TouchGFX 中 Callback 模板实现原理

前言

TouchGFX 为 MCU 带来了炫彩丰富的 GUI 界面，使得基于 STM32 芯片的人机界面开发非常方便而友好，比如可以在 TouchGFX Designer 中创建一个按键，在 interaction 中给按键添加响应；或者创建多个界面，在界面间进行切换；这些功能由 designer 帮我们自动生成代码实现了，那与之对应的功能响应代码具体是如何实现的呢？

TouchGFX 是用 C++编写的，借助 C++的模板特性，TouchGFX 定义了一组 Callback 模板，基于此模板来实现上述响应的功能。

Callback 模板

在 TouchGFX 中，Callback 模板的描述放在 Callback.hpp 文件中，在此定义了两组模板：GenericCallback 与 Callback 模板。

GenericCallback 模板组

GenericCallback 为 Callback 模板的模板基类。在 GenericCallback 模板中，定义了两个接口函数：isValid 与 execute；其中 isValid 是来检测 Callback 是否被初始化过，而 execute 函数用于调用实际要执行的函数。GenericCallback 模板组总共定义了 4 个模板，模板之间的差别在于 execute 函数的参数个数不同，4 个模板分别对应 execute 函数带有 0 个参数，1 个参数，2 个参数与 3 个参数。本文中仅列出 execute 带一个参数的情况。

下面是 execute 函数带 1 个参数的 GenericCallback 模板：

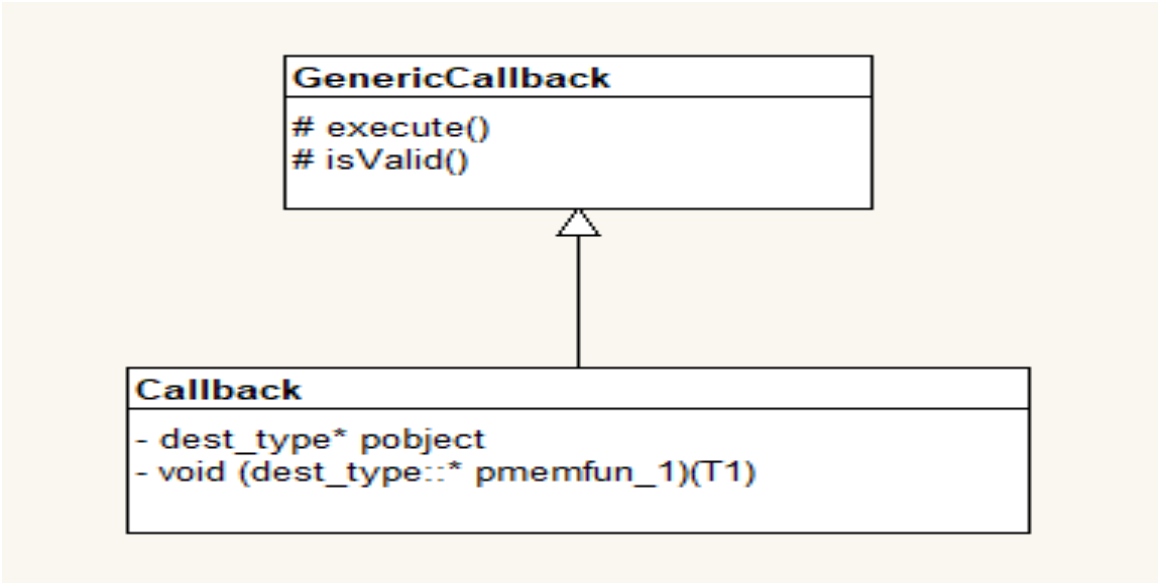
```
template <class T1>
class GenericCallback
{
public:
    virtual ~GenericCallback()
    {
    }
```

```
    virtual void execute(T1 val1) = 0;

    virtual bool isValid() const = 0;
};
```

Callback 模板组

Callback 模板由 GenericCallback 派生而来。Callback 模板组也有 4 个模板，分



别对应于包含不同参数个数 execute 函数的 GenericCallback 模板，继承关系如下图：

下面是 execute 函数带 1 个参数的 Callback 模板：

```
template <class dest_type, typename T1>
struct Callback<dest_type, T1, void, void> : public GenericCallback<T1>
{
    Callback() : pobject(0), pmemfun_1(0) { }

    Callback(dest_type* pobject, void (dest_type::*pmemfun_1)(T1))
    {
        this->pobject = pobject;
        this->pmemfun_1 = pmemfun_1;
```

```
    }

    virtual void execute(T1 t1)
    {
        (pobject->*pmemfun_1)(t1);
    }

    virtual bool isValid() const
    {
        return (pobject != 0) && (pmemfun_1 != 0);
    }

private:
    dest_type* pobject;
    void (dest_type::* pmemfun_1)(T1);
};
```

在 Callback 结构中定义了两个私有的成员指针，pobject 为 dest_type 类型对象的指针，pmemfun_1 为 dest_type 对象的成员函数指针，是指向 dest_type 类型的成员函数。这样当 Callback 结构初始化好了以后，再调用 execute 函数时，就是直接调用了 dest_type 类型对象的成员函数 pmemfun_1。

