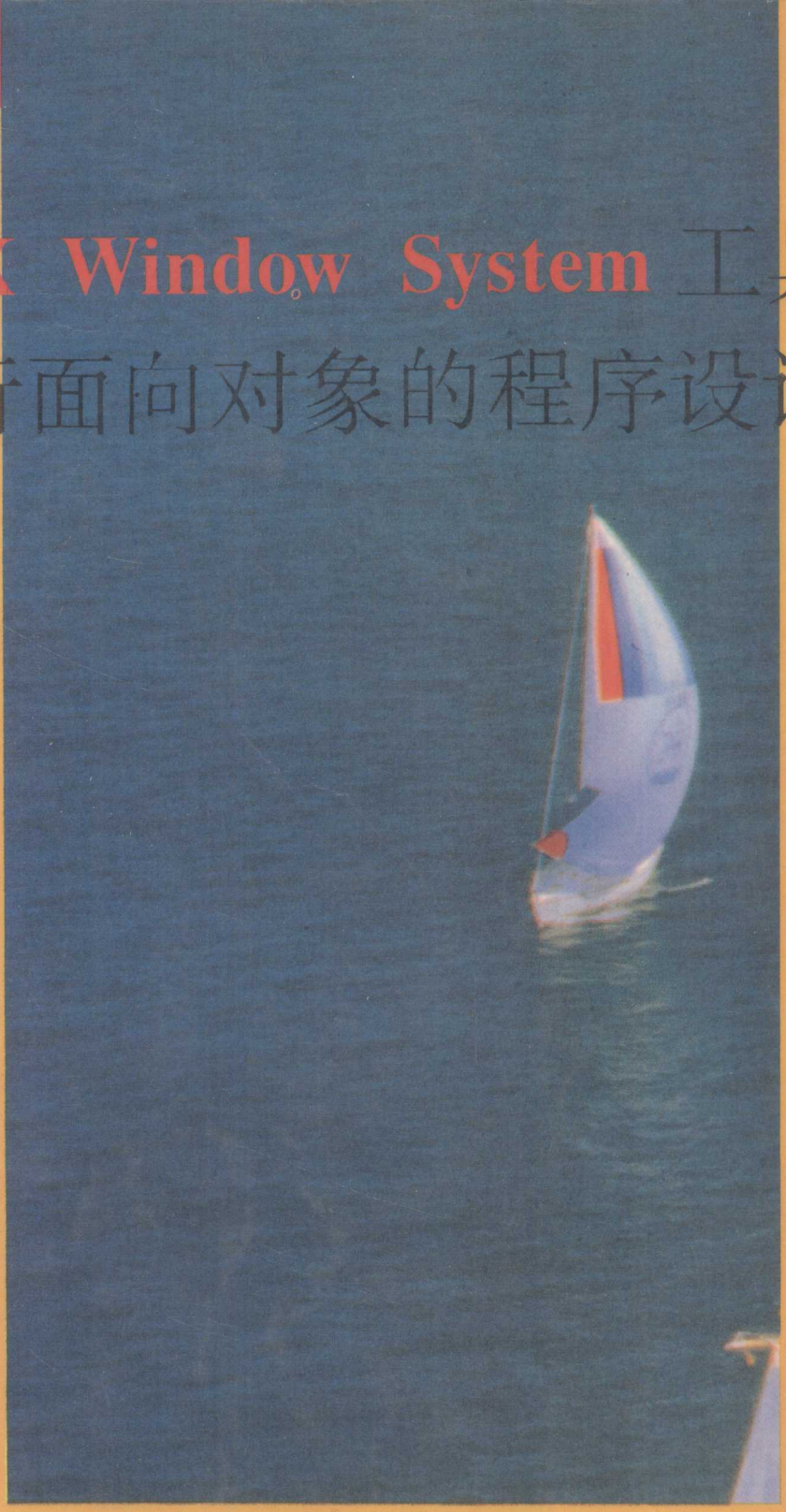
使用 X Window System 工具箱进行面向对象的程序设计



洁　夏　青　等编
望　程时言　审校

海洋出版社

前言

X Window System™ 显然是近年来最重要的软件开发工具之一。一旦当专有的窗口系统具有 PC 标准时，X™ 将推出能支持各种窗口程序、工具箱和接口软件的“开放式”窗口系统。虽然 X 不依赖于任何特定的软硬件系统，但目前，基于 UNIX 的工作站系列从 X 得益很多。将 UNIX 与 X 结合将提供一个功能强大的、支持分布式客户－服务器模式的多窗口、多任务环境。

X 除支持分布式客户－服务器计算之外，一个最重要的特性就是其对窗口操作的“类属”支持，即 X 提供一个丰富而有力的图形窗口系统却较少对支持 X 的高层应用加以约束，包括窗口管理器与工具箱。由于 X 提供完成图形用户接口(GUIs)的机制而不规定用户接口类型，因而 X 的(用户)集将非间接地、一定程度上取决于未来方向的基于 X 的计算环境。

每天，在各种各样的硬件平台上的 X Window System 用户、应用程序员、及软件开发者都要以选择窗口管理系统、应用软件及高层次工具箱的简单形式对未来 X 的的应用“投票”——X 用户是未来 X 的参与者。在很多情况下，一个公司的应用程序将在另一个公司的窗口管理下操作。X 不断动态的前进发展以至于使基于 X 的计算变得更加生动而有趣。

复杂的图形用户接口(GUIs)需要复杂的编程环境。X Window 系统就是在多软件层下完成的。在最低层次，X 协议为客户和服务器提供硬件独立性。特别是当双方都加载了 X 协议，则在一个硬件平台上运行的应用程序能与另一个硬件平台上的服务器通讯。

在下一个层次，X 提供了 Xlib，一个 X 协议的 C 接口。使用 Xlib，程序员可以通过 X 应用程序进行很多操作控制，但必须处理很多由高一级接口自动管理的琐事，特别是 X 工具箱。因而对于一个大的基于 Xlib 的应用程序，软件的主要部分，大部分项目开发时间将花在要求支持现代化的 GUI 目标(如下拉式菜单及弹出式对话框)的相对低层次的编程上。

这本书的假定之一是接口成份应该当作伪目标(在 C 中)执行。意思是说，程序员应当在他的应用程序中寻找公共模块并开发自己的模块来完成这些目标。例如，Xlib 程序员能够开发出高级窗口模块，可重用并且可以十分灵活支持地支持各种扩充程序。因此，一个高层次的 Xlib 窗口集应包括一个 Window 数据结构，二级数据结构以及(指向)完成一个窗口行为的函数。

对于不需要 Xlib 的低层次功能的项目设计，Xt 提供了一个高层次的替代，以确保核实应用程序的最低标准。在很多情况下，Xt 与一个 widget 集相连接，比 Xlib 性能更优，因为它为应用资源和高级接口目标提供了更加完善的支持。本书第二个假定是，无论对于 Xt 还是其它工具，将一个应用程序分解为可重用的模块都是很重要的。正确地分解应用程序对于应用程序的可维护性、可移植性、可读性是十分重要的。

