DuiVision

蓝蚂蚁工作室

http://www.blueantstudio.net

[DUIVISION 控件开发指南]

DuiVision 界面库文档[更新日期：2016-03-17]

目录

DuiVision 控件开发指南

1. 概述

DuiVision 是参考了仿 PC 管家程序、金山界面库、DuiEngine、DuiLib 等多个基于 DirectUI 的界面库开发的。

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。

DirectUI 技术需要解决的主要问题如下：

1、窗口的子类化，截获窗口的消息。

2、封装自己的控件，并将自己的控件绘制到该窗口上。

3、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。

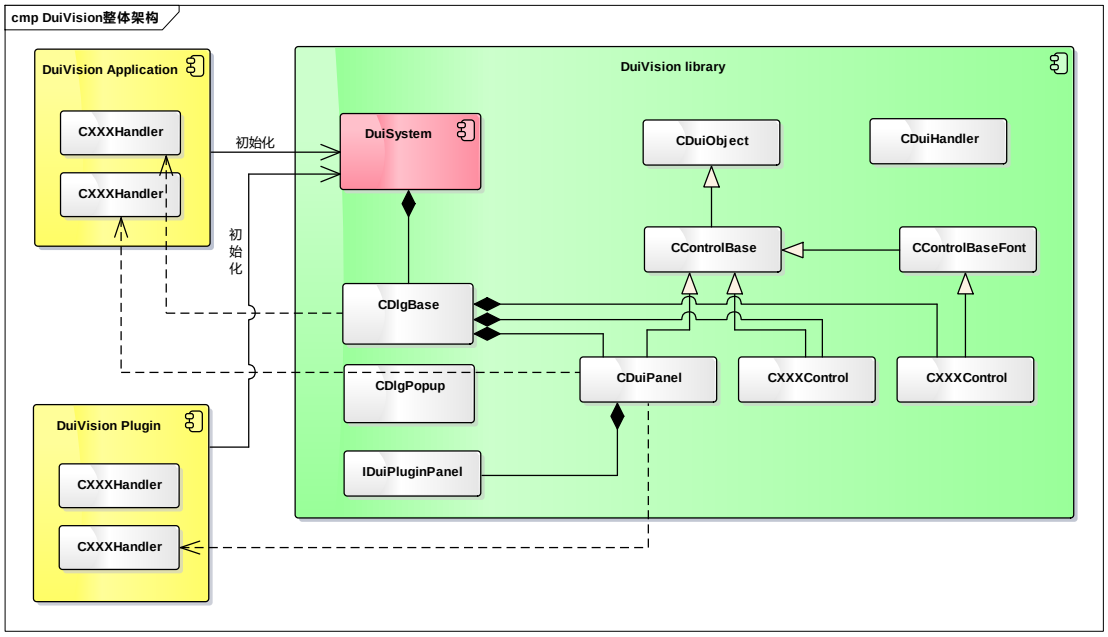
4、根据不同的行为发送自定义消息给窗口，以便程序进行调用。

5、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。

2. DuiVision 代码结构

DuiVision 代码的整体架构如下图所示：



代码可以分为 DuiVision 库、DuiVision 应用程序、DuiVision 插件几部分，其中 DuiVision 应用程序和 DuiVision 插件都是基于 DuiVision 库开发的。

DuiVision 库的类结构中几个重要的类说明如下：

CDuiObject：

是所有 DuiVision 对象的基类，每个对象都包含有 ID、名字、区域这些 DUI 对象基础的东西，同时也可以给对象设置一个事件处理对象（CDuiHandler），DUI 基类还封装了一些虚函数，包括对象加载（每个对象都可以对应到 xml 描述文件中的一个 node 节点，DUI 基类封装了对 xml 节点的加载方法，包括读取 xml 节点名、节点属性，并对基础的属性进行处理）、消息处理、设置对象更新标识等。

CControlBase：

是所有 DUI 控件的基类，定义了所有控件都具有的属性，以及公共的方法、虚函数等，CControlBase 定义的虚函数用于派生的控件实现特定的功能，包括鼠标、键盘操作、控件的画图函数、控件大小和位置变更函数等。

