Fortran 用户指南

Sun™ Studio 11

Sun Microsystems, Inc.
www.sun.com

文件号码 819-4761-10
2005 年 11 月，修订版 A

请将有关本文档的意见和建议提交至：http://www.sun.com/hwdocs/feedback

目录

表

阅读本书之前

《Fortran 用户指南》介绍了 Sun™ Studio Fortran 库中的内部函数和例程。该参考手册适用于具有 Fortran 语言和 Solaris™ 操作环境使用经验的编程人员。

该指南适用于具有 Fortran 语言使用经验以及想了解如何有效地使用 Sun Fortran 编译器的科研人员、工程师和编程人员。同时，假定这些人员熟悉 Solaris 操作环境或 UNIX®。

配套提供的 《Fortran 编程指南》中介绍了 Solaris 操作环境中的 Fortran 编程问题，其中包括输入 / 输出、应用程序开发、库的创建和使用、程序分析、移植、优化和并行化。

印刷约定

表 P-1 字体约定

字体[1]	含义	示例
`AaBbCc123`	命令、文件和目录的名称；计算机屏幕输出。	编辑 `.login` 文件。 使用 `ls -a` 列出所有文件。 `% You have mail.`
`AaBbCc123`	用户键入的内容，与计算机屏幕输出的显示不同。	`% su` `Password:`
AaBbCc123	保留未译的新词或术语以及要强调的词。要使用实名或值替换的命令行变量。	这些称为 *class* 选项。 要删除文件，请键入 **`rm`** *filename*。
新词术语强调	新词或术语以及要强调的词。	您**必须**成为超级用户才能执行此操作。
《书名》	书名	阅读 《用户指南》的第 6 章。

1 浏览器的设置可能会与这些设置不同。

- 符号 Δ 表示有效的空格：

```
ΔΔ36.001
```

- FORTRAN 77 标准使用较旧的约定，以大写字母的形式拼写名称 “FORTRAN”。当前的约定使用小写字母：“Fortran 95”
- 出现的联机手册页参考带有主题名称和章节号。例如，库例程 `GETENV` 的参考显示为 `getenv(3F)`，这意味着访问此手册页的 man 命令为：`man -s 3F getenv`

表 P-2 代码约定

代码符号	含义	表示法	代码示例
[ ]	方括号中包含可选参数。	`O[`*n*`]`	`-O4°¢-O`
{ }	花括号中包含所需选项的选项集合。	`d{y|n}`	`-dy`
|	分隔变量的 “|” 或 “-” 符号，只能选择其一。	`B{dynamic|static}`	`-Bstatic`
:	与逗号一样，分号有时可用于分隔参数。	`R`*dir*`[:`*dir*`]`	`-R/local/libs:/U/a`
…	省略号表示一系列的省略。	`-xinline=`*f1*`[,…`*fn*`]`	`-xinline=alpha,dos`

Shell 提示符

Shell	提示符
C shell	*machine-name*`%`
C shell 超级用户	*machine-name*`#`
Bourne shell、Korn shell 和 GNU Bourne-Again shell	`$`
Bourne shell、Korn shell 和 GNU Bourne-Again shell 的超级用户	`#`

支持的平台

此 Sun Studio 发行版本支持使用 SPARC® 和 x86 系列处理器体系结构（UltraSPARC®、SPARC64、 AMD64、 Pentium 和 Xeon EM64T）的系统。通过访问 `http://www.sun.com/bigadmin/hcl` 中的硬件兼容性列表，可以了解您在使用的 Solaris 操作系统版本的支持系统。这些文档列出了实现各个平台类型的所有差别。

在本文档中，这些与 x86 有关的术语具有以下含义：

- “x86” 是指较大的 64 位和 32 位 x86 兼容产品系列。
- “x64” 表示有关 AMD64 或 EM64T 系统的特定 64 位信息。
- “32 位 x86”表示有关基于 x86 的系统的特定 32 位信息。

有关所支持的系统，请参见硬件兼容性列表。

访问 Sun Studio 软件和手册页

Sun Studio 软件及其手册页未安装到 `/usr/bin/` 和 `/usr/share/man` 标准目录中。要访问软件，必须正确设置 `PATH` 环境变量（请参见第 xix 页的“访问软件”）。要访问手册页，必须正确设置 `MANPATH` 环境变量 （请参见第 xx 页的 “访问手册页”）。

有关 `PATH` 变量的详细信息，请参见 `csh`(1)、 `sh`(1)、 `ksh`(1) 和 `bash`(1) 手册页。有关 `MANPATH` 变量的详细信息，请参见 `man`(1) 手册页。有关设置 `PATH` 变量和 `MANPATH` 变量以访问此发行版本的详细信息，请参见安装指南或询问系统管理员。

注 – 本节中的信息假设 Sun Studio 软件安装在 Solaris 平台上的 `/opt` 目录中和 Linux 平台上的 `/opt/sun` 目录中。如果未将软件安装在默认目录中，请询问系统管理员以获取系统中的相应路径。

访问软件

使用以下步骤决定是否需要更改 `PATH` 变量以访问该软件。

决定是否需要设置 PATH 环境变量

1. **通过在命令提示符后键入以下内容以显示 PATH 变量的当前值。**

```
% echo $PATH
```

2. **在 Solaris 平台上，查看输出中是否包含有 /opt/SUNWspro/bin 的路径字符串。在 Linux 平台上，查看输出中是否包含有 /opt/sun/sunstudio11/bin 的路径字符串。**

   如果找到该路径，则说明已设置了访问该软件的 PATH 变量。如果没有找到该路径，则按照下一步的说明设置 PATH 环境变量。

设置 PATH 环境变量以实现对软件的访问

- **在 Solaris 平台上，将以下路径添加到 PATH 环境变量中。如果以前安装了 Forte Developer 软件、Sun ONE Studio 软件，或其他发行版本的 Sun Studio 软件，则将以下路径添加到这些软件安装路径之前。**

  /opt/SUNWspro/bin

- **在 Linux 平台上，将以下路径添加到 PATH 环境变量中。**

  /opt/sun/sunstudio11/bin

访问手册页

使用以下步骤决定是否需要更改 MANPATH 变量以访问手册页。

决定是否需要设置 MANPATH 环境变量

1. **通过在命令提示符后键入以下内容以请求 dbx 手册页。**

```
% man dbx
```

2. **请查看输出 （如果有）。**

   如果找不到 dbx(1) 手册页或者显示的手册页不是软件当前版本的手册页，请按照下一步中的说明来设置 MANPATH 环境变量。

设置 MANPATH 环境变量以实现对手册页的访问

- **在 Solaris 平台上，将以下路径添加到 MANPATH 环境变量中。**
  `/opt/SUNWspro/man`
- **在 Linux 平台上，将以下路径添加到 MANPATH 环境变量中。**
  `/opt/sun/sunstudio11/man`

访问集成开发环境

Sun Studio 集成开发环境 (integrated development environment, IDE) 提供了创建、编辑、生成、调试 C、 C++ 或 Fortran 应用程序并分析其性能的模块。

启动 IDE 的命令是 `sunstudio`。有关该命令的详细信息，请参见 `sunstudio`(1) 手册页。

IDE 是否可以正确操作取决于 IDE 能否找到核心平台。 `sunstudio` 命令会查找两个位置的核心平台：

- 该命令首先查找 Solaris 平台上的默认安装目录 `/opt/netbeans/3.5V11` 和 Linux 平台上的默认安装目录 `/opt/sun/netbeans/3.5V11`。
- 如果该命令在默认目录中找不到核心平台，则它会假设包含 IDE 的目录和包含核心平台的目录均安装在同一位置上。例如，在 Solaris 平台上，如果包含 IDE 的目录的路径是 `/foo/SUNWspro`，则该命令会在 `/foo/netbeans/3.5V11` 中查找核心平台。在 Linux 平台上，如果包含 IDE 的目录的路径是 `/foo/sunstudio11`，则该命令会在 `/foo/netbeans/3.5V11` 中查找核心平台。

如果核心平台未安装在 `sunstudio` 命令查找它的任一位置上，则客户端系统上的每个用户必须将环境变量 `SPRO_NETBEANS_HOME` 设置为安装核心平台的位置 (*/installation_directory*`/netbeans/3.5V11`)。

在 Solaris 平台上，IDE 的每个用户还必须将 */installation_directory*`/SUNWspro/bin` 添加到其他任何 Forte Developer 软件、 Sun ONE Studio 软件或 Sun Studio 软件发行版本路径前面的 `$PATH` 中。在 Linux 平台上， IDE 的每个用户还必须将 */installation_directory*`/sunstudio11/bin` 添加到其他任何发行版本的 Sun Studio 软件路径前面的 `$PATH` 中。

路径 */installation_directory*`/netbeans/3.5V11/bin` 不能添加到用户的 `$PATH` 中。

访问编译器和工具文档

您可以访问以下位置的文档：

- 可以通过随软件一起安装在本地系统或网络上的文档索引获取文档，位置为 Solaris 平台上的 `file:/opt/SUNWspro/docs/zh/index.html` 和 Linux 平台上的 `file:/opt/sun/sunstudio11/docs/zh/index.html`。

  如果未将软件安装在 Solaris 平台上的 `/opt` 目录中或 Linux 平台上的 `/opt/sun` 目录中，请询问系统管理员以获取系统中的相应路径。

- 大多数的手册都可以从 `docs.sun.com`sm Web 站点获取。以下书目只能从 Solaris 平台上安装的软件中找到：
  - 《标准 C++ 库类参考》
  - 《标准 C++ 库用户指南》
  - 《Tools.h++ 类库参考》
  - 《Tools.h++ 用户指南》
- 适用于 Solaris 平台和 Linux 平台的发行说明可以通过 `docs.sun.com` Web 站点获取。
- 在 IDE 中通过“帮助”菜单以及许多窗口和对话框中的“帮助”按钮，可以访问 IDE 所有组件的联机帮助。

您可以通过 Internet 访问 `docs.sun.com` Web 站点 (`http://docs.sun.com`) 以阅读、打印和购买 Sun Microsystems 的各种手册。如果找不到手册，请参见随软件一起安装在本地系统或网络中的文档索引。

注 – Sun 对本文档中提到的第三方 Web 站点的可用性不承担任何责任。对于此类站点或资源中的（或通过它们获得的）任何内容、广告、产品或其他资料，Sun 并不表示认可，也不承担任何责任。对于因使用或依靠此类站点或资源中的（或通过它们获得的）任何内容、产品或服务而造成的或连带产生的实际或名义损坏或损失，Sun 概不负责，也不承担任何责任。

使用易读格式的文档

该文档采用易读格式提供，以方便残障用户使用辅助技术进行阅读。您还可以按照下表所述，找到文档的易读版本。如果未将软件安装在 /opt 目录中，请询问系统管理员以获取系统中的相应路径。

文档类型	易读版本的格式和位置
手册 （第三方手册除外）	HTML，位于 http://docs.sun.com
第三方手册： • 《标准 C++ 库类参考》 • 《标准 C++ 库用户指南》 • 《Tools.h++ 类库参考》 • 《Tools.h++ 用户指南》	HTML，位于 Solaris 平台上所安装软件中的文档索引 file:/opt/SUNWspro/docs/zh/index.html
自述文件	HTML，位于 Sun Developer Network 门户网站 http://developers.sun.com/prodtech/cc/documentation/
手册页	HTML，位于安装的软件上的文档索引，位置为 Solaris 平台上的 file:/opt/SUNWspro/docs/zh/index.html 和 Linux 平台上的 file:/opt/sun/sunstudio11/docs/zh/index.html。
联机帮助	HTML，可通过 IDE 中的 “帮助”菜单和 “帮助”按钮访问
发行说明	HTML，位于 http://docs.sun.com

相关编译器和工具文档

下表描述的相关文档可以在 file:/opt/SUNWspro/docs/zh/index.html 和 http://docs.sun.com 上获取。如果未将软件安装在 /opt 目录中，请询问系统管理员以获取系统中的相应路径。

文档标题	描述
《Fortran 编程指南》	描述了如何在 Solaris 环境中编写有效的 Fortran 程序；输入/输出、库、性能、调试和并行化。
《Fortran 库参考》	详细说明了 Fortran 库和内部函数。
《OpenMP API 用户指南》	OpenMP 多处理 API 摘要，并提供有关实现的详细信息。
《数值计算指南》	描述关于浮点计算数值精确性的问题。

访问相关的 Solaris 文档

下表描述了可从 `docs.sun.com` Web 站点上获取的相关文档。

文档集合	文档标题	描述
Solaris 参考手册集合	请参见手册页部分的标题。	提供有关 Solaris 操作系统的信息。
Solaris 软件开发者集合	《链接程序和库指南》	介绍了 Solaris 链接编辑器和运行时链接程序的操作。
Solaris 软件开发者集合	《多线程编程指南》	涵盖 POSIX 和 Solaris 线程 API、使用同步对象进行程序设计、编译多线程程序和多线程程序的查找工具。

开发者资源

访问 Sun Developer Network Sun Studio 门户网站 `http://developers.sun.com/prodtech/cc` 以查找以下经常更新的资源：

- 有关编程技术和最佳实例的文章
- 有关编程小技巧的知识库
- 软件的文档，以及随软件一同安装的文档的更正信息
- 有关支持级别的信息
- 用户论坛
- 可下载的代码样例
- 新技术预览

Sun Studio 门户网站是 Sun Developer Network 网站 `http://developers.sun.com` 上的很多额外开发者资源之一。

联系 Sun 技术支持

如果您遇到通过本文档无法解决的技术问题，请访问以下网址：

```
http://www.sun.com/service/contacting
```

Sun 欢迎您提出意见

Sun 致力于提高其文档的质量，并十分乐意收到您的意见和建议。您可以通过以下网址提交您的意见和建议：

```
http://www.sun.com/hwdocs/feedback
```

请在电子邮件的主题行中注明文档的文件号码。例如，本文档的文件号码是 819-4761-10。

第1章

简介

本书与配套的 《Fortran 编程指南》中所介绍的 Sun™ Studio Fortran 95 编译器 `f95` 可在 SPARC®、 UltraSPARC® 和 x64/x86 平台上的 Solaris™ 操作环境下使用。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 标准一致性

- `f95` 设计与 ANSI X3.198-1992、ISO/IEC 1539:1991 和 ISO/IEC 1539:1997 标准文档兼容。
- 浮点运算基于 IEEE 标准 754-1985 和国际标准 IEC 60559:1989。
- `f95` 提供了对 SPARC® 和 x86 系列处理器体系结构优化开发功能的支持：Solaris 平台上的 UltraSPARC® 、 SPARC64、 AMD64、 Pentium 和 Xeon EM64T。
- 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。可能会修订或替代这些编译器遵循的适用标准的版本，因而导致 Sun Fortran 编译器将来版本中的功能与先前版本不兼容。

1.2 Fortran 95 编译器的功能

Sun Studio Fortran 95 编译器提供以下功能和扩展：

- 在例程中对参数、公共区等进行全局程序一致性检查。
- 优化多处理器系统的自动和显式循环并行化。
- VAX/VMS Fortran 扩展，其中包括：
  - 结构、记录、联合和映射
  - 递归
- OpenMP 2.5 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。
- Solaris 系统上相同的调用约定允许将使用 C 或 C++ 编写的例程与 Fortran 程序结合起来。
- UltraSPARC 和 AMD64 平台上支持 64 位的 Solaris 环境。
- 使用 `%VAL` 按值进行调用。
- Fortran 77 和 Fortran 95 程序和对象二进制文件之间的兼容性。
- 区间运算编程。
- 某些 Fortran 2003 功能，其中包括流 I/O。

有关每个软件发行版本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。

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。

- **Sun Studio 性能分析器** — 单线程和多线程应用程序的高级性能分析工具。请参见 analyzer(1)。
- **`asa`** — 此 Solaris 实用程序是一个 Fortran 输出过滤器，用于打印在第一列中包含 Fortran 回车控制符的文件。可使用 `asa` 将按照 Fortran 回车控制约定设置格式的文件转换为按照 UNIX 行打印机约定设置格式的文件。请参见 `asa`(1)。
- **`fdumpmod`** — 显示文件或归档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 文件分成几个文件，每个文件包含一个例程。可使用 FORTRAN 77 或 Fortran 95 源文件上的 `fsplit`。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。
- **Sun Studio `dbx`** — 提供强大、功能丰富的运行时和静态调试器，还包括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 (`www.netlib.org`) 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。

详细信息，请参见 `performance_library` 自述文件和 《Sun 性能库用户指南》。（性能库例程的手册页位于第 3P 节。）

1.6 区间运算

Fortran 95 编译器提供编译器标记 `-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。详细信息，请参见 《Fortran 95 区间运算编程指南》。（只有 SPARC/UltraSPARC 平台支持区间运算功能。）

1.7 手册页

联机手册 (`man`) 页提供了关于命令、函数、例行程序以及收集这些信息的文档。要访问 Sun Studio 手册页，请参见 “前言” 部分关于 `MANPATH` 环境变量的正确设置。

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，可使用 man 命令中的 -s 选项来访问由 ieee_flags(3M) 指示的其他章节：

```
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 Fortran 用户感兴趣的 man 页：

f95(1)	Fortran 95 命令行选项
analyzer(1)	Sun 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
ild(1)	目标文件的增量链接编辑器
ld(1)	目标文件的链接编辑器

1.8 自述文件

Sun Developer Network (SDN) 门户网站 (`http://developers.sun.com/sunstudio`) 上的自述文件页面介绍了新增功能、软件不兼容性、错误以及手册印刷后发现的信息。这些自述文件页面是门户网站上此发行版本文档的一部分，也可以从已安装软件包含的 HTML 文档索引（位于 `file:/opt/SUNWspro/docs`）链接到这些页面。

表 1-1 重要自述文件页面

自述文件页面	描述 ...
`fortran_95`	此版本 Fortran 95 编译器 `f95` 的新增功能、修改的功能、已知限制和文档勘误表。
`fpp_readme`	`fpp` 功能概述
`interval_arithmetic`	`f95` 中的区间运算功能概述
`math_libraries`	可用的优化和专用数学库。
`profiling_tools`	使用性能配置工具 `prof`、`gprof` 和 `tcov`。
`runtime_libraries`	可依照最终用户许可协议的条款重新分发的库和可执行文件。
`performance_library`	Sun 性能库概述
`openmp`	OpenMP 并行化 API 中的新增功能和已更改功能

可使用 `-xhelp=readme` 命令行选项显示每个编译器的自述文件页面的 URL。例如，命令：

```
% f95 -xhelp=readme
```

显示用于查看 SDN 门户网站上此发行版本的 `fortran_95` 自述文件的 URL。

1.9 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

使用 Fortran 95

本章介绍如何使用 Fortran 95 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，编译器还可能会自动调用系统链接程序来生成可执行文件。

Fortran 95 编译器还可以用于：

- 生成并行的可执行文件以用于多处理器 (`-openmp`)。
- 跨源文件和子例程分析程序一致性并生成报告 (`-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。
- 在启用运行时调试的情况下编译可执行文件 (`-g`)。
- 使用运行时语句或过程级文件配置进行编译 (`-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 95 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` 或 `.F95` 的 Fortran 源文件；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并执行该程序：

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%
```

在本示例中，f95 编译源文件 greetings.f，并在默认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。

习惯上，UNIX 编译器将可执行输出写入到名为 a.out 的默认文件中。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，那么在下次运行编译器时它将会被覆盖。因此，请使用 -o 编译器选项来显式地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%
```

在前面的示例中， -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，通常给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 命令来重命名默认的 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、编译器源文件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记。（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 95 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。这些文件将被视为等价于 .f95 和 .F95，并且可以作为表示源文件包含 Fortran 2003 扩展名的一种方式。

第 2-3 页的 2.2.2 节“命令行文件命名约定”描述了该编译器接受的各种源文件的扩展名。

2.2.1 编译和链接序列

在上一示例中，编译器自动生成装载器目标文件 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，目标文件 growth.o 和 fft.o 将保留。此约定允许您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，也不会生成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。如果文件没有扩展名，或者其后缀扩展名不是以下列出的某个扩展名，那么这些文件将被传递给链接程序。

表 2-1 由 Fortran 95 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，将目标文件放在当前目录中；目标文件的默认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

表 2-1 由 Fortran 95 编译器识别的文件名后缀 （续）

后缀	语言	操作
.for	Fortran 77 或 Fortran 95	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。
.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。

第 4 章中介绍了 Fortran 95 自由格式。

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》描述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这两种源文件预处理程序中的某个来扩展源代码“宏”和符号定义。默认情况下，编译器将使用 fpp；-xpp=cpp 选项将默认设置由 fpp 更改为 cpp。（另请参见有关 -D*name* 选项的讨论。）

fpp 是 Fortran 特有的源文件预处理程序。详细信息，请参见 fpp(1) 手册页和 fpp 自述文件。默认情况下，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 Web 站点下载：

```
http://www.netlib.org/fortran/
```

有关标准 Unix C 语言预处理程序的详细信息，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

您可以用分别的步骤编译和链接。 -c 选项编译源文件并生成 .o 目标文件，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                              （创建新的目标文件）
demo% f95 -o prgrm file1.o file2.o file3.o        （创建可执行文件）
```

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目标文件，则链接将会失败，并出现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是至关重要的。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，然后进行分步链接：

```
demo% f95 -c -fast sbr.f
demo% cc -c -fast simm.c
demo% f95 -fast sbr.o simm.o          链接步骤；将 -fast 传递给链接程序
```

2.2.7 无法识别的命令行参数

编译器无法识别的命令行参数将被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 -）生成警告。
- 无法识别的**非选项**（无 -）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，那么将生成链接程序错误消息。

例如：

```
demo% f95 -bit move.f              <- -bit 不是可识别的 f95 选项
f95:Warning: 如果调用了 ld，则将选项 -bit 传递至 ld，否则会将其忽略
demo% f95 fast move.f              <- 用户本意是要键入 -fast
ld:fatal: 文件 fast：打开失败； errno=2
ld:fatal: 文件处理失败。 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，用户想键入 f95 选项 -fast，但却忽略了开头短线。编译器再次将参数传递给链接程序，而后者将其解释为一个文件名。

这些示例表明在编写编译器命令行时应极其小心！

2.2.8 Fortran 95 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。

有关如何设置编写和搜索模块信息文件时的默认路径的信息，请参见 -M*path* 和 -moddir *dirlist* 选项标记。

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。

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

有关详细信息，请参见第 4-21 页的 4.9 节 “模块文件”。

2.3 指令

可使用源代码**指令**（一种 Fortran 注释格式）将有关特殊优化或并行化选项的特定信息传递给编译器。有时，编译器指令也被称为 *pragma*。编译器可识别一组通用指令和并行化指令。 Fortran 95 还可处理 OpenMP 共享内存多处理指令。

第 4-19 页的 4.8 节“指令”中描述了 `f95` 特有的指令。附录 D 中提供了 `f95` 可识别的所有指令的完整摘要。

注 – 指令并不是 Fortran 标准的一部分。

2.3.1 通用指令

通用 Fortran 95 指令的形式包括：

```
C$PRAGMA keyword ( a [ , a ] ...) [ , keyword ( a [ , a ] ...) ] , ...
C$PRAGMA  SUN keyword ( a [ , a ] ...) [ , keyword ( a [ , a ] ...) ] , ...
C$PRAGMA  SPARC keyword ( a [ , a ] ...) [ , keyword ( a [ , a ] ...) ] , ...
```

变量 *keyword* 代表特定的指令。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` 或 `SPARC`，如上所示。）

通用指令使用以下语法：

- 在第一列中，使用以下任何注释指示符：`c`、 `C`、 `!` 或 `*`
- 对于 `f95` 自由格式， `!` 是唯一可识别的注释指示符 (`!$PRAGMA`)。本章中的示例假定采用固定格式。
- 后面 7 个字符为大写或小写的 `$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：

- 和 Fortran 文本一样，在前 8 个字符之后，空格将被忽略，并且大写和小写字母等价。
- 因为是注释，所以指令不能连续，但您可以根据需要使用多个连续的 `C$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`** 的指令时要特别小心。`cpp` 会将小写字母 **`sun`**

替换为预定义的值。另外，如果您定义了一个 cpp 宏 **SUN**，那么它可能与 **SUN** 指令关键字冲突。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）：

```
C$PRAGMA Sun UNROLL=3
```

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	C$PRAGMA C(*list*) 将一系列外部函数的名称声明为 C 语言例程。
IGNORE_TKR 指令	C$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伪参数名称的类型、种类和等级。
UNROLL 指令	C$PRAGMA SUN UNROLL=*n* 建议编译器将以下的循环解开为指定的长度 *n*。
WEAK 指令	C$PRAGMA WEAK(*name*[=*name2*]) 将 *name* 声明为弱符号，或者声明为 *name2* 的别名。
OPT 指令	C$PRAGMA SUN OPT=*n* 将子程序的优化级别设置为 *n*。
PIPELOOP 指令	C$PRAGMA SUN PIPELOOP=*n* 断言下面的循环中在间隔为 *n* 的迭代之间存在依存关系。
NOMEMDEP 指令	C$PRAGMA SUN NOMEMDEP 断言下面的循环中没有内存依存关系。
PREFETCH 指令	C$PRAGMA SPARC_PREFETCH_READ_ONCE(*name*) C$PRAGMA SPARC_PREFETCH_READ_MANY(*name*) C$PRAGMA SPARC_PREFETCH_WRITE_ONCE(*name*) C$PRAGMA SPARC_PREFETCH_WRITE_MANY(*name*) 请求编译器为名称引用生成预取指令。（要求使用 -xprefetch 选项。）
ASSUME 指令	C$PRAGMA [BEGIN] ASSUME (*expression* [, *probability*]) C$PRAGMA END ASSUME 断言编译器可认为程序中某些点的条件为真。

2.3.1.1 C 指令

C() 指令指定其参数为外部函数。它与 EXTERNAL 声明等价，但有一点例外：与普通的外部名称不同，Fortran 编译器在这些参数名称的后面不附加下划线。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，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
C$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伪参数名称的类型、种类和等级。

例如，在下面的过程接口中，此指令指定 SRC 可以是任意数据类型，但 LEN 可以是 KIND=4 或 KIND=8。

此接口块为通用过程名称定义两个特定的过程。
此示例以 Fortran 95 自由格式显示。

```
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
```

子例程调用：

```
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，它调用 BLCK_64。S 的实际类型并不能决定要调用哪个例程。对于基于参数类型、种类或等级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。

注意，无法在此指令中指定假定形状数组、Fortran 指针或可分配数组的伪参数。如果未指定名称，则该指令将应用于过程的所有伪参数，但假定形式数组、Fortran 指针或可分配数组的伪参数除外。

2.3.1.3 UNROLL 指令

UNROLL 指令要求您在 C$PRAGMA 后面指定 SUN。

C$PRAGMA SUN UNROLL=*n* 指令通知编译器在其优化传递过程中将以下的循环解开 *n* 次。（只有在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。选项有：

- 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。详细信息，请参见《Fortran 编程指南》中有关性能和优化的章节。

示例 — 解开循环两次：

```
C$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 主要用在源文件中以创建库。如果链接程序无法解析弱符号，它并不生成错误消息。

```
C$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，它并不生成错误消息。

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。详细信息，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 C$PRAGMA 后面指定 SUN。

OPT 指令设置子程序的优化级别，它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：

```
C$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。

您还必须使用 -xmaxopt[=*n*] 选项来编译例程以识别该指令。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 NOMEMDEP 指令

NOMEMDEP 指令要求您在 C$PRAGMA 后面指定 SUN。

此指令必须紧挨在 DO 循环前面出现。它向优化器断言，在循环的迭代中没有基于内存的依存关系来禁止并行化。要求使用 -parallel 或 -explicitpar 选项。

2.3.1.7 PIPELOOP=*n* 指令

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

此指令必须紧挨在 DO 循环前面出现。*n* 是正整型常数或零，它向优化器断言循环迭代之间是否存在依存关系。值零表示循环中没有迭代间的（即循环带有的）依存关系，优化器可以对循环执行任意管线处理。正值 *n* 意味着，循环的第 I 次迭代与第 (I-*n*) 次迭代之间存在依存关系，每次最多只能对 *n* 个迭代进行管线处理。