对于某些应用设计，一个特定的 widget 集并不提供该应用所需要的所有 GUI 组成成分。在这种情况下，最好另外开发 widget 类以完成所需的功能，而不要把现存的 Widgets 都连接在一起。通常，我们为某一应用的某一特定需要所编写的 GUI 成分，在另一应用中也需要，因此开发一个能为多种应用服务的新 widget 类是非常有意义的。

本书主要分为三个部分。首先基中讨论了 X 工具的编程。为步入 Xt 应用程序和 widget 编程，第一部分描述了 Xlib 编程的各种特性。特别是它提高了基于 Xlib 的应用(应建立在高

层 Xlib 模块之上)的地位。如下面所述,书中将讨论支持(1)面向行列的文本窗口,(2)回调函数,(3)自包含事件循环等几个示例模块。因为它们是基于 GUI 的软件的最具有普遍性的代表,因此我们便可以使用各种格式对话框来说明这些观点。当然,这些原理也适用于 GUI 的其它组成成分。

本书在第二部分主要讨论 Xt 应用编程。第二部分逻辑上是接着第一部分的。即,第一部分说明如何使用 X,Xlib 的低层接口编写回调函数和自包含事件循环;然后在第二部分对同样的编程结构使用 Xt 的内部支持。第二部分首先回顾了基本的 Xt 应用编程,包括回调与和动作函数的使用。然后,说明如何开发执行阻塞和对话框的高级 Xt 模块。第二部分还包括一个相当大的应用程序以说明各种 Xt 编程工具。

本书在第三部分研究 Xt widget 编程。在开始探讨 widget 编程之前,第二部分中作为 Xt 的背景与回顾描述 Xt 应用编程很重要。在第三部分我们开发了六个相兼容的 widget 类并扩展了 Althena widget 集的功能。

本书之所以基于 Althena widget 集主要是因为它适用于标准的 X 版本。很多 widget 程序员开发现存 widget 集的扩展功能。这就是说 widget 程序设计与其它面向对象的程序设计相似,包括从现有的类中产生新的类别。因此,widget 程序员必须仔细阅读理解源码的主旨。由于标准 X 支持 Althena Widget,因此 ALthena widget 集是建立公共 widget 集的理想起点。

另外,在讨论面向对象的 Xt 编程序特性时,我们的主要目标是提供一些有用的应用程序和 widgets。在大部分讨论中,我们使用实际的工具说明 Xlib 和 Xt 应用编程的特点。除最初提到的对话框之外,第一部分使用了两个实用程序:xclk 和 xdump,分别概述了 Xlib 编程并描述了文本处理模块。xlck 是一个简单的数字时钟实用程序,能够以任何字体显示当前时间。xhump 是一个汇编格式,十六进制的转储实用程序,其功能就象 more(1),显示文件的每一页。

第二部分讨论六个用于删除和恢复的模块的 Xt 应用编程。wastebasket 实际上是一对组合工具,delete 与 xwaste。delete 支持命令行文件删除,将"被删除"文件移入废纸篓中而不是永久性删除。xwaste 是一个 GUI 废纸篓工具,它允许用户浏览查看废纸篓并且可恢复或永久删除文件。xwaste 象 delete 一样,也支持将文件删入废纸篓中,只是它使用文本项对话框并且命令按钮替代为命令行。

第三部分讨论了 widget 编程,包括两个示例和复合 widget 类。作为 widget 编程的一部分,第三部分描述了六个 Xt 经常使用的实用程序,这些 widget 包括 XiStrSelect,一个字符串选择 widget;XiAlert,一个阻塞报警 widget;XiButton,一个非命令行按钮 widget;XiChoice,一个多选项选择 widget;以及 XiSimpleText,一个单线文本项 widget,它隐藏了 Althena 文本 widget 版本 3 和版本 4 之间的不同。

本书的所有应用程序都在标准 X 的版本 3 和版本 4 上验证过。另外,它们都在版本 3 和版本 4 的 X 公共服务器及两个不同的 Sun Microsystem CPU 结构上测试过。

本书所描述的应用程序需要大量的源代码。通常,我们不在每一章描述一个应用程序时给出全部的源代码,,而是重点讨论每一个应用程序中最具特色的部分。完整的源代码将在附录中给出。

用这种方法,每一章不必给出全部的应用程序,但读者可在任何时候返回适当的附录作进一步的研究。

第一章 简 介

本章提出几个与使用 X Window System 工具包进行程序设计相关的比较普遍的问题。关于应用程序及 wdiget 程序设计所使用系统的方法的重要性是一个老话题了，这点在本章中简单介绍一下，将在后面章节中详细阐述。本书的一个前提就是 X Window System 是相当复杂而先进的，对于采用逐步深入的方法来开发软件将会有所收获，而对于采用非结构化方法使用 Xlib 及 X 工具，将会给予惩罚。

1.1 面向对象的程序设计及其工具

面向对象(OO)的程序设计(OOP)技术对专业程序设计界正产生着深远的冲击，这一点似乎是毫无疑问的。20 世纪 70 年代末期和 80 年代初期，许多人都认为面向对象程序设计与结构化程序设计展开了竞争。但在另一方面，许多程序设计人员，特别是系统程序员，仍旧喜欢用 C 语言，而不喜欢用 C++语言，而 C++语言是支持面向对象程序设计语法的语言。当然，无论你对此采取何种态度，但都不能否认模块分解、功能抽象、封装(encapsulation)等原理在开发大型应用程序中的重要作用。

X Window System 是用传统的程序设计技术完成的，而且它的标准程序设计接口 Xlib 图形库是基于一种传统的语言而不是面向对象的编程语言(OO)。尽管 X Window System 有传统语言的特性，但它的设计却是相当灵活的，它鼓励开发各种更高一级的接口，不论是基于传统的语言，或是面向对象的语言，如 LISP、C++以及 Pascal 都是支持的。从这点来看，X Window System 给计算机界提供了一个工业标准的、基于网络的窗口平台。随着 OO 与用户接口技术的成熟、工业的发展、新技术的出现，X Window System 的程序设计和用户接口也随着改变，以反映最新的工业发展趋势。

对于应用程序设计人员，目前的趋势是使用一个工具包作为 Xlib 的高级接口。虽然 X Window System 的工具包提供了相当强的程序设计功能，但是将它做为一个高级程序设计的接口功能，则它仍是一项不成熟的技术。用当前工具箱——Xlib，而不频繁地使用低级接口来开发一个实时/商业应用程序是很困难的。从这点来看，在 X Window System 的高级接口领域中有许多课题等待着研究和开发。当然，决定这些接口是否选用面向对象语言、传统的语言或者是伪 OO 语言的问题是超越于工业标准之上的问题。

