第 5 章　CRXed 命令及交互

用户通过 Crxed 命令可以与电子图板进行交互，使电子图板可以得到用 户输入，同时，用户可以访问电子图板选择集，对当前选择进行操作。

5.1　crxedCommand 函数和结果缓冲区

5.1.1 说明

在 ObjectCRX 编程中，可以使用 crxedCommand 或 crxedCmd 函数来执行电子图板内部的命令，这与在电子图板中直接执行相应命令的效果是一样的（甚至在命令窗口中都能找到执行命令的痕迹），本节通过创建圆、对图形进行缩放来说明这两个函数的使用。

结果缓冲区作为一种特殊的数据类型，在某些特定的场合仍有不可替代的作用，因此在实例中通过创建和访问结果缓冲区的内容来帮助读者更号地理解结果缓冲区的结构。

本节所得到的 CRX 文件向电子图板注册三个命令，AddCircle1 和 AddCircle2 命令能够在图形窗口中创建圆，EntInfo 命令能够在命令窗口中显示所选择的圆的参数。

5.1.2 思路

1．crxedCommand 函数

crxedCommand 函数的定义为：

```
int crxedCommand(int rtype, ... unnamed);
```

该函数的参数个数是可变的，并且参数成对出现。参数对中第一个参数表示参数的类型，第二个表示其实际的数据。参数表的最后一个参数必须是 0 或者 RTNONE（使用 RTNONE 更好一些）。

下面的代码用于在 电子图板中创建一个圆心为（0，0）、半径为 10 的圆：

```
crxedCommand(RTSTR, "Circle", // 命令
RTSTR, “0,0,0”, // 圆心
RTSTR, “10”, // 半径
RTNONE); // 结束命令
```

在 AddCircle1 命令的实现函数中，使用变量值作为 crxedCommand 函数的参数，所实现的功能与上面的代码完全一样。

2．crxedCmd 函数

crxedCmd 函数的定义为：

```
int crxedCmd(const struct resbuf * rbp);
```

该函数的参数是一个 resbuf 类型的指针，这里需要的结果缓冲区可以由 crxutBuildList 函数生成。由于 crxedCommand 函数实质上也是为要执行的命令构造了一个 resbuf 结构，因此 crxedCmd 函数和 crxedCommand 函数完全能够实现相同的功能，在 AddCircle2 命令的实现函数中演示了 crxedCmd 函数的用法。

3．结果缓冲区（resbuf）

结果缓冲区（resbuf）是 ObjectCRX 中定义的一个结构体，其定义为：

```
struct resbuf {
struct resbuf *rbnext; // 连接列表的指针
short restype;
union crx_u_val resval;
};
```

其中，联合体 crx_u_val 的定义为：

```
union crx_u_val {
crx_real rreal;
crx_real rpoint[3];
short rint; // 必须声明为short，而不是int
 CxCHAR*rstring;
long rlname[2];
long rlong;
struct crx_binary rbinary;
};
```

resbuf 结构体中的 rbnext 指针可以将多个结果缓冲区连接成一个单链表（很可能你和我一样，并不是计算机专业科班出身，如果你对链表的概念感到模糊，那肯定会影响到你对结果缓冲区链表的理解，请花一个上午的时间阅读数据结构的书籍，很快你就能对此烂熟于胸），当 rbnext 等于 NULL 时表示到达了链表的末尾。在访问结果缓冲区的内容时，通常定义两个 resbuf 指针，一个指向链表的开头（以备在使用完毕后用 crxutRelRb 函数释放结果缓冲区的存储空间），另一个用于遍历链表，在 EntInfo 命令的实现函数中详细演示了这种用法。restype 变量用于指定联合体 crx_u_val 中保存的变量的类型，其取值可以是图 5.1 中的列表中的任何一种。结果缓冲区链表的结构可以用图 5.1 来表示。

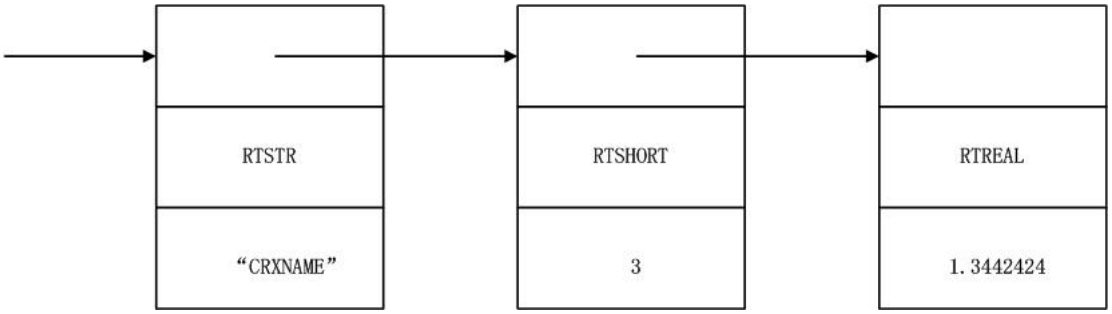

图5.1 结果缓冲区链表的结构

与结果缓冲区相关的两个全局函数：

□ crxutRelRb：释放结果缓冲区链表的存储空间。

□ crxutNewRb：创建一个新的结果缓冲区，并为其分配存储空间。使用该函数分配的存储空间必须在不用的时候用 crxutRelRb 函数手工释放空间。

请务必搞明白结果缓冲区的概念和使用，不然后面学习扩展数据、字典时你就需要回来补课。

5.1.3 步骤

（1）运行 Visual Studio 2010，使用 ObjectCRX 向导创建一个新工程，名称为 Resbuf。注册一个新命令 AddCircle1，使用 crxedCommand 函数创建一个圆，其实现函数为：

```
void CRXAddCircle1()
{

    // 调用crxedCommand函数创建圆
       crxedCommand(RTSTR, _T("Circle"), // 命令
        RTNONE); // 结束命令
}
```

