2.6 内核的源码树目录下一般都会有两个文件：Kconfig 和 Makefile。分布在各目录下的 Kconfig 构成了一个分布式的内核配置数据库，每个 Kconfig 分别描述了所属目录源文件相关的内核配置菜单。在内核配置 make menuconfig(或 xconfig 等)时，从 Kconfig 中读出配置菜单，用户配置完后保存到.config(在顶层目录下生成)中。在内核编译时，主 Makefile 调用这个.config，就知道了用户对内核的配置情况。

上面的内容说明：Kconfig 就是对应着内核的配置菜单。假如要想添加新的驱动到内核的源码中，可以通过修改 Kconfig 来增加对我们驱动的配置菜单，这样就有途径选择我们的驱动，假如想使这个驱动被编译，还要修改该驱动所在目录下的 Makefile。

因此，一般添加新的驱动时需要修改的文件有两种（注意不只是两个）

*Kconfig
*Makefile

要想知道怎么修改这两种文件，就要知道两种文档的语法结构。

First: Kconfig

每个菜单项都有一个关键字标识，最常见的就是 config。

语法：
config symbol

options
<!--[if !supportLineBreakNewLine]-->
<!--[endif]-->

symbol 就是新的菜单项，options 是在这个新的菜单项下的属性和选项

其中 options 部分有：

1、类型定义：
每个 config 菜单项都要有类型定义，bool：布尔类型， tristate 三态：内建、模块、移除， string：字符串， hex：十六进制， integer：整型

例如 config HELLO_MODULE
bool "hello test module"

bool 类型的只能选中或不选中，tristate 类型的菜单项多了编译成内核模块的选项，假如选择编译成内核模块，则会在.config 中生成一个

CONFIG_HELLO_MODULE=m 的配置，假如选择内建，就是直接编译成内核影响，就会在.config 中生成一个 CONFIG_HELLO_MODULE=y 的配置.

2、依赖型定义 depends on 或 requires
指此菜单的出现是否依赖于另一个定义

config HELLO_MODULE
bool "hello test module"
depends on ARCH_PXA
　　这个例子表明 HELLO_MODULE 这个菜单项只对 XScale 处理器有效，即只有在选择了 ARCH_PXA， 该菜单才可见(可配置)。

3、帮助性定义
只是增加帮助用关键字 help 或---help---
<!--[if !supportLineBreakNewLine]-->
<!--[endif]-->

更多详细的 Kconfigconfig 语法可参考：

Second: 内核的 Makefile

内核的 Makefile 分为 5 个组成部分：
Makefile　　最顶层的 Makefile
.config　　内核的当前配置文档，编译时成为顶层 Makefile 的一部分
arch/$(ARCH)/Makefile 和体系结构相关的 Makefile
s/ Makefile.*　　一些 Makefile 的通用规则
kbuild Makefile　　各级目录下的大概约 500 个文档，编译时根据上层 Makefile 传下来的宏定义和其他编译规则，将源代码编译成模块或编入内核。

顶层的 Makefile 文档读取 .config 文档的内容，并总体上负责 build 内核和模块。Arch Makefile 则提供补充体系结构相关的信息。 s 目录下的 Makefile 文档包含了任何用来根据 kbuild Makefile 构建内核所需的定义和规则。

（其中.config 的内容是在 make menuconfig 的时候，通过 Kconfig 文档配置的结果）

在 linux2.6.x/Documentation/kbuild 目录下有详细的介绍有关 kernel makefile 的知识。

最后举个例子：
假设想把自己写的一个 flash 的驱动程式加载到工程中，而且能够通过 menuconfig 配置内核时选择该驱动该怎么办呢？能够分三步：

第一：将您写的 flashtest.c 文档添加到/driver/mtd/maps/ 目录下。

第二：修改/driver/mtd/maps 目录下的 kconfig 文档：
config MTD_flashtest
tristate "ap71 flash"

这样当 make menuconfig 时 ，将会出现 ap71 flash 选项。

第三：修改该目录下 makefile 文档。
添加如下内容：obj-$(CONFIG_MTD_flashtest) += flashtest.o

这样，当您运行 make menucofnig 时，您将发现 ap71 flash 选项，假如您选择了此项。该选择就会保存在.config 文档中。当您编译内核时，将会读取.config 文档，当发现 ap71 flash 选项为 yes 时，系统在调用/driver/mtd/maps/下的 makefile 时，将会把 flashtest.o 加入到内核中。即可达到您的目的

linux2.6.x 的配置文件 kconfig 语法
linux 在 2.6 版本以后将配置文件由原来的 config.in 改为 kconfig，对于 kconfig 的语法在**/Documentation/kbuild/kconfig-language.txt** 中做了详细的说 明，在这里给出 kconfig-language.txt 的中文版。

