**Image Beautify 总体报告**

小组成员：黄文璨、叶昕洋、林锦铿（临时工）

报告日期：2018年7月23日

# 一、MVVM模式简介

MVVM是模型－视图－视图模型 (Model-View-ViewModel) 模式。MVVM模式由视图(View)、视图模型(ViewModel)、模型(Model)三部分组成，通过这三部分实现UI逻辑、呈现逻辑和状态控制、数据与业务逻辑的分离。

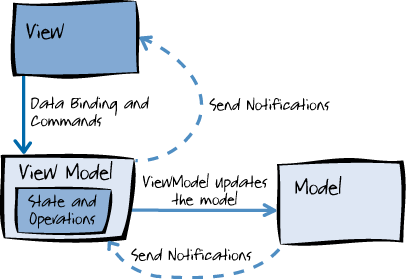


图1-1 MVVM模式结构图

如图1-1所示，在MVVM模式中View层专门负责UI界面的显示和用户交互逻辑，可以是控件容器和集合。ViewModel是View的抽象，负责View与Model之间进行信息转换，同时将View的Command传送到Model。Model层是数据访问层，包括业务逻辑。

View与ViewModel之间通过Data Binding进行数据绑定，通过Commands绑定操作的调用，当属性改变时ViewModel向View发出通知（Send Notifications）。

ViewModel与Model之间的联系较为紧密。ViewModel通过聚合Model对象直接调用Model的方法对Model中的数据进行更新（ViewModel updates the Model）。当Model触发事件（Send Notifications）时，ViewModel负责接受事件。

# 二、MVVM模式的具体实现

## 2.1具体实现概述

在具体实现中，除了基础的View、ViewModel、Model三大层外，还可以增加Common、App、Algorithm等层，如图2-1所示。

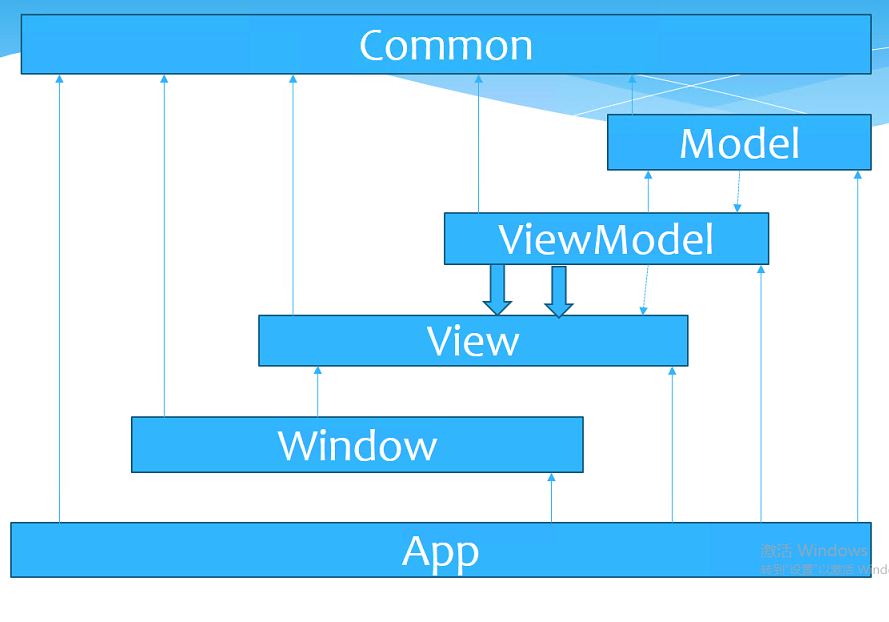


图2-1 MVVM具体实现模块层次图

其中Common层提供基本的类，包括对底层系统的封装，通用性质的数据结构和算法（复杂的算法也可以单独放在Algorithm层）等。其他各层都可以调用Common层提供的类。App层负责装配View、ViewModel、和Model层。

