第一章　AutoCAD C语言开发系统——ADS

1.1　AutoCAD C语言开发系统简介

AutoCAD C语言开发系统——ADS(AutoCAD Development System)是AutoCAD R11最显著的功能之一，在AutoCAD R12中这一功能得到了进一步的增强。ADS使外部可执行文件与AutoCAD核心紧密地联系在一起，为在AutoCAD上进行二次开发提供了一个更为强有力的编程手段。

低版本AutoCAD的编程语言一直是AutoLISP语言。AutoLISP语言完全和AutoCAD环境溶为一体。用户可以随时使用AutoLISP语言。AutoLISP语言是被解释执行的，任何一个语句键入后就能马上执行，所以它对于交互式的程序开发是非常便利的。事实上，在AutoCAD的命令行提示状态下，我们仍然处于AutoLISP语言的“读入—求值—打印”循环中。

但AutoLISP语言又有严重的不足。AutoLISP语言的缺点之一是由于继承了LISP语言的编程规则而导致的繁多的括号，所以经常有人把“LISP”解释为“Lots of Insipid Stupid Parenthesis”，而不是它的本意“List Processing Language”。

在应用系统的开发过程中，许多常用的功能和手段是AutoLISP语言无法提供的，它们包括：

二进制文件的读写
在内存中相互通信
实时的数据库存取
直接的屏幕I/O操作
高级的用户接口
调用操作系统的功能
直接存取硬件设备
高强度的数据处理
程序的快速执行
软件与数据的加密保护

可以说，AutoLISP语言的根本限制是由于它完全包含在AutoCAD之中。

ADS的引入完全克服了上述的AutoLISP语言的缺点。ADS把外部可执行程序和AutoLISP语言连接起来，将它作为AutoLISP语言的外部函数来解释。由于ADS应用程序是外部可执行的应用程序，所以事实上它几乎可以完成任何事情。

用C语言编程，二进制的文件读写功能是C语言运行函数库的基本功能之一；在内存中相互通信只不过是把一个指针指向合适的内存位置，然后写入一些数据(当然，内存保护是需要考虑的)；数据库存取操作可以通过调用现有的数据库操作函数(如ASI接口)来实现；屏幕I/O操作是现代C语言的标准特征之一，很容易实现；高级用户接口也

很容易实现，可以利用各种现有的软件包，只要做适当修改，使其和AutoCAD的屏幕管理不冲突即可；至于运行速度，任何编译型的语言都要比解释执行的语言快许多。

ADS是一组可以用来用C语言编写AutoCAD应用程序的头文件和目标库．ADS应用程序是可以在AutoCAD环境中运行的可执行文件，它和AutoCAD建立通讯联接，向AutoCAD发出命令，并获得命令执行的结果。同时，ADS应用程序拥有C语言运行函数库的全部功能。

1.2 ADS应用程序开发环境

开发ADS应用所需的环境包括ADS提供的头文件和目标库，能支持ADS和AutoCAD平台的操作系统和编译开发工具。

1.2.1 ADS头文件和目标库

每一个ADS应用程序都应该包含三个头文件:

.ads.h

.adscodes.h

.adslib.h

为方便起见，在adslib.h中包含了ads.h和adscodes.h，所以在一个ADS应用程序中只要包含adslib.h就够了。

每个头文件都包含了不同的说明信息:

ads.h

1.ADS特有数据类型的定义

2.ADS库函数的函数原型说明

3.ads_initget()的位码的定义

4.其它定义

adscodes.h

1.对请求码、结果码、结果类型码的定义

2.其它定义

在AutoCAD软件包中，为不同的开发环境提供了不同的目标库，但没有提供目标库的源码．ADS应用程序必须连接合适的目标库才能运行．在1.3节中将指出各种不同的开发环境分别对应哪些目标库。

1.2.2 AutoCAD R11支持的编译器和连接器

AutoCAD R11只支持保护模式开发环境，包括:

.MetaWare High C 386(Version 1.6x，1.71，1.73，not2.3)

.Watocom C 386 7.0

.PharLap 386Link (Wlink也行)

