Glibc Binutils GCC 安装指南

作者：金步国

版权声明

其他作品

本文作者十分愿意与他人分享劳动成果，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的作品集：

- 金步国作品集 [http://www.jinbuguo.com/]

联系方式

由于作者水平有限，因此不能保证作品内容准确无误。如果你发现了作品中的错误(哪怕是错别字也好)，请来信指出，任何提高作品质量的建议我都将虚心接纳。

- Email(QQ)：70171448 在 QQ 邮箱

Glibc 安装指南(2.6.1 → 2.9)

安装信息的来源

- 源码包内的下列文件：configure FAQ INSTALL NOTES Makeconfig README localedata/README
- http://www.gnu.org/software/libc/manual/html_node/System-Configuration.html
- http://www.gnu.org/software/libc/manual/html_node/Installation.html
- http://www.gnu.org/software/libc/manual/html_node/Name-Service-Switch.html

要点提示

编译 Glibc 的时候应该尽可能使用最新的内核头文件，至少要使用 2.6.16 以上版本的内核，先前的版本有一些缺陷会导致"make check"时一些与 pthreads 测试相关的项目失败。使用高版本内核

头文件编译的 Glibc 二进制文件完全可以运行在较低版本的内核上，并且当你升级内核后新内核的特性仍然可以得到充分发挥而无需重新编译 Glibc。但是如果编译时使用的头文件的版本较低，那么运行在更高版本的内核上时，新内核的特性就不能得到充分发挥。更多细节可以查看[八卦故事]内核头文件传奇的跟帖部分。

推荐使用 GCC-4.1 以上的版本编译，老版本的 GCC 可能会生成有缺陷的代码。

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。

源码树下的 Makeconfig 文件中有许多用于特定目的的变量，你可以在编译目录下创建一个 configparms 文件来改写这些变量。执行 make 命令的时候 configparms 文件中的内容将会按照 Makefile 规则进行解析。比如可以通过在其中设置 CFLAGS LDFLAGS 环境变量来优化编译，设置 CC BUILD_CC AR RANLIB 来指定交叉编译环境。

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
--quiet
--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
--docdir=DIR
--htmldir=DIR
--dvidir=DIR
--pdfdir=DIR
--psdir=DIR

--build=BUILD
--host=HOST

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。需要注意的是：没有--target=TARGET 选项。

--prefix=PREFIX

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，如果在 Linux 平台上指定 --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，同时将使用 /etc 作为配置目录，也就是等价于"slibdir=/lib sysconfdir=/etc"。但是如果保持默认值或指定其他目录，那么所有组件都间被安装到 PREFIX 目录下。

--disable-sanity-checks

真正禁用线程(仅在特殊环境下使用该选项)。

--enable-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。

--disable-shared

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 。[提示] --disable-static 选项实际上是不存在的，静态库总是被无条件的编译和安装。

--enable-profile

启用 profiling 信息相关的库文件编译。主要用于调试目的。

--enable-omitfp

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)。

--enable-bounded

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。

--disable-versioning

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。

--enable-oldest-abi=ABI

启用老版本的应用程序二进制接口支持。ABI 是 Glibc 的版本号，只有明确指定版本号时此选项才有效。

--enable-stackguard-randomization

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。

--enable-add-ons[=DIRS...]

为了减小软件包的复杂性，一些可选的 libc 特性被以单独的软件包发布，比如'linuxthreads'(现在已经被废弃了)，他们被称为'add-ons'。要使用这些额外的包，可以将他们解压到 Glibc 的源码树根目录下，然后使用此选项将 DIR1,DIR2,...中的附加软件包包含进来。其中的"DIR"是附加软件包的目录名。默认值"yes"表示编译所有源码树根目录下找到的附加软件包。

--disable-hidden-plt

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。

--enable-bind-now

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。

--enable-static-nss

编译静态版本的 NSS(Name Service Switch)库。仅在/etc/nsswitch.conf 中只使用 dns 和 files 的情况下，NSS 才能编译成静态库，并且你还需要在静态编译应用程序的时候明确的连接所有与 NSS 库相关的库才行[比如：gcc -static test.c -o test -Wl,-lc,-lnss_files,-lnss_dns,-lresolv]。不推荐使用此选项，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。

--disable-force-install

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)。

--enable-kernel=VERSION

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。

--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-gd
--with-gd-include
--with-gd-lib

