Oracle® Solaris Studio 12.2 ： Fortran 用户指南

文件号码 821–2508
2010 年 9 月

ORACLE

版权所有 © 1991, 2010, Oracle 和/或其附属公司。 保留所有权利。

本软件和相关文档是根据许可证协议提供的，该许可证协议中规定了关于使用和公开本软件和相关文档的各种限制，并受知识产权法的保护。除非在许可证协议中明确许可或适用法律明确授权，否则不得以任何形式、任何方式使用、拷贝、复制、翻译、广播、修改、授权、传播、分发、展示、执行、发布或显示本软件和相关文档的任何部分。除非法律要求实现互操作，否则严禁对本软件进行逆向工程设计、反汇编或反编译。

此文档所含信息可能随时被修改，恕不另行通知，我们不保证该信息没有错误。如果贵方发现任何问题，请书面通知我们。

如果将本软件或相关文档交付给美国政府，或者交付给以美国政府名义获得许可证的任何机构，必须符合以下规定：

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations.As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007).Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

本软件或硬件是为了在各种信息管理应用领域内的一般使用而开发的。它不应被应用于任何存在危险或潜在危险的应用领域，也不是为此而开发的，其中包括可能会产生人身伤害的应用领域。如果在危险应用领域内使用本软件或硬件，贵方应负责采取所有适当的防范措施，包括备份、冗余和其它确保安全使用本软件或硬件的措施。对于因在危险应用领域内使用本软件或硬件所造成的一切损失或损害，Oracle Corporation 及其附属公司概不负责。

Oracle 和 Java 是Oracle 和/或其附属公司的注册商标。其他名称可能是各自所有者的商标。

AMD、Opteron、AMD 徽标以及 AMD Opteron 徽标是Advanced Micro Devices 的商标或注册商标。Intel 和 Intel Xeon是Intel Corporation 的商标或注册商标。所有SPARC 商标均是 SPARC International, Inc 的商标或注册商标，并应按照许可证的规定使用。UNIX 是通过 X/Open Company, Ltd 授权的注册商标。

本软件或硬件以及文档可能提供了访问第三方内容、产品和服务的方式或有关这些内容、产品和服务的信息。对于第三方内容、产品和服务，Oracle Corporation 及其附属公司明确表示不承担任何种类的担保，亦不对其承担任何责任。对于因访问或使用第三方内容、产品或服务所造成的任何损失、成本或损害，Oracle Corporation 及其附属公司概不负责。

110414@25097

目录

前言 13

1 简介 17
1.1 标准一致性 17
1.2 Fortran 编译器的功能 18
1.3 其他 Fortran 实用程序 18
1.4 调试实用程序 19
1.5 Sun 性能库 19
1.6 区间运算 19
1.7 手册页 19
1.8 命令行帮助 20

2 使用 Solaris Studio Fortran 21
2.1 快速入门 21
2.2 调用编译器 22
2.2.1 编译和链接序列 23
2.2.2 命令行文件命名约定 23
2.2.3 源文件 24
2.2.4 源文件预处理程序 24
2.2.5 分别编译和链接 25
2.2.6 一致编译和链接 25
2.2.7 无法识别的命令行参数 25
2.2.8 模块 26
2.3 指令 26
2.3.1 通用指令 27
2.3.2 并行化指令 32
2.3.3 **IVDEP** 指令 33

2.4 库接口和 **system.inc** 34
2.5 编译器用法提示 35
2.5.1 确定硬件平台 35
2.5.2 使用环境变量 35
2.5.3 内存大小 36

3 Fortran 编译器选项 39
3.1 命令语法 39
3.2 选项语法 39
3.3 选项摘要 40
3.3.1 常用选项 45
3.3.2 宏标志 46
3.3.3 向后兼容性和传统选项 46
3.3.4 已过时的选项标志 47
3.4 选项参考 47
3.4.1 **–aligncommon**[={ **1**|**2**|**4**|**8**| **16**}] 47
3.4.2 **–ansi** 48
3.4.3 **–arg=local** 48
3.4.4 **–autopar** 48
3.4.5 **–B**{**static**| **dynamic**} 49
3.4.6 **–C** 50
3.4.7 **–c** 50
3.4.8 **–copyargs** 50
3.4.9 **–D***name*[**=***def*] 51
3.4.10 **–dalign** 52
3.4.11 **–dbl_align_all**[**=** {**yes**|**no**}] 52
3.4.12 **–depend**[**=**{ **yes**|**no**}] 53
3.4.13 **–dn** 53
3.4.14 **–dryrun** 53
3.4.15 **–d{ y|n}** 53
3.4.16 **–e** 54
3.4.17 **–erroff**[**=**{ **%all**|**%none**|*taglist*}] 54
3.4.18 **–errtags**[**=**{ **yes**|**no**}] 54
3.4.19 **–errwarn**[**=**{ **%all**|**%none**|*taglist*}] 54
3.4.20 **–ext_names=** *e* 55

3.4.21 **–F** 55
3.4.22 **–f** 55
3.4.23 **–f77**[**=** *list*] 56
3.4.24 **–fast** 57
3.4.25 **–fixed** 59
3.4.26 **–flags** 59
3.4.27 **–fma=**{**none**| **fused**} 59
3.4.28 **–fnonstd** 59
3.4.29 **–fns[={yes| no**}] 60
3.4.30 **–fpover**[={ **yes**|**no**}] 61
3.4.31 **–fpp** 61
3.4.32 **–fprecision=**{**single** |**double**|**extended**} 61
3.4.33 **–free** 62
3.4.34 **–fround=**{**nearest** |**tozero**|**negative**| **positive**} 62
3.4.35 **–fsimple**[={ **1**|**2**|**0**}] 62
3.4.36 **–fstore** 63
3.4.37 **–ftrap=** *t* 63
3.4.38 **–G** 64
3.4.39 **–g** 64
3.4.40 **–h***name* 65
3.4.41 **–help** 65
3.4.42 **–I***path* 65
3.4.43 **-i8** 66
3.4.44 **–inline=**[**%auto**][[**,**][**no%**]*f1*,…[**no%**]*fn*] 66
3.4.45 **–iorounding**[={ **compatible**|**processor-defined**}] 67
3.4.46 **–keeptmp** 67
3.4.47 **–Kpic** 67
3.4.48 **–KPIC** 67
3.4.49 **–L***path* 68
3.4.50 **–l***x* 68
3.4.51 **–libmil** 69
3.4.52 **–loopinfo** 69
3.4.53 **–M***path* 69
3.4.54 **–m32** | **–m64** 70
3.4.55 **–moddir=** *path* 70
3.4.56 **-mt**[={ **yes**|**no**}] 71

3.4.57 **–native** 71
3.4.58 **–noautopar** 71
3.4.59 **–nodepend** 72
3.4.60 **-nofstore** 72
3.4.61 **–nolib** 72
3.4.62 **–nolibmil** 72
3.4.63 **–noreduction** 72
3.4.64 **–norunpath** 73
3.4.65 **–O**[*n*] 73
3.4.66 **–O** 73
3.4.67 **–O1** 74
3.4.68 **–O2** 74
3.4.69 **–O3** 74
3.4.70 **–O4** 74
3.4.71 **–O5** 74
3.4.72 **–o** *name* 75
3.4.73 **–onetrip** 75
3.4.74 **–openmp** 75
3.4.75 **–p** 75
3.4.76 **–pad**[**=***p*] 75
3.4.77 **–pg** 76
3.4.78 **–pic** 77
3.4.79 **–PIC** 77
3.4.80 **–Qoption** *pr ls* 78
3.4.81 **–qp** 78
3.4.82 **–R** *ls* 78
3.4.83 **–r8const** 78
3.4.84 **–recl=a**[**,b**] 79
3.4.85 **–reduction** 79
3.4.86 **–S** 79
3.4.87 **–s** 79
3.4.88 **–sb** 80
3.4.89 **–sbfast** 80
3.4.90 **–silent** 80
3.4.91 **–stackvar** 80
3.4.92 **–stop_status**[**=**{ **yes**|**no**}] 81

3.4.93 **–temp=***dir* 82
3.4.94 **–time** 82
3.4.95 **–traceback**[={ **%none**|**common**|*signals_list*}] 82
3.4.96 **–U** 83
3.4.97 **–U***name* 83
3.4.98 **–u** 84
3.4.99 **–unroll=***n* 84
3.4.100 **–use=** *list* 84
3.4.101 **–V** 84
3.4.102 **–v** 84
3.4.103 **–vax=***keywords* 85
3.4.104 **–vpara** 85
3.4.105 **–w**[*n*] 86
3.4.106 **–Xlist**[*x*] 86
3.4.107 **–xaddr32**[={ **yes**|**no**}] 87
3.4.108 **–xalias**[**=** *keywords*] 88
3.4.109 **–xannotate**[={ **yes**|**no**}] 89
3.4.110 **–xarch=***isa* 90
3.4.111 **–xassume_control**[**=***keywords*] 93
3.4.112 **–xautopar** 94
3.4.113 **–xbinopt=**{**prepare** | **off**} 94
3.4.114 **–xcache=***c* 95
3.4.115 **–xcheck=***keyword* 96
3.4.116 **–xchip=***c* 97
3.4.117 **–xcode=***keyword* 98
3.4.118 **–xcommonchk**[={ **yes**|**no**}] 100
3.4.119 **–xcrossfile**[={ **1**|**0**}] 100
3.4.120 **–xdebugformat=**{**dwarf** |**stabs**} 100
3.4.121 **–xdepend** 101
3.4.122 **–xF** 101
3.4.123 **–xfilebyteorder=** *options* 101
3.4.124 **–xhasc**[={ **yes**|**no**}] 103
3.4.125 **–xhelp=**{**readme**| **flags**} 104
3.4.126 **–xhwcprof[={enable | disable}]** 104
3.4.127 **–xia**[={ **widestneed**|**strict**}] 105
3.4.128 **–xinline=** *list* 105

3.4.129 **–xinstrument=[%no]datarace** 105
3.4.130 **–xinterval**[={ *widestneed*|*strict*|*no*}] 106
3.4.131 **–xipo**[={ **0**|**1**|**2**}] 106
3.4.132 **–xipo_archive**[={ **none**|**readonly**|**writeback**}] 108
3.4.133 **–xivdep**[= *p*] 109
3.4.134 **–xjobs=** *n* 109
3.4.135 **–xkeepframe**[=[**%all,%none**,*name*,**no%** *name*]] 110
3.4.136 **–xknown_lib=***library_list* 110
3.4.137 **–xl** 111
3.4.138 **–xlang=f77** 111
3.4.139 **–xld** 111
3.4.140 **–xlibmil** 111
3.4.141 **–xlibmopt** 111
3.4.142 **–xlic_lib=sunperf** 112
3.4.143 **–xlicinfo** 112
3.4.144 **–xlinkopt**[={ **1**|**2**|**0**}] 112
3.4.145 **–xloopinfo** 113
3.4.146 **–xmaxopt**[= *n*] 113
3.4.147 **–xmemalign**[= <*a*><*b*>] 113
3.4.148 **–xmodel=**[**small** | **kernel** | **medium**] 115
3.4.149 **–xnolib** 115
3.4.150 **–xnolibmil** 115
3.4.151 **–xnolibmopt** 115
3.4.152 **–xO***n* 116
3.4.153 **–xopenmp**[={ **parallel**|**noopt**|**none**}] 116
3.4.154 **–xpad** 117
3.4.155 **–xpagesize=** *size* 117
3.4.156 **–xpagesize_heap=** *size* 118
3.4.157 **–xpagesize_stack=** *size* 118
3.4.158 **–xpec**[={ **yes**|**no**}] 118
3.4.159 **–xpg** 118
3.4.160 **–xpp=**{**fpp**| **cpp**} 119
3.4.161 **–xprefetch**[= *a*[,*a*]] 119
3.4.162 **–xprefetch_auto_type=indirect_array_access** 121
3.4.163 **–xprefetch_level=**{ **1**|**2**|**3**} 121
3.4.164 **–xprofile=***p* 121

3.4.165 **–xprofile_ircache**[=*path*] 124
3.4.166 **–xprofile_pathmap=** *collect_prefix*:*use_prefix* 124
3.4.167 **–xrecursive** 125
3.4.168 **–xreduction** 125
3.4.169 **–xregs=** *r* 125
3.4.170 **–xs** 127
3.4.171 **–xsafe=mem** 127
3.4.172 **–xsb** 127
3.4.173 **–xsbfast** 127
3.4.174 **–xspace** 127
3.4.175 **–xtarget=***t* 128
3.4.176 **–xtime** 130
3.4.177 **–xtypemap=** *spec* 130
3.4.178 **–xunroll=***n* 131
3.4.179 **–xvector**[**=** [[**no%**]**lib**, [**no%**] **simd**, **%none**]] 131
3.4.180 **–ztext** 132

4 Solaris Studio Fortran 的功能与差异 135
4.1 源语言功能 135
4.1.1 续行限制 135
4.1.2 固定格式源代码行 135
4.1.3 制表符格式 135
4.1.4 采用的源代码格式 136
4.1.5 限制和缺省值 137
4.2 数据类型 137
4.2.1 布尔类型 137
4.2.2 数值数据类型的缩写大小表示法 140
4.2.3 数据类型的大小和对齐 141
4.3 Cray 指针 142
4.3.1 语法 142
4.3.2 Cray 指针的用途 143
4.3.3 声明 Cray 指针和 Fortran 95 指针 143
4.3.4 Cray 指针的功能 143
4.3.5 Cray 指针的限制 144
4.3.6 Cray 指针对象的限制 144

4.3.7 Cray 指针的用法 145
4.4 **STRUCTURE** 和 **UNION** (VAX Fortran) 146
4.5 无符号整数 146
4.5.1 算术表达式 147
4.5.2 关系表达式 147
4.5.3 控制构造 147
4.5.4 输入/输出构造 147
4.5.5 内部函数 147
4.6 Fortran 200x 的功能 148
4.6.1 与 C 函数之间的互操作性 148
4.6.2 IEEE 浮点异常处理 148
4.6.3 命令行参数内部函数 149
4.6.4 **PROTECTED** 属性 149
4.6.5 Fortran 2003 异步 I/O 149
4.6.6 扩展的 **ALLOCATABLE** 属性 149
4.6.7 **VALUE** 属性 150
4.6.8 Fortran 2003 流 I/O 150
4.6.9 Fortran 2003 格式化 I/O 功能 150
4.6.10 Fortran 2003 **IMPORT** 语句 151
4.6.11 Fortran 2003 **FLUSH** I/O 语句 151
4.6.12 Fortran 2003 **POINTER INTENT** 功能 152
4.6.13 Fortran 2003 中增强的数组构造函数 152
4.6.14 Fortran 2003 和 Fortran 2008 的其他功能 152
4.7 其他的 I/O 扩展 153
4.7.1 I/O 错误处理例程 153
4.7.2 变量格式表达式 153
4.7.3 **NAMELIST** 输入格式 154
4.7.4 二进制未格式化 I/O 154
4.7.5 各种 I/O 扩展 154
4.8 指令 155
4.8.1 特殊 **f95** 指令行的格式 155
4.8.2 **FIXED** 和 **FREE** 指令 155
4.8.3 并行化指令 156
4.9 模块文件 156
4.9.1 搜索模块 157
4.9.2 **-use=***list* 选项标志 158

4.9.3 **fdumpmod** 命令 158
4.10 内部函数 158
4.11 向前兼容性 159
4.12 混合语言 159

5 FORTRAN 77 兼容性：迁移到 Solaris Studio Fortran 161
5.1 兼容的 **f77** 功能 161
5.2 不兼容问题 165
5.3 与旧版 FORTRAN 77 编译的例程链接 166
5.3.1 Fortran 内部函数 167
5.4 有关迁移到 **f95** 编译器的附加说明 167
5.5 **f77** 命令 168

A 运行时错误消息 169
A.1 操作系统错误消息 169
A.2 **f95** 运行时 I/O 错误消息 169

B 功能发行版历史记录 177
B.1 Oracle Solaris Studio 12.2 Fortran 发行版 177
B.2 Sun Studio 12 Update 1 Fortran 发行版 178
B.3 Sun Studio 12 Fortran 发行版 179
B.4 Sun Studio 11 Fortran 发行版 179
B.5 Sun Studio 10 Fortran 发行版： 180
B.6 Sun Studio 9 Fortran 发行版： 180
B.7 Sun Studio 8 Fortran 发行版： 182
B.8 Sun ONE Studio 7 编译器集合 (Forte Developer 7) 版本： 184

C Fortran 指令摘要 187
C.1 通用 Fortran 指令 187
C.2 特殊的 Fortran 指令 188
C.3 Fortran OpenMP 指令 189

索引 191

前言

Oracle Solaris Fortran 用户指南介绍了 Oracle Solaris Studio Fortran 编译器 **f95** 的环境及命令行选项。本指南的目标读者是精通 Fortran 语言并希望了解如何有效使用 Oracle Solaris Studio Fortran 编译器的科学工作者、工程师和程序员。通常，还假定这些人员熟悉 Solaris 操作环境或 UNIX®。

有关 Oracle Solaris 操作环境中各种 Fortran 编程问题的讨论可在随附的Fortran 编程指南中找到，讨论的问题包括输入/输出、应用程序开发、库的创建和使用、程序分析、移植、优化和并行处理。

受支持的平台

此 Oracle Solaris Studio 发行版支持使用以下 SPARC 和 x86 系列处理器体系结构的系统：UltraSPARC、SPARC64、AMD64、Pentium 和 Xeon EM64T。所运行的 Oracle Solaris 操作系统版本支持的系统可在 `http://www.sun.com/bigadmin/hcl` 上的硬件兼容性列表中找到。这些文档中给出了平台类型间所有实现的区别。

在本文档中，与 x86 相关的术语的含义如下：

- "x86" 泛指 64 位和 32 位的 x86 兼容产品系列。
- "x64" 指出了有关 AMD64 或 EM64T 系统的特定 64 位信息。
- “32 位 x86”指出了有关基于 x86 的系统的特定 32 位信息。

有关受支持的系统，请参阅硬件兼容性列表。

访问 Solaris Studio 文档

可以访问以下位置的文档：

- 可从以下位置的文档索引页面中获取文档：`http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation`。
- IDE、性能分析器、dbxtool 和 DLight 的所有组件的联机帮助可以在这些工具中通过“帮助”菜单、F1 键以及许多窗口和对话框上的“帮助”按钮获取。

采用易读格式的文档

该文档以易读格式提供，以方便残障用户使用辅助技术进行阅读。可以按照下表所述找到文档的易读版本。

文档类型	易读版本的格式和位置
手册	HTML，docs.sun.com 上的 Oracle Solaris Studio 12.2 Collection - Simplified Chinese 中
《Oracle Solaris Studio 12.2 发行版的新增功能》（以前的组件自述文件）	HTML，docs.sun.com 上的 Oracle Solaris Studio 12.2 Collection - Simplified Chinese 中
手册页	使用 man 命令在 Oracle Solaris 终端中显示
联机帮助	HTML，可在 IDE、dbxtool、dbxtool、DLight 和性能分析器中通过“帮助”菜单、“帮助”按钮和 F1 键获取。
发行说明	HTML，docs.sun.com 上的 Oracle Solaris Studio 12.2 Collection - Simplified Chinese 中

相关的第三方 Web 站点引用

本文档引用了第三方 URL，以用于提供其他相关信息。

注 – Oracle 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，Oracle 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，Oracle 概不负责，也不承担任何责任。

开发者资源

要找到以下经常更新的资源，请访问 `http://www.oracle.com/technetwork/server-storage/solarisstudio`：

- 有关编程技术和最佳做法的文章
- 软件文档以及随软件一起安装的文档的更正信息
- 指导您使用 Oracle Solaris Studio 工具逐步完成开发任务的教程
- 有关支持级别的信息
- `http://forums.sun.com/category.jspa?categoryID=113` 上的用户论坛

印刷约定

下表介绍了本书中的印刷约定。

表 P–1 印刷约定

字体	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。 `machine_name% you have mail.`
`AaBbCc123`	用户键入的内容，与计算机屏幕输出的显示不同	`machine_name%` **`su`** `Password:`
aabbcc123	要使用实名或值替换的命令行占位符	删除文件的命令为 `rm` *filename*。
AaBbCc123	《书名》新词或术语以及要强调的词	阅读《用户指南》的第 6 章。 **高速缓存**是存储在本地的副本。 请**勿**保存文件。 **注意**： 有些强调的项目在联机时以粗体显示。

命令中的 shell 提示符示例

下表显示了 Oracle Solaris OS 中包含的 shell 的缺省 UNIX 系统提示符和超级用户提示符。请注意，显示在命令示例中的缺省系统提示符会根据 Oracle Solaris 发行版的不同而有所不同。

表 P–2 Shell 提示符

Shell	提示符
Bash shell、Korn shell 和 Bourne shell	`$`
Bash shell、Korn shell 和 Bourne shell 超级用户	`#`
C shell	`machine_name%`
C shell 超级用户	`machine_name#`

文档、支持和培训

有关其他资源，请参见以下 Web 站点：

- 文档 (http://docs.sun.com)
- 支持 (http://www.oracle.com/us/support/systems/index.html)
- 培训 (http://education.oracle.com) – 单击左侧导航栏中的 Sun 链接。

Oracle 欢迎您提出意见

Oracle 欢迎您针对其文档质量和实用性提出意见和建议。如果您发现任何错误，或有其他任何改进建议，请转至 http://docs.sun.com，然后单击 Feedback（反馈）。请提供文档的标题和文件号码，以及章节和页码（如果有）。如果您需要回复，请告知。

Oracle 技术网络 (http://www.oracle.com/technology/global/cn/index.html) 提供了一系列与 Oracle 软件相关的资源：

- 在讨论论坛 (http://forums.oracle.com) 上讨论技术问题和解决方案。
- 通过 Oracle By Example (http://www.oracle.com/technology/obe/start/index.html) 获得逐步骤实用教程。
- 下载样例代码 (http://www.oracle.com/technology/sample_code/index.html)。

◆ ◆ ◆ 第 1 章

1 简介

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

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

1.1 标准一致性

- **f95** 符合 ISO/IEC 1539-1:1997 Fortran 标准文档的第一部分。
- 浮点运算基于 IEEE 标准 754-1985 和国际标准 IEC 60559:1989。
- **f95** 提供了对 SPARC 和 x86 系列处理器体系结构（ Solaris 和 Linux (x86) 平台上的 UltraSPARC、SPARC64、AMD64、Pentium Pro 和 Xeon Intel® 64）优化开发功能的支持。
- Solaris Studio 编译器符合 OpenMP 3.0 共享内存并行化 API 规范。有关详细信息，请参见《OpenMP API 用户指南》。
- 在本文档中，“标准”是指与上面列出的标准版本相一致。“非标准”或“扩展”是指超出这些标准版本的功能。

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

1.2 Fortran 编译器的功能

Solaris Studio Fortran 编译器 **f95** 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。（仅适用于 Solaris 平台）。
- 优化多处理器系统的自动和显式循环并行化。
- VAX/VMS Fortran 扩展，其中包括：
  - 结构、记录、联合和映射
  - 递归
- OpenMP 3.0 并行化指令。
- 全局、窥孔和潜在的并行化优化可产生高性能的应用程序。基准测试表明优化的应用程序的运行速度比未优化的代码快得多。
- 通用调用惯例允许将使用 C 或 C++ 编写的例程与 Fortran 程序结合使用。
- 支持启用的 64 位 Solaris 和 Linux 环境。
- 使用 **%VAL** 按值进行调用。
- Fortran 77 和 Fortran 95/Fortran 2003 程序与目标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的编译器中添加的新功能和扩展功能的详细信息，请参见附录 B。

另请参见《Oracle Solaris Studio 12.2 发行版的新增功能》，以了解有关本编译器和工具发行版中新增和更改的功能、已知问题、解决方法和限制的最新信息。可以通过 `http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation` 上的本发行版文档索引页访问该“新增功能”指南。

1.3 其他 Fortran 实用程序

以下实用程序可为使用 Fortran 进行软件程序开发提供帮助：

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

1.4 调试实用程序

可以使用以下调试实用程序：

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

1.5 Sun 性能库

Sun 性能库是一个用于计算线性代数和傅立叶变换的优化子例程及函数的库。它基于一般通过 Netlib (**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。

有关详细信息，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

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

1.7 手册页

联机手册 (**man**) 页提供了关于命令、函数、子例程或这些项的集合的当前文档。要访问 Solaris Studio 的手册页，应将用户的 **MANPATH** 环境变量设置为 Solaris Studio 安装目录 **man**。

可以通过运行以下命令来显示手册页：

```
demo% man topic
```

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

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

Fortran 库例程是在手册页第 3F 节中介绍的。

下面列出了对于 Fortran 用户非常重要的 **man** 页：

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

1.8 命令行帮助

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

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

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

◆ ◆ ◆ 第 2 章

2 使用 Solaris Studio Fortran

本章介绍如何使用 Fortran 编译器。

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

编译器还可以用于：

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

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

2.1 快速入门

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

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

示例：此程序在屏幕上显示一条消息：

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

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

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

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

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

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

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

2.2 调用编译器

在 shell 提示符下调用编译器有一个**简单**的命令命令，其语法是：

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

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

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

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

注 – 可以使用 **f95** 或 **f90** 命令来调用 Fortran 编译器。

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

第 23 页中的 “2.2.2 命令行文件命名约定”说明了编译器可以识别的各种源文件扩展名。

2.2.1 编译和链接序列

在上一示例中，编译器自动生成加载器目标文件 **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。此约定使您可以方便地重新链接和重新编译文件。

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

2.2.2 命令行文件命名约定

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

表 2–1 由 Fortran 编译器识别的文件名后缀

后缀	语言	操作
.f	Fortran 77 或 Fortran 95 固定格式	编译 Fortran 源文件，将目标文件放在当前目录中；目标文件的缺省名称是源文件的名称，但具有 **.o** 后缀。
.f95 **.f90**	Fortran 95 自由格式	执行与 **.f** 相同的操作
.f03	Fortran 2003 自由格式	执行与 **.f** 相同的操作
.for	Fortran 77 或 Fortran 95 固定格式	执行与 **.f** 相同的操作。
.F	Fortran 77 或 Fortran 95 固定格式	在编译前，将 Fortran（或 C）预处理程序应用于 Fortran 77 源文件。

表 2–1 由 Fortran 编译器识别的文件名后缀 （续）

后缀	语言	操作
.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	目标文件	将目标文件传递到链接程序。
.a、**.so**、**.so.***n*	库	将库名称传递给链接程序。**.a** 文件是静态库，**.so** 和 **.so.***n* 文件是动态库。

Fortran 95 自由格式在第 135 页中的 “4.1 源语言功能”中进行说明。

2.2.3 源文件

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

2.2.4 源文件预处理程序

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

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

fpp 的源代码可从 Netlib Web 站点获得：

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

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

2.2.5 分别编译和链接

您可以在不同的步骤中编译和链接。**-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                              (Make new object file)
demo% f95 -o prgrm file1.o file2.o file3.o          (Make executable file)
```

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

2.2.6 一致编译和链接

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

第 3 章中的选项描述指明了此类选项。

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

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

2.2.7 无法识别的命令行参数

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

基本区别是：

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

例如：

```
demo% f95 -bit move.f                 <- -bit is not a recognized f95 option
f95: Warning: Option -bit passed to ld, if ld is invoked, ignored otherwise
demo% f95 fast move.f                 <- The user meant to type -fast
```

```
ld: fatal: file fast: cannot open file; errno=2
ld: fatal: File processing errors.  No output written to a.out
```

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

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

这些示例表明在编写编译器命令行时应格外小心！

2.2.8 模块

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

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

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

使用 **fdumpmod**(1) 命令可显示与 **.mod** 模块信息文件内容有关的信息。

有关详细信息，请参见 第 156 页中的 “4.9 模块文件”。

2.3 指令

可使用源代码**指令**（一种 Fortran 注释格式）将有关专门的优化或并行化选择的特定信息传递给编译器。编译器指令有时也称为 *pragma*。编译器可以识别一组通用指令和并行化指令。**f95** 还可处理 OpenMP 共享内存多处理指令。

特定于 **f95** 的指令在第 155 页中的 “4.8 指令”中进行说明。附录 C，Fortran 指令摘要中完整总结了 **f95** 可以识别的所有指令。

注 – 指令并不是 Fortran 标准的一部分。

2.3.1 通用指令

Fortran 通用指令的各种形式包括：

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

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

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

通用指令使用以下语法：

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

请遵循以下约束：

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

  ```
  !$PRAGMA Sun UNROLL=3
  ```

Fortran 编译器可识别以下通用指令：

表 2–2 通用 Fortran 指令摘要

C 指令	**!$PRAGMA C(***list* **)** 将一系列外部函数的名称声明为 C 语言例程。
IGNORE_TKR 指令	**!$PRAGMA IGNORE_TKR** {*name* {, *name*} ...} 在解析特定调用时，编译器会忽略在通用过程接口中出现的指定哑元参数名称的类型、种类和等级。

表 2–2 通用 Fortran 指令摘要 （续）

UNROLL 指令	**!$PRAGMA SUN UNROLL=***n* 建议编译器将下面的循环解开为指定的长度 *n*。
WEAK 指令	**!$PRAGMA WEAK(***name*[**=***name2*]**)** 将 *name* 声明为弱符号，或者声明为 *name2* 的别名。
OPT 指令	**!$PRAGMA SUN OPT=***n* 将子程序的优化级别设置为 *n*。
PIPELOOP 指令	**!$PRAGMA SUN PIPELOOP=***n* 断言下面的循环中在间隔为 *n* 的迭代之间存在依赖性。
PREFETCH 指令	**!$PRAGMA SUN_PREFETCH_READ_ONCE(***name***)** **!$PRAGMA SUN_PREFETCH_READ_MANY(***name***)** **!$PRAGMA SUN_PREFETCH_WRITE_ONCE(***name***)** **!$PRAGMA SUN_PREFETCH_WRITE_MANY(***name***)** 请求编译器为名称引用生成预取指令。（需要使用 **-xprefetch** 选项，缺省情况下启用该选项。编译时使用 **–xprefetch=no** 可以禁用预取指令。目标体系结构也必须支持预取指令，而且编译器优化级别必须大于 **–xO2**。）
ASSUME 指令	**!$PRAGMA** [**BEGIN**} **ASSUME (***expression* [**,***probability*]**)** **!$PRAGMA END ASSUME** 断言编译器可假定程序中某些点处的条件为真。

2.3.1.1 C 指令

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

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

示例－为 **C** 编译 **ABC** 和 **XYZ**：

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

2.3.1.2 IGNORE_TKR 指令

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

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

