统一样式

DUI实现大概思路

一、核心架构设计

1. 颜色解析中间层

```
class CColorResolver {
public:
    static COLORREF Resolve(const std::wstring& alias);
    static void SetTheme(const std::wstring& themeName);
private:
    static std::map<std::wstring, COLORREF> m_themeColors; // 颜色别名映射表
};
```

2. 控件属性扩展

在DUI控件基类中添加颜色别名处理逻辑：

```
class CDuiControl {
protected:
    virtual void ParseAttribute(LPCWSTR name, LPCWSTR value) {
        if (StrCmpW(name, L"color-alias") == 0) {
            m_realColor = CColorResolver::Resolve(value); // 解析颜色别名
        }
    }
private:
    COLORREF m_realColor;
};
```

二、动态主题切换实现

1. 主题配置文件格式

```
<!-- theme_blue.xml -->
<Theme>
    <Color alias="primary" value="#2B579A"/>
    <Color alias="secondary" value="#00A2ED"/>
</Theme>
```

2. 主题加载引擎

```
void CColorResolver::LoadTheme(const std::wstring& xmlPath) {
    CMarkup xml;
    xml.Load(xmlPath);
    if (xml.FindElem(L"Theme")) {
        xml.IntoElem();
        while (xml.FindElem(L"Color")) {
            std::wstring alias = xml.GetAttrib(L"alias");
            COLORREF color =
CRenderEngine::ConvertColor(xml.GetAttrib(L"value"));
            m_themeColors[alias] = color;
        }
    }
}
```

三、控件渲染优化

1. 延迟渲染机制

```
void CDuiButton::DoPaint(HDC hDC) {
    if (m_realColor == CLR_INVALID) { // 首次渲染时解析
        m_realColor = CColorResolver::Resolve(m_colorAlias);
    }
    CRenderEngine::DrawColor(hDC, m_rcPaint, m_realColor);
}
```

2. 主题切换通知

```
void CColorResolver::BroadcastThemeChange() {
    CDialogBuilder::NotifyAllWindows([&](CWindowUI* pWindow) {
        pWindow->Refresh(); // 强制所有窗口重绘
    });
}
```

四、高级功能扩展

1. 颜色表达式支持

```
COLORREF CColorResolver::Resolve(const std::wstring& expr) {
    if (expr.find(L"lighten") != std::wstring::npos) { // 支持颜色运算
        return LightenColor(ParseColor(expr), 20%);
    }
    return m_themeColors[expr];
}
```

2. 运行时主题编辑器

```
void CThemeEditorDlg::OnColorChanged() {
    COLORREF newColor = m_colorPicker.GetColor();
    CColorResolver::SetColor(m_selectedAlias, newColor);
    CColorResolver::BroadcastThemeChange(); // 实时预览修改
}
```

五、性能优化策略

1. 颜色缓存机制

```
class CDuiControl {
private:
    mutable COLORREF m_cachedColor = CLR_INVALID; // 缓存解析结果
    mutable bool m_needColorRefresh = true;
};
```

2. 增量式主题更新

```
void CColorResolver::UpdatePartialTheme(const std::wstring& changedAliases) {
    CDialogBuilder::NotifyRelatedControls(changedAliases); // 仅更新相关控件
}
```

六、完整调用示例

```
// 初始化主题
CColorResolver::LoadTheme(L"themes/default.xml");

// 创建使用颜色别名的按钮
CDuiButton* pBtn = new CDuiButton;
pBtn->SetAttribute(L"color-alias", L"primary");

// 运行时切换主题
CColorResolver::LoadTheme(L"themes/dark.xml");
CColorResolver::BroadcastThemeChange();
```

MFC实现大概思路

1. 颜色别名映射系统

- **建立颜色别名表**：使用`std::map<CString, COLORREF>`或`CMapStringToPtr`存储颜色别名与真实值的映射关系，例如将`"primaryColor"`映射为`RGB(255,0,0)`。
- **初始化映射表**：在应用初始化阶段预定义别名，如：

  ```
  m_colorAliases["primaryColor"] = RGB(255,0,0);
  m_colorAliases["secondaryColor"] = RGB(0,255,0);
  ```

2. 别名解析机制

- **解析函数封装**：创建ResolveColorAlias方法，输入别名返回真实颜色值：

```
COLORREF CMyApp::ResolveColorAlias(const CString& alias) {
    COLORREF color;
    return (m_colorAliases.Lookup(alias, color)) ? color : RGB(0,0,0); // 未找到时返回默认颜色
}
```