在这本书里，我们把重点放在 X Window System 工具包提供的伪 OO 程序设计接口上，特别是标准的 Xt Intrinsics，包括 Athena Widget 集合。Xt 提供了一个基于 C 的程序设计接口，建立在基于 C 的对 X Window System 的 Xlib 接口的顶层上。因为以 C 为基础，所以 Xt 不提供对面向对象程序设计语言编译级别的语法支持。从另一方面讲，Xt 与一个 Widget 集合的连接就是鼓励面向对象风格的程序设计。

Widgets 是 X 工具包的核心，正如对象是面向对象的语言(如 C++)的核心一样。在本书中，我们集中讨论以下在使用一个现存的 widget 集合中的问题，如可移植性、可重用性等，也就是应用程序程序设计的问题。我们也提出了 widget 的开发(widget 集合和层次)，即 widget 程序设计过程。

1.2 X Window System

X Window System 是一个相当大的软件系统，它在工作站上支持交互式的运算。它给低级的基于网络的窗口操作提供了设备独立性支持。

除了这些联网的功能外，总体上讲，在应用程序与工作站的通讯期间，检查由 X Window System 管理的那些活动是很方便的。这些活动可以发生也可以不发生在一个(物理的)网络上。

X Window System 软件的各组成成分，可以被抽象地看作是建立在一个基础窗口系统之上的软件层(见图 1.1 所示)。使用这种分层方法，只有一个软件层(最内层)能与基础窗口系统直接接口。X Window System 带有的低级的程序设计接口可以看作是一个建立在最内层之上的软件层，以此类推。

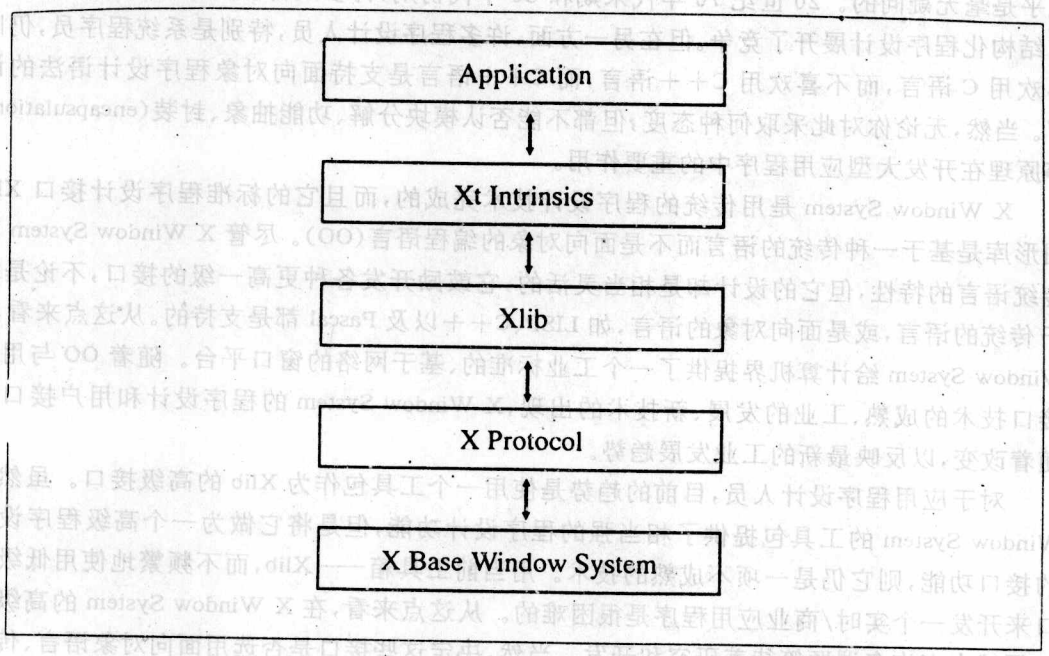


图 1.1 X Window System 软件层

我们提出这种从低级到高级的结构是有原因的。特别要指出的是，本书的目标就是要着重阐明对于 X Window System 用一种逐层深入的方法来进行软件开发的重要性。X Window System 所支持的这种软件的半独立性模块开发对 Xlib 程序员和工具箱程序员及应用程序程序员和被称为 widget 的程序设计人员来说都是很重要的。

虽然本书主要集中讨论 X Window System 工具箱的面向对象的程序设计技术(例如，Xt Intrisics 及 Athena widget 集合)，但是在介绍性的一些章节中仍对 Xlib 图形库作了阐述。关于 Xlib 的讨论是值得的，因为低级的 Xlib 程序设计对于解释一些建立在高一级工具箱中的功能是有帮助的(例如回调函数)。(在本书中，我们经常使用通常意义上的"函数(function)"

一词，包括返回空值的过程。）此外，许多用基本工具箱（例如 Xt）开发的应用程序非常依赖于 Xlib 的一些功能，即那些 Xt 不具有的功能或是用 Xt 处理不方便的功能。最后，在开发新的 widget 过程中，widget 程序设计人员必须经常使用 Xlib。

1.3 应用程序与工作站之间的通讯

在最底层，应用程序通过一个符合规定网络协议的消息与工作站进行通讯。典型的情况是，一个应用程序发出了消息，这些消息在工作站上产生了指令，而指令则要排队执行（例如，在一个指定窗口中进行画线操作）。由于 X Window System 的网络特性，则使应用程序能够在 y 工作站上运行，而输出显示在 z 工作站的一个窗口中。

一个应用程序还能指定它所感兴趣的工作站事件的类型/类，例如点按一下鼠标的按钮 1。当相关的事件发生时，工作站会通知应用程序，而应用程序通过发送附加消息给工作站来作出回答。从这点上看，X Window System 起着通讯组织者和协调者的作用：(1)应用程序向一个或多个工作站发送消息，(2)工作站对消息作出回答，并生成事件，(3)应用程序有选择地对某些事件作出回答，产生更多的消息，如此往复。

一个应用程序经常被称为一个客户，而一个工作站经常被指定为服务器——一个 X 服务器。这个术语是有道理的，因为工作站为应用程序的需要服务，即给客户程序提供一个装置来显示它们的工作。在许多情况下，服务器（服务器）一词更合适，因为 workstation（工作站）带有更多的硬件内涵。从技术上讲，把服务器与基础的窗口系统软件等同起来更恰当，它通过 X 协议消息存取，而不是使用工作站硬件。

1.4 用户的操作环境

从不同的角度来讲，X Window System 或简称为 X，是为多个应用程序同时在一个基本窗口的环境下运行而设计的，这个环境给传统的操作系统（例如 UNIX）提供了一个接口。更重要的是，这个操作环境本身正是另外一个 X 应用程序。X 的强大性能及普及性源于它所提供的及它所不提供的功能。我们可以说 X 工具实现的是机制，而不是策略。

换句话说，X 可以补充一个或多个应用程序运行于一个或多个工作站的基本窗口操作，而一个应用程序的观感（look and feel）是由外部因素决定的。窗口的管理程序——运行在 X 上的一个应用程序，在很大程度上决定着其它应用程序的观感。如果使用了任何一个工具箱来开发应用程序，应用程序的观感直接依赖于该开发工具箱。例如，一个应用程序中的按钮可能看起来并且操作起来与另外一个应用程序中的那些按钮不同，这与当前的窗口管理程序无关，而依赖于所选取的工具箱。

