*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](#_Toc23871878)

[框架概览 7](#_Toc23871879)

[数据类型 9](#_Toc23871880)

[沿用C语言的类型 9](#_Toc23871881)

[系统资源标识符类型 9](#_Toc23871882)

[XDL的简单类型 9](#_Toc23871883)

[XDL的基本结构 10](#_Toc23871884)

[XDL的基本对象 10](#_Toc23871885)

[平台依赖库（XDL For Plateform Kit） 11](#_Toc23871886)

[内存管理。 11](#_Toc23871887)

[线程和子进程。 11](#_Toc23871888)

[原语对象。 11](#_Toc23871889)

[输入输出（IO）对象。 12](#_Toc23871890)

[套接字（socket）。 13](#_Toc23871891)

[Shell。 13](#_Toc23871892)

[剪切板（Clipboard）。 13](#_Toc23871893)

[设备场景（Context）。 13](#_Toc23871894)

[图形资源（GDI）。 14](#_Toc23871895)

[窗体资源（Window）。 14](#_Toc23871896)

[第三方开源库（Open Source Lib） 16](#_Toc23871897)

[加密解密库（Crypt Lib）。 16](#_Toc23871898)

[哈希散列库（Hash Lib）。 16](#_Toc23871899)

[压缩库（Zip Lib）。 16](#_Toc23871900)

[制码库（Code Lib）。 17](#_Toc23871901)

[数据访问接口（XDL For Database Kit） 18](#_Toc23871902)

[方式 18](#_Toc23871903)

[连接。 18](#_Toc23871904)

[检索。 18](#_Toc23871905)

[更新。 18](#_Toc23871906)

[大对象的更新。 18](#_Toc23871907)

[存储过程。 18](#_Toc23871908)

[基本数据结构（XDL Base Struct） 19](#_Toc23871909)

[向量（vector\_t）。 19](#_Toc23871910)

[矩阵（matrix\_t）。 19](#_Toc23871911)

[集合（set\_t）。 19](#_Toc23871912)

[位映射（map\_t）。 19](#_Toc23871913)

[栈（stack）。 19](#_Toc23871914)

[列表（list）。 19](#_Toc23871915)

[字符表（string table）。 19](#_Toc23871916)

[哈希表（hash table）。 20](#_Toc23871917)

[字典表（dict table）。 20](#_Toc23871918)

[AC自动机（ac table）。 20](#_Toc23871919)

[文件表（file table）。 20](#_Toc23871920)

[红黑树（bina tree）。 20](#_Toc23871921)

[B+树（bplus tree）。 20](#_Toc23871922)

[多叉树（multi tree）。 20](#_Toc23871923)

[基本对象（XDL Base Object） 21](#_Toc23871924)

[字符串对象（string\_t）。 21](#_Toc23871925)

[变体对象（variant\_t）。 21](#_Toc23871926)

[存储对象（object\_t）。 21](#_Toc23871927)

[流对象（stream\_t）。 21](#_Toc23871928)

[文件对象（file\_t）。 21](#_Toc23871929)

[画布对象（canvas\_t）。 22](#_Toc23871930)

[网络协议栈（XDL Protocol Stack） 23](#_Toc23871931)

[UDP。 23](#_Toc23871932)

[PNP。 23](#_Toc23871933)

[TFTP。 23](#_Toc23871934)

[TCP。 23](#_Toc23871935)

[SSL。 23](#_Toc23871936)

[HTTP(s)。 23](#_Toc23871937)

[SOAP。 24](#_Toc23871938)

[RESTFul。 24](#_Toc23871939)

[文档和视图（XDL Document/View） 25](#_Toc23871940)

[DOM文档 25](#_Toc23871941)

[XML文档 25](#_Toc23871942)

[JSON文档 25](#_Toc23871943)

[SVG文档 25](#_Toc23871944)

[SOAP/WSDL文档 25](#_Toc23871945)

[Schema文档 26](#_Toc23871946)

[Function文档 26](#_Toc23871947)

[Meta文档 26](#_Toc23871948)

[Form文档／视图 26](#_Toc23871949)

[Grid文档／视图 26](#_Toc23871950)

