

Glibc Binutils GCC 配置选项简介

作者：[金步国](#)

版权声明

本文作者是一位自由软件爱好者，所以本文虽然不是软件，但是本着 GPL 的精神发布。任何人都可以自由使用、转载、复制和再分发，但必须保留作者署名，亦不得对声明中的任何条款作任何形式的修改，也不得附加任何其它条件。您可以自由链接、下载、传播此文档，但前提是必须保证全文完整转载，包括完整的版权信息和作译者声明。

其他作品

本文作者十分愿意与他人共享劳动成果，如果你对我的其他翻译作品或者技术文章有兴趣，可以在如下位置查看现有作品的列表：

- [金步国作品列表](#)

BUG 报告，切磋与探讨

由于作者水平有限，因此不能保证作品内容准确无误，请在阅读中自行鉴别。如果你发现了作品中的错误，请您来信指出，哪怕是错别字也好，任何提高作品质量的建议我都将虚心接纳。如果你愿意就作品中的相关内容与我进行进一步切磋与探讨，也欢迎你与我联系。联系方式：Email: csfrank@citiz.net；QQ: 70171448；MSN: csfrank122@hotmail.com

Glibc 安装指南(适用于 2.3/2.4/2.5/2.6/2.7)

编译前的预备知识与要点提示

Glibc-2.3.6 建议使用 GCC-4.0 编译，Glibc-2.4/2.5 建议使用 GCC-4.1 编译，Glibc-2.6/2.7 建议使用 GCC-4.2 编译。所有这些版本最低要求使用 GCC-3.4 编译。

编译 Glibc 时使用的内核头文件版本可以比实际运行 Glibc 的内核版本高。如果实际运行的内核版本比头文件版本高，那么新内核的新特性将无法使用。更多细节可以查看[\[八卦故事\]内核头文件传奇](#)的跟帖部分。

不要在运行中的系统上安装 Glibc，否则将会导致系统崩溃，至少应当将新 Glibc 安装到其他的单独目录，以保证不覆盖当前正在使用的 Glibc。

Glibc 不能在源码目录中编译，它必须在一个额外分开的目录中编译。这样在编译发生错误的时候，就可以删除整个编译目录重新开始。

在运行 configure 脚本时可以设置 CC CFLAGS LDFLAGS 环境变量来优化编译。语法：configure [OPTION]... [VAR=VALUE]...

需要注意的是有些测试项目假定是以非 root 身份执行的，因此我们强烈建议你使用非 root 身份编译和测试 Glibc。

配置选项

下列选项皆为非默认值[特别说明的除外]

```
--help
--version
--silent
--cache-file=FILE
--config-cache
--no-create
--srcdir=DIR
--exec-prefix=EPREFIX
--bindir=DIR
--sbindir=DIR
--libexecdir=DIR
--sysconfdir=DIR
--sharedstatedir=DIR
--localstatedir=DIR
--libdir=DIR
--includedir=DIR
--oldincludedir=DIR
--datarootdir=DIR
--datadir=DIR
--infodir=DIR
--localedir=DIR
--mandir=DIR
```

```
--do cd ir= D IR
--h tm ld ir= D IR
--d vid ir= D IR
--p d fd ir= D IR
--p sd ir= D IR
--b u i ld= B U I LD
--h o s t= H O S T
```

这些选项的含义基本上通用于所有软件包，这里就不特别讲解了。

```
--p re fix= P R E F I X
```

安装目录，默认为 /usr/local

Linux 文件系统标准要求基本库必须位于 /lib 目录并且必须与根目录在同一个分区上，但是 /usr 可以在其他分区甚至是其他磁盘上。因此，如果指定 --prefix=/usr，那么基本库部分将自动安装到 /lib 目录下，而非基本库部分则会自动安装到 /usr/lib 目录中。但是如果保持默认值或指定其他目录，那么所有组件都将被安装到 PREFIX 目录下。

```
--en ab le-add-on s[= D R 1,D R 2,...]
```

编译 D R 1,D R 2,...中的附加软件包。其中的 “D R”是附加软件包的目录名。未指定列表或指定为“yes”则编译所有源码树根目录下找到的附加软件包。Glibc-2.4/2.5/2.6/2.7 默认值为“yes”，而 Glibc-2.3.6 默认为--disable-add-ons。

```
--d isab le-sh are d
```

不编译共享库(即使平台支持)。在支持 ELF 并且使用 GNU 连接器的系统上默认为 --enable-shared。[提示]--disable-static 选项实际上是无效的，静态库总是被无条件的编译和安装。

```
--en ab le-b ind -n ow
```

禁用“lazy binding”，也就是动态连接器在载入 DSO 时就解析所有符号(不管应用程序是否用得到)，默认行为是“lazy binding”，也就是仅在应用程序首次使用到的时候才对符号进行解析。因为在大多数情况下，应用程序并不需要使用动态库中的所有符号，所以默认的 “lazy binding”可以提高应用程序的加载性能并节约内存用量。然而，在两种情况下，“lazy binding”是不利的：①因为第一次调用 DSO 中的函数时，动态连接器要先拦截该调用来解析符号，所以初次引用 DSO 中的函数所花的时间比再次调用要花的时间长，但是某些应用程序不能容忍这种不可预知性。②如果一个错误发生并且动态连接器不能解析该符号，动态连接器将终止整个程序。在“lazy binding”方式下，这种情况可能发生在程序运行过程中的某个时候。某些应用程序也是不能容忍这种不可预知性的。通过关掉“lazy binding”方式，在应用程序接受控制权之前，让动态连接器在处理进程初始化期间就发现这些错误，而不要到运行时才出乱子。

```
--en ab le-b ound ed
```

