insistYuan

写给自己看的，详聊Q我或者写封信吧！

gcc选项

摘自http://blog.csdn.net/liuchao1986105/article/details/6674822

版本] –0.13
[声明]
这篇文档是我的关于gcc参数的笔记,我很怀念dos年代我用小本子,纪录任何的dos 命令的参数.哈哈,下面的东西可能也不是很全面,我参考了很多的书,和gcc的帮助.不全的原因是,有可能我还没有看到这个参数,另一种原因是,我可能还不会用他 但是,我会慢慢的补齐的.哈哈 假如您要转在本文章请保留我email(pianopan@beeship.com)和文章的全面性.
[介绍]
gcc and g++分别是gnu的c & c++编译器 gcc/g++在执行编译工作的时候，总共需要4步
1.预处理,生成.i的文档[预处理器cpp]
2.将预处理后的文档不转换成汇编语言,生成文档.s[编译器egcs]
3.有汇编变为目标代码(机器代码)生成.o的文档[汇编器as]
4.连接目标代码,生成可执行程式[链接器ld]

[参数详解]
–x language filename
　　设定文档所使用的语言,使后缀名无效,对以后的多个有效.也就是根据约定C语言的后缀名称是.c的，而C++的后缀名是.C或.cpp,假如您很个性，决定您的C代码文档的后缀名是.pig 哈哈，那您就要用这个参数,这个参数对他后面的文档名都起作用，除非到了下一个参数的使用。
　　能够使用的参数吗有下面的这些
　　`c', `objective-c', `c-header', `c++', `cpp-output', `assembler', and `assembler-with-cpp'.
　　看到英文，应该能够理解的。
　　例子用法:
　　gcc –x c hello.pig

–x none filename
　　关掉上一个选项，也就是让gcc根据文档名后缀，自动识别文档类型
　　例子用法:
　　gcc –x c hello.pig –x none hello2.c

–c
　　只激活预处理,编译,和汇编,也就是他只把程式做成obj文档
　　例子用法:
　　gcc –c hello.c
　　他将生成.o的obj文档

–S
　　只激活预处理和编译，就是指把文档编译成为汇编代码。
　　例子用法
　　gcc –S hello.c
　　他将生成.s的汇编代码，您能够用文本编辑器察看

–E
　　只激活预处理,这个不生成文档,您需要把他重定向到一个输出文档里面.

例子用法:

gcc –E hello.c > pianoapan.txt

gcc –E hello.c | more

慢慢看吧,一个hello word 也要和处理成800行的代码

–o

定制目标名称,缺省的时候,gcc 编译出来的文档是a.out,很难听,假如您和我有同感，改掉他,哈哈

例子用法

gcc –o hello.exe hello.c (哦,windows用习惯了)

gcc –o hello.asm –S hello.c

–pipe

使用管道代替编译中临时文档,在使用非gnu汇编工具的时候,可能有些问题

gcc –pipe –o hello.exe hello.c

–ansi

关闭gnu c中和ansi c不兼容的特性,激活ansi c的专有特性(包括禁止一些asm inline typeof关键字,连同UNIX,vax等预处理宏,

–fno–asm

此选项实现ansi选项的功能的一部分，他禁止将asm,inline和typeof用作关键字。

–fno–strict–prototype

只对g++起作用,使用这个选项,g++将对不带参数的函数,都认为是没有显式的对参数的个数和类型说明,而不是没有参数.

而gcc无论是否使用这个参数,都将对没有带参数的函数,认为城没有显式说明的类型

–fthis–is–varialble

就是向传统c++看齐,能够使用this当一般变量使用.

–fcond–mismatch

允许条件表达式的第二和第三参数类型不匹配,表达式的值将为void类型

–funsigned–char

–fno–signed–char

–fsigned–char

–fno–unsigned–char

这四个参数是对char类型进行配置,决定将char类型配置成unsigned char(前两个参数)或 signed char(后两个参数)

–include file

包含某个代码,简单来说,就是便以某个文档,需要另一个文档的时候,就能够用他设定,功能就相当于在代码中使用#include<filename>

例子用法:

gcc hello.c –include /root/pianopan.h

–imacros file

将file文档的宏,扩展到gcc/g++的输入文档,宏定义本身并不出现在输入文档中

–Dmacro

相当于C语言中的#define macro

–Dmacro=defn

相当于C语言中的#define macro=defn

–Umacro

相当于C语言中的#undef macro

随笔档案

待看

makefile学习（多个链接）
B Shell编程系列----脚本命令解析查询

阅读排行榜

–undef

取消对任何非标准宏的定义

–Idir

在您是用#include”file”的时候,gcc/g++会先在当前目录查找您所定制的头文档,假如没有找到,他回到缺省的头文档目录找,假如使用–I定制了目录,他

回先在您所定制的目录查找,然后再按常规的顺序去找.

对于#include<file>,gcc/g++会到–I定制的目录查找,查很难找到,然后将到系统的缺省的头文档目录查找

–I–

就是取消前一个参数的功能,所以一般在–Idir之后使用

–idirafter dir

在–I的目录里面查找失败,讲到这个目录里面查找.

