RealView® 编译工具

4.0 版

链接器参考指南

ARM®

Copyright © 2008 ARM Limited. All rights reserved.
ARM DUI 0381AC

RealView 编译工具
链接器参考指南

Copyright © 2008 ARM Limited. All rights reserved.

版本信息

本手册进行了以下更改。

更改历史记录

日期	发行号	保密性	变更
2008 年 9 月	A	非保密	ARM® RealView® Development Suite 4.0 版

所有权声明

除非本所有权声明在下面另有说明，否则带有 ® 或 ™ 标记的词语和徽标是 ARM Limited 在欧盟和其他国家/地区的注册商标或商标。此处提及的其他品牌和名称可能是其各自所有者的商标。

除非事先得到版权所有人的书面许可，否则不得以任何形式修改或复制本文档包含的部分或全部信息以及产品说明。

本文档描述的产品还将不断发展和完善。ARM Limited 将如实提供本文档所述产品的所有特性及其使用方法。但是，所有暗示或明示的担保，包括但不限于对特定用途适销性或适用性的担保，均不包括在内。

本文档的目的仅在于帮助读者使用产品。对于因使用本文档中的任何信息、文档信息出现任何错误或遗漏或者错误使用产品造成的任何损失或损害， ARM 公司概不负责。

使用 ARM 一词时，它表示 ARM 或其任何相应的子公司。

保密状态

本文档的内容是非保密的。根据 ARM 与 ARM 将本文档交予的参与方的协议条款，使用、复制和公开本文档内容的权利可能会受到许可限制的制约。

受限访问是一种 ARM 内部分类。

产品状态

本文档的信息是开发的产品的最新信息。

网址

`http://www.arm.com`

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

目录
RealView 编译工具
链接器参考指南

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

前言

本前言介绍 《RealView 编译工具链接器参考指南》。本前言分为以下几节：

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

关于本手册

本手册提供了有关随 ARM RealView® 编译工具提供的 ARM® 链接器 armlink 的参考信息。

本手册详细说明了命令行选项、控制文件命令、分散加载描述文件、*基础平台应用程序二进制接口* (BPABI) 和 System V release 4 (SysV) 共享库和可执行文件。有关使用和控制这些工具的一般信息，请参阅 《链接器用户指南》。

适用对象

本手册是为所有使用 RealView 编译工具生成应用程序的开发人员编写的。本手册假定您是一位有经验的软件开发人员，并且熟悉 《RealView 编译工具要点指南》中所介绍的 ARM 开发工具。

使用本手册

本手册由以下章节组成：

第 1 章 *简介*

本章简要介绍链接器。

第 2 章 *链接器命令行选项*

本章列出了链接器所支持的所有命令行选项和控制文件命令。

第 3 章 *分散加载描述文件的形式语法*

本章提供了有关分散加载描述文件的格式的信息。

第 4 章 *BPABI 和 SysV 共享库和可执行文件*

本章提供了有关 BPABI 和 SysV 共享库和可执行文件的信息。

本手册假定 ARM 软件安装在缺省位置。例如，在 Windows 上，这可能是 *volume*:\Program Files\ARM。引用路径名时，假定安装位置为 *install_directory*，如 *install_directory*\Documentation\...。如果将 ARM 软件安装在其他位置，则可能需要更改此位置。

印刷约定

本手册使用以下印刷约定：

monospace　　表示可以从键盘输入的文本，如命令、文件和程序名以及源代码。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

monospace　表示允许的命令或选项缩写。可只输入下划线标记的文本，无需输入命令或选项的全名。

monospace italic

表示此处的命令和函数的变量可用特定值代替。

等宽粗体　表示在示例代码以外使用的语言关键字。

斜体　突出显示重要注释、介绍特殊术语以及表示内部交叉引用和引文。

粗体　突出显示界面元素，如菜单名称。有时候也用在描述性列表中以示强调，以及表示 ARM 处理器信号名称。

更多参考出版物

本部分列出了 ARM 公司和第三方发布的、可提供有关 ARM 系列处理器开发代码的附加信息的出版物。

ARM 公司将定期对其文档进行更新和更正。有关最新勘误表、附录和 ARM *常见问题* (FAQ)，请访问 http://infocenter.arm.com/help/index.jsp。

ARM 公司出版物

本手册包含的参考信息特定于随 RealView 编译工具提供的 ARM 链接器。该套件中包含的其他出版物有：

- 《RealView 编译工具要点指南》 (ARM DUI 0202)
- 《RealView 编译工具编译器用户指南》 (ARM DUI 0205)
- 《RealView 编译工具编译器参考指南》 (ARM DUI 0348)
- 《RealView 编译工具链接器用户指南》 (ARM DUI 0206)
- 《RealView 编译工具实用程序指南》 (ARM DUI 0382)
- 《RealView 编译工具库和浮点支持指南》 (ARM DUI 0349)
- 《RealView 编译工具汇编器指南》 (ARM DUI 0204)
- 《RealView 编译工具开发指南》 (ARM DUI 0203)

有关基本标准、软件接口和 ARM 支持的标准的完整信息，请参阅 *install_directory*\Documentation\Specifications\...。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

此外，有关与 ARM 产品相关的特定信息，请参阅下列文档：

- 《ARM 体系结构参考手册》ARMv7-A™ 和 ARMv7-R™ 版 (ARM DDI 0406)
- 《ARM7-M™ 体系结构参考手册》 (ARM DDI 0403)
- 《ARM6-M™ 体系结构参考手册》 (ARM DDI 0419)
- 《ARM 体系结构参考手册》 (ARM DDI 0100)
- 您的硬件设备的 ARM 数据手册或技术参考手册

其他出版物

有关 Linux 标准基本规范的详细信息，请访问 `http://www.linux-foundation.org`。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

反馈

ARM Limited 欢迎提供有关 RealView 编译工具与文档的反馈。

对 RealView 编译工具的反馈

如果您对 RealView 编译工具有任何问题，请与供应商联系。为便于供应商快速提供有用的答复，请提供：

- 您的姓名和公司
- 产品序列号
- 您所用版本的详细信息
- 您运行的平台的详细信息，如硬件平台、操作系统类型和版本
- 能重现问题的一小段独立的程序
- 您预期发生和实际发生的情况的详细说明
- 您使用的命令，包括所有命令行选项
- 能说明问题的示例输出
- 工具的版本字符串，包括版本号和内部版本号

关于本手册的反馈

如果您发现本手册有任何错误或遗漏之处，请发送电子邮件到 `errata@arm.com`，并提供：

- 文档标题
- 文档编号
- 您有疑问的页码
- 问题的简要说明

我们还欢迎您对需要增加和改进之处提出建议。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

第 1 章
简介

本章介绍随 ARM RealView® 编译工具提供的 ARM® 链接器 armlink。本章分为以下几节：

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

1.1 ARM 链接器

ARM 链接器 armlink 将一个或多个对象文件的内容与一个或多个对象库的选定部分相结合，从而生成：

- ARM *可执行和链接格式* (ELF) 映像
- 一个部分链接的 ELF 对象，可用作后续链接步骤的输入
- 一个符合 《ARM 体系结构的基础平台应用程序二进制接口》 (BPABI) 或 *System V release 4* (SysV) 规范的共享对象，或是一个 BPABI 或 SysV 可执行文件。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

1.2 与旧对象和库的兼容性

如果从早期版本升级到 RealView 编译工具，请务必阅读 《RealView 编译工具要点指南》以了解最新信息。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

第 2 章
链接器命令行选项

本章列出了 ARM 链接器 armlink 所支持的命令行选项和控制文件命令。它包括下面一节：

- 第2-2 页的*命令行选项*
- 第2-63 页的*控制文件命令*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.1 命令行选项

本节按字母顺序列出了链接器所支持的命令行选项。

注意

在某些 fromelf 参数中，可以使用特殊字符选择多个符号名称：

- 通配符 * 可用于匹配任意名称。
- 通配符 ? 可用于匹配任意单个字符。

如果在 Unix 平台上使用特殊字符，则必须将选项括在引号内，以防止命令行解释器扩展选择范围。

2.1.1 --[no_]add_needed

此选项控制没有在命令行中指定的库的共享对象依赖性。

用法

--add_needed 设置应用于命令行中后跟的下一个 --no_add_needed 选项出现之前的所有共享对象。链接器将该共享对象依赖的所有共享对象添加到链接，并递归添加所有依赖共享对象。

缺省选项

如果使用 --arm_linux 选项则缺省为 --add_needed，否则缺省为 --no_add_needed。

示例

示例 2-1 显示如何使用此选项。

示例 2-1 --[no_]add_needed 选项

假定存在下列依赖性：

- cl1.so 依赖于 dep1.so
- cl2.so 依赖于 dep2.so
- cl3.so 依赖于 dep3.so
- dep2.so 依赖于 depofdep2.so。

使用以下命令行选项：

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

```
armlink --no_add_needed cl1.so --add_needed=cl2.so --no_add_needed cl3.so
```

结果是将下列共享对象添加到该链接：

- cl1.so
- cl2.so
- dep2.so
- depofdep2.so
- cl3.so

另请参阅

- *--arm_linux*

2.1.2 --arm_only

此选项使链接器可以仅将 ARM 指令集作为目标。如果链接器检测到任何对象需要 Thumb® 状态，则会生成错误。

2.1.3 --arm_linux

此选项指定创建 Linux 应用程序时使用的缺省设置。

缺省选项

自动指定以下缺省设置：

- --add_needed
- --keep=*(.init)
- --keep=*(.init_array)
- --keep=*(.fini)
- --keep=*(.fini_array)
- --linux_abitag=2.6.12
- --no_ref_cpp_init
- --no_scanlib
- --no_startup
- --sysv

若要覆盖任何缺省设置，请在 --arm_linux 选项后分别指定。

从 RVCT 4.0 版之前的工具链移植时，您可以使用一个 --arm_linux 选项替换所有缺省设置。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

另请参阅

- 第2-2 页的*--[no_]add_needed*
- 第2-30 页的*--keep=section_id*
- 第2-33 页的*--library=name*
- 第2-35 页的*--linux_abitag=version_id*
- 第2-46 页的*--[no_]ref_cpp_init*
- 第2-50 页的*--[no_]scanlib*
- 第2-51 页的*--[no_]search_dynamic_libraries*
- 第2-55 页的*--[no_]startup=symbol*
- 第2-57 页的*--sysv*
- 《编译器参考指南》中第2-8 页的*--arm_linux*

2.1.4 --[no_]autoat

此选项控制 __at 节到执行区的自动分配。__at 节是必须放在特定地址的节。

用法

如果启用，则链接器将自动为每个 __at 节选择执行区。如果不存在适用的执行区，则链接器将创建载入区和执行区以包含 __at 节。

如果禁用，则将应用标准分散加载节选择规则。

缺省选项

缺省为 --autoat。

另请参阅

- 第 3 章 *分散加载描述文件的形式语法*
- 《链接器用户指南》中第5-9 页的*指定区和节地址的示例*

2.1.5 --be8

此选项指定 ARMv6 字节固定寻址大端模式。

这是 ARMv6 大端映像的缺省字节寻址模式，意味着链接器反转指令的端标记，提供已按大端模式编译/汇编的输入对象的小端代码和大端数据。

字节固定寻址模式只在支持 ARMv6 和更高版本的 ARM 处理器上可用。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

另请参阅

- 《开发指南》中第2-11 页的*ARM 体系结构 v6*
- 《ARM 体系结构参考手册》

2.1.6 --be32

此选项指定旧的字固定寻址大端模式，即与 ARMv6 之前的大端映像相同。这会生成大端代码和数据。

字固定寻址模式是所有 ARMv6 之前的大端映像的缺省模式。

另请参阅

- 《开发指南》中第2-11 页的*ARM 体系结构 v6*
- 《ARM 体系结构参考手册》

2.1.7 --[no_]bestdebug

此选项在用于最小代码/数据大小或最佳调试效应的链接之间选择。输入对象可能包含 Comdat 组，但由于存在差异，因此这些组在所有输入对象中可能并不完全相同。例如，内联。

缺省选项

缺省为 --no_bestdebug。这确保了无论是否针对调试进行编译，最终映像的代码和数据都相同。链接时选中最小的 Comdat 组，代价是可能会造成略差的调试效应。

用法

使用 --bestdebug 可选择具有最佳调试视图的 Comdat 组。请注意，最终映像的代码和数据在带或不带调试选项进行构建时，可能会有所不同。

另请参阅

- 《链接器用户指南》中第3-10 页的 *节删除*

2.1.8 --bpabi

此选项创建 BPABI 可执行文件以传递到平台特定的后链接器。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

另请参阅

- 第2-17 页的*--dll*
- 第2-52 页的*--shared*
- 第2-57 页的*--sysv*
- 第 4 章 *BPABI 和 SysV 共享库和可执行文件*

2.1.9 --[no_]branchnop

此选项使链接器将所有跳转替换为重定位，该重定位可解析为具有 NOP 的下一条指令。这是缺省行为。但是，在某些情况下可能需要禁用该选项，例如，执行验证或管道刷新时。

缺省选项

缺省为 --branchnop。

使用 --no_branchnop 可禁用此行为。

另请参阅

- 第2-28 页的*--[no_]inline*
- 第2-58 页的*--[no_]tailreorder*
- 《链接器用户指南》中第3-20 页的*内联*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.1.10 --[no_]callgraph

此选项创建包含函数静态调用图的文件。调用图将为映像中的所有函数提供定义和参考信息。

用法

调用图文件：

- 与生成的映像保存在同一目录中。
- 其名称与链接的映像相同。可以使用 --callgraph_file=*filename* 选项指定其他调用图文件名。
- 缺省输出格式为 HTML。可以使用 --callgraph_output=*fmt* 选项控制输出格式。

注意

如果链接器要计算函数堆栈的使用情况，则在汇编器文件中定义的任意函数必须具有合适的 PROC/ENDP 和 FRAME PUSH/POP 指令。

对于每个函数 func，链接器列出以下内容：

- 编译函数的处理器状态 （ARM 或 Thumb）
- 调用 func 的函数集合
- func 调用的函数集合
- 映像中使用 func 地址的次数。

此外，调用图将标识以下函数：

- 通过交互操作中间代码调用的函数
- 在映像外定义的函数
- 允许保持未定义状态的函数 （弱引用）
- 未被调用但仍在映像中存在的函数。

静态调用图还提供有关堆栈的使用信息。其中列出：

- 每个函数使用的堆栈帧的大小
- 在所有调用序列 （即所有非循环函数调用链）中，函数使用的堆栈的最大大小。

如果有循环，或者链接器在调用链中检测到不带有堆栈大小信息的函数，则将 + Unknown 添加至堆栈使用情况中。将会添加一条原因以指示堆栈使用情况未知的原因。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

如果没有函数调试帧信息，链接器将报告缺少堆栈帧信息。

对于间接函数，链接器不能可靠地确定哪个函数发出了间接调用。这可能会影响为调用链计算最大堆栈用量的方式。链接器列出映像中使用的所有函数指针。

在汇编语言代码中使用帧指令可描述代码使用堆栈的情况。这些指令确保向调试器提供调试帧信息，以执行堆栈展开或性能分析。

缺省选项

缺省为 --no_callgraph。

另请参阅

- *--callgraph_file=filename*
- 第2-9 页的*--callgraph_output=fmt*
- 第2-9 页的*--cgfile=type*
- 第2-10 页的*--cgsymbol=type*
- 第2-10 页的*--cgundefined=type*
- 第 3 章 *分散加载描述文件的形式语法*
- 《汇编器指南》中第 7 章 *指令参考*

2.1.11 --callgraph_file=*filename*

此选项控制调用图的输出文件名。

缺省的文件名与链接的映像相同。

注意

如果使用 --partial 选项创建部分链接的对象，则不创建调用图文件。

另请参阅

- 第2-7 页的*--[no_]callgraph*
- 第2-9 页的*--callgraph_output=fmt*
- 第2-9 页的*--cgfile=type*
- 第2-10 页的*--cgsymbol=type*
- 第2-10 页的*--cgundefined=type*
- 第2-39 页的*--output=file*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

- 第 3 章 *分散加载描述文件的形式语法*

2.1.12 --callgraph_output=*fmt*

此选项控制调用图的输出格式。

语法

--callgraph_output=*fmt*

其中 *fmt* 可以是以下格式之一：

html　　以 HTML 格式输出调用图。

text　　以纯文本格式输出调用图。

缺省选项

缺省为 --callgraph_output=html。

另请参阅

- 第2-7 页的*--[no_]callgraph*
- 第2-8 页的*--callgraph_file=filename*
- *--cgfile=type*
- 第2-10 页的*--cgsymbol=type*
- 第2-10 页的*--cgundefined=type*
- 第 3 章 *分散加载描述文件的形式语法*

2.1.13 --cgfile=*type*

此选项控制使用什么文件获取调用图中包含的符号。

语法

--cgfile=*type*

其中 *type* 可以是以下之一：

all　　包括所有文件的符号。

user　　只包括用户定义的对象和库中的符号。

system　　只包括系统库中的符号。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

缺省选项

缺省为 --cgfile=all。

另请参阅

- 第2-7 页的*--[no_]callgraph*
- 第2-8 页的*--callgraph_file=filename*
- 第2-9 页的*--callgraph_output=fmt*
- *--cgsymbol=type*
- *--cgundefined=type*
- 第 3 章 *分散加载描述文件的形式语法*

