**使用VC/MFC打印功能(Print)**

2018/7/24 15:36:08    来源：apple    阅读：218

第一部分和第二部分是自己的实现及解决方法，第三部分是参考的文章

一：（打印实现的操作流程）

    在VC中实现打印操作，网上虽然看了很多的说明或例子，但具体实现时还是感觉迷惑，自己研究了下，将经验总结如下，如有错误，欢迎大家指正，免得误人误己。

   自己实现的打印是在CView的派生类(如：CMyView)中，关于MFC内部实现，有很多文章都说明了，这里就不重复了，实现步骤如下：

**1**：在CMyView中定义个函数入void PrintView()，函数实现调用this->OnFilePrint();即可，以便开始执行打印操作(内部机制是CView实现)。

**2**：在CMyView中先后实现虚函数OnPreparePrinting, OnBeginPrinting, OnPrint, OnEndPrinting。

**3**：在OnPreparePrinting函数中必须调用DoPreparePrinting(pInfo);函数，可以return DoPreparePrinting(pInfo);就行。在DoPreparePrinting(pInfo);中，会调用打印的小dialog，如果在小dialog出现后点击了取消或关闭，会返回false，如果在这时候你再调用CView::OnPreparePrinting(pInfo);会出现中断，因为你没有设置完毕，DoPreparePrinting函数没有成功的时候要返回false以取消打印。即获取用户选择“打印对话框”的结果，如果在“打印对话框”出现后点击了取消或关闭，会返回FALSE，DoPreparePrinting函数没有成功的时候要返回FALSE以取消打印，如果点击“打印对话框”的“确定”按钮，则要返回TRUE进行打印。

**4**：在OnBeginPrinting函数中，设置最大页数，页眉，页脚等操作，如：pInfo->SetMaxPage(1); //设置最大页数为1页；

**5**：在OnPrint函数中，实现具体的绘图操作，将你要的image或文字信息绘到打印机DC上即可。注：在OnPrint函数中会有系统自动添加的一句话CView::OnPrint(pDC, pInfo);，这句话尽量屏蔽掉，因为在你绘图操作后再调用CView::OnPrint(pDC, pInfo);系统会调用OnPaint操作，会将你的View上显示的内容也绘在打印纸上（我这实现时需要只打印View上的一块内容，不想打印全部view，如果继续调用CView::OnPrint(pDC, pInfo);会出现打印重合的情况，也出现我不想出现的整个view的内容）。

6：在OnEndPrinting函数中，释放申请的资源等，如果在OnBeginPrinting函数中申请了画刷什么的等其他资源在OnEndPrinting函数中释放即可。

二：（图片多大尽量接近显示的大小，可以缩放但不能超出一张纸的大小，关于打印出的内容在打印纸上占用空间很小，如image打印出来后在打印纸上很小，只在打印纸上的左上角一小块地方）

    原因是打印机的DPI（dot per inch ，像素数）远远大于屏幕上的像素数比例，查看OnPrint(CDC\* pDC, CPrintInfo\* pInfo)中pInfo->m\_rectDraw，会发现其区域很大（0, 0, 4761, 6814），所以在你用pDC绘图时会发现你图片的1280\*1024到打印纸上只是很小的一块。在VC知识库中看到篇文章（**在VC中所见即所得打印的简易实现**），知道了解决方法，也进行了相应的修改以适应不是所谓800\*600的机器，因为客户的电脑不一定多大呢，解决方法如下：