启用运行时边界检查(比如数组越界)，这样会降低运行效率，但能防止某些溢出漏洞。

```
--d isab le-force- in stall
```

不强制安装当前新编译的版本(即使已存在的文件版本更新)。

```
--d isab le-h idden -p lt
```

默认情况下隐藏仅供内部调用的函数，以避免这些函数被加入到过程链接表(PLT,Procedure Linkage Table)中，这样可以减小 PLT 的体积并将仅供内部使用的函数隐藏起来。而使用该选项将把这些函数暴露给外部用户。

```
--en ab le-kerne l= V E R S I O N
```

VERSION 的格式是 X.Y.Z，表示编译出来的 Glibc 支持的最低内核版本。VERSION 的值越高(不能超过内核头文件的版本)，加入的兼容性代码就越少，库的运行速度就越快。

```
--en ab le-oldest-abi= A B I
```

启用老版本的应用程序二进制接口支持。ABI 是老 Glibc 的版本号。默认值大部分情况下为 --disable-oldest-abi，建议明确指定为 --disable-oldest-abi

```
--en ab le-check-abi
```

在“make check”时执行“make check-abi”。[提示]在我的机器上始终导致 check-abi libm 测试失败。

```
--d isab le-version ing
```

不在共享库对象中包含符号的版本信息。这样可以减小库的体积，但是将不兼容依赖于老版本 C 库的二进制程序。[提示]在我的机器上使用此选项总是导致编译失败。

```
--en ab le-om itfp
```

编译时忽略帧指示器(使用 -fomit-frame-pointer 编译)，并采取一些其他优化措施。忽略帧指示器可以提高运行效率，但是调试将变得不可用，并且可能生成含有 bug 的代码。使用这个选项还将导致额外编译带有调试信息的非优化版本的静态库(库名称以“_g”结尾)。

```
--d isab le-p ro file
```

禁用 profiling 信息相关的库文件编译。Glibc-2.3.6 默认为 enable，Glibc-2.4/2.5/2.6/2.7 默认为 disable。

```
--d isab le-san ity-check s
```

真正的禁用线程(仅在特殊环境下使用该选项)。

```
--en ab le-stackguard-random ization
```

在程序启动时使用一个随机数初始化 __stack_chk_guard，主要用来抵抗恶意攻击。该选项仅存在于 2.4 及以上版本中。

```
--en ab le-static-nss
```

编译静态版本的 NSS(Name Service Switch)库。不推荐这样做，因为连接到静态 NSS 库的程序不能动态配置以使用不同的名字数据库。

```
--w ith-headers= D IR
```

指定内核头文件的所在目录。

```
--w ith-b inu tils= D IR
```

强制指定编译时使用的 Binutils(as,ld)的位置。

```
--w ith-out-cvs
```

不访问 CVS 服务器。推荐使用该选项，特别对于从 CVS 下载的的版本。

```
--w ith-elf
```

指定使用 ELF 对象格式，建议在支持 ELF 的 Linux 平台上使用此选项明确指定。

```
--w ith-gd= D IR
```

```
--w ith-gd-in clude= D IR
```

```
--w ith-gd-lib= D IR
```

强制指定 libgd 的安装目录(DIR/include 和 DIR/lib)。后两个选项分别指定包含文件和库目录。

```
--w ith-gm p= D IR
```

强制指定 `gmp` 的安装目录。

`--without-selinux`
禁用 `SELinux` 支持。

`--without-fp`
仅在硬件没有浮点运算单元并且操作系统没有模拟的情况下使用。`x86` 与 `x86_64` 的 `CPU` 都有专门的浮点运算单元。而且 `Linux` 有 `FPU` 模拟。简单的说，不要 `without` 这个选项！因为它会导致许多问题！

`--without-tls`
禁止编译支持线程本地存储(TLS)的库。使用这个选项将导致兼容性问题。

以下选项意义不大，一般不用考虑它们

`--without-__thread`
即使平台支持也不使用 `TSL` 特性。建议不要使用该选项。

`--enable-all-warnings`
在编译时显示所有编译器警告，也就是使用 `-Wall` 选项编译。

`--with-cpu=CPU`
在 `gcc` 命令行中加入“`-mcpu=CPU`”。鉴于“`-mcpu`”已经被反对使用，所以建议不要设置该选项，或者设为 `--without-cpu`。

`--with-xcoff`
使用 `XCOFF` 对象格式(主要用于 `windows`)。

`--x-includes=DIR`
`--x-libraries=DIR`
分别指定 `X` 的头文件和库文件目录。

编译与测试

使用 `make` 命令编译，使用 `make check` 测试。如果 `make check` 没有完全成功，就千万不要使用这个编译结果。需要注意的是有些测试项目假定是以非 `root` 身份执行的，因此我们强烈建议你使用非 `root` 身份编译和测试。

测试中需要使用一些已经存在的文件(包括随后的安装过程)，比如 `/etc/passwd`，`/etc/nsswitch.conf` 之类。请确保这些文件中包含正确的内容。

安装与配置

使用 `make install` 命令安装。比如：`make install LC_ALL=C`

如果你打算将此 `Glibc` 安装为主 `C` 库，那么我们强烈建议你关闭系统，重新引导到单用户模式下安装。这样可以将可能的损害减小到最低。

安装后需要配置 `GCC` 以使其使用新安装的 `C` 库。最简单的办法是使用恰当 `GCC` 的编译选项(比如 `-Wl,--dynamic-linker=/lib/ld-linux.so.2`)重新编译 `GCC`。然后还需要修改 `specs` 文件(通常位于 `/usr/lib/gcc-lib/TARGET/VERSION/specs`)，这个工作有点像巫术，调整实例请参考 `LFS` 中的两次工具链调整。

