WPF Modern UI笔记

Contents

[一. 引用 4](#_Toc387396237)

[二. 初始化设置 4](#_Toc387396238)

[三. 操作数据库ISession 5](#_Toc387396239)

[a. 创建Session 5](#_Toc387396240)

[b. CRUD 5](#_Toc387396241)

[c. 多个Session操作CRUD时的行为 6](#_Toc387396242)

[d. Transaction 7](#_Toc387396243)

[e. 将模型更新到数据库——Flush与Commit 7](#_Toc387396244)

[f. 根据主键获取对象（复合主键）——Session.Get<T> 7](#_Toc387396245)

[g. 更新对象时，一个Session中的不同对象拥有相同的OID——Merge 8](#_Toc387396246)

[h. 属性改变时才更新——<class> select-before-update=true 8](#_Toc387396247)

[i. Session管理，清空缓存——clear() 9](#_Toc387396248)

[j. 批量操作——少量进缓存，处理完清缓存 9](#_Toc387396249)

[k. 批量操作——StatelessSession不用缓存，立即执行 10](#_Toc387396250)

[l. 批量操作——最好的办法是直接操作数据库 10](#_Toc387396251)

[m. 数据获取策略——<Set> 延迟加载与左外链接 10](#_Toc387396252)

[n. 使用SQL检索 11](#_Toc387396253)

[o. Rollback回滚对象状态 11](#_Toc387396254)

[四. Mapping 12](#_Toc387396255)

[a. 创建PersistentClass—hbm.xml 12](#_Toc387396256)

[b. 添加Mapping到Configuration 13](#_Toc387396257)

[c. 映射文件节点顺序 13](#_Toc387396258)

[d. 映射ID 13](#_Toc387396259)

[e. 版本控制—— version与TimeStamp 13](#_Toc387396260)

[f. 自增长属性 13](#_Toc387396261)

[g. 将复杂类型的Property映射到列——<component> 14](#_Toc387396262)

[h. 一张表存继承树，扩展基类的表来存储子类——<subclass> 14](#_Toc387396263)

[i. 每个子类一张表并复制基类的表结构（抽象类）——<union-subclass> 15](#_Toc387396264)

[j. 每个子类一张表共同使用基类的表——<joined-subclass> 16](#_Toc387396265)

[k. Insert与Update属性 17](#_Toc387396266)

[l. 计算出来的派生属性——<formula> 17](#_Toc387396267)

[m. 自动更新对象属性—— cascade = “save-update” 17](#_Toc387396268)

[n. 双向关系，避免重复更新—— <set> inverse=”true” 18](#_Toc387396269)

[o. 子类用值表示具体类型—— <discriminator> 18](#_Toc387396270)

[五. 原理 19](#_Toc387396271)

[a. SQL语句 19](#_Toc387396272)

[b. 标识对象——OID 19](#_Toc387396273)

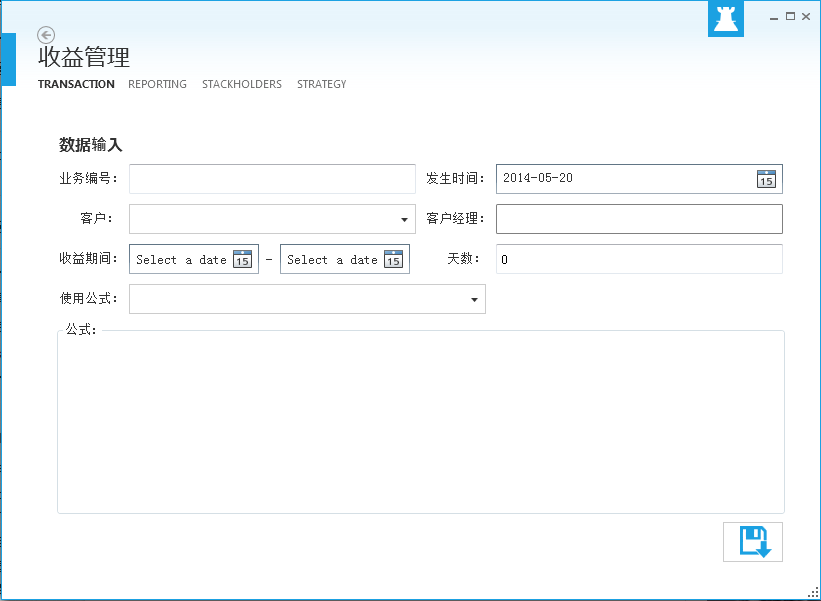
[c. 对象的生命周期 20](#_Toc387396274)

