交叉编译详解

版本：v1.3.0

Crifan Li

摘要

本文主要介绍了什么是交叉编译，为何要有交叉编译；解释了什么是工具链，什么是交叉工具链；详解解释了交叉编译器的名字的命名规则，以及如何获得交叉编译器，制作交叉编译器的各种工具，使用已有的交叉编译器和自己手动编译交叉编译器之间的对比；最后总结了交叉编译方面的心得和注意事项。

本文提供多种格式供：

在线阅读	HTML [1]	HTMLs [2]	PDF [3]	CHM [4]	TXT [5]	RTF [6]	WEBHELP [7]
下载（7zip压缩包）	HTML [8]	HTMLs [9]	PDF [10]	CHM [11]	TXT [12]	RTF [13]	WEBHELP [14]

HTML版本的在线地址为：

http://www.crifan.com/files/doc/docbook/cross_compile/release/html/cross_compile.html

有任何意见，建议，提交bug等，都欢迎去讨论组发帖讨论：

http://www.crifan.com/bbs/categories/cross_compile/

修订历史		
修订 1.3.0	2015-05-23	crl

1. 将帖子内容整理过来
2. 添加了关于交叉编译器命名规则的解释
3. 添加了使用已有的交叉编译器和自己手动编译交叉编译器之间的对比
4. 补充完整帖子引用
5. 添加交叉编译器eldk的下载地址

[1] http://www.crifan.com/files/doc/docbook/cross_compile/release/html/cross_compile.html
[2] http://www.crifan.com/files/doc/docbook/cross_compile/release/htmls/index.html
[3] http://www.crifan.com/files/doc/docbook/cross_compile/release/pdf/cross_compile.pdf
[4] http://www.crifan.com/files/doc/docbook/cross_compile/release/chm/cross_compile.chm
[5] http://www.crifan.com/files/doc/docbook/cross_compile/release/txt/cross_compile.txt
[6] http://www.crifan.com/files/doc/docbook/cross_compile/release/rtf/cross_compile.rtf
[7] http://www.crifan.com/files/doc/docbook/cross_compile/release/webhelp/index.html
[8] http://www.crifan.com/files/doc/docbook/cross_compile/release/html/cross_compile.html.7z
[9] http://www.crifan.com/files/doc/docbook/cross_compile/release/htmls/index.html.7z
[10] http://www.crifan.com/files/doc/docbook/cross_compile/release/pdf/cross_compile.pdf.7z
[11] http://www.crifan.com/files/doc/docbook/cross_compile/release/chm/cross_compile.chm.7z
[12] http://www.crifan.com/files/doc/docbook/cross_compile/release/txt/cross_compile.txt.7z
[13] http://www.crifan.com/files/doc/docbook/cross_compile/release/rtf/cross_compile.rtf.7z
[14] http://www.crifan.com/files/doc/docbook/cross_compile/release/webhelp/cross_compile.webhelp.7z

交叉编译详解:

Crifan Li

版本：v1.3.0

出版日期 2015-05-23

[15] http://www.crifan.com/files/doc/docbook/soft_dev_basic/release/html/soft_dev_basic.html#cc_by_nc

目录

范例清单

前言

1. 本文目的

本文目的在于，介绍关于交叉编译方面的概念，以及如何获得和制作交叉编译器等等内容。

2. 待完成

有空再去自己，不用工具，手动的，从头到尾的，制作一个交叉编译器。然后再把内容整理过来。

第 1 章 交叉编译简介

相关旧帖：

【整理】交叉编译和现存的交叉编译工具 [1]

【整理】关于Toolchain,cross toolchain,cross compiler [2]

1.1. 什么是交叉编译

解释什么是交叉编译之前，先要明白一个概念：本地编译

我们之前常见的软件开发，都是属于本地编译：

在当前的PC下，x86的CPU下，直接编译出来程序，可以运行的程序（或者库文件），其可以直接在当前的环境，即x86的CPU下，当前电脑中，运行。

此时的编译，可以叫做，本地编译，即在当前目标平台下，编译出来的程序，也只是放到当前平台下，就可以运行的。

交叉编译，是一个和，本地编译，相对应的概念。

而所谓的，交叉编译，就是：

在一种平台上编译，编译出来的程序，是放到别的平台上运行

即编译的环境，和运行的环境，不一样，属于交叉的，此所谓cross

交叉编译，这个概念，主要和嵌入式开发有关。

例 1.1. 在x86平台上编译，在ARM平台上运行

一种最常见的例子就是：

在进行嵌入式开发时

手上有个嵌入式开发板，CPU是arm的

然后在x86的平台下开发，比如Ubuntu的Linux，或者是Win7

然后就需要：

在x86的平台上，（用交叉编译器）去编译你写好的程序代码

编译生成的（可执行的）程序，是放到目标开发板，arm的CPU上运行的

此所谓：在x86平台上编译，在ARM平台上运行

交叉编译，英文常写作cross compile，也有其他写法：crosscompile, cross compiling等

1.2. 为何要有交叉编译

之所以要有交叉编译，主要原因是：

[1] http://www.crifan.com/summary_cross_compile_cross_compiler_cross_toolchain/
[2] http://www.crifan.com/summary_about_toolchain_cross_compiler/

嵌入式系统中的资源太少

具体的解释就是：

交叉编译出来的程序，所要运行的目标环境中，各种资源，都相对有限，所以很难进行直接的本地编译

最常见的情况是：

在进行嵌入式开发时，目标平台，即嵌入式开发板，比如是最大主频200MHz的ARM的CPU，加上32M的RAM，加上1G的Nand Flash等等。

在如此相对比较紧张的硬件资源的前提下，在已经运行了嵌入式Linux的前提下，是没法很方便的，直接在嵌入式Linux下，去本地编译，去在ARM的CPU下，编译出来，供ARM的CPU可以运行的程序的。

因为编译，开发，都需要相对比较多的CPU，内存，硬盘等资源，而嵌入式开发上的那点资源，只够嵌入式（Linux）系统运行的，没太多剩余的资源，供你本地编译。

BusyBox中包含make等和编译开发相关的工具

对应的，等你后期熟悉了嵌入式开发，熟悉了Busybox后，

比如在Buildroot中去配置Busybox，或者单独交叉编译BusyBox时：

【记录】Ubuntu下为QEMU的arm平台交叉编译BusyBox [3]

