第 一 篇

X 工具箱内部机制

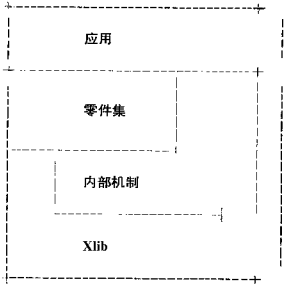
第一章　内部机制和零件

1.1　引言

X 工具箱是由内部机制和零件集这两者共同构成的。内部机制为建造形式多样的零件集和应用提供了必需的基本机制。由于内部机制对零件和应用程序员隐藏了其实现细节，因此用它们建造的零件和应用环境是完全可扩展的，同时它们也支持独立地开发新的部件或扩充原有的部件。零件程序员只要遵循一个较小的规则集，就可以用新方法来扩充其零件集，并使这些扩充零件与已有的设施协调地工作。

内部机制是一个位于 Xlib 之上的库函数包。它提供了一些机制(函数和结构)，用于扩充 XWIN 系统所提供的基本程序设计的抽象能力。内部机制通过为部件间和部件内的相互作用提供机制，提供了另一个功能层，零件集就可借此来建立。

图 1-1：　阐明了这一扩充的三层编程环境



一个典型的 X 工具箱应用多半是一个客户，它使用着一个给定的零件集、一个内部机制函数的子集和一个更小的 Xlib 函数集。通过从左到右浏览图 1-1，就可看出这点。同时，通过自上而下浏览图 1-1 可以看出，零件集是内部机制和 Xlib 的客户，而内部机制则仅仅是 Xlib 的客户。

对应用程序员而言，X 工具箱提供了：

- 编写应用程序的一致性接口(零件集)
- 用于编写应用程序的少量内部机制。

对零件程序员而言，X 工具箱提供了：

- 用于建造零件的内部机制集。
- 用于建造零件和组合零件的结构模型。
- 用于程序设计的一致性接口(零件集)。

应用环境(而非 X 工具箱)定义、实现和增强了下列各项：

- 策略
- 一致性
- 风格

每个零件的实现都定义了其自己的策略。X 工具箱被设计成允许对根本不同的零件的实现进行开发。

1.2 术语

除了已为 X 编程所定义的术语之外(参见《Xlib－C 语言 X 界面》)，全书中还会出现下列针对内部机制的术语：

应用程序员(Application programmer)

使用X工具箱开发应用用户界面的程序员。

类(Class)

某个特定对象归属于其中的通用组。

客户(Client)

在应用中使用零件的函数，或使用零件来组成其它零件的函数。

实例(Instance)

一个特定的零件对象，它的对方就是泛指的零件类。

方法(Method)

零件类实现的函数或过程。

名(Name)

专用于给定客户的某一零件实例的名字。

对象(Object)

由私用数据及在其上操作的公用和私用函数所组成的软件数据抽象。这一抽象的用户只有通过调用对象的公用函数才能与该对象打交道。在 X 工具箱中，对象的某些公用函数可以由应用直接调用，而其它的则在应用调用公用内部机制函数时被间接调用。通常，如果一个函数对所有零件都通用，则对于任何类型的零件，应用就都可使用一个内部机制

函数去调用这个函数。如果某个函数专用于某个零件类型，则该零件就会将这个函数作为另一个"Xt"函数来开放。

资源(Resource)

零件中已命名的一些数据，它们可以由客户、应用或用户缺省值来设置。

用户(User)

与工作站进行交互作用的人。

零件(Widget)

提供用户界面抽象物的对象(例如，滚动杆零件)。

零件类(Widget Class)

一特定零件所属的通用组，也可以叫做零件的类型。

零件程序员(Widget programmer)

为X工具箱增加新零件的程序员。

1·3 内部机制

内部机制提供了简化应用用户界面设计的基本机制(函数和结构)。此外，它还通过提供一组公用的基本用户界面函数，来帮助零件程序员和应用程序员管理：

- 工具箱初始化零件
- 内存
- 窗口、文件和计时器事件
- 零件的几何形状
- 输入焦点
- 选择
- 资源和资源转换
- 事件的转换
- 图形环境
- 像素映像
- 错误和警告

