Makefile工程管理

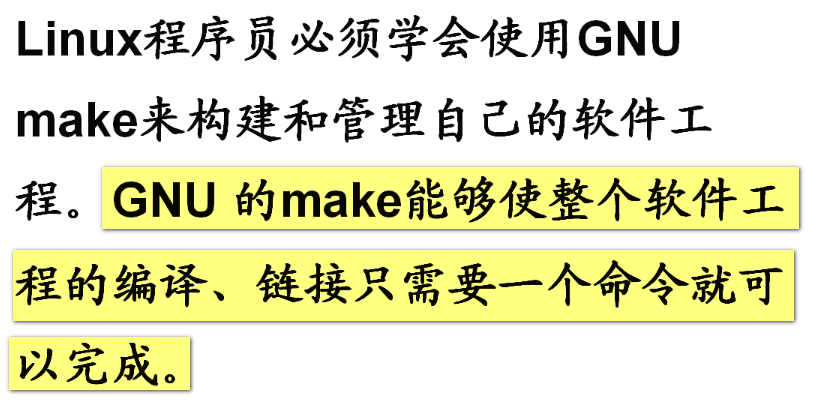


Makefile的工程管理

Make工具是GNU提供的非常有效管理项目工程的工具

前面编译应用程序的时候 编译的源文件只有一个

但是 **软件工程比较大的时候** 程序文件有成千上万个



**2.4.20内核的源文件 有1万个(包括头文件)**

2.6内核 大概2万个

**如果还用GCC手工编译 链接 这么多 不知道搞到什么年月 ----- 一般采用make管理工具 对源代码进行统一的管理 完成编译和连接的工作**

 【**GCC就是javac ----- 这个makefile就又变成了maven的声明式的编译 打包等操作**】

他怎么知道我的这些代码怎么进行链接 动态 静态 哪些头文件 目录

**Make怎么知道如何处理这些问题 【javac-gcc maven-make pom-makefile】**

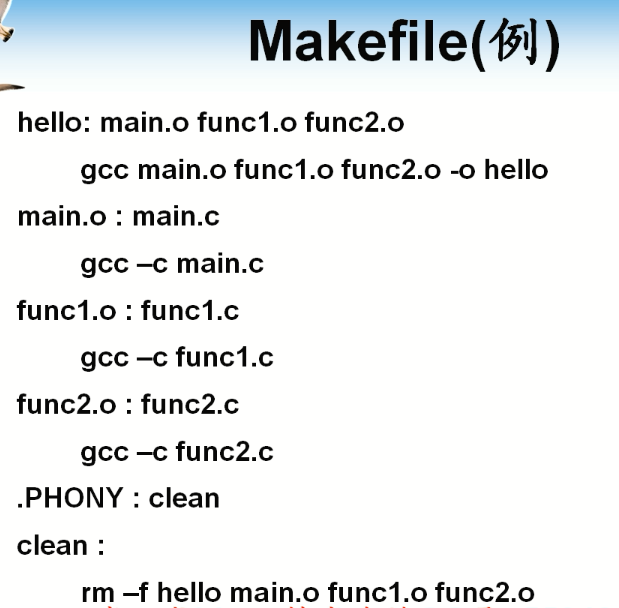
**这个时候 提供一个信息 Makefile文件 --- 按照Makefile文件** 对程序进行编译 链接的操作 ---- make按照make file 对文件进行编译和连接操作

