Image Beautify 模块报告之 Model 层

报告人：叶昕洋
报告日期：2018 年 7 月 27 号

一、MVVM 模式中 Model 层简介

MVVM 架构使得 View 层与 Model 层完全实现解耦合，Model 层专门负责业务逻辑、操作实现，是程序的数据访问层。Model 层与 ViewModel 层的关系最为密切,ViewModel 可以通过聚合 Model 对象直接调用 Model 的方法对 Model 中的数据进行更新。当 Model 触发事件，更新数据或操作完成时，ViewModel 层接受事件并传递给 View 层面。
由此可以发现，Model 层的主要工作有以下两个方面：
1、存储实际数据（或有操作数据库的方法）。实现一系列对 Model 层数据进行操作的方法，并提供对外的接口。
2、实现属性通知的发送器。当接口被调用且数据被成功更新时，发出属性改变的通知给 ViewModel。

二、本项目中的 Model 层的实现

2.1 Model 层概览

Model 层文件结构	描述
model filteroperation filteroperation.cpp filters art_effect.cpp classic_effect.cpp imageoperation aux_image_alg.cpp imageoperation.cpp logoperation aux_log_alg.cpp logoperation.cpp model.cpp	在 Model 层中，我们需要对图片进行操作，我们将对图片的操作分成三类，为滤镜操作、图片代数操作以及日志三个操作。其中由于滤镜操作比较多也比较繁琐，所以我们在 filters 中存储器具体的操作算法，在 filteroperation 中调用 filters 中的函数以实现功能。在 imageoperation 中存储的图片的代数操作算法，在 aux_image_alg.cpp 中存储具体算法，在 imageoperation.cpp 中存储返回 bool 值的函数，若失败则返回 false，成功则传递图片。

	Logoperation 同理，aux_log_alg.cpp 中存储具体算法，在 Logoperation.cpp 中存储返回 bool 值的函数，若失败则返回 false，成功则传递图片。

2.2 Model 层的数据的存储与操作接口

数据的存储

数据存储	描述
private: QImage originImg; QImage mainImg; QImage subImg; QImage tmpImg; Log log;	在该项目中，Model 层需要存储的数据不多，分为图片数据（QImage）和图片管理数据（Log），图片数据有主图、子图、原图和临时图片。

数据的操作接口

基础操作	
函数名	实现功能
bool open_file(const QString &path); bool open_sub_file(const QString &path);	图片的打开功能
bool save_file(const QString &path);	图片的存储功能

bool sub2main();
bool origin2main();
bool main2sub();
bool sub2tmp();
	监视图片的数据变化

bool mix_tmp_main(int alpha);	将图片进行叠加处理
图象操作	

bool imageAdd(double param1, double param2);
bool imageSubtract(double param1, double param2);
bool imageMultiply();
	图象的加减乘除操作

bool getSingleChannel(ChannelType channel); bool grayScale();	图象获得单颜色通道 图象灰度化

bool adjustHue(QVector<int> hueValues);
bool adjustSaturation(int saturation);
bool adjustLightness(int lightness);
	调整图象的色相、饱和度与亮度

bool otsu(); bool dualThreshold(int thresh1, int thresh2);	图象二值化
bool nearnestInterpolation(int scale, int rotation); bool BilinearInterpolation(int scale, int rotation);	插值

bool meanFilter(int col, int row, int x, int y);
bool medianFilter(int col, int row, int x, int y);
bool gaussianFilter(int col, int row, int x, int y, double sigma);
	滤波器

bool sobelEdgeDetection(int threshold);
bool laplacianEdgeDetection(int threshold);
bool cannyEdgeDetection(int lo, int hi);
	边缘检测

bool houghLineDetect(); bool houghCircleDetect(int lo, int hi);	线路检测 圆检测

bool dilation(int size, int x, int y, int *array);
bool erosion(int size, int x, int y, int *array);
bool opening(int size, int x, int y, int *array);
bool closing(int size, int x, int y, int *array);
	图象的腐蚀和膨胀

bool obr(int size, int x, int y, int *array); bool cbr(int size, int x, int y, int *array);	形态学操作

