# RealView® 编译工具

**4.0 版**

## 实用程序指南

ARM®


ARM DUI 0382AC

# RealView 编译工具
## 实用程序指南



### 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 | RealView Development Suite v4.0 4.0 版 |

### 所有权声明



### 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`

# 目录
# RealView 编译工具
# 实用程序指南

# 前言

本前言介绍 《RealView 编译工具实用程序指南》。本前言分为以下几节：

# 关于本手册

本手册提供了有关 ARM RealView® 编译工具所支持的以下 ARM® 实用程序工具的信息：

- ARM 映像转换实用程序 `fromelf`
- ARM 库管理程序 `armar`

本手册提供有关每个工具的命令行选项的详细信息，并说明如何调用这些工具。

## 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***

本章简要介绍了随 RealView 编译工具提供的 ARM 实用程序 `fromelf` 和 `armar`。

**第 2 章 *使用 fromelf***

本章介绍了 `fromelf` 实用程序以及如何使用该工具更改映像格式。

**第 3 章 *使用 armar***

本章说明了创建和访问对象库时所涉及的过程。

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。

`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

本手册包含有关如何使用随 RealView 编译工具提供的 ARM 实用程序工具的信息。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 编译工具链接器参考指南》 (ARM DUI 0381)
- 《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\...。

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)

- 《ARMv7-M 体系结构参考手册》 (ARM DDI 0403)
- 《ARMv6-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

# 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。

# 第 1 章
# 简介

本章介绍随 ARM RealView® 编译工具一起提供的 ARM® 实用程序工具 `fromelf` 和 `armar`。本章内容如下：

## 1.1 关于 ARM 映像转换实用程序

使用 ARM 映像转换实用程序 fromelf 可以处理 ARM 编译器、 ARM 汇编器和 ARM 链接器生成的 ARM *可执行和链接格式* (ELF) 对象文件和映像文件。

fromelf 可以将 ELF 映像转换为其他格式，以供 ROM 工具使用并直接加载到内存中。可用格式有：

- 纯二进制
- Motorola 32 位 S-record
- Intel Hex-32
- 面向字节 （Verilog 内存模型）的十六进制格式
- ELF。可以重新保存为 ELF，例如，用于从 ELF 映像删除调试信息

使用 fromelf 可以保护映像中的*知识产权* (IP) 和传递到第三方的对象。

使用 fromelf 可以指定包含某些信息 （如用于特定配置的命令行选项）的项目模板。随 RVDS 提供了适用于 ARM 处理器和各种 ARM 开发板的项目模板。然而，您可以使用这些模板创建自己的特定于项目的配置。

fromelf 还可以将输入文件的信息 （例如，反汇编输出或符号列表）输出到 stdout 或文本文件。

fromelf 具有以下执行模式：

- ELF 模式 (--elf)，用于将文件重新保存为 ELF
- 文本模式 （--text 以及其他），用于输出有关对象或映像文件的信息
- 格式转换模式 （--bin、--m32、--i32、--vhx）

**注意**

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

有关详细信息，请参阅第 2 章 *使用 fromelf*。

### 1.1.1 使用 fromelf 时的注意事项

有以下几点需要注意：

- 如果使用 fromelf 通过 --bin、--m32、--i32 或 --vhx 选项之一将包含多个加载区的 ELF 映像转换为二进制格式，则 fromelf 会创建一个名为 *destination* 的输出目录，并且为输入映像中的每个加载区都生成一个二进制输出文件。fromelf 将这些输出文件放置在 *destination* 目录中。

  **注意**

  对于多个加载区，将对应加载区中的第一个非空执行区的名称用作文件名。

  如果使用 --m32combined 或 --i32combined 选项转换包含多个加载区的 ELF 映像，则 fromelf 会创建一个名为 *destination* 的输出目录，并为输入映像中的所有加载区生成一个二进制输出文件，然后将该输出文件放置在 *destination* 目录中。

  例如，如果用定义多个加载区的分散加载描述文件生成 ELF 映像，则这些映像会包含多个加载区。

- 使用 fromelf 时，不能进行以下操作：
  - 除了可以使用 --base 选项改变 Motorola S-record 或 Intel Hex 输出的基址之外，不能更改映像结构或地址。
  - 不能将分散加载的 ELF 映像更改为另一格式的非分散加载映像。链接时必须向链接器提供所有结构或寻址信息。

## 1.2 关于 ARM 库管理程序

ARM 库管理程序 armar 可用于在标准格式 ar 库中收集和维护 ELF 对象文件集。您可以将这些库传递到链接器来替代若干 ELF 对象文件。

armar 可以：

- 创建新库
- 向库中添加文件
- 替换库中的各个文件
- 在单个操作中使用指定文件替换库中的所有文件
- 控制文件在库中的放置

使用 armar 可以指定包含某些信息 （如用于特定配置的命令行选项）的项目模板。随 RVDS 提供了适用于 ARM 处理器和各种 ARM 开发板的项目模板。然而，您可以使用这些模板创建自己的特定于项目的配置。

armar 还可以显示指定库的相关信息。例如，可以列出库中的所有成员。

**注意**

当在库中创建、添加或替换对象文件时， armar 在缺省情况下会创建符号表。

另请参阅：

- 第 3 章 *使用 armar*
- 有关链接器如何处理其输入文件的信息，请参阅 《链接器用户指南》中的第 2 章 *ARM 链接器使用入门*。
- 《库和浮点支持指南》。

### 1.2.1 使用库文件时的注意事项

有以下几点需要注意：

- 库与共享对象或*动态链接库* (DLL) 的差异在于：
  - 符号从共享对象或 DLL 中导入
  - 符号的代码或数据从档案提取到要链接的文件中

- 链接对象库文件所生成的结果与链接集中到对象库文件中的所有对象文件所生成的结果可能不相同。这是因为链接器处理输入列表和库的方式不同：
  - 尽管指定 armlink --remove 选项后将删除未使用区，但是输入列表中的每个对象文件都将无条件地出现在输出中。
  - 仅当对象文件或前面处理的库文件引用了某库文件的成员时，该成员才包含在输出中。

  链接器将存储在 ar 格式文件中的 ELF 文件集合视为一个库。每个 ELF 文件的内容构成库中的单个成员。

# 第 2 章
# 使用 fromelf

本章介绍 ARM RealView® 编译工具附带的 ARM® 映像转换实用程序 `fromelf`。本章内容如下：

## 2.1 使用命令行选项

指定命令行选项时，根据选项类型应用下列规则：

**单字母选项**

所有单字母选项都以单短划线 - 开头。

**关键字选项**

所有关键字选项都以双短划线 -- 开头。选项和参数之间需要用 = 或空格字符分隔。

例如：

```
--diag_style=ide
--diag_style ide
```

### 2.1.1 调用 fromelf

fromelf 命令行语法如下：

```
fromelf [build-options] [debug-options] [diagnostic-options] [help-options]
[image-content-options] [license-option] [output-options] [privacy-options]
[project-template-options] input_file
```

*build-options*

使用下列选项控制生成属性的输出格式：

- 第2-16 页的*--decode_build_attributes*
- 第2-19 页的*--dump_build_attributes*
- 第2-22 页的*--extract_build_attributes*

*debug-options*

使用下列选项控制输出文件中的调试信息：

- 第2-15 页的*--[no_]debug*
- 第2-16 页的*--debugonly*

*diagnostic-options*

使用下列选项控制输出文件中的诊断信息：

- 第2-17 页的*--diag_style={arm|ide|gnu}*
- 第2-18 页的*--diag_suppress=tag[,tag,...]*
- 第2-13 页的*--compare=option[,option,...]*
- 第2-14 页的*--continue_on_error*

- 第2-28 页的*--ignore_section=option[,option,...]*
- 第2-28 页的*--ignore_symbol=option[,option,...]*
- 第2-38 页的*--relax_section=option[,option,...]*
- 第2-39 页的*--relax_symbol=option[,option,...]*
- 第2-42 页的*--show_cmdline*

*help-options*

使用下列选项显示编译器的主命令行选项和版本号：

- 第2-25 页的*--help*
- 第2-49 页的*--vsn*

*image-content-options*

使用以下选项可控制影响映像内容的其他因素：

- 第2-7 页的*--base [[object_file::]load_region_ID=]num*
- 第2-11 页的*--cad*
- 第2-12 页的*--cadcombined*
- 第2-14 页的*--cpu=list*
- 第2-14 页的*--cpu=name*
- 第2-17 页的*--device=list*
- 第2-17 页的*--device=name*
- 第2-18 页的*--disassemble*
- 第2-19 页的*--emit=option[,option,...]*
- 第2-22 页的*--expandarrays*
- 第2-22 页的*--fieldoffsets*
- 第2-24 页的*--fpu=list*
- 第2-24 页的*--fpu=name*
- 第2-24 页的*--globalize=option[,option,...]*
- 第2-25 页的*--hide=option[,option,...]*
- 第2-26 页的*--hide_and_localize=option[,option,...]*
- 第2-30 页的*--info=topic[,topic,...]*
- 第2-29 页的*--in_place*
- 第2-31 页的*--interleave=option*
- 第2-33 页的*--[no_]linkview*

- 第2-34 页的*--localize=option[,option,...]*
- 第2-37 页的*--qualify*
- 第2-39 页的*--rename=option[,option,...]*
- 第2-40 页的*--select=select_options*
- 第2-41 页的*--show=option[,option,...]*
- 第2-42 页的*--show_and_globalize=option[,option,...]*
- 第2-42 页的*--source_directory=path*
- 第2-43 页的*--strip=option[,option,...]*
- 第2-44 页的*--[no_]symbolversions*

*license-option*

要进一步尝试获取浮动许可证，请使用第2-32 页的*--licretry*。

*output-options*

使用下列选项控制输出格式：

- 第2-8 页的*--bin*
- 第2-9 页的*--bincombined*
- 第2-9 页的*--bincombined_base=address*
- 第2-10 页的*--bincombined_padding=size,num*
- 第2-19 页的*--elf*
- 第2-27 页的*--i32*
- 第2-27 页的*--i32combined*
- 第2-34 页的*--m32*
- 第2-35 页的*--m32combined*
- 第2-35 页的*--output=destination*
- 第2-44 页的*--text*
- 第2-46 页的*--widthxbanks*
- 第2-49 页的*--vhx*