1：将下列代码添加到CMyView::OnPrint函数中, 对于设定坐标模式也可以在void CMyView::OnPrepareDC中实现

|  |  |
| --- | --- |
|  | pDC->SetMapMode(MM\_ANISOTROPIC); //转化坐标映射模式  int nScreenWidth = GetSystemMetrics(SM\_CXSCREEN); //得到屏幕分辨率的宽和高(我的是1280\*1024)  int nScreenHeight = GetSystemMetrics(SM\_CYSCREEN);  CSize size = CSize(nScreenWidth, nScreenHeight);  pDC->SetWindowExt(size); //设置视窗大小  int xLogPixPerInch = pDC->GetDeviceCaps(LOGPIXELSX);  //得到设备每逻辑英寸的像素数量  int yLogPixPerInch = pDC->GetDeviceCaps(LOGPIXELSY);  float fWidth = (float)xLogPixPerInch / 96 ; //得到电脑屏幕映射到视窗大小比率  float fHeight = (float)yLogPixPerInch / 96; //一般得到的fWidth = fHeight  long xExt = (long)(fWidth \* size.cx); //得到视窗大小  long yExt = (long)(fHeight \* size.cy) ;  pDC->SetViewportExt((int)xExt, (int)yExt); //设置视窗大小(由电脑屏幕映射到视窗大小)    m\_pDoc->PrintDraw(pDC, pInfo, fWidth, fHeight); //m\_pDoc->PrintDraw是和当前View关联的Doc(存储数据等其他信 //息), fWidth和fHeight参数用于在绘图到打印机DC(pDC)时进行相应 //的计算,看是否超出了打印纸能打印区域(pInfo->m\_rectDraw)边界 |

2：CMyDoc::PrintDraw()函数实现如下(为防止代码外泄,代码经过了改动,只剩下有意义代码)：

|  |  |
| --- | --- |
|  | //  CBitmap saveBmp, \*pOldBmp = NULL;  CRect rectDraw; //得到view中要绘图的区域(可能派生自CScrollView,所以不一定小于  GetDrawRect(rectDraw); //屏幕大小)  //得到要进行缩放的比例  float fw = 0.0;  BOOL bw = FALSE;  float fh = 0.0;  BOOL bh = FALSE;  float fr = 1.0;  if ((long)(fWidth \* rectDraw.Width()) > pInfo->m\_rectDraw.Width()) //如果长度由屏幕映射到打印机后大于打印机的可绘图区域  { //(pInfo->m\_rectDraw), 则需要进行缩放操作,缩放后不是图形变形  fw = fWidth \* rectDraw.Width() / pInfo->m\_rectDraw.Width();  bw = TRUE;  }  if ((long)(fHeight \* rectDraw.Height()) > pInfo->m\_rectDraw.Height()) //同上  {  fh = fHeight \* rectDraw.Height() / pInfo->m\_rectDraw.Height();  bh = TRUE;  }  if (bw || bh) //缩放比率应使纸张能容纳下图片大小,且尽可能的按原图大小显示  fr = fw > fh ? fw : fh;  //Recalculate rect of draw  RecalculateRect(rectDraw, fr, rectDraw); //RecalculateRect函数是重新计算应缩放后的rect区域,以便使打印 //的东西不超出打印纸的可打印范围(pInfo->m\_rectDraw)  // Create temp bmp  CDC tmpMemDC;  tmpMemDC.CreateCompatibleDC(m\_memDC);  if( !saveBmp.CreateCompatibleBitmap(&m\_memDC,  rectDraw.Width(),  rectDraw.Height()))  goto error\_return;  pOldBmp = pDC->SelectObject(&saveBmp);  // Paint background  pDC->FillRect(rectDraw, &CBrush(RGB(255, 255, 255)));  // Copy object to save bitmap  POSITION pos = m\_objects.GetHeadPosition(); //我这里需要将一张图片中的几个不确定区域的图片打印出来  while(pos != NULL)  {  CDrawImageObj\* pObj = (CDrawImageObj\*)m\_objects.GetNext(pos);  CRect rectSrc = PToRate(pObj->m\_position, TRUE);  CRect rectDst = rectSrc;  //Recalculate rect of every object  RecalculateRect(rectSrc, fr, rectDst);  CBitmap\* pTmpOldBmp = pTmpOldBmp = tmpMemDC.SelectObject(&pObj->GetBitmap());  // pDC->BitBlt(rect.left, rect.top, rect.Width(), rect.Height(), &tmpMemDC, 0, 0, SRCCOPY); //缩放的话不能使用BitBlt  pDC->StretchBlt(rectDst.left, rectDst.top, rectDst.Width(), rectDst.Height(),  //将原区域图片缩放到计算后的rect  &tmpMemDC, 0, 0, rectSrc.Width(), rectSrc.Height(), SRCCOPY); //中,这样就不会超出打印纸的大小了  tmpMemDC.SelectObject(pTmpOldBmp);  }  if (pOldBmp != NULL)  pDC->SelectObject(pOldBmp);  tmpMemDC.DeleteDC();  saveBmp.DeleteObject();  return TRUE;  error\_return:;  if (pOldBmp != NULL)  pDC->SelectObject(pOldBmp);  tmpMemDC.DeleteDC();  saveBmp.DeleteObject();  return FALSE; |