2.1.14 --cgsymbol=*type*

此选项控制调用图中包含哪些符号。

语法

其中 *type* 可以是以下之一：

all	包括局部符号和全局符号。
locals	只包括局部符号。
globals	只包括全局符号。

缺省选项

缺省为 --cgsymbol=all。

另请参阅

- 第2-7 页的*--[no_]callgraph*
- 第2-8 页的*--callgraph_file=filename*
- 第2-9 页的*--callgraph_output=fmt*
- 第2-9 页的*--cgfile=type*
- *--cgundefined=type*
- 第 3 章 *分散加载描述文件的形式语法*

2.1.15 --cgundefined=*type*

此选项控制调用图中包含哪些未定义的引用。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

语法

--cgundefined=*type*

其中 *type* 可以是以下之一：

all	包括未定义弱引用的函数条目和调用。
entries	包括未定义弱引用的函数条目。
calls	包括对未定义弱引用的调用。
none	在输出中忽略所有未定义的弱引用。

缺省选项

缺省为 --cgundefined=all。

另请参阅

- 第2-7 页的*--[no_]callgraph*
- 第2-8 页的*--callgraph_file=filename*
- 第2-9 页的*--callgraph_output=fmt*
- 第2-9 页的*--cgfile=type*
- 第2-10 页的*--cgsymbol=type*
- 第 3 章 *分散加载描述文件的形式语法*

2.1.16 --[no_]combreloc

此选项启用或禁用链接器重新排序动态重定位，以便动态装入程序更有效地处理这些重定位。

缺省选项

缺省为 --combreloc。

2.1.17 --[no_]compress_debug

此选项使链接器在合理的情况下压缩 .debug_* 节。这将删除部分冗余并减小调试表大小。使用 --compress_debug 会显著增加链接映像所需的时间。调试压缩只能在 DWARF3 调试数据上执行，不能在 DWARF2 上执行。

缺省选项

缺省为 --no_compress_debug。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.1.18 --[no_]cppinit

此选项使链接器可以根据需要使用带有不同初始化符号的备选 C++ 库。

语法

--cppinit=*symbol*

如果未指定 *symbol*，则假定为缺省 __cpp_initialize__aeabi_。

--no_cppinit 不使用 *symbol* 参数。

缺省选项

缺省为 --no_cppinit。

另请参阅

- 第2-46 页的*--[no_]ref_cpp_init*

2.1.19 --cpu=list

此选项列出所支持的可以与 --cpu=*name* 结合使用的体系结构和处理器名称。

另请参阅

- *--cpu=name*

2.1.20 --cpu=*name*

此选项使链接器可以确定目标 ARM 处理器或体系结构。此选项的格式与编译器所支持的格式相同。

语法

--cpu=*name*

其中 *name* 是 ARM 处理器或体系结构的名称。

用法

如果任何组件对象文件依赖的功能与所选处理器不兼容，则链接阶段失败。链接器还使用此选项对选择的系统库以及任何在构建最终映像时需要生成的中间代码进行优化。缺省情况下选择与所有组件对象文件兼容的 CPU。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

另请参阅

- 第2-12 页的*--cpu=list*
- 《编译器参考指南》中第2-28 页的*--cpu=name*

2.1.21 --datacompressor=*opt*

此选项可用于为 RW 数据压缩指定一个提供的算法。如果您未指定数据压缩算法，则链接器将自动选择一种最合适的算法。通常不必覆盖此选择。

语法

--datacompressor=*opt*

其中 *opt* 是以下项之一：

on	启用 RW 数据压缩以最小化 ROM 大小。
off	禁用 RW 数据压缩。
list	列出可供链接器使用的数据压缩器。
id	*id* 是数据压缩算法。指定压缩器后，便会将解压缩器添加至代码区。如果最终映像没有压缩的数据，则不添加解压缩器。

缺省选项

缺省为 --datacompressor=on。

另请参阅

- 《链接器用户指南》中第3-15 页的*RW 数据压缩*

2.1.22 --[no_]debug

此选项控制在输出文件中生成调试信息。调试信息包括调试输入节以及符号/字符串表。

缺省选项

缺省为 --debug。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

用法

使用 --no_debug 从输出文件中排除调试信息。得到的 ELF 映像将变小，但不能在源代码级别对其进行调试。链接器放弃在输入对象和库成员中找到的任何调试输入节，并且不包括映像中的符号和字符串表。仅在载入到调试器时对映像的大小有影响。对下载到目标位置的任何最终二进制映像的大小没有任何影响。

如果使用 --partial，则链接器创建不包含任何调试数据的部分链接的对象。

注意

如果需要 fromelf --fieldoffsets 步骤，则不要使用 --no_debug。如果生成的映像没有调试信息，则 fromelf 无法：

- 将映像转换为其他文件格式
- 生成有意义的反汇编列表。

另请参阅

- 《实用程序指南》中第2-22 页的*--fieldoffsets*

2.1.23 --device=list

此选项列出可用于 --device=*name* 选项的受支持设备名称。

另请参阅

- *--device=name*

2.1.24 --device=*name*

此选项选择特定设备和关联的处理器设置。此选项遵循 ARM 编译器所支持的相同格式。

注意

如果任何组件对象文件依赖的功能与所选处理器不兼容，则链接阶段失败。链接器还使用此选项对选择的系统库以及任何在构建最终映像时需要生成的中间代码进行优化。缺省情况下选择与所有组件对象文件兼容的设备。

语法

--device=*name*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

其中 *name* 是特定设备的名称。

若要获取可用设备的完整列表，请使用 --device=list 选项。

另请参阅

- 第2-14 页的*--device=list*
- 《编译器参考指南》中第2-39 页的*--device=name*

2.1.25 --diag_error=*tag[,tag,...]*

此选项将具有特定标签的诊断消息设置为 “错误”严重性。

语法

--diag_error=*tag[,tag,...]*

其中 *tag* 是以逗号分隔的诊断消息编号的列表。

另请参阅

- *--diag_remark=tag[,tag,...]*
- 第2-16 页的*--diag_warning=tag[,tag,...]*

2.1.26 --diag_remark=*tag[,tag,...]*

此选项将具有特定标签的诊断消息设置为 “备注”严重性。

语法

--diag_remark=*tag[,tag,...]*

其中 *tag* 是以逗号分隔的诊断消息编号的列表。

另请参阅

- *--diag_error=tag[,tag,...]*
- 第2-16 页的*--diag_warning=tag[,tag,...]*

2.1.27 --diag_style=arm|ide|gnu

此选项更改警告和错误消息的格式。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

缺省选项

缺省为 --diag_style=arm。

用法

--diag_style=gnu 与由 GNU 编译器 gcc 报告的格式相匹配。

--diag_style=ide 与由 Microsoft Visual Studio 报告的格式相匹配。

2.1.28 --diag_suppress=*tag[,tag,...]*

此选项禁止所有具有特定标签的诊断消息。

语法

--diag_suppress=*tag[,tag,...]*

其中 *tag* 是必须禁止的诊断消息的编号列表 （以逗号分隔）。

示例

若要禁止显示编号为 L6314W 和 L6305W 的警告消息，请使用以下命令：

```
armlink --diag_suppress=L6314,L6305 ...
```

2.1.29 --diag_warning=*tag[,tag,...]*

此选项将具有特定标签的诊断消息设置为 “警告”严重性。

语法

--diag_warning=*tag[,tag,...]*

其中 *tag* 是以逗号分隔的诊断消息编号的列表。

另请参阅

- 第2-15 页的*--diag_error=tag[,tag,...]*
- 第2-15 页的*--diag_remark=tag[,tag,...]*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.1.30 --dll

此选项创建 BPABI *动态链接库* (DLL)。在 ELF 文件头中， DLL 标记为共享对象。

必须使用 --bpabi 和 --dll 才能生成 BPABI DLL。

另请参阅

- 第2-5 页的*--bpabi*
- 第2-52 页的*--shared*
- 第2-57 页的*--sysv*
- 第 4 章 *BPABI 和 SysV 共享库和可执行文件*

2.1.31 --dynamic_debug

此选项强制链接器输出调试节的动态重定位信息。通过此选项，可使用能够识别 OS 的调试器来调试 armlink 生成的共享库。

请将 --dynamic_debug 与 --sysv 以及 --sysv --shared 映像和共享库结合使用。

另请参阅

- 第2-52 页的*--shared*
- 第2-57 页的*--sysv*
- 第 4 章 *BPABI 和 SysV 共享库和可执行文件*

2.1.32 --dynamiclinker=*name*

此选项指定用于在运行时加载和重定位文件的动态链接器。

用法

与共享对象链接时，要使用的动态链接器存储在可执行文件中。此选项指定在该文件执行时要使用的特定的动态链接器。如果在 ARM Linux 平台上工作，则链接器假定缺省动态链接器为 /lib/ld-linux.so.3。

另请参阅

- 第 4 章 *BPABI 和 SysV 共享库和可执行文件*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.1.33 --edit=*file_list*

此选项可用于指定控制文件，其中包含用于编辑输出二进制中的符号表的命令。您可以在控制文件中指定命令来：

- 隐藏全局符号。使用此选项可隐藏对象文件中的特定全局符号。隐藏符号不是公共可见的。
- 重命名全局符号。使用此选项可解决符号命名冲突。

语法

--edit=*file_list*

其中 *file_list* 可以是由逗号分隔的多个控制文件。不要在逗号后面留有空格。

示例

```
--edit=file1 --edit=file2 --edit=file3
```

```
--edit=file1,file2,file3
```

另请参阅

- 第2-63 页的*控制文件命令*
- 第4-12 页的*隐藏和重命名全局符号*

2.1.34 --entry=*location*

此选项指定映像的唯一初始入口点。

语法

--entry=*location*

其中 *location* 是以下项之一：

entry_address
: 一个数值，例如：--entry=0x0

symbol
: 将映像入口点指定为 *symbol* 的地址，例如：--entry=reset_handler

offset+object(*section*)
: 将映像入口点指定为特定 *object* 的 *section* 中的 *offset*，例如：--entry=8+startup.o(startupseg)

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

在 --entry 的自变量中不能包含空格。匹配输入节和对象名时不区分大小写。可以使用以下简化表示法：

- 如果 offset 为零，则使用 object(section)。
- 如果只有一个输入节，则使用 object。如果 object 中有多个代码输入节，则 armlink 将生成一则错误消息。

注意

如果映像的入口地址在 Thumb 状态下，则地址的最低有效位必须设置为 1。如果指定了符号，则链接器会自动完成此操作。例如，如果入口代码在 Thumb 状态下从地址 0x8000 开始，则必须使用 --entry=0x8001。

用法

该映像可包含多个入口点，但使用此选项指定的初始入口点存储在可执行文件标题中，以供装入程序使用。此选项在命令行中只能使用一次。加载映像时，RealView® Debugger 使用此入口地址来初始化*程序计数器* (PC)。初始入口点必须满足以下条件：

- 映像入口点必须在执行区内
- 执行区必须是非重叠的，而且必须是根执行区 （加载地址 == 执行地址）。

另请参阅

- 第2-55 页的*--[no_]startup=symbol*

2.1.35 --errors=*file*

此选项将诊断消息从标准错误流重定向到 *file*。

指定的文件在链接阶段开始时创建。如果已存在同名的文件，则覆盖它。

如果指定 *file* 时不带路径信息，则将在当前目录中创建该文件。

2.1.36 --[no_]exceptions

此选项控制在最终映像中生成异常表。

缺省选项

缺省为 --exceptions。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

用法

如果删除未使用节之后映像中出现任何异常节，则使用 --no_exceptions 生成错误消息。使用此选项可确保代码中不出现异常。

另请参阅

- 《链接器用户指南》中第3-28 页的*使用命令行选项处理 C++ 异常*

2.1.37 --[no_]exceptions_tables=*action*

此选项指定如何为尚不包含异常展开表的对象生成异常表。

语法

--[no_]exceptions_tables=*action*

其中 *action* 是以下项之一：

nocreate　链接器不创建丢失异常表。

unwind　链接器为不带异常表的映像中的每个节创建一个展开表。

cantunwind　链接器为不带异常表的映像中的每个节创建一个非展开表。

缺省选项

缺省为 --exceptions_tables=nocreate。

另请参阅

- 《链接器用户指南》中第3-28 页的*使用命令行选项处理 C++ 异常*

2.1.38 --[no_]export_all

此选项控制将符号导出到动态符号表。

缺省选项

对于共享库和 DLL 缺省为 --export_all。

对于应用程序缺省为 --no_export_all。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

用法

使用 --export_all 从可执行文件或 DLL 中动态导出所有全局的非隐藏符号。使用 --no_export_all 可禁止将符号导出到动态符号表。

要更精确地控制符号导出，请使用一个或多个控制文件。

另请参阅

- 《链接器用户指南》中第4-12 页的*隐藏和重命名全局符号*

2.1.39 --feedback=*file*

此选项生成反馈文件，以便在下次编译文件时提示编译器有未使用的函数。

下次编译文件时，使用编译器选项 --feedback=*file* 指定要使用的反馈文件。未使用函数放置在各自所在的节中，以便链接器可能在未来进行删除。

另请参阅

- *--feedback_image=option*
- 第2-22 页的*--feedback_type=type*
- 《编译器用户指南》中第2-51 页的*--feedback=filename*
- 《链接器用户指南》中第3-13 页的*反馈*

2.1.40 --feedback_image=*option*

此选项更改链接器在写入反馈文件时的行为。在通常无法创建可执行 ELF 映像的情况下，使用此选项可生成反馈文件。

语法

--feedback_image=*option*

其中 *option* 是以下项之一：

none　使用分散加载文件确定区大小限制。禁用区重叠和区大小溢出消息。不写入 ELF 映像。如果区溢出了 32 位地址空间，则仍会生成错误消息。

noerrors　使用分散加载文件确定区大小限制。发出区重叠和区大小溢出警告消息。写入可能无法执行的 ELF 映像。如果区溢出了 32 位地址空间，则仍会生成错误消息。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

simple　忽略分散加载文件。禁用 ROPI/RWPI 错误和警告。写入可能无法执行的 ELF 映像。

full　启用所有错误和警告消息，并写入有效的 ELF 映像。

缺省选项

缺省选项为 --feedback_image=full。

示例

如果在使用链接器反馈删除未使用的函数之前，代码不适合分散加载文件中所描述的区限制，则使用 --feedback_image=noerrors。

另请参阅

- 第2-21 页的*--feedback=file*
- *--feedback_type=type*
- 《编译器用户指南》中第2-51 页的*--feedback=filename*
- 《链接器用户指南》中第3-13 页的*反馈*

2.1.41 --feedback_type=*type*

此选项控制链接器放到反馈文件中的信息。

语法

--feedback_type=*type*

其中 *type* 是由逗号分隔的以下主题关键字的列表：

[no]iw　控制需要交互操作支持的函数。

[no]unused　控制映像中未使用的函数。

缺省选项

缺省选项为 --feedback_type=unused,noiw。

另请参阅

- 第2-21 页的*--feedback=file*
- 第2-21 页的*--feedback_image=option*
- 《编译器用户指南》中第2-51 页的*--feedback=filename*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

- 《链接器用户指南》中第3-13 页的*反馈*

2.1.42 --[no_]filtercomment

此选项控制 .comment 节以改进常见的注释合并。如果希望禁用筛选器，可以使用 --no_filtercomment。

缺省选项

缺省为 --filtercomment。

2.1.43 --fini=*symbol*

此选项指定用于为最终化代码定义入口点的符号名。当动态链接器卸载可执行文件或共享对象时，将执行此代码。

另请参阅

- 第 4 章 *BPABI 和 SysV 共享库和可执行文件*

2.1.44 --first=*section_id*

此选项将选定的输入节放置为其执行区中的第一个节。例如，这样可以将包含向量表的节放置为映像中的第一个节。

语法

--first=*section_id*

其中 *section_id* 是以下项之一：

symbol 选择定义 *symbol* 的节。由于只能有一个节放置为第一个节，因此不能指定有多个定义的符号。例如：--first=reset

object(*section*)
从 *object* 中选择 *section*。在 *object* 和后面的左括号之间不能有空格。例如：--first=init.o(init)

object 选择 *object* 中的单个输入节。使用此短格式时，如果有多个输入节，则链接器会生成一则错误消息。例如：--first=init.o

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

注意

使用分散加载时，请在分散加载文件中改为使用 +FIRST。

另请参阅

- 《链接器用户指南》中第3-7 页的 *节放置*

2.1.45 --[no_]force_so_throw

此选项控制映像是否能抛出异常。缺省情况下，如果没有代码抛出异常，则丢弃异常表。

缺省选项

缺省为 --no_force_so_throw。

用法

使用 --force_so_throw 指定所有共享对象可能会抛出异常，因此强制链接器保留异常表，而无论映像是否能够抛出异常。如果使用了 --sysv 选项，则自动设置 --force_so_throw。

另请参阅

- 第2-57 页的*--sysv*
- 第 4 章 *BPABI 和 SysV 共享库和可执行文件*

2.1.46 --fpic

此选项可用于链接*位置无关代码* (PIC)，即使用 --apcs=/fpic 限定符编译的代码。只有在代码使用 System V 共享库时才执行相对寻址。

注意

如果使用了 --shared 但未使用 --fpic，则链接器将输出一个可降级错误。

另请参阅

- 第2-52 页的*--shared*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.1.47 --fpu=list

此选项列出所支持的可以与 --fpu=name 选项结合使用的 FPU 体系结构名称。

另请参阅

- *--fpu=name*

2.1.48 --fpu=*name*

此选项使链接器可以确定目标 FPU 体系结构。

如果任何组件对象文件依赖的功能与所选 FPU 体系结构不兼容，则链接器失败。链接器还使用此选项对选择的系统库以及任何在构建最终映像时需要生成的中间代码进行优化。缺省情况下选择与所有组件对象文件兼容的 FPU。