*privacy-options*

使用下列选项保护传递到第三方的映像和对象中的 IP：

- 第2-36 页的*--privacy*
- 第2-43 页的*--strip=option[,option,...]*

*project-template-options*

使用以下选项可控制项目模板的使用：

- 第2-36 页的*--[no_]project=filename*
- 第2-37 页的*--reinitialize_workdir*
- 第2-48 页的*--workdir=directory*

*input_file* 指定要处理的 ELF 文件，具体说明见第2-31 页的*input_file*。

### 2.1.2 排序命令行选项

通常，命令行选项可以按任意顺序出现。不过，一些选项的效果取决于它们如何与其他相关选项组合。

如果同一命令行中的某些选项覆盖了其他选项，则靠近命令行末尾的选项优先。如果某个选项不遵循此规则，则会在该选项的说明中注明。使用 `--show_cmdline` 选项可以查看 `fromelf` 如何处理命令行。命令以标准化方式显示。

**另请参阅**

- 第2-42 页的*--show_cmdline*

## 2.2 命令行选项

本节按字母顺序列出了映像转换实用程序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。

例如，应输入 '*,~*.*' 而不是 *,~*.*。

### 2.2.1 --base [[*object_file*::]*load_region_ID*=]*num*

使用此选项可以更改为 Motorola Srecord 和 Intel Hex 文件格式的一个或多个加载区指定的基址。

#### 限制

必须对此选项使用下列输出格式之一：--i32、--i32combined、--m32 或 --m32combined。

#### 语法

--base [[*object_file*::]*load_region_ID*=]*num*

其中：

*object_file* 是一个可选的 ELF 输入文件。

*load_region_ID*

是一个可选的加载区。它可以是属于某一加载区的执行区的符号名称，也可以是从零开始的加载区编号；例如，如果指第一个区，则为 #0。

*num* 是十进制或十六进制值。

通配符 ? 和 * 可用于 *object_file* 和 *load_region_ID* 参数中的符号名称。可在一个 --base 选项后面跟随逗号分隔的参数列表，从而指定多个基址。第2-8 页的表 2-1 显示了一些示例。

输出文件中编码的所有地址都从基址 *num* 开始。如果未指定 --base 选项，则从加载区地址获取基址。

**表 2-1 使用 --base 的示例**

| | |
|---|---|
| --base 0 | 十进制值 |
| --base 0x8000 | 十六进制值 |
| --base #0=0 | 第一个加载区的基址 |
| --base foo.o::*=0 | foo.o 中的所有加载区的基址 |
| --base #0=0,#1=0x8000 | 第一个和第二个加载区的基址 |

### 另请参阅

- 第2-27 页的*--i32*
- 第2-27 页的*--i32combined*
- 第2-34 页的*--m32*
- 第2-35 页的*--m32combined*
- 第1-3 页的*使用 fromelf 时的注意事项*

## 2.2.2 --bin

此选项为每个加载区分别生成一个纯二进制输出文件。可以使用 *memory_config* 选项将此选项的输出分为多个文件。

### 示例

要将 ELF 文件转换为纯二进制文件 （例如， outfile.bin）文件，请使用：

```
fromelf --bin --output=outfile.bin infile.axf
```

### 另请参阅

- 第2-46 页的*--widthxbanks*
- 第1-3 页的*使用 fromelf 时的注意事项*

### 2.2.3 --bincombined

此选项生成纯二进制输出。它为包含多个加载区的映像生成一个输出文件。缺省情况下，将内存中第一个加载区的起始地址用作基址。fromelf 根据需要在两个加载区之间插入填充，以确保它们之间的相对偏移量正确。通过用这种方式分隔加载区，可将输出文件加载到内存并从基址开始正确对齐。

此选项与 --bincombined_base 和 --bincombined_padding 一起使用时，可更改基址和填充的缺省值。

#### 另请参阅

- *--bincombined_base=address*
- 第2-10 页的*--bincombined_padding=size,num*

### 2.2.4 --bincombined_base=*address*

使用此选项可以降低 --bincombined 输出模式所使用的基址。它所生成的输出文件适合从指定地址开始加载到内存中。

#### 限制

此选项必须与 --bincombined 一起使用。

#### 语法

--bincombined=*address*

其中：

*address* 要在其中加载映像的起始地址：

- 如果指定地址低于第一个加载区的起始地址，则 fromelf 会在输出文件的开头添加填充
- 如果指定地址高于第一个加载区的起始地址，则 fromelf 会生成错误。

#### 缺省选项

缺省情况下，将内存中的第一个加载区的起始地址用作基址。

#### 示例

```
--bincombined --bincombined_base=0x1000
```

### 另请参阅

- 第2-9 页的*--bincombined*
- *--bincombined_padding=size,num*

## 2.2.5 --bincombined_padding=*size*,*num*

使用此选项可以为 --bincombined 输出模式所使用的填充指定不同的形式。

### 限制

此选项必须与 --bincombined 一起使用。

### 语法

--bincombined_padding=*size*,*num*

其中：

*size* 为 1、 2 或 4 字节，用于定义填充是字节、半字还是字。

*num* 是用于填充的值。

**注意**

fromelf 要求在输入文件的适当端标记中指定 2 字节和 4 字节填充值。例如，如果要将大端 ELF 文件转换为二进制文件，则将指定的填充值视为大端字或半字。

### 缺省选项

缺省为 --bincombined_padding=1,0xFF。

### 示例

以下各示例显示如何使用 --bincombined_padding：

--bincombined --bincombined_padding=4,0x12345678

此示例生成纯二进制输出并用重复的 32 位字 0x12345678 填充两个加载区之间的空白。

--bincombined --bincombined_padding=2,0x1234

此示例生成纯二进制输出并用重复的 16 位半字 0x1234 填充两个加载区之间的空白。

--bincombined --bincombined_padding=2,0x01

在指定用于大端内存的情况下，此示例用 0x0100 填充两个加载区之间的空白。

**另请参阅**

- 第2-9 页的*--bincombined*
- 第2-9 页的*--bincombined_base=address*

### 2.2.6 --cad

此选项生成一个包含二进制输出的 C 数组定义或 C++ 数组定义 您可以在另一个应用程序的源代码中使用每个数组定义。例如，您可能需要在另一个应用程序（如嵌入式操作系统）的地址空间中嵌入一个映像。

如果映像只有一个加载区，则缺省情况下输出将定向到 stdout。要将输出保存到文件，请配合文件名使用 --output 选项。

如果映像有多个加载区，则也必须配合文件名使用 --output 选项。除非指定了完整的路径名称，否则该路径相对于当前目录。将为指定目录中的每个加载区分别创建一个文件。每个文件的名称就是相应执行区的名称。

此选项与 --output 一起使用时，可为映像中的每个加载区生成一个输出文件。

**示例**

下面的示例显示如何使用 --cad：

- 要为只有一个加载区的映像生成数组定义，请使用：

```
fromelf --cad myimage.axf
unsigned char LR0[] = {
    0x00,0x00,0x00,0xEB,0x28,0x00,0x00,0xEB,0x2C,0x00,0x8F,0xE2,0x00,0x0C,0x90,0xE8,
    0x00,0xA0,0x8A,0xE0,0x00,0xB0,0x8B,0xE0,0x01,0x70,0x4A,0xE2,0x0B,0x00,0x5A,0xE1,
    0x00,0x00,0x00,0x1A,0x20,0x00,0x00,0xEB,0x0F,0x00,0xBA,0xE8,0x18,0xE0,0x4F,0xE2,
    0x01,0x00,0x13,0xE3,0x03,0xF0,0x47,0x10,0x03,0xF0,0xA0,0xE1,0xAC,0x18,0x00,0x00,
    0xBC,0x18,0x00,0x00,0x00,0x30,0xB0,0xE3,0x00,0x40,0xB0,0xE3,0x00,0x50,0xB0,0xE3,
    0x00,0x60,0xB0,0xE3,0x10,0x20,0x52,0xE2,0x78,0x00,0xA1,0x28,0xFC,0xFF,0xFF,0x8A,
    0x82,0x2E,0xB0,0xE1,0x30,0x00,0xA1,0x28,0x00,0x30,0x81,0x45,0x0E,0xF0,0xA0,0xE1,
    0x70,0x00,0x51,0xE3,0x66,0x00,0x00,0x0A,0x64,0x00,0x51,0xE3,0x38,0x00,0x00,0x0A,
    0x00,0x00,0xB0,0xE3,0x0E,0xF0,0xA0,0xE1,0x1F,0x40,0x2D,0xE9,0x00,0x00,0xA0,0xE1,
.
.
.
    0x3A,0x74,0x74,0x00,0x43,0x6F,0x6E,0x73,0x74,0x72,0x75,0x63,0x74,0x65,0x64,0x20,
    0x41,0x20,0x23,0x25,0x64,0x20,0x61,0x74,0x20,0x25,0x70,0x0A,0x00,0x00,0x00,0x00,
```

```
    0x44,0x65,0x73,0x74,0x72,0x6F,0x79,0x65,0x64,0x20,0x41,0x20,0x23,0x25,0x64,0x20,
    0x61,0x74,0x20,0x25,0x70,0x0A,0x00,0x00,0x0C,0x99,0x00,0x00,0x0C,0x99,0x00,0x00,
    0x50,0x01,0x00,0x00,0x44,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
};
```

- 对于具有多个加载区的映像，以下命令将在目录 *root*\myprojects\multiload\load_regions 中为每个加载区创建一个文件：

  **cd *root*\myprojects\multiload**
  **fromelf --cad image_multiload.axf --output load_regions**

  如果 image_multiload.axf 包含执行区 EXEC_ROM 和 RAM，则将在 load_regions 子目录中创建文件 EXEC_ROM 和 RAM。

**另请参阅**

- *--cadcombined*
- 第2-35 页的*--output=destination*
- 《链接器用户指南》中的第 5 章 *使用分散加载描述文件*

### 2.2.7 --cadcombined

此选项生成一个包含二进制输出的 C 数组定义或 C++ 数组定义 您可以在另一个应用程序的源代码中使用每个数组定义。例如，您可能需要在另一个应用程序（如嵌入式操作系统）的地址空间中嵌入一个映像。

如果映像只有一个加载区，则缺省情况下输出将定向到 stdout。要将输出保存到文件，请配合文件名使用 --output 选项。

如果映像有多个加载区，则也必须配合文件名使用 --output 选项。该文件将包含多个加载区。

**示例**

对于具有多个加载区的映像，指定以下命令将为所有加载区创建一个包含数组定义的文件：

对于具有多个加载区的映像，以下命令将在目录 *root*\myprojects\multiload 中创建文件 load_regions.c：

**cd *root*\myprojects\multiload**
**fromelf --cadcombined image_multiload.axf --output load_regions.c**

### 另请参阅

- 第2-11 页的*--cad*
- 第2-35 页的*--output=destination*
- 《链接器用户指南》中的第 5 章 *使用分散加载描述文件*

## 2.2.8 --compare=*option*[,*option*,...]

此选项对两个输入文件进行比较，并在文本列表中输出两者的差异。两个输入文件必须属于同一类型，例如，两个 ELF 文件或两个二进制文件。库文件将按成员进行比较，并在输出中将成员差异连接起来。

两个输入文件之间的所有差异均报告为错误，除非使用 --relax_section 选项明确将错误降级为警告。

### 语法

--compare=*option*[,*option*,...]

