗口 ID
name	字符串	窗口名称
skin	字符串	窗口绘制背景引用的 ISkinObj 对象名称
ncskin	字符串	窗口绘制非客户区边框时引用的 ISkinObj 对象名称
class	字符串	窗口的属性集合(WndStyle)的名称
data		用户数据
enable	0|1	控件是否可用（0--不可用　1--可用）
visible	0|1	控件是否可见（0--隐藏　1--可见）
show	0|1	控件是否可见（0--隐藏　1--可见）
pos	负数,|,[,],%,@	控件位置
cache	0|1	绘制缓存（0--无绘制缓存（默认） 1--有绘制缓存）
display	0|1	显示（0--隐藏不占位 1--隐藏占位（默认））
tip	字符串	提示框
msgTransparent	0|1	消息穿透（0--接收鼠标键盘消息（默认） 1--不接收）
sep	正整数	窗口默认间距
maxWidth	数字	窗口最大宽度
clipClient	0|1	是否在绘制窗口时限制内容只绘制在客户区 （0--不剪裁（默认） 1--剪裁）
focusable	0|1	窗口是否接受焦点（0--不接受　1--接受（默认））
pos2type	leftTop， center， rightTop， leftBottom， rightBottom	当 pos 属性只指定了两个值时，这两个值的意义 【该属性不建议使用，已弃用】
alpha	[0-255]	绘制窗口的透明度

函数：

函数	是否虚函数	说明

5.2 SwndStyle 类

类名：SwndStyle

控件名：style

基类：SObject

说明：大多数界面控件本质上都是窗口，该类是大部分界面控件的基类；绝大部分控件都拥有该类的属性和方法。

属性：

属性名	类型	说明
textMode	数字	文字绘制的格式
align	字符串	水平显示方式（left，center，right）
valign	字符串	垂直显示方式（top，middle，bottom）
colorBkgnd	color	背景颜色
colorBorder	color	边框颜色
font	字符串	默认字体
fontHover	字符串	悬停时字体
fontPush	字符串	按下时字体
fontDisable	字符串	失效时字体
colorText	color	默认字体颜色
colorTextHover	color	悬停时字体颜色
colorTextPush	color	按下时字体颜色
colorTextDisable	color	失效时字体颜色
margin-x	数字	非客户区 left 及 right 宽度
margin-y	数字	非客户区 top 及 bottom 宽度
margin	数字	非客户区宽度
cursor	字符串	光标（指定光标资源名称）
dotted	0|1	在文件绘制长度超出客户区时是否使用"..." （0--不使用（默认） 1--使用）

函数：

函数	是否虚函数	说明

函数	是否虚函数	说明

示例：

```
<text pos="|0,[5" font="face:微软雅黑,adding:-3" colorText="#000000">文本...</text>
```

5.3 SHostWndAttr 类

类名：SHostWndAttr

控件名：hostwndattr

基类：SObject

说明：大多数界面控件本质上都是窗口，该类是大部分界面控件的基类；绝大部分控件都拥有该类的属性和方法。

属性：

属性名	类型	说明
name		名称
title		标题
size		窗口大小
width		宽度
height		高度
margin		位置
minsize		最小大小
wndStyle		样式
wndStyleEx		扩展样式
resizable	0|1	host 可变大小（0--不可变 1--可变大小）
translucent		host 半透明
appWnd		host 在任务栏显示图标
toolWindow	0|1	host 的扩展样式是否为 WS_EX_TOOLWINDOW（0--无 1--有）
smallIcon	iconname:size	小图标
bigIcon	iconname:size	大图标

函数：

函数	是否虚函数	说明

函数	是否虚函数	说明

5.4 静态文本控件

类名：SStatic

控件名：text

基类：SWindow

说明：静态文本控件可支持多行，有多行属性时，\n 可以强制换行。

属性：

属性名	类型	说明
multiLines	0|1	多行显示（0--否 1--是）
interHeight	数字	多行显示时行间距

函数：

函数	是否虚函数	说明

5.5 超链接控件

类名：SLink

控件名：link

基类：SWindow

说明：… …。

属性：

属性名	类型	说明
href	字符串	超链接

函数：

函数	是否虚函数	说明

5.6 按钮/图片按钮控件

类名：SButton、SImageButton

控件名：button、imgbtn

基类：SWindow、IAcceleratorTarget、ITimelineHandler

说明：显示按钮，是从 SWindow 派生的，所以基类控件具有的属性和函数也可以使用。

按钮的背景图片是由 4 张切图拼接成的大图片，分别是普通状态、鼠标移动状态、鼠标按下状态、禁用状态的图片，背景图片设置可以通过 skin 属性设置。

按钮控件有两种控件名，button 和 imgbtn，唯一的差别是 imgbtn 默认会无法获取焦点，即用 Tab 键无法选中 imgbtn 按钮。

属性：

属性名	类型	说明
accel	加速键格式	键盘加速键
animate	0|1	动画效果（0--关闭 1--启用）

函数：

函数	是否虚函数	说明

示例：

5.7 图片控件

类名：SImageWnd

控件名：img

基类：SWindow

说明：Image Control 图片控件类。

属性：

属性名	类型	说明
skin	字符串	skin 名称
iconIndex	数字	绘制 skin 对象的子图索引

函数：

函数	是否虚函数	说明

5.8 动画图片窗口控件

类名：SAnimateImgWnd

控件名：animateimg

基类：SWindow、ITimelineHandler

说明：图片控件类，此窗口支持动画效果。

属性：

属性名	类型	说明
skin	字符串	绘制窗口的 Skin（基类的 Skin 将失效）
speed	数字	速度
autoStart	0|1	是否自动运行（0--关闭 1--启用）

函数：

函数	是否虚函数	说明

5.9 线条控件

类名：SLine

控件名：hr

基类：SWindow

说明：图片控件类，此窗口支持动画效果。

属性：

属性名	类型	说明
size	数字	大小
mode	字符串	方向（vertical,horizontal,tilt）
lineStyle	字符串	样式（solid,dash,dot,dashdot,dashdotdot）

函数：

函数	是否虚函数	说明

5.10 复选框控件

类名：SCheckBox

控件名：check

基类：SWindow

说明：复选框控件。

属性：

属性名	类型	说明
skin	字符串	默认皮肤
focusSkin	字符串	焦点皮肤
checked	0|1	是否选中（0--不选中 1--选中）

函数：

函数	是否虚函数	说明

5.11 图标控件

类名：SIconWnd

控件名：icon

基类：SWindow

说明：复选框控件。

属性：

属性名	类型	说明
src	字符串	指定 icon 图标（iconname:size）

函数：

函数	是否虚函数	说明

5.12 单选框控件

类名：SRadioBox

控件名：radio

基类：SWindow

说明：复选框控件。

属性：

属性名	类型	说明
skin	字符串	默认皮肤
focusSkin	字符串	焦点皮肤
checked	0|1	是否选中（0--不选中 1--选中）

函数：

函数	是否虚函数	说明

5.13 Toggle 控件

类名：SToggle

控件名：toggle

基类：SWindow

说明：Toggle 控件。

属性：

属性名	类型	说明
skin	字符串	皮肤
Toggled		

函数：

函数	是否虚函数	说明

5.14 组控件

类名：SGroup

控件名：group

基类：SWindow

说明：组控件，<group colorLine1="#b8d5e2" colorLine2="#999999">group text</>。

属性：

属性名	类型	说明
colorLine1	color	颜色
colorLine2	color	颜色
round	数字	半径

函数：

函数	是否虚函数	说明

5.15 可输入 CommboBox 控件

类名：SComboBox

控件名：combobox

基类：SComboBase 〖SWindow、ISDropDownOwner〗

说明：组控件，<group colorLine1="#b8d5e2" colorLine2="#999999">group text</>。

属性：

属性名	类型	说明
colorLine1	color	颜色
colorLine2	color	颜色
round	数字	半径

函数：

函数	是否虚函数	说明

5.16 日历控件

类名：SCalendar

控件名：calendar

基类：SWindow

说明：此类是日历的核心类 大部分函数都是静态函数。

属性：

属性名	类型	说明
titleHeight	数字	标题高度
footerHeight	数字	底部高度
colorWeekend	color	周末颜色
colorTitleBack	color	标题颜色
colorDay	color	日期颜色
daySkin	字符串	默认按钮皮肤
titleSkin	字符串	默认标题皮肤
title-1	字符串	标题周 1 的文字
title-2	字符串	标题周 2 的文字

属性名	类型	说明
title-3	字符串	标题周 3 的文字
title-4	字符串	标题周 4 的文字
title-5	字符串	标题周 5 的文字
title-6	字符串	标题周 6 的文字
title-7	字符串	标题周 7 的文字

函数：

函数	是否虚函数	说明

5.17 标签控件

类名：SCaption

控件名：caption

基类：SWindow

说明：标签类 只需要继承此类即可

属性：

属性名	类型	说明
		无

函数：

函数	是否虚函数	说明

5.18 富文本编辑框控件

类名：SRichEdit

控件名：richedit

基类：SPanel〖SWindow〗

说明：使用 Windowless Richedit 实现的 edit 控件

属性：

属性名	类型	说明
style	数字	richedit 的 style
maxBuf		最大容纳字符
transparent	0|1	指示 edit 控件是否背景透明，默认透明 （0--不透明 1--透明）
rich	0|1	是否为富文本控件，默认 richedit （0--普通 edit 1--richedit）