CControlBaseFont：

所有需要显示文字的控件都可以从此控件派生，这个控件定义了一些文字的公共属性和方法，包括文字标题、字体、对齐方式等。

CDlgBase：

对话框类，实现了对话框的所有功能，可以支持模态和非模态对话框，对话框中会包含若干子控件，对话框可以定义对应的 xml 描述文件，通过 xml 文件可以加载下层的所有子控件，所有的事件处理也都是从对话框发起，然后逐层递归调用到子控件的处理函数。

CDlgPopup：

弹出框的基类，菜单、组合框的下拉框等都是基于这个类派生出来的，这个类和 CDlgBase 的功能类似，但对话框的一些功能在弹出框中没有实现。

DuiSystem：

是 DuiVision 的所有资源的入口，承担了所有资源的统一管理功能（图片、配置、字符串、对话框、事件处理对象都可以作为资源）,同时作为工具类提供了很多公共的函数 ,DuiSystem 是一个单例对象，一个应用程序或插件中只有一个 DuiSystem 的实例对象。

3. 自定义控件的开发

3.1. 控件概述

所有控件都是从 CControlBase 或 ControlBase 的某个派生类派生的，对于一个控件，主要要完成的事情包括：初始化（属性的解析和处理）、位置数据的处理（位置和大小变更）、画图、事件的处理（鼠标事件、键盘事件等）；

初次之外，对于复杂一些的控件，可能还要考虑控件焦点、tooltip、动画、动作响应等事情。下面会依次对控件的这些功能如何开发进行说明。

3.2. 构造函数和析构函数

控件的初始化在控件的构造函数中，控件的构造函数有两个，分别是：

```
CControlBase(HWND hWnd, CDuiObject* pDuiObject);
CControlBase(HWND hWnd, CDuiObject* pDuiObject, UINT uControlID, CRect rc, BOOL IsVisible, BOOL bIsDisable, BOOL bRresponse);
```

第二种构造函数目前没有用到，因此自定义控件只要实现第一种参数的构造函数就可以了，构造函数一般只用于初始化自定义控件类中增加的那些成员变量。

控件的析构函数主要用于释放控件中申请的内存、图片等资源，特别需要注意的是，如果控件中有用到一些图片资源，在析构函数中一定要释放，释放的方法都比较类似，例如 Button 控件中定义了 Button 的图片，在析构函数中就用如下的方法进行图片的释放：

```
CDuiButton::~CDuiButton(void)
{
    if(m_pImageBtn != NULL)
    {
        delete m_pImageBtn;
        m_pImageBtn = NULL;
    }
}
```

3.3. 图片的定义

很多控件中都需要定义图片资源，例如 Button 控件中定义的 Button 的图片，就是由下面这种四种状态的图片组成的，DuiVision 中很多控件的图片都需要包含多个状态的图片：

在 ControlBase.h 中有按钮类图片的状态定义：

```
enum enumButtonState
{
	enBSNormal = 0,
	enBSHover,
	enBSDown,
	enBSDisable,
	enBSHoverDown,
	enBSDisableDown
};
```

完整的状态有六种，分别是正常状态、鼠标移动到控件上的状态、鼠标在控件上按下的状态、禁用状态、按下时鼠标移动到控件上的状态、按下时控件被禁用的状态，一般只会用到前四种状态，后面两种状态是检查框、广播按钮这样的控件才有的。

DuiVision 中的控件用到的这种多状态图片一般是按照每种状态的图片横向排列在一个大图中，顺序就是按照上面定义的顺序，当然对于自定义控件也可以按照其他的方式制作大图，只要画图时候按照定义的方式进行解析就可以了。

除了按钮的图片，下面列出了一些常见的多状态图片。

检查框的六状态图片：

广播按钮的六状态图片：

Edit 控件的外框的四状态图片：

Tabctrl 控件的当前页签热点图片，只有两种状态（鼠标不在页签上和在页签上）：

动画图片，横向的多张图片，实际的小图片数可能很多：

在自定义控件中定义图片资源，可以使用 DuiVision 提供的几个宏来减少代码量，举个例子，例如定义上面按钮的图片资源，需要写的代码如下：

1、在头文件的类定义中使用 DUI_IMAGE_ATTRIBUTE_DEFINE 宏来定义一个图片，这个宏的参数是图片资源的变量名中的一部分，例如 Btn 表示按钮的图片，这个宏一般定义在控件的属性定义部分的上面：

```
DUI_IMAGE_ATTRIBUTE_DEFINE(Btn);          // 定义按钮图片
DUI_DECLARE_ATTRIBUTES_BEGIN()
    DUI_COLOR_ATTRIBUTE(_T("crtext"), m_clrText, FALSE)
    DUI_INT_ATTRIBUTE(_T("animate"), m_bTimer, TRUE)
    DUI_INT_ATTRIBUTE(_T("maxindex"), m_nMaxIndex, TRUE)
    DUI_CUSTOM_ATTRIBUTE(_T("img-btn"), OnAttributeImageBtn)
    DUI_BOOL_ATTRIBUTE(_T("showfocus"), m_bShowFocus, FALSE)
DUI_DECLARE_ATTRIBUTES_END()
```

