第41卷 第6A期 计算机科学 Vol. 41 No. 6A
2014年6月 Computer Science June 2014

Linux内核中编译选项、文件以及函数之间依赖关系的解析方法

江梦涛 潘朋飞 宋 杨 荆 琦
(北京大学软件与微电子学院 北京100871)

摘 要 Linux内核中的编译选项为内核提供了功能定制的可能性,但从传统静态代码分析的角度较难获得编译选项的改变对软件代码实际的影响。基于Linux内核源码,提出一种从编译选项开始,一直到函数调用关系的分析思路,同时给出了与编译选项相关的源代码的分层模型及其具体的分析做法,对于理解Linux内核源码特别是其中的编译依赖有现实意义。

关键词 操作系统,Linux内核,静态代码分析
中图法分类号 TP314 **文献标识码** A

Method of Parsing Dependencies between the Linux Kernel Compiling Options,Source Files and Functions

JIANG Meng-tao PAN Peng-fei SONG Yang JING Qi
(School of Software and Microelectronics,Peking University,Beijing 100871,China)

Abstract Compilation options in Linux Kernel provide the possibilities of functionality customization in operation system,but in traditional method of static code analysis,it is hard to know how a compilation option could affect the source code of Linux kernel. Based on the Linux kernel source,this paper presents a way of analyzing the relationship between compilation options,files and function calls from the top level,and gives the hierarchical model associated with the source code and compile options.

Keywords Operating system,Linux kernel,Static code analysis

1 概述

对大型软件进行静态代码分析时,往往需要分析各个模块、文件以及函数之间的调用关系,甚至跨层级之间的关系,比如模块与函数、模块与文件等等之间的关系。传统的静态代码分析工具一般只从代码层去分析函数或者变量之间的关系,然而对于Linux内核源码这种大型且使用编译选项来控制功能的大型软件来说,编译选项虽然为软件提供了配置的可能,但是对此种过于庞大的系统进行传统的静态代码分析缺少一种自上而下的线索,顶层的编译选项的选择与否对软件的影响难以通过对代码的直接分析来获得。比如某个驱动在编译菜单中只是一个选项,但其背后的实现调用了许多其他已经存在的功能,想要仅仅依靠直接代码分析梳理出其内部的调用关系是比较困难的,因为传统的方法缺乏对顶层编译选项的分析。

同时,编译选项与源文件之间的对应关系解释也是传统静态代码分析所欠缺的地方,这个对应关系实际上非常重要,它是顶层功能描述和实际物理源文件之间的对应,同时也是功能描述依赖的实际体现。通过这一层关系的分析才能最终落实到最底层的函数、变量调用关系的分析。

本文为了解决上述问题,基于Linux内核源码,提出一种从编译选项开始一直到函数调用关系的分析方法,主要是在传统静态代码分析的基础上增加对Kconfig以及Makefile的分析,为不同层次之间的静态代码分析提供了一种思路;同时给出了与编译选项相关的源代码的分层模型及其具体的分析做法,对于理解Linux内核源码特别是其中的编译依赖有现实意义。

2 Linux内核源码中的Kconfig与Makefile及其依赖关系描述

2.1 Linux内核源码中的依赖关系描述

首先,依赖关系的数学描述如下:

设有关系模式$R(U)$,其中$U\{A_1,A_2,\cdots,A_n\}$是关系的属性全集,X、Y是U的属性子集,设t和u是关系R上的任意两个元组,如果t和u在X的投影$t[X]=u[X]$推出$t[Y]=u[Y]$,即:$t[X]=u[X]=>t[Y]=u[Y]$,则称X决定Y,或Y依赖于X。记为$X\rightarrow Y$。

在Linux内核中,并不是所有源码都与编译选项相关,与编译选项相关的往往是一些可定制性的功能。这部分源代码自顶向下的组织结构如图1所示。

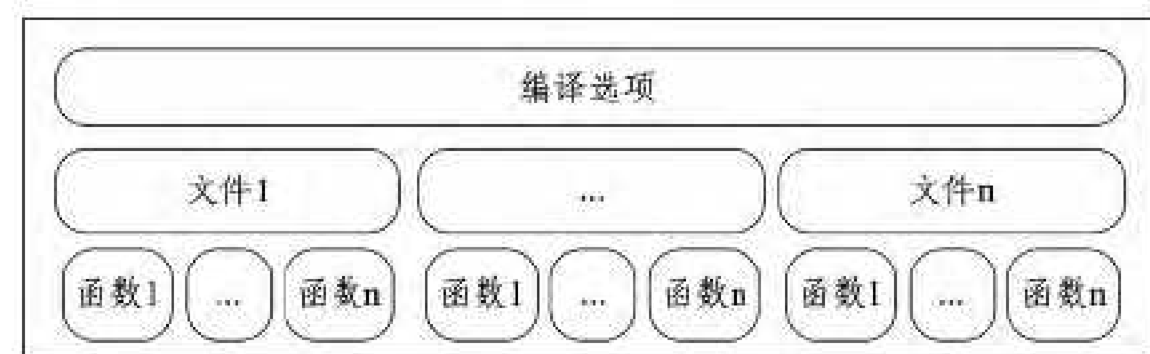

图1 Linux内核中与编译选项相关的源码的组织形式

本文受"核高基"科技重大专项:操作系统内核分析和安全性评估(2012ZX01039-004)资助。

江梦涛(1989－),男,硕士生,主要研究方向为开源操作系统,E-mail:insomniacdoll@gmail.com;潘朋飞(1989－),男,硕士生,主要研究方向为开源操作系统;宋 杨(1988－),男,硕士生,主要研究方向为开源操作系统。

由以上的结构图可知，从依赖关系的角度，与编译选项相关的内核源码可以大致划分为3个层次来描述。最顶层是由Kconfig文件所定义的编译菜单以及编译选项，向下一层是每个编译选项所对应的文件，最底层是文件对应的函数实现。

每个层次内部具有一定的依赖关系，比如函数的调用关系，代入上面的依赖关系的数学描述，即：关系模式$R(U)$，其中$U\{A_1,A_2,\cdots,A_n\}$是函数的定义（属性）全集，X、Y是U中任意两个函数的定义子集，如果在某个条件t下总能从$t[X]=u[X]$推出$t[Y]=u[Y]$，则说明函数之间在条件t下具有依赖关系。表现在具体的代码中即在某一个分支条件下函数X对于函数Y的调用。类似的关系在编译菜单层面和文件层面也广泛存在，具体体现在编译菜单、编译选项、文件之间的依赖关系上。