```
INTERFACE BLCKX

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

SUBROUTINE BLCK_64(LEN,SRC)
  REAL SRC(1)
!$PRAGMA IGNORE_TKR SRC
  INTEGER (KIND=8) LEN
END SUBROUTINE

END INTERFACE
```

The subroutine call:

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

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

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

2.3.1.3 UNROLL 指令

UNROLL 指令要求您在 **!$PRAGMA** 后指定 **SUN**。

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

n 是正整数。选项有：

- 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。有关详细信息，请参见《Fortran 编程指南》中有关性能与优化的章节。

示例—解开循环两次：

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

2.3.1.4 WEAK 指令

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

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

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

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

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

2.3.1.5 OPT 指令

OPT 指令要求您在 **!$PRAGMA** 后面指定 **SUN**。

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

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

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

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

2.3.1.6 PIPELOOP[= *n*] 指令

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

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

```
C    We know that the value of K is such that there can be no
C    cross-iteration dependencies (E.g. K>N)
!$PRAGMA SUN PIPELOOP=0
```

```
DO I=1,N
 A(I)=A(I+K) + D(I)
 B(I)=B(I) + A(I)
END DO
```

有关优化的更多信息，请参见《Fortran 编程指南》。

2.3.1.7 PREFETCH 指令

-xprefetch 选项标志（第 119 页中的 “3.4.161 –xprefetch[= *a*[,*a*]]”）可启用一组 **PREFETCH** 指令，这些指令指示编译器在支持预取的处理器上为指定的数据元素生成预取指令。

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

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

2.3.1.8 ASSUME 指令

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

“点断言”**ASSUME** 的语法是：

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

另外，“范围断言”**ASSUME** 的语法是：

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

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

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

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

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

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

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

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

在将来的版本中，特殊内部函数的列表可能会扩展。

可与 **-xassume_control** 编译器选项结合使用。（请参见第 93 页中的 “3.4.111 **–xassume_control**[=*keywords*]”）例如，在使用 **-xassume_control=check** 进行编译时，如果行程计数变为小于 10,000，则上述示例将生成一条警告。

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

另一个示例：

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

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

2.3.2 并行化指令

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** 选项进行编译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 3.0 共享内存并行化模型。传统的 Sun 和 Cray 并行化指令现已过时，不应再使用它们。

2.3.2.1 OpenMP 并行化指令

Fortran 编译器将 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 **-xopenmp** 进行编译。（请参见第 116 页中的 “3.4.153 –xopenmp[={ parallel|noopt|none}]”。）

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

2.3.2.2 传统的 Sun/Cray 并行化指令

注 – 传统的 Sun 和 Cray 风格的并行化指令现已过时。首选使用 OpenMP 并行化 API。有关如何从传统的 Sun/Cray 指令迁移到 OpenMP 模型的信息，请参见《OpenMP API 用户指南》。

2.3.3 IVDEP 指令

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

例如：

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

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

可以使用 **–xivdep** 编译器选项（请参见第 109 页中的 “3.4.133 –xivdep[=*p*]”）来禁用或确定 **IVDEP** 指令的解释。

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

下面的示例解释向后依赖性与向前依赖性。

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

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

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

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

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

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

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

2.4 库接口和 system.inc

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

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

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

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

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

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

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

```
include 'system.inc'
integer(4) mypid
```

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

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

2.5 编译器用法提示

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

2.5.1 确定硬件平台

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

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

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

2.5.2 使用环境变量

可通过设置 **FFLAGS** 或 **OPTIONS** 变量来指定选项。

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

示例：设置 **FFLAGS**：(C Shell)

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

示例：显式地使用 **FFLAGS**：

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

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

f95 -fast -Xlist *files…*

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

注 – make 使用的缺省隐式规则可能无法识别具有 **.f95** 和 **.mod**（模块文件）扩展名的文件。有关详细信息，请参见《Fortran 编程指南》和 Fortran 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以及所编译文件的大小和复杂性。如果优化器内存不足，它将尝试通过在较低的优化级别上重试当前过程来进行恢复，并以命令行上 **-O***n* 选项指定的原始级别继续执行后续例程。

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

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

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

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

2.5.3.1 交换空间限制

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

示例：使用 **swap** 命令：

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

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

2.5.3.2 增加交换空间

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

```
demo# mkfile -v 90m /home/swapfile
/home/swapfile 94317840 bytes
demo# /usr/sbin/swap -a  /home/swapfile
```

2.5.3.3 虚拟内存的控制

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

在 **sh** shell 中，请使用 **ulimit** 命令。请参见 **sh**(1)。

示例：将虚拟内存限定为 16 MB：

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

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

示例：将虚拟内存限定在 16 MB 以内：

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

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

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

示例：对于 32M 的交换空间，请使用以下命令：

在 **sh** shell 中：

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

在 **csh** shell 中：

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

最佳设置取决于所请求的优化等级以及可用的实际内存量和虚拟内存量。

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

对于 **csh**，请使用：

```
demo% limit datasize unlimited
```

对于 **sh** 或 **ksh**，请使用：

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

有关更多信息，请参见《Solaris（64 位）开发者指南》。

◆ ◆ ◆ 第 3 章

3 Fortran 编译器选项

本章详细说明 **f95** 编译器的命令行选项。

- 从第 39 页中的 “3.1 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 40 页中的 “3.3 选项摘要”开始是按功能排列的选项的摘要。
- 从第 47 页中的 “3.4 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* 是由空格分隔的源文件名、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，而且这些选项可以包括其他文件列表。

3.2 选项语法

典型的编译器选项格式为：

表 3–1 选项语法

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

在说明各个选项时使用以下印刷约定：

表 3–2 选项的印刷表示法

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

括号、管道符和省略号是在选项描述中使用的**元字符**，它们不是选项本身的一部分。

选项的一些常规准则如下：

- **–l***x* 是用于与库 **lib***x***.a** 链接的选项。将 **-l***x* 放在文件名列表之后以确保搜索顺序库，始终是较为安全之举。
- 通常，按从左向右的顺序处理编译器选项，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出现在命令行上的选项标志不带值时所假定的值。例如，**-xhasc** 与 **-xhasc=yes** 等效。
- 源文件、目标文件和库按它们在命令行上出现的顺序编译并链接。

3.3 选项摘要

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

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

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

表 3–3 按功能分组的编译器选项

功能		选项标志
编译模式：		
	仅编译；不生成可执行文件	**-c**
	显示由驱动程序生成的命令，但不进行编译	**-dryrun**
	支持 Fortran 77 扩展和兼容性	**-f77**
	保留在编译期间所创建的临时文件。	**-keeptmp**
	指定要将已编译的 **.mod** 模块文件写入的路径	**-moddir=***path*
	指定要编写的目标文件、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*
已编译的代码：		
	对于外部名称，增加/删除尾随下划线	**-ext_names=***x*
	内联指定的用户函数	**-inline=***list*
	与编译位置无关的代码	**-KPIC/-kpic**
	内联某些数学库例程	**-libmil**
	STOP 将整数状态值返回给 shell	**-stop_status[=***yn***]**
	指定代码地址空间	**-xcode=***x*
	启用预取指令。	**-xprefetch[=***x***]**
	指定可选寄存器的使用	**-xregs=***x*
	指定缺省数据映射	**-xtypemap=***x*
数据对齐：		

表 3–3 按功能分组的编译器选项 （续）

功能		选项标志
	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**
	标志未声明变量的使用	**-u**
	检查 **!$PRAGMA ASSUME** 断言	**-xassume_control=check**
	检查在运行时栈是否溢出	**-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**
	禁止显示指定的警告消息	**-erroff=**
	与错误消息一起显示错误标记名称	**-errtags**
	显示编译器选项的摘要	**-flags**、**-help**
	显示编译器的版本号及其阶段	**-V**
	冗余消息	**-v**
	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：		

表 3–3 按功能分组的编译器选项 （续）

功能		选项标志
	允许/要求动态/静态库	**-B***x*
	只允许动态/静态库链接	**-dy**、**-dn**
	生成动态（共享对象）库	**-G**
	为动态库指定名称	**-h***name*
	将目录增加到库搜索路径	**-L***path*
	与库 **lib***name***.a** 或 **lib***name***.so** 链接	**-l***name*
	不将库搜索路径生成到可执行文件中。	**-norunpath**
	将运行时库搜索路径生成到可执行文件中	**-R***path*
	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***]**
优化与性能：		
	分析循环以了解数据依赖性	**-depend**
	使用所选的选项进行优化	**-fast**
	指定优化级别	**-O***n*

表 3–3 按功能分组的编译器选项 （续）

功能		选项标志
	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***]**
	针对编译后优化进行编译	**-xbinopt=prepare**
	启用/调整编译器生成的预取指令	**-xprefetch=***list*
	控制预取指令的自动生成	**-xprefetch_level=***n*
	启用性能分析数据的生成或使用	**-xprofile=***p*
	断言不会出现基于内存的陷阱	**-xsafe=mem**
	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**
	显示循环的并行化信息	**-loopinfo**
	为手动编码的多线程编程进行编译	**-mt**
	接受 OpenMP API 指令并设置相应的环境	**-xopenmp[=***keyword***]**
	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*
	接受扩展（132 个字符）源行	**-e**
	将预处理程序应用于 **.F** 和/或 **.F90** 及 **.F95** 文件，但不进行编译	**-F**
	接受 Fortran 95 固定格式输入	**-fixed**
	使用 **fpp** 预处理程序对所有源文件进行预处理	**-fpp**

表 3–3 按功能分组的编译器选项 （续）

功能		选项标志
	接受 Fortran 95 自由格式输入	**-free**
	将目录添加到 include 文件搜索路径	**-I***path*
	将目录添加到模块搜索路径	**-M***path*
	区分大小写	**-U**
	在实际参数中将霍尔瑞斯常数视为字符	**-xhasc={yes|no}**
	选择要使用的预处理程序（**cpp** 或 **fpp**）	**-xpp[={fpp|cpp}]**
	允许递归子程序调用	**-xrecursive**
目标平台：		
	指定内存模型（32 位或 64 位）。	**-m32 | –m64**
	为优化器指定目标平台指令集	**-xarch=***a*
	为优化器指定目标高速缓存属性	**-xcache=***a*
	为优化器指定目标处理器	**-xchip=***a*
	为优化器指定目标平台	**-xtarget=***a*

3.3.1 常用选项

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

表 3–4 常用选项

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

3.3.2 宏标志

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

表 3–5 宏选项标志

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

命令行上跟在宏标志后面的设置将覆盖或增加宏扩展。

3.3.3 向后兼容性和传统选项

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

表 3–6 向后兼容性选项

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

建议在生成可移植的 Fortran 程序时不要使用这些选项标志。

3.3.4 已过时的选项标志

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

表 3–7 过时的 **f95** 选项

选项标志	等效于
-a	**-xprofile=tcov**
-cg89	**-xtarget=ss2**
-cg92	**-xtarget=ss1000**
-native	**-xtarget=native**
-noqueue	许可证排队。不再需要。
-p	分析。使用 **-pg** 或性能分析器
-pic	**-xcode=pic13**
-PIC	**-xcode=pic32**
-sb	忽略。
-sbfast	忽略。
-silent	忽略。
-xarch={**v7,v8,v8a**}	使用 **–m32**

3.4 选项参考

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

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

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

3.4.1 –aligncommon[={ 1|2|4|8| 16}]

指定通用块和标准数字序列类型中数据的对齐。

此值表示通用块和标准数值序列类型中数据元素的最大对齐（以字节为单位）。

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

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

该选项不影响自然对齐方式小于指定大小的数据。

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

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

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

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

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

另请参见 **-xmemalign**

3.4.2 –ansi

标识许多非标准扩展。

如果在源代码中使用非标准 Fortran 扩展，则会发出警告消息。

3.4.3 –arg=local

通过 **ENTRY** 语句保留实际参数。

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

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

3.4.4 –autopar

启用自动循环并行化。

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

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

如果程序已经包含对 **libthread** 线程库的显式调用，请避免使用 **-autopar**。请参见第 71 页中的 “3.4.56 **-mt**[={ **yes**|**no**}]”中的注释。

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

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。这会将程序可以创建的最大线程数通知给运行时系统。缺省值为 1。通常会将 **PARALLEL** 或 **OMP_NUM_THREADS** 变量设置为目标平台上可用的虚拟处理器数，该值可使用 Solaris **psrinfo**(1) 命令确定。

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

–reduction 选项与 **–autopar** 一起使用也可能会很有用。

有关并行化的更多信息，请参阅《Fortran 编程指南》。有关用户控制的显式并行化，则使用 OpenMP 指令和 **–xopenmp** 选项。

3.4.5 –B{static| dynamic}

首选动态库链接或要求静态库链接。

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

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

另请注意：

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

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

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

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

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

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

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

有关静态库和动态库的更多信息，请参见《Fortran 编程指南》。

3.4.6 –C

检查数组引用以查找超出范围的下标并在运行时检查一致性。

如果数组下标超过所声明的大小，可能会导致意外结果（包括段故障）。**-C** 选项检查源代码中和执行过程中可能的数组下标违规。**-C** 还添加了对数组语法表达式中数组一致性的运行时检查。

指定 **-C** 可能会使可执行文件更大。

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

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

3.4.7 –c

仅编译；生成目标 **.o** 文件，但禁止链接。

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

3.4.8 –copyargs

允许为常量参数赋值。

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

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

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

3.4.9 –D*name*[=*def*]

为预处理程序定义符号 *name*。

此选项仅适用于 **.F**、**.F90**、**.F95** 和 **.F03** 源文件。

–D*name*=*def* 将 *name* 定义为具有值 *def*

–D*name* 将 *name* 定义为 **1**

在命令行上，此选项将定义 *name*，就如同

#define *name*[=*def*]

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

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

- 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 Solaris Studio 12 发行版，**__SUNPRO_F95** 为 **0x850**。
- 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** 和 **__SunOS_5_10**

  例如，值 **__sparc** 是在 SPARC 系统上定义的。
- 以下预定义值不带下划线，但是在以后的发行版中可能会删除这些值：**sparc**、**unix** 和 **sun**。
- 在 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**。

使用详细选项 (**-v**) 进行编译可查看由编译器创建的定义。

您可以在类似如下的预处理程序条件中使用这些值：

```
#ifdef __sparc
```

缺省情况下，**f95** 使用 **fpp**(1) 预处理程序。与 C 预处理程序 **cpp**(1) 一样，**fpp** 会扩展源代码宏并允许对代码进行条件编译。与 **cpp** 不同的是，**fpp** 能够识别 Fortran 语法，并作为首选的 Fortran 预处理程序。使用 **-xpp=cpp** 标志可以强制编译器明确使用 **cpp** 而非 **fpp**。

3.4.10 –dalign

对齐 COMMON 块和标准数值序列类型，并生成速度更快的多字装入/存储。

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

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

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

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

-dalign 是一个宏，它等效于：

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

-aligncommon=8（在 *32* 位 *x86* 平台上）

-aligncommon=16（在 *64* 位 *x86* 平台上）。

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

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

3.4.11 –dbl_align_all[= {yes|no}]

强制与 8 字节边界对齐数据。

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

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

此标志不改变 COMMON 块或用户定义结构中的数据的布局。

与 **-dalign** 一起使用可以提高多字装入/存储的效率。

如果使用了此标志，则所有例程都必须使用此标志进行编译。

3.4.12 –depend[={ yes|no}]

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

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

要关闭依赖性分析，应使用 **-depend=no** 进行编译。

-xdepend 与 **-depend** 等效。

3.4.13 –dn

禁止动态库。请参见第 53 页中的 “3.4.15 **–d{ y|n}**”。

3.4.14 –dryrun

显示由 **f95** 命令行驱动程序生成的命令，但不进行编译。

此选项在调试时非常有用，它显示编译器为执行编译将调用的命令和子选项。

3.4.15 –d{ y|n}

允许或禁止对整个可执行文件使用**动态**库。

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

如果未指定，则缺省值为 **-dy**。

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

–dy|–dn 是加载器和链接程序选项。如果使用这些选项在不同的步骤中编译和链接，则在链接步骤中需要相同选项。

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

3.4.16 –e

接受扩展长度的输入源代码行。

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

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

禁止由标记名称列出的警告消息。

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

示例：

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

使用 **-errtags** 选项可查看与警告消息关联的标记名称。

3.4.18 –errtags[={ yes|no}]

与每个警告消息一起显示消息标记。

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

缺省情况下不显示标记 (**-errtags=no**)。

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

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

将警告消息视为错误。

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

另请参见 **-errtags**。

3.4.20 –ext_names= *e*

创建带有或不带尾随下划线的外部名称。

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

–ext_names=plain： 不增加结尾下划线。

–ext_names=underscores： 增加结尾下划线。

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

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

提供 **fsecond-underscore** 是为了与 **gfortran** 兼容。

3.4.21 –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，并将处理结果写入同名文件，但将该文件的扩展名更改为 **.f**（或者是 **.f95** 或 **.f03**），不进行编译。

示例：

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

将已处理的源文件写入 **source.f**

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

3.4.22 –f

对齐 COMMON 块中的双精度和四精度数据。

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

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

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

如果使用 *-f*编译程序的**任何**部分，则要求使用 *-f*编译该程序的**所有**子程序。

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

3.4.23 –f77[= *list*]

选择 FORTRAN 77 兼容性模式。

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

list 是从下面可能的关键字中选择的逗号分隔列表：

关键字	含义
%all	启用所有 Fortran 77 兼容性功能。
%none	禁用所有 Fortran 77 兼容性功能。
backslash	在字符串中，将反斜线作为转义序列接受。
input	允许 **f77** 接受的输入格式。
intrinsics	将内部函数的识别限制为仅识别 Fortran 77 内部函数。
logical	接受 Fortran 77 的逻辑变量使用，如： －将整数值赋予逻辑变量；－允许在逻辑条件语句中使用算术表达式，用 **.NE.0** 表示 **.TRUE.**；－允许关系运算符 **.EQ.** 和 **.NE.** 与逻辑操作数一起使用
misc	允许多种 **f77** Fortran 77 扩展。
output	生成 **f77** 样式的格式化输出，包括列表式输出和 **NAMELIST** 输出。
subscript	允许将非整数表达式作为数组下标。

关键字	含义
tab	启用 **f77** 样式的制表符格式，包括无限制的源代码行长度。对于长度小于 72 个字符的源代码行，将不增加空白填充。

对于所有关键字，通过在前面加上 **no%** 可禁用相应功能，如下所示：

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

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

异常捕获与 -f77：

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

有关**f77** 兼容性以及从 Fortran 77 迁移到 Fortran 95 的完整信息，请参见第 159 页中的 “4.12 混合语言”。

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

3.4.24 –fast

选择优化执行性能的选项。

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

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

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

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

–fast 会选用以下选项：

- **-xtarget=native** 硬件目标。
  如果打算在编译计算机之外的其他目标计算机上运行程序，请在 **-fast** 之后加上代码生成器选项。例如： **f95 -fast -xtarget=ultraT2 ...**
- **-O5** 优化级别选项。
- **-depend** ??选项分析循环的数据依赖性和可能的重构。
- **-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 62 页中的 “3.4.35 –fsimple[={ 1|2|0}]”。
- **-dalign** 选项可为通用块中的双精度和四精度数据生成双字装入和存储。使用此选项可以在通用块中生成非标准的 Fortran 数据对齐。
- **-xlibmopt** 选项选择优化的数学库例程。
- **?-pad=local** 在局部变量之间插入填充（如果适用），以提高高速缓存利用率。**(SPARC)**
- **-xvector=lib** 使用向量参数将 DO 循环内的某些数学库调用变换为对向量化库等效例程的单个调用。**(SPARC)**
- **–fns** 选择非标准浮点运算异常处理和渐进下溢。请参见第 60 页中的 “3.4.29 –fns[={yes| no}]”。
- 选择 **-fround=nearest**，因为 **–xvector** 和 **–xlibmopt** 要求使用该选项。**(Solaris)**
- **-ftrap=common** 用于捕获常见的浮点异常，在 **f95** 中处于启用状态。
- **-nofstore** 对强制表达式具有结果精度这一行为加以取消。**(x86)**
- 在 x86 平台上，**-xregs=frameptr** 允许编译器将帧指针寄存器用作通用寄存器。有关详细信息，尤其是在编译混合 C、C++ 和 Fortran 源代码情况下的详细信息，请参见 **-xregs=frameptr** 的描述。在 **-fast** 之后指定 **-xregs=no%frameptr**，帧指针寄存器将不会作为通用寄存器使用。**(x86)**

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

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

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

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

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

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

3.4.25 –fixed

指定固定格式的 Fortran 95 源输入文件。

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

3.4.26 –flags

与 **-help** 等效。

3.4.27 –fma={none| fused}

(SPARC) 启用自动生成浮点乘加指令。**-fma=none** 禁用生成这些指令。**-fma=fused** 允许编译器通过使用浮点乘加指令尝试查找机会来提高代码性能。缺省值为 **-fma=none**。

编译器生成乘加指令的最低要求为 **-xarch=sparcfmaf**，优化级别至少为 **-xO2**。如果已生成乘加指令来避免在不支持它们的平台上执行程序，编译器将标记二进制程序。

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

3.4.28 –fnonstd

按非标准首选项初始化浮点硬件。

此选项是以下选项标志组合的宏：

```
–fns -ftrap=common
```

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

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

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

主程序必须使用此选项进行编译才能有效。

使用此选项初始化浮点硬件，以达到下列目的：

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

有关渐进下溢和次正规数的更多信息，请参见 **-fns**。

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

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

3.4.29 –fns[={yes| no}]

选择非标准浮点模式。

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

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

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

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

表 3–8 低于正常的 REAL 和 DOUBLE

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

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

缺省情况下，对浮点首选项进行标准初始化：

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

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

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

主程序必须使用此选项进行编译才能有效。

3.4.30 –fpover[={ yes|no}]

检测格式化输入中的浮点溢出。

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

3.4.31 –fpp

使用 **fpp** 强制对输入进行预处理。

通过 **fpp** 预处理程序传递在 **f95** 命令行上列出的所有输入源文件，而不管文件扩展名为何。（通常，**fpp** 仅自动预处理扩展名为 **.F**、**.F90** 或 **.F95** 的文件。）另请参见 第 119 页中的 “3.4.160 –xpp={fpp| cpp}”。

3.4.32 –fprecision={single |double|extended}

(**x86**) 初始化非缺省的浮点舍入精度模式。

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

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

只有在 x86 系统上用于编译主程序时，此选项才有效，而如果针对 64 位 (**-m64**) 或启用 SSE2 的 (**xarch=sse2**) 处理器进行编译，将忽略此选项。在 SPARC 系统上也将忽略此选项。

3.4.33 –free

指定自由格式源输入文件。

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

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

设置启动时有效的 IEEE 舍入模式。

缺省值为 **-fround=nearest**。

主程序必须使用此选项进行编译才能有效。

该选项将 IEEE 754 舍入模式设置为：

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

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

3.4.35 –fsimple[={ 1|2|0}]

选择浮点优化首选项。

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

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

缺省值为：

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

各种浮点简化级别如下：

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

–fsimple=1 允许适度的简化。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。

在 **-fsimple=1** 的情况下，优化器可假定：

- 在进程初始化之后，IEEE 754 缺省舍入/捕获模式不发生改变。
- 可以删除不生成可见结果（潜在的浮点异常除外）的计算。
- 以无穷大或 NaN（“不是数”）为操作数的计算不需要将 NaN 传播到其结果；例如，**x*0** 可以由 **0** 替换。
- 计算过程不依赖于零的符号。

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

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

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

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

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

–fast 会选择 **-fsimple=2**。

3.4.36 –fstore

(x86) 强制浮点表达式的精度。

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

3.4.37 –ftrap=*t*

设置在启动时有效的浮点捕获模式。

t 是一个逗号分隔列表，它包含以下项中的一个或多个：

%all, **%none**, **common** , **[no%]invalid** , **[no%]overflow** , **[no%]underflow** , **[no%]division** , **[no%]inexact** 。

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

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

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

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

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

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

主程序必须使用此选项进行编译才能有效。

有关详细信息，请参见《Fortran 编程指南》的“浮点运算”一章。

3.4.38 –G

生成动态共享库，而不是生成可执行文件。

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

3.4.39 –g

针对调试和性能分析进行调试。

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

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

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

要使用性能分析器的完整功能，请使用 **-g** 进行编译。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 **analyzer**(1) 手册页和《Solaris Studio 性能分析器》手册。）

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

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

3.4.40 –h*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》以及《Fortran 编程指南》的“库”一章。

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

如果不使用 **-h***name* 选项，则在库文件中不记录内部名称。

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

如果没有共享库的内部名称，则链接程序将改用共享库文件的特定路径。

3.4.41 –help

显示编译器选项的摘要列表。

另请参见第 104 页中的 “3.4.125 **–xhelp=**{**readme**| **flags**}”。

3.4.42 –I*path*

将 *path* 添加到 **INCLUDE** 文件搜索路径。

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

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

搜索路径还用于查找 **MODULE** 文件。

示例：在 **/usr/app/include** 中搜索 **INCLUDE** 文件：

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

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

INCLUDE 或 **#include** 的相对路径的搜索顺序如下：

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

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

3.4.43 -i8

（**没有 -i8 选项**。）

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

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

启用或禁用指定例程的内联。

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

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

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

示例：对 **xbar**、**zbar** 和 **vpoint** 例程进行内联：

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

下面是一些限制；不发出任何警告：

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

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

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

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

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

为格式化输入/输出设置浮点舍入模式。

以全局方式为所有的格式化输入/输出运算设置 **ROUND=** 说明符。

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

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

3.4.46 –keeptmp

保留在编译期间所创建的临时文件。

3.4.47 –Kpic

（已过时）与 **-pic** 等效。

3.4.48 –KPIC

（已过时）与 **-PIC** 等效。

3.4.49 –L*path*

将 *path* 添加到要在其中搜索库的目录路径的列表中。

将 *path* 添加到目标库搜索目录列表的**前面**。`-L` 和 *path* 之间的空格是可选的。此选项将传递给链接程序。另请参见第 68 页中的 “3.4.50 –l*x*”。

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

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

示例：使用 **-L***path* 指定库搜索目录：

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

3.4.50 –l*x*

将库 **lib***x***.a** 添加到链接程序的搜索库列表中。

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

示例：与库 **libVZY** 进行链接：

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

再次使用 **-l***x* 以便与更多的库链接。

示例：与库 **liby** 和 **libz** 进行链接：

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

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

3.4.51 –libmil

内联所选的用于优化的 **libm** 库例程。

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

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

3.4.52 –loopinfo

显示循环的并行化结果。

显示哪些循环通过 **–autopar** 选项实现了并行化，而哪些未实现并行化。

–loopinfo 显示有关标准错误的消息的列表：

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

3.4.53 –M*path*

指定 **MODULE** 目录、归档或文件。

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

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

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

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

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

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

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

其中 *empty-dir* 是空目录的路径。

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

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

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

有关 Fortran 中模块的更多信息，请参见第 156 页中的 “4.9 模块文件”。

3.4.54 –m32 | –m64

为编译的二进制对象指定内存模型。

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

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

使用 **-m32** 编译的目标文件或库不能同使用 **-m64** 编译的目标文件或库链接。

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

请注意，部分 Linux 平台不支持中等模型。

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

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

3.4.55 –moddir= *path*

指定编译器将编译的 **.mod** MODULE 文件写入的位置。

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

编译器将当前目录用作写入 **.mod** 文件的缺省目录。

有关 Fortran 中模块的更多信息，请参见第 156 页中的 “4.9 模块文件”。

3.4.56 -mt[={ yes|no}]

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

此选项将 **-D_REENTRANT** 传递给预处理程序。

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

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

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

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

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

选项 **-mt** 等效于 **-mt=yes**。

3.4.57 –native

（已过时）优化主机系统的性能。

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

3.4.58 –noautopar

禁用由先前出现在命令行上的 **-autopar** 调用的自动并行化。

3.4.59 –nodepend

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

3.4.60 -nofstore

(x86) 取消命令行上的 **-fstore**。

编译器的缺省值为 **-fstore**。**-fast** 包括 **-nofstore**。

3.4.61 –nolib

禁用与系统库的链接。

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

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

将 **libm**（静态）和 **libc**（动态）与 **f95** 链接：

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

-l*x* 选项的顺序是很重要的。请遵循示例所示的顺序。

3.4.62 –nolibmil

取消命令行上的 **-libmil**。

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

