[深入分析MFC文档视图结构（项目实践）](http://blog.chinaunix.net/uid-23254875-id-3592123.html)

分类： C/C++

2013-04-16 16:29:17

文档视图结构（Document/View Architecture）是MFC的精髓，也是Observer模式的具体实现框架之一，Document/View Architecture通过将数据和其表示分开，提供了很好的数据层次和表现层次的解耦。然而，虽然我们使用MFC AppWizard就可以很轻松地获得一个支持Document/View Architecture的MFC程序框架，Document/View Architecture本身的复杂性加上VC产生的一系列代码足够让我们眼花缭乱，一时陷入云里雾里不可自拔。加上我们更多的人都没有经过Windows SDK Programming 的学习和经历就直接进行MFC Programming的学习和开发，更是面对一堆的代码不知所措。

之于Document/View Architecture，侯捷先生的《深入浅出MFC》一书确实进行了很深入的分析和研究，网络上也有很多在侯捷先生著述的基础上的进一步的文章出现，但是个人觉得这里面有一点瑕疵（仅代表k\_eckel浅见）：太过深入，这些分析和研究都最终会定位到Windows SDK中窗口的创建过程、MFC中对Document/View Architecture支持所提供的复杂无序的宏等对于没有Windows SDK Programming 经验和经历的学习者和对于MFC不是很熟悉的学习者无异于是徒增烦恼，一个本来就很复杂的问题更加地复杂化了。我的观点是这个过程是必要的，然而不是每个人所必需的，或者说不是每个人在初期学习和绝大多数项目开发中所必需的。我向来对众多的仅仅学会了拖拉点拽就以为会了MFC（侯捷先生在《深入浅出MFC》一书中对这个群体有一个照面，这里不罗嗦）不以为然，但是我依然认为轻量级学习成本是重要的，容易上手，易于接受是一门技术成功或者说有价值的一个很大的决定性因素。因此提供一个轻量级的学习过程对于学习来说是很有必要性的，本系列文章就遵循这样一个理念，对MFC中Document/View Architecture进行一个分析（姑且也可以称之为深入），尽量将对Document/View Architecture的理论研究（侯捷先生书中很多内容）和实际的项目开发结合起来，最后提供一个简单但是全面的Document/View Architecture项目开发（主要是界面框架设计和开发）的实际例子，供参考。

**1 必备基础知识概述**

**1.1 MFC文档视图结构程序结构总揽**

当我们使用MFC AppWizard生成一个MFC程序，选用所有默认的设置（当然也是Multiple Documents，本文讨论主要基于Multiple Documents，对于Single Document情况仅以简单表述提及，皆因后者和前者很多相似相同之处，但前者更为复杂，并且更加常用。），假设你的程序名称为A ，则你会得到CMainFrame、CChildFrame、CAboutDlg、CADoc、CAView、CAApp 6个类（Single Document 仅少一个CChildFrame类，其余均同）。这些类的具体含义将在后面给出，这里先要给出一个MFC 支持文档视图结构程序（以下简称App）的主要组成：

       一个App（对应类CAApp）可以包含多个文档模版（CDocTemplate），但是MFC AppWizard（无论是SDI还是MDI）都只是默认生成一个。但是在实际开发中一个文档模版不够，需要自己手工添加（在后面实际项目例子提供示例）。这个结构是通过MFC中CWinApp的成员变量CDocManager\* m\_pDocManager实现的，我们的CAApp正是继承自MFC提供的CWinApp类。

       CDocManager类则拥有一个指针链表CPtrList m\_templateList来维护这些文档模版。这些文档模版都是在CAApp：：InitInstance（）中通过AddDocTemplate(pDocTemplate)。

       CDocTemplate拥有3个成员变量，分别保存着Document、View、Frame的CRuntimeClass指针，另外持有成员变量m\_nIDResource，用来指定这个Document显示时候采用的菜单资源。这4份数据都在CAApp：：InitInstance（）中CDocTemplate的构造函数中指定。在Document中拥有一个回指CDocTemplate的指针（m\_pDocTemplate）。