1.2.3 AutoCAD R12支持的编译器和连接器

AutoCAD R12既支持保护模式开发环境，也支持实模式开发环境。

保护模式开发环境包括:

.Watcom C 386 9.0 (不再需要PharLap Linker)

.MetaWare High C 386 1.x或3.x

PharLap 386Link 4.1

MetaWare debugger MDB 1.3

.Zortech 386 C++ 3.x

PharLap 386Link 4.1

实模式开发环境包括:

.Borland C++ 2.0以上版本

.Microsoft C 6.0以上版本

1.2.4 AutoCAD R12 for Windows支持的编译环境

.Borland C++2.0以上版本

.Microsoft C++7.0

.Microsoft Quick C for Windows

.Watcom C 9.0

.Visual Basic 2.0

.MetaWare High C/C++3.0

1.3 创建ADS应用程序

创建ADS应用程序可在许多平台和环境下进行，但目前最常用的还是DOS平台。因为国内AutoCAD绝大多数都运行在386，486微机的DOS操作系统上，DOS下又有众多流行的C编译器可供选择，所以本节内容主要是针对DOS平台的，但是ADS的编程思想和方法对所有平台都是一致的。

如同标准的C函数库一样，ADS由库函数和头文件所定义。目标库和头文件都安装在AutoCAD的ADS子目录下。库文件根据所用编译器有不同的名字，但头文件都是一样的。

1.3.1 在保护模式开发环境下创建ADS应用程序

保护模式开发环境是AutoCAD最主要的程序开发环境。在微处理器的32位保护模

式下运行ADS应用程序，可以直接利用扩展内存，获得较快的运行速度，并且，在一次AutoCAD运行期间，允许同时装入多个保护模式的ADS应用程序，这些都是实模式ADS应用程序所不能比拟的。目前最流行的保护模式开发环境是MetaWare High C和Watcom C，其中High C还必需用扩展DOS支持。

下面我们将用详细的例子来说明如何编译并连接一个保护模式ADS应用程序(如果你想了解更多的编译细节，可以参考相应编译器的参考手册)。需要注意的是应该连接哪一个库:

ads.lib:	MetaWare High C的保护模式ADS库
wcads90.lib:	Watcom C的保护模式ADS库

*MetaWare High C 386 1.x或3.x

下面给出在MetaWare High C环境下创建ADS应用程序的批处理文件，所得可执行文件带有符号调试信息，便于用MetaWare High C的集成调试器MDB调试。

mkads.bat

```
if x%1x == xx goto usage
if exist %1.err del %1.err
if not exist %1.c goto nosource
if not exist %2\highc\bin\hc386.exe goto nocomp
%2\highc\bin\hc386 %1.c -c -g -I%2\highc\inc -Hobject=%1.obj
                                                    >%1.err
if errorlevel 1 goto error
%2\phar386\bin\386link -cvsym ads.lib %1.obj -lib
                      %2\highc\small\hcc -exe %1
echo off
echo
echo
echo %1.EXP created.
goto exit
:error
echo off
echo
echo **Error**
echo Error occured during the compilation!
echo Review %1.ERR for the details.
goto exit
:nosource
echo off
echo
echo **Error**
echo No %1.C source file found!
goto exit
:nocomp
```

```
echo off
echo
echo **Error**
echo No compiler found!
goto exit
:usage
echo off
echo
echo  Usage: mkads CCC [DDD]
echo     where CCC is your ADS C source file.
echo           DDD is an optional.  Drive which contains compiler
                                    and linker.
echo               Defaults to current drive.
echo
echo     Example: 1.  mkads jis
echo              2.  mkads jis f:      [f: drive contains
                                          compiler and linker]
:exit
echo
```

debugads.bat

```
@echo off
mdb acad
```

*Watcom C 386 9.0(不再需要PharLap Linker)