属性名	类型	说明
vertical	0|1	是否允许垂直（0--禁止　1--允许）
wordWrap	0|1	英文单词可以折断（0--禁止　1--允许）
allowBeep	0|1	错误时有提示音（0--禁止　1--允许）
autoWordSel		自动选择单词
vcenter	0|1	单行垂直居中（0--否　1--是）
inset	数字	edit 的边缘大小（inset="5,5,5,5"）
colorText	color	文本颜色

函数：

函数	是否虚函数	说明

5.19 简单 edit 控件

类名：SEdit

控件名：edit

基类：SRichEdit

说明：

属性：

属性名	类型	说明
		无

函数：

函数	是否虚函数	说明

5.20 表头控件

类名：SHeaderCtrl

控件名：header

基类：SWindow

说明：表头控件

属性：

属性名	类型	说明
itemSkin	字符串	皮肤
sortSkin	字符串	皮肤
fixWidth	0|1	是否固定宽度（0--否　1--是）
itemSwapEnable	0|1	是否允许调整位置（0--禁止　1--允许）

属性名	类型	说明
sortHeader	0|1	是否支持排序（0--否　1--是）

函数：

函数	是否虚函数	说明

5.21 热键控件

类名：SHotKeyCtrl

控件名：hotkey

基类：SWindow、CAccelerator

说明：热键控件

属性：

属性名	类型	说明
invalidComb		无效的组合键
defCombKey		对无效组合键的替换方案,默认方案
combKey		组合键
hotkey		热键

函数：

函数	是否虚函数	说明

5.22 列表框控件

类名：SListBox

控件名：listbox

基类：SScrollView〖SPanel〗

说明：热键控件

属性：

属性名	类型	说明
scrollSpeed	数字	滚动速率
itemHeight	数字	表项高度
itemSkin	字符串	皮肤
iconSkin	字符串	皮肤
colorItemBkgnd	color	表项背景色
colorItemBkgnd2	color	表项背景色

属性名	类型	说明
colorItemSelBkgnd	color	选中表项背景色
colorText	color	文本颜色
colorSelText	color	选中文本颜色
icon-x	数字	图标 x 坐标
icon-y	数字	图标 y 坐标
text-x	数字	文本 x 坐标
text-y	数字	文本 x 坐标
hotTrack	0|1	鼠标移动时高亮显示下面的行（0--不支持　1--支持）

函数：

函数	是否虚函数	说明

5.23 列表控件

类名：SListCtrl

控件名：listctrl

基类：SPanel

说明：列表控件，是从 SPanel 派生的，所以基类控件具有的属性和函数也可以使用。

属性：

属性名	类型	说明
headerHeight	数字	表头高度
itemHeight	数字	表项高度
itemSkin	字符串	皮肤
iconSkin	字符串	皮肤
colorItemBkgnd	color	表项背景色
colorItemBkgnd2	color	表项背景色
colorItemSelBkgnd	color	选中表项背景色
colorText	color	文本颜色
colorSelText	color	选中文本颜色
icon-x	数字	图标 x 坐标
icon-y	数字	图标 y 坐标
text-x	数字	文本 x 坐标
text-y	数字	文本 y 坐标
hotTrack	0|1	鼠标移动时高亮显示下面的行（0--不支持　1--支持）

函数：

函数	是否虚函数	说明

5.24 滚动条控件

类名：SScrollBar

控件名：scrollbar

基类：SWindow

说明：滚动条控件

属性：

属性名	类型	说明
skin	字符串	皮肤
arrowSize	数字	箭头大小
min	数字	最小值
max	数字	最大值
value	数字	当前值
page	数字	翻页大小
vertical	0|1	是否垂直（0-- 否　1--是）

函数：

函数	是否虚函数	说明

5.25 滑块工具条控件

类名：SSliderBar

控件名：sliderbar

基类：SProgress

说明：滑块工具条控件

属性：

属性名	类型	说明
thumbSkin		拖动按钮的皮肤

函数：

函数	是否虚函数	说明

5.26 分割窗口控件

类名：SSplitPane

控件名：pane

基类：SWindow

说明：分割窗口控件

属性：

属性名	类型	说明
idealSize	数字	期望大小
minSize	数字	最小大小
priority	数字	优先级（值越小优先级越高）

函数：

函数	是否虚函数	说明

5.27 分割窗口控件

类名：SSplitWnd

控件名：splitwnd

基类：SWindow

说明：分割窗口控件

属性：

属性名	类型	说明
sepSize	数字	分隔条宽度/高度
adjustable	0|1	是否可以拖动（0-不可拖动，1-可以拖动）
colmode	0|1	分栏模式（0-横向分栏，1-纵向分栏）
skinSep	字符串	分隔条绘制皮肤（皮肤 name）

函数：

函数	是否虚函数	说明

5.28 Tab 标签页面控件

类名：STabPage

控件名：page

基类：SWindow

说明：滚动条控件

属性：

属性名	类型	说明
title	字符串	标题（只能在 STabCtrl 中使用）

函数：

函数	是否虚函数	说明

5.29 Tab 控件

类名：STabCtrl

控件名：tabctrl

基类：SWindow

说明：滚动条控件

属性：

属性名	类型	说明
curSel	数字	选中索引
tabWidth	数字	tab 宽度
tabHeight	数字	tab 高度
tabPos	数字	tab 开始偏移
framePos	坐标	坐标
tabInterSize	数字	tab 标签间隙
tabInterSkin	字符串	分割线皮肤
tabSkin	字符串	tab 皮肤
iconSkin	字符串	icon 皮肤
frameSkin	字符串	框架皮肤
icon-x	数字	图标 x 坐标
icon-y	数字	图标 y 坐标
text-x	数字	文本 x 坐标
text-y	数字	文本 y 坐标
tabAlign	top left	显示方式
animateSteps	int	动画切换页面时动画次数

函数：

函数	是否虚函数	说明

5.30 树控件

类名：STreeCtrl

控件名：treectrl

基类：SScrollView、CSTree<LPTVITEM>

说明：树控件

属性：

属性名	类型	说明
indent	数字	缩进字符数
itemHeight	数字	表项高度
itemMargin		位置（margin 相对位置，比如说上,下，左，右）
checkBox	0|1	是否有复选框（0--没有 1--有）
rightClickSel	0|1	右击选中（0--不支持 1--支持）
itemBkgndSkin	color	表项背景色
itemSelSkin	字符串	皮肤
toggleSkin	字符串	皮肤
iconSkin	字符串	皮肤
checkSkin	字符串	皮肤
colorItemBkgnd	color	表项背景色
colorItemSelBkgnd	color	表项背景色
colorrItemText	color	表项背景色
colorItemSelText	color	表项背景色

函数：

函数	是否虚函数	说明

5.24 按钮控件

类名：SSkinButton

控件名：button

基类：SSkinObjBase

说明：按钮控件

属性：

属性名	类型	说明
colorBorder	color	边框颜色
colorUp	color	正常状态的渐变色上沿色
colorDown	color	正常状态的渐变色下沿色
colorUpHover	color	浮动状态的渐变色上沿色

属性名	类型	说明
colorDownHover	color	浮动状态的渐变色下沿色
colorUpPush	color	下压状态的渐变色上沿色
colorDownPush	color	下压状态的渐变色下沿色
colorUpDisable	color	禁用状态的渐变色上沿色
colorDownDisable	color	禁用状态的渐变色下沿色

函数：

函数	是否虚函数	说明

5.24 滚动条控件

类名：SScrollBar

控件名：scrollbar

基类：SWindow

说明：滚动条控件

属性：

属性名	类型	说明
skin	字符串	皮肤
arrowSize	数字	箭头大小
min	数字	最小值
max	数字	最大值
value	数字	当前值
page	数字	翻页大小
vertical	0|1	是否垂直（0-- 否　1--是）

函数：

函数	是否虚函数	说明

5.24 SSkinGif 控件

类名：SSkinGif

控件名：gif

基类：SWindow

说明：滚动条控件

属性：

属性名	类型	说明
skin	字符串	皮肤
arrowSize	数字	箭头大小
min	数字	最小值
max	数字	最大值
value	数字	当前值
page	数字	翻页大小
vertical	0|1	是否垂直（0-- 否　1--是）

函数：

函数	是否虚函数	说明

5.24 SSkinGradation 控件

类名：SSkinGradation

控件名：gradation

基类：SWindow

说明：滚动条控件

属性：

属性名	类型	说明
skin	字符串	皮肤
arrowSize	数字	箭头大小
min	数字	最小值
max	数字	最大值
value	数字	当前值
page	数字	翻页大小
vertical	0|1	是否垂直（0-- 否　1--是）

函数：

函数	是否虚函数	说明

5.24 SSkinImgList 控件

类名：SSkinImgList

控件名：imglist

基类：SWindow

说明：滚动条控件

属性：

属性名	类型	说明
skin	字符串	皮肤
arrowSize	数字	箭头大小
min	数字	最小值
max	数字	最大值
value	数字	当前值
page	数字	翻页大小
vertical	0|1	是否垂直（0-- 否　1--是）

函数：

函数	是否虚函数	说明

5.24 SSkinScrollbar 控件

类名：SSkinScrollbar

控件名：scrollbar

基类：SWindow

说明：滚动条控件

属性：

属性名	类型	说明
skin	字符串	皮肤
arrowSize	数字	箭头大小
min	数字	最小值
max	数字	最大值
value	数字	当前值
page	数字	翻页大小
vertical	0|1	是否垂直（0-- 否　1--是）

