第 4 章　符号表

在探讨图形数据库的基本结构时已经了解到，图形数据库由符号表和命名对象字典组成。符号表是电子图板中的一种容器对象，保存了对应的符号表记录，用来实现电子图板中的某种对象：

- 块表（CRxDbBlockTable）：包含模型空间、图纸空间和用户创建的块定义，块表记录中保存了图形数据库中的实体。
- 层表（CRxDbLayerTable）:保存了图形中所有的图层,可通过电子图板中的LAYER
- 命令查看。
- 文字样式表（CRxDbTextStyleTable）：存储图形中的文字样式，通过电子图板中的
- STYLE 命令查看。
- 线型表（CRxDbLinetypeTable）：保存了图形中加载的线型，通过电子图板中的 LTYPE 命令查看。
- 视图表（CRxDbViewTable）：存储了图形中保存的视图，通过电子图板中的 VIEW
- 命令查看。
- UCS 表（CRxDbUCSTable）：保存图形中的 UCS（用户坐标系），通过电子图板的 UCS 命令访问。
- 标注样式表（CRxDbDimStyleTable）：保存标注样式，通过电子图板中的 DIMSTYLE
- 命令访问。

从数据库获得各种符号表的方法大同小异，除了可以使用 getSymbolTable 函数，还可根据符号表的具体类型调用相应的函数，如获得块表使用 getBlockTable 函数，获得层表使用 getLayerTable 函数。

所有的符号表都继承自 CRxDbSymbolTable 类，该类包含了下面几个函数：

- add：向符号表添加一条新的记录，各种符号表实现的形式略有不同。
- getAt：获得符号表中特定名称的记录。
- has：判断符号表中是否包含指定的记录。
- newIterator：创建一个符号表遍历器，访问符号表中的所有记录。

4.1　操作图层

4.1.1 说明

本实例演示了创建新的层表记录、删除已有的层表记录、使用“颜色”对话框设置某一图层的颜色，以及导出和导入所有图层及特性的方法，基本涵盖了所有的层表操作。

4.1.2 思路

创建新的图层，实际上就是创建一个新的层表记录，并将其添加到层表中。修改图层的颜色，可以从层表中获得指定的记录，然后使用 CRxDbLayerTableRecord 类的 setColor 函数设置层表记录的颜色。删除一个图层，需要首先从层表中获得指定的层表记录，然后将层表记录设置一个“删除”的标记。导出图层列表和图层特性，需要使用层表遍历器访问每一个层表记录，将层表记录的名称、颜色、线型和线宽以“，”作分隔符连接成一个 Cstring 类型的字符串对象，然后使用 CstdioFile 类的 WriteString 函数写入到文本文件中。导入图层列表的步骤和导出的步骤完全相反，先使用 CstdioFile 类的 ReadString 函数逐行读取文本文件的内容，以“，”作分隔符解析出图层名称、颜色、线型和线宽，并在当前图形中加入这些图层。

4.1.3 步骤

（1）在 Visual Studio 2010 中，使用 ObjectCRX 向导创建一个新项目，命名为 OperateLayer。注册 NewLayer 命令，用于在图形中创建一个新图层，其实现函数为：

```
void CRXNewLayer()
{
    // 提示用户输入新建图层的名称
    CxCHAR layerName[100];
    if (crxedGetString(CAXA::kFalse, _T("\n输入新图层的名称："),layerName) != RTNORM)
    {
        return;
    }
    // 获得当前图形的层表
    CRxDbLayerTable *pLayerTbl;
    crxdbHostApplicationServices()->workingDatabase()
        ->getLayerTable(pLayerTbl, CRxDb::kForWrite);
        // 是否已经包含指定的层表记录
        if (pLayerTbl->has(layerName))
        {
            pLayerTbl->close();
            return;
        }
        // 创建新的层表记录
        CRxDbLayerTableRecord *pLayerTblRcd;
        pLayerTblRcd = new CRxDbLayerTableRecord();
```

```
        pLayerTblRcd->setName(layerName);
        // 将新建的层表记录添加到层表中
        CRxDbObjectId layerTblRcdId;
        pLayerTbl->add(layerTblRcdId, pLayerTblRcd);
        crxdbHostApplicationServices()->workingDatabase()
            ->setClayer(layerTblRcdId);
        pLayerTblRcd->close();
        pLayerTbl->close();
}
```

由于要在块表中添加新的块表记录，在获得块表的时候需要将其以“写”模式打开，对应于代码中就是在 getLayerTable 函数中使用了 CRxDb::kForWrite 参数。向块表添加新的记录之前，可以使用 has 函数判断图形中是否已经包含了同名的层。

CRxDbDatabase 类的 setClayer 函数能够设置图形的当前图层。

（2）注册新的命令 LayerColor，用于修改指定图层的颜色，其实现函数为：

```
void CRXLayerColor()
{
    // 提示用户输入要修改的图层名称
    CxCHAR layerName[100];
    if (crxedGetString(CAXA::kFalse, _T("\n输入图层的名称："),layerName) != RTNORM)
    {
        return;
    }
    // 获得当前图形的层表147
    CRxDbLayerTable *pLayerTbl;
    crxdbHostApplicationServices()->workingDatabase()
        ->getLayerTable(pLayerTbl, CRxDb::kForRead);
    // 判断是否包含指定名称的层表记录
    if (!pLayerTbl->has(layerName))
    {
        pLayerTbl->close();
        return;
    }
    // 获得指定层表记录的指针
    CRxDbLayerTableRecord *pLayerTblRcd;
    pLayerTbl->getAt(layerName, pLayerTblRcd, CRxDb::kForWrite);
    // 弹出“颜色”对话框
    CRxCmColor oldColor = pLayerTblRcd->color();
    int nCurColor = oldColor.colorIndex(); // 图层修改前的颜色
    int nNewColor = 135; // 用户设置的颜色
```

```
        pLayerTblRcd->setColor(nNewColor);
    pLayerTblRcd->close();
    pLayerTbl->close();
}
```

