# 类对象

## class Kconfig(object):

表示Kconfig配置，例如 对于x86或ARM。 这是配置中出现的符号，选项和菜单节点集。 创建任意数量的Kconfig对象（包括针对不同的体系结构）是安全的。 Kconfiglib不保留任何全局状态。

可以使用以下属性。 它们应该被视为只读，有些是通过@property magic实现的。

SYMS**：**包含配置中所有符号的字典，按名称编制索引。 除了常量（带引号）符号外，还包括表达式中引用但从未定义的所有符号。

Symbol.nodes为空时可以识别未定义的符号 - 请参阅模块docstring中的“菜单树简介”部分。

const\_syms：常量（引用）符号的'syms'字典

named\_choices：像'syms'这样的词典用于命名选择（选择FOO）

defined\_syms：包含所有已定义符号的列表，其顺序与Kconfig文件中显示的顺序相同。 在多个位置定义的符号会多次出现。

注意：您可能希望使用“unique\_defined\_syms”。 此属性主要是为了向后兼容而维护的。

unique\_defined\_syms：像“defined\_syms”这样的列表，但删除了重复项。 只为第一个实例保留在多个位置定义的符号。 否则将保留Kconfig订单。

使用此属性而不是'defined\_syms'可以节省工作量，并在编写配置输出时自动给出合理的行为（在多个位置定义的符号只生成一次输出，同时仍然保留Kconfig顺序以便于阅读）。

**choices选择**：包含所有选项的列表，其顺序与配置文件中显示的顺序相同。注意：您可能希望使用“unique\_choices”。 此属性主要是为了向后兼容而维护的。

unique\_choices：类似于'unique\_defined\_syms'，用于选择。 命名选项可以有多个定义位置。

**menus菜单**：包含所有菜单的列表，其顺序与它们在Kconfig文件中显示的顺序相同

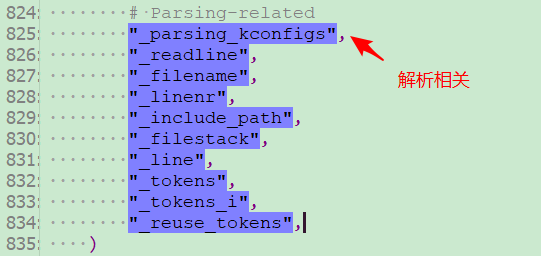
**comments**：包含所有注释的列表，其顺序与它们在Kconfig文件中显示的顺序相同

**kconfig\_filenames：**包含在配置中的所有Kconfig文件的文件名相对于$ srctree（或者如果未设置$ srctree则相对于当前目录），保留除绝对路径（例如'source "/foo/Kconfig")之外的列表原样。

这些文件按照源代码的顺序列出，从顶级Kconfig文件开始。 如果文件多次来源，则会多次出现。 使用set()获取唯一的文件名。

注意：将此用于增量构建是多余的。Kconfig.sync\_deps()

已间接捕获任何更改配置输出的文件修改。



### 接口函数 def \_\_init\_\_

通过解析Kconfig文件创建新的Kconfig对象。请注意，Kconfig文件与.config文件（存储配置符号值）不同。有关影响默认警告设置的一些环境变量（KCONFIG\_WARN\_UNDEF和KCONFIG\_WARN\_UNDEF\_ASSIGN），请参阅模块docstring。

引发语法错误的KconfigError，以及（可能是子类）IO错误上的IOError（'errno'，'strerror'和'filename'可用）。请注意，IOError可以在Python 3上作为OSError捕获。

**filename（默认值：“Kconfig”）：**要加载的Kconfig文件。对于Linux内核，您需要来自顶级目录的“Kconfig”，因为环境变量将确保从那里包含正确的Kconfig（写作时为arch / $ SRCARCH / Kconfig）。

如果设置了$ srctree，将相对于它查找'filename'。 $ srctree还用于在Kconfig文件中查找源文件。请参阅课程文档。

如果您通过'make scriptconfig'使用Kconfiglib，则基本Kconfig文件的文件名将在sys.argv [1]中。它在实践中目前总是“Kconfig”。

**警告（默认值：True）：**如果应该使用与此配置相关的警告，则为True。稍后可以使用config.enable / disable\_warnings()更改此设置。它作为构造函数参数提供，因为在解析期间可能会生成警告。

另请参阅其他Kconfig.enable\_\*\_warnings()函数，这些函数在启用警告时启用或禁止某些警告。所有生成的警告都会添加到Kconfig.warnings列表中。请参阅课程文档。

