面向对象的字符窗口系统的设计

马 亮 (新疆科技情报所)

摘要:图形用户界面(GUI)是一种潮流,但众多的应用系统仍建立在字符的基础之上。本文采用面向对象的设计思想,详细探讨了分层式文本窗口系统的管理机制,并给出主要技术难点算法描述,最后介绍一个与应用接口的实例。

一、窗口系统基础及面向对象的设计思想

1.窗口系统基础

窗口系统一般分堆栈式和分层式两种,我们先来看看描述屏幕各窗口位置关系的几个术语的定义。为了更为形象,以图 1 为例,W1,W2,W3 为所开的三个窗口。

最前端窗口:位于屏幕最前端的窗口(如 W3)。

最底端窗口:位于屏幕最底层的窗口(如 W1)。

前端窗口与底端窗口:相对于屏幕,位于 Wi 前面的窗口为 Wi 的前端窗口,位于 Wi 后面的窗口为 Wi 的底端窗口(如图 1 中 W2、W3 为 W1 之前端窗口,W1、W2 为 W3 之底端窗口)。

当前窗口:当前进行 I/O 等操作的窗口。

窗口背面信息:该窗口从屏幕隐藏后屏幕上其原来所在的矩形区域的内容。

窗口正文信息:该窗口在屏幕上显示时屏幕上其所在的矩形区域的内容。

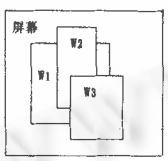


图 1

刷新 Wi:将窗口 Wi 变成最前端窗口的操作。

激活 Wi:将窗口 Wi 变成当前窗口的操作。

序列(W1,…,Wn)用于表示屏幕上由最底端至最前端的 n 个窗口(包含被暂时隐藏的窗口)。在本文中后面的算法中 Wi 还用来代表一个分层式窗口类的对象。

堆栈式窗口系统的当前窗口只能是最前端窗口,而分层式窗口系统的当前窗口则可以为任一窗口。前者不但在算法上比后者容易,而且时间和空间的效率上也比后者高。对于 PC 机来说,一般只实现单任务操作,尽管屏幕上可同时打开多个任务窗口,但当一个任务在执行时,其它任务都被挂起,因而,对于 PC 机软件的界面设计,一般前者即可满足要求。本文将主要针对分层式窗口系统探讨其管理机制,由此可以很容易实现一个堆栈式窗口系统。

2.面向对象的设计思想

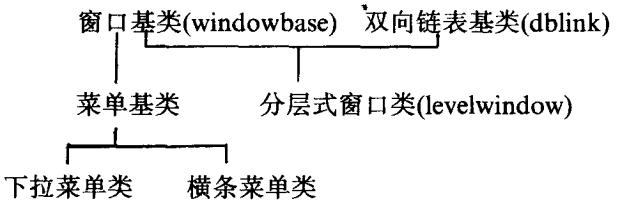
在面向对象的程序设计方法(OOP)中,对待处理的问题域进行划分,成为各自独立的子问题。每个问题中,数据及相应的操作被封装起来成为一个独立的整体。这不仅降低了设计的复杂度,而且使日后维护更为容易。C++ 给出了类(class)的概念,类即是对一组具有相同数据结构和相同操作的对象的描述。

OOP 的另外两个特征是继承性和多态性,继承性使得另一个对象可以获得一个对象的特性,它不仅很好地从概念上划分了类,而且可以充分地使代码得到重用。多态性使得一些具有相似功能的函数可以用同一个名字来定义,它使函数名在概念上更为明确。

软件设计上升为一门艺术,它的魅力在于其内部设计(对问题的分析、文档、源码)及外部表现(功能应用)中的逻辑的清晰性和实现的简明性。采用 OOP 方法,符合人的自然思维,对客观世界的抽象更为容易,程序代码也更加清晰、流畅、易读。

用户界面包括窗口、菜单、列表、对话框等部件,后几种都具有窗口的一些基本性质,可以将它抽象出来以构造一个窗口基类 windowbase。windowbase 具有各类窗口共同的外部状态和内部处理能力,外部状态包括窗口位置、颜色、边框等,内部处理有窗口显隐、打开、关闭等。windowbase 可以用来建立一些独立的临时任务窗口,如