就会看到，后来的BusyBox，功能增加后，也已经包含了一些，和编译开发相关的工具，比如make等等

而这些工具，本来的话，只是，放在PC端使用，即在x86平台下做开发的时候，在交叉编译的时候，才用到的工具，

现在，也在（BusyBox的）嵌入式环境中，支持了。

此时，如果，你在BusyBox中把相关的开发工具都选上的话，

再加上，你的目标开发板的硬件配置足够强大的话，比如CPU都是以GHz为单位，等等

加上相关的开发的库和工具都很全的话

实际上，至少理论上，也是可以在你的嵌入式Linux中，进行，有限的，甚至是很大程度上的，本地开发

即，直接在ARM的开发板上，嵌入式Linux中，直接进行嵌入式开发，进行针对ARM的本地编译

比如，编译出一个helloworld，估计还是可以的

这样，就不存在，或者说，避免了，此处所说的，交叉编译，而变成了本地编译

就相当于，之前在x86的PC端的，编译程序放在x86的CPU上运行的本地编译，

在ARM的CPU，嵌入式Linux中，也实现了

但是很明显，对于更加复杂的程序或者库，在ARM开发板上直接编译的可行性和效率，相对就很低

而且如果是本身折腾Uboot等东西，本身目标运行环境，就没有完整的（嵌入式Linux）系统的话，那么就更加没法在目标平台实现本地编译了。

[3] http://www.crifan.com/ubuntu_download_config_cross_compile_busybox_for_qemu_arm/

则还是只能进行，此处所说的，交叉编译

第 2 章 交叉工具链简介

搞懂了前面介绍的第 1 章 交叉编译简介的概念后，再来解释什么是交叉工具链。

相关旧帖：【整理】关于Toolchain,cross toolchain,cross compiler[1]

2.1. 什么是工具链

所谓的工具链，两部分的含义：

- 工具
  工具，即tool

  工具，是用来干活的

  此处要干的活，目标是为了：生成（可以运行的）程序或库文件

  而为了达成此目标，内部的执行过程和逻辑主要包含了：

  1. 编译
     编译的输入（对象）是：程序代码

     编译输出（目标）是：目标文件

     编译所需要的工具是：编译器

     编译器，常见的编译器，即为gcc

  2. 链接
     链接的输入（对象）是：（程序运行时所依赖的，或者某个库所依赖的另外一个）库（文件）

     链接的输出（目标）是：程序的可执行文件，或者是可以被别人调用的完整的库文件

     链接所需要的工具是：链接器

     链接器，即ld

  即，此处，为了将程序代码，编译成可执行文件，涉及到编译，链接（等其他步骤），要依赖到很多相关的工具，最核心的是编译器gcc，链接器ld。

  而此处，所谓的工具，主要指的就是：和程序编译链接等相关的gcc，ld等工具

binutils包含了ld等工具

实际上，上面所说的ld，只是处理操作目标文件，二进制文件的最主要的一个工具

而和操作目标等文件相关的，还有其他很多工具的：as，objcopy，strip，ar等等工具的

所以，对此，GNU官网，弄出一个binutils，即binary utils，二进制工具（包），集成了这些，和操作二进制相关的工具集合，叫做binutils

所以，之后你所见到的，常见的工具，就是那个著名的GNU Binutils[2]了。

[1] http://www.crifan.com/summary_about_toolchain_cross_compiler/
[2] http://www.gnu.org/software/binutils/

更加详细的解释，参见教程：GNU Binutils详解[3]

- 链
  链，即链条，chain

  之所以能称为链，你是说明不止一个东西，然后，按照对应的逻辑，串在一起，链在一起

  而对应的，涉及到的：

  - 不止一个东西
    指的是就是前面所说的那个工具，即：和程序编译链接等相关的gcc，binutils等工具
  - 按照对应的逻辑
    指的就是，按照程序本身编译链接的先后顺序，即：

    先编译，后链接，再进行后期其他的处理等等，比如用objcopy去操作相应的目标文件等等。

如此的，将：

和程序编译链接等相关的gcc，binutils等工具

按照先编译后链接等相关的编译程序的内在逻辑

串起来，就成了我们所说的：工具链

2.2. 什么是交叉工具链

普通所说的，工具链，指的是当前自己的本地平台的工具链。

用于，交叉编译，的工具链，就叫做交叉工具链

即，那些工具，即编译的gcc，链接的ld，以及相关的工具，用于交叉编译的，工具链，叫做交叉工具链。

交叉工具链，很明显，是用来，交叉编译，跨平台的程序所用的。

交叉工具链，和（本地）工具链类似，也是包含了很多的，对应的工具，交叉编译版本的gcc，ld，as等等。

但是，由于其中最最主要的是用于编译的gcc，所以，我们也常把：

交叉工具链，简称为交叉编译器

即：

严格意义上来说，交叉编译器，只是指的是交叉编译版本的gcc

但是实际上为了叫法上的方便，我们常说的交叉编译器，都是指的是交叉工具链

即，常说的交叉编译版本的gcc，比如arm-linux-gcc，实际上指代了，包含一系列交叉编译版本的交叉工具链（arm-linux-gcc，arm-linux-ld，arm-linux-as等等）

而此文中，后面，所说的，如无特殊指明，均用交叉编译器指代交叉工具链。

下面就对交叉编译器，进行详细的解释。

[3] http://www.crifan.com/files/doc/docbook/binutils_intro/release/html/binutils_intro.html

第 3 章 交叉编译器简介

TODO：

相关旧帖：嵌入式开发之交叉编译器[1]

3.1. 交叉编译器的名字的命名规则

在折腾嵌入式开发，用到交叉编译器的时候，常常会看到这样的名字：

arm-xscale-linux-gnueabi-gcc

arm-liunx-gnu-gcc

等等

其中，对应的交叉编译器的前缀为：

arm-xscale-linux-gnueabi-

arm-liunx-gnu-

而关于这些名字，我之前也是没注意其具体含义，或者说对于其含义也是很模糊的感觉，不是很清楚这些名字是从何而来的。

后来，经过折腾了crosstool-ng后，基本上明白了这些名字，是如何生成的。

其中，貌似此交叉编译器命名的规则，应该是通用的，至少记得是Buildroot中，好像也是这样命名的。

下面，就以crosstool-ng为例，参考我之前折腾crosstool-ng期间：

【整理】crosstool中如何设置xscale的Tuple’s vendor string(CT_TARGET_VENDOR) [2]