函数：

函数	是否虚函数	说明

5.24 SSkinImgFrame 控件

类名：SSkinImgFrame

控件名：imgframe

基类：SWindow

说明：滚动条控件

属性：

属性名	类型	说明
skin	字符串	皮肤
arrowSize	数字	箭头大小
min	数字	最小值
max	数字	最大值
value	数字	当前值
page	数字	翻页大小
vertical	0|1	是否垂直（0-- 否　1--是）

函数：

函数	是否虚函数	说明

5.24 SSkinMenuBorder 控件

类名：SSkinMenuBorder

控件名：border

基类：SWindow

说明：滚动条控件

属性：

属性名	类型	说明
skin	字符串	皮肤
arrowSize	数字	箭头大小
min	数字	最小值
max	数字	最大值
value	数字	当前值
page	数字	翻页大小
vertical	0|1	是否垂直（0-- 否　1--是）

函数：

函数	是否虚函数	说明

6、常见问题

本章节收录了在使用 SOUI 界面库时常遇到的问题，也欢迎大家把遇到的问题发给我们。

6.0 未解决的问题？

1、窗体、静态文本控件的字体设置？

字体颜色：SwndStyle 类属性 colorText

示例：colorText=”#00ff00” 或 colorText=”rgb(255,255,255)”

字体：SwndStyle 类属性 font

示例：font=”???” 字符串怎么填写？

2、……

6.1 模块 utilities 为什么要用 DLL 编译？

SOUI 相对于 DuiEngine 一个重要的变化就是很多模块变成了一个单独的 DLL。

然后很多情况下用户可能希望整个产品就是一个 EXE，原来 DuiEngine 提供了 LIB 编译模式，此时链接 LIB 模式的 DuiEngine 就行了。

但是 SOUI 默认至少 Utilities 那个模块是不提供 LIB 编译模式的。

utilities 之所以默认只提供 DLL 编译是因为 SString 类是由 utilities 实现的。

字符串是编译中碰到的最最见的基本对象之一。在运行库（CRT）动态编译（MD，MDd）时这不是问题，因为所有模块的内存分配都是在一个相同的运行库（CRT）上，这时在不同模块之间传递对象相对简单。如果项目采用运行库静态编译（MT or MTd)，在不同模块之间传递字符串对象是非常困难的，因为一不小心就会发生在 A 模块中分配的字符串对象被 B 模块释放。

utilities 采用 DLL 编译就是为了解决这个字符串对象的跨模块传递。

采用运行库动态编译的情况就不说了，这里主要介绍采用静态库编译的 CRT 的情况。

SOUI 中使用的字符串对象采用了一点技巧：每一个 String 对象中只有一个指针成员变量：

```
    template <class tchar, class tchar_traits>
    class TStringT
    {
    public:
        typedef tchar    _tchar;
        typedef const _tchar * pctstr;
    protected:
        tchar* m_pszData;   // pointer to ref counted string
data
    };
```

虽然 TStringT 是一个模板类，在 SOUI 中采用类导出的方式将该模板的两个特化类导出：

```
#ifdef UTILITIES_EXPORTS
#    define EXPIMP_TEMPLATE
#else
#    define EXPIMP_TEMPLATE extern
#endif

    #pragma warning (disable : 4231)

   EXPIMP_TEMPLATE template class UTILITIES_API
TStringT<char, char_traits>;
   EXPIMP_TEMPLATE template class UTILITIES_API
TStringT<wchar_t, wchar_traits>;
```

通过将 string 类导出，保证 string 的所有运行代码都是在 utilities 这个模块内部，这也就保证了 string 对象的唯一成员变量：

tchar* m_pszData；

的内存分配及释放固定在 utilities 这个模块里。

通过这样处理，无论用户定义 string 是在哪一个模块，真正的内存管理还是在 utilities 里，从而使得 string 对象可以方便的在不同模块之间传递。

比较一下 std::string 就可以发现，如果使用 std::string 在不同模块之间传递对象将是非常危险的，因为 std::string 是模板类，它的代码将会被编译到不同的模块中，也就是说在不同的模块中调用 std::string 的成员函数执行的代码是不一样的，这样在 A 模块中声明的 string 传递到 B 模块再被 B 模块释放程序就崩溃了。

这就是为什么 utilities 模块默认只提供 DLL 编译的原因。

知道了原因就好办了。

对于那些希望整个项目就是一个 EXE 的情况，直接修改 utilities 模块的编译类型为 LIB 就行了，因为这种情况下根本不存在跨模块对象传递的问题。

6.2 为什么在 soui 中加载 JPG 文件失败？

在 SOUI 中解决解码器是一个独立的模块。

不同的解码器决定了程序中能够加载什么样的图片类型。

使用 SComMgr 来加载 SOUI 的模块时，debug 模式下默认的图片解码器是 imgdecoder-png。这个解码器只能解码 PNG 图片。至于为什么用这个解码器作为 debug 版本的默认解码器是为了演示在 SOUI 中使用 APNG 动画，只有这个解码器支持 APNG 解码。

要使用其它解码器只需要在实例化 SComMgr 时提供一个解码器参数就行：

```
class SComMgr
```

```
{
public:
    SComMgr(LPCTSTR pszImgDecoder = NULL)
    {
        if(pszImgDecoder) m_strImgDecoder = pszImgDecoder;
        else m_strImgDecoder = COM_IMGDECODER;
    }

    BOOL CreateImgDecoder(IObjRef ** ppObj)
    {
        if(m_strImgDecoder == _T("imgdecoder-wic"))
            return SOUI::IMGDECODOR_WIC::SCreateInstance(ppObj);
        else if(m_strImgDecoder == _T("imgdecoder-stb"))
            return SOUI::IMGDECODOR_STB::SCreateInstance(ppObj);
        else if(m_strImgDecoder == _T("imgdecoder-png"))
            return SOUI::IMGDECODOR_PNG::SCreateInstance(ppObj);
        else if(m_strImgDecoder == _T("imgdecoder-gdip"))
            return SOUI::IMGDECODOR_GDIP::SCreateInstance(ppObj);
        else
        {
            SASSERT(0);
            return FALSE;
        }
    }
    //...
}
```

可以看出 SOUI 实现了 4 种图片解码器，除了 imgdecoder-png 外，其它 3 个都是全图片格式支持的。

因此只需要使用

SComMgr *pComMgr = new SComMgr("imgdecoder-gdip");

代替

SComMgr *pComMgr = new SComMgr();

即可实现 JPG 的解码。

6.3 不注册 COM 在 Richedit 中使 OLE 支持复制粘贴

正常情况下在 Richedit 中使用 OLE，如果需要 OLE 支持复制粘贴，那么这个 OLE 对象必须是已经注册的 COM 对象。

注册 COM 很简单，关键问题在于注册时需要管理员权限，这样一来，如果希望 APP 做成绿色版本就不好使了。

为什么需要注册成 COM？因为在粘贴时 Richedit 需要能够从 COM 对象的 GUID 实例化出你的 OLE 对象。

从一个 COM 的 GUID 创建一个 COM 对象，必然需要通过 CoCreateInstance(Ex)这个系统 API。那么我们是不是只要 Hook 到这个 API 就可以不需要注册了呢？

通过 Hook CoCreateInstance，我们发现创建已经注册的 COM 确实会到 CoCreateInstance 这个 API 里来。然而在试图粘贴未注册的 COM 对象时，确并没有走到自己 Hook 的 CoCreateInstance，粘贴并没有成功。

为什么呢？

既然是粘贴，我们可以先看一下剪贴板里有些什么东西，随便找一个剪贴板查看的工具，看一下里面的 RTF 格式里有些什么东西。当复制的是注册的 COM 时，RTF 里有这个 COM 的字符串 ID，而当复制没有注册的 COM 时，剪贴板里没有这些信息。

问题出在哪呢？

有 Richedit 的原代码就好了。

正好有一份 Wince 里的 Richedit 的源代码，通过分析 Richedit 的复制的代码，可以发现在复制 OLE 对象时，先要调用 ProgIDFromCLSID 来查询这个对象的 ProgID。

如果一个 OLE 对象没有注册，那么 ProgIDFromCLSID 会返回失败，从而导致复制阶段就失败了。

知道了这个流程就好办了，继续 HOOK，把 ProgIDFromCLSID 加到 HOOK 表就好了。

HOOK 了这个函数后发现粘贴时能够执行 CoCreateInstance 了，我们也可以自己实例化这个等待粘贴的对象了，但事实是还是粘贴失败了，因为在执行这个对象的 Load 方法前还没有设置 ClientSite 对象，而我的这个表情对象需要从这个 ClintSite 来 QueryInterface 出一个自己定义的接口，没有这个接口对象就没有办法初始化。

