如何面对源码

潘爱民

2002-10-15

内容

- 源码的类型
- 阅读源码前的准备
- 工具
- 一些例子
  - Snort
  - MFC
  - ATL
  - Linux

动机

- 好的源码是经验和智慧的结晶
- 写代码 —— 读代码
- 我们的工作离不开优秀的源代码

我接触过的源码

- DOS时代的源码
- MFC
- ATL
- Snort/libpcap
- Crypto++
- Linux
- PGP
- (OpenXXX)
- ……

源代码类型

- 系统一级的源代码
- 可重用库的源代码
- 书籍带的源代码
  - 小例子程序
- 源码开放的软件
- 半公开源码的软件
- ……

阅读源码前的思考

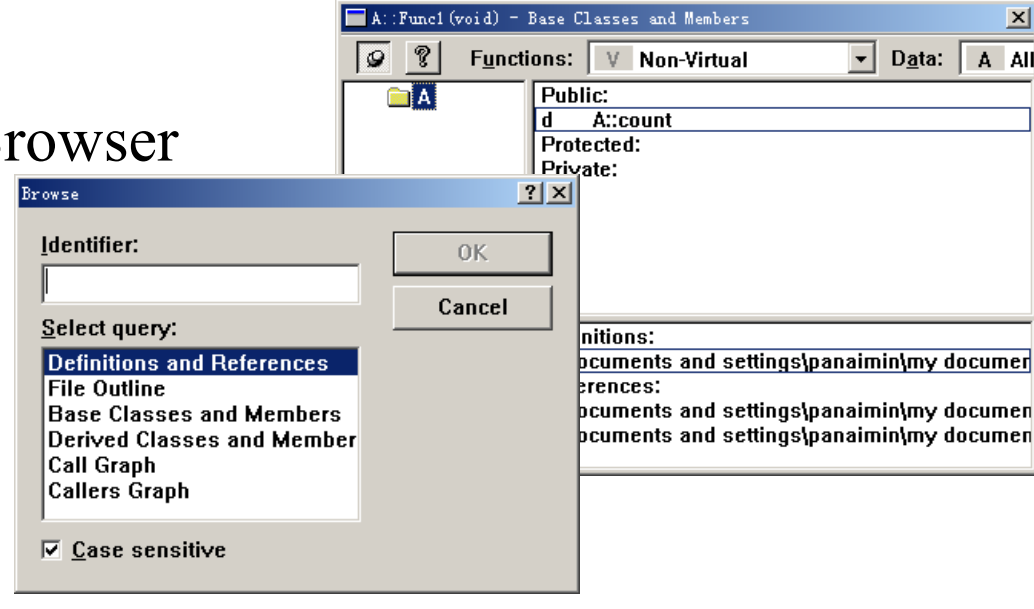
- 熟悉这套源码的背景
  - 首先把自己放在用户的角度试用一下
    - 例如，对于MFC，可以写一些例子用一用
    - 对于crypto++，可以写几个加解密小程序
  - 自己的知识背景是否具有可达性？
    - 简单测试：让你来写这样的源码，困难有多大？
- 判断源码的规模
  - 不同的规模有不同的阅读策略

阅读源码前的准备

- 知识上的准备
  - 如果不懂OO，要想读一套类库是比较困难的
  - 如果不懂有关的原理，要想通过阅读源码来弄懂原理，这是事倍功半的做法
    - 但是，有时候，结合代码来学习原理也是可行的，这种情况下，代码的作用在于具体化
- 文档
  - 找到有效的文档
- 前人的经验
  - 如果能找到别人总结的经验，那是再好不过了
  - 越是通用的，越有可能找得到

阅读源码使用的工具

- 一个拿手的编辑器总是你最需要的
- 检索的功能
  - grep, VC中的find in files
- 调试的功能
  - 一个好的调试器有助于整理调用关系
- 源码分析器
  - VC内置的Source Browser
  - Rose的逆向工程
  - Source Navigator



开始探索

- 有书指导当然最好，没有书怎么办？
- 探索一下目录结构
  - 善用文件搜索器(Windows的资源管理器)
- 找到入手点
  - 有线头找线头
  - 无线头怎么办？发挥想象力
- 熟悉代码风格

三种类型

- 独立的应用
  - Snort/libpcap
  - 大量的OpenSource软件
- 可重用类库
  - MFC/ATL
  - Crypto++
- 大型系统
  - 操作系统
    - Linux
    - OpenBSD
  - 协议栈实现

独立应用snort分析

- 估计一下代码量
- 找到main()，这是重要的入口点
  - 结构化程序设计
    - 找到数据流的途径
    - 或者功能分解
  - 数据流 &功能分解
    - 抓包 -> decode -> 检测 -> 完成
  - 然后依次分析每一部分