虽然所有的内部机制功能都主要是由零件程序员来使用，但也有少量的可为应用程序员所用。内部机制的结构模型的存在，使得零件程序员可以通过使用提供的机制和/或通过组合已存在的零件来创建新零件。因此，用内部机制建立的应用界面层，为零件和组合策略提供了一个协调的场地。某些用内部机制建造的零件为许多应用领域所公用，而其它一些零件则限于专用的应用领域。

内部机制基于一个结构模型，而这个模型又很灵活，足以对付各种不同的应用界面层。此外，所提供的内部机制集具有下列特性：

- 功能完整和策略自由
- 在风格和功能上与XWIN系统原语一致。
- 对语言、计算机体系结构和操作系统而言，均可移植。

使用内部机制的应用程序都必须包含下列头文件：

- <Xll/Intrinsic.h>
- <Xll/StringDefs.h>

此外，它们还可包含头文件：

- <Xll/Xatoms.h>
- <Xll/Shell.h>

最后，零件实现应包含：

- <Xll/IntrinsicP.h>，而不是<Xll/Intrinsic.h>

应用还应包含它将使用的每个零件类的附加头文件(例如，<Xll/Label.h>或<Xll/Scroll.h>)。在基于 UNIX 的系统上，内部机制对象库文件名为 libXt.a，通常可利用-lXt 来将其连入应用中。

1.4 零件

零件是 X 工具箱的基本抽象物和数据类型。零件是 X 窗口及其相关语义的一个组合，它被动态分配并含有状态信息。在逻辑上，零件是带有相关的输入/输出语义的矩形框。一些零件要显示信息(如，文字或图形)；而另一些零件则仅仅是其它零件的容器(如，菜单框)。某些零件仅可输出，而对指示器或键盘输入没有反应；另一些零件则在响应输入后要改变其显示内容，并可以调用应用赋予它们的功能。

每个零件都只属于一个被静态分配和初始化的零件类，该零件类包含着允许对此类零件进行的操作。逻辑上，零件类是属于此类的所有零件的相关联的过程和数据。这些过程和数据可以由子类继承。物理上，零件类是一个指向结构的指针。该结构的内容对零件类的所有零件来说是常量，但各类之间的均各不相同。(在此，常量表示类结构将在编译时被初始化且不再改变，除在建立该类或子类的第一个零件时，要进行的一次性初始化和资源表的现场编译之外。)详见第二章中的"建立零件"。

在公用.h 文件、私用.h 文件和实现的.c 文件之间，如何组织一个新零件类的说明部分和代码部分，将在本章的"零件分类"一节中描述。预定义的零件类遵从这些规则。

零件实例由两部分组成：

- 数据结构，其中含有实例专用值。
- 类结构，其中含有可为此类的所有零件所用的信息。

零件的许多输入/输出特性(例如，字体、颜色、大小、边框、宽度等等)都可由用户来加入塑造。

接下来的三小节将讨论下述三种基本零件类：

- 核心零件
- 复合零件
- 约束零件

本章将以对零件分类的讨论作为结束。

1·4·1 核心零件

核心零件(Core)包含着为全部零件所公用的域的定义。所有零件都是 Core 的子类。Core 由 CoreClassPart 和 CorePart 结构来定义。

