gcc警告选项汇总

el/2023/8/31 14:21:58

参考资料:https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html#Optimize-Options

请求或取消警告的选项

警告是诊断消息，报告的结构本质上不是错误的，但是有风险或表明可能有错误。

以下与语言无关的选项不会启用特定的警告，但会控制GCC生成的诊断类型。
-fsyntax-only
检查代码中的语法错误，但除此之外不要做任何事情。

-fmax-errors= n
将错误消息的最大数量限制为n，此时GCC会缓存，而不是尝试继续处理源代码。如果n为0（默认值），则生成的错误消息数量不受限制。如果还指定了-Wfatal-errors，则重大错误优先于此选项。

-w
禁止所有警告消息。

-Werror
使所有的警告进入错误。

-Werror=
将指定的警告转换为错误。附加警告的说明符；例如-Werror =开关将由-Wswitch控制的警告转换为错误。此开关采用否定形式，用于否定-针对特定警告的错误；例如-Wno-error =开关使得-wswitch警告不是错误，即使在-Werror有效时也是如此。

每个可控警告的警告消息都包含控制警告的选项。那么该选项可以与-Werror =和-Wno-error =一起使用，如上所述。（可以使用-fno-diagnostics-show-option标志禁用警告消息中的选项打印。）

请注意，指定-Werror = foo会自动隐含-W foo 。但是，-Wno-error = foo并不意味着什么。

-Wfatal-errors
此选项会导致编译器在发生第一个错误时中止编译，而不是尝试继续并打印更多错误消息。

例如，您可以使用以“ -W ”开头的选项来请求许多特定的警告 - 可以隐式地请求隐式声明的警告。这些特定警告选项中的每一个都有一个以“ -Wno- ”开始的否定形式来关闭警告；例如，-Wno-implicit 。本手册仅列出两种形式中的一种，无论哪种都不是默认值。有关其他语言特定的选项，还可以参考C ++ Dialect Options和Objective-C以及Objective-C ++ Dialect Options 。

某些选项（如-Wall和-Wextra ）会打开其他选项，例如-Wunused ，这可能会启用其他选项，例如-Wunused-value 。正面和负面形式的综合作用是，更具体的选项优先于不特定的选项，与命令行中的位置无关。对于相同特征的选项，最后一个生效。通过编译指示启用或禁用的选项（请参阅诊断编译指示 ）将起作用，就好像它们出现在命令行末尾一样。

当请求无法识别的警告选项时（例如-Wunknown-warning ），GCC将发出诊断，指出该选项未被识别。但是，如果使用-Wno- form，行为会稍有不同：除非正在生成其他诊断，否则不会生成-Wno-unknown-warning诊断。这允许在旧编译器中使用新的-Wno-选项，但如果出现问题，编译器会警告存在无法识别的选项。

-Wpedantic
-pedantic
发布严格的ISO C和ISO C ++所要求的所有警告；拒绝所有使用禁止扩展的程序，以及其他一些不遵循ISO C和ISO C ++的程序。对于ISO C，遵循由所使用的任何-std选项指定的ISO C标准版本。

有效的ISO C和ISO C ++程序应该在有或没有这个选项的情况下正确编译（尽管极少数需要指定所需版本的ISO C的-ansi或-std选项）。但是，如果没有这个选项，也支持某些GNU扩展和传统的C和C ++特性。有了这个选项，他们被拒绝。

-Wpedantic不会导致使用名称以“ __ ”开头和结尾的替代关键字的警告消息。在__extension__之后的表达式中也禁止了迂腐警告。但是，只有系统头文件应该使用这些转义路由；应用程序应该避免它们。请参阅各用关键字 。

一些用户尝试使用-Wantantic来检查程序是否符合严格的ISO C标准。他们很快就发现它并没有达到他们想要的水平：它找到了一些非ISO的做法，但不是所有的只有ISO C 需要诊断的那些做法，以及其他一些诊断已经添加了的做法。

报告任何不符合ISO C标准的功能在某些情况下可能会有用，但需要大量额外的工作，并且与-Wantantic完全不同。我们不打算在不久的将来支持这种功能。

如果用-std指定的标准表示C的GNU扩展方言，如' gnu90 '或' gnu99 '，则存在相应的基本标准，GNU扩展方言所基于的ISO C版本。-Wpedantic给出的警告在基本标准要求的地方给出。（这种警告只适用于不在指定的GNU C方言中的功能是没有意义的，因为根据定义，C的GNU方言包括编译器支持的所有功能，并且没有任何可警告的内容。）

-pedantic-errors
在基本标准（请参阅-Wpedantic ）需要诊断时，在编译时存在未定义行为的某些情况下以及在某些其他情况下不会妨碍编译符合标准的有效程序的情况下发出错误。这不等于-Werror =迂回，因为这个选项启用了错误，而后者没有启用，反之亦然。

-Wall
这使得所有关于某些用户认为可疑的构造的警告都很容易避免（或修改以防止警告），即使与宏结合使用也是如此。这也启用了C ++方言选项和Objective-C和Objective-C ++方言选项中描述的一些特定于语言的警告。

-打开以下警告标志：

```
-Waddress
-Warray-bounds=1 (only with -O2)
-Wbool-compare
-Wbool-operation
-Wc++11-compat  -Wc++14-compat
-Wcatch-value (C++ and Objective-C++ only)
-Wchar-subscripts
-Wcomment
-Wduplicate-decl-specifier (C and Objective-C only)
-Wenum-compare (in C/ObjC; this is on by default in C++)
-Wformat
-Wint-in-bool-context
-Wimplicit (C and Objective-C only)
-Wimplicit-int (C and Objective-C only)
-Wimplicit-function-declaration (C and Objective-C only)
-Winit-self (only for C++)
-Wlogical-not-parentheses
-Wmain (only for C/ObjC and unless -ffreestanding)
-Wmaybe-uninitialized
-Wmemset-elt-size
-Wmemset-transposed-args
-Wmisleading-indentation (only for C/C++)
-Wmissing-attributes
-Wmissing-braces (only for C/ObjC)
-Wmultistatement-macros
-Wnarrowing (only for C++)
-Wnonnull
-Wnonnull-compare
-Wopenmp-simd
-Wparentheses
-Wpointer-sign
-Wreorder
```

最新文章

- Codeforces Round #712 (Div. 2)
- D. Explorer Space (CF#718 (Div. 1 +
- 1467C - Three Bags (CF#695 Div. 2)
- 2021年度训练联盟热身训练赛第七
- Problem L:Lexicographical
- 2021年度训练联盟热身训练赛第八

```
-Wrestrict
-Wreturn-type
-Wsequence-point
-Wsign-compare (only in C++)
-Wsizeof-pointer-div
-Wsizeof-pointer-memaccess
-Wstrict-aliasing
-Wstrict-overflow=1
-Wswitch
-Wtautological-compare
-Wtrigraphs
-Wuninitialized
-Wunknown-pragmas
-Wunused-function
-Wunused-label
-Wunused-value
-Wunused-variable
-Wvolatile-register-var
```

请注意，一些警告标志并不是由-Wall隐含的。其中一些人警告用户通常不认为有问题的建筑，但有时候您可能希望检查; 其他人警告在某些情况下必要或难以避免的构造，并且没有简单的方法来修改代码来抑制警告。其中一些由-Wextra启用，但其中许多必须单独启用。

-Wextra
这会启用一些未由-Wall启用的额外警告标志。（此选项过去称为-W，旧名称仍然受支持，但更新的名称更具描述性。）

```
-Wclobbered
-Wcast-function-type
-Wempty-body
-Wignored-qualifiers
-Wimplicit-fallthrough=3
-Wmissing-field-initializers
-Wmissing-parameter-type (C only)
-Wold-style-declaration (C only)
-Woverride-init
-Wsign-compare (C only)
-Wtype-limits
-Wuninitialized
-Wshift-negative-value (in C++03 and in C99 and newer)
-Wunused-parameter (only with -Wunused or -Wall)
-Wunused-but-set-parameter (only with -Wunused or -Wall)
```

选项-Wextra还会打印以下情况的警告消息：

指针与整数零与<，<=，>或>=。
（仅限C ++）枚举器和非枚举器都出现在条件表达式中。
（仅限C ++）不明确的虚拟基础。
（仅限C ++）为已声明为register的数组下标。
（仅限C ++）取得已声明register的变量的地址。
（仅限C ++）基类不在派生类的复制构造函数中初始化。
-Wchar-subscripts
警告如果数组下标有char类型。这是错误的常见原因，因为程序员经常忘记这种类型是在某些机器上签名的。此警告由-Wall启用。

-Wchkp
警告由指针界限检查器（-fcheck-pointer-bounds）发现的无效内存访问。

-Wno-coverage-mismatch
如果使用-fprofile-use选项时反馈配置文件不匹配，则警告。如果在使用-fprofile-gen编译和使用-fprofile-use编译时源文件发生更改，则具有配置文件反馈的文件可能无法与源文件匹配，并且GCC无法使用配置文件反馈信息。默认情况下，此警告已启用并被视为错误。-Wno-coverage-mismatch可用于禁用警告或-Wno-error = coverage-mismatch可用于禁用该错误。禁用此警告的错误可能会导致代码质量不佳，并且仅在非常小的更改情况下才有用，例如修复现有代码库的错误。不建议完全禁用该警告。

-Wno-cpp
（仅限于Objective-C，C ++，Objective-C ++和Fortran）

禁止#warning指令发出的警告消息。

-Wdouble-promotion (C, C++, Objective-C and Objective-C++ only)
当float类型的值隐式提升为double时发出警告。具有32位“单精度”浮点单元的CPU实现float硬件，但在软件中模拟double精度。在这样的机器上，由于软件仿真所需的开销，使用double值进行计算要昂贵得多。

使用double意外执行计算很容易，因为浮点文字隐含了double类型。例如，在：

```
float area(float radius)
{return 3.14159 * radius * radius;
}
```

编译器使用double执行整个计算，因为浮点文字是double 。

-Wduplicate-decl-specifier (C and Objective-C only)
警告如果声明有重复的const，volatile，restrict或_Atomic说明符。此警告由-Wall启用。

-Wformat
-Wformat= n
检查对printf和scanf等的调用，以确保提供的参数的类型与指定的格式字符串相匹配，并且格式字符串中指定的转换有意义。这包括标准函数，以及其他由printf，scanf，strftime和strfmon（X / Open扩展，而不是C标准）系列（或其他特定于目标系列）的格式属性（请参阅函数属性）指定的标准函数。哪些函数在没有指定格式属性的情况下被检查取决于所选择的标准版本，并且没有指定属性的函数的这些检查由-freestanding或-fno-builtin禁用。

格式将根据GNU libc版本2.2支持的格式特性进行检查。这些包括所有ISO C90和C99功能，以及Single Unix Specification和一些BSD和GNU扩展的功能。其他库实现可能不支持所有这些功能; GCC不支持关于超出特定图书馆限制的功能的警告。但是，如果-Wpedantic与-Wformat 一起使用，则会提供有关格式特征的警告，但不包括在选定的标准版本中（但不包含strfmon格式，因为这些格式不在C标准的任何版本中）。请参阅控制C语言的选项 。

-Wformat=1
-Wformat
选项-Wformat相当于-Wformat = 1，-Wno-format相当于-Wformat = 0 。由于-Wformat还检查几个函数的空格式参数，-Wformat也意味着-Wnonnull 。这种格式检查级别的某些方面可以通过以下选项禁用：-Wno-format-contains-nul，-Wno-format-extra-args和-Wno-format-zero-length 。-Wformat由-Wall启用。

