SAD/Tkinter 使用指南

生出胜宣

KEK, Oho, Tsukubo, Ibaraki 305, Japan

oide@acsadl.kek.jp

1997 年 10 月 29 日

(适用于 SAD 1.0.5.4.4b)

SAD 是 KEK 从 1986 年起开发至今的加速器设计程序(其概要内容参阅主页 http://www-acc-theory.kek.jp/SAD/sad.html),
现在由 EPICS 通道访问及 Python/Tkinter、Tcl/Tk 解释器等组成。SADScript 解释器语言仅限于加速器模拟设计,今后有可能逐渐被通用系统所使用。SAD/Tkinter 是在 SAD/FFS/SADSkript 脚本语言的基础上,增加了使用 Tk 工具包的函数,而形成的可以编写 Xwindow 应用程序的工具。

这本使用手册的内容、SAD 的程序、以及库文件今后不经预告随时可改编。这本手册的最新版本可从上述主页随时下载。

另外,这本手册不是 SAD/Tkinter 的全部。由于笔者自身的原因,不能体验及把握全部的功能。所以,请读者们同时阅读 Brent Welch:Practical Programming in Tcl and Tk ,1995, for Tcl7.4 and 7.5。此外,本手册中未做说明但 Tcl/Tk 具有的功能亦可利用。

目录

1. SAD 的启动

1.1 SAD 计算机帐户的申请

SAD 是 KEK 开发的一种脚本语言,用于加速器物理计算和控制。从 2002 年底到 2003 年初,BEPCII 控制系统已经把 SAD 移植到 Solaris 工作站，建起了多用户的 SAD 开发平台。物理组使用 SAD 进行 BEPCII 的加速器物理设计；控制组和物理组联合进行基于 SAD 的在线调束软件的分析、移植和开发。

本章介绍的使用方法是基于控制组 Solaris 工作站上提供的 SAD 开发和运行环境。

请在下面网址下载并填写用户申请表格，

http://acc-center.ihep.ac.cn/download/epics_user.doc

经组长或系统负责人签字后交给控制组（联系电话：88236269）即可获得 SAD/EPICS 用户帐户。

目前，SAD 在高能所的主机为 bepc19 和 bepc21，二者都是 Solaris 工作站。

有关 SAD 的帮助信息在 1.3 节给出。

1.2 SAD 应用程序的启动

用户可以从 X 终端登录 bepc19 或 bepc21，也可以在自己的微机上用 Xterm 或 Exceed 仿终端软件登录我们的工作站。

SAD 的运行需要设定一些环境变量，它们是：SAD_PACKAGES、KBFRAMEDIR、SADROOT、LD_LIBRARY_PATH、path。系统已经在/home1/local/sad/sad.cshrc 文件中对它们进行了定义，

所以用户只需在自己主目录下的.cshrc 文件中加入一行：

source /home1/local/sad/sad.cshrc

来引用系统对 SAD 环境变量的定义，即可使用 SAD。启动命令为 runsad。

由于 SAD 是一种脚本语言，用户键入 runsad 命令后，就可以进行交互式的命令调试。

SAD 的运行有两种模式：

- MAIN level 或者称 top-level
- FFS

SAD 启动后进入 MAIN level 模式，没有任何提示符，接收 MAIN level 的命令；其中，键入“FFS;”命令后进入 FFS 模式，有提示符“IN[]”。

SAD 应用程序是由 ASCII 字符组成的纯文本文件，可以用编辑器（如 vi）编写。写好的文件用下述命令启动运行。

Runsad 文件名

1.3 相关链接

SAD 主页：

http://acc-physics.kek.jp/SAD/sad.html

SAD 网络论坛：

http://acc-physics.kek.jp/SAD/BBS/bbs.acgi

SAD 函数使用手册：

http://acc-physics.kek.jp/SAD/SADHelp.HTML

窗口框架 KBFrame 的使用说明：

http://www-kekb.kek.jp:16080/Documentation/KBFrame/manual.html

轨道校正函数使用说明

http://acc-physics.kek.jp/SAD/Correction_manual.html

2. Hello, World!

在此，用 SAD/Tkinter 写出“Hello, World!”作为最简单的应用介绍。(1). X 窗口“Helllo”按钮表示，(2). 该按钮按下后，将在终端输出“Hello, World!”，这是最简单的程序例子。

在 SAD/Tkinter 中，程序如下：

FFS;

```
w = Window[];
b = Button[w,
 Text -> “Hello”,
 Command :> Print[“Hello, World!”]];
TkWait[];
```

下面详细解释：

2.1 FFS

首先，程序第一行的 FFS 指示 SAD 中称为 FFS 的子系统启动，FFS 具体是什么以后再述。

这里请理解为 SAD/Tkinter 全部在 FFS 中执行。

2.2 构件的定义

第二行

```
w = Window[];
```

定义为新的名为 Window 的模块（称为 Widget），X 服务器的画面通常用伴有边框来表示窗口，各种构件可以填充其中。该行程序左边的 w 表示创建一个新的窗口。

顺便说 SADScript 文字的大小写是有区别的，系统内置函数的单词的头一个字母是大写的，其余部分是小写的。例如 ListPlot， LinearSolved 等等。用户可以在系统中定义任何符号，文字。一个符号，数值，公式之中不允许换行。字符行的最后处有\表示换行，换行后可以继续输入。这种情况下换行字符中不能含有数据。若换行文字中有数据则需输入\n（参照 Stringinput）。 除此之外可自由地换行及加入空格。SADScript 的输入格式是相当自由的，因此，程序以易读的方式输入，运行速度不受输入格式的影响。

但是，这行最后有分号连接两个以上的表达式作为一个表达式（参照 10.1.1）。在省略分号的情况下，如果这行最后表达式结束的话，认为程序已结束，在终端输出 Out[n]=（另外，这结果可再利用，参照 10.9.3），因此，如果输出需要的话，一定要加上分号,，有的表达式分号后什么也没有，这时，视为其后有一个特殊符号 Null，Null 没有任何输出。

这行右边的 Window[]中的[]是函数引用表达，其中可置入必要的参数，如下可见：

2.3 构件的作成，属性的指定

程序的第 3，4，5 行

```
b = Button[w,
 Text - > “Hello”,
```

Command : > Print[“Hello, World!”]];

作为 Window w 模块定义 Button b，首先如第三行，参数 w 在 Button 之前已经定义，可直接 Window 的子系统，其它如 Frame 的子系统间接地可在 Window 表示。下面的参数 Text- > “Hello”在 Button 的表面用文字“Hello”表示。记号->的意思后面再述。这样的文字串用双引号括起来，参数之间用逗号分开。这个例子中，构件的属性随之定义

SYMBOL=构件[亲，属性→值，…]; 属性同对什么种类也指定毫无关系。

再有，构件的属性在构件定义之后再定义也可以。

Symbol=[属性]=值;

例如，上面那几行程序也可以写成如下形式：

b = Button[w, Command : > Print[“Hello, World!”]];

b[Text] = “Hello”;

但是，如果程序写成如下形式：

b[Text] = “Hello”;

b = Button[w, Command : > Print[“Hello, World!”]];

结果就完全不同了，这个按钮什么也没有表示，这个出现的时候，b 由于不是 Button，b[Text]=…中 Button 的属性未指定。

2.4 命令（Command）的结合

那么，下面的参数

Command : > Print[“Hello, World!”]

是用于指定当 Button b 被点击后执行的命令的。这虽然再结构上与上述的属性- >值的形式相似，但使用 : >符号代替了- >，尽管 : > 与- >是等价的，但差异在于右边的表达式，现在情况是 Print[“Hello, World!”]并不直接执行，而是过渡到函数。如果改用了 - >，定义 Button 的时候就会执行 Print[“Hello, World!”], 并在终端输出 Hello, World! ，另外, 在 Button 上会有 Print 的返回值 Null。为了避免这种情况，我们便使用 :>，并称此类运算式为延迟运算式，在 SAD 中随处可见。另外，当再定义需要延迟定义的属性时使用 := 来代替=，这样，因为 Command 可把任意的表达式（程序）结合起来，就可以赋予 Button 任意的功能。

2.5 Tk 的实行，TkWait

用此种方式定义的两个构件 Window 与 Button，即使出现在画面上也不会有什么操作。下面的

函数

TkWait[]

被确认后才开始执行。TkWait 函数使得图 1 所示窗口进入等待 event 的状态(例如点击鼠标)。

那么每次点击这个 Button 都会在终端出现

Hello，World!

这章的目的就完成了。

图 1：Hello，World!

函数 TkWait[]是运行 TkReturn 函数等待期间,点击 Button 后反应的必要的动作(参看 4.8.1)。函数 TkReturn 被调用返回参数值，TkWait[]终止运行。现在的 Tkinter 发行这个 TkReturn。如图 2 所示，特别的 Button 在画面左上必须表示出来，这个 Button 如被点击则运行 TkReturn[“ReturnToSAD”]的命令。另外，在这个 Button 所在的窗口四角点击，SAD 立即停止运行，但是这 Button 在 debug 外没有什么作用，将来可能消失。

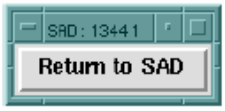

图 2：Return to SAD

3. 捆包 pack

在一个 Window 中有多个构件的情况下，由 SAD/Tkinter 的 packer 来确定各种构件的排列。对各种各样的构件，packer 给予各种指示。

3.1 捆包的方向，Side

捆包对构件最基本的指示是确定构件在 Window 的方向,方向由属性 Side 指定，如下面的例子

```
w = Window[];
a = Button[w, Text -> “1 TOP”,     Side -> “top”    ];
b = Button[w, Text -> “2 LEFT”,    Side -> “left”   ];
c = Button[w, Text -> “3 RIGHT”,   Side -> “right”  ];
```

d = Button[w, Text - > “4 BOTTOM”, Side - > “buttom”];

a, b, c, d 四个 Button 如图 3 在 Window 中分布。再有，数字序号改变 Button 的位置分布，如图 3。图中 Button 的序号表示方向，这样由于 Side 及序号的改变，可以改变 Pack 的结果。

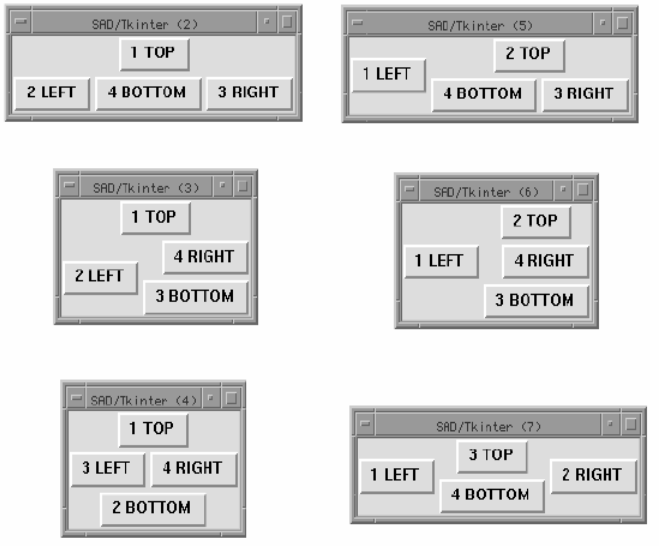

图 3：捆包的结果

3.2 构件与周围的间隔(PadX, PadY)

pack 希望各构件之间保持一定的间隔，属性 PadX 及 PadY 可以指定水平及垂直方向的间隔，这时象素，厘米，英寸都可以作为距离单位来指定间隔（以后再叙述），下面的例子：

```
w = Window[];
a = Button [w, Text -> “1 TOP”,        Side - >“top”,
  PadX - > 20, PadY - > 10];
b = Button[w, Text - > “2 LEFT”,       Side - >“left”,
  PadX - > 20, PadY - > 10];
c = Button[w, Text - > “3 RIGHT”,      Side - >“right”,
  PadX - >20, PadY - > 10];
d = Button[w, Text -> “4 BOTTOM”,    Side - >“bottom”,
  PadX - >20, PadY - > 10];
```

上面的例子可以得到图 4 中的结果。

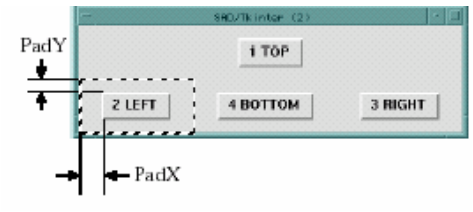

图 4：确保构件之间的距离

3.3 构件的内部间隔 IPadX, IpadY

在有些情况下，构件的内部间隔也需要指定。属性 IpadX，IpadY 可以指定构件内部的水平及垂直方向的间隔。下面的例子：

```
w = Window[];
a = Button [w, Text -> “1 TOP”,        Side - >“top”,
  PadX - > 20, PadY - > 10];
b = Button[w, Text - > “2 LEFT”,       Side - >“left”,
  IPadX - > 20, IPadY - > 10];
c = Button[w, Text - > “3 RIGHT”,      Side - >“right”,
  PadX - >20, PadY - > 10];
d = Button[w, Text -> “4 BOTTOM”,    Side - >“bottom”,
  PadX - >20, PadY - > 10];
```

上面的例子可以得到图 5 中的结果。

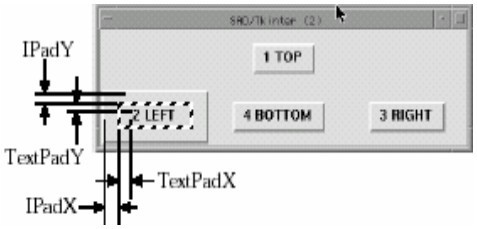

图 5：确保构件内部间隔

另外，Button 等几个构件具有属性 TextPadx，TextPady 。可确保文字之间距离。

3.4 捆包之间的扩展(Expand)

如果 windowyin 因变更大小，构件周围间距不足，构件指定属性 Expand->true 构件周围的间距是构件之间的距离。下面的例子是 Button 用 Expand 指定的效果。图 6 表示结果。

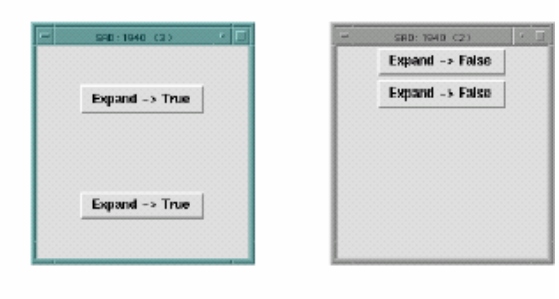

图 6：Expand 的效果

```
w1 = Window[MinsSize - > {200, 200}];
b1 = Button[w1, Expand - > True,
  Text - > “Expand - > True”];
b2 = Button [w1, Expand - > True,
  Text - > “Expand - > True”];
w2 = Window[MinsSize - > {200, 200}];
b3 = Button [w2, Expand - > False,
  Text - > “Expand - > False”];
b4 = Button [w2, Expand - > False,
  Text - > “Expand - > False”];
```

3.5 捆包空间填充 Fill

Fill 是 Expand - > True 等那样一般的 Pack 的空间做出自己的表示域输入的属性。在这里可以对 Fill - > “x”, Fill - > “y”, Fill - > “both”的方向指定。下面的例子 Button 的 Fill 效果表示如图 7 中所示结果。

```
w1 = Window[MinSize - > {250, 200}];
b1 = Button[w1, Expand - > True,
  Fill - > “x”
  Text - > “Expand - > True, Fill - > \ “x\” ”];
b2 = Button[w1, Expand - > True,
  Fill - > “y”
  Text - > “Expand - > True, Fill - > \ “y\” ”];
b3 = Button[w1, Expand - > True,
```

Fill - > “both”

Text - > “Expand - > True, Fill - > \ “both\” ”];

图 7：Fill 的效果

3.6 捆包中构件的几何区域的基准点 Anchor

Anchor 由于 Expand - > true 等的原因被赋予的自身的表示域比自身的 Pack 的空隙还要窄，就得到指定将把表示域设定在其空隙的何处。

Anchor 从“n”，“ne”，“e”，“se”，“s”，“sw”，“w”，“nw”，“c”中取一个值，”c”也可以写成“center”。Anchor 的效果图见图 8。

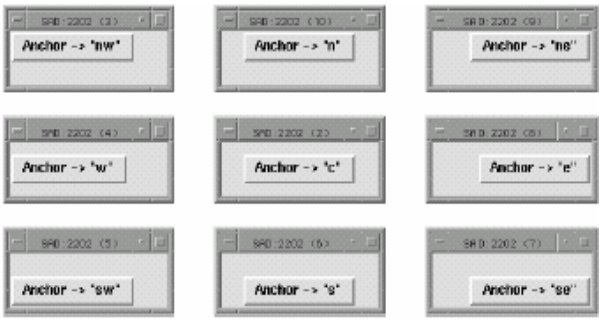

图 8：Anchor 的效果

另外，可以对 Button 等几个构件指定属性 TextAnchor，根据这个可以指定出将表示 text 在构件中的哪一例表示出来。TextAnchor 的参数与 Anchor 是相同的。

4. SAD/Tkinter 中构件的操作

这一章将简要介绍 SAD/Tkinter 中的各种构件的一些通用的操作。常用的构件见下表：

表 1：SAD/Tkinter 的构件

构件	功能	Command	结合变量
Window	窗口		

Frame	框架		
Button	按钮	Command	TextVariable
			Variable
CheckButton	开关按钮	Command	TextVariable
			Variable
RadioButton	多项选择按钮	Command	TextVariable
TextLabel	单行或多行文本标签		TextVariable
TextMessage	带有格式定义的文本		TextVariable
Entry			TextVariable
ListBox	多行字符串的选择框		
ScrollBar	滚动条		
Menu	菜单	PostCommand	
MenuButton			
Scale	滑动标尺	Command	Variable
Canvas	画布		
TextEditor	可以进行各种修饰的文本输入输出框		
OptionMenu	选择菜单		

4.1 结合变量

表 1 中的 Button, CheckButton, RadioButton, TextLabel, TextMessaage, Entry, Scale 等构件，可以以某个标志为结合变量进行编排，例如，将现有的某个 Button d 定义如下：

```
w = Window[];
d = Button[w, Width - > 20, TextVariable : > date];
update : = After[1, date = DateString; update];
update;
```

这里 Width - > 20 指示着把 Button 的宽度确保为 20 个字长。

TextVariable : > date 是指定结合变量。这个 Button 上并没有根据 Text 在 Button 上写的指示，而以编排了 symbol date 坐替换，也就是说，以后若对 symbol date 编排了什么值（字符串），这个值就会在此 Button 的表面表示出来。实际上，在这一例里对于 symbol update

设置了语句After[1, date = DateString; update]，After是SAD/Tkinter上设置的函数，是在经过了相当于第一参数的时间后预约第二参数语句的实行。由于第二参数也包含update，只要update被调用一次，结果就会是update以每秒一次的频率重复运行。Symbol DateString是SAD上设置的函数，此时的时刻用字符串反馈，这样对于symbol date每秒都设定了时间的字符串，Button d的表示也每秒都更新。另外，结合变量如Entry(5,8)将从构件的一侧输入的date用在程序里也是有效的。

4.2 event与构件的结合

可以为构件指定其对鼠标点击、键盘输入等动作事件（event）的反应，例如前述的对于Button的命令的指定。不过一般的，使用函数Bind无论对于什么event都能应对。Bind函数以Bind[构件，event，动作]；的书写形式使用。例如：

```
w = Window[];
d = Button[w, Text - > “Event”];
Bind[d, “<Enter>”, Print[$Event]];
```

如果这样，那么当鼠标光标进入Button d上的时候，终端就会输出$Event的内容：

```
{(Widget : > d), (Tag - > “”), (Type - > “<Enter>”), (X - > 53), (Y - > 28),
 (XRoot - > 769), (YRoot - > 722), (Height - > 0), (Width - > 0),
 (Char - > “??”), (KeySym - > “??”), (SendEvent - > 0), (KeyCode - > 0),
 (State - > 0), (KeySymNum - > 0), (Time - >15202658)}
```

$Event是每当某个event发生时，其信息自动编排的symbol。$Event实际上呈现着8.7节中的用于进行表达式的置换的规则的list的形式（参照第七章）。表2为$Event中出现的符号的意义。Event的表现形式一般为：

“<修饰符－修饰符－型－详细>”

event的型也如表3中所示。

表2：$Event中标识符的意义

符号	值	意义
Widget		event发生的构件
Tag		
Type	event的型名	
X, Y	数值（象素）	构件的位置坐标

RootX，RootY	数值（象素）	光标在窗口内的位置坐标
Height，Width	数值（象素）	构件的高和宽
Char	文字	按下键
KeySym	字符串	表示按下的键
KeySymNum	数值	
KeyCode	数值	按下键的码
Time	数值	按下的时刻，午前零时为零

例如,

“<Key-a>”

“<a>”

“a”

等等，每一个都表示按下 a 键。另外，KeySym 取值如下：

Return，Escape，Up，Down，Left，Right

可以对 event 设置多个修饰符，如对应着键盘的修饰键，或是用于表示双击等鼠标动作的。在各种事件中,与按下键有关的 event 是只有焦点调好的构件才会有反应。构件中有的象 Entry 那样通过鼠标点击就能自动对好焦点，如果不是就必须自己调好焦点。若要给某个构件对焦点，即写成

构件符号[Focus$Set]

另外除掉焦点写成

构件符号[Focus$None]

表 3：event 的型

型	动作
Button,ButtonPress	按下 Button
ButtonRelease	不按 Button
Enter	光标在构件内
Leave	光标移出构件
Motion	光标在构件内移动
Key, KeyPress	按下键

KeyRelease	释放键
Configure	Window 的位置，大小等属性的改变
Destroy	删除 Window
Expose	显示 Window
FocusIn	
FocusOut	

表 4：event 的修饰符

修饰符	动作
Control	控制键（Corl）
Shift	切换键（Shift）
Lock	shift.lock
Meta，M	meta 键
Alt	Alt 键
Button1——Button3	鼠标按键 1—3
Double	双击
Triple	三重点击

4.2.1 event 结合的解除

想要解除 event 的结合，则写成下列形式:

Bind[构件，event]　　或者

Bind[构件，event，]　　或者

Bind[构件，event，Null]

4.3 构件的操作

如前面所述，某些构件最初是以

Symbol ＝ 构件[亲，属性 －＞ 值，…];

的形式得到的。属性 －＞ 值的部分有时也写成属性 ：＞ 值的形式。下面，将这样定义的 Symbol

称作“构件 Symbol”。在这里可以省略亲构件，如果出现省略的情况，将把图 2 中 ReturnToSAD 的表示 Button 的 Window 视为亲的构件。若是重复指定了同样的属性，则先指定的有效。另外，若指定了在其构件上没有意义的属性，则将被忽视。

另外,作为 Symbol 除了单纯的 Symbol,也可以使用象 a[1], a[1, 2, 3]这样带有参数的 Symbol。

再有，如上述所生成的构件，Symbol 将被设置

Widget[Symbol, 构件[亲，属性 - > 值, …]

的值，因此，某个构件 Symbol 的头部（参考 6.2, 6.2.3）经常写作 Widget。

根据属性，其中也有要求复数值的，在这种情况下，将把 list（参考第七章）{值 1，值 2，…} 作为值来使用。某个构件生成后，可以以

构件 Symbol[属性] = 值;　　或者

构件 Symbol[属性] : = 值;

的形式单个的改变其属性（每个构件具体持有什么样的属性将在后面叙述），也有值变成 list 的情况出现。另外，根据属性的不同，也有完全不需要值的。这种情况时写成

构件 Symbol[属性];

此外，有一种属性还可以将其值返回。若出现此种情况就这样使用：

a = 构件 Symbol[属性];

还有一种属性可以一边要求着参数，一边返回值，则写成

a = 构件 Symbol[属性[参数 1，…];

即使是同一属性，也可因书写方式不同而将值的设定和读出两者分别使用。

另外，若想在其后一次设定多个属性，则可用函数 Configure：

Configure[构件 symbol，属性 - > 值，…];

一般的，即使对构件设定了属性，在那一刻表示值也不会改变。为此，必须要调用 Update，TkSense，TkWait（参考 4.8）等等。实行 TkWait 时构件的表示将定期更新。另外 Update[] 将把到此为止定义了的全部构件都更新。

4.4 构件的亲子关系

构件的亲子关系由其生成时的指定来决定。若是子指定亲，亲就必须作为构件而生成。但是，没有必要子就要写在亲的构件内侧。例如，Window 也可以是比自己还小的 Button 的子。

4.5 各构件通用的属性

各个构件都有几个通用的属性，这些将在表 5 中列举出来。其中，捆包关系的属性 Anchor,

Expand, Fill, IPadX, IPadY, PadX, PadY, Side 不适用于 Window。

表 5：各构件的共通属性

属性	值（单位）	功能
Focus$None		除掉构件的焦点
Focus$Set		给其构件对好焦点
Lower		
Raise		
Anchor	“n”, “ne”, “e”, “se”, “s”, “sw”, “w”, “nw”, “center”	
Expand	Ture, False	
Fill	“x”, “y”, “both”, “none”	
IPadX, IPadY		
PadX, PadY		
Side	“top”, “bottom” “left”, “right”	捆包的方向
TextAnchor	“n”, “ne”, “e”, “se”, “s”, “sw”, “w”, “nw”, “center”	
TextPadX, TextPadY		
Background		背景色
BG		Background 的简称
BorderWidth		边界的宽
BD		BorderWidth 的简称
Cursor		鼠标光标的形状
Height		构件的高度
HighlightColor		焦点对上时的背景色
	“flat”, “groove”,	

Relief	“raised”, “ridge” “sunken”	框的立体形状
Width		构件的宽度

表 5 中的单位列就是比如将 1cm 表示成“1c”，将 1inch 表示成“1i”等等，各属性的单位和缺省值因构件的种类而异。今后，若是有在各构件中与标准不同的情况再加以说明。

4.5.1 构件框的立体形状

用属性 Relief 指定的构件框的各种立体形状如图 9 所示：

图 9：构件框的立体形状

4.5.2 字体

若根据属性 Font 来指定文字的字体则这样指定：

Font -> TextFont[“times”, “italic”, “bold”, 12]

但是，哪个 Font 可以使用还取决于 X server。

4.5.3 颜色

用属性 Background 等来指定颜色时写为：

Background -> “red” 或者

Background -> “#FF0000”

在此，什么样的颜色可以使用也取决于 X server。

另外，指定 RGB 如“＃RRGGBB”用 16 进制的字符串表示，此位数取决于 X server。

4.5.4 鼠标光标的形状

当某构件有鼠标光标进入时，光标的形状可以使用属性 Cursor 指定，形式如下：

Cursor -> “形状名称” 或者

Cursor -> “形状名称 文字色” 或者

Cursor -> “形状名称 文字色 背景色”

在这里作为 Cursor 名称，除了图 10 中的以外，只要写成“@文件名”就可以从文件下载。

图 10：鼠标光标的设置

4.5.5 位图

有些构件表面可以贴上位图（Bitmap），以获得生动直观的视觉效果。位图分为两种，一种是 SAD 软件包预定义的，另一种是用户创建并保存在文件中的。其中，预定义的是图 11 中的 8 种图标。位图的使用方法为：用字符串（如“error”、“gray25”）来使用预定义的位图；用“@文件名”的形式来使用自定义的位图文件。另外，位图可用 Unix 的 bitmap 命令简单制作完成。

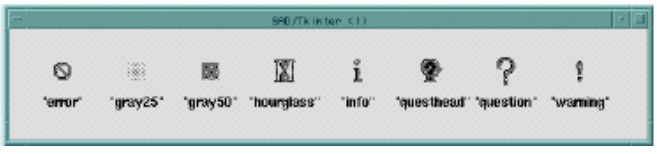

图 11：位图的设置

4.6 构件的几何数据的获取

对于已作成的各个构件，可以使用 WidgetInformation 函数得到各种情报，形式如下：

a = WidgetInformation[构件 Symbol，属性]; 或者

a = WidgetInformation[构件 Symbol，属性，id];

可指定的属性见表 6。

表 6：WidgetInformation 的属性

属性	功能
Geometry	
Height	构件的高度
Width	构件的宽度
X，Y	构件的水平和垂直位置坐标
RootX，RootY	
Screen	
ScreenDepth	
ScreenHeight，ScreenWidth	
ReqHeight，ReqWidth	

a = WidgetInformation[构件 Symbol，{属性 1，属性 2，…}];

若按照以上的操作，结果就会以对应属性 1，属性 2…的列表(list)形式返回，例如：

w = Window[];

WidgetInformation[w, {ScreenWidth, ScreenHeight}]

Out[1] : = {2304, 1720}

输出将为上述的情况。

4.6.1 FromGeometry

表 6 中属性 Geometry 以一种特殊的“宽×高＋x 位置＋y 位置”格式返回象素值。通过使用函数 FromGeometry 可变换成数值 list。反函数是 ToGeometry。

4.6.2 WidgetGeometry

●WidgetGeometry[构件]会返回其构件的{宽，高，x 位置，y 位置}的 list。单位是象素值。

●WidgetGeometry[构件]与 FromGeometry[WidgetInformation[构件，Geometry]]是同一值。

4.6.3 ToGeometry

●ToGeometry[{宽，高，x 位置，y 位置}]会返回 Geomtry 格式的字符串“宽×高＋x 位置＋y 位置”。

●x，y 从左上角向右下角测量。若指定了负数，也可以从右下角测量。

●ToGeometry[{宽，高}]返回“宽×高”。

- ToGeometry[{ ， ， x 位置，y 位置}]返回“x 位置＋y 位置”。

4.7 构件的删除

当 Program 因为终止，切换等原因，已经不再需要制作的构件的时候，一定要通过

DeleteWidget[构件 Symbol1，构件 Symbol2，…]

删除不需要的构件。另外，

构件 Symbol1 = .

