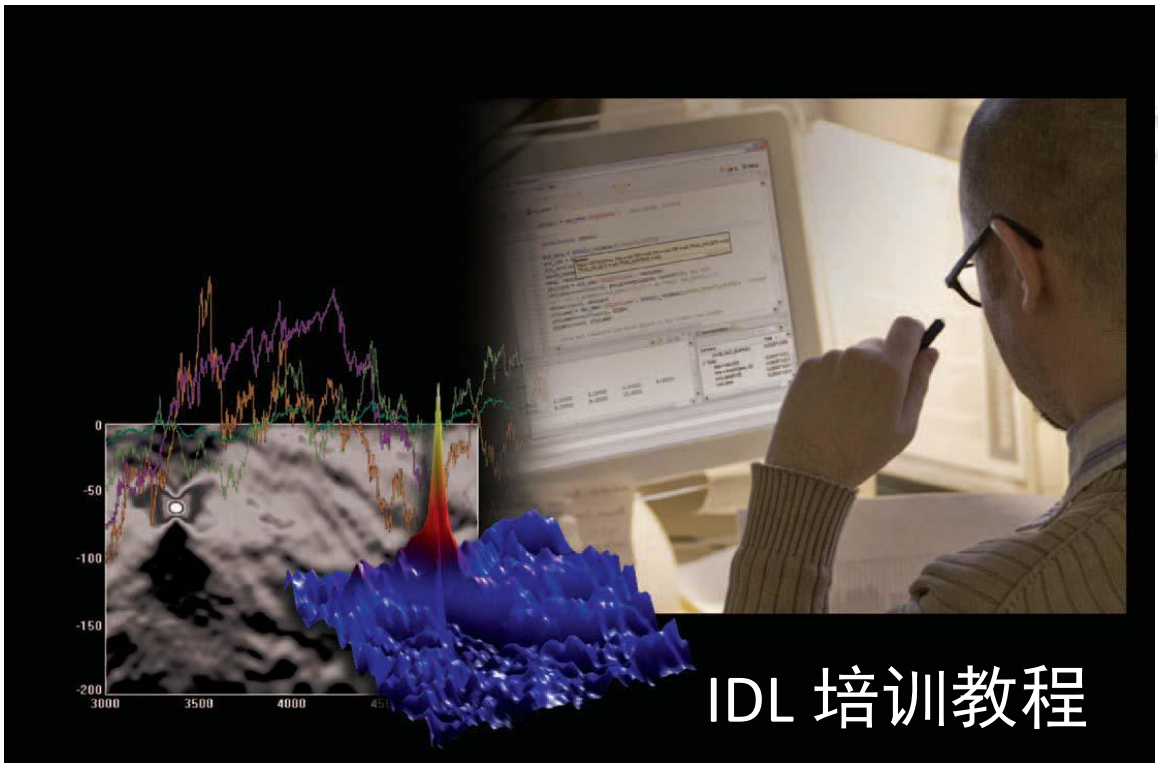


bbs.esrichina-bj.cn
bbs.esrichina-bj.cn

ESRI 中国（北京）有限公司

2010 年 7 月

ESRI China (Beijing)
Geographic Thinking
分 享 地 理 价 值

目 录

ESRI 中国（北京）有限公司 1
1 前言 1
1.1 为什么要编程 1
1.2 为什么要学 IDL 1
1.2.1 IDL 的优势 1
1.3 补充 2
2 IDL 的使用 3
2.1 IDL 的安装 3
2.1.1 运行安装 3
2.1.2 配置许可 5
2.1.3 目录说明 7
2.2 工作台入门 7
2.2.1 启动工作台 8
2.2.2 工作空间 11
2.2.3 工程（project） 12
2.2.4 文件（file） 16
2.2.5 查看器（View） 21
2.2.6 命令行及命令历史查看器 21
2.2.7 透视图 21
2.2.8 获取帮助 24
2.3 编写 IDL 程序 26
2.3.1 命令行模式 26
2.3.2 批处理模式 27
2.3.3 编译模式 28
2.3.4 IDL 程序 28
2.4 网络资源 29
3 IDL 语法基础 30
3.1 变量 30
3.1.1 局部变量 30
3.1.2 系统变量 34
3.2 数组 37
3.2.1 概述 37
3.2.2 创建数组 37
3.2.3 数组的存储 39
3.2.4 使用数组 41
3.2.5 运算 42
3.3 字符及字符串 55
3.3.1 创建字符串和字符串数组 55
3.3.2 特殊字符 56
3.3.3 部分字符操作函数 56
3.4 结构体 58

3.4.1 概述.....58
3.4.2 创建结构体.....58
3.4.3 访问结构体.....59
3.4.4 结构体操作相关函数.....59
3.5 指针.....60
3.5.1 概述.....60
3.5.2 创建和访问指针.....60
3.5.3 空指针.....60
3.5.4 指针的内存控制.....61
3.5.5 指针操作函数.....62
3.6 运算符.....62
3.6.1 数学运算符.....62
3.6.2 逻辑运算符.....63
3.6.3 位运算符.....64
3.6.4 关系运算符.....65
3.6.5 数组运算符.....66
3.6.6 其他运算符.....67
3.6.7 运算符的优先级.....68
3.7 其他符号.....69
3.7.1 $续行符.....69
3.7.2 &同行符.....69
3.7.3 ;注释符.....69
4 IDL 编程基础.....70
4.1 程序控制.....70
4.1.1 循环语句.....70
4.1.2 条件语句.....71
4.1.3 跳转语句.....73
4.2 位置参数及关键字参数.....74
4.2.1 位置参数.....74
4.2.2 关键字参数.....74
4.2.3 参数继承.....75
4.2.4 参数传递机理.....75
4.2.5 相关函数.....76
4.3 错误处理.....77
4.3.1 错误处理程序.....77
4.3.2 数学错误.....78
4.4 调用执行方式.....79
4.4.1 EXECUTE()函数.....79
4.4.2 CALL_FUNCTION 函数.....79
4.4.3 CALL_PROCEDURE.....79
4.5 编译规则.....79
4.6 IDL 高效编程.....80
4.6.1 释放内存.....80

4.6.2 优化运行时间.....80
5 输入与输出.....82
5.1 标准输入输出.....82
5.1.1 输入输出函数.....82
5.1.2 格式化输出.....82
5.2 文件读写.....83
5.2.1 打开文件.....83
5.2.2 读写 ASCII 码文件.....84
5.2.3 读写二进制文件.....87
5.3 读写通用格式文件.....91
5.3.1 图像文件信息查询.....91
5.3.2 JPEG 文件操作.....92
5.3.3 HDF 数据读入.....93
6 IDL 界面创建与事件处理.....95
6.1 iTools 界面.....95
6.2 界面创建.....95
6.2.1 单元组件界面.....95
6.2.2 复合界面.....103
6.2.3 对话框界面.....104
6.2.4 功能界面.....105
6.3 界面的控制.....106
6.3.1 Widget_Control.....106
6.3.2 Widget_Info.....107
6.4 事件的处理.....107
6.4.1 事件的结构.....107
6.4.2 事件的“管家”.....107
6.4.3 事件的处理.....108
7 直接图形法.....109
7.1 概述.....109
7.2 图形显示设备.....109
7.3 显示模式.....110
7.3.1 8 位显示模式.....110
7.3.2 24 位显示模式.....112
7.4 系统的字体.....113
7.4.1 矢量字体.....113
7.4.2 设备字体.....115
7.4.3 TrueType 字体.....116
7.5 坐标系统.....117
7.6 图形图像显示.....117
7.6.1 显示窗口.....117
7.6.2 绘制曲线.....118
7.6.3 显示图像.....128
7.6.4 绘制曲面.....130

7.6.5 相关函数参考.....131
7.6.6 体数据显示.....132
7.7 地图投影.....132
7.7.1 功能描述.....132
7.7.2 举例.....133
8 IDL 图像处理.....134
8.1 图像文件的输入输出.....134
8.2 图像裁剪.....134
8.3 图像的尺寸修改.....135
8.4 图像的位置变换.....135
8.5 图形的域变换.....136
8.6 图像的增强和滤波.....136
8.6.1 图像增强.....136
8.6.2 滤波.....137
8.7 图像平滑.....137
8.7.1 均值平滑.....137
8.7.2 中值平滑.....138
8.8 图像边界检测.....138
8.9 图像形态学处理.....138
8.9.1 腐蚀和膨胀.....139
8.9.2 开运算.....139
8.9.3 闭运算.....139
8.9.4 峰值检测.....139
8.9.5 分水岭边界检测.....139
8.9.6 图像识别.....140
8.9.7 图像的梯度边界检测.....140
8.9.8 图像细化.....140
8.9.9 图像分割.....140
9 IDL 对象图形法.....141
9.1 对象层次结构.....141
9.2 对象基本操作.....141
9.2.1 创建对象.....141
9.2.2 方法调用.....142
9.2.3 销毁对象.....142
9.3 IDL 图形基本对象.....142
9.3.1 IDLgrWindow 对象.....142
9.3.2 IDLgrView 对象.....143
9.3.3 IDLgrModel 对象.....144
9.3.4 原子对象.....145
9.4 二维图形显示.....146
9.4.1 线对象.....146
9.4.2 多边形.....147
9.4.3 文字.....148

9.4.4 坐标轴.....154
9.4.5 曲线.....154
9.5 三维图形显示.....154
9.6 坐标统一.....155
9.7 对象的交互操作.....155
9.8 编写类.....155
9.8.1 编写新对象类.....155
9.8.2 编写继承类.....157
9.9 地图投影.....157
10 智能化工具 iTools.....159
10.1 iTools 概述.....159
10.2 iTools 的使用.....160
10.2.1 数据导入和导出.....160
10.2.2 数据和对象属性的查看.....165
10.2.3 显示操作.....166
10.2.4 添加标注.....168
10.2.5 公共操作.....169
10.2.6 iPlot 使用.....169
10.2.7 iContour 使用.....171
10.2.8 iImage 使用.....173
10.2.9 iMap 使用.....179
10.2.10 iSurface 使用.....183
10.2.11 iVolume 使用.....189
10.2.12 iVector 使用.....190
10.3 开发.....191
11 IDL 程序的部署与发布.....193
11.1 IDL 的部署.....193
11.1.1 Runtime 许可发布.....193
11.1.2 虚拟机免费发布.....193
11.2 IDL 的发布.....193
11.2.1 Sav 文件.....193
11.2.2 可执行文件（Windows 下的 exe）.....195
11.2.3 COM 和 JAVA.....197
12 混合调用.....198
12.1 IDL 功能扩展.....198
12.1.1 直接调用 SPAWN.....198
12.1.2 IDL 调用 ActiveX.....198
12.1.3 IDL 调用 DLL.....204
12.1.4 IDL 使用 DLM.....206
12.2 其他语言与 IDL.....211
12.2.1 对象输出助手.....211
12.2.2 IDLDrawWidget Activex 方式.....220
12.2.3 Callable IDL.....223

13 ENVI 功能扩展 225
13.1 ENVI 波段运算和波谱运算 225
13.1.1 波段运算 225
13.1.2 波谱运算 229
13.1.3 几个例子 229
13.2 用户函数 230
13.2.1 用户函数介绍 230
13.2.2 ENVI 菜单介绍 230
13.2.3 ENVI 组件介绍 232
13.2.4 编写实例 237
13.3 ENVI 的二次开发 239
13.3.1 二次开发模式 240
13.4 常用关键字及函数 240
13.4.1 FID 240
13.4.2 R_FID 和 M_FID 241
13.4.3 DIMS 241
13.4.4 POS 241
13.4.5 文件管理函数 241
13.4.6 打开文件函数 242
13.4.7 获取数据函数 243
13.4.8 感兴趣区函数 243
13.4.9 数据生成函数 243
13.5 实例编写 245
13.5.1 调用 ENVI 实现自动校正 245
13.5.2 批处理格式转换函数实现 245

bbs.esrichina-bj.cn

1 前言

1.1 为什么要编程

这个也是比较古老的话题，一句话概括：让计算机实现自己的想法。因为人总有自己的想法，有些常规的想法前人已经想过，那么现成的软件或程序可用；但如有自己独特的想法，如何在计算机中实现呢，这样就必须得学会编程。

1.2 为什么要学 IDL

计算机技术发展到今天，可以说计算机的语言非常多。

但对一些科学家、技术工程师的非专业编程人士来说，并不一定所要实现的所有功能都要从底层写起。他们的编程需要一般就两点：

- 编写各种满足工作需要的小程序，目的是让工作更加自动化、方便化和快捷化，由于工作的原因一般很少编写大的应用程序。
- 从学习到使用越快越好，不想学了好几个月才刚刚弄明白 C++中的类、对象的使用，需要尽快的从学习语言转到在实践中应用。

对专业编程人士来说，通过 IDL 与其他语言的混合编程，借用 IDL 高效、快速的可视化与分析功能，他们可以快速进行应用程序及编写跨平台软件。

1.2.1 IDL 的优势

IDL（Interactive Data Language）交互式数据语言是进行数据分析、可视化表达和应用开发的第四代可视化，它最初的原型为美国航空航天局（NASA）的一个可视化项目，由于其灵活和强大的数据分析和可视化性能，受到 NASA 的科学家和工程师的喜爱，进而发展成为一门独立的可视化语言。

它的优势主要有以下几个方面：

■ **数据访问** —— 灵活的数据输入、输出方式

在科学研究中，将会使用大量不同格式的数据。IDL 基本上支持各种数据格式，并提供了大量的数据读写工具，免除数据读写的烦恼，可以将大量的精力放在数据分析上。

■ **数据分析** —— 强大的图像处理、信号处理和数据统计分析能力

IDL 集成了完善的数据分析、数据统计、图像处理和信号处理软件包。项目应用 IDL 可以节省大量时间。使用传统的 C 或者 FORTRAN 语言需要数天或数周的项目，使用 IDL 可以在数小时内完成。

■ **数据可视化 —— 先进的图形技术和 OpenGL 图形加速技术的二维和三维数据交互可视化能力**

从简单的二维绘图、多维绘图、体积可视化、图像显示、动画，到利用 OpenGL 硬件加速功能进行交互式的三维图形浏览，以及支持多处理器快速进行体数据渲染，使用 IDL 您可以轻松获得丰富的可视化效果。

■ **软件开发 ——** 强大的多平台支持、跨平台的本地化的用户界面、全新的开发环境、提供了有效的数据管理手段

♦ 可以运行在 Windows、Macintosh 和 UNIX 等多种操作上，从而可以很方便地进行跨平台开发应用程序。

♦ IDL 提供一套完整的、跨平台的本地化用户界面，可以很方便地进行跨平台开发，构建符合本地平台风格的用户界面。

♦ IDL 工作台提供了 Eclipse 的强大支持、全中文界面显示、强大的代码管理开发和调试工具。快速构建原型应用程序：IDL 的是一种解释语言，可以在个别问题上看到立竿见影的结果，同时 IDL 的降低了设计、编译、测试的周期。

■ **集成 —— 灵活的外部语言接口**

IDL 是一个灵活的、可扩展的用于可视化、分析和开发的工具，能够轻松的与其他的软件工具集成。IDL 能够将其他编程语言的代码和函数导入，也可以将 IDL 实现的功能集成到其他语言编写的应用程序中去。

■ **程序共享 —— 灵活多样的程序发布方式和轻松的成果共享**

♦ 你能够很容易快捷的共享 IDL 应用程序。IDL 虚拟机（Virtual Machine），为 IDL 用户和软件开发人员提供了 IDL 应用程序发布的便捷工具。IDL VM 适用于所有 IDL 支持的平台，可以将 IDL 程序进行跨平台的发布。

1.3 补充

教程中各章节的例子程序在源码工作空间[Code]的 Chapter[章节号]目录下。

2 IDL 的使用

2.1 IDL 的安装

下面是 WindowsXP-32 位下的 IDL 安装及许可配置过程。

2.1.1 运行安装

自动运行光盘，选择 Install 32-bit IDL（如果是网站下载，双击运行 exe 文件）

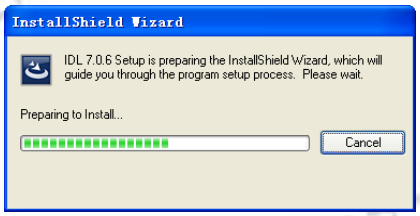


图2-1 自动解压缩

确认安装

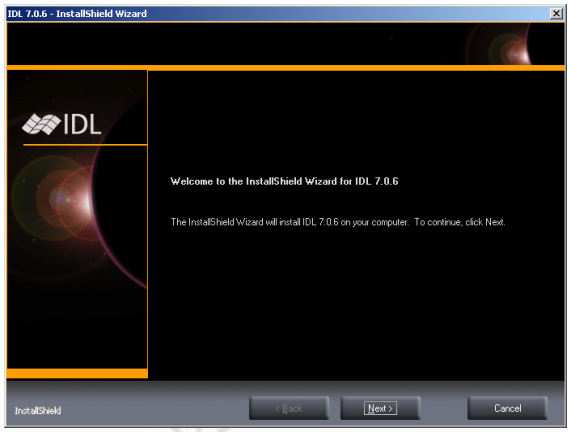


图2-2确认安装

接受软件安装协议

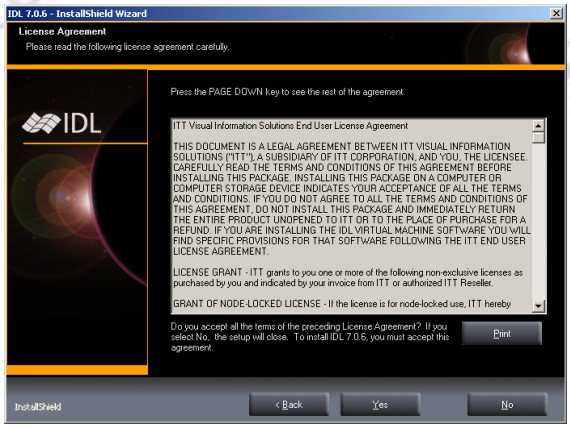


图2-3 接受安装协议

选择安装路径

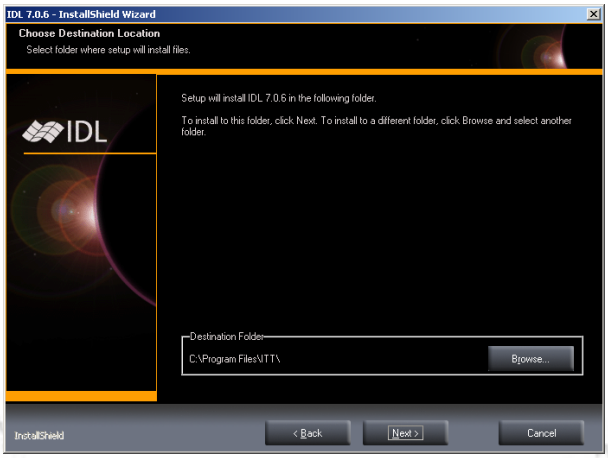


图2-4 选择安装路径

扩展功能选择，默认没有选上的主要是 IDL 的一些扩展，包括医学影像格式网络访问 DICOM Network Services、数据库连接包 IDL DataMinner 和高分辨率数据 Hign Resolution Maps。选择高分辨率数据 Hign Resolution Maps，其他两者根据具体配置情况选择。

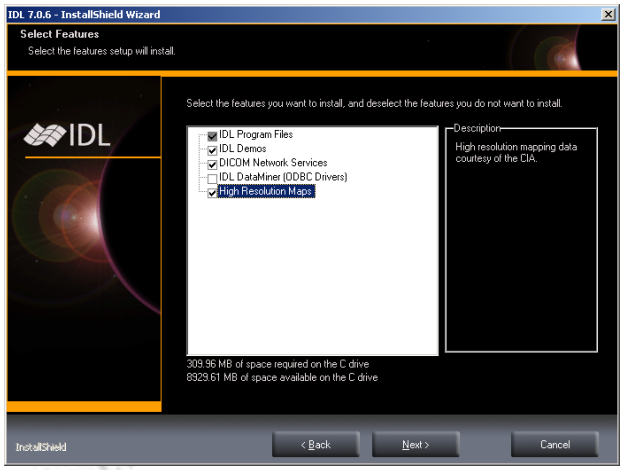


图2-5 模块选择

注册文件类型，默认全选即可。



图2-6 注册文件类型

是否启动 DICOM，如果安装时选择了 DICOM 模块，则会有下面的对话框提示。

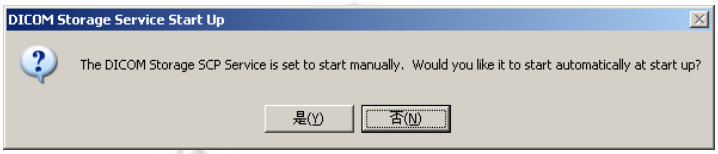


图2-7 是否开机启动DICOM

确认安装信息

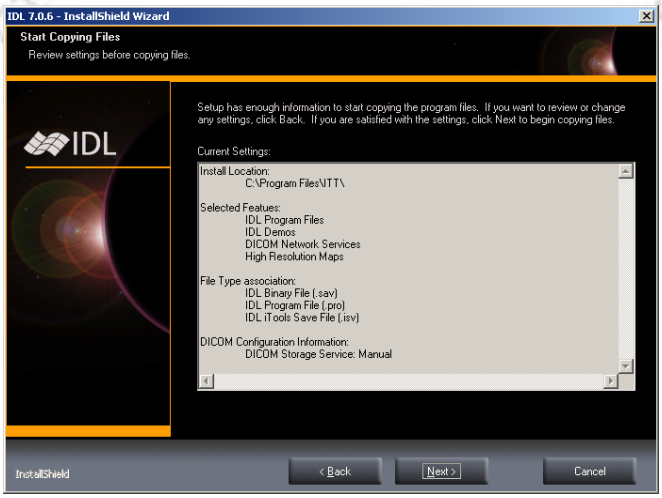


图2-8 确认安装信息

2.1.2 配置许可

在软件安装完成时会跳出如下对话框：

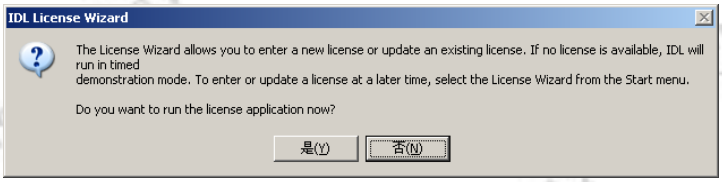


图2-9 许可配置选择

如果选择是，则会跳出许可配置窗口。如果选择否，则可通过开始→idl7.0.6→License Wizard 来进行配置。

添加许可信息，在弹出的界面中选择第三项“Install a license you have received”。

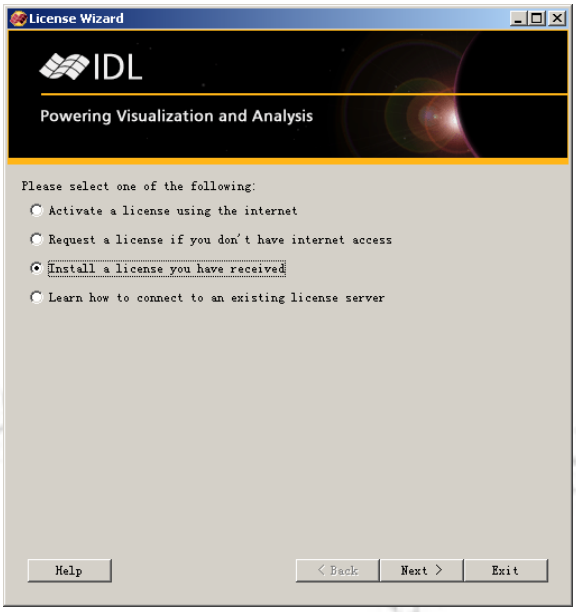


图2-10 许可文件类型选择

选择 Evaluation license，选择产品类型，试用时间和 KEY 码（在其他文件中获得）。

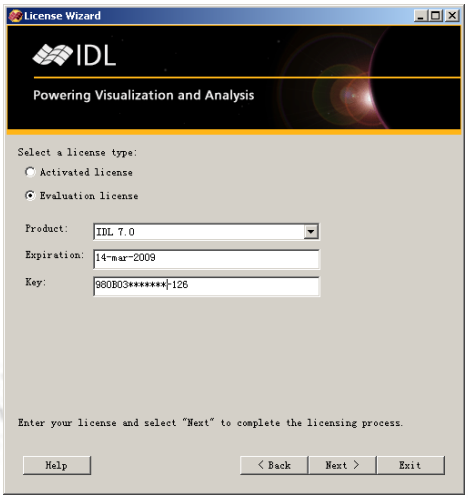


图2-11 填写许可信息

点击 Next，跳出一个保存许可文件的提示，点击 Exit 完成许可的配置。

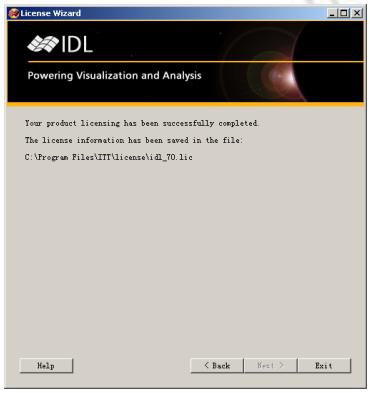


图2-12 保存许可文件

保存许可文件后，系统弹出安装完成界面，查看发布说明选择“View release notes”。

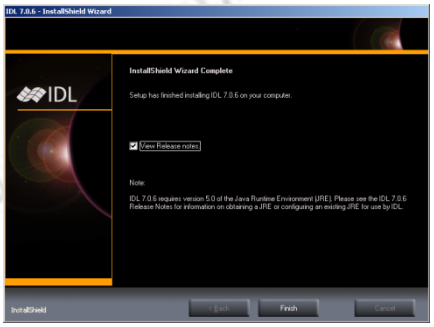


图2-13 完成许可配置

2.1.3 目录说明

IDL7.0.8 如果默认安装，那么安装目录为 C:\Program Files\ITT\IDL708。

Bin：IDL 的虚拟机及部分功能配置文件；

Example：包括 Demo 演示程序集官方提供的例子，很多，功能很强；

External：包括 IDL 功能扩展方面的几个例子；

Help：IDL 帮助；

IDLDE：IDL 工作台启动文件及配置文件；

Lib：强大的函数文件库，大部分是源码，可参考学习；

Products：如存在则是对应的 ENVI 的安装后目录；

Resource：运行及启动的资源库。

2.2 工作台入门

相比以前的 IDL 版本，IDL7.0 在所有的支持的平台下采用了全新的 Eclipse 框架作为工作台。更新的主要原因是之前的版本在 UNIX、LINUX 和 Macintosh 下采用的是 X/Motif 程序，很多的功能受到限制使得用户很难熟练使用。

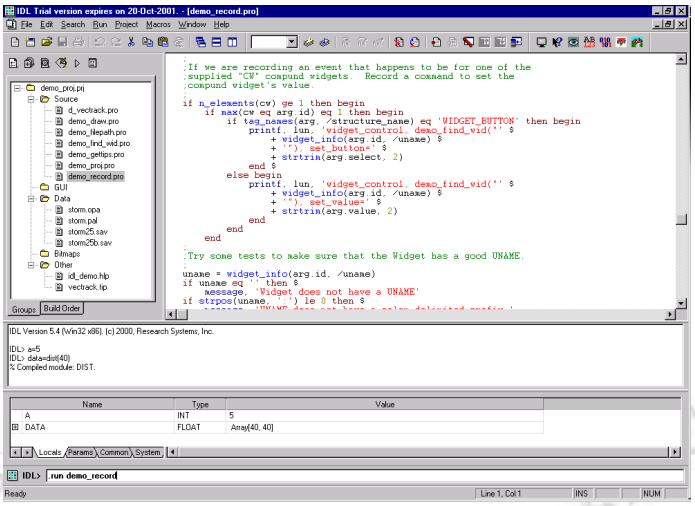


图 2-14 IDL6.4 的版本界面

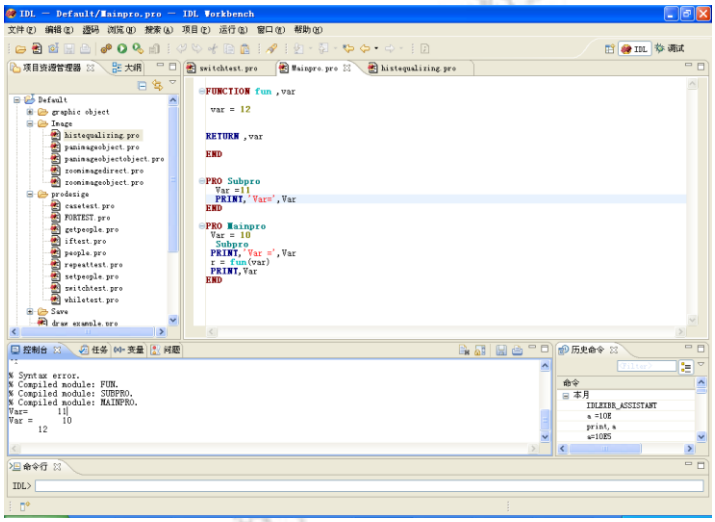


图 2-15 IDL7.0 的工作台

2.2.1 启动工作台

点击**开始→程序→ENVI 4.X(IDL7.X)→idl**，首次运行 IDL 的时候，会启动欢迎界面，见图 2-16，上面有工作台、新增内容、Web 资源和教程四个大按钮，点击则分别进入对应的内容。

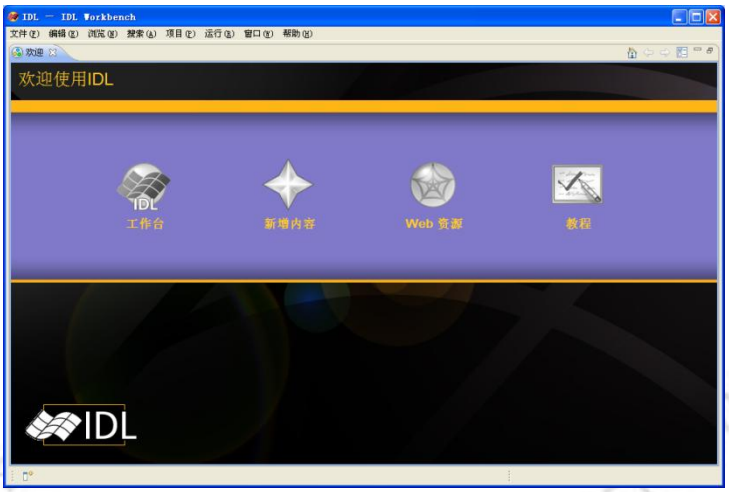


图2-16 IDL7.0的欢迎界面

点击新增内容，显示界面图 2-17，上有 IDL 新特性、最新的的技术技巧两个网址的链接，同时有当前工作台的插件信息和 ITT 上的新闻链接，方便用户及时进行更新。



图2-17 IDL7.0新增内容界面

点击 Web 资源，显示界面如图 2-18，分别有 IDL 网络、用户案例、代码共享库、用户论坛和开发者网络五个链接，用户需要了解、使用 IDL 或遇到问题时可以通过链接寻找网络上相关的资源及帮助信息。



图2-18　IDL7.0Web资源界面

点击教程显示界面如图 2-19，分别对应了 IDL 动态帮助和 IDL 指南中的 IDL 工作台主题、IDL 语言相关的主题。

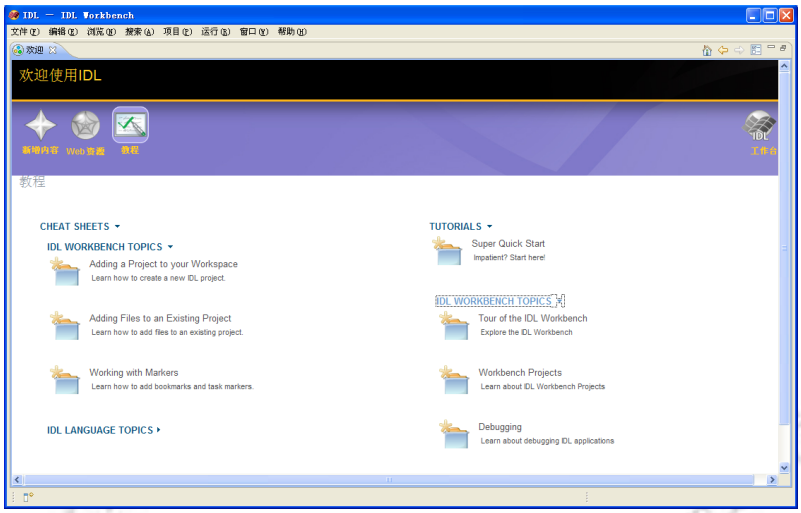


图2-19 IDL7.0教程界面

点击工作台，软件会提示用户选择的工作空间位置，此次我们需要将工作空间设置在一个默认目录，如“d:\code”。



图2-20 创建工作空间

点击 OK 后进入工作台环境，见图 2-21。

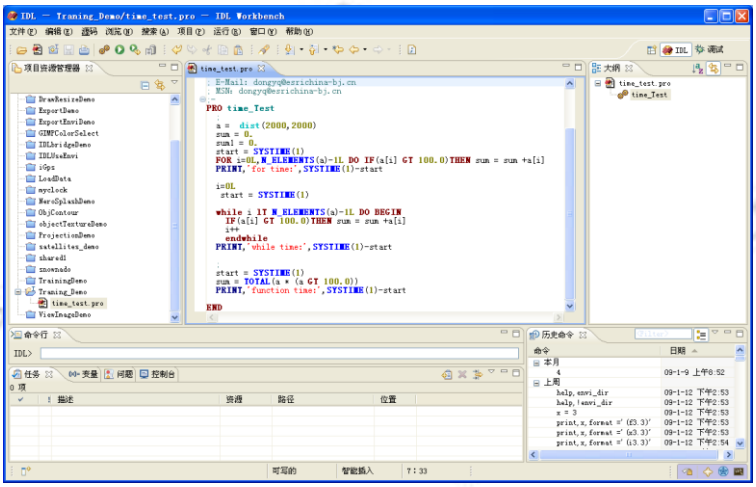


图2-21 IDL工作台

需要先设置下当前工作台的参数，在工作台的菜单[窗口]→[首选项]下，设置常规→工作空间→文本文件编码，选择 GB2312 或 GBK，以避免汉字出现乱码。

2.2.2 工作空间

IDL 下的工作空间是包含一个或多个 project 的目录，project 可以包含 IDL 的源代码 pro 文件和相关的资源文件。初次使用 IDL 的工作台时会提示你创建一个新的工作空间在默认目录（Windows 平台是“Documents and Settings”，Macintosh、Solaris 和 Linux 系统平台下是$HOME 系统变量里面的路径）。

可以创建多个的工作空间，但是当前 IDL 编译环境下只能有一个是活动的。每次 IDL 的工作台启动时它都会显示让你选择哪个工作空间，如禁止这个工作空间选择，在选择界面上勾选“设置为默认工作空间”。如需要切换工作空间，可以在系统菜单下选择“文件”-“切换工作空间”来进行工作空间的切换。

本次培训选择独立建立一个工作空间 code，建立一个工程项目然后把相关的代码导入到当前项目中即可。

先创建工程文件夹 training，然后右键选择[导入]→[文件系统]，在界面中选择源码所在的目录，设置相关的导入参数后就可以进下一步导入了，这种方式是将文件复制到当前的目录中，也可选择[高级]进行链接而不复制文件。

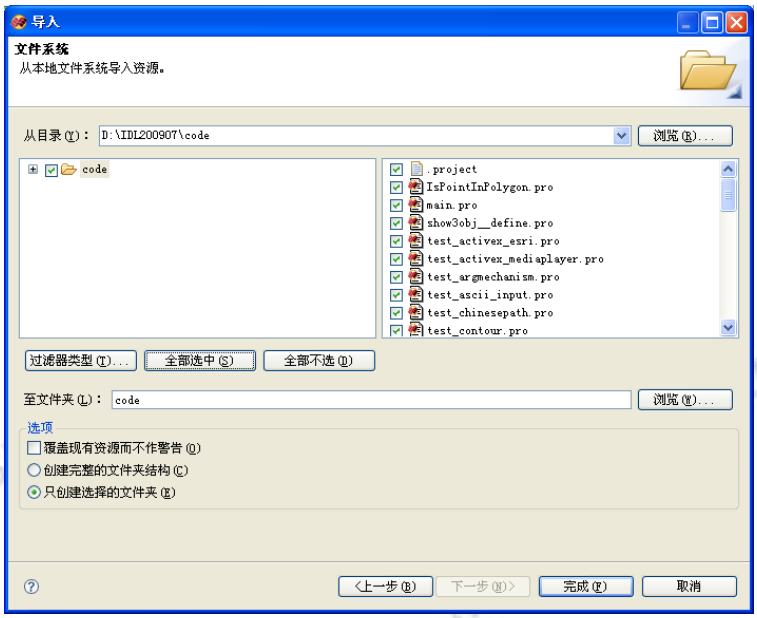


图 2-22 导入文件系统

2.2.3 工程（project）

IDL 的工程项目是使用 project 来组织和管理 IDL 源代码及工程资源文件的。每个工程是当前工作空间的路径或工作空间的子路径，源代码放在工程路径下面对工程使用中的文件搜索、书签创建等都会带来很大的方便。

工程是 pro 和 function 的集合，通过相互的调用来实现复杂的功能。

2.2.3.1 建立新工程

建立一个新的工程是要在菜单中选择“文件”-“新建”-“IDL 工程”，在弹出界面（见图 2-23）中输入工程的名称，选择工程的位置在当前工作台目录还是其他目录，为了调试等操作方便一般建议新建工程在默认的工作台下。

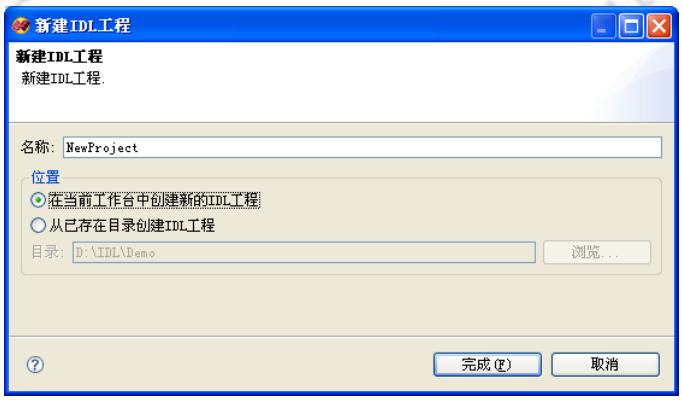


图 2-23 新建 IDL 工程

点击完成，创建的工程在工作空间的项目资源管理器中列出，见图 2-24。

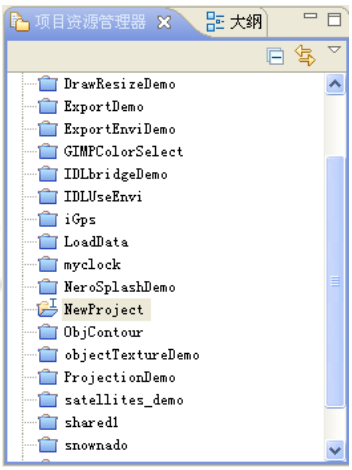


图 2-24IDL 的项目资源管理器

2.2.3.2 工程属性修改

对 NewProject 工程的属性可以在右键菜单中选择“属性”（或 Alt+Enter）查看，项目的属性信息对话框见图 2-25。基本信息框部分包括了项目属性的路径信息，文本文件编码和文本行界定符。

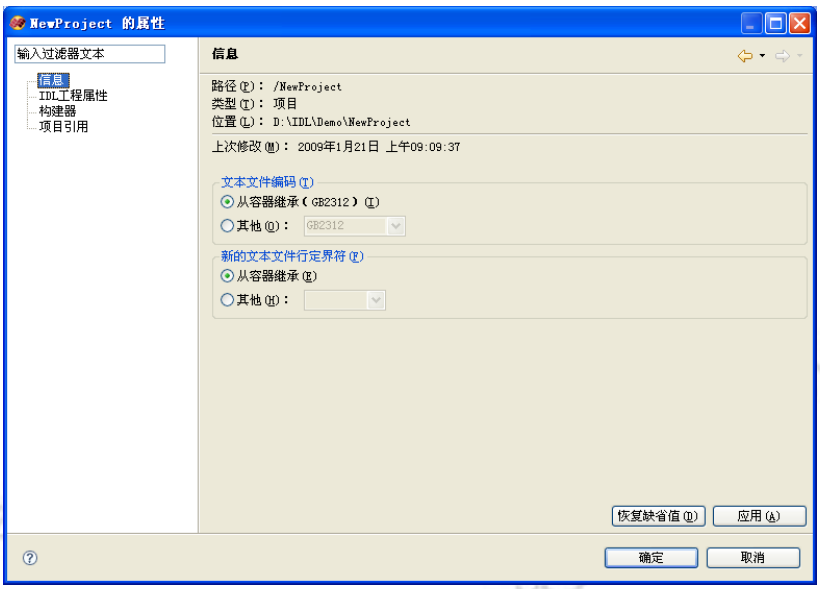


图 2-25IDL 的项目基本信息

在工程属性中可以对项目的编译条件进行设置，比如运行和构建项目前是否执行.Reset_Session 命令，修改工程的运行命令（默认是工程名）和修改构建的 sav 类型等。

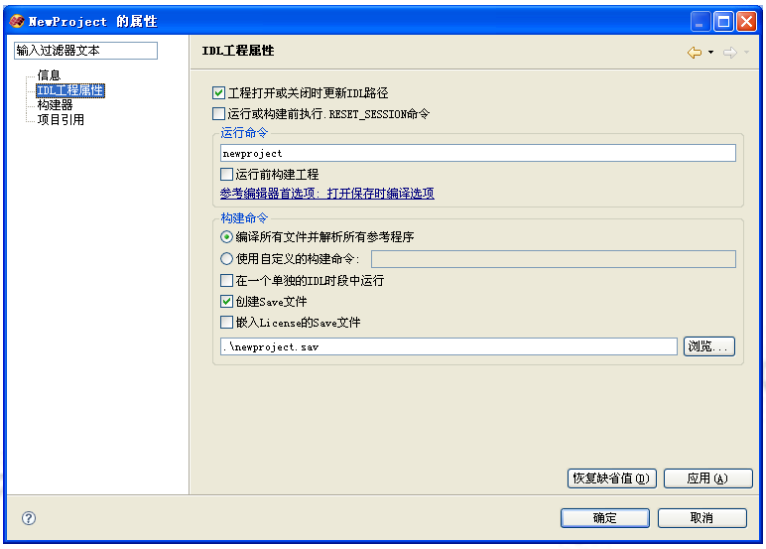


图 2-26 IDL 工程属性

构建器选项图可以让用户选择使用的程序编译器。

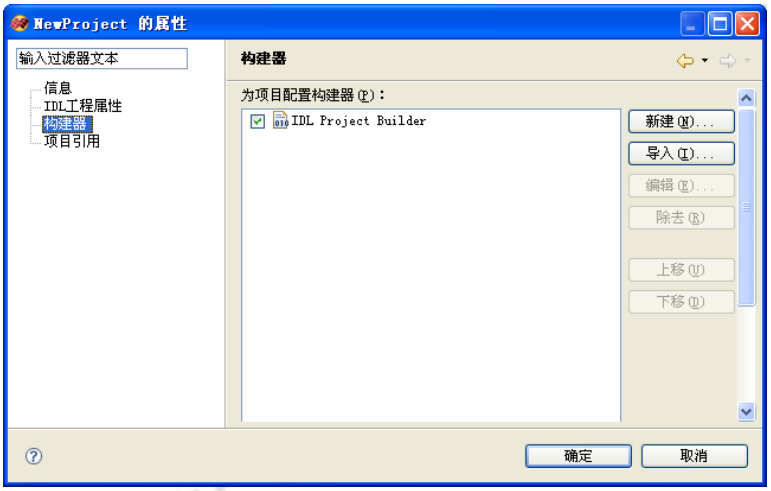


图 2-27 IDL 的项目资源管理器

项目引用对话框中选择这个项目需要引用的项目名称。

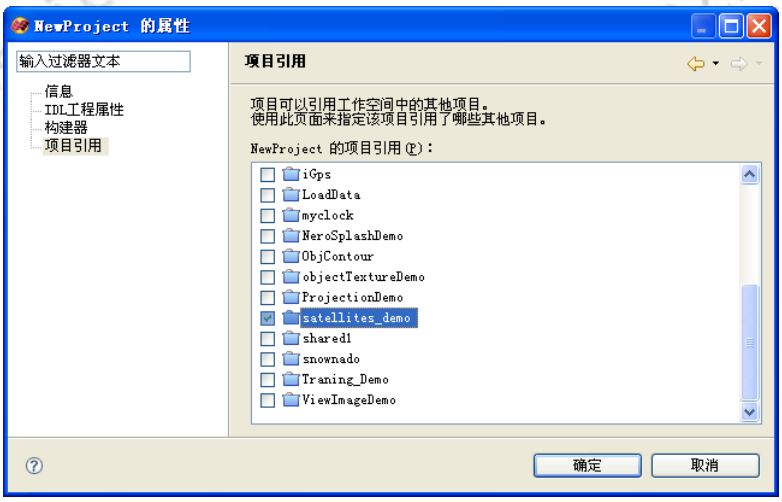


图 2-28 IDL 的项目属性

2.2.3.3 工程添加文件

菜单选择“文件”-“新建”-“IDL 源文件”或在工具栏上选择；

如创建链接文件，在系统菜单或项目右键菜单中选择“新建”-“文件夹”，选择“高级”选项，在高级选项中选择链接的文件夹路径，见图 2-29。

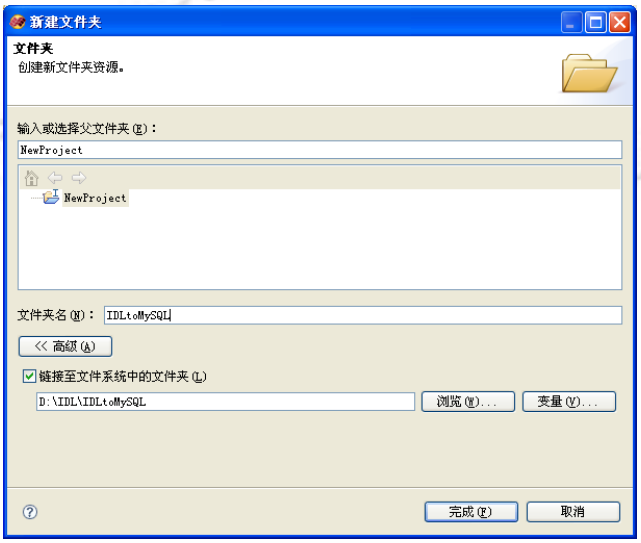


图 2-29 链接文件夹

2.2.3.4 编译工程

- 选择菜单项目→构建项目；
- 在项目资源管理器中该工程目录上点击右键，选择构建项目；
- 快捷键 Ctrl+B；
- 点击工具栏图标；

2.2.3.5 运行工程

- 选择菜单运行→运行工程；
- 在项目资源管理器中该工程目录上点击右键，选择运行工程；
- 快捷键 Ctrl+Shift+F8；
- 点击工具栏图标

2.2.4 文件（file）

2.2.4.1 打开文件

载入一个文件有两种方式：

- 在菜单中选中文件-打开文件；
- 工具栏按钮中点击。

2.2.4.2 创建文件

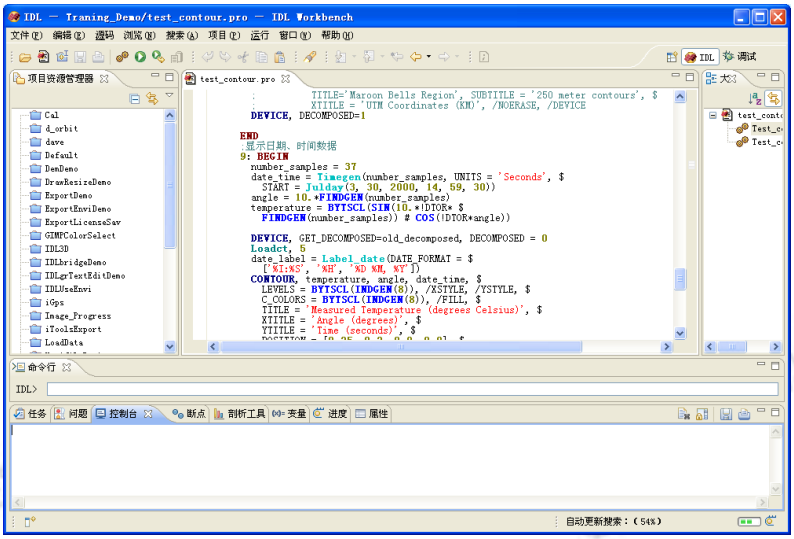
- 创建一个新文件有三种方式：
- 菜单中选择文件-新建-IDL 源文件；
- 或选择文件-新建-其他-IDL 源文件；
- 工具栏按钮中点击。

2.2.4.3 保存文件

- 菜单中选择文件-保存；
- 菜单中选择文件-另存为；
- 菜单中选择文件-全部保存；
- 工具栏按钮中点击。

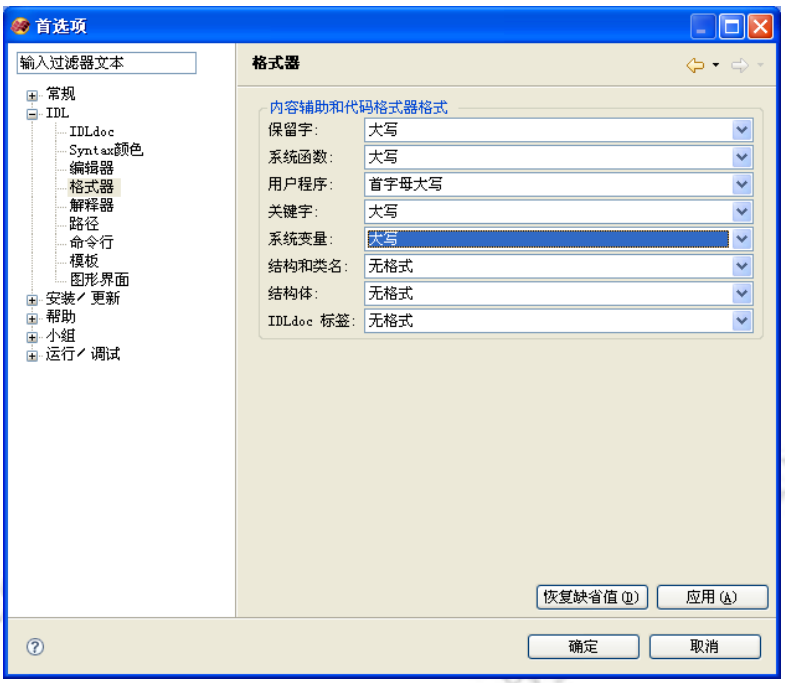
2.2.4.4 编辑工具

- 语法的高亮显示

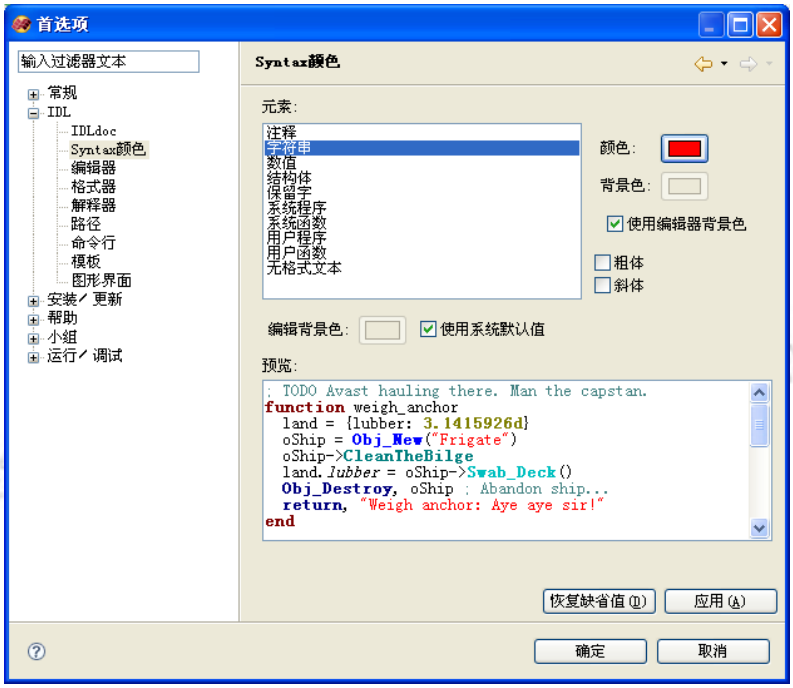


- 自定义显示格式

首选项→IDL→格式器设置



- 自定义显示颜色



➢ 鼠标悬停帮助

对程序中的函数需要帮助时可以把鼠标指针移到函数位置，系统会弹出该函数的帮助，见图 2-30。

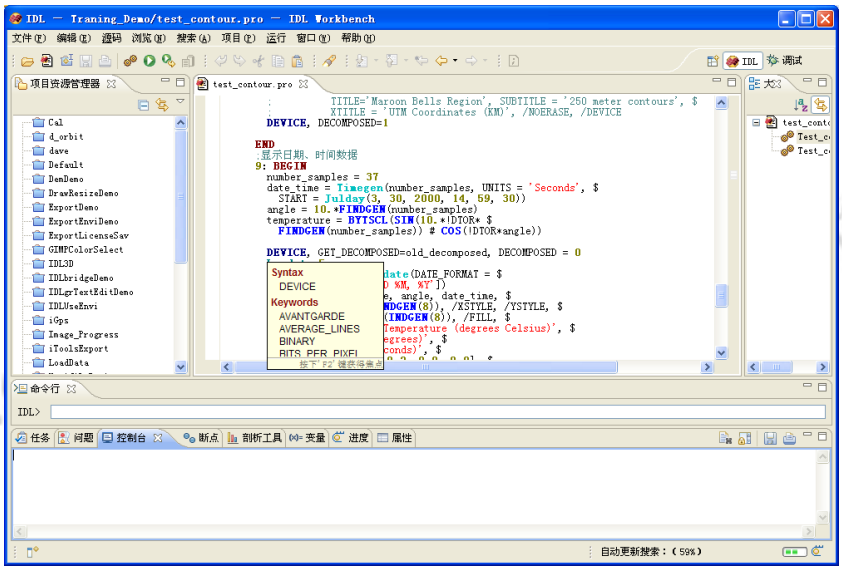
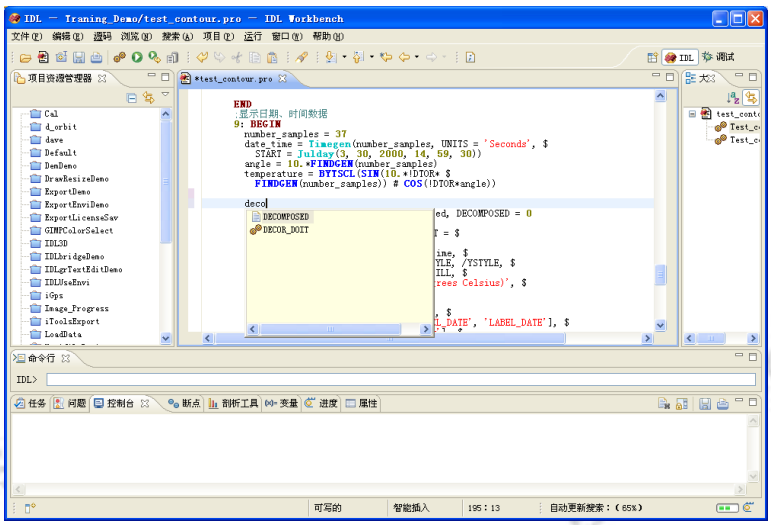
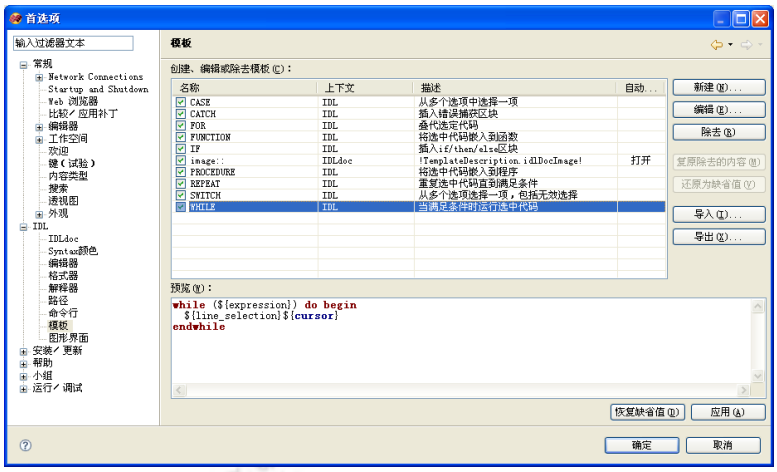


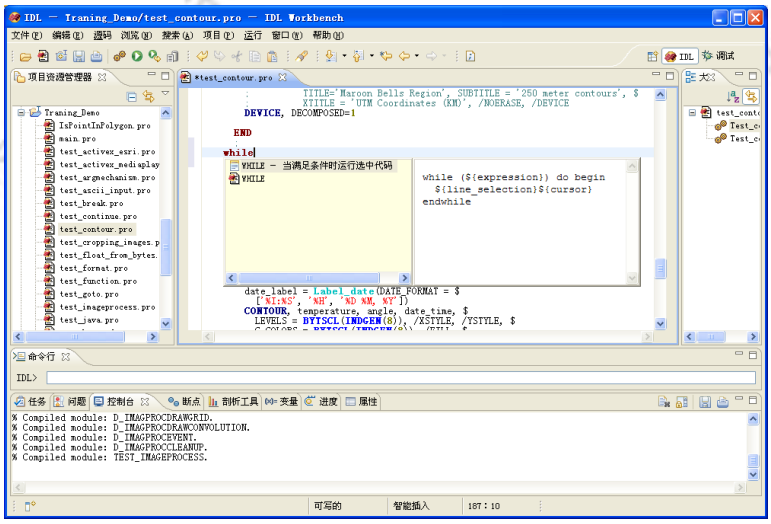
图 2-30 IDL 编辑器中的悬停帮助

➢ 代码辅助

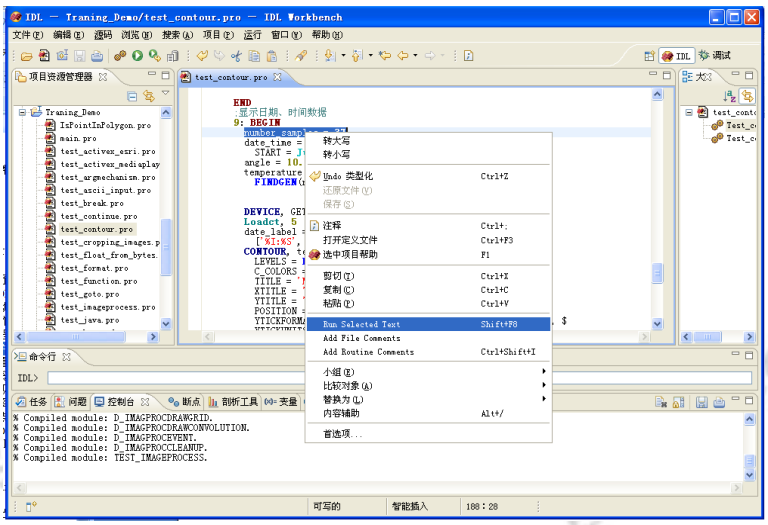

IDL — Traning_Demo/test_contour.pro — IDL Workbench
文件(F) 编辑(E) 逻辑 浏览(N) 搜索(A) 项目(P) 运行 窗口(W) 帮助(H)
项目资源管理器
*test_contour.pro
END
;显示日期、时间数据
9: BEGIN
number_samples = 37
date_time = Timegen(number_samples, UNITS = 'Seconds', $
START = Julday(3, 30, 2000, 14, 59, 30))
angle = 10.*FINDGEN(number_samples)
temperature = BYTSCL(SIN(10.*!DTOR* $
FINDGEN(number_samples)) # COS(!DTOR*angle))
deco
DECOMPOSED
DECOR_DOIT
命令行
IDL>
任务 问题 控制台 断点 剖析工具 变量 进度 属性
可写的
智能插入
195 : 13
自动更新搜索：(65%)

- 模板工具

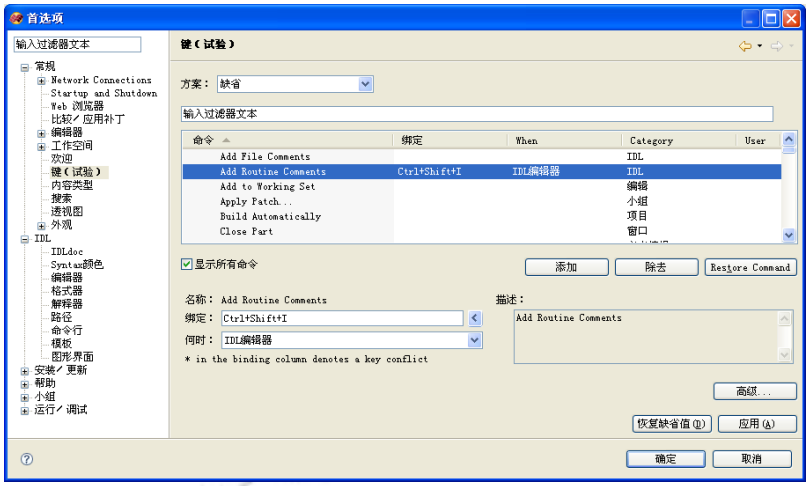

首选项
输入过滤器文本
常规
Network Connections
Startup and Shutdown
Web 浏览器
比较/应用补丁
编辑器
工作空间
欢迎
键(试验)
内容类型
搜索
透视图
外观
IDL
IDLdoc
Syntax颜色
编辑器
格式器
解释器
路径
命令行
模板
图形界面
安装/更新
帮助
小组
运行/调试
模板
创建、编辑或除去模板(C):
名称
上下文
描述
自动...
CASE IDL 从多个选项中选择一项
CATCH IDL 插入错误捕获区块
FOR IDL 叠代选定代码
FUNCTION IDL 将选中代码嵌入到函数
IF IDL 插入if/then/else区块
image:: IDLdoc !TemplateDescription.idlDocImage! 打开
PROCEDURE IDL 将选中代码嵌入到程序
REPEAT IDL 重复选中代码直到满足条件
SWITCH IDL 从多个选项选择一项，包括无效选择
WHILE IDL 当满足条件时运行选中代码
新建(N)...
编辑(E)...
除去(R)
复原除去的内容(U)
还原为缺省值(V)
导入(I)...
导出(X)...
预览(V):
while (${expression}) do begin
${line_selection}${cursor}
endwhile
恢复缺省值(D)
应用(A)
确定
取消