```
demo% f95 -fast -nolibmil ...
```

3.4.63 –noreduction

禁用命令行上的 **-reduction**。

此选项禁用 **-reduction**。

3.4.64 –norunpath

不会将运行时共享库搜索路径生成到可执行文件中。

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

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

有关更多信息，请参见《Fortran 编程指南》的“库”一章。

3.4.65 –O[*n*]

指定优化级别。

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

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

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

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

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

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

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

3.4.66 –O

此选项与 **-O3** 等效。

3.4.67 –01

提供最少的语句级优化。

如果更高的级别会导致编译时间过长，或者超过了可用交换空间，请使用此选项。

3.4.68 –02

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**。）

–03 的使用应优先于 **-02**，除非 **-03** 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。

3.4.69 –03

在函数级别上增加循环解开和全局优化。自动添加 **-depend**。

通常，**-03** 生成的可执行文件较大。

3.4.70 –04

增加包含在同一文件中的例程的自动内联。

通常，**-04** 生成的可执行文件较大（因为进行了内联）。

-g 选项会禁止上述的 **-04** 自动内联。**–xcrossfile** 可增加使用 **-04** 情况下的内联范围。

3.4.71 –05

尝试主动优化。

仅适用于程序中占用计算时间最多的小段代码。**-05** 的优化算法要花费更多的编译时间，而且在用于过大的源程序代码段时会降低性能。

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

3.4.72 –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。 如果不设置此选项，则缺省设置是将可执行文件写入 **a.out**。在与 **-c** 一起使用时， **-o** 指定目标 **.o** 目标文件；在与 **-G** 一起使用时，它指定目标 **.so** 库文件。

3.4.73 –onetrip

启用单行程 **DO** 循环。

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

3.4.74 –openmp

与 **-xopenmp** 同义。

3.4.75 –p

（ **已过时** ）为使用 **prof** 配置程序进行分析而编译。

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

3.4.76 –pad[= *p*]

插入填充以便提高高速缓存的使用效率。

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

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

local	在邻近的**局部**变量之间添加填充。

common	在通用块中的变量之间增加填充。
%none	不增加填充。（编译器缺省值。）

如果同时指定了 **local** 和 **common**，则它们可以以任意顺序显示。

-pad 的缺省值：

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

-pad[=*p*] 选项适用于满足以下条件的项：

- 项是数组或字符变量
- 项是静态本地的或处于通用块中

有关本地或静态变量的定义，请参见第 80 页中的 “3.4.91 **–stackvar**”。

程序必须符合以下限制：

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

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

3.4.77 –pg

为使用 **gprof** 配置程序进行分析而编译。

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

库选项必须跟在源文件和 **.o** 文件**后面**（**–pg** 库是静态的）。

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

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

在 32 位 Solaris 系统中，使用 **prof**(1) 生成的配置文件仅限于可执行文件中的例程。通过将可执行文件与 **-pg** 链接并使用 **gprof**(1)，可以对 32 位共享库进行分析。

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

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

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

3.4.78 –pic

为共享库编译与位置无关的代码。

在 SPARC 上，**–pic** 与 **-xcode=pic13** 等效。有关与位置无关的代码的更多信息，请参见第 98 页中的 “3.4.117 **–xcode=***keyword*”。

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

3.4.79 –PIC

使用 32 位地址编译与位置无关的代码。

在 SPARC 上，**–PIC** 与 **-xcode=pic32** 等效。有关与位置无关的代码的更多信息，请参见第 98 页中的 “3.4.117 **–xcode=***keyword*”。

在 x86 上，**–PIC** 与 **–pic** 等效。

3.4.80 –Qoption *pr ls*

将子选项列表 *ls* 传递到编译阶段 *pr*。

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

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

3.4.81 –qp

与 **-p** 等效。

3.4.82 –R *ls*

将动态库搜索路径生成到可执行文件中。

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

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

此选项的多个实例并置在一起，各个列表由冒号分隔。

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

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

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

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

3.4.83 –r8const

将单精度常量提升为 **REAL*8** 常量。

所有单精度 **REAL** 常量都将提升为 **REAL*8** 常量。双精度 (**REAL*8**) 常量保持不变。此选项仅适用于常数。要同时提升常量和变量，请参见第 130 页中的 “3.4.177 **–xtypemap=** *spec*”。

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

3.4.84 –recl=a[,b]

设置缺省输出记录长度。

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

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

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

3.4.85 –reduction

识别循环中的约简操作。

在自动并行化期间分析循环以了解约简操作。约简操作可能存在舍入误差。

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

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

3.4.86 –S

编译并仅生成汇编代码。

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

3.4.87 –s

将符号表与可执行文件分离。

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

3.4.88 –sb

（已过时－忽略此选项。）

3.4.89 –sbfast

（已过时－忽略此选项。）

3.4.90 –silent

（已过时）禁止显示编译器消息。

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

3.4.91 –stackvar

尽可能在栈上分配局部变量。

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

建议将 **-stackvar** 与任何并行化选项一起使用。

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

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

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

使用 **-stackvar** 将大数组放在栈上可以使栈溢出，从而导致段故障。此时可能需要增加栈大小。

执行程序的初始线程有一个**主**栈，而多线程程序的每个辅助线程都有自己的**线程**栈。

主栈的缺省大小约为 8 兆字节。线程栈在 32 位系统上缺省大小为 4 兆字节，在 64 位系统上缺省大小为 8 兆字节。**limit** 命令（不带参数）可显示当前的主栈大小。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：显示当前的**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（单位为千字节），可以设置每个从线程使用的栈大小：

```
% setenv STACKSIZE 8192
```

将每个从线程的栈大小设置为 8 MB。

STACKSIZE 环境变量还接受带有 **B**（字节）、**K**（千字节）、**M**（兆字节）或 **G**（千兆字节）后缀的数值。缺省单位为千字节。

请注意，**STACKSIZE** 环境变量仅影响使用 **–xopenmp** 或 **–xautopar** 选项编译的程序，而不会影响 Solaris 系统上使用 pthreads 接口的程序。

有关与并行化一起使用 **-stackvar** 的详细信息，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，可启用栈溢出情况的运行时检查。请参见第 96 页中的“3.4.115 **–xcheck=***keyword*”。

3.4.92 –stop_status[={ yes|no}]

允许 **STOP** 语句返回整数状态值。

缺省值为 **-stop_status=no**。

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

STOP 123

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

STOP "*stop string*"

并将状态值 0 返回到环境。

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

3.4.93 –temp=*dir*

为临时文件定义目录。

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

此选项优先于 **TMPDIR** 环境变量的值。

3.4.94 –time

每个编译阶段所用时间。

将显示每次编译过程所用的时间和资源。

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

如果执行过程中发生严重错误，则发出栈跟踪。

当程序生成某些信号时，**-traceback** 选项将使可执行文件向 **stderr** 发出栈跟踪，转储核心，然后退出。如果有多个线程生成信号，将仅为第一个线程生成栈跟踪。

要使用回溯，应在链接时将 **-traceback** 选项添加到编译器命令行。编译时也接受该选项，但除非生成二进制可执行文件，否则将忽略该选项。将 **-traceback** 与 **-G** 结合使用来创建共享库的做法是错误的。

表 3–9 -traceback 选项

选项	含义
common	指定在出现以下常用信号集中的某个信号时发出栈跟踪：**sigill**、**sigfpe**、**sigbus**、**sigsegv** 或 **sigabrt**。

表 3–9 -traceback 选项 （续）

选项	含义
signals_list	以小写形式指定生成栈跟踪的信号名称的逗号分隔列表。可以捕捉以下信号（导致生成核心转储文件的信号）：sigquit、sigill、sigtrap、sigabrt、sigemt、sigfpe、sigbus、sigsegv、sigsys、sigxcpu、sigxfsz。 可以在任何这些信号的后面加上 **no%**，以禁止捕捉该信号。 例如：如果发生 sigsegv 或 sigfpe，**-traceback=sigsegv,sigfpe** 将生成栈跟踪和核心转储文件。
%none 或 **none**	禁用回溯

如果不指定此选项，缺省值为 **-traceback=%none**

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

注意：如果不希望生成核心转储，用户可以使用以下命令将其大小设置为零：

```
% limit coredumpsize 0
```

-traceback 选项不影响运行时性能。

3.4.96 –U

识别源文件中的大写字母和小写字母。

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

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

3.4.97 –U*name*

取消预处理程序宏 *name* 的定义。

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

3.4.98 –u

报告未声明的变量。

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

3.4.99 –unroll=*n*

启用 DO 循环的解开（如果可能）。

n 是正整数。选项有：

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

通常，解开循环可提高性能，但会增加可执行文件的大小。有关此编译器及其他编译器优化的更多信息，请参见《Fortran 编程指南》的“性能与优化”一章。另请参见第 29 页中的 “2.3.1.3 **UNROLL** 指令”。

3.4.100 –use= *list*

指定隐式 **USE** 模块。

list 是模块名称或模块文件名称的逗号分隔列表。

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

有关 Fortran 中模块的更多信息，请参见第 156 页中的 “4.9 模块文件”。

3.4.101 –V

显示每次编译过程的名称和版本。

此选项在编译器执行时打印每个工作循环的名称和版本。

在与 Sun 服务工程师讨论问题时，此信息可能非常有用。

3.4.102 –v

详细模式－显示每次编译过程的详细信息。

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

3.4.103 –vax=*keywords*

指定所启用的传统 VAX VMS Fortran 扩展选项。

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

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

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
```

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

3.4.104 –vpara

显示详细的并行化消息。

在编译器对那些用指令显式标记为并行化的循环进行分析时，它发出有关所检测到的某些数据依赖性的警告消息；但是仍将对循环进行并行化。

与 **-xopenmp** 和 OpenMP API 指令结合使用。

由编译器发出的警告检测到下列情形：

- OpenMP 数据共享属性子句使用不当。例如，将变量声明为共享，而在 OpenMP 并行区域中访问该变量会导致数据争用；或者将变量声明为私有，但在并行区域后面使用其在并行区域中的值。

如果在处理所有并行化指令时没有出现任何问题，则不显示警告。

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

3.4.105 –w[*n*]

显示或禁止警告消息。

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

n 可以是 0、1、2、3 或 4。

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

3.4.106 –Xlist[*x*]

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

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

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

示例：检查例程之间的一致性：

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

上述示例将以下内容写入输出文件 **fil.lst**：

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

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

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

表 3–10 –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**
–Xlists	禁止来自交叉引用表的未引用名称
–Xlistv*n*	将检查级别设置为 *n*（1、2、3 或 4）－缺省值为 2
–Xlistw*[nnn]*	将输出行的宽度设置为 *nnn* 列－缺省值为 79
–Xlistwar*[nnn]*	禁止警告 *nnn* 消息
–XlistX	显示交叉引用表和错误

有关详细信息，请参见《Fortran 编程指南》的“程序分析和调试”一章。

此选项在 Linux 系统上不可用。

3.4.107 –xaddr32[={ yes|no}]

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

按此方式编译的可执行文件会导致创建一个限制在 32 位地址空间的进程。如果指定了 **-xaddr32=no**，则会生成通常的 64 位二进制文件。如果未指定 **-xaddr32** 选项，则使用 **-xaddr32=no**。如果仅指定了 **-xaddr32**，则使用 **-xaddr32=yes**。

此选项仅适用于支持 **SF1_SUNW_ADDR32** 软件功能的 Solaris 平台的 **-m64** 编译。由于 Linux 内核不支持地址空间限制，此选项在 Linux 上不可用。在 Linux 上将忽略 **-xaddr32** 选项。

在链接时，如果有一个目标文件是使用 **-xaddr32=yes** 编译的，则整个输出文件均缺省使用 **-xaddr32=yes** 进行编译。限制到 32 位地址空间的共享对象必须由在受限的 32 位模式地址空间内执行的进程来加载。有关详细信息，请参阅《链接程序和库指南》中的 **SF1_SUNW_ADDR32** 软件功能定义。

3.4.108 –xalias[= *keywords*]

指定要由编译器假定的别名程度。

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

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

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

可以在每个关键字前面加上 **no%**，以指示不存在的别名类型。

别名关键字如下：

表 3–11 -xalias 选项关键字

关键字	含义
dummy	子程序的哑元参数（形式参数）可以互为别名，也可以作为全局变量的别名。
no%dummy	（缺省值）。哑元参数的使用遵循 Fortran 标准，它们不能互为别名，也不能作为全局变量的别名。
craypointer	（缺省值）。Cray 指针可以指向任何全局变量或 **LOC()** 函数采用其地址的局部变量。此外，两个 Cray 指针可能指向同一数据。这是可以禁止某些优化的安全假定。
no%craypointer	Cray 指针仅指向唯一内存地址，如从 **malloc()** 获得的地址。此外，没有两个 Cray 指针是指向同一数据的。此假定允许编译器优化 Cray 指针引用。
actual	编译器将子程序的实际参数视为全局变量。将参数传递给子程序可能会导致通过 Cray 指针命名别名。
no%actual	（缺省值）传递参数不会导致进一步命名别名。

表 3–11 **-xalias** 选项关键字 （续）

关键字	含义
overindex	■ 对 COMMON 块中元素的引用可能引用 COMMON 块或等效组中的任何元素。 ■ 如果将 COMMON 块或等效组中的任何元素作为实际参数传递给子程序，被调用的子程序就可以访问该 COMMON 块或等效组中的任何元素。 ■ 将序列派生类型的变量视为 COMMON 块，此类变量的元素可以作为该变量其他元素的别名。 ■ 可能违反了各数组边界，但除上文所述外，假定引用的数组元素仍然在数组内。过度索引并未考虑数组语法、**WHERE** 和 **FORALL** 语句等因素。如果在这些构造中出现过度索引，应将它们重写为 **DO** 循环。
no%overindex	（缺省值）没有违反数组边界。数组引用没有引用其他变量。
ftnpointer	对外部函数的调用可能使 Fortran 指针指向任何类型、种类或等级的目标变量。
no%ftnpointer	（缺省值）Fortran 指针遵循标准中的规则。

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

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

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

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

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

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

3.4.109 –xannotate[={ yes|no}]

（仅限 Solaris）指示编译器创建稍后可用 **binopt**(1) 等二进制修改工具进行转换的二进制文件。

深入二进制分析、代码覆盖和内存错误检测工具也可以处理使用此选项生成的二进制文件。

使用 **-xannotate=no** 选项可以防止这些工具修改二进制文件。**-xannotate=yes** 选项必须与 **-xO1** 或更高优化级别结合使用才能有效，而且它只在包含新链接程序支持库接口 **-ld_open()** 的系统上有效。如果在不包含此链接程序接口的系统（如 Solaris 9 和 Solaris 10 的早期版本）上使用编译器，则该选项会转换为 **-xannotate=no** 而不发出任何提示。

缺省值为 **-xannotate=yes**，但如果不满足上述两种情况，则缺省转换为 **-xannotate=no**。

此选项在 Linux 系统上不可用。

3.4.110 –xarch=*isa*

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

-xarch 关键字 *isa* 接受的体系结构如表 3–12 所示：

表 3–12 –xarch ISA 关键字

平台	有效的 **-xarch** 关键字
SPARC	**generic**, **generic64**, **native**, **native64**, **sparc**, **sparcvis**, **sparcvis2**, **sparcvis3**, **sparcfmaf**, **v9**, **v9a**, **v9b**
x86	**generic**, **native**, **386** , **pentium_pro**, **sse**, **sse2**, **amd64**, **pentium_proa**, **ssea**, **sse2a**, **amd64a**, **sse3,sse3a, ssse3, sse4_1, sse4_2, amdsse4a**

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

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

会覆盖由 **-xtarget=ultra2** 设置的 **-xarch**。

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

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

请注意以下事项：

- 使用 `generic`、`sparc`、`sparcvis2`、`sparcvis3`、`sparcfmaf`、`sparcima` 编译的目标二进制文件 (`.o`) 可以链接，可以在一起执行，但只能在支持所链接的所有指令集的处理器上运行。
- 使用任何特殊选择时，所生成的可执行文件可能无法在传统体系结构上运行或者运行速度大大降低。此外，由于在任何这些指令集体系结构中都不会实现四精度（**REAL*16** 和 **long double**）浮点指令，因此编译器不会在其生成的代码中使用这些指令。

未指定 **-xarch** 选项时的缺省值为 **generic**。

表 3–13 提供了有关 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	**使用大多数处理器通用的指令集进行编译。**
generic64	**针对大多数的 64 位平台进行编译。** （仅限 Solaris）此选项与 **-m64 -xarch=generic** 等效，并与早期的发行版兼容。使用 **-m64**（而非 **-xarch=generic64**）指定 64 位编译
native	**为了在此系统上获得良好性能而进行编译。** 编译器为运行它的当前系统处理器选择适当的设置。这是 **-fast** 选项的缺省值。
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。** （仅适用于 Solaris）此选项与 **-m64 -xarch=native** 等效，并与早先的发行版兼容。
sparc	**针对 SPARC–V9 ISA 进行编译。** 针对 V9 ISA 进行编译，但不具备可视指令集 (Visual Instruction Set, VIS)，也不具备特定于其他实现的 ISA 扩展。该选项在 V9 ISA 上使编译器生成高性能代码。
sparcvis	**针对具有 UltraSPARC 扩展的 SPARC–V9 ISA 进行编译。** 针对 SPARC-V9 加可视指令集 (Visual Instruction Set, VIS) 版本 1.0 进行编译，并具有 UltraSPARC 扩展。该选项在 UltraSPARC 体系结构上使编译器生成高性能代码。
sparcvis2	**针对具有 UltraSPARC-III 扩展的 SPARC-V9 ISA 进行编译。** 此选项允许编译器在具有 UltraSPARC III 扩展的 UltraSPARC 体系结构以及可视化指令集 (VIS) 2.0 版上生成目标代码。
sparcvis3	**针对 SPARC-V9 ISA 的 SPARC VIS 版本 3 进行编译。** 允许编译器使用 SPARC-V9 指令集、UltraSPARC 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 1.0、UltraSPARC-III 扩展（包括可视指令集版本 2.0 以及混合乘加指令和可视指令集版本 3.0 中的指令。
sparcfmaf	**针对 SPARC-V9 ISA 的 sparcfmaf 版本进行编译。** 允许编译器使用 SPARC-V9 指令集，加 UltraSPARC 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 1.0）、UltraSPARC-III 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 2.0）以及面向浮点乘加的 SPARC64 VI 扩展中的指令。 请注意，必须将 **-xarch=sparcfmaf** 与 **-fma=fused** 以及某些优化级别结合使用，来使编译器尝试寻找机会自动使用乘加指令。

表 3–13 SPARC 平台的 **-xarch** 值 （续）

-xarch=	**含义 (SPARC)**
sparcima	针对 SPARC-V9 ISA 的 sparcima 版本进行编译。使编译器可以使用如下指令集内的指令：SPARC-V9 指令集、UltraSPARC 扩展（包括可视化指令集 (Visual Instruction Set, VIS) 版本 1.0）、UltraSPARC-III 扩展（包括可视化指令集 (Visual Instruction Set, VIS) 版本 2.0）、SPARC64 VI 扩展（用于浮点乘加）和 SPARC64 VII 扩展（用于整数乘加）。
v9	与 **-m64 -xarch=sparc** 等效 使用 **xarch=v9** 来获取 64 位内存模型的传统 makefile 和脚本仅需使用 **-m64**。
v9a	与 **-m64 -xarch=sparcvis** 等效，并与早前的发行版兼容。
v9b	与 **-m64 -xarch=sparcvis2** 等效，并与早期的发行版兼容。

表 3–14 详细介绍了 x86 平台上的每个 **-xarch** 关键字。如果未指定 **-xarch**，x86 上的缺省值为 **generic**（或者如果指定了 **–m64** 则为 **generic64**）。

表 3–14 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。这是缺省值，与 **-xarch=pentium_pro** 等效。
generic64	为了在大多数 64 位 x86 平台上获得良好性能而进行编译。它与 **sse2** 等效。
native	为了在此 x86 体系结构上获得良好性能而进行编译。使用最佳指令集以便在大多数 x86 处理器上获得良好性能。对于每个新发行版，“最佳”指令集的定义可能会有所调整（如果适用）。
native64	为了在此 64 位 x86 体系结构上获得良好性能而进行编译。
386	将指令集限制于 Intel 386/486 体系结构。
pentium_pro	将指令集限制于 Pentium Pro 体系结构。
pentium_proa	将 AMD 扩展（3DNow!、3DNow! 扩展和 MMX 扩展）添加到 32 位 Pentium Pro 体系结构中。
sse	将 SSE 指令集添加到 **pentium_pro** 中。（参见下面的注释。）
ssea	将 AMD 扩展（3DNow!、3DNow! 扩展和 MMX 扩展）添加到 32 位 SSE 体系结构中。
sse2	将 SSE2 指令集添加到 **pentium_pro** 中。（参见下面的注释。）
sse2a	将 AMD 扩展（3DNow!、3DNow! 扩展和 MMX 扩展）添加到 32 位 SSE2 体系结构中。
sse3	向 SSE2 指令集中添加 SSE3 指令集。

表 3–14 x86 平台的 **-xarch** 值 （续）

-xarch=	**含义 (x86)**
amd64	在 Solaris 平台上，这与 **-m64 -xarch=sse2** 等效 使用 **-xarch=amd64** 来获取 64 位内存模型的传统 makefile 和脚本应使用 **-m64**。
amd64a	在 Solaris 平台上，这与 **–m64 –xarch=sse2a** 等效。
sse3a	将 AMD 扩展指令（包括 3DNow!）添加到 SSE3 指令集。
ssse3	将 SSSE3 指令添加到 SSE3 指令集。
sse4_1	将 SSE4.1 指令添加到 SSSE3 指令集。
sse4_2	将 SSE4.2 指令添加到 SSE4.1 指令集。
amdsse4a	将 SSE4a 指令添加到 AMD 指令集。

3.4.110.1 x86/x64 平台的特别注意事项：

当针对 x86 Solaris 平台进行编译时，有几项重要的注意事项。

- 使用设置为 **sse**、**sse2**、**sse2a** 或 **sse3** 等的 **-xarch** 编译的程序必须在支持这些功能和扩展的平台上运行。
- 从 Solaris 9 4/04 开始的操作系统发行版在 Pentium 4 兼容的平台上支持 SSE/SSE2。早期版本的 Solaris OS 不支持 SSE/SSE2。
- 如果在不同的步骤中进行编译和链接，请始终使用编译器和相同的 **-xarch** 设置进行链接，以确保链接正确的启动例程。
- x86 上的运算结果可能与 SPARC 上的结果不同，这是由 x86 80 字节浮点寄存器造成的。为了最大限度减少这些差异，请使用 **-fstore** 选项或使用 **-xarch=sse2** 进行编译（如果硬件支持 SSE2）。
- 从 Solaris Studio 11 和 Solaris 10 OS 开始，会对使用这些专用的 **-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，不进行任何验证，而是由用户负责确保使用这些标志生成的对象已部署在适当的硬件上。
- 如果在没有相应功能或指令集扩展的平台上运行使用这些 **-xarch** 选项编译的程序，则可能会导致段故障或不正确的结果，并且不显示任何显式警告消息。
- 此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（使用 SSE、SSE2、SSE2a 和 SSE3 指令和扩展）的程序。

3.4.111 –xassume_control[=*keywords*]

设置参数以控制 **ASSUME** pragma。

使用此标志可控制编译器对源代码中 **ASSUME** pragma 的处理方式。

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

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

有关 **f95** 编译器可识别的 **ASSUME** pragma 的说明，请参见第 31 页中的 “2.3.1.8 **ASSUME** 指令”。

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

optimize	在 **ASSUME** pragma 上作出的断言会影响程序的优化。
check	编译器生成用于检查标记为“确定”的所有断言是否正确的代码，并在断言被违反时发出运行时消息；如果没有同时指定 **fatal**，则程序继续运行。
fatal	与 **check** 一起使用时，如果标记为“确定”的断言被违反，则程序将终止。
retrospective[:*d*]	*d* 参数是可选的容差值，它必须是小于 1 的正实数常量。缺省值为 ".1"。**retrospective** 编译代码以确定所有断言的真假。那些超出容差值 *d* 的值将在程序终止时的输出中列出。
%none	忽略所有的 **ASSUME** pragma。

编译器的缺省值是

-xassume_control=optimize

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

如果指定不带参数的 **-xassume_control**，则隐含

-xassume_control=check,fatal

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

3.4.112 –xautopar

与 **-autopar** 等效。

3.4.113 –xbinopt={prepare | off}

(SPARC) 准备二进制文件以进行编译后优化。

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

在使用此选项生成时，二进制文件的大小会有所增加，大约增加 5%。

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

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

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

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

缺省值为 **-xbinopt=off**。

3.4.114 –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 是 *i* 级数据高速缓存的大小（千字节）；*li* 是 *i* 级数据高速缓存的行大小（字节）；*ai* 是 *i* 级数据高速缓存的关联性；*ti* 是共享 *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。

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

表 3–15 –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，而不使性能有较大幅度的降低。这是缺省值。

表 3–15 **–xcache** 值 （续）

值	含义
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** 指定以下内容：

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

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

3.4.115 –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：

keyword（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。
no%stkovf	禁用用于检测栈溢出的运行时检查。
init_local	执行特殊的本地变量初始化。 编译器将局部变量初始化为这样一个值：如果在赋予该值之前程序使用了它，则可能会导致运算异常。由 **ALLOCATE** 语句分配的内存也会以这种方式初始化。 不对模块变量、**STATIC** 和 **SAVE** 本地变量及 **COMMON** 块中的变量进行初始化。 有关详细信息，请参见《C 用户指南》。
no%init_local	禁用局部变量初始化。这是缺省值。
%all	打开所有这些运行时检查功能。
%none	禁用所有这些运行时检查功能。

栈溢出（尤其是在栈上分配大数组的多线程应用程序中）可能会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。

3.4.116 –xchip=*c*

为优化器指定目标处理器。

此选项通过指定目标处理器来指定计时属性。

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

下面是 **-xchip=***c* 的一些作用：

- 指令调度
- 编译分支的方式
- 在语义等效的两个选择项之间进行选择

下面的两个表列出了有效的 **-xchip** 处理器名称值：

表 3–16 常用的 **–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。（这是缺省情况。）
native	此主机平台。
sparc64vi	SPARC64 VI 处理器。
sparc64vii	SPARC64 VII 处理器。
ultra	UltraSPARC 处理器。
ultra2	UltraSPARC II 处理器。
ultra2e	UltraSPARC IIe 处理器。
ultra2i	UltraSPARC IIi 处理器。
ultra3	UltraSPARC III 处理器。
ultra3cu	UltraSPARC IIIcu 处理器。
ultra3i	UltraSPARC IIIi 处理器。
ultra4	UltraSPARC IV 处理器。
ultra4plus	UltraSPARC IV+ 处理器。

表 3–16 常用的 **–xchip** SPARC 处理器的名称 （续）

-xchip=	为以下项目进行优化
ultraT1	UltraSPARC T1 处理器。
ultraT2	UltraSPARC T2 处理器。
ultraT2plus	UltraSPARC T2+ 处理器。
ultraT3	UltraSPARC T3 处理器。

在 x86 平台上：-xchip 值为
pentium、**pentium_pro**、**pentium3**、**pentium4**、**generic**、**opteron**、**core2**、**penryn**、**nehalem**、**amdfam10** 和 **native**。

3.4.117 –xcode=*keyword*

(SPARC) 指定 SPARC 平台上的代码地址空间。

keyword 的值包括：

keyword（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 位平台上的缺省值。
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。
abs64	生成 64 位绝对地址。仅在 64 位平台上可用。
pic13	生成与位置无关的代码（小模型）。与 **-pic** 等效。在 32 位平台上，最多允许引用 2**11 个唯一外部符号；在 64 位平台上，最多允许引用 2**10 个。
pic32	生成与位置无关的代码（大模型）。与 **-PIC** 等效。在 32 位平台上，最多允许引用 2**30 个唯一外部符号；在 64 位平台上，最多允许引用 2**29 个。

如果不显式指定 **-xcode=***keyword*，则缺省值是：

–xcode=abs32（32 位平台）；**–xcode=abs44**（64 位平台）。

3.4.117.1 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**。

虽然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），因此会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的指令，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。
- 每次访问全局变量或静态变量时，都会涉及到通过全局偏移表进行额外的间接内存引用。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** 会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_** 的未解析外部引用（用字母 **U** 标记）。

要确定是使用 **-xcode=pic13** 还是 **-xcode=pic32**，请通过使用 **elfdump -c**（有关更多信息，请参见 **elfdump**(1) 手册页）来查看全局偏移表 (Global Offset Table, GOT) 的大小以及节标题 **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，应根据以下准则来确定如何使用 **-xcode**：

- 如果要生成可执行文件，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**。
- 如果要生成共享库，请先使用 **-xcode=pic13**，一旦 GOT 大小超过 8,192 字节，则使用 **-xcode=pic32**。
- 如果要生成用于链接到共享库的归档库，则应该使用 **-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》。

3.4.118 –xcommonchk[={ yes|no}]

启用通用块不一致性的运行时检查。

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

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

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

3.4.119 –xcrossfile[={ 1|0}]

启用跨源文件的优化和内联。（**已过时**）

此选项现在已过时，不应再使用。它与 **-xipo** 的作用相同。应改用 **xipo**。

3.4.120 –xdebugformat={dwarf |stabs}

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

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

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

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

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

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

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

dbx 和性能分析软件可识别 stabs 和 dwarf 格式，因此使用此选项对其中任一工具的功能并无影响。

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

使用 **dwarfdump**(1) 命令确定已编译目标文件或可执行文件中调试信息的格式。

3.4.121 –xdepend

与 **-depend** 等效。

3.4.122 –xF

允许性能分析器进行函数级重组。

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

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

3.4.123 –xfilebyteorder= *options*

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

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

little*max_align*:*spec*

big*max_align*:*spec*

native:*spec*

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

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

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

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

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

spec 必须是包含以下内容的逗号分隔列表：

%all

unit

filename

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

unit 指程序打开的特定 Fortran 单元号。

filename 指程序打开的特定 Fortran 文件名。

3.4.123.1 示例：

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

3.4.123.2 注：

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

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

在此选项中只能声明一次文件名或单元号。

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

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

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

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

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

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

3.4.124 –xhasc[={ yes|no}]

将霍尔瑞斯常量视为实际参数列表中的字符串。

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

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

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

示例：

```
demo% cat hasc.f
                call z(4habcd, 'abcdefg')
                end
                subroutine z(i, s)
                integer i
                character *(*) s
                print *, "string length = ", len(s)
```

```
                return
                end
demo% f95 -o has0 hasc.f
demo% has0
 string length =   4     <-- should be 7
demo% f95 -o has1 -xhasc=no hasc.f
demo% has1
 string length =   7   <-- now correct length for  s
```

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

提供此标记是为了帮助移植传统的 Fortran 77 程序。

3.4.125 –xhelp={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 列出编译器选项标志。与 **-help** 等效。

3.4.126 –xhwcprof[={enable | disable}]

