*The X Developing*

开发者指南

张文权

[powersuite@hotmail.com](mailto:powersuite@hotmail.com)

中国浙江杭州

<https://github.com/jdzwq>

2018年9月

目 录

[*导言* 5](#_Toc524229773)

[1.框架概览 7](#_Toc524229774)

[1.1.平台依赖库（XDL For Plateform Kit） 7](#_Toc524229775)

[1.1.1内存管理。 8](#_Toc524229776)

[1.1.2.线程和子进程。 8](#_Toc524229777)

[1.1.3.原语对象。 8](#_Toc524229778)

[1.1.4.输入输出（IO）对象。 9](#_Toc524229779)

[1.1.5.套接字（socket）。 9](#_Toc524229780)

[1.1.6.图形资源（GDI）。 9](#_Toc524229781)

[1.1.7.窗体资源（Window）。 10](#_Toc524229782)

[1.1.8.Shell。 10](#_Toc524229783)

[1.2.第三方开源库（Open Source Lib） 10](#_Toc524229784)

[1.2.1.Crypt Lib。 10](#_Toc524229785)

[1.1.2.Hash Lib。 11](#_Toc524229786)

[1.2.3.Zip Lib。 11](#_Toc524229787)

[1.2.4.Code Lib。 11](#_Toc524229788)

[1.2.5.Statistics Lib。 11](#_Toc524229789)

[1.3.基本数据结构（XDL Base Struct） 12](#_Toc524229790)

[1.3.1.vector。 12](#_Toc524229791)

[1.3.2.matrix。 12](#_Toc524229792)

[1.3.3.set。 12](#_Toc524229793)

[1.3.4.map。 12](#_Toc524229794)

[1.3.5.stack。 12](#_Toc524229795)

[1.3.6.list。 12](#_Toc524229796)

[1.3.7.dict table。 12](#_Toc524229797)

[1.3.8.hash table。 13](#_Toc524229798)

[1.3.9.ac table。 13](#_Toc524229799)

[1.3.10.file table。 13](#_Toc524229800)

[1.3.11.bina tree。 13](#_Toc524229801)

[1.3.12.bplus tree。 13](#_Toc524229802)

[1.3.13.multi tree。 13](#_Toc524229803)

[1.4.基本对象（XDL Base Object） 13](#_Toc524229804)

[1.4.1.string。 13](#_Toc524229805)

[1.4.2.variant。 14](#_Toc524229806)

[1.4.3.object。 14](#_Toc524229807)

[1.4.4.stream。 14](#_Toc524229808)

[1.4.5.canvas。 14](#_Toc524229809)

[1.5.网络协议栈（XDL Protocol Stack） 14](#_Toc524229810)

[1.5.1.UDP。 14](#_Toc524229811)

[1.5.2.TFTP。 15](#_Toc524229812)

[1.5.2.TCP。 15](#_Toc524229813)

[1.5.3.SSL。 15](#_Toc524229814)

[1.5.4.HTTP。 15](#_Toc524229815)

[1.5.5.HTTPS。 15](#_Toc524229816)

[1.5.6.RESTful。 15](#_Toc524229817)

[1.5.7.SOAP。 16](#_Toc524229818)

[1.5.8.DICOM。 16](#_Toc524229819)

[1.5.9.HL7。 16](#_Toc524229820)

[1.6.文档和视图（XDL Document/View） 16](#_Toc524229821)

[1.6.1.DOM文档 16](#_Toc524229822)

[1.6.2.XML文档 17](#_Toc524229823)

[1.6.3.JSON文档 17](#_Toc524229824)

[1.6.4.SVG文档 17](#_Toc524229825)

[1.6.5.SOAP/WSDL文档 17](#_Toc524229826)

[1.6.6.SCHEMA文档 17](#_Toc524229827)

[1.6.7.META文档 17](#_Toc524229828)

[1.6.8.Form文档／视图 18](#_Toc524229829)

[1.6.9.Grid文档／视图 18](#_Toc524229830)

[1.6.10.Graph文档／视图 18](#_Toc524229831)

[1.6.11.Proper文档／视图 18](#_Toc524229832)

[1.6.12.Topog文档／视图 18](#_Toc524229833)

[1.6.13.Images文档／视图 19](#_Toc524229834)

[1.6.14.Label文档／视图 19](#_Toc524229835)

[1.6.15.List文档／视图 19](#_Toc524229836)

[1.6.16.Tree文档／视图 19](#_Toc524229837)

[1.6.17.Tool文档／视图 19](#_Toc524229838)

[1.6.18.Ttile文档／视图 19](#_Toc524229839)

[1.6. 19.Status文档／视图 19](#_Toc524229840)

[1.6.20.Memo文档／视图 20](#_Toc524229841)

[1.6.21.Tag文档／视图 20](#_Toc524229842)

[1.6.22.Rich文档／视图 20](#_Toc524229843)

[1.6.23.Anno文档／视图 20](#_Toc524229844)

[1.6.24.Menu文档／视图 20](#_Toc524229845)

[1.6.25.Dialog文档／视图 20](#_Toc524229846)

[1.7.服务开发框架（XDL For Service Framework） 20](#_Toc524229847)

[1.7.1.PNP框架 20](#_Toc524229848)

[1.7.2.TCP框架 21](#_Toc524229849)

[1.7.3.HTTP框架 21](#_Toc524229850)

[1.7.4.OSS框架 21](#_Toc524229851)

[1.7.5.XDB框架 21](#_Toc524229852)

[1.7.6.PACS框架 21](#_Toc524229853)

[1.7.7.HL7框架 21](#_Toc524229854)

