Oracle® Solaris Studio 12.4：Fortran 用户指南

文件号码 E57215
2014 年 12 月

ORACLE

目录

使用本文档

- **概述** – 介绍 Oracle Solaris Studio Fortran 编译器和环境
- **目标读者** – 应用程序开发者、系统开发者、架构师、支持工程师
- **必备知识** – 编程经验、软件开发测试以及构建和编译软件产品的能力

产品文档库

有关本产品的最新信息和已知问题均包含在文档库中，网址为：http://docs.oracle.com/cd/E37069_01。

获得 Oracle 支持

Oracle 客户可通过 My Oracle Support 获得电子支持。有关信息，请访问 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info；如果您听力受损，请访问 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs。

反馈

可以在 http://www.oracle.com/goto/docfeedback 上提供有关本文档的反馈。

♦ ♦ ♦ 第 1 章

1 介绍

本书以及配套的Fortran 编程指南中介绍的 Oracle Solaris Studio Fortran 编译器 *f95* 可在 SPARC®、UltraSPARC® 和 x64/x86 平台上的 Oracle Solaris 操作环境下以及 x86/x64 平台上的 Linux 环境下使用。此编译器符合发布的 Fortran 语言标准，并提供很多扩展的功能，其中包括多处理器并行化、高级的优化代码编译以及混合的 C/Fortran 语言支持。

f95 编译器还提供接受大多数传统 Fortran 77 源代码的 Fortran 77 兼容性模式。不再包含单独的 Fortran 77 编译器。有关 FORTRAN 77 兼容性和迁移问题的信息，请参见第 5 章。

1.1 标准符合性

- f95 符合 ISO/IEC 1539-1:1997 Fortran 标准文档的第一部分。
- 浮点运算基于 IEEE 标准 754-1985 和国际标准 IEC 60559:1989。
- 有关 Fortran 2003 和 Fortran 2008 功能以及此发行版中实现的其他语言扩展的列表，请参见第 4 章 Solaris Studio Fortran 的功能与扩展。
- f95 提供了对 SPARC 和 x86 系列处理器体系结构（Solaris 和 Linux (x86) 平台上的 UltraSPARC、SPARC64、AMD64、Pentium Pro 和 Xeon Intel®64）优化开发功能的支持。
- Oracle Solaris Studio 编译器支持 OpenMP API（用于实现共享内存并行化）版本 4.0。有关详细信息，请参见《OpenMP API 用户指南》。
- 在本文档中，“标准”是指与上面列出的标准版本相一致。“非标准”或“扩展”是指超出这些标准版本的功能。

负责标准的一方可能会不时地修订这些标准。这些编译器所遵循的适用标准的版本可能会被修订或替换，从而导致 Oracle Solaris Studio Fortran 编译器未来发行版中的功能与先前发行版不兼容。

1.2 Fortran 编译器的功能

Oracle Solaris Studio Fortran 编译器 f95 提供了以下功能和扩展：

- 在例程中对参数、公共区等进行全局程序一致性检查。（仅适用于 Solaris 平台）。
- 优化多处理器系统的自动和显式循环并行化。
- VAX/VMS Fortran 扩展。
- OpenMP 并行化指令。
- 全局、窥孔和潜在的并行化优化可产生高性能的应用程序。基准测试表明优化的应用程序的运行速度比未优化的代码快得多。
- 通用调用惯例允许将使用 C 或 C++ 编写的例程与 Fortran 程序结合使用。
- 支持启用的 64 位 Solaris 和 Linux 环境。
- 所有平台上都支持 IEEE 四精度 REAL 和 COMPLEX 类型。
- Fortran 77 和 Fortran 95/Fortran 2003 程序与对象二进制文件之间的兼容性。
- 区间运算编程。
- 支持无符号整数类型。
- 许多 Fortran 2003 和 Fortran 2008 功能。

有关每个软件发行版的编译器中添加的新功能和扩展功能的详细信息，请参见附录 B。

另请参见《*Oracle Solaris Studio 12.4* 发行版的新增功能》，以了解有关本编译器和工具发行版中新增和更改的功能、已知问题、解决方法和限制的最新信息。可以通过 http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation 上的本发行版文档索引页访问该“新增功能”指南。

1.3 其他 Fortran 实用程序

以下实用程序可为使用 Fortran 进行软件程序开发提供帮助：

- Solaris Studio 性能分析器 – 单线程和多线程应用程序的性能深入分析工具。请参见 `analyzer`(1)。
- `asa` – 此 Solaris 实用程序是一个 Fortran 输出过滤器，用于打印第一列中包含 Fortran 回车控制符的文件。可使用 `asa` 将按照 Fortran 回车控制惯例设置格式的文件转换为按照 UNIX 行打印机惯例设置格式的文件。请参见 `asa`(1)。
- `fdumpmod` – 显示包含在文件或归档文件中的模块名称的实用程序。请参见 `fdumpmod`(1)。
- `fpp` – Fortran 源代码预处理程序。请参见 `fpp`(1)。
- `fsplit` – 此实用程序将一个包含几个例程的 Fortran 文件拆分成几个文件，每个文件包含一个例程。请参见 `fsplit`(1)。

1.4 调试实用程序

可以使用以下调试实用程序：

- `-Xlist` – 一个用于检查例程中参数、COMMON 块等一致性的编译器选项。（仅限 Solaris 平台）
- Solaris Studio `dbx` – 提供强大、功能丰富的运行时和静态调试器，包含一个性能数据收集器。

1.5 Sun 性能库

Sun 性能库™是一个用于计算线性代数和傅立叶变换的优化子例程及函数的库。它基于一般通过 Netlib 提供的标准库 LAPACK、BLAS1、BLAS2、BLAS3、FFTPACK、VFFTPACK 和 LINPACK。

与标准库版本相比，Sun 性能库中的每个子程序执行相同的操作并且具有相同的接口，但通常这些子程序的速度要快得多且准确得多，这些子程序可以用于多处理环境中。

有关详细信息，请参见《*Sun Performance Library User's Guide*》。（性能库例程的手册页位于第 3P 节。）

1.6 区间运算

`f95` 编译器提供编译器标志 `-xia` 和 `-xinterval` 以启用新的语言扩展，并生成相应的代码以执行区间运算。有关详细信息，请参见《*Fortran 95* 区间运算编程指南》。

1.7 手册页

联机手册 (`man`) 页提供了关于命令、函数、子例程或这些项的集合的当前文档。要访问 Oracle Solaris Studio 的手册页，应将用户的 `MANPATH` 环境变量设置为指向已安装的 Solaris Studio 的 `man` 目录的路径。

可以通过运行以下命令来显示手册页：

```
demo% man topic
```

在整个 Fortran 文档中，出现的手册页参考带有主题名称和手册章节号： 可使用 `man f95` 访问 `f95`(1)。例如，可在 `man` 命令中使用 `-s` 选项来访问 `ieee_flags`(3M) 指示的其他部分：

```
demo% man -s 3M ieee_flags
```

Fortran 库例程是在手册页第 3F 节中介绍的。

下面列出了对于 Fortran 用户非常重要的手册页：

f95(1)	Fortran 95 命令行选项
analyzer(1)	Solaris Studio 性能分析器
asa(1)	Fortran 回车控制打印输出后处理器
dbx(1)	命令行交互调试器
fpp(1)	Fortran 源代码预处理器
cpp(1)	C 源代码预处理器
fdumpmod(1)	显示模块 (.mod) 文件的内容
fsplit(1)	预处理器将 Fortran 源例程分成单个文件
ieee_flags(3M)	检查、设置或清除浮点异常位
ieee_handler(3M)	处理浮点异常
matherr(3M)	数学库错误处理例程
ld(1)	对象文件的链接编辑器

1.8 命令行帮助

可通过调用编译器的 -help 选项来查看 f95 命令行选项的简短描述（如下所示）：

```
%f95 -help=flags
Items within [ ] are optional. Items within < > are variable parameters.
Bar | indicates choice of literal values.
-someoption[={yes|no}] implies -someoption is equivalent to -someoption=yes

_____________________________________________________________
-a                 Collect data for tcov basic block profiling
-aligncommon[=<a>] Align common block elements to the specified
                   boundary requirement; <a>={1|2|4|8|16}
-ansi              Report non-ANSI extensions.
-autopar           Enable automatic loop parallelization
-Bdynamic          Allow dynamic linking
-Bstatic           Require static linking
-C                 Enable runtime subscript range checking
-c                 Compile only; produce .o files but suppress
                   linking
...etc.
```

♦ ♦ ♦ 第 2 章

2 使用 Solaris Studio Fortran

本章介绍如何使用 Fortran 编译器。

所有编译器的主要用途都是将使用过程语言（如 Fortran）编写的程序转换为可由目标计算机硬件执行的数据文件。在此过程中，编译器也可以自动调用系统链接程序来生成可执行文件。

编译器还可以用于：

- 生成并行的可执行文件以便由多线程执行 (`-openmp`)。
- 跨源文件和子例程分析程序的一致性并生成报告 (`-Xlist`)。
- 将源文件转换为：
  - 可重定位的二进制 (`.o`) 文件，可随后将其链接到可执行文件或静态库 (`.a`) 文件。
  - 动态共享库 (`.so`) 文件 (`-G`)。

  将文件链接到可执行文件。
- 在运行时调试处于启用状态的情况下编译可执行文件 (`-g`)。
- 使用运行时语句或过程级分析进行编译 (`-pg`)。
- 检查源代码是否符合 ANSI 标准 (`-ansi`)。

2.1 快速入门

本节简要介绍如何使用 Fortran 编译器来编译和运行 Fortran 程序。下一章将详细介绍命令行选项。

运行 Fortran 应用程序的基本步骤包括：使用编辑器创建带有文件名后缀 `.f`、`.for`、`.f90`、`.f95`、`.F`、`.F90`、`.F95`、`.f03` 或 `.F03` 的 Fortran 源文件（请参见表 2-1 “由 Fortran 编译器识别的文件名后缀”），调用编译器来生成可执行文件；最后通过键入文件名来启动该程序的执行：

示例：此程序在屏幕上显示一条消息：

```
demo% cat greetings.f
```

```
      PROGRAM GREETINGS
      PRINT *, 'Real programmers write Fortran!'
      END
demo% f95 greetings.f
demo% a.out
 Real programmers write Fortran!
demo%
```

在此示例中，f95 编译源文件 greetings.f，并且在缺省情况下将可执行程序链接到文件 a.out。要启动该程序，请在命令提示符下键入可执行文件的名称 a.out。

传统上，UNIX 编译器将可执行输出写入到名为 a.out 的缺省文件中。每次编译都写入到同一个文件是比较笨拙的方法。再者，如果该文件已存在，则在下次运行编译器时它将被覆盖。因此，请使用 -o 编译器选项来显式指定可执行输出文件的名称：

```
demo% f95 -o greetings greetings.f
demo% greetings
 Real programmers write Fortran!
demo%
```

在上面的示例中，-o 选项告知编译器将可执行代码写入到文件 greetings 中。（按照约定，通常会为可执行文件指定与主源文件相同的名称，但可执行文件没有扩展名。）

或者，可在每次编译后通过 mv 命令重命名缺省的 a.out 文件。无论使用哪种方法，都要在 shell 提示符下键入可执行文件的名称来运行程序。

本章的后面几节将讨论 f95 命令使用的约定、编译器源代码行指令，以及有关使用这些编译器的其他问题。下一章将详细介绍命令行语法和所有选项。

2.2 调用编译器

在 shell 提示符下调用的编译器有一个简单的命令，其语法是：

f95 [*options*] *files...*

其中，*files...* 是一个或多个以 .f、.F、.f90、.f95、.F90、.F95 或 .for 结尾的 Fortran 源文件名称；*options* 是一个或多个编译器选项标志。（以 .f90 或 .f95 扩展名结尾的文件是只能由 f95 编译器识别的“自由格式”Fortran 95 源文件。）

在下面的示例中，我们在启用运行时调试的情况下，使用 f95 来编译两个源文件以生成名为 growth 的可执行文件：

```
demo% f95 -g -o growth growth.f fft.f95
```

注 - 可以使用 f95 或 f90 命令来调用 Fortran 编译器。

新增功能：该编译器还接受扩展名为 `.f03` 或 `.F03` 的源文件。这些文件将被视为与 `.f95` 和 `.F95` 等效，并且可以作为一种方式来表示源文件包含 Fortran 2003 扩展名。

第 2.2.2 节 “命令行文件命名约定” [21]说明了编译器可以识别的各种源文件扩展名。

2.2.1 编译和链接序列

在上一示例中，编译器自动生成加载器对象文件 `growth.o` 和 `fft.o`，然后调用系统链接程序以创建可执行程序文件 `growth`。

在编译后，对象文件 `growth.o` 和 `fft.o` 将保留。此约定使您可以方便地重新链接和重新编译文件。

如果编译失败，您将收到每个错误的对应消息。对于出现错误的源文件，不会生成任何 `.o` 文件，也不会写入任何可执行程序文件。

2.2.2 命令行文件命名约定

在命令行上出现的文件名后附加的后缀扩展名决定了编译器处理文件的方式。如果文件名的后缀扩展名不是下面列出的任意一个扩展名，或者没有扩展名，则这些文件名将传递给链接程序。

表 2-1 由 Fortran 编译器识别的文件名后缀

后缀	语言	操作
`.f`	Fortran 77 或 Fortran 95 固定格式	编译 Fortran 源文件，将对象文件放在当前目录中；对象文件的缺省名称是源文件的名称，但具有 `.o` 后缀。
`.f95` `.f90`	Fortran 95 自由格式	执行与 `.f` 相同的操作
`.f03`	Fortran 2003 自由格式	执行与 `.f` 相同的操作
`.for`	Fortran 77 或 Fortran 95 固定格式	执行与 `.f` 相同的操作。
`.F`	Fortran 77 或 Fortran 95 固定格式	在编译前，将 Fortran（或 C）预处理程序应用于 Fortran 77 源文件。
`.F95` `.F90`	Fortran 95 自由格式	在 Fortran 编译 Fortran 95 自由格式源文件前，将 Fortran（或 C）预处理程序应用于该文件。
`.F03`	Fortran 2003 自由格式	与 `.F95` 相同。
`.s`	汇编程序	使用汇编程序汇编源文件。

后缀	语言	操作
.S	汇编程序	在对汇编程序源文件进行汇编之前，将 C 预处理程序应用于该文件。
.il	内联扩展	处理内联扩展的模板文件。编译器将使用模板来扩展选定例程的内联调用。（模板文件是特殊的汇编程序文件；请参见 inline(1) 手册页。）
.o	对象文件	将对象文件传递到链接程序。
.a、.so、.so.*n*	库	将库名称传递给链接程序。.a 文件是静态库，.so 和 .so.*n* 文件是动态库。

Fortran 95 自由格式在第 4.1 节 “源语言功能” [147]中进行说明。

2.2.3 源文件

Fortran 编译器可从命令行接受多个源文件。单个源文件（也称为编译单元）可以包含任意数量的过程（主程序、子例程、函数、块数据、模块等）。可以将应用程序配置为每个文件一个源代码过程，或者将协同工作的过程集中到单个文件中。《*Fortran* 编程指南》介绍了这些配置的优缺点。

2.2.4 源文件预处理程序

f95 支持两种源文件预处理程序：fpp 和 cpp。编译器可以在编译之前调用任一源文件预处理程序来扩展源代码“宏”和符号定义。缺省情况下，编译器将使用 fpp；-xpp=cpp 选项可将缺省设置由 fpp 更改为 cpp。（另请参见有关 -D*name* 选项的论述。）

fpp 是 Fortran 特定的源文件预处理程序。有关详细信息，请参见 fpp(1) 手册页。缺省情况下，系统会对具有 .F、.F90、F95，或 .F03 扩展名的文件调用该预处理程序。

fpp 的源代码可从 Netlib 获得。

有关标准 Unix C 语言预处理程序的信息，请参见 cpp(1)。对于 Fortran 源文件，建议使用 fpp 而不是 cpp。

2.2.5 分别编译和链接

可以在不同的步骤中进行编译和链接。-c 选项编译源文件并生成 .o 对象文件，但不会创建可执行文件。如果不使用 -c 选项，则编译器将调用链接程序。如果通过这种方式将编译和链接步骤分开，那么就不必只为了修复一个文件而重新执行完整的编译，如以下示例所示：

使用单独的步骤来编译一个文件，并将其与其他文件链接在一起：

```
demo% f95 -c file1.f                    (Make new object file)
```

```
demo% f95 -o prgrm file1.o file2.o file3.o          (Make executable file)
```

请确保链接步骤列出了生成完整程序所需的全部对象文件。如果在此步骤中缺少任何对象文件，则链接将失败，并显示未定义的外部引用错误（缺少例程）。

2.2.6 一致编译和链接

每当分步完成编译和链接时，确保编译和链接选项的一致选择至关重要。在使用选项编译程序的任何部分时，必须使用相同的选项进行链接。另外，许多选项要求使用该选项编译所有源文件，包括链接步骤。

第 3 章中的选项描述指明了此类选项。

示例： 用 `-fast` 编译 `sbr.f`，编译 C 例程，然后分步进行链接：

```
demo% f95 -c -fast sbr.f
demo% cc -c -fast simm.c
demo% f95 -fast sbr.o simm.o          link step; passes  -fast  to the linker
```

2.2.7 无法识别的命令行参数

编译器无法识别的任何命令行参数都将解释为可能是链接程序选项、对象程序文件名或库名称。

基本区别是：

- 无法识别的选项（带有 `-`）会生成警告。
- 无法识别的非选项（不带 `-`）不生成警告。但是，这些非选项将被传递给链接程序，如果链接程序无法识别它们，它们将生成链接程序错误消息。

例如：

```
demo% f95 -bit move.f                <-  -bit is not a recognized f95 option
f95: Warning: Option -bit passed to ld, if ld is invoked, ignored otherwise
demo% f95 fast move.f                <-  The user meant to type -fast
ld: fatal: file fast: cannot open file; errno=2
ld: fatal: File processing errors.  No output written to a.out
```

请注意，在第一个示例中，`f95` 无法识别 `-bit`，该选项将被传递给链接程序 (`ld`)，后者试图对其进行解释。因为单字母 `ld` 选项可以串联起来，所以链接程序会将 `-bit` 视为 `-b -i -t`，而这些都是合法的 `ld` 选项！这可能是（也可能不是）用户所希望的结果。

在第二个示例中，用户本想键入 `f95` 选项 `-fast`，但忽略了前导短划线。编译器再次将参数传递给链接程序，而链接程序将参数解释为一个文件名。

这些示例表明在编写编译器命令行时应格外小心！

2.2.8 模块

`f95` 自动为在源文件中遇到的每个 `MODULE` 声明创建模块信息文件，并搜索 `USE` 语句所引用的模块。对于遇到的每个模块 (`MODULE` *module_name*)，编译器都在当前目录中生成相应的文件 *module_name*`.mod`。例如，`f95` 为文件 `mysrc.f95` 中出现的 `MODULE list` 单元生成模块信息文件 `list.mod`。

有关如何设置写入和搜索模块信息文件时所用的缺省路径的信息，请参见 `-M`*path* 和 `-moddir` *dirlist* 选项标志。

有关隐式调用所有编译单元中的 `MODULE` 声明的信息，另请参见 `-use` 编译器选项。

使用 `fdumpmod`(1) 命令可显示有关 `.mod` 模块信息文件内容的信息。

有关详细信息，请参见第 4.9 节 “模块文件” [168]。

2.3 指令

可使用源代码指令（一种 Fortran 注释格式）将有关专门的优化或并行化选择的特定信息传递给编译器。编译器指令有时也称为 *pragma*。编译器可识别一组常规指令和并行化指令，包括 OpenMP 指令。

特定于 `f95` 的指令在第 4.8 节 “指令” [166]中进行说明。附录 C, Fortran 指令摘要中完整总结了 `f95` 可以识别的所有指令。

注 - 指令并不是 Fortran 标准的一部分。

2.3.1 通用指令

Fortran 通用指令的各种形式包括：

`!$PRAGMA` *keyword* `(` *a* [, *a*] … `)` [, *keyword* `(` *a* [, *a*] … `)`] ,…

`!$PRAGMA SUN` *keyword* `(` *a* [, *a*] … `)` [, *keyword* (*a* [, *a*] … `)`] ,…

变量 *keyword* 标识特定的指令。此外也可以使用额外的参数或子选项。（某些指令要求使用额外的关键字 `SUN`，如上所示。）

通用指令使用以下语法：

- 在第一列中，使用以下任一注释指示符：c、C、! 或 *
- 对于 f95 自由格式，! 是唯一可识别的注释指示符 (!$PRAGMA)。本章中的示例采用 Fortran 95 自由格式。
- 后面七个字符为大写或小写的 $PRAGMA（字符之间没有空格）。
- 对于自由格式的源程序，使用 ! 注释指示符的指令可以出现在行中的任意位置。

请遵循以下约束：

- 与 Fortran 文本一样，在前八个字符之后，空格将被忽略，而且大写和小写字母等效。
- 由于它是注释，因此指令无法继续，但如果需要，您可以连续使用许多 !$PRAGMA 行。
- 如果注释符合以上语法，则它应该包含一个或多个编译器可识别的指令；如果不符合以上语法，系统会发出一条警告。
- C 预处理程序 cpp 将扩展注释行或指令行中的宏符号定义；Fortran 预处理程序 fpp 不扩展注释行中的宏。fpp 会识别合法的 f95 指令，并允许在指令关键字外部进行有限的替换。但是，在处理需要关键字 SUN 的指令时要格外小心。cpp 会将小写的 sun 替换为预定义的值。另外，如果您定义了一个 cpp 宏 SUN，则它可能会与 SUN 指令关键字发生冲突。一般规则是，如果源文件将由 cpp 或 fpp 来处理，那么应以混合大小写来拼写这些 pragma，如下所示：

  !$PRAGMA Sun UNROLL=3。

  编译 .F 文件时，添加 -Usun 可能也是一种解决方法。

Fortran 编译器可识别以下通用指令：

表 2-2 通用 Fortran 指令摘要

C 指令	!$PRAGMA C(*list*) 将一系列外部函数的名称声明为 C 语言例程。
IGNORE_TKR 指令	!$PRAGMA IGNORE_TKR {*name* {, *name*} ...} 在解析特定调用时，编译器会忽略在通用过程接口中出现的指定哑元参数名称的类型、种类和等级。
UNROLL 指令	!$PRAGMA SUN UNROLL=*n* 建议编译器将下面的循环解开为指定的长度 *n*。
WEAK 指令	!$PRAGMA WEAK(*name*[=*name2*]) 将 *name* 声明为弱符号，或者声明为 *name2* 的别名。
OPT 指令	!$PRAGMA SUN OPT=*n* 将子程序的优化级别设置为 *n*。
PIPELOOP 指令	!$PRAGMA SUN PIPELOOP=*n* 断言下面的循环中在间隔为 *n* 的迭代之间存在依赖性。

PREFETCH 指令	!$PRAGMA SUN_PREFETCH_READ_ONCE(*name*) !$PRAGMA SUN_PREFETCH_READ_MANY(*name*) !$PRAGMA SUN_PREFETCH_WRITE_ONCE(*name*) !$PRAGMA SUN_PREFETCH_WRITE_MANY(*name*) 请求编译器为名称引用生成预取指令。（需要使用 -xprefetch 选项，缺省情况下启用该选项。编译时使用 –xprefetch=no 可以禁用预取指令。目标体系结构也必须支持预取指令，而且编译器优化级别必须大于 –xO2。）
ASSUME 指令	!$PRAGMA [BEGIN} ASSUME (*expression* [,*probability*]) !$PRAGMA END ASSUME 断言编译器可假定程序中某些点处的条件为真。

2.3.1.1 C 指令

C() 指令指定其参数为外部函数。它与 EXTERNAL 声明等效，但有一点例外，与普通外部名称不同，Fortran 编译器在这些参数名称的后面不附加下划线。有关详细信息，请参见《 *Fortran* 编程指南》中的“C-Fortran 接口”一章。

在每个包含特定函数引用的子程序中，用于该函数的 C() 指令应该出现在对该函数的第一次引用之前。

示例 – 为 C 编译 ABC 和 XYZ：

```
      EXTERNAL ABC, XYZ
!$PRAGMA C(ABC, XYZ)
```

2.3.1.2 IGNORE_TKR 指令

此指令导致编译器在解析特定调用时，忽略在通用过程接口中出现的指定哑元参数名称的类型、种类和等级。

例如，在下面的过程接口中，指令指定 SRC 可以是任何数据类型，但 LEN 可以为 KIND=4 或 KIND=8。接口块为通用过程名称定义了两个特定过程。此示例以 Fortran 95 自由格式说明。

```
INTERFACE BLCKX

SUBROUTINE BLCK_32(LEN,SRC)
  REAL SRC(1)
!$PRAGMA IGNORE_TKR SRC
  INTEGER (KIND=4) LEN
END SUBROUTINE

SUBROUTINE BLCK_64(LEN,SRC)
  REAL SRC(1)
!$PRAGMA IGNORE_TKR SRC
```

```
  INTEGER (KIND=8) LEN
END SUBROUTINE

END INTERFACE
```

The subroutine call:

```
INTEGER L
REAL S(100)
CALL BLCKX(L,S)
```

在进行正常编译时，BLCKX 调用将调用 BLCK_32；在使用 -xtypemap=integer:64 进行编译时，将调用 BLCK_64。S 的实际类型并不能确定要调用哪个例程。对于基于参数类型、种类或等级来调用特定库例程的包装器来说，这可大大简化为其编写通用接口的工作。

请注意，无法在该指令中指定假定形状数组、Fortran 指针或可分配数组的哑元参数。如果未指定名称，则该指令将应用于过程的所有哑元参数，但假定形状数组、Fortran 指针或可分配数组的哑元参数除外。

2.3.1.3 UNROLL 指令

UNROLL 指令要求您在 !$PRAGMA 后指定 SUN。

!$PRAGMA SUN UNROLL=*n* 指令指示编译器在其优化过程中将以下循环解开 *n* 次。（只有当编译器分析认为此类解开有必要时，它才会解开循环。）

n 是正整数。选项包括：

- 如果 *n*=1，则优化器不得解开任何循环。
- 如果 *n*>1，则优化器可以解开循环 *n* 次。

如果实际解开了任何循环，那么可执行文件会变大。有关详细信息，请参见《*Fortran* 编程指南》中有关性能与优化的章节。

示例—解开循环两次：

```
!$PRAGMA SUN UNROLL=2
```

2.3.1.4 WEAK 指令

WEAK 指令定义一个符号，其优先级比以前定义的相同符号要低。此 pragma 主要用于源文件中以创建库。如果链接程序无法解析弱符号，它并不生成错误消息。

```
!$PRAGMA WEAK (name1 [=name2])
```

WEAK (*name1*) 将 *name1* 定义为弱符号。如果链接程序没有找到 *name1* 的定义，它不会生成错误消息。

WEAK (*name1*=*name2*) 将 *name1* 定义为弱符号以及 *name2* 的别名。

如果程序调用 *name1*，但没有对其进行定义，那么链接程序将使用库中的定义。但是，如果程序定义了自己的 *name1* 版本，那么将采用程序的定义，而不是使用库中 *name1* 的弱全局定义。如果程序直接调用 *name2*，则将使用库中的定义；重复的 *name2* 定义将导致错误。有关更多信息，请参见 Solaris《链接程序和库指南》。

2.3.1.5 OPT 指令

OPT 指令要求您在 !$PRAGMA 后指定 SUN。

OPT 指令设置子程序的优化级别，这将覆盖编译命令行中指定的级别。该指令必须紧挨在目标子程序前面出现，并且仅应用于该子程序。例如：

```
!$PRAGMA SUN OPT=2
       SUBROUTINE smart(a,b,c,d,e)
           ...etc
```

在使用指定 -O4 的 f95 命令编译以上内容时，该指令将覆盖此级别，并以 -O2 级别编译子例程。除非该例程后面另有一个指令，否则下一个子程序将以 -O4 级别编译。

还必须使用 -xmaxopt[=*n*] 选项编译例程以便识别该指令。此编译器选项为 PRAGMA OPT 指令指定最大的优化值：如果 PRAGMA OPT 指定的优化级别大于 -xmaxopt 级别，则使用 -xmaxopt 级别。

2.3.1.6 PIPELOOP[=*n*] 指令

PIPELOOP=*n* 指令要求您在 !$PRAGMA 后面指定 SUN。

此指令必须紧挨在 DO 循环前面出现。*n* 是正整数常量或零，它向优化器断言循环迭代之间是否存在依赖性。值零表示循环中没有迭代间的（即循环带有的）依赖性，优化器可以对循环执行任意管道处理。正值 *n* 表示，循环的第 I 次迭代与第 (I-*n*) 次迭代之间存在依赖性，每次最多只能对 *n* 个迭代进行管道处理。（如果未指定 *n*，则缺省为 0）

```
C    We know that the value of K is such that there can be no
C    cross-iteration dependencies (E.g. K>N)
!$PRAGMA SUN PIPELOOP=0
      DO I=1,N
       A(I)=A(I+K) + D(I)
       B(I)=B(I) + A(I)
      END DO
```

有关优化的更多信息，请参见《Fortran 编程指南》。

2.3.1.7 PREFETCH 指令

-xprefetch 选项标志（第 3.4.166 节 “–xprefetch[=a[,a]]” [129]）可启用一组 PREFETCH 指令，这些指令指示编译器在支持预取的处理器上为指定的数据元素生成预取指令。

```
!$PRAGMA SUN_PREFETCH_READ_ONCE(name)
!$PRAGMA SUN_PREFETCH_READ_MANY(name)
!$PRAGMA SUN_PREFETCH_WRITE_ONCE(name)
!$PRAGMA SUN_PREFETCH_WRITE_MANY(name)
```

有关预取指令的详细信息，另请参见《*C* 用户指南》或《*SPARC Architecture Manual, Version 9*》。

2.3.1.8 ASSUME 指令

ASSUME 指令向编译器提供有关程序中某些点的条件的提示。这些断言可以帮助编译器设定其优化策略。程序员也可以在执行过程中使用这些指令检查程序的有效性。ASSUME 有两种格式。

“点断言”ASSUME 的语法是：

```
!$PRAGMA ASSUME (expression [,probability])
```

另外，“范围断言”ASSUME 的语法是：

```
!$PRAGMA BEGIN ASSUME [expression [, probability)
     block of statements
!$PRAGMA END ASSUME
```

请使用点断言格式来声明编译器可在程序中的该点采用的条件。而使用范围断言格式来声明在语句封闭范围内有效的条件。范围断言中的 BEGIN 和 END 对必须正确嵌套。

必需的 *expression* 是一个布尔表达式，该表达式可在程序中的该点求值，并且其中不包含用户定义的运算符或函数调用（下面列出的除外）。

可选的 *probability* 值是一个介于 0.0 和 1.0 之间的实数或者是整数 0 或 1，它给出表达式为真的可能性。probability 的值为 0.0（或 0）意味着永远不会为真；值为 1.0（或 1）则意味着始终为真。如果没有指定，则认为表达式很有可能为真，但并不一定为真。probability 为 0 或 1 以外其他值的断言是非必然断言。类似地，probability 正好为 0 或 1 的断言是必然断言。

例如，如果程序员知道 DO 循环的长度始终大于 10,000，则为编译器提供该提示可使之生成更好的代码。通常，以下循环在使用 ASSUME pragma 时比不使用时运行得要快。

```
!$PRAGMA BEGIN ASSUME(__tripcount().GE.10000,1) !! a big loop
       do i = j, n
          a(i) = a(j) + 1
```

```
      end do
!$PRAGMA END ASSUME
```

有两个内部函数专用于 ASSUME 指令的表达式子句。（请注意，它们的名称前面有两个下划线。）

`__branchexp()`	用于紧挨在分支转移语句（带有布尔控制表达式）前面的点断言。它与控制分支转移语句的布尔表达式生成相同的结果。
`__tripcount()`	生成紧跟在指令后面的或指令所包含的循环的行程计数。在用于点断言时，指令后面的语句必须位于 DO 的第一行。在用于范围断言时，它应用于最外层的封闭循环。

在将来的版本中，特殊内部函数的列表可能会扩展。

可与 `-xassume_control` 编译器选项结合使用。（请参见第 3.4.110 节 “–xassume_control[=*keywords*]” [96]）。例如，在使用 `-xassume_control=check` 进行编译时，如果行程计数变为小于10,000，则上述示例将生成一条警告。

如果使用 `-xassume_control=retrospective` 进行编译，则在程序终止时会生成一个摘要报告，指出所有断言是真还是假。有关 `-xassume_control` 的详细信息，请参见 `f95` 手册页。

另一个示例：

```
!$PRAGMA ASSUME(__tripcount.GT.0,1)
      do i=n0, nx
```

如果使用 `-xassume_control=check` 编译上述示例，则在由于行程计数为零或负数而没有执行循环时，将会发出一条运行时警告。

2.3.2 并行化指令

只有在使用 `-openmp` 进行编译时才能识别 OpenMP 并行化指令。有关 OpenMP 并行化的详细信息，请参见《*OpenMP API* 用户指南》。

Fortran 编译器支持 OpenMP API（用于实现共享内存并行化）版本 4.0。传统的 Sun 和 Cray 并行化指令现已过时，不应再使用它们。

2.3.2.1 OpenMP 并行化指令

Fortran 编译器将 OpenMPAPI（用于实现共享内存并行化）识别为首选的并行编程模型。该 API 是由 OpenMP 体系结构审查委员会 (http://www.openmp.org) 指定的。

要启用 OpenMP 指令，您必须使用命令行选项 `-xopenmp` 进行编译。（请参见第 3.4.157 节 “–xopenmp[={parallel|noopt|none}]” [126]。）

有关 `f95` 接受的 OpenMP 指令的更多信息，请参见《*OpenMP API* 用户指南》。

2.3.2.2 传统的 Sun/Cray 并行化指令

注 - 传统的 Sun 和 Cray 风格的并行化指令现已过时。首选使用 OpenMP 并行化 API。

2.3.3 IVDEP 指令

`!DIR$ IVDEP` 指令指示编译器忽略其在循环中找到的部分或全部对数组引用的循环附带依赖性，使编译器能够在其他循环之间执行各种循环优化，例如微向量化、分布、软件流水化，否则这些优化将无法实现。当用户知道这些依赖性无关紧要或者实际上永远不会发生时，可以使用该指令。

例如：

```
DO I = 1, N
  A(V(I)) = A(V(I)) + C(I)
END DO
```

在此循环中，存在几处对 `A(V(I))` 的循环附带依赖性，因为 `V(I)` 可能会将重复值传递给索引 `A`，对循环重新排序可能会产生不同的结果。但如果已知 `V` 仅包含不同的值，则可以安全地对循环进行重新排序，并且可以使用 `IVDEP` 指令来实现优化。

可以使用 `–xivdep` 编译器选项（请参见第 3.4.136 节 “–xivdep[=*p*]” [117]）来禁用或确定 `IVDEP` 指令的解释。

`IVDEP` 指令的某些传统解释仅能断言不存在向后循环附带依赖性。Fortran 编译器的缺省值是 `–xivdep=loop`，表示 `IVDEP` 指令断言不存在假定的循环依赖性。

下面的示例解释向后依赖性与向前依赖性。

```
 do i = 1, n   ! BACKWARD LOOP-CARRIED DEPENDENCE
... = a(i-1)   ! S1
   a(i) = ...   ! S2
end do

do i = 1, n   ! FORWARD LOOP-CARRIED DEPENDENCE
   a(i) = ...  ! S3
... = a(i-1)  ! S4
end do
```

第一个循环有一个从 S2 到 S1 的向后循环附带依赖性，第二个循环中有一个从 S3 到 S4 的向前循环附带依赖性。由于第二个循环只有一个向前依赖性，因此可以安全地进行分布和/或微向量化，而第一个循环则不能。

下面是使用缺省值 -xivdep=loop 的情况下使用 IVDEP 的示例：

```
integer target a(n)
integer, pointer :: p(:), q(:)
!DIR$ IVDEP
do i = 1, n
  p(i) = q(i)
  a(i) = a(i-1)
end do
```

p(i) 与 q(i) 之间以及 p(i) 与 a(*) 之间的假定依赖性将被忽略，而 a(i) 与 a(i-1) 之间的显式依赖性则不被忽略。该循环可分割为两个循环，所生成的 p(i) = q(i) 循环可以实现微向量化。

IVDEP 指令应用于紧随其后的 DO 循环。在指令与循环之间不允许有其他代码。!DIR$ IVDEP 也可以应用于数组赋值语句 FORALL 或 WHERE 结构。如果特定的循环存在多个指令（例如 IVDEP 和 UNROLL），编译器会尽可能服从所有指令。

2.4 库接口和 system.inc

Fortran 编译器提供一个 include 文件 system.inc，它为大多数非内在库例程定义了接口。请声明此 include 文件以确保所调用函数及其参数的类型得到正确的设置，尤其是在使用 -xtypemap 更改了缺省数据类型时。

例如，以下命令可能会生成一个运算异常，原因是没有显式地设置函数 getpid() 的类型：

```
integer(4) mypid
mypid = getpid()
print *, mypid
```

getpid() 例程返回一个整数值，但如果没有为该函数声明显式类型，则编译器认为它返回一个实数值。此值将进一步转换为整数，这很有可能会导致浮点错误。

要纠正这种错误，您应该显式地设置所调用的 getuid() 及类似函数的类型：

```
integer(4) mypid, getpid
mypid = getpid()
print *, mypid
```

您可以使用 -Xlist（全局程序检查）选项诊断此类问题。Fortran include 文件 ”system.inc’ 为这些例程提供了显式接口定义。

```
include ’system.inc’
integer(4) mypid
mypid = getpid()
```

```
      print *, mypid
```

通过在调用 Fortran 库中例程的程序单元中包含 `system.inc`，可以自动定义接口，并帮助编译器诊断类型不匹配的问题。（有关更多信息，请参见《*Fortran* 库参考》。）

2.5 编译器用法提示

下面几节提供了一些高效使用 Fortran 编译器的建议方法。下一章中给出了完整的编译器选项参考。

2.5.1 确定硬件平台

某些编译器标志允许用户使用一组特定的硬件平台选项来调节代码的生成。编译器的 `-dryrun` 选项可用来确定本机处理程序：

```
<sparc>% f95 -dryrun -xtarget=native
###     command line files and options (expanded):
### -dryrun -xarch=sparcvis2 -xcache=64/32/4:1024/64/4 -xchip=ultra3i

<x64>% f95 -dryrun -xtarget=native
###     command line files and options (expanded):
### -dryrun -xarch=sse2a -xcache=64/64/2:1024/64/16 -xchip=opteron
```

2.5.2 使用环境变量

可通过设置 `FFLAGS` 或 `OPTIONS` 变量来指定选项。

可以在命令行中显式地使用 `FFLAGS` 或 `OPTIONS`。在使用 `make` 的隐式编译规则时，`make` 程序会自动使用 `FFLAGS`。

示例：设置 `FFLAGS`：(C Shell)

```
demo% setenv FFLAGS '-fast -Xlist'
```

示例： 显式地使用 `FFLAGS`：

```
demo% f95 $FFLAGS any.f
```

在使用 `make` 时，如果按上述方式设置了 `FFLAGS` 变量，而且 makefile 的编译规则是隐式的（即没有显式的编译器命令行），则调用 `make` 的编译效果与以下命令相当：

`f95 -fast -Xlist` *files...*

make 是一个功能非常强大的程序开发工具，可以方便地与所有 Oracle Solaris Studio 编译器一起使用。请参见 make(1) 手册页以及《*Fortran* 编程指南》中的“程序开发”一章。

注 - make 使用的缺省隐式规则可能无法识别具有 .f95 和 .mod（模块文件）扩展名的文件。有关详细信息，请参见《*Fortran* 编程指南》。

2.5.3 内存大小

编译可能需要使用大量内存。这取决于选定的优化级别以及所编译文件的大小和复杂性。如果优化器内存不足，它将尝试通过在较低的优化级别上重试当前过程来进行恢复，并以命令行上的 -o*n* 选项指定的原始级别继续执行后续例程。

运行编译器的处理器应该至少具有 64 兆字节内存；建议使用 256 兆字节内存。此外还应该分配足够的交换空间。最低为 200 MB；建议为 300 MB。

内存使用情况取决于每个过程的大小、优化级别、为虚拟内存设置的限制、磁盘交换文件的大小以及各种其他参数。

在编译包含多个例程的单个源文件时，可能会出现编译器内存或交换空间不足的情况。

如果编译器内存不足，请尝试降低优化级别，或者使用 fsplit(1) 将多例程的源文件分成多个文件，使每个文件包含一个例程。

2.5.3.1 交换空间限制

Oracle Solaris 操作系统命令 swap -s 显示可用的交换空间。请参见 swap(1M)。

示例： 使用 swap 命令：

```
demo% swap -s
total: 40236k bytes allocated + 7280k reserved = 47516k used, 1058708k available
To determine the actual real memory:

demo% /usr/sbin/dmesg | grep mem
mem = 655360K (0x28000000)
avail mem = 602476544
```

2.5.3.2 增加交换空间

使用 mkfile(1M) 和 swap(1M) 可增加工作站上交换空间的大小。您必须成为超级用户才能执行此操作。mkfile 创建一个特定大小的文件，而 swap -a 将该文件添加到系统交换空间中：

```
demo# mkfile -v 90m /home/swapfile
```

```
/home/swapfile 94317840 bytes
demo# /usr/sbin/swap -a  /home/swapfile
```

2.5.3.3 虚拟内存的控制

以 -O3 或更高的优化级别编译非常大的例程时，可能需要额外的内存，这可能会降低编译时性能。您可以通过限制单个进程的可用虚拟内存量来控制这种情况。

在 sh shell 中，请使用 ulimit 命令。请参见 sh(1)。

示例：将虚拟内存限定在 16 MB 以内：

```
demo$ ulimit -d 16000
```

在 csh shell 中，请使用 limit 命令。请参见 csh(1)。

示例： 将虚拟内存限定在 16 MB 以内：

```
demo% limit datasize 16M
```

这些命令行的每一个命令行都会使优化器在数据空间达到 16 MB 时尝试恢复。

此限制值不能大于系统的总可用交换空间，实际上，在进行较大的编译时，此限制值必须足够小才能保证正常地使用系统。请确保没有任何编译占用一半以上的空间。

示例： 对于 32M 的交换空间，请使用以下命令：

在 sh shell 中：

```
demo$ ulimit -d 1600
```

在 csh shell 中：

```
demo% limit datasize 16M
```

最佳设置取决于所请求的优化等级以及可用的实际内存量和虚拟内存量。

在 64 位 Solaris 环境中，应用程序数据段大小的软限制为 2 GB。如果应用程序需要分配更多的空间，请使用 shell 的 limit 或 ulimit 命令取消该限制。

对于 csh，请使用：

```
demo% limit datasize unlimited
```

对于 sh 或 ksh，请使用：

```
demo$ ulimit -d unlimited
```

有关更多信息，请参见 Oracle《*Solaris*（*64* 位）开发者指南》。

2.6 用户提供的缺省选项文件

通过缺省编译器选项文件，用户可以指定一组适用于所有编译（除非另行覆盖）的缺省选项。例如，该文件可以指定所有编译的缺省优化级别为 –xO2，或自动包括文件 setup.il。

启动时，编译器会搜索缺省选项文件，并列出应对所有编译包含的缺省选项。环境变量 SPRO_DEFAULTS_PATH 指定要在其中搜索缺省文件的目录的冒号分隔列表。

如果该环境变量未设置，则会使用一组标准缺省设置。如果该环境变量已设置但为空，则不会使用任何缺省设置。

缺省文件名的格式必须是 compiler.defaults，其中 compiler 为以下值之一：cc、c89、c99、CC、ftn 或 lint。例如，用于 Fortran 编译器的缺省文件为 ftn.defaults。

如果在 SPRO_DEFAULTS_PATH 列出的目录中找到编译器的缺省文件，编译器将读取该文件并在命令行上处理各选项之前先处理这些选项。系统将使用找到的第一个缺省文件，并且会终止搜索。