所了解到的内容，来解释解释，这些名字的含义。

3.1.1. 交叉编译器名字举例

此处，以编译crosstool-ng中：

【记录】重试使用最新版本1.18.0的crosstool-ng去配置和编译xscale的交叉编译器 [3]

通过ct-ng list-samples中得到的输出为例，

当做交叉编译器的名字的例子，供参考：

```
CLi@PC-CLI-1 ~/develop/crosstool-ng/crosstool-ng-1.18.0_build
$ ct-ng list-samples
Status  Sample name
[G.X]   alphaev56-unknown-linux-gnu
[G.X]   alphaev67-unknown-linux-gnu
[G.X]   arm-bare_newlib_cortex_m3_nommu-eabi
[G.X]   arm-cortex_a15-linux-gnueabi
```

[1] http://www.crifan.com/embedded_development_cross_compiler_and_toolchain/
[2] http://www.crifan.com/crosstool_tuple_vendor_string_ct_target_vendor/
[3] http://www.crifan.com/reuse_latest_version_crosstool_ng_to_config_and_compile_for_xscale/

```
[G..]   arm-cortex_a8-linux-gnueabi
[G..]   arm-davinci-linux-gnueabi
[G..]   armeb-unknown-eabi
[G.X]   armeb-unknown-linux-gnueabi
[G.X]   armeb-unknown-linux-uclibcgnueabi
[G..]   arm-unknown-eabi
[G..]   arm-unknown-linux-gnueabi
[G.X]   arm-unknown-linux-uclibcgnueabi
[G.X]   armv6-rpi-linux-gnueabi
[G.X]   avr32-unknown-none
[G..]   bfin-unknown-linux-uclibc
[G..]   i586-geode-linux-uclibc
[G.X]   i586-mingw32msvc,i686-none-linux-gnu
[G.X]   i686-nptl-linux-gnu
[G.X]   i686-unknown-mingw32
[G.X]   m68k-unknown-elf
[G.X]   m68k-unknown-uclinux-uclibc
[G.X]   mips64el-n32-linux-uclibc
[G.X]   mips64el-n64-linux-uclibc
[G.X]   mips-ar2315-linux-gnu
[G..]   mipsel-sde-elf
[G..]   mipsel-unknown-linux-gnu
[G.X]   mips-malta-linux-gnu
[G..]   mips-unknown-elf
[G.X]   mips-unknown-linux-uclibc
[G..]   powerpc-405-linux-gnu
[G.X]   powerpc64-unknown-linux-gnu
[G..]   powerpc-860-linux-gnu
[G.X]   powerpc-e300c3-linux-gnu
[G.X]   powerpc-e500v2-linux-gnuspe
[G..]   powerpc-unknown_nofpu-linux-gnu
[G..]   powerpc-unknown-linux-gnu
[G..]   powerpc-unknown-linux-uclibc
[G.X]   s390-ibm-linux-gnu
[G.X]   s390x-ibm-linux-gnu
[G..]   sh4-unknown-linux-gnu
[G..]   x86_64-unknown-linux-gnu
[G..]   x86_64-unknown-linux-uclibc
[G.X]   x86_64-unknown-mingw32
 L (Local)       : sample was found in current directory
 G (Global)      : sample was installed with crosstool-NG
 X (EXPERIMENTAL): sample may use EXPERIMENTAL features
 B (BROKEN)      : sample is currently broken
```

3.1.2. crosstool-ng中交叉编译前缀的命名规则

crosstool-ng中，交叉编译器的（前缀）的名字的命名规则是：

```
arch-vendor-kernel-system
```

对应分别是：

3.1.2.1. 交叉编译器名字中的arch部分

arch，即系统架构

表示交叉编译器，是用于哪个目标系统架构中，用于那个平台中的

即，用此交叉编译器编译出来的程序，是运行在哪种CPU上面的

arch的值，常见的有很多种，比如arm，x86，mips等等。

例 3.1. 举例：交叉编译器中的arch的值

arm-cortex_a8-linux-gnueabi中的arm

mips-ar2315-linux-gnu中的mips

powerpc-e500v2-linux-gnuspe中的powerpc

x86_64-unknown-mingw32中的x86_64

3.1.2.1.1. crosstool-ng中arch的值

crosstool-ng中，和arch对应的值，应该就是"Target options"中的"Target Architecture"的值了。

比如常见的，配置为arm的话，就是：

```
Target options
  Target Architecture (arm)  --->
```

对应的配置参数是：ARCH_arm

3.1.2.2. 交叉编译器名字中的vendor部分

vendor，即生成厂家，提供商

表示谁提供的，即谁制作出来这个交叉编译器的。

vendor的值，貌似是可以自己随便填写的。

其他常见写法，还有写成编译交叉编译器的作者的自己的名字的

比如，我叫crifan，那么就可以写成crifan，然后生成的交叉编译器，就是xxx-crifan-xxx-xxx了。

更加通用的做法，好像是：

把vendor写成，体系架构的值，比如我之前针对xscale的去配置crosstool-ng的时候，就写了个xscale。

或者写成CPU的厂家的名字，或者是开发板的名字等等。

例 3.2. 举例：交叉编译器中的vendor的值

- arm-cortex_a8-linux-gnueabi中的cortex_a8，就属于CPU的名字
- mips-ar2315-linux-gnu中的ar2315
- powerpc-e500v2-linux-gnuspe中的e500v2，也是CPU的内核名
- arm-buildroot-linux-uclibcgnueabi中的buildroot，是之前折腾Buildroot时，看到的，即Buildroot把自己视为当前自己制作出来的交叉编译器的vendor。