**warn\_to\_stderr（默认值：True）：**

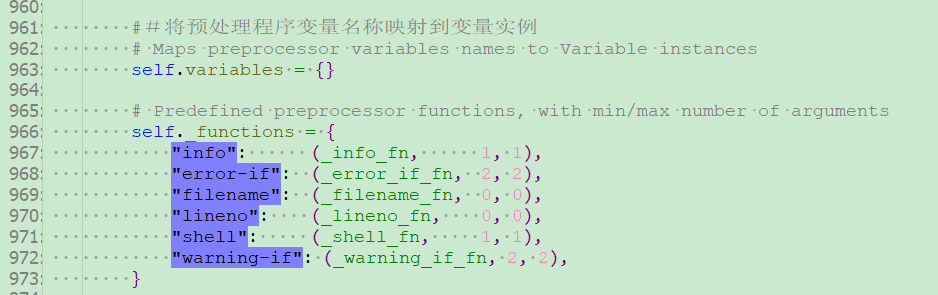
如果除了添加到Kconfig.warnings之外，还应将警告打印到stderr，则为True。

可以使用Kconfig.enable / disable\_stderr\_warnings()更改此设置。

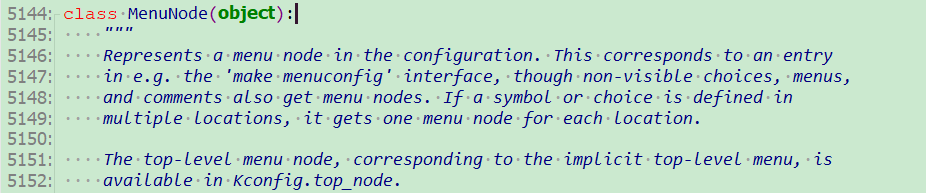
**encoding（默认值：“utf-8”）**：读取和写入文件时使用的编码。如果为None，将使用当前语言环境中指定的编码。

“utf-8”默认值避免了配置为使用C语言环境的系统上的异常，这意味着ASCII编码。

由于实现问题（常规字符串转换为Unicode字符串，这在Python 2中是不同的），此参数对Python 2没有影响。无论如何，Python 2不会解码常规字符串。



## class MenuNode(object)



表示配置中的菜单节点。这对应于例如'make menuconfig'界面，虽然不可见的选项，菜单和注释也获得菜单节点。如果在多个位置定义了符号或选项，则每个位置都会获得一个菜单节点。

顶级菜单节点（对应于隐式顶级菜单）在Kconfig.top\_node中可用。

可以在Symbol / Choice.nodes属性中找到Symbol或Choice的菜单节点。菜单和评论表示为简单

菜单节点，其文本存储在prompt属性（prompt [0]）中。这反映了C实现。MenuNode实例上提供以下属性。它们应被视为只读。

item:项目：

符号，选项或常量菜单和注释之一。菜单和注释表示为普通菜单节点。 Ifs已折叠（与C实现匹配）并且未显示在最终菜单树中。

**next:**

以下菜单节点。如果没有以下节点，则为None。

**列表：**

第一个子菜单节点。如果没有孩子，则没有。

选择和菜单自然会有子项，但由于从依赖项自动创建的菜单，符号也可以有子项（请参阅kconfig-language.txt）。

**parent**:

父菜单节点。如果没有父母，则无。

prompt:提示：

一个（string，cond）元组，带有菜单节点及其条件表达式的提示（如果没有条件，则为self.kconfig.y）。如果没有提示，则无。

对于符号和选项，提示存储在MenuNode中，而不是Symbol或Choice实例中。对于菜单和注释，提示保存文本。

**defaults:**此特定菜单节点的“默认”属性。请参阅symbol.defaults。

在评估默认值时，您应该使用Symbol / Choice.defaults，因为它包含所有菜单节点的属性（符号/选项可以有多个定义位置/菜单节点）。 MenuNode.defaults用于生成文档。

