**操作系统实验**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lab1：

I386实模式（1.1）、保护模式（1.2）、保护模式下加载磁盘中Hello World程序（1.3）

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

院系：人工智能学院

姓名：石睿

学号：211300024

班级：操作系统-2023春季学期

邮箱：[211300024@smail.nju.edu.cn](mailto:211300024@smail.nju.edu.cn)

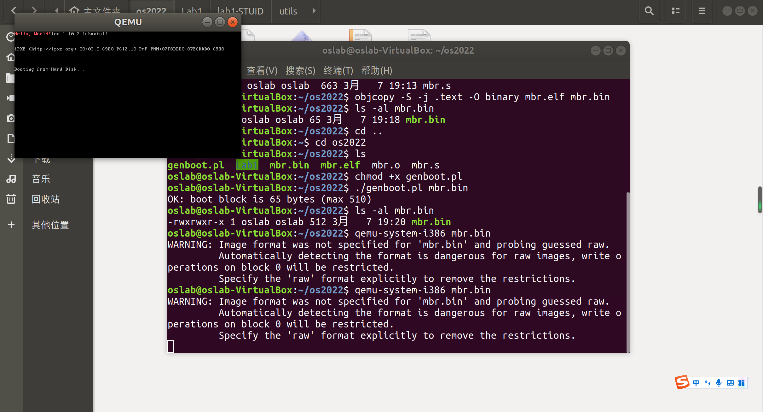
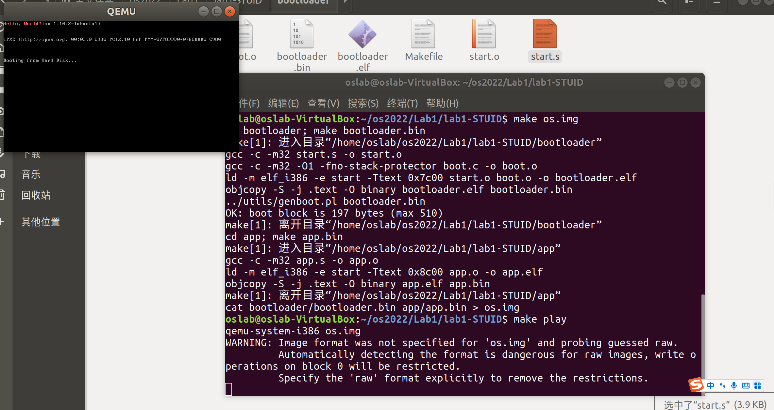
实验时间：2022.3.9

1. **实验进度**

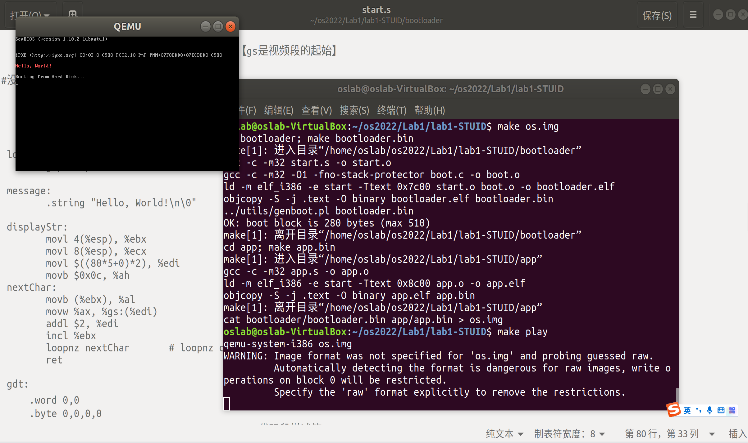
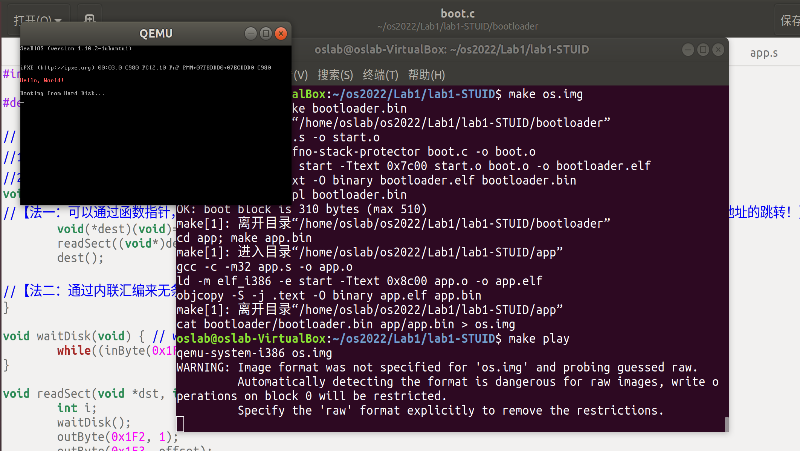
我已经完成了Lab1的所有内容。