       一个文档可以有多个视图，由Document中成员变量CPtrList m\_ViewList维护。

       CFrameWnd拥有一个成员变量CView\* m\_pActiveView指向当前活动的View。

       CView中拥有一个成员变量CDocument\* m\_pDocument指向该视图相关的Document。

 [注解]：①MFC SDI/MDI程序默认都默认生成了一个文档模版，并将这个文档模版Add到其文档模版的链表中，由于这是MFC默认提供的，因此这个文档模版会被插入到文档模版的第一个位置，而MFC也是通过这个文档模版的特定位置可以确定的。默认情况下，当我们点击File（Open）/ File（New）的时候，这个文档模版会被启用。

         除了侯捷先生在《深入浅出MFC中列出的以上的深入分析，我们还应该（很大程度上更加重要）掌握以下的关于MFC SDI/MDI的知识：

       文档的本质：文档是用来保存数据以及关于数据的处理的，每当MFC SDI/MDI响应File（Open）/ File（New）的时候都会打开一份文档。文档可以拥有多个视图。文档和视图的关系可以这样理解：文档是被视图观察的对象。

       视图本质：视图在Windows中就是一个窗口，也就是一个可视化的矩形区域。视图是用来表示文档的数据的。但是每个视图必需依附于一个框架（SDI中是MainFrame，MDI是ChildFrame）。当然你可以自己去Create一个视图，并且去显示它。

       框架的本质：框架实际是也是一个Windows窗口。但是在框架上可以放置菜单、工具栏、状态栏等。而视图则放在框架的客户区。因此MFC中我们看到的窗口实际上Frame和View共同作用的结果。

       在某一时刻，程序中只有一个活动的文档、框架和视图，即当前的文档、框架、视图。

**1.2 MFC SDI/MDI各个类之间的互访**

       在实际项目开发中用的最多就是各个类之间的互访问，这里将网络上和书籍中提到的做了一个总结，也是笔者在实际开发中都用到过的。

|  |  |  |
| --- | --- | --- |
| 访问对象 | 访问位置 | 访问实现 |
| 应用程序App | 任何位置 | ①     AfxGetApp();  ②     在要使用应用程序App的文件中加入：  extern CAApp theApp，然后直接使用全局的theApp变量。 |
| 主框架窗口 | 任何位置 | ①AfxGetMainWnd();  ②AfxGetApp()->m\_pMainWnd; |
| 视图 | 框架类中 | GetActiveView();  //当前的活动视图 |
| 文档类中 | GetFirstViewPosition（）；//可以获取全部视图  GetNextView（）； |
| 文档 | 文档类中 | GetDocument()； |
| 文当模版类中 | GetFirstDocPosition(); //该文档模版对应全部文档  GetNextDoc(); |
| 框架类中 | GetActiveDocument(); //当前活动文当 |
| 子框架类（MDI中） | 主框架类中 | ①MDIGetActive（）；  ②GetActiveFrame（）； |
| 视图类中 | GetParentFrame(); |
| 文档模版 | 文档类中 | GetDocTemplate(); |
| 应用程序App中 | GetFirstDocTemplatePosition();  GetNextDocTemplate(); |

       说明：1）以上给出的都是方法，实际访问中可能还要进行以下简单的处理，如类型转换，循环遍历等；

                2）可能没有列举完所有可能位置的互访问，但可以通过他们的组合得到。

**2 文档、视图、框架之间的关联**

       MFC SDI/MDI中的核心就在于文档、视图、框架之间的关联，形成了一个有机的可运作的整体。MFC提供了默认的关联关系，但是在实际的项目开发中很多时候需要动态进行他们的之间的关联。

**2.1 文档和视图间的关联**

       使用MFC AppWizard声称MFC SDI/MDI程序，在App类的InitInstance（）方法中有如下代码（假设Project名称均为Test）：

       SDI中

|  |
| --- |
| CSingleDocTemplate\* pDocTemplate;  pDocTemplate = new CSingleDocTemplate(                IDR\_MAINFRAME,                RUNTIME\_CLASS(CTestDoc),                RUNTIME\_CLASS(CMainFrame),       // main SDI frame window                RUNTIME\_CLASS(CTestView));         AddDocTemplate(pDocTemplate); |

       MDI中

|  |
| --- |
| CMultiDocTemplate\* pDocTemplate;  pDocTemplate = new CMultiDocTemplate(                IDR\_TESTTYPE,                RUNTIME\_CLASS(CTestDoc),                RUNTIME\_CLASS(CChildFrame), // custom MDI child frame                RUNTIME\_CLASS(CTestView));         AddDocTemplate(pDocTemplate); |

