

[专栏首页](#) [三木的博客](#) [QEMU 2: 参数解析](#)

10

0

分享

# QEMU 2: 参数解析

2018-02-07 阅读 370

## #一、使用gdb分析QEMU代码#

使用gdb不仅可以很好地调试代码，也可以利用它来动态地分析代码。使用gdb调试QEMU需要做一些准备工作：

- 1, 编译QEMU时需要在执行configure脚本时的参数中加入--enable-debug。
- 2, 从QEMU官方网站上下载一个精简的镜像——linux-0.2.img。linux-0.2.img只有8MB大小，启动后包含一些常用的shell命令，用于QEMU的测试。

```
$wget http://wiki.qemu.org/download/linux-0.2.img.bz2
$gzip2 -d ./linux-0.2.img.bz2
```

- 3, 启动gdb调试QEMU：

```
gdb --args qemu-system-x86_64 -enable-kvm -m 4096 -smp 4
linux-0.2.img
```

-smp指定处理器个数。

## #二、参数解析用到的数据结构#

QEMU系统模拟的主函数位于vl.c文件，无论是qemu-system-x86\_64还是qemu-system-ppc64，都是从vl.c中的main函数开始执行。下面先介绍main函数涉及到的一些数据结构。

### ###QEMU链表

QEMU的链表在include/qemu/queue.h文件中定义，分为四种类型：

单链表(singly-linked list)：单链表适用于大的数据集，并且很少有删除节点或者移动节点的操作，也适用于实现后进先出的队列。

链表(list)：即双向链表，除了头节点之外每个节点都会同时指向前一个节点和后一个节点。

简单队列(simple queue)：简单队列类似于单链表，只是多了一个指向链表尾的一个表头，插入节点的时候不仅可以像单链表那样将其插入表头或者某节点之后，还可以插入到链表尾。

尾队列(tail queue)：类似于简单队列，但节点之间是双向指向的。

这里不一一介绍各种链表的用法，只通过NotifierList的定义来说明QEMU链表(list)的用法。在main函数的开头定义的DisplayState结构体使用到了NotifierList，NotifierList就用到了链表。

### a. 表头及节点的定义

定义表头需要用到QLIST\_HEAD，定义如下：

```
86 #define QLIST_HEAD(name, type) \
87 struct name { \
88     struct type *lh_first; /* first element */ \
89 }
```

NotifierList就采用了QLIST\_HEAD来定义表头：

```
27 typedef struct NotifierList
28 {
29     QLIST_HEAD(, Notifier) notifiers;
30 } NotifierList;
```

## 作者介绍



用户1214695

[关注](#)[专栏](#)

文章	阅读量	获赞	作者排名
50	16.1K	348	2265

## 精选专题

[云+社区×知乎「AI与传统…](#)[AI 具有什么能力？能给传统行业带来哪些变革与发展？](#)

## 活动推荐

[腾讯云自媒体分享计划](#)

入驻云加社区，共享百万资源包。

[立即入驻](#)[邀请作者加入自媒体计划](#)

每月最高可拿1800元无门槛代金券。

[了解更多](#)[运营活动](#)

```
94 #define QLIST_ENTRY(type) \
95 struct { \
96     struct type *le_next; /* next element */ \
97     struct type **le_prev; /* address of previous next element */ \
98 }
```

Notifier的节点定义如下：

```
21 struct Notifier
22 {
23     void (*notify)(Notifier *notifier, void *data);
24     QLIST_ENTRY(Notifier) node;
25 };
```

## b. 初始化表头

初始化表头用到QLIST\_INIT：

```
103 #define QLIST_INIT(head) do { \
104     (head)->lh_first = NULL; \
105 } while (/*CONSTCOND*/0)
```

初始化NotifierList就可以这样进行：

```
19 void notifier_list_init(NotifierList *list)
20 {
21     QLIST_INIT(&list->notifiers);
22 }
```

## c. 在表头插入节点

将节点插入到表头使用QLIST\_INSERT\_HEAD：

```
122 #define QLIST_INSERT_HEAD(head, elm, field) do { \
123     if (((elm)->field.le_next = (head)->lh_first) != NULL) \
124         (head)->lh_first->field.le_prev = &(elm)->field.le_next; \
125     (head)->lh_first = (elm); \
126     (elm)->field.le_prev = &(head)->lh_first; \
127 } while (/*CONSTCOND*/0)
```