在获得特定的块表记录指针时，使用了使用了 CRxDbLayerTable 类的 getAt 函数，并且使用 CRxDb::kForWrite 参数，将块表记录以“写”模式打开。

（3）注册一个新命令 DelLayer，用于从图形中删除指定的图层，其实现函数为：

```
void CRXDelLayer()
{
    // 提示用户输入要修改的图层名称
    CxCHAR layerName[100];
    if (crxedGetString(CAXA::kFalse,_T("\n输入图层的名称："),layerName) != RTNORM)
    {
        return;
    }
    // 获得当前图形的层表
    AcDbLayerTable *pLayerTbl;
    crxdbHostApplicationServices()->workingDatabase()
        ->getLayerTable(pLayerTbl, CRxDb::kForRead);
    // 判断是否包含指定名称的层表记录
    if (!pLayerTbl->has(layerName))
    {
        pLayerTbl->close();
        return;
    }
    // 获得指定层表记录的指针
    CRxDbLayerTableRecord *pLayerTblRcd;
    pLayerTbl->getAt(layerName, pLayerTblRcd, CRxDb::kForWrite);
    pLayerTblRcd->erase(); // 为其设置“删除”标记
    pLayerTblRcd->close();
    pLayerTbl->close();
}
```

删除一个数据库对象非常简单：将其以“写”模式打开，调用 CRxDbObject 类的 erase 函数，最后关闭该对象即可。

（4）注册一个新命令 ExportLayer，用于将当前图形中存在的所有图层及其特性导出到一个文本文件中，其实现函数为：

```
void CRXExportLayer()
{
    // 创建所要导出的文本文件
```

```
CStdioFile f;
CFileException e;
CxCHAR *pFileName = _T("C:\\layers.txt");
if (!f.Open(pFileName, CFile::modeCreate | CFile::modeWrite, &e))
{
    crxutPrintf(_T("\n创建导出文件失败！"));
    return;
}
// 获得层表指针
CRxDbLayerTable *pLayerTbl;
CRxDbLayerTableRecord *pLayerTblRcd;
crxdbHostApplicationServices()->workingDatabase()
    ->getLayerTable(pLayerTbl, CRxDb::kForRead);
// 使用遍历器访问每一条层表记录
CRxDbLayerTableIterator *pItr;
pLayerTbl->newIterator(pItr);
for (pItr->start(); !pItr->done(); pItr->step())
{
    pItr->getRecord(pLayerTblRcd, CRxDb::kForRead);
     输出图层的信息
    CString strLayerInfo; // 图层名称
    CxCHAR *layerName;
    pLayerTblRcd->getName(layerName);
    strLayerInfo = layerName;
    free(layerName);
    strLayerInfo += ","; // 分隔符
    CString strColor; // 图层颜色
    CRxCmColor color = pLayerTblRcd->color();
    strColor.Format(_T("%d"), color.colorIndex());
    strLayerInfo += strColor;

        strLayerInfo += ",";
    CString strLinetype; // 图层线型
    CRxDbLinetypeTableRecord *pLinetypeTblRcd;
    crxdbOpenObject(pLinetypeTblRcd,
        pLayerTblRcd->linetypeObjectId(),
        CRxDb::kForRead);
    CxCHAR *linetypeName;
    pLinetypeTblRcd->getName(linetypeName);
    pLinetypeTblRcd->close();
```

```
            strLinetype = linetypeName;
            free(linetypeName);
            strLayerInfo += strLinetype;
            strLayerInfo += ",";
            CString strLineWeight; // 图层的线宽

                CRxDb::LineWeight lineWeight = pLayerTblRcd->lineWeight();
            strLineWeight.Format(_T("%d"), lineWeight);
            strLayerInfo += strLineWeight;
             将图层特性写入到文件中
            f.WriteString(strLayerInfo);
            f.WriteString(_T("\n"));
            pLayerTblRcd->close();
        }
        delete pItr;
        pLayerTbl->close();
}
```

CstdioFile 类的 Open 函数能够打开指定位置的文件，这里使用 CFile::modeCreate 作为打开标记，能够在指定的位置创建文件。

最终的结果是将图层的信息以“图层名称，颜色，线型，线宽”格式输出，因此在获得图层的名称和特性之后，关键在于将这些特性组合成一个 Cstring 类型的变量。所幸，Cstring 类提供了“＋”运算符，能够将两个字符串连接起来组成一个新的字符串，例如：

```
strLayerInfo += strLineWeight;
```