这里通过CDocTemplate（无论是SDI中的CSingleDocTemplate还是MDI中的CMultiDocTemplate）的构造函数，将文当、视图和框架（SDI中与主框架，MDI中与自框架）关联在一起了，形成了一个整体。

手工实现文当和视图的关联

在实际的项目开发时候仅仅依靠MFC AppWizard生成的文当和视图、框架是不够的，因此我们需要掌握手工进行这种关联。手工进行文当和视图的关联可以有以下两种实现方式：

         模仿MFC AppWizard实现，使用CDocTemplate的构造函数：在上面的分

析中我们可以看到通过CDocTemplate（无论是SDI中的CSingleDocTemplate还是MDI中的CMultiDocTemplate）的构造函数我们可以获得文档、视图和框架的关联。因此可以通过模拟这种方式进行关联。具体实现方法如下：

1）  创建新的文档、视图和框架类，方法是使用VC中的Insert MFC Class

实现。注意到框架类选择CMDIChildWnd作为基类，文档类选择CDocument作为基类，而视图类则可以根据需要选择CView或其子类（CEditView）等作为基类。

2）  为该框架添加菜单资源，方法是在VC资源窗口Menu菜单下添加新的菜

单，当然可以通过复制VC提供默认菜单进行修改。

3）  在App类的InitInstance（）中添加如下类似代码：

|  |
| --- |
| CMultiDocTemplate\* m\_pDocTemplate;  m\_pDocTemplate = new CMultiDocTemplate(                IDR\_TESTTYPE,               //改为你新建的菜单资源ID                RUNTIME\_CLASS(CTestDoc),   //改为你新建的文档类                RUNTIME\_CLASS(CChildFrame), //改为你新建的框架类                RUNTIME\_CLASS(CTestView));  //改为你新建的视图类         AddDocTemplate(m\_pDocTemplate); |

4）  为了记录这个文档模版，你可以在App类中添加一个CMultiDocTemplate\*

类型变量来维持这个文档模版。

         上面给出了通过CDocTemplate的构造函数将文档、视图、和框架关联起来，但

是有时候我们并不想创建一个新的文档模版，我们只是想给同一个数据提供不同的结果显示，或者说是为同一个文档添加一个新的视图，并提供他们之间的一个切换。我在《[VC项目开发之单文档多视图实现](http://www.mscenter.edu.cn/blog/k_eckel/articles/3027.html)》一文给出了这样的一个实现。还有一种可能就是我们本来不是一个文档视图结构支持的程序，想为视图添加一个文档，更好进行业务逻辑和表示层的一个分离。因此我们给出了第二种视图和文档关联的方法：我们使用CCreateContext类进行他们之间的关联，具体实现为：

|  |
| --- |
| m\_pAnotherView = new CAnotherView(); //new 一个新的视图，可以改为你新建的视图        //获取一个已有的文档，可以是你新建的文档         CDocument\* m\_pDoc = ((CFrameWnd\*)m\_pMainWnd)->GetActiveDocument();    //文档和视图关联  CCreateContext context;         context.m\_pCurrentDoc = m\_pDoc;           //创建视图         UINT m\_IDFORANOTHERVIEW = AFX\_IDW\_PANE\_FIRST + 1; //创建视图的ID号，你可以自己设置         CRect rect;         m\_pAnotherView->Create(NULL,NULL,WS\_CHILD,rect,m\_pMainWnd,m\_IDFORANOTHERVIEW,&context); |

         在框架和视图关联的时候进行设置，具体见一下框架和视图关联部分。

**2.2 框架和视图的关联**

       在第一部分分析我们知道，框架和视图其实都是windows窗口，不过框架提供了菜单、标题栏、状态栏等资源，而视图则只是一个矩形区域。MFC程序中视图决定大多数时候要依附于一个框架（SDI中的MainFrame和MDI中的子框架窗口），可以这样理解，框架相当于一个窗口容器（当然它本身也是一个windows窗口），而视图则正好是放置在框架内客户区域的内容。

       框架和视图的关联也可以通过模仿MFC AppWizard实现，使用CDocTemplate的构造函数实现，即和2.1中文档和视图间的关联方式相同，这里就不再给出，参看上面的详细实现即可。