Snort-1-6-Win32之源文件列表

acconfig.h
decode.h
getopt.h
log.h
mstring.h
name.h
plugbase.h
prototypes.h
respond.h
rules.h
snort.h
sp_dsize_check.h
sp_icmp_code_check.h
sp_icmp_id_check.h
sp_icmp_seq_check.h
sp_icmp_type_check.h
sp_ip_id_check.h
sp_ipoption_check.h
sp_pattern_match.h
sp_rpc_check.h
sp_session.h
sp_tcp_ack_check.h
sp_tcp_flag_check.h
sp_tcp_seq_check.h
sp_ttl_check.h
spo_alert_syslog.h
spo_log_postgresql.h
spo_log_tcpdump.h
spp_http_decode.h
spp_minfrag.h
spp_portscan.h
syslog.h
decode.c
getopt.c
log.c
mstring.c
plugbase.c
respond.c
rules.c
snort.c
sp_dsize_check.c
sp_icmp_code_check.c
sp_icmp_id_check.c
sp_icmp_seq_check.c
sp_icmp_type_check.c
sp_ip_id_check.c
sp_ipoption_check.c
sp_pattern_match.c
sp_rpc_check.c
sp_session.c
sp_tcp_ack_check.c
sp_tcp_flag_check.c
sp_tcp_seq_check.c
sp_ttl_check.c
spo_alert_syslog.c
spo_log_postgresql.c
spo_log_tcpdump.c
spp_http_decode.c
spp_minfrag.c
spp_portscan.c
syslog.c
README.FLEXRESP
config.guess
config.h.in
configure.in
Makefile.in
stamp-h.in
lib
aclocal.m4
name.mc
README.PLUGINS
name.rc
RULES.SAMPLE
config.sub
README_WIN32
AUTHORS
backdoor-lib
BUGS
ChangeLog
configure
COPYING
CREDITS
INSTALL
install-sh
misc-lib
missing
mkinstalldirs
NEWS
overflow-lib
README
scan-lib
snort-lib
USAGE
web-lib

Snort中的components

- Main: parse the command line
- Rule
- Decode
- Preprocess
- Detect
- Plugin
- libpcap

可重用库

- 非类库
  - Libpcap
  - STL
- 类库
  - MFC
  - ATL
- 首先要熟悉可重用库的API
  - API的设计思想

插：framework

- 领域工程
  - 单个系统——〉一类系统
  - 有较强的抽象能力
- 组件库
- 提供定制功能，允许开发人员对于框架主体部分进行修改
- 不同层次上的framework
  - 基于二进制代码的framework，例如MMC
  - 基于源代码的framework，例如MFC

插：可重用类库的设计(一)

- 在所有的系统设计中，可重用类库的设计是难度比较大的，要做到：
  - 使用：灵活性和易用性
  - 功能：广泛性和效率
- 经验非常重要
  - 实现同样的功能会有许多不同的道路，如何选择？效果怎么样？
- 类库的基础
  - 是否使用其他的类库？是否使用特殊的平台和编译环境？
- 参考成功的类库
  - 起点要高

插：可重用类库的设计(二)

- 接口的设计
  - 这是类库的关键，会影响到类库的使用
    - 接口的类型：C/C++
  - 大而全的接口并不理想
  - 接口的语义一定要清晰
  - facade模式
- 内存管理
  - 保证内存分配和释放的一致性
  - 使用要方便
    - [out]参数的资源由谁来申请？谁知道size？
    - 是否使用自定义的内存分配器，例如针对小对象的分配器

插：可重用类库的设计(三)

- 使用各种模式
  - 模式是经验，成功的典范
  - strategy模式允许使用者定制类
  - 结构型模式有助于建立起更加合理的结构模型，而不至于层次错综复杂
  - 行为型模式有助于各个类之间有更好的协作模型
  - 创建型模式可以提供各种合理的创建机制
- 模板类库的特殊性
  - 利用模板类型实现compile-time的预处理
  - 熟悉编译器的特性
  - 控制模板生成代码

插：可重用类库的设计(四)

- 行为前置和延后
  - 在基类中提供缺省的实现
  - 纯虚函数 —— 强制子类提供实现
  - 利用functor或者函数指针
  - 要求(必须)子类调用父类的实现
- 用宏来封装代码
- 代码风格
- 类库的优化
  - 优化需要用到内部知识，是否暴露这些知识
  - 允许使用者用policy进行配置，用不同的实现配置类
  - 类似于policy的思想，在细节点上用开关进行控制

插：可重用类库的设计(五)

- 类库的调试
  - 类库内部调试，使用assert支持
- 类库的测试
  - 比应用系统的测试更加严格
- 类库的发行
  - 是否提供源代码？
  - 文档。
  - 编译设置