像文件中写入字符串使用了 CstdioFile 类的 WriteString 函数，注意需要单独写入一个换行字符，保证每个图层的特性单独成行。

(5)注册新命令 ImportLayer，能够按照文本文件中的图层列表在当前图形中创建图层，并且符合图层列表中的各项特性，其实现函数为：

```
void CRXImportLayer()
{
    // 打开所要导入的文本文件
    CStdioFile f;
    CFileException e;
    CxCHAR *pFileName = _T("C:\\layers.txt");
    if (!f.Open(pFileName, CFile::modeRead, &e))
    {
        crxutPrintf(_T("\n打开导入文件失败！"));
        return;
    }
    // 获得层表指针
    CRxDbLayerTable *pLayerTbl;
```

```
CRxDbLayerTableRecord *pLayerTblRcd;
crxdbHostApplicationServices()->workingDatabase()
    ->getLayerTable(pLayerTbl, CRxDb::kForWrite);
// 读取文件中的每一行数据
CString strLineText; // 一行文字
while (f.ReadString(strLineText))
{
    // 跳过空行
    if (strLineText.IsEmpty())
        continue;
    // 解析出图层名称、颜色、线型和线宽
    CStringArray layerInfos;
    if (!GetFieldText(strLineText, layerInfos))
        continue;
    // 创建新的层表记录，或者打开存在的块表记录
    CRxDbLayerTableRecord *pLayerTblRcd;
    CRxDbObjectId layerTblRcdId;
    if (pLayerTbl->has(layerInfos.GetAt(0)))
    {
        pLayerTbl->getAt(layerInfos.GetAt(0), layerTblRcdId);
    }

    else
    {
        pLayerTblRcd = new CRxDbLayerTableRecord();
        pLayerTblRcd->setName(layerInfos.GetAt(0));
        pLayerTbl->add(layerTblRcdId, pLayerTblRcd);
        pLayerTblRcd->close();
    }
    crxdbOpenObject(pLayerTblRcd, layerTblRcdId,CRxDb::kForWrite);
    // 设置层表记录的颜色
    CRxCmColor color;
    CAXA::UInt16 colorIndex = _ttoi(layerInfos.GetAt(1));
    color.setColorIndex(colorIndex);
    pLayerTblRcd->setColor(color);
    // 设置线型
    CRxDbLinetypeTable *pLinetypeTbl;
    CRxDbObjectId linetypeId;
    crxdbHostApplicationServices()->workingDatabase()
        ->getLinetypeTable(pLinetypeTbl, CRxDb::kForRead);
```

```
        if (pLinetypeTbl->has(layerInfos.GetAt(2)))
        {
            pLinetypeTbl->getAt(layerInfos.GetAt(2), linetypeId);
        }
        else
        {
            pLinetypeTbl->getAt(_T("Continous"), linetypeId);
        }
        pLayerTblRcd->setLinetypeObjectId(linetypeId);
        pLinetypeTbl->close();
        // 设置线宽
        CRxDb::LineWeight lineWeight =(CRxDb::LineWeight)_ttol(layerInfos.GetAt(3));
        pLayerTblRcd->setLineWeight(lineWeight);

            pLayerTblRcd->close();
    }
    pLayerTbl->close();
}
```

使用 CFile::modeRead 参数调用 CstdioFile 类的 Open 函数，打开指定位置的文本文件，然后使用 ReadString 函数逐行读取文本文件中的内容。

读取一行文本之后，需要根据分隔符（“，”）来解析出图层的名称、颜色、线型和线宽，使用 GetFieldText 函数来实现，该函数的实现代码为：

```
BOOL GetFieldText(CString strLineText, CStringArray &fields)
{
    if (strLineText.Find(_T(","), 0) == -1) // 如果找不到英文逗号，函数退出
    {
        return FALSE;
    }
    int nLeftPos = 0, nRightPos = 0; // 查找分隔符的起始位置
    while ((nRightPos = strLineText.Find(_T(","), nRightPos)) != -1)
    {
        fields.Add(strLineText.Mid(nLeftPos, nRightPos - nLeftPos));
        nLeftPos = nRightPos + 1;
        nRightPos++;
    }
    // 最后一个列的数据
    fields.Add(strLineText.Mid(nLeftPos));
    return TRUE;
}
```

Cstring 类提供了一系列用于查找字符或者切割字符串的函数，使得解析字符串变量非

常简单。例如 Find 函数能在指定的字符串中查找匹配子串的第一个位置，Mid 函数则能解析出指定字符串中指定长度的子串。获得某一图层的名称、颜色、线型和线宽之后，就可以创建新的层表记录，并且使用 setColor 函数设置图层的颜色，setLinetypeObjectId 函数设置图层的线型，setLineWeight 函数设置图层的线宽。

4.1.4 效果

（1）编译运行程序，在电子图板 2011 中执行 newLayer 命令，然后输入 CRXLayer 作为图层名称，完成图层的创建。选择【格式 / 图层】菜单项，系统会弹出如图 4.1 所示的【图层特性管理器】，图层列表中已经包含了新创建的 CRXLayer 图层。