对我们来说 写Makefile 按照Makefile的规则写这个文件 make就能解析

【类似于Dockerfile Ant Maven的pom.xml】

这个**目标是得到hello的应用程序**

---- 这个应用程序 **又有三个源文件构成** === 这个是背景



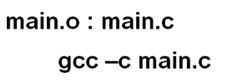
**main.c, func1.c和func2.c**

看一下这个makefile ---- 这个makefile指明了我们的应用程序最后的目标应用程序名就是hello

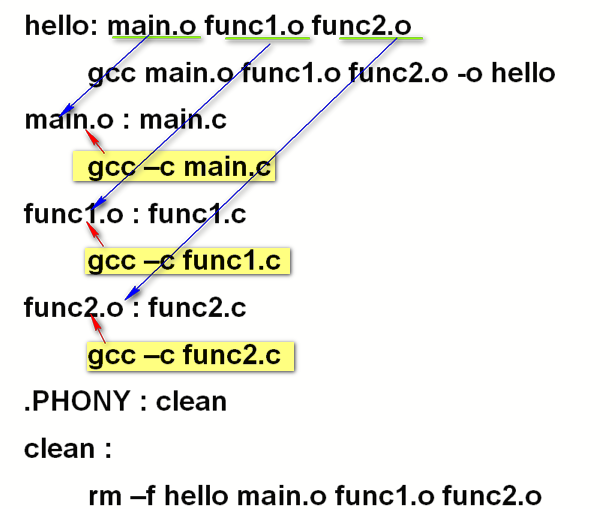


**hello是最终要编译 最后连接出来的应用程序的名字** +所需要的main.o func1.o和func2.o ---- ==== 并指出了 main.o func1.o和fun2.o是如何声称的

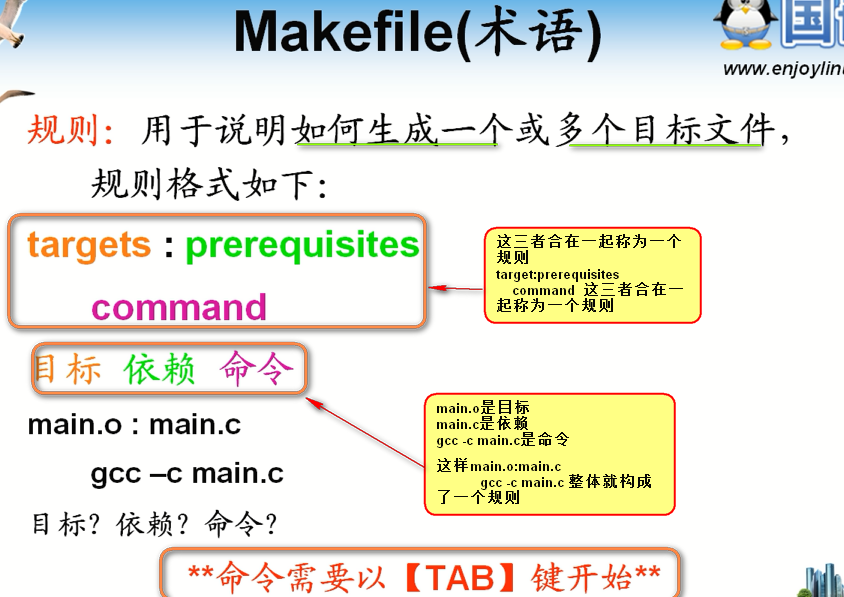
然后



main.o:main.c 说明的是main.o来源于main.c --- 通过何种方式呢 下面一行就是告诉通过gcc –c main.c



makefile中的术语



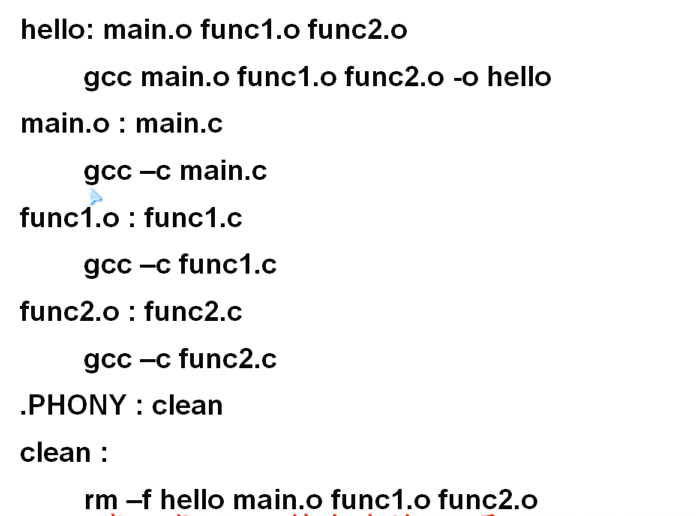
这样一条规则就说出了 我这个目标是如何产生的

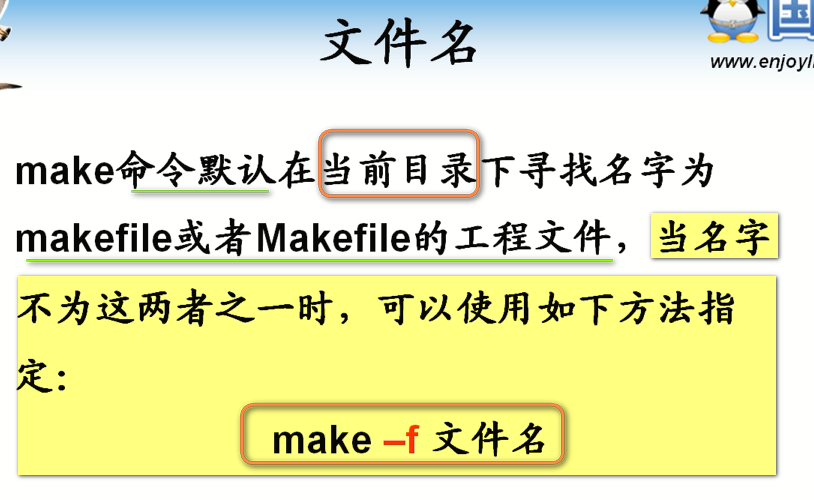
目标就是通过这条规则得到什么 这条规则要输出的是什么

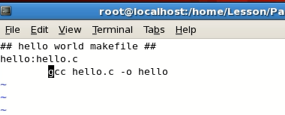
依赖就是 要得到这个目标 需要什么原材料

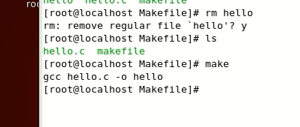
命令 就是 有了这些原材料怎么得到目标 这个就是命令

Main.c通过 命令 gcc –c main.c就会得到main.o

 这里面有四条规则 就有四个目标 ======= **最终的目标 就是第一条规则的目标 就是我们的最终的目标**

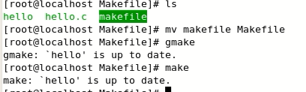


