第29章　定制X Window系统

29.1　使用X的字体和颜色

X支持多种的字体以及几乎无限多种变化的颜色。大多数的应用程序允许你指定应用窗口中各个不同部分的颜色，而几乎所有的X程序均允许你指定你想要使用的字体。

X中的字体：

- 有固定的宽度（像哑终端机的字符）或成比例的间隙。
- 由文本字符或符号组成，或以上两者均有。
- 具有多种的磅尺寸。
- 可以修改以适应特定的屏幕分辨率（例如对于同一点尺寸的某一种字体，你可能对 75 dpi 的屏幕有一种版本，对100 dpi 的屏幕有另一种版本）。
- 有一种标准命名的传统。
- 可以以全名存取，也可以用通配符。
- 保存在特定配置的目录树中，只要服务器在执行，字体便可以加入或移出。

在系统间进行字体的交换有一套标准的格式，并且有工具程序可以将这个格式转换成你的服务器能了解的格式，工具程序也包含了列出可用字体的目录、观察某一特定字体内容等功能。

29.1.1　字体初步

本节的目的是让你尽快地能使用字体，我们将告诉你如何找出有哪些字体可用、指定你欲使用的字体名、看字体的外观、如何在X应用程序中使用字体。

1. 字体命名

有些字体名太长以致使用不便，但很幸运，它们也不常使用，并且， X支持字体名可使用通配符：

- ? 对应任何一个字符。
- * 对应从（字符）长度为零至长度若干的字符串。

这和UNIX外壳传统的通配符文件名相同，使用通配符可使你更容易指定字体名。

注意　如果你在外壳程序的命令行指定一个通配符的字体名，需要把名称加上双引号 。

2. 观察特定的字体

xfd (X font displayer 的缩写) 程序由参数得到字体的名称之后，建立一个窗口并且在窗口中显示此名称的字符字体，例如：

```
xfd -fn "*symbol*-180-*"
```

3. 以X程序使用字体

大多数的X程序使用文字，并且允许你指定使用的字体。如何使用的详细细节可能因不同的程序而异，如果有问题的话可以看联机帮助。但是几乎都是以命令行中选项 -fn fontname 或 -font fontname 来指定字体名， bitmap、xclock、xterm、xload、xmb 和 xedit都是这样操作的。

例如假设你是为了展示的缘故，以很大的字体执行 xterm ，你可以用下列命令行：

```
xterm -fn "*courier-bold-r-*-240-*"
```

注意 如果你给程序的指定对应到一种以上的字体，则服务程序会随便在其中选取一个，例如：如果你省略了上例中的 -r 的指定，则你会使用到斜体字体或反斜体字体，和原来所指定的罗马字体的机会是一样的。

29.1.2 字体命名

在X中，字体可以取成任何名称，但几乎所有的字体均依照它们的本质来命名，这样的命名方式，名字是由几个不相关的部分组合而成，而我们在使用应用程序时，光凭着字体名便可以大略了解字体的内涵。

我们以一个字体名为例，逐一解释它的组件，组件之间是用短横线 (-)分开的，而且可以包含空白，字体名对字符大小写并不会区别，样例如下：

```
-adobe-times-bold-normal--12-120-75-75-p-67-iso8859-1
```

- adobe ：字体的制造厂商。
- times ：字型家族，其他尚包含courier、helvetica 和new century schoolbook。
- bold ：粗体字，其他包含light （细）和medium（中等）。
- normal.字体。例如， r 是roman（罗马体）， i是italic(意大利体）， o是oblique(倾斜体）。
- 12：字符的高度，单位为像素。
- 120 ：字体的磅尺寸，为磅的10倍（120 意为12磅）。
- 75-75 ：字体被设计在显示设备上的水平和垂直的分辨率。
- p ：字和字之间的间隙， p 是proportional（成比例的），相对的是m （monospaced，固定宽度）。

常用的项目为字型、字体粗细、何种斜体字以及字体大小，除了指定这几项的值外，其他的项目不妨借助于通配符的方式去指定。

1. 通配符和字体名

我们曾经解释过通配符的规则：星号（*）表示对应零或多个字符，问号（?）对应任意的单一字符。

你可以随意地使用通配符。当你的设定对应一种以上的可用字体时，服务程序会随便挑一种字体来用，如果你没有设定字体，通常会获得一行信息，而服务程序将会使用缺省字体。

你可以对字体的点尺寸使用通配符，而不是像素尺寸，因为在显示器上一个给定点尺寸的字体对不同的分辨率有不同的像素尺寸，所以用通配符指定点尺寸可以造成与装备无关的效果，上述的样例你可以这样设定：

```
*-times-bold-r-*-120-*
```

也就是说以-120-替换-12-

2. 列出可用的字体 —— xlsfonts

xlsfonts可列出在服务程序中可用的字体（如果你使用命令行中 -display选项，便可列出其他服务程序中可用的字体）。缺省值是列出所有的字体，但是就如同 UNIX的ls命令一样，如果你加上限制，便只会列出合乎限制的项目，例如：

```
xlsfonts  "*-times-*-180-*"
```

列出所有18点Times的字体。

原则上， xlsfonts试图在每行打出尽量多的字体名称，但实际上，大部分的字体名称都很

长，以致一次只能印一个名称，但是要小心，当字体名含有空格时，一行有数个字体名，常常容易混淆。

注意 许多的字体名开头为一短横线(-)，所以xlsfonts会误把此种状况当成命令行的选项来解释以致发生错误，例如：

```
xlsfonts  "-adobe-*"
```

会失败，你可以用选项-fn 加以区分，或者只要在设定之前加一个星号（*）即可：

```
xlsfonts  "*-adobe-*"
xlsfonts  -fn "-adobe-*"
```

29.1.3 观察特定字体的内容——xfd

xfd 是一个字体显示的程序，它建立一个窗口，而后在窗口中将字体的元素显示在长方格中。窗口可能没有大到一次将字体中所有的字符显示出来（尤其是你可能对它重定过大小），但你仍然可以存取它们：

- 向前移动：在xfd 窗口中按鼠标右按钮，窗口的下一页将会出现。
- 向后移动：按鼠标左按钮。
- 获取字符的信息：在字符上按鼠标中按钮， xfd 会给你字符号码，如果你在程序一开始设定命令行选项 -verbose，你将获得一些更多的信息，例如字符的大小以及它在字符“ cell ”中的位置。

29.1.4 保存字体和位置

本节描述字体不同的格式，以及转换两种不同格式的工具，然后讨论服务程序是如何存取字体和你如何更改对字体的选择。最后，我们会给一个完整的样例来说明如何为系统加入一种新的字体。

1. 字体的格式——SNF（Server Natural Format）

字体在服务程序上是以SNF 方式保存，这种格式并不是一种标准，而且为服务程序所专用，所以你不能将字体移到不同类型的服务程序中。

showsnf 程序输出保存在SNF 文件中字体的信息，对字体本身执行 xprop 可获得更多类似的信息(showsnf的参数为文件名，xprop 则为字体名，字体名和文件名并不相关)。

(1) Bitmap Distribution Format（位图分布格式）——BDF

为了克服字体流传的问题，X协会对字体交换指定了一种格式，就是BDF ，BDF 以ASCII的方式表示字符的图形，并且只包含可输出的字符，所以它具有完整的可移植性。

在“ Bitmap Distribution Format ”文件中包含了对BDF 完整的描述。

(2) 从BDF 转换成SNF——bdftosnf

为了让BDF 能够有用，你必须能将BDF 字体文件转换成SNF 文件，目前X协会放弃让这个需求成为X的产品。

在MIT 版中，你可以用bdftosnf来完成转换。

(3) 由其他的格式转换

许多绘图机器拥有它们制造商自己开发的字体，通常特别适合它们的显示器。如果这些字体能在X中使用，那是再好也不过了，但是因为格式的问题，你不能使用它们。

MIT core版并不管这个问题，但是 core版有许多工具程序可将制造商特制的字体转换成BDF 格式，从BDF 你又可以用bdftosnf转换成你自己的SNF。

2. 字体保存在何处 —— 字体目录

字体保存在服务程序上某一个或多个字体目录中，字体目录由三个部分组成：

1) 一个普通的目录，为包含着字体的SNF 文件之所在。

2) 一个被X使用，将SNF 文件名对应到字体名称的数据库。

3) 一个可选择性的别名文件，可以让你用一个以上的名称参考到同一字体（不论你使用了多少个目录，你只需要一个别名文件）。

3. 维护字体目录 —— mkfontdir

mkfontdir 设定新的字体目录并且可以修改它：

1) 在文件目录中搜集了所有你要使用字体的文件，文件可以是 BDF 文件（通常文件名结尾为 .bdf）、SNF 文件(.snf)或被压缩的SNF 文件(.snf.Z)，mkfontdir会自动将非SNF 文件转换为SNF 文件（被压缩的文件是被BSD 压缩程序执行过用以节省文件空间）。

2) 如果你要使用别名，需要在字体目录中建立（或编辑）一个名为 fonts.aliase的文件。有关此文件格式的细节部分在联机帮助中有说明。简单地说，它的格式为每行以空白间隔出两个栏位，第一栏是别名的名称，第二栏则是字体的名称（可包含通配符），例如：

```
tbi12  *-times-bold-i*-120*
```

注意 你对字体定义的第一个别名将造成该字体真正的名称无法使用，以上例而言，你只能以tbi12 来存取字体，这种情形也许下一版会改进，但目前你可以在第二行将第一行反过来即可（但不可使用通配符）。

```
tbi12  *-times-bold-i*-120*
-adobe-times-bold-i-normal--12-120-75-755-p-68-iso8859-1 tbi12
```

3) 执行mkfontdir ，需把文件名当成参数输入，以你使用缺省的 X配置为例 ：

```
mkfontdir /usr/lib/x11/fonts/misc\
      /usr/lib/x11/fonts/75dpi\
      /usr/lib/x11/fonts/100dpi
```

（如果文件目录中没有包含字体数据库， mkfontdir 会忽略它。）

注意 建立字体目录并不会导致服务程序注意它，你必需重新启动服务程序或重设字体搜索路径。

4. 字体搜索路径 —— xset

你可以使用任何数目的字体目录，但如果它们有任何和缺省配置不同的地方，你需明确地告诉服务程序，这些字体目录的列表称之为字体搜索路径或字体路径，你可以设定这个一连串以逗点分隔的文件目录。

• 查看目前的字体路径：使用命令 xset q，这样会输出一大堆信息，其中有一行包含着你的字体路径如下：

```
Font Path : /usr/lib/x11/fonts/misc/,(cond.)
/usr/lib/x11/fonts/75dpi/,/usr/lib/x11/fonts/100dpi/
```

• 设定不同的字体路径：使用命令 xset fp new-path，例如，如果你有大量的本地字体且不想使用多数的标准字体：

```
xset fp /usr/local/xfonts, /usr/lib/x11/fonts/75dpi
```

注意 fp之前并无一短横线(-) ，是fp而非 -fp（-fp 的意义不同，见下述）。

• 当你想重新设定服务程序对字体路径的缺省值时，使用命令：

```
xset fp default
```

• 告诉服务程序重新读入字体的目录，使用命令：

```
xset fp rehash
```

它告诉服务程序你可能已经改变了字体目录的内容而和它必须重读字体数据库，现在新加入的字体可以开始存取了。

• 在现存的路径加入新的字体目录，使用命令：

```
xset +fp dirlist
```

• 在现存路径的左面加入一行由逗号分隔的目录列表，而

```
xset fp+ dirlist
```

则将目录列表加到路径的右面。

• 将字体目录自路径移去：下两个命令行

```
xset -fp dirlist
xset fp- dirlist
```

均可将在dirlist 中的目录自现有路径中移去。

注意 字体路径由服务程序所掌握，而被所有使用该服务程序的客户程序所应用。

字体路径的次序是重要的，我们曾经提过字体设定可以对应一个或多个字体，服务程序会自行选择，但如果对应的字体是在不同的目录中，则服务程序会选择在路径中较早出现者。

你可以利用这个原则来安排最适合你的显示器分辨率的字体。假设你的显示器分辨率为100dpi，则将100dpi字体设在75dpi 之前，例如：

```
xset fp /usr/lib/x11/fonts/100dpi/,\
/usr/lib/x11/fonts/75dpi/
```

如果你指定字体为

```
* -times-bold-r-*-120-*
```

虽然字体有75dpi 和100dpi两种版本，但你会用到100dpi的字体，这正是你所需要的。

29.1.5 例子：在你的服务程序中增加新字体

现在我们将说明如何在你的服务程序中增加一个新的字体的完整样例。为了真实起见，我们以Sun所提供的字体为例，将它转换至BDF ，然后安装它，字体开始时在以下目录：

```
/usr/lib/fonts/fixedwidthfonts/screen.r.7
```

欲将Sun 的字体转换成BDF ，我们需使用contrib 版的软件程序vtobdf（其他系统也有类似的工具）。vtobdf有两个参数，分别是输入文件文件名和欲建立的 BDF 文件文件名，我们可以事先从contrib 磁带中取得此程序，编译它，而后加入到我们可执行的目录中，我们就可以使用它了，我们将或多或少依据 X的标准来命名这个新字体，我们喜欢把输出文件的文件尾名用.bdf，但由于vtobdf会在字型名后自动产生.bdf，所以可以省略它，但以后要重定名称时，则不可省略。

```
venus% cd/tmp
venus% vtobdf /usr/lib/fonts/fixedwidthfonts/screen.r.7\
  -sun-screen--r-normal---70-75-75-m---
```

现在重新命名文件，并将其搬入字体目录：

```
venus% mv- -sun-screen--r-normal---70-75-75-m---\
/usr/lib/x11/fonts/misc/-sun-screen--r-normal---70-75-75-m---.bdf
```

最后，执行mkfontdir 和告诉服务程序重新读入字体目录以便能使用此字体：

```
venus% mkfontdir
venus% xset fp rehash
```

检查一下此字体是否真的可用：

```
venus% xlsfonts "*-sun_screen*"\
-sun-screen--r-normal---70-75-75-m---
```

注意 你的字体可能可以替换其他的缺省字体，但这些字体文件可能因有保护而无法更改，必须问一下你的系统管理员。

29.1.6 使用X的颜色

X允许用日常常用的彩色名，在本节我们描述一些其他指定颜色的方法、解释命令结构如何工作和如何设定一些自己拥有的颜色名。

1. RGB颜色设定

换一种指定颜色的方式，你可以用RGB (Red (红)、Green (绿)、Blue (蓝))三基色来指定，设定形式为

```
#<r><g><b>
```

必须合乎以下的原则：

- 设定必须以井字号(#) 开头。
- 基色需依照红、绿、蓝的次序依序设定。
- 三基色均必须指定。
- 每一个基色为十六进位，共占1～4个字节。因此ffff代表颜色的最大强度，0000代表没有该颜色，例如：

```
#0000ffff0000
```

是最亮的绿色，红色和蓝色一点都没有，同样地，

#000000000000 黑色（什么颜色都没有）
#ffff0000ffff 紫色（全部为红色加蓝色）
#ffffffffffff 白色（全部的颜色）

注意 #rgb和#rrrgggbbb代表的颜色强度是相同的，但后者较亮一些。

- 每一个基色可由1～4个字节代表，但每个基色的位数则相同（例如你不可以用#rrbbbbgg）。

你可以在设定颜色时直接使用颜色名称，例如：

```
xclock -fg #3d7585 -background pink
```

颜色设定的形式往往和你的显示器非常相关，通常没有什么可移植性。

2. X颜色数据库

为了克服#rgb颜色设定不可移植的缺点，而且使系统更易于使用，X使用一个保存颜色名及其相关的rgb 值的数据库。

除非你的系统在设置之后做了明显的改变，应该会有一个 /usr/lib/x11/rgb.txt的文字文件说明数据库的内容。这个文件的前数行类似于：

112 219 147 aquamarine（绿玉色、碧绿色）
50 204 153 medium aquamarine （中度碧绿色）
0 0 0 black （黑色）
0 0 255 blue （蓝色）
95 159 159 cadet blue（学生蓝）

每一行前三个数字表示rgb 的基色值，但这里数值是10进位的，且只为0～255 ， 255 代表

颜色最大强度，第四个部分为颜色名称，允许名称中间有空格。

你可以用程序$TOP/rgb/rgb将此文字文件转换为内部的形式，当你的X系统建立时，并不会设置它。所以，要在你的数据库中加入一个新的颜色，先用文字编辑器将颜色输入rgb.txt文件，然后用以下命令：

```
venus% cd usr/lib/x11
venus% $TOP/rgb/rgb < rgb.txt
```

事实上，rgb并不需要每次均重建内部数据库，只需加入新增（或修改）的项目即可，所以你可以用标准输入来输入颜色：

```
venus% $TOP/rgb/rgb
255  50  50  mypink
…
```

因为没有任何标准的工具程序可以查询内部数据库的内容，因此上面的做法会造成rgb.txt和内部的数据库不一致，所以还是以修改rgb.txt的方式为佳。

29.2 定义和使用图形

一个图的显现是由像素组成的，而像素又对应一个位，当位为“ 1 ”时，像素为“黑色”，而当位为“ 0 ”时，像素为“白色”。X有许多的实用程序来管理图形，你可以用不同的方法来建立、编辑和保存它们。有一些用户程序允许你直接使用它们。其他大部分的程序则以内部的形式使用它们，这些实用程序大都放在X程序库中，使得用户写程序时很容易运用。

29.2.1 系统图形程序库

图形文件的程序库是系统的一部分，缺省保存在下面这个目录中：

```
/usr/include/x11/bitmaps
```

在你的工作站上或许不同，但我们将以此目录为准，并用其中的一些文件作为本节的例子。

29.2.2 交互编辑图形——bitmap

bitmap程序是一个让你以交互式建立或编辑图形的工具，它将位映像以方格子来表示，每一个格子代表一个像素，你可以用鼠标设定或清除像素。

1. 启动bitmap

通过bitmap你可以编辑一个包含一个图形的文件，或从头开始建立一个图形并将它保存为文件。不论是哪一种情况，当你启动bitmap时，你需要给一个文件名，不论是现存的文件或是新建的文件，都要为文件命名。

当建立一个新的图形时，你可以选择性地指定大小，如果你未指定，缺省大小为16×16。举例来说，假如我们想要建立一个比较大一点的十字体数位图像，我们可以用下面的命令行：

```
bitmap big-cross 40×50 &
```

2. 使用bitmap

假如我们要编辑一个现存的文件，可以用下面的命令行启动程序：

```
bitmap /usr/include/×11/bitmaps/cntr-ptr
```

则会有一个窗口出现在屏幕上，右下角以实际大小显示出目前图形的状态，另一个则为相反的图形，其他在右边的“框”你可以用按鼠标按钮的方式来操作它们。

用三钮鼠标编辑图形最简单的方法如下：

- 设定像素：在一个像素上按鼠标左按钮，或者按住左按钮并拖动它，每一个经过的像素方格均会被设定，直到释放按钮为止。
- 清除像素：和上述相同的方法，但是要用鼠标右按钮。
- 反转像素：在一个像素上按鼠标中按钮（也就是黑的像素被清除而白的像素被设定），当你按住中按钮并拖动，所经过的像素格均会反转。

bitmap还有其他的角度：如果你观察接近箭头的上端部分，你可以在其中的一个方格中看到有一个小菱形，这代表了热点，当bitmap用来建造一个光标时会应用到：热点是光标真正动作的点。指向型的光标，热点通常在顶端，而圆形或方形的光标，热点则在中心（你可以用Set Hot Spot和Clear Hot Spot两个命令来更改热点的位置或消去它。）

