ObjectARX 实用指南

——AutoCAD二次开发

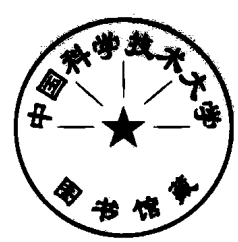
宋延杭　王川　李永宣　编著

人民邮电出版社

ObjectARX 实用指南

——*AutoCAD 二次开发*

宋延杭　王　川　李永宣　编著



人 民 邮 电 出 版 社

内 容 提 要

ObjectARX 2.0 是 AutoDesk 公司推出的基于 AutoCAD R14 的二次开发工具。本书是一本供读者学习 ObjectARX 2.0 的实用教程。全书共分为 19 章，从基本概念到实际应用，系统地介绍了在 AutoCAD 中运用 ObjectARX 进行二次开发的最新技术。

本书内容由浅入深，体系结构合理，理论与实例并重，适合广大 AutoCAD 用户和二次开发人员阅读，也可供具有一定的 C/C++语言基础的计算机爱好者和大专院校师生学习和参考。

Object ARX 实用指南——AutoCAD 二次开发

◆ 编　　著　宋延杭　王　川　李永宣
责任编辑　马　嘉

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号
北京顺义振华印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：787×1092　1/16
印张：27.75
字数：691 千字　　1999 年 8 月第 1 版
印数：1－6 000 册　　1999 年 8 月北京第 1 次印刷

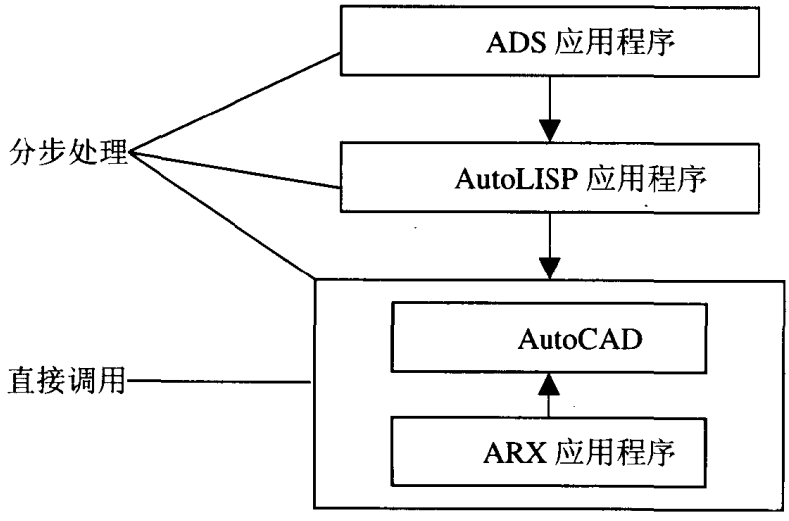
ISBN 7-115-07990-0/TP·1229

定价：41.00 元

前　言

ObjectARX 2.0 是 AutoDesk 公司随 AutoCAD R14 推出的新一代功能强大的 AutoCAD 二次开发工具。ARX 编程使用面向对象的 C++应用程序机制，一个 ARX 应用程序是以动态链接库的形式分享 AutoCAD 的地址空间的，并且可以被 AutoCAD 直接调用。这便于广大 AutoCAD 二次开发程序员能够更加充分地利用 AutoCAD 的开放结构，使程序员能够更直接地完成对 AutoCAD 的深层数据结构、图形系统编程和用户命令的定义。在这一点上，它不同于程序员所熟悉的 AutoLISP 和 ADS。

AutoLISP 是一种解释性语言，它使得程序员可以很方便地为 AutoCAD 增加新的命令。但是从逻辑上讲，AutoLISP 是以一种分步的方式与 AutoCAD 建立通信的。ADS 是使用 C 语言编程和编译的，但是对于 AutoCAD 来说，ADS 与 AutoLISP 的本质相同，因为实质上 ADS 是通过调用 AutoLISP 实现与 AutoCAD 的通信的。ARX 的编程环境与 ADS 和 AutoLISP 的编程环境完全不同，最关键的问题在于一个 ARX 应用程序是一个 DLL，它分享 AutoCAD 的地址空间并实现函数的直接调用，从而避免了解释过程，所以通常使用 ARX 编程的函数的执行速度要比使用 AutoLISP 或者 ADS 编程的函数快。AutoLISP、ADS 以及 ARX 与 AutoCAD 的关系如下图所示：