[1.8.桌面开发框架（XDL Desktop Framework） 21](#_Toc524229855)

[1.8.1.Box小部件 21](#_Toc524229856)

[1.8.2.Editor编辑件 21](#_Toc524229857)

[1.8.3.Control控件 22](#_Toc524229858)

[1.8.4.Menu控件 22](#_Toc524229859)

[1.8.5.Dialog控件 22](#_Toc524229860)

[1.8.6.Frame框架窗体 22](#_Toc524229861)

# *导言*

X Developing是应用软件全栈开发框架，它旨在为传输、存储协议、算法设计、设备驱动、服务应用和桌面应用提供开发库和应用框架生成工具。同时X Developing也是跨平台的开发体系，它对Windows，MacOS，Linux平台均提供良好的支持。它符合多层次、跨平台两个全栈开发的特性。

众所周知，作为应用开发语言，Java比较适合WEB服务端应用开发，C#主要用于构建桌面应用，Python则更擅长于处理数理统计，而对于C语言，通常认为只适合作为操作系统、硬件驱动、通讯协议等底层软件设施的开发语言。对于上层应用软件，很少采用C语言作为主力开发语言，其原因固然有C语言和其开发库本身存在短板，也有在言必面向对象思潮的作祟下对C语言的弃用。

C语言的简洁、高效是毋庸置疑的，这也是操作系统、硬件驱动、协议栈用其作为主要开发语言的原因。但C开发库也存在两个明显的短处，内存管控能力和函数安全机制。C开发库使用内存方式灵活，堆、栈、虚拟内存形式多样，但管控机制很弱，常提的“内存泄漏”就是很头疼的事。鉴于此，大多数面向对象的语言都实现了内存回收机制，内存使用更为安全，这大概也是应用开发拥抱面向对象语言的主要原因。现存C开发基本库中，函数的安全机制比较差，主要表现为不可控的越界读写，这会导致设计错误在此处隐藏，运行错误在彼处引发的情况，即“Bug漂移”。针对内存管控和函数安全的苛刻性要求，X Developing开发库给出了自己的解决办法。

X Developing开发库采用C语言构建，不过摈弃了大多数C标准库，但充分保留了C语言机制，如：跳转机制、断言机制。X Developing将内存、临界区、信号灯、事件对象、定时器、线程、文件、管道、图形、窗体、控制台、Shell等界定为系统资源，X Developing对系统资源访问只负责定义接口，不负责实现，具体的实现交由各平台原生的SDK库，而不是C标准库的运行时函数。

对于内存泄漏的管控，X Developing认为内存的高频、巨额使用通常发生在工作者线程内，而且工作者线程总是频繁地创建、执行和销毁。给出解决办法是为每个线程（主线程和工作者线程）派发私有内存堆，各线程运行过程中无论申请和释放内存是否匹配，只要线程结束，私有内存堆总是会被释放，这样可以保证线程运行结束后不至于产生内存泄漏。不过我们不能假设线程（特别是工作者线程）都是正常销毁的，相反，崩溃是工作者线程的常态。X Developing充分使用了断言和跳转机制来对线程进行异常执行保护，捕获线程崩溃，获得释放线程内存私有堆的机会。

在跨平台方面，解释性语言的可移植性没有任何问题，因为可移植性是由虚拟机或解释执行器来实现，并不归由语言的本身。编译型语言的可移植性要差的多，很多Linux C语言项目需要通过Cygwin这样的环境来辅助向Windows平台移植。即便有了Cygwin这样的编译虚拟机，事实上移植的程序会丧失一些宿主平台SDK特性。而X Developing开发库会充分保留各自平台的SDK特性，并且不依赖于编译虚拟机来实现跨平台移植。

操作系统的嵌入化、微型化是个重要的趋势，系统的微型化意味着可用的系统对象资源越来越少，X Developing正是基于此保持开发库的可裁剪性，设计之初就定义了裁剪路线。

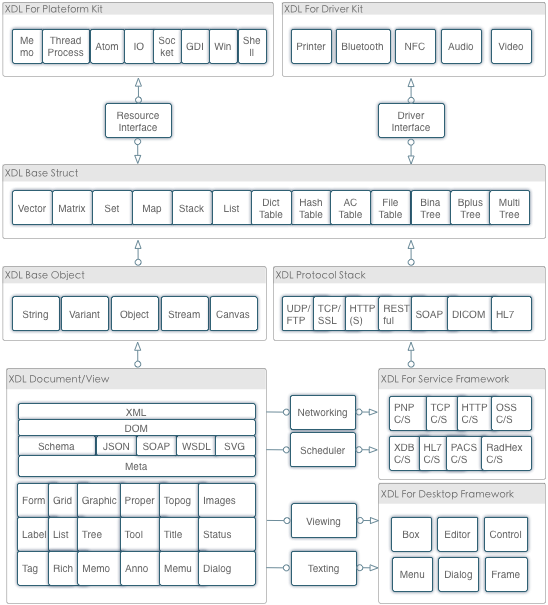
熟悉桌面应用的开发者都清楚，无论是Linux平台还是Windows平台，都具备多种图形库。这些图形库多数是为了保持系统应用向前兼容而保留下来。从客观上讲，我们开发的应用也应该保留向前兼容的特性，但如果选用了特定的图形库，向前兼容就未必行得通。X Developing定义了动态加载资源访问的设计框架，其做法是定义资源访问的一致性接口，然后程序运行时根据系统版本环境来加载合适的资源访问库。