IDL — Traning_Demo/test_contour.pro — IDL Workbench
文件(F) 编辑(E) 逻辑 浏览(N) 搜索(A) 项目(P) 运行 窗口(W) 帮助(H)
项目资源管理器
Traning_Demo
IsPointInPolygon.pro
main.pro
test_activex_esri.pro
test_activex_mediaplay
test_argmechanism.pro
test_ascii_input.pro
test_break.pro
test_continue.pro
test_contour.pro
test_cropping_images.p
test_float_from_bytes.
test_format.pro
test_function.pro
test_goto.pro
test_imageprocess.pro
test_java.pro
*test_contour.pro
TITLE='Maroon Bells Region', SUBTITLE = '250 meter contours', $
XTITLE = 'UTM Coordinates (KM)', /NOERASE, /DEVICE
DEVICE, DECOMPOSED=1
END
while
WHILE - 当满足条件时运行选中代码
WHILE
while (${expression}) do begin
${line_selection}${cursor}
endwhile
date_label = Label_date(DATE_FORMAT = $
['%I:%S', '%H', '%D %M, %Y'])
CONTOUR, temperature, angle, date_time, $
LEVELS = BYTSCL(INDGEN(6)), /XSTYLE, /YSTYLE, $
命令行
IDL>
任务 问题 控制台 断点 剖析工具 变量 进度 属性
% Compiled module: D_IMAGPROCDRAWGRID.
% Compiled module: D_IMAGPROCDRAWCONVOLUTION.
% Compiled module: D_IMAGPROCEVENT.
% Compiled module: D_IMAGPROCCLEANUP.
% Compiled module: TEST_IMAGEPROCESS.
可写的
智能插入
187 : 10

- 运行选择代码



- 常用快捷键



Ctrl+S 保存文件
Ctrl+N 新建
Ctrl+F8 编译文件
Ctrl+B 构建工程
Alt+/ 开启内容辅助
Ctrl+Shift+F 格式化 IDL 源代码文件
Ctrl+Shift+U 更改为大写
Ctrl+U 更改为小写
Ctrl+Shift+I 添加程序辅助

2.2.4.5 编译和运行

编译运行有两种方式：

在菜单中选择**运行-编译 filename.pro** 然后选择**运行 filename**；

在命令行中输入**.Compile filename**，然后输入 **filename** 即可；

2.2.5 查看器（View）

可在菜单[窗口]→[显示视图]下进行显示或关闭设置。查看器是 idl 工作台上面的可以移动的窗口，能够对显示数据、分析和跟命令行进行交互操作，如控制台工具显示了命令行及编译程序时的所有输出信息。见图 2-31，各个查看器可以进行最大、最小及显示、隐藏设置。

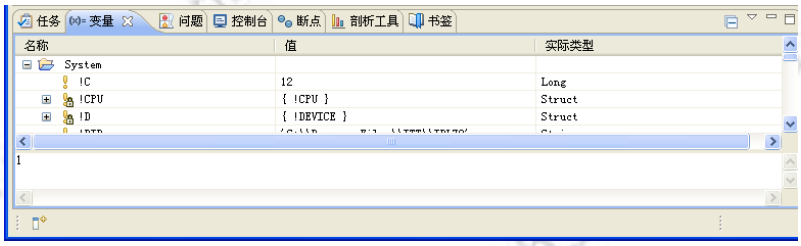


图 2-31 IDL 工作台的查看器

2.2.6 命令行及命令历史查看器

命令行及命令历史查看器分别可以进行命令的输入和历史命令的查看，见图 2-32。

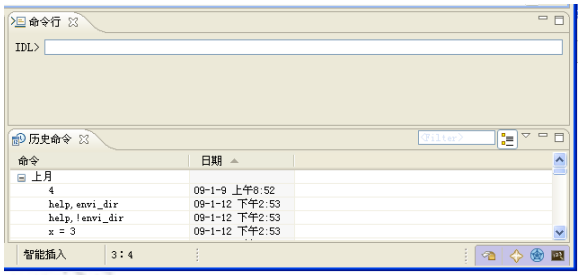


图 2-32 命令行及命令历史查看器窗口

2.2.7 透视图

透视图是工作台窗口的初始设置和排列状态。每种透视窗口都实现一定的功能，如 IDL 的普通透视图用来编辑 IDL 的.pro 文件，而调试透视图则用来调试 IDL 的程序。在使用 IDL 的过程中你可能会频繁的切换两种透视图，透视图的切换见图 2-33。

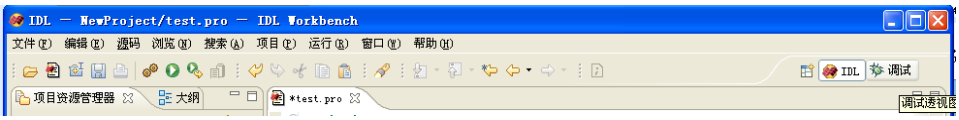


图 2-33 透视图窗口切换

2.2.7.1 断点及调试

2.2.7.1.1 断点

设置断点首先将你把光标放置在你想设置断点的那行，按下 Ctr+Shitf+B 或点击菜单运行-切换行断点或双击该行的最左边。

2.2.7.1.2 变量查看

当运行一个存在断点的程序时，会在断点处停止。此时为了程序的调试需要，一般要看变量的值是否符合程序的编写要求，可以通过命令行、鼠标悬停显示或变量查看器三种方式来实现。变量查看器是一个浮动的窗口，见图 2-34。

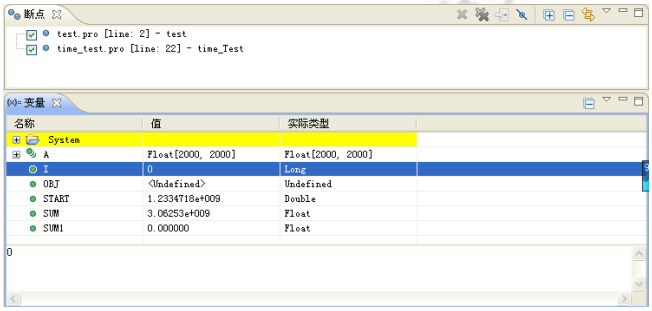


图 2-34 程序调试中的变量查看器

2.2.7.1.3 代码步进和步过

代码在调试状态下，由下面几个方式控制，图 2-35，首先是继续按钮，点击该按钮可以使得当前断点无效，程序继续执行。

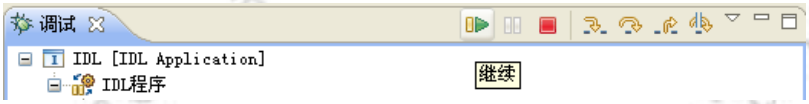


图 2-35 调试视图下继续按钮


红色的为停止按钮，后面依次为单步跳入、单步跳过、单步返回和跳过下段代码。单步跳入（步进）就是调试程序的时候，每一个子程序都跟进去，详细调试每一步代码。相反地，单步跳过（步过）就是不调试子程序。一般来说，在调试关键代码时，用单步跳入，因为我们可不想错过任何一个关键的地方，其他时候就用单步跳过。

2.2.7.1.4 获得程序运行时间

可以通过 PROFILER 查看自定义的进程调用的频率及花费的时间。

```
; First ensure that the routine you want to profile is compiled.
```

```
void = DIST(500)

Now include both user routines and system routines.
PROFILER
PROFILER, /SYSTEM
; Now perform the actual commands to profile.
a = DIST(500)
TV, a; Retrieve the profiling results.
PROFILER, /REPORT
Module                  Type    Count       Only(s)     Avg.(s)       Time(s)     Avg.(s)
DIST                    (U)         1      0.001626    0.001626      0.004146    0.004146
FINDGEN                 (S)         1      0.000006    0.000006      0.000006    0.000006
FLTARR                  (S)         1      0.000020    0.000020      0.000020    0.000020
N_ELEMENTS              (S)         1      0.000003    0.000003      0.000003    0.000003
ON_ERROR                (S)         1      0.000003    0.000003      0.000003    0.000003
SQRT                    (S)       251      0.002489    0.000010      0.002489    0.000010
TV                      (S)         1      0.082959    0.082959      0.082959    0.082959
WIDGET_PROCESS_EVENTS
                        (S)        11      0.004772    0.000434      0.004772    0.000434
```

2.2.7.2 IDL 的路径管理

IDL 的路径管理是非常重要的，IDL 中的!Path 变量中存在着一系列的路径列表来供系统来寻找 pro 和 sav 文件，如果系统解释器遇到了没有编译过的过程名，那么它会在!Path 列表的路径中寻找同名的过程或 sav，找到后会自动编译该过程并继续运行。如在命令行中运行：

```
IDL> map_set,/continents

% Compiled module: MAP_SET.

% Compiled module: CROSSP.

% Compiled module: MAP_CONTINENTS.

% Compiled module: FILEPATH.

% Compiled module: PATH_SEP.

% Compiled module: REVERSE.
```

路径自动更新设置见图 2-36，在当前工程的属性菜单中。

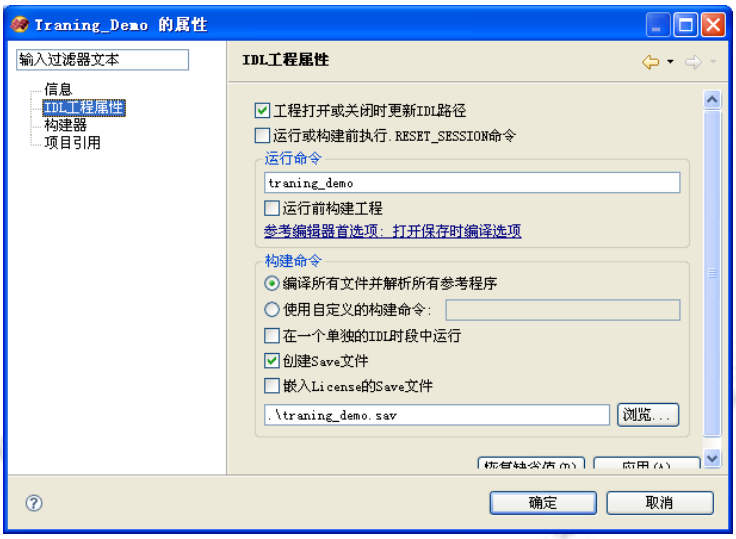


图 2-36 IDL 路径更新

系统的路径管理选项，在菜单的窗口-首选项-IDL-路径中，可分别对 IDL 路径、DLM 路径和帮助路径进行设置，见图 2-37。

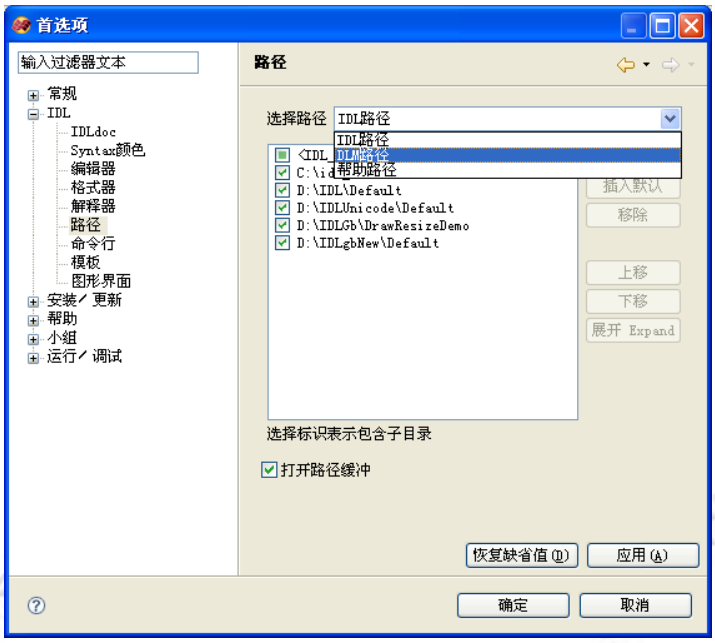


图 2-37 IDL 路径管理

2.2.8 获取帮助

2.2.8.1 函数帮助

在命令行中输入?+A（A 是你所要了解的过程函数或对象）

2.2.8.2 选中内容帮助

鼠标选中要查看帮助的函数后按 F1 或在菜单中选择[帮助]-[选中内容帮助]，则帮助系统会自动查找与选择的内容最接近的帮助部分并显示。

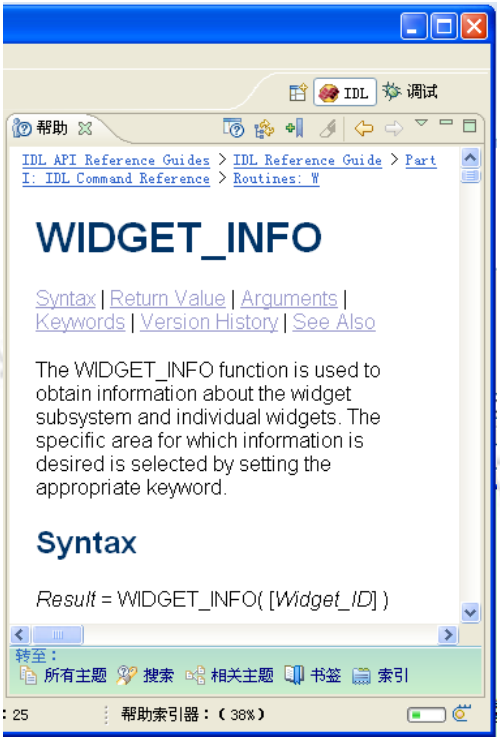


图 2-38 IDL 编辑器的选中帮助功能

2.2.8.3 联机帮助系统

- 可以直接选菜单 Help 下的 Help Contents 按钮
- Windows 下命令行中输入 ONLINE_HELP，UNIX 下输入 IDLHELP

在菜单中选择[帮助]-[帮助内容]会显示帮助界面，见图 2-39。包括了帮助系统的目录、索引、书签和搜索等功能，在浏览时可以进行前进、后退和返回主页，并能进行打印。

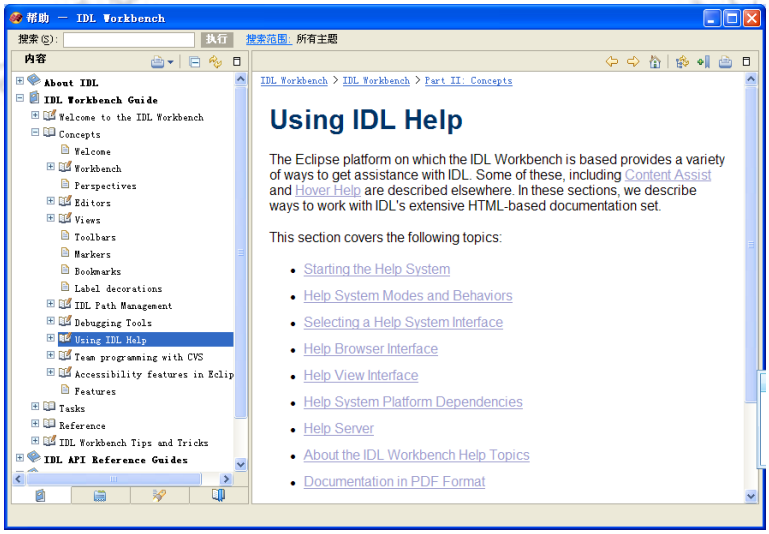


图 2-39 IDL 编辑器中的帮助系统

2.2.8.4 对帮助进行设置

在菜单的窗口-首选项中可以对帮助系统进行一系列的参数设置，见图 2-40。

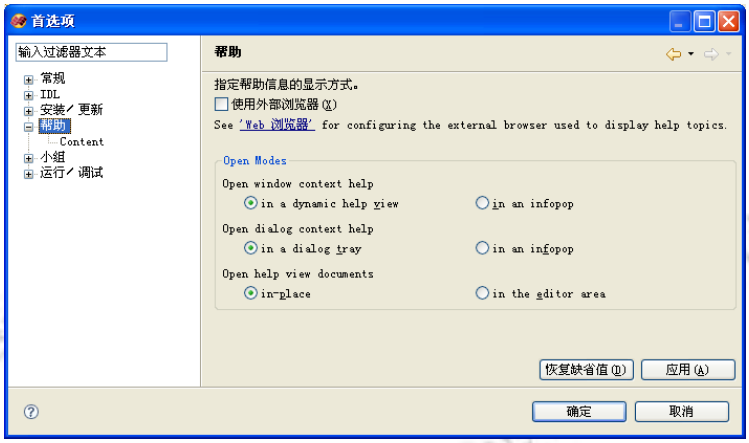
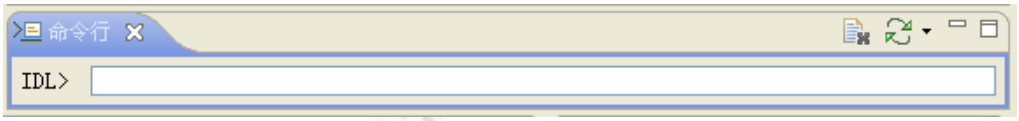


图 2-40 首选项中帮助系统设置

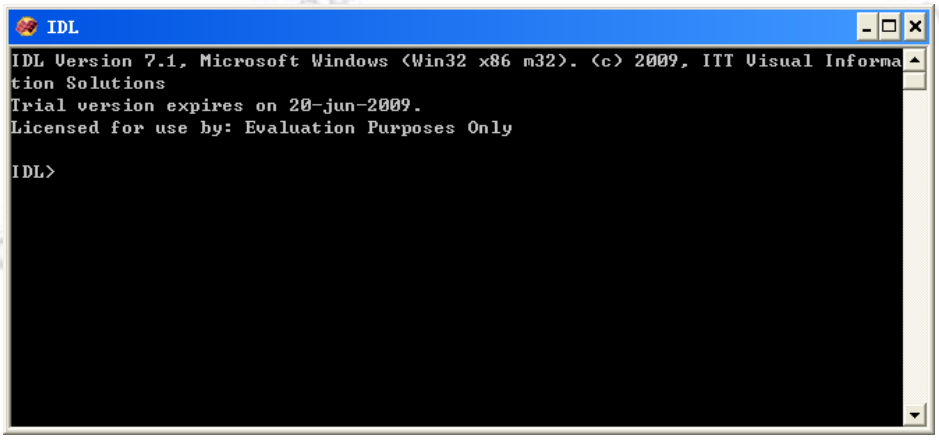
2.3 编写 IDL 程序

2.3.1 命令行模式

命令行方式是 IDL 的一大特色，为程序编写和调试过程中的功能测试提供了极大地便利。



IDL7.1 的 Windows 版本新增加了命令行，如下图所示。



DotCommand

该系列命令即可在命令行中进行源代码（pro 文件）的编辑、编译、调试及运行等工作，且仅能在命令行下执行。

命 令	功 能
.COMPILE	编译代码
.CONTINUE	继续执行代码
.EDIT	在编辑器中打开代码以便编辑
.FULL_RESET_SESSION	编译器完全重置，比.RESET_SESSION 要彻底；
.GO	运行上次编译过的程序
.OUT	继续执行代码直到返回；
.RESET_SESSION	编译器环境重置；
.RETURN	程序返回；
.RENEW	新建一个 pro
.RUN	编译内存中的代码并立即执行；
.SKIP	跳过一个或多个模块；
.STEP	调试程序中的移动到下一个模块；
.STEPOVER	程序运行到另一个节点处；
.TRACE	程序报错时继续执行；

2.3.2 批处理模式

最简单的程序是一个 IDL 批处理文件。一个批处理文件由一系列命令组成，这与在 IDL 命令行敲入的命令完全一样。大多数人用批处理文件是为了自动执行自己在 IDL 命令行一次又一次敲入的命令。

例如，在创建线画图中我们为了要创建数据，并画出图形，那么可以在命令行中键入如下命令：

```
IDL>curve=Indgen(101)
IDL>curve=curve^2
IDL>time=FIndGen(101)*(6.0/100)
IDL>Plot,time,curve,XTitle='Time Axis',$
IDL>YTitle=' Displacement ', Title='Experiment
```

这五行代码并不多，但键入三、四次之后，可能已决定把他们放在一个名为 plotline.pro 文本文件中,这文件就是所谓的批处理文件。**批处理文件同样只有存储在搜索目录下**，IDL 才能调用。注意，批处理文件中只需键入命令行中内容就可以了，无需 PRO、END 之类。plotline.pro 中的内容是：

```
curve=Indgen(101)
curve=curve^2
time=FIndGen(101)*(6.0/100)
Plot,time,curve,XTitle='Time Axis',$
YTitle=' Displacement ', Title='Experiment
```

要执行该文件中的命令，可如下所示：

```
IDL> @plotline.pro
```

IDL 会严格执行批处理文件中的命令，就像在命令行上键入一样。这意味着有必要用行续行符（$）和其他命令行语言来让 IDL 确认键入的命令。如果在文件中的命令输入错误，那么出现的错误结果和在命令行键入命令出现的错误结果是一样的。

2.3.3 编译模式

跟其他语言的运行模式一致，先编写程序文件，然后进行编译运行。

参考 2.2.3 工程，2.2.4 文件章节。

2.3.4 IDL 程序

IDL 程序和批处理文件在很多方面很相似，但也存在着很大的区别。像批处理文件一样，一个主程序也包含一系列命令。但与之不同的是，这些命令必须以 END 语句结束。

批处理文件和 IDL 主程序最大的区别就是主程序的命令语句先由 IDL 编译器编译成程序模块，然后才执行代码。这就是为什么在主程序中可以有多行控制语句的原因。

编写主程序的最大优点是在程序中定义的变量可以在 IDL 开发环境(IDLDE)中使用，IDL 的命令都是在那里键入并被解释的。换句话说，在主程序中定义的变量的作用范围是 IDL 整个开发环境，因而它可以被 IDL 其他命令使用。

新建一个 pro 文件，编写内容如下：

```
pro firstIDL
   print,'first IDL'
   tmp = dialog_Message('first idl',/information)
end
```

但是更为通常的是，往往希望限定变量的作用范围，以使它们不致于占用大量的内存。在编程过程中，使用大量的全局变量会使内存的使用效率大大降低。为此，大部分应用程序都用过程和函数来编写。

2.3.4.1 过程和函数的区别

IDL 命令要么是过程，要么是函数首先，在函数命令中，位置参数和关键字参数放在一对圆括号中的。在过程命令中，参数和关键字仅排列在一个命令行上。但是，最重要的区别是 IDL 函数会返回一个值，等号左边的一个变量用于返回该值。这是 IDL 中函数命令和过程命令根本的区别。

函数命令总是返回一个值，这个数值必须赋给一个变量。函数返回值可能是任一种 IDL 变量，包括数值、数组或结构。

例子

```
PRO test_Function
  a = 1
  b = 2
  result = fun_total(a,b)
  PRINT,result
END
;
FUNCTION fun_total, x,y
  RETURN, x+y
END
```

2.4 网络资源

[http://www.ittvis.com/] ITT VIS
[http://www.esrichina-bj.cn/] ESRI 中国（北京）有限公司
[http://bbs.esrichina-bj.cn/ESRI/index.php] ESRI 中国社区
[http://groups.google.com/group/comp.lang.idl-pvwave/topics]Google-IDLPVWAVE
[http://www.dfanning.com/] IDL 大师 David 个人网站
[http://www.kilvarock.com/] Ronn Kling 个人网站
…

3 IDL 语法基础

3.1 变量

IDL 中的变量分为局部变量和系统变量。在 IDL 程序中，二者的区别在于生命周期不一样，局部变量仅仅在所在的函数或过程中有效，而系统变量则在始终有效或某个程序的周期中有效。

3.1.1 局部变量

3.1.1.1 命名规则

变量名称的最大长度不超过 255 个字符，变量的首位只能是**字母**和**下划线**，中后部只能是**字母、数字、下划线“_”和连接符“$”**。如下正确：

abc_3$d ok_24_bit IDL_type variable _day_month_year

如下错误：

abc.cha one%file 4_lists $file

变量相关函数 IDL_VALIDNAME

该函数用来检测字符串是否可以作为 IDL 下的变量，并能够将无效字符转换为“_”并返回变量。调用格式为 IDL_VALIDNAME (String [, /CONVERT_ALL] [, /CONVERT_SPACES])，

```
IDL> print,IDL_VALIDNAME('idl')
idl
IDL> print,IDL_VALIDNAME('var')
var
IDL> print,IDL_VALIDNAME('5var')

IDL> print,IDL_VALIDNAME(' 5var',/CONVERT_SPACES)
_5var
IDL> print,IDL_VALIDNAME('5var',/convert_all)
_5var
IDL> print,IDL_VALIDNAME('5v#ar',/convert_all)
_5v_ar
```

3.1.1.2 基本数据类型

跟其他语言比如 C 不一样，IDL 下的变量无需事先申明，也就是可以通过赋值语句来进行变量、数组定义的。

IDL 中有 14 种基本数据类型。在表 3-1 中列出了基本数据类型的每个类型的变量的字节大小、创建方式、范围和类型转换函数等。除了数据类型外，一个变

量有一个组织结构。有效的组织结构有标量（例如单个数值）、矢量（真正的一维数组）、数组（最高可达八维）和 IDL 结构（能包含各种数据类型的变量和组织结构，结构中独立的组成部分称为字段）。

表 3-1 变量类型及类型转换函数

数据类型	字节数	范围	创建变量	类型转换
字节型	1	0 至 255	Var=0B	byte()
16 位有符号整型	2	-32768 至 32767	Var=0	fix()
32 位有符号长整型	4	-2^{31} 至 $2^{31}-1$	Var=0L	long()
64 位有符号整型	8	-2^{63} 至 $2^{63}-1$	Var=0LL	long64()
16 位无符号整型	2	0 至 65535	Var=0U	uint()
32 位无符号长整型	4	0 至 $2^{32}-1$	Var=0UL	ulong()
64 位无符号整型	8	0 至 $2^{64}-1$	Var=0ULL	ulong64()
浮点型	4	-10^{38} 至 $10^{38}-1$	Var=0.0	float()
双精度浮点型	8	-10^{308} 至 $10^{308}-1$	Var=0.0D	double()
复数	8	-10^{38} 至 $10^{38}-1$	Var=Complex(0.0，0.0)	complex()
双精度复数	16	-10^{308} 至 $10^{308}-1$	Var=Dcomplex(0.0D，0.0D)	dcomplex()
字符串	0-32767	None	Var=’’或 Var=””	string()
指针	4	None	Var=Ptr_New()	
对象	4	None	Var=Obj_New()	

```
IDL> var = 4B
IDL> help,var
VAR             BYTE      =       4
IDL> var = 256b

var = 256
        ^
Byte constant must be less than 256.
IDL> var = 256
IDL> help,var
VAR             INT       =         256
IDL> var = 256L
IDL> help,var
VAR             LONG      =             256
IDL> var = 256LL
IDL> help,var
VAR             LONG64    =                         256
IDL> var = 256.
IDL> help,var
VAR             FLOAT     =         256.000
IDL> var = 256D
IDL> help,var
```

```
VAR             DOUBLE    =          256.00000
IDL> var ='abcdef'
IDL> help,var
VAR             STRING    = 'abcdef'
IDL> var ="abc'def"
IDL> help,var
VAR             STRING    = 'abc'def'
IDL> var ='abc"def'
IDL> help,var
VAR             STRING    = 'abc"def'
IDL> print,var
abc"def
```

3.1.1.3 类型转换

IDL 下的类型转换函数见表 3-2

表 3-2　变量类型转换操作及结果

类型转换	**函数名称**	**示例**	
		操作	**结果**
字节型	BYTE	BYTE(1.2)	1B
整型	FIX	FIX(2.5)	2
无符号整型	UINT	UINT([**5.5**,-**3**])	5　65533
长整型	LONG	LONG(65538.5)	65538
无符号长整型	ULONG	ULONG([**5.5**,-**3**])	5　4294967293
64 位长整型	LONG64	LONG64([**5.5**,-**3**])	5　-3
无符号 64 位长整型	ULONG64	ULONG64([**5.5**,-**3**])	5 18446744073709551613
浮点型	FLOAT	FLOAT([5.5,-3])	5.50000　-3.00000
双精度类型	DOUBLE	DOUBLE([5.5,-3])	5.5000000　-3.0000000
复数类型	COMPLEX	COMPLEX(1, 2)	(1.00000,　2.00000)
双精度复数类型	DCOMPLEX	DCOMPLEX(1, 2)	(1.0000000,　2.0000000)

在进行数据的转换中需要注意转换后的数据的界限问题。如下面操作：

```
IDL> A = 33000
IDL> B = FIX(A) ;B is silently truncated.
IDL> PRINT, B
```

```
-32536
```

出现这个错误是因为整型变量是 16Bit 的，最大值为 32767。

```
IDL> b = 7b
IDL> print,string(b)
 •
IDL> help,string(Fix(b))
<Expression>    STRING    = '          7'
```

3.1.1.4 变量的特点

3.1.1.4.1 动态性

IDL 在运行过程中能随时修改变量的数据类型和组织结构，这个特性给 IDL 的变量使用带来了极大的便利。如：

```
IDL>num = 6              ;  num 是一个整型变量
IDL>num = num*5.2        ;  num 变为一个浮点型变量
```

变量 num 初始化为一个整数，经过数学运算重新赋值，它被动态地改变成浮点数值。这是因为 IDL 在数学计算中为了保证高精度，将低精度的数据类型提升为高精度的数据类型。当 num 被再赋值（在等号的左边），它被提升为一个浮点数去保持等号右边计算的精度。

注意，下面这种情况下不进行类型转换。

```
IDL> data = bytarr(3,3)
IDL> help,data
DATA                BYTE      = Array[3, 3]
IDL> data[*,0] = [1,4,258]
IDL> print,data
   1   4   2
   0   0   0
   0   0   0
```

3.1.1.4.2 整型变量精度

如果不注意，整型变量运算过程中会出现丢失精度的现象，可以看下面的例子：

```
IDL> result=8/5
```

也许这个表达式的期望结果是 1.6，但结果却是 1。这种情况的发生是因为运算式右边的两个数字为整数。当然，找到这个问题的原因并不难，但有时由整型数据导致的问题会更微小，以致于难以发现。

例如：假如想知道 IDL 图形窗口的比率。窗口的大小（像素点或整数值）被储存在两个系统变量中。会写出如下的 IDL 代码：

```
IDL>aspect=!D.X_Size / !D.Y_Size
```

可能需要花掉很长的时间来找为什么比率为零的错误。正确的方法是在进行除法运算时时刻注意写出一些代码以强制将一个整数值变成一个浮点，如下：

```
IDL>aspect=Float(!D.X_Size) / !D.Y_Size
```

这样处理会后的比率就是一个所期望的浮点数了。

3.1.2 系统变量

IDL 系统中系统变量可以分为两种，一类是预定义系统变量，另一种是自定义的系统变量，下面分别进行介绍。

3.1.2.1 预定义系统变量

在这里介绍几组常用的系统变量，在直接图形法（图形只可输出，不能进行交互操作）输出中会常用到。

3.1.2.1.1 !d 系统变量

提供当前显示设备的信息。

表 3-3 当前显示设备信息

系统变量名	变量描述
!d.name	当前图形设备的名称
!d.n_colors	当前图形设备使用的颜色数
!d.table_size	当前分配的颜色表索引数
!d.x_size	当前图形窗口的宽（象素）
!d.y_size	当前图形窗口的高（象素）
!d.window	当前图形窗口的索引号(没创建和打开图形窗口，值为 - 1)

3.1.2.1.2 !p 系统变量

更改默认的画图设置，此类系统变量包含如下系统变量：

系统变量名	变量描述
!p.color	画图的颜色索引
!p.charsize	标签的字符大小
!p.linestyle	画线时的线性
!p.multi	指定在一个页面上画图的个数

!p.position	指定在一个页面上画图的位置
!p.psym	在画图时所用的符号样式
!p.subtitle	副标题（显示在 x 轴下）
!p.title	画图的标题

3.1.2.1.3 !x，!y，!z 系统变量

更改在画图上坐标轴的默认属性，此类系统变量包含如下系统变量（如前述系统变量，在前面加!x，!y，!z）：

系统变量名	变量描述
minor	在坐标轴上要显示的刻度数
range	设置坐标轴上显示数值的范围
tickbane	为每个主刻度设置一个注记数组（一般是字符串）
ticks	坐标轴上显示的刻度段数
title	坐标轴的标题

3.1.2.1.4 !CPU

```
IDL> help,!cpu,/str
** Structure !CPU, 6 tags, length=24, data length=24:
   HW_VECTOR       LONG                 0
   VECTOR_ENABLE   LONG                 0
   HW_NCPU         LONG                 2
   TPOOL_NTHREADS  LONG                 2
   TPOOL_MIN_ELTS  LONG            100000
   TPOOL_MAX_ELTS  LONG                 0
```

3.1.2.1.5 !DIR

```
IDL>help,!DIR ,/str
<Expression>    STRING    = 'C:\Program Files\ITT\IDL70'
```

3.1.2.1.6 !DLM_PATH

```
IDL> help,!DLM_PATH  ,/str
<Expression>    STRING    = 'C:\Program Files\ITT\IDL70\bin\bin.x86;C:\Program
Files\ITT\IDL'...
```

3.1.2.1.7 !EDIT_INPUT

```
IDL> help,!EDIT_INPUT ,/str
<Expression>    INT       =        1
```

3.1.2.1.8 !HELP_PATH

```
IDL> help,!HELP_PATH ,/str
```

```
<Expression>       STRING      = 'C:\Program Files\ITT\IDL70\help'
```

3.1.2.1.9 !JOURNAL

```
     IDL> help,!JOURNAL,/str
<Expression>       LONG          =           0
```

3.1.2.1.10 !MAKE_DLL

```
     IDL> help,!MAKE_DLL ,/str
     ** Structure !MAKE_DLL, 4 tags, length=48, data length=48:
        COMPILE_DIRECTORY
        STRING 'C:\Documents and Settings\Administrator\.idl\itt\compile_dir-1'...
        COMPILER_NAME     STRING      'Microsoft Visual C++ 8.0'
        CC    STRING    'cl %X -D_DLL -DMSWIN -DWIN32 -D_MT /nologo /I"C:\Program Files'...
   LD STRING 'link /out:%L /nologo /nodefaultlib /dll %O /def:%E "C:\Program'...
```

3.1.2.1.11 !MORE

```
     IDL> help,!MORE ,/str
<Expression>       INT           =           1
```

3.1.2.1.12 !PATH

```
     IDL> help,!PATH ,/str
<Expression> STRING = 'C:\Program Files\ITT\IDL70\lib\hook;C:\Program Files\ITT\IDL70\'...
```

3.1.2.1.13 !PROMPT

```
     IDL> help,!PROMPT ,/str
<Expression>       STRING      = 'IDL> '
```

3.1.2.1.14 !QUIET

```
     IDL> help,!QUIET ,/str
<Expression>       LONG          =                0
```

3.1.2.1.15 !VERSION

```
     IDL> help,!VERSION ,/str
     ** Structure !VERSION, 8 tags, length=76, data length=76:
        ARCH                STRING      'x86'
        OS                  STRING      'Win32'
        OS_FAMILY           STRING      'Windows'
        OS_NAME             STRING      'Microsoft Windows'
        RELEASE             STRING      '7.0.1'
        BUILD_DATE          STRING      'Mar 20 2008'
        MEMORY_BITS         INT                32
   FILE_OFFSET_BITS   INT                64
```

3.1.2.2 自定义系统变量

自定义的系统变量的名字需要!+变量名，是可以修改的系统变量，生命周期为初始化至程序的关闭。在使用中应尽量使之成只读变量而且要避免过多的使用，创建格式为：

Defsysv,变量名,变量值，例如

```
IDL> defsysv,'!sys_var',88
IDL> help,!sys_var
<Expression>    INT       =        88
```

3.2 数组

3.2.1 概述

程序设计中，为了处理方便， 把具有相同类型的若干变量按有序的形式组织起来。这些按序排列的同类数据元素的集合称为数组。IDL 是面向矩阵的第四代语言，对数组的维数为 0～8 维，在数组的处理时非常的灵活、高效。在后面详细的讲述中我们会了解到这些。而且注意，IDL 中数组下标的顺序先是列标，后是行标。这正好与已习惯的线性代数中的矩阵或行列式相反

3.2.2 创建数组

3.2.2.1 赋值创建

数组可以在 IDL 命令行中建立。建立一个两行三列的数组，如下所示：

```
IDL> array=[[1,2,3],[4,5,6]]
IDL> IDL> help,array
ARRAY           INT       = Array[3, 2]
```

3.2.2.2 函数创建

3.2.2.2.1 创建全零和索引数组

1. 通用函数

可以用 IDL 的通用函数进行全零数组或索引数组的创建，表 2-1 是常用的数据类型数组的创建方式。

表 3-4 不同类型的数组创建方式

数据类型	创建全 0 数组	创建索引数组
字节型	bytArr()	bindgen()

16 位有符号整型	intarr()	indgen()
32 位有符号长整型	lonarr()	lindgen()
64 位有符号整型	lon64arr()	l64indgen()
16 位无符号整型	uintarr()	uindgen()
32 位无符号长整型	ulongarr()	ulindgen()
64 位无符号整型	ulon64arr()	ul64indgen()
浮点型	fltarr()	findgen()
双精度浮点型	dblarr()	dindgen()
复数	complexarr()	cindgen()
双精度复数	dcomplexarr()	dcindgen()
字符串	strarr()	sindgen()
指针	ptrarr()	
对象	objarr()	

例如，创建 100×100 初始值为零的字节型数组，输入：

```
IDL>array=BytArr(100,100)
```

创建一个有 100 个元素的浮点型矢量，初始数值为从 0 到 99，输入：

```
IDL>vector=FIndGen(100)
```

2. 常规函数

MAKE_ARRAY()函数：是在程序运行过程中创建特定的数组。

调用格式为 Result = MAKE_ARRAY ([D1 [, ..., D8]] [, /INDEX] [, DIMENSION = vector] [, /NOZERO] [, SIZE=vector] [, TYPE=type_code] [, VALUE=value])

通过关键字的控制可以实现表 2-1 的函数的功能。

```
M = MAKE_ARRAY(3, 4, /INTEGER, VALUE = 5)
IDL> help,m
M               INT       = Array[3, 4]
IDL> print,m
      5       5       5
      5       5       5
      5       5       5
      5       5       5
```

3.2.2.2.2 创建特殊数组

1. RANDOMU()函数

RANDOMU 函数返回一个或多个均匀分布的浮点型数组，值在[0,1]之间。

Result = RANDOMU(Seed [, D1 [, ..., D8]] [[, BINOMIAL=[trials, probability]] [, /DOUBLE] [, GAMMA=integer{>0}] [, /NORMAL] [, POISSON=value] [, /UNIFORM] | [, /LONG]])

通过关键字的控制可以随机生成满足条件的随机数。

2. RANDOMN()函数

RANDOMN 函数返回一个或多个正态分布的浮点型数组，这些数的平均为零，标准偏差为 1 。

Result = RANDOMN(Seed [, D1 [, ..., D8]] [[, BINOMIAL=[trials, probability]] [, /DOUBLE] [, GAMMA=integer{>0}] [, /NORMAL] [, POISSON=value] [, /UNIFORM] | [, /LONG]])

通过关键字的控制可以随机生成满足条件的随机数。

3.2.3 数组的存储

IDL 中的矢量和数组均是以**先行后列**顺序存储的(IDL 最初是设计用来处理行扫描的卫星数据)，下标值的起始值是 0。

3.2.3.1 一维数组的存储

数组 arr[m]的存储方式：

arr[0] → arr[1] →… → arr[m-1]

3.2.3.2 二维数组的存储

m*n 的数组 arr[m,n]的存储方式：

arr[0,0] → arr[1,0] → arr[2,0]…arr[m-1,0] →

arr[0,1] → arr[1,1] → arr[2,1]…arr[m-1,1] →

…

arr[0,n-1] → arr[1, n-1] → arr[2, n-1]…arr[m-1, n-1]

```
IDL> array = indgen(4,3)
IDL> print,array
       0       1       2       3
       4       5       6       7
       8       9      10      11
```

3.2.3.3 三维数组存储

n*m*t 的数组 arr[n,m,t]的存储方式为

arr[0,0,0] → arr[1,0,0] → … → arr[n-1,0,0]

arr[0,1,0] → arr[1,1,0] → … → arr[n-1,1,0]→

… … …

arr[0,m-1,0]→ arr[1,m-1,0]→ … → arr[n-1,m-1,0]

arr[0,0,1] → arr[1,0,1] → … → arr[n-1,0,1]

arr[0,1,1] → arr[1,1,1] → … → arr[n-1,1,1]

… … …

arr[0,m-1,1]→ arr[1,m-1,1] → … → arr[n-1,m-1,1]

… … …

… … …

arr[0,0,t-1] → arr[1,0,t-1] → … → arr[n-1,0,t-1]

arr[0,1,t-1] → arr[1,1,t-1] → … → arr[n-1,1,t-1]

… … …

arr[0,m-1,t-1]→arr[1,m-1,t-1]→…→arr[n-1,m-1,t-1]

```
IDL> arr = indgen(2,3,4)
IDL> print,arr
       0       1
       2       3
       4       5

       6       7
       8       9
      10      11

      12      13
      14      15
      16      17
```

```
        18      19
        20      21
        22      23
```

3.2.4 使用数组

3.2.4.1 下标方式

可以使用下标来存取该数组中的每一个元素。要知道数组元素是以行顺序存储的，所以要获得数组中的第四个元素，可以：

```
IDL>array = indgen(8)
IDL> print,array
       0       1       2       3       4       5       6       7
IDL>Print, array[3]
3
```

3.2.4.2 向量方式

倘若要存取数组中的第一，二，四和第六个元素，可键入：

```
IDL>indices=[0,1,3,5]
IDL>Print，array[indices]
   0    1    3    5
```

3.2.4.3 子数组的提取

创建数组

```
IDL>data = Indgen(20,30)
```

需要取出第 6-10 列和第 12～15 行的数据，可键入：

```
IDL>subarray = data[5:9,11:14]
help,subarray
SUBARRAY          INT          = Array[5, 4]
```

要取出一个第 10 行的矢量，键入：

```
IDL>vector = data[*，9]
```

要提取出一个数据为数组中最后 5 行的数组，键入：

```
IDL>subarray = data[*,25:29]
```

```
IDL>Help，  subarray
SUBARRAY          INT          = Array[20, 5]
```

可以看出子数组是一个 5 行 20 列的数组。

3.2.5 运算

3.2.5.1 数组运算

3.2.5.1.1 数组求大(<)、求小(>)和求余

数组的求大(<)、求小(>)和求余(mod)操作返回的是同维数组的计算结果。

```
IDL> arr = indgen(4)
IDL> print,arr
       0       1       2       3
IDL> print,arr > 3
       3       3       3       3
IDL> print,arr < 3
       0       1       2       3
IDL> arr = indgen(6)
IDL> print,arr mod 3
       0       1       2       0       1       2
```

3.2.5.1.2 数组与数运算

数组与数的运算遵循的基本规律是，数组中的每一个元素与数发生运算关系。例如：

```
IDL>A=IndGen(5)
IDL>Print，A
0       1       2       3       4
IDL>B=A+2
IDL>Print，B
2       3       4       5       6
IDL>C=A*2
IDL>Print，C
```

```
0       2       4       6       8
IDL>D=A^2
IDL>Print，D
1         2         4         8        16
```

3.2.5.1.3 数组与数组运算

数组与数组的运算结果的元素个数与运算的最小数组个数一致。

```
IDL> arr = [2,4,6,8]
IDL> darr = [2,4]
IDL> print,arr+darr
          4          8
IDL> help,arr+darr
<Expression>      INT           = Array[2]
IDL> darr = [2]
IDL> print,arr*darr
          4
IDL> help,arr*darr
<Expression>      INT           = Array[1]
IDL> darr = [2,4]
IDL> help,arr*darr
<Expression>      INT           = Array[2]
IDL> print,arr*darr
          4         16
IDL> darr = [1,2,3]
IDL> print,arr/darr
          2          2          2
IDL> darr1 = [1,2,3,4]
IDL> print,arr/darr1
     2          2          2          2
```

3.2.5.2 相关函数

3.2.5.2.1 数组信息 Size()

Size()函数能够返回数组的结构信息。

调用格式为 Result = SIZE(数组, [/KeyWords])。

如果不设置任何的关键字返回信息结构体，设置了关键字则返回对应的值，/N_DIMENSIONS , /N_ELEMENTS ,/DIMENSIONS , /TNAME, /TYPE：依次返回数组的维数、元素个数、每一维的维数、类型名称和类型代码信息。

```
IDL> data = FindGen(6,6)
IDL>help,size(data)
<Expression>    LONG          = Array[5]
IDL>help,size(data,/N_Elements)
<Expression>    LONG          =        36
IDL>help,size(data,/Dimensions)
<Expression>    LONG          = Array[2]
```

3.2.5.2.2 条件查找 Where()

WHERE()函数返回满足指定条件（即数组表达式）数组元素的下标所组成的一维数组。

调用格式为 Result =WHERE(数组表达式 [, count] [, COMPLEMENT=变量 1] [, /L64] [, NCOMPLEMENT=变量 2])。

关键字：count 返回的是符合指定条件的个数；

变量 1 是不满足条件的数组的下表信息；

变量 2 是不满足条件的数组的个数；

```
IDL> array = INDGEN(10)
IDL> PRINT, 'array = ', array
array =       0       1       2       3       4       5       6       7       8       9
IDL> B = WHERE(array GT 5, count, COMPLEMENT=B_C, NCOMPLEMENT=count_c)
IDL> PRINT, count
           4
IDL> PRINT, B
  6       7       8       9
IDL> PRINT, count_c
  6
IDL> PRINT, B_C
  0       1       2       3       4       5
```

3.2.5.2.3 大小调整

1. Reform()函数

Reform()函数可以使数组在元素个数不修改的前提下，改变数组的维数，方便了数组的动态使用。

调用格式为 *Result* = REFORM(*Array*, D_1 [, ..., D_8] [, 关键字])

```
IDL>a = Intarr(10,10,10)
IDL>c = Reform(a,200,5)
IDL>help,c
C               INT             = Array[200, 5]
IDL>d = Reform(a[0,*,*])
IDL>help,d
D               INT             = Array[10, 10]
IDL>e = Reform(a,[20,5])
% REFORM: New subscripts must not change the number elements in A.
% Execution halted at: $MAIN$
```

2. Rebin()函数

Rebin()函数可以修改数组的维数大小，将维数调整为 D1 [, ..., D8]指定的维数，且大小必须原数据的大小的整数倍，默认抽样算法是双线性内插。

调用格式为 Result = REBIN(数组, D1 [, ..., D8] [, /SAMPLE])

SAMPLE：设置该关键字则使用最近临值抽样法。

```
IDL> arr = [[0,6],[2,8]]
IDL> print,arr
           0       6
           2       8
IDL> print,rebin(arr,4,4)
           0       3       6       6
           1       4       7       7
           2       5       8       8
           2       5       8       8
IDL> print,rebin(arr,4,4,/sample)
           0       0       6       6
           0       0       6       6
           2       2       8       8
           2       2       8       8
IDL> print,rebin(arr,3,4)
% REBIN: Result dimensions must be integer factor of original dimensions
% Execution halted at: $MAIN$
```

3. Congrid()函数

Congrid()函数可以调整数组为同维任意大小的数组，默认在处理一维或二维数组时默认是最近邻重采样方式，处理三维数组时默认是双线性内插值。在数据缩小操作时，Rebin 做了插值处理，而 Congrid 仅仅是重新采样。

调用格式为 Result =Congrid(数组, X,Y,Z [,关键字])

INTERP:抽样采用线性内插法；

CUBIC：采用卷积内插法；

```
IDL> arr = [[0,6],[2,8]]
IDL> print,arr
       0       6
       2       8
IDL> print,congrid(arr,3,3)
       0       6       6
       0       6       6
       2       8       8
IDL> print,congrid(arr,3,4)
       0       6       6
       0       6       6
       2       8       8
       2       8       8
```

4. EXPAND

EXPAND 用来对二维数组进行大小修改，采用双线性内插算法。

格式为 EXPAND, A, Nx, Ny, Result [, FILLVAL=value] [, MAXVAL=value]

其中 A 为源数组，Nx、Ny 为修改后的维数，Result 为修改后的数组，FILLVAL 是填充数据，默认为-1；MAXVAL 是数据的最大值，大于该值的赋为 FILLVAL。

如：

```
IDL> d = indgen(4,4)
IDL> print,d
       0       1       2       3
       4       5       6       7
       8       9      10      11
      12      13      14      15
IDL> EXPAND, d, 5, 5, Result1
IDL> print,result1
       0       0       1       2       3
       3       3       4       5       6
       6       6       7       8       9
```

```
          9          9         10         11         12
         12         12         13         14         15
IDL> EXPAND, d, 5, 5, Result,MAXVAL=10
IDL> print,result
          0          0          1          2          3
          3          3          4          5          6
          6          6          7         -1         -1
         -1         -1         -1         -1         -1
         -1         -1         -1         -1         -1
```

5. INTERPOLATE()函数

INTERPOLATE ()函数可以调整数组为同维任意大小的数组，并能够任意定位插值点。

调用格式为 INTERPOLATE(数组, X [, Y [, Z]] [, 关键字])

GRID 关键字：可以使数组的插值点按网格生成，否则按线性方式生成；

Missing 关键字：如果插值点在当前数组外面时的值；

```
IDL> print,arr
      0.000000      1.00000
       2.00000      3.00000
IDL> PRINT, INTERPOLATE(arr, [0, .5, 1.5], [0, .5, 1.5])
      0.000000      1.50000      3.00000
IDL> PRINT, INTERPOLATE(arr, [0, .5, 1.5], [0, .5, 1.5],/grid)
      0.000000     0.500000      1.00000
       1.00000      1.50000      2.00000
       2.00000      2.50000      3.00000
IDL> PRINT, INTERPOLATE(arr, [0, .5, 1.5], [0, .5, 1.5],/grid,missing = 0)
      0.000000     0.500000     0.000000
       1.00000      1.50000     0.000000
      0.000000     0.000000     0.000000
```

3.2.5.2.4 数组反转 Reverse()

Reverse ()函数可以对数组进行反转。

调用格式：result =Reverse (数组, index[, 关键字])

index 是反转的数组的维数索引，它不能大于数组的维数。

```
IDL> print,arr
      0.000000      1.00000
       2.00000      3.00000
IDL> print,reverse(arr,1) ;行反转
       1.00000     0.000000
       3.00000      2.00000
IDL> print,reverse(arr,2) ;列反转
```

```
      2.00000      3.00000
      0.000000     1.00000
```

3.2.5.2.5 数组转置 Transpose()

Transpose ()函数可以对数组进行反转。

调用格式：result = Transpose (数组, [P])

关键字：P 是调整后的维数数组列表，如不设置则完全反转。

```
IDL> arr = indgen(2,3,4)
IDL> help,arr
ARR             INT       = Array[2, 3, 4]
IDL> help,transpose(arr)
<Expression>    INT       = Array[4, 3, 2]
IDL> help,transpose(arr,[0,1,2])
<Expression>    INT       = Array[2, 3, 4]
IDL> help,transpose(arr,[0,2,1])
<Expression>    INT       = Array[2, 4, 3]
IDL> help,transpose(arr,[2,1,0])
<Expression>    INT       = Array[4, 3, 2]
```

3.2.5.2.6 数组旋转

1. Rotate()函数

Rotate()函数可以以 90° 整倍数的角度对数组进行旋转操作。

调用格式：result = Rotate (数组,direction)

Direction：该关键字为整数，范围 0-7，分别代表的图像旋转方式见下表(与帮助文档不一样，进行了修改)。

Direction	是否转置	顺时针旋转角度	旋转后坐标 X1	旋转后坐标 Y1
0	否	0	X_0	Y_0
1	否	90°	Y_0	$-X_0$
2	否	180°	$-X_0$	$-Y_0$
3	否	270°	$-Y_0$	X_0
4	是	0	Y_0	X_0
5	是	90°	$-X_0$	Y_0
6	是	180°	$-Y_0$	$-X_0$
7	是	270°	X_0	$-Y_0$

```
IDL> arr = indgen(2,3)
IDL> print,arr
      0       1
      2       3
```

```
        4       5
IDL> print,rotate(arr,0)
        0       1
        2       3
        4       5
IDL> print,rotate(arr,1)
        4       2       0
        5       3       1
IDL> print,rotate(arr,2)
        5       4
        3       2
        1       0
IDL> print,arr
        0       1
        2       3
        4       5
IDL> print,rotate(arr,4)
        0       2       4
        1       3       5
IDL> print,rotate(arr,5)
        1       0
        3       2
        5       4
```

2. Rot()函数

Rot ()函数可以以任意的角度对数组进行旋转操作并能进行放大、缩小操作。

调用格式：result = Rot (数组, Angle, [Mag, X0, Y0],[关键字])

Angle：该关键字数组旋转地角度，单位为度(°)。

Mag：为放大的倍数；

X0：为旋转中心的 X 坐标，默认为列中心；

Y0：为旋转中心的 Y 坐标，默认为行中心；

PIVOT：该关键字控制旋转后(X0,Y0)点是否仍然在原图像中的位置，不设置则(X0,Y0)点在图像的中心位置。

```
IDL> a = bytscl(dist(256))
IDL> tv,a
IDL> B = ROT(A, 33, 1.5, /INTERP)
IDL> tv,b
```

3.2.5.2.7 数组平移 Shift()

Shift()函数是按指定的平移量 S1、…、Sn，分别对数组的第 1、…、n 维进行平移后生成新数组。其中 Si 为正表示向前平移；Si 为负表示向后平移。

例：对数组 a [4,3]，输出分别按列和行平移 2 和 1 后的数组。

```
IDL> a = INDGEN(4 , 3)                ；生成数组 a [4, 3]
IDL> PRINT, a , SHIFT(a , 2 , 1)      ；输出结果为：
  0     1     2     3                   10    11     8     9
  4     5     6     7                    2     3     0     1
  8     9    10    11                    6     7     4     5
```

3.2.5.2.8 数组排序 Sort()

Sort()函数返回排序后的数组索引。

调用格式：Result = Sort(数组[, /L64])

```
IDL> array = [5, 2, 1,3,4]
IDL> print,array;sort(array)
          5          2          1          3          4
IDL> print,sort(array)
          2          1          3          4          0
IDL> print,array[sort(array)]
          1          2          3          4          5
```

3.2.5.2.9 数组唯一调整 Uniq()

Uniq ()函数返回数组中唯一元素的索引，该函数只能发现**相邻的**相同的值，所以使用前须对数组进行排序。

调用格式为 Result = UNIQ(数组 [, Index])

```
IDL> array = [1, 2, 1,3,3]
IDL> print,uniq(array);返回的是唯一值的索引
          0          1          2          4
IDL> print,array[uniq(array)]
          1          2          1          3
IDL> print,uniq(array,sort(array))
     0          1          3
```

3.2.5.2.10 数组相同 Array_Equal()

Array_Equal()函数进行数组相同的判断。

调用格式为 Result = Array_Equal (数组,/关键字)

result：返回 0 或 1。

NO_TYPECONV：添加该关键字则判断是否数据类型相同。

```
IDL> arr = [0,0,0]
IDL> help,arr
ARR                     INT           = Array[3]
IDL> print,array_equal(arr,0b)
    1
IDL> print,array_equal(arr,0b,/no_typeconv)
    0
IDL> print,array_equal(arr,0,/no_typeconv)
    1
```

3.2.5.2.11 个数计算 N_Elements()

N_Elements ()函数返回数组元素的个数。

调用格式为 Result =N_Elements(数组)

```
IDL> I = INTARR(4, 5, 3, 6)
IDL> PRINT, N_ELEMENTS(I)
           360
```

3.2.5.2.12 最大值计算 Max()

Max()函数返回满足数组元素的最大值。

调用格式为 Result =Max(数组 [,关键字],min =变量 1)。

result：返回的是数组的最大值；

变量 1：返回的是数组的最小值；

```
IDL> arr = FINDGEN(2,3,2)
IDL> print,arr
      0.000000      1.00000
       2.00000      3.00000
       4.00000      5.00000

       6.00000      7.00000
       8.00000      9.00000
       10.0000      11.0000

IDL> print,max(arr,min = mValue)
       11.0000
IDL> print,mValue
      0.000000
```

3.2.5.2.13 最小值计算 Min()

Min()跟 Max()函数极其类似，返回满足数组元素的最大值。

调用格式为 Result =Min(数组 [,关键字],max =变量 1)。

result：返回的是数组的最大值；

变量 1：返回的是数组的最小值；

```
IDL> arr = FINDGEN(2,3,2)
IDL> print,min(arr,max = mValue)
       0.000000
IDL> print,mValue
 11.0000
```

3.2.5.2.14 和计算 Total()

Total()函数返回满足数组元素和，并可以通过关键字进行求和选项的控制。

调用格式为 Result =Total(数组 ,Dimension,[,关键字])。

result：返回的是数组的元素求和结果；

dimension：数组求和元素的行列控制；

CUMULATIVE：设置该关键字后，返回结果与原数组一致，第 i 个元素的值为 0-i 值的和；

double:返回双精度类型结果；

integer：返回整型数据结果；

PRESERVE_TYPE：返回结果类型与原数组类型一致，设置该关键字的时候/double 等关键字无效。

```
IDL> arr = FINDGEN(2,3)
IDL> print,arr
       0.000000       1.00000
        2.00000       3.00000
        4.00000       5.00000
IDL> print,total(arr)`
        15.0000
IDL> print,total(arr,/CUMULATIVE)
       0.000000       1.00000
        3.00000       6.00000
        10.0000       15.0000
IDL> print,total(arr,1)
        1.00000       5.00000       9.00000
IDL> print,total(arr,2)
```

```
      6.00000      9.00000
```

3.2.5.2.15 乘积计算 Product()

Product()函数计算满足数组元素乘积，并可以通过关键字进行求乘积选项的控制。

调用格式为 Result =Product(数组 ,Dimension,[,关键字])。

关键字控制与 total 函数的关键字基本一致。

```
IDL> arr = FINDGEN(2,3)+1
IDL> print,arr
      1.00000      2.00000
      3.00000      4.00000
      5.00000      6.00000
IDL> print,product(arr)
       720.00000
IDL> print,product(arr,/CUMULATIVE)
      1.0000000      2.0000000
      6.0000000      24.000000
      120.00000      720.00000
IDL> print,product(arr,1)
      2.0000000      12.000000      30.000000
IDL> print,product(arr,2)
      15.000000      48.000000
```

3.2.5.2.16 阶乘计算 Factorial()

Factorial ()函数计算满足数组元素阶乘。

调用格式为 Result =Factorial (数组, [,关键字])。

关键字-STIRLING:返回结果为斯特令近似，公式参考 $N! = \sqrt{2\pi N}\left[\frac{N}{e}\right]^N$ 。

```
DL> arr = indgen(2,3)
IDL> print,arr
      0      1
      2      3
      4      5
IDL> print,Factorial(arr)
1.0000000      1.0000000      2.0000000      6.0000000      24.000000      120.00000
```

3.2.5.2.17 平均值计算 Mean()

Mean ()函数计算数组元素的平均值。

调用格式为 Result =Mean (数组, [,关键字])。

```
IDL> x = [65, 63, 67, 64]
IDL> print,Mean(x)
   64.7500
```

3.2.5.2.18 方差计算 Variance()

Variance ()函数计算数组元素的方差。

调用格式为 Result =Variance (数组, [,关键字])。

```
IDL> arr = [1, 1, 1, 2, 5]
IDL> print,Variance(arr)
   3.00000
```

3.2.5.2.19 标准差计算 Stddev()

Stddev ()函数计算数组元素的方差。

调用格式为 Result =Variance (数组, [,关键字])。

```
IDL> arr = [1, 1, 1, 2, 5]
IDL> print,Stddev(arr)
   1.73205
```

3.2.5.2.20 平均值、方差、倾斜度及频率曲线峰态 Moment ()

Moment ()函数计算数组元素的平均值、方差、倾斜度及频率曲线峰态。

调用格式为 Result = Moment (数组, [,关键字])。

```
IDL> arr = [1, 1, 1, 2, 5]
IDL> print,moment(arr)
   2.00000      3.00000      0.923760     -1.13333
```

3.2.5.3 矩阵运算

矩阵是线性代数中的一个概念，其实矩阵也是数组，只是矩阵的元素只能是数，而数组的元素可以是字符等其他类型，所以可以称矩阵为数组的子集。

3.2.5.3.1 矩阵间的运算

遵循以下两个法则：

A#B 表示 A 的列乘以 B 的行：

$A_{M\times N}$ # $B_{L\times M}$

$A_{M\times N}$ =A[0:M-1，0:N-1]， $B_{L\times M}$ =[0:L-1，0:M-1]

```
IDL> A = [ [0, 1, 2], [3, 4, 5] ]
IDL> B = [ [0, 1],[2, 3], [4, 5] ]
IDL> print,a
       0       1       2
       3       4       5
IDL> print,b
       0       1
       2       3
       4       5
IDL> print,A#B
       3       4       5
       9      14      19
      15      24      33
```

A##B 表示 A 的行乘以 B 的列：

$A_{M\times N}$ ## $B_{N\times L}$

$A_{M\times N}$ =A[0:M-1，0:N-1]， $B_{N\times L}$ =[0:N-1，0:L-1]

```
IDL> print,A##B
      10      13
      28      40
```

3.2.5.3.2 矩阵相关函数

1. 矩阵求逆 INVERT()
2. 行列数求值 DETERM()
3. 矩阵乘积 MATRIX_POWER()
4. …

3.3 字符及字符串

3.3.1 创建字符串和字符串数组

字符串和字符串数组的创建可以通过直接赋值或函数来实现。

```
IDL> ss = 'abcdef'
IDL> print,s
abcdef
IDL> xs = "abcdef"
IDL> print,xs
abcdef
```

只有需要字符串中包括“”或‘’时才需要都使用，如：

```
IDL> xx = "'b'"
IDL> print,xx
'b'
IDL> ms = strarr(6)
IDL> help,ms
MS              STRING    = Array[6]
```

3.3.2 特殊字符

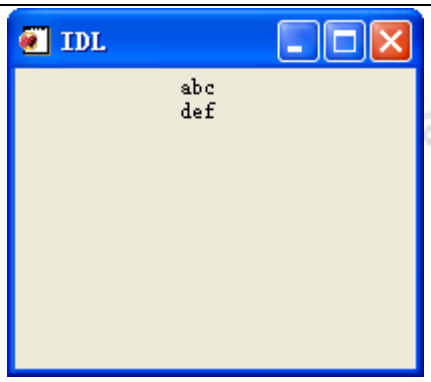
ASCII 字符中有一些不能打印的字符，这些字符的 Byte 值是小于 32 或大于 126 的。