对话框、出错信息窗、帮助窗口等。由 windowbase 可以派生出各种不同的部件类，构成如下的继承关系图：



二、分层式窗口系统中窗口的显示与隐藏

在分层式窗口系统中，因为对任一窗口都可以进行操作，为了能灵活地切换到不同的窗口，每个窗口除了保存背面信息外，还需要保存正文信息。当窗口隐藏时，将它背面信息还给屏幕，当窗口显示时，将它正文信息还给屏幕，这样，可以使屏幕上多个窗口间互不影响、独立操作。对一个窗口操作时，其它窗口的背面及正文信息都有可能发生变化。如何正确的保存每个窗口的背面及正文信息是实现分层式窗口系统管理的关键。

levelwindow 类中用成员 textbuf 保存窗口正文信息。对于窗口中的每一个单元，在 textbuf 中用三个字节保存，第一个字节保存其屏幕特性，第二个字节保存其屏幕属性(颜色、闪烁位)，第三个字节用于存放该单元的可见属性，如为 1 则表示该单元在屏幕上可见，为 0 则为不可见。

窗口操作大部分都同窗口显示隐藏有关，下面探讨窗口显示隐藏及输出时，对背面及正文信息保存算法。

1.坐标体系

在屏幕上可存在两种坐标体系，一是屏幕绝对坐标，屏幕左上角为坐标原点(1,1)。另一种是窗口相对坐标，对于每一个窗口而言，窗口左上角为坐标原点(1,1)。下面如没有特殊申明均采用屏幕绝对坐标。

2.背面信息的保存

通过图 1 可以观察到，如果一个窗口被暂时隐藏了，那么它对其余窗口的显隐不会有任何影响。因此，在保存背面及正文信息时，可不考虑被暂时隐藏的窗口，但在它被重新显示前一定要重新保存它的背面及正文信息。

对任一窗口 WK，WK 中每一单元(x,y)的背面信息总是依次属于下面三种情况：

(1)为距离 WK 最远的覆盖(x,y)的前端窗口(x,y)单元正文信息。

(2)为距离 WK 最近的覆盖(x,y)的底端窗口(x,y)单元正文信息。

(3)为屏幕背景信息。

levelwindow 类用成员函数 cover()判断屏幕上一个单元(x,y)是否被某窗口所覆盖，定义如下：

```
int levelwindow::cover(int x,int y)
{if(!visible)return 0;／／窗口不可见
 else
   if(x>=leftx&& x<=rightx&& y>=upy &&y<=downy) return 1;
   else return 0;
}
```

设窗口序列(W1,…Wk,…Wn)，保存 Wk 中的某一单元(x,y)背面信息的算法如下：

```
savebackinfo(int x,int y)
{for (Wi 从 Wn 至 W1)
    if(Wi 不等于 Wk 且 Wi.cover(x,y))
      用 Wi 的(x,y 单元正文信息替换 Wk(x,y)单元的背面信息并返回;
    用屏幕(x,y)单元的信息替换 Wk(x,y)单元的背面信息;
}
```

由此可很容易得到保存 Wk 中所有单元背面信息的算法 savebackinfo()。它在 levelwindow 类中与保存一个单元背面信息的函数同名，只是不带参数。

从图 1 可见，当任一窗口 Wk 被显示或隐藏后，其余窗口的背面信息仅在 Wk 所在区域内受到影响。以图 1 为例，W2 隐藏前 W1 的背面信息如图 2 所示，隐藏后 W1 背面信息变为图 3。如再将 W2 显示出来，W1 的背面信息又会由图 3 变为图 2.因而对 Wk 操作后，只需重新保存其余窗口与 Wk 重叠部分的背面信息即可。

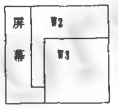


图 2

图 3

算法 saveotherwnds 描述如下：

```
saveotherwnds()
{遍历 Wk 所有单元(i,j)
    对所有 Wk 前端及底端窗口 Wi∈(W1,…,WK-1,WK+1,…,Wn)
    if(Wi.cover(i,j))Wi.savebackinfo(i,j);
}
```

3.正文信息的保存

对于正文信息来说，窗口每个单元保存在 textbuf 中的头两字节由窗口输出函数改变，这里仅考虑其可见属性字节的变化情况。通过图 1 可以观察到，一个窗口内某个单元是否可见取决于它是否在该窗口的某一可见的前端窗口内。