1. **实验结果**

为节省报告的页数，把三个阶段的成果并排摆放。在各个阶段都可以正确在qemu中运行出Hello World呢！结果如下图所示，具体实现过程在第三部分介绍。

**Lab1.1： objcopy缩小mbr程序** **Lab1.：make命令完成编译链接**

**Lab1.2：保护模式下运行Hello World**  **Lab1.3：保护模式下加载OS（app）**

L

1. **实验修改的代码位置**

—简单描述为完成本次实验，修改或添加了那些代码，大致的文件和函数定位即可

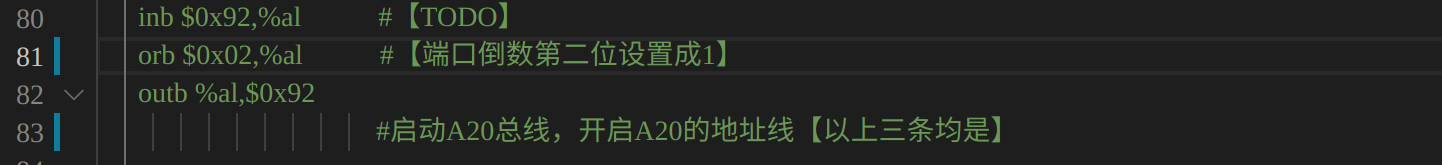
* 1. **Lab1.1**

去掉注释、通过objcopy缩小mbr程序，通过脚本程序genboot.pl生成MBR文件即可。

* 1. **Lab1.2**

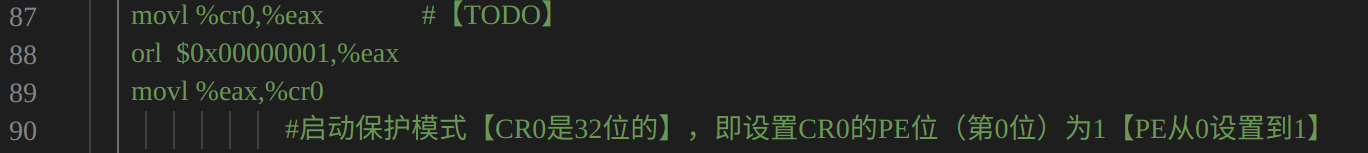
**2.1** 开启A20地址线—提升可访问的内存空间

A20是一个OR逻辑电路门，被放置在第20位的地址总线上。经查阅资料，共有三种方式开启A20地址线：键盘控制器、调用BIOS功能、使用系统端口0x92，本实验采用最后一种方式。0x92端口的bit1：为0时关闭A20总线，为1时开启A20总线。故做出如下修改。