（2）注册一个新命令 AddCircle2，使用 crxedCmd 函数创建一个圆，其实现函数为：

```
void CRXADDCIRCLE2()
{
    struct resbuf *rb; // 结果缓冲区
    int rc = RTNORM; // 返回值
    // 创建结果缓冲区链表
    rb = crxutBuildList(RTSTR, _T("Circle"),
         RTNONE);
    // 创建圆
    if (rb != NULL)
    {
         rc = crxedCmd(rb);
    }
    // 检验返回值
    crxutRelRb(rb);
    // 进行缩放
    crxedCommand(RTSTR, _T("Zoom"), RTSTR, _T("E"), RTNONE);
}
```

首先使用 crxutBuildList 函数构造一个结果缓冲区，将其作为参数传递给 crxedCmd 函数，这样的功能通过 crxedCommand 函数同样可以实现。需要注意的是，crxutBuildList 函

数会构造一个结果缓冲区并自动分配内存，在不需要的时侯必须用 crxutRelRb 函数释放分配的内容。

最后，使用 crxedCommand 函数调用电子图板内部命令 Zoom 对图形窗口进行缩放操作。

5.1.4 效果

（1）编译连接程序，启动电子图版 2011，加载生成的 CRX 文件，执行 AddCircle1 和 AddCircle2 命令，系统会在图形窗口中创建两个圆。如果按下 F2 键显示命令窗口，能够发现其中包含了创建圆的电子图板命令提示语句，如图 5.2 所示。

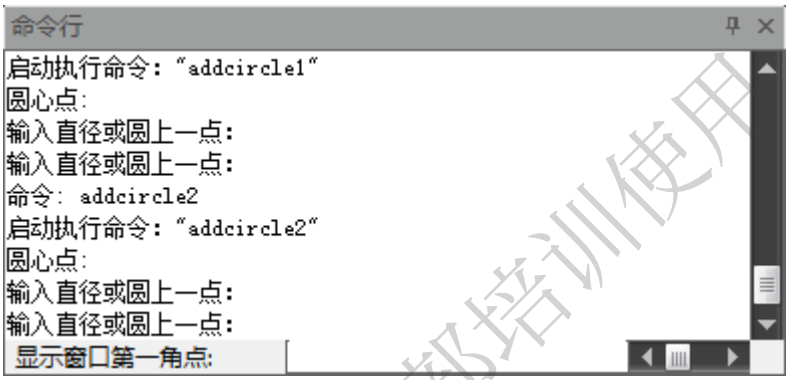

图5.2 查看命令窗口中的命令执行信息

5.1.5 小结

虽然 crxedCommand 和 crxedCmd 函数能够调用电子图板内部命令完成一些操作，但是这两个函数会降低程序的灵活性，因此在 ObjectCRX 编程中不推荐使用。

前面的章节中已经使用过 crxedEntSel 等函数，并且使用过它的返回值 RTNORM 来判断函数是否正确执行。实际上，大多数的 CRXRX 函数的返回值都是下面的一种：

- RTNORM：库函数成功执行。
- RTERROR：库函数执行过程中遇到一个可以捕获的错误。
- RTCAN：用户按下 Esc 键终止函数的执行。
- RTREJ：电子图板拒绝了无效的请求。
- RTFAIL：与 AutoLISP 的连接失败。
- RTKWORD：用户输入了一个关键字。

5.2　和用户交互

5.2.1　说明

ObjectCRX 中提供了多个提示用户输入的全局函数，包括 crxedGetString、crxedGetPoint、crxedGetInt、crxedGetKword 和 crxedGetReal 等。这些方法本身的使用很简单，仅在小结部分提供参考的使用代码，本节所要讨论的主要问题是 crxedGetPoint 函数和关键字的结合应用，例如在电子图板中执行 PLINE 命令时，能够得到如下图 5.3 的命令提示：

图5.3 pline命令

其中的提示“下一点”和下面的四个选线就将提示用户输入点和关键字结合在一起。

本节的实例将要模拟 电子图板中动态创建多段线的效果，并且允许用户通过命令行的关键字来改变多段线的某些特性。

5.2.2　思路

1．动态创建多段线

动态创建多段线，最基本的要求是用户在图形窗口中按顺序拾取多个顶点，每次拾取一点都会将其添加到多段线的末尾，最终按下 Enter 键或者 Esc 键完成多段线的创建。