[Graph文档／视图 27](#_Toc23871951)

[Label文档／视图 27](#_Toc23871952)

[Topog文档／视图 27](#_Toc23871953)

[Memo文档／视图 27](#_Toc23871954)

[Tag文档／视图 27](#_Toc23871955)

[Rich文档／视图 28](#_Toc23871956)

[Anno文档／视图 28](#_Toc23871957)

[Images文档／视图 28](#_Toc23871958)

[Proper文档／视图 28](#_Toc23871959)

[List文档／视图 28](#_Toc23871960)

[Tree文档／视图 29](#_Toc23871961)

[Tool文档／视图 29](#_Toc23871962)

[Ttile文档／视图 29](#_Toc23871963)

[Status文档／视图 29](#_Toc23871964)

[Splitor文档／视图 29](#_Toc23871965)

[Menu文档／视图 29](#_Toc23871966)

[Dialog文档／视图 30](#_Toc23871967)

[文档工具包（docBag） 30](#_Toc23871968)

[图像工具包（imageBag） 30](#_Toc23871969)

[模式工具包（schemaBag） 30](#_Toc23871970)

[属性工具包（propertyBag） 30](#_Toc23871971)

[服务开发框架（XDL For Service Framework） 31](#_Toc23871972)

[UDP框架 31](#_Toc23871973)

[PNP框架 31](#_Toc23871974)

[TCP框架 31](#_Toc23871975)

[SSL框架 31](#_Toc23871976)

[SLOTS框架 31](#_Toc23871977)

[HTTP(s)框架 31](#_Toc23871978)

[桌面开发框架（XDL Desktop Framework） 32](#_Toc23871979)

[Box部件 32](#_Toc23871980)

[Editor编辑件 32](#_Toc23871981)

[Control控件 32](#_Toc23871982)

[Menu控件 33](#_Toc23871983)

[Dialog控件 33](#_Toc23871984)

[Frame框架窗体 33](#_Toc23871985)

# *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。

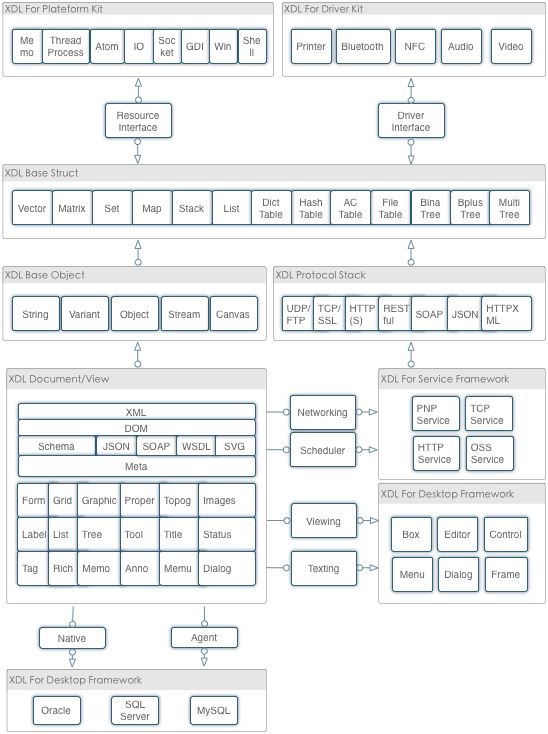
熟悉桌面应用的开发者都清楚，无论是Linux平台还是Windows平台，都具备多种图形库。这些图形库多数是为了保持系统应用向前兼容而保留下来。从客观上讲，我们开发的应用也应该保留向前兼容的特性，但如果选用了特定的图形库，向前兼容就未必行得通。X Developing定义了动态加载资源访问的设计框架，其做法是定义资源访问的一致性接口，然后程序运行时根据系统版本环境来加载合适的资源访问库。

X Developing体系包含了XDK、XDL、XDD、XDC、XDS、XDB开发库集合，以及xDesign，xService、xConsole应用工具集合。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。XDS是服务应用开发的支持库，支持建立基于UDP、TCP、HTTP(s)协议的多种端口服务。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。XDB是各平台一致的数据库读写工具库，支持Oracle、SQL Server、MySQL主流数据库。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），数据库访问框架（XDB），六个层次成。



