# *The X Developing*
# 开发者指南

张文权

powersuite@hotmail.com

中国浙江杭州

https://github.com/jdzwq

2018 年 9 月

# 目 录

# 导言

众所周知，项目开发任务有简单也有复杂。简单的项目需求比较单一，选用一些 RAD 工具可以快速实现。复杂的项目的需求中，一般都会包含一些跨越客户机／服务器，跨越硬件层／应用层以及跨越多种平台的应用集成。当出现这些复杂的需求时，往往会对我们在选择开发工具的时候造成困惑。因为每种开发工具擅长的领域有所不同，比如 JAVA 比较适合服务应用开发，C#.NET 比较适合开发桌面应用，Python 则更擅长于处理数理统计，而对于 C，大家都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。

作为现实的情况，精通各种语言的万能程序员通常少之又少，当然，从个人的技能和专长而言，通识各种语言并没有必要。因为项目的开发并不会对语言的选择提出苛刻的要求，而是看重选用何种语言开发库对周期和成本控制所照成的影响。但从现今的情况看，这种选择性也是有限的。因此，一个成体系的公司，就可能会出现 JAVA 团队，.NET 团队，或甚 C 团队。好比 1 个人的事情需要交给 3 个人去做，每个人难免要做些重复的事情，而且每个人做的情况良莠不齐，由此造成了客观上的协作障碍，降低了项目总体质量，进而摊高了项目总体成本。

如果 JAVA 开发库对任何软件项目能一揽子做到底，那倒是十全十美的选择。既然没有这种可能性，那么 X Delevoping 的一些思路，也许能在软件项目的开发周期和成本控制、源代码质量、跨层设计、跨平台设计方面或许能提供帮助。以下我们从开发库的独立性、安全性、普适性、可移植性、可裁剪性和动态加载几个方面来阐述 XDL 的总体特性。

**独立性**.XDL 以 C 语言构建，但 XDL 本身并不使用 C 语言标准库，只是使用了 C 语言的某些机制，如跳转机制、断言机制。XDL 也没有像 C 标准库那样把部分系统的资源访问揽入库中，而是对系统资源进行统一界定，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 等系统资源的访问规约为平台依赖实现。XDL 只为这些资源的访问定义接口，具体的实现交由各平台的操作系统 SDK 来实现。

**安全性**.对于开发库的安全特性，涉及到三个方面：函数安全、内存安全和线程安全。通常我们知道 C 标准库中的一些函数，比如字符串函数不是操作安全的，其不安全性表现为不可控的越界读写。XDL 的函数设计总体上规避了越界读写的缺陷。以往 C 语言编写的程序，内存管控是件头疼的事，特别是服务端应用，出现内存泄漏而难以查纠，往往成为软件质量的致命伤。因此很多面向对象的语言都内置了失效对象回收机制，但总体上是针对进程空间的全局内存堆管理，垃圾回收很难兼顾时机和性能，也会增加额外的系统负担。XDL 认为内存的高频、巨额使用通常发生在工作者线程内，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，目的是保证线程的运行不会产生内存泄漏。那么如何实现线程的安全就是个关键的问题，通常，我们不能假设线程（特别是工作者线程）是不会崩溃的，相反，崩溃应该是线程的常态。XDL 充分使用了断言和跳转机制来实现了线程崩溃保护，以保证在线程崩溃时获得释放内存私有堆的机会。

**普适性**.林林总总的开发库，擅长做桌面应用却不能胜任开发服务应用，能胜任服务应用开发却不适合做底层软件设施。当项目接手，本来一个团队可以讲好的故事就非得三个团队来讲了。X Developing 主张一个团队来讲故事，在纵向方面，自下而上在硬件驱动、协议栈设计、服务例程、桌面应用各个层面，XDL 都提供了相应的开发支持。在横向方面，XDL 通过分层设计，对不同的操作系统环境提供跨平台支持，并且保持移植成本最低。

**可移植性**.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实现的，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。那 XDL 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。