除了速度上的优势，安全性也是一个重要的方面。ARX 类库采用标准的 C++类库的封装形式，这大大提高了程序员编程的可靠性和效率。同时程序员可以为 ARX 类库增加新的类，这样在其它的应用程序中就可以直接使用。通过程序创造的图形实体在本质上与 AutoCAD 图形编辑器中生成的实体是一致的。

ARX 库的一部分实际上是一个完整的 ADS 函数库，它通常被称为 ADSRX，它在功能上与标准的 C 语言的 ADS 库是完全一致的，但是由于它实际上是被作为 AutoCAD 的一部

分执行的，所以它和其它的 ARX 库一样分享 AutoCAD 的地址空间。它通常被用来实现如下功能：

1. 实体和选择集的操作；
2. 选择集设置操作；
3. 可编程对话框；
4. AutoCAD 使用请求函数，如 ads_trans(), ads_command()和 ads_cmd() ；
5. 数据获取。

可以肯定地说，由于上述的种种优点，ARX 必将逐步取代 AutoLISP 和 ADS，成为 AutoCAD 二次开发的最重要的工具，目前国内在这方面的资料还比较匮乏，希望本书能为广大 AutoCAD 二次开发程序员提供帮助。

由于时间紧促，加之作者水平有限，书中难免存在疏漏之处，敬请关大读者朋友批评指正。

作　者

目　录

第一章　概述 1

1.1　ARX 简介 1

1.1.1　ARX 的库 1

1.1.2　AutoLISP、ADS 和 ARX 1

1.1.3　比较 ADS 和 ARX 的函数调用 3

1.1.4　运行类型的识别 4

1.2　数据库 5

1.2.1　AutoCAD 数据库 5

1.2.2　基本数据库对象 8

1.2.3　打开和关闭 ARX 对象 12

1.3　ARX 程序编译和运行环境 12

第二章　一个完整的 ARX 应用程序 17

2.1　ARX 应用程序的结构 17

2.1.1　acrxEntryPoint()的用法 17

2.1.2　AutoCAD 发给 ARX 应用程序的消息 18

2.2　登记新的命令 19

2.2.1　命令堆栈 19

2.2.2　查找次序 20

2.2.3　全球和地区命令名 20

2.2.4　透明命令 21

2.2.5　查询命令 22

2.2.6　应用实例 22

2.3　调用一个 ARX 应用程序 23

2.4　卸载 ARX 应用程序 23

2.5　需求调用 24

2.5.1　AutoCAD、Windows 系统注册表和 ARX 应用程序 25

2.5.2　在 ARX 应用程序的安装程序中修改注册表 25

2.5.3　需求调用系统变量 27

2.5.4　检测到用户对象时的需求调用 28

2.5.5　用户命令下的需求调用 29

2.5.6　AutoCAD 启动时的需求调用 29

2.5.7　使用系统注册表来管理应用程序 30

2.6　ARX 用户命令 30

2.7　错误处理 31

第三章　数据库操作 37
3.1　初始化数据库 37
3.2　创建和组织一个数据库 37
3.3　保存一个数据库 38
3.4　Wblock 操作 38
3.4.1　从一个已存在的数据库中创建一个新的数据库 38
3.4.2　创建含实体的数据库 38
3.5　插入一个数据库 39
3.6　设置当前数据库的值 40
3.6.1　数据库颜色值 40
3.6.2　数据库线型值 40
3.6.3　数据库线型比例值 40
3.6.4　数据库层值 41
3.7　外部引用 41
3.8　数据库操作实例 41
第四章　数据库对象 44
4.1　打开和关闭对象 44
4.2　删除对象 46
4.3　对象的数据库所有权 46
4.4　加入对象特征数据 47
4.4.1　扩展数据 47
4.4.2　扩充词典 50
4.4.3　ADS 的例子 52
4.5　擦除对象 56
4.6　对象归档 57
第五章　实体 58
5.1　实体定义 58
5.2　实体的从属关系 58
5.3　AutoCAD R14 的实体 60
5.4　共有的实体属性 62
5.4.1　实体颜色 62