当你结束了更改动作，可以按Write Output将图形保存，但不会离开bitmap程序。

离开程序，按Quit按钮，如果你编辑了图形却试图在未保存前离开程序，你将会得到提示，以确定你是否真要这样做。

3. 画图

bitmap的画图功能如下：

- 画一条线：按Line，光标会变成一个大黑点，在所欲画的线的一端按一下按钮，而后在另一端也按一下，bitmap会画出这条线。
- 画一个中空的圆：按Circle按钮，同样，光标会变成一个大黑点，在你所欲画圆的圆心按一下，而后在所欲画圆的圆周上的任一点按一下， bitmap将画出这个圆的圆周。
- 画一个填满的圆：按Filled Circle按钮 ，其余同上。

4. 在长方形区域内工作

命令Clear Area、Set Area和Invert Area 必须在长方形区域下操作，长方形区域的决定方式是你在它的左上角以按住鼠标任意按钮的方式指定，然后拖动到右下角，当你拖动时，目前被指定的区域会以高亮度显示。

你可以拷贝、移动或重叠一个区域。你以拖动的方式指定原始区域，而后在目标区域上的左上角按按钮，各种命令的动作如下：

- 拷贝：目标区域会被消除，而所有对应于原始区域为黑像素的均会被设定。
- 移动：原始区域和目标区域均被清除，目标区域对应于原始区域为黑像素的均会被设定。
- 重叠 ：在目标区域中对应于原始区域被设定的像素均会被设定，其他没有改变。

5.图形的文件格式

一个图形会如同ASCII 文字一样保存到文件中，其格式类似C 语言程序。

例如：文件 /usr/include/X11/bitmaps/cntr_ptr 的内容：

```
#define cntr_ptr_width  16
#define cntr_ptr_height 16
#define cntr_ptr_x_hot  7
#define cntr_ptr_y_hot  1
static char cntr_ptr_bits[]=
0x00, 0x00, 0x80, 0x01, 0x80, 0x01,\
0xc0, 0x03, 0xc0, 0x03, 0xe0, 0x07,
0xe0, 0x07, 0xf0, 0x0f, 0xf0, 0x0f,\
0x98, 0x19, 0x88, 0x11, 0x80, 0x01,
0x80, 0x01, 0x80, 0x01, 0x80, 0x01,\
0x00, 0x00;
```

带有_x_hot和_y_hot的变量仅在热点被指定后才会包含进来。

更多的细节包含在bitmap(1) 的联机帮助中，不过无论如何，你不需要直接以此种格式处理图形，任何你想要做的事均有工具程序来处理。

29.2.3 编辑图形的其他方法

bitmap程序对于一个小的图形工作起来算是相当实用的，但它有一些缺点：

• 它不接受较简单格式的输入文件，例如像一些由扫描现存图形所产生的文件。

• 它必须以交互方式执行，对一些程序性的编辑动作并不实用。

• 你可能希望用它产生一些图形来显示，但它无法在非X系统上执行。

要克服上述的问题，需要以字符图片的形式来建立位映像，并提供这个格式和 bitmap的格式相互转换的程序。字符图格式是非常明显的：每一行的像素用一行的字符来表示，黑的像素用一个指定的字符（缺省为# ），而白的像素用另一个字符（缺省为 -）来表示。

你能以文本编辑器或任何其他系统上任何其他合适的程序编辑这些图形，也可以由扫描仪或其他图像设备产生。

X提供了两个程序做字符图格式和图形格式间的转换：

• atobm : 转换一个字符图为标准的图形。

• bmtoa : 转换一个标准的图形为字符图。

两个程序均允许你指定任何字符来代表黑和白像素。

29.2.4 定制根窗口——xsetroot

xsetroot 让你设定你的根窗口的特征，你可以改变窗口背景的颜色和图案，以及窗口所使用的光标。

1. 设定背景的位图

你可以指定任何图形来当作屏幕的背景（只要它是 X的标准格式），在 xsetroot 的命令行上，-bitmap 选项跟随着图形的文件名。例如：

```
xsetroot -bitmap /usr/include/X11/bitmaps/mensetmanus
```

会出现一个精致的背景图 。

2. 设定背景光标

如果你不要使用缺省的大的“ X ”光标，你可以用选项 -CURSOR 加上cursorbitmap 和maskbitmap两个参数来改变它，两个参数均为图形文件名。例如：设定光标为前节所示的图形，使用命令：

```
xsetroot -cursor /usr/include/X11/bitmaps/cntr_ptr\
/usr/include/X11/bitmaps/cntr_ptrmsk
```

maskbitmap决定了cursorbitmap的哪些像素真正被显示出来，光标像素中只有对应到遮盖像素为黑的部分才会用到，而不会显示光标其他的像素。总的来说，遮盖决定了光标的外形，反之，光标图形则决定了外形的颜色。遮盖和光标的图形必须大小相同。

这种遮盖结构在两种情况下非常有用：

1) 它允许 “ 干净地 ”显示出非长方形光标，而不需显示出多余的空白。例如如果没有遮盖，cntr_ptr会显示成一个 16×16白方形中有一个箭头，当你用它指对象时，对象的一部分会被矩形外框遮盖住。

2) 适当地设定遮盖，你可以保证不论背景的颜色为何均能看得到光标。例如 cntr_ptrmsk比 cntr_ptr 的边均大一个元素，所以光标周围围绕着一圈白边。如果遮盖和光标大小相同的话，

当光标在黑色的区域将会消失不见。

你可以让遮盖和光标使用相同的图形：光标的外形会如你所期望（因为遮盖决定外形，而这外形正是你想要的），它们可以工作，但是当光标进入和它相同颜色的区域时，你就很难看到光标了。实际上，并非所有在 /usr/include/X11/bitmaps中的图形均有相对应的遮盖，如果你使用它们当作光标，你必须使用光标图形当作遮盖。

有兴趣的话，试一试把mensetmanus 当作光标和遮盖（热点是在左上角）。

3. 其他背景设定选项

你可以用命令行选项 -solid colour设定背景为单一颜色（在单色显示器上只有黑色和白色）。你可以用 -grey或 -gray设定颜色的灰度，你也可以用 -mod x y 设定格子图案，x 和 y为 1到16的整数。

4. 重定缺省的背景和光标

如果你不喜欢既有的设定，你可以用下列两者之一恢复缺省的光标和背景：

```
xsetroot -def
xsetroot
```

29.3 定义应用程序的缺省选项——Resources

大多数X程序接受命令行选项，以便让你指定前景和背景的颜色、字体、起始位置等等。这种需求是有必要的，因为如果你在程序内硬性规定使用某种字体，而在执行此程序的机器上并没有这种字体，则将使得程序无法执行，所以你不应硬性规定某些参数。

每次执行程序时不太可能在命令行中指定所有需要的选项，因为有太多种可能的组合了，所以X提供了一个叫做“资源”的一般性结构，用来传递缺省的设定给应用程序。

你在系统中几乎所有的定制动作都将运用到“资源”，事实上你为一个应用程序所选择的每一个选项的设定都要用到“资源”，从简单的项目(例如颜色或字体)，到定制键盘或管理显示器如何工作。“资源”非常实用，而且在系统中到处都用得到。

29.3.1 什么是资源

在X的文献中，“资源”有两种意义。第一种是相当低阶的，意指被服务程序管理或建立而被应用程序使用的东西。窗口、光标、字体等均属于这种资源。

另一种是通常在联机帮助中常看到的“资源”的意义：它是一种传递缺省设定、参数和其他值给应用程序的方法。在本节中，我们局限于讨论这种意义的“资源”。在解释现行系统如何工作前，先回顾一下X的早期版本是如何掌握这些功能的，因为现行的结构由此产生。

1. “缺省”的背景

在X较早的版本，对于像窗口背景颜色、窗口边界的颜色、应用程序所使用的字体这类项目你可以轻易地设定其缺省值。

缺省值的设定方式很直接，你只需指定一个窗口的属性和它的缺省值。例如：

```
.Border : red
```

意即所有的窗口均为红色的边（除非你在命令行中重新设定边的颜色），你也可以把程序的名称放在属性之前，则只有被指名的程序才会改变，所以把以下这个规范

```
xclock.Border：blue
```

和先前的规范结合在一起的意义为：缺省时所有窗口均为红色的边，只有 xclock的窗口为蓝色的边。

每次设定缺省值，程序会自动取用该值，所以你不需每次均指定你的选择，它让你依照适合你的工作习惯来使用字体，不论你要用较小的字体以获得更多的信息显示，或用较大的字体以便阅读，它让你为特定的应用程序选择颜色，你可以定义应用程序的起始位置，所以你可以自行设计一些启动屏幕的布置，因为许多缺省值（字体、颜色等）实际上精确的意义为“资源”，所以“资源”的意义逐渐扩增为“缺省值设定”或“设定缺省选项”。

2. “资源”向应用程序发送信息

随着X的开发，应用程序也随之扩增，需要有一个设施传递大量的信息到应用程序中，以定制或指定它们的行为，而不再只是有关颜色和字体的信息而已，例如你可以告诉 xbiff检查邮件的频率，或定义 xterm的功能键12为插入某一特定的字符串，或在 xedit中连续按两次鼠标中按钮代表选择目前这段文本等等。

所以逐渐地开发了“资源”及缺省设施。到目前对于向一个应用程序传递任何信息已是一个一般性的结构。你可以像文字字符串一样地指定信息，应用程序会在内部解释它：例如把这字符串当作一个鼠标按钮的名称，或一种颜色，或由应用程序所发出的一个功能和资源是如何指定的。

这个结构也逐渐地复杂起来，以便让你能正确地指定在何处应用缺省值。在以前，你只能指定所有的程序或某一个特定的程序。现在的系统你可以设定的缺省值如：“终端机窗口的菜单选项”或“在所有窗口的标签”或甚至“除了 xterm以外的所有编辑器窗口按钮框中的功能按钮”。

X Toolkit使得资源在使用上有很大的兼容性并且增进了应用的精确度。你需要先了解Toolkit是什么，才能适当地使用资源结构。

29.3.2 XToolkit

我们先前曾提过，X并不决定用户界面，它只是提供一些结构，让应用程序设计者能组合成任何形式的界面。理论上这是非常合乎需求的，它使得系统拥有一个一般性目的的工具且没有使用上的限制，但从另外的角度来看，它有很大的缺点：

- 对一个用户而言，不同的应用程序有不同的界面，不只是难学难记，且应用程序无法顺利地相互协调工作（例如无法在窗口之间做剪贴），你得到的是一组个别的、独立的程序，而不是一个一致的、合作无间的系统。
- 从程序员的立场来看，意味着基本窗口系统上的每一件事均需从头做起，菜单、滚动条、时钟、功能钮等等都必须一一生产。甚至在单一的产品中，不同的程序员做了一点稍有不同的事，便会导致许多不相容的情况。

为了克服上述的问题，工具箱的方式应运而生。在某些范围，工具箱会决定用户界面的形式。但是无论如何，它会尽量减少这种影响，并让用户界面开发者有更多选择的可能性，工具箱被分为两个部分：

- 一组基本的结构和函数用以配置用户界面的元素称为工具箱内部工具。不论是什么样的界面，任何工具均需使用到它，所以我们可以把工具箱内部工具视为“固定的”，也就是无可替代的。
- 一组提供特定的用户界面（或界面的形式）的元素，这些元素称为 Widget，而工具箱的第二部分称为Widget组，我们认为这是可以替换的，不同的界面提供不同的 Widget组，甚至它们都使用内部工具。

1. 工具箱的第一个部分 —— 内部工具

内部工具定义的实体称为Widget，并提供了所有建立、管理和毁坏Widget所需的设施。从理论上来说，Widget是一个处理特定动作的用户界面的元素，实际上一个 Widget是X窗口加上规则和功能以决定它的输入和输出的动作，也就是说，它如何对用户有所反应。

为了帮助解释Widget的观念，我们将给一个样例。但是请注意。它们并不是内部工具的一部分，而是我们将于下一小节讨论的一个特定的 Widget组的一部分，在此提出的目的只是为了实用。

- Command Widget（命令Widget）：这是一个在屏幕上含有一些文字的长方形“ 按钮 ”（也就是一个小窗口）。当鼠标指针在这个按钮之上时，它的边会呈现高亮度显示，当一个鼠标按钮在这个Widget被按时，一个被程序员指定的软件过程便会被执行。

你已经使用过命令Widget好几次了：主要在xedit 的命令菜单和在xman的主选项窗口中。

- Scrollbar Widget（滚动条Widget）：同样，你也已经在 xterm 和xedit 中滚动文本，在xman中滚动文本和目录列表中使用过了好几次。
- 内部工具提供了基本的结构，任何一个提供界面的较高级软件均需要使用到它，它提供了以下的功能：
- 建立和毁坏Widget。
- 把一组Widget当成一个单元来管理。
- 掌握位置，包括从最高级（也就是应用程序的 -geometry 选项）到最低阶（管理应用程序用到的sub-Widget的位置（例如菜单按钮的位置））。
- 掌握“ 事件 ”，例如在一个Widget中，当鼠标按钮按时呼叫适当的程序、管理窗口的公布和掌握键盘的输入。
- 管理给资源和缺省的每一个Widget。

2. 工具箱的第二部分 —— Widget集

广义来说，内部工具只提供你建造一个用户界面的骨架， Widget集则实际地提供了一个既定的界面，且不同的 Widget组提供不同形式的界面，虽然对任何范围的需求并无法预防混用Widget，但一般仍希望一个系统固定在一致性的Widget集上。

在X core 版上只有一个Widget集提供，我们描述于下：

Athena Widget Set （雅典娜Widget集）

大部分的MIT core版的应用程序使用工具箱和 Athena Widget Set（名称的来源是X由MIT的Athena计划产生出来的）。这些Widget的定义你已在许多应用程序中用过。

我们在前节提过了Command Widget和Scrollbar Widget，至于Athena Widget Set 的其他部分包含：

- Label Widget：这个你可以想象，在窗口中显示的一个字符串或图（例如在 xman主选项菜单中的Manual Browser的标题。）
- Text Widget ：它提供我们所使用的编辑功能。
- Viewport Widget ：一个具有滚动条的窗口，让你可以滚动窗口的内容， xman使用其中之一用以显示联机帮助的目录。
- Box Widget：它以一个指定大小的框管理sub-Widget的布置，且试着将 sub-Widget尽量集中在一起，例如xmh 的Reply 、Forward 等命令钮即是由Box Widget布置。
- VPaned Widget ：它管理sub-Widget，将它们保存在垂直堆栈中，且显示了在两个 sub-Widget之间的分隔线上的“ 把手 ”(grip)，把手可以选择性地让你改变一个Widget的大小，而且另一个相关的Widget大小亦伴随变化，例如我们在看到的 xman窗口的主要元素是由

VPaned Widget 管理的。

- Form Widget ：另一种管理一组sub-Widget的方法，但对位置的选择有更多的灵活性。
- List Widget ：它管理一组字符串，将它们安排在行列中，任何字符串可以通过它来选择动作：字符串会转为高亮度显示，且呼叫一个指定的函数以完成特定的动作。 xman使用一个List Widget来管理它在有关联机帮助中的列表。

我们现在来看一下如何组合Widget以获得所需要功能，我们仍然以 xman为例。xman的联机帮助的目录在它的低阶是 list Widget ，管理目录页名称的列表以及它本身的内容是用viewport Widget（让用户滚动至列表中所需的位置），将Widget聚集在一起，它们是包含在一个VPaned Widget 中，所以事实上这是一个层次式的 Widget，每一个可以完成它的专门功能，而所有的应用程序所使用的工具箱均含有这三个 Widget结构。

3. Widget：名称和类

“ 资源 ”和缺省结构是在Widget名称的基础下工作的，所以我们将介绍对名称的处理。

工具箱提供给程序员一个面向对象的编程系统。它定义对象的类，也就是指定何时对象被建立或如何操作等等的对象属性。这些对象即是 Widget，系统将确保它们和其他的Widget以及其他部分的应用软件以定义明确的方式交互。

当一个程序员建立一个特定类的Widget，它被称为该类的实例（概括地说，一个类是一个抽象的定义，而一个实例在某些地方实际上符合这些定义）。建立Widget必须有一个名称，由程序员指定，例如：程序代码的实际型号为 Creat a Widget, of the class Box Widget,and call it topBox 。在某些环境下Widget的类名也会引用到。总之，一个 Widget有一个实例名称和类名称；更简单地说，有一个名称和一个类。

29.3.3 管理资源 —— 资源管理器

用“ 资源 ”来做什么？用“ 资源 ”能传递信息给一个应用程序，告诉它以某些方式改变它的一般性动作，例如，将窗口的边以粉红色替换原来的黑色，或使用一些特别的字体。

例如，你设定一个包含许多项资源规范的数据库，每一个资源规范以一个应用程序的某些特征命名，且设定一个值给这个特征当缺省值，也就是说，一个规范 (spec) 的形式为

```
characteristic : value（特征：值）
```

当应用程序开始执行时，它会先询问数据库是否有任何特征符合自己所要的设定，或使用相关的值，例如：

```
xclock*foreground:blue
```

意为将值blue设定给特征xclock*foreground 。用以决定一个程序的需求是否符合在数据库中规范的系统部分，被称作“ 资源管理器 ”。

资源缺省值能应用到一个应用程序中的对象（通常是 Widget），就如同设计整个程序一般（例如，你可以对一个特定的子窗口在某一个命令按钮的背景色设定缺省值，而不是只能针对所有应用程序的窗口背景。）为了能达到这一点，我们需要一些严谨的命名方法，以设定对象应用缺省值。

1. 指定资源预定应用到何处

资源管理器根据特征值决定一个缺省规范是否能应用在特别的情况下，我们可将特征值分为三个部分。

1) 你用以设定缺省值的程序属性，例如：背景色、字体等。

你必须指定属性，也就是说你必须设定一个值。给定一个资源的规范而不说明它的值是无

意义的。

注意 在X的文献和联机帮助中，属性通常被称为“资源”或“资源名”，“Resource”通常也被用来当作特征值。

特征值的其他两个部分指定缺省值在何处使用。例如只在特定的程序或特定类型的对象，使用。

2) 应用到这个规范的应用程序的名称，如果你省略它，规范将应用到所有的应用程序中。

3) 一连串的限定条件：当对象符合限定条件时，才会产生指定的应用。限定通常为 Widget 的名称，你可以指定从零开始的任何数目的限定。例如：

```
xclock*foreground:blue
xedit*row1*Command*Cursor:Cntr_ptr
```

第一个例子没有任何限定，第二个例子有两个限定(row1 和Command)。

三个部分依序排列：

```
[<program name>] [<restrictions>] <attribute>
```

并以特殊的分隔符分开，我们将于稍后说明分隔符的细节，但我们先看一些特征值的例子（为了简单起见，我们在例子中只用到颜色属性）。

• 指定在任何地方中的前景色缺省值为黄色。

```
*foreground:yellow
```

我们未指定任何应用程序的名称，所以此规范可应用到所有的应用程序；我们也未指定任何限制，所以对一个应用程序在任何地方都适用。（“*”这个符号就是我们方才提及的特殊分隔符的一种）

• 指定只有在xclock应用程序中的前景色缺省值为粉红色。

```
xclock*foreground:pink
```

这个规范仅能在xclock中适用，但是只要项目中的属性叫做foreground的均适用。

• 针对一个特定应用程序的特定地方：

```
xman*topBox*foreground:blue
```

这个规范仅能在xman适用，而且只能在xman主选项菜单中名为topBox的对象适用（应该适用于xman中所有叫topBox的对象，但实际上只有一个topBox对象）。

• 在第二个例子（粉红色）中我们包含了应用程序名称，但忽略了限制条件，现在我们反过来：

```
*command*foreground:green
```

也就是说，我们指定在任何应用程序中对象名称为 command 的前景色预设值为绿色。

2. 用类名称使得规范说明一般化

前述的例子说明了我们对缺省值结构所需的大部分功能，但它们有一个限制：你必须知道应用程序员设计在每一个应用程序中的 Widget名称，这些信息有时包含在程序的联机帮助中的一部分，但通常被省略。