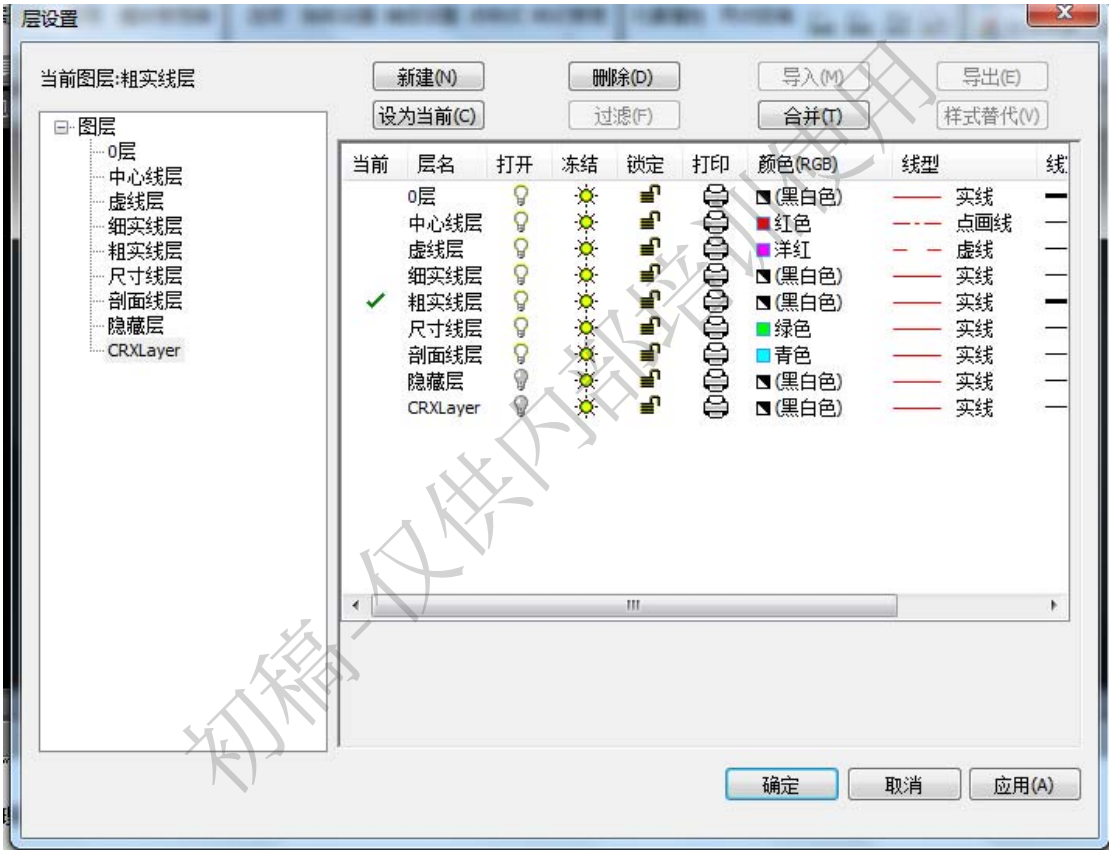

图4.1 显示新创建的图层

（2）使用“图层特性”工具栏，将 0 层作为当前图层。执行 DelLayer 命令，输入 CRXLayer 作为图层名称，按下 Enter 键完成图层的删除。

由于 电子图板不允许删除图形的当前图层，因此在删除 NewLayer 图层之前，必须保证它不是当前图层。

（3）执行 LayerColor 命令，输入中心线层作为图层名称，点击“Enter”执行。然后，在【图层特性管理器】中查看中心线层，就会发现中心线层的颜色已变为蓝色。