3. 控件颜色动态设置

- **重写`OnCtlColor`消息**：在对话框或父窗口类中拦截控件绘制消息，将别名转换为真实颜色

```
HBRUSH CMyDialog::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor) {
    if (pWnd->GetDlgCtrlID() == IDC_CUSTOM_BUTTON) {
        COLORREF realColor = ResolveColorAlias(m_buttonColorAlias); // 获取别名对应的真实颜色
        pDC->SetTextColor(realColor);
        pDC->SetBkColor(RGB(255,255,255));
        return (HBRUSH)GetStockObject(WHITE_BRUSH);
    }
    return CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
}
```

4. 动态更新机制

- **触发颜色更新**：当别名对应的颜色值变化时，调用Invalidate()强制重绘控件

```
void CMyDialog::OnChangeColorAlias() {
    m_colorAliases["primaryColor"] = RGB(0,0,255); // 修改别名映射值
    GetDlgItem(IDC_CUSTOM_BUTTON)->Invalidate();   // 触发按钮重绘
}
```

使用示例

```
// 别名管理类声明
class CColorAliasManager {
public:
    void AddAlias(const CString& alias, COLORREF color);
    COLORREF ResolveAlias(const CString& alias);
private:
    CMapStringToPtr m_aliasMap;
};

// 在对话框中使用别名
void CMyDialog::OnPaint() {
    CPaintDC dc(this);
    COLORREF bgColor = m_colorManager.ResolveAlias("windowBgColor");
    dc.FillSolidRect(GetClientRect(), bgColor);
}
```

QT实现大概思路

1. 颜色别名映射系统

- **创建颜色管理器**：通过单例类ColorManager管理别名与颜色的映射关系，支持运行时修改

```
// ColorManager.h
class ColorManager : public QObject {
    Q_OBJECT
public:
    static ColorManager* instance();
    void setColor(const QString& alias, const QColor& color);
    QColor resolve(const QString& alias) const;
    void loadFromJson(const QString& path); // 从JSON文件加载配置
signals:
    void colorChanged(const QString& alias); // 颜色变化信号
private:
    QHash<QString, QColor> m_colors;
};
```

2. 动态样式生成

- **带占位符的QSS模板**：在样式表中使用{{alias}}标记颜色别名

```
/* style.qss */
QPushButton {
    background-color: {{primaryColor}};
    border: 1px solid {{secondaryColor}};
}
```

- **样式渲染引擎**：替换占位符为真实颜色值

```
QString renderStyleSheet(const QString& templatePath) {
    QFile file(templatePath);
    file.open(QIODevice::ReadOnly);
    QString style = file.readAll();
    auto colorMap = ColorManager::instance()->allColors();
    for (auto it = colorMap.begin(); it != colorMap.end(); ++it) {
        style.replace("{{" + it.key() + "}}", it.value().name());
    }
    return style;
}
```

3. 控件颜色绑定

- **属性绑定装饰器**：通过Q_PROPERTY动态绑定颜色

```
class ThemeAwareWidget : public QWidget {
    Q_OBJECT
    Q_PROPERTY(QColor bgColor READ bgColor WRITE setBgColor NOTIFY
bgColorChanged)
public:
    explicit ThemeAwareWidget(QWidget* parent = nullptr) {
        connect(ColorManager::instance(), &ColorManager::colorChanged,
                this, &ThemeAwareWidget::updateColor);
    }
    void setColorAlias(const QString& alias) {
```

```
        m_alias = alias;
        updateColor();
    }
private slots:
    void updateColor() {
        setBgColor(ColorManager::instance()->resolve(m_alias));
    }
};
```

C#实现

```
private string theme; //当前主题
private Dictionary<string, Dictionary<string, object>> _themeResources; // 主题资源

// 主动获取主题资源
theme =

// 主动获取当前主题
_themeResources =

// 根据当前主题加载资源到应用程序内
var converter = new BrushConverter();
Dictionary<string, object> colorDic = _themeResources[theme];

ResourceDictionary resouceDict = new ResourceDictionary();
foreach(KeyValuePair<string, object> kvp in colorDic)
{
    resouceDict.Add(kvp.Key, converter,ConcertFrom(kvp.Value));
}

if(Application.Current.Resouce.MergedDictionaries.Count == 0)
{
    Application.Current.Resouce.MergedDictionaries.Add(resouceDict);
}
else
{
    Application.Current.Resouce.MergedDictionaries[0] = resouceDict;
}

// 色值实际应用到按钮
DemoButton.SetResourceReference(Button.BackgroundProperty, "color-background2");
DemoButton.SetResourceReference(Button.ForegroundProperty, "color-font5");

// 主题切换事件处理 更换应用资源 事件名：SchemeChange 同上处理
```

示例

具体使用参考： wpf进程间通信中得实现