**整体框架：QMainWindow是为用户提供主窗口程序的类，包含菜单栏、工具栏、锚接部件、状态栏、中心部件等。完全依附于QMainWindow框架，根据框架来增添功能。**

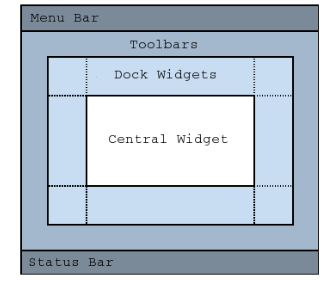
菜单栏：一系列的命令列表，为实现菜单栏、工具按钮、键盘快捷方式的一致性，Qt使用Action来表示这些命令。菜单就是动作的对象集合构成的列表。

状态栏：显示GUI应用程序的状态信息，用户能在状态上添加、使用Qt窗口部件。

工具栏：类似于按钮动作排列而成的面板，有经常使用的命令构成。

锚接部件：作为容器使用，以包容其他窗口部件来实现某些功能。

中心部件：为控制主窗体工具栏和锚接部件的显隐，在默认情况下，主窗体提供上下文菜单，通常在工具栏或锚接部件上单击鼠标即可激活该上下文菜单，或者重写QMainWindow::createPopupMenu()函数。



**①Qt窗口设计**

**1、随机或自定义生成复杂样式的验证码**

基于画布（背景），字体、干扰线（横穿字符）、倾斜度、字号缩放、颜色调配、随机间距（粘连）与及水平偏离（上下偏离）。

重写void paintEvent(QPaintEvent \*event)方法。参数：字符、字体、特征，从编辑框读取。

void drawOutline(QPainter &painter,int w, int h);

void drawCode(QPainter &painter, int nCodeIndex);

void drawNoisyLine(QPainter &painter,int w, int h);

void drawConversion(QPainter &painter);

**2、从文件浏览器导入图片并记录导入路径，默认为系统盘的图片。**

检查路径和操作权限，利用各窗体的路径匹配搜索是否打开过，若无，记录绝对路径，提取文件名作为标题，显示像素值与及支持图片缩放调整。原本是直接绘图，但绘图包含大量随机值，在保存成图片时要绘制在图片上，需保存大量参数。Qt绘图之QGraphicsScene，QGraphicsView，QGraphicsItem详解：场景、视图和坐标系统，采用双缓冲机制解决显示闪烁的问题。

QSettings类提供了持久的跨平台应用程序设置，用户通常期望应用程序记住它的设置所有会话。这些信息通常存储在Windows系统注册表。这里只记录了序号和图片的全路径。启动程序时会导入历史打开记录的信息。setAttribute(Qt::WA\_DeleteOnClose); //设置在主窗口关闭时销毁这个类的对象实例

**3、截屏功能的实现：**

设置主窗体隐藏，调用QGuiApplication::primaryScreen();将整个桌面视为QDialog窗体区域，重写mousePressEvent(QMouseEvent \* event)、void Dialog::mouseMoveEvent(QMouseEvent \* event)和void Dialog::mouseReleaseEvent(QMouseEvent \* event)方法。分别在按下左键时启动截屏，跟踪鼠标的移动，利用画笔生成动态矩形框，记录矩形框最终形态（起止坐标），在右键点击时判断选择的区域是否够大，将图片导入（和从文件浏览器导入一样），生成文件名标签，窗口标题加上\*，最后让主窗体显示出来。

**4、生成手写体绘制窗口**

void DrawDigit::mousePressEvent(QMouseEvent \*e)；//启动记录

void DrawDigit::mouseMoveEvent(QMouseEvent \*e)；

//跟踪鼠标的轨迹，当两点距离大于某个值（与画线粗细相关）才绘制，原本是想记录多个点来绘制光滑曲线，但无法解决出现抖动或断笔的问题。

**5、窗体风格、背景、图标、启动程序的动画、标题**

**②文本编辑**

**1、操作：撤销、重做、剪切、复制、粘贴。**

**2、调整：字体、字号、颜色、加粗、倾斜。**

创建QAction，包括图标，浮动和状态栏提示文本，设置快捷键，关联的槽与及在菜单上罗列出来。connect(Act, SIGNAL(triggered()), this, SLOT(Function ()));