根据这个思路，可以编写下面的代码来完成动态创建多段线：

```
void CRXAddPolyBasic()
{
int index = 2; // 当前输入点的次数
crx_point ptStart; // 起点
if (crxedGetPoint(NULL, "\n输入第一点：", ptStart) != RTNORM)
return;
crx_point ptPrevious, ptCurrent; // 前一个参考点，当前拾取的点
crxdbPointSet(ptStart, ptPrevious);
crxDbObjectId polyId; // 多段线的ID
while (crxedGetPoint(ptPrevious, "\n输入下一点：", ptCurrent) ==
RTNORM)
{
   if (index == 2)
```

```
{
// 创建多段线
CRxDbPolyline *pPoly = new CRxDbPolyline(2);
CRxGePoint2d ptGe1, ptGe2; // 两个节点
ptGe1[X] = ptPrevious[X];
ptGe1[Y] = ptPrevious[Y];
ptGe2[X] = ptCurrent[X];
ptGe2[Y] = ptCurrent[Y];
pPoly->addVertexAt(0, ptGe1);
pPoly->addVertexAt(1, ptGe2);
// 添加到模型空间
polyId = PostToModelSpace(pPoly);
}
else if (index > 2)
{
// 修改多段线，添加最后一个顶点
CRxDbPolyline *pPoly;
crxdbOpenObject(pPoly, polyId, CRxDb::kForWrite);
CRxGePoint2d ptGe; // 增加的节点
ptGe[X] = ptCurrent[X];
ptGe[Y] = ptCurrent[Y];
pPoly->addVertexAt(index - 1, ptGe);
pPoly->close();
}
     index++;
    crxdbPointSet(ptCurrent, ptPrevious);
}
}
```

上面的代码包含了最基本的动态创建多段线的语句，其算法可以描述为：

（1）拾取第一点；

（2）拾取第二点，创建多段线；

（3）如果用户没有按下 Enter 键或者 Esc 键，拾取下一点，并将拾取的点添加到多段线的末尾；

（4）如果用户按下 Enter 键或者 Esc 键，退出程序执行，完成多段线的创建，否则反复执行步骤（3）。

2．在 crxedGetPoint 函数中使用关键字

要在 crxedGetPoint 函数中使用关键字，必须在使用 crxedGetPoint 函数之前定义关键字，并且使用 crxedInitGet 函数来设置关键字，并紧跟这一句调用 crxedGetPoint 函数。下面的代码演示了在 crxedGetPoint 函数中使用关键字的方法：

```
void CRXGetPoint()
```

```
{
int rc; // 返回值
CxCXCHARkword[20]; // 关键字
crx_point pt;
crxedInitGet(RSG_NONULL, "Keyword1 keyWord2");
rc = crxedGetPoint(NULL, "输入一个点或[Keyword1/keyWord2]:", pt);
switch (rc)
{
case RTKWORD: // 输入了关键字
if (crxedGetInput(kword) != RTNORM)
return;
if (strcmp(kword, "Keyword1") == 0)
crxedAlert("选择的关键字是Keyword1!");
else if (strcmp(kword, "keyWord2") == 0)
crxedAlert("选择的关键字是keyWord2!");
break;
case RTNORM:
crxutPrintf("输入点的坐标是(%.2f, %.2f, %.2f)",
pt[X], pt[Y], pt[Z]);
} // switch
}
```

要获得 crxedGetPoint 函数执行过程中用户输入的关键字，可以通过该函数的返回值来判断用户是否输入了关键字，如果返回值是 RTKWORD 就通过 crxedGetInput 函数获得输入关键字的内容。之所以两个关键字的大写字母不都是第一个，分别为 Keyword1 和 keyWord2，是为了让用户可以用不同的大写字母（K 和 W）来代表这两个关键字。

5.2.3 步骤

（1）启动Visual Studio 2010，使用ObjectCRX向导创建一个新工程，其名称AddPolyDynamic。注册一个新命令 AddPolyDynamic，该命令的实现函数被放置在CrxEntryPoint.cpp 文件中。

（2）注册一个新命令 AddPoly，提示用户输入多段线的节点、线宽和颜色，完成多段线的创建，其实现代码为：

```
void CRXAddPoly()
{
    int colorIndex = 5; // 颜色索引值
    crx_real width = 2; // 多段线的线宽
    int index = 2; // 当前输入点的次数
    crx_point ptStart; // 起点
    // 提示用户输入起点
```

```
if (crxedGetPoint(NULL, _T("\n输入第一点:"), ptStart) != RTNORM)
    return;
crx_point ptPrevious, ptCurrent; // 前一个参考点，当前拾取的点
crxdbPointSet(ptStart, ptPrevious);
CRxDbObjectId polyId; // 多段线的ID
// 输入第二点
crxedInitGet(NULL, _T("0"));
int rc = crxedGetPoint(ptPrevious,
    _T("\n输入下一点 <完成(0)>:"), ptCurrent);
while (rc == RTNORM || rc == RTKWORD)
{
    if (rc == RTKWORD) // 如果用户输入了关键字
    {
        CxCHAR kword[20];
        if (crxedGetInput(kword) != RTNORM)
            return;

        else if (_tcscmp(kword, _T("0")) == 0)
        {
            return;
        }


        else
        {
            crxutPrintf(_T("\n无效的关键字."));
        }
    }
    else if (rc == RTNORM) // 用户输入了点
    {
        if (index == 2)
        {
            // 创建多段线
            CRxDbPolyline *pPoly = new CRxDbPolyline();
            CRxGePoint2d ptGe1, ptGe2; // 两个节点
            ptGe1[X] = ptPrevious[X];
            ptGe1[Y] = ptPrevious[Y];
            ptGe2[X] = ptCurrent[X];
            ptGe2[Y] = ptCurrent[Y];
            pPoly->addVertexAt(0, ptGe1);
```

```
                pPoly->addVertexAt(1, ptGe2);

                // 修改多段线的颜色和线宽
                pPoly->setConstantWidth(width);
                pPoly->setColorIndex(colorIndex);

                // 添加到模型空间
                polyId = AddToModelSpace(pPoly);
            }
            else if (index > 2)
            {
                // 修改多段线，添加最后一个顶点
                CRxDbPolyline *pPoly;
                crxdbOpenObject(pPoly, polyId, CRxDb::kForWrite);
                CRxGePoint2d ptGe; // 增加的节点
                ptGe[X] = ptCurrent[X];
                ptGe[Y] = ptCurrent[Y];
                pPoly->addVertexAt(index - 1, ptGe);


                pPoly->close();
            }
            index++;
            crxdbPointSet(ptCurrent, ptPrevious);
        }
        // 提示用户输入新的节点
        crxedInitGet(NULL, _T("0"));
        rc = crxedGetPoint(ptPrevious,
            _T("\n输入下一点 <完成(0)>:"), ptCurrent);
    }
}
```