其中 *option* 为下列项之一：

section_sizes

比较每个 ELF 文件 （或库文件的每个 ELF 成员）的所有节的大小。

section_sizes::*object_name*

比较名称与 *object_name* 匹配的 ELF 对象中的所有节的大小。

section_sizes::*section_name*

比较名称与 *section_name* 匹配的所有节的大小。

sections 比较每个 ELF 文件 （或库文件的每个 ELF 成员）的所有节的大小和内容。

sections::*object_name*

比较名称与 *object_name* 匹配的 ELF 对象中所有节的大小和内容。

sections::*section_name*

比较名称与 *section_name* 匹配的所有节的大小和内容。

function_sizes

比较每个 ELF 文件 （或库文件的每个 ELF 成员）的所有函数的大小。

function_sizes::*object_name*

比较名称与 *object_name* 匹配的 ELF 对象中所有函数的大小。

function_size::*function_name*

比较名称与 *function_name* 匹配的所有函数的大小。

global_function_sizes

比较每个 ELF 文件 （或库文件的每个 ELF 成员）的所有全局函数的大小。

global_function_sizes::*function_name*

比较 ELF 对象中名称与 *function_name* 匹配的所有全局函数的大小。

? 和 * 通配符可用于 *object_name* 和 *section_name* 参数中的符号名称。可在一个 --compare 选项后面跟随逗号分隔的参数列表，从而指定多个选项。

**另请参阅**

- 第2-28 页的*--ignore_section=option[,option,...]*
- 第2-28 页的*--ignore_symbol=option[,option,...]*
- 第2-38 页的*--relax_section=option[,option,...]*
- 第2-39 页的*--relax_symbol=option[,option,...]*

### 2.2.9 --continue_on_error

此选项将所有可恢复错误降级为警告，以便 fromelf 可以处理损坏的 ELF 文件。

### 2.2.10 --cpu=list

此选项列出可用于 --cpu=*name* 的受支持体系结构和处理器的名称。

**另请参阅**

- *--cpu=name*

### 2.2.11 --cpu=*name*

此选项为特定的 ARM 处理器或体系结构选择反汇编。它会影响 fromelf 解释其在输入文件中找到的指令的方式。

### 语法

--cpu=*name*

其中 *name* 是 ARM 处理器或体系结构的名称。

### 示例

要为 ARM1176JZF-S™ 处理器选择反汇编，请使用：

```
--cpu=ARM1176JZF-S
```

### 另请参阅

- 第2-14 页的*--cpu=list*
- 第2-17 页的*--device=list*
- 第2-17 页的*--device=name*
- 第2-18 页的*--disassemble*
- 第2-30 页的*--info=topic[,topic,...]*
- 第2-44 页的*--text*

## 2.2.12 --datasymbols

此选项修改数据节的输出信息以便交叉存取符号定义。

此选项只能与 --text -d 一起使用。

### 另请参阅

- 第2-44 页的*--text*

## 2.2.13 --[no_]debug

此选项控制是否从 ELF 映像中删除调试信息。

--debug 必须与 --elf 一起使用。如果指定了 --no_debug，则影响所有输出格式。

--debug 覆盖 --text -g 选项。

**注意**

此选项仅作为旧式源代码的迁移辅助选项。不建议使用此选项。应改用 --strip=debug,symbols 选项替代此选项。

#### 缺省选项

对于二进制映像，缺省值是 --no_debug。

#### 另请参阅

- 第2-19 页的*--elf*
- 第2-44 页的*--text*
- 第2-43 页的*--strip=option[,option,...]*

### 2.2.14 --debugonly

此选项删除所有代码或数据节的内容。这可确保输出文件中仅包含调试所需的信息，例如，调试节、符号表和字符串表。由于需要将节的头文件作为符号的目标使用，因此将保留节的头文件。

#### 限制

此选项必须与 --elf 一起使用。

#### 另请参阅

- 第2-19 页的*--elf*

### 2.2.15 --decode_build_attributes

此选项以可阅读的格式 （对于标准生成属性）或原始十六进制格式 （对于非标准生成属性）输出生成属性节的内容。

**注意**

标准生成属性在 《ARM 体系结构的应用程序二进制接口》中有文档记载。

#### 限制

此选项只能用在文本模式中。

#### 另请参阅

- 第2-19 页的*--dump_build_attributes*
- 第2-22 页的*--extract_build_attributes*
- 《ARM 体系结构的应用程序二进制接口》。

### 2.2.16 --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*

### 2.2.17 --device=*name*

使用此选项可以指定微控制器或*芯片上系统* (SoC) 设备名称，而不是 CPU 名称。它的格式与 ARM 编译器支持的格式相同。

每个设备对于 CPU 和*浮点单元* (FPU) 都有缺省值。但是，通过在 --device 选项后面指定 --fpu 选项，可以从命令行覆盖 FPU。

有关 CPU 和 FPU 实现的详细信息，请参阅设备文档。

#### 语法

--device=*name*

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*--cpu=list*
- 第2-14 页的*--cpu=name*
- *--device=list*
- 第2-24 页的*--fpu=list*
- 第2-24 页的*--fpu=name*
- 《编译器参考指南》中第2-39 页的*--device=name*
- 《链接器参考指南》中第2-14 页的*--device=name*

### 2.2.18 --diag_style={arm|ide|gnu}

此选项指定用于显示诊断消息的样式。

#### 语法

--diag_style=*string*

其中 *string* 是下列项之一：

arm　　采用 ARM 样式显示消息。

ide　　包含任意出错行的行号和字符计数。这些值将显示在括号中。

gnu　　以 GNU 所使用的格式显示消息。

#### 缺省选项

缺省为 --diag_style=arm。

#### 另请参阅

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

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

此选项禁用具有指定标签的诊断消息。

#### 语法

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

其中：

*tag*[,*tag*,...]　　是一个以逗号分隔的诊断消息编号列表，用于指定要禁止的消息。

### 2.2.20 --disassemble

此选项向 stdout 显示映像的反汇编版本。如果将此选项与 --output *destination* 一起使用，则可以用 armasm 反汇编输出文件。

可以使用此选项对 ELF 映像或 ELF 对象文件进行反汇编。

**——注意——**

此选项的输出不同于 --emit=code 和 --text -c 的输出。

#### 另请参阅

- 第2-14 页的*--cpu=name*
- 第2-19 页的*--emit=option[,option,...]*
- 第2-31 页的*--interleave=option*

- 第2-35 页的*--output=destination*
- 第2-44 页的*--text*

### 2.2.21 --dump_build_attributes

此选项以原始十六进制格式输出生成属性节的内容。

#### 限制

此选项只能用在文本模式中。

#### 另请参阅

- 第2-16 页的*--decode_build_attributes*
- 第2-22 页的*--extract_build_attributes*
- 第2-44 页的*--text*

### 2.2.22 --elf

此选项选择 ELF 输出模式。

与 --strip=debug,symbols 一起使用时，可删除 ELF 映像中的调试信息。

#### 另请参阅

- 第2-18 页的*--disassemble*
- 第2-29 页的*--in_place*
- 第2-35 页的*--output=destination*
- 第2-43 页的*--strip=option[,option,...]*

### 2.2.23 --emit=*option*[,*option*,...]

使用此选项可以指定要显示在文本输出中的 ELF 对象元素。该输出包含 ELF 头信息和节信息。

#### 限制

此选项只能用在文本模式中。

#### 语法

--emit=*option*[,*option*,...]

