使用 Qt 编写简单窗口 GUI 程序的指南

Instructions for using Qt to make a simple GUI window application

By Dr. Meng DING, 2017.5.7
For my CG class students

1 前提条件

你正确安装了某个版本的 c++编译器;

你已经在本地配置好了对应你 c++编译器的某个版本的 Qt 库。这里的配置是指已经产生了 Qt 的众多 lib 文件和 dll 文件（你可以从官网下载预编译版的 Qt 安装包(Pre-build-binary-library)，也可以通过编译源代码形式得到完整的 Qt）;

环境变量也以配好（确定 Qt 里的那些 dll 文件所在目录已经加到你的系统环境变量中）;

IDE 也已经准备好（无论是 vs 还是 qtcreator，如果是 vs，那么你最好下载对应版本的 qt-vs-add-in：对于 vs2010，只需 add-in-1.xx 即可，对于 2015，请下载针对 2015 及以上的 vs-add-in）。

2 编写 Qt 应用程序

如果你想对 Qt 有深入了解，那么请移步 qt 的帮助文档，assistance.exe，以及参看 example 里面的示例代码。

这里只给出基本的操作过程以及你完成图形学课程的实验所需要继承并且完成的主要函数。

2.1 IDE 方式

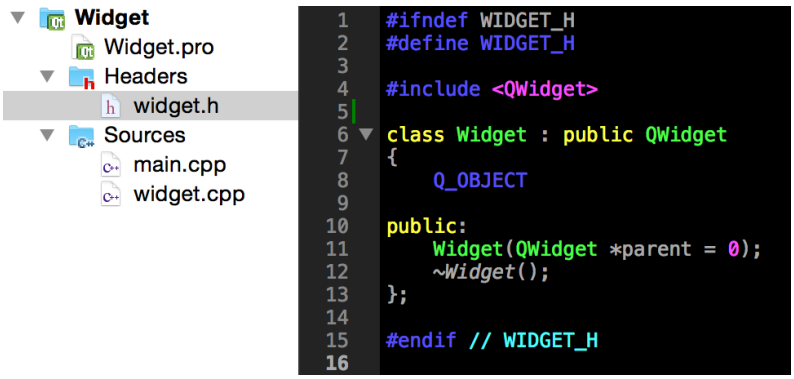
2.1.1 建立工程

对于使用 vs2010 的观众，和建立一般工程一样，选择建立新的工程，但是在向导中选择 Qt4(Qt5)工程，然后同样需要指定工程名以及路径，然后需要选择你的应用程序的基类，大型程序可以选用 QMainWindow，但是我们一般的窗口程序，选择 QWidget 即可，然后选择不生成 ui 界面。

然后一路下一步，就可产生整个工程。这时你可以直接点击编译运行，你就可以看见你的第一个 Qt 窗口程序。

在工程目录视图中，你应该看到三个文件，main.cpp，xxxx.h，xxxx.cpp。这里 xxxx 表示你的工程文件名或者是你在建立工程的过程中指定的文件名.

在 xxxx.h 文件中，就是 Qt 帮你建立好的你自己的窗口类（这里假设工程名为 Widget，那么 xxxx 就应该 widget，类名是 Widget）



2.1.2 修改窗口类声明

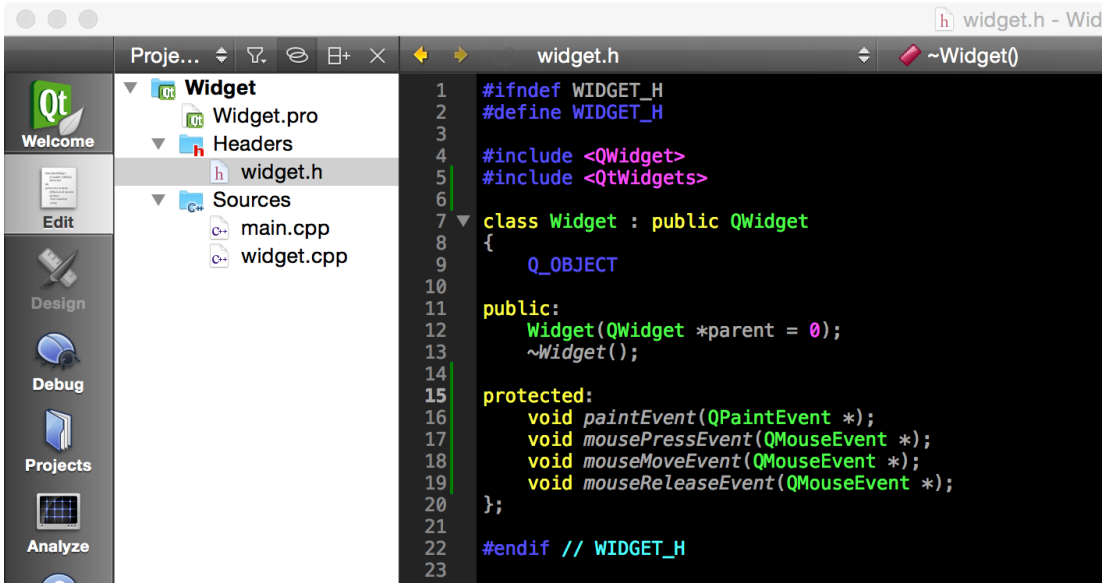
为方便起见，对于使用 Qt 5.0 以上的观众请在#include<QWidget>前面加上 #include<QtWidgets>。

