

万里红客户端 Makefile 简便使用方法

万里红中央研究院

2022 年 3 月 5 日

目录

万里红客户端 Makefile 简便使用方法.....	1
1 前言	3
2 如何更好的使用 Makefile	3
3 简化 Makefile 的好处.....	5
4 总结	7

1 前言

在 Linux 下开发软件时，Makefile 的作用和重要性不言而喻，可以熟练使用 Makefile 是 Linux 下程序员的基本功之一，是必须要掌握的一项重要技能。

Makefile 虽然功能强大，但要从 0 开始写一个 Makefile 还是挺不容易的，需要经过大量学习才能熟练掌握。在真实的使用场景中，几乎没有人会从 0 开始写 Makefile，绝大多数人都是找一个现有的 Makefile 改一下，然后就使用到自己的项目中了。

我一向认为，一个工具应该一看就会，如果需要使用者投入大量的学习成本才能学会使用的话，那这工具就不能称之为一个好工具。对大多数用户来说，其实需要的不是一个“最强大”的工具，而是一个“最合适”的工具。对于 make 这样的工具，如果把大量时间花在编写 Makefile 上，这成本显然是不低的。

作为工具提供商，确实会提供“强大”的工具，但作为用户，为了提高效率，我们可以对工具进行改造，通过一些手段，把这个强大的工具变得更容易用。

2 如何更好的使用 Makefile

如果把公司里能收集到的所有 Makefile 放到一起看就会发现，大多数 Makefile 文件都差不多，尤其是编译参数、链接参数、头文件包含路径、Lib 的路径等，基本上都是一样的。真正不一样的地方，主要集中在：**需要编译的源文件、编译之后生成物名称以及类型（可执行程序或者动态库）这几个地方**。很显然，既然只有这几处不一样，那我们为什么要每次都写这些一样的内容呢？我们可以把一样的内容全部抽取出来放在一个公共的位置，大家直接包含这些公共的文件就可以大幅减少配置 Makefile 的工作量，基于这个思路，我们提供了几个模板，用于帮助大家快速配置 Makefile。

先看一下我们目前使用的典型的 Makefile 文件：

```
1  # ++++++ 这里两行东西，二选一 ++++++ #
2
3  # 如果是动态库，包含这个文件
4  include $(WLH_PUBLIC_LIB_ROOT)/1.automation/linux/makefile.dep/makefile.template.common.dll.dep
5
6  # 如果是可执行程序，包含这个文件
7  # include $(WLH_PUBLIC_LIB_ROOT)/1.automation/linux/makefile.dep/makefile.template.common.exe.dep
8
9  # ----- 上面两行东西，二选一 ----- #
10
11
12 # ++++++ 这条线下面的内容，一般需要修改一下 ++++++ #
13
14 # 当前工程的目录列表( 把需要编译的目录，添加到这个文件中去 )
15 include Makefile.this.project.info.dep
16
17 # ++++++ 这个放到最后，一般不用动 ++++++ #
18
19 include $(WLH_PUBLIC_LIB_ROOT)/1.automation/linux/makefile.dep/makefile.template.final.dep
20
21 # ----- 这个放到最后，一般不用动 ----- #
22
```

从这个 Makefile 中能看出，真正需要我们参与修改的，只有头上两个包含文件里二选一的部分，以及 Makefile.this.project.info.dep 这个文件了(文件名随意，之所以头上加上 Makefile. 前缀，是为了让这个文件在目录中能跟 Makefile 在一起，找起来方便)，很显然对于这样一个 Makefile 来说，几乎没有学习成本。

我们再看一下 this.project.info.dep 文件中包含哪些内容：

```
1  # 输出的模块名称
2  OUTPUT_NAME = wlh_foundation.so
3
4  # 输出的文件全路径
5  TARGET = ./output/$(OUTPUT_NAME)
6
7  # 需要额外添加的包含目录
8  HEADER = -I./ \
9  |      -I./3rdparty/include/ \
10
11 DIR := \
12 |   . \
13 |   3rdparty_wrapper/curl \
14 |   3rdparty_wrapper/openssl \
15 |   base \
16 |   base/async \
17 |   base/base_info \
18 |   base/base_info/linux \
19 |   base/container \
20 |   base/data \
21 |   base/kv \
22 |   base/platform/linux \
23 |   base/text \
```

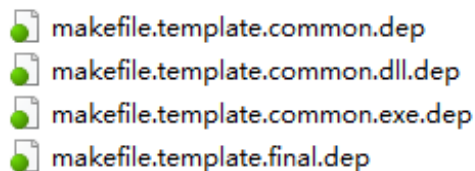
这个文件内容一目了然，主要就设置了输出文件名称，输出文件路径，以及本工程中额外需要添加的一些头文件的包含路径（如果不需要可以不加），最后是本工程中需要编译的各子目录的名称。

各种复杂的 Makefile 的内容，都在这几个 include 中完成：

```
# 如果是动态库，包含这个文件
include $(WLH_PUBLIC_LIB_ROOT)/1.automation/linux/makefile.dep/makefile.template.common.dll.dep

# 如果是可执行程序，包含这个文件
# include $(WLH_PUBLIC_LIB_ROOT)/1.automation/linux/makefile.dep/makefile.template.common.exe.dep
```