实际通过这个宏会定义两个类成员变量，分别是：

```
Image* m_pImageBtn;
CSize m_szieBtn;
```

这两个变量分别是图片对象指针和图片的大小，同时这个宏会定义几个针对这个图片的初始化和通过 xml 属性加载用到的函数，使用这个宏之后我们可以不用关心复杂的图片定义、加载的细节。

2、定义图片的属性，用于 xml 文件中定义图片时候的属性名字，例如针对这个 Btn 图片的属性定义：

```
DUI_CUSTOM_ATTRIBUTE(_T("img-btn"), OnAttributeImageBtn)
```

这里定义的 img-btn 就是在 xml 中对应的按钮的图片属性名，OnAttributeImageBtn 是对应的图片属性加载用到的函数，在上面的图片定义宏中已经隐含的定义了这个函数，所以这里只要按照命名规则写就可以了，命名规则就是 OnAttributeImage+前面定义的图片变量名一部分。

3、析构函数中对图片资源的释放

按照上一节所说的析构函数写法就可以，注意其中的图片资源的变量名就是按照“m_pImage+前面定义的图片变量名一部分”这样的规则。

4、在控件的 cpp 实现文件的析构函数下面增加下面这段代码

```
// 图片属性的实现
DUI_IMAGE_ATTRIBUTE_IMPLEMENT(CDuiButton, Btn, 4)
```

通过上面这个宏可以实现在头文件中定义的图片的初始化、xml 属性加载相关的几个函数，这个宏的第一个参数是控件的类型，第二个参数和头文件中定义的名字一部分相同，第三个参数表示这个图片横向分割为几个小图片。

5、图片的使用

主要是在画图函数中使用，参加控件的画图的章节。

3.4. 位置数据处理

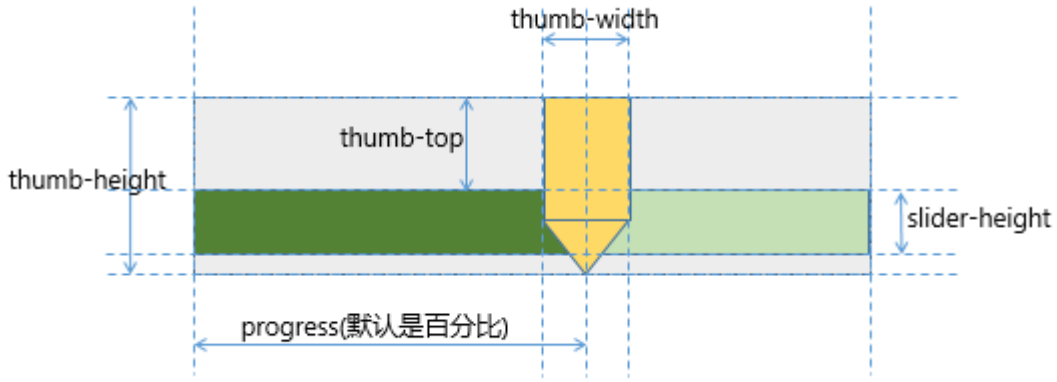
控件的位置数据是根据父控件的位置变化可能需要进行调整的，DuiVision 已经对位置数据的计算进行了封装，包括支持相对位置，当一个控件的位置发生变化时候，控件内的一些位置信息可能需要相应的进行调整，控件内的位置调整就需要控件重载 SetControlRect 虚函数进行调整了。下面看一下滑动条控件的 SetControlRect 的实现内容：

```
void CDuiSlider::SetControlRect(CRect rc)
{
	m_rc = rc;
	if(m_nSliderHeight == 0)
	{
		m_nSliderHeight = m_rc.Height() - m_nThumbTop;
	}
	if(m_nThumbWidth == 0)
	{
		m_nThumbWidth = m_sizeThumb.cx;
	}
	if(m_nThumbHeight == 0)
	{
		m_nThumbHeight = m_sizeThumb.cy;
	}

	// 计算滑块的位置
	int nPos = (int)__max(m_rc.Width() * m_nProgress / m_nMaxProgress - m_nThumbWidth / 2,
```

```
0);
    nPos = (int)__min(nPos, m_rc.Width() - m_nThumbWidth);
    m_rcThumb = CRect(nPos, 0, nPos + m_nThumbWidth, m_nThumbHeight);
    m_rcThumb.OffsetRect(m_rc.left, m_rc.top);
}
```