1·4·1·1 CoreClassPart 结构

所有零件类的公用域都是在 CoreClassPart 结构中定义的:

```
typedef struct {
WidgetClass superclass;                    参见第一章中的"零件分类"
String class_name;                         参见第一章中的"零件分类"
Cardinal widget_size;                      参见第二章中的"建立零件"
XtProc class_initialize;                   参见第一章中的"零件分类"
XtWidgetClassProc
        class_part_initialize;             参见第一章中的"零件分类"
Boolean class_inited;                      参见第一章中的"零件分类"
XtInitProc initialize;                     参见第一章中的"零件分类"
XtArgsProc initialize_hook;                参见第二章中的"建立零件"
XtRealizeProc realize;                     参见第二章中的"建立零件"
XtActionList actions;                      参见第十章
Cardinal num_actions;                      参见第十章
XtResourceList resources;                  参见第九章
Cardinal num_resources;                    参见第九章
XrmClass xrm_class;                        资源管理器私用
Boolean compress_motion;                   参见第七章中的"指示器移动事件的压
                                           缩"
Boolean compress_exposure;                 参见第七章中的"暴露事件的压缩"
Boolean compress_enterleave;               参见第七章中的"进入/离开事件的压
                                           缩"
Boolean visible_interest;                  参见第七章中的"零件暴露和可见性"
XtWidgetProc destroy;                      参见第二章中的"消毁零件"
XtWidgetProc resize;                       参见第六章
```

```
XtExposeProc expose;                    参见第七章中的"零件暴露和可见性"
XtSetValuesFunc set_values;             参见第九章中的"读和写零件状态"
XtArgsFunc set_values_hook;             参见第九章中的"读和写零件状态"
XtAlmostProc set_values_almost;         参见第九章中的"读和写零件状态"
XtArgsProc get_values_hook;             参见第九章中的"读和写零件状态"
XtAcceptFocusProc accept_focus;         参见第七章中的"子零件与输入焦点"
XtVersionType version;                  参见第一章中的"零件分类"
XtOffsetList callback_private;          回调过程私用
String tm_table;                        参见第十章
XtGeometryHandler query_geometry;       参见第六章
XtStringProc display_accelerator;       参见第十章
caddr_t extension;                      参见第一章中的"零件分类"
} CoreClassPart;
```

所有零件类都把核心类域作为其第一个成分。样本零件类 WidgetClass 只用这样的域集来进行定义。如必要，各种例程都可以让零件类指针指向特定零件类类型，例如：

```
typedef struct {
    CoreClassPart core_class;
} WidgetClassRec, *WidgetClass;
```

为 WidgetClassRec 预定义的类记录和指针是：

```
extern WidgetClassRec widgetClassRec;
extern WidgetClass widgetClass;
```

不透明类型 Widget 和 WidgetClass 以及不透明变量 widgetClass，是为零件上的一般动作所定义的。

1.4.1.2 CorePart 结构

所有零件实例的公用域都定义在 CorePart 结构中：

```
typedef struct _CorePart {
Widget self;
WidgetClass widget_class;               参见第一章中的"零件分类"
Widget parent;                          参见第一章中的"零件分类"
XrmName xrm_name;                       资源管理器私用
Boolean being_destroyed;                参见第二章中的"消毁零件"
```

XtCallbackList destroy_callbacks;	参见第二章中的“消毁零件”
caddr_t constraints;	参见第三章中的“受约束复合零件”
Position x;	参见第六章
Position y;	参见第六章
Dimension width;	参见第六章
Dimension height;	参见第六章
Dimension border_width;	参见第六章
boolean managed;	参见第三章
Boolean sensitive;	参见第七章中的“设置和检查零件的灵敏状态”
Boolean ancestor_sensitive;	参见第七章中的“设置和检查零件的灵敏状态”
XtEventTable event_table;	事件管理器私用
XtTMRec tm;	转换管理器私用
XtTranslations accelerators;	参见第十章
Pixel border_pixel;	参见第二章中的“从零件中获取窗口信息”
Pixmap border_pixmap;	参见第二章中的“从零件中获取窗口信息”
WidgetList popup_list;	参见第五章
Cardinal num_popups;	参见第五章
String name;	参见第九章
Screen *screen;	参见第二章中的“从零件中获取窗口信息”
Colormap colormap;	参见第二章中的“从零件中获取窗口信息”
Window window;	参见第二章中的“从零件中获取窗口信息”
Cardinal depth;	参见第二章中的“实现零件”
Pixel background_pixel;	参见第二章中的“从零件中获取窗口信息”
Pixmap background_pixmap;	参见第二章中的“从零件中获取窗口信息”
Boolean visible;	参见第七章中的“零件暴露和可见性”
Boolean mapped_when_managed;	参见第三章

```
} CorePart;
```