无论如何，资源管理器有一个尽量减低这个问题的方式：当你在特征中不论何处用到一个应用程序名称、限制或属性名称，你均可用类名称来代替它。

• 应用程序类名称：描述程序的类型，例如xterm 可以是Term Emul （终端机模拟器）的类，xedit 和emacs是Editor（编辑器）的类（但如果xterm 是xterm 的类、xedit是xedit 的类则失去意义）。

• 限定类名称 ：限定几乎是一定不变的Widget名称，所以在此地你可以用Widget类名称。

• 属性类名称 ：属性是如同Widget一般的一个类型或类的实例。

传统上，所有的类名称以一个大写的字母开头，其后则为小写字母，例如属性“foreground”是属于“Foreground”类。我们将简单地解释你如何去发现你需要用来指定项目的类名称。首先，我们将看一些更多的样例，这次用到了类或一个混合了类和实例的例子。

(1) 含有类名称的资源规范说明例子

这些例子演示出你如何在资源规范中使用类，而较前述以更一般性的方式设定缺省值，它们也解释了你如何能使用一个类来设定一个缺省值给较大范围的情况，和将类与实例结合起来以拒绝缺省值在某些特殊情况下的设定。

• 指定在任何地方前景色的缺省值为黄色

```
*Foreground:yellow
```

这个例子和先前例子的区别在于我们是对Foreground类指定缺省值。这个区别之所以重要，是因为并非所有在类Foreground的属性，它的实例名称都是叫foreground。例如，xclock的指针的颜色可由类Foreground的属性来决定，但它的实例名称不叫foreground而叫hand。

• 我们可以用这种结构来帮助我们在文件不清楚的情况下，借助于以强烈对比的组合设定缺省值来分辨对象的类：

```
xmh*Command*Foreground:khaki（土黄色）
xmh*Command*Background:maroon （栗色）
```

这样将使所有的命令Widget(command Widget)呈现醒目而美丽的颜色。

• 对所有的文本Widget(text Widget) 窗口设定一个缺省值，除了xedit 窗口以外：

```
*Text*Background:pink
xedit*Text*Background:navy
```

• 和上例原理相同：

```
*Command*Backgrond:green
xman*Command*Backgrond:white
xman*manualBrowser*Command*Background:orange
```

(2) 如何发现实例和类名称

这很困难，因为没有简单和一致的Widget名称、类、属性等等的文件，我们只能列出每一个最好的来源，并且提示你如何获得更多的信息。

应用程序实例名称：是你所执行的应用程序名称。如果此程序使用工具箱，你能在命令行以选项-name string明确地指定一个不同的应用程序名称，为何需这样做？因为它让你在单一应用程序中定义超过一组的缺省值，而你可以使用 -name 在其间切换。例如，你可以定义一个xterm 的正常缺省值，但对名为demo的应用程序定义一个很大的窗口尺寸和大尺寸的字体，你可以用

```
xterm -name demo
```

(3) 用来演示或教学的xterm

• 应用程序类名称：这没有文件说明，最简单找寻它的方法是启动应用程序并在窗口中使用xprop ，属性当中的WM_CLASS会给你应用程序实例及类的名称，例如，对 xterm 你会得到

```
WM_CLASS(STRING) = "xterm","XTerm"
```

• Restriction/Object/Widget 实例名称：程序的联机帮助会列出你最想要存取的对象名称，例如：xman列出topBox、help、manualBrowser 等等，如果联机帮助并未给你实例名称，则唯一的方法是如果可能，直接看它们的原始程序代码（这种方法通常无法令人满意）。

• Restriction/Object/Widget 类名称：这容易些，大部分的联机帮助会告诉你想知道的对象类，即使没有的话，大部分的对象也是标准集合中的 Widget，当你从系统中使用它们时，

你通常能猜出它们属于哪一个类（例如： scrollbar 的实例名称的99.9% 是类Scrollbar的Widget）。

- 属性名称和类：大多数的联机帮助会列出名称，通常也会有类， xclock的联机帮助便是非常清楚的样例。

无论如何，利用工具箱写的程序通常使用标准的 Widget，它的属性并不会在联机帮助中列出，但通常由一组全部或部分的属性组成，要找到这些属性，你必须在工具箱文件中寻找：

- “ X 工具箱内部工具 ”联机帮助中的附录E 列出所有标准的“ 资源 ”（也就是属性）的名称和类。实例名称项目看起来类似：

```
#define XtNborderWidth "borderWidth "
```

所有的名称均以XtN 开头，跟随其后的名称则以小写字母开头，而类别的名称则以 XtC 开头，类的项目看起来像

```
#define XtCBorderWidth "BorderWidth"
```

在双引号中的便是名称，也就是说， borderWidth 是实例名称， BorderWidth 是类名称。

- 查看“ X 工具箱Athena Widgets ”联机帮助的2.3 节可看到被所有Widget使用到的资源名单，包括名称、类型、缺省值和一段文字叙述。名称的定法如上所述，也就是以 XtN 开头， XtN 之后则为属性名称。
- 查看“ X 工具箱Athena Widgets ”联机帮助中对Widget的描述，每一个会列出它所使用的“ 资源 ”，和上述相同。
- 如果以上均行不通时，你可以查看 Widget的源代码、资源可用到的部分列在 XtResource 数据中。例如， Athena Scrollbar Widget的程序内包含

```
static XtResource 资源[]=
{XtNwidth, XtCWidth,..）},
XtNheight, XtCHeight,..）}。
```

3. 资源规范分隔符

你可以用星号 (*)或点号 (.)来分隔资源规范的元素，星号比较通用一些，它让你指定那些符合范围的案例的特征。我们看到

```
xclock*foreground:pink
```

用来指定xclock中任何东西均使用 foreground属性，所以在此样例中可以看出，星号具有通用字符的效果，甚至可以再一般化一点：

```
*Foreground:yellow
```

它将适合任何应用程序。而点号只是分隔组件，它表示每个组件都必须一一对应，例如：

```
xman.Manual Browser.Help.background:black
```

并不适用于命令按钮或含有 xman的窗口的不同Widget。在我们对这两种分隔符做更精确描述前，我们需要更详细地看一下资源管理器的操作。

4. 资源管理器如何运作

前面一节，我们曾说过一个应用程序会查询资源缺省规范的数据库看是否符合，现在我们描述查询如何掌握这些规范。

应用程序中的个别对象（通常是 Widget）使用资源，而对象则在应用程序上端hierarchicallyixWidget + hierarchy安排，然后可能由一个Widget管理其他的Widget配置，例如文本窗口、命令菜单等等。

对每一个对象，应用程序欲查询资源数据库时，它必须把对象的实例全名和类全名传递给资源管理器，并传递对象所用的一组属性和类名称的一组属性，例如对 SAVE按钮，应用程序

指定：

```
full instance namexedit.vpaned.row1.Save
full class name   Xedit.VPaned.Box.Command
attribute instance-names borderWidth,cursor,font,label,...
attribute class-namesBorderWidth,Cursor,Font,Label,...
```

而后，资源管理器检查每一个在数据库中的规范，看它是否和应用程序所传来的属性和对象名称相符。如果相符，在数据库中规范值的部分会传回应用程序。

在这种相符的操作中，星号和句号的区别非常重要。简单来说，我们可以想到资源管理器只是以单字为基准来对应文字字符串，句号正是每一个单字的分隔符，星号也是分隔符，但不同的是它可以通用字符的方式代表从零到任意数目的单字，对于对应唯一的限制是在数据库中规范的属性必须对应查询应用程序所传来的属性，你不可对属性用通用字符。

现在你可以看到不同的规范如何工作：

```
*foreground:yellow
```

可以应用于任何应用程序中的任何对象。因为星号对应所有的应用程序和所有的限定和对象名称。

```
*Command.Foreground:violet
```

应用于任何应用程序中任何Command 类型中Foreground类的任何属性。

```
xedit.vpaned.row1.Help.background:navy
```

是一个完整的规范但是将只影响到命名当中的对象名称的属性（本例中，尽管事实上是大写的，Help是一个实例名称，它的类是Command）。

除非你有一些非常特别的需求，最好不要用句点当分隔符，尽量以星号代替，这样可减少错误发生的可能，而且在重写应用程序时，比较不会受到层次结构改变的影响。

5. 多种资源规范对应的优先规则

我们现在有一个非常灵活性的方法来指定应用程序的资源，但正因为它太灵活，以致当一个应用程序查询资源数据库时常常有数种规范与之对应，如何解决呢？

简单地说，如果同时有超过一个规范对应，则使用最具体的一个，资源管理器有一组优先规则用来决定是否一个规范较另一个具体。

- 使用句号作为分隔符较使用星号更为具体，例如：*Command.Foreground 较*Command*Foreground 更为具体。
- 实例名称较类名称更具体，例如：*foreground 较*Foreground 更具体。
- 指定一个元素较省略它更具体，例如：xmh*command*foreground较xmh*foreground更具体。
- 元素靠近规范左边的星号较靠近右边的更具体，例如：xmh*foreground较 *command*foreground更具体。

这些规则相当直接，它们大部分可用另一种方法来说明：“如果一个规范对应到另一个规范而为其子集合者，则前者较后者更具体。”

6. 在工具箱程序中应用程序资源

通常一个应用程序使用资源管理器来定义程序层次中 Widget的属性缺省值，但有时需要有和Widget不直接相关的设定缺省值（或传值）的能力。

为了达到这一点，工具箱提供了一个叫做 Application Resource的设施，它和非工具箱缺省的外表原则上相同 —— 应用程序定义了它本身选择的属性。类名称也相同，所以事实上这些属性和一般常见的层次没有什么不同。

xman使用到一点这个设施，它让你能在帮助窗口中指定不同的文本文件，是否在主选择窗口中指定一个你要的窗口，或当程序启动时直接进入一个联机帮助等。

7. 资源和非工具箱应用程序

并非所有的程序均使用工具箱，但工具箱几乎掌握了所有对一个应用程序的资源管理，特别是应用程序的Widget结构定义了对象和子对象的层次，并能适当地查询资源管理器。但是非工具箱应用程序要如何使用资源管理器？

答案是应用程序只须明确地查询每一个它有兴趣的属性。稍早我们曾说过资源管理器对资源无限制，因此应用程序能使用任何它想要的属性名称，只要程序的文件告诉用户它们在何处，它们就如同其他的应用程序一样。

xcalc 应用程序是一个不使用工具箱的程序样例，它也利用上述方式掌握资源规范。

有几点需要注意：

• 此种类型的缺省值没有类。

• 程序以类似类名称（也就是说，第一个字母大写）来定义属性，例如 xcalc 使用 Background、Foreground、BorderWidth 等等。

• 如果大小写错误，你的规范不会工作，例如：规范

```
xcalc.foreground:green
```

会被xcalc 忽略。

• 即使这个程序定义的属性并非层次的一部分，你仍能使用星号当分隔符，例如：

```
xcalc*Foreground:orange
```

29.3.4 资源的类型 —— 如何指定值

直到现在，我们仍然只看资源规范的“ 左半边 ”，而忽略了值的部分。现在，我们来看一看“ 右半边 ”（值的部分）。

简单地说，值只是一个传递到应用程序的文本字符串和资源管理器完全相关，之后，应用程序以此值做它所要做的事。当然，在实际的操作上，应用程序必须明确地做某些事，而工具箱的确也掌握了大多数这一部分的工作，所以你可获得一致的界面。

所以当我们以一个资源值传递我们所需时，实际上我们使用少数的类型，你已看过它们的大部分，你在任何地方均可以资源规范来使用它们：

• Colours （颜色）：我们已广泛地使用过它们 —— 不需要多做解释。

• Fonts （字体）：在一般的方法中我们已描述过，在资源规范中，你也可使用通用字符或全名。例如：

```
*Font: *-courier-medium-r-*-140-*
xterm*Font: 8*13
xterm*boldFont: 8*13
demo*font: *-courier-medium-r-*-240-*
demo*boldFont: *-courier-bold-r-*-240-*
```

设定一个整体性的缺省字体，但使用一个正常的 xterm 指定一个明确的一对字体，和一对被demo应用程序使用的较大的字体（可用 xterm -name demo）。

• Numeric quantities：在不同的情形中，例如：

```
xclock*update:30
xclock*update:60
BorderWidth:10
xlogo*Width:120
```

```
xterm*saveLines:200
```

• Boolean values：指定yes或no，你可以使用yes、on、true和no、off、false，例如：

```
xterm*scrollBar:false
xman*bothShown:true
```

• Cursor names：指定在/usr/include/X11/bitmaps中包含你所要的光标的文件名，例如：

```
xterm*pointer Shape:cntr_ptr
```

• Geometry spec ：全部或部分。

```
xcalc*Geometry: 180*240-0-0
xcolock*Geometry: -0+0
```

设定一个计算器的缺省尺寸及其启动位置在右下角，时钟的启动位置在右上角。

• 键盘转换(keyboard translations) ：安排特定的字符串给一个键，或安排特殊（非输出）动作给键或按钮，这相当的复杂，在下面会全面专门讨论它。

• Pixmaps ：Pixmaps 是像图形纹理(texture) 一般的图案，像位映像或光标一样地指定它们。当你在单色屏幕上工作时非常实用，一旦为不同类的 Widget设定背景，你便能看到应用程序在何处使用到它们。例如：以下的资源规范：

```
*Pixmap: mensetmanu;
List*backgroundPixmap: scales
Box*backgroundPixmap: cntr_ptr
Command*backgroundPixmap: sipb
```

使应用程序看起来很讨厌 —— 杂乱的窗口，每一个空间都以某种图案填满，但有时这样做可能会有用，backgroundPixmap是类Pixmap的属性。

29.4 实际使用资源

前一节解释X资源的规则，结构如何工作和资源规范的格式。本节继续讨论资源，但较强调实用性，我们告诉你如何及何处设定资源缺省值，来影响系统的一部分或全部。在本节结束前，我们将完成一些例子，指出常见的错误，并告诉你如何克服它们。

在这些例子中，我们假设你的工作站叫做 venus ，并且大部分时间你是使用它。从 venus 的显示器，你可在远程的机器 saturn和mars上执行客户应用程序且和 venus 共享文件系统；neptune 则不可，我们曾在上面描述过。

当你在本节中时，记得资源结构是：传递信息给应用程序，通常这些信息是用来传递一些比较感兴趣的缺省值（例如颜色和字体），但只要应用程序取得协调你就能使用这种设施传递任何信息。所以在一般状况下，我们倾向于把“ 资源规范 ”“ 缺省值 ”“ 资源 ”这三个名词视为同一含意。

29.4.1 在何处保存资源的缺省值

在上一节我们只告诉你输入资源规范到一个数据库中，但未告诉你如何做。事实上有几个不同的地方可以保存缺省值 :这些地方通常是一个你可以用任何编辑器修改的简单的文字文件，但有一个特殊的位置需要特殊的工具来设定它，我们先很快地给你一个概念，再讨论细节部分。

首先它的结构非常复杂：包含命令行选项总共有八种设定资源方法，但有两个重点需要注意：

1) 你最好只使用其中的一种或两种设置，只要你做完启动设定，你将只须改变缺省的设

定。

2) 系统掌握许多不同模式的工作，并满足那些在许多显示器上工作或在一台显示器上工作而存取远程机器的用户。

总之，这些设置让系统尽可能富于灵活性，但无论何时，你将只须存取其中的子集合即可。

1. 设定资源的八种方法

总共有八种方法设定资源，但它们可分为下面几类：

• 应用程序专用的资源：资源的表列，限定文件只能被特定的应用程序读取。

• 服务程序专用的资源：应用设定，不管应用程序在哪一种主机上执行。

• 主机专用的设定：对应用程序在主机上执行有关的设定和显示器无关。

• 命令行选项：在执行时期做一次关闭设定。

(1) 应用程序专用的资源 —— 方法1 和方法2

工具箱程序初始时在和应用程序直接相关的两个文件中寻找资源，这些文件只能被特定的应用程序读取：

1) 应用程序类资源文件：这个文件包含了机器一般性对应用程序的类的缺省值，通常为系统管理员所设定。它的名称就是应用程序类的名称，在标准安装的系统中它是保存在目录/usr/lib/X11/app-defaults 中，例如xterm 的相关文件为：

```
/usr/lib/X11/app-defaults/XTerm
```

在core版中，有一个相关于Xmh 的此种文件，观察此文件可以看所使用之设定的类型。

2) 你自己拥有的应用程序专用的资源文件：这个文件的名称和上述相同，但它存放在不同的地方 —— 由外壳变量$XAPPLRESDIR所指定的目录，如果未定义，则放在 home目录。例如对Xmh 类的程序，它的文件放在下列二者之一：

```
$XAPPLRESDIR/Xmh
$HOME/Xmh
```

你可以使用此种文件，处理方法1 中你不喜欢的文件使其无效。

(2) 服务程序专用的资源 —— 方法3 和方法4

这是对你目前工作的服务程序（显示器）做有关的设定。键盘的设定通常是服务程序专用的（因为不同的显示器有不同的键盘）。另一个服务程序专用的特征为显示器是彩色或单色。

资源和这些有关的项目会被所有与这个终端机相关的应用程序应用到，并且不论应用程序在何主机上执行。（例如，如果你使用的显示器为单色，则不管你的应用程序在何处执行，你还是不会要它使用彩色。）

保存服务程序专用设定的方法是：

3) 服务程序的RESOURCE_MANAGER属性：使用下述的xrdb程序，你可以在服务程序的根窗口的RESOURCE_MANAGER属性中保存资源设定。它的优点如下：

a) 你不需编辑任何文件即可设定缺省值。（当你为了了解系统而实验系统时特别有用）

b) 资源被服务程序掌握，所以不论应用程序在哪一部主机上执行，均能被所有的应用程序应用。在我们的样例中，在neptune 的情况下特别有用，甚至在不和我们的显示机器venus 共享文件系统时，它仍然自动地选出为了使用此显示器所必需的资源设定。

4) 你的$HOME/.Xdefaults文件：（只有在根窗口没有RESOURCE_MANAGER属性定义的情况下使用）。如果你对xrdb尚不熟悉，你便可以此文件取代，但你必须在每一部你执行客户机应用程序的机器上均设定一个。

(3) 主机专用设定 —— 方法5 和方法6

主机专用缺省值和服务程序专用相反，不管应用程序所使用机器的终端机为何，只要应用程序在此主机上执行，均使用主机专用缺省值，你可以用它们来：

- 让应用程序在不同的机器上对不同的文件系统作计算，例如：被一个应用程序读取的数据文件可能在不同的主机上保持不同的位置。
- 区分显示在同一个屏幕上不同的主机的窗口（这些窗口可能由同一个应用程序执行），例如：你可以要所有在mars机器上执行的xterm 的窗口为红色的边框，而在saturn上执行的窗口为黄边。
- 调高一个相同的应用程序在不同的客户机机器上版本的差异，例如： xterm 在venus 是标准的MIT 版，但在neptune 机器上是由第三方厂商修改过以适应机器结构的产品，这两版的xterm 可能并不完全相容。

主机专用Resource保存在：

1) 由$XENVIRONMENT 来的文件名称：如果外壳变量$ENVIRONMENT有定义，它会被解释为一个含有资源设定的文件的完整的路径名称。

2) 你的$HOME/.Xdefaults-thishost 文件：（当$XENVIRONMENT 未被定义时使用）。注意它和我们先前的文件有所不同，它必须附加上主机名称，例如，如果你在 neptune 执行应用程序而在venus 显示（假设RESOURCE MANAGER属性未定义），则服务程序专用资源读取自：

```
Xdefaults
```

而主机专用资源则是：

```
Xdefaults-neptune
```

两者均在neptune 的主目录中。

注意 我们曾说过类似“服务程序专用资源读取自...”这可能造成误导：“如果你实际需要，你可以放置任何类型的资源设定到任何的文件或数据库。”我们真正的意思是你应该放置机器特性或不论资源到任何地方，如果你这样做，你将获得你需要的动作。