其中 *option* 是下列项之一：

addresses 此选项输出全局和静态数据地址（包括结构和联合内容的地址）。它与 --text -a 的作用相同。

此选项只能用于包含调试信息的文件。如果没有调试信息，则会生成一条警告消息。

使用 --select 选项可以输出数据地址的子集。

如果要查看在结构内外展开的数组的数据地址，请使用带此文本类别的 --expandarrays 选项。

build_attributes

此选项以可阅读的格式（对于标准生成属性）或原始十六进制格式（对于非标准生成属性）输出生成属性节的内容。

代码 此选项反汇编代码，并转储正被反汇编的原始二进制数据及指令地址。它与 --text -c 的作用相同。

**注意**

与 --disassemble 不同的是，不能将反汇编输入到 ARM 汇编器。

数据 此选项输出数据节的内容。它与 --text -d 的作用相同。

data_symbols

此选项修改数据节的输出信息以便交叉存取符号定义。

debug_info 此选项输出调试信息。它与 --text -g 的作用相同。

dynamic_segment

此选项输出动态段内容。它与 --text -y 的作用相同。

exception_tables

此选项解码对象的异常表信息。它与 --text -e 的作用相同。

frame_directives

此选项输出反汇编代码中由嵌入在对象模块中的调试信息所指定的 FRAME 指令的内容。

此选项与 --disassemble 一起使用。

got 此选项输出*全局偏移表* (GOT) 对象的内容。

heading_comments

此选项输出位于反汇编开头的标题注释，其中包含 .comment 节中的工具和命令行信息。

此选项与 --disassemble 一起使用。

raw_build_attributes

此选项以原始十六进制格式 （即与数据相同的格式）输出生成属性节的内容。

relocation_tables

此选项输出重定位信息。它与 --text -r 的作用相同。

string_tables

此选项输出字符串表。它与 --text -t 的作用相同。

summary 此选项输出文件中段和节的摘要。它是 fromelf --text 的缺省输出。但是，某些 --info 选项会禁止显示该摘要。如果需要，可使用 --emit summary 显式重新启用该摘要。

symbol_annotations

此选项输出反汇编代码和数据中标有注释 （包含相应的属性信息）的符号。

此选项与 --disassemble 一起使用。

symbol_tables

此选项输出符号表和版本控制表。它与 --text -s 的作用相同。

vfe 此选项输出有关未使用的虚函数的信息。

whole_segments

此选项按段输出反汇编可执行文件或共享库，即使段包含链接视图也如此。

此选项与 --disassemble 一起使用。

可在一个 --emit 选项后面跟随逗号分隔的参数列表，从而指定多个选项。

### 另请参阅

- 第2-18 页的*--disassemble*
- 第2-44 页的*--text*

### 2.2.24 --expandarrays

此选项输出数据地址，包括在结构内外展开的数组。

#### 限制

此选项只能与 --text -a 一起使用。

#### 另请参阅

- 第2-44 页的*--text*

### 2.2.25 --extract_build_attributes

此选项仅输出生成属性，输出格式为：

- 可阅读的格式 （对于标准生成属性）
- 原始十六进制格式 （对于非标准生成属性）。

#### 限制

此选项只能用在文本模式中。

#### 另请参阅

- 第2-16 页的*--decode_build_attributes*
- 第2-19 页的*--dump_build_attributes*
- 第2-44 页的*--text*

### 2.2.26 --fieldoffsets

此选项输出汇编语言 EQU 指令的列表，这些指令将 C++ 类或 C 结构的字段名视为等同于该类或结构的基址的相应偏移量。输入 ELF 文件可以是可重定位对象或映像。

使用 --output 可将该输出重定向到文件。使用 armasm 的 INCLUDE 命令加载生成的文件，并按汇编语言的名称提供对 C++ 类和 C 结构成员的访问。有关 armasm 的详细信息，请参阅 《汇编器指南》。

此选项输出所有结构信息。要输出结构的子集，请使用 --select *select_options*。

如果不需要可以输入到 armasm 的文件，则使用 --text -a 选项，将显示的地址设置为易于阅读的格式。-a 选项只输出结构的地址信息和映像中的静态数据，因为地址在可重定位对象中是未知的。

**注意**

如果需要 fromelf --fieldoffsets 步骤，则不要使用 --no_debug。

### 限制

此选项：

- 在源文件没有调试信息的情况下不可用
- 只能用在文本模式中。

### 示例

下面的示例显示如何使用 --fieldoffsets：

- 要将输出列表生成到 stdout 中，其中包含文件 inputfile.o 中所有结构的全部字段偏移，请使用：

  ```
  fromelf --fieldoffsets inputfile.o
  ```

- 要生成输出到 outputfile.a 的文件列表，其中包含文件 inputfile.o 中名称以 p 开头的结构的所有字段偏移，请使用：

  ```
  fromelf --fieldoffsets --select=p* --output=outputfile.a inputfile.o
  ```

- 要生成输出到 outputfile.a 的列表，其中包含文件 inputfile.o 中名为 tools 或 moretools 的结构的所有字段偏移，请使用：

  ```
  fromelf --fieldoffsets --select=tools.*,moretools.* --output=outputfile.a
  inputfile.o
  ```

- 可使用以下命令将输出文件列表生成到 outputfile.a 中，其中包含文件 inputfile.o 中这样一些结构字段的所有字段偏移：这些字段的名称以 number 开头，并且这些字段位于 tools 结构的 top 结构字段中。

  ```
  fromelf --fieldoffsets --select=tools.top.number* --output=outputfile.a
  inputfile.o
  ```

### 另请参阅


- 第2-15 页的*--[no_]debug*
- 第2-37 页的*--qualify*
- 第2-40 页的*--select=select_options*

- 第2-44 页的*--text*
- 《汇编器指南》

### 2.2.27 --fpu=list

此选项列出可与 --fpu=*name* 选项结合使用的受支持 FPU 体系结构名称。

#### 另请参阅

- *--fpu=name*

### 2.2.28 --fpu=*name*

此选项为特定的 FPU 体系结构选择反汇编。它会影响 fromelf 解释其在输入文件中找到的指令的方式。

#### 语法

--fpu=*name*

其中 *name* 是受支持 FPU 体系结构的名称。

#### 示例

要为 VFPv3 体系结构选择反汇编，请使用：

```
--fpu=VFPv3
```

#### 另请参阅

- 第2-17 页的*--device=list*
- 第2-17 页的*--device=name*
- 第2-18 页的*--disassemble*
- *--fpu=list*
- 第2-30 页的*--info=topic[,topic,...]*
- 第2-44 页的*--text*

### 2.2.29 --globalize=*option*[,*option*,...]

此选项将所选符号转换为全局符号。

#### 限制

此选项必须与 --elf 一起使用。

#### 语法

--globalize=*option*[,*option*,...]

其中 *option* 为下列项之一：

*object_name*::

名称与 *object_name* 匹配的 ELF 对象中的所有符号都转换为全局符号。

*object_name*::*symbol_name*

名称与 *object_name* 匹配的 ELF 对象中符号名称与 symbol_name 匹配的所有符号都转换为全局符号。

*symbol_name* 符号名称与 *symbol_name* 匹配的所有符号都转换为全局符号。

通配符 ? 和 * 可用于 *symbol_name* 和 *object_name* 参数中的符号名称。可在一个 --globalize 选项后面跟随逗号分隔的参数列表，从而指定多个选项。

#### 另请参阅

- 第2-19 页的*--elf*
- *--hide=option[,option,...]*

### 2.2.30 --help

此选项汇总显示主要的命令行选项。

如果未指定任何选项或源文件，则它是缺省选项。

#### 另请参阅

- 第2-42 页的*--show_cmdline*
- 第2-49 页的*--vsn*

### 2.2.31 --hide=*option*[,*option*,...]

此选项更改符号可见性属性，以将所选符号标记为隐藏。

#### 限制

此选项必须与 --elf 一起使用。

#### 语法

--hide=*option*[,*option*,...]

其中 *option* 为下列项之一：

*object_name*::

名称与 *object_name* 匹配的 ELF 对象中的所有符号。

*object_name*::*symbol_name*

名称与 *object_name* 匹配的 ELF 对象中符号名称与 symbol_name 匹配的所有符号。

*symbol_name* 符号名称与 *symbol_name* 匹配的所有符号。

通配符 ? 和 * 可用于 *symbol_name* 和 *object_name* 参数中的符号名称。可在一个 --hide 选项后面跟随逗号分隔的参数列表，从而指定多个选项。

#### 另请参阅

- 第2-19 页的*--elf*
- 第2-41 页的*--show=option[,option,...]*

### 2.2.32 --hide_and_localize=*option*[,*option*,...]

此选项更改符号可见性属性以将所选符号标记为隐藏，并将所选符号转换为局部符号。

#### 限制

此选项必须与 --elf 一起使用。

#### 语法

--hide_and_localize=*option*[,*option*,...]

其中 *option* 为下列项之一：

*object_name*::

名称与 *object_name* 匹配的 ELF 对象中的所有符号都标记为隐藏并转换为局部符号。

*object_name*::*symbol_name*

名称与 *object_name* 匹配的 ELF 对象中符号名称与 symbol_name 匹配的所有符号都标记为隐藏并转换为局部符号。

*symbol_name* 符号名称与 *symbol_name* 匹配的所有符号都标记为隐藏并转换为局部符号。

通配符 ? 和 * 可用于 *symbol_name* 和 *object_name* 参数中的符号名称。可在一个 --hide_and_localize 选项后面跟随逗号分隔的参数列表，从而指定多个选项。

**另请参阅**

- 第2-19 页的*--elf*

### 2.2.33 --i32

此选项生成 Intel Hex-32 格式的输出。它为映像中的每个加载区生成一个输出文件。可以使用 --base 选项指定输出的基址。

**另请参阅**

- 第2-7 页的*--base [[object_file::]load_region_ID=]num*
- 第1-3 页的*使用 fromelf 时的注意事项*

### 2.2.34 --i32combined

此选项生成 Intel Hex-32 格式的输出。此选项为包含多个加载区的映像生成一个输出文件。可以使用 --base 选项指定输出的基址。

**另请参阅**

- 第2-7 页的*--base [[object_file::]load_region_ID=]num*
- 第1-3 页的*使用 fromelf 时的注意事项*

### 2.2.35 --ignore_section=*option*[,*option*,...]

此选项指定要在比较期间忽略的节。如果被比较的文件之间的差异位于这些节中，则忽略这些差异。

#### 限制

此选项必须与 --compare 一起使用。

#### 语法

--ignore_section=*option*[,*option*,...]

其中 *option* 为下列项之一：

*object_name*::

名称与 *object_name* 匹配的 ELF 对象中的所有节。

*object_name*::*section_name*

名称与 *object_name* 匹配的 ELF 对象中节名称与 section_name 匹配的所有节。

*section_name* 名称与 *section_name* 匹配的所有节。

通配符 ? 和 * 可用于 *section_name* 和 *object_name* 参数中的符号名称。可在一个 --ignore_section 选项后面跟随逗号分隔的参数列表，从而指定多个选项。

#### 另请参阅

- 第2-13 页的*--compare=option[,option,...]*
- *--ignore_symbol=option[,option,...]*
- 第2-38 页的*--relax_section=option[,option,...]*

### 2.2.36 --ignore_symbol=*option*[,*option*,...]

此选项指定要在比较期间忽略的符号。如果被比较的文件之间的差异与这些符号相关，则忽略这些差异。

#### 限制

此选项必须与 --compare 一起使用。

### 语法

--ignore_symbol=*option*[,*option*,...]