-Wno-format-contains-nul
如果指定了-Wformat，则不要警告有关包含NUL字节的格式字符串。

-Wno-format-extra-args
如果指定了-Wformat，则不要警告有关printf或scanf格式函数的过多参数。C标准指定了这样的参数被忽略。

如果未使用的参数位于用'$'操作数编号规范指定的已使用参数之间，通常仍会给出警告，因为实现无法知道传递给va_arg类型以跳过未使用的参数。但是，在scanf格式的情况下，如果未使用的参数全部是指针，则此选项会抑制警告，因为Single Unix Specification指出允许使用这些未使用的参数。

-Wformat-overflow
-Wformat-overflow= level
警告对可能溢出目标缓冲区的格式化输入/输出函数（如sprintf和vsprintf 。当由格式指令写入的确切字节数在编译时无法确定时，它将根据启发式进行估计，这取决于级别参数和优化。虽然启用优化在大多数情况下会提高警告的准确性，但也可能导致误报。

-Wformat-overflow
-Wformat-overflow=1

-Wformat启用的-Wformat-overflow的第1级采用了一种保守的方法，只警告最有可能溢出缓冲区的调用。在此级别，格式化具有未知值的指令的数字参数假定值为1，未知长度的字符串为空。已知绑定到其类型的子范围的数字参数，或者其输出受其指令的精度或有限的字符串文字限制的字符串参数被假定为取值范围内的值，该范围导致输出中的大部分字节。例如，下面对sprintf的调用被诊断出来，因为即使a和b都等于零，函数附加到目标缓冲区的终止NUL字符（'\0'）也将被写入其末尾。将缓冲区的大小增加一个字节足以避免警告，尽管它可能不足以避免溢出。

```
void f(int a, int b){char buf [13];sprintf(buf, "a =%i, b =%i \ n", a, b);}
```

-Wformat-overflow=2
级别2还会警告有关可能溢出目标缓冲区的调用，因为调用的参数长度或幅度足够大。在级别2处，未知数值参数被假定为具有大于1的精度的带符号类型的最小可表示值，否则为最大可表示值。未知的字符串参数的长度不能被假定为由指令的精度，或者它们可能评估的字符串文字的有限集合，或者它们可能指向的字符数组所限定，都假定为1个字符长。

在第2级，上述示例中的调用再次被诊断，但是这次因为使用等于32位的INT_MIN，第一个%i指令将在目标缓冲区的末尾写入其一些数字。为了保证呼叫安全，无论两个变量的值如何，目标缓冲区的大小必须增加到至少34个字节。GCC在警告后的信息说明中包含缓冲区的最小大小。

增加目标缓冲区大小的替代方法是限制格式化值的范围。字符串参数的最大长度可以通过指定格式指令中的精度来限制。当格式指令的数值参数可以假定为小于它们类型的精度时，为格式说明符选择适当的长度修饰符将减少所需的缓冲区大小。例如，如果上面示例中的a和b可以假定为处于short int类型的精度范围内，那么使用%hi格式指令或将参数强制转换为short可将缓冲区的最大所需大小减少到24个字节。

```
 void f(int a, int b){char buf [23];sprintf(buf, "a =%hi, b =%i \ n", a, (简写)b);}
```

-Wno-format-zero-length
如果指定了-Wformat，则不要警告有关零长度格式。C标准规定允许零长度格式。

-Wformat=2
启用-Wformat加上其他格式检查。目前相当于-Wformat -Wformat-nonliteral -Wformat-security-Wformat-y2k 。

-Wformat-nonliteral
如果指定了-Wformat，则还会警告格式字符串是否不是字符串文字，因此无法检查，除非格式函数将其格式参数作为va_list 。

-Wformat-security
如果指定了-Wformat，还会警告使用表示可能的安全问题的格式化函数。目前，这会警告printf和scanf函数的调用，其中格式字符串不是字符串文字，也没有格式参数，如printf (foo);。如果格式字符串来自不可信输入且包含' %n '，则这可能是安全漏洞。（这是目前W-format-nonliteral提出的警告的一个子集，但将来警告可能会被添加到-Wformat-security中，而不包括在-Wformat-nonliteral中 。）

-Wformat-signedness
如果指定了-Wformat，则还会警告格式字符串是否需要一个无符号参数并且该参数是否有符号，反之亦然。

-Wformat-truncation
-Wformat-truncation= level
警告关于调用可能导致输出截断的格式化输入/输出函数（如snprintf和vsnprintf警告。当由格式指令写入的确切字节数在编译时无法确定时，它将根据启发式进行估计，这取决于级别参数和优化。虽然启用优化在大多数情况下会提高警告的准确性，但也可能导致误报。除非另有说明，该选项使用相同的逻辑 - 格式溢出 。

-Wformat-truncation
-Wformat-truncation=1
-Wformat启用的-Wformat-truncation的 Level 1采用了一种保守的方法，只警告对有界函数的调用，该函数的返回值未使用，并且很可能导致输出截断。

-Wformat-truncation=2
级别2还警告对使用返回值的有界函数的调用，并且在给定足够长度或幅度的参数时可能导致截断。

-Wformat-y2k
如果指定了-Wformat，还会警告有关可能只产生两位数年份的strftime格式。

-Wnonnull
警告标记为非空值函数属性的参数需要传递空指针。

-Wnonnull包含在-Wall和-Wformat中 。它可以通过-Wno-nonnull选项禁用。

-Wnonnull-compare
在比较标记nonnull空函数属性的参数与函数内的空值时发出警告。

-Wnonnull-compare包含在-Wall中 。它可以通过-Wno-nonnull-compare选项禁用。

-Wnull-dereference
如果编译器检测到由于取消引用空指针而导致触发错误或未定义行为的路径，则发出警告。该选项仅在-fdelete-null-pointer-checks处于活动状态时才有效，这是通过大多数目标中的优化启用的。警告的精度取决于所使用的优化选项。

-Winit-self (C, C++, Objective-C and Objective-C++ only)
警告使用自己初始化的未初始化变量。请注意，此选项只能与-Wuninitialized选项一起使用。

例如，只有在指定-Winit-self时，GCC才会在下面的代码片段中提醒i未初始化：

```
  int f(){int i = i;圆报我;}
```

此警告由C ++中的-Wall启用。

-Wimplicit-int (C and Objective-C only)
当声明没有指定类型时发出警告。此警告由-Wall启用。

-Wimplicit-function-declaration (C and Objective-C only)
在声明之前使用函数时发出警告。在C99模式下（ -std = c99或-std = gnu99 ），默认情况下会启用此警告，并通过-pedantic-errors将其设置为错误 。此警告也由-Wall启用。

-Wimplicit (C and Objective-C only)
与-Wimplicit-int和-Wimplicit-function-declaration相同 。此警告由-Wall启用。

-Wimplicit-fallthrough
-Wimplicit- fallthrough与-Wimplicit- fallthrough = 3相同 ， -Wno-implicit-fallthrough与-Wimplicit- fallthrough = 0相同 。

-Wimplicit-fallthrough= n
当开关盒跌落时发出警告。例如：

```
switch (cond){case 1:a = 1;break;case 2:a = 2;case 3:a = 3;break;}
```

当警告的最后一个语句不能通过时，例如当有一个返回语句或对noreturn属性声明的函数调用时，此警告不会警告。-Wimplicit-fallthrough =也考虑到了控制流程语句，比如ifs，并且只在适当的时候发出警告。例如

```
switch (cond){case 1:if (i > 3) {bar (5);break;} else if (i < 1) {bar (0);} elsereturn;default:…}
```

由于在某些情况下可能会出现切换情况，GCC提供了一个属性__attribute__ ((fallthrough))，该属性将与空语句一起使用来抑制通常会发生的警告：

```
switch (cond){case 1:bar (0);__attribute__ ((fallthrough));default:…}
```

C ++ 17提供了一种标准的方法来抑制使用[[fallthrough]];的-Wimplicit-Fallthrough警告[[fallthrough]]; 而不是GNU属性。在C ++ 11或C ++ 14中，用户可以使用[[gnu::fallthrough]];，这是一个GNU扩展。除了这些属性外，还可以添加一条跌落评论以使警告消失。C或C ++样式注释的整个主体应该与下面列出的给定正则表达式匹配。选项参数n指定接受哪种注释：

-Wimplicit-Fallthrough = 0完全禁用警告。
-Wimplicit-fallthrough = 1匹配.*正则表达式*，*任何评论都被用作fallthrough评论*。
-wimplicit-fallthrough = 2 case nonnsitively matches .falls?[\t-]thr(ough|u).正则表达式。
-Wimplicit-fallthrough = 3大小写敏感地匹配以下正则表达式之一：
-fallthrough
@fallthrough@
lint -fallthrough[\t]
[\t.!](ELSE,? |INTENTIONAL(LY)?)?
FALL(S | |-)?THR(OUGH|U)[\t.!](-[^\n\r])?
[\t.!](Else,? |Intentional(ly)?)?
Fall((s | |-)[Tt]|t)hr(ough|u)[\t.!](-[^\n\r])?
[\t.!]([Ee]lse,? |[Ii]ntentional(ly)?)?
fall(s | |-)?thr(ough|u)[\t.!](-[^\n\r])?
-Wimplicit-fallthrough = 4大小写敏感地匹配以下正则表达式之一：
-fallthrough
@fallthrough@
lint -fallthrough[\t]*
[\t]*FALLTHR(OUGH|U)[\t]*
-Wimplicit-Fallthrough = 5不会将任何注释识别为传递注释，只有属性禁用警告。
注释需要在可选空白和其他注释之后遵循case或default关键字或者某个case或default标签之前的用户标签。

```
switch (cond){case 1:bar (0);/* FALLTHRU */default:…}
```

-Wimplicit-Fallthrough = 3警告由-Wextra启用。

-Wif-not-aligned (C, C++, Objective-C and Objective-C++ only)
控制是否应发出由warn_if_not_aligned属性触发的警告。这是默认启用的。使用-Wno-if-not-aligned来禁用它。

-Wignored-qualifiers (C and C++ only)
如果函数的返回类型具有类型限定符（如const，则发出警告。对于ISO C这样的类型限定符没有效果，因为函数返回的值不是左值。对于C ++来说，警告只是针对标量类型或void发出的。ISO C禁止在函数定义上使用合格的void返回类型，所以这种返回类型总是会在没有这个选项的情况下收到警告。

此警告也由-Wextra启用。

-Wignored-attributes (C and C++ only)
当属性被忽略时发出警告。这与-Wattributes选项不同，它会在编译器决定删除属性时发出警告，而不是该属性未知，在错误的地方使用等。此警告默认情况下处于启用状态。

-Wmain
如果main类型可疑，则发出警告。main应该是一个具有外部链接的函数，返回int，可以使用零参数，两个或三个适当类型的参数。此警告在C ++中是默认启用的，可以通过-Wall或-Wpedantic来启用。

-Wmisleading-indentation (C and C++ only)
当代码的缩进不反映块结构时发出警告。具体而言，针对if，else，while和for子句发出警告，其中警告语句不使用大括号，后跟具有相同缩进的未保护语句。

在下面的例子中，对“bar”的调用被误导地缩进，就好像它受到“if”条件的保护。

```
  if(some_condition())foo();bar();  /* Gotcha: this is not guarded by the "if".  */
```

在混合制表符和空格的情况下，警告使用-ftabstop =选项来确定语句是否排队（默认为8）。

对于涉及多行预处理器逻辑的代码（如以下示例），不会发出该警告。

```
  if (flagA)foo (0);
#if SOME_CONDITION_THAT_DOES_NOT_HOLDif (flagB)
#endiffoo (1);
```