所有零件实例都把核心域作为其第一个成分。样本类型 Widget 仅用这样的域集来进行定义。如必要，各种例程都可以让零件指针指向某个零件类型，如：

```
typedef struct {
    CorePart core;
} WidgetRec, *Widget;
```

1·4·1·3 CorePart 缺省值

由 Core 资源表和 Core 初始化过程所填写的各核心域的缺省值为：

域名	缺省值
$self	零件结构的地址(不能改变)
widget_class	XtCreateWidget的widget_class参数(不能改变)
parent	XtCreateWidget的parent参数(不能改变)
xrm_name	XtCreateWidget的已编码name参数(不能改变)
being_destroyed	父零件的being_destroyed值
destroy_callbacks	NULL(空)
constraints	NULL(空)
x	0
y	0
width	0
height	0
border_width	1
managed	False(假)
sensitive	True(真)
ancestor_sensitive	父零件的sensitive与ancestor_sensitive的按位与
event_table	由事件管理器初始化
tm	由转换管理器初始化
accelerators	NULL(空)
border_pixel	XtDefaultForeground
border_pixmap	NULL(空)
popup_list	NULL(空)
num_popups	0
name	XtCreateWidget的name参数(不能改变)

域名	缺省值
screen	父零件的屏幕，顶层零件可从显示器说明符处获得它(不能改变)
colormap	屏幕的缺省彩色映像
window	NULL(空)
depth	父零件的深度，顶层零件可获得根窗口的深度。
background_pixel	XtDefaultBackground
background_pixmap	NULL(空)
visible	True(真)
map_when_managed	True(真)

1·4·2 复合零件

复合零件是核心零件 Core 的子类(参见第三章)。它是其它零件的容器，并由 CompositeClassPart 和 CompositePart 来结构定义。

1·4·2·1 CompositeClassPart 结构

除了核心零件的类域外，复合零件还具有下列类域：

```
typedef struct{
        XtGeometryHandler
                geometry_manager;       参见第六章
        XtWidgetProc change_managed;    参见第三章
        XtWidgetProc insert_child;      参见第三章
        XtWidgetProc delete_child;      参见第三章
        caddrt extension;               参见第一章中的"零件分类"
} CompositeClassPart;
```

复合零件类的核心域之后便是复合域：

```
typedef struct{
        CoreClassPart                   core_class;
        CompositeClassPart              composite_class;
} CompositeClassRec,                    *CompositeWidgetClass;
```

为 CompositeClassRec 预定义的类记录和指针是：

```
extern  CompositeClassRec  compositeClassRec;
extern  WidgetClass  compositeWidgetClass;
```

不透明类型 CompositeWidget 和 CompositeWidgetClass 以及不透明变量 composite-WidgetClass，是为某些零件上的一般操作所定义的，这些零件是 CompositeWidget 的子类。

1·4·2·2 CompositePart 结构

除了 CorePart 域外，复合零件还具有在 CompositePart 结构中定义的下列域：

```
typedef struct{
        WidgetList children;              参见第一章的"零件分类"
        Cardinal num_children;            参见第一章的"零件分类"
        Cardinal num_slots;               参见第三章
        XtOrderProc insert_position;      参见第三章的"创建零件"
} CompositePart;
```

复合零件中，在 core 域之后便是 composite 域：

```
typedef  struct {
         CorePart core;
         CompositePart composite;
} CompositeRec, *CompositeWidget;
```

1·4·2·3 CompositePart 的缺省值

由 Composite 资源表和 Composite 初始化过程所填写的复合域的缺省值为：

域名	缺省值
children	NULL(空)
num_children	0
num_slots	0
insert_position	内部函数InsertAtEnd