在前面已经分别对动态创建多段线和在 crxedGetPoint 中使用关键字作了详细的介绍，这里的代码基本上就是将两个程序综合在一起，不再详细分析了。上面的代码中，使用了 crxdbPointSet 宏将一个 crx_point 变量的值复制到另一个 crx_point 变量中，其定义为：

```
#define crxdbPointSet(from, to) (*(to)= *(from), (to)[1]=(from)[1], (to)[2]=(from)[2])
```

5.2.4 效果

编译链接程序，启动电子图版 2011，加载生成的 CRX 文件，执行 AddPoly 命令，按

照命令提示进行操作：

命令: AddPolyDynamic

输入第一点:

输入下一点 [宽度(W)/颜色(C)]<完成(O)>:

输入下一点 [宽度(W)/颜色(C)]<完成(O)>:

输入下一点 [宽度(W)/颜色(C)]<完成(O)>:o 【输入O完成创建】

完成操作后，在图形窗口中会得到一条线宽为 2 的蓝色多段线，如图5.4

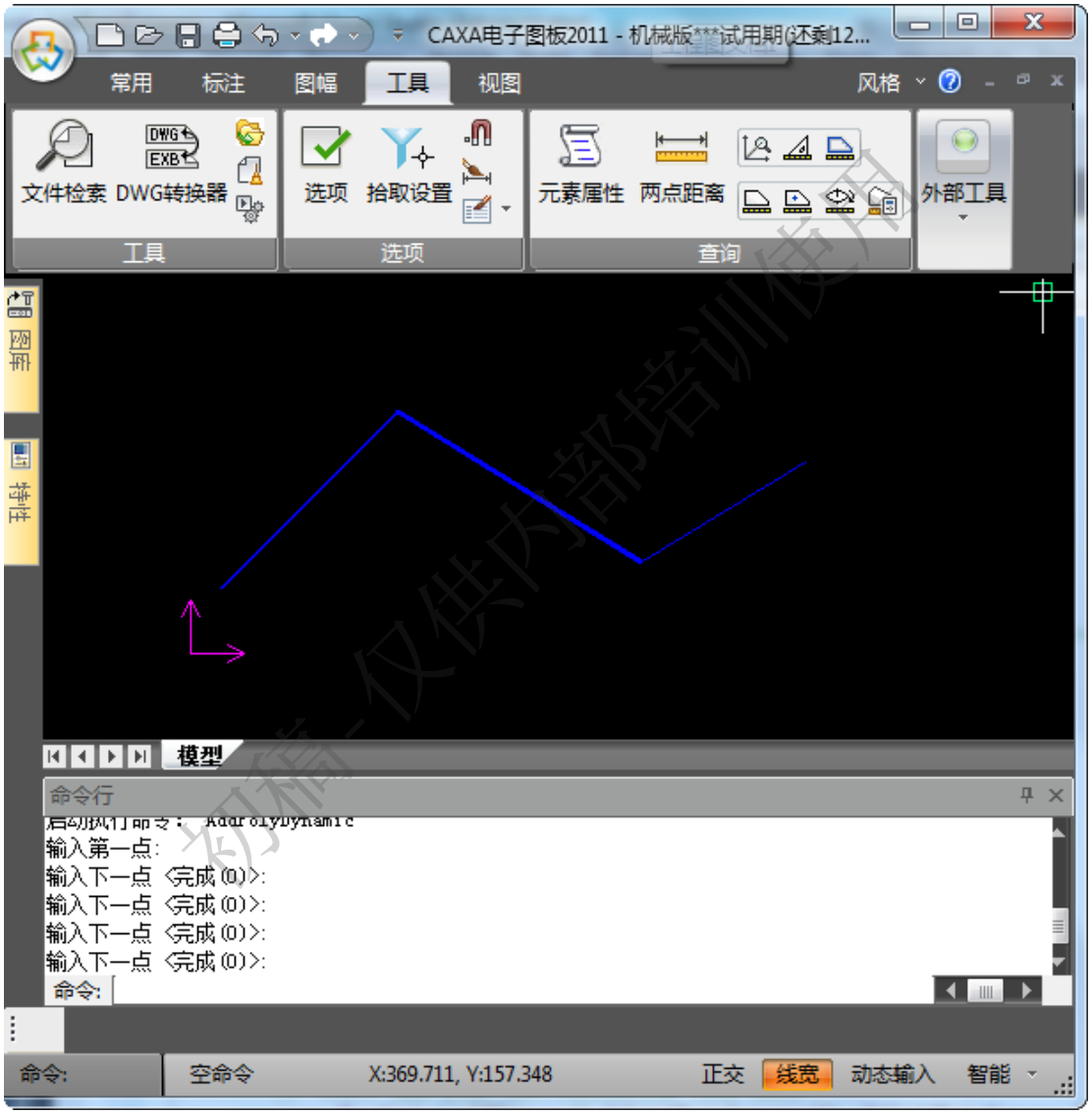

图5.4 多段线创建

5.2.5 小结

这里简单介绍和用户交互的其他函数。

1．其他交互函数

crxedGetInt 函数暂停程序执行，等待用户输入一个整数。该函数定义为：

```
int crxedGetInt(const CxCXCHAR* prompt, int * result);
```

prompt 用于指定显示在命令窗口中的字符串，如果不需要使用可以指定 NULL 作为该参数的值。

crxedGetReal 函数暂停程序执行，等待用户输入一个实数。

crxedGetString 函数暂停程序执行，等待用户输入一个字符串，该函数定义为：

```
int crxedGetString(int cronly, const CxCHAR* prompt, CxCHAR* result);
```

其中，参数 cronly 说明字符串能否包含空格，输入 0 表示不允许字符串中包含空格，1 表示字符串可以包含空格。

crxedGetDist 函数暂停程序执行，等待用户输入直线距离。

crxedGetCorner 函数暂停程序执行，等待用户输入图形窗口的矩形框的对角点，一般和 crxedGetPoint 函数配合使用。

crxedGetAngle 函数暂停程序，等待用户输入一个相对于 ANGBASE 系统变量设置的当前值的角度。

crxedInitGet、crxedGetKword 和 crxedGetInput 配合用于提示用户输入关键字，并根据用户输入的关键字执行不同的操作。

5.3 选择集

5.3.1 说明

在 ObjectCRX 开发过程中，经常需要用户和电子图板之间进行交互操作，除了前面介绍的 crxedGetXX 系列函数之外，选择集是电子图板和用户交互操作的重要手段。选择集允许用户同时选择多个图形对象，同时提供了丰富的手段来选择符合特定条件的实体。

本节的实例详细介绍了选择集的使用，可以分为以下几个方面：

- 选择集的创建和删除。
- 选择集中对象的增加和删除。