调用QTextEdit的方法undo,redo,cut,copy,paste即可。

QColor col = QColorDialog::getColor(activeSubWindow()->getTextEdit()->textColor(), this);

comboFont = new QFontComboBox(); comboSize = new QComboBox();

QFontDatabase db;

foreach(int size, db.standardSizes())

comboSize->addItem(QString::number(size));

**3、查找、关闭、平铺、层叠、窗体列表和界面更新（多窗体视图的管理同步）**

使用QMdiArea类的activeSubWindow()函数来获得中心区域的活动子窗口，使用类型转换为子窗口类指针。利用各窗体的路径匹配搜索，Qt中提供了信号映射器QSignalMapper类。先创建信号映射器，将它的mapped()信号关联到活动窗口槽上去。窗体创建或切换时，将窗口菜单将要显示的信号关联到更新的菜单槽上去。更新菜单的函数先清空了所有窗口的菜单动作，再动态添加。

QList<QMdiSubWindow \*> windows = mdiArea->subWindowList();这里遍历了多窗体区域的各个子窗口，以它们的文件名为文本创建了动作，并添加到窗口菜单上去。根据是否有激活的窗口，激活的是否有文字被选中等情况更新各种文本编辑功能、图像处理功能能否使用。//以下功能不必自个实现

connect(closeAct, SIGNAL(triggered()),mdiArea, SLOT(closeActiveSubWindow()));

connect(closeAllAct, SIGNAL(triggered()),mdiArea, SLOT(closeAllSubWindows()));

connect(tileAct, SIGNAL(triggered()), mdiArea, SLOT(tileSubWindows()));

connect(cascadeAct, SIGNAL(triggered()), mdiArea, SLOT(cascadeSubWindows()));

connect(nextAct, SIGNAL(triggered()),mdiArea, SLOT(activateNextSubWindow()));

connect(previousAct, SIGNAL(triggered()),mdiArea, SLOT(activatePreviousSubWindow()));

更新窗体时：

QFont font = activeSubWindow()->getTextEdit()->currentFont();

comboFont->setCurrentFont(font);

QString sizenum = QString::number(font.pointSize());

comboSize->setCurrentIndex(comboSize->findText(sizenum));

**③图像处理**

**1、支持图片缩放、生成直方图**

QDialog \*hstgrmDialog = new QDialog(this);

QScrollArea \*scrollArea = new QScrollArea(hstgrmDialog);

Histogram \*hstgrm = new Histogram(scrollArea);

**2、灰度二值反色、边缘检测（sobel，prewitt，laplace）**

R’= G’= B’= 0.30R + 0.59G + 0.11B

void MainWindow::imageProcTriggered(QImage (\*handle)(const QImage &),const QString &message)；QImage proc::Binaryzation(const QImage &origin)；

// 根据rgb权重灰度化，ostu二值化

二值化操作可采用最大方差阈值法（Otsu法）来实现，注意要采用最佳的二分阈值，否则将会丢失特征点或残存干扰量，这里尝试阈值T = 0.67Gmax + 0.33 Gmin，G为像素灰度值。

laplace锐化窗口int window[3][3] = {0,-1,0,-1,4,-1,0,-1,0};

**3、基于字符图像的HOG特征和K最近邻分类算法识别。**

**①Histogram of Oriented Gridients，方向梯度直方图，缩写为HOG，是目前计算机视觉、模式识别领域很常用的一种描述图像局部纹理的特征。这个特征名字起的也很直白，就是说先计算图片某一区域中不同方向上梯度的值，然后进行累积，得到直方图，这个直方图呢，就可以代表这块区域了，也就是作为特征，可以输入到分类器里面了。**