[d. Hibernate的Session管理 21](#_Toc387396275)

[e. 用户会话与Hibernate Session 23](#_Toc387396276)

[f. 数据库Connection管理 24](#_Toc387396277)

# ModernWindow



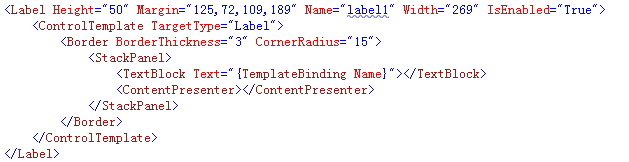
ModernWindow，包括最顶上的渐变色带，左侧标题栏，右上Logo，回退按钮，右上传统Bar，右下拖动区域，标题和菜单。

# WPF中扩展重写控件

WPF中扩展和重写控件是通过重写控件的<ControlTemplate>来实现的。以Label为例



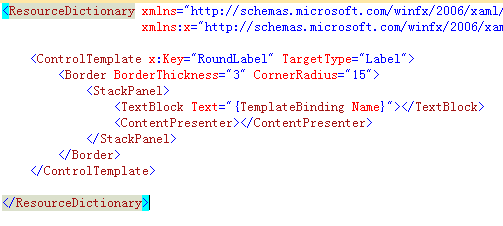
这个Label被加上了一个圆角矩形边框，先显示Name，再显示Content。通过重写Label的ControlTemplate就可以重新设置其Content，它有一个Border来实现圆角，然后一个TextBlock现实Name，最后是ContentPresenter来显示Content。注意TemplateBinding可以将对象（label1）的属性绑定到控件上。对应的Xaml如下：



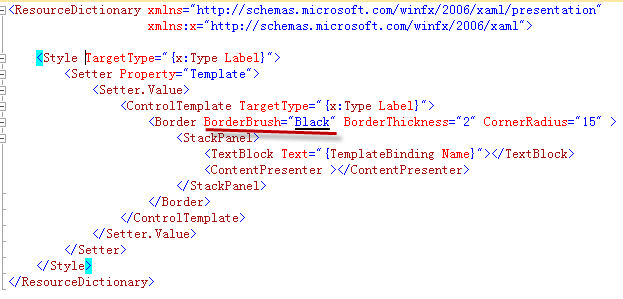
第二步是要使这个Label的Content能重复使用。具体做法是把ContentTemplate放到一个Resource文件中，所有需要这样显示的Label去引用这个Resoruce即可。

我们在根目录下建立一个Asset文件夹，其中放上一个Resource Dictionary Label.xaml，将<ControlTemplate>移到其中，并设置了Key。

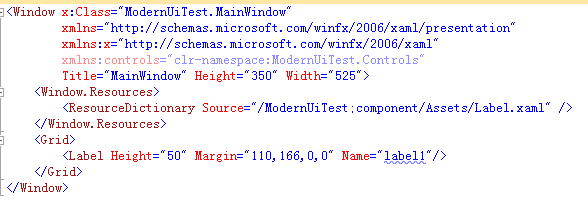




但这样一来Label并不会自动将ControlTemplate应用到Label，解决方法是使用<Style>的<Setter>，将设置的ControlTemplate设置到ControlTemplate属性上。因此Label.xaml需要改成如下形式：



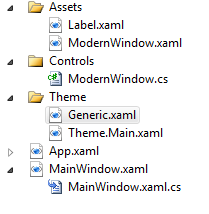
然后将Label.Xaml添加引用到资源，再讲Label中多余的代码删除即可。由于我们在Style中设置的是将所有Label类型的对象都做如下设置，因此Window中Label控件代码变得十分简单。注意ResourceDictioanry中需要设置Border的BorderBrush，因为默认是透明色。



对于其他控件的扩展也是按照这个顺序。

1. 新建一个控件的ResourceDictionary文件，添加Style并改写其Tempalte属性，具体内容为ControlTemplate
2. 将此Style引用到控件上