表 3-5 特殊字符对照表

ASCII 字符	数据值
Bell	7B
后退	8B
水平 TAB	9B
换行	10B
竖直 TAB	11B
走纸	12B
回车	13B
ESC 取消	27B

如

```
pro demo_label;
        wtlb = WIDGET_BASE(/Column,xSize = 200,ySize = 150)
        label = WIDGET_LABEL(wTlb,value ='abc'+STRING(13B)+'def',ySize = 40)
        WIDGET_CONTROL, wTlb,/Realize
END
```



3.3.3 部分字符操作函数

字符操作在数据读取、命令行输入，以及各种输入窗口中非常重要。在 IDL 中对字符的操作主要是通过字符操作函数来进行的。

表 3-6 字符串操作函数

函数名	函数描述
STRCMP(str1,str2,N,/FOLD_CASE)	对两个字符串进行比较,如果 N 存在只对前 N 个进行比较, /FOLD_CASE 表示模糊比较
STRCOMPRESS(str1)	删除字符串 str1 中的空格
STREGEX()	正则表达式
STRJOIN()	字符串相连接；
STRLEN（）	返回字符串的长度
STRLOWCASE（）	将所有的大写字母改写成小写字母
STRMATCH(Str1,str2)	字符串 Str1 中是否存在 Str2，可以使用通配符；
STRMID(Str1,po1,Len,REVERSE_OFFSET)	从字符串 po1 开始取出 Len 个字符。字符串的第一个字符的位置为 0
STRPOS(Exp_Str1,Sea_Str2,Pos,REVERSE_OFFSET,/REVERSE_SEARCH])	从一个字符串中查找与另外一个字符串完全匹配的起始点所在的位置 Pos 查找点的起始位置，默认值为 0，1（如果指定/REVERSE_SEARCH] ）指定时，则表示从开始的 Pos 起，或者从末尾开始的 Pos 其（如果指定 REVERSE_OFFSET ）
STRPUT,Des_Var_str,Sou_str,Pos	将 Sou_str 字符串插入到变量 Des_Var _str 之中 POS 插入点的默认值为 0Sou_str 不为字符串，则按默认格式自动转化从 POS 处开始插入 Des_Var _str，如果插入值的位置超过了 Des_Var 的最大长度，则自然截断
STRSPLIT(Str1)	根据特定的要求拆分字符串 str1；
STRTRIM(str,Flag)	移去字符串中的空格 Flag：0(移去左边空格)，1(移去右边空格)，2（移去两边的空格）
STRUPCASE（）	将所有的小写字母改写成大写字母

✧ 使用举例：

➢ StrTrim

```
IDL> var = 4l
IDL> str = string(var)
IDL> help,str
STR             STRING    = '           4'
IDL> help,strtrim(str,1)
<Expression>    STRING    = '4'
```

；

- StrMid

数据 Modis 的数据 AMOD0320040707140331.hdf，A 表示上午星 TERRA，MOD 表示传感器为 MODIS，03 表示 GEOLOCATION 数据，20040707 表示数据采集日期为 2004 年 7 月 7 日，140331 表示该轨数据是在国际标准时间 14 时 03 分 31 秒入境的。

如果需要对文件提取采集时间及 20040707，那么可以这么来做

```
IDL> file = 'AMOD0320040707140331.hdf'
IDL> print,strmid(file,6,8)
20040707
```

;根据文件名计算文件名的扩展名

```
fName = FILE_BASENAME('c:\temp\data.hdr')
pPos =STRPOS(fName,'.',/REVERSE_SEARCH)
IF pPos NE -1 THEN BEGIN          ;
   IF   STRUPCASE(STRMID(fName,pPos[0]+1,3)) EQ 'HDR' THEN CONTINUE
ENDIF
```

3.4 结构体

3.4.1 概述

结构是标量、数组或结构的集合，也就是一种复合变量，IDL 下面的程序很多都是靠结构体进行参数的传递的。结构体可以分为两类：命名结构体和匿名结构体，结构体可以继承。

3.4.2 创建结构体

3.4.2.1 创建命名结构体

```
IDL> a = {one,v1:0,v2:'123'}
IDL> B={two,Inherits one,v3:intarr(3)}
IDL> help,b,/str
** Structure TWO, 3 tags, length=24, data length=20:
   V1              INT                0
   V2              STRING      ''
   V3              INT         Array[3]
```

等价于

```
IDL> B={two,v1: 0, v2:’123’,v3:intarr(3)}
```

IDL 还提供了结构数组，结构数组的创建格式为：

```
c=replicate(A,10)  或 c=replicate(one，10)
```

3.4.2.2 创建匿名结构体

IDL 运行期间需要改变结构体内；

```
IDL> person={name:'Jack',Country:'USA',work:'ESRI'}
IDL> help,person,/str
** Structure <18974a0>, 3 tags, length=36, data length=36, refs=1:
   NAME            STRING    'Jack'
   COUNTRY         STRING    'USA'
   WORK            STRING    'ESRI'
```

3.4.3 访问结构体

通过结构体变量名.域名及变量名.(index)的方式来访问，如：

```
IDL> print,person.name
Jack
IDL> print,person.(1)
USA
IDL> print,person.(0)
Jack
```

创建了结构体后（有名的或是匿名的），其域的类型和大小就将不能再改变了，但可以进行类型转换。

```
IDL> person.name = strarr(2)
% Expression must be a scalar in this context: <STRING      Array[2]>.
% Execution halted at: $MAIN$
IDL> print,person.name = 2
IDL> help,person.name
<Expression>    STRING    = '          2'
```

3.4.4 结构体操作相关函数

表 3-7 结构体操作函数

函数名	用途
CREATE_STRUCT（）	根据给定的名字和值创建结构体，并能连接结构体；
Help,***,/Struct	返回输入结构体的相关信息；
N_TAGS()	返回结构体中的成员个数；
TAG_NAMES( )	返回结构体成员的名字；

3.5 指针

3.5.1 概述

从 IDL5.0 版本开始，IDL 下有了指针数据类型。创建指针时，其数据存储于一个堆变量中。堆变量在程序运行期间是全局变量，而且只有通过指针名才能访问。IDL 指针是高层次的语言结构，与其他程序语言如 C，C++和 FORTRAN 的指针等有很大不同，它指向的堆变量是可以动态的分配内存的全局变量，不像 C 中的指针指向真正的内存地址。

3.5.2 创建和访问指针

指针的创建是由指针创建函数 Ptr_New()来创建的，指针的访问则是由* 加指针变量名来实现的。

```
IDL> C = PTR_NEW(2.0d)
IDL> help,c
C               POINTER     = <PtrHeapVar97
IDL> print,*c
      2.0000000
```

指针的赋值：

```
IDL> A = 1.0
IDL> B = A
IDL> C = PTR_NEW(2.0d)
IDL> D = C
IDL> help,a,b,c,d
A               FLOAT       =       1.00000
B               FLOAT       =       1.00000
C               POINTER     = <PtrHeapVar98>
D               POINTER     = <PtrHeapVar98>
IDL> *d = 5
IDL> print,*c,*d
      5       5
```

指针的这种特性导致了它的操作的灵活性。

3.5.3 空指针

IDL 中的无效指针是不能被反引用的，因为它们不指向任何堆变量，因此也不含任何数据。若需创建起初不含任何数据但后来又需要包含数据的指针，这就要创建一个分配堆变量的指针，方法是在 PTR_NEW 中使用 ALLOCATE_HEAP 关键字。

```
IDL> pempty=ptr_new(/allocate_heap)
IDL> help,pempty
PEMPTY          POINTER     = <PtrHeapVar6>
IDL> help,* pempty
<PtrHeapVar6>     UNDEFINED = <Undefined>
IDL> print,ptr_valid(pempty)
1
```

使用 ALLOCATE_HEAP 关键字创建了一个包含未定义变量的堆变量，且 pempty 是个有效指针。因此 pempty 可以反引用堆变量的值，而且可以通过 pempty 为该堆变量赋值。

3.5.4 指针的内存控制

3.5.4.1 内存分配

默认状况下，指针创建函数 PTR_NEW 将其参数复制到堆变量中，在局部内存中留一份拷贝。若为 PTR_NEW 设置 NO_COPY 关键字，则不会在局部内存中保留拷贝。

```
IDL> data = indgen(200,200)
IDL> pData = Ptr_New(data)
IDL> help,data,pData
DATA            INT         = Array[200, 200]
PDATA           POINTER     = <PtrHeapVar102>
IDL> help,data,pData,*pData
DATA            INT         = Array[200, 200]
PDATA           POINTER     = <PtrHeapVar102>
<PtrHeapVar102> INT         = Array[200, 200]
IDL> pData = Ptr_New(data,/No_Copy)
IDL> help,data,pData,*pData
DATA            UNDEFINED = <Undefined>
PDATA           POINTER     = <PtrHeapVar103>
<PtrHeapVar103> INT         = Array[200, 200]
```

用 IDL> pData = Ptr_New(data,/No_Copy)相当于是应用了

```
pDtata= Ptr_New(Temporary(data))
```

3.5.4.2 内存释放

指针清除-内存释放只能通过 Ptr_Free 的方法，Ptr_Free 可以释放指针占用的内存。指针如果不靠 Ptr_Free 来清除的话则只能到 IDL 进程结束指针才能释放。

```
IDL> Ptr_Free， pData
```

```
IDL> print,Ptr_Valid(pData)
 0
```

HEAP_GC 对堆变量施行“捡垃圾”处理，它会依次检测当前所有的堆变量并清除所有已无法被引用的堆变量。

3.5.5 指针操作函数

表 3-8 指针操作函数

名称	作用
Ptr_New()	创建新指针；
Ptr_Free	释放指针；
Ptr_Valid	检测指针是否存在；
PtrArr	建立指针数组；

3.6 运算符

3.6.1 数学运算符

IDL 中的数学运算符有加（+）、增运算（++）、减（-）、减运算（--）、乘（*）、除（/）、幂（^）、取余（mod）、取小（<）和取大（>）。

3.6.1.1 增运算和减运算（++、- -）

这两个运算跟 C 里面的是一样的；

```
IDL> var = 20
IDL> print,var++
          20
IDL> print,var
     21
```

3.6.1.2 幂运算（^）

```
IDL> var = 2
IDL> print,var^4
          16
IDL> print,var^5
32
```

3.6.1.3 取余运算（mod）

```
IDL> var = 32
IDL> print,var mod 4
           0
IDL> print,var mod 5
```

```
      2
```

3.6.1.4 取大和取小运算（> 、<）

```
IDL> print,var > 39
      39
IDL> print,var < 39
32
```

3.6.2 逻辑运算符

IDL 中逻辑运算有几个：&&, ||, 和 ~。

3.6.2.1 &&

逻辑的与运算，如果表达式为真或非零则返回 1 否则返回 0。

```
IDL> PRINT, 5 && 2
   1
IDL> PRINT, 5 && 0
   0
IDL> PRINT, "sd" && "d"
   1
IDL> PRINT, "sd" && " "
   1
IDL> PRINT, "sd" && ""
   0
```

3.6.2.2 ||

逻辑的或运算

```
IDL> PRINT, 5 || 2
   1
IDL> PRINT, 5 || 0
   1
IDL> PRINT, 0 || 0
   0
IDL> IF ((5 GT 3) || (4 GT 5))   THEN PRINT, 'True'
True
```

3.6.2.3 ~

逻辑的非运算

```
IDL> print,~3
   0
IDL> print,~0
1
```

3.6.3 位运算符

在计算机中，信息存储均是二进制形式。数字是以补码形式保存的，正数的补码和原码相同，负数的补码是原码取反后加 1。位运算符是基于位进行的，IDL 中包含四个位运算符：AND, NOT, OR, and XOR。

3.6.3.1 位加运算符（AND）

```
IDL> print,5 AND 6
         4
```

原理如下：

5 → 0101

6 → 0110

————————————

4 → 0100

3.6.3.2 位取反运算符（NOT）

```
IDL> print,NOT 1
     -2
```

结果为-2，原理如下：

在计算机里面，二进制编码的最高位表示符号位。如果用四位二进制位来表示，1 为“0001”，位取反后为“1110”，因最高位为符号位，即为负数。负数在计算机中是用补码表示的，补码“110”对应的值是按照“取反加 1”即“010”，十进制中的 2，所以 1 取反后的值为-2。

3.6.3.3 位或运算符（OR）

```
IDL> print,5 or 10
      15
```

5 → 0101

10 →1010

————————————

15 →1111

3.6.3.4 位与或运算符（XOR）

```
IDL> print,3 XOR 5
```

```
       6
```

3 → 0011

5 → 0110

————————————

6 → 0110

3.6.4 关系运算符

IDL 中的关系运算符包括 EQ、NE、GE、GT、LE、LT 几个，关系运算符返回的是真“True”或假“False”。

3.6.4.1 EQ（Equal to）

等运算符，如果运算符两边相同则返回真，否则返回假；

```
IDL> print,2 EQ 2.0
      1
```

3.6.4.2 NE（Not Equal to）

不等运算符，如果运算符两边不等则返回真，否则返回假；

```
IDL> print,2 NE 2.0
   0
IDL> print,2 NE 1
      1
```

3.6.4.3 GE（Greater than or equal to）

大于等于运算符，如果运算符两边左边大于等于右边则返回真，否则返回假；

```
IDL> print,2 GE 1
   1
IDL> print,2 GE 12
   0
```

3.6.4.4 GT（Greater than）

大于运算符，如果运算符两边左边大于右边则返回真，否则返回假；

```
IDL> print,2 GT 1
   1
IDL> print,2 GT 12
   0
IDL> print,2 GT 2
   0
```

3.6.4.5 LE（Less than or equal to）

小于等于运算符，如果运算符两边左边小于等于右边则返回真，否则返回假；

```
IDL> print,2 LE 1
   0
IDL> print,2 LE 12
   1
IDL>  print,2 LE 2
   1
```

3.6.4.6 LT（Less than）

小于运算符，如果运算符左边大于右边则返回真，否则返回假；

```
IDL> print,2 LT 1
   0
IDL> print,2 LT 12
   1
IDL> print,2 LT 2
   0
```

3.6.4.7 数组关系运算

数组关系运算结果是数组的每个元素对应运算的结果。

```
IDL> a = [2,3,5]
IDL> b = [1,4,5]
IDL> print,a GE b
   1   0   1
IDL> print,a LE b
   0   1   1
```

3.6.5 数组运算符

3.6.5.1 #数组乘

数组乘#则是第一个数组的列乘以第二个数组的行。

```
IDL> arr1 = [[1,2,1],[2,-1,2]]
IDL> print,arr1
      1       2       1
      2      -1       2
IDL> arr2 = [[1,3],[0,1],[1,1]]
IDL> print,arr2
      1       3
      0       1
      1       1
IDL> print,arr1#arr2
```

```
          7          -1           7
          2          -1           2
          3           1           3
```

3.6.5.2 ##矩阵乘(行乘列)

```
IDL> arr1 = [[1,2,1],[2,-1,2]]
IDL> print,arr1
      1       2       1
      2      -1       2
IDL> arr2 = [[1,3],[0,1],[1,1]]
IDL> print,arr2
      1       3
      0       1
      1       1
IDL> print,arr1#arr2
          7          -1           7
          2          -1           2
          3           1           3
IDL> print,arr1##arr2
          2           6
          4           7
```

3.6.6 其他运算符

3.6.6.1 小括号()

小括号用来对表达式组合或控制一系列表达式的优先级。

```
IDL> print,3+4*2^2/2
     11
IDL> print,3+(4*2)^2/2
     35
```

3.6.6.2 中括号[]

中括号用来数组连接或对数组进行元素调用。

```
IDL> C = [0, 1, 3]
IDL> print,[C,5]
      0       1       3       5
IDL> print,C[2]
      3
```

3.6.6.3 条件表达式?:.

该运算符的用法是：value = expr1 ? expr2 : expr3

如果 expr1 是 true，那么 value 等于 expr2 否则 value = expr3.

```
IDL> A=6 & B=4
IDL> print,(A GT B) ? A : B
      6
```

3.6.6.4 对象方法调用符->

```
    IDL> oWindow->Draw
```

3.6.6.5 指针引用符*

指针调用符为*，调用格式为*指针变量。

```
IDL> point1 = ptr_New(34)
IDL> print,*point1
         34
```

3.6.7 运算符的优先级

优先级	操作符
一级（最高）	( ) (公式表达式)
	[ ] (数组连接)
二级	. (结构体域表达式)
	[ ] (数组元素调用)
	( ) (函数引用)
三级	* (指针调用)
	^ (幂函数)
	++ (增函数)
	-- (减函数)
四级	* (乘)
	# 和 ## (矩阵乘)
	/(除)
	MOD (取余数)
五级	+ (加)
	- (减和取反)
	< (求小)
	> (求大)
	NOT (位取反)

	~ (逻辑取反)
六级	EQ (等于)
	NE (不等于)
	LE (小于等于)
	LT (小于)
	GE (大于等于)
	GT (大于)
七级	AND (位与)
	OR (位或)
	XOR (位异或)
八级	&& (逻辑与)
	|| (逻辑或)
九级	?: (条件表达式)

3.7 其他符号

3.7.1 $续行符

续行符主要应用在一条语句过长或参数过多时进行格式化显示。例如：

```
PRO using_continuationline
  s = 'abc'
  print,'esri'+s
  print,'esri'+ $
    s
END
```

3.7.2 &同行符

使用了同行符相当于两行代码写在一行之中。

```
IDL> a = 3 & b =5
;相当于
IDL> a = 3
IDL> b =5
```

3.7.3 ;注释符

IDL 中的注释符为“;”，此符号之后的该行所有代码都将看做是注释，不参与编译运行。

4 IDL 编程基础

4.1 程序控制

结构化程序设计是 E.W.Dijikstra 在 1965 年提出的。它的主要观点是采用自顶向下、逐步求精的程序设计方法；使用三种基本控制结构构造程序，任何程序都可由顺序、选择、重复三种基本控制结构构造。编写 IDL 程序也需要对 IDL 中的相关控制语句进行简单介绍。

4.1.1 循环语句

4.1.1.1 For 语句

for 语句每次循环执行一个语句或语句块

for i=v1，v2 do 语句

for i=v1，v2， inc do 语句

for i=v1，v2，inc do begin

语句

endfor

默认条件下增量为 1，也可以自定义增量。

```
FOR i=0,5 DO BEGIN
  PRINT,i
ENDFOR
;
FOR i=0,10,2 DO BEGIN
  print,i
ENDFOR
```

4.1.1.2 While 语句

While…Do

当特定的条件为真，while 语句执行单个语句或语句块

while 条件 do 语句

while 条件 do begin

语句

endwhile

```
i=0
```

```
j=36
WHILE(j LT 44)DO BEGIN
   PRINT,j
   j++
   i=i+1
ENDWHILE
PRINT,i
```

4.1.1.3 Repeat 语句

Repeat…Until

REPEAT 语句 UNTIL 条件

或者

REPEAT BEGIN

语句

ENDREP UNTIL 条件

```
i = 0
REPEAT BEGIN
   i++
   print,i
ENDREP UNTIL(i GE 5)
print,i
```

4.1.2 条件语句

4.1.2.1 IF 语句

if 条件 then begin

语句

endif

if 条件 then 语句 else 语句

if 条件 then begin

语句

endif else begin

语句

endelse

```
IF((num MOD 2) EQ 0) THEN BEGIN
   tmp = DIALOG_MESSAGE(STRING(num)+' 是偶数！')
ENDIF
```

```
IF((num MOD div) EQ 0) THEN BEGIN
   tmp = DIALOG_MESSAGE(STRING(num)+'能够被'+STRING(div)+'整除！')
ENDIF ELSE BEGIN
   tmp = DIALOG_MESSAGE(STRING(num)+'不能够被'+STRING(div)+'整除！')
ENDELSE
```

4.1.2.2 CASE 语句

case 语句根据一个标量的表达式来选择某个语句或语句块运行。

case　表达式　of

　情况 1：

　情况 2：语句

　情况 3：begin

　　语句

　　end

　else:　语句

endcase

当表达式和其中的某个情况匹配，相应的语句被执行， case 语句结束，如果没有匹配的情况，那么执行 else 下的语句，如果没有 else 语句，将会发生错误，建议在 case 语句中都加上 else。

```
index = 1

CASE index OF
   0: PLOT,SIN(FINDGEN(100)*0.25)
   1: SURFACE,dist(32)
   2: BEGIN
      ERASE
      image=dist(400)
      TVSCL,image
   END
   ELSE: PRINT,index
ENDCASE
```

4.1.2.3 SWITCH 语句

Switch 跟 Case 类似，但与 case 的不同在于它能执行到底。

```
x=2
SWITCH x OF
   1: PRINT,'one'
```

```
    2: PRINT,'two'
    3: PRINT,'three'
    4: PRINT, 'four'
ENDSWITCH
print,'end'
```

4.1.3 跳转语句

4.1.3.1 Break

break 提供了一个从循环中（FOR，WHILE，REPEAT）或 CASE、SWITCH 等状态中快速退出的方法。以 WHILE 循环为例，Break 的应用见下面代码。

```
I = 0
WHILE (1) DO BEGIN
  i = i + 1
  IF (i EQ 5) THEN BREAK
  PRINT,i
ENDWHILE
FOR i=2,10 DO BEGIN
  IF i EQ 5 THEN CONTINUE
  PRINT,i
ENDFOR
```

4.1.3.2 Continue

continue 提供了一个从循环中（FOR, WHILE,和 REPEAT）中进入下一步循环的方法。以 FOR 为例，学习下 continue 的用法。

```
FOR I = 1,10 DO BEGIN
  ; If odd, start next iteration:
  IF (I GT 5) THEN CONTINUE
  PRINT, I
ENDFOR
PRINT,'end'
```

4.1.3.3 Goto

goto 可以使程序跳转到某一特定的标签处，见下面例子

```
GOTO, JUMP1
  PRINT, 'Skip this' ; This statement is skipped
  PRINT, 'Skip this' ; This statement is also skipped
  JUMP1: PRINT, 'Do this'
```

4.2 位置参数及关键字参数

IDL 中的过程和函数接受两种参数：位置参数和关键字参数。

4.2.1 位置参数

位置参数在过程或函数中用来传递变量或表达式，比如下面的过程定义：

```
PRO test_parameters, param1,param2
  HELP,param1,param2
  PRINT,N_PARAMS()
END
```

在调用的时候用 test_parameters, ‘par’,’par2’，对一个特定的过程或函数而言它的位置参数有一定的顺序，位置参数也不一定是必须的，有一些是可选的。位置参数的个数可以用函数 N_Params()来检验。

```
IDL> test_parameters
PARAM1          UNDEFINED = <Undefined>
PARAM2          UNDEFINED = <Undefined>
           0
IDL> test_parameters,'asdf',3
PARAM1          STRING    = 'asdf'
PARAM2          INT       =        3
           2
```

4.2.2 关键字参数

关键字参数是 IDL 中一个可选的变量或表达式。关键字参数传入的可以是一个默认是其他地方定义好的参数或一个 Bool 的标志，输出关键字参数用来返回所需要的值。关键字参数没有顺序，它可以放在过程或函数的任意位置。

检测关键字参数 keyword_set()用于检测布尔关键字，如果自变量定义则返回真。见下面例子：

```
PRO test_keywords, input, keyword1 = keyword1, keyword2 = keyword2,swap = swap
  COMPILE_OPT idl2
  HELP,input
  HELP,keyword1
  HELP,keyword2
  PRINT, KEYWORD_SET(swap)
  IF KEYWORD_SET(swap) THEN BEGIN
    print,'swap'
  ENDIF
END
```

在命令行中输入

```
IDL> test_keywords,'3',keyword1 =1,/swap
INPUT                STRING      = '3'
KEYWORD1             INT         =           1
KEYWORD2             UNDEFINED = <Undefined>
           1
Swap
```

4.2.3 参数继承

IDL 下支持关键字的继承，跟面向对象中类的方法继承有点类似，可看下面两个例子。

```
PRO test_ExtraArgm, a, b, _EXTRA = e
   PLOT, a, b,    _EXTRA = e
END
```

4.2.4 参数传递机理

参数传递包括两种，值传递和地址传递，理解参数传递的机理很重要，先看一个例子：

```
PRO test_argMechanism, arg
   arg = 3*arg
END
;
PRO test_exceute
   s =5
   test_argMechanism,s
   PRINT,s
   PRINT,'------------'
   ;
   DEFSYSV,'!sValue',33
   test_argMechanism,!sValue
   PRINT,!sValue
   PRINT,'------------'

   arr = INDGEN(5)
   test_argMechanism, arr
   PRINT,arr
   PRINT,'------------'
   ;
   arr = INDGEN(5)
   test_argMechanism,arr[0:3]
   PRINT,arr
   PRINT,'------------'
END
IDL> test_exceute
```

```
          15
------------
          33
------------
           0       3       6       9      12
------------
           0       1       2       3       4
------------
```

这里的调用出现不同结果的原因是分别进行了地址传递和值传递，具体应用见表。

表 4-1 地址传递和值传递列表

地址传递	值传递
变量	常数
数组	数组的索引
结构体	结构体成员
	系统变量
	表达式
_EXTRA	_REF_EXTRA

4.2.5 相关函数

表 4-2 参数相关函数列表

函数	作用
N_PARAMS()	返回参数个数（不包括关键字参数）
N_ELEMENTS()	返回变量元素个数
KEYWORD_SET()	是否关键字存在
ARG_PRESENTS()	是否存在传递的引用参数

例子程序如下

```
PRO myplot,x,y,position = position
  ;
  PRINT,'N_PARAMS():',N_PARAMS()
  PRINT,'N_ELEMENTS():',N_ELEMENTS(x)
  PRINT,'KEYWORD_SET():',KEYWORD_SET(y)
  PRINT,'ARG_PRESENT',ARG_PRESENT(y)
END

PRO test_paramFunction
  ;
```

```
    myplot,INDGEN(10)
    myplot,INDGEN(10),SIN(INDGEN(10))
    ;indgen(10),sin(indgen(10))
    myPlot,INDGEN(10),SIN(INDGEN(10)),position = [0,0]
END
```

程序运行结果如下：

```
N_PARAMS():              1
N_ELEMENTS():           10
KEYWORD_SET():          0
ARG_PRESENT      0
N_PARAMS():              2
N_ELEMENTS():           10
KEYWORD_SET():          1
ARG_PRESENT      0
N_PARAMS():              2
N_ELEMENTS():           10
KEYWORD_SET():          1
ARG_PRESENT      0
```

4.3 错误处理

IDL 把错误分成三种类别：输入/输出错误、数学错误和其他错误。

4.3.1 错误处理程序

CATCH、ON_ERROE 和 ON_IOERROR。CATCH 它会告诉 IDL 捕获发生在过程或函数中的错误并允许程序员对其作出处理。过程或函数中出错时，IDL 首先在过程或函数中查找错误的发生位置，这就是一个“CATCH”。若未发现任何“CATCH”，IDL 就会返回至调用过程或函数中查找“CATCH”。IDL 会依调用顺序向上查找直到主程序级别，这是 IDL 会停止查找并打印出调用序列。错误处理流程见图 4-1。

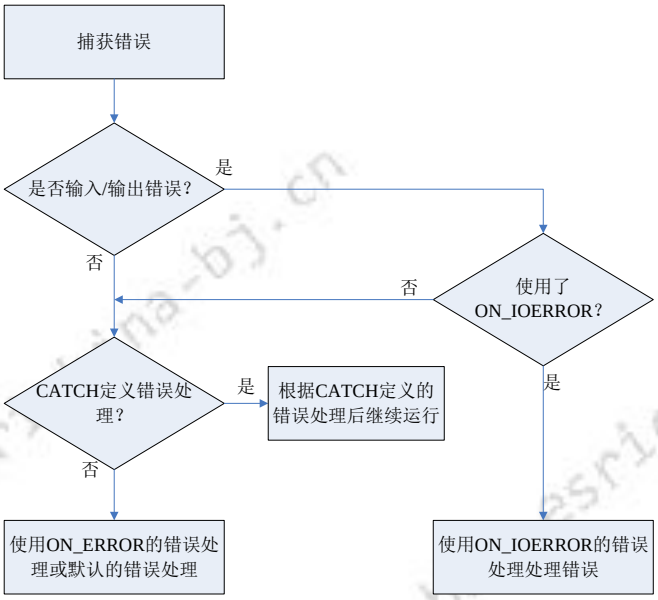


图 4-1 IDL 中的错误处理流程

4.3.2 数学错误

IDL 检测出的数学错误，如除数为零，溢出，试图对负数取对数等运算是依赖于硬件和操作系统，故有些系统比其它的系统有更多的错误。操作系统上执行的 IEEE 浮点标准，当 IDL 编译器检测到系统的浮点数错误时它会用 NaN 和 Infinity 来取代。所有平台上都对会整数除以零进行检测。

```
IDL> PRINT, !VALUES.F_NAN
           NaN
```

使用 FINITE 函数检测数据是否有效，如：

```
IDL> print,FINITE(!VALUES.F_NAN)
    0
IDL> print,FINITE(1)
    1
```

用 IDL 现有的函数进行处理的时候，如有可能出现 NaN 的错误，在使用时可添加 NaN 关键字，见浮点错误例子程序。

```
PRO test_NAN
  ;
  PRINT,!VALUES.F_nan
  NaNValue = 0/0.
  data = FINDGEN(3,4)
  data[*,3] = NaNValue
  ;
  PRINT,data;
  PRINT,MAX(data,/NAN)
  ;
```

```
    PRINT,'max(data)',MAX(data)
END
```

4.4 调用执行方式

4.4.1 EXECUTE()函数

EXECUTE()函数的功能是通过调用字符串中的命令来实现命令执行的功能，可以看下 ENVI 中的波段运算与波普运算功能。如：

```
IDL> com = 'PLOT, [0,1]'
IDL> void = EXECUTE(com)
```

4.4.2 CALL_FUNCTION 函数

call_function 函数通过调用函数字符串来实现函数的执行功能。如：

```
IDL> R = CALL_FUNCTION('SQRT', 4)
IDL> print,r
      2.00000
```

4.4.3 CALL_PROCEDURE

call_procedure 通过调用命令字符串来实现命令的执行功能，如：

```
IDL> CALL_PROCEDURE, 'PLOT', FINDGEN(100), FINDGEN(100), /POLAR
```

4.5 编译规则

COMPILE_OPT 允许你对 idl 编译器的默认编译规则进行稍微修改，详细可参考 IDL 的帮助文档。常添加的关键字有 DEFINT32、STRICTARR 等。

COMPILE_OPT DEFINT32 –把默认 IDL 的整型数据 16 位修改为 32 位；

COMPILE_OPT STRICTARR-默认数组元素的选取用中括号，不用小括号，避免与函数调用的混淆。

常用 COMPILE_OPT IDL2，该语句的功能则相当于同时使用 COMPILE_OPT DEFINT32 和 STRICTARR，即编译规则中添加 IDL 整型数据位 32 位和数组元素的使用必须用中括号。

```
IDL> .reset
IDL> data = indgen(3,4)
IDL> print,data(*)
       0       1       2       3       4       5       6       7       8       9      10      11
IDL> compile_opt strictarr
IDL> print,data(*)
print,data(*)
                ^
% Syntax error.
```

4.6 IDL 高效编程

IDL 程序中有下面几个方法来使得程序高效，也就是程序运行时花费的时间短、占用的内存少。

4.6.1 释放内存

运算过程中变量占用内存的处理可以用函数 DELVAR 及 Temporary()。DELVAR 可以直接删除数组并释放内存，但只能在主程序中使用。Temporary()函数可以应用在变量、数组、指针、结构体或对象上都有效。

```
IDL> a = dist(500,500)
IDL> a = a+1.0
```

对第一行创建浮点型数组，数组 a 占用了 500*500*4=1Million 的 Bytes，第二行代码运行时系统需要开辟出另 1Million Bytes 的字节来做加运算，也就是说这两行代码执行完后系统分配了 2MillionBytes 的字节。

```
IDL> b = temporary(a) + 1.0
IDL> help,a,b
A               UNDEFINED = <Undefined>
B               FLOAT     = Array[500, 500]
```

所以在程序中，a = a+1.0 最好用 a = temporary(a) + 1.0 的方式。

4.6.2 优化运行时间

在 IDL 语言下，应该避免循环的使用，下面的 time_test 例子对 for 循环、while 循环和系统函数做了一个对比时间测试。

```
PRO Test_time
  ;
  a =   dist(2000,2000)
  sum = 0.
  sum1 = 0.
  start = SYSTIME(1)
  FOR i=0L,N_ELEMENTS(a)-1L DO IF(a[i] GT 100.0)THEN sum = sum +a[i]
  PRINT,'for time:',SYSTIME(1)-start

  i=0L
  start = SYSTIME(1)

  WHILE i LT N_ELEMENTS(a)-1L DO BEGIN
    IF(a[i] GT 100.0)THEN sum = sum +a[i]
    i++
  ENDWHILE
```

```
    PRINT,'while time:',SYSTIME(1)-start
    ;
    start = SYSTIME(1)
    sum = TOTAL(a * (a GT 100.0))
    PRINT,'function time:',SYSTIME(1)-start

END
```

for time: 0.71900010
while time: 1.1099999
function time: 0.030999899

bbs.esrichina-bj.cn

5 输入与输出

IDL 中输入输出的基本原则是：“只要有数据，就可以将其读进 IDL”，IDL 对数据的格式没有要求，任何数据格式都能自由的解析，这使得 IDL 成为目前功能最强、最灵活的科学可视化分析语言。

5.1 标准输入输出

5.1.1 输入输出函数

IDL 中读写 ASCII 码文本是很重要的一部分。标准输入是键盘输入，标准输出指的是屏幕输出显示。表 5-1 中列出了标准输入输出的几个函数。

表 5-1 标准输入输出函数

函数	作用
Print	标准输出写出格式化数据
Read	标准输入读入格式化数据
Reads	从字符串中读取格式化数据
String()	格式化输出的字符串数据

5.1.2 格式化输出

数据的格式化输出是按照设计的格式进行输出。IDL 支持“%”和“\”。

标准的输入输出时可以进行格式的自由控制，格式化控制的格式为：[n]FC[+][-][width]，各个字段代表的含义见表 5-2

表 5-2 格式化控制含义

n	代表格式控制符的有效次数，默认为 1 次。
FC	格式控制符，详细说明见表 5-3。
+	在输出的整数前面加“+”符号，默认正数前不输出，负数前有“-”。
-	控制字符串和数字的输出为左对齐，默认输出是右对齐。
width	输出字符的宽度，对特定的宽度部分可补 0 或空格。

表 5-3 格式化控制代码含义

格式代码	格式含义
A	字符
:	若没有有效的变量，则终止输入输出
$	使下一个输出数据输出到该输出数据的所在行的后面（仅用于输
I	十进制整数
O	八进制整数
Z	十六进制整数

F	输出单精度浮点数
D	输出双精度浮点数
E	以指数形式（科学格式）输出浮点数
[n]G[w.d] or	根据数据大小自动选择科学格式或者 F、D 格式输出
T	绝对位置输出变量的值
TL	从当前位置向左移动输出变量的值
TR	从当前位置向右移动输出变量的值
/	换行输出

如下面的例子：

```
PRO Test_format
  COMPILE_OPT idl2
  ;
  PRINT, FORMAT = '(A6)', '123456789'
  PRINT, FORMAT = '(I2,$)', INDGEN(6)
  ;
  PRINT, FORMAT = '("First", 20X, "Last", T10, "Middle")'
  PRINT, FORMAT = '("First", 20X, "Last", T4, "Middle")'
  ;
  PRINT, FORMAT = '("First", 5X, "Last", TL2, "Middle")'
  ;
  PRINT, SYSTIME(/JULIAN)
  PRINT, FORMAT='(C())', SYSTIME(/JULIAN)
  PRINT, FORMAT='(C(CMOI,"/",CDI))',SYSTIME(/JULIAN)
  PRINT, FORMAT='(C(CHI2.2,":",CMI2.2,":",CSF05.2))',SYSTIME(/JULIAN)
  ;
  PRINT, FORMAT='(%"I have %d monkeys, %s,", " and ", I0, " parrots.")', 10, 'Scott', 5
  PRINT, format='(%"I have \xb1%d monkeys \\")', 5
  ;
  print,'0F'X ; 十六进制转十进制
  print,"17    ; 八进制转十进制
  print, 119, format='(B)';十进制转二进制的
  print, 119, format='(O)';十进制转到八进制
  print, 119, format='(Z)';十进制转到十六进制
END
```

5.2 文件读写

5.2.1 打开文件

IDL 从磁盘上的文件读写数据，必须首先把一个逻辑设备号连接到一个特定的文件，然后进行文件操作如打开、关闭和读取等。IDL 中的逻辑设备号范围是

-2—128，其中 1--99 是可以用户任意指定的，100-128 是根据 get_lun 和 free_lun 来管理的，其他是特殊函数专用的。

0 是常规标准输入流，一般是键盘。也就是说下面命令是一致的：

```
IDL> read,x
IDL> read,0,x
```

-1 是常规标准输出流，一般是显示屏幕，下面命令的功能是一致的：

```
IDL> PRINTF, -1, X
IDL> PRINTF, -1, X
```

-2 是错误流，一般是显示屏幕。

文件操作相关的函数见表 5-4。

表 5-4 DL 中文件操作的函数列表

函数名字	作用
OpenR	以只读方式打开已存在文件；
OpenW	创建一个可以读写的新文件；
OpenU	以更新模式打开已存在文件；
File_Search()	对文件名进行特定的查找；
Dialog_Pickfile()	对话框方式选择文件；
Fstat()	返回一个已打开文件的信息；
EOF()	检测是否到文件末；
CLOSE	关闭一个文件；
Free_Lun	释放一个逻辑设备号并关闭文件；

5.2.2 读写 ASCII 码文件

IDL 在读写该文件分为有两种格式化文件：自由文件格式和确定的文件格式。

自由文件格式是用逗号或空白（tab 键和空格键）分开文件中的每个元素的 ASCII 文件。确定的格式文件是用格式说明按照给定的规范进行编排的。二者相比后者比前者更正规一些。

5.2.2.1 读写自由文件格式

IDL 下用 ReadF 从文件中读入自由格式数据，Printf 写入自由格式数据到文件中。IDL 中读写自由格式文件遵循下面的几个规则：

1) 如果读入到字符串变量中，那么，在当前行剩下的所有字符都将读入该

变量中。

2) 输入数据必须用逗号或空白分隔（空格键或 tab 键）。

3) 输入通过数字变量完成。数组和结构都可作为数字变量的集合。

4) 如果当前读入行是空的，并且还有变量要求输入，则读取另一行。

5) 如果当前读入行不是空的，但是没有变量要求输入，则忽略此行剩下的数。

6) 尽量将数据转换为变量所希望的数据类型。

7) 复数数据必须有实数和虚数两部分，用逗号分隔，并用括号括起来。

5.2.2.2 读写确定的文件格式

读写确定文件格式可同样用 ReadF 和 PrintF 命令，它们刚才已用于自由格式文件，但现在文件格式已由 Format 关键字明确声明。（在读写标准输入和输出时，也可将 Format 关键字用于 Read 和 Print 命令）。

5.2.2.2.1 通用的格式说明符

通用格式说明符 Format 关键字的语法在 5.1.1 章节中有详细的描述，这里的用法跟该章节一致。

5.2.2.2.2 逗号分隔的确定格式

有时数据文件必须用确定格式书写，以方便它们被其它软件读取。用逗号分隔的数据文件就是这类文件的典型代表。

Format=’(F10.3, A1, 2x, F10.3, A1, 2x, F10.3)’

5.2.2.2.3 字符串中读取格式数据

ReadS 是一个有用的 IDL 命令，可以从字符串变量而不是从文件中为自由格式或确定格式读取数据。ReadS 运用了和命令 Read 和 ReadF 相同的读取格式数据规则。也就是说，使用 ReadS 就象从数据文件中读取一样，所不同的是所读的对象是一个字符串变量。

当大量信息需从文件头部读取时，此命令特别有用。例如，假设 ASCII 数据文件的第一行说明了数据文件的行数和列数，此文件头可以从文件中读取，并且可创建一个大小正确的数组来读取数据。

```
IDL> firstLine=’ 10 24500 12 June 1996’
IDL> columns=0
```

```
IDL> rows=0
IDL> date=''
IDL> reads,firstline,columns,rows,date
10 24500 12 June 1996
IDL> print,columns;,rows,date
      10
IDL> print,rows;,date
   24500
IDL> print,date
 12 June 1996
```

5.2.2.3 向导式读取函数

IDL 中提供了 ASCII_TEMPLATE，QUERY_ASCII 和 READ_ASCII 函数。

5.2.2.4 综合应用举例

针对上面介绍的三种读取方式，对同一个数据采用了不同的读取方式进行了读取，附代码。

```
asciiFile =currentPath+'\data\ascii.txt'
  ;依次读取
  OPENR,lun,asciifile,/get_lun

  IF lun EQ -1 THEN RETURN;
  tmp = ''
  WHILE(~EOF(lun)) DO BEGIN
    READF,lun,tmp
  ENDWHILE

  FREE_LUN,lun
  ;分块读取
  tmp = STRARR(3)
  data = FLTARR(2,4)
  fdata = INTARR(20)
  OPENR,lun,asciifile,/get_lun

  READF,lun,tmp
  READF,lun,data

  FREE_LUN,lun

  ;自定义格式读取
  tmp   =''
  row =0
  columns =0
```

```
type =0
OPENR,lun,asciifile,/get_lun
;
READF,lun,tmp
READS,tmp,row,columns,type
data =MAKE_ARRAY(row,columns,type = type)
READF,lun,tmp
READF,lun,tmp
READF,lun,data

FREE_LUN,lun

template =Ascii_template(asciiFile)
data = Read_ascii(asciiFile,TEMPLATE=template,count = yNum)
help,data
print,data.(0)
```

5.2.3 读写二进制文件

二进制数据文件基本是以一长串的二进制字节存在文件中，这些字节的含义（也就是说，这些字节如何翻译成特定数据类型和结构的）很难描述的，除非刚开始就知道文件写入的是什么内容。二进制数据比格式化数据紧凑得多，经常用于存储大数据文件。有两种命令读写二进制数据，它们与前面用来读取格式数据文件的 ReadF 和 Print 命令等效。它们是 ReadU 和 WriteU 命令。

要读取非格式数据文件，简单定义变量，打开文件读取，并用 ReadU 命令将字节一个接一个地读入变量中。如果给定了变量的数据类型和组织结构，每个变量按其要求从文件中读出相应的字节数。

5.2.3.1 读写常规二进制文件

```
IDL> B = BYTARR(192, 192, /NOZERO)
IDL> ; Read the data into B:
IDL> READU, 1, B
IDL> ; Close the file:
IDL> CLOSE, 1
IDL> ; Display the image:
IDL> TV, B
```

5.2.3.2 读写带头文件的二进制文件

当数据中含有特定的头文件时，读取的时候有时希望直接跳过，此时需要用 Point_Lun 函数，该函数支持对文件进行字节定位。

调用格式为POINT_LUN, Unit, Position，其中 Unit 是文件打开后的lun，Position 一般是需要跳转的字节数。

5.2.3.3 关联变量读写二进制文件

IDL 关联变量是把一个 IDL 数组或结构变量的组织结构映射到数据文件的内容上。文件被看作是这些重复单元的一个数组。 第一个单元的索引号是 0，第二个单元的索引号 1 等等。关联变量不象常规变量那样将整个数据组都存储在内存里。而是当一关联变量被引用时，IDL 仅对需要的部分数据执行相关的输入或输出请求，这部分数据就是要读入内存的。

有时候大型的二进制数据文件都由一系列的重复单元组成。一个 IDL 关联变量通常是处理这种数据形式的最好方式（有时候是唯一的方式）。

关联变量有下面的几个优点：

- ✧ 当该变量被用于表达式时，才产生文件的输入和输出动作。不需要单独的读或写命令。
- ✧ 数据集的大小不受内存容量的限制，因为有时它可处理大型的数据集。对于物理存储器来说是太大的数据，通过把此数据分成块就能很容易地处理。
- ✧ 不必提前声明用于映射该数据的数组或结构的数量。
- ✧ 关联变量是效率最高的 I/O 形式。

5.2.3.4 向导式读取函数

READ_BINARY 和 BINARY_TEMPLATE

Read_Binary 是直接读取二进制文件的函数，用法可参考下面的语句：

```
vHeadData = READ_BINARY(FILEPATH('head.dat', $
  SUBDIRECTORY=['examples', 'data']), DATA_DIMS=[80,100,57])
IVOLUME, vHeadData
```

5.2.3.5 例子

读 FY-2 的 9210AWX 文件，FY-2C 的卫星黑体亮度温度产品，下面的格式说明是参考国家卫星气候中心的《风云二号 C 卫星业务产品使用手册》。

1) 第一级文件头记录格式

序号	字节序号	字节数	类型	描 述 内 容	具体数值或字符
1	1-12	12	A*12	文件名	TMG#####·AWX
2	13-14	2	I*2	整型数的字节顺序	0
3	15-16	2	I*2	第一级文件头长度	40
4	17-18	2	I*2	第二级文件头长度	80
5	19-20	2	I*2	填充段数据长度	1081
6	21-22	2	I*2	记录长度	1201
7	23-24	2	I*2	文件头专用记录数	2
8	25-26	2	I*2	产品数据专用记录数	1201
9	27-28	2	I*2	产品类别	3
10	29-30	2	I*2	压缩方式	0
11	31-38	8	A*8	格式说明字符串	SAT2004
12	39-40	2	I*2	产品数据质量标记	2

2) 第二级文件头记录格式

序号	字节序号	字节数	类型	描 述 内 容	具体数值或字符
1	41-48	8	A*12	卫星名	FY2C
2	49-50	2	I*2	格点场要素	19
3	51-52	2	I*2	格点数据字节	1
4	53-54	2	I*2	格点数据基准值	100
5	55-56	2	I*2	格点数据比例因子	1
6	57-58	2	I*2	时间范围代码	1(或 2、3、4)
7	59-60	2	I*2	开始年	####
8	61-62	2	I*2	开始月	##
9	63-64	2	I*2	开始日	##
10	65-66	2	I*2	开始时	##
11	67-68	2	I*2	开始分	##
12	69-70	2	I*2	结束年	####
13	71-72	2	I*2	结束月	##
14	73-74	2	I*2	结束日	##
15	75-76	2	I*2	结束时	##
16	77-78	2	I*2	结束分	##
17	79-80	2	I*2	网格左上角纬度	6000
18	81-82	2	I*2	网格左上角经度	4500
19	83-84	2	I*2	网格右上角纬度	-6000
20	85-86	2	I*2	网格右上角经度	16500

21	87-88	2	I*2	格距单位	0
22	89-90	2	I*2	横向格距	10
23	91-92	2	I*2	纵向格距	10
24	93-94	2	I*2	横向格点数	1201
25	95-96	2	I*2	纵向格点数	1201
26	97-98	2	I*2	有无陆地判识值	0
27	99-100	2	I*2	陆地具体判识值	0
28	101-102	2	I*2	有无云判识值	0
29	103-104	2	I*2	云具体判识值	0
30	105-106	2	I*2	有无水体判识值	0
31	107-108	2	I*2	水体具体判识值	0
32	109-110	2	I*2	有无冰体判识值	0
33	111-112	2	I*2	冰体具体判识值	0
34	113-114	2	I*2	是否有质量控制	3
35	115-116	2	I*2	质量控制上限	330
36	117-118	2	I*2	质量控制下限	170
37	119-120	2	I*2	异常点填充值	0

```
PRO test_ReadBinAWX
    ;
    currentPath = sourceroot()
    file = currentPath+'data\FY2C_TBB_IR1_OTG_20061130_AOAD.AWX'
    ;
    OpenR, file_lun, file ,/Get_Lun
        ;定位到信息部分
        point_lun,file_lun,20
        HeadLine =indgen(3)
        readu,file_lun,HeadLine
        ;HeadLine[0]-数据的头文件长度 $
        ;HeadLine[]-文件头文件记录数 $
        ;HeadLine[2]-数据的记录数

        point_lun,file_lun,58
        ;定位到信息部分
        BeginDate=indgen(5) ;依次为年月日时分
        EndDate =indgen(5)    ;依次为年月日时分
        LatLong=indgen(4)     ;依次为左上角纬度经度右下角经度纬度

        readu,file_lun,BeginDate
        readu,file_lun,EndDate
        readu,file_lun,LatLong
        ;
        data = bytarr(HeadLine[2],(HeadLine[0]))
```

```
    ;定位到数据部分
        point_lun, file_lun, HeadLine[0]*HeadLine[1]
        readu, file_lun, data

    Free_Lun, lun
END
```

5.3 读写通用格式文件

IDL 可以读写许多常用文件格式，这些文件格式已在表 5-5 中列出。

表 5-5 IDL 能够读写许多常用的数据文件格式

文件格式	读	写
BMP	Read_BMP	Write_BMP
CDF	参考 CDF 库	参考 CDF 库
DICOM	IDLffDICOM 对象	IDLffDICOM 对象
DXF	IDLffDXF 对象	IDLffDXF 对象
GIF	Read_GIF	Write_GIF
HDF	参考 HDF 库	参考 HDF 库
HDF-EOS	参考 HDF 库	参考 HDF 库
Interfile	Read_Interfile	无
JPEG	Read_JPEG	Write_JPEG
netCDF	参考 netCDF 库	参考 netCDF 库
PICT	Read_PICT	Write_PICT
PBM/PPM	Read_PPM	Write_PPM
PNG	Read_PNG	Write_PNG
PostScript	无	PS 或打印设备
Sun Rasterfiles	Read_SRF	Write_SRF
SYLK	Read_SYLK	Write_SYLK
TIFF/GeoTIFF	Read_TIFF	Write_TIFF
WAVE	Read_WAVE	Write_WAVE
X11-bitmap	Read_X11_Bitmap	无
XWD	Read_XWD	无

5.3.1 图像文件信息查询

IDL 中提供了常用图像文件格式的查询程序。这些程序允许在没有真正读取其数据的情况下，就可以查询图像文件，并可以存取随着图像数据文件一起存储在文件里的元数据（关于数据的一些信息）。

下面是新的图像查询程序列表：

- Query_BMP
- Query_DICOM
- Query_GIF

- Query_JPEG
- Query_PICT
- Query_PNG
- Query_PPM
- Query_SRF
- Query_TIFF

所有这些查询命令都是以同样的方式工作。它们都是返回 0 或 1 的函数，通过返回值确定是否成功地（返回值为 1）读取了图像文件里的元数据。如果它们成功地读取了文件，将保存文件信息的 IDL 结构变量作为输出命令返回给用户。用户通过存取这个结构里面的字段从而获取文件的有关信息。

5.3.2 JPEG 文件操作

5.3.2.1 JPEG 读操作

用 Read_JPEG 命令就可以读取并显示一个 JPEG 文件。

假设正在 8 位显示器上显示 24 位的图像，应该使用 TV 命令里的关键字 True：

```
IDL> TV, thisimage, true=1
```

5.3.2.2 JPEG 写操作

对图形窗口进行拍照可以用 TVRD 函数。

```
IDL> image3D=TVRD(true=1)
```

最后，用 Write_JPEG 命令，将图像用输出为 JPEG 文件，运行命令：

```
IDL> Write_JPEG, ‘test.jpg’, image3D, true=1, quality=75
```

5.3.2.3 举例

```
PRO test_CommonImg
  currentPath = sourceroot()
  imgFile =currentPath+'\data\idl.jpg'
  ;文件信息查询
  queryStatus = QUERY_IMAGE(imgFile, imageInfo,type = type)
  HELP,imageInfo,/str
  IF queryStatus EQ 1 THEN BEGIN
    CASE type OF
      'BMP': BEGIN
      END
      'JPEG': BEGIN
        jpgStatus=QUERY_JPEG(imgFile, fileinfo)
        IF jpgStatus THEN BEGIN
```

```
                HELP,fileInfo,/str
                READ_JPEG,imgFile,data
                TV,data,/true
                 image=TVRD(0,0,127,287,/true)
                Write_Jpeg,currentPath+'\data\result.jpg', image,true=1, quality=75
              ENDIF
            END
            ELSE: BEGIN
            END
          ENDCASE
        ENDIF
      END
```

5.3.3 HDF 数据读入

HDF 有两种基本对象—组（Group）和数据集（Dataset），同时也有一些其他的辅助对象，如数据类型（Datatype）、数据空间（Dataspace）、属性（Attribute）。HDF 组织结构有些类似于 Windows 中的文件结构，组相当于一个文件夹，数据集相当于一个数据文件，文件夹里面可以包含数据文件，也可以包含其他的组对象，这样嵌套下去，就形成了一个复杂的数据对象，这点正好的对应于 HDF 逻辑结构的层次性。

HDF 文件的操作 读取-编辑

HDF_OPEN: 这个过程首先要打开 HDF 文件，通过 create 关键字来确定是读取文件还是创建一个新的文件。

HDF_DFAN_ADDFDS: 添加文件描述；Optionally， add a file description.

HDF_DFAN_ADDFID: 添加文件 Optionally， add a file annotation.

对 HDF 文件添加数据用到的过程：

添加 8 位图像，用 HDF_DFR8_ADDIMAGE 或 DFR8_PUTIMAGE；

添加颜色表，用 HDF_DFP_ADDPAL 或 HDF_DFP_PUTPAL；

添加 24 位图像，用 HDF_DF24_ADDIMAGE 或 HDF_DF24_PUTIMAGE；

添加多数据时，需要用下面的命令：

HDF_SD_CREATE 或 HDF_SD_SELECT 建立一个 SDS 或选择一个现有的 SDS；HDF_SD_DIMSET 设置维数信息；

HDF_SD_ATTRSET 设置属性信息；

HDF_SD_SETINFO 添加一个数据信息；

HDF_SD_ADDDATA 添加数据；

HDF_SD_SETEXTFILE 移动数据到外部存储文件；

HDF_SD_ENDACCESS 结束读取 SDS；

添加数据表（vData）时需要用下面的命令：

HDF_VD_ATTACH 获取一个数据表的标识；to get a Vdata identifier.

HDF_VD_SETINFO 写入数据表的信息；

HDF_VD_FDEFINE to prepare non-trivial fields (optional).

HDF_VD_WRITE 写入数据表；

将数据表(Vdata)添加到数据组合(Vgroup)中时，用下面的命令：

HDF_VG_ATTACH to get a Vgroup identifier.

HDF_VG_SETINFO to set the Vgroup name and class (optional).

HDF_VG_INSERT to add the Vdata to a Vgroup.

HDF_VG_DETACH to close the Vgroup.

HDF_CLOSE to close the file

例子读取 Modis 数据,Test_HDF.pro

bbs.esrichina-bj.cn

6 IDL 界面创建与事件处理

IDL 的界面分下面几种：

1) IDL 命令行界面

2) 智能化工具 iTools 界面

3) 常规组件界面

4) 定制 iTools 界面

6.1 iTools 界面

可见 10 章。

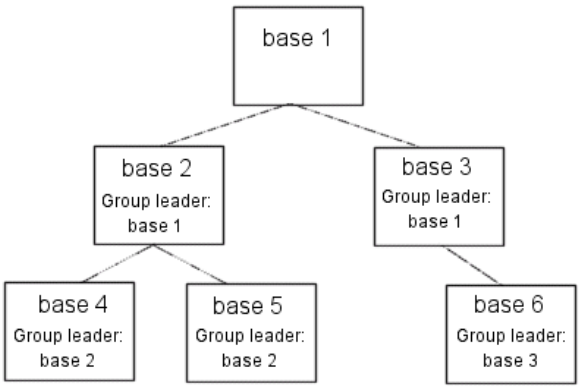
6.2 界面创建

IDL 下的界面创建是比较特殊的，由于 IDL7 的 Eclipse 更新，当前版本下没有了鼠标拖拽创建 GUI 的工具，只能依靠代码创建。

6.2.1 单元组件界面

6.2.1.1 Base 组件（容器界面）

IDL 下的界面有很多的 base，可以理解为容器，而且很多的时候一些组件的排列如果用了 base 会很方便。多层的结构一般如下图



1. 格式：

Result = WIDGET_BASE([Parent] [, /Keywords)

2. 常用关键字

1) Floating

浮动的窗口，类似导航图，仅浮动在窗口上方；

2) Model

是否为模式框，在该窗口关闭前无法进行其他操作；

3) mBar

是否创建菜单栏，返回菜单栏的 ID；

4) TLB_FRAME_ATTR

创建的窗口类型，见表格，类型的值可累加，如 3 则表示 1+2。

值	窗口类型
1	窗口无法进行大小、最大化等操作
2	不显示系统菜单
4	禁止显示标题栏
8	窗口无右上的关闭按钮
16	窗口无法移动

5) TLB_KILL_REQUEST_EVENTS

是否返回关闭事件，可用 TAG_NAMES(event, /STRUCTURE_NAME) 是否为 'WIDGET_KILL_REQUEST'来判断。

3. 参考例子 Test_widget_base

6.2.1.2 按钮组件(widget_button)

Widget_button 创建具体包括：触发按钮、位图按钮（主要用于创建工具栏或者图形按钮）、单选按钮、复选按钮、标准菜单系统、按钮菜单系统和快捷菜单系统。

1. 格式：

Result = WIDGET_BUTTON([Parent] [, /Keywords)

2. 常用关键字

1) TOOLTIP

显示的文字，如下图。



2) Value

a) 字符串，显示 button 上的文字；

b) n x m 字节型数组，显示成黑白图片；

c) n x m x 3 字节型数组，显示成 24 位真彩色图片；

d) Bitmap 控制是否显示图标型 button，如果设置了 Bitmap，则 value 可以直接设置为 bmp 文件名。



3) Menu

创建菜单选项；

4) SEPARATOR

创建分割线；



5) 复选或单选

Base 的 EXCLUSIVE 和 NONEXCLUSIVE 关键字。



4. 参考例子 Test_widget_button

6.2.1.3 标签组件(widget_Label)

widget_label 用来创建标签组件。

1. 格式：

Result = WIDGET_LABEL([Parent] [, /Keywords)

2. 常用关键字

1) value

其值为显示的字符串；

换行的话字符串中间用 String(13b)连接下即可。

3. 参考例子 Test_widget_label

6.2.1.4 显示组件(widget_draw)

widget_draw 用来创建 draw 组件，draw 组件是 IDL 下的矩形的标准图像区域，该组件通过关键字控制创建直接图形法窗口还是对象图形法窗口。

1. 格式：

   Result = WIDGET_DRAW([Parent] [, /Keywords)

2. 常用关键字

1) COLOR_MODEL

   对象图形法下的色彩模式控制，如设置为 1 则使用索引颜色，否则使用 RGB 颜色；

2) GRAPHICS_LEVEL

   直接图形法和对象图形法的选择，如设置为 2 则为对象图形法，系统默认或设置为 1 则为直接图形法窗口。

3) EXPOSE_EVENTS

   是否有“暴露”事件，也就是如果当前窗口被其他窗口遮挡后再显示的时候是否需要产生事件，与 RETAIN 结合加快显示速度。

4) RETAIN

   0：不进行后备存储器操作（处理器（PROCESSOR）暂时不需要的程序和数据）备份；

   1：操作系统进行后备存储器操作；

   2：IDL 进行后备存储器操作；

5) WHEEL_EVENTS

   产生鼠标事件关键字，如进行滚轮放大缩小操作，需要添加该关键字。类似的还有 KEYBOARD_EVENTS 等，在事件中以 event.type 来区别。

4. 参考例子 Test_widget_draw

6.2.1.5 属性列表组件(WIDGET_PROPERTYSHEET)

widget_propertysheet 用来创建属性信息组件，该组件通过关键字来进行信息的修改，用在对象法中。

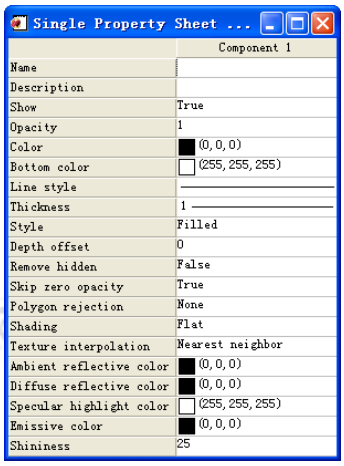
1. 格式：

   Result = WIDGET_PROPERTYSHEET ([Parent] [, /Keywords)

2. 常用关键字

1) Value

   Value 是单个对象或对象集，属性的创建或修改在对象的方法中实现。



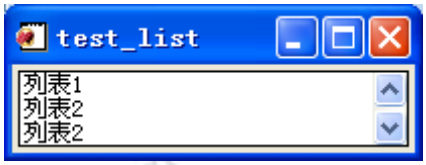
6.2.1.6 列表组件(WIDGET_LIST)

widget_list 用来创建属列表选择组件，该组件通过关键字来进行列表的选取。



1. 格式：

   Result = WIDGET_LIST ([Parent] [, /Keywords)



2. 常用关键字

1) Value

   是列表中的元素值。

   备注：通过 index 来判断当前选择的是哪一个；

3. 参考例子 Test_widget_list

6.2.1.7 下拉列表组件(WIDGET_DROPLIST)

跟 widget_list 组件类似，widget_droplist 用来创建属下拉列表，该组件则是通过关键字来进行列表的选取。



1. 格式：

   Result = WIDGET_DROPLIST ([Parent] [, /Keywords)

2. 常用关键字

1) Value

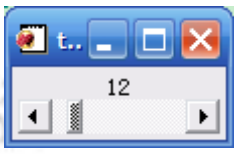
   是下拉列表中的元素值。

   备注：通过 index 来判断当前选择的是哪一个；

3. 参考例子 Test_widget_droplist

6.2.1.8 滑动条组件(widget_slider)

widget_slider 用来创建滑动条组件，交互的获取或者设置指定范围内的一个整数值，该组件则是通过关键字来进行列表的选取。



1. 格式：

   Result = WIDGET_SLIDER ([Parent] [, /Keywords)

2. 常用关键字

1) Value

   是滑动条中的默认值。

3. 参考例子 Test_widget_slider

6.2.1.9 表格组件 (widget_table)

widget_table 用来创建表格组件，交互的获取或者设置指定范围内的一个整数值，该组件则是通过关键字来进行列表的选取。



1. 格式：

   Result = WIDGET_TABLE ([Parent] [, /Keywords)

2. 常用关键字

1) Value

   是滑动条中的默认值。

3. 参考例子 Test_ widget_table

6.2.1.10 Tab 组件(widget_tab)

Widget_Tab 用来创建页面组件。



1. 格式：

   Result = WIDGET_TAB ([Parent] [, /Keywords)

2. 常用关键字

1) Title 是 TAB 窗口中的标题文字；

2) LOCATION 是控制窗口显示的位置；

3. 参考例子 Test_ widget_tab

6.2.1.11 树组件(widget_tree)

WIDGET_TREE 组件是创建的树形列表的组件，如下图。

1. 格式：

Result = WIDGET_TREE ([Parent] [, /Keywords)

2. 常用关键字

1) Value

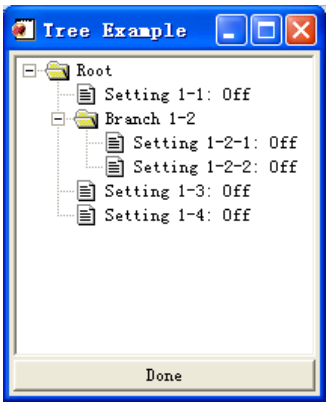
是 tree 的节点显示的名称。

2) Folder

当前节点是文件夹节点，可以包括叶子节点

3) EXPANDED

如果节点是文件夹，此关键字可以进行文件的展开、折叠控制。



3. 参考例子

1) C:\ProgramFiles\ITT\IDL71\examples\doc\widgets\

tree_widget_example.pro

2) Test_widget_tree

6.2.1.12 文字组件(Widget_Text)

WIDGET_TEXT 组件是创建文字输入编辑组件，如下图。

1. 格式：

Result = WIDGET_TTEXT ([Parent] [, /Keywords)

2. 常用关键字

1) Value

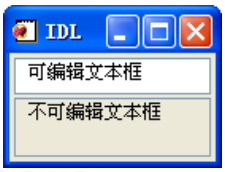
是 text 组件文字框中显示的字符。

2) Editable

当前字符是否能编辑。

3) XSize 与 YSize

这里的 Ysize 及 Xsize 的单位都是字符宽度，而不是像素。



6.2.2 复合界面

复合界面是 IDL 下的一些具备某些功能的界面程序，均以 CW_开头，这类组件具备一定独立的功能，详细可参考 Help 的列表。

先看 CW_FIELD，从该组件的运行效果来看，它就是 Widget_Base、widget_label 和 Widget_text 组件的结合，但从关键字上看，它可以实现其他组件无法实现的效果。

1. 格式：

   Result = CW_FIELD ([Parent] [, /Keywords)

2. 常用关键字

1) Title

   是组件文字框前面显示的字符。

2) NOEDIT

   当前字符不可编辑。

3) 数据类型

这类关键字包括 Floating、Long、Integer、String 等，设置了对应的关键字后则输入时只能对应类型字符，其他类型字符无法输入。这在一定程度上减轻了编程人员的工作量。



其他复合界面的介绍与使用可参考 IDL 帮助，见下图：

Widget Routines, Compound

CW_TMPL - Template for compound widgets that use XMANAGER.