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是端口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。

XDB是数据库访问支持库，它支持Native和Agent两种方式访问数据库。Native方式下，XDB使用ODBC、OCI、MYSQL的原生动态库直接访问数据库，数据连接在本地发起。Agent方式下，XDB使用本地存根（Stub）访问位于xServer上的XDB网关服务，数据连接在远程网关中发起。

# 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）**的使用首先需要创建一个定时队列，队列可以拥有一个或多个定时器，定时器可以设定间隔时间和重复周期，并由用户定义的定时回调函数（PF\_TIMERFUCN）来指派功能，每次被执行的工作函数均运行于独立线程。

## 输入输出（IO）对象。

控制台（Console）、管道（Pipe）、通信端口（COM）、共享内存（Share）、本地文件、网络文件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。

**本地文件（xuncf）**是磁盘数据的组织形式，XDL为本地文件的读写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。

**网络文件（xinet）**为http、tftp文件提供了读写机制，XDL的网络文件读写支持区块读写和压缩传输（deflate、gzip）。

## 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。

## 设备场景（Context）。

操作图形资源，必须先创建设备场景，XDL支持三种设备场景：内存缓存场景、显示设备场景、打印设备场景，XDL通常在内存缓存场景中完成绘制，然后递交至显示设备或打印设备场景，以上设备场景均采用点阵坐标。

## 图形资源（GDI）。

图形资源是操作系统提供文本输出和图形绘制方法库，随着操作系统的发展，图形库的功能也愈来愈强大，但每个系统版本阶段存在图形库不同的情况，XDL会根据运行环境来动态装载合适的图形库。

图形资源操作主要为文本输出、2D和3D图形绘制、图像输出的能力，现有的输出设备都为点阵设备，现实生活中我们所理解的尺寸均以米制坐标系为基础，XDL为点阵坐标系和米制坐标系提供了映射和转换机制，即“所见即所得”。

XDL图形库分别为设备场景（Context）和画布（Canvas）提供了绘制函数集，面向设备场景以象数点阵坐标操作函数集为主，而面向画布则实现了毫米制坐标操作函数集。

## 窗体资源（Window）。

窗体资源是图形操作系统的基础资源，通常以窗体的形式提供，与用户的交互方式一般为键盘、鼠标等输入设备，并通过窗体消息循环，来响应用户的操作。使用系统的窗体资源，必须理解以下一些概念。

**坐标系（Coordinate）**。图形系统有三种固有的坐标，即设备坐标、窗体坐标和客户坐标。设备坐标是用于描述设备的显示区域，并用于编排桌面和桌面中的各种应用程序的窗体。窗体坐标用于描述窗体的显示区域，并用于布局窗体标题、边框以及窗体操作按钮。客户坐标用于描述用户视图的显示区域，也用于编排属于该窗体的各种子窗体。除了支持以上三种坐标系，XDL还提供了画布坐标系，它采用毫米制坐标，并于窗体客户坐标系建立一一映射和转换关系。