该警告不会在#line指令之后发出，因为这通常会指示自动生成的代码，并且不会假定该指令所引用的文件的布局。

此警告是由C和C ++中的-Wall启用的。

-Wmissing-attributes
当函数声明缺少一个或多个属性时，警告声明相关函数，并且其缺少可能会对生成的代码的正确性或效率产生负面影响。例如，在C ++中，当使用属性alloc_align，assume_aligned，assume_aligned，format，format_arg，malloc或assume_aligned声明的主模板的显式特化未声明时，会发出警告。deprecated属性，error和warning抑制警告。（请参阅功能属性）。

-Wmissing-attributes由-Wall启用。

例如，由于下面的主要函数模板的声明使用属性malloc和alloc_size，因此模板的显式特化的声明被诊断，因为它缺少其中一个属性。

```
template <class T>
T* __attribute__ ((malloc, alloc_size (1)))
allocate (size_t);template <>
void* __attribute__ ((malloc))   // missing alloc_size
allocate<void> (size_t);
```

-Wmissing-braces
如果聚合或联合初始值设定项没有完全包围，则发出警告。在以下示例中，a的初始值设定项未完全包围，但b的完全包围。此警告由C中的-Wall启用

```
  int a [2] [2] = {0,1,2,3};int b [2] [2] = {{0,1}. {2,3}};
```

此警告由-Wall启用。

-Wmissing-include-dirs (C, C++, Objective-C and Objective-C++ only)
如果用户提供的包含目录不存在，则发出警告。

-Wmultistatement-macros
警告不安全的多语句宏，这些宏看起来像if，else，for，switch或while这样的子句可以保护，其中只有第一条语句在扩展宏后才实际被保护。

例如：

```
#define DOIT x++; y++
if (c)DOIT;
```

将无条件地增加y，而不仅仅是当c成立时。通常可以通过将宏包装在do-while循环中来解决该问题：

```
#define DOIT do { x++; y++; } while (0)
if (c)DOIT;
```

此警告是由C和C ++中的-Wall启用的。

-Wparentheses
如果在某些上下文中省略了括号，例如当预期有真值的上下文中存在赋值时，或者运算符嵌套，其优先级人员经常会感到困惑时，会发出警告。

还会发出如x<=y<=z等比较结果的警告。这相当于(x<=y ? 1 : 0) <= z，这是与普通数学符号的解释不同的解释。

还警告GNU扩展的危险用途?:省略中间操作数。当条件在? :运算符是一个布尔表达式，省略的值始终为1.程序员通常希望它是在条件表达式内计算的值。

对于C ++，这也会在声明中警告一些不必要的括号，这可能表示试图在函数调用而不是声明：

```
{// Declares a local variable called mymutex.std::unique_lock<std::mutex> (mymutex);// User meant std::unique_lock<st
}
```

此警告由-Wall启用。

-Wsequence-point
由于违反了C和C ++标准中的顺序点规则，因此可能会有未定义语义的代码发出警告。

C和C ++标准定义了C / C ++程序中的表达式按顺序点进行评估的顺序，顺序点表示执行程序各部分之间的部分顺序：在顺序点之前执行的顺序点和在执行之后执行的顺序点它。这些发生在对一个&&，||的第一个操作数进行评估之后，对一个完整表达式（不是更大表达式的一部分）进行评估之后。，？：？：或（逗号）运算符，然后调用一个函数（但在其参数和表达被调用函数的表达式之后）以及某些其他地方。除了顺序点规则所表达的内容之外，没有指定表达式的子表达式的评估顺序。所有这些规则只描述部分顺序而不是全部顺序，例如，如果在一个表达式中调用了两个函数，而它们之间没有顺序点，则函数被调用的顺序未被指定。但是，标准委员会已经裁定函数调用不重叠。

在序列点之间不会指定对对象值的修改生效。行为依赖于此的程序具有未定义的行为; C和C ++标准规定："在前一个和下一个序列点之间，一个对象最多应该通过评估一个表达式修改其存储值。此外，先前值只能读取，以确定要存储的值。" 如果一个程序违反了这些规则，任何特定实现的结果都是完全不可预测的。

具有未定义行为的代码示例是a = a++;，a[n] = b[n++]和a[i++] = i;。一些更复杂的病例不会被这个选项诊断出来，它可能会偶尔出现假阳性结果，但总的来说，在检测程序中的这类问题时发现它相当有效。

C ++ 17标准将在更多情况下定义操作数的评估顺序；特别是它要求在左侧之前评估赋值的右侧，所以上面的例子不再是未定义的。但是这个警告仍然会警告他们，帮助人们避免编写C语言和早期C ++版本中未定义的代码。

这个标准的措辞令人困惑，因此在微妙的情况下，对序列点规则的确切含义存在一些争议。有关问题讨论的链接，包括提出的正式定义，可以在GCC的阅读页面上找到，网址为http://gcc.gnu.org/readings.html 。

对于C和C ++，此警告由-Wall启用。

-Wno-return-local-addr
不要警告在函数返回后将指针（或C ++，引用）返回到超出范围的变量。

-Wreturn-type
每当使用默认为int的返回类型定义函数时发出警告。还会在返回类型不为void的函数中返回没有返回值的return语句（从函数体的末尾落下被认为没有值返回）。

仅对于C，在函数的返回类型为void，使用表达式的return语句发出警告，除非表达式类型也为void 。作为GNU扩展，除非使用-Wantantic，否则后一种情况会被接受而不会有警告。

对于C ++，即使指定了-Wno-return-type，没有返回类型的函数也会生成诊断消息。唯一的例外是在系统头文件中定义的main和函数。

此警告由-Wall启用。

-Wshift-count-negative
如果移位计数为负，则发出警告。该警告默认启用。

-Wshift-count-overflow
如果移位计数> =类型的宽度，则发出警告。该警告默认启用。

-Wshift-negative-value
警告如果左移一个负值。-Wextra在C99和C ++ 11模式（和更新版本）中启用此警告。

-Wshift-overflow
-Wshift-overflow= n
警告左移溢出。此警告在C99和C ++ 11模式（及更新版本）中默认启用。

-Wshift-overflow=1
这是-Wshift-overflow的警告级别，并且在C99和C ++ 11模式（和更新版本）中默认启用。此警告级别不警告将左移1到符号位。（但是，在C语言中，在需要整型常量表达式的上下文中仍会拒绝这样的溢出。）

-Wshift-overflow=2
除非C ++ 14模式处于活动状态，否则此警告级别还会警告左移符号位1。

-Wswitch
如果switch语句具有枚举类型的索引并且缺少该枚举的一个或多个指定代码的case，则发出警告。（default标签的存在会阻止此警告。）case使用此选项时（即使有default标签），枚举范围外的标签也会引发警告。此警告由启用-Wall 。

-Wswitch-default
每当switch陈述没有default案件时警告。

-Wswitch-enum
每当switch语句具有枚举类型的索引并且缺少该枚举case的一个或多个指定代码时发出警告。case枚举范围外的标签在使用此选项时也会引发警告。唯一的区别-Wswitch此选项是即使存在default标签，此选项也会提供有关省略的枚举代码的警告。

-Wswitch-bool
只要switch语句具有布尔类型的索引并且大小写值超出布尔类型的范围，就会发出警告。可以通过将控制表达式转换为除以外的类型来抑制此警告bool。例如：

```
switch ((int) (a == 4)){…}
```

C和C ++程序默认启用此警告。

-Wswitch-unreachable
只要switch语句包含控制表达式和第一个case标签之间的语句，即永远不会执行的语句，就会发出警告。例如：

```
switch (cond){i = 15;…case 5:…}
```

-Wswitch不可达 如果控件表达式和第一个case标签之间的声明只是一个声明，则不会发出警告：

```
switch (cond){int i;…case 5:i = 5;…}
```

C和C ++程序默认启用此警告。

-Wsync-nand (C and C++ only)
当警告__sync_fetch_and_nand和__sync_nand_and_fetch使用的内置功能。这些函数改变了GCC 4.4中的语义。

-Wunused-but-set-parameter
每当一个函数参数被赋值时发出警告，但是除此之外不用（除了声明）。

要抑制此警告，请使用该unused属性（请参阅变量属性）。

此警告也由启用 -Wunused 和...一起 -Wextra 。

-Wunused-but-set-variable
每当一个局部变量被分配时发出警告，但在其他情况下不用（除了声明）。此警告由启用-Wall 。

要抑制此警告，请使用该unused属性（请参阅变量属性）。

此警告也由启用 -Wunused ，这是启用的 -Wall 。

-Wunused-function
每当声明静态函数但未定义或未使用非内联静态函数时发出警告。此警告由启用-Wall 。

-Wunused-label
只要标签被声明但未被使用，就会发出警告。此警告由启用-Wall 。

要抑制此警告，请使用该unused属性（请参阅变量属性）。

-Wunused-local-typedefs (C, Objective-C, C++ and Objective-C++ only)
当没有使用在函数中本地定义的typedef时发出警告。此警告由启用-Wall 。

-Wunused-parameter
除了声明之外，只要函数参数没有被使用，就会发出警告。

要抑制此警告，请使用该unused属性（请参阅变量属性）。

-Wno-unused-result
不要警告标记了属性的函数的调用者是否使用它的返回值warn_unused_result（请参阅函数属性）。默认是-Wunused-结果 。

-Wunused-variable
只要本地或静态变量不在声明中使用，就会发出警告。这个选项意味着-Wunused const的变量= 1为C，但不适用于C ++。此警告由启用-Wall 。

要抑制此警告，请使用该unused属性（请参阅变量属性）。

-Wunused-const-variable
-Wunused-const-variable= n
警告只要一个常量静态变量不用于声明就可以使用。-Wunused const的变量= 1 由...启用 -Wunused可变为C，但不适用于C ++。在C中声明了变量存储，但在C ++中，这不是一个错误，因为const变量取代了#defines。

要抑制此警告，请使用该unused属性（请参阅变量属性）。

-Wunused-const-variable=1
这是由启用的警告级别 -Wunused可变 它只针对主编译单元中定义的未使用的静态常量变量发出警告，但不包括有关任何头中声明的静态常量变量。

-Wunused-const-variable=2
此警告级别还警告标题中的未使用的常量静态变量（不包括系统标题）。这是警告级别-Wunused const的可变 并且必须明确要求，因为在C ++中这不是错误，而在C中可能很难清理包含的所有头文件。

-Wunused-value
当语句计算明确未使用的结果时发出警告。要抑制此警告，请将未使用的表达式转换为void。这包括表达式语句或不包含副作用的逗号表达式的左侧。例如，表达式x[i,j]会导致警告，但x[(void)i,j]不会。

此警告由启用 -Wall 。

-Wunused
All the above -Wunused options combined.

In order to get a warning about an unused function parameter, you must either specify -Wextra -Wunused (note that -Wall implies -Wunused), or separately specify -Wunused-parameter .

-Wuninitialized
Warn if an automatic variable is used without first being initialized or if a variable may be clobbered by a setjmp call. In C++, warn if a non-static reference or non-static const member appears in a class without constructors.

If you want to warn about code that uses the uninitialized value of the variable in its own initializer, use the -Winit-self option.

These warnings occur for individual uninitialized or clobbered elements of structure, union or array variables as well as for variables that are uninitialized or clobbered as a whole. They do not occur for variables or elements declared volatile . Because these warnings depend on optimization, the exact variables or elements for which there are warnings depends on the precise optimization options and version of GCC used.

Note that there may be no warning about a variable that is used only to compute a value that itself is never used, because such computations may be deleted by data flow analysis before the warnings are printed.

-Winvalid-memory-model
Warn for invocations of __atomic Builtins , __sync Builtins , and the C11 atomic generic functions with a memory consistency argument that is either invalid for the operation or outside the range of values of the memory_order enumeration. For example, since the __atomic_store and __atomic_store_n built-ins are only defined for the relaxed, release, and sequentially consistent memory orders the following code is diagnosed:

```
void store (int *i){__atomic_store_n (i, 0, memory_order_consume);}
```

-Winvalid-memory-model is enabled by default.