3.1.2.2.1. crosstool-ng中vendor的值

crosstool-ng中，和vendor对应的值，应该就是"Toolchain options"中的"Tuple's vendor string"的值了。

比如我之前配置为xscale的话，就是：

```
Toolchain options
  (xscale) Tuple's vendor string
```

对应的配置参数是：CT_TARGET_VENDOR

对应的help的解释是：

```
  ┌──────────────────────────────────────────────── Tuple's vendor string ─────────────────────────────────────────────────┐
  │ CT_TARGET_VENDOR:                                                                                                      │
  │                                                                                                                        │
  │ Vendor part of the target tuple.                                                                                       │
  │                                                                                                                        │
  │ A tuple is of the form arch-vendor-kernel-system.                                                                      │
  │ You can set the second part, vendor, to whatever you see fit.                                                          │
  │ Use a single word, or use underscores "_" to separate words.                                                           │
  │ Use neither dash nor space, as it breaks things.                                                                       │
  │                                                                                                                        │
  │ Keep the default (unknown) if you don't know better.                                                                   │
  │                                                                                                                        │
  │ Symbol: TARGET_VENDOR [=xscale]                                                                                        │
  │ Type  : string                                                                                                         │
  │ Prompt: Tuple's vendor string                                                                                          │
  │   Defined at config/toolchain.in:99                                                                                    │
  │   Location:                                                                                                            │
  │     -> Toolchain options                                                                                               │
```

3.1.2.3. 交叉编译器名字中的kernel部分

kernel，直译为，内核

其实指的是，你用此交叉编译器，编译出来的程序，所运行的目标系统

即，此交叉编译器，编译出来的程序，在什么系统中，什么环境中，运行。

而对应的环境或系统，主要有两种：

- Linux
  表示：有OS（此处主要指的是Linux）操作系统的环境

  比如，我用交叉编译器，编译一个helloworld程序，然后下载到嵌入式开发中的嵌入式Linux中运行，

  就属于，用此交叉编译器，编译出来的程序，是要运行于，带OS，即嵌入式Linux系统，环境中的

此处，简称为，有OS的目标系统：Linux

- bare-metal
  bare-metal，直译为：裸金属

  表示：无（此处主要指的是Linux）操作系统的环境，

  比如，用此交叉编译器，去编译一个Uboot，或者是其他一个小程序，是运行在，无嵌入式Linux的时候，单独运行的一个程序。

  比如，你购买的嵌入式系统开发版，常常附带一些小程序，比如点亮LED，跑马灯等程序，就是这种，运行在无OS的环境的

  此处，简称为：无OS系统的：bare-metal

关于，运行在有OS的Linux下，和，无OS的bare-metal，到底有何区别

目前，还没有完全搞懂。

但是，之前遇到一个实际的例子：

之前用比较新的一个交叉编译器去编译比较老的uboot时，会出现一个错误：

【已解决】uboot交叉编译出错：gcc/config/arm/lib1funcs.asm:1266: undefined reference to `raise'[4]

其就是和这个kernel有关：

编译的uboot，目标运行平台，不是Linux，而是裸的开发板，即Linux还没有运行呢，Uboot去运行，去初始化开发板的时候

详细的情况，见该贴中的解释。

例 3.3. 举例：交叉编译器中的kernel的值

- arm-bare_newlib_cortex_m3_nommu-eabi中的bare_newlib_cortex_m3_nommu，此处的bare，应该就是指的是bare-metal，用于运行在无OS的环境下
- powerpc-e300c3-linux-gnu中的linux
- m68k-unknown-uclinux-uclibc中的uclinux，就是指的是编译出来的程序，是运行于没有MMU的uclinux下

3.1.2.3.1. crosstool-ng中kernel的值

crosstool-ng中，和kernel对应的值，应该就是"Operating System"中的"Target OS"的值了。

比如我之前配置为Linux的话，就是：

```
Operating System
  Target OS (linux)  --->
```

对应的配置参数是：GEN_CHOICE_KERNEL中的CT_KERNEL_linux

[4] http://www.crifan.com/uboot_cross_compile_gcc_config_arm_lib1funcs_asm_undefined_reference_to_raise/

对应的help的解释是：

```
 ┌──────────────────────────────────────────────────────────────────────── linux
──────────────────────────────────────────────────────────────────────┐
 │ CT_KERNEL_linux:                                                                 |
 |                                                                                  |
 │ Build a toolchain targeting systems running Linux as a kernel.                                |
 |                                                                                  |
 │ Symbol: KERNEL_linux [=y]                                                                |
 │ Type  : boolean                                                                  |
 │ Prompt: linux                                                                    |
 |   Defined at config.gen/kernel.in:8                                                      |
 |   Depends on: GEN_CHOICE_KERNEL [=y] && KERNEL_linux_AVAILABLE [=y]
|
 |   Location:                                                                      |
 |     -> Operating System                                                                  |
 |       -> Target OS (GEN_CHOICE_KERNEL [=y])                                                      |
 |   Selects: KERNEL_SUPPORTS_SHARED_LIBS [=y]                                                      |
```

3.1.2.4. 交叉编译器名字中的system部分

system，直译为，系统

其实主要表示的，交叉编译器所选择的库函数和目标系统

最常见的一些值有，gnu，gnueabi，uclibcgnueabi等等。

其中，此处有几方面的值，表示了几方面的含义：

3.1.2.4.1. system中的gnu

好像都是gnu

不是很明白，貌似是：

gnu == gnu libc == glibc

即，gnu，就是表示用的是glibc的意思。

3.1.2.4.1.1. crosstool-ng中system为gnu的情况

crosstool-ng中，和system中gnu对应的值，应该就是"C-library"中的"C library"的值设置为"glibc"了。

```
C-library
  C library (glibc)  --->
```

对应的配置参数是：CT_LIBC_glibc

对应的help的解释是：

```
┌──────────────────────────────────────────────────── glibc
──────────────────────────────────────────────────┐
│ CT_LIBC_glibc:                                                                  │
│                                                                                 │
│ The de-facto standard for Linux distributions.                                  │
│ Feature-rich, but large...  Most usefull for desktop-like systems.              │
│                                                                                 │
│ Symbol: LIBC_glibc [=y]                                                         │
│ Type  : boolean                                                                 │
│ Prompt: glibc                                                                   │
│   Defined at config.gen/libc.in:53                                              │
│   Depends on: GEN_CHOICE_LIBC [=y] && LIBC_glibc_AVAILABLE [=y] && !WINDOWS [=n]
&& !\    │
│ BARE_METAL [=n] && ARCH_USE_MMU [=y]                                            │
│   Location:                                                                     │
│     -> C-library                                                                │
│       -> C library (GEN_CHOICE_LIBC [=y])                                       │
│   Selects: LIBC_SUPPORT_NPTL [=y] && CC_CORE_PASSES_NEEDED [=y]                 │
```

3.1.2.4.2. system中的eabi

与此相对应的，之前早期的是oabi

eabi和oabi的对比，详见：

[整理] EABI和OABI [5]