滑动条控件中有一个滑块，如下图所示，当控件位置变化时候，滑块的位置也需要相应的调整，因为滑块的位置是按照 progress 的百分比计算出来的，因此 SetControlRect 函数中就需要根据当前的 progress 百分比，计算出滑块位置之后保存在 m_rcThumb 成员变量中，这个位置信息会在画图、鼠标操作等函数中使用到：



对于自定义控件，如果控件中没有类似需要计算的相对位置，则不需要实现这个函数，如果有的话，就需要重载这个函数进行内部的一些位置数据的计算。

3.5. 画图

控件要在界面显示出来，就必须要实现 DrawControl 虚函数，下面是检查框控件的画图函数，增加了每一步的说明：

```
void CCheckButton::DrawControl(CDC &dc, CRect rcUpdate)
{
    int nWidth = m_rc.Width();
    int nHeight = m_rc.Height();

    if(!m_bUpdate)      // 是否需要重画整个控件
    {
        // 申请内存dc
        UpdateMemDC(dc, nWidth * 6, nHeight);

        // 用内存dc初始化GDI+的graph对象
        Graphics graphics(m_memDC);
        CRect   rcTemp(0, 0, nWidth, nHeight);
```

```
            // 画种状态的图片
            for(int i = 0; i < 6; i++)
            {
                // 将背景信息先复制到内存dc对应状态的位置
                m_memDC.BitBlt(i * nWidth, 0, nWidth, nHeight, &dc, m_rc.left ,m_rc.top,
SRCCOPY);

                // 在内存dc对应状态的位置画检查框对应状态的图片
                graphics.DrawImage(m_pImage, Rect(rcTemp.left, rcTemp.top + (nHeight -
m_sizeImage.cy) / 2,     m_sizeImage.cx, m_sizeImage.cy),
                        i * m_sizeImage.cx, 0, m_sizeImage.cx, m_sizeImage.cy, UnitPixel);

                // 计算下一个状态的内存dc位置
                rcTemp.OffsetRect(nWidth, 0);
            }

            // 如果设置了检查框文字，则画文字到内存dc
            if(!m_strTitle.IsEmpty())
            {
                m_memDC.SetBkMode(TRANSPARENT);

                rcTemp.SetRect(0, 0, nWidth, nHeight);

                // 设置字体
                BSTR bsFont = m_strFont.AllocSysString();
                FontFamily fontFamily(bsFont);
                Font font(&fontFamily, (REAL)m_nFontWidth, m_fontStyle, UnitPixel);
                graphics.SetTextRenderingHint( TextRenderingHintClearTypeGridFit );
                ::SysFreeString(bsFont);

                // 设置对齐方式等参数
                StringFormat strFormat;
                strFormat.SetAlignment(StringAlignmentNear);
                strFormat.SetFormatFlags( StringFormatFlagsNoWrap |
StringFormatFlagsMeasureTrailingSpaces);
                // 计算文字的尺寸和画图的位置
                Size size = GetTextBounds(font, strFormat, m_strTitle);
                CPoint point = GetOriginPoint(nWidth - m_sizeImage.cx - 3, nHeight, size.Width,
size.Height,
                                        GetGDIAlignment(m_uAlignment),
GetGDIVAlignment(m_uVAlignment));
```