X Developing体系包含了开发库集合（XDK、XDL、XDD、XDC），xDesign，xService和xConsole。XDK是平台依赖库，是各操作系统专属的资源访问SDK库封装。XDL是各系统平台一致的数据结构、算法、文档、视图开发库。XDD是各系统平台专属的驱动开发支持库。XDC是各平台一致的框架、窗体、控件、编辑部件、对话框的开发库。xDesign是独立于开发库的XML表单、对话框和文档设计器，用于设计程序运行时的可视化资源。xService是服务程序，它实现了HTTP(s)、TCP、UDP端口服务体系，提供了诸如WEB、共享文件、对象存储、数据库网关、内存数据库、消息队列、物联网络等通用类的服务以及一些HL7 Express、DICOM等专业类的服务。xConsole是运行监视控制台，它为xService提供了性能监视和错误追溯。

X Developing是开源项目，项目工程位于：<https://github.com/jdzwq/Easily-SDK-5.git>。

# 框架概览

X Developing由平台依赖库（XDK），设备驱动支持库（XDD），结构、算法、基本对象、协议栈、文档与视图开发库（XDL）,服务开发框架（XDS），桌面开发框架（XDC）五个层次成。



平台依赖库（XDK）由是系统资源访问库，这些资源包括进程线程、原语对象，IO对象，套接字、图形资源、窗体资源和Shell。各平台的SDK为系统资源访问提供了函数入口，SDK是操作系统专属的，且概念和形态各异。XDK为各平台的SDK做了轻量级的封装，并定义了一致的接口，即每个接口维护了一组函数集，面向XDL提供所需系统资源的访问入口。

驱动支持库（XDD）是为设备驱动开发提供的支持库，由于各操作系统的驱动程序开发与应用开发存在很大差异，所以一般提供独立的DDK开发库来支持，他提供了操作系统内核层次上的资源访问和管理函数入口，XDD同样也为这些内核的资源定了访问接口，并接驳XDL，为应用级的驱动开发提供支持。

XDL是独立于各系统平台的数据结构、算法、基本对象、存储传输协议栈、文档及视图开发库。在X Developing开发体系中，XDL一方面通过动态加载XDK、XDD为访问系统资源提供入口，另一方面为开发服务类应用（XDS）和桌面类应用（XDC）给出了文档结构、视图形式、传输协议、存储规格的定义和实现。

XDS是服务应用开发的支持库，在传输与存储、缓存、代理等网络服务中，它为服务应用构造两种角色服务，即会话（Networking）和调度（Scheduler）。

XDC是桌面应用的开发支持库，它为桌面程序提供了在运行时构造主框架、子窗体、编辑件、对话框的能力，这些部件即为容器，为XDL文档与视图提供了可视机制（Viewing），也为控制器，将结构型XDL文档转换为文本流（Texting）的方式与用户进行交互。