(SPARC) 为数据空间配置启用编译器支持。

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

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

如果分别在单独的步骤中进行编译和链接，最好在链接时使用 **-xhwcprof**。

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

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

-xhwcprof 要求启用优化，并将调试数据格式设置为 dwarf (**-xdebugformat=dwarf**)，这是此发行版 Solaris Studio 的缺省设置。

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

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

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

有关基于硬件计数器的配置的更多信息，请参见《Solaris Studio 性能分析器》手册。

3.4.127 –xia[={ widestneed|strict}]

(Solaris) 启用区间运算扩展并设置合适的浮点环境。

如果未指定此选项，则缺省值为 **-xia=widestneed**。

《Interval Arithmetic Programming Reference》中详细说明了区间运算的 Fortran 扩展。另请参见第 106 页中的 “3.4.130 **–xinterval**[={ *widestneed*|*strict*|*no*}]”。

-xia 标志是一个宏，其扩展如下：

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

3.4.128 –xinline= *list*

与 **-inline** 等效。

3.4.129 –xinstrument=[%no]datarace

指定此选项编译您的程序并为其提供程序设备，以供线程分析器进行分析。

（有关线程分析器的详细信息，请参见 **tha**(1)。）

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

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

必须为 **-xinstrument** 指定一个参数。

如果分步进行编译和链接，则必须在编译和链接步骤中都指定 **-xinstrument=datarace**。

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

此选项还设置 **-g**。

3.4.130 –xinterval[={ *widestneed*|*strict*|*no*}]

(Solaris) 启用区间运算扩展。

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

no	未启用区间运算扩展。
widestneed	将任何混合模式表达式中的所有非区间变量和文字提升为表达式中范围最广的区间数据类型。
strict	禁止混合类型或混合长度区间表达式。所有区间类型和长度转换都必须是显式的。

《Fortran 95 Interval Arithmetic Programming Reference》中详细说明了区间运算的 Fortran 扩展。另请参见第 105 页中的 “3.4.127 –xia[={ widestneed|strict}]”。

3.4.131 –xipo[={ 0|1|2}]

执行过程间的优化。

通过调用一次过程间分析来执行整个程序的优化。与 **-xcrossfile** 不同的是，**-xipo** 将在链接步骤中跨所有目标文件执行优化，并且不仅仅限于编译命令上的源文件。

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

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

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

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

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

在一个编译/链接步骤中使用 **-xipo** 的示例：

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

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

在不同的编译/链接步骤中使用 **-xipo** 的示例：

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

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

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

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

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

关于 **-xipo** 的其他重要信息：

- 至少需要优化级别 **-xO4**
- 与 **-xcrossfile** 冲突；如果一起使用，将导致编译错误
- 如果在构造中使用的目标文件对于并行运行的链接步骤通用，则使用并行生成工具构建采用 **-xipo** 编译的可执行文件会产生问题。每个链接步骤都应有自己的目标文件副本，并且该副本应在链接前进行优化。
- 未使用 **-xipo** 编译的对象可以与使用 **-xipo** 编译的对象自由链接。
- 由于执行跨文件优化时需要附加信息，因此 **-xipo** 选项会生成更大的目标文件。不过，该附加信息不会成为最终的二进制可执行文件的一部分。可执行程序大小的增加是由于执行额外的优化导致的。
- 在此发行版中，跨文件子程序内联是由 **-xipo** 执行的唯一过程间优化。
- **.s** 汇编语言文件不参与过程间分析。
- 如果使用 **-S** 进行编译，则忽略 **-xipo** 标志。

何时不使用 -xipo 进行编译：

在链接步骤中使用目标文件集合时，编译器试图执行整个程序的分析和优化。对于该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，在该目标文件集合外部定义的其他例程将不显式调用 **foo()**

(2) 从该目标文件集合中的任何例程调用 **foo()** 时，将不会受到在该目标文件集合外部定义的不同版本的 **foo()** 的干预。

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

例如，考虑使用您自己的源版本干预 **malloc()** 函数并使用 **-xipo=2** 进行编译。然后，任何库中引用与您的代码链接的 **malloc()** 的所有函数也必须使用 **-xipo=2** 进行编译，并且它们的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，因此您自己的 malloc 不应该使用 **-xipo=2** 进行编译。

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

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

(**SPARC**) 允许跨文件优化包括归档 (**.a**) 库。

值必须是以下项之一：

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

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

3.4.133 –xivdep[= *p*]

禁用或设置 **!DIR$ IVDEP** 指令的解释。

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

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

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

提供这些解释是为了与另一个供应商对 **IVDEP** 指令的解释兼容。

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

有关更多信息，请参见第 33 页中的 “2.3.3 **IVDEP** 指令”。

3.4.134 –xjobs= *n*

使用多个处理器进行编译。

指定 **-xjobs** 选项可设置编译器完成其任务需创建的进程数。在多 CPU 计算机上，该选项可以缩短生成时间。在此发行版的 **f95** 编译器中，**-xjobs** 只能与 **-xipo** 选项一起使用。如果指定 **-xjobs=***n*，过程间优化器就将 *n* 作为其在编译不同文件时可调用的最大代码生成器实例数。

通常，*n* 的安全值等于 1.5 乘以可用虚拟处理器数。由于上下文在产生的作业间切换的开销，使用等于可用虚拟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）。

指定 **-xjobs** 时务必要指定值。否则会发出错误诊断并使编译终止。

在出现最合适的实例之前，**-xjobs** 的多个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的编译，速度比使用相同命令但没有 **-xjobs** 选项时进行的编译快。

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

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

禁止对命名函数 (*name*) 进行与栈相关的优化。

%all－禁止对所有代码进行与栈相关的优化。

%none－允许对所有代码进行与栈相关的优化。

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

3.4.136 –xknown_lib=*library_list*

识别对已知库的调用。

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

library_list 是当前应用于 **blas**、**blas1**、**blas2**、**blas3** 和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，并且能够针对 Sun 性能库实现自由地进行正确优化。编译器将忽略这些库例程的用户提供版本，并链接到 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。与 **-xknown_lib=blas1,blas2,blas3** 等效。
intrinsics	编译器会忽略 Fortran 内部函数的任何显式 **EXTERNAL** 声明，因此将忽略用户提供的任何内例程。（有关内部函数名称列表，请参见《Fortran 库参考》。）

3.4.137 –xl

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

3.4.138 –xlang=f77

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

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

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

使用 **-xlang** 进行编译时，请注意以下事项：

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

3.4.139 –xld

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

3.4.140 –xlibmil

与 **-libmil** 等效。

3.4.141 –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；如果是这样，通常是最后一位不同。该库选项在命令行上的顺序并不重要。

3.4.142 –xlic_lib=sunperf

与 Sun 性能库链接。

例如：

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

与 **-l** 一样，此选项应位于命令行上的所有源文件和目标文件名称之后。

要与 Sun 性能库链接，必须使用此选项。（请参见《Sun Performance Library User's Guide》。）

3.4.143 –xlicinfo

（已过时）**编译器无提示地忽略**。

3.4.144 –xlinkopt[={ 1|2|0}]

(SPARC) 对可重定位目标文件执行链接时优化。

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

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

指定不带值的 **-xlinkopt** 标志即表示 **-xlinkopt=1**。

这些优化在链接时通过分析二进制目标代码来执行。虽然未重写目标文件，但生成的可执行代码可能与初始目标代码不同。

当与分析反馈一起用于编译整个程序时，此选项最有效。

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

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

请注意，仅当编译器链接时才使用级别参数。在以上示例中，即使二进制目标文件是用缺省级别 1 编译的，使用的后优化级别也是 2。

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

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

-xlinkopt 选项将优化位于编译器命令行上的静态库中的代码，但会跳过位于命令行上的共享（动态）库中的代码且不对其进行优化。生成共享库（用 **-G** 编译）时，您也可以使用 **-xlinkopt**。

与运行时分析反馈一起使用时，链接时后优化器最有效。分析功能会展示代码中最常用和最不常用的部分，并指示优化器相应地进行处理。这对大型应用程序尤为重要，因为在链接时执行代码优化放置可降低指令高速缓存未命中数。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
```

有关使用分析反馈的详细信息，请参见 **-xprofile** 选项。

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

3.4.145 –xloopinfo

与 **-loopinfo** 等效。

3.4.146 –xmaxopt[= *n*]

启用优化 pragma 并设置最大优化级别。

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

此选项启用 **!$PRAGMA SUN OPT=***n* 指令（当此指令出现在源输入中时）。不使用此选项时，编译器将这些行视为注释。请参见第 30 页中的 “2.3.1.5 **OPT** 指令”。

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

3.4.147 –xmemalign[= <*a*><*b*>]

(SPARC) 指定未对齐数据访问的最大假定内存对齐和行为。

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

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

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

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

对齐 <*a*> 的允许值有：

1 假定最多 1 字节对齐。

2 假定最多 2 字节对齐。

4 假定最多 4 字节对齐。

8 假定最多 8 字节对齐。

16 假定最多 16 字节对齐。

访问未对齐数据时错误行为 <*b*> 的允许值有：

i 解释访问并继续执行。

s 产生信号 SIGBUS。

f 在 64 位平台上，产生信号 SIGBUS 仅对齐少于或等于 4，否则将解释访问并继续执行。在其他平台上，**f** 与 **i** 等效。

如果在不指定 **-xmemalign** 的情况下进行编译，缺省值为：

- 针对 32 位平台的 **8i**
- 针对采用 C 和 C++ 的 64 位平台的 **8s**
- 针对采用 Fortran 的 64 位平台的 **8f**

对于所有平台，**-xmemalign** 不带值显示时的缺省值为 **1i**。

请注意，**-xmemalign** 本身并不强制进行任何特殊的数据对齐。使用 **-dalign** 或 **-aligncommon** 可强制进行数据对齐。

此外，只要链接到使用 **b** 值 **i** 或 **f** 编译的目标文件，就必须指定 **-xmemalign**。

-dalign 选项是一个宏：

-dalign 是 **-xmemalign=8s -aligncommon=16** 的宏

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

有关详细信息，请参见第 47 页中的 “3.4.1 **–aligncommon**[={ **1**|**2**|**4**|**8**| **16**}]”。

3.4.148 –xmodel=[small | kernel | medium]

(x86) 在 Solaris x64 平台上为共享对象指定数据地址模型。

使用 **-xmodel** 选项，编译器可以为 Solaris x64 平台创建 64 位共享对象，并且只应为此类对象的编译而指定该选项。

仅当在启用了 64 位的 x86 平台 ("x64") 上指定了 **-m64** 时，此选项才有效。

small 此选项按小型模型生成代码，在该模型中，所执行代码的虚拟地址在链接时已知，且确定所有符号都位于 0 到 $2^{31} – 2^{24} - 1$ 范围内的虚拟地址中。

kernel 为内核模型生成代码，在该模型中，所有符号都定义在 $2^{64} - 2^{31}$ 到 $2^{64} - 2^{24}$ 范围内。

medium 按中等模型生成代码，在该模型中， 不对数据段的符号引用范围进行假定。文本段的大小和地址的限制与小型代码模型的限制相同。使用 **–m64** 编译含有大量静态数据的应用程序时，可能需要使用 **–xmodel=medium**。

如果未指定 **-xmodel**，编译器将假定 **-xmodel=small**。如果指定没有参数的 **-xmodel**，将出现错误。

只要您确保所访问的对象位于相应的范围内，就没有必要使用该选项编译所有的例程。

3.4.149 –xnolib

与 **-nolib** 等效。

3.4.150 –xnolibmil

与 **-nolibmil** 等效。

3.4.151 –xnolibmopt

不使用快速数学库。

与 **-fast** 一起使用可以覆盖对优化数学库的链接：

```
f95 -fast -xnolibmopt …
```

3.4.152 –x0*n*

与 **-0***n* 等效。

3.4.153 –xopenmp[={ parallel|noopt|none}]

启用通过 Fortran OpenMP 版本 3.0 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

- 启用对 OpenMP Pragma 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

- 启用对 OpenMP Pragma 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 -xopenmp=noopt**），则编译器将发出错误。如果没有使用 **-xopenmp=noopt** 指定优化级别，则会识别 OpenMP Pragma，并相应地对程序进行并行处理，但不进行优化。
- 定义预处理程序标记 **_OPENMP**。
- 自动调用 **-stackvar**。

none 禁用对 OpenMP Pragma 的识别，并且不更改优化级别。（这是编译器的缺省值。）

指定的无子选项关键字的 **-xopenmp** 与 **-xopenmp=parallel** 等效。**请注意，此缺省值在以后的发行版中可能会有所变化。**

要使用 dbx 调试 OpenMP 程序，请使用 **-g -xopenmp=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 API 用户指南》中对 OpenMP 指令进行了概述。

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。这会将程序可以创建的最大线程数通知给运行时系统。缺省值为 1。通常会将 **PARALLEL** 或 **OMP_NUM_THREADS** 变量设置为目标平台上可用的虚拟处理器数。

要启用嵌套并行操作，必须将 **OMP_NESTED** 环境变量设置为 **TRUE**。缺省情况下，禁用嵌套并行操作。有关嵌套并行性的详细信息，请参见《Solaris 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 是实现所支持的 OpenMP Fortran API 版本的年份和月份名称。对于当前 Solaris Studio 发行版，该值是 200830（表示 OpenMP 3.0 版）。

如果是在不同的步骤中进行编译和链接，还要在链接步骤中指定 **-xopenmp**。在编译包含 OpenMP 指令的库时，这一点尤为重要。

3.4.154 –xpad

与 **-pad** 等效。

3.4.155 –xpagesize= *size*

为栈和堆设置首选页面大小。

在 SPARC 平台上，*size* 值必须是以下值之一：

8K、**64K**、**512K**、**4M**、**32M**、**256M**、**2G**、**16G** 或 **default**

在 x86 平台上，*size* 值必须是以下值之一：

4K、**2M**、**4M** 或 **default**

例如： **-xpagesize=4M**

并非所有这些页面大小在所有平台上都受支持，具体取决于体系结构和 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 **getpagesizes**(3C) 返回。如果不是，此请求在运行时将被忽略。Solaris 环境不保证能够支持页面大小请求。

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

如果指定了 **-xpagesize=default**，该标志将被忽略；如果指定了不带 **size** 值的 *-xpagesize*，则与 **-xpagesize=default** 等效。

此选项是组合使用 **-xpagesize_heap=***size* **-xpagesize_stack=***size* 的宏。这两个选项与 **-xpagesize** 接受相同的参数。可以通过指定 **-xpagesize=***size* 来为二者设置相同的值，或分别为它们指定不同的值。

使用该标志进行编译，与使用等效选项将 **LD_PRELOAD** 环境变量设置为 **mpss.so.1** 或在启动程序之前使用等效选项运行 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。

3.4.156 –xpagesize_heap= *size*

为堆设置首选页面大小。

size 值与所述的 **-xpagesize** 值相同。

有关详细信息，请参见 **-xpagesize**。

3.4.157 –xpagesize_stack= *size*

(SPARC) 为栈设置首选页面大小。

size 值与所述的 **-xpagesize** 值相同。

有关详细信息，请参见 **-xpagesize**。

3.4.158 –xpec[={ yes|no}]

生成 PEC（Portable Executable Code，可移植执行代码）二进制文件。

PEC 二进制文件可用于自动调优系统 (Automatic Tuning System, ATS)。有关 ATS 的更多信息，请访问 `http://cooltools.sunsource.net/ats/index.html`。

使用 **-xpec** 生成的二进制文件通常要比不使用该选项生成的文件大 5 到 10 倍。缺省值为 **-xpec=no**。

不带参数的 **-xpec** 与 **-xpec=yes** 等效。

3.4.159 –xpg

与 **-pg** 等效。

3.4.160 –xpp={fpp| cpp}

选择源文件预处理程序。

缺省值为 **-xpp=fpp**。

编译器使用 **fpp**(1) 来预处理 **.F**、**.F95** 或 **.F03** 源文件。此预处理程序适用于 Fortran。以前的版本使用标准 C 预处理程序 **cpp**。要选择 **cpp**，请指定 **-xpp=cpp**。

3.4.161 –xprefetch[= *a*[,*a*]]

在支持预取的体系结构上启用预取指令。

有关 Fortran **PREFETCH** 指令的说明，请参见第 31 页中的 “2.3.1.7 **PREFETCH** 指令”。

a 必须是以下值之一：

auto 启用预取指令的自动生成

no%auto 禁用预取指令的自动生成

explicit 启用显式预取宏

no%explicit 禁用显式预取宏

latx:*factor* (**SPARC**) 按指定的因子调整编译器的假定预取到装入的延迟和预取到存储的延迟。该因子必须是正浮点数或整数。

如果要在较大的 SPARC 多处理器上运行计算密集的代码，您会发现使用 **-xprefetch=latx:***factor* 有很多优点。该选项指示代码生成器按照指定的因子调节在预取及其相关的装入或存储之间的缺省延迟时间。

预取延迟是从执行预取指令到所预取的数据在高速缓存中可用那一刻之间的硬件延迟。在确定发出预取指令到发出使用所预取数据的装入或存储指令之间的间隔时，编译器就采用预取延迟值。

注 – 在预取和装入之间采用的延迟可能与在预取和存储之间采用的延迟不同。

编译器可以在众多计算机与应用程序间调整预取机制，以获得最佳性能。这种调整并非总能达到最优。对于占用大量内存的应用程序，尤其是要在大型多处理器上运行的应用程序，可以通过增加预取延迟值来提高性能。要增加值，请使用大于 1 的因子。介于 .5 和 2.0 之间的值最有可能提供最佳性能。

对于数据集完全位于外部高速缓存中的应用程序，可以通过减小预取延迟值来提高性能。要减小此值，请使用小于 1 的因子。

要使用 **-xprefetch=latx:***factor* 选项，请首先使用接近 1.0 的因子值并对应用程序运行性能测试。然后适当增加或减小该因子，并再次运行性能测试。继续调整因子并运行性能测试，直到获得最佳性能。以很小的增量逐渐增加或减小因子时，前几步中不会看到性能差异，之后会突然出现差异，然后再趋于稳定。

yes **-xprefetch=yes** 与 **-xprefetch=auto,explicit** 相同

no **-xprefetch=no** 与 **-xprefetch=no%auto,no%explicit** 相同

使用 **-xprefetch**、**-xprefetch=auto** 和 **-xprefetch=yes** 时，编译器就可以将预取指令插入到其生成的代码中。该操作会提高支持预取的体系结构的性能。

3.4.161.1 缺省：

如果未指定 **-xprefetch**，则采用 **-xprefetch=auto,explicit**。

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

如果使用 **-xprefetch** 或 **-xprefetch=yes** 等启用了自动预取，但未指定延迟因子，则假定 **-xprefetch=latx:1.0**。

3.4.161.2 交互：

如果使用 **-xprefetch=explicit**，编译器将能够识别以下指令：

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

-xchip 设置影响假定延迟的决定以及 **latx:***factor* 设置的结果。

仅当在 SPARC 处理器上启用了自动预取 (**auto**) 时，**latx:***factor* 子选项才有效。

3.4.161.3 警告：

显式预取只应在度量支持的特殊环境下使用。

因为编译器可以在众多计算机与应用程序间调整预取机制以获得最佳性能，所以当性能测试指示性能明显提高时，应当只使用 **-xprefetch=latx:***factor*。假定的预取延迟在不同发行版本中是不同的。因此，无论何时切换到不同的发行版本，强烈建议重新测试延迟因子对性能的影响。

3.4.162 –xprefetch_auto_type=indirect_array_access

为间接访问的数据数组生成间接预取。

以生成直接内存访问预取的相同方式来生成由选项 **-xprefetch_level=**{**1**|**2**|**3**} 指定的循环的间接预取。可添加前缀 **no%** 否定声明。

缺省值为 **-xprefetch_auto_type=no%indirect_array_access**。

要求 **-xprefetch=auto** 以及优化级别 **-xO3** 或更高级别。

诸如 **-xdepend** 之类的选项可以影响计算间接预取候选项的主动性，进而影响由于更好的内存别名歧义消除信息而发生的自动间接预取插入的主动性。

3.4.163 –xprefetch_level={ 1|2|3}

控制预取指令的自动生成。

在以下情况下编译时，此选项才有效：

- **-xprefetch=auto**，
- 使用优化级别 3 或更高。
- 在支持预取的平台上。

如果未指定 **-xprefetch_level**，则 **-xprefetch=auto** 的缺省值为级别 2。

预取级别 2 比级别 1 产生更多的预取指令机会。预取级别 3 比级别 2 生成更多的预取指令。

在早期的 SPARC 或 x86 平台上，预取级别 2 和 3 可能不会生效。

3.4.164 –xprofile=*p*

收集用于配置文件的数据或使用配置文件进行优化。

p 必须是 **collect**[**:***profdir*]、**use**[**:***profdir*] 或 **tcov**[**:***profdir*]。

此选项可在执行期间收集并保存执行频率数据，然后在后续运行中可以使用该数据来改进性能。对多线程应用程序来讲，配置文件集合是安全的。也就是说，对执行自身多任务 (**-mt**) 的程序进行文件配置会产生准确的结果。只有指定 **-xO2** 或更高优化级别时，此选项才有效。如果分别执行编译和链接，则链接步骤和编译步骤中必须都出现同一 **-xprofile** 选项。

collect[**:** *profdir*] 在 **-xprofile=use** 时优化器收集并保存执行频率，以供将来使用。编译器生成可测量语句执行频率的代码。

-xMerge、**-ztext** 和 **--xprofile=collect** 不应同时使用。**-xMerge** 会强制将静态初始化的数据存储到只读存储器中，**-ztext** 禁止在只读存储器中进行依赖于位置的符号重定位，而 **-xprofile=collect** 会在可写存储器中生成静态初始化的、依赖于位置的符号重定位。

配置文件目录名 *profdir*（如果指定）是包含已进行文件配置的对象代码的程序或共享库在执行时用来存储配置文件数据的目录的路径名。如果 *profdir* 路径名不是绝对路径，在使用选项 -**xprofile=use:***profdir* 编译程序时将相对于当前工作目录来解释该路径。如果未指定配置文件目录名，配置文件数据将存储在名为 *program***.profile** 的目录中，其中 *program* 是已进行文件配置的主程序的基本名称。

示例[1]：在生成程序所在的同一目录中的 myprof.profile 目录中收集和使用配置文件数据：

```
demo: f95 -xprofile=collect:myprof.profile -xO5 prog.f95 -o prog
demo: ./prog
demo: f95 -xprofile=use:myprof.profile    -xO5 prog.f95 -o prog
```

示例[2]：在目录 /bench/myprof.profile 中收集配置文件数据，然后在优化级别 **-xO5** 的反馈编译中使用收集的配置文件数据：

```
demo: f95 -xprofile=collect:/bench/myprof.profile
\    -xO5 prog.f95 -o prog
...run prog from multiple locations..
demo: f95 -xprofile=use:/bench/myprof.profile
\    -xO5 prog.f95 -o prog
```

可以设置环境变量 **SUN_PROFDATA** 和 **SUN_PROFDATA_DIR** 控制使用 **-xprofile=collect** 编译的程序存储配置文件数据的位置。如果设置了这两个环境变量，**-xprofile=collect** 数据将写入 **$SUN_PROFDATA_DIR/$SUN_PROFDATA**。

这些环境变量同样控制由 **tcov** 写入的配置文件数据文件的路径和名称，如 **tcov**(1) 手册页中所述。如果未设置这些环境变量，配置文件数据就写入当前目录中的目录 *profdir* **.profile**，其中 *profdir* 是可执行文件的名称或在 **-xprofile=collect:** *profdir* 标志中指定的名称。如果 *profdir* 已在 **.profile** 中结束，**-xprofile** 不会将 **.profile** 附加到 *profdir* 中。如果多次运行程序，那么执行频率数据会累积在 *profdir***.profile** 目录中；也就是说，以前执行的输出不会丢失。

如果在不同的步骤中进行编译和链接，应确保使用 **-xprofile=collect** 编译的任何目标文件也使用 **-xprofile=collect** 进行链接。

use[**:** *profdir*]

通过从使用 **-xprofile=collect**[**:***profdir*] 编译的代码中收集的执行频率数据来优化在执行已进行文件配置的代码时执行的工作。*profdir* 是一个目录的路径名，该目录包含运行用 **-xprofile=collect**[**:***profdir*] 编译的程序所收集的配置文件数据。

profdir 路径名是可选的。如果未指定 *profdir*，将使用可执行二进制文件的名称。如果未指定 **-o**，将使用 `a.out`。如果未指定 *profdir*，编译器将查找 **profdir.profile/feedback** 或 *a.out.profile/feedback*。例如：

```
demo: f95 -xprofile=collect    -o myexe prog.f95
demo: f95 -xprofile=use:myexe -xO5 -o myexe    prog.f95
```

程序是使用以前生成并保存在 feedback 文件中的执行频率数据优化的，此数据是先前执行用 **-xprofile=collect** 编译的程序时写入的。

除了 **-xprofile** 选项之外，源文件和其他编译器选项必须与用于编译（该编译过程创建了生成 feedback 文件的编译程序）的相应选项完全相同。编译器的相同版本必须同时用于 **collect** 生成和 **use** 生成。

如果用 **-xprofile=collect:***profdir* 编译，则必须将相同的配置文件目录名 *profdir* 用在优化编译中：**-xprofile=use:** *profdir*。

另请参见 **-xprofile_ircache**，以了解有关加速 **collect** 阶段和 **use** 阶段之间的编译的说明。

tcov[**:** *profdir*]

使用 **tcov**(1) 校验基本块覆盖率分析的目标文件。

如果指定可选的 *profdir* 参数，编译器将在指定位置创建配置文件目录。该配置文件目录中存储的数据可通过 **tcov**(1) 或由编译器通过 **-xprofile=use:***profdir* 来使用。如果省略可选的 *profdir* 路径名，执行已进行文件配置的程序时将创建配置文件目录。只能通过 **tcov**(1) 使用该配置文件目录中存储的数据。使用环境变量 **SUN_PROFDATA** 和 **SUN_PROFDATA_DIR** 可以控制配置文件目录的位置。

如果 *profdir* 指定的位置不是绝对路径名，则编译时相对于要将当前的目标文件写入到的目录来解释该位置。如果为任何目标文件指定了 *profdir*，则必须为同一程序中的所有目标文件指定同一位置。由 *profdir* 指定位置的目录必须在要执行已进行文件配置的程序的所有计算机中都可以访问。除非不再需要配置文件目录中的内容，否则不应删除该目录，因为除非重新编译，否则编译器存储在其中的数据将无法恢复。

示例 [1]：如果用 **-xprofile=tcov:/test/profdata** 编译一个或多个程序的目标文件，编译器会创建一个名为 **/test/profdata.profile** 的目录并将其用来存储描述已进行文件配置的目标文件的数据。该同一目录还可在执行时用来存储与已进行文件配置的目标文件关联的执行数据。

示例 [2]：如果名为 **myprog** 的程序用 **-xprofile=tcov** 编译并在目录 /home/joe 中执行，将在运行时创建目录 **/home/joe/myprog.profile** 并将其用来存储运行时配置文件数据。

3.4.165 –xprofile_ircache[=*path*]

(SPARC) 保存并重用收集阶段和使用配置文件阶段之间的编译数据。

在使用阶段，与 **-xprofile=collect|use** 一起使用会重用收集阶段保存的编译数据，从而可以减少编译时间。

如果指定，*path* 将覆盖高速缓存文件的保存位置。缺省情况下，这些文件会与目标文件保存在同一目录下。当收集阶段和使用阶段出现在两个不同的位置时，指定路径便十分有用。

典型的命令序列可能是：