(4) 命令行选项 —— 方法7 和方法8

最后，你可以借助于命令行选项设定应用程序的值。通常当你设定缺省值时，为的是你不需要使用选项为你的程序作X相关的设定。但你实际上可以用它们来：

- 一次关闭，例如：你暂时性地在屏幕上需要一个极小的xedit。
- 为了区别在相同应用程序中各自的实例。你已看过一个这样的例子，当我们使用命令

```
xterm -name demo
```

来设定应用程序的实例名称给demo，将造成以应用程序名称为demo的资源替换xterm 的资源。

命令行选项分为下列两种：

1) 应用程序专用选项：例如xclock的-chime的xpr 或-scale。

2) 工具箱标准选项：所有用到工具箱的应用程序均接受一些标准的命令行选项，我们看过其中的大部分，包括-fg, -bg, -display,-geometry 等等。

在其中有一个选项-xrm，重要的足以用一个小节来描述。

(5) 工具箱标准选项-xrm

大多数一般的资源均能被命令行选项明确地设定，例如你可以用 -bg colour设定窗口背景颜色。但无论如何，有一些资源并没有符合的选项。为了克服这点，工具箱提供一个“捕捉遗漏”的选项-xrm（X 资源管理器的缩写）。

-xrm以一个参数当做资源规范，就如同你在缺省值文件中输入的相同，例如：你可以输

入：

```
xclock -xrm "*update:30"
```

和

```
xclock -update 30
```

是相等的。

在同一命令行你可以使用数次-xrm，但每一次只能包含一个资源规范，

例如：

```
xclock -xrm "*update:30" -xrm "*chime:on"
```

-xrm的好处在于你可以用它来设定任何资源供应用程序使用，尤其是那些和命令行选项不符合的资源。其中一些非常有用的像：

iconX, iconY：窗口图标左上角x,y 坐标的位置。

iconPixmap：被用来当作窗口图标的图形的名称，你可以用它来指定任何的图形当作应用程序图标。（图形为已有或利用bitmap程序建立。）例如：命令

```
xedit -iconic -xrm "*IconPixmap:cntr_ptr"\
      -xrm "*iconX:500"\
      -xrm "*iconY:400"
```

其意义为将xedit 设定以图标开始启动，图标的左上角坐标为 (500,400) （在大多数的显示器会在屏幕中央），使用名为cntr_ptr的图形来当作图标。

backgroundPixmap：设定用一个图形当作背景。

borderPixmap：设定以一个图形当作窗口的边，例如：

```
xclock -bw 20 -xrm "*backgroundPixmap: scales"\
xrm "*borderPixmap: cntr_ptr"
```

执行xclock，用一个宽达20个像素的边框，窗口的背景为鱼鳞图案，边框则用 cntr_ptr的图形。

所有的这些资源当然也可用类指定。（如IconX ， BorderPixmap等等。）

注意 请记住，-xrm只有在程序有用到工具箱才可应用。

2. 设定资源不同方法的总结

现在我们将如何对一个指定应用程序资源设定的八种方法进行总结：

- 应用程序专用资源：它们被两个文件掌握，且仅能被工具箱使用，其中一个文件通常由系统管理员设定，另一个由你自己设定。
- 服务程序专用的资源：不是存在根窗口的 RESOURCE_MANAGER属性中，便是在你的$HOME/.Xdefaults文件中。
- 主机专用资源：如果外壳变量$XENVIRONMENT 有定义的话，存在其所定义的文件中，否则在你的$HOME/.Xdefaults-host 文件。
- 一次关闭设定：用应用程序的本身命令行选项来设定和用工具箱标准命令行选项，包含“捕捉遗漏”-xrm。

它们以下列顺序处理：

```
if （程序使用工具箱）
    读取 /usr/lib/X11/app-defaults/class文件 (1)
    读取你的 $HOME/class文件 (2)
if （RESOURCE_MANAGER属性被定义）
    处理内含的指定 (3)
else
```

读取你的$HOME/.Xdefaults文件 (4)
if （外壳变量XENVIRONMENT被定义）
读取所定义名称的文件 (5)
else
读取你的$HOME/.Xdefaults-host 文件 (6)
if （程序使用工具箱）
处理标准的资源选项，包含-xrm (7)
处理应用程序本身的选项 (8)

29.4.2 在服务程序上保存缺省值 —— xrdb

大部分缺省值的结构均和文件有关，当应用程序启动时，不同的文件被读取且其内容被处理，这种方式的缺点为你希望所有的客户程序在一个特定的服务程序上使用同一组的缺省值，但客户程序所执行的机器上如果没有一个共同的文件系统，你该怎么办？

答案是在服务器本身保存缺省值。 X的属性设施是一个具有一般性目的的结构。（记住，一个“ 属性 ”是一小段已知格式资料的名称，被保存在服务程序），指定由服务程序根窗口的RESOURCE_MANAGER属性加载，且当应用程序启动时系统会注意此事。当窗口系统启动时，RESOURCE_MANAGER属性未定义：如果你要使用这个设施，你必须明确地设定它。

并没有一个一般性的工具来操作一个属性，所以X提供了一个特殊的程序来处理资源属性，它就是xrdb（the X Resource DataBase 实用程序）。

1. xrdb能做什么

为了实用起见，本节剩余的部分，我们只把 RESOURCE_MANAGER属性和它的内容当成“ 数据库 ”。

xrdb的功能非常简单，它让你能：

- 设定新的数据库。
- 看目前有那些资源在数据库中。
- 在现存的数据库加入一个新的资源。
- 完全去除数据库。

这些是基本操作，且很容易完成。当然也有一些更进一步的功能可以很精确地让你控制资源，但我们先来讨论基本操作。

2. 使用xrdb的基本功能

xrdb的操作类似大多数UNIX的程序：它从一个文件或标准输入读取输入数据，并且你可以用命令行选项来控制它的操作模式，它所读取的输入是我们曾经看过的一系列资源设定，不过比较特别的是它把这些设定加载数据库，让我们看一看它主要的功能：

- 设定一个新的数据库：输入下面命令两者之一：

```
xrdb filename
xrdb < filename
```

用以将一个文件中的设定加载到一个数据库中，如果只键入 xrdb，表示你将由标准输入（通常为键盘）直接输入设定，稍后我们将说明 xrdb所接受的文件格式，但现在先把输入资源设定当作和 .Xdefaults文件或-xrm参数相同的方法，例如，你可以用下列的方式定义 xclock设定：

```
venus% xrdb
xclocks*Background: pink
xclock*update: 30
```

```
xclock*backgroundPixmap: cntr_ptr
<end-of-file>
```

通常你用一个文件当作xrdb的输入，也就是说，xrdb从一个文件加载缺省值作为你的窗口系统初始化的一部分。如果你很有经验，直接输入它的设定也许容易些。

• 查看现存数据库中的内容，输入命令：

```
xrdb -query
```

则xrdb将以纯文本格式输出数据库的内容（-query可以缩写为 -q ）。

你可能记得也可以在根窗口用 xprop 来看数据库的内容，但 xprop 的输出格式不太灵巧，它给你其他一大堆你不需要的信息。

如果需要，你可以抓取xrdb的输出到一个文件，编辑它，更改设定后可再用它当作 xrdb的输入。

注意 查看数据库，你必须使用选项-query。如果忽略这个选项而只输入xrdb，将造成会清除数据库，且xrdb在等待你自标准输入键入新的设定。

• 在现存数据库加入新的设定：加入新的设定到数据库且不要破坏原有的设定，使用命令：

```
xrdb -merge filename
```

（-merge可缩写为-m，如果你省略文件名称，xrdb会自标准输入读取。）xrdb自指定的文件中读取资源设定，并加入现存的数据库中；对于数据库中已存在的资源，如果有新的设定，旧值会为新值替换，否则不会变动。

• 完全移去数据库：如同先前所述，当系统结束时数据库会自动消失，但如果你在系统仍在执行时移去数据库，使用命令：

```
xrdb -remove
```

3. xrdb的文件格式

你已知道大多数的格式细节，你可以用标准的资源规范的形式：

```
characteristic: value
```

上述的格式你已看过多次，但xrdb有两个额外的规则：

1) 注解：每一行的开头如果是惊叹号（!）会被忽略，所以你可以此当作注解。

2) xrdb缺省将它的输入行传到C 预处理器。

让我们进一步看一看预处理器的过程。

让我们看一看一个你可能碰到的典型问题。假设在一般的场景，你使用下列显示器：

venus	彩色屏幕，正常分辨率。
saturn	单色屏幕，正常分辨率。
mars	彩色屏幕，高分辨率。

以上三者共享一个共同网络文件系统。当你在一个显示器上启动 X，你需要定义缺省值来反应显示器的特征。例如：在高分辨率屏幕你可能需要较大的缺省字体，或是你不需要在单色系统上定义彩色缺省值。

如何做呢？让我们看一看，如果你能使用 .Xdefaults-host 文件：在 .Xdefaults-venus我们包含了彩色指定，而在 .Xdefaults-saturn 我们只放入单色类型的参数。行得通吗？当然，但是是有限度的：它只能掌握应用程序在和服务程序相同的机器上执行，如果应用程序在其他的机器上执行会得到它们主机上的缺省文件。所以如果你使用 venus 且在saturn启动远程的客户机，将会用到 .Xdefault-saturn而错失所有的彩色指定。

你能够只使用 .Xdefault 文件来区分机器吗？不能，因为三台主机共享相同的文件系统，

所以$HOME/.Xdefaults会被venus 获得也会被其他的机器获得。

答案是在资源处理程序的某些地方，有一个结构可以分辨出所使用服务程序的某些特征。xrdb可以用相当简单的办法做到这点，它先定义一些说明服务程序特征的 C 预处理器符号，而后将它所有的输入传递到预处理器，最后将处理过的数据加载数据库。联机帮助列出所有的xrdb定义的预处理器的符号，但在此处我们需要用到的是：

X_RESOLUTION=n:n是每公尺长屏幕有多少像素。（根据我们的服务器，我们正常分辨率的屏幕为每公尺3454个像素。）

COLOR ：只有屏幕支持彩色才被定义。

WIDTH,HEIGHT：屏幕的宽度和高度，单位为像素。

• 你可以使用所有预处理器的功能。例如，我们使用它的表示掌握能力：

```
#if X_RESOLUTION > 3600
```

• 你可以在文件中任何地方使用预处理器符号，并不只是前面有 # 号的行，例如，当

```
xload*Width: WIDTH
```

在venus 上xrdb执行到时，它将会读取成：

```
xload:Width: 1152
```

所以由缺省值可知， xload 窗口宽度将和屏幕宽度相同，高为 80个像素，且在屏幕的正上方。

注意 大多数UNIX预处理器定义了一些和它们机器结构与操作系统相关的符号，这些可能会干扰你，特别是UNIX通常定义的符号。现在xrdb定义HOST为显示器名称中主机名称的部分，所以你可能认为你可以像这样使用一个资源规范：

```
demo*title: X demo using display HOST
```

比方在venus 上，可能希望它相当于：

```
demo*title: X demo using display venus
```

事实上，在我们的机器上会得到

```
demo*title: X demo using display 1
```

原因为显示器名称是unix:0.0，所以主机名称部分为unix，但预处理器已定义了unix，所以整个解释的顺序为：

```
HOST -> unix -> 1
```

你可以用xrdb的-u选项来解除符号的定义，用以克服这点，也就是

```
xrdb -UNIX < filename
```

但即使这样，主机名称仍为unix，除非你明确地指定显示器：

```
xrdb -display venus:0 < filename
```

另一个会产生干扰的样例，如何你输入规范

```
xedit*Font: *-sun-screen-*
```

使用xrdb，现在用一个xrdb -query ，你可以看到在数据库中实际地设定：

```
xedit*Font: *-1-screen-*
```

在我们sun 的机器上，预处理器定义成另一个符号。如果你使用和你的机器相关的名称，你可能也会得到相同的效应。如果你决定不需要预处理器的功能，你可以用 xrdb的-nocpp选项停止它的功能。

4. 如何将数据库设定和xrdb输入文件连接在一起

借助于像前述在一个含有大量预处理器命令的文件执行 xrdb，你初始化了数据库，在稍后的期间，交互式的使用xrdb，你将对数据库做大量的更动，现在你需要记录这些设定，且将之

与原来的输入文件连接，以备将来之用。

如果你只使用xrdb -query ，你只能获得目前的设定：所有在输入文件中的条件指令行若和现在的服务器不符则不会被包含。例如在 saturn上执行前述的文件，则所有颜色和高分辨率的设定，均被忽略（当然以 #开头的也不例外）。为了克服这点， xrdb提供-edit 选项，例如命令：

```
xrdb -edit myresf
```

连接目前在数据库中的值到文件 myresf内存在的内容，它借助于比对资源指定特征值的部分做到这点：如果在文件中某一行和数据库中某一项特征相同，则文件中值的部分会被在数据库中的值替换。用此方法，所有的以 #开头的行和条件设定均会保留在文件中。

注意 预处理器完全不可以使用 -edit 选项，那会导致问题，我们看一下当我们使用 venus ，且以前述文件初始化数据库时，会发生什么情况，假设我们做了更动：

```
venus% xrdb -merge
XTerm*font: *-courier-medium-r-*-140-*
<end-of-file>
```

然后用命令：

```
xrdb -edit myresf
```

将设定更改的部分放回文件，我们看到两件事：

- 前处理器符号在规范中值的部分会被字面(literal)值替换，

例如：

```
xload*Width: 1152 会被
xload*Width: WIDTH 替换
```

- 在规范中只要特性符合，值均会被替换，甚至那些在条件段中目前尚未应用到的也不例外。例如，在前述文件，设定 XTerm*font的那两行（一行在高分辨率那段，一行在正常显示器那段）都会被更改，即使我们只需要改变正常显示器也不例外。

29.4.3 常见的错误和修正

你对系统不熟悉的时候，资源看起来相当的复杂。当有些状况不能正常执行，而系统无法帮助你查觉是什么错误，或你在何处犯了错误。这里列出一些常见错误，并提出如何修正它们。

- 如果你未设定一个应用程序的名称和类，确定在你的资源规范之前加一个星号，（如果你省略这个星号，将没有任何东西会对应这个规范）这个错误在你使用 -xrm时特别常见，例如：

```
xclock -xrm "update:3"  （错误）
xclock -xrm "*update:3"  （正确）
```

- 并非所有的应用程序均使用工具箱，非工具箱的程序不使用类，且它们的属性名称也可能不同。例如，规范

```
* Geometry: 300*400+500+600
```

对xclock，xlogo 有效，但对xcalc 无效，因它不使用工具箱。xcalc 使用属性名称Geometry（开头为大写的G ），因为在这种情况，工具箱类名称和 xcalc 的属性名称相同，所以单独一个规范

```
*Geometry: 300*400+500+600
```

可以对所有这类的应用程序有效。

- 你可能在规范中用了错误的属性或Widget的名称，特别是容易把类名称和实例名称搞混，

例如：以下两者均错：

```
xclock*Update: 10
xclock*interval: 10
```

其他常见的错误如：

```
xterm*Text*background:blue
```

它不能执行的原因为xterm 并未使用Text Widget ， xterm 正常的窗口和Tektronix 的窗口分别使用Widget类 VT100和Tek 。最后，当你知道一个Widget是什么类，你可能对实例名称假设错误，不是Widget 本身便是其中之一的属性。试着更换类名称来修正这个问题。

- 即使你已设定实际的Widget和属性名称或类，应用程序可能以不是你预期的方式使用它们。例如：你可能设定如下：

```
xterm*Width: 40
xterm*Height: 10
```

希望用比平常较小的窗口启动xterm ，但它不能执行。 xterm 只能在Tektronix window应用这些值，无法在正常的窗口。

- 你可能所有的设定完全正确，但仍然什么也没发生，例如：

```
xmh -xrm "*inc.Label: Include"
```

是一个正确的方式。但在标准系统的发行版，是没有任何动作发生的，原因是 xmh 有一个应用程序设定缺省值文件/usr/lib/X11/app-defaults/Xmh ，其中有一行：

```
xmh*inc.label:Incorporate New Mail
```

这个规范较我们的设定有较高优先。

- 将规范

```
*Width: 200
```

单独包含在数据库将导致大多数的工具箱程序启动失败，且有一个信息说它的“ shell Widget has zero height or width ”。如果你设定height和width 二者之一，你必须也设定另外一个。

- 如果你用编辑器建立一个资源文件，你可能省略了最后一个新行。这将导致当你试图用xrdb加载它时整个文件均被忽略。为了避免这样，当加载资源时，用一个命令行像：

```
xrdb resfile; xrdb -query
```

如果xrdb无法打输出resfile 中的内容，就是有问题了。

- 你可能忘了用 -xrm选项的参数来获得资源规范，有时有人会把资源规范放入一个文件，而以文件名称为-xrm的参数，预期它自此文件中读取资源。
- 最后，一个非常人性的错误。当你发生问题，你通常会循环动作：编辑 resource文件，保存它，加载资源到数据库，执行应用程序和看一看发生什么状况。但是常常会忽略其中 " 加载资源到数据库 ” 以致你更为困惑。

29.5 定制键盘和鼠标

电脑的键盘通常含有一些特殊功能键，在此有一些方法来制定这些特殊功能键，使它们能完成特定的功能以适合你工作的方式。例如，你可以定义一些键来输入那些你常用的命令，或只需按一个键便能够输入一段程序。

在X中，你能制定的不只是功能键，其他一般的键和鼠标的按钮也都可制定。对每一个应用程序，你均可指定特别的功能给键盘和鼠标按钮，或两者的组合 (例如在 xedit中你可以结合 Shift 键和鼠标的右按钮来让你向前移动一个单字)。所有使用X 工具箱的程序均允许用户利用

一个被称之为键盘转换的设施来执行此种定义，且此种定义借助于正规的资源结构传递给应用程序。那些不使用X工具箱的应用程序，同样地也可以用相同的设施来制定，但它们需个别的定义所以不能广泛地应用，从现在起，我们假设每当讨论有关转换的种种，均为对那些使用 X 工具箱的应用程序而言。

就如同所有的资源一样，转换是当应用程序执行时才处置。例如你可以拥有数个具备不同转换设定的 xedit，在同时一起执行。你可以让一个 xedit 适合编辑文本，另一个适合编辑程序码，而另一个适合编辑文书。

本节讨论转换 —— 包括它们的定义格式，如何将它们设定到应用程序和它们所涵盖功能的范围。我们首先以实例来介绍，逐渐地导引你看到不同的角度。而后比较正式和详细地讨论转换。最后，我们列出当你使用转换时常会碰到的错误，并给你一些如何克服这些问题的提示。

29.5.1 实际使用转换

工具箱转换结构最简单的用途便是让你制定你键盘的键。例如，当你使用 xterm 为一个执行一般外壳命令的窗口时，你可能希望定义一些特殊功能键来输入你常用的命令，且希望指定的关系如下：

当我按下F1这个键时，我希望这个字符串输入

```
F1  rm core *.tmp <newline>
```

利用工具箱达到此目的方法为：给使用转换的 Widget中的资源指定一个值。此值设定应用程序中所必需的定制，且使用工具箱的 Translation Manager (转换管理器) 所处理。此资源属于类Translation,且其实例名称几乎一定是translations。

1. 如何对应用程序指定转换

对前述xterm 的例子，我们定义 (在即将被应用程序读入的资源数据库中或一些资源文件中) 一个规范类似：

```
xterm*VT100*Translations: (contd。)
<key>F1: string("rm core *。tmp")          (注意：不完整！！)
```

其意为在任何类 VT100的 xterm Widget 中，当键 F1 被按下时，插入字符串“ rm core *.tmp ”)。

不幸的是，并没有这么简单。转换管理器会把上面的规范解释为 “ 去掉所有现存的转换，且加入..)”，所以所有正常的像“ A键是插入一个 A ”这种对应关系都会消失。为了克服这点，你必需在资源值之前插入一些称为伪指令的语法：

```
xterm*VT100*Translations: #override(contd.)
<key>F1: string("rm core *.tmp")
```