5.4.2　实体线型 63
5.4.3　实体线型比例 64
5.4.4　实体的可见性 64
5.4.5　实体的层 64
5.5　共有的实体函数 65
5.5.1　对象捕捉点 66
5.5.2　变换函数 66
5.5.3　交叉点 67
5.5.4　GS 标记和子实体 68

5.5.5 炸开实体 85
5.6 创建 AutoCAD 实体实例 85
5.6.1 创建一个简单实体 85
5.6.2 创建一个简单的块表记录 86
5.6.3 创建一个带属性定义的块表记录 87
5.6.4 创建带有属性的块引用 90
5.6.5 遍历一个块表记录 94
5.7 复杂实体 96
5.7.1 创建一个复杂实体 96
5.7.2 遍历 polyline 的顶点 97
5.8 坐标系访问 98
5.8.1 实体坐标系 98
5.8.2 AcDb2dPolylineVertex 99
5.9 曲线函数 99
第六章 容器对象 101
6.1 符号表和词典的比较 101
6.2 符号表 103
6.2.1 块表 105
6.2.2 层表 105
6.2.3 遍历 108
6.3 词典 109
6.3.1 组和组词典 110
6.3.2 Mline 样式词典 112
6.3.3 创建一个词典 112
6.3.4 遍历词典 114
6.4 Xrecords 115
6.4.1 Xrecord 的 DXF 组码 116
6.4.2 实例 116
第七章 派生一个用户 ARX 类 121
7.1 用户类的派生 121
7.2 运行类的声明 122
7.3 声明类的宏 123
7.4 执行类的宏 123
7.5 类初始化函数 125
第八章 从 AcDbObject 类派生 126
8.1 重载 AcDbObject 类的虚拟函数 126
8.1.1 必须进行重载的 AcDbObject 虚拟函数 126
8.1.2 通常情况下需要重载的 AcDbObject 虚拟函数 126
8.1.3 偶尔需要重载的 AcDbObject 虚拟函数 127
8.1.4 很少需要重载的 AcDbObject 虚拟函数 127

8.1.5 很少需要重载的 AcRxObject 虚拟函数 129
8.1.6 需要重载的 AcDbEntity 虚拟函数 129
8.1.7 通常都重载的 AcDbEntity 虚拟函数 130
8.1.8 很少进行重载的 AcDbEntity 虚拟函数 132
8.1.9 需要重载的 AcDbCurve 虚拟函数 133
8.2 成员函数的实现 135
8.3 将对象存为 DWG 和 DXF 文件 136
8.3.1 dwgOut( ) 函数 137
8.3.2 dwgIn( ) 函数 137
8.3.3 dxfOut( ) 函数 138
8.3.4 dxfIn( ) 函数 138
8.3.5 错误检查 138
8.3.6 执行 DWG 存储函数 138
8.3.7 执行 DXF 存储函数 140
8.4 对象的引用 146
8.5 从属关系的引用 147
8.5.1 从属关系的应用 147
8.5.2 从属关系的类型 148
8.5.3 建立一个从属层次 148
8.6 指针引用 159
8.6.1 硬指针 159
8.6.2 软指针 159
8.7 撤消和重复 160
8.7.1 自动撤消 160
8.7.2 部分撤消 160
8.7.3 恢复 162
8.8 SubErase,subOpen, subClose 和 subCancel 函数 163
8.9 AcDbObject 类派生实例 176
8.9.1 头文件 176
8.9.2 源文件 177
8.10 对象版本支持 183
第九章 从 AcDbEntity 类派生 186
9.1 实体的显示 186
9.1.1 AcDbEntity::saveAs()函数的重载 188
9.1.2 AcGi 对象的有效范围 189
9.1.3 再生类型 189
9.1.4 设置实体特性 189
9.1.5 原语 195
9.1.6 镶嵌 207
9.1.7 异构线 207

