GCC源码分析(一) — 输入参数解析

系列文章:

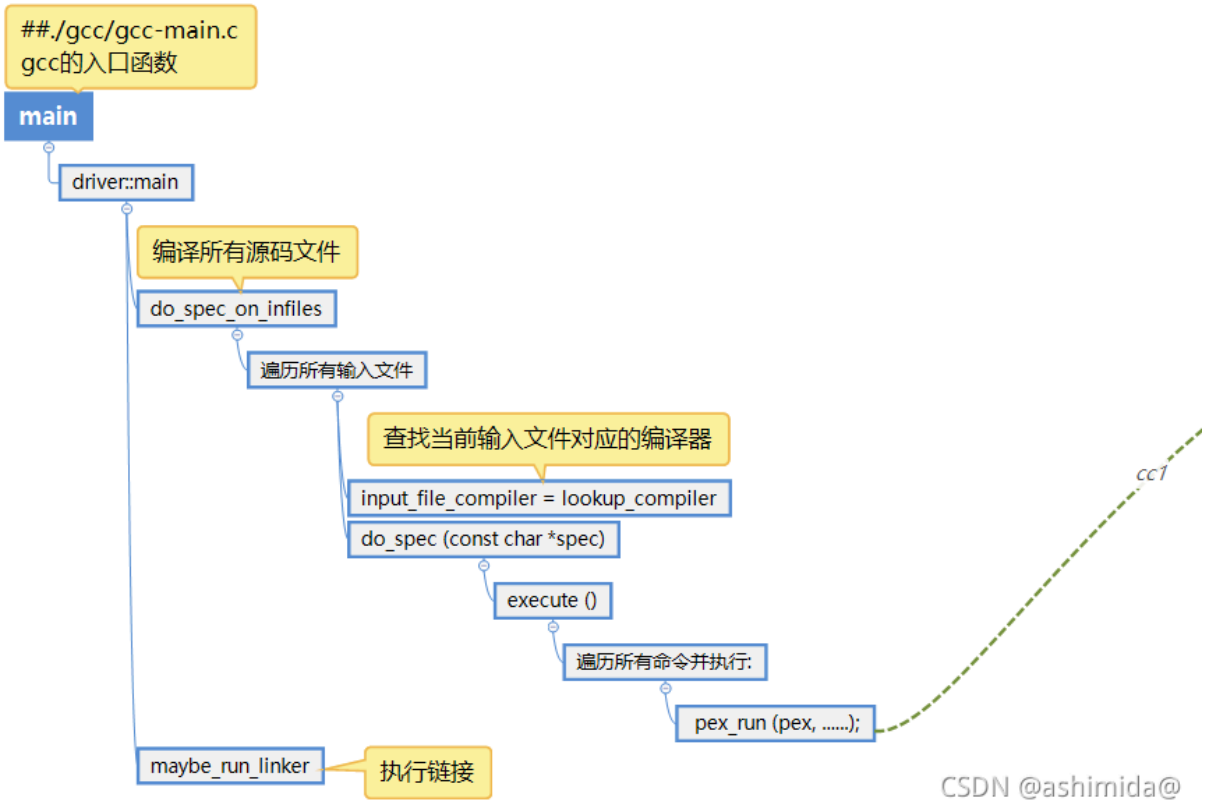
只以编译阶段为例的话，gcc的参数解析分为两部分:

- 一部分是参数从命令行到gcc
- 一部分是参数从gcc到最终的编译程序cc1

---

1. 从命令行到gcc

编译出来的gcc二进制程序实际上只是起到一个桩代码的作用，或者说只是一个编译的驱动，其内部实际上是要分别调用cc1,as,collect来进行编译，汇编与连接。gcc的基本流程如下图:



gcc在解析了输入参数后，会为每一个输入文件启动一次cc1来做具体的编译，而cc1一次只能处理一个源码的编译，如果输入多个文件则会报错，如:

```
$ ./cc1 main.c asm.c

cc1: error: too many filenames given.  Type cc1 --help for usage
```

gcc和cc1的参数解析实际上用的是同一套代码，其首先依赖于编译期间生成的用户参数数组:

```
#./build-gcc/gcc/options.c      //注意这个是gcc编译的输出目录，此文件是编译期间生成的

const struct cl_option cl_options[] =

{

/* [0] = */ {

    "-###",

    NULL,

    NULL,
```

```
14.
15.     NULL,
16.
17.     NULL, NULL, N_OPTS, N_OPTS, 3, /* .neg_idx = */ -1,
18.
19.     CL_DRIVER,
20.
21.     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
22.
23.     (unsigned short) -1, 0, CLVC_BOOLEAN, 0, -1, -1 },
24.
25. /* [1] = */ {
26.
27.     "--all-warnings",
28.
29.     NULL,
30.
31.     NULL,
32.
33.     NULL,
34.
35.     NULL, NULL, OPT_Wall, N_OPTS, 13, /* .neg_idx = */ -1,
36.
37.     CL_Ada | CL_AdaSCIL | CL_AdaWhy | CL_C | CL_CXX | CL_Fortran | CL_ObjC | CL_ObjCXX | CL_WARNING,
38.
39.     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
40.
41.     (unsigned short) -1, 0, CLVC_BOOLEAN, 0, -1, -1 },
42.
43. ......
```