**焦点（Focus）**。焦点是窗体与用户操作的连接点，用于响应用户的输入（通常是键盘等输入设备），窗体可以获得焦点，也可以失去焦点。由于XDL采用动态编辑件技术，响应用户输入必须激活编辑件并赋予其焦点。

**快捷键（Accelerator）**。快捷键是建立一个键盘按键与命令标识的映射表资源，并附加于目标窗体，在用户输入过程中，优先将按键转换为一个命令消息。

**菜单（Menu）**。菜单也是窗体的附加资源，不过XDL的菜单方式与传统菜单资源有所不同，它需要先创建一个菜单文档（MenuDoc），在附加于窗体后被编排为窗体菜单栏。

**滚动栏（ScrollBar）**。当用户视图的宽度或高度超过窗体客户区域时，需要滚动栏来支持将视图内容通过左右或上下滚动的方式掉整至窗体客户区域（可视区域）。由于XDL的视图可以以分页方式显示，所以在滚动栏增加了对页面滚动的支持。

**消息派发器（MessageDispatch）**。窗体采用消息循环机制来响应用户的操作，因此在创建XDL窗体时，需要安装消息派发器（if\_event\_t），它是一个窗体事件回调函数集。

**子类化（Subclass）**。XDL可以通过子类化结构（if\_subproc\_t）来向窗体安装子类化操作，子类操作也是消息回调函数，但它在消息循环中，总是优先被执行。

# 第三方开源库（Open Source Lib）

## 加密解密库（Crypt Lib）。

XDL中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。

## 哈希散列库（Hash Lib）。

XDL中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。

## 压缩库（Zip Lib）。

XDL中收录的压缩库包括：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。

## 制码库（Code Lib）。

XDL中收录的制码库包括：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。

# 数据访问接口（XDL For Database Kit）

数据访问接口是XDL访问数据库的函数集合。XDL目前支持Oracle、SQL Server、MySQL数据库的访问。

## 方式

XDL可以采用odbc、oci、stub三种方式访问访问数据库。

## 连接。

XDL连接数据源由入口参数指定或由DSN文件定义。

## 检索。

XDL的数据库检索操作有SELECT、FETCH、SCHEMA三种形式，SELECT操作由sql语句检索数据库后生成一个网格列集和行集。FETCH操作是由预先定义的网格列集生成sql检索语句，进而检索数据库生成网格行集。SCHEMA操作是由sql语句从数据库检索生成网格的列集的定义但不返回行集数据。

## 更新。

XDL的数据库更新操作为批量操作，支持将一个SQL更新语句包（含多个INSERT、UPDATE、DELETE语句），或者一个网格（含多个需更新的行集数据）一次提交至数据库，一次批量操作的事务是完整性。

## 大对象的更新。

XDL提供了大粒度的二进制对象、文本对象、xml文档的数据库存取方法。

## 存储过程。

XDL提供可变参数集合或Function文档来构造存储过程。

# 基本数据结构（XDL Base Struct）

## 向量（vector\_t）。

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。

## 矩阵（matrix\_t）。

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。

## 集合（set\_t）。

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。

## 位映射（map\_t）。

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。XDL支持用多个比特位来映射一个对象的多个状态。

## 栈（stack）。

栈是一种先进后出的数据结构，栈的操作包括：入栈、出栈。

## 列表（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。

## 字符表（string table）。

字符表是一类按键值的首字母顺序编排的字符串查找表。常用于构造选项列表。

## 哈希表（hash table）。

哈希表是一类按键值的哈希值编排的查找表。常用于构造属性列表。

## 字典表（dict table）。

字典表是一类由变体对象的哈希值为键值构成的查找表。常用于构造内存数据库。

## AC自动机（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。

## 文件表（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划分为页空间，应用程序可以对文件表随机读写，文件表按页对磁盘文件空间写入和读出。

## 红黑树（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。

## B+树（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。

## 多叉树（multi tree）。

一种多叉树结构。常用于构造表达式和计算式。

# 基本对象（XDL Base Object）

## 字符串对象（string\_t）。

字符串对象是一个动态字符数组缓冲区，用于可变长字符串管理。在windows平台下，在UNICODE的系统环境下，XDL字符串采用双字节UCS编码，在非UINICODE的系统环境下，则采用多字节（MBCS）编码。在Linux和macOS平台下，XDL字符串采用UTF8编码。鉴于此，字符串对象提供了编码（UTF-8/UTF-16LIT/UTF-16BIG/GB2312）相互转换功能。

## 变体对象（variant\_t）。

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）。变体对象也支持将其值根据编码类型编码为字节流。

## 存储对象（object\_t）。

存储对象维护一个内存区块，主要用于字符串对象、变体对象以及各种基于DOM的XDL文档正向序列化和反向恢复的操作。对象的序列化操作需指定编码类型（UTF-8/UTF-16LIT/UTF-16BIG/GB2312）。存储对象也支持压缩和解压操作。

## 流对象（stream\_t）。

XDL的流对象为管理输入输出提供了统一的操作机制。终端、通讯口、管道、共享内存、网络套接字、文件等均支持流操作。流对象的读写操作需指定编码方式（UTF-8/UTF-16LIT/UTF-16BIG/GB2312），并且支持必要的BOM头处理。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。