其中 *option* 为下列项之一：

*object_name*::

名称与 *object_name* 匹配的 ELF 对象中的所有符号。

*object_name*::*symbol_name*

名称与 *object_name* 匹配的 ELF 对象中符号名称与 symbol_name 匹配的所有符号。

*symbol_name* 名称与 *symbol_name* 匹配的所有符号。

通配符 ? 和 * 可用于 *symbol_name* 和 *object_name* 参数中的符号名称。可在一个 --ignore_symbol 选项后面跟随逗号分隔的参数列表，从而指定多个选项。

### 另请参阅

- 第2-13 页的*--compare=option[,option,...]*
- 第2-28 页的*--ignore_section=option[,option,...]*
- 第2-39 页的*--relax_symbol=option[,option,...]*

## 2.2.37 --in_place

此选项启用输入文件中 ELF 成员的转换以覆盖以前的内容。

### 限制

此选项必须与 --elf 一起使用。

### 示例

要从库文件成员中删除调试信息，请输入：

```
fromelf --elf --in_place --strip=debug test.a
```

### 另请参阅

- 第2-19 页的*--elf*
- 第2-43 页的*--strip=option[,option,...]*

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

此选项输出有关特定主题的信息。

#### 限制

此选项只能用在文本模式中。

#### 语法

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

其中 *topic* 是一个逗号分隔列表，其中包含选自下列主题关键字的关键字：

instruction_usage
: 对每个输入文件的代码节中定义的 ARM 和 Thumb 指令进行分类，并将其列出。

function_sizes
: 列出一个或多个输入文件中定义的全局函数的名称及大小 （以字节为单位）。

function_sizes_all
: 列出一个或多个输入文件中定义的局部和全局函数的名称及大小（以字节为单位）。

sizes
: 列出映像中的每个输入对象和库成员的 Code、RO Data、RW Data、ZI Data 和 Debug 大小。使用此选项相当于 --info=sizes,totals。

totals
: 列出输入对象和库的 Code、RO Data、RW Data、ZI Data 和 Debug 的总大小。

--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。

**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2-44 页的*--text*

### 2.2.39 *input_file*

此选项指定要处理的 ELF 文件或库文件。在下列条件下，支持使用多个输入文件：

- 输出 --text 格式
- 使用 --compare 选项
- 将 --elf 与 --in_place 一起使用
- 使用 --output 指定输出目录。

如果 *input_file* 是包含多个加载区的分散加载映像，并且输出格式为 --bin、--m32、--i32 或 --vhx 之一，则 fromelf 会为每个加载区创建一个单独的文件。

如果 *input_file* 是包含多个加载区的分散加载映像，并且输出格式为 --m32combined 或 --i32combined，则 fromelf 会创建一个包含所有加载区的文件。

#### 另请参阅

- 第2-8 页的*--bin*
- 第2-13 页的*--compare=option[,option,...]*
- 第2-19 页的*--elf*
- 第2-27 页的*--i32*
- 第2-27 页的*--i32combined*
- 第2-29 页的*--in_place*
- 第2-34 页的*--m32*
- 第2-35 页的*--m32combined*
- 第2-35 页的*--output=destination*
- 第2-44 页的*--text*
- 第2-49 页的*--vhx*

### 2.2.40 --interleave=*option*

如果存在调试信息，则此选项将源代码作为注释插入反汇编中。

此选项与 --emit=code、--text -c 或 --disassemble 一起使用。

#### 语法

--interleave=*option*

其中 *option* 可以为下列选项之一：

`line_directives`
: 交叉存取包含反汇编指令的文件名和行号的 `#line` 指令。

`line_numbers`
: 交叉存取包含反汇编指令的文件名和行号的注释。

`none`
: 禁用交叉存取。如果您有一个已生成的 makefile 文件，其中的 fromelf 命令除了 `--interleave` 之外还有多个选项，则此选项十分有用。这样，可以将 `--interleave=none` 指定为最后一个选项，从而无需重新生成完整的 fromelf 命令就可确保禁用交叉存取。

`source`
: 交叉存取包含源代码的注释。如果源代码不再可用，则 `fromelf` 的交叉存取方式与 `line_numbers` 相同。

`source_only`
: 交叉存取包含源代码的注释。如果源代码不再可用，则 `fromelf` 不会对该代码进行交叉存取。

#### 缺省选项

缺省为 `--interleave=none`。

#### 另请参阅

- 第2-18 页的*--disassemble*
- 第2-19 页的*--emit=option[,option,...]*
- 第2-42 页的*--source_directory=path*
- 第2-44 页的*--text*

### 2.2.41 --licretry

如果您使用的是浮动许可证，则在调用 `fromelf` 时此选项最多进行 10 次获取许可证的尝试。

#### 用法

建议将此选项放入 `RVCT40_FROMELFOPT` 环境变量中。这样便无需修改生成文件。

**注意**

只有在解决了与网络或许可证服务器设置有关的所有其他问题后，才能使用此选项。

### 另请参阅

- 《编译器参考指南》中第2-72 页的*--licretry*
- 《链接器参考指南》中第2-35 页的*--licretry*
- 《汇编器指南》中第3-2 页的*命令语法*
- 《RealView 编译工具要点指南》中第1-6 页的*RVCT 使用的环境变量*
- 《ARM 工具 FLEXnet 许可证管理指南》

## 2.2.42 --[no_]linkview

此选项丢弃 ELF 映像中节级别的视图，只保留段级别的视图 （加载时视图）。

丢失链接视图的节级别时将删除：

- 节的头文件表
- 节的头文件字符串表
- 字符串表
- 符号表
- 所有调试节。

输出中只保留程序的头文件表和程序段。根据 《System V 应用程序二进制接口》(*System V Application Binary Interface*) 规范，程序装入程序在 ELF 文件中能够依赖的就是这些内容。

### 限制

如果使用 --[no_]linkview，则：

- 必须将此选项与 --elf 一起使用。
- 不要将 --no_linkview 选项用于 SysV 映像。

### 示例

要获取 ELF 格式的输出，请输入：

```
fromelf --no_linkview --elf image.axf --output=image_nlk.axf
```

### 另请参阅

- 第2-19 页的*--elf*
- 第2-43 页的*--strip=option[,option,...]*

- 《System V 应用程序二进制接口 – 草案 – 2003 年 12 月 17 日》(*System V Application Binary Interface – DRAFT – 17 December 2003*) 规范。

### 2.2.43 --localize=*option*[,*option*,...]

此选项将所选符号转换为局部符号。

#### 限制

此选项必须与 --elf 一起使用。

#### 语法

--localize=*option*[,*option*,...]

其中 *option* 为下列项之一：

*object_name*::

名称与 *object_name* 匹配的 ELF 对象中的所有符号都转换为局部符号。

*object_name*::*symbol_name*

名称与 *object_name* 匹配的 ELF 对象中符号名称与 symbol_name 匹配的所有符号都转换为局部符号。

*symbol_name* 符号名称与 *symbol_name* 匹配的所有符号都转换为局部符号。

通配符 ? 和 * 可用于 *symbol_name* 和 *object_name* 参数中的符号名称。可在一个 --localize 选项后面跟随逗号分隔的参数列表，从而指定多个选项。

#### 另请参阅

- 第2-19 页的*--elf*
- 第2-25 页的*--hide=option[,option,...]*

### 2.2.44 --m32

此选项生成 Motorola 32 位格式（32 位 S-record）的输出。它为映像中的每个加载区生成一个输出文件。可以使用 --base 选项指定输出的基址。

#### 另请参阅

- 第2-7 页的*--base [[object_file::]load_region_ID=]num*

- 第1-3 页的 *使用 fromelf 时的注意事项*

### 2.2.45 --m32combined

此选项生成 Motorola 32 位格式（32 位 S-record）的输出。此选项为包含多个加载区的映像生成一个输出文件。可以使用 --base 选项指定输出的基址。

#### 另请参阅

- 第2-7 页的 *--base [[object_file::]load_region_ID=]num*
- 第1-3 页的 *使用 fromelf 时的注意事项*

### 2.2.46 --output=*destination*

此选项指定输出文件的名称，或在创建多个输出文件时指定输出目录的名称。

如果将此选项与 --bin 或 --elf 一起使用，则：

- 可指定一个输入文件名和一个输出文件名。
- 如果指定多个输入文件并使用 --elf，则可以使用 --in_place 将每个输入文件的处理输出写入该文件的最上方。
- 如果指定许多个输入文件名并指定一个输出目录，则每个文件的处理输出将写入该输出目录中。每个输出文件名都派生自相应的输入文件。因此，以这种方式指定输出目录是在单次运行 fromelf 时将许多 ELF 文件转换为二进制或十六进制格式的唯一方法。

#### 语法

--output=*destination*

其中 *destination* 可以是文件或目录。例如：

--output=foo 是输出文件的名称

--output=foo/

是输出目录的名称。

#### 另请参阅

- 第2-19 页的 *--elf*
- 第2-44 页的 *--text*

### 2.2.47 --privacy

此选项将节名称更改为缺省值，并用与 --strip symbols 相同的方法去除符号表。此外，除映射和生成属性符号之外，所有局部符号均失去其名称。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**。

#### 另请参阅

- 第2-43 页的*--strip=option[,option,...]*
- 《链接器参考指南》中第2-43 页的*--privacy*

### 2.2.48 --[no_]project=*filename*

控制指定项目模板文件的加载。

#### 语法

--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。

#### 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="fromelf">
        <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：

```
fromelf foo.o
```

将生成以下一行实际的命令：

```
fromelf --cpu=ARM926EJ-S --fpu=vfpv2 foo.o
```

**另请参阅**

- *--reinitialize_workdir*
- 第2-48 页的*--workdir=directory*

### 2.2.49 --qualify

此选项修改 --fieldoffsets 选项的功能，以使每个输出符号的名称都包含对含有相关结构的源文件的指示。这使 --fieldoffsets 选项可生成有效输出，即使两个源文件定义了同名的不同结构也如此。

**示例**

在两个文件头（例如，one.h 和 two.h）中定义了一个名为 foo 的结构。

使用 fromelf --fieldoffsets 时，链接器可能会定义下列符号：

- foo.a、foo.b 和 foo.c
- foo.x、foo.y 和 foo.z

使用 fromelf --qualify --fieldoffsets 时，链接器会定义下列符号：

- oneh_foo.a、oneh_foo.b 和 oneh_foo.c
- twoh_foo.x、twoh_foo.y 和 twoh_foo.z

**另请参阅**

- 第2-22 页的*--fieldoffsets*

### 2.2.50 --reinitialize_workdir

此选项使您可以重新初始化使用 --workdir 设置的项目模板工作目录。

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 一起使用。

#### 另请参阅

- 第2-36 页的*--[no_]project=filename*
- 第2-48 页的*--workdir=directory*

### 2.2.51 --relax_section=*option*[,*option*,...]

此选项将指定节在比较报告中的严重性更改为警告而不是错误。

#### 限制

此选项必须与 --compare 一起使用。

#### 语法

--relax_section=*option*[,*option*,...]