3：下面是重新计算函数，以使屏幕绘图的总大小(rectDraw)，各个图片区域(rectSrc)进行合理转化,rect区域位置需要进行定位,其大小也要重新计算：

|  |  |
| --- | --- |
|  | void CMyDoc::RecalculateRect(const CRect& rectSrc, float fRate, CRect& rectOut)  {  rectOut = rectSrc;  if (fRate == 0.0)  return ;  rectOut.left = (long)((float)rectSrc.left / fRate);  rectOut.top = (long)((float)rectSrc.top / fRate);  rectOut.right = (long)((float)rectSrc.right / fRate);  rectOut.bottom = (long)((float)rectSrc.bottom / fRate);  } |

三：参考文章如下：

1：在VC中所见即所得打印的简易实现（<http://www.vckbase.com/bbs/prime/viewprime.asp?id=326>**）**

　　在VC++6.0中用应用程序向导（AppWizard）生成的单文档或多文档程序提供了对打印功能的实现，但遗憾的是如果对自动生成的框架程序不做任何改进，打印出来的文档或图形和屏幕上的显示相比就会特别小。为什么会这样呢？  
  
　　本文对这种现象的原因和MFC的打印机制进行了深入的分析，并提出了一种特别简单的方法，在原有的程序中只需加入几行代码就能解决这一问题，实现所见即所得的打印。   
  
　　首先，分析MFC的打印机制，把原理弄清楚了，就不难明白现象形成的原因和提出解决办法。MFC应用程序的核心是文档对象以及相关的视图窗口的概念，即CDocument类和CView类的构成和关系，简单地说CDocument类负责数据的生成和储存，CView类负责数据的显示和用户交互。输出到屏幕和输出到打印机都是数据的显示，实质上是一样的，所以打印功能也是由CView类来实现的。   
  
　　在CView类中由应用程序向导自动生成的源代码提供了一个OnDraw(CDC\* pDC)的函数，通过重载这个函数，利用它提供的pDC（设备上下文）指针，可以在屏幕上显示各种图形和数据。CView类的打印是通过OnPrint(CDC\* pDC, CPrintInfo\* pInfo)这个函数实现的，应用程序向导自动生成的源代码中没有这个函数的框架，而这个函数对打印的实现就是简单地调用OnDraw(CDC\* pDC)这个函数，把打印机的设备上下文指针pDC传递给OnDraw(CDC\* pDC)函数。   
  
　　可见CView类对输出到屏幕和输出到打印机的处理都是一样的，只是换了一个设备上下文而已，那么为什么输出到打印机的图像特别小呢？   
  
　　这与VC采用的缺省的坐标映射方式MM\_TEXT有关，这种方式的好处是用户图形坐标和设备的象素完全一致。但是在屏幕的象素大小为800\*600时，每逻辑英寸包含的屏幕象素为96，而打印机的点数却要多好几倍，如当打印机为HP LaserJet 6L时每逻辑英寸包含的打印机点数为600，也就是说打印机的清晰度比屏幕要高得多。   
  
　　这样的后果就是在屏幕上显示出来的满屏图像在打印出来的纸上却只有一点点大，怎么解决这个问题呢？一种简单的方法就是转换坐标映射方式，使得打印时采用的坐标比例比显示时采用的坐标比例相应地大若干倍，就可以解决这一问题。   
  
　　下面将给出详细的方法。  
  
　　注意到CView类在进行显示和打印之前都会调用virtual void OnPrepareDC( CDC\* pDC, CPrintInfo\* pInfo = NULL )这个虚拟成员函数来准备设备上下文，我们可以在CView类中重载这个虚拟成员函数，进行坐标转换。   
  
　　首先用VC的ClassWizard实现对OnPrepareDC( CDC\* pDC, CPrintInfo\* pInfo = NULL )函数的重载，ClassWizard生成的源代码如下： 

|  |  |
| --- | --- |
|  | void CTempView::OnPrepareDC(CDC\* pDC, CPrintInfo\* pInfo)  {      // TODO: Add your specialized code here and /or call the base class      CView::OnPrepareDC(pDC, pInfo);  } |