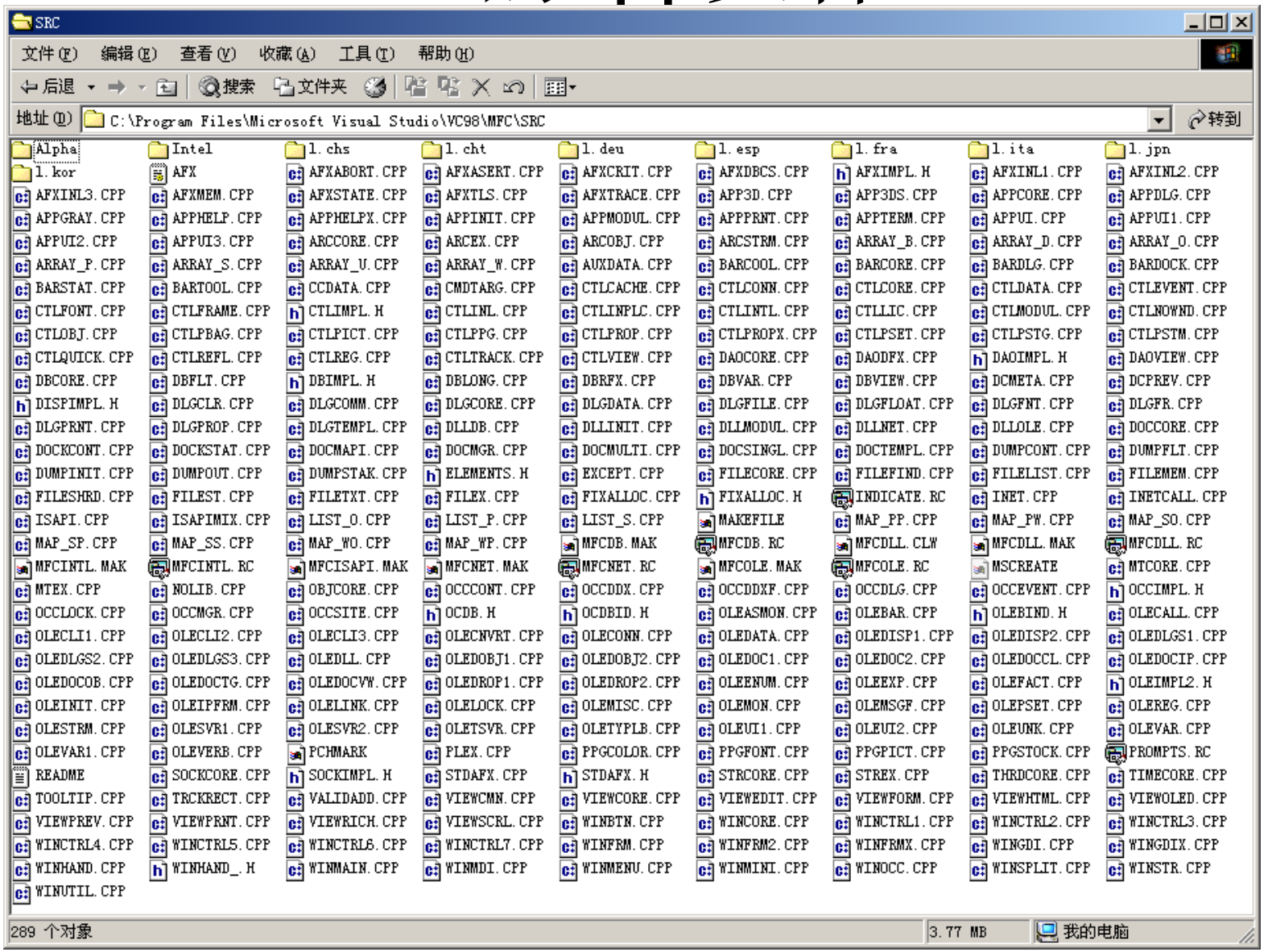
MFC

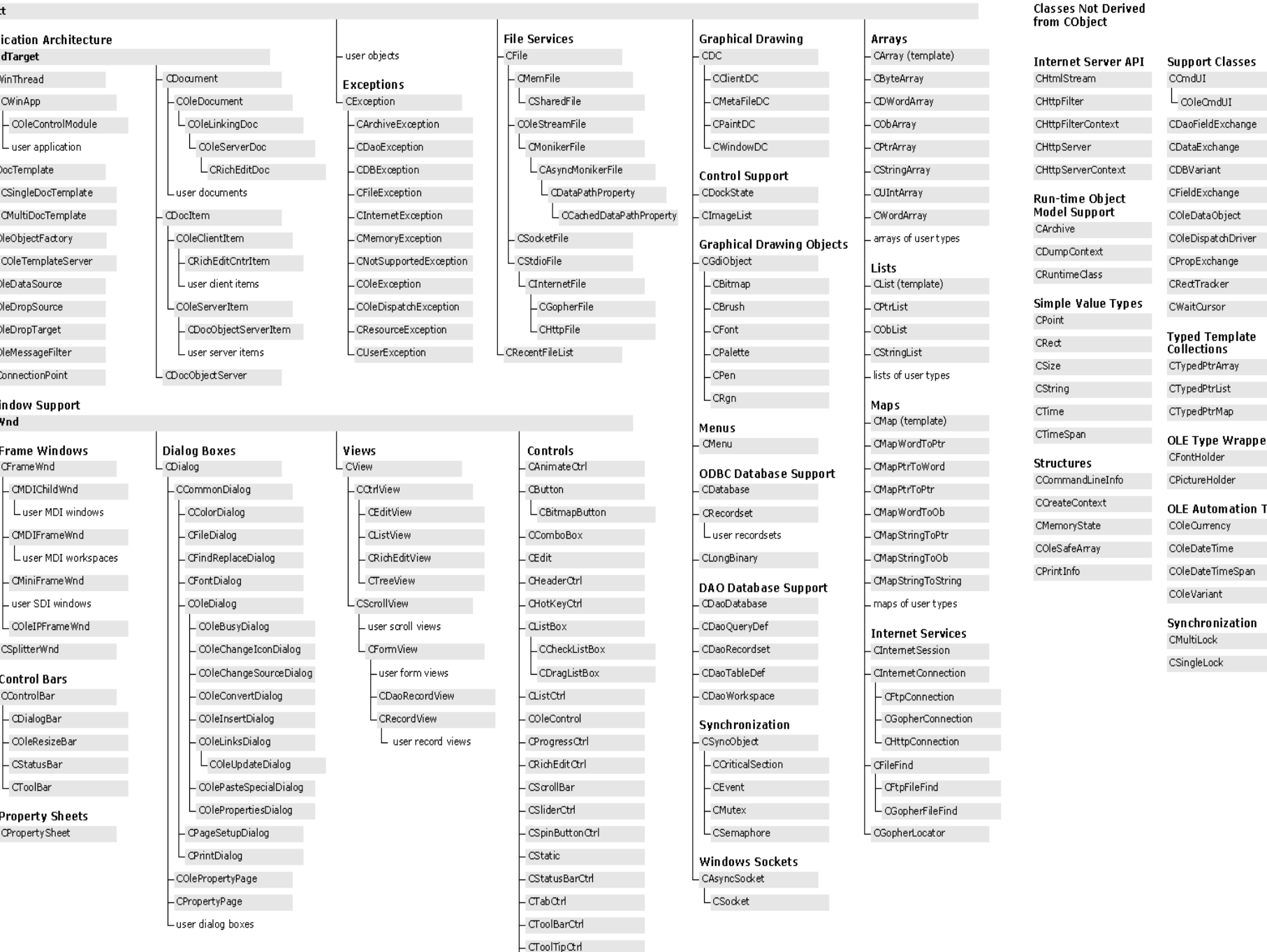
- MFC太庞大了！
  - 已经不适合作为类库的典范来学习
  - 而且，在发展过程中，MFC塞进了很多东西
- 首先熟悉MFC
  - 检查一下MFC的文件
  - 了解MFC的命名规范
    - OnXXX、DoXXX、CreateXXX、…
  - 看看MFC的类图
- 知识基础：Windows SDK，C++、OO
- 根据需要学习某一部分内容