插入Notifier到NotifierList：

```
24 void notifier_list_add(NotifierList *list, Notifier *notifier)
25 {
26     QLIST_INSERT_HEAD(&list->notifiers, notifier, node);
27 }
```

## d. 遍历节点

遍历节点使用QLIST\_FOREACH或者QLIST\_FOREACH\_SAFE，

QLIST\_FOREACH\_SAFE是为了防止遍历过程中删除了节点，从而导致le\_next被释放掉，中断了遍历。

```
147 #define QLIST_FOREACH(var, head, field) \
148     for ((var) = ((head)->lh_first); \
149          (var); \
150          (var) = ((var)->field.le_next)) \
151 \
152 #define QLIST_FOREACH_SAFE(var, head, field, next_var) \
153     for ((var) = ((head)->lh_first); \
154          (var) && ((next_var) = ((var)->field.le_next), 1); \
155          (var) = (next_var))
```

NotifierList在执行所有的回调函数时就用到QLIST\_FOREACH\_SAFE：

```
34 void notifier_list_notify(NotifierList *list, void *data)
35 {
36     Notifier *notifier, *next;
37 }
```

10

0

分享

```
40 }  
41 }
```

### ###Error和QError

为了方便的处理错误信息，QEMU定义了Error和QError两个数据结构。Error在qobject/qerror.c中定义：

```
101 struct Error  
102 {  
103     char *msg;  
104     ErrorClass err_class;  
105 };
```

包含了错误消息字符串和枚举类型的错误类别。错误类别有下面几个：

```
139 typedef enum ErrorClass  
140 {  
141     ERROR_CLASS_GENERIC_ERROR = 0,  
142     ERROR_CLASS_COMMAND_NOT_FOUND = 1,  
143     ERROR_CLASS_DEVICE_ENCRYPTED = 2,  
144     ERROR_CLASS_DEVICE_NOT_ACTIVE = 3,  
145     ERROR_CLASS_DEVICE_NOT_FOUND = 4,  
146     ERROR_CLASS_K_V_M_MISSING_CAP = 5,  
147     ERROR_CLASS_MAX = 6,  
148 } ErrorClass;
```

QEMU在util/error.c中定义了几个函数来对Error进行操作：

```
error_set //根据给定的ErrorClass以及格式化字符串来给Error分配空间并赋值  
error_set_errno //除了error_set的功能外，将指定errno的错误信息追加到格式化字  
error_copy //复制Error  
error_is_set //判断Error是否已经分配并设置  
error_get_class //获取Error的ErrorClass  
error_get_pretty //获取Error的msg  
error_free //释放Error及msg的空间
```

另外，QEMU定义了QError来处理更为细致的错误信息：

```
22 typedef struct QError {  
23     QObject_HEAD;  
24     Location loc;  
25     char *err_msg;  
26     ErrorClass err_class;  
27 } QError;
```

QError可以通过一系列的宏来给err\_msg及err\_class赋值：

```
39 #define QERR_ADD_CLIENT_FAILED \  
40     ERROR_CLASS_GENERIC_ERROR, "Could not add client"  
41  
42 #define QERR_AMBIGUOUS_PATH \  
43     ERROR_CLASS_GENERIC_ERROR, "Path '%s' does not uniquely identify an i  
44  
45 #define QERR_BAD_BUS_FOR_DEVICE \  
46     ERROR_CLASS_GENERIC_ERROR, "Device '%s' can't go on a %s bus"  
47  
48 #define QERR_BASE_NOT_FOUND \  
49     ERROR_CLASS_GENERIC_ERROR, "Base '%s' not found"  
...
```

Location记录了出错的位置，定义如下：

```
20 typedef struct Location {  
21     /* all members are private to qemu-error.c */  
22     enum { LOC_NONE, LOC_CMDLINE, LOC_FILE } kind;  
23     int num;
```

10

0

分享

### ###GMainLoop

QEMU使用glib中的GMainLoop来实现IO多路复用，关于GMainLoop可以参考博客[GMainLoop的实现原理和代码模型](#)。由于GMainLoop并非QEMU本身的代码，本文就不重复赘述。

