AutoCAD作为国际上最流行的CAD绘图软件，在国内的应用日益广泛。在AutoCAD中通过一个称为图案填充的过程来填充闭合区域，以区分工程的部 件或用来表现组成对象的材质。例如，在规划行业的AutoCAD图纸中，地块、绿地、河流等通常采用图案填充来表示。但有时候为了使定义好的填充图案不再 被修改，通常把图案的填充边界删除或设置填充的属性为不关联。这样无论怎样拖动填充边界，填充实体都不会被改变。但是，如果图形不正确，需要再修改填充实 体的时候，或者需要获得该填充的边界信息（如在进行数据格式转换的时候），由于没有关联边界，将无法再改变填充实体，于是就不得不重新绘制。在 AutoCAD软件中并没提供重新生成非关联图案填充边界的命令和方法，但通过ObjectARX二次开发可以重新生成被删除的填充边界。

**1  图案填充边界的获取方法**

在AutoCAD ARX应用程序开发过程中主要通过两种方法访问AutoCAD中的图案填充边界，各有优缺点。

**1.1  通过读取AutoCAD数据库获取边界**

AutoCAD数据库是用来存储组成AutoCAD图形的对象和实体。AutoCAD图其实是一个存储在数据库中的对象的集合。通常打开AutoCAD数 据库通过选择获得对象，并且以ads\_name形式返回。然后，需要将ads\_name转换为AcDbObjectId并打开它。当用读的形式打开对象以 后，就能取得保存在数据库中的实体的所有基本信息。

图案填充在AutoCAD数据库中的是一个AcDbHatch对象，在打开对象以后，可以获取填充的第i个边界的对象ID值，然后通过该ID值来获得边界 的类型。每一个ID值唯一对应一条边界。采用该种方法，不用了解填充在AutoCAD中的结构，只需采用ObjectARX提供的接口方法，直接获得填充 边界的信息，比较简单方便。相关代码如下：

AcDbHatch \*eHatch；

AcDbObjectId eId；

acdbOpenObject(eHatch，eId，AcDb：：kForRead)；

AcDbObjectIdArray dbObjIds；

eHatch->getAssocObjIdsAt(i，dbObjIds)；

但是发现，当eHatch->associative()返回值为0的时候，即图案填充的边界为非关联的时 候，eHatch->numLoops()返回值也为0，即eHatch的边界数为0。也就是说通过这种方法只有当图案填充为关联的时候才能取出它 的边界类型，否则无法重新生成边界。

**1.2  通过读取图案填充的DXF组码获取边界**

其实在AutoCAD中的实体，每个实体还有一个数据的缓冲区。此数据由AutoCAD管理和维护，用户对图元操作后，AutoCAD将对这部分数据编辑 修改和保存。可以通过分析实体的结果缓冲区实现获得图案填充的边界类型。实体的结果缓冲区可以由ObjectARX函数acdbEntGet()返回，如 果该实体还有扩展数据，可以调用acdbEntGetX()函数获得扩展数据。ObjectARX定义了一个名为resbuf的通用结果缓冲区，如下所 示：

union ads\_u\_val {

ads\_real rreal；

ads\_real rpoint[3]；

short rint；

char \*rstring；

long rlname[2]；

long rlong；

struct ads\_binary rbinary；

}

struct resbuf {

struct resbuf \*rbnext；//用于链接下个结果缓冲区。

short restype；//用于指定哪种类型的值被存储在该缓冲区的resval字段。

union ads\_u\_val resval；//存储数据

}

ObjectARX处理实体的函数（acdbEntGet）所返回的结果中restype字段包含有DXF组码，通过DXF组码能够很容易的读出填充图案的边界，因为DXF组码只反映当前实体的所有信息，不随填充是否有边界而改变，它保存了填充边界的所有信息。

acdbEntGet()所检索到的一个图案填充的结果缓冲区链表如图1所示。DXF组码“0”表示实体类型码，图案填充的实体类型为“HATCH”；“2”表示图块名称；其余类型码的含义见图1中所示。

图1  图案填充的结果缓冲区链表

图1中每个结果缓冲区由三行组成，第一行表示地址，第二行表示结果类型码（即DXF组码），第三行表示包含的数据。该填充的结果缓冲区未全部表示出来。

由此可见，通过查询图案填充的DXF码结构可以轻易的获取图案填充的边界信息，而且无论该图案填充是否存在关联边界。

该种方法比较烦琐，开发人员必须清楚了解图案填充在DXF中的存储结构，较第一种方法慢。但是只有这种方法才能完全读取图案填充边界的信息，从而重新生成填充边界。

**2  获取图案填充边界的样例**

**2.1  图案填充边界DXF组码解析**

由于只需要获得图案填充的边界信息，所以只需要完全解析图案填充的边界DXF组码即可。通过查阅《AutoCAD Customization Guide》可以解析图案填充的DXF组码如表１、表２所示。