```
C     我们知道在使用值 K 时，
C     迭代间不能存在依存关系 （例如， K>N）
C$PRAGMA SUN PIPELOOP=0
       DO I=1,N
        A(I)=A(I+K) + D(I)
        B(I)=B(I) + A(I)
      END DO
```

有关优化的详细信息，请参见 《Fortran 编程指南》。

2.3.1.8 PREFETCH 指令

-xprefetch 选项标记第 79 页的 "-xprefetch[=a[,a]]" 允许使用一组 PREFETCH 指令，这些指令建议编译器为指定的数据元素生成预取指令。预取指令仅在 UltraSPARC 平台上可用。

```
C$PRAGMA SPARC_PREFETCH_READ_ONCE(name)
C$PRAGMA SPARC_PREFETCH_READ_MANY(name)
C$PRAGMA SPARC_PREFETCH_WRITE_ONCE(name)
C$PRAGMA SPARC_PREFETCH_WRITE_MANY(name)
```

有关预取指令的进一步信息，另请参见 《C 用户指南》或 《SPARC 体系结构手册》，版本 9。

2.3.1.9 ASSUME 指令

ASSUME 指令向编译器提供有关程序中某些点上的条件的提示。这些断言可以帮助编译器指导其优化策略。编程人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：

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

而 “范围断言” ASSUME 的语法是：

```
C$PRAGMA BEGIN ASSUME [expression [, probability)
     语句块
C$PRAGMA END ASSUME
```

请使用点断言格式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声明适用于闭合的语句范围的条件。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计算出来，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，但并不必然为真。probability 并不恰好为 0 或 1 的断言是**非必然断言**。类似地， probability 恰好为 0 或 1 的断言是**必然断言**。

例如，如果编程人员知道 DO 循环的长度始终大于 10,000，则给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。

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

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

`__branchexp()`	用于紧挨在包含布尔控制表达式的分支转移语句前面的点断言。它产生与控制分支转移语句的布尔表达式相同的结果。
`__tripcount()`	得出紧跟在指令后面的或指令所包含的循环的行程数。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，它应用于最外层的闭合循环。

在将来的版本中，特殊内部函数的列表可能会扩展。

与 -xassume_control 编译器选项结合使用。（请参见第 57 页的 "-xassume_control[=keywords]"）例如，在使用 -xassume_control=check 进行编译时，如果出现行程数小于 10,000 的情况，则上述示例将生成一条警告。

如果使用 -xassume_control=retrospective 进行编译，那么在程序终止时，将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：

```
C$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

并行化指令显式地请求编译器尝试并行处理该指令后面的 DO 循环或代码区域。其语法与一般指令不同。只有在使用 -openmp、-parallel 或 -explicitpar 编译选项时，才能识别并行化指令。有关 Fortran 并行化的详细信息，请参见《OpenMP API 用户指南》和 《 Fortran 编程指南》。

Fortran 编译器支持 OpenMP 2.5 共享内存并行化模型。传统的 Sun 和 Cray 并行化指令现已过时。

编译器的并行化功能在 Solaris x86 平台上也可用。

2.3.2.1 OpenMP 并行化指令

Fortran 95 编译器将 OpenMP Fortran 共享内存多处理 API 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 `-openmp` 进行编译。(请参见第 37 页的 "`-openmp[={parallel|noopt|none}]`"。)

有关 `f95` 接受的 OpenMP 指令的详细信息，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。编译器仍然能够识别这些指令，但在以后的 Sun Studio 发行版本中可能会发生变化。建议使用 OpenMP 并行化 API。《OpenMP API 用户指南》中提供了有关如何从传统的 Sun/Cray 指令迁移到 OpenMP 模型的信息。

Sun 风格的并行化指令是 `-parallel` 和 `-explicitpar` 的默认设置。Sun 指令具有指令**标记** `$PAR`。

Cray 风格的并行化指令具有标记 `MIC$` （这些指令是用 `-mp=cray` 编译器选项启用的）。在 Sun 和 Cray 风格中，类似指令的解释是不同的。详细信息，请参见 《Fortran 编程指南》中有关并行化方面的章节。有关将传统的 Sun/Cray 并行化指令转换为 OpenMP 指令的指导原则，另请参见 《OpenMP API 用户指南》。

Sun/Cray 并行化指令使用以下语法：

- 第一个字符必须在第一列中。
- 第一个字符可以是以下任一字符：`c`、 `C`、 `*` 或 `!`。
- 后 4 个字符可以是大写或小写的 `$PAR` （Sun 风格）或 `MIC$` （Cray 风格），字符之间没有空格。
- 然后是指令关键字和限定符，它们之间用空格分隔。显式的并行化指令关键字有：

  `TASKCOMMON`、 `DOALL`、 `DOSERIAL` 和 `DOSERIAL*`

每个并行化指令都具有自己的一组可选限定符 （放在关键字后面）。

示例：使用共享变量指定一个循环：

```
C$PAR DOALL SHARED(yvalue)      Sun 风格
CMIC$ DOALL SHARED(yvalue)      Cray 风格
```

2.4 库接口和 system.inc

Fortran 95 编译器提供一个包含文件 system.inc，它为大多数非内在库例程定义了接口。请声明此包含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 更改了默认数据类型时。

例如，以下命令可能会产生一个算术异常，原因是没有显式地设定函数 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的类型，则编译器认为它返回一个实数值。此值被进一步转换为整数，很有可能会导致浮点错误。

要纠正这种错误，您应该显式地设定所调用的 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 95 包含文件“system.inc”为这些例程提供显式的接口定义。

```
        include 'system.inc'
        integer(4) mypid
        mypid = getpid()
        print *, mypid
```

通过在调用 Fortran 库中的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。（详细信息，请参见《Fortran 库参考》。）

2.5 编译器用法提示

后面几节向您建议了一些高效地使用 Fortran 95 编译器的方法。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

某些编译器标记允许用户使用一组特定的硬件平台选项来优化代码生成。可以使用编译器的 `-dryrun` 选项确定本机处理器：

```
<sparc>f95 -dryrun -xtarget=native
###     command line files and options (expanded):
### -dryrun -xarch=v8plusb -xcache=64/32/4:1024/64/4 -xchip=ultra3i

<x64>f95 -dryrun -xtarget=native
###     command line files and options (expanded):
### -dryrun -xarch=sse2 -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` 变量并且 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` 将导致相当于以下内容的编译：

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

make 是一个功能很强的程序开发工具，可以方便地将其用于所有的 Sun 编译器。请参见 make(1) 手册页和 《Fortran 编程指南》中的 **“程序开发”**一章。

注 – make 使用的默认隐式规则可能无法识别具有 .f95 和 .mod （Fortran 95 模块文件）扩展名的文件。详细信息，请参见 《Fortran 编程指南》和 Fortran 95 自述文件。

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和待编译文件的大小和复杂性。在 SPARC 平台上，如果优化器内存不足，它将尝试通过在较低的优化级别中重试当前的过程来进行恢复，并以命令行上 -O*n* 选项中指定的原始级别继续后续的例程。

运行编译器的处理器应该至少具有 64 MB 的内存；建议使用 256 MB 的内存。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

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
```

确定实际的真实内存：

```
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) 来增加工作站上交换空间的大小。要执行此操作，您必须成为超级用户。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很大的例程时，可能需要额外的内存，这可能会降低编译时的性能。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
```

这些命令行中的每一个都将导致优化器尝试在数据空间为 16 MB 的情况下进行恢复。

此限值不能大于总的系统可用交换空间，并且实际上在进行较大的编译时，此限值必须很小以保证还能够正常地使用系统。请确保没有任何编译占用一半以上的空间。

示例：在使用 32 MB 的交换空间时，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和可用真实内存和虚拟内存的数量。

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` 命令删除该限制。

对于 `csh`，请使用：

```
demo% limit datasize unlimited
```

对于 `sh` 或 `ksh`，请使用：

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

详细信息，请参见 《Solaris 64 位开发者指南》。

第3章

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 页的 3.1 节 “命令语法”开始是对用于编译器选项标志的语法的描述。
- 从第 3-3 页的 3.3 节 “选项摘要”开始是按功能排列的选项的摘要。
- 从第 3-10 页的 3.4 节“选项参考”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* 是由空格隔开的源文件名、目标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，而且这些选项可能包括其他文件列表。

3.2 选项语法

典型的编译器选项格式如下：

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 ”-”）符号用来隔开参数，您只能选择其中的**一个**参数。	`-B{dynamic|static}` `-Bstatic`
:	与逗号一样，分号有时可用于分隔参数。	`-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，编译器选项是从左向右处理的，允许选择性地覆盖宏选项 （包括其他选项的选项）。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如果显示在命令行的选项标志不带值时假定的值。例如， `-xhasc` 等价于 `-xhasc=yes`。

- 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，按功能对编译器选项进行分组，以便于参考。有关详细信息，请参见以下几节中指明的页。

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。该表不包括已废弃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`
为编写已编译的 `.mod` 模块文件指定路径	`-moddir=`*path*
指定要编写的目标文件、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`
冗余消息	`-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*
已编译代码：	
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`

表 3-3 按功能分组的编译器选项 （续）

功能	选项标志
内联某些数学库例程	`-libmil`
STOP 将整数状态值返回给 shell	`-stop_status[=`*yn*`]`
指定代码地址空间	`-xcode=`*x*
启用 UltraSPARC 预取指令	`-xprefetch[=`*x*`]`
指定可选寄存器的使用	`-xregs=`*x*
指定默认数据映射	`-xtypemap=`*x*
数据对齐：	
指定对齐 COMMON 块中的数据	`-aligncommon[=`*n*`]`
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`
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`
为使用源浏览器浏览而进行编译	`-sb`、 `-sbfast`
标志未声明变量的使用	`-u`
检查 C$PRAGMA ASSUME 断言	`-xassume_control=check`
检查在运行时栈是否溢出	`-xcheck=stkovf`
启用运行时任务普通检查	`-xcommonchk`
为性能分析器进行编译	`-xF`
生成交叉引用列表	`-Xlist`*x*
在没有目标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`
禁止特定错误消息	`-erroff=`*list*
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`
冗余消息	`-v`

表 3-3 按功能分组的编译器选项 （续）

功能	选项标志
冗余的并行化消息	`-vpara`
显示 / 禁止警告消息	`-w` *n*
显示编译器自述文件	`-xhelp=readme`
许可：	
显示许可证服务器信息	`-xlicinfo`
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*
将运行时库搜索路径生成到可执行文件中	`-R`*path*
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`
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 SPARC 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	`-fpround=`*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*`]`

表 3-3 按功能分组的编译器选项 （续）

功能	选项标志
优化与性能：	
分析循环以了解数据相关性	`-depend`
使用所选的选项进行优化	`-fast`
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*`]`
在内存栈上分配局部变量	`-stackvar`
启用循环解开	`-unroll[=`*m*`]`
启用跨源文件的优化	`-xcrossfile[=`*n*`]`
调用过程间优化传递	`-xipo[=`*n*`]`
为 `#pragma OPT` 设置最高优化级别	`-xmaxopt[=`*n*`]`
为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*
启用性能文件配置数据的生成或使用	`-xprofile=`*p*
断言不会出现基于内存的陷阱	`-xsafe=mem`
不执行增加代码大小的优化	`-xspace`
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`
启用由指令显式标记的循环的并行化	`-explicitpar`
显示循环的并行化信息	`-loopinfo`
指定 Cray 风格的并行化指令	`-mp=CRAY`
为手动编码的多线程编程进行编译	`-mt`
接受 OpenMP API 指令并设置相应的环境	`-openmp[=`*keyword*`]`
使用 `-autopar -explicitpar -depend` 组合对循环进行并行化	`-parallel`
识别具有自动并行化的循环中的约简操作	`-reduction`
冗余的并行化消息	`-vpara`
源代码：	
定义预处理程序符号	`-D`*name*`[=`*val*`]`
未定义预处理程序符号	`-U`*name*

表 3-3 按功能分组的编译器选项 （续）

功能	选项标志
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` 预处理程序预处理所有源文件	`-fpp`
接受 Fortran 95 自由格式输入	`-free`
将目录添加到包含文件搜索路径	`-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：	
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

编译器具有许多可以供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，供您一睹为快。

表 3-4 常用选项

操作	选项
调试 — 为确保参数、通用块等的一致性在例程之间进行的全局程序检查。	`-Xlist`
调试 — 产生启用 `dbx` 和调试的其他符号表信息。	`-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，为本机平台产生高效的编译时和运行时。	`-fast`
动态 (`-Bdynamic`) 或静态 (`-Bstatic`) 库绑定。	`-B`*x*

表 3-4 常用选项 （续）

操作	选项
仅编译 — 禁止链接；为每个源文件生成 .o 文件。	`-c`
输出文件 — 命名可执行输出文件 *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

某些选项标志是扩展为由其他标志组成的特定集合的宏。提供这些选项标志的目的是，便于指定通常一起使用以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`	`-xO5 -libmil -fsimple=2 -dalign -xlibmopt -depend -fns -ftrap=common -pad=local -xvector=yes -xprefetch=yes` *(SPARC)* `-xprefetch_level=2` *(SPARC)* `-nofstore` *(x86)*
`-fnonstd`	`-fns -ftrap=common`
`-parallel`	`-autopar -explicitpar -depend`
`-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宏扩展。例如，要使用 `-fast`，但优化级别是 `-O3`，则在命令行上 `-O3` 必须跟在 `-fast` 后面。

3.3.3 向后兼容性和传统选项

提供以下选项的目的是与早期编译器发行版本和某些 Fortran 传统功能向后兼容。

表 3-6 向后兼容性选项

操作	选项
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 — 使用单行程 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 95 程序时不要使用这些选项标志。

3.3.4 已废弃的选项标志

以下的选项被认为是已废弃的，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 已废弃的 `f95` 选项

选项标志	等效于
`-a`	`-xprofile=tcov`
`-cg89`	`-xtarget=ss2`
`-cg92`	`-xtarget=ss1000`
`-explicitpar`	使用 OpenMP 并行化； Sun/Cray 并行化已过时
`-mp`	使用 OpenMP 并行化； Sun/Cray 并行化已过时
`-native`	`-xtarget=native`
`-noqueue`	许可证排队。不再需要。
`-p`	文件配置。使用 `-pg` 或性能分析器
`-pic`	`-xcode=pic13`
`-PIC`	`-xcode=pic32`

表 3-7 已废弃的 f95 选项 （续）

选项标志	等效于
-sb	不再需要。
-sbfast	不再需要。
-silent	不再需要。

3.4 选项参考

本节说明了所有的 f95 编译器命令行选项标志，包括各种风险、限制、警告、交互作用、示例和其他详细信息。

除非特别指定，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 **(***Obsolete***)** 的选项标志已废弃，不能再使用。在许多情况下，它们已经被应该使用的选项或标志取代。

-a

（已废弃）使用 tcov 按基本块进行文件配置，这是旧式用法。

这是 tcov 的基本块文件配置的旧式用法。有关文件配置的新式用法，请参见 -xprofile=tcov；有关更多详细信息，请参见 tcov(1) 手册页。

-aligncommon[={1|2|4|8|16}]

指定通用块和标准数值序列类型中的数据对齐量。

此值表示通用块和标准数值序列类型中数据元素的最大数据对齐量 （以字节为单位）。

注 – 标准数值序列类型是包含 SEQUENCE 语句以及唯一的默认组件数据类型（INTEGER、REAL、DOUBLEPRECISION、COMPLEX，不带 KIND= 或 **size*）的派生类型。任何其他类型 （例如 REAL*8）将使类型成为非标准类型。

例如，-aligncommon=4 将按 4 字节边界对齐自然对齐等于或大于 4 字节的数据元素。

此选项不影响自然对齐小于指定大小的数据。

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

在没有默认值为 1 的情况下指定 -aligncommon — 所有按字节边界对齐的通用块和数值序列类型元素（元素之间无填充）。

在未启用 64 位的平台（除 v9、v9a 或 v9b 之外的平台）上，-aligncommon=16 回复为 -aligncommon=8。

-ansi

标识许多非标准扩展。

如果在源代码中使用非标准 Fortran 95 扩展，则发出警告消息。

-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 使用复制 / 恢复保留伪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。

-autopar

启用自动循环并行化。

查找和并行化相应的循环，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相关性并重构循环。如果未将优化级别指定为 -O3 或更高，则将它自动提升到 -O3。

在使用任何并行化选项（包括 -autopar）时，也要指定 -stackvar 选项。

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 3-33 页的 "-mt" 中的注意事项。

-autopar 选项不适合于单处理器系统，而且已编译的代码通常会运行更慢。

要在多线程环境中运行已并行化的程序，必须在执行之前设置 PARALLEL（或 OMP_NUM_THREADS）环境变量。这通知运行时系统程序可以创建的最大线程数。默认值为 1。一般会将 PARALLEL 或 OMP_NUM_THREADS 变量设置为目标平台上可用的处理器数。

如果使用 祆 topar 并在**同一**步骤中进行编译和链接，则将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**单独**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了相应的库。

-reduction 选项与 -autopar 一起使用也可能是很有用的。其他并行化选项是 -parallel 和 -explicitpar。

有关并行化的详细信息，请参见《Fortran 编程指南》。

-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，则默认值为 -Bdynamic。

- -Bdynamic：首选**动态**链接 （试图找到共享库）。
- -Bstatic：要求**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，则不链接库，并发出警告 “未找到库”。
- 如果指定 dynamic，但链接程序仅找到静态版本，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和动态链接一些库，如下所示：

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

这些是加载器和链接程序选项。使用编译命令上的 -B*x* 在单独步骤中编译和链接时，需亦在链接步骤中使用它。

您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（而不提供 libm.a 和 libc.a） 。这意味着，在 64 位 Solaris 环境中，-Bstatic 和 -dn 可能导致链接错误。这些情况下应用程序必须与动态库链接。

有关静态库和动态库的详细信息，请参见 《Fortran 编程指南》。

-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。

引用的下标超过数组声明大小可能导致意外结果（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 还在运行时检查数组语法表达式中的数组一致性。

指定 -D 可能使可执行文件更大。

如果使用 -C 选项，则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违规，则将它视为编译错误。

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，然后重新编译成品可执行程序，而无须进行下标检查。

-c

仅编译；产生对象 .o 文件，但禁止链接。

为每个源文件编译 .o 文件。如果仅编译单个源文件，则可以使用 -o 选项指定写入的 .o 文件的名称。

-cg89

（已废弃， SPARC）为通用 SPARC 体系结构进行编译。

此选项是以下选项的宏：-xarch=v7 -xchip=old -xcache=64/32/1，它等效于 -xtarget=ss2。

-cg92

（已废弃， SPARC）为 SPARC V8 体系结构进行编译。

此选项是以下选项的宏：
-xarch=v8 -xchip=super -xcache=16/32/4:1024/32/1，它等价于 -xtarget=ss1000。

-copyargs

允许为常量参数赋值。

允许子程序更改其为常量的伪参数。提供此选项只是为了允许编译和执行传统的代码而不发出运行时错误。

- 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。

-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，此选项将这样定义 *name*：就好像

#define *name*[=*def*]

已经出现在源文件中。如果未指定 *=def*，则将名称 *name* 定义为值 1。将宏符号 *name* 传递给预处理程序 fpp （或 cpp — 请参见 -xpp 选项）以进行扩展。

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 - 它们仅出现在 fpp 或 cpp 预处理程序指令中。（请注意两个前导下划线。）

- 产品版本是在 __SUNPRO_F90 和 __SUNPRO_F95 中预定义的（用十六进制表示）。例如，对于 Sun Studio 10 发行版本， __SUNPRO_F95 是 0x810。
- 以下宏是在相应系统上预定义的：

  __sparc、__unix、__sun、__SVR4、__i386、
  __SunOS_5_6、__SunOS_5_7、__SunOS_5_8、__SunOS_5_9、
  __SunOS_5_10

  例如，值 __sparc 是在 SPARC 系统上定义的。
- 以下的预定义值不带下划线，但是在以后的发行版本中可能删除这些值： sparc、unix、sun
- 在 SPARC V9 系统上，还定义了 __sparcv9 宏。
- 在 64 位 x86 系统上，定义了宏 __amd64 和 __x86_64。

使用冗余选项 (-v) 进行编译以查看编译器创建的定义。

您可以在类似如下的预处理程序条件中使用这些值：

```
#ifdef __sparc
```

默认情况下，f95 使用 fpp(1) 预处理程序。与 C 预处理程序 cpp(1) 一样，fpp 扩展源代码宏并启用代码的条件编译。与 cpp 不同， fpp 理解 Fortran 语法，并且作为首选的 Fortran 预处理程序。使用 -xpp=cpp 标志可以将编译器强制为明确使用 cpp，而不是使用 fpp。

-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。

此标志更改 COMMON 块中的数据布局、数值序列类型和 EQUIVALENCE 类，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是根据其“自然”对齐（按 8 字节边界；使用 -xarch=v9 或 v9a 为 64 位环境编译时，四精度数据按 16 字节边界对齐）布局的。默认情况下，按 4 字节边界对齐 COMMON 块中的数据。还允许编译器假定自然对齐并生成速度更快的多字装入 / 存储以引用数据。

使用 -dalign 以及 -xtypemap=real:64,double:64,integer:64 也可以导致 64 位整型变量在 SPARC 处理器上双字对齐。

注 – -dalign 可能导致数据的非标准对齐，这样就可能导致 EQUIVALENCE 或 COMMON 中的变量出现问题，并可能在需要 衔 align 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 编译一个子程序，请使用 -dalign 编译该程序的所有子程序。此选项包括在 -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 3-10 页的 "-aligncommon[={1|2|4|8|16}]"。

-dbl_align_all[={yes|no}]

强制按 8 字节边界对齐数据

值可以是 yes 或 no。如果是 yes，将按 8 字节边界对齐所有变量。默认值是 -dbl_align_all=no。

使用 -xarch=v9 或 v9a 为 64 位环境进行编译时，此标志将按 16 字节边界对齐四精度数据。

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。

-depend[={yes|no}]

分析循环以了解数据相关性并重构循环。

使用 -depend 或 -depend=yes 可启用相关性分析。使用 -depend=no （编译器的默认值）可禁用相关性分析。

如果未指定优化级别，或者指定的级别低于 O3，则此选项将优化级别提升到 O3。-depend 还包括在 -fast、-autopar 和 -parallel 中。另请注意，将优化级别指定为 -O3 或更高将自动增加 -depend。（请参见 《Fortran 编程指南》。）

-dn

禁止动态库。请参见第 3-16 页的 "-d{y|n}"。

-dryrun

显示由 f95 命令行驱动程序生成的命令，但不进行编译。

此选项在调试时是很有用的，它显示编译器为执行编译将调用的命令和子选项。

-d{y|n}

允许或禁止对整个可执行文件使用**动态**库。

- -dy：**允许使用**动态 / 共享库。
- -dn：**不允许使用**动态 / 共享库。

如果未指定，则默认值是 -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。如果使用这些选项在单独步骤中编译和链接，则在链接步骤中需要同一选项。

在 64 位 Solaris 环境中，许多系统库不仅仅作为共享动态库。其中包括 libm.so 和 libc.so（而不提供 libm.a 和 libc.a）。这意味着，衍 n 和 蠕 static 可能导致在 64 位 Solaris 环境、32 位 x86 Solaris 平台以及所有 32 位 Solaris 平台（从 Solaris 10 发行版本开始）上出现链接错误。这些情况下应用程序必须与动态库链接。

-e

接受扩展长度的输入源代码行。

扩展的源代码行的长度最大可以是 132 个字符。编译器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。

-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 %all，则禁止所有警告，它等价于 -w 选项。如果 %none，则不禁止警告。

示例：

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

使用 -errtags 选项可以查看与警告消息关联的标记名称。

-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。单独使用 -errtags 等价于 -errtags=yes。

默认情况下不显示标记 (-errtags=no)。

```
demo% f95 -errtags ink.f
ink.f:
 MAIN:
"ink.f", line 11:Warning:local variable "i" never used
(WDECL_LOCAL_NOTUSED)  <- 警告消息的标记名称
```

-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的标记名列表 （用逗号分隔）。如果使用标记 %all，则将所有警告视为错误。如果使用标记 %none，则不将警告视为错误。

另请参见 -errtags。

-explicitpar

（**已废弃**）对 Sun 或 Cray 指令显式标记的循环进行并行化。

注 – 此选项启用传统的 Sun 或 Cray 并行化指令。这些指令和并行化模型已过时，并且不再受支持。优先使用 OpenMP API，它是受支持的并行化模型。有关将 Sun/Cray 指令转换为 OpenMP 的详细信息，请参见 -xopenmp 选项以及 《OpenMP API 用户指南》。

即使 DO 循环中存在数据相关性 （在循环并行运行时，这些相关性将使循环生成错误结果），编译器也将生成并行代码。对于显式并行化，您需要在用并行化指令标记循环之前，正确地分析它们以了解数据相关性问题。

并行化仅适合于多处理器系统。

此选项启用 Sun 和 / 或 Cray 显式并行化指令。紧随在并行化指令之后的 DO 循环将为这些并行化指令生成线程代码。

要启用 OpenMP 显式并行化指令，请不要使用 -explicitpar。而改用 -openmp。请参见第 3-37 页的 "-openmp[={parallel|noopt|none}]"）

不得使用 -explicitpar 编译已经使用对 libthread 库的调用执行了自己的多线程处理的程序。

要在多线程环境中运行已并行化的程序，必须在执行之前设置 PARALLEL（或 OMP_NUM_THREADS）环境变量。这通知运行时系统程序可以创建的最大线程数。默认值为 1。一般会将 PARALLEL 或 OMP_NUM_THREADS 变量设置为目标平台上可用的处理器数。

如果使用 -explicitpar 并在**同一**步骤中编译和链接，则链接将自动包括多线程库和线程安全的 Fortran 运行时库。如果使用 -explicitpar 并在**单独**步骤中编译和链接，则还必须使用 衔 xplicitpar **进行链接**。

为了提高性能，在使用任何并行选项（包括 -explicitpar）时还要指定 -stackvar 选项。

使用 -mp 选项 （第 3-32 页的 "-mp={%none|sun|cray}"）可选择已启用的并行化指令的式样。默认情况下， -explicitpar 启用的是 Sun 指令。使用 -explicitpar -mp=cray 可启用 Cray 指令。

如果优化级别不是 -O3 或更高，则会自动将它提升到 -O3。

有关详细信息，请参见 《Fortran 编程指南》中的 “并行化”一章。

-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 或 underscores。默认值是 underscores。

-ext_names=plain：不增加结尾下划线。

-ext_names=underscores：增加结尾下划线。

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，也影响调用例程时使用的名称。使用此标志可允许 Fortran 95 例程调用其他编程语言例程 （以及被后者调用）。

-F

调用源文件预处理程序，但不进行编译。

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，并将处理结果写入一个同名文件，但该文件的扩展名更改为 .f （或者 .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 的默认预处理程序。通过指定 -xpp=cpp，可以改为选择 C 预处理程序 cpp。

-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，它等效于 -aligncommon=16。首选使用 -aligncommon。

默认情况下，按 4 字节边界对齐 COMMON 块中的数据。 -f 更改 COMMON 块和 EQUIVALENCE 类中双精度和四精度数据的数据布局，根据 “自然”对齐 （按 8 字节边界；使用 -xarch=v9 或 v9a 为 64 位 SPARC 环境编译时，四精度数据按 16 字节边界对齐）将其放置在内存中。

注 – -f 可能导致数据的非标准对齐，这样可能使 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。

使用 -f 编译程序的**任何**部分都要求使用 -f 编译该程序的**所有**子程序。

此选项本身并不使编译器能够生成针对双精度和四精度数据的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。相对于以前的选项 -f，请优先使用 -dalign。请参见第 3-14 页的 ”-dalign”。因为 -dalign 是 -fast 选项的一部分，因此 -f 也是它的一部分。

-f77[=*list*]

选择 Fortran 77 兼容性模式。

此选项标志允许将传统的 Fortran 77 源程序（包括具有由 f77 编译器接受的语言扩展的那些源程序）移植到 f95 Fortran 95 编译器。

list 是从以下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。
输入	允许 f77 接受的输入格式。
内部函数	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 - 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. 和逻辑操作数一起使用
misc	允许杂项 f77 Fortran 77 扩展。