X 能容纳多种不同应用程序的能力是它的一个最强的性能。X Window System 为执行超越当前图形用户接口（GUI）的操作提供了机制。如果工作站产业决定采用一个各厂家通用的或者是核心的 GUI，它将着力于实现符合作为 X 应用程序的 GUI 设计的窗口管理程序。

1.5 功能的分解和抽象

在一些窗口系统中，提供基本的窗口操作（例如把一个鼠标事件排队）的软件组成部分

与那些产生高级事件的解释操作(例如当按下鼠标时弹出一个后台菜单)的软件组成部分之间存在着紧密的联系。这种紧密联系的方法与许多 PC 机的窗口系统是一致的。

正如我们前面所提到的,X 采取了一种不同的方法。X 及许多其它大的软件系统的基点就是,一个大的系统能够由多个不同的协作模块建立起来。虽然模块的分解和功能的抽象是两个不同的概念;但它们常常是联合使用的。

X 支持功能抽象的一个表现就是,它根据 X 网络协议和 Xlib 图形库层次把基本的窗口操作分层或是分解成因子。前者负责在一个应用程序及工作站之间进行低级通讯(例如,在一个窗口显示一行正文的消息或请求)。后者从函数库调用中产生相应的消息——例如,函数 XDrawImageString(),它符合 X 网络协议,因此可以执行通讯。

把一个大的软件系统分解成为半独立的模块的源动力中,包含了建立实用的好的软件工程的期望。对于 X,一个主要的目的是创建一个操作环境,在从 PC 机到工作站,从小型机到大型机各种硬件结构上实现。

将 X 移植到其它软件/硬件平台上,基本的任务是解释 X Window System 协议,也就是执行已经被编码到在客户与服务器间传送的信息之中的任务。典型地,一个为这种目的服务而设计的硬件/软件系统也会为编程人员提供与系统的高级接口。对于当前的 X 工具来说,这个接口是由一个称为 Xlib 的 C 语言的子程序库提供的。

再者,X Window System 的一个主要性能是程序设计员可以决定是直接用 X 协议工作还是保持一段距离,只用 Xlib 进行工作。更进一步讲就是,程序员可以自行决定是用 Xlib 还是用工具箱来开发应用程序。

1.6 X Window System 工具箱

X 的工具箱和 widget 集合的数量太大了,在这里我们无法一一列举。在本书中,我们主要集中于讨论 X 中公用的标准 intrinsics 及 widget 集合,称为 Xt Intrinsics 及 Athena Widget 集合。在后面的章节中,我们提供几个与 Athena Widget 集合兼容的公用的 widget,并且为由于某种原因没有使用商业 widget 的应用程序程序设计人员扩充 GUI 的功能。在本书中有时也会提到一个商业 widget 集合,OSF/Motif widget 集合(OSF 1990)。我们的示例程序都是基于标准的发布版本,并且面向对所有商业工具箱普遍发生的问题。

有一种观点认为,工具箱和 widget 集合的发展给 X 编程人员提供了更多的选择机会,并且刺激了人们对 X Window System 的认可。从另一方面讲,至少随着时间的推移,程序设计人员和开发人员必须准备好能使用多种工具箱工作。对于那些已经能在不同硬件结构上使用多种软件产品的程序设计人员来说,提供一个带有多用户接口的产品(Motif, Open Windows [Sun Microsystems 1990],等等),选择一个具有多用户接口的特殊的工具箱,这样做如果不是不可能的话,在经济上也会是相当困难的。因此,面向各种各样的编程问题,不局限于一个特定工具箱,是方便和重要的。

1.7 低层次与高层次应用程序开发方法的对比

一些程序设计人员优先考虑汇编语言而不是高级语言,因为他们需要(或了解)汇编语言所提供的更高的控制级别。编译器将一条高级语言语句翻译成几条机器指令。对于一个

给定的应用程序和任务，一个擅长汇编语言的程序设计人员经常能写出一套好的指令。

决定使用 Xlib 还是使用 Xt 的问题有点类似于使用编译器或汇编程序的问题。一旦程序设计人员决定使用一个工具箱，仍然有机会通过有选择地使用 Xlib 函数库来提高效率。对于一个给定的应用程序系统，一个好的 Xlib 程序设计人员可以选择一个 Xlib 操作的最佳组合，它们可以比为执行同样的任务而设计的工具箱函数的效率还要高。

不论怎样，通常真正的问题并不是一套指令在一个指定应用程序中的效率问题，而是这些指令在大量应用程序中的可重用性问题。本书假设，在大多数情况下，已编译好的模块(compiled module)的可重用性，比通过人工使用由一个象工具箱支持的 Xlib 库函数来编制各种窗口的操作而获得的效率方面小的改进要重要得多。(关于 Compiled module，我们指的是汇集在一个文件中的一族 C 语言的函数和公共数据结构。通常，C 语言的程序设计人员就可趁此利用 C 语言的文件级的名称域优点来封装数据及相关的操作。引用 Smith 1990 对这个名词的扩充说明。)如果在一个基于 Xt 的应用程序中使用 Xlib 库函数不能兼顾应用程序的可重用性或者任何一个组成应用程序的已编译模块的可重用性的两方面要求，那么，至少，对于可重用性问题，没有理由不把 Xlib 与一个工具箱连在一起使用。

当然，在一个基于工具箱设计的不用考虑效率问题的应用程序中使用 Xlib 还有一些其它原因。例如，设计一个让航空工程师观察由一定参量的变化而导致飞行器机翼在外形上的变化的应用程序系统。在这里，工具箱可能提供建立起一个应用程序的基于鼠标器的界面所需要的所有 GUI 组件，例如，按钮、量器及菜单。另一方面，一个普遍意义上讲的(接口)工具箱，例如 Xt，不提供飞行器设计软件所需要的绘制图形的低级功能——那是 Xlib 图形库的领地。

Xlib 的第三个也是通常的用途是用于建立新的 widget。(widget 是用一个完成了高级 GUI 组件的公共功能接口封装起来的数据结构的集合。例：菜单)扩充一个工具箱的工具，在这里就是它的 widget 集合，在一类应用程序中引入附加的一致性是很有用的。

例如，考虑一下，一个软件开发人员生产了几个 X 产品，比如说用于 UNIX 的图形工作站的一套软件包。假设，为了应用程序间的一致性，该开发人员在所有的应用程序中，使用了标准的 Xt 工具箱及商用的 widget 集合 xyz。另外，假设每个应用程序需要一个公共的 GUI 组项，该组项在 widget 集合 xyz 中没有，比方说，排它性选择的选择柜。那么，该开发人员有两种选择：(1)让开发小组尽可能地从其它的 GUI 组项来不规则地补充这个 GUI 组项。或者(2)先建立一个新 widget，以实现该 GUI 组项，然后使得每个应用程序的开发人员均可使用该 widget。