　我们只需在源代码中加入以下几行代码即可，如下：

|  |  |
| --- | --- |
|  | void CPrintSameView::OnPrepareDC(CDC\* pDC, CPrintInfo\* pInfo)  {      // TODO: Add your specialized code here and /or call the base class      pDC->SetMapMode(MM\_ANISOTROPIC); //转换坐标映射方式  　　CSize size = CSize(800, 560);  　　pDC->SetWindowExt(size);    　　//确定窗口大小 //得到实际设备每逻辑英寸的象素数量 int xLogPixPerInch = pDC->  GetDeviceCaps(LOGPIXELSX);    　　int yLogPixPerInch = pDC- >GetDeviceCaps(LOGPIXELSY);    　　//得到设备坐标和逻辑坐标的比例 long xExt = (long)size.cx \* xLogPixPerInch/96 ;    　　long yExt = (long)size.cy \* yLogPixPerInch/96 ;  　　pDC->SetViewportExt((int)xExt, (int)yExt);    　　//确定视口大小        CView::OnPrepareDC(pDC, pInfo);  } |

　　如上所示，首先将坐标映射方式改变为MM\_ANISOTROPIC方式，即各向异性的意思，在这种坐标方式下，X轴和Y轴的逻辑单位可以进行任意的缩放。改变坐标映射方式后，就要确定窗口大小和视口大小，注意窗口大小就是我们在屏幕上所见的尺寸，而视口大小则是实际设备，如打印机等，和显示器设备每逻辑英寸的象素数量比较所得的比例尺寸。通过函数得到显示器和打印机每逻辑英寸的象素数量，然后对视口大小进行相应的缩放，就可以使得屏幕上的显示和打印机的输出是一致的了。   
  
　　这样，只通过几行简单的代码，我们就实现了所见即所得的打印。

**2：使用MFC编写打印程序（**<http://www.vckbase.com/document/viewdoc/?id=1618>**）**

**摘要**  
  
　　本文首先介绍了利用MFC提供的文档视图框架来实现一个打印程序，实现打印预览，在此基础上，同时通过对MFC源代码的深入探讨，提出了利用该方法在对话框上用MFC实现打印功能，结果表明，利用MFC实现打印不仅方便，而且功能很强大，能够根据不同的需求很方便的打印出所需要的格式。本文还实现了一个在对话框中利用MFC实现打印功能的一个框架结构，对于使用者只要使用该结构就可以按照自己的要求打印任何内容。  
  
关键词：Visual C++ ，MFC，对话框， 打印 ，打印预览  
  
**引言**  
  
　　打印程序的编写在windows程序设计中非常有用，针对不同的用户需要，通常用sdk方式实现打印代码量比较大，而且要对打印流程的底层有非常清楚的了解，需要一个程序员有非常深入的打印方面的知识，利用MFC提供的文档视图结构，不但可以实现一些常用的标准界面元素，把数据的处理的界面的处理分离出来，而且其提供的打印功能更是方便快捷，功能强大。打印程序的编写本质是是一种GDI绘图，只是绘图的对象是在打印机的设备描述表，如果对于屏幕的GDI绘图比较熟悉的读者，相信掌握打印程序的编写应该比较容易。  
  
**1、文档视图结构的程序的打印程序的编写**  
  
　　通常情况下，一个结构组织的比较好的MFC程序是基于文档视图结构的，这一框架结构给我们提供了很多功能，比如菜单，注册表的关联，文件类型的注册，打印功能，只要我们善于发掘，这些都可以为我们所用，但我们现在只关心如何使用MFC提供的结构来实现打印功能。  
在编写打印程序之前，有必要先介绍一下MFC的框架结构，其中的文档视图结构又是这个框架的重点，我们通过分析MFC实现的视图类的原代码就可以看到一个打印程序的执行流程。读者也可以看侯俊杰的《深入浅出MFC》,上面有关于MFC打印的详细流程解释,下面是MFC的打印的函数的实现，该函数名为OnFilePrint它不是一个虚函数，而是响应缺省的COMMAND消息的处理函数，因为MFC提供了向导生成的菜单和工具栏，关于打印的命令ID为ID\_FILE\_PRINT ，而在视图类的MessageMap里有这样一项，ON\_COMMAND (ID\_FILE\_PRINT, CView::OnFilePrint)，因此实际使用的过程中可以不用原来的ID, 而使用自己的ID如ID\_MYPRINT,再在MessageMap里加入ON\_COMMAND (ID\_MYPRINT, CView::OnFilePrint)即可完成原来一样的功能。ViewPrnt.cpp中有CView的OnFilePrint的函数的具体实现，ViewPrnt.cpp的位置读者自己用windows查找就能找到，这是MFC的源代码，本文把其中的主要代码列出放在下面，直接看下面的分析：

void CView::OnFilePrint()