系统管理员可能会在 *Studio-install-path*/lib/compilers/etc/config 中创建适用于整个系统范围的缺省文件。如果设置了该环境变量，则不会读取已安装的缺省文件。

缺省文件的格式与命令行类似。该文件的每一行都可以包含一个或多个由空格分隔的编译器选项。Shell 扩展（例如通配符和替换）将不会应用于缺省文件中的选项。

SPRO_DEFAULTS_PATH 的值和完全展开的命令行将显示在由 –dryrun 选项生成的详细输出中。

用户在命令行上指定的选项通常会优先于从缺省文件读取的选项。例如，如果缺省文件指定使用 –xO4 进行编译，而用户在命令行上指定了 –xO2，则将使用 –xO2 进行编译。

缺省选项文件中显示的某些选项将附加在命令行中指定的选项之后。这些选项包括预处理程序选项 –I、链接程序选项 –B、–L、–R 和 –l，以及所有文件参数（例如：源文件、对象文件、归档文件和共享对象）。

以下是如何使用用户提供的缺省编译器选项启动文件的示例。

```
demo% cat /project/defaults/ftn.defaults
-I/project/src/hdrs –L/project/libs –llibproj –xvpara
demo% setenv SPRO_DEFAULTS_PATH  /project/defaults
demo% f95 –c –I/local/hdrs –L/local/libs –lliblocal tst.f
```

此命令现在等效于：

```
f95 -fast –xvpara –c –I/local/hdrs –L/local/libs –lliblocal tst.f \
     –I/project/src/hdrs –L/project/libs –llibproj
```

尽管编译器缺省文件提供了可为整个项目设置缺省值的便利方法，但它也可能成为问题难以诊断的原因。将环境变量 SPRO_DEFAULTS_PATH 设置为当前目录以外的绝对路径可避免出现此类问题。

缺省选项文件的接口稳定性未确定。选项处理顺序在以后的发行版中可能会更改。

♦ ♦ ♦ 第 3 章

3 Fortran 编译器选项

本章详细说明 f95 编译器的命令行选项。

- 从第 3.1 节 “命令语法” [39]开始是对用于编译器选项标志的语法的描述。
- 从第 3.3 节 “选项摘要” [40]开始是按功能排列的选项的摘要。
- 从第 3.4 节 “选项参考” [47]开始是详细说明每个编译器选项标志的完整参考。

3.1 命令语法

编译器命令行的通用语法如下：

```
f95 [options] list_of_files  additional_options
```

方括号内的项指示可选参数。方括号不是命令的一部分。*options* 是前面带有短划线 (–) 的选项关键字列表。一些关键字选项将列表中的下一项作为参数。*list_of_files* 是由空格分隔的源文件名、对象文件名或库文件名的列表。此外，有一些选项（例如，-B、-l 和 -L）必须出现在源文件列表之后，而且这些选项可以包括其他文件列表。

3.2 选项语法

典型的编译器选项格式为：

表 3-1 选项语法

语法格式	示例
–flag	–g
–flagvalue	–Dnostep
–flag=value	–xunroll=4
–flag value	–o outfile

在说明各个选项时使用以下印刷约定：

表 3-2 选项的印刷表示法

表示法	含义	示例：文本/实例
[]	方括号包含的参数是可选的。	`-O[n]` `-O4`、`-O`
{}	花括号（大括号）包含必需选项的一组选择。	`-d{y|n}` `-dy`
|	"|" 或 "-" 符号用于分隔多个参数，只能选择其中一个参数。	`-B{dynamic|static}` `-Bstatic`
:	冒号与逗号类似，有时用于分隔多个参数。	`-R`*dir*`[:`*dir*`]` `-R/local/libs:/U/a`
…	省略号表示一系列省略。	`-xinline=`*f1*`[,...`*fn*`]` `-xinline=alpha,dos`

括号、管道符和省略号是在选项描述中使用的元字符，它们不是选项本身的一部分。

选项的一些常规准则如下：

- `–l`*x* 是用于与库 `lib`*x*`.a` 链接的选项。将 `-l`*x* 放在文件名列表之后以确保搜索顺序库，始终是较为安全之举。
- 通常，编译器选项的处理顺序是从左到右，从而允许有选择地覆盖宏选项（包含其他选项的选项）。此规则不适用于链接程序选项。但是，当某些选项（例如 `-I`、`-L` 和 `-R`）在同一命令行上重复出现时，这些选项将累加值，而不是覆盖前面的值。
- 在可选选项列表（例如 `-xhasc[={yes|no}]`）中，所列的第一个选项是出现在命令行上的选项标志不带值时所假定的值。例如，`-xhasc` 与 `-xhasc=yes` 等效。
- 源文件、对象文件和库按它们在命令行上出现的顺序编译并链接。

3.3 选项摘要

在本节中，为了便于参考，将按功能对编译器选项进行分组。有关详细信息，请参见以下几节中相应页面上的内容。

请注意，并非所有选项在 SPARC 和 x64/x86 平台上都可用。有关可用性的说明，请查看详细的参考部分。

下表按功能汇总了 `f95` 编译器选项。该表不包括已过时的和传统的选项标志。某些标志用于多个目的，因此出现多次。

表 3-3 按功能分组的编译器选项

功能		选项标志
编译模式：		
	仅编译；不生成可执行文件	`-c`
	显示由驱动程序生成的命令，但不进行编译	`-dryrun`
	支持 Fortran 77 扩展和兼容性	`-f77`
	不要替换其内容不是由相关 Fortran 模块的编译进行更改的模块文件	`-keepmod`
	保留在编译期间所创建的临时文件。	`-keeptmp`
	指定要将已编译的 `.mod` 模块文件写入的路径	`-moddir=`*path*
	指定要编写的对象文件、库文件或可执行文件的名称	`-o` *filename*
	进行编译并只生成汇编代码	`-S`
	将符号表与可执行文件分离	`-s`
	禁止编译器消息（错误消息除外）	`-silent`
	定义临时文件所在目录的路径	`-temp=`*path*
	显示每个编译阶段占用的时间	`-time`
	显示编译器的版本号及其阶段	`-V`
	详细消息	`-v`
	指定非标准别名情况	`-xalias=`*list*
	使用多个处理器进行编译	`-xjobs=`*n*
	生成 make 依赖项	`-xM`
	(Oracle Solaris) 将调试信息从对象文件链接到可执行文件	`-xs`
已编译的代码：		
	对于外部名称，增加/删除尾随下划线	`-ext_names=`*x*
	内联指定的用户函数	`-inline=`*list*
	与编译位置无关的代码	`-KPIC/-kpic`
	内联某些数学库例程	`-libmil`
	(x86) 在堆栈中保存基于寄存器的函数参数的副本	`-preserve_argvalues`
	STOP 将整数状态值返回给 shell	`-stop_status[=`*yn*`]`
	指定代码地址空间	`-xcode=`*x*
	与 `-inline` 等效。	-xinline
	手动更改编译器用来确定何时内联函数调用的试探式方法	-xinline_param
	生成在编译器内联函数时写入标准输出的报告	-xinline_report
	启用预取指令	`-xprefetch[=`*x*`]`
	指定可选寄存器的使用	`-xregs=`*x*
	指定缺省数据映射	`-xtypemap=`*x*
数据对齐：		
	指定对齐 COMMON 块中的数据	`-aligncommon[=`*n*`]`

功能		选项标志
	强制对齐 COMMON 块数据以允许双字获取/存储	`-dalign`
	强制所有数据按 8 字节边界对齐	`-dbl_align_all`
	按 8 字节边界对齐 COMMON 块数据	`-f`
	支持 little-endian 和 big-endian 平台之间的文件共享。	`-xfilebyteorder`
	指定内存对齐和行为	`-xmemalign[=`*ab*`]`
调试：		
	启用运行时下标范围检查	`-C`
	为使用 `dbx` 调试而进行编译	`-g`
	标志未声明变量的使用	`-u`
	检查 `!$PRAGMA ASSUME` 断言	`-xassume_control=check`
	检查在运行时堆栈是否溢出	`-xcheck=stkovf`
	启用运行时任务普通检查	`-xcommonchk`
	控制发出多少调试和监测信息	`-xdebuginfo`
	为性能分析器进行编译	`-xF`
	控制文件静态变量的全局化，但是不控制函数的全局化	`-xglobalize`
	保留未引用函数和变量的定义	`-xkeep_unref`
	生成交叉引用列表	`-Xlist`*x*
	在没有对象文件的情况下启用调试	`-xs`
诊断：		
	标志非标准扩展名的使用	`-ansi`
	禁止显示指定的警告消息	`-erroff=`
	与错误消息一起显示错误标记名称	`-errtags`
	显示编译器选项的摘要	`-flags, -help`
	显示编译器的版本号及其阶段	`-V`
	详细消息	`-v`
	详细的并行化消息	`-vpara`
	显示/禁止警告消息	`-w`*n*
	(Oracle Solaris) 将调试信息从对象文件链接到可执行文件	`-xs`
链接和库：		
	允许/要求动态/静态库	`-B`*x*
	只允许动态/静态库链接	`-dy, -dn`
	生成动态（共享对象）库	`-G`
	为动态库指定名称	`-h`*name*
	将目录增加到库搜索路径	`-L`*path*
	与库 `lib`*name*`.a` 或 `lib`*name*`.so` 链接	`-l`*name*
	不将库搜索路径生成到可执行文件中。	`-norunpath`
	将运行时库搜索路径生成到可执行文件中	`-R`*path*

功能		选项标志
	禁用递增链接程序 ild	-xildoff
	与优化的数学库链接	-xlibmopt
	与 Sun 性能库链接	-xlic_lib=sunperf
	在各个函数启动之前保留内存区域	-xpatchpadding
	指定程序是否包含对动态绑定符号的引用	-xunboundsym
	链接编辑器选项	-z*x*
	在不重定位的情况下生成纯库	-ztext
数字和浮点：		
	使用非标准浮点首选项	-fnonstd
	选择非标准浮点	-fns
	启用输入过程中的运行时浮点溢出	-fpover
	选择 IEEE 浮点舍入模式	-fround=*r*
	选择浮点优化级别	-fsimple=*n*
	选择浮点捕获模式	-ftrap=*t*
	指定用于格式化输入/输出的舍入方法	-iorounding=*mode*
	将单精度常数提升为双精度常量	-r8const
	启用区间运算并设置相应的浮点环境（包括 -xinterval）	-xia[=*e*]
	启用区间运算扩展	-xinterval[=*e*]
优化与性能：		
	分析循环以了解数据依赖性	-depend
	使用所选的选项进行优化	-fast
	指定程序不能一次在多个线程中执行 I/O。	-fserialio
	指定优化级别	-O*n*
	填充数据布局以便高效使用高速缓存	-pad[=*p*]
	在内存堆栈上分配局部变量	-stackvar
	启用循环解开	-unroll[=*m*]
	调用过程间优化传递	-xipo[=*n*]
	可避免通过编译器进行的初始传递期间的优化而仅在链接时优化，从而缩短编译时间	-xipo_build
	为 #pragma OPT 设置最高优化级别	-xmaxopt[=*n*]
	针对编译后优化进行编译	-xbinopt=prepare
	启用/调整编译器生成的预取指令	-xprefetch=*list*
	控制预取指令的自动生成	-xprefetch_level=*n*
	启用性能分析数据的生成或使用	-xprofile=*p*
	断言不会出现基于内存的陷阱	-xsafe=mem
	使驱动程序在链接行上包括特殊映射文件。	-xsegment_align
	不执行增加代码大小的优化	-xspace

功能		选项标志
	指定当多个进程同时在系统上运行时应用程序将会运行	`-xthroughput`
	自动生成对向量库函数的调用	`-xvector[=`*yn*`]`
并行化：		
	启用 DO 循环的自动并行化	`-autopar`
	与 `-xopenmp=parallel` 等效。	`-fopenmp`
	显示循环的并行化信息	`-loopinfo`
	为手动编码的多线程编程进行编译	`-mt`
	接受 OpenMP API 指令。-xopenmp 选项标志接受以下子选项关键字：`parallel`、`noopt` 和 `none`。	`-xopenmp[=`*keyword*`]`
	识别具有自动并行化的循环中的约简操作	`-reduction`
	详细的并行化消息	`-vpara`
	指定当多个进程同时在系统上运行时应用程序将会运行	`-xthroughput`
源代码：		
	定义预处理程序符号	`-D`*name*`[=`*val*`]`
	取消定义预处理程序符号	`-U`*name*
	接受扩展（132 个字符）源行	`-e`
	将预处理程序应用于 `.F` 和/或 `.F90` 及 `.F95` 文件，但不进行编译	`-F`
	接受 Fortran 95 固定格式输入	`-fixed`
	使用 `fpp` 预处理程序对所有源文件进行预处理	`-fpp`
	接受 Fortran 95 自由格式输入	`-free`
	将目录添加到 include 文件搜索路径	`-I`*path*
	将目录添加到模块搜索路径	`-M`*path*
	区分大小写	`-U`
	在实际参数中将霍尔瑞斯常数视为字符	`-xhasc={yes|no}`
	选择要使用的预处理程序（`cpp` 或 `fpp`）	`-xpp[={fpp|cpp}]`
	允许递归子程序调用	`-xrecursive`
目标平台：		
	指定内存模型（32 位或 64 位）。	`-m32` | `–m64`
	为优化器指定目标平台指令集	`-xarch=`*a*
	为优化器指定目标高速缓存属性	`-xcache=`*a*
	为优化器指定目标处理器	`-xchip=`*a*
	为优化器指定目标平台	`-xtarget=`*a*

3.3.1 常用选项

编译器有许多可通过可选命令行参数选择的功能。下面的简要列表列出了一些常用选项，让您一睹为快。

表 3-4 常用选项

操作	选项
调试 – 在例程之间进行全局程序检查以确保参数、通用块等的一致性。	–Xlist
调试 – 生成其他用于启用 dbx 和调试的符号表信息。	–g
性能 – 调用优化器以生成运行速度更快的程序。	–O[*n*]
性能 – 使用一组预先确定的选项，为本机平台生成高效的编译时和运行时。	–fast
动态 (–Bdynamic) 或静态 (–Bstatic) 库绑定。	–B*x*
仅编译 – 禁止链接；为每个源文件生成一个 .o 文件。	–c
输出文件 – 将可执行输出文件命名为 *nm* 而不是 a.out。	–o nm
源代码 – 编译固定格式的 Fortran 源代码。	-fixed

3.3.2 宏标志

某些选项标志是可扩展为由其他标志组成的特定集合的宏。之所以提供这些选项标志，是为了便于指定通常一起用来选择某项功能的多个选项。

表 3-5 宏选项标志

选项标志	扩展
-dalign	-xmemalign=8s -aligncommon=16
-f	-aligncommon=16
-fast	有关完整的当前扩展，请参见 –fast 说明。
-fnonstd	-fns -ftrap=common
-xia=widestneed	-xinterval=widestneed -ftrap=%none -fns=no -fsimple=0
-xia=strict	-xinterval=strict -ftrap=%none -fns=no -fsimple=0
-xtarget	-xarch=*a* -xcache=*b* -xchip=*c*

命令行上跟在宏标志后面的设置将覆盖或增加宏扩展。

3.3.3 向后兼容性和传统选项

提供以下选项的目的是为了与早期发行版的编译器和某些 Fortran 传统功能实现向后兼容。

表 3-6 向后兼容性选项

操作	选项
通过 ENTRY 语句保留实际参数	-arg=local
允许为常量参数赋值。	–copyargs
在调用参数列表中将霍尔瑞斯常数视为字符或无类型。	-xhasc[={yes|no}]
支持 Fortran 77 扩展和约定	-f77
非标准运算—允许非标准运算。	–fnonstd
为主机系统优化性能。	–native
DO 循环—使用单程 DO 循环。	–onetrip
允许存在传统的别名情况	-xalias=*keywords*

建议在生成可移植的 Fortran 程序时不要使用这些选项标志。

3.3.4 已过时的选项标志

下面的选项被认为是已过时的，不应使用它们。在编译器的以后发行版本中可能会删除这些选项。

表 3-7 过时的 f95 选项

选项标志	等效
-a	-xprofile=tcov
-native	-xtarget=native
-noqueue	许可证排队。不再需要。
-p	分析。使用 -pg 或性能分析器
-pic	-xcode=pic13
-PIC	-xcode=pic32
-silent	已忽略。
-xarch={v7,v8,v8a}	使用 –m32

3.4 选项参考

本节介绍所有 f95 编译器命令行选项标志，包括各种风险、限制、警告、交互、示例和其他详细信息。

除非另行指明，否则每个选项在 SPARC 和 x64/x86 平台上都有效。仅在 SPARC 平台上有效的选项标志标有 (SPARC)。仅在 x64/x86 平台上有效的选项标志标有 (x86)。

标有（已过时）的选项标志已过时，不应使用。在许多情况下，它们已经被其他应该使用的选项或标志取代。

3.4.1 –aligncommon[={1|2|4|8|16}]

指定通用块和标准数字序列类型中数据的对齐。

此值表示通用块和标准数值序列类型中数据元素的最大对齐（以字节为单位）。

注 - 标准数字序列类型是一种派生类型，包含一个 SEQUENCE 语句和纯缺省组件数据类型（不带 KIND= 或 * *size* 的 INTEGER、REAL、DOUBLEPRECISION、COMPLEX）。任何其他类型（例如：REAL*8）都会使数字序列类型成为非标准类型。

例如，-aligncommon=4 会将自然对齐方式为 4 字节或大于 4 字节的数据元素与 4 字节边界对齐。

该选项不影响自然对齐方式小于指定大小的数据。

如果不使用 -aligncommon，则编译器会将通用块和数值序列类型中的元素与（最多）4 字节边界对齐。

如果指定不带值的 -aligncommon，则缺省值为 1 – 所有的通用块和数值序列类型元素都与字节边界对齐（元素之间无填充）。

在未启用 64 位的平台上，-aligncommon=16 恢复为 -aligncommon=8。

不要将 -aligncommon=1 与 -xmemalign 一起使用，因为这些声明会发生冲突，在某些平台和配置上可能会引发段故障。

如果在 SPARC 平台上使用 -aligncommon=1，可能会因未对齐而导致总线错误，需要使用适当的 -xmemalign 选项值。根据具体的应用程序，–xmemalign=1s、–xmemalign=4i 或 –xmemalign=8i 会在避免段故障的同时提供最佳性能。

另请参见 –xmemalign

3.4.2 –ansi

标识许多非标准扩展。

如果在源代码中使用非标准 Fortran 扩展，则会发出警告消息。

3.4.3 –arg=local

通过 ENTRY 语句保留实际参数。

在使用此选项编译具有替换入口点的子程序时，f95 将使用复制/恢复功能保留哑元参数和实际参数之间的关联。

提供此选项的目的是与传统的 Fortran 77 程序兼容。依赖此选项的代码是非标准的。

3.4.4 –autopar

启用自动循环并行化。

查找相应的循环并使之并行化，以便在多个处理器上并行运行。分析循环以了解迭代间的数据依赖性并重构循环。如果未将优化级别指定为 -O3 或更高，则会将其自动提升到 -O3。

在使用任何并行化选项（包括 -autopar）时，也要指定 -stackvar 选项。当使用 -autopar 时，-stackvar 选项可提供更好的性能，因为它可允许优化器为并行检测其他机会。有关如何为主线程堆栈和从线程堆栈设置大小的信息，请参见 -stackvar 选项描述。

如果程序已经包含对 libthread 线程库的显式调用，请避免使用 -autopar。请参见第 3.4.60 节 “-mt[={yes|no}]” [73]中的说明。

-autopar 选项不适用于单处理器系统，而且已编译代码的运行速度通常会更慢。

当运行由 -xautopar 编译器选项自动并行化的程序时，请使用 OMP_NUM_THREADS 环境变量指定要使用的线程数。如果 OMP_NUM_THREADS 未设置，则使用的线程的缺省数量为计算机上的内核数，上限为 32。将 OMP_NUM_THREADS 设置为 1，则会仅使用一个线程运行。为了获得最佳性能，使用的线程数不应超出计算机上的可用硬件线程（或虚拟处理器）数量。在 Oracle Solaris 系统上，可以使用 psrinfo(1M) 命令确定此数量。在 Linux 系统上，可以检查 /proc/cpuinfo 文件来确定此数量。有关更多信息，请参见 *OpenMP API* 用户指南。

除了 OMP_NUM_THREADS 以外，应用于 OpenMP 程序的其他环境变量可用于由 -xautopar 编译器选项自动并行化的程序。有关环境变量的说明，请参见《*Oracle Solaris Studio OpenMP API* 用户指南》。

如果使用 -autopar 并在同一步骤中进行编译和链接，则会自动链接多线程库和线程安全的 Fortran 运行时库。如果使用 -autopar 并在不同的步骤中进行编译和链接，则还必须使用 -autopar 进行链接以确保链接相应的库。

将 -reduction 选项与 -autopar 结合使用可识别循环中的约简操作。

使用 -loopinfo 选项显示哪些循环是并行化的，哪些不是。

有关用户控制的显式并行化，则使用 OpenMP 指令和 -xopenmp 选项。

3.4.5 –B{static|dynamic}

首选动态库链接或要求静态库链接。

在 -B 与 dynamic 或 static 之间不允许有空格。如果未指定 -B，则缺省值为 -Bdynamic。

- –Bdynamic：首选动态链接（尝试找到共享库）。
- –Bstatic：需要静态链接（无共享库）。

另请注意：

- 如果指定 static，但是链接程序仅找到动态库，则不链接该库，同时发出警告“未找到库”。
- 如果指定 dynamic，但链接程序仅找到静态版本的库，则链接该库，并且不发出警告。

您可以在命令行上切换 -Bstatic 和 -Bdynamic。也就是说，通过在命令行上指定 -Bstatic 和 -Bdynamic 任意多次，可以静态链接一些库并动态链接一些库，如下所示：

```
f95 prog.f -Bdynamic -lwells -Bstatic -lsurface
```

这些是加载器和链接程序选项。在不同的步骤中使用编译命令的 -B*x* 选项进行编译和链接时，将要求在链接步骤中也使用该选项。

不能在命令行上同时指定 -Bdynamic 和 -dn，因为 -dn 禁用动态库的链接。

在 64 位 Solaris 环境中，许多系统库仅作为共享动态库提供，其中包括 libm.so 和 libc.so（不提供 libm.a 和 libc.a）。这意味着，在 64 位 Solaris 环境中，-Bstatic 和 -dn 可能会导致链接错误。这些情况下应用程序必须与动态库链接。

不推荐同时使用静态 Fortran 运行时系统库和动态 Fortran 运行时系统库，因为这会导致链接程序错误或无提示的数据损坏。始终保持同最新的共享动态 Fortran 运行时系统库的链接。

有关静态库和动态库的更多信息，请参见《*Fortran* 编程指南》。

3.4.6 –C

检查数组引用以查找超出范围的下标并在运行时检查一致性。

如果数组下标超过所声明的大小，可能会导致意外结果（包括段故障）。`-C` 选项检查源代码中和执行过程中可能的数组下标违规。`-C` 还添加了对数组语法表达式中数组符合性的运行时检查。

指定 `-C` 可能会使可执行文件更大。

如果使用 `-C` 选项，则会将数组下标违规视为错误。如果在编译过程中检测到源代码中存在数组下标范围违规，则会将它视为编译错误。

如果只能在运行时确定数组下标违规，则编译器会将范围检查代码生成到可执行程序中。这可能导致执行时间增加。因此，应该在开发和调试程序时启用完全数组下标检查，然后重新编译最后产生的可执行程序，而不必进行下标检查。

3.4.7 –c

仅编译；生成对象 `.o` 文件，但禁止链接。

针对每个源文件编译 `.o` 文件。如果仅编译一个源文件，则可以使用 `-o` 选项来指定所写入的 `.o` 文件的名称。

3.4.8 –copyargs

允许为常量参数赋值。

允许子程序更改其为常量的哑元参数。提供此选项只是为了允许编译和执行传统代码而不出现运行时错误。

- 在不使用 `-copyargs` 的情况下，如果将常量参数传递给子例程，然后在子例程内尝试更改该常量，则运行将终止。

- 在使用 -copyargs 的情况下，如果将常量参数传递给子例程，然后在子例程内更改该常量，则运行不一定终止。

当然，除非使用 -copyargs 进行编译，否则终止的代码是不符合 Fortran 标准的。此外，这样的代码通常是不可预知的。

3.4.9 –D*name*[=*def*]

为预处理程序定义符号 *name*。

此选项仅适用于 .F、.F90、.F95 和 .F03 源文件。

–D*name*=*def* 将 *name* 定义为具有值 *def*

–D*name* 将 *name* 定义为 1

在命令行上，此选项将定义 *name*，就如同

```
#define name[=def]
```

已经出现在源文件中。如果未指定 =*def*，则名称 *name* 的值定义为 1。宏符号 *name* 会传递到预处理程序 fpp（或 cpp – 请参见 -xpp 选项）进行扩展。

预定义的宏符号具有两个前导下划线。Fortran 语法可能不支持这些宏的实际值 – 它们只应出现在 fpp 或 cpp 预处理程序指令中。（请注意两个前导下划线。）

- 编译器版本是在 _ _SUNPRO_F90 和 _ _SUNPRO_F95 中预定义的（采用十六进制）。例如，对于 Oracle Solaris Studio 12.4 发行版中的 Fortran 编译器 8.6 版，_ _SUNPRO_F95 为 0x860。
- 以下宏是在相应系统上预定义的：

  _ _sparc、_ _unix、_ _sun、_ _SVR4、__i386、_ _SunOS_5_10 和 _ _SunOS_5_11

  符号 _ _sparc、_ _sparcv8 和 _ _sparcv9 是在其各自的 SPARC 系统上定义的。
- 以下预定义值不带下划线，但是在以后的发行版中可能会删除这些值： sparc、unix 和 sun。
- 在 64 位 x86 系统上，定义了宏 __amd64 和 __x86_64。

使用 -v 详细选项编译 .F、.F90、.F95 或 .F03 源文件可查看编译器采用的预处理程序定义。

您可以在类似如下的预处理程序条件中使用这些值：

```
#ifdef _ _sparc
```

缺省情况下，f95 使用 fpp(1) 预处理程序。与 C 预处理程序 cpp(1) 一样，fpp 会扩展源代码宏并允许对代码进行条件编译。与 cpp 不同的是，fpp 能够识别 Fortran 语法，并

作为首选的 Fortran 预处理程序。使用 -xpp=cpp 标志可以强制编译器明确使用 cpp 而非 fpp。

3.4.10 –dalign

对齐 COMMON 块和标准数值序列类型，并生成速度更快的多字装入/存储。

此标志可更改 COMMON 块、数值序列类型和 EQUIVALENCE 类中的数据布局，并使编译器能够为该数据生成速度更快的多字装入/存储。

数据布局效果与 -f 标志的效果相同： COMMON 块和 EQUIVALENCE 类中的双精度和四精度数据在内存中根据其“自然”对齐方式（即，与 8 字节边界对齐）进行布局；如果在 64 位环境中使用 -m64 进行编译，则四精度数据与 16 字节边界对齐。缺省情况下，按 4 字节边界对齐 COMMON 块中的数据。还允许编译器采用自然对齐方式并生成速度更快的多字装入/存储以引用数据。

在 SPARC 处理器上，如果结合使用 -dalign 和 -xtypemap=real:64,double:64,integer:64，还会导致 64 位整数变量进行双字对齐。

注 - -dalign 可能导致数据以非标准方式对齐，从而使 EQUIVALENCE 或 COMMON 中的变量出现问题，并可能在需要 -dalign 的情况下使程序变为不可移植。

-dalign 是一个宏，它等效于：

-xmemalign=8s -aligncommon=16（在 *SPARC* 平台上）

-aligncommon=8（在 *32* 位 *x86* 平台上）

-aligncommon=16（在 *64* 位 *x86* 平台上）。

如果使用 -dalign 编译某个子程序，请使用 -dalign 编译该程序的所有子程序。此选项包含在 -fast 选项中。

请注意，因为 -dalign 调用 -aligncommon，所以此选项还影响标准数值序列类型。请参见第 3.4.1 节 “–aligncommon[={1|2|4|8|16}]” [47]。

3.4.11 –dbl_align_all[={yes|no}]

强制与 8 字节边界对齐数据。

值为 yes 或 no。如果是 yes，所有变量将与 8 字节边界对齐。缺省值为 -dbl_align_all=no。

在 64 位环境中使用 -m64 进行编译时，此标志会使四精度数据与 16 字节边界对齐。

此标志不改变 COMMON 块或用户定义结构中的数据的布局。

与 -dalign 一起使用可以提高多字装入/存储的效率。

如果使用了此标志，则所有例程都必须使用此标志进行编译。

3.4.12 –depend[={yes|no}]

分析循环以了解迭代间数据的依赖性并重构循环。循环重构包括循环互换、循环熔合和标量替换。

如果不指定 -depend，则缺省值为 -depend= es。如果指定 -depend 但不指定参数，则编译器采用 -depend= es。

要关闭依赖性分析，应使用 -depend=no 进行编译。

-xdepend 与 -depend 等效。

3.4.13 –dryrun

显示由 f95 命令行驱动程序生成的命令，但不进行编译。

此选项在调试时非常有用，它显示编译器为执行编译将调用的命令和子选项。

3.4.14 –d{y|n}

允许或禁止对整个可执行文件使用动态库。

- –dy：值为 Yes，允许使用动态/共享库。
- –dn：值为 No，不允许使用动态/共享库。

如果未指定，则缺省值为 -dy。

与 -B*x* 不同，此选项适用于整个可执行文件，并且只需在命令行上出现一次。

–dy|–dn 是加载器和链接程序选项。如果在不同的步骤中使用这些选项进行编译和链接，则需要在链接步骤中使用相同的选项。

在 64 位 Solaris 环境中，许多系统库不只是作为共享动态库提供，其中包括 `libm.so` 和 `libc.so`（不提供 `libm.a` 和 `libc.a`）。这意味着，`-dn` 和 `-Bstatic` 可能会导致在 64 位 Solaris 环境、32 位 x86 Solaris 平台以及所有 32 位 Solaris 平台（从 Solaris 10 发行版开始）中出现链接错误。这些情况下应用程序必须与动态库链接。

3.4.15 –e

接受扩展长度的输入源代码行。

扩展的源代码行中最多可以包含 132 个字符。编译器会在右侧用结尾空白一直填充到第 132 列。如果在使用 `-e` 进行编译时使用续行，则不跨行拆分字符常量；否则，可能会在常量中插入不必要的空白。

3.4.16 –erroff[={%all|%none|*taglist*}]

禁止由标记名称列出的警告消息。

禁止显示在标记名称的逗号分隔列表 *taglist* 中指定的警告消息。如果选项值为 `%all`，则禁止显示所有警告，这在效果上等同于 `-w` 选项。如果选项值为 `%none`，则不禁止显示任何警告。不带参数的 `–erroff` 与 `–erroff=%all` 等效。

示例：

```
f95 -erroff=WDECL_LOCAL_NOTUSED ink.f
```

使用 `-errtags` 选项可查看与警告消息关联的标记名称。

3.4.17 –errtags[={yes|no}]

与每个警告消息一起显示消息标记。

如果使用 `-errtags=yes`，编译器的内部错误标记名称将与警告消息一起显示。`-errtags` 本身与 `-errtags=yes` 等效。

缺省情况下不显示标记 (`-errtags=no`)。

```
demo% f95 -errtags ink.f
ink.f:
 MAIN:
"ink.f", line 11: Warning: local variable "i" never used (WDECL_LOCAL_NOTUSED)
```

3.4.18 –errwarn[={%all|%none|*taglist*}]

将警告消息视为错误。

taglist 指定应视为错误的警告消息对应的标记名称的逗号分隔列表。如果使用 %all，则将所有警告视为错误。如果使用 %none，则不将任何警告视为错误。

另请参见 -errtags。

3.4.19 –ext_names=*e*

创建带有或不带尾随下划线的外部名称。

e 必须是 plain、underscores 或 fsecond-underscore。缺省为 underscores。

–ext_names=plain： 不增加结尾下划线。

–ext_names=underscores： 增加结尾下划线。

–ext_names=fsecond-underscore：在包含一个下划线的外部名称上附加两个下划线，在不包含下划线的外部名称上附加一个下划线。

外部名称是子例程、函数、块数据子程序或标记通用块的名称。此选项既影响例程入口点的名称，又影响调用例程时使用的名称。使用此标志可允许 Fortran 例程调用其他编程语言例程（以及被后者调用）。

提供 fsecond-underscore 是为了与 gfortran 兼容。

3.4.20 –F

调用源文件预处理程序，但不编译。

将 fpp 预处理程序应用于命令行上列出的 .F、.F90、.F95 和 .F03 源文件，并将处理结果写入同名文件，但将该文件的扩展名更改为 .f（或者是 .f95 或 .f03），不进行编译。

示例：

```
f95 -F source.F
```

将已处理的源文件写入 source.f

fpp 是 Fortran 的缺省预处理程序。通过指定 -xpp=cpp，可以改为选择 C 预处理程序 cpp。

3.4.21 –f

对齐 COMMON 块中的双精度和四精度数据。

`-f` 是一个传统的选项标志，它与 `-aligncommon=16` 等效。首选使用 `-aligncommon`。

缺省情况下，按 4 字节边界对齐 COMMON 块中的数据。`-f` 将 COMMON 块和 EQUIVALENCE 类中双精度和四精度数据的数据布局更改为在内存中根据“自然”对齐方式（即，与 8 字节边界对齐）放置；如果在 64 位 环境中使用 `-m64` 进行编译，则四精度数据与 16 字节边界对齐。

注 - `-f` 可能导致数据以非标准方式对齐，从而使 `EQUIVALENCE` 或 `COMMON` 中的变量出现问题，并可能在需要 `-f` 的情况下使程序变为不可移植。

如果使用 `-f` 编译程序的任何部分，则要求使用 `-f` 编译该程序的所有子程序。

此选项本身并不允许编译器针对双精度和四精度数据生成速度更快的多字获取/存储指令。`-dalign` 选项执行此操作并调用 `-f`。相对于以前的 `-f`，请优先使用 `-dalign`。请参见第 3.4.10 节 “–dalign” [52]。由于 `-dalign` 是 `-fast` 选项的一部分，因此 `-f` 也是它的一部分。

3.4.22 –f77[=*list*]

选择 FORTRAN 77 兼容性模式。

此选项标志用于将传统的 FORTRAN 77 源程序（包括包含 Sun WorkShop `f77` 编译器所接受的语言扩展的源程序）移植到 `f95` Fortran 编译器。（不再存在单独的 FORTRAN 77 编译器。）

list 是从下面可能的关键字中选择的逗号分隔列表：

关键字	含义
`%all`	启用所有 Fortran 77 兼容性功能。
`%none`	禁用所有 Fortran 77 兼容性功能。
`backslash`	在字符串中，将反斜线作为转义序列接受。
`input`	允许 `f77` 接受的输入格式。
`intrinsics`	将内部函数的识别限制为仅识别 Fortran 77 内部函数。
`logical`	接受 Fortran 77 的逻辑变量使用，如： ■ 将整数值赋予逻辑变量

关键字	含义
	■ 允许在逻辑条件语句中使用算术表达式，用 .NE.0 表示 .TRUE。 ■ 允许关系运算符 .EQ. 和 .NE. 与逻辑操作数一起使用
misc	允许多种 f77 Fortran 77 扩展。
output	生成 f77 样式的格式化输出，包括列表式输出和 NAMELIST 输出。
subscript	允许将非整数表达式作为数组下标。
tab	启用 f77 样式的制表符格式，包括无限制的源代码行长度。对于长度小于 72 个字符的源代码行，将不增加空白填充。

对于所有关键字，通过在前面加上 no% 可禁用相应功能，如下所示：

```
-f77=%all,no%backslash
```

如果未指定 -f77，则缺省为 -f77=%none。使用不带列表的 -f77 与指定 -f77=%all 是等效的。

异常捕获与 -f77：

指定 -f77 并不会更改 Fortran 捕获模式（即 -ftrap=common）。在运算异常捕获方面，f95 与 Fortran 77 编译器的行为不同。Fortran 77 编译器允许在出现运算异常之后继续执行。使用 -f77 进行编译还会使程序在退出时调用 ieee_retrospective，以报告可能出现的任何运算异常。在命令行上，在 -f77 选项标志之后指定 -ftrap=%none 可以模拟原来的 Fortran 77 行为。

有关 f77 兼容性以及从 Fortran 77 迁移到 Fortran 95 的完整信息，请参见第 4.12 节 “混合语言” [171]。

有关如何处理可能导致错误结果的非标准编程症状，另请参见 -xalias 标志。

3.4.23 –fast

选择优化执行性能的选项。

注 - 该选项定义为其他选项的特殊选择集，它会随版本和编译器的不同而变化。另外，-fast 选择的某些选项并非在所有平台上都可用。使用 -dryrun 标志进行编译可查看 -fast 的扩展。

-fast 可为某些基准测试应用程序提供高性能。但是，对于您的应用程序，选项的特定选择可能是合适的，也可能是不合适的。使用 -fast 是编译应用程序以获得最佳性能的良好起点。但是，仍然可能需要进行其他调整。如果用 -fast 编译时程序不能正常运行，请仔细查看组成 -fast 的各个选项，只调用那些适用于您程序的选项，使程序正常运行。

另请注意，用 -fast 编译的程序对于一些数据集可能会表现出良好的性能和精确的结果，而对于另一些数据集则不然。对于那些依赖浮点运算的特殊属性的程序，请避免用 -fast 进行编译。

由于 -fast 选择的某些选项具有链接含义，因此，如果在不同的步骤中进行编译和链接，还请务必用 -fast 进行链接。

–fast 会选用以下选项：

- -xtarget=native 硬件目标。

  如果程序要在不同于编译计算机的目标上运行，请在 -fast 后加上代码生成器选项。例如： f95 -fast -xtarget=ultraT2 ...
- -O5 优化级别选项。
- -depend 选项分析循环的数据依赖性和可能的重构。（在优化级别 -xO3 和更高级别上进行编译时，此选项始终启用。）
- -libmil 选项，用于系统提供的内联扩展模板。

  对于依赖异常处理的 C 函数，请在 -fast 之后加上 -nolibmil（如 -fast -nolibmil）。如果使用了 -libmil，则使用 errno 或 matherr(3m) 无法检测到异常。
- -fsimple=2 选项，用于主动浮点优化。

  如果要求严格符合 IEEE 754 标准，则 –fsimple=2 是不合适的。请参见第 3.4.36 节 “–fsimple[={1|2|0}]” [63]。
- -dalign 选项可为通用块中的双精度和四精度数据生成双字装入和存储。使用此选项可以在通用块中生成非标准的 Fortran 数据对齐。
- -xlibmopt 选项选择优化的数学库例程。
- -pad=local 在局部变量之间插入填充（如果适用），以提高高速缓存利用率。(SPARC)
- -xvector=lib 使用向量参数将 DO 循环内的某些数学库调用变换为对向量化库等效例程的单个调用。(SPARC)
- -fma=fused 启用自动生成浮点混合乘加指令。
- –fns 选择非标准浮点运算异常处理和渐进下溢。请参见第 3.4.28 节 “–fns[={yes|no}]” [60]。
- 选择 -fround=nearest，因为 –xvector 和 –xlibmopt 要求使用该选项。(Oracle Solaris)
- -ftrap=common 用于捕获常见的浮点异常，在 f95 中处于启用状态。
- -nofstore 对强制表达式具有结果精度这一行为加以取消。(x86)
- 在 x86 平台上，-xregs=frameptr 允许编译器将帧指针寄存器用作通用寄存器。有关详细信息，尤其是在编译混合 C、C++ 和 Fortran 源代码情况下的详细信息，请参见 -xregs=frameptr 的描述。在 -fast 之后指定 -xregs=no%frameptr，帧指针寄存器将不会作为通用寄存器使用。(x86)

可以对此列表进行增减，方法是在 -fast 选项之后加上其他选项，如下所示：

```
f95 -fast -fsimple=1 -xnolibmopt ...
```

它会覆盖 -fsimple=2 选项，并禁用由 -fast 选择的 -xlibmopt。

由于 -fast 会调用 -dalign、-fns 和 -fsimple=2，因此用 -fast 编译的程序会导致非标准浮点运算、非标准数据对齐以及非标准表达式求值顺序。对于大多数程序来说，这些选择可能是不合适的。

请注意，由 -fast 标志选择的一组选项会随各个编译器发行版而发生变化。使用 -dryrun 调用编译器可显示 -fast 扩展：

```
<sparc>% f95 -dryrun -fast |& grep ###
          ###     command line files and options (expanded):
          ### -dryrun -xO5 -xarch=sparcvis2 -xcache=64/32/4:1024/64/4
              -xchip=ultra3i -xdepend=yes -xpad=local -xvector=lib
              -dalign -fsimple=2 -fns=yes -ftrap=common -xlibmil
              -xlibmopt -fround=nearest
```

3.4.24 –fixed

指定固定格式的 Fortran 95 源输入文件。

无论采用哪个文件扩展名，命令行上的所有源文件都将被解释为固定格式文件。通常，f95 仅将 .f 文件解释为固定格式文件，而将 .f95 解释为自由格式文件。

3.4.25 –flags

与 -help 等效。

3.4.26 –fma[={none|fused}]

启用自动生成浮点混合乘加指令。-fma=none 禁用这些指令的生成。-fma=fused 允许编译器通过使用浮点混合乘加指令，尝试寻找改进代码性能的机会。

缺省值为 -fma=none。

要生成混合乘加指令，最低体系结构要求是 -xarch=sparcfmaf（对于 SPARC）和 -xarch=avx2（对于 x86）。如果生成了混合乘加指令，编译器会标记此二进制程序，

以防止该程序在不支持混合乘加指令的平台上执行。如果未使用最低体系结构，则 -fma=fused 不起作用。

混合乘加指令可以免除乘法和加法之间的中间舍入步骤。因此，如果使用 -fma=fused 编译，程序可能会生成不同的结果，但精度通常会增加而不是降低。

3.4.27 –fnonstd

按非标准首选项初始化浮点硬件。

此选项是以下选项标志组合的宏：

```
–fns  -ftrap=common
```

指定 -fnonstd 大致等效于 Fortran 主程序开始处的以下两个调用。

```
i=ieee_handler("set", "common", SIGFPE_ABORT)
call nonstandard_arithmetic()
```

nonstandard_arithmetic() 例程替代了早期发行版中已过时的 abrupt_underflow() 例程。

主程序必须使用此选项进行编译才能有效。

使用此选项初始化浮点硬件，以达到下列目的：

- 在出现浮点异常时终止（捕获）该异常。
- 如果下溢结果将提高速度，而不是生成 IEEE 标准所要求的次正规数，则将该结果刷新为零。

有关渐进下溢和次正规数的更多信息，请参见 -fns。

通过 -fnonstd 选项，可以针对浮点溢出、被零除和无效运算异常启用硬件陷阱。这些情况将转换为 SIGFPE 信号，而且如果程序没有 SIGFPE 处理程序，它将以转储内存而终止。

有关更多信息，请参见 ieee_handler(3m) 和 ieee_functions(3m) 手册页，以及《数值计算指南》和《*Fortran* 编程指南》。

3.4.28 –fns[={yes|no}]

选择非标准浮点模式。

缺省值为标准浮点模式 (–fns=no)。（请参见《*Fortran* 编程指南》的“浮点运算”一章。）

可以选择使用 =yes 或 =no，提供了一种在包括 -fns 的其他宏标志（如 -fast）之后切换该标志的方式。-fns 不带值等同于 -fns=yes。

此选项标志在程序开始执行时启用非标准浮点模式。在 SPARC 平台上，指定非标准浮点模式会禁用“渐进下溢”，从而导致将微小的结果刷新为零，而不是生成次正规数。此外，还会导致次正规操作数在无提示的情况下替换为零。在那些不支持硬件中的渐进下溢和次正规数的 SPARC 系统上，使用此选项将显著提高某些程序的性能。

下表中的 *x* 不会导致总下溢，当且仅当 |x| 处于所示范围之一时，*x* 才是一个次正规数：

表 3-8 低于正常的 REAL 和 DOUBLE

数据类型	范围
REAL	0.0 < |x| < 1.17549435e–38
DOUBLE PRECISION	0.0 < |x| < 2.22507385072014e–308