- 对象选择的方法。
- 选择集过滤器的使用。

5.3.2 思路

1．选择集的创建和删除

选择集是电子图板当前图形中的一组实体，通过图元名进行引用，也就是一个 crx_name 变量。创建选择集可以使用 crxedSSAdd 和 crxedSSGet 函数，其中 crxedSSGet 函数提供了绝大多数创建选择集的方法：

- 提示用户选择实体。
- 使用 PICKFIRST 选择集（在未执行命令时用户已经选择的图形集合，也就是电子图板中的先选择、再输入命令）、最后一个（Last）、前一个（Previous）、窗口（Window）等方式，也可以指定一个点或者一系列点来明确地限定所要选择的实体。
- 指定选择实体所要满足的一系列属性和条件来过滤当前数据库，可以和前面的选择方式配合使用。

无论使用 crxedSSAdd 还是 crxedSSGet 函数，都需要在程序结束之前使用 crxedSSFree 函数释放选择集的内存空间。

2．选择集中元素的增加和删除

这里所说的元素增加和删除，仅指在已经获得对象引用的情况下，使用 crxedSSAdd 和 crxedSSDel 函数对选择集进行元素的增加和删除。crxedSSAdd 函数定义为：

```
int crxedSSAdd(const crx_name ename, const crx_name sname, crx_name result);
```

其中，ename 指定要添加到选择集的实体的图元名；sname 指定选择集的图元名；result 返回被创建或者更新的选择集。根据 ename 和 sname 的不同取值，crxedSSAdd 有以下几种可能的执行结果：

- 如果 ename 和 sname 都是空指针，则创建一个未包含任何成员的选择集。
- 如果 ename 指向一个有效的实体，但 sname 是空指针，则创建一个选择集，该选择集仅包含一个成员 ename。

□ 如果 ename 指向有效的实体，且 sname 指向有效的选择集，则将 ename 所指向的实体添加到 sname 所指向的选择集中。

crxedSSDel 函数定义为：

```
int crxedSSDel(const crx_name ename, const crx_name ss);
```

其中，ename 指定了要从选择集中删除的实体；ss 指定了所要操作的选择集。

3．对象选择的方法

所谓对象选择的方法，就是以某种方式从图形窗口中获得满足某些条件的图形对象。这里要介绍的是使用 crxedSSGet 函数所实现的选择对象的方法，该函数被定义为：

```
int crxedSSGet (const CxCHAR*str,
const void *pt1,
const void *pt2,
const struct resbuf *entmask,
crx_name ss);
```

其中，str 参数描述了创建选择集的方法，可以使用的参数值参见表 5-1；pt1 和 pt2 为相关的创建方式提供了点参数，如果不需要指定可以输入 NULL 作为参数值；entmask 用于指定选择实体的过滤条件；ss 则指定了要操作的选择集的图元名。