**2.2** 启动保护模式—从实模式转变为保护模式

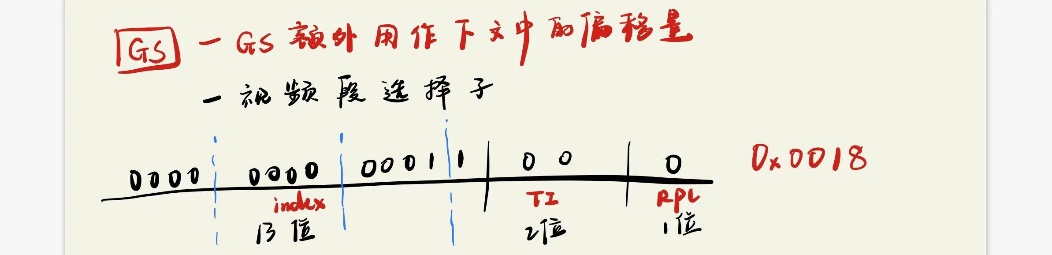
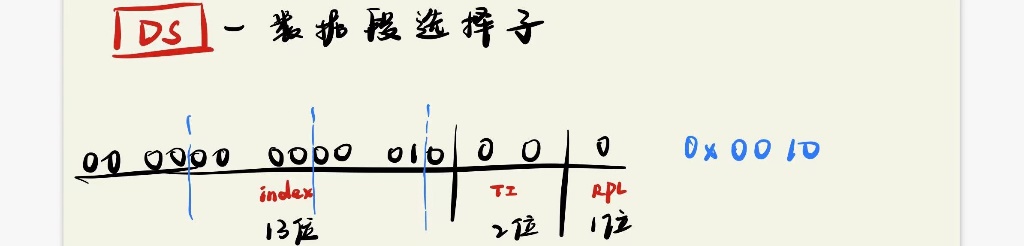
通过将CR0中的PE位设置为1,来开启保护模式。其中PE是CR0的最后一位。

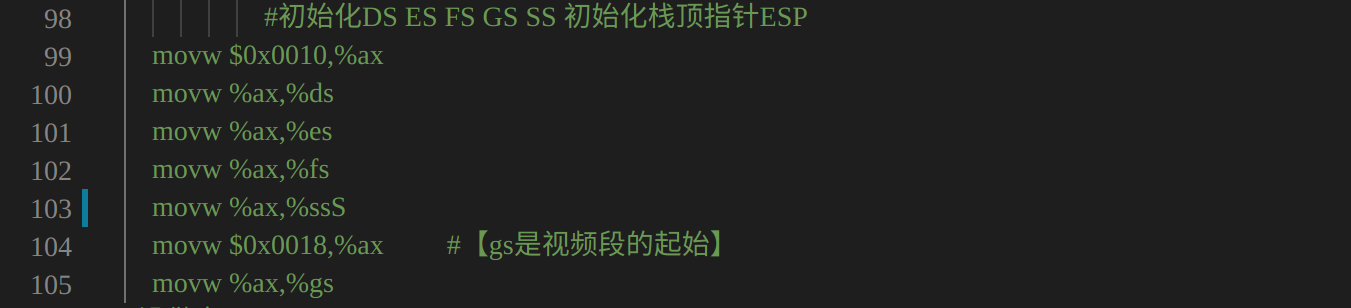


**2.3** 初始化DS ES FS GS SS寄存器

I386中段寄存器在保护模式中为使用在GDT上的段选择子，

其构成为index+TI+RPL，其中index是本.s文件中新定义的全局描述符中的下标位置。针对DS和GS的初始化如下图所示（ES FS SS没有用到，就都给0x10啦）。