和 DeleteWidget[构件 Symbol1]的效果是一样的。

●DeleteWidget　若对亲进行操作，则属于亲的子构件也将全部被删除。

●DeleteWidget[]　将删除所有的构件。

●DeleteWidget[a]　若 a 自身不是构件时，设置成 a[…]形状的构件将被全部删除。

对于结合到某构件的结合变量，删除其最初结合的构件时，该变量也会被自动删除。

也可以利用

DeleteVariable[变量 1，变量 2，…]

的形式，使用 DeleteVariable 来删除变量。

4.8 SAD/Tkinter 的控制

前面也提到了，SAD/Tkinter 的构件若只是定义了还什么都不能表示。通常，函数 TkWait[]，TkSense[]，或是 Update[]被调用时，表示更新，event 才会起反应。

4.8.1 TkWait，TkReturn

●TkWait[]　基于迄今为止的构件的定义，把所有的构件制作，更新，进入等待 event 的状态。一旦发生 event，对应各构件的定义的动作将进行。

●结合的命令中若实行了 TkReturn[式 1]，TkWait 就中断，式 1 的值作为 TkWait[]的结果被返回。

- Button ReturnToSAD（参考第二章）执行 TkRetrun[“RetrunTOSAD”]。

- TkWait[] 在结合了 event 的命令中可以被多次调用。这可以作为对话框在等待使用者的应答时使用。

4.8.2 TkSense

- TkSense[秒] 只在指定的秒数内操作 TkWait[]，在这段时间内若执行 TkReturn[式 2]就立刻返回式 2 的结果，若没有返回就会在指定的时间后返回 Null。
- TkSense[] 与 TkSense[0.3]是同值。

4.8.3 Update

- Update[] 基于迄今为止的构件的定义，会把所有的构件制作并更新。

4.8.4 WaitExpression

- WaitExpression[式 1] 直到式 1 的值变化为止，都会持续 TkWait[]的动作。其间执行了 TkReturn[式 2]就会立刻返回式 2 的结果。

4.8.5 After

- After[秒，式 1] 只在经过了指定的秒数后预约对式 1 的操作。
- After 在 TkWait 和 TkSense 被调用时转入式 1 的执行。

4.8.6 Bell

- Bell[] 将使 X server 的 bell 响一次。

5. SAD/Tkinter 的各种构件

5.1 Window

Window 不同于其他构件，例如不具有捆包属性等这样的特异性，从而有人考虑不把它作为构件。

表 7：Window 的属性

属性	（单位）	功能
Deiconify		
Iconify		
Geometry	“WWWxHHH+dX+dY”	

MaxSize	{H, W}
MinSize	{H, W}
OverrideRedirect	True, False
State	“normal”, “iconic”’ “withdrawn”
Title	字符串
Withdraw	

表 7 中，虽然属性 Geometry 要求“宽×高＋x 位置＋y 位置”这样一个特殊格式的字符串，若使用函数 ToGeometry(参考 4.6.3)可以将{宽，高，x 位置，y 位置}变换成“宽×高＋x 位置＋y 位置”。

Window 还可以进行除此之外的特殊操作，其中有一个是 AdjustWindowGeometry, 是用于将 Window 的尺寸变更到表示其中捆包的构件最低需要的大小。使用形式为：

AdjustWindowGeometry[Window Symbol];

5.2 Frame

Frame 是使 Window 中各个构件整齐排列并收容的框。另外还可以有立体表示方式，框以及背景的着色。例如：

```
w = Window[];
f1 = Frame[w, Relief -> “raised”,
   Side -> “left”, BorderWidth -> 5];
b11 = Button [f1, Text -> “Frames”,
   Side -> “top”, PadX -> 20, PadY -> 10];
b12 = Button [f1, Text -> “are used”,
   Side -> “top”, PadX -> 20, PadY -> 10];
b13 = Button [f1, Text -> “to align”,
   Side -> “top”, PadX -> 20, PadY -> 10];

f2 = Frame[w, Relief -> “ridge”,
```

```
   Side -> “left”, BorderWidth -> 5];
B21 = Button [f2, Text -> “widgets”,
   Side -> “top”, PadX -> 20, PadY -> 10];
B22 = Button [f2, Text -> “in a window.”,
   Side -> “top”, PadX -> 20, PadY -> 10];

f3 = Frame[f2, Relief -> “sunken”,
   Side -> “top”, BorderWidth -> 5,
     PadX -> 10, PadY -> 10];
b31 = Button [f3, Text -> “Frames”,
   Side -> “left”, PadX -> 20, PadY -> 10];
B32 = Button [f3, Text -> “can be nested.”,
   Side -> “left”, PadX -> 20, PadY -> 10];
```

若这样就得到图 12 中的结果，这种情况框的形状由 Relief 和 BorderWidth 决定。若是缺省值就是 BorderWidth -> 0，只有 Relief 无任何效果。Frame 重叠多少都可以。

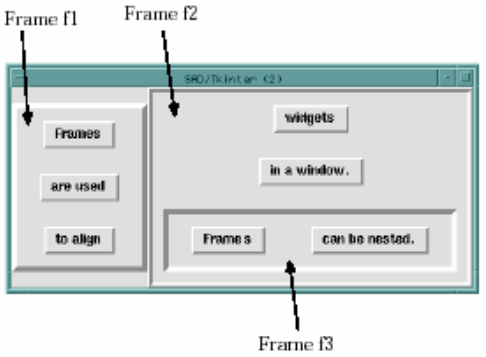

图 12：Frame 的例子

5.3 Button

Button 的属性在表 8 中列举。

表 8：Button 的属性

属性	（单位）	缺省值	功能
ActiveBackground	色		鼠标光标覆上时的背景色
ActiveForeground	色		鼠标光标覆上时的文字色

Bitmap			
Command			
DisableForeground	色		
Foreground	色	“black”	文字色
FG			Foreground 的简称
Font	字型		
Text	字符串		Button 表面的表示字符串
TextVariable	Symbol		
BorderWidth（BD）		2	边界的宽度
Width	文字数		
Relief	“flat”，“groove”，“raised”，“ridge”，“sunken”	“raised”	框的立体形状
TextAnchor	“c”，“n”，“ne”，“e”，“se”，“s”，“sw”，“w”，“nw”，“center”	“c”	
TextPadX	象素	9	
TextPadY	象素	3	
Flash			
Invoke			

5.4 CheckButton

CheckButton 是表示 ON/OFF 的带有核对标志（check mark）的 Button。用 Variable -> Symbol

赋予的 Symbol（结合变量），若有被设置用 OnValue－＞ 值决定的值，核对标志（check mark）灯就会亮。除此之外的情况，如果其值不是 OffValue－＞ 值决定的值，核对标志（check mark）灯也会灭。另外，操作结合变量时，根据此刻 Button 的状态，将返回由 OnValue －＞ 值或 OffValue －＞ 值设定的值。另外，当多个 CheckButton 共有一个变量时，其操作结果是返回最初定义的 CheckButton 的状态，对其变量的值的设定会反映在所有的 CheckButton。例如：

```
w = Window[];
b1 = CheckButton [w, Text －＞ “Linac/BT OK”,
   Variable : ＞ linac];
b2 = CheckButton [w, Text －＞ “LER OK”,
   Variable : ＞ ler];
b3 = CheckButton [w, Text －＞ “HER OK”,
   Variable : ＞ her];
b4 = CheckButton [w, Text －＞ “Belle OK”,
   Variable : ＞ belle];
linac = 1;
ler = 0;
her = 1;
ber = 1;
belle = 1;
```

这样将会出现图 13 中的结果。

图 13：CheckButton 的例子

表 9：CheckButton 的属性

属性	（单位）	缺省值	功能
ActiveBackground	色		鼠标光标覆上时的背景色
ActiveForeground	色		鼠标光标覆上时的文字色
Bitmap			

Command			
DisableForeground	色		
Foreground(FG)	色	“black”	文字色
Font	字型		
Text	字符串		Button 表面的表示字符串
TextVariable	Symbol		
BorderWidth（BD）		2	边界的宽度
Width	文字数		
Relief	“flat”，“groove”，“raised”，“ridge”，“sunken”	“flat”	框的立体形状
Variable			结合变量
OffValue	数值	0	Button OFF 时的返回值
OnValue	数值	1	Button ON 时的返回值
TextAnchor	“c”，“n”，“ne”，“e”，“se”，“s”，“sw”，“w”，“nw”，“center”	“c”	
TextPadX	象素	9	
TextPadY	象素	3	
Flash			
Invoke			
Deselect			
Select			

5.5 RadioButton

RadioButton 是用于进行多者选一型的带标记（mark）的 Button，当以 Variable - > Symbol 赋予的 Symbol(结合变量)被设置了值时，其值与由 Value - > 值决定的值一致的 RadioButton 的标记（mark）灯就会亮。其他的情况 mark 灯都会灭。另外，操作了的结合变量，会根据此刻 Button 的状态返回值。例如：

```
w = Window[];
b1 = RadioButton [w, Text - > “2 ns spacing”,
   Variable : > sb, Value - > 2];
b2 = RadioButton [w, Text - > “4 ns spacing”,
   Variable : > sb, Value - > 4];
b3 = RadioButton [w, Text - > “6 ns spacing”,
   Variable : > sb, Value - > 6];
sb = 2;
```

这样就得到了图 14 中的结果。

图 14：RadioButton 的例子

表 10：RadioButton 的属性

属性	（单位）	缺省值	功能
ActiveBackground	色		鼠标光标覆上时的背景色
ActiveForeground	色		鼠标光标覆上时的文字色
Bitmap			
Command			
DisableForeground	色		
Foreground(FG)	色	“black”	文字色
Font	字型		

Text	字符串		Button 表面的表示字符串
TextVariable	Symbol		
BorderWidth（BD）	象素	2	边界的宽度
Width	文字数		
Relief	“flat”，“groove”，“raised”，“ridge”，“sunken”	“flat”	框的立体形状
Variable			结合变量
Value	数值		Button ON 时的值
TextAnchor	“c”，“n”，“ne”，“e”，“se”，“s”，“sw”，“w”，“nw”，“center”	“c”	
TextPadX	象素	9	
TextPadY	象素	3	
Flash			
Invoke			
Deselect			
Select			

5.6 TextLabel

TextLabel 是用于表示一行或多行的字符串的标签（label）。其内容用 Text - > 字符串 或者 TextVariable - > Symbol 指定。若是后者，其结合变量的内容将被转换成字符串的形式表示出来。另外，字符串中若有 “\n”就在该处换行。

表 11：TextLabel 的属性

属性	（单位）	缺省值	功能
Bitmap			
Foreground(FG)	色	“black”	文字色
Font	字型		
Justify	“left”，“center”，“right”	“center”	
Text	字符串		表示字符串
TextVariable	Symbol		
BorderWidth（BD）	象素	2	边界的宽度
Width	文字数		表示宽度
Relief	“flat”，“groove”，“raised”，“ridge”，“sunken”	“flat”	框的立体形状
TextAnchor	“c”，“n”，“ne”，“e”，“se”，“s”，“sw”，“w”，“nw”，“center”	“c”	
TextPadX	象素	1	
TextPadY	象素	1	

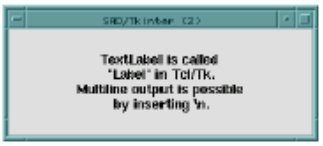

图 15：TextLabel 的例子

5.7 TextMessage

TextMessage 是将长的字符串用规定的宽度一边格式化（Format），一边表示出来。其内容用 Text -> 字符串 或 TextVariable -> Symbol 指定。若利用后者，将把其结合变量的内容转换成字符串表示出来。

表 12：TextMessage 的属性

属性	（单位）	缺省值	功能
Aspect	%	150	宽/高
Foreground(FG)	色	“black”	文字色
Font	字型		
Justify	“left”，“center”，“right”	“left”	
Text	字符串		表示字符串
TextVariable	Symbol		
BorderWidth（BD）	象素	2	边界的宽度
Width	象素	116	表示宽度
Relief	“flat”，“groove”，“raised”，“ridge”，“sunken”	“flat”	框的立体形状

	“c”, “n”, “ne”, “e”, “se”, “s”, “sw”, “w”, “nw”, “center”	
TextAnchor		“c”
TextPadX	象素	1
TextPadY	象素	1

图 16：TextMessage 的例子

5.8 Entry

Entry 是用于将字符串用键盘输入的框。使用 TextVariable，通过把某个 Symbol 指定给结合变量，输入的字符串就可以立刻利用。

表 13：Entry 的属性

属性	（单位）	缺省值	功能
ExportSelection	True， False	True	
Foreground(FG)	色	“black”	文字色
Font	字型		
InsertBackground	色	“black”	
InsertOffTime	msec		
InsertOnTime	msec		
InsertWidth	象素	2	
SelectBackground	色	“green”	
SelectForeground	色	“black”	
SelectBorderWidth	象素	1	
ShowText	文字	“”	

State	“disabled”, “normal”	“normal”	
XscrollCommand			
Justify	“left”, “center”, “right”	“left”	
TextVariable	Symbol		
BorderWidth (BD)	象素	2	边界的宽度
Width	象素	20	表示宽度
Relief	“flat”, “groove”, “raised”, “ridge”, “sunken”	“sunken”	框的立体形状

Entry 具有表 14 中的各种输入编辑功能。

表 14：Entry 的输入编辑功能

	功能
“<Button-1>”	
“<Control-Button-1>”	
“<B1-Motion>”	
“<Shift-B1-Motion>”	
“<Double-Button-1>”	
“<Control-slash>”	
“<Button-2>”	

“<B2-Motion>”

“<Left>” “<Control-b>”

“<Shift-Left>”

“<Control-Left>” “<Meta-b>”

“<Control-Shift-Left>”

“<Right>” “<Control-f>”

“<Shift-Right>”

“<Control-Right>” “<Meta-f>”

“<Control-Shift-Right>”

“<Home>” “<Control-a>”

“<Shift-Home>”

“<End>” “<Control-e>”

“<Shift-End>”

“<Select>” “<Control-Space>”

“<Shift-Select>”

“<Control-backslash>”

“<Delete>”

“<Backspace>” “<Control-h>”

“<Control-d>”

“<Control-w>”

“<Meta-d>”

“<Control-k>”

“<Control-x>”

“<Control-t>”

对于 Entry 可以增加各种操作。下面的例子是 Entry 中最简单的用例，结果如图 17 所示：

```
w = Window[];
f1 = Frame[w];
t1 = TextaLabel[f1, Text - > “Username: ”,
```

```
    Side -> “left”, PadX -> 10, PadY -> 10];
e1 = Entry [f1, TextVariable :> user,
    Side -> “left”, PadX -> 5];

f2 = Frame[w];
t2 = TextaLabel[f2, Text -> “Password: ”,
    Side -> “left”, PadX -> 10, PadY -> 10];
e2 = Entry [f2, TextVariable :> pwd,
    Side -> “left”, PadX -> 5];
    ShowText -> “*”];
```

图 17：Entry 的例子

5.9 Scale

Scale 由槽（trough）和在上滑动的滑片（slider）组成。槽表示着某个变量的可变范围，滑片(slider)表示其现在值。若给 Scale 带上一个结合变量，就可以通过它设定和读出 slider。下面是 Scale 的最简单的例子：

```
w = Window[];
s = Scale[w,
   From -> -10,
   To   -> 10,
   Length -> 200,
   Orient -> “horizontal”,
   Variable :> v,
   Command :> Print[{$Arg, v}]];
```

在这里赋予 From 和 To 各自可变范围的上限和下限。另外，Length 和 Orient 指定 Scale 整体的长度和方向。这是的结合变量是 v，它在 Command 中也被引用。Command 则指定出当 slider 的位置变化时将被执行的表达式。另外$Arg 是当此命令执行时作为 slider 的值递交过去的，

它具有与 v 相同的值。此例得到图 18 中的结果。此例中每当 slider 被移动时终端就会有{-3, -3}这样的表示值显示出来。

图 18：Scale 的例子

表 15：Scale 的属性

属性	（单位）	缺省值	功能
BigIncrement	数值		
Command			
Digits	正整数		
Foreground(FG)	色	“black”	文字色
Font	字型		
From	数值		下限值
To	数值		上限值
Label	字符串		
Length	象素	100	槽的长度
Orient	“horizontal” “vertical”	“vertical”	槽的方向
Resolution	数值		
ShowValue	True，False	True	
SliderLength	象素		
State	“normal”，“active”，“disabled”		
TickInterval	数值	0	
TroughColor	色		槽的颜色

Variable			结合变量
BorderWidth（BD）	象素	2	边界的宽度
Width	象素	15	槽的宽度
Relief	“flat”， “groove”， “raised”， “ridge”， “sunken”	“flat”	框的立体形状

表 16：Scale 的输入功能

event	功能
“<button-1>”	击键在槽上滑动块前进一步
“<control- button-1>”	向滑块端末前进
“<right>” “<up”	滑块增加一步
“<control-right>” “<control-up>”	滑块增加一大步
“<lest>” “<down>”	滑块减少一步
“<control-left>” “control-down>”	滑块减少一大步
“<home>”	滑块移动到最左端或最下端
“<End>”	滑块移动到最右端或最上端

5.10 ScrollBar

ScrollBar 与 Entry、ListBox、TextEditor、Canvas 等等一起使用。ScrollBar 的任务得以完成。ScrollBar 与其他的构件的结合非常简单，如下所示去做。

Physicists=

{“Copernicus”，“Galileo Galilei”，“Kepler”，“hooke”，“Newton”，“Euler”，

“Lagrange”，

“Gauss”，“Faraday”，“Maxwell”，“Boltzmenn”，“Lorentz”，“Einstein”，“Bohr”，

“Heisenberg”,“Schrodinger”,“Pauli”,“Dirac”,“Fermi”};

w=Window[];

sb=Scrollbar[w,Orient->“vertical”,

lb=Listbox[w,YscrollCommand:>sb[set],

Insert->{“end”,physicists},

Side->“right”];

这样，(1) 预先定义 ScrollBar (2) 在欲连接构件中这样写就可以。

ScrollCommand:>ScrollBar_Symbol[set]

在这样情况下，滚动命令 YScrollCommand 、滚动条符号是 sb，相反的定义顺序是不可以的，另外，有关 ListBox 在后面叙述。附带说一句，这个例子导致下面的结果。

ScrollBar 的输入功能及属性总结在表 17.18 中。

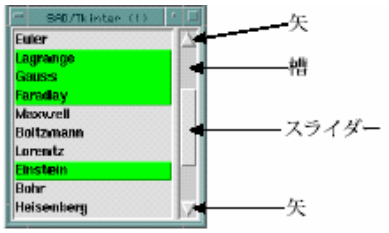

图 19：ScrollBar 的例子

表 17：ScrollBar 的输入功能

event	功能
“<Button-1>” “<Button-2>”	点击箭头滑块前进一步
“<B1-Motion>” “<B2-Motion>”	滑块的牵引
“<Control- Button-1>” “< Control- Button-2>”	滑块向端点移动
“<up>” “down>”	滚动一行
“<Cotrol-up>” “<Control-down>”	上下滚动一帧
“<Left>” “<right>”	左右滚动一单位
“<Control-Left>” “<Control-right>”	左右滚动一帧
“<Home>”	滑块最上端或最左端移动
“<End>”	滑块最下端或最右端移动

表 18: ScrollBar 的属性

属性	输入输出值（单位）	defult	功能
Command	Command	自动设定	滑块变化时执行
Jump	True,False	False	True:Jump.Scroll
Orient	“horizontal”	vertical	槽的方向
	“vertical”		
TroughColor	颜色		槽的颜色
BorderWidth(BD)	象素	2	边界的幅度
Width	象素		槽的宽度

5.11 ListBox

ListBox 是字符串的集合中选出一个或多个要素的工具，基本的功能是 ListBox 做成时字符串的集合给出选择完了后接受选择的要素。选择多个属性 SelectMode 设定是可以的。 另外，对字符串的集合可以随时进行插入、消除等操作。SAD 要素集合的表（参照第七章）已给出。List 全部用{ 和 }括起来，要素之间用，分隔。图 19 的例中：

```
Physicists=
{“Copernicus”,“Galileo Galilei”,“Kepler”,“hooke”,“Newton”,“Euler”,
“Lagrange”,
“Gauss”,“Faraday”,“Maxwell”,“Boltzmenn”,“Lorentz”,“Einstein”,“Bohr”,
“Heisenberg”,“Schrodinger”,“Pauli”,“Dirac”,“Fermi”};
```

分配给符号 physicistsy 右边字符串作为要素。然后，Insert 作为 ListBox 的属性。

```
Ib=ListBox[w,YscrollCommand:>sb[Set],
Insert->{“end”,physicists},
Side->“right”];
```

传送给表 physicistsy，这个例子是如果符号 physicistsy 被设置，当然，

```
Ib=ListBox[w,YscrollCommand:>sb[Set],
Insert->{“end”,physicists},
```

Physicists=

{“Copernicus”, “Galileo Galilei”, “Kepler”, “hooke”, “Newton”,“Euler”,

“Lagrange”,

“Gauss”,“Faraday”,“Maxwell”,“Boltzmenn”,“Lorentz”,“Einstein”,“Bohr”,

“Heisenberg”,“Schrodinger”,“Pauli”,“Dirac”,“Fermi”};

Side->“right”]

直接输入也没关系。

那么，Insert 的最初的参数的“End”的插入点意味最后行。ListBox 中如表 19 中行的指定可以任意使用。

例如，前面插入的情况下指定为零。

表 19：ListBox 的行的指定方法

值	含义
数值 n	最初第 n 行、最初＝1
“Active”	被激活行
“Anchor”	Anchor 指定的行
“End”	最终行
“@x,y”	离坐标（x,y）最近的行

ListBox 有四个选择模式 SelectMode, “browse”、“single”、“extended”、“multiple”。这前两者只能选单项，后两者可选多项。各种各样模式下输入的事件结合有许多差异。我通常是这样，不必要分开使用四种模式。仅仅“browse”、“extended”是不够的。在这里首先说明这两者的输入功能，其他请参照文献。

5.11.1 “browse”模式

“browse”模式是默认的选择模式。这里选择只限于一个。

表 20：“browse”模式的输入功能

Event	功能
“<button-1>”	选择被点击的项目.激活
“<B1-Motion>”	牵引移动选择的项目

“<Shift- button-1>”	激活点击的项目、不选择
“<Up>” “Down>”	激活项目一项上下移动
“<Control-Home>”	选择最初行.激活
“<Control-End>”	选择最后行.激活
“<Left>” “<Right>”	一单位左右滚动
“<Space>” “<Select>” “<Control-Slash>”	选择被激活的行

5.11.2 “Extended”模式

“Extended”模式可选择任意数目的项目。

表 21“Extended”模式的输入功能

Evet	功能
“<Button-1>”	选择点击项目、作为起点
“<B1-Motion>”	选择从起点开始牵引的范围
“<Button-Release-1>”	激活这项目
“<Shift-Button-1>”	选择从起点到点击项目的范围
“<Shift-B1-Motion>”	连续选择从起点开始的牵引范围
“<Control-Button-1>”	作为反转起点点击项目的选择
“< Shift-B1-Motion>”	从起点开始牵引的范围与起点状态一致
“<Up>” “<Down>”	激活项目对于起点一行上下移动
“< Shift-Up>” “<ShiftDown>”	从激活的项目开始扩大选择范围
“<Control-Home>”	选择最初行激活
“< Control-Shift-Home>”	选择至最初行的范围激活
“<Control-End>”	选择最后行激活
“< Control-Shift-End>”	选择至最后行的范围激活
“<Left>” “<Right>”	左右移动一个单位
“<Space>” “<Select>”	选择激活的行
“<Control-Slash>”	选择全体
“< Control-Backslash>”	解除选择

“<Escape>”	消除以前的选择操作

5.11.3 取出选择的项目

选择后如果取出选择的项目，那么返回由选择项目序号组成的 list,即使取出一个项目也返回 list。最初项目的序号为 1。

```
a=list box symbol[selection];
```

如希望取出字符串时，以 list 返回字符串。

```
a=list box symbol[GetText[selection]];
```

另外，如下书写无论有无选择都返回这序号的字符串。

5.11.4 ListBox 的属性

表 22 示出 ListBox 的属性，这里行的指定如在表 19 中那样。

表 22：ListBox 的属性

属性	输入输出值（单位）	缺省值	功能
BorderWidth(BD)	象素	2	边界（表示为立体）的大小
Delete	行 1 或 {行 1，行 2}		消除行 1 或从行 1 到行 2 消除
ExportSelection	True,False	True	传达给 X 选择的范围
Foreground(FG)	颜色	“Black”	文字的颜色
Font	Font		文字的字型
Height	行数		表示的行数
	“flat”		
	“groove”		
Relief	“raised”	“sunken”	框的立体形状
	“ridge”		
	“sunken”		
See	行		在表示范围内移动指定的行
SelectBackground	颜色	“green”	选择范围的背景色
SelectBorderwidth	象素	1	选择范围的边界大小
SelectForeground	颜色	“black”	选择范围内的文字颜色

	“browse”		
	“single”		
SelectMode	“extended”	“browse”	选择 Mode
	“multiple”		
Select$Clear	行 1 或{行 1，行 2}		行 1 或从行 1 到行 2 解除选择
Select$Set	行 1 或{行 1，行 2}		选择行 1 或选择行 1 到行 2
SetGrid	True,False	False	如为 True 时限制变更大小
Width	文字数		表示大小
XscrollCommand	Command		分配横向的滚动条
Xview	文字位置		横向表示的位置
YscrollCommand	Command		分配纵向的滚动条
Yview	行数		纵向表示的位置

5.12 Menu 与 MenuButton

Menu 是在其中贴有几个称为 Menu entry 构件的框。贴的构件有 Button、CheckButton、RadioButton、Separator、和 Cascade 。在此，出现三种 Button，三种单独的 Button 做几乎同样的动作。Separator 是为在构件与构件间进行区分。Cascade 是为使多段式的 Menu 连续运行的构件。

MenuButton 在这里分配 Menu,当按下 MenuButton 时出现下拉 Menu, MenuButton 与 Button 有相同的属性（参照表 8)。下面的例子对 MenuButton mb 分配 Menu mn 之后 mn 追加各种构件。另外，被 mn Cascade 的 mmenu mn1 在追加前应定义好。也就是说必须按 mn、mn1、Cascade 顺序定义否则不能很好地执行。这就是说，当自己是亲或子，而参照的构件在自己被定义前定义。

```
 W=Window[];
Mb=MenuButton[w,Text->“MENU”];
Mn=Menu[mb,TearOff->True];
Mn1=Menu[mb,TearOff->True,
Add->{
RadioButton[Text->“Rare”,
```

```
Value->1,Variable:>Cook],
RadioButton[Text->“Medium”,
Value->2,Variable:>Cook],
RadioButton[Text->“Well-done”,
Value->3,Variable:>Cook]};
Mn[Add]={
Button[Text->“Salad”],
Saparator[],
Cascade[Text->“Steak”,Menu,->mn1],
Button[Text->“Fish”],
Separator[],
CheckButton[Text->“Dessert”,Variable:>dessert]};
Cook=1;
Dessert=1;
```