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。

--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！

--with-binutils=DIR

明确指定编译时使用的 Binutils(as,ld)所在目录。

--with-elf

指定使用 ELF 对象格式，默认不使用。建议在支持 ELF 的 Linux 平台上明确指定此选项。

--with-selinux
--without-selinux

启用/禁用 SELinux 支持，默认值自动检测。

--with-xcoff

使用 XCOFF 对象格式(主要用于 windows)。

--without-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。

--with-headers=DIR

指定内核头文件的所在目录，在 Linux 平台上默认是'/usr/include'。

--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。

--without-__thread

即使平台支持也不使用 TSL 特性。建议不要使用该选项。

--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 。

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'。

Binutils 安装指南(2.18 → 2.19.1)

安装信息的来源

- 源码包内的下列文件：各级目录下的 configure 脚本　README {bfd,binutils,gas,gold,libiberty}/README

要点提示

如果想与 GCC 联合编译，那么可以将 binutils 包的内容解压到 GCC 的源码目录中(tar -xvf binutils-2.19.1.tar.bz2 --strip-components=1 -C gcc-4.3.3)，然后按照正常编译 GCC 的方法编译即可。这样做的好处之一是可以完整的将 GCC 与 Binutils 进行一次 bootstrap。

推荐用一个新建的目录来编译，而不是在源码目录中。编译完毕后可以使用"make check"运行测试套件。这个测试套件依赖于 DejaGnu 软件包，而 DejaGnu 又依赖于 expect，expect 依赖于 tcl。

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。

配置选项

下列选项皆为*非默认值*[特别说明的除外]

--help
--version
--quiet
--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
--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。

--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)。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。例如，在 x86_64 平台上，默认既可以生成 64 位代码，也可以生成 32 位代码，若使用此选项，那么将只能生成 64 位代码。

--enable-cgen-maint=CGENDIR

编译 cgen 相关的文件[主要用于 GDB 调试]。

--enable-shared[=PKG[,...]]
--disable-shared
--enable-static[=PKG[,...]]
--disable-static

允许/禁止编译共享或静态版本的库和可执行程序，全部可识别的 PKG 如下：

binutils,gas,gprof,ld,bfd,opcodes,libiberty(仅支持作为静态库)。static 在所有目录下的默认值都是"yes"；而 shared 在不同子目录下默认值不同，有些为"yes"(binutils,gas,gprof,ld)有些为"no"(bfd,opcodes,libiberty)。

--enable-install-libbfd
--disable-install-libbfd

允许或禁止安装 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)。本地编译或指定--enable-shared 的情况下默认值为"yes"，否则默认值为"no"。

--enable-64-bit-bfd

让 BFD 支持 64 位目标，如果希望在 32 位平台上编译 64 程序就需要使用这个选项。如果指定的目标(TARGET)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)。

--enable-elf-stt-common

允许 BFD 生成 STT_COMMON 类型的 ELF 符号。[2.19 版本新增选项]

--enable-checking
--disable-checking

允许 as 执行运行时检查。正式发布版本默认禁用，快照版本默认启用。

--disable-werror

禁止将所有编译器警告当作错误看待(因为当编译器为 GCC 时默认使用-Werror)。

--enable-got=target|single|negative|multigot

指定 GOT 的处理模式。默认值是"target"。[2.19 版本新增选项]

--enable-gold

使用 gold 代替 GNU ld。gold 是 Google 开发的连接器，2008 年捐赠给 FSF，目的是取代现有的 GNU ld，但目前两者还不能完兼容。[2.19 版本新增选项]

--enable-plugins

启用 gold 连接器的插件支持。[2.19 版本新增选项]

--enable-threads

编译多线程版本的 gold 连接器。[2.19 版本新增选项]

--with-lib-path=dir1:dir2...

指定编译出来的 binutils 工具(比如：ld)将来默认的库搜索路径，在绝大多数时候其默认值是"/lib:/usr/lib"。这个工作也可以通过设置 Makefile 中的 LIB_PATH 变量值完成。

--with-libiconv-prefix[=DIR]
--without-libiconv-prefix

在 DIR/include 目录中搜索 libiconv 头文件，在 DIR/lib 目录中搜索 libiconv 库文件。或者根

本不使用 libiconv 库。

--with-libintl-prefix[=DIR]
--without-libintl-prefix

在 DIR/include 目录中搜索 libintl 头文件，在 DIR/lib 目录中搜索 libintl 库文件。或者根本不使用 libintl 库。

--with-mmap

使用 mmap 访问 BFD 输入文件。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。

--with-pic
--without-pic

试图仅使用 PIC 或 non-PIC 对象，默认两者都使用。

以下选项仅在与 GCC 联合编译时才有意义，其含义与 GCC 相应选项的含义完全一样，默认值也相同。

--enable-bootstrap
--disable-bootstrap

--enable-languages=lang1,lang2,...