selects：像MenuNode.defaults一样，用于选择。

implies：像MenuNode.defaults一样，暗示。

ranges：与MenuNode.defaults一样，用于范围。

help:：符号和选项菜单节点的帮助文本。如果没有帮助文本，则无。始终存储在节点中而不是符号或选择中。

如果在多个位置定义符号，则可以在每个位置具有单独的帮助文本。

从帮助文本中删除尾随空格（包括最终换行符）。在Kconfiglib 10.21.0之前的情况并非如此，其中格式未记录。

**DEP：**

“依赖于”菜单节点的依赖关系，如果没有依赖关系，则取决于self.kconfig.y。父依赖项将传播到此属性，然后此属性又传播到符号和选择的属性。

如果在多个位置定义了符号或选项，则只有在特定位置定义的属性才会获得传播给它们的相应MenuNode.dep依赖项。

visibility：

能见度菜单节点（必须代表菜单）的'visible if'依赖项，如果没有'visible if'依赖项，则为self.kconfig.y。'可见，如果'依赖关系递归地传播到菜单中的符号和选项的提示。

referenced:引用一个set()，其中包含菜单节点的属性和属性条件中引用的所有符号和选项。

还包括从周围菜单继承的依赖项以及if。选项出现在选择符号的依赖项中。

is\_menuconfig:

如果菜单节点的子项应显示在单独的菜单中，则设置为True。这是以下项目的情况：

- Menus (node.item == MENU)

- Choices

- Symbols defined with the 'menuconfig' keyword. - 使用'menuconfig'关键字定义的符号。子项来自隐式创建的子菜单，应该显示在单独的菜单中而不是缩进。'is\_menuconfig'只是提示如何显示菜单节点。除了打印符号外，Kconfiglib在内部忽略它。

filename/linenr:

菜单节点出现的位置。文件名是相对于$ srctree（如果没有设置$ srctree，则是当前目录），除了传递给'source'的绝对路径和Kconfig .\_\_ init \_\_()被保留。

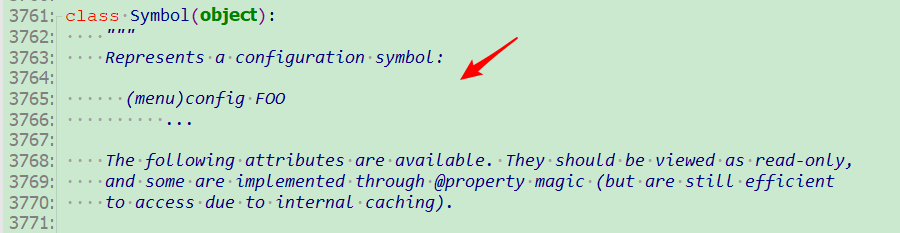
include\_path：

（filename，linenr）元组的元组，给出“源”语句的位置，通过该语句包含包含此菜单节点的Kconfig文件。 第一个元素是传递给Kconfig .\_\_ init \_\_()等的顶级Kconfig文件中'source'语句的位置。

请注意，菜单节点本身的Kconfig文件不包括在内。 检查'filename'和'linenr'。

  kconfig命令： 菜单节点来自的Kconfig实例。

## class Symbol(object)



表示配置符号：

可以使用以下属性。它们应该被视为只读，有些是通过@property magic实现的（但由于内部缓存仍然有效访问）。

注意：提示，帮助文本和位置存储在Symbol的MenuNode中，而不是存储在Symbol本身中。检查MenuNode类和Symbol.nodes属性。该组织与C工具匹配。

name：符号的名称，例如“FOO”代表'配置FOO'。

type：符号的类型。 BOOL，TRISTATE，STRING，INT，HEX，UNKNOWN之一。

UNKNOWN用于未定义的符号，（非特殊的）常量符号，以及没有类型定义的符号。

当没有模块运行时（MODULES的值为n），TRISTATE符号将类型神奇地改为BOOL。对于“y”模式中的选项内的符号也会发生这种情况。这与C工具相匹配，对于类似menuconfig的功能也很有意义。

orig\_type： Kconfig文件中给出的类型，没有应用任何魔法。在打印符号时使用。

str\_value：符号的值为字符串。给出string / int / hex符号的值。对于bool / tristate符号，给出“n”，“m”或“y”。

这是在关系表达式中使用的符号值（A = B，A！= B等）

问题：对于int / hex符号，必须经常保留值的确切格式（例如，在编写.config文件时），因此无法直接将其作为int获取。执行int（int\_sym.str\_value）或int（hex\_sym.str\_value，16）以获取整数值。