介绍

在配置数据库的配置选项是以树的形式组织的：

```
+- Code maturity level options
| +- Prompt for development and/or incomplete code/drivers
+- General setup
| +- Networking support
| +- System V IPC
| +- BSD Process Accounting
| +- Sysctl support
+- Loadable module support
| +- Enable loadable module support
|    +- Set version information on all module symbols
|    +- Kernel module loader
+- ...
```

每个选项都有其自己的依赖关系。这些依赖关系决定了选项是否是可见的。父选

项可见，子选项才能可见。

菜单选项

大多数的选项都定义了一个配置选项，其它选项则有助于对它们进行组织。(原文：Most entries define a config option, all other entries help to organize them.)一个配置选项定义可以是下面的形式：

```
config MODVERSIONS
bool "Set version information on all module symbols"
depends MODULES
help
  Usually, modules have to be recompiled whenever you switch to a new
  kernel
```

每行都是以关键字开始，并可以接多个参数。"config" 为定义了一新的配置选项。下面的几行定义了该配置选项的属性。属性可以是该配置选项的类型，输入提示(input prompt)，依赖关系，帮助信息和默认值。一
配置选项可以用相同的名字定义多次，但每个定义只能有一个输入提示并且类型还不能冲突。

菜单属性

菜单选项可以有多个属性。并不要求这些属性可以用在任何地方(见语法)。

类型定义："bool"/"tristate"/"string"/"hex"/"int"
每个配置选项都必须指定类型。有两个基本类型：tristate 和 string，其他类型都是基于这两个基本
类型。类型定义可以用输入提示，所以下面的两个例子是等价的：

```
bool "Networking support"
```

和

```
bool
prompt "Networking support"
```

输入提示： "prompt" <prompt> ["if" <expr>]
每个菜单选项最多只能有一个显示给用户的输入提示。可以用 "if" 来表示该提示的依赖关系，当然这是可选的。

默认值："default" <expr> ["if" <expr>]
一个配置选项可以有任意多个默认值。如果有多个默认值，那么只有第一个被定义的值是可用的。默认值并不是只限于应用在定义他们的菜单选项。这就意味着默认值可以定义在任何地方或被更早的定义覆盖。
如果用户没有设置(通过上面的输入提示)，配置选项的值就是默认值。如果可以显示输入提示的话，就会把

默认值显示给用户，并可以让用户进行修改。
默认值的依赖关系可以用 "if" 添加。(可选项)

依赖关系："depends on"/"requires" <expr>
为一菜单选项定义依赖关系。如果定义了多个依赖关系，它们之间用 '&&' 间隔。
依赖关系也可以应用到
该菜单中所有的其它选项(同样接受 if 表达式)，所以下面的两个例子是等价的：

bool "foo" if BAR
default y if BAR

depends on BAR
bool "foo"
default y

- 反向依赖关系："select" <symbol> ["if" <expr>]
尽管普通的依赖关系可以降低一选项的上限，反向依赖能将这一限制降的更低。
当前菜单选项的值是 symbol
的最小值。如果 symbol 被选择了多次，上限就是其中的最大值。
反向依赖只能用在 boolean 或 tristate 选项上。

- 数据范围："range" <symbol> <symbol> ["if" <expr>]
为 int 和 hex 类型的选项设置可以接受输入值范围。用户只能输入大于等于第一个 symbol，小于等于第二个
symbol 的值。

- 帮助信息： "help" or "---help---"
定义一帮助信息。帮助信息的结束就由缩进的水平决定的，这也就意味着信息是在第一个比帮助信息开始行
的缩进小的行结束。
"---help---" 和 "help" 在实现的作用上没有区别，"---help---" 有助于将文件中的配置逻辑与给开发人员的提示分开。

菜单依赖关系

依赖关系决定了菜单选项是否可见，也可以减少 tristate 的输入范围。tristate 逻辑比 boolean 逻辑在表达式中用更多的状态(state)来表示模块的状态。依赖关系表达式的语法如下：