# ModernWindow的扩展

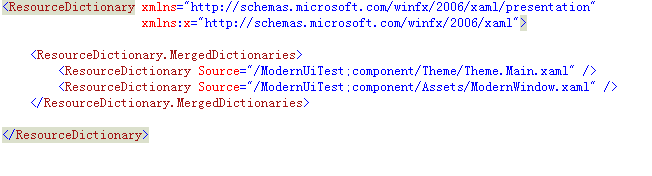


首先建立一个ModernWindow.xaml的ResourceDictionary

然后再创建一个ModernWindow的Class，让他继承Window。这个Class有两个作用，一是在Style和ControlTemplate中做TargtType，二是可以通过给它添加DependencyProperty用来绑定到控件上作为输入和输出。



第三，建立一个Theme，Generic.xaml，将所有用到的xaml都添加进去，供App.xaml引用。



最后将MainWindow的类型改成ModernWindow即可。要改Xaml和代码端。

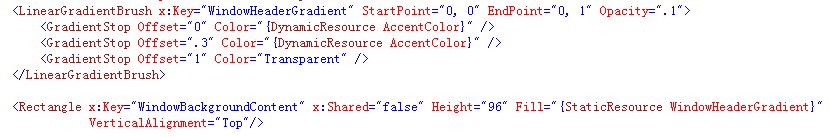
# 窗口顶端渐变色

之后的所有工作就是修改ModernWIndow.xaml，往ControlTemplate里添加东西即可。

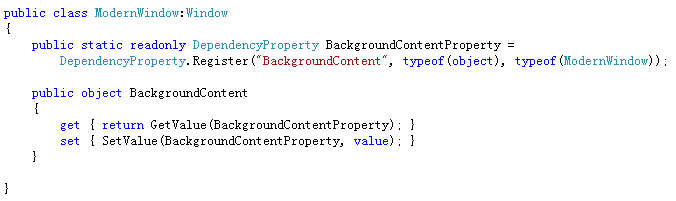
渐变色在WPF里可以使用LinearGradientBrush来实现；最基本的添加一个Item的控件是ContentControl，将它的Content设置成一个画出渐变色的矩形；最后将ContentControl布局到窗口上。

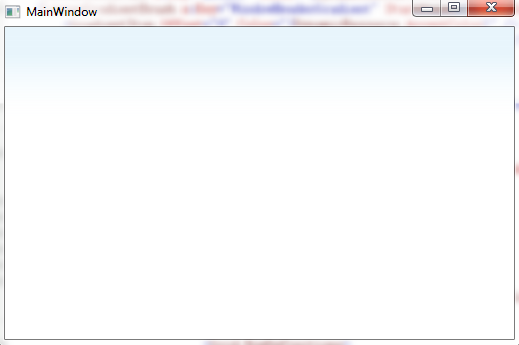


Xaml里<ContentControl>的Content属性被绑定到BackgroundContent上；在第一个Setter中，BackgroundContent已经被设置成了WindowBackgroundContent，而WindowBackgroundContent是一个使用了渐变色Brush的矩形，因此就实现了窗口顶端渐变色的效果。



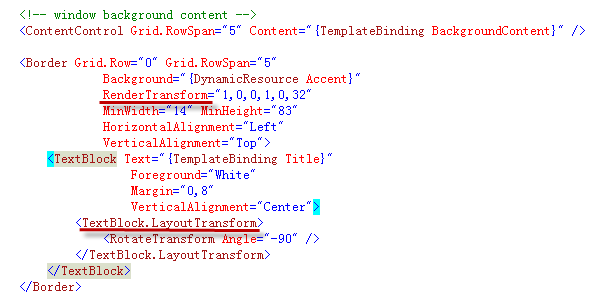
接着要在ModernWindow中添加BackgroundContent。既然要绑定，它就必须是个DependencyProperty。因此





# 左侧标题栏—— 平移与旋转

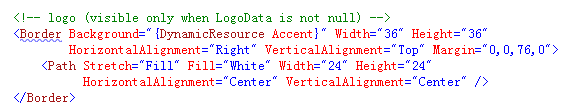
左侧标题栏其实是一个旋转了的TextBlock；但是TextBlock的长度会随其中文字的长短而改变，因此要先在TextBlock外画一个框；TextBlock用于显示窗口的Title，因此要绑定到Title属性上。