       同上面的分析，在很多的时候我们并不是需要提供一个新的文档模版，我们只是需要显示一个新的窗口（MDI程序），例如我们在作MIS系统界面管理的时候，经常出现的情况就是用户点击一个菜单选项，即弹出一个处理窗口。而要显示一个新的窗口，我们可以通过我在《[VC 多文档用户界面设计](http://www.mscenter.edu.cn/blog/k_eckel/articles/3401.html)》一文中给出的方法（通过CDocTemplate的OpenDocumentFile（）方法打开一个文档实现），这样就建立一套的文档、视图和框架的体系。上面已经分析到，我们看到MFC的窗口实际上框架和视图的一个结合体，我们并不一定要提供文档、视图、框架的整个体系，我们只需要框架和视图的结合即可实现窗口的显示，这就要经过两个步骤实现：第一步将视图和框架关联，第二步显示框架（也就是一个windows窗口的显示）。以下给出框架和视图关联的具体实现：

|  |
| --- |
| CChildFrame\* pFrm = new CChildFrame();   //框架可以是你新建或者定制的框架类           CCreateContext context;         context.m\_pNewViewClass = RUNTIME\_CLASS(CDemoView);  //视图可以是你想显示的视图           pFrm->LoadFrame(IDR\_TEST2TYPE,WS\_CHILD |  WS\_OVERLAPPEDWINDOW,this,&context);  //菜单资源你可以修改           pFrm->ShowWindow(SW\_SHOW);           //显示窗口           pFrm->InitialUpdateFrame(NULL,true);        //调用视图的OnInitialUpdate（）和框架的ActiveFrame（），你可以在这里设置窗口的标题 |

       当然你可以在这里添加视图和文档的关联，具体实现是添加以下代码：

|  |
| --- |
| context.m\_pCurrentDoc = m\_pDoc;   // m\_pDoc就是你要关联到的文档对象 |

**3 代码实例**

本部分将以一个通用的MDI项目界面设计开发为例，将上面的分析附诸实践。

**3.1 情景描述**

       一个常见界面逻辑为：用户打开一个系统，显示基本的菜单，用于用户的登录、注销和用户管理（当然这也可以通过一个用户登录的对话框实现）。用户正确登录后，显示系统的功能操作界面，当用户点击一个菜单项后（对应一个或者多个业务逻辑），弹出一个处理操作界面（并非对话框）。各个功能操作界面可共存于一个框架内，可以最大化、最小化或者关闭。

**3.2 代码实现**

       下面就将上面的提到的情景用前面提到的技术，给出详细的实现方案。

**Step 1**：新建一个MFC 项目，名称为Demo，选择不用Document/View Architecture支持（第二步去掉默认的复选框即可）。这样系统为你默认生成5个类：CDemoApp、CMainFrame、CChildFrame、CAboutDlg、CDemoView。各个类的含义上面已经分析了，不罗嗦。另外系统还提供了两个默认的菜单：IDR\_DEMOTYPE和IDR\_MAINFRAME，将IDR\_MAINFRAME的菜单的“文件”改名为“开始”（好像更加专业，不该也没有什么，本来就是Demo），然后将这个菜单项的子菜单中改为“登录”、“注销”，并使用默认的子项，并将前两者的ID号改为：“ID\_LOGIN”和“ID\_LOGOUT”。拷贝（Ctrl + C）粘贴（Ctrl + V），则得到一个名称为IDR\_MAINFRAME1的菜单资源，删除原有的IDR\_DEMOTYPE菜单（注一先保存名称），再将IDR\_MAINFRAME1的名称改为IDR\_DEMOTYPE。再为IDR\_DEMOTYPE添加一个菜单项“功能”，添加两个子菜单项“业务逻辑一”和“业务逻辑二”，ID号分别为：ID\_FUNC\_ONE和ID\_FUNC\_TWO。

**Step 2**：给CDemoApp添加两个变量，保存菜单资源：

|  |
| --- |
| HMENU m\_hOPMenu;         HMENU m\_hInitMenu; |

并在BOOL CDemoApp::InitInstance()中添加代码：

|  |
| --- |
| m\_hInitMenu = ::LoadMenu(hInst, MAKEINTRESOURCE(IDR\_MAINFRAME));  m\_hOPMenu = ::LoadMenu(hInst, MAKEINTRESOURCE(IDR\_DEMOTYPE)); |