本工程在MVVM框架下按照上述模块层次图设计了如下文件结构。

表2-1 工程文件结构与描述

|  |  |
| --- | --- |
| 文件结构 | 描述 |
| 屏幕快照%202018-07-23%20下午8.39.15.png | 总体采用通用的App + Common + View/ViewModel/Model模块布局。 |
| 屏幕快照%202018-07-23%20下午8.39.49.png | Common层中包含各元素基类etlbase.h文件，其中定义了指令基类、信号基类、属性通知类和命令通知类。  Commands中传递的参数封装为parameters.h中的参数基类。  各种命令的类型在type.h中定义为枚举变量。  各模块中所需要的公共函数定义在util.h/cpp中。  Model层与算法层联系紧密，所以算法层filteroperation、filters、imageoperation、logoperation直接同model.h在一个文件夹下。 |
| 屏幕快照%202018-07-23%20下午8.40.04.png | View层包含主窗口mainview和多个子窗口dialogs，并且有接收器类sinks负责接收来自ViewModel的信号然后更新视图。  ViewModel层包含所有命令在commands文件夹中，并且有接收器类sinks负责接收来自Model的信号然后更新数据后转发到View层。 |

## 2.2 MVVM框架中的消息传输机制概览

MVVM框架中的消息传输对应属性改变的通知与接收、命令执行结果的通知与接收。前者发生在Model与ViewModel，ViewModel与View之间；后者发生在ViewModel与View之间。

View通过Commands绑定（**Commands Binding**）将界面交互参数直接传递到ViewModel中的Commands层，Commands进行界面交互参数的解压并调用ViewModel中的执行函数，这时该执行函数调用绑定到ViewModel层的Model中的相应函数对Model中的数据进行更改，同时发出属性改变的通知（**Property Notification**）或者运行失败返回错误代号。这时若命令执行失败，Commands层将发出命令执行结果为失败的通知（**Commands Notification**）。若命令成功执行，ViewModel层的接受器收到信号调用自身函数更新与View绑定过的数据（**Data Binding**），并转发属性通知到View层。此时View的属性接收器（**Property Sinks**）收到信号调用View层函数更新视图。

其中命令与数据的绑定直接通过std::shared\_ptr实现，通知与信号接受器通过C++对象继承与派生机制实现。

## 2.3 MVVM通用基类的设计实现

在MVVM框架中所有命令Commands都要继承一个命令基类ICommandBase，并实现SetParameter和Exec两个方法，即命令参数传递和命令执行。

class ICommandBase

{

public:

    virtual void SetParameter(const std::shared\_ptr<ParametersBase>& param) = 0;

virtual void Exec() = 0;

};

属性通知与命令通知都继承一个通知基类INotification，这个基类存储一个m\_array成员数组用于保存通知类的指针。

template <class T>

class INotification

{

public:

    void Clear() throw()

    {

        m\_array.clear();

    }

    void AddNotification(const std::shared\_ptr<T>& p)

    {

        m\_array.push\_back(p);

    }

protected:

std::vector<std::shared\_ptr<T> > m\_array;

};

属性通知代理继承通知基类，并以属性通知类作为模板参数，需要实现AddPropertyNotification和Fire\_OnPropertyChanged两个方法。命令通知代理与之相似。

template <class T>

class Proxy\_PropertyNotification : public INotification<IPropertyNotification>

{

public:

void AddPropertyNotification(const std::shared\_ptr<IPropertyNotification>& p)

    {

        AddNotification(p);

    }

    void Fire\_OnPropertyChanged(const propertyType ppt)

    {

        auto iter(m\_array.begin());

        for( ; iter != m\_array.end(); ++ iter ) {

            (\*iter)->OnPropertyChanged(ppt);

        }

    }

};

## 2.4 各个子模块的实现

见子模块报告。

# 三、项目管理工具链

## 3.1工具链概览

本工程采用的项目管理工具链如下表。

表3-1 项目管理工具链

|  |  |
| --- | --- |
| **名称** | **工具** |
| 版本控制 | Git, GitHub |
| 持续集成 | Appveyor |
| 构建工具 | CMake, QMake |
| 规范检查 | cpplint |
| 静态分析 | cppcheck |
| 复杂度分析 | cppncss |
| 单元测试覆盖率 | OpenCppCoverage |