tri\_value：符号的三态值为整数。 0,1,2中的一个，代表n，m，y。对于非bool / tristate符号，始终为0（n）。

这是在关系表达式之外使用的符号值（A，！A，A && B，A || B）。

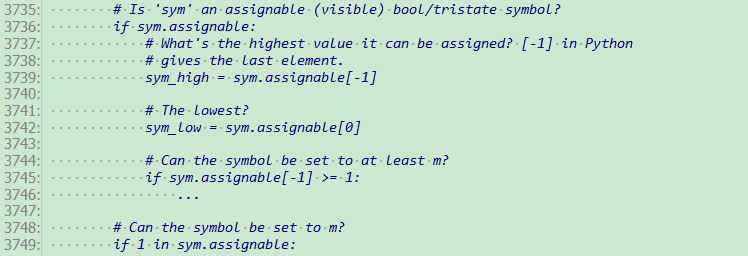
assignable:分配：

包含当前可以分配给符号（将被尊重）的三态用户值的元组，从最低（0，表示n）到最高（2，表示y）排序。这对应于menuconfig界面中可用的选项。可分配值的集合根据符号的可见性和选择/隐含来计算。

返回非bool / tristate符号的空集和具有可见性n的符号。其他可能的值是（0,2），（0,1,2），（1,2），（1，）和（2，）。 A（1，）或（2，）结果表示符号可见但通过选择“锁定”到m或y，可能与可见性结合。 menuconfig分别表示为-M- and -\*-。

对于string / hex / int符号，请检查Symbol.visibility是否为非0（非n），以确定是否可以更改该值。

一些方便的'可指定'成语：



**visibility:**能见度：符号的可见性。 0,1,2中的一个，代表n，m，y。有关符号值和可见性的概述，请参阅模块文档。

user\_value：

符号的用户值。如果没有分配用户值，则为None（通过Kconfig.load\_config()或Symbol.set\_value()）。

对于bool / tristate符号，保持0,1或2;对于其他符号类型，保留字符串。警告：不要直接分配给它。它会破坏事物。使用Symbol.set\_value()。

config\_string：由Kconfig.write\_config()为符号写出的.config赋值字符串。如果没有写出.config赋值，则返回空字符串。

通常，可见符号，具有（活动）默认值的符号和所选符号被写出。这包括所有非n值bool / tristate符号，以及所有可见的string / int / hex符号。

带有（不再需要）'选项env = ...'选项的符号不会生成配置输出，也不会生成特殊的'option defconfig\_list'符号。

提示：此字段在生成自定义配置输出时很有用，即使对于非.config类格式也是如此。要只写出要写入.config文件的符号，请执行以下操作：



这是write\_autoconf()写出的符号的超集。该函数跳过所有n值符号。

只编写所有符号通常不会有太大的危害，尽管你可能会得到一些特殊的符号，并且可能会有一些“冗余”的n值符号条目。

nodes:节点：

此符号的MenuNodes列表。对于大多数符号，将包含单个MenuNode。未定义和常量符号具有空节点列表。

在多个位置定义的符号为每个位置获取一个节点。

**choice：**

保留选择符号的父选项，非选择符号保留None。双打作为符号是否为选择符号的标志。

defaults：符号的'default'属性的（默认，cond）元组列表。例如，'默认A && B如果C || D'表示为（（AND，A，B），（OR，C，D））。如果没有给出条件，'cond'是self.kconfig.y。

请注意，“取决于”并且父依赖项会传播到“默认”条件。

**selects:选择：**

符号的“选择”属性的（符号，cond）元组列表。例如，'如果B && C'则选择A表示为（A，（AND，B，C））。如果没有给出条件，'cond'是self.kconfig.y。

请注意，“取决于”并且父依赖关系将传播到“选择”条件。

implies:暗示：

像'选择'一样，暗示。

ranges:范围：

符号的“范围”属性的（低，高，cond）元组列表。例如，'范围1 2，如果A'表示为（1,2，A）。如果没有条件，'cond'是self.config.y。

请注意，“取决于”并且父依赖关系会传播到“范围”条件。

问题：上面的1和2将表示为（未定义）符号而不是普通整数。未定义的符号将其名称作为其字符串值，因此可以解决这个问题。 C工具的工作方式相同。

rev\_dep：

选择此符号的其他符号的反向依赖关系表达式。

多个选项一起进行OR运算。 select上的条件与选择符号进行AND运算。

例如，如果A有'选择FOO'而B有'选择FOO如果C'，那么FOO的rev\_dep将是（OR，A，（AND，B，C））。

**weak\_rev\_dep：**

Like rev\_dep, for imply.