通常你会希望保持大部分现存的对应关系，而只是把你明确指定的值覆盖上去。所以你一般都是在你的转换表中，指定 #override。

现在这个规范可以开始工作了。通过启动一个 xterm，且把此规范 (在两个单引号 (’)中间的部分) 当成选项 -xrm 的参数来测试它：

```
xterm -xrm 'xterm*VT100*Translations: ...'
```

按下特殊功能键 F1,你将看到指定的字符串成功的插入，但并未包含新行字符，你可以用一点语法的技巧来克服它，像：

```
 xterm*VT100*Translations: #override(contd.)
  <key>F1: string("rm core *.tmp")string(0xd)
```

这解释了以下两点：

1）string()的作用和它的参数相关。你可以直接输入文本（例如 string(lpq)），但如果文本包含空白或非字母字符，则必需在文本前后加上双引号。如果参数是以“ 0x ”开头，则将其后解释为十六进位，并插入相对的ASCII 字符。(例如，0xd是RETURN)

2）在你指定此功能时可结合一个以上的作用。在上例，我们用到 string()作用两次，如果我们知道其他的作用，我们也一样可以对应起来。

你可以根据需求在一个表中定义许多的转换。假设，我们在前面的转换中增加对应关系：

当按下F2这个键时， 我希望这个字符串被输入

```
F2  lpq-Plpa3 <newline>
```

对此的转换为：

```
<Key>F2: string("lpq-Plpa3")string(0xd)
```

所以可以将本行加入前面的表中。但是转换管理器的格式规则告诉我们必需将两个转换以“ \n ”分开且独立成为一行：

```
xterm*VT100*Translations: #override(contd.)
<key>F1: string("rm core *.tmp")sting(0xd) \n(cond.)
<Key>F2: string("lpq-Plpa3")string(0xd)
```

以上的形式将造成管理上的困难。你可以借助于包含“ 隐藏的新行字符 ”来使它容易理解： (新行字符以倒斜线 “ \ ”处理)

```
xterm*VT100*Translations: #override\n\
<key>F1: string("rm core *.tmp")sting(0xd)\n\
<Key>F2: string("lpq-Plpa3")string(0xd)
```

你可以放置任意多个你所需的“ 隐藏的新行字符 ” ,且几乎在任何地方均可，它们只是被忽略而已。(只要和转换管理器相关，甚至你每隔一个单字便使用一个也没关系。但千万不要在一个规范的资源特征部分使用它们，资源管理器无法解释它们，也没有相同的效果。) 如果你感觉有些混淆，不用担心。简单地说，资源结构需要的是要在一行中的一个资源规范的“ 值 ”的部分，而转换管理器以分开的行来分开 (也就是以\n终结)，而用户刚好以每一个实际分开的行代表一个意义以增加可读性，所以规则很简单：

在除了最后一行的每一个转换行均加上一个 “ \n\ ”。

2. 转换可对应许多型号的作用

上述的 xterm例子，演示了如何能够当你按下一个键时，插入任意的字符串。但转换结构的功能比这更多，它可以将任何Widget所提供的作用对应到按键，让我们详细一点地看一下这些作用。

前述的例子，我们在 xterm的 VT100 Widget完成了键F1和F2在 string()上的对应。我们将仍以 xterm为例，说明更多的作用。

查阅xterm 的联机帮助，在标题KEY TRANSLATIONS 和 KEY/BUTTON BINDINGS你将发现列有数个作用。我们将定义一个转换对应键 F3到insert-selection()作用之上，所以我们可以用键盘来替换鼠标，将先前“ 剪 ”下的文本“ 贴 ”出。联机帮助告诉我们此作用需要一个参数，从列出的缺省对应，我们可以看出缺省的 “ 剪贴 ”结构为使用 CUT_BUFFER0, 所以我们将CUT_BUFFER0当作参数。我们的资源规范是：

```
xterm*VT100*Translations: #override\n\
<key>F3: insert-selection(CUT_BUFFER0)
```

到目前为止，这只是一点小小的使用。然而，假定说你花了许多时间在文本文件上工作，你用 tbl格式化，你用 nroff在屏幕上预览它们，用troff 排版，且将输出送到你的一个用过滤器为tr2printer的打印机上。设定转换为：

```
xterm*VT100*Translations: #override\n\
<key>F3: string("ed") insert-selection(CUT_BUFFER0)\
string(0xd)\n\
<key>F4: string("tbl") insert-selection(CUT_BUFFER0)\
string("| nroff -man") string(0xd)\n\
<key>F5: string("tbl") insert-selection(CUT_BUFFER0)\
string("| troff -man -t | tr2printer") string(0dx)
```

xterm 会确定这些转换是以xrdb自数据库加载或是在一个资源文件中，并加以处理。现在当你启动xterm，用鼠标“剪”取你所需的工作的文件名称。接下来，便可按 F3键编辑它，按F4键预览它和按F5键在硬拷贝上排版它。

(1) 更多的Widget作用样例 —— xbiff

查阅xbiff 的联机帮助：在ACTIONS 的标题下，你将看到Mailbox Widget所支持作用的名单。它惟一缺省的转换为当你按下任何按钮时降下邮件邮件的标记(作用unset())。我们将设定转换让你以键盘来运用这些作用，将这些作用对应到“？”和“UP”“DOWN”两个方向键如下：

?	check()有新的邮件吗?
UP	set() 升起邮件的标记
DOWN	unset()降下邮件的标记

以下是相关的转换表：

```
xbiff*Mailbox*Translations: #override\n\
<Key>?: check()\n\
<Key>Down: unset()\n\
<Key>Up: set()
```

以此测试之：用xrdb从你的资源数据库加载这些设定，然后启动xbiff ，将鼠标指针移到窗口内。重复地按下Up和Down光标控制键以升起和降下邮件标记。

(2) 找出提供哪些作用

你对Widget作用将和Widget名称遇到相同的问题：如何找出某个 Widget到底提供哪些作用以及它们能做些什么？同样地，没有一个完美的解答。但有一个合理的方法来处理：

1) 查看应用程序的联机帮助。大多数的应用程序有它们自己专门的作用文件。例如： xbiff有一节叫做ACTION，而xterm 有两节关于转换和作用的文件 —— KEY TRANSLATIONS 和 KEY/BUTTON BINDING。

2) 最初的联机帮助可能给你提示，或甚至直接告诉你它用到何种 Widget的类，所以你可以查看它的Widget集文件中的特定的Widget。(在 core 版中惟一的Widget集为Athena，所以你在此不易出错)。即使联机帮助未告诉你 Widget的类，当你对系统熟悉之后，你将对一个 Widget是否为标准类型较具有概念，如果还是不行 ...

3) 查看程序的源代码，看看用到什么Widget的类，以及Widget提供了哪些作用。

3. 转换对应作用到一序列事件，不只是单一键

我们已经看到转换使得你设定插入，转换结构也能让你对应这些作用：它可以是单一的键，或是一序列的键，或者是事实上一序列任何的X事件。

让我们继续以xbiff 为例，看看如何转换一序列的键盘字符。例如我们定义字符字符串的转换如下：

```
look   check()
raise   set()
lower   unset()
```

以下为相关的转换表：

```
xbiff*Mailbox*Translation: #override\n\
<Key>l,<Key>o,<Key>o,<Key>k: check()\n\
<Key>r,<Key>a,<Key>i,<Key>s,<Key>e: set()
<Key>l,<Key>o,<Key>w,<Key>e,<Key>r: unset()
```

以此测试之 —— 加载设定和启动xbiff ，将鼠标指针移到窗口内。现在你可借助于输入完整的字符串来升起和降下标记。例如键入五个字符 r、a、i、s、e以升起标记。对xbiff 的两个表有几点值得说明：

- 键的名称可以用不同的方式指定。正常的输出字符直接指定 (如<Key>w), 其他的字符则拼出全名(如<Key>Down)。
- 对字符字符串，你必需一一指定，并以逗点分开(如<Key>l,<Key>o, <Key>o, <Key>k) 。
- 转换可允许相同开头的键，例如 look和lower均拥有相同的开头lo，对转换管理器不会形成问题。

(1) 找出键的名称

找出转换所需的键的名称，最简单的方法为执行 xev ，将鼠标指针移到窗口内，按下你所需的键，则键的名称会出现在括号内字符串 keysym和一个十六进位数之后。例如在xev 的窗口内按下光标控制键DOWN，在其中你会看到：

- (keysym 0xff54, Down)。
- 也就是说，键的名称为Down。
- 你可以在转换中使用任何类型的事件。

到目前为止，我们所写的转换都是对应作用到一个按下的键盘字符。但我们曾说过，转换结构可对应作用到任何事件，而不只于按下键盘而已。可能的事件类型非常的多，在此我们只提及一小部分：

<Key>	按下一个键
<KeyDown>	按下一个键 (只是另一个名称)
<KeyUp>	释放一个键
<BtnDown>	按下一个鼠标按钮
<BtnUp>	释放一个鼠标按钮
<Enter>	鼠标指针进入窗口内
<Leave>	鼠标指针移出窗口外

我们已经使用过按下一个键的事件，让我们绑定xbiff 作用到鼠标按钮以替换之：

```
xbiff*Mailbox*Translations: #override\n\
<BtnDown>Button1: unset()\n\
<BtnDown>Button2: check()\n\
<BtnDown>Button3: set()
```

你可以看到语法和前面相似：你先给定一般性的事件类型 (例如 <Key> 或<BtnDown>), 其后跟着你所需事件的事件细节部分，例如s和Button3 (Button 1、2、3分别对应到左、中、右按钮)。

(2) 对一序列的事件的转换

就如同我们定义了一序列按下键事件的转换 (set、unset和check)，我们当然也可以定义一序列的鼠标事件。事实上你转换的一序列的事件可以任意组合在一起，你可以在一个转换的左边随意混合事件的类型。所以你可以定义如下的转换表：

```
xbiff*Mailbox*Translations: #override\n\
<BtnDown>Button1, <Key>?, <BtnDown>Button3: check()\n\
```

```
<BtnDown>Button1: <Key>u, <BtnDown>Button3: unset()\n\
<BtnDown>Button1: <Key>s, <BtnDown>Button3: set()
```

也就是说，用到check()，你必需依序先按下按钮1 (左按钮)，然后按下“？”键，最后按下按钮3 (右按钮)。这个样例并不是很好，但对于一些危险或不可取消 (irreversible) 的作用 (例如删除一个文件，或是覆写一个缓冲区的内容) ，你可以依照这种方式来使用转换。你需要使用一个非常谨慎的命令序列，才能用到此作用，这样使得用户不可能因意外而输入此命令。

(3) 使用非键盘和非鼠标事件的转换

通常你是对按下或释放鼠标按钮或键盘的键定义转换。但我们曾经说过，你可以对任何事件设定转换，例如鼠标指针移入或移出一个 Widget的窗口。让我们以xman的主选项窗口为样例来解释它。这是一个相当人为的样例，因为它没有任何用途。但无论如何，它很容易被看出在做些什么操作。

查看xman的联机帮助，在X DEFAULTS标题下，你将看到概括的xman所用到的Widget的名称和类：主选择项窗口Widget的名称叫topBox，类名为Command。这是一个好的猜测，因为在菜单操作框的方法。我们可用上面提过的技巧来确认它，使用以下的命令：

```
xman -xrm "*Command*backgroundPixmap: scales"
```

这和我们先前的样例有一个重要的不同：我们所用到的作用不是由特定的应用程序指定，而是由标准的Widget提供 (本例中为Command Widget，在“X 工具箱Athena Widget”使用联机帮助中有描述)。

在我们定义任何东西之前，先来看一看此Widget缺省的功用，以便我们能够了解有些什么事发生和有哪些Widget的作用会做。启动xman，移动鼠标指针进入 Help 框，你会看到框的外框变成高亮度显示 ，这是 highlight() 在作用。将指标移出，框的外框恢复正常，这是 unhighlight() 作用。将鼠标指针再度移入 Help 框，按下一个鼠标按钮，保持按住不放。则框内的颜色反转 (框内的文字变成缺省的背景色，而原来窗口的背景变成窗口的前景色)，这是set() 在作用。继续保持按住鼠标按钮，将鼠标指针移出窗口外，框内颜色恢复正常，这是 reset() 在作用。一个正常“按一下”(clicking on) Help框的次序为：

1) 移动鼠标指针进入框中：highlight()将外框变为高亮度显示。

2) 按下按钮：set() 反转框中的颜色。

3) 释放按钮：notify()开始作用，造成程序建立求助窗口 (help window)。在进行中时，框的颜色保持反相。当窗口建立完成之后， reset()反转框内的颜色为正常，但外框仍保持高亮度显示。

4) 将鼠标指针移出窗口：unhighlight()将外框恢复正常。

现在你了解了有哪些作用，我们将定义一些转换来改变原先进出窗口的作用：

```
*Command*translations: #override\n\
<EnterWindow>: reset()\n\
<LeaveWindow>: set()
```

用这个奇怪的转换表，当你一开始移动鼠标指针进入框中，什么事也不会发生，但当你移出鼠标指针时，颜色会反转。如果你再度移动鼠标指针进入框中，颜色会变回正常。其他的作用和前述相同。

(4) 使用修饰键来修饰事件规范

有时你指定的转换希望能同时按下一或多个修饰键，例如你要对应一个作用到和 META键同时按下的一个键，或是当Ctrl和Shift 同时按下的鼠标按钮。到目前为止我们还没有任何办法可指定这样。我们不能用事件序列完成这点，因为它是依序定义的，而我们需要的是指定同时，

例如“按下X键且ctrl键同时被按下”。

欲在转换中指定修饰键，你只需在事件名称之前加上你所需的修饰键名。例如在 xterm 中，定义meta-i为“贴”上一次“剪”的文本，使用：

```
*VT100*Translations: #override\
Meta <Key>i: insert-selection(PRIMARY, CUT_BUFFER0)
```

因为这种修饰键／事件类型的组合十分常见，转换管理器允许使用一种缩写的形式。相等于上面第二行的写法为：

```
<Meta>i: insert-selection(PRIMARY, CUT_BUFFER0)
```

我们可以对鼠标事件做同样的处理。让我们对 xedit 定义转换，使得使用鼠标可以在文本上实用地移动，我们首先的尝试如下：

```
*Text*Translation: #override\
Shift <Btn1Down>: forward-character()\n\
Shift <Btn2Down>: forward-word()\n\
Shift <Btn3Down>: next-line()\n\
Ctrl <Btn1Down>: backward-character()\n\
Ctrl <Btn2Down>: backward-word()\n\
Ctrl <Btn3Down>: previous-line()
```

如果你测试它，奇怪的现象会发生——光标好像会自行其是，而且文本段会一下子被选择，一下子又取消选择。发生这种现象的原因是 Text Widget 的缺省对应仍然会作用，它包含的转换像：

```
<Btn1Up>: extend-end(PRIMARY, CUT_BUFFER0)
```

你可能认为这不会影响你，因为当你释放按钮时你总是按着 Shift键或Ctrl键。但事实上会作用：转换管理器对于你未定义的修饰键解释为你不在乎它们的影响，所以释放 Button1 时会对应到上述的规范。为了克服这点，我们对那些可能不小心便会发生的按钮释放事件定义转换，并对应绑定到一个空作用。这些转换当被对应到时会盖掉缺省的转换。对使用 Text Widget 我们需再增加两行，才是一个完整的转换表：

```
*Text*Translation: #override\
Shift <Btn1Down>: forward-character()\n\
Shift <Btn2Down>: forward-word()\n\
Shift <Btn3Down>: next-line()\n\
Ctrl <Btn1Down>: backward-character()\n\
Ctrl <Btn2Down>: backward-word()\n\
Ctrl <Btn3Down>: previous-line()\n\
Shift <BtnUp>: do-nothing()\n\
Ctrl <BtnUp>: do-nothing()
```

这说明了下列几点：

- 我们对鼠标事件使用了缩写的语法，也就是先前的语法像 <BtnDown>Button1以<Btn1Down>替换。转换管理器容许一些缩写的语法存在。我们在前面看到的 <Meta> 也是一例。
- 我们用 do-nothing() 当作一个哑作用，就好像它是列在 Text Widget的文件中一样。事实上这个作用是不存在的，因此会导致错误的信息出现，但因为我们本来就是要用它来什么事也不做的，所以无需介意。
- 对于我们方才指定的哑作用，我们用了一个事件 <BtnUp>便代表了三个按钮。相同地，转换管理器把从缺的修饰规范的解释为“对任何”，在一个事件中缺少细节部分 (例如在规范“<BtnUp>Button1”中“Button1”的部分) 解释为“对任何所有的细节部分”。

这点在转换中有一个非常常用的形式为：

```
<Key>: ...
```

因为缺少细节部分，所以可被用于所有按下键 (key-press)事件，也就是对所有的键。事实上在Text Widget 上有一个缺省的转换为：

```
<Key>: insert-char()
```

insert-char()作用的功能为当一个键被按下时，插入相对应的 ASCII字符。

4. 复合的转换表及例子

到目前为止，我们把所有的转换均应用于整体的 Widget类。但你能对个别的Widget指定转换，就如同资源一般。在此我们将对 xman定义更多的转换。我们将对Help框Widget (对应作用到助忆符)只用到键盘事件，对 Quit框只用到窗口事件。为了达到此点，我们将对转换应用到的Widget 给予明确的名称。我们的转换表如下：

```
*Help*translations: \
<Key>h: highlight()\n\
<Key>u: unhighlight()\n\
<Key>n: notify()\n\
<Key>s: set()\n\
<Key>r: reset()\n\
<Key>LineFeed: set() notify()
Quit*translations: #override\n\
<EnterWindow>: reset()\n\
<LeaveWindaw>: set()
```

有几点特别的语法需要注意：

- 在此我们对相同类中不同的 Widget指定不同的转换，所以我们需要知道实例名称。不幸的是，这些实例名称 (Help、Quit、Manual Page)并不明显。如果它们在文件中找不到(本例即找不到)，那你只能用猜的或是去查看原始程序了。
- 对于Help，我们省略了常用的#override，因为我们对此Widget不需要考虑任何缺省的绑定。特别的是，当鼠标指针进入窗口时，我们不要此 Widget呈现高亮度显示，这样我们才能看出这个转换的效用。
- 由于省略 #override，我们将这个转换规范移至第一行。如果不这么做，而且对第一行仍以\n\ 作结束，我们将得到错误：

```
X Toolkit Warning: translation table syntax er-
ror: Missing ':' after event sequence.
X Toolkit Warning: ...found while parsing ''
```

因为\n是用来区隔转换规范或类似像 #override 指令的。而将此行和第一个规范以隐藏的新行字符区隔，就如同：

```
*Help*translations: \
<Key>h: highlight()\n\
...
```

- 对LineFeed那一行的转换，包含了复合的作用，和前面 xterm 中复合的string()作用类似。

29.5.2 转换 —— 格式和规则

转换是一个由工具箱提供的一般性结构，它让用户指定当某些特定的事件由 Widget接收到时，一个Widget应完成何种作用。工具箱中处理转换的部分被称之为转换管理器。

转换由Widget指定，它的确是一个Widget的每一个实例。一个转换的集合称之为一个转换

表，而这个表借助于标准的资源结构传递给应用程序。Widget 会有一个Translation 类的资源属性，通常的实例名称为 translation。这个转换资源期待的一个值，即一个转换表。就像所有其他的资源一般，你可以在同一个应用程序对不同的 Widget指定不同的资源，而且你能以类名称或实例名称或二者混合来指定它们。