## 文件对象（file\_t）。

文件对象为本地文件、网络文件提供了一致的读写接口，读写何种文件资源，由文件路径名甄别。

## 画布对象（canvas\_t）。

画布基于设备场景创建，但采用毫米制坐标。XDL支持两种类型画布，图形图像输出画布和SVG文档输出画布。图形图像画布附着于缓存、显示设备和打印设备场景，并通过画布图形函数实现点阵输出。SVG画布附着于SVG文档，也是通过画布图形函数输出为SVG文档。

# 网络协议栈（XDL Protocol Stack）

## 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，UDP的特点是非面向连接、数据传输无序、无传输质量保证，但UDP也具有数据收发速度快、实时性能好的优点。

## PNP。

PNP（Peer Numerous Point Protocol）是XDL在UDP基础上实现的多点传输协议。PNP保留了UDP的优点，也为数据包的有序传输提供了保障机制。PNP可以用于多对一的数据传输，也可用于一对多的消息广播。

## TFTP。

TFTP（Trivial File Transfer Protocol），由UDP实现的文件传输协议。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。

## 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。

## SSL。

SSL（Secure Sockets Layer）是在TCP的基础上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为SSL提供了完整的实现。

## HTTP(s)。

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提供了HTTP协议的完整实现，同时支持stream流对象在HTTP端到端的读写。XDL的HTTP框架可以自由选择采用TCP或SSL作为基础的传输协议，选用SSL协议即支持HTTPS。

## 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WSDL、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。在XDL中可以轻松地构造SOAP和WSDL文档，并实现了SOAP服务框架，其中包括支持WSDL自动生成。

## 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。XDL服务框架中，大多数的Web服务都采用RESTFul风格实现。

# 文档和视图（XDL Document/View）

## 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。每个节点由节点名称、属性包、选项包和节点文本组成，节点名称支持命名空间和别名，属性包维护节点的属性集合，选项包为节点文本提供键与值的枚举。DOM提供了节点遍历访问的通用API，这些API是以下具体应用文档构造文档访问方法的基础，即创建、访问、销毁DOM文档的方法和以下具体应用文档的操作函数具备功能的等同性。

## 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-16/GB2312），XML文档是DOM文档的序列化形式。任何基于DOM文档构建的业务文档，都可以通过XML文档来传输和存储。

## 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的内存数据结构使用DOM文档来构造。

## 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。

## 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。

## 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。

## Function文档

函数（Function）文档，由一个函数体和一个参数集合构成，用于构造XML-RPC的远程过程调用。

## Meta文档

Meta文档主要用于存储可视、可设计的业务文档（表单、网格、图形等），包含文档头（HEAD）和文档体（BODY）两部分，文档头用于保存多组元数据用以描述设计目的、作者等信息等。文档体则保存具体的业务表单文档。

## 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。表单视图支持分页显示。

## 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。网格的列可定义为数据列或计算列，支持分组计算及行列合计。网格和列集可以绑定数据源用以生成SQL检索语句，经编辑后的网格文档可以根据行的状态生成插入、删除、更新的SQL语句。网格文档支持更新多个数据表。

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。网格视图支持多个子域分布展示，支持分页显示。

## 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组定义了一组标尺（度量方式）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轴，Y轴和G组集合的编排、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，支持分页显示。

## 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。

标签视图是按照条目集合的编排、样式以及序列关系将文档的可视内容输出到画布（Canvas）的操作接口。

## Topog文档／视图

地形（Topog）文档定义了一个地形矩阵（matrix）和一个地形点位（spot）集合。地形按照矩阵的值来描述地形概貌，并通过点位标识来描述特定位置的属性值。

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。

## Memo文档／视图

文本（Memo）文档定义了一个行文（line）集合。每个文本行都是自由输入域，行文支持缩进。

文本视图是根据文本文档的绘制属性，将行文格式为文本流输出到画布（Canvas）的操作接口。文本视图支持分页显示。

## Tag文档／视图

标识（Tag）文档定义了一个文本段集合和一个接合标识（joint）集合，接合标识嵌入于文本中，并可被文本替换。文本段和接合标识都是自由输入域。标识文档常用模版文档制作。