这例子成为图 20 的结果。

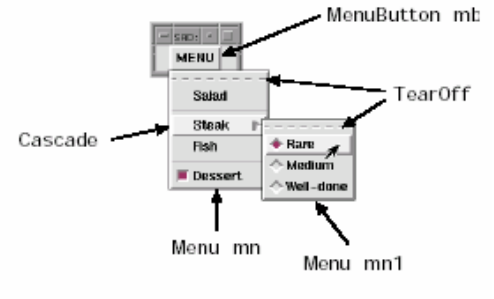

图 20：MenuButton 的例子

5.12.1 Menu 的属性

Menu 的属性示于表 23

表 23: Menu 的属性

属性	输入输出值（单位）	defalt	功能
BorderWidth(BD)	象素	2	边界的大小（三维表示）
Delete	构件序号 1		消除部件序号 1 或者从部件序号

	{构件序号 1，构件序号 2}		1 至部件序号 2 的部件
DisabledForeground	颜色	Disabled	状态的构件文字的颜色
EntryConfigure	{构件序号、属性->值,…}		变更构件序号的构件的属性
Font	Font		文字字型
Foreground（FG）	颜色	“black”	文字颜色
Invoke	构件序号		执行命令
Postcommand	command		Menu 出现时直接执行
SelectColor	颜色	“red”	Check,Radio 选择的颜色
TearOff	True,Fals	False	当 True 时附加 TearOff 构件，把 Menu 从 Button 上剥离，使 Menu 变成可移动。
Yposition	构件序号		具有构件序号的构件从屏幕的位置返回。

表 23 中构件序号是用来识别贴附在 MENU 的各构件，其从 1 开始。另外，“active”（鼠标指定的构件）、“Last”（最后的构件）、“@y”（y 坐标处的构件）也可指定。

5.12.2 Tearoff

表 23 中 Tearoff 的属性为 True(默认为 False)如图 20 的那样各 MENU 的上面附加的虚线。虚线也是一种构件。如果选择这个，则其 menu 会成为从 menubutton 独立出来的一个 window. 这在重复使用一个 menu 时非常方便。Tearoff 为 ture 时，tearoff 为构件 1，而其它构件从序号 2 开始编号。

5.12.3 附在 Menu 的构件的属性

附在 Menu 的构件具有与其对应的单独构件的属性的一部分。下面表示出它们的功能。

表 24　贴在 Menu 构件的属性

属性	输入输出值	默认值	功能
ActiveBackground	颜色		鼠标指处的背景颜色
ActiveForeground	颜色		鼠标指处的文字颜色
Accelerator			

Bitmap			Button 表面上表示 Bitmap
Command	表达式		选中时执行的表达式
Font	Font		文字的字型
ForeGround(FG)	颜色	Black	文字颜色
Justify	“center” “right” “left”	“center”	Text 列的整理
offvalue	数值		Checkbutton 为 off 时的值
onvalue	数值		Checkbutton 为 on 时的值
Selectcolor	颜色	“red”	CheckMark 的颜色
State	“normal” “actave” “disabled”	“normal”	结束的状态
Text	字符串		Button 表面表示字符串
UnderLin	数值		标注下线文字的位置，0 表示最初的文字
Value	数值		RadioButton 在 on 时的值
Variable	Symbol		结合变量

5.12.4 Underline

若对于附在 Menu 的构件指定了属性 Underline->文字位置，其位置（让 0 是最初的文字）的文字将会加上下划线。然后在 Menu 表示中的状态下敲 其文字的键，那个构件的 command 就会执行。还有若敲<space>或<Return>键，现在被选中的构件的命令将被执行。另外<escape>键用于消除 Menu 的表示。

5.13 OptionMenu

OptionMenu 从若干 Menu 中选择一个，Button 表面显示名称的功能被选择则可执行。

如下例：

```
w=window[];
om=OptionMenu[w,
```

Items->{“lion”, “giraffe”, “zebra”, “hippopotamus”, “rhinoceros”}];

Africananimal=“giraffe”;

Items 指定的字符串的 list 中(参照第七章)的一项 TextVariable 指定的参数，分配给 africananimal。而后，此参数的值在 button 的表面表示（如图 21A)。还有，当用鼠标选择时 menu 以现在设定处为中心展开（图 21B)。

Optionmenu 与 menubutton 一样可贴附上 menu。还有,Optionmenu 的属性除了 Item 外与 button 一样。

图 21：OptionMenu 的例子

5.14 其他的构件

SAD 中除了上述的 Cavas、TextEditor 等还有较之更复杂的构件。这里先解说 SAD 语言的构成要素及语法，想在此后其余的构件陆续给予说明。

6. SAD 的构成要素

6.1 元素

SAD 最基本的构成要素（原子 atom）有如下几种。

- 实数　8 个 byte 长度的浮点数
- 字符串　每个字符一个 byte 的任意长度。
- 符号　使用英文、$、数字的任意长度的名字。 不能用数字开头，大小写有区别。
- pattern 为描述与其它要素所对应的要素。

6.1.1 实数输入的表述

与通使常用 E 的 FORTRAN、C 不同。使用 e 表示 10 的幂乘。因为没有复数的表述元素，

使用如 2＋3*I 表述，I 为虚数单位的符号。照现在这样实数与整数没有区别，演算的结果一定要注意实数的误差。

6.1.2 字符串的输入方法

字符串用“”括起来，使用 \ 输入特殊的功能

表 25：\的特殊功能

\e	escape
\n	换行
\r	return
\t	tab
\\	backslash
\nnn	ASC‖码 8 进制数 nnn 表示的字符
\其它的字符	其它的字符
行的末尾\	继续下一行、不包括其他行的字符。

6.2 复合要素、表达式

SAD 中有要素的复合体即复合要素、还有称为表达式的。表达式一般表述为　要 素　0[要 素 1，要素 2，…]　　。要素的数量没有限制。例如，a[b…][c…]…[d…]是表达式的一种。要素 0 是表达式的头部，称为 head。

6.2.1 用运算符构成表达式

使用如上述表达式完全可以构造 SAD 语言。

表 26: 运算符及它们的优先顺序

运算符	完整表达式	组合化
（）		
＃Symbol	Slot[Symbol]	
%正整数	Out[正整数]	
含有_的 Symbol		
式 1:：式 2	MessageName[式 1，式 2]	

式1 ？ 式2	PatternTest[式1，式2]	
{式1，式2，…}	List[式1，式2，…]	
式1[式2，…]	式1[式2，…]	(式1[式2])[式3]
式1[[式2，…]]	Part[式1，式2，…]	(式1[[式2]])[[式3]]
式++	Increment[式]	
式--	Decrement[式]	
++式	Increment[Null, 式]	
--式	Decrement[Null, 式]	
式1@式2	式1[式2]	式1[式2[式3]]
式1/@式2	Map[式1，式2]	式1/@(式2/@式3)
式1//@式2	MapAll[式1，式2]	式1//@(式2//@式3)
式1@@式2	Apply[式1，式2]	式1@@(式2@@式3)
式1//式2//式3	StringJoin[式1，式2]	
式1.式2.式3	Dot[式1，式2，式3]	
式1^式2	Power[式1，式2]	式1^(式2^式3)
-式	Times[-1，式]	
+式	式	
式1/式2	式1*式2^-1	
式1*式2*式3	Times[式1，式2，式3]	
(式1 式2 式3)	Times[式1，式2，式3]	
式1+式2+式3	Plus[式1，式2，式3]	
式1-式2	式1+(-1*式2)	
式1= =式2	Equal[式1，式2]	
式1<>式2	Unequal[式1，式2]	
式1><式2	Unequal[式1，式2]	
式1>式2	Greater [式1，式2]	
式1>=式2	GreaterEqual[式1，式2]	
式1=>式2	GreaterEqual[式1，式2]	
式1<式2	Less[式1，式2]	
式1<=式2	LessEqual[式1，式2]	

式 1=<式 2	LessEqual[式 1，式 2]
式 1= = =式 2	SameQ[式 1，式 2]
式 1< = >式 2	UnsameQ[式 1，式 2]

表 27: 运算符及它们的优先顺序(续)

运算符	完整表达式	组合化
~式	Not[式]	
式 1&&式 2&&式 3	And[式 1，式 2，式 3]	
式 1 ｜｜ 式 2 ｜｜ 式 3	Or[式 1，式 2，式 3]	
式 ..	Repeated[式]	
式 …	RepeatedNull[式]	
式 1 ｜ 式 2 ｜ 式 3	Alternatives[式 1，式 2，式 3]	
Symbol:式	Pattern[Symbol，式]	
式 1 -> 式 2	Rule[式 1，式 2]]	式 1->(式 2->式 3)
式 1 :> 式 2	RuleDelayed[式 1，式 2]	式 1:>(式 2:>式 3)
式 1 /. 式 2	ReplaceAll[式 1，式 2]	
式 / /. 式 2	ReplaceRepeated[式 1，式 2]	
式 1 += 式 2	AddTo[式 1，式 2]	
式 1 -= 式 2	SubtractFrom[式 1，式 2]	
式 1 *= 式 2	TimesBy[式 1，式 2]	
式 1 /= 式 2	DivideBy[式 1，式 2]	
式&	Function[式]	
式 1 = 式 2	Set[式 1，式 2]	式 1=(式 2=式 3)
式 1 := 式 2	SetDelayed[式 1，式 2]	
式 1 ^= 式 2	UpSet[式 1，式 2]	
式 1 ^:= 式 2	UpSetDelayed[式 1，式 2]	
式=.	UnSet[式]	
式 1 ; 式 2 ; 式 3	CompoundExpression[式 1，式 2，式 3]	

式 1 ； 式 2 ；　　CompoundExpression[式 1，式 2，Null]

6.2.2 特殊运算符的用法

特殊运算式的用法

symbol 之间，或是式中间的空白，在前面的式还未结束时，都被看成是 Times(*)以连续的运算式连结的表达式。例如 a<=b<c 等等将转换成 Inequality[a,LessEqual,b,Less,c].这样可以防止中央的表达式被重复操作。这里关系运算表达式就是 Equal(==),Unequal(<>),Less(<),LessEqual(<=),Greater(>),GreaterEqual(>=),SameQ(= = =)以及 UnsameQ(<=>)。

6.2.3 表达式头部内容的取出

某个表达式的头部用一个叫 Head 的函数取出。f[x,y,...]的头部是 f。另外，对于原子 Head 将返回以下值。复素数的头部是 complex。某个表达式具有的意义在大多数情况都由头部决定。表达式在有些时候是函数的引用，若被操作就产生某种作用，返回形式不同的表达式作为结果。另外在其他情况时会被看作是一种 data 的集合体，这时即使被操作也不变换形式。

Head[f[x,y]]

Out[1]:=f

Head[1]

Out[2]:=Real

Head[“abc”]

Out[3]:=String

Head[abc]

Out[4]:=Symbol

Head[abc_]

Out[5]:=Pattern

6.2.4 表达式部分内容的取出

*一个表达式的一部分可用 Part 函数取出。一个表达式 f 的第 n1 个要素用 f[[n1]]取出。取出的表达式中的第 n2 个要素用 f[[n1，n2]]取出。此时 Part[f，n1，n2，.....]和 f[[n1,n2,...]]是等价的。还有 f[[n1]][[n2]]和 f[[n1,n2]]是等价的。

*Indexn1,n2...，0 是头部，1 是最初的要素。若是负的则顺序是从后往前倒数的。函数 Part 还有些特殊的使用方法，将在 list 第 7 项说明。

6.3 标识符（symbol）

symbol 用于在 SAD 中设置某个原子，表达式或是函数。symbol 在没有可设置的东西的时候，它自身将被设定，从而也可以不设置地立刻使用。只要一次性设定了 On[General:: newsym]，以后就可以 检出新的 symbol 的生成。有 n 个 symbol 已被 system 事先定义好了。例如排入函数。构成这样 symbol 的的前头是大字，其他是小字表记。

6.3.1 Set

可使用函数 Set(运算式=)对 symbol 设置一个值。

*symbol1=式 1 把将式 1 操作后的值赋予 symbol 1。

*式（symbol 1 =式 1）自身返回式 1 的值。

*symbol 1 =symbol 2 =...式 1 只对式 1 操作一次，然后把结果赋予 symbol 1，symbol 2。

*{symbol1,symbol2,...}={式 1，式 2...}对 symbol n 设置了式 n 的值，此时必须要 symbol 与式的数量一致。

其他的如后述所说，Set 也可用于 list 的要素的值的设置和函数的定义。

6.3.2 SetDelayed

*symbol 1：=式 1 不操作式 1，将其形式直接赋予 symbol 1。

*除了不操作右边，其他方面 SetDelayed 和 Set 是一样的。

*若对右边进行部分操作要使用后述的 With。

6.3.3 AddTo，SubtractFrom，TimesBy，DivideBy，AppendTo，PrependTo

SAD 有一个函数(运算式)，它可对 symbol 现在值进行运算，再将其结果再赋给同一个 symbol。表 28 中列举，如下。

表 28：Symbol 的再赋值函数

函数	运算表达式	等价表达式
AddTo	Symbol += 式	Symbol = Symbol + 式
SubtractFrom	Symbol -= 式	Symbol = Symbol - 式
TimesBy	Symbol *= 式	Symbol = Symbol * 式
DivideBy	Symbol /= 式	Symbol = Symbol / 式

AppendTo	Symbol = Append[Symbol, 式]
PrependTo	Symbol = Prepend[Symbol, 式]

6.3.4 特殊常数标识符

表 29 的 symbol 群是 system 事先就将常数值设定好的。这些 symbol 具有属性 Constant，输入时立刻会被操作。

表 29: 特殊常数的 Symbol

Symbol	值
True	1
False	0
Pi	2*ArcSin[1]
I	Complex[0,1]
GoldenRatio	(1+Sqrt[2])/5
Degree	Pi/180
SpeedOfLight	299792458
Infinity	(INF)
INF	(INF)
NaN	(NaN)

6.3.5 解除对标识符的值的设置

若对某个 symbol 的已设定的值想解除,就使用 10.7 中的 Clear 和 Unset（=.）。

6.4 表达式的操作

SAD 对表达式进行如下操作:

*其要素中全部由常数和运算式组成的部分式在输入时就被操作。

*其他的表达式在表达式输入结束时被操作。

*函数的参数根据其属性被操作。

7. list

list 作为表达式或表示 data 的集合，在 SAD 中被相当频繁地使用。list 整体用{}围住，要素间用“,”隔开表示。当然 list 的要素是什么都可以，因此 list 里重叠多少个 list 都可以。另外，list 就是头部为 symbolList 的一个表达式而已。即{1，2，3}和 List{1，2，3}是等价的。因此，可对 List 起作用的函数，大多数情况对头部不是 List 的一般表达式也能起作用。

7.1 list 的运算

算术运算是对 list 各要素并排地实行，结果仍为 list。这时如果两个被运算项都是 list，则两项必须长度一致。下一例因为最后一行没有满足这个条件，就无法进行运算。

另外，很多的数学函数对 list 起作用后，对其各要素并排地产生作用，将结果以 list 的形式返回。以上的特性在处理大量的 data 时非常重要，通过编程尽量使 list 整体受到运算和函数的作用，可提升实行速度。

这样的 programming 的方向，用一句话概括就是“尽量简洁”。这种方法，到某种程度而言，就是具有将 program 简略化使之易读的效果。但是越过了某种程度，易读程度和实行效率就成反比。怎样使它平衡很难解答。

list 还被认为是表现矢量，行列，张量的东西

```
{1,2,3}*2
Out[1]:={2,4,6}
{1,2,3}+{4,5,6}
Out[2]:={5,7,9}
{1,2,3}+{{4,5},{6,7},{8,9}}
Out[3]:={{5,6},{8,9},{11,12}}
{1,2,3}+{4,5,6,7}
Out[4]:={{1,2,3}+{4,5,6,7}}
```

7.2 list 的生成

list 直接指定各要素而生成，但也可以由某种规则作出来。

7.2.1 Table，阶数变量

Table 生成具有给予的树的要素的 list。

*Table[式 1，{数 1}]是从将式 1 操作数 1 回得到的 list 返回。

*Table[式 1，{i，数 1}]是将 symbol I 一边从 1 到数 1 每次增加一个，一边操作式 1 的值得到的 list 返回。

*Table[式 1，{i，数 1，数 2}]是将 symbol I 一边从数 1 到数 2 每增加一个，一边操作式 1 得到的值的 list 返回。

*Table[式 1，{i，数 1，数 2，数 3}]是将 symbol I 一边从数 1 到数 2 每次递增数 3 个单位，一边操作式 1 得到值的 list 返回。

这时，symbol I 只在 Table 中有意义。当然，在上面写成数 1 等等的地方，是将某个实际数字赋予它的表达式之意，不一定就是定量。但是这些值在反复开始时被操作，即便在往返途中更改了也不影响往返次数。以上使用的{i，数 1，数 2，数 3}这样指定的方式是称为阶数变量 iterator，在其他的 Do，Sum，Product 中也被使用。还有 Table[式 1 ，阶数变量 1，。。。。。阶数变量 n]与 Table[Table[式 1，阶数变量 n]，阶数变量 1，。。。。。阶数变量 n—1]等价，都是表示 n 维张量的。由后面写的阶数变量决定其变化快慢（呈正比）。

7.2.2 Range

*Range[数 1]返回从 1 到数 1 递增 1 的 list。

*Range[数 1，数 2]返回从数 1 到数 2 递增 1 的 list。

*Range[数 1，数 2，数 3]返回从数 1 到数 2 每递增数 3 个单位的 list。

7.2.3 IdentityMatrix

*IdentityMatrix[n]返回 n 行 n 列的单位行列。

7.2.4 DiagonalMatrix

*DiagonalMatrix[list]返回把 list 1 的各要素作为对角成分的正方行列。

7.3 list 的操作

以下各函数，list 头部不是 List 也可成立，多数情况，有必要使全体的头部和部分的 list 的头部保持一致。例如 Dimensions[a[a[1，2，3]，a[4，5，6]]]是{2，3}。但 Dimensions[a[{1，2，3}，a[4，5，6]]]是{2}。

7.3.1 Length

*Length[list 1]返回 list 的要素个数。

*Length[原子]是 0。

7.3.2 Dimensions

*Dimensions[list 1]将 list 看成张量，把各维的要素数作成 list 返回。Dimensions[{{1，2,3},{4,5,6}}]→{2,3},从而各要素的要素未凑齐的 list 的维就变浅。例如 Dimensions[{{1，2，3}，{4，5}}]→{2}。

*Dimensions[原子]是{}。

7.3.3 Depth

*Depth[list 1]返回 list 1 的最大阶数+1，即由最大多重的部分 list（子 list）组成的数。Depth[{1，2，3}]→2。Depth[{1，{{2，3}，4}，5}]→4。

* Depth[原子]为 1。

*忽视头部一致。Depth[1，a[2，3]] →3。

7.3.4 Level，阶数变量

*Level[list1，阶数 1]返回在 list 1 的 1 阶到阶数之间包含的所有要素。

*Level[{1，2，{{3，4}，5}，2}→{1，2，{3，4}，5，{{3，4}，5}}。

*阶数为负数,则返回 1 阶到各个枝之间的(最上阶+阶数 1+1)阶之间的全部要素的 list[负阶数].下面意思同上.

*Level [{1，2，{{3，4}，5}，-1}→{1，2，3，4，{3，4}，5，{{3，4}，5}}。

*Level[list1{阶数 1}]返回 list 1 的（阶数 1）阶的要素的 list。

*Level[list 1，{阶数 1，阶数 2}]返回 list 1 的从（阶数 1）阶到（阶数 2）阶的要素的 list。

*忽视头部一致。

*阶数 0 指 list 全体。

象以上的 Level 的第 2 参数这样的指定 list 或式的阶数范围的方法称为阶数变量，对 Map，Apply，Position，Count，Cases，DeleteCases，MapIndexed，Scan 都通用。

7.3.5 Take，要素变量 a

*Take[list 1，n1]返回 list 1 的从第 1 个到第 n1 的要素的 list。

*n1 为负值时，Take[list 1，n1] 返回从 list1 的第（全长+n1+1）个要素到最后一个要素的 list。-1 为最后一个要素。

*Take [list 1，{n1，n2}]返回 list1 的从 n1 到 n2 的要素的 list。

*即使将要素变量写成 0，也并不是说它是头部。不管怎样结果的头部都与 list 的头部变的一致。象以上这样的要素以及要素范围的指定（要素变量 a）跟 Drop 等是共通的。

7.3.6 Drop

*Drop[list 1，要素变量 a 1]返回从 list1 中除去了用要素变量 a1 指定的要素的 list。这时头部与 list 1 的头部一致。

*要素变量 a 请参照 Take 7.3.5。

7.3.7 First

*First[list 1]返回 list 1 的最初的要素。

7.3.8 Last

*Last[list 1]返回 list 1 的最后的要素。

7.3.9 Rest

*Rest[list 1]返回除去了 list 1 的最初的要素的剩余部分的 list。这时头部与 list1 的头部一致。

7.3.10 Reverse

*Reverse[list 1]返回将 list 1 中的要素全部逆排列的 list。（顺序倒置）

7.3.11 Append

*Append[list 1，要素 1]返回在 list 1 的最后追加一个要素 1 的 list。

7.3.12 Prepend

*Prepend[list 1，要素 1]返回在 list 1 前头追加一个要素 1 的 list。

7.3.13 Join

*Join[list 1，list 2]返回将 list 1 和 list 2 两方的要素按照此顺序结合的 list。这时两者的头部必须对齐，结果的头部也与之一致。

7.3.14 Flatten

*Flatten[list 1]返回将 list 1 的全部阶数中的全部要素在第 1 阶里平坦地排列的 list。Flatten[{1，2，{{3，4}，5}}]→{1，2，3，4，5}。这时有必要让头部一致。Flatten[{1，2，{a[3，4]，5}}]→{1，2，a[3，4]，5}。

*Flatten[list 1，阶数 1] 使 1 阶到阶数 1 变平坦。

*Flatten[list 1，阶数 1，头部 1]使且仅使 1 阶到阶数 1 中头部为头部 1 的项变平坦。

7.3.15 Thread

*Thread[{list 1，list 2,。。。}]返回以 list 1，list 2,。。。的各自第 n 要素组成的 list。

* Thread[{{1，2，3}，{4，5，6}}]→{{1，4}，{2，5}，{3，6}。

*有必要使头部一致。

*list 1，list 2,。。。中如果包含有与最初的 list 长度不一样的，或是头部不一致的，再或者不是 list 的项，其要素会包含在对结果的每一个 list 通用并包含在其中。Thread[{{1，2，3}，7，{4，5，6}}]→{{1，7，4}，{2，7，5}，{3，7，6}。

*对于长方形的 list，Thread 和 Transpose 都返回同样的结果。

7.3.16 Partition

*Partition[list 1，n]返回把 list 1 的要素从头按每 n 个分配的 list 群。

*list 1 的长度不是 n 的倍数时多余的要素将被遗弃

*Partition[{1，2，3，4，5}，2] →{{1，2}，{3，4}}。

* Partition[list 1，n，m]返回将 list 1 的要素从头以每隔 m 个，允许重复地，每次分配 n 个的形式组合的 list 群。

*Partition[{1，2，3，4，5，6，7}，3，2] →{{1，2，3}，{3，4，5}，{5，6，7}}。

*Partition[list 1，n] 与 Partition[list 1，n，n]是等价的。

7.3.17 Sort

*Sort[list 1]返回将 list 1 的各要素按照标准顺序（参照表 30）重新排列的 list。

*Sort[{4，2，1，3}→{1，2，3，4}。

*Sort[list 1，test 1]按照 2 变量函数 test 1 进行重新排列。Test 1 返回 True 或 False。这样的函数的定义方式请参照 8 函数的定义。

*Sort[{4，2，1，3}，Greater] →{4，3，2，1}。

表 30：标准顺序

表达式类型	类型里的顺序
实数	从小到大
字符串	按最初的不同字符串的顺序排列，首字为“无”的字符串排在前面，首字非字母的按 ASCII 码的顺序

	排列，首字为字母的按 aAbB…zZ 顺序排列。
Symbol 及 Pattern 元素	表示名称的字符串的顺序
List	从元素少的开始排列，若元素个数相同则按头部的序，若头部相同则按第一个元素的顺序，以下同上。

7.3.18 Union

*Union[list 1，list 2,。。。]返回一个将 list 1，list2,。。。的某个里面包含的要素不重复地收集，并遵从标准顺序重新排列的 list。

*Union[{3，-1,“c”}，{“c”，3，3}，{1,“a”}]→{-1，1，3,“a”,“c”}。

*Union[list 1]返回将 list 1 中的要素不重复地按照标准顺序重新排列的 list。

7.3.19 Intersection

*Intersection[list 1，list 2,。。。]返回把在 list 1，list 2,。。。均包含的要素 按照标准顺序重新排列的 list。

*Intersection[{3，1，-2}，{1,“a”}，{1，1}]→{1}。

7.3.20 Complement

*Complement[list 0，list 1，list 2,。。。]收集并返回包含在 list 0 中，且不存在于 list 1，list 2,。。。中的要素，且以标准顺序（参照表 30）重新排列的 list。

*Complement[{3，2，1，3，0}，{2，1}，{4，1}]→{0，3}。

7.4 作用于 list 要素的函数

7.4.1 Part[[]]

由于 list 也是一个表达式，要取出 list 的各个要素，使用可取出式的要素的函数 Part[[]]即可。

- listf 的第 n1 个要素用 f[[n1]]取出。取出的 list 或是式的第 n2 个要素用 f[[n1，n2]]取出。这时 Part[f，n1，n2,。。。]与 f[[n1，n2。。。]]是等价的，此外 f[[n1]][[n2]] 和 f[[n1，n2]]也是等价的。

  index n1，n2。。。里 0 是头部，1 是最初的要素，若是负数的情况就变为从后向前的逆顺序。

- 在 Part 中不必使头部一致。

- f[[{nl1，nl2,。。。}，n2，]]中 indexnl1，nl2,。。。indexn2list{{1，2}，{3，4}，{5，6}，{7，8}[[{2，4}，2]] =>{4，8}。
- 省略 index 使用 null,index 从 1 开始全部元素的值,返回不包括其它的值的 list.
  {{1,2},{3,4},{5,6},{7,8}}[[,2]]=>(2,4,6,8)。

7.4.2 Insert,要素指定子 b

- Insert[list 1,新要素 1,n1] list 的第 n1 个位置插入新要素 1，返回其 list。
- Insert[list 1,新要素 1,{ n1，n2,..}] list 的第{n1,n2,..}位置(list 1{n1,n2,..})插入新要素 1 之后，返回其 list。
- Insert[list 1,新要素 1,{ { n1..}，{n2,..}}] list 的指定位置群插入新要素 1 之后，返回其 list。
- 如上要素位置指定法(要素位置指定 b),在 Take 的情况下有些不同,可使用 Delete,Extract,FlattenAt,MapAt,...。

7.4.3 Delete

- Delete [list 1, 要素指定子 b1] 返回 List 1 的要素指定子 b1 指定的（参照 Insert7.4.2）要素后的 List。

7.4.4 ReplacePart

- ReplacePart [list 1,新要素 1, 要素指定子 b1] 返回 List 1 的要素指定子 b1 指定的（参照 Insert7.4.2）位置群代入新要素 1 后的 List。

即是：

- ReplacePart [list 1,新要素 1, n1] 返回 List 1 的 n1 位置代入新要素 1 后的 List。
- ReplacePart [list 1,新要素 1,{ n1,n2,..}] 返回 List 1 的{n1,n2,..}位置 List 1 {n1,n2,..}代入新要素 1 后的 List。
- ReplacePart [list 1,新要素 1,{{ n1,..},{n2,..}}] 返回 List 1 的几个位置群代入新要素 1 后的 List。

例如 ReplacePart [a ,new ,..]

??????