设置 WK 中的一个单元可见属性的算法如下:

```
setvisible (int x,int y)
{for (Wi 从 WK+1 至 Wn)
    if(Wi.cover(x,y))
    {  WK 中 textbuf 相应单元的可见属性=0
       返回;
    }
    WK 中 textbuf 相应单元的可见属性=1;
}
```

由此可得设置 WK 中所有单元可见属性的算法 setvisible()。

从图 1 可以看到,当窗口 WK 被隐藏或显示后,仅影响 WK 底端窗口的可见属性,前端窗口不受影响,并且仅影响 WK 所在区域部分。WK 显示或隐藏的情况略有不同,WK 显示后,其底端窗口在 WK 区域部分立刻不可见,而 WK 隐藏后 WK 所在区域其底端窗口是否可见还需经过判断才能知道。

设置其余窗口可见属性算法 setotherwnds 如下:

```
setotherwnds(int option)/ / option 为 1:WK 显示,为 0:WK 隐藏
{遍历 WK 所有单元(i,j)
   对所有 WK 底端窗口 Wi∈(W1,…,WK-1)
     if(Wi.cover(i,j))
       if(option)Wi 的 textbuf 中(i,j)单元可见属性=0;
       else Wi.setvisible(i,j);
```

4.窗口的输出操作

对一个窗口进行输出操作时,不仅该窗口正文信息发生变化,同时其余窗口中有一部分背面信息也将发生变化,在输出函数中必须重新保存它们。输出函数还应能判断在输出单元窗口是否可见,如不可见则不在屏幕上打印。

对窗口 WK 输出时,输出的单元有两种情况,一是该单元在屏幕上可见,此时所有覆盖该单元的 WK 的底端窗口相应的背面信息应变为 WK 的输出信息。如果该单元不可见,则需重新保存所有覆盖该单元的 WK 的前端窗口相应的背面信息。

窗口 WK 字符输出函数 wputch()算法如下:

```
wputch(char ch)
{改变 textbuf 中光标对应单元头两字节为 ch 和 wcolor;
   if(光标所在单元的可见属性==1)
   {  在屏幕上打印 ch;
     对所有 WK 底端窗口 Wi∈(WK-1,…,Wn)
      if(Wi.cover(curx+1eftx-1,cury+upy-1))
        Wi.savebackinfo(curx+leftx-1,cury+upy-1);
   }
   else
     对所有 WK 底端窗口 Wi∈(WK-1,…,Wn)
      if(Wi.cover(curx+1eftx-1,cury+upy-1))
        wi.savebackinfo(curx+1,eftx-1,cury+upy-1);
     调整光标;
}
```

三、分层式窗口系统的管理机制

1.利用双向链表建立多个任务窗口间的联系

双向链表是一种数据结构,但它也可以构造成独立的类 dblink,dblink 本身并没有多少意义,但可以用它来派生出具有双向链表结构的新类。

dblink 说明如下:

```
class dblink{
  dblink * next, * prior;
public:
  dblink(){next=prior=NULL;}
  void tstore(dblink * * start,dblink * * end);/ /将对象添加到表尾
  void hstore(dblink * * start,dblink * * end);/ /将对象添加到表头
  void remove(dblink * * start,dblink * * end);/ /将对象从表中删除
  dblink * getnext(){return next;}
  dblink * getprior(){return prior;}
};
```

窗口链由指针 wrear/wfront 指示头尾。在窗口管理时我们让 wrear 和 wfront 始终指向最底端窗口和最前端窗口。链表中从 wrear 至 wfront 的节点依次对应屏幕上最底端窗口关闭一个窗口时,将它从窗口链中删除。刷新窗口时先将它从链表中删除,然后再插入窗口链尾。利用双向链表,可以灵活地实现不同任务窗口间的切换。

levelwindow 说明如下:

```
class levelwindow:public windowbase,public dblink{
  int winno;/ /窗口号
  char * textbuf;/ /用于保存文本信息
   …
  int cover(int x,int y);
public:
   …
  void setvisible(int x,int y);
  void setvisible();
  void setotherwnds();
  void savebackinfo(int x,int y);
  void savebackinfo();
  void saveotherwnds(int option);
  void open(dblink * * s,dblink * * e);
  void close(dblink * * s,dblink * * e);
  void show();
  void hide();
  int brush(dblink * * s,dblink * * e);
   …
};
```

2.窗口操作的实现

对窗口的操作通过向窗口发送消息实现,消息包括:打开、关闭、隐藏、显示、激活、刷新、移动、缩放及改变窗口颜色等。它们被分别定义为 levelwindow 类的成员函