另外，每个层次内部的依赖关系并不是所有关系的全集，依赖关系也大量跨层次存在。比如编译选项与文件的对应关系、文件与函数定义的对应关系，都可以被认为是上下层次之间的依赖关系。比如两个编译选项之间的依赖关系的实现，内部依赖于文件之间的相互引用，而文件之间的相互引用具体体现在代码中就是大量的跨文件函数调用。

根据上面的描述以及大量的工程实践，可以得知大型软件源代码的内部关系是相当复杂的，如果想对其进行静态分析，必须对其层次内部以及层次之间的依赖关系进行挖掘。在Linux内核源码中，编译菜单、选项以及文件之间的对应关系是由Kconfig和Makefile两种文件来定义的。下面将按照图1的分层结构对Linux内核中与编译选项相关的源码进行依赖关系的分析与说明。

2.2 基于Linux内核源码中的Kconfig文件的编译选项依赖关系分析

Kconfig是现今一种事实上的编译配置的标准机制，它在许多著名的开源项目中均有应用，比如Linux kernel、Busybox以及uClibc。它有基本而简洁的语法，允许开发者自由地添加各种类型的配置、创建配置间的依赖关系以及为配置增加描述信息。相应地，有一系列相关的实用工具可以读取并解析Kconfig文件，并将结果输出为工程的标准配置文件。通常是以.config命名的一种配置，Makefile和autoconf.h等工具可以根据其中的内容来决定编译时的源文件名称。

Kconfig的其他优点还包括：可以为图形或者文本界面自动生成配置选项的菜单[1]，从而方便管理相关的配置选项；使用Kconfig，无需为工程的编译指定任何的构建标识符。比如在Linux驱动开发中，常见的做法是，在Kconfig配置文件中增加一个新的驱动配置选项，来区分内核是否要使用此种驱动。这种配置编译选项的做法可以方便快速地进行驱动的配置与编译，增加了Linux内核的可扩展性。

Kconfig文件描述的是编译选项菜单以及编译选项之间的依赖关系。这些关系实际上是相关源代码之间依赖的一次抽象，即编译菜单和选项之间的依赖关系最终将体现到文件之间的依赖关系，再向下就是函数或者变量之间的调用关系。本节将着重介绍最顶层的编译选项的组织结构以及与它们之间依赖关系相关的语法，因为分析将从这些结构与语法入手。

2.2.1 Linux内核源码中Kconfig文件的组织结构

Linux内核中的Kconfig配置文件是以目录结构进行组织的。每一个Kconfig配置文件的影响范围只在自己的层级（即目录结构）之中。比如，在每一个驱动或者体系架构支持的目录中都可以找到一个相关的配置文件，这些配置文件只能影响相对应的驱动或者体系结构目录本身，以及其所在目录中子目录的Kconfig配置文件。

图2是文件arch/x86/Kconfig的内容片段。

```
config X86_32
	def_bool !64BIT
	select CLKSRC_I8253

config X86_64
	def_bool 64BIT
	select X86_DEV_DMA_OPS
```

图2 Kconfig内容样例片段

Kconfig文件的每一行均以一个关键字为起始，其后可以跟随多个参数。Config关键字代表一个新的配置选项。其下的行定义了这个配置选项的一些属性，属性可以是配置选项的类型、输入提示、依赖关系以及默认值。一个配置选项可以用相同的名字被定义多次，但是每一次定义均必须有一个唯一的输入提示，同时其类型必须不能冲突[1]。

2.2.2 Kconfig依赖关系语法描述

本文关注的是关于依赖关系的语法描述，故以下只重点介绍Kconfig中的依赖相关的语法，其他语法可以参考官方文档中的kconfig-language.txt。

编译选项的依赖主要有两种类型，下面分别介绍。

(1)依赖关系，以下是语法描述：

"depends on" <expr>

这个表达式为菜单选项定义了一个依赖。如果有多个依赖需要定义，那么这些表达式之间需要使用"&&"符号来连接。依赖定义将被应用到当前菜单选项的所有其他定义中。同时由于菜单选项还接受if表达式，因此以下两个表达方式是等价的：

```
bool "foo" if BAR
  default y if BAR
```

以及

```
depends on BAR
  bool "foo"
  default y
```

(2)逆向依赖关系，以下是语法描述：

"select" <symbol> ["if" <expr>]

普通的依赖定义降低了符号的使用频率上限，而逆向依赖定义可以用作强制另一个符号的使用频率的下限。当前菜单符号的值被用作<symbol>符号可以被设置的最小值，如果<symbol>符号被选择了许多次，上限就是其中的最大值。

逆向依赖只能被使用在布尔值或者三态符号中。

select应当谨慎使用，因为select符号会绕过其依赖强制为一个符号赋值。滥用select可能会导致如下情况：当FOO依赖BAR选项而BAR未被赋值时，仍然可以对FOO这个符号执行select操作。

通常情况下，select应只被用在非可见符号中，同时这些符号必须没有任何依赖。这种使用方式会降低select符号的可用性，但同时会有效避免非法的编译选项配置[1]。

接下来是菜单选项之间的依赖关系。

一个菜单选项在菜单树中的位置可以用两种方式来指

定。首先,它可以被显示指定,比如:

```
menu "Network device support"
  depends on NET
config NETDEVICES
  …
endmenu
```

以上面的菜单结构为例,在"menu"…"endmenu"区块中的所有的选项都会被定义为"Network device support"菜单的子菜单。所有的子菜单将从父菜单继承依赖选项。还以上面的结构为例,当配置 NETDEVICES 被选中的时候,将自动为其添加名为 NET 的依赖。

另一个生成菜单结构的方法是分析菜单选项的依赖。如果一个菜单选项依赖前一个菜单选项,那么前者就可以被认为是后者的子菜单。当然,父菜单必须在依赖列表之中,同时两者的选择条件均需要为真。

当父菜单的选择为"n"时,子菜单一定是不可见的;当父菜单的属性为可见,那么子菜单必须也只能是可见状态[1]。

2.3 基于 Linux 内核源码中的 Kconfig 文件的编译选项依赖关系分析

在软件开发中,make 是一个工具程序(Utility software),经由读取名为"Makefile"的文件,自动化建构软件。它是一种转化文件形式的工具,转换的目标称为"target";与此同时,它也检查文件的依赖关系,如果需要,它会调用一些外部软件来完成任务[2]。它的依赖关系检查系统非常简单,主要根据依赖文件的修改时间进行判断。大多数情况下,它被用来编译源代码,生成结果代码,然后把结果代码连接起来生成可执行文件或者库文件。它使用叫做"Makefile"的文件来确定一个 target 文件的依赖关系,然后把生成这个 target 的相关命令传给 shell 去执行。