### #三、QEMUOption、QemuOpt及QEMU参数解析

QEMU定义了QEMUOption来表示执行qemu-system-x86\_64等命令时用到的选项。在vl.c中定义如下：

```
2123 typedef struct QEMUOption {
2124     const char *name; //选项名，如 -device, name的值就是device
2125     int flags; //标志位，表示选项是否带参数，可以是0，或者HAS_ARG(值为0x1)
2126     int index; //枚举类型的值，如-device, 该值就是QEMU_OPTION_device
2127     uint32_t arch_mask; // 选项支持架构的掩码
2128 } QEMUOption;
```

vl.c中维护了一个QEMUOption数组qemu\_options来存储所有可用的选项，并利用qemu-options-wrapper.h和qemu-options.def来给该数组赋值。赋值语句如下：

```
2130 static const QEMUOption qemu_options[] = {
2131     {"h", 0, QEMU_OPTION_h, QEMU_ARCH_ALL},
2132 #define QEMU_OPTIONS_GENERATE_OPTIONS
2133 #include "qemu-options-wrapper.h"
2134     { NULL },
2135 };
```

#define QEMU\_OPTIONS\_GENERATE\_OPTIONS选择qemu-options-wrapper.h的操作，qemu-options-wrapper.h可以进行三种操作：

QEMU\_OPTIONS\_GENERATE\_ENUM: 利用qemu-options.def生成一个枚举值列表，  
QEMU\_OPTIONS\_GENERATE\_HELP: 利用qemu-options.def生成帮助信息并输出到  
QEMU\_OPTIONS\_GENERATE\_OPTIONS: 利用qemu-options.def生成一组选项列表

可以通过下面的方法来展开qemu-options-wrapper.h来查看上述操作的结果，以生成选项为例。

1. 在qemu-options-wrapper.h第一行写入#define QEMU\_OPTIONS\_GENERATE\_OPTIONS.
2. 执行命令 `gcc -E -o options.txt qemu-options-wrapper.h`
3. 查看文件options.txt即可

给qemu\_options数组赋值后，QEMU就有了一个所有可用选项的集合。之后在vl.c中main函数的一个for循环根据这个集合开始解析命令行。for循环的框架大致如下：

```
1  for(;;) {
2      if (optind >= argc)
3          break;
4      if (argv[optind][0] != '-') {
5          hda_opts = drive_add(IF_DEFAULT, 0, argv[optind++], HD_OPTS);
6      } else {
7          const QEMUOption *popt;
8
9          pop = lookup_opt(argc, argv, &optarg, &optind);
10         if (!(popt->arch_mask & arch_type)) {
11             printf("Option %s not supported for this target\n", popt->name);
12             exit(1);
13         }
14         switch(popt->index) {
15             case QEMU_OPTION_M:
16                 .....
17             case QEMU_OPTION_hda:
18                 .....
19             case QEMU_OPTION_watchdog:
```



10

0

分享

```
95 typedef struct QemuOptDesc {
96     const char *name;
97     enum QemuOptType type;
98     const char *help;
99 } QemuOptDesc;
```

子选项的类型可以是：

```
88 enum QemuOptType {
89     QEMU_OPT_STRING = 0, // 字符串
90     QEMU_OPT_BOOL,      // 取值可以是on或者off
91     QEMU_OPT_NUMBER,    // 数字
92     QEMU_OPT_SIZE,      // 大小，可以有K, M, G, T等后缀
93 };
```

QEMU维护了一个QemuOptsList\*的数组，在util/qemu-config.c中定义：

```
10 static QemuOptsList *vm_config_groups[32];
```

在main函数中由qemu\_add\_opts将各种QemuOptsList写入到数组中：

```
2944 qemu_add_opts(&qemu_drive_opts);
2945 qemu_add_opts(&qemu_chardev_opts);
2946 qemu_add_opts(&qemu_device_opts);
2947 qemu_add_opts(&qemu_netdev_opts);
2948 qemu_add_opts(&qemu_net_opts);
2949 qemu_add_opts(&qemu_rtc_opts);
2950 qemu_add_opts(&qemu_global_opts);
2951 qemu_add_opts(&qemu_mon_opts);
2952 qemu_add_opts(&qemu_trace_opts);
2953 qemu_add_opts(&qemu_option_rom_opts);
2954 qemu_add_opts(&qemu_machine_opts);
2955 qemu_add_opts(&qemu_smp_opts);
2956 qemu_add_opts(&qemu_boot_opts);
2957 qemu_add_opts(&qemu_sandbox_opts);
2958 qemu_add_opts(&qemu_add_fd_opts);
2959 qemu_add_opts(&qemu_object_opts);
2960 qemu_add_opts(&qemu_tpmdev_opts);
2961 qemu_add_opts(&qemu_realtime_opts);
2962 qemu_add_opts(&qemu_msg_opts);
```