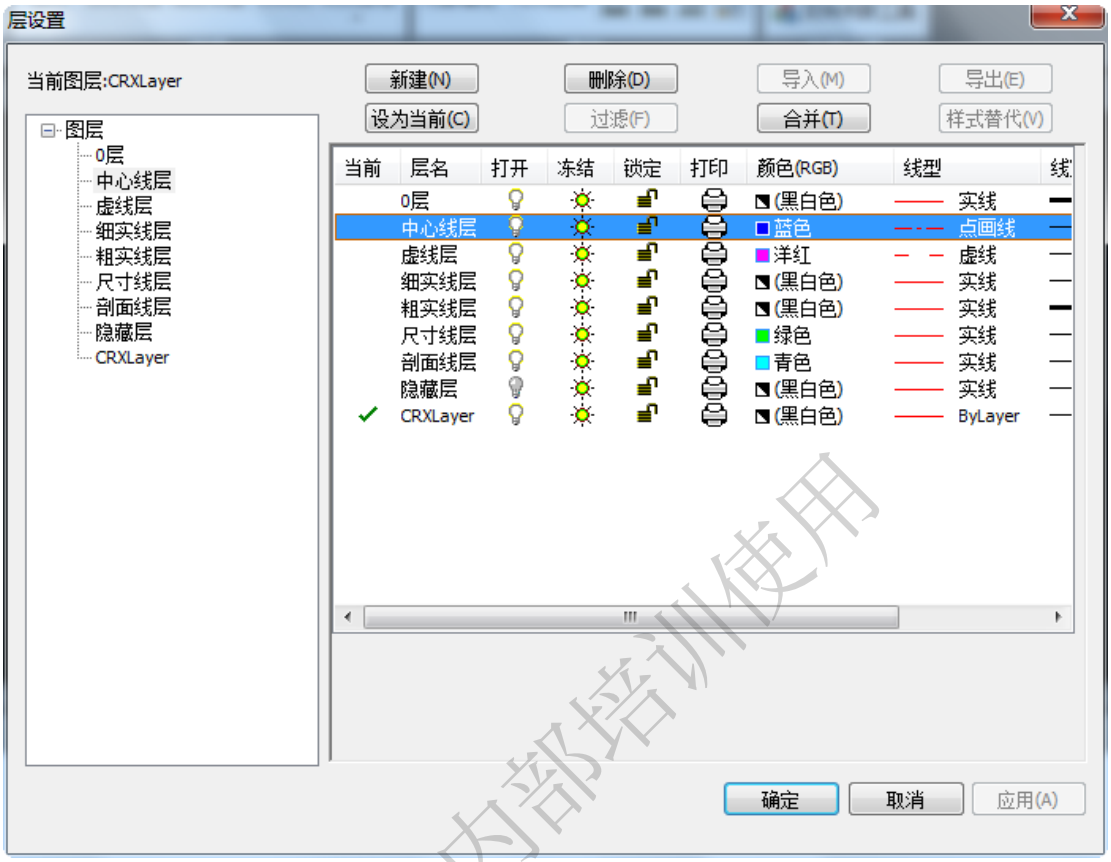

图4.2 设置中心线层的颜色

（4）在【图层特性管理器】中，创建若干个新图层，并且设置适当的颜色、线型和线宽，如图 4.3 所示。

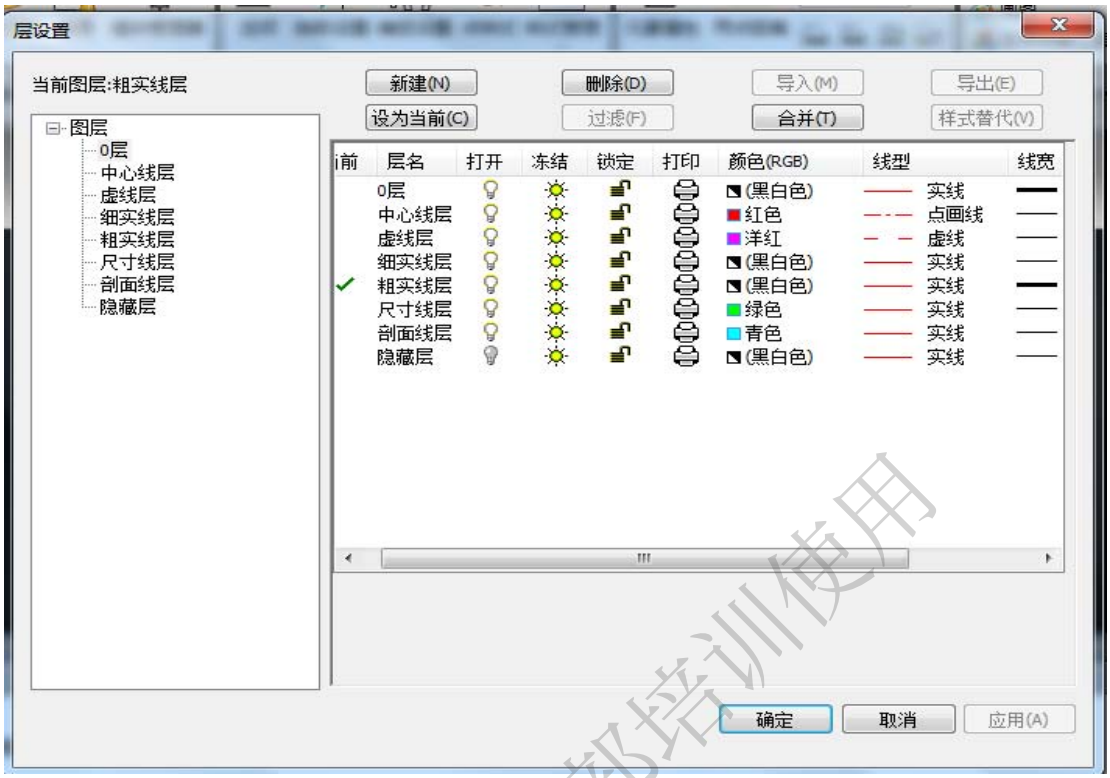

图4.3 创建要导出的图层、设置图层特性

（5）执行 ExportLayer 命令，将图层列表及图层特性导出到文本文件中，得到的文件 layers.txt 内容如图 4.4 所示。

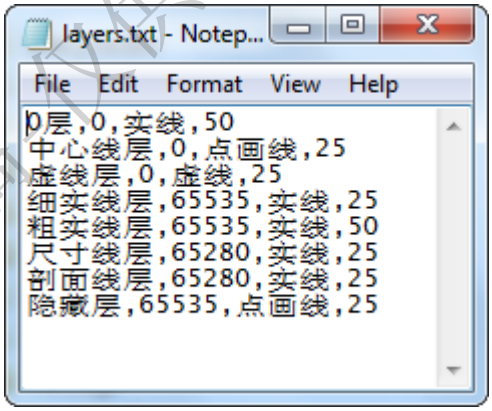

图4.4 导出文件的内容

（6）关闭当前的图形，创建一个新图形，执行 ImportLayer 命令，然后在【图层特性管理器】中查看图层列表，会发现新图形的图层及其特性与上一个图形完全一致。

4.1.5 小结

学习本节内容之后，读者需要掌握下面的要点：

□ 删除数据库对象的方法。

- 逐行读写文本文件的方法。
- 获得和设置图层的颜色、线型和线宽。

4.2 创建字体样式

4.2.1 说明

在电子图板中可以使用 STYLE 命令创建新的字体样式，包括设置样式名、选择字体文件和确定字体效果三个步骤。本实例演示了创建字体样式的方法。

4.2.2 思路

使用 ObjectCRX 创建字体样式，需要执行下面的步骤：
（1）获得当前图形的字体样式表；
（2）创建新的字体样式表记录对象；
（3）用 setName 函数设置字体样式表记录的名称；
（4）用 setFileName 函数设置字体样式表记录的字体；
（5）用 setXScale 函数设置字体样式的高宽比；
（6）将新的字体样式表记录添加到字体样式表中。