此选项的格式与编译器所支持的格式相同。

另请参阅

- *--fpu=list*
- 《编译器参考指南》中第2-56 页的*--fpu=name*

2.1.49 --help

此选项汇总显示主要的命令行选项。

缺省选项

如果指定 armlink 时未指定任何选项或源文件，则它是缺省选项。

另请参阅

- 第2-52 页的*--show_cmdline*
- 第2-61 页的*--vsn*

2.1.50 --info=*topic[,topic,...]*

此选项输出有关特定主题的信息。

语法

--info=*topic[,topic,...]*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

其中 *topic* 是由逗号分隔的以下主题关键字的列表：

`architecture` 通过列出 CPU、 FPU 和字节顺序来简要说明映像体系结构。

`common` 列出从映像中删除的所有公共节。使用此选项相当于 `--info=common,totals`。

`debug` 列出因使用 `--remove` 而从映像中删除的所有被拒绝的输入调试节。使用此选项相当于 `--info=debug,totals`。

`exceptions` 提供异常表生成和优化的信息。

`inline` 列出由链接器内联的所有函数以及内联的总数 （如果使用了 `--inline`）。

`inputs` 列出输入符号、对象和库。

`libraries` 列出为链接阶段自动选择的每个库的完整路径名。
可以将此选项与 `--info_lib_prefix` 一起使用，以显示有关特定库的信息。

`merge` 列出链接器合并的 **`const`** 字符串。每项列出合并结果、合并的字符串以及关联的对象文件。

`pltgot` 列出为可执行文件或 DLL 构建的 PLT 条目。

`sizes` 列出映像中每个输入对象和库成员的代码和数据 （RO 数据、 RW 数据、 ZI 数据和调试数据）大小。使用此选项相当于 `--info=sizes,totals`。

`stack` 列出所有全局符号的堆栈使用情况。

`summarysizes` 简要说明映像的代码和数据大小。

`summarystack` 简要说明所有全局符号的堆栈使用情况。

`tailreorder` 列出因为使用 `--tailreorder` 而移动到其目标上方的所有尾调用节。

`totals` 列出输入对象和库的代码和数据（RO 数据、RW 数据、ZI 数据和调试数据）的总大小。

`unused` 列出因使用 `--remove` 而从映像中删除的所有未使用节。

`veneers` 列出链接器生成的中间代码。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

veneercallers

列出链接器生成的中间代码，包括每个中间代码的调用者的附加信息。与 --verbose 结合使用，单独列出每个调用。

visibility 列出符号的可见性信息。此选项可以与 --info=inputs 或 --verbose 一起使用以增强输出。

--info=sizes,totals 的输出始终在输入对象和库的总大小中包含填充值。

如果使用 RW 数据压缩（缺省设置），或已使用 --datacompressor=*id* 选项指定了压缩器，则 --info=sizes,totals 的输出将在 Grand Totals 下方包括一个条目，反映映像的真实大小。

注意

列表中的主题关键字之间不允许有空格。例如，可以输入 --info=sizes,totals，但不能输入 --info=sizes, totals。

另请参阅

- 第2-13 页的*--datacompressor=opt*
- *--info_lib_prefix=opt*
- 第2-28 页的*--[no_]inline*
- 第2-38 页的*--[no_]merge*
- 第2-47 页的*--[no_]remove*
- 第2-58 页的*--[no_]tailreorder*
- 第2-59 页的*--verbose*
- 《链接器用户指南》中第3-15 页的*RW 数据压缩*
- 《链接器用户指南》中第3-29 页的*获得有关映像的信息*

2.1.51 --info_lib_prefix=*opt*

此选项是 --info=libraries 选项的筛选器。链接器仅显示具有与筛选器相同的前缀的库。

语法

armlink --info=libraries --info_lib_prefix=*opt*

其中 *opt* 是所需库的前缀。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

示例

- 不使用筛选器显示库的列表：

  ```
  armlink --info=libraries test.o
  ```

  生成库的列表，例如：

  ```
  install_directory\...\lib\armlib\c_4.l
  install_directory\...\lib\armlib\fz_4s.l
  install_directory\...\lib\armlib\h_4.l
  install_directory\...\lib\armlib\m_4s.l
  install_directory\...\lib\armlib\vfpsupport.l
  ```

- 使用筛选器显示库的列表：

  ```
  armlink --info=libraries --info_lib_prefix=c test.o
  ```

  生成具有指定前缀的库的列表，例如：

  ```
  install_directory\...\lib\armlib\c_4.l
  ```

2.1.52 --init=*symbol*

此选项指定用于定义初始化代码的符号名。当动态链接器加载可执行文件或共享对象时，将执行此代码。

另请参阅

- 第 4 章 *BPABI 和 SysV 共享库和可执行文件*

2.1.53 --[no_]inline

此选项允许或禁止跳转内联优化映像中的小函数调用。

缺省选项

缺省为 --no_inline。

——注意——

缺省情况下此跳转优化是关闭的，因为启用该功能会更改映像，从而可能导致调试信息不正确。如果启用，则链接器不会尝试更正调试信息。

另请参阅

- 第2-6 页的*--[no_]branchnop*
- 第2-58 页的*--[no_]tailreorder*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

- 《链接器用户指南》中第3-20 页的*内联*

2.1.54 --[no_]inlineveneer

此选项允许或禁止生成内联中间代码，以更好地控制链接器如何放置节。

缺省选项

缺省为 --inlineveneer。

另请参阅

- 第2-40 页的*--[no_]piveneer*
- 第2-59 页的*--[no_]veneershare*
- 《链接器用户指南》中第3-17 页的*中间代码*

2.1.55 *input_file_list*

这是以空格分隔的对象、库或*符号定义* (symdefs) 文件的列表。

用法

链接器将整个输入文件列表按照一定的顺序排列。如果链接器无法解决输入文件问题，则生成诊断消息。

可以将 symdefs 文件包括在列表中，为先前生成的映像文件提供全局符号值。

可以用以下方式来使用输入文件列表中的库：

- 指定要添加到库列表中的一个库，用于提取解析了任何未解析的非弱引用的成员。例如，指定输入文件列表中的 mystring.lib。

  注意

  仅当此列表中的库成员解析了未解析的非弱引用时，才将该列表中的库成员添加到映像中。

- 指定从库中提取的特定成员，并将它们作为单个对象添加到映像中。成员从由逗号分隔的可包含通配符的模式列表中选择。允许使用空格，但如果使用了空格，则必须用引号将整个输入文件列表引起来。

  以下分别是包含空格和不包含空格的输入文件列表的示例：

  mystring.lib(strcmp.o,std*.o)

  “mystring.lib(strcmp.o, std*.o)”

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

链接器自动搜索适当的 C 和 C++ 库以便为映像选择最佳的标准函数。您可以使用 --no_scanlib 来防止自动搜索标准库。

链接器按以下顺序处理输入文件列表：

1. 对象无条件添加到映像中。
2. 使用模式将从库中选择的成员无条件地添加到映像中 （如同添加对象一样）。例如，若要从 mystring.lib 添加所有 a*.o 对象和 stdio.o，请使用以下命令：
   "mystring.lib(stdio.o, a*.o)"
3. 将标准 C 或 C++ 库添加到库列表中，以后用于解析任何遗留的引用。

另请参阅

- 第2-50 页的*--[no_]scanlib*
- 《链接器用户指南》中第3-32 页的*库搜索、选择和扫描*
- 《链接器用户指南》中第4-9 页的*访问其他映像中的符号*

2.1.56 --keep=*section_id*

此选项指定删除未使用节时不能删除的输入节。

语法

--keep=*section_id*

其中 *section_id* 是以下项之一：

symbol
: 指定在删除未使用节时保留定义 *symbol* 的输入节。如果 *symbol* 存在多个定义，则 armlink 将生成一则错误消息。

  例如，您可能使用 --keep=int_handler。

  若要保留定义以 _handler 结尾的符号的所有节，请使用 --keep=*_handler。

object(*section*)
: 指定在删除未使用节时保留 *object* 中的 *section*。例如，若要保留 vectors.o 对象的 vect 节，请使用：--keep=vectors.o(vect)

  若要保留 vectors.o 对象中节名称的前三个字母是 vec 的所有节，请使用：
  --keep=vectors.o(vec*)

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

object 指定在删除未使用节时保留 *object* 中的单个输入节。使用此短格式时，如果 *object* 中存在多个输入节，则链接器会生成一则错误消息。

例如，您可能使用 --keep=dspdata.o。

要保留名称以 dsp 开头的每个对象中的单个输入节，请使用 --keep=dsp*.o。

section_id 参数的所有形式都可以包含 * 和 ? 通配符。可以在命令行中指定多个 --keep 选项。

另请参阅

- 《链接器用户指南》中第3-2 页的*指定映像结构*

2.1.57 --[no_]largeregions

此选项控制节在大执行区中的排序顺序，以最小化相互调用的节之间的距离。

用法

如果执行区中包含的代码超出跳转指令的范围，则链接器将切换至大区模式。在此模式中，链接器根据每节的大致平均调用深度对其升序排序。链接器可能还会在代码节之间放置分布中间代码以最小化中间代码的数量。

注意

大区模式下，即使输入发生很小变更，映像布局也会出现很大变动。

若要禁用大区模式切换回字典顺序，请使用 --no_largeregions。这样更容易判断节的位置和比较映像。但有些跳转可能无法到达目标导致链接步骤失败。如果出现这种情况，必须使用相应的分散加载说明文件或书写自定义中间代码显式放置代码/数据节。

缺省选项

缺省选项为 --no_largeregions。但如果至少一个执行区中包含的代码超出最小节间跳转的范围，则缺省选项为 --largeregions。最小节间跳转取决于该区和目标处理器中的代码：

32Mb 执行区仅包含 ARM。

16Mb 执行区包含 Thumb，支持 Thumb-2。

4Mb 执行区包含 Thumb，不支持 Thumb-2。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

另请参阅

- 第2-53 页的*--sort=algorithm*
- 《链接器用户指南》中第3-17 页的*中间代码*

2.1.58 --last=*section_id*

此选项将选定的输入节放置为其执行区中的最后一个节。例如，这样可以强制将包含校验和的输入节放置为 RW 节中的最后一个节。

语法

--last=*section_id*

其中 *section_id* 是以下项之一：

symbol 选择定义 *symbol* 的节。由于只能有一个节放置为最后一个节，因此不能指定有多个定义的符号。例如：--last=checksum

object(*section*)

从 *object* 中选择 *section*。在 *object* 和后面的左括号之间不能有空格。例如：--last=checksum.o(check)

object 选择 *object* 中的单个输入节。如果 *object* 中有多个输入节，则 armlink 将生成一则错误消息。

注意

使用分散加载时，请在分散加载文件中改为使用 +LAST。

另请参阅

- 《链接器用户指南》中第3-7 页的 *节放置*

2.1.59 --[no_]legacyalign

缺省情况下，链接器假定执行区和加载区是四字节对齐的。此选项使链接器可以将插入到映像中的填充数量减至最小。

--nolegacyalign 选项指示链接器插入填充以强制自然对齐。自然对齐是该区最高级别的已知对齐。

使用此选项可确保严格遵循 ELF 规范。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

缺省选项

缺省为 --legacyalign。

另请参阅

- 《链接器用户指南》中第3-7 页的 *节放置*

2.1.60 --library=*name*

此选项根据命令行上此点是否启用了动态库搜索，搜索名为 lib*name*.so 或 lib*name*.a 的库。

用法

动态搜索由 --[no_]search_dynamic_libraries 选项控制。

动态装入程序在运行时将对共享库的引用添加到映像并解析到库。将这些引用解析到库的顺序是命令行上指定库的顺序。这也是动态链接器解析依赖性的顺序。可以使用 --runpath 选项指定库的运行时位置。

示例

示例 2-2 搜索 libfoo.so 之后搜索 libfoo.a，但仅搜索 libbar.a。

示例 2-2 搜索动态库

使用带以下命令行选项的 armlink：

```
--arm_linux --shared --fpic --search_dynamic_libraries --library=foo
--no_search_dynamic_libraries --library=bar
```

另请参阅

- 第2-3 页的*--arm_linux*
- 第2-49 页的*--runpath=pathlist*
- 第2-51 页的*--[no_]search_dynamic_libraries*

2.1.61 --libpath=*pathlist*

此选项指定用于搜索 ARM 标准 C 和 C++ 库的路径列表。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

包含 ARM 库的父目录的缺省路径由 RVCT*ver*LIB 环境变量指定。此处指定的任何路径都将覆盖由环境变量指定的路径。

语法

--libpath=*pathlist*

其中 *pathlist* 是由逗号分隔的路径列表，仅用于搜索所需的 ARM 库。指定多个路径名时，在逗号和路径名之间不要留有空格，例如 *path1*,*path2*,*path3*,...,*pathn*。

注意

此选项不影响搜索用户库。应改为使用 --userlibpath。

另请参阅

- 第2-59 页的*--userlibpath=pathlist*
- 《链接器用户指南》中第3-32 页的*库搜索*、*选择和扫描*

2.1.62 --library_type=*lib*

此选项选择要在链接时使用的库。

注意

此选项可用于编译器、汇编器或链接器。

将此选项与链接器一起使用可覆盖所有其他 --library_type 选项。

语法

--library_type=*lib*

其中 *lib* 可以是下列项之一：

standardlib　　指定在链接时选择完整的 RealView 编译工具运行时库。

microlib　　指定在链接时选择 *C 微型库* (microlib)。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

缺省选项

如果在链接时不指定 --library_type 且没有对象文件指定引用，则链接器假定 --library_type=standardlib。

另请参阅

- 《链接器用户指南》中第3-4 页的*使用 microlib 构建应用程序*

2.1.63 --licretry

此选项指示当遇到某些 FLEXnet 错误代码时，链接器重新尝试验证许可证最多 10 次，每次间隔大约 10 秒。

2.1.64 --linux_abitag=*version_id*

此选项可为正在构建的可执行文件指定兼容的最低 Linux 内核版本。

另请参阅

- 第 4 章 *BPABI 和 SysV 共享库和可执行文件*

2.1.65 --list=*file*

此选项将 --info、--map、--symbols、 --verbose、--xref、--xreffrom 和 --xrefto 命令的输出中的诊断消息重定向到 *file*。

输出诊断消息时创建指定的文件。如果已存在同名的文件，则覆盖它。不过，如果不输出诊断消息，则不创建文件。这种情况下，任何现有的同名文件的内容将保持不变。

如果指定的 *file* 不带路径，则将在输出目录 （即写入输出映像的目录）中创建该文件。

另请参阅

- 第2-25 页的*--info=topic[,topic,...]*
- 第2-37 页的*--[no_]map*
- 第2-56 页的*--[no_]symbols*
- 第2-59 页的*--verbose*
- 第2-62 页的*--[no_]xref*
- 第2-62 页的*--[no_]xrefdbg*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.1.66 --[no_]list_mapping_symbols

此选项允许或禁止在由 --symbols 生成的输出中添加映射符号。例如：

$a ARM 代码

$t Thumb 代码

$d 数据。

映射符号用于标记 ARM 代码、 Thumb 代码和数据之间的转换。

缺省选项

缺省为 --no_list_mapping_symbols。

另请参阅

- 《ARM 体系结构的 ELF》(AAELF)。

2.1.67 --[no_]locals

此选项允许或禁止在生成可执行映像时向输出符号表中添加局部符号。

如果要减小输出符号表的大小，则 --no_locals 是非常有用的优化选项。

缺省选项

缺省为 --locals。

2.1.68 --ltcg

此选项启用链接时代码生成优化。如果任何输入对象由 --ltcg 编译，则必须使用此选项。

另请参阅

- 第2-44 页的*--profile=filename*
- 《编译器参考指南》中第2-78 页的*--ltcg*
- 《编译器参考指南》中第2-93 页的*--profile=filename*

2.1.69 --[un]mangled

此选项指示链接器在诊断消息以及 --xref、--xreffrom、--xrefto 和 --symbols 选项生成的列表中显示重整或未重整的 C++ 符号名。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

缺省选项

缺省为 --unmangled。

用法

如果选择 --unmangled，则 C++ 符号名以源代码中的方式显示。

如果选择 --mangled，则 C++ 符号名以对象符号表中的方式显示。

另请参阅

- 第2-56 页的*--[no_]symbols*
- 第2-62 页的*--[no_]xref*
- 第2-62 页的*--[no_]xrefdbg*
- 第2-62 页的*--xref{from|to}=object(section)*

2.1.70 --[no_]map

此选项启用或禁用内存映射打印。

映射包含映像中每个加载区、执行区和输入节 （包括链接器生成的输入节）的地址和大小。可以使用 --list 将其输出到文本文件。

缺省选项

缺省为 --no_map。

另请参阅

- 第2-35 页的*--list=file*
- 第2-52 页的*--section_index_display=type*

2.1.71 --match=crossmangled

此选项指示链接器将以下组合作为整体进行匹配：

- 对具有重整定义的未重整符号的引用
- 对具有未重整定义的重整符号的引用。

库和匹配组合按以下方式操作：

- 如果库成员定义了重整定义，并且存在未解析的未重整引用，则加载成员以满足它。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

- 如果库成员定义了未重整定义，并且存在未解析的重整引用，则加载成员以满足它。

注意

此选项与部分链接一起使用时无效果。即使存在重整定义，部分对象也包含所有对未重整符号的未解析引用。只在最后的链接步骤中完成匹配。

2.1.72 --max_visibility=*type*

此选项控制所有符号定义的可见性。

语法

--max_visibility=*type*

其中 *type* 可以是下列项之一：

default　　缺省可见性。

protected　　受保护可见性。

用法

使用 --max_visibility=protected 限制所有符号定义的可见性。如果指定此选项，则通常情况下拥有缺省可见性的全局符号定义将被赋予受保护可见性。