--enable-stage1-checking

--enable-stage1-languages

--disable-libada

--disable-libgcj

--disable-libgomp

--disable-libmudflap

--disable-libssp

--enable-objc-gc

--disable-cloog-version-check

--disable-ppl-version-check

--with-gnu-as
--with-gnu-ld

--with-gmp=GMPDIR
--with-gmp-include=GMPINCDIR
--with-gmp-lib=GMPLIBDIR

--with-mpfr=MPFRDIR
--with-mpfr-include=MPFRINCDIR
--with-mpfr-lib=MPFRLIBDIR

--with-cloog=CLOOGDIR
--with-cloog_include=CLOOGINCDIR
--with-cloog_lib=CLOOGLIBDIR

--with-ppl=PPLDIR
--with-ppl_include=PPLINCDIR
--with-ppl_lib=PPLLIBDIR

--with-stabs

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。

--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-target-subdir=SUBDIR

为 target 在 SUBDIR 子目录中进行配置。

--with-newlib

将'newlib'(另一种标准 C 库，主要用于嵌入式环境)指定为目标系统的 C 库进行使用。

--with-build-sysroot=sysroot

在编译时将'sysroot'当作指定 build 平台的根目录看待。仅在已经使用了--with-sysroot 选项的时候，该选项才有意义。

--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。默认值是 SUBDIR。

--with-build-time-tools=path

在给定的 path 中寻找用于编译 Binutils 自身的目标工具。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。例如，当编译 Binutils 的系统的文件布局和将来部署 Binutils 的目标系统不一致时就需要使用此选项。

--with-cross-host=HOST

这个选项已经被反对使用，应该使用--with-sysroot 来代替其功能。

以下选项意义不大，一般不用考虑它们

--disable-dependency-tracking

禁止对 Makefile 规则的依赖性追踪。

--disable-largefile

禁止支持大文件。[2.19 版本新增选项]

--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)

--disable-build-warnings

禁止显示编译时的编译器警告，也就是使用"-w"编译器选项进行编译。

--disable-fast-install

禁止为快速安装而进行优化。

--enable-maintainer-mode

启用无用的 make 规则和依赖性(它们有时会导致混淆)

--enable-commonbfdlib
--disable-commonbfdlib

允许或禁止编译共享版本的 BFD/opcodes/libiberty 库。分析 configure 脚本后发现这个选项事实上没有任何实际效果。

--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，没有实际效果(相当于 disable)。

--enable-secureplt

使得 binutils 默认创建只读的 plt 项。相当于将来调用 gcc 时默认使用 -msecure-plt 选项。仅对 powerpc-linux 平台有意义。

--enable-targets=TARGET,TARGET,TARGET...

使 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。此选项在所有目录下都没有默认值。但对于 2.19 版本，此选项在 gold 子目录下的默认值是"all"。

--with-bugurl=URL
--without-bugurl

指定发送 bug 报告的 URL/禁止发送 bug 报告。默认值是"http://www.sourceware.org/bugzilla/"。

--with-datarootdir=DATADIR

将 DATADIR 用作数据根目录，默认值是[PREFIX/share]

--with-docdir=DOCDIR
--with-htmldir=HTMLDIR
--with-pdfdir=PDFDIR

指定各种文档的安装目录。DOCDIR 默认值的默认值是 DATADIR，HTMLDIR 和 PDFDIR 的默认值是 DOCDIR。

--with-included-gettext

使用软件包中自带的 GNU gettext 库。如果你已经使用了 Glibc-2.0 以上的版本，或者系统中已经安装了 GNU gettext 软件包，那么就没有必要使用这个选项。默认不使用。

--with-pkgversion=PKG

在 bfd 库中使用"PKG"代替默认的"GNU Binutils"作为版本字符串。比如你可以在其中嵌入编译时间或第多少次编译之类的信息。

--with-separate-debug-dir=DIR

在 DIR 中查找额外的全局 debug 信息，默认值：${libdir}/debug

--with-debug-prefix-map='A=B C=D ...'