4.2.3 步骤

在 Visual Studio 2010 中，使用 ObjectCRX 向导创建一个新的项目，命名为 AddTextStyle，注册一个新命令 AddTextStyle，用于创建新的字体样式，其实现函数为：

```
void CRXTextStyle()
{
    CRxDbDatabase *pDb = crxdbHostApplicationServices()->workingDatabase();

    CRxDbObjectId syTxtId;
    CRxDbTextStyleTable* pTextStyleTbl = NULL;
    if ( CDraft::eOk != pDb->getTextStyleTable(pTextStyleTbl, CRxDb::kForWrite))
    {
        return;
    }

    if (!pTextStyleTbl->has(_T("newTxtStyle")))
    {
        CRxDbTextStyleTableRecord *pTxtStyle = new CRxDbTextStyleTableRecord;
```

```
        //设置文字样式 名称
        pTxtStyle->setName(_T("newTxtStyle"));
        //西文字体
        pTxtStyle->setEFont(_T("simsun"));
        //中文字体
        pTxtStyle->setCFont(_T("simsun"));
        //倾斜角
        pTxtStyle->setObliquingAngle(0.0);
        //缺省字高
        pTxtStyle->setTextSize(10);
        //西文宽度系数
        pTxtStyle->setXScaleChina(1.0);
        //西文宽度系数
        pTxtStyle->setXScaleWest(0.5);
        //字符间距
        pTxtStyle->setXIntFac(0.0);
        //行度系数
        pTxtStyle->setYIntFac(2.0);

        if ( pTextStyleTbl->add( syTxtId, pTxtStyle) != CDraft::eOk)
        {
            pTxtStyle->close();
            pTextStyleTbl->close();
            return;
        }
        pTxtStyle->close();
    }
    pTextStyleTbl->close();

}
```

上面的代码中使用的是 TrueType 字体，如果电子图板自身的 SHX 字体，就无需指定字体文件的扩展名，例如：

```
pTxtStyle->setEFont(_T("simsun"));
```

此外，字体的名称不一定与字体文件的名称相同。打开控制面板，进入“字体”文件夹，右键单击“仿宋体”图标，从弹出的快捷菜单中选择【属性】菜单项，系统会弹出如图 4.5 所示的对话框，显示了字体文件的名称。

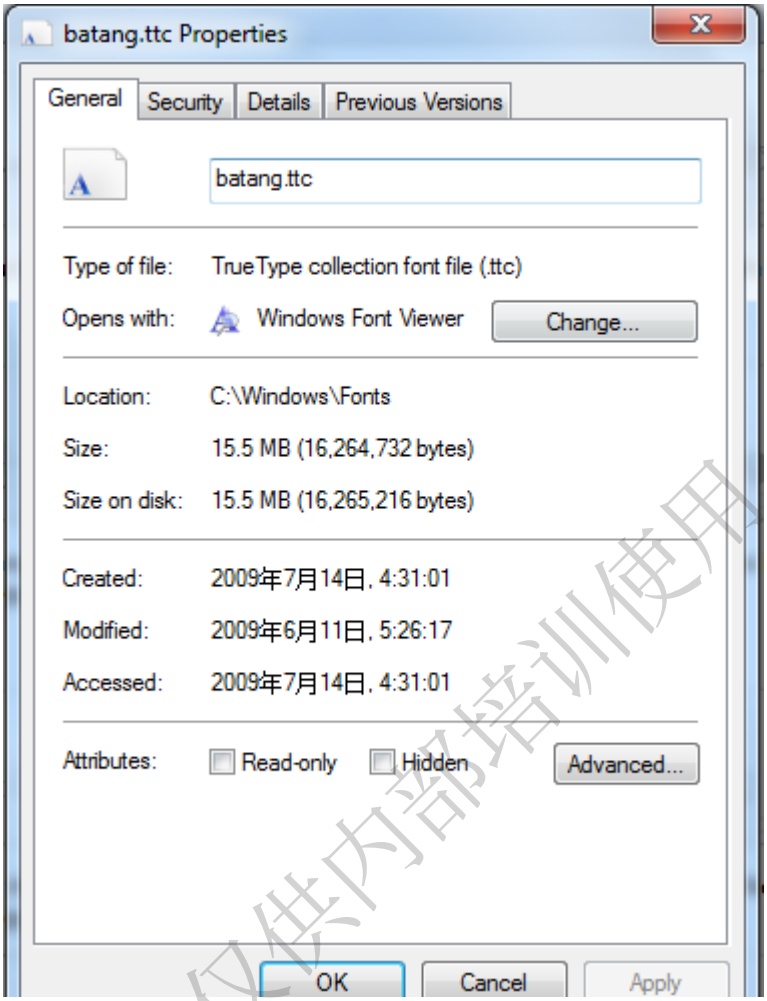

图4.5 查看字体的文件名

4.2.4 效果

编译运行程序，在电子图板 2011 中执行 AddStyle 命令，完成字体样式的创建。选择【标注/样式管理器 / 文字样式】菜单项，系统会弹出如图 4.6 所示的【字体样式】对话框。在【样式名】列表中，已经包含了新建的【newTxtStyle】样式。