**direct\_dep：**

'依赖'依赖关系。如果在多个位置定义了符号，则每个位置的依赖关系将进行“或”运算。

在内部，这用于实现'imply'，仅当隐含符号具有expr\_value（self.direct\_dep）！= 0.'依赖于'并且父依赖关系自动传播到条件时才适用

 属性，所以通常检查直接依赖项是多余的。

**referenced：**

一个set()，其中包含符号的属性和属性条件中引用的所有符号和选项。还包括从周围菜单继承的依赖项以及if。选项出现在选择符号的依赖项中。

**env\_var：**

如果符号有'option env =“FOO”'选项，则它包含环境变量的名称（“FOO”）。没有'选项env'的符号没有。

'option env =“FOO”'就像一个'默认'属性，其值是$ FOO的值。

具有'option env'的符号永远不会写入.config文件，即使它们是可见的。 env\_var对应于C实现中名为SYMBOL\_AUTO的标志。

**is\_allnoconfig\_y：**

如果符号上设置了“option allnoconfig\_y”，则为True。这在内部没有任何影响（打印符号时除外），但可以通过脚本进行检查。

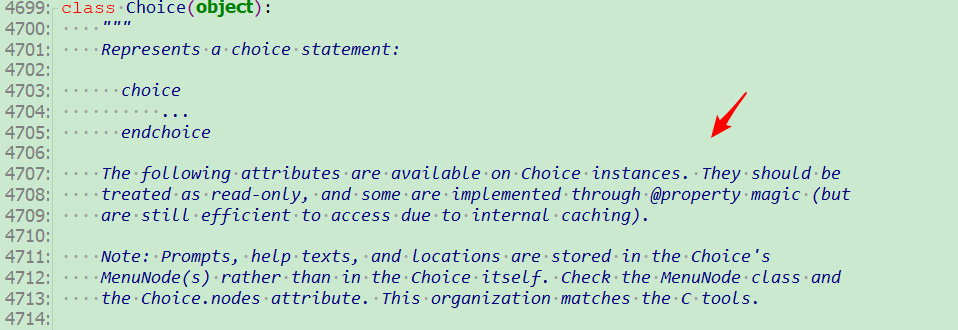
**is\_constant：**

如果符号是常量（带引号）符号，则为True。

**kconfig：**

此符号来自的Kconfig实例。

## class Choice(object):



表示选择语句：

Choice实例上提供以下属性。它们应该被视为只读，有些是通过@property magic实现的（但由于内部缓存仍然有效访问）。

注意：提示，帮助文本和位置存储在Choice的MenuNode中，而不是存储在Choice本身中。检查MenuNode类和Choice.nodes属性。该组织与C工具匹配。

**name：**选择的名称，例如“FOO”表示“选择FOO”，如果“选择”没有名称则为“无”。

**type：**选择的类型。 BOOL，TRISTATE，UNKNOWN之一。 UNKNOWN用于定义没有类型的选项，其中任何包含的符号都没有类型（否则选择继承用类型定义的第一个符号的类型）。

当没有模块运行时（CONFIG\_MODULES = n），TRISTATE选择神奇地将类型更改为BOOL。这与C工具相匹配，对于类似menuconfig的功能也很有意义。

orig\_type：Kconfig文件中给出的类型，没有应用任何魔法。在打印选项时使用。

tri\_value：选择的三态值（模式）。选择可以是以下三种模式之一：

**0（n）** - 禁用该选项，不能选择符号。对于可见选择，此模式仅适用于设置了“可选”标志的选项（请参阅kconfig-language.txt）。

**1（m）** - 任意数量的选择符号可以设置为m，其余的将是n。

**2（y）** - 一个符号为y，其余为n。

只有三态选择可以是m模式。选择的可见性是模式的上限，而模式又是选择符号可见性的上限。要更改模式，请使用Choice.set\_value()。

实施说明：

C工具在内部将选项表示为一种符号，在许多代码路径中都有特殊外壳。这就是为什么与Symbol有很多相似之处。选择的值（模式）实际上只是一个正常的符号值，隐式反向依赖关系强制其下限为m，用于可见的非可选选项（反向依赖是'm && <visibility>'）。

选项中的符号将选择传播为对其属性的依赖关系。这将选择的模式转换为例如上限。选择符号的可见性，并解释与模块docstring中提到的打印选择符号相关的问题。

Kconfiglib仅使用单独的Choice类，因为它使代码和接口不那么混乱（特别是在面向用户的界面中）。

为了一致性和兼容性，对应的属性在Symbol和Choice类中具有相同的名称。

assignable：请参阅符号类文档。给出可分配值（模式）。

visibility：请参阅Symbol类文档。对值（模式）采取行动。

selection：当前所选符号的Symbol实例。如果Choice不在y模式或没有选择符号（由于对选择符号的依赖性不满意），则为None。

警告：不要直接分配给它。它会破坏事物。在您要选择的选择符号上调用sym.set\_value（2）。

**user\_value：**

用户通过Choice.set\_value()选择的值（模式）。如果用户未选择模式，则为0,1或2或无。请参见Symbol.user\_value。

警告：不要直接分配给它。它会破坏事物。请改用Choice.set\_value()。

user\_selection：用户选择的符号（通过将其设置为y）。如果选择不在y模式中，则忽略，但仍然记住，如果模式更改回y，则选择“快照”回到用户选择。由于不满意的依赖性，这可能与“选择”不同。