表 5-1 crxedSSGet 函数的选择模式选项

值（选择模式）	说　　明
NULL	单点选择（如果指定了pt1）或者提示用户选择（如果pt1的值为NULL）
#	非几何选择模式（包括All、Last 和Previous 选择模式）
:$	仅提供提示（Prompts supplied）
.	用户选择模式
:?	其他回调选择模式（Other callbacks）
A	全部选择
B	框选模式
C（保留字段）	窗交选择模式
CP（保留字段）	圈交选择模式（选择多边形（通过在待选对象周围指定点来定义）内部或与之相交的所有对象）
:D（保留字段）	允许复制选择模式（Duplicates OK）
:E（保留字段）	小孔中的所有实体（Everything in aperture）
F（保留字段）	栏选模式
G（保留字段）	选择编组
I	获得当前图形窗口中已经选择的实体（PickFirst 选择集）
:K（保留字段）	键盘回调选择模式（Keyword callbacks）
L	选择最近一次创建的可见实体
M（保留字段）	指定多次选择而不高亮显示对象，从而加快对复杂对象的选择过程
P	选择最近创建的选择集
:S（保留字段）	单一对象选择模式
W（保留字段）	窗口选择模式

WP（保留字段）	圈围选择模式
X	过滤选择模式

4．使用选择集过滤器

在使用各种选择对象的方法时，可以使用过滤器来限定选择的对象。例如，可以指定仅选择图层 0 上的直线对象，也可以指定仅选择蓝色的半径大于 30 的圆，等等。如果仅使用一个过滤条件，可以使用下面的代码：

```
struct resbuf rb;
CxCHARsbuf[10]; // 存储字符串的缓冲区
crx_name ssname;
rb.restype = 0; // 实体名
strcpy(sbuf, "CIRCLE");
rb.resval.rstring = sbuf;
rb.rbnext = NULL; // 不需要设置其他的属性
// 选择图形中所有的圆
crxedSSGet("X", NULL, NULL, &rb, ssname);
crxedSSFree(ssname);
```

上面的代码中虽然使用了结果缓冲区，但是仅是在栈上声明，由编译器自动管理它所使用的内存空间，不需要使用 crxutRelRb 函数来手工销毁它。

如果要指定多个过滤条件，可以使用 crxutBuildList 函数来构造结果缓冲区。如果要选择当前图形窗口中位于 0 层上的所有直线，就可以使用下面的方法：

```
struct resbuf *rb; // 结果缓冲区链表
crx_name ssname;
rb = crxutBuildList(RTDXF0, "LINE", // 实体类型
8, "0", // 图层
RTNONE);
// 选择图形中位于0层上的所有直线
crxedSSGet("X", NULL, NULL, rb, ssname);
crxutRelRb(rb);
crxedSSFree(ssname);
```

在标准的 DXF 组码中，0 用于表示实体类型，但是在 crxutBuildList 函数中 0 也可以用于表示结束链表，因此用 RTDXF0 来代替 0。

过滤器列表由成对的参数组成。第一个参数标识过滤器的类型（例如对象），第二个参数指定要过滤的值（例如圆）。过滤器类型是指定使用哪种过滤器的 DXF 组码，下面列出了一些最常用的过滤器类型：

15：颜色索引（整数）16：图层索引（整数）17 线型索引（整数）

□ 15：颜色索引（整数。

□ 16：图层索引（整数。

□ 17：线型索引（整数。

值得一提的是，在大部分选择模式中均可以使用选择集过滤器，下面的代码提示用户选择实体的同时使用了选择集过滤器：

```
crxedSSGet(NULL, NULL, NULL, rb, ssname);
```

5.3.3 步骤

1．选择集的创建和删除

启动 Visual Studio 2010，使用 ObjectCRX 向导创建一个新工程，其名称为 SelectionSet。注册一个新命令 CreateSSet，用于演示选择集的创建和删除，其实现函数为：

```
void CRXCreateSSet()
{
    crx_name sset; // 选择集名称
    // 选择图形数据库中所有的实体
    crxedSSGet(_T("A"), NULL, NULL, NULL, sset);
    // 进行其他操作
    crxedSSFree(sset);
}
```

2．对象选择的方法

（1）注册一个新命令 SelectEnt，使用多种不同的模式创建选择集，其实现函数为：

```
void CRXSelectEnt()
{
    crx_point pt1, pt2, pt3, pt4;
    struct resbuf *pointlist; // 结果缓冲区链表
    crx_name ssname; // 选择集的图元名
    pt1[X] = pt1[Y] = pt1[Z] = 0.0;
    pt2[X] = pt2[Y] = 5.0; pt2[Z] = 0.0;
    // 如果已经选择到了实体，就获得当前的PICKFIRST选择集
    // 否则提示用户选择实体
    crxedSSGet(NULL, NULL, NULL, NULL, ssname);
    // 如果存在，就获得当前的PickFirst选择集
    crxedSSGet(_T("I"), NULL, NULL, NULL, ssname);
    // 选择最近创建的选择集
    crxedSSGet(_T("P"), NULL, NULL, NULL, ssname);
    // 选择最后一次创建的可见实体
    crxedSSGet(_T("L"), NULL, NULL, NULL, ssname);
    // 选择通过点(5,5)的所有实体
    crxedSSGet(NULL, pt2, NULL, NULL, ssname);
    // 选择位于角点(0,0)和(5,5)组成的窗口内所有的实体
    crxedSSGet(_T("W"), pt1, pt2, NULL, ssname);
```

```
        // 选择被指定的多边形包围的所有实体
        pt3[X] = 10.0; pt3[Y] = 5.0; pt3[Z] = 0.0;
    pt4[X] = 5.0; pt4[Y] = pt4[Z] = 0.0;
    pointlist = crxutBuildList(RTPOINT, pt1, RTPOINT, pt2,
        RTPOINT, pt3, RTPOINT, pt4, 0);

    crxedSSFree(ssname);
}
```