对于使用 Qt4.x 的观众，请加上 #include <QtGui>。

为了完成基本的绘制和响应鼠标操作，你需要修改继承自基类的如下虚函数：

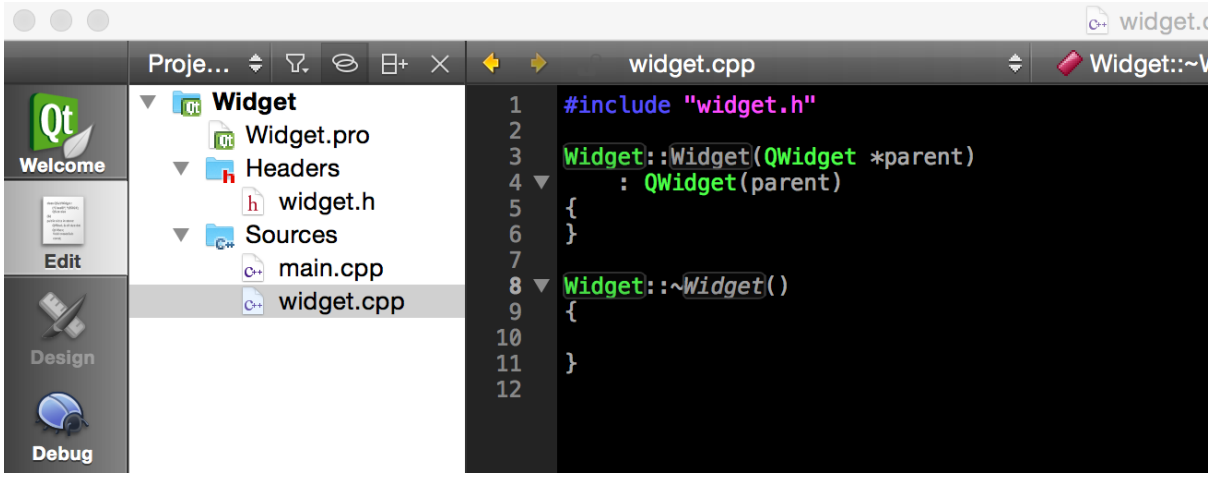
```
protected:
    virtual void paintEvent(QPaintEvent *event);
    virtual void mousePressEvent(QMouseEvent* event);
    virtual void mouseReleaseEvent(QMouseEvent *event);
    virtual void mouseMoveEvent(QMouseEvent *event);
```

于是得到类似如下的 Widget 类的声明



2.1.3 实现窗口类的新增成员函数

打开 widget.cpp 文件，你应该看到类似如下的内容：



在这里，添加刚才新声明的四个成员函数的实现：

```
void Widget::mouseMoveEvent(QMouseEvent *event){

}

void Widget::mouseReleaseEvent(QMouseEvent *event){

}

void Widget::mousePressEvent(QMouseEvent *event){

}

void Widget::paintEvent(QPaintEvent *event){

}
```