其中 *option* 为下列项之一：

*object_name*::

名称与 *object_name* 匹配的 ELF 对象中的所有节。

*object_name*::*section_name*

名称与 *object_name* 匹配的 ELF 对象中节名称与 section_name 匹配的所有节。

*section_name* 名称与 *section_name* 匹配的所有节。

通配符 ? 和 * 可用于 *section_name* 和 *object_name* 参数中的符号名称。可在一个 --relax_section 选项后面跟随逗号分隔的参数列表，从而指定多个选项。

#### 另请参阅

- 第2-13 页的*--compare=option[,option,...]*
- 第2-28 页的*--ignore_section=option[,option,...]*

- *--relax_symbol=option[,option,...]*

### 2.2.52 --relax_symbol=*option*[,*option*,...]

此选项将指定符号在比较报告中的严重性更改为警告而不是错误。

#### 限制

此选项必须与 --compare 一起使用。

#### 语法

--relax_symbol=*option*[,*option*,...]

其中 *option* 为下列项之一：

*object_name*::

名称与 *object_name* 匹配的 ELF 对象中的所有符号。

*object_name*::*section_name*

名称与 *object_name* 匹配的 ELF 对象中符号名称与 symbol_name 匹配的所有符号。

*symbol_name* 名称与 *symbol_name* 匹配的所有符号。

通配符 ? 和 * 可用于 *symbol_name* 和 *object_name* 参数中的符号名称。可在一个 --relax_symbol 选项后面跟随逗号分隔的参数列表，从而指定多个选项。

#### 另请参阅

- 第2-13 页的*--compare=option[,option,...]*
- 第2-28 页的*--ignore_symbol=option[,option,...]*
- 第2-38 页的*--relax_section=option[,option,...]*

### 2.2.53 --rename=*option*[,*option*,...]

此选项重命名输出 ELF 对象中的指定符号。

#### 限制

此选项必须与 --elf 和 --output 一起使用。

### 语法

--rename=*option*[,*option*,...]

其中 *option* 为下列项之一：

*object_name*::*old_symbol_name=new_symbol_name*

这将替换 ELF 对象 *object_name* 中符号名称与 old_symbol_name 匹配的所有符号。

*old_symbol_name=new_symbol_name*

这将替换符号名称与 *old_symbol_name* 匹配的所有符号。

通配符 ? 和 * 可用于 *old_symbol_name*、new_symbol_name 和 *object_name* 参数中的符号名称。可在一个 --rename 选项后面跟随逗号分隔的参数列表，从而指定多个选项。

### 示例

此示例将 dhrystone.axf 映像中的 clock 符号重命名为 myclock，并创建一个名为 dhry.axf 的新文件：

```
fromelf --elf --rename=clock=myclock --output=dhry.axf dhrystone.axf
```

### 另请参阅

- 第2-19 页的*--elf*
- 第2-35 页的*--output=destination*

## 2.2.54 --select=*select_options*

此选项只选择与指定模式列表匹配的字段。

此选项与 --fieldoffsets 或 --text -a 一起使用。

### 语法

--select=*select_options*

其中 *select_options* 是要匹配的模式列表。使用特殊字符可以选择多个字段：

- 使用逗号分隔列表可将各选项组合在一起，例如：

  a*,b*,c*

- 使用通配符 * 可匹配任意名称。

- 使用通配符 ? 可匹配任意单个字母。
- 通过在 *select_options* 字符串前面添加 + 前缀可指定要包括的字段。这是缺省行为。
- 通过在 *select_options* 字符串前面添加 ~ 前缀可指定要排除的字段。

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。

**另请参阅**

- 第2-22 页的*--fieldoffsets*
- 第2-44 页的*--text*

### 2.2.55 --show=*option*[,*option*,...]

此选项更改所选符号的符号可见性属性，以用缺省可见性标记这些符号。

**限制**

此选项必须与 --elf 一起使用。

**语法**

--show=*option*[,*option*,...]

其中 *option* 为下列项之一：

*object_name*::

名称与 *object_name* 匹配的 ELF 对象中的所有符号都标记为具有缺省可见性。

*object_name*::*symbol_name*

名称与 *object_name* 匹配的 ELF 对象中符号名称与 symbol_name 匹配的所有符号都标记为具有缺省可见性。

*symbol_name* 符号名称与 *symbol_name* 匹配的所有符号都标记为具有缺省可见性。

通配符 ? 和 * 可用于 *symbol_name* 和 *object_name* 参数中的符号名称。可在一个 --show 选项后面跟随逗号分隔的参数列表，从而指定多个选项。

**另请参阅**

- 第2-19 页的*--elf*

- 第2-25 页的*--hide=option[,option,...]*

### 2.2.56 --show_and_globalize=*option*[,*option*,...]

此选项更改所选符号的符号可见性属性以用缺省可见性标记这些符号，并将这些符号转换为全局符号。

#### 限制

此选项必须与 --elf 一起使用。

#### 语法

--show_and_globalize=*option*[,*option*,...]

其中 *option* 为下列项之一：

*object_name*::

名称与 *object_name* 匹配的 ELF 对象中的所有符号。

*object_name*::*symbol_name*

名称与 *object_name* 匹配的 ELF 对象中符号名称与 symbol_name 匹配的所有符号。

*symbol_name* 符号名称与 *symbol_name* 匹配的所有符号。

通配符 ? 和 * 可用于 *symbol_name* 和 *object_name* 参数中的符号名称。可在一个 --show_and_globalize 选项后面跟随逗号分隔的参数列表，从而指定多个选项。

#### 另请参阅

- 第2-19 页的*--elf*

### 2.2.57 --show_cmdline

此选项显示 fromelf 是如何处理命令行的。命令以标准化方式显示，并展开所有 via 文件的内容。

### 2.2.58 --source_directory=*path*

此选项显式指定源代码的目录。缺省情况下，假定源代码位于相对于 ELF 输入文件的目录中。多次使用此选项可指定涉及多个目录的搜索路径。

此选项与 --interleave 一起使用。

**另请参阅**

- 第2-31 页的*--interleave=option*

### 2.2.59 --strip=*option*[,*option*,...]

此选项保护传递到第三方的映像和对象中的 IP。它还有助于减小输出映像的大小。

**限制**

此选项必须与 --elf 一起使用。

**语法**

--strip=*option*[,*option*,...]

其中 *option* 为下列项之一：

all 对于对象模块，此选项从 ELF 文件中删除所有调试、注释、说明和符号。对于可执行文件，此选项与 --no_linkview 的作用相同。

**注意**

不要将 --strip=all 选项用于 SysV 映像。

debug 从 ELF 文件中删除所有调试节。

comment 从 ELF 文件中删除 **.comment** 节。

filesymbols 从 ELF 中删除 STT_FILE 符号。

localsymbols 从 ELF 文件中删除所有不用作重定位目标的局部符号。在所有情况下均会删除 STT_FILE 符号。

notes 从 ELF 文件中删除 **.notes** 节。

pathnames 从类型为 STT_FILE 的所有符号中删除路径信息。例如，名为 C:\work\myobject.o 的 STT_FILE 符号将被重命名为 myobject.o。

symbols 对于对象，此选项从不用作重定位目标的 ELF 文件中删除所有局部符号。

对于可执行文件，此选项删除所有静态符号。如果将其中任意静态符号用作静态重定位目标，则也会删除这些重定位。在所有情况下均会删除 STT_FILE 符号。

**注意**

去除符号、路径名或文件符号可能会增加文件的调试难度。

#### 示例

要生成一个新的输出文件，该文件等同于从 ELF 文件 （最初是使用 --debug 链接器选项生成的）使用 --no_debug 链接器选项生成，请使用：

```
fromelf --strip=debug --elf --output=outfile.axf infile.axf
```

#### 另请参阅

- 第2-19 页的*--elf*
- 第2-33 页的*--[no_]linkview*

### 2.2.60 --[no_]symbolversions

此选项关闭符号版本表的解码功能。有关详细信息，请参阅第4-15 页的*符号版本控制*和 《ARM 体系结构的基础平台 ABI》 (BPABI)。

### 2.2.61 --text

此选项以文本格式输出映像信息。使用此选项可以对 ELF 映像或 ELF 对象文件进行解码。这是缺省值，即如果没有指定代码输出格式，则假定使用 --text。

如果未使用 --output 选项指定 *destination* 或未指定 --output，则在 stdout 上显示信息。

使用以下一个或多个选项指定显示的内容：

-a 输出全局和静态数据地址 （包括结构和联合内容的地址）。

此选项只能用于包含调试信息的文件。如果不存在调试信息，则会显示以下警告：

```
Warning: Q0186E: This option requires debugging information to be present
```