–iprefix prefix

–iwithprefix dir

一般一起使用,当–I的目录查找失败,会到prefix+dir下查找

–nostdinc

使编译器不再系统缺省的头文档目录里面找头文档,一般和–I联合使用,明确限定头文档的位置

–nostdin C++

规定不在g++指定的标准路经中搜索,但仍在其他路径中搜索,.此选项在创libg++库使用

–C

在预处理的时候,不删除注释信息,一般和–E使用,有时候分析程式，用这个很方便的

–M

生成文档关联的信息。包含目标文档所依赖的任何源代码您能够用gcc –M hello.c来测试一下,很简单。

–MM

和上面的那个相同，但是他将忽略由#include<file>造成的依赖关系。

–MD

和–M相同，但是输出将导入到.d的文档里面

–MMD

和–MM相同，但是输出将导入到.d的文档里面

–Wa,option

此选项传递option给汇编程式;假如option中间有逗号,就将option分成多个选项,然后传递给会汇编程式

–Wl.option

此选项传递option给连接程式;假如option中间有逗号,就将option分成多个选项,然后传递给会连接程式.

–llibrary

定制编译的时候使用的库

例子用法

gcc –lcurses hello.c

评论排行榜

推荐排行榜

最新评论

使用ncurses库编译程式

–Ldir

定制编译的时候，搜索库的路径。比如您自己的库，能够用他定制目录，不然编译器将只在标准库的目录找。这个dir就是目录的名称。

–O0

–O1

–O2

–O3

编译器的优化选项的4个级别，–O0表示没有优化,–O1为缺省值，–O3优化级别最高

–g

只是编译器，在编译的时候，产生调试信息。

–gstabs

此选项以stabs格式声称调试信息,但是不包括gdb调试信息.

–gstabs+

此选项以stabs格式声称调试信息,并且包含仅供gdb使用的额外调试信息.

–ggdb

此选项将尽可能的生成gdb的能够使用的调试信息.

–static

此选项将禁止使用动态库，所以，编译出来的东西，一般都很大，也无需什么动态连接库，就能够运行.

–share

此选项将尽量使用动态库，所以生成文档比较小，但是需要系统由动态库.

–traditional

试图让编译器支持传统的C语言特性

gcc提供了大量的警告选项，对代码中可能存在的问题提出警 告，通常可以使用–Wall来开启以下警告:

–Waddress –Warray–bounds (only with –O2) –Wc++0x–compat
–Wchar–subscripts –Wimplicit–int –Wimplicit–function–declaration
–Wcomment –Wformat –Wmain (only for C/ObjC and unless
–ffreestanding) –Wmissing–braces –Wnonnull –Wparentheses
–Wpointer–sign –Wreorder –Wreturn–type –Wsequence–point
–Wsign–compare (only in C++) –Wstrict–aliasing –Wstrict–overflow=1
–Wswitch –Wtrigraphs –Wuninitialized (only with –O1 and above)
–Wunknown–pragmas –Wunused–function –Wunused–label –Wunused–value
–Wunused–variable

unused–function:警告声明但是没有定义的static函数;
unused– label:声明但是未使用的标签;
unused–parameter:警告未使用的函数参数;
unused–variable:声明但 是未使用的本地变量;
unused–value:计算了但是未使用的值;
format:printf和scanf这样的函数中的格式字符 串的使用不当;
implicit–int:未指定类型;
implicit–function:函数在声明前使用;
char– subscripts:使用char类作为数组下标(因为char可能是有符号数);

––zxc单飞

2. Re:RSA算法原理（简单易懂）

密文＝明文EmodN密文＝明文EmodN
这句话是格式乱了吗，不太明白 ”明文EmodN密文“ 这个 运算。

––fakar

3. Re:RSA算法原理（简单易懂）

学到了

––aJream

4. Re:RSA算法原理（简单易懂）

@lianggexl 209都不是质数，有约数11...

––RobertLee

5. Re:RSA算法原理（简单易懂）

这个有问题啊，如果p用17，q用19确实可以解密出明文来，但是p或q如果值选大一些，就无法解密出明文了，例如q用200以上的数如209，不知道楼主自己有没试过

––lianggexl

missingbraces:大括号不匹配;
parentheses: 圆括号不匹配;
return-type:函数有无返回值以及返回值类型不匹配;
sequence-point:违反顺序点的代码,比如 a[i] = c[i++];
switch:switch语句缺少default或者switch使用枚举变量为索引时缺少某个变量的case;
strict- aliasing=n:使用n设置对指针变量指向的对象类型产生警告的限制程度,默认n=3;只有在-fstrict-aliasing设置的情况下有 效;
unknow-pragmas:使用未知的#pragma指令;
uninitialized:使用的变量为初始化,只在-O2时有 效;

以下是在-Wall中不会激活的警告选项:
cast-align:当指针进行类型转换后有内存对齐要求更严格时发出警告;
sign- compare:当使用signed和unsigned类型比较时;
missing-prototypes:当函数在使用前没有函数原型时;
packed:packed 是gcc的一个扩展,是使结构体各成员之间不留内存对齐所需的空 间,有时候会造成内存对齐的问题;
padded:也是gcc的扩展,使结构体成员之间进行内存对齐的填充,会 造成结构体体积增大.
unreachable-code:有不会执行的代码时.
inline:当inline函数不再保持inline时 (比如对inline函数取地址);
disable-optimization:当不能执行指定的优化时.(需要太多时间或系统资源).
可以使用 -Werror时所有的警告都变成错误,使出现警告时也停止编译.需要和指定警告的参数一起使用.

优化:
gcc默认提供了5级优 化选项的集合:
-O0:无优化(默认)
-O和-O1:使用能减少目标文 件大小以及执行时间并且不会使编译时间明显增加的优化.在编译大型程序的时候会显著增加编译时内存的使用.
-O2: 包含-O1的优化并增加了不需要在目标文件大小和执行速度上进行折衷的优化.编译器不执行循环展开以及函数内联.此选项将增加编译时间和目标文件的执行性 能.
-Os:专门优化目标文件大小,执行所有的不增加目标文件大小的-O2优化选项.并且执行专门减小目标文件大小的优化选项.
-O3: 打开所有-O2的优化选项并且增加 -finline-functions, -funswitch-loops,-fpredictive-commoning, -fgcse-after-reload and -ftree-vectorize优化选项.