有关次正规数的详细信息，请参见《数值计算指南》；有关此选项和类似选项的更多信息，请参见《*Fortran* 编程指南》的“浮点运算”一章。（一些算术家使用术语非正规数来代替次正规数。）

缺省情况下，对浮点首选项进行标准初始化：

- IEEE 754 浮点运算是不停止的（即出现异常时不终止）。
- 下溢是渐进式的。

在 x86 平台上，此选项仅对 Pentium III 和 Pentium 4 处理器（SSE 或 SSE2 指令集）启用。

在 x86 上，-fns 选择 SSE 刷新为零模式以及非正规数为零模式（如果可用的话）。此标志导致将次正规结果刷新为零。如果可用的话，此标志还导致将次正规操作数视为零。此标志对使用 SSE 或 SSE2 指令集的传统 x87 浮点运算没有影响。

主程序必须使用此选项进行编译才能有效。

3.4.29 -fopenmp

与 -xopenmp=parallel 相同。

3.4.30 –fpover[={yes|no}]

检测格式化输入中的浮点溢出。

如果指定了 -fpover=yes ，则 I/O 库将检测格式化输入中的运行时浮点溢出并返回错误条件 (1031)。缺省情况下不进行此类溢出检测 (–fpover=no)。不带值的 -fpover 与 -fpover=yes 等效。同 –ftrap 组合使用可获得完整的诊断信息。

3.4.31 –fpp

使用 fpp 强制对输入进行预处理。

通过 fpp 预处理程序传递在 f95 命令行上列出的所有输入源文件，而不管文件扩展名为何。（通常，fpp 仅自动预处理扩展名为 .F、.F90 或 .F95 的文件。）另请参见 第 3.4.165 节 “–xpp={fpp|cpp}” [129]。

3.4.32 –fprecision={single|double|extended}

(x86) 初始化非缺省的浮点舍入精度模式。

在 x86 平台上，将浮点精度模式设置为 single、double 或 extended。

如果值为 single 或 double，此标志会在程序启动时将舍入精度模式相应地设置为单精度或双精度。如果值为 extended,，或在缺省情况下且未指定 -fprecision 标志，则舍入精度模式将初始化为扩展精度。

该选项仅在 x86 系统上且仅在编译主程序时才有效，但如果编译 64 位 (-m64) 或 SSE2 启动的 (-xarch=sse2) 处理器，忽略此选项。在 SPARC 系统上也忽略此选项。

3.4.33 –free

指定自由格式源输入文件。

命令行上的所有源文件都将被解释为 f95 自由格式源文件，而不管文件扩展名为何。通常，f95 将 .f 文件解释为固定格式文件，而将 .f95 解释为自由格式文件。

3.4.34 –fround={nearest|tozero|negative|positive}

设置启动时有效的 IEEE 舍入模式。

缺省值为 -fround=nearest。

主程序必须使用此选项进行编译才能有效。

该选项将 IEEE 754 舍入模式设置为：

- 可以由编译器在对常量表达式求值时使用。
- 是在程序初始化过程中在运行时建立的。

如果值为 `tozero`、`negative` 或 `positive`，该选项在程序开始执行时将舍入方向相应地设置为舍入为零、舍入为负无穷大或舍入为正无穷大。如果未指定 `-fround`，则将 `-fround=nearest` 用作缺省值，舍入方向是舍入为最接近的值。其含义与 `ieee_flags` 函数相同。（请参见《*Fortran* 编程指南》的“浮点运算”一章。）

3.4.35 –fserialio

指定程序不能一次在多个线程中执行 I/O 的链接选项。它允许 Fortran I/O 语句在不执行同步的情况下执行，以避免出现争用情况。此选项仅应在创建可执行程序时指定。在创建共享对象库时不应指定，在程序包含使用早于 Sun Forte 7 发行版的 Sun f77 版本编译的代码时也不应指定。

3.4.36 –fsimple[={1|2|0}]

选择浮点优化首选项。

允许优化器作出有关浮点运算的简化假定。（请参见《*Fortran* 编程指南》的“浮点运算”一章。）

为了获得一致的结果，请使用同一 `-fsimple` 选项编译程序的所有单元。

缺省值为：

- 如果不使用 `-fsimple` 标志，则编译器缺省为 `-fsimple=0`
- 如果使用不带值的 `-fsimple`，则编译器使用 `-fsimple=1`

各种浮点简化级别如下：

`–fsimple=0` 不允许简化假定。保持严格的 IEEE 754 一致性。

`–fsimple=1` 允许适度的简化。结果代码未严格符合 IEEE 754。

在 `-fsimple=1` 的情况下，优化器可假定：

- 在进程初始化之后，IEEE 754 缺省舍入/捕获模式不发生改变。
- 可以删除不生成可见结果（潜在的浮点异常除外）的计算。
- 以无穷大或 NaN（“不是数”）为操作数的计算不需要将 NaN 传播到其结果；例如，`x*0` 可以由 `0` 替换。

- 计算过程不依赖于零的符号。

如果使用 `-fsimple=1`，则不允许优化器进行完全优化，而不考虑舍入或异常。特别是，在运行时舍入模式包含常量的情况下，浮点计算不能由产生不同结果的计算替换。

`–fsimple=2`

除 `–fsimple=1` 外，还允许主动浮点优化。这会导致某些程序因表达式求值方式的变化而生成不同的数值结果。尤其是，使用 `-fsimple=2` 可能会违反如下 Fortran 标准规则：要求编译器用显式圆括号将子表达式括起来以控制表达式求值顺序。对于依赖此规则的程序，这会导致数值舍入差异。

例如，如果使用 `-fsimple=2`，编译器可能将 `C-(A-B)` 计算为 `(C-A)+B`，从而违反了有关显式圆括号的标准规则（如果生成的代码已更好地进行了优化）。编译器还可能将 *x/y* 的重复计算替换为 *x*z*，其中的 *z=1/y* 计算一次并暂时保存，以消除成本较高的除法运算。

对于依赖浮点运算的特定属性的程序，不得使用 `-fsimple=2` 进行编译。

即使使用 `-fsimple=2`，也仍然不允许优化器在程序中引入浮点异常，如果不在这样的程序中引入浮点异常，该程序将不生成任何异常。

`–fast` 会选择 `-fsimple=2`。

3.4.37 –fstore

(x86) 强制浮点表达式的精度。

对于赋值语句，此选项将所有浮点表达式强制为目标变量的精度。这是缺省值。但是，`-fast` 选项包括可用来禁用此选项的 `-nofstore`。`-fast` 后跟 `-fstore` 可以重新打开此选项。

3.4.38 –ftrap=*t*

设置在启动时有效的浮点捕获模式。

t 是一个逗号分隔列表，它包含以下项中的一个或多个：

`%all`, `%none`, `common`, `[no%]invalid`, `[no%]overflow`, `[no%]underflow`, `[no%]division`, `[no%]inexact`.

`-ftrap=common` 是 `-ftrap=invalid,overflow,division` 的宏。

f95 的缺省值为 -ftrap=common。这与 C 和 C++ 编译器的缺省值 (-ftrap=none) 不同。

设置在启动时有效的 IEEE 754 捕获模式，但不安装 SIGFPE 处理程序。可以使用 ieee_handler(3M) 或 fex_set_handling(3M) 启用陷阱并同时安装 SIGFPE 处理程序。如果指定多个值，则按从左到右顺序处理列表。按照定义，常见异常包括无效、被零除和溢出。

示例： -ftrap=%all,no%inexact 表示设置除 inexact 以外的所有陷阱。

-ftrap=*t* 的含义与 ieee_flags() 基本相同，不同之处是：

- %all 打开所有捕获模式，并会导致捕获伪异常和预期异常。请改用 common。
- %none 关闭所有捕获模式。
- no% 前缀关闭该特定捕获模式。

主程序必须使用此选项进行编译才能有效。

有关详细信息，请参见《*Fortran* 编程指南》的“浮点运算”一章。

3.4.39 –G

生成动态共享库，而不是生成可执行文件。

指示链接程序生成共享动态库。如果不使用 -G，则链接程序生成可执行文件。如果使用 -G，它将生成动态库。将 -o 与 -G 一起使用可以指定要写入的文件的名称。有关详细信息，请参见《*Fortran* 编程指南》的“库”一章。

3.4.40 -g

请参见 -g[*n*]。

3.4.41 –g[*n*]

针对调试和性能分析进行调试。

生成其他符号表信息，以便使用 dbx(1) 调试实用程序进行调试，并使用性能分析器进行性能分析。

虽然在不指定 -g 的情况下也可以进行一些调试，但是 dbx 和 debugger 的完整功能只供那些使用 -g 编译的编译单元使用。

与 -g 一起指定的其他选项的某些功能可能是有限的。有关详细信息，请参见 dbx 文档。

要使用性能分析器的完整功能，请使用 -g 进行编译。虽然某些性能分析功能不需要使用 -g，但必须使用 -g 进行编译，以便查看注释的源代码、部分函数级别信息以及编译器注释消息。（请参见 analyzer(1) 手册页和*Solaris Studio* 性能分析器手册。）

使用 -g 生成的注释性消息说明编译器在编译程序时进行的优化和变换。通过 er_src(1) 命令，可以显示与源代码交错的消息。

请注意，仅当编译器实际执行了优化时，才会出现注释性消息。如果请求高优化级别（如使用 -xO4 或 -fast），则更有可能看到注释性消息。

-g 作为宏实施，扩展到多个其他更原始的选项。有关扩展的详细信息，请参见 -xdebuginfo。

-g	生成标准调试信息。
-gnone	不生成任何调试信息。这是缺省值。
-g1	生成文件和行号以及在事后调试期间视为至关重要的简单参数信息。
-g2	与 -g 相同。
-g3	生成附加调试信息，当前只包括宏定义信息。与仅使用 -g 相比，此附加信息会增大生成的 .o 和可执行文件中调试信息的大小。

3.4.42 –h*name*

指定所生成的动态共享库的名称。

此选项将被传递给链接程序。有关详细信息，请参见 Oracle Solaris《链接程序和库指南》以及《*Fortran* 编程指南》的“库”一章。

-h*name* 选项将名称 *name* 记录到共享动态库中，作为库的内部名称创建。-h 和 *name* 之间的空格是可选的（除非库名称是 elp，此时要求使用空格）。通常，*name* 必须与跟在 -o 后面的内容相同。如果不同时指定 -G，则使用此选项是无意义的。

如果不使用 -h*name* 选项，则在库文件中不记录内部名称。

如果库具有内部名称，则每当运行引用该库的可执行程序时，运行时链接程序将在其搜索的任何路径中搜索具有相同内部名称的库。在指定了内部名称的情况下，在运行时链接过程中搜索库更为灵活。此选项还可用于指定共享库的版本。

如果没有共享库的内部名称，则链接程序将改用共享库文件的特定路径。

3.4.43 –help

显示编译器选项的摘要列表。

另请参见第 3.4.125 节 “–xhelp=flags” [109]。

3.4.44 –I*path*

将 *path* 添加到 INCLUDE 文件搜索路径。

在 *INCLUDE* 文件搜索路径的开始处插入目录路径 path。在 -I 和 *path* 之间允许有空格。无效目录将被忽略，并且不显示警告消息。

include 文件搜索路径是在其中搜索 INCLUDE 文件（出现在预处理程序 #include 指令或 Fortran INCLUDE 语句中的文件名）的目录的列表。

搜索路径还用于查找 MODULE 文件。

示例：在 /usr/app/include 中搜索 INCLUDE 文件：

```
demo% f95 -I/usr/app/include growth.F
```

在命令行上可以出现多个 -I*path* 选项。每个选项都添加到搜索路径列表的顶部（搜索的第一个路径）。

INCLUDE 或 #include 的相对路径的搜索顺序如下：

1. 包含源文件的目录
2. 在 -I 选项中指定的目录
3. 编译器内部缺省列表中的目录
4. /usr/include/

要调用预处理程序，您必须使用 .F、.F90、.F95 或 .F03 后缀来编译源文件。

3.4.45 -i8

（没有 -i8 选项。）

使用 –xtypemap=integer:64 指定该编译器的 8 字节 INTEGER。

3.4.46 –inline=[%auto][[,][no%]*f1*,...[no%]*fn*]

启用或禁用指定例程的内联。

请求优化器对出现在函数和子例程名称列表（用逗号分隔）中的用户编写例程进行内联。在例程名之前加上 no% 可禁用对该例程的内联。

内联是一种优化方法，编译器可以通过该方法有效地将子程序引用（如 CALL 或函数调用）替换为实际的子程序代码本身。内联通常为优化器提供更多生成高效代码的机会。

指定 %auto 可以在优化级别 -O4 或 -O5 上启用自动内联。如果使用 -inline 指定了显式内联，则这些优化级别的自动内联会正常关闭。

如果在未指定任何函数或 %auto 的情况下指定了 -xinline=，则表示不内联源文件中的任何例程。

示例：对 xbar、zbar 和 vpoint 例程进行内联：

```
demo% f95 -O3 -inline=xbar,zbar,vpoint *.f
```

下面是一些限制；不发出任何警告：

- 必须在 -O3 或更高级别进行优化。
- 例程的源代码必须位于所编译的文件中，除非还指定了 -xipo 或 –xcrossfile。
- 编译器确定实际内联是否有利和安全。

-inline 与 -O4 一起使用可禁用编译器通常执行的自动内联，除非还指定了 %auto。如果使用 -O4，则编译器通常会尝试对用户编写的所有适当的子例程和函数进行内联。-inline 与 -O4 一起使用可能会降低性能，因为优化器只能对列表中的那些例程进行内联。在这种情况下，请使用 %auto 子选项启用 -O4 和 -O5 级别的自动内联。

```
demo% f95 -O4 -inline=%auto,no%zpoint *.f
```

在上例中，用户在禁止对编译器可能尝试的例程 zpoint() 进行任何可能的内联的同时，还启用了 -O4 级别的自动内联。

3.4.47 –iorounding[={compatible|processor-defined}]

为格式化输入/输出设置浮点舍入模式。

以全局方式为所有的格式化输入/输出运算设置 ROUND= 说明符。

如果使用 -iorounding=compatible，数据转换后的值是与两个最接近表示更接近的值，如果值正好在两者中间，则是离 0 远的值。

如果使用 -iorounding=processor-defined，则舍入模式是处理器的缺省模式。在未指定 -iorounding 时，这是缺省值。

3.4.48 –keepmod[={yes|no}]

如果模块文件存在且其内容未由最新编译更改，则即使编译应该创建同名的新模块文件，也不会替换该文件。由于模块文件的内容未由编译更改，因此唯一的作用是，将保留现有模块文件的时间戳。

如果未指定 -keepmod，则缺省为 -keepmod=yes。请注意，该缺省值不同于以前的 Oracle Solaris Studio Fortran 发行版。

该选项最好与 -xM 编译选项生成的依赖项一起使用。通过在模块文件内容未更改时保留其时间戳，该选项可防止对依赖于该模块文件的源文件进行级联编译。这对于增量生成非常有用，可显著缩短生成时间。

当该选项与用户指定的依赖项一起使用且用户具有关于如何创建与相应源文件存在依赖性的模块的显式生成规则时，该选项会导致重新编译多次源文件，即使源文件只因模块文件的时间戳过期修改过一次。

3.4.49 –keeptmp

保留在编译期间所创建的临时文件。

3.4.50 –Kpic

（已过时）与 -pic 等效。

3.4.51 –KPIC

（已过时）与 -PIC 等效。

3.4.52 –L*path*

将 *path* 添加到要在其中搜索库的目录路径的列表中。

将 *path* 添加到对象库搜索目录列表的前面。`-L` 和 *path* 之间的空格是可选的。此选项将传递给链接程序。另请参见第 3.4.53 节 “–l*x*” [70]。

在生成可执行文件时，`ld`(1) 在 *path* 中搜索归档库（`.a` 文件）和共享库（`.so` 文件）。`ld` 在搜索缺省目录之前搜索 *path*。（有关库搜索顺序的信息，请参见《*Fortran* 编程指南》的“库”一章。）有关 `LD_LIBRARY_PATH` 和 `-L`*path* 之间的相对顺序，请参见 `ld`(1)。

注 - 使用 `-L path` 指定 `/usr/lib` 或 */usr/ccs/lib* 可能会阻止链接未绑定的 `libm`。缺省情况下，将搜索这些目录。

示例： 使用 `-L`*path* 指定库搜索目录：

```
demo% f95 -L./dir1 -L./dir2 any.f
```

3.4.53 –l*x*

将库 `lib`*x*`.a` 添加到链接程序的搜索库列表中。

将 `-l`*x* 传递给链接程序，以指定供 `ld` 在其中搜索未解析引用的其他库。`ld` 与对象库 `lib`*x* 链接。如果共享库 `lib`*x*`.so` 可用（且未指定 `-Bstatic` 或 `-dn`），则 `ld` 使用它，否则，`ld` 使用静态库 `lib`*x*`.a`。如果它使用共享库，则将该名称生成到 `a.out` 中。在 `-l` 和 *x* 字符串之间不允许有空格。

示例： 与库 `libVZY` 进行链接：

```
demo% f95 any.f -lVZY
```

再次使用 `-l`*x* 以便与更多的库链接。

示例： 与库 `liby` 和 `libz` 进行链接：

```
demo% f95 any.f -ly -lz
```

有关库搜索路径和搜索顺序的信息，另请参见《*Fortran* 编程指南》的“库”一章。

3.4.54 –libmil

内联所选的用于优化的 `libm` 库例程。

某些 libm 库例程有内联模板。此选项会选择那些为当前使用的浮点选项和平台生成速度最快的可执行文件的内联模板。

有关更多信息，请参见 libm_single(3F) 和 libm_double(3F) 手册页。

3.4.55 -library=sunperf

与 Oracle Solaris Studio 提供的性能库进行链接。（请参见《*Sun Performance Library User's Guide*》。）

3.4.56 –loopinfo

显示循环并行化结果。

显示哪些循环通过 –autopar 选项实现了并行化，而哪些未实现并行化。

–loopinfo 显示有关标准错误的消息的列表：

```
demo% f95 -c -fast -autopar -loopinfo shalow.f
...
"shalow.f", line 172: PARALLELIZED, and serial version generated
"shalow.f", line 173: not parallelized, not profitable
"shalow.f", line 181: PARALLELIZED, fused
"shalow.f", line 182: not parallelized, not profitable
...
...etc
```

3.4.57 –M*path*

指定 MODULE 目录、归档或文件。

在路径中查找当前编译中引用的 Fortran 模块。在当前目录之外的目录中搜索此路径。

path 可以指定目录、预编译模块文件的 .a 归档文件，或 .mod 预编译模块文件。编译器通过检查文件的内容来确定其类型。

要在其中搜索模块的 .a 归档文件必须在 -M 选项标志上显式指定。缺省情况下，编译器不搜索归档文件。

只搜索与出现在 USE 语句中的 MODULE 名称同名的 .mod 文件。例如，语句 USE ME 使编译器仅查找模块文件 me.mod。

搜索模块时，编译器为在其中写入模块文件的目录指定更高的优先级。这是由 `-moddir` 编译器选项或 `MODDIR` 环境变量控制的。如果上述两者都未指定，则缺省写入目录为当前目录。如果两者均已指定，则写入目录是 `-moddir` 标志指定的路径。

这意味着，如果只出现了 `-M` 标志，则先在当前目录中搜索模块，然后再在 `-M` 标志上列出的任何对象中进行搜索。要模拟以前发行版的行为，请使用：

```
-moddir=empty-dir -Mdir -M
```

其中 *empty-dir* 是空目录的路径。

如果在所搜索的任何其他位置均未找到模块文件，则将在 `-I` *path* 所指定的目录中搜索这些文件。

`-M` 和路径之间可以有空格例如，`-M /home/siri/PK15/Modules`

在 Solaris 上，如果路径标识一个非归档文件或模块文件的常规文件，则编译器会将该选项传递给链接程序 `ld`，链接程序会将该选项作为链接程序映射文件。与 C 和 C++ 编译器类似，此功能作为公用选项提供。

有关 Fortran 中模块的更多信息，请参见第 4.9 节 “模块文件” [168]。

3.4.58 –m32 | –m64

为编译的二进制对象指定内存模型。

使用 `-m32` 创建 32 位可执行文件和共享库。使用 `-m64` 创建 64 位可执行文件和共享库。

在所有 Solaris 平台和不支持 64 位的 Linux 平台上，ILP32 内存模型（32 位 `int`、`long`、`pointer` 数据类型）为缺省值。在启用了 64 位的 Linux 平台上缺省为 LP64 内存模型（64 位 long 和指针数据类型）。`-m64` 仅允许在启用了 LP64 模型的平台上使用。

使用 `-m32` 编译的对象文件或库不能同使用 `-m64` 编译的对象文件或库链接。

在 x64 平台上编译具有大量静态数据的应用程序时，可能还需要使用 `-m64, -xmodel=medium`。

请注意，部分 Linux 平台不支持中等模型。

注意，在以前的编译器发行版中，通过选择带有 `-xarch` 的指令集来实现内存模型 ILP32 或 LP64 。从 Solaris Studio 12 编译器开始，就不再是这样了。在大多数平台上，仅需向命令行添加 `-m64` 即可创建 64 位对象。

在 Solaris 上，缺省为 `-m32`。在支持 64 位程序的 Linux 系统上，缺省为 `-m64 -xarch=sse2`。

3.4.59 –moddir=*path*

指定编译器将编译的 .mod MODULE 文件写入的位置。

编译器会将它编译的 .mod MODULE 信息文件写入由 *path* 指定的目录。也可以使用 MODDIR 环境变量指定目录路径。如果同时使用这两种方法指定了目录路径，则此选项标志优先。

编译器将当前目录用作写入 .mod 文件的缺省目录。

有关 Fortran 中模块的更多信息，请参见第 4.9 节 “模块文件” [168]。

3.4.60 -mt[={yes|no}]

使用此选项，可以通过 Oracle Solaris 线程或 POSIX 线程 API 编译和链接多线程代码。-mt=yes 选项确保按正确的顺序链接库。

此选项将 -D_REENTRANT 传递给预处理程序。

在 Linux 平台上，只有 POSIX 线程 API 可用。（Linux 平台上没有 libthread）。因此，Linux 平台上的 -mt=yes 会添加 -lpthread，而不是 -lthread。要在 Linux 平台上使用 POSIX 线程，请使用 -mt 进行编译。

请注意，当使用 -G 进行编译时，-lthread 和 -lpthread 均不会自动包括在 -mt=yes 内。生成共享库时，您需要显式列出这些库。

-xopenmp 选项（用于使用 OpenMP 共享内存并行化 API）自动包括 -mt=yes。

如果使用 -mt=yes 进行编译并在单独的步骤中进行链接，则除了在编译步骤中外，还必须在链接步骤中使用 -mt=yes 选项。如果使用 -mt=yes 编译和链接一个转换单元，则必须使用 -mt=yes 编译和链接该程序的所有单元。

-mt=yes 是编译器的缺省行为。如果不需要此行为，请使用 -mt=no 进行编译。

选项 -mt 等效于 -mt=yes。

3.4.61 –native

（已过时）优化主机系统的性能。

此选项与首选设置 -xtarget=native 等效。-fast 选项设置 -xtarget=native。

3.4.62 –noautopar

禁用由先前出现在命令行上的 -autopar 调用的自动并行化。

3.4.63 –nodepend

取消先前出现在命令行上的任何 -depend。-depend=no 的使用优先级高于 -nodepend。

3.4.64 -nofstore

(x86) 取消命令行上的 -fstore。

编译器的缺省值为 -fstore。-fast 包括 -nofstore。

3.4.65 –nolib

禁用与系统库链接。

不自动与任何系统库或语言库链接；也就是说，不将任何缺省的 -l*x* 选项传递给 ld。正常行为是将系统库自动链接到可执行文件，而无需用户在命令行上指定它们。

使用 -nolib 选项，可以更轻松地静态链接其中的一个库。最终执行需要系统库和语言库。手动链接它们是您的责任。通过此选项，您可以完全控制与库的链接。

将 libm（静态）和 libc（动态）与 f95 链接：

```
demo% f95 -nolib any.f95 -Bstatic -lm -Bdynamic -lc
```

-l*x* 选项的顺序是很重要的。请遵循示例所示的顺序。

3.4.66 –nolibmil

取消命令行上的 -libmil。

在 *-fast* 选项之后使用此选项，可禁用 `libm` 数学例程的内联：

```
demo% f95 -fast -nolibmil …
```

3.4.67 –noreduction

禁用命令行上的 `-reduction`。

此选项禁用 `-reduction`。

3.4.68 –norunpath

不会将运行时共享库搜索路径生成到可执行文件中。

编译器通常将一个路径生成到可执行文件中，从而告知运行时链接程序查找所需共享库的位置。该路径取决于具体的安装。`-norunpath` 选项阻止将该路径生成到可执行文件中。

如果已将库安装在一些非标准位置，而且您不希望在另一位置运行可执行文件时让加载器搜索这些路径，则此选项是很有用的。请与 `-R`*paths* 进行比较。

有关更多信息，请参见《*Fortran* 编程指南》的“库”一章。

3.4.69 –o[*n*]

指定优化级别。

n 可以是 `1`、`2`、`3`、`4` 或 `5`。在 `-O` 和 *n* 之间不允许有空格。

如果未指定 `-O[`*n*`]`，则仅执行非常基本级别的优化，即限于局部公共子表达式消除和无用代码分析。与不使用优化相比，使用优化级别进行优化可以大大提高程序的性能。对于大多数程序，建议使用 `-O`（它设置 `-O3`）或 `-fast`（它设置 `-O5`）。

每个 `-O`*n* 级别的优化都包括在低于它的级别上执行的优化。通常，编译程序时使用的优化级别越高，获得的运行时性能也越高。但是，优化级别越高，编译时间会越长，可执行文件也越大。

使用 `-g` 进行调试不会抑制 `-O`*n*，但是 `-O`*n* 在某些方面会限制 `-g`；请参见 `dbx` 文档。

-O3 和 -O4 选项降低调试的效用，这样您无法从 dbx 显示变量，但仍可以使用 dbx where 命令获取符号回扫。

如果优化器内存不足，则它尝试在较低优化级别上再次进行，即继续在原始级别上对后续例程进行编译。

有关优化的详细信息，请参见《*Fortran* 编程指南》的“性能分析”和“性能与优化”这两章。

–0 此选项与 -O3 等效。

–O1 提供最少的语句级优化。

如果更高的级别会导致编译时间过长，或者超过了可用交换空间，请使用此选项。

–O2 启用基本块级别的优化。

通常，此级别产生的代码大小是最小的。（另请参见 -xspace。）

–O3 的使用应优先于 -O2，除非 -O3 导致编译时间过长、超过交换空间或生成过大的可执行文件。

–O3 在函数级别上增加循环解开和全局优化。自动添加 -depend。

通常，-O3 生成的可执行文件较大。

–O4 增加包含在同一文件中的例程的自动内联。

通常，-O4 生成的可执行文件较大（因为进行了内联）。

-g 选项会禁止上述的 -O4 自动内联。–xcrossfile 可增加使用 -O4 情况下的内联范围。

–O5 尝试主动优化。

仅适用于程序中占用计算时间最多的小段代码。-O5 的优化算法要花费更多的编译时间，而且在用于过大的源程序代码段时会降低性能。

如果使用分析反馈进行优化，则此级别上的优化更有可能提高性能。请参见 -xprofile=*p*。

3.4.70 –o *filename*

指定要写入的可执行文件的名称。

必须使用空格来分隔 -o 和文件名。如果没有此选项，缺省值是将可执行文件写入 a.out。在与 -c 一起使用时，-o 指定目标 .o 对象文件；在与 -G 一起使用时，它指定目标 .so 库文件。

3.4.71 –onetrip

启用单行程 DO 循环。

编译 DO 循环以便至少执行它们一次。如果上限小于下限，则在标准 Fortran 中根本不执行 DO 循环；这一点与 Fortran 的某些传统实现是不同的。

3.4.72 –openmp

与 -xopenmp 等效。

3.4.73 –p

（已过时）为使用 prof 分析器进行分析而编译。

准备对象文件以进行分析，请参见 prof (1)。如果在不同的步骤中编译和链接，并且使用 -p 选项编译，请确保使用 -p 选项链接。将 -p 与 prof 结合使用主要是为了与老系统兼容。使用 gprof 进行 -pg 分析可能是更好的方式。有关详细信息，请参见《*Fortran* 编程指南》的“性能分析”一章。

3.4.74 –pad[=*p*]

插入填充以便提高高速缓存的使用效率。

如果数组或字符变量是静态的局部数组或变量且未初始化，或者位于通用块中，则此选项在数组之间或字符变量之间插入填充。额外填充将数据定位以便更好地利用高速缓存。在任意一种情况下，数组或字符变量都不能是等效的。

如果 *p* 存在，则 p 必须为 %none，或者为 local 或 common 之一（或两者）：

local	在邻近的局部变量之间添加填充。
common	在通用块中的变量之间增加填充。
%none	不增加填充。（编译器缺省值。）

如果同时指定了 local 和 common，则它们可以以任意顺序显示。

-pad 的缺省值：

- 缺省情况下编译器不进行填充。
- 如果指定了 -pad 但不带值，则它与 -pad=local,common 等效。

-pad[=*p*] 选项适用于满足以下条件的项：

- 项是数组或字符变量
- 项是静态本地的或处于通用块中

有关本地或静态变量的定义，请参见第 3.4.88 节 “–stackvar” [83]。

程序必须符合以下限制：

- 数组或字符串都不是等效的
- 如果为了对引用通用块的某个文件进行编译而指定了 -pad=common，则在对引用该通用块的所有文件进行编译时也必须指定它。此选项更改通用块内变量的间距。如果一个程序单元是使用该选项编译的，而另一个程序单元不是用该选项编译的，则在应该引用通用块中的同一位置时，可能会引用不同的位置。
- 如果指定了 -pad=common，不同程序单元中的通用块变量的声明除变量名称之外必须相同。通用块变量之间插入的填充量取决于这些变量的声明。如果变量在不同程序单元中的大小或等级不同，即使是在同一文件内，变量的位置也可能不同。
- 如果指定了 -pad=common，则用警告消息标记那些涉及通用块变量的 EQUIVALENCE 声明，而且不填充该块。
- 如果指定了 -pad=common，请避免通用块中出现索引超出数组边界的情况。如果更改已填充通用块中邻近数据的位置，将导致过度索引 (overindexing) 以不可预知的方式失败。

使用 -pad 时，程序员应确保以一致的方式编译通用块。如果不同程序单元中的 common 块没有一致地使用 -pad=common 进行编译，则会导致错误。如果同名的 common 块在不同的程序单元中具有不同的长度，则将报告正在使用 -Xlist 进行编译。

3.4.75 –pg

为使用 gprof 分析器进行分析而编译。（-xpg 与 -pg 等效）

以 -p 的方式编译自分析代码，但调用一种运行时记录机制，该机制可保存更广泛的统计信息并在程序正常终止时生成 gmon.out 文件。通过运行 gprof 生成执行分析概要。有关详细信息，请参见 gprof(1) 手册页和《*Fortran* 编程指南》。

库选项必须跟在源文件和 .o 文件之后（–pg 库是静态的）。

注 - 如果指定了 -pg，则使用 -xprofile 进行编译并无优势。这两个功能中的任一功能不会准备或使用由另一个功能提供的数据。

使用 prof(1) 或 gprof(1)（在 64 位 Solaris 平台上）或者仅使用 gprof（在 32 位 Solaris 平台上）生成的分析中包括大致的用户 CPU 时间。这些时间来自主可执行文件中的例程以及共享库中例程（链接可执行文件时将共享库指定为链接程序参数）的 PC 示例数据（请参见 pcsample(2)）。其他共享库（在进程启动后使用 dlopen(3DL) 打开的库）不进行分析。

在 32 位 Oracle Solaris 系统中，使用 prof(1) 生成的分析仅限于可执行文件中的例程。通过使用 -pg 链接可执行文件并使用 gprof(1)，可以对 32 位共享库进行分析。

Solaris 10 软件不包括使用 -p 编译的系统库。因此，在 Solaris 10 平台上收集的分析不包含系统库例程的调用计数。

不应当使用编译器选项 -p、-pg 或 -xpg 来编译多线程程序，因为这些选项的运行时支持不是线程安全的。如果利用这些选项来编译使用多个线程的程序，则可能会在运行时产生无效结果或段故障。

为执行 gprof 分析而使用 -xpg 编译的二进制代码不应与 binopt(1) 一起使用，因为这两者不能兼容，结合使用会导致内部错误。

如果在不同的步骤中进行编译和链接，并使用 -pg 进行编译，请确保使用 -pg 进行链接。

在 x86 系统上，-pg 与 -xregs=frameptr 不兼容，这两个选项不应一起使用。还请注意，-fast 中包括 -xregs=frameptr。

3.4.76 –pic

为共享库编译与位置无关的代码。

在 SPARC 上，–pic 与 -xcode=pic13 等效。有关与位置无关的代码的更多信息，请参见第 3.4.116 节 “-xcode[=v]” [101]。

在 x86 上，生成与位置无关的代码。生成共享库时使用该选项编译源文件。对全局数据的每个引用都生成为全局偏移表中指针的非关联化。每个函数调用都通过过程链接表在 pc 相对地址模式中生成。

3.4.77 –PIC

使用 32 位地址编译与位置无关的代码。

在 SPARC 上，–PIC 与 -xcode=pic32 等效。有关与位置无关的代码的更多信息，请参见第 3.4.116 节 “-xcode[=*v*]” [101]。

在 x86 上，–PIC 与 –pic 等效。

3.4.78 –preserve_argvalues[=simple|none|complete]

(x86) 在堆栈中保存基于寄存器的函数参数的副本。

如果指定了 none 或者如果未在命令行上指定 -preserve_argvalues 选项，则编译器行为像往常一样。

指定 simple 时，最多保存六个整数参数。

指定 complete 时，堆栈跟踪中所有函数参数的值按正确顺序显示给用户。

对于形式参数的赋值，值在函数生命周期内不会更新。

3.4.79 –Qoption *pr ls*

将子选项列表 *ls* 传递到编译阶段 *pr*。

必须使用空格来分隔 Qoption、*pr* 和 *ls*。Q 可以是大写的，也可以是小写的。该列表是一个逗号分隔的子选项列表，其中不包含空格。每个子选项都必须适合于该程序阶段，而且可以以负号开头。

提供此选项主要是为了供支持人员调试编译器的内部。使用 LD_OPTIONS 环境变量可以将选项传递给链接程序。请参见《*Fortran* 编程指南》中有关链接和库的章节。

3.4.80 –qp

与 -p 等效。

3.4.81 –R *ls*

将动态库搜索路径生成到可执行文件中。

如果使用此选项，则链接程序 ld(1) 将动态库搜索路径列表存储到可执行文件中。

ls 是一个用冒号分隔的库搜索路径目录列表。-R 和 *ls* 之间的空格是可选的。

此选项的多个实例并置在一起，各个列表由冒号分隔。

该列表由运行时链接程序 ld.so 在运行时使用。在运行时，将扫描所列出路径中的动态库以满足任何未解析的引用。

使用此选项，用户可以在不使用特殊路径选项查找所需动态库的情况下运行现有的可执行文件。

使用 -R*paths* 生成可执行文件，可将目录路径添加到缺省路径（始终最后搜索它）。

有关更多信息，请参见《*Fortran* 编程指南》的“库”一章以及 Oracle Solaris《链接程序和库指南》。

3.4.82 –r8const

将单精度常量提升为 REAL*8 常量。

所有单精度 REAL 常量都将提升为 REAL*8 常量。双精度 (REAL*8) 常量保持不变。此选项仅适用于常数。要同时提升常量和变量，请参见第 3.4.183 节 “–xtypemap=*spec*” [142]。

请小心使用此选项标志。当使用已提升为 REAL*8 常量的 REAL*4 常量调用需要 REAL*4 参数的子例程或函数时，此选项标志可能会导致接口问题。此选项标志还可能导致那些读取无格式数据文件的程序出现问题，这些文件是由无格式写入功能使用 I/O 列表上的 REAL*4 常量写入的。

3.4.83 –recl=a[,b]

设置缺省输出记录长度。

为预连接单元输出（标准输出）和/或错误（标准错误）设置缺省记录长度（单位为字符）。此选项必须按下面的格式之一来指定：

- –recl=out:*N*
- –recl=error:*N*
- –recl=out:*N1*,error:*N2*
- –recl=error:*N1*,out:*N2*

- `–recl=all:`*N*

其中，*N*、*N1*、*N2* 为 72 到 2147483646 之间的正整数值。`out` 指标准输出，`error` 指标准错误，`all` 为这两者设置缺省记录长度。缺省值为 `-recl=all:80`。仅在当前编译的程序有 Fortran 主程序时，此选项才有效。

3.4.84 –reduction

识别循环中的约简操作。

在自动并行化期间分析循环以了解约简操作。约简操作可能存在舍入误差。

约简操作将数组元素累加为单个标量值。例如，对向量元素求和是典型的约简操作。虽然这些操作违反了可并行化标准，但是编译器可以识别它们，并在指定了 `-reduction` 时作为特殊情况对它们进行并行化。有关编译器可识别的约简操作的信息，请参见《*Fortran* 编程指南》的“并行化”一章。

此选项只能与自动并行化选项 `–autopar` 一起使用。否则，它将被忽略。对于约简操作，不分析显式并行化的循环。

3.4.85 –S

编译并仅生成汇编代码。

编译指定的程序，并在后缀为 `.s` 的相应文件中保留汇编语言输出，而不创建 `.o` 文件。

3.4.86 –s

将符号表与可执行文件分离。

此选项使可执行文件变得更小，并使逆向工程的实施更困难。但是，此选项禁止使用 `dbx` 或其他工具进行调试，而且覆盖 `-g`。

3.4.87 –silent

（已过时）禁止显示编译器消息。

通常，f95 编译器在编译过程中不发出除错误诊断之外的消息。提供此选项标志是为了与传统的 f77 编译器兼容；除非与 -f77 兼容性标志一起使用，否则使用它是多余的。

3.4.88 –stackvar

尽可能在堆栈上分配局部变量。

此选项使编写递归代码和可重入代码更容易，并在并行化循环时为优化器提供更多自由。

建议将 -stackvar 与任何并行化选项一起使用。

局部变量是除哑元参数、COMMON 变量、从外部作用域继承的变量或可通过 USE 语句访问的模块变量之外的变量。

在 -stackvar 有效的情况下，局部变量是在堆栈上分配的，除非它们具有属性 SAVE 或 STATIC。请注意，显式初始化的变量是使用 SAVE 属性隐式声明的。缺省情况下，未显式初始化但其某些组件已初始化的结构变量，没有使用 SAVE 隐式声明。此外，与具有 SAVE 或 STATIC 属性的变量等效的变量，也隐式具有 SAVE 或 STATIC。

以静态方式分配的变量隐式初始化为零，除非程序为其显式指定了初始值。在堆栈上分配的变量并未隐式初始化（缺省情况下可进行初始化的结构变量的组件除外）。

使用 -stackvar 将大数组放在堆栈上可以使堆栈溢出，从而导致段故障。此时可能需要增加堆栈大小。

执行程序的初始线程有一个主堆栈，而多线程程序的每个从线程都有自己的线程堆栈。

对于从线程，在 32 位系统上线程堆栈的缺省大小为 4 兆字节，在 64 位系统上的缺省大小为 8 兆字节。limit 命令（不带参数）显示当前的主堆栈大小。如果使用 -stackvar 时出现段故障，请尝试增加主堆栈和线程堆栈的大小。

示例： 显示当前的主堆栈大小：

```
demo% limit
cputime         unlimited
filesize        unlimited
datasize        523256 kbytes
stacksize       8192 kbytes      <——
coredumpsize    unlimited
descriptors     64
memorysize      unlimited
demo%
```

示例：将主堆栈大小设置为 64 兆字节：

```
demo% limit stacksize 65536
```

通过设置 STACKSIZE 或 OMP_STACKSIZE 环境变量，可以设置每个从线程使用的线程堆栈大小：有关这些环境变量的更多信息，请参见《*OpenMP API* 用户指南》。

使用 -xcheck=stkovf 进行编译，可启用堆栈溢出情况的运行时检查。有关更多信息，请参见 -xcheck。

3.4.89 –stop_status[={yes|no}]

允许 STOP 语句返回整数状态值。

缺省值为 -stop_status=no。

如果使用 -stop_status=yes，则 STOP 语句可以包含整型常量。在程序终止时，该值将传递到环境：

```
STOP 123
```

该值的范围必须介于 0 到 255 之间。大于这个范围的值将被截断，并且会发出运行时消息。请注意，尽管将发出编译器警告消息，但是仍将接受

STOP "*stop string*'

并将状态值 0 返回到环境。

环境状态变量是 $status（对于 C shell csh）和 $?（对于 Bourne shell sh 和 Korn shell ksh）。

3.4.90 –temp=*dir*

为临时文件定义目录。

将编译器所用临时文件的目录设置为 *dir*。在此选项字符串中不允许有空格。如果不使用此选项，则将这些文件放在 /tmp 目录中。

此选项优先于 TMPDIR 环境变量的值。

3.4.91 –time

每个编译阶段所用时间。

将显示每次编译过程所用的时间和资源。

3.4.92 –traceback[={%none|common|*signals_list*}]

如果执行中出现严重错误，将发出堆栈跟踪。

当程序生成某些信号时，-traceback 选项会导致可执行文件向 stderr 发出堆栈跟踪、转储信息并退出。如果多个线程都生成一个信号，则只为第一个生成堆栈跟踪。

要使用回溯，请在链接时将 -traceback 选项添加到编译器命令行中。编译时也接受该选项，除非生成可执行二进制文件，否则将忽略此选项。使用 -traceback 和 -G 创建共享库是个错误。

表 3-9 -traceback 选项

选项	含义
common	指定应在出现以下任意一组常见信号时发出堆栈跟踪：sigill、sigfpe、sigbus、sigsegv 或 sigabrt。
signals_list	指定应生成堆栈跟踪的信号名称的逗号分隔列表，采用小写形式。可以捕捉以下信号（导致生成信息转储文件的信号）：sigquit、sigill、sigtrap、sigabrt、sigemt、sigfpe、sigbus、sigsegv、sigsys、sigxcpu、sigxfsz。 可以在上述任一信号前加上 no% 以禁用信号缓存。 例如：如果发生 sigsegv 或 sigfpe，-traceback=sigsegv,sigfpe 将生成堆栈跟踪和信息转储。
%none 或 none	禁用回溯

如果不指定该选项，则缺省值为 -traceback=%none

单独的 -traceback（无值）表示 -traceback=common

注意：如果不需要信息转储，用户可以使用以下方法将 coredumpsize 限制设置为零：

```
% limit coredumpsize 0
```

-traceback 选项不影响运行时性能。

3.4.93 –U

识别源文件中的大写字母和小写字母。

不将大写字母视为与小写字母等效。缺省情况下，将大写字母视为小写字母（字符串常量中的字母除外）。如果使用此选项，编译器会将 Delta、DELTA 和 delta 视为不同的符号。此选项不会影响对内部函数的调用。

可移植性以及将 Fortran 与其他语言混合使用可能要求使用 `-U`。请参见《*Fortran* 编程指南》中有关将程序移植到 Solaris Studio Fortran 的章节。

3.4.94 –U*name*

取消预处理程序宏 *name* 的定义。

此选项仅适用于那些调用 `fpp` 或 `cpp` 预处理程序的源文件。它会删除同一命令行上由 `-D`*name* 创建的预处理程序宏 *name* 的任何初始定义（包括由命令行驱动程序隐式放在此处的那些内容）而不管选项出现的顺序如何。它对源文件中的任何宏定义都没有影响。在命令行上可以出现多个 `-U`*name* 标志。`-U` 和宏 *name* 之间不得有空格。

3.4.95 –u

报告未声明的变量。

使所有变量的缺省类型都成为未声明的，而不是使用 Fortran 隐式确定类型，就好像 `IMPLICIT NONE` 出现在每个编译单元中一样。此选项警告存在未声明的变量，并且不覆盖任何 `IMPLICIT` 语句或显式 *type* 语句。