为什么呢？为什么呢？

如果有 Windows 的原代码查一下就好了。

对了，可以看看 Wine 里 OleLoad 是怎么做的（通过调用栈可以知道 Richedit 里直接调用的是 OleLoad)。Wine 是一个在 Linux 上运行 Windows 程序的开源框架，里面有各种 Windows API 的实现，虽然和 Windows 还是不一样，但大体流程差不多了。

https://source.winehq.org/ 这个网站不错，想看哪个 API 的实现搜索一下就出来。

```
/***********************************************************************
***
*            OleLoad       [OLE32.@]
 */
HRESULT WINAPI OleLoad(
  LPSTORAGE      pStg,
```

```
  REFIID        riid,
  LPOLECLIENTSITE pClientSite,
  LPVOID*       ppvObj)
{
  IPersistStorage* persistStorage = NULL;
  IUnknown*       pUnk;
  IOleObject*     pOleObject     = NULL;
  STATSTG         storageInfo;
  HRESULT         hres;

  TRACE("(%p, %s, %p, %p)\n", pStg, debugstr_guid(riid), pClientSite,
ppvObj);

  *ppvObj = NULL;

  /*
   * TODO, Conversion ... OleDoAutoConvert
   */

  /*
   * Get the class ID for the object.
   */
  hres = IStorage_Stat(pStg, &storageInfo, STATFLAG_NONAME);
  if (FAILED(hres))
    return hres;

  /*
   * Now, try and create the handler for the object
   */
  hres = CoCreateInstance(&storageInfo.clsid,
            NULL,
            CLSCTX_INPROC_HANDLER|CLSCTX_INPROC_SERVER,
            riid,
            (void**)&pUnk);

  /*
   * If that fails, as it will most times, load the default
   * OLE handler.
   */
  if (FAILED(hres))
  {
    hres = OleCreateDefaultHandler(&storageInfo.clsid,
                NULL,
```

```
                riid,
                (void**)&pUnk);
  }

  /*
   * If we couldn't find a handler... this is bad. Abort the whole thing.
   */
  if (FAILED(hres))
    return hres;

  if (pClientSite)
  {
    hres = IUnknown_QueryInterface(pUnk, &IID_IOleObject, (void
**)&pOleObject);
    if (SUCCEEDED(hres))
    {
        DWORD dwStatus;
        hres = IOleObject_GetMiscStatus(pOleObject, DVASPECT_CONTENT,
&dwStatus);
    }
  }

  /*
   * Initialize the object with its IPersistStorage interface.
   */
  hres = IUnknown_QueryInterface(pUnk, &IID_IPersistStorage,
(void**)&persistStorage);
  if (SUCCEEDED(hres))
  {
    hres = IPersistStorage_Load(persistStorage, pStg);

    IPersistStorage_Release(persistStorage);
    persistStorage = NULL;
  }

  if (SUCCEEDED(hres) && pClientSite)
    /*
     * Inform the new object of its client site.
     */
    hres = IOleObject_SetClientSite(pOleObject, pClientSite);

  /*
   * Cleanup interfaces used internally
```

```
   */
  if (pOleObject)
    IOleObject_Release(pOleObject);

  if (SUCCEEDED(hres))
  {
    IOleLink *pOleLink;
    HRESULT hres1;
    hres1 = IUnknown_QueryInterface(pUnk, &IID_IOleLink, (void **)&pOleLink);
    if (SUCCEEDED(hres1))
    {
      FIXME("handle OLE link\n");
      IOleLink_Release(pOleLink);
    }
  }

  if (FAILED(hres))
  {
    IUnknown_Release(pUnk);
    pUnk = NULL;
  }

  *ppvObj = pUnk;

  return hres;
}
```

上面是 Wine 1.9.4 里 OleLoad 的源代码。

可以看到在执行 IPersistStorage_Load 前会先调用 IOleObject_GetMiscStatus 这个方法，然而从代码来看它并没有什么用啊？

哦，我差点忘记了，这是 Wine，并不是真正的 Windows 的代码。赶紧查一下 IOleObject_GetMiscStatus 应该返回什么。

不看不知道，一看吓一跳，原来这里有一个 OLEMISC_SETCLIENTSITEFIRST 这个标志，看名字就知道有这个标志时，会先调用 SetClientSite 再调用 Load。

好了，改写 OLE 的这个方法，不去查注册表，直接返回这个标志就好了。

到这里，一个不需要注册的 OLE 对象就完成了。

写这么多，希望看到的人能够有所启发。

7、其它未收录的博客

7.1 谈谈 SOUI 与 WTL

如果你想使用 SOUI 最好有点 WTL 基础，一点点就行了。

SOUI 不依赖于 WTL，但是 SOUI 的编码风格基本和 WTL 一样的：SOUI 抄袭了 WTL 的消息处理形式，SOUI 的事件处理也是模仿了 WTL 的消息映射宏。

抄袭 WTL 的消息处理形式表现在两个层次：

1、在 SWindow 及其派生类中处理消息使用 WTL 基本一致的消息映射宏：

```
    SOUI_MSG_MAP_BEGIN()
        MSG_WM_PAINT_EX(OnPaint)
        MSG_WM_DESTROY(OnDestroy)
        MSG_WM_LBUTTONDOWN(OnLButtonDown)
        MSG_WM_MOUSEMOVE(OnMouseMove)
        MSG_WM_MOUSELEAVE(OnMouseLeave)
        MSG_WM_KEYDOWN(OnKeyDown)
    SOUI_MSG_MAP_END()
```

上面是一个 SOUI 控件中处理消息映射的映射表，除了 SOUI_MSG_MAP_BEGIN/END() 和 WTL 有点区别外，中间的宏基本是一样的。不同在于 SOUI 中 WM_PAINT 消息的参数不是 HDC，而且 IRenderTarget，所以使用 SOUI 扩展的映射宏：MSG_WM_PAINT_EX 来处理。

2、在 CSimpleWnd 的派生类（包括 SHostWnd, SHostDialog）中直接使用 WTL 的消息映射宏处理真窗口消息：

```
//HOST 消息及响应函数映射表
BEGIN_MSG_MAP_EX(CMainDlg)
    MSG_WM_CREATE(OnCreate)
    MSG_WM_INITDIALOG(OnInitDialog)
    MSG_WM_DESTROY(OnDestory)
    MSG_WM_CLOSE(OnClose)
    MSG_WM_SIZE(OnSize)
    MSG_WM_COMMAND(OnCommand)
    MSG_WM_SHOWWINDOW(OnShowWindow)
    CHAIN_MSG_MAP(SHostWnd)
    REFLECT_NOTIFICATIONS_EX()
END_MSG_MAP()
```

由于 SWindow 的消息来自 SHostWnd 的消息分发，如果在 SHostWnd 或者 SHostDialog 的派生类中处理了一个消息，如果没有将该消息交给 SHostWnd 继续处理，可能导致 SOUI 不能正常工作。

没有 WTL 使用经历的朋友可能想知道如何将一个消息交给 SHostWnd 继续处理。当用户在自己的消息映射表中增加一个消息处理函数，而且是插入在映射表的 CHAIN_MSG_MAP(SHostWnd)前（也应该在此之前，否则很可能就收不到消息），默认情况下会自动标志该消息已经被处理了，如此一来就不会继续交给 SHostWnd 处理，解决办法很简单，在你的消息处理函数中增加一行：

```
    SetMsgHandled(FALSE);
```

有了这一行，你就不用担心你的消息处理影响到 SHostWnd 的处理了。

很多朋友在使用 SOUI 时处理自己的 Timer，结果导致 SOUI 中的动画不动了，就是因为这个原因：SOUI 内部需要处理自己的定时器消息，但它被最外层的消息映射表截断了。

基本上会上面这样一点 WTL 相关的知识就可以玩转 SOUI 了。

7.2 在 SOUI 中非半透明窗口如何实现圆角窗口？

如果 SOUI 的宿主窗口没有包含子窗口，直接使用窗口的半透明属性：translucent=1 就可以解决了，整个窗口形状完全由背景图决定，可以实现完美的圆角。

然后窗口半透明时，窗口中的子窗口(非 SWindow)就不能正常显示，所以有时候不得不使用 translucent=0，这时窗口就成了方形。

实际上这个问题已经和 SOUI 没有什么关系了，你的问题变成了窗口如何做圆角，还不是在 SOUI 中窗口如何做圆角。

网上一搜索一大堆，可惜经常有人要问。

给窗口做圆角或者异形好像只有一个办法：SetWindowRgn，自己创建一个 HRGN，再调用这个 API 就可以了，关键问题是在窗口大小变化时注意重新设置。

好人做到底，这里帖一份专业做圆角的代码：

```
template <class T>
class CWHRoundRectFrameHelper
{
protected:
    SIZE m_sizeWnd;
    void OnSize(UINT nType, CSize size)
    {
        T *pT = static_cast<T*>(this);
        if (nType == SIZE_MINIMIZED)
            return;
        if (size == m_sizeWnd)
            return;
```

```
        CRect rcWindow, rcClient;
        CRgn rgnWindow, rgnMinus, rgnAdd;
        pT->CSimpleWnd::GetWindowRect(rcWindow);
        pT->CSimpleWnd::GetClientRect(rcClient);
        pT->CSimpleWnd::ClientToScreen(rcClient);
        rcClient.OffsetRect(- rcWindow.TopLeft());
        rgnWindow.CreateRectRgn(rcClient.left, rcClient.top + 2,
rcClient.right, rcClient.bottom - 2);
        rgnAdd.CreateRectRgn(rcClient.left + 2, rcClient.top, rcClient.right -
2, rcClient.top + 1);
        rgnWindow.CombineRgn(rgnAdd, RGN_OR);
        rgnAdd.OffsetRgn(0, rcClient.Height() - 1);
        rgnWindow.CombineRgn(rgnAdd, RGN_OR);
        rgnAdd.SetRectRgn(rcClient.left + 1, rcClient.top + 1, rcClient.right
- 1, rcClient.top + 2);
        rgnWindow.CombineRgn(rgnAdd, RGN_OR);
        rgnAdd.OffsetRgn(0, rcClient.Height() - 3);
        rgnWindow.CombineRgn(rgnAdd, RGN_OR);
        pT->CSimpleWnd::SetWindowRgn(rgnWindow);
        pT->SetMsgHandled(FALSE);
        m_sizeWnd = size;
    }
public:
    BOOL ProcessWindowMessage(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM
lParam, LRESULT& lResult, DWORD dwMsgMapID = 0)
    {
        BOOL bHandled = TRUE;
        switch(dwMsgMapID)
        {
        case 0:
            if (uMsg == WM_SIZE)
            {
                OnSize((UINT)wParam, _WTYPES_NS::CSize(GET_X_LPARAM(lParam),
GET_Y_LPARAM(lParam)));
                lResult = 0;
            }
            break;
        }
        return FALSE;
    }
};
```