缺省选项

缺省为 --max_visibility=default。

2.1.73 --[no_]merge

此选项允许或禁止合并由编译器放置在可共享节中的 **const** 字符串。如果 **const** 字符串之间有相似性，则使用 --merge 可减小映像的大小。

要获得合并的 **const** 字符串的列表，可使用 --info=merge。

缺省选项

缺省为 --merge。

另请参阅

- 第2-25 页的*--info=topic[,topic,...]*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.1.74 --[no_]muldefweak

此选项启用或禁用符号的多重弱定义。

如果启用该选项，则链接器选择它遇到的第一个定义，并放弃所有其他重复定义。如果禁用该选项，则链接器为所有多重定义的弱符号生成错误消息。

缺省选项

缺省为 --no_muldefweak。

2.1.75 --output=*file*

此选项指定输出文件的名称。该文件可以是部分链接对象或可执行映像。

语法

--output=*file*

如果没有指定 --output=*file*，则链接器使用以下缺省文件名：

__image.axf　　如果输出是可执行映像

__object.o　　如果输出是部分链接对象。

如果指定 *file* 时不带路径信息，则它将在当前工作目录中创建。如果指定了路径信息，则该目录将成为缺省的输出目录。

另请参阅

- 第2-8 页的*--callgraph_file=filename*
- 第2-40 页的*--partial*

2.1.76 --override_visibility

此选项允许控制文件中的 EXPORT 和 IMPORT 指令覆盖符号可见性。

另请参阅

- 第2-64 页的*EXPORT*
- 第2-66 页的*IMPORT*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.1.77 --pad=*num*

此选项可用于设置填充字节的值。链接器将此值赋给插入加载区或执行区的所有填充字节。

语法

--pad=*num*

其中 *num* 是一个整数，可以用十六进制格式赋值。例如，将 *num* 设置为 0xFF 可能会有助于缩短 ROM 编程时间。如果 *num* 大于 0xFF，则填充字节将设置为 (char)*num*。

注意

填充只能插入：

- 载入区中。在载入区*之间*没有填充。
- 固定执行区之间 （除了强制对齐之外）。填充不能插入已达到最大长度的载入区，除非其顶部有固定的执行区。
- 节之间，以确保它们遵循对齐约束。

2.1.78 --partial

此选项创建一个可在后续链接步骤中使用的部分链接对象。

另请参阅

- 《链接器用户指南》中第2-3 页的*部分链接*

2.1.79 --[no_]piveneer

此选项允许或禁止为从 PI 代码到绝对代码的调用生成中间代码。使用 --no_piveneer 时，如果链接器检测到从 PI 代码到绝对代码的调用，则生成一则错误消息。

缺省选项

缺省为 --piveneer。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

另请参阅

- 第2-29 页的*--[no_]inlineveneer*
- 第2-59 页的*--[no_]veneershare*
- 《链接器用户指南》中第3-17 页的*中间代码*

2.1.80 --pltgot=*type*

此选项指定对应于不同 BPABI 寻址模式而使用的*过程链接表* (PLT) 和*全局偏移表* (GOT) 的类型。

语法

--pltgot=*type*

其中 *type* 为下列项之一：

none	将对导入符号的引用添加为动态重定位，供平台特定的后链接器处理。
direct	将对导入符号的引用解析为指向导入符号的只读指针。这些是直接指针引用。
indirect	链接器将为导入的符号创建 GOT 并在可能情况下创建 PLT 条目。该引用指向 PLT 或 GOT 条目。
sbrel	与 indirect 的引用相同，但有一项例外。GOT 条目将作为偏移量存储，该偏移量是运行时在 R9 中保留的段的静态基址的偏移量。

缺省选项

使用选项 --bpabi 或 --dll 时，缺省为 --pltgot=direct。

2.1.81 --pltgot_opts=*mode*

此选项在生成 PLT 条目时启用或禁用弱引用。

语法

--pltgot_opts=*mode*

其中 *mode* 为下列项之一：

weakrefs	弱引用生成 PLT 条目。这些引用必须在后面的链接阶段中解析。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

noweakrefs　为函数调用生成 NOP，或为数据生成零。不生成 PLT 条目。对导入符号的弱引用保留为未解析状态。

缺省选项

缺省为 --pltgot_opts=noweakrefs。

2.1.82 --predefine="*string*"

此选项使命令可以传递到分散加载文件的第一行中指定的预处理器。您还可以使用同义词：--pd="*string*"。

语法

--predefine=" *string*"

在命令行上可以使用多个 --predefine 选项。请将此选项与 --scatter 结合使用。

示例

示例 2-3 显示了分散加载文件在预处理之前的外观。

示例 2-3 预处理之前的分散加载文件

```
Scatter file:
#! armcc -E
lr1 BASE
{
    er1 BASE
    {
        *(+RO)
    }
    er2 BASE2
    {
        *(+RW+ZI)
    }
}
```

使用带以下命令行选项的 armlink：

```
--predefine="-DBASE=0x8000" --predefine="-DBASE2=0x1000000" --scatter=file
```


Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

这会将命令行选项：-DBASE=0x8000 -DBASE2=0x1000000 传递到编译器以预处理分散加载文件。

示例 2-4 显示了分散加载文件在预处理之后的外观：

示例 2-4 预处理之后的分散加载文件

```
lr1 0x8000
{
    er1 0x8000
    {
        *(+RO)
    }
    er2 0x1000000
    {
        *(+RW+ZI)
    }
}
```

2.1.83 --[no_]prelink_support

此选项允许或禁止链接器将附加的空程序头表条目添加到应用程序，并将某些附加的 DT_NULL 动态标签添加到应用程序和共享库。预链接工具使用此保留的空间书写动态装入程序所需的附加信息。

缺省选项

如果指定了 --sysv，则缺省为 --prelink_support。

可以使用 --no_prelink_support 强制链接器不保留此空间。

另请参阅

- 第2-57 页的*--sysv*

2.1.84 --privacy

此选项将节名称和所有本地符号更改为缺省值，但映射和构建属性符号除外。使用此选项可以隐藏传递到第三方的映像和对象中的 IP。

例如，代码节的名称将更改为 **.text**。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

另请参阅

- 《实用程序指南》中第2-36 页的*--privacy*
- 《实用程序指南》中第2-43 页的*--strip=option[,option,...]*

2.1.85 --profile=*filename*

此选项允许将性能分析驱动反馈传递回编译器。

用法

如果使用链接时代码生成 (--ltcg)，则可以将性能分析器文件 *filename* 传递回编译器，以便编译器可以使用性能分析驱动反馈。

另请参阅

- 第2-36 页的*--ltcg*
- 《编译器参考指南》中第2-78 页的*--ltcg*
- 《编译器参考指南》中第2-93 页的*--profile=filename*

2.1.86 --[no_]project=*filename*

此选项指示链接器加载指定的项目模板文件。

语法

```
--no_project
--project=filename
```

其中 *filename* 是项目模板文件的名称。

注意

若要使用 *filename* 作为缺省项目文件，请将 RVDS_PROJECT 环境变量设置为 *filename*。

--no_project 禁止使用环境变量 RVDS_PROJECT 所指定的缺省项目模板文件。

缺省选项

缺省为 --no_project。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

限制

仅当项目模板文件中的选项与命令行中已设置的选项不发生冲突时，才会设置前者。如果项目模板文件中的选项与现有命令行选项发生冲突，则后者优先。

示例

请考虑以下项目模板文件：

```
<!-- suiteconf.cfg -->
<suiteconf name="Platform Baseboard for ARM926EJ-S">
    <tool name="armlink">
        <cmdline>
            --cpu=ARM926EJ-S
            --fpu=vfpv2
        </cmdline>
    </tool>
</suiteconf>
```

当将 RVDS_PROJECT 环境变量设置为指向此文件时，以下命令：

```
armlink foo.o
```

将生成以下一行实际的命令：

```
armlink --cpu=ARM926EJ-S --fpu=VFPv2 foo.o
```

另请参阅

- 第2-47 页的*--reinitialize_workdir*
- 第2-61 页的*--workdir=directory*

2.1.87 --[no_]reduce_paths

此选项允许或禁止删除文件路径中的冗余路径名信息。

模式

仅在 Windows 系统中有效。

缺省选项

缺省为 --no_reduce_paths。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

用法

Windows 系统对文件路径有 260 个字符的限制。如果存在其绝对路径名长度超过 260 个字符的路径名，则可以使用 --reduce_paths 选项，该选项通过将目录与对应的 .. 实例相匹配并成对删除 directory/.. 序列，可以缩短绝对路径名的长度。

注意

建议优先使用 --reduce_paths 选项来尽量缩短路径长度，并避免使用长文件路径和深层嵌套文件路径。

示例

链接以下文件：

```
..\..\..\xyzzy\xyzzy\objects\file.c
```

该文件位于以下目录中：

```
\foo\bar\baz\gazonk\quux\bop
```

这会生成以下实际路径：

```
\foo\bar\baz\gazonk\quux\bop\..\..\..\xyzzy\xyzzy\objects\file.o
```

如果使用 --reduce_paths 选项从同一目录中链接同一文件，则会生成以下实际路径：

```
\foo\bar\baz\xyzzy\xyzzy\objects\file.c
```

2.1.88 --[no_]ref_cpp_init

此选项允许或禁止链接器添加到 RVCT 库中的 C++ 静态对象初始化例程的引用。添加的缺省引用是 __cpp_initialize__aeabi_。可以使用 --cppinit 进行更改。

缺省选项

缺省为 --ref_cpp_init。

另请参阅

- 第2-12 页的*--[no_]cppinit*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.1.89 --reinitialize_workdir

此选项使您可以使用 --workdir 重新初始化项目模板工作目录。

当使用 --workdir 设置的目录引用包含已修改项目模板文件的现有工作目录时，指定此选项时会导致删除该工作目录并用原始项目模板文件的新副本重新创建该目录。

限制

此选项必须与 --workdir 选项结合使用。

另请参阅

- 第2-44 页的*--[no_]project=filename*
- 第2-61 页的*--workdir=directory*

2.1.90 --reloc

此选项创建一个包含几个连续执行区的可重定位加载区。

注意

此选项仅用于带有此种可重定位的 ELF 映像的旧系统。生成的映像可能与 ARM 体系结构规范的 ELF 稍有不同。

对于新系统，可以考虑使用适用于 BPABI 的映像。

另请参阅

- 《链接器用户指南》中第5-31 页的*类型 1，一个加载区和几个连续执行区*

2.1.91 --[no_]remove

此选项允许或禁止从映像中删除未使用的输入节。如果输入节包含映像入口点，或者被已使用节引用，则认为它是已使用节。

缺省选项

缺省为 --remove。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

用法

在调试时使用 --no_remove 可在最终映像中保留所有输入节 （即使它们是未使用的）。

使用 --keep 选项可在正常构建中保留指定节。

另请参阅

- 第2-30 页的*--keep=section_id*
- 《链接器用户指南》中第3-10 页的 *节删除*

2.1.92 --ro_base=*address*

此选项将包含 RO 输出节的区的加载和执行地址都设置到指定的地址。

语法

--ro_base=*address*

其中 *address* 必须是字对齐的。

缺省选项

如果未指定此选项，并且没有指定分散加载文件，则缺省为 --ro_base=0x8000。

另请参阅

- *--ropi*
- 第2-49 页的*--rosplit*
- 第2-49 页的*--rw_base=address*
- 第2-50 页的*--rwpi*
- 第2-51 页的*--scatter=file*

2.1.93 --ropi

此选项将包含 RO 输出节的加载区和执行区设置为与位置无关。如果未使用此选项，则将区标记为绝对地址区。通常每个只读输入节必须为*只读位置无关* (ROPI)。如果选择了此选项，则链接器将：

- 检查节之间的重定位信息的有效性
- 确保链接器本身生成的所有代码 （如交互操作中间代码）都为 ROPI。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

注意

如果使用不带 --rwpi 或 --rw_base 的 --ropi，则链接器将显示一个可降级错误。

另请参阅

- 第2-48 页的*--ro_base=address*
- *--rosplit*
- *--rw_base=address*
- 第2-50 页的*--rwpi*

2.1.94 --rosplit

此选项将缺省 RO 加载区分为两个 RO 输出节，一个用于 RO-CODE，另一个用于 RO-DATA。

另请参阅

- 第2-48 页的*--ro_base=address*
- 第2-48 页的*--ropi*
- *--rw_base=address*
- 第2-50 页的*--rwpi*

2.1.95 --runpath=*pathlist*

此选项指定要添加到搜索路径的路径的列表。Linux 动态链接器使用这些路径定位所需的共享对象。

语法

--runpath=*pathlist*

其中 *pathlist* 是用逗号分隔的路径列表。指定多个路径名时，在逗号和路径名之间不要留有空格，例如 *path1*,*path2*,*path3*,...,*pathn*。

2.1.96 --rw_base=*address*

此选项将包含 RW 输出节的区的执行地址设置到指定的地址。

语法

--rw_base=*address*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

其中 *address* 必须是字对齐的。

另请参阅

- 第2-48 页的*--ro_base=address*
- 第2-48 页的*--ropi*
- 第2-49 页的*--rosplit*
- *--rwpi*
- 第2-54 页的*--split*

2.1.97 --rwpi

此选项将包含 RW 和 ZI 输出节的加载区和执行区设置为与位置无关。如果未使用此选项，则将区标记为绝对地址区。此选项需要 --rw-base 的值。如果未指定 --rw-base，则假定为 --rw-base=0。通常每个可写输入节必须为可读写的位置无关代码 (RWPI)。

如果选择了此选项，则链接器将：

- 检查是否在任何读写执行区的输入节上设置了 PI 属性
- 检查节之间的重定位信息的有效性
- 生成与表 Region$$Table 中的静态基址相关的条目。

  这在复制、解压缩或初始化区时使用。

另请参阅

- 第2-48 页的*--ro_base=address*
- 第2-48 页的*--ropi*
- 第2-49 页的*--rosplit*
- 第2-49 页的*--rw_base=address*

2.1.98 --[no_]scanlib

此选项允许或禁止扫描缺省库以解析引用。如果想链接自己的库，可以使用 --no_scanlib。

缺省选项

缺省为 --scanlib。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.1.99 --scatter=*file*

此选项使用指定文件中包含的分散加载描述创建映像内存映射。该描述提供映像中各个区和节的分组和布局的详细信息。

语法

--scatter=*file*

其中 *file* 是分散加载文件的名称。

用法

--scatter 选项不能与
--bpabi、--dll、--partial、--ro-base、--rw-base、--ropi、--rwpi
--rosplit、--split、--reloc、--shared、--startup 和 --sysv 一起使用。

另请参阅

- 《链接器用户指南》中第 5 章 *使用分散加载描述文件*

2.1.100 --[no_]search_dynamic_libraries

此选项允许或禁止按照 --library 选项的指定搜索动态库。

用法

--search_dynamic_libraries 设置应用于命令行中后跟的下一个
--no_search_dynamic_libraries 选项出现之前的所有 --library 选项。链接器按照命令行顺序重复搜索库，直到所有引用得以解析或指定的库无法进一步解析引用。

缺省选项

缺省为 --search_dynamic_libraries。

另请参阅

- 第2-3 页的*--arm_linux*
- 第2-33 页的*--library=name*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.1.101 --section_index_display=*type*

此选项会在打印内存映射输出时更改索引列的外观。请将此选项与 --map 结合使用。

语法

--section_index_display=*type*

其中 *type* 是以下项之一：

internal　索引值按链接器创建每节的顺序显示。

input　索引值显示原始输入文件中每节的节索引。

用法

如果要在输入对象中精确查找某节，可以一起使用 --map 和 --section_index_display=input。

缺省选项

缺省为 --section_index_display=internal。

另请参阅

- 第2-37 页的*--[no_]map*

2.1.102 --shared

此选项创建 SysV 共享对象。

另请参阅

- 第2-5 页的*--bpabi*
- 第2-17 页的*--dll*
- 第2-49 页的*--runpath=pathlist*
- 第2-53 页的*--soname=name*
- 第2-57 页的*--sysv*
- 第 4 章 *BPABI 和 SysV 共享库和可执行文件*

2.1.103 --show_cmdline

此选项显示处理命令行选项的方式。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

命令以其首选形式显示，并展开所有 via 文件的内容。

另请参阅

- 第2-25 页的*--help*
- 第2-61 页的*--vsn*

2.1.104 --soname=*name*

此选项指定共享对象运行时名称，该名称由链接到此共享对象的任何对象用作依赖性名称。此依赖性存储在生成的文件中。

另请参阅

- 第 4 章 *BPABI 和 SysV 共享库和可执行文件*

2.1.105 --sort=*algorithm*

此选项指定在执行区中确定节顺序的排序算法，以最小化相互调用的节之间的距离。排序算法与标准规则一致，将输入节按属性升序排序。

也可以使用 SORTTYPE 关键字在分散加载文件中为每个执行区分别指定排序算法。

语法

--sort=*algorithm*

其中 *algorithm* 是以下项之一：

List　提供可用排序算法的列表。链接器在显示列表后再确定。

Lexical　根据节的名称排序，如果名称相同则根据输入顺序排序。

AvgCallDepth　将所有 Thumb 代码排在 ARM 代码前面，然后根据每节的大致平均调用深度升序排序。

使用此算法有助于最小化长跳转中间代码数量。

注意

大致平均调用深度取决于输入节的顺序。因此，此排序算法比其他算法 （比如 RunningDepth）更依赖于输入节的顺序。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