MFC头文件


Include
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退 搜索 文件夹
地址(D) C:\Program Files\Microsoft Visual Studio\VC98\MFC\Include
转到
1.chs 1.cht 1.deu 1.esp 1.fra
1.ita 1.jpn 1.kor Res AFX.H
AFX.INL AFXADV.H AFXBLD_.H AFXCMN.H AFXCMN.INL
AFXCMN2.INL AFXCOLL.H AFXCOLL.INL AFXCOM_.H AFXCONV.H
AFXCTL.H AFXCTL.INL AFXCTL.RC AFXCVIEW.H AFXCVIEW.INL
AFXDAO.H AFXDAO.INL AFXDB.H AFXDB.INL AFXDB.RC
AFXDB_.H AFXDD_.H AFXDISP.H AFXDLGS.H AFXDLGS.INL
AFXDLL_.H AFXDLLX.H AFXDOCOB.H AFXDTCTL.H AFXDTCTL.INL
AFXEXT.H AFXEXT.INL AFXHELP.HM AFXHTML.H AFXHTML.INL
AFXINET.H AFXINET.INL AFXISAPI.H AFXISAPI.INL AFXISAPI.RC
AFXMSG_.H AFXMT.H AFXMT.INL AFXODLGS.H AFXOLE.H
AFXOLE.INL AFXOLECL.RC AFXOLEDB.H AFXOLESV.RC AFXPLEX_.H
AFXPRINT.RC AFXPRIV.H AFXPRIV2.H AFXRES.H AFXRES.RC
AFXRICH.H AFXRICH.INL AFXSOCK.H AFXSOCK.INL AFXSTAT_.H
AFXTEMPL.H AFXTLS_.H AFXV_CFG.H AFXV_CPU.H AFXV_DLL.H
AFXV_W32.H AFXVER_.H AFXWIN.H AFXWIN1.INL AFXWIN2.INL
MFCSAMPS.MAK MSCREATE WINRES.H
83 个对象
1.23 MB
我的电脑