在调试信息中建立 A-B,C-D, ... 这样的映射关系。默认为空。[2.19 版本新增选项]

环境变量

tooldir

可执行文件的安装目录。其默认值是"$(exec_prefix)/$(target_alias)"。

GCC 安装指南(4.3 → 4.4)

安装信息的来源

- 源码包内的下列文件：各级目录下的 configure 脚本　ABOUT-NLS　gcc/testsuite/README*　INSTALL/*　libstdc++-v3/docs/html/manual/bk01pt01ch02.html
- http://gcc.gnu.org/faq.html
- http://gcc.gnu.org/install/
- http://gcc.gnu.org/onlinedocs/libstdc++/faq.html
- http://gcc.gnu.org/onlinedocs/libstdc++/manual/bk01pt01ch02.html

要点提示

从 GCC-4.3 起，安装 GCC 将依赖于 GMP-4.1 以上版本和 MPFR-2.3.2 以上版本。如果将这两个软件包分别解压到 GCC 源码树的根目录下，并分别命名为"gmp"和"mpfr"，那么 GCC 的编译程序将自动将两者与 GCC 一起编译。建议尽可能使用最新的 GMP 和 MPFR 版本。

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，再在另外一个空目录中进行编译。

如果想要安装 C++编译器，那么 libstdc++ 将要求系统的 C 库必须至少带有 de_DE locale 支持，如果使用了 --enable-clocale=gnu 配置选项(很可能就是默认值)，那么还需要下列 locale：

```
locale              字符集
-------------------------------------
```

```
de_DE                ISO-8859-1
de_DE@euro           ISO-8859-15
en_HK                ISO-8859-1
en_PH                ISO-8859-1
en_US                ISO-8859-1
en_US.ISO-8859-1     ISO-8859-1
en_US.ISO-8859-15    ISO-8859-15
en_US.UTF-8          UTF-8
es_ES                ISO-8859-1
es_MX                ISO-8859-1
fr_FR                ISO-8859-1
fr_FR@euro           ISO-8859-15
is_IS                UTF-8
it_IT                ISO-8859-1
ja_JP.eucjp          EUC-JP
se_NO.UTF-8          UTF-8
ta_IN                UTF-8
zh_TW                BIG5
```

不过，这些 locale 并非严格必须，即使缺少上述 locale ，C++编译器也不会失效，只是 libstdc++ 就不能提供"named locale"特性了，并且测试程序也会跳过与此相关的测试。

配置选项

[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)、在多处理机上编写并行程序的应用编程接口 GOMP 库(libgomp)、大杂烩的 libiberty 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其他语言的选项。

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*。

--help
--version
--quiet
--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
--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。

--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)。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。

--enable-bootstrap
--disable-bootstrap

"bootstrap"的意思是用第一次编译生成的程序来第二次编译自己，然后又用第二次编译生成的程序来第三次编译自己，最后比较第二次和第三次编译的结果，以确保编译器能毫无差错的编译自身，这通常表明编译是正确的。非交叉编译的情况下 enable 是默认值；交叉编译的情况下，disable 是默认值。提示：stage2 出来的结果是"最终结果"。

--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。在"gcc"子目录下，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)，在"libgcc"子目录下，默认值始终是"no"。可以从"assert,df,fold,gc,gcac,misc,rtlflag,rtl,runtime,tree,valgrind"中选择你想要检查的项目(逗号隔开的列表，"all"表示全部)，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-languages=lang1,lang2,...

只安装指定语言的编译器及其运行时库，可以使用的语言是：ada, c, c++, fortran, java, objc, obj-c++ ，若不指定则安装所有默认可用的语言(ada 和 obj-c++为非默认语言)。

--disable-multilib

禁止编译适用于多重目标体系的库。例如，在 x86_64 平台上，编译器默认既可以生成 64 位代码，也可以生成 32 位代码，若使用此选项，那么将只能生成 64 位代码。

--enable-shared[=PKG[,...]]
--disable-shared
--enable-static[=PKG[,...]]
--disable-static