CallTree 链接器将调用树压缩到一个列表，该列表中包含来自启用了 CallTree 排序的所有执行区的只读代码节。

此列表中的节将被复制回它们的执行区，放在所有非只读代码节之前，按字典顺序排序。这样做可确保相互调用的节之间距离较近。

注意

此排序算法与 RunningDepth 或 AvgCallDepth 相比，更不依赖于输入节的顺序。

RunningDepth 将所有 Thumb 代码排在 ARM 代码前面，然后根据每节的运行深度升序排序。节 S 的运行深度是调用 S 的所有节的平均调用深度，并根据它们调用 S 的次数加权。

使用此算法有助于最小化长跳转中间代码数量。

缺省选项

缺省算法为 --sort=Lexical。但只要有一个执行区超出最大跳转大小，缺省算法就为 --sort=AvgCallDepth。

另请参阅

- 第3-6 页的*执行区描述*
- 《链接器用户指南》中第3-7 页的 *节放置*

2.1.106 --split

此选项将包含 RO 和 RW 输出节的缺省加载区分成以下加载区：

- 一个包含 RO 输出节的区。缺省的加载地址是 0x8000，但可以使用 --ro_base 选项指定其他地址。
- 一个包含 RW 和 ZI 输出节的区。使用 --rw_base 选项指定加载地址。此选项需要为 --rw_base 指定值。如果未指定 --rw_base，则假定为 --rw_base=0。

两个区都是根区。

另请参阅

- 第2-48 页的*--ro_base=address*
- 第2-49 页的*--rw_base=address*
- 《链接器用户指南》中第3-2 页的*指定映像结构*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.1.107 --[no_]startup=*symbol*

此选项使链接器可以根据需要使用带有不同启动符号的备选 C 库。

语法

--startup=*symbol*

缺省情况下， *symbol* 设置为 __main。

--no_startup 不使用 *symbol* 参数。

缺省选项

缺省为 --startup=__main。

用法

如果存在对由 C 库启动代码定义的符号的引用，则链接器将包括 C 库启动代码。此符号引用称为启动符号。它在链接器发现 main() 的定义时自动创建。使用 --startup 选项可以更改此符号引用。

- 如果链接器找到了 main() 的定义，但找不到对 *symbol* 的引用 （或定义），则链接器会生成错误。
- 如果链接器找到了 main() 的定义和对 *symbol* 的引用 （或定义），但没有指定入口点，则链接器将生成警告。

另请参阅

- 第2-18 页的*--entry=location*

2.1.108 --strict

此选项指示链接器报告可能会导致失败 （出现错误，而不是警告）的情况。这种情况的一个示例是：从非交互函数获取交互函数的地址。

2.1.109 --[no_]strict_relocations

此选项可用于确保旧对象或第三方对象的*应用程序二进制接口* (ABI) 兼容性。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

用法

可使用 --strict_relocations 指示链接器报告已过时的和不提倡使用的重定位的实例。

在链接使用先前版本的 ARM 工具构建的对象文件时，最有可能发生重定位错误和警告。

缺省选项

缺省为 --no_strict_relocations。

2.1.110 --[no_]symbols

此选项允许或禁止列出链接步骤中使用的每一个局部和全局符号及其值。

注意

这不包括映射符号。使用 --list_mapping_symbols 可在输出中包括映射符号。

缺省选项

缺省为 --no_symbols。

另请参阅

- 第2-36 页的*--[no_]list_mapping_symbols*

2.1.111 --symdefs=*file*

此选项从输出映像创建包含全局符号定义的文件。

缺省选项

缺省情况下，所有全局符号都写入到 symdefs 文件中。如果名为 *file* 的 symdefs 文件已经存在，则链接器将其输出限制为此文件中已列出的符号。

注意

如果不希望出现此行为，请确保在链接步骤之前删除所有现有的 symdefs 文件。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

用法

如果指定的 *file* 不带路径信息，则链接器将在写入输出映像的目录中搜索它。如果未找到该文件，则在该目录中创建它。

在与其他映像链接时，可以将符号定义文件用作输入。

另请参阅

- 《链接器用户指南》中第4-9 页的*访问其他映像中的符号*

2.1.112 --symver_script=*file*

此选项启用隐式符号版本控制，其中 *file* 是符号版本脚本。

另请参阅

- 《链接器用户指南》中第4-15 页的*符号版本控制*

2.1.113 --symver_soname

此选项启用隐式符号版本控制以强制实施静态绑定。如果符号没有已定义的版本，链接器将使用所链接的文件的 SONAME。

缺省选项

如果要生成 BPABI 兼容的可执行文件，但其中未使用选项 --symver_script 指定版本脚本，则这是缺省设置。

另请参阅

- 《链接器用户指南》中第4-15 页的*符号版本控制*
- 《ARM 体系结构的基本平台 ABI》(BPABI)。

2.1.114 --sysv

此选项创建可在 ARM Linux 上使用的 SysV 格式的 ELF 可执行文件。

另请参阅

- 第2-5 页的*--bpabi*
- 第2-17 页的*--dll*
- 第2-43 页的*--[no_]prelink_support*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

- 第2-49 页的*--runpath=pathlist*
- 第2-52 页的*--shared*
- 第 4 章 *BPABI 和 SysV 共享库和可执行文件*

2.1.115 --[no_]tailreorder

此选项在可能的情况下将尾调用节移到紧靠其目标的前面，以优化节末尾的跳转指令。尾调用节是在节的末尾包含跳转指令的节。跳转必须具有在节的开始位置定位函数的重定位信息。

缺省选项

缺省为 --no_tailreorder。

限制

链接器：

- 只能为每个尾调用目标移动一个尾调用节。如果存在对单个节的多个尾调用，则具有相同节名称的尾调用节将移到目标前面。如果在具有匹配名称的尾调用节中未找到节名称，则链接器将移动所遇到的*第一个*节。
- 不能将尾调用节移动到其执行区之外。
- 不会将尾调用节移动到内联中间代码之前。

另请参阅

- 第2-6 页的*--[no_]branchnop*
- 第2-28 页的*--[no_]inline*
- 《链接器用户指南》中第3-20 页的*内联*

2.1.116 --unresolved=*symbol*

此选项提取每个对未定义符号的引用并将其与指定符号的全局定义匹配。

语法

--unresolved=*symbol*

其中 *symbol* 必须已定义且是全局的，否则它会显示在未定义符号列表中，且链接步骤失败。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

用法

此选项在自上而下的开发中尤为有用，因为它可以通过将每个对缺失函数的引用与虚拟函数匹配来测试部分实现的系统。

2.1.117 --userlibpath=*pathlist*

此选项指定用于搜索用户库的路径列表。

语法

--userlibpath=*pathlist*

其中 *pathlist* 是由逗号分隔的路径列表，用于搜索所需的库。指定多个路径名时，在逗号和路径名之间不要留有空格，例如 *path1*,*path2*,*path3*,...,*pathn*。

另请参阅

- 第2-33 页的*--libpath=pathlist*
- 《链接器用户指南》中第3-32 页的*库搜索*、*选择和扫描*

2.1.118 --[no_]veneershare

此选项启用或禁用中间代码共享。中间代码共享可以显著地减小映像大小。

缺省选项

缺省为 --veneershare。

另请参阅

- 第2-29 页的*--[no_]inlineveneer*
- 第2-40 页的*--[no_]piveneer*
- 《链接器用户指南》中第3-17 页的*中间代码*

2.1.119 --verbose

此选项输出关于链接操作的详细信息，包括所包含的对象和从中提取对象的库。此输出在跟踪未定义的符号引用或多重定义的符号时非常有用。因为此输出通常很长，所以可能需要将此命令与 --list=*file* 命令结合使用，将信息重定向到 *file*。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

使用 --verbose 可将诊断消息输出到 stdout。

另请参阅

- 第2-39 页的*--[no_]muldefweak*
- 第2-58 页的*--unresolved=symbol*

2.1.120 --vfemode=*mode*

虚函数删除 (VFE) 是一种使链接器可以识别更多未使用节的技术。

使用此选项指定如何删除 VFE 和*运行时类型信息* (RTTI) 对象。

语法

--vfemode=*mode*

其中 *mode* 为下列项之一：

on
使用命令行选项 --vfemode=on 可启用链接器 VFE 识别。

在此模式下，链接器会根据对象文件的内容选择 force 或 off 模式：

- 如果每个对象文件都包含 VFE 信息或未引用 C++ 库，则链接器将使用 force 模式，并继续进行删除。
- 如果有任何对象文件缺少 VFE 信息并引用了 C++ 库，例如使用以前版本的 ARM 工具编译的代码，则链接器将使用 off 模式，自动禁用 VFE。此时选择 off 模式以禁用 VFE，可以确保链接器不会删除不含 VFE 信息的对象使用的虚函数。

off
使用命令行选项 --vfemode=off 可使 armlink 忽略编译器提供的任何附加信息。在此模式下，最终映像将与 VFE 不可识别情况下编译和链接生成的映像相同。

force
使用命令行选项 --vfemode=force 可启用链接器 VFE 识别，并强制应用 VFE 算法。如果某些对象文件不包含 VFE 信息，例如，使用以前版本的 ARM 工具编译的文件，链接器会在继续删除的同时显示一则警告消息，提示您可能发生错误。

force_no_rtti
使用命令行选项 --vfemode=force_no_rtti 可以启用链接器 VFE 识别，并强制删除所有 RTTI 对象。在此模式下，所有虚函数都将被保留。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

缺省选项

缺省为 --vfemode=on。

另请参阅

- 《链接器用户指南》中第3-10 页的 *节删除*

2.1.121 --via=*file*

此选项从 *file* 中读取输入文件名和链接器选项的详细列表。

可以在链接器命令行中输入多个 --via 选项。--via 选项也可以包括在 via 文件中。

另请参阅

- 《编译器参考指南》中第A-2 页的*via 文件概述*

2.1.122 --vsn

此选项显示版本信息和许可证详细信息。

另请参阅

- 第2-25 页的*--help*
- 第2-52 页的*--show_cmdline*

2.1.123 --workdir=*directory*

此选项用于为项目模板提供工作目录。

注意

项目模板只有在包含文件 （如 RealView Debugger 配置文件）时才需要工作目录。

语法

--workdir=*directory*

其中 *directory* 是项目目录的名称。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

限制

如果使用 --workdir 指定项目工作目录，则必须使用 --project 指定项目文件。

另请参阅

- 第2-44 页的*--[no_]project=filename*
- 第2-47 页的*--reinitialize_workdir*

2.1.124 --[no_]xref

此选项列出输入节之间的所有交叉引用。

缺省选项

缺省为 --no_xref。

另请参阅

- *--xref{from|to}=object(section)*

2.1.125 --[no_]xrefdbg

此选项列出输入调试节之间的所有交叉引用。

缺省选项

缺省为 --no_xrefdbg。

2.1.126 --xref{from|to}=*object(section)*

此选项列出交叉引用：

- 从 *object* 中的 输入 *section* 到其他输入节
- 从其他输入节到 *object* 中的 输入 *section*。

如果想知道来自/指向特定输入节的引用，可以使用 --xref 链接器选项生成的列表子集。可以多次使用此选项，列出来自/指向多个输入节的引用。

另请参阅

- *--[no_]xref*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.2 控制文件命令

本节按字母顺序列出了链接器所支持的控制文件命令。

可以使用控制文件命令执行以下操作：

- 管理符号表中的符号
- 控制将符号从静态符号表复制到动态符号表中的过程
- 存储有关链接单元所依赖的库的信息。

注意

控制文件命令仅控制全局符号。局部符号不受任何命令的影响。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.2.1 EXPORT

EXPORT 命令指定可以由其他共享对象或可执行文件访问符号。

语法

EXPORT *pattern* [AS *replacement_pattern*] [,*pattern* [AS *replacement_pattern*]] *

其中：

pattern 是一个与零个或多个已定义的全局符号相匹配的字符串，可以选择在其中包含通配符 （* 或 ?）。如果 *pattern* 与任何已定义的全局符号均不匹配，链接器将忽略该命令。此操作数只能与已定义的全局符号相匹配。

replacement_pattern

是一个要将已定义的全局符号重命名为的字符串，可以选择在其中包含通配符 （* 或 ?）。此处的通配符必须在 *pattern* 中具有对应的通配符。*replacement_pattern* 通配符匹配的字符将替换 *pattern* 通配符。

例如：

```
EXPORT my_func AS func1
```

将已定义的符号 my_func 重命名为 func1 并导出。

用法

无法将符号导出为已存在的名称。EXPORT 中仅允许使用一个通配符 （* 或 ?）。

如果存在动态符号表，则将已定义的全局符号包括在动态符号表中 （如果指定，则作为 *replacement_pattern*；否则，作为 *pattern*）。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.2.2 HIDE

HIDE 命令将符号表中已定义的全局符号变为匿名符号。

语法

HIDE *pattern* [,*pattern*] *

其中：

pattern 是一个与零个或多个已定义的全局符号相匹配的字符串，可以选择在其中包含通配符。如果 *pattern* 与任何已定义的全局符号均不匹配，链接器将忽略该命令。无法隐藏未定义的符号。

用法

可以使用 HIDE 和 SHOW，将输出映像或部分链接的对象中的某些全局符号变为匿名对象。可以将在对象文件或库中隐藏符号作为一种保护知识产权的方法，如示例 2-5 中所示。此示例生成一个部分链接的对象并隐藏所有全局符号，以 os_ 开头的全局符号除外。

示例 2-5 使用 HIDE 命令

```
steer.txt

HIDE *            ; Hides all global symbols
SHOW os_*         ; Shows all symbols beginning with ' os_'
```

将此示例与以下命令链接在一起：

```
armlink --partial input_object.o --edit steer.txt -o partial_object.o
```

可以将结果部分对象与其他对象链接在一起，但前提是这些对象不包含对隐藏符号的引用。如果在输出对象中隐藏了符号，后续链接步骤中的 SHOW 命令对这些符号无效。将从输出符号表中删除隐藏引用。

另请参阅

- 第2-71 页的*SHOW*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.2.3 IMPORT

IMPORT 命令指定符号是运行时在共享对象中定义的。

语法

IMPORT *pattern* [AS *replacement_pattern*] [,*pattern* [AS *replacement_pattern*]] *

其中：

pattern 是一个与零个或多个未定义的全局符号相匹配的字符串，可以选择在其中包含通配符 （* 或 ?）。如果 *pattern* 与任何未定义的全局符号均不匹配，链接器将忽略该命令。此操作数只能与未定义的全局符号相匹配。

replacement_pattern

是一个要将符号重命名为的字符串，可以选择在其中包含通配符（* 或 ?）。此处的通配符必须在 *pattern* 中具有对应的通配符。*pattern* 通配符匹配的字符将替换 *replacement_pattern* 通配符。

例如：

```
IMPORT my_func AS func
```

导入未定义的符号 my_func，并将其重命名为 func。

用法

无法导入已在当前共享对象或可执行文件中定义的符号。IMPORT 中仅允许使用一个通配符 （* 或 ?）。

如果存在动态符号表，则将未定义的符号包括在动态符号表中 （如果指定，则作为 *replacement_pattern*；否则，作为 *pattern*）。

注意

IMPORT 命令仅影响未定义的全局符号。共享库已解析的符号将隐式地导入到动态符号表中。链接器忽略针对于隐式导入的符号的任何 IMPORT 指令。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.2.4 RENAME

RENAME 命令重命名已定义和未定义的全局符号名称。

语法

RENAME *pattern* AS *replacement_pattern* [,*pattern* AS *replacement_pattern*] *

其中：

pattern 是一个与零个或多个全局符号相匹配的字符串，可以选择在其中包含通配符 （* 或 ?）。如果 *pattern* 与任何全局符号均不匹配，链接器将忽略该命令。此操作数可以与已定义和未定义的符号相匹配。

replacement_pattern

是一个要将符号重命名为的字符串，可以选择在其中包含通配符（* 或 ?）。此处的通配符必须在 *pattern* 中具有对应的通配符。*pattern* 通配符匹配的字符将替换 *replacement_pattern* 通配符。

例如，对于名为 func1 的符号：

```
RENAME f* AS my_f*
```

将 func1 重命名为 my_func1。

用法

无法将符号重命名为已存在的符号名，即使正在重命名目标符号名本身。RENAME 中仅允许使用一个通配符 （* 或 ?）。例如，假定有一个包含符号 func1、 func2 和 func3 的映像：

```
EXPORT func1 AS func2    ;invalid, func2 exists
RENAME func3 AS b2
EXPORT func1 AS func3    ;invalid, func3 exists, even though it is renamed to
b2
```

链接器在执行任何替换之前先处理控制文件。因此，不能在第 1 行中使用 RENAME A AS B 并在第 2 行中使用 RENAME B AS A。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.2.5 REQUIRE

REQUIRE 命令在动态数组中创建 DT_NEEDED 标签。DT_NEEDED 标签指定与应用程序使用的其他共享对象之间的依赖性，例如共享库。

语法

REQUIRE *pattern* [,*pattern*] *

其中：

pattern 是表示文件名的字符串。不允许使用通配符。

用法

链接器将值为 *pattern* 的 DT_NEEDED 标签插入到动态数组中。这会指示动态装入程序，它当前所加载的文件需要加载 *pattern*。

注意

由于 REQUIRE 命令而插入的 DT_NEEDED 标签将添加到位于命令行中的共享对象或 DLL 生成的 DT_NEEDED 标签后面。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.2.6 RESOLVE