使用 --select 选项可以输出数据地址的子集。

如果要查看在结构内外展开的数组的数据地址，请使用带此文本类别的 --expandarrays 选项。

-c 此选项反汇编代码，并转储正被反汇编的原始二进制数据及指令地址。

**注意**

与 --disassemble 不同的是，不能将反汇编输入到 ARM 汇编器。

-d 输出数据节的内容。

-e 解码对象的异常表信息。对映像进行反汇编时与 -c 结合使用。

-g 输出调试信息。

-r 输出重定位信息。

-s 输出符号表和版本控制表。

-t 输出字符串表。

-v 输出映像的每个段和节的头文件的详细信息。

-w 删除换行。

-y 输出动态段内容。

-z 输出代码和数据大小。

仅在文本模式下识别这些选项。

### 示例

下面的示例显示如何使用 --text：

- 要生成包含 ELF 映像的反汇编版本以及符号表的纯文本输出文件，请使用：
  ```
  fromelf --text -c -s --output=outfile.lst infile.axf
  ```
- 要在 stdout 中列出所有全局和静态数据变量以及所有结构字段地址，请使用：
  ```
  fromelf --text -a --select=* infile.axf
  ```
- 要生成一个文本文件，其中包含 inputfile.axf 中的所有结构地址，但不包含任何全局或静态数据变量信息，请使用：
  ```
  fromelf --text -a --select=*.* --output=structaddress.txt infile.axf
  ```

- 要生成仅包含嵌套结构的地址的文本文件，请使用：
  ```
  fromelf --text -a --select=*.*.* --output=structaddress.txt infile.axf
  ```
- 要生成一个文本文件，其中包含 inputfile.axf 中的所有全局或静态数据变量信息，但不包含任何结构地址，请使用：
  ```
  fromelf --text -a --select=*,~*.* --output=structaddress.txt infile.axf
  ```

**另请参阅**


- 第2-14 页的*--cpu=name*
- 第2-18 页的*--disassemble*
- 第2-19 页的*--emit=option[,option,...]*
- 第2-22 页的*--expandarrays*
- 第2-30 页的*--info=topic[,topic,...]*
- 第2-31 页的*--interleave=option*
- 第2-35 页的*--output=destination*
- 第2-40 页的*--select=select_options*
- 第3-29 页的*获得有关映像的信息*


### 2.2.62 *--width*x*banks*

此选项为多个存储体输出多个文件。

如果指定了多个配置，则 fromelf 使用最后指定的配置。

**限制**

此选项必须与 --output 一起使用。

**语法**

*--width*x*banks*

其中：

*banks* 指定目标内存系统中存储体的数量。它确定为每个加载区生成的输出文件的数量。

*width* 是目标内存系统中的内存宽度 （8 位、 16 位、 32 位或 64 位）。

有效配置有：

```
--8x1
--8x2
--8x4
--16x1
--16x2
--32x1
--32x2
--64x1
```

如果映像有一个加载区，则 fromelf 生成与指定的 *banks* 数量相同的文件。文件名从 --output=*destination* 参数派生，派生时采用下列命名约定：

- 如果只有一个存储体 (*banks*=1)，则输出文件名为 *destination*。
- 如果有多个存储体 (*banks*>1)，则 fromelf 生成 *banks* 个名为 *destinationN* 文件，其中 *N* 介于 0 到 *banks*-1 的范围内。如果为输出文件名指定文件扩展名，则将数字 *N* 放在文件扩展名的前面。例如：

  ```
  fromelf --vhx --8x2 test.axf --output=test.txt
  ```

  此示例生成名为 test0.txt 和 test1.txt 的两个文件。

如果映像有多个加载区，则 fromelf 创建一个名为 *destination* 的目录，并为该目录中的每个加载区生成 *banks* 文件。每个加载区对应的文件命名为 *load_regionN*，其中 *load_region* 是加载区的名称，而 *N* 介于 0 到 *banks*-1 的范围内。例如：

```
fromelf --vhx --8x2 multiload.axf --output=regions
```

此示例可能在 regions 目录中生成下列文件：

```
EXEC_ROM0
EXEC_ROM1
RAM0
RAM1
```

由 *width* 指定的内存宽度控制在每个输出文件的单行中存储的内存量。每个输出文件的大小等于要读取的内存大小除以创建的文件数。例如：

- fromelf --vhx --8x4 test.axf --output=file 生成四个文件 （file0、file1、file2 和 file3）。每个文件包含单字节行，例如：

  ```
  00
  00
  2D
  00
  2C
  8F
  ...
  ```

- `fromelf --vhx --16x2 test.axf --output=file` 生成两个件（`file0` 和 `file1`）。每个文件都包含双字节的行，例如：

```
0000
002D
002C
...
```

**另请参阅**

- 第2-35 页的*--output=destination*
- 第2-49 页的*--vhx*

### 2.2.63 --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。

**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。

**错误**

如果在必需 `--workdir` 时尝试使用不带 `--workdir` 的 `--project`，则会生成错误消息。

**另请参阅**

- 第2-36 页的*--[no_]project=filename*
- 第2-37 页的*--reinitialize_workdir*

### 2.2.64 --vhx

此选项生成面向字节 （Verilog 内存模型）的十六进制格式输出。此格式适合加载到*硬件描述语言* (HDL) 仿真器的内存模型中。可以使用 *--widthxbanks* 选项将此选项的输出分为多个文件。

**另请参阅**

- 第2-46 页的*--widthxbanks*
- 第1-3 页的*使用 fromelf 时的注意事项*

### 2.2.65 --vsn

此选项显示 fromelf 版本信息。

# 第 3 章
# 使用 armar

本章介绍随 ARM RealView® 编译工具提供的 ARM® 库管理程序 `armar`。本章内容如下：

## 3.1 使用命令行选项

指定命令行选项时，根据选项类型应用下列规则：

**单字母选项**

所有单字母选项都以单短划线 - 开头。如果选项有一个参数，则可以在选项与该参数之间使用空格，或者该参数可以紧跟选项之后。

例如：

```
-a obj2.o
-aobj2.o
```

**关键字选项**

所有关键字选项都以双短划线 -- 开头。选项和参数之间需要用 = 或空格字符分隔。

例如：

```
--diag_style=ide
--diag_style ide
```

### 3.1.1 调用 armar

armar 命令行语法如下：

armar [*diagnostic-options*][*help-options*] [*input-options*] [*managing-options*] [*order-options*] [*output-options*][*project-template-options*] [*updating-options*] *archive* [*file_list*]

*diagnostic-options*

使用以下选项可控制诊断消息：

- 第3-6 页的*-c*
- 第3-7 页的*--diag_style={arm|ide|gnu}*
- 第3-12 页的*--show_cmdline*

*help-options*

使用以下选项可显示库管理程序的主要命令行选项和版本号：

- 第3-8 页的*--help*
- 第3-14 页的*--vsn*

*input-options*

使用以下选项可从文件获取命令行参数：

- 第3-14 页的*--via=file*

*managing-options*

使用以下选项可提供库入口点、文件名和目录信息：

- 第3-7 页的*--entries*
- 第3-8 页的*file_list*
- 第3-5 页的*archive*

*order-options*

使用以下选项可控制库文件的放置：

- 第3-5 页的*-a pos_name*
- 第3-6 页的*-b pos_name*
- 第3-8 页的*-i pos_name*
- 第3-9 页的*-m pos_name*

*output-options*

使用以下选项可控制库管理程序输出：

- 第3-9 页的*-n*
- 第3-9 页的*-p*
- 第3-12 页的*-s*
- 第3-12 页的*--sizes*
- 第3-13 页的*-t*
- 第3-14 页的*-v*
- 第3-15 页的*-x*
- 第3-15 页的*--zs*
- 第3-16 页的*--zt*

*project-template-options*

使用以下选项可控制项目模板的使用：

- 第3-10 页的*--[no_]project=filename*
- 第3-11 页的*--reinitialize_workdir*
- 第3-14 页的*--workdir=directory*

*updating-options*

使用以下选项可创建、更新和删除库文件：

- 第3-6 页的*-C*
- 第3-6 页的*--create*
- 第3-6 页的*-d*
- 第3-11 页的*-r*
- 第3-13 页的*-T*
- 第3-13 页的*-u*

*archive* 库的文件名，在第3-5 页的*archive* 中进行了介绍。必须总是指定库文件。

*file_list* 要处理的文件的列表，在第3-8 页的*file_list* 中进行了介绍。

### 3.1.2 排序命令行选项

通常，命令行选项可以按任意顺序出现。不过，一些选项的效果取决于它们如何与其他相关选项组合。

如果同一命令行中的某些选项覆盖了其他选项，则靠近命令行末尾的选项优先。如果某个选项不遵循此规则，则会在该选项的说明中注明。使用 `--show_cmdline` 选项可查看 armar 如何处理命令行。命令以标准化方式显示，并展开所有 via 文件的内容。

**另请参阅**

- 第3-12 页的*--show_cmdline*
- 第3-14 页的*--via=file*

## 3.2 命令行选项

本节按字母顺序列出了库管理程序所支持的命令行选项。

**注意**

在 RealView 编译工具 v3.0 及更高版本中，armar 命令行选项必须以 - 开头。这与早期版本的 armar 以及某些第三方档案管理程序不同。

**注意**

在某些 armar 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。

例如，应输入 '*,~*.*' 而不是 *,~*.*。

### 3.2.1 *archive*

指定要创建、修改或读取的库的位置。

**注意**

如果在 *file_list* 中包含了文件列表，则必须在库文件之后指定这些文件。

### 3.2.2 -a *pos_name*

将新文件放置在库中 *pos_name* 文件的后面。如果在同一命令行上使用 -b （或 -i），则否定此选项的作用。

#### 示例

若要在 mylib 中紧靠 obj2.o 之后的位置上添加或替换文件，请使用：

```
armar -r -a obj2.o mylib obj3.o obj4.o ...
```

#### 另请参阅

- 第3-6 页的 *-b pos_name*
- 第3-8 页的 *-i pos_name*

- 第3-9 页的*-m pos_name*
- 第3-11 页的*-r*

### 3.2.3 -b *pos_name*

将新文件放置在库中 *pos_name* 文件的前面。如果在同一命令行上使用 -a，则此选项优先。

**另请参阅**

- 第3-5 页的*-a pos_name*
- 第3-8 页的*-i pos_name*
- 第3-9 页的*-m pos_name*
- 第3-11 页的*-r*