9.1.8 视点 207
9.1.9 变形 208
9.2 基本实体函数 218
9.2.1 对象点捕捉函数的实现 219
9.2.2 对象点抓取函数的实现 221
9.2.3 对象点拉伸函数的实现 224
9.2.4 重载变形函数 226
9.2.5 重载相交函数 227
9.3 实体功能扩展 234
9.4 使用 ARX 的 AcEdJig 类 234
9.4.1 从 AcEdJig 中派生新类 235
9.4.2 使用 AcEdJig 类的基本步骤 235
9.4.3 顺序拖动的参数设置 235
9.4.4 拖动循环 235
9.4.5 实现 sampler( )函数、update( )函数和 entity( )函数 236
9.4.6 添加实体到数据库中 239
9.4.7 程序举例 239
第十章 代理对象 246
10.1 代理对象定义 246
10.2 代理对象的生命周期 246
10.3 用户与代理对象的关系 247
10.4 代理实体的显示 248
10.5 编辑代理实体 248
10.6 应用程序的卸载 249
第十一章 通告 250
11.1 什么是通告 250
11.1.1 反应器类 250
11.1.2 对象反应器的类型 251
11.2 使用反应器 251
11.2.1 AcDbobject 类和数据库通告事件 252
11.2.2 用户自定义通告 253
11.2.3 使用编辑反应器 253
11.2.4 使用数据库反应器 253
11.2.5 使用对象反应器 258
11.3 通告使用注意事项 268
第十二章 交易管理 270
12.1 交易管理简介 270
12.2 交易管理器 271
12.3 交易的嵌套 271
12.4 交易的边界 272

12.5 交易中对象指针的获取 272
12.6 交易中的对象新建 273
12.7 交易管理中的操作恢复 273
12.8 交易模型与打开/关闭机制的共存 274
12.9 交易管理中的图形创成 274
12.10 交易反应器 274
12.11 交易嵌套实例 275
第十三章 深层克隆 289
13.1 深层克隆概述 289
13.1.1 使用函数 clone( )和函数 deepClone( ) 289
13.1.2 克隆的基本概念 290
13.1.3 典型的深层克隆操作 291
13.1.4 克隆不同所有者中的对象 293
13.2 在用户类中使用 deepClone() 297
13.2.1 使用深层克隆的 AutoCAD 命令 297
13.2.2 克隆阶段 298
13.2.3 转换阶段 298
13.2.4 命名的对象字典 300
13.2.5 重载 deepClone( )函数 305
13.2.6 重载 wblockClone( )函数 309
13.2.7 追加方法：AcDbBlockTableRecord::appendAcDbEntity() 316
13.2.8 块克隆过程中 AcDbEntities 硬引用的处理 317
13.2.9 插入操作 319
13.2.10 编辑反应器通告函数 319
第十四章 协议扩展 324
14.1 协议扩展的定义 324
14.2 协议扩展的实现 324
14.2.1 声明和定义协议扩展类 324
14.2.2 注册协议扩展类 325
14.2.3 协议扩展的缺省类 326
14.2.4 卸载应用程序 327
14.2.5 协议扩展机能在应用程序中的使用 327
14.3 “MATCH”命令的协议扩展 327
14.4 协议扩展应用实例 327
第十五章 在 ARX 应用程序中使用 MFC 334
15.1 AutoCAD 中的 MFC 334
15.1.1 动态/静态链接的 MFC 库 334
15.1.2 MFC 创建无模态对话框 334
15.2 动态链接 MFC 的 ARX 应用程序 335
15.2.1 与 MFC 库动态链接的 Visual C++工程的设置 335