<expr> ::= <symbol> (1)
<symbol> '=' <symbol> (2)

<symbol> '!=' <symbol> (3)
'(' <expr> ')' (4)
'!' <expr> (5)
<expr> '&&' <expr> (6)
<expr> '||' <expr> (7)

表达式是以优先级的降序列出的。

(1) 将 symbol 赋给表达式。boolean 和 tristate 类型的 symbol 直接赋给表达式。所有其它类型的 symbol
都赋 'n'。
(2) 如果两个 symbol 相等，返回'y'，否则为'n'。
(3) 如果两个 symbol 相等，返回'n'，否则为'y'。
(4) 返回表达式的值。用于改变优先级。
(5) 返回 (2-/expr/) 的结果。
(6) 返回 min(/expr/,/expr/) 的结果。
(7) 返回 max(/expr/,/expr/) 的结果。

一个表达式的值可以是'n'，'m'或'y'(或者是计算的结果 0,1,2）。当表达式的值为'm'或'y'的时候，菜
单项才是可见的。

symbol 有两种类型：不可变的和可变的。不可变的 symbol 是最普通的，由'config'语句定义，完全由数字
、字母和下划线组成(alphanumeric characters or underscores)。
不可变的 symbol 只是表达式的一部分。经常用单引号或双引号括起来。在引号中，可以使用任何字符，使用引
号要用转义字符'\'。

菜单结构

菜单在树中的位置可由两种方法决定。第一种可以是这样：

```
menu "Network device support"
depends NET

config NETDEVICES
...

endmenu
```

所有的在"menu" ... "endmenu" 之间都是"Network device support"的子菜单。所有的子菜单选项

都继承了父菜单的依赖关系，比如，"NET"的依赖关系就被加到了配置选项 NETDEVICES 的依赖列表中。

还有就是通过分析依赖关系生成菜单的结构。如果菜单选项在一定程度上依赖于前面的选项，它就能成为该选
项的子菜单。首先，前面的(父)选项必须是依赖列表中的一部分并且它们中必须有满足下面两个条件的选项：
- 如果父选项为'n'，子选项必须不可见。
- 如果父选项可见，子选项才能可见。

```
config MODULES
bool "Enable loadable module support"

config MODVERSIONS
bool "Set version information on all module symbols"
depends MODULES

comment "module support disabled"
depends !MODULES
```

MODVERSIONS 直接依赖 MODULES，这就意味着如果 MODULES 不为'n'，该选项才可见。换句话说，当
MODULES 可见时，选项才可见(MODULES 的(空)依赖关系也是选项依赖关系的一部分)。

Kconfig 语法

配置文件描述了菜单选项，每行都是以一关键字开头(除了帮助信息)。下面的关键字结束一菜单选项：
- config
- menuconfig
- choice/endchoice
- comment
- menu/endmenu
- if/endif
- source

前 5 个同样可以用在菜单选项定义的开始。

config:

```
"config" <symbol>
<config options>
```

定义了一配置选项 <symbol> 并且可以接受任何前面介绍的属性。

menuconfig:
"menuconfig" <symbol>
<config options>

此关键字和前面的关键字很相似，但它在前面的基础上要求所有的子选项作为独立的行显示。(This is
similar to the simple config entry above, but it also gives a hint to front
ends, that all suboptions should be displayed as a separate list of options.)

choices:

"choice"
<choice options>
<choice block>
"endchoice"

该关键字定义了一组选择项，并且选项可以是前面描述的任何属性。尽管 boolean 只允许选择一个配置选项，
tristate 可以抒多个配置选项设为'm'，但选项只能是 boolean 或 tristate 类型。这可以在一个硬件有多
个驱动的情况下使用，最终只有一个驱动被编译进/加载到内核，，但所有的驱动都可以编译成模块。
选项可以接受的另一个选项是"optional"，这样选项就被设置为'n'，没有被选中的。

comment:

"comment" <prompt>
<comment options>

这里定义了在配置过程中显示给用户的注释，该注释还将写进输出文件中。唯一可用的可选项是依赖关系。

menu:

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

这里定义了一个菜单，详细信息请看前面的"菜单结构"。唯一可用的可选项是依赖关系。

if:

"if" <expr>
<if block>
"endif"

这里定义了 if 结构。依赖关系<expr>被加到所有在 if ... endif 中的菜单选项中。