-Wmaybe-uninitialized
对于自动（即局部）变量，如果存在从函数入口到被初始化的变量的使用的路径，但是存在其他变量未初始化的其他路径，则编译器在不能初始化的情况下发出警告证明在运行时未执行未初始化的路径。

这些警告只能在优化编译时使用，否则GCC不会跟踪变量的状态。

这些警告是可选的，因为即使出现错误，GCC也可能无法确定代码何时正确。以下是可能发生的一个例子：

```
{int x;switch (y){case 1: x = 1;break;case 2: x = 4;break;case 3: x = 5;}foo (x);
}
```

如果值y始终为1,2或3，则x始终进行初始化，但GCC不知道这一点。要取消警告，您需要提供一个默认情况下的断言（0）或类似的代码。

此选项还会在非易失性自动变量被呼叫更改时发出警告longjmp。编译器只能看到呼叫setjmp。它不知道哪里longjmp会叫；实际上，信号处理程序可以在代码中的任何位置调用它。因此，即使实际上没有问题，您也可能会收到警告，因为longjmp实际上不会在会导致问题的地方调用。

如果您声明所有永远不会返回的函数，则可以避免一些虚假警告noreturn。请参阅功能属性。

此警告由启用 -Wall 要么 -Wextra 。

-Wunknown-pragmas

> **pragma遇到GCC无法理解的指令时发出警告。如果使用此命令行选项，则甚至会为系统头文件中的未知编译指示发出警告。如果只有警告才被启用，情况并非如此-Wall 命令行选项。**

-Wno-pragmas
不要警告有关编译指示的误用，例如参数不正确，语法无效或编译指示冲突。也可以看看-Wunknown的编译指示 。

-Wstrict-aliasing
该选项仅在激活时才有效 -fstrict走样活跃。它警告可能会破坏编译器用于优化的严格别名规则的代码。警告并没有涵盖所有情况，但确实试图抓住更常见的陷阱。它包含在中-Wall。这相当于-Wstrict-aliasing=3.

-Wstrict-aliasing=n
该选项仅在激活时才有效 -fstrict走样 is active. It warns about code that might break the strict aliasing rules that the compiler is using for optimization. Higher levels correspond to higher accuracy (fewer false positives). Higher levels also correspond to more effort, similar to the way -O works. -Wstrict-aliasing is equivalent to -Wstrict-aliasing=3 .

Level 1: Most aggressive, quick, least accurate. Possibly useful when higher levels do not warn but -fstrict-aliasing still breaks the code, as it has very few false negatives. However, it has many false positives. Warns for all pointer conversions between possibly incompatible types, even if never dereferenced. Runs in the front end only.

Level 2: Aggressive, quick, not too precise. May still have many false positives (not as many as level 1 though), and few false negatives (but possibly more than level 1). Unlike level 1, it only warns when an address is taken. Warns about incomplete types. Runs in the front end only.

Level 3 (default for -Wstrict-aliasing): Should have very few false positives and few false negatives. Slightly slower than levels 1 or 2 when optimization is enabled. Takes care of the common pun+dereference pattern in the front end: *(int)*&some_float . If optimization is enabled, it also runs in the back end, where it deals with multiple statement cases using flow-sensitive points-to information. Only warns when the converted pointer is dereferenced. Does not warn about incomplete types.

-Wstrict-overflow
-Wstrict-overflow= n
This option is only active when signed overflow is undefined. It warns about cases where the compiler optimizes based on the assumption that signed overflow does not occur. Note that it does not warn about all cases where the code might overflow: it only warns about cases where the compiler implements some optimization. Thus this warning depends on the optimization level.

An optimization that assumes that signed overflow does not occur is perfectly safe if the values of the variables involved are such that overflow never does, in fact, occur. Therefore this warning can easily give a false positive: a warning about code that is not actually a problem. To help focus on important issues, several warning levels are defined. No warnings are issued for the use of undefined signed overflow when estimating how many iterations a loop requires, in particular when determining whether a loop will be executed at all.

-Wstrict-overflow=1
Warn about cases that are both questionable and easy to avoid. For example the compiler simplifies x + 1 > x to 1 . This level of -Wstrict-overflow is enabled by -Wall ; higher levels are not, and must be explicitly requested.

-Wstrict-overflow=2
Also warn about other cases where a comparison is simplified to a constant. For example: abs (x) >= 0 . This can only be simplified when signed integer overflow is undefined, because abs (INT_MIN) overflows to INT_MIN , which is less than zero. -Wstrict-overflow (with no level) is the same as -Wstrict-overflow=2 .

-Wstrict-overflow=3
Also warn about other cases where a comparison is simplified. For example: x + 1 > 1 is simplified to x > 0 .

-Wstrict-overflow=4
Also warn about other simplifications not covered by the above cases. For example: (x * 10) / 5 is simplified to x * 2 .

-Wstrict-overflow=5
Also warn about cases where the compiler reduces the magnitude of a constant involved in a comparison. For example: x + 2 > y is simplified to x + 1 >= y . This is reported only at the highest warning level because this simplification applies to many comparisons, so this warning level gives a very large number of false positives.

-Wstringop-overflow
-Wstringop-overflow= type
Warn for calls to string manipulation functions such as memcpy and strcpy that are determined to overflow the destination buffer. The optional argument is one greater than the type of Object Size Checking to perform to determine the size of the destination. See Object Size Checking . The argument is meaningful only for functions that operate on character arrays but not for raw memory functions like memcpy which always make use of Object Size type-0. The option also warns for calls that specify a size in excess of the largest possible object or at most SIZE_MAX / 2 bytes. The option produces the best results with optimization enabled but can detect a small subset of simple buffer overflows even without optimization in calls to the GCC built-in functions like __builtin_memcpy that correspond to the standard functions. In any case, the option warns about just a subset of buffer overflows detected by the corresponding overflow checking built-ins. For example, the option will issue a warning for the strcpy call below because it copies at least 5 characters (the string “blue” including the terminating NUL) into the buffer of size 4.

```
enum Color { blue, purple, yellow };
const char* f (enum Color clr)
{static char buf [4];const char *str;switch (clr){case blue: str = "blue"; break;case purple: str = "purple"; break;c
}
```

Option -Wstringop-overflow=2 is enabled by default.

-Wstringop-overflow
-Wstringop-overflow=1
The -Wstringop-overflow=1 option uses type-zero Object Size Checking to determine the sizes of destination objects. This is the default setting of the option. At this setting the option will not warn for writes past the end of subobjects of larger objects accessed by pointers unless the size of the largest surrounding object is known. When the destination may be one of several objects it is assumed to be the largest one of them. On Linux systems, when optimization is enabled at this setting the option warns for the same code as when the _FORTIFY_SOURCE macro is defined to a non-zero value.

-Wstringop-overflow=2
The -Wstringop-overflow=2 option uses type-one Object Size Checking to determine the sizes of destination objects. At this setting the option will warn about overflows when writing to members of the largest complete objects whose exact size is known. It will, however, not warn for excessive writes to the same members of unknown objects referenced by pointers since they may point to arrays containing unknown numbers of elements.

-Wstringop-overflow=3
The -Wstringop-overflow=3 option uses type-two Object Size Checking to determine the sizes of destination objects. At this setting the option warns about overflowing the smallest object or data member. This is the most restrictive setting of the option that may result in warnings for safe code.

-Wstringop-overflow=4
The -Wstringop-overflow=4 option uses type-three Object Size Checking to determine the sizes of destination objects. At this setting the option will warn about overflowing any data members, and when the destination is one of several objects it uses the size of the largest of them to decide whether to issue a warning. Similarly to -Wstringop-overflow=3 this setting of the option may result in warnings for benign code.

-Wstringop-truncation
Warn for calls to bounded string manipulation functions such as strncat , strncpy , and stpncpy that may either truncate the copied string or leave the destination unchanged.

In the following example, the call to strncat specifies a bound that is less than the length of the source string. As a result, the copy of the source will be truncated and so the call is diagnosed. To avoid the warning use bufsize - strlen (buf) - 1) as the bound.

```
 void append (char *buf, size_t bufsize){strncat (buf, ".txt", 3);}
```

As another example, the following call to strncpy results in copying to d just the characters preceding the terminating NUL, without appending the NUL to the end. Assuming the result of strncpy is necessarily a NUL-terminated string is a common mistake, and so the call is diagnosed. To avoid the warning when the result is not expected to be NUL-terminated, call memcpy instead.

```
 void copy (char *d, const char *s){strncpy (d, s, strlen (s));}
```

In the following example, the call to strncpy specifies the size of the destination buffer as the bound. If the length of the source string is equal to or greater than this size the result of the copy will not be NUL-terminated. Therefore, the call is also diagnosed. To avoid the warning, specify sizeof buf - 1 as the bound and set the last element of the buffer to NUL .

```
 void copy (const char *s){char buf[80];strncpy (buf, s, sizeof buf);...}
```

在字符数组旨在存储字节序列且不终止NUL的情况下，可以使用属性nonstring对其进行注释以避免此警告。但是，这样的数组对于期望NUL终止字符串的函数不适合。为了帮助检测这种阵列的意外滥用，GCC发出警告，除非它能证明使用是安全的。请参阅常用变量属性。

-Wsuggest-attribute= [pure | const | noreturn | format | cold | malloc]
警告添加属性可能会有好处的情况。下面列出了当前支持的属性。

-Wsuggest-attribute=pure
-Wsuggest-attribute=const
-Wsuggest-attribute=noreturn
-Wsuggest-attribute=malloc
发出警告的功能，可能是属性候选人pure，const或noreturn或malloc。编译器只警告在其他编译单元或（在的情况下，可见功能pure和const），如果它不能证明该功能正常返回。如果函数不包含无限循环或通过抛出，调用abort或陷印异常返回，函数将正常返回。该分析需要选项-fipa纯const的，这在默认情况下是启用的 -O和更高。更高的优化级别可以提高分析的准确性。

-Wsuggest-attribute=format
-Wmissing-format-attribute
警告可能是format属性候选的函数指针。请注意，这些只是可能的候选人，而不是绝对的。GCC猜测具有format赋值，初始化，参数传递或返回语句中使用的属性的函数指针format在结果类型中应该具有相应的属性。即分配或初始化的左侧，参数变量的类型或包含函数的返回类型应分别具有一个format属性以避免警告。

海湾合作委员会也警告可能是format属性候选的函数定义。再次，这些只是可能的候选人。GCC猜测format属性可能适用于任何调用类似vprintfor 的函数vscanf，但这可能不总是这种情况，并且format可能不会检测到属性适当的某些函数。

-Wsuggest-attribute=cold
警告可能是cold属性候选项的函数。这是基于静态检测的，一般只会警告那些总是会导致调用另一个cold函数的函数，比如C ++的包装器throw或致命的错误报告函数abort。

-Wsuggest-final-types
如果类型是用C ++ 11 final说明符声明的，或者如果可能的话，声明在匿名名称空间中，则可通过虚拟方法警告代码质量将得到改进的类型。这允许GCC更积极地虚拟化多态呼叫。链接时间优化时，此警告更有效，其中关于类层次结构图的信息更加完整。

-Wsuggest-final-methods
如果方法是用C ++ 11 final说明符声明的，或者如果可能的话，它的类型是在匿名名称空间或说明final符中声明的，则会警告虚拟方法的代码质量会提高。对于链接时优化，此警告更有效，其中关于类层次结构图的信息更加完整。建议首先考虑建议-Wsuggest决赛类型然后用新的注释重建。