可以在 `make install` 命令行使用‘`install_root`’变量指定安装实际的安装目录(不同于 `--prefix` 指定的值)。这个在 `chroot` 环境下或者制作二进制包的时候通常很有用。‘`install_root`’必须使用绝对路径。

被‘`grantpt`’函数调用的辅助程序‘`/usr/libexec/pt_chown`’以 `setuid 'root'` 安装。这个可能成为安全隐患。如果你的 `Linux` 内核支持‘`devptsfs`’或‘`devfs`’文件系统提供的 `pty slave`，那么就不需要使用 `pt_chown` 程序。

安装完毕之后你还需要配置时区和 `locale`。使用 `localedef` 来配置 `locale`。比如使用‘`localedef -i de_DE -f ISO-8859-1 de_DE`’将 `locale` 设置为‘`de_DE.ISO-8859-1`’。可以在编译目录中使用‘`make localedata/install-locales`’命令配置所有可用的 `locale`，但是一般不需要这么做。

时区使用‘`TZ`’环境变量设置。`tzselect` 脚本可以帮助你选择正确的值。设置系统全局范围内的时区可以将 `/etc/localtime` 文件连接到 `/usr/share/zoneinfo` 目录下的正确文件上。比如对于中国人可以‘`ln -s /usr/share/zoneinfo/PRC /etc/localtime`’。

编译优化提示

由于 `Glibc` 是系统的两大核心之一(还有一个是内核)，虽然 `LFS` 指导书反对优化编译，但很多玩家觉得不优化编译心有不甘。因此，下面有几个基于 `gcc-4.0.4 + Binutils-2.17 + Linux-Libc-Headers-2.6.12.0` 环境的实践提示。

能够通过 `make check` 的优化设置：

```
CPPFLAGS= '-DNDEBUG'
CFLAGS= '-O3 -finline-limit=400 -fomit-frame-pointer -falign-functions=32 -pipe -fno-bounds-check
-march=pentium3 -maccumulate-outgoing-args -fforce-addr -fmerge-all-constants -fgcse-sm -fgcse-las
-minline-all-stringops -ftree-loop-linear -fvopts -ftree-vectorize -fprefetch-loop-arrays -fw eb -frenam e-registers
-fbranch-target-load-optimize'
LDFLAGS= '-s -Wl,-O1,--sort-common,-s,--enable-new-dtags,--as-needed'
[提示]建议 AMD64 打开 -frenam e-registers；-freg-struct-return 虽然能通过测试但是却会导致 GCC 的测试程序出现“FAIL:
gcc.dg/struct-ret-libc.c execution test”因此该优化选项未包含在其中。此外，-fbranch-target-load-optimize -
fbranch-target-load-optimize2 不能同时使用。
```

不能通过编译或测试的 CFLAGS : -fv isibility=hidden -m fpm ath=sse -m align-double -m 128bit-long-double -m regpam =3 -m sseregparm -ftracer--param m ax-gcse-passes=2 --param m ax-gcse-m em ory=100M -W a,-R

与 CPU L1/L2 cache 有关的提示[对于 AMD,cache 是L1 (皆为128K)+ L2 (256,512,1M,2M)的大小; 对于 Intel,cache 是L2 的大小]:

cache <= 512K 推荐使用 -fin line-lim it=200 -falign-functions=16
512K < cache <= 1M 推荐使用 -fin line-lim it=400 -falign-functions=32
cache > 1M 推荐使用 -fin line-lim it=600 -falign-functions=64

Binutils 配置选项简介(适用于 2.16/2.17/2.18)

下列选项皆为非默认值[特别说明的除外]
[提示]如果只想编译 ld 可以使用“make all-ld”, 如果只想编译 as 可以使用“make all-gas”。类似的还有 clean-ld clean-as distclean-ld distclean-as check-ld check-as 等。

```
--help
--version
--silent
--cache-file=FILE
--config-cache
--no-create
--srcdir=DIR
--prefix=PREFIX
--exec-prefix=EPREFIX
--bindir=DIR
--sbindir=DIR
--libexecdir=DIR
--datadir=DIR
--sysconfdir=DIR
--sharedstatedir=DIR
--localstatedir=DIR
--libdir=DIR
--includedir=DIR
--oldincludedir=DIR
--infodir=DIR
--mandir=DIR
--sourcedir=DIR
--program-prefix=PREFIX
--program-suffix=SUFFIX
--program-transform-name=PROGRAM
--build=BUILD
--host=HOST
--target=TARGET
```

