EasyGraphics

概述

EasyGraphics是一个轻量的构建图形界面应用的跨平台C++图形化接口，以易使用、源码易读、易扩展为设计目标，目前支持嵌入式Linux系统和Windows系统。

设计结构

EasyGraphics主要由事件系统、动画系统、计时系统、控件系统、渲染系统、属性系统、层次构造系统组成。

编译

EasyGraphics支持C++17及以上版本。在嵌入式平台编译时只需要使用cmake：

```
mkdir build && cd build
cmake .. && make
```

在其他平台编译时，请修改CMakeLists.txt中Compiler选项。

以上过程将编译EasyGraphics提供的四个示例，如果希望编译用户自己的程序，可以参考CMakeLists.txt，替换其中的示例文件。

使用

EasyGraphics提供给用户的函数和类主要包括各种控件类、属性类、延时调用函数、渲染控制函数、层次构造函数。一般而言，仅使用控件类的各种方法和渲染控制函数，就足够创建一个小型的图形界面程序。以计算器为例：

```
#include "include/OverlapPanel.hh"
#include "include/Grid.hh"
#include "include/Label.hh"
#include "system/SystemIO.hh"
#include <iostream>
#include <string>
using namespace easy;

int as_int(std::string s) {
    const char* str = s.c_str() + 2;
    int x = 0;
    bool neg = false;
    if (str[0] == '-') str++, neg = true;
    while (*str) {
        x = 10 * x + *str - '0';
        str++;
    }
    return neg ? -x : x;
}

int main() {
    Register<Renderer>();
```

```
    int result = 0;
    char op = '\0';
    bool wait_new = true;

    Label input = MakeLabel();
    input->Margin = { 20 };
    input->SpecSize = { 700, 70 };
    input->BackgroundColor = Color::FromARGB(0xF0F0F0);
    input->FontSize = FontSizeType::Large;
    input->FontColor = Colors::Black;
    input->VerticalAlignment = VerticalAlignType::Center;
    input->HorizontalAlignment = HorizontalAlignType::Left;
    input->FontHorizontalAlignment = HorizontalAlignType::Left;
    input->FontVerticalAlignment = VerticalAlignType::Center;

    Grid btns = MakeGrid({ 80,80,80,80 }, { 80,80,80,80 });
    btns->Margin = { 20 };


    auto MakeBtn = []() {
        Label btn = MakeLabel();
        btn->SpecSize = { 76, 76 };
        btn->VerticalAlignment = VerticalAlignType::Center;
        btn->HorizontalAlignment = HorizontalAlignType::Center;
        btn->BackgroundColor = Color::FromARGB(0xF0F0F0);
        return btn;
    };

    for (int i = 0; i < 10; ++i) {
        Label btn = MakeBtn();
        btn->Text = std::to_string(i);
        btn->Click += [&, i](Element, MouseEventArgs) {
            if (wait_new) input->Text = "  " + std::to_string(i), wait_new =
false;
            else input->Text += std::to_string(i);
            Renderer::Invalidated() = true;
        };
        btn->BackgroundColor = Color::FromARGB(0xFAFAFA);
        if (i) btns->Set((i - 1) / 3, (i - 1) % 3, btn);
        else btns->Set(3, 1, btn);
    }


    for (int i = 0; i < 4; ++i) {
        Label btn = MakeBtn();
        btn->Text = "+-*/"[i];
        btn->Click += [&, i](Element, MouseEventArgs) {
            op = "+-*/"[i];
            wait_new = true;
            result = as_int(input->Text);
            Renderer::Invalidated() = true;
        };
        btns->Set(i, 3, btn);
    }

    Label eq = MakeBtn();
    eq->Text = "=";
```

```
    eq->Click += [&](Element, MouseEventArgs) {
        if (op == '+') input->Text = "  " + std::to_string(result +
as_int(input->Text));
        else if (op == '-') input->Text = "  " + std::to_string(result -
as_int(input->Text));
        else if (op == '*') input->Text = "  " + std::to_string(result *
as_int(input->Text));
        else if (op == '/' && as_int(input->Text)) input->Text = "  " +
std::to_string(result / as_int(input->Text));
        op = '\0';
        result = as_int(input->Text);
        wait_new = true;
        Renderer::Invalidated() = true;
    };
    btns->Set(3, 0, eq);

    Label cl = MakeBtn();
    cl->Text = "C";
    cl->Click += [&](Element, MouseEventArgs) {
        input->Text = "  0";
        op = '\0';
        result = 0;
        wait_new = true;
        Renderer::Invalidated() = true;
    };
    btns->Set(3, 2, cl);


    Grid form = MakeGrid({ 130, 0 }, { 0 });
    form->BackgroundColor = Color::FromARGB(0xE6E6E6);
    form->Set(0, 0, input);
    form->Set(1, 0, btns);
    Renderer::MainLoop(form);
}
```

用户只需要关心控件的使用，而不需要控制时钟、动画、渲染、事件触发等细节（不过，注册事件往往需要用到lambda表达式，一般来说这是用户必须掌握的语言特性）；另一方面，EasyGraphics的控件API从设计上尽量直观易用，大部分参考C# WPF风格设计：

- 向elem的拖拽事件注册一个侦听函数：

```
elem->Drag +=
    [](Element sender, MouseEventArgs args) {
        sender->Margin.Top += args.offset.Y;
        sender->Margin.Left += args.offset.X;
        Renderer::Invalidated() = true;
    };
```

- 创建一个持续500ms、渐出的平移动画，作用于控件grids[k][j]：

```
g->BeginAnimation(
        grids[k][j], &_Element::Margin,
        grids[k][j]->Margin,
        grids[k][j]->Margin +
               Rect{ 0, emptySlot * (gsize + margin), 0, 0 },
        500, EaseOutBounce,
        true);
```

具体文档可参考快速入门。