```
demo% f95 -xO5 -xprofile=collect -xprofile_ircache t1.c t2.c
demo% a.out      collects feedback data
demo% f95 -xO5 -xprofile=use -xprofile_ircache t1.c t2.c
```

对于大程序，通过用这种方式保存中间数据，可以显著缩短使用阶段的编译时间。但这将以显著增加所用的磁盘空间为代价。

3.4.166 –xprofile_pathmap= *collect_prefix*:*use_prefix*

(SPARC) 设置配置文件数据文件的路径映射。

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

如果编译器找不到用 **-xprofile=use** 编译的目标文件的配置文件数据，请使用 **-xprofile_pathmap**，并且：

- 当前使用 **-xprofile=use** 进行编译所用的目录，不是先前使用 **-xprofile=collect** 编译时使用的目录。
- 目标文件会共享配置文件中的公共基名，但可以根据它们在不同目录中的位置互相区分。

collect-prefix 是目录树的 UNIX 路径名的前缀，该目录树中的目标文件是使用 **-xprofile=collect** 编译的。

use-prefix 是目录树的 UNIX 路径名的前缀，该目录树中的目标文件是使用 **-xprofile=use** 编译的。

如果指定了 **-xprofile_pathmap** 的多个实例，编译器将按照这些实例的出现顺序对其进行处理。将 *-xprofile_pathmap* 实例指定的每个 **use-prefix** 与目标文件路径名进行比较，直至找到匹配的 *use-prefix* 或发现最后一个指定的 *use-prefix* 与目标文件路径名也不匹配。

3.4.167 –xrecursive

允许不带 **RECURSIVE** 属性的例程以递归方式调用它们自己。

通常，只有使用 **RECURSIVE** 属性定义的子程序才能以递归方式调用它们自己。

使用 **-xrecursive** 进行编译时，子程序将可以调用自身，即使它们没有使用 **RECURSIVE** 属性进行定义也是如此。但是，与定义了 **RECURSIVE** 的子例程不同，缺省情况下，使用此标志不会导致在栈上分配局部变量。要使局部变量在子程序的每个递归调用中具有不同的值，还应该使用 **-stackvar** 进行编译以便将局部变量放在栈上。

间接递归（例程 A 调用例程 B，而例程 B 又调用例程 A）可在高于 **-xO2** 的优化级别上生成不一致的结果。使用 **-xrecursive** 标志进行编译，可以保证使用间接递归的正确性，即使优化级别更高也是如此。

使用 **-xrecursive** 进行编译会导致性能下降。

3.4.168 –xreduction

与 **-reduction** 等效。

3.4.169 –xregs= *r*

为生成的代码指定寄存器用法。

r 是一个逗号分隔列表，它包含下面的一个或多个子选项：`appl`、`float`、`frameptr`。

用 `no%` 作为子选项的前缀会禁用该子选项。

请注意，`-xregs` 子选项仅限于特定的硬件平台。

示例：`-xregs=appl,no%float`

表 3–17 -xregs 子选项

值	含义
appl	(SPARC) 允许编译器将应用程序寄存器用作临时寄存器来生成代码。应用程序寄存器是： g2、g3 或 g4（在 32 位平台上） g2 或 g3（在 64 位平台上） 强烈建议使用 -xregs=no%appl 编译所有系统软件和库。系统软件（包括共享库）必须为应用程序保留这些寄存器值。这些值的使用将由编译系统控制，而且在整个应用程序中必须保持一致。 在 SPARC ABI 中，这些寄存器表示为**应用程序**寄存器。由于需要更少的装入和存储指令，因此使用这些寄存器可提高性能。但是，这样使用可能与某些用汇编代码编写的旧库程序冲突。
float	(SPARC) 允许编译器通过将浮点寄存器用作整数值的临时寄存器来生成代码。使用浮点值可能会用到与该选项无关的这些寄存器。如果希望代码没有任何对浮点寄存器的引用，需要使用 -xregs=no%float 并确保代码不以任何方式使用浮点类型。
frameptr	(x86) 允许编译器将帧指针寄存器（IA32 上为 %ebp，AMD64 上为 %rbp）用作通用寄存器。 缺省值为 -xregs=no%frameptr 通过 -xregs=framptr，编译器可以自由使用帧指针寄存器来提高程序性能。但是，这可能会限制调试器和性能测量工具的某些功能。栈跟踪、调试器和性能分析器不能对通过 -xregs=frameptr 编译的函数生成报告。 如果直接调用或从 C 或 Fortran 函数间接调用的 C++ 函数会引发异常，不应该用 -xregs=frameptr 编译 C、Fortran 和 C++ 混合代码。如果使用 -fast 编译此类混合源代码，请在命令行中的 -fast 选项后添加 -xregs=no%frameptr。 由于 64 位平台上的可用寄存器更多，因此相对于 64 位代码相比，使用 -xregs=frameptr 编译更容易改进 32 位代码的性能。 如果同时指定了 -xpg，编译器会忽略 -xregs=frameptr 并发出警告。

SPARC 缺省值为 -xregs=appl,float。

x86 缺省值为 -xregs=no%frameptr。-xregs=frameptr 包含在 -fast 的扩展中。

强烈推荐您用 -xregs=no%appl,float 编译那些用于与应用程序链接的共享库的代码。至少共享库应该显式说明它如何使用应用程序寄存器，以便与这些库链接的应用程序知道这些寄存器分配。

例如，在某种全局意义上使用寄存器（例如，使用寄存器指向一些关键数据结构）的应用程序，需要确切地知道其代码未使用 `-xregs=no%appl` 编译的某个库如何使用应用程序寄存器，以便安全地与该库链接。

3.4.170 –xs

允许 **dbx** 在没有目标 (**.o**) 文件的情况下进行调试。

如果使用 **-xs**，则所有调试信息将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** 并忽略目标 (**.o**) 文件。如果不能保留 **.o** 文件，请使用该选项。

如果不使用 **-xs**，在移动可执行文件的情况下，您必须同时移动源文件和目标 (**.o**) 文件，或者使用 dbx **pathmap** 或 **use** 命令设置路径。

3.4.171 –xsafe=mem

(SPARC) 允许编译器假定未发生违反内存保护的情况。

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

仅当与优化级别 `-xO5` 及以下 `-xarch` 值中的一个一起使用时，此选项才能有效：`sparc`、`sparcvis`、`sparcvis2` 或 `sparcvis3`（用于 `-m32` 和 `-m64`）。

注意 – 由于在发生诸如地址未对齐或段违规的故障时，无故障装入不会导致陷阱，因此您应该只对不会发生此类故障的程序使用该选项。因为只有很少的程序会导致基于内存的陷阱，所以您可以安全地将该选项用于大多数程序。对于显式依赖基于内存的陷阱来处理异常情况的程序，请勿使用该选项。

3.4.172 –xsb

（已过时）与 **-sb** 等效。

3.4.173 –xsbfast

（已过时）与 **-sbfast** 等效。

3.4.174 –xspace

不执行会增加代码大小的优化。

示例：如果增加代码大小，则不会解开循环或并行化循环。

3.4.175 –xtarget=*t*

为指令集和优化指定目标平台。

t 必须是以下值之一： **native**、**native64**、**generic**、**generic64** 和 *platform–name*。

-xtarget 选项允许简便快捷地指定发生在实际平台上的 **-xarch**、**-xchip** 和 **-xcache** 组合。**-xtarget** 的唯一含义在其扩展中。

通过为编译器提供目标计算机硬件的精确描述，某些程序的性能可得到提高。当程序性能很重要时，目标硬件的正确指定是非常重要的。在较新的 SPARC 处理器上运行时，尤其是这样。不过，对大多数程序和较旧的 SPARC 处理器来讲，性能的提高微不足道，因此指定 **generic** 就足够了。

-xtarget 值的实际扩展可能会因发行版的不同而异。通过使用 **-dryrun** 标志，您始终可以确定编译器将使用的扩展：

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

请注意，针对特定名称平台的 **–xtarget** 扩展不得与同一平台上的 **–xtarget=native** 相同。

3.4.175.1 通用平台与本机平台

native	优化主机平台（32 位）的性能。 扩展至 **–m32 –xarch=native –xchip=native –xcache=native**
native64	已过时。改用 **–xtarget=native –m64**。
generic	为大多数 32 位平台获得最佳性能。 这是缺省值并可扩展至： **-m32 -xarch=generic -xchip=generic -xcache=generic**
generic64	已过时。改用 **-xtarget=generic -m64**。
platform–name	获取以下列出的指定平台的最佳性能。

3.4.175.2 SPARC 平台

下表列出了编译器接受的常用系统平台的名称。

表 3–18 常用的 **-xtarget** 系统平台的扩展

-xtarget= *platform-name*	-xarch	-xchip	-xcache
sparc64vi	**sparcfmaf**	**sparc64vi**	**128/64/2:5120/64/10**
sparc64vii	**sparcima**	**sparc64vii**	**64/64/2:5120/256/10**
ultra	**sparcvis**	**ultra**	**16/32/1:512/64/1**
ultra1/140	**sparcvis**	**ultra**	**16/32/1:512/64/1**
ultra1/170	**sparcvis**	**ultra**	**16/32/1:512/64/1**
ultra1/200	**sparcvis**	**ultra**	**16/32/1:512/64/1**
ultra2	**sparcvis**	**ultra2**	**16/32/1:512/64/1**
ultra2/1170	**sparcvis**	**ultra**	**16/32/1:512/64/1**
ultra2/1200	**sparcvis**	**ultra**	**16/32/1:1024/64/1**
ultra2/1300	**sparcvis**	**ultra2**	**16/32/1:2048/64/1**
ultra2/2170	**sparcvis**	**ultra**	**16/32/1:512/64/1**
ultra2/2200	**sparcvis**	**ultra**	**16/32/1:1024/64/1**
ultra2/2300	**sparcvis**	**ultra2**	**16/32/1:2048/64/1**
ultra2e	**sparcvis**	**ultra2e**	**16/32/1:256/64/4**
ultra2i	**sparcvis**	**ultra2i**	**16/32/1:512/64/1**
ultra3	**sparcvis2**	**ultra3**	**64/32/4:8192/512/1**
ultra3cu	**sparcvis2**	**ultra3cu**	**64/32/4:8192/512/2**
ultra3i	**sparcvis2**	**ultra3i**	**64/32/4:1024/64/4**
ultra4	**sparcvis2**	**ultra4**	**64/32/4:8192/128/2**
ultra4plus	**sparcvis2**	**ultra4plus**	**64/32/4/1:2048/64/4/2:32768/64/4/2**
ultraT1	**sparc**	**ultraT1**	**8/16/4/4:3072/64/12/32**
ultraT2	**sparcvis2**	**ultraT2**	**8/16/4:4096/64/16**
ultraT2plus	**sparcvis2**	**ultraT2plus**	**8/16/4:4096/64/16**
ultraT3	**sparcvis3**	**ultraT3**	**8/16/4:6144/64/24**

-m64 标志表示针对启用了 64 位的平台上的 64 位 Solaris OS 进行编译。如果指定了 **-xtarget**，则 **-m64** 必须出现在 **-xtarget** 标志之后，如下所示：

```
-xtarget=ultra2   ...   -m64
```

否则，将使用缺省的 32 位内存模型。

3.4.175.3 X86 平台

对于 x86 系统，有效的 **-xtarget** 平台名称如下：

generic、**native**、**pentium**、**pentium_pro**、**pentium3**、**pentium4**、**woodcrest**、**penryn**、**nehalem**、**barcelona** 和 **opteron**。

表 3–19 -xtarget 值（x86 平台）

-xtarget=	-xarch	-xchip	-xcache
generic	generic	generic	generic
opteron	sse2	opteron	64/64/2:1024/64/16
pentium	386	pentium	generic
pentium_pro	pentium_pro	pentium_pro	generic
pentium3	sse	pentium3	16/32/4:256/32/4
pentium4	sse2	pentium4	8/64/4:256/128/8
nehalem	sse4_2	nehalem	32/64/8:256/64/8:8192/64/16
penryn	sse4_1	penryn	2/64/8:4096/64/16
woodcrest	ssse3	core2	32/64/8:4096/64/16
barcelona	amdsse4a	amdfam10	64/64/2:512/64/16

-m64 标志表示针对启用了 64 位的 x86 平台上的 64 位 Solaris OS 进行编译。例如，使用 **-xtarget=opteron** 进行编译是不必要的或不够的。如果指定了 **-xtarget**，则 **-m64** 选项必须出现在 **-xtarget** 标志之后，如下所示：

```
-xtarget=opteron -m64
```

否则，编译将为 32 位 x86。

3.4.176 –xtime

与 **-time** 等效。

3.4.177 –xtypemap= *spec*

指定缺省数据映射。

此选项提供了一种为缺省数据类型指定字节大小的灵活方法。此选项适用于缺省大小的变量和常量。

规范字符串 *spec* 可以包含以下任何或全部项（采用逗号分隔列表形式）：

real:*size***double:** *size***integer:** *size*

每个平台上允许的组合包括：

- **real:32**
- **real:64**
- **double:64**
- **double:128**
- **integer:16**
- **integer:32**
- **integer:64**

例如：

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

可同时将缺省的 **REAL** 和 **DOUBLE** 映射到 8 字节。

此选项适用于使用缺省规范（不带显式字节大小）声明的所有变量，例如 **REAL XYZ**（生成 64 位 **XYZ**）。此外，所有的单精度 **REAL** 常量将被提升为 **REAL*8** 常量。

请注意，**INTEGER** 和 **LOGICAL** 的处理方式相同，**COMPLEX** 映射为两个 **REAL**。此外，**DOUBLE COMPLEX** 将按照 **DOUBLE** 的映射方式进行处理。

3.4.178 –xunroll=*n*

与 **-unroll=***n* 等效。

3.4.179 –xvector[= [[no%]lib, [no%] simd, %none]]

启用对向量库函数调用的自动生成。

-xvector 选项需要 **-xO3** 或更高的优化级别。如果优化级别未指定或低于 **-xO3**，编译将不会继续，同时会发出消息。

在使用 **-xvector** 进行编译时，此选项要求使用缺省舍入模式 **-fround=nearest** 进行编译。

-xvector=lib（仅适用于 Solaris 平台），允许编译器将循环内的数学库调用转换为对等效向量数学例程的单个调用（如果能够进行此类转换的话）。此类转换可提高那些循环计数较大的循环的性能。**-xvector=no%lib** 将禁用此功能。

-xvector=simd 指示编译器使用本机 x86 SSE SIMD 指令来提高某些循环的性能。在 x86 中，缺省情况下以可产生有利结果的优化级别 3 和更高级别使用流扩展。可以使用子选项 no%simd 将其禁用。

仅当目标体系结构中存在流扩展（即目标 ISA 至少为 SSE2）时，编译器才会使用 SIMD。例如，可在现代平台中指定 -xtarget=woodcrest、-xarch=generic64、-xarch=sse2、-xarch=sse3 或 -fast 来使用它。如果目标 ISA 没有流扩展，子选项将无效。

如果您同时指定了 **-xvector=simd** 和 **-fsimple=2**，则可以获得比单独指定 **-xvector=simd** 更好的性能。但是，浮点结果可能会稍有不同，因为 **-fsimple=2** 允许对浮点运算重新排序。

在 x86 平台上的缺省值为 -xvector=simd，在 SPARC 平台上的缺省值为 -xvector=%none。如果指定不带子选项的 -xvector，在 x86、SPARC (Solaris) 和 Linux 上，编译器将分别采用 -xvector=simd,lib、-xvector=lib 和 -xvector=simd。

在装入步骤中，编译器会包含 libmvec 库。如果您在编译时指定了 **-xvector=lib**，则还必须在链接时指定它。

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

3.4.180 –ztext

在不重定位的情况下仅生成纯库。

-ztext 的一般用途是验证所生成的库是否为纯文本；指令都是与位置无关的代码。因此，它通常与 **-G** 和 **-pic** 一起使用。

使用 **-ztext** 时，如果 **ld** 在 *text* 段中找到了不完整的重定位，则不会生成库。如果它在 *data* 段中找到了不完整的重定位，则通常会生成库；数据段是可写入的。

不使用 **-ztext** 时，**ld** 会生成库，与有无重定位无关。

如果您不知道目标文件是否是使用 **-pic** 生成的，则一种典型用法是利用源文件和目标文件生成库。

示例：利用源文件和目标文件生成库：

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

另一种用法是确认代码是否与位置无关：不带 **-pic** 进行编译，但确认是否为纯文本。

示例：确认是否为纯文本－即使不带 **-pic** 进行编辑：

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

选项 **-ztext** 和 **-xprofile=collect** 不应同时使用。**-ztext** 会阻止只读存储中与位置有关的符号重定位，而 **-xprofile=collect** 会在可写存储中生成静态初始化且与位置有关的符号重定位。

如果使用 **-ztext** 进行编译时 **ld** 不生成库，则可以在不使用 **-ztext** 的情况下重新编译，此时 **ld** 将生成库。使用 **-ztext** 生成失败意味着无法对库的一个或多个组件共享，不过，也许能共享某些其他组件。此时就产生了性能问题，这最好由您－程序员来解决。

◆ ◆ ◆ 第 4 章

4 Solaris Studio Fortran 的功能与差异

本附录介绍了标准 Fortran 与 Solaris Studio Fortran 编译器 **f95** 之间的一些主要功能差异。

4.1 源语言功能

f95 编译器提供标准 Fortran 的以下源语言功能和扩展。

4.1.1 续行限制

f95 允许 999 个续行（1 个初始行和 999 个续行）。对于固定格式，标准 Fortran 允许 19 个续行；对于自由格式，允许 39 个续行。

4.1.2 固定格式源代码行

在固定格式源代码中，行的长度可以超过 72 个字符，但忽略第 73 列以后的任何内容。标准 Fortran 95 仅允许 72 个字符长的行。

4.1.3 制表符格式

f95 固定格式源代码文本的定义如下：

- 如果第 1 列至第 6 列的任一列中有制表符，都会使该行成为制表符格式的源代码行。
- 制表符前面可以有注释指示符或语句编号。
- 如果制表符是第一个非空字符，会出现以下情况：
  - 如果制表符后面的字符不是非零数字，则制表符后面的文本是初始行。

- 如果第一个制表符后面是非零数字，则该行是续行。非零数字后面的文本是语句的下一个部分。

对于固定格式，**f95** 的缺省最大行长度是 72 列；对于自由格式，则为 132 列。使用 **-e** 编译器选项，可将固定格式源代码中的行扩展到 132 列。

示例：左侧的制表符格式源代码的处理方式如右侧所示。

```
!^IUses of tabs                              !       Uses of tabs
^ICHARACTER *3 A = 'A'                               CHARACTER *3 A = 'A'
^IINTEGER B = 2                                      INTEGER B = 2
^IREAL C = 3.0                                       REAL C = 3.0
^IWRITE(*,9) A, B, C                                 WRITE(*,9) A, B, C
9^IFORMAT(1X, A3,                            9       FORMAT(1X, A3,
^I1 I3,                                             1 I3,
^I2 F9.1 )                                          2 F9.1 )
^IEND                                                END
```

在以上示例中，"**^I**" 代表制表符，以 "1" 和 "2" 开头的行是续行。显示此代码的目的在于说明各种制表符情形，而不是提倡任一样式。

f95 中的制表符会强制填充剩余的行直到第 72 列。如果制表符出现在持续到下一行的字符串中，则可能会引发意外结果：

源文件：

```
^Iprint *, "Tab on next line
^I1this  continuation line starts with a tab."
^Iend
```

运行代码：

```
Tab on next line                                          this  continuation
 line starts with a tab.
```

此外，对于制表符格式允许使用 **–f77** 选项。

4.1.4 采用的源代码格式

f95 采用的源代码格式取决于选项、指令和后缀。

具有 **.f** 或 **.F** 后缀的文件采用固定格式。带 **.f90**、**.f95**、**.F90** 或 **.F95** 后缀的文件假定采用自由格式。

表 4–1 F95 源代码格式命令行选项

选项	操作
-fixed	将所有源文件解释为 Fortran **固定**格式
-free	将所有源文件解释为 Fortran **自由**格式

如果使用 **-free** 或 **-fixed** 选项，则它覆盖文件名后缀。如果使用 **!DIR$ FREE** 或 **!DIR$ FIXED** 指令，则它覆盖选项和文件名后缀。

4.1.4.1 混合格式

允许混合使用某些源代码格式。

- 在同一个 **f95** 命令中，一些源文件可以是固定格式，而另一些源文件可以是自由格式。
- 在同一个文件中，**可以**通过使用 **!DIR$ FREE** 和 **!DIR$ FIXED** 指令将自由格式与固定格式混合使用。
- 在同一个程序单元中，**可以**将制表符格式与自由格式或固定格式混合使用。

4.1.4.2 大小写

缺省情况下，Solaris Studio Fortran 不区分大小写。这意味着，变量 **AbcDeF** 的处理方式与将其拼写为 **abcdef** 时相同。要让编译器区别处理大写字母和小写字母，请使用 **-U** 选项进行编译。

4.1.5 限制和缺省值

- 一个 Fortran 程序单元最多可定义 65,535 个派生类型和 16,777,215 个不同的常量。
- 变量和其他对象的名称最多可包含 127 个字符，标准长度为 31 个字符。

4.2 数据类型

本节介绍 Fortran 数据类型的功能和扩展。

4.2.1 布尔类型

f95 支持布尔类型的常量和表达式。但是，没有布尔变量或数组，也没有布尔类型语句。

4.2.1.1 控制布尔类型的各种规则

- 对于屏蔽操作，按位逻辑表达式具有布尔结果；它的每个位都是对相应操作数位进行一个或多个逻辑运算的结果。
- 用于二进制算术运算符和关系运算符：
  - 如果一个操作数是布尔型，则在执行运算时不进行转换。
  - 如果两个操作数均是布尔型，则在执行运算时就当它们是整数一样。
- 用户指定的函数均不能生成布尔结果，但某些（非标准的）内部函数可以。
- 布尔和逻辑类型具有以下差异：
  - 变量、数组和函数可以是逻辑类型，但它们不能是布尔类型。
  - 可以使用 **LOGICAL** 语句，但不能使用 **BOOLEAN** 语句。
  - 逻辑型的变量、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（位 29、30 和 31）的内容。
- 如果位数不足 11 个，则该值右对齐，即它表示字最右侧的位：位 n 至 31。其他位为 0。
- 忽略空格。

在 I/O 格式规范中，字母 **B** 表示**二进制**数字；而在其他地方则表示**八进制**数字。

十六进制

X'*ddd***'** 或 **X"***ddd* **"**，其中 *d* 是任意十六进制数字

- 可以包含 1 至 8 个十六进制数字（0 至 9，**A**-**F**）。
- 任何字母既可以是大写也可以是小写（**X**、**x**、**A**-**F**、**a**-**f**）。

- 数字必须用撇号或引号括起来。
- 忽略空格。
- 十六进制数字可以以 + 或 - 符号开头。
- 8 个十六进制数字表示一个完整的 32 位字，等效的二进制数字对应于 32 位字中每个位的内容。
- 如果位数不足 8 个，则该值右对齐，即它表示字最右侧的位：位 n 至 31。其他位为 0。

霍尔瑞斯

接受的霍尔瑞斯数据格式为：

*n*H…	'…'H	"…"H
*n*L…	'…'L	"…"L
*n*R…	'…'R	"…"R

上面的 "…" 是字符串，*n* 是字符数。

- 霍尔瑞斯常量是布尔类型。
- 如果任何字符常量是按位逻辑表达式，则该表达式的计算结果为霍尔瑞斯型。
- 霍尔瑞斯常量可以包含 1 至 4 个字符。

示例：八进制和十六进制常量。

布尔常量	32 位字的内部八进制数
0B	00000000000
77740B	00000077740
X"ABE"	00000005276
X"-340"	37777776300
X'1 2 3'	00000000443
X'FFFFFFFFFFFFFFFF'	37777777777

示例：赋值语句中的八进制和十六进制数。

```
i = 1357B
j = X"28FF"
k = X'-5A'
```

在算术表达式中使用八进制或十六进制常量可产生未定义的结果，并且不会生成语法错误。

4.2.1.3 布尔常量的替代上下文

f95 允许在非 **DATA** 语句中使用 BOZ 常量。

B'*bbb*'	**O**'*ooo*'	**Z**'*zzz*'
B"*bbb*"	**O**"*ooo*"	**Z**"*zzz*"

如果将它们赋值给实数变量，则不进行类型转换。

标准 Fortran 只允许在 **DATA** 语句中使用它们。

4.2.2 数值数据类型的缩写大小表示法

f95 允许在声明语句、函数语句和 **IMPLICIT** 语句中使用以下非标准的类型声明格式。第一列中的格式虽然已被广泛使用，但它们是非标准的 Fortran 格式。第二列中的种类数字可能会因供应商不同而异。

表 4–2 数值数据类型的大小表示法

非标准	声明符	简短形式	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）

表 4–2 数值数据类型的大小表示法 （续）

非标准	声明符	简短形式	含义
`COMPLEX*32`	`COMPLEX(KIND=16)`	`COMPLEX(16)`	四精度复数（每个部分 16 个字节）

4.2.3 数据类型的大小和对齐

存储和对齐始终以字节为单位。可以划分为单字节的值按字节对齐。

类型的大小和对齐取决于各种编译器选项和平台以及变量的声明方式。COMMON 块中的缺省最大对齐位置是 4 字节边界。

使用特殊选项（如 **-aligncommon**、**-f**、**-dalign**、**-dbl_align_all**、**- xmemalign** 和 **-xtypemap**）进行编译，可以更改缺省的数据对齐和存储分配。本手册中的缺省描述假定这些选项无效。

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

下表汇总了缺省的大小和对齐，并忽略类型和选项的其他方面。

表 4–3 缺省的数据大小和对齐（以字节为单位）

Fortran 数据类型	大小	缺省对齐	COMMON 中的对齐
`BYTE X`	1	1	1
`CHARACTER X`	1	1	1
`CHARACTER*`*n* `X`	n	1	1
`COMPLEX X`	8	4	4
`COMPLEX*8 X`	8	4	4
`DOUBLE COMPLEX X`	16	8	4
`COMPLEX*16 X`	16	8	4
`COMPLEX*32 X`	32	8/16	4
`DOUBLE PRECISION X`	8	8	4
`REAL X`	4	4	4
`REAL*4 X`	4	4	4
`REAL*8 X`	8	8	4
`REAL*16 X`	16	8/16	4

表 4–3 缺省的数据大小和对齐（以字节为单位） （续）

Fortran 数据类型	大小	缺省对齐	COMMON 中的对齐
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**：在 64 位环境（使用 **-m64** 进行编译）中，缺省对齐位置是 16 字节（而非 8 字节）边界，如表中 8/16 所示。该数据类型通常称为**四精度**。
- 数组和结构按照其元素或字段对齐。数组对齐方式与数组元素相同。结构对齐方式与具有最宽对齐的字段相同。

选项 **-f** 或 **-dalign** 可强制将所有 8、16 或 32 字节数据与 8 字节边界对齐。选项 **-dbl_align_all** 可使所有数据与 8 字节边界对齐。如果程序依赖于这些选项的使用，则可能无法进行移植。

4.3 Cray 指针

Cray **指针**是一个变量，其值是另一个实体（称为**指针对象**）的地址。

f95 支持 Cray 指针；标准 Fortran 95 不支持。

4.3.1 语法

Cray **POINTER** 语句采用以下格式：

```
POINTER  ( pointer_name, pointee_name [array_spec] ), ...
```

其中，*pointer_name*、*pointee_name* 和 *array_spec* 如下所示：

pointer_name 指向相应 *pointee_name* 的指针。

pointer_name 包含 *pointee_name* 的地址。必须是一个标量变量名称（但不是派生类型）。不能是常量、结构名称、数组或函数。

pointee_name	指向相应 *pointer_name* 的指针对象 必须是：变量名、数组声明符或数组名称
array_spec	如果 *array_spec* 存在，则它必须是显形（常量或非常量边界）或者假定大小。

例如，可以声明指向两个指针对象的 Cray 指针：

```
POINTER ( p, b ),  ( q, c )
```

以上示例声明 Cray 指针 **p** 及其指针对象 **b** 以及 Cray 指针 **q** 及其指针对象 **c**。

还可以声明指向数组的 Cray 指针：

```
POINTER ( ix, x(n, 0:m) )
```

以上示例声明 Cray 指针 **ix** 及其指针对象 **x**；并将 **x** 声明为 **n** x **(m+1)** 维数组。

4.3.2 Cray 指针的用途

通过将变量与存储块中的特定位置动态关联起来，您可以使用指针访问用户管理的存储。

Cray 指针允许访问绝对内存地址。

4.3.3 声明 Cray 指针和 Fortran 95 指针

Cray 指针声明如下：

POINTER (*pointer_name*, *pointee_name* [*array_spec*] **)**

Fortran 95 指针声明如下：

POINTER *object_name*

不能混用这两种类型的指针。

4.3.4 Cray 指针的功能

- 无论何时引用指针对象，**f95** 均使用当前的指针值作为指针对象的地址。
- Cray 指针类型语句声明指针和指针对象。
- Cray 指针为 Cray 指针类型。

- 在 32 位处理器中，Cray 指针的值占用一个存储单元；在 64 位处理器中，Cray 指针的值占用两个存储单元。
- Cray 指针可以出现在 **COMMON** 列表中，也可以作为哑元参数。
- 在定义 Cray 指针的值之前，Cray 指针对象没有地址。
- 如果将数组命名为指针对象，则该数组称为**指针对象数组**。

  其数组声明符可以出现在：

  - 单独的类型语句
  - 单独的 **DIMENSION** 语句中
  - 指针语句本身

  如果数组声明符在子程序中，则维数赋值可以引用：

  - COMMON 块中的变量或
  - 作为哑元参数的变量

  每个维的大小是在进入子程序时计算的，而不是在引用指针对象时计算的。

4.3.5 Cray 指针的限制

- *pointee_name* 不得是类型为 **CHARACTER*(*)** 的变量。
- 如果 *pointee_name* 是数组声明符，则它必须是显形（常量或非常量边界）或假定大小。
- 不允许使用 Cray 指针数组。
- Cray 指针不能：
  - 是由另一个 Cray 指针或 Fortran 指针指向的指针。
  - 是结构的组件。
  - 声明为任何其他数据类型。

  Cray 指针不能出现在：
  - **PARAMETER** 语句或包含 **PARAMETER** 属性的类型声明语句中。
  - **DATA** 语句中。

4.3.6 Cray 指针对象的限制

- Cray 指针对象不能出现在 **SAVE**、**DATA**、**EQUIVALENCE**、**COMMON** 或 **PARAMETER** 语句中。
- Cray 指针对象不能是哑元参数。
- Cray 指针对象不能是函数值。
- 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** 是块公用区中长度为 1000 的数组
- **b** 跟在 **a** 的后面，其长度为 **n**
- **c** 跟在 **b** 的后面
- **a**、**b** 和 **c** 与 **pool** 相关联
- **word64** 与 **blk(17)** 相同，因为 Cray 指针是字节地址，而且 **blk** 的每个整数元素都是 4 字节长

4.4 STRUCTURE 和 UNION (VAX Fortran)

为帮助迁移用旧版 FORTRAN 77 编写的程序，**f95** 接受 VAX Fortran **STRUCTURE** 和 **UNION** 语句，它是 Fortran 95 中"派生类型"的前身。有关语法的详细信息，请参见《FORTRAN 77 Language Reference》手册。

STRUCTURE 中的字段声明可以是以下内容之一：

- 子结构－另一个 **STRUCTURE** 声明或一个先前定义的记录。
- **UNION** 声明。
- **TYPE** 声明，它可以包含初始值。
- 具有 **SEQUENCE** 属性的派生类型。（这是 **f95** 所特有的。）

与传统 **f77** 编译器相同，**POINTER** 语句不能用作域声明。

f95 还允许：

- "**.**" 或 "**%**" 均可用作结构域非关联化符号：**struct.field** 或 **struct%field**。
- 结构可以出现在格式化 I/O 语句中。
- 可以在 **PARAMETER** 语句中初始化结构；格式与派生类型初始化相同。
- 结构可以作为派生类型中的组件，但必须使用 **SEQUENCE** 属性声明派生类型。

4.5 无符号整数

Fortran 编译器接受新的数据类型 **UNSIGNED** 作为对该语言的一种扩展。**UNSIGNED** 接受四个 **KIND** 参数值：1、2、4 和 8，分别对应于 1、2、4 和 8 字节无符号整数。

无符号整型常量的形式是：数字串后跟大写或小写字母 **U**，再后跟一个下划线和种类参数（可选）。下面的示例显示了无符号整型常量的最大值：

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

如果没有使用种类参数 (**12345U**)，则缺省值与缺省整数相同。缺省值为 **U_4**，但可以使用 **-xtypemap** 选项更改它，这会更改缺省无符号整数的种类类型。

可使用 **UNSIGNED** 类型说明符声明无符号整型变量或数组：

