装系统教程专业版

前言

请问您会装系统吗？也许不会，我以前也不会。网上有人说装系统是最没有技术含量的一门技术，百度一下人人都会。好像有点儿道理，不过我不太认同，因为如果没有技术含量的话我也就不会出这个专业的教程了。

不仅仅是大多数普通计算机用户不会装系统，就连大多数学习计算机相关专业的人也不会装系统，会装的也是按网上的教程来的，稀里糊涂地就装完了，具体什么原理也不清楚。为什么？因为大学里就没有开设装系统这门课程。哈哈，好像还真没有，毕竟是“旁门左道”雕虫小技，30 块钱的技术登不了大雅之堂。

不过还好，想学习装机这方面技术的您总算是在茫茫的知识汪洋中发现了这份教程。我打算写这份教程已有好几个月了，由于种种原因不得不推迟（**其实是因为我太懒了**），历经数百天的风风雨雨，本教程总算是完工了。（最初是 2019 年下半年开始写的，后来又陆陆续续有所增补）

让我们踏上装系统的征程吧。

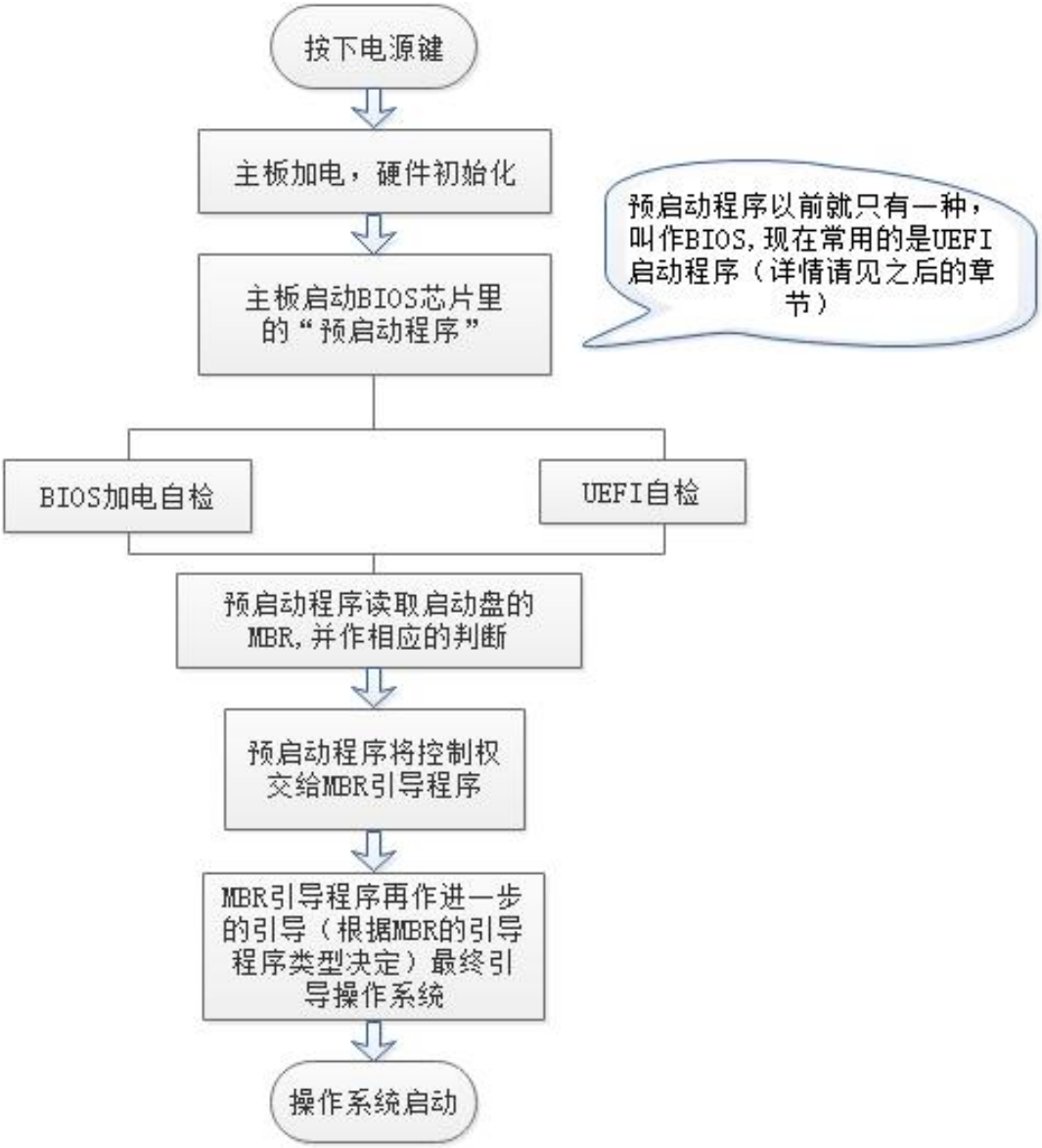
作者：李茂福

邮箱：sysyear@163.com

更新日期：2024-10-14

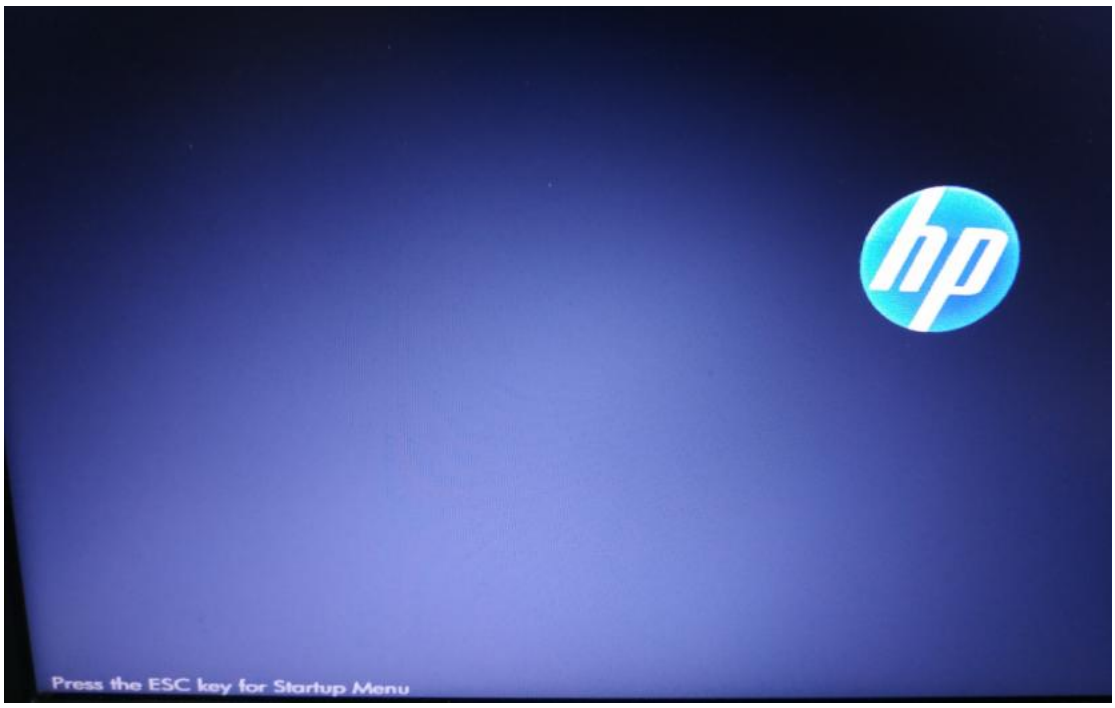
第 0 章、计算机的启动过程

大概的过程如下图：

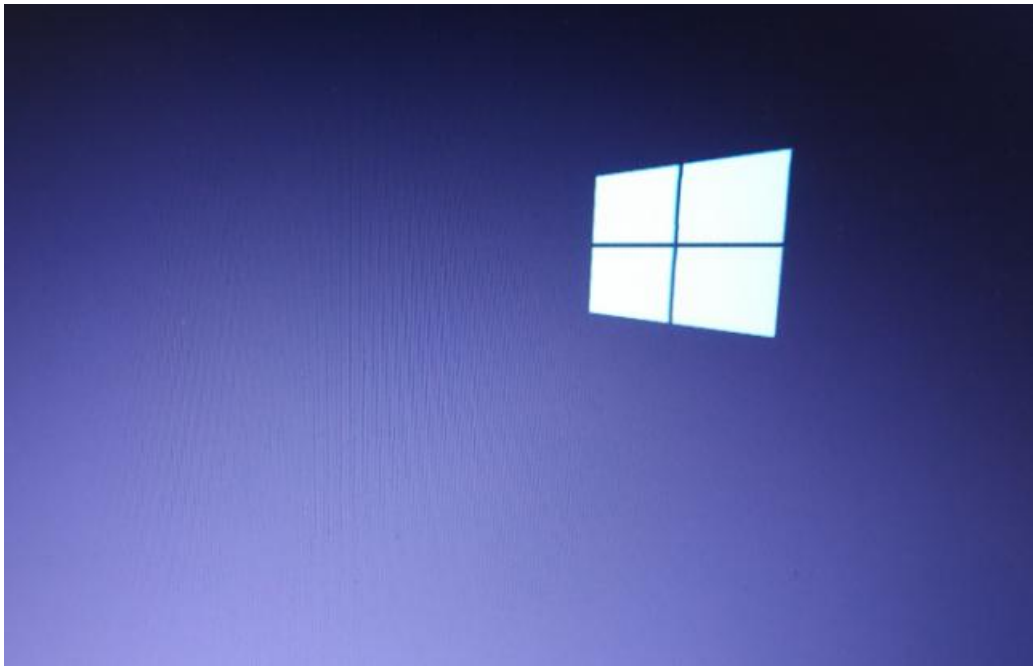


我们来看一下真实的电脑的启动过程（以惠普笔记本电脑为例）

1.首先是按下电源键，显示如下图，注意看屏幕左下角的一行字 Press the ESC key for Startup Menu，意思是按下 ESC 键可进入启动菜单，我们不按任何按键，因为这台电脑本来装有 windows10 系统，我们就只看看怎么进入系统的。



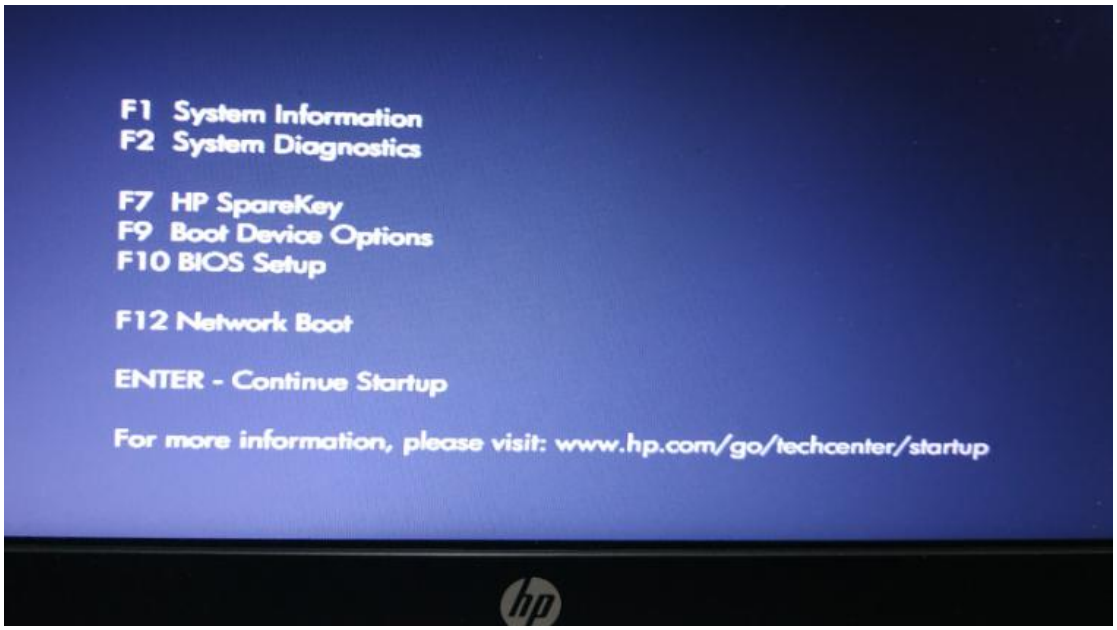
2.过了几秒钟，显示如下图，可见是 windows 系统启动了



3.再过几秒钟，显示如下图，操作系统启动了，启动过程结束了。

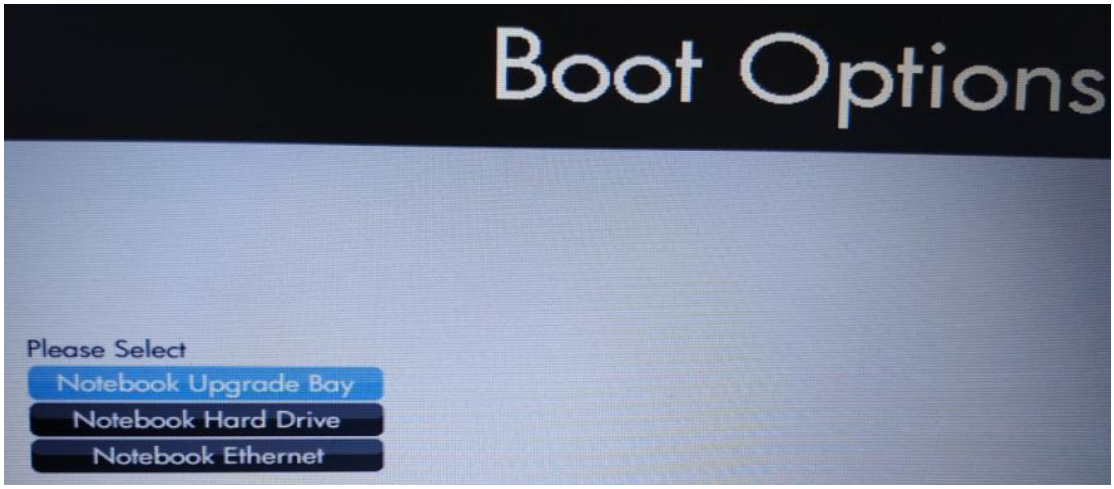
所以，计算机的启动过程中，并不会显示太多的消息。除非中途某个过程出了问题。那么计算机怎么知道我要从哪个硬盘启动呢，我又怎么知道计算机是用了哪种预启动程序呢？

4.根据第一张图的提示去操作，我们在开机时按下 ESC 键，进入启动菜单查看相关的信息，如下图所示，我们按下 F9 键（Boot Device Options）可以选择从哪个设备启动

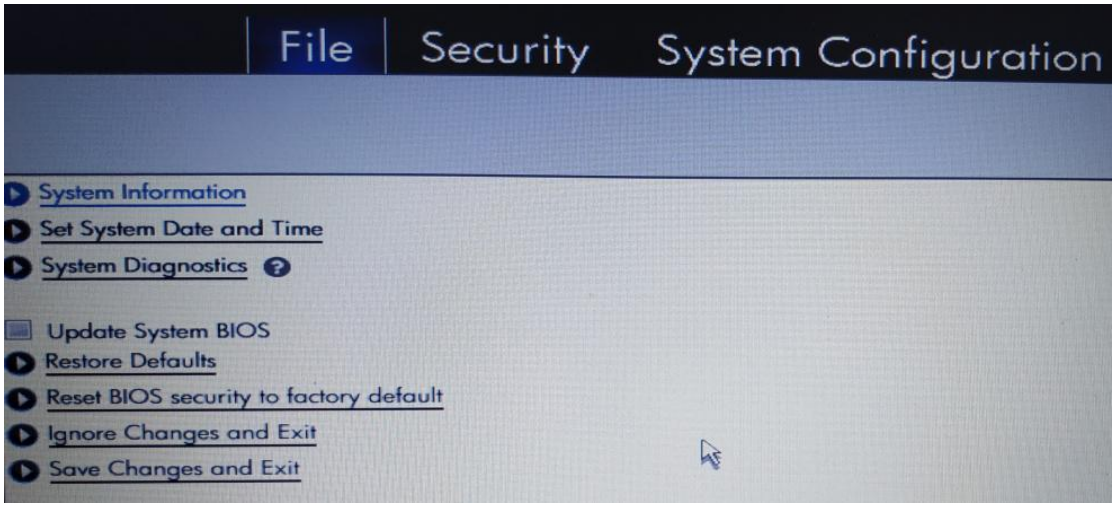


其实出现上图的界面也说明了该电脑是使用的传统的 BIOS 预启动程序，因为它不支持鼠标，界面很单调，只有文字界面。

5.如下图，此计算机可以有 3 种启动设备，中间一个是从硬盘启动，此计算机只有一块硬盘。其他 2 个先不管。按下 ESC 键可以回到第 4 步的启动菜单



6.在启动菜单里按下 F10（BIOS Setup）进入 BIOS 设置，这台惠普的型号是 ProBook4431s，比较旧了，2012 年 3 月上市的，并不完全支持 UEFI，那个时候一般都是用的传统的 BIOS 启动。如果您的电脑**预装**的是 windows 8 及以上的系统则说明是用 UEFI 启动的。



本章思考问题：

1.什么是预启动程序？

2.什么是 MBR？

3.传统 BIOS 启动的电脑也能装 windows 10 吗？

第 1 章、预启动程序 BIOS 和 UEFI

BIOS 全称 Basic Input Output System（基本输入输出系统）是一组固化在计算机主板上的一个 ROM 芯片里的程序，该 ROM 芯片也叫 BIOS 芯片，保存着计算机最重要的基本输入输出程序，开机后的加电自检程序和系统自启动程序等。它可以从 CMOS 中读写系统设置的具体信息。也就是说预启动程序存放在 BIOS 芯片里，它的配置存放在 CMOS 芯片里。

什么是基本输入输出呢，就是基本的输入和基本的输出，比如键盘，屏幕，扬声器等。BIOS 预启动程序最早是存放在 ROM 介质中的，ROM 为只读存储器，不可以修改里面的程序。后来这个介质升级了，换了材料，于是可以修改或升级里面的 BIOS 程序了，BIOS 芯片先后使用过的介质如下：

ROM（Read Only Memory）只读存储器
EPROM（Erasable Programmable ROM）可擦除可编程的 ROM（一般用紫外线去擦除）
EEPROM（Electrically Erasable Programmable ROM）电可擦除可编程的 ROM
NORFlash　可在操作系统里运行软件进行 BIOS 程序更新的介质

BIOS 预启动程序的主要功能：
①加电自检（Power On Self Test）电脑刚接通电源时对硬件部分的检测，包括对 CPU，640K 的基本内存，1MB 以上扩展内存，ROM 芯片，主板，CMOS 存储器，串口，并口，显示卡，软盘硬盘子系统以及键盘 等进行测试。（没有鼠标，这也就是传统的 BIOS 界面不支持鼠标的原因）
②初始化，包括创建中断向量，设置寄存器，对一些外部设备进行初始化等。
③引导操作系统，先寻找可引导的设备，根据 CMOS 的设置选择启动的磁盘，再查找启动盘上的引导记录，最后把电脑的控制权交给引导记录。

在刚加电后大概 1 到 2 秒我们可以按下 ESC，DEL，F1~F12 等键进入 BIOS 的设置菜单，并保存设置到 CMOS 里，这样下次电脑重启时就按我们的设置启动了（具体是按下哪个按键，得看屏幕上的输出提示了，一般都会有输出提示的）

BIOS 预启动程序并不是一个具体的程序，它是指一个大的种类，具体的 BIOS 程序是由主板厂商去开发的，常见的主板 BIOS 有：Award, AMI, Phoenix, Insyde, BYOCORE 等

特点：BIOS 运行于 16 位模式，使用静态链接，仅支持 1M 以下的内存，固定编址，汇编代码，图形界面不丰富，不支持鼠标操作。

EFI/UEFI

英特尔公司从 2000 年开始开发了 EFI 可扩展固件接口（Extensible Firmware Interface），EFI 用以规范 BIOS 的开发。支持 EFI 的 BIOS 也称为 EFI BIOS，不支持 EFI 的 BIOS 则统称为 Legacy BIOS（传统的 BIOS）。之后为了推广 EFI，多家厂商共同成立了 统一可扩展固件接口论坛（UEFI Forum），英特尔公司把 EFI 1.1 规范贡献给业界，用以制订新的国际标准 UEFI 规范。

特点：EFI 运行于 32 位或 64 位模式，使用动态链接，99%的编码由 C 语言完成，模块化的部件，支持鼠标操作等。

EFI 组成部分：

Pre-EFI 初始化模块（PEI）
EFI 驱动执行环境（DXE）
EFI 驱动程序
兼容性支持模块（CSM）
EFI 高层应用
GUID 磁盘分区
……

UEFI 是由 EFI 1.10 为基础发展起来的，与传统的 BIOS 的区别主要是它支持的硬件更多了，界面更丰富了，支持第三方的开发，支持 UEFI 驱动程序。

UEFI 体系的**驱动程序**并不是可直接在 cpu 上运行的二进制代码，而是 EFI Byte Code，即 **EFI 字节码**，类似于 Java 的 ByteCode。EFI 字节码必须在 UEFI 驱动执行环境下（类似于 Java 虚拟机）被解释运行。一个带有 UEFI 驱动的扩展设备可安装在多种不同的操作系统下，跨平台的，因为它的代码是 EFI 字节码，解释型的，这样就无须考虑操作系统的兼容性问题。

UEFI 使用模块化设计，在逻辑上分为硬件控制和 OS 软件管理 2 部分，硬件控制为所有 UEFI 版本所共有，而 OS 软件管理是一个开放的可编程接口，借助这个接口，主板厂商可以实现各种丰富的功能。

★在本文档中，把传统的 BIOS（Legacy BIOS）及 UEFI BIOS 都统称为“预启动程序”
如无特别说明，则 Legacy BIOS 仍称为 BIOS，把 UEFI BIOS 称为 UEFI

第 2 章、硬盘寻址方式

机械硬盘是较早使用的磁盘之一，也是计算机中最常用的外部存储设备，机械硬盘是怎么存储数据的呢？

先看看机械硬盘的结构，以 3.5 英寸 SATA 接口的硬盘为例：

1.外观就长这样，厚度约 2.5cm，宽约 10cm，长约 14.5cm，容量 500GB，转速 7200 转每分钟（一秒转 120 圈）转速非常快

正面金属盖上帖有相关信息的标签，背面有一块控制电路板

2.拆开前面的盖板后，里面的情况如下图

主要部分为一个金属盘片，左下部分为磁头及磁头控制部分，

家用的 SATA 盘一般只有一块盘片，分上下 2 面，磁头也有一对（上下 2 个磁臂，夹着这块盘片，如下图

★机械硬盘的大概的工作原理：

1.金属盘片表面涂有磁化介质，这些磁化介质用以存储数据

2.磁头连接控制电路，磁头加电使盘片的某个点磁化，加电的状态不同可以磁化出不同的极性；磁头也可以读取盘片上某个点的磁性，并转成电信号输出到控制电路板中。（和磁带复读机的原理一样）

3.盘片中央部分连着一个马达（电机），马达可以使盘片旋转，当磁头不动时，盘片旋转一圈所形成的圆形轨迹就对应盘片的一个数据记录轨道，叫作磁道

4.磁头尾部也有线圈，上下夹着磁铁片，所以在电路的控制下磁头也可以移动，这样磁头就可以定位到盘片的任何一个轨道

以前的机械硬盘把盘片上的每一个轨道都划分为 63 段用以存储数据，每一段叫作扇区，一个扇区可以存储 512 字节的数据。盘片内圈和外圈的轨道（或者说每一个轨道）的长度是不一样长的，但是都划为 63 个可存储数据的扇区，每扇区对应的圆心角是一样的。这样方便磁头对扇区的定位。磁头读取或写入数据时是以扇区为单位的，一次写入或读出 一个扇区的数据。

当然最新的硬盘由于技术的提升，可能每一个轨道的扇区划分数已经不同了，不再是固定的 63 扇区，扇区的字节数也可能不再是 512 字节了，有 1KB，2KB 或者更大的扇区。以前的机械硬盘盘片上的轨道数量也比较少，一般在 1024 条轨道以内。盘片可以有多片。（这点大家了解一下就行，**逻辑**上我们还是认为一个磁道有 63 个扇区，一个扇区为 512 字节）

要想读取机械硬盘上的某个数据，就要知道目标数据存放在哪个盘面上，盘面的哪条磁道上，磁道的哪个扇区中，最小单位是扇区。就算是要读取一个字节，也是要读取一个扇区，再由磁盘的控制电路去析取具体的字节，这属于软件的操作了。所以硬盘上的数据的地址就要有 3 个字段：

盘面（即对应的磁头号）+磁道号+扇区号

这个地址最初只设计了 3 个字节的长度（24 位）来记录，所以最多能表示 2 的 24 次方减 1 个扇区，一个扇区默认为 512 字节。多个盘片上半径相同的磁道构成了一个柱面，所以**磁道号也叫柱面号**。

因为柱面 Cylinder，磁头 Head,扇区 Sector 的英文首字母为 CHS，所以机械硬盘的寻址方式也叫作 CHS 寻址。在表示硬盘数据的地址中顺序不一定是 CHS 的顺序，比如在 MBR 中，该地址的前 8 位用于表示磁头号，接下来的 2 位及最后 8 位用于表示柱面号，中间剩下的 6 位表示扇区号

CHS 地址： **00000000 00 000001 00000000** （表示 0 磁头 1 扇区 0 柱面）

红色的 8 位为磁头号，棕色的 6 位为扇区号，蓝色的 10 位为柱面号（磁道）

磁头号表示范围 0 至 255，扇区号表示范围 1 至 63，柱面号表示范围 0 至 1024，**CHS 的扇区号是从 1 开始编号的**！

CHS 地址最大只能表示 7.8GB 的大小（256x63x1024 个扇区），所以当磁盘容量大于 7.8GB 时，CHS 寻址方式就不适用了，虽然在某些磁盘管理软件里能看到大于 1024 的柱面号，那也**只是为了方便理解**而转换成 CHS 的表示方式，实际上肯定已经不是用传统的 CHS 寻址了。U 盘和固态硬盘也肯定不是用 CHS 寻址。

后来出现了 **LBA 寻址**（Logical Block Addressing）逻辑块地址，地址长度大于 3 个字节，可以有 28 位，32 位，48 位，64 位，视具体的情况而定，单位也是扇区，一个扇区默认为 512 字节。LBA 只是一个逻辑上的地址表示，在机械硬盘中肯定还是要进行转换的。LBA 地址从 **0 号开始**给扇区编号，也只表示扇区号，以扇区为单位。

第 3 章、MBR 主引导记录

MBR（Master Boot Record）主引导记录是位于磁盘最前边的一段数据，它占一个扇区的大小，512 字节，位于磁盘的 0 柱面 0 磁头 1 号扇区，该扇区也叫主引导扇区。

主引导记录由 3 个部分组成：

①主引导程序，446 字节长度，它负责判断磁盘分区的正确性和分区引导信息的定位，以及引导下一步的代码

②DPT 磁盘分区表（Disk Partition Table），占 64 字节，由 4 个分区表项组成（每项 16 字节），负责说明磁盘上的分区情况

③结束标识，占 2 字节，其值逻辑上为 0xAA55，存储时为小端字节序（55AA）

MBR 磁盘分区表项各字节含义（每一项为 16 字节）：

第~字节	该字节（组）表示的含义
1	引导标志，0x80 表示活动分区，0x00 表示非活动分区
2，3，4	表示该分区的起始地址，使用 CHS 地址，大于 7.8G 时其值无效
5	分区类型，详见后面的 “磁盘分区类型”章节
6，7，8	表示该分区的结束地址，使用 CHS 地址，大于 7.8G 时其值无效
9，10，11，12	该分区之前已经使用了的扇区数，也可用于表示该分区的起始 LBA 地址
13，14，15，16	该分区的总扇区数，该数值加上前面的已使用的扇区数 之合就表示该扇区的结束 LBA 地址

MBR 磁盘分区的规则：

1.一个分区的所有扇区必须连续，一个磁盘最多可有 4 个主分区，或 3 个主分区和一个扩展分区。（扩展分区最多只能有一个，关于扩展分区详见之后的章节）

2.Windows 系统安装的分区必须为激活的分区，即该分区的 分区表项的第 5 个字节为 0x80（活动分区）

3.一个分区至少要占一个柱面（7MB），最大不超过 2TB（一个分区的总扇区数用 4 个字节表示，所以最大为 2TB）

4. MBR 磁盘最大可利用空间为 4TB，划分多个分区，最后一个分区设为 2TB，前面的几个分区空间之合不超过 2TB，由于一般装系统时不会这样去划分分区，所以常规情况下 **MBR 磁盘最大可利用空间为 2TB**

本来 MBR 中的引导代码是用来引导操作系统的，但随着操作系统的复杂化，MBR 中的代码也越来越复杂，但由于 446 个字节的限制，使得 MBR 扇区中已放不下那么多的代码了，所以剩余的代码放到了其他的扇区中，在 MBR 引导之后执行的代码也就叫作第二阶段的引导代码了。第二阶段的引导代码可以位于紧挨 MBR 之后的几个扇区中，也可位于分区中的头几个扇区中。

第 4 章、MBR 磁盘分区类型

磁盘分区的类型往往和该分区使用的文件系统相关联，MBR 中的每一个分区表项中的第 5 个字节表示分区的类型，常见的类型如下表：

第 5 字节的值 Hex	表示的分区及文件系统类型
00	空
01	FAT12
04	FAT16 <32MB
06	FAT16 >32MB
07	NTFS
0B	FAT32
0C	FAT32 LBA
0E	FAT16 LBA
0F	Win95 Extend >8GB
05	扩展分区
11	Hidden FAT12 隐藏的分区
14	Hidden FAT16 隐藏的分区
16	Hidden FAT16 >32MB 隐藏的分区
17	Hidden NTFS 隐藏的分区
1B	Hidden FAT32 隐藏的分区
1C	Hidden FAT32 LBA 隐藏的分区
1E	Hidden FAT16 LBA 隐藏的分区
35	OS/2 JFS
80	Old Minix
81	Minix /Old Linux
82	Linux Swap
83	Linux
85	Linux Extended 扩展分区
8E	Linux LVM
86	NTFS volume set
87	NTFS volume set
A5	Free BSD, NetBSD, 386BSD
EE	EFI Header MBR（GPT 保护分区，表示使用 GPT 分区表）
EF	Partition with an EFI file system

MBR 分区特点：

1.有激活（活动的）与未激活的区分

2.有隐藏与未隐藏的区分

第 5 章、EBR 扩展分区

MBR 的分区表只有 4 个分区表项，最多能表示 4 个主分区，若要划分多于 4 个分区时，只能使用扩展分区了，就是把未分配给主分区的磁盘剩余空间放到一个分区里（扩展分区），然后再在这个扩展分区里划分多个逻辑上的分区。实现的方法是在扩展分区里再使用一张分区表，类似于 MBR 主引导记录里的分区表，在扩展分区里的类似于 MBR 的结构叫作 **EBR 扩展引导记录**（Extended Boot Record）

EBR 和 MBR 的区别是 EBR 中没有引导代码和磁盘签名，其他的一样。

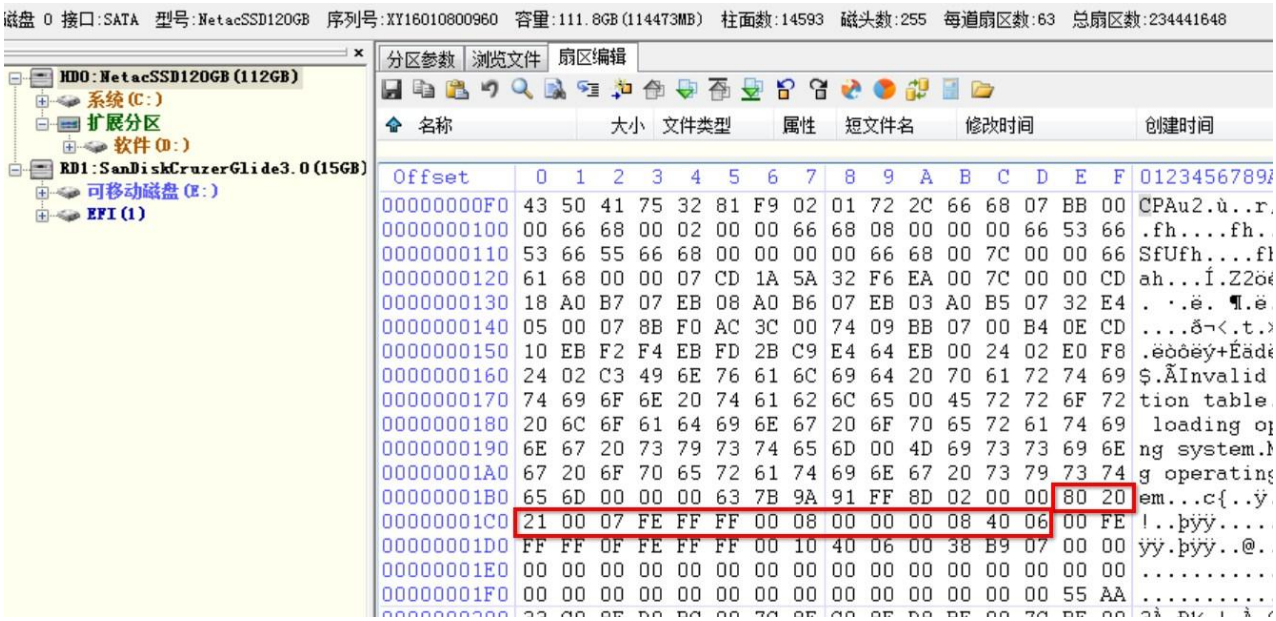
MBR 与 EBR 的详情请看下表：

<table>
<tr><td rowspan="15">整个磁盘
↓</td><td rowspan="6">MBR 主引导记录</td><td>引导代码</td><td colspan="2">446 字节长度</td></tr>
<tr><td>分区表项 1</td><td colspan="2">指向第 1 个分区的数据区（每个表项 16 字节）</td></tr>
<tr><td>分区表项 2</td><td colspan="2">指向第 2 个分区的数据区</td></tr>
<tr><td>分区表项 3</td><td colspan="2">指向第 3 个分区的数据区</td></tr>
<tr><td>分区表项 4</td><td colspan="2">指向第 4 个分区（扩展分区）</td></tr>
<tr><td>0x55AA</td><td colspan="2">2 字节，逻辑值为 0xAA55，存储时使用小端字节序</td></tr>
<tr><td>分区 1 的数据区</td><td colspan="3">由 PBR 分区引导记录及数据组成</td></tr>
<tr><td>分区 2 的数据区</td><td colspan="3"></td></tr>
<tr><td>分区 3 的数据区</td><td colspan="3"></td></tr>
<tr><td rowspan="9">扩展分区（4）</td><td rowspan="5">逻辑分区（5）</td><td rowspan="4">EBR 扩展引导记录</td><td>分区表项 1（表示当前分区）</td></tr>
<tr><td>分区表项 2（指向下一个逻辑分区(6)）</td></tr>
<tr><td>分区表项 3（未使用）</td></tr>
<tr><td>分区表项 4（未使用）</td></tr>
<tr><td colspan="2">当前分区的数据区</td></tr>
<tr><td rowspan="5">逻辑分区（6）</td><td rowspan="4">EBR 扩展引导记录</td><td>分区表项 1（表示当前分区）</td></tr>
<tr><td>分区表项 2（指向下一个逻辑分区(7)）</td></tr>
<tr><td>分区表项 3（未使用）</td></tr>
<tr><td>分区表项 4（未使用）</td></tr>
<tr><td colspan="2">当前分区的数据区</td></tr>
<tr><td>逻辑分区(xx)</td><td colspan="2">.....</td></tr>
</table>

不一定是第四个分区项指向扩展分区，也可以是只划分一个主分区，然后剩下的磁盘空间都划给扩展分区，再在扩展分区里划分多个分区。这种磁盘分区划分方式常见于 windows 7 及之前的系统。

第 6 章、实战分析 MBR 分区记录

打开 DiskGenius，查看硬盘上的第一个扇区（扇区 0）即 mbr 扇区，如下图所示



MBR 引导代码有 446 个字节，所以分区表从 447 字节开始，用十六进制表示为 01BE 的地址，所在行为 00000001B0,所在列为 E 开始，上图中圈出的 16 个字节即为第一个分区项（都是十六进制数）