1·4·3 约束零件

约束零件是复合零件的子类(参见第三章的"受约束复合零件"一节)，它维持着每个子

零件的额外状态数据，例如，客户所定义的对子零件几何形状的约束。约束零件由 ConstraintClassPart 和 ConstraintPart 结构定义。

1.4.3.1 ConstraintClassPart 结构

除复合类的各域外，约束零件还具有下列类域：

```
typedef struct{
XtResourceList resources;      参见第三章的"受约束复合零件"
Cardinal num_resources;        参见第三章的"受约束复合零件"
Cardinal constraint_size;      参见第三章的"受约束复合零件"
XtInitProc initialize;         参见第三章的"受约束复合零件"
XtWidgetProc destroy;          参见第三章的"受约束复合零件"
XtSetValuesFunc set_values;    参见第三章的"受约束复合零件"
carddr_t extension;            参见第一章的"零件分类"
} ConstraintClassPart;
```

各约束零件类的复合域后便是约束域：

```
typedef struct{
        CoreClassPart       core_class;
        CompositeClassPart composite_class;
        ConstraintClassPartcomstraint_class;
} ConstraintClassRec,       *ConstraintWidgetClass;
```

为 ConstraintClass 预定义的类记录和指针是：

```
extern   ConstraintClassRec   constraintClassRec;
extern   WidgetClass  constraintWidgetClass;
```

不透明类型 ConstraintWidget 和 ConstraintWidgetClass 以及不透明变量 ConstraintWidgetClass，是为零件上的一般操作所定义的，这些零件是 ConstraintWidgetClass 的子类。

1.4.3.2 ConstraintPart 结构

除 CompositePart 域外，约束零件具有下列在 ConstraintPart 结构中所定义的域：

```
typedef struct {int empty; } ConstraintPart;
```

各约束零件的 composite 域后便是 constraint 域：

```
typedef struct{
        CorePart core;
        CompositePart composite;
        ConstraintPart constraint;
} ConstraintRec, *ConstraintWidget;
```

1·5 零件分类

零件的 widget_class 域指向其零件类的结构，该结构包含了对此类的所有零件来说为常数的信息。因此，零件类通常不直接实现可调用过程，而是实现可通过其零件类结构来调用的过程。这些方法由通用过程所调用，而这些过程使公共操作被零件类实现的过程所遮蔽，此类的所有零件及其子类的零件都可以使用这些过程。

所有零件类都是 Core 的子类，并可以被进一步子类化。若新零件类类似于一个已存在的类，子类化就可以减少产生一个新零件类所需的代码和说明。例如，不必在 XtResourceList 中描述新零件使用的每个资源，而只需描述新零件具有但其超类没有的资源。子类通常要继承其超类的许多过程(例如，暴露过程、几何处理器等)。

但是，子类化可能进行得过分。如果创建了一个没有继承到其超类的任何过程的子类，就应考虑是否选择了最合适的超类。

为了很好地进行子类化，零件说明和命名规则都予以高度地格式化。零件由三个文件所构成，即：

- 公用的.h文件，由客户零件或应用所使用。
- 私用的.h文件，由属于零件子类的零件所使用。
- .c文件，实现零件类。

1·5·1 零件命名规则

内部机制为程序员提供了用以创建新零件并将一组零件组织到应用中去的手段。为了保证应用不需处理它所使用的许多零件类的大小写格式和拼法，在编写新的零件时，应遵循下列准则：

- 保持与应用的X命名规则。例如，记录的域名都用小写字母表示，对于复合词可用下划线(_)连接(如，bakground_pixmap)。类型和过程名则以大写字母开头并对复合词的各单词也以大写字母开头(如，ArgList 或 XtNValues)。

  用大写而非下划线连接外，资源名串完全与域名的拼写一样。为了发现拼写错误，每个资源名都应该有一个带 XtN 前缀的宏定义。例

如，background pixmap 域具有对应的资源名标识符 XtNbackgroundPixmap，后者被定义为串“backgroundPixmap”。许多预定义的名字都列在<X11/StringDefs.h>头文件中。在创造新名字之前，应确信这个名字还没有定义过或你能使用的名字中还不存在。