MFC的cpp文件


SRC
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退 搜索 文件夹
地址(D) C:\Program Files\Microsoft Visual Studio\VC98\MFC\SRC
转到
Alpha
Intel
1.chs
1.cht
1.deu
1.esp
1.fra
1.ita
1.jpn
1.kor
AFX
AFXABORT.CPP
AFXASERT.CPP
AFXCRIT.CPP
AFXDBCS.CPP
AFXIMPL.H
AFXINL1.CPP
AFXINL2.CPP
AFXINL3.CPP
AFXMEM.CPP
AFXSTATE.CPP
AFXTLS.CPP
AFXTRACE.CPP
APP3D.CPP
APP3DS.CPP
APPCORE.CPP
APPDLG.CPP
APPGRAY.CPP
APPHELP.CPP
APPHELPX.CPP
APPINIT.CPP
APPMODUL.CPP
APPPRNT.CPP
APPTERM.CPP
APPUI.CPP
APPUI1.CPP
APPUI2.CPP
APPUI3.CPP
ARCCORE.CPP
ARCEX.CPP
ARCOBJ.CPP
ARCSTRM.CPP
ARRAY_B.CPP
ARRAY_D.CPP
ARRAY_O.CPP
ARRAY_P.CPP
ARRAY_S.CPP
ARRAY_U.CPP
ARRAY_W.CPP
AUXDATA.CPP
BARCOOL.CPP
BARCORE.CPP
BARDLG.CPP
BARDOCK.CPP
BARSTAT.CPP
BARTOOL.CPP
CCDATA.CPP
CMDTARG.CPP
CTLCACHE.CPP
CTLCONN.CPP
CTLCORE.CPP
CTLDATA.CPP
CTLEVENT.CPP
CTLFONT.CPP
CTLFRAME.CPP
CTLIMPL.H
CTLINL.CPP
CTLINPLC.CPP
CTLINTL.CPP
CTLLIC.CPP
CTLMODUL.CPP
CTLNOWND.CPP
CTLOBJ.CPP
CTLPBAG.CPP
CTLPICT.CPP
CTLPPG.CPP
CTLPROP.CPP
CTLPROPX.CPP
CTLPSET.CPP
CTLPSTG.CPP
CTLPSTM.CPP
CTLQUICK.CPP
CTLREFL.CPP
CTLREG.CPP
CTLTRACK.CPP
CTLVIEW.CPP
DAOCORE.CPP
DAODFX.CPP
DAOIMPL.H
DAOVIEW.CPP
DBCORE.CPP
DBFLT.CPP
DBIMPL.H
DBLONG.CPP
DBRFX.CPP
DBVAR.CPP
DBVIEW.CPP
DCMETA.CPP
DCPREV.CPP
DISPIMPL.H
DLGCLR.CPP
DLGCOMM.CPP
DLGCORE.CPP
DLGDATA.CPP
DLGFILE.CPP
DLGFLOAT.CPP
DLGFNT.CPP
DLGFR.CPP
DLGPRNT.CPP
DLGPROP.CPP
DLGTEMPL.CPP
DLLDB.CPP
DLLINIT.CPP
DLLMODUL.CPP
DLLNET.CPP
DLLOLE.CPP
DOCCORE.CPP
DOCKCONT.CPP
DOCKSTAT.CPP
DOCMAPI.CPP
DOCMGR.CPP
DOCMULTI.CPP
DOCSINGL.CPP
DOCTEMPL.CPP
DUMPCONT.CPP
DUMPFLT.CPP
DUMPINIT.CPP
DUMPOUT.CPP
DUMPSTAK.CPP
ELEMENTS.H
EXCEPT.CPP
FILECORE.CPP
FILEFIND.CPP
FILELIST.CPP
FILEMEM.CPP
FILESHRD.CPP
FILEST.CPP
FILETXT.CPP
FILEX.CPP
FIXALLOC.CPP
FIXALLOC.H
INDICATE.RC
INET.CPP
INETCALL.CPP
ISAPI.CPP
ISAPIMIX.CPP
LIST_O.CPP
LIST_P.CPP
LIST_S.CPP
MAKEFILE
MAP_PP.CPP
MAP_PW.CPP
MAP_SO.CPP
MAP_SP.CPP
MAP_SS.CPP
MAP_WO.CPP
MAP_WP.CPP
MFCDB.MAK
MFCDB.RC
MFCDLL.CLW
MFCDLL.MAK
MFCDLL.RC
MFCINTL.MAK
MFCINTL.RC
MFCISAPI.MAK
MFCNET.MAK
MFCNET.RC
MFCOLE.MAK
MFCOLE.RC
MSCREATE
MTCORE.CPP
MTEX.CPP
NOLIB.CPP
OBJCORE.CPP
OCCCONT.CPP
OCCDDX.CPP
OCCDDXF.CPP
OCCDLG.CPP
OCCEVENT.CPP
OCCIMPL.H
OCCLOCK.CPP
OCCMGR.CPP
OCCSITE.CPP
OCDB.H
OCDBID.H
OLEASMON.CPP
OLEBAR.CPP
OLEBIND.H
OLECALL.CPP
OLECLI1.CPP
OLECLI2.CPP
OLECLI3.CPP
OLECNVRT.CPP
OLECONN.CPP
OLEDATA.CPP
OLEDISP1.CPP
OLEDISP2.CPP
OLEDLGS1.CPP
OLEDLGS2.CPP
OLEDLGS3.CPP
OLEDLL.CPP
OLEDOBJ1.CPP
OLEDOBJ2.CPP
OLEDOC1.CPP
OLEDOC2.CPP
OLEDOCCL.CPP
OLEDOCIP.CPP
OLEDOCOB.CPP
OLEDOCTG.CPP
OLEDOCVW.CPP
OLEDROP1.CPP
OLEDROP2.CPP
OLEENUM.CPP
OLEEXP.CPP
OLEFACT.CPP
OLEIMPL2.H
OLEINIT.CPP
OLEIPFRM.CPP
OLELINK.CPP
OLELOCK.CPP
OLEMISC.CPP
OLEMON.CPP
OLEMSGF.CPP
OLEPSET.CPP
OLEREG.CPP
OLESTRM.CPP
OLESVR1.CPP
OLESVR2.CPP
OLETSVR.CPP
OLETYPLB.CPP
OLEUI1.CPP
OLEUI2.CPP
OLEUNK.CPP
OLEVAR.CPP
OLEVAR1.CPP
OLEVERB.CPP
PCHMARK
PLEX.CPP
PPGCOLOR.CPP
PPGFONT.CPP
PPGPICT.CPP
PPGSTOCK.CPP
PROMPTS.RC
README
SOCKCORE.CPP
SOCKIMPL.H
STDAFX.CPP
STDAFX.H
STRCORE.CPP
STREX.CPP
THRDCORE.CPP
TIMECORE.CPP
TOOLTIP.CPP
TRCKRECT.CPP
VALIDADD.CPP
VIEWCMN.CPP
VIEWCORE.CPP
VIEWEDIT.CPP
VIEWFORM.CPP
VIEWHTML.CPP
VIEWOLED.CPP
VIEWPREV.CPP
VIEWPRNT.CPP
VIEWRICH.CPP
VIEWSCRL.CPP
WINBTN.CPP
WINCORE.CPP
WINCTRL1.CPP
WINCTRL2.CPP
WINCTRL3.CPP
WINCTRL4.CPP
WINCTRL5.CPP
WINCTRL6.CPP
WINCTRL7.CPP
WINFRM.CPP
WINFRM2.CPP
WINFRMX.CPP
WINGDI.CPP
WINGDIX.CPP
WINHAND.CPP
WINHAND_.H
WINMAIN.CPP
WINMDI.CPP
WINMENU.CPP
WINMINI.CPP
WINOCC.CPP
WINSPLIT.CPP
WINSTR.CPP
WINUTIL.CPP
289 个对象
3.77 MB
我的电脑