7.4.5 Extract

- Extract [List 1, 要素指定子 b1] 返回 List 1 的要素指定子 b1 指定的（参照

Insert7.4.2）位置群。

- Extract [List 1, 要素指定子 b1，函数 1] 返回 List 1（表达式亦可） 的要素指定子 b1 指定的（参照 Insert7.4.2）未对函数 1 进行操作的要素群的结果。
- Extract[hold[{Sin[1],Cos[1],}],{1,2},hold]=>Hold[Cos[1]]（参照 Hold 10.6.1)。

7.4.6 FlattenAt

FlattenAt[List1,要素指定子 b1] 返回 List 1 的要素指定子 b1 指定的(参照 Insert7.4.2）要素及直接下属的要素取出排成一列的 List。

- FlattenAt[{1，{2，{3，4}}，5}，{2}]=>{1,2,{3,4},5}。
- 如上例那样，给出指定要素直属下的要素，仅 2 和{3，4}，更深层的如 3，4 不表达出来。
- 头部不必要一致。
- FlattenAt[{1,a[2,{3.4}],b[5]},{{2},{3}}=>{1,2,{3,4},5}。

7.5 List 要素的值的设定

当 Symbol a (或者表达式)被分配时，set 演算式 part，setdelayed 的表达式在左边。

a[[n1,n2,...]]=式 1

这方法注意勿与 Rplacepart 混同。8.函数的定义

SAD 中可以随时定义必要的函数并使用。举个最简单的例子，如果现在如下定义：

f[x_] ：=x^2

就变成这样。这里，上述定义式左边的 x_带有下划线（_）具有相当重要的意义。符号 1 _ 这个记号在 SAD 中被视为[具有符号 1 的名字的 pattern 原子]。如果没有写下划线，就如下变成完全不同的结果。但实际上后者这样的书写方式有时也可以是有效的。

8. 函数的定义

SAD 可随时定义必要的函数，作为一个最简单的例子：

F[x_]:=x^2 这样定义,那么：

F[4] :f 表示参数的平方。

Out[1]:=16

F[1+a] :参数为表达式亦可。

Out[2]:=((1+a)^2

F[3x+x^2]

Out[3]:=(((3x+(x^2))^2)

这里，使用以上定义左边的 x_中（_）表示极为重要的意思。

Symboll_在 SAD 中被视为[具有符号 1 的名字的 pattern 原子]。如果没有写下划线，就如下变成完全不同的结果。但实际上后者这样的书写方式有时也可以是有效的。

f[x]:=x^2　　:左边参数没有_ 。

f[x]

out[1]:=f[4]　　:f 不是平方函数。

f[x]

out[2]:=（x^2）　　:这样，参数 x 给定的话，右边的值则确定。

8.1 参数的置换

(1).可以对应任意形式的参数。

(2).将右边的定义式里出现的同名符号(x)，全部置换成参数的值，然后运算右边。它会如上操作。从而，右边的 x 当右边运算时会是什么由其函数被调用时决定。上面的第二项的右边符号的置换，仅仅置换在右边很明显地出现的符号。也就是说，右边的表达式运算的结果，即使同名符号出现，那也已经与参数无关了。例如：

```
a:=x^3
f[x_]:=x^2+a ;
f[4]
out[1]:=(16+(x^3))
```

象这样，f[4]首先把右边的 x 用参数 4 将 4^2+a 置换，然后运算。如果运算了符号 a 就会碰到同名符号 x，但是这已经与参数的值无关。当然如果如下操作，

```
f[x_]:=(a:=x^3;x^2+a);
f[4]
out[1]:=80
```

这次在右边的 a 的定义式中 x 被明显地写出来了因此参数将置换它。上面如 a：=x^3 这样，a 的定义是用 SetDelayed 书写的，定义 a 时 a 的右边不被运算。但是尽管这样，这时 a 的定义也不是 x^3 而是 4^3 了。也就是说，参数的置换终归是置换，而不是运算。置换会在右边里的所有地方同时开始，并且无法阻止。

x_这种 pattern，对于函数的运算：

8.2 根据参数的不同而对同一符号的多次的函数定义

,通过选择参数的形态，可以对同一个符号给予不同的定义。例如下面这个最简单的例子：

```
f[x_]:=Sin[x]/x;
f[0]=1;
```

我们将函数 f 定义。假如只有第一行，当参数是 0 时就成了 0/0，会出现运行错误。如果象第二行这样仅添写了对于特殊参数值的定义，当参数是 0 时会实行第二行的定义防止错误产生。SAD 运算某个函数时，若其参数与函数的定义参数的 pattern 符合就会实行定义。这时定义越特殊顺序越靠前。从而，上例中虽然参数 0 在哪个 pattern 都能对应，第二行比第一行更为特殊则第二行优先对应并实行。

当然象以上的简单情况就可以把定义合为一个，在右边设定条件判断也能得到同样的结果。但是一般地，由于 pattern 的差异，将定义区分开来能让记述更简洁，实行的效率也会提高，另外参数的个数差异和表达式的形态也会根据 pattern 轻松地区分开。

8.3 函数的定义的解除

如果想解除对某个符号的函数定义，就要使用 10.7 中的 Clear 和 Unset(=.)

8.4 pattern 的变种

以下的[pattern 表达式]指的是有可能包含 pattern 的表达式。不是 pattern 的一般的表达式只和与自身相同的表达式对应。

8.4.1 对于一个或一个以上个数的系列的对应

x_ _这个 pattern 在函数运算时进行以下操作：

(1) 对于一个或一个以上个数的任意参数的系列相对应。

(2) 将右边的定义式中出现的同名符号（x），全部置换成参数系列，并在其后运算右边。

在这里[系列]Sequence 是个特殊的 list，它具有这样的性质：假如它进入了其他的 list 或表达式中，它的要素就会成为不加改动地成为母 list 的要素。；例如：

```
f[x__] := {x};
```

如果照以上方式定义函数 f，f[1]就会返回{1}，f[1，2，3]会返回{1，2，3}，诸如此类。

对应时，是从最初的pattern开始，由系列的长度1的时候开始顺序地尝试。例如下面会这样对应：f[x__,y__] := {{x}, {y}};

```
f[1, 2, 3]
Out[1]:= {{1},{2,3}}
```

8.4.2 对于 0 或是 0 以上个数的系列的对应

x_ _这个 pattern 在函数运算时进行以下操作：

(1) 对于 0 个或 0 个以上个数的任意参数的系列相对应。

(2) 将右边的定义式中出现的同名符号（x），全部置换成参数系列，并在其后运算右边。

但是,对应时是从最初的 pattern 开始从系列的长度 0 的时候开始顺序地尝试.例如下面会这样对应：

```
f[x___,y___] := {{x}, {y}};
f[1, 2, 3]
Out[1]:= {{},{1,2,3}}
```

8.4.3 没有符号的 pattern

- _•_ _•_ _ _这种 pattern 在对应的性质上都和上述的一样，但是右边运算时不会进行任何的置换。这些在仅仅判定参数的形态时是有用的。

8.4.4 指定头部的 pattern

称做符号 1_头部 1 的 pattern 是头部仅对头部 1 的参数对应。

作为头部 1，现在只有符号是可指定的。

比如，x_Real 只对实数对应。

8.4.5 对于表达式的对应

x：pattern 式 1 在函数运算时：

(1). 只对对应于 pattern 式 1 的参数对应。

(2). 把右边的定义式中出现的所有同名的符号（x）全部置换成参数。另外，对于 pattern 式 1 中的 pattern 群也进行同样的置换。随后对右边进行运算。

- 例如：f[p : a|b|c] := Print[p];

是当参数是 a，b，c 中的某一个时才把它 Print 的。

- Pattern 式 1 在对应前不会被运算。例如：

```
Clear[a,f];
f[p : a] := Print[p];      (1)
```

a = 1;　　(2)

f[1]　　(3)

Out[4]:= f[1]

像这样，(1) 时因为 a 没有值，此函数的参数只对符号 a 对应。从而，(2) 中即使后来设定 a 的值，也会像 (3) 中一样并不对应于新值。

8.4.6 PatternTest

- 称做 Pattern 式 1? test 1 的表达式首先要检测参数是否对应于 Pattern 式 1。如果对应了，则会运算表达式：

test 1[对应的值]

如果其结果是 True（非 0 实数），就会判定原参数对应于 Pattern 式 1? test 1。

在这里作为 test 1 ，通常使用下一章将要提到的纯函数。

8.4.7 Alternatives

Pattern 式 1 | Pattern 式 2 | ……是对应于 Pattern 式 1， Pattern 式 2，……中的任一个的 Pattern。

8.4.8 Repeated，..

- Pattern 式 1 ••• 是对应于，其任意要素都与 Pattern 式 1 对应的，一个或一个以上的任意个数的参数的系列的 Pattern。

8.4.9 RepeatedNull，…

- Pattern 式 1 ••• 是对应于，其任意要素都与 Pattern 式 1 对应的，0 个或 0 个以上的任意个数的参数的系列的 Pattern。

8.4.10 不对应时的值的指定

- Pattern 式 1：式 2 是一个当参数不对应于 Pattern 式 1 时，把式 2 的值赋予给 Pattern 式 1，并使之对应的 Pattern。这是一个对 Pattern 赋予省略了参数时的 Default 值的一个办法。

这个记述法与上述的[对表达式的对应]的记述法是同一个形式，因此常常成为混乱的根源。这时仅当[Pattern 式 1]的部分是符号的时候，解释成为[对表达式的对应]。

8.4.11 包含 Pattern 的表达式

一般地，Pattern 式（包含 Pattern 的表达式）中即使包含了带有相同符号的 Pattern 也没有

关系。这时，有一个条件，即同一个符号必须对应于同一个对象。例如，f[1，2]虽然对应于f[x_ ,y_],但是不对应于 f[x_ ,x_]。

8.5 参数的运算

大多数函数中，其参数会在与函数定义对应前被运算。但是，比如也会有不按照函数 Set 的左边，SetDelayed 的两边，Table 的各参数这样运算参数，而是不改变其形式直接递交给函数的情况发生。这样的参数运算的属性可以用 SetAttributes 来指定。

8.5.1 SetAttributes

- SetAttributes[符号 1，HoldAll]用于命令对符号 1 的所有参数不全运算。
- SetAttributes[符号 1，HoldFirst]用于命令不运算符号 1 的第一个参数。
- SetAttributes[符号 1，HoldRest]用于命令不运算符号 1 的第二个以后的参数。
- SetAttributes[符号 1，HoldNone]用于命令对符号 1 的所有参数都运算。
- SetAttributes[符号 1，Constant]将符号 1 的看作是定值，当它被输入时立刻运算。
- SetAttributes[{符号 1，符号 2，•••••}，属性 1]对符号 1，符号 2，••• 赋予属性 1。

8.5.2 NULL

当引用某个函数时 f[……,,……]象这样在逗号之间什么都不写的就认为在此处有符号 Null。

8.6 上方值

迄今为止叙述的对符号的函数定义的设置毕竟是被限制在其符号为函数的头部的范围。对此，设置函数的符号还可以有其他表达式的参数写有的函数定义的方式。比如现在，通过演算子 UpsetDelayed（^:=），

(seed = x_) ^:= SeedRandom[x];

这样，以后称做 seed=n 的表达式会实行 SeedRandom[n]。（SeedRandom 请参照 11.7.3）在这里，UpsetDelayed 的左边的头部是 Set，符号 seed 出现在其第一参数的位置上。这样的对于参数的符号的函数设置叫做上方值 upvalue 的设置，应用范围是相当广的。（对此我们把通常的函数的定义称做下方值 downvalue）

上方值定义的解除通过 Clear10.7.1 进行。

8.6.1 Upset

- 头部的[参数 1，参数 2，……] ^= 右边 是参数 1，参数 2，……为符号的情况或者是头部为符号式的情况时，对于这所有的符号 将把运算了右边的值作为上方值来设置。

- 即使参数是符号[……][……]……[……]这样的形式也会对其符号设置上方值。
- 对于在左边的第二阶以上书写的符号，不会进行设置。
- 当符号被 Protect 保护时不会进行设置。
- 头部 0 无论是什么样的 Pattern 式都可以。

8.7 表达式的置换

SAD 中可以做到把某表达式的一部分置换成别的表达式。这样的置换，已经在函数运算时对于参数的 Pattern 被实行了。置换使用 ReplaceAll（演算子/.），对于什么样的表达式都可以实行。

8.7.1 ReplaceAll，/.

式 0 /.Pattern 式 1->式 1 是起如下作用的：

- 式 0 的值的各部分（除去头部）中如果有能够和 Pattern 式 1 对应的东西，就把它与式 1 的值置换。
- 式 1 中，如果更是有与包含于 Pattern 式 1 的 Pattern 一致的符号，就把它们与式 0 的对应的值置换。

例如：{a, b} /. x _-> x + 1 =>{a + 1, b + 1}。

对于演算子/.的右边赋予规则。规则就是，头部为 Rule（->）或者是 RuleDelayed(:>)的表达式,又或者是由这样的表达式形成的 list。

8.7.2 Rule

Pattern 式 1->式 1 赋予置换的规则.这时两边都会事先被运算。

8.7.3 RuleDelayed

Pattern 式 1：>式 1 赋予置换的规则。这时左边会事先被运算，而右边是进行置换后才运算。规则中目前还没有不运算左边的简单方法，今后会根据需要导入的。

8.7.4 ReplaceRepeated ，//.

表达式 0//.规则 1 只要进行着置换,就会续行通过规则 1 的置换。

8.7.5 With

With 把某个表达式的一部分用别的表达式置换后再运算其表达式。

- With[{式 1，……}，式 0]把式 0 中与式 1 对应的部分与运算了式 1 的值置换后，再运算式 0。式 1 在对应前不会被运算。

- With[{式 1=式 2，……}，式 0]把把式 0 中与式 1 对应的部分与运算了式 2 的值置换后，再运算式 0。式 1 在对应前不会被运算。
- With[{式 1：=式 2，……}，式 0]把把式 0 中与式 1 对应的部分与式 2 置换后，再运算式 0。式 1 在对应前不会被运算。式 2 在置换前不会被运算。

例如,像 SetDelayed 和 RuleDelayed 的右边那样即使是一般不会被运算的表达式,使用了 With 也可以运算其中一部分。另外，With 也能把复杂的表达式变得容易浏览。还有，通过把表达式中的定数部分事先运算，可以提高实行的效率。

With 与后面要提到的 Module 和 Block 构造上相似，但机能完全不同。后者可以生成局部的符号而 With 没有这样的作用。例如，上面的式 1 中出现的符号只是在对应的时候使用到，运算式 0 的时候式 1 本身已经被置换了。

8.8 标识符的作用范围

函数的定义中，大多数情况会需要仅在其函数内部使用的符号。我们把这样的符号称做**局部**符号。局部符号通常用函数 Module 定义。Module 有两个参数，Module[{局部符号 1，……}，式 1]要这样使用。第一个参数是局部符号的 list。这样，式 1 中出现的同名符号全部被看作这个局部符号并被运算。Module 会这样运算式 1 并将结果返回。例如：

```
a := b;          (1)
Module[{b},    (2)
b = c;           (3)
a + b            (4)
]
```

如果这样，(1) 仅仅在 Module 的外面把符号 b 设定给符号 a。在这里因为使用了 SetDelayed，右边的符号 b 还没有被运算。(2) 是来定义只在 Module 中有效的局部符号 b。(3) 和 (4) 相当于 Module 的第二参数。首先，(3)中对局部符号 b 设定了别的符号 c。然后 (4) 中尽管求出符号 a 和局部符号 b 的和并想要把这个和作为 Module 的结果,但其结果是 b+c,而不是 c+c。符号 b 未被运算就设定给了符号 a，符号 b 若是被运算了看上去似乎要变成符号 c，但实际上有一个规则就是：在 Module 的外面书写的符号与在 Module 的内侧被定义的局部符号是完全不同的两个东西。因此得到了以上的结果。

局部符号的值的设定，在 Module 的实行结束时会被全部解除。

8.8.1 Module

- Module[{局部符号 1，……}，式 0]是把仅在 Module 内部才有效的局部符号 1，……定义之后，再运算式 0 并返回其结果。
- Module[{局部符号 1=值 1，……}，式 0]是定义了局部符号 1，并作为其初期值设置了值 1 再运算式 0。
- Module[{{局部符号 1，……}={值 1，……}，……}，式 0]还可以像这样，将对局部符号 1，……的初期值的设定用 list 的形式进行。

8.8.2 Block

函数 Block 用于：不定义局部符号，而仅仅将全局的符号值局部地设定和变更。被声明了的全局的符号的值随着 Block 的结束会返回原来的样子。

- Block[{全局符号 1，……}，式 0]是将局部地使用的全局符号 1，……声明之后，运算式 0 并返回其结果。
- Block[{全局符号 1=值 1，……}，式 0] 是将局部地使用的全局符号 1，……声明，并作为其初期值设置了值 1 再运算式 0。
- Block [{{局部符号 1，……}={值 1，……}，……}，式 0]还可以像这样，将局部地使用的全局符号 1，……的初期值的设定用 list 的形式进行。

8.8.3 局部符号的表示

局部符号自身由 ToString 和 Print 变换成字符串时就会像符号名$nnn 这样在符号名后加上文脉序号。这个文脉序号每当 Module 被调用一次就会递增一个。

8.9 函数库 Library

使用者可以通过将函数的定义写到文件里从而构造一个函数库（Library）。这样的文件只要和 SAD 的输入文件是同样的书写格式就可以了。但是要排列复合表达式时，表达式之间要用分号（参照 CompoundExpression 10.1.1）分隔开才可以。倘若不这样做，就会被认为是在表达式之间有演算子 Times。

Library 的 load 使用 Get（13.1.5）或 AutoLoad（10.7.6）。

9. 构造的演算

9.1 纯函数

迄今为止我们讲过了对于某个符号 f 定义函数 f[……]的方法，SAD 中还有不使用这样的符号

f 也可以定义的函数的形式。我们把它叫做纯函数 pure function。纯函数在构造的演算时较多使用。

9.1.1 纯函数 1，运算符&和 Slot

- （式 1）&表示纯函数。式 1 是包含 Slot（#，#n，##，##n）的表达式。纯函数的参数通过这个 Slot 递交。
- （式 1）&[参数 1，参数 2，……]把纯函数作为参数 1，……**应用**。式 1 的参数通过 Slot 被递交的同时被运算，并返回其结果。
- #表示参数 1。
- #n 表示参数 n。
- ##表示参数全体的系列。
- ##n 表示参数 n 以后的系列。

例如，(Sin[#] / Cos[#2]) &[a,b] →Sin[a]/Cos[b]等等。

纯函数本身即使被运算了也不改变形式。

在纯函数中可以多重地使用包含纯函数的表达式。但是如果是这种情况，不可避免的是 Slot 和参数的对应会**不太清晰**。尽可能的话还是希望使用者能通过 With 给纯函数加上名称。另外，根据情况也会有用 Slot 无法得到参数的对应的时候。

9.1.2 纯函数 2 Function

通过纯函数的多重使用，在 Slot 无法对应时可以使用 Funtion 进行 Slot 不参与的参数的对应的纯函数的定义。

- Function[{符号 1，……}，式 1]是纯函数，它的参数对应于符号 1，……。式 1 是包含符号 1，……的表达式，是纯函数的躯干。
- Function[{x，y}，Sin[x]/Cos[y]][a，b] →Sin[a]/Cos[b]。
- 参数的符号中，只有在纯函数的躯干上显露地书写的才会置换成对应的值。

9.2 构造的演算

SAD 中对于每个原子和 list 的要素当然是可以进行各种演算的，但还可以进一步地对某个 list 和表达式的整体一次演算。例如，我们已经提到了算术演算可以对 list 的要素并列地起作用。实际上对于一般的函数这样的演算也是可以的。在这里，我们把对 list 和表达式的整体起作用的演算称做[构造的演算]。构造的演算中有代表性的是 Map（演算子/@）和 Apply（演算子@@）。这些在 SAD 中经常要使用，**因此可以说是特殊的演算子**。

现在假设有一个 2 阶的 list 1 ，要做一个以其各要素的长度为要素的 list。这个问题如果使用了 Map（/@），只须写成：

Length /@ l

就可以了。一般地对于某个函数 f，f/ @{a_1，a_2，……}实行{f[a_1]，f[a_2]，……}。

然后，对于把数值作为要素的某个 list 1，要求其合计，平均值，**2 乘平均值**。这些要使用 Apply（@@）分别写成：

```
Plus @@ l
Plus @@ l / Length[l]
Plus @@ (l^2) / Length[l]
```

一般地对于某个函数 f，f@@{ a_1，a_2，……}实行 f[a_1，a_2，……]。

像这样，通过构造的演算，可以把复杂的演算非常简洁地写出来。构造的演算不仅是简洁，还可以提高速度。但是如果把构造的演算在一个表达式中多次重叠用，会降低 Program 的可读性，使用时要多加注意。

9.3 各种构造的演算

在以下的情况中演算起作用的 list 头部不一定必须是 List。

在这些当中，对于 Map,MapAll,MapIndexed,Apply,Scan,Cases,DeleteCases，可以通过阶数指定子（Level 7.3.4 参照）指定演算的范围.

9.3.1 Map，/@

- f/@ list 1 是使函数 f 对 list 1 的各要素起作用。
- Map[f，list 1，阶数指定子]是 list 1 的阶数指定子（Level 7.3.4 参照），使函数 f 对被指定阶的各要素起作用。
- f/@ list 1 与 Map[f，list 1]和 Map[f，list 1，{1}]是等价的。

9.3.2 MapAll，//@

- f//@ list 1 是使函数 f 对 list 1 的所有阶（包括 0 阶）的全部要素起作用。
- f//@ list 1 与 Map[f，list 1，{0，Infinity}]等价。
- 结果的头部会成为原式的头部。
- MapAll[f，list 1，Heads ->True]是使函数 f 对 list 1 的所有阶的全部要素包括头部起作用。

9.3.3 MapIndexed

- MapIndexed[f，list 1]是使函数 f 对各要素起作用，此时把在此要素的 list 1 里的位置作为第二参数递交给 f。
- MapIndexed[f，list1，阶数指定子]是 list 1 的阶数指定子（请参照 Level7.3.4），并使函数 f 对指定的各要素起作用，把其要素的 list 1 中的位置作为第二参数递交给函数 f。

9.3.4 Apply,@@

- f@@list 1 是把由从 list 1 的全要素形成的系列作为参数来运算函数 f 的。
- Apply[f,list 1，阶数指定子 1]是 list 1 的阶数指定子（请参照 Level7.3.4），把由指定的阶的全要素形成的系列作为参数来运算函数 f。
  - f@@list 1 和 Apply[f，list 1]是与 Apply[f,list 1,{0}]等价的。
  - 结果的头部如果 Apply 没对其阶起作用，原来的表达式的头部会被保留。

例如：Apply[f,g[h[1,2],h[3,4]],{1}]→g[f[1,2],f[3,4]]。

9.3.5 Scan

Scan[函数 1，list 1]将函数 1 应用于 list 1 的一阶的各要素。最后将返回 Null。

Scan[函数 1，list 1，阶数指定子 1]是将函数 1 应用于 list 1 的阶数指定子 1（请参照 Level7.3.4）中指定的各要素。最后返回 Null。

Scan[函数 1，list 1]虽然与 Do[函数 1[list 1[[i]]],{i,Length[list 1]}]的结果相同，但一般来讲速度更快效率更高。（尽量避免添字。）

9.3.6 Position

position[式 1，pattern 式 1] 返回给 list，式 1 全部分式中与 pattern 式 1 对应的元素位置参数。

position[式 1，pattern 式 1，阶数变量 1] 返回给 list，式 1 全部分式中与阶数变量 1（参照 level 7.3.4）指定阶数的 pattern 式 1 对应的元素位置参数。

position[式 1，pattern 式 1，阶数变量 1，n1] 返回给 list，式 1 全部分式中与阶数变量 1 指定阶数的 pattern 式 1 对应的元素的最初 n 个位置参数。

9.3.7 Count

Count［式 1，pattern 式 1］ 返回式 1 全部分式中与 pattern 式 1 对应的部分式个数。

Count［式 1，pattern 式 1，阶数变量 1］返回阶数变量 1（参照 level 7.3.4）指定阶数的且与 pattern 式 1 对应的部分式的个数。

Count［式 1，pattern 式 1，阶数变量 1，n1］返回阶数变量 1 指定阶数的且与 pattern 式 1 对应的部分式个数。

9.3.8 Cases

cases［list1，pattern 式 1］ 返回给 list1，list1 的要素中与 pattern 式 1 对应的要素群。

cases［list1，pattern 式 1，阶数变量 1］返回阶数变量 1（参照 level 7.3.4）指定阶数的且与 pattern 式 1 对应的部分式。

Cases［list1，pattern 式 1，阶数变量 1，n1］返回阶数变量 1 指定阶数的且与 pattern 式 1 对应的部分式群中最初的 n1 个。

如果把上述 pattern 式 1 写成 pattern 式 1→变更值 1 的形式，根据这个规则结果将进行置换。

9.3.9 Deletecases

DeleteCases[List1,pattern 式 1] 除去 List1 中与 pattern1 的元素对应的元素并返回。

DeleteCases[List1,pattern 式 1，阶数变量 1] 阶数变量 1（参照 Level 7.3.4）指定的阶数的除去 List1 中与 pattern1 的元素对应的元素并返回。

9.3.10 MapAt

MapAt［函数 1,List1,n1］使用函数 1 于 List1 的第 n1 个位置，将置换的结果返回 List。

MapAt［函数 1，List1,{n1，n2,…}］使用函数 1 于 List1 的第{n1,n2…}位置，将置换的 List 的结果返回。

MapAt［函数 1，List1,{{n1，…}{n2,…}}］使用函数 1 于 List1 的若干个位置群，将置换的 List 的结果返回

9.3.11 MapThread

MapThread［f,{{a1,a2,…},{b1,b2,…},…}］返回{f[a1,a2,…],f[b1,b2,…],…}。

MapThread［函数 1，{式 1，式 2，…}，阶数变量 1］函数 1 用于阶数变量 1 指定的式 1，式 2，…的部分表达式群。

9.3.12 Nest

Nest[函数 1，式 1，n］函数 1 用于式 1

Nest[f,x,3] => f[f[f[3]]]

9.3.13 Select

Select[List1,函数 1] 函数 1 用于 List1 的各元素，一些结果成为 True(非 0)的元素构成的 List 返回。

Select[List1,函数 1，n1] 函数 1 用于 List1 的各元素，一些结果为 True(非 0)的元素中最初的 n1 各构成的 List 返回。

9.3.14 SwitchCases

SwitchCases [List1,pattern 式 1，pattern 式 2，…]] List 中的各要素按与 pattren 式 1,pattern 式 2,…，分类并返回这样形成的 List。

如果不能归于任一类，而对于结果是必要的情况下，把它加到 pattren 式的 List 的最后。

9.3.15 SelectCases

SelectCases [List 1,{函数 1，函数 2，…}]函数 1，函数 2，…用于 List 的各要素，返回结果为 True(非 0)的要素分类的 List 。

SelectCases[{1，2，3，4}，{（#<2）&,（#<3）&}]=> {{1},{3,4}}。

如果不能归于任一类，而对于结果是必要的情况下，把 True&它加到 pattren 式的 List 的最后。

10. 编程

除了定义函数和执行函数，SAD 还提供条件判断、循环控制等手段实现对程序流程的复杂控制。

10.1 表达式的连接

10.1.1 CompoundExpression

式 1；式 2；…；式 n 从式 1 开始到式 n 顺序操作，返回最后表达式的结果。

式 1；式 2；…；式 n； 从式 1 开始到式 n 顺序操作，返回 Null。

TracePrint 10.9.1 中各种表达式 操作前按原来样子写出来。

10.1.2 Goto 和 Label

SAD 虽然不怎麼使用但 Goto 仍然存在

Goto[Label1]是将控制向同一或比它低阶 CompoundExpression 的 Label[Label1 的值]转

移，这里 Label 最好是表达式。Label 自己甚麼也不做，返回自己。

10.2 条件表达式

SAD 的条件变量，逻辑变量是实数 0 时为伪，非 0 的实数时为真。虽然除了 0 以外的任意实数都为真，但实际上演算中为真的时候返回实数 1。逻辑表达式的结果象这样的实数可以混在其他的算术式中运算。另外，系统中准备 true,false 两个符号，这两个符号实际上与 1 和 0 是等价的。另外，要求条件判断时使用普通的算术式书写也没关系。

条件演算式中（ SameQ = = =）和 UnsameQ（〈=〉）以外的即真，伪以外的值， 也就是说，有时会置换条件式的值。例如，条件式 a= =1，如果 a 任何值都没被分配时，这条件式的结果既不是真也不是伪，实际上这表达式 a= =1 既为本身。另外，当 a 被分配给实数以外的值时，(a 的值）= = 1 这样的表达式被返回。(成为这样的理由是 a= =1 这样的表达式在许多函数中把它作为方程式来看待）对此，演算式 SameQ = = = 和 UnsameQ（〈=〉）一定返回真或伪的结果。