每一个Widget定义了它所提供的作用，不论是在数量或类型上，它们都是极富变化的。

转换可被各种不同类型的事件指定，不仅只于键盘和鼠标事件而已。任何序列的事件均能被处理，就如同单一事件一般。

1. 转换表的格式

一个转换表大体上的格式如下：

[optional-directive\n] list-of-translations

每一个 list-of-translations 由一或多个转换组成，格式如下：

event-sequence : list-of-actions

当event-sequence发生时，规范中的list-of-actions 会由Widget所完成。如果在一个表中，有多于一个的转换，每一个需以“ \n ”区隔开。

2. 转换伪指令 —— #override 等等

选项伪指令告诉转换管理器，它应对任何已设定之相关 Widget在此转换集合中应如何处理。

#replace : 清除所有现存的对应，只采用在转换表中所含有的 (只使用新的)。

#override : 强制留下现有的对应，加入转换表中。如果在表中有任何项目设定，旧有的即被覆写。也就是说，旧有的被新有的替换。结合旧有的和新的，但新的比较重要。

#augment : 强制留下现有的对应，加入转换表中。如果在表中有任何项目设定在现有的设定存在，使用旧的而忽略新的(结合旧有的和新的，但旧的比较重要)。

如果未设定伪指令，缺省为 #replace 。

3. 个别的转换规范格式

每一个转换的格式为：

event-sequence : list-of-actions

让我们来看一看此规范的两个部分。

(1) 事件和事件序列的格式

一个事件序列包含一或多个事件规范, 其格式为：

[modifiers] <event-type> [repeat-count] [detail]

除了事件类型外，均为可选择。(<>中为必需)。

modifiers : 这是基本设计中比较精巧的部分，我们在下一段说明。

event-type : 指定我们有兴趣的事件的类型，例如按键 (<KeyDown>)、释放按钮(<BtnUp>)或鼠标指针离开窗口(<Leave>)等等。

detail : 指定我们有兴趣的特定类型。如果你省略细节栏 (detail field)，事件规范将对应到任何detail，这样，<Key>将对应到所有的按键事件。此格式指定到每一个事件类型。对指定事件类型的细节栏为：

- 对<Key>、<KeyUp>和<KeyDown>事件，细节如果不是键的名称 (例如<Key>s)，便是keysym (keysym是按键以开头为0x的十六进位数表示，将于下一节详细解释)。
- 对于按钮事件，细节就是按钮的名称，也就是 Button1到Button5 中的一个。例如我们先前使用过的<BtnDown>Button1。

类型/细节的缩写：常用于转换管理器的一些事件类型和细节的组合，允许你对它们使用

缩写：

缩写	相等的全名
<Btn1Down>	<BtnDown>Button1
...	
<Btn5Down>	<BtnDown>Button5
<Btn1Up>	<BtnUp>Button1
...	
<Btn5Up>	<BtnUp>Button5

repeat-count：这指定了事件需要的次数。如果指定，它们包含在括号之中。例如：

```
<Btn1Down>(2)
```

指定需对一号按钮(button-1)按两次。如果你在后面再加上加号 (+)，其意为按的数目需大于或等于指定。例如：

```
<Btn1Down>(3+)
```

意为需按三或更多次。缺省的重复次数为一次。

一个事件序列以一或多个事件规范组成，以逗点分开。当这个事件的序列在其 Widget发生时，相关的作用便会运作。

当序列发生时，转换管理器会根据一些规则决定它自己是否被满足。我们用一个例子以便仔细地观察，假设你对两个字符序列set 和unset定义了转换：

- 简单地说，如果个别的事件依序发生，转换管理器会被满足，其他的事件 (那些你未指定的事件) 如果在指定的序列中间发生，不会妨碍序列被满足。例如， set 可被sweat 和serpent 对应。
- 如果介于其间的未指定事件，启动了转换表中的另一个事件序列，转换管理器会放弃原先的序列，而尝试着去满足新的序列。例如， set 不会被sauerkraut对应，因为u 会使得转换管理器对应到unset。
- 如果在一个事件的集合中有超过一个的事件序列发生，转换管理器只会应用到一个转换：
- 如果一个序列对应到结束 (右端)，较短的那个序列只有在不包含于较长的序列才会发生。所以如果unset 被对应到，对set 转换将不会作用。
- 如果一个序列是在另一个序列的中间发生，例如，如果你定义序列 at和rate，则较长的那个永远不会被对应到。

(2) 事件修饰键

修饰键是一些键或按钮，系指当主要事件发生时，那些必需被按下才会让转换管理器满足的键或按钮。你可以对键、按钮、移动、进出窗口等事件指定修饰键。常见的修饰键为：

```
Button1  ...Button5
Ctrl  Shift  Meta
Lock
```

如果未指定任何的修饰键，转换管理器会解释为：“ 当事件发生时，不论修饰键是否被按下，均会被接受 ”。例如，<BtnDown>会被满足，不论当时Shift 或Meta键是否有被按下。

如果真的需要指定 “ 只有在没有修饰键被按下时才接受此事件 ”。则需使用伪修饰键None。例如， None <BtnDown> 会使得当按钮按下时若META键也被按下则不会满足。

对一个事件指定一些修饰键意为 “ 只要符合转换中指定的修饰键，其他的修饰键不需介意 ”。它并没有 “ 一定要完全恰好符合才可以 ” 的意思。例如， Ctrl <Key>a 在你按下 meta-Ctrl-shift-a 时仍会被满足。

如果你真的要指定 “ 只有刚好符合修饰键的才要 ”，则需要在修饰键之前加一个惊叹号 (!)。

例如，!Ctrl <Key>a 在你按下 meta-Ctrl-shift-a 时不会被满足。

对一个修饰键的集合 (可能是空集合) 作限制，意为“ 除了这些修饰键不接受 ”，需要在不接受的修饰键之前加一个 (~)号。例如， Shift~Meta <Key>t会被Ctrl-shift-t满足，不会被meta-shift-t满足。

键事件通常忽略大小写，如果你要区分，需在之前加一个冒号 (:)。例如，不论 H或 h均可符合<Key>H，但只有H 才符合 :<Key>H。

就如同对常用的事件类型 /细节配对有缩写一般，转换管理器对常用的修饰键 /事件类型配对同样地提供缩写：

缩写	相等的全名
<Ctrl>	Ctrl <KeyDown>
<Shift>	Shift <KeyDown>
<Meta>	Meta <KeyDown>
<Btn1Motion>	Button1 <Motion>
...	
<Btn5Motion>	Button5 <Motion>
<BtnMotion>	任何按钮的 <Motion>

(3) 作用的格式和作用的表列

每一个转换在一或多个作用之上对应一个序列的一或多个事件。在表列中的个别作用是以空白分开的。不可用逗点分开，那将会导致错误。

个别的作用格式如下：

```
action-name(parameters)
```

即使没有参数被指定，在作用名称 (action-name)后的括号，仍然不可省略。

例如：

```
start-selection()
```

如果在作用名称和左括号中间留有空白，你将会得到一个错误。

作用名称只包含了字母、数字、美元号 ($)、底线(_)四种字符。每一个 Widget提供它自己的作用集合 (如果有的话)，且自我包含这些作用名称的硬代码列表。

参数是一个零到多个字符字符串的列表，中间以逗点分开。参数的意义为对特定的作用作指定 (事实上大多数的作用并没有任何参数)。参数字符串可以不加引号，例如：

```
insert-selection(PRIMARY)
```

或者前后加上双引号，这种情形通常为参数字符串内包含了空白或一个逗点，例如：

```
string("plot<x,y>")
```

没有一个一般性的方法，让你在参数字符串中的任何位置包含一个双引号，虽然像这样 string(ab"cd) ’ 将双引号放在字符串中间是可以处理的。也没有一般性的方法在同一个参数字符串中同时包含字符串和双引号。因为这样，有些 Widget在解释它们自己的参数时，可以自行加入它们自己的语法规则。例如：对 xterm 的VT100 Widget的 string()作用，如果一个不带双引号且开头为“ 0x ”的字符串，此字符串被解释为代表一个 ASCII 字符的十六进位数。

现在结束我们对转换规范及格式的描述。由此，你应有能力了解在不同 X联机帮助列出的转换，且可写你自己的转换。为了帮助你，下节列出你常见的问题，以及如何克服它们。

29.5.3 转换规范中常见的问题

转换在观念上简单，但实际上很混乱。即使你常常使用，语法仍然复杂而难解。无论如何，如果你是初学者，最好的方式是你以别人的转换当作自己的转换的基础。在联机帮助中有几个

对xbiff、xdm、xterm 的转换样例，将对你有所帮助。

如果你发现转换有错误的话，有几点值得去检查：

- 转换只能应用在使用工具箱的程序上。如果你试图对非工具箱应用程序定义转换，看起来不会有任何问题，只是转换不会作用而已。

让我们来看一下为什么，以对xcalc (这是一个非工具箱程序)使用转换为例。你对一个资源名称像 *xcalc*translations定义一个转换表，且用xrdb加载至你的数据库。xrdb并不会抱怨，因为它不知道是那一个应用程序使用到资源。它只会设定数据库，稍后供资源管理器查询。现在你执行xcalc ，它对转换是一无所知，所以不会向数据库查询转换，当然也绝不会编译它们了。

- 不要省略 #override，除非你确实知道你要做什么。如果你因错误省略它，例如在xedit 中，你将发现没有任何的键可输入任何的东西 (因为缺省的转换“ <Key>:insert- char() ”被去掉了)。
- 检查你的每一行均有结束符。如果你在转换表中的一行忽略了“ \n\ ”或“ \n ”，在其后所有的转换都会被忽略。如果你在最后一行的末端加上一个倒斜线 (\) ，或是省略了文件中最后一个新行字符，整个转换表都会被忽略 (不过这是xrdb的问题，而非转换管理器的问题)。这种错误在编辑现存的转换表时特别容易发生。
- 当你定义的转换和缺省有冲突时，可能会导致奇怪的行为，特别是对鼠标按钮事件，每一次按下或Down事件，会相关到一个释放或 Up事件，当你对此部分没有明确定义时，可能会有一个缺省的对应仍然存在(键盘的按下和释放也是成对的事件)。所以：

1) 检查缺省绑定的文件。

2) 如果你只对按下/释放配对的一半指定一个转换，确定另一半并非缺省转换的一部分，如果是的话，需对它明确地指定一个转换。

3) 如果你仍然不能解决，暂时由表中移去#override，这将去掉所有的预设转换，让你了解问题是由于和缺省转换冲突所造成，还是因为你的转换表有错误。

- 转换管理器对语法不正确的问题，无法很好的告诉你原因何在。例如如果你有一个转换像：

```
<Key>F6: string("abc""def")
```

参数的语法并不正确， F6键将没有作用，但你也看不到错误信息。

- 如果你转换一序列的事件，且需要对每一个均指定修饰键，你必需明确地对每一个都指定。例如，如果你需要一个转换使用Ctrl-X Ctrl-K：

```
Ctrl <Key>X, Ctrl <Key>K:...
```

而如果使用：

```
Ctrl <Key>X, <Key>K: ...
```

你的指定为 Ctrl-X K

- 检查你所需的Widget是否有你指定的名称和类。例如对 xterm ，你可以在一个表的开头指定：

```
xterm*Text*translations:
```

这将什么事也没作， xterm 正规窗口Widget的类VT100。通常，不论xrdb或转换管理器均不会有反应，因为看起来没错。

- 转换可能指定正确，也可以工作，但它的作用和你预期的不符。例如对 xterm 的转换：

```
Meta Ctrl <Key>m: mode-menu()
```

是正确的，且会工作。但mode-menu()实际上检查鼠标左或中按钮是否有招唤它，其他方面不做任何事。

- 在一个转换中不指定修饰键，并不意味着当修饰键按下时转换会无效。它真正的意义为：

“我并不在乎有没有修饰键”。如果需要的话，使用“None”，“”或!符号。使用时要小心缺省的转换是否会妨碍到你。

- 转换是针对Widget而指定的，所有在转换中的作用必需由Widget提供。在你指定转换资源名称的地方很容易忘掉这一点。例如：

```
xman*translations: \
<EnterWindow>: reset()\n\
<LeaveWindow>: set()
```

将导致许多错误：set()和reset()作用只有被Command Widget定义，但xman有数种其他类型的Widget可接受转换，且转换管理器会抱怨这些 Widget并未提供set()和reset()。解决的办法为更完整些的指定资源名称，例如在本例为 xman*Command*translations 。

- 对任何给定的资源，当资源数据库被询问时，资源管理器会传回一个值给 Widget。这个传回的值的“特征值”(资源名称)大多与Widget的和属性的完整类/实例名称相符。所以你对所有的Text Widget指定一个一般性的转换后，又对xedit指定一个转换，希望它们并存是不可能的，只有一个转换表会传给Widget。例如：

```
*Text*Translation: #override\
(对Text一般性的转换)
...
 xedit*Text*Translation: #override\
(对xedit 的Text特定的转换)
...
```

你只能得到在xedit 中特定的转换，或是在别处得到一般性的转换。

#override 会有所混淆，它的意义为“把转换加入现存之中”。但这完全由转换管理器处理，当时候到时，转换管理器会决定传递哪个值给由资源管理器所造的Widget。对资源管理器而言，#override只是传递给Widget值的部分中的一个文字字符串而已。

因为你使用资源来指定转换，所以错误可能在两个领域均会发生。为了减少错误的范围，当你对转换颇有经验时，在你已加载转换资源之后，最好能明确地打输出你的资源数据库。例如：如果你对xprog 写入转换，且转换在文件mytrans 中，以下列命令来执行程序：

```
xrdb mytrans ; xrdb -q ; xprog ...
```

29.6 键盘和鼠标——对应和参数

在前节我们看到了工具箱所提供的转换结构，它让你对于一个应用程序的个别实例，定制你的键盘和鼠标。在本节，我们来看另一种较低层次的定制，它是由服务程序管理，称之为映射，你只需要告诉服务程序你的键盘所需的不同的配置，它就会被每一个连接到你服务器上的应用程序应用到。例如：替换通常的 QWERTY键盘，你可能希望重新安排键盘以适应那些对键盘并不熟悉的用户 (你可能把键盘按ABCDEF..）重新排过，当然键盘按钮上所印的字也需更改成相符)。你也能对一些Control、Shift等等的修饰键作指定。对鼠标按钮，一样有一个相关的映射，可将“逻辑的”按钮映射到实际动作。总而言之，你使用这些键盘和鼠标的映射的频率，将小于转换。

此外，尚有非常常用的第三种类型的定制可用：你可以设定有关你键盘和鼠标各种不同的参数。例如响铃声音的大小，按下键时是否有滴答声等等。

29.6.1 键盘和鼠标映射—— xmodmap

服务器本身处理一个层次的定制，它对于所有使用到此服务器或显示器的应用程式均发生

效用：这就是键盘映射。

每一个键，有一个单独的码映射它，称之为键码。键和键码之间的关系是绝对固定的 (粗略来说，你可以说“键码就是键”)。

连接到每一个键码 (或键)的是一个keysym的列表。一个keysym是一个代表印在键盘符号上的数字常数。在缺省的情况，大多数的键只有一个 keysym 与之映射，例如 Shift、A、B、Delete、Linefeed等等。keysym既非ASCII或EBCDIC字符，也非服务器用以维持keysym和字符的关系。你可以对每一个键有两个 keysym。在缺省映射中，有很多连接到两个 keysym的键，例如冒号(:) 和分号(;)，7 和 &等等。对一个键附属的keysym列表中，第一个keysym是未按下修饰键的状况下的键。第二个keysym是指当Shift(或Lock)已被同时按下时的键，如果在列表中只有一项，且为字母，则系统自动假设第二项为相对的大写字母。超过两项的 keysym并没有特别的意义，键盘和keysym之间的关系被称之为键盘映射。

应当尽量地使用服务程序处理一般的键和 keysyms 。它对键码没有附属意义，且它自己本身不会使用映射从键码映射至keysyms ：它只是传递信息给客户应用程序。特别的是，服务程序对ASCII 或其他的字符集合毫无概念；它只是说明“某键被按下，某修饰键也同时被按下，keysym列表中某keysym和某键相关”。它是客户机程序 (典型的使用标准的 X Library) 对keysym和修饰键附属的意义：例如，它决定如果 keysym产生时ctrl也被按下，它必需被解释为ASCII 字符 hex 0x1，也就是说Ctrl-A。特定的客户机可以决定特殊的修饰键的意义；例如在xterm 中，当你和MTEA键同时按下一个键，程序将此转换为ESC 后面跟随着被按下的字符(也就是说，如果你按下meta-A，实际上会产生两个字符 ASCII 0x1b, ASCII 0x41)。

服务程序在此领域内提供一个额外的设施。你可以定义让服务程序将键码解释成修饰键，例如“当键码为若干的键被按下时，它相同于 CONTROL修饰键被实际按下”。这种定义并不互斥：如果你定义键F7为Shift 修饰键，它并不会影响任何现存的修饰键。此种设施称之为修饰键映射。 X提供八个修饰键： Shift、Lock(caps-lock)、Control、Mod1到Mod5。习惯上，Mod1被解释为Meta。

最后，对鼠标按钮有一个类似的鼠标指针的映射。对每一个实际的按钮，你可以对它们指定一个相关的逻辑按钮数字。

实际上，如果你改变你的键盘或鼠标的映射，你相当于是说制造厂商对你的输入设备配置不当，你将把它修正为适合你所需要的。当然，如果你改变了映射，你应该把映射键上面所印的符号也随之修改；不过，通常更改的都是一些控制和修饰键，所以就不是那么需要了。换句话说，如果你改变了映射，使得键盘配置和一个特定国家标准 (例如：法国或德国) 相符，你必需更换实际键盘上的符号。

你可以想象得到，改变键盘映射是一件相当稀罕的事，你可能设定它一次之后就不再改变它。在以下几节，我们将很快的看一看如何使用程序 xmodmap，查看现有的映射和修改它们。

1. 查看现有的映射

你使用xmodmap 来列出现有的映射，就如同改变它们一样。你可以指定不同的命令行选项，来选择想要输出的不同的映射：

- 列出现有键的映射：指定 -pk选项。
- 列出现有修饰键的映射：指定 -pm选项 (或是什么选项也不选，因为这是xmodmap的缺省作用)。
- 列出现有鼠标指针 (按钮)的映射：指定 -pp选项。

例如，将所有的映射一起输出，使用命令：

```
xmodmap -pm -pk -pp
```

当xmodmap 用来改变或设定映射，它可以处理一或多个表达式的作用。你可以把这些输入在一个文件中，假设此文件名称叫myfile，可用下列命令两者之一：

```
xmodmap myfile
xmodmap - <myfile
```

第二行的短横线是必需的，如果少了它，程序将只完成缺省的作用 (列出修饰键的映射) 。除了在文件中输入规范之外，你也可以在命令行中用 -e 选项直接指定它们：

```
xmodmap -e expression
xmodmap -e expression-1  -e expression-2
```

为了得到更多有关xmodmap 作用的信息，可以指定冗赘选项，-v 或-verbose。你可以借助于使用 -n 选项不实际的改变映射而获得相同的打印输出 。(此功能和UNIX中make命令的 -n 选项相同，其意为“ 假装执行我要求你做的事，正确告诉我你将如何进行，但并不实际完成作用 ”)。这个选项对新手或不确定自己是否做的正确的情况非常有用。

每一个表达式的语法并不相同，但一般性的格式为：

```
keyword target = value(s)
```

(等号的两边均需为空白)。

2. 改变鼠标指针的映射

鼠标指针的映射是一个逻辑按钮数字的表。 (逻辑的button-1我们称为LEFT，逻辑的button-2 称为MIDDLE等等，实际的 button-1 是鼠标左边的按钮，button-2是隔壁的按钮等等，所以缺省的逻辑的按钮和实际的一致。)在表中的第一个项目是逻辑的按钮和实际的 button-1 的关系，下一个则是对实际的 button-2的关系，以此类推。例如，颠倒按钮的次序，使用命令：

```
xmodmap -e "pointer = 3 2 1"
```