Makefile 文件描述的是编译选项与文件之间的依赖关系。这些关系实际上是源代码之间依赖在编译选项依赖基础之上的细化,或者是具体源代码之间依赖的抽象。本节将着重介绍 Makefile 中与上述依赖关系相关的组织结构与语法。

2.3.1 Linux 内核源码中 Makefile 文件的组织结构

Linux 内核文档 Makefile. txt 的说明如图 3 所示。

```
The Makefiles have five parts:

    Makefile            the top Makefile.
    .config             the kernel configuration file.
    arch/$(ARCH)/Makefile   the arch Makefile.
    scripts/Makefile.*  common rules etc. for all kbuild Makefiles.
    kbuild Makefiles    there are about 500 of these.
```

图 3 Linux 内核文档 Makefile. txt 的说明

Linux 的 Make 体系由如下几部分组成:

①顶层 Makefile:通过读取配置文件,递归编译内核代码树的相关目录,从而产生两个重要的目标文件:vmlinux 和模块。

②内核相关 Makefile:位于 arch/$(ARCH) 目录下,为顶层 Makefile 提供与具体硬件体系结构相关的信息。

③公共编译规则定义文件:包括 Makefile. build、Makefile. clean、Makefile. lib、Makefile. host 等。这些文件位于 scripts 目录中,定义了编译需要的公共的规则和定义。

④内核配置文件. config:通过调用 make menuconfig 或者 make xconfig 命令,用户可以选择需要的配置来生成期望的目标文件。

⑤Kbuild Makefile:主要为整个 Makefile 体系提供各自模块的目标文件定义,上层 Makefile 根据它所定义的目标来完成各自模块的编译。

对于编译选项对应文件的抓取主要是关系到上述的②和⑤。之后的章节会进行详细分析说明。

2.3.2 依赖关系语法描述

本文关注的是关于依赖关系的语法描述,故以下只重点介绍 Makefile 中的依赖相关的语法,其他语法可以参考官方文档中的 Makefiles. txt。

根据 Linux 内核文档 Makefile. txt 的说明以及参考有关 Makefile 经典书籍,总结出 Makefile 中编译选项定义文件有 4 种方法。

①直接定义

```
$(obj-m) specify object files which are built as loadable
kernel modules.

A module may be built from one source file or several source
files. In the case of one source file, the kbuild makefile
simply adds the file to $(obj-m).

Example:
    #drivers/isdn/i4l/Makefile
    obj-$(CONFIG_ISDN_PPP_BSDCOMP) += isdn_bsdcomp.o

Note: In this example $(CONFIG_ISDN_PPP_BSDCOMP) evaluates to 'm'
```

图 4 Linux 内核文档关于 Makefile 直接定义的说明

此例中,编译选项 ISDN_PPP_BSDCOMP 对应文件中包含 isdn_bsdcomp. c。

②间接定义

```
Note: In this example $(CONFIG_ISDN_PPP_BSDCOMP) evaluates to 'm'

If a kernel module is built from several source files, you specify
that you want to build a module in the same way as above; however,
kbuild needs to know which object files you want to build your
module from, so you have to tell it by setting a $(<module_name>-y)
variable.

Example:
    #drivers/isdn/i4l/Makefile
    obj-$(CONFIG_ISDN_I4L) += isdn.o
    isdn-y := isdn_net_lib.o isdn_v110.o isdn_common.o

In this example, the module name will be isdn.o. Kbuild will
compile the objects listed in $(isdn-y) and then run
"$(LD) -r" on the list of these files to generate isdn.o.
```

图 5 Linux 内核文档关于 Makefile 间接定义的说明

此例中,编译选项 ISDN_I4L 对应 isdn. o,而 isdn. c 不是一个文件,而是中间目标文件,它对应到 isdn_net_lib. c、isdn_v110. c 和 isdn_common. c 3 个文件。从而编译选项 ISDN_I4L 对应文件中包含这 3 个文件。

③条件表达式

```
〈conditional-directive〉
  〈text-if-true〉
endif
```

以及:

```
〈conditional-directive〉
〈text-if-true〉
    else
    〈text-if-false〉
    endif
```

其中〈conditional-directive〉表示条件关键字,条件关键字有 4 个:

• ifeq(〈arg1〉,〈arg2〉):比较参数"arg1"和"arg2"的值是否相同,如果相同,则为真;

• ifneq(〈arg1〉,〈arg2〉):比较参数"arg1"和"arg2"的值是否不相同,如果不同,则为真;

• ifdef 〈variable-name〉：如果变量〈variable-name〉的值非空，那么表达式为真；

• ifndef 〈variable-name〉：如果变量〈variable-name〉的值为空，那么表达式为真。

以 ifeq 为例（见图 6）：

```
ifeq ($(CONFIG_BLOCK),y)
obj-y +=    buffer.o bio.o block_dev.o direct-io.o mpage.o ioprio.o
```

图 6　Makefile 中 if 分支条件定义样例

此例中，条件语句中定义的 buffer.c、bio.c 等文件都属于编译选项 BlOCK。

④目录递归

```
A Makefile is only responsible for building objects in its own
directory. Files in subdirectories should be taken care of by
Makefiles in these subdirs. The build system will automatically
invoke make recursively in subdirectories, provided you let it know of
them.

To do so, obj-y and obj-m are used.
ext2 lives in a separate directory, and the Makefile present in fs/
tells kbuild to descend down using the following assignment.

Example:
    #fs/Makefile
    obj-$(CONFIG_EXT2_FS) += ext2/

If CONFIG_EXT2_FS is set to either 'y' (built-in) or 'm' (modular)
the corresponding obj- variable will be set, and kbuild will descend
down in the ext2 directory.
Kbuild only uses this information to decide that it needs to visit
the directory, it is the Makefile in the subdirectory that
specifies what is modules and what is built-in.
```