10.2.1 SameQ ，= = =

式 1 = = = 式 2 两边作为 SAD 的要素为同一值（式）时，返回 true（1）否则返回 false（0）。

10.2.2 UnsameQ ，〈=〉

式 1 = = = 式 2 两边作为 SAD 的要素不为同一值（式）时，返回 true（1）否则返回 false（0）。

10.2.3 MatchQ

Match Q [式 1，pattern 式 1] 当式 1 与 pattern 式 1 符合时返回 true（1），不符合时返回 false（0）。

10.2.4 MemberQ

- MemberQ [式 1，pattern 式 1] 当式 1 全部中有部分式与 pattern 式 1 符合则返回 true（1），不符合时返回 false（0）。
- MemberQ [式 1，pattern 式 1，阶数变量 1] 当式 1 的指定的阶数变量 1（参照 Level7.3.4）有部分式与 pattern 式 1 符合则返回 true（1），不符合时返回 false（0）。

10.2.5 FreeQ

- FreeQ [式 1，pattern 式 1] 当式 1 全部中没有一个部分式与 pattern 式 1 符

合则返回 true（1），当式 1 全部中即使有一个部分式与 pattern 式 1 符合则返回 false（0）。

- FreeQ［式 1，pattern 式 1，阶数变量 1］ 当式 1 的阶数变量 1 指定的（参照 Level7.3.4）阶数的全部中没有一个部分式与 pattern 式 1 符合则返回 true（1），当式 1 全部中即使有一个部分式与 pattern 式 1 符合则返回 false（0）。

10.2.6 VectorQ

- Vector Q［式 1］当式 1 的值是一阶的 list 时为 true（1），其他的情况为 false（0）。
- Vector Q［式 1，test1］当式 1 的值当所有的要素被 test1 作用时是真（非 0）时为 true（1），其他的情况为 false（0）。

10.2.7 Matrix Q

- Mtrix Q［式 1]的值为行列式时是 True（1），其他的情况为 false（0）。
- Mtrix Q［式 1，test1］的值当行列式的各要素在函数 test1 的作用下为真（非 0）的情况下是 True（1），其他的情况为 false（0）。

10.2.8 Complex Q

- Complex Q［式 1］当式 1 的值是非实数的复数，或很少至少有一个复数的 list 的情况下为 True（1），其他的情况为 false（0）。

10.3 条件判断

10.3.1 If

- If［式 1，真 1］对式 1 进行操作，如果结果为 True 则返回真 1，如不是那样的情况则返回 Null。
- If［式 1，真 1，伪 1］对式 1 进行操作，如果结果为 True 则返回真 1，如为伪则返回伪，如两种情况都不是则返回 Null。
- If［式 1，真 1，伪 1，它 1］对式 1 进行操作，如果结果为 True 则返回真 1，如为伪则返回伪，如两种情况都不是则返回它 1。

10.3.2 Or，‖

- 式 1 ‖ 式 2 ‖... 是各式的逻辑和，将一直操作下去直至结果为真，剩下的不再操作，并返回（ 1 ）。从而可用来作为条件判断。

10.3.3 And，&&

- 式 1 && 式 2 && …是逻辑积，对式 1，式 2…的操作持续至结果为伪。此时，剩下的不再操作，返回 False(0),从而可用来做条件判断。

10.3.4 Not, ~

- ~式 1 是逻辑非的操作，对式 1 操作，如果结果为 0 以外的实数则返回 False(0),结果为 0 返回 True(1)，如果结果不是上述两种情况就返回 ~式 1 其表达式本身。

10.3.5 Switch

- Switch［式 0，pattern1，式 1，pattern2，式 2，•••］首先操作式 0，如果其结果与 pattern1 对应就会返回将表达式 1 操作后的值。如果不是，就回 2 一直操作到与 pattern2，••• 对应为止。如果与任何一个 pattern 都不对应就会返回 Switch 的表达式本身。
- 如果与哪一个都不对应时想操作式 a，就在参数的最后写上_,式 a 就可以执行了。

10.3.6 Which

- Which［条件 1，式 1，条件 2，式 2，…］ 对条件 1 操作如果为真则返回对式 1 操作的结果。如果不是这样则继续对式 2 操作…如此进行，如果哪一个条件式都不是真则返回 Which 的表达式本身。
- 如果哪一个条件式都不是真，还想对式 a 进行操作，参数的最后写上 True,式 1 可以执行了。

10.4 循环

10.4.1 Do

- Do ［式 1，循环变量 ］。按循环变量（参照 7.2.1）指定的次数对式 1 进行操作，最后返回 Null。
- 运行中如果 break[ ]被调用则中断循环。
- 运行中 如果 continue[ ] 被调用则中断对式 1 的操作，继续循环。

10.4.2 While

- While［条件式 1，式 2 ］。每次对条件式 1 操作，结果为真（非 0）时则继续对式 2 进行操作，最后返回 Null。
- 运行中 break[ ]被调用则中断循环。

10.4.3 For

For［初始化 1，条件 1 ，增量 1，式 1 ］。首先执行初始化 1，然后循环：当条件 1 为真的情况下，先执行式 1 再执行增量 1 的操作。例：

```
In[2]:= For[i=1,i<5,i++,Print[i]]
1
2
3
4
```

运行中 break[ ]被调用则中断循环。

10.4.4 Scan

- Scan[函数 1，list 1]。list 1 的一阶的各要素用于函数 1 ，最后返回 Null。
- Scan[函数 1，list 1 ，阶数变量 1 ］阶数变量 1（参照 7.3.4）指定的 list 1 的各要素用于函数 1 ，最后返回 Null。
- Scan[函数 1，list 1 ］与 Do［ 函数 1，［ list 1[［ i］］］,{I ,Length[list 1］｝］有相同的结果，一般地执行速度更快。(尽可能地不增加字数)。

10.4.5 Sum

- Sum[式 1，循环变量 ］按循环变量 （参照 Table 7.2.1 ）指定的次数对式 1 进行操作，返回结果的和。

10.4.6 Product

Product［ 式 1，循环变量 ］按循环变量（参照 Table 7.2.1 ）指定的指定的次数对式 1 进行操作，返回结果的积。

10.5 Program 的中断和异常处理

10.5.1 Break

- Break［ ］ 将 Do , While , For 把控制转移到它们的下一个表达式。

10.5.2 Continue

- Continue［ ］Do 中断正在执行中的那次循环，继续循环。

10.5.3 Return

- Return ［ 式 1 ］中断函数的操作，函数的值作为式 1 的值返回。

10.5.4 Eixt

- Eixt [ ] 中止 SAD。

10.5.5 Throw

Throw [式 1] 在 Catch 中使用，中断 Program 并将式 1 的值作为 Catch 的值返回。

10.5.6 Catch

- Catch [式 1] 对式 1 进行操作，并返回其值。运行中调用 Catch [式 2] Program 在此中断，返回式 2 的值。

10.6 对表达式操作的控制

10.6.1 Hold

- Hold [式 1] 对式 1 不操作，将 Hold [式 1]这样的表达式作为结果返回。
- Hold [Sin [1]]=> Hold [Sin [1]]
- 如果对式 1 的一部分操作，使用 With 8.7.5。
- 如果想要对这表达式一部分不操作而只将其取出，则需把 Extract 7.4.5 和 Hold 组合使用。
- Extract [Hold [{ Sin [1], Cos [1] }], { 1,2 }, Hold]=>Hold [Cos [1]]。

10.6.2 ReleaseHold

- ReleaseHold [Hold [式 1] 对式 1 进行操作返回其结果。
- ReleaseHold [Hold [Sqrt [4]]]=> 2 。
- 如上述的表达式那样但在参数头部没有 Hold 的情况下，ReleaseHold [式 1]对式 1 进行操作并返回其结果。
- ReleaseHold [{ Hold [Sqrt [4]] }]=》{ Hold [Sqrt [4]] }

10.6.3 Evaluate

- f […,Evaluate [式 1，…] 无论 f 的参数的操作有无指定都将操作式 1 并传送其值给函数 f 。
- Hold [Evaluate [sqrt [4]]]=> Hold [2] 。
- 如上述的表达式那样但在参数头部外部出现 Evaluate 的情况下，如果对 ReleaseHold [式 1]进行操作则对式 1 进行操作并返回其结果。

- Evaluate [sqrt [4]]=>2 。
- Hold [{Evaluate [sqrt [4]]}]=> Hold [{Evaluate [sqrt [4]]}] 。
- 对式 1 的一部分进行操作时使用 With 8.7.5 。

10.6.4 Unevaluate

- f […,Unevaluate [式 1], …] ,无论 f 的参数的操作有无指定都将不对式 1 操作并原样传送给函数 f 。

10.7 标识符的设定，诊断

10.7.1 Clear

- Clear [Symbol 1,Symbol 2,…] 解除 Symbol 1,Symbol 2,…分配的全部的值及其函数的定义。
- Clear [Symbol 1 [参数 1],…] Symbol 1 被分配的上方值（8.6 处论述），Symbol 1 [参数 1],…]的形式 被解除 。

10.7.2 Unset，=.

- Symbol 1=. Symbol 1 分配的全部的值及函数关系被解除，
- Symbol 1 [参数 1，…] =. 将 Symbol 1 [参数 1，…]分配的函数定义解除。
- 上方值（参照 8.6）Unset 不能解除。
- 对于控件 Symbol 1 来说，Symbol 1=.与 DleateWidget[控件 Symbol 1]是做同样的操作。

10.7.3 Names

- Names [字符串 pattren 1] (参照 12.5 字符串的符合) 返回已经定义的 Symbol 中名字与字符串 pattren 1 符合的名字群的 List。
- Names [“Arc*”]=>

  {“ArcTanh”, “ArcCosh” “ArcSinh”, “ArcTanh” “ArcCos”, “ArcSin”}。

10.7.4 Protect

- Protect [Symbol 1, Symbol 2,…] 禁止以后分配 Symbol 1, Symbol 2,…的值及函数关系。
- 局部的 Symbol（8.8）不能进行 Protect。
- System 准备好的函数 Symbol 已经被 Protect。

10.7.5 Unprotect

Unprotect［Symbol 1, Symbol 2,…］以后许可分配 Symbol 1, Symbol 2,…的值及函数关系。

10.7.6 AutoLoad

根据需要要想把某个函数库 load，可以使用 Autoload。

- Autoload［Symbol 1,Symbol 2,…,File 1］为一个假设定义：即当 Symbol 1,Symbol 2,…在下一次操作时会操作 File 1。
- 对于 File 1 来说要写上 Symbol 1, Symbol 2,…的真的定义。

10.7.7 Order

- Order［式 1，式 2 ］式 1 式 2 的值的大小顺序为标准顺序为 1，如果式 1 式 2 的值相等为 0，如果不为标准顺序则为-1。

10.8 Message 和 error 的处理

SAD 在函数操作时若有 error 发生，则会生成与之对应的 Message。 随即此函数被中断，函数结果为其成为被调出时的形态。在此刻虽然 Message 会输出到端末，但也有因函数 Off 而抑制输出的情况发生。另外根据 Message 的重要性不同，其后的 Program 的实行会继续或中断。Message 的发生数可以由函数 Check 检出。

下一例便是这样的 Message 发生时的例子。

```
Log[1,2,3]+Sin[1,2]
???General::narg: Number of arguments is expected 1 or 2 in Log[1,2,3]
???General::narg: Number of arguments is expected 1 or 2 in Sin[1,2]
Out[4]:=(Log[1,2,3]+Sin[1,2])
```

在这里 General::narg 是能识别这个 Message 的表达式。在此例中首先有 Log[1，2，3]发生错误，但直接继续并实行其后的 Sin[1，2]，在那里仍生成同样的 Message。这些虽是同样的 Message，但末尾不同，实际上这个称为 General::narg 的 Message，是这样一个东西。

```
General::narg:
Out[5]：=“Number of arguments is expected ‘1’”
```

在这里的 1 可代入别的字符串。另外句首的??? 和句尾的 in…… 以后是系统会附加的东西。

使用者还可以在自己的 Program 中发送随时 Message。这样的 Message 的登陆由函数

MessageName 进行。

10.8.1 MessageName，::

- Symbol1::tag1　用 Symbol1 和 tag1 识别并返回登陆的 Message。
- Symbol1::tag1 的 Message 未登陆的情况下，返回 Symbol1::tag1 的 Message。
- Symbol1::tag1=Message1 登录用 Symbol1 和 tag1 识别的 Message1。Symbol1 和 tag1 可以是任意的表达式，通常使用符号。再者 ，Symbol1 和 tag1 不运算原样使用。
- Print[Definition[MessageName]] 在终端显示登录 Message。

10.8.2 Off

- Off[Symbol1::tag1]　以后停止 Symbol1::tag1 的 Message 输出。
- Off 执行后 Message 不能再 check.

10.8.3 On

- on[Symbol1::tag1]从此可以输出 Symbol1::tag1Message。
- on[General::newsym] 执行后每当新的名字符号生成时输出 General::newsymMessage。当检出 symbol 书写错误时可给予帮助。

10.8.4 $MessageList

符号$MessageList 当进行输入时，Message 识别符的 List 被分配，？？？

10.8.5 MessageList

- MessageList[n]　返回输出 n 行时发生的 Message 的识别子的 List，各识别子被 Hold。
- 下一输出行的序号分配给 Symbol$List（参照 out 10.9.3）。

10.8.6 Check

- Check[式 1，式 2] 对式 1 进行运算，这中间如果产生 Message 的情况时对式 2 进行运算，并返回其运算值。如果没有错误产生时返回式 1 运算的值。
- Check[式 1，式 2，Symbol1,tag1,..]当 Symbol1,tag1,..被识别产生 Message 的情况时仅对式 2 进行运算。
- Off 执行后 Check 对 Message 没有反应。

10.8.7 Message

- Message[Symbol1::tag1] Symbol1,tag1 被识别时生成 Message。

- Message[Symbol1::tag1，字符串 1，..] Symbol1::tag1 中的字符串“1”，..用字符串 1,..置换作为 Message 生成。

10.9 Debug 的方法

现在 SAD Debug 的工具还没有得到充实，下述作为原始的方法来考虑

- 较之 TracePrint[式 1]特定 Error 发生的场所。
- 关键处插入 Print[式 1] 或 Print[Definition[式 1]等。
- 较之 On[General::newsym]可检出新生成的 Symbol。
- 在 Program 最上层 Level 插入 end。

有时发生的 Error

- 大写与小写错误。
- 忘记写分号。
- 分号与逗号书写错误。
- 函数的参数类型，个数不对。

函数的用法未定义时，

???

10.9.1 Traceprint

- Traceprint[式 1]是在 CompoundExpression(;)的各种各样表达式执行前于终端显示。

10.9.2 Difinition

- Difinition[Symbil1]返回分配给 Symbil1 的定义式。

10.9.3 Out、%

- Out[n]在终端输出 out[n]:=…,返回对第 n 项运算结果的运算。
- %表示返回最后的运算结果。
- Out[n]与%Out[n]是等价的。
- 最后的运算结果的序号被设置给标识符$Line。虽然也可以自己重设这个序号，但此时新 $Line 以前的 Out[n]的值就会丢失。
- $Line 每当表达式的运算完结时都会递增 1，表达式的运算结果为 Null 时$Line 不更新。

10.9.4 标识符 end

如果在输入 program 的最上位的表达式与表达式的断开处插入标识符 end， SAD

在此中断，终端处于等待状态。用对话方式可确认 Symbol 的状态。

再运行时输入 in 77。

10.9.5 MemoryCheck

- MemoryCheck[]返回那时 Memory 的使用情况。检查系统的 bug 引起 Memory 的何种程度破坏，如果出现被破坏的情况则输出信息。
- 如果有信息输出请发给笔者报告。

10.9.6 STACKSIZ

- STACKSIZ 是指定运行 SADScript 解释语言必要的 STACK 大小的符号。
- 最初 FFS 对话激活前 STACKSIZ=n;由于指定过 STACKSIZ，以后可以改变 STACK 的大小。默认 STACKSIZ=200000。
- FFS 对话激活后，STACKSIZ 以后改变就没有效果了。

10.10 System 与 SAD 的相互作用

10.10.1 System

- System[Command 字符串] Command 字符串表现为系统的命令来执行。
- 欲读 command 执行的结果时使用 Openread(参照 13.1.1)

10.10.2 Evironment

- Evironment[环境参数字符串]返回用字符串表示的环境参数的值。
- GetEnv 与 Evironment 的值相同。

10.10.3 Directory

- Directory[] 返回当前程序的路径名。

10.10.4 SetDirectory

- SetDirectory[路径名 1] 当前程序的运行路径设定为路径名 1，并返回其完整的路径名称。

10.10.5 HomeDirectory

- HomeDirectory[] 返回 HomeDirectory 的名称。

10.10.6 GetPID

- GetPID[] 返回 SAD 的 processde ID 序号。

10.10.7 GetUID

- GetUID[]返回 SAD 的用户的 ID 序号。

10.10.8 GatGIG

- GatGIG[] 返回 SAD 用户的组号（GroupID）序号。

10.11 进程控制

10.11.1 Pause

- Pause[秒]。程序在指定的秒数内暂停运行。
- 一秒以内的指定也可以。
- Sleep 与 Pause 一样。

10.11.2 Fork

- Fork[] 复制运行中的进程,运行子进程。
- Fork[] 对于父进程返回子进程的进程号，对于子进程返回 0。

10.11.3 Wait

- Wait[] 等待子进程运行中止。
- Wait[] 返回 List{ SubProcess 序号，中止 code}。SubProcess 正常中止时中止 code 为 0。

10.12 其它的函数

10.12.1 Date

- Date[] 返回当时的年，月，日，时，分，秒的 List。

10.12.2 Day

- Day[] 返回当时星期几的名称。
- Day[{年，月，日}] 返回当天是星期几的名称。
- Day[年，月，日，时，分，秒] 返回当天是星期几的名称。

10.12.3 FromDate

- FromDate[{年，月，日，时，分，秒}] 返回从 1900 年 1 月 1 日 0 时 0 分 0 秒 起算到 List 表示的时刻经过的秒数。

10.12.4 ToDate

- ToDate[秒] 从 1900 年 1 月 1 日 0 时 0 分 0 秒 起算经过的秒数。以 List{年，月，日，时，分，秒}形式返回。

10.12.5 DateString

- DateString 以字符串的形式返回调用的时刻。

10.12.6 TimeUsed

- TimeUsed[] 以秒为单位返回 SAD 程序运行消耗的时间。

10.12.7 Timing

- Timing[式 1] 对式 1 进行运算，返回 List{式 1 的结果，消耗 Cpu 的时间。

11. 数值函数

11.1 初等函数

SAD 包含了以下初等函数。这原则上也适用于复数。还有，这函数如下所示，List 中的要素同样起作用。

- ArcCos ArcCosh ArcSin ArcSinh ArcTan ArcTanh Cos Cosh Exp Log Sin Sinh Sqrt Tan Tanh.

11.1.1 Log

- Log[z] 是 z 的自然对数
- Log[a,z] 是以 a 为底 z 的自然对数

11.1.2 ArcTan

- ArcTan[z]＝$Tan^{-1}z$
- ArcTan[x,y]=arg(x+iy)取±π之间的值 。

11.2 特殊函数

至今已有如下特殊函数，对应实际的要求看来还不够丰富。它们原则上可适用于复数，有时仅限于实数。如果必须引入复数时参数用 u,v,w,z, v 表示。这些函数在表中的参数同样起作用。

11.2.1 BesselI

- BesselI[ν,z]=$I_{\nu}(z)$

11.2.2 BsseIJ

- BsseIJ[ν,z]=$J_{\nu}(z)$

11.2.3 BsseIK

- BsseIK[ν,z]=$K_{\nu}(z)$

11.2.4 BsseIY

- BsseIY[ν,z]=$Y_{\nu}(z)$

11.2.5 Gamma

- Gamma[z]

$\Gamma(z)=\int_0^{\infty}t^{z-1}e^{-t}dt$ 。

- Gamma[a,x]

$$\Gamma(a,x)=\int_x^{\infty}t^{a-1}e^{-t}dt$$ 。

11.2.6 Factorial

- Factorial[z]与 Gamma[z＋1]具有相同的值。

11.2.7 LogGamma

- LogGamma[z] 定义与 Log[Gamma[z]]具有相同的值。可避免计算中 Gamma[z]的溢出。

11.2.8 LogGamma1

- LogGamma1[z] 定义与 Log[Gamma[z＋1]]具有相同的值。可避免计算中 Gamma[z＋1]的溢出。

11.2.9 GammaRegularrizedQ

- GammaRegularrizedQ[a,x]:

$Q(a,x)=\frac{\Gamma(a,x)}{\Gamma(a)}$ 。

11.2.10 GammaRegularrized

- GammaRegularrized[a,x]与 GammaRegularrizedQ 具有相同的值。

11.2.11 GammaRegularrizedP

- GammaRegularrizedP[a,x]:

$P(a,x)=1-Q(a,x)=1-\frac{\Gamma(a,x)}{\Gamma(a)}$ 。

11.2.12 Erf

- Erf[z]:

$\mathrm{erf}(z)=\frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}dt$ 。

11.2.13 Erfc

- Erfc[z]:

$\mathrm{erfc}(z)=1-\mathrm{erf}(z)$ 。

11.3 数值函数

SAD 备有以下函数

- Abs Ceiling Floor Max Min Mod Round Sign

其中 Abs Ceiling Floor Round Sign 对于 List 中各元素同等地起作用。

11.4 复数运算

- Complex ComplexQ Conjugate Im Re

11.4.1 Complex

- Complex[x,y]用 x+y*I 来表示，这里 x 与 y 是否复数没有关系。
- 符号 I 与 Complex[0,1]具有相同的值。

11.4.2 ComplexQ

- ComplexQ[式 1] 式 1 的值不是实数的复数，或者至少有一个元素的数值是那样的数 的 list 时 ComplexQ[式 1]为 True(1),其他的情况时为 False(0)。

11.5 傅立叶变换

11.5.1 Fourier

- Fourier[List 1]是求取由复数构成的 List 1={x1,x2,…}的高速 Fourier 变换。

$$\bar{x}_m = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k \exp\left(\frac{2\pi i k m}{N}\right)$$

返回复数 List 的结果。这里 N 是 List 的长度。

N 不是 2 的幂时，求其 2 的幂后加零即可变换。

11.5.2 InverseFourier

- InverseFourier[List 1] 求取由复数构成的 List 1={x1,x2,…}的高速 Fourier 逆变换。

$$\bar{x}_m = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k \exp\left(-\frac{2\pi i k m}{N}\right)$$

返回复数 List 的结果。这里 N 是 List 的长度。

N 不是 2 的幂时，求其 2 的幂后加零即可变换。

11.6 行列式运算

以下用向量用一阶 List,行列式用二阶 List 表示。这种情况下认为行列式是向量的 List。

SAD 准备了下列行列式的运算函数。

- Det Dot Eigensystem IdentityMatrix Inner LinearSolve Outer SingularValues Transpose

11.6.1 Dot

- 行列式 1.行列式 2 返回两个行列式的积。

11.6.2 Transpose

- Transpose[行列式 1] 返回行列式 1 的转置行列式。

11.6.3 LinearSolve

- LinearSolve[行列式 1，Vector 1] 求方程式（行列式 1.解＝＝Vector1 的唯一解），行列式 1 不一定是方阵。
- 用 LinearSolve 可简单地进行线形回归计算。
- LinearSolve[行列式 1，Vector 1，Tolorance→ε] 设定 Singular Values decomposition 的阈值 ε 后求解。Tolerance 的默认值为 10^{-18}。

11.6.4 SingularValues

- SingularValues[行列 1]求出行列 1 的 singular value decomposition。
- 尽管 SingularValues[行列 1]返回 list{u,w,v}，行列 1 的实效的逆行列用 Transpose[v].DiagonalMatrix[w].u 来表示。也就是说向量 w 是 singular value 的实效的逆

数的 list。

- SingularValues[行列 1，Tolerance －＞ε］把 singular value decomposition 的しきい値设定给ε。Tolerance 的 default 值为 10^{-18}。

在这里，『实效的逆数』是把某个 singular value 作为 wi、把最大的 singular value 作为 wmax 的时候，用 wi/(ε²w²max＋w²i)来表示的量。

11.6.5 Eigensystem

- Eigensystem[行列 1]对于正方行列行列 1，返回一个叫做{固有值群，右固有向量群}6 的 list。

11.6.6 Inner

- Inner[积 1，list 1，list2，和 1]返回 list 1 和 list2 的被一般化的内积。在这里积 1 与和 1 是分别完成积与和分工的函数。

11.6.7 Outer

- Outer[积 1，list 1，list2，……]返回 list 1，list2……被一般化的外积。这里积 1 是完成积的任务的函数。
- list 1，list2……虽然头部可以不是 List，但是也需要保持头部的一致。

11.7 随机数

11.7.1 Random

- Random[]返回 0 和 1 之间的一个随机数。
- Random[n]把 0 和 1 之间的 n 个随机数作成 list 返回。
- Random[n1，n2，……］把 0 和 1 之间的随机数群变成各阶要素数为 n1，n2，……的向量返回。越是往后的 index 越先执行。例：

```
In[7]:= Random[3,4]
Out[7]:=
{{.585043451981619,.803789990255609,.117949665756896,.023285944247618},
 {.996465738164261,.621301805833355,.237956767203286,.773597963387147},
 {.056013017194346,.605192655930296,.652846464188769,.759217258775607}}
```

11.7.2 GaussRandom

- GaussRandom[]返回平均值 0、分散 1 的正规拟似随机数。
- GaussRandom[n] 把平均值 0、分散 1 的 n 个正规拟似随机数变为 list 返回。

- GaussRandom[n1，n2，……]把平均值 0、分散 1 的 n 个正规拟似随机数变成各阶要素数为 n1，n2，……的张量返回。越是往后的 index 越快。

11.7.3 SeedRandom

- SeedRandom[]返回此刻的拟似随机数的种的值。
- SeedRandom[奇数 1]把拟似随机数的种设定为奇数 1。奇数 1 的绝对值不会大于 2^{31}。

11.8 方程式的近似解

11.8.1 FindRoot

- FindRoot[式 1＝＝式 2，{未知数 1，初期值 1}，{未知数 2，初期值 2}，…，option]意思为方程式式 1＝＝式 2 的近似解是把未知数 1、未知数 2，…作为未知数来求解。未知数 1，…为标识符，必须在式 1＝＝式 2 中完全显露出来才可以。
- 未知数 1、未知数 2，…的初期值用初期值 1，……表示。
- FindRoot 把结果用{未知数 1－>解 1}这个 list 返回。
- FindRoot[x==Cos[x],{x,0}]→{x ->.739085133215161}。
- FindRoot[{式 11＝＝式 12，式 21＝＝式 22…},{未知数 1，初期值 1}，{未知数 2，初期值 2},…，option]是用来解连立方程式的近似解的。
- 把各未知数的指定如{未知数，初期值，{下限值，上限值}}来写，就可以限制未知数的检索范围了。
- FindRoot 以牛顿法为基本法来求解。
- 解不存在时把**残差**的 2 乘和的极小值作为解返回。
- FindRoot 有表 31 的 option。

表 31: FindRoot 的选项

选项	值	缺省值	效果
AccuracyGoal	数值	10^{-20}	相对于 2 乘残差两边的 2 乘最大值的相对精度
MaxIterations	数值	50	在迭代中计算方程式的最大回数

11.9 非线性回归

11.9.1 Fit

- Fit[list 1,式 0，标识符 0，{参量 1，初期值}，…，option]是对于用 list 1 表示的数据点，要移动参量 1，…求出式 0 的 X^2-Fit。
- 式 0 是暴露地包含标识符 0，参量 1，…的表达式。Fit 的第二个以后的参数因为不会事先被运算，为了得到这样的『暴露地包含的表达式』 必须在这里写 Evaluate[式 00]。（参照 Evaluate 10.6.3）在这里式 00 是把式 0 作为结果的表达式。
- 把各个参量变成{参量,初期值，{下限值，上限值}},就可以限制其参量的检索范围。
- list 1 是如{{x1,y1, δy1},{ x2,y2, δy2},…}这样的构造。第三个数值为 error bar 的大小，是 X^2的重量。
- 各数据点的第三个的数值要被指定的时候必须对所有的数据点而指定。
- 各数据点的第三个的数值不存在时，各点的**重量**是均等的，error bar 的大小为了使 X^2 与 Fit 的结果一致而决定。
- list 1 的数据的构造与 ListPlot 14.3.11 是共通的。
- FitPlot 14.8 可以把结果用图表表示出来。
- Fit 把规则为 标识符->值 的 list 作为结果返回。在这里标识符的意义示于表 32。
- 用 Option MaxIterations 可以指定探查中的式 0 的运算次数的上限值。（default：40）。