标识视图是根据标识文档的绘制属性，将文本段和接合标识格式为文本流输出到画布（Canvas）的操作接口。

## Rich文档／视图

富文本（Rich）文档定义了一个文本锚点（anchor）集合，锚点由标题和文本组成，每个文本锚点都是自由输入域。

富文本视图是根据文档的绘制属性，将锚点的标题和文本格式化为文本流输出到画布（Canvas）的操作接口。富文本视图支持分页显示

## Anno文档／视图

标注（Annotation）文档定义了一个标注物件（article）集合，标注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。

标注视图是将标注物件根据位置和绘制属性输出到画布（Canvas）的操作接口。

## Images文档／视图

图像列表（Images）文档定义了一组图像条目（item）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。

图像列表视图是将图像条目列表按照垂直或水平编排输出到画布（Canvas）的操作接口。

## 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。

属性视图是将节集合和实体集合编排为两级树形结构并输出到画布（Canvas）的操作接口。

## 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。

列表视图是将指定层级的条目集合进行编排输出到画布（Canvas）的操作接口。

## 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。

树型视图是将按层次展开的条目集合进行编排输出到画布（Canvas）的操作接口。

## Tool文档／视图

工具（Tool）文档定义了一个分组（group）集合，每个分组包含一个条目（item）集合，每个条目可以定义为一项操作。工具文档常用于构造框架窗体菜单栏目。

工具视图是将按分组的条目集合进行编排输出到画布（Canvas）的操作接口。

## Ttile文档／视图

标题（Title）文档定义了一个主题（item）集合，每个主题都可以绑定某项外部事物，标题文档常用来管理工作区窗体栈。

标题视图是将主题条目集合以及方位设定进行编排输出到画布（Canvas）的操作接口。

## 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。

状态视图是将栏目集合以及次序设定进行编排输出到画布（Canvas）的操作接口。

## Splitor文档／视图

拆分（Splitor）文档定义了一个树形条目集合，每个被拆分的条目含有两个分支，拆分操作可以水平也可以垂直。拆分文档用于框架窗体的多级拆分，每个拆分区作为子窗体的容器区域。

拆分视图是将拆分栏按水平或垂直编排输出到画布（Canvas）的操作接口。

## 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。

菜单视图是将菜单条目按水平或垂直编排输出到画布（Canvas）的操作接口。

## 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，在运行时对话框及部件被窗体化。

对话框视图是将各种类型的部件按其绘制样式输出到画布（Canvas）的操作接口。

## 文档工具包（docBag）

文档工具包定义了一组持久化函数集合，用于以上基于DOM的各种设计类型的文档与文件对象进行持久化操作。

## 图像工具包（imageBag）

图像工具包定义了一组持久化函数集合，用于位图的base编码以及与文件对象进行持久化操作。

## 模式工具包（schemaBag）

模式工具包定义了一组数据操作函数集合，用于将表单、网格、图形文档通过模式文档的定义从DOM文档提取结构数据或将结构数据注入DOM文档。

## 属性工具包（propertyBag）

属性工具用于将可设计文档（表单、网格、图形、地形等文档）的文档属性及其子部件的属性按分类写入属性文档或从属性文档中读出。在设计期间，这些属性集合可以在属性窗体中进行可视化编辑。

# 服务开发框架（XDL For Service Framework）

XDL服务框架在此指得是调度服务框架。所有部署于xService下的业务服务例程均接受服务框架的调度。调度服务运行于多个（可按CPU的核数配置）独立的线程，负责响应客户的连接请求，响应的模式支持套接字事件模式和IO异步队列模式。工作者模块是业务服务的执行体，它必须输出调度入口函数给调度服务，调度方式可配置为线程方式或进程方式，在两种方式下，工作者例程运行于独立的线程或进程空间。