t
ication Architecture
dTarget
WinThread
CWinApp
COleControlModule
user application
DocTemplate
CSingleDocTemplate
CMultiDocTemplate
DleObjectFactory
COleTemplateServer
DleDataSource
DleDropSource
DleDropTarget
DleMessageFilter
ConnectionPoint
CDocument
COleDocument
COleLinkingDoc
COleServerDoc
CRichEditDoc
user documents
CDocItem
COleClientItem
CRichEditCntrItem
user client items
COleServerItem
CDocObjectServerItem
user server items
CDocObjectServer
user objects
Exceptions
CException
CArchiveException
CDaoException
CDBException
CFileException
CInternetException
CMemoryException
CNotSupportedException
COleException
COleDispatchException
CResourceException
CUserException
File Services
CFile
CMemFile
CSharedFile
COleStreamFile
CMonikerFile
CAsyncMonikerFile
CDataPathProperty
CCachedDataPathProperty
CSocketFile
CStdioFile
CInternetFile
CGopherFile
CHttpFile
CRecentFileList
Graphical Drawing
CDC
CClientDC
CMetaFileDC
CPaintDC
CWindowDC
Control Support
CDockState
CImageList
Graphical Drawing Objects
CGdiObject
CBitmap
CBrush
CFont
CPalette
CPen
CRgn
Menus
CMenu
ODBC Database Support
CDatabase
CRecordset
user recordsets
CLongBinary
DAO Database Support
CDaoDatabase
CDaoQueryDef
CDaoRecordset
CDaoTableDef
CDaoWorkspace
Synchronization
CSyncObject
CCriticalSection
CEvent
CMutex
CSemaphore
Windows Sockets
CAsyncSocket
CSocket
Arrays
CArray (template)
CByteArray
CDWordArray
CObArray
CPtrArray
CStringArray
CUIntArray
CWordArray
arrays of user types
Lists
CList (template)
CPtrList
CObList
CStringList
lists of user types
Maps
CMap (template)
CMapWordToPtr
CMapPtrToWord
CMapPtrToPtr
CMapWordToOb
CMapStringToPtr
CMapStringToOb
CMapStringToString
maps of user types
Internet Services
CInternetSession
CInternetConnection
CFtpConnection
CGopherConnection
CHttpConnection
CFileFind
CFtpFileFind
CGopherFileFind
CGopherLocator
indow Support
Wnd
Frame Windows
CFrameWnd
CMDIChildWnd
user MDI windows
CMDIFrameWnd
user MDI workspaces
CMiniFrameWnd
user SDI windows
COleIPFrameWnd
CSplitterWnd
Control Bars
CControlBar
CDialogBar
COleResizeBar
CStatusBar
CToolBar
Property Sheets
CPropertySheet
Dialog Boxes
CDialog
CCommonDialog
COleDialog
COleBusyDialog
COleChangeIconDialog
COleChangeSourceDialog
COleConvertDialog
COleInsertDialog
COleLinksDialog
COleUpdateDialog
COlePasteSpecialDialog
COlePropertiesDialog
CColorDialog
CFileDialog
CFindReplaceDialog
CFontDialog
CPageSetupDialog
CPrintDialog
COlePropertyPage
CPropertyPage
user dialog boxes
Views
CView
CCtrlView
CEditView
CListView
CRichEditView
CTreeView
CScrollView
user scroll views
CFormView
user form views
CDaoRecordView
CRecordView
user record views
Controls
CAnimateCtrl
CButton
CBitmapButton
CComboBox
CEdit
CHeaderCtrl
CHotKeyCtrl
CListBox
CCheckListBox
CDragListBox
CListCtrl
COleControl
CProgressCtrl
CRichEditCtrl
CScrollBar
CSliderCtrl
CSpinButtonCtrl
CStatic
CStatusBarCtrl
CTabCtrl
CToolBarCtrl
CToolTipCtrl
Classes Not Derived from CObject
Internet Server API
CHtmlStream
CHttpFilter
CHttpFilterContext
CHttpServer
CHttpServerContext
Run-time Object Model Support
CArchive
CDumpContext
CRuntimeClass
Simple Value Types
CPoint
CRect
CSize
CString
CTime
CTimeSpan
Structures
CCommandLineInfo
CCreateContext
CMemoryState
COleSafeArray
CPrintInfo
Support Classes
CCmdUI
COleCmdUI
CDaoFieldExchange
CDataExchange
CDBVariant
CFieldExchange
COleDataObject
COleDispatchDriver
CPropExchange
CRectTracker
CWaitCursor
Typed Template Collections
CTypedPtrArray
CTypedPtrList
CTypedPtrMap
OLE Type Wrappe
CFontHolder
CPictureHolder
OLE Automation T
COleCurrency
COleDateTime
COleDateTimeSpan
COleVariant
Synchronization
CMultiLock
CSingleLock

文档-视结构

- 了解Doc-View结构的思想
- CView
- CDocument
- CFrameWnd
- CWinApp
- CDocTemplate

举一个例子：类厂的实现

- 在源码中搜索IClassFactory
  - 发现，COleObjectFactory实现了此接口
- 找到COleObjectFactory的定义
  - 技巧：用析构函数
- 在CreateInstance函数中调用到了OnCreateObject成员函数
  - 并且创建的时候用到了m_pRuntimeClass
  - 而这是在构造函数中对m_pRuntimeClass赋值的
- 谁来调用构造函数？
  - 线索断了？

类厂实现

- 从DllGetClassObject入手
  - 找到AfxDllGetClassObject
- 阅读AfxDllGetClassObject的代码
  - 我们可以发现，有一个factoryList
- 我们猜想这样的factoryList是全局存在的
  - 确实如此：存在于pModuleState之中
  - AFX_MODULE_STATE->m_factoryList
  - 关于AFX_MODULE_STATE省略

继续查找m_factoryList的使用情况

- 在COleObjectFactory构造函数中发现
  - pModuleState->m_factoryList.AddHead(this);
- 在析构函数中
  - pModuleState->m_factoryList.Remove(this);
- 相当于由COleObjectFactory自维护的
- 现在的问题是：
  - 谁来构造COleObjectFactory对象？

将类厂与COM对象联系起来

- 创建一个支持COM的测试工程
- 在COM对象类中发现两个宏
  - DECLARE_OLECREATE(CMyObject)
  - IMPLEMENT_OLECREATE(CMyObject, …

```
#define DECLARE_OLECREATE(class_name) \
public: \
    static AFX_DATA COleObjectFactory factory; \
    static AFX_DATA const GUID guid; \

#define IMPLEMENT_OLECREATE(class_name, external_name, l, w1, w2, b1, b2, b3, b4
    AFX_DATADEF COleObjectFactory class_name::factory(class_name::guid, \
        RUNTIME_CLASS(class_name), FALSE, _T(external_name)); \
    AFX_COMDAT const AFX_DATADEF GUID class_name::guid = \
        { l, w1, w2, { b1, b2, b3, b4, b5, b6, b7, b8 } }; \
```