2.1.4 实现 paintEvent 函数

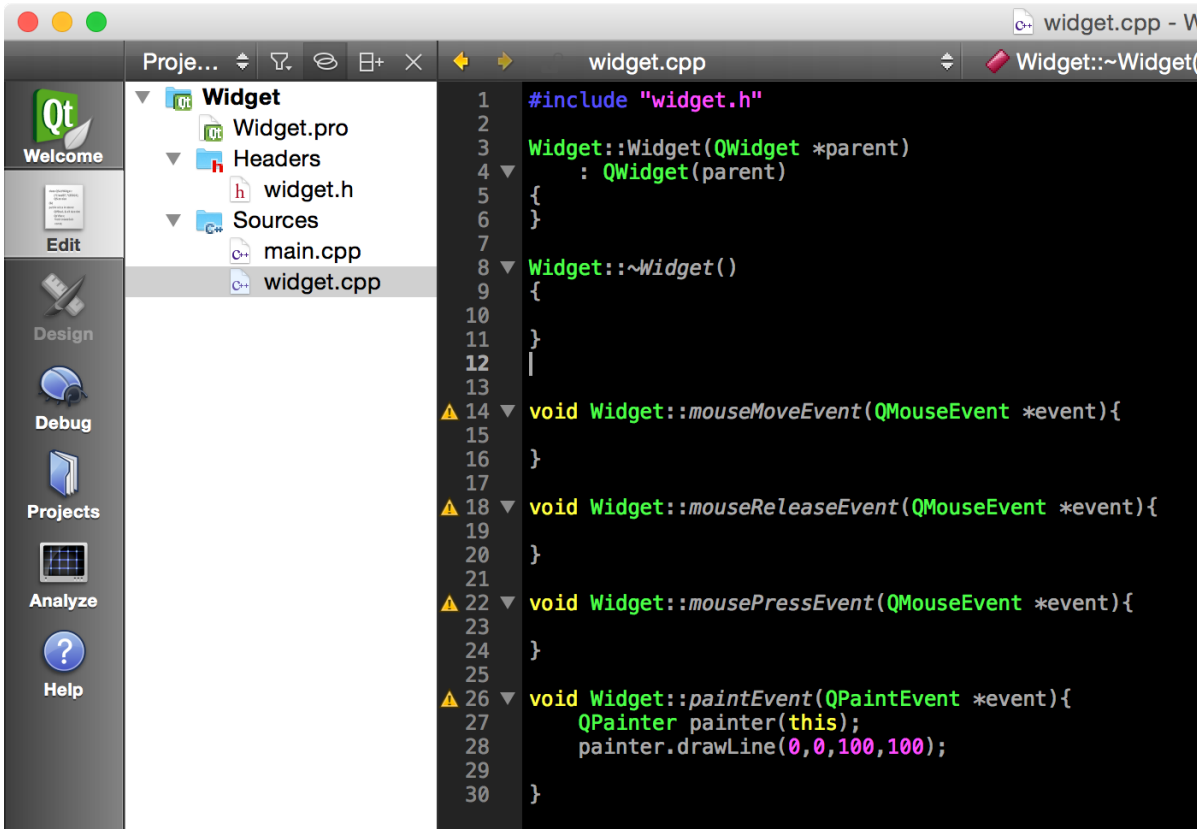
paintEvent 函数负责响应窗口的绘制事件。你应该在这个函数画你想要的东西，一种方案是：

在 paintEvent 函数中声明一个 QPainter 的对象。（你可以把 QPainter 看成是一个画笔，这个画笔需要关联一个“画布”，由于是在窗口上绘制，因此窗口就是这个画布，你也可以建立一个 QImage 对象，即一个图像，然后把画笔和这个 QImage 对象关联起来）。

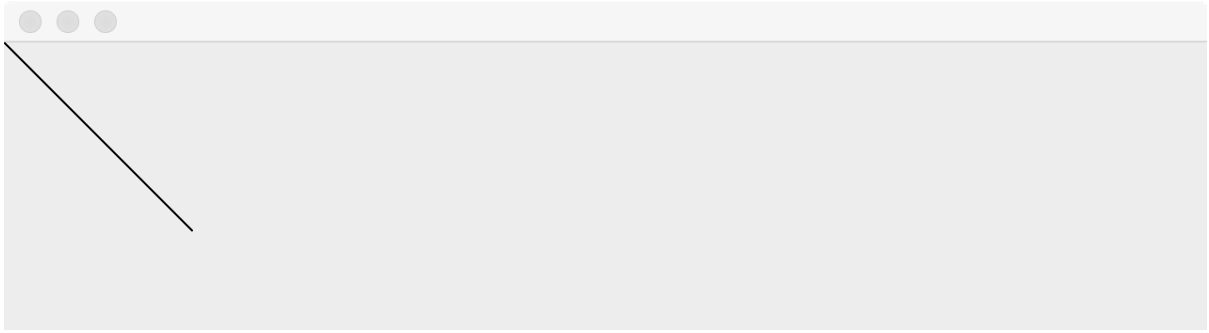
```
void Widget::paintEvent(QPaintEvent *event){
    QPainter painter(this);
    painter.drawLine(0,0,100,100);

}
```