大多数的程序设计人员和开发人员会喜欢第二种方法。前一种方法，补充 GUI 选项作为一个 widget 会导致应用程序系统对每个应用程序在用户接口和源码之间需要更大的一致性。用程序设计方法学的观点，widget 加强了可重用性、功能与数据的抽象等等。第二种方法的潜在不利之处是 widget 程序设计方法比应用程序程序设计方法从规则上讲级别要低。在某些情况下，通过工具箱来安装 widget 比安装应用程序代码要困难得多。在本书的后面我们将提出通过 widget 程序设计方法创建高级对象，而不采用在应用程序程序设计中使用的编译模块的方法。

目前，我们已经讨论了具有基于 Xlib 图形库操作的工具箱应用程序的长处和合理性。我们所提出的 Xlib 与工具箱对比的最后一个问题是是否要完全使用工具箱。假设你打算开发一个小的程序设计人员的编辑器，或者作为一个标准的独立的产品，或者作为一个可在不

同应用程序中使用的模块，当然，你决定使用 Xlib 图形库或使用一个工具箱之间要依赖于很多问题，例如你以前和未来的应用是否以工具箱为基础(界面的一致性)，以及你所希望的伴随着工具箱的用法而带来的限制、独立性以及优点方面的问题。

工具箱的一个优点就是它所提供程序设计的能力。使用一个工具箱，相对很少量的源代码就可以创建一个全部具有下拉式菜单、按钮、以及弹出式窗口的强有力的界面。无论怎样，每个基于工具箱的应用程序被自动地赋予在一个应用程序中可能是很有用的功能，但在另外一个应用程序中可能就成为累赘的功能。例如，有这样的应用程序，该应用程序不能简单地利用(不需要)一个工具箱的全部功能来处理 X 资源。

把一个从工具箱创建起来的简单编辑器的效率与一个只用 Xlib 创建的定制的编辑器的效率相比，前者是可以超过后者的。后者的效率是前者的一半。在许多情况下，共享库的使用可以减小单用 Xlib 创建的应用程序与用工具箱创建的应用程序可执行代码尺寸间的不一致性。再者，还必须要考虑应用程序间的一致性问题，现存模块的可重用性等等的问题。

从给定的这本书的题目来看，很显然我们倾向于支持基于工具箱的程序设计的益处。工业上的一致意见是，对于许多应用程序，即使不是大多数应用程序，一个工具箱可以把应用程序程序设计人员与许多 X 相关的问题隔离开来，这些问题把我们所关切的正常应用程序开发过程的问题牵得太远了。

虽然我们正规地用工具箱开发应用程序，但是我们也要为我们偏爱的纯 Xlib 程序的低级特性开发应用程序。对于那些完全喜爱不掺杂 Xlib 的程序设计的人员来说，考虑下列方案。

早些时期，具体说就是 1900，一个从事家庭建筑行业的木匠，他对从草图建造房子而得意，可能有蔑视使用预制窗户的思想。今天，在大多数房屋中预制窗户是很正常的。在今天的家庭建筑环境下工作，同是一个木匠，在建造设计得最认真的房屋(或大楼)时，多半会选择使用一个最好的预制窗户，在需要的地方用上一个或多个定制的窗户。

X 工具箱提供了高质量的窗口：编辑正文窗口，滚动窗口的内容，从菜单选择等。产生一定应用程序的特殊事件，提供了许多完成一个特殊工具箱的预制的 GUI 组件的 Xlib 及 widget 程序设计的良机。从某种意义上讲，对一个 X 程序员的衡量标准(象衡量一个木匠一样)，是以他(她)的生产一个预制的及定制的组件的精美组合件的能力为标准。

第二章　面向对象的程序设计和 X Window System

本章中我们讨论几个面向对象程序设计的功效问题，然后集中讨论对封装(eacapsulation)、函数抽象、X 工具箱可用的功能等等的高一级支持。为了转移到我们集中讨论 Xt 的面向对象(OO)程序设计能力的阶段，在 C++与 Xt 之间作对比是很方便的。对于 C++的研究是较次要的。本章假定读者没有关于 C++的预备知识。在后继章节中，大部分内容我们把我们的讨论限制在用 X 及 Xt 进行面向对象程序设计的问题方面。

2.1　面向对象程序设计原理：

2.1.1　封闭(Encapsulation)

传统的编程语言允许程序设计者把有关的数据装进一个语法单元中。例如：使用一个 C 语言中的结构(struct)或者是使用一个 Pascal 语言中的记录(record)。这些程序单元对于描述不同类别的程序实体是很方便的。例如：描述一个窗口。习惯上，程序设计人员最终把这种程序实体想象成一组程序语句的形式，由一个混合的数据结构来代表——定义对象的数据结构，在我们的例子中就是一个窗口。而根据可能发生的操作来描述对象是次要的。传统编程语言反映了程序设计的这种观点。

面向对象语言，例如：C++ (Stroustrup，1986；Lippman，1989)以及 Smalltalk (Goldberg and Robson，1983)提供了形式化的机制，把一个或多个数据结构与操作于那些结构上的过程包装起来。这种封装(encapsulcation)鼓励程序设计人员根据数据及其可能发生的过程来考虑程序实体。例如典型地，一个窗口是一个对象，它占据部分显示屏幕、用边框来规定范围的，可以由一个位置移动到另一个位置的，可以放在一个同胞窗口上面或下面的，可被做成图标。

假设你走近一个热情的 X Window System 程序设计人员或用户，声称对现代的 GUI 环境毫不了解，并且问这样的的问题："什么是一个窗口？为什么在我的计算机屏幕上需要一个窗口？"X 的热衷者很可能对窗口作这样的解释，它是一个矩形的区域，信息在这里输入或显示，然后开始讨论你对窗口可做的所有事情：移动、改变大小、隐含、覆盖它等等。也就是说定义窗口的动态特征不会少于静态特征。

根据动态和静态属性来观察一个实体是面向对象程序设计的实质。甚至在面向对象语言成熟并流行以前，许多程序设计人员就认识到了封装对软件可重用性的重要作用。尤其 C 语言允许程序设计人员使用一种面向对象风格的程序设计方法。更特殊的，程序设计人员能使用 C 语言对文件级名称域的支持，把数据和相关的函数收集在一个包中。使用 static 修改程序，程序设计人员能使一个变量对在一个文件中(已编译模块)的每个函数都是公有的变量。但在下述情况中是专用的：

```
...
static int height, width;
...
int get_width() {...}
void set_width(int w) {...}
...
```

Smith (1990)广泛地讨论了这些技术。

使用这种方法来开发完成正文编辑窗口、对话框等功能的模块是一个直接途径。一旦这些件被开发好，就可在不同的应用程序中使用了。例如：一个对话框模块可以被编译并且存在一个库中。从某种意义上来讲，这种程序设计方法促进或加强了象窗口这类事物的面向对象的观点。