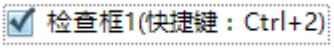
```
                // 每一种状态的内存dc中叠加上文字的内容
                for(int i = 0; i < 6; i++)
                {
                    SolidBrush solidBrush(enBSDisable == i ? Color(128, 128, 128) : m_clrText);

                    RectF rect((Gdiplus::REAL)(m_sizeImage.cx + 3 + point.x + i * nWidth),
(Gdiplus::REAL)point.y, (Gdiplus::REAL)(nWidth - m_sizeImage.cx - 3 - point.x),
(Gdiplus::REAL)size.Height);
                    BSTR bsTitle = m_strTitle.AllocSysString();
                    graphics.DrawString(bsTitle, (INT)wcslen(bsTitle), &font, rect, &strFormat,
&solidBrush);
                    ::SysFreeString(bsTitle);

                    // 画焦点框(虚线框)
                    if(m_bIsFocus && m_bShowFocus)
                    {
                        Pen pen(Color(128, 128, 128), 1);
                        pen.SetDashStyle(DashStyleDot);
                        RectF rectFocus((Gdiplus::REAL)(point.x + i * nWidth),
(Gdiplus::REAL)point.y, (Gdiplus::REAL)(m_sizeImage.cx + 6 + size.Width),
(Gdiplus::REAL)size.Height);
                        graphics.DrawRectangle(&pen, rectFocus);
                    }
                }
            }
        }

        // 将保存的内存dc内容中对应当前状态的部分复制到实际界面显示的dc
        dc.BitBlt(m_rc.left,m_rc.top, m_rc.Width(), m_rc.Height(), &m_memDC, m_enButtonState *
nWidth, 0, SRCCOPY);
}
```

检查框控件的实际显示效果如下：



DrawControl 函数的实现思路：

1、传入的参数 dc 和 rcUpdate 分别表示实际画图的 dc 和刷新的区域，在 DrawControl 中第一次需要执行一次完整的画图的流程，将显示内容存储到一个内存 dc 中，以后如果这个控件的界面显示没有变化的话，DrawControl 只需要将保存的内存 dc 的内容复制到显示的 dc

就可以了，这样可以加快显示的速度，如果所有控件都没有变化，实际上整个界面的显示就是把所有控件的内存 dc 内容一层一层复制到显示 dc，先后一次性显示出来，复制时候是按照控件添加的顺序进行复制的。DrawControl 中用于控制是否重新画图还是仅复制内存 dc 的开关是 m_bUpdate 变量，如果为 true 表示仅复制内存 dc，false 表示要重画控件。

2、如果是重画整个控件，则需要使用 UpdateMemDC 函数申请内存 dc，每个控件类中都有保存一个内存 dc 成员变量 ,UpdateMemDC 函数中会判断如果申请的宽度或高度和以前保存的不一样，就会先释放掉以前保存的内存 dc，然后在按照新的尺寸申请一个新的，如果和以前一样大小就使用以前的。注意申请内存 dc 时候一般并不是只申请和控件大小相同的内存 dc，而是按照控件有多少种状态，就申请几倍大小的内存 dc，例如检查框控件有六种状态，一般的按钮有四种状态，如果是简单显示一个文字或图形，没有多种状态的，则申请和控件大小相同的内存 dc 就可以了。对于多个状态的内存 dc 申请，可以按照每种状态的图片在内存 dc 中横向排列或纵向排列，或者复杂的控件可能会包含了多个子图形的多个状态混合排列在内存 dc 中 ,上面检查框的代码中是按照横向排列的 ,因此申请时候是 width 乘以 6。

3、DuiVision 大部分的控件都使用的 GDI+进行画图，因此会使用内存 dc 创建一个 GDI+的图形对象，Graphics graphics(m_memDC);然后就是具体的控件的画图操作。

4、对于检查框的具体画图操作 ,是先画 6 种状态下的检查框图片到内存 dc 对应图片的位置，然后再画文字到每种状态对应的位置，这里检查框图片对应每种状态是不同的，文字在每种状态下都是相同的，但也需要重复画到每种状态的位置。

5、对于检查框，还需要考虑控件处于焦点状态下时候要画控件周围的虚线框 m_bIsFocus 成员变量表示当前控件是否处于焦点状态，m_bShowFocus 表示这个控件是否要显示焦点框，这是由 showfocus 属性决定的，只有这两个变量都为 true 时候才需要画焦点框。

6、DrawControl 最后将内存 dc 复制到显示 dc 时候，需要注意的是根据当前的状态，决定把内存 dc 中对应状态部分的内容复制到显示 dc，检查框的内存 dc 是按照状态横向排列的，所以使用 m_enButtonState * nWidth 定位到横向的具体内存 dc 位置。