3.1.2.4.2.1. crosstool-ng中system为eabi的情况

crosstool-ng中，和system中eabi对应的值，应该就是"Target options"中的"-*- Use EABI"的值了。

```
Target options
  -*- Use EABI
```

注意：此处是选择的ARM，EABI在此处，已经是默认，无法修改的值，只能是EABI了。

好像是因为，Linux内核等其他选择导致的，此处无法更改EABI。

对应的配置参数是：CT_ARCH_ARM_EABI

对应的help的解释是：

```
┌──────────────────────────────────────────────────── Use EABI
──────────────────────────────────────────────────┐
│ CT_ARCH_ARM_EABI:                                                               │
│                                                                                 │
```

[5] http://www.crifan.com/order_eabi_and_oabi/

```
| Set up the toolchain so that it generates EABI-compliant binaries.
|
| If you say 'n' here, then the toolchain will generate OABI binaries.
| OABI has long been deprecated, and is now considered legacy.
|
| Symbol: ARCH_ARM_EABI [=y]
| Type  : boolean
| Prompt: Use EABI
|   Defined at config/arch/arm.in.2:54
|   Depends on: ARCH_arm [=y]
|   Location:
|     -> Target options
|   Selected by: ARCH_ARM_EABI_FORCE [=y] && ARCH_arm [=y]
```

3.1.2.4.3. system中的uclibc

uclibc，是c库中的一种

crosstool-ng中，目前支持三种：glibc,eglibc,uclibc

关于三种的关系，详见：

【整理】uclibc,eglibc,glibc之间的区别和联系 [6]

3.1.2.4.3.1. crosstool-ng中system为uclibc的情况

crosstool-ng中，和system中uclibc对应的值，应该就是"C-library"中的"C library"的值设置为"uClibc"了。

```
C-library
  C library (uClibc)  --->
```

对应的配置参数是：CT_LIBC_uClibc

对应的help的解释是：

```
 ┌──────────────────────────────────────────────────── uClibc
──────────────────────────────────────────────────────┐
| CT_LIBC_uClibc:
|
| The de-facto standard for embeded linux systems.
|
| Highly configurable, thus as feature-rich as you
| need, without compromising for size.
|
| Symbol: LIBC_uClibc [=y]
| Type  : boolean
| Prompt: uClibc
|   Defined at config.gen/libc.in:24
```

[6] http://www.crifan.com/relation_between_uclibc_glibc_eglibc/

```
 |   Depends on: GEN_CHOICE_LIBC [=y] && LIBC_uClibc_AVAILABLE [=y] && !WINDOWS
[=n] && !\   |
 | BARE_METAL [=n]                                                    |
 |   Location:                                                      |
 |     -> C-library                                                  |
 |       -> C library (GEN_CHOICE_LIBC [=y])                                  |
 |   Selects: LIBC_SUPPORT_LINUXTHREADS [=y] && LIBC_SUPPORT_THREADS_NONE [=y]
&& \          |
 | CC_CORE_PASSES_NEEDED [=y]                                                 |
```

所以，针对上述，gnu，eabi，uclibc等几个选项，对应的常见的一些组合的含义是：

- gnu
  等价于：glibc+oabi

- gnueabi
  等价于：glibc+eabi

- uclibc
  等价于：uclibc+oabi（待确认）

例 3.4. 举例：交叉编译器中的system的值

- arm-cortex_a8-linux-gnueabi中的gnueabi，即glibc+eabi
- mips-ar2315-linux-gnu中的gnu，即glibc+oabi
- powerpc-e500v2-linux-gnuspe中的gnuspe，没搞懂啥意思。。
- x86_64-unknown-mingw32中的mingw32，用的是Windows下的mingw32的库

3.2. 如何得到交叉编译器

了解了之前的交叉编译器的命名规则后，也就明白了交叉编译，针对不同架构，平台，目标系统等的区别。

而对于嵌入式开发，想要获得，针对我们所需要的，目标的CPU所合适的交叉编译器，就成了接下来，顺其自然要考虑的事情。

想要得到，可用的，针对自己的CPU的交叉编译器，主要有两种方式：

3.2.1. 拿来主义：直接去下载和使用别人已经编译好的交叉编译器

难度等级：1级

这个没有啥特殊要求解释的，就是，网上，总会有，免费的，好用的各种资源的。

其中就包括，别人，已经针对某些CPU，某些平台，编译好了交叉编译器了

而你要做的事情就是：找到合适的，并下载下来使用。

关于网上，现存的，可用的，针对arm的交叉编译器，可以参考我之前整理的一些：

【整理】arm的交叉工具链（交叉编译器）下载 [7]

常见的，获得交叉编译器的方式

其实，相对比较常见的，获得可用的，交叉编译器的方式是：

当你购买某一家的（嵌入式）开发板的时候，然后开发板厂家，提供你对应的硬件开发板的同时，也提供对应的整套开发软件。

此处，整套的开发软件，其中就包括，对应的，用来编译其厂家提供的BSP的软件源码的，交叉编译器。

即：一般来说，你买了某家的某款的嵌入式开发板的时候，就送了对应的交叉编译器

例 3.5. 购买了TQ2440的开发板，就送了对应的交叉编译器

拿我之前，购买的TQ2440的开发板为例

买了TQ2440的开发板的时候，就送了对应的光盘了。

光盘里面，就包含了对应的各种开发资料和交叉编译器

其实，对于交叉编译器本身，TQ2440的厂家，叫做天嵌公司，其本身有自己的论坛，论坛里面，也可以免费下载到对应的交叉编译器的

[光盘下载] 2010年6月 最新TQ2440光盘下载 （Linux内核，WinCE的eboot，uboot均有更新） [8]

3.2.2. 自己动手，丰衣足食：自己去编译出来一套自己想要的交叉编译器

如果网上没有合适的交叉编译器，那么就需要你手动去制作了。

自己手动制作交叉编译器，又分两种：

3.2.2.1. 白手起家从头开始制作交叉编译器

难度等级：10级

此法，目前我也还没折腾过

只是知道，难度，相对是最大的

等抽空折腾了之后，再总结出来。

3.2.2.2. 借助别人的工具去制作交叉编译器

难度等级：6级

[7] http://www.crifan.com/arm_cross_toolchain_cross_compiler_download/

[8] http://www.armbbs.net/forum.php?mod=viewthread&tid=859&extra=page%3D1%26filter%3Dtypeid%26typeid%3D181%26typeid%3D181