# 数据类型

## 沿用C语言的类型

int/unsigned int：4字节的整型和无符号整型。XDL用其表示状态值，或字符串（多子节、双字节）的长度。

long/unsigned long：4字节的整型和无符号整型。XDL用其表示整型坐标、尺寸大小。

float：4字节的浮点数，XDL用其表示浮点小数，浮点型的坐标、尺寸大小。

double：8字节的浮点数，XDL主要用其表示高精度的浮点小数。

## 系统资源标识符类型

res\_glob\_t：等同于平台SDK的全局内存句柄。

res\_heap\_t：等同于平台SDK的堆句柄。

res\_file\_t：等同于平台SDK的文件句柄或标识符。

res\_even\_t：等同于平台SDK的事件句柄或标识符。

res\_mutx\_t：等同于平台SDK的互斥锁句柄或标识符。

res\_crit\_t：等同于平台SDK的临界区句柄或标识符。

res\_sema\_t：等同于平台SDK的信号灯句柄或标识符。

res\_ctx\_t：等同于图形场景描述符。

res\_font\_t：等同于字体对象描述符。

res\_bmp\_t：等同于位图对象描述符。

res\_rgn\_t：等同于区块对象描述符。

res\_acl\_t：等同于快捷映射描述符。

res\_win\_t：等同于窗体对象描述符。

## XDL的简单类型

byte\_t：1字节等同于unsigned char，表示字节。

schar\_t：1字节等同于char，表示多字节字符串。

wchar\_t：等同于WCHAR，windows平台为2字节，linux平台为4字节，表示双节字符串。

bool\_t：布尔类型，值1为真，值0为假，32位平台为2字节，64位平台为4字节。

sword\_t：2字节无符号整型。大都用其作移位操作。

dword\_t：4字节无符号整型。大都用其表示字节数，以及移位操作。

lword\_t：8字节的无符号整型。大都用其作移位操作。

var\_long：可变长的整型，32位平台为4字节等同于long，64位平台为8字节等同于long long。大都用作传递整型参数或指针参数。

stamp\_t：8字节无符号整型，表示时间戳。

key32\_t：4字节符号整型，表示32位的哈希值。

key64\_t：8字节符号整型，表示64位的哈希值。

key128\_t：16字节符号整型，表示128位的哈希值。

## XDL的基本结构

xhand\_t：XDL的句柄结构，用于对系统资源标识符进行统一的封装。

link\_t：连接件，是XDL中最重要的数据结构，数据部件都以其作为联结点，构成线性、树型等文档体。连接件也是访问文档体或子部件的标识符。

vector\_t：向量结构，支持向量的基本操作。

matrix\_t：矩阵结构，支持矩阵的基本操作。

set\_t：集合结构，支持集合的基本操作。

map\_t：位图结构，支持位图的基本操作。

## XDL的基本对象

string\_t：文本对象，维护一个可变长的字符串数组。

variant\_t：变体对象，维护一个多种XDL简单类型及数组的联合体，并支持简单类型字符串化。

stream\_t：流对象，为各种系统IO资源提供流式读写操作，并支持编码方式自动转换。

object\_t：序列化对象，支持XDL数据结构及文档体与存储对象进行序列化和反序列化操作。

canvas\_t：画布对象，为图形操作维护一个内存像素位图。支持像素点阵坐标到米制坐标的映射和转换。

file\_t：文件对象，为本地路径、网络路径、http、ftp、oss、文件提供统一的读写操作。

# 平台依赖库（XDL For Plateform Kit）

## 内存管理。

XDL实现了四种内存管理方式的函数集，即：堆内存、页内存、全局内存和缓存。**堆内存（xmem）**为程序在运行时提供了按需使用和释放内存的机制，但从何种堆里分配内存，取决于XDL的启动标志，如果是线程寓所模式（XDL\_THREAD\_APARTMENT）,所需内存从线程私有堆里分配，如果是进程寓所模式（XDL\_PROCESS\_APARTMENT）,则从进程默认堆里分配。**页内存（pmen）**是一种以页面大小（4096字节）作为最小分配和释放单位的内存管理机制，页内存可以单页、也可以连续页的方式分配，页可以被锁定常驻于物理内存，但这样会占用宝贵的内存资源，所以锁定操作务必慎重，根据应用一些特殊性需要，可以对页进行只读保护。**全局内存（gmem）**主要用于进程之间共享数据而实现的内存管理机制，常用于剪切板操作。**缓存（vmem）**为用户提供了一个逻辑连续的虚拟线性地址空间，然后采用内存映射（MapView）的方式来读写数据，32位和64位操作系统，vmem线性空间大小分别为2G和4G。

## 线程和子进程。

XDL提供了两种独立的执行单元，即：子进程和线程。传统的线程没有独立的堆内存空间，但如果XDL的启动方式为线程寓所模式（XDL\_THREAD\_APARTMENT），则线程也拥有独立的堆内存空间，可以通过堆内存（xmem）函数来申请、释放。

**子进程（process）**的创建需依赖于一个可执行文件，子进程被创建后即独立运行。XDL为父进程和子进程的通信提供了三种途径，一是通过命令参数（command line）方式传入，二是父子进程建立双工的管道（SHARE\_PIPE）或套接字（SHARE\_SOCK）进行通信，三是通过跨进程分配内存空间（process\_alloc）来传递数据。子进程通常也需要从运行环境中获取一些用户配置信息，XDL提供了profile的读写函数，profile通常为独立的配置文件或系统注册表存储。

**线程（xthread）**通常以一个工作函数（PF\_THREAD\_FUNC）为其执行体，线程被创建后即执行。父进程在创建线程时可以通过为工作函数传入一个参数作为通信的基础，在线程执行过程中，双方可以通过共同维护信号量或互斥体来编排执行秩序。线程也可以通过join方法加入父进程的维护队列，父进程将等待队列中所有线程结束后才终止。

## 原语对象。

系统的原语对象主要用于竞争性资源的排它性控制，线程执行顺序编排和异步操作调度，XDL提供的原语对象有：临界区、互斥锁、事件、信号灯、异步队列、定时器。

**临界区（xcrit）**用于线程级的代码排它性执行控制，获准进入临界区的代码被执行，未获准进入的代码将等待。

**互斥锁（xmutex）**用于进程级的排他行执行控制，获得互斥锁的进程被执行，其他进程将等待。

**事件（xevent）**维护着一个触发态和非触发态的状态量，事件的状态可以按需重置，主要用于编排多个线程的执行顺序，或者用于线程向父进程报告执行状态。

**信号灯（xsemap）**用于进程级的有限资源使用控制，信号灯使用一个信号量表示资源可用数，对信号量的P操作（信号量减1）表示申请资源，V操作（信号量加1）表示释放资源。P操作成功的进程予以执行，否则将等待。

**异步队列（xqueue）**主要用于将一个IO资源句柄投放到系统异步IO队列中，当该句柄相关的IO读写操作可用时，异步队列返回该句柄，读写线程从等待状态中恢复执行。

**定时器（xtimer）**主要用于。

## 输入输出（IO）对象。

控制台（Console）、管道（Pipe）、通信端口（COM）、共享内存（Share），文件（File）都属于操作系统平台的输入输出资源。

**控制台（xcons）**是系统特殊的输入输出对象，它以STDIN为输入系统资源句柄，STDOUT为输出系统资源句柄。一个进程通常只能拥有一个控制台，XDL除了支持控制台的读写操作外，还可以为控制台安装信号中断处理例程，如Ctrl+C中断信号、控制台退出信号等。

**管道（xpipe）**是进程之间通信的常用机制，通常数据在管道的一端写入，而后在另一端读出。XDL提供了命名管道的实现，采用Client/Server的方式交互，首先需要创建一个命名管道服务方（pipe\_srv）,然后可以创建多个管道客户方（pipe\_cli）等待管道服务可用，连接后进行一对一读写。

**通信端口（xcomm）**用于和符合RS-232/RS-485通讯规程的仪器进行数据交换。XDL支持两种读写通信端口的方式，一是事件驱动方式，即监听通信端口事件，根据事件类型采取相应的读写动作。二是通过文件流的方式进行直接读写。

**共享内存（xshare）**也是进程之间共享数据比较高效的一种方式，XDL提供了命名共享内存的实现，也是采用Client/Server的方式交互，当一个命名共享内存服务方（share\_srv）创建后，可以被多个客户端（share\_cli）读写，独占性写入共享内存操作由同名互斥锁控制。