这些选项的含义基本上通用于所有软件包，这里就不特别讲解了。

```
--with-lib-path=dir1:dir2...
    指定不同于默认目录的库搜索路径，这个工作也可以通过设置 Makefile 中的 LIB_PATH 变量值或使用这个选项完成。如果你想
    要做一个交叉连接器，那么可能需要用 -l 指定不同于默认目录的库搜索路径。

--disable-nls
    禁用本地语言支持(默认为启用)。编译时出现“undefined reference to ‘libintl_gettext’”错误则必须禁用。

--disable-rpath
    不在二进制文件中硬编码库文件的路径

--disable-multilib
    禁止编译适用于多重目标体系的库。

--enable-shared [=PKGS]
--enable-static [=PKGS]
    将指定的 PKGS (逗号 分隔的列表)编译为共享/静态库，目前仅可使用: bfd,opcodes 。shared 在不同子目录下默认值不同，有
    些为“yes”有些为“no”。而 static 在所有子目录下默认值皆为“yes”。[提示]LFS-第五章 pass1,pass2 都应当使用“
    --disable-shared --enable-static”(其他所有可行的设置在本质上效果都与此相同)

--enable-64-bit-bfd
    支持 64 位目标。如果指定的目标是 64 位则此选项默认打开，否则默认关闭(即使 --enable-targets=all 也是如此)。要开启此
    选项，编译器必须支持 64 位整数(比如 gcc)。

--enable-cgen-maint=dir
    编译 cgen 相关的文件[主要用于 GDB 调试]。

--enable-libada
    编译 libada 目录

--enable-libgcc-math
    编译 libgcc-math 目录

--enable-libgcj
    启用 Java 前端库

--enable-libssp
    编译 libssp 目录
```



```
--enable-object-c
    允许在 Objective-C 运行时库中使用 Boehm 垃圾回收器
--disable-werror
    使用 -Werror 来将所有编译器警告当作错误来看待。
--with-gnu-ld
    明确告诉编译器使用的连接器是 GNU ld ，默认为未指定。
--with-gmp=PATH
--with-gmp-dir=PATH
--with-gmp-include=PATH
--with-gmp-lib=PATH
    指定 GMP 库的安装目录/源代码目录/头文件目录/库目录
--with-mpfr=PATH
--with-mpfr-dir=PATH
--with-mpfr-include=PATH
--with-mpfr-lib=PATH
    指定 MPFR 库的安装目录/源代码目录/头文件目录/库目录
--with-included-gettext
    使用软件包中自带的 GNU gettext 库
--with-libiconv-prefix [=DIR]
    在 DIR/include 目录中搜索 libiconv 头文件，在 DIR/lib 目录中搜索 libiconv 库文件。
--with-libintl-prefix [=DIR]
    在 DIR/include 目录中搜索 libintl 头文件，在 DIR/lib 目录中搜索 libintl 库文件。
--with-pic
--without-pic
    试图仅使用 PIC 或 non-PIC 对象，默认两者都使用。
```

以下选项意义不大，一般不用考虑它们

```
--enable-dependency-tracking
    启用常规的依赖性追踪(指的是 Makefile 规则)，允许多次编译。默认禁止依赖性追踪，这样可以加速一次性编译。
--enable-maintainer-mode
    启用无用的 make 规则和依赖性(它们有时会导致混淆)
--disable-fast-install
    禁止为快速安装而进行优化。
--disable-libtool-lock
    禁止 libtool 锁定以加快编译速度(可能会导致并行编译的失败)
--enable-target-optspace
    使用“-Os”而不是“-O2”来编译目标库文件，在某些平台上默认为“-Os”，某些平台上默认为“-O2”。
--enable-bfd-assembler
    简而言之，你不需要明确指定该选项，完全可以忽略它的存在，因此也就不需要理解它的含意。
--enable-commonbfdlib
    编译共享版本的 BFD/opcodes 库。建议不要使用该选项，而由 --enable-shared 来控制。
--enable-install-libbfd
    允许安装 libbfd 以及相关的头文件(libbfd 是二进制文件描述库,用于读写目标文件“.o”,被GDB/ld/as等程序使用)。建议不要使用该选项，而由 --enable-shared/static 来控制。
--enable-install-libiberty
    安装 libiberty 的头文件(libiberty.h)，许多程序都会用到这个库中的函数(getopt, strerror, strtol, strtoul)。这个选项经过实验，无论 enable 还是 disable 都没有实际效果(相当于 disable)。
--enable-checking
    允许 as 执行运行时检查
--enable-bootstrap
    本地编译时默认打开，交叉编译时默认关闭。建议不要明确指定，否则可能会出现配置错误。
--enable-build-warnings
    允许显示编译时的编译器警告
--enable-stage1-checking=all
    对处于 stage1 状态的编译器执行额外的检查。
--enable-stage1-languages=all
    在 stage1 时编译更多的语言支持，该选项一般仅供编译器开发者使用。
--enable-secureplt
    默认创建只读的 plt 项，仅对 powerpc-linux 平台有效。
--with-dochdir=DIR
--with-htmldir=DIR
--with-pdfdir=DIR
    指定各种文档的安装目录
--with-datarootdir
    将 DIR 用作数据根目录，默认值是[PREFIX/share]
--with-mmapi
    使用 mmap 访问文件，某些平台上速度较快，某些平台上速度较慢，某些平台上无法正常工作[很不幸,Linux 是其中之一]。
--with-pkgversion=PKG
    在 bfd 库中使用“PKG”代替“Linux/GNU Binutils”作为版本字符串
--with-stabs
    指定编译程序产生的调试信息为 stabs 格式，而不是宿主系统的默认格式。通常 GCC 产生的默认调试信息是 ECOFF 格式，但是它包含的调试信息没有 stabs 多。某些平台上是默认值，某些平台是非默认值。
```

```
--x-include=DIR
--x-libraries=DIR
    分别指定 X 的头文件和库文件目录。

以下选项仅用于交叉编译环境

--enable-serial[={host,target,build}]configure
    强制为 host, target, build 顺序配置子包，如果使用“all”则表示所有子包。
--enable-targets=TARGET,TARGET,TARGET...
    使其BFD 在默认格式之外再支持多种其它 平台的二进制文件格式，“all”表示所有已知平台。在 32 位系统上，即使使用“all”也只能支持所有 32 位目标，除非同时使用 --enable-64-bit-bfd 选项。由于目前 gas 并不能使用内置的默认平台之外的其它目标，因此这个选项没什么实际意义。
--with-sysroot=dir
    将 dir 看作目标系统的根目录。目标系统的头文件、库文件、运行时对象都将被限定在其中。
--with-cross-host=HOST
    用于配置交叉编译器(反对使用该选项)。
--with-build-subdir=SUBDIR
    为 build 在 SUBDIR 子目录中进行配置
--with-build-libsubdir=DIR
    指定 build 平台的库文件目录
--with-build-sysroot=sysroot
    在编译时将’sysroot’当作指定 build 平台的根目录看待
--with-build-time-tools=path
    在编译过程中使用给定的 path 寻找 target tools。该目录中必须包含 ar, as, ld, nm , ranlib, strip 程序，有时还需要包含 objdump 程序。
--with-target-subdir=SUBDIR
    为 target 在 SUBDIR 子目录中进行配置
--with-headers=DIR
    指定目标平台的头文件目录
--with-libs=DIR
    指定目标平台的库文件目录
--with-newlib
    将’newlib’(另一种标准 C 库)指定为目标系统的 C 库进行使用。当 --with-headers 和 --with-libs 都指定的时候隐含该选项。
```