{

    // get default print info

    if (OnPreparePrinting(&printInfo))

    {

        if (dlg.DoModal() != IDOK)

        {

            return;

        }

    }

}

　　其中加粗的代码行为可以重载的虚函数，根据不同的用户，其内容会不同。对于 OnPreparePrinting() 函数的具体内容必须有 return DoPreparePrinting(pInfo);这是在一个打印过程中最先调用的。当然也可以包含一些其它的打印初始化操作。我们最主要的是要重载三个函数：

OnBeginPrinting();

OnPrint();

OnEndPrinting();

　　而以 OnPrint 最为复杂，它是我们要写大量代码实现我们打印功能的地方。对于默认的OnPrint实现是调用CView的OnDraw，也就是和绘制视图类的客户区的内容完全相同的方法来在打印机上绘图。实际中我们在两种地方绘图的内容是完全不同的，可能用户在客户区绘的是一个曲线，而在打印机上要绘制表格和数据。OnPrint(CDC\* pDC, CPrintInfo\* pInfo)的第二个参数是一个CPrintInfo类型的指针，我们可以从这个指针指向的对象中获得很多信息，如总共的页数，当前的页数，这在打印页眉页脚时可能是很有用的信息。CPrintInfo的定义如下：

struct structCPrintInfo // Printing information structure

{

CPrintInfo();

~CPrintInfo();

CPrintDialog\* m\_pPD;     // pointer to print dialog

BOOL m\_bDocObject;       // TRUE if printing by IPrint interface

BOOL m\_bPreview;         // TRUE if in preview mode

BOOL m\_bDirect;          // TRUE if bypassing Print Dialog

BOOL m\_bContinuePrinting;// set to FALSE to prematurely end printing

UINT m\_nCurPage;         // Current page

UINT m\_nNumPreviewPages; // Desired number of preview pages

CString m\_strPageDesc;   // Format string for page number display

LPVOID m\_lpUserData;     // pointer to user created struct

CRect m\_rectDraw;        // rectangle defining current usable page area

// these only valid if m\_bDocObject

UINT m\_nOffsetPage;      // offset of first page in combined IPrint job

DWORD m\_dwFlags;         // flags passed to IPrint::Print

void SetMinPage(UINT nMinPage);

void SetMaxPage(UINT nMaxPage);

UINT GetMinPage() const;

UINT GetMaxPage() const;

UINT GetFromPage() const;

UINT GetToPage() const;

UINT GetOffsetPage() const;

};

　　OnBeginPrinting()通常用来设定要打印的总页数，以及一些和页面尺寸有关的初始化工作，在OnBeginPrinting()中设定打印的页数是必要的，默认的页数是只有一页，如果开发人员打印的页数大于1，则必须在此函数中设定打印的页数。然后在OnPrint(CDC\* pDC, CPrintInfo\* pInfo)中用pInfo-> m\_nCurPage获取当前的页码，根据当前的页码打印该页相应的内容。OnEndPrinting用来释放在OnBeginPrinting中申请的资源，如果没有申请，则不需重载该函数。  
　　关于打印预览只需要将自己的执行打印预览功能的命令ID和CView::OnFilePrintPreview关联起来就行了，具体方法是在用户的视图类的MessageMap中加入：

ON\_COMMAND(ID\_FILE\_PRINT\_PREVIEW, CView::OnFilePrintPreview);

其中ID\_FILE\_PRINT\_PREVIEW是默认的ID，开发人员也可以使用自己的ID。其实只要重载了OnPrint函数，在打印和打印预览中就可以重用该函数了。到现在为止，相信读者已经对利用MFC的文档视图结构来实现一个包含打印和打印预览功能的程序有了一个总体的认识了，本文还针对该方法给出了一个示例代码，代码来自Jeff Prosise 的《MFC windows程序设计》，见参考文献[1]。  
  