上述代码将在窗口中画一条线从(0,0)到(100,100)的直线



运行结果如下图所示



另一种方案（推荐）是，在将要画的内容用 QPainter 的对象画在一个 QImage 对象上，然后在 paintEvent 函数中用 QPainter 把 QImage 画在窗口上。

```
void drawLine(QImage &image, QPoint pt1, QPoint pt2){
    QPainter painter(&image);    //新建一个QPainter对象，和image关联
    painter.drawLine(pt1, pt2); //用painter在image上画线段
}

void Widget::paintEvent(QPaintEvent *event){
    QPainter painter(this);
    QImage image = QImage(800, 600, QImage::Format_RGB888);//建立一个宽800，高600，24位的3通道RGB图像
    image.fill(QColor(255,255,255)); //将图像背景色刷为白色
    drawLine(image, QPoint(0,0), QPoint(100,100));  //调用非成员函数drawLine，在image上画线段pt1,pt2
    painter.drawImage(0,0,image); //用和widget对象关联的painter，将image画在窗口上。

}
```

效果如下：

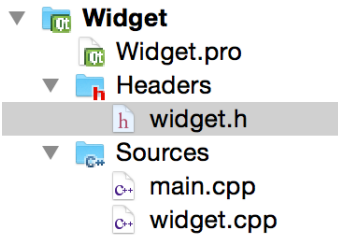
2.1.5 实现鼠标按下操作响应函数

当鼠标按下时，会触发 mousePress 事件，如果你的 Widget 类重新写了 void mousePressEvent 函数，那么这个函数就会被调用。你可以在这个函数里完成你想要做的事情。函数会接受一个参数 event，这个 event 里面就包含了所有当这个函数被触发时，鼠标以及键盘的所有状态。

比如可以记录鼠标按下时的位置信息，鼠标的哪个键被按下了等。

以下代码就记录了鼠标按下时的位置，将其存放在了成员变量 QPoint startPt 中.

在.h 文件中声明两个成员变量 QPoint startPt, endPt;




```
#ifndef WIDGET_H
#define WIDGET_H

#include <QtWidgets>
#include <QWidget>


class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = 0);
    ~Widget();

protected:
    void paintEvent(QPaintEvent *);
    void mousePressEvent(QMouseEvent *);
    void mouseMoveEvent(QMouseEvent *);
    void mouseReleaseEvent(QMouseEvent *);

private:
    QPoint startPt, endPt;
};

#endif // WIDGET_H
```

在.cpp 文件中做如下修改:

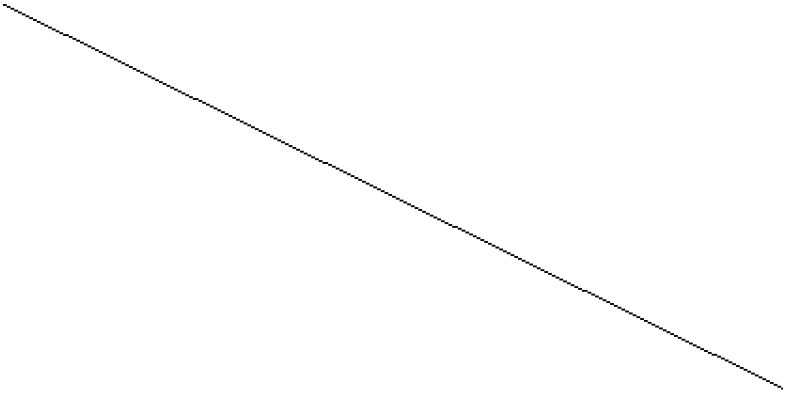
```
14  void Widget::mouseMoveEvent(QMouseEvent *event){
15
16  }
17
18  void Widget::mouseReleaseEvent(QMouseEvent *event){
19
20  }
21
22  void Widget::mousePressEvent(QMouseEvent *event){
23      startPt = event->pos();
24  }
```

2.1.6 实现鼠标抬起和鼠标移动操作操作函数

和 mousePressEvent 函数类似，用来响应鼠标释放事件。我们在 mouseMoveEvent 和 mouseReleaseEvent 中添加如下代码，并修改 paintEvent 中的画线函数，将 startPt 和 endPt 传入 drawLine 函数。

```
14  void Widget::mouseMoveEvent(QMouseEvent *event){
15      endPt = event->pos();
16      update();
17  }
18
19  void Widget::mouseReleaseEvent(QMouseEvent *event){
20      endPt = event->pos();
21      update();    //强制窗口重绘,即调用paintEvent函数
22  }
23
24  void Widget::mousePressEvent(QMouseEvent *event){
25      startPt = event->pos();
26  }
27
28  void drawLine(QImage &image, QPoint pt1, QPoint pt2){
29      QPainter painter(&image);     //新建一个QPainter对象，和image关联
30      painter.drawLine(pt1, pt2); //用painter在image上画线段
31  }
32
33  void Widget::paintEvent(QPaintEvent *event){
34      QPainter painter(this);
35      QImage image = QImage(800, 600, QImage::Format_RGB888);//建立一个宽800，高600，24位的3通道RGB图像
36      image.fill(QColor(255,255,255)); //将图像背景色刷为白色
37      drawLine(image, startPt, endPt);  //调用非成员函数drawLine，在image上画线段pt1,pt2
38      painter.drawImage(0,0,image); //用和widget对象关联的painter，将image画在窗口上。
39
40  }
```