11.10 函数的最小化

表 32：表示 Fit 的结果的标识符。在这里，n、m 表示各自数据点、参量的数。

Symbol	值	含义
参数 1, …	实数值	各参数的最合适值
ChiSquare	实数值	$x^2 = \Sigma^n_i (f(x_i)-y_i)^2/\sigma_i^2$
GoodnessOfFit	实数值	$\Gamma[(n-m)/2,\ x^2/2]/\Gamma[(n-m)/2]$
ConfidenceInterval	元素数 m 的	参数 1, …的推导域
CovarianceMatrix	m×m 矩阵	协方差矩阵

12. 字符串的处理

12.1 部分字符串的取出

- 字符串 1[n]返回字符串 1 的第 n 个文字。
- n 为负的时候，字符串 1 把倒数第－n 个文字返回。
- 字符串 1[n1,n2]把字符串的从第 n1 个文字到第 n2 个文字包含的字符串返回。n1，n2 都为负数也没有关系。

12.2 向字符串的变换

SAD 的原子和表达式都可以转换成字符串。一部分的函数、Print、Write，StringJoin 等参数可以自动被变换成字符串。

12.2.1 ToString

- ToString[式 1]返回表现表达式 1 的字符串。
- ToString[式 1，FormatType －>InputForm]在式 1 为字符串的时候，会把它变换为输入形式。输入形式就是指：以 Print，Write 形式输出的东西直接输入进去就变成原本的字符串。
- 包含于式 1 的实数值遵从$FORM 的值而变换。

12.2.2 $FORM

$FORM 是把实数值变换成字符串时来指定 format 的标识符。此值一旦指定了就会被保存。对$FORM 赋予字符串。

- $FORM＝“珩数 1.少数部 1”把变换的全珩数指定为珩数 1，少数部的长度指定为少数部 1。如果要变换的数在指定的范围内无法表现完全的话会自动地切换成指数表示。
- $FORM＝“F 珩数 1.少数部 1” 把变换的全珩数指定为珩数 1，少数部的长度指定为少数部 1。如果要变换的数在指定的范围内无法表现完全的话仅有珩数的长度用“*”表示。
- $FORM＝“S 珩数 1.少数部 1” 把变换的全珩数指定为珩数 1，少数部的长度指定为少数部 1。如果要变换的数在指定的范围内无法表现完全的话会自动地切换成指数表示。先导的空白将被消除，而且终端部的 0 会被消除。
- $FORM＝“M 珩数 1.少数部 1” 把变换的全珩数指定为珩数 1，少数部的长度指定为少数部 1。如果要变换的数在指定的范围内无法表现完全的话会自动地切换成指数表示。先导的空白将被消除，而且终端部的 0 会被消除。指数部的表示不是“E”而会变成“10^”。
- $FORM＝“”为标准的 format（“S18.15”）。

12.2.3 PageWidth

PageWidth 用于指定输出文件中记录的长度。其初值为 131 到端末的表示幅－1 之间的较小值。超过 PageWidth 的输出会自动地有改行插入。

12.2.4 StandardForm

- StandardForm[式 1]是设定了$FORM=“” ；PageWidth=2147483647 以后，再运算式 1，并返回其结果。另外实行后会把$FORM 以及 PageWidth 的值变回原来的值。
- StandardForm[$FORM=format；式 1]，写成这样，就可以局部地把$FORM 设定成 format1。

12.2.5 SymbolName

- SymbolName[标识符 1]会返回标识符 1 的字符串表现。这与 ToString[标识符 1]是等价的。

12.3 字符串的结合

12.3.1 StringJoin,//

- 字符串 1// 字符串 2// …是返回把字符串 1，字符串 2，…按这个顺序结合的字符串。
- 式 1// 式 2// …是把式 1，式 2，…按照这个顺序变换成字符串的字符串再结合成字符串返回。

12.4 字符串的比较

12.4.1 Equal,==

- 字符串 1＝＝字符串 2 的意思是：当字符串 1 与字符串 2 相等时返回 True（1）、不相等时返回 False（0）。两边形式不同的时候会把这个表达式本身返回。

12.4.2 Unequal,<>

- 字符串 1<>字符串 2 的意思是：当字符串 1 与字符串 2 不相等时返回 True（1）、相等时返回 False（0）。两边形式不同的时候会把这个表达式本身返回。

12.5 字符串的对应

StringMatchQ 等若干函数里可以做到通过 **wildcard** 实现的字符串的对应。但这毕竟只是作为字符串的对应，与前面说过的通过 pattern 实现的对应是完全两个概念。

12.5.1 通配符(wildcard)

- “*”对应于 0 或者 0 以上的任意长的字符串。
- “%”对应于任意的 1 个文字。
- “{…}”对应于被它包围的文字群中含有的任意一个文字。

- “{^…}”对应于不含在被它包围的文字群中的任意一个文字。
- “…|…|…”是对应于用“|”区分的部分中的至少一个。

12.5.2 StringMatchQ

- StringMatchQ[字符串 1，字符串 pattern 1]是当字符串 1 与字符串 pattern 1 对应时返回 True（1）、不对应时返回 False（0）。
- StringMatchQ[“aabcdee”,“a*e”] →True。

12.6 字符串的演算

12.6.1 StringLength

StringLength[字符串 1]返回字符串 1 的文字数。

12.6.2 StringPosition

- StringPosition[字符串 0，字符串 1]把包含在字符串 0 里的部分字符串字符串 1 的位置群以{{始点 1，终点 1}，{始点 2，终点 2}}的 list 形式返回。

  StringPosition[“abccddcce”,“cc”]{{3，4}，{7，8}}。
- 位置群有可能互相重叠的情况。
- StringPosition[字符串 0，字符串 1，n]仅仅返回最初的 n 个位置群。
- StringPosition[字符串 0，{字符串 1，字符串 2，…}]把字符串 1，字符串 2…的所有位置群返回。

12.6.3 StringInsert

- StringInsert[字符串 0，字符串 1，n]把字符串 1 插入到字符串 0 的位置 n（负的也可以）的新字符串返回。

12.6.4 StringDrop

- StringDrop[字符串 1，n]把除去了字符串 1 最初的 n 个文字的字符串返回。
- StringDrop[字符串 1，－n]把除去了字符串 1 最后的 n 个文字的字符串返回。
- StringDrop[字符串 1，{n}]把除去了字符串 1 的第 n 个文字的字符串返回。
- StringDrop[字符串 1，{n1,n2}]把除去了字符串 1 的第 n1 到第 n2 个文字的字符串返回。

12.6.5 StringFill

- StringFill[字符串 1，字符串 2，n]是在字符串 1 的后面反复补加字符串 2，做一个长度

为 n 的字符串。

- 当字符串 1 比 n 短的时候，可以切到 n 个文字。
- StringFill[“abc”,“def”, 10] →“abcdefdefd” 。

12.6.6 ToCharacterCode

- ToCharacterCode[字符串 1]把字符串 1 的各个文字的 ASCII code 以 list 形式返回。
- ToCharacterCode[“Hello!”] →{72,101,108,108,111,33}。

12.6.7 FromCharacterCode

- FromCharacterCode[list 1]把从 ASCII code 形成的 list 1 的对应于各自文字的字符串返回。
- FromCharacterCode[{72,101,108,108,111,33}] → “Hello!”。

12.6.8 Characters

- Characters[字符串 1]返回包含于字符串的各文字的 list。
- Characters[“A string.”] →{“A”,“”,“s”,“t”,“r”,“i”,“n”,“.”}

12.6.9 ToUpperCase

- ToUpperCase[字符串 1]把字符串中的小写英文字母全部变换成大写字母并作成字符串返回。

12.6.10 ToLowerCase

- ToLowerCase[字符串 1]把字符串中的大写英文字母全部变换成小写字母并作成字符串返回

12.7 作为字符串的表达式的运算

12.7.1 ToExpression

- ToExpression[字符串 1]对于字符串表现的任意表达式进行运算，并返回其结果。
- ToExpression[“Sqrt[2I]”] →1+I。

12.7.2 Symbol

- Symbol[字符串 1]制作一个以字符串 1 为名称的标识符，并运算它。

13. 输入输出

SAD 中的文件输入输出原则上通过 Chenal 进行。Chenal 通过 OpenRead,OpenWrite,OpenAppend 等等与文件连接在一起。另外，通过$Input,$Output 等也可以把当时的输入输出的流通作为 Chenal 使用。除此之外，Pipe 也可作为通道使用。

13.1 文件输入

基本的文件输入根据以下的方式进行。

(1)a = OpenRead[“test.dat”]

(2)x = Read[a,Real]

(3)Close[a]

首先，象（1）这样，把想输入的文件名（字符串）输进 OpenRead，就可以打开某个输入通道。其通道的编号就被设定为符号 a。然后象(2)这样，使用通道 a 重复必要的 **Input** 作业结束后，象(3)这样，用 Close 把这个通道释放。

输入文件是由若干个 Record 形成的。Record 是被改行文字（“\n”）分切的任意长度的字符串。改行文字自身并不包含在 Record 里。另外，也不能改变 Record 的分切的文字。

13.1.1 OpenRead

- OpenRead [文件名 1]对于用文件名 1（字符串）表示的文件打开输入通道，并把其通道编号作为结果返回。
- OpenRead[“！命令 1”]执行实为系统命令的命令 1，根据其输出结果打开输入通道，并将其通道编号作为结果返回。例：OpenRead[“！ls —l”]。
- 如果发生某种错误了，OpenRead 将把符号 $Failed 作为结果返回，而不是编号。

13.1.2 Read

函数 Read 按以下的形式使用。

Read[通道，对象 1，选择 1，•••]

这里的通道 1 是用 OpenRead 打开的通道编号，或是$Output。

$Output 表示此时的输入的流通。

Read 的第二个参数用于指定被读入的对象。对各自的对象有可用第三个以后的参数指定的选择。选择是使用符号 —>值的规则来表示的。

选择（Option）的作用表示在表格 33。选择省略时就被设定成表中的默认值。

如果对 Read 的第二个参数指定了对象群的 list，Read 就会遵从其 list 从文件中依次读入对应的对象群，并把与指定的 list 同类型的 list 作为结果返回。另外，此对象 list 的阶数是

任意值。

对于 Read 当文件到达终端时，Read 将返回符号 EndOfFile。

- Read[通道 1]是从通道 1 开始读取表达式，并返回运算的结果。这时，如果一个 Record 不能让表达式完结，就继续读取直到完结。
- Read[通道 1，Word，选择 1，•••]是从通道 1 开始读入 1 语，并将其返回。
- Read[通道 1，Real，选择 1，•••]是从通道 1 开始读入 1 语，并返回一个将其作为表达式运算后的值。
- Read[通道 1，Expression，选择 1，•••] 是从通道 1 开始读入 1 语，并返回一个将其作为表达式运算后的结果。
- Read[通道 1，Character，选择 1，•••]是从通道 1 开始读入 1 个文字，并将其返回。
- Read[通道 1，String，选择 1，•••]是从通道 1 开始一直读到其 Record 的终端，并将其作为字符串返回。

表 33：Read 的选项

选项	缺省值	作用
WordSeparators	“,\t”	表示语句分隔的文字行
NullWords	False	当 True 时，在语句分隔文字连续处有“”
ReadNewRecord	Ture	当 False 时，不读其后的 Record 而返回“”

13.1.3 Skip

- Skip[通道 1，对象 1，n，选择 1，•••]是把 Read[通道 1，对象 1，选择 1，•••]重复 n 次，并把最终结果返回。
- Skip[通道 1，对象 1]与 Read[通道 1，对象]是等价的。

13.1.4 Close

- Close[通道 1]解放通道 1。

13.1.5 Get

- Get[文件名 1]是将写在文件名 1（字符串）上的表达式群依次运算，并将最后结果返回。
- 对于 Get 来说 OpenRead，Close 是没有必要的。
- 对于函数图书馆的载入要使用 Get。

- 如果使用 Autoload（10.7.6），可以只在必要的时候载入函数。

13.2 文件输出

基本的文件输出根据以下方式进行。

(1)a = OpenWrite[“test.dat”];

(2)Write[a,x, y, …];

(3)Close[a];

首先，像（1）这样，通过把想要输出的的文件名（字符串）输进 OpenWrite，可打开某个输出通道。其通道的编号就被设定为符号 a。然后象(2)这样，使用通道 a 重复必要的 Input 作业结束后，象(3)这样，用 Close 把这个通道释放。

输出通道除了用 OpenWrite，还可以用 OpenAppend 打开。这时文件将从终端开始写入。

输出文件的 Record 长度由符号 PageWidth 决定。PageWidth 的初期值是 131 或是端末的表示幅-1 的较小值。超出 PageWidth 的输出将被自动改行。

13.2.1 OpenWrite

- OpenWrite[文件名 1]是对于用文件名 1（字符串）表示的文件将打开输出通道，并返回其通道编号。文件将从先头开始写入。
- 若发生某种错误，OpenWrite 将返回符号$Failed,而不是其编号。

13.2.2 OpenAppend

- OpenAppend[文件名 1]是对于用文件名 1（字符串）表示的文件打开输出通道，并把其通道编号返回。文件将从终端开始被写入。
- 如果发生某种错误，OpenWrite 将返回符号$Failed,而不是其编号。

13.2.3 Write

- Write[通道 1，表达式 1，表达式 2，•••]对于通道 1 将运算表达式 1，并输出把其结果变换成字符串的结果。此动作对表达式 2 以后也同样进行，最后输出改行文字。
- 在表达式与表达式之间的分切字中什么都没有。
- 数值的向字符串的变换，要遵从$FORM 的值。假若途中想要改变$FORM 就写成 Write[•••, $FORM=格式化 1，表达式 1，•••]，这样表达式 1 以后的$FORM 就会变为格式化 1。若想保存原有的格式化可以利用 StandardForm 等。
- 作为通道 1 可以指定用 OpenWrite，OpenAppend 打开的通道编号或是$Output。$Output

表示此时的输出的流通。

13.2.4 Print

- Print[表达式 1，表达式 2，•••]对于此时的输出的流通$Output,将运算表达式 1，并输出把其结果变换成字符串的结果。此动作对表达式 2 以后也同样进行，最后输出改行文字。
- 在表达式与表达式之间的分切字中什么都没有。
- 数值的向字符串的变换，要遵从$FORM 的值。假若途中想要改变$FORM 就写成 Print[•••, $FORM=格式化 1，表达式 1，•••]，这样表达式 1 以后的$FORM 就会变为格式化 1。若想保存原有的格式化可以利用 StandardForm 等。

13.2.5 WriteString

- WriteString[通道 1，表达式 1，表达式 2，•••]对于通道 1，将运算表达式 1，并输出把其结果变换成字符串的结果。此动作对表达式 2 以后也同样进行，但是最后不输出改行文字。
- 在表达式与表达式之间的分切字中什么都没有。
- 数值的向字符串的变换，要遵从$FORM 的值。假若途中想要改变$FORM 就写成 WriteString[•••, $FORM=格式化 1，表达式 1，•••]，这样表达式 1 以后的$FORM 就会变为格式化 1。若想保存原有的格式化可以利用 StandardForm 等。
- 作为通道 1 可以指定用 OpenWrite，OpenAppend 打开的通道编号或是$Output。$Output 表示此时的输出的流通。

13.2.6 Close

Close[通道 1]解放通道 1。

14. 制图法

14.1 制图法的例子

以下的例子，是根据 SAD/Tkinter 制成的最为简单的制图法例题。

FFS;

```
W = Window[];                                          (1)
C = Canvas[w,Height -> 400,Width -> 600];              (2)
$DisplayFunction = CanvasDrawer;                       (3)
```

```
Canvas$Widget = c;                                          (4)
data = Table[{x,Sin[x] + 0.1 * GaussRandom[]};              (5)
  {x, -Pi,Pi,Pi/10}];
ListPlot[data];                                             (6)
TkWait[];                                                   (7)
```

此例会如下图所示在 Window w（中的 Canvas c）里表示出图表。

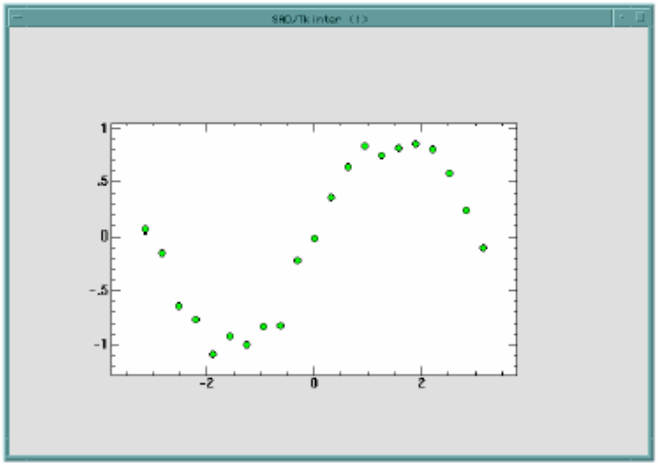

图 22：根据 ListPlot 制成的简单图表例。

上例中，首先（1），（7）是普通的 Tkinter 的操作，请分别参照 2.2 和 2.5。

(2)用于在 Window w 中制作 Tkinter 部品 Canvas，并将其尺寸指定为宽 600 像素，高 400 像素。

（3）指定了在以后使用的制图法输出函数。

（4）指定了成为制图法输出端的 Canvas 部品 c。

（5）制作以下我们要介绍的与制图有关的 Message 数据。数据的形式为{{x1，y1}，{x2，y2}，・・・｝。此例是对 sine 曲线加上正规拟似乱数的结果。请参照 Table7.2.1 ,GaussRandom11.7.2。

（6）会根据（オ）中制作的数据输出图表。在 SAD 中制图会被一个表达式表示出来，这个表达式是由 ListPlot,Plot,FitPlot 等生成的 object，并带有头部 Graphyics。这个 object 通过 ListPlot,Plot,FitPlot 和 Show 被输出。这些函数会把生成的 Graphyics object 作为结果返回。

如此例，图表的位置范围等，即使不指定，它也会自动调节，但是也可以通过指定选择而随意地改变。

选择本身作为制图输出函数的 TopDrawer（默认值）时，（1）--（4）以及（7）操作都不需要了。

14.2 制图法的输出函数

输出函数由选择 DisplayFunction 指定。DisplayFunction 的默认值是$DisplayFunction,而$DisplayFunction 在之前已被设定了 TopDrawer。

输出函数也可象前面的例子那样选择 CanvasDrawer。CanvasDrawer 将对 SAD/Tkinter 部品之一的 Canvas 进行输出。

我们输出的制图法会由 TopDrawer，CanvasDrawer 尽可能的调整到相同，但是由于机种的差异以及实际安装的滞后还会有很多不完善的地方。我们准备今后根据需要继续调整使之完备。

14.3 图表的制成

现在 SAD 备有表 34 的用于制成图表的函数。

无论哪个函数都可以作为参数指定选择。这些选择的大部分对哪个函数都是通用的，选择以选择符号->值 形式出现。表 35 中展示了这些选择。除此之外，对于每个函数还有固定的函数，我们将在各自函数的项目中说明。

下面我们以 ListPlot 为例说明各个选择。

表 34: 各函数的作图

函数	作成的图像
ListPlot	数值列表图像
Plot	函数图像
Columplot	柱状图
OpticsPlot	光学参数的图像
FitPlot	非线性回归图像

14.3.1 要 Plot 的数据的范围，PlotRange

一般地，要 Plot 的数据的范围通常都会由 Message 数据本身自动地设定，但也可以通过选择 PlotRange 来调节。符号 Automatic 指示着使用自动设定值。

- PlotRange ->{ymin,ymax}将 y 轴范围定义为下限 min 和上限 max。
- PlotRange ->{ymin,Automatic}将 y 轴范围定义为下限 min 和上限为自动设定值。

- PlotRange ->{{xmin,xmax}，{ ymin,ymax }}将 x 轴范围定义为下限 xmin 和上限 xmax，y 轴范围定义为下限 ymin 和上限 ymax。
- PlotRange ->{{xmin,xmax}，Automatic}将 x 轴范围定义为下限 xmin 和上限 xmax，并将 y 轴范围定义为自动设定值。
- 若把图 22 例变成 ListPlot[data,PlotRange ->{{-10,10},{-2,3}}]的样子，结果则如图 23 所示。

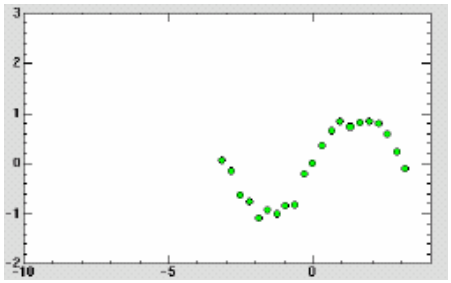

图 23：由 PlotRange 得到的表示的数据的范围指定。

表 35：作图时函数的选项

选项	值	缺省值	效果
AspectRatio	实数值	GoldenRatio	横/纵比
DisplayFunction	输出函数	TopDrawer	图像输出函数
Epilog	图形元素的 List	{}	输出图像后显示的图像
ErrorBarTickSize	实数值	1	ErrorBar 键的相对长
Frame	True，False	True	图像的外框
FrameLabel	{下，左，上，右}	{“”，“”，“” ，“”}	座标轴的字符串
Initialize	True，False	True	座标系是否初始化
Plot	True，False	ListPlot:True Plot:False	数据点的表示
PlotClor	颜色	“black”	图像线条的颜色
PlotJoined	Ture，False	ListPlot:False Plot:True	是否用线条把数据点连起来
PlotLabel	字符串	“”	加在图像整体上的标签
PlotRange	座标系 x 轴，y 轴的范围	{Automatic,Automatic}	绘图信息的范围

	{x 范围，y 范围}		
PlotRegion	{{xmin, xmax},{ymin,ymax}}	{{0,1},{0,1}}	图像的相对位置
PointSize	实数值	1	点的大小
PointColor	颜色	“green”	点的颜色
Prolog		{}	输出图像前显示的图像
Scale	{xscale, yscale}	{Linear,Linear}	选择 Linear 或是 Log
Tags	True, False	False	在 Canvas$ID 上记录项目号
Background	颜色	“#ffffd0”	图像框色的背景色

14.3.2 Plot 的在 Canvas 中的位置指定，PlotRegion

Plot 通常从指定的 Canvas 画面自动决定其位置和大小，但也可通过 PlotRegion 变更为自由的位置和大小。

- 把可自动决定的位置范围设为{{0，1}，{0，1}}，用对应于自动值的相对值象 PlotRegion -> {{x 下限，x 上限}，{y 下限，y 上限}}这样指定。
- 若把图 22 例变成 ListPlot[data,PlotRegion ->{{0,0.5},{0.5,1}}]的样子，结果则如图 24 所示。

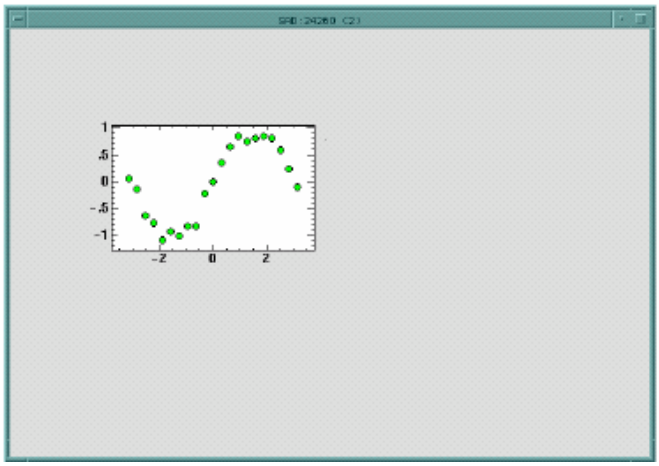

图 24：由 PlotRegion 得到的表示的位置的指定。

14.3.3 横/纵比，AspectRatio

- 通过 AspectRatio ->数值，可以指定 Plot 的横/纵比。

- 若把图 22 例变成 ListPlot[data，AspectRatio ->1]的样子，结果则如图 25 所示。

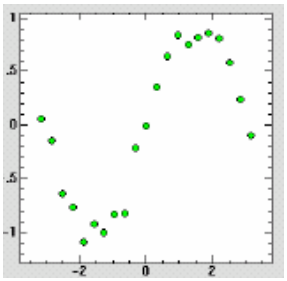

图 25：由 AspectRatio 得到的 Plot 的横/纵比的指定。

14.3.4 图表的外框及标尺的有无，Flame

14.3.5 附在图表外框的标签，FlameLabel

- FlameLabel ->{下标签字符串，左 FlameLabel
- 若把图 22 例变成 ListPlot[data,FrameLabel ->{“Bottom”，“Left”}]的样子，结果则如图 26 所示。

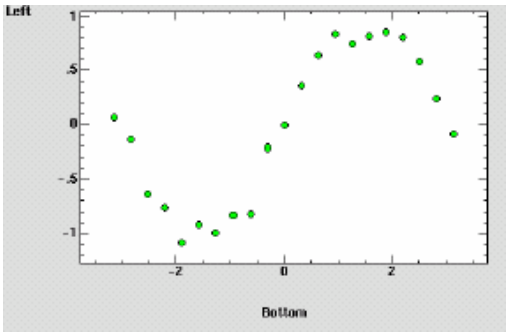

图 26：由 FrameLabel 得到的框的标签的指定。

14.3.6 图表整体的标签，PlotLabel

- 通过 PlotLabel ->字符串，可以对 Plot 整体贴上标签。
- 若把图 22 例变成 ListPlot[data，PlotLabel -> “PlotLabel”]的样子，结果则如图 27 所示。

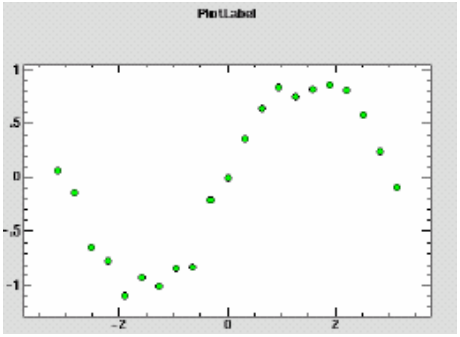

图 27：由 PlotLabel 得到整体的标签的指定。

14.3.7 在 Plot 前后的图形的写入，Prolog 与 Epilog

- 通过 Prolog ->制图法原子的 list，可以在 Plot 前写入其他的图形群。
- 通过 Epilog ->制图法原子的 list，可以在 Plot 后写入其他的图形群。
- 若把图 22 例变成 ListPlot[data，Prolog -> {Rectangle[{-1,-0.5},{1,0.5},FillColor -> “gray”]}]的样子，结果则如图 28 所示。(Rectangle 是后面将要提到的制图法原子之一，可画长方形)。

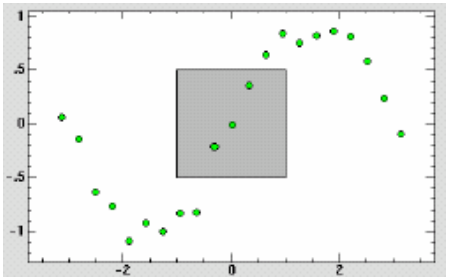

图 28：用 Prolog 在 Plot 前画入图形。

14.3.8 把数据点用线连接，PlotJoined 以及数据点的表示，Plot

- 通过 PlotJoined ->True，可以把数据点用线连接。
- 连线的颜色可以用 PlotColor 指定。另外，数据点的标注的表示可以用 Plot 切入。
- 若把图 22 例变成 ListPlot[data，PlotJoined -> True]的样子，结果则如图 29 所示。
- 若把图 22 例变成 ListPlot[data，PlotJoined -> True ，Plot -> False]的样子，结果则如图 30 所示。