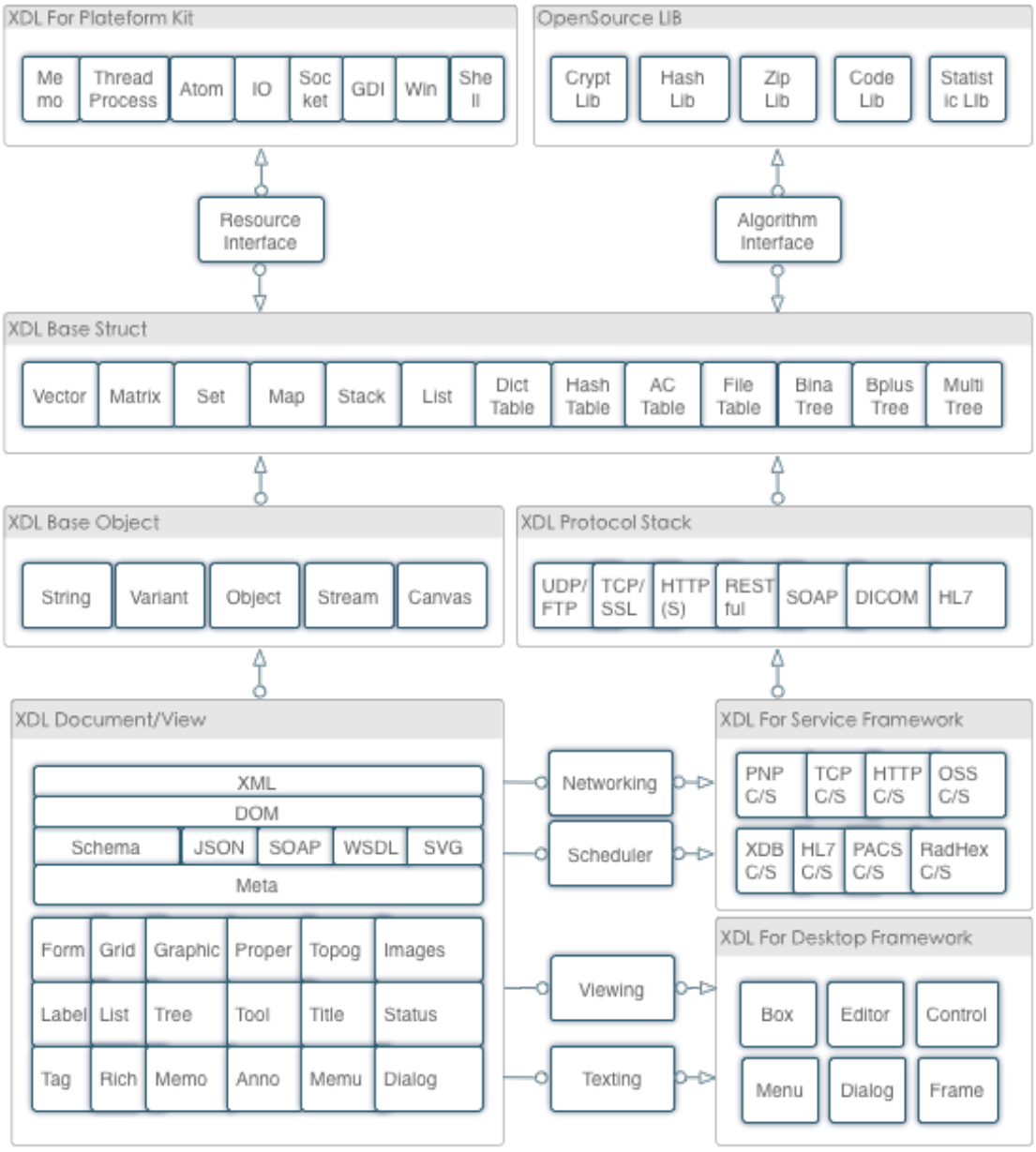
**可裁剪性**.操作系统的微型化是个重要的趋势，应用微型化就会存在刚性需求。如果一个号称功能强大的开发库开发的应用，无法适应微型化操作系统环境移植，我们才会意识到问题的严重性。系统的微型化意味着可用的系统对象资源越来越少，XDL 正是基于此来设计开发库的可裁剪性，设计之初就明确了裁剪路线。

**动态加载**.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。XDL 将前后兼容性作为重要的特性之一，其做法是定义图形库的一致性接口，程序运行时再根据系统版本环境来确定加载合适的图形库。

综上所述，XDL 总是希望为项目开发提供全栈支持，但仅有平台 SDK 的挂载，自身函数库的实现是远远不够的，所以，XDL 总是会遴选一些合适的数学、协议栈、算法、图像、语音、视频等开源库加入，以加大对应用开发的支持。鉴于此，XDL 亦是开源项目，项目工程位于：
https://github.com/jdzwq/Easily-SDK-5.git。

# 1.框架概览

X-开发者框架由平台依赖库（XDK），第三方开源库（GPL-LIB），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）五个层次组成。