允许/禁止编译共享或静态版本的库，全部可识别的库如下：libgcc,libstdc++,libffi,zlib,boehm-gc,ada,libada,libjava,libobjc,libiberty(仅支持作为静态库)。static 在所有目录下的默认值都是"yes"；shared 除了在 libiberty 目录下的默认值是"no"外，在其它目录下的默认值也都是"yes"。

--enable-decimal-float[=bid|dpd]
--disable-decimal-float

启用或禁用 libdecnumber 库符合 IEEE 754-2008 标准的 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。

--disable-libgomp

不编译在多处理机上编写并行程序的应用编程接口 GOMP 库(libgomp)。

--disable-libmudflap

不编译执行运行时边界检查的库(libmudflap)。

--disable-libssp

不编译保护缓冲区溢出的运行时库。

--disable-symvers

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。它还会导致 libstdc++ 的 abi_check 测试失败，但你可以忽略这个失败。

--enable-threads=posix|aix|dce|gnat|mach|rtems|solaris|vxworks|win32|nks
--disable-threads

启用或禁用线程支持，若启用，则必须同时明确指定线程模型(不同平台支持的线程库并不相同，Linux 现在一般使用 posix)。这将对 Objective-C 编译器、运行时库，以及 C++/Java 等面向对象语言的异常处理产生影响。

--enable-version-specific-runtime-libs

将运行时库安装在编译器特定的子目录中(${libdir}/gcc-lib/${target_alias}/${gcc_version})，而不是默认的${libdir}目录中。另外，'libstdc++'的头文件将被安装在 ${libdir}/gcc-lib/${target_alias}/${gcc_version}/include/g++ 目录中(除非同时又指定了 --with-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，libgfortran,libjava,libmudflap,libstdc++,libobjc 都支持该选项。

--enable-werror
--disable-werror

是否将所有编译器警告当作错误看待(使用-Werror 来编译)。对于开发中的版本和快照默认为"yes"，对于正式发布的版本则默认为"no"。

--with-as=pathname
--with-ld=pathname

指定将来 GCC 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。

--with-datarootdir=DATADIR

将 DATADIR 用作数据根目录，默认值是[PREFIX/share]

--with-docdir=DOCDIR
--with-htmldir=HTMLDIR
--with-pdfdir=PDFDIR

指定各种文档的安装目录。DOCDIR 默认值的默认值是 DATADIR，HTMLDIR 和 PDFDIR 的默认值是 DOCDIR。

--with-gmp=GMPDIR
--with-gmp-include=GMPINCDIR
--with-gmp-lib=GMPLIBDIR

指定 GMP 库的安装目录/头文件目录/库目录。指定 GMPDIR 相当于同时指定了：GMPINCDIR=GMPDIR/include,GMPLIBDIR=GMPDIR/lib 。

--with-mpfr=MPFRDIR
--with-mpfr-include=MPFRINCDIR
--with-mpfr-lib=MPFRLIBDIR

指定 MPFR 库的安装目录/头文件目录/库目录。指定 MPFRDIR 相当于同时指定了：MPFRINCDIR=MPFRDIR/include,MPFRLIBDIR=MPFRDIR/lib 。

--with-cloog=CLOOGDIR
--with-cloog_include=CLOOGINCDIR
--with-cloog_lib=CLOOGLIBDIR

指定 CLooG(Chunky Loop Generator)的安装目录/头文件目录/库目录。指定 CLOOGDIR 相当于同时指定了：CLOOGINCDIR=CLOOGDIR/include,CLOOGLIBDIR=CLOOGDIR/lib 。[GCC-4.4 新增选项]

--with-ppl=PPLDIR
--with-ppl_include=PPLINCDIR
--with-ppl_lib=PPLLIBDIR

指定 PPL(Parma Polyhedra Library)的安装目录/头文件目录/库目录。指定 PPLDIR 相当于同时指定了：PPLINCDIR=PPLDIR/include,PPLLIBDIR=PPLDIR/lib 。[GCC-4.4 新增选项]

--with-gxx-include-dir=DIR

G++头文件的安装目录，默认为"prefix/include/c++/版本"。

--with-libiconv-prefix[=DIR]
--without-libiconv-prefix

在 DIR/include 目录中搜索 libiconv 头文件，在 DIR/lib 目录中搜索 libiconv 库文件。或者根本不使用 libiconv 库。