**文件（xfile）**是磁盘数据的组织形式，XDL为本地文件、网络路径文件、http文件、tftp文件、oss文件定义了统一的读写形式，提供了同步和异步两种读写机制，除了支持常规的顺序读写外，还支持区块随机读写。

## 套接字（socket）。

套接字是系统进行网络通讯的基本IO输入输出对象，套接字有三种类型，即流套接字（socket-stream），数据报套接字（socket-dgram）和原始套接字（socket-raw）。

XDL中可以创建和操作tcp、udp、icmp三种套接字，其中tcp套接字面向连接，并按网络编程模型分为三种功能套接字：侦听套接字、服务套接字和客户套接字。客户套接字负责和服务主机侦听建立连接，而后为客户端应用收发网络数据。侦听套接字负责监听通讯端口，等待客户套接字连接申请，并负责创建服务套接字。服务套接字用于服务端应用和客户套接字进行数据交互。XDL还提供了进程间传递套接字的方法，这样使得服务主机负责调度响应的进程和数据服务进程相互分离。

XDL中的套接字读写可以是阻塞方式，也可以为异步方式，两种读写方式均支持基于事件（EVENT）或IO队列（QUEUE）的调度模式。

## Shell。

Shell也是一种操作系统资源，我们称之为系统管理壳。在shell中可以调用操作系统提供的命令，也可以完成一些批量操作。XDL实现了一些shell操作函数，为应用程序管理系统资源（文件、文件夹、快捷方式等）提供了便捷。

## 剪切板（Clipboard）。

剪切板也是应用之间传递数据的一种方式，XDL使用全局内存来设置、获取剪切板数据，支持多字节字符、双字节字符和位图数据格式。XDL中的文档体和子部件均支持基于剪切板的复制、剪切和粘贴操作。

### 1.1.6.图形资源（GDI）。

图形资源是操作系统提供以图形方式输出可视化内容的方法库。Linux的基本图形库在xlib中实现，windows提供了gdi和gdiplus图形库，对于构造桌面应用的人机交互界面，这些图形库的确够用了。但对于大型工程制图类应用（比如CAD），实时游戏等算法、渲染、实时性要求较高的应用场景，需要更高级的图形库支持，如第三方跨平台的opengl、微软的directX技术。

### 1.1.7.窗体资源（Window）。

Windows是原生的图形操作系统，窗体服务是一项系统服务，除了需要管理好众多窗体的秩序，窗体服务还将鼠标、键盘、窗体编排、窗面曝光等事件发送至当前焦点窗体，让用户程序响应这些事件。Linux天生不是图形操作系统，但它通过x-window提供窗体框架资源。x-window的运行框架是x server/client，Server是控制显示器和输入设备（键盘和鼠标）的软件，它负责创建窗体、绘制显示设备、将键盘鼠标的事件发送给client，client也是用户软件，负责和server建立通讯连接，向server发送绘制请求。Server和client可以跨网络部署，这是x-window特有的灵活机制，但也因此多了协议通讯层，在图形应用程序的执行效率和图形设备可直接操作性方面，x-window无法和windows、macos图形操作系统比肩。

## 1.2.第三方开源库（Open Source Lib）

### 1.2.1.Crypt Lib。

已收录与加解密相关的库包括：aes，des，arc4，rsa，md5，sha，dhm，asn，x509。AES是块加解密算法，区块长度固定为128比特，密钥长度则可以是128、192或256比特。DES也是块块加解密算法，区块长度为64比特，密钥长度为56位。3DES是DES更安全的算法改进，但DES已逐步被AES取代。ARC4是一种流加密算法，密钥长度可变。加解密使用相同的密钥，属于对称加密算法。RSA算法是一种非对称密码算法，需要一对密钥：公钥和私钥，公钥用于加密，私钥用于解密。RSA密钥至少为500位长，一般推荐使用1024位。RSA算法也是一个能同时用于加密和数字签名的算法，为数据安全传输和身份认证提供了便捷性。MD5是一种消息摘要算法，可以产生出一个128比特的哈希散列值，用于验证数据传输的完整一致性。MD5主要用于数字签名。SHA是一个密码散列函数家族，SHA-1可以输出160比特的哈希散列值，SHA-2可以输出224/256/384/512比特的哈希值。SHA和MD5一样主要用于数字签名，SHA在计算上比MD5费时，但在防碰撞性方面强于MD5。DHM（Diffie-Hellman算法）是一种解决密钥交换的方法，而不是加密方法，通常用于端对端传输对称加密体系的密钥。ASN.1即抽象语法标记，描述了一种对数据进行表示、编码、传输和解码的数据格式，在此用于x509证书数据格式的表达、存储和传输。x509一种通用的证书格式，它包含一些标准字段的集合，这些字段是有关用户或设备及其相应公钥的信息。

### 1.1.2.Hash Lib。

已收录的哈希算法库包括：crc32，murmurhash，siphash。MD5、SHA都属于加密哈希函数，此处收录的是非加密哈希函数。CRC的全称是循环冗余校验，CRC32它可以把一个字符串哈希成32比特的值，虽然CRC的碰撞概率高，但它最大的优势是可以采用硬件简单实现。MurMurHash3可以输出128比特的哈希值，算法效率高，碰撞率低。SipHash可以输出64比特的哈希值，主要特点是可以有效减缓hash flooding攻击。