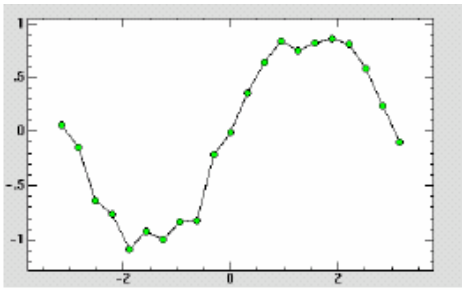

图 29：用 PlotJoined -> True 把数据点用线连接起来。

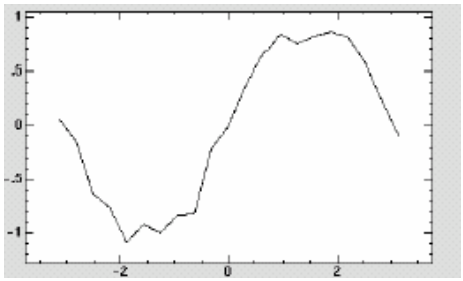

图 30：用 Plot -> False 把数据点的标注擦去。

14.3.9 标注的大小和颜色 PointSize，PointColor

- 使用 PointSize -> 数值可以指定标注大小，这种场合把标准值定为 1，以其倍率表示大小。
- 使用 PointColor -> 数值可以指定标注内部颜色。颜色的指定请参看 4．5．3。
- 若把图 22 例变成 ListPlot[data，PointSize -> 3，PointColor -> “White”]的样子，结果则如图 31 所示。

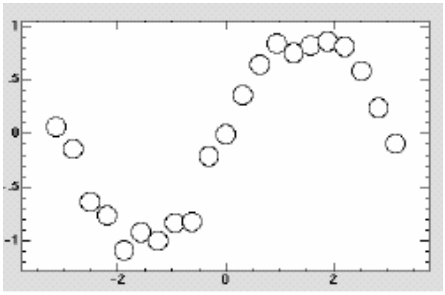

图 31：由 PointSize，PointColor 指定数据点的マーカー的大小和内部颜色。

14.3.10 对数图表，Scale

通过 Scale -> {刻度 x，刻度 y}可以将 x 以及 y 轴的刻度分别指定为 Linear 或者是 Log。Scale -> 刻度 y 与 Scale -> Linear，刻度 y 是等价的。

下面的例子是制作图 32。

```
x =  Range[0,5,0.5]
data =  Thread[{x,Exp[-x^2/2] / Sqrt[2 Pi]}];
ListPlot[data,Scale -> {Linear,Log}];
```

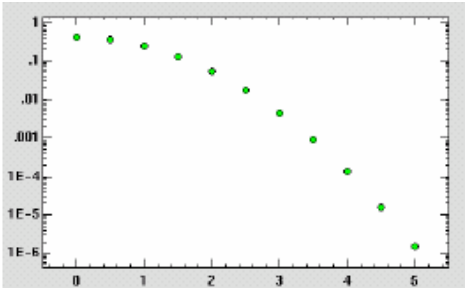

图 32：由 Scale -> {Linear,Log}(Scale ->Log 也可以)得到的片对数图表。

14.3.11 ListPlot，以及 ErrorBar 的表示

ListPlot[list 1，选择 1，•••]由上述所介绍的，是从数值数据的 list，list 1 里制作图表。

- List 1 以{{x1，y1}，{x2，y2}，•••}的形式表示各个数据点。
- 当 list 是一维数据{y1，y2，•••}时，认为 x 轴与之相对应的是 1，2，・・・。
- 当 list 如{{x1，y1，δy1}，・・・}这样各自由三个数值组成的情况时，第三个数值表示 y 方向的 ErrorBar 长度。当 list 如{{x1，y1，δx1，δy1}，・・・}这样各自由四个数值组成的情况时，第三个数值和第四个数值分别表示 x 方向和 y 方向各自的 ErrorBar 长度。
- ErrorBar 的键长可以由选择 ErrorbarTickSize 指定。
- 不论是哪种情况，list 1 都是长方形即各要素都必须是相同形状。
- 下面例子是连同图 22，附加了各点 y 方向 0.1 的 ErrorBar，并使用 ListPlot 表示的示图。结果如图 33 所示。

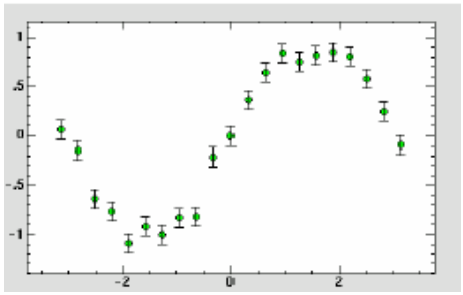

图 33：由 LisPlot 得到的 ErrorBar 的表示。

- 下面例子是连同图 22，附加了各点 x 方向 0.2,y 方向 0.1 的 ErrorBar，并使用 ListPlot 表示的示图。此例中还把键长设为 0。结果如图 34 所示。

```
data = Join[#,{0.2,0.1}]& /@ data;
ListPlot[data1, ErrorBarTickSize -> 0];
```

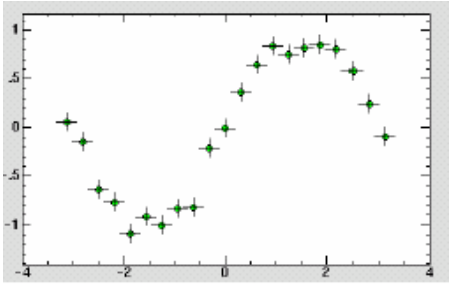

图 34：由 ListPlot 得到的双向的 ErrorBar 的表示。写为 ErrorbarTickSize -> 0。

14.4 Plot

Plot 用于制作 1 变数的函数的 Plot。例如，进行如下简单操作便可得到如图 35 这样的 Plot。

```
Plot[Sin[x], {x, -Pi, Pi}];
```

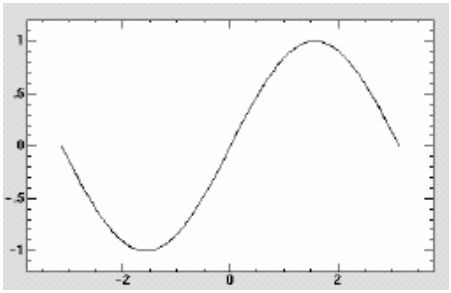

图 35：由 Plot 得到的 Sin[x]函数的表示。

另外，

```
Plot[{SiN[x], Cos[x]}, {x, -Pi, Pi}];
```

象上式这样，若把第一参数变成复数表达式的 list，就能得到图 36 所示的 Plot。

这时,各表达式的表示色会按照

{“black”,“red”,“blue”,“green”,“grey”,“magenta”,“cyan”}

的顺序循环地被选择，也可以通过指定 选择 PlotColor -> 颜色 list 从而随意地设定。

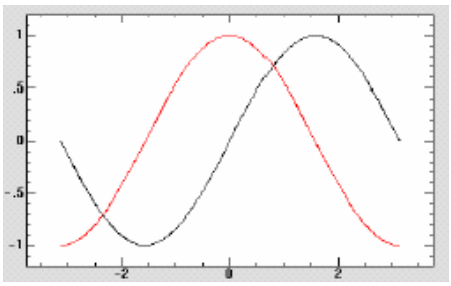

图 36：由 Plot 得到的复数函数 Sin[x]，Cos[x]的同时表示。Cos[x]的表示色为“red”。

Plot 的构文如下。

- Plot[表达式 1，{符号 1，下限值，上限值}，选择，•••]或者是 Plot[{表达式 1，表达式 2，・・・}，{符号 1，下限值，上限值}，选择，•••]。
- 在这里，表达式 1，表达式 2，・・・是将[独立变数]符号 1 不加遮饰地包含的表达式。

14.4.1 Plot 的选择

Plot 连同表 35，表 36 的 Plot 的固有选择也可以指定。

表 36：Plot 的固有选项

选项	值	缺省值	效果
MaxBend	实数值	0.04	线条弯曲角度的最大值
PlotDivision	实数值	250	表示区间的最大分割数
PlotPoints	实数值	25	分割点数目值
PlotColor	图像颜色	{“black”，“red”，“blue”，“green”，“gray”，“magenta”，“cyan”，“yellow”}	图像线条的颜色

14.5 由 ColumnPlot 得到的柱形图表的制成

ColumnPlot[数据 1，选择，…]可制成柱形图表。

数据 1 若是一维的 list，可制成单纯为柱形的图表。

ColumnPlot[1，4，9，16，25，16，9，4，1] → 图 37。

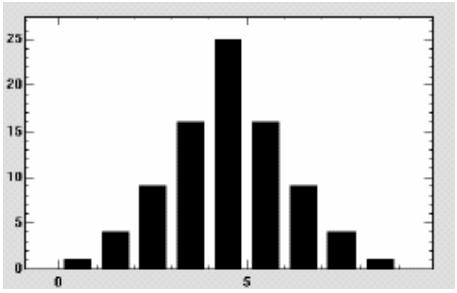

图 37：由一维 list 得到的 ColumnPlot。

- 数据 1 若是二维的 list，可在一个框里制成把多个柱组合起来的柱形图表。此时，数据 1 必须是长方形。
- ColumnPlot[{{1,3},{4,5},{9,7},{16,9},{25,11}}] → 图 38。
- 各个柱根据选择 FillColor 的值分颜色。

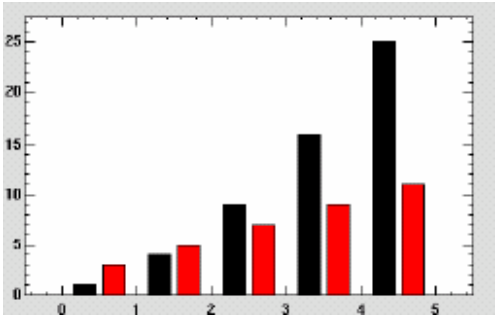

图 38：由二维的 list 得到的 ColumnPlot。

当数据 1 是三维的时候，第三阶的数据将把累积的柱堆积起来表示。

ColumnPlot[{{{1,5，9},{2,5，7}},{{3，3，3},{4，3，2}},{{9，2，1}，{6，5,4}}}]

→ 图 39。

堆积是根据选择 MeshStyle 的值，改变填充 Pattern 而表示的。

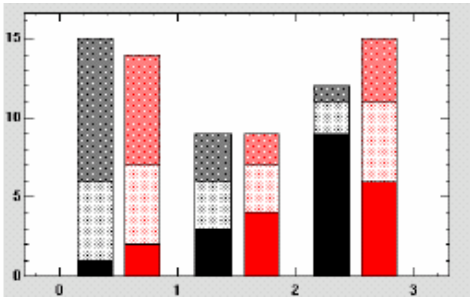

图 39：由三维的 list 得到的 ColumnPlot。

14.5.1 ColumnPlot 的选择

ColumnPlot 连同表 35 的选择，以及表 37 的 ColumnPlot 固有选择都可以指定。

表 37：ColumnPlot 的固有选项

选项	值	缺省值	效果
ColumnPlot	数值	0.15	柱与框之间缝隙的比率
FillColar	颜色	{“black”，“red”，“blue”，“green”，“gray”，“magenta”，“cyan”，“yellow”}	柱的颜色
MeshStyle	填充网格	{Null，“gray25”，“gray50”}	柱内填充网格，Null 时 100%涂满
Orientation	Horozontal，Vertical	Vertical	柱的方向

14.6 图表的合成，Show

由 ListPlot 和 Plot 制成的图表可以合成一个图表表示出来。下例可以制成如图 40 的图表。

```
g1 = ListPlot[data,DisplayFunction -> Identity];
g2 = Plot[Sin[x],{x,0,2Pi},DisplayFunction -> Identity];
Show[g1,g2];
```

在这里，要对 ListPlot 和 Plot 加上 DisplayFunction -> Identity 这样一个选择。

这里，在各个函数的阶段并不实行 plot，而只是为了把结果的 graphics object 分别设定给符号 g1，g2。然后到了最后的 Show 才首次执行 plot。图 40 中要注意轴的范围要设定成将 g1，g2 的两边包围的样子。

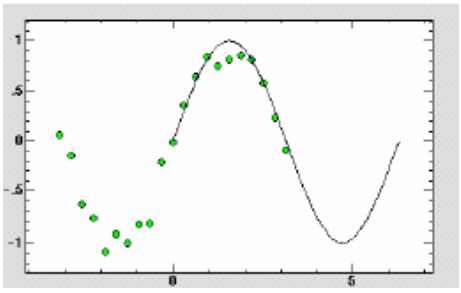

图 40：图表的合成

- Show[图象 1，图象 2，••• ，选择 1，•••]是将若干个图象合成并制作为一个 plot。这

里图象 1，••• 是 Graphics object 或者是它们的 list。

- 如果在各图象的选择中只能选一个的时候，就会选择最初的图象。
- Show 的选择优先于各图象的选择。
- 图表的范围虽然会被自动地设定成把整个图象包围的样子，但若把 PlotRange 对于 Show 设定好就可以随意地设置范围了。

14.7 图表的表示位置的设定

在 Canvas 中的某个图表的表示位置通常会由其 Canvas 的大小自动决定。但是使用函数 Rectangle 就可以随意地设定其位置，并且还能在一个 Canvas 中表示多个图表。例如，下例可以制作图 41 的图表。

```
g1 = ListPlot[data,DisplayFunction -> Identity];
g2 = Plot[Sin[x],{x,-Pi,Pi},DisplayFunction -> Identity];
Show[Graphics[ {
   Rectangle[{0,0},{1,0.5},g1],
   Rectangle[{0,0.5},{1,1},g2]} ]];
```

这里 Rectangle 本身不是 Graphics object 而是图象原子，因此要想交给 Show（复数的情况变成 list 形式）就要把 Graphics 覆盖住。

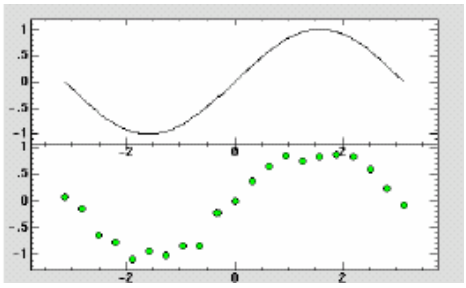

图 41：Rectangle 表示位置的设定

- Rectangle[{x 下限，y 下限}，{x 上限，y 上限}，图象 1]是在以{x 下限，y 下限}，{x 上限，y 上限}指定的长方形内部描绘图象 1 的图象原子。
- 标准的图表的位置是{0，0}，{1，1}，Rectangle 的参数是以此为基准，用相对值来表示的。
- 如果 Rectangle 的第三参数不是图象时，Rectangle 将表示长方形的图象原子（参照 14.9.3）。

14.8 FitPlot

FitPlot 是将非线形回归函数 Fit（参照 11.9.1）和 ListPlot，Plot 组合起来的东西。

- FitPlot[list 1，式 0，符号 0，{参数，初期值}，•••，选择]是对于用 list 1 表示的数据点，移动参数 1，••• 求出式 0 的 X^2-Fit，并将其结果 Plot。
- 式 0 是将符号 0，参数 1，••• 不加掩饰地包含的表达式。FitPlot 的第二个以后的参数由于没有提前运算过，因此若想得到这种「不加掩饰地包含的表达式，有时就必须在这里写上 Evaluate[式 00]」（参照 Evaluate10.6.3）。在这里表达式 00 是将表达式 0 作为结果带来的表达式。
- 将各参数写为{参数，初期值，{下限值，上限值}就可以限制其参数的检索范围。
- List 1 的数据的构造与 ListPLot 是相通的。出现 ErrorBar 时它将成为 X^2 的权。当没有 ErrorBar 时权为均等的。
- FitPlot 把{Fit 的结果，Graphics object}这个 list 作为结果返回。
- 下例是对于图 33 的例使用 FitPlot 根据 sine 曲线求出 X^2-Fit，并将其结果 plot 了的东西。

```
data 1=Append[#,0.1]& /@ data;
FitPlot[data 1,e Sin[f x + g],x,{e,1},{f,1},{g,0}];
```

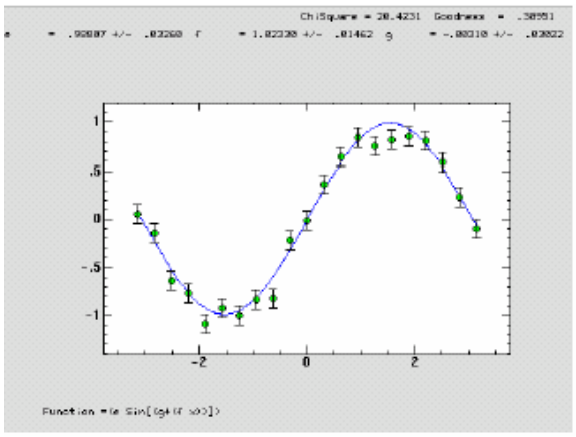

图 42：FitPlot 的例子。Fit 的式是 e*Sin[f x + g],独立变数是 x,参数是 e,f,g.

14.9 制图原子

制图原子是构成 SAD 图表的基本要素。制图原子带上符号 Graphics 就变成 Graphics object，通过 Show 变为被 Plot 的东西。ListPlot 和 Plot 返回的也都是这样的 Graphics object。现在，Point,Line,Text,Rectangle 也作为这样的制图原子而被定义。各原子通过选择指定可

以具有各种属性。我们今后也会根据需要完善这样的选择。

14.9.1 Point

Point[list1，选择 1，•••]显示的是表示数据点的 list1 的各点的标记。

- List1 的形式与 ListPlot 14.3.11 相同。从而会有伴有 Errorbar 的情况。
- 选择见表 38。

表 38: Point 的选项

选项	值	缺省值	效果
ErrorBarTickSize	实数值	1	ErrorBar 键的相对长度
PointSize	实数值	1	点的大小
PointColor	颜色	“green”	点的颜色

14.9.2 Line

- Line[line 1，选择 1，•••]表示的是将显示着数据点的 list 1 各点连接的线。
- List1 的形式与 ListPlot 14.3.11 相同。从而会有伴有 Errorbar 的情况。
- 选择见表 39。

表 39: Line 的选项

选项	值	缺省值	效果
ErrorBarTickSize	实数值	1	ErrorBar 键的相对长度
Plot	True, False	False	数据点的表示标记
PlotJoined	True, False	True	是否用线条连接数据点
PlotColor	颜色	“black”	图像线条的颜色
PointSize	实数值	1	点的大小
PointColor	颜色	“green”	点的颜色

14.9.3 Rectangle

- Rectangle[{x1，y1}，{x2，y2}，选择 1，•••]表示的是由数据点{x1，y1}，{x2，y2}决定的长方形。

- Rectangle[{x1，y1}，{x2，y2}，Graphics[•••]]表示的是长方形里另画了别的图形的东西。
- 选择见表 40。

表 40: Rectangle 的选项

选项	值	缺省值	效果
FillColor	颜色	Null	矩形内部的颜色，Null 时只涂框
MeshStyle	填充网格	Null	柱内填充网格，Null 时 100%涂满
PlotColor	颜色	“black”	框的颜色
Tags	字符串		附加的 Tag（参照 15.1.3）

14.9.4 Text

- Text[{字符串 1，{x，y}}，选择 1，•••]表示的是在位置{x，y}上表示的字符串。
- 坐标{x，y}是 CanvasDrawer 和 TopDrawer 的固有坐标，它以 Canvas 的左下角设为{0，0}，右上为正方向。通常图表的左下为{2.4,1.8}。
- 作为坐标如果如{Scaled[x]，Scaled[y]}这样指定，就可以得到由数据点决定的坐标。
- 选择见表 41。

表 41: Text 的选项

选项	值	缺省值	效果
PlotColor	颜色	“black”	文本的颜色
Tags	字符串		附加的 Tag（参照 15.1.3）
TextAlign	“left”，“center”，“right”	“left”	文本的标准位置
TextFont	字型	$DefaultFont	字型（参照 4.5.2）
TextSize	实数值	1	文字的大小

15. 画布（Canvas）

在前面我们已经看到了，Canvas 作为表示 ListPlot 等绘图的领域而被使用着。如果只是简单

的图表的表示，使用 ListPlot 就足够了，但有时对于 Canvas **会有用到直接作用的必要。**例如，点击描绘的图形的一部分进行某种操作，或是进行图形的移动和属性的变更等工作。在这里用简单的例子对这样的 Cancas 特有的操作进行说明。

15.1 Canvas 的 item

Canvas 中描绘的部品被分类成若干个 item。表 42 是这些 item 的一览表。

表 42: Canvas 的 item

选项	所描绘的图形
Arc	扇形，弦，弓形
Bitmap	位图
Image	Image
Line	直线
Oval	椭圆
Polygon	多边形
Rectangle	矩形
Text	字符串
Window	要嵌入其他构件的框

这些 item 用 Canvas 如下制成。

```
FFS;
w = Window[];
c = Canvas[w, Width -> 300, Height -> 300];
c[Create$Rectangle] = {50, 50, 250, 250,
Tags -> "rectarc rectoval"}};
c[Create$Oval] = {50, 50, 250, 250,
Fill -> "blue", Stipple -> "gray25",
Tags -> "rectoval"};
c[Create$Arc] = {50, 50, 250, 250,
Style -> "pieslice", Fill -> "red",
Start -> 60, Extent -> 90,
Tags -> "rectarc"};
```

TkWait[];

此例中，把 Rectangle,Oval,Arc 这三个 item 如图 43 画出。

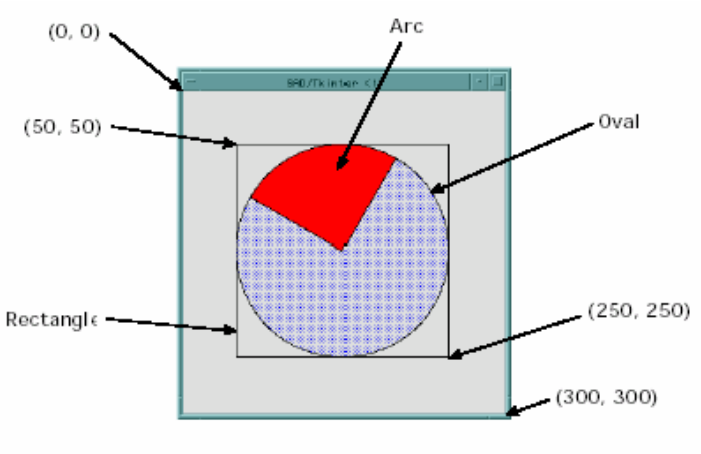

图 43：Canvas 的例子。座标如图所给。

象这样，如果想要对 Canvas 追加某个 item，就写成：

Canvas 符号[Create$item]={パラメータ 1,...パラメータ n,属性 1->值 1,属性 2->值 2,...}

在这里，パラメータ 1,... パラメータ n，是一组或一组以上的座标。这些分别写或是写在一起作 list 都可以。上例中，Create$Rectangle 的四个参数相当于此パラメータ。把下面的属性的指定根据需要进行。パラメータ群必须在属性指定之前书写。

15.1.1 Canvas 的座标

Canvas 中座标的指定与 Window 的座标指定一样，把左上角设为（0，0）,并向右下增加，单位是像素。此座标与 CanvasDrawer 的座标不同，请勿混淆。

15.1.2 item 序号

对于一个 Canvas,制作的各 item 会按制作顺序自动添加 item 序号。此序号从 1 开始顺序递增。即使从途中因为 Delete 等操作使 item 被消去，相同的序号也不会再被利用。此序号是在后面对各 item 进行各种操作时用到。

15.1.3 Tags

Canvas 的 item 具有一个叫 Tags 的属性，各 item 分成组，还可以加上名称。之后在各组就可以容易的进行各样的操作。名称为不包含空白，{},等文字的任意字符串。

一个 item 也可以所属于多个组。例如上例中，对于 Retangle 有“rectarc”和“rectoval”

两个组指定。（虽然书写方式有点怪但先这样）

例如上例中，如果写成这样，就会如图 44 结果所示。

```
c[Move] = {"rectoval", 30, -20};
c[Update];
```

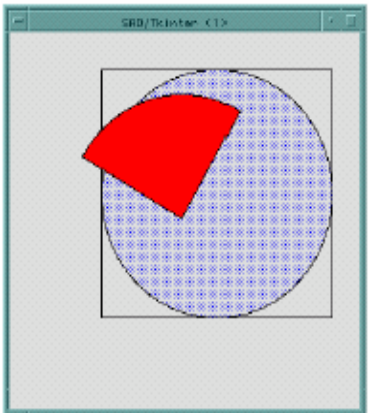

图 44： Tags 的组化的 item 群的移动。包含了 Retangle 和 Oval 的组（“rectoval”）仅移动了（30，－20）。

15.1.4 Arc

A rc 描绘扇形，弓形，弧。Create$Arc 是把其 A rc 作为一部分弧的椭圆的外结长方形的两个对角顶点的座标作为最初的四个参数。

表 43 有其属性。

表 43: Arc 的 属性

选项	值	缺省值	效果
Extent	角度	360	弧逆时针方向的开角
Fill	颜色	无指定	内部颜色
Outline	颜色	“black”	轮廓线的颜色
Start	角度	0	弧的起始角，按逆时针方向
Stipple	位图		全面涂抹后的位图
Style	“pieslice”，“chord”，“arc”	“pieslice”	扇形，弓形或弧的选择
Tags	字符串		加在其他项目上的 Tag 群

Width	数值	1	轮廓线的宽度

另外，下例中结果如图 45 所示。

```
w = Window[];
c = Canvas[w, Width -> 380, Height -> 140];
styles = {"pieslice", "chord", "arc"};
x = 20;
Scan[
(c[Create$Arc] = {x, 20, x + 80, 100,
Style -> #, Fill -> "red",
Start -> -30, Extent -> -120};
c[Create$Text] = {x + 40, 120,
Text -> #};
x += 120)&,
styles];
```

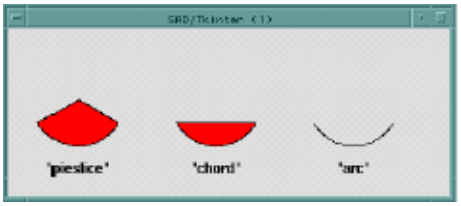

图 4 5：A rc 的三种 Style。

15.1.5 Bitmap

Bitmap 在 Canvas 上表示 Bitmap。

Create&Bitmap 把指定位置的一组座标作为参数。另外，其座标和与 Bitmap 的相对位置用属性 Anchor 指定。 Bitmap 的属性揭示在表 44。另外下例程序是制作 4.5.5 节的图 11。

```
w = Window[];
c = Canvas[w, Width -> 600, Height -> 100];
bitmap = {"error", "gray25", "gray50", "hourglass",
"info", "questhead", "question", "warning"};
x = 50;
```

```
Scan[(
c[Create$Bitmap]={x, 40, Bitmap -> #};
c[Create$Text]={x, 70, Text -> "\""//#//"\""};
x += 70)&,
bitmap];
```

表 44: Bitmap 的属性

属性	值	缺省值	效果
Anchor	“c” “n” “ne” “e” “se” “s” “sw” “w” “nw”	“c”	座标系上的相对位置
Background	颜色	无指定	背景色
Bitmap	位图		位图（字符串或是“@文件名”）
Foreground	颜色	“black”	前景色
Tags	字符串		加在其他项目上的 Tag 群

15.1.6 Image

表 45: Image 的属性

属性	值	缺省值	效果
Anchor	“c” “n” “ne” “e” “se” “s” “sw” “w” “nw”	“c”	座标系上的相对位置
Image	Image		Image
Tags	字符串		加在其他项目上的 Tag 群

15.1.7 Line

Line 描绘折线。

Create&Line 的参数是把座标按照 $x_1, y_1, x_2, y_2, \cdots\cdots$的顺序排列。属性 Smooth 为 True 时各点由スプライン曲线连结，False 时以直线连结。属性 CapStyle 和 JoinStyle 制定线分的端部和接合部的形状。（参照图 46）Line 的属性揭示在表 46。另外，下一个例子制作图 46。