下面给出在Watcom C环境下创建ADS应用程序的批处理文件，所得可执行文件带有符号调试信息，便于用Watcom C的集成调试器wvideo调试。

mkads.bat

```
@echo off
set WATCOM=E:\WATCOM\.
set PATH=C:\DOS;E:\WATCOM\BIN;E:\WATCOM\BINB;C:\TOOLS;F:\ACAD12
set INCLUDE=E:\WATCOM\H
set ACAD=F:\ACAD12\SUPPORT;F:\ACAD12\FONTS;F:\ACAD12\ADS
set ACADCFG=F:\ACAD12\CFG
set ACADDRV=F:\ACAD12\DRV
set DOSX=-priv
@echo on
wcc386p -d2 %1 -fpi287 -3s -s -oailt
wlinkp debug all system ads file %1 library wcads90
```

debugads.bat

```
@echo off
wvideo /TRap=ADS /SWAP
```

1.3.2 在实模式开发环境下创建ADS应用程序

实模式开发环境是AutoCAD R12最新推出的支持Borland C和Microsoft C等常用编译器的开发环境。它使用和保护模式相同的头文件，但对于不同的编译器则有不同的库。实模式开发环境使得开发者可以使用自己熟悉的编译器开发实模式ADS应用程序。实模式ADS应用程序总是自动释放掉被标准C库函数malloc()、calloc()和realloc()分配的空间；在其它情况下，应用程序必须负责释放掉它所分配的空间。在文件处理上，应用程序应负责关闭它所打开的文件，否则打开的文件句柄会继续占据文件句柄表的空间，使得其它应用程序不能再打开文件。需要注意的是，实模式ADS应用程序必须用大模式编译。

下面我们将用详细的例子来说明如何编译并连接一个实模式ADS应用程序(如果你想了解更多的编译细节，可以参考相应编译器的参考手册)。需要注意的是应该连接哪一个库：

mscads6.lib:　Microsoft C的实模式ADS库
tbcads3.lib:　Borland C的实模式ADS库

*Microsoft C

你可以用Microsoft C 5.1或更高的版本编译ADS应用程序，编译器命令是cl，它同时也能够连接应用程序。举例说明，如果你的应用程序名是hello.c，则下面的命令行可以编译它并经连接产生最后的可执行文件hello.exe：

```
cl /AL /EM /DADS /FPi87 /Zi /Fehello hello.c /link llibc7
                                    mscads6.lib  /NOE /NOI /CO
```

各编译开关意义如下：

/AL　确定适用大模式编译，这对于实模式ADS应用程序是必须的
/EM　通知编译器在编译源程序时使用扩展内存
/DADS定义符号“ADS”，ADS的某些头文件将需要它
/FPi87产生协处理器专用浮点指令
/Zi　通知编译器在目标文件中包含行号和完整的符号信息
/Fehello将可执行文件取名为hello.exe
/link　通知编译器在完成编译后，自动连接并提供开关：
　　llibc7　　包括实模式运行函数库
　　mcsads6.lib　包括实模式ADS库
/NOE　禁止连接器搜索内部符号表
/NOI　保持大小写敏感性
/CO　通知编译器在可执行文件中包含调试信息

当应用程序调试完毕后，应该用以下开关对程序进行优化：

/Oailt 不考虑别名使用，产生内部函数，执行循环优化和速度优化

/Gs 取消堆栈检查

对于复杂的应用程序，应该使用Microsoft C的make工具进行编译和连接。

*Borland C

你可以用Borland C 2.0或更高版本编译实模式的ADS应用程序，编译命令是bcc，连接命令是tlink。假定我们还是编译hello.c程序，可用以下命令行：

```
bcc -I\ads;\borlandc\include -AT -1 -ml -vi -f287 -N- -G
    -DTURBOC -DPROTOTYPES -DRMADS -c hello.c

tlink /x /d /c /Tde /v c01.obj
     hello.obj,hello.exe,,tbcads3.lib FP87 mathl cl
```

各编译开关意义如下：

-I\ads;\borlandc\include

指明编译寻找包含文件的路径是\ads和\borlandc\include

-AT 通知编译器识别Borland C的特有保留字，如“cdecl”、“near”、“far”和“asm”等

-1 通知编译器产生80186和80286指令

-ml 指明用大模式编译

-vi 通知编译器在目标文件中产生符号调试信息

-f287 通知编译器产生80287浮点指令

-N- 禁止编译器产生堆栈检查指令

-G 通知编译器执行速度优化

-DTURBOC -DPROTOTYPES -DRMADS

定义符号“TURBOC”、“PROTOTYPES”和“RMADS”