关键字	含义
输出	生成 f77 式样的格式化输出，包括列表控制的输出和 NAMELIST 输出。
subscript	允许将非整数表达式作为数组下标。
tab	启用 f77 式样的制表符格式，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。

在所有关键字前面都可以加上 no%，以禁用该功能，如下所示：

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

如果未指定 -f77，则默认为 -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 95 捕获模式，即 -ftrap=common。在算术异常捕获方面，Fortran 95 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执行继续。使用 -f77 进行编译还可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 -f77 选项标志之后指定 -ftrap=%none 可以模仿原来的 Fortran 77 行为。

有关 f77 兼容性和从 Fortran 77 到 Fortran 95 的迁移的完整信息，请参见第 5 章。

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

-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，它会随版本及编译器的不同而变化。另外，-fast 选用的某些选项并非在所有平台上都可用。用 -dryrun 标志编译可查看 -fast 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 编译时程序行为不正常，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，以使程序保持正确的行为。

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。避免用 -fast 编译那些依赖浮点运算特殊属性的程序。

由于 -fast 选用的某些选项具有链接蕴含式，因此，如果分步进行编译和链接，还请务必用 -fast 进行链接。

-fast 会选用以下选项：

- -dalign

- `-depend` **(SPARC)**
- `-fns`
- `-fsimple=2`
- `-ftrap=common`
- `-libmil`
- `-xtarget=native`
- `-O5`
- `-xlibmopt`
- `-pad=local` **(SPARC)**
- `-xvector=yes` **(SPARC)**
- `-xprefetch=auto,explicit` **(SPARC)**
- `-xprefetch=auto` **(x86)**
- `-xprefetch_level=2`
- `-nofstore` **(x86)**

下面是有关 `-fast` 选择的选项的详细信息：

- `xtarget=native` 硬件目标。
  如果打算在与编译计算机不同的目标计算机上运行程序，请在 `-fast` 之后加上某个代码生成器选项。例如：

```
f95 -fast -xtarget=ultra ...
```

- `-O5` 优化级别选项。
- `-depend` 选项分析循环以了解数据相关性并重构循环（如有可能）。 **(SPARC)**
- `-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。
  如果要求严格遵循 IEEE 754 标准，则 `-fsimple=2` 是不合适的。请参见第 3-25 页的 "`-fsimple[={1|2|0}]`"。
- `-dalign` 选项，用于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（如果适当）以提高高速缓存利用率。 **(SPARC)**
- `-xvector=yes` 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。
- `-fns` 选择非标准浮点运算异常处理和渐进下溢。请参见第 3-23 页的 "`-fns[={yes|no}]`"。
- `-ftrap=common`，用于捕获常见的浮点异常，在 Fortran 95 中处于启用状态。
- 通过 `-xprefetch=auto,explict`，编译器能够在支持它的平台上生成硬件预取指令（如有必要），并且允许使用由用户提供的 `PREFETCH` 指令。 **(SPARC)**

- 通过 -xprefetch=auto，编译器能够在支持它的 x86 SSE2 平台上生成硬件预取指令（如有必要）。 **(x86)**
- -xprefetch_level=2 设置 插入预取指令的默认级别。
- -nofstore 取消强制表达式具有结果的精度。 **(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 ...
```

它覆盖 -fsimple=2 选项并禁用由 -fast 选择的 -xlibmopt。

因为 -fast 会调用 -dalign、-fns、-fsimple=2，所以用 -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。对于大多数应用程序，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可能随各个编译器发行版本而发生改变。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=v8plusb -xcache=64/32/4:1024/64/4
          -xchip=ultra3i -xdepend=yes -xpad=local -xvector=lib
          -xprefetch=auto,explicit -dalign -fsimple=2 -fns=yes
          -ftrap=common -xlibmil -xlibmopt -fround=nearest
```

-fixed

指定固定格式的 Fortran 95 源输入文件。

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

-flags

与 -help 等效。

-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本中已废弃的 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，而不是产生 IEEE 标准要求的次正规数，则将该结果刷新为零。

有关渐进下溢和次正规数的详细信息，请参见 -fns。

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

有关详细信息，请参见 ieee_handler(3m) 和 ieee_functions(3m) 手册页、《数值计算指南》和 《Fortran 编程指南》。

-fns[={yes|no}]

选择非标准浮点模式。

默认值是标准浮点模式 (-fns=no)。（请参见《Fortran 编程指南》的“浮点运算”一章。）

可选择使用 =yes 或 =no，此方法使您可以切换跟在某个其他宏标志 （如 -fast）后面的且包含于其中的 -fns 标志。不带值的 -fns 与 -fns=yes 等效。

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”，导致将微小的结果刷新为零，而不是产生次正规数。还会导致低于正常的操作数被默置为零。在不支持硬件中的渐进下溢和次正规数的那些 SPARC 系统上，使用此选项将大大提高某些程序的性能。

其中的 *x* 不会导致总下溢，当且仅当 |x| 在所示范围之一内时 *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?8
DOUBLE PRECISION	0.0 < |x| < 2.22507385072014e?08

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详细信息，请参见《Fortran 编程指南》的“浮点运算”一章。（一些算术家使用术语 *denormalized number* 代替 *subnormal number*。）

默认情况下，对浮点首选项进行标准初始化：

- IEEE 754 浮点运算是**不停止的**（在出现异常时不终止）。
- 下溢是渐进的。

在 x86 平台中，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 刷新为零模式以及反向规格数为零模式（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。此标志不会影响采用 SSE 或 SSE2 指令集的传统 x87 浮点运算。

主程序必须使用此选项进行编译才能有效。

-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)。默认情况下，不进行这样的溢出检测 (`-fpover=no`)。不带值的 `-fpover` 等效于 `-fpover=yes`。

-fpp

使用 `fpp` 强制预处理输入。

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 3-78 页的 ”-xpp={fpp|cpp}”。

-fprecision={single|double|extended}

(x86) 初始化非默认的浮点型舍入精度模式。

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

带有单精度或双精度值时，此标志将导致程序启动时分别将舍入精度模式设置为单精度或双精度。带有扩展值或在默认情况下，如果未指定 `-fprecision` 标志，舍入精度模式初始化为扩展精度。

此选项仅对 x86 系统且仅在编译主程序时使用才有效。

-free

指定自由格式的源输入文件。

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 文件解释为自由格式文件。

-fround={nearest|tozero|negative|positive}

设置启动时有效的 IEEE 舍入模式。

默认值是 -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。

当值是 tozero、 negative 或 positive 时，在程序开始执行时，此选项将舍入方向分别设置为**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 用作默认值，而且舍入方向是**舍入为最接近值**。含义与 ieee_flags 函数的相同。（请参见 《Fortran 编程指南》的 “浮点运算”一章。）

-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。）

为了获得一致结果，请使用相同的 -fsimple 选项编译程序的所有单元。

默认值为：

- 如果不使用 -fsimple 标志，则编译器默认为 -fsimple=0
- 使用不带值的 -fsimple 时，编译器使用 -fsimple=1

不同的浮点简化级别如下：

-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。产生的代码与 IEEE 754 不完全一致，但多数程序所产生的数值结果没有更改。

使用 -fsimple=1，优化器可以假定以下内容：

- 在进程初始化之后， IEEE 754 默认舍入 / 捕获模式不发生改变。
- 可以删除除潜在的浮点异常外不产生可见结果的计算。
- 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

允许主动浮点优化。这可以使某些程序因表达式求值方式的变化而产生不同的数值结果。尤其是，使用 -fsimple=2 可能会违反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的 Fortran 标准规则。对于依赖此规则的程序，这可能导致数值舍入差异。

例如，如果使用 -fsimple=2，则编译器可能将 C-(A-B) 计算为 (C-A)+B，这违反了有关显式圆括号的标准规则 （如果对结果代码进行了更好的优化）。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。

不得使用 -fsimple=2 编译依赖浮点计算的特定属性的程序。

即使是使用 -fsimple=2，也仍然不允许优化器在程序（否则，它不产生任何异常）中引入浮点异常。

衒 ast 选择 -fsimple=2。

-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。这是默认设置。但是，-fast 选项包括 -nofstore，可以禁用此选项。 -fstore 后跟随 -fast 可以重新打开此选项。

-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 是以下选项的宏
-ftrap=invalid,overflow,underflow,division。

f95 默认值是 -ftrap=common。这与 C 和 C++ 编译器的默认值不同，后者为 -ftrap=%none。

在启动时，设置有效的 IEEE 754 捕获模式，但不安装 SIGFPE 处理程序。您可以使用 ieee_handler(3M) 或 fex_set_handling(3M) 启用自陷，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。根据定义，常见异常是无效、被零除和溢出。

示例：-ftrap=%all,no%inexact 意味着设置所有陷阱，但 inexact 除外。

-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，并将导致捕获伪异常和预料异常。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。

有关进一步的信息，请参见 《Fortran 编程指南》的 “浮点运算”一章。

-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，则它生成动态库。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。

-g

为调试和性能分析进行编译。

产生其他符号表信息，以便使用 dbx(1) 调试实用程序进行调试以及使用性能分析器进行性能分析。

虽然在不指定 -g 的情况下有可能进行一些调试，但是 dbx 和 debugger 的完整功能只能供使用 -g 编译的那些编译单元使用。

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。尽管一些性能分析功能不要求使用 -g，但是您必须使用 -g 进行编译才能查看带注释的源代码、一些函数级信息和编译器注释性消息。（请参见分析器 (1) 手册页和手册 Sun Studio 《性能分析器》。）

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任何优化时，才出现注释性消息。在请求高优化级别 （如使用 -xO4 或 -fast）时，更有可能看到注释性消息。

-D*name*

指定已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 Solaris 《链接程序和库指南》和 《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，该名称正作为库的内部名称进行创建。-h 和 *name* 之间的空格是可选的（除非库名称是 elp，该库名称要求使用空格）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都将在链接程序正在搜索的任何路径中搜索具有相同内部名称的库。在指定内部名称的情况下，在运行时链接搜索库更灵活。此选项还可以用于指定共享库的**版本**。

如果没有共享库的内部名称，则链接程序改用共享库文件的特定路径。

-help

显示编译器选项的摘要列表。

另请参见第 3-68 页的 "-xhelp={readme|flags}"。

-I*path*

将 *path* 增加到 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* 之间不允许有空格。忽略无效目录，且不出现警告消息。

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 语句上的文件名）的目录的列表。

示例：在 /usr/app/include 中搜索 INCLUDE 文件：

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

多个 -I*path* 选项可能出现在命令行上。每个选项都增加到搜索路径列表的顶部（搜索的第一个路径）。

INCLUDE 或 #include 上的相对路径的搜索次序如下：

1. 包含源文件的目录
2. 在 -I 选项中指定的目录
3. 编译器内部默认列表中的目录
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。

-inline=[%auto][[,][no%]*f1*,_[no%]*fn*]

启用或禁用指定例程的内联。

要求优化器内联在函数和子例程名称列表中 （用逗号分隔）显示的用户编写的例程。在例程名之前加上 no% 可禁用该例程的内联。

内联是一种优化方法，通过该方法编译器可以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机会以产生高效代码。

指定 %auto 可以在优化级别 -O4 或 -O5 上启用自动内联。使用 -inline 指定显式内联时，在这些优化级别的自动内联正常关闭。

示例：内联例程 xbar、 zbar 和 vpoint：

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

以下是一些限制；不发出任何警告：

- 优化必须是 -O3 或更高级别。
- 例程的源代码必须位于正编译的文件中，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，则编译器通常试图内联所有适当的用户编写子例程和函数。 -inline 与 -O4 一起使用可能降低性能，因为优化器只能对列表中的那些子例程进行内联。在这种情况下，请使用 %auto 子选项启用 -O4 和 -O5 的自动内联。

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

在上面的示例中，用户在禁止编译器可能尝试的例程 zpoint() 的任何可能内联期间，还同时启用了 -O4 的自动内联。

-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 / 输出操作设置 ROUND= 说明符。

如果使用 -iorounding=compatible，数据转换后的值是更接近于两个最接近表示的值或偏离零的值 （如果它是介于两个最接近表示之间的值）。

如果使用 -iorounding=processor-defined，则舍入模式是处理器的默认模式。在未指定 -iorounding 时，这是默认值。

-KPIC

（**已废弃**）等效于 -xcode=pic13。

-KPIC

（**已废弃**）等效于 -xcode=pic32。

-L*path*

将 *path* 增加到在其中搜索库的目录路径的列表。

将 *path* 增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。此选项被传递给链接程序。另请参见第 3-30 页的 "-lx"。

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 在搜索默认目录之前搜索 *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 可能阻止链接未绑定的 libm。默认情况下，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
```

-l*x*

将库 lib*x*.a 增加到链接程序的搜索库列表中。

将 -l*x* 传递给链接程序可以指定其他库供 ld 在其中搜索非分辨的引用。 ld 与对象库 lib*x* 链接。如果共享库 lib*x*.so 是可用的 （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。

-libmil

内联所选的 libm 库例程以进行优化。

某些 libm 库例程有内联模板。此选项选择为当前使用的浮点选项和平台产生速度最快的可执行文件的那些内联模板。

有关详细信息，请参见手册页 libm_single(3F) 和 libm_double(3F)

-loopinfo

显示循环的并行化结果。

显示哪些循环是使用 -parallel、-autopar 或 -explicitpar 选项并行化的，以及哪些循环是没有使用这些选项并行化的。（选项 -loopinfo 必须与其中的一个并行化选项一起使用。）

-loopinfo 显示有关标准错误的消息的列表：

```
demo% f95 -c -fast -parallel -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
"shalow.f", line 193:not parallelized, not profitable
"shalow.f", line 199:PARALLELIZED, and serial version generated
"shalow.f", line 200:not parallelized, not profitable
"shalow.f", line 226:PARALLELIZED, and serial version generated
"shalow.f", line 227:not parallelized, not profitable
...etc
```

-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 95 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。在默认情况下，编译器并不搜索归档文件。

将仅搜索与出现在 USE 语句上的 MODULE 名称同名的 .mod 文件。例如，语句 USE ME 导致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，则默认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。要模拟以前发行版本的行为，请使用：

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

其中 *empty-dir* 是空目录的路径。

-M 和路径之间不能有空格。例如， -M/home/siri/PK15/Modules

有关 Fortran 95 中模块的详细信息，请参见第 4-21 页的 4.9 节 “模块文件”。

-moddir=*path*

指定编译器将写入已编译的 .mod MODULE 文件的位置。

编译器将它编译的 .mod MODULE 信息文件写入由 *path* 指定的目录。目录路径也可以使用 MODDIR 环境变量指定。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 文件的默认目录。

有关 Fortran 95 中模块的详细信息，请参见第 4-21 页的 4.9 节 “模块文件”。

-mp={**%none**|**sun**|**cray**}

选择 Sun 或 Cray 并行化指令。

注 – Sun 和 Cray 并行化指令已过时。请改用 OpenMP 并行化 API。《OpenMP API 用户指南》介绍了如何将应用程序迁移到 OpenMP。

如果未指定 -explicitpar，则默认值是 -mp=%none。

如果指定 -explicitpar，则默认值是 -mp=sun。

-mp=sun	接受 Sun 式样的指令：C$PAR 或 !$PAR 前缀。
-mp=cray	接受 Cray 式样的指令：CMIC$ 或 !MIC$ 前缀。
-mp=%none	忽略所有并行化指令。

还必须指定 -explicitpar（或 -parallel）以启用并行化。为了确保正确无误，还要指定 -stackvar：

```
-explicitpar -stackvar -mp=cray
```

要为 OpenMP 并行化进行编译，请使用 -openmp 标志。请参见第 3-37 页的 "-openmp[={parallel|noopt|none}]"。

在同一编译单元中， Sun 指令和 Cray 指令不能都处于活动状态。

Sun 和 Cray 并行化指令的摘要在本手册的附录 D 中给出。有关详细信息，请参见《Fortran 编程指南》。

-mt

要求链接到线程安全库。

如果您进行自己的低级线程管理 （例如，通过调用 libthread 库），则使用 衙 t 进行编译可防止冲突。

如果将 Fortran 与调用 libthread 库的多线程 C 代码混合在一起，请使用 -mt。另请参见 Solaris 《多线程编程指南》。

在使用 -autopar、 -explicitpar 或 -parallel 选项时，自动隐含 -mt。

请注意以下事项：

- 执行 I/O 的函数子程序本身不应该作为 I/O 语句的一部分被引用。在使用 -mt 时，这样的**递归** I/O 可能使程序出现死锁。
- 通常，**不要**使用 -autopar、-explicitpar 或 -parallel 编译自己的多线程代码。编译器生成的对线程库的调用和程序拥有的调用可能冲突，导致非预期结果。
- 在单处理器系统上，使用 -mt 选项可能会降低性能。

-native

（**已废弃**）优化主机系统的性能。

此选项等效于 -xtarget=native，这是首选设置。 -fast 选项设置 -xtarget=native。

-noautopar

禁用先前命令行中的 -autopar 调用的自动并行化。

-nodepend

(SPARC) 取消任何先前出现在命令行中的 -depend。

-noexplicitpar

禁用先前命令行中的 衔 xplicitpar 调用的显式并行化。

-nofstore

(x86) 取消命令行上的 -fstore。

编译器的默认值是 -fstore。 -fast 包括 -nofstore。

-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，不将任何默认的 衛 *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中，而无须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。此选项提供完全控制。

将 libm（静态地）和 libc（动态地）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。

-nolibmil

取消命令行上的 -libmil。

在 -fast 选项之后使用此选项可禁用内联 libm 数学例程：

```
demo% f95 -fast -nolibmil _
```

-noreduction

禁用命令行上的 衬 eduction。

此选项禁用 -reduction。

-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（该路径将通知运行时链接程序在其中查找所需的共享库）。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。

如果库已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。比较 -R*paths*。

有关详细信息，请参见《Fortran 编程指南》的“库”一章。

-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*]，则仅执行非常基本的优化级别（即仅进行本地公共子表达式消除和停用代码分析）。与不使用优化相比，使用优化级别进行优化可能会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，更高的优化级别可能导致编译时间更长，可执行文件更大。

使用 -g 进行调试不禁止 -O*n*，但是 -O*n* 在某些方面限制 -g；请参见 dbx 文档。

-O3 和 -O4 选项降低调试的效用，以便您无法从 dbx 显示变量，但是您仍然可以使用 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，继续执行原始级别上的后续例程编译。

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

-O

此选项等效于 -O3。

-O1

提供最少的语句级优化。

如果更高的级别导致编译时间过长或超过可用交换空间，则使用此选项。

-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 的使用优先于 -O2 的使用，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。自动增加 -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 选项禁止如上所述的 -O4 自动内联。
-xcrossfile 增大使用 -O4 进行内联的范围。

-O5

尝试主动优化。

仅适合于使用计算时间最大部分的一小段程序。 -O5 的优化算法所用的编译时间较长，在应用于太大的源程序段时也可能降低性能。

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

-o *name*

指定要写入的可执行文件的名称。

在 -o 和 *name* 之间必须有一个空格。如果不使用此选项，则默认为将可执行文件写入 a.out。在与 -c 一起使用时，-o 指定目标 .o 目标文件；在与 -G 一起使用时，它指定目标 .so 库文件。

-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。

-openmp[={parallel|noopt|none}]

启用通过 Fortran 95 OpenMP 版本 2.0 指令进行的显式并行化。

此标志接受以下可选关键字子选项：

`parallel`	• 启用 OpenMP 编译指示的识别，并相应地并行化程序。 • `-xopenmp=parallel` 的最低优化级别是 `-xO3`。如有必要，编译器将优化级别从较低级别更改为 `-xO3`，并发出警告。 • 定义预处理程序标记 `_OPENMP` • 自动调用 `-stackvar`。
`noopt`	• 启用 OpenMP 编译指示的识别，并相应地并行化程序。 • 如果优化级别低于 `-xO3`，则编译器不提升它。如果将优化显式设置为低于 `-xO3` 的级别（如 `-xO2 -openmp=noopt`），则编译器将发出错误。如果没有使用 `-openmp=noopt` 指定优化级别，则识别 OpenMP 编译指示，并相应地并行化程序，但不执行优化。 • 定义预处理程序标记 `_OPENMP` • 自动调用 `-stackvar`。
`none`	禁用 OpenMP 编译指示的识别，并且不更改优化级别。（这是编译器的默认值。）

指定的无子选项关键字 `-openmp` 等效于 `-openmp=parallel`。**请注意，此默认值在以后的发行版本中可能会有所变更。**

要使用 dbx 调试 OpenMP 程序，请使用 `-g -openmp=noopt` 进行编译，以便能够在并行区域内设置断点并显示变量的内容。

OpenMP 指令在 《OpenMP API 用户指南》中进行概述。

要在多线程环境中运行已并行化的程序，必须在执行之前设置 `PARALLEL`（或 `OMP_NUM_THREADS`）环境变量。这通知运行时系统程序可以创建的最大线程数。默认值为 1。一般会将 `PARALLEL` 或 `OMP_NUM_THREADS` 变量设置为目标平台上可用的处理器数。

要启用嵌套并行操作，必须将 `OMP_NESTED` 环境变量设置为 `TRUE`。默认情况下，禁用嵌套并行操作。有关嵌套并行操作的详细信息，请参见 Sun Studio 《 OpenMP API 用户指南》。

OpenMP 要求预处理程序符号 `_OPENMP` 的定义中包含十进制值 YYYYMM，其中 YYYY 和 MM 是实现支持的 OpenMP Fortran API 版本的年份和月份名称。对于 Sun Studio 11 发行版本， OpenMP 2.5 版的值为 200525。

当在单独步骤中编译和链接时，还要在链接步骤中指定 -openmp。当编译包含 OpenMP 指令的库时，这一点尤为重要。

`-PIC`

（**已废弃，SPARC)** 编译与位置无关且具有 32 位地址的代码。

`-PIC` 等效于 `-xcode=pic32`。有关与位置无关的代码的详细信息，请参见第 3-62 页的 ″-xcode=keyword″。

-p

（**已废弃**）为使用 prof 配置程序进行文件配置而编译。

准备目标文件以进行文件配置，请参见 prof (1)。如果在单独步骤中编译和链接，而且使用 -p 选项进行编译，请确保使用 -p 选项进行链接。 -p 与 prof 一起使用主要是为了与旧系统兼容。使用 gprof 的 -pg 文件配置可能是一个更好的备用方案。有关详细信息，请参见 《Fortran 编程指南》的 “性能剖析”一章。

-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本地的且未初始化，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。在其中的任一情况下，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 local 或 common 之一 （或两者）：

本地	在邻近**本地**变量之间增加填充。
common	在通用块中的变量之间增加填充。
%none	不增加填充。（编译器默认值。）

如果同时指定了 local 和 common，则它们可以以任意顺序显示。

-pad 的默认值：

- 默认情况下编译器不进行填充。
- 指定 -pad，但不带值时的指定该项等效于 -pad=local,common。

-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 3-44 页的 "-stackvar"。

程序必须符合以下限制：

- 数组或字符串都不是等效的
- 如果为编译引用通用块的文件而指定了 衜 ad=common，则在编译引用该通用块的所有文件时也必须指定它。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进行编译的，而其他程序单元不是使用该选项进行编译的，则在应该引用通用块内同一位置的情况下可能引用不同位置。
- 如果指定 -pad=common，则不同程序单元中通用块变量的声明必须是相同的 （变量名称除外）。在通用块中的变量之间插入的填充量取决于那些变量的声明。如果变量在不同程序单元中的大小或等级不同，甚至是在同一文件内变量的位置也可能不同。

- 如果指定 -pad=common，则用警告消息标记涉及通用块变量的 EQUIVALENCE 声明，而且不填充该块。
- 如果指定 -pad=common，请避免为通用块中的数组建立过多索引。更改已填充通用块中邻近数据的位置将导致过多索引的建立以不可预知的方式失败。

使用 -pad 时，编程人员应负责始终编译通用块。不同程序单元中出现的没有始终使用 -pad=common 编译的通用块将导致错误。如果同名的通用块在不同的程序单元中具有不同的长度，则将报告使用 -Xlist 进行编译。

-parallel

并行化，使用：-autopar、 -explicitpar、 -depend

注 – 此选项包含 -explicitpar，它可启用传统的 Sun 和 Cray 并行化指令。这些指令现已过时并且不再受支持。优先使用 OpenMP API，它是 Sun Studio 编译器支持的并行化模型。有关将传统的 Sun/Cray 指令迁移到 OpenMP 的详细信息，请参见 Sun Studio《OpenMP API 用户指南》。

对编译器自动选择的循环以及用户提供的指令显式指定的循环进行并行化。如果优化级别较低，则将它自动提升到 -O3。另请参见第 3-17 页的 "-explicitpar"。

为提高性能，在使用任何并行化选项 （包括 -autopar）时，还要指定 -stackvar 选项。

默认情况下， Sun 式样的并行化指令处于启用状态。使用 -mp=cray 可选择 Cray 式样的并行化指令。（注：对于 OpenMP 并行化，请使用 -openmp，不要使用 -parallel。）

如果您进行自己的线程管理，请避免使用 -parallel。请参见第 3-33 页的 "-mt"。

并行化选项（如 -parallel）用于产生将在多处理器系统上运行的可执行程序。在单处理器系统上，并行化通常会降低性能。

要在多线程环境中运行已并行化的程序，必须在执行之前设置 PARALLEL （或 OMP_NUM_THREADS）环境变量。这通知运行时系统程序可以创建的最大线程数。默认值为 1。一般会将 PARALLEL 或 OMP_NUM_THREADS 变量设置为目标平台上可用的处理器数。

如果使用 -parallel，而且在**同一**步骤中进行编译和链接，则链接自动包括多线程库和线程安全的 Fortran 运行时库。如果使用 -parallel，而且在**单独**步骤中进行编译和链接，则还必须使用 -parallel 进行**链接**。

有关进一步的信息，请参见 《Fortran 编程指南》的 “并行化”一章。

-pg

为使用 gprof 配置程序进行文件配置而编译。

以 -p 的方式编译自配置代码，但调用一种运行时记录机制（该机制保存更广泛的统计信息，以及在程序正常终止时产生 gmon.out 文件）。通过运行 gprof 生成执行性能分析数据。有关详细信息，请参见 gprof(1) 手册页和《Fortran 编程指南》。

库选项必须跟在源和 .o 文件**后面**（-pg 库是静态的）。

注 – 如果指定 -pg，则使用 -xprofile 进行编译并无优势。这两种功能没有准备或使用另一种功能提供的数据。

在 64 位 Solaris 平台上使用 prof(1) 或 gprof(1) 或者仅在 32 位 Solaris 平台上使用 gprof 时，所生成的性能分析数据包含近似的用户 CPU 时间。对于在链接可执行文件时指定为链接程序参数的主可执行文件以及共享库中的例程，这些时间是通过 PC 样例数据计算出来的（请参见 pcsample(2)）。没有分析其他共享库（在使用 dlopen(3DL) 启动进程后打开的库）。

在 32 位 Solaris 系统上，使用 prof(1) 生成的性能分析数据仅限于用于可执行文件中的例程。通过将可执行文件与 -pg 相链接并使用 gprof(1)，可以分析 32 位共享库。

Solaris 10 软件不包含使用 -p 编译的系统库。因此，在 Solaris 10 平台上收集的性能分析数据不包含系统库例程的调用计数。

不应使用编译器选项 -p、-pg 或 -xpg 来编译多线程程序，因为这些选项的运行时支持并不能确保线程安全。如果使用这些选项来编译使用多线程的程序，运行时可能会产生无效结果或段故障。

如果在单独步骤中进行编译和链接，而且使用 -pg 进行编译，请确保使用 -pg 进行链接。

-pic

（**已废弃，SPARC**）为共享库编译与位置无关的代码。

-pic 等效于 -xcode=pic13。有关与位置无关的代码的详细信息，请参见第 3-62 页的 "-xcode=keyword"。

-Qoption *pr ls*

在编译阶段 *pr* 传递子选项列表 *ls*。

必须使用空格来分隔 Qoption、*pr* 和 *ls*。Q 可以是大写的或小写的。该列表是一个逗号分隔的子选项列表，其中不包含空格。每个子选项必须适合于该程序阶段；允许使用以负号开头的子选项。

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》中有关链接和库的一章。

-qp

与 -p 等效。

-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) 将动态库搜索路径存储到可执行文件中。

ls 是一个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，将扫描列出路径中的动态库以满足任何非分辨的引用。

使用此选项，用户可以在不使用特殊路径选项以查找所需动态库的情况下运行可装运的可执行文件。

使用 -R*paths* 生成可执行文件，可将目录路径增加到默认路径 /opt/SUNWspro/lib（始终最后搜索它）。

有关详细信息，请参见 《Fortran 编程指南》的 “库”一章以及 Solaris 《链接程序和库指南》。

-r8const

将单精度常数提升为 REAL*8 常数。

所有单精度 REAL 常数都将被提升为 REAL*8 常数。不更改双精度 (REAL*8) 常数。此选项仅适用于常数。要同时提升常数和变量，请参见第 3-89 页的 "-xtypemap=spec"。

请小心使用此选项标志。当使用已提升为 REAL*8 常数的 REAL*4 常数调用需要 REAL*4 参数的子例程或函数时，此选项标志可能导致接口问题。此选项标志还可能导致读取无格式数据文件 （该文件是由无格式写入使用 I/O 列表上的 REAL*4 常数写入的）的程序出现问题。

-reduction

识别循环中的约简操作。

在自动并行化期间分析循环中的约简操作。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性的准则，但是编译器可以识别它们并在指定 -reduction 时作为特殊情况对它们进行并行化。有关编译器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 或 -parallel 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。

示例：使用**约简**自动进行并行化：