80　20 21 00　07　FE FF FF　00 08 00 00　00 08 40 06

解析如下：

第 1 字节 80 表示此分区为活动分区，即激活的分区

第 2,3,4 字节表示该分区的起始地址（CHS 地址 20 21 00）大于 8GB 的硬盘可忽略此字段

第 5 字节表示分区类型，07 表示 NTFS

第 6,7,8 字节表示该分区的结束地址（CHS 地址 FE FF FF）可忽略此字段

第 9 到 12 字节表示本分区之前已使用了的扇区数，使用主机字节序（小端字节序）

所以 00 08 00 00 应该从右往左看，即 00 00 08 00 对应十进制的 2048 扇区

第 13 到 16 字节表示该分区的总扇区数，00 08 40 06 从右往左看是 06 40 08 00 对应十进制的 104859648 扇区，所以该分区总大小为 104859648 X 512 字节（50GB）

50GB，刚刚好是 C 盘的大小，所以上面的第一个分区项是记录 C 盘这个分区的。

接下来的第 2 个分区记录为：

00　FE FF FF　0F　FE FF FF　00 10 40 06　00 38 B9 07

解析如下：

第一字节 00 表示此分区为非活动分区，即未激活的分区

第 2,3,4 字节表示该分区的起始地址（CHS 地址 FE FF FF 大于 8G 的硬盘通常忽略 CHS 地址）

第 5 字节表示分区类型，0F 表示扩展分区

第 6,7,8 字节表示该分区的结束地址（CHS 地址 FE FF FF，可忽略此地址）

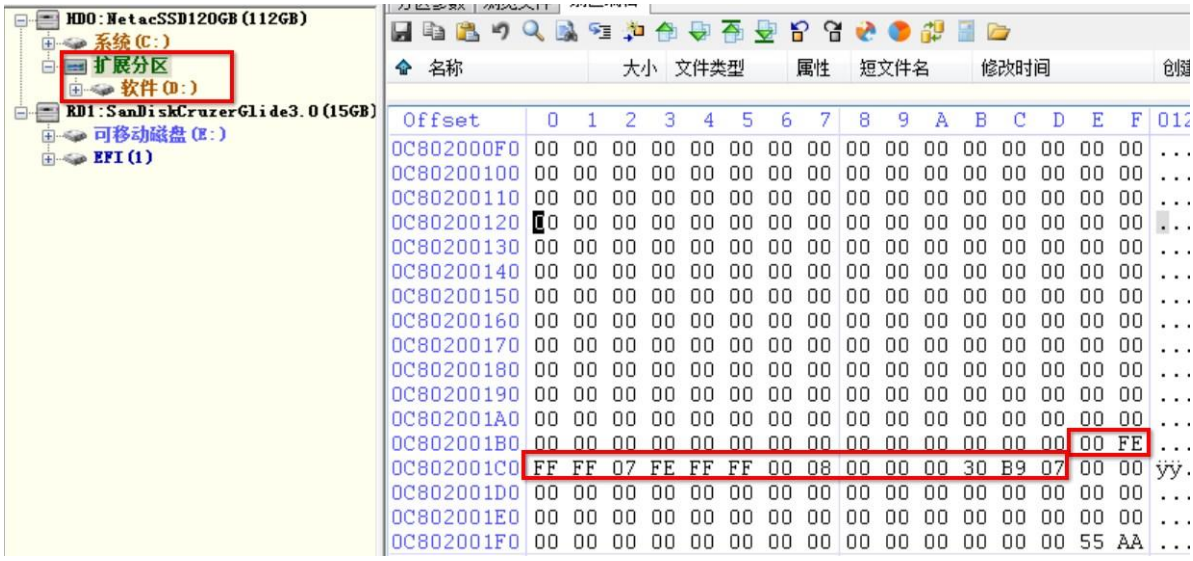
第 9 到 12 字节表示本分区之前已使用了的扇区数，使用主机字节序，所以 00 10 40 06

应该从右往左看，即 06 40 10 00 对应十进制的 104861696（扇区）

第 13 到 16 字节表示该扩展分区的总扇区数，00 38 B9 07 从右往左看是 07 B9 38 00

对应十进制的 129579008 扇区，对应大小为 129579008X512 字节（61GB）

我们查看一下该扩展分区的第一扇区（第 104861696 扇区）：



该**扩展分区**只有一个逻辑分区记录（对应 D 盘），分析一下：

00 FE FF FF 07 FE FF FF 00 08 00 00 00 30 B9 07

第一字节 00 表示此分区为非活动分区，即未激活的分区

第 2,3,4 字节表示该分区的起始地址（CHS 地址 FE FF FF 大于 8G 的硬盘通常忽略 CHS 地址）

第 5 字节表示分区类型，07 表示 NTFS 分区

第 6,7,8 字节表示该分区的结束地址（CHS 地址 FE FF FF，可忽略此地址）

第 9 到 12 字节表示本分区之前已使用了的扇区数，使用主机字节序，所以 00 08 00 00

应该从右往左看，即 00 00 08 00 对应十进制的 2048（扇区）这个地址是相对于扩展分区的，也就是说要加上扩展分区的起始扇区数 104861696

第 13 到 16 字节表示该扩展分区的总扇区数，00 30 B9 07 从右往左看是 07 B9 30 00

对应十进制的 129576960 扇区，对应大小为 61GB

第 7 章、GPT 分区表

GPT 全称 GUID 磁盘分区表（GUID Partition Table），也叫全局唯一标识磁盘分区表，它是 EFI 标准的一部分，被用于替代 MBR。

MBR 主引导主录中的分区表也叫 MBR 分区表，GPT 分区表和 MBR 分区表都是用来记录磁盘分区信息的，这是 2 种不同的分区记录方式。一个磁盘同时只能使用其中一种记录方式。这两种磁盘分区记录表本身对磁盘机械特性没有任何影响。它们只是存储于磁盘上的一段数据而已。

出于兼容性考虑，磁盘中的第一个扇区仍然用作 MBR，之后才是 GPT 头部，GPT 分区头部位于磁盘上的第 2 个扇区，大小占一个扇区，头部之后紧接着 GPT 分区表项，EFI 标准要求分区表最小要有 128 个分区项，每个分区项有 128 个字节，所以 GPT 分区表项就至少占 32 个扇区。

		GPT 分区表	
扇区：	LBA 0	LBA 1	LBA 2 至 LBA 33 （共 32 个扇区）
	MBR	GPT 头部	GPT 分区表项 1，项 2，项 3.....项 128

GPT 头部之前的 MBR 是用于防止不支持 GPT 的磁盘管理工具误识别并破坏硬盘中的数据，所以这个 MBR 也叫**保护 MBR**。保护 MBR 没有引导代码，一般也只记录了一个分区（分区类型为 0xEE），该分区大小为整个磁盘，且把分区指向了 GPT 分区表的表头，然后进一步的分区信息就由 GPT 去完成。

GPT 分区表可以记录至少 128 个分区信息，每个分区的起始和终止地址用 8 个字节的 LBA 编址（8 个字节的编址能表示 2 的 64 次方个扇区），所以它可以支持超大容量磁盘。

和 MBR 分区相比，GPT 下的分区没有激活（活动）与未激活的区分，但有隐藏与未隐藏的区分，GPT 的每个分区都是主分区，没有扩展分区的说法。且 GPT 本身不含引导代码。

大多数操作系统都能识别 GPT 分区的磁盘，但和 能从 GPT 磁盘启动是两回事儿，能不能从 GPT 磁盘启动是由预启动程序决定的，只有 UEFI BIOS 预启动程序支持从 GPT 磁盘启动，传统的 Legacy BIOS 只支持从 MBR 磁盘启动。开启了 CSM 兼容性支持模块的 UEFI BIOS 也支持从 MBR 磁盘启动。

GPT 头部各字节含义：

偏移地址 Hex	字段长度字节	含义	数据类型	数值（示例）
000	8	Signature（签名）	ascii 文本	EFI PART
008	4	Revision	Hex	00 00 01 00
00C	4	Header Size 头部大小，字节	Int	92
010	4	CRC 32 Checksum	Hex	65 c4 8a f3
018	8	Primary LBA（主 GPT 头部地址）	Int	1
020	8	Backup LBA（备份 GPT 头部地址）	Int	468862127
028	8	First Usable LBA	Int	34
030	8	Last Usable LBA	Int	468862140
038	16	Disk GUID	GUID	{65dcfe-xxxxxx}

048	4	Partition Entries LBA	Int	2
050	4	Max No. of Partitin Entries	Int	128
054	4	Size of Partition Entry	Int	128
058	4	Partition Entry Array	Hex	3f 60 f2 21

GPT 条目各字节含义：

偏移地址 Hex	字段长度字节	含义	数据类型	数值（示例）
000	16	Partition Type 分区类型	GUID	{xxxx}
010	16	Partition ID 分区 ID	GUID	{xxxd}
020	8	Starting LBA 起始地址	Int	2048
028	8	Ending LBA 结束地址	Int	1023997
030	8	Attributes 属性	Hex	01 00 xxxxx
038	72	Partition Name	Unicode TXT	EFI system partition

GPT 分区特点：

1.分区没有激活（活动的）与未激活的区分

2.分区有隐藏与未隐藏的区分

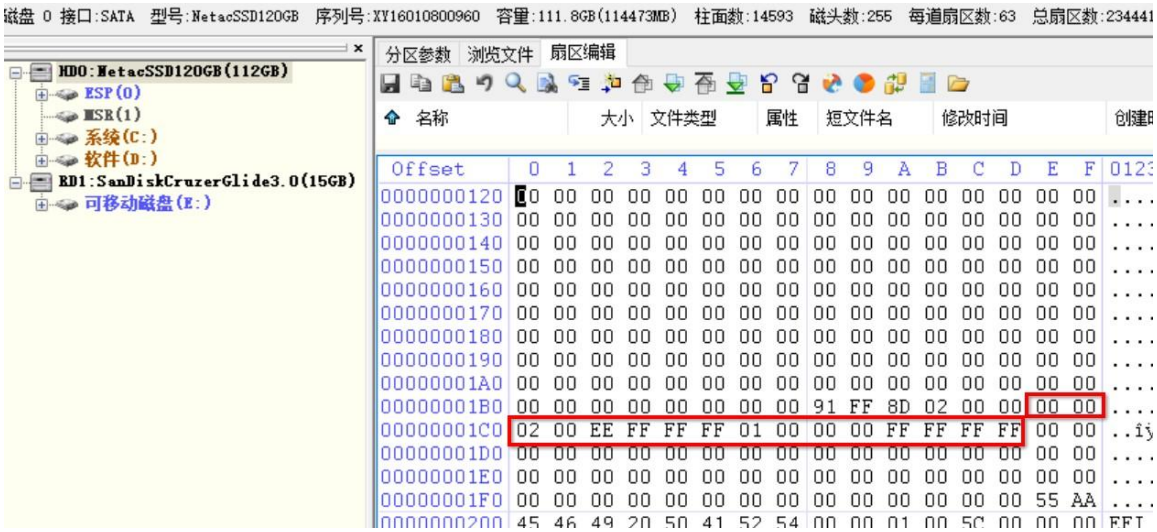
本章疑问：

GPT 分区表本身没有引导代码，那么使用 GPT 的磁盘是怎么进入系统的？

第 8 章、实战分析 GPT 分区表

在 UEFI 启动的模式下，windows 系统的分区情况一般如下：

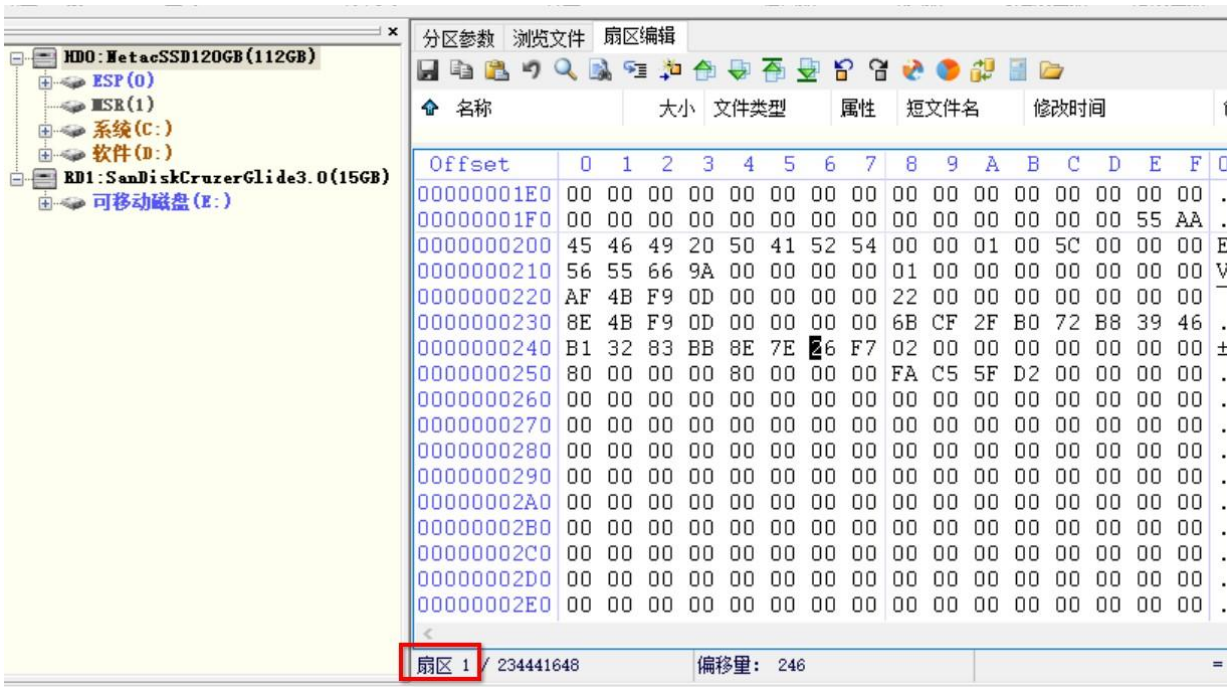
使用 DiskGenius 查看该磁盘的第一个扇区：



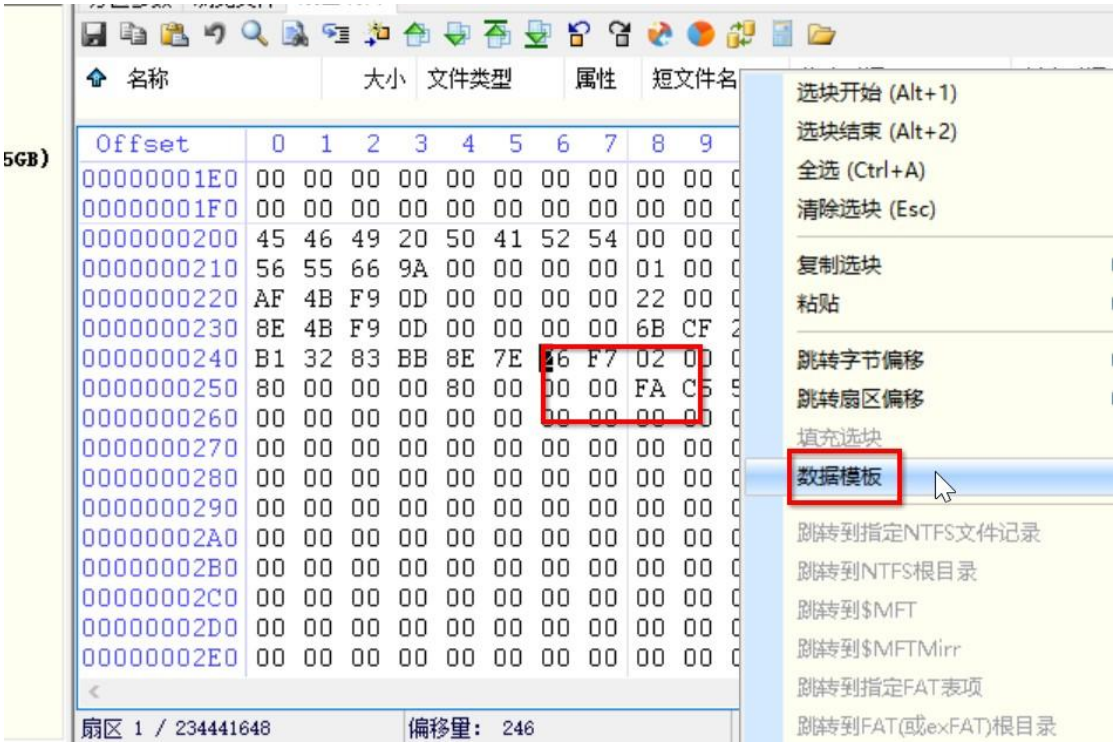
MBR 引导代码处全部为 0,没有代码，分区表项只有一项：

00 00 02 00 EE FF FF FF 01 00 00 00 FF FF FF FF

第 5 字节表示分区类型，EE 代表使用的是 GPT 分区表，后边 01 00 00 00（小端字节序）表示该分区起始扇区为 LBA 地址的 1 号扇区，就是 MBR 保护扇区紧接后的一个扇区。LBA 的 1 号扇区就是 GPT 头部，查看一下：



可以根据第 7 章的 GPT 头部参数来分析，也可以用 DiskGenius 的自带功能来查看，在 LBA 1 号扇区上点击鼠标右键，选择“数据模板”



在数据模板上选择 GUID Partition Table Header 类型，可查看详细情况：

模板类型: GUID Partition Table Header

偏移	描述	数值
0x200 (0x0)	Signature: EFI PART	EFI PART
0x208 (0x8)	Revision	00 00 01 00 (65536)
0x20C (0xC)	Header Size	92 (0x0000005C)
0x210 (0x10)	Header CRC32	56 55 66 9A (2590397782)
0x214 (0x14)	Reserved	00 00 00 00
0x218 (0x18)	LBA of This Header	1 (0x0000000000000001)
0x220 (0x20)	LBA of Alternate Header	234441647 (0x00000000DF94BAF)
0x228 (0x28)	First Usable LBA	34 (0x0000000000000022)
0x230 (0x30)	Last Usable LBA	234441614 (0x00000000DF94B8E)
0x238 (0x38)	Disk GUID	{B02FCF6B-B872-4639-B132-83BB8E7...
0x248 (0x48)	LBA of Partition Entry	2 (0x0000000000000002)
0x250 (0x50)	Number of Partition Entries	128 (0x00000080)
0x254 (0x54)	Size of Each Partition Entry	128 (0x00000080)
0x258 (0x58)	CRC32 of Partition Entry Array	FA C5 5F D2 (3529491962)
0x25C (0x5C)	Padding	00 00 00 00 00 00 00 00 00 00 00...

由上图可见，该 GPT 头部 LBA 地址为 1 号扇区，备份头部为 234441647 号扇区，分区项有 128 个，每个分区项大小为 128 字节。

在 LBA 1 号扇区（GPT 头部扇区）后紧接着就是 GPT 分区项，同样在 LBA 2 号扇区上点击鼠标右键，选择“数据模板”，查看 GUID Partition Entry Item 详细信息：

Offset	0	1	2	3	4	5	6	7	8	9
0000000400	28	73	2A	C1	1F	F8	D2	11	BA	4B
0000000410	97	5C	A1	98	C1	D3	76	43	BC	4E
0000000420	00	08	00	00	00	00	00	00	FF	67
0000000430	00	00	00	00	00	00	00	80	45	00
0000000440	73	00	79	00	73	00	74	00	65	00
0000000450	61	00	72	00	74	00	69	00	74	00
0000000460	00	00	00	00	00	00	00	00	00	00
0000000470	00	00	00	00	00	00	00	00	00	00
0000000480	16	E3	C9	E3	5C	0B	B8	4D	81	7D
0000000490	32	27	E2	67	0A	EF	3D	4E	87	89
00000004A0	00	68	09	00	00	00	00	00	FF	67
00000004B0	00	00	00	00	00	00	00	00	4D	00
00000004C0	6F	00	73	00	6F	00	66	00	74	00
00000004D0	73	00	65	00	72	00	76	00	65	00
00000004E0	61	00	72	00	74	00	69	00	74	00
00000004F0	00	00	00	00	00	00	00	00	00	00
0000000500	A2	A0	D0	EB	E5	B9	33	44	87	C0
0000000510	EF	62	AC	1F	9A	B3	D8	4F	B5	5F
0000000520	00	68	0D	00	00	00	00	00	FF	67
0000000530	00	00	00	00	00	00	00	00	42	00
0000000540	63	00	20	00	64	00	61	00	74	00

扇区 2 488397168 偏移量: 400

数据模板

模板类型: GUID Partition Entry Item

偏移	描述	数值
Partition Entry 1		
0x400 (0x0)	Partition Type GUID	{C12A7328-F81F-11D2-BA4B-00A0C93..
0x410 (0x10)	Unique Partition GUID	{98A15C97-D3C1-4376-BC4E-329CD98..
0x420 (0x20)	Starting LBA	2048 (0x0000000000000800)
0x428 (0x28)	Ending LBA	616447 (0x00000000000967FF)
0x430 (0x30)	Attribute Bits	9223372036854775808 (0x800000000..
0x438 (0x38)	Partition Name	EFI system partition
Partition Entry 2		
0x480 (0x80)	Partition Type GUID	{E3C9E316-0B5C-4DB8-817D-F92DF00..
0x490 (0x90)	Unique Partition GUID	{67E22732-EF0A-4E3D-8789-FF81C76..
0x4A0 (0xA0)	Starting LBA	616448 (0x0000000000096800)
0x4A8 (0xA8)	Ending LBA	878591 (0x00000000000D67FF)
0x4B0 (0xB0)	Attribute Bits	0 (0x0000000000000000)
0x4B8 (0xB8)	Partition Name	Microsoft reserved partition
Partition Entry 3		
0x500 (0x100)	Partition Type GUID	{EBD0A0A2-B9E5-4433-87C0-68B6B72..
0x510 (0x110)	Unique Partition GUID	{1FAC62EF-B39A-4FD8-B55F-BA8AA8F..
0x520 (0x120)	Starting LBA	878592 (0x00000000000D6800)
0x528 (0x128)	Ending LBA	269314047 (0x000000000100D67FF)
0x530 (0x130)	Attribute Bits	0 (0x0000000000000000)
0x538 (0x138)	Partition Name	Basic data partition

第 9 章、文件系统

每一个分区都需要一个文件系统 才能存储文件，文件系统就是在分区里存储数据的一种组织结构。
文件系统是以单元（也叫簇）来存储文件的，每个簇由若干个扇区组成。一个单元内不能存储多个文件的内容，哪怕一个文件只有 1 个字节的内容，它也要占用一个单元，该单元剩下的字节就浪费了。所以当需要存储大量的小文件时，最好是把单元（簇）设置得小一点。大单元的优点是读写的速度快，但是当小文件较多时总的占用空间也比较大。

★文件系统的功能：
1.管理和调度文件的存储空间
2.提供文件的逻辑结构，物理结构，存储方法
3.实现文件从标识到实际地址的映射
4.实现文件的控制操作和存取操作

各文件系统相关参数如下表：

文件系统	FAT 16	FAT 32	NTFS	exFAT	ext3	ext4	xfs
支持的操作系统	MS-DOS，Win95 及之后	Win95 OS R2 及之后的	Win 2000 及之后的	Win Vista,7 及之后的			
簇大小	1 扇区至 32KB	1 扇区至 64KB	1 扇区至 64KB	1 扇区至 32768KB		4KB	1 扇区至 64KB 受系统内存 pagezise 影响最大为 4KB
同目录最大文件数量		65535（2 字节表示）	42 亿（4 字节表示）		32000 个子目录	目录无数量限制	无数量限制
单个文件最大大小	2GB	4GB	256TB	16EB	2TB	16TB	8EB
分区最大容量	2GB	2TB（NT 内核限制为 32GB）	256 TB（受 MBR 的限制）	64ZB（受 MBR 限制）	16TB	1EB	8EB

单位换算：
1KB=1024B　　1K（Kilo）为 2^10
1MB=1024KB　　1M（Mega）为 2^20
1GB=1024MB　　1G（Giga）为 2^30
1TB=1024GB　　1T（Tera）为 2^40
1PB=1024TB　　1P（Peta）为 2^50
1EB=1024PB　　1E（Exa）为 2^60
1ZB=1024EB　　1Z（Zetta）为 2^70
1YB=1024ZB　　1Y（Yotta）为 2^80

第 10 章、分区引导扇区参数表

PBR（Partition Boot Record）分区引导记录 位于分区里的第一个逻辑扇区（这里我们把它称作分区引导扇区），共 512 个字节。由 3 部分组成：

BIOS 参数块：用于记录分区文件系统相关的参数

引导代码：MBR 引导代码执行后的第二阶段的代码，进一步引导操作系统

结束标志：最后 2 字节的 0x55AA

不同的文件系统，其参数块也是不同的，以 FAT32 和 NTFS 文件系统为例：

①**FAT32** 分区引导扇区参数表

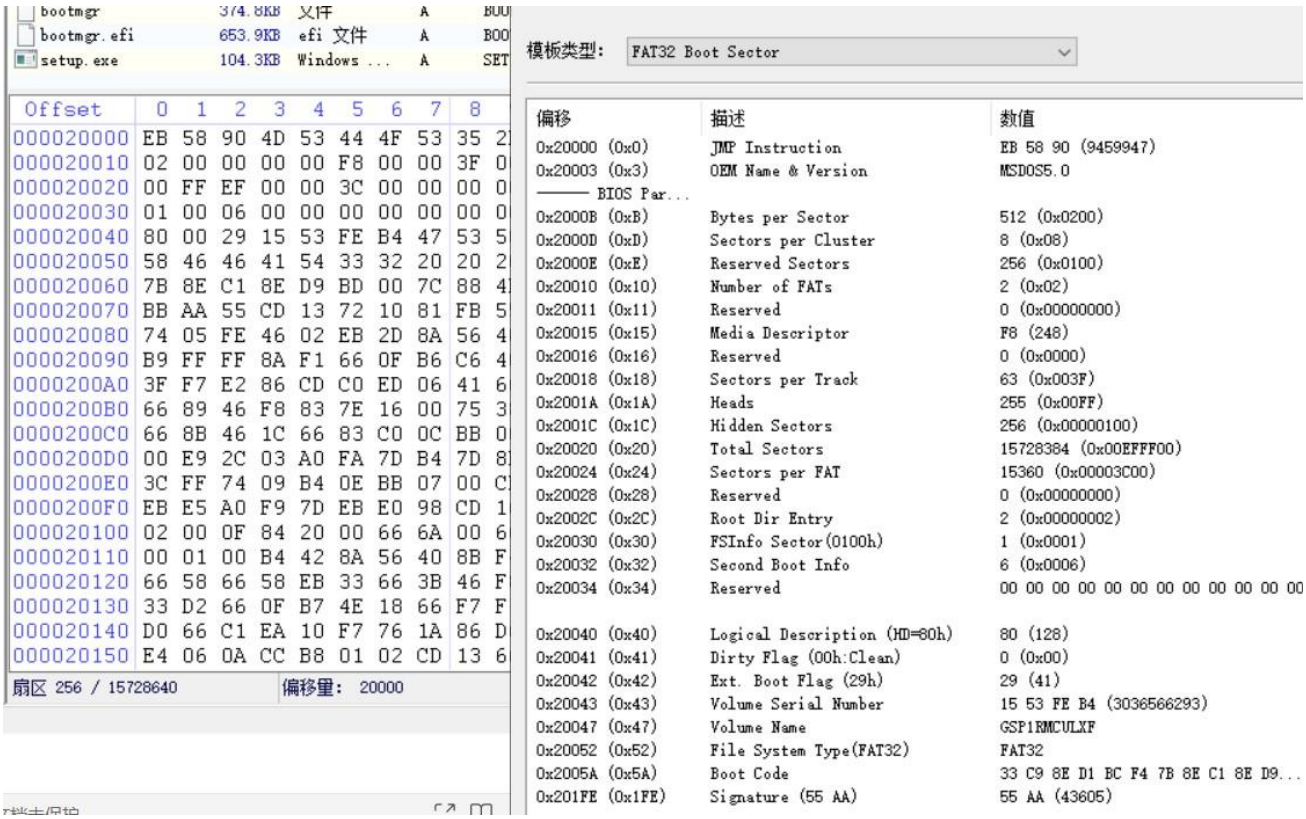
偏移地址 Hex	字段大小 byte	字段含义	数据类型	值（例）
000	3	Jump Instruction 跳转字段	Hex	EB 58 90
003	8	OEM Name & Version	Atxt	Syslinux
00B	2	Bytes per sector 每扇区字节数	Int	512
00D	1	Sector per Cluster 簇大小	Int	16 扇区
00E	2	Reserved sectors 保留扇区数	Int	256
010	1	Number of FATs（FAT 份数）	Int	2
011	4	Reserved	Hex	00 00 00 00
015	1	Media Descriptor 存储介质	Hex	F8 表示可移动存储介质
016	2	Sector per FAT(small vol)	Int	0
018	2	Sector per Track 每磁道扇区数	Int	63
01A	2	Heads 磁头数	Int	255
01C	4	Hidden Sectors	Int	256
020	4	Sectors (large vol)分区总大小	Int	30605056 扇区
024	4	Sectors per FAT	Int	15104
028	4	Reserved	Hex	00 00 00 00
02C	4	Root dir 1st Cluster	Int	2
030	2	FS Info Sector	Int	1
032	2	Backup boot sector	Int	6
034	12	Unused	Hex	00000000
040	1	BIOS driver number	Hex	80
041	1	Unused	Hex	00
042	1	Ext. boot signarture	Hex	29
043	4	Volume serial number	Hex	15 53 FE 87
047	11	Volume label 卷标	Atxt	Centos 7
052	8	File system name 文件系统名称	Atxt	FAT32
05A		Boot Code 引导代码		
1FE	2	Signature	Hex	55 AA

②**NTFS** 分区引导扇区参数表

偏移地址 Hex	字段大小 byte	字段含义	数据类型	值（例）

000	3	Jump Instructure 跳转字段	Hex	EB 52 90
003	8	File system ID 文件系统 ID	Atxt	NTFS
00B	2	Bytes per sector 每扇区字节数	Int	512
00D	1	Sectors per Cluster 簇大小	Int	8
00E	7	Reserved Sectors	Hex	0000000000000
015	1	Media Descripter 存储介质	Hex	F8
016	2	Unused	Hex	00 00
018	2	Sectors per Track 每磁道扇区数	Int	63
01A	2	Heads 磁头数	Int	255
01C	4	Hidden Sectors	Int	1261568
020	4	Unused	Hex	00 00 00 00
024	4	Logical Description(80 00 80 00)	Hex	80 00 80 00
028	8	Total Sectors 分区总大小	Int	261619488 扇区
030	8	Lcn of $MFT	Int	786432
038	8	Lcn of $MFTmirr	Int	2
040	1	FILE record size indicator	Hex	F6
041	3	Unused	Hex	00 00 00
044	1	INDX record size indicator	Hex	01
045	3	Unused	Hex	00 00 00
048	8	Serial number	Hex	xxxxx
050	4	Unused	Hex	00 00 00 00
054		Boot Code 引导代码字段		
.....				
1FE	2	Signature	Hex	55 AA

使用 DiskGenius 分别查看 FAT32 和 NTFS 分区的第一个扇区，然后对照上面的参数表分析。



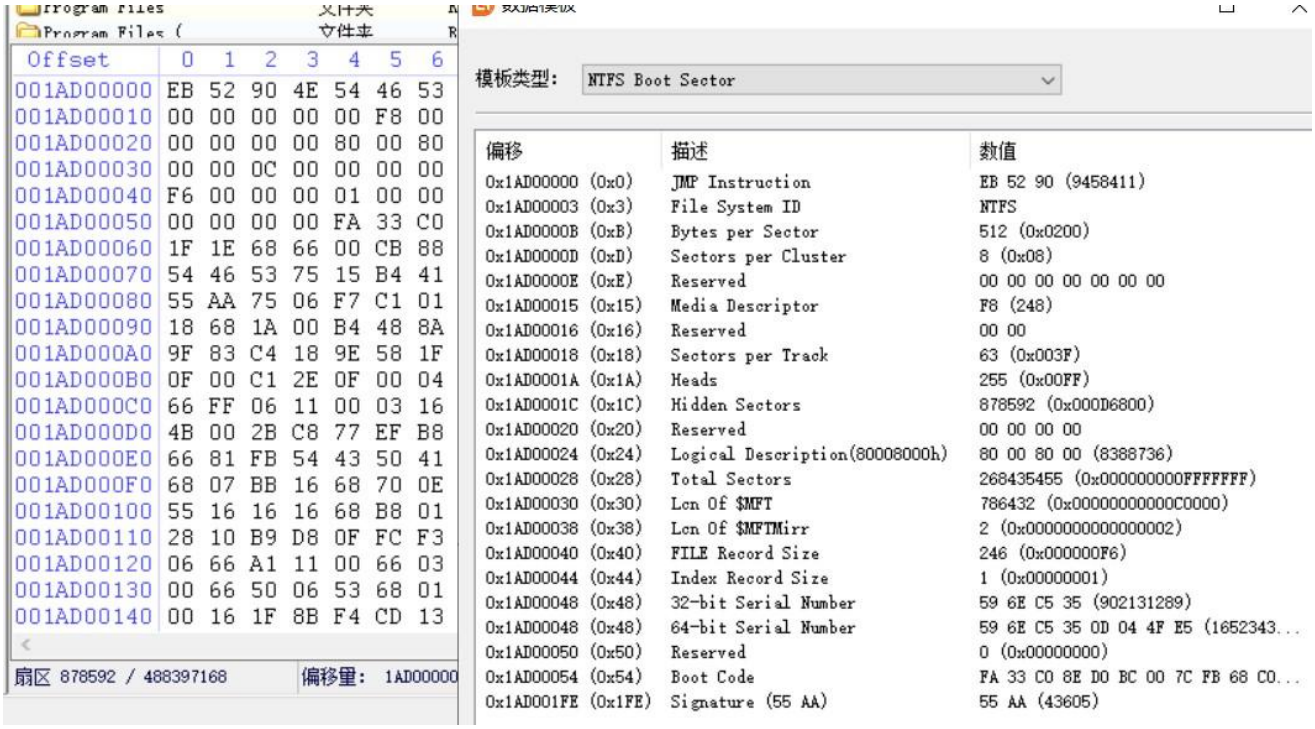
文件	大小	类型	属性
bootmgr	374.8KB	文件	A
bootmgr.efi	653.9KB	efi 文件	A
setup.exe	104.3KB	Windows ...	A

Offset	0	1	2	3	4	5	6	7	8
000020000	EB	58	90	4D	53	44	4F	53	35
000020010	02	00	00	00	00	F8	00	00	3F
000020020	00	FF	EF	00	00	3C	00	00	00
000020030	01	00	06	00	00	00	00	00	00
000020040	80	00	29	15	53	FE	B4	47	53
000020050	58	46	46	41	54	33	32	20	20
000020060	7B	8E	C1	8E	D9	BD	00	7C	88
000020070	BB	AA	55	CD	13	72	10	81	FB
000020080	74	05	FE	46	02	EB	2D	8A	56
000020090	B9	FF	FF	8A	F1	66	0F	B6	C6
0000200A0	3F	F7	E2	86	CD	C0	ED	06	41
0000200B0	66	89	46	F8	83	7E	16	00	75
0000200C0	66	8B	46	1C	66	83	C0	0C	BB
0000200D0	00	E9	2C	03	A0	FA	7D	B4	7D
0000200E0	3C	FF	74	09	B4	0E	BB	07	00
0000200F0	EB	E5	A0	F9	7D	EB	E0	98	CD
000020100	02	00	0F	84	20	00	66	6A	00
000020110	00	01	00	B4	42	8A	56	40	8B
000020120	66	58	66	58	EB	33	66	3B	46
000020130	33	D2	66	0F	B7	4E	18	66	F7
000020140	D0	66	C1	EA	10	F7	76	1A	86
000020150	E4	06	0A	CC	B8	01	02	CD	13