无论怎样，使用一个传统的语言，把一个面向对象的观点强加到数据上(及程序设计人员身上)是没有益处的。使用C语言的文件级名字域建立的伪对象(Pseudo－object)的一个最明显的问题，那就是不便于处理一种同时具有实例数据及类数据的对象。

类数据(类 data)就是在一个对话框的多个事件中相同的数据，能在已编译的模块中，象上面所描述的那样由静态变量来处理。实例数据(Instance date)就是对话框对象每次出现的专用数据，不能简单的用这种机制来处理。例如：如果对话框的高度和宽度是以简单的静态变量存在，那么就无法处理多个不同大小的对话框了。

当然，对于一个对象的多个实例的情况，可以先建立混含数据结构的模板(typedef)，然后设计操作于对话框上的相关函数。该对话框作为指针传递给每个函数。

```
...
typedef struct {
    int height, width;
    ...
} response_box;
...
int get_width(response_box *resp_box) {...}
...
```

使用这种技术，使用对话框模板的程序设计人员必须在应用程序中为每个这样的窗口分配一个对话框结构和一个ID(指针)。

```
...
response_box resp_box1, *resp_box_ptr1 = &resp_box1;
...
...
window_activate(resp_box_ptr1, ...);
...
```

在这种情况下，用多个窗口进行程序设计变成了一个摆弄窗口指针的练习。从定义来看，在一个应用程序中指针的分配和控制组成了一种低级的程序设计形式。

相反，一种象C++这样的语言提供了一个形式化的语法机制，把一个对象和它的操作封装在一起。因此，对于我们的例子，为创建多个对话框，应用程序程序设计人员可以简单的分配每个窗口，发送适当的信息：

```
...
response_box resp_box1;
...
resp_box1.activate(...);
...
```

当面向对象语言刚开始得到承认时，许多传统程序设计人员都在回避面向对象的方法，例如：辨解说 C 语言程序设计人员可以象 C++程序设计人员一样做面向对象程序设计，不用再学一种新语言。前面提供的两个同样的代码段是用来证明传统的方法与面向对象方法几乎是同样简洁的。但是简洁并不是最重要的考虑。第二个代码段说明抽象性的增加，在大的复杂的应用程序中是很有意义的。

2.1.2 继承性(inheritance)

封装(Encapsulation)是一个可利用的最有力的程序设计技术；它在开发可扩展的及可重用的软件模块方面是至关重要的。正象我们已经说明的，封装可由传统的语言或面向对象语言来发展。忽视了面向对象及伪面向对象程序设计技术重要性的传统编程人员没有认识到用面向对象语言的第二个主要的程序设计特征，叫作继承性(inheritance)。

对象是很少孤立存在的。正象传统程序设计人员需要混合的数据结构把相关的变量组合成一组一样，面向对象程序设计人员则必须能从简单的对象和从简单的变量中建立起复杂的对象。对于传统的编程语言，合成(composition)是建立高级数据结构唯一可用的机制。从我们前面所举的例子来看，一个对话框是一些变量(包括 height 及 width)的集合或混合体。

C 语言的 struct 和 typedef 结构对于建造简单的混合结构是足够的。但是，考虑一个应用程序或一系列应用程序，就需要一个对混合数据结构更加复杂的协调。一个窗口系统就是一个例子。在这种情况下，一个基本窗口的结构应该定义对所有窗口都通用的变量，例如：高度和宽度。在下一个较高的层次里，另一个混合结构能够支持显示和正文入口一类的窗口提供变量。到更高一层，使用另外一个混合结构来定义一种对话窗口，如此往复。

对于 C 语言，使用简单的合成来处理这种复杂的安排当然是可能的。但无论怎样，C 语言语法上的限制使得这种方法很不方便："Window. text _ window. response _ box. <whatever >")。面向对象语言除了对混合结构的传统支持外，又提供对类(类，一种数据结构)继承的全部语法支持。因此，面向对象程序设计者能更容易地创建相互联系的，复杂的数据结构，例如：窗口的层次。对类继承规范化的语法支持鼓励程序设计人员在设计数据结构之前先做好计划，以便于其它的应用程序能重用现存的模块。

对类继承的语法支持也提供了其它的益处。例如：C++为初始化(复杂)的对象提供了一个机制，同时分配它们的存储器，以便于初始化的数据通过实例层次向下遗传。用这个观点看我们前面的例子，width 和 height 的值就被传播到基本窗口对象。

```
// width == 75, height == 25
response_box resp_box1(75, 25, <other initializers>...);
```

2.1.3 动态联编(Dynamic Binding)

面向对象的第三个主要特征就是动态联编。动态联编与封装性(encapsulaction)和继承性(inheritance)一起起作用。还是使用我们的窗口例子，考虑一个包括只读浏览器、对话框、正文编辑窗口等等窗口层次的例子。每个窗口(对象)都带有它自己的操作功能。对于一个浏览器，这些操作包括在窗口中显示正文和将一个文件分页的操作。对于一个正文编辑窗口，这些操作包括正文浏览操作及正文修改操作。

用动态联编，一个程序可以在运行期间决定一个窗口(对象)的类别，并允许适当的操

作。在运行期间联编是很重要的，例如：把不同类别的对象传送给高一级的函数，如一个命令调度器，一个 C 的程序设计人员可以在数据结构中，在 case 结构中使用标签，达到用程序来区别不同的对象。无论怎样这种方法都给软件的可重用性带来了几个否定的结果。更多的 C++和 Smalltalk 所支持的动态联编形式支持/鼓励实际上不能用象 C 这样的语言来实现的程序设计方法。这些话题的所有讨论超出了这本书的范围，参看 Lippman(1989)或者 Smith(1990)对封装、继承性及动态联编的进一步的论述。

2.2 使用 Xlib 图形库进行面向对象的程序设计

C++和 Smalltalk 所支持的面向对象程序设计类型不可能适用于 Xlib。这个评论并不意味着对 x 或者 Xlib 的否定，Xlib 图形库是 X Window System 的低级程序设计接口。我们在前面讨论了 X 的高一级的程序设计接口，随着 X 相关技术的成熟和发展，这种程序设计接口能够并且也将被提供。

X 是一个十分先进的窗口系统。前面我们提出了这样一个观点，X 程序设计人员应该为不同 X 应用程序建立连续的较高级模块的层次(layers)，作为加强原码的可读性及可重用性的方法。例如，Xlib 为创建窗口提供了一个叫做 XCreateSimpleWindow()的函数。这是一个高级的函数，它的参数包括窗口的宽度、高度、父亲等等。因此，由 XCreateSimpleWindow()创建的窗口的一些特性是在窗口创建期间就被设定了。

更重要的，窗口的其它属性不能由这个函数来设定，但是必须由下面这些函数如：XChangeWindowAttributes()、XSetNormalHints()、XSetStandardProperties()以及其它函数。此外，一些窗口属性根据时间而变化，对于 X，每个窗口有固定数量的与其相关的窗口属性；属性值可由函数 XGetWindowAttributes()来查到。对于许多应用程序，补充一些属性是必要的，例如，以象素制尺寸及行/列尺寸来保持窗口的大小；或者为了方便，建立起你自己设定的每个窗口间的联系以及图形上下文、字体、光标或其它的任何属性。