相关旧帖：【整理】交叉编译和现存的交叉编译工具[9]

对于，制作交叉编译器这样的事情，本身是很需要技术含量，和很耗时的事情

所以，对此，现在现存很多相关的工具，以简化制作交叉编译器这个事情的复杂度，帮你节省很多精力和时间

而你所要做的事情就只是：

了解有哪些工具，选个合适的工具，搞懂如何使用，用其制作出你所需要的交叉编译器，即可。

关于现在有哪些交叉编译器的制作工具，正是下文正要详细解释的：

第 3.3 节 “各种制作交叉编译器的工具”

3.3. 各种制作交叉编译器的工具

下面，就针对，现存已知的，交叉编译器，的制作工具，进行简单的总结和介绍：

3.3.1. crosstool-NG

crosstool-ng的主页：http://crosstool-ng.org/

关于crosstool-ng的更多介绍和使用，可以参考我的另外一个教程：

crosstool-ng详解 [10]

3.3.2. Buildroot

Buildroot主页：http://www.buildroot.net/

特点：不仅能制作交叉工具链，而且还可以制作根文件系统rootfs。而且还支持同时编译对应的Linux内核和Uboot。

关于Buildroot的更多介绍和使用，可以参考我的另外一个教程：

Buildroot详解 [11]

3.3.3. crosstool

http://kegel.com/crosstool/

现在用的最多的是那个0.43的版本：

crosstool-0.43.tar.gz [12]

也可以去在线浏览对应的源码：在线浏览crosstool-0.43源码[13]

3.3.4. Embedded Linux Development Kit (ELDK)

http://www.denx.de/wiki/DULG/ELDK

[9] http://www.crifan.com/summary_cross_compile_cross_compiler_cross_toolchain/
[10] http://www.crifan.com/files/doc/docbook/crosstool_ng/release/html/crosstool_ng.html
[11] http://www.crifan.com/files/doc/docbook/buildroot_intro/release/html/buildroot_intro.html
[12] http://kegel.com/crosstool/crosstool-0.43.tar.gz
[13] http://kegel.com/crosstool/crosstool-0.43/

也是和交叉编译相关的。

提供编译好的东西供使用。

可以去这里：

http://www.denx.de/wiki/view/DULG/ELDKAvailability

去下载。

3.3.5. OpenEmbedded的BitBake

OpenEmbedded是一个创建嵌入式Linux的整套框架，其中包括了制作对应的交叉编译器的工具，叫做BitBake

OpenEmbedded简称OE。

OpenEmbedded主页：http://www.openembedded.org/wiki/Main_Page

OpenEmbedded的在线文档：OpenEmbedded User Manual[14]

关于BitBake可去参考：

http://en.wikipedia.org/wiki/BitBake

中的：

http://developer.berlios.de/projects/bitbake

3.3.6. Crossdev

http://www.gentoo.org/proj/en/base/embedded/handbook/

中的：

http://www.gentoo.org/proj/en/base/embedded/handbook/?part=1&chap=1

3.3.7. OSELAS.Toolchain()

http://www.pengutronix.de/oselas/toolchain/index_en.html

3.4. 使用已有的交叉编译器和自己手动编译交叉编译器之间的对比

后来在：

http://ymorin.is-a-geek.org/projects/crosstool

中发现作者ymorin之前的一些会议演讲：

http://crosstool-ng.org/publis/

等内容，比如：

[14] http://docs.openembedded.org/usermanual/usermanual.html

200910-ELCE-Morin-Building_our_own_toolchains.pdf [15]

其中，对于下载别人已有的交叉编译器和自己手动编译一个交叉编译器之间，做了些对比，分析了各自的优缺点。

个人觉得其说的比较全面，特此整理如下：

3.4.1. 使用已有交叉编译器的优点

- 安装和使用都很方便
  别人发布的，已经编译好的交叉编译器，基本都是压缩包

  然后你解压后，即可得到对应的，可用的，交叉编译器

  其效果，类似于，之前自己编译出来的交叉编译器，有对应的交叉编译版本的gcc，ld等等程序，即：

  arm-xscale-linux-gnueabi-gcc

  arm-xscale-linux-gnueabi-ld

  等等文件的了。

  比如：

  【记录】Ubuntu下用crosstool-ng为xscale建立交叉编译器arm-xscale-linux-gnueabi-gcc[16]

  中的：

```
crifan@ubuntu:~/develop/crosstool-ng/crosstool-ng-1.18.0_build$ ls /home/crifan/
develop/crosstool-ng/x-tools/arm-xscale-linux-gnueabi/bin -lh
total 18M
-r-xr-xr-x 1 crifan crifan 605K Aug  8 01:10 arm-xscale-linux-gnueabi-addr2line
-r-xr-xr-x 2 crifan crifan 633K Aug  8 01:10 arm-xscale-linux-gnueabi-ar
-r-xr-xr-x 2 crifan crifan 1.1M Aug  8 01:10 arm-xscale-linux-gnueabi-as
-r-xr-xr-x 2 crifan crifan 276K Aug  8 01:10 arm-xscale-linux-gnueabi-c++
lrwxrwxrwx 1 crifan crifan   28 Aug  8 00:54 arm-xscale-linux-gnueabi-cc -> arm-xscale-
linux-gnueabi-gcc
-r-xr-xr-x 1 crifan crifan 605K Aug  8 01:10 arm-xscale-linux-gnueabi-c++filt
-r-xr-xr-x 1 crifan crifan 276K Aug  8 01:10 arm-xscale-linux-gnueabi-cpp
-r-xr-xr-x 1 crifan crifan 3.1K Aug  7 23:57 arm-xscale-linux-gnueabi-ct-ng.config
-r-xr-xr-x 1 crifan crifan  26K Aug  8 01:10 arm-xscale-linux-gnueabi-elfedit
-r-xr-xr-x 2 crifan crifan 276K Aug  8 01:10 arm-xscale-linux-gnueabi-g++
-r-xr-xr-x 2 crifan crifan 272K Aug  8 01:10 arm-xscale-linux-gnueabi-gcc
-r-xr-xr-x 2 crifan crifan 272K Aug  8 01:10 arm-xscale-linux-gnueabi-gcc-4.6.0
-r-xr-xr-x 1 crifan crifan  30K Aug  8 01:10 arm-xscale-linux-gnueabi-gcov
-r-xr-xr-x 1 crifan crifan 2.7M Aug  8 01:10 arm-xscale-linux-gnueabi-gdb
-r-xr-xr-x 1 crifan crifan 2.7M Aug  8 01:10 arm-xscale-linux-gnueabi-gdbtui
-r-xr-xr-x 1 crifan crifan 670K Aug  8 01:10 arm-xscale-linux-gnueabi-gprof
-r-xr-xr-x 4 crifan crifan 1.1M Aug  8 01:10 arm-xscale-linux-gnueabi-ld
-r-xr-xr-x 4 crifan crifan 1.1M Aug  8 01:10 arm-xscale-linux-gnueabi-ld.bfd
-r-xr-xr-x 1 crifan crifan  11K Aug  8 01:10 arm-xscale-linux-gnueabi-ldd
-r-xr-xr-x 2 crifan crifan 617K Aug  8 01:10 arm-xscale-linux-gnueabi-nm
-r-xr-xr-x 2 crifan crifan 775K Aug  8 01:10 arm-xscale-linux-gnueabi-objcopy
-r-xr-xr-x 2 crifan crifan 943K Aug  8 01:10 arm-xscale-linux-gnueabi-objdump
```