3.4.96 –unroll=*n*

启用 DO 循环的解开（如果可能）。

n 是正整数。选项包括：

- *n*=1 禁止解开所有循环。
- *n*>1 建议优化器尝试解开循环 *n* 次。

通常，解开循环可提高性能，但会增加可执行文件的大小。有关此编译器及其他编译器优化的更多信息，请参见《*Fortran* 编程指南》的“性能与优化”一章。另请参见第 2.3.1.3 节 “UNROLL 指令” [27]。

3.4.97 –use=*list*

指定隐式 `USE` 模块。

list 是模块名称或模块文件名称的逗号分隔列表。

使用 `-use=`*module_name* 进行编译，可将 `USE` *module_name* 语句添加到正在编译的每个子程序或模块中。使用 `-use=`*module_file_name* 进行编译，可为包含在指定文件中的每个模块添加 `USE` *module_name*。

有关 Fortran 中模块的更多信息，请参见第 4.9 节 “模块文件” [168]。

3.4.98 –V

显示每次编译过程的名称和版本。

此选项在编译器执行时打印每个工作循环的名称和版本。

3.4.99 –v

详细模式 – 显示每次编译过程的详细信息。

与 `-V` 一样，此选项在编译器执行时显示每个工作循环的名称，以及驱动程序使用的选项、宏标志扩展和环境变量的详细信息。

3.4.100 –vax=*keywords*

指定对启用的传统 VAX VMS Fortran 扩展的选择。

keywords 说明符必须是以下子选项之一或从中选择的子选项的逗号分隔列表。

`blank_zero`	**在内部文件上将格式化输入中的空格解释为零。**
`debug`	将以字符 'D' 开头的行解释为正规 Fortran 语句，而不是像 VMS Fortran 中那样解释为注释。
`rsize`	将无格式的记录大小解释为以字为单位，而不是解释为以字节为单位。
`struct_align`	内存中 VAX 结构的布局组件，与 VMS Fortran 中一样，没有填充。注意：这会导致数据无法对齐，为避免此类错误，应与 `–xmemalign` 一同使用。
`%all`	启用所有这些 VAX VMS 功能。
`%none`	禁用所有这些 VAX VMS 功能。

可以单独选择或关闭子选项（通过在前面加上 `no%`）。

示例：

```
-vax=debug,rsize,no%blank_zero
```

缺省值为 `-vax=%none`。指定不带任何子选项的 `-vax` 等效于 `-vax=%all`。

3.4.101 –vpara

显示并行化警告消息。

发出有关 OpenMP 程序中可能存在的并行编程相关问题的警告。

与 `-xopenmp` 选项结合使用。

编译器在检测到下列情形时会发出警告。

- 循环是使用 OpenMP 指令并行化的，而这些指令中的不同循环迭代之间存在数据依赖性。
- OpenMP 数据共享属性子句存在问题。例如，声明在 OpenMP 并行区域中的访问可能导致数据争用的变量 "shared"，或者声明其在并行区域中的值在并行区域之后使用的变量 "private"。

如果所有并行化指令在处理期间均未出现问题，则不显示警告。

3.4.102 -Wc,*arg*

将参数 *arg* 传递给指定的组件 *c*。

前后参数之间只能用逗号分隔。所有 `-W` 参数均在其余的命令行参数之后进行传递。要在参数中包括逗号，请在紧靠逗号之前使用转义符 \（反斜杠）。所有 `-W` 参数均在常规命令行参数之后进行传递。

例如，`-Wa,-o,objfile` 按顺序将 `-o` 和 `objfile` 传递给汇编程序。此外，`-Wl,-I,`*name* 将导致链接阶段覆盖动态链接程序的缺省名称 `/usr/lib/ld.so.1`。

将参数传递给工具时采用的顺序（相对于所指定的其他命令行选项）在以后的编译器发行版中可能会更改。

下表列出了 *c* 的可能值。

表 3-10 `-W` 标志

标志	含义
a	汇编程序：(fbe); (gas)
c	Fortran 代码生成器：(cg) *(SPARC)*

标志	含义
d	f95 驱动程序
l	链接编辑器 (ld)
m	mcs
O（大写的 o）	过程间优化器
o（小写 o）	后优化器
p	预处理程序 (cpp)
0（零）	编译器 (f90comp)
2	优化器：(iropt)

注意：无法使用 -Wd 将 f95 选项传递给 Fortran 编译器。

3.4.103 –w[*n*]

显示或禁止警告消息。

此选项显示或禁止大多数警告消息。但是，如果一个选项覆盖命令行上先前选项的全部或部分，您确实会收到一条警告消息。

n 可以是 0、1、2、3 或 4。

-w0 仅显示错误消息。它等效于 -w。-w1 显示错误和警告。这是不带 -w 情况下的缺省设置。-w2 显示错误、警告和注意。-w3 显示错误、警告、注意和说明。-w4 显示错误、警告、注意、说明和注释。

3.4.104 **-Xlinker** *arg*

将 *arg* 传递给链接程序 ld(1)。与 –z *arg* 等效

3.4.105 –Xlist[*x*]

（仅限 *Solaris*）生成列表并进行全局程序检查 (global program checking, GPC)。

使用此选项可查找潜在的编程错误。它调用额外的编译过程，以便在全局程序中检查子程序调用参数、通用块和参数的一致性。此选项还生成带行号的源代码列表，包括交叉引用表。由 -Xlist 选项发出的错误消息是建议性警告，不会阻止程序的编译和链接。

注 - 请确保在使用 `-Xlist` 进行编译之前，更正源代码中的所有语法错误。如果运行有语法错误的源代码，可能会产生不可预知的报告。

示例： 检查例程之间的一致性：

```
demo% f95 -Xlist  fil.f
```

上述示例将以下内容写入输出文件 `fil.lst`：

- 带行号的源代码列表（缺省）
- 有关例程间不一致性的错误消息（嵌入在列表中）
- 标识符的交叉引用表（缺省）

缺省情况下，将列表写入文件 `name.lst`，其中 `name` 采用命令行上列出的第一个源文件。

许多子选项为操作选择提供了更多灵活性。它们是由 `-Xlist` 主选项的后缀指定的，如下表所示：

表 3-11 `–Xlist` 子选项

选项	特性
`–Xlist`	显示错误、列表和交叉引用表
`–Xlistc`	显示调用图和错误
`–XlistE`	显示错误
`–Xlisterr`*[nnn]*	禁止错误 *nnn* 消息
`–Xlistf`	显示错误、列表和交叉引用，但不显示对象文件
`–Xlisth`	如果检测到错误，则终止编译
`–XlistI`	分析 `#include` 和 `INCLUDE` 文件以及源文件
`–XlistL`	仅显示列表和错误
`–Xlistl`*n*	将页面长度设置为 *n* 行
`-XlistMP`	检查 OpenMP 指令 (SPARC)
`–Xlisto` *name*	将报告文件输出到 *name*，而不是 *file*`.lst`
`–Xlists`	禁止来自交叉引用表的未引用名称
`–Xlistv`*n*	将检查级别设置为 *n*（1、2、3 或 4） – 缺省值为 2
`–Xlistw`*[nnn]*	将输出行的宽度设置为 *nnn* 列 – 缺省值为 79
`–Xlistwar`*[nnn]*	禁止警告 *nnn* 消息
`–XlistX`	显示交叉引用表和错误

有关详细信息，请参见《*Fortran* 编程指南》的“程序分析和调试”一章。

此选项在 Linux 系统上不可用。

3.4.106 –xaddr32[={yes|no}]

（仅限 x86/x64）`-xaddr32=yes` 编译标志将生成的可执行文件或共享对象限制在 32 位地址空间。

以这种方式编译的可执行文件会导致创建限定为 32 位地址空间的进程。指定 `-xaddr32=no` 时，会生成常见的 64 位二进制文件。如果未指定 `-xaddr32` 选项，则假定 `-xaddr32=no`。如果仅指定了 `-xaddr32`，则假定 `-xaddr32=yes`。

此选项仅适用于 `-m64` 编译，并且仅仅在支持 `SF1_SUNW_ADDR32` 软件功能的 Solaris 平台上适用。由于 Linux 内核不支持地址空间限制，此选项在 Linux 上不可用。在 Linux 上将忽略 `-xaddr32` 选项。

链接时，如果单个对象文件是使用 `-xaddr32=yes` 编译的，则假定整个输出文件是使用 `-xaddr32=yes` 编译的。限定为 32 位地址空间的共享对象必须由在受限的 32 位模式地址空间内执行的进程装入。有关更多信息，请参阅《链接程序和库指南》中介绍的 `SF1_SUNW_ADDR32` 软件功能定义。

3.4.107 –xalias[=*keywords*]

指定要由编译器假定的别名程度。

一些非标准编程方法会产生干扰编译器优化策略的情况。使用过度索引、指针以及将全局或非唯一变量作为子程序参数进行传递，会产生歧义别名情况，从而使代码无法按预期方式运行。

使用 `-xalias` 标志可将程序偏离 Fortran 标准的别名要求的程度通知给编译器。

此标志可能带有关键字列表，也可能不带关键字列表。*keywords* 列表由逗号分隔，各个关键字指示程序中存在的别名情况。

可以在每个关键字前面加上 `no%`，以指示不存在的别名类型。

别名关键字如下：

表 3-12 `-xalias` 选项关键字

关键字	含义
`dummy`	子程序的哑元参数（形式参数）可以互为别名，也可以作为全局变量的别名。
`no%dummy`	（缺省值）。哑元参数的使用遵循 Fortran 标准，它们不能互为别名，也不能作为全局变量的别名。
`craypointer`	（缺省值）。Cray 指针可以指向任何全局变量或 `LOC()` 函数采用其地址的局部变量。此外，两个 Cray 指针可能指向同一数据。这是可以禁止某些优化的安全假定。

关键字	含义
no%craypointer	Cray 指针仅指向唯一内存地址，如从 malloc() 获得的地址。此外，没有两个 Cray 指针是指向同一数据的。此假定允许编译器优化 Cray 指针引用。
actual	编译器将子程序的实际参数视为全局变量。将参数传递给子程序可能会导致通过 Cray 指针命名别名。
no%actual	（缺省值）传递参数不会导致进一步命名别名。
overindex	■ 对 COMMON 块中元素的引用可能引用 COMMON 块或等效组中的任何元素。 ■ 如果将 COMMON 块或等效组中的任何元素作为实际参数传递给子程序，被调用的子程序就可以访问该 COMMON 块或等效组中的任何元素。 ■ 将序列派生类型的变量视为 COMMON 块，此类变量的元素可以作为该变量其他元素的别名。 ■ 可能违反了各数组边界，但除上文所述外，假定引用的数组元素仍然在数组内。过度索引并未考虑数组语法、WHERE 和 FORALL 语句等因素。如果在这些构造中出现过度索引，应将它们重写为 DO 循环。
no%overindex	（缺省值）没有违反数组边界。数组引用没有引用其他变量。
ftnpointer	对外部函数的调用可能使 Fortran 指针指向任何类型、种类或等级的目标变量。
no%ftnpointer	（缺省值）Fortran 指针遵循标准中的规则。

指定不带列表的 -xalias 将为不违反 Fortran 别名规则的大多数程序提供最佳性能，不带列表的 -xalias 等效于：

```
no%dummy,no%craypointer,no%actual,no%overindex,no%ftnpointer
```

为了提高效率，在使用优化级别 -xO3 及更高级别进行编译时，应该使用 -xalias。

如果未指定 -xalias 标志，则编译器在缺省情况下假定程序符合 Fortran 标准（Cray 指针除外）：

```
no%dummy,craypointer,no%actual,no%overindex,no%ftnpointer
```

有关各种别名情况的示例以及如何使用 -xalias 指定它们，请参见《*Fortran* 编程指南》的“移植”一章。

3.4.108 –xannotate[={yes|no}]

创建可供优化和监测工具 binopt(1)、code-analyzer(1)、discover(1)、collect(1) 和 uncover(1) 以后使用的二进制文件。

在 Oracle Solaris 上的缺省值为 -xannotate=yes。在 Linux 上的缺省值为 -xannotate=no。指定不带值的 -xannotate 等效于 -xannotate=yes。

要最好地利用优化和监测工具，在编译和链接时都必须采用 -xannotate=yes。如果不使用优化和监测工具，则使用 -xannotate=no 进行编译和链接可以生成略小的二进制文件和库。

3.4.109 –xarch=*isa*

指定指令集体系结构 (instruction set architecture, ISA)。

下表列出了 SPARC 和 x86 平台通用的 -xarch 关键字。

表 3-13 对 SPARC 和 x86 平台通用的 -xarch 关键字

标志	含义
generic	使用大多数处理器通用的指令集。这是缺省值。
generic64	为在大多数 64 位平台上获得良好性能而编译。 此选项等效于 -m64 -xarch=generic，提供此选项的目的是为了与早期的发行版兼容。
native	为了在此系统上获得良好性能而进行编译。编译器为运行它的当前系统处理器选择适当的设置。
native64	为了在 64 位系统上获得良好性能而进行编译。 该选项等效于 -m64 -xarch=native，提供该选项是为了与早期的发行版兼容。

请注意，尽管 -xarch 可以单独使用，但它是 –xtarget 选项扩展的一部分，并且可用于覆盖由特定 -xtarget 选项设置的 -xarch 值。例如：

```
% f95 -xtarget=ultra2 -xarch=sparcfmaf ...
```

会覆盖由 -xtarget=ultra2 设置的 -xarch。

通过只允许指定的指令集，此选项可将编译器生成的代码限定于指定指令集体系结构的指令。此选项不保证使用任何特定于目标的指令。

如果此选项与优化一起使用，则在指定的体系结构上，适当的选择可以为可执行文件提供良好性能。如果选择不当，则会导致二进制程序在预定的目标平台上无法执行。

请注意下列内容：

- 使用 generic、sparc、sparcvis2、sparcvis3、sparcfmaf、sparcima 编译的对象二进制文件 (.o) 可以链接起来并一起执行，但只能在支持链接的所有指令集的处理器上运行。
- 对于任何特定选择，生成的可执行文件在传统体系结构中可能不运行或运行缓慢。此外，由于在任何这些指令集体系结构中都不会实现四精度（REAL*16 和 long double）浮点指令，因此编译器不会在其生成的代码中使用这些指令。

未指定 -xarch 选项时的缺省值为 generic。

表 3-14 “SPARC 平台的 -xarch 值” 提供了有关 SPARC 平台上每个 -xarch 关键字的详细信息。

表 3-14 SPARC 平台的 -xarch 值

-xarch=	含义 (SPARC)
sparc	针对 SPARC–V9 ISA 进行编译。针对 V9 ISA 进行编译，但不具备可视指令集 (Visual Instruction Set, VIS)，也不具备特定于其他实现的 ISA 扩展。该选项在 V9 ISA 上使编译器生成高性能代码。
sparc4	针对 SPARC-V9 ISA 的 SPARC4 版本进行编译。 允许编译器使用 SPARC-V9 指令集中的指令、UltraSPARC 扩展（包括 VIS 1.0）、UltraSPARC-III 扩展（包括 VIS2.0）以及浮点混合乘加指令、VIS 3.0 和 SPARC4 指令。
sparc4b	针对 SPARC-V9 ISA 的 SPARC4B 版本进行编译。允许编译器使用 SPARC-V9 指令集、UltraSPARC 扩展（包括 VIS 1.0）、UltraSPARC-III 扩展（包括 VIS2.0）、面向浮点乘加的 SPARC64 VI 扩展、面向整数乘加的 SPARC64 VII 扩展中的指令以及 SPARC T4 扩展中的 PAUSE 和 CBCOND 指令。
sparc4c	针对 SPARC-V9 ISA 的 SPARC4C 版本进行编译。允许编译器使用 SPARC-V9 指令集、UltraSPARC 扩展（包括 VIS 1.0）、UltraSPARC-III 扩展（包括 VIS2.0）、面向浮点乘加的 SPARC64 VI 扩展、面向整数乘加的 SPARC64 VII 扩展、VIS 3.0 指令的 VIS3B 子集、SPARC T3 扩展的子集中的指令以及 SPARC T4 扩展中的 PAUSE 和 CBCOND 指令。
sparcvis	针对具有 UltraSPARC 扩展的 SPARC–V9 ISA 进行编译。针对 SPARC-V9 加可视指令集 (Visual Instruction Set, VIS) 版本 1.0 进行编译，并具有 UltraSPARC 扩展。该选项在 UltraSPARC 体系结构上使编译器生成高性能代码。
sparcvis2	针对具有 UltraSPARC-III 扩展的 SPARC-V9 ISA 进行编译。此选项允许编译器在具有 UltraSPARC III 扩展的 UltraSPARC 体系结构以及可视指令集 (VIS) 2.0 版上生成对象代码。
sparcvis3	针对 SPARC-V 9 ISA 的 SPARC VIS 版本 3 进行编译。允许编译器使用 SPARC-V9 指令集、UltraSPARC 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 1.0、UltraSPARC-III 扩展（包括可视指令集版本 2.0 以及混合乘加指令和可视指令集版本 3.0 中的指令。
sparcfmaf	针对 SPARC-V9 ISA 的 sparcfmaf 版本进行编译。允许编译器使用 SPARC-V9 指令集，加 UltraSPARC 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 1.0）、UltraSPARC-III 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 2.0）以及面向浮点乘加的 SPARC64 VI 扩展中的指令。 请注意，必须将 -xarch=sparcfmaf 与 -fma=fused 以及某些优化级别结合使用，来使编译器尝试寻找机会自动使用乘加指令。
sparcace	针对 SPARC-V9 ISA 的 sparcace 版本进行编译。使编译器可以使用如下指令集内的指令：SPARC-V9 指令集、UltraSPARC 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 1.0）、UltraSPARC-III 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 2.0）、SPARC64 VI 扩展（用于浮点乘加）和 SPARC64 VII 扩展（用于整数乘加）和 SPARC64 X 扩展（用于 ACE 浮点）。
sparcaceplus	针对 SPARC-V9 ISA 的 sparcaceplus 版本进行编译。允许编译器使用 SPARC-V9 指令集、UltraSPARC 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 1.0）、UltraSPARC-III 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 2.0）、面向浮点乘加的 SPARC64 VI 扩展、面向整数乘加的 SPARC64 VII 扩展、面向 SPARCACE 浮点的 SPARC64 X 扩展以及面向 SPARCACE 浮点的 SPARC64 X+ 扩展中的指令。
sparcima	针对 SPARC-V9 ISA 的 sparcima 版本进行编译。使编译器可以使用如下指令集内的指令：SPARC-V9 指令集、UltraSPARC 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 1.0）、UltraSPARC-III 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 2.0）、SPARC64 VI 扩展（用于浮点乘加）和 SPARC64 VII 扩展（用于整数乘加）。
v9	与 -m64 -xarch=sparc 等效 使用 xarch=v9 来获取 64 位内存模型的传统 makefile 和脚本仅需使用 -m64。

-xarch=	含义 (SPARC)
v9a	与 -m64 -xarch=sparcvis 等效，并与早前的发行版兼容。
v9b	与 -m64 -xarch=sparcvis2 等效，并与早期的发行版兼容。

表 3-15 “x86 平台的 -xarch 值” 详细介绍了 x86 平台上的每个 -xarch 关键字。如果未指定 -xarch，x86 上的缺省值为 generic（或者如果指定了 –m64 则为 generic64）。

表 3-15 x86 平台的 -xarch 值

-xarch=	含义 (x86)
386	将指令集限制于 Intel 386/486 体系结构。
pentium_pro	将指令集限制于 Pentium Pro 体系结构。
pentium_proa	将 AMD 扩展（3DNow!、3DNow! 扩展和 MMX 扩展）添加到 32 位 Pentium Pro 体系结构中。
sse	将 SSE 指令集添加到 pentium_pro 中。（参见下面的注释。）
ssea	将 AMD 扩展（3DNow!、3DNow! 扩展和 MMX 扩展）添加到 32 位 SSE 体系结构中。
sse2	将 SSE2 指令集添加到 pentium_pro 中。（参见下面的注释。）
sse2a	将 AMD 扩展（3DNow!、3DNow! 扩展和 MMX 扩展）添加到 32 位 SSE2 体系结构中。
sse3	向 SSE2 指令集中添加 SSE3 指令集。
sse3a	将 AMD 扩展指令（包括 3dnow）添加到 SSE3 指令集。
amd64	在 Solaris 平台上，这与 -m64 -xarch=sse2 等效 使用 -xarch=amd64 来获取 64 位内存模型的传统 makefile 和脚本应使用 -m64。
amd64a	在 Solaris 平台上，这与 –m64 –xarch=sse2a 等效。
sse3a	将 AMD 扩展指令（包括 3DNow!）添加到 SSE3 指令集。
ssse3	将 SSSE3 指令添加到 SSE3 指令集。
sse4_1	将 SSE4.1 指令添加到 SSSE3 指令集。
sse4_2	将 SSE4.2 指令添加到 SSE4.1 指令集。
amdsse4a	将 SSE4a 指令添加到 AMD 指令集。
aes	添加 Intel 高级加密标准指令集。请注意，指定了 -xarch=aes 时，编译器不会自动生成 AES 指令，除非源代码包含使用了 AES 指令（或引用了 AES 内部函数）的 .il 内联代码、_asm 语句或汇编程序代码。
avx	使用 Intel 高级向量扩展指令集。
avx_i	将 Intel 高级向量扩展指令集与 RDRND、FSGSBASE 和 F16C 指令集一起使用。
avx2	使用 Intel 高级向量扩展 2 指令集。

3.4.109.1 x86/x64 平台的特别注意事项：

当针对 x86 Solaris 平台进行编译时，有几项重要的注意事项。

- 如果在 x86 平台上使用 –m64 编译或链接了某个程序的任意部分，则该程序的所有部分也都必须使用这些选项之一进行编译。有关各种 Intel 指令集体系结构（SSE、SSE2、SSE3、SSSE3 等）的详细信息，请参阅 Intel-64 和 IA-32《*Intel Architecture Software Developer's Manual*》
- 使用设置为 sse、sse2、sse2a 或 sse3 等的 -xarch 编译的程序必须在支持这些功能和扩展的平台上运行。
- 在本发行版中，-xarch=generic 的缺省指令集和含义已更改为 sse2。现在，如果在不指定目标平台选项的情况下进行编译，会导致 sse2 二进制与早期的 Pentium III 或早期的系统不兼容。
- 如果在不同的步骤中进行编译和链接，请始终使用编译器和相同的 -xarch 设置进行链接，以确保链接正确的启动例程。
- x86 上的运算结果可能与 SPARC 上的结果不同，这是由 x86 80 字节浮点寄存器造成的。为了最大限度减少这些差异，请使用 -fstore 选项或使用 -xarch=sse2 进行编译（如果硬件支持 SSE2）。
- 如果在没有相应功能或指令集扩展的平台上运行使用这些 -xarch 选项编译的程序，则可能会导致段故障或不正确的结果，并且不显示任何显式警告消息。
- 如果程序中采用的 .il 内联汇编语言函数或 __asm() 汇编程序代码使用了 SSE、SSE2、SSE2a、SSE3 以及更高版本的指令和扩展，则此警告也适用于这类程序。

3.4.110 –xassume_control[=*keywords*]

设置参数以控制 ASSUME pragma。

使用此标志可控制编译器对源代码中 ASSUME pragma 的处理方式。

ASSUME pragma 为程序员提供了一种断言特殊信息（编译器使用这些特殊信息实现较佳的优化）的方法。可以使用可能值限定这些断言。可能值为 0 或 1 的断言将被标记为“确定”，否则视为不确定。

也可以使用可能性或确定性来断言将要执行的 DO 循环的行程计数，或断言将要采取的分支。

有关 f95 编译器可识别的 ASSUME pragma 的说明，请参见第 2.3.1.8 节 “ASSUME 指令” [29]。

-xassume_control 选项上的 keywords 可以是一个子选项关键字，也可以是关键字的逗号分隔列表。可以识别的关键字子选项如下：

optimize	在 ASSUME pragma 上作出的断言会影响程序的优化。
check	编译器生成用于检查标记为“确定”的所有断言是否正确的代码，并在断言被违反时发出运行时消息；如果没有同时指定 fatal，则程序继续运行。

`fatal`	与 `check` 一起使用时，如果标记为“确定”的断言被违反，则程序将终止。
`retrospective[:`*d*`]`	*d* 参数是可选的容差值，它必须是小于 1 的正实数常量。缺省值为 ".1"。`retrospective` 编译代码以确定所有断言的真假。那些超出容差值 *d* 的值将在程序终止时的输出中列出。
`%none`	忽略所有的 `ASSUME` pragma。

编译器的缺省值是

```
-xassume_control=optimize
```

这意味着，编译器可识别 `ASSUME` pragma，而且后者将影响优化，但不进行检查。

如果指定不带参数的 `-xassume_control`，则隐含

```
-xassume_control=check,fatal
```

在这种情况下，编译器接受并检查所有确定的 `ASSUME` pragma，但是后者不影响优化。无效的断言将导致程序终止。

3.4.111 –xautopar

与 `-autopar` 等效。

3.4.112 –xbinopt={prepare | off}

(SPARC) 此选项已过时，在编译器的将来发行版中会被删除。请参见第 3.4.108 节 “–xannotate[={yes|no}]” [92]。

准备二进制文件以进行编译后优化。

`binopt`(1) 将启用已编译的二进制文件，供以后进行优化、变换和分析使用。在生成可执行文件或共享对象时可以使用此选项，但是它必须与 `-O1` 或更高优化级别一起使用才有效。

在使用此选项生成时，二进制文件的大小会有所增加，大约增加 5%。

如果在不同的步骤中进行编译和链接，则 `-xbinopt` 必须既出现在编译步骤中，也出现在链接步骤中。

如果应用程序的源代码并不都是使用 `-xbinopt` 编译的，则 `-xbinopt` 标志仍应当出现在用于生成程序二进制文件的最终链接步骤中，如下所示：

```
example% f95 -0 program -xbinopt=prepare a.o b.o c.f95
```

只有使用 `-xbinopt` 编译的代码才能用 `binopt`(1) 优化。

为执行 `gprof` 分析而使用 `-xpg` 编译的二进制代码不应与 `binopt` 一起使用，因为这两者不能兼容，结合使用会导致内部错误。

缺省值为 `-xbinopt=off`。

3.4.113 –xcache=*C*

为优化器定义高速缓存属性。

该选项指定了优化器可以使用的缓存属性，不保证使用每个特定的缓存属性。

尽管此选项可以单独使用，但它是 `–xtarget` 选项扩展的一部分；提供它是为了允许覆盖特定 `-xtarget` 选项所隐含的 `-xcache` 值。

表 3-16 `–xcache` 值

值	含义
`generic`	定义高速缓存属性，以便在大多数处理器上获得良好性能，而不使性能有较大幅度的降低。这是缺省值。
`native`	定义高速缓存属性，以便在此主机平台上获得良好性能。
s1/*l1*/*a1*[/*t1*]	定义 1 级高速缓存属性。
s1/*l1*/*a1*[/*t1*]:*s2*/*l2*/*a2*[/*t2*]	定义 1 级和 2 级高速缓存属性。
s1/*l1*/*a1*[/*t1*]:*s2*/*l2*/*a2*[/*t2*]:*s3*/*l3*/*a3*[/*t3*]	定义 1 级、2 级和 3 级高速缓存属性

si/*li* /*ai*/*ti* 字段的定义如下所示：

si 级别为 *i* 的数据高速缓存的大小，以千字节为单位

li 级别为 *i* 的数据高速缓存的行大小，以字节为单位

ai 级别为 *i* 的数据高速缓存的关联性

ti 在级别 *i* 共享缓存的硬件线程数（可选）

示例：`xcache=16/32/4:1024/32/1` 指定以下内容：

一个 1 级高速缓存具有以下属性：16K 字节、32 字节行大小、4 路关联性。

一个 2 级高速缓存具有以下属性： 1024K 字节、32 字节行大小、直接映射关联性。

3.4.114 –xcheck[=*keyword*[,*keyword*]]

生成特殊的运行时检查和初始化。

keyword 必须是以下项之一：

keyword	特性
stkovf[*action*]	生成代码以便在运行时检测堆栈溢出错误，可以选择指定检测到堆栈溢出错误时要执行的操作。 当设置的线程堆栈指针超过为线程分配的堆栈边界时，便会出现堆栈溢出错误。如果堆栈地址的新顶端可写入，则无法检测到错误。 如果内存访问违规作为错误的直接结果发生，从而发出关联信号（通常是 SIGSEGV），则会检测到堆栈溢出错误。我们说由此发出的信号与错误关联。 如果指定了 -xcheck=stkovf[*action*]，当堆栈帧大于系统页面大小时，编译器会生成代码来检测堆栈溢出错误。代码包括库调用以强制内存访问违规而非将堆栈指针设置为无效但有可能映射的地址（请参见 _stack_grow(3C)）。 如果指定，可选的 *action* 必须为 :detect 或 :diagnose。 如果 *action* 为 :detect，则通过执行通常与错误关联的信号处理程序来处理检测到的堆栈溢出错误。 如果 *action* 为 :diagnose，则通过捕获关联信号并调用 stack_violation(3C) 诊断错误来处理检测到的堆栈溢出错误。这是没有指定action时的缺省行为。 如果内存访问违规诊断为堆栈溢出错误，则会将以下消息打印到 stderr： ERROR: stack overflow detected: pc=<inst_addr>, sp=<sp_addr> 其中 <*inst_addr*> 是在其中检测到错误的指令的地址，<*sp_addr*> 是检测到错误时堆栈指针的值。检查堆栈溢出并打印以上消息（如果适用）后，控制能力传递给通常与错误关联的信号处理程序。 -xcheck=stkovf:detect 可将进入时的堆栈边界检查添加到堆栈帧大于系统页面大小的例程（请参见 _stack_grow(3C)）。在大多数应用程序中，额外边界检查的相对成本应该微乎其微。 -xcheck=stkovf:diagnose 可将系统调用添加到线程创建（请参见 sigaltstack(2)）。额外系统调用的相对成本取决于应用程序创建和销毁新线程的频率。 -xcheck=stkovf 仅在 Oracle Solaris 上受支持。Linux 上的 C 运行时库不支持堆栈溢出检测。
no%stkovf	禁用用于检测堆栈溢出的运行时检查。
init_local	执行特殊的本地变量初始化。 编译器将局部变量初始化为这样一个值：如果在赋予该值之前程序使用了它，则可能会导致运算异常。由 ALLOCATE 语句分配的内存也会以这种方式初始化。

keyword	特性
	不对模块变量、STATIC 和 SAVE 本地变量及 COMMON 块中的变量进行初始化。 有关详细信息，请参见《C 用户指南》。
no%init_local	禁用局部变量初始化。这是缺省值。
%all	打开所有这些运行时检查功能。
%none	禁用所有这些运行时检查功能。

堆栈溢出（尤其是在堆栈上分配大数组的多线程应用程序中）可能会在邻近线程堆栈中导致无提示的数据损坏。如果怀疑存在堆栈溢出，请使用 -xcheck=stkovf 编译所有例程。但请注意，使用此标志进行编译不保证将检测到所有堆栈溢出情况，因为它们可能出现在不是使用此标志编译的例程中。

3.4.115 –xchip=*c*

为优化器指定目标处理器。

此选项通过指定目标处理器来指定计时属性。

尽管此选项可以单独使用，但它是 –xtarget 选项扩展的一部分；提供它是为了允许覆盖特定 -xtarget 选项所隐含的 -xchip 值。

下面是 -xchip=*c* 的一些作用：

- 指令调度
- 编译分支的方式
- 在语义等效的两个选择项之间进行选择

下面的两个表列出了有效的 -xchip 处理器名称值：

表 3-17 常用的 –xchip SPARC 处理器的名称

-xchip=	优化：
generic	大多数 SPARC 处理器。（这是缺省情况。）
native	此主机平台。
sparc64vi	SPARC64 VI 处理器
sparc64vii	SPARC64 VII 处理器
sparc64viiplus	SPARC64 VII+ 处理器
sparc64x	SPARC64 X 处理器
sparc64xplus	SPARC64 X+ 处理器
ultra	UltraSPARC 处理器

-xchip=	优化：
ultra2	UltraSPARC II 处理器
ultra2e	UltraSPARC IIe 处理器
ultra2i	UltraSPARC IIi 处理器
ultra3	UltraSPARC III 处理器
ultra3cu	UltraSPARC IIIcu 处理器
ultra3i	UltraSPARC IIIi 处理器
ultra4	UltraSPARC IV 处理器
ultra4plus	UltraSPARC IV+ 处理器
ultraT1	UltraSPARC T1 处理器
ultraT2	UltraSPARC T2 处理器
ultraT2plus	UltraSPARC T2+ 处理器
ultraT3	UltraSPARC T3 处理器
T3	SPARC T3 处理器
T4	SPARC T4 处理器
T5	使用 SPARC T5 处理器的计时属性。
M5	使用 SPARC M5 处理器的计时属性。

注 - 以下 SPARC -xchip 值已过时，在将来的发行版中可能会删除：ultra、ultra2、ultra2e、ultra2i、ultra3、ultra3cu、ultra3i、ultra4 和 ultra4plus。

在 x86 平台上，-xchip 的值为：pentium、pentium_pro、pentium3、pentium4、generic、opteron、core2、penryn、nehalem、amdfam10、sandybridge、ivybridge、haswell、westmere 和 native。

3.4.116 -xcode[=*v*]

(SPARC) 指定代码地址空间。

注 - 应通过指定 -xcode=pic13 或 -xcode=pic32 生成共享对象。尽管您可以通过 -m64 -xcode=abs64 生成可工作的共享对象，但它们的效率将会很低。使用 -m64 -xcode=abs32 或 -m64 -xcode=abs44 生成的共享对象将不起作用。

下表列出了 *v* 的值。

表 3-18 -xcode 标志

值	含义

abs32	这是 32 位体系结构的缺省值。生成 32 位绝对地址。代码 + 数据 + BSS 的大小不超过 2**32 字节。
abs44	这是 64 位体系结构的缺省值。生成 44 位绝对地址。代码 + 数据 + BSS 的大小不超过 2**44 字节。只适用于 64 位体系结构。
abs64	生成 64 位绝对地址。只适用于 64 位架构。
pic13	生成共享库（小模型）中使用的与位置无关的代码。与 -Kpic 等效。允许在 32 位体系结构中最多引用 2**11 个唯一的外部符号，而在 64 位体系结构中可以最多引用 2**10 个。
pic32	生成共享库（大模型）中使用的与位置无关的代码。与 -KPIC 等效。允许在 32 位体系结构中最多引用 2**30 个唯一的外部符号，而在 64 位体系结构中可以最多引用 2**29 个。

对于 32 位体系结构，缺省值是 -xcode=abs32。64 位体系结构的缺省值是 -xcode=abs44。

生成共享动态库时，缺省 -xcode 值 abs44 和 abs32 将与 64 位体系结构一起使用。改为指定 -xcode=pic13 或 -xcode=pic32。在 SPARC 上使用 -xcode=pic13 和 -xcode=pic32 时的两项名义性能开销为：

- 用 –xcode=pic13 或 –xcode=pic32 编译的例程会在入口点执行一些附加指令，以将寄存器设置为指向用于访问共享库的全局变量或静态变量的表 (_GLOBAL_OFFSET_TABLE_)。
- 对全局或静态变量的每次访问都会涉及通过 _GLOBAL_OFFSET_TABLE_ 的额外间接内存引用。如果编译包括 -xcode=pic32，则每个全局和静态内存引用中都会有两个附加指令。

在考虑这些开销时，请记住：由于受到库代码共享的影响，使用 -xcode=pic13 和 -xcode=pic32 会大大减少系统内存需求。共享库中使用 -xcode=pic13 或 -xcode=pic32 编译的每页代码都可以供使用该库的每个进程共享。如果共享库中的代码页包含非 pic（即绝对）内存引用，即使仅包含单个非 pic 内存引用，该页也将变为不可共享，而且每次执行使用该库的程序时都必须创建该页的副本。

确定是否已经使用 -xcode=pic13 或 -xcode=pic32 编译 .o 文件的最简单方法是使用 nm 命令：

```
% nm file.o | grep _GLOBAL_OFFSET_TABLE_ U _GLOBAL_OFFSET_TABLE_
```

包含与位置无关的代码的 .o 文件将包含对 _GLOBAL_OFFSET_TABLE_ 无法解析的外部引用（用字母 U 标记）。

要确定是使用 -xcode=pic13 还是使用 -xcode=pic32，请使用 elfdump -c 检查全局偏移表 (Global Offset Table, GOT) 的大小并查找节头 sh_name: .got。sh_size 值是 GOT 的大小。如果 GOT 小于 8,192 字节，请指定 -xcode=pic13。否则，请指定 -xcode=pic32。有关更多信息，请参见 elfdump(1) 手册页。

按照以下准则来确定应如何使用 -xcode：

- 如果要生成可执行文件，请勿使用 -xcode=pic13 或 -xcode=pic32。

- 如果要生成仅用于链接到可执行文件的归档库，请勿使用 -xcode=pic13 或 -xcode=pic32。
- 如果要生成共享库，请先使用 -xcode=pic13，一旦 GOT 大小超过 8,192 字节，再使用 -xcode=pic32。
- 如果要生成用于链接到共享库的归档库，请使用 -xcode=pic32。

3.4.117 –xcommonchk[={yes|no}]

启用通用块不一致性的运行时检查。

此选项提供了检测使用 TASK COMMON 及并行化的程序中的通用块不一致性的调试检查。（请参见《*Fortran* 编程指南》的“并行化”一章中有关 TASK COMMON 指令的讨论。）

缺省值为 -xcommonchk=no；即，用于检测通用块不一致性的运行时检查处于禁用状态，因为它会导致性能降低。请仅在程序开发和调试过程中使用 -xcommonchk=yes，而不应将其用于符合最终产品质量的程序。

使用 -xcommonchk=yes 进行编译会启用运行时检查。如果在一个源程序单元中声明为正规通用块的通用块出现在 TASK COMMON 指令上的某个其他位置，则程序将停止并显示一条错误消息，指出第一个此类不一致。不带值的 -xcommonchk 与 -xcommonchk=yes 等效。

3.4.118 –xdebugformat={dwarf|stabs}

Solaris Studio 编译器正在将调试器信息的格式从 "stabs" 格式迁移到 "dwarf" 格式。此发行版的缺省设置为 -xdebugformat=dwarf。

如果要维护读取调试信息的软件，您现在可以选择将工具从 stabs 格式转换为 dwarf 格式。

出于移植工具的目的，可以通过此选项来使用新的格式。除非您要维护读取调试器信息的软件，或者特定工具要求使用这些格式之一的调试器信息，否则不需要使用此选项。

-xdebugformat=stabs 生成使用 stabs 标准格式的调试信息。

-xdebugformat=dwarf 生成的调试信息采用 dwarf 标准格式。

如果未指定 -xdebugformat，编译器将假定 -xdebugformat=dwarf。指定此选项而不带参数是不正确的。

此选项影响使用 -g 选项记录的数据的格式。该信息的某些格式也可以使用此选项进行控制。因此，即使不使用 -g，-xdebugformat 仍会产生影响。

dbx 和性能分析器软件可识别 stabs 和 dwarf 格式，因此使用此选项对任何工具的功能都没有影响。

这是过渡性接口，因此会在发行版之间发生更改而不兼容，即使在发行版更新较少时也是如此。stabs 或 dwarf 格式的任何特定字段或值的详细资料也在不断改进。

使用 dwarfdump(1) 命令确定已编译对象文件或可执行文件中调试信息的格式。

3.4.119 -xdebuginfo=*a*[,*a*...]