15.2.2 使用动态 MFC 调试 ARX 应用程序 335
15.2.3 资源管理 336
15.2.4 ARX 应用程序的升级 339
15.3 静态链接 MFC 的 ARX 应用程序 340
15.3.1 与 MFC 库动态链接的 Visual C++工程的设置 340
15.3.2 MFC 的初始化 340
15.3.3 应用程序对象 341
15.3.4 使用 AutoCAD 中的框架和视图窗口句柄 341
15.3.5 静态 MFC 链接的 ARX 应用程序举例 341
15.3.6 静态链接 MFC 库的应用程序的升级 347
第十六章 ActiveX 自动化 349
16.1 ActiveX 自动化的定义 349
16.2 创建 ActiveX 自动化 ARX 应用程序的基本步骤 349
16.3 ARX 对自动操作的支持 350
16.4 自动操作应用实例 350
16.4.1 ODL 文件（第二步） 350
16.4.2 类描述头文件（第三步） 352
16.4.3 文件注册（第四步） 356
16.4.4 初始化 ActiveX（第五步） 356
16.4.5 IDispatch 接口（第六步） 357
16.4.6 IAcadBaseObject 接口（第七步） 360
16.4.7 类代理坊（第八步） 362
16.4.8 析构器（第九步） 366
16.4.9 发布 ActiveX（第十步） 367
16.4.10 kOleUnloadAppMsg 消息响应（第十一步） 367
16.5 创建自定义实体的应用程序接口 369
16.6 ARX 应用程序与 AutoCAD 的运行期兼容性 371
16.7 ARX 应用程序的自动化方案 371
16.8 使用 Visual Basic 编程 371
第十七章 移植 ADS 程序到 ARX 373
17.1 为什么要移植到 ARX 373
17.1.1 ARX 入口函数 acrxEntryPoint 373
17.1.2 移植 ADS 应用程序必须包含的 ARX 头文件 374
17.2 在ARX 环境下加载应用程序 374
17.3 在 ARX 编程环境下创建应用程序 375
17.4 应用程序实例 375
第十八章 ObjectARX 的几何类库 383
18.1 几何类库简介 383
18.1.1 AcGe 类库的继承关系 383
18.1.2 AcGe 类库中的全局函数 386

18.1.3 几何类库中公差的处理 386
18.2 使用基本的几何类型 387
18.2.1 点和向量的操作 387
18.2.2 矩阵的操作 388
18.3 使用 Line 和 Plane 类 388
18.4 参数化几何类 390
18.4.1 参数化曲线 390
18.4.2 参数化表面 392
18.5 特殊的求值类 393
18.6 持续化的 AcGe 实体 398
第十九章 使用边界表述类库 404
19.1 使用 AcBr 库 404
19.2 拓扑和几何 405
19.3 用户程序中的基本元素 406
19.4 使用横断面 406
19.5 使用边界元素类 407
19.5.1 AcBrBrep 类 407
19.5.2 AcBrFace 类 408
19.5.3 AcBrLoop 类 408
19.5.4 AcBrEdge 类 408
19.5.5 AcBrVertex 类 409
19.6 横断面类 409
19.6.1 AcBrBrepFaceTraverser 类 409
19.6.2 AcBrFaceLoopTraverser 类 410
19.6.3 AcBrLoopEdgeTraverser 类 410
19.6.4 AcBrLoopVertexTraverser 类 410
19.6.5 AcBrEdgeLoopTraverser 类 411
19.7 错误返回码 411
19.8 应用实例 412
19.8.1 形体模型中拓扑元素和几何元素的数据访问 413
19.8.2 三维模型空间边界块的计算 418
19.8.3 创建 AutoCAD/ObjectARX/ADS 的应用程序接口 421

第一章 概述

1.1 ARX 简介

一个 ARX 应用程序是一个动态链接库，它分享 AutoCAD 的地址空间并可以直接被 AutoCAD 调用。因为出于扩展的目的，ARX 库包括很多的宏，它们将有助于用户定义新的类或者给现有的 AutoCAD 的类增加功能。ARX 库可以同 ADS（AutoCAD Development System）和 AutoLISP 相关联使用。ARX 编程环境是一个面向对象的 C++编程环境，它对用户使用、定制和扩展 AutoCAD 非常有利。ARX 库包含很多的开发工具，方便用户开发 AutoCAD 的开放环境、数据结构和图形系统。AutoCAD 的实时扩展编程环境提供功能强大的 C++库，它可以帮助用户扩展 AutoCAD 应用程序、AutoCAD 的类和协议，并且能创造出同 AutoCAD 内部命令类似的新命令。