3.6. 动画处理

有一些控件需要支持动画效果，目前控件中实现的动画效果如下：

1）按钮控件如果启用了动画，则鼠标移动到按钮时候会使用动画显示渐变效果

2）tabctrl 控件切换 tab 页时候的动画显示

第一种渐变效果的实现原理如下：

1）按钮控件如果属性中启用了动画，则 m_bTimer 为 true，同时要设置动画的最大帧数 m_nMaxIndex，在 DrawControl 函数中分配内存 dc 时候会按照按钮默认的四种状态再加上 m_nMaxIndex 种状态，所有状态的图片都会事先计算好保存在内存 dc 中；

2）渐变效果对应的这些图片帧使用了 GDI+的 ColorMatrix 颜色矩阵技术，按照索引调整图片的亮度作为中间状态的动画图片，具体可以参考 CDuiButton 的 DrawControl 函数；

3）需要重载控件的虚函数 OnControlTimer 函数，这个函数用于控件的动画定时器，固定的每 30 毫秒会被调用一次，每次被调用时候会更新一个当前帧的索引变量，表示当前需要显示渐变状态中的第几幅图片，具体代码如下：

```
BOOL CDuiButton::OnControlTimer()
{
	if(!m_bRunTime)
	{
		return FALSE;
	}

	if(enBSNormal == m_enButtonState)
	{
		m_nIndex--;
		if(m_nIndex < 0)
		{
			m_nIndex = 0;
		}
	}
	else if(enBSHover == m_enButtonState)
	{
		m_nIndex++;
		if(m_nIndex > m_nMaxIndex)
		{
			m_nIndex = m_nMaxIndex;
		}
	}
	if(0 == m_nIndex || m_nIndex == m_nMaxIndex)
	{
		m_bRunTime = false;
	}

	UpdateControl();

	return true;
}
```

第二种界面切换时候的动画原理如下（tabctrl 控件的页签切换动画）：

1）使用到了控件的 DrawSubControl 虚函数，DrawSubControl 函数用于画子控件，在 DrawSubControl 函数中会判断当前是否启用了动画，如果启用了，则才会进行动画的显示，对于每个控件，默认的画图顺序是先调用 DrawControl，然后再调用 DrawSubControl；

2）tabctrl 控件切换时候也会有动画帧数的概念，根据帧数可以计算出每一帧动画需要在横向或纵向修改多少分隔位置，当前帧同样由重载的 OnControlTimer 函数中来变更，在 DrawSubControl 函数中则根据当前帧计算出之前 tab 页面和新的 tab 页面之间要如何分隔，然后将两个页面都进行画图之后选取两个 tab 图片的内容拼接之后输出到内存 dc 中用于显示。

3.7. 事件处理

控件的事件处理最常用的是鼠标和键盘事件的处理，相关的几个需要实现的虚函数如下：

```
virtual BOOL OnControlMouseMove(UINT nFlags, CPoint point);
virtual BOOL OnControlLButtonDown(UINT nFlags, CPoint point);
virtual BOOL OnControlLButtonUp(UINT nFlags, CPoint point);
virtual BOOL OnControlKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
```

检查框的鼠标移动事件处理函数如下：

```
BOOL CCheckButton::OnControlMouseMove(UINT nFlags, CPoint point)
{
    enumButtonState buttonState = m_enButtonState;
    if (!m_bIsDisable && !m_bMouseDown)
    {
        if(m_rc.PtInRect(point))
        {
            if(m_bDown)
            {
                m_enButtonState = enBSHoverDown;
            }
            else
            {
                m_enButtonState = enBSHover;
            }
        }
        else
        {
            if(m_bDown)
            {
                m_enButtonState = enBSDown;
            }
```

```
                else
                {
                    m_enButtonState = enBSNormal;
                }
            }
        }

        if(buttonState != m_enButtonState)
        {
            UpdateControl();
            return true;
        }
        return false;
}
```

主要实现思路就是判断鼠标位置是否在控件的范围内，从而决定控件当前的状态应该更改为什么，并且要判断鼠标按下和非按下情况下，对应的状态是不一样的。

根据这些判断可以计算出新的控件状态，在函数进入的时候会保存之前的状态，最后需要比较一下新旧状态是否一致，不一致情况下就需要重新刷新一下控件的界面，刷新方法是调用 UpdateControl 函数。