ATL举例

- ATL怎么实现多线程支持
- 做法：
  - 创建一个测试工程
  - 插入两个ATL对象，分别支持单线程、多线程
  - 然后看这两个对象的代码
  - 找到CComSingleThreadModel和CComMultiThreadModel
  - 找到这两个类的定义，发现只有两个静态函数

两个实现线程模型的ATL类

```
class CComMultiThreadModel
{
public:
    static ULONG WINAPI Increment(LPLONG p) {return InterlockedIncrement(p);}
    static ULONG WINAPI Decrement(LPLONG p) {return InterlockedDecrement(p);}
    typedef CComAutoCriticalSection AutoCriticalSection;
    typedef CComCriticalSection CriticalSection;
    typedef CComMultiThreadModelNoCS ThreadModelNoCS;
};

class CComSingleThreadModel
{
public:
    static ULONG WINAPI Increment(LPLONG p) {return ++(*p);}
    static ULONG WINAPI Decrement(LPLONG p) {return --(*p);}
    typedef CComFakeCriticalSection AutoCriticalSection;
    typedef CComFakeCriticalSection CriticalSection;
    typedef CComSingleThreadModel ThreadModelNoCS;
};
```

结合线程类和对象类

```
class ATL_NO_VTABLE CATLObject1 :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CATLObject1, &CLSID_ATLObject1>,
    public IDispatchImpl<IATLObject1, &IID_IATLObject1, &LIBID_ATLTESTLib>
{
```

```
class ATL_NO_VTABLE CATLObject2 :
    public CComObjectRootEx<CComMultiThreadModel>,
    public CComCoClass<CATLObject2, &CLSID_ATLObject2>,
    public IDispatchImpl<IATLObject2, &IID_IATLObject2, &LIBID_ATLTESTLib>
{
```

- 我们需要找到CComObjectRootEx的定义

CComObjectRootEx的定义

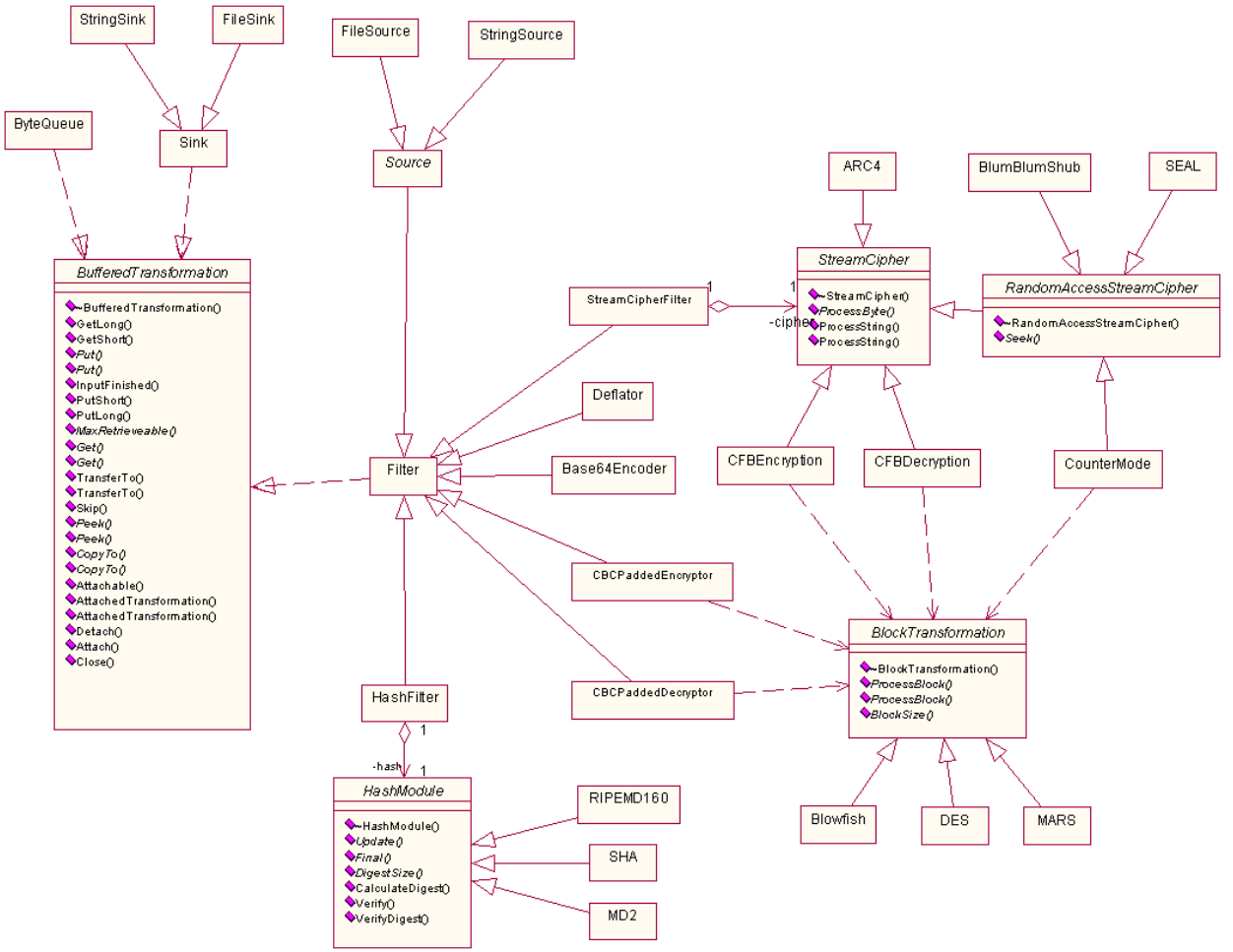
```
template <class ThreadModel>
class CComObjectRootEx : public CComObjectRootBase
{
public:
    typedef ThreadModel _ThreadModel;
    typedef _ThreadModel::AutoCriticalSection _CritSec;
    typedef CComObjectLockT<_ThreadModel> ObjectLock;

    ULONG InternalAddRef()
    {
        ATLASSERT(m_dwRef != -1L);
        return _ThreadModel::Increment(&m_dwRef);
    }
    ULONG InternalRelease()
    {
        ATLASSERT(m_dwRef > 0);
        return _ThreadModel::Decrement(&m_dwRef);
    }
```