在使用 Callback 模板时，根据模板的特性，C++编译器会自动选择最准确的模板匹配，来生成对应的模板类或模板函数。因此在使用 Callback 模板时，可以使用不同的类型参数个数来匹配使用不同的模板。

Callback 模板在 TouchGFX 中的使用

页面切换 - pendingScreenTransitionCallback

在 TouchGFX 启动过程中会初始化 MVPApplication 对象，在 MVPApplication 类中定义了一个成员 pendingScreenTransitionCallback，它是 GenericCallback<>*类型的指针，并初始化为 0；

TouchGFX 会调用 handlePendingScreenTransition 函数进行页面切换，最终调用 evaluatePendingScreenTransition 函数，在此函数中会用 isValid()函数来检查 Callback 是否已经初始化；如果初始化了，就调用 execute 函数 (因为 Callback 为 GenericCallback<>*类型，因此 execute 函数不带参数) 。

```
void evaluatePendingScreenTransition()
{
    if (pendingScreenTransitionCallback &&
pendingScreenTransitionCallback->isValid())
    {
        pendingScreenTransitionCallback->execute();
        pendingScreenTransitionCallback = 0;
    }
}
```

具体的页面切换过程（进入第一个 screen）：
在 TouchGFX 初始化过程中，会初始化 FrontendApplication 对象 app，在此对象的基类中包含一个 transitionCallback 成员，其类型为：
touchgfx::Callback<FrontendApplicationBase> transitionCallback
在进入第一个 Screen 时，会调用 app.gotoScreen1ScreenNoTransition()，在此函数中，会将 pendingScreenTransitionCallback 设置为指向 transitionCallback 的指针，最终由 TouchGFX 框架调用 execute 函数，执行 makeTransition 完成页面的切换。

// 初始化 FrontendApplicationBase::transitionCallback，transitionCallback 的私有成员初始化：
// pobject 赋值为 this(指向 FrontendApplicationBase)
// pmemfun_1 赋值为 &FrontendApplication::gotoScreen1ScreenNoTransitionImpl

```
void FrontendApplicationBase::gotoScreen1ScreenNoTransition()
{
    transitionCallback = touchgfx::Callback<FrontendApplicationBase>(this,
&FrontendApplication::gotoScreen1ScreenNoTransitionImpl);
    pendingScreenTransitionCallback = &transitionCallback;
}
```

//execute 函数内实际执行的函数(pobject->*pmemfun_1)()，即为如下函数：

```
void FrontendApplicationBase::gotoScreen1ScreenNoTransitionImpl()
{
    makeTransition<Screen1View, Screen1Presenter, touchgfx::NoTransition,
Model >(&currentScreen, &currentPresenter, frontendHeap, &currentTransition,
&model);
}
```

按键响应 - flexButtonCallback

在 TouchGFX designer 中创建一个按键，并设置 interaction 后，会在 MainViewBase 类中创建两个私有成员：

//按键响应函数，实现用户的动作响应

void flexButtonCallbackHandler(const touchgfx::AbstractButtonContainer& src)

//Callback 对象

touchgfx::Callback<MainViewBase, const touchgfx::AbstractButtonContainer&> flexButtonCallback

以及一个按键成员，例如 clickButton：

touchgfx::TextButtonStyle< touchgfx::ImageButtonStyle< touchgfx::ClickButtonTrigger > > clickButton;

在 MainViewBase 的构造函数中，会初始化 flexButtonCallback 成员，并且调用 clickButton->setAction 函数，设置此按键的 AbstractButton::action 为 flexButtonCallback，

MainViewBase::MainViewBase():flexButtonCallback(this, MainViewBase::flexButtonCallbackHandler)

其中：

flexButtonCallback 的 pobject 成员初始化为 this(指向 MainViewBase 的指针)

flexButtonCallback 的 pmemfun_1 成员初始化为 &MainViewBase::flexButtonCallbackHandler

由于 flexButtonCallback 的类型是

struct Callback<dest_type, T1, void, void> : public GenericCallback<T1>

因此 flexButtonCallback 的 execute 函数带有一个参数，参数类型为 const touchgfx::AbstractButtonContainer&，最终此参数会传递给 flexButtonCallbackHandler 函数。

在程序运行时，点击按键后，TouchGFX 框架会捕获此事件，并在 AbstractButton::handleClickEvent 函数中调用 action->execute(*this)函数，最终即执行了 flexButtonCallbackHandler 函数，实现了用户的动作响应。

小结

本文介绍了 TouchGFX 中 Callback 模板的基本原理，并结合两个例子，简要说明了 Callback 模板在框架中的使用方法。只要注意 Callback 的类型、初始化过程，以及最终 execute 函数执行的函数主体，就能理解 Callback 模板了，这对深入理解 TouchGFX 也有一定帮助。

重要通知 - 请仔细阅读

意法半导体公司及其子公司（“ST”）保留随时对ST 产品和/ 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在
订货之前应获取关于ST 产品的最新信息。ST 产品的销售依照订单确认时的相关ST 销售条款。

买方自行负责对ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的ST 产品如有不同于此处提供的信息的规定，将导致ST 针对该产品授予的任何保证失效。

ST 和ST 徽标是ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。