Animation

CW_ANIMATE - Creates a compound widget for animation.

CW_ANIMATE_GETP - Gets pixmap window IDs used by CW_ANIMATE.

CW_ANIMATE_LOAD - Loads images into CW_ANIMATE.

CW_ANIMATE_RUN - Displays images loaded into CW_ANIMATE.

Color Manipulation

6.2.3 对话框界面

对话框组件是 Modal 对话框的组件，对话框是一种用户界面，它的主要功能是输出信息和接收用户的输入。如常用的文件选择对话框 Dialog_PickFile 和信息提示 Dialog_Message。

对话框组件 Dialog_PickFile

1. 格式：

   Result = Dialog_PickFile ([Parent] [, /Keywords)

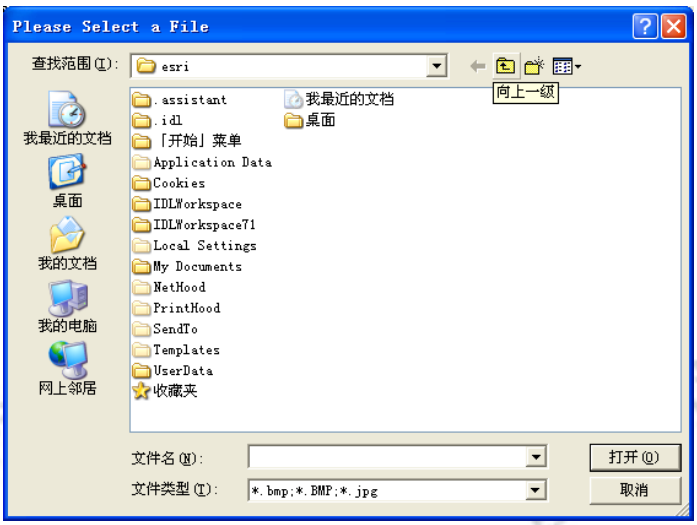
2. 常用关键字

1) Title

   对话框的标题字符。

2) Filter

   文件类型过滤，如 filters = ['*.jpg', '*.tif', '*.png']则指定文件输入为上面三种类型。







IDL API Reference Guides > IDL Reference Guide > Part I: IDL Command Reference > Functional List of IDL Routines

Dialog Routines

DIALOG_MESSAGE - Creates modal message dialog.

DIALOG_PICKFILE - Creates native file-selection dialog.

DIALOG_PRINTERSETUP - Opens native dialog used to set properties for a printer.

DIALOG_PRINTJOB - Opens native dialog used to set parameters for a print job.

DIALOG_READ_IMAGE - Presents GUI for reading image files.

DIALOG_WRITE_IMAGE - Presents GUI for writing image files.

6.2.4 功能界面

功能界面是 IDL 自带部分特殊函数，这些函数能够实现一些特殊的功能并能够被调用，大部分有源码可以参考。见下表

名称	概述
EFONT	矢量字体编辑器和显示的工具
IDLEXBR_ASSISTANT	打开输出助手
SLIDE_IMAGE	创建一个滚动的图形窗口用来大图像

XBM_EDIT	创建，编辑 IDL 编译器工具按钮的位图图标
XDISPLAYFILE	显示 ASCII 码文本文件的组件
XDXF	查看和操控 DXF 对象
XFONT	选择和浏览 X Windows 字体的模式窗口
XINTERANIMATE	动态显示序列图像
XLOADCT	选择和设置颜色表的工具
XMTOOL	查看 XMANAGER 管理的组件的工具
XOBJVIEW	查看图形对象的工具
XOBJVIEW_ROTATE	旋转 XOBJVIEW 里面的对象；
XOBJVIEW_WRITE_IMAGE	将 XOBJVIEW 中的对象输出成图片
XPALETTE	创建和修改颜色表的工具
XPCOLOR	颜色查看器
XPLOT3D	创建和手动编辑三维曲线
XROI	直接图形法下的感兴趣编辑工具
XSURFACE	图形方式查看 Surface
XVAREDIT	IDL 下的变量修改工具
XVOLUME	查看和操作体
XVOLUME_ROTATE	对 XVOLUME 显示的体对象进行旋转
XVOLUME_WRITE_IMAGE	将 XVOLUME 中的对象输出成图片

6.3 界面的控制

界面组件创建后，需要对组件进行属性的设置或获得组件的状态，IDL 提供了 Widget_Contrl 和 Widget_Info 两个函数对组件信息进行控制。

6.3.1 Widget_Control

Widget_Control 有下面四种用途：

- ✧ 组件初始化显示；
- ✧ 组件属性信息的设置和读取；
- ✧ 组件销毁；

调用格式如下：Widget_Control，id，[关键字]

组件常用关键字简要列表如下

关键字	用处
set_uvalue	设置组件的 uvalue
get_uvalue	获得组件的 uValue
hourglass	控制等待鼠标沙漏指针

scr_xsize	组件的屏幕宽度
scr_ysize	组件的屏幕高度

6.3.2 Widget_Info

用 Widget_Info 是获得已经存在组件的信息。

调用格式如下：result= Widget_Info(id，[关键字])

常用关键字简要列表如下

关键字	用处
geometry	设置组件的位置偏移和大小信息
parent	获得组件的父组件
child	获得组件的孩子组件
uname	获得组件的 uname
VALID_ID	组件是否有效
FIND_BY_UNAME	通过 UNAME 查找组件

6.4 事件的处理

IDL 的图形窗口中，每发生一个动作都会产生事件，每个都会返回一个事件结构，可以分四个部分来学习。一是事件结构的内容，二是要有一个对事件进行处理的“管家”，三是需要获得事件并进行区分，最后不同的事件中做不同的处理。

6.4.1 事件的结构

事件结构体中包含了组件的相关信息。事件结构式 IDL 的结构变量，事件的结构中通用的字段有 ID、Top 和 Handler 三个，这三个字段都是长整型的变量。

ID，通常是长整型，是事件过程中组件的惟一标示符。

Top 是组件层次结构中最顶级的 base 的标识；

Handle 是与事件处理程序相关联的组件的标识；

6.4.2 事件的“管家”

XManager 能够对产生界面的事件，可称为是 IDL 事件的管家。常规界面事件程序生成的步骤可分为：创建界面、显示界面和 Xmanager 进行事件的生成三

部分。XManager 产生事件的生命周期是界面存在的生命周期，直到界面销毁 XManager 的事件才会消失。

调用方式：Xmanager，Name，ID，[关键字]

Name 是程序的文件名，ID 是组件的 ID 号。

常用关键字有 EVENT_HANDLER 和 NO_BLOCK，其中 EVENT_HANDLER 可以指定事件的处理程序名，如不指定事件程序名则默认是 Name_event 程序。NO_BLOCK 关键字是在程序运行过程中对命令行是否进行屏蔽。

参考例子[Traning\chapter6\test_xmanager.pro]

6.4.3 事件的处理

界面程序中可以先产生事件，然后在事件处理程序中，用 Tag_names 对产生的事件类型进行获得或根据 Uname 事件，tag_names 的功能是获取结构体的名字。

例子程序[Traning\chapter6\test_event.pro]对 draw 中的各种事件进行了演示，包括了鼠标事件、键盘事件、暴露事件、关闭事件和 resize 事件的常用事件。

bbs.esrichina-bj.cn

7 直接图形法

7.1 概述

直接图形法是 IDL 的图像或图形直接绘制在用户指定的图形设备上，可在很短的时间内生长高质量的图形。执行内置直接图形程序，将会建立坐标系，创建轴和边界。组成直接图形系统的例行程序，如 plot，contour 及 surface 等等，可以完成大部分关于图形输出的操作。直接绘图程序一旦生成直接图形，结果就会被输出到当前设备，并且删除掉所有相关数据。若要对已生成的直接图形做出改动或是将结果输出到另外的设备，就必须要重新执行。

直接图形法对内存有默认的管理方法，占用内存少，用户不必自己释放内存，方便、简洁易用。

7.2 图形显示设备

直接图形法的显示是显示在设备上的，IDL 中的设备包括（显示器、打印机等），可以通过下面的 IDL 命令来查看当前显示设备，表 7-1 中列出了 IDL 支持的图形设备名字。

```
IDL> print,!d.name
WIN
```

表 7-1 IDL 直接图形法设备名称及描述

设备名称	描述
CGM	计算机图元文件
HP	惠普图形语言 (HP-GL)
METAFILE	Windows 图元文件格式 (WMF)
NULL	没有图形输出
PCL	惠普打印机控制语言(PCL)
PRINTER	系统打印机
PS	PostScript
REGIS	吉斯图形议定书(DEC systems only)
TEK	泰克兼容终端
WIN	微软 Windows

X	X Window 系统
Z	Z 轴缓冲伪设备

7.3 显示模式

直接图形法有两种显示模式：8 位显示模式和 24 位显示模式。

7.3.1 8 位显示模式

8 位显示模式下，只有 256 种颜色，在 IDL 中通过 Device 设置颜色模式，Decomposed=0 表示 8 位显示模式，颜色索引从 0 到 255。此时的显示的颜色有颜色表来控制，对输入的颜色索引进行颜色表中查找，查找最接近的匹配值并显示。颜色表的控制用 XloadCT 和 LoadCT 函数。

```
IDL> Device,decomposed = 1
IDL> tvscl,dist(400)
```




```
IDL> Device,decomposed = 0
;获得系统的颜色表
IDL> LOADCT
```

0- B-W LINEAR	14- STEPS	28- Hardcandy
1- BLUE/WHITE	15- STERN SPECIAL	29- Nature
2- GRN-RED-BLU-WHT	16- Haze	30- Ocean
3- RED TEMPERATURE	17- Blue - Pastel - R	31- Peppermint
4- BLUE/GREEN/RED/YE	18- Pastels	32- Plasma
5- STD GAMMA-II	19- Hue Sat Lightness	33- Blue-Red
6- PRISM	20- Hue Sat Lightness	34- Rainbow
7- RED-PURPLE	21- Hue Sat Value 1	35- Blue Waves
8- GREEN/WHITE LINEA	22- Hue Sat Value 2	36- Volcano

```
 9- GRN/WHT EXPONENTI   23- Purple-Red + Stri   37-            Waves
10-         GREEN-PINK   24-              Beach   38-        Rainbow18
11-           BLUE-RED   25-         Mac Style   39-   Rainbow + white
12-           16 LEVEL   26-             Eos A   40-   Rainbow + black
13-            RAINBOW   27-             Eos B
```

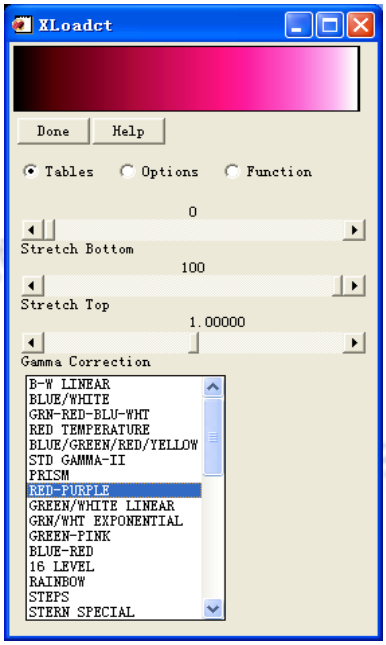
Enter table number: 1

```
% LOADCT: Loading table BLUE/WHITE
```

IDL> tvscl,dist(400)



IDL> xloadct



IDL> tvscl,dist(200,200)

自定义颜色表需要用 Tvscl 进行调入。

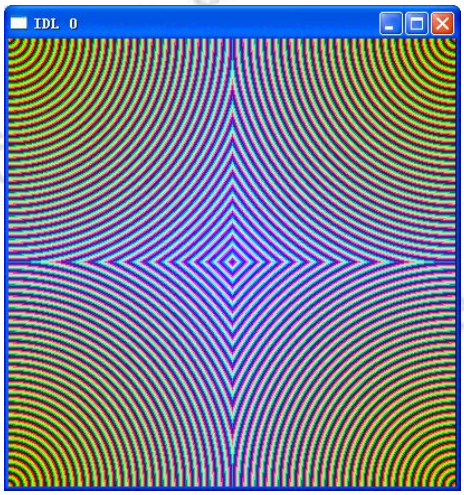
IDL> DEVICE, DECOMPOSED = 0

IDL> ; Create a set of R, G, and B colormap vectors:

IDL> R = BYTSCL(SIN(FINDGEN(256)))

IDL> G = BYTSCL(COS(FINDGEN(256)))

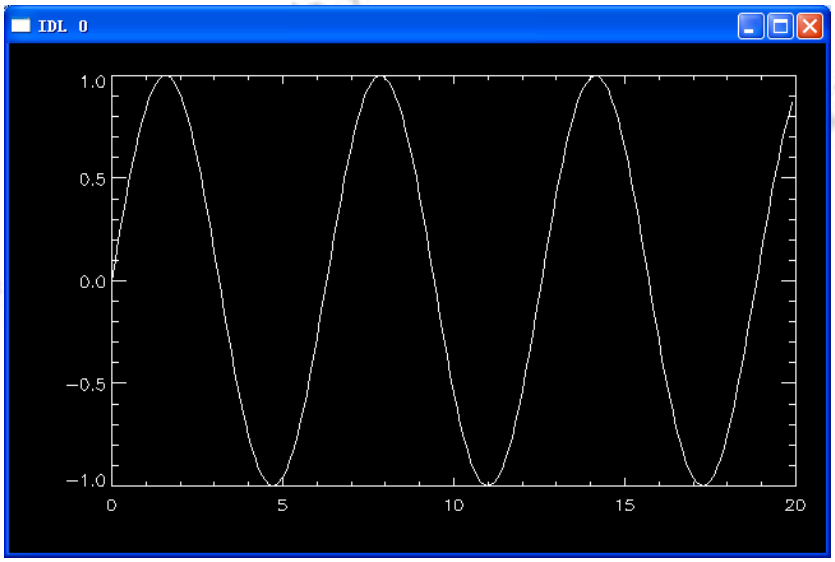
```
IDL> B = BINDGEN(256)
IDL> ; Load these vectors into the color table:
IDL> TVLCT, R, G, B
IDL> ; Display an image to see the effect of the new color table:
IDL> TVSCL, DIST(400)
```



7.3.2 24 位显示模式

24 位显示模式下有 256*256*256 种颜色，如绘制一正弦曲线，可依次输入下列命令:

```
IDL> Device,decomposed = 1
IDL> xdata = findgen(200)/10
IDL> plot,xData,sin(xData)
```



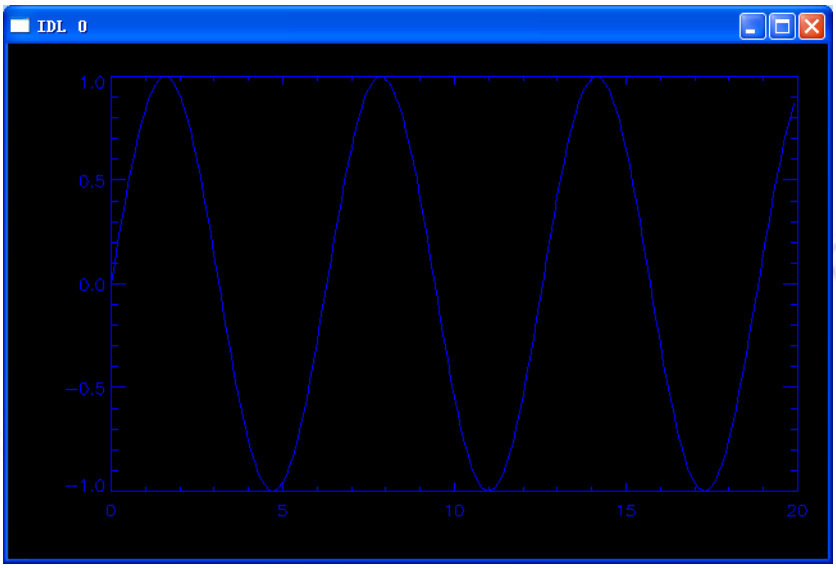
此时颜色的 RGB 值也需要转换为索引值以便使用，转换的方法为：

index=long(red)+256L*long(green)+256L^2*long(blue)

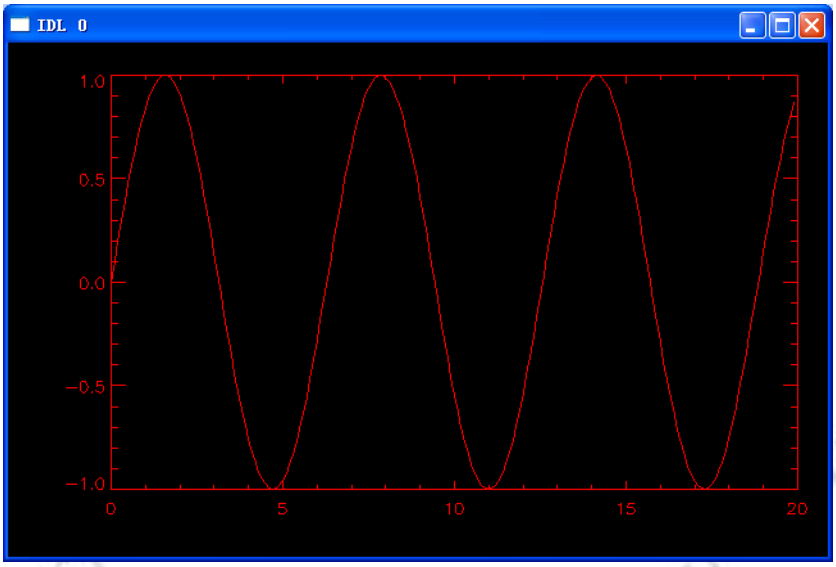
如 RGB[0,0,255]转换为索引值为 16711680L，

RGB[255,0,0]转换为索引值是 255L。

```
IDL> plot,xData,sin(xData),color = 16711680L
```




```
IDL> plot,xData,sin(xData),color = 255
```



7.4 系统的字体

IDL 下的字体分为三种，可以根据系统变量!P.Font 的值进行判断。值为-1 代表矢量字体（也可称为 Hershey 字体），值为 0 代表设备字体或系统字体和 1 代表 TureType 轮廓字体。

7.4.1 矢量字体

矢量字体是直接图形法下的缺省字体，它们有不依赖于平台的优点且极易在三维空间旋转。IDL 系统中有下面这些类型的矢量字体。

Font 3, Simplex Roman	Font12, Simplex Script
Font 4, Simplex Greek	Font 13, Complex Script
Font 5, Duplex Roman	Font 14, Gothic Italian
Font 6, Complex Roman	Font 15, Gothic German
Font 7, Complex Greek	Font 16, B Cyrilic
Font 8, Complex Italic	Font 17, Triplex Roman
Font 9, Math and Special	Font 18, Triplex Italic
Font 11, Gothic English	Font 20, Miscellaneous

可查看系统帮助链接

http://127.0.0.1:57564/help/topic/com.rsi.idl.doc.core/Hershey_Vector_Font_Samples.html?resultof=%22%66%6f%6e%74%22%20

在缺省值情况下，直接图形使用的是矢量字体，即 AXIS, CONTOUR, PLOT, SHADE_SURF, SURFACE, XYOUTS.等功能函数中如设置关键词 font 为-1，则使用的字体为矢量字体。

字体选择可通过字符串前添加”!index”来进行，其中 index 为字体的索引，取值范围 3~20，文字大小可通过关键字 charsize 进行控制。

```
IDL> XYOUTS,.1, .1, '!3Hershey Font ' ,    /NORM, CHARSIZE=4
IDL> XYOUTS,.1, .4, '!4Hershey Font ' ,    /NORM, CHARSIZE=4
IDL> XYOUTS,.1, .7, '!5Hershey Font ' ,    /NORM, CHARSIZE=4
```



7.4.2 设备字体

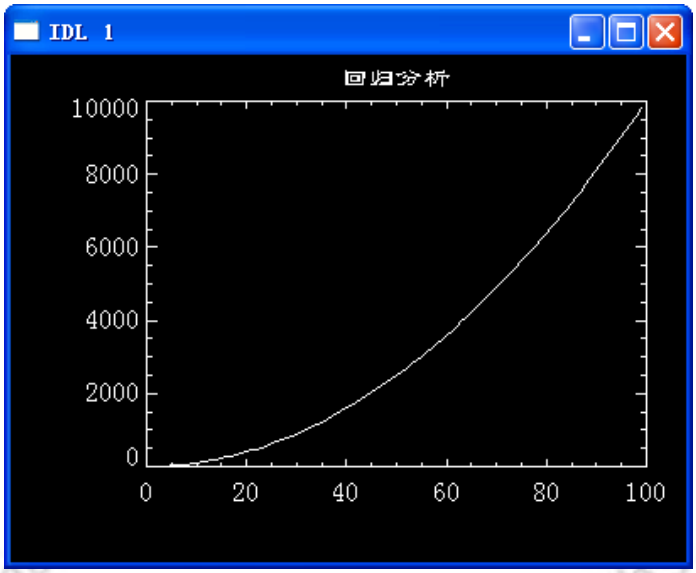
设备字体（也称为系统字体）通过设置!P.Font 系统变量或 Font 关键字为 0 来加以选择。

如在 AXIS, CONTOUR, PLOT, SHADE_SURF, SURFACE, XYOUTS.等中进行系统字体的显示时，需要采用这种方式。系统字体的获得可以通过 Device 函数来获得，获得系统字体个数及系统字体名字的方法如下：

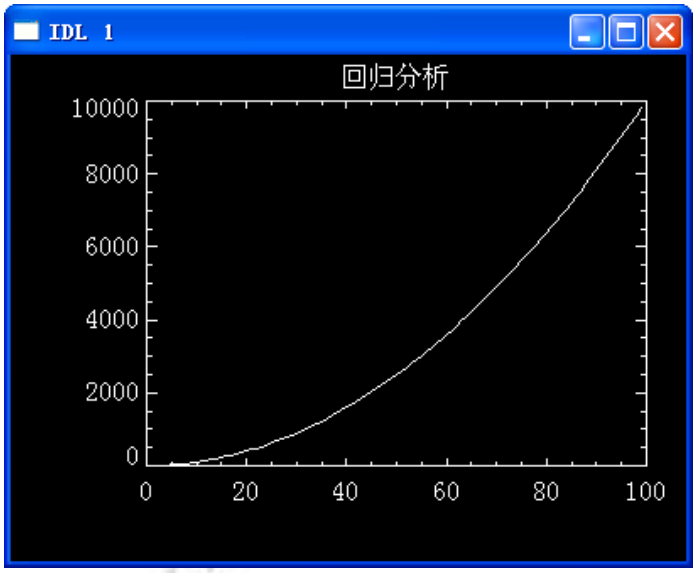
```
IDL> Device,Set_Font='*',Get_FontNum = deviceFontsNum
IDL> print,deviceFontsNum
223
IDL> Device,Set_Font='*',Get_FontNames = DeviceFonts
IDL> help,DeviceFonts
DEVICEFONTS       STRING      = Array[223]
```

如显示输出汉字的方法；

```
IDL> Window,1,xsize = 400,ySize =300
IDL> x=FINDGEN(100)
IDL>   y=x^2
IDL>;设置自定义字体模式
IDL>!p.font = 0
IDL> PLOT, x, y, TITLE='回归分析'
```




```
IDL>;设置字体
IDL> DEVICE,Set_Font     ='宋体'
IDL> PLOT,x,y ,title ='回归分析'
```



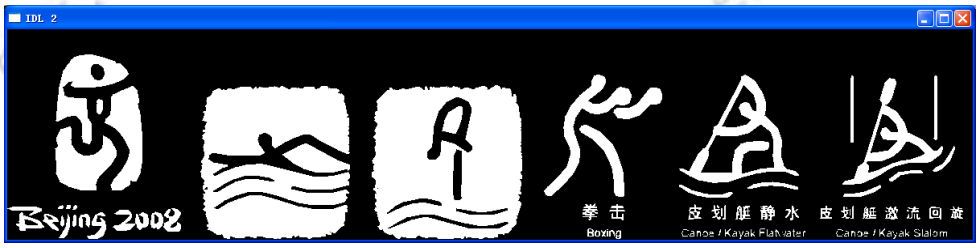
7.4.3 TrueType 字体

该字体也称为轮廓字体，这个字体由一系列的外形轮廓描述的，这些轮廓通过创建一系列的多边形来填充。IDL 有四种 TrueType 字体家族系列：Times、Helvetica、Courier 和 Symbol。TrueType 字体渲染需要花更长的时间，因为这种字体首先必须刻绘出来，然后创建相应的填充多边形，最后填充。TrueType 字体是 IDL 对象图形系统的一种缺省字体。

添加自定义 TrueType 字体的方法，将字体 ttf 文件拷贝到 IDL 安装目录的 \resource\fonts\tt 下，然后修改 ttfont.map 文件（ASCII 码文本），在最后一行添

加"beijing2008" Olympic_Beijing_Pictos.ttf 1.0 1.0，其中 Fontname-“beijing2008”是使用该字体时的标示符，Filename-“Olympic_Beijing_Pictos.ttf”是拷贝的字体文件名， 调用的方法见例子下面例子。

```
pro test_ttfont
  ;
  device, set_font ='beijing2008',/tt_font
  ;
  !p.font =1
  window,2,xsize = 1200,ysize =260
  xyouts,0,0,'012abc',charsize =20,/device
end
```



7.5 坐标系统

数据的图形表现都是通过数据坐标体系作为基本支架来完成。IDL 系统的坐标体系为 IDL 图形实现创造了很好的机制，主要包括：

- ✧ 数据坐标体系
- ✧ 设备坐标体系（显示屏幕、打印机纸张）
- ✧ 规一化坐标体系

无论采用任何的坐标系，显示数据的坐标都要统一，只有统一了坐标系才能正确的显示数据。

直接图形法下大部分的坐标系是数据坐标，就是根据像素点进行。对象图形法下的坐标系应用具体可见章节 9.8 坐标系统的统一。

7.6 图形图像显示

7.6.1 显示窗口

7.6.1.1 创建窗口

如果需要显示新的窗口则创建新的窗口用到 Window，该过程可以创建一个新的指定大小等窗口，并将窗口的索引指定给系统变量!D.Window。

```
IDL> window,8,xsize = 200,ysize=200
IDL> print,!d.window
```

8

7.6.1.2 选择窗口

对 IDL 的很多函数来说，函数自身不会自动选择哪个窗口，只会在当前窗口上面进行显示，Wset 可以指定当前显示的窗口。

```
IDL> wset,8
```

7.6.1.3 暴露指定窗口

对一个窗口来说，有两种状态，暴露和隐藏，也就是当前暴露的窗口只有一个。通过 WShow 可以进行窗口的暴露显示的指定。

```
IDL> wshow,8
```

7.6.1.4 删除指定窗口

对一个已经存在的窗口来说，可以用 Wdelete 进行窗口的删除。

```
IDL> wdelete,8
IDL> print,!d.window
          -1
```

7.6.2 绘制曲线

7.6.2.1 Plot 过程

Plot 过程用来绘制矢量数据线，除了数据本身外，并没有多少其它信息。然而，可用来定制和标注线画图，Plot 可以被 50 多种不同的关键字修饰来创建不同的曲线。下面对 Plot 常规的几种曲线绘制使用进行描述。

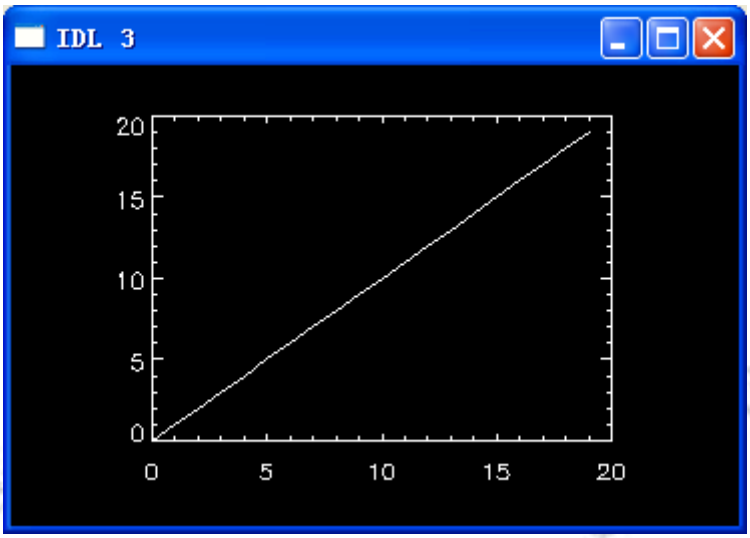
首先准备相关的数据，有下面几个：

```
SOCKEYE=[463, 459, 437, 433, 431, 433, 431, 428, 430, 431, 430]
COHO=[468, 461, 431, 430, 427, 425, 423, 420, 418, 421, 420]
CHINOOK=[514, 509, 495, 497, 497, 494, 493, 491, 492, 493, 493]
HUMPBACK=[467, 465, 449, 446, 445, 444, 443, 443, 443, 443, 445]
```

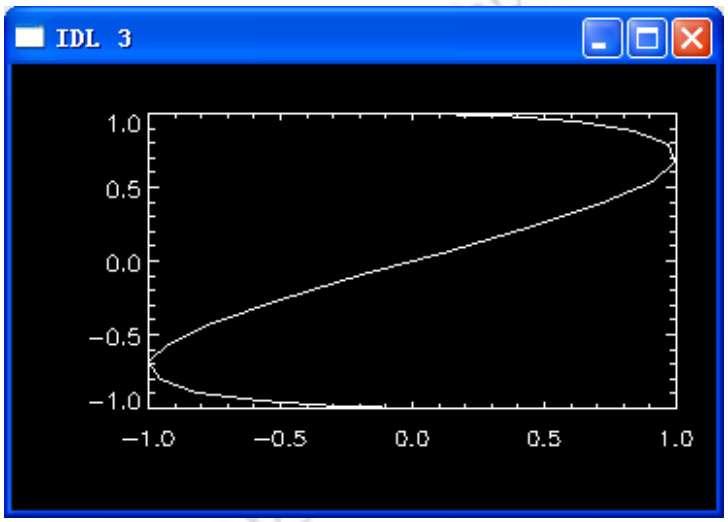
7.6.2.1.1 简单绘制线

简单的曲线绘制

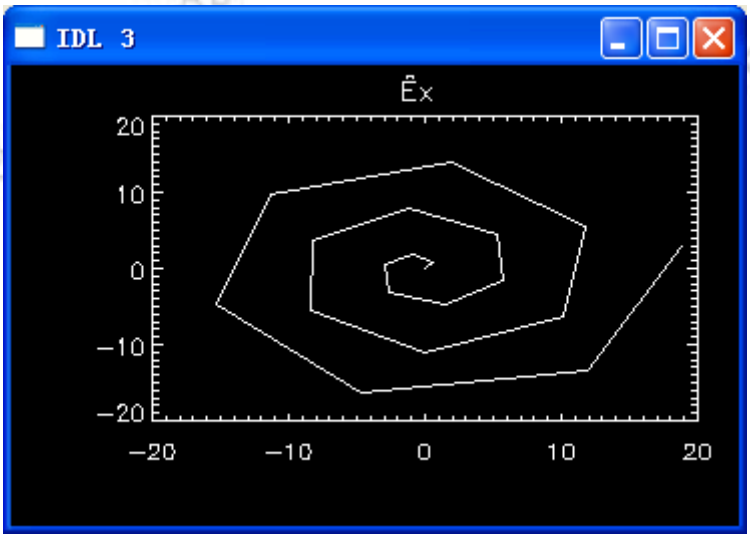
```
IDL> data = FINDGEN(20)
IDL> PLOT,data
```



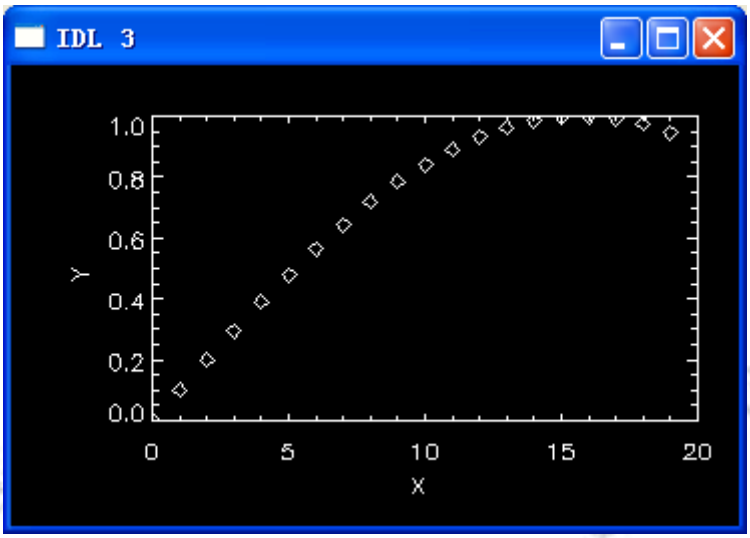
IDL> PLOT, SIN(data/3), COS(data/6)



IDL> PLOT, data, data, /POLAR, TITLE = '极射'

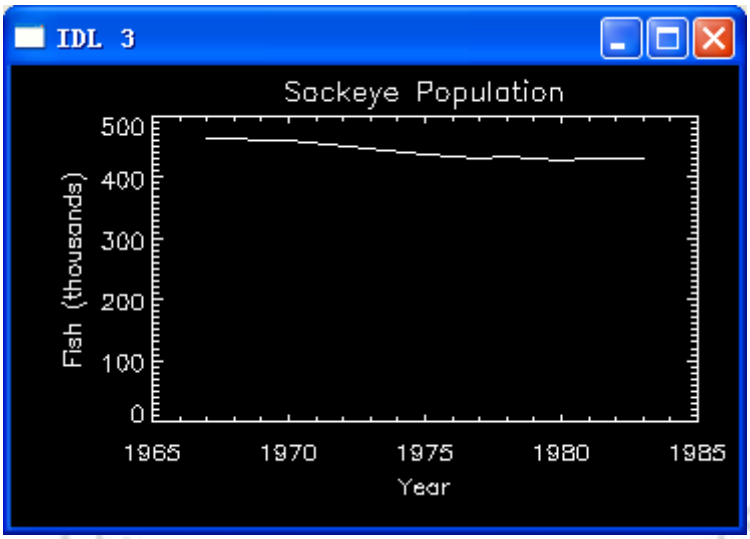


IDL> PLOT, SIN(data/10), PSYM=4, XTITLE='X ', YTITLE='Y '



7.6.2.1.2 点对绘制线

IDL> PLOT, YEAR, SOCKEYE, TITLE='Sockeye Population', XTITLE='Year', YTITLE='Fish (thousands)'



7.6.2.1.3 坐标轴风格

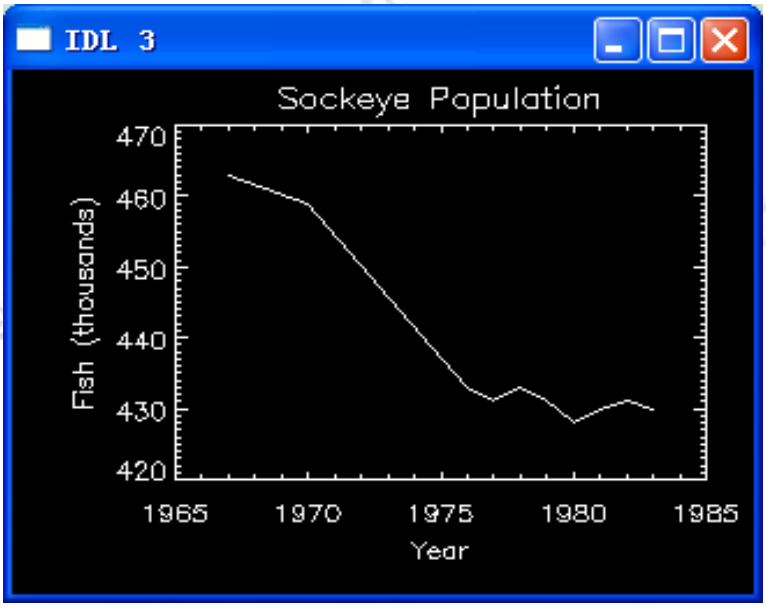
画线时坐标轴的显示风格可以用 XSTYLE、YSTYLE 和 ZSTYLE 关键字进行更改，下表列出了不同值代表的类型。

注意：这些属性值可以进行累加！

值	坐标轴的属性更改
1	精确的坐标轴范围
2	扩展坐标轴范围
4	不显示整个坐标轴
8	不显示外框（只画坐标轴）
16	屏蔽 Y 轴起始值为 0（只有 Y 轴有此属性）

PLOT, YEAR, SOCKEYE, $

```
TITLE='Sockeye Population', XTITLE = 'Year', $
YTITLE = 'Fish (thousands)', $
/YNOZERO, $
XRANGE = [1975, 1983], /XSTYLE
```



7.6.2.1.4 绘制标题

对曲线绘制添加标题则是启用 x[y,z]Title 关键字，例：

```
IDL> PLOT, YEAR, SOCKEYE, /YNOZERO, $
IDL>       TITLE='Sockeye Population', XTITLE = 'Year', $
IDL>       YTITLE = 'Fish (thousands)', XRANGE = [1975, 1983], /XSTYLE
```

7.6.2.1.5 绘制符号

假如想用符号代替线条表示数据，可以用关键字 PSYM 类解决。PSYM 通过不同的以确定不同的线画图符号。表列出了符号的索引编号。

```
PLOT, YEAR, SOCKEYE, /YNOZERO, PSYM = i
```

索引号	绘图符号
0	无符号，通过线条连接点
1	加号+
2	星号*
3	点.
4	菱形◇
5	三角形△
6	方形
7	X
8	自定义符号（可用 UserSym 来定义）
9	未用
10	直方图

7.6.2.1.6 绘制柱形图

对柱形图的绘制，可参考下面的程序。

```
XVAL = FINDGEN(4)/5. + .2
; Make a vector of scores from first year, corresponding to
; the name vector from above:
YVAL = [COHO[0], SOCKEYE[0], HUMPBACK[0], CHINOOK[0]]
; Make the axes with no data. Force x range to [0, 1],
; centering xval, which also contains the tick values.
; Force three tick intervals making four tick marks.
; Specify the tick names from the names vector:
PLOT, XVAL, YVAL, /YNOZERO, XRANGE = [0,1], XTICKV = XVAL, $
      XTICKS = 3, XTICKNAME = NAMES, /NODATA, $
      TITLE = 'Salmon Populations, 1967'
; Draw the boxes, centered over the tick marks.
; !Y.CRANGE[0] is the y value of the bottom x-axis.
FOR I = 0, 3 DO Ex_box, XVAL[I] - .08, !Y.CRANGE[0], $
      XVAL[I] + 0.08, YVAL[I], 128
;绘制填充矩形函数
PRO Ex_box, X0, Y0, X1, Y1, color
   ; Call POLYFILL:
   POLYFILL, [X0, X0, X1, X1], [Y0, Y1, Y1, Y0], COL = color
END
```

```
test_barplot
```

7.6.2.1.7 对数曲线

```
X = FLTARR(256)
; Make a step function. Array elements 80 through 120 are set to 1
X[80:120] = 1
; Make a filter:
FREQ = FINDGEN(256)
; Mak e the filter symmetrical about the value x = 128:
FREQ = FREQ < (256-FREQ)
; Second order Butterworth, cutoff frequency = 20.
FIL = 1./(1+(FREQ/20)^2)
; Plot with a logarithmic x-axis. Use exact axis range:
PLOT, /YLOG, FREQ, ABS(FFT(X,1)), $
        XTITLE = 'Relative Frequency', YTITLE = 'Power', $
        XSTYLE = 1
; Plot graph:
OPLOT, FREQ, FIL
```

7.6.2.1.8 多图形绘制

在系统的同一个窗口中绘制多个图像的关键是不同图像的位置，IDL 下可以通过系统变量!P.Multi 来进行控制，该系统变量的使用使在显示窗口内创建多个图形非常的快捷、方便。!P.Multi 由以下五个元素进行矢量定义。

!P.Multi[0]　　!P.Multi 的第一个元素包括剩下的要在显示窗口或者 PostScript 页上绘制的图形数目。这有点不直观，下面可以看到它是如何使用的。通常设置为 0，意思是，没有剩下要在显示窗口输出的图形。接下来的图形命令将删除显示的图形，并且开始绘制新的多个图形中的第一个。

!P.Multi[1]：规定了该页上图形的列数

!P.Multi[2]：规定了该页上图形的行数

!P.Multi[3]：规定了在 Z 方向上叠加的图形数目（仅适用已经建立了三维坐标系的情况下）

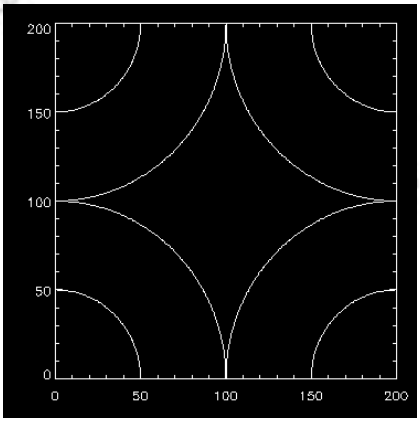
!P.Multi[4]：规定了是先按行显示图形（!P.Multi[4]=0），还是先按列显示图形（!P.Multi[4]=1）。

7.6.2.2 Contour 函数

7.6.2.2.1 简单等值线绘制

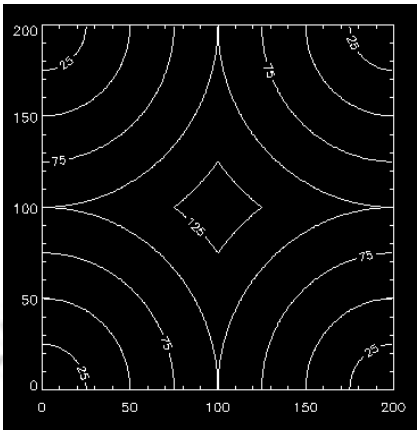
等值线的绘制可以分很多种，

```
IDL>zData = Dist(200)
IDL>CONTOUR,zData
```



7.6.2.2.2 等值线上标注

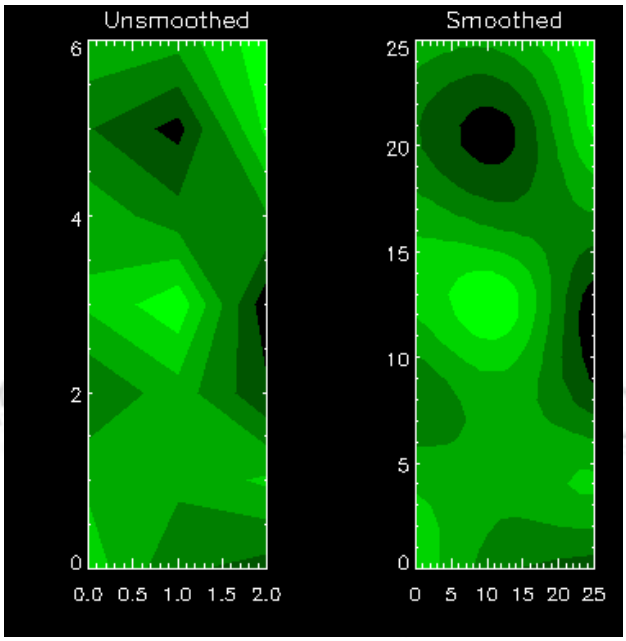
```
zData = Dist(200)
CONTOUR,zData, NLevels =5,/follow
```



7.6.2.2.3 等值线多边形平滑

插值函数的 MIN_CURVE_SURF 可以对规律或离散的网格点进行差值平滑，进而生成最小曲率表面或薄板样条曲面。

```
; 彩色显示
DEVICE, DECOMPOSED=0
;
Loadct,14
; 创建随机数
data = RANDOMU(seed, 3, 7)
;需要显示两个图像
!P.MULTI = [0,2,1]
;绘制原始数据等值线
CONTOUR, data, TITLE='Unsmoothed', $
        /Fill,NLevels = 5,C_Color = Indgen(5)+2
;绘制平滑后数据等值线
CONTOUR, Min_curve_surf(data), TITLE='Smoothed', $
        /Fill,NLevels = 5,C_Color = Indgen(5)+2
; 恢复系统窗口设置
!P.MULTI=0
```



7.6.2.2.4 等值线填充

Fill 和 C_FILL 控制填充方式，完全填充和线填充；

Nlevels：等值线的层数；

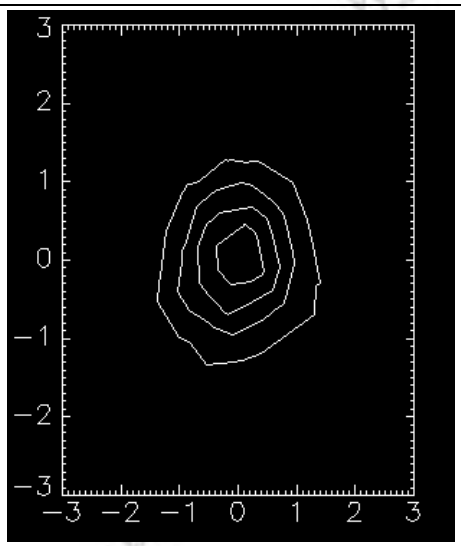
C_ORIENTATION-填充图形曲线的倾斜度（角度）。

```
;颜色填充
data = RANDOMU(seed, 7, 7)
; Create a basic, solid-color, filled CONTOUR plot
; with 6 evenly-spaced levels.
CONTOUR, data, NLEVELS=6, /FILL
; Overplot contour outlines:
CONTOUR, data, NLEVELS=6, /NOERASE
;线填充
WAIT,1
; Create a vector of orientations for the fill lines:
ANGLES = [0, 45, -45]
; Create a vector of colors to use:
C = [90, 180, 225, 270]
; Create contours filled with lines.
CONTOUR, data, NLEVELS=10, C_ORIENT=ANGLES, C_COLORS=C
; Overplot contour outlines:
CONTOUR, data, NLEVELS=10, /NOERASE
```

7.6.2.2.5 不规则网格创建三角形绘制等值线

对离散点，不需要先进行插值成网格数据，可以通过 triangulation 关键字直接进行等值线绘制。

```
; Make 50 normal X, Y points:
x = RANDOMN(seed, 50)
y = RANDOMN(seed, 50)
; Make the Gaussian:
Z = EXP(-(x^2 + y^2))
; Get triangulation:
;TRIANGULATE-根据离散点生成 Delaunay 三角网
TRIANGULATE, X, Y, tri
; 绘制等值线
CONTOUR, Z, X, Y, TRIANGULATION = tri
window,2,xsize =400,ysize =400
data= TRIGRID(x, y, z, tri)
;显示格网后图形
TVscl,congrid(data,400,400)
```



7.6.2.2.6 数字高程数据的等值线

先读取数字高程图像，然后直接在图像上面进行等值线绘制。

```
CD,!DIR+'/examples/data'
RESTORE, 'marbells.dat'
CONTOUR, elev
;
WINDOW,3,xsize =350,ysize =450,title ='Dem 数据'
TVSCL,elev
```

7.6.2.2.7 等值线标注

关键字：

C_LABELS：是否在等值线上进行文本标注；

LEVEL-每级等值线上对应的数值；

C_ANNOTATION-等值线的文本标注；

```
SEED = 20 & DATA = RANDOMU(SEED, 3, 8)
;设置
!P.MULTI=[0,2,1]
CONTOUR, LEVEL=[0.2, 0.5, 0.8], C_LABELS=[1, 1, 1], $
        C_CHARSIZE = 1.25, DATA
CONTOUR, LEVEL=[0.2, 0.5, 0.8], C_LABELS=[1, 1, 1], $
        C_ANNOTATION = ["Low", "Medium", "High"], $
        C_CHARSIZE = 1.25, DATA
!P.MULTI=0
```

7.6.2.2.8 绘制等值线叠加图像

```
DEVICE, DECOMPOSED=0
Loadct,14
CD,!DIR+'/examples/data'
RESTORE, 'marbells.dat'
;CONTOUR, elev
image = BYTSCL(elev, MIN=2658, MAX=4241)
new = REBIN(elev, 350/5, 450/5)
;!X.window-绘图起点归一化坐标，
;PLOT, CONTOUR, SHADE_SURF, SURFACE 有效
;!D.x_Vsize 绘图区域大小，像素
PX = !X.WINDOW * !D.X_VSIZE
PY = !Y.WINDOW * !D.Y_VSIZE
; Desired size of image in pixels.
SX = PX[1] - PX[0] + 1
SY = PY[1] - PY[0] + 1
; Display the image with its lower-left corner at
; the origin of the plot window and with its size
; scaled to fit the plot window.
TVSCL, Congrid(image, SX, SY), PX[0], PY[0]
CONTOUR, new,   LEVELS = 2750 + FINDGEN(6) * 250., $
      MAX_VALUE = 5000, XSTYLE = 1, YSTYLE = 1, $
      TITLE = 'Maroon Bells Region', $
      UBTITLE = '250 meter contours', $
      XTITLE = 'UTM Coordinates (KM)', /NOERASE
DEVICE, DECOMPOSED=1
```

7.6.2.2.9 等值线上绘制时间、日期

演示的是如何在等值线上绘制时间、日期。

```
number_samples = 37
;生成日期时间
date_time = Timegen(number_samples, UNITS = 'Seconds', $
      START = Julday(3, 30, 2000, 14, 59, 30))
angle = 10.*FINDGEN(number_samples)
```

```
temperature = BYTSCL(SIN(10.*!DTOR* $
        FINDGEN(number_samples)) # COS(!DTOR*angle))
;设置颜色
DEVICE, GET_DECOMPOSED=old_decomposed, DECOMPOSED = 0
Loadct, 5
;创建 label 的日期
date_label = Label_date(DATE_FORMAT = $
        ['%I:%S', '%H', '%D %M, %Y'])
CONTOUR, temperature, angle, date_time, $
        LEVELS = BYTSCL(INDGEN(8)), /XSTYLE, /YSTYLE, $
        C_COLORS = BYTSCL(INDGEN(8)), /FILL, $
        TITLE = 'Measured Temperature (degrees Celsius)', $
        XTITLE = 'Angle (degrees)', $
        YTITLE = 'Time (seconds)', $
        POSITION = [0.25, 0.2, 0.9, 0.9], $
        YTICKFORMAT= ['LABEL_DATE', 'LABEL_DATE', 'LABEL_DATE'], $
        YTICKUNITS = ['Time', 'Hour', 'Day'], $
        YTICKINTERVAL = 5, $
        YTICKLAYOUT = 2
; Applying contour lines over the original contour display.
CONTOUR, temperature, angle, date_time, /OVERPLOT, $
        LEVELS = BYTSCL(INDGEN(8))
;恢复
DEVICE, DECOMPOSED=old_decomposed
```

7.6.3 显示图像

显示图像可以用 TV，TVSCL 函数。

7.6.3.1 功能描述

TV 和 TVSCL 这两个命令几乎在各个方面都是一样的，包括能与之一起使用的关键字。仅仅在一个方面不同：TVScl 将图像数据调整为与 IDL 运行时所用颜色数目相适应的字节型数值。例如：如果在使用 IDL 时用 250 种颜色，则在图像显示之前 TVScl 将图像数据调整为 0～249 之间的字节型数值。

7.6.3.2 使用举例

1) 显示单幅及多幅灰度图形

```
D = BYTSCL(DIST(256)) & TV, D
TV, D, 0
TV, D, 1
```

这里的 0,1 是显示区域中对当前图像计算所得的索引，如对一个 512*512 的区域中，显示大小 128*128 的图像，索引依次为：

0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15

但如果这么写，TV,D,0,1，那 [0,1]代表的就是以左下角为[0,0]的像素坐标偏移量了。

```
TV,D,100,100
```

2) 显示彩色图像

对彩色图像，即三维数组[m,n,k]（其中一个为 3）的显示，使用 TV 时添加 True 关键字，其中 True 的值可参考下表。

TRUE 值	图像的维数
1	(*3*, *c*, *l*)
2	(*c*, *3*, l)
3	(c, l, 3)

参考下面例子源码

```
IDL> file = FILEPATH('rose.jpg', $
IDL>       SUBDIRECTORY = ['examples', 'data'])
IDL> image = READ_IMAGE(file)
IDL> TV, image
IDL> TV, image,/true
```

其中对 True 的值可以由下面的方法获得，

```
IDL> queryStatus = QUERY_IMAGE(file, imageInfo)
IDL> imageDims = SIZE(image, /DIMENSIONS)
IDL> imagesize = imageinfo.dimensions
IDL> interleaving = WHERE((imageDims NE imageSize[0]) AND $
IDL>       (imageDims NE imageSize[1])) + 1
```

3) 拉伸显示

Tvscl 的应用跟 tv 一样，只是 tvscl 做了一个拉伸处理，通过下面命令的对比就可以看出来。

```
IDL> TV, DIST(200),0
IDL> TVSCL, DIST(200),1
```

7.6.4 绘制曲面

在 IDL 程序中，二维的数组都可以用 Surface 命令生成一个曲面图。

7.6.4.1 曲面显示 surface

使用格式：surface,Z[,X,Y][关键字]，跟 plot 的使用方法基本类似，连续使用 surface 时原来的内容将被擦除。

首先，创建数据文件,通过 Dist()函数可以创建一个基于数字出现频率生成的数组，键入：

```
D = Dist(30)
; Plot a simple wire-mesh surface representation of D:
SURFACE, D
```

7.6.4.1.1 曲面图外观

显示一个曲面图的边缘。使用 Skirt 关键字来指定边缘该画到何处。试试下面命令：

```
D = Dist(30)
; Plot a simple wire-mesh surface representation of D:
SURFACE, D
; Create a wire-mesh plot of D with a title and a "skirt" around
; the edges of the dataset at Z=0:
        SURFACE, D, SKIRT=0.0, TITLE = 'Surface Plot', CHARSIZE = 2
```

7.6.4.1.1 曲面的旋转

在观察曲面图时希望能将曲面图旋转一个角度以便达到更好的视觉效果，曲面图可以用 Ax 关键字使其绕 X 轴或用 Az 关键字使其绕 X 轴旋转。当从轴上的正值向原点观察时，曲面图以逆时针方向，按某个角度值旋转。当 Az 和 Ax 关键字被忽略时其缺省值是 30 度。例如，使曲面图绕 Z 轴旋转 60 度，绕 X 轴旋转 35 度，则可键入：

```
Z = SHIFT(Dist(40), 20, 20)
; Make Gaussian with a 1/e width of 10:
Z = EXP(-(Z/10)^2)
; Call SURFACE to display plot:
SURFACE, Z
wait,1
;旋转
Surface,    Z,    Az=60,   Ax=35
```

7.6.4.2 阴影曲面显示 shade_surf

该函数用法跟 surface 基本一致，阴影的属性设置由 SET_SHADING 控制显示。

```
SET_SHADING, LIGHT = [1, 0, 0]
; Create a simple dataset:
D = Dist(40)
; Display the dataset as a light-source shaded surface:
SHADE_SURF, D, TITLE = 'Shaded Surface'
WAIT,0.7
; Create an array of shades to use:
S = SIN(D)
; Now create a new shaded surface that uses the array of shading
; values instead of the light source:
SHADE_SURF, D, SHADES = BYTSCL(S)
WAIT,0.7

Z = SHIFT(Dist(40), 20, 20)
; Make Gaussian with a 1/e width of 10:
Z = EXP(-(Z/10)^2)
SHADE_SURF, Z
```

7.6.5 相关函数参考

IDL 自带了非常丰富的功能函数，下面几个是收集的跟上面讲述的曲线、等值线相关的函数，大家可以根据帮助中的内容参考学习下。

IMAGE_CONT

FLOW3

PLOT_FIELD

VEL

VELOVECT

SHOW3

POLAR_CONTOUR

POLYSHADE

…

7.6.6 体数据显示

体数据的显示和操作可以通过交互式体绘制工具 XVOLUME 及 iTools 下的 iVolumne 来进行，XVOLUME 的用法如下

XVOLUME,voldata, [, 关键字]

体数据是一个真实三维数据，可以对其作切片处理，IDL 提供了切片处理的函数 EXTRACT_SLICE 和交互式切片提取 SLICER3。

函数 EXTRACT_SLICE 的调用格式为

Result = EXTRACT_SLICE(Voldata,Xsize,Ysize,Xcenter,Ycenter,Zcenter,Xrot, Yrot, Zrot,[,关键字])

Voldata：体数据的数据；

Xsize(Ysize)：获取切片数据的维数大小；

Xcenter(Ycenter,Zcenter)：切片数据的中心点坐标；

Xrot(Yrot,Zrot)：切片分别绕 X(Y、Z)坐标轴旋转的角度；

注意 SLICE3 的处理数据是指针。

例子源码程序文件 test_volumne

7.7 地图投影

IDL 提供交互式和静态地图显示功能，您可以使用 IMAP 的交互方式来显示地图，如果需要静态展示，可以使用地图投影的函数,以下主要讲述地图投影的函数。

7.7.1 功能描述

投影的相关函数有下面几个：

MAP_SET – 确立地图投影的类型和范围；

MAP_IMAGE – 根据当前投影对图像进行纠正并返回图像（图像比窗口大）；

MAP_PATCH – 根据当前投影对图像进行纠正并返回图像（图像比窗口小）；

MAP_CONTINENTS – 在图像上叠加绘制大陆边界线、政治边界、海岸线、河流，投影信息由 MAP_SET 来设置。

MAP_GRID – 绘制地图投影上的经纬线；

LL_ARC_DISTANCE – 给一个点的弧度距离和方位角计算该点的经纬度；

MAP_2POINTS – 函数返回一个二维的矢量的信息，可通过不同的关键字进行计算两点之间大圆的距离或方位，或两个点之间的大圆连线。

MAP_PROJ_FORWARD – 转换经纬度坐标到笛卡尔坐标；

MAP_PROJ_INVERSE – 转换地图坐标从笛卡尔坐标到经纬度坐标；

MAP_PROJ_IMAGE – 将图像从经纬度坐标纠正为指定的地图投影；

MAP_PROJ_INFO – 获得当前地图或投影的信息；

MAP_PROJ_INIT – 初始化地图投影，可调用 IDL 自身的投影或根据通用测绘转换包（GCTP）投影

7.7.2 举例

调试程序：Test_Map.pro

Test_map_image.pro

气象符号绘制 Test_wf_draw

bbs.esrichina-bj.cn

8 IDL 图像处理

当今社会，医疗行业、天文学、物理学、化学、遥感、制造和国防等许多的领域基本都要依靠图像进行资料的存储、显示。科学家和工程师们所面临的挑战就是如何从原始图像数据中快速提取有价值的信息。这也是图像处理的主要目的-转换图像信息。

IDL 自身包括了很多图像处理和分析的库程序，从而使得它的图像处理功能非常强大，通过使用 IDL，你可以轻松的进行图像文件的读取和信息的提取。

8.1 图像文件的输入输出

图像文件的输入输出操作可以见 4.3。

8.2 图像裁剪

剪裁图像是从原来的影像中提取矩形的感兴趣区域，需要对某一具体部分的地区形象特别关注，那么基于图像剪裁与图像放大，可以放大某一部分的影像。从技术上说就是获取数组的一部分。

```
IDL> world = READ_PNG(FILEPATH('avhrr.png', $
SUBDIRECTORY = ['examples', 'data']), R,G,B)
;建立一显示窗口
IDL> WINDOW, 0, XSIZE = worldSize[0], YSIZE = worldSize[1]
;设置显示模式
IDL> DEVICE, DECOMPOSED = 0, RETAIN = 2
;默认的颜色表
IDL> TVLCT, R, G, B
;获得图像的大小
IDL> worldSize = SIZE(world, /DIMENSIONS)
;建立同等大小的窗口
IDL> WINDOW, 0, XSIZE = worldSize[0], YSIZE = worldSize[1]
;显示图像
IDL> TV, world
;裁剪非洲区域
IDL> africa = world [312:475, 103:264]
;建立裁剪区域同大小的窗口
IDL> WINDOW, 2, XSIZE =(475-312 + 1), YSIZE =(264-103 + 1)
;显示裁剪后图像
IDL> TV, africa
```

8.3 图像的尺寸修改

图像数据的维数修改也就对图像矩阵进行修改，IDL 下面可以用 congrid 和 rebin 函数，系统的 lib 目录下有 expand 函数。

CONGRID：在处理一维或二维数组时默认是最近邻重采样方式，处理三维数组时默认是双线性内插值。

REBIN 默认是双线性内插。

如使用 CONGRID 进行数据的大小修改，

```
IDL> ;读取数据
IDL> file = FILEPATH('convec.dat', SUBDIRECTORY = ['examples', 'data'])
IDL> image = READ_BINARY(file, DATA_DIMS = [248, 248])
IDL> help,image
IMAGE                BYTE          = Array[248, 248]
IDL> magnifiedImg = CONGRID(image, 600, 600, /INTERP)
IDL> help,magnifiedImg
MAGNIFIEDIMG       BYTE          = Array[600, 600]
```

8.4 图像的位置变换

图像的位置变换包括图像的水平翻转、竖直翻转和旋转，分别由函数来实现，包括 shift、reverse、rotate 和 transpose。

SHIFT 函数用来对数据做平移处理。

用法：Result = SHIFT(Array, S1, ..., Sn)

其中 Si 是 i 维的平移个数，若 i=1 则按照右平移方式进行，若 i>1 则数据按照矩阵的行、列模式分别进行平移。

```
IDL> a = indgen(5)
IDL> print,a
       0       1       2       3       4
IDL> print,shift(a,1)
       4       0       1       2       3
;数组右移了一个元素。
IDL> print,shift(a,-1)
       1       2       3       4       0
;数组右移了负一个元素（左移）。
IDL> a = indgen(4,4)
IDL> print,shift(a,2)
      14      15       0       1
       2       3       4       5
       6       7       8       9
```

```
      10        11        12        13
;数组整体右移了两个元素
IDL> print,shift(a,1)
       15         0         1         2
        3         4         5         6
        7         8         9        10
       11        12        13        14
;数组右移了一个元素
IDL> print,shift(a,1,0)
        3         0         1         2
        7         4         5         6
       11         8         9        10
       15        12        13        14
;数组 X 方向右移了一个元素，Y 方向无偏移
IDL> print,shift(a,1,1)
       15        12        13        14
        3         0         1         2
        7         4         5         6
   11         8         9        10
;数组 X 方向右移了一个元素，Y 方向偏移一个元素。
```

8.5 图形的域变换

域变换是图像和信号处理技术，首先是把图像的数据空间转换到图像的频域空间，然后在其频域空间中对图像进行频域处理，然后将频域空间转换到图像的数据空间，从而实现图像处理的过程。图像变换的目的是为了图像处理的过程简化，可以用来对图像进行平滑、锐化、降噪和特征提取等。

IDL 下的域变换函数分下面三个部分快速傅里叶变换 FFT，小波变换 WTN，Hough 和 radon 变换 HOUGH、RADON，详细使用可参考 Help 中的章节：IDL Users' Guides > Image Processing > Transforming Between Domains。

8.6 图像的增强和滤波

8.6.1 图像增强

图像的增强包括字节拉伸和直方图拉伸。

8.6.1.1 字节拉伸

字节拉伸是指的对数据从一个范围拉伸到另一个数据范围内的处理，通过拉伸处理从而增强了对比度，IDL 提供了 BYTSCL 函数进行拉伸处理。

调用格式为：Result = BYTSCL(Array [, MAX=value] [, MIN=value] [, /NAN] [, TOP=value])

MAX：拉伸后范围的最大值；

MIN：拉伸后范围的最小值；

TOP：拉伸数据的最大值；

Test_bytescal

8.6.1.2 直方图均衡化

图像的直方图表示的是图像范围内的某个像素值的像素个数，即图像中像素的分布情况。直方图均衡化包括两种方法：标准直方图均衡和自适应直方图均衡。

标准直方图均衡：使用函数 HISTOGRAM()和 HIST_EQUAL()。参考例子 Test_EaualizingHist

自适应直方图均衡：使用函数 HISTOGRAM()和 ADAPT_HIST_EQUA()L。参考例子 Test_AdaptiveEqualizing

8.6.2 滤波

图像滤波在图像的平滑，噪声消除和边缘检测中都非常有用，IDL 提供了低通滤波、高通滤波、定向滤波及拉普拉斯滤波等滤波的方法。

低通滤波：在频率域中，低通信息畅通无阻，通过高频信息衰减的过程叫低通滤波。低通滤波在图像处理中抑制了反映灰度聚变边界特征的高频信息及孤立的噪声点，起到平滑图像去噪声的作用。

高通滤波：衰减或抑制低频分量，让高频分量畅通的滤波称为高通滤波。高通滤波使得图像得到了锐化处理。

定向滤波和拉普拉斯滤波：定向滤波和拉普拉斯滤波都为边缘检测打下一定的基础。

参考 Test_Filtering

8.7 图像平滑

IDL 中的图像平滑可以通过两种方式来进行，一是均值平滑，二是中值平滑。

8.7.1 均值平滑

由函数 Smooth 完成的，该函数调用格式为：

Result = SMOOTH(Array, Width [, /EDGE_TRUNCATE] [, MISSING=value] [, /NAN])

Width：可以是单个长度或数组，如给定宽度是 3*3，那么像素由它和周围八个像素值的平均值代替。

Edge_Truncate:该关键字可复制图像边缘附近的像素，以便实现整幅图像的平滑。如果不使用该关键字，图像边缘附近的像素仅仅是简单复制，而没有平滑。

8.7.2 中值平滑

由 Median 来实现，这个函数与 Smooth 不同之处在于 Median 命令计算相邻像素的中间值而不是平均值。这就有两个重要作用。第一，它能删除图像中的极端值。第二，它不使那些尺寸比邻域范围大的图像棱边或特征变模糊。

平滑的例子参考程序 test_Smooth。

8.8 图像边界检测

边界检测是图形处理中对图像的灰度或纹理特征进行不连续性或突变的一种检测技术。边界检测可以通过滤波的定向滤波和拉普拉斯滤波来实现，IDL 中还有下面的边界检测函数：EDGE_DOG、EMBOSS、LAPLACIAN、PREWITT、ROBERTS、SHIFT_DIFF 和 SOBEL。

应用可以参考例子 detecting_edges_doc.pro

8.9 图像形态学处理

数学形态学是由一组形态学的代数运算子组成的，它的基本运算有 4 个：膨胀（或扩张）、腐蚀（或侵蚀）、开启和闭合， 它们在二值图像和灰度图像中各有特点。基于这些基本运算还可推导和组合成各种数学形态学实用算法，用它们可以进行图像形状和结构的分析及处理，包括图像分割、特征抽取、边界检测、图像滤波、图像增强和恢复等。

IDL 中的形态学处理和分析方法主要包括：腐蚀、扩张、开、闭、细化、边界检测和形状分析等。

8.9.1 腐蚀和膨胀

腐蚀通常减小对象的尺寸，并且可以消除毛刺和去除噪声。膨胀通常增大对象的尺寸，并且可以填充空洞和断开区域，图像的腐蚀和膨胀可以使用函数ERODE()和 DILATE()。

参考程序:../examples/doc/image/ morpherodedilate.pro。

8.9.2 开运算

开运算是指先腐蚀再膨胀的运算，即开运算对图像先进行侵蚀运算，然后再进行膨胀运算。图像的开运算可以使用函数 MORPH_OPEN()来实现，其用法为：

格式：Result =MORPH_OPEN(Image, Structure [, /GRAY])

参考程序:../examples/doc/image/morphopenexample.pro。

8.9.3 闭运算

闭运算是指先膨胀再腐蚀的运算，即开运算对图像先进行膨胀运算，然后再进行侵蚀运算。图像的闭运算可以使用函数 MORPH_CLOSE()来实现，其用法为：

格式：Result =MORPH_CLOSE(Image, Structure [, /GRAY])

参考程序:../examples/doc/image/morphcloseexample.pro.