这是一个模板类，下面给一个 SOUI 中使用的示例代码：

```
class CFuckDialog : public SHostDialog , public
CWHRoundRectFrameHelper<CFuckDialog>
{
//xxxxx
    //HOST 消息及响应函数映射表
    BEGIN_MSG_MAP_EX(CMainDlg)
        CHAIN_MSG_MAP(CWHRoundRectFrameHelper<CFuckDialog >)//重要
        CHAIN_MSG_MAP(SHostDialog)
        REFLECT_NOTIFICATIONS_EX()
    END_MSG_MAP()
//xxxx
};
```

这样你的 Dialog 就有圆角了。

7.3 谈谈 UI 神器-SOUI

前言

在 Windows 平台上开发客户端产品是一个非常痛苦的过程，特别是还要用 C++的时候。尽管很多语言很多方法都可以开发 Windows 桌面程序，目前国内流行的客户端产品都是 C++开发的，比如 QQ，YY 语音，迅雷等。

快速，稳定是我认为的应用软件开发框架最基本的要求，对于 UI 还有两个要求就是界面美观，配置灵活。

C++语言满足了快速的要求，传统的客户端软件开发框架如 MFC，WTL 等满足了稳定的要求。然而界面美观，配置灵活是 MFC，WTL 这样的开发框架所不能满足的。

腾讯是做客户端发家的，他们的 UI 经验积累非常好，有自己专门的 UI 框架；迅雷有一个专业的团队开发自己的 UI 框架；然而大多数公司只希望有一个能够快速完成项目开发的 UI 库来使用，它们没有专业的团队来维护 UI 库。国企有钱任性，所以成就了 UIPower：一个商业化的 DirectUI 库（具体怎么样不好说，优点在于有人给你服务），一般的小公司没有谁愿意当这个冤大头。这就是 Duilib 这样一个简单到简陋的 UI 库（请原谅我这样说）为什么这样流行的原因（百度一下 Duilib 就知道它有多少人在用）。

Duilib 基本满足了界面美观 ，配置灵活的需求，然而由于框架本身的限制，要实现复杂的效果将不可避免的遇到各种坑。好在 Duilib 代码量很少，随便一个有经验的 UI 开发工程师都能够相对容易的使用并修改它，所以在一般的应用中使用并不会有太大的问题，这也应该是为什么会有那么多的 Duilib 变种的原因：每一个使用它的公司或者个人都会有一份独一无二的副本。

其实上面我还漏了说 QT， QT 在国外有专业的团队维护，文档也很好，但至少有两个缺点：1、它是跨平台的，跨平台即是优点，也是缺点，为了实现跨平台，很多时候需要做出取舍，就算抽象的 100%的完美，它也不可避免的带来体积庞大；2、代码量太大，普通人很难驾驭：就算是看懂都不容易，更别说修改了，这样的结果就是一旦在使用中遇到问题你唯一的选择就是提交 BUG 给 QT 开发小组等待补丁（要知道不存在没有 BUG 的产品）。

SOUI 是什么？

SOUI 是一套和 Duilib 类似的开源 C++ UI 开发框架。它的祖宗是金山卫士开源版本中使用的 UI 库 Bkwin，之后由启程软件（也就是我了）开发维护升级为 Duiengine，最后历经多次重构改名为 SOUI，寓意"瘦 UI"，"UI, just so so!"。使用 MIT 开源协议，公司、个人兼可免费作用，只需要发布时带上 SOUI 的 license。

SOUI 代码的获取

SVN： http://code.taobao.org/svn/soui2/trunk 不要在浏览器中打开该网址，只能使用 SVN 客户端签出。

GIT：https://github.com/setoutsoft/soui

SOUI 界面效果


SOUI演示 version:1.5
soui ctrl
教程
flash
animate
layout
treebox
misc.
skia
about
tab1
tab2
tab3
启程软件
博客园 首页 新随笔 联系 订阅 管理
置顶随笔
[置顶]SOUI开发者论坛
摘要: http://www.lumaba.cn/forum.php?mod=forumdisplay&fid=2需要的朋友可以上去交流。
阅读全文
posted @ 2015-06-05 11:01 启程软件 阅读(561) 评论(2) 编辑
[置顶]基于SOUI开发的应用展示
组织机构
市场部
南一区
北二区
西三区
第一分队
宣传部
南一区
北二区
西三区
click the right grip to show or hide the pane
组织机构
市场部
南一区
北二区
西三区
第一分队
宣传部
click the left grip to show or hide the pane

group1	
text1.1	value 1.1
text1.1.3	value 1.1.3
text1.1.1	value 1.1.1
text1.1.2	value 1.1.2
text1.1.2	value 1.1.2
text1.2	value 1.2
group2	
option2.1	false
size2.1	200,300
宽	200
高	300
text2.2	value 2.2
color2.1	#00ff00ff

SOUI 的特点

使用层：高速，稳定，美观，可配置

代码层：精心设计，模块低耦合，插件化设计，对象可靠的命令周期管理，类似 WTL 的编码方式，现代化的事件处理模型及优异的扩展能力。

代码量：核心模块代码量 4W+，编译后 DLL Release 版本在 900K 左右。得益于精心组织的代码框架，虽然代码量较 Duilib 这样的 UI 库有比较大的提高（核心框架更完善，控件更多，注释量更大），但是阅读代码还是很轻松的（大量实际用户的亲身体验）。

高速主要体现在 3 个方面：

1、框架设计扁平化，层次简单（和 QT 相比）：从宿主窗口收到消息到控件响应消息只有一个中间层。

2、简单有效的刷新策略：通过对剪裁区及刷新时机的有较控制， 能有效的提高刷新效率。

3、高效的渲染引擎：通过 将渲染引擎接口化，成功的将 skia 渲染引擎引入到 SOUI 中，Skia 是 Google 的 Chrome 的渲染引擎，Chrome 比 IE 渲染速度快，Skia 功不可没。

稳定性方面，SOUI 脱胎于 Bkwin，再经过本人的不断精心重构，已经在多个大量用户的产品中应用，包括最近开发的瑞雪医生客户端，多玩魔盒 2.0, Dota2 游戏盒子及多玩多个游戏盒子中使用， 及百度云管家的大部分界面。

百度云管家据说最初使用的是腾讯 QQ 界面库的早期版本（无从考证），然而 QQ 界面库大量使用 COM 技术，扩展非常麻烦，使用很是不便，在后续的 UI 需求中开始大量使用 SOUI 的前身 DuiEngine。

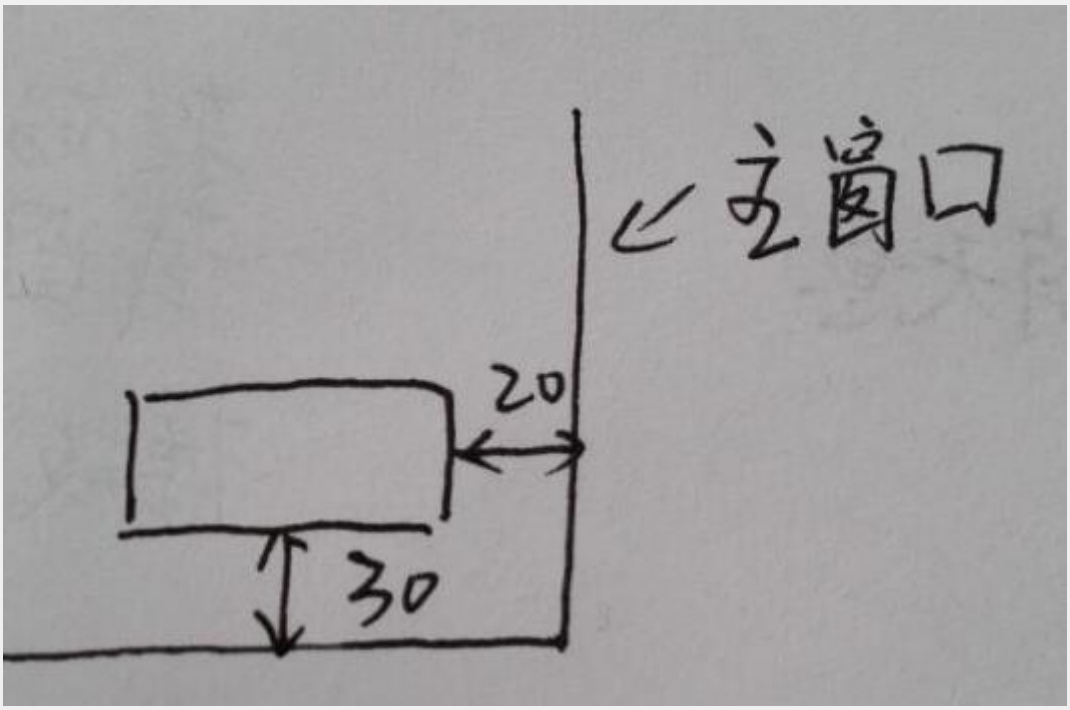
美观方面，SOUI 原生支持 Alpha 通道，能够实现各种半透明效果，包括主窗体半透明，DUI 窗口半透明，DUI 窗口模仿 LayeredWindow（分层窗口）效果等，轻松实现各种异形效果。