-O1包含的选项-O1通常可以安全的和调试的选项一起使用:

```
-fauto-inc-dec -fcprop-registers -fdce -fdefer-pop -fdelayed-branch
-fdse -fguess-branch-probability -fif-conversion2 -fif-conversion
-finline-small-functions -fipa-pure-const -fipa-reference
-fmerge-constants -fsplit-wide-types -ftree-ccp -ftree-ch
-ftree-copyrename -ftree-dce -ftree-dominator-opts -ftree-dse
-ftree-fre -ftree-sra -ftree-ter -funit-at-a-time
```

以下所有的优化选项需要在名字 前加上-f,如果不需要此选项可以使用-fno-前缀
defer-pop:延迟到只在必要时从函数参数栈中pop参数;
thread- jumps:使用跳转线程优化,避免跳转到另一个跳转;
branch-probabilities:分支优化;
cprop- registers:使用寄存器之间copy-propagation传值;
guess-branch-probability:分支预测;
omit- frame-pointer:可能的情况下不产生栈帧;

-O2:以下是-O2在-O1基础上增加的优化选项:

```
-falign-functions  -falign-jumps -falign-loops  -falign-labels
-fcaller-saves -fcrossjumping -fcse-follow-jumps  -fcse-skip-blocks
```

-fdelete-null-pointer-checks -fexpensive-optimizations -fgcse
-fgcse-lm -foptimize-sibling-calls -fpeephole2 -fregmove
-freorder-blocks -freorder-functions -frerun-cse-after-loop
-fsched-interblock -fsched-spec -fschedule-insns
-fschedule-insns2 -fstrict-aliasing -fstrict-overflow -ftree-pre
-ftree-vrp

cpu架构的优化选项,通常是-mcpu(将被取消);-march,-mtune

Debug选项:
在 gcc编译源代码时指定-g选项可以产生带有调试信息的目标代码,gcc可以为多个不同平台上帝不同调试器提供调试信息,默认gcc产生的调试信息是为 gdb使用的,可以使用-gformat指定要生成的调试信息的格式以提供给其他平台的其他调试器使用.常用的格式有
-ggdb:生成gdb专 用的调试信息,使用最适合的格式(DWARF 2,stabs等)会有一些gdb专用的扩展,可能造成其他调试器无法运行.
-gstabs:使用 stabs格式,不包含gdb扩展,stabs常用于BSD系统的DBX调试器.
-gcoff:产生COFF格式的调试信息,常用于System V下的SDB调试器;
-gxcoff:产生XCOFF格式的调试信息,用于IBM的RS/6000下的DBX调试器;
-gdwarf- 2:产生DWARF version2 的格式的调试信息,常用于IRIXX6上的DBX调试器.GCC会使用DWARF version3的一些特性.
可 以指定调试信息的等级:在指定的调试格式后面加上等级:
如: -ggdb2 等,0代表不产生调试信息.在使用-gdwarf-2时因为最早的格式为-gdwarf2会造成混乱,所以要额外使用一个-glevel来指定调试信息的 等级,其他格式选项也可以另外指定等级.

gcc可以使用-p选项指定生成信息以供porf使用.

GCC常用选项

选项	含义
--help --target-help	显示 gcc 帮助说明。‘target-help’是显示目标机器特定的命令行选项。
--version	显示 gcc 版本号和版权信息 。
-o*outfile*	输出到指定的文件。
-x*language*	指明使用的编程语言。允许的语言包括：c c++ assembler none 。‘none’意味着恢复默认行为，即根据文件的扩展名猜测源文件的语言。
-v	打印较多信息，显示编译器调用的程序。
-###	与 -v 类似，但选项被引号括住，并且不执行命令。
-E	仅作预处理，不进行编译、汇编和链接。如上图所示。
-S	仅编译到汇编语言，不进行汇编和链接。如上图所示。
-c	编译、汇编到目标代码，不进行链接。如上图所示。
-pipe	使用管道代替临时文件。
-combine	将多个源文件一次性传递给汇编器。