## 3.2 GitHub 开源项目与组员贡献度概览

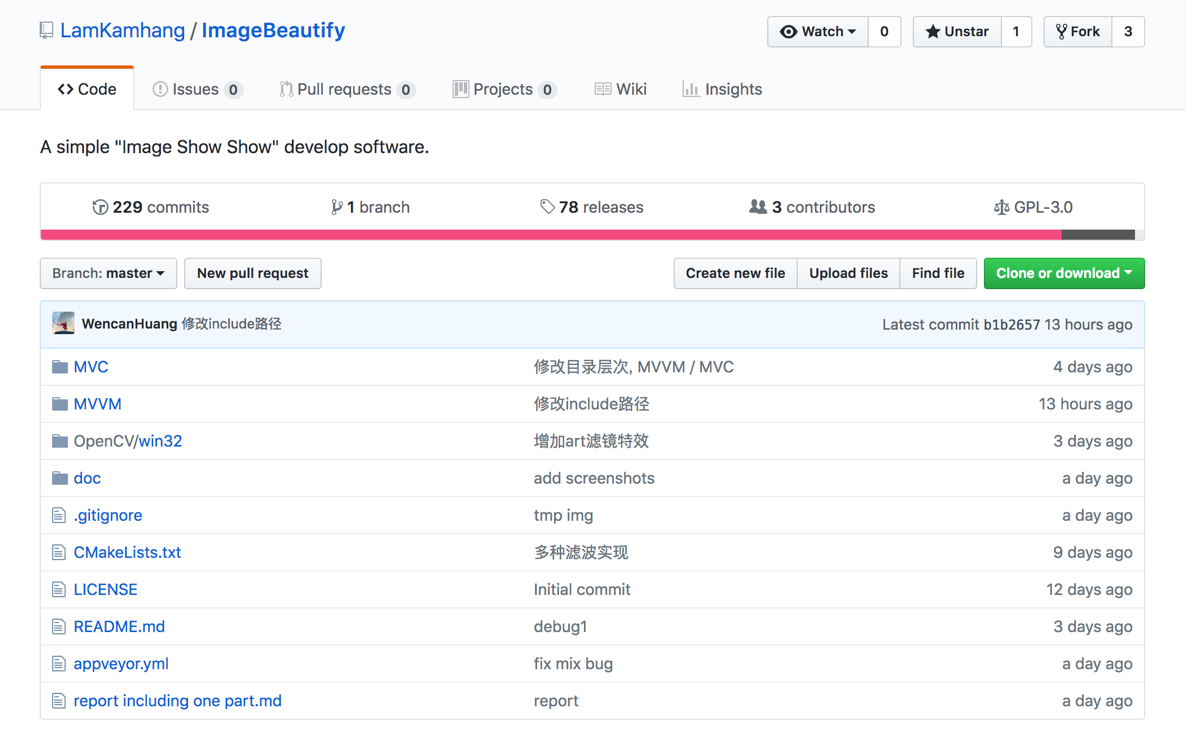


图3-2-1 项目主页



图3-2-2 组员贡献度

## 3.3 持续集成配置与版本发布

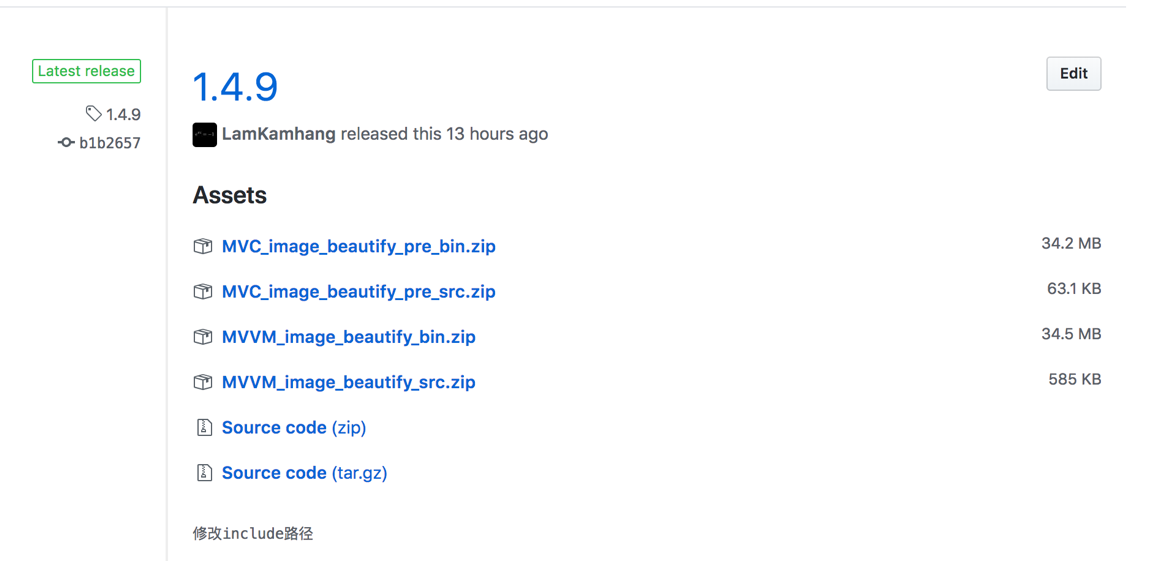


图3-3 版本发布示例

# 四、项目成果展示

|  |
| --- |
| 主界面 |
| ../gui/ImageBeautify/doc/screenshots/mainview.png |
| 主界面 |
| ../gui/ImageBeautify/doc/screenshots/tools.png |
| 工具菜单 |
| ../gui/ImageBeautify/doc/screenshots/special%20effects.png |
| 特效菜单 |
| 工具子窗口展示 |
| ../gui/ImageBeautify/doc/screenshots/scale.png |
| Scale/Rotation Dialog |
| ../gui/ImageBeautify/doc/screenshots/level.png |
| Level Adjustation Dialog |
| ../gui/ImageBeautify/doc/screenshots/histogram.png |
| Histogram Dialog |
| ../gui/ImageBeautify/doc/screenshots/filter.png |
| Filter Dialog |
| ../gui/ImageBeautify/doc/screenshots/dual.png |
| Dual Threshold Dialog |
| ../gui/ImageBeautify/doc/screenshots/curve.png  Contrast Curve Dialog |
| ../gui/ImageBeautify/doc/screenshots/clip.png |
| Clip Dialog |
| ../gui/ImageBeautify/doc/screenshots/morphology.png |
| Morphology Dialog |