-Wsuggest-override
警告重写未使用override关键字标记的虚拟函数。

-Walloc-zero
警告调用饰属性分配函数alloc_size指定零个字节，包括那些功能内置形式aligned_alloc，alloca，calloc，malloc，和realloc。由于这些函数在零大小调用时的行为在不同实现中（并且在realloc已被弃用的情况下）有所不同，因此依赖它可能会导致巧妙的可移植性错误，因此应该避免。

-Walloc-size-larger-than= n
警告关于对使用属性装饰的函数的调用，alloc_size该属性尝试分配大于指定字节数的对象，或者以无限精度的整数类型的大小计算结果超过该值SIZE_MAX / 2。在选项参数N可以在标准后缀指定字节的倍数中的一个端部如kB和KiB为千字节和kibibyte分别MB和MiB为兆字节和mebibyte，等等。请参阅功能属性。

-Walloca
此选项会警告alloca源中的所有用途。

-Walloca-larger-than= n
此选项会对调用发出警告，该调用alloca不受限制其整数类型参数为最多n个字节的控制谓词限制，或者调用alloca限制未知的位置。非整数类型的参数被认为是无界限的，即使它们似乎受限于预期的范围。

例如，一个有界的情况alloca可能是：

```
void func (size_t n)
{void *p;if (n <= 1000)p = alloca (n);elsep = malloc (n);f (p);
}
```

在上面的示例中，传递-Walloca-larger-than=1000不会发出警告，因为调用alloca已知最多为1000个字节。但是，如果-Walloca-larger-than=500通过，编译器会发出警告。

另一方面，无界用途是alloca没有控制谓词约束其整数参数的用法。例如：

```
 void func(){void * p = alloca(n);f(p);}
```

如果-Walloca-larger-than=500通过，上述情况会触发警告，但这次是因为缺少边界检查。

请注意，即使看起来正确的代码涉及有符号整数可能会导致警告：

```
void func (signed int n)
{if (n < 500){p = alloca (n);f (p);}
}
```

在上面的例子中，n可能是负数，导致比期望的参数更隐含地投入到alloca调用中。

此选项还会alloca在循环中使用时发出警告。

此警告未启用 -Wall ，并且只在有效时才有效 -ftree-VRP 是活动的（默认为 -O2 以上）。

也可以看看 -Wvla-greater-than = n 。

-Warray-bounds
-Warray-bounds= n
该选项仅在激活时才有效 -ftree-VRP 是活动的(默认为 -02以上)。它警告下标总是超出范围的数组。此警告由启用-Wall 。

-Warray-bounds=1
这是警告级别 -Warray界 并被启用 -Wall; 更高层次不是，并且必须明确要求。

-Warray-bounds=2
此警告级别还会警告结构体末端的数组以及通过指针访问的数组的出界限。此警告级别可能会导致大量的误报，并且默认情况下会停用。

-Wattribute-alias
警告使用alias目标与别名类型不兼容的类似属性的声明。请参阅声明函数的属性。

-Wbool-compare
与不同于true/ 的整数值相比，关于布尔表达式的警告false。例如，以下比较始终是错误的：

```
int n = 5;
…
if ((n > 1) == 2) { … }
```

此警告由启用 -Wall 。

-Wbool-operation
警告布尔类型表达式的可疑操作。例如，按位取反布尔值很可能是程序中的一个错误。对于C来说，这个警告还会警告增加或减少布尔值，这很少有意义。(在C ++中，递减布尔值总是无效的。在C ++ 17中递增布尔值是无效的，否则不推荐使用。)

此警告由启用 -Wall 。

-Wduplicated-branches
当if-else具有相同的分支时发出警告。此警告检测类似的情况

```
if (p != NULL)return 0;
elsereturn 0;
```

当两个分支只包含一个空语句时它不会发出警告。此警告还警告有条件的操作员：

```
int i = x?* p:* p;
```

-Wduplicated-cond
在if-else-if链中警告重复的条件。例如，警告以下代码：

```
if (p->q != NULL) { … }
else if (p->q != NULL) { … }
```

-Wframe-address
警告当' __builtin_frame_address ' 要么 ' __builtin_return_address'被调用的参数大于0.这种调用可能会返回不确定的值或崩溃程序。警告包含在-Wall 。

-Wno-discarded-qualifiers (C and Objective-C only)
不要警告指针上的类型限定符是否被丢弃。通常，编译器会警告const char *变量是否传递给了一个带char *参数的函数。这个选项可以用来抑制这样的警告。

-Wno-discarded-array-qualifiers (C and Objective-C only)
不要警告在指针目标数组上的类型限定符是否被丢弃。通常，编译器会警告const int (*)[]变量是否传递给了一个带int (*)[]参数的函数。这个选项可以用来抑制这样的警告。

-Wno-incompatible-pointer-types (C and Objective-C only)
当不兼容类型的指针之间发生转换时，不要发出警告。此警告适用于未涵盖的情况-Wno指针-SIGN ，它警告指针参数传递或具有不同签名的分配。

-Wno-int-conversion (C and Objective-C only)
不要警告不兼容的整数指针和指向整数转换的指针。此警告是关于隐式转换的; 用于显式转换警告-Wno-INT到指针铸造 和 -Wno指针到INT-投 可能用过了。

-Wno-div-by-zero
不要警告编译时的整数除零。零点浮点除法没有被警告，因为它可以是获得无穷大和NaN的合法方式。

-Wsystem-headers
打印系统头文件中的结构的警告消息。系统头文件中的警告通常会被抑制，假设它们通常不表示真正的问题，并且只会使编译器输出更难以阅读。使用这个命令行选项可以告诉GCC从系统标题中发出警告，就好像它们出现在用户代码中一样。但是，请注意使用-Wall与此选项一起并没有警告的系统头，对于未知编译指示，-Wunknown的编译指示 也必须使用。

-Wtautological-compare
警告如果自我比较总是评估为真或假。此警告检测各种错误，如：

```
int i = 1;
…
if (i > i) { … }
```

此警告还会警告总是评估为真或假的比特比较，例如：

```
if ((a & 16) == 10) { … }
```

将永远是错误的。

此警告由启用 -Wall 。

-Wtrampolines
警告为指向嵌套函数生成的蹦床。蹦床是在运行时在堆栈上创建的一小段数据或代码，当嵌套函数的地址被采用时，用于间接调用嵌套函数。对于某些目标，它仅由数据组成，因此不需要特殊处理。但是，对于大多数目标来说，它是由代码组成的，因此需要使堆栈可执行才能使程序正常工作。

-Wfloat-equal
如果在相等比较中使用浮点值，则发出警告。

这背后的想法是，有时候对于程序员来说，将浮点值视为对无限精确实数的逼近是很方便的。如果你这样做，那么你需要计算(通过分析代码，或者以其他方式)计算引入的最大或可能的最大误差，并且在执行比较时允许它(并且当产生输出时，但是这是一个不同的问题)。特别是，您应该检查两个值是否具有重叠范围，而不是测试相等性。这是通过关系运算符完成的，所以平等比较可能是错误的。

-Wtraditional (C and Objective-C only)
警告某些在传统和ISO C中表现不同的结构。还警告有关没有传统C等价物的ISO C结构和/或应避免的有问题的结构。

在宏体中出现在字符串文字中的宏参数。在传统的C宏替换发生在字符串文字中，但在ISO C中不。
在传统的C中，一些预处理器指令不存在。传统的预处理器只有在'#'出现在第1行上。因此-Wtraditional 警告传统C理解但忽略的指令，因为' #'不会显示为该行的第一个字符。它也建议你#pragma通过缩进来隐藏传统C无法理解的指令。一些传统的实现不承认#elif，所以这个选项建议完全避免它。
类似于函数的宏，不带参数出现。
一元加运算符。
'' ü '整数常量后缀，或' F ' 要么 ' 大号'浮点常量后缀。(传统的C确实支持'大号 '后缀在整型常量上。)注意，这些后缀出现在大多数现代

系统的系统头文件中定义的宏中，例如'_Min '/' _MAX'宏' <limits.h>。在用户代码中使用这些宏通常可能会导致虚假警告，但是GCC的集成预处理器有足够的上下文来避免在这些情况下发出警告。
一个块在外部声明的函数，然后在块结束后使用。
一个switch语句有一个类型的操作数long。
一个非static函数声明static。这个构造不被一些传统的C编译器接受。
整型常量的ISO类型与传统类型具有不同的宽度或符号。如果该常数的基数为十，则仅发出该警告。即，通常表示位模式的十六进制或八进制值没有被警告。
检测到ISO字符串串联的用法。
自动聚合的初始化。
标识符与标签冲突。传统C缺乏标签的单独名称空间。
初始化工会。如果初始值设定为0，则省略警告。这是在假设用户代码中的零初始化符出现在例如__STDC__以避免丢失初始化器警告并且在传统C情况下依靠默认初始化为零的假设下完成的。
原型在固定/浮点值之间进行转换，反之亦然。与传统C编译时缺少这些原型会导致严重问题。这是可能的转换警告的一个子集; 为全套使用-Wtraditional转换 。
使用ISO C风格函数定义。这个警告是有意不发出原型声明或可变参数的功能，因为这些ISO C功能使用libiberty传统C兼容性宏时出现在你的代码，PARAMS和VPARAMS。对于嵌套函数，此警告也被绕过，因为该功能已经是GCC扩展，因此与传统的C兼容性无关。
-Wtraditional-conversion (C and Objective-C only)
如果原型导致类型转换与在没有原型的情况下发生相同的参数不同，则发出警告。这包括固定点到浮动的转换，反之亦然，转换会改变定点参数的宽度或符号，除非与默认提升相同。

-Wdeclaration-after-statement (C and Objective-C only)
在块中的语句之后发现声明时发出警告。这个从C ++中知道的构造是用ISO C99引入的，默认情况下允许在GCC中使用。它不受ISO C90的支持。见混合声明。

-Wshadow
每当局部变量或类型声明影响另一个变量，参数，类型，类成员（以C ++）或实例变量（在Objective-C中）或每当内置函数被映射时都会报警。请注意，在C ++中，编译器会警告局部变量是否会影响显式的typedef，但是如果影响struct / class / enum则不会。与...一样-Wshadow =全球 。

-Wno-shadow-ivar (Objective-C only)
只要本地变量在Objective-C方法中隐藏实例变量，就不要警告。

-Wshadow=global
默认为 -Wshadow。警告任何（全局）阴影。

-Wshadow=local
警告局部变量会影响另一个局部变量或参数。此警告由启用-Wshadow =全球 。

-Wshadow=compatible-local
当局部变量隐藏另一个局部变量或类型与阴影变量兼容的参数时发出警告。在C ++中，在这里键入兼容性意味着可以将阴影变量的类型转换为阴影变量的类型。这个标志的创建（除了-Wshadow =本地 ）基于这样的想法：当局部变量隐藏另一个不兼容类型时，它最有可能是故意的，而不是错误或拼写错误，如以下示例所示：

```
 for(SomeIterator i = SomeObj.begin(); i!= SomeObj.end(); ++ i){for(int i = 0; i <N; ++ i){...}...}
```

由于上述示例中的两个变量具有不兼容的类型，因此仅启用-Wshadow =兼容本地不会发出警告。因为它们的类型是不兼容的，所以如果程序员意外地使用一个来代替另一个，那么类型检查会捕获它并发出错误或警告。所以在这种情况下不要警告（关于影子）不会导致未被发现的错误。使用这个标志而不是-Wshadow =本地 可能会减少故意投影引发的警告数量。