3 其他GCC选项

更多有用的GCC选项：

命令	描述
–l *library* –l*library*	进行链接时搜索名为library的库。 例子： $ gcc test.c –lm –o test
–I*dir*	把*dir*加入到搜索头文件的路径列表中。 例子： $ gcc test.c –I../inc –o test
–L*dir*	把*dir*加入到搜索库文件的路径列表中。 例子： $ gcc –I/home/foo –L/home/foo –ltest test.c –o test
–D*name*	预定义一个名为*name*的宏，值为1。 例子： $ gcc –DTEST_CONFIG test.c –o test
–D*name=definition*	预定义名为*name*，值为*definition*的宏。
–ggdb –ggdb*level*	为调试器 gdb 生成调试信息。*level*可以为1，2，3，默认值为2。
–g –g*level*	生成操作系统本地格式的调试信息。–g 和 –ggdb 并不太相同， –g 会生成 gdb 之外的信息。*level*取值同上。
–s	去除可执行文件中的符号表和重定位信息。用于减小可执行文件的大小。
–M	告诉预处理器输出一个适合make的规则，用于描述各目标文件的依赖关系。对于每个 源文件，预处理器输出 一个make规则，该规则的目标项(target)是源文件对应的目标文件名，依赖项(dependency)是源文件中 `#include引用的所有文件。生成的规则可 以是单行，但如果太长，就用`\’–换行符续成多行。规则 显示在标准输出，不产生预处理过的C程序。
–C	告诉预处理器不要丢弃注释。配合`–E’选项使用。
–P	告诉预处理器不要产生`#line’命令。配合`–E’选项使用。
–static	在支持动态链接的系统上，阻止连接共享库。该选项在其它系统上 无效。
–nostdlib	不连接系统标准启动文件和标准库文件，只把指定的文件传递给连接器。
Warnings	
–Wall	会打开一些很有用的警告选项，建议编译时加此选项。
–W –Wextra	打印一些额外的警告信息。
–w	禁止显示所有警告信息。
–Wshadow	当一个局部变量遮盖住了另一个局部变量，或者全局变量时，给出警告。很有用的选项，建议打开。 –Wall 并不会打开此项。
–Wpointer–arith	对函数指针或者void *类型的指针进行算术操作时给出警告。也很有用。 –Wall 并不会打开此项。
–Wcast–qual	当强制转化丢掉了类型修饰符时给出警告。 –Wall 并不会打开此项。

-Waggregate-return	如果定义或调用了返回结构体或联合体的函数，编译器就发出警告。
-Winline	无论是声明为 inline 或者是指定了-finline-functions 选项，如果某函数不能内联，编译器都将发出警告。如果你的代码含有很多 inline 函数的话，这是很有用的选项。
-Werror	把警告当作错误。出现任何警告就放弃编译。
-Wunreachable-code	如果编译器探测到永远不会执行到的代码，就给出警告。也是比较有用的选项。
-Wcast-align	一旦某个指针类型强制转换导致目标所需的地址对齐增加时，编译器就发出警告。
-Wundef	当一个没有定义的符号出现在 #if 中时，给出警告。
-Wredundant-decls	如果在同一个可见域内某定义多次声明，编译器就发出警告，即使这些重复声明有效并且毫无差别。
Optimization	
-O0	禁止编译器进行优化。默认为此项。
-O -O1	尝试优化编译时间和可执行文件大小。
-O2	更多的优化，会尝试几乎全部的优化功能，但不会进行“空间换时间”的优化方法。
-O3	在 -O2 的基础上再打开一些优化选项：-finline-functions， -funswitch-loops 和 -fgcse-after-reload 。
-Os	对生成文件大小进行优化。它会打开 -O2 开的全部选项，除了会那些增加文件大小的。
-finline-functions	把所有简单的函数内联进调用者。编译器会探索式地决定哪些函数足够简单，值得做这种内联。
-fstrict-aliasing	施加最强的别名规则（aliasing rules）。
Standard	
-ansi	支持符合ANSI标准的C程序。这样就会关闭GNU C中某些不兼容ANSI C的特性。
-std=c89 -iso9899:1990	指明使用标准 ISO C90 作为标准来编译程序。
-std=c99 -std=iso9899:1999	指明使用标准 ISO C99 作为标准来编译程序。

-std=c++98	指明使用标准 C++98 作为标准来编译程序。
-std=gnu9x -std=gnu99	使用 ISO C99 再加上 GNU 的一些扩展。
-fno-asm	不把asm, inline或typeof当作关键字，因此这些词可以用做标识符。用__asm__， __inline__和__typeof__能够替代它们。 `-ansi' 隐含声明了`-fno-asm'。
-fgnu89-inline	告诉编译器在 C99 模式下看到 inline 函数时使用传统的 GNU 句法。
C options	
-fsigned-char -funsigned-char	把char定义为有/无符号类型，如同signed char/unsigned char。
-traditional	尝试支持传统C编译器的某些方面。详见GNU C手册。
-fno-builtin -fno-builtin-function	不接受没有 __builtin_ 前缀的函数作为内建函数。
-trigraphs	支持ANSI C的三联符（ trigraphs）。`-ansi'选项隐含声明了此选项。
-fsigned-bitfields -funsigned-bitfields	如果没有明确声明`signed'或`unsigned'修饰符，这些选项用来定义有符号位域或无符号位域。缺省情况下，位域是有符号的，因为它们继承的基本整数类型，如int，是有符号数。
-Wstrict-prototypes	如果函数的声明或定义没有指出参数类型，编译器就发出警告。很有用的警告。
-Wmissing-prototypes	如果没有预先声明就定义了全局函数，编译器就发出警告。即使函数定义自身提供了函数原形也会产生这个警告。这个选项 的目的是检查没有在头文件中声明的全局函数。
-Wnested-externs	如果某extern声明出现在函数内部，编译器就发出警告。
C++ options	
-ffor-scope	从头开始执行程序，也允许进行重定向。
-fno-rtti	关闭对 dynamic_cast 和 typeid 的支持。如果你不需要这些功能，关闭它会节省一些空间。
-Wctor-dtor-privacy	当一个类没有用时给出警告。因为构造函数和析构函数会被当作私有的。
-Wnon-virtual-dtor	当一个类有多态性，而又没有虚析构函数时，发出警告。-Wall会开启这个选项。
-Wreorder	如果代码中的成员变量的初始化顺序和它们实际执行时初始化顺序不一致，给出警告。
-Wno-	使用过时的特性时不要给出警告。

deprecated	
-Woverloaded-virtual	如果函数的声明隐藏住了基类的虚函数，就给出警告。
Machine Dependent Options (Intel)	
-mtune=*cpu-type*	为指定类型的 CPU 生成代码。*cpu-type*可以是：i386，i486，i586，pentium，i686，pentium4 等等。
-msse -msse2 -mmmx -mno-sse -mno-sse2 -mno-mmx	使用或者不使用MMX，SSE，SSE2指令。
-m32 -m64	生成32位/64位机器上的代码。
-mpush-args -mno-push-args	（不）使用 push 指令来进行存储参数。默认是使用。
-mregparm=*num*	当传递整数参数时，控制所使用寄存器的个数。