[15] http://crosstool-ng.org/publis/ELC-E/2009/200910-ELCE-Morin-Building_our_own_toolchains.pdf
[16] http://www.crifan.com/ubuntu_crosstool_ng_cross_compile_for_xscale_arm_xscale_linux_gnueabi/

```
-r-xr-xr-x 1 crifan crifan  11K Aug  8 01:10 arm-xscale-linux-gnueabi-populate
-r-xr-xr-x 2 crifan crifan 633K Aug  8 01:10 arm-xscale-linux-gnueabi-ranlib
-r-xr-xr-x 1 crifan crifan 317K Aug  8 01:10 arm-xscale-linux-gnueabi-readelf
-r-xr-xr-x 1 crifan crifan 609K Aug  8 01:10 arm-xscale-linux-gnueabi-size
-r-xr-xr-x 1 crifan crifan 605K Aug  8 01:10 arm-xscale-linux-gnueabi-strings
-r-xr-xr-x 2 crifan crifan 775K Aug  8 01:10 arm-xscale-linux-gnueabi-strip
crifan@ubuntu:~/develop/crosstool-ng/crosstool-ng-1.18.0_build$
```

然后，你把包含了上述arm-xscale-linux-gnueabi-gcc等文件的路径，加到环境变量PATH中，然后就可以使用了。

- 已验证和测试
  别人发布的，交叉编译器，一般都是，经过相应的验证和测试

  保证了，可以使用，不会出问题的。

  所以，你可以放心使用，不会担心，出现编译程序出错的时候，确保不会是由于交叉编译器问题。
- 已优化
  别人发布的交叉编译器，一般都是经过，在制作的时候，加了一些优化参数

  使得针对某些方面，比如性能等等，做了相应的优化

  使得，交叉编译器编译出来的程序，对于目标的CPU，是相对最优的
- （售后）支持比较好
  就像你买东西，很多时候，不出问题的时候，售后，看不出有啥用

  但是当出问题，就发现，售后其实很重要

  而搞嵌入式开发，尤其是交叉编译

  很多时候，涉及到很多技术细节

  有时候遇到问题的话，如果你不熟悉，不了解，不会解决

  这时候，去找到，交叉编译器的提供者，去咨询

  对于你来说，就显得很有价值，很重要了。

  别人提供的交叉编译器的话，往往都是提供后续的技术支持的

  对于多数的开发者，这点，还是有价值的

3.4.1.1. 已有的交叉工具链的下载

参考：http://www.cnblogs.com/esion/archive/2012/08/28/2660033.html，其实是可以安装后，设置好环境变量，之后就可以一直不变的去使用的。

http://www.myir-tech.com/download.asp中提供了很多东西下载，包括工具链。

3.4.1.1.1. ELDK的下载

http://www.denx.de/en/News/WebHome

-> ftp://ftp.denx.de/pub/eldk/5.3/

-> ftp://ftp.denx.de/pub/eldk/5.3/iso/

中有各种的，基于arm的eldk，比如：

- eldk-5.3-armv4t.iso
- eldk-5.3-armv5te.iso
- eldk-5.3-armv6.iso
- eldk-5.3-armv6.iso
- eldk-5.3-armv7a-hf.iso
- eldk-5.3-armv7a.iso