-c 通知编译器只编译不连接

hello.c 指明源程序文件名

连接器的开关意义如下：

/x 通知连接器不产生映象文件

/d 通知连接器在符号重复时产生警告信息

/c 通知连接器以大小写敏感的方式处理符号

/Tdc 通知连接器产生一个DOS下可执行文件格式

/v 通知连接器产生包含符号调试信息的可执行程序

c01.obj 指明应该使用的起动代码模块

hello.obj 指明要连接的目标文件名

hello.exe 将可执行文件取名为hello.exe

thcads3.lib FP87 mathl cl

通知连接器将这些库与应用程序连接

1.4 ADS应用程序的加载与执行

AutoCAD R11只支持保护模式的ADS程序，保护模式应用程序的扩展名是“exp”；在AutoCAD R12中又增加了对实模式应用程序的支持，实模式应用程序的扩展名是“exe”。

1.4.1 在图形编辑器中加载ADS应用程序

在图形编辑器中加载一个已编译好的ADS应用程序和加载一个AutoLISP应用程序十分类似：加载AutoLISP应用程序用(Load)函数，而加载ADS应用程序用(xload)函数；同样，还需要一个文件名，即(xload"文件名")。(xload)函数将查找文件名指定的文件，把它加载到内存中，并立即执行其余的初始化操作。假如现在我们要在图形编辑器中加载ADS应用程序hello.exp，应该按以下的形式进行：

```
Command:(xload "hello")
```

由于在AutoCAD R11中只存在以“exp”为扩展名的ADS应用程序，所以(xload)函数会自动为程序加上正确的扩展名。如果在AutoCAD R12版中，AutoCAD会首先寻找hello.exp，如果找到就将其加载；如果未找到，AutoCAD才会寻找hello.exe，并在找到后将其加载。所以，在AutoCAD R12中，加载应用程序最好带上扩展名。如果找到了应用程序，并成功地装入了，那么(xload)函数将把在函数调用时使用的名字作为字符串返回，在上述例子中将返回字符串“hello”；如果不能成功加载，AutoCAD将显示出错信息。

AutoCAD是按照以下次序寻找ADS应用程序的：

(1)当前目录

(2)包含当前图形文件的目录

(3)由环境变量ACAD指定的目录

(4)包含AutoCAD程序文件的目录

当然，如果你提供了一个全路径(xload)函数，例如(xload"d:\\working\\hello")，AutoCAD就不会再去查寻其它目录了。

1.4.2 在AutoCAD初始化时加载ADS应用程序

在AutoCAD初始化时，它会检查当前是否存在一个名为acad.ads的文件，如果存在，它会自动加载acad.ads中列出的每个ADS应用程序。acad.ads是一个简单的文本文件，它

必须包含一个或多个ADS应用程序名，这些程序名可以带或不带扩展名，每个程序名占文本的一行。

1.4.3 调用ADS外部函数

每个ADS应用程序都可能定义了一组对AutoLISP来说是外部函数的函数。一旦一个ADS应用程序已用(xload)函数加载，你就可以调用该程序所定义的那些外部函数，方法和调用内部函数或用户定义的AutoLISP函数一样，即输入一个包含外部函数名的AutoLISP表达式。AutoLISP的变量可以作为外部函数的自变量传递。同别的函数一样，外部函数将返回一个结果。

外部函数可以提示用户从键盘上或用指定设备拾取点的方法输入数据。外部函数可被AutoLISP函数调用，但ADS应用程序却不能调用AutoLISP函数。使用象AutoLISP那样的加上“C:”的方法，外部函数也可以定义一个新的AutoCAD命令。此时，在“Command:”提示符下，直接键入函数名(无需带括号)，就可调用该函数。应该注意的是，在重名的情况下，后定义的外部函数将替换掉原来的函数。如果两个ADS应用程序定义了同名的外部函数，那么先加载进来的就会丢失。