小结一下

- 线程模型作为对象类中的一个类型
- 该类型中定义了两个静态函数(加一和减一)
  - 通过typedef把不同的静态函数映射到同一个名字的类型上
- 实际上，线程模型还有其他一些用途
- 用户以模板参数的形式指定线程模型
  - 这是一种策略模式(Strategy Pattern)
- 背景知识
  - Strategy Pattern、template、namespace

关于Crypto++

- 利用有限的文档



分成几大部分

- 对称密钥部分
- 消息摘要部分
- 公钥部分
- Source/sink/filter
- 其他

阅读方法

- 按照UML类图结构，大致了解整体的思路
- 看一下文件的组织情况
- 大致浏览几个文件，看一下代码的风格
- 针对每一部分，看一下每个类有哪些成员，这些成员的大致关系如何
- 利用test工程，运行
  - 跟踪和调试
  - 逐步深入了解细节
- 总结和笔记

Linux源码阅读

- Linux是一个操作系统
  - 系统内核
  - 系统调用
  - 系统工具
  - ……
- 资源比较丰富
  - 所以，事先找一本合适的参考书了解一下
- 最好先装一下用一用，感觉一下
  - 建议找一本介绍Linux使用的书看一看，并实践一下
  - 目的是建立起总体的印象

Linux内核

- 几大部分
  - 进程调度
    - 包括进程的基本结构、调度的算法等等
  - 内存管理
    - 需要了解底层硬件的内存管理机制，以及Linux的虚存管理技术
  - 文件系统
    - Linux有一套虚拟文件系统，以及extX文件系统
  - 设备管理
    - 特别是块设备、字符设备和网络设备，同时还涉及到Linux的模块扩展机制
  - 系统调用如何实现，即应用程序和系统如何接口

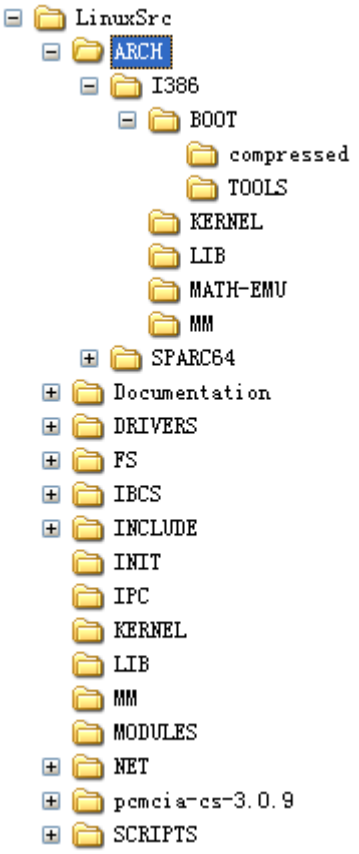
让Linux活起来

- 从机器启动引导，到Linux接管控制
- 各种初始化工作
  - 包括系统内核的初始化，例如init进程、各种内部表格的初始化，等等
- 一直到login和shell
- 试着编译一下Linux
- 如果有可能的话，设法调试Linux(?)
- 脉络清晰了吗？有没有把Linux据为己有的快感？

Linux其他

- 除了系统本身之外，Linux是一个丰富的源码资源
  - 还有网络的协议栈(内含包过滤防火墙)
  - Linux各种命令，包括网络服务、网络客户端工具等
  - 编译器的实现(gcc)
  - 编辑器的实现(emacs)
  - ……

Linux中的入口




```
struct task_struct {
/* these are hardcoded - don't touch */
    volatile long state;    /* -1 unrunnable,
    unsigned long flags;    /* per process fla
    int sigpending;
    mm_segment_t addr_limit;    /* thread addr
                        0-0xBFFFFFFF for user-
                        0-0xFFFFFFFF for kerne
                    */
    struct exec_domain *exec_domain;
    long need_resched;

/* various fields */
    long counter;
    long priority;
    cycles_t avg_slice;
/* SMP and runqueue state */
    int has_cpu;
    int processor;
```

OpenBSD

```
/*
 * Thread structure.
 */
struct pthread {
    /*
     * Magic value to help recognize a valid thread structure
     * from an invalid one:
     */
#define PTHREAD_MAGIC       ((u_int32_t) 0xd09ba115)
    u_int32_t       magic;
    char            *name;

    /*
     * Lock for accesses to this thread structure.
     */
    spinlock_t      lock;

    /* Queue entry for list of all threads: */
```

用Linux指导OpenBSD

- OpenBSD也是源码开放的操作系统
- OpenBSD的源码资料相对比较少
- 但是，两者同属于UNIX-like OS
- 用Linux的经验来指导OpenBSD的学习
  - 态度：我们可能会碰上一些困难，要有思想准备
  - 做法：从内核开始，到系统的启动和初始化
    然后逐个分析重要模块，特别是跟系统安全和网络安全相关的模块，比如防火墙、IPSec、Kerberos、文件系统保护，等等

参考材料

- 侯捷，“源码追踪经验谈”，《程序员》杂志
- MFC源码
- ATL源码
- Linux源码
- ……