```
// 建構函式：利用 isZoomedWindow 區分身分
MainWindow::MainWindow(QWidget *parent, bool isZoomedWindow)
    : QMainWindow(parent), isZoomedWindow(isZoomedWindow) // 初始化成員變數，儲存此視窗是否為放大視窗
{
    isSelecting = false;     // 初始化：目前是否正在拖拉選取範圍（主視窗用）
    overlay = nullptr;       // 初始化：選取框的半透明遮罩層
    saveAsAction = nullptr;  // 初始化：另存新檔的動作（預設為空）

    // 初始化繪圖相關參數（放大視窗用）
    isDrawing = false;       // 初始化：目前是否正在畫線上
    penColor = Qt::red;      // 設定畫筆預設顏色為紅色
    penWidth = 5;            // 設定畫筆預設寬度為 5 像素

    // 如果是「主視窗」
    if (!isZoomedWindow)
    {
```

```
    // 在中央元件上建立一個用於繪製選取矩形的覆蓋層 (Overlay)
    overlay = new SelectionOverlay(central);
    // 設定覆蓋層大小與中央元件一致
    overlay->setGeometry(0, 0, central->width(), central->height());
    // 將覆蓋層提升到最上層，確保選取框不被圖片遮住
    overlay->raise();
    overlay->show();
  }

  // 根據視窗類型加載不同的選單與動作
  if (isZoomedWindow)
  {
    createZoomedWindowActions(); // 建立放大視窗專用的：存檔、顏色選擇
    createZoomedWindowMenus();  // 建立放大視窗專用的：檔案選單、畫筆選單
  }
  else
  {
```

```
    // 主視窗：建立完整的功能選單、工具列與其他初始化函數
    createActions1();
    createActions2();
    createMenus1();
    createMenus2();
    createToolBars();
    b();
    s();
  }
}
```

當圖片放大後，使用者需要儲存檔案或更換畫筆顏色

```
//-------------------------------------------------------------
// 建立放大視窗專用的動作（另存新檔、關閉、顏色切換）
void MainWindow::createZoomedWindowActions()
{
  // 建立「另存新檔」動作並綁定快速鍵 Ctrl+S
  saveAsAction = new QAction (QStringLiteral("另存新檔(&A)"),this);
  saveAsAction->setShortcut (tr("Ctrl+S"));
```

```
    connect (saveAsAction, SIGNAL (triggered()), this, SLOT (saveAsImage()));


    // 建立「關閉」動作並綁定快速鍵 Ctrl+Q
    exitAction = new QAction (QStringLiteral("關閉(&Q)"),this);
    exitAction->setShortcut (tr("Ctrl+Q"));
    connect (exitAction, SIGNAL (triggered()), this, SLOT (close()));


    // 建立各色畫筆動作，並透過 setData 儲存顏色資訊供後續讀取
    redPenAction = new QAction (QStringLiteral("紅色(&R)"),this);
    redPenAction->setData(QColor(Qt::red)); // 儲存顏色值
    connect (redPenAction, SIGNAL (triggered()), this, SLOT (setPenColor()));


    bluePenAction = new QAction (QStringLiteral("藍色(&B)"),this);
    bluePenAction->setData(QColor(Qt::blue));
```

```
    connect (bluePenAction, SIGNAL (triggered()), this, SLOT (setPenColor()));


    // ... (綠、黑、黃、白動作重複此邏輯)
}


// 將動作加入選單列
void MainWindow::createZoomedWindowMenus()
{
    fileMenu = menuBar ()->addMenu (QStringLiteral("檔案 &F")); // 建立檔案選單
    fileMenu->addAction(saveAsAction); // 加入另存新檔
    fileMenu->addAction(exitAction);   // 加入關閉


    // 在檔案選單下建立一個子選單「畫筆顏色」
    QMenu *penColorMenu = fileMenu->addMenu(QStringLiteral("畫筆顏色(&C)"));
    penColorMenu->addAction(redPenAction); // 加入各色選項
    penColorMenu->addAction(bluePenAction);
    // ... (加入其他顏色動作)
```

```
}
```