```
UNSIGNED U
UNSIGNED(KIND=2) :: A
UNSIGNED*8 :: B
```

4.5.1 算术表达式

- 二进制运算（如 **+ - * /**）不能混合使用有符号操作数和无符号操作数。即，如果将 **U** 声明为 **UNSIGNED** 并且 **N** 是有符号的 **INTEGER**，则 **U*N** 是非法的。
  - 可使用 **UNSIGNED** 内部函数将二进制运算中的混合操作数组合起来，例如 **U*UNSIGNED(N)**。
  - 有一种例外情况是，如果一个操作数是无符号整数，而另一个操作数是具有正值或零值的有符号整数常量表达式，则结果是无符号整数。
  - 此类混合表达式的结果的类型，是操作数最常见的类型。

  有符号值的幂是有符号的值；而无符号值的幂是无符号的值。
- 无符号值的一元负值是无符号的值。
- 无符号操作数可以与实数、复数操作数任意混合使用。（无符号操作数不能与区间操作数混合使用。）

4.5.2 关系表达式

可以使用内部关系运算来比较有符号和无符号整型操作数。其结果基于未修改的操作数的值。

4.5.3 控制构造

- CASE 构造接受无符号整数作为条件表达式。
- 不允许将无符号整数作为 DO 循环控制变量，也不允许在算术 IF 控制表达式中使用它。

4.5.4 输入/输出构造

- 可以使用 I、B、O 和 Z 编辑描述符来读取和写入无符号整数。
- 还可使用列表式 I/O 和名称列表 I/O 读取和写入无符号整数。列表式 I/O 或名称列表 I/O 下的无符号整数的写入格式与带符号的正整数使用的格式相同。
- 也可以使用未格式化 I/O 读取或写入无符号整数。

4.5.5 内部函数

- 允许在内部函数中使用无符号整数，但 **SIGN** 和 **ABS** 除外。
- 新的内部函数 **UNSIGNED** 与 **INT** 类似，但生成无符号类型的结果。格式为

UNSIGNED(*v* [**,***kind*] **)**.

- 另一个新的内部函数 **SELECTED_UNSIGNED_KIND(** *var***)** 返回 *var* 的种类参数。
- 内部函数不允许同时使用有符号整型操作数和无符号整型操作数，但 **MAX** 和 **MIN** 函数除外。仅当至少有一个 **REAL** 类型的操作数时，才允许这两个函数同时使用有符号整型操作数和无符号整型操作数。
- 无符号数组不能作为数组内部函数的参数。

4.6 Fortran 200x 的功能

此发行版的 Solaris Studio Fortran 编译器中引入了 Fortran 2003 标准中的许多新功能。有关详细信息，请参阅 Fortran 2003 标准。同时也引入了 Fortran 2008 草案标准中所建议的一些功能。有关这些功能的详细信息，请参见相应的草案出版物。

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（相反的功能是，一种指定可从 C 函数中引用 Fortran 子程序的方法），以及
- 一种声明与外部 C 变量相链接的全局变量的方法

ISO_C_BINDING 模块提供了对命名常量的访问，这些命名常量是种类类型参数，它们代表了与 C 类型兼容的数据。

该标准还引入了 **BIND(C)** 属性。如果 Fortran 派生类型具有 **BIND** 属性，则它可以与 C 之间进行互操作。

此发行版的 Fortran 编译器实现了该标准第 15 章中描述的这些功能。如标准中第 4 章所述，Fortran 还提供了用于定义与 C 类型对应的派生类型和枚举的工具。

4.6.2 IEEE 浮点异常处理

在 Fortran 语言中，新的内部模块 **IEEE_ARITHMETIC** 和 **IEEE_FEATURES** 提供了对异常和 IEEE 算法的支持。对这些功能提供完整支持的是：

```
USE, INTRINSIC :: IEEE_ARITHMETIC

USE, INTRINSIC :: IEEE_FEATURES
```

INTRINSIC 关键字是 Fortran 2003 中的新增功能。这些模块定义了一组派生类型、常量、舍入模式、查询函数、基本函数、种类函数、基本和非基本子例程。有关详细信息，请参见 Fortran 2003 标准的第 14 章。

4.6.3 命令行参数内部函数

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

- **GET_COMMAND(***command, length, status***)**
  以 *command* 返回调用该程序的整个命令行。
- **GET_COMMAND_ARGUMENT(***number, value, length, status***)**
  以 *value* 返回命令行参数。
- **GET_ENVIRONMENT_VARIABLE(***name, value, length, status, trim_name***)**
  返回环境变量的值。

4.6.4 PROTECTED 属性

现在，Fortran 编译器接受 Fortran 2003 的 **PROTECTED** 属性。**PROTECTED** 对模块实体的使用进行了限制。具有 **PROTECTED** 属性的对象只能在声明这些对象的模块中定义。

4.6.5 Fortran 2003 异步 I/O

编译器可识别 I/O 语句中的 **ASYNCHRONOUS** 说明符：

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

此语法是在 Fortran 2003 标准第 9 章中提出的。在与 **WAIT** 语句结合使用的情况下，允许程序员指定可能与计算重叠的 I/O 进程。虽然编译器可以识别 **ASYNCHRONOUS='YES'**，但标准不要求实际的异步 I/O。在本发行版的编译器中，I/O 始终是同步的。

4.6.6 扩展的 ALLOCATABLE 属性

Fortran 2003 扩展了 **ALLOCATABLE** 属性所允许的数据实体。以前，仅限本地存储的数组变量使用该属性。现在，允许将它用于：

- 结构的数组组件
- 哑元数组
- 数组函数结果

在可分配实体可能与存储关联的所有位置，仍然禁止使用可分配实体： **COMMON** 块和 **EQUIVALENCE** 语句。可分配数组组件可以以 **SEQUENCE** 类型出现，但在 **COMMON** 和 **EQUIVALENCE** 中禁止使用此类型的对象。

4.6.7 VALUE 属性

f95 编译器接受 Fortran 2003 **VALUE** 类型声明属性。

如果使用此属性指定子程序哑元输入参数，则表明“按值”传递实际参数。以下示例说明如何将 **VALUE** 属性用于一个 C 主程序，该主程序将文字值作为参数来调用 Fortran 子程序：

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

Fortran code:
       subroutine to_fortran(i)
       integer, value :: i
       print *, i
       end
```

4.6.8 Fortran 2003 流 I/O

Fortran 2003 标准定义了一个新的“流”I/O 方案。流 I/O 访问将数据文件作为可按从 1 开始的正整数编址的持续字节序列来处理。可以连接数据文件用于有格式访问或无格式访问。

可以在 **OPEN** 语句中使用 **ACCESS='STREAM'** 说明符来声明流 I/O 文件。字节地址文件定位要求 **READ** 或 **WRITE** 语句中有 **POS=***scalar_integer_expression* 说明符。**INQUIRE** 语句接受 **ACCESS='STREAM'**、说明符 **STREAM=***scalar_character_variable* 和 **POS=***scalar_integer_variable*。

4.6.9 Fortran 2003 格式化 I/O 功能

在 **f95** 中，已实现了三个新的 Fortran 2003 格式化 I/O 说明符。它们可以出现在 **OPEN**、**READ**、**WRITE**、**PRINT** 和 **INQUIRE** 语句中：

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

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

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

  为格式化 I/O **D**、**E**、**EN**、**ES**、**F** 和 **G** 编辑设置缺省舍入模式。如果使用 **'COMPATIBLE'**，数据转换后的值是与两个近似值较近的那个值，如果值正好在两者中间，则是离 0 远的值。如果使用 **'PROCESSOR_DEFINED'**，舍入模式取决于处理器的缺省模式，如果未指定 **ROUND**，则为编译器缺省设置。

  例如，**WRITE(*,'(f11.4)')** **0.11115** 在缺省模式下打印 **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** 语句中的缺省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** 输出项中打印逗号而不是句点。

要了解如何更改格式化 I/O 的浮点舍入模式，另请参见 **-iorounding** 编译器命令行选项。（第 67 页中的 “3.4.45 **–iorounding**[={ **compatible**|**processor-defined**}]”）

4.6.10 Fortran 2003 IMPORT 语句

IMPORT 语句指定主机作用域单元中可由主机关联访问的实体。仅在接口主体中允许使用。

4.6.11 Fortran 2003 FLUSH I/O 语句

f95 编译器接受 Fortran 2003 **FLUSH** 语句。**FLUSH** 语句使写入外部文件的数据可用于其他进程；通过除 Fortran 以外的其他方法使外部文件中的数据可用于 **READ** 语句。

4.6.12 Fortran 2003 POINTER INTENT 功能

现在，Fortran 编译器支持 **POINTER** 哑元参数的 **INTENT** 属性：可以为指针 dummy 指定 **INTENT(IN)**、**INTENT(OUT)** 或 **INTENT(INOUT)**。

例如，

```
subroutine sub(P)
integer, pointer, intent(in) :: p
...

end
```

指针的 **INTENT** 属性应用于指针，而非指针所指向的元素，因此对于 **INTENT(IN)** 指针，以下语句是非法的，因为这些语句将修改指针：

```
 p => t
 allocate(p)
 deallocate(p)
```

但以下语句对于 **INTENT(IN)** 指针是合法的，因为它修改指针所指向的元素：

```
 p = 400
```

4.6.13 Fortran 2003 中增强的数组构造函数

数组构造函数中允许使用方括号代替 (/ 和 /)：

```
X = [ 3.2, 4.01, 6.5 ]
```

Fortran 2003 标准允许使用方括号作为数组构造函数。这可能会与区间常量冲突。如果在不带 **-xia** 选项（或用于启用区间运算的类似选项）的情况下使用方括号，这些方括号将被视为数组构造函数。如果使用 **-xia** 选项，这些方括号将被视为常量。区间用户应继续使用 (/ 和 /) 数组构造函数，以避免出现编译错误。

4.6.14 Fortran 2003 和 Fortran 2008 的其他功能

有关下列 Fortran 2003 功能的详细信息，请参阅已发布的 Fortran 2003 标准。发布的 Fortran 200x 草案文档中介绍了 Fortran 2008 的功能。

- 2003 对可分配数组的扩展－赋值重新分配及可分配的标量
- 2003 对 **ALLOCATE/DEALLOCATE** 语句的扩展－ **ERRMSG** 和 **SOURCE**
- 2003 对 **MOVE_ALLOC** 内部函数的扩展
- 2003 对带重映射的指针赋值的扩展

- 2003 扩展：**MIN/MAX**、**MIN/MAXVAL** 和 **MIN/MAXLOC**，使用字符参数。
- 2003 内部函数 **IS_IOSTAT_END**、**IS_IOSTAT_EOR**、**NEW_LINE**
- 2003 内部函数 **SELECTED_CHAR_KIND**
- 内部函数 **SYSTEM_CLOCK** 的 **COUNT_RATE** 参数的 2003 **REAL** 类型
- 2003 对复合 **SQRT** 内部函数结果的新限制
- 2008：使用空指针作为缺少的可选参数
- x86 平台上的 IEEE 内部模块支持。
- 2008 的位内部函数：`BGE, BGT, BLE, BLT, DSHIFTL, DSHIFTR, LEADZ, POPCNT, POPPAR, TRAILZ, MASKL, MASKR, SHIFTA, SHIFTL, SHIFTR, MERGE_BITS, IALL, IANY, IPARITY`

4.7 其他的 I/O 扩展

本节介绍一些 Fortran 95 输入/输出处理扩展，**f95** 编译器接受这些扩展，但它们不是 Fortran 2003 标准的一部分。某些扩展是在 Fortran 77 编译器 **f77** 中出现的 I/O 扩展，现在这些扩展已成为 Fortran 编译器的一部分。

4.7.1 I/O 错误处理例程

通过两个新函数，用户可以为逻辑单元上的格式化输入指定自己的错误处理例程。当检测到格式错误时，运行时 I/O 库会调用指定的由用户提供的处理程序例程，同时将数据指向输入行中导致错误的字符。处理程序例程可以提供一个新字符，并允许 I/O 操作在检测到错误的点上使用新字符继续运行；或者采用缺省的 Fortran 错误处理操作。

新例程 **SET_IO_ERR_HANDLER(**3f**)** 和 **GET_IO_ERR_HANDLER(**3f**)** 是模块子例程，这两个例程要求在调用它们的例程中使用 **USE SUN_IO_HANDLERS**。要详细了解这些例程，请参见手册页。

4.7.2 变量格式表达式

Fortran 77 允许用尖括号括起的任意表达式来代替具有某种格式的任何整数常量：

1 FORMAT(… **<** *expr* **>** … **)**

变量格式表达式不能作为 *n*H… 编辑描述符中的 *n* 出现在 **ASSIGN** 语句引用的 **FORMAT** 语句中，或者出现在并行区域内的 **FORMAT** 语句中。

这种功能是在 **f95** 中自动启用的，并且不要求使用 **-f77** 兼容性选项标志。

4.7.3 NAMELIST 输入格式

- 输入中的组名前面可以是 **$** 或 **&**。**&** 是 Fortran 95 标准接受的唯一格式，并且是 **NAMELIST** 输出所写入的内容。
- 接受 **$** 作为用来终止输入的符号，但以下情况除外：组中最后一个数据项是 **CHARACTER** 数据，此时将 **$** 作为输入数据处理。
- 允许 **NAMELIST** 输入从记录的第一列开始。

4.7.4 二进制未格式化 I/O

使用 **FORM='BINARY'** 打开文件与使用 **FORM='UNFORMATTED'** 具有大致相同的效果，所不同的是文件中没有嵌入记录长度。如果没有此数据，则无法知道一条记录的开始或结束位置。因此，无法对 **FORM='BINARY'** 文件执行 **BACKSPACE** 操作，这是因为不知道要退格到什么位置。对 **'BINARY'** 文件执行 **READ** 操作时，将按需要读取尽可能多的数据来填充输入列表中的变量。

- **WRITE** 语句：以二进制的形式将数据写入文件，并按输出列表中指定的数量传输字节。
- **READ** 语句：将数据读取到输入列表中的变量，并传输该列表所要求数量的字节。因为文件中没有记录标记，所以不进行“记录结束”错误检测。检测到的唯一错误是“文件结束”或异常系统错误。
- **INQUIRE** 语句：对使用 **FORM="BINARY"** 打开的文件执行 **INQUIRE** 返回：

  FORM="BINARY"ACCESS="SEQUENTIAL"DIRECT="NO"FORMATTED="NO"UNFORMATTED="YES"RECL= AND NEXTREC= are undefined
- **BACKSPACE** 语句：不允许使用－返回一个错误。
- **ENDFILE** 语句：在当前位置照常截断文件。
- **REWIND** 语句：将文件照常重新定位到数据的开头。

4.7.5 各种 I/O 扩展

- 在不同单元上可能出现的递归 I/O（这是因为 **f95** I/O 库为 "MT-Warm"）。
- **RECL=2147483646** (2^{31}-2) 是顺序格式化输出、列表式输出和名称列表输出中的缺省记录长度。
- 可以按《FORTRAN 77 Language Reference》手册中所述识别和实现 **ENCODE** 和 **DECODE**。
- 非前进式 I/O 是使用 **ADVANCE='NO'** 启用的，如下所示：

```
write(*,'(a)',ADVANCE='NO') 'n= ' read(*,*) n
```

4.8 指令

编译器**指令**指示编译器执行某些特殊的操作。指令又称 *pragma*。

可以将编译器指令作为一个或多个文本行插入到源程序中。每一行看起来就像注释一样，但其中包含的其他字符可将其标识为不仅仅是该编译器的注释。对于大多数其他编译器，它被处理为注释，因此具有一定的代码移植性。

有关 Fortran 指令的完整摘要，请参见附录 C，Fortran 指令摘要。

4.8.1 特殊 f95 指令行的格式

除了第 20 页中的 “1.8 命令行帮助”中介绍的指令外，**f95** 还可识别其自己的特殊指令。这些指令使用以下语法：

```
!DIR$ d1, d2, ...
```

4.8.1.1 固定格式源代码

- 将 **CDIR$** 或 **!DIR$** 放在第 1 至第 5 列中。
- 指令在第 7 列及后面的列中列出。
- 忽略第 72 列后面的列。
- **初始**指令行的第 6 列为空。
- **连续**指令行的第 6 列非空。

4.8.1.2 自由格式源代码

- 将后跟空格的 **!DIR$** 放在行中的任意位置。

  !DIR$ 字符是行中的第一个非空字符（实际上就是非空白）。
- 指令在空格后面列出。
- 在**初始**指令行中，紧跟在 **!DIR$** 之后的位置中为空格、制表符或换行符。
- 在**连续**指令行中，紧跟在 **!DIR$** 之后的位置中为空格、制表符或换行符以外的字符。

因此，第 1 至第 5 列中的 **!DIR$** 既用于自由格式源代码又用于固定格式源代码。

4.8.2 FIXED 和 FREE 指令

这些指令指定指令行后面行的源代码格式。

4.8.2.1 作用域

它们适用于所在**文件**的其余部分，或者在遇到下一个 **FREE** 或 **FIXED** 指令之前的部分。

4.8.2.2 用法

- 它们用于切换源文件中的源代码格式。
- 它们用于切换 **INCLUDE** 文件的源代码格式。您可以将指令插入在 **INCLUDE** 文件的开头。在处理 **INCLUDE** 文件后，源代码格式恢复为处理 **INCLUDE** 文件之前使用的格式。

4.8.2.3 限制

FREE/FIXED 指令：

- 每个指令必须单独出现在编译器指令行中（没有续行）。
- 每个指令可以出现在源代码中的任意位置。其他指令必须出现在它们所影响的程序单元中。

示例：一个 **FREE** 指令。

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

4.8.3 并行化指令

并行化指令是一种特殊的注释，它指示编译器尝试并行处理下一个 DO 循环。附录 D 中对这些指令进行概述，《Fortran 编程指南》中有关并行化的一章介绍了这些指令。Sun 和 Cray 样式并行化指令现已过时，应首选使用 OpenMP Fortran API 指令和并行化模型。《OpenMP API 用户指南》中介绍了 OpenMP 并行化。

4.9 模块文件

在编译包含 Fortran 95 **MODULE** 的文件时，会为在源代码中遇到的每个 **MODULE** 生成模块接口文件（**.mod** 文件）。文件名是从 **MODULE** 的名称中派生的；将为 **MODULE xyz** 创建文件 **xyz.mod**（全部小写）。

编译还会为包含 **MODULE** 语句的源文件生成 **.o** 模块实现目标文件。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* 标志或 **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，则编译器在当前工作目录中写入 **.mod** 文件。

在编译 **USE** *modulename* 语句时，编译器在当前工作目录中查找接口文件。使用 **-M***path* 选项，可以为编译器指定其他搜索路径。必须在链接步骤的命令行中显式列出模块实现目标文件。

通常，程序员为每个文件定义一个 **MODULE**，并为 **MODULE** 及包含它的源文件指定相同的名称。但是，这并不是必需的。

在本示例中，同时编译所有的文件。模块源文件在主程序使用它们之前就已出现。

```
demo% cat mod_one.f90
 MODULE one
    ...
 END MODULE
demo% cat mod_two.f90
 MODULE two
    ...
 END MODULE
demo% cat main.f90
 USE one
 USE two
   ...
 END
demo% f95 -o main mod_one.f90 mod_two.f90 main.f90
```

编译创建以下文件：

mainmain.oone.modmod_one.otwo.modmod_two.o

下一个示例单独编译每个单元，并将它们链接在一起。

```
demo% f95 -c mod_one.f90 mod_two.f90
demo% f95 -c main.f90
demo% f95 -o main main.o mod_one.o mod_two.o
```

在编译 **main.f90** 时，编译器在当前目录中搜索 **one.mod** 和 **two.mod**。必须先编译这些文件，再编译在 **USE** 语句中引用模块的任何文件。链接步骤要求模块实现目标文件 **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

在发行的 7.0 版的 Fortran 编译器中，可以将 **.mod** 文件存储在归档 (**.a**) 文件中。要在归档中搜索模块，必须在命令行的 **-M***path* 标志中显式指定它。在缺省情况下，编译器并不搜索归档文件。

仅搜索与 **USE** 语句中出现的名称同名的 **.mod** 文件。例如，Fortran 语句 **USE mymod** 使编译器缺省搜索模块文件 **mymod.mod**。

在搜索过程中，编译器为在其中写入模块文件的目录指定更高的优先级。可以使用 **-moddir=***dir* 选项标志和 **MODDIR** 环境变量对此进行控制。这意味着，如果仅指定了 **-M***path* 选项，则首先搜索当前目录，然后再搜索 **-M** 标志上列出的目录和文件。

4.9.2 -use=*list* 选项标志

-use=*list* 标志强制将一个或多个隐式 **USE** 语句添加到每个使用该标志编译的子程序或模块子程序中。通过使用该标志，使得在库或应用程序的某个功能要求使用模块或模块文件时，不必修改源程序。

使用 **-use=***module_name* 进行编译，可将 **USE** *module_name* 语句添加到正在编译的每个子程序或模块中。使用 **-use=***module_file_name* 进行编译，与为 *module_file_name* 文件中包含的每个模块添加 **USE** *module_name* 具有相同的效果。

4.9.3 fdumpmod 命令

使用 **fdumpmod**(1) 命令可显示有关已编译模块信息文件的内容的信息。

```
demo% fdumpmod x.mod group.mod
x 1.0 v8,i4,r4,d8,n16,a4 x.mod
group 1.0 v8,i4,r4,d8,n16,a4 group.mod
```

fdumpmod 命令将在单个 **.mod** 文件、通过串联 **.mod** 文件形成的文件以及 **.mod** 文件的 **.a** 归档中显示有关模块的信息。显示的内容包含模块名称、版本号、目标体系结构，以及用来指示与模块兼容的编译选项的标志。有关详细信息，请参见 **fdumpmod**(1) 手册页。

4.10 内部函数

f95 支持某些内在过程，它们是超出标准的扩展。

表 4–4 非标准的内部函数

名称	定义	函数类型	参数类型	参数	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

Notes: P: 名称可以作为参数传递。E: 在运行时调用内部函数的外部代码。

有关内部函数（包括 Fortran 编译器可识别的 Fortran 77 内部函数）的更完整论述，请参见《Fortran 库参考》。

4.11 向前兼容性

以后的 **f95** 发行版在源代码上将与此发行版兼容。

不能保证此发行版的 **f95** 所生成的模块信息文件与以后的发行版兼容。

4.12 混合语言

可以将使用 C 编写的例程与 Fortran 程序结合使用，因为这些语言具有通用的调用约定。有关 C 和 Fortran 例程之间如何进行交互操作的详细信息，请参见《Fortran 编程指南》的“C/Fortran 接口”一章。

◆ ◆ ◆ 第 5 章

5 FORTRAN 77 兼容性：迁移到 Solaris Studio Fortran

不再提供单独的 Fortran 77 编译器。Solaris Studio Fortran 编译器 **f95** 可编译大多数的旧版 FORTRAN 77 程序，其中包括采用以前由 Sun WorkShop **f77** 编译器编译的非标准扩展的程序。

f95 可直接接受很多 FORTRAN 77 功能。其他功能要求在 FORTRAN 77 兼容性模式 (**f95 -f77**) 下进行编译。

本章介绍 **f95** 接受的 FORTRAN 77 功能，并列出了那些与 **f95** 不兼容的 **f77** 功能。有关 Sun WorkShop **f77** 编译器所接受的任何非标准 FORTRAN 77 扩展的详细信息，请参见 `http://docs.sun.com/source/806-3594/index.html` 上的旧版《FORTRAN 77 Language Reference》手册。

有关 **f95** 编译器所接受的其他 Fortran 语言扩展的信息，请参见第 4 章，Solaris Studio Fortran 的功能与差异。

f95 可编译符合标准的 FORTRAN 77 程序。要确保持续的可移植性，使用非标准 FORTRAN 77 功能的程序应该迁移到符合标准的 Fortran 95/2003。使用 **-ansi** 选项进行编译，会标出程序中所有不符合标准的用法。

5.1 兼容的 f77 功能

f95 直接接受或在 **-f77** 兼容性模式下进行编译时接受旧版 FORTRAN 77 编译器 **f77** 的以下非标准功能：

源代码格式

- 续行可以在第一列中以 '**&**' 开头。[**-f77=misc**]
- include 文件中的第一行可以是续行。[**-f77=misc**]
- 使用 **f77** 制表符格式。[**-f77=tab**]
- 制表符格式可以将源代码行扩展到第 72 列以后。[**-f77=tab]**

- 如果字符串扩展到续行上，则 **f95** 制表符格式不会将字符串一直填充到第 72 列。[**-f77**]

I/O：

- 在 Fortran 95 中，可以使用 **ACCESS='APPEND'** 打开文件。
- 列表式输出使用与 **f77** 编译器类似的格式。[**-f77=output**]
- **f95** 允许对直接访问文件使用 **BACKSPACE**，但不允许使用 **ENDFILE**。
- **f95** 允许在格式编辑描述符中隐式指定字段宽度。例如，允许使用 **FORMAT(I)**。
- **f95** 可识别输出格式中的 **f77** 转义序列（例如，**\n \t \'**）。[**-f77=backslash**.]
- **f95** 可识别 **OPEN** 语句中的 **FILEOPT=**。
- **f95** 允许使用 **STATUS='KEEP'** 来关闭 **SCRATCH** 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** 文件。
- 允许对内部文件使用直接 I/O。[**-f77**]
- **f95** 可识别 FORTRAN 77 格式编辑描述符 **A**、**$** 和 **SU**。[**-f77**]
- **FORM='PRINT'** 可以出现在 **OPEN** 语句中。[**-f77**]
- **f95** 可识别传统的 FORTRAN 输入/输出语句 **ACCEPT** 和 **TYPE**。
- 使用 **-f77=output** 进行编译可写入 FORTRAN 77 样式的 **NAMELIST** 输出。
- 在仅指定 **ERR=**（没有 **IOSTAT=** 或 **END=** 分支）的情况下，当检测到 EOF 时，**READ** 将 **ERR=** 分支作为 **END=** 处理。[**-f77**]
- 在 **OPEN** 语句中，接受 VMS Fortran **NAME='***filename* **'**。[**-f77**]
- **f95** 接受 **READ()** 或 **WRITE()** 后面的一个额外逗号。[**-f77**]
- **END=** 分支可以出现在具有 **REC=** 的直接访问 **READ** 中。[**-f77=input**]
- 允许使用格式编辑描述符 **E***w*.*d*.*e*，并将其作为 **E***w*.*d*.**E***e* 处理。[**-f77**]
- 可以在输入语句的 FORMAT 中使用字符串。[**-f77=input**]
- **IOSTAT=** 说明符可以出现在 **ENCODE/DECODE** 语句中。
- 在 **ENCODE/DECODE** 语句中允许使用列表式 I/O。
- 在 I/O 语句中用作逻辑单元时，星号 (*****) 可用于表示 **STDIN** 和 **STDOUT**。
- 数组可以出现在 **FMT=** 说明符中。[**-f77=misc**]
- **PRINT** 语句接受名称列表组名称。[**-f77=output**]
- 编译器接受 **FORMAT** 语句中多余的逗号。
- 在执行 **NAMELIST** 输入时，如果输入问号 (**?**)，则会以正在读取的名称列表组的名称响应。[**-f77=input**]

数据类型、声明和用法：

- 在程序单元中，**IMPLICIT** 语句可以放在单元中任何其他声明语句的后面。
- **f95** 接受 **IMPLICIT UNDEFINED** 语句。

- **f95** 接受 **AUTOMATIC** 语句（FORTRAN 77 扩展）。
- **f95** 接受 **STATIC** 语句，处理方式与 **SAVE** 语句一样。
- **f95** 接受 VAX **STRUCTURE**、**UNION** 和 **MAP** 语句。（请参见第 146 页中的 “4.4 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（如 **DO WHILE**）中。[**-f77=logical**]
- Cray 指针可以出现在内部函数调用中。
- **f95** 可接受在类型声明中使用斜杠的数据初始化。例如： **REAL MHW/100.101/, ICOMX/32.223/**
- **f95** 允许将 Cray 字符指针赋值给非指针变量，以及其他非字符指针的 Cray 指针。
- **f95** 允许将同一个 Cray 指针指向不同类型大小（例如，**REAL*8** 和 **INTEGER*4**）的项。
- 在将 Cray 指针声明为 **POINTER** 的同一程序单元中，可以将其声明为 **INTEGER**。**INTEGER** 声明将被忽略。[**-f77=misc**]
- Cray 指针可以用于除法和乘法运算。[**-f77=misc**]
- **ASSIGN** 语句中的变量可以是 **INTEGER*2** 类型。[**-f77=misc**]
- 交替的 **RETURN** 语句中的表达式可以是非整型。[**-f77=misc**]
- 具有 **SAVE** 属性的变量可以与 **COMMON** 块的元素等效。
- 相同数组的初始化函数可以是不同的类型。示例： **REAL*8 ARR(5) /12.3 1, 3, 5.D0, 9/**
- 名称列表项的类型声明可以放在 **NAMELIST** 语句后面。
- **f95** 接受 **BYTE** 数据类型。
- **f95** 允许将非整数用作数组下标。[**-f77=subscript**]
- **f95** 允许将关系运算符 **.EQ.** 和 **.NE.** 用于逻辑操作数。[**-f77=logical**]
- **f95** 可接受传统的 **f77 VIRTUAL** 语句，并将它作为 **DIMENSION** 语句处理。
- 可以采用与 **f77** 编译器兼容的方式，对不同的数据结构进行等效处理。[**-f77=misc**]
- 与 **f77** 编译器类似，**f95** 允许在 **PARAMETER** 语句的初始化表达式中出现许多内部函数。
- **f95** 允许将整数值赋值给 **CHARACTER*1** 变量。[**-f77=misc**]
- BOZ 常量可用作指数。[**-f77=misc**]
- 可以将 BOZ 常量赋值给字符变量。例如： **character*8 ch ch ="12345678"X**
- BOZ 常量可以用作内部函数调用的参数。[**-f77=misc**]
- 可以使用 **DATA** 语句中的整数值来初始化字符变量。变量中的第一个字符将被设置为该整数值，而字符串的其余部分将填充空白（如果字符串多于一个字符的话）。
- 可以将霍尔瑞斯字符的整数数组用作格式描述符。[**-f77**].