### 1.2.3.Zip Lib。

已收录的压缩库包括：deflate，gzip，lzf，png，jpg。Deflae和gzip算法一致，是使用了LZ77算法和哈夫曼编码的一种无损数据压缩算法，gzip仅使用了deflate中的数据体部分，不处理压缩头部分。两者都常用于互联网协议中的数据压缩传输。LZF是采用lz77和lzss的混合编码的一种无损数据压缩算法，特点是速度快、能效高，常用于讲究节能的应用场合。png是沿用LZ77的一种无损图像压缩算法，jpg是有损图像压缩算法，两者都常用于位图压缩，jpg的压缩比更高，但是建立在牺牲图像质量的基础上的。

### 1.2.4.Code Lib。

已收录的制码库包括：code128，pdf417，qrcode。CODE128码是一种高密度一维条码，每个字符由3个条、3个空、11个单元构成，字符串可变长。PDF417是一种高密度、高信息含量二维条码，每一个字符由4个条和4个空共17个模块构成，最多可容纳1850个字符或1108个字节的二进制数据。QRCODE是一种矩阵二维码，它是在一个矩形空间通过黑、白像素在矩阵中的不同分布进行编码，可容纳数字数据7089个字符或字母数据4296个字符，而且具备全方位（360°）识读的特点。

### 1.2.5.Statistics Lib。

已收录统计库包括：hll。HLL（Hyperloglog）是用来做海量对象的基数统计算法。

## 1.3.基本数据结构（XDL Base Struct）

### 1.3.1.vector。

向量是一组线性数据，数据条目的个数即向量的维度。一个向量表征了相应维度空间内的一个坐标点。数值全为1的向量称为单位向量。向量的运算有：求模、相加、相减、数乘、数量积和向量积。

### 1.3.2.matrix。

矩阵是一组向量的集合，单位向量的集合称为单位矩阵。基本矩阵运算包括：加法、减法、数乘、转置、乘法。

### 1.3.3.set。

集合是一组元素构成的整体，元素可以是数据，也可以是集合。集合的运算包括：枚举、并集、交集、补集。

### 1.3.4.map。

位图是一个比特位的矩阵，每比特0或1值用于反映特定对象是否存在或状态真假与否。

### 1.3.5.stack。

栈是一种先进后出的数据结构，栈的操作包括：入栈、出栈。

### 1.3.6.list。

队列是一种先进先出的数据结构，队列的操作包括：入列、出列。

### 1.3.7.dict table。

字典表是一类键值查找表。字典编排按首字母顺序。字典的操作包括：插入键值、删除键值、查找键值。

### 1.3.8.hash table。

哈希表也是一类键值查找表。键值的编排按键的哈希值。

### 1.3.9.ac table。

AC自动机是一种多模匹配算法，它由一个trie树和一个匹配失败转移列表组成。Trie树负责构造单词查找表，匹配失败转移列表用于KMP匹配算法。AC自动机常用于用于统计和排序大量的字符串。

### 1.3.10.file table。

文件表用于管理文件磁盘空间，文件表将文件的磁盘空间分割为页空间，应用程序可以对文件表随机读写，文件表按页、按需对磁盘文件空间进行读写。

### 1.3.11.bina tree。

红黑树是一种非完全平衡的二叉树，但它也是一种查找性能较高的键值树。

### 1.3.12.bplus tree。

B+树是一种键值二叉树，非叶子节点为键索引集，叶子节点为值集合。B+树通常用于基于索引管理的数据库文件读写。

### 1.3.13.multi tree。

一种多叉树结构。

## 1.4.基本对象（XDL Base Object）

### 1.4.1.string。

字符串对象是一个动态缓冲区，用于可变长字符串管理。字符串对象也提供了编码转换（UTF-8/UTF-16LIT/UTF-16BIG/GB2312）功能。

### 1.4.2.variant。

变体对象支持各种基本数据类型：布尔值、布尔值数组、字节、字节数组、字符、字符数组、宽字符、宽字符数组、整型、整型数组、长整型、长整型数组、浮点数、浮点数数组、双精度数、双精度数数组，字符串、字符串数组。数组型变体，支持实体值（由变体管理分配和释放）和引用值（变体不负责分配和释放）。

### 1.4.3.object。

存储对象是更大粒度的变体。除了支持变体对象的序列化外，也通过对DOM对象的支持来实现各种XDL文档的序列化操作。序列化的操作需指定编码方式（UTF-8/UTF-16LIT/UTF-16BIG/GB2312）。

### 1.4.4.stream。

流对象是一种管理输入输出的对象。终端、串口、管道、文件、网络均支持流操作。流对象支持设定特定的编码方式（UTF-8/UTF-16LIT/UTF-16BIG/GB2312）下，数据写入和读出将会按编码方式进行转换。流对象的读写支持流模式（persist）和块模式（chunk）。流模式下数据读写是持续的过程，块模式下，数据实体中插入块分割标识进行中继传输。

### 1.4.5.canvas。

画布对象用于可视化绘制，使用内存位图实现。图形应用程序通过调用图形函数在画布上完成绘制后再提交至屏幕、打印机或SVG文件。画布采用0.1毫米制度量，提交至具体设备时再做像素转换，因此，画布满足所见即所得的输出特性。

## 1.5.网络协议栈（XDL Protocol Stack）

### 1.5.1.UDP。

UDP（User Datagram Protocol）即用户数据报协议，是OSI参考模型中一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务。XDL UDP提供了面向连接和非连接的开发框架，面向连接的udp框架主要用于点到点的消息或文件传输，非连接的udp框架主要用于消息广播（心跳感知）。

