|  |  |  |
| --- | --- | --- |
| 编号 | GCC选项 | 功能 |
| gdb | -g | debug选项，gdb调试必选选项，默认-g2，可选-g1，-g2，-g3即 -glevel， -ggdblevel |
|  | -fno-omit-frame-pointer | 保留函数调用回溯，避免被优化。 |
|  | .-funwind-tables  -fasynchronous-unwind-tables  -fexceptions  -rdynamic |  |
|  |  |  |
| 程序优化 | -O | 优化选项，-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和时间。  -O2: 此选项将增加编译时间和目标文件的执行性能。  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。  -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。 |
|  |  |  |
| 警告 | -Wall | 打开警告信息，在编译时将所有警告输出。 |
|  | -Wextra | 尽管 gcc 编译器的 -Wall 选项涵盖了绝大多数警告标记，依然有一些警告不能生成。为了生成它们，请使用 -Wextra 选项。 |
|  | -Wfloat-equal | 此选项会强制 gcc 编译器生成一个与浮点值比较相关的警告。 |
|  | -Wno-unused-but-set-variable -Wno-enum-compare -Wno-sizeof-pointer-memaccess -Wno-address  -Wno-pointer-sign | 常用的警告选项，变量被定义以及赋值但逻辑中没有使用等等，此处只列出几个，感兴趣的可以去查gcc手册，警告选项有很多。 |
|  | -Werror | 将警告看做error，消除代码所有隐患，正式release程序时需要此选项保证程序无警告信息。 |
|  |  |  |
| 宏 | -DMACRO | 命令行编译时传递宏 |
|  | -U DEBUG | 命令行编译时取消宏 |
|  |  |  |
|  | \_\_attribute\_\_ | GCC使用\_\_attribute\_\_关键字来描述函数，变量和数据类型的属性，用于编译器对源代码的优化。  下面列举函数属性的几个重要的关键字：  void main\_enter() \_\_attribute\_\_((constructor));//main\_enter函数在进入main函数前调用  void main\_exit() \_\_attribute\_\_((destructor));//main\_exit函数在main函数返回后调用  void fun() \_\_attribute\_\_ ((noinline));//fun函数不能作为inline函数优化  constructor和destructor关键字主要用于剖析(profiling)源代码的，这两个属性还有一个重要作用，就是函数调用堆栈被破坏的比较严重时，可以追踪到函数coredump的点。  \_\_attribute\_\_((deprecated)): deprecated,弃用. 如果在源文件在任何地方地方使用deprecated attribute函数,编译器将会发出警告.  \_\_attribute\_\_((aligned(ALIGNMENT))): 指定变量或结构体最小字节对齐数,以byte为单位.ALIGNMENT: 指定的字节对齐操作数，如果编程时遇到变量的值与预期不一致时可以考虑是否是内存对齐的问题，查看此选项是否设置。  \_\_attribute\_\_((cleanup(cleanup\_function)): 当一个变量的作用域消失时，便会执行后面的clean\_function函数.  \_\_attribute\_\_((packed)): 使变量或者是结构体按照最小的对齐方式，对于变量是1byte对齐，对于字段，也就是field指bit对齐.  上面只是对常见的一些属性操作的解释，对于其他的用法应当参照GCC提供的文档。 |
|  |  |  |
| 程序调优 | -fprofile-arcs  --coverage  -ftest-coverage | 代码覆盖率选项，可以在运行时统计出程序的代码覆盖率，对于测试有一定参考意义，配合gcov/lcov使用。 |
|  | -pg | 改进应用程序的性能是一项非常耗时耗力的工作，但是究竟程序中是哪些函数消耗掉了大部分执行时间，这通常都不是非常明显的。GNU 编译器工具包所提供了一种剖析工具 GNU profiler（gprof）。gprof 可以为 Linux平台上的程序精确分析性能瓶颈。gprof精确地给出函数被调用的时间和次数，给出函数调用关系。配合gcc编译器-pg选项，找出最需优化路径。 |
|  |  |  |
|  | -fdiagnostics-color | 命令行编译时支持彩色输出错误以及警告信息。 |
|  |  |  |
| 程序DEBUG | -fstack-protector-all  -fstack-protector-strong | Debug堆栈溢出选项。 |
|  | -finstrument-functions -finstrument-functions-exclude-function-list=sym,sym,... | 函数调用探测功能,通过-finstrument-functions 获取函数动态调用图，对于快速理解大型程序有一定帮助。 |
|  | -fsanitize=address | 检查内存访问错误，包括buffer读写越界，返回临时内存，全局数据区检查等等一系列内存问题，gcc 4.8版本从clang编译器移植实现。 |
|  | -fsanitize=thread | 检查多线程资源数据竞态问题，包括多线程死锁等等，gcc版本4.8以上支持。 |
|  | -fsanitize=leak | 检查内存泄漏问题，遗憾的是arm平台暂不支持，gcc4.8 以上版本，x86,x86\_x64以及mips平台支持。 |
|  | -fsanitize=undefined  -fsanitize=shift  -fsanitize=integer-divide-by-zero  -fsanitize=unreachable  -fsanitize=vla-bound  -fsanitize=null  -fsanitize=return  -fsanitize=signed-integer-overflow  -fsanitize=bounds  -fsanitize=alignment  -fsanitize=object-size  -fsanitize=float-divide-by-zero  -fsanitize=float-cast-overflow  -fsanitize=nonnull-attribute  -fsanitize=returns-nonnull-attribute  -fsanitize=bool  -fsanitize=enum  -fsanitize=vptr | Sanitize家族指令，帮助找出代码潜在问题，这些不常用，有兴趣的可以研究。 |