**2、没有文档视图结构的程序中利用MFC进行打印程序的编写**  
  
　　如果程序不是文档视图结构的，我们要使用MFC来进行打印，则可以通过建立一个虚拟的文档视图结构来进行打印，其实MFC的打印的强大功能是在CView里提供的，而CView类的对象是一个子窗口，它必须是某一个框架窗口的子窗口，而在对话框程序中，我们只需要打印，而不需要显示这个框架窗口和视图。我们以按下按钮"打印"来执行打印程序，例如按钮为ID为IDC\_PNT，消息相应函数为OnPnt()，即：

ON\_BN\_CLICKED(IDC\_PNT, OnPnt)；

　　需要在OnPnt中建立一个框架窗口，同时使某个CView类的对象为该窗口的子窗口。因此笔者建立了两个类，一个为框架窗口类CPrintFrame，另一个为CPrintView，具体的内容见示例代码。在新建一个用于打印的虚拟框架窗口时，需要将执行打印的对话框的指针传给框架窗口，这以便在对话框来响应WM\_BEGIN\_PRINTING和WM\_END\_PRINTING消息，使对话框可以完成打印的初始化和释放操作。在执行一个打印任务时，将打印的流程交给CView来进行，而这个CView是虚拟的，只是用来完成打印一些操作，其它内容则完全不负责处理，而当要执行CView::OnPrint时，则又将处理的具体内容传回到对话框，而对话框则只需要响应WM\_MY\_PRINT消息即可：

pFrame->m\_pCallerDlg->SendMessage(WM\_MY\_PRINT,(WPARAM) pDC, (LPARAM) pInfo);

　　使打印的具体处理又传回到对话框中，使开发人员根据具体的需要写WM\_MY\_PRINT的处理函数就可以实现打印，而CView::OnPrint(CDC\* pDC, CPrintInfo\* pInfo)的参数也从WM\_MY\_PRINT的消息参数传出来，在用户的对话框程序中,需要写的代码就很少，主要有以下几个步骤，

1. 建立一个CPrintFrame的对象,设该对象的指针为pFrame，并将对话框的指针传给该对象的m\_pCallerDlg，即pFrame->m\_pCallerDlg = this；
2. 调用对象的Create函数创建框架窗口；例如pFrame->Create(NULL,"频谱打印",WS\_OVERLAPPEDWINDOW,CRect(0,0,0,0));
3. 如果要执行打印，则调用pFrame->m\_pView->OnMyPrint();
4. 如果要执行打印预览，则调用：

pFrame->m\_pView->OnMyPrintPreview();

例如：

void CDlgPrintDlg::OnPrint() //执行打印功能

{

CPrintFrame \*pFrame = new CPrintFrame;

pFrame->m\_pCallerDlg = this;

pFrame->Create(NULL,"Curve

Print",WS\_OVERLAPPEDWINDOW,CRect(0,0,0,0));

pFrame->m\_pView->OnMyPrint();

}

void CDlgPrintDlg::OnPrintPreview() //执行打印预览功能

{

CPrintFrame \*pFrame = new CPrintFrame;

pFrame->m\_pCallerDlg = this;

pFrame->Create(NULL,"Curve Print

Preview",WS\_OVERLAPPEDWINDOW,CRect(0,0,0,0));

pFrame->m\_pView->OnMyPrintPreview();

}

1. 在对话框中响应 WM\_BEGIN\_PRINTING， WM\_END\_PRINTING，WM\_MY\_PRINT消息，分别完成打印的初始化，释放和具体的打印操作；

如在示例程序中添加了三个消息响应函数来执行该功能。

ON\_MESSAGE(WM\_BEGIN\_PRINTING,OnBeginPrinting)

ON\_MESSAGE(WM\_END\_PRINTING,OnEndPrinting)

ON\_MESSAGE(WM\_MY\_PRINT,OnMyPrint)