为了显示效果，在BOOL CDemoApp::InitInstance()中添加代码：

|  |
| --- |
| pFrame->SetWindowText("武汉大学计算机学院-未登录");  m\_nCmdShow = SW\_SHOWMAXIMIZED;    *pFrame->ShowWindow(m\_nCmdShow);  //系统提供*  *pFrame->UpdateWindow();* |

为菜单项“登录”添加响应函数（使用Class Wizard，Class Name选择CDemoApp）

|  |
| --- |
| void CDemoApp::OnLogin()  {         // TODO: Add your command handler code here         SetMenu(AfxGetApp()->m\_pMainWnd->m\_hWnd,m\_hOPMenu);         AfxGetApp()->m\_pMainWnd->SetWindowText("武汉大学计算机学院-已登录");  } |

同上，为菜单项“注销”添加响应函数：

|  |
| --- |
| void CDemoApp::OnLogout()  {         // TODO: Add your command handler code here         if(((CMainFrame \*)AfxGetMainWnd())->m\_pFuncOneFrame != NULL)                ((CMainFrame \*)AfxGetMainWnd())->m\_pFuncOneFrame->SendMessage(WM\_CLOSE);           if(((CMainFrame \*)AfxGetMainWnd())->m\_pFuncTwoFrame != NULL)                ((CMainFrame \*)AfxGetMainWnd())->m\_pFuncTwoFrame->SendMessage(WM\_CLOSE);           SetMenu(AfxGetApp()->m\_pMainWnd->m\_hWnd,m\_hInitMenu);           AfxGetApp()->m\_pMainWnd->SetWindowText("武汉大学计算机学院-未登录");  } |

这就实现了登录和注销的功能（当然实际中可能还会有一个验证用户权限和合法性的对话框，这里从略），并实现了登录注销时刻用户操作菜单的转变。注意：这里OnLogout中前两行代码是在注销的时候要把已经打开的窗口关闭而添加的，m\_pFuncOneFrame和m\_pFuncTwoFrame的定义和作用请参见后面定义。

**Step 3**：添加一个新的子框架类CDemoFrame，其基类为CMDIChildWnd。添加一个两个新的视图类CFuncOneView和CFuncTwoView类，前者的基类为CView，后者为CFormView。当然为了添加CFuncTwoView类，需要先Insert一个对话框资源，并将ID改为IDD\_FUNC\_TWO\_DLG，属性Style修改为“child”（默认为Popup）。这样在新建CFuncTwoView的时候选择该DialogID为IDD\_FUNC\_TWO\_DLG。注意将CDemoFrame的构造函数改为public（默认是protected）。

**Step 4**：为CMainFrame添加两个成员变量记录各个业务逻辑对应的窗口，在MainFrm.h中添加：

|  |
| --- |
| CDemoFrame\* m\_pFuncOneFrame;  CDemoFrame \* m\_pFuncTwoFrame; |

       并在CMainFrame::CMainFrame()中初始化：

|  |
| --- |
| CMainFrame::CMainFrame()  {         // TODO: add member initialization code here         m\_pFuncOneFrame = NULL;         m\_pFuncTwoFrame = NULL;  } |

**Step 5**：为“业务逻辑一”添加响应函数（CMianFrame中）：

|  |
| --- |
| void CMainFrame::OnFuncOne()  {         // TODO: Add your command handler code here         if (m\_pFuncOneFrame != NULL)         {                m\_pFuncOneFrame->MDIActivate();                  return ;         }    m\_pFuncOneFrame = new CDemoFrame();           CCreateContext context;         context.m\_pNewViewClass = RUNTIME\_CLASS(CFuncOneView);           m\_pFuncOneFrame->LoadFrame(IDR\_MAINFRAME,WS\_MAXIMIZE|WS\_OVERLAPPEDWINDOW,this,&context);           m\_pFuncOneFrame->SetWindowText("业务逻辑一");         m\_pFuncOneFrame->ShowWindow(SW\_SHOWMAXIMIZED);         m\_pFuncOneFrame->InitialUpdateFrame(NULL,true);  } |