1.1.1 ARX 的库

ARX 包括以下的库：

(1) AcRx

用以生成一个运行类的注册和声明的应用程序的类。

(2) AcEd

用以注册命令和系统事件通告的类。

(3) AcDb

AutoCAD 数据库类。

(4) AcGi

用以渲染 AutoCAD 实体的图形界面。

(5) AcGe

用于线性和集合对象的的应用库。

(6) ADS

用以生成 AutoCAD 应用程序的 C 语言库。通常 ARX 使用这个库来完成诸如实体选择、选择方式的设置和数据获取等功能。

1.1.2 AutoLISP、ADS 和 ARX

AutoLISP 是一种解释性语言，它提供了一个向 AutoCAD 增加命令的简单机制。虽然随着运行平台的不同会有些变化，但是从根本上讲它和 AutoCAD 之间是通过 IPC(interprocess communication)相联系的，如图 1-1 所示。

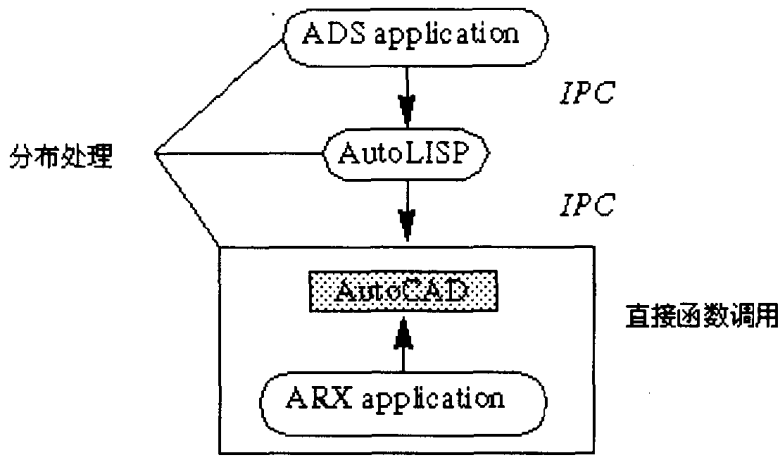


图 1-1 AutoLISP、ADS 和 ARX 与 AutoCAD 的关系

ADS 是使用 C 语言的应用程序，但是对于 AutoCAD 而言，ADS 应用程序同 AutoLISP 是相同的，一个 ADS 应用程序是以一系列外部函数，通过 AutoLISP 上载和调用的方式来完成其功能的，ADS 应用程序也是通过 IPC 同 AutoLISP 相联系的。

ARX 编程环境在很多地方与 ADS 和 AutoLISP 的编程环境是不同的。最重要的差异是一个 ARX 应用程序是一个 DLL，它分享 AutoCAD 的地址空间，被 AutoCAD 直接调用，这样就避免了使用 IPC 作为中介。这样，一个程序的运行速度就比 ADS 和 AutoLISP 要快。

除了速度的提高，用户还可以使用 ARX 向 AutoCAD 增加新的类，并向其它的应用程序输出。使用 ARX 创建的实体和用户在 AutoCAD 图形编辑环境中创建的实体是相同的。同时，用户可以在已经存在的 AutoCAD 类的基础上扩展 ARX 协议。

实际上，一个完整的 ADS 库函数是 ARX 环境的一部分，通常称为 ADSRX，它和标准的 C 语言版本的 ADS 库在功能上是完全一致的。但是作为 AutoCAD 的一部分，它同样分享 AutoCAD 的地址空间。通常使用 ADS 库实现如下的功能：