3．使用选择集过滤器

（1）在过滤器中使用通配符。注册一个新命令 Filter1，创建一个带有通配符的过滤器，其相关代码为：

```
void CRXFilter1()
{
    struct resbuf *rb; // 结果缓冲区链表215
    crx_name ssname;
    rb = crxutBuildList(RTDXF0, _T("TEXT"), // 实体类型
        8, "0,图层1", // 图层
        1, _T("*caxa*"), // 包含的字符串
        RTNONE);
    // 选择复合要求的文字
    crxedSSGet(_T("X"), NULL, NULL, rb, ssname);
    long length;
    crxedSSLength(ssname, &length);
    crxutPrintf(_T("\n实体数:%d"), length);
    crxutRelRb(rb);
    crxedSSFree(ssname);
}
```

上面的代码构建的过滤器，能够获得位于图层“0”或者“图层 1”中、字符串内容包含 cadhelp 的所有文字对象。其中，“0,图层 1”中的逗号（，）是一个通配符，用于分隔两个模式，表示两种情况均可；“*mjtd*”中的星号（*）是一个通配符，用于匹配任意的字符序列，表示任何包含 cadhelp 的字符串均可。

（2）在过滤器中使用逻辑运算符。注册一个新命令 Filter2，创建包含逻辑运算符的过滤器，其相关代码为（已省略与 Filter1 命令实现函数相同的部分代码，请读者自行补全）：

```
rb = crxutBuildList(-4, "<OR", // 逻辑运算符开始
RTDXF0, "TEXT", // 一个条件
RTDXF0, "MTEXT", // 另一个条件
-4, "OR>", // 逻辑运算符结束
```

```
RTNONE);
```

上面构建的过滤器，能够选择图形中所有的文字和多行文字。过滤器列表中的逻辑运算符用 DXF 组码－4 来指示。逻辑运算符是字符串但必须成对出现，运算符以小于号开始（<），以大于号结束（>）。

（3）在过滤器中使用关系运算符。注册一个新命令 Filter3，创建包含关系运算符的过滤器，其相关代码为（已省略与 Filter1 命令实现函数相同的部分代码，请读者自行补全）：

```
rb = crxutBuildList(RTDXF0, "CIRCLE", // 实体类型
-4, ">=", // 关系运算符
40, 30, // 半径
RTNONE);
```

上面构建的过滤器，能够选择图形中半径大于或等于 30 的所有圆，其中的组码 40 用于指定圆的半径。过滤器列表中的关系运算符以 DXF 组码－4 来指示，用字符串来表示关系运算符。

（4）结合使用通配符和关系运算符，创建更为复杂的过滤器。注册一个新命令 Filter4，创建包含关系运算符和通配符的过滤器，其相关代码为：

```
rb = crxutBuildList(RTDXF0, "CIRCLE", // 实体类型
-4, ">,>,*", // 关系运算符和通配符
10, pt1, // 圆心
-4, "<,<,*", // 关系运算符和通配符
10, pt2, // 圆心
RTNONE);
```

上面构建的过滤器，能够选择图形中圆心在 pt1 和 pt2 两点构成的矩形内的圆，其中的组码 10 用于指定圆的圆心。

（7）使用过滤器过滤扩展数据。注册一个新命令 Filter5，创建过滤扩展数据的过滤器，其相关代码为：

```
rb = crxutBuildList(1001, "XData", // 扩展数据的应用程序名
RTNONE);
```

上面构建的过滤器，能够选择图形中所有包含“Xdata”应用程序扩展数据的图元。令人遗憾的是，除轻量多段线之外，其他的实体都不支持扩展数据内容的过滤，也就是说，如果使用下面的代码：

```
rb = crxutBuildList(1000, "Road", // 扩展数据中的ASCII字符串
RTNONE);
```