警告：不要直接分配给它。它会破坏事物。在要选择的选择符号上调用sym.set\_value（2）。

**syms：**选项中包含的符号列表。

隐藏的问题：如果一个符号依赖于选择中的前一个符号，以便创建一个隐式菜单，它将不是一个选择符号，并且不会包含在'syms'中。

**nodes:节点：**

此选项的MenuNodes列表。实际上，列表可能总是包含一个MenuNode，但可以选择一个名称并在多个位置定义它。

defaults：选项'defaults'属性的（symbol，cond）元组列表。例如，'默认A，如果B && C'表示为（A，（AND，B，C））。如果没有条件，'cond'是self.config.y。

请注意，“取决于”并且父依赖项会传播到“默认”条件。

direct\_dep：请参阅Symbol.direct\_dep。

referenced：

一个set()，其中包含在所选属性和属性条件中引用的所有符号。还包括从周围菜单继承的依赖项以及if。

is\_optional：如果选择设置了“可选”标志并且可以处于n模式，则为真。

kconfig：此选项来自的Kconfig实例。

# 接口函数

load\_allconfig

函数：def load\_allconfig(kconf, filename):

帮助所有\*配置。 加载（合并）KCONFIG\_ALLCONFIG指定的配置文件（如果有）。 请参阅Linux内核中的Documentation / kbuild / kconfig.txt。

在调用期间（disable\_override\_warnings()+ disable\_redun\_warnings()）禁用配置文件中重复分配的警告，并在最后启用它们。 KCONFIG\_ALLCONFIG配置文件应覆盖符号。

如果设置了KCONFIG\_ALLCONFIG，则退出sys.exit()（引发SystemExit异常）并向stderr输出错误但是

  配置文件无法打开。

kconf：用于加载配置的Kconfig实例。

     filename：特定于命令的配置文件名 - “allyes.config”，“allno.config”等。

## 函数：Symbol.user\_value

从.config格式的文件加载符号值。相当于调用Symbol.set\_value()来设置每个值。

在.config文件中“＃CONFIG\_FOO未设置”将FOO的用户值设置为n。 C工具的工作方式相同。

对于每个符号，Symbol.user\_value属性保存在.config文件中分配符号的值（如果有）。如果存在不满足的依赖关系，则用户值可能与Symbol.str / tri\_value不同。

调用此函数还会更新Kconfig.missing\_syms属性，其中包含配置文件中未定义符号的所有分配列表。如果'replace'为True，则清除Kconfig.missing\_syms，否则附加。另请参阅Kconfig.missing\_syms的文档。

在IO错误上引发（可能是子类）IOError（'errno'，'strerror'和'filename'可用）。请注意，IOError可以在Python 3上作为OSError捕获。

filename（默认值：None）：从（字符串）加载配置的路径。如果设置则尊重$ srctree（参见类文档）。

如果'filename'为None（默认值），则自动计算要加载的配置文件（如果有），给出您通常需要的行为：

1.如果设置了KCONFIG\_CONFIG环境变量，它将提供要加载的配置文件的路径。否则，使用“.config”。请参阅standard\_config\_filename()。

2.如果（1.）的路径不存在，则加载kconf.defconfig\_filename给出的配置文件，该文件派生自'option defconfig\_list'符号。

3.如果（1.）和（2.）未能找到要加载的配置文件，则不加载配置文件，并且符号保留其当前值（例如，它们的默认值）。这不是错误。

另请参阅返回值。

replace（默认值：True）：如果为True，则在加载.config之前将清除所有现有用户值。传递False以合并配置。

verbose（默认值：True）：如果为True且文件名为None（自动推断配置文件），则会向stdout打印一条消息，告知哪个文件已加载（或者没有加载文件）。这是为了减少工具中的样板。

如果加载了现有配置（不是来自'option defconfig\_list'符号），则返回True，否则返回False。这与filename = None非常有用，否则将始终返回True。