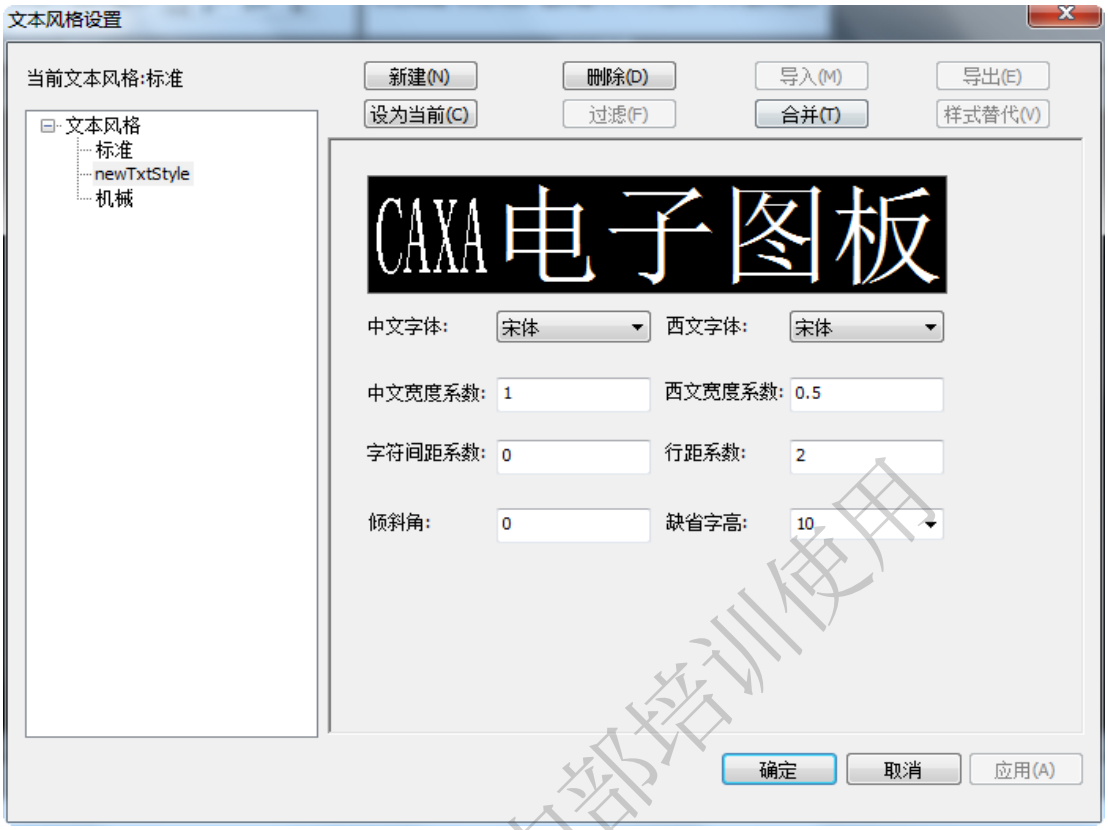

图4.6 查看新建的字体样式

4.2.5 小结

学习本节内容之后，读者杨掌握下面的要点：

- 设置字体样式的方法。
- 查看某一字体的字体文件名称。

4.3 创建标注样式

4.3.1 说明

本实例能够创建一个新的标注样式，在程序中设置了标注样式的名称、箭头大小、尺寸界线超出尺寸线的长度、文字和标注线的位置关系，以及标注文字的高度。

4.3.2 思路

与创建文字样式类似，在 ObjectCRX 中创建标注样式可以按照下面的步骤进行：

（1）创建一个新的标注样式表记录对象；

（2）设置标注样式表记录的各项特性，例如标注样式的名称、文字高度、箭头大小等；

（3）将新的标注样式表记录添加到当前图形的标注样式表中。

4.3.3 步骤

在 Visual Studio 2010 中，使用 ObjectCRX 向导创建一个新项目，命名为 AddDimStyle。注册一个命令 AddDimStyle，用于创建新的标注样式，其实现函数为：

```
void CRXAddDimStyle()
{
    // 获得要创建的标注样式名称
    CxCHAR styleName[100];
    if (crxedGetString(CAXA::kFalse, _T("\n输入新样式的名称："),styleName) != RTNORM)
    {
        return;
    }
    // 获得当前图形的标注样式表
    CRxDbDimStyleTable *pDimStyleTbl;
    crxdbHostApplicationServices()->workingDatabase()
        ->getDimStyleTable(pDimStyleTbl, CRxDb::kForWrite);
    if (pDimStyleTbl->has(styleName))
    {
        pDimStyleTbl->close();
            return;
    }
    // 创建新的标注样式表记录
    CRxDbDimStyleTableRecord *pDimStyleTblRcd;
    pDimStyleTblRcd = new CRxDbDimStyleTableRecord();
    // 设置标注样式的特性
    pDimStyleTblRcd->setName(styleName); // 样式名称
    pDimStyleTblRcd->setDimasz(3); // 箭头长度
    pDimStyleTblRcd->setDimexe(3); // 尺寸界线与标注点的偏移量
        pDimStyleTblRcd->setDimtad(1); // 文字位于标注线的上方
        pDimStyleTblRcd->setDimtxt(3); // 标注文字的高度
    // 将标注样式表记录添加到标注样式表中
    pDimStyleTbl->add(pDimStyleTblRcd);
    pDimStyleTblRcd->close();
    pDimStyleTbl->close();
}
```

4.3.4 效果

（1）编译运行程序，在电子图板 2011 中执行 AddDimStyle 命令，输入 MyStyle 作为新建的标注样式的名称，按下 Enter 键完成标注样式的创建。