图 7　Linux 内核文档关于 Makefile 目录递归的说明

Makefile 文件负责编译当前目录下的目标文件，子目录中的文件由子目录中的 makefile 文件负责编译。此例中，ext2 是一个子目录，EXT2-FS 对应到 ext2 目录下的 makefile 文件，使用目录递归进行解析。子目录中以 obj-y 定义的文件，将属于编译选项 EXT2-FS；其他正常以编译选项规则定义的文件，继续根据①、②和③规则进行解析。

本节通过对 Linux 源码中 Kconfig 文件以及 Makefile 文件的组织结构以及与依赖关系相关的语法的介绍，明确了 Linux 内核源码中使用 Kconfig 进行定义的编译菜单和选项以及使用 Makefile 所定义的编译选项与实际文件之间的对应关系的具体方式，为之后解析编译选项之间以及编译选项与文件之间的依赖关系做了铺垫。

3　Linux 内核源码中的 Kconfig 与 Makefile 及其依赖关系描述

3.1　编译选项之间的依赖关系解析

根据上一节对于 Linux 内核源码中编译选项的组织结构以及语法的相关说明，可以知道，要解析编译选项之间的依赖关系，需要解析出的内容包括：编译选项名字、编译选项类型、编译选项与菜单项的父子关系，以及编译选项之间的依赖关系。下面就以上各类内容的解析作出说明。

3.1.1　编译选项名字的解析

Linux 内核文档 kconfig-language.txt 中关于 Kconfig 符号的说明如图 8 所示。

```
Kconfig syntax
--------------

The configuration file describes a series of menu entries, where every
line starts with a keyword (except help texts). The following keywords
end a menu entry:
- config
- menuconfig
- choice/endchoice
- comment
- menu/endmenu
- if/endif
- source
The first five also start the definition of a menu entry.
```

图 8　内核中 Kconfig 编译选项的语法说明

对于编译选项名称，需要解析与抓取的是 config、menuconfig 以及 menu 关键字后双引号中的内容。

根据 Kconfig 语法说明，choice 选项中依然使用 config 来定义编译选项，source 选项后面的属性为引入的外部 Kconfig 文件的路径，comment 选项只用来给用户编译内核时提供提示信息，因此所有的编译选项名称均位于 config、menuconfig 和 menu 关键字之后。

3.1.2　编译选项类型的解析

图 9、图 10 是 Linux 内核源码文档中 kconfig-language 关于编译选项类型的说明。

```
- type definition: "bool"/"tristate"/"string"/"hex"/"int"
  Every config option must have a type. There are only two basic types:
  tristate and string; the other types are based on these two. The type
  definition optionally accepts an input prompt, so these two examples
  are equivalent:
```

图 9　内核中 Kconfig 编译选项类型的语法说明(1)

```
- type definition + default value:
  "def_bool"/"def_tristate" <expr> ["if" <expr>]
  This is a shorthand notation for a type definition plus a value.
  Optionally dependencies for this default value can be added with "if".
```

图 10　内核中 Kconfig 编译选项类型的语法说明(2)

根据以上说明可以得知，在编译选项的属性中关于编译选项的类型的关键字有 bool、tristate、string、hex、int 5 种。这 5 种关键字可以归类为两类，即 string 和 tristate。另外 def—bool 或者 def-trisate 类型是 default 关键字与 bool、trisate 关键字的缩写版本，并非新的类型关键字。

因此，只需要解析每个 config、menuconfig 选项的属性中的上述关键字，就可以得到其编译选项的类型。

3.1.3　编译选项、菜单项的父子关系的解析

Linux 内核文档 kconfig-language.txt 中关于编译菜单的说明如图 11 所示。

```
mainmenu:

  "mainmenu" <prompt>

This sets the config program's title bar if the config program chooses
to use it. It should be placed at the top of the configuration, before any
other statement.
```

图 11　内核中 Kconfig 编译菜单的语法说明(1)

mainmenu 是整个内核编译选项菜单的起始点，同时关注 source 关键字如下（见图 12）：

```
source:

  "source" <prompt>

This reads the specified configuration file. This file is always parsed.
```

图 12　内核中 Kconfig 编译菜单的语法说明(2)

内核的所有 Kconfig 文件是通过 source 关键字联系起来的，首个 Kconifg 文件位于内核源码根目录下，如图 13 所示。

```
#
# For a description of the syntax of this configuration file,
# see Documentation/kbuild/kconfig-language.txt.
#
mainmenu "Linux/$ARCH $KERNELVERSION Kernel Configuration"

config SRCARCH
    string
    option env="SRCARCH"

source "arch/$SRCARCH/Kconfig"
```

图 13　内核中 Kconfig 编译菜单的语法说明(3)

通过 SRCARCH 选项决定了整个内核源码的编译所针

对的计算机体系结构类型。

所以，将该 kconfig 文件作为起点，递归地遍历 source 关键字引用的 kconfig 文件，利用栈结构实现对所有编译选项之间父子关系的记录。

```
menu:

  "menu" <prompt>
  <menu options>
  <menu block>
  "endmenu"

This defines a menu block, see "Menu structure" above for more
information. The only possible options are dependencies and "visible"
attributes.
```

图 14　内核中 Kconfig 编译菜单的语法说明(4)