数。每个对象收到消息后进行自身的处理。对于改变颜色或窗口移动这样的操作来说,应当允许用户反复选择,不断向窗口发送消息,直到满意为止。每个成员函数仅处理各自对象本身的操作,对其余对象的处理(正文、背面信息的保存)需要在该对象操作完后进行。

函数 windowoperate(levelwindow * * wp,int method)对 * wp 所指的窗口对象进行相应操纵处理。它并不是 levelwindow 类的成员函数,实际上,它提供了一个分层式窗口系统与应用之间接口。method 用于传送在应用中定义的窗口操作键,比如用 F3 打开一个窗口,F4 切换窗口显隐状态,F5 移动窗口等。对 F5 的处理过程为:使用上下左右箭头键移动窗口,直到用回车键确定新的位置。

3.窗口设计中的其它因素

在实际应用中,往往需要处理键盘、鼠标器、视频显示等各类设备驱动问题。此外窗口本身也可能具有诸如滚动杆、滚动条之类的高级特性。对窗口正文、标题、边框颜色的不同处理,边框类型的变化,窗口号的显示,光标类型的控制也都是实际应用中应实现的基本功能。另外还可以考虑如何利用 PC 机的扩充内存存放 backbuf 与 textbuf,以便为程序的其它部分留出更多的可用内存。

四、一个与应用接口的实例

在应用系统中,可以将所有对窗口的操作处理放在函数 windowoperate 中完成,下面提供一个 windowoperate 的实例,读者可以从中进一步了解到分层式窗口系统与应用接口机制的实现。在下面的 windowoperate 中用 F3 至 F7 以及 ALT 1、ALT 2 等键进行窗口和各种操作,在每次操作后都需重新保存其余窗口背面及正文信息。

```
void windowoperate(levelwindow * * wp,int method)
{ / / wp 为指向当前窗口的指针的地址
  levelwindow * tmp;
  int ch;
  switch(method){
    case F3: / / open
      tmp = new levelwindow(12,5,68,20,7,1,15,2,"");
      tmp->open(&wrear, * wfront);
      tmp->saveotherwnds();
      tmp->setotherwnds(1);
      ( * wp) = tmp;
      break;
  case F4: / / close
      tmp = ( * wp);
      ( * wp)->close(&wrear,&wfront);
      ( * wp)->setotherwnds(0);
      ( * wp)->saveotherwnds();
      if(!(( * wp) = (levelwindow * )(tmp->getprior())))
        ( * wp) = (levelwindow * )(tmp->getnext());
      break;
  case ALT 1: / / activate window 1
    windowactivate(&( * wp),1);
    break;
  case ALT 2: / / activate window 2
    ……
  case F5: / / show or hide
    if(( * wp)->isvisible()){
      ( * wp)->hide();
      ( * wp)->setotherwnds(0);
    }
  else{
      ( * wp)->setvisible(); / / 显示时重新保存背面及
      ( * wp)->savebackinfo(); / / 正文信息
      ( * wp)->show();
      ( * wp)->setotherwnds(1);
    }
    ( * wp)->saveotherwnds();
    break;
  case F6: / / brush
    ( * wp)->brush(&wrear,&wfront);
    ( * wp)->setotherwnds(1);
    ( * wp)->saveotherwnds();
    break;
  case F7: / / move
    if(!( * wp)->isvisible())break;
    while((ch = getkey())! = ENTER)
    switch(ch){
      case LEFT:( * wp)->move (LEFT);break;
      case RIGHT:( * wp)->move (RIGHT);break;
      case UP:( * wp)->move (UP);break;
      case DOWN:( * wp)->move (DOWN);break;
    }
    tmp = (levelwindow * )wrear;
    while(tmp){
      tmp->setvisible();
      tmp->savebackinfo();
      tmp = (levelwindow * )(tmp->getnext());
    }
    break;
  }
}
void windowactivate(levelwindow * * wp,int wno)
{ / / 激活窗口号为 wno 的窗口
  levelwindow * tmp;
  if(( * wp)->getwinno()! = wno){
  tmp = (levelwindow * )wrear;
  while(tmp &&(tmp->getwinno())! = wno)
    tmp = (levelwindow * )(tmp->getnext());
  if(tmp)( * wp) = tmp;
  }
}
```

参考文献(略)