（2）选择【标注/样式管理器 / 尺寸】菜单项，系统会弹出如所示的【标注风格设置】，在标注样式列表中已经包含了新建的标注样式。

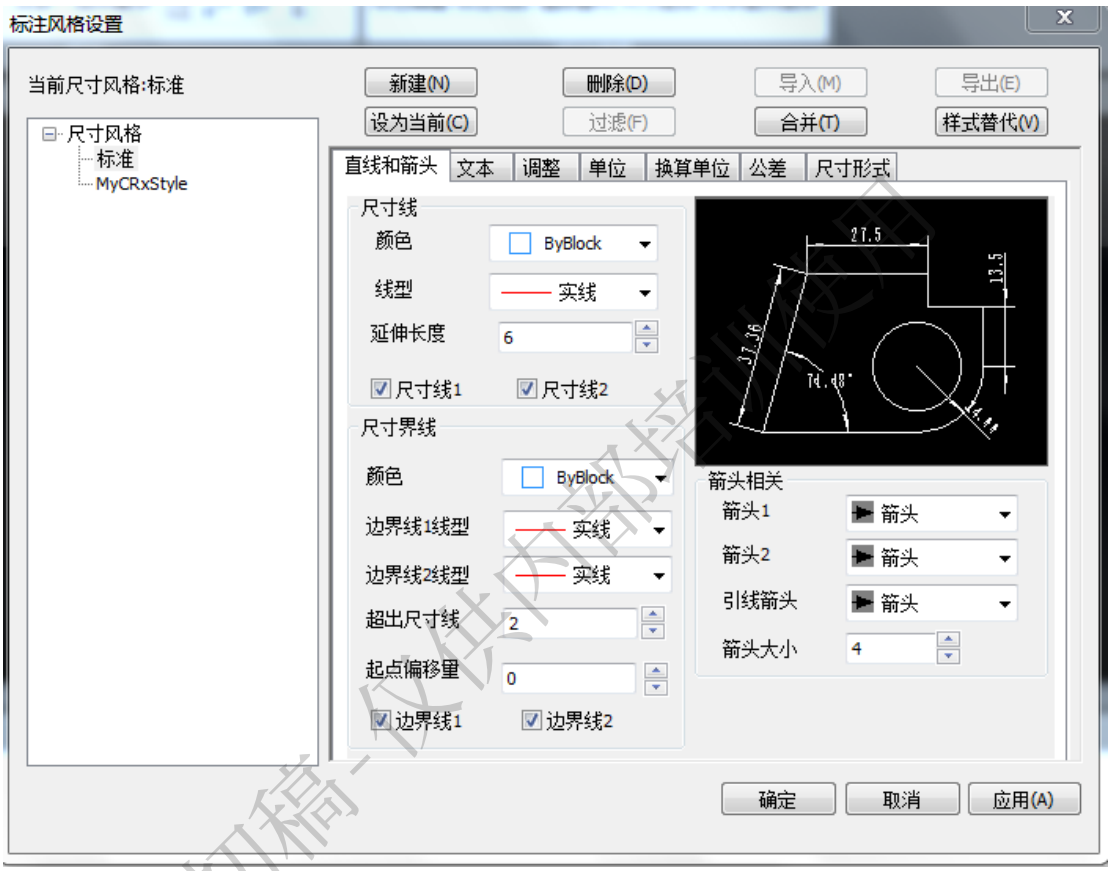

图4.7 创建了新的标注样式

（3）从样式列表中选择 MyStyle 样式，可以查看该样式的各种特性。

4.3.5 小结

在电子图板中创建标注样式时，用户可以选择一种已经存在的标注样式作为基础样式，如图 4.9 所示。创建新的标注样式之后，新样式的所有特性与基础样式保持一致，用户只需按照自己的要求对部分特性进行修改即可，非常方便。

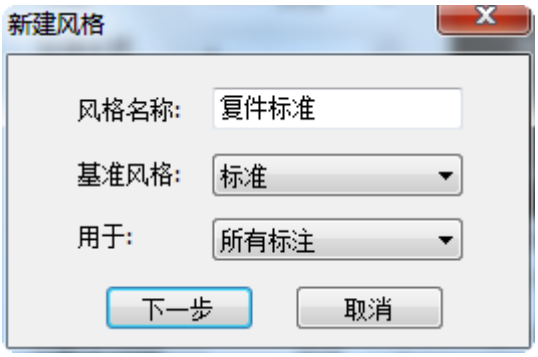

图4.9 选择基础样式

因此，在设置新标注样式的特性时，可以先获得系统中存在的标注样式，用已有的标注样式的特性来设置新的标注样式即可。下面的代码演示用已有的标注样式的部分特性来设置新建的标注样式：

```
// 创建新的标注样式表记录
CRxDbDimStyleTableRecord *pDimStyleTblRcd;
pDimStyleTblRcd = new CRxDbDimStyleTableRecord();
// 获得已经存在的标注样式ISO-25
CRxDbDimStyleTableRecord *pOldStyle;
pDimStyleTbl->getAt("ISO-25", pOldStyle, CRxDb::kForRead);
// 设置新标注样式的特性
pDimStyleTblRcd->setName(styleName);
pDimStyleTblRcd->setDimtxt(pOldStyle->dimtxt());
pDimStyleTblRcd->setDimasz(pOldStyle->dimasz());
pDimStyleTblRcd->setDimexe(pOldStyle->dimexe());
pDimStyleTblRcd->setDimtad(pOldStyle->dimtad());
```