让我们先看看 Makefile 规则中的编译命令通常是怎么写的。

大多数软件包遵守如下约定俗成的规范：

```
#1,首先从源代码生成目标文件(预处理,编译,汇编)，"-c"选项表示不执行链接步骤。
$(CC) $(CPPFLAGS) $(CFLAGS) example.c   -c   -o example.o
#2,然后将目标文件连接为最终的结果(连接)，"-o"选项用于指定输出文件的名字。
$(CC) $(LDFLAGS) example.o   -o example

#有一些软件包一次完成四个步骤：
$(CC) $(CPPFLAGS) $(CFLAGS) $(LDFLAGS) example.c   -o example
```

当然也有少数软件包不遵守这些约定俗成的规范，比如：

```
#1,有些在命令行中漏掉应有的Makefile变量(注意：有些遗漏是故意的)
$(CC) $(CFLAGS) example.c    -c   -o example.o
$(CC) $(CPPFLAGS) example.c  -c   -o example.o
$(CC) example.o   -o example
$(CC) example.c   -o example
```

```
#2,有些在命令行中增加了不必要的Makefile变量
$(CC) $(CFLAGS) $(LDFLAGS) example.o   -o example
$(CC) $(CPPFLAGS) $(CFLAGS) $(LDFLAGS) example.c   -c   -o example.o
```

当然还有极个别软件包完全是"胡来"：乱用变量(增加不必要的又漏掉了应有的)者有之，不用$(CC)者有之，不一而足.....

尽管将源代码编译为二进制文件的四个步骤由不同的程序(cpp,gcc/g++,as,ld)完成，但是事实上 cpp, as, ld 都是由 gcc/g++ 进行间接调用的。换句话说，控制了 gcc/g++ 就等于控制了所有四个步骤。从 Makefile 规则中的编译命令可以看出，编译工具的行为全靠 CC/CXX CPPFLAGS CFLAGS/CXXFLAGS LDFLAGS 这几个变量在控制。当然理论上控制编译工具行为的还应当有 AS ASFLAGS ARFLAGS 等变量，但是实践中基本上没有软件包使用它们。

那么我们如何控制这些变量呢？一种简易的做法是首先设置与这些 Makefile 变量同名的环境变量并将它们 export 为全局，然后运行 configure 脚本，大多数 configure 脚本会使用这同名的环境变量代替 Makefile 中的值。但是少数 configure 脚本并不这样做(比如GCC-3.4.6和Binutils-2.16.1的脚本就不传递LDFLAGS)，你必须手动编辑生成的 Makefile 文件，在其中寻找这些变量并修改它们的值，许多源码包在每个子文件夹中都有 Makefile 文件，真是一件很累人的事！

CC 与 CXX

这是 C 与 C++ 编译器命令。默认值一般是 "gcc" 与 "g++"。这个变量本来与优化没有关系，但是有些人因为担心软件包不遵守那些约定俗成的规范，害怕自己苦心设置的 CFLAGS/CXXFLAGS/LDFLAGS 之类的变量被忽略了，而索性将原本应当放置在其它变量中的选项一股老儿塞到 CC 或 CXX 中，比如：CC="gcc -march=k8 -O2 -s"。这是一种怪异的用法，本文不提倡这种做法，而是提倡按照变量本来的含义使用变量。

CPPFLAGS

这是用于预处理阶段的选项。不过能够用于此变量的选项，看不出有哪个与优化相关。如果你实在想设一个，那就使用下面这两个吧：

-DNDEBUG
"NDEBUG"是一个标准的 ANSI 宏，表示不进行调试编译。
-D_FILE_OFFSET_BITS=64
大多数包使用这个来提供大文件(>2G)支持。

CFLAGS 与 CXXFLAGS

CFLAGS 表示用于 C 编译器的选项，CXXFLAGS 表示用于 C++ 编译器的选项。这两个变量实际上涵盖了编译和汇编两个步骤。大多数程序和库在编译时默认的优化级别是"2"(使用"-O2"选项)并且带有调试符号来编 译，也就是 CFLAGS="-O2 -g", CXXFLAGS=$CFLAGS 。事实上，"-O2"已经启用绝大多数安全的优化选项了。另一方面，由于大部分选项可以同时用于这两个变量，所以仅在最后讲述只能用于其中一个变量的选 项。[提醒]下面所列选项皆为非默认选项，你只要按需添加即可。

先说说"-O3"在"-O2"基础上增加的几项：

-finline-functions
允许编译器选择某些简单的函数在其被调用处展开，比较安全的选项，特别是在CPU二级缓 存较大时建议使用。
-funswitch-loops
将循环体中不改变值的变量移动到循环体之外。
-fgcse-after-reload
为了清除多余的溢出，在重载之后执行一个额外的载入消除步骤。

另外：

-fomit-frame-pointer
对于不需要栈指针的函数就不在寄存器中保存指针，因此可以忽略存储和检索地址的代 码，同时对许多函数提供一个额外的寄存器。所有"-O"级别都打开它，但仅在调试器可以不依靠栈指针运行时才有效。在AMD64平台上此选项默认打开，但 是在x86平台上则默认关闭。建议显式的设置它。
-falign-functions=N
-falign-jumps=N