效果如下：鼠标左键按下后，随着鼠标的移动，会在屏幕上画出一条黑色的直线。

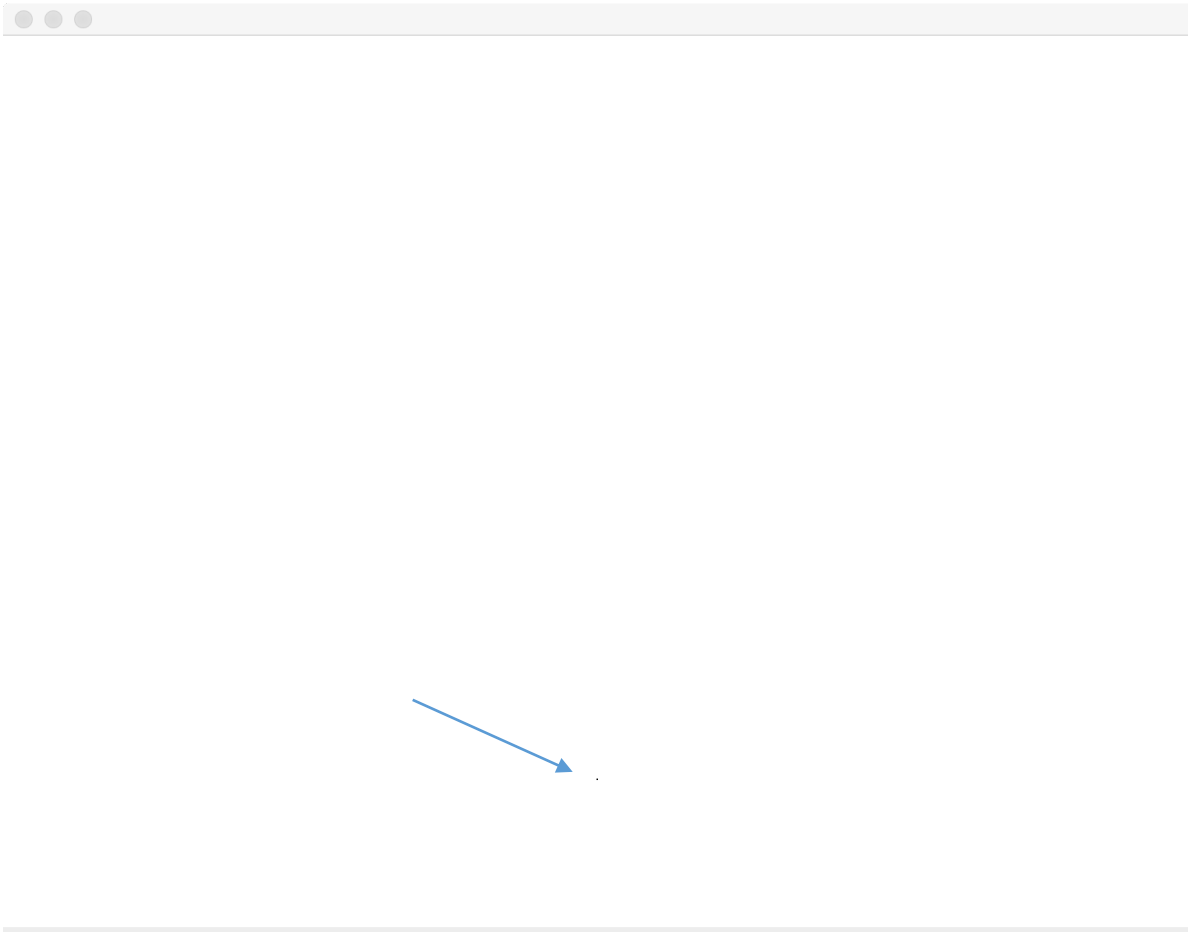


2.1.7 实现自己的画线函数

现在修改 drawLine 函数，不调用 QPainter 的 drawLine 成员函数，而是自己写一个 DDA 算法实现画线，你可以使用 painter 的 drawPoint 函数画点，也可以直接对 image 中的图像数据进行操作。

```
28 ▼ void drawLine(QImage &image, QPoint pt1, QPoint pt2){
29       QPainter painter(&image);    //新建一个QPainter对象，和image关联
30       //你自己的DDA算法
31
32       //算直线上每一个点的坐标,然后用painter.drawPoint(x,y);
33
34       //或者直接对图像数据进行操作,方法是
35       int x = 400, y = 500;   //表示图像的第(x,y)像素，即数组中第y行第x列的元素，
36                   //由于每个像素是rgb三通道，每个通道8位，因此在同
37                   //一行上每移动一个像素，指针位置应该移动3个8位的长度(char)
38                   //
39       unsigned char *ptrRow = image.scanLine(y); //scanLine函数返回图像第y行像素的首地址
40       ptrRow[x * 3 + 0] = 0;  //(x,y)像素的r通道
41       ptrRow[x * 3 + 1] = 0;  //(x,y)像素的g通道
42       ptrRow[x * 3 + 2] = 0;  //(x,y)像素的b通道 (0,0,0)是黑色
43
44   }
```

效果如下



2.2 命令行形式

该方式需要你已经写好了应用程序的最基本代码。一般来说就是 main.cpp，然后是你的主窗口类程序 xxx.cpp， xxx.h。例如对于 2.1 中的示例,假设我写好了同样的 main.cpp, widget.h, widget.cpp 文件。

main.cpp

```cpp
#include "widget.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Widget w;
    w.show();

    return a.exec();
}
```

同样的 widget.h

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QtWidgets>
#include <QWidget>


class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = 0);
    ~Widget();

protected:
    void paintEvent(QPaintEvent *);
    void mousePressEvent(QMouseEvent *);
    void mouseMoveEvent(QMouseEvent *);
    void mouseReleaseEvent(QMouseEvent *);

private:
    QPoint startPt, endPt;
};

#endif // WIDGET_H
```

同样的 widget.cpp

```cpp
#include "widget.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