　　其中OnMyPrint是跟具体要打印什么内容有关的开发人员要重点完成的代码，可以打印表格，图片，数据，只要GDI绘图可以进行的操作在这里都可以完成。由于打印预览的一部分工作在CView类里完成，因此在用户程序中只需要相应WM\_MY\_PRINT消息就可以执行打印预览的功能，而不需要另外编写打印预览代码。  
　　本文提供的CPrintFrame和CPrintView类是连个可重用的类，开发者只需要把这两个类对应的四个文件拷贝到工程文件所在目录中(PrintFrame.h, PringtView.h,PrintFrame.cpp, PrintView.cpp)，并将这四个文件加入工程，并在需要执行打印功能的代码处加入

#include "PrintFrame.h"

#include "PrintView.h"

　　然后按照上述5个步骤进行即可以实现一个功能完整的打印程序，利用上述类实现对话框打印不但节省开发者许多时间，而且功能很强大，能达到很专业的水平，但是该方法有一个缺点，笔者发现如果开发者使用静态连接的MFC库时则会出错，只适用于Use MFC in a Shelled DLL情况，而且必须使程序为Debug版本的。  
  
**3、示例代码的执行效果**

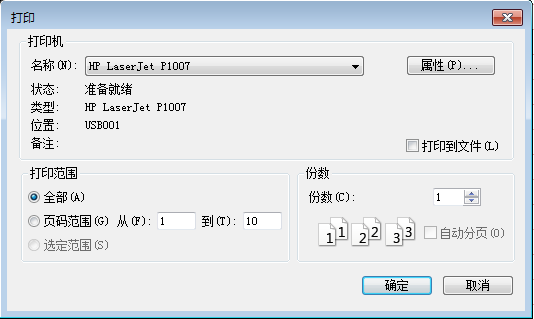