控制发出多少调试和监测信息。

术语 *tagtype* 是指标记的类型：struct、union、enum 和 class。

以下列表包含子选项 *a* 的可能值。应用于某子选项的前缀no%会禁用该子选项。缺省值为 -xdebuginfo=%none。禁止指定不带子选项的 -xdebuginfo。

%none	不生成任何调试信息。这是缺省值。
[no%]line	发出行号和文件信息。
[no%]param	发出参数的位置列表信息。发出标量值（例如 int、char *）的完整类型信息以及类型名称，但是不发出 *tagtypes* 的完整定义。
[no%]variable	发出词法上全局和局部变量的位置列表信息，包括文件和函数静态信息，但是不包括类静态和 externs 信息。发出标量值（例如 int 和 char *）的完整类型信息以及类型名称，但是不发出 *tagtypes* 的完整定义。
[no%]decl	发出关于函数和变量声明、成员函数以及 class 声明中的静态数据成员的信息。
[no%]tagtype	发出从 param 和 variable 数据集引用的 *tagtypes* 的完整类型定义以及模板定义。
[no%]macro	发出宏信息。
[no%]codetag	发出 DWARF 代码标记（又称为 Stabs N_PATCH）。这是有关由 RTC 和 discover 使用的位字段、结构副本以及溢出的信息。
[no%]hwcpro	生成对硬件计数器分析至关重要的信息。该信息包括 ldst_map – 从 ld/st 指令到引用的符号表条目的映射，以及分支目标地址的 branch_target 表 – 用于验证回溯是否未跨分支目标。有关更多信息，请参见 -xhwcprof。

注 - `ldst_map` 需要提供 *tagtype* 信息。如果不满足这项要求，驱动程序会发出错误。

这些是扩展到 `-xdebuginfo` 组合及其他选项的宏，如下所示：

```
-g = -g2

-gnone =
        -xdebuginfo=%none
        -xglobalize=no
        -xpatchpadding=fix
        -xkeep_unref=no%funcs,no%vars

-g1 =
        -xdebuginfo=line,param,codetag
        -xglobalize=no
        -xpatchpadding=fix
        -xkeep_unref=no%funcs,no%vars

-g2 =
        -xdebuginfo=line,param,decl,variable,tagtype,codetag
        -xglobalize=yes
        -xpatchpadding=fix
        -xkeep_unref=funcs,vars

-g3 =

        -xdebuginfo=line,param,decl,variable,tagtype,codetag,macro
        -xglobalize=yes
        -xpatchpadding=fix
        -xkeep_unref=funcs,vars
```

3.4.120 –xdepend

与 `-depend` 等效。

3.4.121 –xF

允许性能分析器进行函数级重组。

允许使用编译器、性能分析器和链接程序重组核心映像中的函数（子程序）。如果使用 `-xF` 选项编译，然后运行分析器，则可以生成映射文件，该文件可用于优化函数在内存中的排序（具体取决于一起使用这些函数的方式）。可以通过使用链接程序的 `-Mmapfile` 选项，来指示此后用于生成可执行文件的链接使用该映射。它将可执行文件中的每个函数都放置到单独的部分中。（`f95 –M` *path* 选项还将正规文件传递给链接程序，请参见 `f95 –M`*path* 选项的说明）。

仅当应用程序文本缺页时间占用了应用程序的大部分时间时，对内存中的子程序进行重组才有用。否则，重组不能提高应用程序的总体性能。有关分析器的详细信息，请参见 *Program Performance Analysis Tools* 手册。

3.4.122 `–xfilebyteorder=`*options*

支持 little-endian 和 big-endian 平台之间的文件共享。

该标志标识无格式 I/O 文件数据的字节顺序和字节对齐。*options* 必须指定下面的任一组合，但必须至少有一个规范：

`little`*max_align*`:`*spec*

`big`*max_align*`:`*spec*

`native:`*spec*

max_align 为目标平台声明最大字节对齐。允许的值为 1、2、4、8 和 16。对齐适用于 Fortran VAX 结构和 Fortran 派生类型，它们使用依赖于平台的对齐来获得与 C 语言结构的兼容性。

`little` 指定平台上的 "little-endian" 文件，其中最大字节对齐为 *max_align*。例如，`little4` 指定 32 位 x86 文件；而 `little16` 描述 64 位 x86 文件。

`big` 指定最大对齐为 *max_align* 的 "big-endian" 文件。例如，`big8` 描述 32 位 SPARC 文件，而 `big16` 描述 64 位 SPARC 文件。

`native` 指定字节顺序和对齐与编译处理器平台所用的字节顺序和对齐相同的“本机”文件。假定以下内容为“本机”：

平台	“本机”对应于：
32 位 SPARC	`big8`
64 位 SPARC	`big16`
32 位 x86	`little4`
64 位 x86	`little16`

spec 必须是包含以下内容的逗号分隔列表：

`%all`

unit

filename

`%all` 指所有文件和逻辑单元，但用 `"SCRATCH"` 打开的或在 `-xfilebyteorder` 标志中的其他位置显式命名的文件和逻辑单元除外。`%all` 只能出现一次。

unit 指程序打开的特定 Fortran 单元号。

filename 指程序打开的特定 Fortran 文件名。

3.4.122.1 示例：

```
-xfilebyteorder=little4:1,2,afile.in,big8:9,bfile.out,12
-xfilebyteorder=little8:%all,big16:20
```

3.4.122.2 注：

此选项不适用于使用 `STATUS="SCRATCH"` 打开的文件。对这些文件执行的 I/O 操作始终依照本机处理器的字节顺序和字节对齐。

如果命令行中没有出现 `-xfilebyteorder`，则第一个缺省设置为 `-xfilebyteorder=native:%all`。

在此选项中只能声明一次文件名或单元号。

如果在命令行中出现了 `-xfilebyteorder`，则它必须至少带有 little、big 或 native 规范之一。

将此标志没有显式声明的文件假定为本机文件。例如，如果使用 `-xfilebyteorder=little4:zork.out` 进行编译，就会将 `zork.out` 声明为最大数据对齐为 4 字节的 little-endian 32 位 x86 文件。程序中的所有其他文件均是本机文件。

如果为文件指定的字节顺序与本机处理器相同，但指定了不同的对齐，即使没有进行字节交换，也会使用相应的填充。例如，如果针对 64 位 x86 平台使用 `-m64` 进行编译，并指定了 `-xfilebyteorder=little4:`*filename*，就会出现这种情况。

大端字节序和小端字节序平台之间共享的数据记录中的声明类型必须具有相同的大小。例如，使用 `-xtypemap=integer:64,real:64,double:64` 编译的 x86 可执行文件不能读取使用 `-xtypemap=integer:64,real:64,double:128` 编译的 SPARC 可执行文件生成的文件，因为二者的缺省双精度数据类型具有不同的大小。（但请注意，从 Solaris Studio 12 Update 1 发行版开始，x64 处理器上接受 `double::128`。）

如果在一个平台上使用了更改 VAX 结构内的组件对齐方式的选项（如 `–vax=struct_align`）或更改派生类型内的组件对齐方式的选项（如 `–aligncommon` 或 `–dalign`），则在共享同一无格式数据文件（其内容受对齐选项影响）的其他平台上也要使用同一对齐选项。

如果使用整个 UNION/MAP 数据对象对指定为非本机的文件执行 I/O 操作，则会导致运行时 I/O 错误。只能使用 MAP 的个别成员（而不能使用包含 UNION/MAP 的整个 VAX 记录）对非本机文件执行 I/O 操作。

3.4.123 -xglobalize[={yes|no}]

控制文件静态变量的全局化，但是不控制函数的全局化。

全局化是修复并继续和过程间优化所需的一项技术，文件静态符号借以提升到全局范围，同时为名称添加前缀以使命名相同的符号不同。

缺省值为 -xglobalize=no。指定 -xglobalize 与指定 -xglobalize=yes 等效。

3.4.123.1 交互

请参见 -xpatchpadding。

-xipo 也需要全局化，并且将覆盖 -xglobalize。

3.4.124 –xhasc[={yes|no}]

将霍尔瑞斯常量视为实际参数列表中的字符串。

如果使用 -xhasc=yes，则当霍尔瑞斯常量在子例程或函数调用中作为实际参数出现时，编译器将霍尔瑞斯常量视为字符串。这是缺省值，并且符合 Fortran 标准。（编译器生成的实际调用列表包含每个字符串的隐藏字符串长度。）

如果使用 -xhasc=no，则霍尔瑞斯常量将被视为子程序调用中的无类型值，并且只将它们的地址放在实际参数列表中。（传递到子程序的实际调用列表中不会生成字符串长度。）

如果例程调用带有霍尔瑞斯常量的子程序，并且调用的子程序要求参数为 INTEGER（或除 CHARACTER 以外的任意类型），请使用 -xhasc=no 编译例程。

示例：

```
demo% cat hasc.f
               call z(4habcd, 'abcdefg')
               end
               subroutine z(i, s)
               integer i
               character *(*) s
```

```
               print *, "string length = ", len(s)
               return
               end
demo% f95 -o has0 hasc.f
demo% has0
 string length =   4    <-- should be 7
demo% f95 -o has1 -xhasc=no hasc.f
demo% has1
 string length =   7  <-- now correct length for s
```

将 4habcd 传递到 z 是通过使用 -xhasc=no 进行编译来正确处理的。

提供此标记是为了帮助移植传统的 Fortran 77 程序。

3.4.125 –xhelp=flags

列出编译器选项标志。与 -help 等效。

3.4.126 –xhwcprof[={enable | disable}]

(SPARC) 为数据空间分析启用编译器支持。

启用 -xhwcprof 后，编译器将生成信息，该信息可帮助工具将已进行分析的装入指令和存储指令与它们所引用的数据类型和结构成员（结合使用 -g 生成的符号信息）相关联。它将分析数据同目标文件的数据空间（而不是指令空间）相关联，并对行为进行洞察，而这仅从指令分析中是无法轻易获得的。

当可以使用 -xhwcprof 编译指定的对象文件集时，如果对应用程序中的所有对象文件应用该选项，则该选项最为有用。它能全面识别并关联分布在应用程序对象文件中的所有内存引用。

如果在不同的步骤中进行编译和链接，最好在链接时使用 -xhwcprof。

-xhwcprof=enable 或 -xhwcprof=disable 的实例将会覆盖同一命令行中 -xhwcprof 的所有以前的实例。

在缺省情况下，禁用 -xhwcprof。指定不带任何参数的 -xhwcprof 与 -xhwcprof=enable 等效。

-xhwcprof 要求启用优化，并将调试数据格式设置为 dwarf (-xdebugformat=dwarf)，这是当前 Oracle Solaris Studio 编译器的缺省设置。不允许在同一命令行上出现 -xhwcprof 和 -xdebugformat=stabs。

-xhwcprof 使用 -xdebuginfo 自动启用所需的最少量调试信息，所以不需要 -g。

`-xhwcprof` 和 `-g` 的组合会增加编译器临时文件的存储需求，而且比单独指定 `-xhwcprof` 和 `-g` 所引起的增加总量还多。

`-xhwcprof` 作为宏实施，扩展到多个其他更原始的选项，如下所示：

```
-xhwcprof
        -xdebuginfo=hwcprof,tagtype,line
-xhwcprof=enable
        -xdebuginfo=hwcprof,tagtype,line
-xhwcprof=disable
        -xdebuginfo=no%hwcprof,no%tagtype,no%line
```

下列命令可编译 `example.f`，并可为硬件计数器分析以及针对使用 DWARF 符号的数据类型和结构成员的符号分析指定支持：

```
f95 -c -O -xhwcprof -g  example.f
```

有关基于硬件计数器的分析的更多信息，请参见 *Oracle Solaris Studio* 性能分析器手册。

3.4.127 –xia[={widestneed|strict}]

(Solaris) 启用区间运算扩展并设置合适的浮点环境。

如果未指定此选项，则缺省值为 `-xia=widestneed`。

《*Interval Arithmetic Programming Reference*》中详细介绍了区间运算计算的 Fortran 扩展。另请参见第 3.4.132 节 “–xinterval[={widestneed|strict|no}]” [113]。

`-xia` 标志是一个宏，其扩展如下：

`-xia` 或 `-xia=widestneed`	`-xinterval=widestneed -ftrap=%none -fns=no -fsimple=0`
`-xia=strict`	`-xinterval=strict -ftrap=%none -fns=no -fsimple=0`

3.4.128 –xinline=*list*

与 `-inline` 等效。

3.4.129 -xinline_param=*a*[,*a*[,*a*]...]

使用该选项可手动更改编译器用来确定何时内联函数调用的试探式方法。

该选项仅在 -O3 或更高级别上起作用。当自动内联处于启用状态时，以下子选项仅在 -O4 或更高级别上起作用。

在以下子选项中，*n* 必须是正整数；*a* 可以是以下几项之一：

表 3-19 `-xinline_param` 子选项

子选项	含义
`default`	将所有子选项的值设置为缺省值。
`max_inst_hard`[:*n*]	自动内联只将小于n个伪指令（在编译器的内部表示中计数）的函数视为可能的内联候选函数。 在任何情况下，都不会考虑对大于该值的函数进行内联。
`max_inst_soft`[:*n*]	将内联函数的大小限制设置为 *n* 个伪指令（在编译器的内部表示中计数）。 有时可能会对大小大于该值的函数进行内联。 与 `max_inst_hard` 交互时，`max_inst_soft` 的值应该等于或小于 `max_inst_hard` 的值，即 `max_inst_soft` <= `max_inst_hard`。 一般而言，编译器的自动内联程序仅内联所调用函数的大小小于 `max_inst_soft` 的值的调用。在某些情况下，当函数大小大于 `max_inst_soft` 的值但小于 `max_inst_hard` 的值时，可能会内联函数。例如，当传递到函数中的参数是常量时。 当确定是否为了将一个特定调用点内联到函数中而更改 `max_inst_hard` 或 `max_inst_soft` 的值时，请使用 `-xinline_report=2` 报告详细的内联消息，并遵循内联消息中的建议。
`max_function_inst`[:*n*]	允许函数因自动内联而最多增加 *n* 个伪指令（在编译器的内部表示中计数）。
`max_growth`[:*n*]	允许自动内联程序将程序大小最多增大 *n*%，其中大小以伪指令数度量。
`min_counter`[:*n*]	为了考虑要自动内联的函数而通过分析反馈 (`-xprofile`) 度量的最小调用点频率计数。 仅当使用分析反馈 (`-xprofile=use`) 编译应用程序时，该选项才有效。
`level`[:*n*]	使用该子选项可控制所应用自动内联的程度。`-xinline_param=level` 的设置越高，编译器内联的函数越多。 *n* 必须为 1、2 或 3 之一。 当未指定该选项时或者当指定不带 :*n* 的选项时，*n* 的缺省值为 2。 指定自动内联的 `level`： `level:1` 基本内联 `level:2` 中等内联（缺省值） `level:3` 主动内联 `level` 决定了为以下内联参数组合指定的值： `max_growth` + `max_function_inst`

子选项	含义
	+ max_inst + max_inst_call 当 level = 1 时，所有参数都是缺省值的一半。 当 level = 2 时，所有参数都是缺省值。 当 level = 3 时，所有参数都是缺省值的两倍。
max_recursive_depth[:*n*]	当函数直接或间接调用自身时，我们说它进行递归调用。 该子选项可使递归调用自动内联到最高 *n* 级别。
max_recursive_inst[:*n*]	指定递归函数的调用方通过执行自动递归内联可增大到的最大伪指令数（在编译器的内部表示中计数）。 当 max_recursive_inst 与 max_recursive_depth 之间发生交互时，直到 max_recursive_depth 个递归调用或者直到内联到的函数的大小超过 max_recursive_inst，才会内联递归函数调用。这两个参数的设置控制小型递归函数的内联程度。

如果指定了 -xinline_param=default，编译器会将子选项的所有值设置为缺省值。

如果不指定该选项，则缺省值为 -xinline_param=default。

值和选项的列表从左至右进行累积。所以对于 -xinline_param=max_inst_hard:30,..,max_inst_hard:50 的规范，值 max_inst_hard:50 将传递给编译器。

如果在命令行上指定了多个 -xinline_param 选项，则子选项的列表同样从左至右进行累积。例如，

```
 -xinline_param=max_inst_hard:50,min_counter:70 ...
   -xinline_param=max_growth:100,max_inst_hard:100
```

的效果等同于

```
-xinline_param=max_inst_hard:100,min_counter:70,max_growth:100
```

3.4.130 -xinline_report[=*n*]

该选项可生成在编译器内联函数时写入标准输出的报告。报告的类型取决于 *n* 的值，它必须为 0、1 或 2。

0　　不生成报告。

1　　生成内联参数缺省值的摘要报告。

2	生成内联消息的详细报告，显示已内联和未内联的调用点，并注明未内联调用点的简短原因。在某些情况下，该报告将包括针对可用于内联未内联的调用点的 `-xinline_param` 建议的值。

如果未指定 `-xinline_report`，则 *n* 的缺省值为 0。如果指定了不带 *=n* 的 `-xinline_report`，则缺省值为 1。

如果存在 `-xlinkopt`，则有关未内联的调用点的内联消息可能并不准确。

上述情况仅限于由一种编译器执行的内联，这种编译器受可由 `-xinline_param` 选项控制的试探的限制。出于其他原因由编译器内联的调用点可能不会报告这种情况。

3.4.131 –xinstrument=[%no]datarace

指定此选项编译并检测您的程序，以供线程分析器进行分析。

（有关线程分析器的详细信息，请参见 `tha`(1)。）

通过使用此选项进行编译，您可以随后使用性能分析器通过 `collect -r races` 运行已检测的程序，来创建数据争用检测试验。可以单独运行已检测的代码，但其运行速度将非常缓慢。

指定 `-xinstrument=no%datarace` 关闭该功能。这是缺省值。

必须为 `-xinstrument` 指定一个参数。

如果在不同的步骤中进行编译和链接，则在编译和链接步骤都必须指定 `-xinstrument=datarace`。

此选项定义了预处理程序令牌 `__THA_NOTIFY`。可以指定 `#ifdef __THA_NOTIFY` 来保护对 `libtha`(3) 例程的调用。

此选项还设置 `-g`。

3.4.132 –xinterval[={widestneed|strict|no}]

(Oracle Solaris) 启用区间运算扩展。

可选值可以是 `no`、`widestneed` 或 `strict` 之一。如果未指定，则缺省值为 `widestneed`。

`no`	未启用区间运算扩展。
`widestneed`	将任何混合模式表达式中的所有非区间变量和文字提升为表达式中范围最广的区间数据类型。

strict	禁止混合类型或混合长度区间表达式。所有区间类型和长度转换都必须是显式的。

《*Fortran 95 Interval Arithmetic Programming Reference*》中详细介绍了区间运算计算的 Fortran 扩展。另请参见第 3.4.127 节 “-xia[={widestneed|strict}]” [110]。

3.4.133 -xipo[={0|1|2}]

执行过程间优化。

通过调用一次过程间分析传递来执行整个程序的优化。-xipo 可以在链接步骤中的所有对象文件间执行优化，而不是仅限于在编译命令中的源文件。

在编译和链接大型多文件应用程序时，-xipo 特别有用。用该标志编译的对象文件具有在这些文件内编译的分析信息，这些信息实现了在源码和预编译的程序文件中的过程间分析。不过，分析和优化仅限于用 -xipo 编译的对象文件，而不扩展到库的对象文件。

-xipo=0 可禁用过程间分析，-xipo=1 可启用过程间分析。-xipo=2 可添加过程间别名分析以及内存分配和布局优化，以便改善高速缓存性能。缺省值为 -xipo=0，如果指定了不带值的 -xipo，则使用 -xipo=1。

如果使用 -xipo=2 进行编译，未使用 -xipo=2 编译的函数或子例程（例如，库）不能调用使用 -xipo=2 编译的函数或子例程。

例如，如果您干预 malloc() 函数并使用 -xipo=2 编译您自己的 malloc()，则引用与您的代码链接的任何库中的 malloc() 的所有函数也必须使用 -xipo=2 进行编译。由于这对于系统库不大可能，因此您自己的 malloc 不应该使用 -xipo=2 进行编译。

在不同的步骤中进行编译和链接时，必须在这两个步骤中都指定 -xipo 才有效。

在一个编译/链接步骤中使用 -xipo 的示例：

```
demo% f95 -xipo -xO4 -o prog  part1.f part2.f part3.f
```

优化器在三个源文件之间执行交叉文件内联。该操作是在最终的链接步骤中完成的，因此源文件的编译工作不必全在一个编译过程中完成，可以跨多个不同的编译过程，但每个编译过程都需要指定 -xipo。

在不同的编译/链接步骤中使用 -xipo 的示例：

```
demo% f95 -xipo -xO4 -c part1.f part2.f
demo% f95 -xipo -xO4 -c part3.f
demo% f95 -xipo -xO4 -o prog  part1.o part2.o part3.o
```

在编译步骤中创建的对象文件具有在文件内部编译的附加分析信息，这样就可以在链接步骤中执行跨文件优化。

即使使用 -xipo 进行编译，也存在库不参与跨文件过程间分析的限制，如下例所示：

```
demo% f95 -xipo -xO4 one.f two.f three.f
demo% ar -r mylib.a one.o two.o three.o
...
demo% f95 -xipo -xO4 -o myprog main.f four.f mylib.a
```

在此例中，过程间优化将在 one.f、two.f 和 three.f 之间以及 main.f 和 four.f 之间执行，但不在 main.f 或 four.f 与 mylib.a 上的例程之间执行。（第一个编译可能生成有关未定义符号的警告，但仍可执行过程间优化，因为过程间优化是编译和链接的一个步骤。）

关于 -xipo 的其他重要信息：

- 至少需要优化级别 -xO4
- 如果在内部版本中使用的对象文件对于并行运行的链接步骤通用，则使用并行 make 工具生成采用 -xipo 编译的可执行文件会产生问题。每个链接步骤都应有自己的对象文件副本，并且该副本应在链接前进行优化。
- 未使用 -xipo 编译的对象可以与使用 -xipo 编译的对象自由链接。
- 由于执行跨文件优化时需要附加信息，因此 -xipo 选项会生成更大的对象文件。不过，该附加信息不会成为最终的二进制可执行文件的一部分。可执行程序大小的增加是由于执行额外的优化导致的。
- 在此发行版中，跨文件子程序内联是由 -xipo 执行的唯一过程间优化。
- .s 汇编语言文件不参与过程间分析。
- 如果使用 -S 进行编译，则忽略 -xipo 标志。

何时不使用 -xipo 进行编译：

在链接步骤中使用对象文件集合时，编译器试图执行整个程序的分析和优化。对于该对象文件集合中定义的任何函数或子例程 foo()，编译器作出以下两个假定：

1. 运行时，在该对象文件集合外部定义的其他例程将不显式调用 foo()
2. 从该对象文件集合中的任何例程调用 foo() 时，将不会受到在该对象文件集合外部定义的不同版本的 foo() 的干预。

如果假定 (1) 对给定应用程序不成立，请勿使用 -xipo=2 进行编译。如果假定 (2) 不成立，请勿使用 -xipo=1 或 -xipo=2 进行编译。

例如，考虑使用您自己的源版本干预 malloc() 函数并使用 -xipo=2 进行编译。然后，任何库中引用与您的代码链接的 malloc() 的所有函数也必须使用 -xipo=2 进行编译，并且它们的对象文件将不需要参与链接步骤。由于这对于系统库不大可能，因此您自己的 malloc（） 不应该使用 -xipo=2 进行编译。

另举一例，假定您使用以下两个外部调用来生成共享库：两个不同的源文件中的 foo() 和 bar()，并且 bar() 调用其主体内的 foo()。如果有可能在运行时干预函数调用 foo()，则不要使用 -xipo=1 或 -xipo=2 编译 foo() 或 bar() 的任何一个源文件。否则，foo() 可以内联到 bar()，这会导致在使用 -xipo 编译时出现不正确的结果。

3.4.134 –xipo_archive[={none|readonly|writeback}]

(SPARC) 允许跨文件优化包括归档 (.a) 库。

值必须是以下项之一：

none	不执行归档文件的处理。编译器不会对使用 -xipo 编译和在链接时从归档库中提取的对象文件应用跨模块内联或其他跨模块优化。因此，必须在链接时同时指定 -xipo 和 -xipo_archive=readonly（或 -xipo_archive=writeback）。
readonly	生成可执行文件之前，编译器使用通过 -xipo 编译的对象文件（驻留在归档库 (.a) 中）来优化传递到链接程序的对象文件。 选项 -xipo_archive=readonly 启用对在链接时指定的归档库中的对象文件的跨模块内联和过程间数据流分析。但是，它不启用对归档库代码的跨模块优化，除非代码已经通过跨模块内联插入到其他模块中。 要对归档库中的代码应用跨模块优化，需要使用 -xipo_archive=writeback。请注意，这样做将修改从中提取代码的归档库的内容。
writeback	生成可执行文件之前，编译器使用通过 -xipo 编译的对象文件（驻留在归档库 (.a) 中）来优化传递到链接程序的对象文件。库中包含的在编译期间优化的任何对象文件都会替换为其优化后的版本。 对于使用归档库通用集的并行链接，每个链接都应创建自己的归档库备份，从而在链接前进行优化。

如果未指定 -xipo_archive 的设置，编译器将假定 -xipo_archive=none。

3.4.135 -xipo_build=[yes|no]

在不设置 -xipo_build 的情况下生成 -xipo 涉及通过编译器的两次传递，一次是在生成对象文件时，再一次是随后在链接时执行跨文件优化时。设置 -xipo_build 可避免初始传递期间的优化而仅在链接时优化，从而缩短编译时间。无需对对象文件进行优化，与 -xipo 一样，将在链接时执行优化。如果使用 -xipo_build 生成的未优化对象文件链接起来而未包括 -xipo 来执行优化，则应用程序将因无法解析的符号错误而无法链接。

3.4.135.1 -xipo_build 示例

以下示例将执行 .o 文件的快速生成，后跟链接时跨文件优化：

```
% cc -O -xipo -xipo_build -o code1.o -c code1.c
% cc -O -xipo -xipo_build -o code2.o -c code2.c
% cc -O -xipo -o a.out code1.o code2.o
```

-xipo_build 将在创建 .o 文件时禁用 -O，以快速生成这些文件。链接时将执行完全 -O 优化，作为 -xipo 跨文件优化的一部分。

以下示例在不使用 -xipo 的情况下执行链接。

```
% cc -O -o a.out code1.o code2.o
```

如果使用 -xipo_build 生成了 code1.o 或 code2.o，结果将发生链接时故障，指明符号 __unoptimized_object_file 无法解析。

单独生成 .o 文件时，缺省行为是 -xipo_build=no。但是，当从源文件的一次传递中生成可执行文件或库时，将隐式启用 -xipo_build。例如：

```
% cc -fast -xipo a.c b.c c.c
```

将为生成 a.o、b.o 和 c.o 的第一次传递隐式启用 -xipo_build=yes。包括选项 -xipo_build=no 可禁用该行为。

3.4.136 –xivdep[=*p*]

禁用或设置 !DIR$ IVDEP 指令的解释。

IVDEP 指令指示编译器忽略其在循环中找到的部分或全部对数组引用的循环附带依赖性，使编译器能够在其他循环之间执行各种循环优化，例如微向量化、分布、软件流水化，否则这些优化将无法实现。当用户知道这些依赖性无关紧要或者实际上永远不会发生时，可以使用该指令。

!DIR$ IVDEP 指令的解释取决于 -xivdep 选项的值。*p* 的以下值解释如下：

loop – 忽略假定的循环附带向量依赖性
loop_any – 忽略所有循环附带向量依赖性
back – 忽略假定的向后循环附带向量依赖性
back_any – 忽略所有向后循环附带向量依赖性
none – 不忽略任何依赖性（禁用 IVDEP 指令）

提供这些解释是为了与另一个供应商对 IVDEP 指令的解释兼容。

不指定 -xivdep 以及指定不带参数的 -xivdep 时，缺省值都是 -xivdep=loop，意味着按缺省启用 !DIR$ IVDEP 指令。

有关更多信息，请参见第 2.3.3 节 “IVDEP 指令” [31]。

3.4.137 -xjobs{=*n*|auto}

使用多个进程编译。如果未指定该标志，则缺省行为是 -xjobs=auto。

指定 -xjobs 选项可设置编译器完成其任务需创建的进程数。在多 CPU 计算机上，该选项可以缩短生成时间。目前，-xjobs 只能与 -xipo 选项一起使用。如果指定 -xjobs=*n*，过程间优化器就将 *n* 作为其在编译不同文件时可调用的最大代码生成器实例数。

通常，*n* 的安全值等于 1.5 乘以可用处理器数。如果使用的值是可用处理器数的数倍，则会降低性能，因为有在产生的作业间进行的上下文切换开销。此外，如果使用很大的数值会耗尽系统资源（如交换空间）。

如果指定了 -xjobs=auto，则编译器将自动选择适当数量的并行作业。

指定 -xjobs 时务必要指定值。否则，会发出错误诊断并使编译终止。

如果未指定 -xjobs，则缺省行为是 -xjobs=auto。通过将 -xjobs=*n* 添加到命令行可覆盖该行为。出现最合适的实例之前，-xjobs 的多重实例在命令行上会互相覆盖。

3.4.137.1 -xjobs 示例

以下示例将与 -xipo 的最多三个并行进程链接：

```
% cc -xipo -xO4 -xjobs=3 t1.o t2.o t3.o
```

以下示例将与 -xipo 的一个进程串行链接：

```
% cc -xipo -xO4 -xjobs=1 t1.o t2.o t3.o
```

以下示例将与为 -xipo 选择作业数量的编译器并行链接：

```
% cc -xipo -xO4  t1.o t2.o t3.o
```

请注意，这与显式指定 -xjobs=auto 时的行为完全相同：

```
% cc -xipo -xO4 -xjobs=auto t1.o t2.o t3.o
```

3.4.138 -xkeep_unref[={[no%]funcs,[no%]vars}]

保留未引用函数和变量的定义。no% 前缀使编译器有可能删除这些定义。

缺省值为 no%funcs,no%vars。指定 -xkeep_unref 等效于指定 -xkeep_unref=funcs,vars，表示 -keep_unref 保留一切。

3.4.139 –xkeepframe[=[%all,%none,*name*,no%*name*]]

禁止对命名函数 (*name*) 进行与堆栈相关的优化。

%all　　禁止对所有代码进行与堆栈相关的优化。

`%none` 允许对所有代码进行与堆栈相关的优化。

此选项是累积性的，可以多次出现在命令行中。例如，`–xkeepframe=%all –xkeepframe=no%func1` 表示应当为除 `func1` 以外的所有函数保留堆栈帧。而且，`–xkeepframe` 优先于 `–xregs=frameptr`。例如，`–xkeepframe=%all –xregs=frameptr` 表示应保留所有函数的堆栈，但会忽略 `–xregs=frameptr` 的优化。

如果命令行中未指定，编译器将采用 `-xkeepframe=%none` 作为缺省值。如果指定了但没有值，编译器将采用 `-xkeepframe=%all`。

3.4.140 –xknown_lib=*library_list*

识别对已知库的调用。

如果指定此选项，编译器会将对某些已知库的调用视为内部函数，从而忽略用户提供的任何版本。这样，编译器就可以根据它具备的有关该库的专业知识来对库例程调用进行优化。

library_list 是当前应用于 `blas`、`blas1`、`blas2`、`blas3` 和 `intrinsics` 的关键字列表（用逗号分隔）。编译器能够识别对以下 BLAS1、BLAS2 和 BLAS3 库例程的调用，并且能够针对 Sun 性能库实现自由地进行正确优化。编译器将忽略这些库例程的用户提供版本，并使用 Sun 性能库中的 BLAS 例程或对例程进行内联。

要与 Sun 性能库进行链接，需要使用 `–library=sunperf` 选项。

-xknown_lib=	特性
`blas1`	编译器能够识别对以下 BLAS1 库例程的调用： `caxpy ccopy cdotc cdotu crotg cscal csrot csscal cswap dasum daxpy dcopy ddot drot drotg drotm drotmg dscal dsdot dswap dnrm2 dzasum dznrm2 icamax idamax isamax izamax sasum saxpy scasum scnrm2 scopy sdot sdsdot snrm2 srot srotg srotm srotmg sscal sswap zaxpy zcopy zdotc zdotu zdrot zdscal zrotg zscal zswap`
`blas2`	编译器能够识别对以下 BLAS2 库例程的调用： `cgemv cgerc cgeru ctrmv ctrsv dgemv dger dsymv dsyr dsyr2 dtrmv dtrsv sgemv sger ssymv ssyr ssyr2 strmv strsv zgemv zgerc zgeru ztrmv ztrsv`
`blas3`	编译器能够识别对以下 BLAS2 库例程的调用： `cgemm csymm csyr2k csyrk ctrmm ctrsm dgemm dsymm dsyr2k dsyrk dtrmm dtrsm sgemm ssymm ssyr2k ssyrk strmm strsm zgemm zsymm zsyr2k zsyrk ztrmm ztrsm`
`blas`	选择所有 BLAS 例程。与 `-xknown_lib=blas1,blas2,blas3` 等效。
`intrinsics`	编译器会忽略 Fortran 内部函数的任何显式 `EXTERNAL` 声明，因此将忽略用户提供的任何内例程。（有关内部函数名称列表，请参见《*Fortran* 库参考》。）

3.4.141 –xl

（已过时）不再支持此传统 f77 选项。要获得当前 Fortran 编译器中的等效选项，请使用：-f77=%all,no%backslash -vax=$all,no%debug

3.4.142 –xlang=f77

(SPARC) 为链接与早期版本的传统 f77 编译器所创建的对象兼容的运行时库做准备。

f95 -xlang=f77 表示与 f77compat 库进行链接，这是将 f95 对象文件与更早的 Fortran 77 对象文件相链接的简便方法。使用此标志进行编译，可确保正确的运行时环境。

如果将 f95 和 f77 已编译对象一起链接到一个可执行文件，请使用 f95 -xlang=f77。

使用 -xlang 进行编译时，请注意以下事项：

- 不要同时使用 -xnolib 和 -xlang 编译。
- 将 Fortran 对象文件和 C++ 混合使用时，请使用 C++ 编译器进行链接，并在 CC 命令行上指定 -xlang=f95。
- 将 C++ 对象与使用任何并行化选项编译的 Fortran 对象文件混合使用时，链接 CC 命令行还必须指定 -mt。

要确定将哪个驱动程序用于混合语言链接，请使用下列语言分层结构：

C++	使用 CC 命令。有关详细信息，请参见《*C++* 用户指南》。
Fortran 95（或 Fortran 90）	使用 f95 命令。
Fortran 77	使用 f95 -xlang=f77。
C	使用 cc 命令。有关详细信息，请参见《*C* 用户指南》。

3.4.143 –xld

（已过时）不再支持该 (f77) 选项。要获得当前 Fortran 编译器中的等效选项，请使用：-f77=%all,no%backslash -vax=$all,no%debug

3.4.144 –xlibmil

与 -libmil 等效。

3.4.145 –xlibmopt

使用优化数学例程的库。

使用为速度进行了优化的选定数学例程。此选项通常生成更快的代码。它可能生成稍有不同的结果；如果是这样，通常是最后一位不同。该库选项在命令行上的顺序并不重要。

3.4.146 –xlic_lib=sunperf

已过时。使用 –library=sunperf 可与 Sun 性能库进行链接。

3.4.147 –xlinkopt[={1|2|0}]

对可重定位对象文件执行链接时优化。

后优化器在链接时对二进制对象代码执行一些高级性能优化。可以使用可选值来设置执行的优化级别，可选值必须为 0、1 或 2。

0	禁用后优化器。（这是缺省情况。）
1	在链接时根据控制流分析执行优化，包括指令高速缓存着色和分支优化。
2	在链接时执行附加的数据流分析，包括无用代码删除和地址计算简化。

指定不带值的 -xlinkopt 标志即表示 -xlinkopt=1。

这些优化在链接时通过分析二进制对象代码来执行。虽然未重写对象文件，但生成的可执行代码可能与初始对象代码不同。

当与分析反馈一起用于编译整个程序时，此选项最有效。

如果在不同的步骤中进行编译，则 -xlinkopt 必须既出现在编译步骤中，也出现在链接步骤中。

```
demo% f95 -c -xlinkopt a.f95 b.f95
demo% f95 -o myprog -xlinkopt=2 a.o b.o
```

请注意，仅当编译器链接时才使用级别参数。在上述示例中，即使二进制对象代码是用隐含级别 1 编译的，使用的后优化级别仍然是 2。

不能将链接时后优化器与增量链接程序 `ild` 一起使用。`-xlinkopt` 标志会将缺省链接程序设置为 `ld`。如果使用 `-xildon` 标志显式启用增量链接程序，将禁用 `-xlinkopt` 选项（如果同时指定了二者）。

要使 `-xlinkopt` 选项有用，至少程序中的一些例程（但未必是全部例程）必须使用此选项编译。优化器仍可以对未使用 `-xlinkopt` 进行编译的二进制对象执行部分受限的优化。

`-xlinkopt` 选项优化出现在编译器命令行上的静态库代码，但不会优化出现在命令行上的共享（动态）库代码。生成共享库（用 `-G` 编译）时，您也可以使用 `-xlinkopt`。

`-xlinkopt` 选项需要分析反馈 (`-xprofile`) 来优化程序。分析功能会展示代码中最常用和最不常用的部分，从而使优化器相应地进行处理。链接时优化对大型应用程序尤为重要，因为在链接时执行代码优化放置可降低指令高速缓存未命中数。另外，`-xlinkopt` 在用于编译整个程序时最有效。此选项的用法如下所示：

```
demo% f95 -o progt -xO5 -xprofile=collect:prog file.f95
demo% progt
demo% f95 -o prog -xO5 -xprofile=use:prog -xlinkopt file.95
```

有关使用分析反馈的详细信息，请参见 `-xprofile` 选项。

请注意，使用此选项编译会略微延长链接时间。对象文件的大小也会增加，但可执行文件的大小保持不变。如果使用 `-xlinkopt` 和 `-g` 标志进行编译，则会因包括调试信息而增加了可执行文件的大小。

3.4.148 –xloopinfo

与 `-loopinfo` 等效。

3.4.149 –xM

生成 make 依赖项。

该选项可在标准输出中为编译的源文件生成 make 依赖项。该选项涵盖源文件的所有 make 依赖项，包括头文件和 Fortran 模块。

对于模块依赖项，该选项使用基于对象的模块依赖项方案，以便无需显式生成规则即可创建模块文件。

该选项不能与 -c、-S、-Xlist 或会生成不同编译输出的其他任何编译选项一起使用。

生成的依赖项输出不包含任何生成规则，只包含文件的依赖项。用户需要为内部版本所需的所有文件指定生成规则。但是，对于模块文件，无需显式生成规则，因为模块文件与关联的对象文件同时创建。因此，模块文件仅需具有通用生成规则：

```
%.mod:
        @ echo $@ is already up to date.
```

模块文件生成规则只需要阻止 'make' 进程在模块文件没有生成规则时剥离与其相关的所有依赖项。除此之外，生成规则不执行任何操作，如上例中所示。

与 -keepmod 选项一起使用时，-xM 选项生成的依赖项将防止因不必要更新模块文件产生编译级联，并防止因使用 -keepmod 选项导致对相同源文件进行重新编译的问题，以防止对模块文件进行不必要的更新。

该选项可与 -M、-I 和 -moddir 选项结合使用，为内部版本中所需的模块文件确定适当的目录。预编译的模块文件（例如，由第三方提供的模块文件）应该位于由 -M 选项指向的目录，以便生成正确的依赖项。

3.4.150 –xmaxopt[=*n*]

启用优化 pragma 并设置最大优化级别。

n 具有值 1 至 5，分别对应于优化级别 -O1 至 -O5。如果未指定，编译器将使用 5。

此选项启用 !$PRAGMA SUN OPT=*n* 指令（当此指令出现在源输入中时）。不使用此选项时，编译器将这些行视为注释。请参见第 2.3.1.5 节 “OPT 指令” [28]。

如果此 pragma 与某个优化级别一起出现，而该优化级别高于 -xmaxopt 标志上的最高级别，则编译器将使用由 -xmaxopt 设置的级别。

3.4.151 –xmemalign[=<*a*><*b*>]

(SPARC) 指定未对齐数据访问的最大假定内存对齐和行为。

对于可在编译时确定对齐的内存访问，编译器会为数据对齐生成相应的装入/存储指令序列。

对于不能在编译时确定对齐的内存访问，编译器必须假定一个对齐以生成所需的装入/存储序列。

使用 -xmemalign 标志，用户可以指定因上述未确定情况编译器要假定的数据最大内存对齐。它还指定了在运行时发生未对齐内存访问时的错误行为。

指定的值包含两个部分：数值对齐值 <*a*>，以及字母行为标志 <*b*>。

对齐 <*a*> 的允许值有：

1	假定最多 1 字节对齐。
2	假定最多 2 字节对齐。
4	假定最多 4 字节对齐。
8	假定最多 8 字节对齐。
16	假定最多 16 字节对齐。

访问未对齐数据时错误行为 <*b*> 的允许值有：

i	解释访问并继续执行。
s	产生信号 SIGBUS
f	在 64 位平台上，产生信号 SIGBUS 仅对齐少于或等于 4，否则将解释访问并继续执行。在其他平台上，f 与 i 等效。

如果在不指定 -xmemalign 的情况下进行编译，缺省值为：

- 针对 32 位平台的 8i
- 针对采用 C 和 C++ 的 64 位平台的 8s
- 针对采用 Fortran 的 64 位平台的 8f

对于所有平台，-xmemalign 不带值显示时的缺省值为 1i。

请注意，-xmemalign 本身并不强制进行任何特殊的数据对齐。使用 -dalign 或 -aligncommon 可强制进行数据对齐。

此外，只要链接到使用 b 值 i 或 f 编译的对象文件，就必须指定 -xmemalign。

-dalign 选项是一个宏：

-dalign 是 -xmemalign=8s -aligncommon=16 的宏

不要将 -aligncommon=1 与 -xmemalign 一起使用，因为这些声明会发生冲突，在某些平台和配置上可能会引发段故障。

有关详细信息，请参见第 3.4.1 节 “–aligncommon[={1|2|4|8|16}]” [47]。

3.4.152 –xmodel=[small | kernel | medium]

(x86)在 Solaris x64 平台上为共享对象指定数据地址模型。

使用 `-xmodel` 选项，编译器可以为 Oracle Solaris x64 平台创建 64 位共享对象，并且只应对此类对象的编译指定该选项。

仅当在启用了 64 位的 x86 平台 ("x64") 上指定了 `-m64` 时，此选项才有效。

`small`	此选项可为小模型生成代码，其中执行代码的虚拟地址在链接时已知，并且已知在 0 到 2^{31} - 2^{24} - 1 的虚拟地址范围内可以找到所有符号。
`kernel`	为内核模型生成代码，在该模型中，所有符号都定义在 2^{64} - 2^{31} 到 2^{64} - 2^{24} 范围内。
`medium`	按中等模型生成代码，在该模型中，不对数据段的符号引用范围进行假定。文本段的大小和地址的限制与小型代码模型的限制相同。使用 `–m64` 编译含有大量静态数据的应用程序时，可能需要使用 `-xmodel=medium`。

如果未指定 `-xmodel`，编译器将假定 `-xmodel=small`。如果指定没有参数的 `-xmodel`，将出现错误。

只要您确保所访问的对象位于相应的范围内，就没有必要使用该选项编译所有的例程。

3.4.153 –xnolib

与 `-nolib` 等效。

3.4.154 –xnolibmil

与 `-nolibmil` 等效。

3.4.155 –xnolibmopt

不使用快速数学库。

与 -fast 一起使用可以覆盖对优化数学库的链接：

```
f95 -fast -xnolibmopt …
```

3.4.156 –x0*n*

与 -O*n* 等效。

3.4.157 –xopenmp[={parallel|noopt|none}]

启用通过OpenMP指令进行的显式并行化。

该标志接受以下子选项关键字：

parallel — 启用 OpenMP pragma 的识别。-xopenmp=parallel 时的优化级别为 -xO3。如有必要，编译器会将优化级别提高到 -xO3 并发出警告。

此标志还定义处理器宏 _OPENMP。_OPENMP 宏定义为具有十进制值 *yyyymm*，其中 *yyyy* 和 *mm* 是实现所支持的 OpenMP API 版本的年份和月份标示。有关特定发行版的 _OPENMP 宏的值，请参阅《*Oracle Solaris Studio OpenMP API* 用户指南》。

noopt — 启用 OpenMP pragma 的识别。如果优化级别低于 -xO3，则编译器不提升它。如果将优化级别显式设置为低于 -xO3，如同在 f95 -xO2 -xopenmp=noopt 中一样，编译器会发出错误。如果没有使用 -xopenmp=noopt 指定优化级别，则会识别 OpenMP Pragma，并相应地对程序进行并行处理，但不进行优化。此子选项还定义预处理程序宏 _OPENMP。

none — 不启用对 OpenMP pragma 的识别，不更改程序的优化级别并且不定义任何预处理程序宏。在未指定 -xopenmp 时，这是缺省值。

如果指定了 -xopenmp 但未指定子选项关键字，编译器将假定 -xopenmp=parallel。如果根本未指定 -xopenmp，编译器将假定 -xopenmp=none。

子选项 parallel 和 noopt 将自动调用 -stackvar。

如果使用 dbx 调试 OpenMP 程序，那么编译时使用 -g -xopenmp=noopt 可以在并行区设置断点并显示变量内容。

在以后的发行版中，-xopenmp 的缺省值可能会更改。可以通过显式指定适当的优化级别来避免警告消息。

使用 OMP_NUM_THREADS 环境变量可指定在运行 OpenMP 程序时要使用的线程数。如果未设置 OMP_NUM_THREADS，用于执行并行区域的线程的缺省数量为计算机上的可用内核数，上限为 32。可以通过以下方法指定不同线程数：设置 OMP_NUM_THREADS 环境变量，或调用 omp_set_num_threads（） OpenMP 运行时例程，或者在并行区域指令中使用 num_threads 子句。为了获得最佳性能，用于执行并行区域的线程数不应超出计算机上的可用硬件线程（或虚拟处理器）数量。在 Oracle Solaris 系统上，可以使用 psrinfo(1M) 命令确定此数量。在 Linux 系统上，可以检查 /proc/cpuinfo 文件来确定此数量。有关更多信息，请参见 *OpenMP API* 用户指南。

缺省情况下，禁用嵌套并行操作。要启用嵌套并行操作，必须将 OMP_NESTED 环境变量设置为 TRUE。请参见《*OpenMP API* 用户指南》。

如果在不同的步骤中进行编译和链接，请在编译步骤和链接步骤中都指定 -xopenmp。与链接步骤配合使用时，-xopenmp 选项将与 OpenMP 运行时支持库 libmtsk.so 链接。

为了获得最新功能和性能，请确保系统上安装了 OpenMP 运行时库 libmtsk.so 的最新修补程序。

有关用于生成多线程应用程序的 OpenMP Fortran 95、C 和 C++ 应用程序接口 (application program interface, API) 的更多信息，请参见《*Oracle Solaris Studio OpenMP API* 用户指南》。

3.4.158 –xpad

与 -pad 等效。

3.4.159 –xpagesize=*size*

为堆栈和堆设置首选页面大小。

在 SPARC 平台上，*size* 值必须是以下值之一：

8K、64K、512K、4M、32M、256M、2G、16G 或 default

在 x86 平台上，*size* 值必须是以下值之一：

4K、2M、4M 或 default

例如： -xpagesize=4M

并非所有这些页面大小在所有平台上都受支持，具体取决于体系结构和 Oracle Solaris 环境。指定的页面大小对于目标平台上的 Oracle Solaris 操作环境必须是有效的页面大小。如果不是，此请求在运行时将被忽略。

使用 `pagesize`(1) Oracle Solaris 命令可以确定页面中的字节数。操作系统不保证支持页面大小请求。但是，可以使用适当段对齐来增加获取请求的页面大小的可能性。有关如何设置段对齐，请参见 `-xsegment_align` 选项。可以使用 `pmap`(1) 或 `meminfo`(2) 来确定目标平台的页面大小。

如果指定了 `-xpagesize=default`，该标志将被忽略；如果指定了不带 `size` 值的 *-xpagesize*，则与 `-xpagesize=default` 等效。

此选项是组合使用 `-xpagesize_heap=`*size* `-xpagesize_stack=`*size*.这两个选项与 `-xpagesize` 接受相同的参数。可以通过指定 `-xpagesize=`*size* 来为二者设置相同的值，或分别为它们指定不同的值。