貌似是下载下来，解压后，去运行安装脚本后，就可以使用了。

3.4.2. 使用已有交叉编译器的缺点

- 无针对你自己的CPU的优化
  别人提供的，已有的交叉编译器，相对来说，更多时候，都是针对某个系列的CPU，进行制作出来的，进行优化的

  即，其优化，也是相对比较通用的

  如果你的CPU和其不同，则其就没有对应的优化了

- 专有化
  和上面有点类似，即没有针对你自己的CPU，进行优化

- 太老，太旧
  别人给的，网上可以下载到的，很多交叉编译器，相对来说，版本都比较旧了，都是很老的，之前的了

  其所用的，其他各个组件，模块，也都是旧版本的

  旧的软件，对于新出来的CPU，新的功能等，自然没有对应的支持

  也就无法发挥新的硬件特性了了

- 其所用的源码不清楚
  别人给你编译好的交叉编译器，你是可以用了

  但是，其针对于具体什么版本的源码，以及是否打了相应补丁等情况

  你是无法，也很难知道的

  所以，即无法掌控此已有交叉编译器的所用的源码的确切情况

  此点，针对于你对自己的CPU很熟悉的情况下，想要完全了解已有交叉编译器的背后的情况而言，算是个缺点

  即不能完全在你掌握范围内，不清楚后面的情况

- 未必适合你的系统
  或许是不可重载（relocatable）的

  没有很方便的方式去获得一些系统库文件

所以，未必真正适合你自己的，嵌入式系统环境

3.4.3. 自己手动编译交叉编译器的优点

- 自定义各种组件及版本
  对于自己去制作交叉编译器的话，则其所用的各种组件，模块的版本，都可以自己选择和定制

- 针对你自己的CPU进行特定的优化

- 对于各种补丁包，已经很清楚了
  既然是自己制作交叉编译器，那么自然选择了，相对较新的源码包，以及各种模块

  其自然，已经现存有很多补丁包

  这些补丁包，修复了很多bug，或者是增加了很多新的功能支持等

  这时候，你就可以找到并打上这些补丁包

  以实现，修复该模块的已知的bug，增加新的功能的支持了

- Same source for all targets
  暂未搞懂此点的确切含义。

- 可以同步更新最新补丁
  相应的，如果有最新的补丁，也可以及时打上

- 可重复
  对于自己制作交叉编译器的话，

  制作完成后，就确定了对应的各个模块的版本，有了对应的配置文件

  此配置，拿到别的地方，别人用同样的工具和类似的环境，是可以重新编译为其自己的

  即，所谓的可重复，别人可以利用你已经验证，可以正常制作和使用的各个版本的模块和配置，实现自己也制作一个属于自己的交叉编译器

- 有社区支持
  当然，对应的，在制作交叉编译器期间，使用交叉编译器期间，

  出了问题，有任何疑问，都有对应的社区的，热心人，帮你解决，回答你的问题。

- 适合你自己的系统
  如前所述，拿别人已制作好的交叉编译器，未必真正完全适合你自己的系统

  而如果你自己制作的话，当然就完全根据自己的需求，去制作出最适合自己的系统的交叉编译器了

3.4.4. 自己手动编译交叉编译器的缺点

- 制作交叉编译器相对比较复杂

制作交叉编译器，相对来说，是个技术活加体力活

不仅仅要求有对应的知识背景，解决各种问题的能力，

还要准备一定的时间和精力，去真正实现，真正去建立出来一个可用的交叉编译器

- 可能存在的，对于你的处理器支持度不是足够好
  既然制作交叉编译器期间，可能需要打上很多的patch等等

  那也就存在，比如缺少了对应的补丁，补丁不太完整，补丁不合适等等特殊情况

  使得，对于对应的，有些处理器，支持度，不是很好
- 验证
  在制作出了交叉编译器后，其实还需要一定的验证，保证其可用，好用。

  这也需要对应的技术能力，和精力，去实现。

- 社区的支持
  对于社区的支持，有其之前所说的优点，那就是，社区活跃的话，可能解决问题的效率就相对较高

  但是当社区不够活跃，社区中没有合适的人，那么可能你的问题，就很难得到解决

  即，有问题的时候，依靠社区去解决，未必是完全靠谱的。

第 4 章 交叉编译心得和注意事项

4.1. 交叉编译心得和注意事项

待整理：

相关旧帖：【整理】交叉编译心得和注意事项[1]

[1] http://www.crifan.com/summary_cross_compile_library_note/

参考书目

[1] GNU Binutils[1]

[2] [光盘下载] 2010年6月 最新TQ2440光盘下载（Linux内核，WinCE的eboot，uboot均有更新）[2]

[3] http://kegel.com/crosstool/

[4] http://www.denx.de/wiki/DULG/ELDK

[5] http://www.openembedded.org/wiki/Main_Page

[6] http://en.wikipedia.org/wiki/BitBake

[7] http://www.gentoo.org/proj/en/base/embedded/handbook/

[8] http://www.pengutronix.de/oselas/toolchain/index_en.html

[9] http://anyhu.blog.sohu.com/214755651.html

[10] http://ymorin.is-a-geek.org/projects/crosstool

[11] 200910-ELCE-Morin-Building_our_own_toolchains.pdf[3]

[12] 【整理】交叉编译和现存的交叉编译工具[4]

[13] 【整理】关于Toolchain,cross toolchain,cross compiler[5]

[14] 【记录】Ubuntu下为QEMU的arm平台交叉编译BusyBox[6]

[15] 嵌入式开发之交叉编译器[7]

[16] 【整理】crosstool中如何设置xscale的Tuple’s vendor string(CT_TARGET_VENDOR)[8]

[17] 【记录】重试使用最新版本1.18.0的crosstool-ng去配置和编译xscale的交叉编译器[9]

[18] 【已解决】uboot交叉编译出错：gcc/config/arm/lib1funcs.asm:1266: undefined reference to `raise’[10]

[19] [整理]EABI和OABI[11]

[20] 【整理】uclibc,eglibc,glibc之间的区别和联系[12]

[21] 【整理】arm的交叉工具链（交叉编译器）下载[13]

[22] crosstool-ng详解[14]

[23] Buildroot详解[15]

[1] http://www.gnu.org/software/binutils/
[2] http://www.armbbs.net/forum.php?mod=viewthread&tid=859&extra=page%3D1%26filter%3Dtypeid%26typeid%3D181%26typeid%3D181
[3] http://crosstool-ng.org/publis/ELC-E/2009/200910-ELCE-Morin-Building_our_own_toolchains.pdf
[4] http://www.crifan.com/summary_cross_compile_cross_compiler_cross_toolchain/
[5] http://www.crifan.com/summary_about_toolchain_cross_compiler/
[6] http://www.crifan.com/ubuntu_download_config_cross_compile_busybox_for_qemu_arm/
[7] http://www.crifan.com/embedded_development_cross_compiler_and_toolchain/
[8] http://www.crifan.com/crosstool_tuple_vendor_string_ct_target_vendor/
[9] http://www.crifan.com/reuse_latest_version_crosstool_ng_to_config_and_compile_for_xscale/
[10] http://www.crifan.com/uboot_cross_compile_gcc_config_arm_lib1funcs_asm_undefined_reference_to_raise/
[11] http://www.crifan.com/order_eabi_and_oabi/
[12] http://www.crifan.com/relation_between_uclibc_glibc_eglibc/
[13] http://www.crifan.com/arm_cross_toolchain_cross_compiler_download/
[14] http://www.crifan.com/files/doc/docbook/crosstool_ng/release/html/crosstool_ng.html
[15] http://www.crifan.com/files/doc/docbook/buildroot_intro/release/html/buildroot_intro.html

[24] 【记录】Ubuntu下用crosstool-ng为xscale建立交叉编译器arm-xscale-linux-gnueabi-gcc[16]

[25] 【整理】交叉编译心得和注意事项[17]

[16] http://www.crifan.com/ubuntu_crosstool_ng_cross_compile_for_xscale_arm_xscale_linux_gnueabi/
[17] http://www.crifan.com/summary_cross_compile_library_note/