扇区 256 / 15728640　　偏移量: 20000

模板类型: FAT32 Boot Sector

偏移	描述	数值
0x20000 (0x0)	JMP Instruction	EB 58 90 (9459947)
0x20003 (0x3)	OEM Name & Version	MSDOS5.0
——— BIOS Par...		
0x2000B (0xB)	Bytes per Sector	512 (0x0200)
0x2000D (0xD)	Sectors per Cluster	8 (0x08)
0x2000E (0xE)	Reserved Sectors	256 (0x0100)
0x20010 (0x10)	Number of FATs	2 (0x02)
0x20011 (0x11)	Reserved	0 (0x00000000)
0x20015 (0x15)	Media Descriptor	F8 (248)
0x20016 (0x16)	Reserved	0 (0x0000)
0x20018 (0x18)	Sectors per Track	63 (0x003F)
0x2001A (0x1A)	Heads	255 (0x00FF)
0x2001C (0x1C)	Hidden Sectors	256 (0x00000100)
0x20020 (0x20)	Total Sectors	15728384 (0x00EFFF00)
0x20024 (0x24)	Sectors per FAT	15360 (0x00003C00)
0x20028 (0x28)	Reserved	0 (0x00000000)
0x2002C (0x2C)	Root Dir Entry	2 (0x00000002)
0x20030 (0x30)	FSInfo Sector(0100h)	1 (0x0001)
0x20032 (0x32)	Second Boot Info	6 (0x0006)
0x20034 (0x34)	Reserved	00 00 00 00 00 00 00 00 00 00 00 00
0x20040 (0x40)	Logical Description (HD=80h)	80 (128)
0x20041 (0x41)	Dirty Flag (00h:Clean)	0 (0x00)
0x20042 (0x42)	Ext. Boot Flag (29h)	29 (41)
0x20043 (0x43)	Volume Serial Number	15 53 FE B4 (3036566293)
0x20047 (0x47)	Volume Name	GSP1RMCULXF
0x20052 (0x52)	File System Type(FAT32)	FAT32
0x2005A (0x5A)	Boot Code	33 C9 8E D1 BC F4 7B 8E C1 8E D9...
0x201FE (0x1FE)	Signature (55 AA)	55 AA (43605)



Offset	0	1	2	3	4	5	6
001AD00000	EB	52	90	4E	54	46	53
001AD00010	00	00	00	00	00	F8	00
001AD00020	00	00	00	00	80	00	80
001AD00030	00	00	0C	00	00	00	00
001AD00040	F6	00	00	00	01	00	00
001AD00050	00	00	00	00	FA	33	C0
001AD00060	1F	1E	68	66	00	CB	88
001AD00070	54	46	53	75	15	B4	41
001AD00080	55	AA	75	06	F7	C1	01
001AD00090	18	68	1A	00	B4	48	8A
001AD000A0	9F	83	C4	18	9E	58	1F
001AD000B0	0F	00	C1	2E	0F	00	04
001AD000C0	66	FF	06	11	00	03	16
001AD000D0	4B	00	2B	C8	77	EF	B8
001AD000E0	66	81	FB	54	43	50	41
001AD000F0	68	07	BB	16	68	70	0E
001AD00100	55	16	16	16	68	B8	01
001AD00110	28	10	B9	D8	0F	FC	F3
001AD00120	06	66	A1	11	00	66	03
001AD00130	00	66	50	06	53	68	01
001AD00140	00	16	1F	8B	F4	CD	13

扇区 878592 / 488397168　　偏移量: 1AD00000

模板类型: NTFS Boot Sector

偏移	描述	数值
0x1AD00000 (0x0)	JMP Instruction	EB 52 90 (9458411)
0x1AD00003 (0x3)	File System ID	NTFS
0x1AD0000B (0xB)	Bytes per Sector	512 (0x0200)
0x1AD0000D (0xD)	Sectors per Cluster	8 (0x08)
0x1AD0000E (0xE)	Reserved	00 00 00 00 00 00 00
0x1AD00015 (0x15)	Media Descriptor	F8 (248)
0x1AD00016 (0x16)	Reserved	00 00
0x1AD00018 (0x18)	Sectors per Track	63 (0x003F)
0x1AD0001A (0x1A)	Heads	255 (0x00FF)
0x1AD0001C (0x1C)	Hidden Sectors	878592 (0x000D6800)
0x1AD00020 (0x20)	Reserved	00 00 00 00
0x1AD00024 (0x24)	Logical Description(80008000h)	80 00 80 00 (8388736)
0x1AD00028 (0x28)	Total Sectors	268435455 (0x000000000FFFFFFF)
0x1AD00030 (0x30)	Lcn Of $MFT	786432 (0x00000000000C0000)
0x1AD00038 (0x38)	Lcn Of $MFTMirr	2 (0x0000000000000002)
0x1AD00040 (0x40)	FILE Record Size	246 (0x000000F6)
0x1AD00044 (0x44)	Index Record Size	1 (0x00000001)
0x1AD00048 (0x48)	32-bit Serial Number	59 6E C5 35 (902131289)
0x1AD00048 (0x48)	64-bit Serial Number	59 6E C5 35 0D 04 4F E5 (1652343...
0x1AD00050 (0x50)	Reserved	0 (0x00000000)
0x1AD00054 (0x54)	Boot Code	FA 33 C0 8E D0 BC 00 7C FB 68 C0...
0x1AD001FE (0x1FE)	Signature (55 AA)	55 AA (43605)

第 11 章、MBR 引导种类

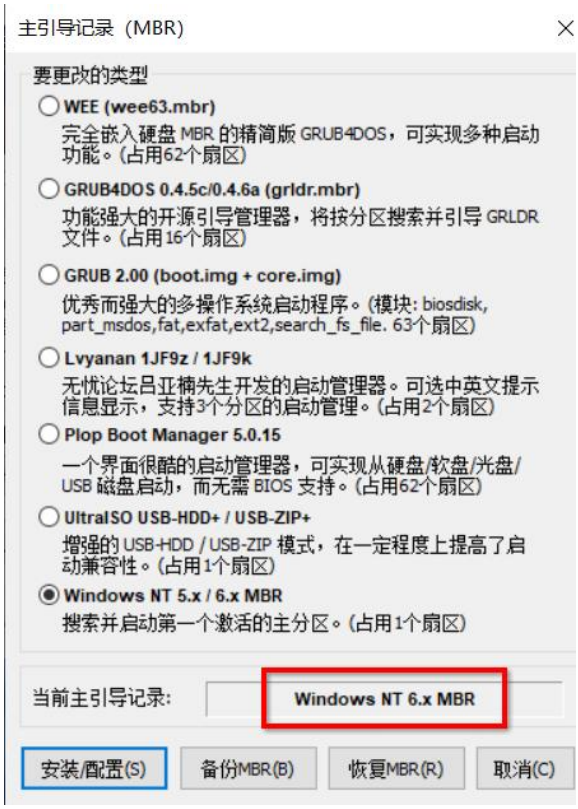
MBR 引导代码（引导程序）并不是唯一的，各个厂商及个人都可以编写自己的 MBR 引导程序，引导代码的目的就是引导第二阶段的代码或直接引导操作系统。MBR 引导代码只在传统的 BIOS 启动模式下有效，因为 UEFI 启动使用的是 GPT 分区表，而 GPT 分区表不含引导代码。

常用的引导程序如下表：

MBR 引导种类	引导大小（扇区）	说明	加载过程
USB-HDD	1	硬盘仿真模式	MBR→激活的主分区 PBR→系统启动文件
USB-HDD+	1	增强的硬盘仿真模式	MBR→激活的主分区 PBR→系统启动文件
USB-ZIP	无 MBR 扇区	大容量软盘仿真模式	分区 PBR→系统启动文件
USB-ZIP+	1	增强的 USB-ZIP	MBR→分区 PBR→系统启动文件
USB-FDD	无 MBR 扇区	软驱仿真模式	分区 PBR→系统启动文件
GRUB	18	支持多种文件系统，可引导多种操作系统	
Grub4Dos	16	可装在 MBR 上也可装在 PRB 上	MBR→GRLDR 文件
GRUB2.0	63	GRUB 的升级版	MBR→MBR_gap→分区里的其他模块
Fbinst by Bean	64		MBR→UD 区→引导文件
WEE	62	完全嵌入硬盘 MBR 的精简版 Grub4Dos	
Lvyanan 1Jf9	2	无忧论坛吕亚楠开发的	MBR→分区 PRB→引导文件
PLoP bootManager	62	支持多种介质启动，超小体积且支持图形界面	MBR→分区 PRB→引导文件
NT 5.x	1	从激活的主分区启动，有多个激活的分区就报错	MBR→分区 PRB→引导文件 NTLDR
NT 6.x	1	支持多个激活的主分区，从第一个激活的主分区启动	MBR→分区 PBR→引导文件 Bootmgr
SysLinux	1	最初用于从 FAT32 上启动 Linux 系统	

这里不作详细的讲解，将在以后的章节中细细说明。这里只是让大家看一下 MBR 的种类，明白以上各种 MBR 类型是并列的关系。以上的各种 MBR 的区别仅在于引导代码的不同，MBR 扇区的结构都是一样的（如果有 MBR 扇区的话），有的引导代码较多，会占用 MBR 扇区之后的若干个扇区。

可以使用 BootICE 工具查看磁盘的 MBR 引导类型，如上图，选择目标磁盘，点击“主引导记录”



可以看到当前选中磁盘的 MBR 引导类型为 NT6.x

第 12 章、初探 windows 系统的引导过程

经过以上几个章节的学习，大家有了一定的理论的基础，现在我们一起来探索一下 Windows 系统的引导过程。先回到第 0 章，计算机的启动过程。
那里讲解到 BIOS 把系统控制权交给 MBR 引导代码，再由 MBR 引导代码做进一步的引导操作。接下来以 Windows 7 和 Windows 10 为例讲解一下。

①Windows 7

使用 Win7 的计算机一般是用传统的 BIOS 预启动程序进行启动，然后根据主板 CMOS 的配置记录，选择默认的启动磁盘，再从启动磁盘里读取第一个扇区（MBR 扇区），Windows 7 的 MBR 引导代码为 NT6.1，NT6.1 引导主要负责检查磁盘分区表的正确性，然后再把引导权交给第一个激活的分区的分区引导（PBR），一般是 NTFS 分区的引导，NTFS 分区引导再查找本分区根目录下的 bootmgr 文件，把控制权交给 bootmgr，然后分区引导的任务完成。

BOOTMGR 文件（不区分大小写）也是一个引导文件，比较高级的，做的事情比较多，我们没必要了解太多的细节，它主要是去查找系统启动项文件，一般是去找它所在分区里的\boot 目录下的 BCD 文件，BCD 文件就是 Boot Configuration Data，里面记录了 windows 系统的每一个启动项。

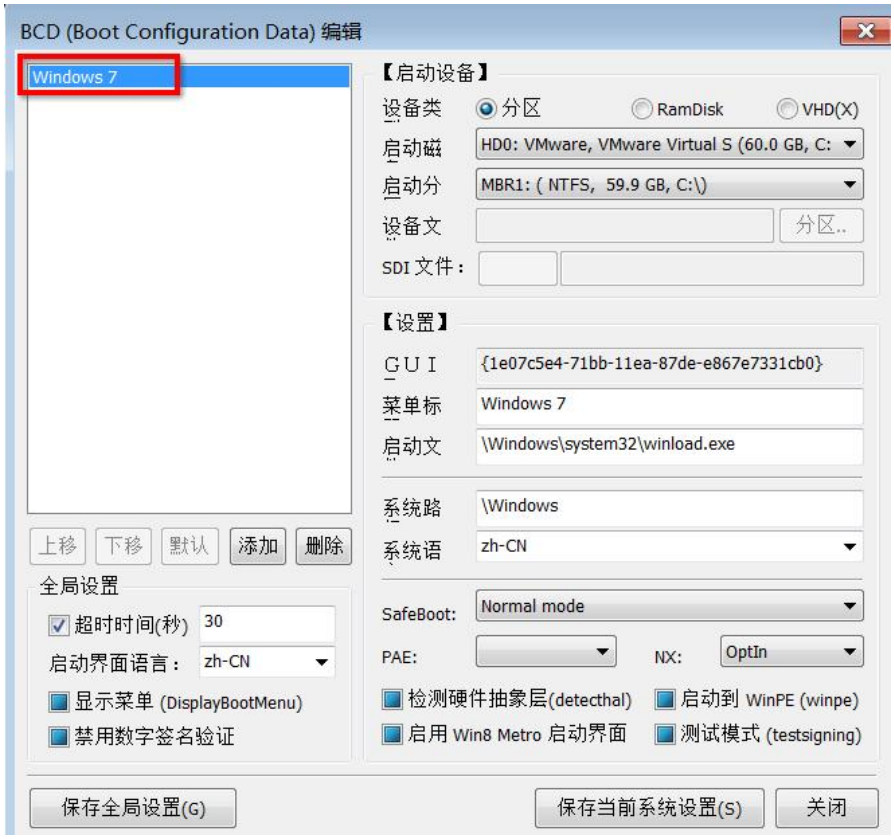
使用 DiskGenius 工具查看一下 Windows7 系统磁盘的分区情况：

上图可见，第一个激活的分区不一定是 C 盘，而是系统保留这个分区，它的属性是 A（激活的），所以 MBR 引导会去找该分区里的 bootmgr 文件，并执行它。

我们再进入“系统保留”分区，浏览文件：

上图可见，“系统保留”分区下有 bootmgr 文件，也有一个 Boot 目录，Boot 目录里有 BCD 文件，还有一些 zh-cn，zh-tw 之类的文件夹，那是系统语言相关的文件。

bootmgr 读取了 BCD 启动项文件后，如果只有一个启动项，系统直接进入默认的启动项对应的系统，如果有多个启动项，会给几秒种的时间让用户去选择要进入的系统。使用 Bootice 工具查看一下 BCD 的相关参数：



上图可见，该 BCD 只有一个启动项，名为 Windows 7，启动磁盘为 HD0，设备类型为分区，启动分区为 C 盘，启动文件为 C 盘里的\Windows\system32\winload.exe

系统语言为 zh-CN，当 bootmgr 执行了该默认的 Windows 7 启动项后，就正式地把计算机的所有权交给了 winload.exe 程序，windows 启动。

Win 7 系统启动过程	
第 1 步	BIOS 预启动程序读取默认启动磁盘的 MBR 扇区
第 2 步	MBR 引导代码（NT6.1）选择从第一个激活的分区启动
第 3 步	第一个激活的分区引导 PBR 查找根目录下的 bootmgr 文件
第 4 步	bootmgr 代码读取\boot\BCD 启动项配置文件

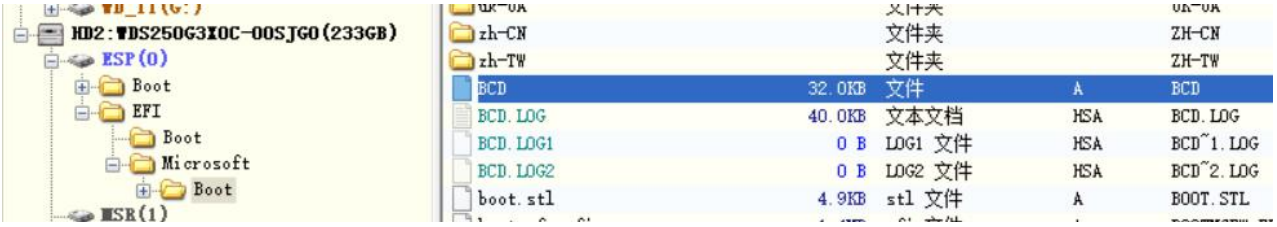
第 5 步	用户选择 BCD 启动项（只有一个启动项时，自动选择），然后以 BCD 启动项指定的方式启动操作系统（C 盘里的\Windows\system32\winload.exe 程序）

②Windows 10

使用 Win10 系统的计算机一般是用 UEFI 预启动程序进行启动，然后根据主板 CMOS 的配置记录，选择默认的启动磁盘，再从启动磁盘里读取第一个扇区（保护 MBR 扇区），一般情况下这个保护 MBR 扇区没有引导代码，分区记录也只有一项，分区类型为 EE，UEFI 程序读取到分区类型为 EE 这个信息时，就明白了该磁盘使用的是 GPT 分区表，然后读取第 2 个扇区（LBA 地址为 1 的扇区）即 GPT 头部，头部会有一些 CRC 校验，校验通过后，再去读取接下来的 GPT 分区项，从 LBA2 到 LBA34 共 32 个扇区，UEFI 预启动程序默认会选择执行第一个 ESP 分区的引导文件，引导文件在哪里呢？经过作者的探究，终于明白了。它不是去执行 bootmgr，以前我一直以为 UEFI 模式下的 windows 也是去执行 bootmgr，直到遇到下面的情况：

如上图，这个 ESP 分区里根本就没有 bootmgr 文件，这也证实了从 UEFI 启动的 windows 不需要 bootmgr 文件，因为 UEFI 预启动程序会主动去寻找\EFI\boot\目录下的 bootx64.efi 文件，然后它再读取 BCD 文件，

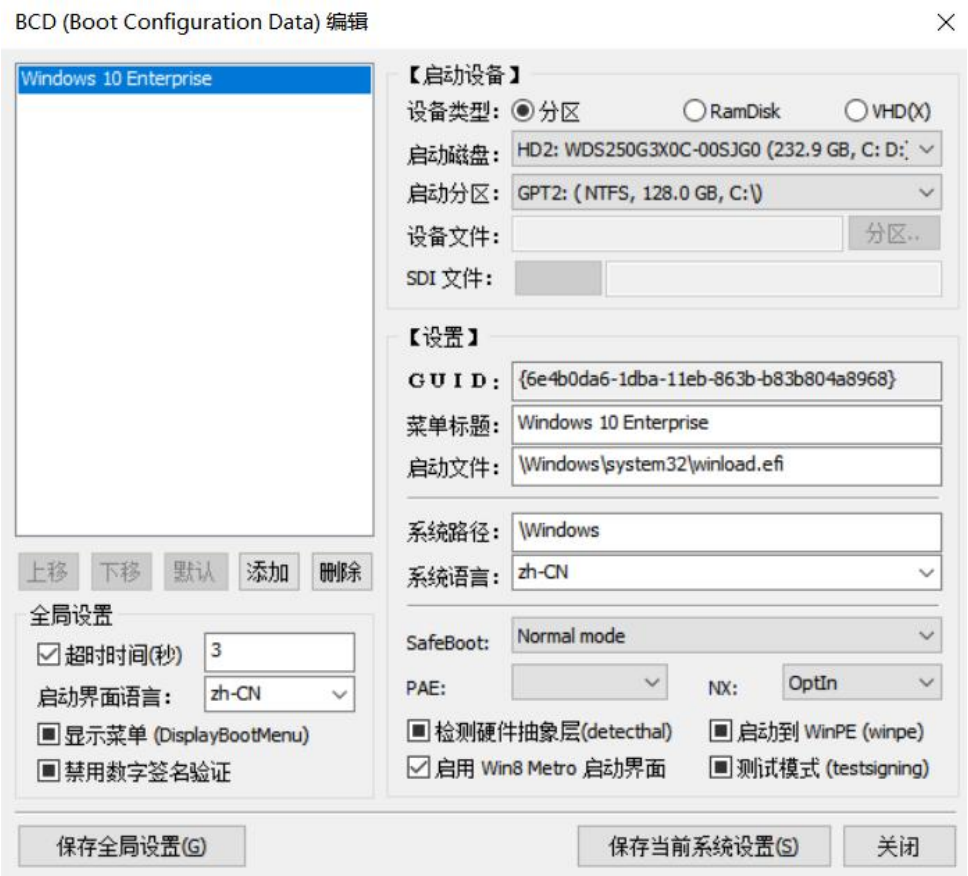
UEFI 启动下的 BCD 文件不在\boot 目录下，而是在\EFI\Microsoft\Boot\目录里。



最后根据 BCD 启动项里的方式去进一步启动系统。

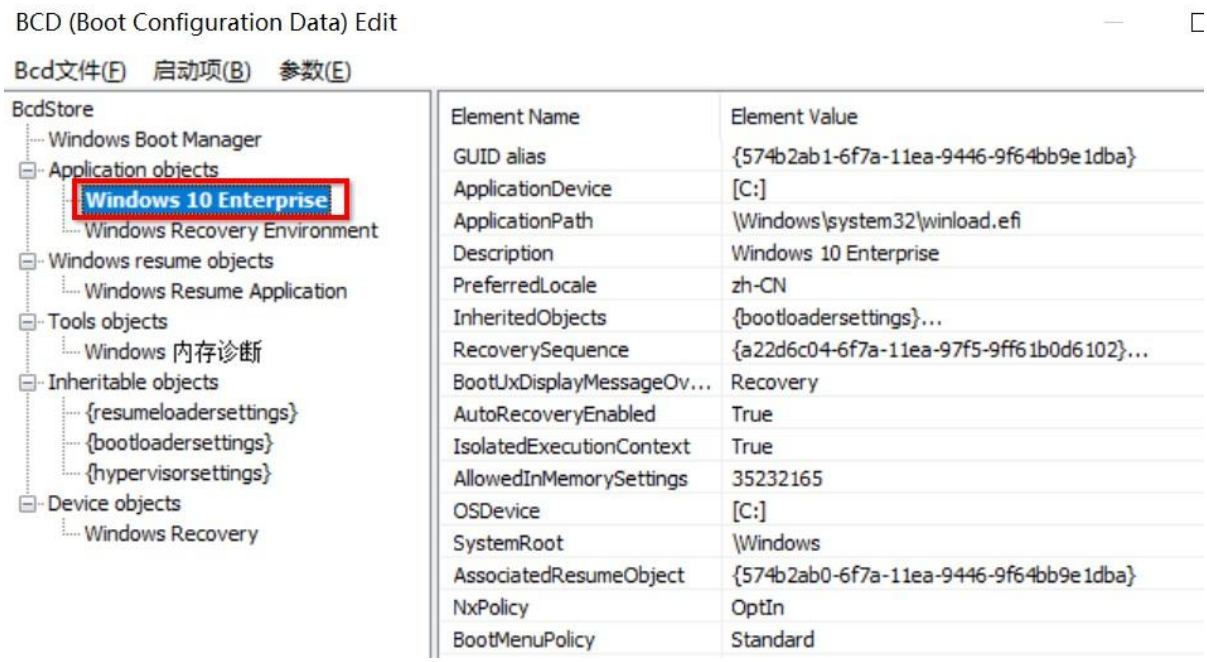
上面也解答了第 7 章的疑惑，GPT 分区表没有引导代码了，系统是怎么引导的？原来是直接从 UEFI 程序跳到了 ESP 分区上的\EFI\boot\bootx64.efi 文件了，相比传统 BIOS 启动，UEFI 启动就少了 MBR 引导这一步。

使用 BootIce 查看一下 win10 的 BCD 启动项参数是怎么样的：



上图可见，该 BCD 也只有一个默认的启动项，启动文件为 C 盘里的\Windows\system32**winload.efi** 文件

使用 BootIce 的高级编辑模式可以查看更为详细的信息



Win 10 系统启动过程	
第 1 步	UEFI 预启动程序读取默认启动磁盘的 GPT 分区表
第 2 步	UEFI 预启动程序选择从第一个 ESP 分区启动
第 3 步	执行第一个 ESP 分区\EFI\boot\bootx64.efi 引导文件
第 4 步	bootx64.efi 读取\EFI\Microsoft\Boot\BCD 启动项配置文件
第 5 步	用户选择 BCD 启动项（只有一个启动项时，自动选择），然后以 BCD 启动项指定的方式启动操作系统（一般为 C 盘里的\Windows\system32\winload.efi 程序）

第 13 章、装系统的方式

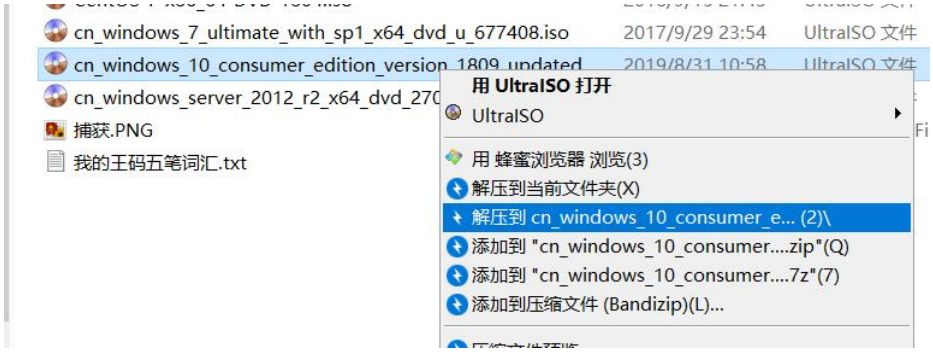
经过前面十几章的理论学习，我们要开始研究一下该怎么装系统了。操作系统有很多，我们由简入难，先学一下 Windows 7 和 Windows 10 的安装方法。

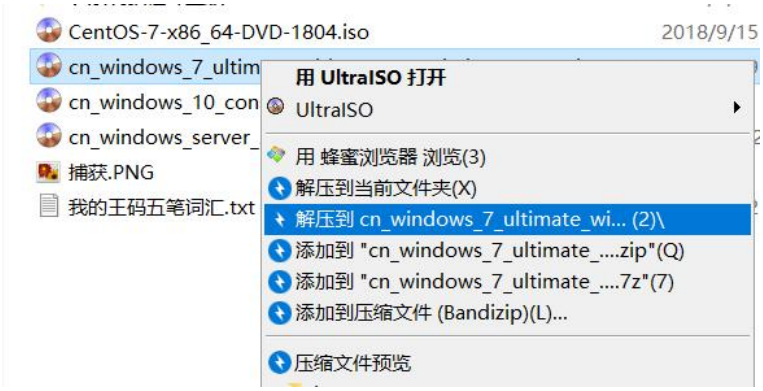
首先要获得系统安装包，可以到“CSDN 我告诉你”网站去下载官方原版系统

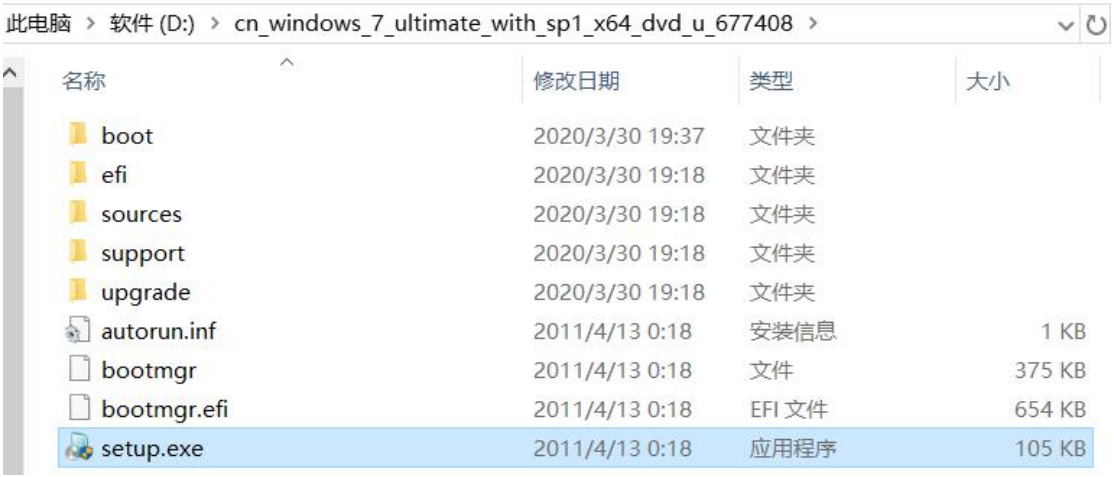
下载 win7 和 win10 系统安装包（文件格式为.iso 光盘镜像文件）到本地 D 盘上，安装包有了，具体怎么安装到计算机上呢？常用的装系统方法**有 3 种**。

①把 iso 文件解压到某目录下，然后进入解压目录，双击 Setup.exe 文件，进入安装向导

（该方法的前提是目标计算机能进入现有的 windows 系统）



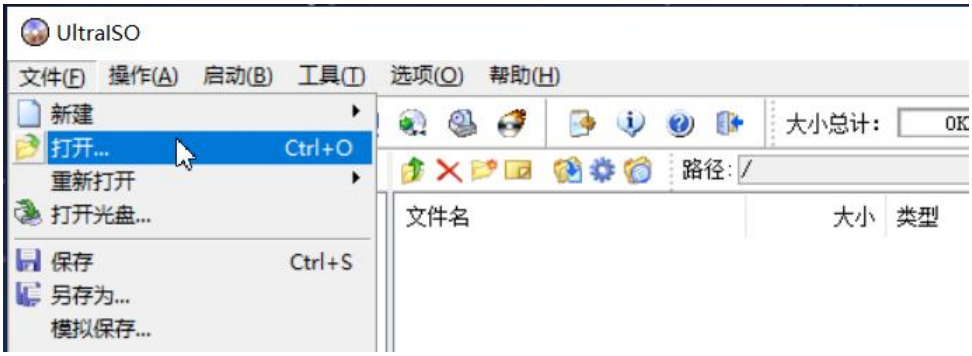




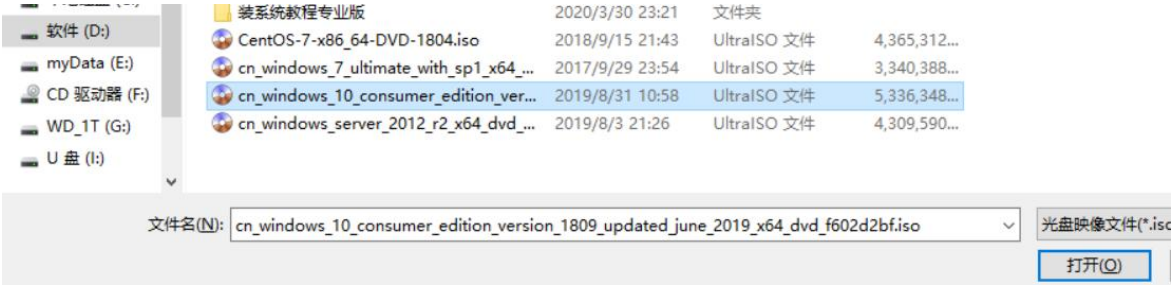
②使用 UltraISO 刻录工具把 iso 文件刻录进 U 盘，然后从 U 盘启动进行安装

先在能进入 windows 系统的计算机上安装 UltraISO 软件，软件可到官网下载 https://cn.ezbsystems.com/ultraiso

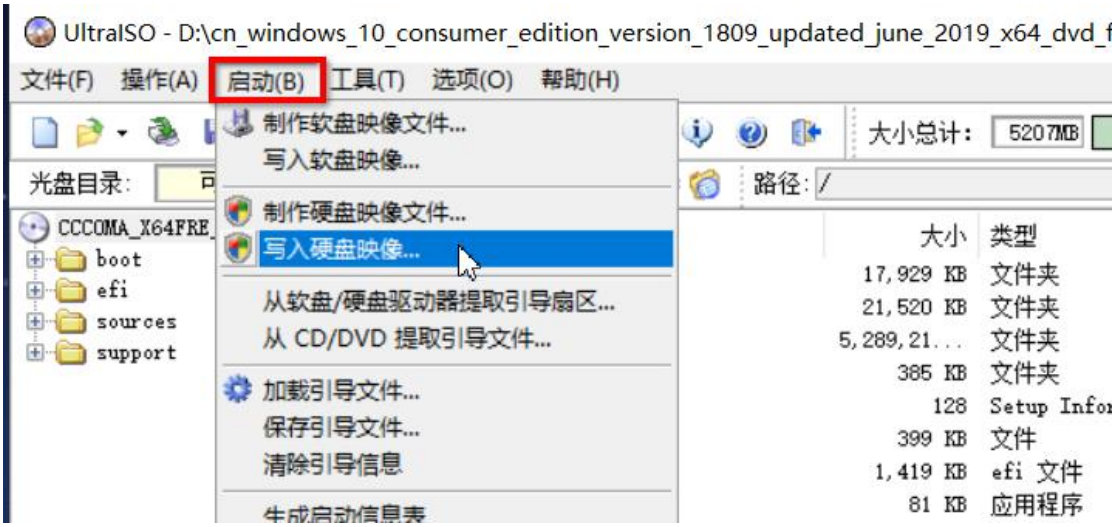
安装后，打开 Ultraiso 软件，



点击左上角的“文件”→“打开”，



选择目标安装包 iso 文件，



然后点击菜单栏的“启动”→“写入硬盘映像”

选择目标 U 盘（该 **U 盘里的资料要先备份**），写入方式为 USB-HDD+，然后点击“写入”，等几分钟，完成。

然后把该安装 U 盘插入目标计算机，启动计算机时选择从 U 盘启动，就可以进入安装系统向导了。

③制作 PE 启动 U 盘，进入 PE 系统后，使用其他软件进行安装

制作 PE 启动 U 盘的工具软件有很多，常见的有：

可以到相应的官网去下载，安装，然后插入 U 盘，进行 PE 启动盘的制作。

这些 PE 制作工具软件功能上都差不多，使用的 PE 系统也基本上是差不多的。


U大师-U盘启动盘制作工具（4.7.41.83）
U盘启动
本地模式
快速备份
PE
PE工具
硬件检测
帮助中心
官网首页
U大师
www.udashi.com
选择U盘
RM3: SanDisk Cruzer Glide 3.0 (14.5 GE
文件系统
NTFS
FAT
个性化设置
默认设置
个性设置
开始制作


无法复制大于4G文件怎么办
U盘升级
模拟启动
格式化U盘
启动查询
微PE工具箱 64 2.0
立即安装进系统
已充分阅读 安装指南与协议
其它安装方式 >
安装PE到U盘


安装PE到U盘
安装方法
方案一：UEFI/Legacy全能三分区方式(推荐)
帮助
待写入U盘
(hd3)I: SanDisk Cruzer Glide 3.0 (14.6GB)
格式化
NTFS
USB-HDD
U盘卷标
微PE工具箱
PE壁纸(可选)
包含DOS工具箱（仅Legacy启动可用）
个性化盘符图标
同时复制安装包
购买微PE优盘
立即安装进U盘
返回上一步


U盘启动
使用教程
官方网站
U盘商城
老毛桃
laomaotao.org
默认模式
ISO模式
本地模式
升级启动U盘
归还空间
格式转换
模拟启动
快捷键查询
选择设备：
I:(hd3)SanDiskCruzer_Glide_3.0 (13.71G)
写入模式：
HDD-FAT32
ZIP-FAT32
HDD-FAT16
ZIP-FAT16
U盘分区：
智能模式
兼容模式
增强模式
支持UEFI启动
个性化设置：
默认设置
高级设置
一键制作
在线客服
如何使用默认模式？


U盘启动
备份/还原
帮助中心
装机必备
官方网站
U大侠
默认模式(隐藏分区)
ISO模式(深度隐藏)
本地急救(无需U盘)
选择设备：
I: (hd3) SanDiskCruzer Glide 3.0 (U盘) 13.71GB
一般参数：
NTFS
U盘模式：
HDD-FAT32
系统默认
制作启动U盘
高级设置
本机U盘启动快捷键
F11/ESC
格式转化
升级启动盘
归还空间
模拟启动
快捷键查询
计算机系统版本：Win10 64
人工服务
软件版本号：4.2.26.1224

操作界面都差不多，有些不支持生成 NTFS 的分区，所以不能把大于 4GB 的 iso 镜像文件复制到 U 盘里。
推荐使用支持生成 NTFS 分区的工具去制作 PE 启动盘。
记得 **U 盘里的资料要先备份！！！**

制作完成后，就可以把 U 盘插到目标计算机上，启动时选择从 U 盘启动，然后就进入启动项，选择启动 win10PE 或 win8PE，然后到 PE 里，使用工具安装，具体的操作会单独做成一章来讲解。

小结：
各安装方式的优劣：
1.直接解压后，双击 Setup.exe 安装的方式不能用于新组装的计算机和无法进入系统的计算机
2.使用 Ultraiso 刻录到 U 盘，从 U 盘启动安装的，要安装其他的系统时，得重新刻录
3.使用 PE 启动 U 盘，进入 PE 系统，在 PE 系统里使用工具安装的方式：各 PE 软件商自带的那个一键安装、还原之类的软件，往往在安装系统时会夹带捆绑软件和篡改主页。所以在进入 PE 系统后，不要用自带的一键安装的工具去安装系统，推荐使用 WinNTsetup.exe 工具