可配置方面，SOUI 中所有 UI 资源都采用 XML 描述，调整 UI 效果一般只需要修改 XML 资源即可完成。

说到代码层的设计很难用语言描述，只有亲自阅读代码方能理解。为大部分需要在外部（APP 层）经常引用的 UI 相关对象提供引用计数设计能够有效减少 C++开发中常见的野指针问题，这一点还是很好体会，同时系统中也重点解决了如消息分发的分层设计，窗口对象的消息重入等影响 UI 使用体验的关键性问题。

- SOUI 亮点

宽泛的说 SOUI 多好大家并没直观的感觉，下面从一些具体的点来介绍 SOUI。
界面布局
也许初学者对于 SOUI 的布局还不太适应，特别是对于那些习惯了 Duilib 的布局方式的朋友。事实上 SOUI 的布局应该是最接近程序思维的布局方式。前段时间开发 Android，仅仅是它的 5 大 Layout 就能让人崩溃，而且不同的 layout 对应的布局属性还不一样。
SOUI 的布局非常简单，只有两个布局属性：pos + offset，具体参考博客：
http://www.cnblogs.com/setoutsoft/p/3925952.html
通常使用一个 pos 属性就解决布局问题了，pos 在 XML 中使用"x1,y1,x2,y2"这样的 4 个坐标定义一个控件在父窗口中的相对位置，而 offset 则定义通过 pos 计算出来的位置后在 X，Y 两个方向需要叠加的偏移，偏移值需要乘上窗口大小。
例如下面这个需求：



只知道窗口需要靠右下角，不知道窗口大小的情况，在 SOUI 中只需要使用属性 pos="-20,-30" offset="-1,-1"即可。

渲染流程

一个 UI 中的界面元素最后会通过各级子窗口形成一个树状结构。一般的渲染流程自然是从根节点一层一层的直到渲染完成所有叶结点。这个过程很简单，可能很多 UI 库也就做到这个层次（例如 DuiLib）。但是对于一个高性能的 UI 库仅做到这个层次是不够的，举例来说：一个画笔程序需要在 OnMouseMove 里面绘制新拾取的线条，本能的做法是获取窗口画布，绘制完成后再提交画布（类似 Windows API: GetDC and ReleaseDC），而不是每一次绘制只能请求宿主刷新（请求宿主立即刷新依赖于系统对 UpdateWindow 这个 API 的响应速度）。

因此一个成熟的 UI 引擎有必要实现 GetDC 及 ReleaseDC 这样的接口。和基于 HWND 的窗口获取 HDC 不同，在一套 DirectUI 系统中实现 GetDC 及 ReleaseDC 要更加复杂：最关键的问题在于获取前绘制窗口的背景，以及提交后绘制窗口的前景，要实现窗口背景前景的分开绘制又需要系统提供绘制在指定 Z-Order 范围内的窗口的能力，当然前提是系统中有 Z-Order 这样的概念。

就算实现了窗口的背景与前景的分别绘制，对于一个高性能的 UI 引擎可能还是不够的。因为有些时候一个窗口中的内容是不需要和背景混合的，窗口刷新的时候绘制背景是没有意义的（如视频播放窗口），就是需要另一种技术：窗口的跨层渲染（不知道这样命名是不是合适）。当一个视频窗口需要刷新的时候，它的刷新流程和基本的刷新流程是不一样的，渲染时它会跳过它的所有父窗口直接到这个窗口层来，从而大大加速渲染过程。

分层窗口

Windows 的分层窗口是 Windows 2000 提供的一项重要更新。苹果系统的 UI 很漂亮，有了分层窗口，Windows 系统上开发的应用也可以同样漂亮。

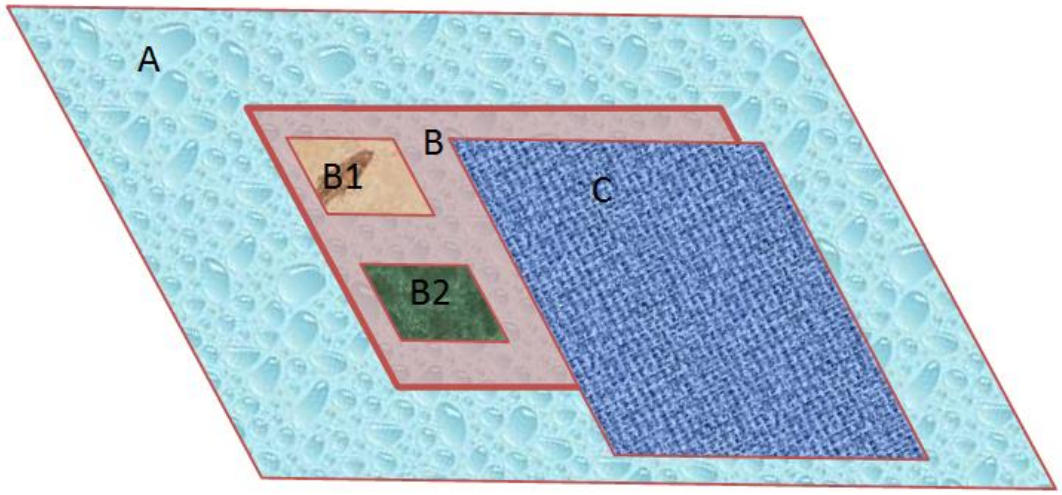
这里说的分层窗口有两个层次：一个是 DirectUI 的宿主窗口中使用分层窗口技术；另一层是在 DirectUI 的 DUI 窗口系统内部实现分层窗口技术。

使用分层窗口技术听起来比较简单，不就是设计一个 WS_EX_LAYEREDWINDOW 属性再使用 UpdateLayeredWindow(EX)更新窗口吗？！如果 SOUI 只达到这个层次，那和 codeproject 上随便找一个 demo 也没有什么区别。

首先要搞清楚，SOUI 是一套 DirectUI 系统，而不是 Demo，因此它不能停留在加载一个 32 位 PNG 图片并显示出来这样的层次上。它必须要能够让用户能够调用各种绘制图形，图像，文字的 API 来组合出一个最终需要呈现的 32 位位图。这一点要求看起来简单，在 Windows 系统上实现起来并不简单，因为 Windows 上最基本的绘制 API（GDI）都是不支持 alpha 通道的。有一个简单的选择：GDIPlus。然而 GDIPlus 有一个毛病就是速度太慢，这对于一个通用的 UI 引擎来说，全部依赖 GDIPlus 基本上就宣判了这个引擎的死刑。在 SOUI 中采用渲染引擎抽象的方法实现了两种渲染引擎：Skia + GDI。前面不是说 GDI 不支持 Alpha 通过不能用吗？没错，直接用 GDI 函数是不行的，我们需要适当的改造（具体方法参见代码）。

解决了绘制方法，要更新到窗口中显示也还是有技巧的。有人可能知道，使用 UpdateLayeredWindow 这个 API 更新的窗口将收不到 WM_PAINT 消息。由于在半透明窗口中不能直接支持有窗口句柄的子窗口的显示（如 IE 控件），SOUI 还必须为那些需要容纳窗口句柄子窗口的情况提供支持，即通过配置同时支持半透明窗口与不透明窗口。但是我不愿意为两种不同的最终位图呈现模型提供两套不同的机制。解决的办法很简单，通过为半透明类型的窗口设计一个辅助窗口，使用它来接收 WM_PAINT 消息，收到该消

息时调用 UpdateLayeredWindow 更新窗口。注： 这个技术是学习另一套 UI 库 MetalBone 实现的。
讲完了使用宿主窗口分层窗口，下面讲讲 DUI 窗口的分层窗口技术的实现。
使用分层窗口技术能够使 UI 效果更漂亮，关键技术就在这个层。层是什么？层是一组窗口的绘制容器，它将该层下所有子窗口的绘制内容绘制到一个独立的缓冲区上， 最后再一起绘制到分层窗口的上一层绘制缓冲区中。如下图：