结果按下鼠标右边的按钮，会被解释成LEFT。

3. 改变键映射

xmodmap 让你将一个键 (也就是说键码) 链接到一个新的keysym表，使用表达式：

```
keycode keycode = keysym-1 [keysym-2 ...]
```

安排keysym-1链接到键时没有修饰键，当Shift按下时keysym-2链接到键，如果还有下一个keysym的话，对keycode 而言是第三顺位等等。(请记住，在前两个之后的keysym，系统并未附属特别的意义，应用程序如果需要的话可以附属意义)。

让我们举一个实际的例子。一些键盘把一些非字母数字键放在不标准的地方，所以我们假设你要将 F6键重定义当没有修饰键按下时为“ 9 ”，当Shift 按下时为“ (”。要写入这个xmodmap 的表达式，你需要知道三件事： F6的键码和“ 9 ”与“ (”的keysym。我们在第１２节提到过，执行xev 便可获得这些：分别按下“ F6 ”,“ 9 ”,“ (”三个键，你便可得到它们的键码和keysym。然后将它们放入你的表达式中。例如在我们的系统中我们使用命令：

```
xmodmap -e "keycode 21 = 9 parenleft"
```

为了容易一些，你通常不需要查问键码， xmodmap 允许你使用下列格式：

```
keysym target-keysym = keysym-1 [keysym-2 ...]
```

它的意义为“ 附属在此键的keysym列表现在改由target-keysym来附属 ”。例如针对我们方才的样例，我们可以用：

```
xmodmap -e "keysym F6 = 9 parenleft"
```

如果将相同的 keysym附属到数个键， xmodmap 会搞混掉，像这种情况你应坚持使用keycode 这种符号表示法。

4. 改变修饰键映射

在服务器中修饰键映射是一个表的集合，每个修饰键有一个表。对一个修饰键的表中，包含了所有当此修饰键被按下时会有意义的键 (键码)。xmodmap允许你在一个表中增加项目，去除项目，或完全清除一个表。对此三个操作的格式为：

```
add modifier = list-of-keysyms
remove modifier = list-of-keysyms
clear modifier
```

不幸的是，语法有点儿混淆，因为替换你所需的键码，你必需指定 keysym附属到键码。

举一个例子：假如你需要在你键盘的右边有一个第二个的 Ctrl键。在我们的键盘上有一个 Alternate 键没有被用来做任何事，所以我们将修改它，命令为：

```
xmodmap -e "add Control = Alt_R"
```

为了多解释一些情况，让我们假设你没有一个多余的键，但有一个第二个的 Meta键在键盘的右手边，而我们要用它。我们首先必需去除它的Mod1映射 (你必须使用Mod1，Meta没有用)，而后将它加入 Control 映射。(如果有需要的话，我们可以拥有双重的映射，所以在 Control-Meta组合键时才会有作用，在一些编辑器中常会用到)。命令为：

```
remove Mod1 = Meta_r
add Control = Meta_r
```

将上述命令行放入一个比方说叫mymaps的文件中，执行命令xmodmap mymaps。它可以工作，但如果你用xmodmap -pm 去查看，你会发觉Control 和Meta混合在一起，所以最好改变键上的keysym为：

```
remove Mod1 = Meta_R
add Control = Meta_R
keysym Meta_R = Control_R
```

在xmodmap 的联机帮助中，有几个更多的交换修饰键的样例。

注意 当增加一个键到修饰键映射，keysym只是用来指定xmodmap 中的键。它完全是 xmodmap 本地的，且只是一个符号而已：只有当相关的键码传递到服务器，才实际上的改变映射。同样地，keysym 和 keycode 表达式对修饰键映射绝对没有影响。一个常见的错误是执行下面这个命令：

```
xmodmap -e "keysym F1 = Contrl_R"
```

这个命令期望F1键能像一个control 键般作用，但它不会。因为你相当于告诉系统“我已经把这个符号印在F1键上面”而已。你应该这样作：

```
xmodmap -e "add Control = F1"
```

如果你合并上一行的命令会使得映射表行看起来清楚些。

我们对不同映射的处理的描述到此结束。

29.6.2 键盘和鼠标参数设定——xset

最后我们来看一看对键盘、鼠标和屏幕设定不同的参数。这些参数使用 xset程序 (我们曾经用来控制服务程序的字体搜索路径) 来设定。在以下的叙述中，我们只用一组参数来演示 xset，但你可以同时指定多组不同定义的设定。

1. 控制终端机响铃

用xset你可以让铃声响或不响，设定它的音调和它持续的时间 (假设你的机器提供这些操作)：

让铃声不响	xset -b
	xset b off
让铃声能响	xset b
	xset b on
设定铃声的音量	xset b vol
(最大音量的vol%)	例：xset b 50
设定铃声的音量和音调(单位Hz)	xset b vol p
	例：xset b 50 300
设定铃声的音量，音调	xset b vol p d
和持续的时间(单位百万	例：xset b 50 300 100
分之一秒)	
控制键的滴答(click)	
让键的滴答不作用	xset -c
	xset c off
让键的滴答作用	xset c
	xset c on
设定滴答声的音量	xset c vol
(最大音量之vol%)	例：xset c 50
控制键的自动重复	
让键的自动重复不作用	xset -r
	xset r off
让键的自动重复作用	xset r
	xset r on

2. 鼠标参数

鼠标指针在屏幕上的移动和鼠标的移动是成比例的。加速值是应用在鼠标指针移动上的一个乘数，例如如果加速值是 4，当你移动鼠标时，鼠标指针将以正常 4倍的速度移动(如果鼠标指针正常时移动n 个像素，现在则会移动 4×n个像素)。

当你希望在屏幕上将鼠标指针移动一段长距离时，相当高的加速值是很实用的，但当你要作一些细部的伪指令时，它看起来就很笨拙 —— 鼠标指针看起来在来回跳动。

为了克服这点，服务程序提供了一个阈值：如果当鼠标指针一次移动超过阈值个像素，加速值也会被带进来执行。

设定鼠标的加速值到a	xset m a
	例：xset m 5
设定加速值为a，设定阈值到t	xset m a t
	例：xset m 5 10

3. 控制屏幕保护程序结构

屏幕保护程序是一种设施，它的目的是降低一个固定的图案老是在屏幕上出现的机率。它的原理是屏幕损害大都起因于让系统闲置很长时间，所以屏幕保护程序在一段特定的时间内如果没有输入动作后，不是整体性地闪动屏幕，便是显示一个不同的图案。

如果你选择的是显示一个不同的图案，根窗口的背景涵盖整个屏幕，一个大 X 的光标出现在屏幕上，且会移动。当大 X 光标在移动时，会改变大小，而且背景也会随机的变动。 (在背

景图案较小时你可能不会注意到，但若比较大时，你可以看到它在跳动。)

当屏幕保护程序结束作用后，如果要花许多时间才能重画应用窗口，你可以指定只有在重画屏幕而不需产生任何窗口暴露事件 (也就是不必要求应用程序重画它们自己的窗口) 的情况下，屏幕保护程序才会作用。这只应用于显示不同的图案的情况，整体性的闪动屏幕纯为硬件作用，不会影响到应用程序。

让屏幕保护程序能作用	xset s
让屏幕保护程序不能作用	xset s off
用屏幕闪动的方式	xset s blank
只有在无曝光事件下才作用	xset s noexpose
允许有曝光事件下仍然作用	xset s expose
用不同图案的方式	xset s noblank
当系统闲置t 秒后作用	xset s t
	例：xset s 600
每p 秒之后改变图案	xset s t p
	例：xset s 600 10

让我们将这些组合起来，假设我们希望屏幕保护程序在系统闲置 80秒后开始执行，用不同的图案的方式，会话为3 秒，不介意曝光事件是否发生：

```
xset s noblank s 80 3 s expose
```

注意 xset s 并不提供 on 这个值。

29.7 进一步介绍和定制uwm

在27.6节，你学到如何使用uwm 来完成基础的窗口配置工作需求，而能以一个舒服的方式使用窗口。现在我们继续谈窗口，集中于两个主要的范围：

1) 提供特别功能，如：

- 不使用菜单，直接使用鼠标配置窗口。
- 我们尚未描述过的一些菜单选择。
- 编辑现存图标的标题。

2) 定制uwm，包含：

- 对任何你所需的命令定义你自己的菜单。
- 将各种不同的窗口管理器功能对应到鼠标按钮和修饰键 (Shift、Control等等)。

29.7.1 uwm 的新特征

现在我们来讨论一些在先前介绍窗口管理器时，为了保持尽量地简单，而省略的标准的uwm 功能。

1. 不使用uwm 的菜单管理窗口

目前，你仍然依赖着uwm 的菜单来配置你的窗口 —— 移动它们、对它们重定大小等等。如果所有的情况都使用菜单，是相当慢的，所以uwm 提供直接完成它的命令选项。

可以使用鼠标按钮和修饰键，来指定要执行的功能和所要操作的窗口。你现在应该已非常熟悉各种不同的窗口管理器功能和它们如何工作，所以我们将很快地说明如何不使用菜单来选择这些功能。

(1) 移动一个窗口

1) 按下Meta键，保持按住。

2) 鼠标指针位置所在的窗口将被移动。

3) 用右按钮，拖动窗口到新的位置。

(2) 重定一个窗口的大小

1) 按下Meta键，保持按住。

2) 鼠标指针位置所在的窗口将被重定大小。

3) 用中按钮，拖动窗口的外框到新的大小。

(3) 将一个窗口送到堆栈的底部

1) 按下Meta键，保持按住。

2) 将欲被送到堆栈的底部的窗口，按一下左按钮。

(4) 将一个窗口升到堆栈的顶端。

1) 按下Meta键，保持按住。

2) 将欲被送到堆栈的顶端的窗口，按一下右按钮。

(5) 将最底层被遮蔽的窗口升到最上层，你有两种选择：

1) 按下Meta键，保持按住。

2) 在根窗口上，按一下右按钮。

或

1) 同时按下Meta和Shift 键，保持按住。

2) 在屏幕上的任何地方，按一下右按钮。

(6) 将最上层的窗口移到最底层：

作法同Circulate Up，但改为左按钮。

(7) 图标化一个新的窗口

1) 按下Meta键，保持按住。

2) 将鼠标指针位置移至欲被图标化的窗口。

3) 按下Left按钮，保持按住。

4） 拖动图标的外框到你所需的位置。

5) 释放按钮和Meta键。

注意它和Lower 操作程序的不同点，在此你是按下、拖动、释放鼠标按钮，而对 Lower，你只是按一下按钮。

(8) 图标化一个曾经图标化过的窗口

1) 同时按下Meta和Ctrl键，保持按住。

2) 在你欲图标化的窗口上，按一下左按钮。

如果你对先前并未图标化的窗口执行这个操作，或经由资源结构无法取得图标的位置，图标将出现在鼠标指针所在的位置。

(9) 将图标还原为它的窗口 (在窗口原来的位置)：

1) 按下Meta键，保持按住。

2) 在图标上，按一下中按钮。

如果你觉得这些对鼠标按钮功能的结合十分笨拙且不易记忆，别担心，很多人都是这样。有更好的法子，刚才那些只是缺省的设定，你可以完全由自己来配置。在本节的后半部，我们将告诉你如何做。现在我们先来看一看，在标准菜单的一些功能和它们能做些什么。

2. 更多的菜单选择

这是一些我们在27.6节中没有解释的标准的菜单选择。

(1) 让你设定键盘的焦点

也就是说，将键盘附属于一个窗口，所以不论屏幕上的鼠标指针在何处，键盘的输入总是在同一个窗口。一般键盘的输入总是指向目前鼠标指针所在的窗口。

(2) 设定焦点到一个特定的窗口

选择focus，出现手指形光标，在你所欲指定的窗口按一下按钮。

(3) 恢复正常

选择focus，在背景窗口上按一下。

(4) 停止uwm，重新启动它

重新读入配置文件（下节说明）且执行它。在你改变配置文件且希望马上执行新的设定时(否则将等到你重新启动一个新的会话) 使用此选择。

(5) 暂停屏幕上所有的显示

当你要对你的屏幕拍照时可以使用这个选择。

(6) 重新恢复显示

所有的窗口会立即更新。

(7) 中止uwm

当你要删除uwm 时使用，例如在启动一个不同的窗口管理器之前。

(8) Perferences 菜单

我们在上面提过，有两种方法激活 uwm 的WindowOps菜单，在背景窗口上按下中按钮，或在按住Meta和Shift两个键的情况下，在任何地方按一下中按钮。用第二种方法让你调出第二个菜单，只要将鼠标指针移到WindowOps菜单的外边，标头为Preferences 的窗口就会出现。

在Preference中的选择，只是一些xset程序中设定鼠标和键盘的选项而已。

注意 Lock On 和 Lock Off选择是和记录有关的，可能会导致在你的主控台窗口输出一个错误的信息。

3. 改变现存图标上的标题

如果你对同一个应用程序执行数次拷贝后会有缺点，例如有三个 xterm 的图标，你无法明确的区分它们。为了克服这点， uwm 允许你可以编辑图标中的字符串为你所需的任何字符串。(这只能在uwm 自己缺省的图标使用，例如你无法编辑在xclock的特定图标中的字符串。)

编辑在一个图标中的名称方法如下：

1) 将鼠标指针移至所要编辑的图标。

2) 键入你所希望的任何文字。

3) 你可以去掉文字，不论是先前存在或方才才输入的，方法如下：

• 去掉前一个字符：按Delete。

• 去掉整个名称：按Ctrl-U。

29.7.2 定制uwm

uwm 具有高度的可配置性。你可以将整个范围的参数和定义保存在一个配置文件中，当 uwm 启动时会将之读入。我们前节曾经提过，你可以在中途改变配置文件，用 WindowOps菜单中Restart选项，告诉uwm 重新读入它。

1. uwm 的配置文件

缺省uwm 有两个配置文件，其中之一为：

/usr/lib/X11/uwm/system.uwmrc

通常由系统管理员设定，且第一个被读入。另一个

$HOME/.uwmrc

是你自己的配置文件。两个文件均需要存在， uwm 硬性规定了缺省设定。

注意 如果你用不正确的语法设定一个配置文件，当uwm 读入时，你会得到一个错误信息，像：

```
uwm: /usr/nmm/.uwmrc: 38: syntax error
uwm: Bad .uwmrc file...aborting
```

uwm 将不会启动。当在一个新的会话启动时，这没有什么大问题。然而，如果你在中途重新设定uwm ，你可以结束但没有窗口管理器，且没有 xterm ，没有编辑窗口来编辑这个错误的文件，无法启动其他的窗口。如果此种情况发生，你必需从其他的终端机或机器关闭 X，或毁坏你的系统。

2. uwm 的命令行选项

如果你不需要系统配置文件，也不需要任何缺省的设定，你可以借助于 uwm 的命令行选项 -b 抑制它们。

如果你要使用其他的文件，就像两个缺省的配置文件一般，你可以用 -f filename 来指定它。

3. 把功能对应到键和按钮

uwm 让你定义当一个特定的鼠标按钮按下时，有某个功能会作用。例如当你在一个窗口中按一下中按钮，它将被升到堆栈的顶层。这种对应结构和工具箱转换并没有牵连，它完全由 uwm 本身来完成。

为了让这些结构更有用，你可以指定其他的条件来运用更多的功能，或许一个修饰键 (像 Meta) 需被按下，或许作用只发生在鼠标指针位于一个图标上而非应用程序窗口或背景窗口。

用.uwmrc(或其他的配置文件)所包含的对应规范来指定对应。规范的格式和上面的表格类似，就像：

```
uwm-function = modifiers : window context : mouse events
( uwm 功能  =  修饰键  : 窗口的环境:  鼠标事件  )
```

这些元素为：

- uwm 功能：uwm 的内建功能之一的名称。例如功能 f.move即是移动窗口的功能， f.lower 将窗口降低一层等等。这些功能将在下面更完整地描述。

功能名称必需跟随着一个等号(=)。

- 修饰键：在运用上述功能时，当指定的鼠标事件发生时，必需被按下的修饰键表列。正确的修饰键名称为：

  ctrl(或c)，对Control键。

  meta(或m或mod1)，对Meta键。

  shift(或s)，对Shift键。

  lock(或l)，对CapsLock键。

这些名称必需正确地列出。你可以使用 1～2个修饰键，如果你使用两个键，用一个“ |”符号来分开它们。

你可以省略整个修饰键列表 (即此功能映射于鼠标事件发生时并没有修饰键被按下), 但尾端的冒号“ : ”不可省略。

- 窗口的环境：限制只有鼠标指针在屏幕上指定位置的类型符合特定条件时，功能才会发生。正确的环境如下：

window(或w)：鼠标指针必需位于一个应用窗口中。

icon(或i)：鼠标指针必需位于一个图标中。

root(或r)：鼠标指针必需位于根窗口或背景窗口中。

你可以指定任何数目的环境，用“ | ”来区隔它们。如果你没有指定，则功能的发生与鼠标指针位置无关。

- 鼠标事件：何种鼠标事件映射到此功能。指定的事件为一个按钮名称 —— 任何的

left(或l)。

middle(或m)。

right(或r)。

跟随着一个动作：

down : 当按钮被按下时会符合。

up : 当按钮被释放时会符合。

delta : 当按钮被按下且移动超过一定数目的像素时会符合。

所有的这些你已实际使用过它们，在本节开头所描述的一些作用的对应为：

```
f.resize = meta : window : middle delta
f.iconify = meta : icon : middle up
f.raise = meta : window|icon : right down
```

uwm 的缺省对应在文件 $TOP/clients/uwm/default.uwmrc。

4. uwm 的内建功能

uwm 的联机帮助列出可应用的功能。你可以看出，功能是和 WindowOps及Preferences中的选项相关。

然而，有一个有关 pushing窗口(f.pushleft, f.pushup等等)的功能集合你从未见过。pushing的意思为，朝一个特定的方向移动一个窗口，移动的距离固定。与 f.move不同的是，后者以交互的方式指定窗口移动的方向和距离。

缺省f.pushdown对应到同时按下Control和Meta键，且按住中按钮。试它几次，你将发现你的窗口稍微移动了一点 。push 功能对微小移动窗口非常有用。

另一个功能为f.moveopaque。它也移动一个窗口，但不像 f.move，它并不会给你一个指示窗口新的位置的方格，你直接拖动整个窗口本身。这可以让整个屏幕清爽些，但比较慢，且一般窗口移动时会有抖动的现象。

5. 定义菜单

f.menu是一个非常强大的uwm 的功能：它允许定义自己的菜单。此菜单可选用到 uwm 本身的功能，或任何的外壳命令，或一个特定的动作，像是在一个剪缓冲区插入文本。

在配置文件中定义一个菜单共有两个步骤。首先定义菜单上所需的对应，其次定义菜单本身的内容。对应的部分像我们先前所用过的，但在尾端增加了一栏菜单名称。例如 WindowOps 菜单 (借助于在背景窗口中按下中按钮来呼叫)的对应是：

```
f.menu = : root : middle down : "WindowOps"
```

在此，菜单名称既是用以显示菜单出现时的名称，也链接到配置文件中的菜单内容规范。

菜单内容的格式很简单：对每一个选择项，包含了一行当选择项出现在选单的名称和当它

被选择到时所做的动作。让我们观察一个省略的WindowOps定义：

```
menu = "WindowOps" {
New Window : !"xterm &"
RefreshScreen : f.refresh
Redraw : f.redraw
Move : f.move
}
```