此警告由启用 -Wshadow =本地 。

-Wlarger-than= len
每当定义大于len字节的对象时发出警告。

-Wframe-larger-than= len
如果函数帧的大小大于len字节，则发出警告。完成确定堆栈帧大小的计算是近似的而不是保守的。即使您没有收到警告，实际需求可能会比len略高。另外，alloca在确定是否发出警告时，编译器不包括通过可变长度数组或相关结构分配的任何空间。

-Wno-free-nonheap-object
尝试释放未在堆中分配的对象时，不要警告。

-Wstack-usage= len
警告函数的堆栈使用情况可能大于len字节。确定堆栈使用情况的计算是保守的。alloca在确定是否发出警告时，由编译器包含通过，可变长度数组或相关结构分配的任何空间。

这个消息符合输出 -fstack使用率 。

如果堆栈使用情况完全是静态的，但超过了指定的数量，则是：
警告：堆栈使用量为1120字节(warning: stack usage is 1120 bytes)
如果堆栈使用情况（部分）是动态的但有界限的，则是：
警告：堆栈使用可能是1648字节(warning: stack usage might be 1648 bytes)
如果堆栈使用情况（部分）是动态的并且不受限制，那么：
警告：堆栈使用可能是无限的(warning: stack usage might be unbounded)
-Wunsafe-loop-optimizations
警告如果循环无法优化，因为编译器无法假设循环索引的范围内的任何内容。同-funsafe循环的优化 警告如果编译器做出这样的假设。

-Wno-pedantic-ms-format (MinGW targets only)
与...结合使用时 -Wformat 和 -pedantic无GNU扩展，此选项禁用约非ISO的警告printf/ scanf格式宽度说明I32，I64以及I在Windows目标，其中依赖于MS运行时使用。

-Waligned-new
警告需要更大对齐类型的新表达式，alignof(std::max_align_t)但使用不带明确对齐参数的分配函数。该选项由启用-Wall 。

通常这只会警告全局分配函数，但是 -Waligned全新=所有 也警告类成员分配功能。

-Wplacement-new
-Wplacement-new= n
警告使用未定义行为放置新表达式，例如在小于对象类型的缓冲区中构造对象。例如，下面的放置新表达式被诊断，因为它试图在只有64个字节大的缓冲区中构造64个整数的数组。

```
 char buf [64];
new(buf)int [64];
```

该警告默认启用。

-Wplacement-new=1
这是默认的警告级别 -Wplacement新。在这个级别上，对于一些严格未定义的结构，GCC允许将其作为与旧代码兼容的扩展来发布警告。例如，new即使根据C ++标准具有未定义的行为，也会在该级别诊断下列表达式，因为它会写入超过一个元素数组的末尾。

```
 struct S {int n.a [1];  };
S * s =(S *)malloc(sizeof * s + 31 * sizeof s-> a [0]);
new(s-> a)int [32]();
```

-Wplacement-new=2
在这个级别上，除了诊断与级别1相同的所有构造外，还会发布一个诊断信息，用于放置新构造一个对象的最终构造，该构造的最后一个构件的类型是单个元素的数组，并且其大小较小比正在构建的对象的大小。虽然前面的示例将被诊断，但以下构造使用灵活的成员数组扩展来避免第2级的警告。

```
 struct S {int n.a [];  };
S * s =(S *)malloc(sizeof * s + 32 * sizeof s-> a [0]);
```

```
new(s-> a)int [32]();
```

-Wpointer-arith
警告任何取决于“功能类型”或“功能类型”的大小void。GNU C将这些类型的大小指定为1，以方便计算void *指针和指向函数的指针。在C ++中，当算术运算涉及时也会发出警告NULL。此警告也由启用-Wpedantic 。

-Wpointer-compare
如果指针与零字符常量进行比较，则发出警告。这通常意味着指针被解除引用。例如：

```
const char *p = foo ();
if (p == '\0')return 42;
```

请注意，上面的代码在C ++ 11中无效。

该警告默认启用。

-Wtype-limits
如果由于数据类型范围有限而导致比较始终为真或始终为false，但不警告常量表达式。例如，警告如果将一个无符号变量与<或与0进行比较>=。此警告也由启用-Wextra 。

-Wcomment
-Wcomments
每当评论开始序列' / * '出现在' / * '评论，或者每当一个反斜线换行符出现在' //'评论。此警告由启用-Wall 。

-Wtrigraphs
如果遇到可能会改变程序含义的三字母警告，则发出警告。评论中的三字代码不会被警告，除了那些会形成转义换行符的代码。

这个选项是隐含的 -Wall。如果-Wall没有给出，除非启用三字母，否则该选项仍然有效。要获得没有警告的三字母转换，但要获得另一个-Wall 警告，使用' -trigraphs -Wall -Wno-trigraphs "。

-Wundef
警告如果在#if指令中评估未定义的标识符。这些标识符被替换为零。

-Wexpansion-to-defined
每当' 定义 '在扩展宏时遇到（包括宏被扩展为' ＃如果'指令）。这种用法是不便携的。此警告也由启用-Wpedantic 和 -Wextra 。

-Wunused-macros
警告主文件中定义的未使用的宏。如果扩展或测试存在至少一次，则使用宏。预处理器还会警告，如果宏在重新定义或未定义时尚未使用。

内建的宏，命令行中定义的宏和包含文件中定义的宏不会被警告。

注意：如果实际使用宏，但仅用于跳过的条件块，则预处理器将其报告为未使用。为了避免在这种情况下发出警告，可以通过例如将其移动到第一个跳过的块中来改善宏定义的范围。或者，您可以提供一个类似以下内容的虚拟用途：

```
 #if defined the_macro_causing_the_warning#endif
```

-Wno-endif-labels
不论何时#else或#endif之后都有文字，不要警告。这有时会发生在具有表单代码的较早程序中

```
 #if FOO...
#else FOO...
#endif FOO
```

第二和第三位FOO应该在评论中。此警告默认开启。

-Wbad-function-cast (C and Objective-C only)
当函数调用转换为不匹配类型时发出警告。例如，警告如果对返回整数类型的函数的调用转换为指针类型。

-Wc90-c99-compat (C and Objective-C only)
警告ISO C90中不存在的功能，但存在于ISO C99中。例如，警告使用可变长度数组，long long类型，bool类型，复合文字，指定初始值设定项等等。该选项独立于标准模式。在下面的表达式中警告被禁用__extension__。

-Wc99-c11-compat (C and Objective-C only)
警告ISO C99中不存在的功能，但存在于ISO C11中。例如，警告使用匿名结构和联合，_Atomic类型限定符，_Thread_local存储类说明符，_Alignas说明符，Alignof运算符，_Generic关键字等。该选项独立于标准模式。在下面的表达式中警告被禁用__extension__。

-Wc++-compat (C and Objective-C only)
警告关于ISO C和ISO C ++的公共子集之外的ISO C构造，例如请求将隐式转换void *为非void类型指针。

-Wc++11-compat (C++ and Objective-C++ only)
警告ISO C ++ 1998和ISO C ++ 2011之间含义不同的C ++结构，例如ISO C ++ 1998中作为ISO C ++ 2011关键字的标识符。此警告将打开 -Wnarrowing 并被启用 -Wall 。

-Wc++14-compat (C++ and Objective-C++ only)
警告ISO C ++ 2011与ISO C ++ 2014之间含义不同的C ++结构。此警告由。启用 -Wall 。

-Wc++17-compat (C++ and Objective-C++ only)
警告ISO C ++ 2014和ISO C ++ 2017之间含义不同的C ++结构。此警告由。启用 -Wall 。

-Wcast-qual
每当指针被强制转换时都会发出警告，以便从目标类型中删除类型限定符。例如，警告如果a const char *被投给普通人char *。

在进行以非安全方式引入类型限定符的演员时也会发出警告。例如，铸造char **到const char **是不安全的，因为在这个例子：

```
  /* p is char ** value.  */const char **q = (const char **) p;/* Assignment of readonly string to const char * is O
```

-Wcast-align
每当指针被施放时发出警告，以便增加目标所需的对齐。例如，警告如果将char *为整数只能在两个或四个字节边界处访问的机器上的int * 。

-Wcast-align=strict
每当指针被施放时发出警告，以便增加目标所需的对齐。例如，警告如果a char *投射到某个int *目标机器。

-Wcast-function-type
当函数指针转换为不兼容的函数指针时发出警告。在涉及具有可变参数列表的函数类型的转换中，仅考虑所提供的初始参数的类型。指针类型的任何参数都与任何其他指针类型匹配。在整型任何良性的差异被忽略，就像int主场迎战long上ILP32目标。同样，类型限定符也被忽略。函数类型void (*) (void)是特殊的，并匹配所有内容，可用于抑制此警告。在涉及指向成员类型的指针的转换中，只要类型转换将指针更改为成员类型，此警告就会发出警告。此警告由启用-Wextra 。

-Wwrite-strings
编译C时，给字符串常量类型，以便将一个地址复制到非指针中会产生警告。这些警告帮助您在编译时查找可以尝试写入字符串常量的代码，但前提是您在声明和原型中使用时一直非常小心。否则，这只是一个滋扰。这就是我们没有制造的原因const char[length]const char *const-Wall 请求这些警告。

编译C ++时，警告关于从字符串文字到不再使用的转换char *。C ++程序默认启用此警告。

-Wcatch-value

-Wcatch-value= n (C++ and Objective-C++ only)
警告关于捕捉处理程序，不通过参考捕获。同-Wcatch值= 1（要么 -Wcatch价值简称）警告关于被值捕获的多态类类型。同-Wcatch值= 2警告所有类型的价值。同-Wcatch值= 3 警告所有没有被引用捕获的类型。-Wcatch价值 由...启用 -Wall 。

-Wclobbered
警告可能由longjmp或更改的变量vfork。此警告也由启用-Wextra 。

-Wconditionally-supported (C++ and Objective-C++ only)
警告有条件支持的（C ++ 11 [intro.defs]）结构。

-Wconversion
警告可能会改变值的隐式转换。这包括实数和整数，等之间转换abs (x)时x是double; 有符号和无符号之间的转换，如unsigned ui = -1; 并转换为较小的类型，例如sqrtf (M_PI)。不要警告像abs ((int) x)和的明确转换ui = (unsigned) -1，或者如果值没有像转换那样改变abs (2.0)。有关已签名和未签名整数之间转换的警告可以通过使用禁用-Wno-符号转换 。

对于C ++，也会警告混淆用户定义转换的重载解析; 以及从不使用类型转换运算符的转换：转换为void，相同类型，基类或对它们的引用。在C ++中，有关已签名和未签名整数之间转换的警告在默认情况下是禁用的，除非-Wsign转换 被明确启用。

-Wno-conversion-null (C++ and Objective-C++ only)
不要警告NULL非指针类型之间的转换。-Wconversion空 是默认启用的。

-Wzero-as-null-pointer-constant (C++ and Objective-C++ only)
警告当字面' 0'用作空指针常量。这对于促进nullptr在C ++ 11中的转换很有用。

-Wsubobject-linkage (C++ and Objective-C++ only)
如果类类型具有基类或字段类型使用匿名命名空间或依赖于没有链接的类型，则发出警告。如果类型A依赖于没有或内部链接的类型B，则将其定义为多个翻译单元将会违反ODR，因为每个翻译单元中B的含义不同。如果A只出现在单个翻译单元中，则最好的方法是将其置于匿名命名空间中以使其与内部关联。编译器不会针对主要.C文件中定义的类型给出此警告，因为这些警告不可能有多个定义。-Wsubobject联动 是默认启用的。

-Wdangling-else
警告可能会混淆分支所属的if声明的构造else。以下是这种情况的一个例子：

```
{if (a)if (b)foo ();elsebar ();
}
```

在C / C ++中，每个else分支都属于最内层的可能if语句，在本例中为if (b)。这通常不是程序员所期望的，正如上例中程序员选择的缩进所示。当可能出现这种混淆时，GCC会在指定此标志时发出警告。为了消除这个警告，在最里面的if语句周围添加显式大括号，这样就else不可能属于这个封闭if。生成的代码如下所示：

```
{if (a){if (b)foo ();elsebar ();}
}
```

此警告由启用 -Wparentheses 。

-Wdate-time
当遇到宏或遇到警告时__TIME__，可能会阻止按位重复编译。__DATE____TIMESTAMP__

-Wdelete-incomplete (C++ and Objective-C++ only)
Warn when deleting a pointer to incomplete type, which may cause undefined behavior at runtime. This warning is enabled by default.