## 1.1.平台依赖库（XDL For Plateform Kit）

平台依赖库由各操作系统提供的 C/C++SDK 库实现，目的是定义一致的 API 接口，实现对操作系统资源、原语对象、系统服务的读写访问。这些一致定义的资

源函数集合按分类的接口组聚合，通过运行时动态加载向 XDL 库提供 API 服务。

### 1.1.1 内存管理。

XDK 内存管理函数集提供了全局内存（Global）、进程堆（Heap）页内存（Page）、块内存（Block）、虚拟内存（Virtual Memory）以及内存映射（MapView）的内存空间管理。全局内存一般用于进程之间可共享的内存空间管理，常用于剪切板类内存使用。进程或线程在运行时，内存的动态使用一般在进程堆内申请和释放。在频繁使用大内存的场景中，比如数据库缓存、存储管理，会采用页内存和块内存来管理数据集的读写。虚拟内存为应用提供了可大于物理内存容量的连续线性空间读写机制，在 32 位系统中，地址空间可达 3GB，64 位的系统理论上可以访问 256TB 地址空间。虚拟内存空间的读写，需遵循申请保留（Reserve）、申请交付（Commit）、提交驻留（Protect）、解除驻留、交付释放和保留释放的正逆向流程。内存映射（MapView）将磁盘文件映射至线性虚存空间，为随机读写磁盘文件块提供了较好的性能。

### 1.1.2.线程和子进程。

线程是进程内一个可以被独立调度和分派的基本执行单元（指令集、寄存器、内存栈），它没有独立的堆内存，但可以与其他线程共享全部父进程资源。在 linux 中使用 fork 创建的子进程是父进程的副本，几乎继承了父进程的全部属性（包括各种打开的资源描述符）。然而在 windows 中并无严格意义的子进程，新的进程在父进程中被创建之后即为独立的运行实体，与父进程进程之间共享资源描述符需通过额外的机制（DuplicateHandle）。因此，在并发任务处理中，linux 系统采用多子进程服务是常用的机制，尽管其 posix 子系统也定义了线程（pthread）管理。而在 windows 系统中，则会被推荐使用子线程（thread）来应对并发任务。相比较而言，子线程的调度效率一定高于子进程，但需要额外的同步机制来协调对父进程资源的读写，可控性不如子进程。因此，对于应用服务而言，多线程服务的健壮性不如多进程服务。

### 1.1.3.原语对象。

临界区（critical section）、互斥量（mutex）、事件（event）、信号灯（semaphonre），都是系统提供的原子操作，用于控制资源对象的访问安全。临界区和互斥量都是资源的排他性访问锁，区别是控制粒度和使用的场合不同，临界区用于线程之间对执行某段代码的进入点进行排他性控制，而互斥量可以命名创建，可以跨进程使用，通常用于对多个工作目的相同的线程进行排他性运行控制，比如一次只允许一个线程对同一个文件进行写操作。事件也是线程粒度级的排他性锁，同样可

以命名创建，支持跨进程使用，与互斥量不同的是，通常用于对多个工作目的不同，但有序贯性关系的线程进行次序控制，比如先让一个线程先写入完成，再允许另一个线程去读取数据。信号灯提供了对可共享、但有限额的资源进行竞争性访问的控制机制，它是资源额度计数器，当前可用额度，决定了多少个线程能够同时访问该资源。

### 1.1.4.输入输出（IO）对象。

控制台（Console）、管道（Pipe）、串口（COM）、文件（File）都属于系统提供的输入输出资源。在非图形操作系统中，控制台是字符方式的人机界面。在图形操作系统中，控制台通常用于服务监控台，输出服务事件消息或程序追踪信息。管道是一种特殊的 IO 操作，数据在一端写入，在另一端读出，可用于本机进程之间的数据读写，也可以跨越网络，以 Client/Server 的方式读写数据。串口采用文件的方式打开，监听串口事件并收发数据，通常用于和符合 RS-232/RS-485 通讯规程的仪器进行数据交换。文件是磁盘数据的组织形式，读写本机文件可以采用相对路径或绝对路径，读写网路文件需遵循 UNC 路径命名。当以创建方式打开文件时，文件被截空，以只读或追加方式打开文件时，原文件内容被保留。文件支持顺序读写和随机读写，使用内存映射文件（MapView）可以优化文件随机读写的性能。

### 1.1.5.套接字（socket）。

套接字是主机访问以太网络的接口，分为三种套接字类型，流套接字（socket-stream），数据报套接字（socket-dgram）和原始套接字（socket-raw）。流套接字采用 TCP 协议通讯，TCP 是面向连接，实施传输控制的 IP 报文协议，为端到端提供可靠的数据传输。数据报套接字采用 UDP 协议通讯，UDP 为无连接协议，传输过程不保证数据传输质量。原始套接字对较低层次的协议进行访问（如 IP、ICMP）,通常用于网络通讯诊断或网络设备控制。安全套接字（SSL）是位于 TCP 协议之上，采用公钥 / 私钥、CA 证书体系，对数据传输进行非对称加密 / 解密及报文签名的通讯机制。

### 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。

### 1.1.8.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。Shell 提供了资源管理器对象，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。

## 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。