从这里，我们可以看到其语法为：

```
menu = "menu name" {
...
selection lines
...
}
```

菜单名称和对应所指定的相同。选择项列包含了选择项名称，分隔的冒号和负责的动作。这些动作为下列三者之一：

1）一个uwm 的功能：只用到它们的名称，在上例为move那一行。

2）一个外壳命令：命令包含在双引号中间 (用外壳的& 语法使其在背景窗口中执行) 且在前面加一个惊叹号。在上例为 xterm 那一行。(如果省略& ， uwm 将被挂起来，等待命令的完成，如果此程序为X的应用程序，它需要uwm 来安排它的窗口，这将会招致麻烦。)

3）一个文本字符串：这将插入到一个“ 剪 ”的缓冲区，而后你可以像平常一样的“ 贴 ”它。

(1) 多种的菜单链接对应到同一个键

通常对一个特定的键 /按钮的组合，只会对应到一个菜单，但可以对同一个键对应有多种菜单：如果在一个菜单中不选择任何项目且把鼠标指针移动到菜单的边上，将得到下一个菜单。你已经实际看过这种例子：在同时按下 Meta和Shift键的情况下按下中按钮，可以得到WindowOps 菜单，然后是Preferences菜单。

对应多种菜单非常容易，只要在定义每一个对应时当作其他的对应并不存在，而在定义菜单的内容时用标准的方式即可。例如uwm 的缺省设定包含了对应：

```
f.menu = meta | shift : : middle down : "WindowOps"
f.menu = meta | shift : : middle down : "Preferences"
```

注意 一个菜单只能定义一次，但可以用它来做任意多次的对应。查看缺省设定，将看到WindowOps 菜单被定义了一次但使用到两次。

(2) 指定菜单的颜色

你可以指定在一个菜单中所用的颜色。对菜单名称标题、每一个选择项、鼠标指针所在的高亮度显示选择项，你都可以指定一个前景色和背景颜色。一个有颜色的菜单的格式如下：

```
menu = "menu name" (head-fg : head-bg : hilite-bg : hilite-fg) {
...
selection-name : (item-fg : item-bg) : action
...
}
```

以下为一个混合的样例，使你的WindowOps 能拥有更多的颜色：

```
menu = "WindowOps" (yellow : blue : red : green) {
New Window : !"xterm &"
RefreshScreen : f.refresh
Redraw : (navy : magenta) : f.redraw
```

```
Move : f.move
}
```

此菜单标题为蓝底黄字，大多数的选择项为白底黑字 (缺省值)，只有Move选择项为紫红色底海蓝色字，而目前鼠标指针所在的选择项为绿底红字。

6. 控制uwm 的参数变量

到目前为止，你可以用指定鼠标和键的前后关系，来改变所指定的功能。有一个另一种类型的uwm 的定制可以改变许多内建功能操作的模式和样式，例如可以指定在 resize或move操作下，指示窗口新的位置的九字格，改变为只是一个外框而已。在联机帮助中列出所有的变量和它的意义，在此我们只介绍一些特别有用的和比较模糊的类型。

• 让缺省配置文件中的设定无效。

uwm 并没有结构抑制读取系统和用户配置文件。(-b 不会影响 $HOME/.uwmrc。)欲取消早先文件中的设定，可以含入 uwm的变量resetbinding、resetmenus和resetvariables，将会分别取消早先定义的对应、菜单和变量。(确保你将这些变量放在文件的顶端，否则它将取消在文件中所有在它之前的定义。)

• 限制窗口和图标在屏幕范围以内。

X允许你指定窗口位于屏幕的任何位置，甚至部分或全部在屏幕之外，这样有时会引起麻烦。当你建立一个窗口， uwm 并未提供任何帮助。但当你使用 f.newiconify对一个图标作解除图标化，如果变量normalw 被设定，则窗口会被完整地放在屏幕中，且尽量接近你用鼠标指针指定的位置。(如果你包含了normali 变量，同样可用于图标。)

• 控制push作用。

缺省f.pushxxx 功能将一个窗口往适当的方向推动一个像素的距离。可以指定 push=num 来推动num 个像素。也可以完全地改变操作的作法：不用通常的推动固定数目的像素的作法 (叫做pushabsolute)，可以指定pushrelative，这种情况窗口会被推动num 分之一大小的窗口。例如如果你指定：

```
push=5
pushrelative
```

则一个f.pushup将把窗口向上推动窗口本身高度五分之一的距离。

• 防止uwm 功能锁定应用程序。

一些缺省 uwm 的操作，像重新定义大小和移动会导致所有其他的客户程序被冻结，也就是说，防止它们输出到它们的窗口。可以用 nofreeze取消它。

如果你需要获得一些 uwm 所属窗口的打印，则这是必需的。它的副作用为当使用重新定义大小和移动时，外框格会大量地闪动，以致难以看到。

29.8 显示器管理器——xdm

现在我们已经讲述了有关 X的所有单个项目，例如，如何启动系统，如何设定一个窗口管理器的执行，如何执行应用程序，如何从不同的角度定制系统，如何退出系统。

本节中，我们把这些分开的部分放在一起，且描述一个完整的文件设定，用来定制涵盖所有的系统机器环境。本节将介绍最后一个 X工具：显示管理器 —— xdm，它提供一个启动X的精巧和清楚的方法。

29.8.1 需要做些什么

当我们启动X之后，我们需要安排屏幕，让一些在整个执行期间中都会使用的应用程序打

开，让一些偶然用到的应用程序以图标为开始时的表示方式。我们需要执行窗口管理器，对某些种类的功能做一些设定。详细来说，我们需要下列的程序：

- 一个xterm 的主控台，在屏幕左上角。
- uwm 在后台执行。
- 一个我们的（正常）编辑器的全屏幕xterm 窗口，以图标方式启动。
- 在右上角一个（较一般为小）的时钟。
- xbiff 在时钟之下。
- 一个计算器在右下角。
- 一个用到我们所有最小的字体的图标化的xterm ，它的高度比屏幕高。
- 在xbiff 之下，排列我们使用远程机器的频率图。

除了程序之外的项目：

- 设定背景窗口为亮灰色。
- 启动键盘输入功能。
- 从我们常用的网络主机访问到我们的服务器。
- 加载我们对所有客户程序用到的服务程序设定的资源，在 29.4节定义的 $HOME/.Xresource 文件中，根窗口RESOURCE_MANAGER属性之上。
- 启动一个屏幕保护程序。

并且需要uwm 有菜单以便：

- 容易地访问在网络上其他的主机。
- 更改一些键盘和鼠标的设定，且设定背景窗口的颜色。
- 启动那些我们偶而会用到的应用程序。
- 启动一些被选定的示例程序。

对这些我们自己的设定，在网络上其他的用户需要不同的初始设定，所以需要安排每一个用户依自己的喜好设定。理想上，用户应能自行设定而不需要系统管理员的帮助，下一节我们来看程序xdm 如何能帮助我们完成这些目的。

29.8.2 xdm

xdm 为X 显示管理器。xdm 管理一个或多个显示器，xdm 可在同一机器或远程的机器上执行。它可以做到所有xinit 能做到的功能，而且更多。它所隐含的概念为它应控制当你在 X工作时的会话，也就是从进入直到结束窗口系统的会话。 (用xinit ，有效会话为当执行xinit 开始，到注销最初的xterm窗口和关闭服务程序。)

xdm 较这更进一步，可以用它执行一个不确定的会话。当一个结束，下一个便准备开始。实际上，如果有需要，它可以不变地指定一个显示器。

xdm可完全替换xinit 。从现在起你可忘掉xinit，而且不再需要使用它，我们在最初使用xinit 的原因为它较易观察和了解系统的运作。

xdm 是一个非常灵活的程序，几乎可用它配置任何所需要的功能，在进一步深入之前，让我们观察一个样例会话的缺省行为，然后我们来看一看如何改进当一个用户进入 X系统所看到的初始界面。

1. 用xdm 的例子会话

我们将使用xdm 来设定X。机器已经启动，但尚未有窗口系统在其上执行。用下列的命令启动xdm：

```
xdm
```

xdm 开始执行，几乎立刻又看到外壳提示。然后屏幕背景更改为通常灰色形式，且看到一个大的X光标，所以知道服务程序已经启动。

接下来是一长段修止状态(大约持续15秒或更久)，而后，一个带着欢迎标题的窗口出现了，要求你输入登录名称和密码。输入你的用户名称和密码后，又过了一会儿，你可以看到一个xterm 窗口在左上角出现，以后的工作的方式和以前叙述的相同 —— 启动窗口管理器，执行应用程序等等。

需要结束时，也可用以前相同的方式结束： logout（注销）最初始的 xterm窗口。但这里xdm 和xinit 有不相同的地方，关掉服务程序的方法是，回到非 X的环境，屏幕回到最初始灰色的背景，过一会之后，又再度看到X的登录窗口。事实上，xdm 是执行一个循环的会话。

注意 就像许多的UNIX程序一样，最大的登录名称长度为8 个字符，如果超过这个长度，登录将会失败。（如果实际登录程序允许使用较长的名称，这种限制也许让人感到奇怪。）

2. 关闭xdm

有时可能需要完全地关闭X。为了做到这点，需要关闭xdm 。

在MIT 版中的服务程序，如果收到UNIX信号SIGTERM ，便会执行中止程序。xdm 利用到这点，如果送给它一个 SIGTERM ，它将中止所有它所控制的服务程序后离开。这就是中止系统的方法。

要实际中止xdm，可以在一个xterm 窗口（在你的机器上）用 ps来找出xdm 的process-id，而后用kill送给它SIGTERM。例如，在我们的机器上执行中止的动作如下：

```
venus% ps ax 1 grep xdm
1997 ? IW 0:00 xdm
1998 ? IW 0:00 xdm
2000 ? IW 0:00 xdm
2078 p0 S 0:00 grep xdm
venus% kill -TERM 1997
```

所有的应用程序将被强迫中止，服务程序也随之关闭。

注意 当相关于X的每一件事都结束后，屏幕可能只显示通常X背景的灰色形式，没有任何的shell 提示或任何事。此时，你的shell 已准备好接受你的命令，按下Return键将会看到。因为在以交互式下xdm 命令之后，外壳已将提示信号送出，所以不再重复,除非你按下Return键。

29.8.3 xdm的更多信息

我们在前所述的是xdm 的缺省模式下的操作，所以看起来并没有比 xinit 提供得更多。如果使用一个正常的工作站或显示器，一些外貌将不是很有趣。无论如何， X终端机是一个日渐增加的大众化设备，而xdm 可大量地简化管理类似的系统。X终端机通常没有它自己的文件系统，且不能支持一般目的的程序，必需在网络的某处执行包含窗口管理器和显示管理器的控制终端机软件，xdm 正是符合此需要的软件。

xdm 在下列这些场合比xinit 好：

- 它可控制数个服务程序，也就是说，其中有一些为远程的服务程序，也许是在 X终端机或相当小的工作站上。

- 它提供密码来存取系统，同样地，在 X终端机上非常有用（但在一个你已经登录的工作站会有一点困扰）。
- 它提供无限期的X的会话。你可以配置显示器总是以X方式操作，所以用户不需要担心如何启动系统。
- 它具有高度的可配置性。系统管理员可以设定根据机器特性启动和结束程序，掌握这些项目以供记帐、授权、文件系统等之用，且能让每一个个别的用户全范围性地修定他们所需的自己的环境。
- 从用户的观点，它提供一个干净而简单的方法来启动系统。

所以大体上，xdm 主要是一个系统管理工具，但它也提供让一个普通用户定制他希望的一致和一贯的系统结构。

xdm 的联机帮助包含了大量的有关如何使用系统的教学信息和伪指令，在此我们不再重复，我们将在以下说明如何正确地配置xdm 以提供在本节一开头所描述的环境。

xdm 非常灵活，且可以用许多不同的方式选择设定，我们将使用最简单的处理，并试着大致和联机帮助的描述保持一致。偶而我们在一些文件中使用不同的名称，用以强调此名称并非硬性的规定。

在我们工作的会话中，请注意我们事实上在扮演两个不同的角色：第一是系统管理员，为使用系统的人设定xdm ，第二是一般的用户，为我们自己的需求设定xdm 。

1. 系统管理员对xdm 的配置

缺省xdm 先查看文件：

/usr/lib/X11/xdm/xdm-config

如果它存在，会把它当成多设定几个其他参数的资源文件。我们将使用它，因为它可简化我们的工作。

联机帮助会列出所有能通过xdm-config文件设定的参数，但与我们紧密相关的参数有：

- 包含一个服务程序的目录的文件名称。
- 当任何错误发生时，xdm 用来记录的文件名称。
- 包含和启动系统有关的文件系统名称。
- 当服务程序启动后执行程序的名称，这个程序定义了你的“会话”——当这个程序中止时，xdm 视其为你的会话已结束，且回到它登录时的顺序，缺省时这个程序为 xterm ，就和使用xinit 一样，在退出你初始的xterm之前，会话将一直持续。

这是我们已在系统上定义的设定：

```
DisplayManager.severs:   /usr/lib/X11/xdm/our-server
DisplayManager.errorLogFile:   /usr/lib/X11/xdm/errors
DisplayManager*resource:   /usr/lib/X11/xdm/our-resource
DisplayManager*session:   /usr/lib/X11/xdm/our-session
```

我们已选择在目录/usr/lib/X11/xdm保持所有xdm 相关的文件，这只是代表名称，可以用任何你喜欢的目录。

所以可以看到我们使用 xdm-config，实际上是两个步骤，首先我们定义在 xdm-config中的一些文件名称，接着我们来设定方才命名的文件。现在我们来看一下我们在 xdm-config中定义的每一个资源。

(1) xdm 的服务程序的名单

这个被DisplayManager.servers设定的文件资源包含了一个 xdm能管理的服务程序的名单。每一行中包含了服务程序的名称（也就是显示器），服务程序的类型和类型有关的项目。

类型指出了显示器是本地的或远程的和是否为无限或单一的会话（详见 xdm 联机帮助）。我们将使用类型localTransient(单一会话在本地显示器上)。因为以此方式，如果发生任何错误，我们不致于陷入无穷循环中。稍后，当我们每件事都设定好且执行无误的话，我们会将类型改为本地而循环的会话。

对本地的显示器而言，和类型相关的信息是在此显示器上执行的服务程序的名称及其任何所需的参数。对远程的显示器，此信息可被忽略，但你仍需输入一个假的程序名称。

所以，在我们所建立的文件/usr/lib/xdm/our-服务程序包含这一行：

```
:0 localTransient   /usr/bin/X11/X  :0
```

如果我们喜欢执行循环会话，此文件便不再需要 —— 缺省设定会做到我们所需要的。所以我们在配置文件中不需定义DisplayManagers.servers resource。

(2) xdm 的错误日志文件

此文件从xdm 和xdm 的会话程序接收所有错误的信息, 且如果xdm 设定工作发生问题的话，这是第一个需要查看的地方。

当你开始设定系统，把此文件设定成任何人均可写入，否则，有问题的程序可能因没有写入权限而无法在文件中记录。

(3) 启动时的资源文件

此文件包含一个资源的名单，在 Authentication Widget 启动之前被xrdb加载。因此，能用它来为那些Widget设定资源。当然也能放置任何其他的资源规范，但通常会话程序的用户设定加载时会凌驾其上，所以通常不把其他的规范放在这里。

authentication Widget 资源的缺省设定在某些情况是很细的，但为了举例，我们只设定和bunner不同的标题，我们建立我们的文件 /usr/bin/X11/xdm/our-resource 包含一行：

```
xlogin.Login.greeting:X-Window on the Plants network
```

(4) xdm 的会话程序

可以指定任何程序为会话中所需的程序。可是当会话结束，通常选择一个程序让你能启动其他的程序。通过xdm 的缺省设定可以执行xterm ，但这种方式每当xterm 执行时你仍必需手动设置所有的设定。我们需要定义会话程序来执行所有的设定，且让会话程序保持活动的状态直到结束为止。但记住，我们希望用户如果需要能定义自己的会话程序，所以我们将使用两阶段的处理。如果是系统管理员，我们将设定一个一般性目的、基础的会话程序来调用一个用户自己的程序（如果它存在）。但其他方面将执行一个合理的缺省值。

基础的会话程序非常简单。如果用户设定有文件$HOME/.Xsession，则可以使用它，否则，将执行合理的缺省值 —— 启动uwm， 而后向一个xterm（xterm为我们指定在屏幕左上方的那一个）传递控制信息。但在这之前，我们先检查是否用户设定了文件 $HOME/.X资源，如果有的话，可以使用xrdb加载它。

2. 用户对于xdm 的配置

现在我们改变角色：我们不再是系统管理员，而是一个用户。我们可以依赖系统管理员已定义的缺省会话，但我们比较喜欢定义自己的会话，所以我们要获取那些说明我们所需的起始设定。

实例 .Xsession

我们已建立自己的$HOME/.Xsession，此程序的操作十分简单。我们假设你的会话程序是一个外壳。虽然不一定如此，但通常都是这样，除非你要写一个 xetrm 的复杂的代替品。

• 文件中的命令依序排列，所以最后一行所执行的是一个程序。它可以在整个会话中持续。

因为，当此程序结束，则会话程序结束，且每一件事也均将结束。

- 除了后台的最后一个命令，所有的命令均被执行。也就是说，在命令行最后加一个 &符号。在实例程序中，如果在uwm那一行省略“ & ”号，uwm 也会启动。但在uwm 结束时，它的下一行将不会继续执行，这样是不对的！
- 最后的命令必为exec命令，所以它继续执行且保持会话继续活动。如果你像其它命令一样在后台中执行它，它会好好地执行，但此会话程序执行至文件结束将会中断，而结束会话。如果你不用 exec，且省略“ & ”号，则它会执行，且此会话将地持续工作。你只是较你所需要的多执行了一个处理，就如同你仍有最后的程序和会话程序本身。
- 对所有的程序建立窗口时设定几何位置规范 ，否则，当它们启动后，你将以“ 手动 ”方式指定它们的位置。
- 在文件中最后一行的程序通常用来启动 xterm ，因为它定义了会话的生命期，在执行X时此窗口总是存在，所以通常设定两个特别的选项：

1) 使用-C选项使得xterm 为一个主控制台，所以系统信息会在此窗口显示。

2) 设定-ls 选项使它的外壳为登录外壳。这样使得外壳读入 .login或.profile文件，所以你的环境变量会适当地设定。

3) 此会话程序文件必需有执行许可。使用上述站点范围的会话程序，这对用户会话脚本不是绝对需要，它实际是对站点范围的程序本身。如果那不能执行，你只能获得 xdm 的缺省设定。

在设置 .Xsession 和依赖它启动窗口会话之前，最好能从一个 .xterm 窗口启动 .Xsession 以便严格测试它。

29.8.4 uwm 配置

我们需要设定四个uwm 菜单：一是连接到其他的主机，二是执行一些 X的应用程序，三是设定一些键盘和鼠标参数（有点儿像缺省的 Preference菜单），四是执行演示程序。

对主机菜单，我们现在希望只要借助于从菜单中选取主机名称便可在任何主机上启动 xterm 。我们常常需在mars上做一些系统管理，所以我们将设定选择为超级用户，我们将在左下角建立一个超级用户窗口。但对一般的 xterm，省略几何位置规范，所以当它建立时，可以明确地定位它。我们将以Meta-Shift-Left 对应此菜单。所以在$HOME/.uwmrc中包含了此行。

uwmrc 剩余的部分可以用来设定定制的对应和一般窗口配置操作的参数。注意下列几点：

- 选择一个较缺省值稍大的字体 (用menufont=fixed)，降低菜单选项中的空白空间 (用vmenupad=1)，所以菜单不会很大(menufont 可能未在联机帮助中描述)。
- 我们设定为可重设所有的菜单、对应和变量。这就清除了 uwm 的配置。
- 如果可能，我们较愿意使用鼠标的UP事件函数而非DOWN事件。这种方式能在释放按钮之前按下其他的按钮，来改变操作或中止操作。
- 我们已包含一些uwm 菜单的功能 —— 一个是删除应用程序窗口，另一个是重新启动uwm 。它们不是必要的，但当你对系统很有经验时会很有用。