表1  图案填充边界路径数据组码

|  |  |
| --- | --- |
| 图案填充边界路径数据组码 | |
| 组码 | 说明 |
| 92 | 边界路径类型标志（按位编码）： 0 = 默认；1= 外部；2 = 多段线 4 = 导出；8 = 文本框；16 = 最外层 |
| 不确定 | 多段线边界类型数据（仅当边界 = 多段线时）。参见下面的多段线边界数据表 |
| 93 | 该边界路径中的边数（仅当边界不是多段线时） |
| 72 | 边类型（仅当边界不是多段线时）： 1 = 直线；2 = 圆弧；3 = 椭圆弧；4 = 样条曲线 |
| 不确定 | 边类型数据（仅当边界不是多段线时）。  参见下面的相应边数据表 |

表2  直线变数据组码

|  |  |
| --- | --- |
| 直线边数据组码 | |
| 组码 | 说明 |
| 10 | 起点（在 OCS 中） DXF：X 值；APP：二维点 |
| 20 | DXF：起点的 Y 值（在 OCS 中） |
| 11 | 端点（在 OCS 中） DXF：X 值；APP：二维点 |
| 21 | DXF：端点的 Y 值（在 OCS 中） |

以上只列出了直线边的数据组码，通过查阅《AutoCAD Customization Guide》可以获得其它边界数据的组码，如弧边、椭圆边和样条曲线边等。

通过下述全局函数可以返回图案填充的结果缓冲区eb。

struct resbuf \* acdbEntGet(const ads\_name ent)；

结果类型码为92时表示以后地址开始读取边界类型。可有多个类型码为92的地址，表示该填充有多条边界。从缓冲区中找到结果类型码为92的链表，即 eb->restype==92的时候，就能通过对比图案填充边界数据组码中92的值来判断图案填充的边界类型。如果此时eb-> resval.rlong==1，就能判断此时边界类型为“外部”，然后就通过组成填充外部边界数据组码来决定下一步所读取的数据。在填充边界类型中， “外部”表示填充的边界由多种AutoCAD简单实体组成，如直线、圆弧等，不包括多段线。

**2.2  程序样例**

下面是填充边界为直线时的VC程序样例的部分代码。

*struct resbuf \*eb；*

*eb=acdbEntGet(ent)；   //获得指定实体的结果缓冲区*

*for (eb=entdata；eb!=NULL；eb=eb->rbnext){//循环读取结果缓存区，直到结果类型码为92*

*if((eb->restype==92)&&((eb->resval.rlong==1)){*

*eb=eb->rbnext；           //读取下一个地址*

*while(eb->restype!=93){     //直到组码为93*

*if(eb->rbnext!=NULL)*

*eb=eb->rbnext；*

*else*

*return；*

*}*

*loopsNum=eb->resval.rlong；       //获得组成边界的简单实体个数*

*while(eb->restype!=72){*

*if (eb->rbnext!=NULL)*

*eb=eb->rbnext；*

*else*

*return；*

*}*

*}*

*for (int i=0；i<loopsNum；i++，eb=eb->rbnext){*

*//判断边界类型1 = Line； 2 = Circular arc； 3 = Elliptic arc；4 = Spline*

*if(eb->restype==72){*

*if(eb->resval.rint==1){    //当边界类型码为1时，即边界为直线时*

*eb=eb->rbnext；//读取下个地址*

*AcGePoint3d spt，ept；*

*spt.x=eb->resval.rpoint[X]；\\读取组码为10的直线起点值*

*spt.y=eb->resval.rpoint[Y]；*

*spt.z=eb->resval.rpoint[Z]；*

*eb=eb->rbnext；*

*ept.x=eb->resval.rpoint[X]； \\读取组码为11的直线端点值*

*ept.y=eb->resval.rpoint[Y]；*

*ept.z=eb->resval.rpoint[Z]；*

*AcDbLine \*el=new AcDbLine(spt，ept)；\\按起始点生成直线*

*postToDatabase(el)；\\保存直线实体到AutoCAD数据库中*

*el->close()；*

*}*

*…//类似上面的代码，分别读取圆弧，椭圆弧，样条曲线边界*

*}*

*}*

*}*

通过大量的实验数据，发现当eb->restype==92且eb->resval.rlong的值为1、5的时候，后面的DXF组码都代表 “外部”边界类型；eb->resval.rlong的值为3、7、22的时候，后面的DXF组码都代表“多段线”边界类型。这在AutoCAD的 《DXF参考》中并没有列出。而且在外部边界类型为圆弧或椭圆弧，当eb->restype==73并且eb->resval. rint==0即弧为顺时针的时候，取出的起点角度和端点角度都需算逆时针的值，即应该用2∏减去该角度，否则圆弧边界与其他边界间无法正常连接。