**1.分割图像：**overlap和non-overlap，overlap指的是分割出的区块（patch）互相交叠，有重合的区域。non-overlap指的是区块不交叠，没有重合的区域。

2.计算每个区块的方向梯度直方图：

A.利用任意一种梯度算子，例如：sobel，laplacian等，对该patch进行卷积，计算得到每个像素点处的梯度方向和幅值。

B.将360度。根据需要分割成若干个bin，然后根据每个像素点的梯度方向，利用双线性内插法将其幅值累加到直方图中。

C.（可选）将图像分割成更大的Block，并利用该Block对其中的每个小patch进行颜色、亮度的归一化，这一步主要是用来去掉光照、阴影等影响的。

**3.组成特征**

将从每个patch中提取出的“小”HOG特征首尾相连，组合成一个大的一维向量，这就是最终的图像特征。可以将这个特征送到分类器中训练了。

CV\_WRAP HOGDescriptor(Size \_winSize, Size \_blockSize, Size \_blockStride,   Size \_cellSize,

 int \_nbins, int \_derivAperture=1, double \_winSigma=-1,

int \_histogramNormType,HOGDescriptor::L2Hys,  double \_L2HysThreshold=0.2, bool \_gammaCorrection=false,  int \_nlevels=HOGDescriptor::DEFAULT\_NLEVELS) ;

HOGDescriptor是HOG特征类，使用该类时，涉及到的参数有：ImageSize，winSize，blockSize，cellSize，blockStride， winStride。检测算法以4层嵌套循环在计算，每一层循环均为二维：

 用设置的winSize以winStride为单位遍历ImageSize ; 用设置的blockSize以blockStride为单位遍历winSize;用cellSize为单位遍历blockSize ; 遍历cellSize中的每一个像素。为了加快HOG提取的速度，HOG算法的作者使用了查表法，将pixData和blockData计算完成后存在内存中，在遍历过程中用到相应数据时，直接查表即可，这样可以省去大量重复计算的时间。

HOGDescriptor \*hog = new HOGDescriptor(Size(70, 90), Size(14, 18), Size(7, 9), Size(7, 9), 9);

vector<float> descriptors;

hog->compute(img, descriptors);

Mat dst(1, static\_cast<int>(descriptors.size()), CV\_32FC1, descriptors.data());

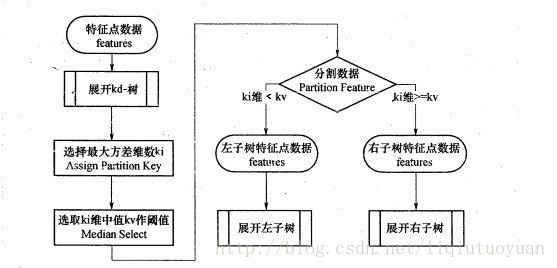
②kNN是一种基本的分类和回归方法。kNN的输入是测试数据和训练样本数据集，输出是测试样本的类别。kNN没有显示的训练过程，在测试时，计算测试样本和所有训练样本的距离，根据最近的K个训练样本的类别，通过多数投票的方式进行预测。从kNN的算法描述中可以发现，有三个元素很重要，分别是距离度量，k的大小和分类规则，这便是kNN模型的三要素。

11距离度量：要根据具体情况选择合适的距离度量方式。常用的是闵可夫斯基距离。

22 一般选择较小k并且k是奇数。通常采用交叉验证的方法来选取合适的k值。

33 kNN中的分类决策规则通常是多数表决，即由测试样本的k个临近样本的多数类决定测试样本的类别。

多数表决等价于经验风险最小化。所谓的kd树就是k个特征维度的树，注意这里的k和kNN中的k的意思不同。kNN中的k代表k个邻近样本，kd树中的k代表样本特征的维数。为了防止混淆，后面我们称特征维数为m。



1）计算已知类别数据集中的点与当前点之间的距离

2）按距离递增次序排序

3）选取与当前点距离最小的k个点

4）统计前k个点所在的类别出现的频率

5）返回前k个点出现频率最高的类别作为当前点的预测分类.