8.9.4 峰值检测

图像的峰值检测需要使用顶环运算 MORPH_TOPHAT（亦称峰值检测器）。首先通过对图像进行开运算，并从原始图像中去除开运算的结果，该运算可以从原始的灰度图像中提取最亮的像素。

参考程序:../examples/doc/image/morphtophatexample.pro.

8.9.5 分水岭边界检测

分水岭边界检测算法可以检测灰度图像的边界。通常在检测图像的边界之前，先对图像进行平滑处理，消除图像中噪声。

基于分水岭的边界检测技术可以使用函数 WATERSHED()来实现。该函数可以提取灰度图像的边界线。其用法为：

格式：Result = WATERSHED (Image [, CONNECTIVITY={4 | 8}])

参考程序:../examples/doc/image/watershedexample.pro.

8.9.6 图像识别

图像识别是指按照指定的要求识别图像中的某个对象。数学形态学操可以通过构造不同的结构元素来识别二值图像或者灰度图像中的特殊形状。

图像识别在数学形态学中可以使用 MORPH HITORMISS()函数，

格式：Result = MORPH_HITORMISS (Image, HitStructure, MissStructure)

参考程序:../examples/doc/image/morphhitormissexample.pro.

8.9.7 图像的梯度边界检测

梯度边界检测算法是根据图像中像素的梯度变来检测灰度图像的边界。

基于梯度的边界检测技术可以使用函数 MORPH_GRADIENT()来实现。该函数可以提取灰度图像的边界线。其用法为：

格式：Result = MORPH_GRADIENT (Image, Structure)

参考程序:../examples/doc/image/morphgradientex.pro.

8.9.8 图像细化

图像细化是指用原图像减去原图像与指定结构元素的击中击不中变换的图像处理过程。 一般情况下细化运算被反复地运行，最后只留下图像对象的像素宽度的线性表示。当没有像素可以从图像中去除时，细化运算停止。

图像的细化可以使用函数 MORPH_THIN()来实现。该函数可以提取图像的细化信息。其用法为：

格式：Result = MORPH_THIN (Image, HitStructure, MissStructure)

参考程序:../examples/doc/image/morphthinexample.pro

8.9.9 图像分割

图像分割是指根据图像处理和分析的需要，把图像中指定的对象从图像中分割出来的图象处理技术。

参考程序:../examples/doc/image/EDIT removebridges.pro.

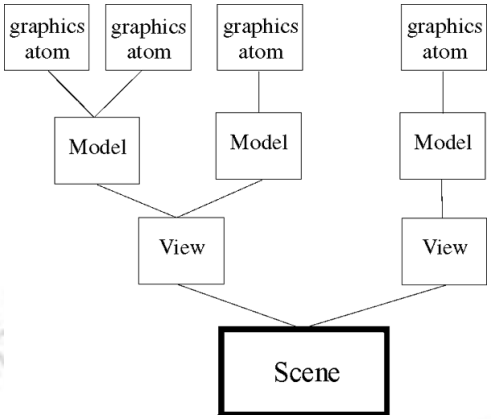
9 IDL 对象图形法

传统的开发方法开发的程序重用性、可修改性及可维护性都比较差，对象法是 IDL5.0 引入面向对象编程概念后出现的，面向对象的基础也就是对象类的使用。对象类允许编程者将数据和方法封装成一个包，称之为对象。一个对象类可以重复利用生成多个对象。

在 IDL 的虚拟图形构架体系中有一套对象显示结构对象，而且提供了很多的对象图形原子，利用这些对象和对象图形原子之间可以进行互相的加载，相互作用形成显示对象，灵活的运用这些图形原子就可以在视图中得到丰富多彩的图形或图像，甚至可以制作三维的动画。

9.1 对象层次结构

IDL 的对象层次是由一系列的对象类构成的，显示层次见图 9-1。其中（Graphics atom）是原子对象，Model(IDLgrModel)是显示的框架，View(IDLgrView)是视图场景对象，Scene（IDLgrScene）。



参考 RevolutionIDL

9.2 对象基本操作

9.2.1 创建对象

创建一个新对象的方法为：

object= OBJ_NEW([ObjectClassName [, Arg1……Argn]])

```
IDL> object = Obj_New("IDLgrModel")
IDL> help,object
OBJECT          OBJREF      = <ObjHeapVar795(IDLGRMODEL)>
```

9.2.2 方法调用

对象的方法调用的格式为 Object->Mathod

```
IDL> num    = object->Count()
IDL> print,num
           0
```

9.2.3 销毁对象

销毁对象用过程 Obj_Destroy,object

```
IDL> Obj_Destroy,object
IDL> help,object
OBJECT              OBJREF      = <ObjHeapVar795>
IDL> help,obj_valid(object)
<Expression>      BYTE          =      0
```

9.3 IDL 图形基本对象

9.3.1 IDLgrWindow 对象

IDLgrWindow 可以看做是一个独立的对象，也可以看做是 Widget_Draw。也就是说 IDLgrWindow 是一个显示的组件，介绍下它的 retain、renderer 参数。

9.3.1.1 retain 参数

该参数是用来定义图像的绘制时是否进行备份的，如为 0 则不进行备份，为 1 时操作系统进行备份，为 2 时 IDL 进行备份，如设置了备份则显示刷新会变慢。

9.3.1.2 renderer 参数

着色方式，如设置 renderer 为 0，则显示绘制时采取 OpenGL 硬件加速，为 1 则绘制采取软件加速。

例子：

```
IDL> owindow = Obj_new('idlgrwindow',dimension = [200,200])
IDL> oView = Obj_New('IDLgrView',color = [255,0,0])
IDL> oWindow->Draw,oView
```

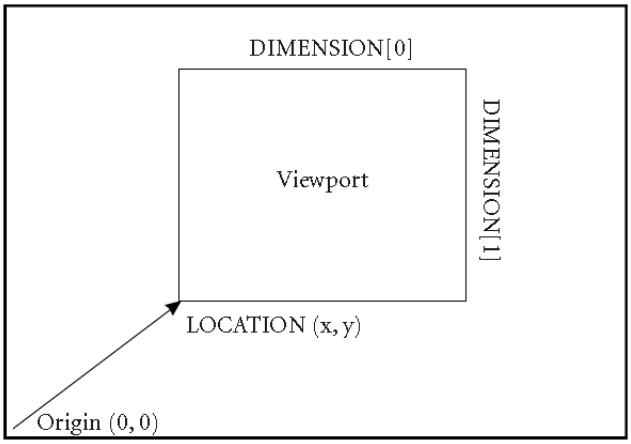
用其他 Windows 窗口遮挡后再查看该窗口，窗口显示的内容已经没有了。这是因为 retain 默认为 0，如设置为 2 即 IDL 自动备份则显示正常。

```
IDL> owindow = Obj_new('idlgrwindow',dimension = [200,200],retain =2)
IDL> oWindow->Draw,oView
```

9.3.2 IDLgrView 对象

IDLgrView 是一个矩形区域，图像元对象都可以在其中显示。常用关键字有 dimension、depth_cue、double、eye 、Location、projection、viewPlane_rect、transparent 及 zClip 等。

9.3.2.1 dimension、location 和 viewPlane_rect

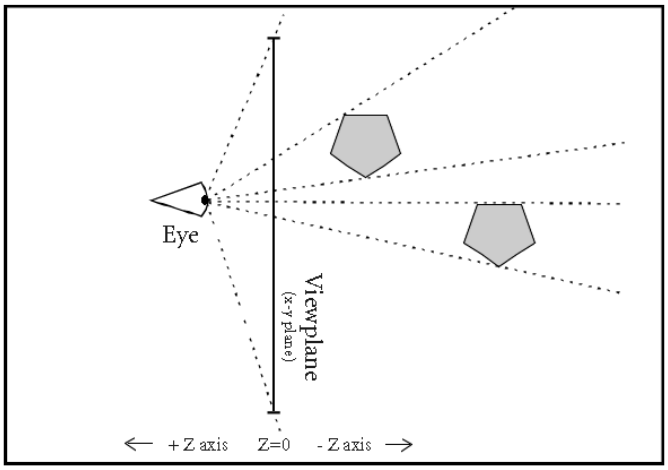


dimension 和 location 是 view 的维数和位置属性，默认单位是像素。而 viewPlane_rect 值是 viewPort（显示区域），可以这么形象的理解，viewPort 看做是一个照相机的镜头，我们可以通过变焦和相机移动对镜头显示的内容进行更改，但镜头的大小和相对于照相机的位置是固定的。

实例：Viewimagedemo

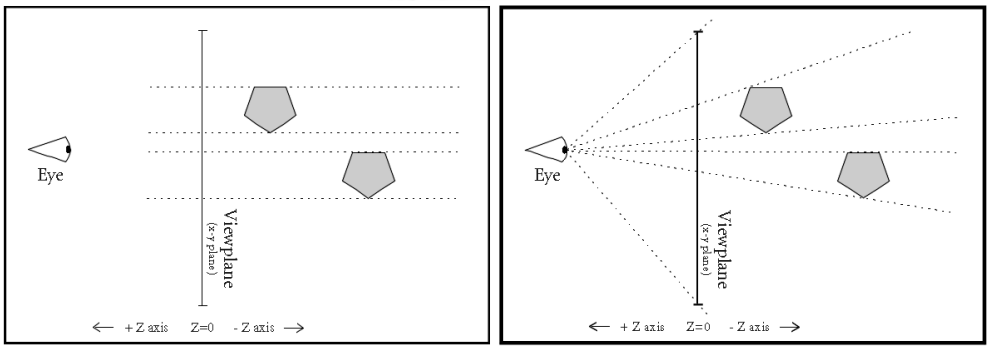
9.3.2.2 Eye 参数

Eye，也就是“眼睛”，可定义为视点的位置，示意图如下



9.3.2.3 projection

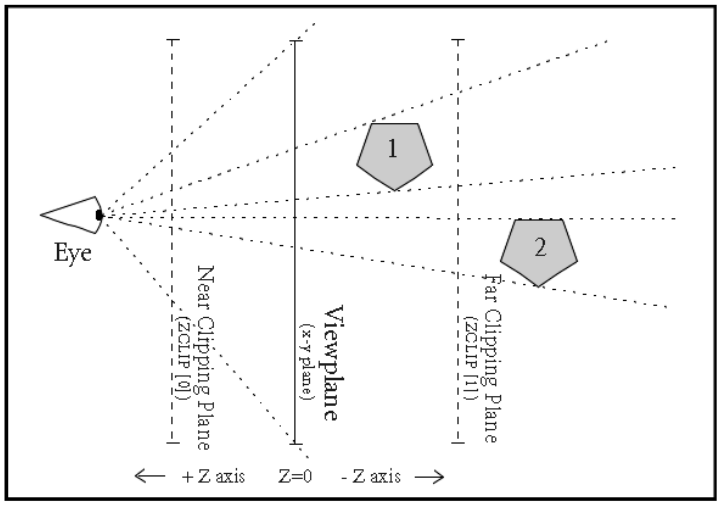
投影类型参数，这里的投影不是地图投影而是三维图像转二维显示时所用到的投影，分平行投影和透视投影两种。二者的定义可看下面两图



平行投影和透视投影

9.3.2.4 zClip 三维剪切面

该关键字定义了 view 范围内的 z 方向的可视距离。



如设置 ZCLIP 为 [2.0, -3.0]（图中的两竖虚线位置的值），结果会是 object1 可见，object2 不可见。

9.3.3 IDLgrModel 对象

IDLgrModel 是 IDL 对象图形法层次中的容器，它可以容纳所有类型的原子对象，然后才方便的对原子对象进行平移、旋转和放缩操作。

例子：显示一空矩形柱并进行缩放旋转显示。

```
IDL> oWindow = Obj_New('IDLgrWindow',dimension =[400,400],retain = 2)
```

```
IDL> MESH_OBJ, 3, Vertex_List, Polygon_List, $
IDL> Replicate(0.25, 48, 64), P4=0.5
IDL> oView = Obj_New('IDLgrView',viewPlane_Rect = [-.5,-.5,1,1],eye = 3,zClip = [2,-2])
IDL> oWindow->Draw,oView
IDL> oModel = Obj_New('IDLgrModel')
IDL> oPolygon = Obj_New('IDLgrPolygon',vertex_list,polygons = polygon_list)
IDL> oView->Add,oModel
IDL> oModel->Add,oPolygon
IDL> oWindow->Draw,oView
IDL> oModel->Rotate,[0,1,0],30
IDL> oWindow->Draw,oView
IDL> oPolygon->SetProperty,color = [200,200,0]
IDL> oWindow->Draw,oView
IDL> omodel->Scale, .8, 1.0, 1.0
IDL> oWindow->Draw,oView
```

9.3.4 原子对象

IDLgrAxis
IDLgrBuffer
IDLgrClipboard
IDLgrColorbar
IDLgrContour
IDLgrFilterChain
IDLgrFont
IDLgrImage
IDLgrLegend
IDLgrLight
IDLgrModel
IDLgrMPEG
IDLgrPalette
IDLgrPattern
IDLgrPlot
IDLgrPolygon
IDLgrPolyline
IDLgrPrinter
IDLgrROI
IDLgrROIGroup
IDLgrScene
IDLgrShader
IDLgrShaderBytscl
IDLgrShaderConvol3
IDLgrSurface
IDLgrSymbol
IDLgrTessellator
IDLgrText
IDLgrTextEdit
IDLgrView
IDLgrViewgroup
IDLgrVolume
IDLgrVRML
IDLgrWindow

9.4 二维图形显示

IDL 的对象图形法的操作性，使得二维图形的绘制及控制非常容易，我们分别讲述线、多边形、坐标轴及图像的显示及部分处理。

准备工作，先建立一个显示的体系结构，在后面的基本对象构建时都在该显示体系中显示。分别输入下面代码：

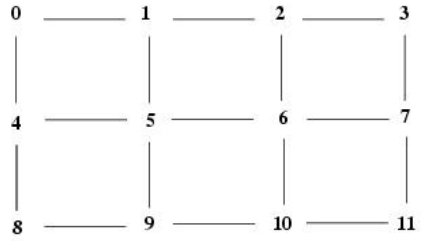
```
IDL> oWindow = Obj_New('IDLgrWindow',dimension =[400,400],retain = 2)
IDL> oView = Obj_New('IDLgrView',viewPlane_Rect = [-1,-1,2,2],eye = 3,zClip = [2,-2])
IDL> oModel = Obj_New('IDLgrModel')
IDL> oView->Add,oModel
IDL> oWindow->Draw,oView
```

9.4.1 线对象

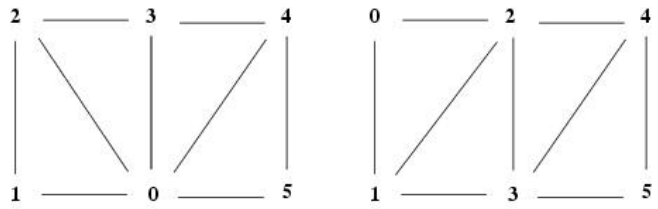
用 IDLgrPolyline 对象来进行线的绘制，定义线对象赋数据值并显示：

```
IDL> x = [-.5,.5]
IDL> y = [-.5,.5]
IDL> oPolyline = Obj_New('IDLgrPolyline',x,y)
IDL> oModel->Add,oPolyline
IDL> oWIndow->Draw,oView
;对线对象进行颜色的修改
IDL> oPolyline->Setproperty,color =[255,0,0]
IDL> oWIndow->Draw,oView
;进行线粗细的修改
IDL> oPolyline->Setproperty,thick = 5
IDL> oWIndow->Draw,oView
;创建一折线
IDL> data = fltarr(2,4)
IDL> data[0,*]= [-.5,-.5,.5,.5]
IDL> data[1,*]= [-.5,.5,.5,-.5]
IDL> oPolyline->Setproperty,data = data
IDL> oWIndow->Draw,oView
;自定义顶点连接顺序
;polylines 关键字，这里讲下四边形和三角形顶点顺序。
verts = [v0, v1, v2, v3, v4, v5, v6, v7, v8, v9, v10 ,v11]
polyline_list = [5, 0, 1, 5, 4, 0,$
        5, 1, 2 ,6, 5, 1, $
        5, 2, 3, 7, 6, 2, $
        5, 4, 5, 9, 8, 4, $
        5, 5, 6, 10, 9, 5, $
      5, 6, 7, 11, 10, 6]
;或
```

```
polygon_list =[4, 0, 1, 5, 4, $
        4, 1, 2 ,6, 5, $
        4, 2, 3, 7, 6, $
        4, 4, 5, 9, 8, $
        4, 5, 6, 10, 9, $
      4, 6, 7, 11, 10]
```




```
verts = [v0, v1, v2, v3, v4, v5]
polyline_list[4, 0, 1, 2, 0 ,$
        3, 0, 2, 3, 0 ,$
        3, 0, 3, 4, 0 ,$
      3, 0, 4, 5,0])
;或
polygon_list =[3, 0, 1, 2, $
        3, 0, 2, 3, $
        3, 0, 3, 4, $
        3, 0, 4, 5])
```




```
;交叉图形
IDL> polylines = [4,0,1,3,2]
IDL> oWIndow->Draw,oView
;正方形
IDL> polylines = [5,0,1,2,3,0]
IDL> oPolyline->Setproperty,polylines = polylines
IDL> oWIndow->Draw,oView
;如绘制一个×，则连接关系为：
polylines = [2,0,2,2,1,3]
```

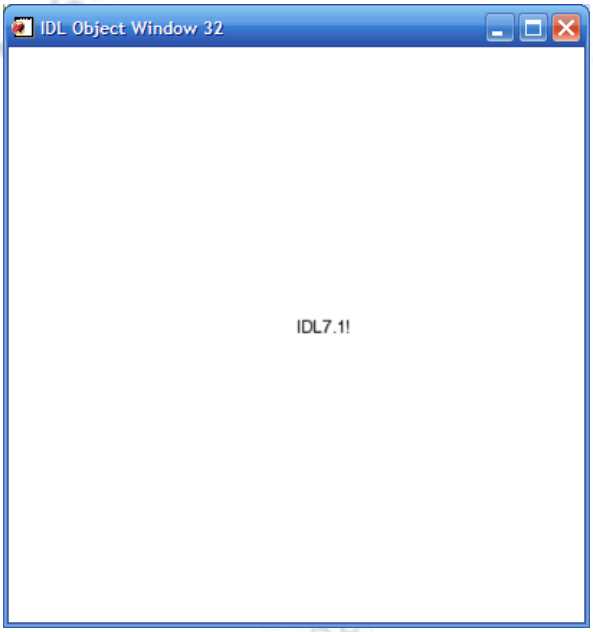
9.4.2 多边形

多边形是 IDLgrPolygon 对象，当然了，IDLgrPolyline 的东西也是可以用 IDLgrPolyline 来代替的。可参考章节 9.5。

9.4.3 文字

该对象是创建文本对象，并可以在任意空间位置、任意平面上，产生任意大小、粗细、走向的文字。常用关键字：Color，align，vertical_align，baseline，updir，char_dimensions， onglass， location， recompute_dimension， palette， strings，envable_formatting。

```
IDL> oText = Obj_New('IDLgrText','IDL7.1!')
IDL> oWindow->Draw,oView
IDL> oText = Obj_New('IDLgrText','IDL7.1!')
IDL> oModel->Add,oText
IDL> oWindow->Draw,oView
```

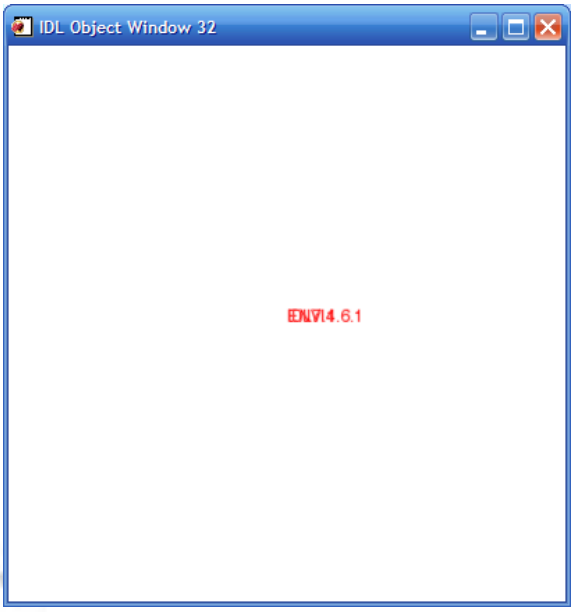


颜色修改 color

```
IDL> oText->SetProperty, color=[255,0,0]
IDL> oWindow->Draw,oView
```

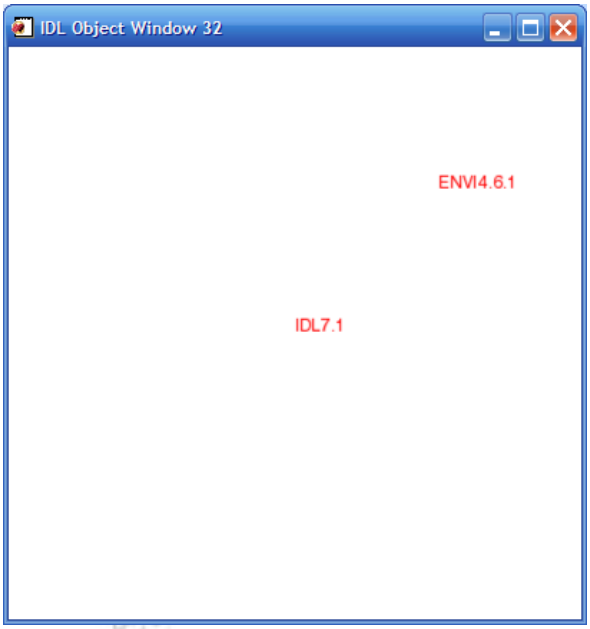
多文字时

```
IDL> oText->SetProperty, strings = ['IDL7.1','ENVI4.6.1']
IDL> oWindow->Draw,oView
```



IDL> oText->SetProperty, location = [[0,0],[.5,.5]]

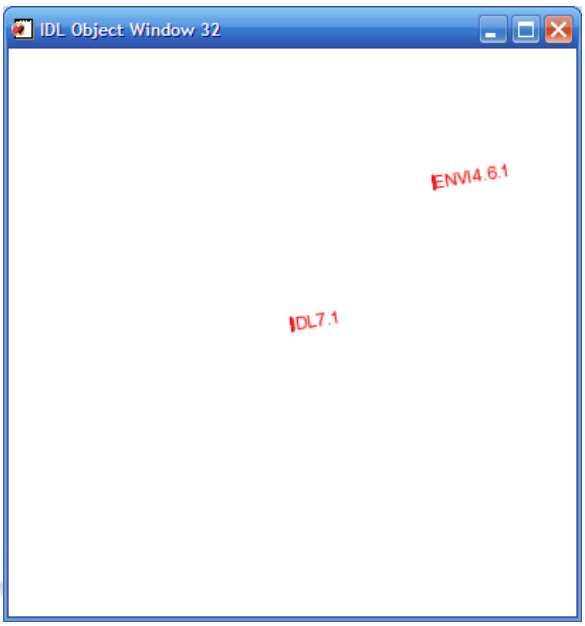
IDL> oWindow->Draw,oView



baseline 定义的是文字的基准方向，由二维或三维的矢量组成。

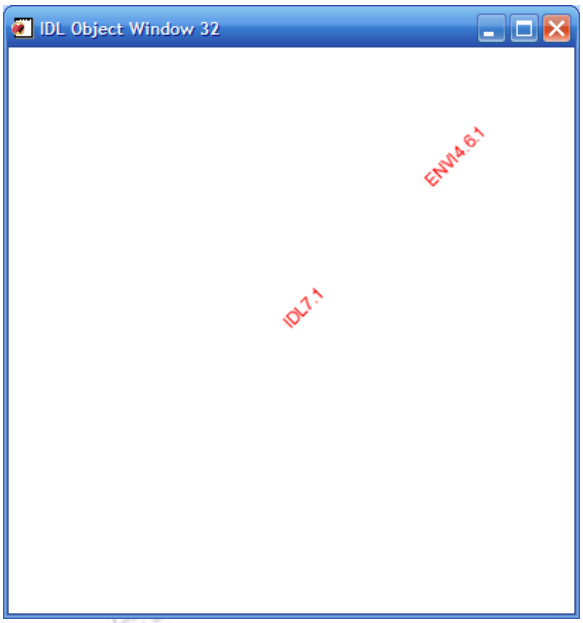
IDL> oText->SetProperty, BASELINE =[6,1]

IDL> oWindow->Draw,oView



IDL> oText->SetProperty, BASELINE =[1,1]

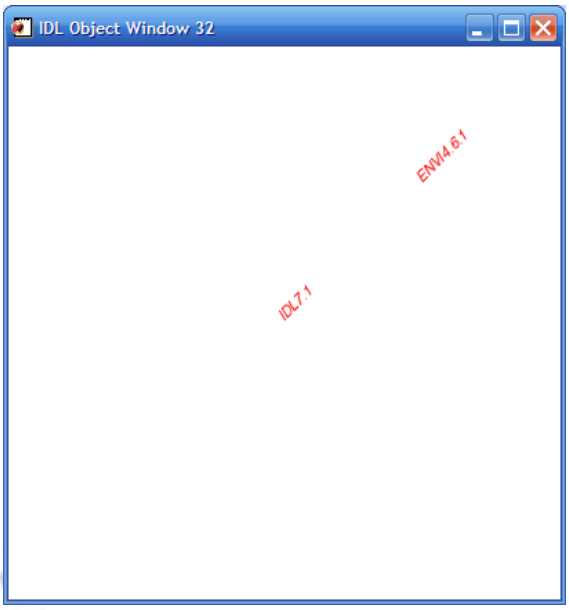
IDL> oWindow->Draw,oView



IDL> oText->SetProperty, BASELINE =[1,1,.8]

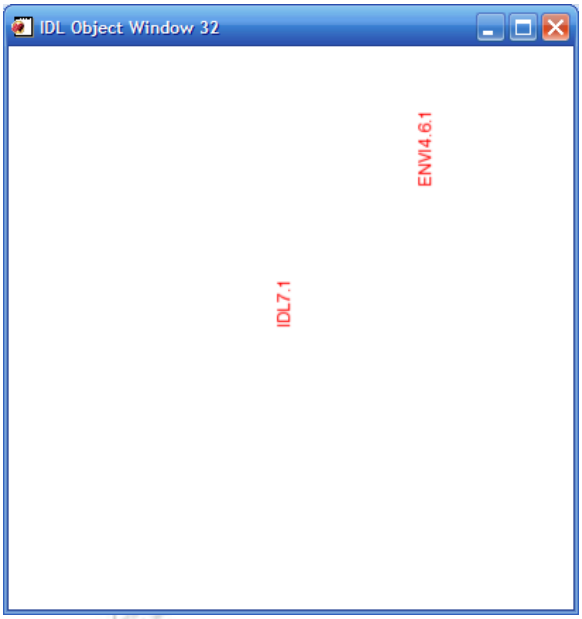
IDL> oWindow->Draw,oView

bbs.esrichina-bj.cn



IDL> oText->SetProperty, BASELINE =[0,1]

IDL> oWindow->Draw,oView

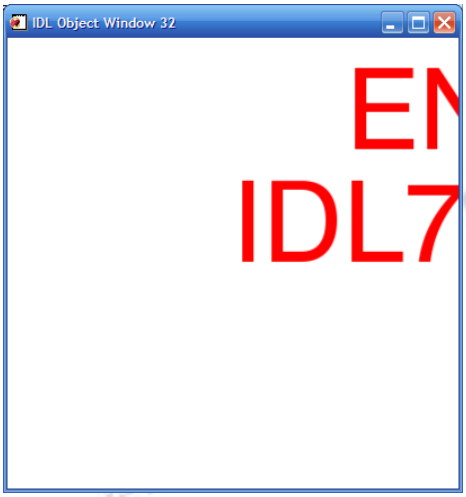


char_dimension 指的是文字占的大小

IDL> oText->SetProperty,baseline=[1,0],updir =[0,1], CHAR_DIMENSION = [.5,.5]

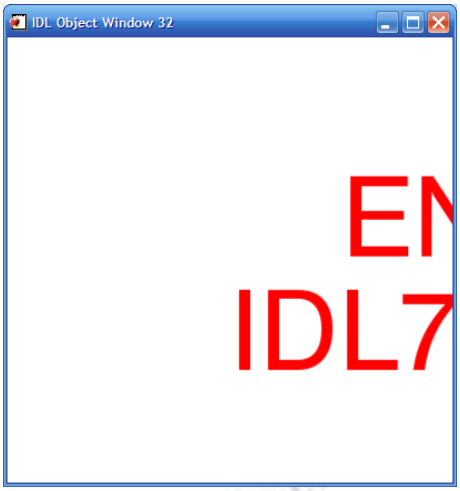
IDL> oWindow->Draw,oView

bbs.esrichina-bj.cn



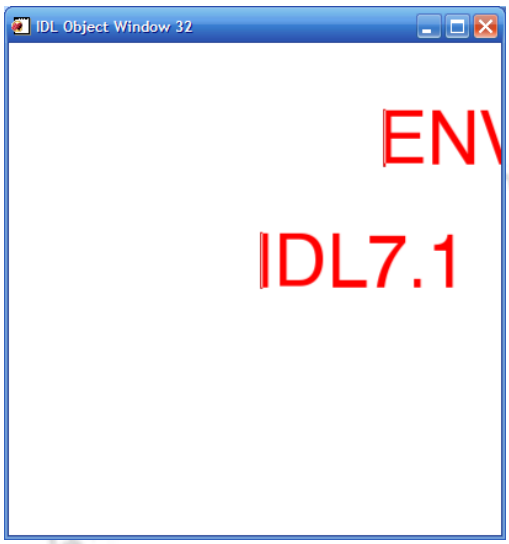
VERTICAL_ALIGNMENT 是指文字在 y 方向的位置，为 0 则文字的下边靠近基准面，为 1 则文字的顶部靠近基准面。

```
IDL> oText->SetProperty,VERTICAL_ALIGNMENT=1
IDL> oWindow->Draw,oView
```



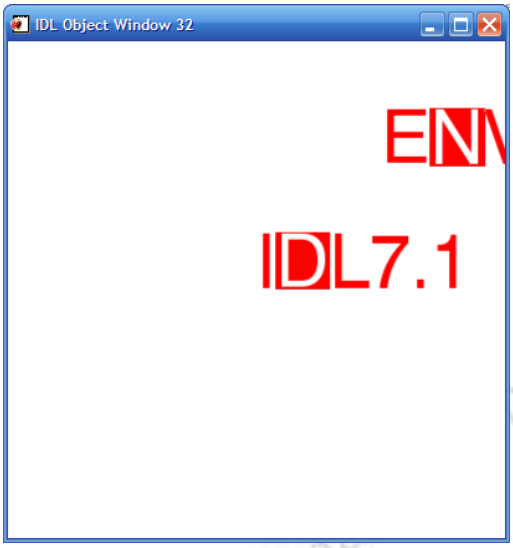
DRAW_CURSOR 关键字，设置是否文字中间有那个光标的，相关的关键字还有 SELECTION_START 和 SELECTION_LENGTH。

```
IDL> oText->SetProperty,draw_cursor =1
IDL> oWindow->Draw,oView
```



IDL> oText->SetProperty, SELECTION_START=1,SELECTION_LENGTH =1

IDL> oWindow->Draw,oView



字体 IDLgrFont 对象

字体的基础知识在直接图形法中已经进行了介绍，对象法下的字体获取跟直接图形法下的字体获取有些类似，如下：IDLgrWindow-> GetFontnames

	直接图形法	对象图形法
获取字体列表	DEVICE, GET_FONTNAMES=fontnames, SET_FONT='*'	IDLgrWindow-> GetFontnames
获取字体个数	DEVICE, GET_FONTNUM=numfonts, SET_FONT='*'	
设置字体	DEVICE, SET_FONT='字体名', /TT_FONT	oFont=Obj_New('IDLgrFont', 字体名)

		oText->Setproperty,font = oFont
字体查看工具	EFONT，SHOWFONT	

创建一个新的字体对象用下面的命令：

```
IDL> myFont = OBJ_NEW('IDLgrFont', 'times', SIZE=20)
IDL> oText->Setproperty, strings ='IDL7.0',font = oFont
IDL> oWindow->Draw,oView
```

9.4.4 坐标轴

坐标轴类用 IDLgrAxis，

Major：标号个数；

Minor：标号中间的间隔；

TextPos：坐标轴的文字位置控制；

轴类型	TextPos=0	TextPos =1
X	标注于 X 轴下方	标注于 X 轴上方
Y	标注于 Y 轴左方	标注于 Y 轴右方
Z	标注于 Z 轴左方，与 X 轴负方向一致	标注于 Z 轴右方，与 X 轴正方向一致；

TickDir 坐标轴标注的控制；与 TextPos 的控制一致；

TextAlign：坐标轴标注的位置控制，由一个二维矢量[xPos,yPos]来控制，其中 xPos 和 yPos 的范围均为[0,1]

TextBaseline=[-1,0,0]

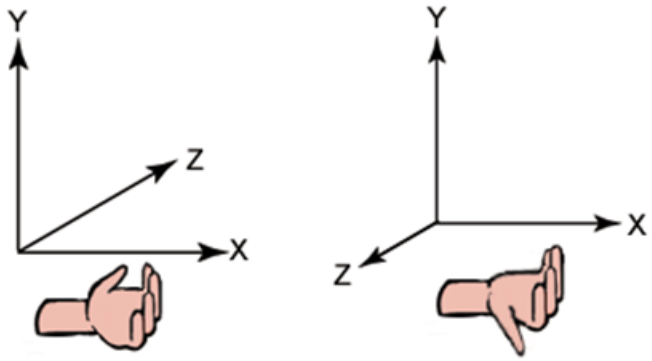
参考程序 test_IDLgrAxis

9.4.5 曲线

跟直接图形法的 plot 功能类似，对象法下也有 IDLgrPlot 对象绘制曲线，详细可参考：test_IDLgrPlot

9.5 三维图形显示

IDL 的三维坐标系使用的是右手笛卡尔坐标系，与 Microsoft Direct3D 的左手坐标系相区别，示意图如下。



左手坐标系　　右手坐标系

参考程序：Test_3D.pro

9.6　坐标统一

了解了对象图形法，下面专门介绍下坐标系统，因如要显示图像图形则需要先明确一个显示范围。不论你使用什么坐标系，要明确一个基准的坐标系统，所有的对象对该范围进行相对坐标定位，这样就能相应的显示了。

如不同对象有不同的坐标体系，则需要进行坐标转换。

为了方便的进行坐标变换，IDL 下的对象自带坐标变换的方法。每个对象通过坐标变换可以修改对象的位置和放缩，这点在坐标变换时非常有用。

如系统 View 的坐标使用归一化坐标，对象的数据范围在该范围内则能完整的显示，否则需要进行坐标转换。

X[YZ]COORD_CONV 的格式为 [s0, s1],，通过其转换后的坐标为 v= s0 + s1 * oriV，如果转换为归一化坐标，也就是[0，1]，如数据的最大最小值为 Xmax 和 Xmin，那么转换因子应为[-Xmin/(Xmax-Xmin),Xmax/(Xmax-Xmin)]。

9.7　对象的交互操作

IDL 的原子对象正确的显示之后，我们需要对其进行一系列的操作，比如选择操作，鼠标位置的数据值获取等操作。

运行分析实例程序 sel_obj.pro，SURF_TRACK.pro

9.8　编写类

9.8.1 编写新对象类

创建对象类，创建于类名一致的结构体，IDL 中规定创建的格式如下：

PRO ClassName__DEFINE

struct = { ClassName, data1:value1, … , dataN:valueN }

END

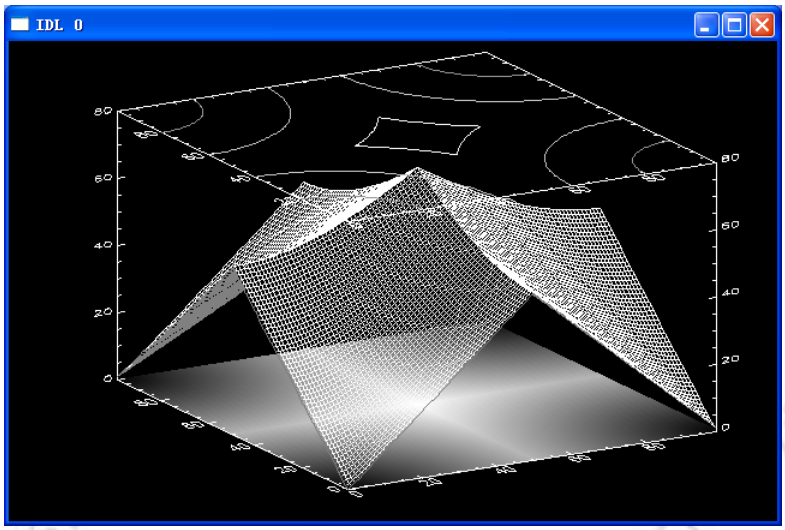
注意：类的名称 ClassName 和过程的名称 ClassName__DEFINE 的关系，必须严格按照上述格式。即：过程的名称是由类的名称、两个下划线“__”和“DEFINE”组合而成。

用户在设计方法的时候，IDL 提供了一个方便控制对象本身的隐含对象，即：对象 SELF。

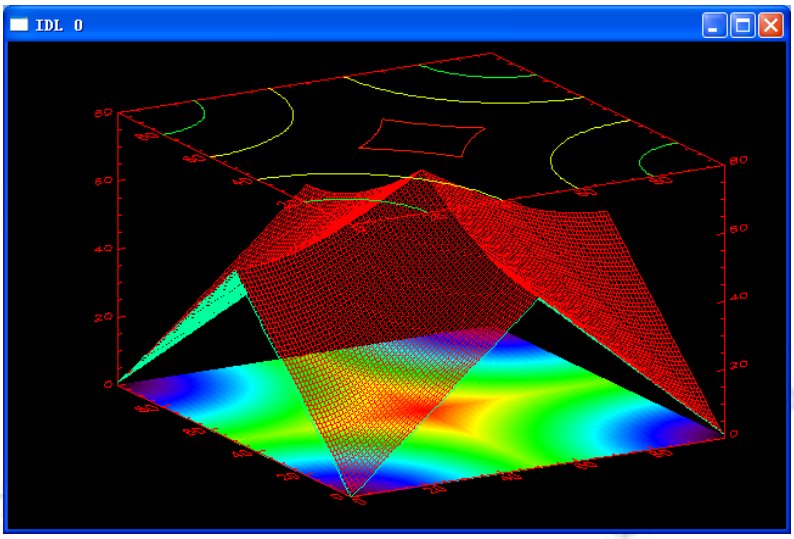
分析程序 show3obj__define.pro。

通过下面的语句对该类进行初始化及调用。

```
IDL> oShow3 = OBJ_NEW ('SHOW3OBJ', DIST (100) )
IDL> oShow3->display
% Compiled module: SHOW3.
% Compiled module: POLYWARP.
```




```
IDL> oShow3 -> Loadcolor, 13
% LOADCT: Loading table RAINBOW
```




```
IDL> Obj_Destroy,oshow3
SHOW3OBJ object successfully destroyed.
```

9.8.2 编写继承类

根据类的继承性，可以使用基本类（即基类）创建子类。创建子类可以使用包含基类和指定结构变量，并且具有如下格式的过程：

格式：PRO ClassName__DEFINE

```
    struct={ClassName,INHERITS BaseClass, $
            data1:value1,…,dataN:valueN}
    END
```

参考例子 AnnotateDemo。

9.9 地图投影

对象法下的地图投影可以参考例子，Projectiondemo，该例子的学习参考部分有以下几个方面：

1) 对象图形法系统的编写方式；
2) 坐标系统的统一；
3) 图像、矢量叠加显示；
4) 图像及矢量的平移及缩放操作；
5) 继承类的使用；
6) 投影参数选择与投影创建；

7) 图像及矢量的投影转换；

bbs.esrichina-bj.cn

10 智能化工具 iTools

10.1 iTools 概述

iTools 是 IDL6.0 以后版本提供的功能强大的交互式分析工具。IDL 初学者可以利用这套智能工具快捷轻松地实现数据读入，数据分析和数据可视化，并且可以输出制作令人眩目的精美图形图像。有经验的 IDL 程序员使用它，可以减少程序开发工作量，甚至可以在 iTools 框架的基础上创建全新的可视化应用环境。

iTools(intelligent tools)的出现是 IDL 走向易用性和易扩展性的重要一步。我们可以引用 Mohsen Farid 博士的一句话给予 iTools 一个精确描述：“iTools are excellent interactive visualization tools that allow complete interactivity and control over datasets. iTools are well suited for exploratory type work。”

以往版本的 IDL 对于数据的可视化工作对采用直接图形法和命令行的方式，虽然简单快捷并且得到了一批用户的认可，但是面对计算机应用水平的不断提高，更多的用户选择基于鼠标即可完成的简单操作，而不是需要键入指令，iTools 则应运而生。iTools 是由一批工具箱组成，其中包括 iPlot、iSurface、iContour、iImage、iVolume 等等。IDL 使用者可以通过鼠标进行拖放即可完成数据的可视化以及分析工作。

iTools 目前提供了下面几种智能工具，如下表所示：

iTools 预定义智能工具一览表

工具名称	工具用途
iPlot	用于二维和三维图形。
iSurface	曲面分析，用于2D数组数据和不规则取样数据。
iContour	等值线、等高线的绘制、展示和操作。
iImage	图像数据的显示、研究，感兴趣图像区域的圈定和基本操作。
iVolume	对体数据的剖析和展示工具
iMap	用户显示矢量和具有地理坐标的影像数据
iVector	用于显示和编辑矢量数据

10.2 iTools 的使用

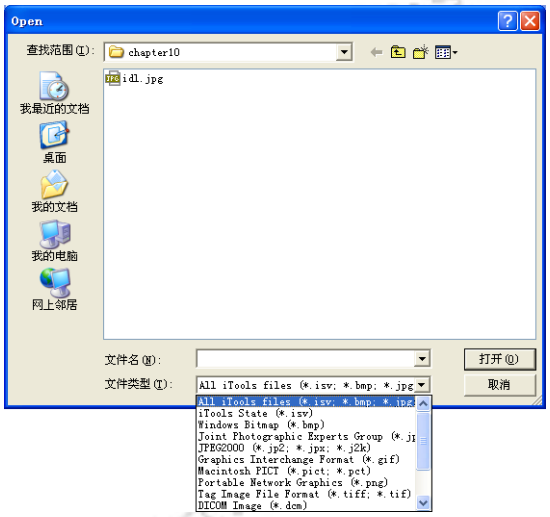
10.2.1 数据导入和导出

iTools 作为智能化工具，可以方便的进行可视化，但可视化的前提是有数据，数据的导入有下面方式。

10.2.1.1 数据导入

10.2.1.1.1 使用文件打开

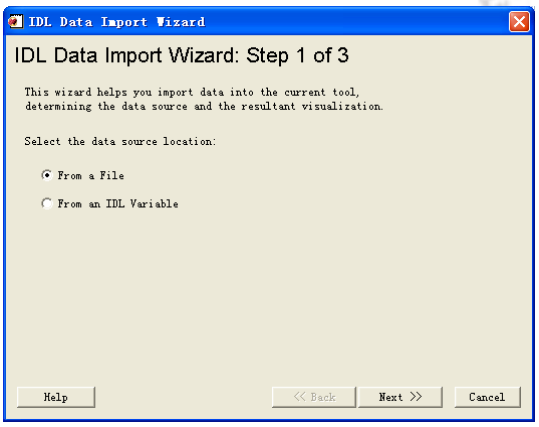
运行 iimage 启动可视化界面，然后在菜单[File]中选择 Open，弹出如下对话框，选择需要打开的文件即可。



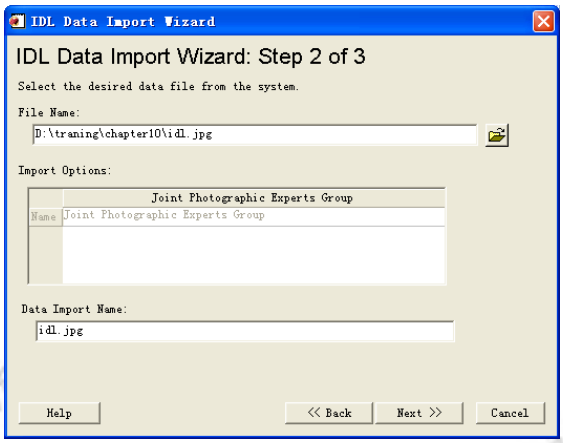
10.2.1.1.2 使用导入向导

点击菜单[FILE]→Import，弹出导入向导。

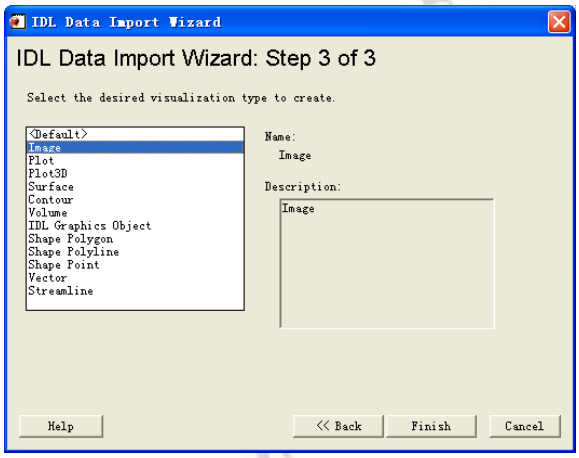
1) 从文件导入



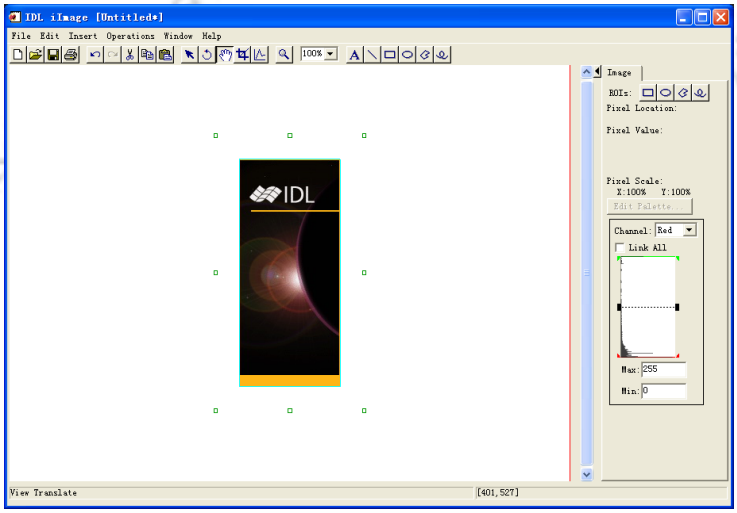
（1） 选择输入文件名



（2） 根据导入数据选择进行可视化的类型

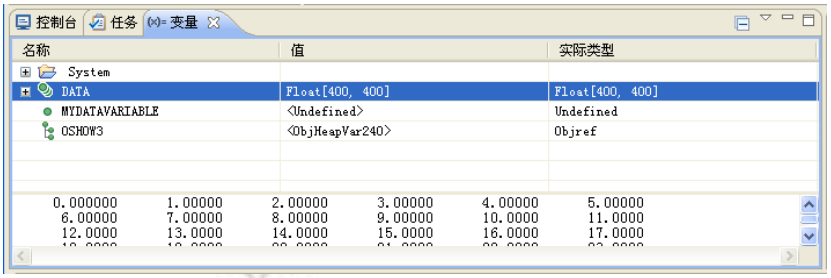


（3） 点击 Finish 完成数据导入

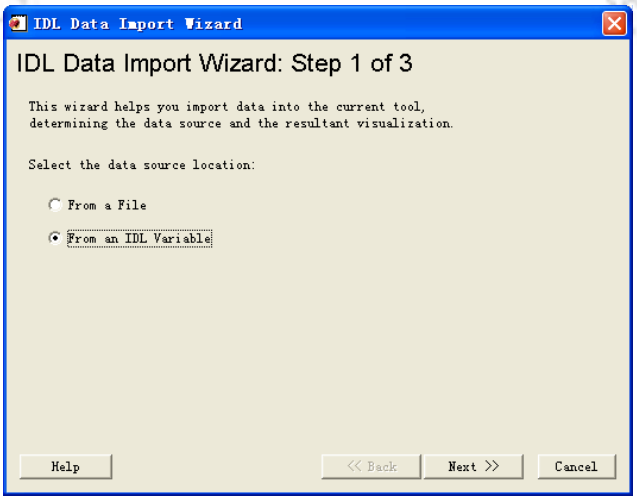


2) 从 IDL 下的变量导入

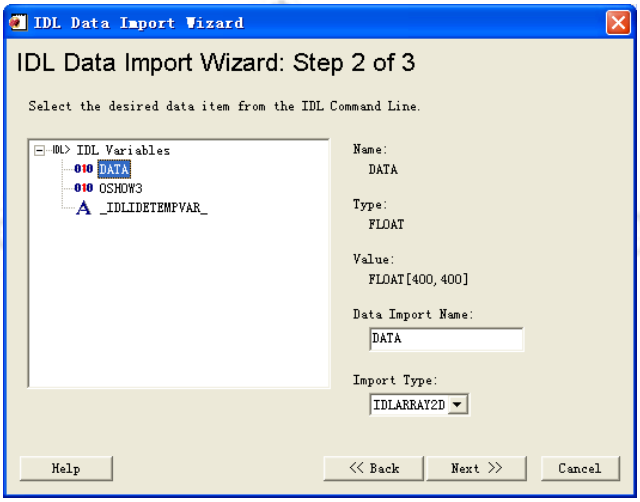
（1） 当前 IDL 系统下的变量列表



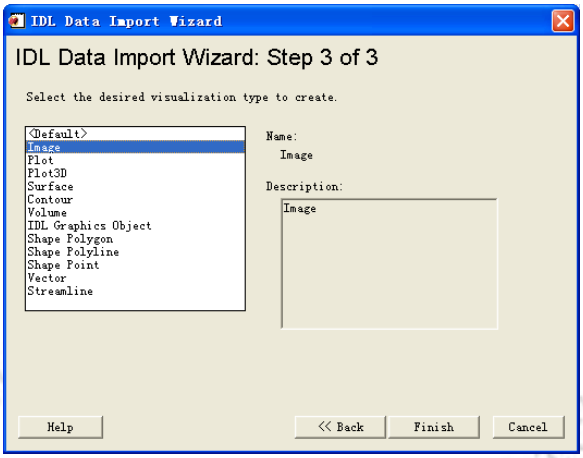
（2） 根据向导选择从变量导入



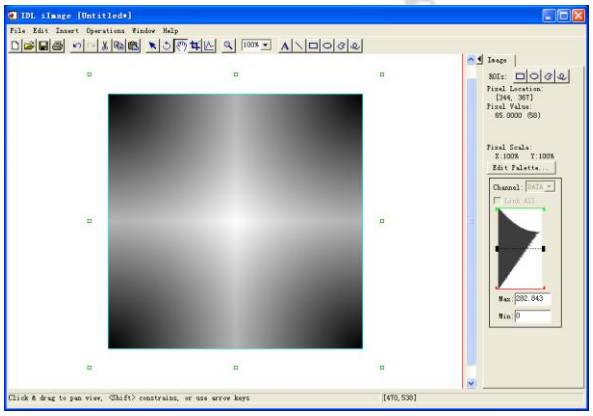
（3） 选择要导入的变量名



（4） 根据导入数据选择进行可视化的类型



（5） 点击 Finish 完成数据导入

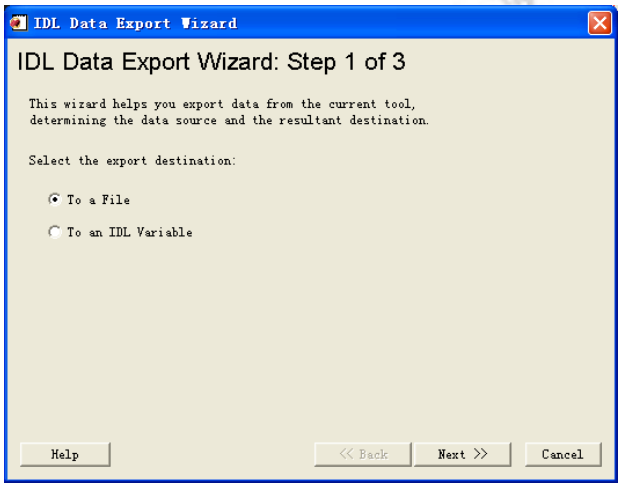


10.2.1.2 数据导出

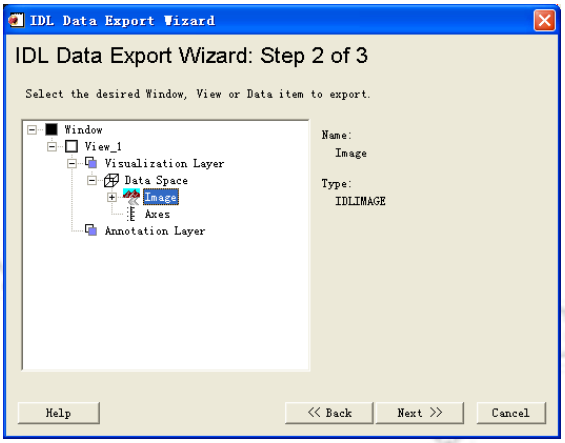
点击菜单[FILE]→Export，弹出导入向导。

1) 导出为文件

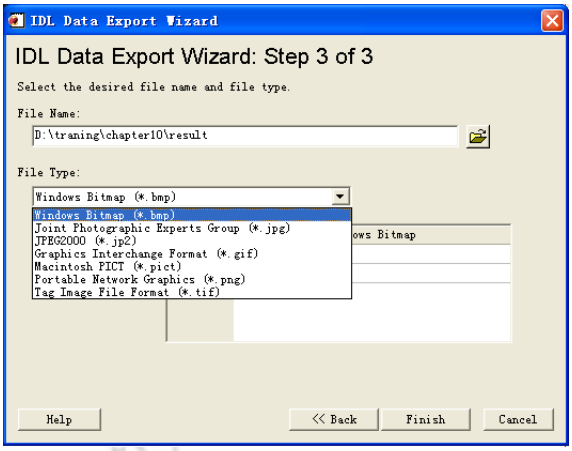
（1） 选择导出为文件



（2） 选择导出的变量



（3） 设置导出的文件名

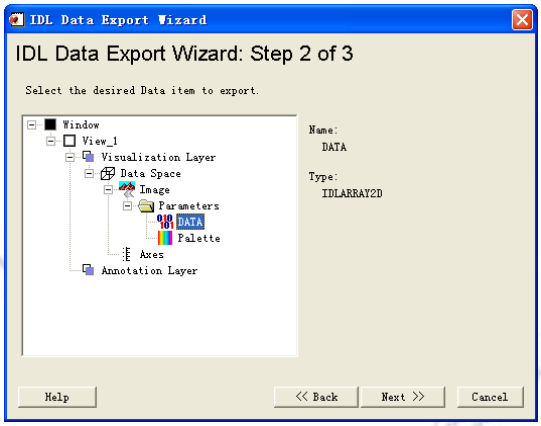


2) 导出为 IDL 下变量

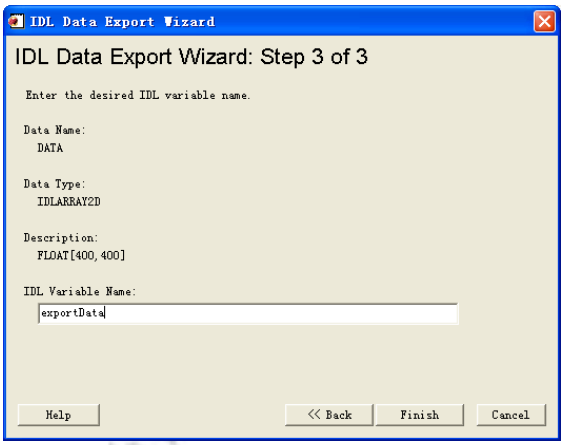
（1） 选择导出为文件



（2） 选择导出的变量

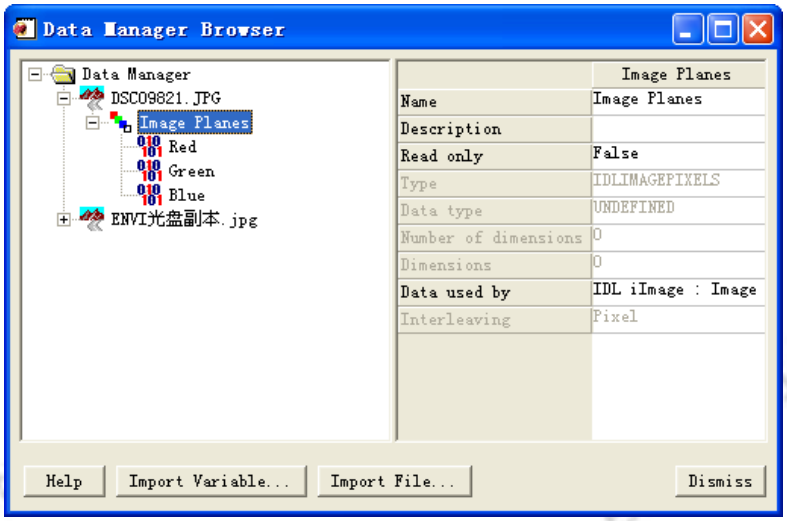


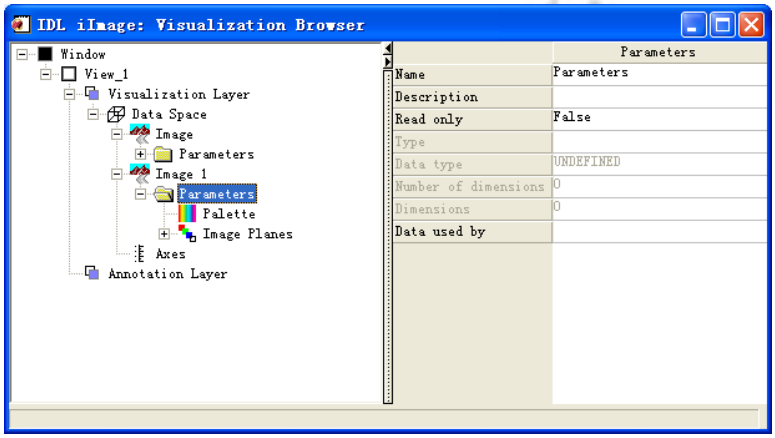
（3） 设置导出后变量名



10.2.2 数据和对象属性的查看

从菜单[Window]下选择[Data Manager]及[Visualization Browser]，分别弹出数据属性窗口和对象属性窗口。





10.2.3 显示操作

10.2.3.1 对象选择

选用工具栏上的;

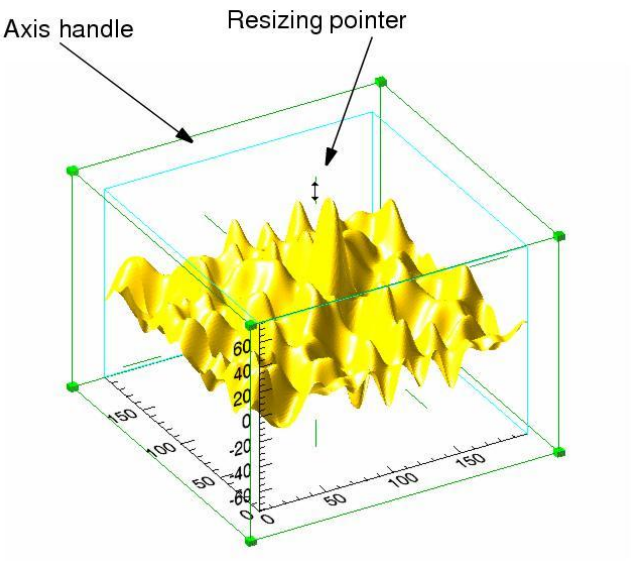
10.2.3.2 撤销/重做

点击工具栏上的或菜单下[Edit]→[UnDo/ReDo];

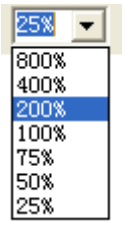
10.2.3.3 旋转

点击工具栏上进行旋转处理;

10.2.3.4 数据缩放



10.2.3.5 视图缩放

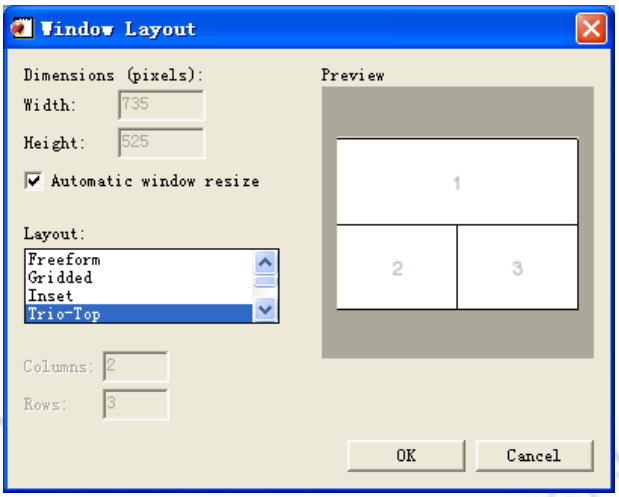


10.2.3.6 平移

选择工具栏上的按钮进入鼠标平移状态。

10.2.3.7 视图窗口控制

选择菜单[Window]→[Window Layout]在下面窗口界面中进行个数及位置的设定。



10.2.4 添加标注

10.2.4.1 添加图例

选择菜单 Insert → New Legend.