**3  结束语**

AutoCAD为用户提供了多种方法来读取实体的具体信息，让用户更深层次的了解AutoCAD中实体的存储结构。本文介绍的获得完整的图案填充边界信息 的方法，可供提取其它复杂实体的信息时参考。由于AutoCAD具有广大的用户群，DWG格式的图形数据已成为GIS的重要数据源，只有深入分析这些复杂 实体的存储结构，才能真正实现DWG格式与其它多种GIS图形数据格式之间的无损转换，以减少GIS应用系统建立时的数据成本。

如果你打算升级你的ARX或者想在同一个IDE（譬如vs2010)编译多个版本的ARX,那么我希望这篇帖子对你有帮助  
首先你应该简单了解Objectarx开发的版本对应情况：  
R15   --- 2000-2002  --- objectarx 2000,2002   --- vc6.0  
R16   --- 2004-2006   --- objectarx 2004-2006   --- vc7.0  
R17   --- 2007-2009  --- objectarx 2007-2009   --- vc8.0  
R18   --- 2010-2012  --- objectarx 2010-2012   --- vc9.0  
R19   --- 2013-2014（目前）  --- objectarx 2013,2014   --- vc10.0  
  
**1.字符问题**  
这个是最普遍的问题  
R17及其以上版本，必须以UNICODE编译。意味着绝大部分的字符串都应当以\_T包围起来，譬如以前写的:

普通浏览复制代码

1. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gifacutPrintf ("hello");

代码应当改写为：

普通浏览复制代码

1. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gifacutPrintf (\_T("hello"));

固然也可以写成

普通浏览复制代码

1. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gifacutPrintf (L"hello");

不过这样在多重目标编译时候你不能向下兼容了。  
char 的类型需要改成TCHAR类型  
然后所对应的跟字符串有关的函数都需要做相对应的修改：  
譬如 strcpy 应当改为 \_tcscpy ; strcmp要改为\_tcscmp;或者相应的unicode函数。  
关于如何知道他们想对应的形式，可以用google搜索关键字：  
例如 "strcpy msdn"  
一般来说第一个链接应该就是：  
[http://msdn.microsoft.com/zh-cn/library/kk6xf663](http://bbs.mjtd.com/plugin.php?id=laoyang_wailianx&url=http://msdn.microsoft.com%2Fzh-cn%2Flibrary%2Fkk6xf663)(v=vs.80).aspx  
然后就可以得到相应的  
  
TCHAR.H     \_tcscpy  
未定义      strcpy  
\_MBCS       \_mbscpy  
\_UNICODE    wcscpy  
  
因而，你应当包含TCHAR.H  
不过一般情况下这个文件已经包含了。  
  
另外在vs2008或者vs2010上编译R16或者以下版本，可能要检查以下编译配置：  
C/C++->general-->use UNICODE Response files 这个地方可能要设置为No  
C/C++->Treat wchar\_t as Built-in Type  这个地方要设置为NO  
Linker的General也是如此。  
  
**2. for 循环问题**  
  
由 vc6升级来的程序，如下for循环可以在vc6下成立:

普通浏览复制代码

1. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.giffor (int i = 0;i< n;i++)
2. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif{...}
3. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gifreturn i;

但是对于其上版本，不成立，会出现编译错误：  
error C2065: 'i' : undeclared identifier  
for循环内定义的i一旦放在外面，则视作未定义符号。它的作用域仅仅在于循环内。  
  
因而有两种办法：  
第一种：修改代码，把 i定义到循环外面:

普通浏览复制代码

1. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gifint i;
2. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.giffor (i=0;i<n;i++)
3. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif...

