第 6 章　扩展数据、扩展记录和对象字典

在编程中，某些时候不可避免地要向图形中添加一些用户数据，例如，将一条直线解释为输电线、道路中心线，或者其他类型的对象，也可能要将当前图形的编号随图形一起保存起来。

要向图形中的实体追加一些数据，可以使用扩展数据或者扩展记录；要向图形本身追加一些数据，则可以使用命名对象字典。

6.1　扩展数据

6.1.1 说明

对于同一种类型的实体，可能将其作为管线、道路中心线和建筑物轮廓线等。为了实现这个目的，就可以在实体上追加扩展数据，要获得某个实体类型的时候，就可以读取其扩展数据得到类型信息。

本节演示了向实体追加扩展数据和显示一个实体的扩展数据，并且加入了容错的处理：向已经包含扩展数据的实体添加扩展数据，程序会自动退出；要求显示不包含任何扩展数据的实体的扩展数据，程序同样会退出。

6.1.2 思路

扩展数据能被添加到任何实体上，由一个结果缓冲区链表组成，并且随 电子图板图形一起保存（电子图板不会使用扩展数据）。在许多情况下，扩展数据是向实体追加用户数据的一个有效途径，但是每个实体上所附加的扩展数据不能超过 16K。

CRxDbObject 类的 setXData 函数用于设置一个对象的扩展数据，其定义为：

```
virtual CDraft::ErrorStatus
CRxDbObject::setXData(const resbuf* xdata);
```

CRxDbObject 类的 xData 函数用于获取一个对象的扩展数据，其定义为：

```
virtual resbuf*
CRxDbObject::xData(const char* regappName = NULL) const;
```

任何一个应用程序都能将扩展数据附加到实体上，因此所有的扩展数据都需要一个惟一的应用程序名称，该名称不超过 31 个字符。为了注册一个应用程序，可以使用全局函数 crxdbRegApp。

初稿-仅供内部培训使用

6.1.3 步骤

（1）在 Visual Studio 2010 中，使用 ObjectCRX 向导创建一个新的项目，命名为 Xdata。注册一个命令 AddXData，用于向实体追加指定的扩展数据，其实现函数为：

```
void CRXAddXData()
{
    // 提示用户选择所要添加扩展数据的图形对象
    crx_name en;
    crx_point pt;
    if (crxedEntSel(_T("\n选择所要添加扩展数据的实体："), en, pt) != RTNORM)
        return;
    CRxDbObjectId entId;
    CDraft::ErrorStatus es = crxdbGetObjectId(entId, en);
    // 扩展数据的内容
    struct resbuf* pRb;
    CxCHAR appName[] = {_T("XData")};
    CxCHAR typeName[] = {_T("道路中心线")};
    // 注册应用程序名称
    crxdbRegApp(_T("XData"));
    // 创建结果缓冲区链表
    pRb = crxutBuildList(CRxDb::kDxfRegAppName, appName, //应用程序名称
        CRxDb::kDxfXdAsciiString, typeName, //字符串
        CRxDb::kDxfXdInteger32, 2,// 整数
        CRxDb::kDxfXdReal, 3.14, //实数
        CRxDb::kDxfXdWorldXCoord, pt, // 点坐标值
        RTNONE);
    // 为选择的实体添加扩展数据
    CRxDbEntity *pEnt;
    crxdbOpenCRxDbEntity(pEnt, entId, CRxDb::kForWrite);

    struct resbuf *pTemp;
    pTemp = pEnt->xData(_T("XData"));
    if (pTemp != NULL) // 如果已经包含扩展数据，就不再添加新的扩展数据
    {
        crxutRelRb(pTemp);
        crxutPrintf(_T("\n所选择的实体已经包含扩展数据！"));
    }
    else
```

初稿，仅供内部培训使用

```
    {
        pEnt->setXData(pRb);
    }
    pEnt->close();
    crxutRelRb(pRb);
}
```

在构建存储扩展数据的结果缓冲区时，除了应用程序名称对应的数据类型为 CRxDb::kDxfRegAppName 之外，其他的数据类型前缀均为 CRxDb::kDxfXd，所有的与扩展数据有关的数据类型均带有这个前缀。

在函数结束之前，记得删除结果缓冲区。

（2）注册一个 ViewXData 命令，用于查看指定的实体的扩展数据，其实现函数为：

```
void CRXViewXData()
{
    // 提示用户选择所要查看扩展数据的图形对象
    crx_name en;
    crx_point pt;
    if (crxedEntSel(_T("\n选择所要查看扩展数据的实体："), en, pt) != RTNORM)
        return;
    CRxDbObjectId entId;
    CDraft::ErrorStatus es = crxdbGetObjectId(entId, en);
    // 打开图形对象，查看是否包含扩展数据
    CRxDbEntity *pEnt;
    crxdbOpenCRxDbEntity(pEnt, entId, CRxDb::kForRead);
    struct resbuf *pRb;
    pRb = pEnt->xData(_T("XData"));
        pEnt->close();
    if (pRb != NULL)
    {
        // 在命令行显示所有的扩展数据
        struct resbuf *pTemp;
        pTemp = pRb;
        // 首先要跳过应用程序的名称这一项
        pTemp = pTemp->rbnext;
        crxutPrintf(_T("\n字符串类型的扩展数据是：%s"),
            pTemp->resval.rstring);
        pTemp = pTemp->rbnext;
        crxutPrintf(_T("\n整数类型的扩展数据是：%d"), pTemp->resval.rint);
        pTemp = pTemp->rbnext;
        crxutPrintf(_T("\n实数类型的扩展数据是：%.2f"),
```