–falign–loops=N

–falign–labels=N

这 四个对齐选项在”–O2”中打开，其中的根据不同的平台N使用不同的默认值。如果你想指定不同于默认值的N，也可以单独指定。比如，对于L2– cache>=1M的cpu而言，指定 –falign–functions=64 可能会获得更好的性能。建议在指定了 –march 的时候不明确指定这里的值。

调试选项：

–fprofile–arcs

在使用这一选项编译程序并运行它以创建包含每个代码块的执行次数的文件后，程序可以再次使用 –fbranch–probabilities 编译，文件中的信息可以用来优化那些经常选取的分支。如果没有这些信息，gcc将猜测哪个分支将被经常运行以进行优化。这类优化信息将会存放在一个以源文 件为名字的并以”.da”为后缀的文件中。

全局选项：

–pipe

在编译过程的不同阶段之间使用管道而非临时文件进行通信，可以加快编译速度。建议使用。

目录选项：

––sysroot=dir

将dir作为逻辑根目录。比如编译器通常会在 /usr/include 和 /usr/lib 中搜索头文件和库，使用这个选项后将在 dir/usr/include 和 dir/usr/lib 目录中搜索。如果使用这个选项的同时又使用了 –isysroot 选项，则此选项仅作用于库文件的搜索路径，而 –isysroot 选项将作用于头文件的搜索路径。这个选项与优化无关，但是在 CLFS 中有着神奇的作用。

代码生成选项：

–fno–bounds–check

关闭所有对数组访问的边界检查。该选项将提高数组索引的性能，但当超出数组边界时，可能会 造成不可接受的行为。

–freg–struct–return

如果struct和union足够小就通过寄存器返回，这将提高较小结构的效率。如果 不够小，无法容纳在一个寄存器中，将使用内存返回。建议仅在完全使用GCC编译的系统上才使用。

–fpic

生成可用于共享库的位置独立代码。所有的内部寻址均通过全局偏移表完成。要确定一个地址，需要将代码自身的内存位置 作为表中一项插入。该选项产生可以在共享库中存放并从中加载的目标模块。

–fstack–check

为防止程序栈溢出而进行必要的检测，仅在多线程环境中运行时才可能需要它。

–fvisibility=hidden

设置默认的ELF镜像中符号的可见性为隐藏。使用这个特性可以非常充分的提高连接和加 载共享库的性能，生成更加优化的代码，提供近乎完美的API输出和防止符号碰撞。我们强烈建议你在编译任何共享库的时候使用该选项。参见 –fvisibility–inlines–hidden 选项。

硬件体系结构相关选项[仅仅针对x86与x86_64]：

–march=cpu–type

为特定的cpu–type编译二进制代码(不能在更低级别的cpu上运行)。Intel可以 用：pentium2, pentium3(=pentium3m), pentium4(=pentium4m), pentium–m, prescott, nocona, core2(GCC–4.3新增）。AMD可以用：k6–2(=k6–3), athlon(=athlon–tbird), athlon–xp(=athlon–mp), k8(=opteron=athlon64=athlon–fx)

–mfpmath=sse

P3和athlon–xp级别及以上的cpu支持”sse”标量浮点指令。仅建议在P4和K8以上级 别的处理器上使用该选项。

–malign–double

将double, long double, long long对齐于双字节边界上；有助于生成更高速的代码，但是程序的尺寸会变大，并且不能与未使用该选项编译的程序一起工作。

–m128bit–long–double

指定long double为128位，pentium以上的cpu更喜欢这种标准，并且符合x86-64的ABI标准，但是却不附合i386的ABI标准。

-mregparm=N

指定用于传递整数参数的寄存器数目(默认不使用寄存器)。0<=N<=3 ； 注意：当N>0时你必须使用同一参数重新构建所有的模块，包括所有的库。

-msseregparm

使用SSE寄存器传递float和double参数和返回值。注意：当你使用了这个选项以后，你必须 使用同一参数重新构建所有的模块，包括所有的库。

-mmmx

-msse

-msse2

-msse3

-m3dnow

-mssse3(没写错!GCC-4.3 新增)

-msse4.1(GCC-4.3新增)

-msse4.2(GCC-4.3新增)

-msse4(含4.1和 4.2,GCC-4.3新增)

是否使用相应的扩展指令集以及内置函数，按照自己的cpu选择吧!

-maccumulate-outgoing-args

指定在函数引导段中计算输出参数所需最大空间，这在大部分现代cpu中 是较快的方法；缺点是会明显增加二进制文件尺寸。

-mthreads

支持Mingw32的线程安全异常处理。对于依赖于线程安全异常处理的程序，必须启用这个选项。使用这个选 项时会定义"-D_MT"，它将包含使用选项"-lmingwthrd"连接的一个特殊的线程辅助库，用于为每个线程清理异常处理数据。

-minline-all-stringops

默认时GCC只将确定目的地会被对齐在至少4字节边界的字符串操作内联进程序代 码。该选项启用更多的内联并且增加二进制文件的体积，但是可以提升依赖于高速 memcpy, strlen, memset 操作的程序的性能。

-minline-stringops-dynamically

GCC-4.3新增。对未知尺寸字符串的小块操作使用内联代 码，而对大块操作仍然调用库函数，这是比"-minline-all-stringops"更聪明的策略。决定策略的算法可以通 过"-mstringop-strategy"控制。