第二种：修改C/C++编译配置，把force confromance in for loop scope 修改为NO   
建议按第一种来，修改代码，使得符合C++  标准。(VC6只有不到80%的符合C++标准)  
  
**3. 4430问题**  
error C4430: missing type specifier - int assumed. Note: C++ does not support default-int  
  
在VC6中，如果没有显示的指定返回值类型，编译器将其视为默认整型；但是vs2002及以上不支持默认整型. 这个错  
  
误，解决方法如下：  
打开project->BaseClasses properties->configuration->C/C++ ->Command Line，增加/wd4430选项。  
或者在文件里面（一般来说是stdafx.h)添加 #pragma warning(disable: 4430)  
  
**4. 版本定义问题**  
  
在一些程序中，必须将

普通浏览复制代码

1. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif#ifndef WINVER
2. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif#define WINVER 0x0400
3. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif#endif

修改为

普通浏览复制代码

1. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif#ifndef WINVER
2. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif#define WINVER 0x0501
3. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif#endif

因为有的版本的arx 最低版本为xp,意味着在windows 2000及其以下均不获支持。  
  
**5. 模式定义(.def)文件和一些库文件**  
在R2004的版本 上，基本不需要.def  
但如果为使得你的程序向下兼容，可以设置.def,并在附加依赖库上添加rxapi.lib  
另外，随着版本的升级，其中 objectarx的一些库文件也发生了变化。需用户自己检查。  
在高版本，基本可以不添加什么附加依赖项。  
  
**6.如何让多重目标的编译自动适应各个版本：**  
  
可以这样做：  
在project->BaseClasses properties->configuration->C/C++ ->;Preprocessor (预处理指令)  
中添加 ADS = 16 之类，16这里代表版本号R16。  
然后在代码中增加如下判断:  
#if (ADS<16)  
//R2004以下版本执行的代码  
#else  
//R2004以上版本执行的代码  
#endif  
  
**7.一些虚函数的变化和升级**  
譬如在自定义实体时可能碰到的错误：  
error C3248: 'AcGiDrawable::worldDraw': function declared as 'sealed' cannot be overridden by   
  
'XXX\_Entity::worldDraw'  
  
你应当加如下判断：

普通浏览复制代码

1. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif//ADS按照上面设置
2. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif#if(ADS>17)
3. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif        virtual Adesk::Boolean subWorldDraw(AcGiWorldDraw \*mode) ;
4. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif        virtual void subViewportDraw(AcGiViewportDraw \* mode);
5. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif#else
6. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif        virtual Adesk::Boolean worldDraw(AcGiWorldDraw \*mode) ;
7. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif        virtual void viewportDraw(AcGiViewportDraw \* mode);
8. http://bbs.mjtd.com/source/plugin/imc_colorcode/images/jssc_none.gif#endif

R18及其以上版本用subWorldDraw代替了worldDraw，等等。  
  
**8.预编译头问题。**  
在一些旧的版本，CPP文件可能都按照默认设置为 Not using precompiled headers （不使用预编译头）  
应当改成 Use(/YU),stdafx.cpp改成 Create(/YC)  
这样一来可以极大地提高编译速度，二来减少一些升级时带来的问题。  
待续。。。

1、AutoCAD .NET API由不同的DLL文件组成，它们提供用于访问图形文件或AutoCAD应用程序的包含丰富的类、结构、方法和事件。每一个DLL文件都定义不同的使用基于功能的库组织组件的命名空间。

下面是你将频繁地要使用的AutoCAD .NET API 的三个主要的DLL文件:

**AcDbMgd.dll. 当使用图形文件中的对象时使用这个DLL文件。**

**AcMgd.dll. 当使用AutoCAD应用程序时使用这个DLL文件。**

**AcCui.dll. 当使用自定义文件时使用这个DLL文件。**

注意：在创建CAD项目时，需要引用到以上DLL，一般来说前面两个使用的较多。AutoCAD .NET API DLL被引用,就必须设置引用DLL文件的**“复制到本地”属性为False。**复制到本地属性确定当Microsoft Visual Studio编译工程时是否创建一个引用DLL文件的副本并将它放在与工程的程序集文件相同的目录下。由于引用文件已经是AutoCAD附属文件，因此， 创建引用文件的副本后当加载程序集文件到AutoCAD中时将可能引发意想不到的结果。

2、在.NET 类中加入CAD能调用的命令，必须用到CommandMethod属性,这个属性由Runtime命名空间提供。代码如下所示

[复制代码](javascript:void(0);)