 这个makefile中就有一条规则 目标是hello 依赖是hello.c 命令就是后面的



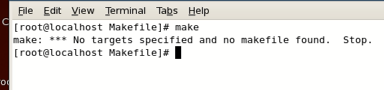
这个使用make命令之后 就得到了hello这个目标

默认情况下 make命令会找MAKEFILE



再次修改一个名字 makefile修改为makefile1

执行make命令



就会报错

找不到这个makefile这个文件

要么修改能识别的名字

要么使用-f



**只有命令 没有依赖的目标 就成为伪目标**



clean:

rm –f hello main.o func1.o func2.o 就是一个伪目标 没有依赖

然后使用 .PHONY 把这个clean这个目标声明为伪目标

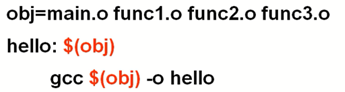
这个后面没有依赖 这个动作就是把编译出来的东西都给删掉

Makefile中可以使用变量



**看第二个答案**

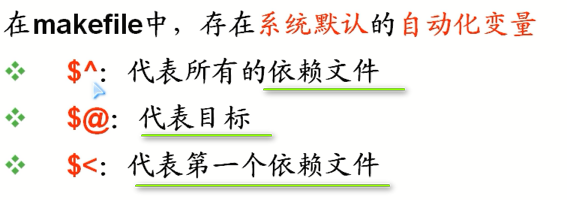
**obj = main.o func1.o func2.o func3.o**

 这个仅仅有一行要修改

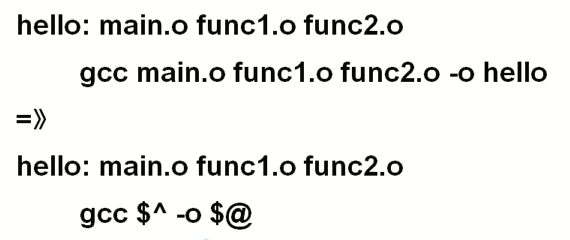
如果makefile非常多 那么 每一行都需要修改

有了变量 再添加func3.o func4.o直接在obj后面添加

**这就是变量的好处 ---- 不需要定义变量的类型 直接使用$()使用就可以了**



进行改写



这样就更简单了

编写Makefile的时候 使用命令的时候 不是敲入8个空格 –而是一个tab按键





加上@符

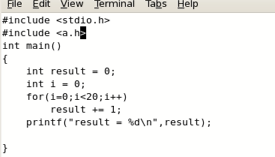


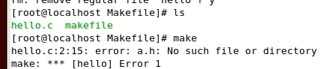
这样 就不会显示这个编译的过程 ----- 没有显示编译的命令的 ---取消了屏幕的回显

编写内核的时候 如果不取消这些回显 会满屏都是 所以取消

**如果编译的时候出错了**

搞一个例子





**即时是回显关闭 出错也会被告知**