-Wuseless-cast (C++ and Objective-C++ only)
Warn when an expression is casted to its own type.

-Wempty-body
Warn if an empty body occurs in an if , else or do while statement. This warning is also enabled by -Wextra .

-Wenum-compare
Warn about a comparison between values of different enumerated types. In C++ enumerated type mismatches in conditional expressions are also diagnosed and the warning is enabled by default. In C this warning is enabled by -Wall .

-Wextra-semi (C++, Objective-C++ only)
Warn about redundant semicolon after in-class function definition.

-Wjump-misses-init (C, Objective-C only)
如果goto语句或switch语句在变量的初始化过程中向前跳转，或者在变量初始化后向后跳转到标签，则发出警告。这只会对在声明时初始化的变量发出警告。此警告仅支持C和Objective-C; 在C ++中，这种分支在任何情况下都是错误的。

-Wjump门柱-INIT 包含在内 -Wc ++ - COMPAT。它可以被禁用-Wno跳门柱-INIT 选项。

-Wsign-compare
当有符号值和无符号值之间的比较可能会在有符号值转换为无符号时产生错误结果时发出警告。在C ++中，这个警告也是由-Wall。在C中，它也被启用-Wextra 。

-Wsign-conversion
警告可能会改变整数值符号的隐式转换，如将有符号整数表达式分配给无符号整数变量。明确的演员沉默警告。在C中，这个选项也可以通过-Wconversion 。

-Wfloat-conversion
警告隐式转换会降低实际值的精度。这包括从实数到整数的转换，以及从更高精度实数到更低精度实数值的转换。该选项也可以通过-Wconversion 。

-Wno-scalar-storage-order
不要警告涉及反向标量存储顺序的可疑构造。

-Wsized-deallocation (C++ and Objective-C++ only)
警告未定义的释放函数的定义

```
 void operator delete(void *)noexcept;
void operator delete [](void *)noexcept;
```

而没有定义相应大小的释放函数

```
 void operator delete(void *, std :: size_t)noexcept;
void operator delete [](void *, std :: size_t)noexcept;
```

或相反亦然。启用-Wextra 随着 -fsized，释放 。

-Wsizeof-pointer-div
警告两个sizeof表达式的可疑分割，这两个sizeof表达式将指针大小除以元素大小，这是计算数组大小的常用方法，但不能正确使用指针。此警告警告，例如关于sizeof (ptr) / sizeof (ptr[0])如果ptr不是数组，而是指针。此警告由启用-Wall 。

-Wsizeof-pointer-memaccess
如果参数使用，则向某些字符串和内存内置函数警告可疑长度参数sizeof。例如，这个警告触发了memset (ptr, 0, sizeof (ptr));if ptr不是一个数组，而是一个指针，并且提出了一个可能的修正或关于memcpy (&foo, ptr, sizeof (&foo)); 。-Wsizeof指针-memaccess还会警告有关字符串复制函数的调用，strncat或者strncpy指定为sizeof源数组的表达式。例如，在以下函数中，调用strncat将源字符串的大小指定为边界。这几乎肯定是一个错误，因此呼叫被诊断。

```
 void make_file(const char * name){char path [PATH_MAX];strncpy(path, name, sizeof path - 1);strncat(path, ".text", siz
```

该 -Wsizeof指针-memaccess 选项由启用 -Wall 。

-Wsizeof-array-argument
当sizeof运算符应用于在函数定义中声明为数组的参数时发出警告。C和C ++程序默认启用此警告。

-Wmemset-elt-size
memset如果第一个参数引用一个数组，并且第三个参数是一个等于元素数的数字，但不等于内存中数组的大小，则警告对内置函数的可疑调用。这表明用户已经省略了元素大小的乘法。此警告由启用-Wall 。

-Wmemset-transposed-args
memset如果第二个参数不为零且第三个参数为零，则警告对内置函数的可疑调用。这警告了例如memset (buf, sizeof buf, 0)最可能的memset (buf, 0, sizeof buf)意思。只有第三个参数为零时才会发出诊断信息。如果它的某个表达式被折叠为零，对于某种类型的零等等，则用户错误地交换参数并且不发出警告的可能性就小得多。此警告由启用-Wall 。

-Waddress
警告内存地址的可疑使用。这些包括在一个条件表达式中使用一个函数的地址，比如void func(void); if (func)，和一个字符串文字的内存地址比较，比如if (x == "abc")。这种用法通常表示程序员错误：函数的地址总是计算为真，因此它们在条件中的使用通常表示程序员在函数调用中忘记了括号；并且与字符串文字的比较导致未指定的行为并且在C中不可移植，所以它们通常表示程序员打算使用strcmp。此警告由启用-Wall 。

-Wlogical-op
在表达式中警告逻辑运算符的可疑用法。这包括在可能期望按位运算符的上下文中使用逻辑运算符。此外还警告逻辑运算符的操作数是否相同：

```
extern int a;
if (a < 0 && a < 0) { … }
```

-Wlogical-not-parentheses
警告关于逻辑不用在比较的左侧操作数上。如果右操作数被认为是布尔表达式，该选项不会警告。其目的是检测如下的可疑代码：

```
int a;
…
if (!a > 1) { … }
```

可以通过将LHS包括在括号中来抑制警告：

```
if ((!a) > 1) { … }
```

此警告由启用 -Wall 。

-Waggregate-return
警告是否定义或调用了返回结构或联合的任何函数。（在可以返回数组的语言中，这也会引发警告。）

-Wno-aggressive-loop-optimizations
如果在具有恒定迭代次数的循环中发出警告，编译器在一次或多次迭代过程中检测到某些语句中的未定义行为。

-Wno-attributes
不要警告是否使用了意外__attribute__情况，例如无法识别的属性，应用于变量的功能属性等。这不会阻止错误使用受支持属性的错误。

-Wno-builtin-declaration-mismatch
如果使用错误的签名或非功能声明了内置函数，则发出警告。该警告默认启用。

-Wno-builtin-macro-redefined
如果某些内置宏被重新定义，请不要警告。这抑制了警告的重新定义__TIMESTAMP__，__TIME__，__DATE__，__FILE__，和__BASE_FILE__。

-Wstrict-prototypes (C and Objective-C only)
警告如果一个函数被声明或定义而没有指定参数类型。（如果前面有指定参数类型的声明，则允许旧式函数定义不带警告。）

-Wold-style-declaration (C and Objective-C only)
根据C标准，在声明中警告过时的用法。例如，警告存储类说明符static不是声明中的第一件事情。此警告也由启用-Wextra 。

-Wold-style-definition (C and Objective-C only)
如果使用旧式函数定义，则发出警告。即使存在以前的原型，也会发出警告。

-Wmissing-parameter-type (C and Objective-C only)
在K&R样式函数中声明函数参数时没有类型说明符：

```
 void foo(bar){}
```

此警告也由启用 -Wextra 。

-Wmissing-prototypes (C and Objective-C only)
警告如果没有先前的原型声明定义全局函数。即使定义本身提供了原型，也会发出此警告。使用此选项可检测头文件中没有匹配原型声明的全局函数。此选项对C ++无效，因为所有函数声明都提供了原型，而非匹配声明声明了重载而不是与先前的声明冲突。使用-Wmissing申述 在C ++中检测缺少的声明。

-Wmissing-declarations
警告如果没有先前的声明定义全局函数。即使定义本身提供了原型，也是如此。使用此选项可检测未在头文件中声明的全局函数。在C中，对于以前的非原型声明的函数不会发出警告；使用-Wmissing的原型检测丢失的原型。在C ++中，不会为函数模板，内联函数或匿名命名空间中的函数发出警告。

-Wmissing-field-initializers
如果结构的初始化程序缺少某些字段，则发出警告。例如，下面的代码会导致这样的警告，因为xh它隐含地为零：

```
 struct s {int f, g, h;  };
struct sx = {3, 4};
```

此选项不会警告指定的初始化程序，因此以下修改不会触发警告：

```
 struct s {int f, g, h;  };
struct sx = {.f = 3, .g = 4};
```

在C中，这个选项不会警告通用零初始化器' {0} '：

```
 struct s {int f, g, h;  };
struct sx = {0};
```

同样，在C ++中，此选项不会警告空{}初始化程序，例如：

```
 struct s {int f, g, h;  };
sx = {};
```

此警告包含在中 -Wextra。获得其他-Wextra 没有这个警告，使用 -Wextra -Wno-missing-field-initializers 。

-Wno-multichar
不要警告如果多字符常量（' 'FOOF''）用来。通常，它们表示用户代码中存在拼写错误，因为它们具有实现定义的值，不应将其用于可移植代码中。

-Wnormalized= [none | id | nfc | nfkc]
在ISO C和ISO C ++中，如果两个标识符是不同的字符序列，则它们是不同的。但是，有时使用基本ASCII字符集之外的字符时，可以有两个看起来相同的不同字符序列。为避免混淆，ISO 10646标准规定了一些规范化规则，在应用时确保两个看起来相同的序列变成相同

的序列。如果您使用尚未正常化的标识符，GCC可以警告您；该选项控制该警告。

GCC支持四级警告。默认是-Wnormalized = NFC，它警告任何非ISO 10646“C”规范化形式的标识符NFC。NFC是大多数用途的推荐形式。这相当于-Wnormalized 。

不幸的是，ISO C和ISO C ++在标识符中允许有一些字符在变成NFC时不允许用于标识符。也就是说，无法在便携式ISO C或C ++中使用这些符号，并且在NFC中使用所有标识符。-Wnormalized = ID抑制这些字符的警告。希望未来版本的标准能够纠正这个问题，这就是为什么这个选项不是默认选项。

您可以通过书写来关闭所有角色的警告 -Wnormalized =无 要么 -Wno标准化。如果您使用其他规范化方案（如“D”），则只应执行此操作，否则您可以轻松创建几乎不可能看到的错误。

ISO 10646中的一些字符具有不同的含义，但在某些字体或显示方法中看起来完全相同，尤其是在应用格式化之后。例如\u207F，“SUPERSCRIPT LATIN SMALL LETTER N”，就像一个n放在上标中的常规一样显示。ISO 10646定义了NFKC规范化方案，将所有这些转换为标准格式，如果您使用的代码不在NFKC中，GCC会发出警告-Wnormalized = nfkc。此警告与包含字母O的每个标识符的警告相当，因为它可能与数字0混淆，所以不是默认值，但如果编程环境无法修复以显示这些内容，则可能作为本地编码约定有用字符清晰。

-Wno-deprecated
不要警告使用已弃用的功能。请参阅弃用功能。

-Wno-deprecated-declarations
不要警告使用属性标记为弃用的函数（请参阅函数属性），变量（请参阅变量属性）和类型（请参阅类型属性）deprecated。