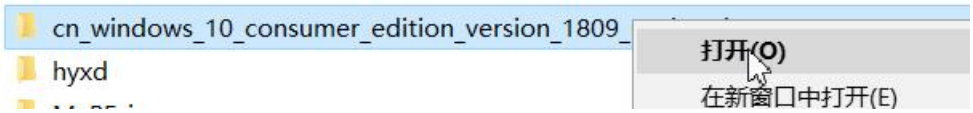
使用光盘安装和 PXE 网络启动安装，这 2 个方式本章先不讲了。

第 14 章、iso 镜像文件的结构及安装的原理

系统安装包一般为.iso 格式的文件，它是一个光盘镜像文件（相当于一个虚拟的光盘，有引导扇区的），里面究竟有些什么文件呢？安装系统时究竟是调用了哪些文件呢？我们这章来一探究竟。

一般的解压缩工具也都能认识.iso 文件，可以直接解压到某个目录下，我们先用解压缩工具解压一下 win10 的.iso 镜像文件

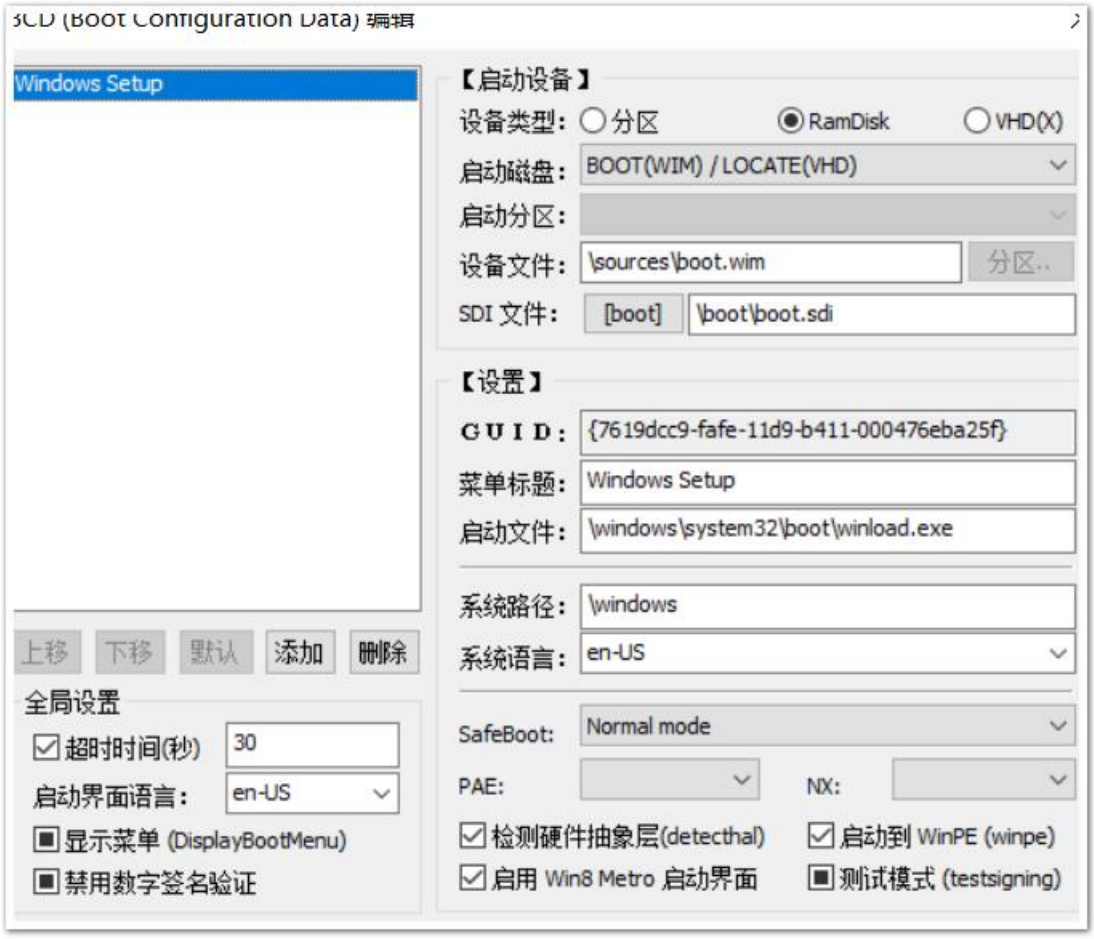
解压完成后，进入解压目录查看一下具体有些什么文件：



上图可见，iso 镜像文件里有 bootmgr 启动文件，有 setup.exe 安装向导程序，还有\boot 和\efi 目录，根据第 12 章的知识，这两个目录一般是 BCD 文件所在的目录，这两个目录也说明该镜像文件是可以从 BIOS 和 UEFI 预启动程序里启动的，查看一下这 2 个目录下的 BCD 文件，看看默认的启动项是怎么个启动方式。

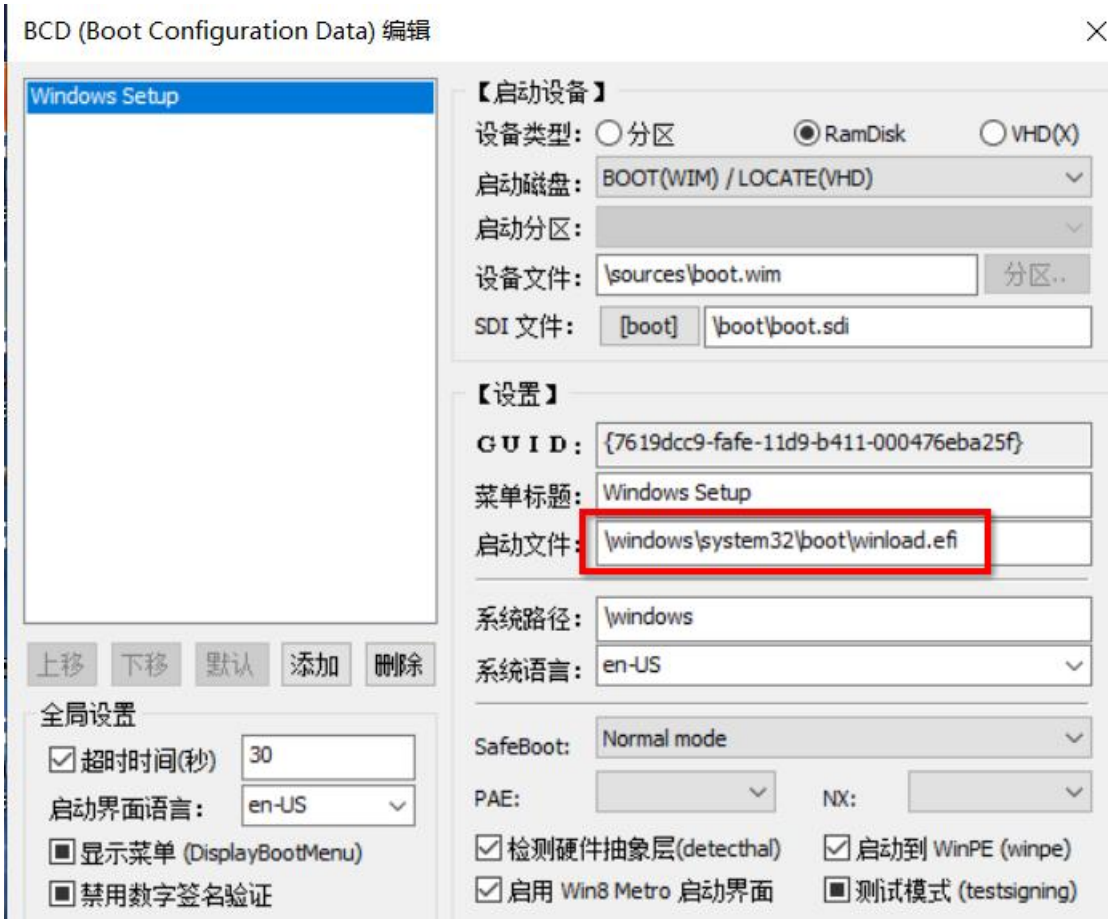
使用 Bootice 工具查看，BCD 编辑项里，选择“其他 BCD 文件”，选中解压目录里的**\boot\bcd** 文件，使用“智能编辑模式”查看

查看结果如下：



上图中，启动项名为 Windows Setup（windows 安装），启动设备类型为 RamDisk（从内存中虚拟出来的磁盘），设备文件为\sources\boot.wim 文件，SDI 文件为\boot\boot.sdi 文件。启动文件为\windows\system32\boot\winload.exe

\efi\microsoft\boot\bcd 文件查看的结果如下，只有一点不同，就是启动文件不同，传统的的 bios 启动是从\boot\bcd 菜单启动的，系统装载文件是 winload.**exe** 而 UEFI 启动是从\EFI\microsoft\boot\bcd 菜单启动的，系统装载文件是 winload.**efi**



设备文件和 SDI 文件路径前没有盘符，不用加盘符，它就是相对于启动磁盘分区而言的。

什么是 RamDisk，boot.wim，boot.sdi ？

WIM 是 Microsoft Windows Imaging format 的简称，.wim 是一种映像文件格式，wim 文件里存储着一个或多个操作系统的映像。

也就是说，只要用恰当的工具，把.wim 文件里的操作系统映像解压到某个分区里，该分区就成了系统分区了，可以设置引导，去该分区里启动操作系统。

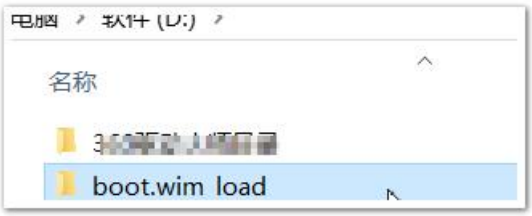
SDI 是 System Deployment Image 的简称，sdi 文件是一个虚拟磁盘文件，就是在内存里模拟出一个磁盘分区来，boot.wim 文件里的映像就是挂载（解压）到 boot.sdi 虚拟磁盘里的，该虚拟磁盘的各参数是由 bootmgr 去指定的，它的盘符一般显示为 X:盘，这个 X 盘就是 **RamDisk**

boot.wim 映像解压到由 boot.sdi 虚拟出的 X 盘里，然后就启动该 X 盘里的\windows\system32\boot\winload.exe 文件，从而启动操作系统，不过这个系统是精简的功能有限的 windows 系统，它的主要功能是用来安装正式的 windows 操作系统，所以 boot.wim 里的这个精简的系统也叫 windows PE 系统（windows 预安装环境，极度精简的 windows 系统）

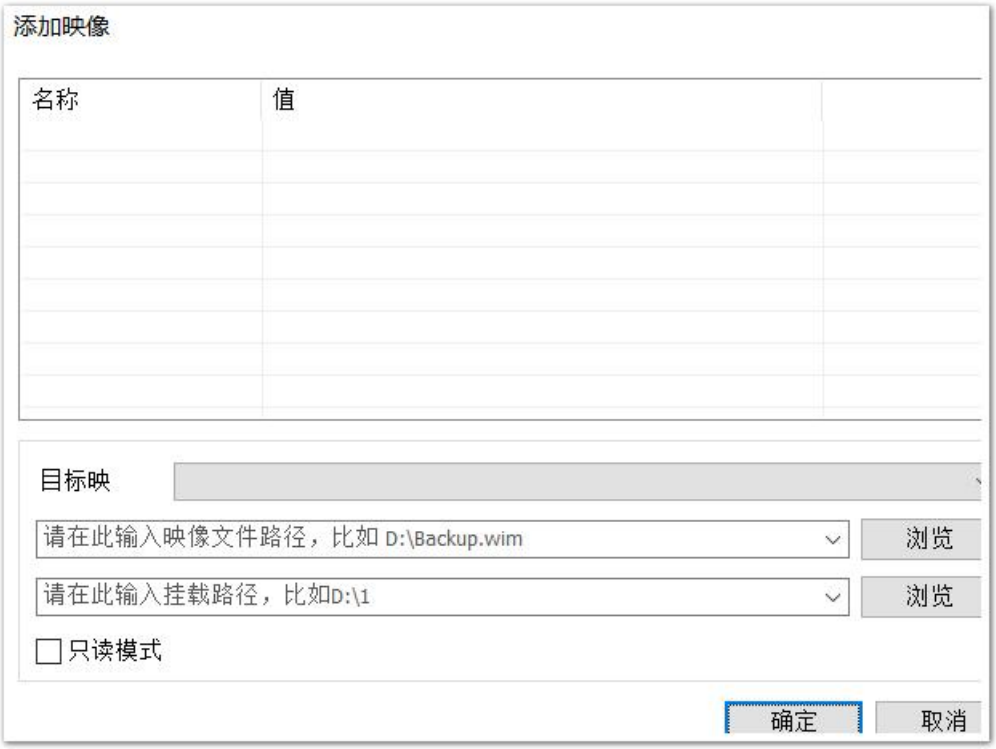
\sources\boot.wim 文件里的映像是 PE 系统，用于安装操作系统的，那么正式的操作系统在哪里呢？

我们要安装的正式的操作系统在\sources**install.wim** 文件里，.wim 格式的文件可以用 WimTool 和 Dism++工具去编辑（这里先使用 Dism++）

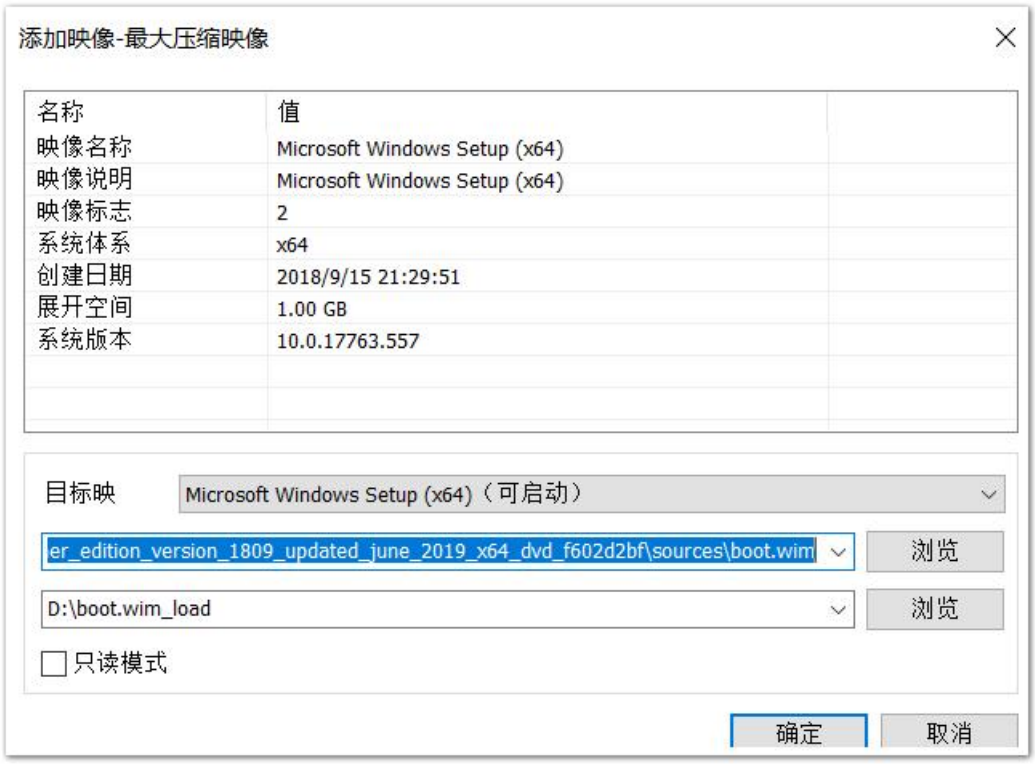
用 Dism++工具软件去查看一下 boot.wim 里面到底有什么，
先在 D 盘创建一个文件夹，名称为 boot.wim_load



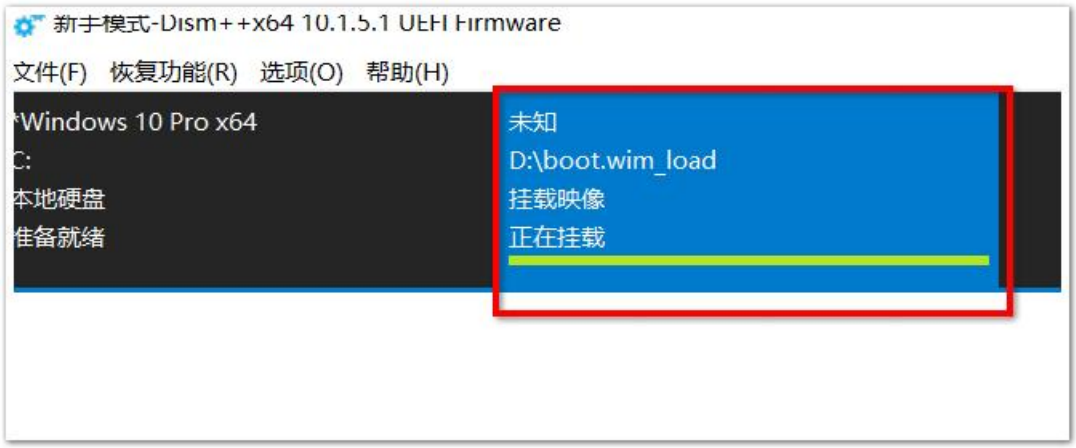
然后，打开 Dism++软件，点击菜单栏的“文件”→“挂载映像”



在弹出的添加映像对话框中，选择映像文件为 win10 ISO 文件解压目录下的\sources\boot.wim 文件，挂载路径为刚刚在 D 盘创建的 boot.wim_load 文件夹，目标映像为 Microsoft Windows Setup (x64)

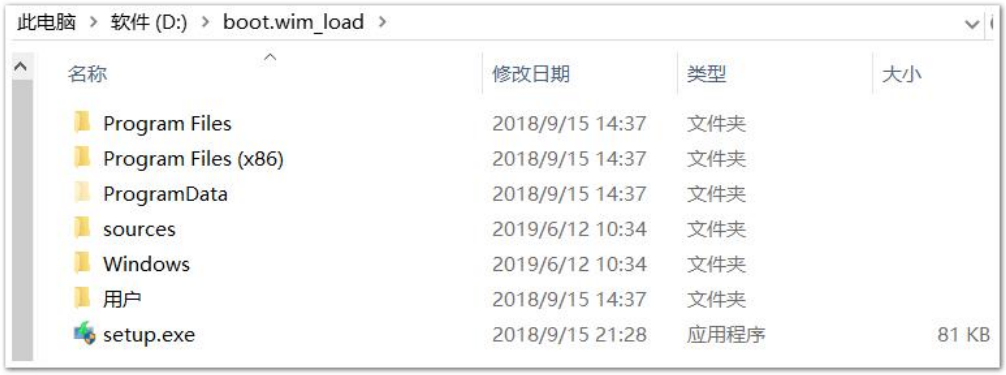


然后点击“确定”，Dism++的主界面上会显示未知的映像，正在挂载，



挂载完成后，点击下面空白处的“打开会话”

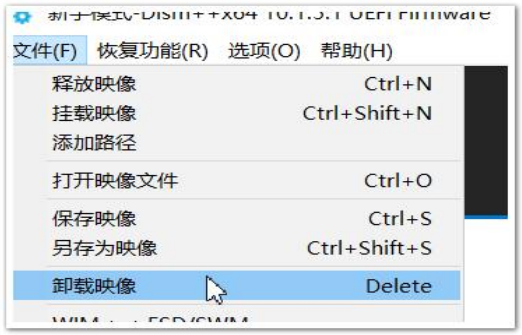
我们暂时不管这个 Dism++软件了，直接去 D 盘的 boot.wim_load 文件夹看看



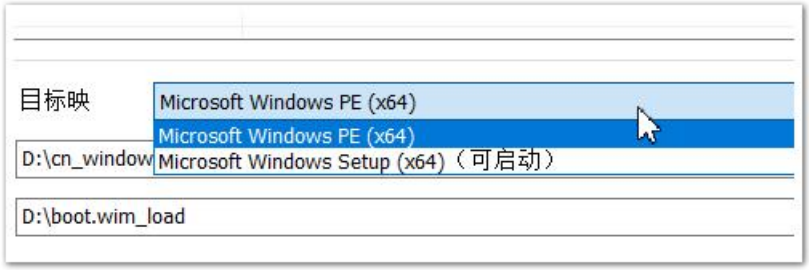
上图可见，boot.wim 的挂载目录结构和我们系统盘 C 盘里的差不多，这就是 windows 安装光盘镜像文件里的 PE 系统，它启动后，直接运行 setup.exe 程序，该程序就是我们在安装系统时的安装向导，如下图：

安装向导再根据用户的设置（要安装的系统版本及相关参数等）去部署 install.wim 文件里的操作系统到目标磁盘 C 盘里。

其他的这里不深入讲解，回到 Dism++软件的主界面，点击菜单栏的“文件”→“卸载映像”，然后 D 盘的 boot.wim_load 文件夹又变为空了，里面的文件没了。

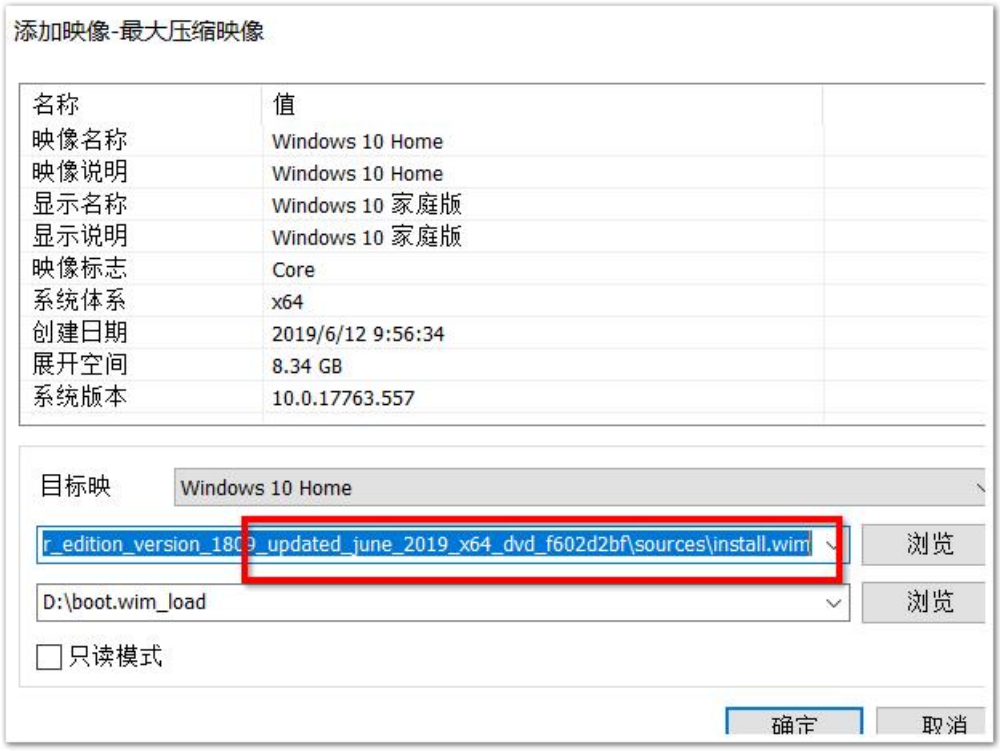


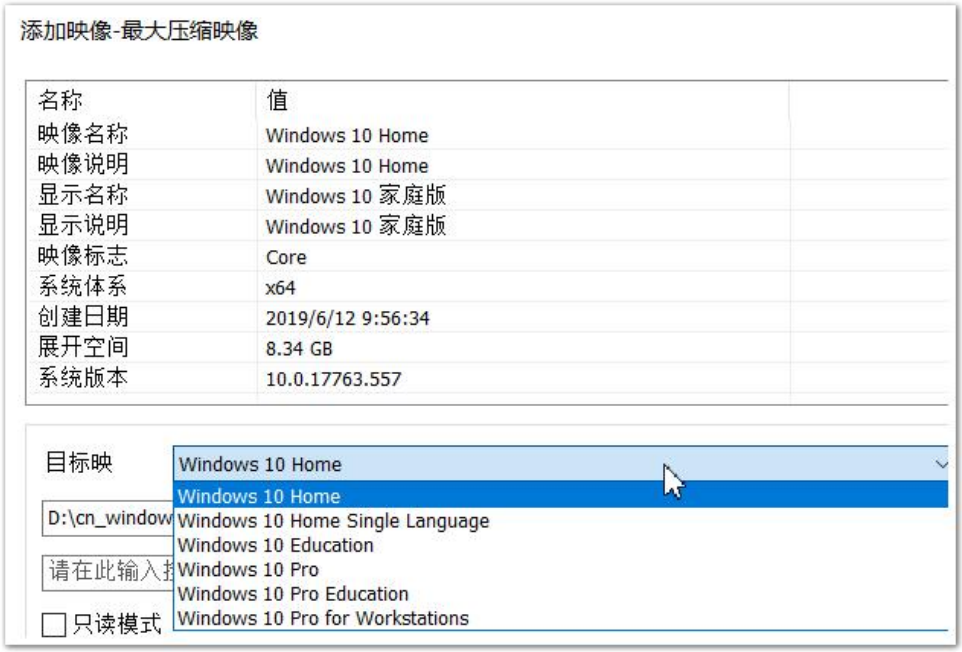
到这里基本了解了 boot.wim 文件的内容了。不过刚刚在挂载映像时，我们选择的目标映像为 Microsoft Windows Setup (x64)，



boot.wim 里可不止一个映像，我们怎么确定 BCD 文件里的启动目标映像就是 Microsoft Windows Setup(x64)这个呢？BCD 文件里的启动项里并没有说明从哪个映像启动啊，
因为就只有它显示是可启动的。这个映像是可以加载到 SDI 虚拟磁盘里，并启动此 PE 系统。

我们最后再看看\sources\install.wim 文件里有几个映像，各自的情况如何。
同上，点击菜单栏的“文件”→“挂载映像”，添加映像文件为 win10 ISO 解压目录里的\sources\install.wim





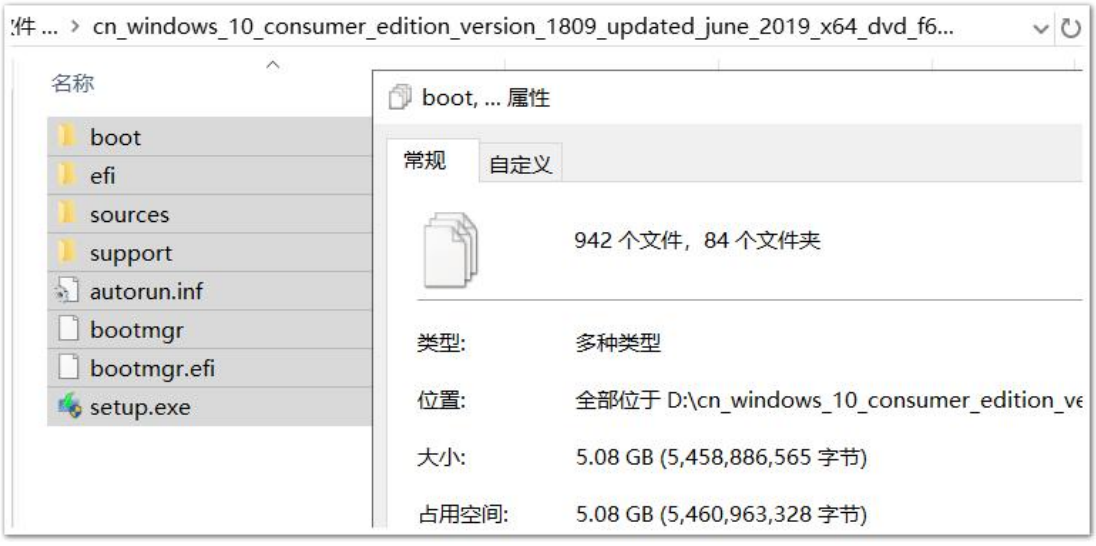
上图可见，该 windows 安装光盘里的 install.wim 文件里有 6 个版本的 windows 系统可供安装，每个版本的映像具体信息在上方也有显示。

上文是通过使用解压缩软件把 iso 安装镜像文件解压到了 D 盘的某个目录，然后对解压目录下的文件结构做了一下分析。我们在制作安装 U 盘时，使用的是 Ultraiso 刻录工具，把 iso 镜像文件刻录到了 U 盘里，这和直接解压到 U 盘里有什么区别呢？我们对比一下，

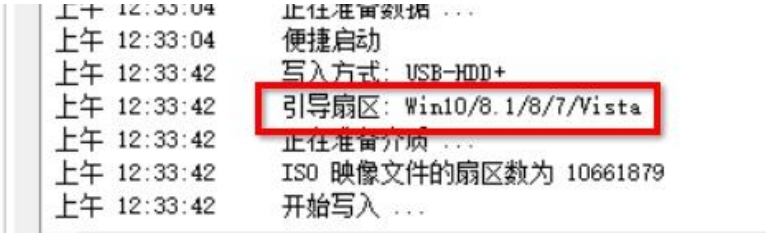
如上图，先把 win10 安装镜像文件刻录到 U 盘（U 盘文件要提前备份）里，

再打开 U 盘（本例中为 H 盘）查看一下文件组成：

根之前使用解压缩工具直接解压相比，好像没什么区别，文件数目都是一样的，大小的字节数略微不同，那也只是由于文件系统的实际差异引起的。



其实两者的根本差异只有一个，那就是“引导”，在直接解压缩时，只是把 iso 镜像里的文件复制出来了，而在刻录到 U 盘时，还把 iso 文件里的引导也刻到了 U 盘的 MBR 里。

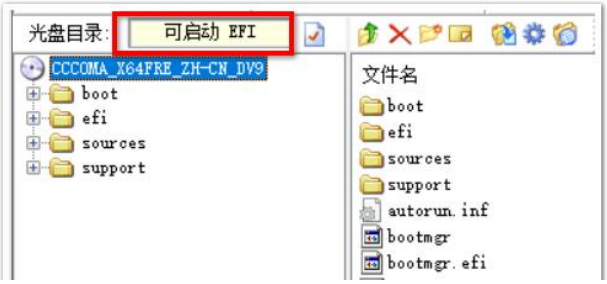


不然，Ultraiso 在刻录时，怎么知道要写入的引导扇区为 win10/8.1/8/7/vista 呢？

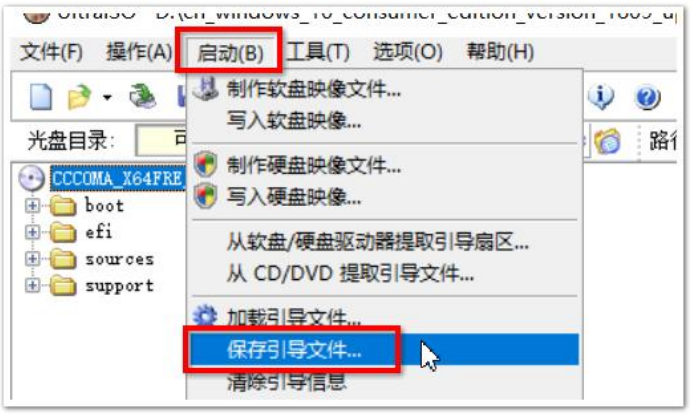
那么 iso 文件里的引导代码在哪里呢？

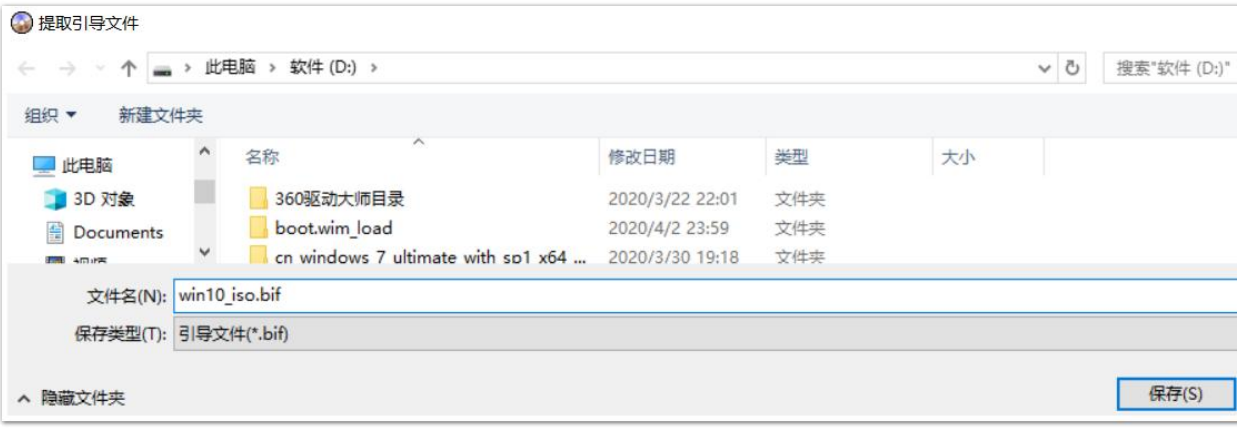
原来，iso 格式的文件是基于扇区的，可以把它当成一个虚拟磁盘文件，头几个扇区是类似于硬盘上的 MBR 扇区的。

在 Ultraiso 工具里可以看到“可启动 EFI”等字样，这个引导也是可以导出来的。



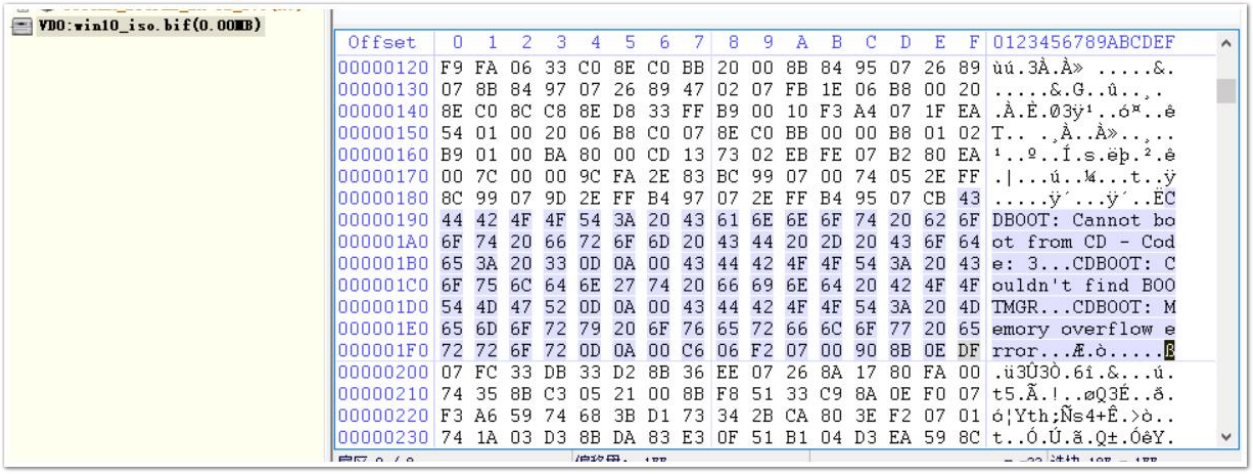
在 Ultraiso 主界面里，点击菜单栏的“启动”→“保存引导文件...”