}

Widget::~Widget()
{

}

void Widget::mouseMoveEvent(QMouseEvent *event){
    endPt = event->pos();
    update();
}

void Widget::mouseReleaseEvent(QMouseEvent *event){
    endPt = event->pos();
    update();    //强制窗口重绘,即调用paintEvent函数
}

void Widget::mousePressEvent(QMouseEvent *event){
    startPt = event->pos();
}

void drawLine(QImage &image, QPoint pt1, QPoint pt2){
    QPainter painter(&image);    //新建一个QPainter对象，和image关联
    painter.drawLine(pt1, pt2);
}

void Widget::paintEvent(QPaintEvent *event){
    QPainter painter(this);
    QImage image = QImage(800, 600, QImage::Format_RGB888);//建立一个宽800，高600，24位的3通道RGB图像
    image.fill(QColor(255,255,255)); //将图像背景色刷为白色
    drawLine(image, startPt, endPt);  //调用非成员函数drawLine，在image上画线段pt1,pt2
    painter.drawImage(0,0,image); //用和widget对象关联的painter，将image画在窗口上。
}
```

打开 vs2010 的 64 位命令行窗口，进入到你的工程文件所在目录，会看到如下文件夹结构。Windows 观众请用 dir 代替 ls:

```
Last login: Sun May  7 22:22:36 on ttys001
Dings-MacBook-Pro:Widget Dmm$ ls
main.cpp        widget.cpp      widget.h
Dings-MacBook-Pro:Widget Dmm$
```

然后依次键入:

```
qmake -project      //这一步产生 Qt 的工程文件（.pro）
```

```
Dings-MacBook-Pro:Widget Dmm$ qmake -project
Dings-MacBook-Pro:Widget Dmm$ ls
Widget.pro      main.cpp        widget.cpp      widget.h
Dings-MacBook-Pro:Widget Dmm$
```

（这里，如果你用的是 qt 5.0 以上版本，那请你用记事本打开这个 pro 文件，在里面添加这样一句话，如下图所示

```
QT += gui core widgets
```

```
#############################################################################
# Automatically generated by qmake (3.0) Mon May 8 00:33:34 2017
#############################################################################

QT += core gui widgets

TEMPLATE = app
TARGET = Widget
INCLUDEPATH += .

