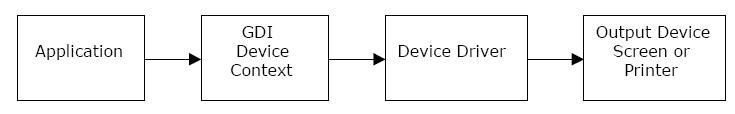
**GDI+编程基础（一）GDI+ Vs GDI**

**一、GDI**  
　　GDI是位于应用程序与不同硬件之间的中间层，这种结构让程序员从直接处理不同硬件的工作中解放出来，把硬件间的差异交给了GDI处理。GDI通过将应用程序与不同输出设备特性相隔离，使Windows应用程序能够毫无障碍地在Windows支持的任何图形输出设备上运行。例如，我们可以在不改变程序的前提下，让能在Epson点式打印机上工作的程序也能在激光打印机上工作。它把windows系统中的图形输出转换成硬件命令然后发送给硬件设备。GDI是以文件的形式存储在系统中，系统需要输出图形时把它载入内存，如果转换成硬件命令时遇到非GDI命令，系统还可能载入硬件驱动程序，驱动程序辅助GDI把图形命令转换成硬件命令。   
  
  
  
**二、设备环境**  
　　Windows系统是用来给应用程序提供设备独立性的工具，它是windows系统为了处理输出设备而使用的一种内部数据结构，设备环境是windos程序，驱动程序，和输出设备(如打印机，绘图仪)之间的纽带，GDI是一组C++类，它在驱动程序的协助下把数据描绘在硬件上，它位于应用程序与硬件之间，把数据从一方传到另一方。在Visual Studio .NET中Micro$oft解决了GDI中的许多问题，并让它变得易用，GDI的.net版本叫做GDI+。   
  
**三、GDI+**  
　　GDI+是GDI的下一个版本，它进行了很好的改进，并且易用性更好。GDI的一个好处就是你不必知道任何关于数据怎样在设备上渲染的细节，GDI+更好的实现了这个优点，也就是说，GDI是一个中低层API,你还可能要知道设备，而GDI+是一个高层的API，你不必知道设备。例如你如果要设置某个控件的前景和背景色，只需设置BackColor和ForeColor属性。   
  
**四、编程模式的变化**   
“GDI uses a stateful model, whereas GDI+ uses a stateless”——GDI是有状态的，GDI+是无无状态的。  
  
*1、不再使用设备环境或句柄*   
　　在使用GDI绘图时，必须要指定一个设备环境(DC)，用来将某个窗口或设备与设备环境类的句柄指针关联起来，所有的绘图操作都与该句柄有关。而GDI+不再使用这个设备环境或句柄，取而代之是使用Graphics对象。与设备环境相类似，Graphics对象也是将屏幕的某一个窗口与之相关联，并包含绘图操作所需要的相关属性。但是，只有这个Graphics对象与设备环境句柄还存在着联系，其余的如Pen、Brush、Image和Font等对象均不再使用设备环境。   
  
*2、Pen、Brush，Font，Image等对象是图形对象独立的*画笔对象能与用于提供绘制方法的图形对象分开创建于维护，Graphics绘图方法直接将Pen对象作为自己的参数，从而避免了在GDI使用SelectObject进行繁琐的切换，类似的还有Brush、Path、Image和Font等。   
  
*3、“当前位置”*GDI绘图操作(如画线)中总存在一个被称为"当前位置"的特殊位置。每次画线都是以此当前位置为起始点，画线操作结束之后，直线的结束点位置又成为了当前位置。设置当前位置的理由是为了提高画线操作的效率，因为在一些场合下，总是一条直线连着另一条直线，首尾相接。有了当前位置的自动更新，就可避免每次画线时都要给出两点的坐标。尽管有其必要性，但是单独绘制一条直线的场合总是比较多的，因此GDI+取消这个"当前位置"以避免当无法确定"当前位置"时所造成的绘图的差错，取而代之的是直接在DrawLine中指定直线起止点的坐标。 　　   
  
*4、绘制和填充*  
　　GDI总是让形状轮廓绘制和填充使用同一个绘图函数，例如Rectangle。轮廓绘制需要一个画笔，而填充一个区域需要一个画刷。也就是说，不管我们是否需要填充所绘制的形状，我们都需要指定一个画刷，否则GDI采用默认的画刷进行填充。这种方式确实给我们带来了许多不便，现在GDI+将形状轮廓绘制和填充操作分开而采用不同的方法，例如DrawRectangle和FillRectangle分别用来绘制和填充一个矩形。 　　   
  
*5、区域的操作*  
　　GDI提供了许多区域创建函数，如：CreateRectRgn、CreateEllpticRgn、CreateRoundRectRgn、CreatePolygonRgn和CreatePolyPolygonRgn等。诚然，这些函数给我们带来了许多方便。但在GDI+中，由于为了便于将区域引入矩阵变换操作，GDI+简化一般区域创建的方法，而将更复杂的区域创建交由Path接管。由于Path对象是与设备环境分离开来的，因而可以直接在Region构造函数中加以指定。   
  
**五、GDI+新特色**   
GDI+与GDI相比，增加了下列新的特性：  
  
*1、渐变画刷*  
　　以往GDI实现颜色渐变区域的方法是通过使用不同颜色的线条来填充一个裁剪区域而达到的。现在GDI+拓展了GDI功能，提供线型渐变和路径渐变画刷来填充一个图形、路径和区域，甚至也可用来绘制直线、曲线等。这里的路径可以视为由各种绘图函数产生的轨迹。 　　   
  