导出并保存到 D 盘的 win10_iso.bif 文件里，该引导文件只有 4KB 大小

在使用 DiskGenius 磁盘工具查看时可见 CD boot 启动时相关的提示语。



本章小结：

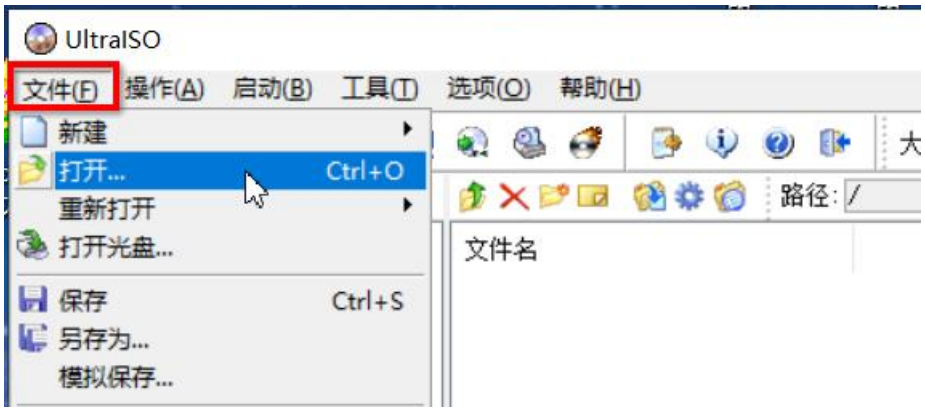
.iso 光盘镜像文件的直接解压和刻录到 U 盘是有区别的，直接解压无引导，刻录是有引导的，可启动的。

.iso 文件本来就是对安装光盘的一个打包，如果直接使用光盘启动，预启动程序会把控制交给光盘里的 CD boot 引导，然后调用 bootmgr 文件，由它进行进一步的引导，再去读取 BCD 文件，BCD 文件里只有一个默认的启动项，就是从 boot.wim 映像启动 windows PE 系统，PE 系统启动后，直接运行 setup.exe 安装向导程序，安装向导再根据用户的设置去部署 install.wim 里的操作系统到目标磁盘分区里。操作系统就安装完成。

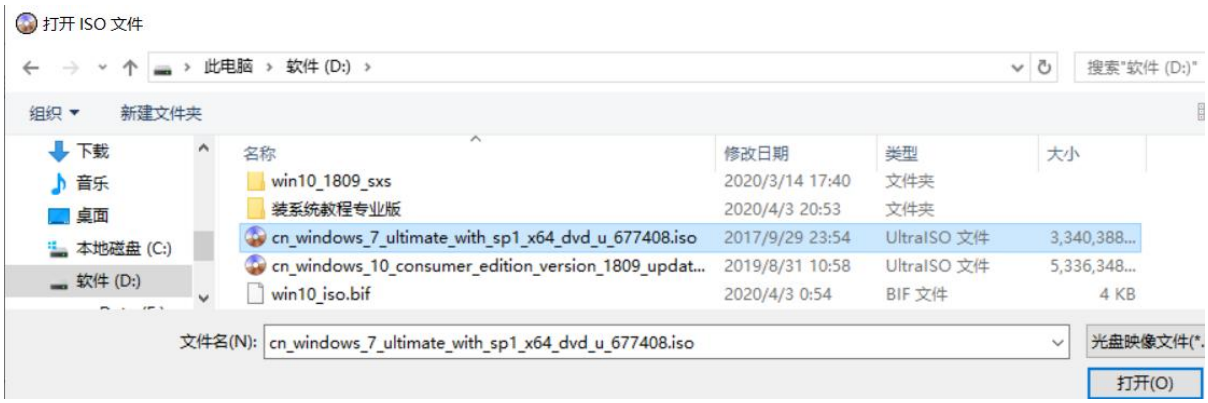
第 15 章、使用 U 盘安装 windows 7

在“msdn 我告诉你”官网下载我们需要安装的 win7 和 win10 系统，然后制作安装 U 盘，本章主要是使用刻录工具 Ultraiso 把 windows 镜像文件刻录到 U 盘里，然后把 U 盘插入目标计算机，启动计算机时，按下设置 BIOS 的快捷键，不同的计算机需按下不同的键，具体的可以到网上查询，一般在计算机启动时也会在屏幕上有提示。
因为是直接刻录 iso 文件到 U 盘里，所以同时只能刻录一个系统的，本章先讲 windows 7 下章再讲 windows 10

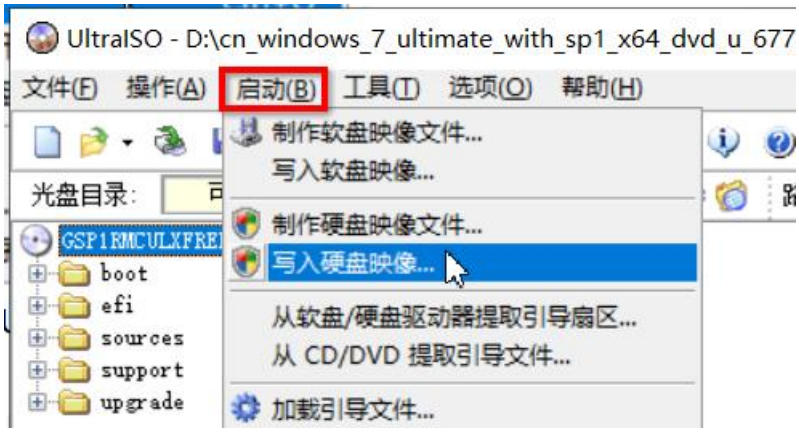
①刻录 win7 镜像文件到 U 盘



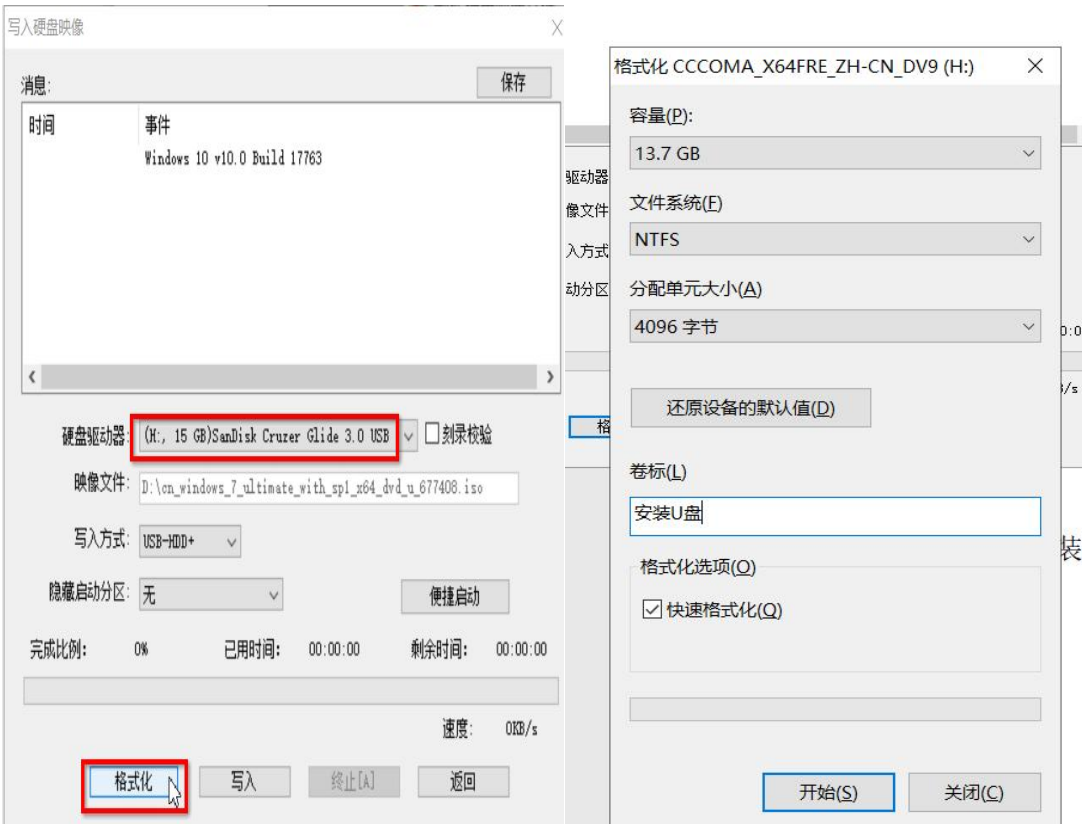
首先打开 UltraISO，点击左上角的“文件”→“打开”



打开 windows 7 的安装镜像文件，



然后，点击菜单栏的“启动”→“写入硬盘映像”



先择目标 U 盘，先格式化（要提前备份 U 盘里的文件），格式化完成后，返回刻录界面

点击“写入”

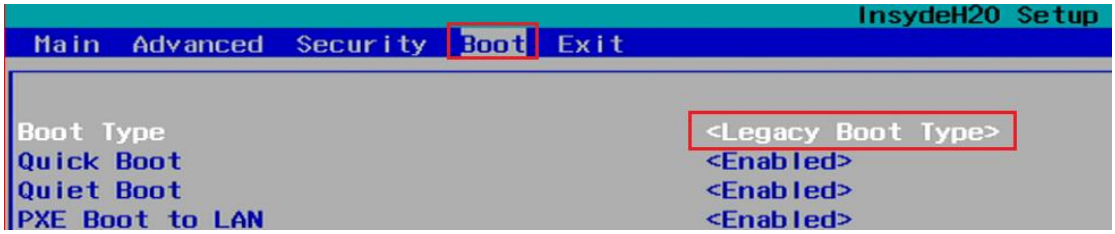
确定继续操作

刻录完成后，弹出 U 盘。

②设置目标计算机的 BIOS 启动模式

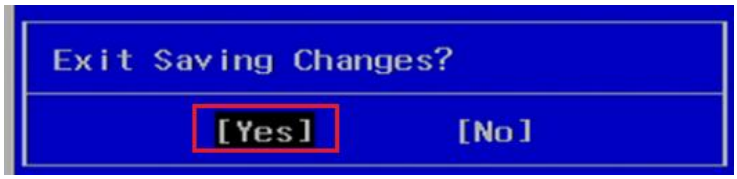
启动目标计算机时，一般会有提示按哪个键进入 boot 设置，如上图，此计算机要按下 ESC 键，

此计算机是支持 UEFI 启动的，所以 boot 界面比较丰富，不会的可以网上查询如何设置启动模式，文本致远计算机是点击“SCU”进入 boot 设置



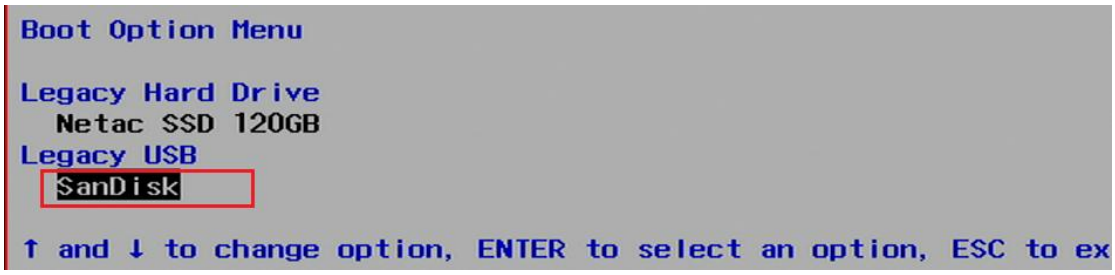
在 boot 项，选择 boot type 为 Legacy boot type 传统的 BIOS 启动方式，
windows 7 一般是用传统的 BIOS 启动的。

然后根据提示，按下 F10 键，保存设置并退出，



Yes，回车，计算机重启，再次按下 ESC 键进入 boot 设置界面，如下图

点击 Boot Manager，选择启动磁盘



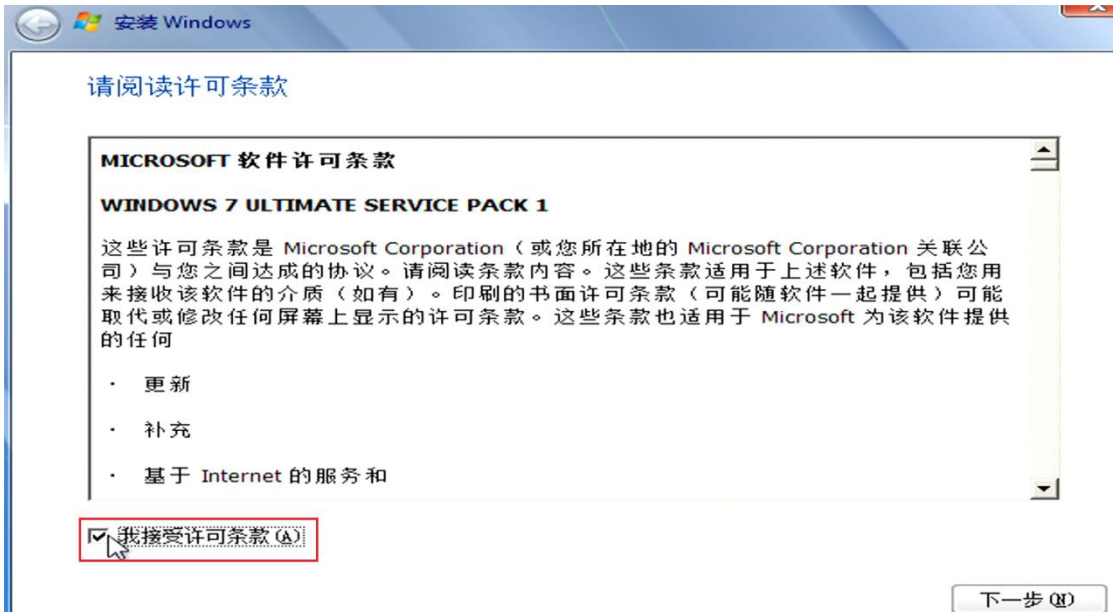
上图中，我们选择从安装 U 盘（闪迪的）启动，

出现上图这个界面表示已经从安装 U 盘加载 windows pe 系统了，系统启动后，会直接运行 Setup.exe 安装向导

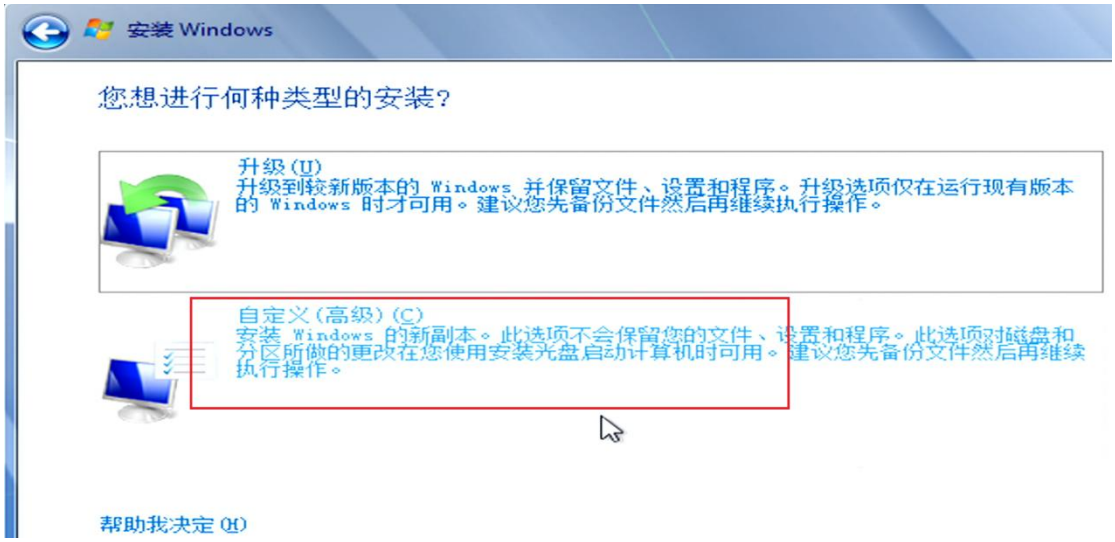
③安装 win7 系统

点击下一步，

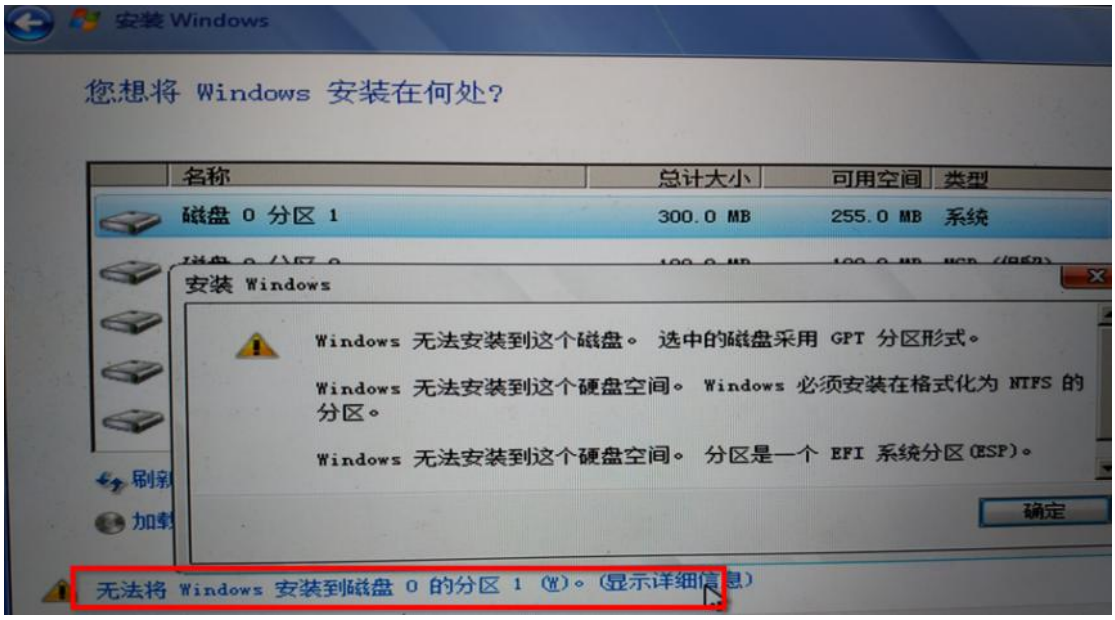
点击“现在安装”



勾选“我接受许可条款”，点击“下一步”



选择“自定义（高级）”

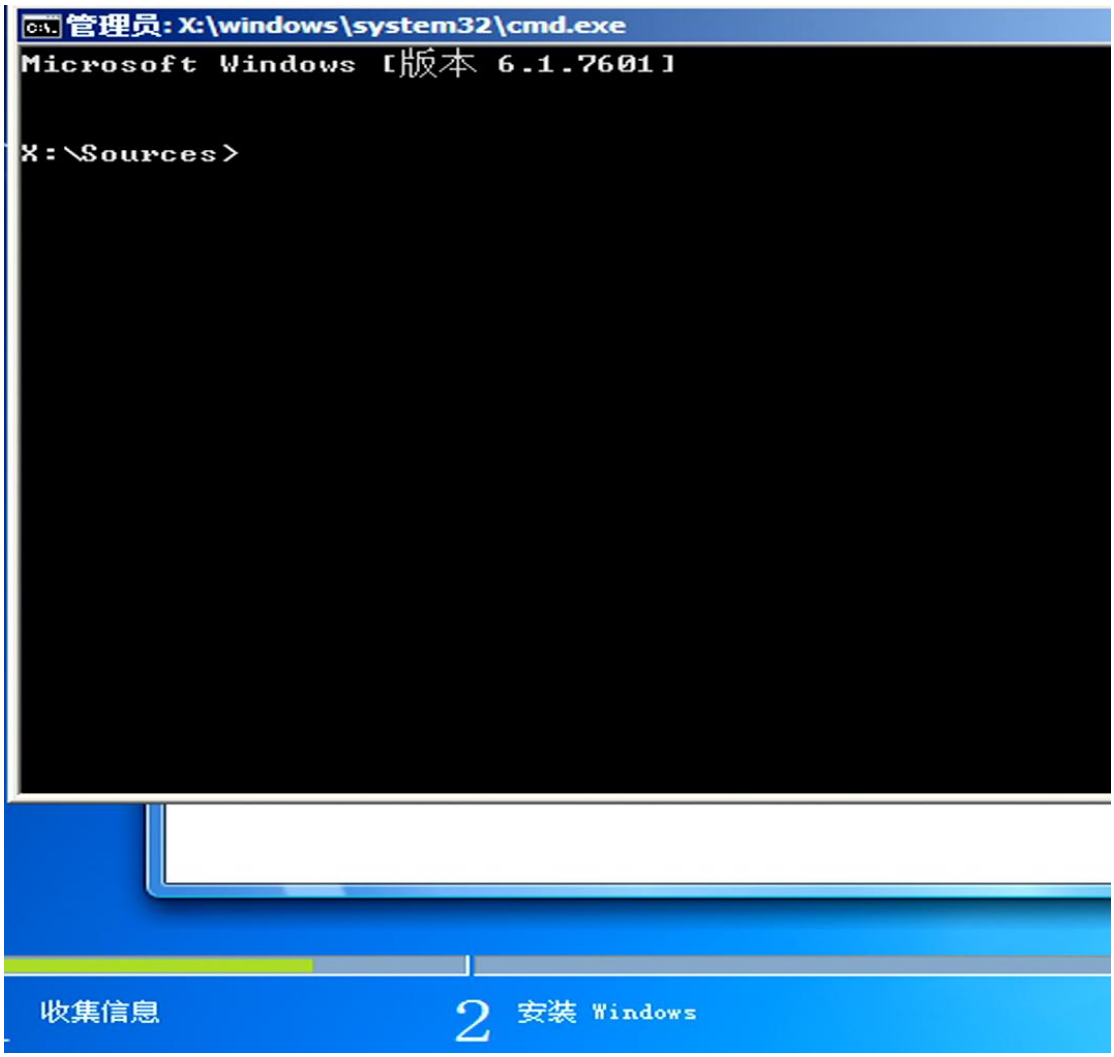


“您想将 windows 安装在何处”对话框有提示：无法将 windows 安装到磁盘 0 的分区，点击查看原因，原来选中的磁盘采用的是 GPT 的分区表，而传统的 Legacy BIOS 启动无法从 GPT 分区启动，所以要先将此磁盘（计算机里的硬盘）转为 MBR 分区表。

如果分区表是 MBR 的，但已有分区不是 ntfs，得格式化为 ntfs，可跳过转换分区表的操作

★轮换 GPT 分区表类型为 MBR 分区表

先点击“确定”关闭此提示框，再按下 Shift 键和 F10 键，弹出 CMD 命令行



在 CMD 命令行里，使用命令去操作，输入以下命令

输入 diskpart 命令，进入磁盘分区交互界面，再输入 list disk 查看计算机上的磁盘，上图显示有 2 个，**磁盘 0 为 111GB**，是计算机上的硬盘，磁盘 1 为 28GB，是 U 盘，我们要把系统安装到计算机上的硬盘，所以要操作磁盘 0,

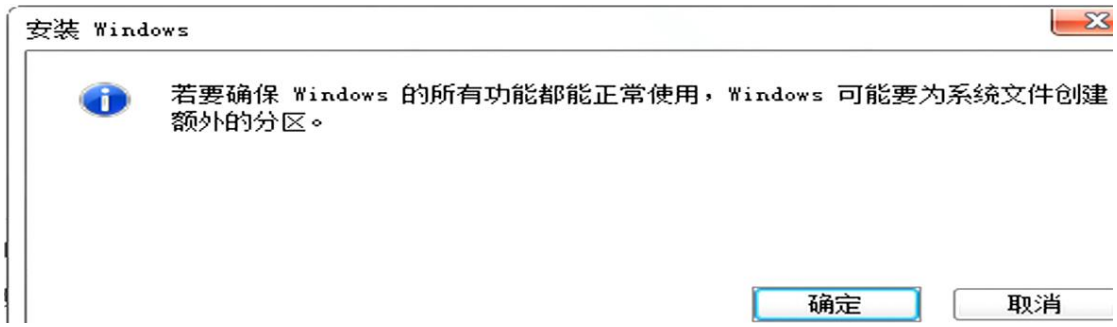
如上图操作，先 select disk 0 选择磁盘 0，
再 clean 清除磁盘上的分区，
再 convert mbr 转换磁盘分区表为 MBR 分区表，
成功就可以关闭此 CMD 命令行，

关闭 CMD 命令行后，又回到“您想将 windows 安装在何处”的对话框，点击“刷新”，

可见磁盘 0 已经清除了所有分区了，我们再点击“驱动器先项(高级)”进行分区操作，

点击“新建”，

本例中只建一个分区，所以大小为默认的 111GB，点击“应用”



会弹出一个提示“windows 要为系统文件创建额外的分区”这是什么意思？根据前面章节的知识，可知这是要创建一个系统保留分区，用来存放引导文件的，比如 bootmgr 和 BCD 等文件。

点击“确定”

上图可见，果然是创了一个“系统保留”分区，我们选择分区 2（主分区）那块，表示把系统安装在分区 2，点击“下一步”

然后就开始把系统安装到计算机磁盘上了，只管等它完成并重启，

直到出现上图界面，说明已经安装完毕，接下来只是个性化设置了，具体的就不讲了。出现需要输入密钥的地方，可以输入购买的密钥，或者先跳过，之后再激活系统。

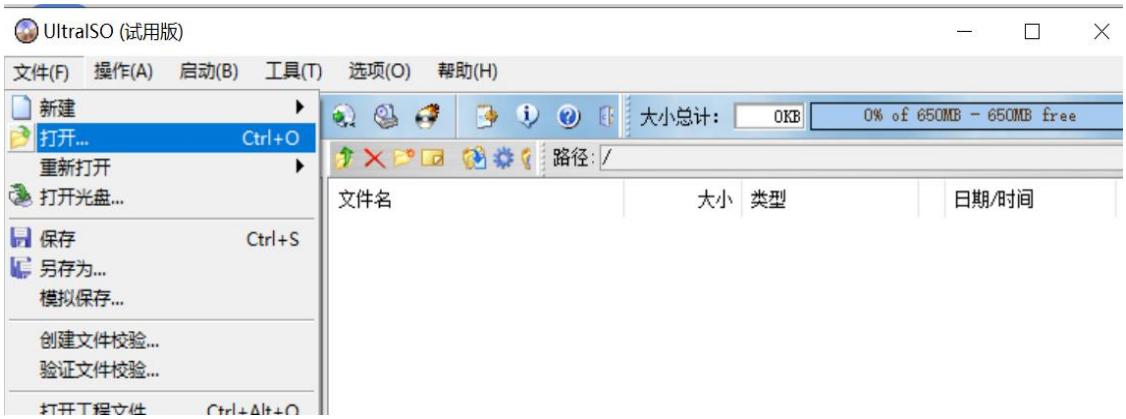
小结：

首先下载 win7 系统安装镜像文件，再使用 UltraISO 工具刻录到 U 盘里，再把 U 盘插入目标计算机，启动目标计算机，根据提示按下 ESC 或 F1~12 键进入 boot 设置界面，设置启动模式为传统的 Legacy BIOS 模式，再保存，重启后再进入 boot 设置界面，选择从 U 盘启动，然后进入 win7 安装向导，如果计算机原来的硬盘为 gpt 分区表，则需要转为 mbr 分区表，再创建新分区，将系统安装到目标分区，完毕！

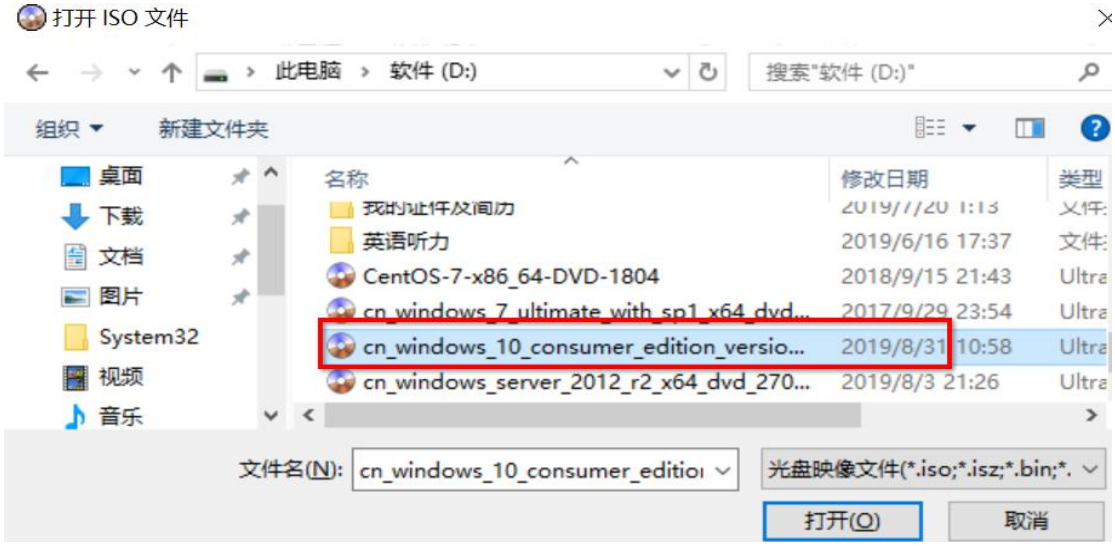
第 16 章、使用 U 盘安装 windows 10

①刻录 win10 镜像文件到 U 盘

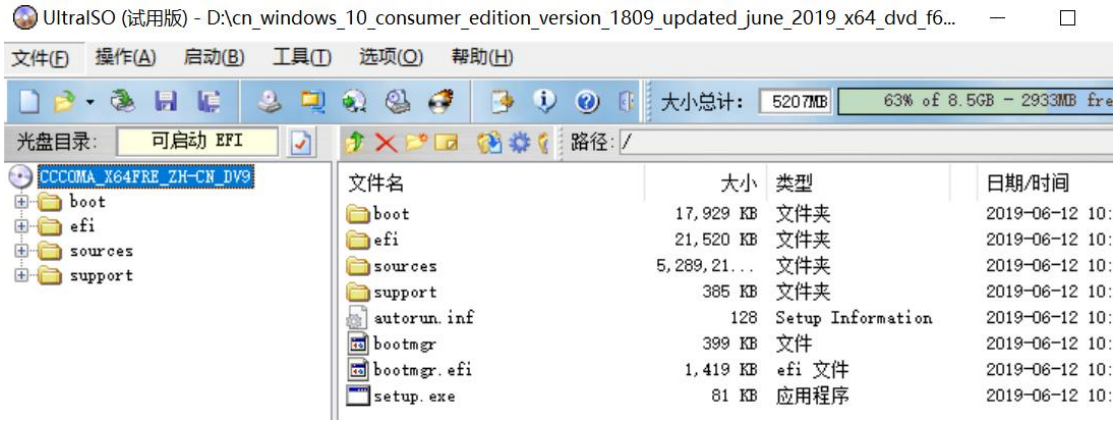
（1）打开 UltraISO 软件，点击菜单栏的“文件”、“打开”



（2）在“打开 ISO 文件”选择框里选择要使用的操作系统镜像文件，本例使用 Windows 10



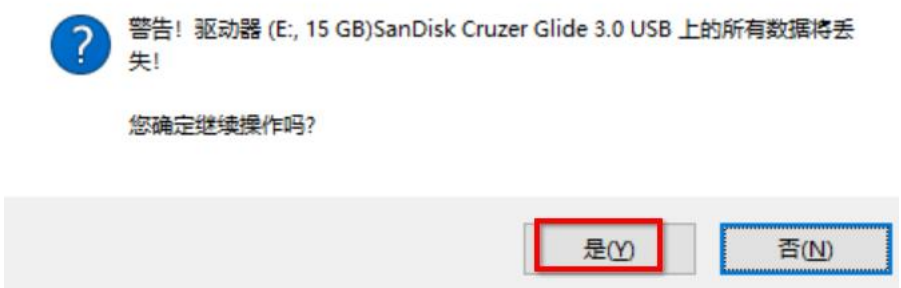
（3）上一步点击“打开”后，可以见到系统镜像里的文件



（4）点击菜单栏的“启动”、“写入硬盘映像”

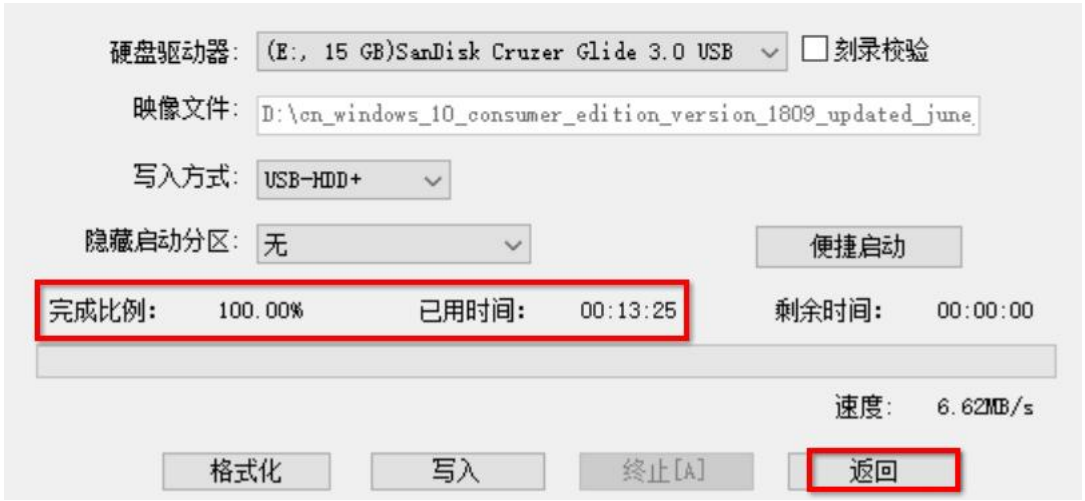
（5）然后选择要刻录的 U 盘，写入方法为 USB-HDD+，点击“写入”

（6）然后有提示，点击“是”



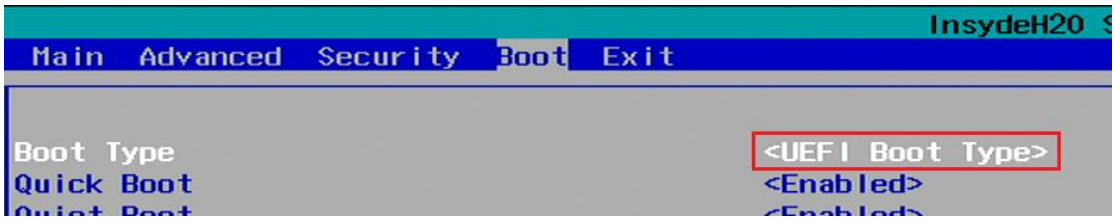
（7）接下来只要不提示错误，就可以等待它完成写入工作。

（8）要有耐心，大概 15 分钟吧，刻录完毕



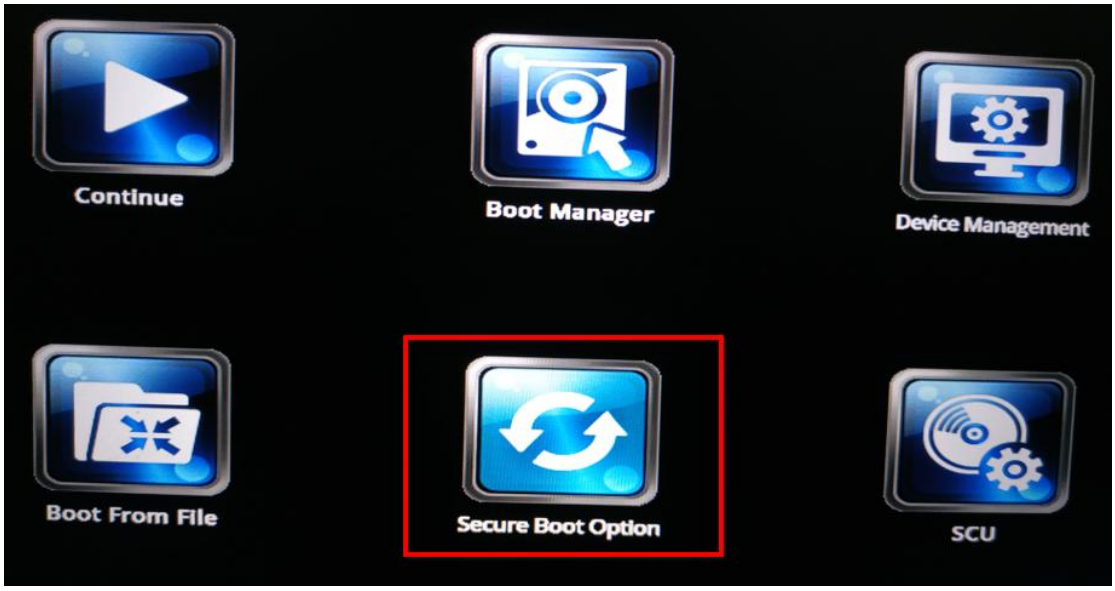
②在目标主机上用刚刚制作的启用盘安装系统

首先插入 U 盘，启动计算机，按下 ESC/DEL/F1~F12 等键进入 BOOT 界面，具体是按哪个键要根据具体的主板型号而定，可以网上查询，一般计算机启动时也会有提示