menu/endmenu 关键字同样决定了父子关系，我们这里的父子关系定义 menu 为父亲，次 menu 中的编译选项为其孩子。

具体实现流程如图 15 所示。

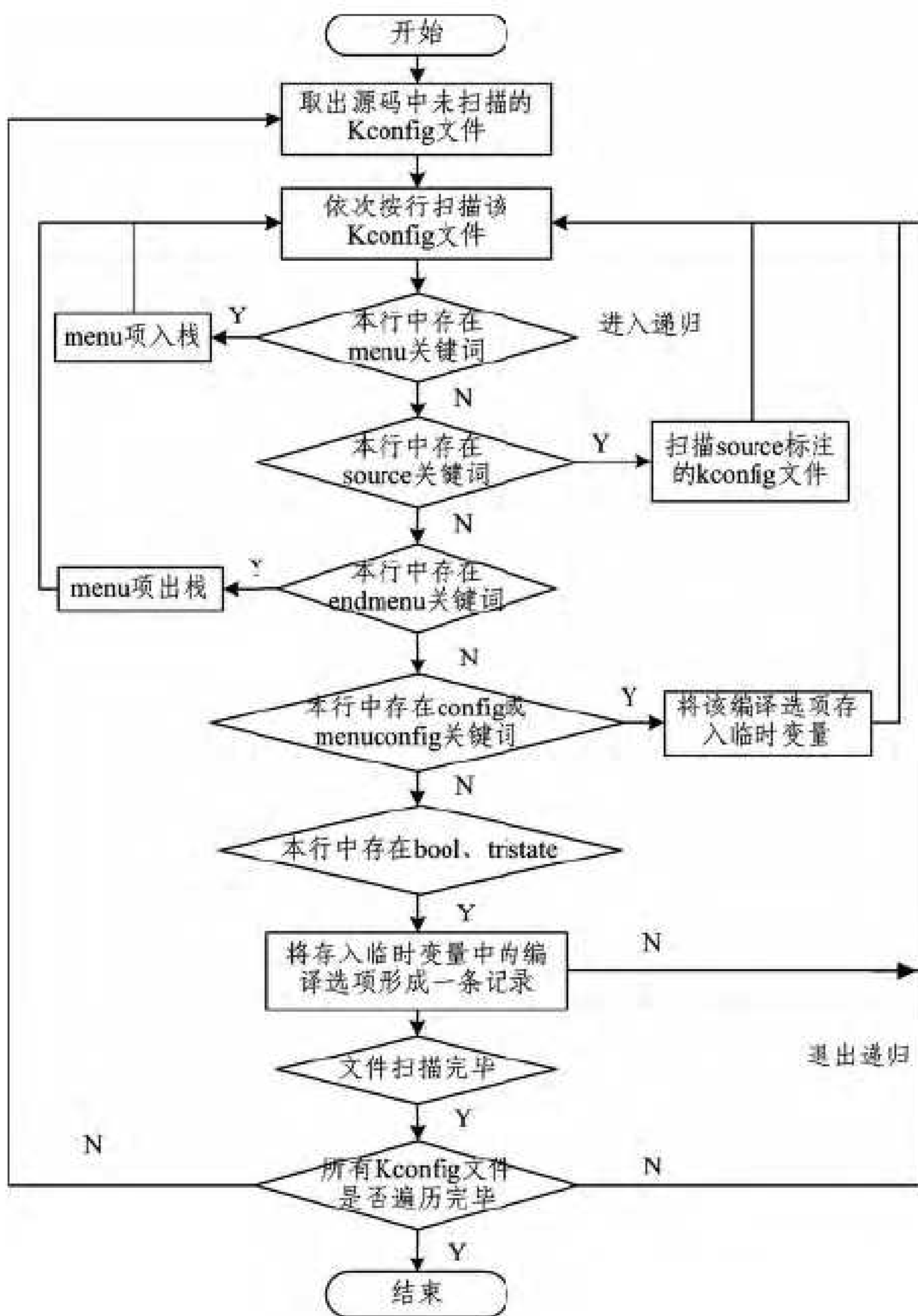

图 15　内核中 kconfig 编译菜单选项的抓取流程

图中的步骤描述如下：

(1)以 linux/Kconfig 为起点，设置 SRCARCH 变量为某一种体系架构如 X86 或 ARM；

(2)mainmenu 入栈，作为菜单结构的根，若遇到"source"则递归进入"source"所引用的 Kconfig 文件；

(3)若遇到 config 或者 menuconfig 关键字，记录其编译选项的名字，并读取下一行，同时记录编译选项的类型，最后读取栈顶的 menu 作为其父亲，形成一条记录保存起来；

(4)若遇到 menu 关键字，则将 menu 入栈，同时将 menu 记录保存起来。

3.1.4　编译选项之间的依赖关系解析

编译选项之间的依赖关系使用 depends on/select 关键字时，若一个编译选项与多个编译选项之间存在依赖关系，还会使用逻辑表达式的形式来表现，因此解析时需要记录整个逻辑表达式。此外，使用 if 语句也可以表现编译选项之间的依赖关系，因此，这种情况的依赖关系也不能忽略。

具体的做法是，解析 config 关键字时，不仅需要解析其名称，还需要解析其下层级的 depends on/select 关键字与其之后的名称，并将这种依赖关系存储起来。

