[QEMU1.3.0的源码分析一 : 源码目录简介](http://blog.csdn.net/snsn1984/article/details/8265495)

最近在研究QEMU，读了一些QEMU的源码，因为涉及的东西比较多，找到的资料又都比较破碎，不太完整。所以将最近的成果总结一下。

相比其他的开源软件来说，QEMU源码下面目录比较多，下面就先把这些目录的内容大致整理一下。

docs/ 包含了一些文档，说实话，对初学者来说，读这些文档压根没有头绪

hw/   包含了所有支持的硬件设备  
include/  包含了一些头文件  
linux-user/  包含了linux下的用户模式的代码  
target-XXX/   包含了QEMU目前所支持guset端的处理器架构。包括：alpha,arm,cris,i386,lm32,m68k,microblaze,mips,openrisc,ppc,s390x,sh4,sparc,unicore32,xtensa. 此处的XXX就是指这其中的一种架构。包含的代码的主要功能是将该guest架构的指令翻译成TCG OP代码。也就是target-arm下的代码就是将arm架构的指令翻译成TCG OP。这些目录占了源码目录的很大一部分。  
tcg/   包含了动态翻译工具tcg的源码部分，主要是将TCG OP转化为host binary的部分。这个目录下也包含了多个架构名字命名的目录，每个目录下存放着针对该架构的代码。后续会详细介绍。  
test/ 从名字上可以看出，应该是存放测试部分的代码，但是目前这部分代码还没读。

[QEMU1.3.0源码分析之二：TCG](http://blog.csdn.net/snsn1984/article/details/8268434)

TCG是Tiny Code Generator的简称，它之前是一个后端编译器，现在是作为一个动态翻译器来使用。在QEMU中，它主要用来将虚拟出来的系统的指令转化成真正硬件支持的指令中的从中间代码到硬件支持的机器代码的过程。前端的将指令翻译成中间代码的过程，是一个反汇编的过程。

反汇编的过程的源码的主要地址：qemu source code/target-XXX。此处的XXX指的是模拟出来的系统的架构。

TCG的源码的位置是：qemu source code/tcg。这个目录下有很多文件夹，每个文件夹都代表一个目标架构。这里的目标架构指的是真正的硬件架构，也就是说运行QEMU的架构。

在qemu source code/tcg目录下，有一个README文件，介绍了tcg的主要内容。

在qemu source code/tcg/arm目录下，只有两个文件，实现了生成arm架构的内容。tcg-target.c和tcg-target.h两个文件。和arm同级目录的ia64,hppa,ppc,s390,i386,mips,ppc64,sparc,tci等目录下也是同样的名字的两个文件，当然文件的内容并不相同。关于这两个文件的内容，README是这么描述的：tcg-target.h contains the target specific definitions. tcg-target.c contains the target specific code.

动态翻译只是在必要的时候才进行翻译，而尽可能的将时间花费在执行host code上。TB(Translation Block)翻译之后得到的host code会存放在code cache中，因为有很多TB会被重复执行，所以这样会达到更加的效果。

[QEMU1.3.0的源码分析三：user model之linux](http://blog.csdn.net/snsn1984/article/details/8281566)

从源码目录来看，user model有两块内容bsd-user和linux-user。我主要研究了下linux-user这种情况。

首先要提一下通常容易关注的焦点，linux-user下的函数入口点：/源码目录/linux-user/main.c中的

Line:3388    int main(int argc, char \*\*argv, char \*\*envp).

找到了入口函数，就可以根据这个main函数中的调用关系来看看这个情况下的主要执行流程和动作了。

int main(int argc, char \*\*argv, char \*\*envp)  
{  
  
    module\_call\_init(MODULE\_INIT\_QOM);  
  
    qemu\_cache\_utils\_init(envp);

    /\*初始化了tcg的相关部分，包含了cpu动态转化的一些初始化操作。\*/

    tcg\_exec\_init(0);  
    cpu\_exec\_init\_all();

    /\*包含了虚拟cpu的初始化\*/

    env = cpu\_init(cpu\_model);

    /\*加载可执行程序，即Guest code\*/

    ret = loader\_exec(filename, target\_argv, target\_environ, regs,

        info, &bprm);

    target\_set\_brk(info->brk);

    /\*系统调用初始化\*/

    syscall\_init();

    /\*信号初始化\*/

    signal\_init();

   /\*此函数是主要的循环体，通过这个函数来实现对指令的动态翻译，并且执行翻译之后的Host Code。

   通过最终调用cpu\_gen\_code()函数（位于translate-all.c文件中）来实现

    动态翻译，其中调用了两个关键函数。一个关键函数是gen\_intermediate\_code()

   函数（位于target-arm/translate.c，此处以guest指令集为arm为例，其他的可以自行替换），

   这个函数的主要功能是根据Guest Code生成TCG Operations。另外一个重要的函数是

  tcg\_gen\_code()函数(位于tcg/tcg.c)，这个函数主要是把TCG Operations转化成Host code。\*/

    cpu\_loop(env);

    /\* never exits \*/

    return 0;

}

下面来分析下刚才介绍的重要函数cpu\_loop(). cpu\_loop()函数在linux-user/main.c中有多个版本，区别在于参数，参数是不同的cpu state，下面举例仍然以arm为主。

void cpu\_loop(CPUARMState \*env)  
{  
    int trapnr;  
    unsigned int n, insn;  
    target\_siginfo\_t info;  
    uint32\_t addr;  
  
    for(;;) {  
        cpu\_exec\_start(env);  
        trapnr = cpu\_arm\_exec(env);

        cpu\_exec\_end(env);

   ...............

}

可以看到for循环里有三个函数调用，分别是cpu\_exec\_start，cpu\_arm\_exec，cpu\_exec\_end。其中最重要的

cpu\_arm\_exec函数，通过target-arm/cpu.h中的宏定义#define cpu\_exec cpu\_arm\_exec调用了cpu-exec.c文件

中的cpu\_exec()函数。

cpu\_exec()是整个qemu中的一个重要函数，它负责整个核心的从guest code 到host code的翻译和执行。

cpu\_exec()首先会去调用tb\_find\_fast()，tb\_find\_fast()会判断取回来的tb是否合法，如果不合法会去调用tb\_find\_slow()函数。

tb\_find\_slow()会试图通过物理mapping去寻找tb，如果寻找失败则会调用tb\_gen\_code()去翻译代码。

cpu\_exec()函数调用tb\_find\_fast()之后会调用tcg\_qemu\_tb\_exec()去执行所找到的tb。最后再调用cpu\_exec\_nocache()去执行剩下的代码。