# Input
HEADERS += widget.h
SOURCES += main.cpp widget.cpp
```

然后存盘退出。）

qmake　　//这一步产生编译器相关的工程文件(Makefile)

```
Dings-MacBook-Pro:Widget Dmm$ qmake
Dings-MacBook-Pro:Widget Dmm$ ls
Makefile        Widget.pro      main.cpp        widget.cpp      widget.h
Dings-MacBook-Pro:Widget Dmm$
```

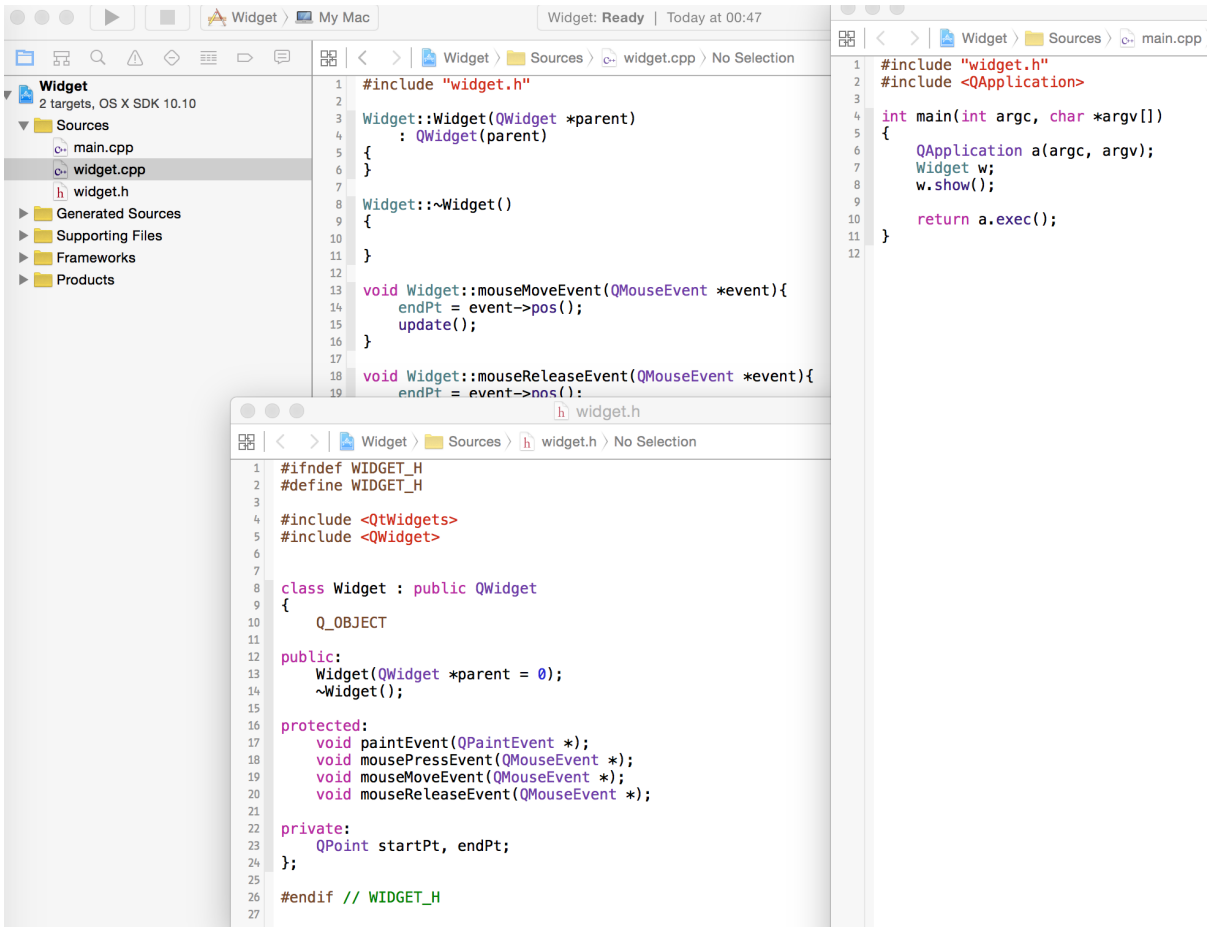
make（对于 vs 用户,请使用 nmake） //这一步以命令行的形式编译和链接程序.效果类似下图所示.