使用该标志进行编译，与使用等效选项将 `LD_PRELOAD` 环境变量设置为 `mpss.so.1` 或在启动程序之前使用等效选项运行 Oracle Solaris 命令 `ppgsz`(1) 具有相同的效果。有关详细信息，请参见 Oracle Solaris 手册页。

3.4.160 –xpagesize_heap=*size*

为堆设置首选页面大小。

size 值与所述的 `-xpagesize` 值相同。

有关详细信息，请参见 `-xpagesize`。

3.4.161 –xpagesize_stack=*size*

(SPARC) 为堆栈设置首选页面大小。

size 值与所述的 `-xpagesize` 值相同。

有关详细信息，请参见 `-xpagesize`。

3.4.162 -xpatchpadding[={fix|patch|*size*}]

在各个函数启动之前保留内存区域。如果指定了 `fix`，编译器将保留 fix 所需的空间量并继续。这是第一个缺省值。如果指定了 `patch` 或未指定任何值，则编译器将保留特定于

平台的缺省值。值 `-xpatchpadding=0` 表示保留 0 字节的空间。在 x86 上 *size* 的最大值是 127 字节，在 SPARC 上是 2048 字节。

3.4.163 –xpec[={yes|no}]

生成 PEC（Portable Executable Code，可移植执行代码）二进制文件。

此选项将程序中间表示置于对象文件和二进制文件中。该二进制文件可在以后用于调整和故障排除。

使用 `-xpec` 生成的二进制文件通常要比不使用该选项生成的文件大 5 到 10 倍。缺省值为 `-xpec=no`。

不带参数的 `-xpec` 与 `-xpec=yes` 等效。

3.4.164 –xpg

与 `-pg` 等效。

3.4.165 –xpp={fpp|cpp}

选择源文件预处理程序。

缺省值为 `-xpp=fpp`。

编译器使用 `fpp`(1) 来预处理 `.F`、`.F95` 或 `.F03` 源文件。此预处理程序适用于 Fortran。以前的版本使用标准 C 预处理程序 `cpp`。要选择 `cpp`，请指定 `-xpp=cpp`。

3.4.166 –xprefetch[=*a*[,*a*]]

在支持预取的体系结构上启用预取指令。

有关 Fortran `PREFETCH` 指令的说明，请参见第 2.3.1.7 节 “PREFETCH 指令” [29]。

a 必须是以下值之一：

`auto`	启用预取指令的自动生成
`no%auto`	禁用预取指令的自动生成

`explicit` 启用显式预取宏

`no%explicit` 禁用显式预取宏

`latx:`*factor* (SPARC) 按指定的因子调整编译器的假定预取到装入的延迟和预取到存储的延迟。该因子必须是正浮点数或整数。

如果要在较大的 SPARC 多处理器上运行计算密集的代码，您会发现使用 `-xprefetch=latx:`*factor* 有很多优点。该选项指示代码生成器按照指定的因子调节在预取及其相关的装入或存储之间的缺省延迟时间。

预取延迟是从执行预取指令到所预取的数据在高速缓存中可用那一刻之间的硬件延迟。在确定发出预取指令到发出使用所预取数据的装入或存储指令之间的间隔时，编译器就采用预取延迟值。

注 - 在预取和装入之间采用的延迟可能与在预取和存储之间采用的延迟不同。

编译器可以在众多计算机与应用程序间调整预取机制，以获得最佳性能。这种调整并非总能达到最优。对于占用大量内存的应用程序，尤其是要在大型多处理器上运行的应用程序，可以通过增加预取延迟值来提高性能。要增加值，请使用大于 1 的因子。介于 .5 和 2.0 之间的值最有可能提供最佳性能。

对于数据集完全位于外部高速缓存中的应用程序，可以通过减小预取延迟值来提高性能。要减小此值，请使用小于 1 的因子。

要使用 `-xprefetch=latx:`*factor* 选项，请首先使用接近 1.0 的因子值并对应用程序运行性能测试。然后适当增加或减小该因子，并再次运行性能测试。继续调整因子并运行性能测试，直到获得最佳性能。以很小的增量逐渐增加或减小因子时，前几步中不会看到性能差异，之后会突然出现差异，然后再趋于稳定。

`yes` `-xprefetch=yes` 与 `-xprefetch=auto,explicit` 相同

`no` `-xprefetch=no` 与 `-xprefetch=no%auto,no%explicit` 相同

使用 `-xprefetch`、`-xprefetch=auto` 和 `-xprefetch=yes` 时，编译器就可以将预取指令插入到其生成的代码中。该操作会提高支持预取的体系结构的性能。

3.4.166.1 缺省值：

如果未指定 `-xprefetch`，则假定 `-xprefetch=auto,explicit`。

如果仅指定了 `-xprefetch`，则假定为 `-xprefetch=auto,explicit`。

如果使用 `-xprefetch` 或 `-xprefetch=yes` 等启用了自动预取，但未指定延迟因子，则假定 `-xprefetch=latx:1.0`。

3.4.166.2 交互：

如果使用 -xprefetch=explicit，编译器将能够识别以下指令：

!$PRAGMA SUN_PREFETCH_READ_ONCE (*name*)
!$PRAGMA SUN_PREFETCH_READ_MANY (*name*)
!$PRAGMA SUN_PREFETCH_WRITE_ONCE (*name*)
!$PRAGMA SUN_PREFETCH_WRITE_MANY (*name*)

-xchip 设置影响假定延迟的决定以及 latx:*factor* 设置的结果。

仅当在 SPARC 处理器上启用了自动预取 (auto) 时，*latx:*factor 子选项才有效。

3.4.166.3 警告：

显式预取只应在度量支持的特殊环境下使用。

因为编译器可以在众多计算机与应用程序间调整预取机制以获得最佳性能，所以当性能测试指示性能明显提高时，应当只使用 -xprefetch=latx:*factor*。假定的预取延迟在不同发行版本中是不同的。因此，无论何时切换到不同的发行版本，强烈建议重新测试延迟因子对性能的影响。

3.4.167 –xprefetch_auto_type=indirect_array_access

为间接访问的数据数组生成间接预取。

以生成直接内存访问预取的相同方式来生成由选项 -xprefetch_level={1|2|3} 指定的循环的间接预取。可添加前缀 no% 否定声明。

缺省值为 -xprefetch_auto_type=no%indirect_array_access。

要求 -xprefetch=auto 以及优化级别 -xO3 或更高级别。

诸如 -xdepend 之类的选项可以影响计算间接预取候选项的主动性，进而影响由于更好的内存别名歧义消除信息而发生的自动间接预取插入的主动性。

3.4.168 –xprefetch_level={1|2|3}

控制预取指令的自动生成。

在以下情况下编译时，此选项才有效：

- -xprefetch=auto，

- 使用优化级别 3 或更高。
- 在支持预取的平台上。

如果未指定 `-xprefetch_level`，则 `-xprefetch=auto` 的缺省值为级别 2。

预取级别 2 比级别 1 产生更多的预取指令机会。预取级别 3 比级别 2 生成更多的预取指令。

在早期的 SPARC 或 x86 平台上，预取级别 2 和 3 可能不会生效。

3.4.169 –xprofile=*p*

收集用于分析的数据或使用分析进行优化。

p 必须是 `collect`[:*profdir*]、`use`[:*profdir*] 或 `tcov`[:*profdir*]。

此选项可在执行期间收集并保存执行频率数据，然后在后续运行中可以使用该数据来改进性能。对多线程应用程序来讲，分析收集 (Profile collection) 是一种安全的方法。也就是说，对执行其自己的多任务处理 (`-mt`) 的程序进行分析可产生准确的结果。只有指定 `-xO2` 或更高的优化级别时，此选项才有效。如果分别执行编译和链接，则链接步骤和编译步骤中必须都出现同一 `-xprofile` 选项。

`collect`[:*profdir*]

在 `-xprofile=use` 时优化器收集并保存执行频率，以供将来使用。编译器生成可测量语句执行频率的代码。

`-xMerge`, `-ztext`, 和 `-xprofile=collect` 不应同时使用。`-xMerge` 会强制将静态初始化的数据存储到只读存储器中，`-ztext` 禁止在只读存储器中进行依赖于位置的符号重定位，而 `-xprofile=collect` 会在可写存储器中生成静态初始化的、依赖于位置的符号重定位。

分析目录名 *profdir*（如果指定）是包含已分析的对象代码的程序或共享库在执行时用来存储分析数据的目录路径名。如果 *profdir* 路径名不是绝对路径，在使用选项 `-xprofile=use:`*profdir* 编译程序时将相对于当前工作目录来解释该路径。

如果未使用 `–xprofile=collect:` *prof_dir* 或 `–xprofile=tcov:` *prof_dir* 指定任何分析目录名，则分析数据在运行时将存储在名为 *program*`.profile` 的目录中，其中 *program* 是已分析进程的主程序的根基名称。在此情况下，环境变量 `SUN_PROFDATA` 和 `SUN_PROFDATA_DIR` 可用于控制在运行时将分析数据存储在何处。如果已设置，分析数据将写入 `$SUN_PROFDATA_DIR/$SUN_PROFDATA` 指定的目录。如果在编译时指定了分析目录名，则 `SUN_PROFDATA_DIR` 和 `SUN_PROFDATA` 在运行时不起任何作用。这些环境变量同样控制由 `tcov` 写入的分析数据文件的路径和名称，如 `tcov`(1) 手册页中所述。

如果未设置这些环境变量，分析数据将写入当前目录中的目录 *profdir*`.profile`，其中 *profdir* 是可执行文件的名称或在 `-xprofile=collect:`*profdir* 标志中指定的名称。如果 *profdir* 已在 `.profile` 中结束，`-xprofile` 不会将 `.profile` 附加到 *profdir* 中。如果多次运行程序，那么执行频率数据会累积在 *profdir*`.profile` 目录中；也就是说，以前执行的输出不会丢失。

如果在不同的步骤中进行编译和链接，应确保使用 `-xprofile=collect` 编译的任何对象文件也使用 `-xprofile=collect` 进行链接。

`use`[:*profdir*]

通过从使用 `–xprofile=collect`[: *profdir*] 或 `–xprofile=tcov`[: *profdir*] 编译的代码中收集的执行频率数据针对在执行已分析代码时执行的工作进行优化。*profdir* 是某个目录的路径名，该目录包含通过运行使用 `–xprofile=collect`[: *profdir*] 或 `–xprofile=tcov`[: *profdir*] 编译的程序收集的分析数据。

要生成 `tcov` 和 `–xprofile=use`[:*profdir*] 都能使用的数据，必须在编译时使用 `–xprofile=tcov`[:*profdir*] 选项指定一个分析目录。必须在 `–xprofile=tcov:` *profdir* 和 `–xprofile=use:` *profdir* 中指定相同的分析目录。为最大限度地减少混淆情况，请将 *profdir* 指定为绝对路径名。

profdir 路径名是可选的。如果未指定 *profdir*，将使用可执行二进制文件的名称。如果未指定 `-o`，将使用 `a.out`。如果未指定 *profdir*，编译器将查找 `profdir.profile/feedback` 或 *a.out.profile/feedback*。例如：

```
demo% f95 -xprofile=collect -o myexe prog.f95
demo% f95 -xprofile=use:myexe -xO5 -o myexe prog.f95
```

程序是使用以前生成并保存在 feedback 文件中的执行频率数据优化的，此数据是先前执行用 `-xprofile=collect` 编译的程序时写入的。

除了 `-xprofile` 选项之外，源文件和其他编译器选项必须与用于编译（该编译过程创建了生成 feedback 文件的编译程序）的相应选项完全相同。编译器的相同版本必须同时用于 `collect` 生成和 `use` 生成。

如果用 `-xprofile=collect:`*profdir* 编译，则必须将相同的分析目录名 *profdir* 用在优化编译中：`-xprofile=use:` *profdir*。

另请参见 `-xprofile_ircache`，以了解有关加速 `collect` 阶段和 `use` 阶段之间的编译的说明。

`tcov`[:*profdir*]

使用 `tcov`(1) 检测对象文件以进行基本块覆盖分析。

如果指定可选的 *profdir* 参数，编译器将在指定位置创建分析目录。该分析目录中存储的数据可通过 `tcov`(1) 或由编译器通过 `-`

xprofile=use:*profdir* 来使用。如果省略可选的 *profdir* 路径名，执行已进行分析的程序时将创建分析目录。只能通过 tcov(1) 使用该分析目录中存储的数据。使用环境变量 SUN_PROFDATA 和 SUN_PROFDATA_DIR 可以控制分析目录的位置。

如果 *profdir* 指定的位置不是绝对路径名，则在编译时将相对于编译时的当前工作目录来解释该位置。如果为任何对象文件指定了 *profdir*，则必须为同一程序中的所有对象文件指定同一位置。由 *profdir* 指定位置的目录必须在要执行已进行分析的程序的所有计算机中都可以访问。除非不再需要分析目录中的内容，否则不应删除该目录，因为除非重新编译，否则编译器存储在其中的数据将无法恢复。

示例 [1]：如果用 -xprofile=tcov:/test/profdata 编译一个或多个程序的对象文件，编译器会创建一个名为 /test/profdata.profile 的目录并将其用来存储描述已进行分析的对象文件的数据。该同一目录还可在执行时用来存储与已进行分析的对象文件关联的执行数据。

示例 [2]：如果名为 myprog 的程序用 -xprofile=tcov 编译并在目录 /home/joe 中执行，将在运行时创建目录 /home/joe/myprog.profile 并将其用来存储运行时分析数据。

3.4.170 –xprofile_ircache[=*path*]

(SPARC)保存并重用收集阶段和使用分析阶段之间的编译数据。

在使用阶段，与 -xprofile=collect|use 一起使用会重用收集阶段保存的编译数据，从而可以减少编译时间。

如果指定，*path* 将覆盖高速缓存文件的保存位置。缺省情况下，这些文件会与对象文件保存在同一目录下。当收集阶段和使用阶段出现在两个不同的位置时，指定路径便十分有用。

典型的命令序列可能是：

```
demo% f95 -xO5 -xprofile=collect -xprofile_ircache t1.c t2.c
demo% a.out     collects feedback data
demo% f95 -xO5 -xprofile=use -xprofile_ircache t1.c t2.c
```

对于大程序，通过用这种方式保存中间数据，可以显著缩短使用阶段的编译时间。但这将以显著增加所用的磁盘空间为代价。

3.4.171 –xprofile_pathmap=*collect_prefix*:*use_prefix*

(SPARC) 设置分析数据文件的路径映射。

请将 -xprofile_pathmap 选项与 -xprofile=use 选项一起使用。

如果编译器找不到用 -xprofile=use 编译的对象文件的分析数据，请使用 -xprofile_pathmap，并且：

- 当前使用 -xprofile=use 进行编译所用的目录，不是先前使用 -xprofile=collect 编译时使用的目录。
- 对象文件在分析中共享公共基名，但可以根据它们在不同目录中的位置互相区分。

collect-prefix 是目录树的 UNIX 路径名的前缀，该目录树中的对象文件是使用 -xprofile=collect 编译的。

use-prefix 是目录树的 UNIX 路径名的前缀，该目录树中的对象文件是使用 -xprofile=use 编译的。

如果指定了 -xprofile_pathmap 的多个实例，编译器将按照这些实例的出现顺序对其进行处理。将 *-xprofile_pathmap* 实例指定的每个 use-prefix 与对象文件路径名进行比较，直至找到匹配的 *use-prefix* 或发现最后一个指定的 *use-prefix* 与对象文件路径名也不匹配。

3.4.172 –xrecursive

允许不带 RECURSIVE 属性的例程以递归方式调用它们自己。

通常，只有使用 RECURSIVE 属性定义的子程序才能以递归方式调用它们自己。

使用 -xrecursive 进行编译时，子程序将可以调用自身，即使它们没有使用 RECURSIVE 属性进行定义也是如此。但是，与定义了 RECURSIVE 的子例程不同，缺省情况下，使用此标志不会导致在堆栈上分配局部变量。要使局部变量在子程序的每个递归调用中具有不同的值，还应该使用 -stackvar 进行编译以便将局部变量放在堆栈上。

间接递归（例程 A 调用例程 B，而例程 B 又调用例程 A）可在高于 -xO2 的优化级别上生成不一致的结果。使用 -xrecursive 标志进行编译，可以保证使用间接递归的正确性，即使优化级别更高也是如此。

使用 -xrecursive 进行编译会导致性能下降。

3.4.173 –xreduction

与 -reduction 等效。

3.4.174 –xregs=*r*

为生成的代码指定寄存器用法。

r 是一个逗号分隔列表，它包含下面的一个或多个子选项：`appl`、`float`、`frameptr`。

用 `no%` 作为子选项的前缀会禁用该子选项。

请注意，`-xregs` 子选项仅限于特定的硬件平台。

示例：`-xregs=appl,no%float`

表 3-20 `-xregs` 子选项

值	含义
`appl`	(SPARC) 允许编译器将应用程序寄存器用作临时寄存器来生成代码。应用程序寄存器是： `g2`、`g3`、`g4`（在 32 位平台上） `g2`、`g3`（在 64 位平台上） 强烈建议使用 `-xregs=no%appl` 编译所有系统软件和库。系统软件（包括共享库）必须为应用程序保留这些寄存器值。这些值的使用将由编译系统控制，而且在整个应用程序中必须保持一致。 在 SPARC ABI 中，这些寄存器表示为应用程序寄存器。由于需要更少的装入和存储指令，因此使用这些寄存器可提高性能。但是，这样使用可能与某些用汇编代码编写的旧库程序冲突。
`float`	(SPARC) 允许编译器通过将浮点寄存器用作整数值的临时寄存器来生成代码。使用浮点值可能会用到与该选项无关的这些寄存器。如果希望您的代码没有任何对浮点寄存器的引用，需要使用 `-xregs=no%float` 并确保您的代码不会以任何方式使用浮点类型。
`frameptr`	(x86) 允许编译器将帧指针寄存器（IA32 上为 `%ebp`，AMD64 上为 `%rbp`）用作通用寄存器。 缺省值为 `-xregs=no%frameptr` 通过 `-xregs=framptr`，编译器可以自由使用帧指针寄存器来改进程序性能。但是，这可能会限制调试器和性能测量工具的某些功能。堆栈跟踪、调试器和性能分析器不能对通过 `-xregs=frameptr` 编译的函数生成报告。 如果直接调用或从 C 或 Fortran 函数间接调用的 C++ 函数会引发异常，不应该用 `–xregs=frameptr` 编译 C、Fortran 和 C++ 混合代码。如果使用 `–fast` 编译此类混合源代码，请在命令行中的 `–fast` 选项后添加 `–xregs=no%frameptr`。 由于 64 位平台上的可用寄存器更多，因此相对于 64 位代码相比，使用 `–xregs=frameptr` 编译更容易改进 32 位代码的性能。 如果同时指定了 `-pg`，编译器会忽略 `-xregs=frameptr` 并发出警告。而且，`-xkeepframe` 优先于 `-xregs=frameptr`。

SPARC 缺省值为 -xregs=appl,float。

x86 缺省值为 -xregs=no%frameptr。-xregs=frameptr 包含在 -fast 的扩展中。

强烈推荐您用 -xregs=no%appl,float 编译那些用于与应用程序链接的共享库的代码。至少共享库应该显式说明它如何使用应用程序寄存器，以便与这些库链接的应用程序知道这些寄存器分配。

例如，在某种全局意义上使用寄存器（例如，使用寄存器指向一些关键数据结构）的应用程序，需要确切地知道其代码未使用 -xregs=no%appl 编译的某个库如何使用应用程序寄存器，以便安全地与该库链接。

在 x86 系统上，-pg 与 -xregs=frameptr 不兼容，这两个选项不应一起使用。还请注意，-fast 中包括 -xregs=frameptr。

3.4.175 -xs[={yes|no}]

(Oracle Solaris) 将调试信息从对象文件链接到可执行文件。

-xs 与 -xs=yes 相同。

-xdebugformat=dwarf 的缺省值与 -xs=yes 相同。

-xdebugformat=stabs 的缺省值与 -xs=no 相同。

该选项控制可执行文件大小与为了调试而需要保留对象文件之间的权衡。对于 dwarf，请使用 -xs=no 使可执行文件保持较小但依赖于对象文件。对于 stabs，请使用 -xs 或 -xs=yes 以较大的可执行文件为代价避免对对象文件的依赖。该选项对程序的 dbx 性能或运行时性能几乎没有影响。

当编译命令强制链接时（即未指定 -c），没有对象文件和调试信息必须放置在可执行文件中。在这种情况下，将忽略 -xs=no（隐式或显式）。

该功能的实施方法是让编译器调整发出的对象文件中的节标志和/或节名称，然后告知链接程序对该对象文件的调试信息执行什么操作。因此它是一个编译器选项，而非链接程序选项。可以使一个可执行文件的某些对象文件使用 -xs=yes 进行编译，而其他对象文件使用 -xs=no 进行编译。

Linux 编译器接受但是忽略 -xs。Linux 编译器不接受 -xs={yes|no}。

3.4.176 –xsafe=mem

(SPARC)允许编译器假定未发生违反内存保护的情况。

使用此选项可允许编译器假定未发生基于内存的陷阱。此选项允许在 SPARC V9 平台上使用推测装入指令。

仅当与优化级别 -xO5 及以下 -xarch 值中的一个一起使用时，此选项才能有效：sparc、sparcvis、sparcvis2 或 sparcvis3（用于 -m32 和 -m64）。

注意 - 由于在发生诸如地址未对齐或段违规的故障时，无故障装入不会导致陷阱，因此您应该只对不会发生此类故障的程序使用该选项。因为只有很少的程序会导致基于内存的陷阱，所以您可以安全地将该选项用于大多数程序。对于显式依赖基于内存的陷阱来处理异常情况的程序，请勿使用该选项。

3.4.177 -xsegment_align=*n*

(Oracle Solaris) 此选项使驱动程序在链接行上包括特殊映射文件。映射文件将文本、数据和 bss 段对齐到 *n* 指定的值。使用非常大的页面时，在适当的边界上对齐堆和堆栈段非常重要。如果未对齐这些段，将使用小页面直到下一个边界，这会导致性能下降。映射文件确保在适当边界上对齐段。

n 值必须是以下项之一：

SPARC：以下值有效：8K、64K、512K、2M、4M、32M、256M、1G 和 none。

x86：以下值有效：4K、8K、64K、512K、2M、4M、32M、256M、1G 和 none。

SPARC 和 x86 的缺省值为 none。

建议的用法如下所示：

```
SPARC 32-bit compilation: -xsegment_align=64K
SPARC 64-bit compilation: -xsegment_align=4M

x86 32-bit compilation: -xsegment_align=8K
x86 64-bit compilation: -xsegment_align=4M
```

驱动程序将包括相应的映射文件。例如，如果用户指定 -xsegment_align=4M，驱动程序会将 -M *install-directory*/lib/compilers/mapfiles/map.4M.align 添加到链接行，其中 *install-directory* 是安装目录。然后，将在 4M 边界上对齐上述段。

3.4.178 –xspace

不执行会增加代码大小的优化。

示例： 如果增加代码大小，则不会解开循环或并行化循环。

3.4.179 –xtarget=*t*

为指令集和优化指定目标平台。

t 必须是以下值之一： native、native64、generic、generic64 和 *platform–name*。

-xtarget 选项允许简便快捷地指定发生在实际平台上的 -xarch、-xchip 和 -xcache 组合。-xtarget 的唯一含义在其扩展中。

通过为编译器提供目标计算机硬件的精确描述，某些程序的性能可得到提高。当程序性能很重要时，目标硬件的正确指定是非常重要的。在较新的 SPARC 处理器上运行时，尤其是这样。不过，对大多数程序和较旧的 SPARC 处理器来讲，性能的提高微不足道，因此指定 generic 就足够了。

-xtarget 值的实际扩展可能会因发行版的不同而异。通过使用 -dryrun 标志，您始终可以确定编译器将使用的扩展：

```
demo% f95 -dryrun -xtarget=ultra4plus
###     command line files and options (expanded):
### -dryrun -xarch=sparcvis
    -xcache=64/32/4/1:2048/64/4/2:32768/64/4/2 -xchip=ultra4plus
```

请注意，针对特定名称平台的 –xtarget 扩展不得与同一平台上的 –xtarget=native 相同。

3.4.179.1 通用平台与本机平台

native	优化主机平台（32 位）的性能。 扩展至 –m32 –xarch=native –xchip=native –xcache=native
native64	已过时。改用 –xtarget=native –m64。
generic	为大多数 32 位平台获得最佳性能。 这是缺省值并可扩展至：-m32 -xarch=generic -xchip=generic -xcache=generic
generic64	已过时。改用 -xtarget=generic -m64。
platform–name	获取以下列出的指定平台的最佳性能。

3.4.179.2 SPARC 平台

下表列出了编译器接受的常用系统平台的名称。

表 3-21 常用的 -xtarget 系统平台的扩展

-xtarget= *platform-name*	**-xarch**	**-xchip**	**-xcache**
sparc64vi	sparcfmaf	sparc64vi	128/64/2:5120/64/10
sparc64vii	sparcima	sparc64vii	64/64/2:5120/256/10
sparc64viiplus	sparcima	sparc64viiplus	64/64/2:11264/256/11
sparc64x	sparcace	sparc64x	64/128/4/2:24576/128/24/32
sparc64xplus	sparcaceplus	sparc64xplus	64/128/4/2:24576/128/24/32
ultra	sparcvis	ultra	16/32/1:512/64/1
ultra1/140	sparcvis	ultra	16/32/1:512/64/1
ultra1/170	sparcvis	ultra	16/32/1:512/64/1
ultra1/200	sparcvis	ultra	16/32/1:512/64/1
ultra2	sparcvis	ultra2	16/32/1:512/64/1
ultra2/1170	sparcvis	ultra	16/32/1:512/64/1
ultra2/1200	sparcvis	ultra	16/32/1:1024/64/1
ultra2/1300	sparcvis	ultra2	16/32/1:2048/64/1
ultra2/2170	sparcvis	ultra	16/32/1:512/64/1
ultra2/2200	sparcvis	ultra	16/32/1:1024/64/1
ultra2/2300	sparcvis	ultra2	16/32/1:2048/64/1
ultra2e	sparcvis	ultra2e	16/32/1:256/64/4
ultra2i	sparcvis	ultra2i	16/32/1:512/64/1
ultra3	sparcvis2	ultra3	64/32/4:8192/512/1
ultra3cu	sparcvis2	ultra3cu	64/32/4:8192/512/2
ultra3i	sparcvis2	ultra3i	64/32/4:1024/64/4
ultra4	sparcvis2	ultra4	64/32/4:8192/128/2
ultra4plus	sparcvis2	ultra4plus	64/32/4/1:2048/64/4/2:32768/64/4/2
ultraT1	sparc	ultraT1	8/16/4/4:3072/64/12/32
ultraT2	sparcvis2	ultraT2	8/16/4:4096/64/16
ultraT2plus	sparcvis2	ultraT2plus	8/16/4:4096/64/16
ultraT3	sparcvis3	ultraT3	8/16/4:6144/64/24
T3	sparcvis3	T3	8/16/4:6144/64/24
T4	sparc4	T4	16/32/4:128/32/8:4096/64/16
T5	sparc4	T5	16/32/4/8:128/32/8/8:8192/64/16/128
M5	sparc4	M5	16/32/4/8:128/32/8/8:49152/64/12/48

注 - 以下 SPARC -xtarget 值已过时，在将来的发行版中可能会删除：ultra、ultra1/140、ultra1/170、ultra1/200、ultra2、ultra2e、ultra2i、ultra2/1170、ultra2/1200、ultra2/1300、ultra2/2170、ultra2/2200、ultra2/2300、ultra3、ultra3cu、ultra3i、ultra4 和 ultra4plus。

-m64 标志表示针对启用了 64 位的平台上的 64 位 Solaris OS 进行编译。如果指定了 -xtarget，则 -m64 必须出现在 -xtarget 标志之后，如下所示：

```
-xtarget=ultra2   ...   -m64
```

否则，将使用缺省的 32 位内存模型。

3.4.179.3 X86 平台

下表显示了针对 x86 系统的有效 -xtarget 平台名称及其扩展。

表 3-22 -xtarget 值（x86 平台）

-xtarget=	-xarch	-xchip	-xcache
pentium	386	pentium	generic
pentium_pro	pentium_pro	pentium_pro	generic
pentium3	sse	pentium3	16/32/4:256/32/4
pentium4	sse2	pentium4	8/64/4:256/128/8
opteron	sse2a	opteron	64/64/2:1024/64/16
woodcrest	ssse3	core2	32/64/8:4096/64/16
barcelona	amdsse4a	amdfam10	64/64/2:512/64/16
penryn	sse4_1	penryn	2/64/8:6144/64/24
nehalem	sse4_2	nehalem	32/64/8:256/64/8:8192/64/16
westmere	aes	westmere	32/64/8:256/64/8:30720/64/24
sandybridge	avx	sandybridge	32/64/8/2:256/64/8/2:20480/64/20/16
ivybridge	avx_i	ivybridge	32/64/8/2:256/64/8/2:20480/64/20/16
haswell	avx2	haswell	32/64/8/2:256/64/8/2:20480/64/20/16

-m64 标志表示针对启用了 64 位的 x86 平台上的 64 位 Solaris OS 进行编译。例如，使用 -xtarget=opteron 进行编译是不必要的或不够的。如果指定了 -xtarget，则 -m64 选项必须出现在 -xtarget 标志之后，如下所示：

```
-xtarget=opteron -m64
```

否则，编译将为 32 位 x86。

3.4.180 -xtemp=*path*

等效于 -temp*path*。

3.4.181 -xthroughput[={yes|no}]

-xthroughput 选项可告知编译器当多个进程同时在系统上运行时应用程序将会运行

如果 -xthroughput=yes，编译器首选这样的优化：会稍微降低单个进程的性能，同时提高系统上所有进程完成的工作量。例如，编译器可能会选择在预取数据时不太主动。这样的选择会减少该进程消耗的内存带宽，由此该进程运行速度可能会减慢，但是也会留出更多内存带宽供其他进程共享。

缺省值为 -xthroughput=no。

3.4.182 –xtime

与 -time 等效。

3.4.183 –xtypemap=*spec*

指定缺省数据映射。

此选项提供了一种为缺省数据类型指定字节大小的灵活方法。此选项适用于缺省大小的变量和常量。

规范字符串 *spec* 可以包含以下任何或全部项（采用逗号分隔列表形式）：

real:*size*,double:*size*,integer:*size*

每个平台上允许的组合包括：

- real:32
- real:64
- double:64

- double:128
- integer:16
- integer:32
- integer:64

例如：

- –xtypemap=real:64,double:64,integer:64

可同时将缺省的 REAL 和 DOUBLE 映射到 8 字节。

此选项适用于使用缺省规范（不带显式字节大小）声明的所有变量，例如 REAL XYZ（生成 64 位 XYZ）。此外，所有的单精度 REAL 常量将被提升为 REAL*8 常量。

请注意，INTEGER 和 LOGICAL 被视为相同类型，COMPLEX 映射为两个 REAL。此外，对待 DOUBLE COMPLEX 的方式与映射 DOUBLE 的方式相同。

3.4.184 -xunboundsym={yes|no}

指定程序是否包含对动态绑定符号的引用。

-xunboundsym=yes 表示程序包含对动态绑定符号的引用。

-xunboundsym=no 表示程序不包含对动态绑定符号的引用。

缺省值为 -xunboundsym=no。

3.4.185 –xunroll=*n*

与 -unroll=*n* 等效。

3.4.186 -xvector[=*a*]

启用向量库函数调用自动生成，或在支持 SIMD（Single Instruction Multiple Data，单指令多数据）的 x86 处理器上启用 SIMD 指令生成。使用此选项时，必须通过指定 -fround=nearest 来使用缺省的舍入模式。

-xvector 选项需要 -xO3 或更高的优化级别。如果优化级别未指定或低于 -xO3，编译将不会继续，同时会发出消息。

下表列出了 *a* 的可能值。no% 前缀可禁用关联的子选项。

表 3-23 -xvector 子选项

值	含义
[no%]lib	(*Oracle Solaris*) 允许编译器将循环内的数学库调用转换为对等效向量数学例程的单个调用（如果能够进行此类转换）。此类转换可提高那些循环计数较大的循环的性能。使用 no%lib 可以禁用此选项。
[no%]simd	(*SPARC*) 对于 -xarch=sparcace 和 -xarch=sparcaceplus，指示编译器使用浮点和整数 SIMD 指令来提高某些循环的性能。与其他 SPARC 平台的该项相对，-xvector=simd 在指定了任何 -xvector 选项（-xvector=none 和 -xvector=no%simd 除外）的 -xarch=sparcace 和 -xarch=sparcaceplus 下始终有效。此外，对于 -xvector=simd，-O 需要大于 3，否则会被跳过而不显示任何警告。 对于所有其他 -xarch 值，指示编译器使用可视指令集 [VIS1、VIS2、ViS3 等] SIMD 指令来提高某些循环的性能。从根本上来说，使用显式 -xvector=simd 选项，编译器将执行循环转换，从而允许生成特殊向量化的 SIMD 指令以减少循环迭代数。仅当 -O 大于 3 并且 -xarch 为 sparcvis3 及以上时，-xvector=simd 选项才有效。否则，将跳过 -xvector=simd 而不显示任何警告。
[no%]simd	(*x86*) 指示编译器使用本机 x86 SSE SIMD 指令来提高某些循环的性能。在 x86 中，缺省情况下以可产生有利结果的优化级别 3 和更高级别使用流扩展。可以使用 no%simd 禁用此选项。 仅当目标体系结构中存在流扩展（即目标 ISA 至少为 SSE2）时，编译器才会使用 SIMD。例如，可在现代平台中指定 -xtarget=woodcrest、-xarch=generic64、-xarch=sse2、-xarch=sse3 或 -fast 来使用它。如果目标 ISA 没有流扩展，子选项将无效。
%none	完全禁用此选项。
yes	此选项已过时；请改为指定 -xvector=lib。
no	此选项已过时；请改为指定 -xvector=%none。

在 x86 平台上的缺省值为 -xvector=simd，在 SPARC 平台上的缺省值为 -xvector=%none。如果指定不带子选项的 -xvector，则在 x86 Solaris、SPARC Solaris 和 Linux 平台上，编译器将分别采用 -xvector=simd,lib、-xvector=lib 和 -xvector=simd。

在装入步骤中，编译器包含 libmvec 库。

如果使用单独的命令进行编译和链接，应确保也在链接 CC 命令中使用 -xvector。

3.4.187 –ztext

在不重定位的情况下仅生成纯库。

-ztext 的一般用途是验证所生成的库是否为纯文本；指令都是与位置无关的代码。因此，它通常与 -G 和 -pic 一起使用。

使用 -ztext 时，如果 ld 在 *text* 段中找到了不完整的重定位，则不会生成库。如果它在 *data* 段中找到了不完整的重定位，则通常会生成库；数据段是可写入的。

不使用 -ztext 时，ld 会生成库，与有无重定位无关。

如果您不知道对象文件是否是使用 -pic 生成的，则一种典型用法是利用源文件和对象文件生成库。

示例： 利用源文件和对象文件生成库：

```
demo% f95 -G -pic -ztext -o MyLib -hMyLib a.f b.f x.o y.o
```

另一种用法是确认代码是否与位置无关： 不带 -pic 进行编译，但确认是否为纯文本。

示例：确认是否为纯文本 – 即使不带 -pic 进行编辑：

```
demo% f95 -G -ztext -o MyLib -hMyLib a.f b.f x.o y.o
```

选项 -ztext 和 -xprofile=collect 不应同时使用。-ztext 会阻止只读存储中与位置有关的符号重定位，而 -xprofile=collect 会在可写存储中生成静态初始化且与位置有关的符号重定位。

如果使用 -ztext 进行编译时 ld 不生成库，则可以在不使用 -ztext 的情况下重新编译，此时 ld 将生成库。使用 -ztext 生成失败意味着无法对库的一个或多个组件共享，不过，也许能共享某些其他组件。此时就产生了性能问题，这最好由您 – 程序员来解决。

♦ ♦ ♦ 第 4 章

Solaris Studio Fortran 的功能与扩展

本章介绍了 Oracle Solaris Studio Fortran 编译器 `f95` 中的一些主要语言功能和扩展。

4.1 源语言功能

`f95` 编译器提供标准 Fortran 的以下源语言功能和扩展。

4.1.1 续行限制

`f95` 允许 999 个续行（1 个初始行和 999 个续行）。对于固定格式，标准 Fortran 95 允许 19 个续行；对于自由格式，允许 39 个续行。

4.1.2 固定格式源代码行

在固定格式源代码中，行的长度可以超过 72 个字符，但忽略第 73 列以后的任何内容。标准 Fortran 95 仅允许 72 个字符长的行。

4.1.3 制表符格式

`f95` 固定格式源代码文本的定义如下：

- 如果第 1 列至第 6 列的任一列中有制表符，都会使该行成为制表符格式的源代码行。
- 制表符前面可以有注释指示符或语句编号。
- 如果制表符是第一个非空字符，会出现以下情况：
  - 如果制表符后面的字符不是非零数字，则制表符后面的文本是初始行。
  - 如果第一个制表符后面是非零数字，则该行是续行。非零数字后面的文本是语句的下一个部分。

对于固定格式，f95 的缺省最大行长度是 72 列；对于自由格式，则为 132 列。使用 -e 编译器选项，可将固定格式源代码中的行扩展到 132 列。

示例： 左侧的制表符格式源代码的处理方式如右侧所示。

```
!^IUses of tabs                              !       Uses of tabs
^ICHARACTER *3 A = 'A'                               CHARACTER *3 A = 'A'
^IINTEGER B = 2                                      INTEGER B = 2
^IREAL C = 3.0                                       REAL C = 3.0
^IWRITE(*,9) A, B, C                                 WRITE(*,9) A, B, C
9^IFORMAT(1X, A3,                            9       FORMAT(1X, A3,
^I1 I3,                                             1 I3,
^I2 F9.1 )                                          2 F9.1 )
^IEND                                                END
```

在以上示例中，"^I" 代表制表符，以 "1" 和 "2" 开头的行是续行。显示此代码的目的在于说明各种制表符情形，而不是提倡任一样式。

f95 中的制表符会强制填充剩余的行直到第 72 列。如果制表符出现在持续到下一行的字符串中，则可能会引发意外结果：

源文件：

```
^Iprint *, "Tab on next line
^I1this  continuation line starts with a tab."
^Iend
```

运行代码：

```
Tab on next line                                        this  continuation
 line starts with a tab.
```

将制表符格式与 –f77 选项结合使用时，行的长度将不受 132 个字符限制。行的长度可以更长。

4.1.4 采用的源代码格式

f95 采用的源代码格式取决于选项、指令和后缀。

具有 .f 或 .F 后缀的文件采用固定格式。带 .f90、.f95、.F90 或 .F95 后缀的文件假定采用自由格式。

表 4-1 F95 源代码格式命令行选项

选项	操作
-fixed	将所有源文件解释为 Fortran 固定格式
-free	将所有源文件解释为 Fortran 自由格式

如果使用 -free 或 -fixed 选项，则它覆盖文件名后缀。如果使用 !DIR$ FREE 或 !DIR$ FIXED 指令，则它覆盖选项和文件名后缀。

4.1.4.1 混合格式

允许混合使用某些源代码格式。

- 在同一个 f95 命令中，一些源文件可以是固定格式，而另一些源文件可以是自由格式。
- 在同一个文件中，可以通过使用 !DIR$ FREE 和 !DIR$ FIXED 指令将自由格式与固定格式混合使用。
- 在同一个程序单元中，可以将制表符格式与自由格式或固定格式混合使用。

4.1.4.2 大小写

缺省情况下，Solaris Studio Fortran 95 不区分大小写。这意味着，变量 AbcDeF 的处理方式与将其拼写为 abcdef 时相同。要让编译器区别处理大写字母和小写字母，请使用 -U 选项进行编译。

4.1.5 限制和缺省值

- 一个 Fortran 程序单元最多可定义 65,535 个派生类型和 16,777,215 个不同的常量。
- 变量和其他对象的名称最多可包含 127 个字符，标准长度为 31 个字符。

4.2 数据类型

本节介绍 Fortran 数据类型的功能和扩展。

4.2.1 布尔类型

f95 支持布尔类型的常量和表达式。但是，没有布尔变量或数组，也没有布尔类型语句。

4.2.1.1 控制布尔类型的各种规则

- 对于屏蔽操作，按位逻辑表达式具有布尔结果；它的每个位都是对相应操作数位进行一个或多个逻辑运算的结果。

- 用于二进制算术运算符和关系运算符：
  - 如果一个操作数是布尔型，则在执行运算时不进行转换。
  - 如果两个操作数均是布尔型，则在执行运算时就当它们是整数一样。
- 用户指定的函数均不能生成布尔结果，但某些（非标准的）内部函数可以。
- 布尔和逻辑类型具有以下差异：
  - 变量、数组和函数可以是逻辑类型，但它们不能是布尔类型。
  - 可以使用 LOGICAL 语句，但不能使用 BOOLEAN 语句。
  - 逻辑变量、常量或表达式仅可以表示两个值：.TRUE. 或 .FALSE.。布尔型变量、常量或表达式可以表示任何二进制值。
  - 逻辑型实体在算术表达式、关系表达式或按位逻辑表达式中无效。布尔型实体在所有 3 种表达式中都有效。

4.2.1.2 布尔常量的替代格式

f95 允许使用以下替代格式（没有二进制）的布尔常量（八进制、十六进制或霍尔瑞斯）。不能将变量声明为布尔型。标准 Fortran 不允许使用这些格式。

八进制

*dddddd*B，其中 *d* 是任意八进制数字

- 可以使用字母 B 或 b。
- 可以是 1 至 11 个八进制数字（0 至 7）。
- 11 个八进制数字表示完整的 32 位字，最左侧的数字可以是 0、1、2 或 3。
- 每个八进制数字指定 3 位的值。
- 最后一个（最右侧的）数字指定最右侧 3 位（位 29、30 和 31）的内容。
- 如果位数不足 11 个，则该值右对齐，即它表示字最右侧的位：位 n 至 31。其他位为 0。
- 忽略空格。

在 I/O 格式规范中，字母 B 表示二进制数字；而在其他地方则表示八进制数字。

十六进制

X'*ddd*' 或 X"*ddd*"，其中 *d* 是任意十六进制数字

- 可以包含 1 至 8 个十六进制数字（0 至 9，A-F）。
- 任何字母既可以是大写也可以是小写 (X, x, A-F, a-f)。
- 数字必须用撇号或引号括起来。
- 忽略空格。
- 十六进制数字可以以 + 或 - 符号开头。

- 8 个十六进制数字表示一个完整的 32 位字，等效的二进制数字对应于 32 位字中每个位的内容。
- 如果位数不足 8 个，则该值右对齐，即它表示字最右侧的位：位 n 至 31。其他位为 0。

霍尔瑞斯

接受的霍尔瑞斯数据格式为：

*n*H...	'...'H	"..."H
*n*L...	'...'L	"..."L
*n*R...	'...'R	"..."R

上面的 "..." 是字符串，*n* 是字符数。

- 霍尔瑞斯常量是布尔类型。
- 如果任何字符常量是按位逻辑表达式，则该表达式的计算结果为霍尔瑞斯型。
- 霍尔瑞斯常量可以包含 1 至 4 个字符。

示例： 八进制和十六进制常量。

布尔常量	32 位字的内部八进制数
0B	00000000000
77740B	00000077740
X"ABE"	00000005276
X"-340"	37777776300
X'1 2 3'	00000000443
X'FFFFFFFFFFFFFFFF'	37777777777

示例： 赋值语句中的八进制和十六进制数。

```
i = 1357B
j = X"28FF"
k = X'-5A'
```

在算术表达式中使用八进制或十六进制常量可产生未定义的结果，并且不会生成语法错误。

4.2.1.3 布尔常量的替代上下文

f95 允许在非 DATA 语句中使用 BOZ 常量。

B'*bbb*'	O'*ooo*'	Z'*zzz*'
B"*bbb*"	O"*ooo*"	Z"*zzz*"