### 1.5.2.TFTP。

TFTP（Trivial File Transfer Protocol），由面向连接的UDP框架实现，TFTP总是由一个固定的端口（69）发起连接，然后由动态端口来控制文件块的传输顺序。

### 1.5.2.TCP。

TCP（Transmission Control Protocol）即数据传输和控制协议，是OSI参考模型中一种面向连接的、可靠的、基于字节流的传输层通信协议。XDL TCP提供了基于stream流对象的开发框架。

### 1.5.3.SSL。

SSL通过传输层对数据进行加密，为网络通信提供安全及数据完整性的安全协议。XDL SSL提供的开发框架符合SSL后继者TLS的规范。

### 1.5.4.HTTP。

HTTP（Hyper Text Transfer Protocol）协议是用于从WWW服务器传输超文本到本地浏览器的传输协议，http连接和会话通常采用tcp协议。HTTP协议采用了请求/响应模型。客户端向服务器发送一个请求，请求头包含请求的方法、URL、协议版本、以及包含请求修饰符、客户信息和内容的类似于MIME的消息结构。服务器以一个状态行作为响应，响应的内容包括消息协议的版本，成功或者错误编码加上包含服务器信息、实体元信息以及可能的实体内容。XDL HTTP提供了http协议的完整模型，同时支持stream流对象在http端到端的读写。

### 1.5.5.HTTPS。

HTTPS的连接会话采用SSL协议，请求/响应模型与HTTP相同。

### 1.5.6.RESTful。

REST（Representational State Transfer）提供了一个网络资源描述和访问的框架，它比目前主流的SOAP、XML-RPC更加简单明了。在服务器端，应用程序状态和功能可以分为各种资源。资源的例子有：应用程序对象、数据库记录、算法等等。每个资源都使用 URI得到一个唯一的地址。所有资源都共享统一的接口，以便在客户端和服务器之间传输状态。使用的是标准的 HTTP 方法，比如 GET、PUT、POST 和 DELETE、HEAD、OPTIONS。以REST风格实现的Web服务称为RESTfull，已逐渐成为SOAP技术的替代方案。

### 1.5.7.SOAP。

SOAP（Simple Object Access Protocol）即简单对象访问协议，是一种轻量的、简单的、基于XML的交换数据协议，它被设计成在WEB上交换结构化的和固化的信息。SOAP和WDL、UDDI一起构成WebService三要素，其中SOAP用来传递结构化数据， WSDL用来描述如何访问具体的接口，UDDI用来管理、分发、查询WEB服务。

### 1.5.8.DICOM。

DICOM（Digital Imaging and Communications in Medicine）即医学数字成像和通信，涵盖了医学数字图像的采集、归档、通信、显示及查询等信息交换的协议。

### 1.5.9.HL7。

HL7（Health Level 7）即卫生信息交换标准，主要用于规范HIS/RIS/LIS系统及其设备之间的通信，它涉及到病房和病人信息管理、化验系统、药房系统、放射系统、收费系统等各个方面。HL7的宗旨是开发和研制医院数据信息传输协议和标准，规范临床医学和管理信息格式，降低医院信息系统互连的成本，提高医院信息系统之间数据信息共享的程度。

## 1.6.文档和视图（XDL Document/View）

### 1.6.1.DOM文档

DOM文档一种是树形数据结构，是XML结构化文档的内存构造形式。每个节点由一个属性包和一个变长文本组成。DOM提供了节点遍历访问的通用API。DOM文档是构造任何XML格式文档的基础，且具有等同性。比如表单文档（Form），它等同于DOM文档，对表单字段（Field）的访问等同于对DOM节点的访问。

### 1.6.2.XML文档

XML文档由一个XML头和一个DOM文档体构成，XML头规约了DOM文档序列化的编码格式（UTF-8/UTF-16LIT/UTF-16BIG/GB2312），因此，XML文档是DOM的序列化形式。任何基于DOM文档构建的业务文档，都通过XML文档来传输和存储。

### 1.6.3.JSON文档

JSON（JavaScript Object Notation）是一种轻量级的数据交换格式，主要用于在JSON-PRC的远程调用过程中传输数据。在XDL中，JSON结构使用DOM文档来构造。

### 1.6.4.SVG文档

SVG是一种用XML定义的语言，用来描述二维矢量及矢量/栅格图形。SVG提供了3种类型的图形对象：矢量图形（vectorgraphicshape）、图象(image)、文本(text)。SVG文档主要用于矢量图的传输和存储，其内容可被各种浏览器解释生成可视化输出。

### 1.6.5.SOAP/WSDL文档

SOAP和WSDL文档都是DOM文档，在WEB应用框架中，SOAP用于装载、传输业务数据，WSDL用于描述和定位SOAP-RPC接口。

### 1.6.6.SCHEMA文档

模式（Schema）文档，是一种DOM文档，是设计者文档（表单、网格、图形等）与业务数据文档（XML结构化文档）的交互桥梁。模式文档提供了业务数据的范式定义，比如数据类型、长度、精度、数据命名、数据组织的序列、层次，即“模式”。通过模式，设计者文档向数据服务方交付业务数据，也可以从数据服务方获取业务数据并注入到设计者文档。

### 1.6.7.META文档

META是一种DOM文档，主要用于存储设计者文档（表单、网格、图形等），除了设计目标文档主体外，它包含一组元数据用以描述设计目的、作者等信息。

### 1.6.8.Form文档／视图

表单（Form）文档定义了一个字段（Field）集合，每个字段都是独立的输入输出域。除了像文本、标签、图像等常规的字段外，表单还支持表单、网格、图形等设计者文档作为嵌入子文档对象。表单视图是按照表单字段集合的编排、样式将表单可视内容输出到画布（Canvas）的操作接口。