--with-libintl-prefix[=DIR]
--without-libintl-prefix

在 DIR/include 目录中搜索 libintl 头文件，在 DIR/lib 目录中搜索 libintl 库文件。或者根本不使用 libintl 库。

--with-local-prefix=DIR

指定本地包含文件的安装目录，不管如何设置--prefix，其默认值都为 /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 ，也不能指定为安装 GCC 自身头文件的目录(默认为$libdir/gcc/$target/$version/include)，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。

--with-long-double-128
--without-long-double-128

指定 long double 类型为 128-bit 或 64-bit(等于 double) 。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 ； 但是可以使用这个选项强制指定。

--with-pic
--without-pic

试图仅使用 PIC 或 non-PIC 对象，默认两者都使用。

--with-slibdir=DIR

共享库(libgcc)的安装目录，默认等于 --libdir 的值。

--with-system-libunwind

使用系统中已经安装的 libunwind 库，默认自动检测。

--with-system-zlib

使用系统中的 libz 库，默认使用 GCC 自带的库。

以下选项仅适用于 C++ 语言：

--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 C++标准对析构函数的处理规定。启用它相当于在将来调用 gcc 时默认使用 -fuse-cxa-exit 选项。该选项仅在使用 Glibc 的时候才有意义。

--disable-c99

禁止支持 C99 标准。该选项将导致 ABI 接口发生改变。

--enable-cheaders=c|c_std|c_global

为 g++ 创建 C 语言兼容的头文件，默认为"c_global"。

--enable-clocale[=gnu|ieee_1003.1-2001|generic]

指定目标系统的 locale 模块，默认值为自动检测。建议明确设为"gnu"，否则可能会编译出 ABI 不兼容的 C++ 库。

--enable-clock-gettime[=yes|no|rt]

指明如何获取 C++0x 草案里面 time.clock 中 clock_gettime()函数："yes"表示在 libc 和 libposix4 库中检查(而 libposix4 在需要的时候还可能会链接到 libstdc++)。"rt"表示还额外在 librt 库中查找，这一般并不是一个很好的选择，因为 librt 经常还会连接到 libpthread 上，从而使得单线程的程序产生不必要的锁定开销。默认值"no"则完全跳过这个检查。[GCC-4.4 新增选项]

--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-cstdio=PACKAGE

使用目标平台特定的 I/O 包，PACKAGE 的默认值是"stdio"，也是唯一可用的值。使用这个选项将导致 ABI 接口发生改变。

--enable-cxx-flags=FLAGS

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-fully-dynamic-string

该选项启用了一个特殊版本的 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-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(前者的子集)。

--enable-libstdcxx-allocator[=new|malloc|mt|bitmap|pool]

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-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=FLAGS

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"

--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。该选项的默认值等于 hosted-libstdcxx 的值。

--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。

--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。建议不要使用此选项。

--disable-visibility

禁止 -fvisibility 编译器选项的使用(使其失效)。

--disable-wchar_t

禁止使用模板支持多字节字符类型'wchar_t'。该选项将导致 ABI 接口发生改变。

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。其默认值是 ${gcc_tooldir}/sys-root 。

--with-target-subdir=SUBDIR

为 target 在 SUBDIR 子目录中进行配置。

--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。

--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。默认值是 SUBDIR。

--with-build-sysroot=sysroot

在编译时将'sysroot'当作指定 build 平台的根目录看待。仅在已经使用了--with-sysroot 选项的时候，该选项才有意义。

--with-build-time-tools=path

在给定的 path 中寻找用于编译 GCC 自身的目标工具。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。例如，当编译 GCC 的系统的文件布局和将来部署 GCC 的目标系统不一致时就需要使用此选项。

--with-cross-host=HOST

这个选项已经被反对使用，应该使用--with-sysroot 来代替其功能。

以下选项意义不大，一般不用考虑它们：

--enable-cld

启用它相当于将来对 32 位 x86 平台调用 GCC 时默认使用-mcld 命令行选项，主要用于兼容一些老旧的平台。

--disable-cloog-version-check

禁止检测 CLooG(Chunky Loop Generator)的版本是否满足要求。[GCC-4.4 新增选项]

--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，在需要进行 coverage 分析的时候使用"noopt"(默认)，在需要进行性能分析的时候使用"opt"。

--disable-dependency-tracking

禁止对 Makefile 规则的依赖性追踪。

--disable-fast-install

禁止为快速安装而进行优化。

--enable-fixed-point

启用 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，特别是在具有相应硬件支持的处理器(比如 MIPS)上。在 MIPS 平台上默认开启，在其他平台上则默认关闭。