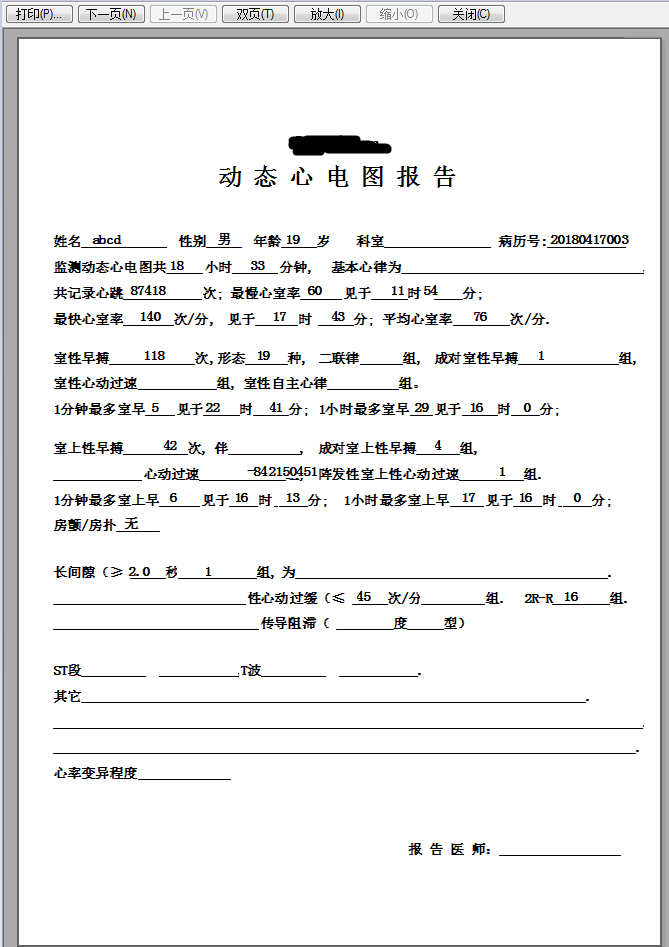
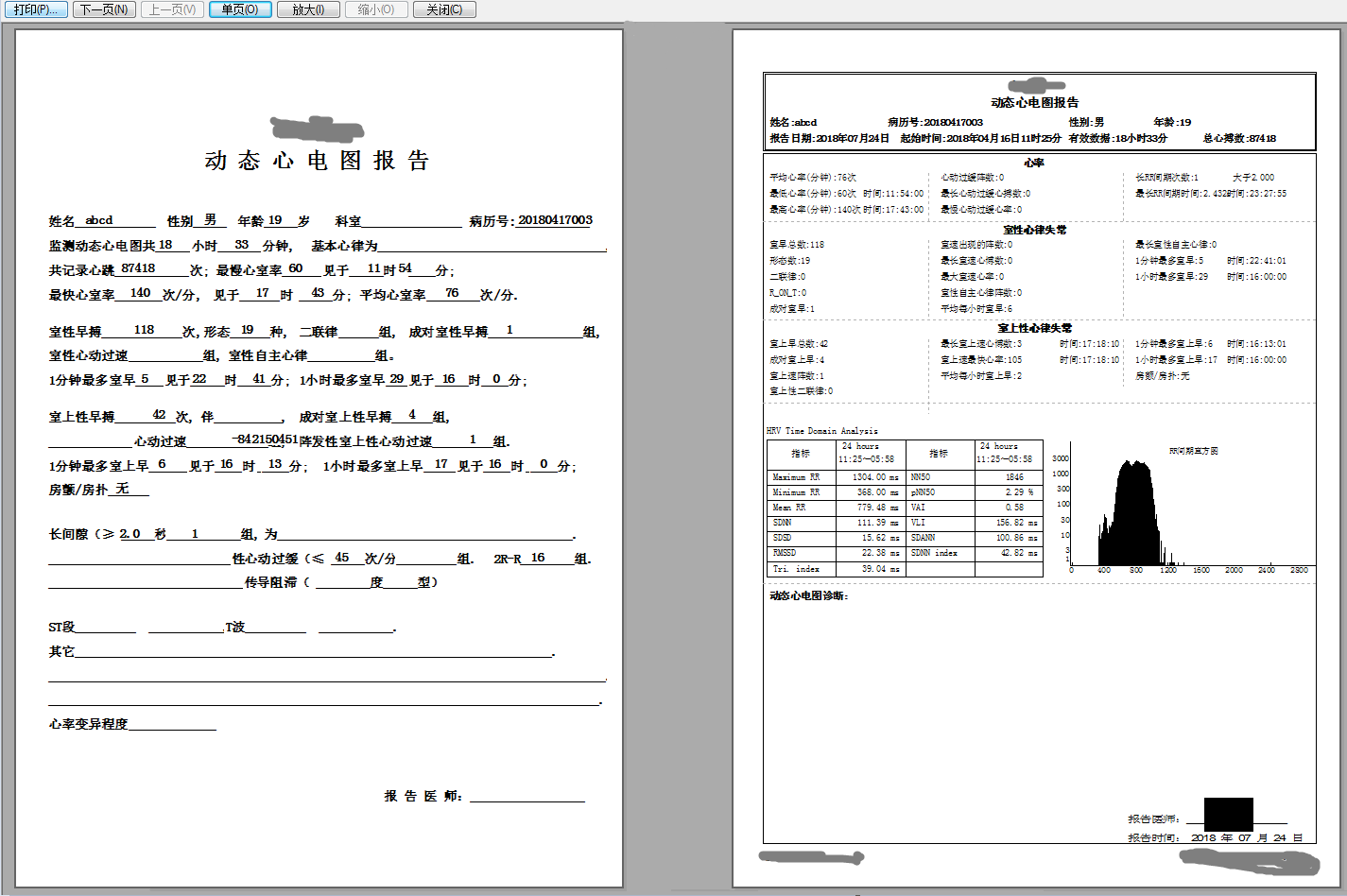


图1 执行打印功能的对话框&打印机选择对话框

当按下打印预览后则会产生一个框架窗口，显示打印预览的内容，如图2所示：

  
  
图2 打印预览效果图  
可以在上图的界面上按两页同时对两页预览，如图3所示：  
  
图3 两页同时预览效果图  
  
　　但有一点需要注意，在预览界面上的打印按钮不可用，如果按该"打印"钮则直接等于将预览窗口关掉，因此要执行打印功能必须另外在对话框的界面上放一个打印按钮，如果执行了"打印"功能，则会弹出一个选择打印机的对话框，如图1所示。这个对话框是MFC的打印结构内置的，不可以消除，当用户选择了正确的打印机后则可以打印出具体的内容了。图1 打印机选择对话框。  
  
**4、 结束语**  
　　本文从分析MFC的原代码入手，利用MFC的CView类提供的打印和打印预览功能进行了在对话框中的打印和打印预览。利用面向对象的C++写了两个可重用类CPrintFrame和CPrintView，实现在对话框中的打印和打印预览功能，极大的简化了对话框打印程序的编写。