如果将它们赋值给实数变量，则不进行类型转换。

标准 Fortran 只允许在 DATA 语句中使用它们。

4.2.2 数值数据类型的缩写大小表示法

f95 允许在声明语句、函数语句和 IMPLICIT 语句中使用以下非标准的类型声明格式。第一列中的格式虽然已被广泛使用，但它们是非标准的 Fortran 格式。第二列中的种类数字可能会因供应商不同而异。

表 4-2 数值数据类型的大小表示法

非标准	声明符	简短格式	含义
INTEGER*1	INTEGER(KIND=1)	INTEGER(1)	有符号的单字节整数
INTEGER*2	INTEGER(KIND=2)	INTEGER(2)	有符号的双字节整数
INTEGER*4	INTEGER(KIND=4)	INTEGER(4)	有符号的 4 字节整数
LOGICAL*1	LOGICAL(KIND=1)	LOGICAL(1)	单字节逻辑值
LOGICAL*2	LOGICAL(KIND=2)	LOGICAL(2)	双字节逻辑值
LOGICAL*4	LOGICAL(KIND=4)	LOGICAL(4)	四字节逻辑值
REAL*4	REAL(KIND=4)	REAL(4)	IEEE 单精度 4 字节浮点值
REAL*8	REAL(KIND=8)	REAL(8)	IEEE 双精度 8 字节浮点值
REAL*16	REAL(KIND=16)	REAL(16)	IEEE 四精度 16 字节浮点值
COMPLEX*8	COMPLEX(KIND=4)	COMPLEX(4)	单精度复数（每个部分 4 个字节）
COMPLEX*16	COMPLEX(KIND=8)	COMPLEX(8)	双精度复数（每个部分 8 个字节）
COMPLEX*32	COMPLEX(KIND=16)	COMPLEX(16)	四精度复数（每个部分 16 个字节）

4.2.3 数据类型的大小和对齐

存储和对齐始终以字节为单位。可以划分为单字节的值按字节对齐。

类型的大小和对齐取决于各种编译器选项和平台以及变量的声明方式。COMMON 块中的缺省最大对齐位置是 4 字节边界。

使用特殊选项（如 -aligncommon、-f、-dalign、-dbl_align_all、- xmemalign 和 -xtypemap）进行编译，可以更改缺省的数据对齐和存储分配。本手册中的缺省描述假定这些选项无效。

《*Fortran* 编程指南》提供了有关某些平台上数据类型和对齐方式特例的其他信息。

下表汇总了缺省的大小和对齐，并忽略类型和选项的其他方面。

表 4-3 缺省的数据大小和对齐（以字节为单位）

Fortran 数据类型	大小	缺省对齐	COMMON 中的对齐
`BYTE X`	1	1	1
`CHARACTER X`	1	1	1
`CHARACTER*`*n* `X`	n	1	1
`COMPLEX X`	8	4	4
`COMPLEX*8 X`	8	4	4
`DOUBLE COMPLEX X`	16	8	4
`COMPLEX*16 X`	16	8	4
`COMPLEX*32 X`	32	8/16	4
`DOUBLE PRECISION X`	8	8	4
`REAL X`	4	4	4
`REAL*4 X`	4	4	4
`REAL*8 X`	8	8	4
`REAL*16 X`	16	8/16	4
`INTEGER X`	4	4	4
`INTEGER*2 X`	2	2	2
`INTEGER*4 X`	4	4	4
`INTEGER*8 X`	8	8	4
`LOGICAL X`	4	4	4
`LOGICAL*1 X`	1	1	1
`LOGICAL*2 X`	2	2	2
`LOGICAL*4 X`	4	4	4
`LOGICAL*8 X`	8	8	4

请注意下列内容：

- `REAL*16` 和 `COMPLEX*32`：在 64 位环境（使用 `-m64` 进行编译）中，缺省对齐位置是 16 字节（而非 8 字节）边界，如表中 8/16 所示。该数据类型通常称为四精度。
- 数组和结构按照其元素或字段对齐。数组对齐方式与数组元素相同。结构对齐方式与具有最宽对齐的字段相同。

选项 `-f` 或 `-dalign` 可强制将所有 8、16 或 32 字节数据与 8 字节边界对齐。选项 `-dbl_align_all` 可使所有数据与 8 字节边界对齐。如果程序依赖于这些选项的使用，则可能无法进行移植。

4.3 Cray 指针

Cray 指针是一个变量，其值是另一个实体（称为指针对象）的地址。

`f95` 支持 Cray 指针；标准 Fortran 95 不支持。

4.3.1 语法

Cray `POINTER` 语句采用以下格式：

```
POINTER  ( pointer_name, pointee_name [array_spec] ), …
```

其中，*pointer_name*、*pointee_name* 和 *array_spec* 如下所示：

pointer_name
: 指向相应 *pointee_name* 的指针。
pointer_name 包含 *pointee_name* 的地址。必须是一个标量变量名称（但不是派生类型）。不能是常量、结构名称、数组或函数。

pointee_name
: 指向相应 *pointer_name* 的指针对象
必须是： 变量名、数组声明符或数组名称

array_spec
: 如果 *array_spec* 存在，则它必须是显形（常量或非常量边界）或者假定大小。

例如，可以声明指向两个指针对象的 Cray 指针：

```
POINTER ( p, b ),  ( q, c )
```

以上示例声明 Cray 指针 `p` 及其指针对象 `b` 以及 Cray 指针 `q` 及其指针对象 `c`。

还可以声明指向数组的 Cray 指针：

```
POINTER ( ix, x(n, 0:m) )
```

以上示例声明 Cray 指针 `ix` 及其指针对象 `x`；并将 `x` 声明为 `n` X `(m+1)` 维数组。

4.3.2 Cray 指针的用途

通过将变量与存储块中的特定位置动态关联起来，您可以使用指针访问用户管理的存储。

Cray 指针允许访问绝对内存地址。

4.3.3 声明 Cray 指针和 Fortran 95 指针

Cray 指针声明如下：

`POINTER` (*pointer_name*, *pointee_name* [*array_spec*])

Fortran 95 指针声明如下：

`POINTER` *object_name*

不能混用这两种类型的指针。

4.3.4 Cray 指针的功能

- 无论何时引用指针对象，`f95` 均使用当前的指针值作为指针对象的地址。
- Cray 指针类型语句声明指针和指针对象。
- Cray 指针为 Cray 指针类型。
- 在 32 位处理器中，Cray 指针的值占用一个存储单元；在 64 位处理器中，Cray 指针的值占用两个存储单元。
- Cray 指针可以出现在 `COMMON` 列表中，也可以作为哑元参数。
- 在定义 Cray 指针的值之前，Cray 指针对象没有地址。
- 如果将数组命名为指针对象，则该数组称为指针对象数组。

  其数组声明符可以出现在：

  - 单独的类型语句
  - 单独的 `DIMENSION` 语句中
  - 指针语句本身

  如果数组声明符在子程序中，则维数赋值可以引用：

  - COMMON 块中的变量或
  - 作为哑元参数的变量

  每个维的大小是在进入子程序时计算的，而不是在引用指针对象时计算的。

4.3.5 Cray 指针的限制

- *pointee_name* 不得是类型为 `CHARACTER*(*)` 的变量。
- 如果 *pointee_name* 是数组声明符，则它必须是显形（常量或非常量边界）或假定大小。

- 不允许使用 Cray 指针数组。
- Cray 指针不能：
  - 是由另一个 Cray 指针或 Fortran 指针指向的指针。
  - 是结构的组件。
  - 声明为任何其他数据类型。

  Cray 指针不能出现在：
  - PARAMETER 语句或包含 PARAMETER 属性的类型声明语句中。
  - DATA 语句中。

4.3.6 Cray 指针对象的限制

- Cray 指针对象不能出现在 SAVE、DATA、EQUIVALENCE、COMMON 或 PARAMETER 语句中。
- Cray 指针对象不能是哑元参数。
- Cray 指针对象不能是函数结果变量。
- Cray 指针对象不能是结构名称或结构组件。
- Cray 指针对象不能是可终结型。

4.3.7 Cray 指针的用法

可以将 Cray 指针赋值如下：

- 设置为绝对地址

  示例：`q = 0`
- 赋值给整数变量、加或减表达式或从整数变量、加或减表达式中赋值

  示例：`p = q + 100`
- Cray 指针不是整数。不能将它们赋值给实数变量。
- LOC 函数（非标准）可用于定义 Cray 指针。

  示例： `p = LOC( x )`

示例： 按上述方式使用 Cray 指针。

```
SUBROUTINE  sub ( n )
COMMON pool(100000)
INTEGER blk(128), word64
REAL a(1000), b(n), c(100000-n-1000)
POINTER ( pblk, blk ), (ia, a ), ( ib, b ), &
        ( ic, c ), ( address, word64 )
DATA address / 64 /
pblk = 0
ia = LOC( pool )
ib = ia + 4000
```

```
ic = ib + n
...
```

有关以上示例的说明：

- word64 引用绝对地址 64 的内容
- blk 是占用内存前 128 个字的数组
- a 是块公用区中长度为 1000 的数组
- b 跟在 a 的后面，其长度为 n
- c 跟在 b 的后面
- a、b 和 c 与 pool 相关联
- word64 与 blk(17) 相同，因为 Cray 指针是字节地址，而且 blk 的每个整数元素都是 4 字节长

4.4 STRUCTURE 和 UNION (VAX Fortran)

为帮助迁移用传统 FORTRAN 77 编写的程序，f95 接受 VAX Fortran STRUCTURE 和 UNION 语句，它是 Fortran 95 中“派生类型”的前身。有关语法的详细信息，请参见《*FORTRAN 77 Language Reference*》手册。

STRUCTURE 中的字段声明可以是以下内容之一：

- 子结构 – 另一个 STRUCTURE 声明或一个先前定义的记录。
- UNION 声明。
- TYPE 声明，它可以包含初始值。
- 具有 SEQUENCE 属性的派生类型。（这是 f95 所特有的。）

与传统 f77 编译器相同，POINTER 语句不能用作域声明。

f95 还允许：

- "." 或 "%" 均可用作结构域非关联化符号：struct.field 或 struct%field。
- 结构可以出现在格式化 I/O 语句中。
- 可以在 PARAMETER 语句中初始化结构；格式与派生类型初始化相同。
- 结构可以作为派生类型中的组件，但必须使用 SEQUENCE 属性声明派生类型。

4.5 无符号整数

Fortran 编译器接受新的数据类型 UNSIGNED 作为对该语言的一种扩展。UNSIGNED 接受四个 KIND 参数值： 1、2、4 和 8，分别对应于 1、2、4 和 8 字节无符号整数。

无符号整型常量的形式是：数字串后跟大写或小写字母 U，再后跟一个下划线和种类参数（可选）。下面的示例显示了无符号整型常量的最大值：

```
255u_1
65535u_2
4294967295U_4
18446744073709551615U_8
```

如果没有使用种类参数 (12345U)，则缺省值与缺省整数相同。缺省值为 U_4，但可以使用 -xtypemap 选项更改它，这会更改缺省无符号整数的种类类型。

可使用 UNSIGNED 类型说明符声明无符号整型变量或数组：

```
UNSIGNED U
UNSIGNED(KIND=2) :: A
UNSIGNED*8 :: B
```

4.5.1 算术表达式

- 二进制运算（如 + - * /）不能混合使用有符号操作数和无符号操作数。即，如果将 U 声明为 UNSIGNED 并且 N 是有符号的 INTEGER，则 U*N 是非法的。
  - 可使用 UNSIGNED 内部函数将二进制运算中的混合操作数组合起来，例如 U*UNSIGNED(N)。
  - 有一种例外情况是，如果一个操作数是无符号整数，而另一个操作数是具有正值或零值的有符号整数常量表达式，则结果是无符号整数。
  - 此类混合表达式的结果的类型，是操作数最常见的类型。

  有符号值的幂是有符号的值；而无符号值的幂是无符号的值。
- 无符号值的一元负值是无符号的值。
- 无符号操作数可以与实数、复数操作数任意混合使用。（无符号操作数不能与区间操作数混合使用。）

4.5.2 关系表达式

可以使用内部关系运算来比较有符号和无符号整型操作数。其结果基于未修改的操作数的值。

4.5.3 控制构造

- CASE 构造接受无符号整数作为条件表达式。
- 不允许将无符号整数作为 DO 循环控制变量，也不允许在算术 IF 控制表达式中使用它。

4.5.4 输入/输出构造

- 可以使用 I、B、O 和 Z 编辑描述符来读取和写入无符号整数。
- 还可以使用列表控制和名称列表 I/O 读取和写入无符号整数。无符号整数在列表控制或名称列表 I/O 下的写入格式与用于带符号正整数的格式相同。
- 也可以使用未格式化 I/O 读取或写入无符号整数。

4.5.5 内部函数

- 允许在内部函数中使用无符号整数，但 `SIGN` 和 `ABS` 除外。
- 新的内部函数 `UNSIGNED` 与 `INT` 类似，但生成无符号类型的结果。格式为 `UNSIGNED(`*v* `[,`*kind*`] )`。
- 另一个新的内部函数 `SELECTED_UNSIGNED_KIND(` *var*`)` 返回 *var* 的种类参数。
- 内部函数不允许同时使用有符号整型操作数和无符号整型操作数，但 `MAX` 和 `MIN` 函数除外。仅当至少有一个 `REAL` 类型的操作数时，才允许这两个函数同时使用有符号整型操作数和无符号整型操作数。
- 无符号数组不能作为数组内部函数的参数。

4.6 Fortran 200x 功能

Oracle Solaris Studio Fortran 编译器的此发行版中引入了 Fortran 2003 和 Fortran 2008 标准中的许多新功能。有关详细信息，请参阅相应的 Fortran 标准。

4.6.1 与 C 之间的互操作性

新的 Fortran 标准提供了以下内容：

- 一种引用 C 语言过程的方法（相反的功能是，一种指定可从 C 函数中引用 Fortran 子程序的方法），以及
- 一种声明与外部 C 变量相链接的全局变量的方法

`ISO_C_BINDING` 模块提供了对命名常量的访问，这些命名常量是种类类型参数，它们代表了与 C 类型兼容的数据。

该标准还引入了 `BIND(C)` 属性。如果 Fortran 派生类型具有 `BIND` 属性，则它可以与 C 之间进行互操作。

Fortran 编译器实现了 Fortran 标准第 15 章中介绍的功能。第 15 章中定义的所有内部函数均已实现。如标准中第 4 章所述，Fortran 还提供了用于定义与 C 类型对应的派生类型和枚举的工具。

4.6.2 IEEE 浮点异常处理

在 Fortran 语言中，新的内部模块 IEEE_ARITHMETIC 和 IEEE_FEATURES 提供了对异常和 IEEE 算法的支持。对这些功能提供完整支持的是：

```
USE, INTRINSIC :: IEEE_ARITHMETIC

USE, INTRINSIC :: IEEE_FEATURES
```

INTRINSIC 关键字是 Fortran 2003 中的新增功能。这些模块定义了一组派生类型、常量、舍入模式、查询函数、基本函数、种类函数、基本和非基本子例程。有关详细信息，请参见 Fortran 2003 标准的第 14 章。

4.6.3 命令行参数内部函数

Fortran 2003 标准引入了三个新的内部函数，用来处理命令行参数和环境变量。包括：

- GET_COMMAND(*command, length, status*)
  以 *command* 返回调用该程序的整个命令行。
- GET_COMMAND_ARGUMENT(*number, value, length, status*)
  以 *value* 返回命令行参数。
- GET_ENVIRONMENT_VARIABLE(*name, value, length, status, trim_name*)
  返回环境变量的值。

4.6.4 PROTECTED 属性

现在，Fortran 编译器接受 Fortran 2003 的 PROTECTED 属性。PROTECTED 对模块实体的使用进行了限制。具有 PROTECTED 属性的对象只能在声明这些对象的模块中定义。

4.6.5 Fortran 2003 异步 I/O

编译器可识别 I/O 语句中的 ASYNCHRONOUS 说明符：

```
ASYNCHRONOUS=['YES' | 'NO']
```

此语法是在 Fortran 2003 标准第 9 章中提出的。在与 WAIT 语句结合使用的情况下，允许程序员指定可能与计算重叠的 I/O 进程。虽然编译器可识别 ASYNCHRONOUS='YES'，但此标准不需要实际的异步 I/O。在编译器的此发行版中，I/O 始终是同步的。

4.6.6 扩展的 ALLOCATABLE 属性

Fortran 2003 扩展了 ALLOCATABLE 属性所允许的数据实体。以前，仅限本地存储的数组变量使用该属性。现在，允许将它用于：

- 结构的数组组件
- 哑元数组
- 数组函数结果
- CLASS 类型说明符中的多态实体

在可分配实体可能与存储关联的所有位置，仍然禁止使用可分配实体：COMMON 块和 EQUIVALENCE 语句。可分配数组组件可以以 SEQUENCE 类型出现，但在 COMMON 和 EQUIVALENCE 中禁止使用此类型的对象。

4.6.7 VALUE 属性

f95 编译器接受 Fortran 2003 VALUE 类型声明属性。

如果使用此属性指定子程序哑元输入参数，则表明“按值”传递实际参数。以下示例说明如何将 VALUE 属性用于一个 C 主程序，该主程序将文字值作为参数来调用 Fortran 子程序：

```
C code:
#include <stdlib.h>
int main(int ac, char *av[])
{
    to_fortran(2);
}

Fortran code:
       subroutine to_fortran(i)
       integer, value :: i
       print *, i
       end
```

4.6.8 Fortran 2003 流 I/O

Fortran 2003 标准定义了一个新的“流”I/O 方案。流 I/O 访问将数据文件作为可按从 1 开始的正整数编址的持续字节序列来处理。可以连接数据文件用于有格式访问或无格式访问。

可以在 OPEN 语句中使用 ACCESS='STREAM' 说明符来声明流 I/O 文件。字节地址文件定位要求 READ 或 *WRITE* 语句中有 POS=scalar_integer_expression 说明

符。INQUIRE 语句接受 ACCESS='STREAM'、说明符 STREAM=*scalar_character_variable* 和 POS=*scalar_integer_variable*。

4.6.9 Fortran 2003 IMPORT 语句

IMPORT 语句指定主机作用域单元中可由主机关联访问的实体。仅在接口主体中允许使用。

4.6.10 Fortran 2003 FLUSH I/O 语句

f95 编译器接受 Fortran 2003 FLUSH 语句。FLUSH 语句使写入外部文件的数据可用于其他进程；通过除 Fortran 以外的其他方法使外部文件中的数据可用于 READ 语句。

4.6.11 Fortran 2003 POINTER INTENT 功能

现在，Fortran 编译器支持 POINTER 哑元参数的 INTENT 属性：可以为指针 dummy 指定 INTENT(IN)、INTENT(OUT) 或 INTENT(INOUT)。

例如，

```
subroutine sub(P)
integer, pointer, intent(in) :: p
...

end
```

指针的 INTENT 属性应用于指针，而非指针所指向的元素，因此对于 INTENT(IN) 指针，以下语句是非法的，因为这些语句将修改指针：

```
 p => t
 allocate(p)
 deallocate(p)
```

但以下语句对于 INTENT(IN) 指针是合法的，因为它修改指针所指向的元素：

```
 p = 400
```

4.6.12 Fortran 2003 中增强的数组构造函数

数组构造函数中允许使用方括号代替 (/ 和 /)：

```
X = [ 3.2, 4.01, 6.5 ]
```

Fortran 2003 标准允许使用方括号作为数组构造函数。这可能会与区间常量冲突。如果在不带 `-xia` 选项（或用于启用区间运算的类似选项）的情况下使用方括号，这些方括号将被视为数组构造函数。如果使用 `-xia` 选项，这些方括号将被视为常量。区间用户应继续使用 `(/` 和 `/)` 数组构造函数，以避免出现编译错误。

数组构造函数中的数组组件可以具有以下两种格式：

type-spec `::`

或

`[`*type-spec* `::]` *ac-value-list*

当提供了可选的 *type-spec* 时，各个数组组件的类型和种类不必相同，只要数组组件的类型符合 *type-spec* 即可。

type-spec 可以是内部类型或派生类型。

4.6.13 面向对象的 Fortran 支持

Fortran 2003 提供了对多态性的完整支持。

4.6.14 FINAL 子例程支持

支持 FINAL 子例程，这是一种 Fortran 2003 功能。

4.6.15 过程指针支持

支持过程指针，这是一种 Fortran 2003 功能。

4.6.16 其他 Fortran 2003 和 Fortran 2008 功能

有关详细信息，请参阅已发布的 Fortran 2003 和 Fortran 2008 标准。

- 2003 对可分配数组的扩展 – 赋值重新分配及可分配的标量。
- 2003 对 `ALLOCATE/DEALLOCATE` 语句的扩展 – `ERRMSG` 和 `SOURCE`。
- 2003 对 `MOVE_ALLOC` 内部函数的扩展。

- 2003 对带重映射的指针赋值的扩展。
- 2003 扩展：MIN/MAX、MIN/MAXVAL 和 MIN/MAXLOC，使用字符参数。
- 2003 内部函数 IS_IOSTAT_END、IS_IOSTAT_EOR、NEW_LINE。
- 2003 内部函数 SELECTED_CHAR_KIND。
- 内部函数 SYSTEM_CLOCK 的 COUNT_RATE 参数的 2003 REAL 类型。
- 2003 对复合 SQRT 内部函数结果的新限制。
- 2008：使用空指针作为缺少的可选参数。
- 2008 的位内部函数：BGE, BGT, BLE, BLT, DSHIFTL, DSHIFTR, LEADZ, POPCNT, POPPAR, TRAILZ, MASKL, MASKR, SHIFTA, SHIFTL, SHIFTR, MERGE_BITS, IALL, IANY, IPARITY。
- 增强的结构构造函数：使用组件名称构造结构常量。
- 模块派生类型和组件上的增强 PUBLIC/PRIVATE 访问控制。
- 更多 Fortran 2008 数学内部函数支持。在 x86 平台上，除了 ERFC_SCALED、NORM2 以及某些 REAL*16 变量外，现在多数 Fortran 2008 数学内部函数都受支持。
- 不带组件的派生类型。
- KIND 参数已添加到 ICHAR、IACHAR、ACHAR、SHAPE、UBOUND、LBOUND、SIZE、MINLOC、MAXLOC、COUNT、LEN、LEN_TRIM、INDEX、SCAN 和 VERIFY 内部函数中。
- BACK 参数已添加到 MINLOC 和 MAXLOC 内部函数中。
- 添加了新的内部函数 FINDLOC 和 STORAGE_SIZE。
- 新的关键字 ERRMSG、SOURCE 和 MOLD 已添加到 ALLOCATE 语句中，ERRMSG 已添加到 DEALLOCATE 语句中。
- 使用 ENUM 进行枚举。
- VOLATILE 关键字。
- 各组件上的 PUBLIC/PRIVATE 可访问性。
- 专用类型的公共实体。
- 增强的复数常量。
- Fortran 2003 ISO_FORTRAN_ENV 模块。
- 内部函数的新可选参数 KIND=。
- 名称的长度最长可为 127 个字符，但模块名称除外（它的长度不能超过 31 个字符）。
- INQUIRE 语句中的 ID= 和 PENDING= 说明符。
- 数据传送和 INQUIRE 语句中的 POS= 说明符。
- BLANK、DECIMAL、DELIM、PAD、ROUND、SIZE 说明符。
- DC、DP、RD、RC、RN、RP、RU、RZ 编辑描述符。
- USE 中的 INTRINSIC 和 NON_INTRINSIC 关键字。
- IS_IOSTAT_END 和 IS_IOSTAT_EOR 内部函数。
- 对延迟长度字符声明的支持。例如，CHARACTER (LEN=:)、POINTER :: STR。

- 支持将 TARGET 对象传递到 INTENT(IN) 指针哑元。这是一个 Fortran 2008 功能。

4.7 其他的 I/O 扩展

本节介绍一些 Fortran 95 输入/输出处理扩展，f95 编译器接受这些扩展，但它们不是 Fortran 2003 标准的一部分。某些扩展是在 Fortran 77 编译器 f77 中出现的 I/O 扩展，现在这些扩展已成为 Fortran 编译器的一部分。

4.7.1 I/O 错误处理例程

通过两个新函数，用户可以为逻辑单元上的格式化输入指定自己的错误处理例程。当检测到格式错误时，运行时 I/O 库会调用指定的由用户提供的处理程序例程，同时将数据指向输入行中导致错误的字符。处理程序例程可以提供一个新字符，并允许 I/O 操作在检测到错误的点上使用新字符继续运行；或者采用缺省的 Fortran 错误处理操作。

新例程 SET_IO_ERR_HANDLER(3f) 和 GET_IO_ERR_HANDLER(3f) 是模块子例程，这两个例程要求在调用它们的例程中使用 USE SUN_IO_HANDLERS。要详细了解这些例程，请参见手册页。

4.7.2 变量格式表达式

Fortran 77 允许用尖括号括起的任意表达式来代替具有某种格式的任何整数常量：

```
1 FORMAT( ... < expr > ... )
```

变量格式表达式不能作为 *n*H... 编辑描述符中的 *n* 出现在 ASSIGN 语句引用的 FORMAT 语句中，或者出现在并行区域内的 FORMAT 语句中。

这种功能是在 f95 中自动启用的，并且不要求使用 -f77 兼容性选项标志。

4.7.3 NAMELIST 输入格式

- 输入中的组名前面可以是 $ 或 &。& 是 Fortran 95 标准接受的唯一格式，并且是 NAMELIST 输出所写入的内容。
- 接受 $ 作为用来终止输入的符号，但以下情况除外：组中最后一个数据项是 CHARACTER 数据，此时将 $ 作为输入数据处理。
- 允许 NAMELIST 输入从记录的第一列开始。

4.7.4 二进制未格式化 I/O

使用 FORM='BINARY' 打开文件与使用 FORM='UNFORMATTED' 具有大致相同的效果，所不同的是文件中没有嵌入记录长度。如果没有此数据，则无法知道一条记录的开始或结束位置。因此，无法对 FORM='BINARY' 文件执行 BACKSPACE 操作，这是因为不知道要退格到什么位置。对 'BINARY' 文件执行 READ 操作时，将按需要读取尽可能多的数据来填充输入列表中的变量。

- WRITE 语句： 以二进制的形式将数据写入文件，并按输出列表中指定的数量传输字节。
- READ 语句： 将数据读取到输入列表中的变量，并传输该列表所要求数量的字节。因为文件中没有记录标记，所以不进行"记录结束"错误检测。检测到的唯一错误是"文件结束"或异常系统错误。
- INQUIRE 语句：对使用 FORM="BINARY" 打开的文件执行 INQUIRE 返回：

  FORM="BINARY"ACCESS="SEQUENTIAL"DIRECT="NO"FORMATTED="NO"UNFORMATTED="YES"RECL= AND NEXTREC= are undefined
- BACKSPACE 语句： 不允许使用 – 返回一个错误。
- ENDFILE 语句： 在当前位置照常截断文件。
- REWIND 语句： 将文件照常重新定位到数据的开头。

4.7.5 各种 I/O 扩展

- 在不同单元上可能出现的递归 I/O（这是因为 f95 I/O 库为 "MT-Warm"）。
- RECL=2147483646 (2^{31}-2) 是顺序格式化输出、列表式输出和名称列表输出中的缺省记录长度。
- 可以按《FORTRAN 77 Language Reference》手册中所述识别和实现 ENCODE 和 *DECODE*。
- 非前进式 I/O 是使用 ADVANCE='NO' 启用的，如下所示：

```
write(*,'(a)',ADVANCE='NO') 'n= ' read(*,*) n
```

4.8 指令

编译器指令指示编译器执行某些特殊的操作。指令又称 *pragma*。

可以将编译器指令作为一个或多个文本行插入到源程序中。每一行看起来就像注释一样，但其中包含的其他字符可将其标识为不仅仅是该编译器的注释。对于大多数其他编译器，它被处理为注释，因此具有一定的代码移植性。

有关 Fortran 指令的完整摘要，请参见附录 C, Fortran 指令摘要。

4.8.1 特殊 f95 指令行的格式

除了第 1.8 节 “命令行帮助” [18]中介绍的指令外，f95 还可识别其自己的特殊指令。这些指令使用以下语法：

```
!DIR$ d1, d2, …
```

4.8.1.1 固定格式源代码

- 将 CDIR$ 或 !DIR$ 放在第 1 至第 5 列中。
- 指令在第 7 列及后面的列中列出。
- 忽略第 72 列后面的列。
- 初始指令行的第 6 列为空。
- 连续指令行的第 6 列非空。

4.8.1.2 自由格式源代码

- 将后跟空格的 !DIR$ 放在行中的任意位置。

  !DIR$ 字符是行中的第一个非空字符（实际上就是非空白）。
- 指令在空格后面列出。
- 在初始指令行中，紧跟在 !DIR$ 之后的位置中为空格、制表符或换行符。
- 在连续指令行中，紧跟在 !DIR$ 之后的位置中为空格、制表符或换行符以外的字符。

因此，第 1 至第 5 列中的 !DIR$ 既用于自由格式源代码又用于固定格式源代码。

4.8.2 FIXED 和 FREE 指令

这些指令指定指令行后面行的源代码格式。

4.8.2.1 作用域

它们适用于所在文件的其余部分，或者在遇到下一个 FREE 或 FIXED 指令之前的部分。

4.8.2.2 用法

- 它们用于切换源文件中的源代码格式。
- 它们用于切换 INCLUDE 文件的源代码格式。您可以将指令插入在 INCLUDE 文件的开头。在处理 INCLUDE 文件后，源代码格式恢复为处理 INCLUDE 文件之前使用的格式。

4.8.2.3 限制

FREE/FIXED 指令：

- 每个指令必须单独出现在编译器指令行中（没有续行）。
- 每个指令可以出现在源代码中的任意位置。其他指令必须出现在它们所影响的程序单元中。

示例： 一个 FREE 指令。

```
!DIR$ FREE
    DO i = 1, n
        a(i) = b(i) * c(i)
    END DO
```

4.8.3 并行化指令

并行化指令是一种特殊的注释，它指示编译器尝试并行处理代码区域。Sun 和 Cray 样式并行化指令现已过时。应首选使用 OpenMP 指令和并行化模型。《*OpenMP API* 用户指南》中介绍了 OpenMP 并行化。

4.9 模块文件

在编译包含 Fortran 95 MODULE 的文件时，会为在源代码中遇到的每个 MODULE 生成模块接口文件（.mod 文件）。文件名是从 MODULE 的名称中派生的；将为 MODULE xyz 创建文件 xyz.mod（全部小写）。

编译还会为包含 MODULE 语句的源文件生成 .o 模块实现对象文件。可与模块实现对象文件以及所有其他对象文件链接在一起以创建可执行文件。

编译器在 -moddir=*dir* 标志或 MODDIR 环境变量指定的目录中创建模块接口文件和实现对象文件。如果没有指定，则编译器在当前工作目录中写入 .mod 文件。

在编译 USE *modulename* 语句时，编译器在当前工作目录中查找接口文件。使用 -M*path* 选项，可以为编译器指定其他搜索路径。必须在链接步骤的命令行中显式列出模块实现对象文件。

通常，程序员为每个文件定义一个 MODULE，并为 MODULE 及包含它的源文件指定相同的名称。但是，这并不是必需的。

在本示例中，同时编译所有的文件。模块源文件在主程序使用它们之前就已出现。

```
demo% cat mod_one.f90
 MODULE one
```

```
    ...
 END MODULE
demo% cat mod_two.f90
 MODULE two
    ...
 END MODULE
demo% cat main.f90
 USE one
 USE two
   ...
 END
demo% f95 -o main mod_one.f90 mod_two.f90 main.f90
```

编译创建以下文件：

```
mainmain.oone.modmod_one.otwo.modmod_two.o
```

下一个示例单独编译每个单元，并将它们链接在一起。

```
demo% f95 -c mod_one.f90 mod_two.f90
demo% f95 -c main.f90
demo% f95 -o main main.o mod_one.o mod_two.o
```

在编译 `main.f90` 时，编译器在当前目录中搜索 `one.mod` 和 `two.mod`。必须先编译这些文件，再编译在 `USE` 语句中引用模块的任何文件。链接步骤要求模块实现对象文件 `mod_one.o` 和 `mod_two.o` 与所有其他对象文件一起出现，以创建可执行文件。

4.9.1 搜索模块

在发行的 7.0 版的 Fortran 编译器中，可以将 `.mod` 文件存储在归档 (`.a`) 文件中。要在归档中搜索模块，必须在命令行的 `-M`*path* 标志中显式指定它。在缺省情况下，编译器并不搜索归档文件。

仅搜索与 `USE` 语句中出现的名称同名的 `.mod` 文件。例如，Fortran 语句 `USE mymod` 使编译器缺省搜索模块文件 `mymod.mod`。

在搜索过程中，编译器为在其中写入模块文件的目录指定更高的优先级。可以使用 `-moddir=`*dir* 选项标志和 `MODDIR` 环境变量对此进行控制。这意味着，如果仅指定了 `-M`*path* 选项，则首先搜索当前目录，然后再搜索 `-M` 标志上列出的目录和文件。

4.9.2 `-use=`*list* 选项标志

`-use=`*list* 标志强制将一个或多个隐式 `USE` 语句添加到每个使用该标志编译的子程序或模块子程序中。通过使用该标志，使得在库或应用程序的某个功能要求使用模块或模块文件时，不必修改源程序。

使用 -use=*module_name* 进行编译，可将 USE *module_name* 语句添加到正在编译的每个子程序或模块中。使用 -use=*module_file_name* 进行编译，与为 *module_file_name* 文件中包含的每个模块添加 USE *module_name* 具有相同的效果。

4.9.3 fdumpmod 命令

使用 fdumpmod(1) 命令可显示有关已编译模块信息文件的内容的信息。

```
demo% fdumpmod x.mod group.mod
x 1.0 v8,i4,r4,d8,n16,a4 x.mod
group 1.0 v8,i4,r4,d8,n16,a4 group.mod
```

fdumpmod 命令将在单个 .mod 文件、通过串联 .mod 文件形成的文件以及 .mod 文件的 .a 归档中显示有关模块的信息。显示的内容包含模块名称、版本号、目标体系结构，以及用来指示与模块兼容的编译选项的标志。有关详细信息，请参见 fdumpmod(1) 手册页。

4.10 内部函数

f95 支持某些内在过程，它们是超出标准的扩展。

表 4-4 非标准的内部函数

名称	定义	函数类型	参数类型	参数	注释
COT	余切	实型	实型	([X=]*x*)	P、E
DDIM	正偏差	双精度	双精度	([X=]*x*,[Y=]*y*)	P、E

注：P：名称可以作为参数传递。E：在运行时调用内部函数的外部代码。

有关内部函数（包括 Fortran 编译器可识别的 Fortran 77 内部函数）的更完整论述，请参见《*Fortran* 库参考》。

4.11 向前兼容性

以后的 f95 发行版在源代码上将与此发行版兼容。

不能保证此发行版的 f95 所生成的模块信息文件与以后的发行版兼容。

4.12 混合语言

可以将使用 C 编写的例程与 Fortran 程序结合使用 ，因为这些语言具有通用的调用约定。有关 C 和 Fortran 例程之间如何进行交互操作的详细信息，请参见《*Fortran* 编程指南》的“C/Fortran 接口”一章。

♦ ♦ ♦ 第 5 章

5 FORTRAN 77 兼容性：迁移到 Solaris Studio Fortran

不再提供单独的 Fortran 77 编译器。Oracle Solaris Studio Fortran 编译器 `f95` 可编译大多数的传统 FORTRAN 77 程序，其中包括采用以前由 Sun WorkShop `f77` 编译器编译的非标准扩展的程序。

`f95` 可直接接受很多 FORTRAN 77 功能。其他功能要求在 FORTRAN 77 兼容性模式 (`f95 -f77`) 下进行编译。

本章介绍 `f95` 接受的 FORTRAN 77 功能，并列出了那些与 `f95` 不兼容的 `f77` 功能。有关 Sun WorkShop `f77` 编译器所接受的任何非标准 FORTRAN 77 扩展的详细信息，请参见 *http://docs.sun.com/source/806-3594/index.html* 上的传统《`FORTRAN 77 Language Reference`》手册。

有关 `f95` 编译器所接受的其他 Fortran 语言扩展的信息，请参见第 4 章 Solaris Studio Fortran 的功能与扩展。

`f95` 可编译符合标准的 FORTRAN 77 程序。要确保持续的可移植性，使用非标准 FORTRAN 77 功能的程序应该迁移到符合标准的 Fortran 95/2003。使用 `-ansi` 选项进行编译，会标出程序中所有不符合标准的用法。

5.1 兼容的 `f77` 功能

`f95` 直接接受或在 `-f77` 兼容性模式下进行编译时接受传统 FORTRAN 77 编译器 `f77` 的以下非标准功能：

源代码格式

- 续行可以在第一列中以 '&' 开头。[`-f77=misc`]
- include 文件中的第一行可以是续行。[`-f77=misc`]
- 使用 `f77` 制表符格式。[`-f77=tab`]
- 制表符格式可以将源代码行扩展到第 72 列以后。[`-f77=tab`]

- 如果字符串扩展到续行上，则 f95 制表符格式不会将字符串一直填充到第 72 列。 [-f77]

I/O：

- 在 Fortran 95 中，可以使用 ACCESS='APPEND' 打开文件。
- 列表式输出使用与 f77 编译器类似的格式。[-f77=output]
- f95 允许对直接访问文件使用 BACKSPACE，但不允许使用 ENDFILE。
- f95 允许在格式编辑描述符中隐式指定字段宽度。例如，允许使用 FORMAT(I)。
- f95 将以输出格式识别 f77 转义序列（例如，\n \t \'）。[-f77=backslash]。
- f95 可识别 OPEN 语句中的 FILEOPT=。
- f95 允许使用 STATUS='KEEP' [-f77] 来关闭 SCRATCH 文件。在程序退出时，不会删除临时文件。在使用 -f77 进行编译时，也可以使用 FILE=*name* 来打开 SCRATCH 文件。
- 允许对内部文件使用直接 I/O。[-f77]
- f95 可识别 FORTRAN 77 格式编辑描述符 A、$ 和 SU。 [-f77]
- FORM='PRINT' 可以出现在 OPEN 语句中。 [-f77]
- f95 可识别传统的 FORTRAN 输入/输出语句 ACCEPT 和 TYPE。
- 使用 -f77=output 进行编译可写入 FORTRAN 77 样式的 NAMELIST 输出。
- 在仅指定 ERR=（没有 IOSTAT= 或 END= 分支）的情况下，当检测到 EOF 时，READ 将 ERR= 分支作为 END= 处理。[-f77]
- 在 OPEN 语句中，接受 VMS Fortran NAME='*filename*'。 [-f77]
- f95 接受 READ() 或 WRITE() 后面的一个额外逗号。 [-f77]
- END= 分支可以出现在具有 REC= 的直接访问 READ 中。 [-f77=input]
- 允许使用格式编辑描述符 E*w*.*d*.*e*，并将其作为 E*w*.*d*.E*e* 处理。 [-f77]
- 可以在输入语句的 FORMAT 中使用字符串。[-f77=input]
- IOSTAT= 说明符可以出现在 ENCODE/DECODE 语句中。
- 在 ENCODE/DECODE 语句中允许使用列表式 I/O。
- 在 I/O 语句中用作逻辑单元时，星号 (*) 可用于表示 STDIN 和 STDOUT。
- 数组可以出现在 FMT= 说明符中。[-f77=misc]
- PRINT 语句接受名称列表组名称。[-f77=output]
- 编译器接受 FORMAT 语句中多余的逗号。
- 在执行 NAMELIST 输入时，如果输入问号 (?)，则会以正在读取的名称列表组的名称响应。[-f77=input]

数据类型、声明和用法：

- 在程序单元中，IMPLICIT 语句可以放在单元中任何其他声明语句的后面。
- f95 接受 IMPLICIT UNDEFINED 语句。
- f95 接受 AUTOMATIC 语句（FORTRAN 77 扩展）。
- f95 接受 STATIC 语句，处理方式与 SAVE 语句一样。

- f95 接受 VAX STRUCTURE、UNION 和 MAP 语句。（请参见第 4.4 节 “STRUCTURE 和 UNION (VAX Fortran)” [157]）
- LOGICAL 和 INTEGER 变量可以互换使用。[-f77=logical]
- INTEGER 变量可以出现在条件表达式（如 DO WHILE）中。[-f77=logical]
- Cray 指针可以出现在内部函数调用中。
- f95 可接受在类型声明中使用斜杠的数据初始化。例如：REAL MHW/100.101/, ICOMX/32.223/
- f95 允许将 Cray 字符指针赋值给非指针变量，以及其他非字符指针的 Cray 指针。
- f95 允许将同一个 Cray 指针指向不同类型大小（例如，REAL*8 和 INTEGER*4）的项。
- 在将 Cray 指针声明为 POINTER 的同一程序单元中，可以将其声明为 INTEGER。INTEGER 声明将被忽略。 [-f77=misc]
- Cray 指针可以用于除法和乘法运算。[-f77=misc]
- ASSIGN 语句中的变量可以是 INTEGER*2 类型。 [-f77=misc]
- 交替的 RETURN 语句中的表达式可以是非整型。[-f77=misc]
- 具有 SAVE 属性的变量可以与 COMMON 块的元素等效。
- 相同数组的初始化函数可以是不同的类型。示例：REAL*8 ARR(5) /12.3 1, 3, 5.D0, 9/
- 名称列表项的类型声明可以放在 NAMELIST 语句后面。
- f95 接受 BYTE 数据类型。
- f95 允许将非整数用作数组下标。[-f77=subscript]
- f95 允许将关系运算符 .EQ. 和 .NE. 用于逻辑操作数。 [-f77=logical]
- f95 可接受传统的 f77 VIRTUAL 语句，并将它作为 DIMENSION 语句处理。
- 可以采用与 f77 编译器兼容的方式，对不同的数据结构进行等效处理。 [-f77=misc]
- 与 f77 编译器类似，f95 允许在 PARAMETER 语句的初始化表达式中出现许多内部函数。
- f95 允许将整数值赋值给 CHARACTER*1 变量。 [-f77=misc]
- BOZ 常量可用作指数。 [-f77=misc]
- 可以将 BOZ 常量赋值给字符变量。例如： character*8 ch ch ="12345678"X
- BOZ 常量可以用作内部函数调用的参数。[-f77=misc]
- 可以使用 DATA 语句中的整数值来初始化字符变量。变量中的第一个字符将被设置为该整数值，而字符串的其余部分将填充空白（如果字符串多于一个字符的话）。
- 可以将霍尔瑞斯字符的整数数组用作格式描述符。[-f77]。
- 如果常量折叠产生浮点异常，则在编译时不进行常量折叠。[-f77=misc]
- 在使用 -f77=misc 进行编译时，f95 将以 f77 编译器的方式，将赋值、数据和参数语句中的 REAL 常量自动提升为适当的种类（REAL*8 或 REAL*16）。 [-f77=misc]
- 在赋值 GOTO 中允许使用等价变量。 [-f77]
- 可以将非常量字符表达式赋值给数值变量。

- 使用 -f77=misc 进行编译允许将 **kind* 附加到类型声明中的变量名称之后。[-f77=misc]. 例如 REAL Y*4, X*8(21)) INTEGER FUNCTION FOO*8(J)
- 字符子串可以作为隐含 DO 目标出现在 DATA 语句中。[-f77=misc] 例如： DATA (a(i:i), i=1,n) /n*'+'/
- 括号内的整数表达式可以作为类型大小出现。例如： PARAMETER (N=2) INTEGER*(N+2) K