1.4.4 检查当前已加载的ADS应用程序

在AutoCAD的“Command:”提示符下调用AutoLISP函数(ads)，将得到当前已加载的全部ADS应用程序的信息。

1.4.5 ADS应用程序的卸载

如果内存中加载了过多的ADS应用程序，将会影响系统的运行速度，此时应该从内存中卸载一些暂时不用的ADS应用程序。卸载ADS应用程序用(xunload)函数。例如，要卸载前面装入的hello.exp程序，应按以下形式进行：

```
Command:(xunload"hello")
```

在(且仅在)下面两种情况下，AutoLISP会自动地卸载一个ADS应用程序：

(1)该应用程序本身报告一个致命错误。

(2)在退出AutoCAD时。

第二章 ADS编程基础

2.1 编写第一个ADS程序

在经典著作《C语言程序设计》(Prenice Hall，1988年第二版)中，Brian Kernighan和Denis Ritchie开始使用一个现在已经很有名的“Hello，world!”程序来讨论C:

```
#include<stdio.h>
main()
{
  printf("Hello,world!\n");
}
```

在本章一开始，我们将给出用ADS编写的类似的程序。我们称这个程序为HelloADS，它在AutoCAD的“Command:”提示符下，打印出“Hello, ADS!”。

2.1.1 HelloADS.c源程序

```
/*-------HelloADS------*/

#include <stdio.h>
#include <string.h>
#include "adslib.h"

#define ELEMENTS(array) (sizeof(array)/sizeof((array)[0]))

static int loadfunc();
static int dofunc();
int hello();

struct {
    char *fname;
    int (*func)();
}functab[]={
   {"hello",hello},
 /*{......,......},
   {......,......},
    other external functions */
};

void main(int argc,char *argv[])
{
   int stat;
   short scode=RSRSLT;
```

```
    ads_init(argc,argv);
    while(1){
        if((stat=ads_link(scode))<0){
            printf("Bad status from ads_link()=%d\n",stat);
            exit(1);
        }
        scode=RSRSLT;
        switch(stat){
            case RQXLOAD:
             scode=loadfunc()==RTNORM?RSRSLT:RSERR;
             break;
            case RQSUBR:
             scode=dofunc()==RTNORM?RSRSLT:RSERR;
             break;
            default:
             break;
        }
    }
}

static int loadfunc()
{
    int i;
    char ccbuf[40];
    strcpy(ccbuf,"c:");
    for (i=0;i<ELEMENTS(functab);i++) {
        strcpy(ccbuf+2,functab[i].fname);
        if(ads_defun(ccbuf,i)!=RTNORM) return RTERROR;
    }
    return RTNORM;
}

static int dofunc()
{
    int val;

    if((val=ads_getfuncode())<0){
        ads_fail("Received invalid function code");
        return RTERROR;
    }
    (*functab[val].func)();
    return RTNORM;
}

int hello()
{
     ads_printf("Hello,ADS!\n");
     return RTNORM;
}
```

2.1.2 HelloADS.c源程序说明

读者也许会奇怪，为什么简单地打印一句“Hello,ADS!”需要这么多行代码。其实，其中的大多数语句都是在和AutoCAD通讯，这种通讯对每一个ADS程序都是必须的，我们稍后会详细介绍。

大致地看一看这个程序，我们就会发现，这个程序主要包含四个函数：main()，loadfunc()，dofunc()和hello()。函数main()是程序的入口点，等价于标准C语言程序的main函数。函数loadfunc()和dofunc()则分别用来装入和执行外部函数。在本例中，唯一的外部函数就是Hello()。函数Hello()是这个ADS程序中真正执行打印任务的函数，也就是被装入和执行的外部函数。

2.1.3 ADS函数调用

HelloADS至少调用了6个ADS函数，下面以它们在HelloADS中出现的顺序列出这些函数及对它们的简明描述：

ads_init( )	初始化和AutoLISP的通讯接口
ads_link( )	通知AutoLISPADS程序已经准备就绪并返回AutoLISP请求码
ads_defun( )	定义一个外部函数
ads_getfuncode( )	得到请求的外部函数的代码
ads_getargs( )	获得命令行参数
ads_printf( )	类似标准C库函数中的printf函数，但它并不是向标准输出设备输出，而是输出到AutoCAD的“Command:”命令行上