1 Imports Autodesk.AutoCAD.Runtime

2 Imports Autodesk.AutoCAD.ApplicationServices

3 Imports Autodesk.AutoCAD.DatabaseServices

4

5 Public Class MyHello

6 <CommandMethod("Hello")> \_

7 Public Sub Hello()

8 '获取当前活动文档的Editor对象，

9 Dim Doc As Autodesk.AutoCAD.ApplicationServices.Document = Application.DocumentManager.MdiActiveDocument

10 Dim Ed As Autodesk.AutoCAD.EditorInput.Editor = Doc.Editor

11 Ed.WriteMessage("Hello")

12

13 End Sub

14 End Class

[复制代码](javascript:void(0);)

3、CAD项目生成类库后,可在 AutoCAD 的命令提示符中,输入 *netload* 命令，并选中生成的DLL，加载CAD项目，即可在CAD中使用CommandMethod指定的命令。

4、CAD调试（以CAD2010、VS2010为例）：右键打开项目属性-->调试-->选择启动外部程序  \AutoCAD2010\acad.exe

断点不起作用如下

（解决方法摘自：http://www.cnblogs.com/junqilian/archive/2010/04/29/1724403.html）

断点不起作用的原因在于Visual Studio 2010在调试时启动了默认的debugger(v4.0),但是为了让我们的.net应用程序能在AutoCAD Map (或Revit)等上运行，我们需要.net 2.0 framework( 3.5也是运行在CLR 2.0之上的)。

解决办法有如下两个方案：

方案1

编辑宿主程序的config 文件(acad.exe.config, revit.exe.config等等)，这个文件在AutoCAD的安装目录下。在</configuration>前面加上下面的代码（CAD2010该行代码已被注释）：

<startup>

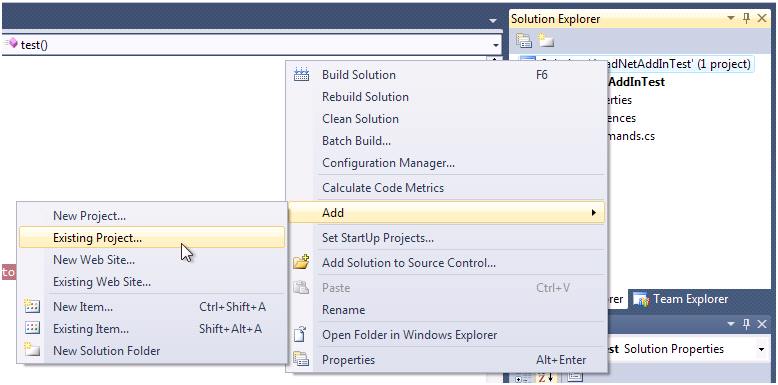
<supportedRuntime version="v2.0.50727" />

</startup>

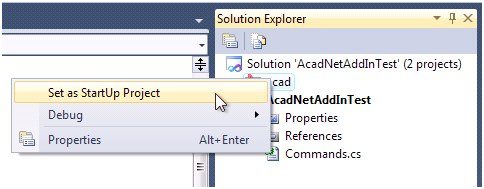
方案2

把宿主exe文件作为一个已有的项目加到您的解决方案中，并把debugger设置为v2.0

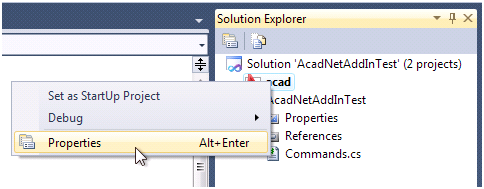
       右键单击Solution Explorer，选择Add->Existing Project,浏览到AutoCAD的启动文件Acad.exe



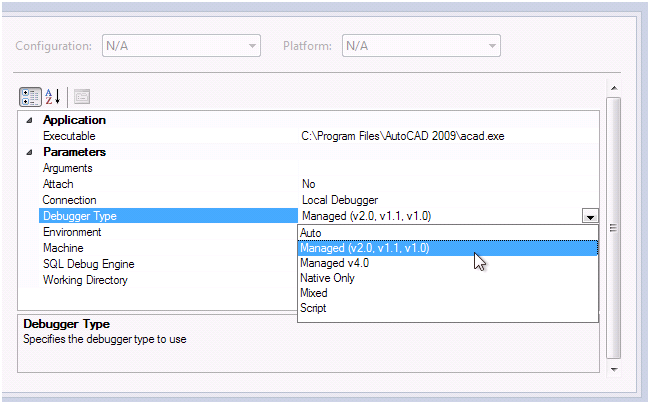
        右键单击该项目，把它设置为启动项目 Set as StartUp Project