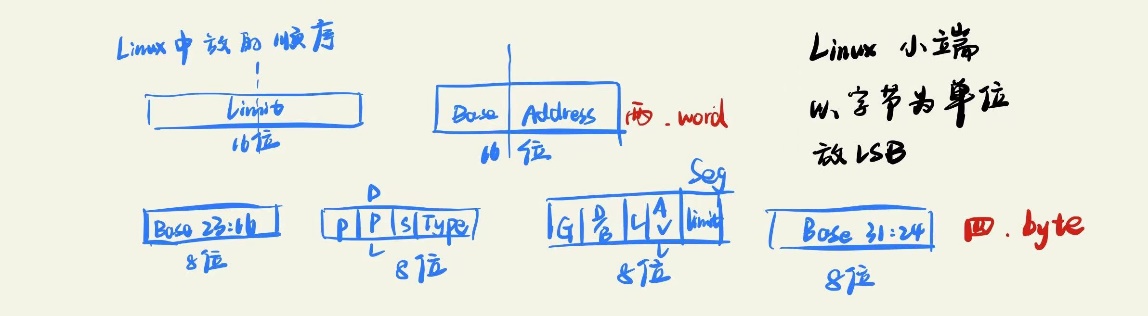


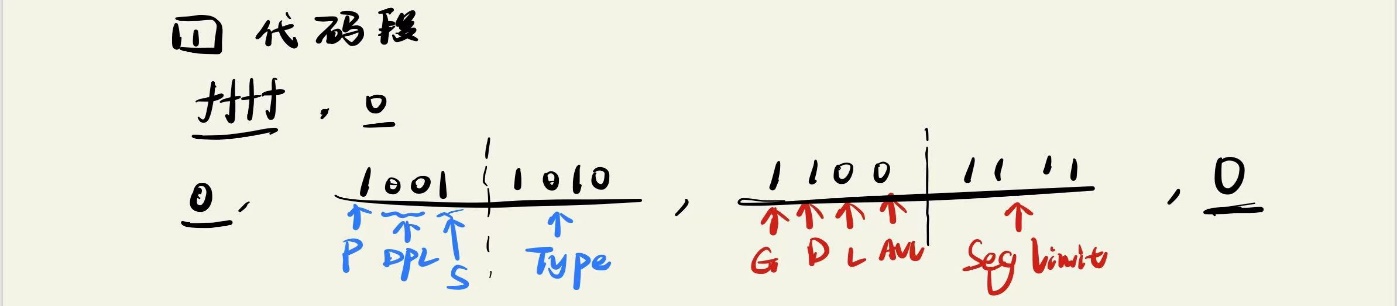


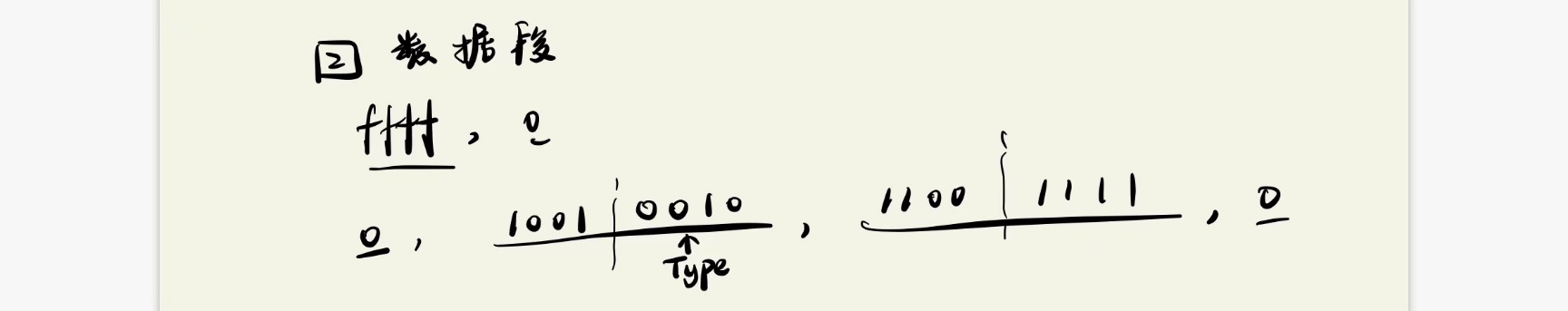
**2.4** 填写GDT

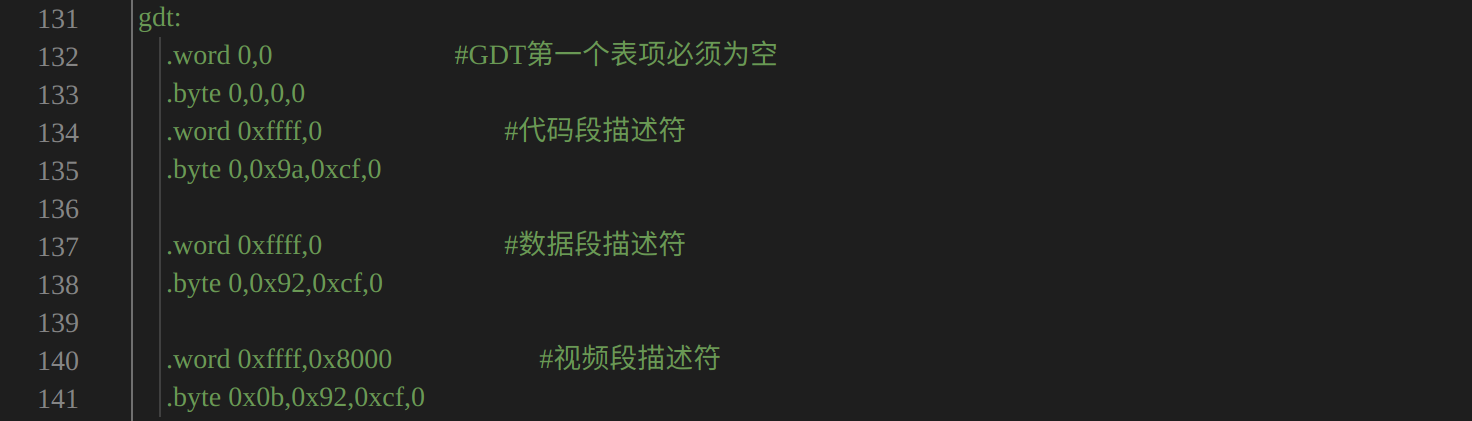
查阅资料得到，Linux内核代码段和数据段的**基地址Base**均为0x00000000,**段限长Limit**均为0xffff,代码段的**Type**为0xa，数据段的Type为0x2。两者**粒度G**均为1，**特权级DPL**均为0。