这些函数均可以在本书最后一章“库函数详解”中查到，读者也可以在adslib.h中看到它们的函数声明。在本书各章剖析范例程序的同时，我们将随时讨论所遇到的函数。

2.1.4 大写字母标识符

读者可能已注意到，HelloADS.c中有几个大写的标识符，这些标识符是在adscodes.h中定义的。

首先遇到的是ADS程序结果码和AutoLISP请求码。通常，ADS程序处于ads_link( )调用后等待AutoLISP返回请求码的状态。AutoLISP请求码共有六个，消息分发循环必须识别请求码并采取相应的动作。在此，我们使用switch语句，并对不能识别的请求做出缺省的处理。事实上，我们无需关心请求码和结果码的具体数值，而只需知道它们是整型常量及它们各自代表的意义。

AutoLISP请求码：

RQXLOAD	请求定义外部函数。通知ADS程序：它已被AutoLISP加载，并且应定义其外部函数。ADS程序随后必须多次调用ads_defun( )以定义其全部外部函数。

RQSUBR　请求执行外部函数。通知ADS程序：用户或一个AutoLISP函数已经调用了ADS程序的一个外部函数。ADS程序必须调用ads_getfuncode()以取得函数的代码，然后根据数字代码的值选择被请求的函数并调用它。函数代码是ADS程序在前面通过调用ads_defun()定义的。

RQXUNLOAD　请求卸载ADS程序。通知ADS程序：用户或AutoLISP函数已经调用(xunload)函数请求卸载ADS程序。在多数情况下，ADS程序将返回一个正常状态值；ADS程序也可以做一些清除工作，比如释放已分配的内存空间或关闭已打开的文件。

RQSAVE　请求保存当前图形。通知ADS程序：AutoCAD的Save命令已被调用，在保存图形之前AutoCAD将等待ADS程序对此请求的响应。

RQEND　请求保存并退出。通知ADS程序：AutoCAD的End命令已被调用，在保存图形和退出图形编辑器之前AutoCAD将等待ADS程序对此请求的响应。

RQQUIT　请求作废当前图形并退出。通知ADS程序：AutoCAD的Quit命令已被调用，并已确认作废当前图形，在退出图形编辑器之前AutoCAD将等待ADS程序对此请求的响应。

在大多数ADS程序中，对最后三个请求码不做处理。

ADS程序结果码：

RSRSLT　通知AutoLISP：ADS程序已完成任务，并等待进一步请求。

RSERR　通知AutoLISP：在ADS程序中产生了错误。如果这个结果是在RQXLOAD请求之后，AutoLISP将终止该ADS程序并将其卸载；如果这个结果是在RQSUBR请求之后，将显示以下消息：

```
error: ADS error in evaluation
```

2.1.5 HelloADS.c源程序分析

下面我们开始一行一行地剖析HelloADS.c这个程序。

程序的头三条语句为：

```
#include <stdio.h>
#include <string.h>
#include "adslib.h"
```

读者对头文件stdio.h和string.h可能已非常熟悉，它们是C语言标准库函数的头文件，由于在本程序中引用了标准C函数printf和strcpy，所以必须包含这两个头文件。

头文件adslib.h包含了对ADS的函数、数据结构和数值常量的声明，这是每个ADS程序所必须包含的头文件。

紧接着程序定义了一个宏：

```
#define ELEMENTS(array) (sizeof(array)/sizeof((array)[0]))
```

这个宏能方便地求出数组中定义的元素的个数。

接下来是对函数loadfunc()、dofunc()和hello()的预先声明:

```
static int loadfunc();
static int dofunc();
int hello();
```