初稿，仅供内部培训使用

```
                pTemp->resval.rreal);
            pTemp = pTemp->rbnext;
            crxutPrintf(_T("\n点坐标类型的扩展数据是：(%.2f, %.2f, %.2f)"),
                pTemp->resval.rpoint[X], pTemp->resval.rpoint[Y],
                pTemp->resval.rpoint[Z]);
            crxutRelRb(pRb);
        }
        else
        {
            crxutPrintf(_T("\n所选择的实体不包含任何的扩展数据！"));
        }
}
```

使用 CRxDbObject 类的 xData 函数能够获得一个结果缓冲区链表，该实体的所有扩展数据都保存在该链表中，因此可以通过遍历结果缓冲区的方法获得扩展数据。

6.1.4 效果

（1)编译运行程序,在电子图板 2011 中,使用 LINE 命令创建两条直线。执行 AddXData 命令，选择其中的一条直线，为其添加扩展数据。

（2）再次执行 AddXData 命令，仍然选择已经添加扩展数据的那条直线，系统会在命令行给出提示：所选择的实体已经包含扩展数据！

（3）执行 ViewXData 命令，选择未添加扩展数据的直线，系统会在命令窗口给出提示：所选择的实体不包含任何的扩展数据！

（4）再次执行 ViewXData 命令，选择已经添加扩展数据的直线，系统会在命令窗口提示：

```
命令: VIEWXDATA
启动执行命令："viewxdata"
选择所要查看扩展数据的实体:
字符串类型的扩展数据是：道路中心线
整数类型的扩展数据是：2
实数类型的扩展数据是：3.14
点坐标类型的扩展数据是：(0.00, 0.00, -2.00)
```

6.1.5 小结

学习本节内容之后，读者应该掌握下面的要点：

- 向实体追加各种类型的扩展数据。
- 判断对象是否已经包含扩展数据。
- 遍历结果缓冲区链表的方法。

初稿-仅供内部培训使用

6.2　有名对象字典

6.2.1　说明

扩展记录与扩展数据类似，但是其数据存储量和能够存储的数据类型都要多于扩展数据。扩展记录可以保存在实体的扩展字典或有名对象字典中。

有名对象字典直接保存在图形数据库中，不与特定的实体有关，因此可用于保存与实体无关的设计参数。

字典与符号表类似，其中包含一个惟一的字符串关键字索引和对象 ID 号，通过关键字来访问字典中保存的内容。

本节的实例分别在实体和图形中保存与上节实例相同的数据，让读者对比扩展数据和扩展记录使用的异同，更好地理解这两种保存数据的机制。

6.2.2　思路

1．访问有名对象字典

电子图板每个图形数据库中都包含一个有名对象字典，默认情况下该字典中包含了组、多线样式、布局和打印等信息。例如，用户在电子图板创建一个组，就会有一个代表改组的元素被添加到组字典中。

如果需要在有名对象字典中保存自己的数据，一般可以在有名对象字典中添加一个根字典，然后再向根字典中添加新的字典，进而在新字典中保存数据。这样的好处是不会与有名对象字典的基本字典相混淆。

使用 CRxDbDatabase 对象的 getNamedObjectsDictionary 函数可以获得图形的有名对象字典（根字典），可以通过 setAt 函数向根字典添加一个字典，或者通过 getAt 函数获得其中的一个字典。获得字典之后，向字典中保存数据的方法与扩展字典完全一致。

6.2.3　步骤

（1）注册一个新命令 AddNameDict，向当前的图形数据库中添加一个字典，并在其中保存自定义数据，其实现函数为：

```
void CRXAddNameDict()
{
    // 要在扩展记录中保存的字符串
    CxCHAR entType[] = {_T("直线")};
    struct resbuf *pRb;


        // 获得有名对象字典，向其中添加指定的字典项
```

初稿仅供内部培训使用

```
        CRxDbDictionary *pNameObjDict, *pDict;
    crxdbHostApplicationServices()->workingDatabase()
        ->getNamedObjectsDictionary(pNameObjDict,
        CRxDb::kForWrite);
    // 检查所要添加的字典项是否已经存在
    CRxDbObjectId dictObjId;
    if (pNameObjDict->getAt(_T("MyDict"), (CRxDbObject*&)pDict,
        CRxDb::kForWrite) == CDraft::eKeyNotFound)
    {
        pDict = new CRxDbDictionary;
        pNameObjDict->setAt(_T("MyDict"), pDict, dictObjId);
        pDict->close();
    }
    pNameObjDict->close();
    // 向新建的字典中添加一个扩展记录
    CRxDbObjectId xrecObjId;
    CRxDbXrecord *pXrec = new CRxDbXrecord;
    crxdbOpenObject(pDict, dictObjId, CRxDb::kForWrite);
    pDict->setAt(_T("XRecord"), pXrec, xrecObjId);
    pDict->close();
    // 设置扩展记录的内容
    crx_point pt;
    pt[X] = 100;
    pt[Y] = 100;
    pt[Z] = 0;
    pRb = crxutBuildList(CRxDb::kDxfText, entType,
        CRxDb::kDxfInt32, 12,
        CRxDb::kDxfReal, 3.14,
        CRxDb::kDxfXCoord, pt,
        RTNONE);
    pXrec->setFromRbChain(*pRb);
    pXrec->close();
    crxutRelRb(pRb);
}
```