-momit-leaf-frame-pointer

不为叶子函数在寄存器中保存栈指针，这样可以节省寄存器，但是将会使调试 变的困难。注意：不要与 -fomit-frame-pointer 同时使用，因为会造成代码效率低下。

-m64

生成专门运行于64位环境的代码，不能运行于32位环境，仅用于x86_64[含EMT64]环境。

-mcmodel=small

[默认值]程序和它的符号必须位于2GB以下的地址空间。指针仍然是64位。程序可以静态连接也 可以动态连接。仅用于x86_64[含EMT64]环境。

-mcmodel=kernel

内核运行于2GB地址空间之外。在编译linux内核时必须使用该选项！仅用于 x86_64[含EMT64]环境。

-mcmodel=medium

程序必须位于2GB以下的地址空间，但是它的符号可以位于任何地址空间。程序可以静态连接也可 以动态连接。注意：共享库不能使用这个选项编译！仅用于x86_64[含EMT64]环境。

其它优化选项：

-fforce-addr

必须将地址复制到寄存器中才能对他们进行运算。由于所需地址通常在前面已经加载到寄存器中了，所以这 个选项可以改进代码。

-finline-limit=n

对伪指令数超过n的函数，编译程序将不进行内联展开，默认为600。增大此值将增加编译时间 和编译内存用量并且生成的二进制文件体积也会变大，此值不宜太大。

–fmerge–all–constants
试图将跨编译单元的所有常量值和数组合并在一个副本中。但是标准C/C++要求每 个变量都必须有不同的存储位置，所以该选项可能会导致某些不兼容的行为。
–fgcse–sm
在全局公共子表达式消除之后运行存储移动，以试图将存储移出循环。gcc–3.4中曾属于”–O2”级别的 选项。
–fgcse–las
在全局公共子表达式消除之后消除多余的在存储到同一存储区域之后的加载操作。gcc–3.4中曾属于”–O2”级别的选项。
–floop–optimize
已废除(GCC–4.1曾包含在”–O1”中)。
–floop–optimize2
使用改进版本的循环优化器代替原来”–floop–optimize”。该优化器将使用不同 的选项(–funroll–loops, –fpeel–loops, –funswitch–loops, –ftree–loop–im)分别控制循环优化的不同方面。目前这个新版本的优化器尚在开发中，并且生成的代码质量并不比以前的版本高。已废除，仅存在 于GCC–4.1之前的版本中。
–funsafe–loop–optimizations
假定循环不会溢出，并且循环的退出条件不是无穷。这将可以在一个比较 广的范围内进行循环优化，即使优化器自己也不能断定这样做是否正确。
–fsched–spec–load
允许一些装载指令执行一些投机性的动作。
–ftree–loop–linear
在trees上进行线型循环转换。它能够改进缓冲性能并且允许进行更进一步的循环优化。
–fivopts
在trees上执行归纳变量优化。
–ftree–vectorize
在trees上执行循环向量化。
–ftracer
执行尾部复制以扩大超级块的尺寸，它简化了函数控制流，从而允许其它的优化措施做的更好。据说挺有效。
–funroll–loops
仅对循环次数能够在编译时或运行时确定的循环进行展开，生成的代码尺寸将变大，执行速度可能变快 也可能变慢。
–fprefetch–loop–arrays
生成数组预读取指令，对于使用巨大数组的程序可以加快代码执行速度，适合数据库 相关的大型软件等。具体效果如何取决于代码。
–fweb
建立经常使用的缓存器网络，提供更佳的缓存器使用率。gcc–3.4中曾属于”–O3”级别的选项。
–ffast–math
违反IEEE/ANSI标准以提高浮点数计算速度，是个危险的选项，仅在编译不需要严格遵守IEEE规 范且浮点计算密集的程序考虑采用。
–fsingle–precision–constant
将浮点常量作为单精度常量对待，而不是隐式地将其转换为双精度。
–fbranch–probabilities
在使用 –fprofile–arcs 选项编译程序并执行它来创建包含每个代码块执行次数的文件之后，程序可以利用这一选项再次编译，文件中所产生的信息将被用来优化那些经常发生的分支代码。 如果没有这些信息，gcc将猜测那一分支可能经常发生并进行优化。这类优化信息将会存放在一个以源文件为名字的并以”.da”为后缀的文件中。
–frename–registers
试图驱除代码中的假依赖关系，这个选项对具有大量寄存器的机器很有效。gcc–3.4中 曾属于”–O3”级别的选项。
–fbranch–target–load–optimize
–fbranch–target–load–optimize2
在 执行序启动以及结尾之前执行分支目标缓存器加载最佳化。

–fstack–protector
在关键函数的堆栈中设置保护值。在返回地址和返回值之前，都将验证这个保护值。如果出现了 缓冲区溢出，保护值不再匹配，程序就会退出。程序每次运行，保护值都是随机的，因此不会被远程猜出。
–fstack–protector–all
同上，但是在所有函数的堆栈中设置保护值。
––param max–gcse–memory=xxM
执行GCSE优化使用的最大内存量(xxM)，太小将使该优化无法进 行，默认为50M。
––param max–gcse–passes=n
执行GCSE优化的最大迭代次数，默认为 1。

传递给汇编器的选项：

–Wa,options
options是一个或多个由逗号分隔的可以传递给汇编器的选项列表。其中的每一个均可作为命令行选项传递给汇编器。
–Wa,––strip–local–absolute
从输出符号表中移除局部绝对符号。
–Wa,–R
合并数据段和正文段，因为不必在数据段和代码段之间转移，所以它可能会产生更短的地址移动。
–Wa,––64
设置字长为64bit，仅用于x86_64，并且仅对ELF格式的目标文件有效。此外，还需要使 用”––enable–64–bit–bfd”选项编译的BFD支持。
–Wa,–march=CPU
按照特定的CPU进行优化：pentiumiii, pentium4, prescott, nocona, core, core2; athlon, sledgehammer, opteron, k8 。