*2、样条曲线*对于曲线而言，最具实际意义的莫过于样条曲线。样条曲线是在生产实践的基础上产生和发展起来的。模线间的设计人员在绘制模线时，先按给定的数据将型值点准确地"点"到图板上。然后，采用一种称为"样条"的工具(一根富有弹性的有机玻璃条或木条)，用压铁强迫它通过这些型值点，再适当调整这些压铁，让样条的形态发生变化，直至取得合适的形状，才沿着样条画出所需的曲线。如果我们把样条看成弹性细梁，那么压铁就可看成作用在这梁上的某些点上的集中力。GDI+的Graphics:: DrawCurve函数中就有一个这样的参数用来调整集中力的大小。除了样条曲线外，GDI+还支持原来GDI中的Bezier曲线。   
  
*3、独立的路径对象*  
　　在GDI中，路径是隶属于一个设备环境(上下文)，也就是说一旦设备环境指针超过它的有效期，路径也会被删除。而GDI+是使用Graphics对象来进行绘图操作，并将路径操作从Graphics对象分离出来，提供一个GraphicsPath类供用户使用。这就是说，我们不必担心路径对象会受到Graphics对象操作的影响，从而可以使用同一个路径对象进行多次的路径绘制操作。  
  
*4、矩阵和矩阵变换*  
　　在图形处理过程中常需要对其几何信息进行变换以便产生复杂的新图形，矩阵是这种图形几何变换最常用的方法。为了满足人们对图形变换的需求，GDI+提供了功能强大的Matrix类来实现矩阵的旋转、错切、平移、比例等变换操作，并且GDI+还支持Graphics图形和区域(Region)的矩阵变换。  
  
*5、Alpha通道合成运算*  
　　在图像处理中，Alpha用来衡量一个像素或图像的透明度。在非压缩的32位RGB图像中，每个像素是由四个部分组成：一个Alpha通道和三个颜色分量(R、G和B)。当Alpha值为0时，该像素是完全透明的，而当Alpha值为255时，则该像素是完全不透明。 　　  
　　Alpha混色是将源像素和背景像素的颜色进行混合，最终显示的颜色取决于其RGB颜色分量和Alpha值。它们之间的关系可用下列公式来表示   
　　显示颜色 = 源像素颜色 X alpha / 255 + 背景颜色 X (255 - alpha) / 255   
GDI+的Color类定义了ARGB颜色数据类型，从而可以通过调整Alpha值来改变线条、图像等与背景色混合后的实际效果。  
  
*6、多图片格式的支持*  
　　GDI+提供了对各种图片的打开，存储功能。通过GDI+，我们能够直接将一幅BMP文件存储成JPG或其它格式的图片文件。  
　　除了上述新特性外，GDI+还将支持重新着色、色彩修正、消除走样、元数据以及Graphics容器等特性。   
  
**六、VC.net中使用GDI+的方法** 　  
在Visual C++.NET使用GDI+一般遵循下列步骤： 　　  
  
(1)、在应用程序中添加GDI+的包含文件gdiplus.h以及附加的类库gdiplus.lib。通常gdiplus.h包含文件添加在应用程序的stdafx.h文件中，而gdiplus.lib可用两种进行添加：第一种是直接在stdafx.h文件中添加下列语句：

#pragma comment( lib, "gdiplus.lib" )

　　另一种方法是：选择"项目->属性"菜单命令，在弹出的对话框中选中左侧的"链接器->输入"选项，在右侧的"附加依赖项"框中键入gdiplus.lib，   
  
(2)、在应用程序项目的应用类中，添加一个成员变量，如下列代码：

ULONG\_PTR m\_gdiplusToken;

其中，ULONG\_PTR是一个DWORD数据类型，该成员变量用来保存GDI+被初始化后在应用程序中的GDI+标识，以便能在应用程序退出后，引用该标识来调用Gdiplus:: GdiplusShutdown来关闭GDI+。  
  
(3)、在应用类中添加ExitInstance的重载，并添加下列代码用来关闭GDI+：

int CGDIPlusApp::ExitInstance()

{

　Gdiplus::GdiplusShutdown(m\_gdiplusToken);

　return CWinApp::ExitInstance();

}

(4)、在应用类的InitInstance函数中添加GDI+的初始化代码：

BOOL CGDIPlusApp::InitInstance()

{

　CWinApp::InitInstance();

　Gdiplus::GdiplusStartupInput gdiplusStartupInput;

　Gdiplus::GdiplusStartup(&m\_gdiplusToken, &gdiplusStartupInput, NULL);

　...

}

(5)、在需要绘图的窗口或视图类中添加GDI+的绘制代码：

void CGDIPlusView::onDraw(CDC \*pDC)

{

Graphics graphics( pDC->m\_hDC );

GraphicsPath path; // 构造一个路径

path.AddEllipse(50, 50, 200, 100);

// 使用路径构造一个画刷

PathGradientBrush pthGrBrush(&path);

// 将路径中心颜色设为蓝色

pthGrBrush.SetCenterColor(Color(255, 0, 0, 255));

// 设置路径周围的颜色为蓝芭，但alpha值为0

Color colors[] = {Color(0, 0, 0, 255)};

INT count = 1;

pthGrBrush.SetSurroundColors(colors, &count);

graphics.FillRectangle(&pthGrBrush, 50, 50, 200, 100);

LinearGradientBrush linGrBrush(

Point(300, 50),

Point(500, 150),

Color(255, 255, 0, 0), // 红色

Color(255, 0, 0, 255)); // 蓝色

graphics.FillRectangle(&linGrBrush, 300, 50, 200, 100);

}

注：本作者熟悉C++/MFC，熟悉GDI/GDI+,英文文档翻译，接做文档翻译，小型系统开发，技术难题突破等......Email：realman1981@sohu.com,电话：13679278016