### 1.6.9.Grid文档／视图

网格（Grid）文档定义了一个列（Col）集合和一个行（Row）集合。由行与列指定的单元格是独立的输入输出域。网格视图是按照行与列的编排、样式将网格可视内容输出到画布（Canvas）的操作接口。

### 1.6.10.Graph文档／视图

图形（Graph）文档定义了一个Y轴集合、一个X轴集合和一个G组集合。Y轴定义的是值阶梯或等级，标示了值的度量维度。X轴定义了范围或时间分割，是值的时效维度。G组不是轴线，它定义了一个度量方式（相当于一种数据类）集合，Y轴线可以从G组中选择合适的度量方式，以便在图形视图中输出可视化数据。图形视图是按照X,Y,G集合的编排、样式以及组合关系将文档的可视内容输出到画布（Canvas）的操作接口。图形视图目前支持点线图、柱状图、饼图的输出。

### 1.6.11.Proper文档／视图

属性（Proper）文档定义了一个节（Section）集合，每个节定义了一个实体（entity）集合，每个实体由键（key）和值（value）组成。属性文档主要用于维护一组可分类的属性集合。属性视图是按节顺序和节内的实体顺序输出的2级树形到画布（Canvas）的操作接口。

### 1.6.12.Topog文档／视图

地形（Topog）文档定义了一个地形矩阵和一个地形标识集合。地形视图是将地形矩阵和地形标识两个图层输出到画布（Canvas）的操作接口。

### 1.6.13.Images文档／视图

图像（Image）文档定义了一组图像集合。图像文档用于应用程序对各种图像资源进行统一管理，便于各程序模块共享图像，缩短图像的加载时间。图像视图的操作接口可以为画布输出图像预览。

### 1.6.14.Label文档／视图

标签（Label）文档定义了一个条目集合。每个条目（Item）都带有一组属性列表。标签文档主要用于组织、展示某项事物的摘要。

### 1.6.15.List文档／视图

列表（List）文档定义了一个层级条目集合，每个条目（Item）拥有子条目集合。列表文档主要用于分层管理事物对象。

### 1.6.16.Tree文档／视图

树型（Tree）文档定义了一个多级层次条目集合，每个条目（Item）拥有子条目列表。树型文档主要用于分级管理事物对象。

### 1.6.17.Tool文档／视图

工具（Tool）文档定义了一个按钮（Button）集合，每个按钮可以定义为一项操作。工具文档常用于构造框架窗体。

### 1.6.18.Ttile文档／视图

标题（Title）文档定义了一个主题集合，每个主题都可以关联到某项外部事物，比如使用标题来管理工作区窗体栈。工具文档常用于构造框架窗体。

### 1.6. 19.Status文档／视图

状态（Status）文档定义了一个静态栏目集合以及一个动态栏目。状态文档常用于构造框架窗体的状态栏，每个栏目用于显示应用程序的状态信息。

### 1.6.20.Memo文档／视图

文本（Memo）文档定义了一个行文本集合。

### 1.6.21.Tag文档／视图

标识（Tag）文档定义了一个多行文本和一个标识集合，标识嵌入于文本中，并可被文本替换。标识文档常用模版文档制作。

### 1.6.22.Rich文档／视图

富（Rich）文档定义了一个文本锚点集合，每个文本锚点都是自由输入域。

### 1.6.23.Anno文档／视图

注释（Annotation）文档定义了一个注释条目集合，注释文档通常绑定于一个图像文档，通过独立的图层为图像文档提供注释。

### 1.6.24.Menu文档／视图

菜单（Menu）定义了一个条目集合，每个条目对应为一个操作。菜单文档通常用户管理运行时弹出菜单。

### 1.6.25.Dialog文档／视图

对话框（Dialog）文档定义了一个部件集合，对话框用于设计期间设计、存储对话框样式，运行时对话框及部件被窗体化。

## 1.7.服务开发框架（XDL For Service Framework）

### 1.7.1.PNP框架

PNP框架提供了非连接的客户／主机服务，每个节点即是客户也是主机。

### 1.7.2.TCP框架

TCP框架提供了面向连接的客户／主机服务，节点需明确界定是属于客户还是主机。

### 1.7.3.HTTP框架

HTTP框架提供了HTTP客户／主机服务的完整实现。

### 1.7.4.OSS框架

OSS框架是云存储对象的一个RESTful风格的资源管理实现。

### 1.7.5.XDB框架

XDB框架是数据库资源访问的RESTful实现。

### 1.7.6.PACS框架

PACS框架是一个DICOM协议下的TCP和RESTful的资源访问实现。

### 1.7.7.HL7框架

HL7框架是一个卫生健康信息通讯框架的实现。

## 1.8.桌面开发框架（XDL Desktop Framework）

### 1.8.1.Box小部件

Box小部件提供了一些用于对话框的操作部件，如日期框、滑动栏、页导航等。

### 1.8.2.Editor编辑件

Editor编辑件用于表单、网格、属性控件输入域的数据操作部件，如文本框、列表框、日期选择框、弹出网格等。

### 1.8.3.Control控件

控件是文档与视图的窗体容器，用于编辑文档。如表单控件、网格控件、图形控件等。

### 1.8.4.Menu控件

弹出菜单控件。

### 1.8.5.Dialog控件

对话框是弹出和浮动的窗体，对话框可以模态运行，也可以非模态运行。

### 1.8.6.Frame框架窗体

框架窗体是控件的容器，框架窗体支持拆分、入坞等操作，用于构造应用程序运行界面。