## UDP框架

UDP协议调度服务框架。

## PNP框架

基于UDP协议实现的多点传输的调度服务框架。

## TCP框架

TCP协议调度服务框架。

## SSL框架

SSL协议调度服务框架。

## SLOTS框架

基于TCP或SSL协议实现的套接字调度服务框架。

## HTTP(s)框架

基于TCP或SSL协议实现的HTTP／HTTPS调度服务框架。

# 桌面开发框架（XDL Desktop Framework）

## Box部件

Box部件提供了一些用于构造对话框的操作部件，如数字输入框（numBox）、页导航（naviBox）、文本提示框（tipBox）、日期框（dateBox）、时间框（timeBox）、按钮框（pushBox）、旋动框（spinBox）、滑动框（slideBox）、单选框（radioBox）、检查框（checkBox）、静态框（staticBox）、形状框（shapeBox）、图标框（iconBox）、列表框（listBox）、下拉框（dropBox）、词汇框（wordsBox）、弹出菜单（menuBox）、小键盘（keyBox）、编辑框（editBox）等。Box部件在运行时被持久窗体化，作为对话框与用户进行交互的数据输入部件。

Box部件向父窗体发送Command消息来告知部件状态和数据的变化，主要的消息代码有：文本改变（COMMAND\_UPDATE）、对象改变（COMMAND\_CHANGE）。

## Editor编辑件

Editor编辑件用于表单、网格、图形、属性等文档输入域的数据操作部件，如文本编辑件（fireEdit）、日期编辑件（fireDate）、时间编辑件（fireTime）、检查编辑件（fireCheck）、数字编辑件（fireNum）、列表编辑件（fireList）、网格编辑件（fireGrid）、词汇编辑件（fireWords）等。编辑件可由各文档子部件的输入域属性指定，并在运行时按需动态创建及销毁。XDL的编辑件充分支持无键盘模式下的手势操作。

Editor编辑件向父窗体发送Command消息来告知编辑件状态和数据的变化，主要的消息代码有：文本改变（COMMAND\_UPDATE）、对象改变（COMMAND\_CHANGE）、提交编辑（COMMAND\_COMMIT）、撤销编辑（COMMAND\_ROLLBACK）。在编辑件失去输入焦点，或者发送了提交和撤销编辑的消息后，编辑件即被销毁。

## Control控件

Control控件是文档与视图的窗体容器，将文档附着于窗体，并通过视图输出可视化的结构形式和内容。在运行时，结构固定的文档采用输入域创建动态编辑件的方式与用户进行交互。结构自由的文档则采用全文本输入方式与用户交互。XDL的结构固定的文档控件主要有：矢量图控件（svgctrl）、表单控件（formctrl）、网格控件（gridctrl）、图形控件（graphctrl）、对话框控件（dialogctrl）、图表控件（chartctrl）、图像控件（imagesctrl）、地形控件（topogctrl）、位图控件（photoctrl）、字符表控件（tablectrl）、标签控件（labelctrl）、列表控件（listctrl）、属性控件（properctrl）、状态栏控件（statusctrl）、标题栏控件（titlectrl）、工具栏控件（toolctrl）、树形控件（treectrl）。结构自由的文档控件有：多文本控件（memoctrl）、富文本控件（richctrl）、标识文本控件（tagctrl）。

控件向父窗体发送NOTICE消息来告知控件操作通知。

## Menu控件

菜单是弹出和浮动的模态运行窗体。XDL可以根据菜单文档来创建弹出菜单。此外提供了一些常用的弹出菜单，比如：文本操作菜单（textorMenu）、字体名称菜单（fontnameMenu）、字号菜单（fontsizeMenu）、字形菜单（fontstyleMenu）、字重菜单（fontweightMenu）、颜色菜单（colorMenu）、形状菜单（shapeMenu）。

每个菜单条目的标识符（iid）将作为消息代码通过COMMAND消息发送给拥有者窗体。

## 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。XDL可以根据对话框设计文档来创建对话框。此外提供了一些常用的对话框，比如：消息对话框（msgDlg）、输入对话框（inputDlg）、属性对话框（properDlg）、字符表表对话框（tableDlg）、列表对话框（listDlg）、文本对话框（textDlg）、网格对话框（gridDlg）、图形标注对话框（annoDlg）、预览对话框（previewDlg）。

对话框按钮定义的操作码将在对话框模态运行结束后返回给应用程序。

## 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。