- 如果常量折叠产生浮点异常，则在编译时不进行常量折叠。[**-f77=misc**]
- 在使用 **-f77=misc** 进行编译时，**f95** 将以 **f77** 编译器的方式，将赋值、数据和参数语句中的 **REAL** 常量自动提升为适当的种类（**REAL*8** 或 **REAL*16**）。[**-f77=misc**]
- 在赋值 GOTO 中允许使用等价变量。[**-f77**]
- 可以将非常量字符表达式赋值给数值变量。
- 如果使用 **-f77=misc** 进行编译，则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** 语句具有 *name*。
- **CALL** 语句可以将函数当作子例程进行调用。[**-f77**]
- 对函数不一定要定义其返回值。[**-f77**]
- 交替返回说明符（***label*** 或 **&*label***）可以出现在实际参数列表中，也可以出现在不同位置上。[**-f77=misc**]
- **%VAL** 可用于类型为 **COMPLEX** 的参数。[**-f77=misc**]
- 可以使用 **%REF** 和 **%LOC**。[**-f77=misc**]
- 子例程可以递归调用其自身，而无需使用 **RECURSIVE** 关键字对自身进行声明。[**-f77=misc**] 但是，还应该使用 **-xrecursive** 标志对执行间接递归（例程 A 调用例程 B，而例程 B 又调用例程 A）的程序进行编译以使其正常工作。
- 即使当哑元参数列表中没有交替返回列表时，也可以调用具有交替返回的子例程。
- 如果使用 **-f77=misc** 进行编译，可以使用非 **INTEGER** 或 **REAL** 类型的参数来定义语句函数；实际参数将转换为语句函数所定义的类型。[**-f77=misc**]
- 允许使用空实际参数。例如： **CALL FOO(I,,,J)** 在第一个 **I** 和最后一个 **J** 参数之间使用了两个空参数。
- **f95** 将函数 **%LOC()** 调用作为 **LOC()** 调用处理。[**-f77=misc**]
- 允许在其他运算符（如 **** 或 ***）后面使用一元加号和一元减号。
- 即使第一个参数的类型是 **COMPLEX**，也允许 **CMPLX()** 内部函数使用第二个参数。在这种情况下，使用第一个参数的实部。[**-f77=misc**]
- 允许 **CHAR()** 内部函数的参数超过 255，并且只生成警告而不生成错误。[**-f77=misc**]
- 允许移位计数为负，并且只生成警告而不生成错误。
- 在当前目录中以及 **-I** 选项指定的目录中搜索 **INCLUDE** 文件。[**-f77=misc**]
- 允许进行连续的 **.NOT.** 运算，例如 **.NOT..NOT..NOT.(I.EQ.J)**。[**-f77=misc**]

其他

- **f95** 通常不会将进度消息发送到标准输出。而 **f77** 编译器则发送进度消息，并显示它所编译的例程的名称。在使用 **-f77** 兼容性标志进行编译时，保留了这一约定。
- **f77** 编译器编译的程序并不捕获运算异常，而是在退出时自动调用 **ieee_retrospective** 来报告在执行过程中可能发生的任何异常。如果使用 **-f77** 标志进行编译，可模拟 **f77** 编译器的这种行为。缺省情况下，**f95** 编译器捕获遇到的第一个运算异常，并且不调用 **ieee_retrospective**。
- 在需要更高精度的上下文中，**f77** 编译器处理 **REAL*4** 常量的方式就好像它具有更高的精度一样。 在使用 **-f77** 标志进行编译时，如果将 **REAL*4** 常量与双精度或四精度操作数一起使用，则 **f95** 编译器允许该常量相应地具有双精度或四精度。
- 允许在循环中重新定义 DO 循环变量。[**-f77=misc**]
- 显示所编译的程序单元的名称。[**-f77=misc**]
- 允许在 **DIMENSION** 语句之后声明 **DIMENSION** 语句中使用的变量类型。示例：

```
SUBROUTINE FOO(ARR,G)
DIMENSION ARR(G)
INTEGER G
RETURN
END
```

有关旧版 Sun WorkShop FORTRAN 77 编译器所接受的非标准语言扩展语法和语义的详细信息，请参见 `http://docs.sun.com/source/806-3594/index.html` 上归档的《FORTRAN 77 Language Reference》。

5.2 不兼容问题

下面列出了在使用此发行版的 **f95** 编译和测试传统 **f77** 程序时出现的已知不兼容问题。这些问题是由于 **f95** 中缺少相当的功能或者行为方式不同造成的。这些项目是旧版 Sun WorkShop FORTRAN 77 编译器支持的 FORTRAN 77 非标准扩展，但当前的 **f95** 编译器不支持这些扩展。

源代码格式

- 在指定 **-f77** 选项时，如果名称长度超过 6 个字符，则会发出 ANSI 警告。

I/O：

- **f95** 不允许对直接访问文件使用 **ENDFILE**。
- **f95** 无法识别在直接访问 I/O 中指定记录编号时使用的 **'***n* 格式：**READ (2 '13) X,Y,Z**
- **f95** 无法识别传统 **f77** "**R**" 格式编辑描述符。
- **f95** 不允许在 **CLOSE** 语句中使用 **DISP=** 说明符。
- 不允许在 **WRITE** 语句中使用位常量。
- Fortran 95 **NAMELIST** 不允许使用长度可变的数组和字符串。
- 使用 **RECL=1** 打开的直接访问文件不能用作“流”文件。请改用 **FORMAT='STREAM'**。

- Fortran 95 将非法 I/O 说明符报告为错误，而 **f77** 只发出警告。

数据类型、声明和用法：

- **f95** 只允许 7 个数组下标；而 **f77** 允许 20 个。
- **f95** 不允许在 **PARAMETER** 语句中使用非常量。
- 不能在 **CHARACTER** 类型声明的初始化函数中使用整数值。
- **REAL()** 内部函数返回复数参数的实部，而不是将参数转换为 **REAL*4**。当参数为 **DOUBLE COMPLEX** 或 **COMPLEX*32** 时，这将产生不同的结果。
- Fortran 95 不允许在数组声明之前在边界表达式中使用数组元素。例如：

```
subroutine s(i1,i2)
integer i1(i2(1):10)
dimension i2(10)
...ERROR: "I2" has been used as a function,
therefore it must not be declared with the explicit-shape DIMENSION attribute.


end
```

程序、子例程、函数和语句：

- 名称的最大长度为 127 个字符。

命令行选项：

- **f95** 无法识别 **f77** 编译器选项 **-dbl -oldstruct -i2 -i4**，以及 **-vax** 的一些子选项。

f95 不支持 FORTRAN 77 库例程：

- POSIX 库。
- **IOINIT()** 库例程。
- 磁带 I/O 例程 **topen**、**tclose**、**twrite**、**tread**、**trewin**、**tskipf**、**tstate**。
- **start_iostats** 和 **end_iostats** 库例程。
- **f77_init()** 函数。
- **f95** 不允许通过使用相同名称定义用户自己的例程来绕过 **IEEE_RETROSPECTIVE** 子例程。

5.3 与旧版 FORTRAN 77 编译的例程链接

- 要混合使用 **f77** 和 **f95** 目标二进制文件，应通过 **-xlang=f77** 选项利用 **f95** 编译器进行链接。即使主程序是 **f77** 程序，也使用 **f95** 执行链接步骤。
- 示例：编译包含 **f77** 目标文件的 **f95** 主程序。

```
demo% cat m.f95
CHARACTER*74 :: c = 'This is a test.'
   CALL echo1( c )
END
```

```
demo% f95 -xlang=f77 m.f95 sub77.o
demo% a.out
 This is a test.
demo%
```

- 在 **f95** 程序中可以使用 FORTRAN 77 库和内部函数，《Fortran 库参考》手册中列出了这些库和内部函数。

  示例： **f95** 主程序调用 FORTRAN 77 库中的例程。

```
demo% cat tdtime.f95
        REAL e, dtime, t(2)
        e = dtime( t )
        DO i = 1, 100000
           as = as + cos(sqrt(float(i)))
        END DO
        e = dtime( t )
        PRINT *, 'elapsed:', e, ', user:', t(1), ', sys:', t(2)
        END
demo% f95 tdtime.f95
demo% a.out
elapsed: 0.14 , user: 0.14 , sys: 0.0E+0
demo%
```

  请参见 **dtime**(3F)。

5.3.1 Fortran 内部函数

Fortran 标准支持 FORTRAN 77 中没有的内部函数。《Fortran 库参考》手册中介绍了完整的 Fortran 内部函数集，其中包括非标准的内部函数。

如果在程序中将《Fortran 库参考》中列出的任何内部函数名称用作函数名称，则必须添加 **EXTERNAL** 语句才能让 **f95** 使用您的例程而不是内例程。

《Fortran 库参考》还列出了以前发行版的 **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为 **f77** 编译器识别的那些函数，而忽略 Fortran 内部函数。

5.4 有关迁移到 f95 编译器的附加说明

- **floatingpoint.h** 头文件替代了 **f77_floatingpoint.h**，在源程序中应按如下方式使用它：

  #include "floatingpoint.h"

- 格式为 **f77/** *filename* 的头文件引用应改为删除 **f77/** 目录路径。

- 对于某些使用非标准别名技术（通过为数组设置过度索引，或者重叠 Cray 或 Fortran 指针）的程序，可通过使用相应的 **-xalias** 标志进行编译而受益。请参见第 88 页中的 “3.4.108 **–xalias**[= *keywords*]”。《Fortran 编程指南》中有关移植“旧式”程序的一章中举例说明了这个问题。

5.5 f77 命令

Solaris Studio 软件不再包括单独的 FORTRAN 77 编译器 **f77**。最新的发行版已将许多 FORTRAN 77 功能迁移到 Fortran 95 编译器 **f95** 中。现在，Fortran 95 编译器中提供了旧版 FORTRAN 77 编译器的许多功能。当前 Solaris Studio 编译器发行版提供一个 **f77** 脚本，它用一组适当的缺省选项调用 **f95** 编译器。调用 **f77** 相当于：

```
f95 -f77 -ftrap=%none
```

如果需要链接使用先前发行的 **f77** 编译器编译的库例程，请将 **-xlang=f77** 添加到命令行中。但如果在单独的步骤中进行编译和链接，并显式指定 **-xlang=f77**、**-lM77**、**-lF77** 或 **-lsunmath**，则必须使用 **f95**（或 **f77** 脚本）进行链接，而不能使用 **cc** 或 **CC**。如果还使用 **-fast** 标志进行编译，应将 **-ftrap=%none** 添加到 **-fast** 之后，以保留 FORTRAN 77 对运算异常的捕获行为，因为 **-fast** 将捕获模式设置为 "common"。

```
f77 -fast -ftrap=%none
```

如果调用 **f77** 脚本，将发出消息，警示您正在 **-f77** 兼容模式下使用 **f95** 编译器。您可以在命令行中添加 **-errtags=INVOKE** 来禁用此消息。有关更多信息，请参见第 56 页中的 “3.4.23 **–f77**[= *list*]”。

◆ ◆ ◆ 附 录 A

运行时错误消息

本附录介绍由 Fortran 运行时 I/O 库和操作系统生成的错误消息。

A.1 操作系统错误消息

操作系统错误消息包括系统调用失败、C 库错误和 shell 诊断。系统调用错误消息可在 **intro**(2) 中找到。通过 Fortran 库进行的系统调用并不直接生成错误消息。Fortran 库中的以下系统例程将调用 C 库例程，而这些库例程可生成错误消息：

```
        integer system, status
        status = system("cp  afile bfile")
        print*, "status = ", status
        end
```

显示以下消息：

```
cp: cannot access afile
 status =  512
```

A.2 f95 运行时 I/O 错误消息

f95 I/O 库在运行时检测到错误的时候发出诊断消息。下面是一个使用 **f95** 编译和运行的示例：

```
demo% cat wf.f
      WRITE( 6 ) 1
      END
demo% f95 -o wf wf.f
demo% wf

 ******  FORTRAN RUN-TIME SYSTEM  ******
 Error 1003:  unformatted I/O on formatted unit
 Location:  the WRITE statement at line 1 of "wf.f"
```

```
 Unit:  6
 File:  standard output
Abort
```

因为 **f95** 消息包含对原始源代码文件名和行号的引用，所以应用程序开发者应该考虑在 I/O 语句中使用 **ERR=** 子句以软件方式捕获运行时 I/O 错误。

表 A–1 列出了由 **f95** 发出的运行时 I/O 消息。

表 A–1 f95 运行时 I/O 消息

错误	消息
1000	**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	**顺序访问单元上的直接存取 I/O**
1005	**直接访问单元上的顺序存取 I/O**
1006	**设备不支持 BACKSPACE**
1007	**偏离记录开始**
1008	**无法对文件执行 stat**
1009	**重复计数后没有 ***
1010	**记录过长**
1011	**截断失败**
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	**需要查找能力**
1021	**非法参数**

表 A–1 f95 运行时 I/O 消息 （续）

错误	消息
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	缺省的输入单元已关闭
1052	缺省的输出单元已关闭
1053	来自未连接单元的直接访问 READ
1054	来自未连接单元的直接访问 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式 I/O
1059	未格式化单元上的名称列表 I/O
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

表 A–1 f95 运行时 I/O 消息 （续）

错误	消息
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 组名与程序中的名称不匹配
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

表 A–1 f95 运行时 I/O 消息 （续）

错误	消息
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
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'

表 A–1 f95 运行时 I/O 消息 （续）

错误	消息
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	直接访问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' 且文件为悬空符号链接
1150	状态为 'NEW' 且文件为符号链接
1151	没有可用的临时文件名
1152	缺省单元的说明符 ACCESS='STREAM'
1153	对缺省单元的流访问
1161	设备不支持 REWIND
1162	BACKSPACE 需要读取权限
1163	直接访问单元上的 BACKSPACE
1164	二进制单元上的 BACKSPACE
1165	退格时遇到了文件结束
1166	ENDFILE 需要写入权限
1167	直接访问单元上的 ENDFILE
1168	对顺序或直接访问单元的流访问

表 A–1 f95 运行时 I/O 消息 （续）

错误	消息
1169	对未连接单元的流访问
1170	对流访问单元的直接访问
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	流访问单元的 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	正在对一个无联系的指针解除分配
1183	正在对一个未分配的可分配数组解除分配
1184	正在通过指针对一个可分配数组解除分配
1185	正在对一个未由 ALLOCATE 语句分配的对象解除分配
1186	正在对一个对象的一部分解除分配
1187	正在解除分配的对象比分配的大
1191	未分配的数组被传递给数组内部函数
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	未连接单元上的异步 I/O

表 A–1 f95 运行时 I/O 消息 （续）

错误	消息
1214	同步单元上的异步 I/O
1215	数据编辑描述符与 I/O 列表项类型不兼容
1216	当前 I/O 列表项与任何数据编辑描述符不匹配
1217	非法 CORR_ACCTION 值
1218	由于启用 I/O 处理程序而发生死循环
1220	所请求的字节数大于目标平台上所支持的字节数
1221	UNION 中的数据无法从不兼容的文件类型读取，也不能写入不兼容的文件类型
2001	无效的常量、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)
2022	使用锁定变量时死锁 (OpenMP)
2023	未设置锁定 (OpenMP)

◆ ◆ ◆ 附 录 B

功能发行版历史记录

本附录列出此发行版及早期发行版 Fortran 编译器中新增和更改的功能。

B.1 Oracle Solaris Studio 12.2 Fortran 发行版

Solaris Studio Fortran 95 编译器 8.5 版本是 Oracle Solaris Studio 12.2 发行版中的一个组件。

- 支持 SPARC VIS3 版本的 SPARC-V9 指令集。如果使用 **-xarch=sparcvis3** 选项进行编译，编译器可以使用 SPARC-V9 指令集、UltraSPARC 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 1.0）、UltraSPARC-III 扩展（包括可视指令集 (Visual Instruction Set, VIS) 版本 2.0）、混合乘加指令和可视指令集 (Visual Instruction Set, VIS) 版本 3.0）中的指令。
- 在基于 x86 的系统上，**-xvector** 选项的缺省值已更改为 **-xvector=simd**。在基于 x86 的系统上，缺省情况下以可产生有利结果的优化级别 3 和更高级别使用流扩展。可以使用子选项 **no%simd** 将其禁用。在基于 SPARC 的系统上，缺省值为 **-xvector=%none**。请参见第 131 页中的 “3.4.179 **–xvector**[**=** [[**no%**]**lib**, [**no%**] **simd**, **%none**]]”
- 现在支持 AMD SSE4a 指令集。使用 **-xarch=amdsse4a** 选项进行编译。
- 新增的 **-traceback** 选项使可执行文件在出现严重错误时显示栈跟踪。使用此选项时，可执行文件将捕获一组信号，并显示栈跟踪和核心转储，然后退出。如果多个线程都生成一个信号，则只为第一个线程生成栈跟踪。要使用回溯，请在使用 **f95**、**cc** 或 **CC** 链接程序时添加 **-traceback** 选项。为方便起见，编译时也会接受该选项，不过会将其忽略。将 **-traceback** 选项与 **-G** 选项结合使用来创建共享库的做法是错误的。请参见第 82 页中的 “3.4.95 **–traceback**[**=**{ **%none**|**common**|*signals_list*}]”
- **-mt** 选项已更改为 **-mt=yes** 或 **-mt=no**。**-mt=yes** 选项确保按正确的顺序链接库。请参见第 71 页中的 “3.4.56 **-mt**[**=**{ **yes**|**no**}]”
- **-xprofile=tcov** 选项经过增强，支持可选的配置文件目录路径名，还能够生成与 tcov 兼容的反馈数据。请参见第 121 页中的 “3.4.164 **–xprofile=***p*”。

- 新增的 **-xkeepframe**[=[**%all,%none**] 选项禁止对指定的函数进行与栈相关的优化。**%all** 禁止对所有代码进行与栈相关的优化。**%none** 允许对所有代码进行与栈相关的优化。缺省值为 **-xkeepframe=%none**。请参见第 110 页中的 “3.4.135 **–xkeepframe**[=[**%all,%none**,*name*,**no%** *name*]]”
- 已经实现了其他 F2003 功能。请参见第 148 页中的 “4.6 Fortran 200x 的功能”
- **IVDEP** 指令指示编译器忽略在循环中找到的部分或全部对数组引用的循环附带依赖性，以进行优化。这样，编译器将可以执行通过其他方式不能实现的各种循环优化。**-xivdep** 可用于禁用 **IVDEP** 指令或确定指令的解释方式。请参见第 33 页中的 “2.3.3 **IVDEP** 指令”

B.2 Sun Studio 12 Update 1 Fortran 发行版

- 对于编译器在 x86 平台上的 Solaris OS 或者 Linux OS 上创建的目标文件，如果应用程序代码包含参数或返回值为 _m128/_m64 数据类型的函数，则这些目标文件与先前版本的编译器不兼容。使用 .il 内联函数文件、汇编程序代码或调用这些函数的 asm 内联语句的用户，也需要了解这些不兼容的情况。
- 新增 x86 **–xtarget** 值 **woodcrest**、**penryn**、**nehalem**。
- 新增 SPARC **–xtarget** 值 **ultraT2plus** 和 **sparc64vii**。
- 新增 x86 **–xarch** 和 **–xchip** 值 **ssse3**、**sse4_1**、**sse4_2**、**core2**、**penryn**、**nehalem**、**barcelona**。
- 新增 SPARC **–xarch** 和 **–xchip** 值 **sparcima**、**sparc64vii** 和 **ultraT2plus**。
- **-xprofile=collect** 和 **-xprofile=use** 选项在多线程动态链接应用程序的分析方面提供了改进的支持功能。
- **-xcrossfile=1** 选项成为 **-xipo=1** 选项的别名。
- 在 Solaris 平台上，**-xpec**[= **yes**|**no**] 选项会生成可重新编译用于自动调优系统 (Automatic Tuning System, ATS) 的 PEC 二进制文件。
- 现在，针对 **-x03** 或更高优化级别隐式启用 **-xdepend** 选项，该选项不再包括在 **-fast** 选项的扩展中。
- 支持 OpenMP 3.0 任务处理。
- **-xannotate**[=**yes**| **no**]（仅限 SPARC 平台）指示编译器创建稍后可用 **binopt**(1) 等二进制修改工具转换的二进制文件。
- 四精度 (REAL*16) 已在 x86 平台上实现。REAL*16 是 128 位 IEEE 浮点。
- 编译器通常会在 **/tmp** 目录中创建临时文件。可以通过设置 **TMPDIR** 环境变量来指定其他目录。
- **cpu_time**() Fortran 内例程的行为在 Solaris 和 Linux 平台之间有所不同。
- Fortran 2003 **IMPORT** 语句现已实现。

B.3 Sun Studio 12 Fortran 发行版

- 目前，Fortran 编译器适用于以下 Linux（x86 和 x64）分发包： SuSe Linux Enterprise Server 9 with Service Pack 3（或更新版本）、Red Hat Enterprise Linux 4 以及其他基于 2.6 内核的 Linux 分发包（尽管不会对这些分发包提供正式支持）。
- 使用 **-m64** 创建 64 位可执行文件和共享库。
- 使用 **-xarch** 新标志替换已过时的标志。
- **-xtarget** 和 **-xchip** 的新值为 UltraSPARC T2 和 SPARC64VI 处理器提供代码生成功能。
- **-fma=fused** 新标志允许在支持乘加指令的处理器上生成这些指令。
- **-xhwcprof** 新标志为数据空间分析启用编译器支持。
- **-xinstrument** 新标志使线程分析器支持性能分析。
- **-xregs=frameptr** 已添加至 x86 的 **-fast**。
- 提供 **-xarch=sse2** 和 **-xia** 选项支持在 Solaris x86 平台上执行区间运算。
- x86 平台和 SPARC 平台均接受显式预取指令。(**–xprefetch=explicit**)
- 调试信息的缺省格式由 "stabs" 标准格式更改为 "dwarf" 标准格式。(**-xdebugformat=dwarf**).

B.4 Sun Studio 11 Fortran 发行版

- **新的 -xmodel 选项**：使用新的 -xmodel 选项，可以在 64 位 AMD 体系结构上指定内核、小型或中型内存模型。如果全局变量和静态变量的大小超过 2 GB，请指定 **-xmodel=medium**。否则，请使用缺省设置 **-xmodel=small**。请参见第 115 页中的 “3.4.148 **–xmodel=**[**small** | **kernel** | **medium**]”。
- **为 x86 SSE2 平台扩展的 -xvector 选项**：使用 **-xvector** 选项，可以自动生成向量库函数调用和/或生成 SIMD（Single Instruction Multiple Data，单指令多数据）指令。现在，此选项在 x86 SSE2 平台上提供扩展的语法。请参见第 131 页中的 “3.4.179 **–xvector**[**=** [[**no%**]**lib**, [**no%**] **simd**, **%none**]]”。
- **STACKSIZE 环境变量已增强**：**STACKSIZE** 环境变量的语法已加以改进，可以包括单位关键字。
- **-xpagesize 选项在 x86 平台上可用**：现在，在 x86 平台以及 SPARC 上，启用了 **-xpagesize**、**-xpagesize_heap** 和 **-xpagesize_stack** 选项。请参见第 117 页中的 “3.4.155 **–xpagesize=** *size*”。
- **启用了新的 UltraSPARC T1 和 UltraSPARC IV+ 目标**：**-xarch**、**-xchip**、**-xcache** 和 **-xtarget** 的值支持新的 UltraSPARC 处理器。请参见第 128 页中的 “3.4.175 **–xtarget=***t*”。

B.5 Sun Studio 10 Fortran 发行版：

- **针对 AMD-64 处理器进行编译**

  此发行版引入了 **-xarch=amd64** 和 **-xtarget=opteron**，用于编译要在 64 位 x86 平台上运行的应用程序。

- **在 big-endian 和 little-endian 平台之间进行文件共享**

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

- **在 Solaris 操作系统 x86 平台上提供了 OpenMP**

  对于此发行版的 Solaris Studio，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性。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 库链接目标文件。例如：**% f95 omp_ignore.c -lompstubs**

  不支持同时与 **libompstubs.a** 和 OpenMP 运行时库 **libmtsk.so** 进行链接，因为这样可能会导致意外的行为。

B.6 Sun Studio 9 Fortran 发行版：

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

  此发行版 Solaris Studio 实现了在 Solaris OS x86 平台上使用 Fortran 编译器。使用 **-xtarget** 值（**generic**、**native**、**386**、**486**、**pentium**、**pentium_pro**、**pentium3** 或 **pentium4**）进行编译，可以在 Solaris x86 平台上生成可执行文件。x86 平台上的缺省值为 **-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 内部模块 IEEE_EXCEPTIONS、IEEE_ARITHMETIC 和 IEEE_FEATURES
  - **sun_io_handler** 模块
  - 并行化选项（如 **-autopar** 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** 和 **-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** 外，在 x86 平台上，**-fast** 增加了 **-nofstore**。

  提高了运行时性能：

使用此发行版，大多数应用程序的运行时性能会显著提高。为获得最佳效果，请使用较高的优化级别 **-xO4** 或 **-xO5** 进行编译。在这些优化级别上，编译器现在可以内联所包含的过程，以及那些带假定形状参数、可分配参数或指针参数的过程。

- **Fortran 2003 命令行内部函数：**

  Fortran 2003 标准引入了三个新的内部函数，用来处理命令行参数和环境变量。这三个函数已在此发行版的 **f95** 编译器中实现。新增的内部函数包括：

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

    以 command 返回调用该程序的整个命令行。

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

    以值返回命令行参数。

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

    返回环境变量的值。

新增和更改的命令行选项：

以下 **f95** 命令行选项是此发行版中新增的选项。有关详细信息，请参见第 3 章。

- **-xipo_archive=**{ **none** | **readonly** | **writeback** }

  允许跨文件优化以包括归档 (**.a**) 库。（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*

  设置配置文件数据文件的路径映射。如果分析所用的目录不是以前使用 **-xprofile=collect** 进行编译时所用的目录，请将 **-xprofile_pathmap** 选项和 **-xprofile=use** 选项一起使用。

在此发行版的 **f95** 中，以下命令行选项的缺省值已更改。

- **-xprefetch** 的缺省值为 **-xprefetch=no%auto,explicit**。
- **-xmemalign** 的缺省值为 **-xmemalign=8i**；当使用其中一个 **-xarch=v9** 选项进行编译时，缺省值为 **-xmemalign=8f**。
- 当使用其中一个 **-xarch=v9** 选项进行编译时，-xcode 的缺省值为 **abs44**。

  要使用以前的编译器版本中使用的缺省值进行编译，请明确指定以下选项：

  -xarch=v8 -**xmemalign=4s** -x**prefetch=no**（32 位编译）；**-xcode=abs64 -xprefetch=no**（64 位编译）。

缺省 SPARC 体系结构是 V8PLUS：

缺省 SPARC 架构不再是 V7。在此 Solaris Studio 9 发行版中，对 **-xarch=v7** 的支持有限。新的缺省值为 V8PLUS (UltraSPARC)。使用 **-xarch=v7** 进行编译将被作为使用 **-xarch=v8** 进行编译来处理，因为 Solaris 8 操作系统仅支持 **-xarch=v8** 或更高版本。

要在 SPARC V8 系统（例如，SPARCStation 10）上进行部署，请使用 **-xarch=v8** 显式编译。提供的系统库将在 SPARC V8 架构上运行。

要在 SPARC V7 系统（例如，SPARCStation 1）上进行部署，请使用 **-xarch=v7** 显式编译。提供的系统库将使用 SPARC V8 指令集。对于 Solaris Studio 9 发行版，只有 Solaris 8 OS 支持 SPARC V7 体系结构。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已从 128 增至 256。

- **OpenMP：自动确定变量的作用域：**

  此发行版的 Fortran 编译器中实现了可进行共享内存并行编程的 OpenMP API，因而能够在并行区域中自动确定变量的作用域。有关详细信息，请参见《OpenMP API 用户指南》。（此版本只在 SPARC 平台上实现了 OpenMP。）

B.7 Sun Studio 8 Fortran 发行版：

- **增强的 -openmp 选项：**

  -openmp 选项标志已增强，有助于调试 OpenMP 程序。要使用 **dbx** 调试 OpenMP 应用程序，请使用以下选项进行编译：

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

  随后您可以使用 **dbx** 在并行区域内设置断点，并显示变量的内容。

- **多进程编译：**

  将 **-xjobs=***n* 和 **-xipo** 一起指定，过程间优化器将调用最多 *n* 个代码生成器实例，以编译命令行上列出的文件。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 109 页中的 “3.4.134 **–xjobs=** *n*”。

- **使用 PRAGMA ASSUME进行断言：**

  ASSUME pragma 是此发行版编译器中的新增功能。该 pragma 提示编译器在编译过程中某些点上的条件程序员确信为真。这有助于编译器更好地优化代码。程序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 31 页中的 “2.3.1.8 **ASSUME** 指令”和第 93 页中的 “3.4.111 **–xassume_control**[=*keywords*]”。

- **更多 Fortran 2003 功能：**

  在此版本的 Fortran 编译器中，已经实现了 Fortran 2003 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：

    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。请参见第 148 页中的 “4.6.2 IEEE 浮点异常处理”。

  - 与 C 之间的互操作性：

Fortran 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 148 页中的 “4.6.1 与 C 函数之间的互操作性”。