GCC 配置选项简介(适用于 3.4/4.0/4.1/4.2/4.3)

推荐用一个新建的目录来编译GCC，而不是在源码目录中，这一点玩过LFS的兄弟都很熟悉了。另外，如果先前在编译中出现了错误，推荐使用 make distclean 命令进行清理，然后重新运行 configure 脚本进行配置，再进行编译。

[注意]这里仅包含适用于 C/C++ 语言编译器、十进制数字扩展库(libdecnumber)、GOMP 库(libgomp)、大杂烩的 libliberty 库、执行运行时边界检查的库 (libmudflap)、保护堆栈溢出的库(libssp)、标准 C++ 库(libstdc++) 相关的选项。也就是相当于 gcc-core 与 gcc-g++ 两个子包的选项。并不包括仅仅适用于 Ada Fortran Java Objective C 语言的选项。

每一个 --enable 选项都有一个对应的 --disable 选项，同样，每一个 --with 选项也都用一个对应的 --without 选项。每一对选项中必有一个是默认值(依赖平台的不同而不同)。下面所列选项若未特别说明皆为非默认值。

```
--cache-file=FILE
--help
--no-create
--quiet
--silent
--version
--prefix=PREFIX
--exec-prefix=EPREFIX
--bindir=DIR
--sbindir=DIR
--libexecdir=DIR
--datadir=DIR
--sysconfdir=DIR
--sharedstatedir=DIR
--localstatedir=DIR
--libdir=DIR
--includedir=DIR
--oldincludedir=DIR
--infodir=DIR
--mandir=DIR
--sourcedir=DIR
--program-prefix=PREFIX
--program-suffix=SUFFIX
--program-transform-name=PROGRAM
--build=BUILD
--host=HOST
```

```
--target= TARGET
    这些选项的含义基本上通用于所有软件包，这里就不特别讲解了。
--with-local-prefix= DIR
    编译程序用来查找安装在本地的包含文件目录的前缀，默认为 /usr/local。只有在系统已经建立了某些特定的目录规则，而不再是在 /usr/local/include 中查找本地安装的头文件的时候，该选项才使必须的。不能指定为 /usr，也不能指定为包含系统标准头文件的目录，因为安装的头文件会和系统的头文件混合，从而造成冲突，导致不能编译某些程序。
--enable-languages= lang1, lang2, ...
    指定只安装指定的语言支持，可以使用的语言是：all (默认值，等价于“c, c++, fortran, java, objc”), ada, c, c++, fortran, java, objc, objc++, treelang，若不指定该选项则安装默认装所有可用的语言。
--enable-bootstrap
--disable-bootstrap
    允许或禁止 bootstrap。非交叉编译的情况下，3.4/4.0/4.1 版本 disable 是默认值，4.2/4.3 版本 enable 是默认值；交叉编译的情况下，disable 是默认值。
--enable-checking [= LIST]
    该选项会在编译器内部生成一致性检查的代码，它并不改变编译器生成的二进制结果。这样导致编译时间增加，并且仅在使用 GCC 作为编译器的时候才有效，但是对输出结果没有影响。对从 CVS 下载的版本默认值是“yes”(= assert, misc, tree, gc, rtlflag, runtime)，对于正式发布的版本则是“release”(= assert, runtime)，其他值还有 no 和 all。可以从 “assert, df, fold, gc, gcac, misc, rtlflag, rtl, runtime, tree, valgrind, types” 中选择 你想要检查的项目 (逗号隔开的列表)，其中 rtl, gcac, valgrind 非常耗时。使用 --disable-checking 完全禁止这种检查会增加未能检测内部错误的风险，所以不建议这样做。
--enable-largelfile
    启用大文件支持
--disable-nls
    禁用本地语言支持 (它允许按照非英语的本地语言显示警告和错误消息)。
--disable-rpath
    不硬编码运行时库的路径。
--disable-multilib
    禁止编译适用于多重目标体系的库。
--enable-targets= all
--enable-targets= target_list
    对于 powerpc64 和 x86_64 来说，GCC 将生成复合体系结构的编译器，也就是说既可以生成 64 位代码，也可以生成 32 位代码。而在其相应的 32 位平台 (powerpc, x86) 上，则只能生成 32 位代码。这个选项使得在 32 位平台上也可以生成 64 位代码。当前此选项仅支持 powerpc-linux 和 x86-linux。
--disable-cpp
    不安装用户可见的 cpp 程序。
--enable-shared [= PKG [...]]
--enable-static [= PKG [...]]
    编译共享/静态版本的库，默认值“yes”表示全部可用的库。当前可用的库名称有：libgcc, libcpp, libdecnumber, libgomp, libmudflap, libssp, libstdc++, libobjc, ada, libada, libgnat, libf2c, libgfortran, Boehm-gc, fastjar, libffi, libjava, zlib。另外，libiberty 和 libsupc++ 仅支持作为静态库。
--disable-libada
    禁止编译 libada 库
--disable-libgcj
    不编译 Java 的垃圾收集器 libgcj 库
--disable-libgomp
    禁止编译 libgomp 库
--disable-libmudflap
    不编译边界检查函数的运行时库。
--disable-libssp
    不编译保护缓冲区溢出的运行时库。
--disable-libdecnumber
    禁止编译 libdecnumber 库 [在我的机器上总是导致 GCC-4.2 编译失败]
--disable-libiberty
    禁止编译 libiberty 库 [在我的机器上总是导致 GCC-4.2 编译失败]
--enable-decimal-float [= bid | dpd]
--disable-decimal-float
    启用或禁用 libdecnumber 库的 C 语言十进制浮点扩展，还可以进一步选择浮点格式 bid 是 i386 与 x86_64 的默认值 dpd 是 PowerPC 的默认值)。在 PowerPC/i386/x86_64 GNU/Linux 系统默认启用，在其他系统上默认禁用。
--enable-install-libiberty
    安装 libiberty 的头文件 (libiberty.h)，许多程序都会用到这个库中的函数 (getopt, strerror, strtol, strtoul)。
--enable-linux-futex
    使用 Linux 的 futex 系统调用 (快速用户空间互斥体)，默认自动检测，建议明确启用。
--enable-threads [= posix | aix | dce | gnat | mach | rtems | posix95 | solaris | vxworks | win32 | nks]
    启用线程支持，这是默认值。还可以进一步指定线程模型 (不同平台支持的线程模型并不相同)。
--enable-tls
    指定目标系统支持 TLS (线程本地存储)，一般情况下不需要明确指定这个选项，因为 configure 脚本可以自动检测。仅在你认为检测不正确的情况下 (比如汇编器支持 TLS 但 C 却不支持或汇编器检测错误) 才使用这个选项明确指定。
--enable-symvers [= gnu | gnu-versioned-name-space | darwin | darwin-export]
--disable-symvers
    启用或禁用共享库对象中符号包含的版本信息。使用这个选项将导致 ABI 发生改变。禁用版本信息可以减小库的体积，但是将不兼容依赖于老版本库的二进制程序。[提示] --disable-symvers 可能会导致 libstdc++ 的 abi_check 失败。
--with-arch= cpu
    指定将来调用 gcc 时 -m arch 选项的默认值。
```