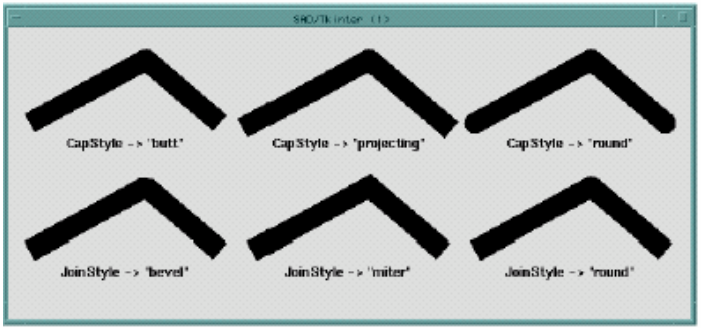

图 46：Line 的 CapStyle 和 JoinStyle 属性。

```
w = Window[];
c = Canvas[w, Width -> 640, Height -> 270];
options = {
CapStyle -> "butt",
CapStyle -> "projecting",
CapStyle -> "round",
JoinStyle -> "bevel",
JoinStyle -> "miter",
JoinStyle -> "round"};
{x, y} = {20, 90};
Scan[(
c[Create$Line] = {x, y, x + 110, y - 60, x + 180, y,
Width -> 20, #};
c[Create$Text] = {x + 90, y + 20,
Text -> #[[1]]//" -> \""//#[[2]]//"\""};
x += 210;
If[x > 500, x = 20; y += 120])&,
options];
```

表 46：Line 的属性

属性	值	缺省值	效果
Arrow	“none” “first” “last” “both”	“none”	箭头的有无

ArrowShape			箭头的形状
CapStyle	“butt” “projecting” “round”	“butt”	末端的形状
Fill	颜色	无指定	线条颜色
JoinStyle	“bevel” “miter” “round”	“round”	结合处的形状
Smooth	True False	False	True:花键，False:折线
SplineSteps	数值		花键的线条数
Stipple	位图		全面涂抹后的位图
Tags	字符串		加在其他项目上的 Tag 群
Width	数值	1	线的宽度

15.1.8 Oval

Oval 描绘椭圆。Create&Oval 是把其椭圆的外结长方形的两个对角顶点的座标作为最初的四个参数。

表 47: Oval 的属性

属性	值	缺省值	效果
Fill	颜色	无指定	内部颜色
Outline	颜色	“black”	轮廓线的颜色
Stipple	位图		全面涂抹后的位图
Tags	字符串		加在其他项目上的 Tag 群
Width	数值	1	轮廓线的宽度

15.1.9 Polygon

Polygon 描绘封闭的多边形。但无外形线，而把全体作为一体对待。

Create&Polygon 的参数是把顶点座标按照 $x_1, y_1, x_2, y_2, \cdots\cdots$的顺序排列。属性 Smooth 为 True 时各点由スプライン曲线连结，False 时以直线连结。表 48 揭示其属性，下一例做成图 47。

```
w = Window[];
```

```
c = Canvas[w, Width -> 600, Height -> 160];
param := {x, y, x + 40, y - 40,
x + 80, y, x + 120, y + 40,
x + 160, y, x + 120, y - 40,
x + 80, y, x + 40, y + 40};
{x, y} = {20, 60};
c[Create$Polygon] = {param, Smooth -> False};
c[Create$Text] = {x + 80, y + 60,
Text -> "Smooth -> False"};
{x, y} = {200, 60};
c[Create$Polygon] = {param, Smooth -> True};
c[Create$Text] = {x + 80, y + 60,
Text -> "Smooth -> True"};
{x, y} = {380, 60};
c[Create$Polygon] = {param, Smooth -> True,
SplineSteps -> 3};
c[Create$Text] = {x + 80, y + 60,
Text -> "Smooth -> True"};
c[Create$Text] = {x + 80, y + 76,
Text -> "SplineSteps -> 3"};
```

表 48: Polygon 的属性

属性	值	缺省值	效果
Fill	颜色	“black”	整体色
Smooth	True False	False	True:花键，False:折线
SplineSteps	数值		花键的线条数
Stipple	位图		全面涂抹后的位图
Tags	字符串		加在其他项目上的 Tag 群

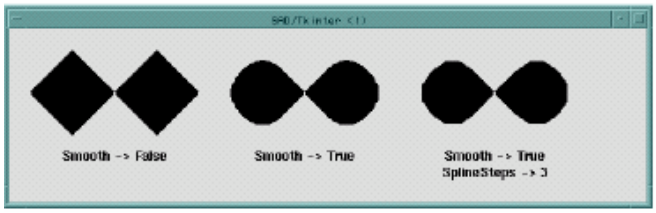

图 47：Polygon 的例子。通过 Smooth一>True，要使用スプライン。

15.1.10 Rectangle

Rectangle 是无倾斜的长方形。Create& Rectangle 是把 其两个对角顶点的座标 作为参数。

表 49: Rectangle 的属性

属性	值	缺省值	效果
Fill	颜色	无指定	内部颜色
Outline	颜色	“black”	轮廓线的颜色
Stipple	位图		加在其他项目上的 Tag 群
Tags	字符串		全面涂抹后的位图
Width	数值	1	轮廓线的宽度

15.1.11 Text

Text 表示字符串。另外，也可像 Entry 部品那样将其编辑，在这里不多说明了。Create&Text 把用于指定其位置的一组座标作为参数。还有，其座标和 Text 的相对位置由属性 Anchor 和 Justify 指定。Text 的属性揭示在表 50。

表 50: Text 的属性

属性	值	缺省值	效果
Anchor	“c” “n” “ne” “e” “se” “s” “sw” “w” “nw”	“c”	座标系上的相对位置
Fill	颜色	“black”	文字的颜色
Font	字型		字型
Justify	“left” “right” “center”	“left”	文字的方向
Stipple	位图		全面涂抹后的位图

Tags	字符串		加在其他项目上的 Tag 群
Text	字符串	“”	表示字符串

15.1.12 Window

Window 是用于在 Canvas 上粘贴其他任意的 Tkinter 的部品。Create&Window 把用于指定粘贴部品位置的一组座标作为参数，使用属性 Window 指定部品。例如若如下输入，就可以得到图 48 的结果。

```
w = Window[];
c = Canvas[w, Height -> 300, Width -> 400];
$DisplayFunction = CanvasDrawer;
Canvas$Widget = c;
Plot[Sin[x], {x, -Pi, Pi}];
b = Button[c, Text -> "Button on Canvas"];
c[Create$Window] = {150, 100, Window -> b};
```

以我使用到现在的情况来看，它似乎还不能把贴上的 Frame 等变成透明。

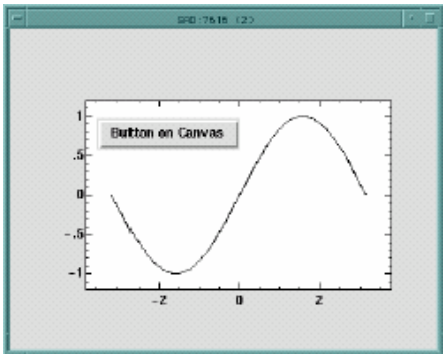

图 48：Canvas 中的 Window 的例子。

15.2 item 的操作

对于 Canvas 上制作的 item 在后来还可以有各种各样的操作。以下说明中的 c 是用于表示某个 Canvas 部品的。

15.2.1 属性的变更以及调查，ItemConfigure

- c[ItemConfigure]= {**tag** 或是 item 序号，属性一>值，……}是用于变更以 item 序号指

定的 item 或是以 **tag** 指定的 item 群的属性。另外，item 的[上下]是把此刻的表示的重叠情况是前面出现的称为上，**虽然越是后来制作的 item 越会到上边来，但也可以变更。**

- c[ItemConfigure[item 序号，属性]]将返回用 item 序号指定的 item 的属性设定值。

15.2.2 标记的附加，AddTag

- c[AddTag$Above]={tag 1，tag2 或是 item 序号 2}对于以 item 序号 2 指定的 item 或是以 tag2 指定的 item 中最前面的 item 的**直上的 item** 附加 tag1。
- c[AddTag$All]＝tag 1 是对迄今为止制作的所有 item 群附加 tag 1。
- c[AddTag$Below]={tag 1,tag2 或是 item 序号 2}对于以 item 序号 2 指定的 item 或是以 tag2 指定的 item 中最后面的 item 的**直下的 item** 附加 tag1。
- c[AddTag$Closest]＝{tag 1,x,y}对距离位置（x，y）最近的 item 群中最前面的 item 附加 tag 1。
- c[AddTag$Enclosed]＝{tag 1,x1,y1,x2,y2}对于完全包含于指定的长方形内的 item 群附加 tag1。
- c[AddTag$WithTag]＝{tag 1,tag2 或是 item 序号 2}对于以 item 序号 2 指定的 item 或是以 tag2 指定的 item 群附加 tag1。

15.2.3 item 的移动，Move

- c[Move]={tag 或是 item 序号，Δx，Δy}只移动由 item 序号指定的 item 或是由 tag 指定的 item 群（Δx，Δy）个单位。

15.2.4 位置的变更以及调查，Coords

- c[Coords]={item 序号，x1，y1，……}把由 item 序号指定的 item 的位置变更为 x1，y1，……。
- c[Coords[item 序号]]返回由 item 序号指定的 item 的位置。

15.2.5 item 的消去，Delete

- c[Delete]={tag 或是 item 序号 1，……}是用于消去由 tag1 或是 item 序号 1，……指定的 item 群。

15.2.6 item 的表示面的重叠的移动

c[Lower]=tag 或是 item 序号 是把用 item 序号指定的 item 或是用 tag 指定的 item 群移动到表示的最后面。

- c[Lower]={tag 1 或是 item 序号 1,tag2 或是 item 序号 2}是把用 item 序号 1 指定的 item 或是用 tag1 指定的 item 群，移动到 tag2 或是 item 序号 2 的秒的直下的表示面。
- c[Raise]=tag 或是 item 序号 是把用 item 序号指定的 item 或是用 tag 指定的 item 群移动到表示的最前面。
- c[Raise]={tag 1 或是 item 序号 1,tag2 或是 item 序号 2}是把用 item 序号 1 指定的 item 或是用 tag1 指定的 item 群，移动到 tag2 或 item 序号 2 的 item 的直上的表示面。

15.2.7 tag 的解除，Dtag

- c[Dtag]=tag 1 是把对 tag 1 的 item 设置命令解除。
- c[Dtag]＝{tag 1,tag2}是把对于用 tag1 指定的 item 群的 tag2 设置命令解除。

15.2.8 tag 的调查，GetTags

- c[GetTags[tag 或是 item 序号]]是把附加在用 item 序号指定的 item 上的或是用 tag 指定的 item 中最初的 item 上的 tag 群变成 list 返回。

15.3 对 item 的 event 结合

Canvas 的各 item，或是带有 tag 的各组可以通过 Bind，各自独立与 event 结合。其方法是例如：

像这样只是对 Bind 加上 option Tags->（tag 或是 item 序号，又或者是它们的 list），接下来的就是和对于一般的部品的结合（参照 4.2）完全一样了。另外，对变量$Event 发生 event 的 item 的序号或是 tag 会被返回。

15.4 用 plot 函数制作的图表上的 item 的操作

15.4.1 与 Canvas$ID 有关的 item 序号的记录

ListPlot 和 Plot 等的 plot 函数也用 Canvas 的 item 的组合来表现图表。从而，如果知道他们的 item 序号，就可以对各个 item 添加上述的各种操作。ListPlot 制作的 item 的序号通过 option Tags->True，每次都被记录在标识符 Canvas$ID。例如：

```
w = Window[];
c = Canvas[w, Width -> 600, Height -> 400];
$DisplayFunction = CanvasDrawer;
Canvas$Widget = c;
data = {{1, 2, 0.5}, {2, 3, 2},
```

{3, 5, 0.5}, {4, 2, 0.5}, {5, -2, 0.5}};

ListPlot[data, Tags -> True,

Plot -> True, PlotJoined -> True,

PointSize -> 2];

Print[Canvas$ID];

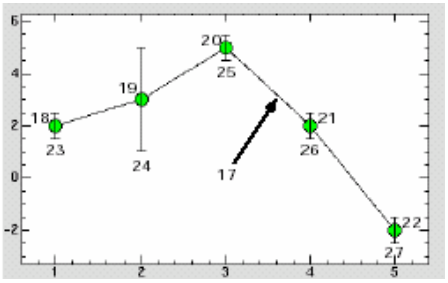

图 49：用 ListPlot plot 了的图表的 item 序号。通过 Tags->True，就会在 Canvas$ID 上记录 item 序号。这时，图表的线是序号 17，数据点的标记是序号 18－22。

15.4.2 通过 Bind 的对 item 的 event 结合

用以上的方法已弄清各 item 的 item 序号，这次用它对各 item 绑定 event。基本上这是对在 15.3 说明的功能的应用。首先，对上例：

ld = Length[data];

markers = Canvas$ID[[1, 1]] + Range[ld];

Bind[c, "<Button-1>", cmd, Tags -> markers];

这样，对于 markers 就会有 marker 群的 item 序号作为 list 进入。此时就变成 {18,19,20,21,22}.其后，通过用 Bind 制定成 Tags－>markers，各 marker 每当事件“<Button-1>”(鼠标按钮 1 的按下)发生时都会运算并实行表达式 cmd。例如：

(switch[#] = True)& /@ markers; (1)

markcolor[True] = "green"; (2)

markcolor[False] = "red"; (3)

errorbarcolor[True] = "black"; (4)

errorbarcolor[False] = "gray"; (5)

cmd := Module[{id = ToExpression[Tag /. $Event]}, (6)

switch[id] = ~switch[id]; (7)

```
c[ItemConfigure] = {id,
Fill -> markcolor[switch[id]]};                    (8)
c[ItemConfigure] = {id + ld,
Fill -> errorbarcolor[switch[id]]}];               (9)
```

像这样的每当点击各 marker 就会出现如图 50 所示，marker 和 error bar 的颜色会肘接（弯头连接？）

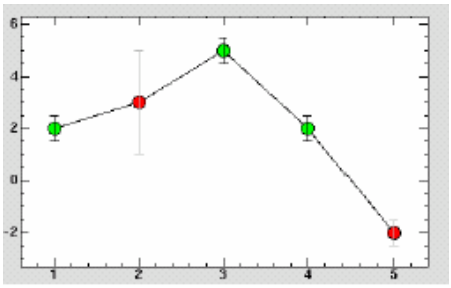

图 50：点击各 marker，marker 和 error bar 的颜色就会肘接（弯头连接？）

在此首先，（1）是将标识符 switch 定义为保持 marker 的状态的函数。（2）到（5）是对与各状态对应的颜色的定义。（6）到（9）是对标识符 cmd 的定义。（6）是通过对局部标识符 id 使用了标识符 Tag 的$Event 的置换，而取出被点击的 item 的 item 序号。$Event 此时是这种规则的 list。

```
{(Widget:>c),(Tag->"20"),(Type->"<Button>"),(X->291),(Y->124),(XRoot->693),
(YRoot->542),(Height->0),(Width->0),(Char->"??"),(KeySym->"??"),
(SendEvent->0),(KeyCode->1),(State->0),(KeySymNum->1),(Time->6784003)}
```

$Event 为大范围的标识符，但事件一发生，其结合的命令每当被调用时，其值都会被局部地设定。（参照 8.8 节）有关于包含在$Event 的标识符请参照表 2。

（7）是 switch 的弯头。（8）、（9）将把序号 id 的 marker 以及序号 id＋id 的 error bar 的颜色重新设置。这样，将 item 和事件结合的基本的动作就确认完毕了。

16. 构件的合成

SAD/Tkinter 可以让使用者把迄今为止讲解的系统与附备的构件组装起来，将新的构件定义后，和附备的构件一样地使用。

16.1 LabeledEntry

现在举个简单的例子，我们来定义一下把 TextLabel 和 Entry 组合安装的 LebeledEntry 这个构件。此构件只是在某个 Frame 中的左侧放 TextLabel，右侧放 Entry 而已。一种方法如下：

```
(a_ = LabeledEntry[w_, opt___]) ^:= (                    (1)
DeleteWidget[a];                                         (2)
Clear[a];                                                (3)
a[Frame] = Frame[w, opt];                                (4)
a[Label] = TextLabel[a[Frame], Side -> "left", opt];     (5)
a[Entry] = Entry[a[Frame], Side -> "left", opt];         (6)
a[b_] := (
a[Frame][b];                                             (7)
a[Label][b];                                             (8)
a[Entry][b]);                                            (9)
(a[b_] = c_) ^:= (
a[Frame][b] = c;                                         (10)
a[Label][b] = c;                                         (11)
a[Entry][b] = c);                                        (12)
(a[b_] := c_) ^:= (
a[Frame][b] := c;                                        (13)
a[Label][b] := c;                                        (14)
a[Entry][b] := c);                                       (15)
);
```

在这里，首先（1）是为了把 LabeledEntry 这个新构件能够像普通构件一样以如下的形式定义使用了上方值（参照 8.6）

Symbol=LabeleEntry[亲，属性,…]

在此之前标识符 a 可能被设定的一些构件和值，由(2)(3)可以解除。

LabeledEntry 这个构件是由 Frame、TextLabel、Entry 这三个从属构件组成的。这些从属构件的定义是（4）～（6）。左边的形状是不是很贴切地表达了它们是从属构件的意思呢。在这里，选项 opt 赋予这三个从属构件以共同性。也就是说，这三种都会被指定同样的属性。例如：Background 属性对这三个是共通的，所以对于哪个都适用。但比如 Text 属性只在 TextLabel 里有，只适用于 TextLabel，而其他的构件就会无视它的存在。另外，Side 属性如

（5）（6）所示，在 TextLabel 和 Entry 中，先写的 Side—>“left”被优先考虑，所以只是对 Frame 有效。如果想个别地设置属性，就可以按如下例子操作：

Symbol[Entry][属性]=值

（7）～（15）是为了把对 LabeledEntry 整体的共通操作，属性的变更与普通的构件一样地进行而进行定义。使用 LabeledEntry，写出以下语句就可以作出如图 51 所示的面板。

图 51：构件 LabeledEntry

```
w = Window[];
le1 = LabeledEntry[w,
Text -> "username: ",Side->"top"];
le2 = LabeledEntry[w,
Text -> "password: ",Side->"top", ShowText -> "*"];
```

图 51：合成的构件 LabeledEntry 的例子。

此外，使用 DeleteWidget（参照 4.7）可以消除用 LabeledEntry 作成的构件整体。

如上所述，LabeledEntry 在这样的简单情况下基本上可以出色完成构件的合成。到底最复杂的能到什么程度对于作者也是个未知数，还是请大家来挑战一下吧。

17. 例题

17.1 SimpleDialog

下面所示的 SimpleDialog，是在众多程序中广泛使用的，与操作者进行单纯的应答的面板。

首先，其定义如下：

```
SimpleDialog[message_, texts_]:=Module[
{w, tl, fr, i, b, l = Length[texts], r},                    (1)
w = Window[];
tl = TextLabel[w, Text -> message,
PadX -> 20, PadY -> 10];                                    (2)
fr = Frame[w];                                              (3)
```

```
i = 1;
Scan[(                                                    (4)
With[{m = #},                                             (5)
b[i] = Button[fr, Text -> m,                              (6)
Command :> TkReturn[m],                                   (7)
BD -> 4, PadX -> 10, PadY -> 10,
Side -> "left", Relief -> "ridge"]];
i++)&,
texts];
b[1][Relief] = "raised";                                  (8)
Bind[b[1], "<Key-Return>",                                (9)
TkReturn[texts[1]]];
b[1][Focus$Set];                                          (10)
r = TkWait[];                                             (11)
w =.;                                                     (12)
Update[];                                                 (13)
r];                                                       (14)
```

使用方法如下：

```
Symbol=SimpleDialog[
```

在这里，message 是字符串，button list 是写在 button 表面的字符串的 list。例如：

```
result = SimpleDialog[
"Do you want to save?",
{"Cancel", "Don't save", "Save"}];
```

像这样，就会出现如图 52 所示的 Window。在此点击任一个 button，就会返回其名称，并设定给标识符 result。另外，最后一个 button（此例中为“Save”）会作为 default，（9）（10）使它即使按 Return 键也会和点击是同样的结果。

图 52：SimpleDialog 的例子。

SimpleDialog 是利用了 Module 的函数，制作的窗口和内部的构件都设定给由（1）定义的局部标识符，实行后由（16）（17）全部消去。

可制成的 button 的数量不限。在内部由（4）的 Scan 把 button 设置成 b[i]。（5）中虽然使用了 With，但它是为了能把（7）的命令中 TkReturn[m]的 m 的值预先设定。通过 With，m 为值＃，即 texts 的各要素。如果把这里写作 Command ：>TkReturn[#],因为：>右边不会被运算，＃按原样保留，后来即使作为 TkReturn[#]，#也会变成未定义的。

（11）通过 TkWait[]等待应答。SimpleDialog 本身作为其他的 button 的应答命令而使用的时候，TkWait[]会重叠几层，但是并不要紧。

17.2 让图表的大小缩放至窗口的大小

实际应用当中，使用者经常会要根据需要而改变画图的窗口大小。下面就是说明在这种情况下，将窗口里画的图表根据窗口的大小再次描画的例子。

```
FFS;
$DisplayFunction = CanvasDrawer;
w = Window[];
c1 = Canvas[w, Height -> 100, Width -> 150,
Side -> "left",
Expand -> True, Fill -> "both"];                    (1)
c2 = Canvas[w, Height -> 100, Width -> 150,
Side -> "left",
Expand -> True, Fill -> "both"];                    (2)
pl:=(
Canvas$Widget = c1;
Plot[Sin[x],{x,-Pi,Pi}];
Canvas$Widget = c2;
Plot[Cos[x],{x,-Pi,Pi}]);                           (3)
pl;
Bind[w, "<Configure>", pl];                         (4)
TkWait[];
```

此例是用（3）的命令 pl 使（1）（2）作成的左右排列的 Canvas c1、c2 输出图像。这里 Canvas 的属性指定为 Expand->True , Fill->“both”，可以保证 Canvas 将追随窗口的伸缩而随时变化。（4）中通过把命令 pl 和窗口属性以及伴随它们变更的事件“<Configure>”结合到一起，可在每当窗口伸缩的时候使其再描画并追从图表的大小。(但因此时各 Canvas 要把最初指定的尺寸保持为最小尺寸，如果尺寸过小，两个图表的大小就不均等了。)

此例如图 53 所示。

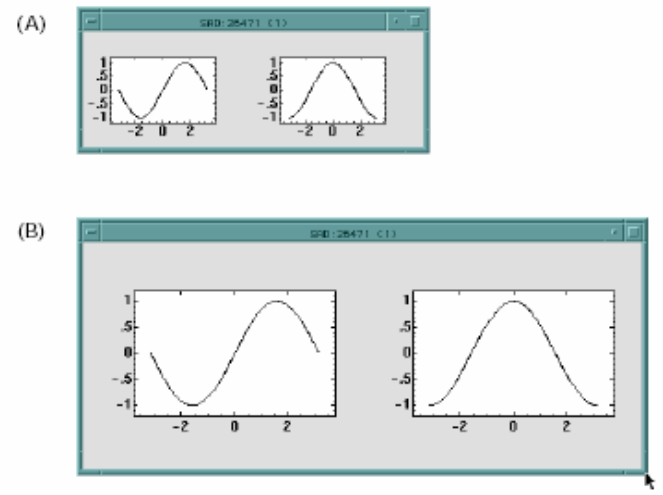

图 53：事件通过“<Configure>”让图表的大小追从于窗口尺寸的变更。（A）初期状态。（B）用鼠标扩大窗口的例子。

17.3 图表的缩放

下面是在用 Plot 书写的图表上把 drag 了的领域扩大后再次 plot，是一个能够实现司空见惯的功能的例子。

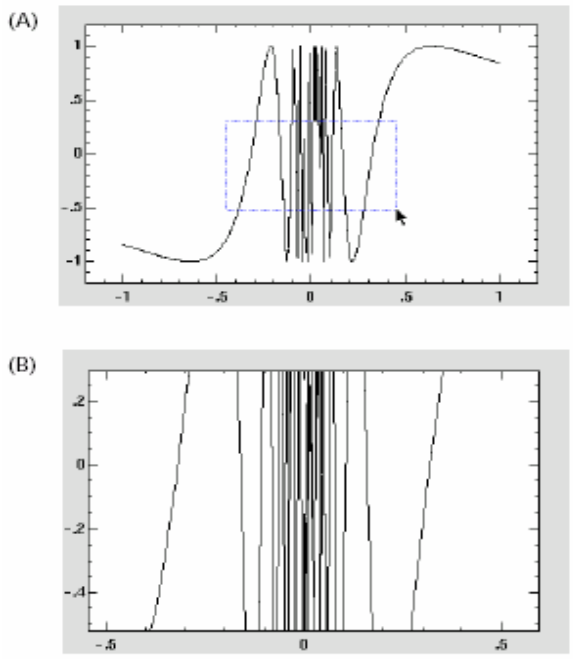

图 54：（A）用鼠标拖出一个范围（B）扩大表示。

```
FFS;
StartSelect := Module[                                          (1)
{{x, y, c} = {X, Y, Widget}/.$Event},
With[{c},
c[Create$Line]={x, y, x, y,
Fill -> "blue", Stipple -> "gray50",
165
Tags -> "selection"}];
Bind[c, "<Motion>", DragSelect];                                (2)
Bind[c, "<ButtonRelease-1>", RedrawSelect];                     (3)
selstart=selend={x,y}];                                         (4)
DragSelect := Module[                                           (5)
{{x, y, c} = {X, Y, Widget}/.$Event},
With[{c, x0 = selstart[[1]], y0 = selstart[[2]]},
c[Delete] = "selection";
c[Create$Line] = {x0, y0, x, y0, x, y, x0, y, x0, y0,
```

```
Fill -> "blue", Stipple -> "gray50",
Tags -> "selection"};                                          (6)
selend = {x,y}]];
RedrawSelect := Module[                                        (7)
{c = Widget/.$Event},
c[Delete] = "selection";
If[selstart <> selend,
z0 = (selstart - Canvas$Offset) / Canvas$Scale;
z1 = (selend - Canvas$Offset) / Canvas$Scale;
pl[z0, z1]];
Bind[c, "<Motion>"];                                           (8)
Bind[c, "<ButtonRelease-1>"]];                                 (9)
pl[z0_,z1_] :=                                                 (10)
Plot[f[x],
{x, Min[z0[[1]], z1[[1]]], Max[z0[[1]], z1[[1]]]},
PlotRange ->
{Min[z0[[2]], z1[[2]]], Max[z0[[2]], z1[[2]]]}];
plinit := pl[{-1, -1.2}, {1, 1.2}];
w = Window[];
c = Canvas[w, Width -> 600, Height -> 400];
$DisplayFunction = CanvasDrawer;
Canvas$Widget = c;
f[0] = 0;
f[x_] := Sin[1/x];
plinit;
Bind[c, "<Button-1>", StartSelect];                            (11)
Bind[c, "<M-Button-1>", plinit];                               (12)
TkWait[];
```

此例首先在初期状态做一个如图 54（A）的 plot。如图所示一边按住鼠标 button 1 一边 drag 出长方形，放开 button 就如（B）所示，被选部分的 x、y 座标同时被扩大并再次 plot。

这个选择、扩大的动作由三个事件完成。首先按下 button，调用了（1）的 StartSelect。StartSelect 如（2）（3）所示，把鼠标游标的移动和 button 的解放与 DragSelect 和 RedrawSelect 结合起来。然后用（4）记录选择范围的起点。下一步 DragSelect 把此刻的选择范围用四边形重新描画。这个四边形会带上附签“Selection”。最后 RedrawSelect 确定选择范围，把 Canvas 座标换算成 Plot 的数据座标并再次 plot。另外，如（8）（9）所示，解除了对移动和 button 解放的两个事件的结合。这样最终得到了图 54（B）的扩大图。

此例还由（12）通过事件 button（メターボタン）把 plot 恢复成初期状态。

谢辞：

SAD/Tkinter 的开发得到了山本升（Python/Tkinter 的导入）、赤坂展昌（构件的开发）两个人的大力支持，同时还要深深感谢小机晴代对于此手册编集的协助。

译者后记：

由于译者对 EPICS 没有丰富的实际经验，本手册的译文必有不准确之处，敬请各位专家给予指正。其中，雷革编写第 1 章，陈梦翻译第 11 至第 17 章，薛鹏编辑图片和表格。雷革、薛鹏对内容给予校正，赵籍九研究员对全书进行了审核，特此表示感谢。

删除的内容：薛鹏编辑图片和表格，

译者：陈刚

2005－1－15

索引

Symbols

A

B

C

D

E

F

G

H

I

T

U

V

W

X

Y

Z