10.2.4.2 添加标注

1) 文字标注

选择工具栏上的进行文字的标注；

常用快捷键：
Ctrl+Enter 回车换行；
Ctrl+U 输入上标文字；
Ctrl+D 输入下标文字；
Ctrl+N 输入正常状态文字；

2) 线标注

选择工具栏上的进行线绘制；

3) 矩形标注

选择工具栏上的进行矩形绘制；

4) 椭圆标注

选择工具栏上的进行椭圆绘制；

5) 多边形标注

选择工具栏上的进行多边形绘制；

6) 自由画线标注

选择工具栏上的进行自由画线绘制；

10.2.5 公共操作

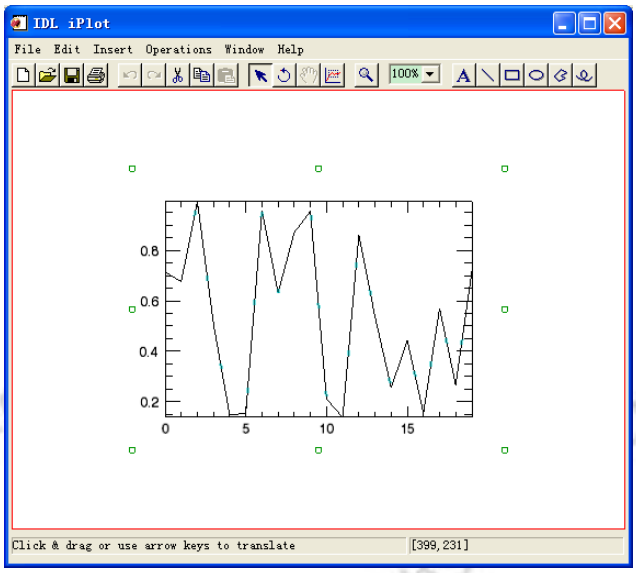
- 文件菜单
- 旋转操作
- 图像变换
- 滤波
- 形态学处理
- 直方图
- 统计分析
- ROI 操作
- 打印操作

10.2.6 iPlot 使用

使用 iPlot 可以进行二维或三维曲线的绘制，直接在命令行中输入 iPlot 即可出现 iPlot 主界面，可直接创建数据或章节 10.2.1 的数据导入功能导入数据。

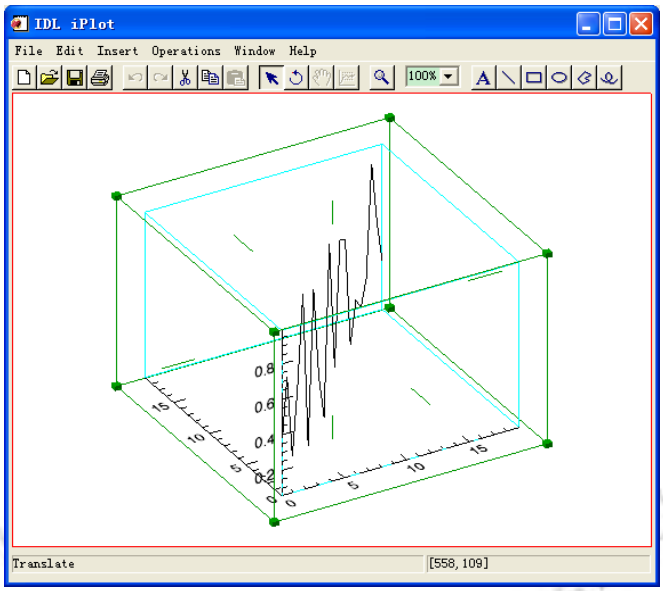
10.2.6.1 创建二维的曲线

```
IDL> iPlot, RANDOMU(seed, 20)
```



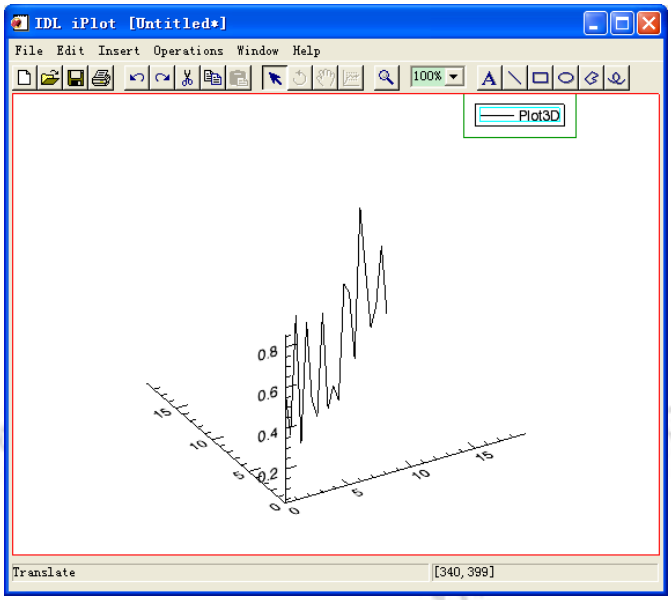
10.2.6.2 创建三维曲线

```
IDL> iPlot, FINDGEN(20), FINDGEN(20), RANDOMU(seed, 20)
```



10.2.6.3 添加图例

选择 Insert → New Legend



10.2.6.4 添加颜色棒

Insert → Colorbar

10.2.6.5 曲线滤波处理

首先选择曲线，然后进行滤波处理，下面列出不同滤波方法对应的操作。

平滑滤波：Operations → Filter → Smooth

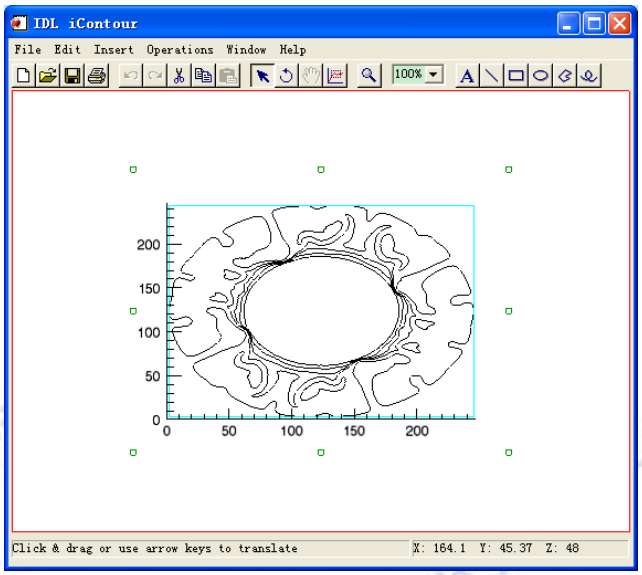
卷积滤波：Operations → Filter → Convolution

中值滤波：Operations → Filter → Median.

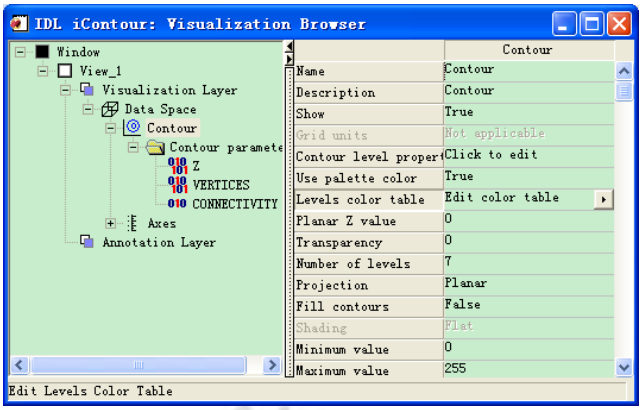
10.2.7iContour 使用

10.2.7.1 交互式绘制等值线

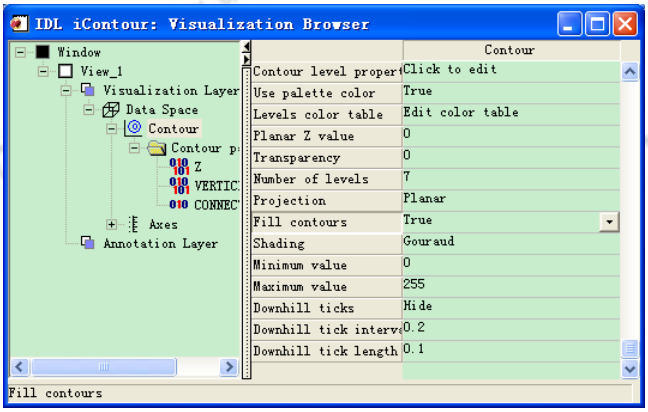
```
IDL> file = FILEPATH('convec.dat', SUBDIRECTORY = ['examples', 'data'])
IDL> data = READ_BINARY(file, DATA_DIMS = [248, 248])
% Compiled module: READ_BINARY.
IDL> iContour, data
```



10.2.7.2 颜色设置



填充设置



10.2.7.3 等值线滤波处理

首先选择曲线，然后进行滤波处理，下面列出不同滤波方法对应的操作。

平滑滤波：Operations → Filter → Smooth

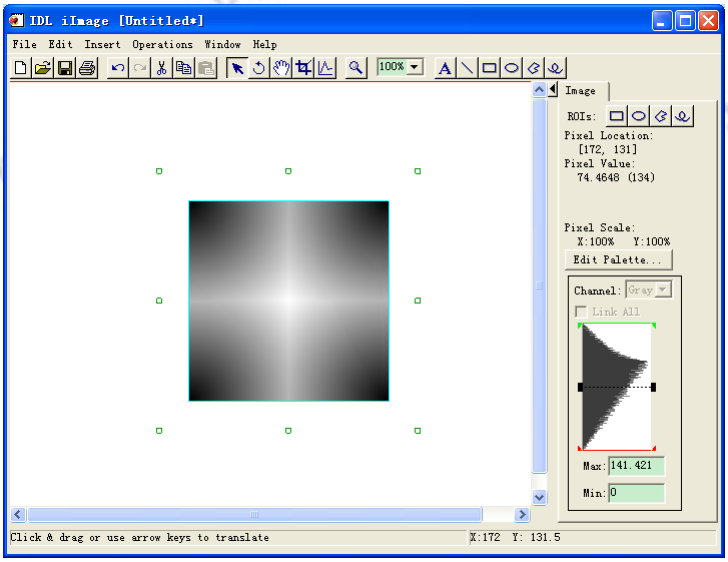
卷积滤波：Operations → Filter → Convolution

中值滤波：Operations → Filter → Median..

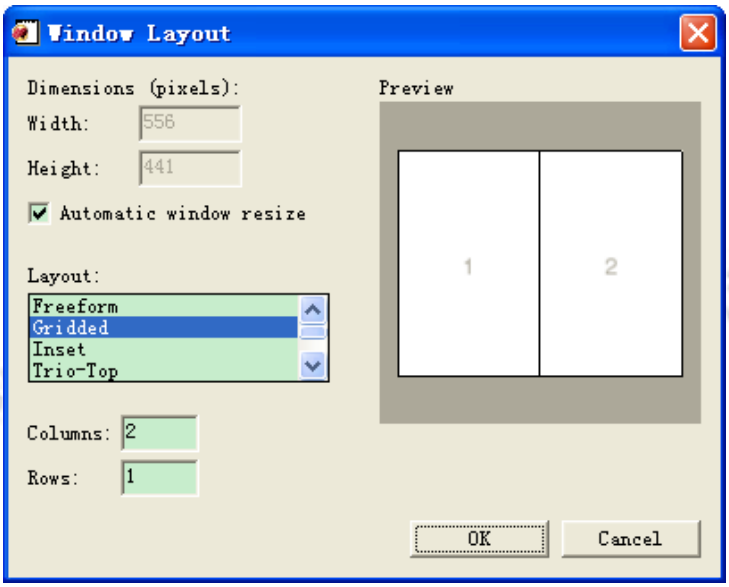
10.2.8iImage 使用

10.2.8.1 显示图像

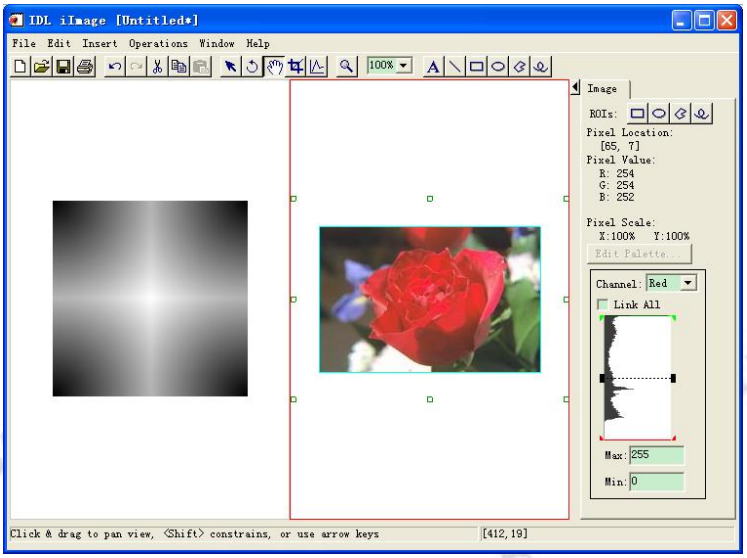
```
IDL> iimage, dist(200,200)
```



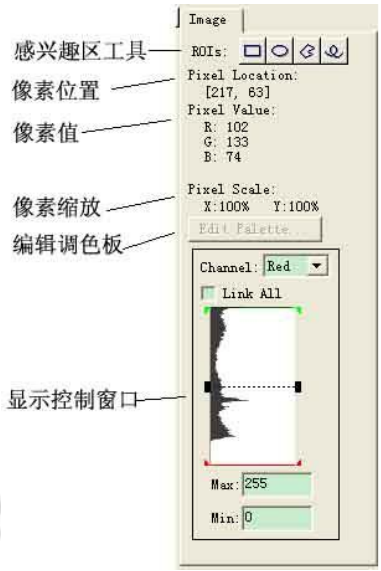
多数据显示时先设置窗口数，选择菜单 Window → Layout.



选中第二个 view 后再点击 open，显示效果如图



10.2.8.2 使用图像控制面板

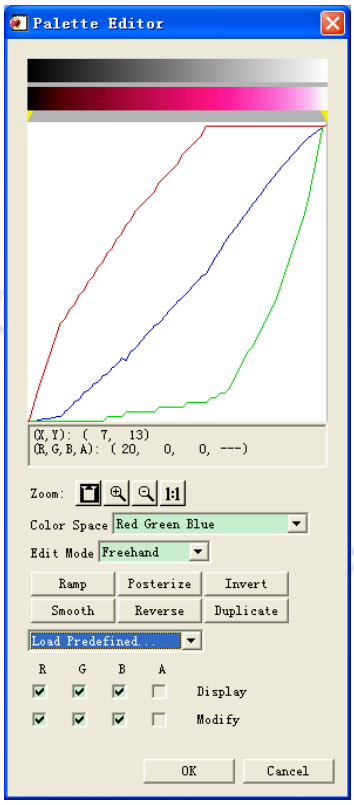


感兴趣区（ROI）工具

常用的矩形、椭圆、多边形和任意曲线来对处理的图像进行感兴趣区的创建和处理。

编辑调色板

使用该功能对灰度图像的进行伪彩色的显示，弹出的颜色表设置界面如下。

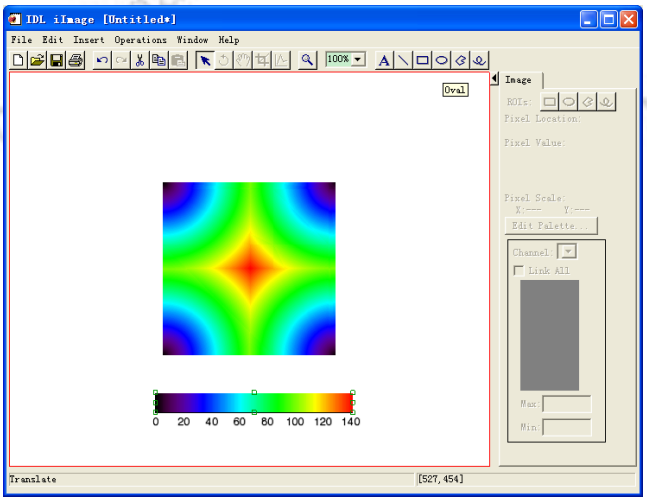


显示控制窗口

通过对直方图的调节，调节显示的亮度、对比度，如 RGB 三通道数据则可以进行单通道的修改。

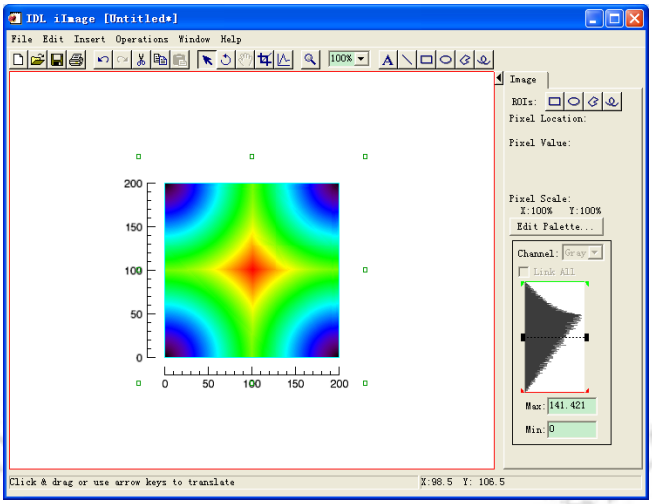
10.2.8.3 添加颜色棒

Insert → Colorbar.



10.2.8.4 添加坐标轴

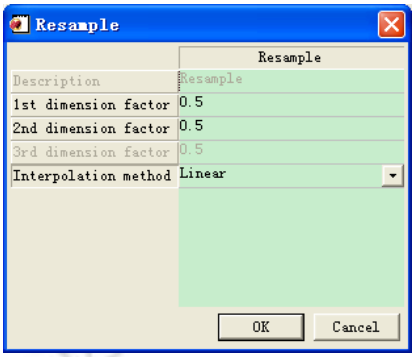
Insert → Axis



10.2.8.5 图像处理

1) 重采样

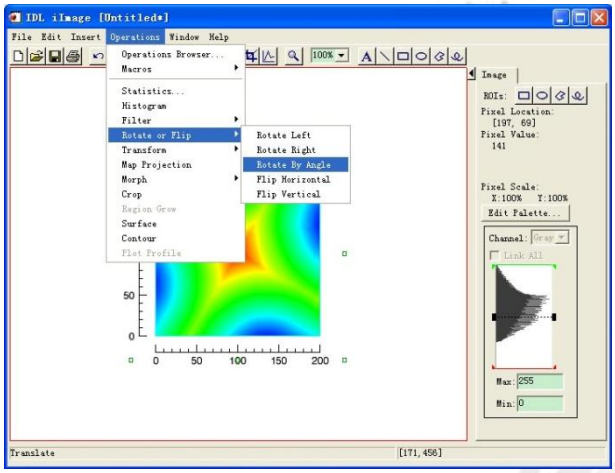
Operation → Transform → Resample.



2) 旋转：数据的旋转

Operations-Rotate or Flip-

有左翻转、右旋转、按角度旋转、竖直翻转和水平翻转等几种方式。



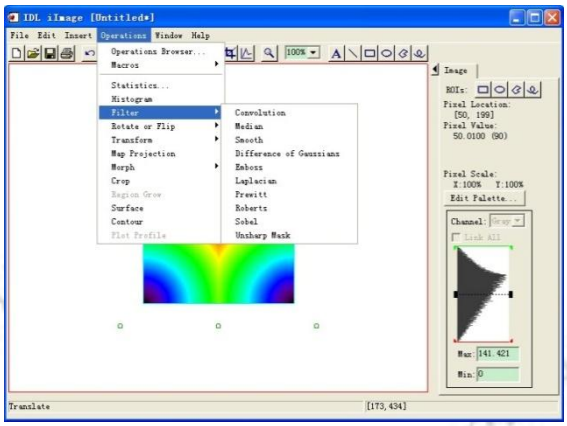
3) 显示的旋转

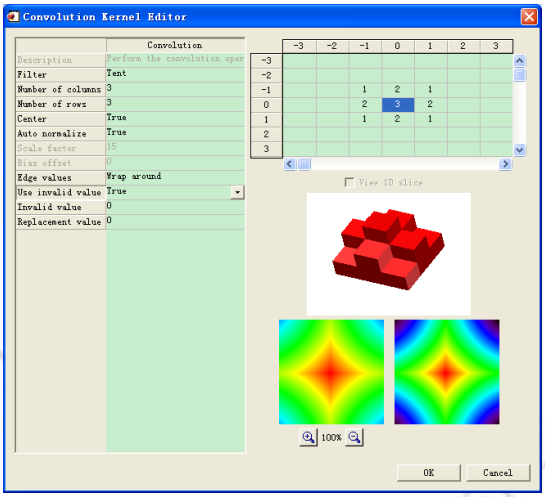
Operations-transform-rotate



4) 滤波

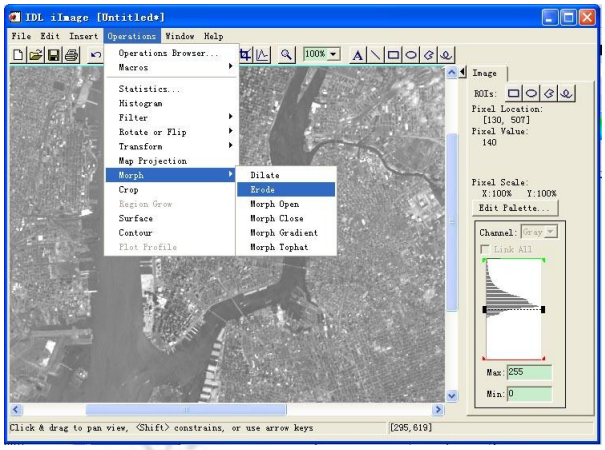
Operations → Filter

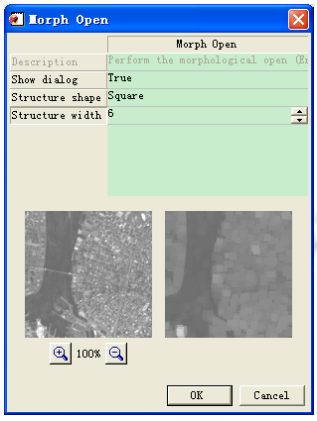




5) 形态学

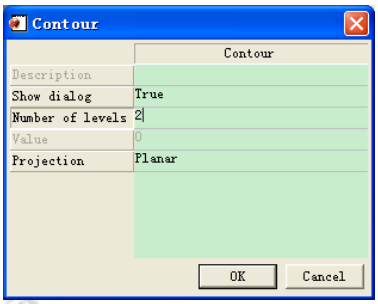
可以提供膨胀，侵蚀，变形开放，形态接近，变形梯度等形态学处理。

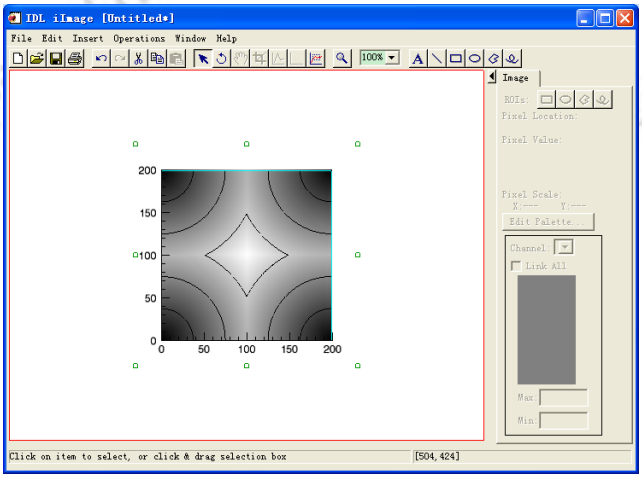




10.2.8.6 添加等值线

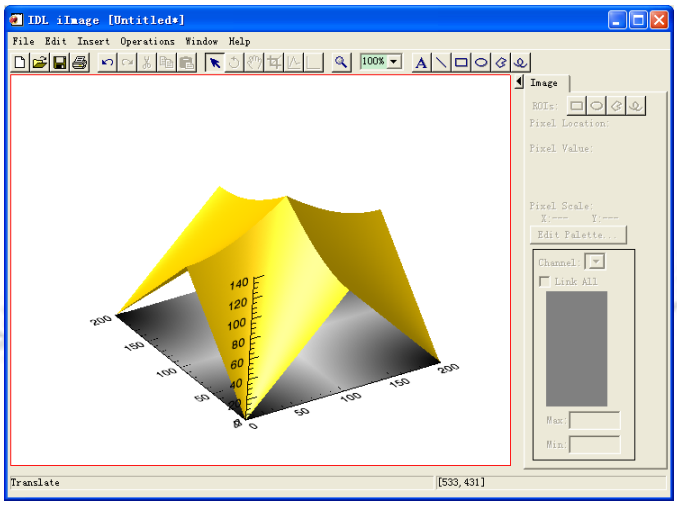
Operations → Contour





10.2.8.7 添加曲面

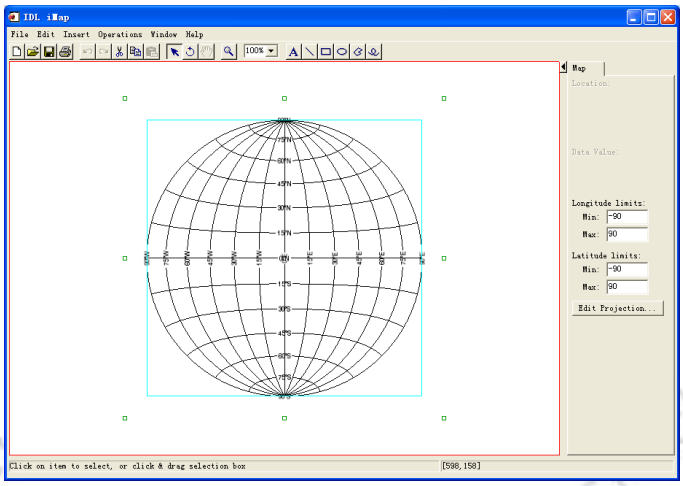
Operations → Surface



10.2.9 iMap 使用

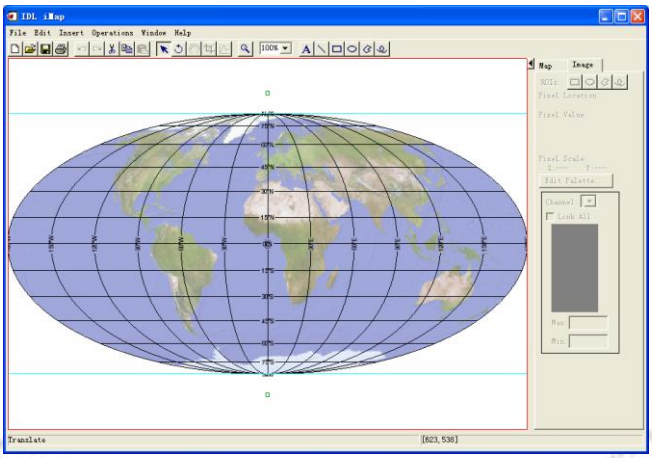
10.2.9.1 创建兰伯特投影

IMAP, MAP_PROJECTION='Lambert'



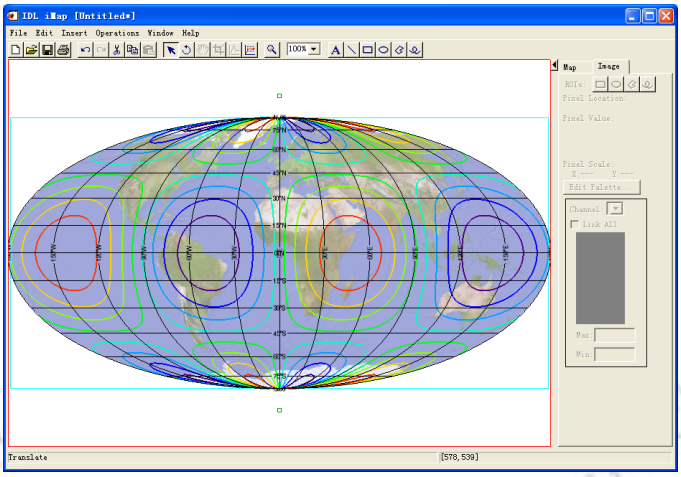
10.2.9.2 创建全球的 Mollweide 投影数据

```
file = FILEPATH('avhrr.png', SUBDIRECTORY=['examples','data'])
data = READ_PNG(file, r, g, b)
IMAP, data, LIMIT=[-90,-180,90,180], $
  MAP_PROJECTION='Mollweide', RGB_TABLE=[[r],[g],[b]], $
  IMAGE_TRANSPARENCY=50, GRID_UNITS=2, $
  IMAGE_LOCATION=[-180,-90], IMAGE_DIMENSIONS=[360,180]
```



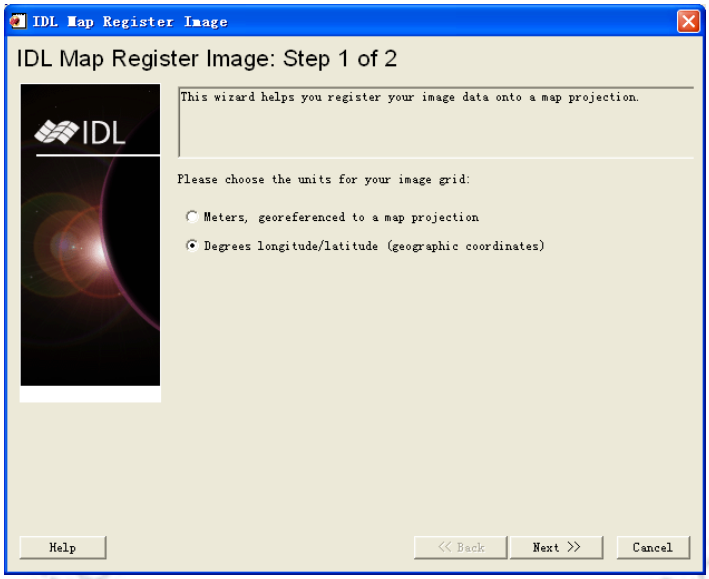
10.2.9.3 在其基础上添加等值线

```
clons = FINDGEN(360) - 179.5
clats = FINDGEN(180) - 89.5
cdata = SIN(clons/30) # COS(clats/30)
IMAP, cdata, clons, clats, /CONTOUR, /OVERPLOT, $
  RGB_TABLE=39, GRID_UNITS=2, $
  N_LEVELS=10, C_THICK=REPLICATE(2,10)
```

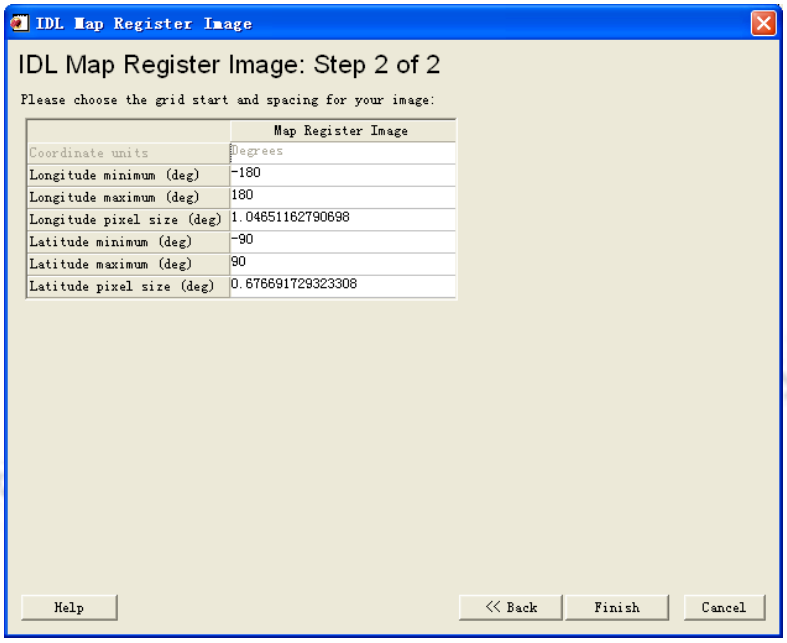


10.2.9.4 导入数据

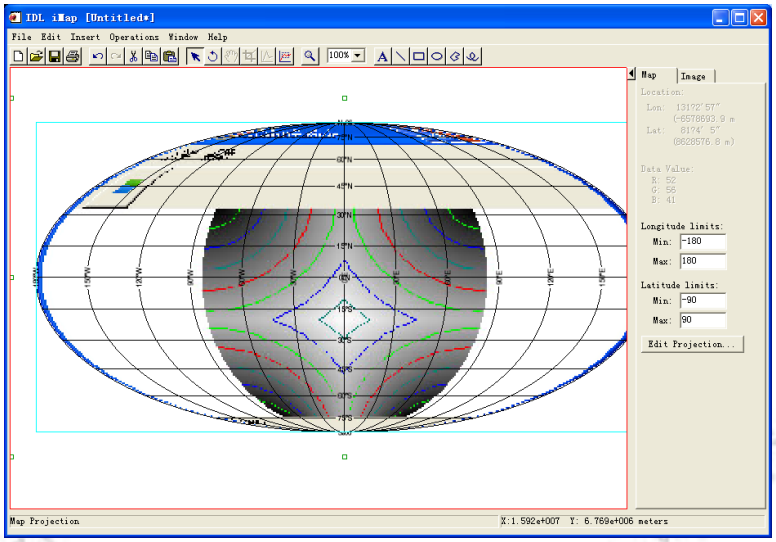
1) 选择控制点标类型；



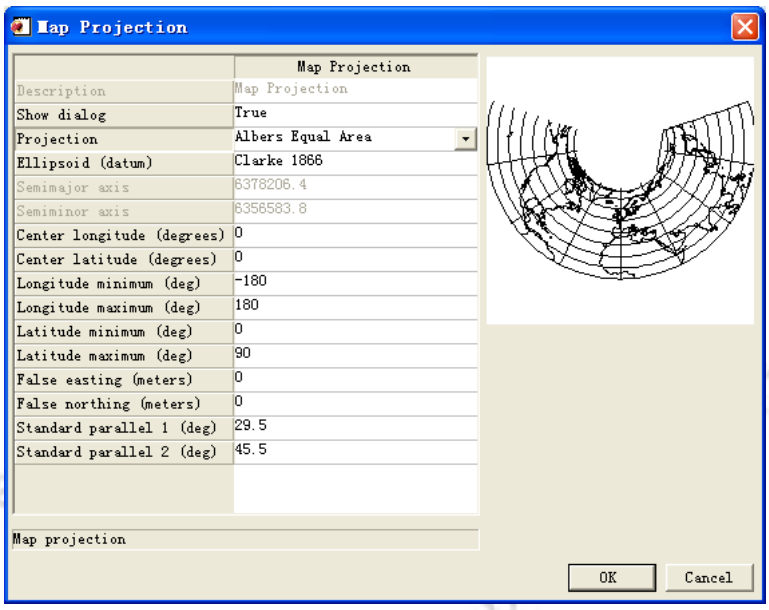
2) 选择影像坐标范围默认全球的。



3) 点击 Finish 后纠正后图像



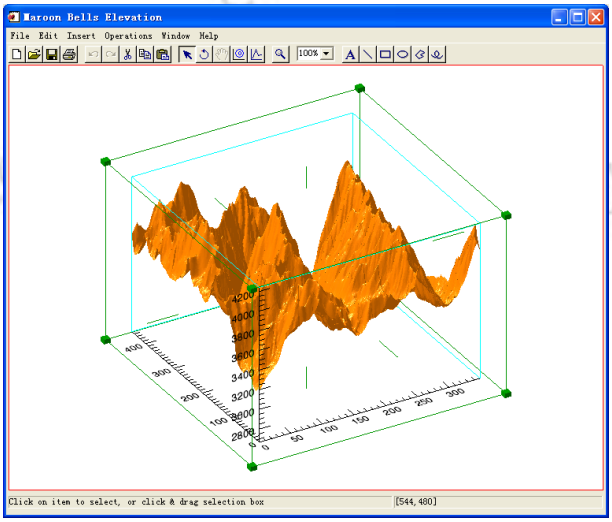
4) 可选期右侧 Map 面板上的 Edit Projection 进行投影转换



10.2.10 iSurface 使用

10.2.10.1命令行下启动 iSurface

```
file = FILEPATH('surface.dat', $
      SUBDIRECTORY = ['examples', 'data'])
data = READ_BINARY(file, DATA_DIMS = [350, 450], DATA_TYPE = 2, $
  ENDIAN = 'little')
ISURFACE, data, TITLE = 'Maroon Bells Elevation', $
  COLOR = [255, 128, 0]
```

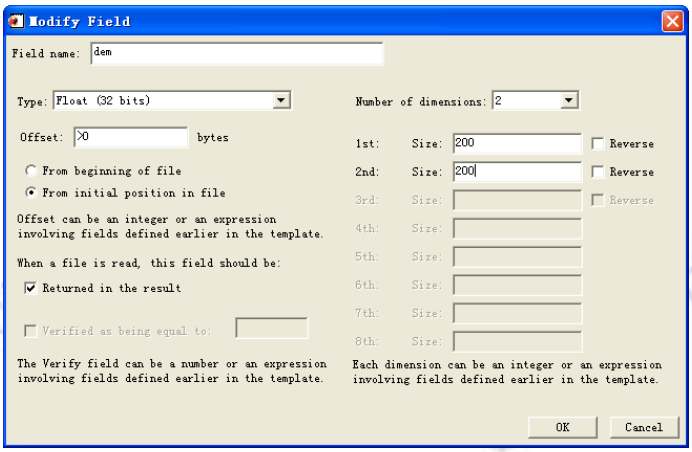


10.2.10.2 iSurface 打开数据

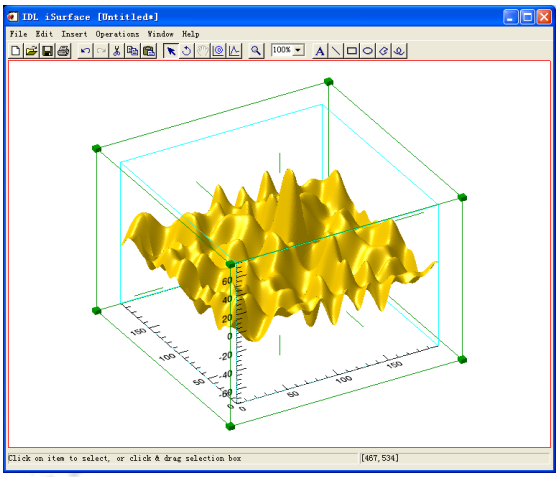
1) 先启动 iSurface

2) 然后点击菜单 file→open，选择 examples/data/idemosurf.dat

3) 参数设置如下图

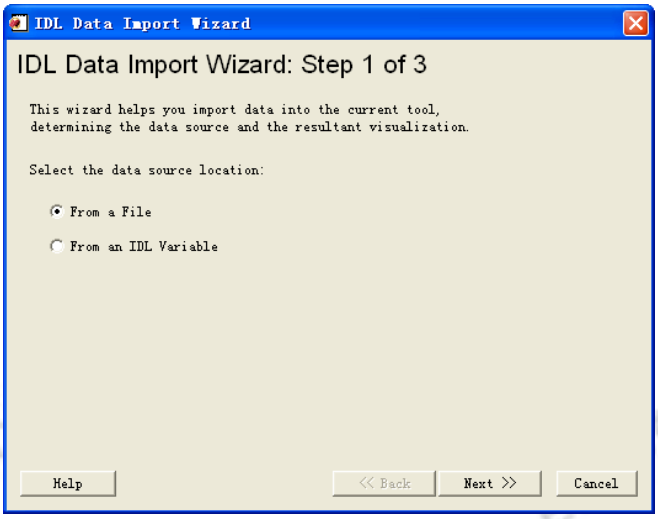


4) 点击 OK

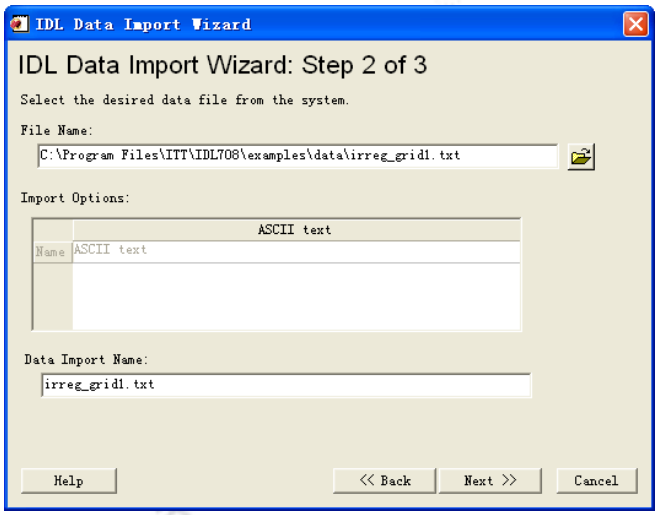


10.2.10.3iSurface 导入数据

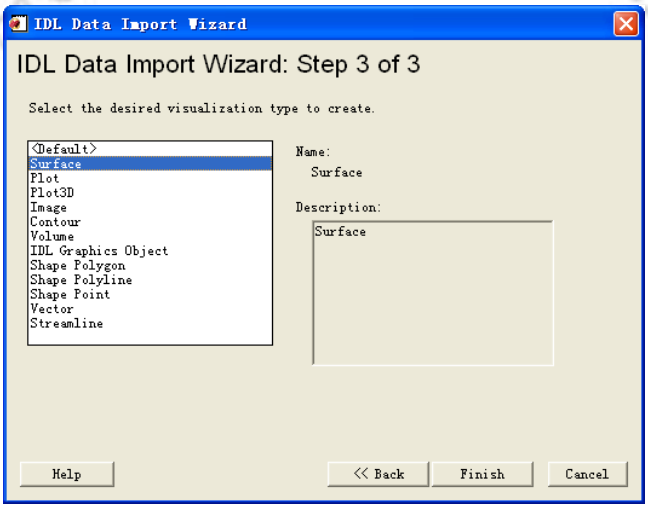
1) 选择菜单[File]下的 Impot，在向导界面中选择 From a File。



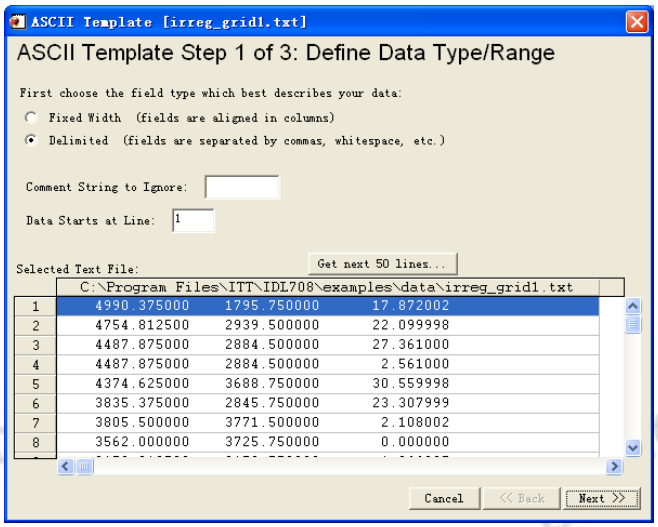
2) 选择\examples\data\irreg_grid1.txt 为输入文件名



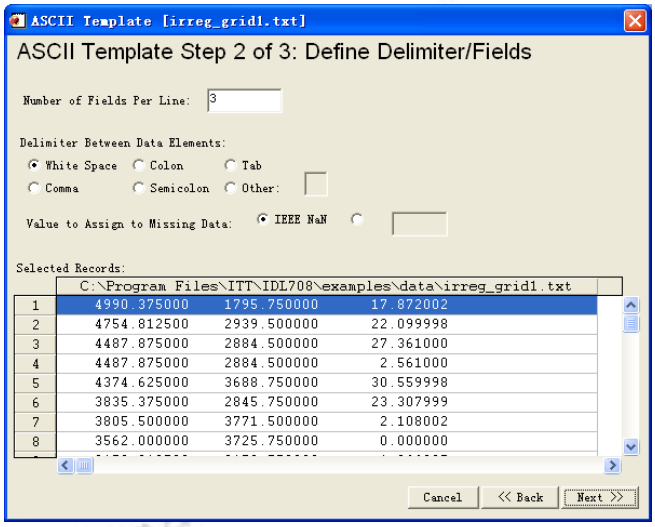
3) 根据结果选择显示为 Surface



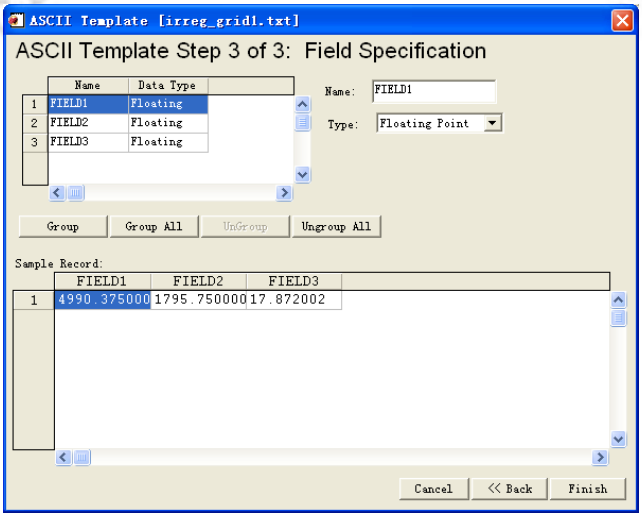
4) 启动 ASCII 文件导入向导



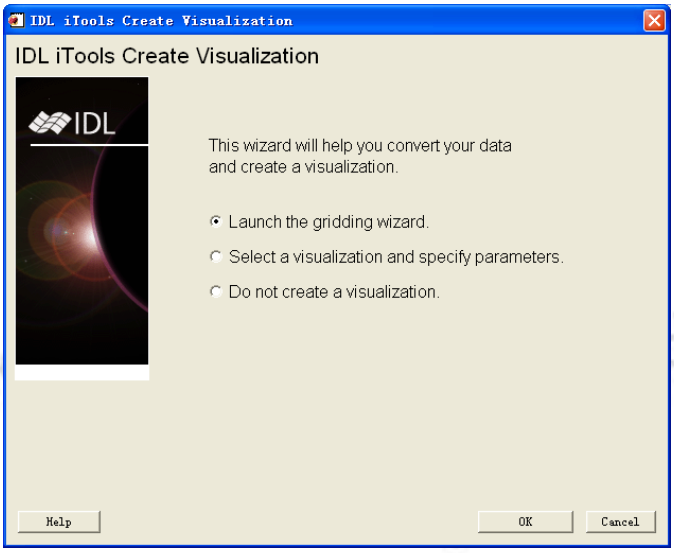
5) 定义分隔符位空格



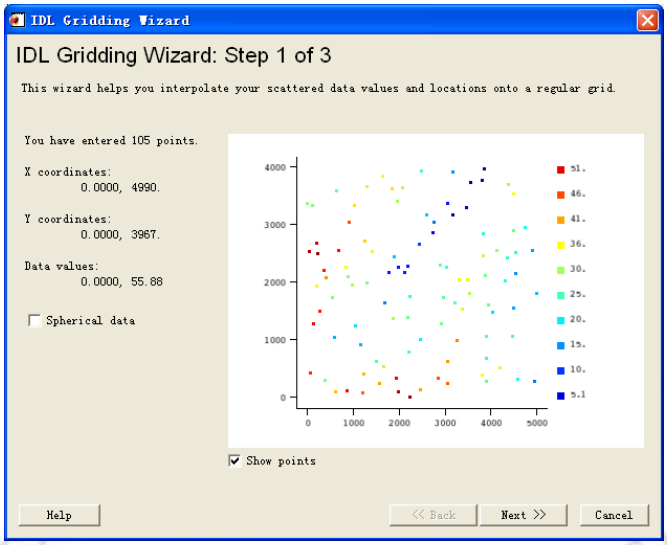
6) 设定导入后的变量名及类型



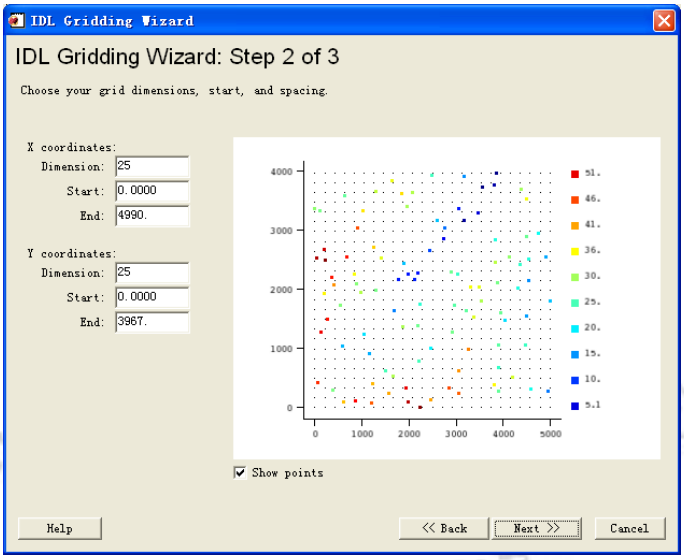
7) 启动插值向导界面



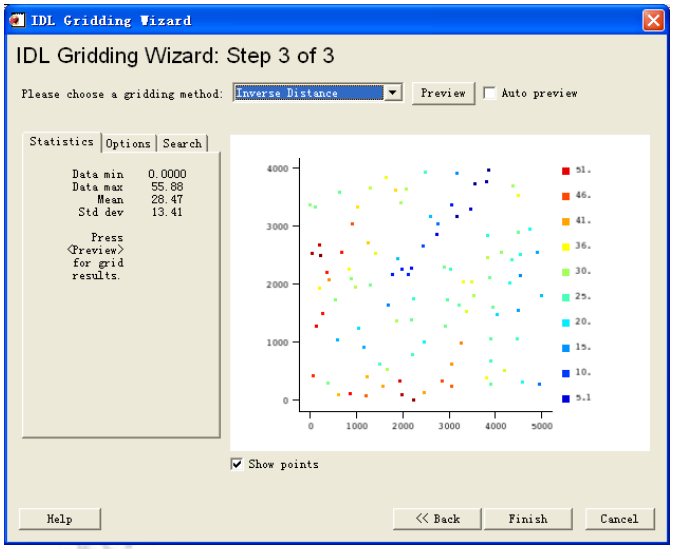
8) 待插值点信息



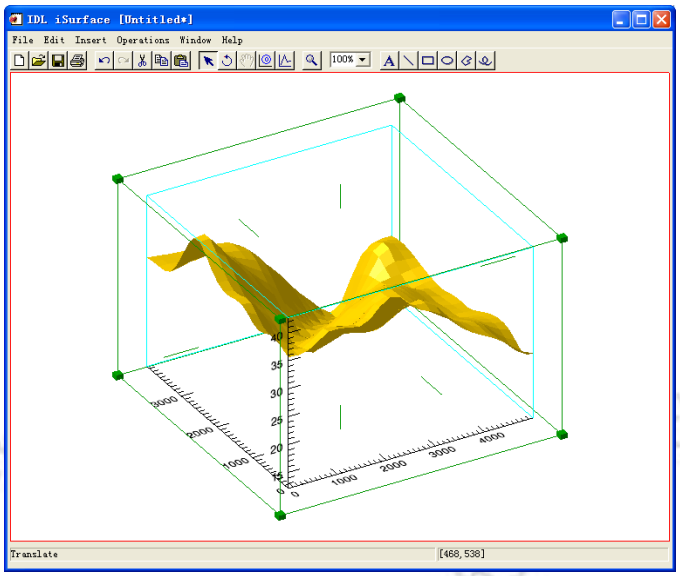
9) 设定插值维数参数



10) 选择插值算法



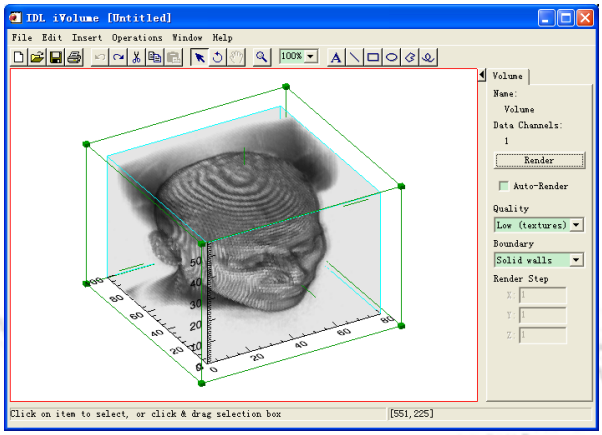
11) 点击 Finish，插值完成



10.2.11 iVolume 使用

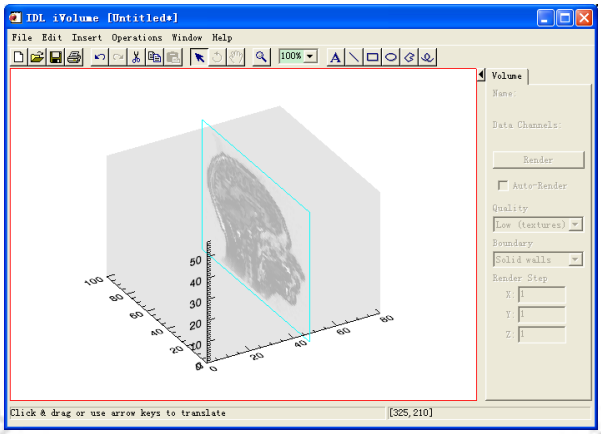
1) 命令行下启动 iVolume

```
IDL> file = FILEPATH('head.dat', $
IDL>        SUBDIRECTORY = ['examples', 'data'])
IDL> data = READ_BINARY(file, DATA_DIMS = [80, 100, 57])
IDL> IVOLUME, data
```



2) 进行三维切片

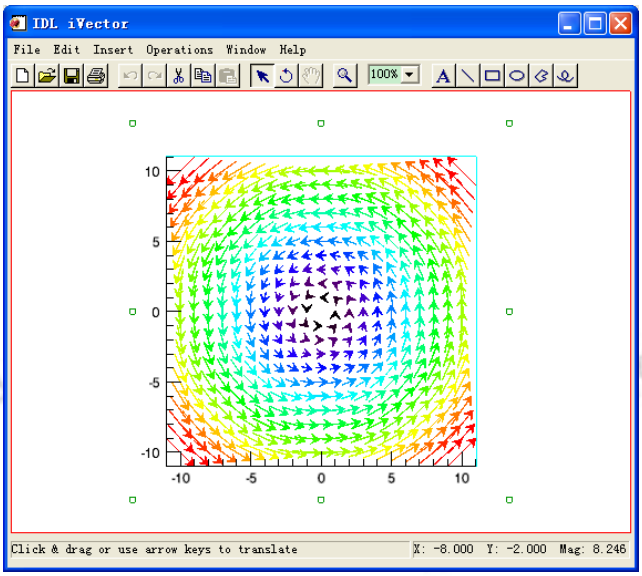
Operations → Volume → Image Plane



10.2.12 iVector 使用

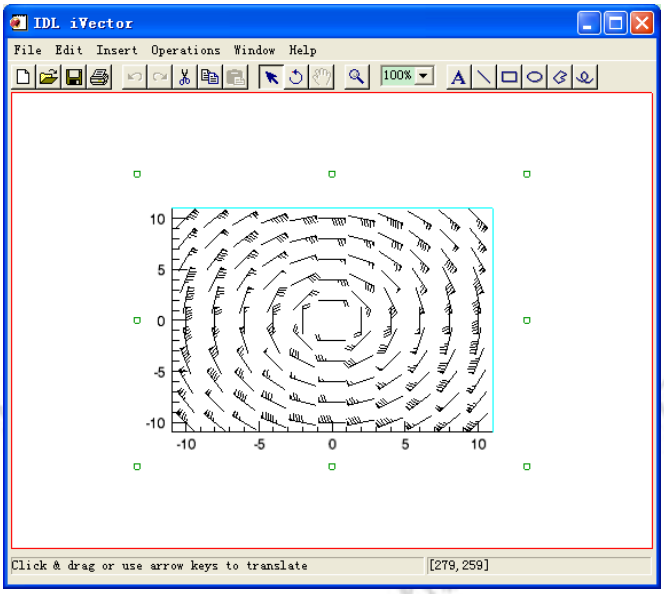
1) 矢量场显示

```
IDL> x = (y = FINDGEN(21) - 10)
IDL> u = REBIN(-TRANSPOSE(y),21,21)
IDL> v = REBIN(x,21,21)
IDL> IVECTOR, u, v, x, y, AUTO_COLOR=1, $
IDL>       RGB_TABLE=39, SCALE_ISOTROPIC=1
```



