通用Makefle的解析

执行 make 命令时，它会去当前目录下查找名为“Makefile”的文件，并根据它的指示去执行操作，生成第一个目标。

- 我们可以使用“-f”选项指定文件，不再使用名为“Makefile”的文件， 比如：

```
make -f Makefile.build
```

- 我们可以使用“-C”选项指定目录，切换到其他目录里去，比如：

```
make -C a/ -f Makefile.build
```

- 我们可以指定目标，不再默认生成第一个目标：

```
make -C a/ -f Makefile.build other_target
```

- 变量的导出(export)

```
export CFLAGS LDFLAGS
#使其他Makefile能用到前面已经执行的Makefile的变量，即其他Makefile由导出变量的Makefile调用执行
```

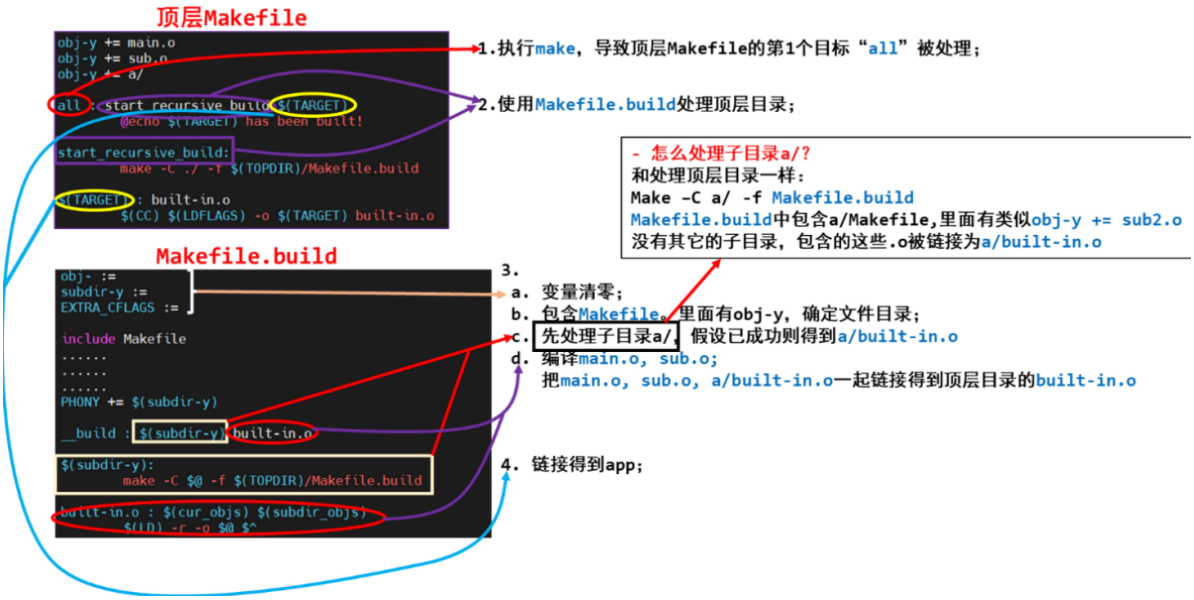
- Makefile 中可以使用 shell 命令：

```
TOPDIR := $(shell pwd)
#这是个立即变量，TOPDIR 等于 shell 命令 pwd 的结果。
```

- 在 Makefile 中怎么放置第 1 个目标：
  执行 make 命令时如果不指定目标，那么它默认是去生成第 1 个目标。 所以“第 1 个目标”，位置很重要。有时候不太方便把第 1 个目标完整地放 在文件前面，这时可以在文件的前面直接放置目标，在后面再完善它的依赖与 命令。比如：

```
First_target: # 这句话放在前面
 # 其他代码
First_target : $(xxx)  #在后面来完善
    $(command)
```

调用流程


顶层Makefile
obj-y += main.o
obj-y += sub.o
obj-y += a/
all : start_recursive_build $(TARGET)
@echo $(TARGET) has been built!
start_recursive_build:
make -C ./ -f $(TOPDIR)/Makefile.build
$(TARGET) : built-in.o
$(CC) $(LDFLAGS) -o $(TARGET) built-in.o
1.执行make，导致顶层Makefile的第1个目标“all”被处理；
2.使用Makefile.build处理顶层目录；
- 怎么处理子目录a/?
和处理顶层目录一样：
Make -C a/ -f Makefile.build
Makefile.build中包含a/Makefile,里面有类似obj-y += sub2.o
没有其它的子目录，包含的这些.o被链接为a/built-in.o
Makefile.build
obj- :=
subdir-y :=
EXTRA_CFLAGS :=
include Makefile
.....
.....
.....
PHONY += $(subdir-y)
__build : $(subdir-y) built-in.o
$(subdir-y):
make -C $@ -f $(TOPDIR)/Makefile.build
built-in.o : $(cur_objs) $(subdir_objs)
$(LD) -r -o $@ $^
3.
a. 变量清零；
b. 包含Makefile，里面有obj-y，确定文件目录；
c. 先处理子目录a/，假设已成功则得到a/built-in.o
d. 编译main.o, sub.o;
把main.o, sub.o, a/built-in.o一起链接得到顶层目录的built-in.o
4. 链接得到app;