- **PROTECTED** 属性

  现在，Fortran 编译器接受 Fortran 2003 的 **PROTECTED** 属性。**PROTECTED** 对模块实体的使用进行了限制。具有 **PROTECTED** 属性的对象只能在声明这些对象的模块中定义。第 149 页中的 “4.6.4 **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'**]

  请参见第 149 页中的 “4.6.5 Fortran 2003 异步 I/O”。

与传统 f77 之间的兼容性得到增强：

很多新增功能增强了 Fortran 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 (variable format expression, 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：**

  通过两个新函数，用户可以为逻辑单元上的格式化输入指定自己的错误处理例程。这些例程在第 153 页中的 “4.7.1 I/O 错误处理例程”、相关手册页以及《Fortran 库参考》中进行说明。

- **无符号整数：**

  在此发行版中，Fortran 编译器接受一种新的数据类型 (**UNSIGNED**) 作为对该语言的一种扩展。请参见第 146 页中的 “4.5 无符号整数”。

- **设置首选的栈/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的栈和堆页面大小。请参见第 117 页中的 “3.4.155 **–xpagesize=** *size*”。

- **更快和增强的分析：**

  此发行版引入了新的命令行选项 **-xprofile_ircache=** *path*，可减少分析反馈过程中“使用”阶段的编译时间。请参见第 124 页中的 “3.4.165 **–xprofile_ircache**[**=***path*]”。另请参见第 124 页中的 “3.4.166 **–xprofile_pathmap=** *collect_prefix*:*use_prefix*”。

- **增强的“已知库”：**

  -xknown_lib 选项已增强，可以包括基本线性代数库 (Basic Linear Algebra library, BLAS) 中的更多例程。请参见第 110 页中的 “3.4.136 **–xknown_lib=***library_list*”。

- **链接时优化：**

  使用新的 **-xlinkopt** 标志进行编译和链接，可以在链接时调用后优化器，以便对所生成的二进制目标代码应用一些高级性能优化。请参见第 112 页中的 “3.4.144 **–xlinkopt**[**=**{ **1**|**2**|**0**}]”。

- **局部变量初始化：**

  对 **-xcheck** 选项标志进行了新扩展，从而允许对局部变量进行特殊的初始化。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，则会将该变量初始化为一个有可能导致运算异常的值。请参见第 96 页中的 “3.4.115 **–xcheck=***keyword*”。

B.8 Sun ONE Studio 7 编译器集合 (Forte Developer 7) 版本：

- **Fortran 95 编译器中并入了 Fortran 77 功能**

  此发行版的 Forte Developer 软件用 **f95** 编译器中的新增功能替代了 **f77** 编译器。**f77** 命令是一个调用 **f95** 编译器的脚本：

the command:
`f77` *options files libraries*
becomes a call to the `f95` *compiler::*
`f95 -f77=%all -ftrap=%none` *options files* `-lf77compat` *libraries*

有关 Fortran 77 兼容性和不兼容性的详细信息，请参见第 159 页中的 “4.12 混合语言”。

- **Fortran 77 兼容性模式：**

  新的 **-f77** 标志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 结构和约定。请参见第 56 页中的 “3.4.23 **–f77**[**=** *list*]”和第 159 页中的 “4.12 混合语言”。

- **对使用非标准别名的“旧式”程序进行编译：**

  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、指针和过度索引来设置变量别名的标准。很多“旧式”传统程序特意使用别名技术解决早期版本的 Fortran 语言中存在的缺点。请使用新的 **-xalias** 标志，让编译器知道程序与标准偏离的程度，以及预计会出现哪种别名问题。在某些情况下，只有在指定了正确的 **-xalias** 子选项时，编译器才会生成正确的代码。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 88 页中的 “3.4.108 **–xalias**[**=** *keywords*]”以及《Fortran 编程指南》中有关移植的章节。

- **增强的 MODULE 功能：**
  - 新标志 **-use=***list* 可强制将一个或多个隐式 **USE** 语句添加到每个子程序中。请参见第 84 页中的 “3.4.100 **–use=** *list*”。
  - 新标志 **-moddir=***path* 可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 70 页中的 “3.4.55 **–moddir=** *path*”。新环境变量 **MODDIR** 也可以控制写入 .mod 文件的位置。
  - -M*path* 标志现在可以接受在目录路径、归档 (**.a**) 文件或模块 (**.mod**) 文件中搜索 **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 69 页中的 “3.4.53 **–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。

  有关详细信息，请参见第 156 页中的 “4.9 模块文件”。

使用 -Xlist 执行增强的全局程序分析 :

此发行版的 **f95** 编译器在由 **-Xlist** 标志提供的全局程序分析中添加了许多新检查。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。有关详细信息，请参见第 86 页中的 “3.4.106 **–Xlist**[*x*]”、Forte Developer 的《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，并忽略用户提供的任何版本。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。在此发行版中，已知库的名称仅限于 **blas** 和 **intrinsics**，前者代表 Sun 性能库中 BLAS 例程的一个子集，后者则用于忽略 Fortran 95 标准内部函数以及由用户提供的任何上述例程版本的显式 **EXTERNAL** 声明。请参见第 110 页中的 “3.4.136 **–xknown_lib=***library_list*”。

- **忽略接口中的哑元参数类型 :**

  新增指令 **!$PRAGMA IGNORE_TKR** { *list_of_variables*} 使编译器在解析特定调用时，忽略在通用过程接口中出现的指定哑元参数名称的类型、种类和等级。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 28 页中的 “2.3.1.2 **IGNORE_TKR** 指令”。

- **增强的 -C 运行时数组检查 :**

  在此发行版的 **f95** 编译器中，使用 **-C** 选项进行的运行时数组下标范围检查得到了增强，可以包括数组一致性检查。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 50 页中的 “3.4.6 **–C**”。

- **引入 Fortran 2003 功能 :**

  在此发行版的 **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。这些功能包括 **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。有关详细信息，请参见第 150 页中的 “4.6.9 Fortran 2003 格式化 I/O 功能”。

- **格式化 I/O 中的舍入 :**

  新增的选项标志 **-iorounding** 可以为格式化 I/O 设置缺省舍入模式。该模式（processor-defined 或 compatible），对应于作为 Fortran 2003 功能的一部分而实现的 **ROUND=** 说明符。请参见第 67 页中的 “3.4.45 **–iorounding**[**=**{**compatible**|**processor-defined**}]”。

- **删除的过时标志 :**

  下列标志已从 **f95** 命令行中删除 :

  -db -dbl

f95 编译器中没有实现下列 **f77** 编译器标志，因此这些标志也被视为过时：

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

- **栈溢出检查：**

  在使用新的 **-xcheck=stkovf** 标志进行编译时，将在子程序入口处添加对栈溢出条件的运行时检查。如果检测到栈溢出，就会出现 **SIGSEGV** 段故障。对于在栈中分配了较大数组的多线程应用程序，其中的栈溢出可能导致相邻线程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 96 页中的 “3.4.115 **–xcheck=***keyword*”。

- **新的缺省线程栈大小：**

  在此版本中，SPARC V8 平台上的缺省从属线程栈大小已增加到 4 MB；而 SPARC V9 平台则增加到 8 MB。有关详细信息，请参见《Fortran 编程指南》的“并行化”一章中有关栈和栈大小的讨论。

- **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。此发行版增加了 **-xipo=2**，可实现增强的过程间别名分析、内存分配和布局优化，从而提高高速缓存性能。请参见第 106 页中的 “3.4.131 **–xipo**[={ **0**|**1**|**2**}]”。

- **使用 -xprefetch_level=n 控制预取指令：**

  使用新标志 **-xprefetch_level=***n* 可控制通过 **-xprefetch=auto** 进行的预取指令自动插入。使用时，要求 **-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 121 页中的 “3.4.163 **–xprefetch_level=**{ **1**|**2**|**3**}”。

您可以在 `http://docs.sun.com` Web 站点上的早期版本文档集中找到 Forte Developer 7 之前各版本的功能历史记录。

◆ ◆ ◆ 附 录 C

Fortran 指令摘要

本附录汇总了 **f95** Fortran 编译器可识别的指令：

- 通用 Fortran 指令
- Sun 并行化指令
- Cray 并行化指令
- OpenMP Fortran 95 指令、库例程和环境

C.1 通用 Fortran 指令

第 26 页中的 “2.3 指令”部分对 **f95** 接受的通用指令进行了说明。

表 C–1 通用 Fortran 指令摘要

格式	
!$PRAGMA *keyword* **(** a **[,]** a **] …) [** , *keyword* (a [, a **] …)] ,…** **!$PRAGMA SUN** *keyword* **(** a **[,]** a **] …) [** , *keyword* (a [, a **] …)] ,…** 第一列中的注释指示符可以是 **c**、**C**、**!** 或 *****。（在这些示例中使用的是 **!**。**f95** 自由格式必须使用 **!**。）	
C 指令	**!$PRAGMA C(***list* **)** 将一系列外部函数的名称声明为 C 语言例程。
IGNORE_TKR 指令	**!$PRAGMA IGNORE_TKR** {*name* {, *name*} ...} 在解析特定调用时，编译器会忽略在通用过程接口中出现的指定哑元参数名称的类型、种类和等级。
UNROLL 指令	**!$PRAGMA SUN UNROLL=***n* 建议编译器将下面的循环解开为指定的长度 *n*。

表 C–1 通用 Fortran 指令摘要 （续）

WEAK 指令	**!$PRAGMA WEAK(***name*[**=***name2*]**)** 将 *name* 声明为弱符号，或者声明为 *name2* 的别名。
OPT 指令	**!$PRAGMA SUN OPT=***n* 将子程序的优化级别设置为 *n*。
PIPELOOP 指令	**!$PRAGMA SUN PIPELOOP**[**=** *n*] 断言循环中在间隔为 *n* 的迭代之间存在依赖性。
PREFETCH 指令	**!$PRAGMA SUN_PREFETCH_READ_ONCE (***name* **)** **!$PRAGMA SUN_PREFETCH_READ_MANY (***name* **)** **!$PRAGMA SUN_PREFETCH_WRITE_ONCE (***name* **)** **!$PRAGMA SUN_PREFETCH_WRITE_MANY (***name* **)** 请求编译器为名称引用生成预取指令。（需要使用 **-xprefetch** 选项，缺省情况下启用该选项。编译时使用 **–xprefetch=no** 可以禁用预取指令。目标体系结构也必须支持预取指令，而且编译器优化级别必须设置为大于 **–xO2**。）
ASSUME 指令	**!$PRAGMA** [**BEGIN**} **ASSUME (***expression* [**,***probability*]**)** **!$PRAGMA END ASSUME** 断言编译器可假定程序中某些点处的条件为真。

C.2 特殊的 Fortran 指令

下列指令仅用于 **f95**。有关详细信息，请参见第 155 页中的 “4.8.2 **FIXED** 和 **FREE** 指令”。

表 C–2 特殊的 Fortran 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$** *directive*... 该行必须在第 1 列开始。对于自由格式的源文件，该行之前可以有空格。
FIXED/FREE 指令	**!DIR$ FREE!DIR$ FIXED** 这些指令指定指令后面的行的源代码格式。它们适用于所在源代码文件的其余部分，或者在遇到下一个 **FREE** 或 **FIXED** 指令之前的部分。

表 C–2 特殊的 Fortran 指令 （续）

IVDEP	**!DIR$ IVDEP** 断言以下 **DO**、**FORALL** 或 **WHERE** 循环不含循环附带的依赖性，并且可以进行优化。**-xivdep** 选项确定的解释。请参见第 33 页中的 “2.3.3 **IVDEP** 指令”

C.3 Fortran OpenMP 指令

Solaris Studio Fortran 编译器支持 OpenMP 3.0 Fortran API。**-openmp** 编译器标志允许使用这些指令。（请参见第 116 页中的 “3.4.153 **–xopenmp**[={ **parallel**|**noopt**|**none**}]”）。

有关完整的详细信息，请参见《OpenMP API 用户指南》。

索引

A

abrupt_underflow，59
ALLOCATABLE，扩展，149
asa，Fortran 打印实用程序，18
ASSUME 指令，31

C

C(..) 指令，28
CALL，用 **-inline** 内联子程序调用，66
command-line options，**-xprefetch**，31
COMMON
  TASKCOMMON 一致性检查，100
  对齐，47
  全局一致性，**-Xlist**，86
  填充，75
cpp，C 预处理程序，24, 52, 55
Cray
  指针，142
  指针和 Fortran 指针，143

D

dbx，使用 **-g** 选项编译，64

F

f95 命令行，22, 39
fdumpmod 用于查看模块内容，26, 158
FFLAGS 环境变量，35
FIXED 指令，155–156, 156
FLUSH 语句，151
Fortran
  处理非标准的 Fortran 77 别名设置，168
  功能和扩展，18
  模块，156–158
  实用程序，18
  与传统版本兼容，48
  与传统程序的不兼容性，165–166
  与传统程序的兼容性，56–57, 161–168
  预处理程序，52
    使用 **-F** 调用，55
  指令，155–156
Fortran 200x，148
Fortran 95
  Forte Developer 7 发行版，184–186
  I/O 扩展，153–154
  大小写，137
  功能，135
  与 Fortran 77 链接，166–167
fpp，Fortran 预处理程序，24, 52, 55, 61
FREE 指令，155–156, 156
fsecond-underscore，55
fsplit，Fortran 实用程序，18

G

gprof，**-pg**，分步进行分析，76

I

I/O 扩展，153–154
#ifdef，24
IGNORE_TKR 指令，28
IMPORT 语句，151
#include 路径，65
#include，24
INCLUDE 文件，65
  floatingpoint.h，167
  system.inc，34
ISA, 指令集体系结构，90
IVDEP 指令，33, 109

L

libm, 缺省情况下搜索，68
limit
  命令，37
  栈大小，81

M

memory, actual real memory, display，36
.mod 文件, 模块文件，156
MODDIR 环境变量，71

N

nonstandard_arithmetic()，59

O

OpenMP，32–33
  指令摘要，189
OPT 指令，30
  -xmaxopt 选项，113

P

Pentium，130
PIPELOOP 指令，30–31
POSIX 库, 不支持的，166
POSIX 线程，71
pragma, **请参见**指令
PREFETCH directive，31
PREFETCH 指令，31

S

shell, 限制，37
SIGFPE, 浮点异常，60
Solaris 线程，71
SPARC 平台
  代码地址空间，98
  高速缓存，95
  芯片，97
  指令集体系结构，91
STACKSIZE 环境变量，81
STOP 语句, 返回状态，81
strict（区间运算），106
swap 命令，36
system.inc，34–35

T

tcov, **-xprofile**，123
traceback，82–83

U

ulimit 命令，37
UNROLL 指令，29

V

VAX VMS Fortran 扩展，85, 146

W

WEAK 指令，30

widestneed（区间运算），106

X
x86 上的精度
  -fprecision，61
  -fstore，63

安
安装, 路径，66

八
八进制，138

版
版本, 每次编译过程的 ID，84

帮
帮助
  命令行，20
  自述文件信息，104

绑
绑定, 动态/共享库，53

保
保留大小写，83

本
本地变量初始化，96

编
编译过程，85
编译和链接，23, 25
  动态（共享）库，53
  和 **-B**，50
  仅编译，50
  生成动态共享库，64
编译器
  计时，82
  命令行，22
  驱动程序, 使用 **-dryrun** 显示命令，53
  显示版本，84
  详细消息，85

变
变量
  对齐，141
  局部，80
  未声明的，84

标
标志, **请参见**命令行选项
标准
  include 文件，66
  标识非 ANSI 扩展, **-ansi** 标志，48
  一致性，17
标准数值序列类型，48

别
别名，88
  -xalias，88

并
并行化
  OpenMP，32–33, 116–117
  汇总的 OpenMP 指令，189
  消息，85

并行化（**续**）
循环信息，69
约简操作，79
指令，156
指令（**f77**），32
自动，49

捕
捕获，浮点异常，63

不
不兼容性，FORTRAN 77，165–166

布
布尔
常量，替代格式，138
类型，常量，137

参
参数，全局一致性，**Xlist**，86
参数，一致，**Xlist**，86

常
常量参数，**-copyargs**，50

成
成为，制表符，135

处
处理器，指定目标处理器，97
处理顺序，选项，40

磁
磁带 I/O，不支持的，166

错
错误消息
f95，169–176
使用 **-erroff** 禁止，54
消息标记，54

打
打印，**asa**，18

大
大文件，36
大小写，保留大小写，83

代
代码大小，127

单
单行程 **DO** 循环，75

地
地址空间，87

递
递归子程序，125

调
调试
- **-g** 选项，64
- **-Xlist**，19
- 不包括目标文件，127
- 交叉引用表，86
- 实用程序，19
- 使用 **-C** 检查数组下标，50
- 使用 **-dryrun** 显示编译器命令，53
- 用 **-Xlist** 进行全局程序检查，86
- 优化，64

动
动态库
- 命名动态库，65
- 生成，**-G**，64

堆
堆页面大小，117，118

对
对函数重组，101
对齐
- **另请参见**数据
- **-dalign**，52
- 使用 **-aligncommon** 将 COMMON 中的数据对齐，47

多
多线程，71
多线程处理，**请参见**并行化

二
二进制 I/O，153–154

发
发行历史记录，177–186

分
分析，**-pg**，**-gprof**，76
分析器编译选项，**xF**，101

浮
浮点
- 捕获模式，63
- 非标准，60
- 区间运算，106
- 舍入，62
- 首选项，**-fsimple**，62

符
符号表，**dbx**，64

覆
覆盖率分析 (**tcov**)，123

高
高速缓存
- 填充，75
- 指定硬件高速缓存，95

功
功能
- Fortran 95，135
- 发行历史记录，177–186

功能和扩展，18

共
共享库
- 纯, 无重定位，132
- 禁止链接, **-dn**，53
- 命名共享库，65
- 生成, **-G**，64

固
固定格式的源，59

过
过度索引, 别名，88
过时的命令行选项，47

函
函数, 外部 C，28
函数级重组，101

宏
宏命令行选项，46

后
后缀
- 编译器识别的文件名，23
- 可由编译器识别的文件名，136

环
环境, 由 **STOP** 终止程序，81
环境变量，35–36

汇
汇编代码，79

霍
霍尔瑞斯，139

兼
兼容性
- Fortran 77，56–57, 161–168
- 使用 C，159
- 向前，159

将
将符号表与可执行文件分离, **s**，79

交
交叉引用表, **Xlist**，86
交换空间
- 显示实际的交换空间，36
- 限制磁盘的交换空间量，36

接
接口, 库，34

禁
禁止
- 警告，86
- 链接，50
- 隐式确定类型，84
- 由标记名称列出的警告, **-erroff**，54

警

警告
- 禁止消息，86
- 使用 **-erroff** 禁止，54
- 使用非标准扩展，48
- 未声明的变量，84
- 消息标记，54

静

静态, 绑定，53

可

可执行文件
- 动态库的内置路径，78
- 将符号表分离，79
- 名称，75

库

库
- Sun 性能库，19, 112
- 接口，34
- 禁用系统库，72
- 可执行文件中的动态搜索路径，78
- 可执行文件中共享库的路径，73
- 命名共享库，65
- 生成, **-G**，64
- 用 **-l** 链接，68
- 与位置无关和纯，132

扩

扩展
- **ALLOCATABLE**，149
- **VALUE**，150
- VAX 结构与联合，146
- 非 ANSI, **-ansi** 标志，48
- 格式化 I/O，150
- 流 I/O，150

扩展（**续**）
- 其他 I/O，153–154

扩展和功能，18

类

类型声明的替代格式，140

链

链接
- 禁用系统库，72
- 链接程序的 **-Mmapfile** 选项，101
- 启用动态链接, 共享库，53
- 弱名称，30
- 使用自动并行化, **-autopar**，49
- 通过编译，23
- 一致编译和链接，25
- 用 **-l** 指定库，68
- 与编译分开进行，25
- 与编译一致，25

链接时优化，112

临

临时文件, 目录，82

流

流 I/O，150

路

路径
- 到标准 include 文件，66
- 可执行文件中的动态库，78
- 库搜索，68

名

名称
参数, 不附加下划线，28
对象, 可执行文件，75

命

命令行, 帮助，20
命令行选项
所有选项标志的参考，47–133
按功能分组，40
摘要，40–47
-a（已过时），47
-aligncommon，47
-ansi，48
-arg=local，48
-autopar, 自动并行化，48–49
-Bdynamic，49
-Bstatic，49
-C, 检查下标，50
-c, 仅编译，50
-cg89, -cg92（已过时），47
-copyargs, 允许存储到文本参数，50–51
-D*name*, 定义符号，51
-dalign，52, 58
-dbl_align_all, 强制对齐数据，52
-depend，58
数据依赖性分析，53
-dryrun，53
-dy，53
-e, 扩展的源代码行，54
-erroff, 禁止警告，54
-errtags, 一起显示消息标记和警告，54
-errwarn, 错误警告，54
-ext_names, 没有下划线的外部名称，55
-F，55
-f, 与 8 字节边界对齐，55–56
-f77，56–57
-fast，57–59
-fixed，59
-flags，59
-fma，59
-fnonstd，59–60
-fns，58, 60–61
命令行选项（续）
-fpp, Fortran 预处理程序，61
-fprecision, x86 精度模式，61
-free，62
-fround=*r*，62
-fsimple，58
简单浮点模型，62–63
-fstore，63
-ftrap，63–64
-G，64
-g，64–65
-h*name*，65
-help，65
-I*dir*，65–66
-i8－使用 **-xtypemap=integer:64** 代替，66
-inline，66–67
-iorounding，67
-keeptmp，67
-KPIC，67
-Kpic，67
-L*dir*，68
-l库，68
-libmil，58, 69
-loopinfo, 显示并行化，69
-M*dir*, f95 模块，157
-m32 | **-m64**，70
-moddir，70
-mt, 多线程安全库，71
-native，71
-native（已过时），47
-noautopar，71
-nodepend，72
-nofstore，72
-nolib，72
-nolibmil，72
-noqueue（已过时），47
-noreduction，72
-norunpath，73
-O*n*，58, 73
-o, 输出文件，75
-onetrip，75
-openmp，75
-p, 配置文件（已过时），75
-pad=*p*，58, 75–76

命令行选项（续）
-pg, 按过程配置，76–77
-PIC，77
-pic，77
-PIC（已过时），47
-pic（已过时），47
-Qoption，78
-R 列表，78
-r8const，78–79
-recl=a[**,b**]，79
-S，79
-s，79–80
-sb, -sbfast（已过时），47
-sb, 已过时，80
-sbfast，80
-silent，80
-stackvar，80–81, 125
-stop_status，81
-temp，82
-time，82
-traceback，82–83
-U, 不转换为小写，83
-U*name*, 取消预处理程序宏的定义，83
-u，84
-unroll, 解开循环，84
-use，158
-V，84
-v，84–85
-vax，85
-vpara，85–86
-w，86
-xaddr32，87
-xalias=*list*，88–89
-xannotate[**=**{**yes**|**no**}]，89–90
-xarch=*isa*，90–93
-xassume_control，32, 93
-xautopar，94
-xbinopt，94–95
-xcache=*c*，95–96
-xchip=*c*，97–98
-xcode=*c*，98
-xcommoncheck，100
-xcrossfile（已过时），100
-xdebugformat，100

命令行选项（续）
-xdepend，101
-xF，101
-xhasc, 霍尔瑞斯常量作为字符，103–104
-xhelp=*h*，104
-xhwcprof，104
-xia, 区间运算，105
-xinline，105
-xinstrument，105
-xinterval=*v* 用于区间运算，106
-xipo, 过程间优化，106–108
-xipo_archive，108
-xivdep，109
-xjobs, 多处理器编译，109
-xkeepframe, 禁止与栈相关的优化，110
-xknown_lib, 优化库调用，110–111
-xl,（已过时），111
-xlang=f77, 与 Fortran 77 库链接，111
-xld,（已过时），111
-xlibmil，111
-xlibmopt，58, 111
-xlic_lib=sunperf，112
-xlicinfo（已过时），112
-xlinkopt，112–113
-xlinkopt, 链接时优化，112–113
-Xlist, 全局程序检查，86–87
-xloopinfo，113
-xmaxopt，113
-xmemalign，113–115
-xnolib，115
-xnolibmopt，115
-xO*n*，116
-xopenmp，116–117
-xpagesize，117–118
-xpagesize_heap，118
-xpagesize_stack，118
-xpec，118
-xpg，118
-xpp=*p*，119
-xprefetch，31
-xprefetch_auto_type，121
-xprofile_ircache，124
-xprofile=*p*，121–124
-xprofile_pathmap=*param*，124

命令行选项（**续**）
-xrecursive，125
-xreduction，125
-xregs=*r*，125–127
-xs，127
-xsafe=mem，127
-xsb，127
-xsbfast，127
-xspace，127
-xtarget=native，58
-xtarget=*t*，128–130
-xtime，130
-xtypemap，130–131
-xunroll，131
-xvector，58, 131–132
-ztext，132–133
不支持的已过时 **f77** 标志，166
常用，45–46
处理顺序，40
宏，46
将选项传递到编译阶段，78
无法识别的选项，25–26
已过时，47
语法，39–40
传统，46
命令行选项列表，65
命令选项, **-dn**，53

模
模板, 内联，69
模块，156–158
fdumpmod 用于查看模块内容，26
fdumpmod 用于显示模块文件，158
.mod 文件，156
-use，158
创建和使用，26
缺省路径，71

目
目标库搜索目录，68
目标文件
仅编译，50
名称，75
目录, 临时文件，82

内
内部函数
扩展，158
传统的 Fortran，167
内存
限制虚拟内存，37
优化器内存不足，36
内例程, 接口，34
内联
模板, **-libmil**，69
使用 **-fast**，58
使用 **-inline**，66
用 **-O4** 自动，74

配
配置文件数据路径映射，124

区
区间运算
-xia 选项，105
-xinterval 选项，106

全
全局程序检查, **-Xlist**，86
全局符号, 弱，30

缺
缺省
include 文件路径，66
数据大小和对齐，141

弱
弱链接程序符号，30

舍
舍入，62, 63

十
十六进制，138

实
实用程序，18

手
手册页，19

数
数据
  COMMON, 使用 **aligncommon** 对齐，47
  大小和对齐，141
  将常量提升为 **REAL*8**，78
  使用 **-dbl_align_all** 对齐，52
  用 **-f** 对齐，55
  用 **-xtypemap** 映射，131
  与 **-xmemalign** 对齐，113–115
数据类型的对齐，141
数据依赖性, **-depend**，53
数学库
  和 **-L** *dir* 选项，68
  优化版本，111
数值序列类型，48
数组边界检查，50

顺
顺序, 函数，101

搜
搜索, 目标库目录，68

填
填充，75

外
外部 C 函数，28
外部名称，55

为
为 **cpp** 定义符号, **D**name，51

未
未对齐数据, 指定行为，113

文
文档, 访问，13–14
文档索引，13
文件
  可执行文件，23
  目标，23
  文件太大，36
文件名
  编译器识别的，23
  可由编译器识别，136
文件配置, **-xprofile**，121

无
无效, 浮点，63

下
下标范围，50
下划线，55
　不附加到外部名称，28
下溢
　捕获浮点，63
　渐进，61

显
显式, 确定类型，84
显式并行化指令，32

线
线性代数例程，112

限
限制, Fortran 编译器，137

陷
陷阱, 内存，127

向
向后兼容性, 命令行选项，46

消
消息
　并行化，69, 85
　详细，85
消息（**续**）
　用 **-silent** 禁止，80
　运行时，169

性
性能
　Sun 性能库，19
　优化，57
性能库，112

续
续行，54, 135

选
选项, **请参见**命令行选项

循
循环
　并行化消息，69
　依赖性分析, **-depend**，53
　用 **-unroll** 解开，84
　用指令解开，29
　执行一次, **-onetrip**，75
　自动并行化，49

页
页面大小, 设置栈或堆，117, 118

已
已编译代码的大小，127

异

异常, 浮点，63
  捕获，63

易

易读文档，14

溢

溢出
  捕获浮点，63
  栈，80

硬

硬件体系结构，90, 97

用

用法, 编译器，22

优

优化
  OPT 指令，30, 113
  PIPELOOP 指令，30–31
  PREFETCH 指令，31
  别名，88
  调试，64
  浮点，62
  过程间，106
  级别，73
  跨源文件，100, 106
  链接时，112
  目标硬件，71
  内联用户编写的例程，66
  使用 **-fast**，57
  数学库，111
  循环解开，84

优化（续）
  用 **-xvector** 进行向量库变换，131–132
  用指令解开循环，29
  指定处理器，97
  指定高速缓存，95
  指定指令集体系结构，90

与

与位置无关的代码，77, 98

语

语法
  f95 命令，22, 39
  编译器命令行，39
  命令行选项，39–40

预

预处理程序, 源文件
  fpp, **cpp**，24
  定义符号，51
  强制 **-fpp**，61
  取消定义符号，83
  使用 **-xpp=***p* 指定，119

源

源代码格式
  选项 (f95)，136
  源代码行的混合格式 (f95)，137
源代码行
  保留大小写，83
  固定格式，59
  扩展的，54
  行长度，135
  预处理程序，119
  自由格式，62
源文件, 预处理，24

约
约定, 文件名后缀，23

运
运算, **请参见**浮点

栈
栈
  设置页面大小，117, 118
  溢出，80
  增加栈大小，81
栈溢出，96

指
指令
  ASSUME，31–32
  FIXED，155–156
  Fortran 77，26
  FREE，155–156
  IGNORE_TKR，28
  OpenMP (Fortran)，32–33, 189
  并行化，32–33, 156
  弱链接，30
  所有指令的摘要，187
  特殊 Fortran，155
  循环解开，29
  优化级别，30
指令列表，187
指令中的 **!DIR$**，155
指令中的 **CDIR$**，155
指针，142
  别名，88
指针对象，142

制
制表符, 成为源代码制表符，135

注
注释, 作为指令，155

传
传统编译器选项，46

自
自动读取 (**dbx**)，127
自述文件，104
自由格式源，62