`--w ith-long-double-128`
指定 `long double` 类型为 `128-bit`。基于 `Glibc 2.4` 或以上版本编译时默认为 `128-bit`，其他情况默认为 `64-bit`。

`--w ith-gnu-as`
`--w ith-gnu-ld`
指定编译器使用的为 `GNU` 汇编器/连接器，如何你实际使用的不是 `GNU` 汇编器/连接器，指定这个选项会引起混淆；另一方面如果你实际使用的是 `GNU` 汇编器/连接器，但是却没有指定这个选项，也有可能造成混淆。

`--w ith-as=pathnam e`
`--w ith-ld=pathnam e`
指定编译器使用的汇编器/连接器的位置，必须使用绝对路径。如果 `configure` 的默认查找过程找不到汇编器/连接器，就会需要该选项。或者系统中有多个汇编器/连接器，也需要它来指定使用哪一个。如果使用 `GNU` 的汇编器，那么你必须同时使用 `GNU` 连接器。

`--w ith-included-gettext`
如果启用了 `NLS`，该选项指出构建进程在使用系统安装的版本以前，优先使用自己的 `gettext` 副本。

`--w ith-libiconv-prefix=DIR`
使用系统中已经安装的 `libiconv` 库，并在 `DIR/include` 目录中搜索 `libiconv` 头文件，在 `DIR/lib` 目录中搜索 `libiconv` 库文件。

`--w ith-libintl-prefix=DIR`
使用系统中已经安装的 `libintl` 库，并在 `DIR/include` 目录中搜索 `libintl` 头文件，在 `DIR/lib` 目录中搜索 `libintl` 库文件。

`--w ith-gmp[=PATH]`
`--w ith-gmp-dir=PATH`
`--w ith-gmp-include=PATH`
`--w ith-gmp-lib=PATH`
指定 `GMP` 库的安装目录/源代码目录/头文件目录/库目录

`--w ith-mpfr[=PATH]`
`--w ith-mpfr-dir=PATH`
`--w ith-mpfr-include=PATH`
`--w ith-mpfr-lib=PATH`
指定 `MPFR` 库的安装目录/源代码目录/头文件目录/库目录

`--w ith-system-libunwind`
使用系统中已经安装的 `libunwind` 库

`--w ith-system-zlib`
使用系统中已经安装的 `libz` 库

`--w ith-pic`
`--w ithout-pic`
试图仅使用 `PIC` 或 `non-PIC` 对象，默认两者都使用。

`--w ith-x`
使用 `X Window` 系统

以下选项仅适用于 `C++` 语言[“`libstdc++-v3/docs/html/configopts.html`”很值得参考]:

`--enable-cstd io`
使用目标平台特定的 `I/O` 包，等价于“`--enable-cstd io=std io`”。使用这个选项将导致 `ABI` 接口发生改变。更多信息可以参看源码树下的“`libstdc++-v3/docs/html/explanations.html#cstd io`”文档。

`--disable-hosted-libstdcxx`
默认编译特定于主机环境的 `C++` 库。使用该选项将仅编译独立于主机环境的 `C++` 运行时库，这个库是默认情况的一个子集。

`--enable-locale=gnu|ieee_1003.1-2001|generic`
指定目标系统的 `locale` 模块，默认值自动检测。建议明确设为“`gnu`”，否则可能会编译出 `ABI` 不兼容的 `C++` 库。

`--enable-__cxa_atexit`
用 `__cxa_atexit` 代替 `atexit` 来登记 `C++` 对象的本地静态和全局析构函数以符合标准对析构函数的处理规定，相当于在将来调用 `gcc` 时默认使用 `-fuse-cxa-exit` 选项。它还会影响到 `C++` `ABI`，因此生成的 `C++` 共享库在其他的 `Linux` 发行版上也能使用。该选项仅在使用 `Glibc` 的情况下有效。

`--enable-cxx-flags=FLAG S`
编译 `libstdc++` 库文件时传递给编译器的编译标志，是一个引号界定的字符串。默认为空，表示使用环境变量 `CXXFLAGS` 的值。

`--enable-libstdcxx-debug`
额外编译调试版本的 `libstdc++` 库文件，并默认安装在 `${libdir}/debug` 目录中。

`--enable-libstdcxx-debug-flags=FLAG S`
编译调试版本的 `libstdc++` 库文件时使用的编译器标志，默认为“`-g3 -O0`”

`--enable-sjlj-exceptions`
强制使用旧式的 `setjmp/longjmp` 异常处理模型，使用这个选项将导致 `ABI` 接口发生改变。默认使用可以大幅降低二进制文件尺寸和内存占用的新式的 `libunwind` 库进行异常处理。

`--enable-libstdcxx-allocator=new|malloc|m t|pool|b itm ap`
指定目标平台特定的底层 `std::allocator`，默认自动检测。使用这个选项将导致 `ABI` 接口发生改变。

`--enable-concept-checks`
打开额外的实例化库模板编译时检查(以特定的模板形式)，这可以帮助用户在他们的程序运行之前就发现这些程序在何处违反了 `STL` 规则。

`--enable-fully-dynamic-string`
该选项启用了特殊版本的 `basic_string` 来禁止在预处理的静态存储区域中放置空字符串的优化手段。参见 `PR libstdc++/16612` 获取更多细节。

`--disable-long-long`
禁止使用模板支持 `'long long'` 类型。`'long long'` 是 `C99` 新引进的类型，也是 `GNU` 对 `C++98` 标准的一个扩展。该选项将导致 `ABI` 接口发生改变。

`--disable-c-mbchar`
`--disable-wchar_t`
禁止使用模板支持多字节字符类型 `'wchar_t'`。前者用于 `3.4/4.0` 版本，后者用于 `4.1/4.2/4.3` 版本。该选项将导致 `ABI` 接口发

生改变。

`--disable-c99`
禁止支持 C99 标准。该选项将导致 ABI 接口发生改变。

`--enable-initfini-array`
为构造函数和析构函数强制使用 `.init_array` 和 `.fini_array` (而不是 `.init` 和 `.fini`) 节。一般不需要指定该选项, 因为 `configure` 会自动检测。

`--disable-visibility`
禁止 `-fvisibility` 编译器选项的使用 (使其失效)。

`--enable-headers=c|c_std|c_global`
为 g++ 创建 C 语言兼容的头文件, GCC-4.3 默认为“`c_global`”, 其他版本默认为“`c_std`”。具体含义参见源码树下的 `libstdc++-v3/include/README` 文件。

`--disable-libstdcxx-pch`
禁止创建预编译的 `libstdc++` 头文件(`stdc++.h.gch`), 这个文件包含了所有标准 C++ 的头文件。

`--with-gxx-include-dir=dir`
G++ 头文件的安装目录, 默认为 `prefix/include/c++/版本`。

以下选项仅用于交叉编译:

`--enable-serial=[{host,target,build}]configure`
强制为 `host, target, build` 顺序配置子包, 如果使用“`all`”则表示所有子包。

`--with-sysroot=DIR`
将 `dir` 看作目标系统的根目录。目标系统的头文件、库文件、运行时对象都将被限定在其中。

`--with-build-sysroot=sysroot`
在编译时将“`sysroot`”当作 `build` 平台的根目录看待

`--with-build-subdir=SUBDIR`
为 `build` 在 `SUBDIR` 子目录中进行配置

`--with-build-libsubdir=DIR`
指定 `build` 平台的库文件目录

`--with-build-time-tools=path`
指定编译 GCC 自身时目标系统的工具链(汇编器,连接器,等等)目录。该目录中必须包含 `ar, as, ld, nm, ranlib, strip` 程序, 有时还需要包含 `objdump` 程序。