```
Dings-MacBook-Pro:Widget Dmm$ make
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/clang++ -c -pip
e -g -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/Mac
OSX10.10.sdk -mmacosx-version-min=10.6 -Wall -W -fPIE -DQT_WIDGETS_LIB -DQT_GUI_LIB -DQT_CORE_LIB -I/u
sr/local/Qt-5.3.2/mkspecs/macx-clang -I. -I. -I/usr/local/Qt-5.3.2/lib/QtWidgets.framework/Versions/5/
Headers -I/usr/local/Qt-5.3.2/lib/QtGui.framework/Versions/5/Headers -I/usr/local/Qt-5.3.2/lib/QtCore.
framework/Versions/5/Headers -I. -I/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platfor
m/Developer/SDKs/MacOSX10.10.sdk/System/Library/Frameworks/OpenGL.framework/Versions/A/Headers -I/Appl
ications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.10.sdk/System/
Library/Frameworks/AGL.framework/Headers -F/usr/local/Qt-5.3.2/lib -o widget.o widget.cpp
widget.cpp:32:38: warning: unused parameter 'event' [-Wunused-parameter]
void Widget::paintEvent(QPaintEvent *event){
                                     ^
1 warning generated.
/usr/local/Qt-5.3.2/bin/moc -DQT_WIDGETS_LIB -DQT_GUI_LIB -DQT_CORE_LIB -D__APPLE__ -D__GNUC__=4 -I/us
r/local/Qt-5.3.2/mkspecs/macx-clang -I/Users/Dmm/Project/Teaching/Computer_Graphics/Widget -I/Users/Dm
m/Project/Teaching/Computer_Graphics/Widget -I/usr/local/Qt-5.3.2/lib/QtWidgets.framework/Headers -I/u
sr/local/Qt-5.3.2/lib/QtGui.framework/Headers -I/usr/local/Qt-5.3.2/lib/QtCore.framework/Headers -I/Ap
plications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.10.sdk/usr/i
nclude/c++/4.2.1 -I/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs
/MacOSX10.10.sdk/usr/include/c++/4.2.1/backward -I/Applications/Xcode.app/Contents/Developer/Toolchain
s/XcodeDefault.xctoolchain/usr/lib/clang/6.0/include -I/Applications/Xcode.app/Contents/Developer/Tool
chains/XcodeDefault.xctoolchain/usr/include -I/Applications/Xcode.app/Contents/Developer/Platforms/Mac
OSX.platform/Developer/SDKs/MacOSX10.10.sdk/usr/include -F/usr/local/Qt-5.3.2/lib widget.h -o moc_widg
et.cpp
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/clang++ -c -pip
e -g -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/Mac
OSX10.10.sdk -mmacosx-version-min=10.6 -Wall -W -fPIE -DQT_WIDGETS_LIB -DQT_GUI_LIB -DQT_CORE_LIB -I/u
sr/local/Qt-5.3.2/mkspecs/macx-clang -I. -I. -I/usr/local/Qt-5.3.2/lib/QtWidgets.framework/Versions/5/
Headers -I/usr/local/Qt-5.3.2/lib/QtGui.framework/Versions/5/Headers -I/usr/local/Qt-5.3.2/lib/QtCore.
framework/Versions/5/Headers -I. -I/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platfor
m/Developer/SDKs/MacOSX10.10.sdk/System/Library/Frameworks/OpenGL.framework/Versions/A/Headers -I/Appl
ications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.10.sdk/System/
Library/Frameworks/AGL.framework/Headers -F/usr/local/Qt-5.3.2/lib -o moc_widget.o moc_widget.cpp
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/clang++ -header
pad_max_install_names -Wl,-syslibroot,/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.plat
form/Developer/SDKs/MacOSX10.10.sdk -mmacosx-version-min=10.6 -o Widget.app/Contents/MacOS/Widget main
.o widget.o moc_widget.o   -F/usr/local/Qt-5.3.2/lib -framework QtWidgets -framework QtGui -framework
QtCore -framework OpenGL -framework AGL
Dings-MacBook-Pro:Widget Dmm$ ls
Makefile        Widget.pro      main.o          moc_widget.o    widget.h
Widget.app      main.cpp        moc_widget.cpp  widget.cpp      widget.o
Dings-MacBook-Pro:Widget Dmm$
```

没有错误的情况下，就会在当前目录下的 Debug 目录里（或是当前目录下，取决于你是 windows 还是 mac 或 linux 用户）面得到你的应用程序 Widget.app 或是 widget.exe。

其中在第二步，你也可以通过 qmake -tp vc（或是 qmake -spec win32-msvc2010)来产生针对 vs2010 的工程文件，然后即可双击工程文件，回到你熟悉的 IDE 形式进行编程。对于 mac 环境，你可以通过 qmake -spec macx-xcode 来产生 xcode 格式的工程文件。

```
Dings-MacBook-Pro:Widget Dmm$ ls
Widget.pro      main.cpp        widget.cpp      widget.h
Dings-MacBook-Pro:Widget Dmm$ qmake -spec macx-xcode
Dings-MacBook-Pro:Widget Dmm$ ls
Info.plist              Widget.xcodeproj        widget.cpp
Widget.pro              main.cpp                widget.h
Dings-MacBook-Pro:Widget Dmm$
```



关于 qmake 的具体用法，也可通过在 assistant 中查询 qmake 来仔细阅读。