        打开项目属性对话框 Properties



       设置 Debugger Type 为 Managed v2.0



5、对于不同的AutoCAD版本，推荐使用的.NET Framework版本如下：

**2007-2009 -> Framework 2.0**

2010-2011 -> Framework 3.5

2012-2013 -> Framework 4.0

##### [基于.NET的CAD二次开发学习笔记二：AutoCAD .NET中的对象](http://www.cnblogs.com/rliverpool/p/3949606.html)

1、CAD对象：一个CAD文件（DWG文件）即对应一个数据库，数据库中的所有组成部分，看的见（包括点、多段线、文字、圆等）和看不见（图层、线型、颜色等）的都是对象，数据库本身也是一个对象。

* 直线、圆弧、文字和标注等图形对象都是对象。
* 线型与标注样式等样式设置都是对象。
* 图层、编组和块等组织结构都是对象。
* 视图与视口等图形显示都是对象。
* 甚至图形和 AutoCAD 应用程序本身也是对象。

2、CAD中的容器：对于任何一个DWG文件，都会创建下列容器对像：九个固定的符号表（如下）和有名对像字典（包含Group字典和Mline样 式字典）。CAD中的所有对象都放在这些容器对象中，符号表及有名对象字典都具有同样的功能，即组织和管理CAD数据库中的对象。即它们含有实体对象，可 以通过文字字符串作为关键字对对象实体进行查询，并向这些容器添加记录（记录即对象），修改记录。

3、符号表与字典的不同：符号表如下所示有9种，每种符号表只能存储特定类型的对象(如块表只能存储块表记录，层表只能存储层表记录)，而字典则是一种通用容器，可存储任何类型的对象。（一般多是通过符号表操作对象）

**9种符号表：**

1)块表(BlockTabLe)

2)尺寸标注样式表(DimStyleTable)

3)层表(LayerTable)

4)线型表(LinetypeTable)

5)应用程序注册表(RegAppTable)

6)文字样式表(TextStyleTable)

7)用户坐标系表(UCSTable)

8)视口表(ViewportTable)

9)视图表(ViewTable)

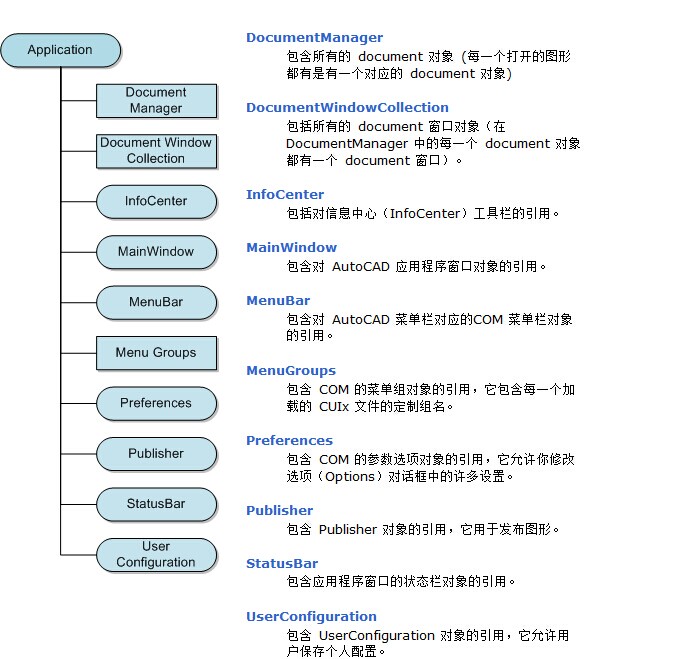
4、AutoCAD .NET API中的对象层次

1）Application对象层次

Application 对象是 AutoCAD .NET API 对象模型的根对象。通过 Application 对象，用户可以访问主窗口以及任何打开的图形。一旦用户获得了图形，就可以访问图形中的对象。

例如，Application 对象具有 DocumentManager 特性，该特性可以返回 DocumentManager 对象。**该对象提供对AutoCAD中当前图形的访问并允许用户并允许用户创建、保存和打开图形文件。**通过 Application 对象提供的其它特性可以专用数据，例如信息中心（InfoCenter）、主窗口（main window）和状态栏（status bar）。MainWindow 特性允许用户访问应用程序的名称、 大小、位置和可见性。

虽然 Application 对象的特性允许访问 AutoCAD .NET API 的大多数对象，也有一些 AutoCAD ActiveX® Automation 对象的引用。这些特性包括 application 对象的 COM 版本(AcadApplication)，菜单栏(MenuBar),加载的菜单组 (MenuGroups)和参数选项 (Preferences)。