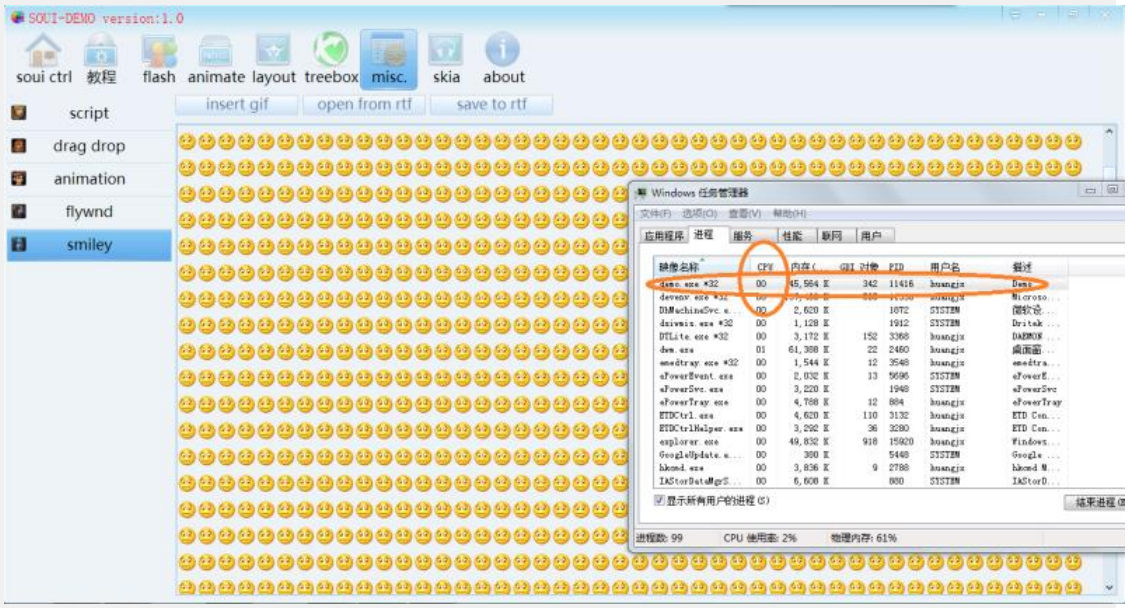
A、B、B1、B2、C 为 DUI 系统中 5 个 DUI 窗口。其中，B、B1、B2 是同一个渲染层。也就是说设计需要它们先绘制好后再和 A，C 做融合。 类似的需求对于一个漂亮的 UI 来说可能会很常见。如果在 UI 引擎中没有层的概念是不可能实现的。如果不需要实现前面提到的背景和前景分别渲染的情况，实现会层窗口其实也不难，只需要在渲染到 B 窗口时创建一个缓冲区，把从 B 开始的内容渲染到这个缓冲区，完成后再回到正常渲染流程，就像没有 B1、B2 一样。但是 SOUI 是支持背景前景分别渲染的，实现这个过程的代码逻辑就可能很复杂了（可以自己想象一下）。

非客户区
HWND 的非客户区用来绘制滚动条及边框及标题栏，菜单栏。客户区是用户绘制的常规区域，在设计上将窗口的显示区域划分为客户区和非客户区，有利于用户在重写客户区的绘制代码时不被非客户区干扰，也有利于代码的复用。
在 DuiLib 中，一个控件如 Richedit 需要显示滚动条，它需要给这个控件组合两个滚动条控件。这种方式虽然看上去没有什么大的问题，如果由于窗口中内容的变化需要动态显示隐藏滚动条时可能会很麻烦，至少它会引起窗口布局系统的重排，因为滚动条显示和隐藏时控件的客户区大小是变化的。
而在 SOUI 系统中，滚动条和 HWND 一样，用户根本不需要关心，因为内部已经自动处理好了滚动条，也不会引起布局系统的重排。

资源加载
一般来说 SOUI 中引用的所有资源都在 XML 中描述。刚入门的朋友通常反映 SOUI 中使用资源的方式不如 DuiLib 直接，很难入门。但是一旦真正理解了 SOUI 的这种资源组织方式一定会更喜欢 SOUI。

SOUI 提供 3 种资源加载方式：文件，PE 资源，ZIP 包。
首先 SOUI 的资源包必须提供一个文件索引表，对于使用 PE 资源的资源包，索引表就是资源的类型及 ID，而对于直接使用文件或者 ZIP 包的资源，索引表则是一个 XML 文件。在索引表中，定义每一个资源的 type 及 name 两个 KEY，SOUI 界面布局中只能使用 type 和 name 两个 key 来引用资源。
用户只需要准备一套文件资源，如果需要将资源编译到 PE 文件中，系统提供一个工具直接从文件资源的索引 XML 转换成 rc 编译器可以识别的 rc 文件；而如果用户需要使用 ZIP 资源包，则只需要使用一个 ZIP 工具如 rar, 7z 将资源文件夹打包即可（推荐使用 7z 打包资源，SOUI 内自带的 zlib 1.2.5 能够识别 7z 打包的带密码的 zip 包，但不能识别 rar 打包的带密码的 zip 包。

窗口动画改进
一般情况下我们推荐使用窗口定时器来创建动画。使用窗口定时器创建动画的好处是定时器和 UI 是同一个线程，而 SOUI 不支持多线程同步更新 UI（事实上一般的 DirectUI 库都不推荐在工作线程中操作 UI，如 Android）。那么问题来了，如果为每一个 DUI 窗口创建很多定时器，那么系统的消息队列中将充满定时器消息，严重时可能大大降低 UI 性能。
解决方案：在主窗口中创建一个 10ms 间隔的定时器，需要处理动画的窗口向系统注册使用该定时器，动画记录下一次动画需要等待的时间，使用该统一的定时器计数。
我们看一下面 DEMO 中显示大量动画表情时 SOUI 的效率：



这一 CPU 占用率甚至比 QQ 中同样情况下还低。

容器分层
什么叫容器分层？在 DirectUI 中所有的 DUI 窗口都必须生存在一个容器中。DUI 窗口的绘制请求等最终需要由这个容器来实现。在容器不分层的情况下，所有 DUI 窗口在容器中的物理坐标都是从(0,0)开始。这样有什么问题呢？如果要在列表控件的列表项中使用 DUI 控件就会变得非常麻烦，因为在窗口滚动时你可能不得不同时更新所有这些控件的坐标。

有了窗口层的概念就不一样了，每珍上列表项是一个新的容器，无论列表项显示在哪，列表项中的控件（容器中的控件）的坐标都不需要调整。因为有了容器分层，在 SOUI 中实现包含子控件的列表变得非常简单（参考下节：高性能列表控件）。

高性能列表控件

Windows 系统中提供的列表控件非常简单，只能满足简单的数据显示需求。注意，是显示。然而现在的 UI 需求中经常出现那种即时修改列表控件内容的情况，你将不得不花大量的时间对列表控件进行自绘，而效果只能说勉强。

通过研究 Android 系统中提供的列表控件的代码，借鉴 Android 中 ListView 的思想，SOUI 实现了一套高性能的列表控件 SListView 及 SMcListView。

SListView 及 SMcListView 都是基于虚表技术，同时只创建当前正在显示的及部分备用的列表项容器，将资源占用缩小到最少。同时 ListView 在滚动时能够高效刷新，实现了海量数据的高性能显示及更新。

实现这个高性能列表控件的关键有两点：

首先是 SOUI 中实现的容器层的概念，使得列表位位置变化时，容器内部的控件不需要调整坐标。

其次就是容器数据的充分重用。

注：上面列表中只测试了 7W 行数据，实际上 listview 中显示的数据量多少完全不影响 UI 性能，亲测 700W 行数据和 7W 行效果一样。

无窗口 Richedit

Edit 控件是 UI 中最常用的控件之一。在允许存在子窗口句柄的情况下，系统 Edit 控件已

经能够很好的满足我们的需求。然而在不允许子窗口句柄的情况下，实现一个 Edit 控件会非常麻烦。
当然，程序可以选择自己去重新实现一套 edit，Edit 也许还可行，一般情况下要实现一个 Richedit 基本不可行。 好在实现 Richedit 的模块 riched20.dll 中把 UI 和逻辑分离开来，即可以用它直接创建有窗口的 Richedit，也可以用它来创建提供无窗口 Richedit 的 ITextServices 接口。然而即使是这样，程序员需要为 ITextServices 实现一个 ITextHost 接口。尽管 MSDN 上有相关的文档及示例，但是根据它们提供的这些资料实现的效果很不理想。必竟只是 Demo，不是完整的代码，它不能演示开发中可能遇到的每一个细节。然而恰好是这些细节是影响 UI 用户体验的关键。
所以我们需要另辟蹊径来解决这个问题。我解决这些细节，关键在于理解它们的逻辑。SOUI 的办法是找到 riched20.dll 的源代码。好在网络上流传着一份从 WinCE 源代码中分离出来的 Riched20.dll 的源代码，虽然用它编译出来的 Richedit 有很多 BUG，但利用它可以让我们更好的理解各种细节。大家可以测试 SOUI 中的 Edit，效果应该是各种类似库中最好的一个之一。

XML+LUA
部分模块在 SOUI 中采用了接口化设计，如前面提到的渲染引擎，以及后面要说的多语言翻译，以及这里要说的脚本模块。
脚本语言方便灵活，更新简单，LUA 脚本还有高效的特点。和 WEB 的 HTML+JS 类似，SOUI 实现了 XML+LUA 的 UI 开发解决方案。XML 实现 UI 布局，LUA 实现逻辑控制。
实现方法：
在 XML 中使用<script>标签声明 UI 中需要脚本支持。
通过 XML 创建 UI 时自动从脚本模块为该 UI 实例化脚本对象。
采用 lua_tinker 自动导出 C++类到 LUA 脚本空间，包含控件对象，及控件事件对象。
在 LUA 脚本中处理事件响应。

多语言翻译
多语言翻译对于需要国际化的应用来说可能非常重要。SOUI 通过一个语言翻译接口来执行特定上下文的多语言翻译并且实现了一个类似 QT 语言翻译功能的基本 XML 的语言翻译模块。用户只需要按照 Demo 中的语言翻译文件的组织方式组织翻译 XML 就可以了。