我们打开 `$(WLH_PUBLIC_LIB_ROOT)/1.automation/linux/makefile.dep/` 这个目录可以看到，目录下面有几个 Makefile 的模板，各 Makefile 中相同的内容，都放在这些文件里。



- makefile.template.common.dep
- makefile.template.common.dll.dep
- makefile.template.common.exe.dep
- makefile.template.final.dep

3 简化 Makefile 的好处

此处主要分享一下为什么要花时间对 Makefile 做这样的简化，以及这样简化的意义是什么，具体列举如下：

- 没有学习成本，使用简单

对于工具来说，使用简单，没有学习成本是很重要的，好工具应该让人一看就知道怎么用，拿起来就能用好，这一点微软的 Visual Studio 做的就比较出色。

使用简单不但意味着上手快，还意味着不容易出错。作为程序员我们知道，在项目中真正花在编码的时间其实并不多，大量的时间花在排除各种问题，修复各种 Bug、以及由于试错环节中反复折腾下来的时间上。一个东西越简单，需要人工参与的环节越少，就越不容易出错，因此就能节约出大量的时间来，效率更高。

- **便于统一处理问题**

从我职业生涯开始，每当有人要从我这拷贝代码的时候，我都会特别担心，万一我这份原始代码里有 Bug，别人把这些 Bug 也一起拷贝走，那岂不是会害的别人的产品也出现问题？随着时间的推移，如果代码被越来越多的人拷贝，那这个 Bug 就会扩散。如果某一天原始代码中真的被发现一个 Bug，那所有人都会因为这个 Bug 的存在而投入同样多的时间去修复，而显然这些时间和资源的浪费是毫无意义的。

事实上我们也无法保证写出来的代码一定没有问题，所以我们需要提前做些事，尽可能让这种可能出现的问题在我们可控制的范围中。

基于这个原因，我极其反对程序员通过复制粘贴的方式来编码，对于重复的代码，对内应该封装成函数，对外应该封装成动态库，以接口的方式输出，这样万一有 Bug，只需要改一处地方，修复的成本大幅降低，虽然前期会多花一点点的时间，但避免了后期十倍、百倍时间的浪费。

前面曾经提到，基本上所有人写 Makefile 的时候，都是拷贝一个现有的 Makefile 过来，简单改一下就投入使用。我们认为 Makefile 文件等同于代码，万一初始的 Makefile 里有问题，那基于这个 Makefile 修改的所有 Makefile 都会有同样的问题。又或者某一天由于某种原因，需要对公司所有的模块，编译的时候都要加上一个特殊的编译选项，按照目前的做法，所有的 Makefile 文件都需要修改，不但很费事，而且容易漏改。而现在这种简便方式中，所有的复杂信息都在公共库提供的模板里，所有 Makefile 都在包含这个模板，那万一真出现问题，只需要修改模板即可，所有人都不需要再修改了，这显然会方便很多。

事实上目前确实无法保证 Makefile 的模板里的各种编译参数一定是最合理的，但这不是什么大问题，如果有需要，随时调整就行。

4 总结

我们在工作中，应该随时关注效率问题，避免重复性的工作，工作中的任何重复性的事情，都是我们可以改进的地方，只要是重复性的事情，一般来说都可以通过自动化的手段，用较小的成本带来效率的大幅提升。

从我一开始接触 Makefile 就觉得，程序员把时间浪费编写 Makefile 是无意义的，哪怕花 10 分钟编写 Makefile 都是浪费时间，编译一个工程应该是一件简单的事。

事实上行业内大部分公司的程序员，在重复性工作的事情上都浪费了大量的时间，很多时候其实大家都知道这些事情是重复性的，效率很低，做起来也很郁闷，但大部分人往往会凑合着过。以 Makefile 为例，大部分程序员会觉得，我弄这些公共模板的时间，这个 Makefile 早就改完了，还不如直接拷贝一个过来改算了。对于单次的 Makefile 编写来说确实如此，但如果从公司整体来看，会有几百甚至几千个 Makefile，每个 Makefile 由不同的人在维护，都有可能出错，这么一计算可能浪费在 Makefile 上的时间就很可观了。Makefile 如此，平时我们写程序过程中的各环节也类似，因此平时我们一定要避免复制粘贴，在编码过程中避免出现重复的代码，有重复的代码应该封装成函数，集中收口在一个地方，便于统一修改和维护。有别的同事要用的时候，一定不要拷贝代码，而应该提供接口，这样才有利于提高公司整体的效率。

行业内大多数公司的程序员，研发效率其实都不高，这其实是我们的机会所在，如果我们公司能在各种细节上提高效率，那就能在同样的时间内做更多的事，并且能提高产品质量，而产品质量的提高又能节约更多的时间，更有效率，我们就能比同行做的更快更好。

最后希望大家也能在这些过程中感受到编码的乐趣，在工作中得到能力的提升，并且让工作变得更加轻松愉快。