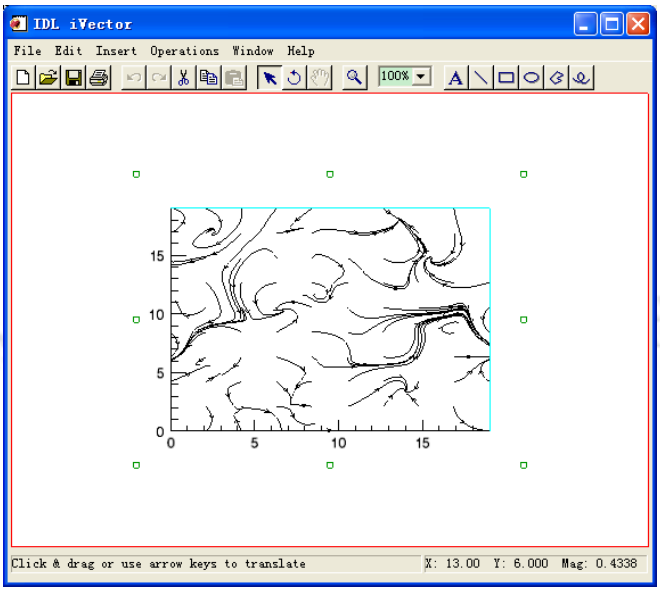
2) 风场标示

```
IDL> x = (y = 2*FINDGEN(11) - 10)
IDL> u = 9*REBIN(-TRANSPOSE(y),11,11)
IDL> v = 9*REBIN(x,11,11)
IDL> IVECTOR, u, v, x, y, VECTOR_STYLE=1
```



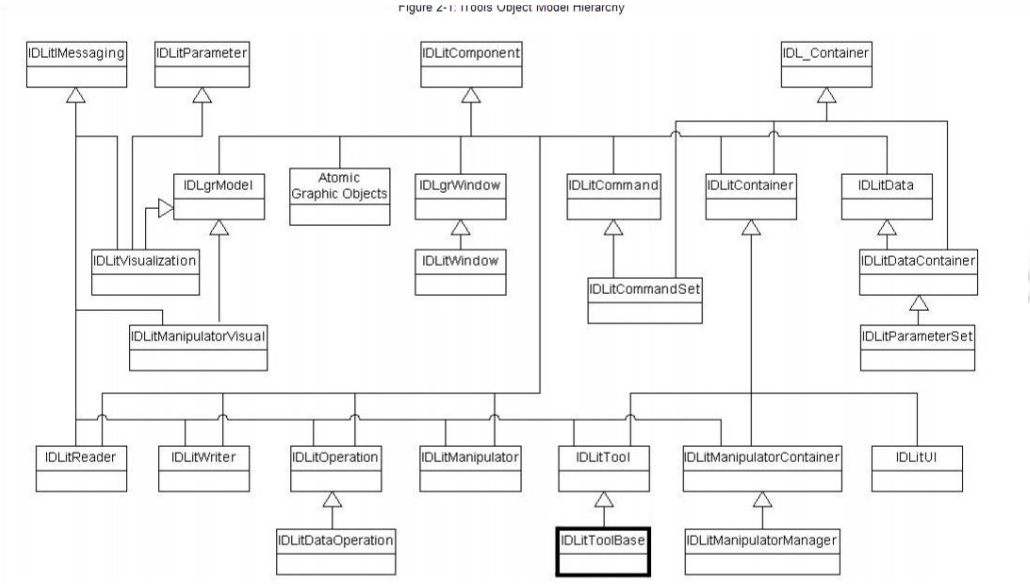
3) 流线型

```
IDL> u = RANDOMU(1,20,20) - 0.5
IDL> v = RANDOMU(2,20,20) - 0.5
IDL> IVECTOR, u, v, /STREAMLINES, $
IDL>       X_STREAMPARTICLES=10, Y_STREAMPARTICLES=10, $
IDL>       HEAD_SIZE=0.1, STREAMLINE_NSTEPS=200
```



10.3 开发

iTools 是一个完整的框架，从下图可以看出来。当然，理解这样的框架是需要一定的时间的。



有兴趣的读者可参考 help 的 IDL Programmers' Guides > iTool Programming 部分。

bbs.esrichina-bj.cn

11 IDL 程序的部署与发布

11.1 IDL 的部署

11.1.1 Runtime 许可发布

IDL 的 Runtime 发布使得 IDL 程序可以脱离开发环境，独立运行 IDL 自身的数据分析和显示功能，使用前 IDL 的代码都必须编译并发布为二进制的 sav 文件格式。

需要注意的是，Runtime 发布需要独立的 Rumtime 许可才能运行，其次是程序不能进行编译执行 IDLpro 源代码文件。

11.1.2 虚拟机免费发布

IDL 的虚拟机是一种免费发布可执行 IDL 的.sav 程序的方式。虚拟机可以运行在 IDL 支持的所有软件平台上，这种方式可以使得 IDL 程序可以自由、免费的运行在其他人的电脑上。

虚拟机发布存在以下限制：

1）虚拟机必须通过 ITT Visual Information Solutions.提供的程序进行安装，不允许进行更改；

2）启动时会弹出虚拟机启动画面；

3）虚拟机只能运行 IDL6.0 或更高版本的 IDL 生成的 sav 文件；

4）不允许执行 EXECUTE 函数；

5）不支持 Callable 技术；

6）虚拟机不安装 DataMiner, IDLffDicomEX feature, IDL-Java bridge 等功能组件，如果用到了这些组件需要安装 IDL 的完全版。

11.2 IDL 的发布

11.2.1 Sav 文件

IDL 自 6.0 版本之后的程序和数据都可以保存在 sav 文件中，sav 文件可用虚拟机来执行，创建 sav 文件有两种方式。

11.2.1.1 单个或多文件创建 sav

对已经编译过的单个或多个 pro 文件生成 sav 时，可以用 save 函数。如自定

义的函数 myFun.pro 已经编译，可以用下面命令进行 sav 的生成。

```
IDL>SAVE, /ROUTINES, 'myFun', FILENAME = ‘c:\myfun.sav'
```

11.2.1.2 工程创建 sav

对工程的 sav 文件创建，首先需要在工程的属性中选择创建 save 文件，如下图。

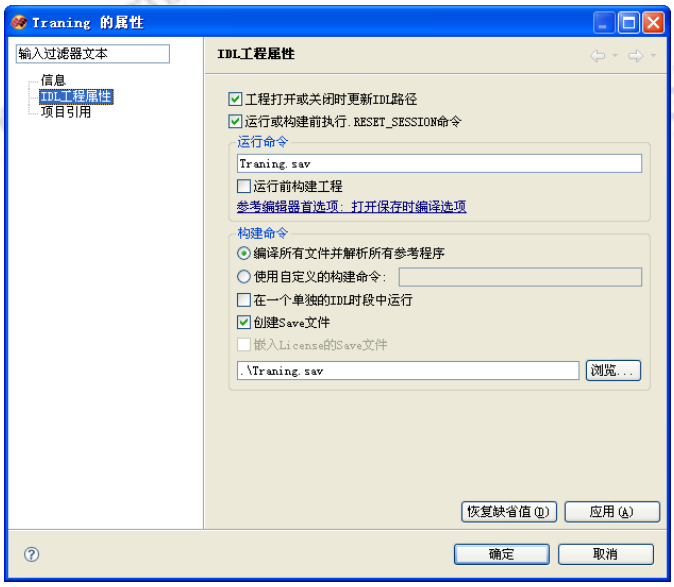


图 11-1 工程的构建属性设置

然后选择菜单工程-构建（Ctrl+B）或工具栏上的快捷方式进行工程的构建，在控制台上会对生成的文件名进行输出。

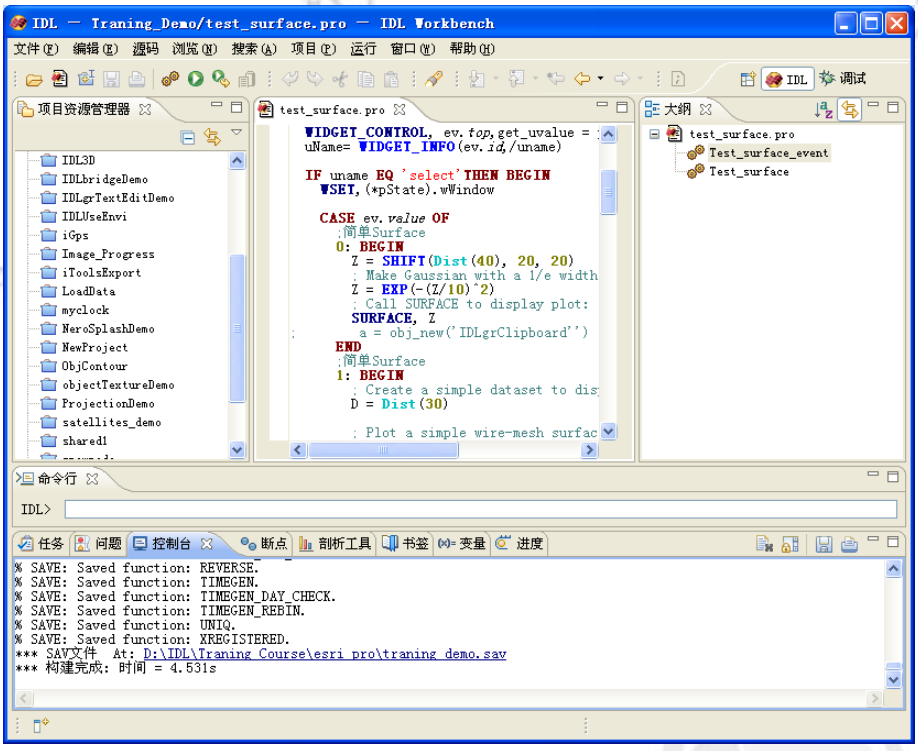


图 11-2 工程构建的控制台信息

11.2.2 可执行文件（Windows 下的 exe）

11.2.2.1 单独 IDL 程序

IDL 自 7.0 版本起下用 MAKE_RT 命令代替了 Export 功能，可以依照下面步骤进行。

1) 首先收集相关信息：

A. 选择你要运行的程序的名字；myApp

B. 确定发布的完整输出路径存在，否则 make_rt 会报错；OutDir

C. 知道包含你程序所有代码的 sav 文件完整路径；savefile

2) 运行MAKE_RT过程

MAKE_RT, 'myApp', OutDir, SAVEFILE=savefile

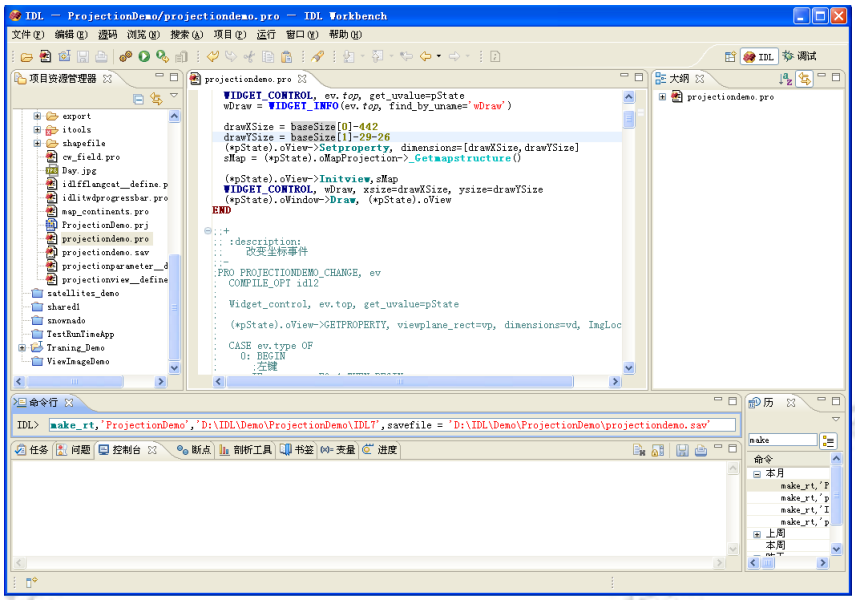


图 11-3 make_rt 的使用方式

3) 添加额外文件到发布的程序中；

程序中使用到了某些配置文件或基础数据文件则需要拷贝到对应的目录下。如运行系统中没有安装 IDL，则需要先安装一个文件来确保其运行。32 位系统运行 systemdll32_setup.exe，64 位系统运行 systemdll64_setup.exe，这两个文件的对应目录为 IDL_PATH\ bin\make_rt。

4) 修改启动脚本-myApp.ini;

myApp.ini 包含的内容

[DIALOG]

Show=False ;对话框界面是否弹出<True 和 False>

BackColor=&H6B1F29 ;对话框的背景颜色

Caption=<any string> ;对话框的标题文字

Picture=.\splash.bmp ;对话框需要显示的图像，标准大小 480 x 335，若图像大则大的部分显示背景色；

DefaultAction=<path to application> ；运行时执行的代码；

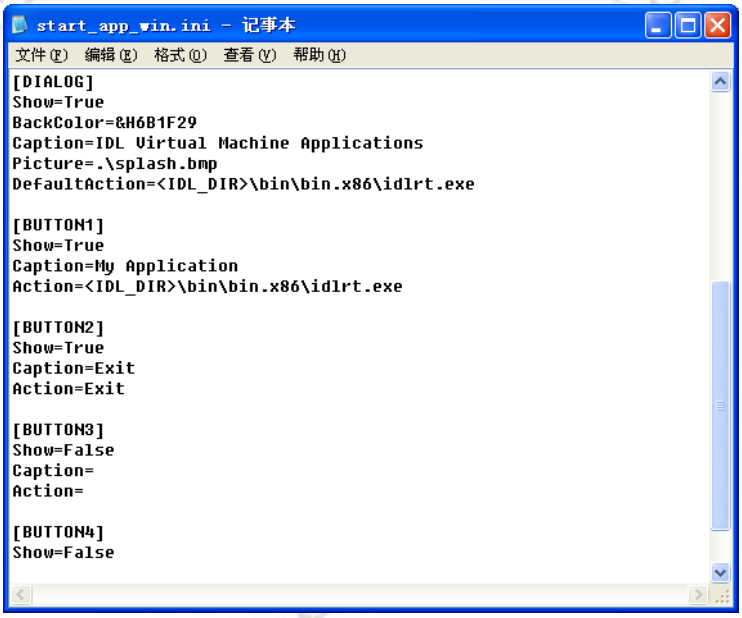
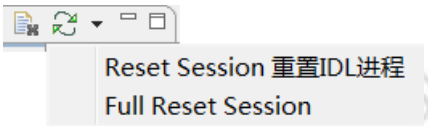


图 11-4 修改启动脚本文件

11.2.2.2 程序中调用 ENVI

如果 IDL 程序中使用了 ENVI 的函数，则在 IDL7.0 下发布的操作步骤为：

1) 重启 idl 编译器进程

点击按钮 

或在 IDL 的命令行中输入.Reset_Session 。

2) 编译当前 pro

IDL>.compile myApp.pro

3) 编译系统的基本 function

IDL>Resolve_All, /continue_on_error, skip_routines='envi'

4) 保存 sav 文件

IDL>SAVE, file=' myApp.sav', /routines

5) 后面用 make_rt 发布

IDL>MAKE_RT, ' myApp ', Outdir, SAVEFILE= saveFile

6) 修改发布后文件夹里面的 myApp.ini 文件

把文件中[Dialog]下的 DefaultAction 后面修改为系统安装 IDL 的 IDLrt.exe 的完整路径，例如：C:\Program Files\ITT\IDL706\bin\bin.x86\Idlrt.exe -rt=myApp.sav。



图 11-5 修改启动脚本文件

11.2.3 COM 和 JAVA

使用对象导出向导可以将 IDL 对象输出到 COM 和 JAVA 环境中，进而直接缩短接口开发时间，参考 12.2 中的对象输出助手章节部分。

12 混合调用

12.1 IDL 功能扩展

12.1.1 直接调用 SPAWN

使用 SPAWN 方式。SPAWN 产生一个子的过程来执行一个命令或一系列的命令。如执行系统的画图程序可以用下面命令：

```
IDL> spawn,'mspaint'
```

系统会自动启动一个 Dos 的命令行界面并启动画图程序；

加/NOSHELL，不启动 Dos 命令行界面；

加/LOG_OUTPUT，启动的程序会最小化；

```
IDL> spawn,'mspaint',/NoShell
IDL> spawn,'mspaint',/NoShell,/LOG_OUTPUT
```

也可直接调用文件通过系统默认程序来打开，若格式未知则需要手动选择程序打开；

```
IDL> spawn,'c:\ IDL-test.docx '
```

12.1.2 IDL 调用 ActiveX

IDL 可以调用微软的组件对象模型（COM）和 ActiveX 控件，通过这种方式实现 IDL 功能的扩展弥补 IDL 某些方面的不足。com 组件的调用是 IDLcomIDispatch 类实现的，ActiveX 控件的调用是 IDLcomActiveX 类实现的。

12.1.2.1 Media Player 控件调用

根据 Demo 程序[\chapter12\ Test_activex_mediaplayer]分析调用 ActiveX 的方法步骤。

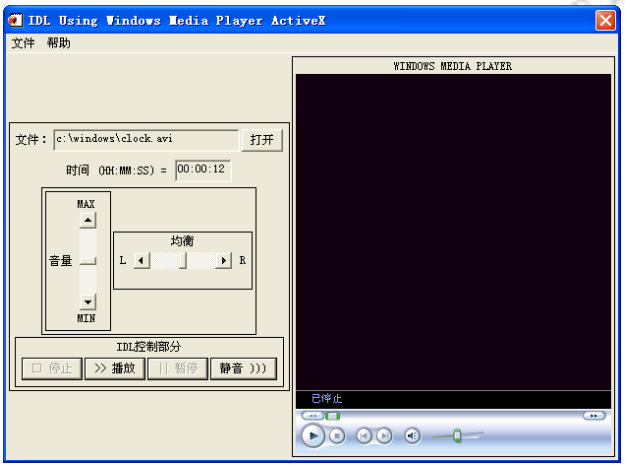


图 12-1 IDL 调用 Windows Media Player 控件

使用 ActiveX 控件的重点在两个地方：

1) 组件调用为 Result = WIDGET_ACTIVEX(Parent, COM_ID)；

使用 WIDGET_ACTIVEX 的方式跟 IDL 使用其他组件的方式一样，调用返回的 Result 为 IDLcomActiveX 类对象，该类继承了 COM 组件类 IDLcomIDispatch，提供了 SetProperty 和 GetProperty 两种方法，并可根据组件的接口描述文件调用组件相关的方法。

2) 组件自身里面的属性和方法，可以按照下面的方式查看其接口描述文件。

该组件的 MSDN 帮助里面有详细的说明，链接如下

ms-help://MS.MSDNQTR.v80.chs/MS.MSDN.v80/MS.WIN32COM.v10.en/wmplay10/mmp_sdk/interfaces.htm 如下图

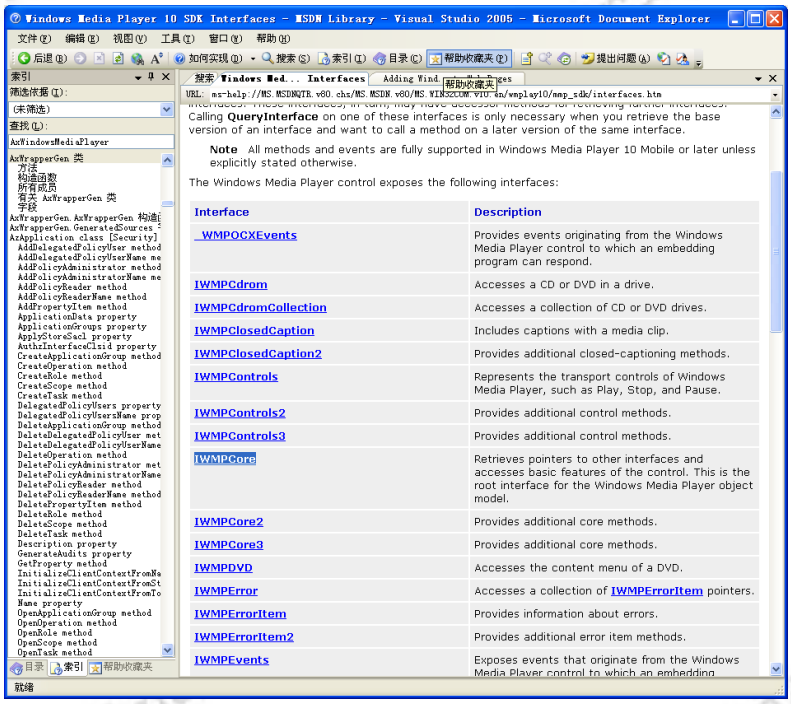


图 12-2 MSDN 中的控件信息

通过 OLE/COM Object Viewer 查看所调用的 ActiveX 控件，将信息转存为接口描述语言(.IDL)文件，该文件是 ASCII 码文件，直接打开查看即可。

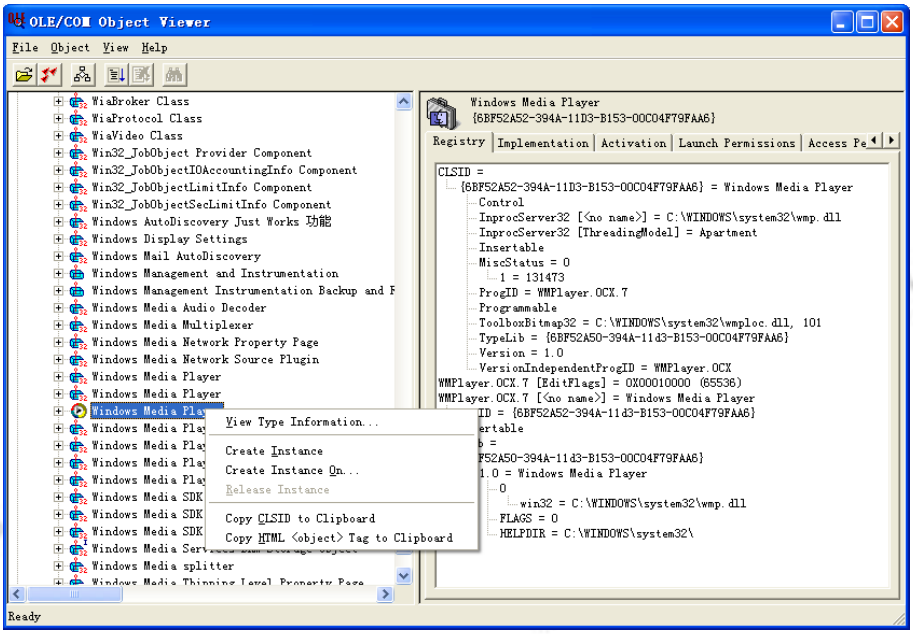


图 12-3 查看信息

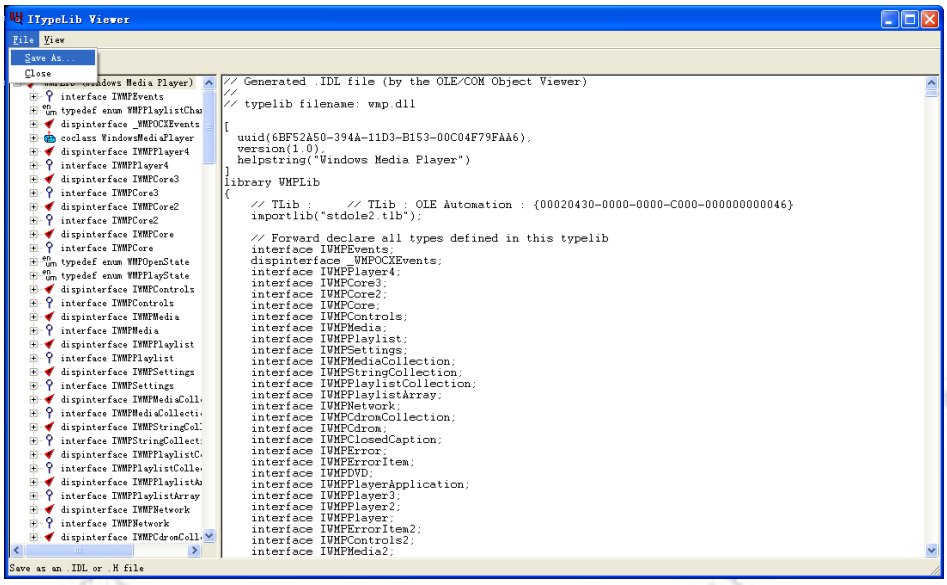


图 12-4 保存显示信息

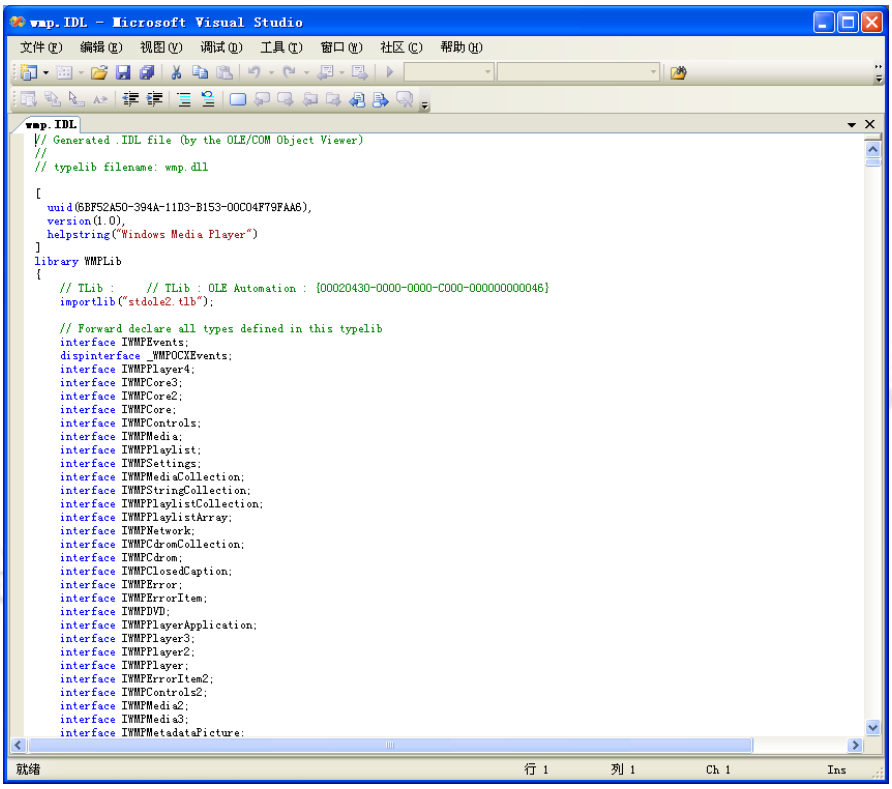


图 12-5 在 VisualStudio 中查看 IDL 文件

该描述文件的接口内容分下面几类：

（1） 定义中包含 retval 的带返回值的方法在 IDL 中作为函数进行调用，如下面格式；

[id(0x00000001)]

HRESULT GetCLSID([out, **retval**] BSTR* pBstr);

（2） 定义中不含 retval 的无返回值的方法在 IDL 中作为过程进行调用；

[id(0x00000033), helpstring("Begins playing media")]

HRESULT play();

[id(0x00000034), helpstring("Stops play of media")]

HRESULT stop();

（3） 定义中包含 propget 的方法在 IDL 中通过 IDLcomIDispatch::GetProperty 来返回获得一个对象，如下面的:

[id(0x00000001), **propget**, helpstring("Returns or sets the URL")]

HRESULT URL([out, retval] BSTR* pbstrURL);

[id(0x00000004), **propget**, helpstring("Returns the control handler")]

HRESULT controls([out, retval] IWMPControls** ppControl);

（4） 定义中包含 propput 的属性则可以通过 IDLcomIDispatch::SetProperty 方法设置一个对象；

[id(0x00000001), **propput**, helpstring("Returns or sets the URL")]

HRESULT URL([in] BSTR pbstrURL);

（5） 方法的定义中如包括可选择性输入，如下面例子，包括 optional 关键字的参数是可选择性的；

[id(0x00000004)]

HRESULT Msg1or2InParams([in] BSTR str,[in, **optional**] int val,

[out, retval] BSTR* pVal);

（6） 方法中如果提供了 defaultvalue()而不是 optional，那么后面的参数在调用该方法时是必须的。如下，第二个参数的默认值为 15:

HRESULT

Msg1or2InParams([in] BSTR str, [in, **defaultvalue(15)**] int val, [out, retval] BSTR* pVal)

（7） 如果方法的返回是 HURSULY，那么该方法是不需要调用的。

3) 在 MediaPlayer 例子中的应用

获得组件的控制组件部分，即 control、setting 和版本等信息，其中接口部分

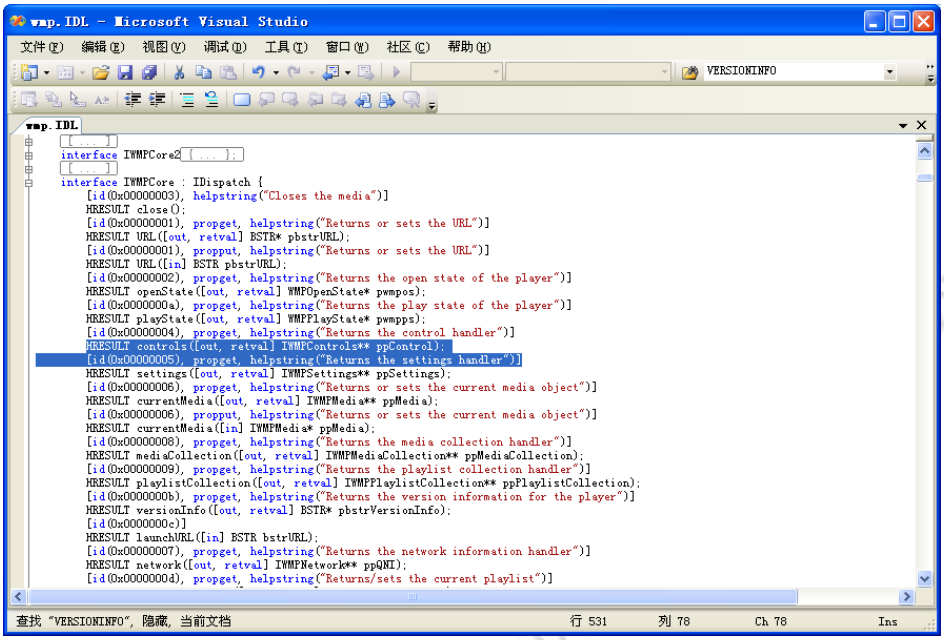


图 12-6 控件的主控接口

（1） control 下的列表

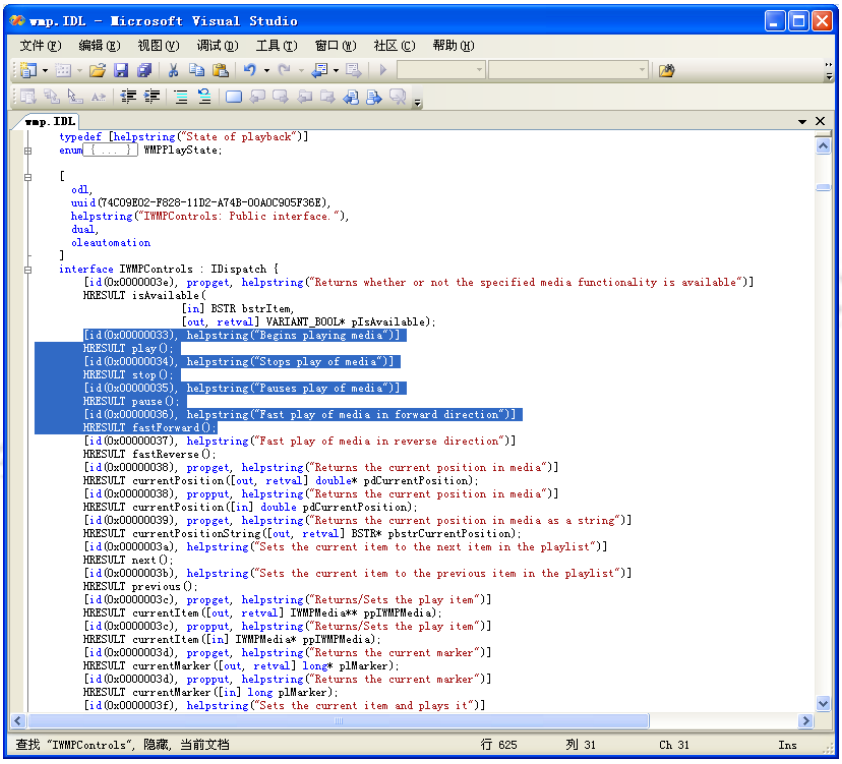


图 12-7 控制部分的接口

（2） setting 下的接口列表

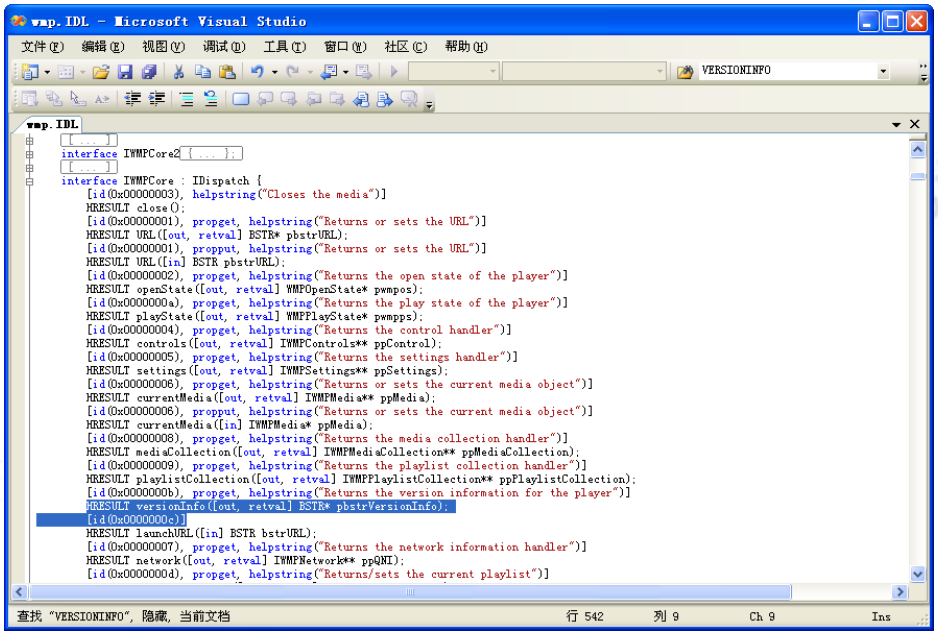


图 12-8 设置部分的接口

4) 可用 Help 来查看 IDL 系统中的控件信息；

```
IDL> help,/objects
** Object class IDLCOMACTIVEX$CLSID$6BF52A52_394A_11D3_B153_00C04F79FAA6, 1 direct superclass, 2 known methods
        Superclasses:
```

```
            IDLCOMACTIVEX    <Direct>
            IDLCOMIDISPATCH
        Known                                Function                                Methods:
IDLCOMACTIVEX$CLSID$6BF52A52_394A_11D3_B153_00C04F79FAA6::VERSIONINFO
        Known                                Procedure                               Methods:
IDLCOMACTIVEX$CLSID$6BF52A52_394A_11D3_B153_00C04F79FAA6::GETPROPERTY
    ** Object class IDLCOMIDISPATCH$CLSID$8DA61686_4668_4A5C_AE5D_803193293DBE, 1
direct superclass, 0 known methods
        Superclasses:
            IDLCOMIDISPATCH    <Direct>
    ** Object class IDLCOMIDISPATCH$CLSID$A1D1110E_D545_476A_9A78_AC3E4CB1E6BD, 1
direct superclass, 0 known methods
        Superclasses:
            IDLCOMIDISPATCH    <Direct>
    ** Object class IDLCOMIDISPATCH$CLSID$FDA937A4_EECE_4DA5_A0B6_39BF89ADE2C2, 1
direct superclass, 1 known method
        Superclasses:
            IDLCOMIDISPATCH    <Direct>
        Known                                Procedure                               Methods:
IDLCOMIDISPATCH$CLSID$FDA937A4_EECE_4DA5_A0B6_39BF89ADE2C2::GETPROPERTY
```

12.1.2.2 日期选择控件调用

参考程序[\chapter12\ test_inputDate]



图 12-9 日期选择控件 Demo 界面

12.1.3 IDL 调用 DLL

12.1.3.1 VC++创建动态连接库（DLL）

1) 打开 VC++编译器

2) 创建一个 Win32 Dynamic-Link Library

Project Name：imageProcess；

动态库类型选择：A simple DLL project。

3) 编写线性拉伸的函数

（1） new 一个 class，起名为 ClinearStretch

（2） 打开 linearStretch.h，编写函数接口定义：

```
long    ExtendImg(BYTE * Bits,int width,int height,int minpixel,int maxpixel);
```

（3） 打开 linearStretch.cpp，写函数内容：

```
long    ClinearStretch::ExtendImg(BYTE * Bits,int width,int height,int minpixel,int maxpixel)
{
    if(minpixel<0) minpixel=0;
    if(maxpixel>255) maxpixel=255;
    if(Bits==NULL || width<0 || height<0 )
        return 0;
    float a,b;
    a=(float)255/(maxpixel-minpixel);
    b=(float)-minpixel*255/(maxpixel-minpixel);
    for(int i=0;i<width*height;i++)
    {
        float temp=((float)Bits[i]-minpixel)/(maxpixel-minpixel)*255;

        if(temp<0) temp=0.0;
        if(temp>255) temp=255.0;
        Bits[i]=(BYTE)temp;
    }
    return 1;
}
```

4) 将 IDL 安装路径下的\external\include\ idl_export.h 文件拷贝的工程路径下。并将 idl_export.h 文件添加到工程中。

5) 打开 imageProcess.cpp 文件。添加如下代码：

```
#include "idl_export.h"
#include "linearStretch.h"
```

接着定义 C 语言的接口类型，添加如下代码

```
#ifdef WIN32
#define IDL_LONG_RETURN extern "C" _declspec(dllexport) long
#else
#define IDL_LONG_RETURN long
#endif
```

接下来写接口函数，注意，输入参数都在 argv[]数组里，由于 IDL 的所有变量都是矩阵形式，也就是数组，所以输入变量在 C 语言里都是指针变量，因此

应该定义指针变量接收外部参数。

```
IDL_LONG_RETURN ImgExtend(int argc,void *argv[])
{
    //用指针类型变量获得外部数据的地址
    BYTE *SrcBits=(BYTE *)argv[0];
    int *iw=NULL;
    iw=(int *)argv[1];
    int *ih=NULL;
    ih=(int *)argv[2];
    int *minp=NULL;
    minp=(int *)argv[3];
    int *maxp=NULL;
    maxp=(int *)argv[4];
    ClinearStretch    Img;
    return Img.ExtendImg(SrcBits,*iw,*ih,*minp,*maxp);
}
```

6) 打开 stdafx.h 文件。添加如下代码：

```
#include <afxwin.h>
```

7) Build

12.1.3.2 IDL 下调用动态连接库

将 imageProcess.dll 拷贝到 IDL 安装路径下：\bin\bin.x86\imageProcess.dll，如不放置则调用的时候 dll 应写完整路径。

test_UsingVCDLL

12.1.4 IDL 使用 DLM

DLM 是 Dynamically Loadable Modules 的缩写，它的组成是包括 VC 等编译器建立的一个动态库（DLL）和一个 DLM 的 ASCII 文件。可以进行标量，数据，字符串，结构等的互相传递和共享。

IDL 启动时可以自动的载入 DLM，只需将 dll 和 dlm 两个文件拷贝到 bin 目录，而且使用的时候 DLM 中的函数或方法 IDL 可以直接的进行调用，故此种方式是扩展 IDL 功能的最好的方式。

例编写一个对灰度图像做线性拉伸的动态库文件和 DLM 文件，流程如下：

12.1.4.1 编写动态连接库和 DLM

1) 打开 VC++编译器
2) 创建一个 Win32 Dynamic-Link Library

工程名字：imageStretch；

动态库类型选择：An empty DLL project。

3) 编写 ImageStretch.dlm 文件（ASCII 文件）“//”中为注释！

```
MODULE ImageStretch
//MODULE 模块名称-只能占一行
DESCRIPTION LineFunction,args [byteValue，Width，Length，minpixel，maxpixel]
//描述信息
VERSION 1.0
SOURCE DYQ
BUILD_DATE 2007-8-23
PROCEDURE IMAGESTRETCH 5 5
//格式为：FUNCTION RtnName[MinArgs][MaxArgs][Options…]
//格式为： PROCEDURE RtnName[MinArgs][MaxArgs][Options…]
```

4) 将 X:\....\external 目录下的 export.h 加入到工程中。

5) 将 X:\...\bin\bin.x86 目录下的 IDL.lib 加入到工程中。

6) 建立文本文件 imageStretch.def 添加到工程中，并编辑输入以下内容：

```
LIBRARY ImageStretch
DESCRIPTION 'ImageStretch'
EXPORTS IDL_Load @1
```

7) 建立 imageStretch.h 加入工程中并编辑：

```
#ifndef IDL_C_H
#define IDL_C_H
#include "export.h"
/* 信息数量.....*/
#define imageStretch_ERROR 0
#define imageStretch_NOSTRINGARRAY -1

/*宏*/
#define ARRLEN(arr) (sizeof(arr)/sizeof(arr[0]))

extern IDL_MSG_BLOCK msg_block;

/*定义将 C 函数添加到 IDL 中的功能和退出*/
extern void IMAGESTETCH_exit_handler(void);
extern int IMAGESTETCH_Startup(void);

#endif
```

8) 编写 imageStretch.c 函数加入工程中

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include "imageStretch.h"
#include "export.h"
```

```
//初始化定义错误变量
static IDL_MSG_DEF msg_arr[]=
{
    {   "imageStretch_ERROR",                   "%NError:%s."},
    {   "imageStretch_NOSTRINGARRAY",           "%Nstring arrays not
allowed %s"}
};

IDL_MSG_BLOCK msg_block;

int IDL_Load(VOID)//IDL 接口
{

    if(!(msg_block = IDL_MessageDefineBlock("imageStretch",ARRLEN(msg_arr),
        msg_arr))) {
        return IDL_FALSE ;
    }
    // 调用 IDLtoC_scalarExamplesStartup()
    if(!IMAGESTETCH_Startup()) {
        IDL_MessageFromBlock(msg_block,imageStretch_ERROR,
            IDL_MSG_RET,"不能初始化程序");
    }
    return IDL_TRUE;
}
```

9) 创建 imageStretch_ExamplesStartup.c 添加到工程中，源码为

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include "imageStretch.h"
#include "export.h"

// 过程声明
extern void IDL_CDECL IMAGESTRETCH_STRETCH(int argc, IDL_VPTR argv[], char *argk);

static char statusBuffer[256];
//这里的 imageStretch 和 DLM 文件中的过程名字一定要一致，
//是 IDL 调用 DLM 的接口命令
static IDL_SYSFUN_DEF2 imageStretch_procedures[] = {
    {(IDL_FUN_RET) IMAGESTRETCH_STRETCH, "IMAGESTRETCH", 5, 5, 0, 0},
};

#if defined(WIN32)
#include <windows.h>
#endif

int IMAGESTETCH_Startup(VOID)
{
```

```
        //procedures 的运行
        if (!IDL_SysRtnAdd(imageStretch_procedures, FALSE,
ARRLEN(imageStretch_procedures)))
        {
            return IDL_FALSE;
        }

        IDL_ExitRegister(IMAGESTETCH_exit_handler);
        return(IDL_TRUE);

    }

    VOID IDL_CDECL IMAGESTRETCH_STRETCH(int argc,IDL_VPTR argv[],char *argk)
    {
            int numElements,width,height,minpixel,maxpixel;
            int i;
            byte *pInArray;

            IDL_ENSURE_ARRAY(argv[0]);

            //存储元素的总个数
            numElements =argv[0]->value.arr->n_elts;
            // ndims = argv[0]->value.arr->n_dim;

            IDL_ENSURE_SCALAR(argv[1]);
            width = IDL_LongScalar(argv[1]);

            IDL_ENSURE_SCALAR(argv[2]);
            height = IDL_LongScalar(argv[2]);

            IDL_ENSURE_SCALAR(argv[3]);
            minpixel = IDL_LongScalar(argv[3]);

            IDL_ENSURE_SCALAR(argv[4]);
            maxpixel = IDL_LongScalar(argv[4]);

            //获取输入的 byte 数据
            pInArray = argv[0]->value.arr->data;

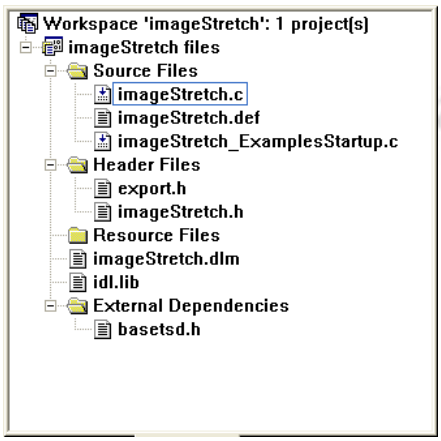
            if(minpixel<0) minpixel=0;
            if(maxpixel>255) maxpixel=255;

            for(i=0;i<width*height;i++)
            {
            float temp=((float)pInArray[i]-minpixel)/(maxpixel-minpixel)*255;
            if(temp<0) temp=0.0;
            if(temp>255) temp=255.0;
            //计算后赋值
            pInArray[i]= (byte)temp;
```

```
        }

    }
    // 当 IDL 关闭时调用-析构函数
    VOID IMAGESTETCH_exit_handler(VOID)
    {
        //什么都不做
    };
```

10) 最后该工程的结构如下图：



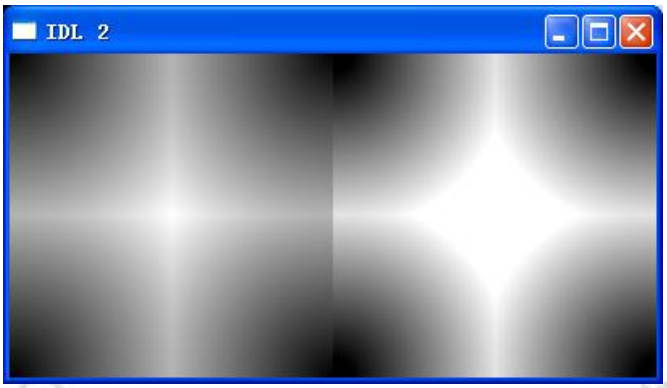
编译运行该工程，Debug 目录下会生成 imageStretch.dll。

12.1.4.2 IDL 调用 DLM

首先拷贝前面工程中 Debug 目录下的 imageStretch.dll 和工程目录下的 imageStretch.dlm 到 X:\...\bin\bin.x86 下，然后启动 IDL，在 IDL 命令行中写如下代码，测试 DLM：

```
    IDL> help,/dlm,'imageStretch'
    ** IMAGESTRETCH - DESCRIPTION LineFunction,args [byteValue£?Width£?Length
£?minpixel£?maxpixel] (not loaded)
        Version: 1.0, Build Date: 2007-8-23, Source: DYQ
        Path: C:\Program Files\ITT\IDL708\bin\bin.x86\imageStretch.dll
    % Compiled module: TEST_USINGDLM.
    % Compiled module: TEST_USINGDLM.
    % Compiled module: DIST.
    % Loaded DLM: IMAGESTRETCH.
    IDL> help,/dlm,'imageStretch'
    ** IMAGESTRETCH - DESCRIPTION LineFunction,args [byteValue£?Width£?Length
£?minpixel£?maxpixel] (loaded)
        Version: 1.0, Build Date: 2007-8-23, Source: DYQ
        Path: C:\Program Files\ITT\IDL708\bin\bin.x86\imageStretch.dll
```

运行程序 test_usingDLM 对拉伸效果进行检验。



12.2 其他语言与 IDL

12.2.1 对象输出助手

可以将 IDL 的数据可视化与分析的功能添加到其他应用软件中，能够很容易地将 IDL 对象输出到 Java 和 COM 环境，可以通过程序的自动化处理直接缩短接口开发。

12.2.1.1 输出为 COM 组件

12.2.1.1.1 可视化控件 ocx 输出

将 traning\chapter12 目录下的三个类文件 idlitwindowexample__define.pro、idlitdirectwindowexample__define.pro 和 idlgrwindowexample__define.pro 输出成 COM 组件并在.net 环境下调用步骤如下：

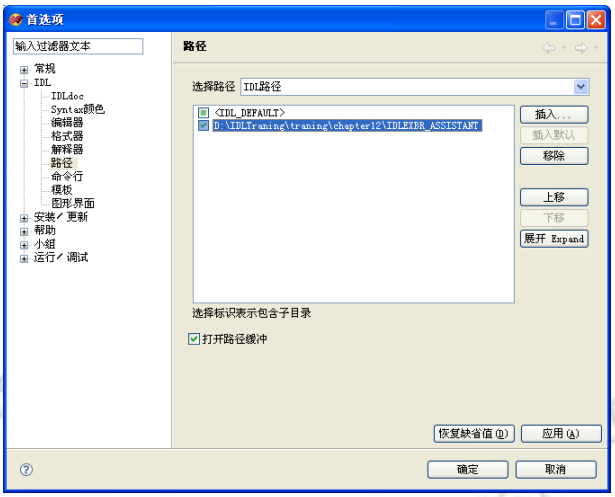
1) 确保导出对象类程序能正确的运行；
2) 设置 IDL 源码路径

源码路径的解决方案有两种

（1）文件放在 IDL 安装路径的相关目录下，如..\IDL708\examples\doc\bridges ；

（2） 将源码所在路径添加到 IDL 的路径配置下，添加方法为：

菜单[窗口]→[首选项]窗口中左边[IDL]→[路径]下添加 pro 所在的路径并进行勾选。



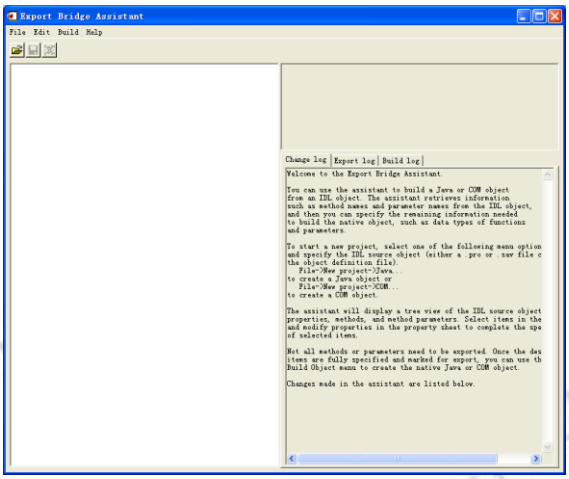
3) 添加编译器路径到环境变量

在系统变量中添加.net 编译器的路径，我的电脑右键属性 高级→环境变量 →系统变量 path 中添加： C:\Program Files\Microsoft Visual Studio 8\Common7\IDE



4) 启动导出助手

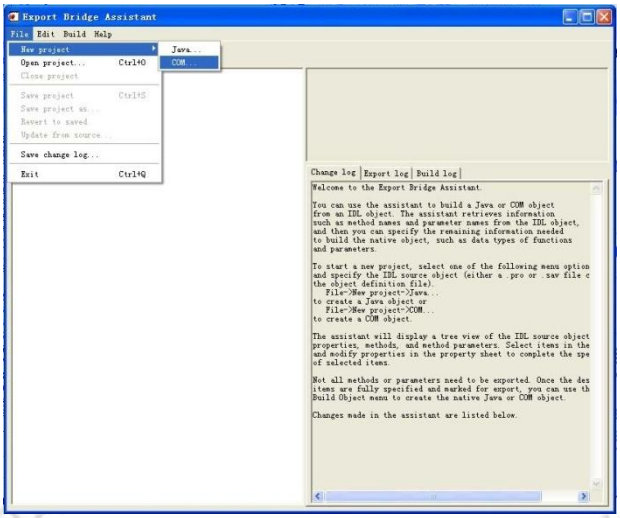
在IDL命令行中输入IDLEXBR_ASSISTANT回车，启动输出助手界面。



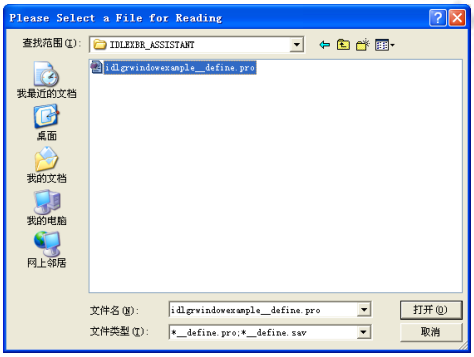
5) 导出 IDL 对象类

以 IDLgrWindowExample__define 为例进行 COM 导出。

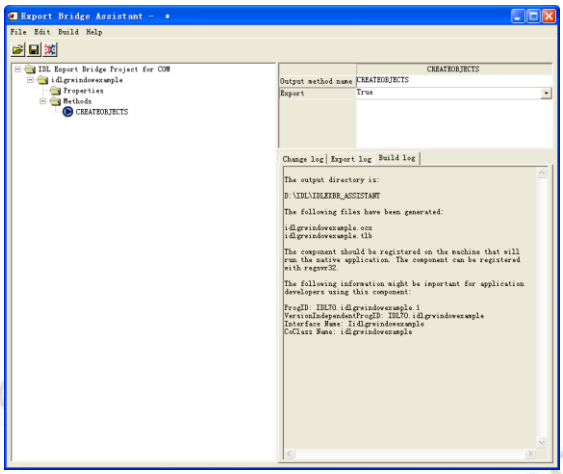
（1） 在输出助手界面中新建 COM 工程。



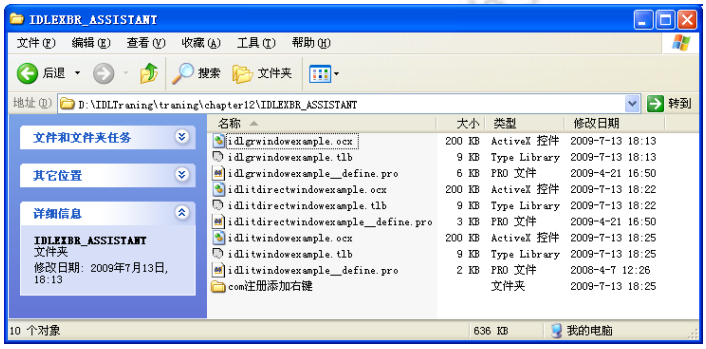
（2） 选择 pro 文件



（3） 设置输出参数

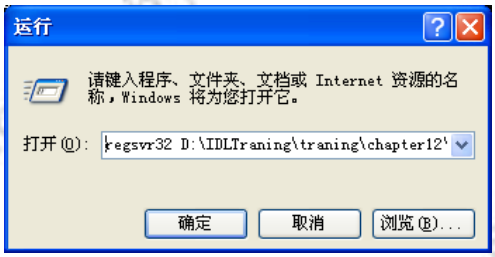


同样方式进行另外两个对象的导出，最终的目录如下图所示

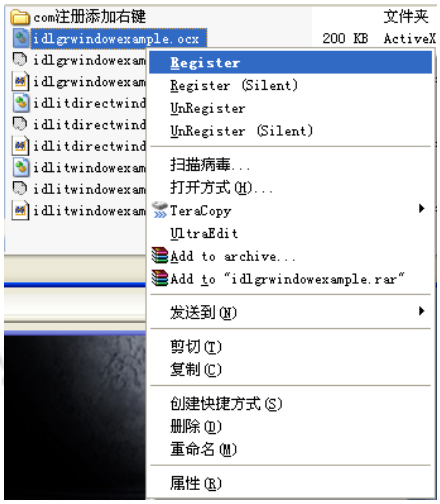


6) 对 com 组件进行注册

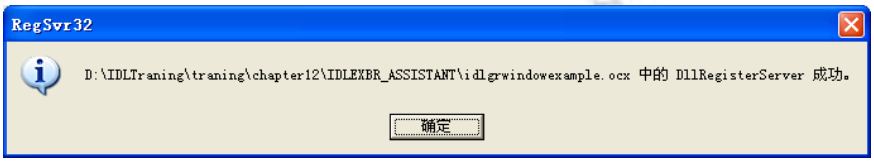
一是用 regsvr32 命令在运行中进行，如下图



二是运行目录下的[com 注册添加右键]下的 register_in_menu.reg，然后在组件的右键菜单中会添加一个选项 Register，点击即可。

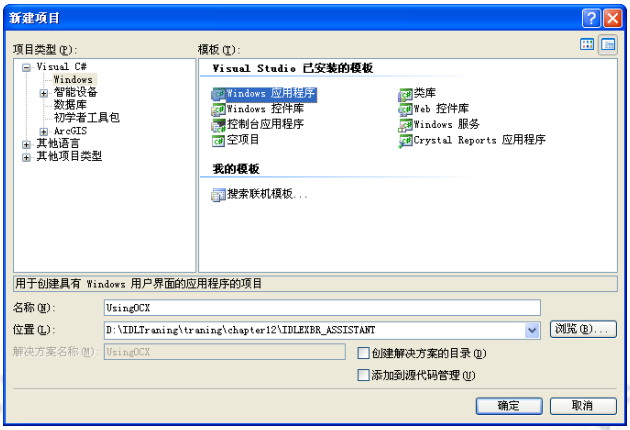


组件提示注册成功。

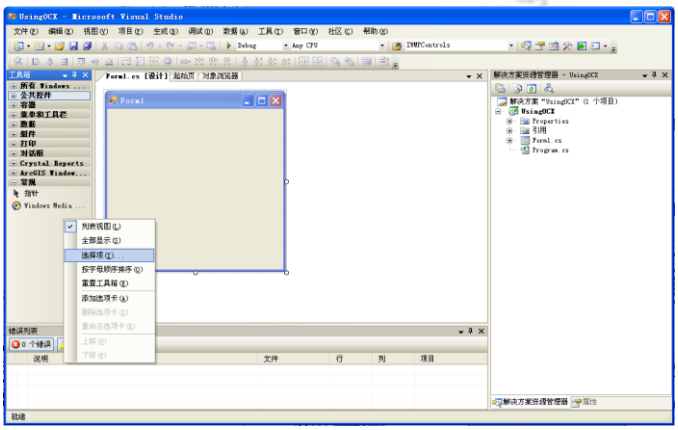


依次对其他两个组件进行注册。

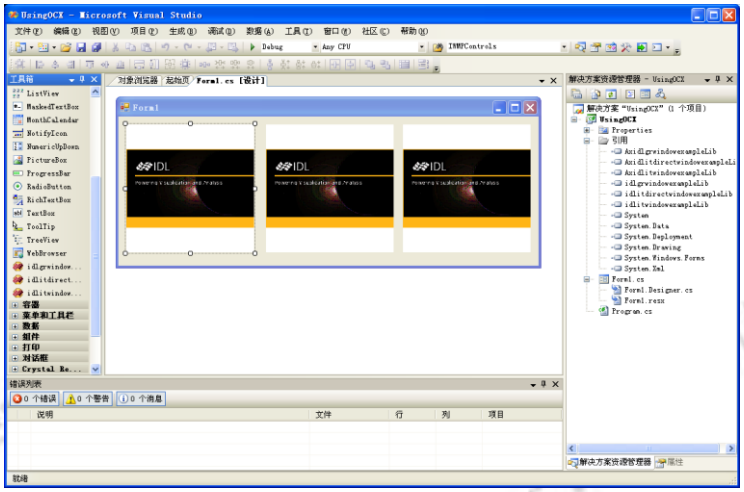
7) 在.net 环境下新建一 Windows 标准程序



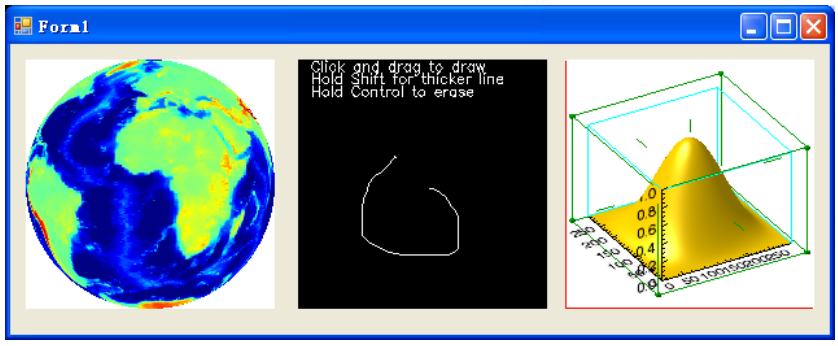
8) 添加刚刚注册的组件到控件列表中



9) 将控件拖放到界面上并进行排列

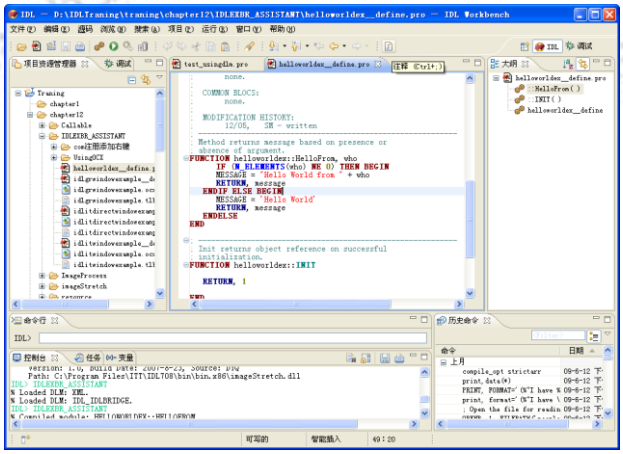


10) 运行效果如下

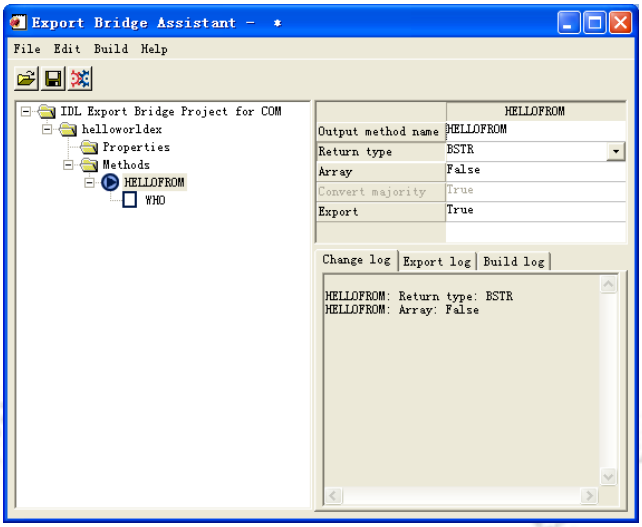


12.2.1.1.2 DLL 组件

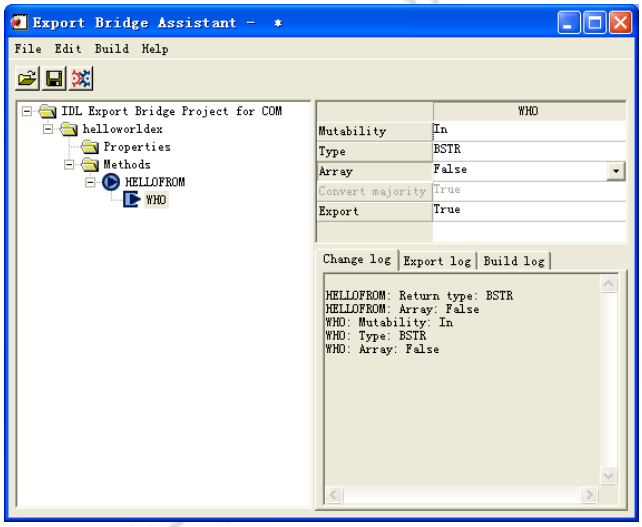
以 helloworldex__define.pro 为例看一下导出为 dll 并进行数据传递的例子。查看分析下源码中的方法。