bool linearContrastAdjust(int x1, int y1, int x2, int y2);
bool pieceLinContrastAdjust(int x1, int y1, int x2, int y2);
bool logContrastAdjust(double a, double b);
	对比度调整

bool expContrastAdjust(double a, double b);	
bool histogramEqualization(int *histo); bool colorLevel(PColorLevelData clData);	直方图均衡化
日志操作	
bool redo(); bool undo();	撤回操作
bool clear();	清空操作
滤镜操作	
void _emboss();	// 浮雕
void _sculpture();	// 雕刻
void _dilate();	// 虚幻
void _erode();	// 惊悚
void _frostGlass();	// 磨砂玻璃
void _sketch();	// 手稿
void _oilPaint();	// 油画
void _woodCut();	// 木刻
void _inverted();	// 反色
void _memory();	// 回忆
void _freezing();	// 冰冻
void _casting();	// 熔铸
void _comicStrip();	// 黑白漫画
void _sharpen();	// 锐化
void _colortoblack();	// 黑白
void _defog();	// 去雾气
void _soft();	// 柔和
void _balance();	// 均衡图
void _nostalgia();	// 怀旧
void _BlackComic();	// 连环画
void _timetuunel();	// 时光隧道
void _classiclomo();	// 经 典 lomo
void _whiteFace();	// 美白
void _beautifyFace();	// 美颜
void _pinkLady();	// 粉红佳人

2.3 Model 层属性通知信号发送器

Model 通知与 ViewModel 接收的具体实现：

代码	描述
class Model : public Proxy_PropertyNotification<Model> {	Model 类继承属性通知代理，用于发出通知

void ViewModel::bindModel(std::shared_ptr<Model> model){
this->model = model;
model->AddPropertyNotification(
std::static_pointer_cast<IPropertyNotification>(viewModelSink)
);
}
	ViewModel 在绑定 Model 的同时,Model 的属性通知代理也绑定了 ViewModel 的通知接收器

Model 层的属性通知信号发送器负责当图片的属性发生改变时发出相应的通知给 ViewModel 层，通过 ViewModel 层传递给 View 层。每一个操作都在判断其是否成功，然后发送器发出指令。

```
bool Model::sub2main()
{
    if (subImg.isNull())
    {
        return false;
    }
    else
    {
        mainImg = subImg.copy();
        Fire_OnPropertyChanged(MAIN_IMAGE);
        return true;
    }
}

bool Model::origin2main()
{
    if (originImg.isNull())
    {
        return false;
    }
    else
    {
        mainImg = originImg.copy();
        Fire_OnPropertyChanged(MAIN_IMAGE);
        return true;
    }
}

bool Model::main2sub()
{
    if (mainImg.isNull())
    {
        return false;
    }
    else
    {
        subImg = mainImg.copy();
        Fire_OnPropertyChanged(SUB_IMAGE);
        return true;
    }
}
```

三个函数监视着数据中图片的变化，若属性发生变化，则立刻返回正确的布尔值给 ViewModel 层。

2.4 Model 层命令通知接收器

接下来以图象的相加为例来解释当图片的操作成功时是如何返回布尔值给 Modelview。在 Model 层中有一个实体操作，和一个返回布尔值的操作。后者调用前者实现相应的操作，同时当操作成功时，后者会返回一个正确的布尔值给 ViewModel 层。

```
bool Model::imageAdd(double param1, double param2)
{
    if (mainImg.height() != subImg.height() || mainImg.width() != subImg.width())
    {
        return false;
    }
    else
    {
        QImage tmp(ImageOperations::imageAdd(mainImg, subImg, param1, param2));
        mainImg = tmp;
        Fire_OnPropertyChanged(MAIN_IMAGE);
        return true;
    }
}
```

在这个函数中，我们先调用了 imageAdd 这个实体操作函数，得到我们想要的结果后赋值给 mainImg 这个存储结构，监视其数据是否发生变化，返回布尔值。

其他操作结构与图象的加法结构大致相同，引用一个实体的操作函数，同时监视属性值的变化，若变化，则返回一个正确的布尔值。