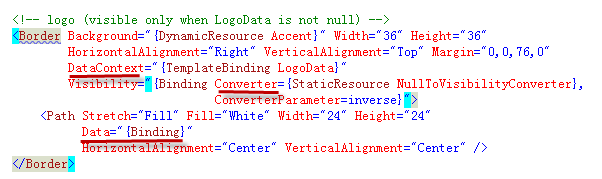
XAML中的Border就是外框，它从Row0开始占据5列的高度，然后通过RenderTransform将边框向下平移，以（1 0， 0 1）为原点向下平移32，从而空出一段，实际上这里RenderTransform使用的是TranslateTransform；然后加入一个textBlock来显示文本，文本要做-90度的旋转，因此使用RotateTransform。

# Logo栏——属性为空不显示

Logo栏与左侧标题栏的实现相似，先设置一个边框，然后在其中添加一个Path来显示图形。边框在设置时，先射程右对齐，然后通过margin来设置距离右边的位置。Path可以绑定属性。



接下来要实现的功能是，Logo栏要绑定到LogoData属性，当LogoData属性为空时，Logo栏不显示。为了实现这个功能，就必须将属性绑定到Border而不是Path，因为LogoData为空时，Border也不能显示，但真正使用Binding的却是Path，因此可以设置Border的DataContext为LogoData，在Path中再绑定。



具体的做法是将Visibility绑定到一个Converter上，将LogoData作为Border的DataContext，当LogoData为空时，Converter返回Visibility.Collapsed，否则返回Visibility.Visible。

# WPF笔记

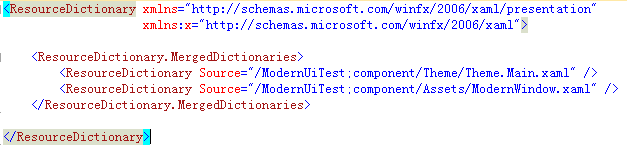
# APP.xaml中引用Resource Dictionary

路径格式为/dll;component/xaml文件的路径



# 引用多个Resource Dictionary

先新建一个ResourceDictionary，将所有需要的文件都添加到MergedDictionarties中，再由app.xaml应用它即可。



# 引用Style

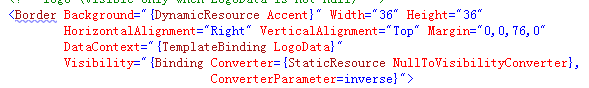
使用<Style>的BasedOn属性

# 窗台Xaml引用Namespace

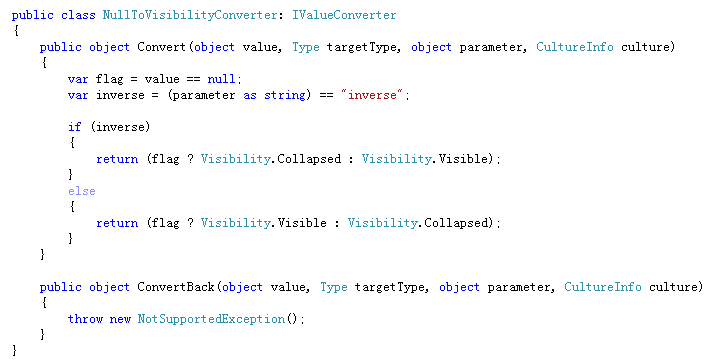


# 自定义Converter — IValueConverter

Convert的作用是把某个绑定的输入值转换成被绑定的属性的赋值。如下图例，Boder的DataContext是一个LogData对象，它的Visibility属性被绑定了一个NullToVIsibilityConverter，它负责将LogData的值转换成Visibility所需的Visibility.Collapsed或者Visibility.Visible。



要使用NullToVIsibilityConverter我们还需自定义它。如下图示，我们创建一个NullToVIsibilityConverter Class，让他实现IValueConverter接口。



IValueConverter接口有两个函数，其中主要的是Convert函数，value参数传入的是控件的DataContext，TargerType就是属性的类型，此处是Vlisibility枚举，parameter是ConverterParameter设定的值。