RESOLVE 命令将未定义的特定引用与已定义的全局符号相匹配。

语法

RESOLVE *pattern* AS *defined_pattern*

其中：

pattern 是一个必须与已定义的全局符号相匹配的字符串，可以选择在其中包含通配符。

defined_pattern

是一个与零个或多个已定义的全局符号相匹配的字符串，可以选择在其中包含通配符。如果 *defined_pattern* 与任何已定义的全局符号均不匹配，链接器将忽略该命令。无法将未定义的引用与未定义的符号相匹配。

用法

RESOLVE 是现有 armlink --unresolved 命令行选项的扩展。不同之处在于，--unresolved 允许所有未定义的引用与一个单一定义相匹配；而 RESOLVE 允许引用与符号之间进行更明确的匹配。

未定义的引用将从输出符号表中删除。

RESOLVE 适用于执行部分链接和正常链接的情况。

例如，您可能使用两个名为 file1.c 和 file2.c 的文件，如第2-70 页的示例 2-6 中所示。您可以创建一个 ed.txt 文件（包含一行 RESOLVE MP3* AS MyMP3*），然后执行以下命令：

```
armlink file1.o file2.o --edit ed.txt --unresolved foobar
```

此命令产生以下结果：

- file1.o 中的引用 （foo、 MP3_Init() 和 MP3_Play()）与 file2.o 中的定义（分别为 foobar、 MyMP3_Init() 和 MyMP3_Play()）相匹配，这是由控制文件 ed.txt 指定的。
- ed.txt 中的 RESOLVE 命令与 MP3 函数相匹配；而 --unresolved 选项与任何其他剩余的引用相匹配，在本示例中， foo 与 foobar 相匹配。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

- 输出符号表（不论是映像还是部分对象）不包含 foo、MP3_Init 或 MP3_Play 符号。

示例 2-6 使用 RESOLVE 命令

```
file1.c

extern int foo;
extern void MP3_Init(void);
extern void MP3_Play(void);

int main(void)
{
  int x = foo + 1;
  MP3_Init();
  MP3_Play();
  return x;
}


file2.c:

int foobar;
void MyMP3_Init()
{
}
void MyMP3_Play()
{
}
```


Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

2.2.7 SHOW

SHOW 命令使全局符号变为可见符号。如果要取消隐藏使用 HIDE 命令和通配符隐藏的特定符号，可使用此命令。

语法

SHOW *pattern* [,*pattern*] *

其中：

pattern 是一个与零个或多个全局符号相匹配的字符串，可以选择在其中包含通配符。如果 *pattern* 与任何全局符号均不匹配，链接器将忽略该命令。

用法

SHOW 的用法与 HIDE 的用法密切相关。

另请参阅

- 第2-65 页的*HIDE*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

第 3 章
分散加载描述文件的形式语法

本章介绍分散加载描述文件的格式，包括以下各节：

有关如何使用分散加载描述文件的详细信息，请参阅 《链接器用户指南》中的第 5 章 *使用分散加载描述文件*。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

3.1 BNF 表示法和语法

表 3-1 概括了用于描述形式语言的 *Backus-Naur 格式* (BNF) 符号。

表 3-1 BNF 语法

符号	说明
"	引号用于指示，通常作为 BNF 语法一部分的字符将用作定义中的文字字符。例如，只能将定义 B"+"C 替换为模式 B+C； 而可以将定义 B+C 替换为 BC、BBC 或 BBBC。
A ::= *B*	将 *A* 定义为 *B*。例如，A::= B"+" | C 表示 *A* 等效于 B+ 或 C。::= 表示法用于以组件形式定义更高级别的结构。每个组件也可能具有 ::= 定义，以便以甚至更简单的组件对其进行定义。例如，A::= B 和 B::= C | D 表示定义 *A* 等效于模式 C 或 D。
[*A*]	可选元素 *A*。例如，A::= B[C]D 表示定义 *A* 可以扩展为 BD 或 BCD。
A+	元素 *A* 可能出现一次或多次。例如，A::= B+ 表示定义 *A* 可以扩展为 B，BB 或 BBB。
*A**	元素 *A* 可能不出现或出现多次。
A | *B*	可能出现元素 *A* 或 *B*，但不能同时出现。
(*A B*)	元素 *A* 和 *B* 组合在一起。在使用 | 运算符时或重复复杂模式时，这种组合特别有用。例如，A::=(B C)+ (D | E) 表示定义 *A* 定义可以扩展为 BCD，BCE、BCBCD、BCBCE、BCBCBCD 或 BCBCBCE 中的任何一个。

注意

本节的 BNF 定义包含额外的换行符和空格以提高可读性。分散加载定义中不需要使用这些字符，如果文件中包含这些字符，则会将其忽略。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

3.2 分散加载描述文件语法概述

图 3-1 显示了一个典型的分散加载描述文件的组件和组织结构。

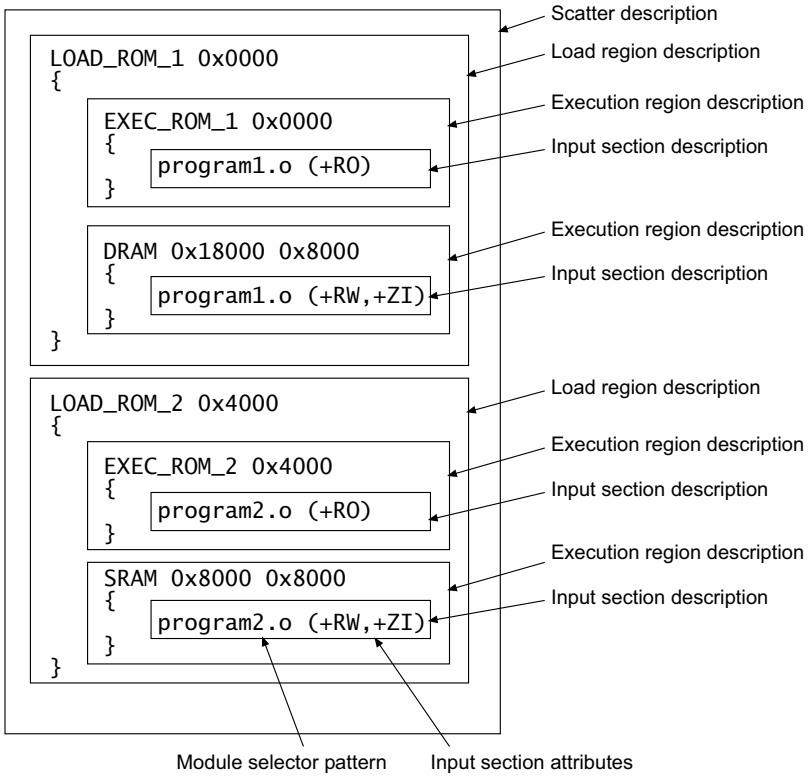

图 3-1 分散加载描述文件的组件

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

3.3 加载区描述

加载区具有：

- 名称 （链接器使用它来识别不同的加载区）
- 基址 （加载视图中的代码和数据的起始地址）
- 指定加载区特性的属性
- 指定的最大大小 （可选）
- 一个或多个执行区

图 3-2 显示了典型加载区描述的组件。

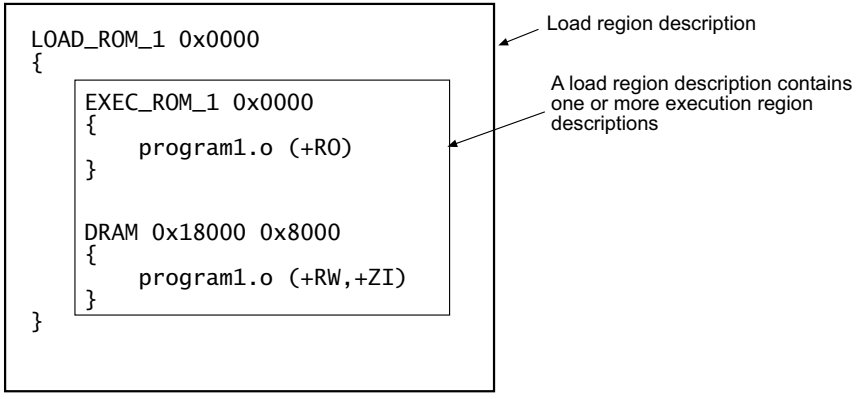

图 3-2 加载区描述的组件

3.3.1 加载区描述的语法

BNF 语法为：

```
load_region_description ::=
 load_region_name  (base_address  | ("+" offset)) [attribute_list] [max_size]
       "{"
          execution_region_description+
       "}"
```

其中：

load_region_name

命名加载区。

base_address 指定要在其中链接加载区中对象的地址。*base_address* 必须是字对齐的。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

+offset 描述一个基址，该基址超出前一个加载区末尾 *offset* 个字节。*offset* 的值必须能被 4 整除。如果这是第一个加载区，则 *+offset* 表示基址从零后面的 *offset* 个字节开始。

attribute_list

指定加载区内容的属性：

ABSOLUTE 绝对地址。区的加载地址由基址指示符指定。

ALIGN *alignment*. 将加载区的对齐约束从 4 增加到 *alignment*。*alignment* 必须为 2 的正数幂。如果加载区具有 *base_address*，则必须为 *alignment* 对齐。如果加载区具有 *+offset*，则链接器将计算得到的区基址与 *alignment* 边界对齐。

NOCOMPRESS 缺省情况下，将启用 RW 数据压缩。使用 NOCOMPRESS 关键字可以指定不能在最终映像中压缩加载区的内容。

OVERLAY 使用 OVERLAY 关键字可以使多个加载区位于同一个地址上。ARM 工具没有提供重叠机制。若要在同一地址使用多个加载区，必须提供自己的重叠区管理器。

PI 此区与位置无关。

RELOC 此区可重定位。

max_size 指定加载区的最大大小。这是在进行任何解压缩或零初始化之前的加载区大小。如果指定可选的 *max_size* 值，并且为区分配的字节数多于 *max_size*，则 armlink 会生成错误。

execution_region_description

指定执行区的名称、地址和内容。请参阅第3-6 页的*执行区描述*。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

3.4 执行区描述

执行区具有：

- 名称 （链接器使用它来识别不同的执行区）
- 基址 （绝对或相对）
- 指定执行区特性的属性
- 指定的最大大小 （可选）
- 一个或多个输入节描述 （放在此执行区中的模块）。

图 3-3 显示了典型执行区描述的组件。

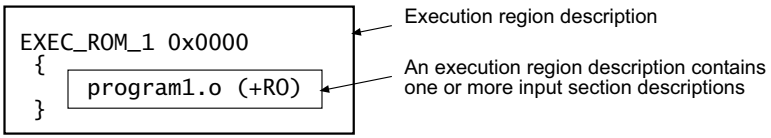

图 3-3 执行区描述的组件

3.4.1 执行区描述语法

执行区描述的语法 （BNF 语法）为：

```
execution_region_description ::=

  exec_region_name (base_address | "+" offset) [attribute_list] [max_size | length]
        "{"
            input_section_description*
        "}"
```

其中：

*exec_region_name*命名执行区。

base_address 指定要在其中链接执行区中对象的地址。*base_address* 必须是字对齐的。

+offset 描述一个基址，该基址超出前一个执行区末尾 *offset* 个字节。*offset* 的值必须能被 4 整除。

如果前面没有执行区 （即，这是加载区中的第一个执行区），则 *+offset* 表示基址从该执行区所在的加载区的基址后面 *offset* 个字节开始。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

attribute_list

它指定执行区内容的属性：

ABSOLUTE
绝对地址。区执行地址是由基址指示符指定的。

ALIGN *alignment*
将执行区的对齐约束从 4 增加到 *alignment*。*alignment* 必须为 2 的正数幂。如果执行区具有 *base_address*，则必须为 *alignment* 对齐。如果执行区具有 *+offset*，则链接器将计算得到的区基址与 *alignment* 边界对齐。

EMPTY
在执行区中保留一个给定长度的空白内存块，通常供堆或堆栈使用。不能将任何节放置在具有 EMPTY 属性的区中。

FILL
创建一个链接器生成的区，其中包含一个值。如果指定 FILL，则必须为其赋值，例如：FILL 0xFFFFFFFF。FILL 属性会替代以下组合：EMPTY ZEROPAD PADVALUE。

在某些情况下 （例如仿真），长时间的零循环是比较合适的。

FIXED
固定地址。链接器尝试通过插入填充使执行地址等于加载地址。这会使区成为根区。如果这无法完成，则链接器会生成错误。

NOCOMPRESS
缺省情况下，将启用 RW 数据压缩。使用 NOCOMPRESS 关键字可以指定不能在最终映像中压缩执行区中的 RW 数据。

OVERLAY
用于具有重叠的地址范围的节。将为具有 OVERLAY 属性且基址偏移为 +0 的连续执行区指定相同的基址。有关详细信息，请参阅 《链接器用户指南》中第5-22 页的*使用重叠区放置节*。

PADVALUE
定义任何填充的值。如果指定 PADVALUE，则必须为其赋值，例如：

```
EXEC 0x10000 PADVALUE 0xFFFFFFFF EMPTY ZEROPAD 0x2000
```

这会创建一个大小为 0x2000 且充满 0xFFFFFFFF 的区。

PADVALUE 必须为一个字大小。将忽略加载区中的 PADVALUE 属性。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

PI
: 此区仅包含与位置无关的节。

SORTTYPE
: 指定执行区的排序算法，例如：
ER1 +0 SORTTYPE=CallTree
有关详细信息，请参阅第2-53 页的*--sort=algorithm*。

UNINIT
: 用于创建包含未初始化的数据或内存映射的 I/O 的执行区。

ZEROPAD
: 零初始化的节作为零填充块写入 ELF 文件，因此，运行时无需使用零进行填充。
只有根执行区能够使用 ZEROPAD 属性进行零初始化。如果将 ZEROPAD 属性用于非根执行区，则会生成警告并忽略该属性。
在某些情况下 （例如仿真），长时间的零循环是比较合适的。

max_size
: 对于标记为 EMPTY 或 FILL 的执行区， *max_size* 值会解释为区的长度。在其他情况下， *max_size* 值会解释为执行区的最大大小。

[-]length
: 只能与 EMPTY 一起使用，以表示在内存中向下增长的堆栈。如果指定的长度为负值，则将 *base_address* 作为区结束地址。

input_section_description
: 指定输入节的内容。请参阅第3-12 页的*输入节描述*。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

3.5 寻址属性

加载和执行区属性中的一些属性可将有关区的内容以及区在链接后的行为方式的信息通知给链接器。这些属性有：

ABSOLUTE　内容放置在不会在链接后更改的固定地址处。

PI　内容不依赖于任何固定地址，可以在链接后不经过任何额外处理就进行移动。

RELOC　内容依赖于固定地址，会输出重定位信息以使内容可以由其他工具移动到其他位置。

OVERLAY　内容放置在不会在链接后更改的固定地址处。内容可能与具有 OVERLAY 的其他区重叠。

一般情况下，同一加载区中的所有执行区都具有相同的寻址属性。若要易于进行选择，寻址属性可以从前一个区继承，从而只需在一个地方设置这些属性。设置和继承寻址属性的规则有：

- 显式设置寻址属性：
  - 可以使用 ABSOLUTE、 PI、 RELOC 或 OVERLAY 属性显式设置加载区。
  - 可以使用 ABSOLUTE、 PI 或 OVERLAY 属性显式设置执行区。对于要使用 RELOC 属性进行设置的执行区，该区必须从父加载区继承。
- 在未指定任何寻址属性时，会隐式设置这些属性：
  - 不能继承 OVERLAY 属性。具有 OVERLAY 属性的区不能进行继承。
  - 基址加载区或执行区始终缺省设置为 ABSOLUTE
  - +offset 加载区从前一个加载区继承寻址属性，如果不存在前一个加载区，则为 ABSOLUTE。
  - A +offset 执行区从前一个执行区继承寻址属性，如果不存在前一个执行区，则从父加载区继承。

示例 3-1、第3-10 页的示例 3-2 和第3-11 页的示例 3-3 中演示了适用于设置寻址属性的继承规则。

示例 3-1 加载区继承

```
LR1 0x8000 PI
{
    ...
}
```


Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

```
LR2 +0              ; LR2 inherits PI from LR1
{
    ...
}
LR3 0x1000          ; LR3 does not inherit because it has no relative base,
                      gets default of ABSOLUTE
{
    ...
}
LR4 +0              ; LR4 inherits ABSOLUTE from LR3
{
    ...
}
LR5 +0 RELOC        ; LR5 does not inherit because it explicitly sets RELOC
{
    ...
}
LR6 +0 OVERLAY      ; LR6 does not inherit, an OVERLAY cannot inherit
{
    ...
}
LR7 +0              ; LR7 cannot inherit OVERLAY, gets default of ABSOLUTE
{
    ...
}
```

示例 3-2 执行区继承

```
LR1 0x8000 PI
{
    ER1 +0          ; ER1 inherits PI from LR1
    {
        ...
    }
    ER2 +0          ; ER2 inherits PI from ER1
    {
        ...
    }
    ER3 0x10000     ; ER3 does not inherit, because it has no relative base,
                       gets default of ABSOLUTE
    {
        ...
    }
    ER4 +0          ; ER4 inherits ABSOLUTE from ER3
    {
        ...
    }
```


Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

```
    ER5 +0 PI      ; ER5 does not inherit, it explicitly sets PI
    {
        ...
    }
    ER6 +0 OVERLAY ; ER6 does not inherit, an OVERLAY cannot inherit
    {
        ...
    }
    ER7 +0         ; ER7 cannot inherit OVERLAY, gets default of ABSOLUTE
    {
        ...
    }
}
```