之所以要预先声明，是因为在函数main()中和其它地方有些代码引用了这些函数。

在普通的C程序中，入口点是main()函数，这是程序开始执行的地方。ADS程序的入口点也是main()函数，它通常定义为如下格式:

```
void main(int argc,char *argv[])
```

main()函数有两个参数: argc和argv。读者可以按普通C程序来理解它们的含义。事实上程序中只有函数ads_init()用到了这两个参数。为了确保所有ADS库函数与AutoLISP通讯的初始化接口，函数ads_init()必须是ADS程序的main()函数中的第一个ADS库函数调用。从函数ads_init()返回的任何值都表示该调用成功。如果调用失败，该函数就不会把控制返回给ADS程序。

初始化完成后，就进入了消息分发循环。通常，ADS程序处于ads_link()调用后等待AutoLISP返回请求码的状态。读者可能已注意到，在说明变量scode时已将其初始化为结果码RSRSLT，这表明在调用ads_link(scode)时ADS程序正在等待AutoLISP请求。

```
if((stat=ads_link(scode))<0){
     printf("Bad status from ads_link()=%d\n",stat);
     exit(1);
}
```

以上的语句进行了必要的出错处理，即: 如果ads_link()调用失败则退出。注意此处使用了printf这一标准C函数，因为如果ads_link()调用失败，就不能调用函数ads_printf()。如果ads_link()调用成功，则重新把scode置为结果码RSRSLT，表明调用成功，并根据具体请求进行相应的处理。

```
scode=RSRSLT;
switch(stat){
    case RQXLOAD:
     scode=loadfunc()==RTNORM?RSRSLT:RSERR;
     break;
    case RQSUBR:
     scode=dofunc()==RTNORM?RSRSLT:RSERR;
     break;
```

```
        default:
         break;
    }
```

如果stat=RQXLOAD，则要求在AutoLISP请求加载ADS程序时，ADS程序必须多次调用ads_defun()来定义它所有的外部函数。在本程序中这是通过函数loadfunc()来实现的。

```
static int loadfunc()
{
   int i;
   char ccbuf[40];
   strcpy(ccbuf,"c:");
   for (i=0;i<ELEMENTS(functab);i++) {
      strcpy(ccbuf+2,functab[i].fname);
      if(ads_defun(ccbuf,i)!=RTNORM) return RTERROR;
   }
   return RTNORM;
}
```

首先介绍两个常量RTNORM和RTERROR。它们是ADS库函数定义的结果码，RTNORM表明函数调用成功，RTERROR表明函数调用失败。

下面说明一下程序头部定义的结构数组functab:

```
struct {
    char *fname;
    int (*func)();
}functab[]={
   {"hello",hello},
};
```

结构中定义了两个域：函数名指针和函数指针。由于ADS程序定义的外部函数通常都不止一个，所以定义结构数组是个很方便的办法。虽然在本程序中只定义了一个外部函数hello()，在函数loadfunc()的循环中也只能循环一次，但我们仍然定义了结构数组，这么做的目的是为读者提供一个标准的样板程序作参考，读完本章的其余部分，读者自然会明白这一点。

下面我们分析一下函数loadfunc()。首先我们调用串拷贝函数在串ccbuf中的前两项拷入“C:”；然后在for循环中，在“C:”后又拷入了函数名。在本例中，串ccbuf的内容变为“c:hello”。

然后调用函数ads_defun()来定义外部函数，即:

```
ads_defun("c:hello",0)
```

这样我们就定义了一个外部函数hello()，“0”是它的函数码。这个函数码是以后

AutoLISP识别该函数时所要求的。

之所以在函数名前加“C:”，是因为这样定义的函数可以象AutoCAD的命令一样被使用。比如，当我们要调用函数hello()时，可直接键入:

```
Command:hello
```

否则，我们就必须象调用LISP函数一样调用该函数:

```
Command:(hello)
```

函数ads_defun()要放在for循环中，因为这样可以一次完成对多个外部函数的定义。程序对函数ads_defun()的返回结果做了判断，如果调用成功则返回RTNORM，如果失败则返回RTERROR，这样便于在消息分发循环中处理，即

```
scode=loadfunc()==RTNORM?RSRSLT:RSERR;
```

如果函数loadfunc()调用不成功，则scode=RSERR，则在ads_link(scode)调用中会终止ADS程序，并显示出错信息。这一点对函数dofunc()是相同的。