- 资源类串用一个大写字母开始并对复合名字使用大写连接形式（例如，“BorderWidth”）。每个资源类串都应有一个带 XtC 前缀的宏定义（如，XtCBorderWidth）。
- 资源表示串完全与类型名的拼写一样（如，“TranslationTable”）。每个表示串应有一个带 XtR 前缀的宏定义（如，XtRTranslationTable）。
- 新零件类以一个大写字母开始，并对复合词使用大写字母连接。给定一个新类名 AbcXyz，应能衍生出多个名字：
  - 部分零件实例结构名，AbcXyzPart
  - 完整的零件实例结构名AbcXyzRec和_AbcXyzRec。
  - 零件实例指针类型名AbcXyzWidget。
  - 部分类结构名AbcXyzClassPart。
  - 完整的类结构名AbcXyzClassRec和_AbcXyzClassRec。
  - 类结构变量abcXyzClassRec。
  - 类指针变量abcXyzWidgetClass。
- 转换说明可使用的动作过程应遵循与一般过程一样的命名规则，即它们以一个大写字母开始，且复合名字使用大写字母连接（如，“Higlight”和“NotifyClient”）。

1·5·2 公用·h文件中的零件子类化

一个零件类的公用·h文件由客户引入。它包含如下内容：

- 对超类的公用·h文件的一个引用。
- 该零件添加到其超类中的各新资源的名字和类。
- 创建零件实例所使用的类记录指针。
- 说明该类的零件实例所使用的C类型。
- 新的类方法的入口点。

例如，下面是可用来实现某个 Label（标签）零件的公用·h 文件：

```
#ifndef LABEL_H
#define LABEL_H

/*New resources */
#define XtNjustify                "justify"
```

```
#define XtNforeground            "foreground"
#define XtNlabel                 "label"
#define XtNfont                  "font"
#define XtNinternalWidth         "internalWidth"
#define XtNinternalHeight        "internalHeight"

/*Class record pointer*
extern WidgetClass labelWidgetClass;

/*C Widget type definition */
typedef struct _LabelRec     *LabelWidget;

/*New class method entry points */
extern void Label SetText();
    /*Widget w */
    /*String text */

extern String Label GetText();
    /*Widget w */

#endif LABEL_H
```

该文本的条件包含语句，允许应用程序包含用于不同零件的头文件，而不必关心这些头文件是否已作为另一零件的超类而包含进了某个文件。

为了符合带文件名长度限制的操作系统的要求，公用·h 文件的名字将取零件类的前10个字符。例如，约束零件的公用·h 文件名为 Xll/Constraint.h。

1·5·3 私用·h 文件中的零件子类化

零件的私用·h 文件由作为该零件的子类的那些零件类引入，私用·h 文件包含如下的内容：

- 对类的公用·h文件的一个引用。
- 对超类的私用·h文件的一个引用。
- 零件实例添加到其超类的零件结构中的新域。
- 该零件的完整的零件实例结构。
- 在零件是Constraint(约束零件)的子类时，该零件添加到其超类的Constraint结构中的新域。

- 在零件是Constraint(约束零件)的子类时，完整的Constraint结构。
- 该零件类添加到其超类的零件类结构中的新域。
- 该零件的完整的零件类结构。
- 通用零件类结构的一个常量名。
- 子类的一个继承过程，这些子类希望为零件类结构中的每个新过程继承一个超类操作。

例如，下面是一个可能的标签零件的私用.h文件：

```
#ifndef LABELP_H
#define LABEIP_H

#include <X11/Label.h>

/*New fields for the Label widget record */
typedef struct {
/*Settable resources */
  Pixel foreground;
  XFontStruct *font;
  String label;                    /*text to display */
  XtJustify justify;
  Dimension internal_width;        /*#of pixels horizontal borde */
  Dimension internal_height;       /*#of pixels vertical border */
/*Data derived from resources */
  GC normal_GC;
  GC gray_GC;
  Pixmap gray_pixmap;
  Position label_x;
  Position label_y;
  Dimension label_width;
  Dimension label_height;
  Cardinal label_len;
  Boolean display_sensitive;
} LabelPart;

/*Full instance record declaration */
typedef struct _LabelRec {
```