--enable-gold

仅在与 Binutils 联合编译时才有意义。使用 gold 代替 GNU ld。gold 是 Google 开发的连接器，2008 年捐赠给 FSF，目的是取代现有的 GNU ld，但目前两者还不能完兼容。[GCC-4.4 新增选项]

--enable-gather-detailed-mem-stats

允许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(比如创建正式发布的 tarball)。

--enable-initfini-array

为构造函数和析构函数使用 .init_array 和 .fini_array (而不是 .init 和 .fini) 节。该选项的默认值由 configure 脚本自动检测决定。

--enable-intermodule

仅用一步来编译 compiler ，以达到内部模块最佳化。(什么意思?)

--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，没有实际效果(相当于 disable)。

--disable-largefile

禁止支持大文件。[GCC-4.4 新增选项]

--disable-libada

禁止编译 GNAT 的运行时库(libada，ADA 编译器的运行时库)和相应的工具。

--disable-libgcj

禁止编译 GCJ 的运行时库(libgcj，Java 编译器的运行时库)。

--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-linux-futex

在 libgomp 和 libstdc++库中使用 Linux 的 futex 系统调用(快速用户空间互斥体)。默认值根据内核头文件 sys/syscall.h 中是否包含 futex 函数的定义而定。

--enable-maintainer-mode

启用无用的 make 规则和依赖性(它们有时会导致混淆)。

--enable-objc-gc

允许在 Objective-C 运行时库中使用 Boehm 垃圾回收器。当前并不支持 Boehm，所以该选项没有实际意义。

--disable-ppl-version-check

禁止检测 PPL(Parma Polyhedra Library)的版本是否满足要求。[GCC-4.4 新增选项]

--enable-secureplt

使编译器默认创建只读的 plt 项，相当于将来调用 gcc 时默认使用 -msecure-plt 选项。仅对 powerpc-linux 平台有意义。

--enable-stage1-checking

对处于 stage1 状态的编译器执行额外的检查。默认值等于--enable-checking 选项的值。

--enable-stage1-languages

在 bootstrap 时，在 stage1 时使用系统原有的 C 编译器编译更多的语言支持，而不是使用 stage1 编译出来的 C 编译器来编译他们。该选项一般仅供编译器开发者使用。无默认值，可用的取值为：no, yes, all, ada, c, c++, fortran, java, objc, obj-c++ 。

--enable-tls
--disable-tls

允许或禁止目标系统支持 TLS(线程本地存储)，"gcc"子目录下没有默认值，"libgcc libgomp libmudflap libstdc++-v3"子目录下默认为"yes"。一般情况下不需要明确指定，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-twoprocess

Choose two-process fix methodology(啥意思?)。对于那些不支持双向 pipe 的系统，必须使用 two-process 方法。在 mingw32*,beos* 平台上默认为"yes"，其它平台上默认为"no"。

--enable-werror-always

不管编译器是否支持，总是使用-Werror 来编译，也就是将所有编译器警告当作错误看待。

--enable-win32-registry=KEY

仅对 Windows 平台有意义。而且即使在 Windows 平台上也可以忽略该选项。

--with-gnu-as
--with-gnu-ld

指定编译器将来使用的是 GNU 汇编器/连接器，默认值为未指定。如果你实际使用的不是 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，也有可能会造成混淆。此选项仅在 hppa, sparc, sparc64 平台上才有意义。

--with-bugurl=URL

提示用户发送 bug 报告的 URL。默认值是"http://gcc.gnu.org/bugs.html"。

--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-debug-prefix-map='A=B C=D ...'

在调试信息中将 A 映射 B,C 映射到 D...