(1) 实体选择和构造选择集。

(2) 选择操作的设置。

(3) 可编程对话框。

(4) AutoCAD 应用请求，例如 ads_trans(), ads_command()和 ads_cmd()等命令。

(5) 数据获取。

用户可以在 ARX 中使用 ads_defun()函数和 acedRegCmds()宏来登记 AutoCAD 的新命令。使用 ADS 库命令，请求先发送给 AutoLISP，然后到达应用程序。使用 ARX 命令登记，该命令就加入到了 AutoCAD 的命令集里。

需要指出的是，ARX 和 ADS 同 AutoCAD 的通信方式是不同的。一个 ADS 应用程序包含一个单层的无限循环，它一直在等待着 AutoLISP 的请求。ARX 有一个为使用消息提供的入口。当用户登记命令时，它实现了同 AutoCAD 的通信；当用户重载在 ARX 库中的 C++ 类的虚函数时，这些函数就实现了应用程序的入口功能。

1.1.3 比较 ADS 和 ARX 的函数调用

通常，ARX 的 API 要比 ADS 的 API 简单。例如，在 ARX 中用户可以使用如下的代码来改变线的层：

```
void
changeLayer(const AcDbObjectId& entId,
    const char* pNewLayerName)
{
    AcDbEntity *pEntity;
    acdbOpenObject(pEntity, entId, AcDb::kForWrite);
    pEntity->setLayer(pNewLayerName);
    pEntity->close();

}
```

在 ADS 中，关于实体的信息是保存在一个结果缓冲区的链表("resbufs")中的，要改变线的层，有基本的四个步骤：

(1) 使用 ads_entget()函数获得实体信息；

(2) 寻找层的值；

(3) 改变链表中的值；

(4) 调用 ads_entmod()函数实现数据库在链表中的变化。

以下是改变线的层的 ADS 代码：

```
void
changeLayerADS(ads_name entityName,
    const char* pNewLayerName)
{
    struct resbuf *pRb, *pTempRb;
    pRb = ads_entget(entityName);

//实体层信息
//
    for (pTempRb = pRb; pTempRb->restype != 8;
        pTempRb = pTempRb->rbnext)
    { ; }

    free(pTempRb->resval.rstring);
    pTempRb->resval.rstring
        = (char*) malloc(strlen(pNewLayerName) + 1);
    strcpy(pTempRb->resval.rstring, pNewLayerName);

    ads_entmod(pRb);
```

```
    ads_relrb(pRb);
    ads_retvoid();

}
```

以下是实现同样功能的 AutoLISP 代码

```
(defun asdk_changeLayerLISP(ename newLayer / eList)

    (setq eList (entget ename))

    ; substitute the new layer name for the old
    ;
    (setq eList
        (subst (cons 8 newLayer) (assoc 8 eList) eList))

        (entmod eList)
    (princ)
)
```

1.1.4 运行类型的识别

每一个 AcRxObject 的子类都可以有一个描述符对象（AcRxClass 类型）与之相联系，这些描述符对象用以识别运行类型。ARX 提供函数用以测试一个对象属于一个基类或是一个派生的类。它帮助用户决定是否两个对象有相同的类，并返回类的描述符对象。由 AcRxObject 类提供的重要的函数来识别运行类型，包含如下的内容：

(1) desc()，静态成员函数，它返回一个特定的类的描述符对象。

(2) cast()，静态成员函数，它返回一个指定类型的对象；如果对象不是一个所需要的或者是一个派生的类，则返回 NULL。

(3) isKindOf()，返回一个对象是否属于指定的类，或者是派生的类。

(4) isA()，返回一个对象的类未知的描述符对象。

当用户一个对象属于什么类的时候，可以使用 AcRxObject::isA()。这个函数返回类的描述符对象（一个 AcRxClass 的实例）。它的语法是：

```
AcRxClass* isA() const;
```

如果用户已知一个对象的类，可以使用 desc()函数获得类描述符对象：

```
static AcRxClass* desc();
```

以下的例子是寻找一个 AcDbEllipse 的实例，也可以是从它派生出来的类的实例。使用 isKindOf()和 AcDbEllipse::desc()静态成员函数。

```
AcDbEntity* curEntity = somehowGetAndOpenAnEntity();

if (curEntity->isKindOf(AcDbEllipse::desc())) {
```