String 及其它基于模板的集合对象的参数传递
由于 String 通常要同时支持 char 及 wchar_t 这两种字符类型，通常 String 在一个类库都都是以模板形式存在，比如 WTL，ATL，（MFC 太久不用，记不清了）。使用模板实现的对象有一个特点，那就是代码会编译到使用它的模块中。如此一来，如果在这些模板类中直接调用 malloc, new 等内存分配函数时会在调用模块的堆上分配内存，相应地，内存的翻译也需要在调用者模块中执行。
这有什么问题呢？最大的问题莫过于这样的对象不宜在不同的模块之间比参数进行传递（当然，const 参数是没问题的）。如果一个这样的对象在 A 模块中分配的内存到 B 模块中被翻译，结果只有崩溃。（如果所有模块采用 MD 方式动态链接 VS 的运行库是没有问题的）
很多小软件是不希望采用 MD 编译的，因为这样的话为了确保程序的正常运行，还需要带

上 VS 中相对应的运行库，尽管体积不大，但也麻烦。
SOUI 中采用了一点技巧，所有上述模板类都在一个独立的模块中实现，同时改写了这些类中的内存分配及释放代码。将它们重位向到该模块中的两个内存分配释放方法。经过这样处理后，不管这些模板类在哪一个模块中实例化，它们要在堆上申请及翻译内存时都是在这个独立模块中。通过这个简单的技术有效的解决了这些模板对象不能在不同模块之间传递参数的问题。

先进的事件处理模型
SOUI 同时支持类似 WTL 消息映射表方式的事件映射表来响应事件，也支持新式事件订阅的方式响应事件。事件映射表处理事件的优点在于能够规范化的把所有事件处理方法在代码水平集中到一起，方便代码的阅读；而事件订阅则提供了事件的动态处理能力，能够在任意时刻灵活的响应不同控件发出的事件。

结束语

除上述亮点，我相信还有很多细节的处理都体现了 SOUI 的工匠精神，相信用心的朋友一定可以在阅读及使用代码的过程中更深的体会到。SOUI 是启程软件历时 5 年心血的结晶，重复一下我以前做《启程输入之星》时说的那句话：因为努力，所以美丽！
希望能够为您能够喜欢。

7.4 一种高效的可变行高列表行定位算法

列表控件是数据显示时使用的一种常用的控件。
一般情况下列表中的行是等高的，这种情况下无论列表包含多少行，都能够在 O(1)的时间定位到指定行。
但是当显示的内容格式不一致时，使用相等的行高可能就意味着显示空间的浪费，也意味说用户需要更多的滚动操作，影响用户体验。
要实现一个支持可变行高的列表控件，首先要解决的问题就是快速定位列表行。
假定一个列表中的表项按照下面的高度排列：
1,2,3,1,2,3,1,2,3,1,2,3,4,5
可以知道总高度为：33
程序员需要解决从一个随机的[0,32]的值(x)定位到哪一行的问题。
当然最简单的办法就是从第一行开始逐行的数，直到数到的那一行正好包含 x，可以知道这个算法的时间复杂度为 O(n)。
当 n 很大时，这个算法基本上不可行。
一行一行的数显然是很浪费时间的，解决的办法就是一段一段的数。
要实现分段数，一个前提就是我们需要为这些数据提前建立好索引表。对于上面序列假定以 3 个元素一组为单位建立索引表就可以获得:(6),(6),(6),(6),(9)，如此一来，要定位一行，我们最多需要数 5+3 次就能找到一行。

对于数据量比较少的情况，可能上述分组方法就能解决问题了。

但是对于数据量更大的情况如何处理呢？方法很简单，那就是分组再分组，直到最后所有的分组数据形成一棵索引树。树上每一个结点代表该结点下所有子节点的高度和。

通过构造索引树，无论多少数据量，都可以在 O(log(n))的时间定位到任意行。

另外，对于大量数据，我们可能在初始化的时候并不知道总数有多少，而是在显示到哪一行时再通过计算获得。

对于这种情况，我们需要动态更新索引树。更新过程也很简单，当一行更新高度时，只需要找到该行所在的叶节点，更新叶节点高度，再逐级更新父节点即可，时间复杂度同样是 O(log(n))。

具体实现可以参考 SOUI 的 ListView 中用于可变行高支持的类：

SListViewItemLocatorFlex

附录、帮助手册排版<格式说明>

1、一级标题

大纲级别：一级

字体：微软雅黑　五号 粗体

行距：固定 20 磅

2、二级标题

大纲级别：二级

字体：微软雅黑　五号 粗体

行距：固定 20 磅

3、三级标题

大纲级别：三级

字体：微软雅黑　五号 粗体

行距：固定 20 磅

4、正文

大纲级别：正文文本

字体：微软雅黑　五号

行距：固定 20 磅

5、word 插入语法高亮代码

把 xml、C++代码复制到 UltraEdit 中，并使用特殊复制功能<复制为 HTML>，已自定义快捷键：Ctrl+Shift+C。

6、设置代码 背景

设置 UltraEdit→视图→设置颜色→纯文本→背景颜色

7、

SOUI 更新到 2.0

更新：

1、修改 uiresbuilder，增加资源 ID 自动生成功能。包括自动提取所有布局中控件的 name，自动生成 ID，自动提取字符串表，颜色表。具体使用方式参见下一篇。

2、修改布局中引用字符串的方式。原来使用%str-name%这种方式来引用在字符串表中定义的字符串，修改为使用和 android 一样的方式：@string/str-name，不支持嵌套。

3、增加颜色表，可以在布局 XML 中使用@color/color-name 这种方式使用颜色表中定义的颜色值。

4、给 EventArg 增加一个 bubbleUp 属性，默认为 true，应用中处理一个事件后可以将 bubbleUp 设置成 false，此时之前订阅的事件处理方法将不再调用（中断事件的冒泡过程）。

5、增加一个 STreeView 控件，STreeView 中一个采用 MVC 模式设计的高性能 TreeCtrl，目前该控件还在测试阶段（欢迎使用，欢迎补充功能）。

6、在 SOUI 中增加 soui-version.h，版本号移动到该文件中定义。

博客收录情况：

BuildFilePath 及打开文件对话框

第一篇：SOUI 是什么？【1.1、1.2、1.3】

第二篇：SOUI 源码的获取及编译【2.1、2.2】

第三篇：用 SOUI 能做什么？【部分含手工创建 SOUI 项目】

第四篇：SOUI 资源文件组织【4.1.1】

第五篇：在 SOUI 中使用 XML 布局属性指引(pos, offset, pos2type) 【4.1.2】

第六篇：在 SOUI 中用九宫格拉伸方式显示一个图片资源【4.4】

第七篇：创建一个 SOUI 的 Hello World【3.1、3.2】

第八篇：SOUI 中控件事件的响应【4.8.1、4.8.2】

第九篇：在 SOUI 中使用多语言翻译【4.9】

第十篇：扩展 SOUI 的控件及绘图对象(ISkinObj) 【4.10.2、4.10.3】

第十一篇：SOUI 系统资源管理【4.2、4.3】

第十二篇：SOUI 的 utilities 模块为什么要用 DLL 编译？【6.1】

第十三篇：在 SOUI 中使用有窗口句柄的子窗口【4.10】

第十四篇：在 SOUI 中使用定时器【4.11】

为 GDI 函数增加透明度处理【】

第十五篇：在 SOUI 中消息通讯【4.12】

第十六篇：SWindow 的布局属性 pos2type 及 offset【4.1.3】

第十七篇：使用窗口的 cache 属性加速 SOUI 的渲染【4.13】

第十八篇：在 SOUI 中实现 PreTranslateMessage【4.14】

基于 SOUI 开发的应用展示【】

第十九篇：提高 SOUI 应用程序渲染性能的三种武器【4.15】

第二十篇：在 SOUI 中使用分层窗口【4.16】

实现一种快速查找 Richedit 中可见区域内 OLE 对象的方法【】

一种快速刷新 richedit 中内嵌动画的方法的实现【】

使用 SOUI 开发的界面集锦【】

第二十一篇：SOUI 中的控件注册机制【4.17】

第二十二篇：在 SOUI 中使用代码向窗口中插入子窗口【4.18】

第二十三篇：在 SOUI 中使用 LUA 脚本开发界面【4.19】

第二十四篇：导出 SOUI 对象到 LUA 脚本【4.20】

第二十五篇：在 SOUI 中做事件分发处理【4.21】

SOUI 开发者论坛【】

为什么在 soui 中加载 JPG 文件失败【6.2】

第二十六篇：两个 SOUI 新控件 ---- SListView 和 SComboView（借用 Andorid 的设计）【4.24】

第二十七篇：SOUI 中控件属性查询方法【4.25】

一种高效的可变行高列表行定位算法【7.4】

第二十八篇：SOUI 中自定义控件开发过程【4.10.1】

第二十九篇：使用 SOUI 的 SMCListView 控件【】

UI 神器-SOUI【7.3】

第三十篇：SOUI 模块结构图及 SOUI 框架图【1.4】

在 SOUI 中非半透明窗口如何实现圆角窗口？【7.2】

SOUI 与 WTL【7.1】

第三十一篇：SOUI 布局之相对于特定兄弟窗口【4.1.3】

SOUI 更新到 2.0【】

第三十二篇：在 SOUI2.0 中像 android 一样使用资源【4.23】

不注册 COM 在 Richedit 中使 OLE 支持复制粘贴【】

SOUI 中做的一个磁力吸附效果【】

第三十三篇：使用 uiresImporter 生成 uires.idx 及 skin.xml【3.3】

第三十四篇：在 SOUI 中使用异步通知【4.22】

搜索引擎广告过滤 Chrome 插件【】

SOUI Editor 使用教程【3.3】

SOUI taobao SVN 目录结构说明【2.1】

在 SOUI 中使用线性布局【4.1.4】

END