| ../gui/ImageBeautify/doc/screenshots/algebraic.png |
| Algebraic Operation Dialog |
| ../gui/ImageBeautify/doc/screenshots/art%20effect.png |
| 特效子窗口 |

# 五、小组协作分工分析与总结

## 5.1 小组协作分工

|  |  |  |
| --- | --- | --- |
| 角色 | 姓名 | 主要工作 |
| 组长 | 黄文璨 | MVVM框架搭建、View层的主要实现、ViewModel层的主要实现 |
| 组员 | 叶昕洋 | 滤镜算法层的主要实现 |
| 临时工 | 林锦铿 | Appveyor配置、Model层与ViewModel层的主要实现 |

## 5.2 总结与心得

这次短学期围绕多人协作桌面应用程序开发的主题，以工业界、学术界专业的软件开发标准作为指导，从先进开发框架MVVM着手展开，内容充实精彩。十多天的工业级软件开发体验，总结下来遇到了不少艰难困苦，也收获了宝贵的知识与经验，更使大家懂得了团队协作的重要性，以及如何才能更好地团队协作。

具体地讲，开发ImageBeautify的过程中遇到的主要困难在于对MVVM框架的理解一直不够深刻。由于在此之前大家都主要采用单人开发的模式，界面、算法、功能实现全由一人独担，各个模块耦合紧密，当初次接触到MVVM这样的完全解耦合框架时感到有些不适应。MVVM框架中的消息传输机制、数据与命令的绑定机制精巧绝伦，我们琢磨了数天时间才大致有了实现的思路。在具体编程实现的过程中，受到袁老师的不断点拨，我们的框架一次又一次向着MVVM靠齐。最终MVVM框架深入每一个组员的心中，View、ViewModel、Model各个模块之间的联系，模块间数据的绑定与消息的传输流程终于被深刻理解并实现出来。

在MVVM框架搭建好后，团队协作变得方便高效起来。产品更新换代的速度很快，每次增加一个新功能只需要三个组员各司其职，在不同的模块写入对应的代码即可。代码的可维护性也极高，出错率很低，即使出现bug也能快速定位到某个Command并进行排除。

在这个项目的设计过程中，大家学到了完整的工业开发软件流程，体会到团队协作开发软件的过程，深刻理解了MVVM的架构模式，对图像处理的应用也有了初步的了解，收获颇丰。