儲存與顏色設定邏輯

```
//-----------------------------------------------------------------------
--
void MainWindow::saveAsImage()
{
    if (img.isNull()) return; // 若無圖片則不執行

    // 彈出系統存檔視窗，讓使用者選擇路徑與格式
    QString filePath = QFileDialog::getSaveFileName(this,
QStringLiteral("另存新檔"), "", "PNG Files (*.png);;JPEG
Files (*.jpg)");

    if (!filePath.isEmpty())
    {
        if (img.save(filePath)) // 呼叫 QImage 的 save 函數存檔
            statusBar()->showMessage(QStringLiteral("檔案已成
功儲存"), 3000);
    }
}
```

```
void MainWindow::setPenColor()
{
    // 取得是哪一個選單選項觸發了這個函式
    QAction *action = qobject_cast<QAction*>(sender());
    if (action)
    {
        // 從該動作的 Data 中取出預存的顏色值
        penColor = action->data().value<QColor>();
        statusBar()->showMessage(QStringLiteral("畫筆顏色已更換"), 2000);
    }
}
```

滑鼠選取繪圖

```
//------------------------------------------------------------------
// 當滑鼠按下時
if (!isZoomedWindow && !img.isNull()) // 主視窗：開始準備選取
{
    isSelecting = true;
    selectionStart = event->pos(); // 紀錄起始座標
```

```
}
else if (isZoomedWindow && !img.isNull()) // 放大視窗：開始準備繪圖
{
    isDrawing = true;
    // 計算滑鼠在視窗內相對於圖片的位置（扣除選單與狀態列高度）
    int imgX = event->pos().x() - imgwin->geometry().x();
    int imgY = event->pos().y() - imgwin->geometry().y() - menuBar()->height() - statusBar()->height();
    lastDrawPoint = QPoint(imgX, imgY); // 紀錄繪圖起點
}
// 當滑鼠移動時
if (!isZoomedWindow && isSelecting) // 主視窗：更新選取框範圍
{
    selectionEnd = event->pos(); // 更新終點座標
    selectionRect = QRect(selectionStart, selectionEnd).normalized(); // 建立標準化矩形

    // 將視窗座標轉換為中央元件座標並交給 overlay 繪製
```

```
    QPoint centralTopLeft = central->mapFrom(this, selectionRect.topLeft());
    overlay->setSelectionRect(QRect(centralTopLeft, ...), true);
}
else if (isZoomedWindow && isDrawing) // 放大視窗：執行繪圖
{
    // 1. 計算目前的滑鼠座標
    QPoint currentPos = event->pos();
    // 2. 座標轉換：將視窗座標轉為圖片內部的實際像素座標（考慮縮放比例）
    double scaleX = (double)img.width() / imgwin->width();
    double scaleY = (double)img.height() / imgwin->height();

    int realX = (currentPos.x() - offset) * scaleX;
    int realY = (currentPos.y() - offset) * scaleY;

    // 3. 使用 QPainter 直接在 img (QImage) 上畫線
    QPainter painter(&img);
```

```
    painter.setPen(QPen(penColor, penWidth, Qt::SolidLine, Qt::RoundCap));
    painter.drawLine(lastRealX, lastRealY, realX, realY);


    // 4. 更新畫面顯示
    imgwin->setPixmap(QPixmap::fromImage(img));
    lastDrawPoint = currentPos; // 更新上一個繪圖點
}
// 當放開滑鼠時
if (!isZoomedWindow && isSelecting) // 主視窗：執行切圖並開啟新視窗
{
    isSelecting = false;
    // ... 略過座標計算邏輯 ...
    // 1. 根據選取範圍裁切圖片 (img.copy)
    QImage croppedImg = img.copy(imgX, imgY, imgW, imgH);
    // 2. 將裁切後的圖片放大兩倍
    QImage zoomedImg = croppedImg.scaled(imgW * 2, imgH * 2, Qt::KeepAspectRatio);
```

```
    // 3. 建立一個新的 MainWindow，參數帶入 true (標記為放大視窗)
    MainWindow *newIPWin = new MainWindow(nullptr, true);
    newIPWin->setWindowTitle(QStringLiteral("放大視窗 - 2x"));
    newIPWin->setImage(zoomedImg); // 將放大後的圖傳給新視窗
    newIPWin->show();


    overlay->setSelectionRect(QRect(), false); // 清除主視窗的選取框
}
```

視窗尺寸變動

```
//-----------------------------------------------------------------------
void MainWindow::resizeEvent(QResizeEvent * event)
{
    QMainWindow::resizeEvent(event); // 呼叫父類別處理
    // 如果是主視窗，則讓覆蓋層的大小跟隨中央元件變動
    if (!isZoomedWindow && overlay)
    {
```

```
        overlay->setGeometry(0, 0, central->width(), central->height());
    }
}
```