1) 在输出助手界面中新建 COM 工程。



2) 设置 HelloFrom 的输出参数

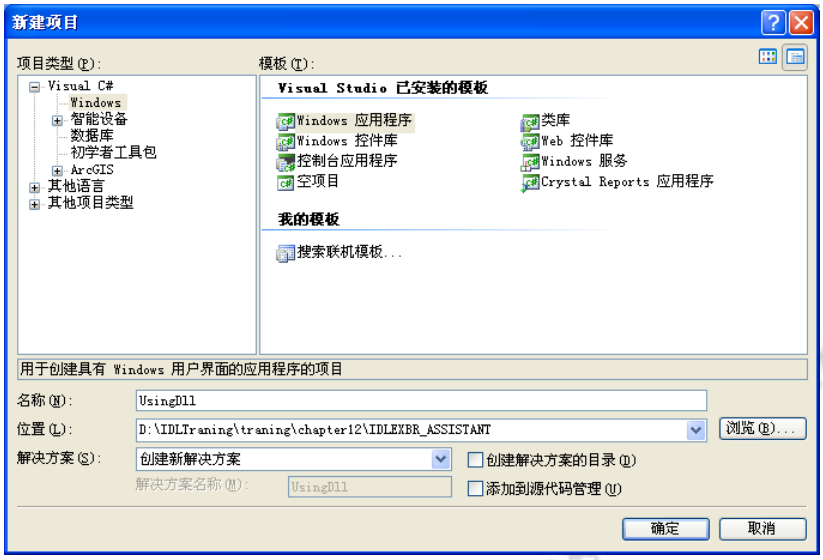


3) 发布后注册注册组件

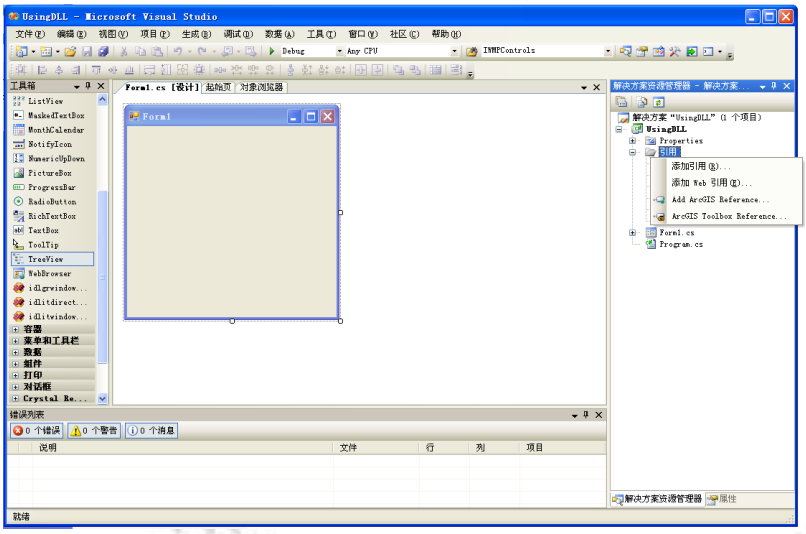


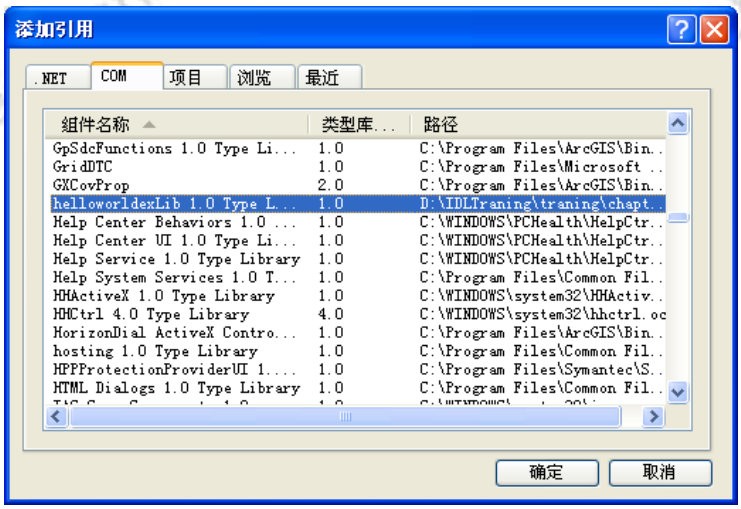
4) 调用组件

1) 新建 Windows 应用程序

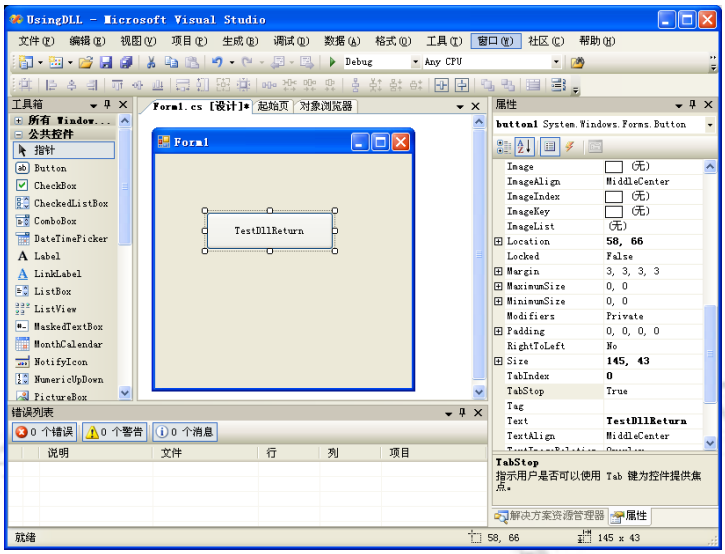


2) 添加 dll 的引用





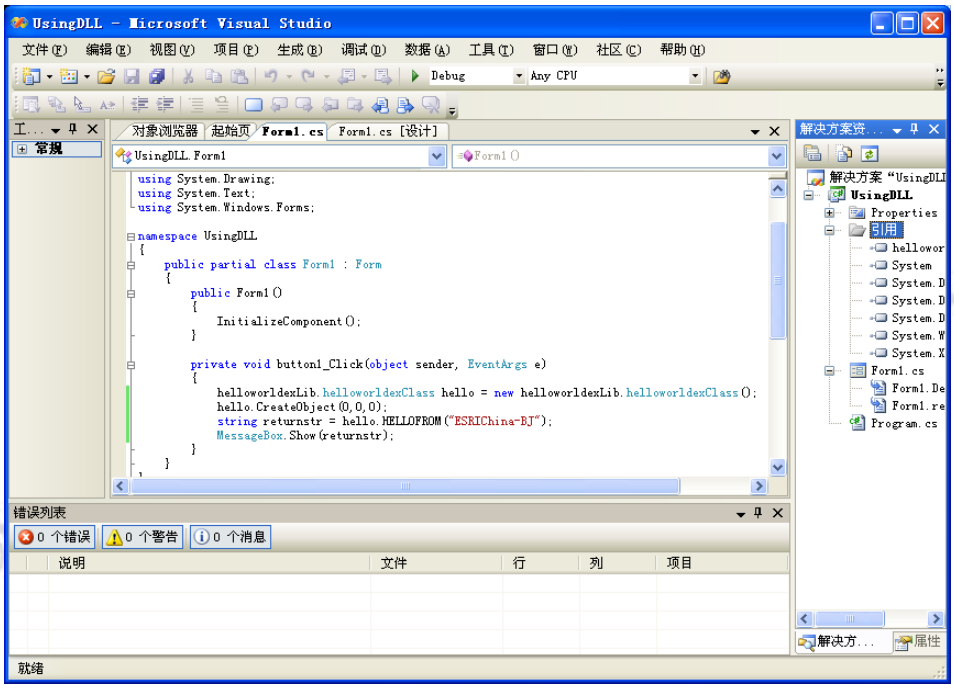
3) 在程序中添加一按钮 Button



4) 在 button 事件中添加下列语句

```
helloworldexLib.helloworldexClass hello = new helloworldexLib.helloworldexClass();
hello.CreateObject(0,0,0);
string returnstr = hello.HELLOFROM("ESRIChina-BJ");
MessageBox.Show(returnstr);
```

添加代码后程序界面



5) 运行程序



6) 点击 button，正确返回信息



12.2.1.2 输出为 JAVA 类

与 COM 类似，可参考 IDL Programmers' Guides > IDL Connectivity Bridges > Part II: Exporting from IDL-Using Exported Java Objects

12.2.2 IDLDrawWidget Activex 方式

IDLDrawWidget Activex 方式是之前版本调用 IDL 的主要方式，逐渐被 com 组件方式替代了。但其简单实用，仍然是开发混合调用系统常用的方法。

12.2.2.1 该方法具备下面的特点：

1) 能够通过控件直接显示直接图形或对象图形；
2) 自身提供了鼠标事件的响应；
3) 能够与外部进行数据的交换传输；

12.2.2.2 数据传递方法

通过 SetNamedData 在 C#下进行 IDL 中变量的创建；

通过 SetNamedArray 在 C#下进行 IDL 中变量数组的创建；

通过 GetNamedData 在 C#下进行 IDL 中变量的读取；

ExecuteStr 进行命令行的执行，同时也能传递数据；

C#下与 IDL 进行数据传递示例源码

```
//字节型数据
```

```
byte[,] var = new Byte[200, 200];
byte varv = 3;
this.axIDLDrawWidget1.SetNamedData("data", varv);
this.axIDLDrawWidget1.ExecuteStr("help,data,out = o");
this.axIDLDrawWidget1.ExecuteStr("s = dialog_message(o,title ='字节变量')");

this.axIDLDrawWidget1.SetNamedArray("data", var, false);
this.axIDLDrawWidget1.ExecuteStr("help,data,out = o");
this.axIDLDrawWidget1.ExecuteStr("s = dialog_message(o,title ='字节数组')");

//短整型
short[,] svar = new short[200, 200];
short svarv = 3;
this.axIDLDrawWidget1.SetNamedData("data", svarv);
this.axIDLDrawWidget1.ExecuteStr("help,data,out = o");
this.axIDLDrawWidget1.ExecuteStr("s = dialog_message(o)");

this.axIDLDrawWidget1.SetNamedArray("data", svar, true);
this.axIDLDrawWidget1.ExecuteStr("tv,data");
this.axIDLDrawWidget1.ExecuteStr("help,data,out = o");
this.axIDLDrawWidget1.ExecuteStr("s = dialog_message(o)");
//长整型数据
int[,] ivar = new int[200, 200];
int ivarv = 3;
this.axIDLDrawWidget1.SetNamedData("data", ivarv);
this.axIDLDrawWidget1.ExecuteStr("help,data,out = o");
this.axIDLDrawWidget1.ExecuteStr("s = dialog_message(o)");

this.axIDLDrawWidget1.SetNamedArray("data", ivar, true);
this.axIDLDrawWidget1.ExecuteStr("tv,data");
this.axIDLDrawWidget1.ExecuteStr("help,data,out = o");
this.axIDLDrawWidget1.ExecuteStr("s = dialog_message(o)");
//浮点型数据
float[,] fvar = new float[200, 200];
float fvarv = 3;
this.axIDLDrawWidget1.SetNamedData("data", fvarv);
this.axIDLDrawWidget1.ExecuteStr("help,data,out = o");
this.axIDLDrawWidget1.ExecuteStr("s = dialog_message(o)");

this.axIDLDrawWidget1.SetNamedArray("data", fvar, true);
this.axIDLDrawWidget1.ExecuteStr("tv,data");
this.axIDLDrawWidget1.ExecuteStr("help,data,out = o");
this.axIDLDrawWidget1.ExecuteStr("s = dialog_message(o)");
//双精度型数据
```

```
double[,] dvar = new double[200, 200];
double dvarv = 3;
this.axIDLDrawWidget1.SetNamedData("data", dvarv);
this.axIDLDrawWidget1.ExecuteStr("help,data,out = o");
this.axIDLDrawWidget1.ExecuteStr("s = dialog_message(o)");

this.axIDLDrawWidget1.SetNamedArray("data", dvar, true);
this.axIDLDrawWidget1.ExecuteStr("tv,data");
this.axIDLDrawWidget1.ExecuteStr("help,data,out = o");
this.axIDLDrawWidget1.ExecuteStr("s = dialog_message(o)");

//字符串类型
string[,] strvar = new string[200, 200];
string strvarv = "esri";
this.axIDLDrawWidget1.SetNamedData("data", strvarv);
this.axIDLDrawWidget1.ExecuteStr("help,data,out = o");
this.axIDLDrawWidget1.ExecuteStr("s = dialog_message(o)");

//测试数据读出
this.axIDLDrawWidget1.ExecuteStr("data = 'esrichina-bj'");

string s = (string)this.axIDLDrawWidget1.GetNamedData("data");
MessageBox.Show(s);

this.axIDLDrawWidget1.ExecuteStr("out = 123");
Int16 ints = (Int16)this.axIDLDrawWidget1.GetNamedData("out");
MessageBox.Show(ints.ToString());

this.axIDLDrawWidget1.ExecuteStr("fout = 123.456");
float fints = (float)this.axIDLDrawWidget1.GetNamedData("fout");
MessageBox.Show(fints.ToString());

//测试变量是否存在
bool t = this.axIDLDrawWidget1.VariableExists("data");
MessageBox.Show(t.ToString());
t = this.axIDLDrawWidget1.VariableExists("data3");
MessageBox.Show(t.ToString());
```

12.2.2.3 C#下的使用步骤：

1) 建立一 windows 应用程序；
2) 在工具箱上右键→选择项→COM 组建选中；
3) 拖动 IDLDrawWidget Control 3.0 控件到窗体上 ；

4) axIDLDrawWidget1.IdlPath = 设定 IDL 库文件目录；

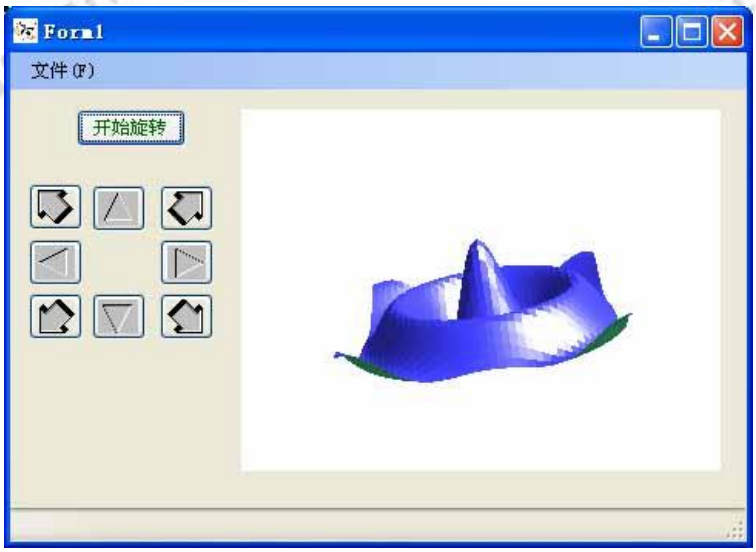
5) 初始化 IDL 控件

Int n= axIDLDrawWidget1.InitIDL((int)this.Handle);

axIDLDrawWidget1.CreateDrawWidget();

axIDLDrawWidget1.ExecuteStr(“”);执行 IDL 命令

例子源码 WindowsApplication

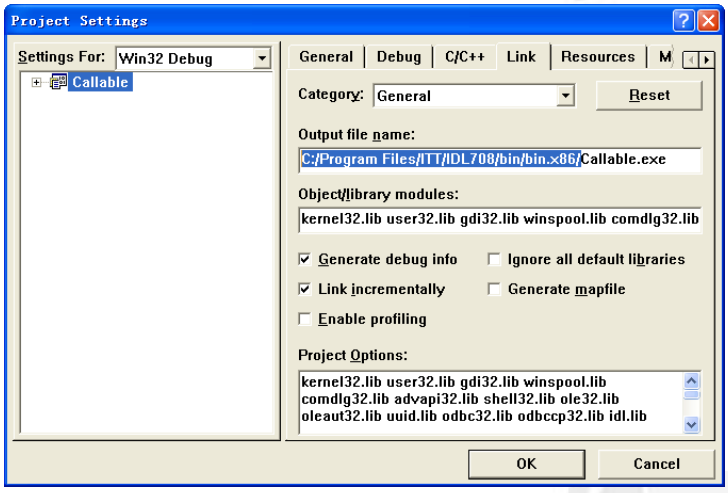


12.2.3 Callable IDL

Callable 技术是指的利用动态链接库的方式调用 IDL 的方法。通过该技术，外部程序可以像在 IDL 环境中一样执行命令行语句和调用相关程序或函数。

1) 建立一个工程

程序中添加 IDL_Expot.h，指定输出目录到 IDL 的路径\bin\bin.x86 下，这样运行的时候不需要拷贝动态库到输出目录中，如下图：



2) 初始化 IDL

int IDL_Initialize(IDL_INIT_DATA *init_data)

3) 执行命令

int IDL_Execute(int argc, char *argv[]);

其中： argc 是指在 argv[]中的命令数；argv[]是由 IDL 命令字符串组成的字符串数组。

int IDL_ExecuteStr(char *cmd)；其中： cmd 是一个 IDL 命令字符串。

4) 退出 IDL

int IDL_Cleanup(int just_cleanup);

参考 Demo[chapter12\Callable\ Callable.dsw]

bbs.esrichina-bj.cn

13 ENVI 功能扩展

ENVI 作为功能强大的遥感软件，扩展 ENVI 的功能包括创建波段和波谱数学函数、自定义数据输入、交互式用户程序和 ENVI 二次开发等。

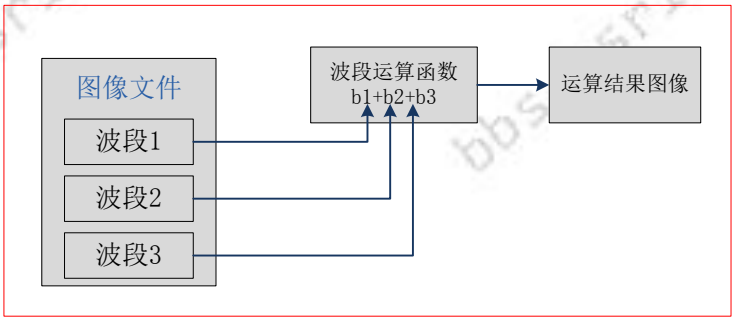
ENVI 的扩展，包括波段和波谱运算函数，自定义的空间、波谱，或是感兴趣区域（ROI）的处理，用户函数，自定义文件输入程序，批处理，以及其它如报告和绘图工具等。ENVI 提供了一系列工具为程序员使用，能够极大地简化自定义程序的开发，并保持和 ENVI 一致的外观。

13.1 ENVI 波段运算和波谱运算

对于用户来说，扩展 ENVI 的功能最简单方法就是使用波段和波谱运算函数。用户可以交互式通过波段和波谱运算表达式对话框输入大多数的波段和波谱运算函数。波段运算函数允许用户从任何波段或是文件中输入数据，处理数据，并输出整个波段。波谱运算函数允许用户从图表或文件中输入波谱数据，处理数据，并输出一个波谱。这种模式下，所有的数据的输入输出、获取、波段或波谱函数的调用及结果的输出都由 ENVI 中控制。因此使用波段或波谱函数进行处理时，用户不需要修改菜单、不必创建参数输入部件，不用执行 I/O 操作，只需在用户编写的函数中提供数据的计算处理功能。

13.1.1 波段运算

下图通过三个波段的相加说明了波段运算的处理过程。表达式中的每一个波段都对应于一个输入的影像波段，对这三个波段求和并输出结果影像。



这个表达式中的每个变量不仅可以对应于单一波段，也可以对应与一个文件。例如：在表达式 b1+b2+b3 中，如果 b1 映射为文件而 b2,b3 映射为单一波段则结果为 b1 所对应的文件的所有波段分别和 b2、b3 进行求和。

波段运算可以方便的进行特定波段的自定义处理，快速高效的使用波段运算需要用户能灵活的应用 IDL 下的数学运算，参考下表运算符。

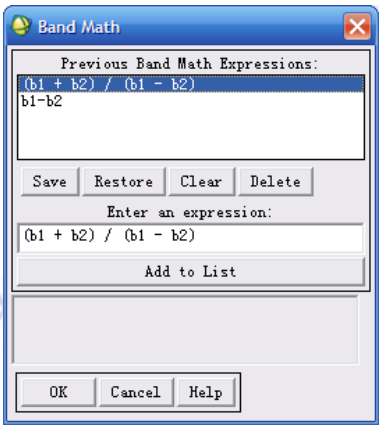
Series and Scalar 数学	三角函数	其它波段运算选项
加 （+）	正弦 （sin（x））	关系运算符 （EQ、NE、LE、LT、GE、GT）
减 （-）	余弦 （cos（x））	逻辑运算符 （AND、OR、XOR、NOT）
乘 （*）	正切 （tan（x））	类型转换函数（byte， fix， long，float， double， complex）
除 （/）	反正弦 （asin（x））	IDL 返回数组结果的函数
最小运算符 （<）	反余弦 （acos（x））	IDL 返回数组结果的程序
最大运算符 （>）	反正切 （atan（x））	User IDL 函数和程序
绝对值 （abs（x））	双曲正弦 （sinh（x））	
平方根 （sqrt（x））	双曲余弦 （cosh（x））	
指数 （^）	双曲正切 （tanh（x））	
自然指数 （exp（x））		
自然对数 （alog（x））		
以 10 为底的对数 （alog10（x））		

波段运算时变量的定义必须以“b”或“B”开头，后面为数字且数字位数不超过 5。ENVI 波段运算工具的使用有两种方式，数学表达式和波段运算函数。

13.1.1.1 数学表达式

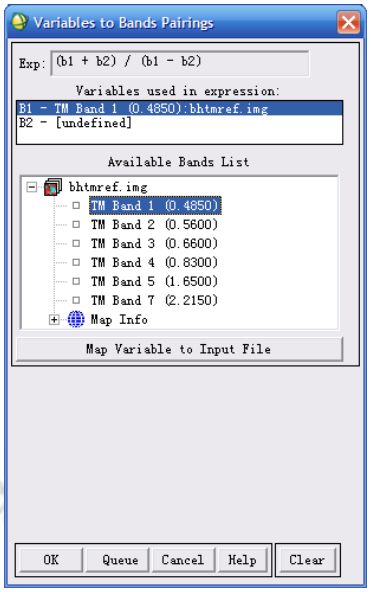
编写数学表达式进行波段运算的步骤如下：

1) 在 ENVI 主菜单下，选择[Basic Tools]→[Band Math]，在弹出的 BandMath 界面中，在[Enter an Expresion]里面输入表达式。



2) 点击 Add to list 按钮，然后在[Previous Band Math Expressions]里面选择输入的表达式点击 OK。

3) 在弹出的界面中选择表达式中变量赋需要计算的波段。



13.1.1.2 波段运算函数

编写波段运算函数跟数学表达式相比，在操作上更灵活，函数中可以添加更多的控制，如运算中的错误检测等。

波段运算函数的编写非常简单，接受输入波段，处理数据，并返回结果。以下面的方式定义：

```
Function bm_func,b1,[b2,…,bn, parameters and keywords]
    processing steps
    return,result
end
```

以编写（b1+b2）/ (b1-b2)功能为例，编写波段运算函数步骤如下：

1) 编写函数并保存

```
;函数定义为两个输入波段b1和b2以及一个check关键字
FUNCTION Bm_ratio, b1, b2, check=check
  ;计算差值
  den=FLOAT(b1)-b2
  ;如果设置了check关键字，检查被0除问题
  IF(KEYWORD_SET(check)) THEN ptr=WHERE(den EQ 0., count) $
  ELSE count=0
  IF (count GT 0) THEN den [ptr]=1.0
  ;继续计算比率结果
  result=(FLOAT(b1)+b2)/den
  IF(count GT 0) THEN result[ptr]=0.0
  ;返回结果
  RETURN, result
END
```

2) 编译函数

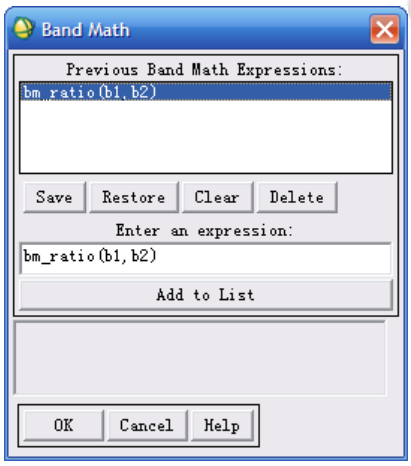
编译函数的方法有两种，一是直接编译 pro，一是拷贝函数或保存的 sav 文件到相应的目录下重启 ENVI+IDL。

（1） 在 ENVI 的主菜单下，选择[File]→[Compile IDL Module]，选择函数进行编译；

（2） 把 bm_ratio.pro 或 sav 文件放入 ENVI 安装目录下的..\envi46\save_add 下，重新启动 ENVI+IDL 即可。

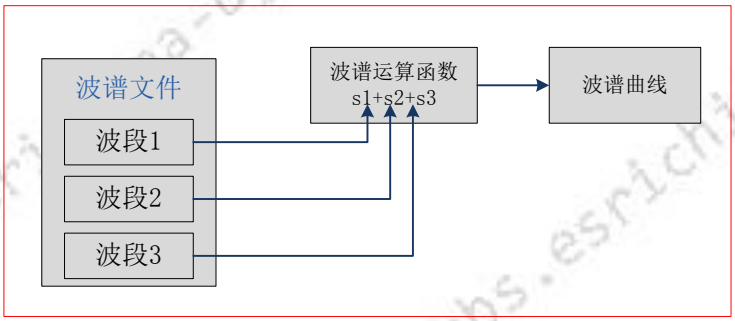
3) 执行函数

在 ENVI 主菜单下，选择[Basic Tools]→[Band Math]，在弹出的 BandMath 界面中，在[Enter an Expresion]里面输入函数调用点击[OK]执行。



13.1.2 波谱运算

下图三个波谱的相加来说明波谱运算的过程。表达式中的每一个变量都被映射为一个输入波谱，对这三个波谱求和并将结果输出到绘图窗口。表达式中的变量也可以映射为文件而不仅是单一波谱，在这种情况下，结果将是一幅新的影像。



波谱运算的操作步骤跟波段运算基本一致，不同之处有：

1) 变量命名以“S”或“s”开头；
2) 执行时 ENVI 主菜单下，选择[Basic Tools]→[Spectral Math]

13.1.3 几个例子

1) 波段中数据小于 0 的赋予 0

波段运算语句： b1>0

2) 波段中数据值小于 0 的赋予-999

波段运算语句： (b1 LT 0)*-999+(b1 GE 0)*b1

3) 三个波段求平均值，要求：如该波段小于 0 则不参加运算。即：某点 b1:4;b2:6;b3:0；那么平均值 ave = (b1+b2+b3)/(1+1);

波段运算语句：(b1>0+b2>0+b3>0)/(((b1 ge 0) + (b2 ge 0)+(b3 ge 0)) >1)

4) 两波段图像 b1 和 b2，波段 b1 中的云部分（象元值大于 200）用波段 b2 代替

波段运算语句：(b1 GT 200)*b2+(b1 LE 200)*b1

5) 对图像波段，需做以下运算，如 b1<0 则 b1=0，如 0<=b1<=10 那么 b1=b1*100，如 b1>10 则 b1=b1*10

波段运算语句： (b1 LE 0)*b1>0 +((b1 ge 0)and (b1 le 10))*b1*100+ (b1 gt 10)*10

13.2 用户函数

13.2.1 用户函数介绍

用户函数允许用户为 ENVI 添加新的功能并通过 ENVI 菜单进行访问。可以添加任意数量的用户函数，并且每个函数都可以获得它自己的菜单选项。当用户通过菜单选择该函数对应的菜单项时，将会执行这些函数，就如同 ENVI 的自身函数一样。

用户函数可以是 pro 或 sav，如是 pro 则与普通的 IDL 程序无任何区别，一般放到 ENVI 安装目录下的 Save_add 目录中，启动 ENVI+IDL 时用户函数可以被自动编译，单独启动 ENVI 时 pro 源码不被编译识别。如果是 sav 文件，启动 ENVI 即可识别。

用户函数 pro 代码可以随时进行修改，修改后 ENVI 只需重新编译函数即可，不需要重新启动。

13.2.2 ENVI 菜单介绍

ENVI 的菜单系统，包括主菜单和显示窗口菜单，是由 ENVI 安装目录下 menu 目录下的 envi.men 和 display.men 这两个 ASCII 码文件定义的。Windows 系统上 menu 的路径为 itt\idlxx\products\envixx\menu。

1) envi.men 文件结构

该文件定义了 ENVI 主菜单中的选项，display.men 文件定义了显示窗口菜单中的选项。每次 ENVI 启动的时候，这两个文件被读入并根据其内容构建 ENVI 的菜单。在菜单中添加内容，只需在这两个文件中添加相应的内容并重启 ENVI 即可。

使用任何文本编辑器就可以打开 envi.men 文件。在文件的顶部有些介绍的注释文本。注释结束后，就是如下的文件的结构：

```
0 {File}
   1 {Open Image File} {open envi file} {envi_menu_event}
```

1 {Open Vector File} {open vector file} {envi_menu_event}

1 {Open Remote File} {open remote file} {envi_menu_event}

1 {Open External File} {separator}

2 {Landsat}

3 {Fast} {open eosat tm} {envi_menu_event}

每一行开始的数据定义了菜单项的层次。0 表示最顶层，1 表示一级子菜单，2 表示二级子菜单，如此类推。

{Open Image File}第一个大括号括起来的部分定义了显示在菜单上的内容。

{open envi file}第二个大括号括起来的部分定义了为菜单项所赋给的用户值。用户值在同一用户函数处理多个菜单项时非常有用，可以区别那个菜单项被选中。

{envi_menu_event}第三个大括号定义了菜单项事件处理程序的名称，即编写的用户函数名。该处使用的是用户函数名，而不是用户函数所在的文件名，所以没有后缀。

2) 修改方法

一是可以直接修改菜单配置文件 envi.mem 中添加“自定义函数”和“文件信息”菜单选项。

二是用函数来修改，ENVI 自身有个 ENVI_DEFINE_MENU_BUTTON，是这么使用的，如扩展功能函数名称为 Envi_file_info,event，那么只需添加个功能过程 PRO Envi_file_info_define_buttons, buttonInfo，然后菜单的添加工作就可以在这个过程下完成了。

ENVI_DEFINE_MENU_BUTTON 的调用格式如下：

ENVI_DEFINE_MENU_BUTTON, ButtonInfo [, /DISPLAY], $ EVENT_PRO=string | /MENU | UVALUE=string [, POSITION=long integer or string] [, REF_INDEX=long integer] [, REF_UVALUE=variable], REF_VALUE=string [, SEPARATOR={0 | 1 | -1}] [, /SIBLING], VALUE=string

其中： Display-是创建在主菜单还是显示菜单；

VALUE -菜单的显示信息 -必须的；

UVALUE -必须的；

EVENT_PRO-程序事件名称；

Menu- 是否为菜单；

REF_INDEX-如 REF_VALUE 不唯一则可用此来设置；

REF_VALUE -作为 parent 的已存在菜单名；

SIBLING -是否创建同级菜单；

参考 Envi_file_info_define_buttons



13.2.3 ENVI 组件介绍

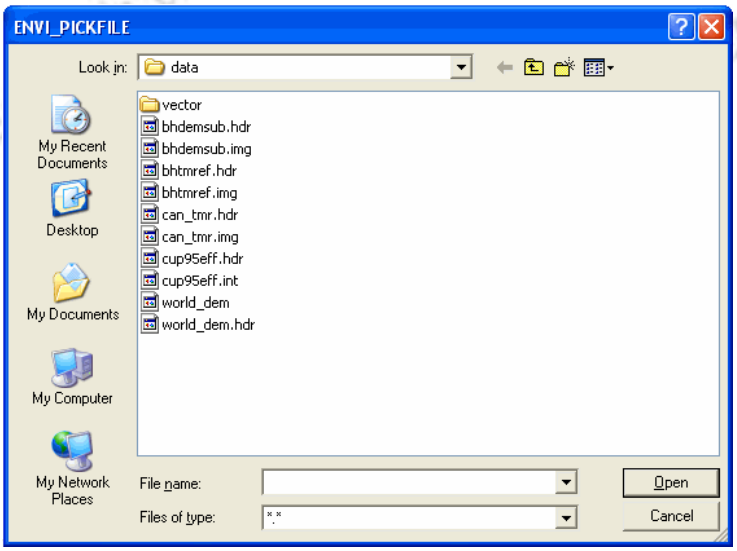
用户函数可以实现扩展 ENVI 的功能部分，如果实现仅仅是无用户交互的简单功能函数编写就足够了，但如果涉及到了用户的交互式操作，如参数输入等模块，就会显得复杂了。虽然 IDL 可以编写通常的组建来满足需求，但在 ENVI 的功能扩展下面事件的控制就显得不那么方便了。

ENVI 提供了以 WIDGET_开头的 20 多个功能组件来满足用户的需要，很多部件都包括“OK”和“Cancel”等按钮方便用户进行交互操作。

13.2.3.1 ENVI 下组件介绍

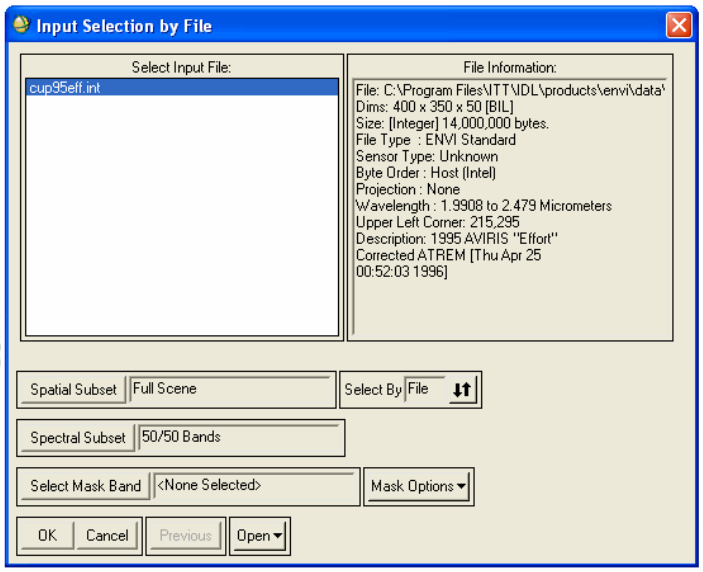
1) ENVI_PICKFILE 组件

该部件进行文件选择操作。

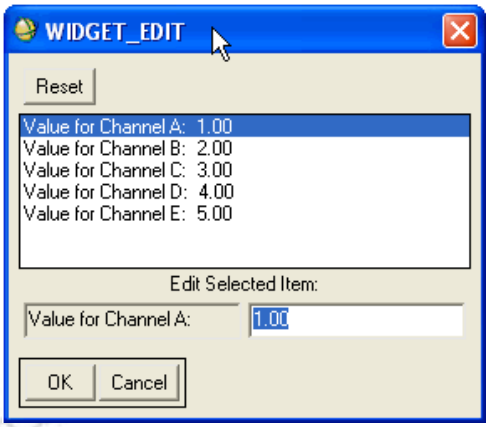


2) ENVI_SELECT 组件

于 ENVI 中一样，该部件提供文件的波段和范围选择操作。



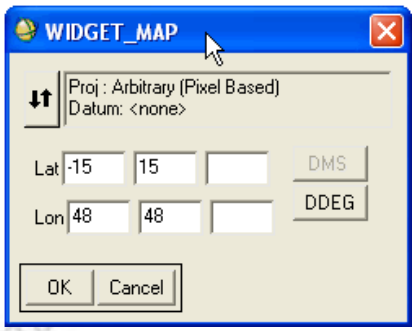
3) WIDGET_EDIT 组件



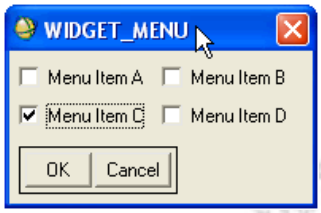
4) WIDGET_GEO 组件



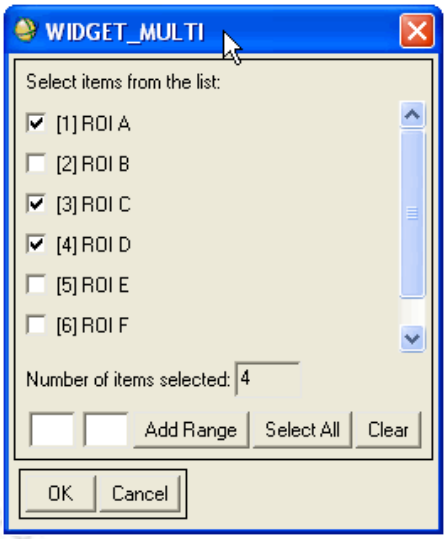
5) WIDGET_MAP 组件



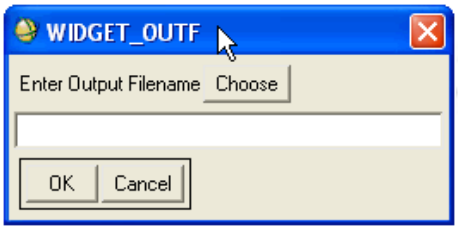
6) WIDGET_MENU 组件



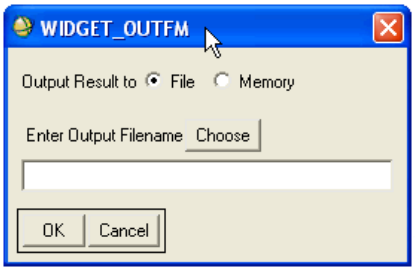
7) WIDGET_MULTI 组件



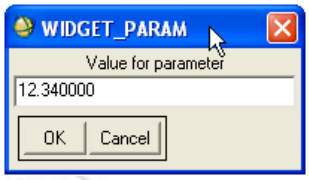
8) WIDGET_OUTF 组件



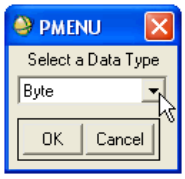
9) WIDGET_OUTFM 组件



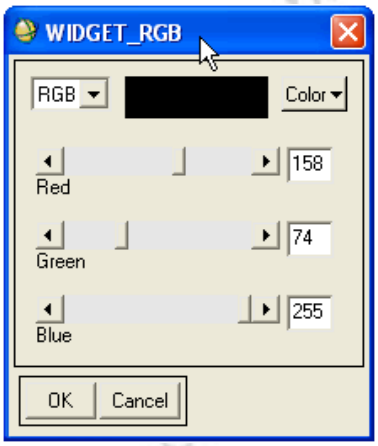
10) WIDGET_PARAM 组件



11) WIDGET_PMENU 组件



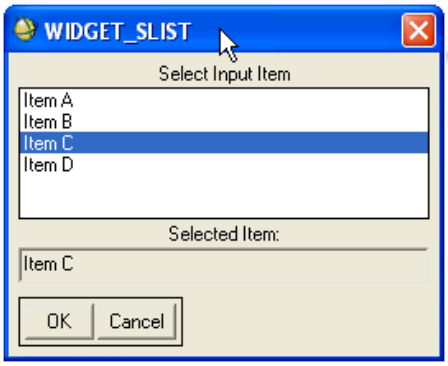
12) WIDGET_RGB 组件



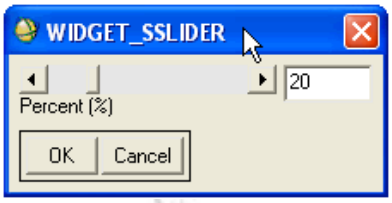
13) WIDGET_SLABEL 组件



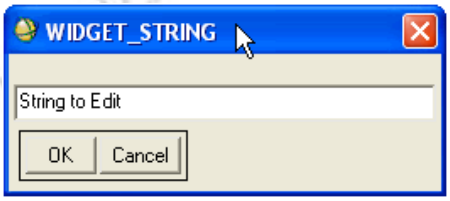
14) WIDGET_SLIST 组件



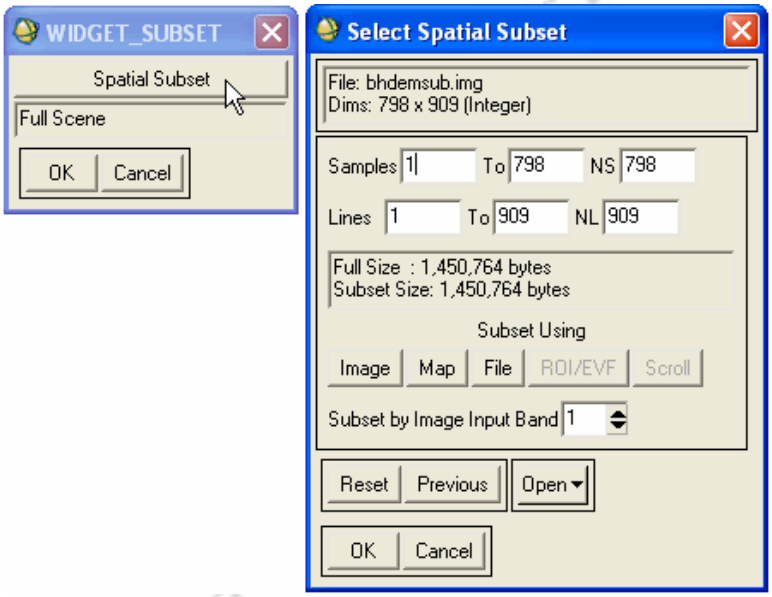
15) WIDGET_SSLIDER 组件



16) WIDGET_STRING 组件



17) WIDGET_SUBSET 组件



18) WIDGET_TOGGLE 组件



13.2.3.2 事件管理

IDL 编写程序时，界面组件要先用 Widget_Base 创建顶组件，然后需要用 XMANAGER 程序进行事件的传递和获取。

ENVI 下自动管理事件的组件容器则需要用 WIDGET_AUTO_BASE 来创建顶级容器（Top Level Base）和 AUTO_WID_MNG 函数进行事件的管理，此时组件需要添加关键字 AUTO_MANAGE。这种方式的应用使得 ENVI 下的函数扩展看起来比 IDL 简单的多。

13.2.4 编写实例

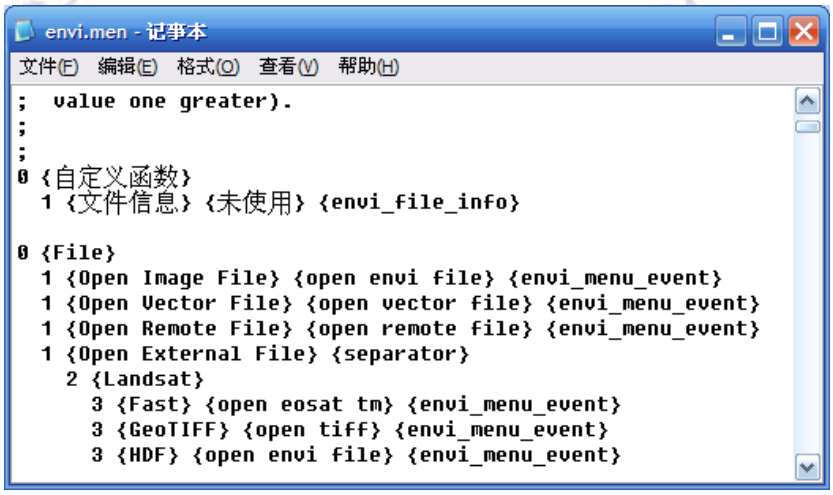
以两个例子的方式讲述了简单函数和界面组件函数扩展 ENVI 的步骤。

13.2.4.1 扩展 ENVI 简单函数编写

编写一个显示数据文件名及基本信息的功能模块。

1) 修改 ENVI 下的菜单

在 ENVI 的菜单配置文件 envi.mem 中添加“自定义函数”和“文件信息”菜单选项。



2) 编写功能函数

```
PRO Envi_file_info,event
  ;添加编译严格检查
  COMPILE_OPT STRICTARR
  ;选择文件
  ENVI_OPEN_FILE, fname, r_fid=fid
  ;如无效则返回
  IF fid[0] EQ -1 THEN BEGIN
    msg = DIALOG_MESSAGE('数据无效！',/Error)
    RETURN
  ENDIF
  ;数据信息查询
  ENVI_FILE_QUERY, fid, ns=ns, nl=nl, nb=nb, fname=fname
  ;提示
  msg = DIALOG_MESSAGE('文件名：'+ fName + STRING(13B)+   $
    '波段数：'+STRING(nb)+ STRING(13B)+ $
    '大小：'+STRING(ns)+'*'+STRING(nl),$
    /Information)
END
```

3) 添加功能函数

将功能函数添加到系统的\ save_add 下有两种方式，一是将 pro 源码文件拷贝到\ save_add 目录，一是保存为 sav 文件放入\ save_add 目录。

a) 源码文件(*.pro)保存 sav 文件的步骤

i. IDL 进程重启

IDL> .Reset_Session

ii. 编译函数

点击菜单[运行]→[编译*.pro]或鼠标点击工具栏上编译按钮 ；

按快捷键 Ctrl+F8 或在命令行中输入.COMPILE '*.pro'；

编译函数中调用到的函数：

IDL> Resolve_All, /continue_on_error, skip_routines='envi'

iii. 保存为 save 文件

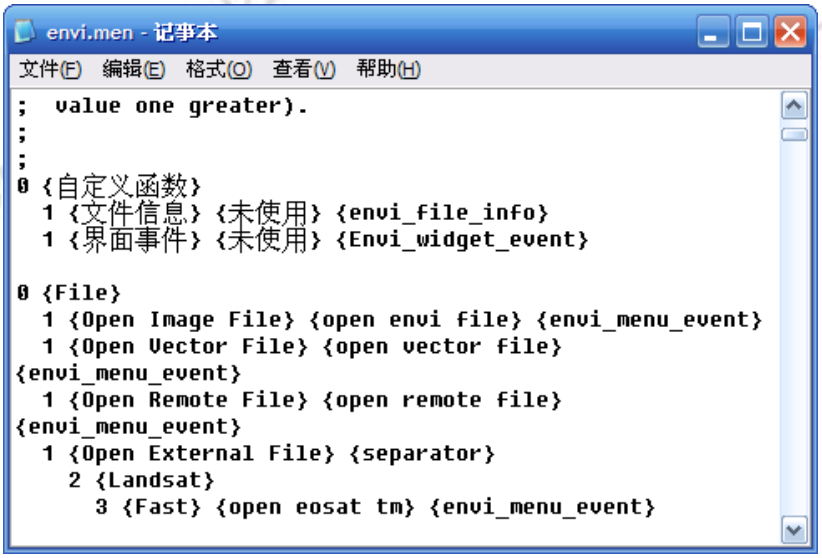
SAVE, file='*.sav', /routines

13.2.4.2 编写 ENVI 组件事件管理模块

编写一个显示数据文件名及基本信息的功能模块。

1) 修改 ENVI 下的菜单

在 ENVI 的菜单配置文件 envi.mem 中添加“自定义函数”和“文件信息”菜单选项。



2) 编写功能函数

```
PRO Envi_widget_event,event
  ;添加编译严格检查
```

```
    COMPILE_OPT STRICTARR
    ;选择文件
    ENVI_OPEN_FILE,file,r_fid = fid
    IF fid[0] EQ -1 THEN BEGIN
      msg = DIALOG_MESSAGE('数据无效！',/Error)
      RETURN;
    ENDIF
    ;创建 TLB
    tlb = WIDGET_AUTO_BASE(title='ENVI 组件事件测试')
    ;创建裁剪组件
    ws = WIDGET_SUBSET(tlb, uvalue='subset', fid=fid,$
      dims=dims, /auto)
    result = AUTO_WID_MNG(tlb)
    ;显示选择结果
    IF result.accept EQ 1 THEN BEGIN
      dims = result.subset
      msg = DIALOG_MESSAGE('选择了 OK!'+ STRING(13B)+$
        '是 ROI？'+ STRING(dims[0])+ STRING(13B)+$
        '采样起始列位置：'+ STRING(dims[0])+ STRING(13B)+$
        '采样终止列位置：'+ STRING(dims[1])+ STRING(13B)+$
        '采样起始行位置：'+ STRING(dims[2])+ STRING(13B)+$
        '采样终止行位置：'+ STRING(dims[3])+ STRING(13B), $
        /information)
    ENDIF ELSE BEGIN
      msg = DIALOG_MESAGE('选择了 Cancel',/information)
    ENDELSE
  END
```

3) 添加功能函数

将功能函数或生成的 sav 文件拷贝到系统的\ save_add 下。

13.3 ENVI 的二次开发

ENVI 的二次开发能够让用户在命令模式下使用 ENVI。这种方式在以下几种情况下非常有用：

1) 用户主要使用 IDL 工作但偶尔需要用到 ENVI 的函数；

2) 用户希望创建定制的应用程序其中混合了 IDL 代码和 ENVI 函数；

3) 用户希望进行大量的 ENVI 处理而无需人工干预时。

二次开发是通过一系列特定的函数库来执行 ENVI 的功能，因此为了获取 ENVI 库函数，有必要了解一下 ENVI 程序的结构。

ENVI 的功能函数分散在大约 50 个 IDL 的 二进制 save 文件中，这些二进制文件存放在 ENVI 安装路径下的 Save 目录下， 在 Windows 上这些 sav 文件存放在 ITT\idlxx\products\envixx\save 目录下。核心 save 文件包括了 ENVI 的基本功能函数，动态运行函数及 ENVI 函数运行所需的内部变量。

13.3.1 二次开发模式

ENVI 二次开发模式的初始化是恢复 ENVI 的二进制 save 文件并调用 ENVI 的命令 ENVI_BATCH_INIT。步骤如下：

1) 启动一个新的 IDL 编译器，并输入以下命令：

IDL>ENVI, /restore_base_save_files

如果无关键字 RESTORE_BASE_SAVE_FILES，最终会启动 ENVI；

2) 调用 ENVI_BATCH_INIT 程序

调用 ENVI_BATCH_INIT 等于启动了一个没有 GUI 的新 ENVI。

下面的是批处理初始化的例子：

```
pro envi_init
 envi,/restore_base_save_files
 envi_batch_init,log_file='batch.log'
end
```

如果希望在运行 ENVI_BATCH_EXIT 时不希望 IDL 也关闭，那么可以在 ENVI 的主菜单[File]→[Perference]→[Miscellaneous]选项，并确定 Exit IDL on Exit from ENVI 选项设置为 No。

13.4 常用关键字及函数

当您熟练使用 ENVI 二次开发时，将会注意到有几个关键字基本上所有的程序都是具有的，这些关键字涉及了文件的输入和输出。

13.4.1 FID

文件 ID（FID）是一个长整型的标量。FID 为 ENVI 的程序员提供了一个命名变量，可以被数个 ENVI 程序所使用，来打开或选择文件。ENVI 程序对该文件进行的所有操作都是通过 FID 完成的。但是，如果用户同时使用 IDL 直接读取文件，请注意 FID 和 LUN 不是等同的。

13.4.2 R_FID 和 M_FID

ENVI 处理程序所产生的影像结果也包括一个 R_FID，或者称为返回 FID 关键字。如果结果是存在内存中的，设置 R_FID 关键字是访问该数据的唯一方法。在掩模处理程序还包括一个 M_FID，或者称为掩模 FID 关键字，用于确定用作掩模波段的文件。

如果打开文件发生错误，FID 将被设置为-1。值为-1 的 FID 是无效的，不能进行进一步的处理。因此在 ENVI 中进行编程时，总是要检查 FID、R_FID 以及 M_FID 是否有效。当遇到一个无效的 FID，通常从当前的处理程序中返回。

13.4.3 DIMS

DIMS 关键字是一个 5 个元素长整型数组。它定义了处理数据的空间子集。当需要确定 FID 的时候，用户必须同时使用 DIMS 关键字确定该文件的空间子集。

DIMS[0]：用于存储指向一个打开的 ROI 区域的指针，仅在 ROI 被定义的时候使用，其它时候设为-1

DIMS[1]：采样的起始位置（一个基于 0 的 IDL 数组下标）

DIMS[2]：采样的终止位置

DIMS[3]：行的起始位置

DIMS[4]：行的结束位置

13.4.4 POS

POS 关键字定义了用于处理的波段位置。POS 关键字是一个变长的长整型数组。因为 ENVI 处理的文件可能具有多个波段，而使用 POS 矢量来确定用于处理的波谱子集。波段从 0 开始（band1=0,band2=1,…）；例如，要处理多波段文件的第三波段和第四波段数据，POS=[2,3]。

13.4.5 文件管理函数

ENVI 文件处理函数为程序员提供了相当大的灵活性。有以下的函数可供编程使用，用户可以根据所需的情况选择所需的函数。

13.4.5.1 ENVI_PICKFILE

ENVI_PICKFILE 函数产生一个提示用户选择文件的对话框。该函数产生的界面和使用 ENVI 主菜单选择 File->Open Image File 一样的界面。该函数并不真正的打开文件，它只是以字符串的形式返回用户所选择的全路径文件名。

13.4.5.2 ENVI_SELECT

ENVI_SELECT 产生对话框提示用户从 ENVI 中已经打开的文件中选择一个文件。该函数产生 ENVI 标准的文件选择对话框，其中包括空间和波谱子区裁剪按钮，以及掩模波段选取按钮。

13.4.5.3 ENVI_OPEN_FILE

该函数返回一个文件的 FID，它是打开 ENVI 文件的最直接和简单的方法。默认情况下它将文件信息添加到可用波段列表中，可以使用 NO_REALIZE 可以阻止文件信息加入到可用波段列表中。

注：如果可用波段列表已打开，该关键字无效。

13.4.5.4 ENVI_FILE_MNG

该函数可以打开、关闭或者删除硬盘上的文件。无需用户交互。

13.4.5.5 ENVI_GET_FILE_IDS

该函数返回所有当前打开的文件的 FID。

13.4.6 打开文件函数

ENVI 能够读取相当广泛的数据格式，虽然 ENVI_OPEN_FILE 仅能够打开具有 ENVI 头文件的影像文件。ENVI 也提供了一些特定的处理程序能够打开和返回外部格式的文件：

ENVI_OPEN_DATA_FILE

该函数打开 ENVI 所支持的外部文件（通过关键字指定文件类型）并返回 FID，无需用户交互。

13.4.7 获取数据函数

当影像文件非常大时，不适合使用 IDL 的 READU 命令将它全部读入到内存中。因此，ENVI 提供了两个处理函数能够以小的、易管理的数据块方式读取影像数据。这两个函数也提供了数据逻辑组织，一次一个波段或是一次光谱切片。

13.4.7.1 ENVI_GET_DATA

该函数从一个打开的文件中获取影像数据。它每次只返回某一波段的数据。如果所需的空间数据不止一个波段，必需多次调用该程序以获得该相应波段的数据。数据的范围由 DIMS 关键字控制。

13.4.7.2 ENVI_GET_SLICE

该函数从一个打开的文件中获取波谱影像数据，它返回影像某一行所有波段的数据值。结果以 BIP 或 BIL 的格式返回。

13.4.8 感兴趣区函数

很多 ENVI 函数提供了使用 ROI 进行影像的空间选取选项。在 ENVI 的函数中，DIMS 关键字用于定义空间选取。DIMS 变量的第一个元素称为 ROI 指针，如果它被定义，则表明影像是基于 ROI 进行空间选取的。如果 ROI 指针设置为-1，表示没有使用 ROI。用户可用使用 ENVI_GET_ROI_DIMS_PTR 正确的设置 ROI 指针。

13.4.9 数据生成函数

ENVI 影像格式可能是最简单的数据格式。它是二进制文件，栅格影像数据以二进制数据流方式按 BSQ、BIL 或是 BIP 的存储顺序存储。文件中只有影像数据，头信息没有嵌入到文件中。当使用 WRITEU 程序将二维或三维影像数据写入到磁盘时，IDL 自动生成二进制格式文件。ENVI 格式的影像文件可用使用任意的名称，并且无需扩展名。

同时每一个二进制文件都伴随有一个 ASCII 格式的头文件，这个头文件描述了影像的基本特征以及附加信息。为了 ENVI 能够识别这个头文件，它必需和影像文件具有同样的文件名，并以.hdr 做为扩展名。

13.4.9.1 将影像数据保存到内存

当用户函数的结果是包含在 IDL 数组中的影像数据时，这些数据可用以内存方式被 ENVI 所使用。

ENVI_ENTER_DATA

该函数将 IDL 数组中的数据输入到可用波段列表中，该程序自动的设置 ENVI 的头文件，该文件同样也存储在内存中，并返回内存影像的 FID。一旦影像出现的可用波段列表中，它就可以像其它 ENVI 影像一样使用，也能够被存入磁盘。

13.4.9.2 将影像数据存入硬盘

由于 IDL 的 WRITEU 函数能够产生 ENVI 格式的文件，ENVI 没有提供单独的程序来将 IDL 的数组写入到磁盘。可以直接使用 IDL 的 WRITEU 函数

```
OpenW, unit, 'new_envi_image_file.img', /Get_LUN
WriteU, unit, image_array
Free_LUN, unit
```

注：必需考虑到，要随影像文件同时写入 ENVI 的头文件。

13.4.9.3 ENVI_SETUP_HEAD

使用该函数写某个已存入磁盘的影像数据的 ENVI 头文件。使用 OPEN 关键字，允许将影像文件输入到可用波段列表。ENVI_SETUP_HEAD 函数也能够返回磁盘上影像文件的 FID。

13.4.9.4 添加投影坐标信息

对数据文件添加投影信息，也就是将投影信息写到 ENVI 数据的头文件中去。明确 ENVI 的头文件信息数据中投影坐标信息的内容和格式。

以一个全球的数据[180,90]为例，如写入 WGS-84 的经纬度投影信息，则需要明确下面的内容。

mc = [0.5D,0.5D,-180,90];这里的前面两个值代表的是图像的起点位置，[0.5,0.5]表示的是左上角第一个像素的中心位置为起点；[-180,90]表示左上角该点对应的地理坐标信息，我们想显示全球的数据，故设为[-180,90]。

ps = [2,2; 代表的是像素的分辨率，经纬度投影默认以度为单位，[180,90]的数据代表的是全球[36,180]，故设置为[2,2]。

创建经纬度的投影类型:proj = ENVI_PROJ_CREATE(/geographic)

创建经纬度的 map 信息: mapInfo = ENVI_MAP_INFO_CREATE($

name = 'Geographic', $

mc = mc, ps = ps, $

proj = proj, /GEOGRAPHIC)

后面就可以进行头文件信息的设置了，添加关键字 map_info 即可。

```
;图像分辨率
  ps = [2, 2]
;转换左上角经纬度坐标
  mc = [0.5D,0.5D,-180,90]
  proj = ENVI_PROJ_CREATE(/geographic)
  ;创建 ENVI 格式的 mapinfo
  mapInfo = ENVI_MAP_INFO_CREATE( $
    name = 'Geographic', $
    mc = mc, $
    ps = ps, $
    proj = proj, $
/GEOGRAPHIC)
;设置头文件
  ENVI_SETUP_HEAD, fname=file, $
      ns=ns, nl=nl, nb=nb, $
      data_type=data_type, $
      MAP_INFO = MAPINFO , $
      INTERLEAVE =0, $
  offset=0, /write
```

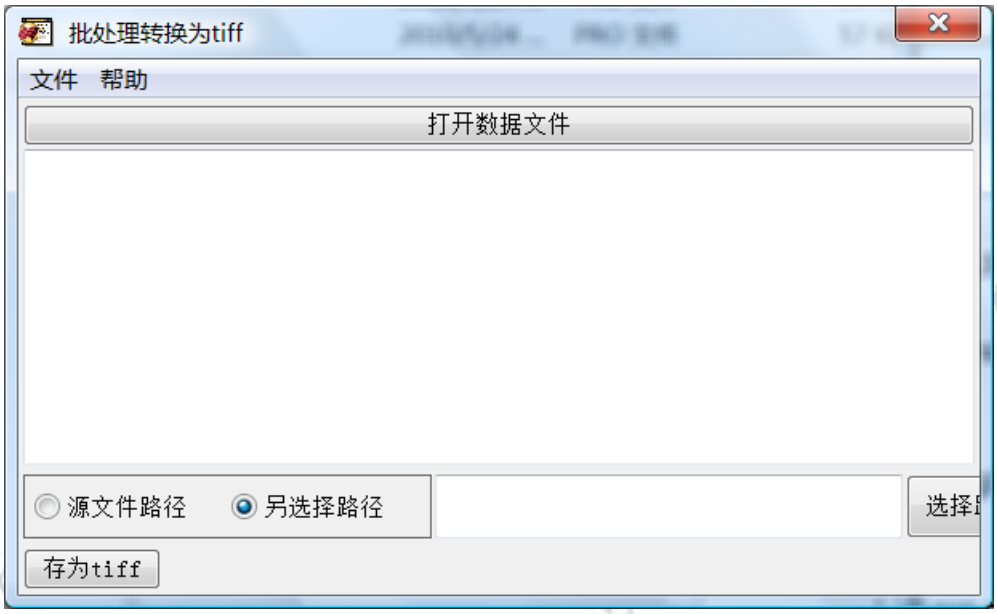
13.5 实例编写

13.5.1 调用 ENVI 实现自动校正

参考 ENVI 帮助中的 envi_auto_tie_points_doit 和 envi_register_doit，调用数据[chapter13\自动配准]；

13.5.2 批处理格式转换函数实现

创建简单界面，自动对文件进行重命名，能够批处理将文件转换为 tif 格式。



bbs.esrichina-bj.cn