程序、子例程、函数和可执行语句：

- f95 不要求 PROGRAM 语句具有 *name*。
- CALL 语句可以将函数当作子例程进行调用。[-f77]
- 对函数不一定要定义其返回值。[-f77]
- 交替返回说明符（**label* 或 &*label*）可以出现在实际参数列表中，也可以出现在不同位置上。 [-f77=misc]
- %VAL 可用于类型为 COMPLEX 的参数。 [-f77=misc]
- 可以使用 %REF 和 %LOC。[-f77=misc]
- 子例程可以递归调用其自身，而无需使用 RECURSIVE 关键字对自身进行声明。[-f77=misc] 但是，还应该使用 -xrecursive 标志对执行间接递归（例程 A 调用 例程 B，而例程 B 又调用例程 A）的程序进行编译以使其正常工作。
- 即使当哑元参数列表中没有交替返回列表时，也可以调用具有交替返回的子例程。
- 如果使用 -f77=misc 进行编译，可以使用非 INTEGER 或 REAL 类型的参数来定义语句函数；实际参数将转换为语句函数所定义的类型。 [-f77=misc]
- 允许使用空实际参数。例如：CALL FOO(I,,,J) 在第一个 I 和最后一个 J 参数之间使用了两个空参数。
- f95 将函数 %LOC() 的调用作为 LOC() 调用处理。 [-f77=misc]
- 允许在其他运算符（如 ** 或 *）后面使用一元加号和一元减号。
- 即使第一个参数的类型是 COMPLEX，也允许 CMPLX() 内部函数使用第二个参数。在这种情况下，使用第一个参数的实部。 [-f77=misc]
- 允许 CHAR() 内部函数的参数超过 255，并且只生成警告而不生成错误。 [-f77=misc]
- 允许移位计数为负，并且只生成警告而不生成错误。
- 在当前目录中以及 -I 选项指定的目录中搜索 INCLUDE 文件。[-f77=misc]
- 允许进行连续的 .NOT. 运算，例如 .NOT..NOT..NOT.(I.EQ.J)。 [-f77=misc]

其他

- f95 通常不会将进度消息发送到标准输出。而 f77 编译器则发送进度消息，并显示它所编译的例程的名称。在使用 -f77 兼容性标志进行编译时，保留了这一约定。
- f77 编译器编译的程序并不捕获运算异常，而是在退出时自动调用 ieee_retrospective 来报告在执行过程中可能发生的任何异常。如果使用 -f77 标志进行编译，可模拟 f77 编译器的这种行为。缺省情况下，f95 编译器捕获遇到的第一个运算异常，并且不调用 ieee_retrospective。

- 在需要更高精度的上下文中，f77 编译器处理 REAL*4 常量的方式就好像它具有更高的精度一样。在使用 -f77 标志进行编译时，如果将 REAL*4 常量与双精度或四精度操作数一起使用，则 f95 编译器允许该常量相应地具有双精度或四精度。
- 允许在循环中重新定义 DO 循环变量。[-f77=misc]
- 显示所编译的程序单元的名称。 [-f77=misc]
- 允许在 DIMENSION 语句之后声明 DIMENSION 语句中使用的变量类型。示例：

```
SUBROUTINE FOO(ARR,G)
DIMENSION ARR(G)
INTEGER G
RETURN
END
```

有关传统 Sun WorkShop FORTRAN 77 编译器所接受的非标准语言扩展语法和语义的详细信息，请参见 http://docs.sun.com/source/806-3594/index.html 上归档的《*FORTRAN 77 Language Reference*》。

5.2 不兼容问题

下面列出了在使用此发行版的 f95 编译和测试传统 f77 程序时出现的已知不兼容问题。这些问题是由于 f95 中缺少相当的功能或者行为方式不同造成的。这些项目是传统 Sun WorkShop FORTRAN 77 编译器支持的 FORTRAN 77 非标准扩展，但当前的 f95 编译器不支持这些扩展。

源代码格式

- 在指定 -f77 选项时，如果名称长度超过 6 个字符，则会发出 ANSI 警告。

I/O：

- f95 不允许对直接访问文件使用 ENDFILE。
- f95 无法识别在直接访问 I/O 中指定记录编号时使用的 *'n* 格式：READ (2 '13) X,Y,Z
- f95 无法识别传统 f77 "R" 格式编辑描述符。
- f95 不允许在 CLOSE 语句中使用 DISP= 说明符。
- 不允许在 WRITE 语句中使用位常量。
- Fortran 95 NAMELIST 不允许使用长度可变的数组和字符串。
- 使用 RECL=1 打开的直接访问文件不能用作“流”文件。请改用 FORMAT='STREAM'。
- Fortran 95 将非法 I/O 说明符报告为错误，而 f77 只发出警告。

数据类型、声明和用法：

- f95 只允许 7 个数组下标；而 f77 允许 20 个。
- f95 不允许在 PARAMETER 语句中使用非常量。

- 不能在 CHARACTER 类型声明的初始化函数中使用整数值。
- REAL() 内部函数返回复数参数的实部，而不是将参数转换为 REAL*4。当参数为 DOUBLE COMPLEX 或 COMPLEX*32 时，这将产生不同的结果。
- Fortran 95 不允许在数组声明之前在边界表达式中使用数组元素。例如：

```
subroutine s(i1,i2)
integer i1(i2(1):10)
dimension i2(10)
...ERROR: "I2" has been used as a function,
therefore it must not be declared with the explicit-shape DIMENSION attribute.


end
```

程序、子例程、函数和语句：

- 名称的最大长度为 127 个字符。

命令行选项：

- f95 无法识别 f77 编译器选项 -dbl -oldstruct -i2 -i4，以及 -vax 的一些子选项。

f95 不支持 FORTRAN 77 库例程：

- POSIX 库。
- IOINIT() 库例程。
- 磁带 I/O 例程 topen、tclose、twrite、tread、trewin、tskipf、tstate。
- start_iostats 和 end_iostats 库例程。
- f77_init() 函数。
- f95 不允许通过使用相同名称定义用户自己的例程来绕过 IEEE_RETROSPECTIVE 子例程。

5.3 与传统 FORTRAN 77 编译的例程链接

- 要混合使用 f77 和 f95 对象二进制文件，应通过 -xlang=f77 选项利用 f95 编译器进行链接。即使主程序是 f77 程序，也使用 f95 执行链接步骤。
- 示例： 编译使用 f77 对象文件的 f95 主程序。

```
demo% cat m.f95
CHARACTER*74 :: c = 'This is a test.'
   CALL echo1( c )
END
demo% f95 -xlang=f77 m.f95 sub77.o
```

```
demo% a.out
 This is a test.
demo%
```

- 在 f95 程序中可以使用 FORTRAN 77 库和内部函数，《*Fortran* 库参考》手册中列出了这些库和内部函数。

  示例： f95 主程序调用 FORTRAN 77 库中的例程。

```
demo% cat tdtime.f95
        REAL e, dtime, t(2)
        e = dtime( t )
        DO i = 1, 100000
           as = as + cos(sqrt(float(i)))
        END DO
        e = dtime( t )
        PRINT *, 'elapsed:', e, ', user:', t(1), ', sys:', t(2)
        END
demo% f95 tdtime.f95
demo% a.out
elapsed: 0.14 , user: 0.14 , sys: 0.0E+0
demo%
```

  请参见 dtime(3F)。

5.3.1 Fortran 内部函数

Fortran 标准支持 FORTRAN 77 中没有的内部函数。《*Fortran* 库参考》手册中介绍了完整的 Fortran 内部函数集，其中包括非标准的内部函数。

如果在程序中将《*Fortran* 库参考》中列出的任何内部函数名称用作函数名称，则必须添加 EXTERNAL 语句才能让 f95 使用您的例程而不是内例程。

《*Fortran* 库参考》还列出了早期 f77 编译器发行版可识别的所有内部函数。f95 编译器可识别这些名称和内部函数。

如果使用 -f77=intrinsics 进行编译，则会将编译器识别的内部函数限定为 f77 编译器识别的那些函数，而忽略 Fortran 内部函数。

5.4 有关迁移到 f95 编译器的附加说明

- floatingpoint.h 头文件替代了 f77_floatingpoint.h，在源程序中应按如下方式使用它：

```
#include "floatingpoint.h"
```

- 格式为 `f77/` *filename* 的头文件引用应改为删除 `f77/` 目录路径。
- 对于某些使用非标准别名技术（通过为数组设置过度索引，或者重叠 Cray 或 Fortran 指针）的程序，可通过使用相应的 `-xalias` 标志进行编译而受益。请参见第 3.4.107 节 “–xalias[=*keywords*]” [91]。《*Fortran* 编程指南》中有关移植“旧式”程序的一章中举例说明了这个问题。

5.5 f77 命令

Solaris Studio 软件不再包括单独的 FORTRAN 77 编译器 `f77`。最新的发行版已将许多 FORTRAN 77 功能迁移到 Fortran 95 编译器 `f95` 中。现在，Fortran 95 编译器中提供了传统 FORTRAN 77 编译器的许多功能。当前 Solaris Studio 编译器发行版提供一个 `f77` 脚本，它用一组适当的缺省选项调用 `f95` 编译器。调用 `f77` 相当于：

```
f95 -f77 -ftrap=%none
```

如果需要链接使用先前发行的 `f77` 编译器编译的库例程，请将 `-xlang=f77` 添加到命令行中。但如果在单独的步骤中进行编译和链接，并显式指定 `-xlang=f77`、`-lM77`、`-lF77` 或 `-lsunmath`，则必须使用 `f95`（或 `f77` 脚本）进行链接，而不能使用 `cc` 或 `CC`。如果还使用 `-fast` 标志进行编译，应将 `-ftrap=%none` 添加到 `-fast` 之后，以保留 FORTRAN 77 对运算异常的捕获行为，因为 `-fast` 将捕获模式设置为 "common"。

```
f77 -fast -ftrap=%none
```

如果调用 `f77` 脚本，将发出消息，警示您正在 `-f77` 兼容模式下使用 `f95` 编译器。您可以在命令行中添加 `-errtags=INVOKE` 来禁用此消息。有关更多信息，请参见第 3.4.22 节 “–f77[=*list*]” [56]。

♦ ♦ ♦ 附 录 A

运行时错误消息

本附录介绍由 Fortran 运行时 I/O 库和操作系统生成的错误消息。

A.1 操作系统错误消息

操作系统错误消息包括系统调用失败、C 库错误和 shell 诊断。系统调用错误消息可在 `intro`(2) 中找到。通过 `Fortran` 库进行的系统调用并不直接生成错误消息。`Fortran` 库中的以下系统例程将调用 C 库例程，而这些库例程可生成错误消息：

```
integer system, status
status = system("cp  afile bfile")
print*, "status = ", status
end
```

显示以下消息：

```
cp: cannot access afile
 status =  512
```

A.2 `f95` 运行时 I/O 错误消息

`f95` I/O 库在运行时检测到错误的时候发出诊断消息。下面是一个使用 `f95` 编译和运行的示例：

```
demo% cat wf.f
      WRITE( 6 ) 1
      END
demo% f95 -o wf wf.f
demo% wf

 ******  FORTRAN RUN-TIME SYSTEM  ******
 Error 1003:  unformatted I/O on formatted unit
 Location:  the WRITE statement at line 1 of "wf.f"
 Unit:  6
 File:  standard output
Abort
```

因为 f95 消息包含对原始源代码文件名和行号的引用，所以应用程序开发者应该考虑在 I/O 语句中使用 ERR= 子句以软件方式捕获运行时 I/O 错误。

表 A-1 “f95 运行时 I/O 消息” 列出了由 f95 发出的运行时 I/O 消息。

表 A-1 f95 运行时 I/O 消息

错误	消息
1000	format error（格式错误）
1001	illegal unit number（非法的单元编号）
1002	formatted I/O on unformatted unit（未格式化单元上的格式化 I/O）
1003	unformatted I/O on formatted unit（格式化单元上的未格式化 I/O）
1004	direct-access I/O on sequential-access unit（顺序访问单元上的直接存取 I/O）
1005	sequential-access I/O on direct-access unit（直接访问单元上的顺序存取 I/O）
1006	device does not support BACKSPACE（设备不支持 BACKSPACE）
1007	off beginning of record（超出记录的开头）
1008	can’t stat file（无法对文件执行 stat 操作）
1009	no * after repeat count（重复计数后没有 *）
1010	record too long（记录太长）
1011	truncation failed（截断失败）
1012	incomprehensible list input（无法理解的列表输入）
1013	out of free space（可用空间不足）
1014	unit not connected（单元未连接）
1015	read unexpected character（读取意外的字符）
1016	illegal logical input field（非法逻辑输入字段）
1017	’new’ file exists（“新”文件已存在）
1018	can’t find ’old’ file（无法找到“旧”文件）
1019	unknown system error（未知的系统错误）
1020	requires seek ability（需要查找功能）
1021	illegal argument（非法参数）
1022	negative repeat count（重复计数为负）
1023	illegal operation for channel or device（通道或设备的操作非法）
1024	reentrant I/O（重进入 I/O）
1025	incompatible specifiers in open（打开的说明符不兼容）
1026	illegal input for namelist（名称列表的输入非法）
1027	error in FILEOPT parameter（FILEOPT 参数中有错误）
1028	writing not allowed（不允许写入）
1029	reading not allowed（不允许读取）
1030	integer overflow on input（输入上的整数溢出）

错误	消息
1031	floating-point overflow on input（输入上的浮点溢出）
1032	floating-point underflow on input（输入上的浮点下溢）
1051	default input unit closed（缺省输入单元已关闭）
1052	default output unit closed（缺省输出单元已关闭）
1053	direct-access READ from unconnected unit（从未连接单元直接进行 READ 访问）
1054	direct-access WRITE to unconnected unit（对未连接单元直接进行 WRITE 访问）
1055	unassociated internal unit（非关联的内部单元）
1056	null reference to internal unit（对内部单元的空引用）
1057	empty internal file（空内部文件）
1058	list-directed I/O on unformatted unit（未格式化单元上的列表引导 I/O）
1059	namelist I/O on unformatted unit（未格式化单元上的名称列表 I/O）
1060	tried to write past end of internal file（写操作已超过内部文件的结尾）
1061	unassociated ADVANCE specifier（非关联 ADVANCE 说明符）
1062	ADVANCE specifier is not 'YES' or 'NO'（ADVANCE 说明符不是 'YES' 或 'NO'）
1063	EOR specifier present for advancing input（出现针对高级输入的 EOR 说明符）
1064	SIZE specifier present for advancing input（出现针对高级输入的 SIZE 说明符）
1065	negative or zero record number（记录数为负或零）
1066	record not in file（记录不在文件中）
1067	corrupted format（格式已破坏）
1068	unassociated input variable（非关联的输入变量）
1069	more I/O-list items than data edit descriptors（I/O 列表项比数据编辑描述符多）
1070	zero stride in subscript triplet（下标三重态中的零跨距）
1071	zero step in implied DO-loop（隐含 DO 循环中的零步骤）
1072	negative field width（字段宽度为负）
1073	zero-width field（字段宽为零）
1074	character string edit descriptor reached on input（到达输入上的字符串编辑描述符）
1075	Hollerith edit descriptor reached on input（到达输入上的霍尔瑞斯编辑描述符）
1076	no digits found in digit string（数字字符串中没有数字）
1077	no digits found in exponent（指数中没有数字）
1078	scale factor out of range（比例因子超出范围）
1079	digit equals or exceeds radix（数字等于或超过基数）
1080	unexpected character in integer field（整数字段中的意外字符）
1081	unexpected character in real field（实数字段中的意外字符）
1082	unexpected character in logical field（逻辑字段中的意外字符）
1083	unexpected character in integer value（整数值中的意外字符）
1084	unexpected character in real value（实数值中的意外字符）
1085	unexpected character in complex value（复数值中的意外字符）

错误	消息
1086	unexpected character in logical value（逻辑值中的意外字符）
1087	unexpected character in character value（字符值中的意外字符）
1088	unexpected character before NAMELIST group name（NAMELIST 组名前的意外字符）
1089	NAMELIST group name does not match the name in the program（NAMELIST 组名与程序中的名称不匹配）
1090	unexpected character in NAMELIST item（NAMELIST 项中的意外字符）
1091	unmatched parenthesis in NAMELIST item name（NAMELIST 项名中的括号不匹配）
1092	variable not in NAMELIST group（变量不在 NAMELIST 组中）
1093	too many subscripts in NAMELIST object name（NAMELIST 对象名称中的下标太多）
1094	not enough subscripts in NAMELIST object name（NAMELIST 对象名称中的下标不足）
1095	zero stride in NAMELIST object name（NAMELIST 对象名称中的零跨距）
1096	empty section subscript in NAMELIST object name（NAMELIST 对象名称中的空节下标）
1097	subscript out of bounds in NAMELIST object name（NAMELIST 对象名称中的下标超出界限）
1098	empty substring in NAMELIST object name（NAMELIST 对象名称中的空子字符串）
1099	substring out of range in NAMELIST object name（NAMELIST 对象名称中的子字符串超出范围）
1100	unexpected component name in NAMELIST object name（NAMELIST 对象名称中的意外组件名称）
1111	unassociated ACCESS specifier（非关联 ACCESS 说明符）
1112	unassociated ACTION specifier（非关联 ACTION 说明符）
1113	unassociated BINARY specifier（非关联 BINARY 说明符）
1114	unassociated BLANK specifier（非关联 BLANK 说明符）
1115	unassociated DELIM specifier（非关联 DELIM 说明符）
1116	unassociated DIRECT specifier（非关联 DIRECT 说明符）
1117	unassociated FILE specifier（非关联 FILE 说明符）
1118	unassociated FMT specifier（非关联 FMT 说明符）
1119	unassociated FORM specifier（非关联 FORM 说明符）
1120	unassociated FORMATTED specifier（非关联 FORMATTED 说明符）
1121	unassociated NAME specifier（非关联 NAME 说明符）
1122	unassociated PAD specifier（非关联 PAD 说明符）
1123	unassociated POSITION specifier（非关联 POSITION 说明符）
1124	unassociated READ specifier（非关联 READ 说明符）
1125	unassociated READWRITE specifier（非关联 READWRITE 说明符）
1126	unassociated SEQUENTIAL specifier（非关联 SEQUENTIAL 说明符）
1127	unassociated STATUS specifier（非关联 STATUS 说明符）
1128	unassociated UNFORMATTED specifier（非关联 UNFORMATTED 说明符）
1129	unassociated WRITE specifier（非关联 WRITE 说明符）

错误	消息
1130	zero length file name（零长度文件名）
1131	ACCESS specifier is not 'SEQUENTIAL' or 'DIRECT'（ACCESS 说明符不是 'SEQUENTIAL' 或 'DIRECT'）
1132	ACTION specifier is not 'READ', 'WRITE' or 'READWRITE'（ACTION 说明符不是 'READ'、'WRITE' 或 'READWRITE'）
1133	BLANK specifier is not 'ZERO' or 'NULL'（BLANK 说明符不是 'ZERO' 或 'NULL'）
1134	DELIM specifier is not 'APOSTROPHE', 'QUOTE', or 'NONE'（DELIM 说明符不是 'APOSTROPHE'、'QUOTE' 或 'NONE'）
1135	unexpected FORM specifier（意外的 FORM 说明符）
1136	PAD specifier is not 'YES' or 'NO'（PAD 说明符不是 'YES' 或 'NO'）
1137	POSITION specifier is not 'APPEND', 'ASIS', or 'REWIND'（POSITION 说明符不是 'APPEND'、'ASIS' 或 'REWIND'）
1138	RECL specifier is zero or negative（RECL 说明符为零或负数）
1139	no record length specified for direct-access file（没有为直接访问文件指定记录长度）
1140	unexpected STATUS specifier（意外的 STATUS 说明符）
1141	status is specified and not 'OLD' for connected unit（指定了状态，但不是针对单元的 'OLD'）
1142	STATUS specifier is not 'KEEP' or 'DELETE'（STATUS 说明符不是 'KEEP' 或 'DELETE'）
1143	status 'KEEP' specified for a scratch file（为临时文件指定了状态 'KEEP'）
1144	impossible status value（不可能的状态值）
1145	a file name has been specified for a scratch file（已经为临时文件指定了文件名）
1146	attempting to open a unit that is being read from or written to（正在尝试打开在下列内容中读取或写入的单元）
1147	attempting to close a unit that is being read from or written to（正在尝试关闭在下列内容中读取或写入的单元）
1148	attempting to open a directory（正在尝试打开目录）
1149	status is 'OLD' and the file is a dangling symbolic link（状态是 'OLD'，文件是不定符号链接）
1150	status is 'NEW' and the file is a symbolic link（状态是 'NEW'，文件是符号链接）
1151	no free scratch file names（没有可用的临时文件名）
1152	specifier ACCESS='STREAM' for default unit（对于缺省单元，说明符 ACCESS= 'STREAM'）
1153	stream-access to default unit（对缺省单元的流访问）
1161	device does not support REWIND（设备不支持 REWIND）
1162	read permission required for BACKSPACE（BACKSPACE 需要读权限）
1163	BACKSPACE on direct-access unit（直接访问单元上的 BACKSPACE）
1164	BACKSPACE on binary unit（二进制单元上的 BACKSPACE）
1165	end-of-file seen while backspacing（在 backspace 操作期间看到文件结尾）
1166	write permission required for ENDFILE（ENDFILE 需要写权限）
1167	ENDFILE on direct-access unit（直接访问单元上的 ENDFILE）

错误	消息
1168	stream-access to sequential or direct-access unit（对连续或直接访问单元的流访问）
1169	stream-access to unconnected unit（对未连接单元的流访问）
1170	direct-access to stream-access unit（对流访问单元的直接访问）
1171	incorrect value of POS specifier（POS 说明符的值错误）
1172	unassociated ASYNCHRONOUS specifier（非关联 ASYNCHRONOUS 说明符）
1173	unassociated DECIMAL specifier（非关联 DECIMAL 说明符）
1174	unassociated IOMSG specifier（非关联 IOMSG 说明符）
1175	unassociated ROUND specifier（非关联 ROUND 说明符）
1176	unassociated STREAM specifier（非关联 STREAM 说明符）
1177	ASYNCHRONOUS specifier is not 'YES' or 'NO'（ASYNCHRONOUS 说明符不是 'YES' 或 'NO'）
1178	ROUND specifier is not 'UP', 'DOWN', 'ZERO', 'NEAREST', 'COMPATIBLE' or 'PROCESSOR-DEFINED'（ROUND 说明符不是 'UP'、'DOWN'、'ZERO'、'NEAREST'、'COMPATIBLE' 或 'PROCESSOR-DEFINED'）
1179	DECIMAL specifier is not 'POINT' or 'COMMA'（DECIMAL 说明符不是 'POINT' 或 'COMMA'）
1180	RECL specifier is not allowed in OPEN statement for stream-access unit（在流访问单元的 OPEN 语句中不允许有 RECL 说明符）
1181	attempting to allocate an allocated array（正在尝试分配已分配的数组）
1182	deallocating an unassociated pointer（正在取消分配非关联的指针）
1183	deallocating an unallocated allocatable array（正在取消分配未分配的可分配数组）
1184	deallocating an allocatable array through a pointer（正取消分配可通过指针分配的数组）
1185	deallocating an object not allocated by an ALLOCATE statement（正在取消分配不是由 ALLOCATE 语句分配的对象）
1186	deallocating a part of an object（正在取消分配对象的一部分）
1187	deallocating a larger object than was allocated（正在取消比已分配的对象大的对象）
1191	unallocated array passed to array intrinsic function（向数组固有函数传递了未分配的数组）
1192	illegal rank（非法的等级）
1193	small source size（源尺寸小）
1194	zero array size（数组尺寸为零）
1195	negative elements in shape（形状中的元素为负）
1196	illegal kind（非法的种类）
1197	nonconformable array（不一致的数组）
1213	asynchronous I/O on unconnected unit（未连接单元上的异步 I/O）
1214	asynchronous I/O on synchronous unit（同步单元上的异步 I/O）
1215	a data edit descriptor and I/O list item type are incompatible（数据编辑描述符与 I/O 列表项类型不兼容）

错误	消息
1216	current I/O list item doesn't match with any data edit descriptor（当前 I/O 列表项与所有数据编辑描述符都不匹配）
1217	illegal CORR_ACCTION value（非法的 CORR_ACCTION 值）
1218	infinite loop occurred due to I/O handler enabled（由于启用 I/O 处理程序而发生死循环）
1220	the number of requested bytes is greater than is supported on the target platform（所请求的字节数大于目标平台上所支持的字节数）
1221	data in a UNION cannot be read from or written to an incompatible file type（UNION 中的数据无法从不兼容的文件类型读取，也不能写入不兼容的文件类型）
2001	invalid constant, structure, or component name（无效的常数、结构或组件名称）
2002	handle not created（未创建句柄）
2003	character argument too short（字符变量太短）
2004	array argument too long or too short（数组变量太长或太短）
2005	end of file, record, or directory stream（文件、记录或目录流的结尾）
2021	lock not initialized (OpenMP)（未将锁定初始化 (OpenMP)）
2022	deadlock in using lock variable (OpenMP)（正在使用锁定变量的死锁 (OpenMP)）
2023	lock not set (OpenMP)（未设置锁定 (OpenMP)）

♦ ♦ ♦ 附 录 B

功能发行版历史记录

本附录列出了 Fortran 编译器的此发行版和最近的发行版中的新增和更改的功能。

B.1 Oracle Solaris Studio 12.4 Fortran 发行版

Oracle Solaris Studio Fortran 95 编译器 8.7 版是 Oracle Solaris Studio 12.4 发行版中的一个组件。

- 用于 x86 上的 Intel Ivy Bridge 处理器的新的 `-xarch`、`-xchip` 和 `-xtarget` 值。
- 用于 SPARC T5、M5、M6 和 M10+ 处理器的新的 `-xarch`、`-xchip` 和 `-xtarget` 值。
- 支持 Ivy Bridge 汇编程序指令。
- 支持 Ivy Bridge 内部函数，可在 *solstudio-install-dir*`/lib/compilers/include/cc/immintrin.h` 中找到这些函数。
- `-xarch=generic` 的缺省值针对 x86 上的 `-m32` 设置为 `sse2`。
- 支持 x86 上的 `-xlinkopt`。对适用于现代 Intel 处理器的大型企业应用程序进行的模块间、过程间代码排序优化。大型应用程序中的完全优化的二进制代码使性能提升了 5%。
- 增强的 `-xs` 选项，用于在可执行文件大小与为了调试而保留对象文件的需求之间进行权衡。
- 支持 Linux 上的 `-xanalyze` 和 `-xannotate`。
- 支持 `-fopenmp` 与 `-xopenmp=parallel` 等效。
- 对使用模块的应用程序进行编译的时间有明显改进，消除了由于模块处理而导致的内存溢出。
- `#pragma ident` 可用于源文件中，以识别编译对象的源版本。
- 支持延迟类型参数（冒号）作为在声明中使用的字符类型中的 LEN 类型参数。例如：

  ```
  character(LEN=:), pointer :: str
  ```

- 支持过程指针。
- 支持 Fortran 2003 标准中 `ISO_C_BINDING` 模块的 `C_F_PROCPOINTER` 函数。`C_FUNLOC` 函数扩展为允许过程指针作为参数。

- 支持 Fortran 2003 功能的 ABSTRACT 接口。
- 完全支持面向对象的 Fortran。现在允许使用具有 GENERIC、DEFERRED、NON_OVERRIDABLE、PASS 和 NOPASS 属性的类型绑定过程。
- 支持 Fortran 2003 功能，使派生类型和通用函数有相同的名称。
- 支持 Fortran 2008 功能，允许将 TARGET 对象传递到 INTENT(IN) 指针哑元。
- 支持 Fortran 2003 功能，允许在派生类型中完成例程。
- 新编译器选项：
  - -fma，启用自动生成浮点混合乘加指令。
  - -fserialio，指定程序不能一次在多个线程中执行 I/O。
  - (x86) -preserve_argvalues，在堆栈中保存基于寄存器的函数参数的副本。
  - -xdebuginfo，控制发出多少调试和监测信息。
  - -xglobalize，控制文件静态变量的全局化，但是不控制函数的全局化。
  - -xinline_param，可用于更改编译器用来确定何时内联函数调用的试探式方法。
  - -xinline_report，在编译器内联函数时生成报告并写入标准输出。
  - -xipo_build，通过避免在初始传递期间通过编译器进行优化（仅在链接时优化）而缩短编译时间。
  - -xkeep_unref，保留未引用函数和变量的定义。
  - -keepmod，保留编译时未发生更改的模块。缺省值是 -xkeepmod=yes，它代替了每次创建新模块文件时的旧行为，即使与上次编译相比没有任何变化。
  - -xM，自动生成 makefile 依赖性。与新的 -keepmod=yes 选项结合使用，允许使用模块在 Fortran 应用程序上实现最佳增量生成。
  - -xpatchpadding，在各函数启动之前保留内存区域。
  - (Oracle Solaris) -xsegment_align，使驱动程序在链接行上包括特殊映射文件。
  - -xthroughput，指示当多个进程同时在系统上运行时运行应用程序。
  - -xunboundsym，指定程序是否包含对动态绑定符号的引用。
- 自从 2005 年发行 Sun Studio 10 以来，Studio Fortran 编译器就没有使用 SPARC 平台上的库 libfmaxlai、libfmaxvai、libfminlai、libfminvai、libfprodai 和 libfsumai。

  未来的发行版中将删除这些库。到那时候，将不再使用由 Sun Studio 10 发行版之前的 Studio 编译器生成的对象文件和可执行文件，并且必须在更新的 Studio 编译器中对这些文件重新进行编译。如果您有需要使用任何这些库的旧对象文件和可执行文件，但重新编译又不可行，则应保留旧编译器安装，或将所需的特定库从旧编译器安装复制到新编译器安装。

B.2 Oracle Solaris Studio 12.3 Fortran 发行版

Oracle Solaris Studio Fortran 95 编译器 8.6 版是 Oracle Solaris Studio 12.3 发行版中的一个组件。

- Fortran 运行时库现在支持大于 2GB 的连续访问无格式记录。
- 支持新 SPARC T4 平台：`–xtarget=T4`, `–xchip=T4`, `–xarch=sparc4`
- 支持新的 x86 平台 Sandy Bridge / AVX：`–xtarget=sandybridge –xchip=sandybridge –xarch=avx`
- 支持新的 x86 平台 Westmere / AES：`–xtarget=westmere –xchip=westmere –xarch=aes`
- 新编译器选项：`–Xlinker` *arg* 将参数传递给链接程序 `ld`(1)。与 `–Wl,`*arg* 等效。（第 3.4.104 节 “`-Xlinker` *arg*” [89]）
- OpenMP 缺省线程数 `OMP_NUM_THREADS` 现在为 2（过去为 1）。（第 3.4.157 节 “`–xopenmp[={parallel|noopt|none}]`” [126]）
- 支持 OpenMP 3.1 共享内存并行化规范。（第 3.4.157 节 “`–xopenmp[={parallel|noopt|none}]`” [126]）
- 使用 `–library=sunperf` 可链接到 Sun 性能库。这将废弃 `-xlic_lib=sunperf`。(第 3.4.55 节 “`-library=sunperf`” [71])
- 内部函数例程 `LEADZ`、`POPCNT` 和 `POPPAR` 以前的返回类型与参数类型相同。在此发行版中，为符合 Fortran 2008 标准，这些内部函数将返回一个缺省整数，无论参数为何种类型。这引入了与以前发行版的轻微不兼容性。
- 现在支持与多态性相关的面向对象的 Fortran 功能：
  - 支持的 OOF 功能：类型扩展和多态实体：`CLASS` 语句、无限制多态性、`SELECT TYPE` 构造、`ABSTRACT` 派生类型、`EXTENDS_TYPE_OF` 和 `SAME_TYPE_AS` 内部函数，以及到无限制指针的序列类型分配。
  - 不支持的 OOF 功能：`typebound` 过程：`typebound PROCEDURE` 声明、`GENERIC`、`DEFERRED`、`NON_OVERRIDABLE`、`PASS`、`NOPASS`。
- 其他 F2003/2008 新增功能：
  - 增强的结构构造函数：使用组件名称构造结构常量。
  - 模块派生类型和组件上的增强 `PUBLIC`/`PRIVATE` 访问控制。
  - 更多 Fortran 2008 数学内部函数支持。在 x86 平台上，除了 `ERFC_SCALED`、`NORM2` 以及某些 `REAL*16` 变量外，现在多数 Fortran 2008 数学内部函数都受支持。
  - 不带组件的派生类型。
  - `KIND` 参数已添加到 `ICHAR`、`IACHAR`、`ACHAR`、`SHAPE`、`UBOUND`、`LBOUND`、`SIZE`、`MINLOC`、`MAXLOC`、`COUNT`、`LEN`、`LEN_TRIM`、`INDEX`、`SCAN` 和 `VERIFY` 内部函数中。
  - `BACK` 参数已添加到 `MINLOC` 和 `MAXLOC` 内部函数中。
  - 添加了新的内部函数 `FINDLOC` 和 `STORAGE_SIZE`。
  - 新的关键字 `ERRMSG`、`SOURCE` 和 `MOLD` 已添加到 `ALLOCATE` 语句中，`ERRMSG` 已添加到 `DEALLOCATE` 语句中。

B.3 Oracle Solaris Studio 12.2 Fortran 发行版

Oracle Solaris Studio Fortran 95 编译器 8.5 版是 Oracle Solaris Studio 12.2 发行版中的一个组件。

- 支持 SPARC VIS3 版本的 SPARC-V9 指令集。如果使用 `-xarch=sparcvis3` 选项进行编译，编译器可以使用 SPARC-V9 指令集、UltraSPARC 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 1.0）、UltraSPARC-III 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 2.0）、混合乘加指令和可视指令集 (Visual Instruction Set, VIS) 版本 3.0）中的指令。
- 在基于 x86 的系统上，`-xvector` 选项的缺省值已更改为 `-xvector=simd`。在基于 x86 的系统上，缺省情况下以可产生有利结果的优化级别 3 和更高级别使用流扩展。可以使用子选项 `no%simd` 将其禁用。在基于 SPARC 的系统上，缺省值为 `-xvector=%none`。请参见第 3.4.186 节 “-xvector[=*a*]” [143]。
- 现在提供了对 AMD SSE4a 指令集的支持。使用 `-xarch=amdsse4a` 选项进行编译。
- 新增的 `-traceback` 选项使可执行文件在出现严重错误时显示堆栈跟踪。使用此选项时，可执行文件将捕获一组信号，并显示堆栈跟踪和信息转储，然后退出。如果多个线程都生成一个信号，则只为第一个线程生成堆栈跟踪。要使用回溯，请在使用 `f95`、`cc` 或 `CC` 链接程序时添加 `-traceback` 选项。为了方便起见，也可在编译时接受此选项，但是会将其忽略。将 `-traceback` 选项与 `-G` 选项结合使用来创建共享库的做法是错误的。请参见第 3.4.92 节 “–traceback[={%none|common|*signals_list*}]” [85]。
- `-mt` 选项已更改为 `-mt=yes` 或 `-mt=no`。`-mt=yes` 选项确保按正确的顺序链接库。请参见第 3.4.60 节 “-mt[={yes|no}]” [73]
- `-xprofile=tcov` 选项经过增强，支持可选的分析目录路径名，还能够生成与 `tcov` 兼容的反馈数据。请参见第 3.4.169 节 “–xprofile=*p*” [132]。
- 新增的 `-xkeepframe[=[%all,%none ]` 选项禁止对指定的函数进行与堆栈相关的优化。`%all` 禁止对所有代码进行与堆栈相关的优化。`%none` 允许对所有代码进行与堆栈相关的优化。缺省值为 `-xkeepframe=%none`。请参见第 3.4.139 节 “–xkeepframe[=[%all,%none,*name*,no%*name*]]” [118]。
- 已经实现了其他 F2003 功能。请参见第 4.6 节 “Fortran 200x 功能” [159]。
- `IVDEP` 指令指示编译器忽略在循环中找到的部分或全部对数组引用的循环附带依赖性，以进行优化。这样，编译器将可以执行通过其他方式不能实现的各种循环优化。`-xivdep` 选项可用于禁用 `IVDEP` 指令或确定应如何解释指令。请参见第 2.3.3 节 “IVDEP 指令” [31]。

B.4 Sun Studio 12 Update 1 Fortran 发行版

- 对于编译器在 x86 平台上的 Solaris OS 或者 Linux OS 上创建的对象文件，如果应用程序代码包含参数或返回值为 _m128/_m64 数据类型的函数，则这些对象文件与先前版本的编译器不兼容。使用 .il 内联函数文件、汇编程序代码或调用这些函数的 asm 内联语句的用户，也需要了解这些不兼容的情况。

- 新增 x86 –xtarget 值 woodcrest、penryn、nehalem。
- 新增 SPARC –xtarget 值 ultraT2plus 和 sparc64vii。
- 新增 x86 –xarch 和 –xchip 值 ssse3、sse4_1、sse4_2、core2、penryn、nehalem、barcelona。
- 新增 SPARC –xarch 和 –xchip 值 sparcima、sparc64vii 和 ultraT2plus。
- -xprofile=collect 和 -xprofile=use 选项在多线程动态链接应用程序的分析方面提供了改进的支持功能。
- 在 Solaris 平台上，-xpec[=yes|no] 选项会生成可重新编译用于自动调优系统 (Automatic Tuning System, ATS) 的 PEC 二进制文件。
- 现在，针对 -xO3 或更高优化级别隐式启用了 -xdepend 选项，该选项不再包括在 -fast 选项的扩展中。
- 支持 OpenMP 3.0 任务处理。
- -xannotate[=yes|no]（仅限 SPARC 平台）指示编译器创建稍后可用 binopt(1) 等二进制修改工具转换的二进制文件。
- 四精度 (REAL*16) 已在 x86 平台上实现。REAL*16 是 128 位 IEEE 浮点。
- 编译器通常会在 /tmp 目录中创建临时文件。可以通过设置 TMPDIR 环境变量来指定其他目录。
- cpu_time() Fortran 内例程的行为在 Solaris 和 Linux 平台之间有所不同。
- Fortran 2003 IMPORT 语句现已实现。

B.5 Sun Studio 12 Fortran 发行版

- 目前，Fortran 编译器适用于以下 Linux（x86 和 x64）分发包：SuSe Linux Enterprise Server 9 with Service Pack 3（或更新版本）、Red Hat Enterprise Linux 4 以及其他基于 2.6 内核的 Linux 分发包（尽管不会对这些分发包提供正式支持）。
- 使用 -m64 创建 64 位可执行文件和共享库。
- 使用 -xarch 新标志替换已过时的标志。
- -xtarget 和 -xchip 的新值为 UltraSPARC T2 和 SPARC64VI 处理器提供代码生成功能。
- -fma=fused 新标志允许在支持混合乘加指令的处理器上生成这些指令。
- -xhwcprof 新标志为数据空间分析启用编译器支持。
- -xinstrument 新标志使线程分析器支持性能分析。
- -xregs=frameptr 已添加至 x86 的 -fast。
- 提供 -xarch=sse2 和 -xia 选项支持在 Solaris x86 平台上执行区间运算。
- x86 平台和 SPARC 平台均接受显式预取指令。(–xprefetch=explicit)
- 调试信息的缺省格式由 "stabs" 标准格式更改为 "dwarf" 标准格式。(-xdebugformat=dwarf)。

B.6 Sun Studio 11 Fortran 发行版

- **新的 `-xmodel` 选项：**使用新的 `-xmodel` 选项，可以在 64 位 AMD 体系结构上指定内核、小型或中型内存模型。如果全局变量和静态变量的大小超过 2 GB，请指定 `-xmodel=medium`。否则，请使用缺省设置 `-xmodel=small`。请参见第 3.4.152 节 “–xmodel=[small | kernel | medium]” [125]。
- **为 x86 SSE2 平台扩展的 `-xvector` 选项：**使用 `-xvector` 选项，可以自动生成向量库函数调用和/或生成 SIMD（Single Instruction Multiple Data，单指令多数据）指令。现在，此选项在 x86 SSE2 平台上提供扩展的语法。
- **`STACKSIZE` 环境变量已增强：**`STACKSIZE` 环境变量的语法已加以改进，可以包括单位关键字。
- **`-xpagesize` 选项在 x86 平台上可用：**现在，在 x86 平台以及 SPARC 上，启用了 `-xpagesize`、`-xpagesize_heap` 和 `-xpagesize_stack` 选项。请参见第 3.4.159 节 “–xpagesize=*size*” [127]。
- **启用了新的 UltraSPARC T1 和 UltraSPARC IV+ 目标：**`-xarch`、`-xchip`、`-xcache` 和 `-xtarget` 的值支持新的 UltraSPARC 处理器。请参见第 3.4.179 节 “–xtarget=*t*” [139]。

♦ ♦ ♦ 附 录 C

Fortran 指令摘要

本附录汇总了 f95 Fortran 编译器可识别的指令：

- 通用 Fortran 指令
- Sun 并行化指令
- Cray 并行化指令
- OpenMP Fortran 95 指令、库例程和环境

C.1 通用 Fortran 指令

第 2.3 节 “指令” [24]部分对 f95 接受的通用指令进行了说明。

表 C-1 通用 Fortran 指令摘要

格式	
!$PRAGMA *keyword* (*a* [, *a*] …) *[* , *keyword (a [* , *a*] …)] ,…	
!$PRAGMA SUN *keyword* (*a* [, *a*] …) *[* , *keyword (a [* , *a*] …)] ,…	
第 1 列中的注释指示符可以是 c、C、! 或 *。（在这些示例中，我们使用了 !。f95 自由格式必须使用 !。）	
C 指令	!$PRAGMA C(*list*) 将一系列外部函数的名称声明为 C 语言例程。
IGNORE_TKR 指令	!$PRAGMA IGNORE_TKR {*name* {, *name*} ...} 在解析特定调用时，编译器会忽略在通用过程接口中出现的指定哑元参数名称的类型、种类和等级。
UNROLL 指令	!$PRAGMA SUN UNROLL=*n* 建议编译器将下面的循环解开为指定的长度 *n*。
WEAK 指令	!$PRAGMA WEAK(*name*[=*name2*]) 将 *name* 声明为弱符号，或者声明为 *name2* 的别名。
OPT 指令	!$PRAGMA SUN OPT=*n* 将子程序的优化级别设置为 *n*。
PIPELOOP 指令	!$PRAGMA SUN PIPELOOP[=*n*]

	断言循环中在间隔为 *n* 的迭代之间存在依赖性。
PREFETCH 指令	!$PRAGMA SUN_PREFETCH_READ_ONCE (*name*) !$PRAGMA SUN_PREFETCH_READ_MANY (*name*) !$PRAGMA SUN_PREFETCH_WRITE_ONCE (*name*) !$PRAGMA SUN_PREFETCH_WRITE_MANY (*name*) 请求编译器为名称引用生成预取指令。（需要使用 -xprefetch 选项，缺省情况下启用该选项。编译时使用 –xprefetch=no 可以禁用预取指令。目标体系结构也必须支持预取指令，而且编译器优化级别必须设置为大于 –xO2。）
ASSUME 指令	!$PRAGMA [BEGIN] ASSUME (*expression* [,*probability*]) !$PRAGMA END ASSUME 断言编译器可假定程序中某些点处的条件为真。

C.2 特殊的 Fortran 指令

下列指令仅用于 f95。有关详细信息，请参见第 4.8.2 节 “FIXED 和 FREE 指令” [167]。

表 C-2 特殊的 Fortran 指令

格式	!DIR$ 指令： 初始行 !DIR$& ...：续行 对于固定格式源代码，也可以接受 C 作为指令指示符： CDIR$ 指令... 该行必须在第 1 列开始。对于自由格式的源文件，该行之前可以有空格。
FIXED/FREE 指令	!DIR$ FREE!DIR$ FIXED 这些指令指定指令后面的行的源代码格式。它们适用于所在源代码文件的其余部分，或者在遇到下一个 FREE 或 FIXED 指令之前的部分。
IVDEP	!DIR$ IVDEP 断言以下 DO、FORALL 或 WHERE 循环不含循环附带的依赖性，并且可以进行优化。-xivdep 选项确定的解释。请参见第 2.3.3 节 “IVDEP 指令” [31]。

C.3 Fortran OpenMP 指令

Oracle Solaris Studio Fortran 编译器支持 OpenMP 3.1 Fortran API。-openmp 编译器标志允许使用这些指令。（请参见第 3.4.157 节 “–xopenmp[={parallel|noopt|none}]” [126]）。

有关完整的详细信息，请参见《*OpenMP API* 用户指南》。

索引

D

E

F