```
demo% f95 -parallel -reduction any.f
```

-S

编译并仅生成汇编代码。

编译指定的程序，并在后缀为 .s 的对应文件上保留汇编语言输出。不创建 .o 文件。

-s

将符号表与可执行文件分离。

此选项使可执行文件更小，并使实施逆向工程更困难。但是，此选项禁止使用 dbx 或其他工具进行调试，而且覆盖 -g。

-sb

（**已废弃**）为源代码浏览器产生表信息。

注 – -sb 不能在编译器通过 fpp 或 cpp 预处理程序自动传递的源文件 （即扩展名为 .F、 .F90、 .F95 或 .F03 的文件）上使用，也不能与 -F 选项一起使用。

-sbfast

（**已废弃**）**仅**生成源代码浏览器表。

仅为源代码浏览器生成表信息。不汇编、链接或生成目标文件。

注 – -sbfast 不能在编译器通过 fpp 或 cpp 预处理程序自动传递的源文件（即扩展名为 .F、 .F90、 .F95 或 .F03 的文件）上使用，也不能与 -F 选项一起使用。

-silent

（**已废弃**）禁止编译器消息。

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。

-stackvar

尽可能在栈上分配局部变量。

此选项使编写递归代码和重入代码更容易，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。

局部变量是除伪参数、 COMMON 变量、从外部作用域继承的变量或 USE 语句使其可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。默认情况下，未显式初始化但是其某些组件已初始化的结构变量不是使用 SAVE 显式声明的。此外，等效于具有 SAVE 或 STATIC 属性的变量的变量隐式具有 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初始值。在栈上分配的变量不是隐式初始化的 （在默认情况下进行初始化的结构变量组件除外）。

使用 -stackvar 将大数组放置在栈上可以使栈溢出，导致段故障。可能需要增加栈大小。

执行程序的初始线程具有一个**主**栈，而多线程程序的每个助手线程都具有自己的**线程**栈。

对于主栈，默认栈大小大约是 8 兆字节；对于每个线程栈，默认栈大小是 4 兆字节 （在 SPARC V9 平台上是 8 兆字节）。 limit 命令 （不带参数）显示当前主栈大小。如果使用 -stackvar 时出现段故障，请尝试增加主栈和线程栈的大小。

示例：显示当前**主**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**栈大小设置为 64 兆字节：

```
demo% limit stacksize 65536
```

示例：将每个**线程**栈大小设置为 8 兆字节：

```
demo% setenv STACKSIZE 8192
```

通过为 STACKSIZE 环境变量指定值（以千字节为单位），可以设置每个从属线程使用的栈大小：

```
% setenv STACKSIZE 8192
```

将每个从属线程的栈大小设置为 8 兆字节。

STACKSIZE 环境变量也接受带有 B、K、M 或 G 后缀的数值，它们分别表示字节、千字节、兆字节或千兆字节。默认值为千字节。

有关与并行化一起使用 -stackvar 的进一步信息，请参见《Fortran 编程指南》的“并行化”一章。有关 limit 命令的详细信息，请参见 csh(1)。

使用 -xcheck=stkovf 进行编译可以启用栈溢出状态的运行时检查。请参见第 3-59 页的 "-xcheck=keyword"。

-stop_status[={yes|no}]

允许 STOP 语句返回整数状态值。

默认值为 -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，该值将被传递到环境：

```
STOP 123
```

该值必须在 0 到 255 的范围内。超出此范围的值将被截断并发出运行时消息。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）或 $? （对于 Bourne shell sh 和 Korn shell ksh）。

-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。

-time

每个编译阶段所用时间。

将显示在每个编译器遍中所用时间和资源。

-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。默认情况下，将大写字母视为小写字母（字符串常量中的除外）。如果使用此选项，则编译器将 `Delta`、`DELTA` 和 `delta` 视为不同的符号。

可移植性以及将 Fortran 与其他语言混合可能要求使用 鑾。请参见《Fortran 编程指南》中有关将程序移植到 Fortran 95 的一章。

-U*name*

未定义预处理程序宏 *name*。

此选项仅应用于调用 `fpp` 或 `cpp` 预处理程序的源文件。它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。可以在命令行上出现多个 `-U`*name* 标志。在 `-U` 和宏 *name* 之间不得有空格。

-u

报告未声明的变量。

使所有变量的默认类型都是**未声明的**，而不是使用 Fortran 隐式类型处理，就像在每个编译单元中出现 `IMPLICIT NONE`。此选项警告未声明的变量，并且不覆盖任何 `IMPLICIT` 语句或显式 *type* 语句。

-unroll=*n*

启用 DO 循环的解开（如果可能）。

n 是正整数。选项有：

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

通常，循环解开可提高性能，但将增加可执行文件的大小。有关此编译器和其他编译器优化的详细信息，请参见《Fortran 编程指南》中的“性能与优化”一章。另请参见第 2-10 页的 2.3.1.3 节 “`UNROLL` 指令”。

-use=*list*

指定隐式 `USE` 模块。

list 是一个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* 语句增加到正编译的每个子程序或模块。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增加 USE *module_name*。

有关 Fortran 95 中模块的详细信息，请参见第 4-21 页的 4.9 节 “模块文件”。

-V

显示每个编译器传递的名称和版本。

此选项在编译器执行时打印每个传递的名称和版本。

在与 Sun 服务工程师讨论问题时，此信息可能很有用。

-v

冗余模式 - 显示每个编译器传递的详细信息。

与 -V 一样，此选项在编译器执行时显示每个传递的名称，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。

-vax=*keywords*

指定启用 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：这可能导致数据无法对齐。
%all	启用所有这些 VAX VMS 功能。
%none	禁用所有这些 VAX VMS 功能。

通过在子选项前面加上 no%，可以单独选择或关闭该子选项。

示例：

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

-vpara

显示冗余并行化消息。

在编译器分析用指令显式标记为并行化的循环时，它发出有关所检测到的某些数据相关性的警告消息；但是仍将对循环进行并行化。

示例：冗余并行化警告：

```
demo% f95 -explicitpar -vpara any.f
any.f:
 MAIN any:
"any.f", line 11:Warning:the loop may have parallelization
inhibiting reference
```

与 -xopenmp 和 OpenMP API 指令结合使用，或者与 -explicitpar 和 C$MIC DOALL 传统并行化指令结合使用。

使用编译器发出的警告可检测到以下情况：

- 错误地使用 OpenMP 数据共享属性子句，如声明了一个共享变量，并且其在 OpenMP 并行区域中的访问可能导致数据争用；或者声明了一个私有变量，并且在并行区域后面使用其在该并行区域中的值。
- C$MIC DOALL 对在不同循环迭代之间具有数据相关性的循环进行并行化。

如果处理所有并行化指令时没有出现问题，则不会显示任何警告。

注 – Sun Studio 编译器支持 OpenMP API 并行化模型。因此，C$MIC 并行化指令已过时。有关迁移到 OpenMP API 的信息，请参见 《OpenMP API 用户指南》。

-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前面选项的全部或部分，您确实会收到一个警告。

n 可以是 0、 1、 2、 3 或 4。

-w0 仅显示错误消息。它等价于 -w
-w1 显示错误和警告。在不使用 -w 的情况下，这是默认值。
-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。

示例：极 仍然允许显示某些警告：

```
demo% f95 -w -parallel any.f
f95:Warning: 优化器级别已从 0 改为 3 以支持并行化代码
demo%
```

-Xlist[*x*]

生成列表并进行全局程序检查 (GPC)。

使用此选项可查找潜在的编程错误。它调用额外的编译器传递，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 – 确保在使用 -Xlist 进行编译之前，更正了源代码中的所有语法错误。在具有语法错误的源代码上运行时，可能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 （默认）
- 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 （默认）

默认情况下，将列表写入文件 name.lst，其中 name 取自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进一步的灵活性。它们是由 -Xlist 主选项的后缀指定的，如下表所示

表 3-9 -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，但不显示目标文件
-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

表 3-9 -Xlist 子选项 （续）

选项	特性
-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） - 默认值是 2
-Xlistw*[nnn]*	将输出行的宽度设置为 *nnn* 列 - 默认值是 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。

-x386

(**x86**) 等效于 -xtarget=386

-x486

(**x86**) 等效于 -xtarget=486

-xa

等效于 -a。

-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“建立过多索引”、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的情况，使生成的代码无法按预期结果运行。

使用 -xalias 标志可通知编译器程序偏离 Fortran 标准的别名要求的程度。

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 3-10 -xalias 选项关键字

关键字	含义
dummy	子程序的伪 （形式）参数可以互为别名，也可以作为全局变量的别名。
no%dummy	（默认值）。伪参数的使用遵循 Fortran 标准，它们不能互为别名，也不作为全局变量的别名。
craypointer	（默认值）。 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。
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	（默认值）传递参数不会导致进一步命名别名。
索引越界	• 对 COMMON 块中元素的引用可能引用 COMMON 块或等价组中的任何元素。 • 将 COMMON 块或等价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 块或等价组中的任何元素。 • 将序列派生类型的变量视为 COMMON 块，此类变量的元素可能作为该变量其他元素的别名。 • 可能违反了单个数组边界，但除上文所述外，假定引用的数组元素仍然在数组内。 没有考虑数组语法、WHERE 和 FORALL 语句以便建立过多索引。如果索引越界出现在这些构造中，应将它们改写为 DO 循环。
no%overindex	（默认值）没有违反数组边界。数组引用不引用其他变量。
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	（默认值） 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的性能，不带列表的 -xalias 等价于：

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

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

如果未指定 -xalias 标志，则默认情况下编译器假定程序符合 Fortran 95 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》中的 “移植”一章。

-xarch=*isa*

指定指令集架构 (ISA)。

-xarch 关键字 *isa* 接受的架构如表 3-11 所示：

表 3-11 -xarch ISA 关键字

平台	有效的 -xarch 关键字
SPARC	generic、generic64、native、native64、v7、v8a、v8、v8plus、v8plusa、 v8plusb、 v9、 v9a、 v9b
x86	generic、 native、 386、 pentium_pro、 sse、 sse2、 amd64

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

```
% f95 -xtarget=ultra2 -xarch=v8plusb ...
```

覆盖由 -xtarget=ultra2 设置的 -xarch=v8

通过只允许指定的指令集，此选项使编译器生成的代码只包含指定指令集架构的指令。该选项不保证使用任何目标 - 特定的指令。

如果此选项与优化一起使用，则适当的选择可以在指定的架构上为可执行文件提供良好性能。由不适当的选择而产生的二进制程序在预定目标平台上是不可执行的。

表 3-12 汇总了 SPARC 平台上最常用的 -xarch 选项：

表 3-12 SPARC 平台上最常用的 -xarch 选项

-xarch=	性能
generic	• 在所有支持的平台上运行都可以获得足够的性能
v8plusa	• 在 UltraSPARC-II 处理器上以 32 位模式运行可以获得最佳性能
v8plusb	• 在 UltraSPARC-III 处理器上以 32 位模式运行可以获得最佳性能 • 不能在其他平台上执行
v9a	• 在 UltraSPARC-II 处理器上以 64 位模式运行可以获得最佳性能 • 不能在其他平台上执行
v9b	• 在 UltraSPARC-III 处理器上以 64 位模式运行可以获得最佳性能 • 不能在其他平台上执行

另请注意：

- SPARC 指令集架构 V7、 V8 和 V8a 全部是二进制兼容的。
- 由 `v8plus` 和 `v8plusa` 编译的对象二进制文件 (`.o`) 可以同时链接和执行，但必须在与 SPARC V8plusa 兼容的平台上运行。
- 由 `v8plus`、 `v8plusa` 和 `v8plusb` 编译的对象二进制文件 (`.o`) 可以同时链接和执行，但必须在与 SPARC V8plusb 兼容的平台上运行。
- `-xarch` 值 `v9`、 `v9a` 和 `v9b` 只能在 UltraSPARC 64 位 Solaris 环境中使用。
- 使用 `v9` 和 `v9a` 编译的对象二进制文件 (`.o`) 可以进行链接并可一起执行，但是将只能在 SPARC V9a 兼容平台上运行。
- 使用 `v9`、`v9a` 和 `v9b` 编译的对象二进制文件 (`.o`) 可以进行链接并可一起执行，但是将只能在 SPARC V9b 兼容平台上运行。

对于任何特定选择，生成的可执行文件在早期架构中可能运行更缓慢。此外，虽然在多数指令集架构中都可以使用四精度 （`REAL*16` 和 `long double`）浮点指令，但编译器不在它生成的代码中使用这些指令。

在 SPARC 平台上，未指定 `-xarch` 时的默认值为 `v8plus` ； 而在 x86 平台上则为 `386`。

表 3-13 提供有关 SPARC 平台上每个 `-xarch` 关键字的详细信息。

表 3-13 SPARC 平台的 `-xarch` 值

`-xarch=`	**含义 (SPARC)**
`generic`	**为了在大多数 32 位系统上获得良好性能而进行编译。** 这是默认行为。该选项在多数处理器上使用高性能的最佳指令集，同时不降低处理器的主要性能。对于每个新的发行版本，“最佳”指令集的定义可能会有所调整 （如果适当），当前解释为 `v8plus`。
`generic64`	**为了在大多数启用 64 位的系统上获得良好性能而进行编译。** 此选项使用最佳指令集以便在大多数启用 64 位的处理器上获得良好性能，而不使其中任一处理器上的性能有较大幅度的降低。对于每个新的发行版本，“最佳”指令集的定义可能会有所调整 （如果适当），当前解释为 `v9`。
`native`	**为了在此系统上获得良好性能而进行编译。** 这是 `-fast` 选项的默认值。编译器为运行它的当前系统处理器选择适当的设置。
`native64`	**为了在此系统上获得 64 位模式下的良好性能而进行编译。** 与 `native` 一样，编译器在运行它的当前系统处理器上为 64 位模式选择适当的设置。
`v7`	**编译用于 SPARC-V7 ISA。** 在 V7 ISA 上使编译器生成高性能代码。 这等价于在 V8 ISA 上使用高性能的最佳指令，但不包括整数 `mul` 和 `div` 指令和 `fsmuld` 指令。 示例：SPARCstation 1, SPARCstation 2

表 3-13 SPARC 平台的 -xarch 值 （续）

-xarch=	含义 (SPARC)
v8a	**为 SPARC-V8 ISA 的 V8a 版本进行编译** 按照定义， V8a 是指不包含 fsmuld 指令的 V8 ISA。 该选项在 V8a ISA 上使编译器生成高性能代码。 示例：基于 microSPARC I 芯片架构的任何系统
v8	**为 SPARC-V8 ISA 进行编译**。 使编译器能够生成用于在 V8 架构上获得良好性能的代码。 示例：SPARCstation 10
v8plus	**编译用于 SPARC-V9 ISA 的 V8plus 版本**。 根据定义，V8plus 意味着 V9 ISA，但只限于由 V8plus ISA 规范所定义的 32 位子集，而不包括可视化指令集 (VIS) 和特定实现的 ISA 扩展。 • 该选项在 V8plus ISA 上使编译器生成高性能代码。 • 产生的对象代码是 SPARC-V8+ ELF32 格式且只在 Solaris UltraSPARC 环境下执行 （不能在 V7 或 V8 处理器上运行）。 示例：基于 UltraSPARC 芯片架构的任何系统
v8plusa	**编译用于 SPARC-V9 ISA 的 V8plusa 版本**。 根据定义， V8plusa 意味着 V8plus 架构加可视化指令集 (VIS) 版本 1.0 和 UltraSPARC 扩展。 • 该选项使编译器在 UltraSPARC 架构上生成高性能代码，但只限于 V8plus 规范定义的 32 位子集。 • 产生的对象代码是 SPARC-V8+ ELF32 格式且只在 Solaris UltraSPARC 环境下执行 （不能在 V7 或 V8 处理器上运行）。 示例：基于 UltraSPARC 芯片架构的任何系统
v8plusb	**为 SPARC-V8plus ISA 的 V8plusb 版本 （带有 UltraSPARC-III 扩展）进行编译**。 使编译器能够生成用于带有可视指令集 (VIS) 2.0 版和 UltraSPARC-III 扩展的 UltraSPARC 架构的对象代码。 • 生成的对象代码是 SPARC-V8+ ELF32 格式的，只能在 Solaris UltraSPARC-III 环境中执行。 • 如果使用此选项进行编译，则将使用用于在 UltraSPARC-III 架构上获得良好性能的最佳指令集。

表 3-13 SPARC 平台的 -xarch 值 （续）

-xarch=	含义 (SPARC)
v9	**为 SPARC-V9 ISA 进行编译**。 使编译器能够生成用于在 V9 SPARC 架构上获得良好性能的代码。 • 产生的 .o 对象文件是 ELF64 格式且只能与相同格式的其他 SPARC-V9 对象文件链接。 • 产生的可执行文件只能在 UltraSPARC 处理器上运行，该处理器运行具有 64 位内核的 64 位 Solaris 操作环境。 • 在启用 64 位的 Solaris 环境下编译时，只有 -xarch=v9 可用。
v9a	**编译用于具有 UltraSPARC 扩展的 SPARC-V9 ISA**。 将可视化指令集 (VIS) 和 UltraSPARC 处理器的特定扩展增加到 SPARC-V9 ISA，并使编译器在 V9 SPARC 架构上生成高性能代码。 • 产生的 .o 对象文件是 ELF64 格式且只能与相同格式的其他 SPARC-V9 对象文件链接。 • 产生的可执行文件只能在 UltraSPARC 处理器上运行，该处理器运行具有 64 位内核的 64 位 Solaris 操作环境。 • 在启用 64 位的 Solaris 环境下编译时，只有 -xarch=v9a 可用。
v9b	**为带有 UltraSPARC-III 扩展的 SPARC-V9 ISA 进行编译** 将 UltraSPARC-III 扩展和 VIS 2.0 版增加到 SPARC-V9 ISA 的 V9a 版本。如果使用此选项进行编译，则将使用用于在 Solaris UltraSPARC-III 环境中获得良好性能的最佳指令集。 • 产生的对象代码是 SPARC-V9 ELF64 格式且只能与相同格式的其他 SPARC-V9 对象文件链接。 • 只能在运行启用 64 位且具有 64 位内核的 Solaris 操作环境的 UltraSPARC-III 处理器上运行生成的可执行文件。 • 在启用 64 位的 Solaris 环境下编译时，只有 -xarch=v9b 可用。

表 3-14 x86 平台上的每个 -xarch 关键字的详细信息。在 x86 平台上，如果未指定 -xarch，则默认值为 generic。

表 3-14 x86 平台的 -xarch 值

-xarch=	含义 (x86)
generic	为了在大多数 32 位 x86 平台上获得良好性能而进行编译。这是默认值，等价于 -xarch=386。
generic64	为了在大多数 64 位 x86 平台上获得良好性能而进行编译。当前将其解释为 amd64。
native	为了在此 x86 体系结构上获得良好性能而进行编译。为了在大多数 x86 处理器上获得良好性能使用最佳指令集。如果需要，在新的发行版本中会调整 “最佳”指令集的定义。
native64	为了在此 64 位 x86 体系结构上获得良好性能而进行编译。

表 3-14 x86 平台的 -xarch 值 （续）

-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。（请参见下面的注释。）
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。（请参见下面的注释。）
sse2a	将 AMD 扩展 （3DNow!、 3DNow! 扩展和 MMX 扩展）添加到 32 位 SSE2 体系结构中。
amd64	为 AMD64 64 位 x86 指令集进行编译。
amd64a	将 AMD 扩展（3DNow!、3DNow! 扩展和 MMX 扩展）添加到 AMD64 体系结构中，并生成 64 位 ELF 格式二进制文件。

x86/x64 平台的特别注意事项：

- 使用 -xarch (设置为sse 或sse2) 而为兼容 Solaris x86 SSE/SSE2 Pentium 4 平台编译的程序只能在支持 SSE/SSE2 的平台上运行。
- 使用 -xarch (设置为 ssea、sse2a 或 pentium_proa) 而编译的程序必须在支持 AMD 3DNow!、 3DNow! 扩展以及 SSE/SSE2 的平台上运行。
- 从 Solaris 9 4/04 开始的操作系统发行版本在兼容 Pentium 4 平台上支持 SSE/SSE2。更早版本的 Solaris 操作系统不支持 SSE/SSE2。
- 类似地，使用 -xarch=amd64 为 Solaris x86 AMD64 平台编译的程序必须在支持 AMD 64 位体系结构的平台上运行。请注意， AMD64 体系结构支持 SSE/SSE2。
- 使用 -xarch=amd64a 编译的程序必须在支持 AMD 64 位体系结构以及 AMD 3DNOW! 和 AMD 3DNow! 扩展的平台上运行。
- 从 Sun Studio 11 和 Solaris 10 操作系统开始，对使用这些专用的 -xarch 硬件标志编译和生成的程序二进制文件进行验证，看其是否在相应的平台上运行。
- 在 Solaris 10 以前的系统上，不执行任何验证，而是由用户来负责确保将使用这些标志生成的对象部署在相应的硬件上。
- 如果在没有使用相应功能或指令集扩展来启用的平台上运行使用这些 -xarch 选项编译的程序，则可能会导致发生段故障或错误结果，并且不会显示任何显式警告消息。
- 这一警告还会扩展到采用 .il 内联汇编语言函数或 __asm() 汇编程序代码 （采用 SSE、 SSE2、 AMD 64 和 AMD 3DNow! 指令以及 AMD 3DNow! 扩展）的程序。
- 如果单独进行编译和链接，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。要最大限度地减少这些差异，请使用 `-fstore` 选项，或者使用 `-xarch=sse2` 进行编译 （如果硬件支持 SSE2 的话）。
- 如果在 x86 平台上使用 `-xarch=generic64` 或 `-xarch=amd64` 编译或链接程序的某一部分，则也必须使用这些选项之一来编译程序的所有部分。

-xassume_control[=*keywords*]

设置参数以控制 ASSUME 编译指示。

使用此标志可控制编译器处理源代码中 ASSUME 编译指示的方式。

ASSUME 编译指示为编程人员提供了一种断言特殊信息 （编译器使用这些特殊信息可实现较佳的优化）的方法。可以使用可能值限定这些断言。将可能值为 0 或 1 的断言标记为 “确定”，否则视为不确定。

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

有关 `f95` 编译器识别的 ASSUME 编译指示的描述，请参见第 2-12 页的 2.3.1.9 节 “ASSUME 指令”。

`-xassume_control` 选项上的 *keywords* 可以是单个子选项关键字，也可以是逗号分隔的关键字列表。识别的关键字子选项如下：

`optimize`	在 ASSUME 编译指示上作出的断言影响程序的优化。
`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 的正实常量。默认值是 “.1”。`retrospective` 编译代码以确定所有断言的真假。容差值 *d* 之外的那些值在程序终止时的输出结果中列出。
`%none`	忽略所有 ASSUME 编译指示。

编译器的默认值是

```
-xassume_control=optimize
```

这意味着，编译器识别 ASSUME 编译指示，而且后者将影响优化，但是不进行检查。

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 编译指示，但是后者不影响优化。无效的断言将导致程序终止。

-xautopar

等效于 -autopar。

-xbinopt={prepare | off}

(SPARC) 为编译后优化准备二进制文件。

binopt(1) 启用编译的二进制文件以随后进行优化、转换和分析。可以在生成可执行文件或共享对象时使用该选项，它必须用于 -O1 或更高的优化级别才有效。

在使用此选项进行生成时，二进制文件大小大约会增加 5%。

如果在单独的步骤中进行编译和链接， -xbinopt 必须同时出现在编译和链接步骤中。

如果使用 -xbinopt 来编译应用程序的部分源代码，则 -xbinopt 标志仍然会出现在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
```

binopt(1) 只能优化使用 -xbinopt 编译的代码。

默认值为 -xbinopt=off。

-xcache=*c*

为优化器定义高速缓存属性。

c 必须是以下值之一：

- generic
- native
- *s1/l1/a1[/t1]*
- *s1/l1/a1[/t1]:s2/l2/a2[/t2]*
- *s1/l1/a1[/t1]:s2/l2/a2[/t2]:s3/l3/a3[/t3]*

si/*li*/*ai*/*ti* 的定义如下：

si *i* 级数据高速缓存的大小 （千字节）
li *i* 级数据高速缓存的行大小 （字节）
ai *i 级数据高速缓存的关联性*
ti 共享 *i* 级高速缓存的硬件线程数 （**可选**）。

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。

虽然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的 -xtarget 选项所隐含的 -xcache 值。

表 3-15 -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，而且不使性能有较大幅度的降低。这是默认设置。
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

示例：-xcache=16/32/4:1024/32/1 指定了：

一个 1 级高速缓存具有以下属性：16K 字节、 32 字节行大小、 4 向结合性。

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

-xcg89

（**SPARC/ 已废弃**）等效于 -cg89。

-xcg92

（**SPARC/ 已废弃**）等效于 -cg92。

-xcheck=*keyword*

生成特殊的运行时检查和初始化。

keyword 必须是以下项之一：

关键字	特性
stkovf	打开用于检测子程序入口上是否有栈溢出的运行时检查。如果检测到栈溢出，则将引发 SIGSEGV 段故障。（仅限 **SPARC**）
no%stkovf	禁用用于检测栈溢出的运行时检查。（仅限 **SPARC**）
init_local	执行局部变量的特殊初始化。 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。 不对模块变量、 SAVE 变量和 COMMON 块中的变量进行初始化。
no%init_local	禁用局部变量初始化。这是默认设置。
%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。

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

-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。

虽然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的 -xtarget 选项所隐含的 -xchip 值。

-xchip=*c* 的一些作用是：

- 指令调度
- 编译分支的方式
- 在语义等价的两个选择项之间作出选择

以下的两个表列出有效的 -xchip 处理器名称值：

表 3-16 常用 -xchip SPARC 处理器的名称

-xchip=	为以下项目进行优化
generic	大多数 SPARC 处理器。（这是默认情况。）
native	此主机平台。
ultra	UltraSPARC 处理器。

表 3-16 常用 -xchip SPARC 处理器的名称

-xchip=	为以下项目进行优化
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
ultra3iplus	UltraSPARC IIIi+ 处理器
ultra4	UltraSPARC IV 处理器
ultra4plus	UltraSPARC IV+ 处理器
ultraT1	UltraSPARC T1 处理器