考虑到IA-32+Linux为小端机器，填写GDT中的表项如下所示。







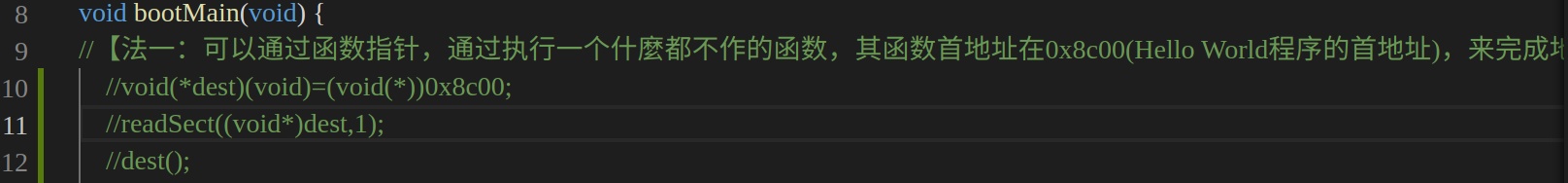


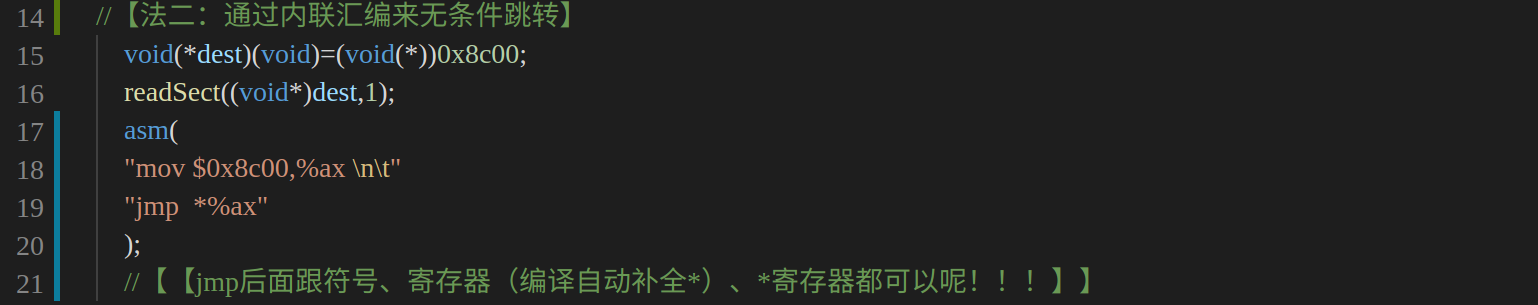
其中视频描述段手册中提到了为0x8000，且是数据段的一部分！

* 1. **Lab1.3**

把MBR之后扇区中的程序到内存的特定位置并跳转到0x8c00，有两种实现方式：

1采用函数指针进行跳转void(\*dest)(void)，2采用内联汇编中的无条件跳转jmp。





1. **实验心得**

在去年的ICS课程中，我是非常抵触阅读大篇幅的Makefile的。但在Lab1.3中，为

了搞清楚脚本程序genboot.pl、加载程序bootloade、以及建议操作系统app之间的关系，我阅读了不同目录下的Makefile文件，最终得到以下结论。

bootloader是加载程序，被放到了0x7c00，并且通过脚本程序genboot.pl产生了MBR。是BIOS把设备第一个扇区放到了0x7c00(i386规定)。同时也是一个加载程序，被BIOS启动之后，会进一步加载OS（本实验中就是app啦）。app.bin被放到了0x8c00的位置，且文件大小好像是58字节，小于512字节，所以bootMain函数中只需要读取一个磁盘扇区。

所以阅读Makefile对搞清楚编译、链接，不同文件之间的关系还是非常重要哒。

此外，Lab1也锻炼了我信息检索能力，比如查A20总线的开启、GDT的填写，也迫使我静下心来阅读部分Linux的内核源码；Lab1也让我对i386架构的开机程序以及几个不同模块之间的关系更加清楚啦。

**五、思考题的看法**

EX1：CPU、内存、BIOS、磁盘、主引导扇区、加载程序、操作系统之间的关系

**CPU**加电之后第一条指令在内存中，**内存**包含ROM（随机访问存储器，断电内容无）、RAM（只读存储器，断点可保存）。**BIOS（基本输入输出系统）**就在RAM之中，且CPU加电跳转的第一条指令在内存的RAM的BIOS固件中，进行开机自检。

BIOS运行，step1：检查所安装的RAM数量、键盘和其他设备可否正常响应。Step2：，扫描PCI总线上的所有设备。