### 3.2.4 -c

禁止产生在创建库时通常写入 stderr 的诊断消息。

### 3.2.5 -C

指示库管理程序在执行提取时，不用名称相似的文件替换现有文件。当同时使用 -T 禁止用截断的文件名替换具有相同前缀的文件时，此选项很有用。

**另请参阅**

- 第3-13 页的*-T*

### 3.2.6 --create

创建一个新库，该库仅包含 *file_list* 中指定的文件。如果该库已存在，则丢弃其中已有的内容。

**示例**

若要通过添加当前目录中的所有对象文件来创建新库，请使用：

```
armar --create mylib *.o
```

### 3.2.7 -d

从库中删除在 *file_list* 中指定的一个或多个文件。

### 3.2.8 --diag_style={arm|ide|gnu}

指定用于显示诊断消息的样式。

#### 语法

--diag_style=*string*

其中 *string* 是下列项之一：

arm　　采用 ARM 样式显示消息。

ide　　包含任意出错行的行号和字符计数。这些值将显示在括号中。

gnu　　以 GNU 所使用的格式显示消息。

#### 缺省选项

如果未指定 --diag_style 选项，则采用 --diag_style=arm。

### 3.2.9 --entries

列出库中具有使用汇编器 ENTRY 指令定义的入口点的所有对象文件。

列表的格式为：

ENTRY at offset *num* in section *name* of *member*

#### 示例

下面的示例列出 myasm.a 中每个对象文件的入口点：

```
> armar --entries myasm.a
ENTRY at offset 0 in section adrlabel of adrlabel.o
ENTRY at offset 0 in section ARMex of armex.o
ENTRY at offset 0 in section Block of blocks.o
ENTRY at offset 0 in section Jump of jump.o
ENTRY at offset 0 in section LDRlabel of ldrlabel.o
ENTRY at offset 0 in section Loadcon of loadcon.o
ENTRY at offset 0 in section StrCopy of strcopy.o
ENTRY at offset 0 in section subrout of subrout.o
ENTRY at offset 0 in section Tblock of tblock.o
ENTRY at offset 0 in section ThumbSub of thumbsub.o
ENTRY at offset 0 in section Word of word.o
```

**另请参阅**

- 第3-12 页的*--sizes*
- 第3-16 页的*--zt*
- 《汇编器指南》中第7-69 页的*ENTRY*：

### 3.2.10 *file_list*

要处理文件的列表。每个文件由其路径和名称完全指定。路径可以是绝对路径、相对于驱动器和根的路径，或是相对于当前目录的路径。

**注意**

必须在库文件的后面指定文件列表。

当与库中的文件名进行比较时，仅使用路径末尾的文件名。如果两个或多个路径操作数以相同的文件名结束，则结果将无法预料。可以使用通配符 * 和 ? 来指定文件。

如果其中一个文件是库，则 armar 将输入库中的所有成员复制到目标库。命令行中的成员顺序会保留。因此，提供库文件在逻辑上相当于按成员在库中存储的顺序来提供库的所有成员。

### 3.2.11 --help

汇总显示主要的命令行选项。

如果未指定任何选项或源文件，则它是缺省选项。

**另请参阅**

- 第3-14 页的*--vsn*

### 3.2.12 -i *pos_name*

将新文件放置在库中 *pos_name* 成员的前面 （等效于 -b *pos_name*）。

**另请参阅**

- 第3-5 页的*-a pos_name*
- 第3-6 页的*-b pos_name*
- 第3-9 页的*-m pos_name*
- 第3-11 页的*-r*

### 3.2.13 -m *pos_name*

移动文件。如果与 *pos_name* 一起指定了 -a、-b 或 -i，则文件将移动到新位置。否则，文件将移到库的末尾。

**另请参阅**

- 第3-5 页的*-a pos_name*
- 第3-6 页的*-b pos_name*
- 第3-8 页的*-i pos_name*

### 3.2.14 -n

禁止在库中创建符号表。

**注意**

缺省情况下，当您创建对象文件的库时，armar 始终会创建符号表。

可以使用 -s 选项在库中重新创建符号表。

**示例**

若要创建不含符号表的库，请使用：

```
armar -n --create mylib.a *.obj
```

**另请参阅**

- 第3-12 页的*-s*

### 3.2.15 --new_files_only

更新旧文件。与 -r 选项结合使用时，只有当相应文件的修改时间晚于库中文件的修改时间时，才替换库中的文件。

**另请参阅**

- 第3-11 页的*-r*
- 第3-13 页的*-u*

### 3.2.16 -p

将 *library* 中的文件的内容输出到 stdout。

### 3.2.17 --[no_]project=*filename*

指示编译器加载指定的项目模板文件。

#### 语法

--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。

#### 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="armar">
        <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：

```
armar mylib foo.o
```

将生成以下一行实际的命令：

```
armar --cpu=ARM926EJ-S --fpu=vfpv2 mylib foo.o
```

#### 另请参阅

- 第3-11 页的*--reinitialize_workdir*

- 第3-14 页的*--workdir=directory*

### 3.2.18 -r

在库中替换或添加文件。如果库不存在，则创建一个新的库文件并将诊断消息写入标准错误。-q 是 -r 的别名。

如果未指定 *file_list* 而库存在，则结果无法预料。替换现有文件的文件不会更改库的顺序。

如果使用 -u 选项，则只替换修改日期晚于库文件的那些文件。

如果使用 -a、-b 或 -i 选项，则 *pos_name* 必须存在并指定是将新文件放在 *pos_name* 之后 (-a) 还是之前 （-b 或 -i）。否则新文件将放在末尾。

#### 示例

若要在库中添加或替换指定文件，请使用：

```
armar -r mylib obj1.o obj2.o obj3.o ...
```

若要在库中替换文件，且只有库中的文件比指定文件更早时才替换，请使用：

```
armar -ru mylib k*.o
```

#### 另请参阅

- 第3-5 页的*-a pos_name*
- 第3-6 页的*-b pos_name*
- 第3-8 页的*-i pos_name*
- 第3-13 页的*-u*

### 3.2.19 --reinitialize_workdir

使您可以重新初始化使用 --workdir 设置的项目模板工作目录。

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3-10 页的*--[no_]project=filename*
- 第3-14 页的*--workdir=directory*

### 3.2.20 -s

在库中创建符号表。此选项对于通过以下方式创建的库十分有用：

- 使用 -n 选项创建的库
- 使用不自动创建符号表的档案管理程序创建的库。

**注意**

缺省情况下，当您创建对象文件的库时， armar 始终会创建符号表。

**示例**

若要在使用 -n 选项创建的库中创建符号表，请使用：

```
armar -s mylib.a
```

**另请参阅**

- 第3-9 页的*-n*
- 第3-15 页的*--zs*

### 3.2.21 --show_cmdline

此选项显示 armar 是如何处理命令行的。命令以标准化方式显示，并展开所有 via 文件的内容。

### 3.2.22 --sizes

列出库中每个成员的 Code、 RO Data、 RW Data、 ZI Data 和 Debug 大小。

**示例**

下面的示例显示 mylib.a 中的 app_1.o 和 app_2.o 的大小：

```
> armar --sizes mylib.a

Code    RO Data    RW data    ZI Data    Debug   Object Name
```

```
 464          0          0          0     8612   app_1.o
3356          0          0      10244    11848   app_2.o
3820          0          0      10244    20460   TOTAL
```

**另请参阅**

- 第3-7 页的*--entries*
- 第3-16 页的*--zt*

### 3.2.23 -t

输出库的内容表。由 *file_list* 指定的文件会包含在写入列表中。如果未指定 *file_list*，则按归档顺序包含库中的所有文件。

**示例**

若要以详细信息模式列出库的内容表，请使用：

```
armar -tv mylib
```

### 3.2.24 -T

允许截断已提取文件的文件名 （如果其库名长度超过文件系统能够支持的长度）。缺省情况下，提取文件名过长的文件时会出错。系统将写入一条诊断消息，并且不提取该文件。

### 3.2.25 -u

更新旧文件。与 -r 选项结合使用时，只有当相应文件的修改时间不早于库中文件的修改时间时，才替换库中的文件。

**另请参阅**

- 第3-9 页的*--new_files_only*
- 第3-11 页的*-r*

### 3.2.26 -v

提供冗余输出。

输出随使用的其他选项而变化：

-d、-r 和 -x

写入有关库创建、组成文件和维护活动的各个文件的详细描述。

-p 将文件本身写入 stdout 之前，将文件名写入标准输出。

-t 包含有关库中文件信息的长列表。

-x 在每次提取之前输出文件名。

**另请参阅**

- 第3-6 页的*-d*
- 第3-9 页的*-p*
- 第3-11 页的*-r*
- 第3-13 页的*-t*
- 第3-15 页的*-x*

### 3.2.27 --via=*file*

指示库管理程序使用 *file* 中指定的选项。

**另请参阅**

- 《编译器参考指南》中附录 A *via 文件语法*

### 3.2.28 --vsn

将版本号输出到 stderr。

**另请参阅**

- 第3-8 页的*--help*

### 3.2.29 --workdir=*directory*

可用于为项目模板提供一个工作目录。

**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。

#### 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。

#### 错误

如果在必需 `--workdir` 时尝试使用不带 `--workdir` 的 `--project`，则会生成错误消息。

#### 另请参阅

- 第3-10 页的*--[no_]project=filename*
- 第3-11 页的*--reinitialize_workdir*

### 3.2.30 -x

从库中提取 *file_list* 中的文件。库的内容不会发生更改。如果未提供文件操作数，则提取库中的所有文件。如果从库中提取的文件的文件名比目标目录中支持的文件名长，则结果将无法预料。

### 3.2.31 --zs

显示库中所有文件的符号表。

#### 示例

若要列出库中的符号表，请使用：

```
armar --zs mylib
```

#### 另请参阅

- 第3-9 页的*-n*

- 第3-12 页的*-s*

### 3.2.32 --zt

列出库中所有文件的成员大小和入口点。有关输出格式，请参阅 --sizes 和 --entries。

**另请参阅**

- 第3-7 页的*--entries*
- 第3-12 页的*--sizes*