仅可用于 CFLAGS 的选项：

–fhosted
按宿主环境编译，其中需要有完整的标准库，入口必须是main()函数且具有int型的返回值。内核以外几乎 所有的程序都是如此。该选项隐含设置了 –fbuiltin，且与 –fno–freestanding 等价。
–ffreestanding
按独立环境编译，该环境可以没有标准库，且对main()函数没有要求。最典型的例子就是操作系 统内核。该选项隐含设置了 –fno–builtin，且与 –fno–hosted 等价。

仅可用于 CXXFLAGS 的选项：

–fno–enforce–eh–specs
C++标准要求强制检查异常违例，但是该选项可以关闭违例检查，从而减小生成代码 的体积。该选项类似于定义了”NDEBUG”宏。
–fno–rtti
如果没有使用’dynamic_cast’和’typeid’，可以使用这个选项禁止为包含虚方法的类生成 运行时表示代码，从而节约空间。此选项对于异常处理无效(仍然按需生成rtti代码)。
–ftemplate–depth–n
将最大模版实例化深度设为’n’，符合标准的程序不能超过17，默认值为500。
–fno–optional–diags
禁止输出诊断消息，C++标准并不需要这些消息。
–fno–threadsafe–statics
GCC自动在访问C++局部静态变量的代码上加锁，以保证线程安全。如果你不 需要线程安全，可以使用这个选项。
–fvisibility–inlines–hidden
默认隐藏所有内联函数，从而减小导出符号表的大小，既能缩减文件的大 小，还能提高运行性能，我们强烈建议你在编译任何共享库的时候使用该选项。参见 –fvisibility=hidden 选项。

LDFLAGS

LDFLAGS 是传递给连接器的选项。这是一个常被忽视的变量，事实上它对优化的影响也是很明显的。

[提示]以下选项是在完整的阅读了ld-2.18文档之后挑选出来的选项。
http://blog.chinaunix.net/u1/41220/showart_354602.html 有2.14版本的中文手册。

-s
删除可执行程序中的所有符号表和所有重定位信息。其结果与运行命令 strip 所达到的效果相同，这个选项是比较安全的。
-Wl,options
options是由一个或多个逗号分隔的传递给链接器的选项列表。其中的每一个选项均会作为命令行选项提供给链接器。
-Wl,-On
当n>0时将会优化输出，但是会明显增加连接操作的时间，这个选项是比较安全的。
-Wl,--exclude-libs=ALL
不自动导出库中的符号，也就是默认将库中的符号隐藏。
-Wl,-m<emulation>
仿真<emulation>连接器，当前ld所有可用的仿真可以 通过"ld -V"命令获取。默认值取决于ld的编译时配置。
-Wl,--sort-common
把全局公共符号按照大小排序后放到适当的输出节，以防止符号间因为排布限制而出现间隙。
-Wl,-x
删除所有的本地符号。
-Wl,-X
删除所有的临时本地符号。对于大多数目标平台，就是所有的名字以'L'开头的本地符号。
-Wl,-zcomberloc
组合多个重定位节并重新排布它们，以便让动态符号可以被缓存。
-Wl,--enable-new-dtags
在ELF中创建新式的"dynamic tags"，但在老式的ELF系统上无法识别。
-Wl,--as-needed
移除不必要的符号引用，仅在实际需要的时候才连接，可以生成更高效的代码。
-Wl,--no-define-common
限制对普通符号的地址分配。该选项允许那些从共享库中引用的普通符号只在主程序 中被分配地址。这会消除在共享库中的无用的副本的空间，同时也防止了在有多个指定了搜索路径的动态模块在进行运行时符号解析时引起的混乱。
-Wl,--hash-style=gnu
使用gnu风格的符号散列表格式。它的动态链接性能比传统的sysv风格(默认)有 较大提升，但是它生成的可执行程序和库与旧的Glibc以及动态链接器不兼容。

最后说两个与优化无关的系统环境变量，因为会影响GCC编译程序的方式，下面两个是咱中国人比较关心的：

LANG
指定编译程序使用的字符集，可用于创建宽字符文件、串文字、注释；默认为英文。[目前只支持日文"C-JIS,C- SJIS,C-EUCJP"，不支持中文]
LC_ALL
指定多字节字符的字符分类，主要用于确定字符串的字符边界以及编译程序使用何种语言发出诊断消息；默认设置与 LANG相同。中文相关的几项："zh_CN.GB2312，zh_CN.GB18030，zh_CN.GBK，zh_CN.UTF-8，zh_TW.BIG5"

insistYuan
粉丝 - 16 关注 - 2

« 上一篇： 软件开发文档模板 （学习）
» 下一篇： 编译时

编辑推荐：

- CSS 奇思妙想之酷炫倒影
- gRPC 入门与实操（.NET 篇）
- dotnet 代码优化 聊聊逻辑圈复杂度
- 一个棘手的生产问题，但是我写出来之后，就是你的了
- 你可能不知道的容器镜像安全实践

阅读排行：

- 看我是如何用C#编写一个小于8KB的贪吃蛇游戏的
- 一个专科生的 2022 年终总结——默默努力，成为更好的自己
- 工作流引擎架构设计
- Blazor WebAssembly的初次访问慢的优化
- CSS 奇思妙想之酷炫倒影