具体实现流程如图 16 所示。

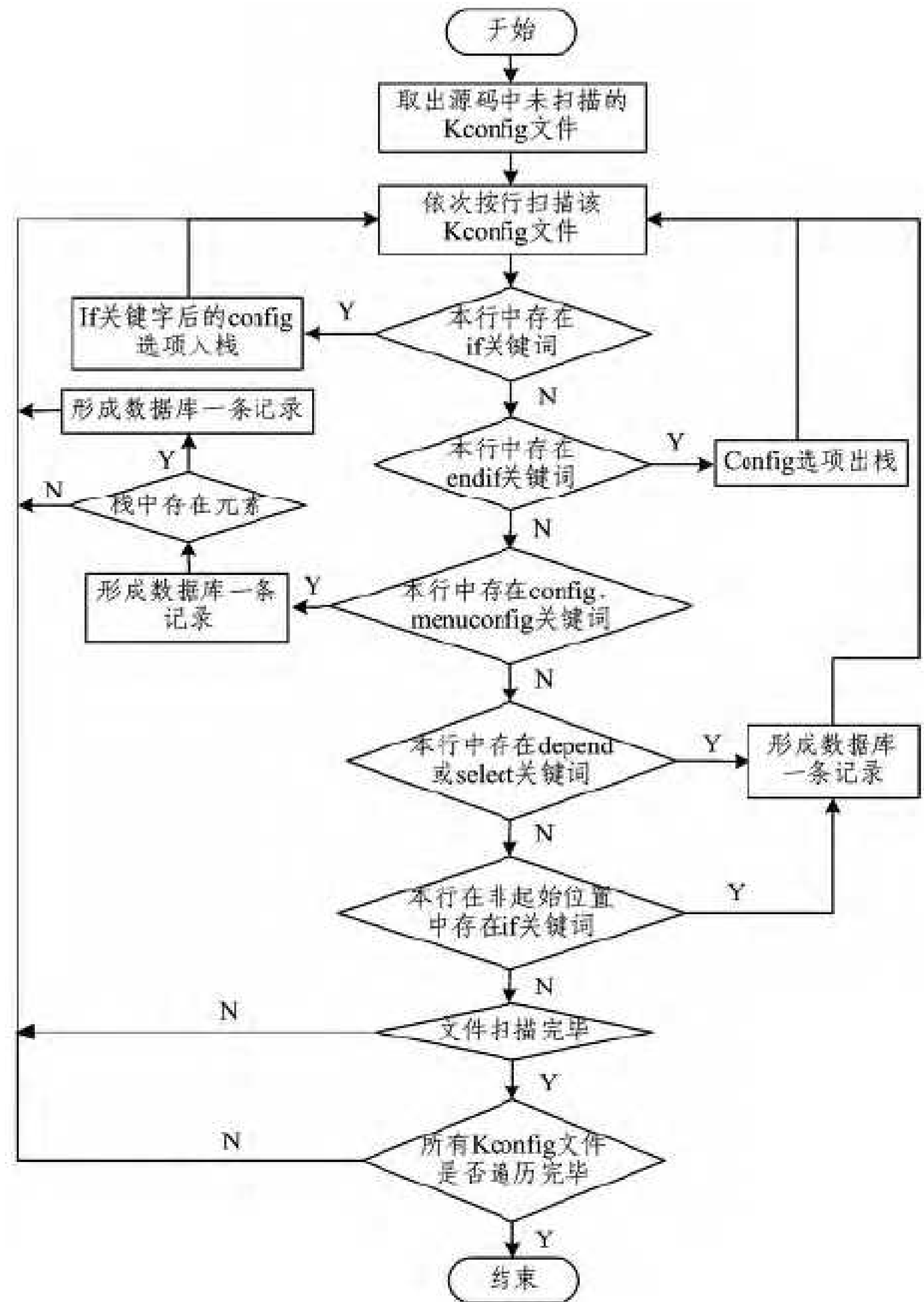

图 16　内核中 Kconfig 编译选项依赖关系的抓取流程

图中的步骤描述如下：

(1)遍历内核源码中所有的 Kconfig 文件，扫描文件中的每一行，若遇到 menu 或 menuconfig 关键字，将编译选项存入临时变量，直到再次遇到该关键字，则更新临时变量；

(2)若遇到 if 语句则将其后的编译选项入栈，遇到 endif 则出栈，若栈中存在元素，则栈定元素始终与新读取到的编译选项存在依赖关系，若遇到"depend on"或"selecet"关键字，则形成一条依赖关系的记录。

3.2　编译选项与文件之间的依赖关系解析

第 2.3.2 节中的介绍基本包括了定义编译选项所对应的文件时用到的所有语法。这些语法在实际的 Makefile 中是复杂结合的，所以要解析出编译选项对应的文件，需要基于上述语法结构做语法分析，得到相应的语法树，将其存储起来。

由于 Makefile 是一种基于文本行的描述文件，因此解析程序的编写也需要基于行来进行[2]。

具体流程如图 17 所示。

图中的步骤描述如下：

(1)从 directory 表中依次取出所有的 makefile 文件，保

存在 list 数据结构中；

(2)遍历此 list，依次对每个 makefile 进行分析，每次读取一行，进行分析；

(3)识别文本内容中特殊标签：

①ifeq、ifneq、ifdef、ifndef、else、endif，利用树形结构来保存其层次；

②进行编译选项提取，主要涉及到 3 种情况：config 名称；config 中间变量；包含于①中 if 标签中的 obj-y 标签；

③提取编译选项与文件的对应关系，文件的定义主要涉及 3 种情况：基本的.o 定义；"\"标识出的多行定义；"/"标识出的子目录定义。

(4)将步骤(3)中识别出的编译选项与其对应的文件存入 map<string，list<string>>数据结构中；

(5)进行 map 分析，遍历 map，逐项分析：

①取出 map 的 key 值，在 module 表进行查找匹配，如若为空，进行下次遍历；

②取出 map 的 value 值，value 为 list 类型，需要遍历分析。value 值有 3 种情况：源码的.c 文件直接进行存储即可；源码中的 makefile 文件，进行递归分析；中间变量，以其为 key 值，取出 map 中对应的 value 进行再分析。