现在我们再回到消息分发循环，如果stat=RQSUBR，则请求执行函数时，将调用函数dofunc()来执行外部函数。

```
static int dofunc()
{
   int val;

   if((val=ads_getfuncode())<0){
      ads_fail("Received invalid function code");
      return RTERROR;
   }
   (*functab[val].func)();
   return RTNORM;
}
```

函数dofunc()首先调用ads_getfuncode()以获得用ads_defun()所定义的函数码。如果调用成功，则根据得到的函数码执行相应的外部函数，在本例中即执行函数hello()。函数hello()的执行结果就是在“Command:”提示符后显示“Hello,ADS!”。

读者也许会问：函数ads_retvoid()是做什么用的？其实，函数ads_retvoid()并不做具体的工作，而仅仅是禁止函数的返回值。熟悉LISP语言的读者也许知道，在AutoLISP程序中常常是调用(princ)来禁止函数的返回值的，在这里，ads_retvoid()起到了同样的作用。如果不用ads_retvoid()，我们就会经常看到令人讨厌的nil返回值。读者可以删去这个函数，重新编译连接再运行，就会明白它的用处了。

2.1.6 运行HelloADS

理解这个程序之后，读者就可以试着将这个程序编译连接成HelloADS.exe或HelloADS.exp了。如果遇到出错信息，请仔细核对，看看是否程序输入有误。如果编译和连接都成功了，就可以运行这个程序了。你可以按以下步骤进行:

1.进入AutoCAD环境;

2.在“Comnand:”提示符下键入(xload“HelloADS”);

3.在“Comnand:”提示符下键入命令hello;

此时你将看到“Hello, ADS!”出现在下一个命令行上。

2.2 ADS程序的工作流程

经过对HelloADS.c源程序的分析，读者可能已经发现ADS程序与普通的C程序有很多不同之处。每一个ADS程序都必须能支持并且使用ADS环境所定义的与AutoLISP的接口程序。这个接口程序要求ADS程序按一定的次序、使用确定的值来调用确定的ADS库函数。在HelloADS.c中，我们用的正是这种接口程序。下面我们简要地概括一下这种接口程序中的调用流程:

1.ADS程序通过调用ads_init(), 初始化和AutoLISP的通讯接口。

2.ADS程序通过调用ads_link(), 向AutoLISP返回一个ADS程序的结果码RSRSLT, 并指示接受和处理来自AutoLISP的请求。

3.AutoLISP通过ads_link()向ADS程序返回一个请求码RQXLOAD。

4.ADS程序通过对每个子函数调用一次ads_defun(), 将其定义为外部函数。

5.如果ADS程序检查出一个错误，它将向AutoLISP返回一个结果码RSERR; 否则，ADS程序将再一次调用ads_link(), 并向AutoLISP返回一个结果码RSRSLT。

6.当用户或AutLISP函数要调用ADS程序的某个外部函数时，AutoLISP通过ads_link()向ADS程序返回一个请求码RQSUBR。

7.如果外部函数调用成功，ADS程序将调用ads_link()并向AutoLISP返回一个结果码RSRSLT; 如果调用失败，ADS程序将向AutoLISP返回一个结果码RSERR。

从以上分析可以看出，一旦ADS程序被加载，则ADS程序的所有外部函数都处于非激活状态，直到AutoLISP向它发出一个请求消息; 而在ADS程序对AutoLISP的请求消息作出响应的过程中，AutoCAD和AutoLISP又都处于非激活状态，等待着从ADS函数返回的结果。这就是ADS程序由消息驱动的机制。

读者应该注意到，在HelloADS.c中，这一消息分发机制是用一个无限循环来实现的。事实上，如果希望ADS程序能很好地在AutoCAD环境下执行，就必须使用这种机制。

HelloADS.c为读者提供了一个很好的样板程序。因为在通常情况下，接口都是相同的，所以读者在编制自己的ADS程序时，可以把HelloADS.c中的大部分代码照搬过来。此时，读者真正需要做的就是加入自己的特殊的外部函数，就象HelloADS中的hello()函数一样。