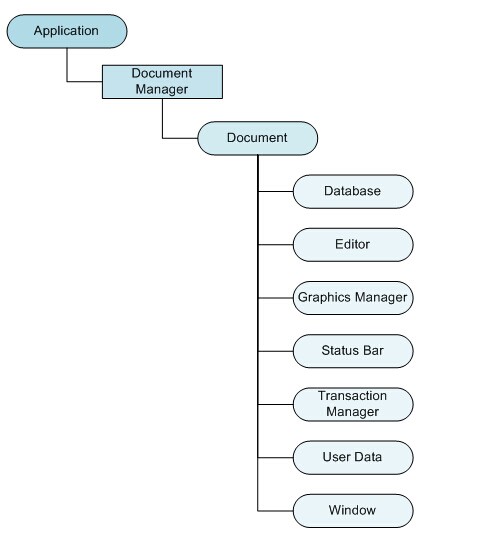


2) Document的对象层次

Document 对象实际上就是 AutoCAD 图形，它是 DocumentCollection 对象的一部分,并提供访问与 Document 对象相关的 Database 对象。 Database 对象包括所有的图形和大部分非图形的 AutoCAD 对象。

连同 Database 对象一起, Document 对象提供访问图形状态栏（StatusBar），document 对象被打开的窗口（Window），Editor （Editor） 和事务管理器（TransactionManager）的对象。Editor 提供用户以输入形式获取的点、输入字符串或数字值的功能。

事务管理器对象用于访问将多个 database 对象下的单个操作当作一个 transaction 对象。事务可以被嵌套，当用户完成事务时，可以提交或放弃已做的改变。



3)DataBase对象层次

Database 对象包括所有的图形和大部分非图形的 AutoCAD 对象。 被包含在 Database 中的一些对象有实体，符号表，命名字典。实体在 Database 中表示图纸内部的图形对象。直线、圆、圆弧、文本、填充和多段线都是实体的一个例子。用户可以在屏幕上看到一个实体并可以操作它。

用户通过 Document 对象的 Database 成员属性可以访问当前文档的 Database 对象。

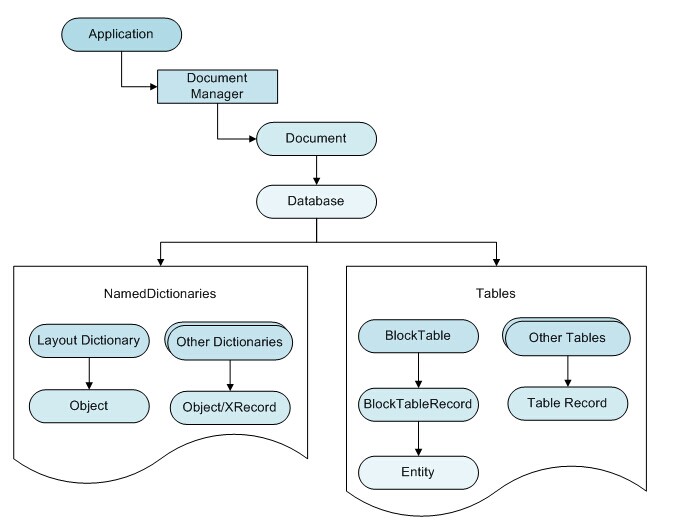
Application.DocumentManager.MdiActiveDocument.Database

符号表和字典

符号表和字典对象提供对非图形对象的访问（块、图层、线型、布局等等）。在AutoCAD中的每一个图形都包含9个固定的符号表，然而图形中的字典的数量是可以根据功能和应用程序的使用类型来改变的。新的符号表不能被添加到 Database 中去。

例如，层表 (LayerTable)就是符号表，它包含层表记录，块表也是符号表，它包含块表记录。所有的图形实体 (直线、圆、圆弧等等) 都是属于一个块表记录。默认情况下，每一个图形都为模型空间和图纸空间预定义了块表记录。每一个图纸空间布局都有它自己的块表记录。

字典是一个可以包含任何 AutoCAD 对象或 Xrecord 对象的容器。



4）图形对象，也称为图元，是组成图形的可见对象（例如直线、圆、光栅图像等）。添加图形对象到图形中是通过正确的块表记录引用，然后使用 AppendEntity 方法将一个新对象添加到图形中来完成的。

要修改或查询这些对象，请从正确的块表记录获得对象引用，然后使用对象本身的方法或特性。每一个图形对象都拥有允许应用程序执行大部分 AutoCAD 编辑命令的方法，例如复制、删除、移动、镜像等。