-Wno-overflow
不要警告常量表达式中的编译时溢出。

-Wno-odr
在链接时间优化期间警告一个定义规则违规。需要-flto-ODR型合并 启用。默认启用。

-Wopenmp-simd
如果向量化器成本模型覆盖用户设置的OpenMP simd指令，则发出警告。该-fsimd成本模型=无限 选项可以用来放宽成本模型。

-Woverride-init (C and Objective-C only)
如果在使用指定初始值设定项时覆盖没有副作用的初始化域，请警告（请参阅指定初始值设定项）。

此警告包含在中 -Wextra。获得其他-Wextra 没有这个警告，使用 -Wextra -Wno-override-init 。

-Woverride-init-side-effects (C and Objective-C only)
警告如果使用指定的初始值设定项时会覆盖具有副作用的初始化字段（请参阅指定的初始值设定项）。该警告默认启用。

-Wpacked
如果给定结构的打包属性，但打包的属性对结构的布局或大小没有影响，则警告。这样的结构可能会失去对齐的好处。例如，在这段代码中，变量fxin struct bar没有对齐，尽管struct bar它本身没有packed属性：

```
 struct foo {int x;char a, b, c, d;
} __attribute __((packed));
结构栏{char z;struct foo f;};
```

-Wpacked-bitfield-compat
GCC的4.1,4.2和4.3系列忽略了packed类型位字段上的属性char。这已经在GCC 4.4中得到了修复，但是这种改变会导致结构布局的差异。当GCC 4.4中这个字段的偏移已经改变时，GCC会通知你。例如，字段a和b结构之间不再有4位填充：

```
 struct foo{char a:4;char b:8;
} __attribute__((packed));
```

该警告默认启用。使用-Wno-填充位字段，COMPAT 禁用此警告。

-Wpacked-not-aligned (C, C++, Objective-C and Objective-C++ only)
如果在打包结构或联合中显式指定对齐的结构字段未对齐，则警告警告。例如，在此代码中将会发出警告struct S，如warning: alignment 1 of ‘struct S’ is less than 8：

```
 struct __attribute__((aligned(8)))S8 {char a [8];  };
struct __attribute__((packed))S {struct S8 s8;};
```

此警告由启用 -Wall 。

-Wpadded
警告如果填充包含在结构中，则要对齐结构的某个元素或对齐整个结构。有时候发生这种情况时，可以重新排列结构的字段以减少填充，从而使结构更小。

-Wredundant-decls
如果在同一范围内多次声明任何内容，即使在多重声明有效且不做任何更改的情况下也会发出警告。

-Wno-restrict
当由一个restrict-qualified参数引用的对象（或者在C ++中，一个__restrict-qualified参数）被另一个参数别名时，或者当这些对象之间的副本重叠时引发警告。例如，对strcpy下面函数的调用尝试通过用最后四个字符替换其初始字符来截断字符串。但是，由于呼叫将终止NUL写入a[4]，副本重叠并且呼叫被诊断。

```
 void foo(void){char a [] =“abcd1234”;strcpy(a, a + 4);...}
```

该 -Wrestrict 选项检测到一些简单重叠的情况，即使没有进行优化，但最适用于 -02以上。它包含在中-Wall 。

-Wnested-externs (C and Objective-C only)
警告如果extern在函数中遇到声明。

-Wno-inherited-variadic-ctor
当继承的基类具有C可变参数构造函数时，禁止使用C ++ 11继承构造函数的警告；警告默认打开，因为省略号不被继承。

-Winline
警告如果声明为内联的函数不能内联。即使使用此选项，编译器也不会警告内联函数在系统头文件中声明的函数失败。

编译器使用各种启发式来确定是否内联一个函数。例如，编译器考虑了内联函数的大小以及当前函数中已经完成的内联量。因此，源程序中看似微不足道的变化可能会导致警告-Winline 出现或消失。

-Wno-invalid-offsetof (C++ and Objective-C++ only)
禁止将offsetof宏应用于非POD类型的警告。根据2014年ISO C ++标准，应用于offsetof非标准布局类型是未定义的。然而，在现有的C ++实现中，offsetof通常会给出有意义的结果。此标志适用于意识到他们正在编写非易用代码并且故意选择忽略有关警告的用户。

offsetof在未来的C ++标准版本中，这些限制可能会放宽。

-Wint-in-bool-context
在可能使用布尔值的情况下警告使用整数值，例如条件表达式（?:）在布尔上下文中使用非布尔整型常量，例如if (a <= b ? 2 : 3)。或者在布尔上下文中左移有符号整数，如for (a = 0; 1 << a; a++);。同样，对于所有类型的乘法，无论数据类型如何。此警告由启用 -Wall 。

-Wno-int-to-pointer-cast
将强制转换的警告抑制为不同大小整数的指针类型。在C ++中，转换为较小大小的指针类型是错误的。温特对指针的投 是默认启用的。

-Wno-pointer-to-int-cast (C and Objective-C only)

禁止来自指针的强制转换为不同大小整数类型的警告。

-Winvalid-pch
警告如果在搜索路径中找到预编译头（请参阅预编译头），但无法使用。

-Wlong-long
警告如果使用long long类型。这是由任一启用-Wpedantic 要么 -Wtraditional在ISO C90和C ++ 98模式下。要禁止警告消息，请使用-Wno-长隆 。

-Wvariadic-macros
如果在ISO C90模式下使用可变宏，或者在ISO C99模式下使用GNU替代语法，则发出警告。这是由任一启用-Wpedantic 要么 -Wtraditional。要禁止警告消息，请使用-Wno-可变参数的宏 。

-Wvarargs
警告用于处理可变参数的宏的可疑用法va_start。这是默认设置。要禁止警告消息，请使用 -Wno-可变参数 。

-Wvector-operation-performance
如果矢量操作未通过体系结构的SIMD功能实现，则发出警告。主要用于性能调整。可以实现向量操作piecewise，这意味着标量操作在每个向量元素上执行; in parallel，这意味着向量操作是使用更宽类型的标量来实现的，这通常是更高的性能效率; 并且as a single scalar，这意味着矢量装配到一个标量类型。

-Wno-virtual-move-assign
使用非平凡的C ++ 11移动赋值运算符来抑制有关从虚拟基础继承的警告。这是很危险的，因为如果虚拟基地沿着多条路径可到达，它会多次移动，这可能意味着两个对象最终都处于移动状态。如果写入移动赋值运算符以避免从移出的对象移动，则可以禁用此警告。

-Wvla
如果代码中使用了可变长度的数组，则会发出警告。-Wno-VLA 防止 -Wpedantic 警告变长数组。

-Wvla-larger-than= n
如果使用此选项，编译器将警告使用可变长度数组，其大小要么是无限的，要么是大于n字节的参数。这与如何相似-Walloca-greater-than = n 工作，但与变长数组。

请注意，GCC可能会将已知值的小型可变长度数组优化为普通数组，因此可能无法触发此警告。

此警告未启用 -Wall ，并且只在有效时才有效 -ftree-VRP 是活动的（默认为 -02 以上）。

也可以看看 -Walloca-greater-than = n 。

-Wvolatile-register-var
如果寄存器变量被声明为volatile，则发出警告。volatile修饰符不会禁止可能会消除读取和/或写入寄存器变量的所有优化。此警告由启用-Wall 。

-Wdisabled-optimization
如果请求的优化传递被禁用，则发出警告。此警告通常并不表示您的代码有任何问题; 它只是表明GCC的优化器无法有效处理代码。通常，问题在于你的代码太大或太复杂; 当优化本身可能花费过多时间时，GCC拒绝优化程序。

-Wpointer-sign (C and Objective-C only)
警告指针参数传递或具有不同签名的分配。该选项仅支持C和Objective-C。它暗示着-Wall 并通过 -Wpedantic ，可以禁用 -Wno指针-SIGN 。

-Wstack-protector
该选项仅在激活时才有效 -fstack保护器活跃。它警告那些不能防止堆栈粉碎的功能。

-Woverlength-strings
警告字符串常量的长度超过C标准中指定的“最小最大长度”。现代编译器通常允许比标准的最小限制长得多的字符串常量，但是非常便携的程序应该避免使用更长的字符串。

该限制适用于字符串常量级联之后，并且不计算尾随NUL。在C90中，限制为509个字符; 在C99中，它被提升到了4095. C ++ 98没有规定最小最大规范，所以我们不用诊断C ++中的超长字符串。

这个选项是隐含的 -Wpedantic ，并可以禁用 -Wno-超长串 。

-Wunsuffixed-float-constants (C and Objective-C only)
对任何没有后缀的浮动常量发出警告。与...一起使用时-Wsystem报头它会在系统头文件中警告这些常量。在准备代码以使用FLOAT_CONST_DECIMAL64从C99到十进制浮点扩展的编译指示时，这非常有用。

-Wno-designated-init (C and Objective-C only)
使用位置初始化程序初始化已标记了designated_init属性的结构时，禁止警告。

-Whsa
当编译的函数或OpenMP构造不能释放HSAIL时发出警告。

相关文章

暂无图片

live555学习(一)编译live555

live555学习(一)通读Makefile编译live555 live555 编译live555 学习开源 live555学习(一)通读Makefile编译live555 前言live555简介下载live555live的编译 生成MakefileMake的生成目标前言 拿到一份开

阅读更多...

暂无图片

linuxC/C++面试问题总结整理

linuxC/C面试问题总结整理 因为一些原因重新找工作了，面的linux c/c，这里把面试中经常碰到的问题总结一下。linuxC/C面试问题总结整理单元测试关键字const关键字static关键字volatile

阅读更多...

暂无图片

linux环境编程-守护进程

linux编程-守护进程编写守护进程（Daemon）是运行在后台的一种特殊进程。它独立于控制终端并且周期性地执行某种任务或等待处理某些发生的事件。守护进程是一种很有用的进程。

阅读更多...

暂无图片

层级目录结构的Makefile递归编译方法

层级目录结构的Makefile递归编译方法.层级目录结构的Makefile编写方法.0.前言1.如何编译整个工程2.过滤每层不需要编译的目录3.将所有输出文件定向输出.0.前言 假如现在有这样一个目录结构:要怎

阅读更多...

GPS数据包格式及数据包解析

GPS数据包解析GPS数据包解析目的GPS数据类型及格式数据格式数据解释解析代码结构体定义GPRMC解析函数GPGGA解析函数测试样例输出gps数据包格式 gps数据解析 车联网目的 任务很简

阅读更多...

Linux读二进制串口数据异常,数据校验出错.

今天做USB的Gsensor程序发现读到的数据总是校验不过,无法进一步解析数据,而在Windows下通过工具读出来的数据均是正常的。于是做出了串口读上来的数据有加过工的可能，因为该Sensor

阅读更多...

修改Markdown表格宽度,去掉Markdown表格头加粗效果.

修改Markdown表格宽度,去掉Markdown表格头加粗效果. 最近整理函数的时候使用Markdown制作表格,但是发现表格的头行总是加粗的,看着很不愉快, 哎,没办法只好把描述放在头行忍忍, 但是导出来

阅读更多...

Linux信号量同步共享内存实验.

Linux信号量同步共享内存实验. Linux信号量同步共享内存实验. 简述程序流程信号量和共享内存的系统函数 信号量系统函数及接口共享内存系统函数及接口写程序读程序 简述 本文主要内容是自己对

阅读更多...

linux错误errno对照表

转自 : https://www.linuxidc.com/Linux/2013-09/89525.htm errno 在 <errno.h> 中定义，错误 Exx 的宏定义在 /usr/include/asm-generic 文件夹下面的 errno-base.h 和 errno.h，分别定义了

阅读更多...

linux通过c++实现线程池类

线程池的实现 线程池的实现前言线程池的概念使用原因及适用场合线程池的实现原理任务调度逻辑程序测试 前言 初学C,想封装点常用的C类,已经写好了mutex,cond,thread的类,想用起来写点东西,于

阅读更多...