下面是较旧且较不常用的 -xchip 处理器名称，在此处列出仅供参考：

表 3-17 不常用的 -xchip SPARC 处理器的名称

-xchip=	为以下项目进行优化
old	pre-SuperSPARC 处理器。
super	SuperSPARC 处理器。
super2	SuperSPARC II 处理器。
micro	MicroSPARC 处理器。
micro2	MicroSPARC II 处理器。
hyper	HyperSPARC 处理器。
hyper2	HyperSPARC II 处理器。
powerup	Weitek PowerUp 处理器。

在 x86 平台上：-xchip 值为 386、 486、 pentium、 pentium_pro、 pentium3、 pentium4、 opteron、 generic 和 native。

-xcode=*keyword*

(SPARC) 指定 SPARC 平台上的代码地址空间。

keyword 的值有：

关键字	特性
`abs32`	生成 32 位绝对地址。代码 + 数据 +bss 的大小不应超过 2**32 字节。下面是 32 位平台上的默认值：`-xarch=generic, v7, v8, v8a, v8plus, v8plusa`
`abs44`	生成 44 位绝对地址。代码 + 数据 +bss 的大小不应超过 2**44 字节。仅在 64 位平台上可用：`-xarch=v9, v9a`
`abs64`	生成 64 位绝对地址。仅在 64 位平台上可用：`-xarch=v9, v9a`
`pic13`	生成与位置无关的代码（小模型）。等价于 `-pic`。在 32 位平台上允许至多引用 2**11 个唯一外部符号；在 64 位平台上为 2**10 个。
`pic32`	生成与位置无关的代码（大模型）。等价于 `-PIC`。在 32 位平台上允许至多引用 2**30 个唯一外部符号；在 64 位平台上为 2**29 个。

默认值（如果不显式指定 `-xcode=`*keyword*）是：

`-xcode=abs32` 在 SPARC V8 和 V7 平台上。
`-xcode=abs44` 在 UltraSPARC V9（`-xarch=v9` 平台上）

与位置无关的代码：

在创建动态共享库以提高运行时性能时，可使用 `-xcode=pic13` 或 `-xcode=pic32`。

尽管动态可执行文件内的代码通常绑定到内存中的固定地址，但是可以将与位置无关的代码装入进程地址空间中的任意位置。

在使用与位置无关的代码时，将生成可重定位引用，作为通过全局偏移表的间接引用。在使用 `-xcode=pic13` 或 `-xcode=pic32` 进行编译时，由于不要求大量的重定位（这些重定位是与位置有关的代码所强制的），从而使共享对象中的频繁访问项获益。

全局偏移表的大小不应超过 8Kb。

`-xcode={pic13|pic32}` 有两个名义性能成本：

- 使用 `-xcode=pic13` 或 `-xcode=pic32` 编译的例程在入口处执行几个额外指令，以便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全局偏移表。
- 每次对全局变量或静态变量的访问都会涉及通过全局偏移表的额外间接内存引用。如果编译是使用 `pic32` 进行的，则每个全局和静态内存引用都有两个额外指令。

在考虑上述成本时，请记住：由于受到库代码共享的影响，使用 -xcode=pic13 或 -xcode=pic32 可以大大减少系统内存需求。共享库中已使用 -xcode=pic13 或 -xcode=pic32 编译的每个代码页都可以由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_
```

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

要确定使用 -xcode=pic13 还是 -xcode=pic32，请通过使用 elfdump -c 查看全局偏移表 (GOT) 的大小（有关详细信息，请参见 elfdump(1) 手册页）以及段标题 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，否则指定 -xcode=pic32。

通常，确定应该如何使用 -xcode 时应遵循以下准则：

- 如果是生成可执行文件，则不应该使用 -xcode=pic13 或 -xcode=pic32。
- 如果是生成仅用于链接到可执行文件的归档库，则不应该使用 -xcode=pic13 或 -xcode=pic32。
- 如果是生成共享库，请先使用 -xcode=pic13，一旦 GOT 大小超过了 8,192 字节，则使用 -xcode=pic32。
- 如果是生成用于链接到共享库的归档库，则只应该使用 -xcode=pic32。

在生成动态库时，建议使用 -xcode=pic13 或 pic32 （或者 -pic 或 -PIC）选项进行编译。请参见 Solaris 《链接程序和库指南》。

-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。）

默认值是 -xcommonchk=no；用于检测通用块不一致性的运行时检查已禁用，因为它会导致性能的降低。应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 等效于 -xcommonchk=yes。

示例：在 tc.f 中缺少 TASKCOMMON 指令

```
demo% cat tc.f
                common /x/y(1000)
                do 1 i=1,1000
 1              y(i) = 1.
                call z(57.)
                end
demo% cat tz.f
                subroutine z(c)
                common /x/h(1000)
C$PAR TASKCOMMON X
C$PAR DOALL
                do 1 i=1,1000
1                       h(i) = c* h(i)
                return
                end
demo% f95 -c -O4 -parallel -xcommonchk tc.f
demo% f95 -c -O4 -parallel -xcommonchk tz.f
demo% f95 -o tc -O4 -parallel -xcommonchk tc.o tz.o
demo% tc
ERROR(libmtsk):inconsistent declaration of
threadprivate/taskcommon
    x_: not declared as threadprivate/taskcommon at line 1 of tc.f
demo%
```

-xcrossfile[={1|0}]

启用跨源文件的优化和内联处理。

通常，在命令行上编译器的分析范围限制到每个独立的文件。例如， -O4 的自动内联处理仅限于同一源文件中定义和引用的子程序。

编译器使用 -xcrossfile 可以分析在命令行上命名的所有文件，结果就好像这些文件被连接到单一的源文件中。

仅当与 -O4 或 -O5 一起使用时， -xcrossfile 才有效。

交叉文件内联创建可能但通常不存在的的源文件相互依赖性。如果文件集中任何使用 -xcrossfile 编译的文件发生了更改，则必须重新编译所有文件，以确保正确地内联新代码。请参见第 3-29 页的 "-inline=[%auto][[,][no%]f1,_[no%]fn]"。

在命令行上不带 -xcrossfile 的默认值为 -xcrossfile=0，并且不执行交叉文件优化。要启用交叉文件优化，请指定 -xcrossfile （等效于 -xcrossfile=1）。

编译中的任何 .s 汇编程序源文件不参与交叉文件分析。同时，使用 -S 编译时 -xcrossfile 标记被忽略。

-xdebugformat={stabs|dwarf}

Sun Studio 编译器将把调试器信息的格式从“stabs”格式迁移为“dwarf”格式。在此发行版本中，默认设置为 -xdebugformat=stabs。

如果要维护读取调试信息的软件，您可以使用此选项将工具从 stabs 格式转换为 dwarf 格式。

使用此选项可作为一种为移植工具而存取新格式的方法。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，否则没有必要使用此选项。

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

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

如果您未指定 -xdebugformat，则编译器假定 -xdebugformat=stabs。指定此选项而不带参数是一个错误。

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 仍有影响。

dbx 和性能分析软件可识别 stabs 和 dwarf 格式，因此使用此选项对任何工具的功能性并无影响。

这是过渡性接口，因此发行版本之间会发生更改而不兼容，即使在发行版本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不断改进。

-xdepend

等效于 -depend。

-xexplicitpar

等效于 -explicitpar。

-xF

允许性能分析器在函数级别重组。

允许使用编译器、性能分析器和链接程序重组核心图像中的函数（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，用于优化函数在内存中的排序（具体取决于共同使用函数的方式）。可以通过使用链接程序 -Mmapfile 选项，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放置到单独的部分中。

仅当应用程序文本缺页时间消耗了应用程序的大部分时间时，重组内存中的子程序才有用。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《程序性能分析工具》手册。

-xfilebyteorder=*options*

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

该标志标识未格式化 I/O 文件数据的字节顺序和字节对齐。*options* 必须指定以下任何组合，但必须至少出现一个规范：

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。允许值为 1、2、4、8 和 16。对齐适用于 Fortran VAX 结构和 Fortran 95 派生类型，它们使用依赖于平台的对齐以便与 C 语言结构保持兼容。

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 SPARCV8（32 位）文件；而 `big16` 描述 SPARC V9 （64 位）文件。

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

平台	“本机” 对应于：
32 位 SPARC V8	`big8`
64 位 SPARC V9	`big16`
32 位 x86	`little4`
64 位 x86 (`amd64`)	`little16`

spec 必须是包含以下内容的以逗号分隔的列表：

%all

unit

filename

%all 指所有文件和逻辑单元 （以 `"SCRATCH"` 打开的或在 `-xfilebyteorder` 标志中其他地方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。

示例：

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

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，则第一个默认设置为 -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。

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，将会使用相应的填充，即使没有进行字节交换。例如，如果使用 -xarch=amd64 为 64 位 x86 平台进行编译并指定了 -xfilebyteorder=little4:*filename*，就会出现这种情况。

big-endian 和 little-endian 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，因为默认双精度数据类型具有不同的大小。

请注意，在 x86 平台上不能使用包含 REAL*16 数据的无格式文件，这些平台不支持 REAL*16。

如果使用全部 UNION/MAP 数据对象对指定为非本机的文件执行 I/O 操作，则会出现运行时 I/O 错误。只能使用 MAP 的各别成员（而非包含 UNION/MAP 的全部 VAX 记录）对非本机文件执行 I/O 操作。

-xhasc[={yes|no}]

将霍尔瑞斯常量作为实际参数列表中的字符串。

如果 -xhasc=yes，则霍尔瑞斯常量在子例程或函数调用上作为实际参数出现时，编译器将霍尔瑞斯常量视为字符串。这是默认设置，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 -xhasc=no，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上不生成字符串长度。）

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

示例：

```
demo% cat hasc.f
                call z(4habcd, 慳bcdefgí)
                             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。

-xhelp={readme|flags}

显示摘要帮助信息

-xhelp=readme 显示此发行版的编译器的联机自述文件。
-xhelp=flags 列出编译器选项标记。等价于 -help。

-xia[={widestneed|strict}]

(SPARC) 启用区间运算扩展并设置合适的浮点环境。

未指定时的默认值为 -xia=widestneed。

《区间运算编程指南》中详细说明了区间运算的 Fortran 95 扩展。另请参见第 3-69 页的 "-xinterval[={widestneed|strict|no}]"。

-xia 标记是一个宏，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

-xinline=*list*

等效于 -inline。

-xinterval[={*widestneed* | *strict* | *no*}]

(SPARC) 启用区间运算扩展。

可选值可以是 no、widestneed 或 strict。如果未指定，则默认值是 widestneed。

否	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 区间运算编程指南》中详细说明了区间运算的 Fortran 95 扩展。另请参见第 3-68 页的 "-xia[={widestneed|strict}]"。

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

(SPARC) 执行过程间优化。

通过调用过程间分析传递，执行整个程序优化。不同于 -xcrossfile， -xipo 将在链接步骤中对所有目标文件执行优化，并且不仅限于编译命令的源文件。

在编译和链接大型多文件应用程序时， -xipo 特别有用。使用此标志编译的目标文件在其内部编译分析信息，从而在源文件和预编译的程序文件间启用过程间调用分析。不过，分析和优化只限于用 -xipo 编译的目标文件，而不扩展到库的目标文件。

-xipo=0 禁用过程间分析，-xipo=1 则启用过程间分析。-xipo=2 添加过程间别名分析和内存分配以及布局优化，以便改善缓存性能。默认值为 -xipo=0，如果未使用值指定 -xipo，则使用 -xipo=1。

使用 -xipo=2 进行编译时，没有使用 -xipo=2 编译的函数或子例程（例如，库）不能调用使用了 -xipo=2 编译的函数或子例程。

例如，如果您干预函数 malloc() 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 不该使用 -xipo=2 进行编译。

在不同的步骤中编译和链接时，必须在两个步骤中都指定 -xipo 才是有效的。

在单个编译 / 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。该操作在链接的最后一步完成，因此源文件的所有编译不必全在单一的编译中完成，而且这些源文件的编译会覆盖许多独立的编译，其中每个编译都会指定 -xipo。

在单独的编译 / 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这些文件内编译的附加分析信息，这样就可以在链接步骤中执行交叉文件优化。

即使用 -xipo 编译，仍存在库不参与交叉文件过程间调用分析的约束，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，过程间优化将在以下例程之间执行：one.f、 two.f 和 three.f 之间、main.f 和 four.f 之间，但不在 main.f 或 four.f 和 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
- 与 -xcrossfile 冲突；如果一起使用，将导致编译错误
- 不使用 -xipo 编译的对象可以与使用 -xipo 编译的对象自由链接。
- 由于优化文件时需要附加信息，因此 -xipo 选项会生成更大的目标文件。不过，该附加信息不会成为最后可执行的二进制文件的一部分。可执行程序大小的增加都是由于执行的附加优化导致的
- 在此发行版中，交叉文件程序内联是由 -xipo 执行的唯一过程间优化。
- .s 汇编程序语言文件不参与过程间分析。
- 使用 -S 编译时 -xipo 标记被忽略。

不使用 -xipo 编译时：

在链接步骤中使用目标文件集合时，编译器试图执行整个程序分析和优化。对于该目标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， foo() 将无法被在该目标文件集合之外定义的其他例程显式调用，并且

(2) 来自该目标文件集合中的任何例程的 foo() 调用将不受在该目标文件集合以外定义的不同版本的 foo() 的干预。

如果假定 (1) 对于给定的应用程序不成立，则不使用 -xipo=2 编译。
如果假定 (2) 不成立，则既不使用 -xipo=1 也不使用 -xipo=2 编译。

例如，考虑使用您自己的源版本干预函数 malloc() 并使用 -xipo=2 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 编译，并且它们的目标文件将不需要参与链接步骤。由于这对于系统库不大可能，因此您的版本的 malloc 不该使用 -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()，这会导致使用 -xipo 编译时出现不正确的结果。

-xipo_archive[={none|readonly|writeback}]

(SPARC) 允许交叉文件优化包括归档 (.a) 库。

值必须是以下项之一：

writeback	生成可执行文件之前，编译器使用通过驻留在 .a 归档库中的 -xipo 编译的目标文件来优化传递到链接程序的对象文件。在编译期间优化的库中包含的任何目标文件都替换为其优化后的版本。
readonly	生成可执行文件之前，编译器使用通过驻留在 .a 归档库中的 -xipo 编译的目标文件优化传递到链接程序的目标文件。
none	不执行归档文件的处理。

如果不指定 -xipo_archive 的设置，编译器将假定 -xipo_archive=none。

-xjobs=*n*

使用多个处理器进行编译。

指定 -xjobs 选项可以设定编译器创建多少个进程来完成它的任务。在多 cpu 机器上，该选项可以减少生成时间。在本发行版的 f95 编译器中，-xjobs 只能和 -xipo 选项一起使用。如果指定 -xjobs=*n*，过程间调用优化器在编译不同的文件时会用 *n* 作为它能启动的代码生成器实例的最大数。

通常，*n* 的安全值等于 1.5 乘以可用处理器的数量。由于上下文在产生的作业间切换的开销，使用等于可用处理器数量整数倍的值会降低性能。此外，如果使用很大的数值会耗尽系统资源 （如交换空间）。

必须带值来指定 -xjobs。否则会发出错误诊断并使编译终止。

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。

以下示例在有两个处理器的系统上进行编译时比不使用 -xjobs 选项而使用相同命令进行编译时更为快速。

```
example% f95 -xipo -xO4 -xjobs=3 t1.f t2.f t3.f
```

-xknown_lib=*library_list*

识别对已知库的调用。

如果指定此选项，编译器将把对某些已知库的调用视为内在函数，而忽略任何用户提供的版本。这样，编译器就可以基于其对该库的特殊了解来对库例程调用进行优化。

library_list 是当前应用于 `blas`、`blas1`、`blas2`、`blas3` 和内部函数的关键字列表（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，并且能为 Sun Performance Library 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的版本，并链接到 Sun 性能库中的 BLAS 例程。

-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。等效于 `-xknown_lib=blas1,blas2,blas3`
内部函数	编译器忽略任何用于 Fortran 95 内部函数的显式 `EXTERNAL` 声明，因此忽略任何用户提供的内存函数例程。

-xlang=f77

(SPARC) 准备链接使用更早版本的 `f77` 编译的运行时库。

f95 -xlang=f77 表示使用 f77compat 库进行链接，这是使用更旧的 Fortran 77 目标文件链接 Fortran 95 目标文件的快速方法。使用此标记编译可以确保正确的运行时环境。

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

使用 -xlang 编译时请注意以下事项：

- 不要同时使用 -xnolib 和 -xlang 编译。
- 将 Fortran 目标文件和 C++ 混合使用时，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 目标文件混合使用时，链接 CC 命令行还必须指定 -mt。

-xlibmil

等效于 -libmil。

-xlibmopt

使用优化的数学例程库。

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；如果这样，通常是最后那个位不同。该库选项在命令行上的顺序并不重要。

-xlic_lib=sunperf

与 Sun 性能库链接。

例如：

```
f95 -o pgx -fast pgx.f -xlic_lib=sunperf
```

如同 -l 一样，此选项应在命令行中显示在所有源和目标文件名之后。

要与 Sun 性能库链接，必须使用此选项。（请参见 《Sun 性能库用户指南》。）

-xlicinfo

显示许可证信息。

使用此选项可以返回有关已安装的编译器软件的序列号授权信息。

-xlinkopt[={1|2|0}]

(SPARC) 在可重定位目标文件中执行链接时优化。

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。（这是默认情况。）

1　在链接时根据控制流分析执行优化，其中包括指令缓冲着色和分支优化。

2　在链接时执行附加的数据流分析，其中包括终止的代码排除和地址计算简化。

指定不带有值的 -xlinkopt 标志意味着 -xlinkopt=1。

这些优化在链接时通过分析对象二进制代码执行。虽然未重写对象文件，但产生的可执行代码可能与初始对象的代码不同。

当与性能分析反馈一起用于编译整个程序时，这个选项非常有效。

如果在不同的步骤中编译，则 -xlinkopt 必须同时出现在编译和链接步骤中。

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

注意，仅当链接编译器时才使用级别参数。在以上示例中，即使目标二进制代码是用默认级别 1 编译的，使用的后优化级别仍然是 2。

您不能同时使用链接时间后优化器和递增链接程序 ild。-xlinkopt 标记将把默认链接程序设置为 ld。如果使用 -xildon 标记显式启用增量式链接程序，并且同时指定两者，将禁用 -xlinkopt 选项。

要使 -xlinkopt 选项有用，至少程序中的部分例程 （但并不需要全部例程）必须使用此选项编译。优化器仍可以在未使用 -xlinkopt 进行编译的对象二进制文件上执行部分受限的优化。

-xlinkopt 选项将优化出现在编译器命令行上的静态库的代码，但会跳过出现在命令行上的共享 （动态）库代码而不对其进行优化。创建共享库 （用 -G 编译）时，您也可以使用 -xlinkopt。

与运行时性能分析反馈一起使用时，链接时后优化器最有效。配置会显示代码中最常用和最少用的部分并相应地指导优化器集中其努力方向。这对大型应用非常重要，因为链接时执行代码地优化放置可以减少指令的高速缓存缺失。通常，会按照以下方式进行编译：

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

有关使用性能分析反馈的详细信息，请参见 -xprofile 选项

注意，使用该选项编译会略微延长链接的时间。也会增加对象文件的大小，但可执行文件的大小保持不变。使用 -xlinkopt 和 -g 标记编译会将调试信息包括在内，从而增加了可执行文件的大小。

-xloopinfo

等效于 衛 oopinfo。

-xmaxopt[=*n*]

启用优化 pragma 并设置最高优化级别。

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。

当此指令显示在源输入中时，此选项启用 C$PRAGMA SUN OPT=*n* 指令。不使用此选项时，编译器将这些行视为注释。请参见第 2-10 页的 2.3.1.5 节 “OPT 指令”。

如果 pragma 显示时优化级别大于 -xmaxopt 标记上的最高级别，编译器将使用 -xmaxopt 设置的级别。

-xmemalign[=<*a*><*b*>]

(SPARC) 指定最大的假定内存对齐和未对齐的数据访问行为。

对于可在编译时决定对齐的内存访问，编译器会为数据对齐生成适当的装入 / 存储指令序列。

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

xmemalign 标志允许用户在这些未确定情况下指定编译器要假定的数据最大内存对齐。它还指定在运行时发生未对齐内存存取时的错误行为。

指定的值包含两个部分：数值对齐值 <*a*>，以及字母行为标记 <*b*>。

对齐 <*a*> 的允许的值为：

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。

访问未对齐的数据 <*b*> 的错误行为的允许值为：

i 解释访问并继续执行
s 产生信号 SIGBUS
f 仅限在 -xarch=v9 变体上，为小于或等于 4 的对齐产生信号 SIGBUS，否则解释访问权限并继续执行。在其他平台上， f 等价于 i。

不指定 -xmemalign 进行编译时的默认值为：

- 8i （对于 -xarch=generic、v7、v8、v8a、v8plus、v8plusa）
- 8s （对于 C 和 C++ -xarch=v9 变体）
- 8f （对于 Fortran -xarch=v9 变体）

对于所有平台， -xmemalign 不带值显示时的默认值为 1i。

请注意， -xmemalign 本身并不强制发生任何特殊的数据对齐。使用 -dalign 或 -aligncommon 可以强制数据对齐。

-dalign 选项是一个宏：

-dalign 是一个宏，用于：-xmemalign=8s -aligncommon=16

有关详细信息，请参见第 3-10 页的 "-aligncommon[={1|2|4|8|16}]"。

-xmodel=[small | kernel | medium]

(x86) 在 Solaris x86 平台上创建 64 位共享对象。

通过使用 -xmodel 选项，编译器可以为 Solaris x86 平台创建 64 位共享对象，并只应为此类对象的编译指定此选项。

仅当还指定了 -xarch=generic64、-xarch=amd64 或 -xarch=amd64a 时，该选项才有效。

small

此选项为小模型生成代码，在此模型中，在链接时已知所执行代码的虚拟地址，并且已知所有符号均位于范围在 0 至 2^31 - 2^24 - 1 之间的虚拟地址。

kernel

为内核模型生成代码，在此模型中，将所有符号定义为位于 2^64 - 2^31 和 2^64 - 2^24 之间的范围内。

medium

为中模型生成代码，在此模型中，没有假定有关数据部分的符号引用的范围。文本部分的大小和地址与小代码模型具有相同的限制。

如果未指定 -xmodel，则编译器假定 -xmodel=small。如果指定不带参数的 -xmodel，则会出现错误。

只要确保所访问的对象在范围之内，则无需使用该选项来编译所有例程。

-xnolib

等效于 -nolib。

-xnolibmil

等效于 -nolibmil。

-xnolibmopt

不使用快速数学库。

与 -fast 一起使用时可以覆盖对优化数学库的链接：

f95 衒 ast 祄 nolibmopt _

-xO*n*

等效于 -O*n*。

-xopenmp

(SPARC) 等效于 -openmp。

-xpad

等效于 -pad。

-xpagesize=*size*

设置优先使用的栈和堆页面大小。

在 SPARC 平台上， *size* 值必须是以下之一：

8K 64K 512K 4M 32M 256M 2G 16G 或默认值

在 x86 平台上， *size* 值必须是以下之一：

4K 2M 4M 或默认值

例如：-xpagesize=4M

并非所有平台上都支持所有这些页面大小，具体取决于体系结构和 Solaris 环境。指定的页面大小对于目标平台上的 Solaris 操作环境必须是有效的页面大小，其值由 getpagesize(3C) 返回。如果不是，请求在运行时将被忽略。 Solaris 环境不保证支持页面大小的请求。

您可以使用 pmap(1) 或 meminfo(2) 来确定运行的程序是否收到请求的页面大小。

如果指定 -xpagesize=default，将忽略标记；如果指定的 -xpagesize 不带有 *size* 值，则等效于 -xpagesize=default。

此选项是以下对象的宏

```
-xpagesize_heap=size -xpagesize_stack=size
```

这两个选项接受与 -xpagesize 参数相同的参数：8K、64K、512K、4M、32M、256M、2G、16G、默认值。您可以通过指定 -xpagesize=*size* 来为二者设置相同的值，或分别为它们指定不同的值。

使用该标记进行编译与使用等价的选项将 LD_PRELOAD 环境变量设置为 mpss.so.1 或在启动程序之前使用等价的选项运行 Solaris 9 命令 ppgsz(1) 具有相同的效果。有关详细信息，请参见 Solaris 9 手册页。

注意，该特性在 Solaris 7 和 8 环境中不可用。在 Solaris 7 和 8 环境上不链接使用该选项编译的程序。

-xpagesize_heap=*size*

设置优先使用的堆页面大小。

size 值与所述的 -xpagesize 值相同。

有关详细信息，请参见 -xpagesize。

-xpagesize_stack=*size*

(SPARC) 设置堆栈的首选页面大小。

size 值与所述的 -xpagesize 值相同。

有关详细信息，请参见 -xpagesize。

-xparallel

等效于 衢arallel。

-xpg

等效于 -pg。

-xpp={fpp|cpp}

选择源文件预处理程序。

默认值为 -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。以前版本使用标准 C 预处理程序 cpp。要选择 cpp，请指定 -xpp=cpp。

-xprefetch[=*a*[,*a*]]

在支持预取的那些体系结构上启用预取指令，如 UltraSPARC II 或 UltraSPARC III、Pentium 3、Pentium 4 或 AMD Opteron（-xarch=v8plus、v8plusa、v9plusb、v9、v9a 或 v9b、sse、sse2、generic64、native64 或 amd64）

有关 Fortran PREFETCH 指令的说明，请参见第 2-12 页的 2.3.1.8 节 “PREFETCH 指令”。

a 必须是以下值之一：

a 是	含义
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
显式	启用显式预取宏 （仅限 SPARC）
no%explicit	禁用显式预取宏 （仅限 SPARC）
latx:*factor*	按照指定的因子调整编译器假定的预取到装入和预取到存储的延迟。该因子必须是正浮点数或整数。（仅限 SPARC）
是	-xprefetch=yes 与 -xprefetch=auto,explicit 相同
否	-xprefetch=no 与 -xprefetch=no%auto,no%explicit 相同

使用 -xprefetch,-xprefetch=auto 和 -xprefetch=yes，编译器就可以将预取指令自由插入到它生成的代码中。该操作会提高支持预取的架构的性能。

如果正在较大的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默认延迟时间。

预取延迟是执行预取指令和预取数据在缓存中可用时间之间的硬件延迟。编译器决定放置使用预取数据的预取指令和装入或存储指令的距离时，假定预取延迟值。

注 – 在预取和装入之间假定的延迟可能与在预取和存储之间假定的延迟不同。

编译器在多个机器和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。对于内存密集的应用程序，尤其是要在较大的多处理器上运行的应用程序，您可以通过增加预取延迟值获得更高的性能。要增加值，请使用大于 1 的因子。在 .5 和 2.0 之间的值最有可能提供最高的性能。

对于具有完全位于外部缓存内的数据集的应用程序，您可以通过减小预取延迟值来获得更高的性能。要减小此值，请使用小于 1 的因子。

要使用 -xprefetch=latx:*factor* 选项，请在开始时使用接近 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。获得最优性能之前，可以不断调整因子并运行性能测试。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，然后会突然出现差异，最后再趋于稳定。

默认：

如果未指定 -xprefetch，则假定为 -xprefetch=no%auto,explicit。

如果仅指定了 -xprefetch，则假定为 -xprefetch=auto,explicit。

只有使用没有任何参数或具有 auto 或 yes 参数的 -xprefetch 进行显式覆盖时，才假定 no%auto 的默认。例如， -xprefetch=explicit 与 -xprefetch=explicit,no%auto 相同。

只有在使用 no%explicit 的参数或 no 的参数进行显式覆盖时，才假定 explicit 的默认。例如， -xprefetch=auto 与 -xprefetch=auto,explicit 相同。