--with-demangler-in-ld

尝试在 GNU ld 中使用 demangler

--with-dwarf2

指定编译程序产生的调试信息默认为 DWARF2 格式。

--with-gc=[page|zone]

指定编译过程中使用的垃圾回收方案(默认为"page")。

--with-included-gettext

使用软件包中自带的 GNU gettext 库。如果你已经使用了 Glibc-2.0 以上的版本，或者系统中已经安装了 GNU gettext 软件包，那么就没有必要使用这个选项。默认不使用。

--with-pkgversion=PKG

使用"PKG"代替默认的"GCC"作为版本字符串，这个信息将会在"gcc --version"命令下显示。

比如你可以在其中嵌入编译时间或第多少次编译之类的信息。

--with-stabs

指定将来编译器产生的调试信息默认为 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。

编译、测试、安装

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和相应库文件的执行效率。

编译完毕后可以使用"make check"运行测试套件，然后可以和 http://gcc.gnu.org/buildstat.html 里面列出来的结果进行对比，只要"unexpected failures"不要太多就好说。这个测试套件依赖于 DejaGnu 软件包，而 DejaGnu 又依赖于 expect，expect 依赖于 tcl。如果只想运行 C++测试，可以使用"make check-g++"命令；如果只想运行 C 编译器测试，可以使用"make check-gcc"。还可以制定只运行某些单项测试：比如使用 make check RUNTESTFLAGS="compile.exp -v" 运行编译测试。另一方面，GCC 并不支持使用"make uninstall"进行卸载，建议你将 GCC 安装在一个特别的目录中，然后在不需要的时候直接删除这个目录。

因为 GCC 的安装依赖于 GMP 和 MPFR，所以下面附上 GMP 和 MPFR 的安装信息，主要是 configure 选项。

GMP-4.2.4

下面所列选项若未特别说明皆为*非默认值*。并且仅选择有实际意义的选项介绍：

--enable-assert

启用断言检查，主要用于调试目的。

--enable-alloca=alloca|malloc-reentrant|malloc-notreentrant

分配临时工作区内存的方法：
alloca - 使用 libc 或编译器内置的方法
malloc-reentrant - 在堆上使用可重入的(re-entrant)方法分配
malloc-notreentrant - 在堆上使用全局变量的方法分配
默认值是优先使用 alloca，不可用时使用 malloc-reentrant。

--enable-cxx

启用 C++支持(必须同时拥有 C++编译器支持)。也就是将要安装 libgmpxx.la 库和 gmpxx.h 头文件。

--enable-fat

在运行时根据 CPU 型号选择相应的底层子程序。这将使得代码变得臃肿但可以获得更好的

性能。

--disable-fft

默认情况下，GMP 使用 Karatsuba, 3-way Toom, Fermat FFT 三种算法进行乘法运算。而 Fermat FFT 算法仅用于操作数非常巨大的场合，所以，如果你预计到并不需要操作非常巨大的数字，那么可以禁用这个算法，这样可以减小一些二进制文件的体积。

--enable-mpbsd

编译与 Berkeley MP 接口兼容的库文件(libmp.{so,a})和头文件(mp.h)

--enable-nails=偶数

在 limbs 中使用 nail(?何意?)

--enable-profile

启用 profiling 信息相关的库文件编译。主要用于调试目的。

--disable-shared
--disable-static

禁止编译共享或静态版本的库。

--with-readline

在 calc 演示程序中使用 readline 库，默认值自动检测。

--with-pic
--without-pic

试图仅使用 PIC 或 non-PIC 对象，默认两者都使用。

MPFR-2.3.2

此包依赖于 GMP，并且总是在 http://www.mpfr.org/mpfr-current/patches 存放最新版本的 patch，可以使用

```
patch -N -Z -p1 < patches
```

命令打补丁。

下面所列选项若未特别说明皆为*非默认值*。并且仅选择有实际意义的选项介绍：

--enable-assert

启用断言检查，主要用于调试目的。

--enable-decimal-float

编译与十进制浮点数之间的转换函数(要求 GCC>=4.2)

--enable-logging

启用 MPFR 日志(必须要有底层操作系统的支持)

--disable-shared
--disable-static

禁止编译共享或静态版本的库。

--enable-tests-timeout=NUM

设定测试程序的超时秒数(NUM<=9999)，默认没有超时限制。

--enable-thread-safe

编译线程安全的 MPFR 库

--enable-warnings

允许 MPFR 将错误输出到 stderr

--with-gmp=GMPDIR
--with-gmp-include=GMPINCDIR
--with-gmp-lib=GMPLIBDIR

指定 GMP 库的安装目录/头文件目录/库目录。指定 GMPDIR 相当于同时指定了：GMPINCDIR=GMPDIR/include,GMPLIBDIR=GMPDIR/lib 。

--with-mulhigh_size=NUM

mulhigh 的内置阈值表大小，没有默认值。

--with-pic
--without-pic

试图仅使用 PIC 或 non-PIC 对象，默认两者都使用。