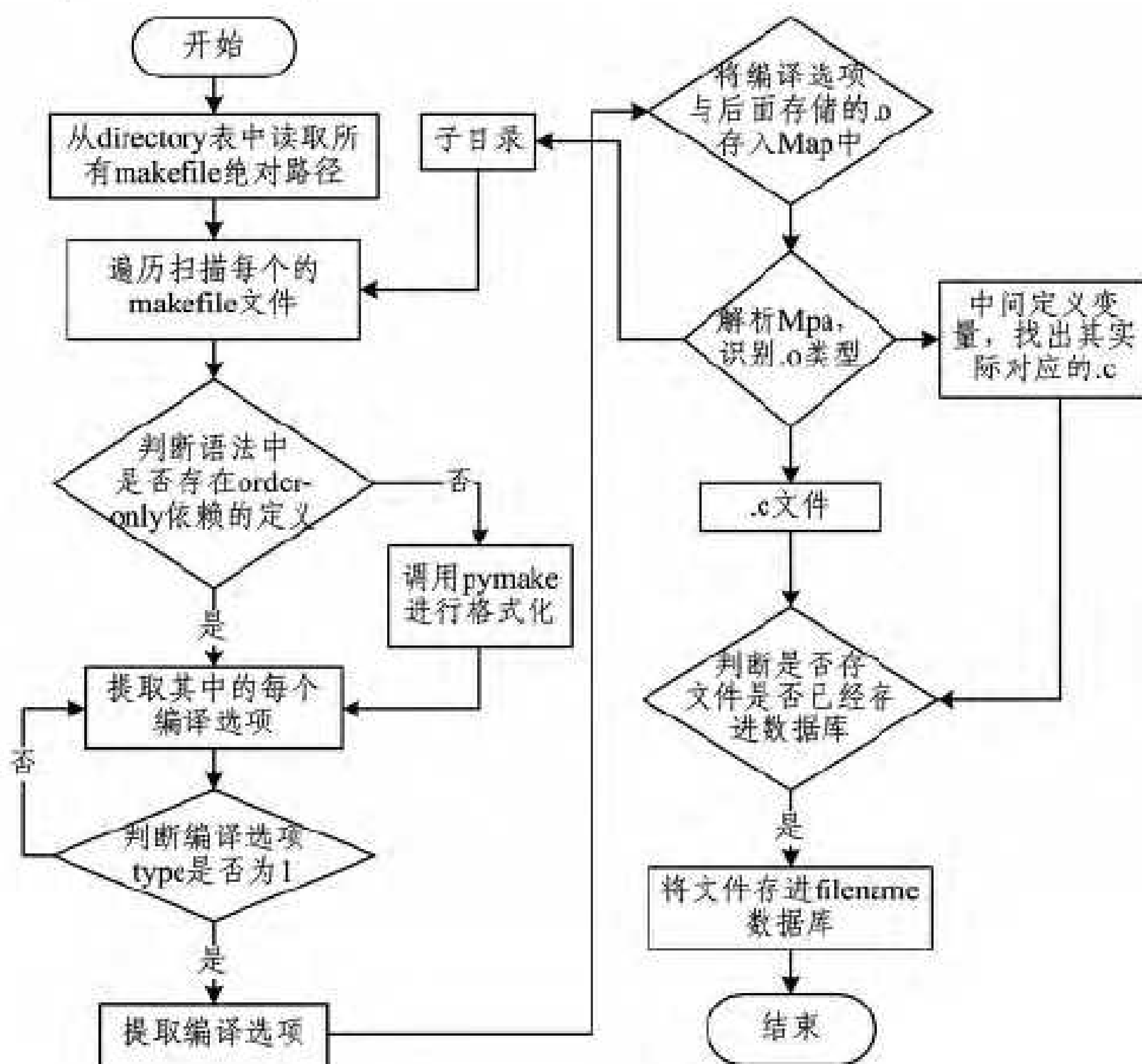

图 17　内核中 Makefile 的编译选项与文件对应关系的抓取流程

3.3　函数之间的依赖关系解析

由于编译选项之间的依赖关系，在内核代码中主要体现为对 EXPORT 函数的调用，因此我们关注的依赖关系是函数调用了被 EXPORT 关键字所引用的函数，从而形成的依赖关系。在此基础上剖析编译选项之间依赖关系与函数调用关系之间的关联。

函数间的依赖关系解析可以借助正则表达式来进行。首先扫描整个需要解析的源码，找出被 EXPORT 关键字所标识的函数，然后使用正则表达式匹配每一个函数声明及其函数体，再次使用正则表达式匹配每一个函数体中的函数调用，判断被调用的函数是否在之前解析好的被 EXPORT 关键字所标识的函数列表中，如果存在，则保存关系，否则跳过。

可以匹配函数定义的正则表达式如图 18 所示。

```
1  |(?:(extern|static)(?:\s+))?                               #修饰符
2  (?:                                                         #返回值类型
3      (
4          (?:[a-zA-Z_]\w*)                                    #有效的符号
5          (?:(?:\s(?:\s*\*\s*)*)|(?:(?:\s*\*\s*)*\s))         #指针类型符号
6      )
7      ([a-zA-Z_]\w*)\s*                                       #函数名
8  )
9  (\([^\)]*\)?)\s*                                            #参数列表
10 (?:                                                         # {} 或者 ;
11     (\{                                                     #左大括号
12         (?:[^\{\}]*)                                        #函数体
13         (?:
14             (?:(?'brace'\{)[^\{\}]*)+
15             (?:(?'-brace'\})[^\{\}]*)+
16         )*
17         (?(brace)(?!))
18     \})                                                     #右大括号
19     |;
20 )
```

图 18　匹配 C 语言函数定义的正则表达式

通过以上步骤，就可以解析出函数间的依赖关系。

3.4　文件之间的依赖关系解析

文件之间的依赖关系基于函数之间的依赖关系。一旦一个文件的函数调用了另外一个文件中被 EXPORT 关键字所标识的函数，那么这两个文件之间也具有依赖关系，将关系保存起来即可。

经过以上的步骤对 Linux 内核源码进行解析之后，可以得到各个层级之间的内部关系以及层级之间的依赖。具体如下：

(1)编译菜单以及编译选项之间的关系由 3.1 节中对 Kconfig 文件的解析获得；

(2)编译选项与文件之间的关系由 3.2 节中对 Makefile 文件的解析获得；

(3)文件之间的依赖关系由 3.3 节中函数调用关系反向推出；

(4)文件与函数、函数与函数之间的依赖关系由 3.3 节中对实际源码的解析获得。

4　依赖关系的展示

根据第 3 节中的描述，可以对 Linux Kernel 源码进行编译选项、文件以及函数依赖关系的分析。为了便于其展示，将其关系保存起来，然后使用一定的可视化技术将这些关系展示出来即可。比如可以为这些关系建立一系列的数据库表。样例 E-R 图如图 19 所示。

对 kconfig 文件进行语法分析，将分析结果存进数据库。编译选项在数据库中主要是填写 module 表以及 moduleRelation 表；module 表存储 kconfig 文件中的所有编译选项及其编译选项的类型。其中重要的字段为：moduleName，即编译选项的名称；Moduletype，即便以选项的类型，bool，trsitate 或者 string 等类型；Deep 存储了此编译选项的深度，是为了前台显示层级关系而增加的字段；ParentNodeId 记录了当前节点的父节点 Id。

moduleRelation 表存储编译选项与其他编译选项的依赖关系。脚本爬取 kconfig 文件中的 config 或 menuconfig 选项及其依赖(depend on 和 select)的选项。LogicRelation 记录了编译选项之间依赖关系的逻辑表达式，即对某模块的依赖的与或非等关系。

编译选项在数据库中主要是填写 FileName 表。

(下转第 454 页)

法比以一般huffman思想为基础的构造最优Delta传播树的物化视图增量保持算法在效率上有所提高。本文实验基于生产线质量控制系统，下一步我们将把该算法应用到其他的实验环境中，进一步验证算法有效性的同时对算法进行进一步的优化。

参 考 文 献

[1] Blakeley J A, Larson P, Tompa F W. Efficiently Updating Materialized Views[C]// Proceedings of ACM SIGMOD Conference. 1986:61-71

[2] Gupta A, Mumick I S, Subrahmanian V S. Maintaining views incrementally[C]// Proceedings of ACM SIGMOD Conference. 1993:157-166

[3] Lee K-Y, Son J-H, Kim M-H. Efficient Incremental View Maintenance in Data Warehouses[C]// Proceedings of CIKM01. Atlanta Georgia USA, 2001

[4] 王新军，洪晓光，孙明，等. 物化视图增量保持的改进算法[J]. 计算机工程, 2003, 29(21)

[5] 王新军，洪晓光，王海洋，等. 数据仓库中多源物化视图的一种有效更新算法[J]. 计算机研究与发展, 2004, 41(5)

[6] 孙建青，李风云. 多源物化视图更新的一种流水线并行算法[J]. 山东师范大学学报, 2004, 19(1)

[7] Zhou Jing-ren, Larson P, Elmongui H G. Lazy Maintenance of Materialized Views[C]// VLDB'07. ACM, Vienna, Austria, September 2007