示例 3-3 继承 RELOC

```
LR1 0x8000 RELOC
{
    ER1 +0 ; inherits RELOC from LR1
    {
        ...
    }
    ER2 +0 ; inherits RELOC from ER1
    {
        ...
    }
    ER3 +0 RELOC ; Error cannot explicitly set RELOC on an execution region
    {
        ...
    }
}
```


Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

3.6 输入节描述

输入节描述是一种按以下内容标识输入节的模式：

- 模块名称 （对象文件名、库成员名或库文件名）。模块名称可以使用通配符。
- 输入节名称或输入节属性，如 READ-ONLY 或 CODE。
- 符号名。

图 3-4 显示了典型输入节描述的组件。

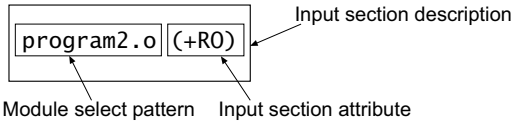

图 3-4 输入节描述的组件

3.6.1 输入节描述的语法

输入节描述的语法 （BNF 语法）为：

```
input_section_description ::=

  module_select_pattern
        [ "(" input_section_selector ( "," input_section_selector )* ")" ]

input_section_selector ::=

        ("+" input_section_attr | input_section_pattern | input_symbol_pattern)
```

其中：

module_select_pattern

一个由文字文本构成的模式。通配符 * 与零个或零个以上的字符相匹配；? 与任何单个字符相匹配。

匹配不区分大小写，即使是在区分文件命名大小写的主机上。

可以使用 *.o 匹配所有对象； 而使用 * 匹配所有对象文件和库。

当 *module_select_pattern* 与以下内容之一相匹配时，则表示输入节与模块选择器模式相匹配：

- 包含节的对象文件的名称。
- 库成员名称 （不带前导路径名）。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

- 从中提取节的库的完整名称 （包括路径名）。如果名称包含空格，则可以使用通配符简化搜索。例如，使用 *libname.lib 匹配 C:\lib dir\libname.lib。

通过使用特殊模块选择器模式 .ANY，您可以将输入节分配给执行区，而无需考虑其父模块。可以使用 .ANY 以*任意*分配方式填充执行区。

使用 .ANY*num* 可指定特定的优先级顺序，其中 *num* 是一个大于零的正整数后缀。使用最大整数指定的优先级最高。

示例 3-4 带有整数后缀的 .ANY

```
lr1 0x8000 1024
{
    er1 +0 512
    {
        .ANY1(+RO) ; evenly distributed with er3
    }
    er2 +0 256
    {
        .ANY2(+RO) ; Highest priority, so filled first
    }
    er3 +0 256
    {
        .ANY1(+RO) ; evenly distributed with er1
    }
}
```

注意

- 只有同时与 *module_select_pattern* 及至少一个 *input_section_attr* 或 *input_section_pattern* 相匹配的输入节才包含在执行区中。

  如果省略 (+ *input_section_attr)* 和 *(input_section_pattern)*，则缺省为 +RO。
- 不要依赖编译器生成的输入节名称或由 ARM 库代码使用的输入节名称。这些名称可能会因不同的编译而异，例如，使用不同编译器选项时。另外，不能保证编译器使用的节命名约定在不同版本之间保持不变。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

input_section_attr

属性选择器与输入节属性相匹配。每个 *input_section_attr* 的前面都有一个 +。

如果指定一个模式以匹配输入节名称，名称前面必须有一个 + 号。可以省略紧靠 + 号前面的任何逗号。

选择器不区分大小写。可以识别以下选择器：

- RO-CODE
- RO-DATA
- RO，同时选择 RO-CODE 和 RO-DATA
- RW-DATA
- RW-CODE
- RW，同时选择 RW-CODE 和 RW-DATA
- ZI
- ENTRY，即，包含 ENTRY 点的节。

可以识别以下同义词：

- CODE 表示 RO-CODE
- CONST 表示 RO-DATA
- TEXT 表示 RO
- DATA 表示 RW
- BSS 表示 ZI。

可以识别以下伪属性：

- FIRST
- LAST

以下属性选择器模式描述了节在执行区中的放置顺序：

第一个和最后一个节

如果节位置顺序很重要 （例如，如果特定输入节必须是区中的第一个输入节，而包含校验和的输入节必须是最后一个输入节），则可以使用 FIRST 和 LAST 标记执行区中的第一个节和最后一个节。

input_section_selector 列表中只能有一个 FIRST 或 LAST 属性，且该属性必须在单个 *input_section_attr* 的后面。例如：

```
*(section, +FIRST)
```

此模式是正确的。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

*(+FIRST, section)

此模式不正确，会生成错误消息。

特殊模块选择器

通过使用特殊模块选择器模式 .ANY，您可以将输入节分配给执行区，而无需考虑其父模块。可以使用一个或多个 .ANY 模式以*任意*分配方式填充执行区。在大多数情况下，使用单个 .ANY 等效于使用 * 模块选择器。

修改的选择器

一个分散加载文件中不能包含两个 * 选择器。但是，可以使用两个修改的选择器（如 *A 和 *B），也可将 .ANY 选择器与 * 模块选择器配合使用。* 模块选择器的优先级比 .ANY 高。如果删除了文件中包含 * 选择器的部分，.ANY 选择器将变为活动状态。

未分配的节

在解析所有其他（非 .ANY）输入节描述后，才会解析具有 .ANY 模块选择器模式的输入节描述。所有未分配给执行区的节将分配给 .ANY 区。

如果存在多个 .ANY 模式，链接器将使用未分配给执行区的最大节，并将该节分配给具有足够可用空间的最明确的 .ANY 执行区。当 armlink 进行此项选择时，它将 .ANY(.text) 视为比 .ANY(+RO) 更明确。

如果几个执行区具有同等的明确性，则将该节分配给可用剩余空间最大的执行区。

例如：

- 如果两个执行区具有同等的明确性，其中一个执行区的大小限制为 0x2000，另一个执行区没有限制，则将所有节分配给第二个没有限制的 .ANY 区。
- 如果两个执行区具有同等的明确性，其中一个执行区的大小限制为 0x2000，另一个执行区的大小限制为 0x3000，则将要放置的第一批节分配给第二个大小限制为 0x3000 的 .ANY 区，直至第二个 .ANY 的剩余大小减少到 0x2000。此后，将在两个 .ANY 执行区之间交替分配节。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

input_section_pattern

一个与输入节名称相匹配的模式，不区分大小写。它由文字文本构成。通配符 * 与零个或零个以上的字符相匹配，而 ? 与任何单个字符相匹配。

注意

如果使用多个 *input_section_pattern*，请确保不同执行区中没有重复的模式，以避免出现歧义错误。

input_symbol_pattern

可以按输入节定义的全局符号名称来选择该节。这样，即可从部分链接的对象中选择具有相同名称的各个节。

:gdef: 前缀可以区分全局符号模式和节模式。例如，可以使用 :gdef:mysym 选择定义 mysym 的节。以下示例说明了一个描述文件，其中的 ExecReg1 包含一个定义全局符号 mysym1 的节以及一个包含全局符号 mysym2 的节：

```
LoadRegion 0x8000
{
    ExecReg1 +0
    {
        *(:gdef:mysym1)
        *(:gdef:mysym2)
    }
                        ; rest of scatter description
}
```

注意

如果使用多个 *input_symbol_pattern*，请确保不同执行区中没有重复的模式，以避免出现歧义错误。

输入节描述符的顺序并不重要。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

3.7 解析多个匹配

如果某个节与多个执行区相匹配，则按下述方式解析匹配。但是，如果找不到唯一匹配，则链接器将分散加载描述作为错误处理。每个节都是通过 *module_select_pattern* 和 *input_section_selector* 选择的。

module_select_pattern 规范的示例有：

- `*` 与任何模块或库相匹配
- `*.o` 与任何对象模块相匹配
- `math.o` 与 `math.o` 模块相匹配
- `*armlib*` 与 ARM 提供的所有 C 库相匹配
- `*math.lib` 与以 `math.lib` 结尾的任何库路径相匹配。例如，`C:\apps\lib\math\satmath.lib`。

input_section_selector 规范的示例有：

- `+RO` 是与所有 RO 代码和所有 RO 数据相匹配的输入节属性
- `+RW,+ZI` 是与所有 RW 代码、所有 RW 数据和所有 ZI 数据相匹配的输入节属性
- `BLOCK_42` 是与名为 `BLOCK_42` 的汇编文件区域相匹配的输入节模式。

注意

编译器生成可由输入节模式标识的区域，例如 `.text`、`.data`、`.constdata` 和 `.bss`。但是，这些名称将来可能会发生改变，您必须避免使用这些名称。

如果要匹配 C 或 C++ 文件中的特定函数或 **extern** 数据，请执行以下任一操作：

— 在单独模块中编译该函数或数据，然后匹配模块对象名

— 使用 `#pragma arm section` 或 `__attribute__` 指定包含所需代码或数据的节的名称。有关编译指示的详细信息，请参阅 《编译器参考指南》中第4-50 页的*编译指示*。

以下变量用于描述多个匹配：

- *m1* 和 *m2* 表示模块选择器模式
- *s1* 和 *s2* 表示输入节选择器。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

如果出现多个匹配，则链接器根据最明确的 *module_select_pattern* 和 *input_section_selector* 对来确定将输入节分配到的区。

例如，如果输入节 A 匹配执行区 R1 的 *m1,s1*，并且 A 还匹配执行区 R2 的 *m2,s2*，则链接器：

- 将 A 分配给 R1 （如果 *m1,s1* 比 *m2,s2* 更明确）
- 将 A 分配给 R2 （如果 *m2,s2* 比 *m1,s1* 更明确）
- 诊断分散加载描述出错 （如果 *m1,s1* 不比 *m2,s2* 更明确， *m2,s2* 也不比 *m1,s1* 更明确）。

armlink 按以下顺序确定最明确的 *module_select_pattern* 和 *input_section_selector* 对：

1. 对于模块选择器模式：

   如果文本字符串 *m1* 与模式 *m2* 相匹配，而文本字符串 *m2* 与模式 *m1* 不匹配，则 *m1* 比 *m2* 更明确。

2. 对于输入节选择器：
   - 如果 *s1* 和 *s2* 都是与节名称相匹配的模式，则使用与模块选择器模式相同的定义。
   - 如果 *s1* 和 *s2* 中的一个与输入节名称相匹配，另一个与输入节属性相匹配，则 *s1* 和 *s2* 的顺序出现混乱并将描述诊断为出错。
   - 如果 *s1* 和 *s2* 都与输入节属性相匹配， *s1* 是否比 *s2* 更明确是由以下关系定义的：
     — ENTRY 比 RO-CODE、 RO-DATA、 RW-CODE 或 RW-DATA 更明确
     — RO-CODE 比 RO 更明确
     — RO-DATA 比 RO 更明确
     — RW-CODE 比 RW 更明确
     — RW-DATA 比 RW 更明确
     — 节属性之间没有其他的关系 （*s1* 比 *s2* 更明确）成员。

3. 对于 *module_select_pattern, input_section_selector* 对，仅当任何以下条件成立时， *m1,s1* 才比 *m2,s2* 更明确：

   a. *s1* 是文字输入节名称 （即，它不包含模式字符），并且 *s2* 与 +ENTRY 之外的输入节属性相匹配

   b. *m1* 比 *m2* 更明确

   c. *s1* 比 *s2* 更明确。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

测试这些条件的顺序为条件 a 优先于条件 b 和 c，而条件 b 优先于条件 c。

这种匹配策略产生以下结果：

- 描述不依赖于它们在文件中的写入顺序。
- 一般来说，对象的描述越明确，它包含的输入节的描述也就越明确。
- 不检查 *input_section_selector*，除非：
  - 对象选择不确定。
  - 一个选择器完全命名输入节，另一个选择器按属性进行选择。在这种情况下，显式地输入节名称比 ENTRY （可精确地从一个对象中选择一个输入节）之外的任何属性更明确。即使与输入节名称关联的对象选择器不如属性的对象选择器明确，上述结论也成立。

示例 3-5 演示了多个执行区和模式匹配。

示例 3-5 多个执行区和模式匹配

```
LR_1 0x040000
{
    ER_ROM 0x040000              ; The startup exec region address is the same
    {                            ; as the load address.
        application.o (+ENTRY)   ; The section containing the entry point from
    }                            ; the object is placed here.
    ER_RAM1 0x048000
    {
        application.o (+RO-CODE) ; Other RO code from the object goes here
    }
    ER_RAM2 0x050000
    {
        application.o (+RO-DATA) ; The RO data goes here
    }
    ER_RAM3 0x060000
    {
        application.o (+RW)      ; RW code and data go here
    }
    ER_RAM4 +0                   ; Follows on from end of ER_R3
    {
        *.o (+RO, +RW, +ZI)      ; Everything except for application.o goes here
    }
}
```


Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

3.8 解析路径名

链接器将分散加载文件中的通配符模式与它在路径名中找到的任何正斜杠和反斜杠组合进行匹配。如果路径是从环境变量或多个来源中提取的，或者要使用相同的分散加载文件在 Windows 或 Unix 平台上进行构建，这可能是非常有用的。

注意

ARM 建议在路径名中使用正斜杠，以确保在 Windows 和 Unix 平台上能够识别它们。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

3.9 分散加载文件中的表达式求值

分散加载文件通常包含指定为表达式的数值常数。链接器还提供一个名为 ScatterAssert 的断言函数，该函数将表达式作为参数。如果此表达式的计算结果不是 true，则会生成一条错误消息。有关详细信息，请参阅第3-22 页的 *ScatterAssert 函数*。

3.9.1 表达式用法

可以将表达式用于以下位置：

- 加载和执行区 *base_address*
- 加载和执行区 *+offset*
- 加载和执行区 *max_size*
- ALIGN、 FILL 或 PADVALUE 关键字的参数
- ScatterAssert 函数的参数

3.9.2 表达式规则

表达式遵循 C 优先级规则并由以下内容组成：

- 十进制或十六进制数字。
- 算术运算符：+、 -、 /、 *、 ~、 OR 和 AND

  OR 和 AND 运算符分别映射到 C 运算符 | 和 &。
- 逻辑运算符：LOR、 LAND 和 !

  LOR 和 LAND 运算符分别映射到 C 运算符 || 和 &&。
- 关系运算符：<、 <=、 >、 >= 和 ==

  如果表达式的计算结果为 false，则返回零；如果计算结果为 true，则返回非零值。
- 条件运算符：*Expression* ? *Expression1* : *Expression2*

  它与 C 条件运算符相匹配。如果 *Expression* 的计算结果不为零，则对 *Expression1* 进行求值，否则对 *Expression2* 进行求值。
- 返回数字的函数。有关详细信息，请参阅第3-22 页的*内置函数*。

所有运算符在含义和优先级方面与对应的 C 运算符完全一致。

表达式不区分大小写，为了清晰起见，表达式可以使用括号。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

3.9.3 内置函数

仅当指定了 *base_address* 或 *+offset* 值时，才能使用与执行地址有关的函数。它们映射到表 3-2 中所示的链接器定义的符号组合。

表 3-2 与执行地址有关的函数

函数	链接器定义的符号值
ImageBase(*region_name*)	Image$$*region_name*$$Base
ImageLength(*region_name*)	Image$$*region_name*$$Length + Image$$region_name$$ZI$$Length
ImageLimit(*region_name*)	Image$$region_name$$Base + Image$$region_name$$Length + Image$$region_name$$ZI$$Length

有关与区有关的链接器符号的详细信息，请参阅 《链接器用户指南》中第4-3 页的表 4-1。

region_name 参数可以是加载区或执行区名称。不允许进行正向引用。*region_name* 只能引用已定义的加载区或执行区。请参阅第3-23 页的示例 3-7。

3.9.4 ScatterAssert 函数

可以在顶层或加载区内使用 ScatterAssert(*expression*) 函数。在链接完成后，会对其进行求值，如果 *expression* 的计算结果为 false，则会生成错误消息。第3-25 页的示例 3-10 演示了如何使用 ScatterAssert 函数编写比区的 *max_size* 限制更复杂的大小检查。

与加载地址有关的函数只能在 ScatterAssert 函数内部使用。它们映射到表 3-3 中所示的三个链接器定义的符号值。

表 3-3 与加载地址有关的函数

函数	链接器定义的符号值
LoadBase(*region_name*)	Load$$*region_name*$$Base
LoadLength(*region_name*)	Load$$*region_name*$$Length
LoadLimit(*region_name*)	Load$$*region_name*$$Limit

region_name 参数可以是加载区或执行区名称。不允许进行正向引用。*region_name* 只能引用已定义的加载区或执行区。请参阅第3-23 页的示例 3-7。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

3.9.5 与符号有关的函数

如果未定义 *global_symbol_name*，则与符号有关的函数 defined(*global_symbol_name*) 返回零；否则，返回非零值。

3.9.6 示例

示例 3-6 根据表达式指定最大大小

```
LR1 0x8000 (2 * 1024)
{
    ER1 +0 (1 * 1024)
    {
        *(+RO)
    }
    ER2 +0 (1 * 1024)
    {
        *(+RW +ZI)
    }
}
```

示例 3-7 将一个执行区放在另一个执行区后面

```
LR1 0x8000
{
    ER1 0x100000
    {
        *(+RO)
    }
}
LR2 0x100000
{
    ER2 (ImageLimit(ER1))               ; Place ER2 after ER1 has finished
    {
        *(+RW +ZI)
    }
}
```


Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

示例 3-8 根据符号是否存在有条件地设置基址

```
LR1 0x8000
{
    ER1 (defined(version1) ? 0x8000 : 0x10000)   ; Base address is 0x8000
                                                 ; if version1 is defined
                                                 ; 0x10000 if not
    {
        *(+RO)
    }
    ER2 +0
    {
        *(+RW +ZI)
    }
}
```