每个QemuOptsList存储了大选项所支持的所有小选项，如-realtime大选项QemuOptsList的定义：

```
507 static QemuOptsList qemu_realtime_opts = {
508     .name = "realtime",
509     .head = QTAILQ_HEAD_INITIALIZER(qemu_realtime_opts.head),
510     .desc = {
511         {
512             .name = "mlock",
513             .type = QEMU_OPT_BOOL,
514         },
515         { /* end of list */ }
516     },
517 };
```

-realtime只支持1个子选项，且值为bool类型，即只能是on或者off。

在调用qemu\_opts\_parse解析子选项之前，QEMU会调用qemu\_find\_opts("realtime")，把QemuOptsList \*从qemu\_add\_opts中找出来，和optarg一起传递给qemu\_opts\_parse去解析。QEMU可能会多次使用同一个大选项来指定多个相同的设备，在这种情况下，需要用id来区分。QemuOpts结构体就表示同一id下所有的子选项，定义如下：

```
46 struct QemuOpts {
47     char *id;
48     QemuOptsList *list;
49     Location loc;
```

其中list是同一个大选项下不同id的QemuOpts链表。

本文参与[腾讯云自媒体分享计划](#)，欢迎正在阅读的你也加入，一起分享。

虚拟化

编程算法

数据结构

举报

点赞 10

分享

0 条评论

我来说两句

[登录](#) 后参与评论

## 相关文章

### apache配置https服务

1、创建自己签名的证书 #创建CA签名的证书,需要用到openssl yum install openssl ...

用户 1214695

### 插入法排序

何谓算法？算法就是计算机解决问题的方法和步骤。之所以强调计算机三个字，是因为计算机处理问题的方式和我们人类解决问题的方式有所不同。比如，在电视剧《宫》...

用户 1214695

### ls命令实现分析

###一、ls命令的功能分析 使用man ls命令查看ls命令手册（功能描述和主要选项摘录如下）： List information about the ...

用户 1214695

### VB6源码 webbrowser 自动登录网页批量下载文件 IE下载弹窗控制

VB6源码 webbrowser 网抓 自动登录网页批量下载文件 IE下载弹窗控制,网页元素控制等！！

巴西\_prince

### ubuntu关闭和开启防火墙

yaohong

使用原生Canvas API 绘制散点图。 (散点图及数据来自于百度Echarts官方示例库【查看示例链接】)。

大史不说话

## asp.net core发布到docker报Microsoft.ApplicationInsights.As...

dotnet core 2.1的asp.net core在docker下部署的时候发生下面的错误。 Error: An assembly specified...

kkllldog

## VB.NET 默认打印机设置及相关配置获取

VB.NET利用winspool.drv设置默认打印机,通过PrintDocument获取打印机相关配置!

巴西\_prince

## 超详细 | 逻辑回归大解析（手写推导+Python代码实现）

二十世纪早期，逻辑回归曾在生物科学中被使用，在那之‘后也在许多社会科学中被广泛运用。逻辑回归通常被应用于因变量（目标）是分类的场景，比如：

量化投资与机器学习微信公众号

## 照片解锁手机不能忍？教你用OpenCV做活...

名叫Adrian Rosebrock的程序猿，写了份事无巨细的教程，从构建数据集开始，一步步教大家用AI分辨真人和照片，...

量子位

更多文章

社区

专栏文章  
互动问答  
技术沙龙  
技术快讯  
团队主页  
开发者手册  
智能钛AI

活动

原创分享计划  
自媒体分享计划


资源

腾讯云大学  
技术周刊  
社区标签  
开发者实验室

关于

社区规范  
免责声明  
联系我们

云+社区



扫码关注云+社区  
领取腾讯云代金券

热门产品	域名注册	云服务器	区块链服务	消息队列	网络加速	云数据库	域名
	云存储	视频直播					
热门推荐	人脸识别	腾讯会议	企业云	CDN 加速	视频通话	图像分析	MySQL 数据库
	SSL 证书	语音识别					
更多推荐	数据安全	负载均衡	短信	文字识别	云点播	商标注册	小程序开发
	网站监控	数据迁移					





分享