上面的过滤器仅能选择到图形中所有包含字符串“Road”扩展数据的轻量多段线，其他类型的实体不起作用。

讲到这里，相信你一定感觉在创建选择集过滤器时，如何获得所需要的组码是一个令人头疼的问题。如果你很着急解决这个问题，现在就可以去看本节的小结。

5.3.4 效果

编译链接程序，启动电子图版 2011，加载生成的 CRX 文件，执行其中定义的命令，根据系统提示的选择到的对象数量，可以对相应命令的执行结果进行测试。

测试一些函数时发现了一些奇怪的问题。为了测试的方便，创建如 图 5.10 所示的图形。其中，矩形两个角点分别为（0，0）和（100，100）。

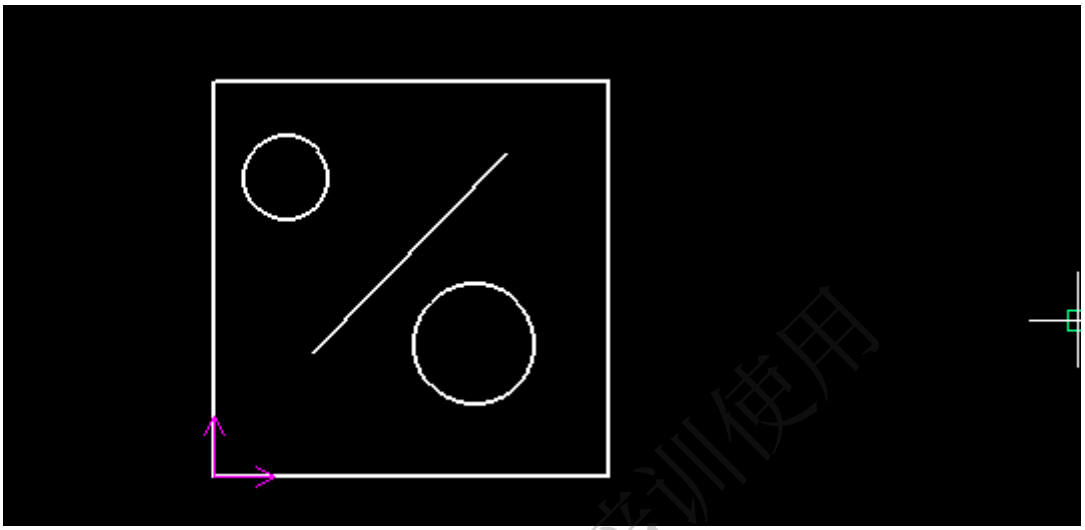

图5.10 测试所用图形

注册一个新命令 Test2，用于进行创建选择集的测试，其实现函数为：

```
void CRXTEST2()
{
    crx_name ssname;
    crx_point pt1, pt2;
    pt1[X] = pt1[Y] = pt1[Z] = 0;
    pt2[X] = pt2[Y] = 100;
    pt2[Z] = 0;
    // 选择图形中与pt1和pt2组成的窗口相交的所有对象
    crxedSSGet(_T("C"), pt1, pt2, NULL, ssname);
    long length;
    crxedSSLength(ssname, &length);
    crxutPrintf(_T("\n实体数:%d"), length);
    crxedSSFree(ssname);
}
```

确保整个矩形都位于图形窗口内部，在电子图版 2011 中执行定义的命令 Test2，得到的结果如图 5.11 所示。这个结果容易理解，包含矩形在内，符合条件的对象是 4 个。

图5.11 执行结果（一）

如果平移图形，使一些实体位于图形窗口外部，如 图 5.12 所示。再次执行 Test2 命令，能够得到如图 5.13 所示的结果。

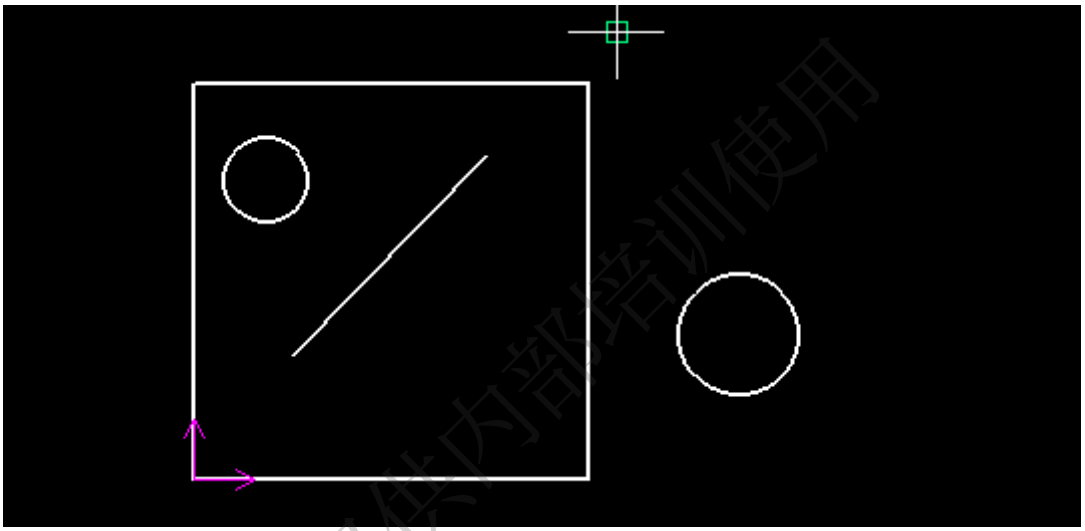

图5.12 平移图形

图5.13 执行结果（二）

这里的结果就让人费解了，矩形内部有 3 个对象，但是此处选择集仅获得了 3 个对象。实际上，使用 crxedSSGet 函数的许多选择模式（如果使用“A”作为第一个参数，也就是使用全部选择模式不受影响）仅能获得位于图形窗口（视口）内部的实体，对于视口外的实体不起作用。从图中看，当前视口中确实包含了 3 个符合要求的实体，这个结果的含义也就明朗了。

5.3.5 小结

本节的实例详细介绍了选择集的使用，学习本节内容之后，读者需要掌握下面的几个知识点：

- 选择集的创建和删除。
- 选择集中对象的增加和删除。
- 对象选择的方法。
- 选择集过滤器的使用。