示例 3-9 中使用预处理器宏和表达式组合，将紧密压缩的执行区复制到页面边界上的执行地址。通过使用 ALIGN 分散加载关键字，可以将 ER2 和 ER3 的加载地址以及执行地址对齐。

示例 3-9 在执行空间中对齐基址，但在加载空间中仍紧密压缩

```
#! armcc -E
#DEFINE START_ADDRESS  0x100000
#DEFINE PAGE_ALIGNMENT 0x100000

#DEFINE MY_ALIGN(address, alignment) ((address +
(alignment-1)) AND ~(alignment-1))

LR1 0x8000
{
    ER0 +0
    {
        *(InRoot$$Sections)
    }
    ER1 START_ADDRESS
    {
        file1.o(*)
    }
    ER2 MY_ALIGN(ImageLimit(ER1), PAGE_ALIGNMENT)
    {
        file2.o(*)
    }
    ER3 MY_ALIGN(ImageLimit(ER2), PAGE_ALIGNMENT)
    {
```


Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

```
        file3.o(*)
    }
}
```

示例 3-10 使用 ScatterAssert 检查多个区的大小

```
LR1 0x8000
{
    ER0 +0
    {
        *(+RO)
    }
    ER1 +0
    {
        file1.o(+RW)
    }
    ER2 +0
    {
        file2.o(+RW)
    }
    ScatterAssert((LoadLength(ER1) + LoadLength(ER2)) < 0x1000)
                                          ; LoadLength is compressed size
    ScatterAssert((ImageLength(ER1) + ImageLength(ER2)) < 0x2000)
                                          ; ImageLength is uncompressed size
}
ScatterAssert(ImageLength(LR1) < 0x3000) ; Check uncompressed size of
LoadRegion
```


Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

第 4 章
BPABI 和 SysV 共享库和可执行文件

本章介绍 ARM® 链接器 armlink 如何支持 BPABI 和 SysV 共享库和可执行文件。本章分为以下几节：

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

4.1 关于 BPABI

许多嵌入式系统都使用操作系统来管理设备上的资源。在许多情况下，这种操作系统就是包含与应用程序紧密集成的*实时操作系统*(RTOS) 的一个大型可执行文件。其他更复杂的*操作系统*(OS) 称为平台操作系统，例如 ARM Linux。这些操作系统能够根据需要加载应用程序和共享库。

要在平台操作系统上运行应用程序或使用共享库，必须遵循该平台的 ABI 以及 ARM 体系结构的 ABI。这可能会涉及对链接器输出 （例如，自定义文件格式）的大量更改。为支持各种各样的平台， ARM 体系结构的 ABI 提供了 BPABI。

BPABI 提供了可从中派生平台 ABI 的基本标准。链接器会生成一个符合 BPABI 的 ELF 映像或共享库作为输出。一种称为后链接器的平台特定工具会将此 ELF 输出文件转换为可由平台操作系统加载的文件。后链接器工具由平台操作系统供应商提供。图 4-1 显示了 BPABI 工具流。

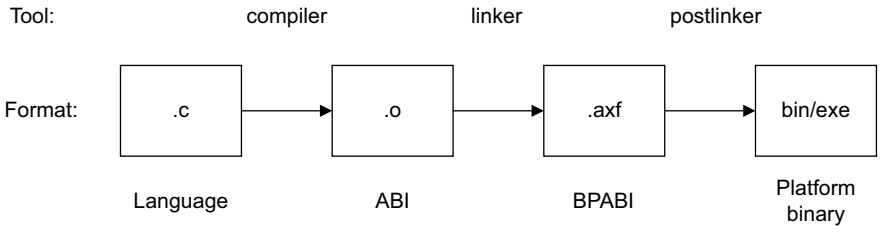

图 4-1 BPABI 工具流

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

4.2 BPABI 支持的平台

BPABI 根据共享库的类型定义了三种平台模型：

裸机 裸机模型设计用于脱机动态装入程序或简单的模块装入程序。各模块之间的引用由装入程序直接解析，而无需任何其他支持结构。

类似于 DLL 类似于 DLL 的模型通过牺牲动态库与静态库之间的透明性来换取加载时和运行时效率的提高。

SysV SysV 模型模糊了动态库与静态库之间的差异。ARM Linux 使用此格式。

4.2.1 BPABI 的链接器支持

ARM 链接器支持所有这三种 BPABI 模型，使您可以：

- 将对象和库集合链接到：
  - 裸机可执行映像
  - BPABI DLL 或 SysV 共享对象
  - BPABI 或 SysV 可执行文件

4.2.2 ARM Linux 的链接器支持

链接器生成的 SysV 可执行文件和共享库与 ARM Linux 兼容，因此不需要单独执行后链接步骤。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

4.3 所有 BPABI 模型共有的概念

通过使用链接器，您可以构建 BPABI 共享库并将对象链接到共享库上。本节介绍符号的导入、导出、版本控制和可见性。这些是所有 BPABI 模型所共有的概念。

4.3.1 导入和导出符号

在传统链接中，必须在链接时定义所有符号，链接器才能为其提供地址。在支持动态链接的平台中，可将符号绑定延迟到加载时，在某些情况下甚至可以延迟到运行时。链接器将这些符号放在动态符号表中。

可以使用两种方法控制动态符号表的内容：

- 可从 ELF 符号可见性属性推断内容的自动规则
- 位于控制文件中的手动指令

对于 SysV 模型，这些规则会略有不同。有关详细信息，请参阅第4-6 页的*自动动态符号表规则*。

4.3.2 符号可见性

每个符号都有一个可见性属性。如果符号是一个引用，则可见性将控制链接器可用于定义该符号的定义。如果符号是一个定义，则可见性将控制能否将符号设置为在当前模块外部可见。

有四个可见性选项：

表 4-1 符号可见性

符号	参考	定义
STV_DEFAULT	符号可绑定到共享对象中的定义。	符号可设置为在模块外部可见。符号可由动态链接器通过其他模块中的定义预先占用。
STV_PROTECTED	符号必须在模块内进行解析。	符号可设置为在模块外部可见。符号不能在运行时通过其他模块中的定义预先占用。
STV_HIDDEN STV_INTERNAL	符号必须在模块内进行解析。	符号在模块外部不可见。

符号预占在 SysV 系统中最常见。请参阅 SysV ELF 规范的动态链接部分。

在 BPABI 的类似于 DLL 的实现中可能发生符号预占。平台所有者定义其工作方式。有关详细信息，请参阅特定平台的文档。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

有关为符号设置符号可见性的详细信息，请参阅：

- 《汇编器指南》中第7-71 页的*EXPORT 或 GLOBAL*
- 《编译器参考指南》中第2-45 页的*--[no_]dllexport_all*
- 《编译器参考指南》中第2-46 页的*--[no_]dllimport_runtime*
- 《编译器参考指南》中第2-63 页的*--[no_]hide_all*

4.3.3 自动导入和导出

链接器可自动导入和导出符号。如果输出是可执行文件或共享库，并且平台模型为 SysV，则此行为取决于符号可见性的组合。这因所用链接模型的类型而异。有关详细信息，请参阅第4-6 页的*使用 SysV 模型*和第4-9 页的*使用裸机和类似于 DLL 的模型*。

4.3.4 手动导入和导出

可使用链接器控制文件手动控制动态导入和导出。有三个命令可用于控制动态符号表内容。有关详细信息，请参阅：

- 第2-64 页的*EXPORT*
- 第2-66 页的*IMPORT*
- 第2-68 页的*REQUIRE*

4.3.5 符号版本控制

符号版本控制提供了一种严密控制共享库接口的方法。

在从包含区分版本的符号的共享库中导入符号时，链接器会绑定到该符号的最新 （缺省）版本。当平台操作系统在加载时或运行时解析符号版本时，该版本将始终解析为链接器所选的版本，即使存在更新的版本也如此。此过程是自动执行的。

从可执行文件或共享库导出符号时，可为该符号指定版本。链接器支持隐式符号版本控制，即从共享对象名称 （由 --soname 设置）派生版本；也支持显式符号版本控制，即使用脚本明确定义版本。有关详细信息，请参阅 《链接器用户指南》中第4-15 页的*符号版本控制*。

4.3.6 RW 压缩

RW 压缩数据的解压缩器紧密集成在 ARM C 库中的启动代码中。在平台操作系统上运行应用程序时，必须由该平台或平台库提供此功能。缺省情况下，由于没有解压缩器，在链接 BPABI 或 SysV 文件时会关闭 RW 压缩。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

4.4 使用 SysV 模型

链接器可用于生成和链接 SysV 共享库。此外，它还可以创建 SysV 可执行文件。本节介绍动态符号表规则、寻址模式和线程局部存储模型。

表 4-2 启用 SysV 支持

命令行选项	
armlink --sysv	生成 SysV 可执行文件
armlink --sysv --shared	生成 SysV 共享库

4.4.1 内存模型

SysV 文件有一个如通用 ELF 规范所述的标准内存模型。有多种平台操作系统（例如， ARM Linux）使用 SysV 格式。

没有可用于 SysV 内存模型的配置选项。链接器忽略在命令行上指定的所有分散文件，使用通用 ELF 规范所定义的标准内存映射。

4.4.2 自动动态符号表规则

以下规则适用：

可执行文件 未定义的符号引用是未定义的符号错误。

具有 STV_HIDDEN 或 STV_INTERNAL 可见性的全局符号从不导出到动态符号表。

具有 STV_PROTECTED 或 STV_DEFAULT 可见性的全局符号不导出到动态符号表，除非设置了 --export_all。

共享库 具有 STV_DEFAULT 可见性的未定义符号引用被视为已导入并放入动态符号表中。

没有 STV_DEFAULT 可见性的未定义符号引用是未定义的符号错误。

具有 STV_HIDDEN 或 STV_INTERNAL 可见性的全局符号从不导出到动态符号表。

注意

重新分配所需的 STV_HIDDEN 或 STV_INTERNAL 全局符号可放在动态符号表中，但链接器会将其更改为局部符号，以防止从共享库外部访问它们。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

具有 STV_PROTECTED 或 STV_DEFAULT 可见性的全局符号始终导出到动态符号表。

符号定义

为提高 SysV 与 glibc 的兼容性，链接器会定义下列符号 （如果引用这些符号）：

- __init_array_start
- __init_array_end
- __fini_array_start
- __fini_array_end
- __exidx_start
- __exidx_end

4.4.3 寻址模式

SysV 有一种已定义模型，用于访问程序及导入的数据和代码。链接器自动生成所需的*过程链接表* (PLT) 和*全局偏移表* (GOT) 节。

位置无关代码

SysV 共享库几乎总是通过使用 --apcs=/fpic 编译器命令行选项与位置无关代码一起编译。有关详细信息，请参阅 《编译器参考指南》中第2-4 页的 *--apcs=qualifer...qualifier*。

必须同时使用链接器命令行选项 --fpic 声明共享库是与位置无关的，因为这会影响 PLT 和 GOT 节的构建。

注意

缺省情况下，如果指定 --shared 命令行选项而不带 --fpic 选项，链接器会生成一条错误消息。如果必须创建一个不是位置无关的共享库，则可以使用 --diag_suppress=6403 关闭该错误消息。

4.4.4 线程局部存储

链接器支持 ARM Linux 线程局部存储模型。有关链接器实现的详细信息，请参阅 《ARM 体系结构的 ABI 勘误表和附录》（ABI 附录）。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

4.4.5 相关链接器命令行选项

- 第2-24 页的*--fpic*
- 第2-17 页的*--dynamic_debug*
- 第2-17 页的*--dynamiclinker=name*
- 第2-20 页的*--[no_]export_all*
- 第2-35 页的*--linux_abitag=version_id*
- 第2-49 页的*--runpath=pathlist*
- 第2-52 页的*--shared*
- 第2-57 页的*--sysv*

更改为命令行缺省设置

ARM RealView® 编译工具的工具链不提供包含 C 和 C++ 系统库的共享库。SysV 支持的专用模型将使用平台附带的系统库。例如，在 ARM Linux 中，该库为 `libc.so`。

为了使用 `libc.so`，链接器会对缺省行为应用下列更改：

- 设置 `--no_ref_cpp_init` 以防止包含 RealView 编译工具 C++ 初始化代码
- 链接器定义所需符号以确保与 `libc.so` 兼容
- 设置 `--force_so_throw` 以强制链接器保留异常表。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

4.5 使用裸机和类似于 DLL 的模型

本节介绍链接器对 BPABI 规范的支持。如果要为特定的平台操作系统开发应用程序或 DLL，必须将下列信息与平台文档结合使用。如果要实现平台操作系统，必须将下列信息与 BPABI 规范结合使用。

表 4-3 启用 BPABI 支持

命令行选项	
armlink --bpabi	生成 BPABI 可执行文件
armlink --bpabi --dll	生成 BPABI DLL

4.5.1 内存模型

BPABI 文件有一个如通用 BPABI 规范所述的标准内存模型。通过使用 --bpabi 命令行选项，链接器会自动应用此模型并忽略在命令行上指定的任何分散加载描述文件。这相当于以下映像布局：

```
LR_1 <read-only base address>
{
    ER_RO  +0
    {
        *(+RO)
    }
}
LR_2 <read-write base address>
{
    ER_RW  +0
    {
        *(+RW)
    }
    ER_ZI  +0
    {
        *(+ZI)
    }
}
```

自定义内存模型

如果指定了 --ropi 选项，则会将 LR_1 标记为与位置无关。同理，如果指定了 --rwpi 选项，则会将 LR_2 标记为与位置无关。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

注意

在大多数情况下，需要指定 --ro_base 和 --rw_base 开关，因为其相应的缺省值 0x8000 和 0 可能并不适用于您的平台。这些地址并不需要反映在运行时将映像重定位到的实际地址。

有关详细信息，请参阅：

- 第2-48 页的*--ropi*
- 第2-48 页的*--ro_base=address*
- 第2-49 页的*--rosplit*
- 第2-50 页的*--rwpi*
- 第2-49 页的*--rw_base=address*

4.5.2 强制性的符号版本控制

类似于 DLL 的 BPABI 模型需要静态绑定。这是因为后链接器可能会将 BPABI DLL 中的符号信息转换为按序号编制索引的导入或导出表。在这种情况下，不能在运行时搜索符号。

在 BPABI 中会使用符号版本控制强制实施静态绑定。对于 BPABI 文件，命令行选项 --symver_soname 在缺省情况下处于开启状态，这意味着将根据 DLL 的名称为所有导出的符号指定版本。

有关详细信息，请参阅：

- 第2-53 页的*--soname=name*
- 第2-57 页的*--symver_script=file*
- 第2-57 页的*--symver_soname*
- 《链接器用户指南》中第4-15 页的*符号版本控制*

4.5.3 自动动态符号表规则

本节介绍可执行文件和 DLL 的符号表规则。

可执行文件

未定义的符号引用是未定义的符号错误。

具有 STV_HIDDEN 或 STV_INTERNAL 可见性的全局符号从不导出到动态符号表。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

具有 STV_PROTECTED 或 STV_DEFAULT 可见性的全局符号不导出到动态符号表，除非设置了 --export_all。有关详细信息，请参阅第2-20 页的*--[no_]export_all*。

DLL

未定义的符号引用是未定义的符号错误。

具有 STV_HIDDEN 或 STV_INTERNAL 可见性的全局符号从不导出到动态符号表。

注意

重新分配所需的 STV_HIDDEN 或 STV_INTERNAL 全局符号可放在动态符号表中，但链接器会将其更改为局部符号，以防止从共享库外部访问它们。

具有 STV_PROTECTED 或 STV_DEFAULT 可见性的全局符号始终导出到动态符号表。

4.5.4 寻址模式

裸机与类似于 DLL 的 BPABI 模型之间的主要区别在于用于访问已导入和自有程序代码和数据的寻址方式。提供了与 BPABI 规范中的类别相对应的四个选项：

- 无
- 直接引用
- 间接引用
- 相对静态基址引用

所需寻址模式的选择将由下列命令行选项控制：

- 第2-41 页的*--pltgot=type*
- 第2-41 页的*--pltgot_opts=mode*

4.5.5 C++ 初始化

DLL 支持通过初始值设定项表来初始化静态构造函数，这些初始值设定项被赋予了一种特殊的节类型 SHT_INIT_ARRAY。这些初始值设定项包含类型 R_ARM_TARGET1 到执行初始化的符号的重定位。

ELF ABI 规范将 R_ARM_TARGET1 描述为相对格式或绝对格式。

ARM C 库使用相对格式。例如，如果链接器检测到 ARM C 库 __cpp_initialize_aeabi 的定义，则会使用 R_ARM_TARGET1 的相对格式；否则将使用绝对格式。

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential

4.5.6 相关链接器命令行选项

- 第2-5 页的*--bpabi*
- 第2-17 页的*--dll*
- 第2-17 页的*--dynamic_debug*
- 第2-20 页的*--[no_]export_all*
- 第2-41 页的*--pltgot=type*
- 第2-41 页的*--pltgot_opts=mode*
- 第2-48 页的*--ropi*
- 第2-48 页的*--ro_base=address*
- 第2-49 页的*--rosplit*
- 第2-49 页的*--runpath=pathlist*
- 第2-49 页的*--rw_base=address*
- 第2-50 页的*--rwpi*
- 第2-53 页的*--soname=name*
- 第2-57 页的*--symver_script=file*
- 第2-57 页的*--symver_soname*

另请参阅

- 《链接器用户指南》中的以下内容：
  - 第4-15 页的*符号版本控制*

Copyright © 2008 ARM Limited. All rights reserved.
Non-Confidential