Step3：尝试CMOS（内存的一种，不易失）存储器中的设备清单决定启动设备。Step4：启动设备的第一个扇区，即**磁盘的主引导扇区（MBR）**，BIOS读入内存加载到0x7c00（i386规定）并执行**加载程序**。其主要的功能是：把**操作系统**的代码和数据从磁盘加载到内存中，再跳转到OS的起始地址，并启动OS。

EX2：中断向量表是什么？

是中断服务程序入口地址的偏移量与段基址，一个中断向量占据4字节空间。它把所有的中断向量集中起来，按中断类型号从小到大的顺序存放到存储器的某一区域内（i386放在0x0000-0x03FF的区间），故中断向量表即**中断服务程序入口地址表**，总共存储256个中断向量。在中断响应过程中，CPU通过从接口电路获取的中断类型号（中断向量号）计算对应中断向量在表中的位置，并从中断向量表中获取中断向量，将程序流程转向中断服务程序的入口地址。

EX3：为什么段的大小最大为64KB？

I386在实模式中，寻址空间20位，段寄存器和偏移地址16位，其物理地址的计算为**物理地址=段寄存器<<4+偏移地址**。段寄存器左移四位是本段（数据段、代码段、堆栈段……）的基址，偏移地址是相对于基址的偏移量。所以为了保证可以访问到某段的所有内容，在偏移地址只有16位的前提下，以及段寄存器后四位均为0时，段大小只能最大为216=64KB大小啦！

EX4：对于脚本程序genboot.pl，先打开mbr.bin，然后检查文件是否大于510字节，说明在此步之后做了什么？为什么这么做？

首先，$n = sysread(SIG, $buf, 1000);读取了文件大小，并且检查这个文件大小是否大于510字节【因为要把一个.bin文件制作成MBR，一种512字节的最后两字节为\x55和\xAA的文件，所以只有510字节可供文件放置本来的信息】。

然后做了如下两个动作，$buf .= "\0" x (510-$n); 把剩余的位置都放\0，标志着文件有效信息的结束

$buf .= "\x55\xAA";给MBR文件配置末尾的魔数，为了在BIOS检查文件的时候可以通过。

EX5：简述电脑从加电开始，到OS开始执行为止，计算机是如何运行的

同EX1的回答啦！

【助教哥哥还请见谅，因为写了五个思考题超过了四页啦ε(┬┬﹏┬┬)3】