对某一个应用程序可能很方便的，或者需要其它应用程序来确保其可重用性的数据结构之间的第二级联系，是 Xlib 采用一个分层的方法进行应用程序开发的动力。特别地，即使使用与 X 的低级接口，也很可能建立高级的"对象"，例如，支持一个已经固定的操作集合的对话框。这些模块可以被编译、存储在一个库中。

当我们简单地了解了 X，并且介绍了后面章节中的低级 X 程序来讨论不同的类似于 Xt 的能力后，我们将给出一个完整的例子，基于我们的窗口及本章前面有关对话框的讨论。例如，我们将讨论怎样创建带有支持回调函数按钮的高一级的对话框。

2.3 用 Xt 进行面向对象的程序设计

一个 X Window System 工具箱由工具箱 intrinsics 和一个伴随的 Widget 集合组成。本书中我们的重点在于标准的 X 工具箱 Xt，特别是 Xt Intrinsics 及 AtlhenaWidget 集合。我们将偶然地与商业的 OSF/motif Widget 集合的一些特征作比较及讨论。

按照 X 软件分层的观点，Xlib 是 X 网络协议上的最低一级，Xt Intrinsics 建立在 Xlib 的上面，见图 2.1。本质上 Athena Widget 集合在 Xt Intrinsics 之上形成了一个软件层。为了方便起见，本书中除非另加说明"Xt"是指 Xt Intrinsics 和 Athena Widget 集合。当一个应用程序

建立在 Xt Intrinsics 和 Athena Widget 集合之上时，组成了第四个软件层。但是，我们曾经提到过：许多基于 Xt 的应用程序也必须直接涉及到 Xlib，因此，这些层不必要分开。

我们曾经提到过使用软件层来加强 X 应用程序的可重用性的重要性。因此，在后面的章节中，我们将提出一个程序设计人员建立第五层软件的不同的步骤。某些情况下，在工具箱和应用程序之间建立一个软件层有助于把修改工具箱所产生的影响隔离开来，且可以简化从一个工具箱转到另一个工具箱的处理过程。

在前面，我们把一个对象描述为一个数据集合，并且带有施加于那种数据上操作。C++对对象语法上的支持加强了这种观念。那就是，对于一个给定的对象，(公有的)变量的组成部分(称为成员变量)及过程组成部分(称为成员函数)都提供了同样的语法。例如，考虑一个假设的询问用户帐号的弹出式对话框：

```
...
resp_box1.prompt = "Please enter your 75-digit account number";
...
resp_box1.activate();
result = resp_box1.get_response();
...
```

在大多数情况下，禁止外界对一个如例中 Prompt 变量的访问，而使用一个函数例如，函数 activate()来代替设置用户的提示符是比较合适的，这里的观点就是，访问一个对象的成员变量的语法和成员函数的语法是相同的。

因为 Xt 依赖于 C 来完成它的对象或者 Widget，因此对象必须使用函数来创建、销毁及操纵——应用程序程序设计人员不应该直接修改 Widget 实例变量。自此以后，我们可以把 Object 和 Widget 互换使用。因为根据定义，一个 Widget 是一个(window)对象的描述。

对于不同的 Widget 类的共同操作，例如：Widget 的创建、删除，Xt Intrinsics 提供了必要的(公共的)函数。但是对一个特殊 Widget 类所特有的那些操作，例如：向询问一个选择框用户的回答情况，Widget 程序设计人员必须提供适当的函数才行。

在大多数方面，C 语言中对对象的语法支持的不是对于使用 Xt 的应用程序程序设计人员并不是一个问题。特别地，软件工程的原理要求，通过函数调用的方法来操纵对象，而不是直接修改对象的实例变量。Xt Intrinsics 的一个主要的函数就是为发生在不同类别的对象(Widget 类)的共同任务提供了一个一致的接口。因此，应用程序程序设计人员可以从 Athena 集合或从任何其它 Widget 类中使用 XtCreateManagedWidget()函数来创建任何一个 Widge 的实件。

2.4 标准的协议与 Xt Intrinsics

我们已经提出过这种思想，就是给在不同的 Widget 类上执行的相似操作的函数起一个共同的函数名字 Xt Intrinsics 支持这个一致性。从本质上讲，象在大多数面向对象语言中提到的一样，这就是标准协议的原理。在现实世界的应用程序中，过高估计软件可读性的标准协议的重要性是不可能的。

无论你是使用 C 给不同类型窗口创建可编译模块，或是使用 C++来创建一个窗口类，为在不同模块/类上执行的相似操作的函数起一个共同的名字是很重要的。例如：假设一

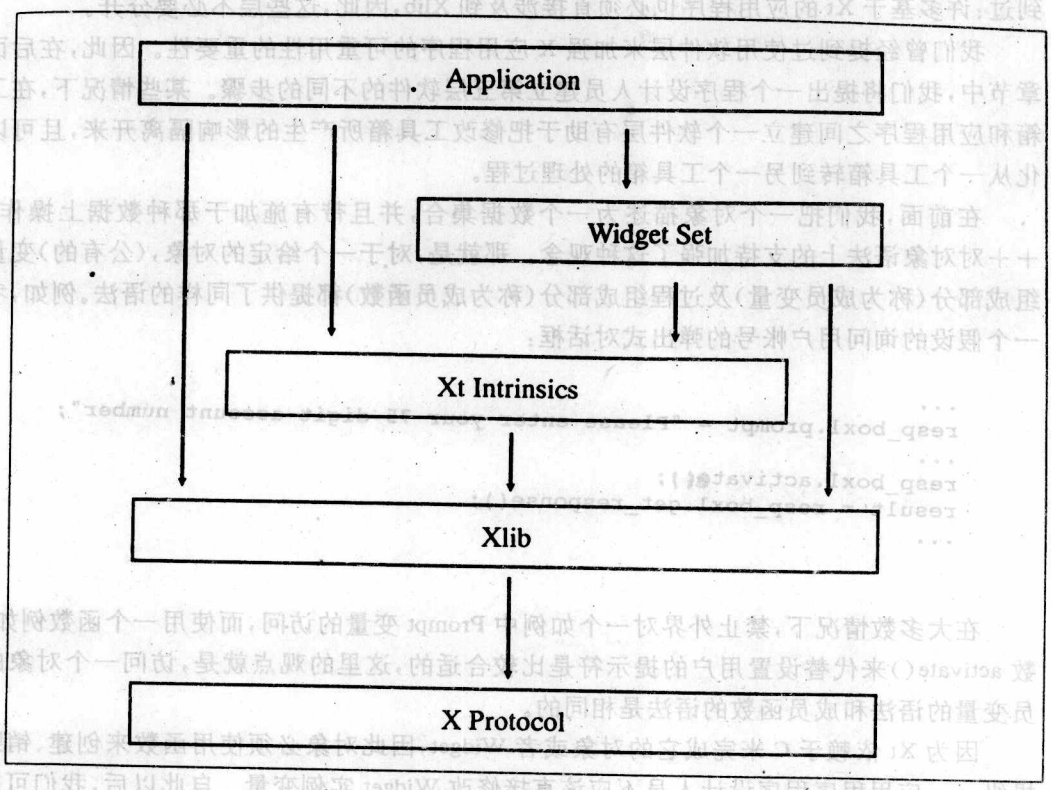


图 2.1 X Window 软件层

个对话框被称为"response—activate (resp—box1)"的函数激活，并且一个报警框被称为"pop-up—alert(alert—box1)"的函数激活，并且这种不一致性贯穿于你的整个软件系统。显然，未来的程序员，包括最初的软件开发者也很难记住这些函数的名字。