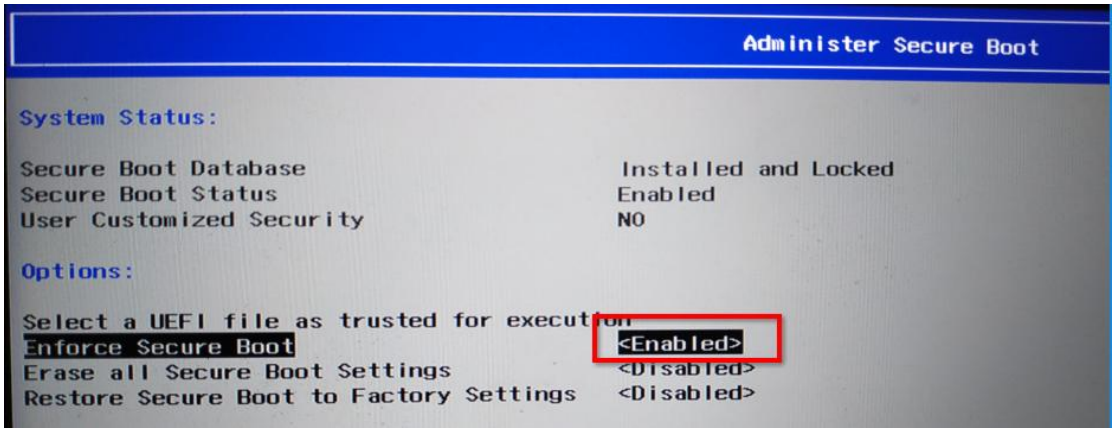


在 Boot 里设置 boot type 为 UEFI Boot Type，

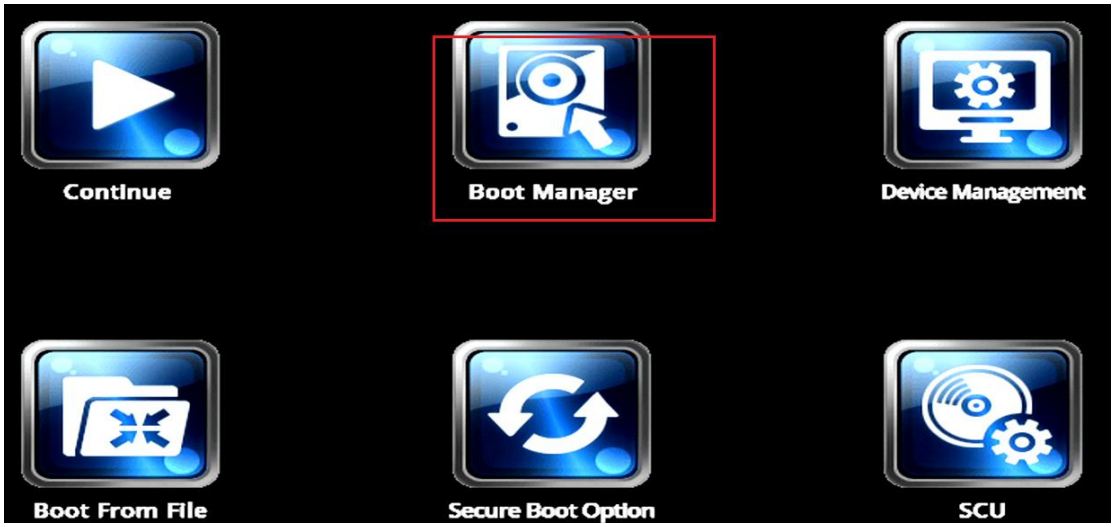
按下 F10，保存退出，计算机重启时，再次按下快捷键进入 Boot 设置界面



点击 Secure Boot Option（安全启动选项），



因为是要安装 windows 10 系统，所以一般要开启 secure boot，把 Enforce Secure Boot 项设置为 Enabled，保存退出，（较新版本可以不开启）



计算机再次启动时，按下快捷键进入 Boot 界面，点击 Boot Manager，选择要启动的磁盘，

选择从 U 盘（闪迪）启动，该 U 盘正是我们之前刻录了 win10 安装镜像的 U 盘。

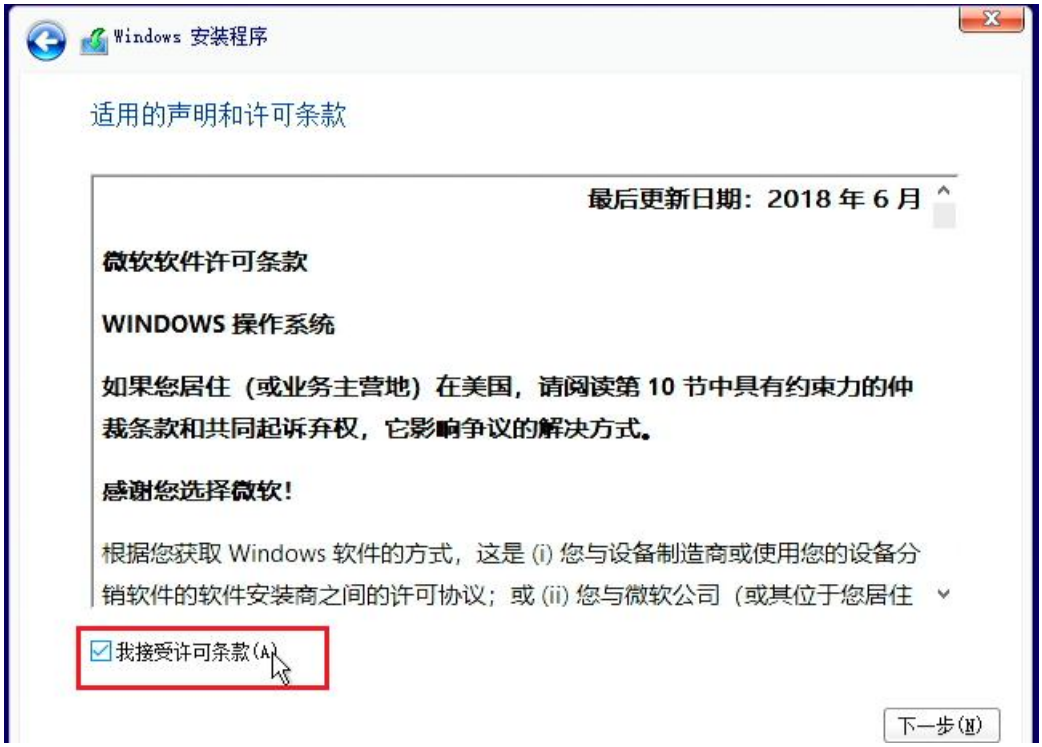
③从 U 盘启动后，就正式开始安装 win10 系统了

上图为 win10 的 Setup.exe 安装向导，点击“下一步”

点击“现在安装”

可以点击“我没有产品密钥”跳过输入密钥的步骤，而待安装系统之后再激活系统。

选择要安装的 windows 10 版本，点击“下一步”



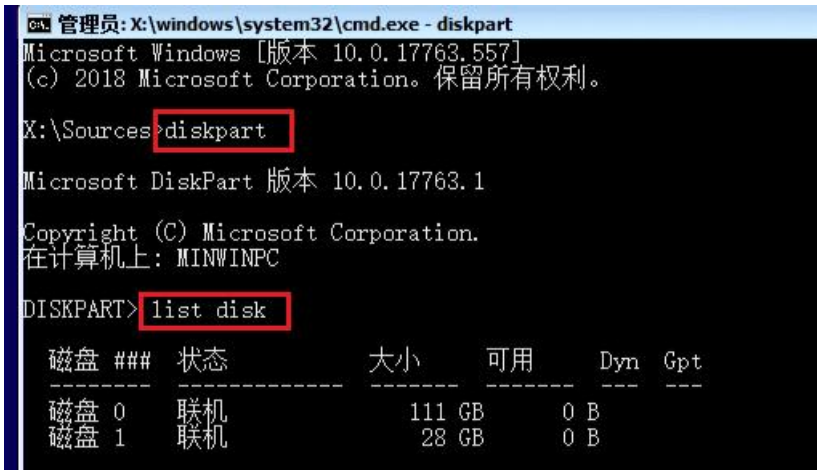
勾选“我接受许可条款”，点击“下一步”

点击“自定义：仅安装 windows(高级)”

因为目标计算机在上一章的演示中已安装过 win7，所以磁盘分区表为 MBR，UEFI 启动下的磁盘分区表应为 GPT，所以提示“无法在驱动器 0 的分区上安装 windows”，点击查看详情。点击“确定”

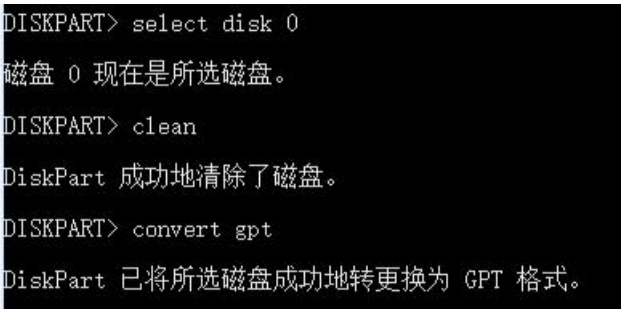
这里和使用 U 盘安装 win 7 时操作差不多，也是在安装向导中按下 Shift 键和 F10 键打开 CMD 命令行

在命令行里把计算机上的磁盘分区表转为 GPT 分区表



CMD 命令行里，输入 diskpart 进入磁盘操作交互界面，

再输入 list disk 列出计算机上的磁盘，上图可见磁盘 0 大小 111GB，磁盘 1 为 14GB，所以磁盘 1 为 U 盘，磁盘 0 为计算机上的硬盘，我们要将系统安装到计算机上，所以要把磁盘 0 的分区表转为 GPT 分区表



输入 select disk 0 选择磁盘 0

输入 clean 清除分区

输入 convert gpt 转换分区表类型为 GPT 分区表

成功后，关闭 CMD 命令行，回到安装向导界面，

刷新一下，

再新建分区，点击“新建”

因为磁盘大小有限，我们默认就创建一个分区，点击“应用”

看来 Windows 还希望我们创建额外的分区，点击确定，看看额外分区有哪些，

分区 1 为恢复分区（499MB），分区 2 为 ESP 系统分区（100MB，UEFI 启动模式下的引导文件就在此分区中）分区 3 为 MSR 保留分区，分区 4 为 111GB 的主分区（操作系统就安装在此）
其中分区 1 恢复分区可以不要，分区 3MSR 保留分区也可以不要，必要的有 ESP 分区和一个主分区即可。

在“您想将 windows 安装在哪里”对话框中，选择分区 4 主分区，然后点击“下一步”

然后就开始安装系统到计算机磁盘上了。中间可能会多次重启系统，

直到看到上图就说明系统已安装成功了，接下来就是自己的操作了。

小结：

首先下载 win10 系统安装镜像文件，再使用 UltraISO 工具刻录到 U 盘里，再把 U 盘插入目标计算机，启动目标计算机，根据提示按下 ESC 或 F1~12 键进入 boot 设置界面，设置启动模式为 UEFI 模式，再保存退出。重启后再进入 boot 设置界面，设置 Secure Boot 为开启(Enabled)，再保存退出。再次重启后进入 boot 设置界面，选择从 U 盘启动，然后进入 win10 安装向导，如果计算机原来的硬盘为 mbr 分区表，则需要转为 gpt 分区表，再创建新分区，将系统安装到目标分区，完毕！

第 17 章、Windows 7 无法安装在新计算机上？

有时候我们在安装 win 7 时，会出现安装失败的情况，主要症状是鼠标和键盘失灵了，无法继续安装，或者在安装时直接提示找不到安装介质。
这是什么原因引起的呢？是新的 CPU 不支持 win7 系统了？

不是的，CPU 怎么知道我要安装的是 win7？经过一番查找和试验，原来是 windows 7 安装镜像文件里，不带有新的 USB3.0 的驱动，导致安装时，无法识别 USB 设备，自然也就找不到带有安装文件的 U 盘介质和 USB 鼠标、键盘了。

新的主板（一般指 2015 年以后的）使用的主机控制器接口为 XHCI（可扩展的主机控制器接口，由 Intel 开发的 usb 主机控制器，主要面向 usb3.0 标准。而 windows 7 官方安装包（2009 年发布的）里不带有 XHCI 的驱动，所以会不识别一切 USB 设备。

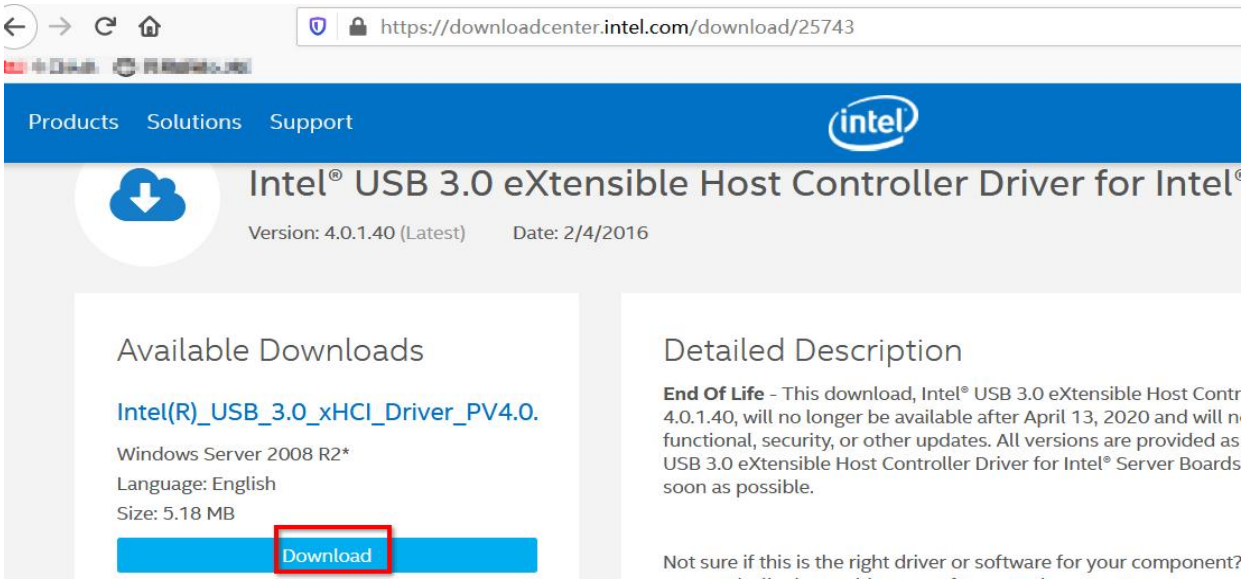
怎么在新的计算机上安装 win 7 呢？安装之后再安装 XHCI 驱动？问题是安装都无法完成，怎么进入系统呢？解决方法只有一个，就是在安装之前，在安装镜像文件里添加 XHCI 驱动，再进行安装。可以用 Dism++工具软件对 win 7 的 ISO 镜像文件进行修改，添加 XHCI 的 usb3.0 驱动。

具体操作如下：

首先，把 win7 镜像文件解压到 cn_windows7_xxx...目录下，
使用 U 盘安装时，首先引导起来的是 boot.wim 里的 PE 系统，然后 PE 里再通过 setup.exe 安装向导去部署正式的系统到计算机磁盘里。所以我们要在 boot.wim 里的映像添加 usb3.0 驱动，还要在 install.wim 里的正式映像里也添加 usb3.0 驱动。
在 boot.wim 里添加 usb3.0 驱动是为了能正常运行 win PE 系统，
在 install.wim 里添加 usb3.0 驱动是为了在安装好系统后，能正常使用系统

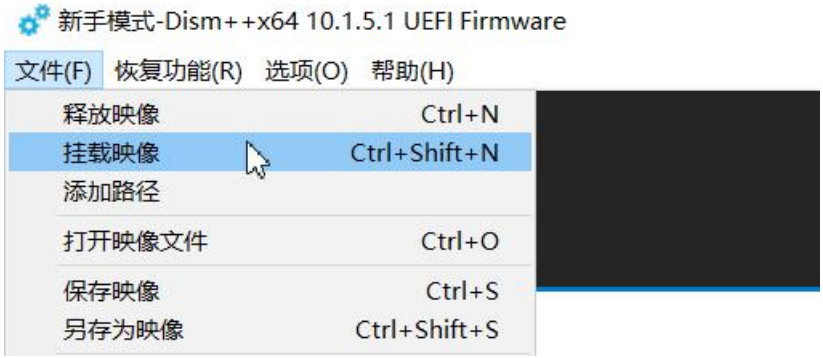
在 D 盘创建 2 个文件夹，名称分别为 boot.wim_load 和 install.wim_load，用于挂载临时的 wim 映像解压文件

然后从网上下载 usb3.0 驱动
https://downloadcenter.intel.com/download/25743

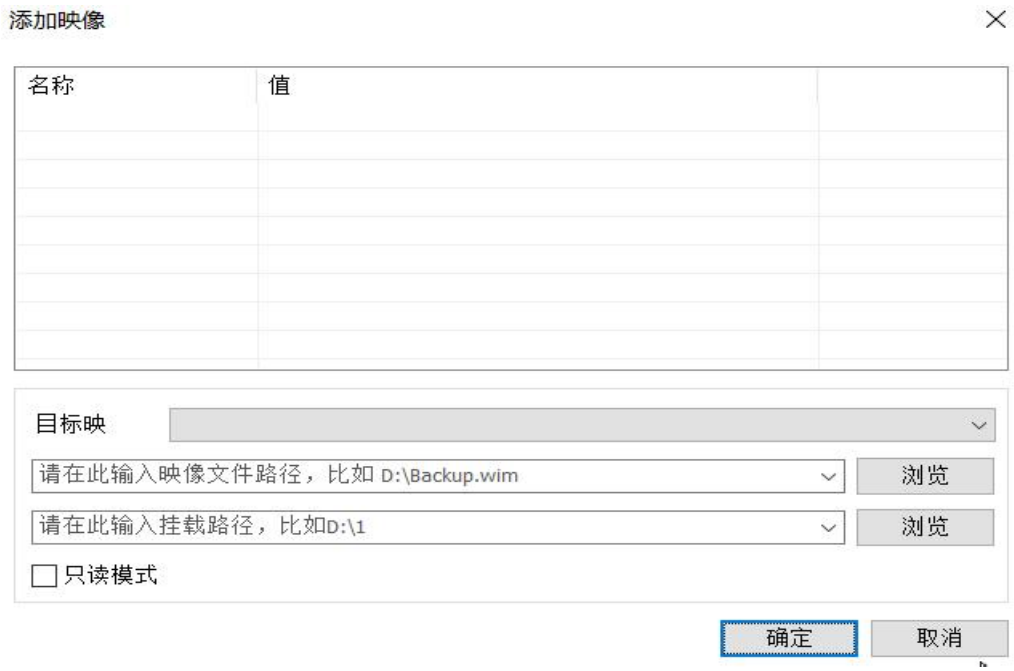


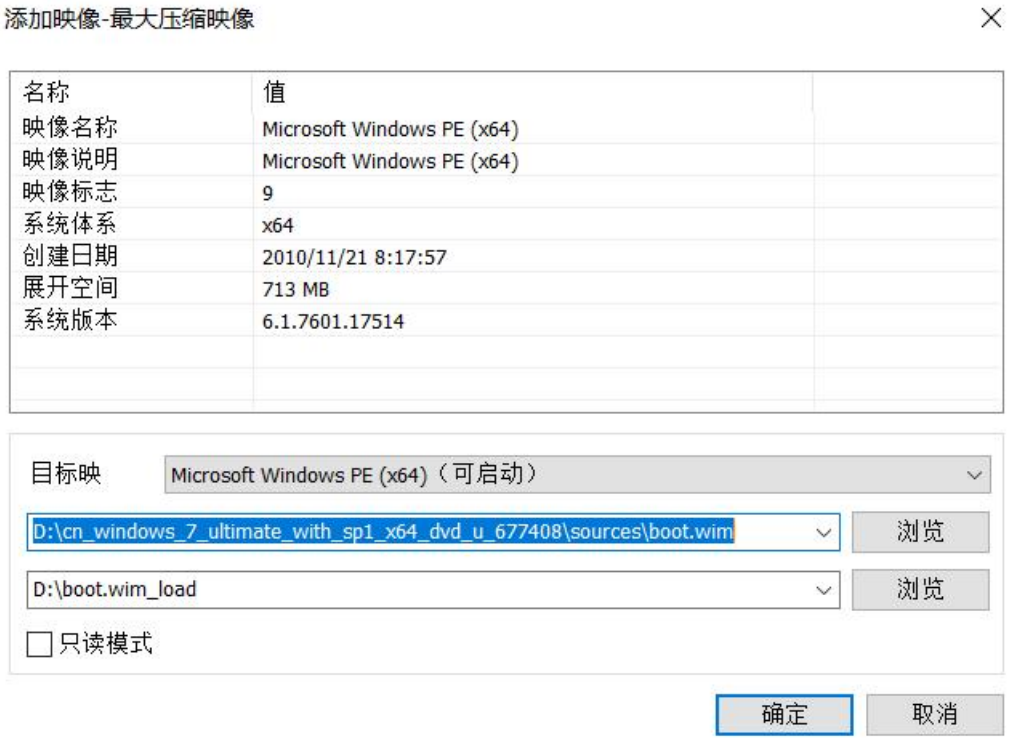
下载到本地后，解压到当前文件夹，然后会出现一个解压后的文件夹 Intel(R)_USB_3.0_xHCI_driver_PV4.0.1.40

打开 Dism++软件，

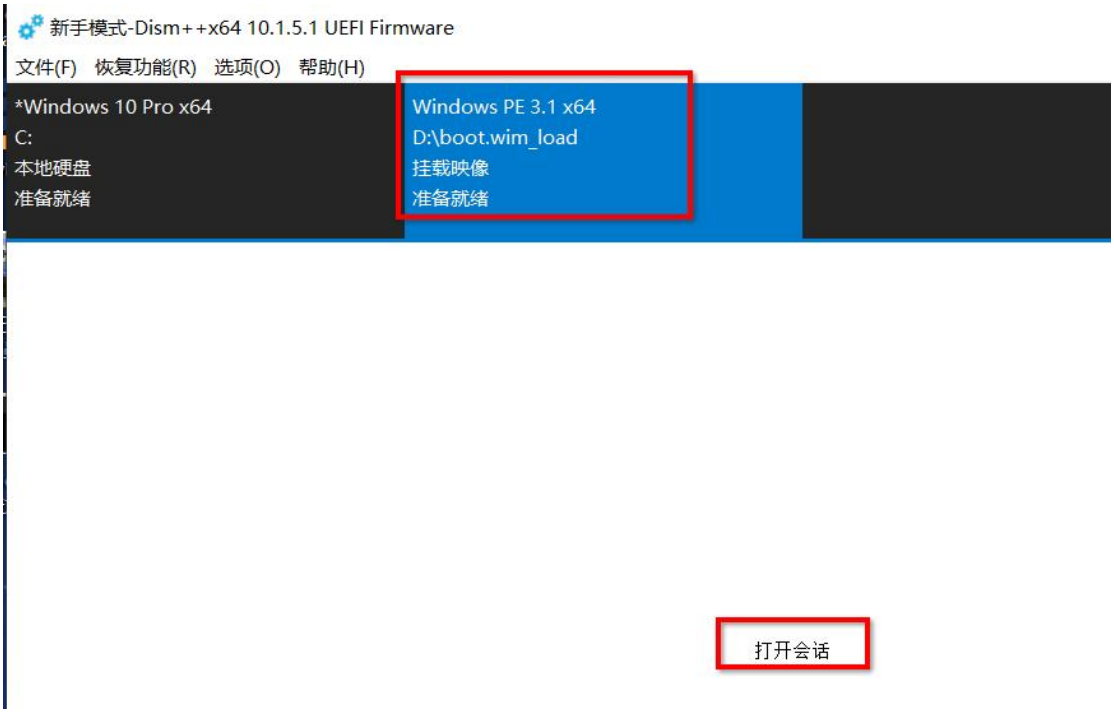


点击主界面左上角的“文件”→“挂载映像”



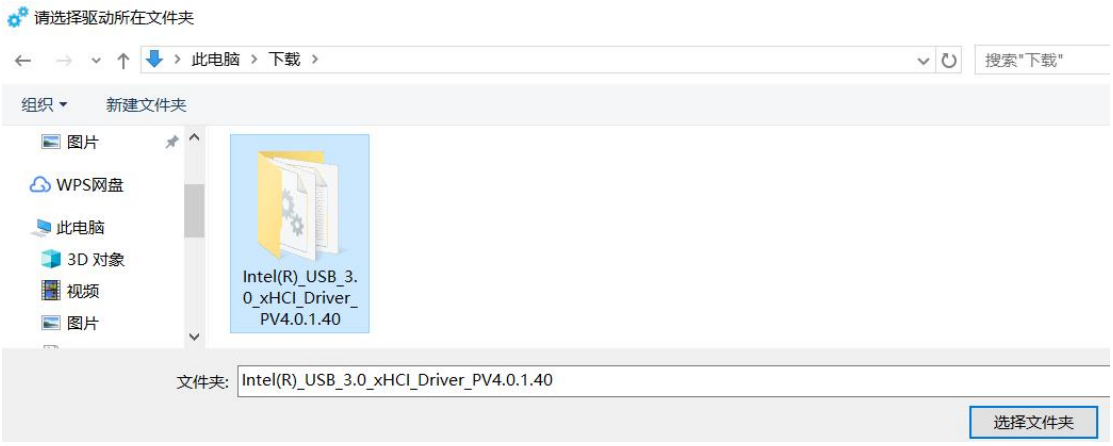


映像文件路径为 win7 解压目录里的 sources\boot.wim
挂载路径为 D:\boot.wim_load
目标映像为 Microsoft Windows PE(x64)（可启动）
点击“确定”

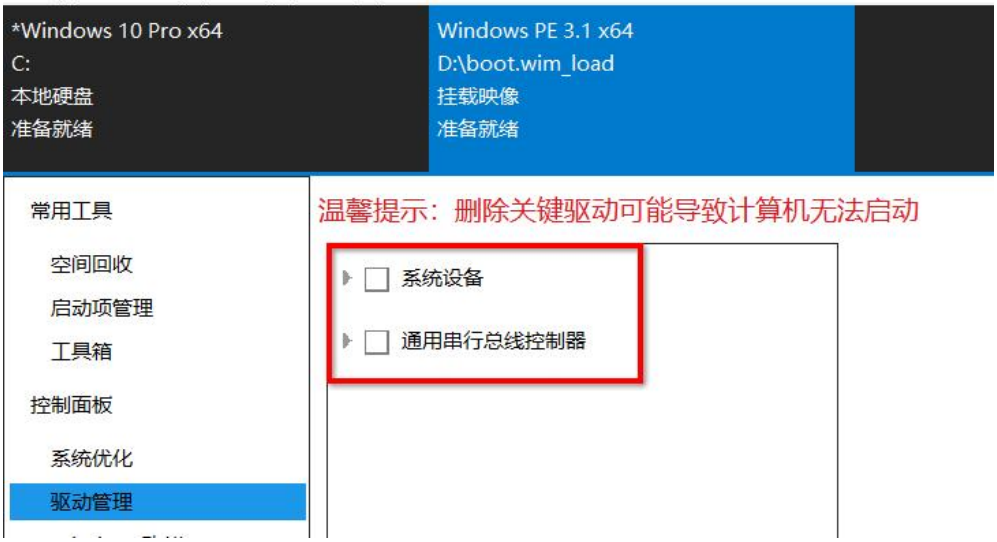


挂载完毕，点击中间空白处的“打开会话”

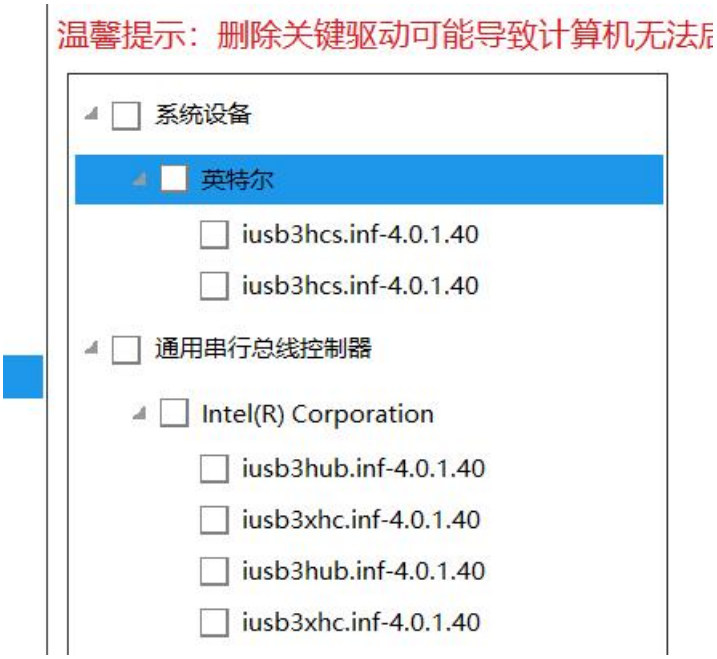
在左边控制面板下点击“驱动管理”，再在右下角点击“添加驱动”



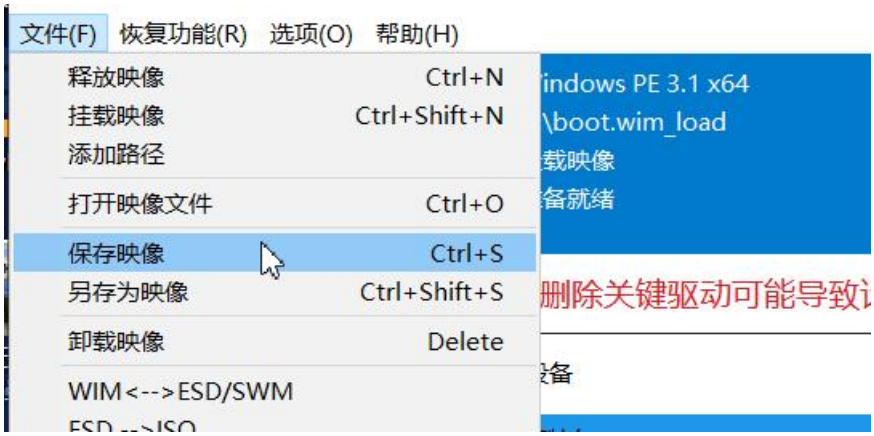
选择刚刚驱动解压后的文件夹，点击“选择文件夹”确认，
然后就开始安装驱动了，



驱动安装完成，在空白处多了两个驱动目录，可以展开看看



确实是 usb3.0 的驱动。

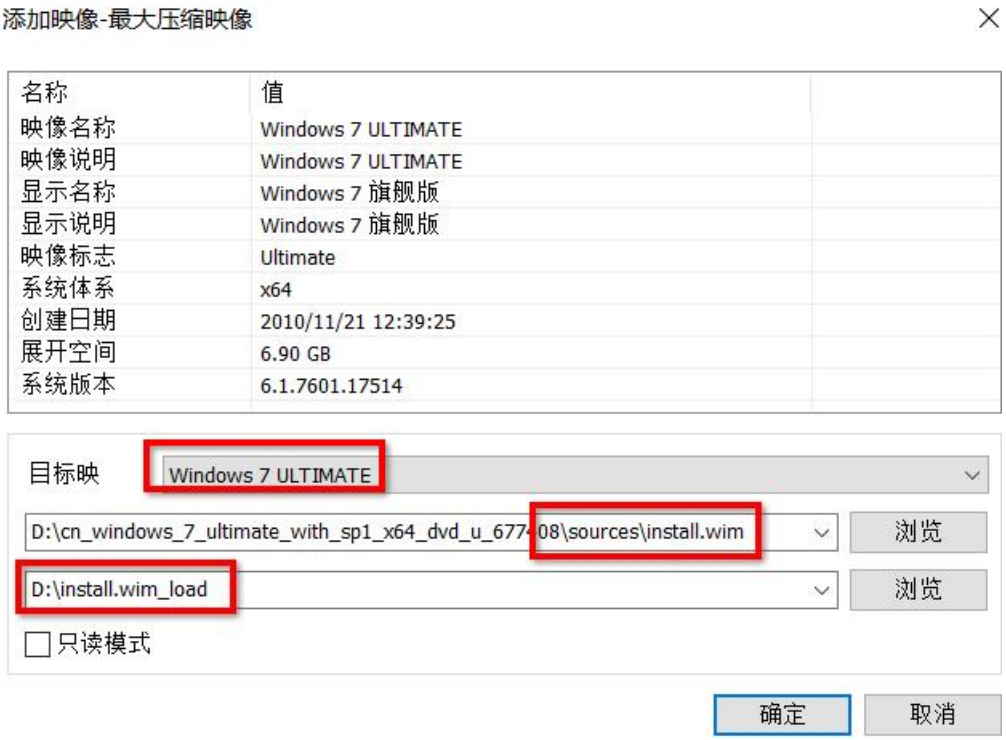


添加驱动后，再点击菜单栏的“文件”→“保存映像”

选择“直接保存”

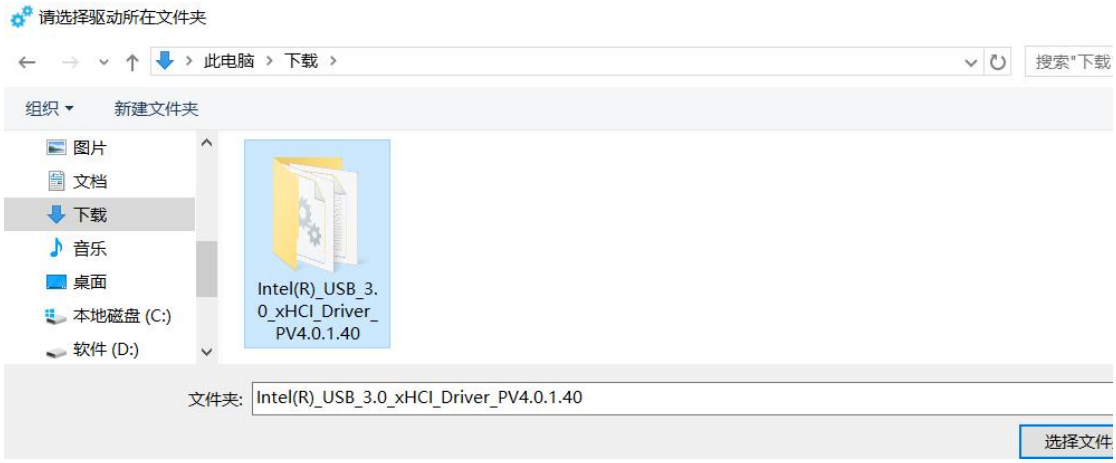
保存后，再卸载映像就行了，boot.wim 里的映像就有了 usb3.0 的驱动，可以正常安装 win7 系统了。

接下来，再挂载 install.wim 里的系统，假如我们要安装的是 win7 旗舰版，那么就挂载该版本的映像，

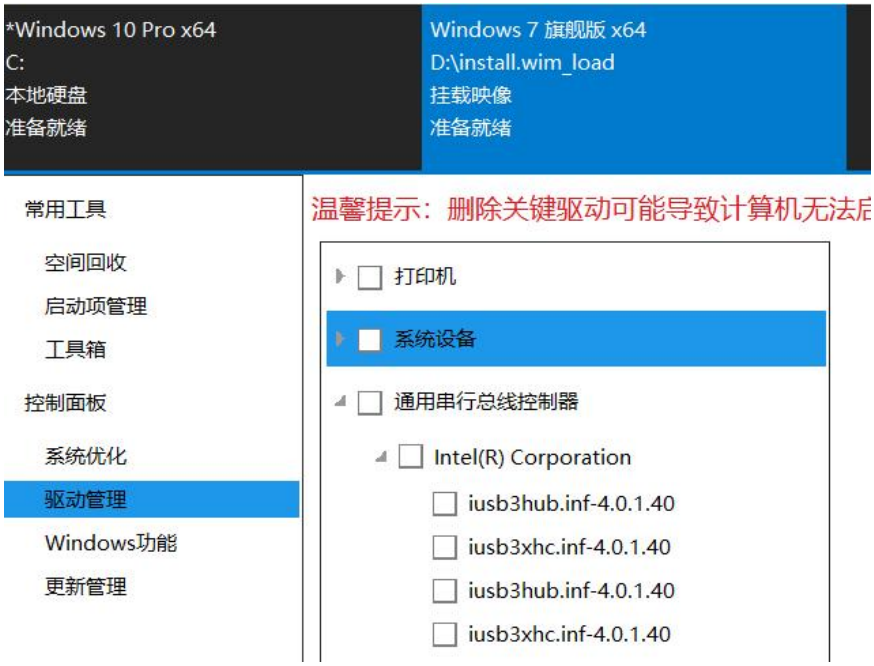


点击“确定”，等映像挂载完成，同样是点击“打开会话”