Ptr<ml::KNearest> kNearest;

Ptr<ml::KNearest> kNearest = ml::KNearest::create(); //KNN

kNearest->train(charROI, ml::ROW\_SAMPLE, charToRec);

Mat matRecChar(0, 0, CV\_32F);

kNearest->findNearest(dst, 1, matRecChar);

charRec = static\_cast<int>(matRecChar.at<float>(0, 0));

三、优化：

**1、多窗口的管理和同步：**

保证上下文环境的一致性，在更新窗口时会大量读取窗口的信息，设置布尔标签等，根据情况来设置当前的环境状态，包括窗口是否被修改，是否保存，文本框是否可被选中，选中后文字状态（加粗，颜色，字号等），与及对不同窗口要关联不同的信号和槽，与及启用对应的功能。

**2、关于鼠标绘制字符的光滑化处理。**

减少绘制取点。采用栈来记录点，并逐个用光滑的曲线来代替，抗锯齿，过大会出现抖动。断点处理，在绘制完毕后扫描取点，绘制光滑曲线。

**3、字符区域的锁定，干扰线的去除、粘连字符的分割，训练参数的调制。**

干扰背景的去除是首要步骤，若图像包含大量随机噪点，可用滤波等方法除去，如均值滤波、高斯滤波、中值滤波和双边滤波。这里对椒盐噪声推荐中值滤波，而高斯噪声采用双边滤波。为加快验证码和背景的分离，可采用最大方差阈值法（Otsu法）二值化图像。经过上述方法通常能获取到低干扰背景、骨架明朗的随机字符型二值图像，但通常无法去除粗细达到与字符等同的干扰线。为统一化字符和干扰线的线型粗细风格，可用固定线宽抽取图像骨架。所谓细化抽取骨架，即根据某点的若干近邻点的像素情况来判断该点是否能删除，使所有的字符、干扰线粗细较为一致。针对字符的分割操作，考虑采用漫水填充法和单方向像素投影法。漫水填充法对孤立不粘连的字符的分割极为有效。对于粘连的字符就要得采取像素投影的方案，在字符连接最为薄弱处切割字符。

难度很大的操作是在最大限度保证验证码特征点的条件下有效去除干扰线。为了去除干扰线，尝试使用霍夫变换。霍夫变换其实是把图像坐标空间转化到参数空间，实现直线的拟合。由于验证码中的加入的干扰直线往往具有一定的宽度，相当于多条参数极其接近的单像素宽直线，对应于参数空间中相邻的多个累加器。因此，当每找到一个当前最大的峰值点后，将该点及其附近点清零，以防算法检测出多条极其邻近的“假”直线，但单靠阈值还无法有效去除干扰线，还会使字符本身也模糊掉，这里还并不深入探讨如何去掉干扰线，特别是弥漫整个图片空间的弧形干扰线，只讨论噪点的去除工作。

为切割字符，采取漫水填充法和单方向（水平和垂直）像素投影法。

漫水填充法是一种用特定像素值来填充连通域的方法，可设置其填充范围达到不同的连通效果。用这种方法分割孤立不黏连的字符串的正确率很高，几乎不会出现分割错误的情况。但若字符粘连程度较大，就难以将其隔离开。对粘连的字符，则要从中切割开来。切割点如何选择呢？显然是在连接最为薄弱之处，但无法做到曲线切除，只能使用一刀切，难免会使两边的字符都切除不干净。

所谓水平和垂直投影法，即沿着特定方向扫描，获取该方向像素的投影分布、字符所在位置常意味着对应投影分布的某个峰值，波峰宽度为其所包含的区域。为准确字符串所在区域，可对投影进行累加和聚类，具体而言是按投影扫描方向的顺序，若某行的像素累加与上行的像素的累加和的比值小于给定的阈值，则将该像素累加和清零。统计归类后取连续不为零的累加和的宽度即为字符所在的位置。采取水平和垂直方向扫面可获得各字符的具体位置。