       同上，为“业务逻辑二”添加响应函数：

|  |
| --- |
| void CMainFrame::OnFuncTwo()  {         // TODO: Add your command handler code here         if (m\_pFuncTwoFrame != NULL)         {                m\_pFuncTwoFrame->MDIActivate();                  return ;         }           m\_pFuncTwoFrame = new CDemoFrame();           CCreateContext context;         context.m\_pNewViewClass = RUNTIME\_CLASS(CFuncTwoView);           m\_pFuncTwoFrame->LoadFrame(IDR\_MAINFRAME,WS\_MAXIMIZE|WS\_OVERLAPPEDWINDOW,this,&context);           m\_pFuncTwoFrame->SetWindowText("业务逻辑二");         m\_pFuncTwoFrame->ShowWindow(SW\_SHOWMAXIMIZED);         m\_pFuncTwoFrame->InitialUpdateFrame(NULL,true);  } |

       这样，上述的需求情景基本是做到了，但是需要说明的是：

1）  需要在适当的地方加入适当的头文件，就是说在使用类的时候要include其实现的头文件。

2）  上面其实是很多的MIS管理系统的通用界面操作模版，大家可以在实际的项目开发中作相应的修改（主要是实现相应的业务逻辑等）；

3）  这里提供的默认的主框架窗口比较简单（默认），要添加一个背景，请参看笔者的《[VC—主框架窗口绘制背景](http://www.mscenter.edu.cn/blog/k_eckel/articles/3417.html)》一文添加合适的背景；

4）  上面的例子中，是让其生成不支持MFC Document/View Architecture，如果已经有了默认的支持MFC Document/View Architecture的程序，请将App中模仿上面的实现即可。

5）  上面可以使用MFC AppWizard生成的CChildFrame类代替CDemoFrame类，但是建议新建框架类，业务逻辑一和业务逻辑二可以使用不同的框架类，模仿实现即可。

当然，你肯能需要为视图添加文档以实现业务逻辑和表现层的松耦合，下面就为CFuncOneView添加文档视图结构支持。

**Step 6**：添加一个文档类CDemoDoc，基类为CDocument并将CDemoDoc的构造函数改为public（默认为protected），为了演示，为CDemoDoc添加函数GetData（）：

|  |
| --- |
| CString CDemoDoc::GetData()  {         return "Hello world";  } |

**Step 7**：为CFuncOneView添加函数GetDocument：

|  |
| --- |
| CDemoDoc\* CFuncOneView::GetDocument()  {         return (CDemoDoc\*)m\_pDocument;  } |

并修改OnDraw函数：

|  |
| --- |
| void CFuncOneView::OnDraw(CDC\* pDC)  {         //CDocument\* pDoc = GetDocument();         // TODO: add draw code here         CDemoDoc\* pDoc = GetDocument();           pDC->TextOut(50,50,pDoc->GetData());  } |

**Step 8**：将CDemoDoc和CFuncTwoView关联：修改void CMainFrame::OnFuncOne()函数为：

|  |
| --- |
| void CMainFrame::OnFuncOne()  {         // TODO: Add your command handler code here         if (m\_pFuncOneFrame != NULL)         {                m\_pFuncOneFrame->MDIActivate();                  return ;         }           m\_pFuncOneFrame = new CDemoFrame();           CDemoDoc\* m\_pDoc = new CDemoDoc();           CCreateContext context;         context.m\_pNewViewClass = RUNTIME\_CLASS(CFuncOneView);           context.m\_pCurrentDoc = m\_pDoc;           m\_pFuncOneFrame->LoadFrame(IDR\_MAINFRAME,WS\_MAXIMIZE|WS\_OVERLAPPEDWINDOW,this,&context);           m\_pFuncOneFrame->SetWindowText("业务逻辑一");         m\_pFuncOneFrame->ShowWindow(SW\_SHOWMAXIMIZED);         m\_pFuncOneFrame->InitialUpdateFrame(NULL,true);  } |

至此，就完成了整个过程。需要说明的是：

1）  需要在适当的地方加入适当的头文件，就是说在使用类的时候要include其实现的头文件。

2）  如果是已经有了文档视图支持的程序，上面的文档可以使用系统中提供的文档；

上面提供了一个简单通用的界面操作的实现，大家可以参照实现，例如添加更多的视图、实现自己的业务逻辑等。

3）本文提供了实现的[整个代码](http://www.mscenter.edu.cn/blog/Files/k_eckel/Demo(src).rar)，请参考。