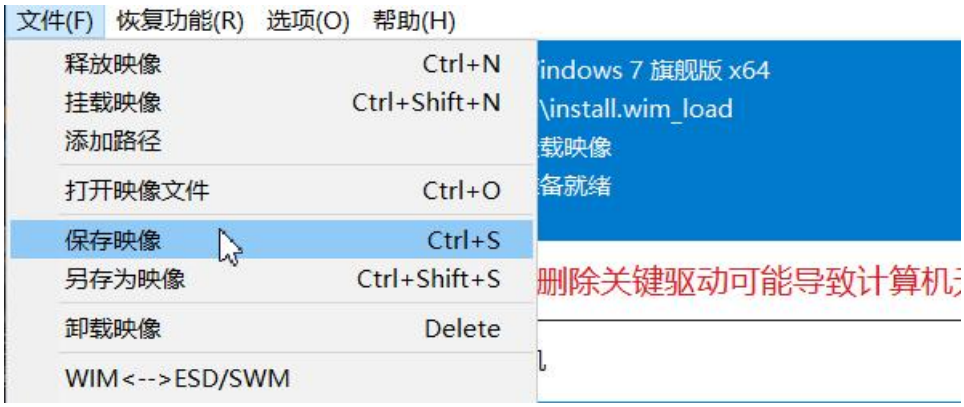
在“驱动管理”右侧，点击“添加驱动”



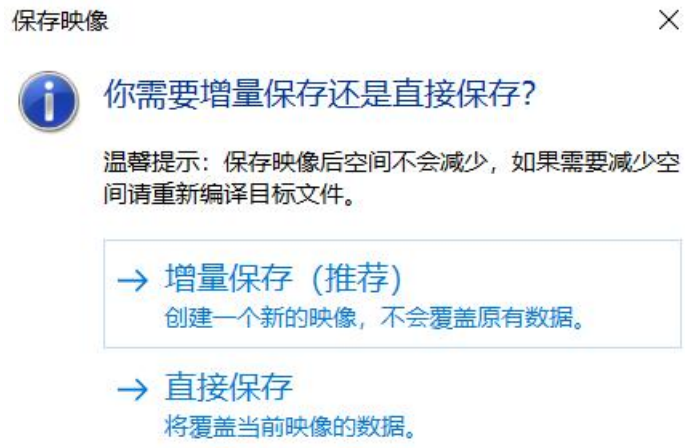
选择 usb3.0 解压文件夹，确定，



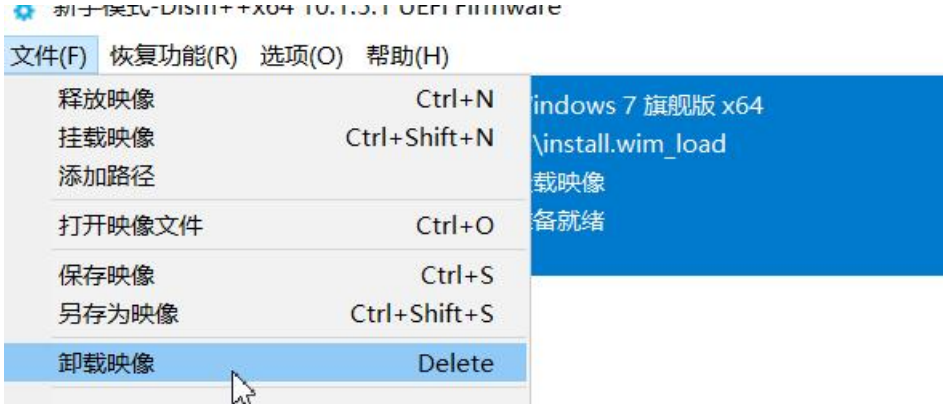
添加驱动成功，



再“保存映像”



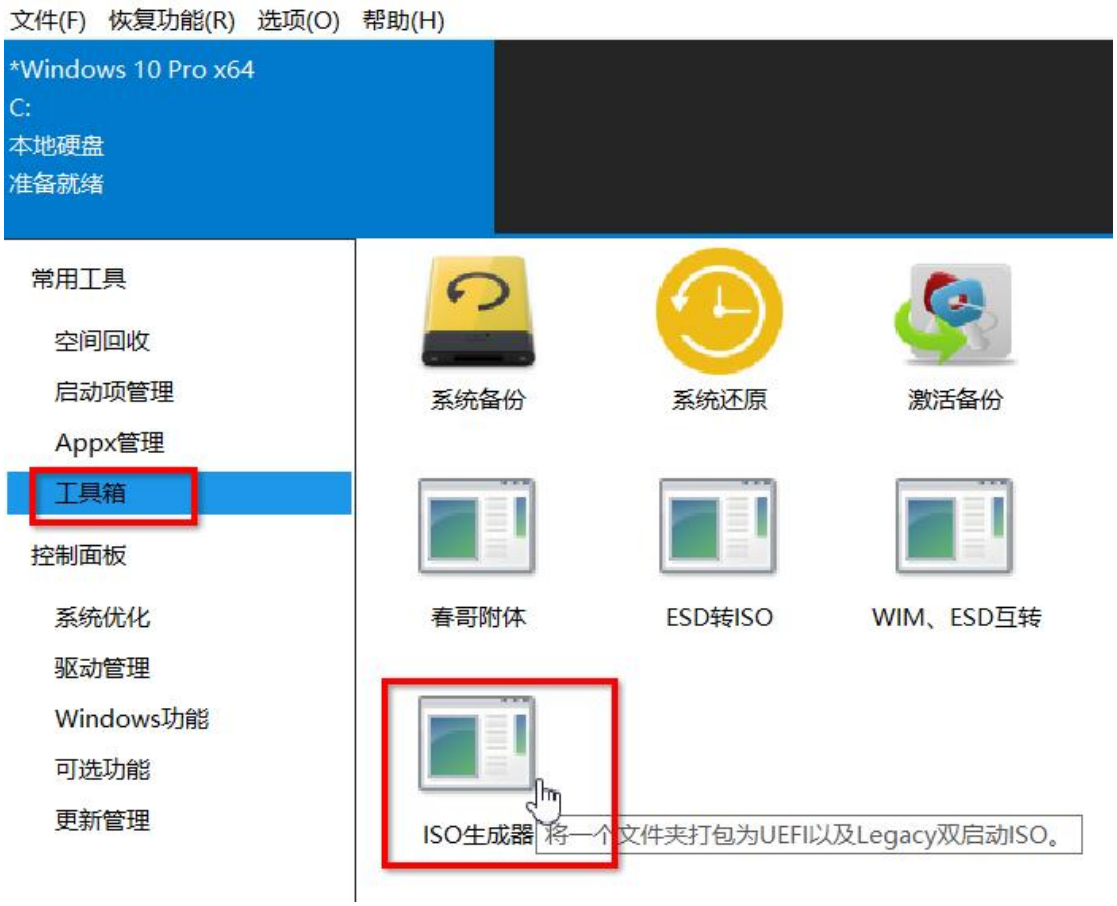
“直接保存”



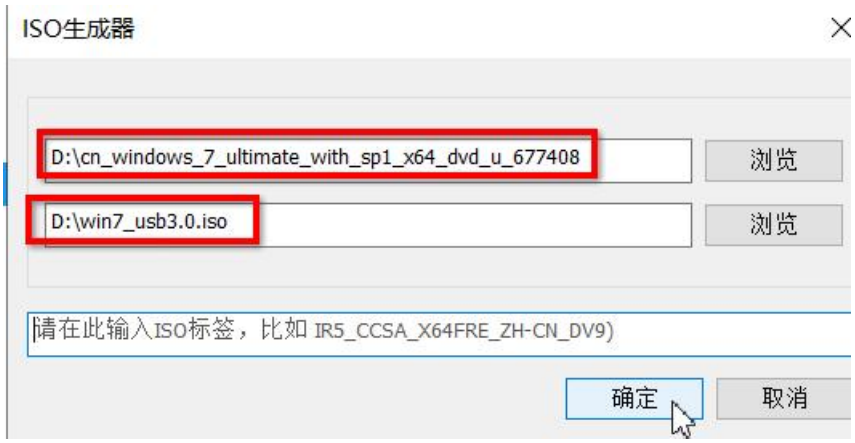
最后“卸载映像”就行了，

以上只是在 win7 旗舰版里添加了 usb3.0 的驱动，如果其他版本也要添加，可以按上文的操作，一一挂载、添加驱动、保存映像、卸载映像。

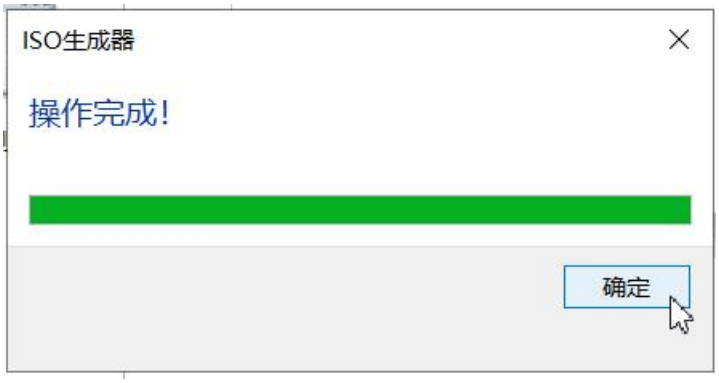
驱动添加完成后，再把原来 win7 的镜像解压目录，打包成一个新的 iso 格式的镜像文件，可以使用 dism++软件，



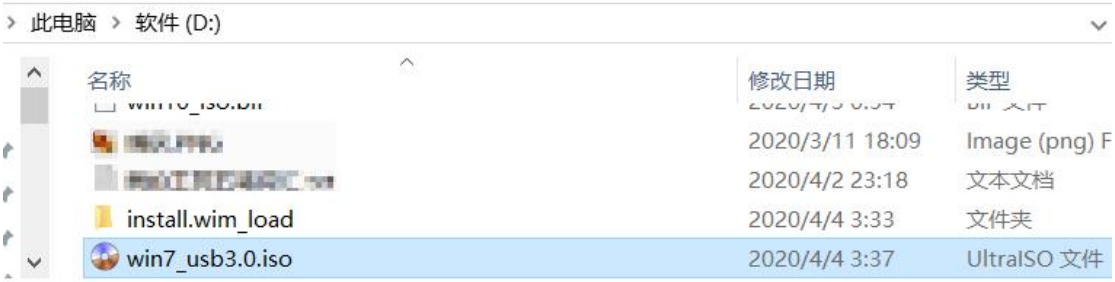
点击“工具箱”→“ISO 生成器”



选择原 win7 解压目录，生成的 iso 文件名为 win7_usb3.0.iso
点击“确定”



完成后，在 D 盘就生成了新的 iso 文件，是我们刚刚添加了 usb3.0 驱动的 win7 安装镜像文件，可以把此文件刻录到 U 盘，然后在新计算机上安装 win7 系统



小结：
win7 无法在新计算机上安装，只是缺少相应的驱动，下载相关的驱动文件后，再使用 Dism++添加到 boot.wim 和 install.wim 里的映像，再保存即可。
不止是添加 usb3.0 驱动，还可以添加其他的比如显卡驱动，声卡、网卡驱动以及 nvme 硬盘驱动等。

第 18 章、PE 系统的使用

PE 系统（Preinstallation Environment）预安装系统，windows PE 系统就是极简的 windows 系统，功能有限，一般不能联网，只是用来协助安装正式的 windows 系统，它是给 setup.exe 安装向导提供一个可以正常运行的环境。或者说，在 PE 系统里，setup.exe 才可以运行，因为 setup.exe 安装向导只是一个程序，要在 windows 环境下运行的。

目前我们常用的 PE 系统不是微软官方原版的，因为官方的 PE 系统只有 cmd 命令行，不能进入桌面，于是为了方便使用和添加更多的系统安装或维护工具，网上的各大高手给官方的 PE 添加了桌面，且进一步精简了系统文件，于是各种各样的个性化的 PE 系统横空出世。

使用 PE 系统，我们可以更方便灵活地安装各版本的操作系统，也可以在 PE 环境下对无法正常进入系统的计算机进行资料的备份和引导的修复以及修改管理员密码，制作双系统或多系统等。

怎么制作 PE 启动 U 盘呢？我们可以在网上下载合适的 PE 系统 iso 文件，然后用 Ultraiso 刻录到 U 盘，也可以下载制作 PE 启动盘的工具软件。网上各种宣称纯净的 PE 系统 iso 文件，因为并不是某些正规企业发布的，所以也没法确保它的安全，万一有病毒呢。所以我们唯一可信的是一些勉强信得过的正规一点儿的 PE 制作工具，比如 微 PE，老毛桃，大白菜等。（当然如果有信心，我们也可以自己基于微软的 PE 再添加桌面和其他工具软件，做成自己的 PE 系统也是可以的，不过这个难度有点儿大，本教程不教授）

本章讲一下 wepe（微 PE）的使用。

①PE 优盘的制作

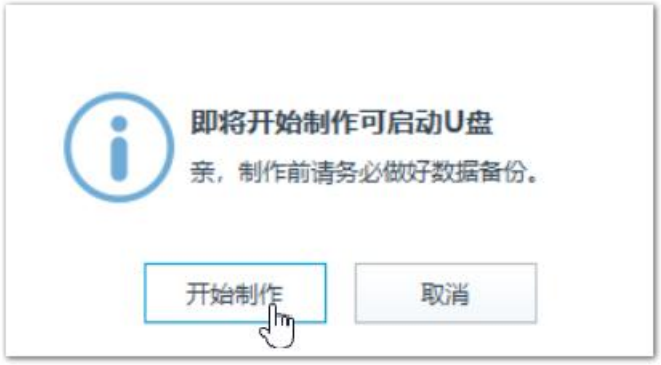
首先到微 PE 官网下载 PE 制作软件，http://www.wepe.com.cn/download.html

下载后，双击安装，运行 wepe，主界面如下：

然后，插入要制作成 PE 启动盘的 U 盘，

点击右下角的 U 盘图标，安装 PE 到 U 盘，

确定待写入 U 盘为我们刚刚插入的 U 盘，千万别选错了，该 U 盘里的资料也要提前备份。然后点击下面的“立即安装进 U 盘”

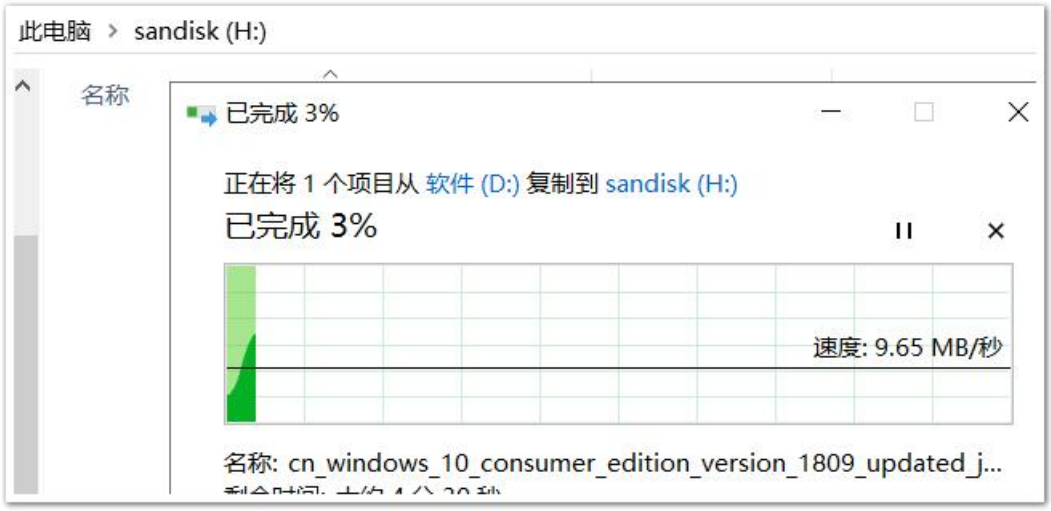


点击“开始制作”等待制作完成

点击“完成安装”然后弹出 U 盘，该 U 盘就可以使用了。

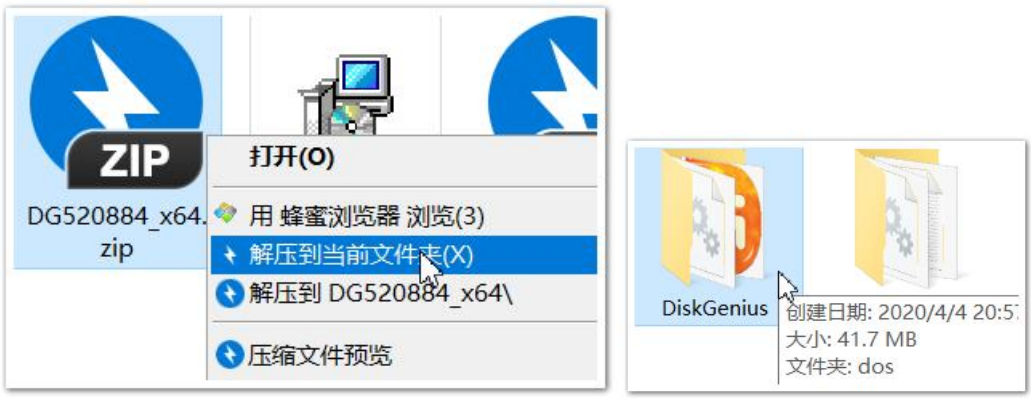
②PE 优盘的使用（使用 PE 盘进行系统的安装）

首先，把下载好的 windows 系统镜像文件复制到刚刚制作了 PE 系统的 U 盘里



然后，还得下个磁盘工具软件，DiskGenius 磁盘精灵，官网地址（要下载 5.x 的版本 ）

http://www.diskgenius.cn/download.php

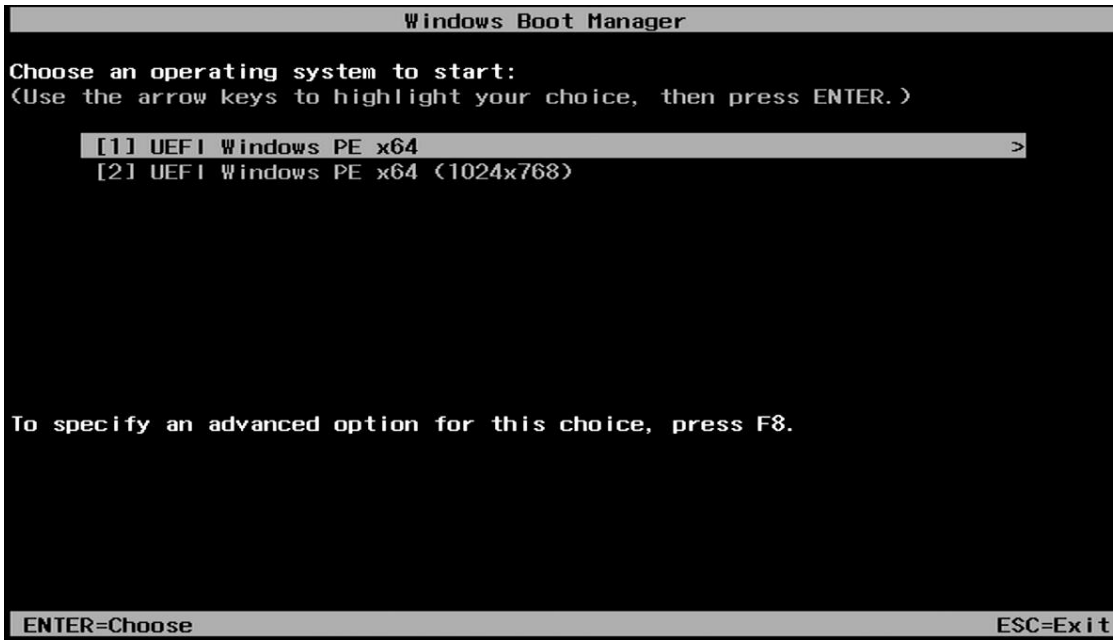


下载之后，是个 zip 压缩文件，要解压到当前文件夹，然后多了一个解压后的文件夹名为 DiskGenius，为了方便区别版本，建议改名为 DiskGenius5.2

然后把这个 DiskGenius5.2 文件夹复制到 PE 优盘里，

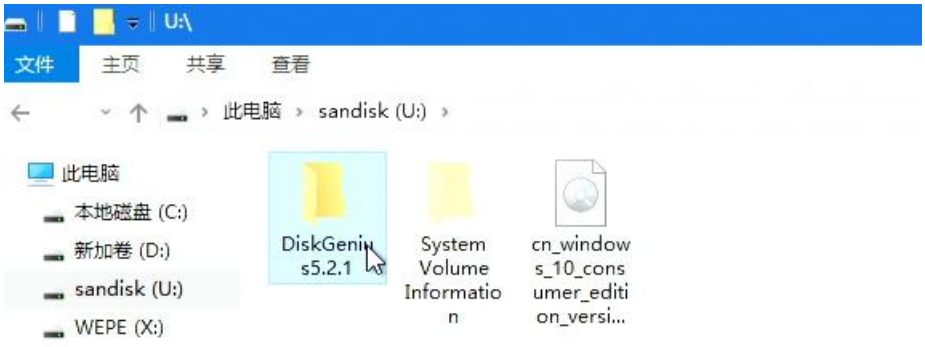
这样就可以拔出 U 盘了，准备插到目标计算机上，使用 PE 系统进行安装系统

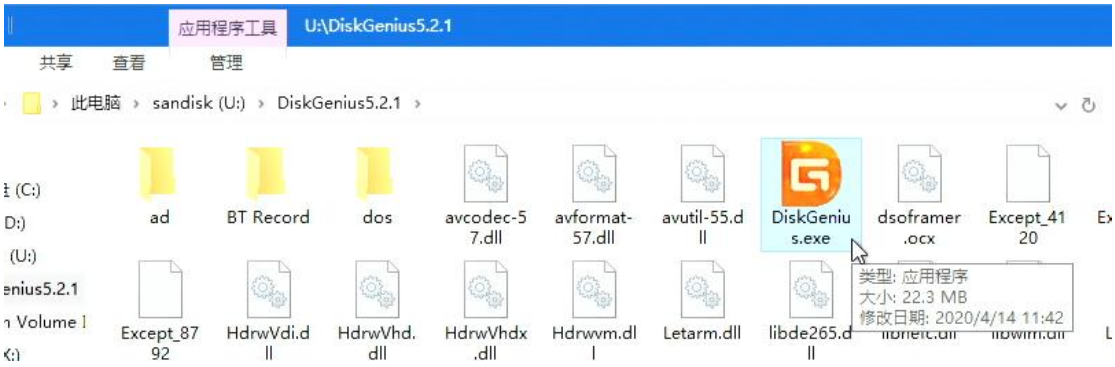
因为是要安装 windows 10，所以建议用 UEFI 启动模式，再进入 boot 设置，选择从 U 盘启动（EFI USB Device(闪迪)）



UEFI 模式下 从 U 盘启动后，出现上图这个界面，一闪而过，时间短，不管它

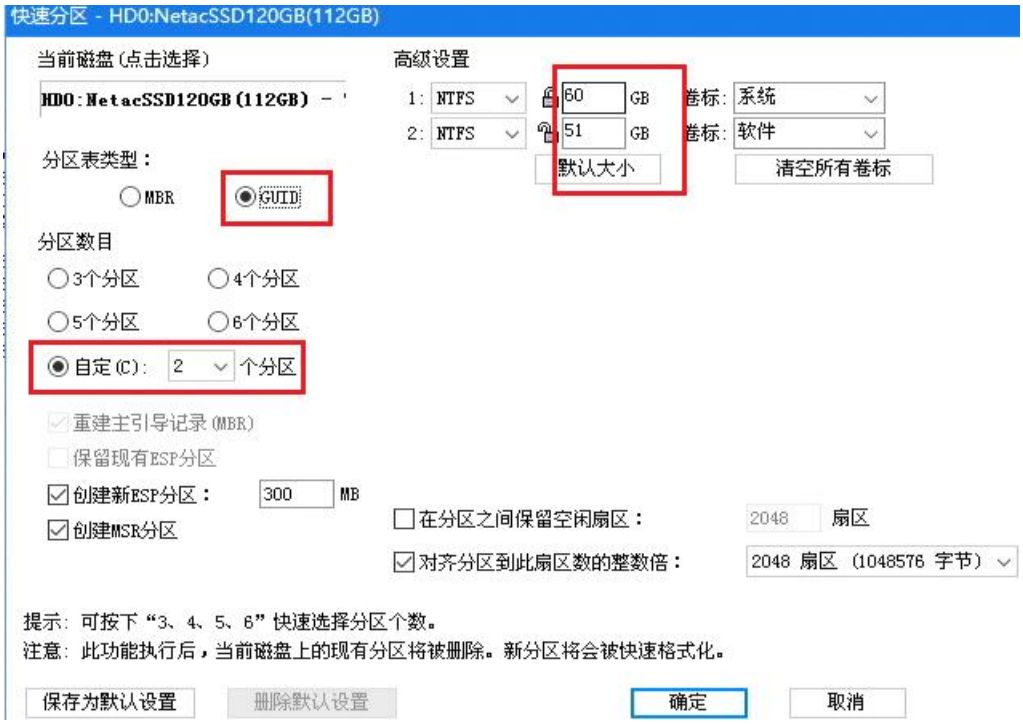
之后便进入了 PE 系统的桌面，如上图，带有一些维护工具，自带的那个分区工具 DiskGenius 版本较低，不能转换磁盘分区表类型（GPT 和 MBR），所以要到我们之前复制到启动 U 盘里，去运行 5.2 版本的 DiskGenius5.2



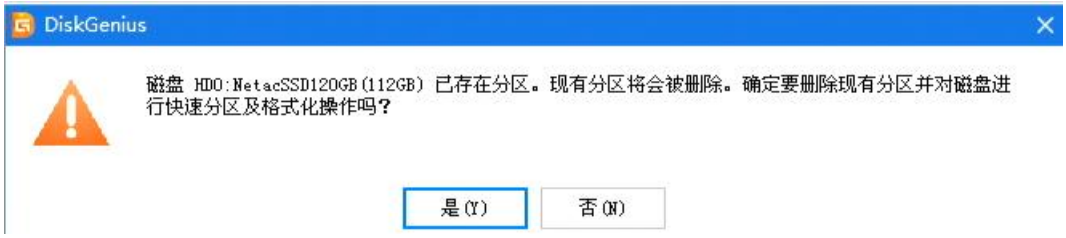


双击 DiskGenius.exe 运行磁盘精灵

选中目标计算机上的磁盘，然后点击工具栏的“快速分区”



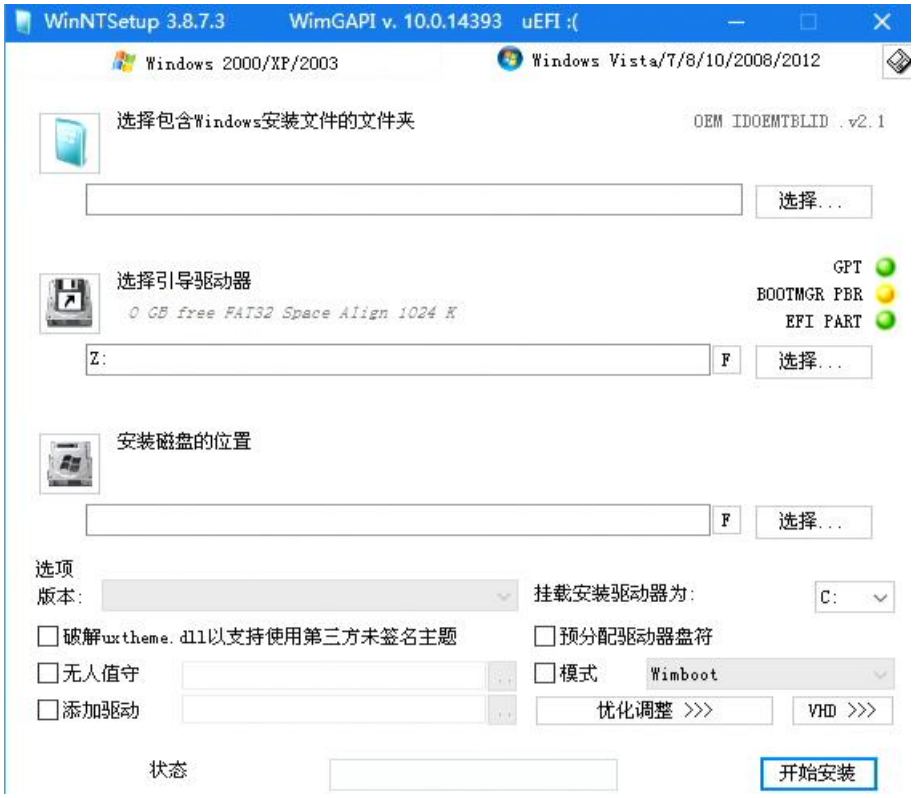
快速分区的参数如上，分区表类型为 GUID（即 GPT 分区表），我们分 2 个主要的分区，一个 60GB 作为 C 盘，一个 51GB 作为 D 盘，还有额外的分区，一个 ESP 分区和 MSR 分区，

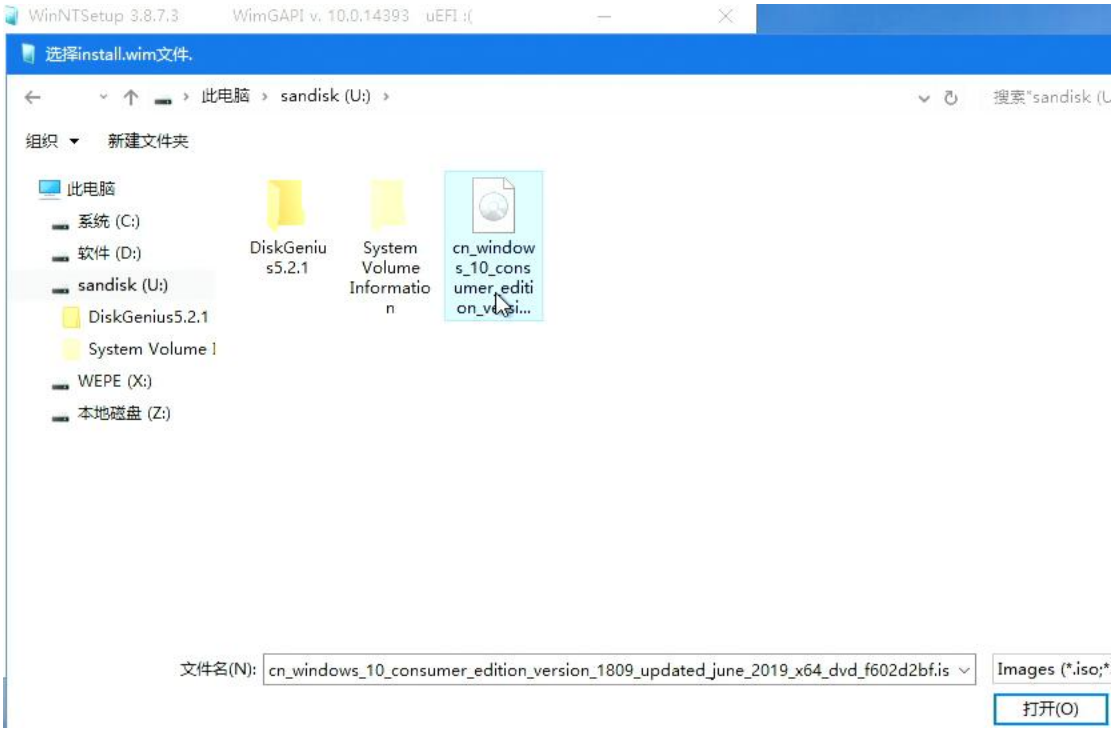


确保原来的磁盘里数据都备份了，就可以进行快速分区了，点击“是”

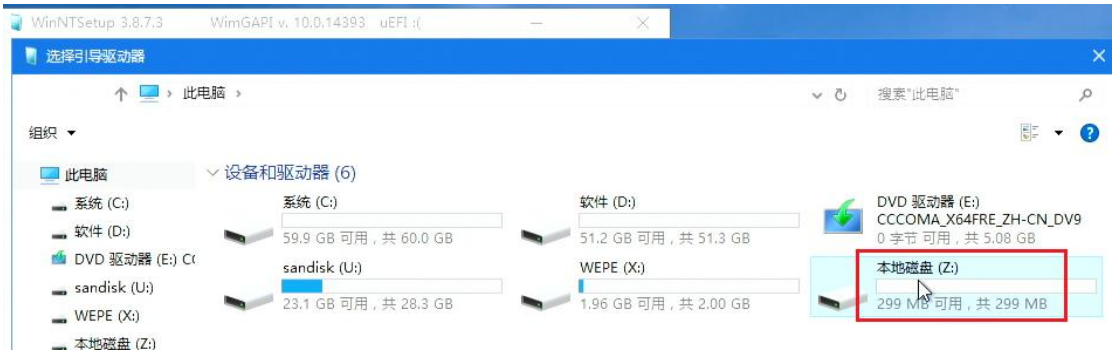
如上图，分区完成，记住各分区的大小，

回到 PE 系统的桌面，点击 Windows 安装器，这个就是 WinNTsetup.exe 工具，微软官方推出的用来安装 windows 系统的，

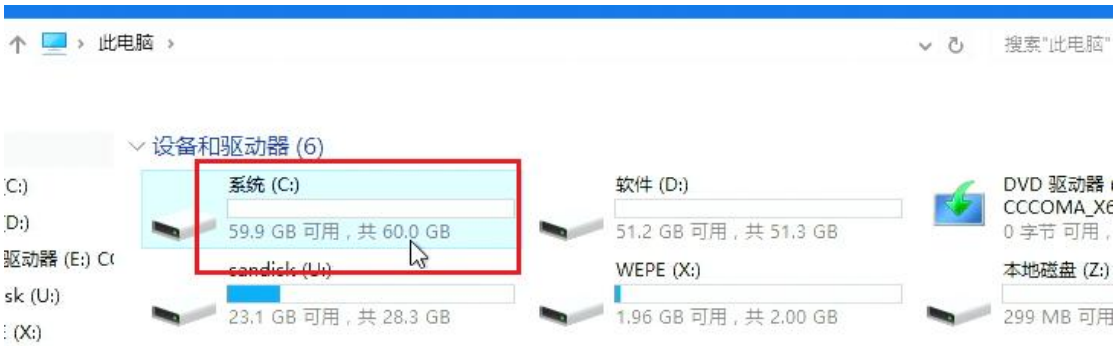




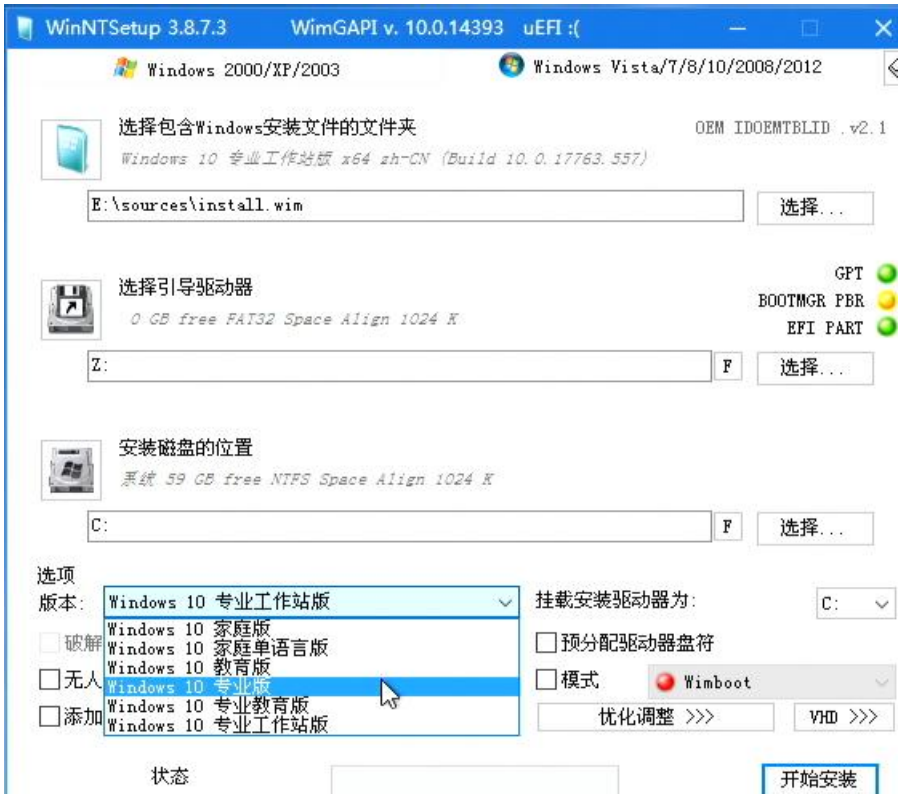
在“选择包含 Windows 安装文件的文件夹”那里，选择我们之前复制到 U 盘里的 win10 镜像文件，“打开”