Xt 用两种方法促进标准协议的使用。第一通过提供大量适合于任何 Widget 类的函数(如：XtCreateManagedWidget()，XtMapWidgetXtParent()，以及其它的许多函数)来遵循这种作法。无论你想将哪类 Widget 映象显示，你只要知道共同函数 XtMapWidget()将做这一项工作即可。此外，Xlib 和 Xt 也遵循其它的命名协议。例如：一般函数中的动词，象"get"，"install"，"manage"，"set"出现在 Xlib 中紧跟在"x..."后面，Xt 中紧跟在"Xt..."后面。除了使用函数命名的协议外，Xlib 和 Xt 还遵循一定的关于函数调用中参数/变量次序的协议。例如，对于 Widget 和窗口尺寸的函数总是按顺序给定：...<width>,<height>,...

在可能的情况下，应用程序程序设计人员在开发诸如 mak—window()，create—GC()等高级函数中，应该按照这种办法来取名。我们并不希望所有创建的函数都使用相同诸如"make"或"create"等等的动词。作为程序设计人员，我们不希望受到教条的程序设计方法的限制，我们希望有创造和表达我们独特方式的机会。然而，有许多机会来增加传递给其他程序设计人员源代码的可读性。

对于 Widget 程序设计人员，Xt Intrincics 和 Athena Widget 集合都是为发展标准协议的使用而设计的。首先，正如我们提到的，只要你遵循已经建立起来的 Widget 的书写协定，Xt

Intrincics 能在任何你新开发的 Widget 中提供使用公有函数。例如，如果你开发一个新的 Widget 在它的创建期或它的使用期的任何时刻来动态分配内存，你必须简单地提供释放内存的方案，在 Widget 类的适当记录中注册。Xt Intrincics 自动给应用程序程序设计者为这个方法提供一个称作 XtDestroyWidget()的公共接口。

Athena 和 Motif Widget 集合在对每个 Widget 的公用接口使用命名约定中，支持标准协议。考虑如下的说法：XtDialogGetValueString()来自 Athera Dialog Widget 类，XtFormDolayout()来自 Athena Form Widget 类，而 XtPanedAllowResize()和 XtPanedSetMinMax()来自 Athera 的 Vpaned Widget 类(对于 X11 Release 3)。更重要地，Xt 支持其它 Widget 集合与 Athera Widget 集合的平滑集成。Nye 和 Okeilly(1990，附录 D)讨论了 Widget 程序设计中假定的命名约定。我们将在关于 Widget 程序设计的后面章节中提出更详细的细节。

2.5 Xt 的 Widgets 与 C++的对象的比较

虽然我们在本书中对 C++没有特殊的兴趣，但我们还是应该对 Xt Widgets 与 C++对象之间在设计上的区别作一个简要的了解。因为在这里我们假设没有任何有关 C++的知识，因此我们只考虑开发 C++类的核心问题就可以了。同时，类的概念对 C++的类及 Xt Widget 的类也是足够的。一个关于 C++类的开发的更完整的讨论，在 Stroustrup (1986)，Lippman (1989)，Smith (1990)及许多其他人的书中给出，Widget 程序设计的更完整的讨论在后面的章节中给出。

2.5.1 C++类的层次

考虑如下的 C++类的定义：

```
class window {
protected:
    int width, height;        // every window has these variables
    ...
    <other variables>...
    window(<parameter types>...);          // the class constructor
    ...
    void activate(<parameter types>...);  // method prototypes
    <other methods>...
public:
    <public methods>...                   // the class interface
};
```

如果你对 C++不熟悉，只要注意类(类)是结构(struct)的一个增强型就足够了。类可以有专用的、保护的及公有的部分，及其成员变量和成员函数(方法)。专用部分的(变量和方法)成员只能被它所属的类所存取；保护成员只能被它所属的类及其派生类存取；公有成员可被任何外部访问。

window 是一个抽象的基类。window 是抽象的，因为它是不完全的；它只包括那些对所有窗口都相同的变量。window 是一个基类，因为它为派生其它类(子类)的用途而设计；它在一个窗口类层次中是根类。

使用 C++，控制对变量和方法的访问是很容易的。正如所提到的，一个类定义的保护部分的成员只能由该类及该类的子孙所访问。关键字 public 用于指出类的接口。典型地，只

有方法能被设置成公有的；一个类的成员变量可由该类和它的子孙类的方法(mothod)访问，或者由外部通入一个存取函数/方法来访问。(我们省略了C++的friend构成成份)。

每个类有一个构造函数(在下面例子中显示)以及一个析构函数(没有举例)。在为一个对象分配内存期间，要调用构造函数。典型地，就是为这个对象设计接收初始化信息。

```
response_box resp_box1(<width>, <height>, ...);
```

在释放存储空间时要调用析构函数。

前面我们讨论了继承性——面向对象语言所支持的三个主要概念中的一个。继承性促进根据区别来进行程序设计的方法。类的层次被设计，以致根类成为一个通用的基类，被设置在这个层次末端点的类是更专用的，也就是更专业化。

一个类是由它的超类(基类)派生来的，是通过增加一些与它的超类区别的类变量及方法来实现的。

假设我们希望从Window类派生一个更专用的类——支持某些正文处理操作。用C++，派生类及基类的名字按如下指定：

```
class text_window : public window {
protected:
    int cursor_x, cursor_y;      // text cursor coordinates
    ...
    <other variables>...
    <methods>...
public:
    int get_cursor_x(void);      // public method prototypes
    ...
    <other public methods>...
};
```

首先，指定新的类，在这里是text _ window。此处，基类在关键字public后面指定。类的定义包括那些把一个正文窗口同一个一般窗口区别开的变量及方法。根据协定，新类继承了它的超类的变量和方法(除了那些在带有关键字private的专用节中所指定的变量及方法之外)。

最后，一个更专用的类就生成了，也就是一个支持正文处理的类。

```
class response_box : public text_window {
protected:
    char *response;
    ...
public:
    char *get_response(void);        // public method prototypes
    ...
};
```

在这里，我们的类定义为如下含义：当一个对话框被激活之后，一个用户作出了回答；而当该对话框未被激活，用户的回答能像在任何时候使用访问方法函数get－response()得到，该函数返回一个指向变量response所引用数据的指针。