如果启用了诸如使用 -xprefetch 或 -xprefetch=yes 的自动预取，但不指定延迟因子，则假定为 -xprefetch=latx:1.0。

交互：

-xprefetch=explicit 时，编译器将识别指令：

```
$PRAGMA SPARC_PREFETCH_READ_ONCE (name)
$PRAGMA SPARC_PREFETCH_READ_MANY (name)
$PRAGMA SPARC_PREFETCH_WRITE_ONCE (name)
$PRAGMA SPARC_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。

只有自动预取启用时才可以使用 latx:*factor* 子选项。也就是说，除非与 auto 一起使用，否则忽略 latx:*factor*。

警告：

显式预取只应在度量支持的特殊环境下使用。

因为编译器在多个机器和应用程序之间调节预取机制以获得最优性能，所以当性能测试指示性能明显提高时，应该仅使用 -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。

-xprefetch_auto_type=[no%]indirect_array_access

生成间接预取，以间接访问数据数组。

[不要]以生成直接内存存取预取的相同方式生成由选项 -xprefetch_level={1|2|3} 指示的循环的间接预取。前缀 no% 否定声明。

如果不指定 -xprefetch_auto_type 的设置，编译器将把它设置为 -xprefetch_auto_type=no%indirect_array_access。

要求 -xprefetch=auto 以及优化级别 -xO3 或更高级别。

类似 -xdepend 的选项可以影响计算候选间接预取的主动性，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。

-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（-xarch=v8plus、v8plusa、v8plusb、v9、v9a、v9b、generic64、 native64）。

未指定 -xprefetch_level 时的 -xprefetch=auto 的默认值为级别 2。

预取级别 2 生成预取指令大于级别 1 的其他机会。预取级别 3 生成大于级别 2 的其他预取指令。

预取级别 2 和 3 仅在 UltraSPARC III 平台（-xarch=v8plusb 或 v9b）或者 x86 Pentium 4 或 AMD Opteron （-xarch=sse2 或 amd64）上有效

-xprofile={collect[:*name*]|use[:*name*]|tcov}

使用运行时分析数据收集或优化，或执行基本块覆盖分析。

为编译器提供运行时的性能反馈增强了使用高优化级别 (-xO5) 进行编译的效果。为了生成编译器执行最佳优化所需的性能分析反馈，您必须使用 -xprofile=collect 编译，对典型数据集运行可执行文件，然后用最高的优化级别并使用 -xprofile=use 重新编译。

collect[:*name*]

优化器使用 -xprofile=use 收集并保存执行频率数据以备将来使用。编译器生成测量语句执行频率的代码。

name 是被分析的程序的名称。该名称是可选的。如果未指定 *name*，则假定 a.out 为可执行程序的名称。

在运行时，使用 -xprofile=collect:*name* 编译的程序在默认情况下会创建子目录 *name*.profile 来保存运行时的反馈信息。程序将其运行时性能分析数据写入该子目录中名为 feedback 的文件里。如果多次运行程序，那么执行频率数据会累积在 feedback 文件中，也就是说以前运行的输出不会丢失。

您可以设置环境变量 SUN_PROFDATA 和 SUN_PROFDATA_DIR 来控制使用 -xprofile=collect 编译的程序在其中写入其运行时性能分析数据的文件和目录。设置这些变量后，使用 -xprofile=collect 编译的程序将其性能分析数据写入 $SUN_PROFDATA_DIR/$SUN_PROFDATA。

这些环境变量同样控制 tcov 写入的性能分析数据文件的路径和名称，tcov(1) 手册页描述了此点。

性能分析数据集合为 “多线程安全” 。也就是说，通过使用 -mt 编译并直接调用多任务库来执行自身多任务的程序将产生准确的结果。

在单独步骤中编译和链接时，链接步骤还必须指定 -xprofile=collect （如果它显示在编译步骤中）。

use[:*nm*]

使用执行频率数据有策略地在优化级别 -xO5 进行优化。

与 collect:nm 一起使用时， *nm* 是可选的，可用于指定程序的名称。

程序是使用以前生成并保存在性能分析文件中的执行频率数据优化的，此数据由先前执行用 -xprofile=collect 编译的程序写入。

源文件和其他编译器选项必须与用于编译的源文件和编译器选项完全一致，该编译创建了生成 feedback 文件的编译程序。如果使用 -xprofile=collect:*nm* 进行编译，则相同的程序名称 *nm* 必须出现在优化编译：-xprofile=use:*nm* 中。

有关加速收集和使用阶段之间的编译的说明，另请参见 -xprofile_ircache。

有关控制编译器在哪里查找性能分析文件的说明，另请参见 -xprofile_pathmap。

tcov

使用 “新”样式 tcov 的基本块覆盖分析。优化级别必须是 -O2 或更高级别。

代码指令与 -a 选项的代码指令类似，但不再为每个源文件生成 .d 文件。相反却生成单一文件，并且该文件的名称是按照最后的可执行文件命名的。例如，如果 stuff 是可执行文件，则 stuff.profile/tcovd 是数据文件。

运行 tcov 时，您必须将 -x 选项传递给它，以使它使用新式样数据。否则，tcov 使用旧式 .d 文件 （如果有），这是数据的默认值，并产生不可预测的输出。

与 -a 不同，TCOVDIR 环境变量在编译时没有影响。但是，在程序运行时会使用它的值来确定在哪里创建性能分析子目录。

有关详细信息，请参见 tcov(1) 手册页、 Fortran Programming Guide 中的 “Performance Profiling”一章和 Program Performance Analysis Tools 手册。

注 – 如果由于 -O4 或 -inline 存在子程序的内联处理，则 tcov 生成的报告可能不可靠。不会记录对已经内联的例程的调用的覆盖。

-xprofile_ircache[=*path*]

(SPARC) 保存并重用收集和使用性能分析数据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* 将覆盖保存缓存文件的位置。默认情况下，这些文件会作为目标文件保存在同一目录中。收集和使用阶段出现在两个不同的位置中时，指定路径才是有用的。

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
```

编译大程序时，由于中间数据的保存，使得使用阶段的编译时间大大减少。但这将以可能大大增加磁盘空间的占用为代价。

-xprofile_pathmap=*collect_prefix*:*use_prefix*

(SPARC) 设置性能分析文件的路径映射。

将 -xprofile_pathmap 选项与 -xprofile=use 选项一起使用。

如果编译器无法找到用 -xprofile=use 编译的目标文件的文件配置数据，请使用 -xprofile_pathmap：

- 您在使用 -xprofile=use 编译到某个目录，而该目录不是先前使用 -xprofile=collect 编译时使用的目录。
- 对象文件会共享文件配置中的公共基名，但却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，在该目录树中使用 -xprofile=collect 编译对象文件。

use-prefix 是目录树的 UNIX 路径名的前缀，在该目录树中使用 -xprofile=use 编译对象文件。

如果指定了 -xprofile_pathmap 的多个实例，那么编译器将按照这些实例出现的顺序对其进行处理。在标识了匹配的 *use-prefix* 或发现最后指定的 *use-prefix* 与对象文件路径名不匹配之前，每个由 -xprofile_pathmap 的实例指定的 *use-prefix* 都会与对象文件路径名进行比较。

-xrecursive

允许不带 RECURSIVE 属性的例程以递归方式调用它们。

通常，只有使用 RECURSIVE 属性定义的子程序可以以递归方式调用它们。

使用 -xrecursive 编译可以允许子程序调用它们自身，即使它们没有使用 RECURSIVE 属性进行定义。但是，与定义了 RECURSIVE 的子例程不同，使用此标记不会导致默认情况下在堆栈上分配本地变量。要使本地变量在每个子程序的递归调用中具有单独的值，还应该使用 -stackvar 进行编译以便将本地变量放在堆栈上。

间接递归（例程 A 调用例程 B，而例程 B 又调用例程 A）可以生成优化级别大于 -xO2 的不一致的结果。使用 -xrecursive 标记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。

使用 -xrecursive 进行编译会导致性能下降。

-xreduction

等效于 -reduction。

-xregs=*r*

指定寄存器的用法。

r 是一个逗号分隔列表，它包含以下一项或多项：

[no%]appl、[no%]float。

其中 % 已显示，它是必须的字符。

示例：-xregs=appl,no%float

- appl
  （仅限 SPARC）允许编译器将应用程序寄存器作为临时寄存器使用。

  在 SPARC 系统上，某些寄存器被描述为**应用程序**寄存器。由于需要较少装入和存储指令，因此使用这些寄存器可提高性能。但是，此类使用可能与某些用汇编代码编写的旧库程序冲突。

  应用程序寄存器集合依赖于 SPARC 平台：

  - -xarch=v8 或 v8a — 注册 %g2、 %g3 以及 %g4
  - -xarch=v8plus 或 v8plusa — 注册 %g2、 %g3 和 %g4
  - -xarch=v9 或 v9a — 注册 %g2 和 %g3
- no%appl
  不使用 appl 寄存器。
- float
  （仅限 SPARC）允许编译器将浮点寄存器作为整型值的临时寄存器使用。此选项不会影响编译器为浮点值使用浮点寄存器。

- no%float
  不使用浮点寄存器。使用此选项时，源程序不能包含任何浮点代码。
- frameptr
  （仅限 x86）允许编译器将帧指针寄存器（x86 处理器上为 %ebp， AMD64 处理器上为 %rbp）用作未分配的被调用方保存寄存器以提高程序性能。但要注意，使用 -xregs=frameptr 进行编译可能会降低某些工具检查和跟踪栈的功能，如性能分析器和 DTRACE。栈检查是系统性能测量和调节的一项重要要求。如果还使用 -xpg 或 -p 进行编译，则会忽略 -xregs=frameptr。
- no%frameptr
  不使用帧指针寄存器。

默认值为：-xregs=appl,float,no%frameptr。

对于与应用程序相链接的共享库，强烈推荐您使用 -xregs=no%appl,float 来编译用于这些库的代码。至少，共享库应该显式说明它如何使用应用程序寄存器，从而用那些库链接的应用程序寄存器知道如何处理该问题。

例如，在某些全局检测中使用寄存器的应用程序（例如，使用寄存器指向一些临界数据结构）将需要确切地知道带有未使用 -xregs=no%appl 编译的代码的库如何使用应用程序寄存器以便安全链接该库。

-xs

允许在用 dbx 调试时不包括对象 (.o) 文件。

使用 -xs 时，所有调试信息都复制到可执行文件中。如果将可执行文件移至另一个目录，则可以使用 dbx 并忽略对象 (.o) 文件。如果不能保存 .o 文件，请使用该选项。

不带 -xs 时，如果移动可执行文件，则必须同时移动源文件和对象 (.o) 文件，或使用 dbx pathmap 或 use 命令来设置路径。

-xsafe=mem

(SPARC) 允许编译器假定不违反内存保护。

使用此选项可允许编译器假定未发生基于内存的陷阱。该选项允许在 SPARC V9 平台上使用推测装入指令。

该选项只有与优化级别 -O5 以及以下体系结构 (-xarch) 之一一起使用时才有效：v8plus、 v8plusa、 v8plusb、 v9、 v9a 或 v9b

注意 – 由于在发生诸如地址未对齐或段违规的故障时，非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。因为很少的程序会导致基于内存的自陷，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。

-xsb

（已废弃）等效于 -sb。

-xsbfast

（已废弃）等效于 -sbfast。

-xspace

不执行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。

-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、*platform-name*。

-xtarget 选项是一个宏，它允许快捷、轻松的指定发生在真实平台上的 -xarch、xchip 和 -xcache 的组合。 -xtarget 的唯一含义位于它的扩展中。

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改善。当程序性能很重要时，目标硬件的正确说明会是非常重要的。在较新的 SPARC 处理器上运行时这一点尤其重要。不过，对多数程序和较旧的 SPARC 处理器来讲，性能的获取是可以忽略的，而通用规范已经足够了。

native：优化主机平台的性能。

编译器生成为主机平台优化的代码。它决定了运行编译器的计算机的可用架构、芯片和缓存属性。

native64：为本地 64 位环境编译。

为编译器所运行的计算机上的 64 位环境设置体系结构、芯片和高速缓存属性。

generic：获取通用架构、芯片和缓存的最佳性能。

编译器将 -xtarget=generic 扩展到：

```
-xarch=generic -xchip=generic -xcache=generic
```

这是默认值。

generic64：为通用 64 位环境编译。

它扩展为 -xarch=v9 -xcache=generic -xchip=generic

platform-name：获取指定平台的最佳性能。

SPARC 平台

使用 fpversion(1) 命令在运行的系统上决定 -xtarget=native 的扩展。

请注意，在该平台上进行编译时，特定主机平台的 -xtarget 不能扩展到与 -xtarget=native 相同的 -xarch、 -xchip 或 -xcache 设置上。

下表列出了编译器接受的常用系统平台的名称。附录 C 列出了较旧和较不常用的系统平台的名称。

表 3-18 常用的 -xtarget 系统平台的扩展

-xtarget=*platform-name*	**-xarch**	**-xchip**	**-xcache**
generic	generic	generic	generic
generic64	v9	generic	generic
entr150	v8plusa	ultra	16/32/1:512/64/1
entr2	v8plusa	ultra	16/32/1:512/64/1
entr2/1170	v8plusa	ultra	16/32/1:512/64/1
entr2/1200	v8plusa	ultra	16/32/1:512/64/1
entr2/2170	v8plusa	ultra	16/32/1:512/64/1
entr2/2200	v8plusa	ultra	16/32/1:512/64/1
entr3000	v8plusa	ultra	16/32/1:512/64/1
entr4000	v8plusa	ultra	16/32/1:512/64/1
entr5000	v8plusa	ultra	16/32/1:512/64/1
entr6000	v8plusa	ultra	16/32/1:512/64/1
ultra	v8plusa	ultra	16/32/1:512/64/1
ultra1/140	v8plusa	ultra	16/32/1:512/64/1
ultra1/170	v8plusa	ultra	16/32/1:512/64/1
ultra1/200	v8plusa	ultra	16/32/1:512/64/1
ultra2	v8plusa	ultra2	16/32/1:512/64/1
ultra2/1170	v8plusa	ultra	16/32/1:512/64/1
ultra2/1200	v8plusa	ultra	16/32/1:1024/64/1
ultra2/1300	v8plusa	ultra2	16/32/1:2048/64/1
ultra2/2170	v8plusa	ultra	16/32/1:512/64/1
ultra2/2200	v8plusa	ultra	16/32/1:1024/64/1
ultra2/2300	v8plusa	ultra2	16/32/1:2048/64/1
ultra2e	v8plusa	ultra2e	16/32/1:256/64/4

表 3-18 常用的 -xtarget 系统平台的扩展 （续）

-xtarget=*platform-name*	-xarch	-xchip	-xcache
ultra2i	v8plusa	ultra2i	16/32/1:512/64/1
ultra3	v8plusa	ultra3	64/32/4:8192/512/1
ultra3cu	v8plusa	ultra3cu	64/32/4:8192/512/2
ultra3i	v8plusa	ultra3i	64/32/4:1024/64/4
ultra3iplus	v8plusa	ultra3iplus	64/32/4:4096/64/4
ultra4	v8plusa	ultra4	64/32/4:8192/128/2
ultra4plus	v8plusa	ultra4plus	64/32/4/1:2048/64/4/2:32768/64/4/2
ultraT1	v8plusa	ultraT1	8/16/4/4:3072/64/12/32

-xarch=v9 或 -xarch=v9a 标志指示 UltraSPARC V9 平台上的 64 位 Solaris 操作系统上的编译 。设置 -xtarget=ultra 或 ultra2 是不必要的或不足够的。如果指定了 -xtarget，则 -xarch=v9 或 v9a 选项必须出现在 -xtarget 标志之后（如下所示）：

```
-xtarget=ultra2 ...-xarch=v9
```

否则， -xtarget 设置 将把 -xarch 还原为 v8plusa。

X86 平台

对于 x86 系统有效的 -xtarget 平台名称如下：

generic、native、386、486、pentium、pentium_pro、pentium3、pentium4 和 opteron。

-xarch=amd64 标志指示 64 位 x86 AMD Opteron 平台上的 64 位 Solaris 操作系统上的编译 。使用 -xtarget=opteron 进行编译是不必要的或不足够的。如果指定了 -xtarget，则 -xarch=amd64 选项必须出现在 -xtarget 标志之后 （如下所示）：

```
-xtarget=opteron -xarch=amd64
```

否则，编译将还原为 32 位 x86。

-xtarget 值的实际扩展可能会随不同的发行版本而发生变化。始终可以使用 -dryrun 标志来确定编译器将使用的扩展：

```
demo% f95 -dryrun -xtarget=ultra4plus
###     command line files and options (expanded):
### -dryrun -xarch=v8plusa
-xcache=64/32/4/1:2048/64/4/2:32768/64/4/2 -xchip=ultra4plus
```

-xtime

等效于 -time。

-xtypemap=*spec*

指定默认数据映射。

此选项提供了灵活的方法，用于为默认数据类型指定字节大小。此选项应用于默认大小的变量和常量。

规范字符串 *spec* 可能包含列表中 （用逗号分隔）以下任何或所有项目：

```
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:32
- integer:64

例如：

- -xtypemap=real:64,double:64,integer:64

将默认的 REAL 和 DOUBLE 映射到 8 字节。

此选项应用于使用默认规范（不带显式字节大小）声明的所有变量，如同 REAL XYZ（产生 64 位 XYZ）中的变量。同时，所有单精度 REAL 常数都将被提升为 REAL*8 常数。

请注意，INTEGER 和 LOGICAL 处理的方式相同，并且 COMPLEX 映射为两个 REAL。同时，处理 DOUBLE COMPLEX 的方式如同映射 DOUBLE 的方式。

-xunroll=*n*

等效于 -unroll=*n*。

-xvector[=[{yes|no}] [[no%]lib,[no%]simd,%none]]

启用对向量库函数调用的自动生成。

在使用 -xvector 进行编译时，此选项要求使用默认舍入模式 -fround=nearest 进行编译。

如果循环内的数学库调用可转换为对等价向量数学例程的单个调用，则 -xvector=lib 允许编译器进行此类转换。此类转换可提高那些循环计数较大的循环的性能。-xvector=no%lib 禁用此功能。

-xvector=simd 允许编译器使用本机 x86 SSE SIMD 指令来提高某些循环的性能。仅当目标体系结构支持 SIMD 指令时，编译器才会接受此开关。例如，您必须指定 -xarch=amd64 或 -xarch=generic64。还必须指定 -xO3 或更高的优化级别以及 -xdepend 和 -xvector=simd。 -xvector=no%simd 禁用此功能。

如果同时指定了 -xvector=simd 和 -fsimple=2，则可以获得比单独指定 -xvector=simd 更好的性能。但是，浮点结果可能会略有不同，因为 -fsimple=2 允许重新对浮点运算进行排序。

默认值为 -xvector=%none。如果指定 -xvector，但没有提供子选项，则编译器假定 -xvector=lib。

在装入步骤中，编译器包含 libmvec 库。如果在编译时指定 -xvector=lib，则还必须在链接时指定它。

此选项覆盖以前的实例，因此， -xvector=%none 覆盖以前指定的 -xvector=lib。

在以后的发行版本中，-xvector=yes 选项可能会过时。请改为指定 -xvector=lib。

在以后的发行版本中，-xvector=no 选项可能会过时。请改为指定 -xvector=no%lib,no%simd。

-ztext

在不重定位的情况下仅生成纯库。

-ztext 的一般用途是验证生成的库是否为纯文本；指令是否全部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，则通常会依然生成库；数据段是可写入的。

不带 -ztext 时， ld 生成库、重定位或不生成。

如果您不知道目标文件是否是使用 -pic 生成的，则一种典型用法是使库同时来自于源文件和目标文件。

示例：使库同时来自于源文件和目标文件：

```
demo% f95 -G -pic -ztext -o MyLib -hMyLib a.f b.f x.o y.o
```

另一种替代用法是询问代码是否已经与位置无关。不带 -pic 进行编译，但询问是否为纯文本。

示例：询问它是否已经是纯文本椛踔敛淮 -pic：

```
demo% f95 -G -ztext -o MyLib -hMyLib a.f b.f x.o y.o
```

如果使用 -ztext 编译，并且 ld 不生成库，则您可以不使用 -ztext 重新编译，此时 ld 将生成库。使用 -ztext 生成失败意味着库的一个或多个组件无法共享；但是，某些其他组件可能可以共享。这就产生了性能问题，最好由您 — 编程人员来解决。

第 4 章

Fortran 95 功能和差异

本附录介绍了标准 Fortran 95 和 Fortran 95 编译器 `f95` 之间的一些主要功能差异。

4.1 源语言功能

Fortran 95 编译器提供 Fortran 95 标准的以下源语言功能和扩展。

4.1.1 续行限制

`f95` 允许 99 个续行 （1 个初始行和 99 个续行）。在固定格式中，标准 Fortran 95 允许 19 个续行；在自由格式中，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 只允许使用 72 个字符长的行。

f95 中的制表符强制将行的其余部分填充到第 72 列。如果制表符出现在延续到下一行的字符串内，这可能会导致出现意外结果：

```
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.
```

4.1.3 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 后缀的文件使用固定格式。具有 .f90、.f95、.F90 或 .F95 后缀的文件使用自由格式。

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。

4.1.3.1 混合格式

允许混合使用某些源代码格式。

- 在相同的 f95 命令中，某些源文件可以是固定格式，而某些源文件是自由格式。
- 在相同的文件中，**可**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。

4.1.3.2 大小写

在默认情况下， Sun 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。

4.1.4 限制和默认值

- 单个 Fortran 95 程序单元可定义最多 65,535 个派生类型和 16,777,215 个不重复的常量。
- 变量和其他对象的名称最多为 127 个字符长。标准长度为 31 个。

4.2 数据类型

本节介绍 Fortran 95 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

- **屏蔽** — 具有布尔结果的按位逻辑表达式；它的每个位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。
  - 逻辑型的变量、常量或表达式仅有两个值：`.TRUE.` 或 `.FALSE.`。布尔型的变量、常量或表达式可以表示任意二进制值。
  - 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（第 29、第 30 和第 31 位）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。其他位均为 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* 是任意的十六进制数字

- 可以是 1 至 8 个十六进制数字（0 至 9， `A-F`）。
- 任何字母都可以是大写或小写字母（`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，等价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。其他位均为 0。

霍尔瑞斯

接受的霍尔瑞斯数据格式为：

```
nH...          '...'H          "..."H
nL...          '...'L          "..."L
nR...          '...'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获得了广泛使用，但它们是非标准的 Fortran 95 格式。第二列中的种类数字可能会随供应商而发生变化。

表 4-2 数值数据类型的大小表示法

非标准	声明符	简短形式	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)	4 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 块中的默认最大对齐位置是 4 字节边界。

可使用特殊选项进行编译以更改默认的数据对齐和存储分配，如 -aligncommon、 -f、-dalign、 -dbl_align_all、-xmemalign 和 -xtypemap。本手册中的默认描述假定这些选项无效。

《Fortran 编程指南》的第 11 章提供了有关某种平台上数据类型和对齐方式特例的其他信息。

下表汇总了默认的大小和对齐，并忽略类型和选项的其他方面。

表 4-3 默认的数据大小和对齐 （以字节为单位）

Fortran 95 数据类型	大小	默认对齐	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

请注意以下事项：

- REAL*16 和 COMPLEX*32：正如表中的 8/16 所示，在 64 位环境 （使用 -xarch=v9 或 v9a 进行编译）中，默认对齐位置是 16 字节 （而非 8 字节）边界。数据类型“四倍精度”在 x86 平台上不可用。

- 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边界的字段相同。

选项 `-f` 或 `-dalign` 强制在 8 字节边界上对齐所有 8、16 或 32 字节数据。选项 `-dbl_align_all` 导致在 8 字节边界上对齐所有数据。依赖于这些选项如何使用的程序可能无法进行移植。

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` 语句使用以下格式：

```
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：标量变量名 （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 如果包含 *array_spec*，则它必须是显形 （常量或非常量边界）或假定大小。

示例：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`、 Cray 指针 `q` 及其指针对象 `c`。

示例：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 m+1` 维数组。

4.3.2 Cray 指针的用途

将变量与存储块中的特定位置动态关联起来后，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 SPARC V9 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，或者作为伪参数出现。
- 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
  - 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
- 作为伪参数的变量

- 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* 不能是类型为 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。

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 指针对象不能是伪参数。
- Cray 指针对象不能是函数值。
- Cray 指针对象不能是结构或结构组件。
- Cray 指针对象不能是派生类型。

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 是空白公共区中长度为 1000 的数组
- b 在 a 的后面，其长度为 n
- c 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，并且 blk 的每个整数元素为 4 字节长

4.4 STRUCTURE 和 UNION (VAX Fortran)

为了帮助从 f77 迁移程序， f95 接受 VAX Fortran STRUCTURE 和 UNION 语句，它们是 Fortran 95 中“派生类型”的前身。有关语法的详细信息，请参见《FORTRAN 77 语言参考》手册。

STRUCTURE 中的字段声明可以是以下内容之一：

- 子结构 6 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 f77 相同，不能将 POINTER 语句用作字段声明。

f95 还允许：

- '.' 或 '%' 可用作结构字段非关联化符号：
  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。
- 结构可以作为组件出现在派生类型中，但必须使用 SEQUENCE 属性声明派生类型。

4.5 无符号整数

Fortran 95 编译器接受新的数据类型 UNSIGNED 作为该语言的扩展。四种 KIND 参数值可以使用 UNSIGNED：1、2、4 和 8，分别对应于 1、2、4 和 8 字节无符号整数。

无符号整数常量的形式是以大写或小写字母 U 结尾的数字串，后面可以跟一个下划线和种类参数。以下的例子显示无符号整数常量的最大值：

```
255u_1
65535u_2
4294967295U_4
18446744073709551615U_8
```

如果没有使用种类参数表达 (12345U)，则默认值与默认整数相同。默认值为 U_4，但可以使用 -xtypemap 选项更改它，这将更改默认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数变量或数组：

```
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)。

  - 例外情况是：一个操作数是无符号整数，而另一个操作数是具有正数或零值的有符号整数常量表达式，则该结果是无符号整数。
  - 此类混合表达式的结果种类是操作数的最大种类。
- 有符号值的幂指数是有符号的值；而无符号值的幂指数是无符号的值。
- 无符号值的一元减法是无符号的值。
- 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

可以使用内在关系运算来比较有符号和无符号整数操作数。结果基于未修改的操作数值。

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 读取和写入它们。使用列表控制或名称列表 I/O 的无符号整数的写入格式与用于有符号正整数的格式相同。
- 还可以使用未格式化 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数和无符号整数作操作数，`MAX` 和 `MIN` 函数除外。仅当至少有一个操作数为 `REAL` 类型时，这两个函数才允许同时使用有符号整数和无符号整数作操作数。
- 无符号数组不能作为数组内部函数的参数出现。

4.6 Fortran 2003 功能

此发行版本的 `f95` 编译器中，引入了 Fortran 2003 标准中提出的一些新功能。

4.6.1 与 C 函数之间的互操作性

新的 Fortran 标准提供了以下内容：

- 一种引用 C 语言过程的方法（反过来说，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。

草案标准还引入了 `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。

此 Fortran 95 编译器版本实现了草案标准第 15 章中描述的那些功能。如标准中第 4 章中所述， 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 中新引入的关键字。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。详细信息，请参见 Fortran 2003 草案标准的第 14 章。

4.6.3 命令行参数内函数

Fortran 2003 标准引入了三个新的内部函数来处理命令行参数和环境变量。包括：

- `GET_COMMAND` (*command, length, status*)

  以命令返回调用该程序的整个命令行。

- `GET_COMMAND_ARGUMENT` (*number, value, length, status*)

以值返回命令行参数。

- 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 95 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 语句结合使用时，它允许编程人员指定可能与计算重叠的 I/O 进程。虽然编译器可以识别 ASYNCHRONOUS='YES'，但草案标准并不要求实际使用异步 I/O。在此版本的编译器中，I/O 总是同步的。

4.6.6 扩展的 ALLOCATABLE 属性

Fortran 2003 为 ALLOCATABLE 属性提供了数据实体扩展。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
- 伪数组
- 数组函数结果

在可分配实体可能与存储关联的所有地方，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出现在 SEQUENCE 类型中，但随后在 COMMON 和 EQUIVALENCE 中禁止此类型的对象。

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伪输入参数，则表明“按值”传递实际参数。以下示例说明将 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 95 子程序：

C 代码：

```
#include <stdlib.h>
int main(int ac, char *av[])
{
    to_fortran(2);
}
```