`--with-target-subdir=SUBDIR`
Configuring in a subdirectory for target

`--with-cross-host=HOST`
用于为 `HOST` 配置交叉编译器 (反对使用该选项)。

`--with-headers=DIR`

`--with-libs=DIR`
指定目标平台的头文件和库的目录。

`--with-newlib`
将“`newlib`”指定为目标系统的 C 库进行使用。这将导致 `libgcc.a` 中的 `__eprintf` 被忽略, 因为它被假定为由“`newlib`”提供。如果同时指定了 `with-headers` 和 `--with-libs` 那么就隐含指定了这个选项。

以下选项意义不大, 一般不用考虑它们:

`--enable-stage1-checking [=all]`

`--enable-stage1-languages [=all]`
选择在 `bootstrap stage1` 的过程中执行额外检查的包/语言。

`--enable-dependency-tracking`
启用常规的依赖性追踪 (指的是 `Makefile` 规则), 允许多次编译。默认禁止依赖性追踪, 这样可以加速一次性编译。

`--enable-maintainer-mode`
启用无用的 `make` 规则和依赖性 (它们有时会导致混淆), 要求从源代码重新生成 `gcc.pot` 文件。该文件是主消息分类, 包括编译程序产生的所有错误和警告的诊断消息, 普通用户不需要了解这些信息。仅在你改动过源代码的情况下才有可能需要使用这个选项。要使该选项正常工作, 需要完整的源代码树个 `gettext` 的最新版本。

`--disable-fast-install`
禁止为快速安装而进行优化。

`--disable-libtool-lock`
禁止 `libtool` 锁定以加快编译速度 (可能会导致并行编译的失败)

`--enable-target-optspace`
指定目标库应当应当优化尺寸而不是速度。仅在 `m32r` 平台上是默认值。

`--enable-coverage [=opt|noopt]`
在编译器每次编译时收集自身的 `coverage` 信息。这个仅用于内部测试的目的, 并且仅在使用 GCC 编译的时候才有效。参数控制着是否在编译编译器时使用优化 (默认为“`noopt`”)。在需要进行 `coverage` 分析的时候使用“`noopt`”, 在需要进行性能分析的时候使用“`opt`”。

`--enable-gather-detailed-mem-stats`
收集详细的内存使用信息, 将来在调用 `gcc` 时如果使用了 `-fmem-report` 选项就可以打印这些信息。

`--enable-generated-files-in-srcdir`
将生成的文件的副本保存在源代码目录中, 以便于没有 `texinfo, perl, bison, flex` 的用户创建源代码的归档。

`--enable-intmodule`
仅用一步来编译 `compiler??`

`--enable-fixed-point`

`--disable-fixed-point`
启用或禁用 C 定点浮点运算 (`fixed-point arithmetic`), 这是一种非常快速的模拟浮点运算的方法, 特别是在没有 FPU 部件的处

理器 (比如 M IPS)上。在没有硬件浮点单元的 cpu 上默认开启，其它默认关闭。

--enable-mapped-location
location_t is file line integer cookie (啥意思?知道的告诉我一声)

--enable-secureplt
指定将来调用 gcc 时默认使用 -m secure-plt 选项，仅对 powerpc-linux 平台有效。

--enable-tls
使用一个单独的过程来应用各种 fixes (啥意思?知道的告诉我一声)

--enable-version-specific-runtime-libs
将运行时库安装在编译器特定的子目录 (libdir/gcc)中。另外，'libstdc++'的头文件将被安装在 libdir 中 (除非指定了 -w ith-gxx-include-dir)。如果你打算同时安装几个不同版本的 GCC，这个选项就很有用处了。当前，'libgomp'，'libgfortran'，'libjava'，'libmudflap'，'libstdc++'，'libobjc'都支持该选项。

--enable-werror-always
--enable-werror
前者表示全程使用 -W error 标志编译。后者表示在 bootstrap stage2 及之后使用 -W error 标志编译。

--with-cpp-install-dir=DIR
除了将用户可见的 cpp 程序安装到默认的 PREFIX/bin 目录外，还将安装到 prefix/DIR 目录。

--with-slibdir=DIR
共享库的安装目录，默认等于 --libdir 的值。

--with-datarootdir=DIR
将 DIR 用作数据根目录，默认值是[PREFIX/share]

--with-docdir=DIR
--with-html-dir=DIR
--with-pdfdir=DIR
指定各种特定格式的文档的安装目录。

--with-stabs
指定编译程序产生的调试信息为 stabs 格式，而不是宿主系统的默认格式。通常 GCC 产生的默认调试信息是 ECOFF 格式，但是它包含的调试信息没有 stabs 多。

--with-dwarf2
指定编译程序产生的调试信息默认为 DWARF2 格式。

--with-gc[=simple|page|zone]
指定编译过程中使用的垃圾回收方案 (默认为"page")。

--with-demangler-in-ld
尝试在 GNU ld 中使用 demangler (啥意思?)

--with-debug-prefix-map='A=B C=D ...'
map A to B, C to D ... in debug information

--x-include=DIR
--x-libraries=DIR
指定 X 的头文件和库的所在目录

一些补充说明

除了使用 CFLAGS,LDFLAGS 之外，还可以使用 LIBCFLAGS,LIBCXXFLAGS 控制库文件 (由 stage3 编译)的编译器选项。可以在 make 命令行上使用 BOOT_CFLAGS,BOOT_LDFLAGS 来控制 stage2,stage3 的编译。可以使用 make bootstrap4 来增加步骤以避免 stage1 可能被错误编译所导致的错误。可以使用 make profiledbootstrap 在编译 stage1 时收集一些有用的统计信息，然后使用这些信息编译最终的二进制文件，这样可以提升编译器的执行效率。