[8] Zhou Li-juan, Shi Qian, Geng Hai-jun. The minimum incremental maintenance of Materialized Views in Data Warehouse[C]// 2nd International Asia Conference on Informatics in Control, Automation and Robotics. 2010

[9] Nica A. Incremental maintenance of Materialized Views with outerjoins[J]. Information Systems, 2012, 37: 430-442

[10] 武彤，赵雪，赵洵. 动态更新实物化视图以提高OLAP查询效率[J]. 计算机科学, 2012, 39(6A)

[11] Christopher J V W. Data Structures and C Programs [M]// Reading. Mass.: Addison-Wesley, 1988

[12] [美]Thomas H, Cormen Charles E, Leiserson Ronald L, 等. 算法导论(第三版)[M]. 殷建平，徐云，王刚，等译. 机械工业出版社, 2013

(上接第450页)

函数调用关系主要填写两张表格：FunctionInfo表用于存储export函数、变量与文件的对应关系；FunctionCall表用于存储函数之间的调用关系。

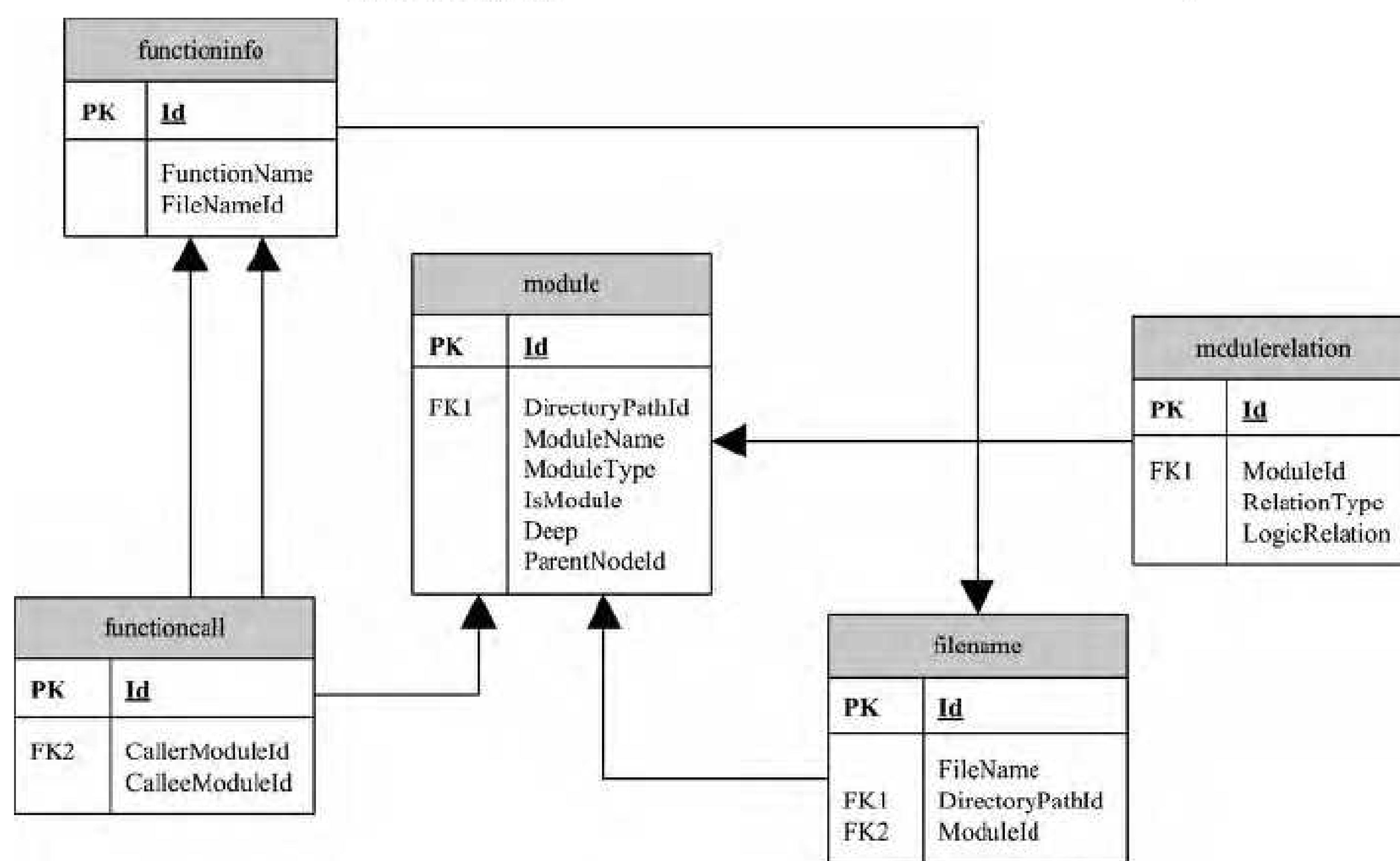

图19 Linux内核源码依赖关系E-R图表示

以上ER图描述可以看作是针对Linux内核源码进行了分层模型的建立。从顶层抽象描述了编译选项的module实体不仅可以获得本层次之间的关系(moduleRealtion)，也可以获取到其与下层文件以及函数之间的关系。即由以上描述，可以从编译选项的依赖得出文件的依赖，进而追溯到函数调用之间的依赖，顶层抽象的宏观的关系最终落实到了底层的具体的关系之上。

结束语 本文基于Kconfig与Makefile这两种在工程中大量使用的工具的语法，介绍了一种解析Linux内核源码中编译选项、文件以及函数之间的依赖关系的方法，同时为大型软件的源码的静态分析、阅读和学习提供了一种思路。另外本文还提供了编译选项、文件以及函数之间关系展示的简单样例。本文的不足之处在于，所有分析只局限于静态，一些动态运行时才会体现的关系难以通过现有手段进行捕获与存储，希望在以后的研究中能够继续完善。

参 考 文 献

[1] Linux Kernel Organization, Inc. Kconifg Introductionin Linux Kernel Source Code[OL]. https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt

[2] Linux Kernel Organization, Inc. Makefile Introductionin Linux Kernel Source Code[OL]. https://www.kernel.org/doc/Documentation/kbuild/makefiles.txt

[3] Linux Kernel Organization, Inc. Building External Modules in Linux Kernel[OL]. https://www.kernel.org/doc/Documentation/kbuild/modules.txt