Fortran 代码：

```
       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连续的字节序列来处理，可由从正整数 1 开始进行寻址。可以连接数据文件以进行格式化或非格式化访问。

可以在 OPEN 语句中使用 ACCESS='STREAM' 说明符来声明流 I/O 文件。字节地址文件定位需要在 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 格式化 I/O 功能

在 f95 中，已实现了 3 个新的 Fortran 2003 格式化 I/O 说明符。它们可能出现在 OPEN、READ、WRITE、PRINT 和 INQUIRE 语句中：

- DECIMAL=['POINT'|'COMMA']

  更改默认的十进制编辑模式。对于使用 D、E、EN、ES、F 和 G 编辑格式的浮点数，默认使用句点来分隔整数和小数部分。'COMMA' 更改默认设置以使用逗号而不是句点来打印，例如，123,456。默认设置为 'POINT'，它使用句点来打印，例如，123.456。

- ROUND=['PROCESSOR_DEFINED' | 'COMPATIBLE']

为格式化 I/O D、 E、 EN、 ES、 F 和 G 编辑设置默认舍入模式。在使用 'COMPATIBLE' 时，数据转换后的数值结果是两个最相近的表示中更接近的一个，如果值正好在两者中间，则选择离 0 远的值。在使用 'PROCESSOR_DEFINED' 时，舍入模式取决于处理器的默认模式，如果未指定 ROUND，则为编译器默认设置。

例如，WRITE(*,'(f11.4)') 0.11115 在默认模式下打印 0.1111，而在 'COMPATIBLE' 模式下打印 0.1112。

- IOMSG=*character-variable*

  将错误消息作为字符串在指定字符变量中返回。这与标准输出中显示的错误消息相同。用户应该分配足够大的字符缓冲区以保存最长的消息。（CHARACTER*256 应该足够了。）

在 INQUIRE 语句中使用时，这些说明符声明一个字符变量以返回当前值。

新的编辑描述符 DP、DC、RP 和 RC 将单个 FORMAT 语句中的默认设置分别更改为小数点、小数逗号、处理器定义的舍入以及兼容的舍入。例如：

```
WRITE(*,'(I5,DC,F10.3)') N, W
```

在 F10.3 输出项目中打印逗号而不是句点。

要了解如何在格式化 I/O 中更改浮点舍入模式，另请参见 -iorounding 编译器命令行选项。（第 3-29 页的 "-iorounding[={compatible|processor-defined}]"。）

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 草案标准的一部分。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 95 编译器的一部分。

4.7.1 I/O 错误处理例程

新函数使用户可以为逻辑单元上的带格式输入指定自己的错误处理程序。当检测到格式错误时，运行时 I/O 库会调用特定的由用户提供的处理例程，同时将数据指向输入行中导致错误的字符。处理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后继续；或者采用默认的 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，并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 允许使用尖括号包含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 兼容性选项标记。

4.7.3 NAMELIST 输入格式

- 输入中的组名称前面可以是 $ 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 BACKSPACE a FORM='BINARY' 文件，这是因为不知道要退格到什么位置。在对 'BINARY' 文件执行 READ 操作时，将按所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到文件中，并按照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：在使用 FORM= 扒 INARYí 打开的文件中， INQUIRE 返回：

  ```
  FORM="BINARY"
  ACCESS="SEQUENTIAL"
  DIRECT="NO"
  FORMATTED="NO"
  UNFORMATTED="YES"
  RECL=  AND  NEXTREC= 没有定义
  ```

- 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递归的 I/O（这是因为 `f95` I/O 库为 "MT-Warm"）。
- `RECL=2147483646` (2^{31}-2) 是顺序格式化、列表控制和名称列表输出中的默认记录长度。
- 可以识别和实现 `ENCODE` 和 `DECODE`，详细信息，请参见《FORTRAN 77 语言参考手册》。
- 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**编译指示**。

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，但具有其他的字符，可将它识别为不仅仅是此编译器的注释。对于大多数其他编译器，将它处理为注释，因此具有一定的代码移植性。

Sun 风格的并行化指令是 `f95 -explicitpar` 的默认设置。要切换到 Cray 风格的指令，请使用 `-mp=cray` 编译器命令行标记。OpenMP 指令的显式并行化要求使用 `-openmp` 进行编译。

附录 D 提供了 Fortran 指令的完整摘要。

4.8.1 特殊 `f95` 指令行的格式

除了第 2 章中介绍的指令外，`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 （实际上是非空白）。
- 指令在空格后面列出。
- 在**初始**指令行中，紧靠 !DIR$ 后面的位置中为空格、制表符或换行符。
- 在**连续**指令行中，紧靠 !DIR$ 后面的位置中为空格、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 范围

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。可将指令插入在 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：FREE 指令。

```
!DIR$ FREE
       DO i = 1, n
           a(i) = b(i) * c(i)
       END DO
```

4.8.3 并行指令

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下一个 DO 循环。附录 D 和《Fortran 编程指南》中有关并行化的章节中介绍了这些指令。 `f95` 可识别 Sun 和 Cray 风格的并行化指令以及 OpenMP Fortran API 指令。《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，就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` 模块实现目标文件。可与模块实现目标文件以及所有其他目标文件链接在一起以创建可执行文件。

编译器在 `-moddir=`*dir* 标记或 `MODDIR` 环境变量指定的目录中创建模块接口文件和实现目标文件。如果没有指定，则编译器在当前的工作目录中写入 `.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。`-M`*path* 选项用于给编译器指定其他的搜索路径。必须在链接步骤的命令行中显式地列出模块实现目标文件。

通常，编程人员为每个文件定义一个 `MODULE`，并给 `MODULE` 和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中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：

```
main
main.o
one.mod
mod_one.o
two.mod
mod_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，编译器在当前的目录中搜索 `one.mod` 和 `two.mod`。编译在 `USE` 语句中引用这些模块的任何文件之前，必须先编译这些文件。链接步骤要求模块实现目标文件 `mod_one.o` 和 `mod_two.o` 与所有其他目标文件一起出现以创建可执行文件。

4.9.1 搜索模块

使用此发行版本的 7.0 版 Fortran 95 编译器时，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* 标记中显式地指定它。在默认情况下，编译器并不搜索归档文件。

而是仅搜索与 `USE` 语句中出现的名称相同的 `.mod` 文件。例如， Fortran 95 语句 `USE mymod` 导致编译器默认搜索模块文件 `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* 选项标记和 `MODDIR` 环境变量对它进行控制。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` 标记上列出的目录和文件。

4.9.2 `-use=`*list* 选项标记

`-use=`*list* 标记强制将一个或多个隐式的 `USE` 语句加入到每个使用该标记编译的子程序或模块子程序中。通过使用该标记，在库或应用程序的某个功能要求使用某个模块或模块文件时，不必修改源程序。

使用 `-use=`*module_name* 进行编译与将 `USE` *module_name* 添加到编译的每个子程序或模块中具有相同的效果。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

可使用 fdumpmod(1) 命令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、拼接 .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。显示包含模块名称、版本号、目标体系结构以及指示模块兼容的编译选项的标记。详细信息，请参见 fdumpmod(1) 手册页。

4.10 内部函数

f95 支持某些特定的内在过程，它们是超出标准的扩展。

表 4-4 非标准的内部函数

Name	定义	函数类型	参数类型	参数	说明
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
LEADZ	获取前导 0 的位数	整型	布尔、整型、实型或指针	([I=]*i*)	NP, I
POPCNT	获取设置位的数量	整型	布尔、整型、实型或指针	([I=]*i*)	NP, I
POPPAR	计算位总体奇偶性	整型	布尔、整型、实型或指针	([X=]*x*)	NP, I

有关上表的说明：

P	名称可以作为参数传递。
NP	名称不能作为参数传递。
E	在运行时调用内部函数的外部代码。
I	f95 为内在过程生成内联代码。

有关内部函数 （包括 Fortran 95 编译器可识别的 Fortran 77 内部函数）更完整的讨论，请参见 《Fortran 库参考》。

4.11 向前兼容性

将来的 `f95` 版本在源代码上将与该版本兼容。

不能保证此 `f95` 版本生成的模块信息文件与未来的版本兼容。

4.12 混合语言

在 Solaris 系统上，可以将使用 C 编写的例程与 Fortran 程序结合起来，因为这些语言具有相同的调用约定。有关 C 和 Fortran 例程如何进行交互操作的详细信息，请参见《Fortran 编程指南》中的“C-Fortran 接口”一章。

第 5 章

FORTRAN 77 兼容性：迁移到 Fortran 95

Fortran 95 编译器 `f95` 可编译大多数传统的 FORTRAN 77 程序，其中包括利用 `f77` 编译器以前编译的非标准扩展的程序。

`f95` 可直接接受很多的 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，并列出与 `f95` 不兼容的那些 `f77` 功能。有关 `f77` 编译器接受的任何非标准 FORTRAN 77 扩展的详细信息，请参见位于以下网址的《FORTRAN 77 语言参考》手册：
`http://docs.sun.com/source/806-3594/index.html`。

有关 `f95` 编译器接受的其他 Fortran 95 语言扩展的信息，请参见第 4 章。

`f95` 可编译符合标准的 FORTRAN 77 程序。为确保连续可移植性，利用非标准 FORTRAN 77 功能的程序应该迁移到符合标准的 Fortran 95 中。在使用 `-ansi` 选项进行编译时，程序中所有非标准的用法都将被标记出来。

5.1 兼容的 `f77` 功能

`f95` 直接接受或在 `-f77` 兼容性模式下编译时接受 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`]
  - 包含文件中的第一行可以是续行。 [`-f77=misc`}
  - 请使用 `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` 制表符格式不会对这些字符串进行填充直至第 72 列。 [`-f77`]

- **I/O：**
  - 在 Fortran 95 中，可以使用 ACCESS= 'APPEND' 打开文件。
  - 列表控制的输出使用与 f77 编译器类似的格式。 [-f77=output]
  - f95 允许在直接存取文件中使用 BACKSPACE，但不允许使用 ENDFILE。
  - f95 允许在格式编辑描述符中隐式地指定字段宽度。例如，允许使用 FORMAT(I)。
  - f95 可识别输出格式中的 f77 换码序列（例如，\n \t \'）。[-f77=backslash]
  - f95 可识别 OPEN 语句中的 FILEOPT=。
  - f95 允许使用 STATUS= 'KEEP' 来关闭 SCRATCH （临时）文件 [-f77]。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 （临时）文件。
  - 允许对内部文件使用直接 I/O。 [-f77]
  - f95 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。
  - 可使用 -f77=output 进行编译以写出 FORTRAN 77 风格的 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= 的直接存取 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控制的 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 (?)，就会响应正在读取的名称列表组的名称。 [-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，并且对其做与 SAVE 语句类似的处理。

- f95 接受 VAX STRUCTURE、 UNION 和 MAP 语句。（请参见第 4-11 页的 4.4 节“STRUCTURE 和 UNION (VAX Fortran)”）
- 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中，如 DO WHILE。 [-f77=logical]
- Cray 指针可以出现在内部函数调用中。
- f95 可接受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的 Cray 指针。
- f95 允许相同的 Cray 指针指向不同类型大小的项目（例如， REAL*8 和 INTEGER*4）。
- 在将 Cray 指针声明为 POINTER 的程序单元中，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 RETURN 语句中的表达式可以是非整数类型。 [-f77=misc}
- 具有 SAVE 属性的变量可以等价于 COMMON 块的元素。
- 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/
- 名称列表项目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。
- 可以使用与 f77 编译器兼容的方式等价处理不同的数据结构。 [-f77=misc}
- 与 f77 编译器类似， f95 允许很多内部函数出现在 PARAMETER 语句的初始化表达式中。
- 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）。
- 可以将一个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 自动以 f77 编译器的方式，将赋值、数据和参数语句中的 REAL 常量提升为适当的种类 （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。
- 在使用 -f77=misc 进行编译时，允许在类型声明中将 **kind* 放在变量名称后面。[-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 语句具备 *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 可以用于类型为 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 标记对执行间接递归 （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确工作。
  - 即使当伪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。
  - 在使用 -f77=misc 进行编译时，可以使用类型不是 INTEGER 或 REAL 的参数来定义 statement 函数；实际参数将被转换为 statement 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的参数。
  - f95 将函数 %LOC() 调用作为 LOC() 调用处理。[-f77=misc}
  - 允许在另一个运算符 （如 ** 或 *）后面使用一元加法和一元减法。
  - 即使当第一个参数为 COMPLEX 类型时，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。
  - 在当前目录以及 -I 选项中指定的目录中搜索 INCLUDE 文件。[-f77=misc}

- 允许进行连续的 .NOT. 运算，如 .NOT..NOT..NOT.(I.EQ.J)。[-f77=misc}

- **杂项**
  - 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 兼容性标记进行编译时，保留了这一约定。
  - f77 编译器编译的程序并不捕获算术异常，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。使用 -f77 标记进行的编译将模拟 f77 编译器的这种行为。在默认情况下，f95 编译器捕获遇到的第一个算术异常，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 标记进行编译时，如果将 REAL*4 常量与双精度或四精度操作数一起使用，则 f95 编译器允许该常量分别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
```

有关非标准语言扩展的语法和语义的详细信息，请参见位于以下网址的《FORTRAN 77 语言参考》：http://docs.sun.com/source/806-3594/index.html。

5.2 不兼容问题

下面列出了在使用此版本的 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 f77 中支持的 Fortran 77 非标准扩展，但在 f95 中不支持这些项目。

- **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 不允许对直接存取文件使用 ENDFILE。
  - f95 无法识别在直接存取 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 打开的直接存取文件不能用作 “流”文件。应使用 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，这将导致不同的结果。
  - 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)
... 错误："I2" 已经被用作函数，因此不得为其声明显式形状
DIMENSION 属性。

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 不支持的 f77 **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 f77 编译的例程链接

- 要混合使用 f77 和 f95 目标二进制文件，请使用带有 -xlang=f77 选项的 f95 编译进行链接。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 目标文件的 f95 主程序。

```
demo% cat m.f95
CHARACTER*74 ::c = 慣his is a test.í
   CALL echo1( c )
END
demo% f95 -xlang=f77 m.f95 sub77.o
demo%  a.out
 This is a test.
demo%
```

- 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 95 内部函数

Fortran 95 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 95 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 语句才能使 f95 使用您的例程而不是内在例程。

《Fortran 库参考》还列出了以前版本的 f77 编译器能够识别的所有内部函数。f95 编译器也将这些名称识别为内部函数。

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在 f77 编译器已知的函数范围内，而忽略 Fortran 95 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 f77_floatingpoint.h，在源程序中应按如下方式使用它：

  ```
  #include "floatingpoint.h"
  ```

- 格式为 f77/ 文件名的头文件引用应改为删除 f77/ 目录路径。
- 某些利用了非标准别名设置技术 （通过对数组进行重复索引或重叠 Cray 或 Fortran 指针）的程序可使用相应的 -xalias 标记进行编译。请参见第 3-50 页的 ″-xalias[=keywords]″。《Fortran 编程指南》中有关移植 “过时软件”程序的章节用示例讨论了这一问题。

附录 A

运行时错误消息

本附录介绍 Fortran 95 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。可在 intro(2) 中找到系统调用错误消息。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信息。以下是一个使用 Fortran 95 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 运行时系统  ******
 Error 1003: 格式化单元上的未格式化 I/O
 位置: 在 "wf.f" 的第一行中的 WRITE 语句
 单元:  6
 文件: 标准输出
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 列出了 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	格式错误
1001	非法单元编号
1002	未格式化单元上的格式化 I/O
1003	格式化单元上的未格式化 I/O
1004	顺序存取单元上的直接存取 I/O
1005	直接存取单元上的顺序存取 I/O
1006	设备不支持 BACKSPACE
1007	偏离记录开始
1008	无法 stat 文件
1009	重复计数后没有 *
1010	记录过长
1011	截断失败

表 A-1 f95 运行时 I/O 消息 （续）

错误	消息
1012	无法理解的列表输入
1013	可用空间不足
1014	单元未连接
1015	读取意外的字符
1016	非法逻辑输入字段
1017	‘新’文件存在
1018	无法找到 ‘旧’文件
1019	未知系统错误
1020	需要寻找能力
1021	非法参数
1022	负重复计数
1023	通道或设备非法操作
1024	可重入 I/O
1025	不兼容的说明符处于打开状态
1026	名称列表非法输入
1027	FILEOPT 参数中有错误
1028	不允许写入
1029	不允许读取
1030	输入时整数溢出
1031	输入时浮点溢出
1032	输入时浮点下溢
1051	默认的输入单元已关闭
1052	默认的输出单元已关闭
1053	来自未连接单元的直接存取 READ
1054	来自未连接单元的直接存取 WRITE
1055	无联系的内部单元
1056	对内部单元的空引用
1057	空内部文件
1058	未格式化单元上的列表控制 I/O
1059	未格式化单元上的名称列表 I/O

表 A-1 f95 运行时 I/O 消息 （续）

错误	消息
1060	试图在内部文件的结束之后写入
1061	无联系的 ADVANCE 说明符
1062	ADVANCE 说明符不是 'YES' 或 'NO'
1063	前进式输入中存在 EOR 说明符
1064	前进式输入中存在 SIZE 说明符
1065	记录编号为负数或零
1066	记录不在文件中
1067	被破坏的格式
1068	无联系的输入变量
1069	I/O 列表项多于数据编辑描述符
1070	在下标三元组中有零跨距
1071	在隐含 DO 循环中有零步长
1072	负的字段宽度
1073	零宽度字段
1074	输入时遇到了字符串编辑描述符
1075	输入时遇到了霍尔瑞斯编辑描述符
1076	在数字串中未发现数字
1077	在指数中未发现数字
1078	比例因子超出范围
1079	数字等于或超出基数
1080	在整数字段中有意外的字符
1081	在实数字段中有意外的字符
1082	在逻辑字段中有意外的字符
1083	在整数值中有意外的字符
1084	在实数值中有意外的字符
1085	在复数值中有意外的字符
1086	在逻辑值中有意外的字符
1087	在字符值中有意外的字符
1088	在 NAMELIST 组名前有意外的字符
1089	NAMELIST 组名不匹配程序中的名称

表 A-1 f95 运行时 I/O 消息 （续）

错误	消息
1090	在 NAMELIST 项中有意外的字符
1091	在 NAMELIST 项名中有不配对的括号
1092	变量不在 NAMELIST 组中
1093	NAMELIST 对象名中的下标过多
1094	NAMELIST 对象名中的下标不够
1095	NAMELIST 对象名中有零跨距
1096	NAMELIST 对象名中有空段下标
1097	NAMELIST 对象名中的下标超出界限
1098	NAMELIST 对象名中有空子串
1099	NAMELIST 对象名中的子串超出范围
1100	NAMELIST 对象名中有意外的组件名
1111	无联系的 ACCESS 说明符
1112	无联系的 ACTION 说明符
1113	无联系的 BINARY 说明符
1114	无联系的 BLANK 说明符
1115	无联系的 DELIM 说明符
1116	无联系的 DIRECT 说明符
1117	无联系的 FILE 说明符
1118	无联系的 FMT 说明符
1119	无联系的 FORM 说明符
1120	无联系的 FORMATTED 说明符
1121	无联系的 NAME 说明符
1122	无联系的 PAD 说明符
1123	无联系的 POSITION 说明符
1124	无联系的 READ 说明符
1125	无联系的 READWRITE 说明符
1126	无联系的 SEQUENTIAL 说明符
1127	无联系的 STATUS 说明符
1128	无联系的 UNFORMATTED 说明符
1129	无联系的 WRITE 说明符

表 A-1 f95 运行时 I/O 消息 （续）

错误	消息
1130	零长度的文件名
1131	ACCESS 说明符不是 'SEQUENTIAL' 或 'DIRECT'
1132	ACTION 说明符不是 'READ'、'WRITE' 或 'READWRITE'
1133	BLANK 说明符不是 'ZERO' 或 'NULL'
1134	DELIM 说明符不是 'APOSTROPHE'、'QUOTE' 或 'NONE'
1135	意外的 FORM 说明符
1136	PAD 说明符不是 'YES' 或 'NO'
1137	POSITION 说明符不是 'APPEND'、'ASIS' 或 'REWIND'
1138	RECL 说明符为零或负数
1139	直接存取文件未指定记录长度
1140	意外的 STATUS 说明符
1141	连接的单元指定了状态且不是 'OLD'
1142	STATUS 说明符不是 'KEEP' 或 'DELETE'
1143	为临时文件指定了状态 'KEEP'
1144	状态值不可能
1145	为临时文件指定了文件名
1146	正在试图打开正被读取或写入的单元
1147	正在试图关闭正被读取或写入的单元
1148	正在试图打开目录
1149	状态为 'OLD' 且文件为不存在的符号连接
1150	状态为 'NEW' 且文件为符号连接
1151	没有可用的临时文件名
1152	默认单元的说明符 ACCESS='STREAM'
1153	对默认单元的流存取
1161	设备不支持 REWIND
1162	BACKSPACE 需要读取许可
1163	直接存取单元上的 BACKSPACE
1164	二进制单元上的 BACKSPACE
1165	退格时遇到了文件结束
1166	ENDFILE 需要写入许可

表 A-1 f95 运行时 I/O 消息 （续）

错误	消息
1167	直接存取单元上的 ENDFILE
1168	对顺序或直接存取单元的流存取
1169	对未连接单元的流存取
1170	对流存取单元的直接存取
1171	POS 说明符的值不正确
1172	无联系的 ASYNCHRONOUS 说明符
1173	无联系的 DECIMAL 说明符
1174	无联系的 IOMSG 说明符
1175	无联系的 ROUND 说明符
1176	无联系的 STREAM 说明符
1177	ASYNCHRONOUS 说明符不是 'YES' 或 'NO'
1178	ROUND 说明符不是 'UP'、'DOWN'、'ZERO'、'NEAREST'、'COMPATIBLE' 或 'PROCESSOR-DEFINED'
1179	DECIMAL 说明符不是 'POINT' 或 'COMMA'
1180	流存取单元的 OPEN 语句中不允许 RECL 说明符
1181	正在试图分配一个已分配的数组
1182	正在释放一个无联系的指针
1183	正在释放一个未分配的可分配数组
1184	正在通过指针来释放一个可分配数组
1185	正在释放一个未由 ALLOCATE 语句分配的对象
1186	正在释放一个对象的一部分
1187	正在释放的对象比分配的大
1191	未分配的数组被传递给数组内函数
1192	非法等级
1193	源大小较小
1194	数组大小为零
1195	形状中有负元素
1196	非法种类
1197	非整合的数组
1213	未连接单元上的不同步 I/O
1214	同步单元上的不同步 I/O

表 A-1 f95 运行时 I/O 消息 （续）

错误	消息
1215	数据编辑描述符与 I/O 列表项类型不兼容
1216	当前 I/O 列表项不匹配任何数据编辑描述符
2001	无效的常数、结构或组件名
2002	未创建句柄
2003	字符参数太短
2004	数组参数太长或太短
2005	文件、记录或目录流结束
2021	锁定未初始化 (OpenMP)
2122	使用锁定变量时死锁 (OpenMP)
2123	未设锁定 (OpenMP)

附录 B

功能版本历史

本附录列出了此版本和以前版本的 Fortran 95 编译器中的新增功能和经修改的功能。

Fortran 95 编译器版本 8.2 是随 Sun Studio 11 发布的一个组件。

B.1 Sun Studio 11 Fortran 发行版本

- `-xmodel` **的新选项：**
  通过使用新的 -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。如果全局和静态变量的大小超过了 2 千兆字节，请指定 `-xmodel=medium`。否则，请使用 `-xmodel=small` 默认设置。请参见第 3-76 页的 "`-xmodel=[small | kernel | medium]`"。
- **为 x86 SSE2 平台扩展的** `-xvector` **选项：通过使用**
  `-xvector` 选项，可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，该选项在 x86 SSE2 平台上提供了扩展语法。请参见第 3-89 页的 "`-xvector[=[{yes|no}] [[no%]lib, [no%]simd, %none]]`"。
- 增强的 `STACKSIZE` **环境变量**：
  `STACKSIZE` 环境变量的语法已增强，可以包含单位关键字。
- 在 x86 平台上可用的 `-xpagesize` **选项：**
  现在，在 x86 平台以及 SPARC 平台上启用了 `-xpagesize`、`-xpagesize_heap` 和 `-xpagesize_stack` 选项。请参见第 3-77 页的 "`-xpagesize=size`"。
- **启用了新的 UltraSPARC T1 和 UltraSPARC IV+ 目标：**
  `-xarch`、`-xchip`、`-xcache` 和 `-xtarget` 值支持新的 UltraSPARC 处理器。请参见第 3-86 页的 "`-xtarget=t`"。

B.2 Sun Studio 10 Fortran 发行版本：

- **为 AMD-64 处理器进行编译**

  此版本引入了 `-xarch=amd64` 和 `-xtarget=opteron` 以编译在 64 位 x86 平台上运行的应用程序。

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

  新的编译器标志 `-xfilebyteorder` 提供了跨平台的二进制 I/O 文件支持。

- **Solaris 操作系统 x86 平台上可用的 OpenMP**

  对于此 Sun Studio 发行版本，可以使用 OpenMP API 在 Solaris x86 平台以及 Solaris SPARC 平台上实现共享内存并行操作。这两种平台现在都启用相同的功能。

- **不再支持 OpenMP 选项** `-openmp=stubs`

  OpenMP “存根”库是为了方便用户而提供的。要编译调用 OpenMP 库函数但忽略 OpenMP pragma 的 OpenMP 程序，请使用 -openmp 选项编译该程序，并使用 libompstubs.a 库链接对象文件。例如：