这些对象还提供了一些方法，用来设置和检索对象的扩展数据(xdata)，亮显和更新对象，以及从另外的图元设置属性。大多数图形对象具有诸如 LayerId、LinetypeId、Color 和 Handle 之类的典型特性。每个对象也具有一些特有的特性，例如 Center、StartPoint、Radius 和 FitTolerance。

非图形对象是指属于图形的一部分但不可见的（提示性的）对象，例如 Layers、Linetypes、DimStyles、TableStyle 等。若要创建新的符号表记录，请使用表所有者的 Add 方法或使用 SetAt 方法添加一个字典到命令对象字典中。要修改或查询这些对象，请使用对象本身的方法或特性。每一个非图形对象都有用于特定目的的方法和特性，都有设置和检索 扩展数据以及删除自己的方法。

    前不久在程序中遇到个问题，我要删除一个对象，明明代码也执行了，但是始终那对象删不掉，折腾了N久无果，后来把删除代码放到了事务处理外 面，发现居然可以了，继续折腾，终于找到原因，原来我有嵌套事务处理，但是在最外面一个事务处理中由于跳转语句，没有执行到Commit()，结果就是所 有的操作都滚回了，查了点资料。事务处理相关，见如下：

开始新事务并打开对象

**事务管理器是从当前 Database 的 TransactionManager 属性访问的。一旦对事务管理器进行了引用，就可以使用 StartTransaction 方法启动一个新事务。StartTransaction 会创建一个 Transaction 对象的实例并允许用户使用 GetObject 方法打开对象。**

**在事务结束的时候，所有在事务打开期间打开的对象都会被关闭。若要结束事务，请调用 Transaction 对象的 Dispose 方法。如果利用关键字 Using 和 End 表示事务的启动和结束，就不需要调用 Dispose 方法。**

**在 Transaction 销毁前，应用使用 Commit 方法提交所有的更改。如果 Transaction 在销毁前更改没有被提交，所有的修改都会回滚到先前 Transaction 启动时状态。**

[提交与回滚更改](http://www.01vb.com/Translate/NetGuide2010/files/WS73099cc142f48755f2fc9df120970276f7a46.htm#WS73099cc142f48755f2fc9df120970276f7a47)

在使用事务处理的时候，用户能够决定什么时候将修改过的对象保存到图形数据库中。**在一个事务内使用 Commit 方法保存一个对象已完成的修改。如果你的程序遇到了一个错误，就可以使用 Abort 方法回滚一个事务内所有的修改。**

**如果在没有调用 Commit 之前调用了 Dispose 方法，事务内所有的修改都会被回滚。无论 Commit 或 Abort 是否被调用，都必须调用Dispose 以标示事务的结束。如果 transaction 对象是以 Using 语句开始，就不必调用 Dispose。**

同时可以启动多个事务。活动事务的数量可以通过检索 TransactionManager 对象的 NumberOfActiveTransactions 属性得到，而最近创建的事务可以通过 TopTransaction 属性检索到。

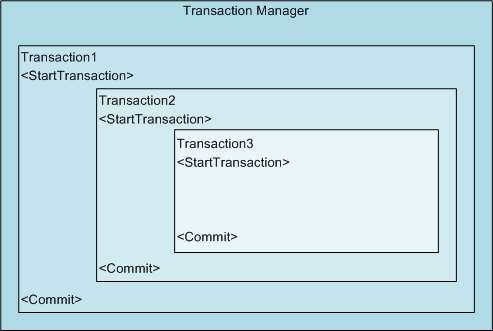
事务可以由一个嵌套到另一个中，以回滚某些程序执行期间所做的修改。

[嵌套事务](http://www.01vb.com/Translate/NetGuide2010/files/WS73099cc142f48755f2fc9df120970276f7a46.htm#WS73099cc142f48755f2fc9df120970276f7a47)

事务可以由一个嵌套到另一个中。外部的事务能够撤消你的程序所做的所有修改而内部事务仅仅能撤消部分修改。当你使用嵌套事务时，启动顶部事务也就等于启动外部事务。

当启动新事务时，它们也被添加进了前一个事务中。嵌套事务必须按与他们创建顺序相反的顺序提交或终止事务。因此，如果你有三个事务，在关闭第三个前就必须先关闭第三个或者说是最里面的一个，最后关闭第一个。如果终止了第一个事务，通过三个事务所做的修改都会撤消。

下面的插图显示在嵌套时事务如何出现。

[](http://photo.blog.sina.com.cn/showpic.html#blogid=69e8fdf00100n2ku&url=http://s7.sinaimg.cn/orignal/69e8fdf0h95028bd46fe6)