鼠标按下和放开的事件处理和上面的函数是类似的，另外像按钮或检查框等控件，在鼠标放开的时候需要发送一个点击事件（鼠标点击就是鼠标在一个控件上按下又放开，一般都是在放开时候触发一个点击事件），检查框的鼠标放开事件代码如下，可以看到通过调用 SendMessage 函数可以发送一个点击事件：

```
BOOL CCheckButton::OnControlLButtonUp(UINT nFlags, CPoint point)
{
    enumButtonState buttonState = m_enButtonState;
    if (!m_bIsDisable)
    {
        if(m_rc.PtInRect(point))
        {
            if(m_bMouseDown)
            {
                m_bDown = !m_bDown;
                SendMessage(MSG_BUTTON_UP, 0, 0);
            }
            if(m_bDown)
            {
                m_enButtonState = enBSHoverDown;
            }
```

```
                else
                {
                    m_enButtonState = enBSHover;
                }
            }
            else
            {
                if(m_bDown)
                {
                    m_enButtonState = enBSDown;
                }
                else
                {
                    m_enButtonState = enBSNormal;
                }
            }
        }
        m_bMouseDown = false;

        if(buttonState != m_enButtonState)
        {
            UpdateControl();
            return true;
        }
        return false;
}
```

4. 控件的使用

按照以上控件的开发方法开发了一个控件之后，如果将控件应用在 DuiVision 应用程序中，有两种方法，可以将控件添加到 DuiVision 库代码中或者仅作为自定义控件，添加到自己的应用程序中。

如果你开发的控件是一个很多人都会用到的控件，建议添加到 DuiVision 库中，可以提交到 github 项目中，如果觉得合适会合入主线版本，这样其他人也都可以使用。

如果仅是某个项目中使用的特殊控件，建议按照自定义控件的方式添加到自己的应用程序工程中。

4.1. 控件添加到 DuiVision 库中的方法

1）首先把控件的头文件和 cpp 分别放在 DuiVision 库的 include 和 source 目录下，然后添加到 DuiVision 工程中；

2）在 DuiVision.h 中添加对控件头文件的引用；

3）在 DuiSystem 的 LoadDuiControls 函数中添加如下的控件注册代码：

REGISTER_DUICONTROL(CDuiUserControl, NULL);

完成以上几步就可以使用新的控件了。

4.2. 控件添加到自己的应用程序工程中的方法

自定义的控件代码开发完成后，将代码添加到用户自己的代码工程中，然后在控件使用之前注册到 DuiVision 库中就可以，建议注册代码放在主程序的 DuiSystem 初始化之后，下面的代码演示了在主程序中注册类名为 CDuiUserControl 的自定义控件的方法，蓝色部分代码就是用于注册自定义控件的代码：

```
BOOL CDuiVision1App::InitInstance()
{
	CWinApp::InitInstance();

	AfxEnableControlContainer();

	// TODO: 应适当修改该字符串，
	// 例如修改为公司或组织名
	SetRegistryKey(_T("DuiVision1"));

	// 初始化DuiVision界面库,可以指定语言,dwLangID为表示自动判断当前语言
	// 11160是应用程序ID,每个DUI应用程序应该使用不同的ID,ID主要用于进程间通信传递命令行时候区分应用
	DWORD dwLangID = 0;
	new DuiSystem(m_hInstance, dwLangID, _T("DuiVision1.ui"), 11160, IDD_DUIVISIONAPP_DIALOG, _T(""));

	// 注册用户自定义控件控件
	REGISTER_DUICONTROL(CDuiUserControl, NULL);

	// 创建主窗口
	CDlgBase* pMainDlg = DuiSystem::CreateDuiDialog(_T("dlg_main"), NULL, _T(""), TRUE);
	// 给主窗口注册事件处理对象
	CDuiHandlerMain* pHandler = new CDuiHandlerMain();
	pHandler->SetDialog(pMainDlg);
```

```
    DuiSystem::RegisterHandler(pMainDlg, pHandler);

    // 初始化提示信息窗口
    DuiSystem::Instance()->CreateNotifyMsgBox(_T("dlg_notifymsg"));

    // 按照非模式对话框创建主窗口,可以默认隐藏
    pMainDlg->Create(pMainDlg->GetIDTemplate(), NULL);
    INT_PTR nResponse = pMainDlg->RunModalLoop();

    // 释放DuiVision界面库的资源
    DuiSystem::Release();

    // 由于对话框已关闭，所以将返回FALSE 以便退出应用程序，
    //  而不是启动应用程序的消息泵。
    return FALSE;
}
```