这个数组共有上千项，其中记录了在gcc和cc1中所有可用的参数，参数分为以下几大类:

```
1. ##./build-gcc/gcc/options.h    //具体语言相关的参数
2.
3. #define CL_Ada        (1U << 0)
4.
5. #define CL_AdaSCIL    (1U << 1)
6.
7. #define CL_AdaWhy     (1U << 2)
8.
9. #define CL_BRIG       (1U << 3)
10.
11. #define CL_C          (1U << 4)
12.
13. #define CL_CXX        (1U << 5)
14.
15. #define CL_D          (1U << 6)
16.
17. #define CL_Fortran    (1U << 7)
18.
19. #define CL_Go         (1U << 8)
20.
21. #define CL_LTO        (1U << 9)
22.
23. #define CL_ObjC       (1U << 10)
24.
25. #define CL_ObjCXX     (1U << 11)
26.
27. #define CL_LANG_ALL   ((1U << 12) - 1)
28.
29.
30. ##./gcc/opts.h //非语言相关的参数
31.
32. #define CL_PARAMS               (1U << 16) /* Fake entry.  Used to display --param info with --help.  */
33.
34. #define CL_WARNING        (1U << 17) /* Enables an (optional) warning message.  */
35.
36. #define CL_OPTIMIZATION        (1U << 18)  //优化相关的参数
37.
38. #define CL_DRIVER        (1U << 19)  //仅供gcc使用的参数
39.
40. #define CL_TARGET        (1U << 20) //目标平台特有参数
41.
42. #define CL_COMMON        (1U << 21) //通用参数
```

数组中cl_optons[n].flags 为CL_DRIVER的参数，即为gcc driver使用的参数，这样的参数如 -Wl, --entry等，除了CL_DRIVER外，其他的参数基本都是要传给cc1的。

---

2. 从gcc到cc1

在cc1中参数解析一共经历了三个阶段:

1. argvs => save_decoded_options

2. save_decoded_options =>global_options等真正影响编译的flags

3. 对global_options等参数的处理

其中:

1. argvs => save_decoded_options

save_decoded_options[]数组中的每一个元素都是一个cl_decoded_option结构体，其记录了输入参数的字符串，参数值，与此参数相关的cl_options数组索引等，此索引与枚举类型

opt_code的下标一一对应。

```
1. struct cl_decoded_option
2.
3. {
4.
5.   size_t opt_index;            //代表当前参数的类型的index,也是此参数在 cl_options 数组中的索引,如 OPT_SPECIAL_input_file 代表此参数是输入文件名
6.
7.   const char *warn_message;    //使用此参数是否要先提示个warning信息
8.
9.   const char *arg;             //解析出的此参数的具体字符串，如"-fdump-passes"
10.
11.  const char *orig_option_with_args_text;
12.
13.  const char *canonical_option[4];
14.
15.  size_t canonical_option_num_elements;
16.
17.  HOST_WIDE_INT value;         //参数的值，对于大部分没有值的参数会是0/1
18.
19.  int errors;                  //代表检查出此参数设置错误
20.
21. };
22.
23. struct cl_decoded_option *save_decoded_options;
24.
25. unsigned int save_decoded_options_count;            //save_decoded_options数组的元素个数
```

这一步实际上是在decode_cmdline_options_to_array中完成的，其调用路径为:

```
## toplev::main => decode_cmdline_options_to_array_default_mask => decode_cmdline_options_to_array
```

最终实际上调用的函数展开后是:

```
decode_cmdline_options_to_array(argc, argv, CL_C|CL_COMMON|CL_TARGET, &save_decoded_options, &save_decoded_options_count);
```

2.save_decoded_options =>global_options等真正影响编译的flags

save_decoded_options只是用来记录输入参数相关信息的一个数组，输入参数最终都是以flag的形式在编译过程中起作用的，而global_options中则记录了大部分参数相关的flags，这一步的作用就是将所有参数翻译为各种flags.

这一步是在decode_options中完成的，其调用路径为: toplev::main => decode_options

decode_options的参数如下，其将saved_decoded_options中的大部分参数都转化为global_options中的flags

```
1. decode_options (&global_options, &global_options_set, save_decoded_options, save_decoded_options_count,
2.   UNKNOWN_LOCATION, global_dc, targetm.target_option.override);
```

此外handle_common_deferred_options ();函数中会对部分标记为CLVC_DEFER的参数做统一的解析:

```
## toplev::main =>handle_common_deferred_options
```

3.对global_options等参数的处理

这里包括了对要解析的编译单元文件的读取(=>parse_in),检查输入文件个数等，其他后续补充:

```
## toplev::main => do_compile => process_options
```