从上面的代码可以看出，在有名对象字典中保存数据与在扩展字典中保存数据相比，仅仅是获得字典的方法有所不同。

（2）注册一个新命令 ViewNameDict，检查当前图形中是否包含指定的用户字典，并在命令窗口中显示字典中保存的自定义数据，其实现函数为：

```
void CRXViewNameDict()
{
```

初稿-仅供内部培训使用

```
// 获得对象有名字典中指定的字典项
CRxDbDictionary *pNameObjDict, *pDict;
CDraft::ErrorStatus es;
crxdbHostApplicationServices()->workingDatabase()
    ->getNamedObjectsDictionary(pNameObjDict,
    CRxDb::kForRead);
es = pNameObjDict->getAt(_T("MyDict"), (CRxDbObject*&)pDict,
    CRxDb::kForRead);
pNameObjDict->close();
// 如果不存在指定的字典项，退出程序
if (es == CDraft::eKeyNotFound)
    return;
// 获得指定的对象字典
CRxDbXrecord *pXrec;
pDict->getAt(_T("XRecord"), (CRxDbObject*&)pXrec, CRxDb::kForRead);
pDict->close();
// 获得扩展记录的数据链表并关闭扩展数据对象
struct resbuf *pRb;
pXrec->rbChain(&pRb);
pXrec->close();
if (pRb != NULL)
{
    // 在命令行显示扩展记录内容
    struct resbuf *pTemp;
    pTemp = pRb;
    crxutPrintf(_T("\n字符串类型的扩展数据是：%s"),
        pTemp->resval.rstring);
    pTemp = pTemp->rbnext;


        crxutPrintf(_T("\n整数类型的扩展数据是：%d"), pTemp->resval.rint);
    pTemp = pTemp->rbnext;
    crxutPrintf(_T("\n实数类型的扩展数据是：%.2f"),
        pTemp->resval.rreal);
    pTemp = pTemp->rbnext;
    crxutPrintf(_T("\n点坐标类型的扩展数据是：(%.2f, %.2f, %.2f)"),
        pTemp->resval.rpoint[X], pTemp->resval.rpoint[Y],
        pTemp->resval.rpoint[Z]);
    crxutRelRb(pRb);
}
```

初稿-仅供内部培训使用

}

6.2.4 效果

（1）执行 AddNameDict 命令向图形的有名对象字典中添加用户字典，并且在其中保存自定义数据。执行 ViewNameDict 命令查看用户字典中保存的数据，能够在命令窗口中得到如图 6.1 所示的结果。

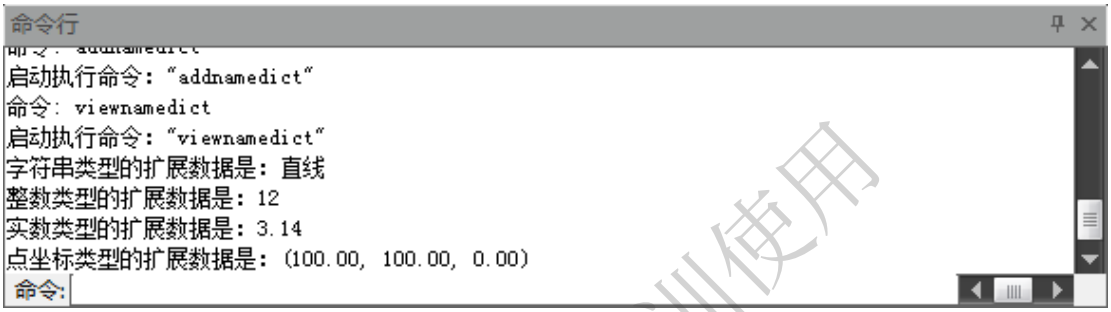

图6.1 查看有名对象字典中保存的数据

更有意思的是，即使在执行 ViewXRecord 和 ViewNameDict 命令之前保存并关闭了图形，再次打开图形仍然可以查询到这些数据。实际上，扩展记录所保存的数据已经被保存在图形数据库中。

6.2.5 小结

学习本节内容之后，读者需要掌握下面的几个知识点：

□ 使用对象命名字典来保存与对象无关联的数据。

扩展记录是 电子图板设计用来代替扩展数据的，因此在向对象附加数据的时候，应优先考虑使用扩展记录。

初稿仅供内部培训使用