  ```
  % f95 omp_ignore.c -lompstubs
  ```

  不支持同时使用 `libompstubs.a` 和 OpenMP 运行时库 `libmtsk.so` 进行链接，这种链接方式可能会导致出现意外行为。

B.3 Sun Studio 9 Fortran 发行版本：

- **在 x86 Solaris 平台发行的 Fortran 95 编译器：**

  此发行版本的 Sun Studio 使 Fortran 95 编译器可在 Solaris OS x86 平台上使用。使用值为 `generic`、`native`、`386`、`486`、`pentium`、`pentium_pro`、`pentium3` 或 `pentium4` 的 `-xtarget` 进行编译，以便在 Solaris x86 平台上生成可执行文件。x86 平台上的默认值为 `-xtarget=generic`。

  以下 `f95` 功能尚未在 x86 平台上实现，只能在 SPARC 平台上使用：

  - 区间运算 （编译器选项 `-xia` 和 `-xinterval`）
  - 四倍 （128 位）运算 （例如 REAL*16）
  - IEEE 内模块 IEEE_EXCEPTIONS、 IEEE_ARITHMETIC 和 IEEE_FEATURES
  - `sun_io_handler` 模块
  - 并行化选项 （如 `-autopar`、 `-parallel`、 `-explitipar` 和 `-openmp`）。

  以下 `f95` 命令行选项只能在 x86 平台上使用，不能在 SPARC 平台上使用：
  `-fprecision`、 `-fstore`、 `-nofstore`

以下 f95 命令行选项只能在 SPARC 平台上使用，不能在 x86 平台上使用：-xcode、-xmemalign、-xprefetch、-xcheck、-xia、-xinterval、-xipo、-xjobs、-xlang、-xlinkopt、-xloopinfo、-xpagesize、-xprofile_ircache、-xreduction、-xvector、-depend、-openmp、-parallel、-autopar、-explicitpar、-vpara、-XlistMP。
同时，在 x86 平台上，-fast 增加了 -nofstore。

- **提高了运行时性能：**

  采用此发行版本后，大多数应用程序的运行时性能应有显著提高。为获得最佳效果，请用较高的优化级别 -xO4 或 -xO5 进行编译。在这些优化级别上，编译器现在可以内联包含的过程，以及那些带假定形式参数、可分配参数或指针参数的过程。

- **Fortran 2003 命令行内部函数：**

  Fortran 2003 标准草案引入了三个新的内部函数，来处理命令行参数和环境变量。这三个函数已在本发行版本的 f95 编译器中实现。新增的内部函数包括：

  - GET_COMMAND(*command, length, status*)

    以命令形式返回调用该程序的整个命令行。

  - GET_COMMAND_ARGUMENT(*number, value, length, status*)

    以值的形式返回命令行参数。

  - GET_ENVIRONMENT_VARIABLE(*name, value, length, status, trim_name*)

    返回环境变量的值。

- **新增和更改的命令行选项：**

  以下 f95 命令行选项是此发行版本中新增的选项：请参见第 3 章了解详细信息。

  - -xipo_archive={ none | readonly | writeback }

    允许交叉文件优化包括归档库 (.a)。（仅适用于 SPARC）

  - -xprefetch_auto_type=[no%]indirect_array_access

    为间接访问的数组生成间接预取。（仅适用于 SPARC）

  - -xprofile_pathmap=*collect_prefix*:*use_prefix*

    设置性能分析数据的路径映射。如果分析的目录不是以前采用 -xprofile=collect 进行编译时使用的目录，请使用 -xprofile_pathmap 选项和 -xprofile=use 选项。

  在此发行版本的 f95 中，以下命令行选项默认值已更改。

  - -xprefetch 的默认值为 -xprefetch=no%auto,explicit。
  - -xmemalign 的默认值为 -xmemalign=8i；在使用其中一个 -xarch=v9 选项进行编译时，默认值为 -xmemalign=8f。
  - 在使用其中一个 -xarch=v9 选项进行编译时，-xcode 的默认值为 abs44。

    要使用以前的编译器版本中使用的默认值进行编译，请明确指定以下选项：

要进行 32 位编译，请指定 -xarch=v8 -xmemalign=4s -xprefetch=no
要进行 64 位编译，请指定
-xcode=abs64 -xprefetch=no

- **默认 SPARC 架构是 V8PLUS：**

  默认 SPARC 架构不再是 V7。此版本的 Sun Studio 9 对 -xarch=v7 的支持有限。新的默认值为 V8PLUS (UltraSPARC)。使用 -xarch=v7 编译将被视作使用 -xarch=v8 编译，因为 Solaris 8 OS 只支持 -xarch=v8 或更高版本。

  要在 SPARC V8 系统（如 SPARCStation 10）上进行部署，请明确指定使用 -xarch=v8 编译。提供的系统库将在 SPARC V8 架构上运行。

  要在 SPARC V7 系统（如 SPARCStation 1）上进行部署，请明确指定使用 -xarch=v7 编译。提供的系统库将使用 SPARC V8 指令集。对于 Sun Studio 9 发行版，只有 Solaris 8 OS 支持 SPARC V7 架构。如果遇到 SPARC V8 指令，该 OS 将在软件中解释该指令。程序将会运行，但性能会下降。

- **OpenMP：增大了最大线程数：**

  OMP_NUM_THREADS 和多任务库的最大线程数量已从 128 增至 256。

- **OpenMP：自动设置变量的作用域：**

  在此版本的 Fortran 95 编译器中，针对共享内存并行编程的 OpenMP API 实现能够设置并行区域中的变量作用域。有关详情，请参见《OpenMP API 用户指南》。（此版本只在 SPARC 平台上实现了 OpenMP。）

B.4 Sun Studio 8 Fortran 发行版本：

- **增强的 -openmp 选项：**

  -openmp 选项标记经过增强，便于调试 OpenMP 程序。要使用 dbx 调试 OpenMP 应用程序，请使用以下编译选项：

  ```
  -openmp=noopt -g
  ```

  然后便可以使用 dbx 在并行区内设置断点，并显示变量的内容。请参见第 3-37 页的 "-openmp[={parallel|noopt|none}]"。

- **多进程编译：**

  带 -xipo 指定 -xjobs=*n*，过程间优化器调用至多 *n* 个代码生成器实例来编译命令行所列的文件。该选项可大大缩短在多 CPU 机器上生成较大应用程序所需的时间。请参见第 3-71 页的 "-xjobs=n"。

- **使用 PRAGMA ASSUME 作断言：**

  ASSUME pragma 是本发行版本编译器的新特性。该 pragma 提示编译器在编译过程中某些点上的条件编程人员确信为真。这有助于编译器更好地优化代码。编程人员也可以使用断言在程序运行时检查程序的合理性。请参见第 2-12 页的 2.3.1.9 节 “ASSUME 指令” 和第 3-57 页的 "-xassume_control[=keywords]"。

- **更多 Fortran 2000 功能：**

  在此版本的 Fortran 95 编译器中，已经实现了 Fortran 2000 草案标准中的以下功能。第 4 章中描述了这些功能。

  - 异常和 IEEE 运算：

    新增的内部模块 IEEE_ARITHMETIC 和 IEEE_FEATURES 在 Fortran 语言中提供了对异常和 IEEE 运算的支持。请参见第 4-14 页的 4.6.2 节“IEEE 浮点异常处理”。

  - 与 C 之间的互操作性：

    新的 Fortran 草案标准提供了一种引用 C 语言过程的方法，以及一种指定 Fortran 子程序可从 C 函数引用的方法。它还提供了一种声明与外部 C 变量链接的全局变量的方法。请参见第 4-14 页的 4.6.1 节 “与 C 函数之间的互操作性”。

  - `PROTECTED` 属性

    现在，Fortran 95 编译器接受 Fortran 2000 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。第 4-15 页的 4.6.4 节 “`PROTECTED` 属性” .

  - `ASYNCHRONOUS` I/O 说明符

    编译器可识别 I/O 语句中的 `ASYNCHRONOUS` 说明符：

    ```
    ASYNCHRONOUS=['YES' | 'NO']
    ```

    此语法是在草案标准中提出的。请参见第 4-15 页的 4.6.5 节“Fortran 2003 异步 I/O”。

- **与传统 `f77` 之间的兼容性更强：**

  很多新增功能增强了 Fortran 95 编译器与传统 Fortran 77 编译器 `f77` 之间的兼容性。其中包括变量格式表达式 (VFE)、长标识符、 `-arg=local` 和 `-vax` 编译器选项。请参见第 3 章和第 4 章。

- **I/O 错误处理程序：**

  新函数使用户可以为逻辑单元上的带格式输入指定自己的错误处理程序。第 4-17 页的 4.7.1 节“I/O 错误处理例程”、手册页和《Fortran 库参考》中描述了这些例程。

- **无符号整数：**

  在本发行版本中， Fortran 95 编译器接受一种新的数据类型：`UNSIGNED`，作为对语言的一种扩展。请参见第 4-12 页的 4.5 节 “无符号整数”。

- **设置首选堆栈 / 堆页面大小：**

  新的命令行选项 `-xpagesize` 允许正在运行的程序在程序启动时设置首选堆栈和堆的页面大小。请参见第 3-77 页的 ″`-xpagesize=size`″。

- **更快和增强的文件配置：**

  此版本引入了新的命令行选项 `-xprofile_ircache=` *path*，可加速文件配置反馈过程中的“use”编译阶段。请参见第 3-83 页的 ″`-xprofile_ircache[=path]`″。另请参见第 3-83 页的 ″`-xprofile_pathmap=collect_prefix:use_prefix`″。

- **增强的 “已知库”：**

  -xknown_lib 选项得到了增强，以便包括更多基本线性代数库 (BLAS) 中的例程。请参见第 3-72 页的 ″-xknown_lib=library_list″。

- **链接时优化：**

  使用新的 -xlinkopt 标记进行编译和链接，以便在链接时调用后优化器对生成的二进制对象代码进行一些高级性能优化。请参见第 3-73 页的 ″-xlinkopt[={1|2|0}]″。

- **局部变量的初始化：**

  这是对 -xcheck 选项标志新的扩展，启用了特殊的局部变量初始化。在使用 -xcheck=init_local 进行编译时，如果在程序对局部变量进行赋值前使用了该变量，则编译器会给该变量初始化一个容易导致算术异常的值。请参见第 3-59 页的 ″-xcheck=keyword″。

B.5 Sun ONE Studio 7 编译器集合 (Forte Developer 7) 版本：

- **Fortran 95 编译器中包含的 Fortran 77 功能**

  此 Forte Developer 软件版本用 `f95` 编译器替代了 `f77` 编译器，并增加了一些功能。 `f77` 命令是一个调用 `f95` 编译器的脚本：

  命令：
  `f77` *options files libraries*
  变成对 `f95` 编译器的调用：
  `f95 -f77=%all -ftrap=%none` *options files* `-lf77compat` *libraries*

  有关 Fortran 77 兼容性和不兼容性问题的详细信息，请参见第 5 章。

- **Fortran 77 兼容性模式：**

  新的 `-f77` 标记选择各种不同的兼容性功能，使编译器能够接受很多通常与 Fortran 95 不兼容的 Fortran 77 构造和约定。请参见第 3-19 页的 "`-f77`[=list]" 和第 5 章。

- **编译使用非标准别名的 “过时软件” 程序：**

  `f95` 编译器必须假定它所编译的程序符合 Fortran 95 有关通过子程序调用、全局变量、指针和重复索引来设置变量别名的标准。很多 ‘过时软件’ 传统程序有意识地利用别名设置技术解决以前版本的 Fortran 语言中存在的缺点。请使用新的 `-xalias` 标记向编译器建议：程序与标准有多大偏差，以及预计会出现哪些别名问题。在某些情况下，只有在指定了正确的 `-xalias` 子选项时，编译器才能生成正确的代码。严格遵循标准的程序可以建议编译器不必考虑别名问题，从而可使性能得到一定的提高。请参见第 3-50 页的 "`-xalias`[=keywords]" 以及《Fortran 编程指南》中有关移植的章节。

- **增强的 MODULE 功能：**
  - 新标记 `-use=`*list* 强制在每个子程序中使用一个或多个隐式的 `USE` 语句。请参见第 3-46 页的 "`-use=`list"。
  - 新标记 `-moddir=`*path* 控制编译器在哪里写入已编译的 `MODULE` 子程序 （`.mod` 文件）。请参见第 3-32 页的 "`-moddir=`path"。新环境变量 `MODDIR` 也可以控制写入 .mod 文件的位置。
  - -M*path* 标记现在可以接受目录路径、归档 (`.a`) 文件或模块 (`.mod`) 文件用于搜索 `MODULE` 子程序。编译器通过检查其内容来确定文件的类型；实际的文件扩展名会被忽略。请参见第 3-31 页的 "`-M`path"。
  - 现在，编译器在搜索模块时，先在写入模块文件的目录中查找。

  详细信息，请参见第 4-21 页的 4.9 节 “模块文件”。

- **使用 -Xlist 执行更强的全局程序分析：**

此 f95 编译器版本将一些新的检查添加到 -Xlist 标记提供的全局程序分析中。新的 -XlistMP 子选项在静态程序分析方面开创了一个新的领域：OpenMP 并行化指令验证。详细信息，请参见第 3-49 页的 "-Xlist[x]"、Forte Developer《OpenMP API 用户指南》，以及 《Fortran 编程指南》中有关程序分析和调试的章节。

- **使用 -xknown_lib=library 标识已知库：**

  新选项 -xknown_lib=*library* 指示编译器将对某些已知库的引用作为内部函数处理，并忽略任何用户提供的版本。这样，编译器就可以基于其对库的特殊了解来对库调用进行优化。在此版本中，已知库名称仅限于 blas 和 intrinsics，前者代表 Sun Performance Library 中 BLAS 例程的一个子集，后者则用于忽略 Fortran 95 标准内部函数以及任何由用户提供的此类例程版本的显式 EXTERNAL 声明。请参见第 3-72 页的 "-xknown_lib=library_list"。

- **忽略接口中的伪参数类型：**

  新指令 !$PRAGMA IGNORE_TKR {*list_of_variables*} 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伪参数名称的类型、种类和等级。对于基于参数类型、种类或等级调用特定库例程的包装器来说，使用此指令可大大简化为其编写通用接口的工作。有关详细信息，请参见第 2-9 页的 2.3.1.2 节 "IGNORE_TKR 指令"。

- **增强的 -C 运行时数组检查：**

  在此 f95 编译器版本中，使用 -C 选项进行的运行时数组下标范围检查得到了增强，以便包含数组一致性检查。如果在数组段不一致的地方执行数组语法语句，就会导致运行时错误。请参见第 3-12 页的 "-C"。

- **引入 Fortran 2000 功能：**

  在此版本的 f95 中，实现了某些为下一个 Fortran 标准而建议的新的格式化 I/O 功能。这些功能是 DECIMAL=、 ROUND= 和 IOMSG= 说明符，它们可以出现在 OPEN、READ、WRITE、PRINT 和 INQUIRE 语句中。另外，还实现了 DP、DC、RP 和 RC 编辑描述符。详细信息，请参见第 4-16 页的 4.6.9 节 "Fortran 2003 格式化 I/O 功能"。

- **格式化 I/O 中的舍入：**

  新的选项标记 -iorounding 为格式化 I/O 设置默认舍入模式。这些模式 （处理器定义的或兼容的）与作为 Fortran 2000 功能一部分而实现的 ROUND= 说明符相对应。请参见第 3-29 页的 "-iorounding[={compatible|processor-defined}]"。

- **删除的过时标记：**

  以下标记已从 f95 命令行中删除：

  ```
  -db     -dbl
  ```

  f95 编译器中没有实现以下 f77 编译器标记，因此这些标记也被视为过时：

  ```
  -arg=local -i2 -i4 -misalign -oldldo -r8 -vax
  -xl -xvpara -xtypemap=integer:mixed
  ```

- **堆栈溢出检查：**

在使用新的 -xcheck=stkovf 标记进行编译时，将在子程序入口添加对堆栈溢出条件的运行时检查。如果检测到堆栈溢出，就会出现 SIGSEGV 段错误。对于在堆栈中分配了较大数组的多线程应用程序，其中的堆栈溢出可能导致相邻线程堆栈中出现隐性数据破坏。如果怀疑存在栈溢出，请使用 -xcheck=stkovf 编译所有例程。请参见第 3-59 页的 "-xcheck=keyword"。

- **新的默认线程堆栈大小：**

  在此版本中，SPARC V8 平台上的默认从属线程堆栈大小已增加到 4 MB；而 SPARC V9 平台则增加到 8 MB。详细信息，请参见《Fortran 编程指南》的“并行化”一章中有关堆栈和堆栈大小的讨论。

- **增强的过程间优化：**

  编译器可使用 -xipo=1 跨所有源文件执行内联操作。此版本增加了 -xipo=2，用于增强过程间别名分析和内存分配以及布局优化以便改善缓存性能。请参见第 3-69 页的 "-xipo[={0|1|2}]"。

- **使用 -xprefetch_level=n 控制预取指令：**

  通过使用新标记 -xprefetch_level=*n* 来控制 -xprefetch=auto 的预取指令自动插入。使用时要求采用 -xO3 或更高的优化级别以及支持预取的目标平台（-xarch 平台 v8plus、v8plusa、v8plusb、v9、v9a、v9b、generic64 或 native64）。请参见第 3-81 页的 "-xprefetch_level={1|2|3}"。

您可以在 http://docs.sun.com 网站上的早期版本文档集中找到早于 Forte Developer 7 的各版本的功能历史记录。

附录 C

传统 -xtarget 平台扩展

本附录详细描述了较旧的和不常用的 -xtarget 选项平台系统名称及其扩展。它们在此仅用于参考。 UltraSPARC 平台的值是按照第 3 章中的 -xtarget 选项描述指定的。较新版本的 Solaris 操作环境可能不再支持此处列出的某些系统平台。

正如下表中所示，每个特定的 -xtarget 值可扩展为 -xarch、-xchip 和 -xcache 选项的一组特定的值。可在任何系统上运行 fpversion(1) 以确定目标定义。

例如：

```
-xtarget=sun4/15
```

表示

```
-xarch=v8a  -xchip=micro  -xcache=2/16/1
```

表 C-1 传统 -xtarget 扩展

-xtarget=	-xarch	-xchip	-xcache
cs6400	v8	super	16/32/4:2048/64/1
sc2000	v8	super	16/32/4:2048/64/1
solb5	v7	old	128/32/1
solb6	v8	super	16/32/4:1024/32/1
ss1	v7	old	64/16/1
ss10	v8	super	16/32/4
ss10/20	v8	super	16/32/4
ss10/30	v8	super	16/32/4
ss10/40	v8	super	16/32/4
ss10/402	v8	super	16/32/4
ss10/41	v8	super	16/32/4:1024/32/1
ss10/412	v8	super	16/32/4:1024/32/1
ss10/50	v8	super	16/32/4

表 C-1 传统 -xtarget 扩展 （续）

-xtarget=	-xarch	-xchip	-xcache
ss10/51	v8	super	16/32/4:1024/32/1
ss10/512	v8	super	16/32/4:1024/32/1
ss10/514	v8	super	16/32/4:1024/32/1
ss10/61	v8	super	16/32/4:1024/32/1
ss10/612	v8	super	16/32/4:1024/32/1
ss10/71	v8	super2	16/32/4:1024/32/1
ss10/712	v8	super2	16/32/4:1024/32/1
ss10/hs11	v8	hyper	256/64/1
ss10/hs12	v8	hyper	256/64/1
ss10/hs14	v8	hyper	256/64/1
ss10/hs21	v8	hyper	256/64/1
ss10/hs22	v8	hyper	256/64/1
ss1000	v8	super	16/32/4:1024/32/1
ss1plus	v7	old	64/16/1
ss2	v7	old	64/32/1
ss20	v8	super	16/32/4:1024/32/1
ss20/151	v8	hyper	512/64/1
ss20/152	v8	hyper	512/64/1
ss20/50	v8	super	16/32/4
ss20/502	v8	super	16/32/4
ss20/51	v8	super	16/32/4:1024/32/1
ss20/512	v8	super	16/32/4:1024/32/1
ss20/514	v8	super	16/32/4:1024/32/1
ss20/61	v8	super	16/32/4:1024/32/1
ss20/612	v8	super	16/32/4:1024/32/1
ss20/71	v8	super2	16/32/4:1024/32/1
ss20/712	v8	super2	16/32/4:1024/32/1
ss20/hs11	v8	hyper	256/64/1
ss20/hs12	v8	hyper	256/64/1
ss20/hs14	v8	hyper	256/64/1

表 C-1 传统 -xtarget 扩展（续）

-xtarget=	-xarch	-xchip	-xcache
ss20/hs21	v8	hyper	256/64/1
ss20/hs22	v8	hyper	256/64/1
ss2p	v7	powerup	64/32/1
ss4	v8a	micro2	8/16/1
ss4/110	v8a	micro2	8/16/1
ss4/85	v8a	micro2	8/16/1
ss5	v8a	micro2	8/16/1
ss5/110	v8a	micro2	8/16/1
ss5/85	v8a	micro2	8/16/1
ss600/120	v7	old	64/32/1
ss600/140	v7	old	64/32/1
ss600/41	v8	super	16/32/4:1024/32/1
ss600/412	v8	super	16/32/4:1024/32/1
ss600/51	v8	super	16/32/4:1024/32/1
ss600/512	v8	super	16/32/4:1024/32/1
ss600/514	v8	super	16/32/4:1024/32/1
ss600/61	v8	super	16/32/4:1024/32/1
ss600/612	v8	super	16/32/4:1024/32/1
sselc	v7	old	64/32/1
ssipc	v7	old	64/16/1
ssipx	v7	old	64/32/1
sslc	v8a	micro	2/16/1
sslt	v7	old	64/32/1
sslx	v8a	micro	2/16/1
sslx2	v8a	micro2	8/16/1
ssslc	v7	old	64/16/1
ssvyger	v8a	micro2	8/16/1
sun4/110	v7	old	2/16/1
sun4/15	v8a	micro	2/16/1
sun4/150	v7	old	2/16/1

表 C-1 传统 -xtarget 扩展 （续）

-xtarget=	-xarch	-xchip	-xcache
sun4/20	v7	old	64/16/1
sun4/25	v7	old	64/32/1
sun4/260	v7	old	128/16/1
sun4/280	v7	old	128/16/1
sun4/30	v8a	micro	2/16/1
sun4/330	v7	old	128/16/1
sun4/370	v7	old	128/16/1
sun4/390	v7	old	128/16/1
sun4/40	v7	old	64/16/1
sun4/470	v7	old	128/32/1
sun4/490	v7	old	128/32/1
sun4/50	v7	old	64/32/1
sun4/60	v7	old	64/16/1
sun4/630	v7	old	64/32/1
sun4/65	v7	old	64/16/1
sun4/670	v7	old	64/32/1
sun4/690	v7	old	64/32/1
sun4/75	v7	old	64/32/1

附录 D

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

D.1 通用 Fortran 指令

第 2 章中描述了 f95 接受的通用指令。

表 D-1 通用 Fortran 指令摘要

格式	
C$PRAGMA *keyword* (a [, a] ...) [, *keyword* (a [, a] ...)] , ... C$PRAGMA SUN *keyword* (a [, a] ...) [, *keyword* (a [, a] ...)] , ... C$PRAGMA SPARC *keyword* (a [, a] ...) [, *keyword* (a [, a] ...)] , ...	
第一列中的注释指示符可能是 c、C、! 或 *。（在本示例中使用 C。f95 自由格式必须使用 !。）	
C 指令	C$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	C$PRAGMA IGNORE_TKR {*name* {, *name*} ...} 在解析特定调用时，编译器会忽略在通用过程接口中出现的指定伪参数名称的类型、种类和等级。
UNROLL 指令	C$PRAGMA SUN UNROLL=*n* 建议编译器将以下的循环解开为指定的长度 *n*。

表 D-1 通用 Fortran 指令摘要 （续）

WEAK 指令	C$PRAGMA WEAK(*name*[=*name2*]) 将 *name* 声明为弱符号，或者声明为 *name2* 的别名。
OPT 指令	C$PRAGMA SUN OPT=*n* 将子程序的优化级别设置为 *n*。
NOMEMDEP 指令	C$PRAGMA SUN NOMEMDEP 断言以下的循环中没有内存依存关系。 （要求使用 -parallel 或 -explicitpar。）
PIPELOOP 指令	C$PRAGMA SUN PIPELOOP=*n* 断言循环在迭代 *n* 前后两部分之间存在依存关系。
PREFETCH 指令	C$PRAGMA SPARC_PREFETCH_READ_ONCE (*name*) C$PRAGMA SPARC_PREFETCH_READ_MANY (*name*) C$PRAGMA SPARC_PREFETCH_WRITE_ONCE (*name*) C$PRAGMA SPARC_PREFETCH_WRITE_MANY(*name*) 请求编译器为名称引用生成预取指令。（要求使用 -xprefetch 选项。）
ASSUME 指令	C$PRAGMA [BEGIN} ASSUME (*expression* [,*probability*]) C$PRAGMA END ASSUME 断言编译器可认为程序中某些点的条件为真。

D.2 特殊的 Fortran 95 指令

以下指令仅用于 f95。有关详细信息，请参见第 4-20 页的 4.8.2 节“FIXED 和 FREE 指令”。

表 D-2 特殊的 Fortran 95 指令

格式	!DIR$ *directive* 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 ... ；该行必须从第一列开始。 对于自由格式源代码，该行可以以空格开头。
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的其余部分，或者在遇到下一个 FREE 或 FIXED 指令之前的部分。

D.3 Fortran 95 OpenMP 指令

Sun Fortran 95 编译器支持 OpenMP 2.5 Fortran API。-openmp 编译器标记允许使用这些指令。（请参见第 3-37 页的 "-openmp[={parallel|noopt|none}]"）。

有关完整的详细信息，请参见 《OpenMP API 用户指南》。

D.4 Sun 并行化指令

注 – 传统的 Sun 和 Cray 并行化指令现已过时。在 Solaris SPARC 和 x86 平台上优先使用 OpenMP API 以实现并行化。有关将传统的应用程序迁移到 OpenMP 的信息，请参见 《OpenMP API 用户指南》。

OpenMP 并行化是 Fortran 95 首选的并行化模型。此处描述了传统应用程序的 Sun 风格的并行化指令，并且《Fortran 编程指南》中有关并行化的章节对此进行了详细介绍。

表 D-3 Sun 风格并行化指令摘要

格式	C$PAR *directive [optional_qualifiers]*：初始行 C$PAR& *[more_qualifiers]*：续行 固定格式，指令指示符可以是 C（如此处所示）、c、* 或 !。 使用逗号分隔多个限定符。除非指定 -e 编译器选项，否则忽略第 72 列后面的字符。
TASKCOMMON 指令	C$PAR TASKCOMMON *block_name* 将通用块 *block_name* 中的变量声明为线程专用：线程专用，但线程内为全局。在声明通用块 TASKCOMMON 时，要求该指令出现在该块的**每个**公共声明的后面。
DOALL 指令	C$PAR DOALL *[qualifiers]* 并行化处理后面的 DO 循环。限定符为： PRIVATE(*list*) 将列表上的名称声明为 PRIVATE SHARED(*list*) 将列表上的名称声明为 SHARED MAXCPUS(*n*) 使用不多于 *n* 个线程 READONLY(*list*) 在循环中没有修改列出的变量 SAVELAST 保存所有专用变量的最后一个值 STOREBACK(*list*) 保存列出变量的最后一个值 REDUCTION(*list*) 列出变量是约简变量 SCHEDTYPE(*type*) 使用调度类型：（默认设置为 STATIC） STATIC SELF(*nchunk*) FACTORING[(*m*)] GSS[(*m*)]
DOSERIAL 指令	C$PAR DOSERIAL 禁用并行化处理后面的循环。
DOSERIAL* 指令	C$PAR DOSERIAL* 禁用并行化处理后面的循环嵌套。

D.5 Cray 并行化指令

注 – 传统的 Sun 和 Cray 并行化指令现已过时。在 Solaris SPARC 和 x86 平台上优先使用 OpenMP API 以实现并行化。有关将传统的应用程序迁移到 OpenMP 的信息，请参见 《OpenMP API 用户指南》。

《Fortran 编程指南》中有关并行化的章节详细介绍了 Cray 风格的并行化指令。要求使用 `-mp=cray` 编译器选项。

表 D-4 Cray 并行化指令摘要

格式	`CMIC$` *directive qualifiers*：初始行 `CMIC$&` *[more_qualifiers]*：续行 固定格式。指示符可以是 `C`（如此处所示）、`c`、`*` 或 `!`。对于 `f95` 自由格式，可以在 `!MIC$` 前面出现前导空格。

`DOALL` 指令	`CMIC$ DOALL SHARED(`*list*`), PRIVATE(`*list*`)` *[, more_qualifiers]*
	
	并行化处理后面的循环。限定符为：
	要求使用作用域限定符（除非 *list* 为空）ó 循环中的变量必须出现在 `PRIVATE` 或 `SHARED` 子句中：
	`PRIVATE(`*list*`)` 将列表上的名称声明为 PRIVATE
	`SHARED(`*list*`)` 将列表上的名称声明为 SHARED
	`AUTOSCOPE` 自动确定变量的作用域
	以下限定符是可选的：
	`MAXCPUS(`*n*`)` 使用不多于 *n* 个线程
	`SAVELAST` 保存所有专用变量的最后一个值
	只能出现一个调度限定符：
	`GUIDED` 相当于 Sun 风格的 `GSS(64)`
	`SINGLE` 相当于 Sun 风格的 `SELF(1)`
	`CHUNKSIZE(`*n*`)` 相当于 Sun 风格的 `SELF(n)`
	`NUMCHUNKS(`*m*`)` 相当于 Sun 风格的 `SELF(n/m)`
	默认调度为 Sun 风格的 `STATIC`，它没有等价的 Cray 风格限定符。对于这些 Sun 和 Cray 风格的调度限定符，它们的解释是不同的。详细信息，请参见 《Fortran 编程指南》。

表 D-4 Cray 并行化指令摘要 （续）

TASKCOMMON 指令	CMIC$ TASKCOMMON *block_name* 将命名通用块中的变量声明为**线程专用** - 线程专用，但线程内为全局。在声明通用块 TASKCOMMON 时，要求此指令出现在紧靠该块的**每个**公共声明的后面。
DOSERIAL 指令	CMIC$ DOSERIAL 禁用并行化处理后面的循环。
DOSERIAL* 指令	CMIC$ DOSERIAL* 禁用并行化处理后面的循环嵌套。

索引

C

D

E

F

G

H

I

J

T

U

V

W

X