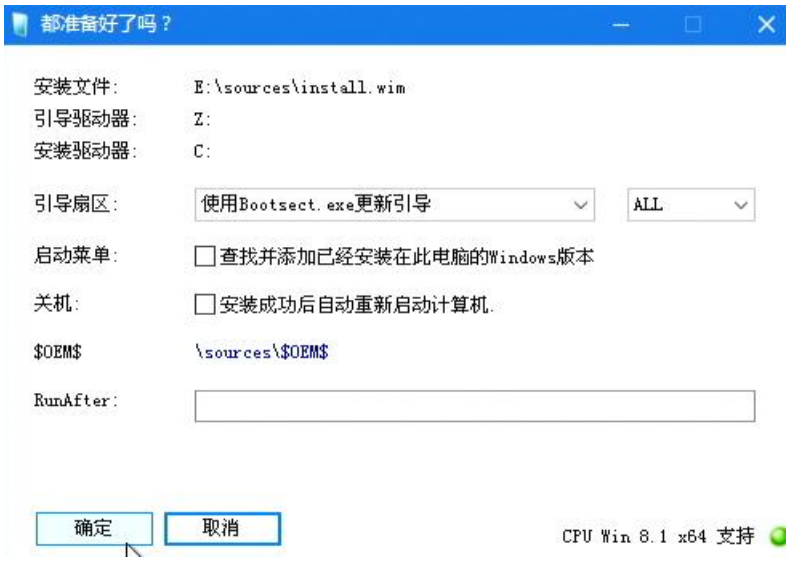
在“选择引导驱动器”那里选择大小约 300M 的 ESP 分区，在 PE 系统里显示为 Z 盘，大小为 299MB，



在“安装磁盘的位置”那里，选择 C 盘，60GB 的那个。

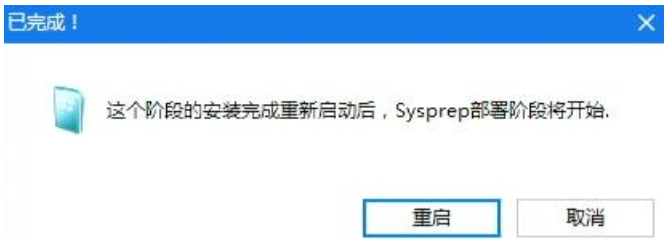


然后选择要安装的 win10 版本，如上图，再点击右下角的“开始安装”



点击“确定”，等待安装完成，大约 5 到 15 分钟，

然后弹出已完成的提示框：



点击“重启”，然后系统并没有重启，因为 PE 系统功能有限，组件有限，该程序的重启操作并没有让系统感知，所以要到 PE 的 win 菜单里，手动点击重启，

系统黑屏后，马上拔出 U 盘，等待系统完成最后的部署，

看到上图，就知道系统已经安装完成了，接下来就是个性化的设置了

小结：

使用 PE 制作工具进行制作 PE 优盘，然后把下载的 windows 系统及 5.0 以上版本的 DiskGenius 工具复制到 U 盘里。再把 PE 优盘插到目标计算机，启动计算机时选择从 U 盘启动，进入 PE 系统后，先使用 DiskGenius 进行磁盘分区的划分，再使用 winNTsetup.exe（Windows 安装器）进行系统的安装。

第 19 章、PE 制作软件制作 PE 优盘的原理

我们很好奇 那些 PE 制作工具软件 是怎么把我们的普通 U 盘制作成了 PE 启动盘呢？
我们来探究一下。

如上图，在制作时，一般是使用全能三分区的方式，其他的如老毛桃、大白菜、U 大师等也基本上是这么做的。

什么是三分区？我们用 DiskGenius 磁盘精灵查看一下我们的 U 盘制作后，究竟变成了什么情况，

上图可见，PE 优盘有 2 个分区，分区 0 为 sandisk(H:)，分区文件系统为 exFAT，
分区 1 为 EFI(I:)，分区文件系统为 FAT16
还有一个分区呢？在哪？注意看上面的那个蓝色的像进度条一样的东西，

在分区前面还有一小部分“空闲”的区域，点击一下查看属性，

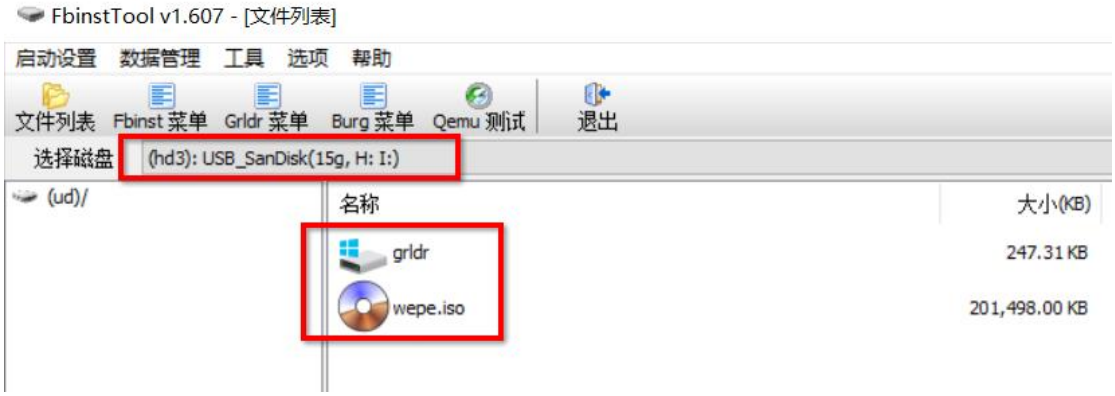
该空闲部分有 300MB 大小，第三个分区很可能就是这里，只是由于分区类型特殊，导致 DiskGenius 工具无法识别出来。

没错，该空闲部分是 Fbinst 的 UD 区，什么是 Fbinst 和 UD 区呢？

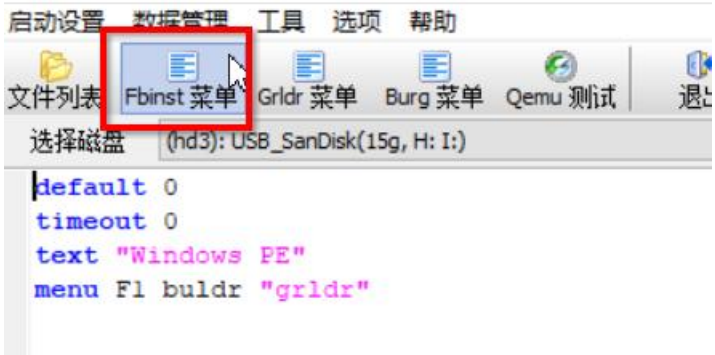
Fbinst 是一种 MBR 引导，和 windows NT5.x 6.x 是并列的关系，只是引导的过程和做法不一样，Fbinst 是开源社区的 bean 大神开发的，该引导基本解决了 BIOS 下的 CHS/LBA 参数适应调整问题，使 U 盘 BOOT 成功率大大提高。
因为 Fbinst 是 MBR 引导，所以只在传统的 BIOS 启动模式下有效，在 UEFI 启动模式下是无法识别该引导的。

在传统的 Legacy BIOS 启动模式下，预启动程序会读取 U 盘的 MBR 扇区，然后执行 MBR 里的引导代码，Fbinst 引导代码能识别 UD 区的文件，它根据 Fbinst 菜单的指引会去寻找 UD 区里的 grldr 文件（类似于 NT6.x 引导去寻找 bootmgr 一样），找到后，把控制权交给 grldr，再读取 grldr 菜单，显示出各启动菜单项，供用户选择，用户选择后，再从指定的启动项启动。

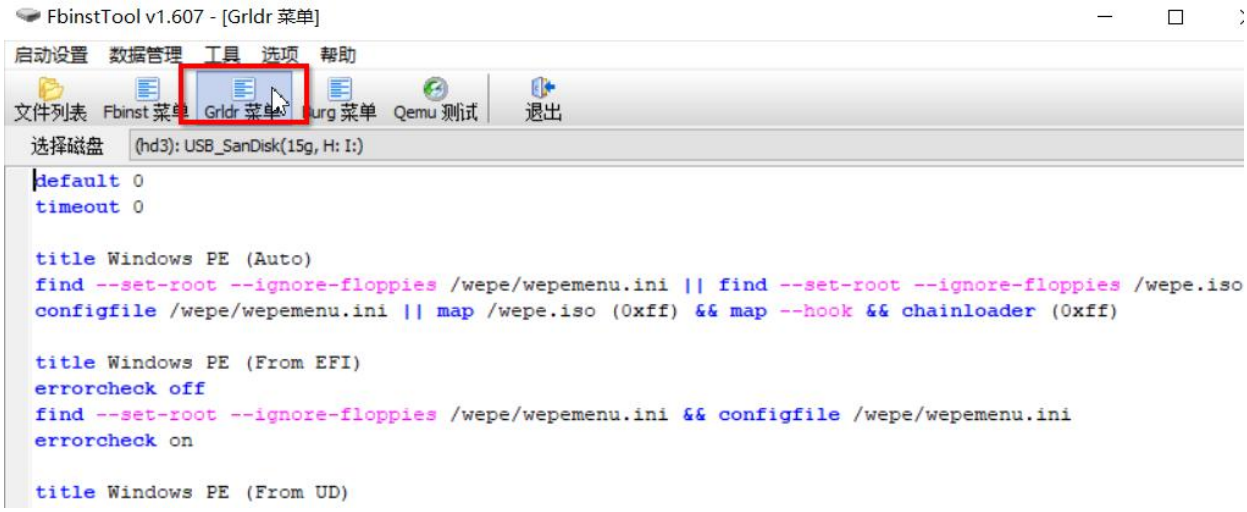
怎么查看 Fbinst 的 UD 区里的文件呢？可以从网上下载 FbinstTool 工具软件查看，FbinstTool 好像没有官网，要下载 1.6 的版本，不要用 1.7 的，因为目前各 PE 制作软件使用的都是 1.6 的版本。



打开 FbinstTool1.6 工具软件，可以看到 PE 优盘有一个 UD 区，里面只有 2 个文件，不同的 PE 制作软件制作出来的 UD 区文件结构不尽相同，上图是 wepe 微 PE 制作出的 UD 区的文件结构。
grldr 文件是可以内置启动菜单的，



首先查看 Fbinst 菜单，点击菜单栏的“Fbinst 菜单”查看，如上图。
只有一个 menu 菜单项，就是去寻找并执行 grldr 文件，



如上图，点击工具栏的“Grldr 菜单”可以查看 grldr 启动菜单

timeout 0 表示不给时间让人们选择要进入的启动项，default 0 表示默认的启动项为 0 号启动项（即 title Windows PE(Auto) 这个）

find 后表示要查找的文件（wepe.iso），找到后就装入内存，并解压、启动该 PE 系统。

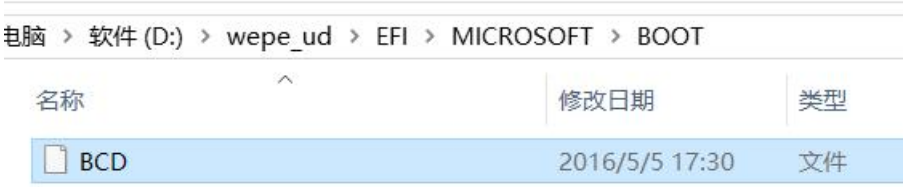
wepe.iso 就是我们上一章进入的 PE 系统镜像文件，有 200MB 大小，我们可以从 UD 区导出来看看，

右键点击 wepe.iso 文件，选择“导出文件”

导出到 D 盘，名为 wepe_ud.iso

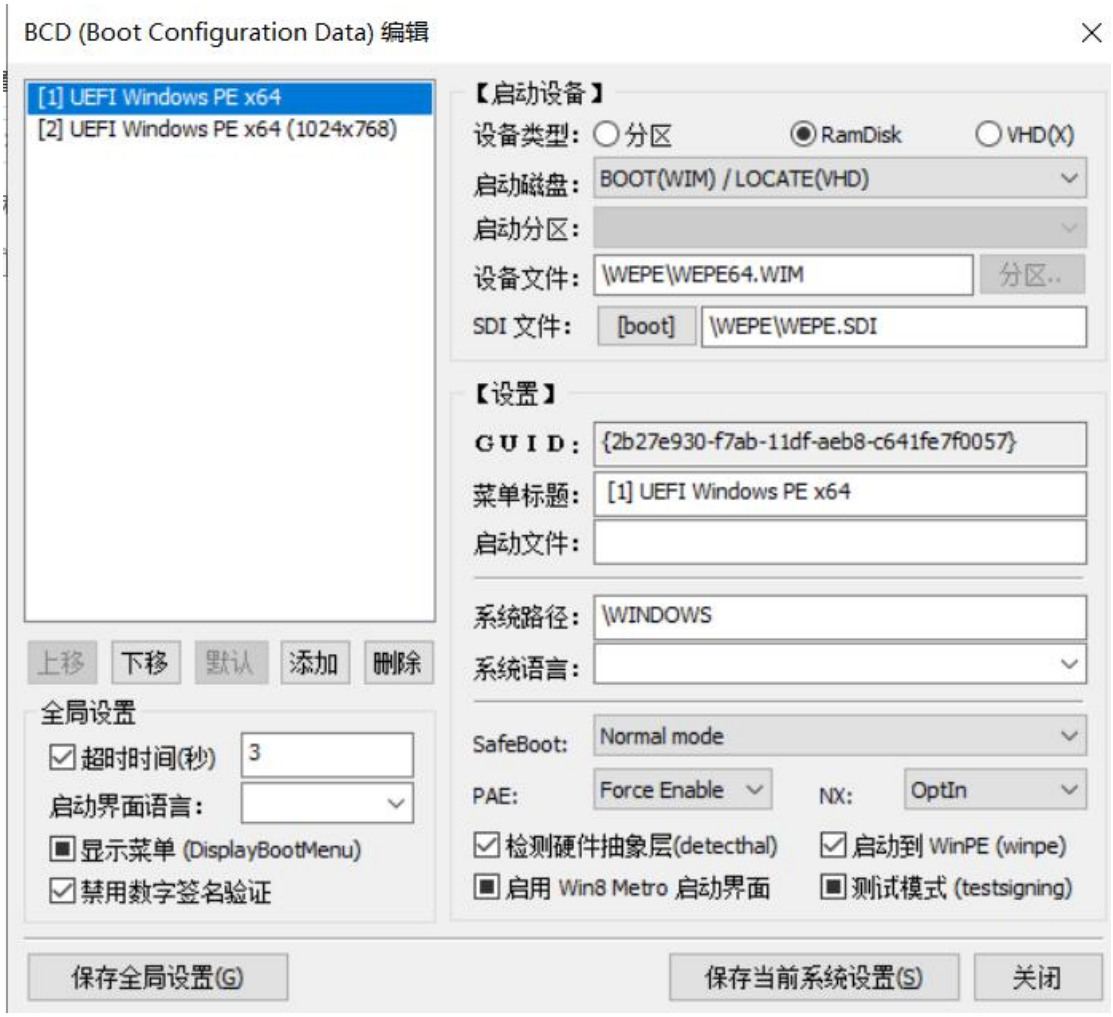
然后解压此 iso 文件到 wepe_ud 文件夹里

进入 wepe_ud 文件夹查看一下有什么文件，
如上图，有 bootmgr 文件，



该 bootmgr 文件会去读取\EFI\microsoft\boot\里的 bcd 文件，

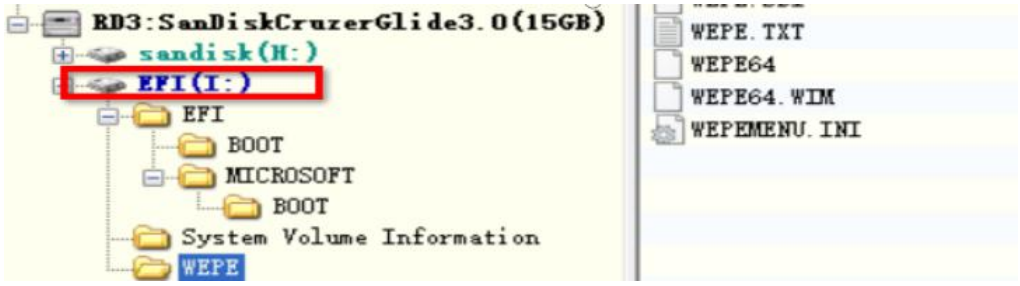
我们用 Bootice 工具查看一下该 bcd 启动菜单，



该 bcd 文件有 2 个启动菜单项，其实都是一样的，就是从 RamDisk 设备启动，设备文件为\wepe\wepe64.wim，sdi 文件为\wepe\wepe.sdi

基本上就是这样，wepe.iso 文件就是一个系统镜像，启动后，加载 bootmgr 再读取 bcd 菜单，再加载 wepe64.wim 映像，启动 PE 系统。

该 wepe.iso 文件解压后，得到的文件就是 PE 优盘的 EFI 分区里的文件，



该 EFI 分区文件类型为 FAT16，大小 298MB，和 UD 区的 300MB 差不多

综上，PE 优盘的三分区方案，就是：

1.划分一个 300MB 的 UD 区，里面主要是 grldr 和 wepe.iso 系统镜像文件

2.再划分一个 FAT16 分区，分区卷标为 EFI，大小约 300M，里面的文件就是 UD 区里的 wepe.iso 的解压文件

3.最后剩下的所有磁盘空间单独再划为一个 exFAT 格式的分区。

为什么要做成三分区？

UD 区是为了从传统的 BIOS 正常启动（兼容更多的老旧设备），FAT16 分区是为了从 UEFI 启动（该 FAT16 分区是隐藏的，一般在电脑上看不到，不过 win10 系统里可见），exFAT 分区是为了存放文件，存放 windows 安装包等大文件，作为正常的 U 盘使用。

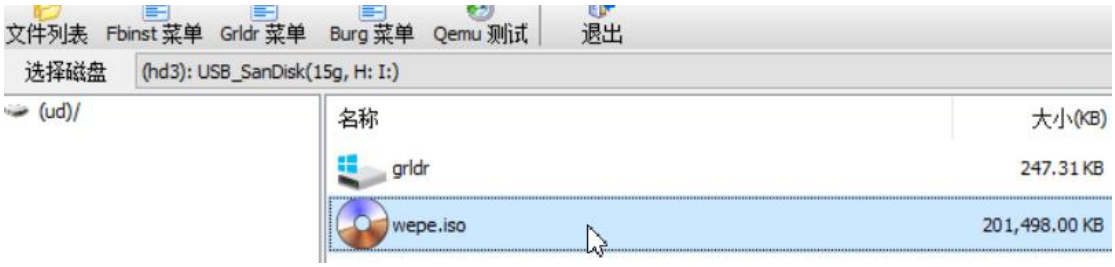
第 20 章、个性化 PE 系统

目前网上的 PE 制作软件非常多，可是制作出来的 PE 系统里的工具软件总有一些是没有及时更新的，虽然制作 PE 的软件（如 wepe，老毛桃等）是有最新版本，但它们制作出的那个 pe 系统的镜像还是比较旧的，PE 系统启动后，里面自带的软件版本较低，功能可能不够用。我们有没有办法自己更新里面的工具软件呢？答案是可以的。

大概的做法是把 PE 系统映像解压到某目录下，再修改里面的内容，更换新的软件进去，再打包成新的映像文件。

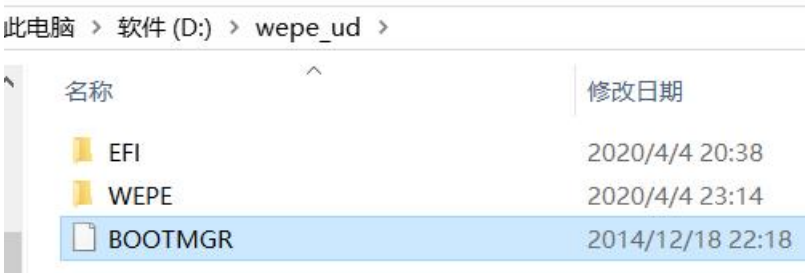
以 wepe 微 PE 为例，微 PE 一般默认的做法是制作成三分区的，UD 区和 EFI 分区里有 PE 系统，另一分区是用来存放个人文件的。

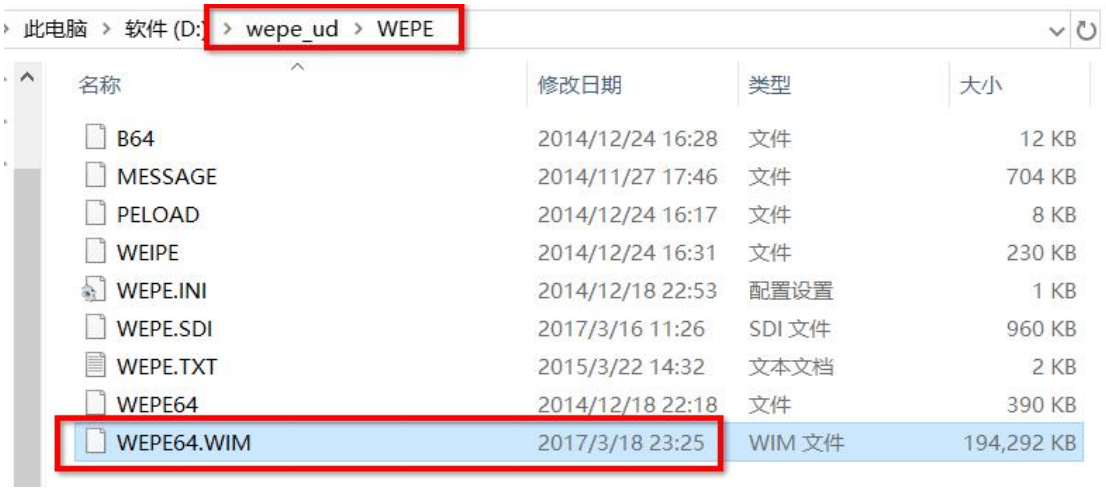
我们使用 FbinstToolv1.6 工具先把 UD 区里的 wepe.iso 导出来，再解压到 wepe_ud 目录下，



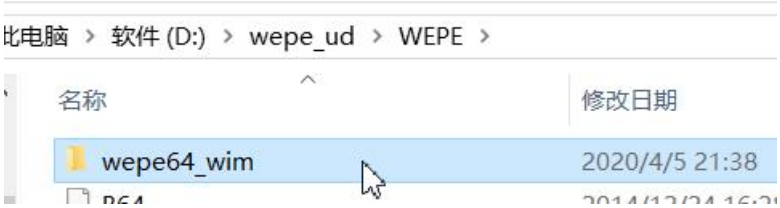
右键点击 wepe.iso 文件，选择“导出文件”导出到 D 盘，名为 wepe_ud.iso

然后解压此 iso 文件到 wepe_ud 文件夹里





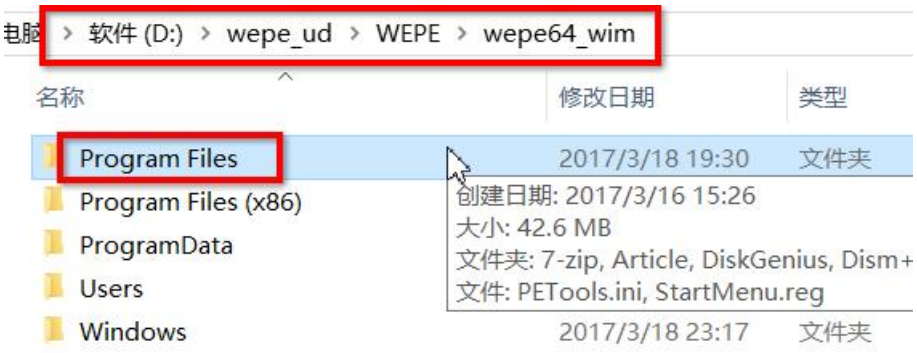
进入 wepe_ud 解压目录下的 WEPE 目录下，可见 WEPE64.WIM 此映像文件是 2017 年 3 月的，已经三年不更新了，我们要修改的就是此文件，先在当前目录下创建一个用于解压 wepe64.wim 映像的文件夹，名为 wepe64_wim



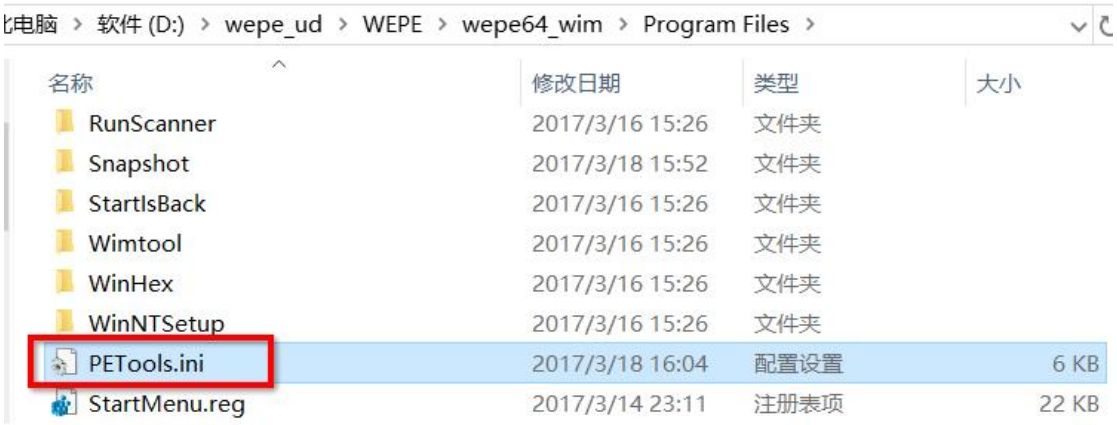
然后**以管理员身份运行** WimTool.exe 工具软件，用于解压 wim 映像文件的

如上图，“选择要挂载或解开的 wim 文件”为我们解开的 wepe.iso 镜像里的 wepe64.wim 文件，“选择挂载或解开的目标文件夹”为我们之前创建的 wepe64_wim 文件夹，点击右下角的“解开映像”

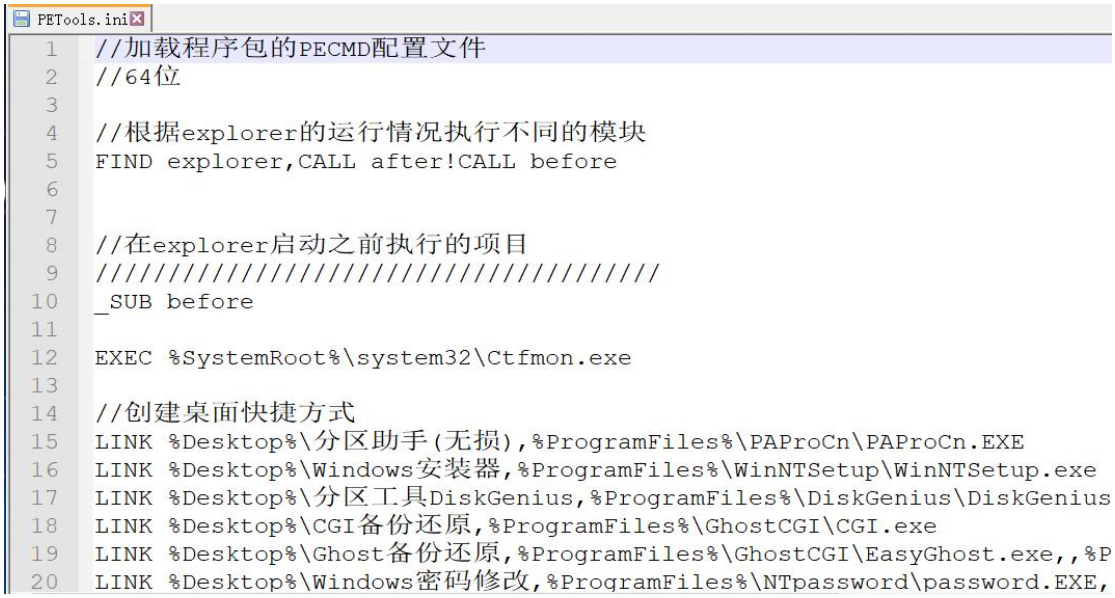
待到 100%完成后，就行了，进入 wepe64_wim 挂载目录看看



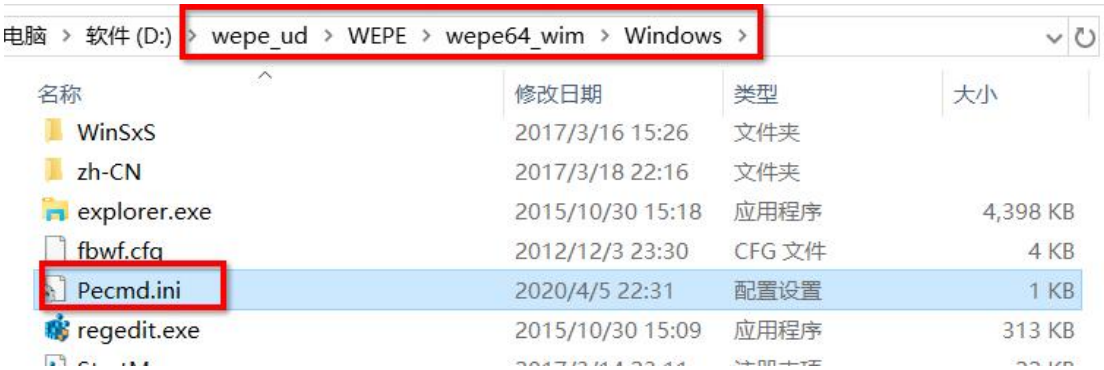
wepe64_wim 映像挂载目录里的 Program Files 目录里面就是进入 PE 系统后的工具软件，我们先把里面的软件更新一下，用新的替换旧的，也可添加原来没有的。



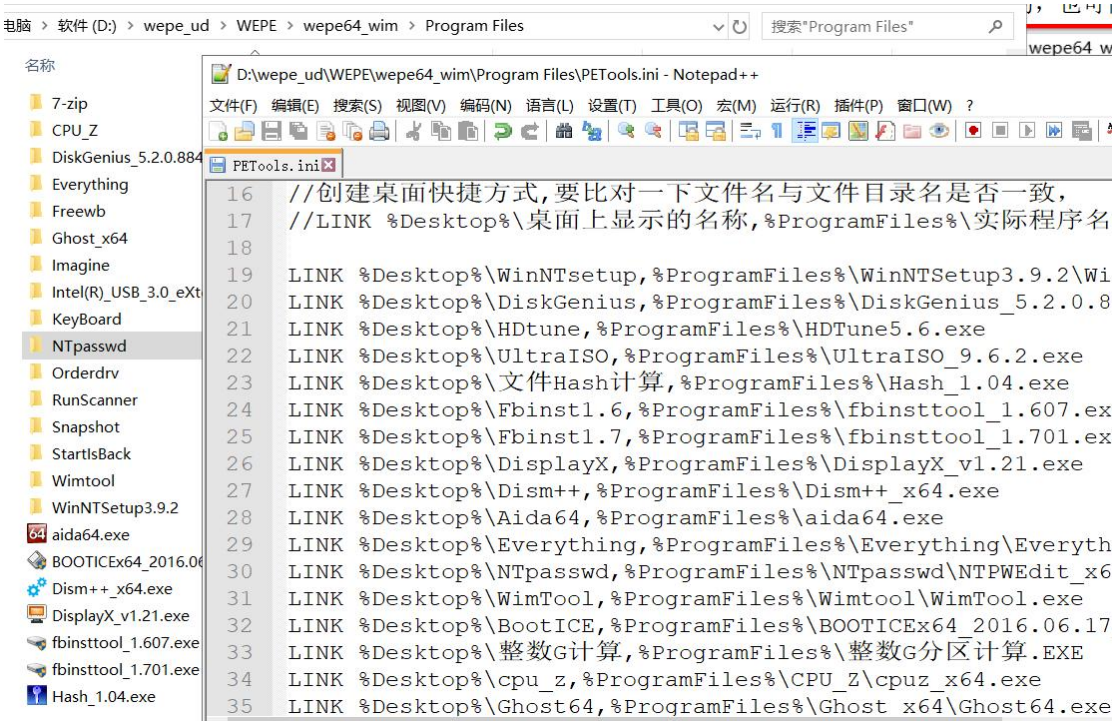
注意到那个 PETools.ini 文件了吗，它是进入 PE 系统后加载的配置文件，因为工具软件更新了，新的工具软件名称可能变了，所以我们也要更新一下这个配置文件，用文本编辑工具打开此 petools.ini 文件



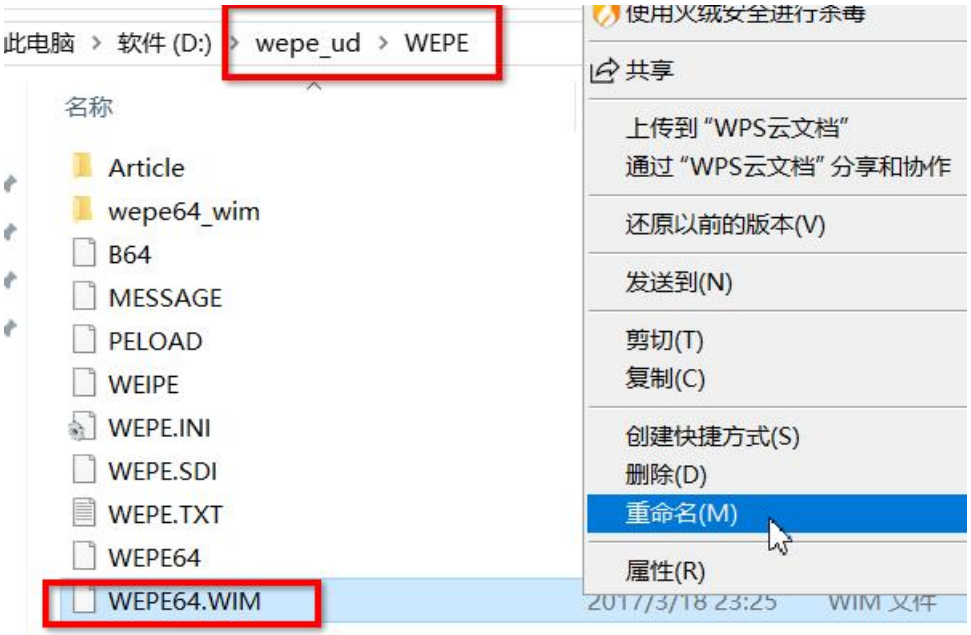
这个配置文件其实是 PECMD 的配置文件之一，我们可以照着原有的修改一番，桌面快捷方式那里可以添加新的，也可修改旧的，改完后，保存即可。



另一个配置文件是 wepe64_wim\Windows\里的 Pecmd.ini ，感举趣的也可以打开看看，一般这里不用修改

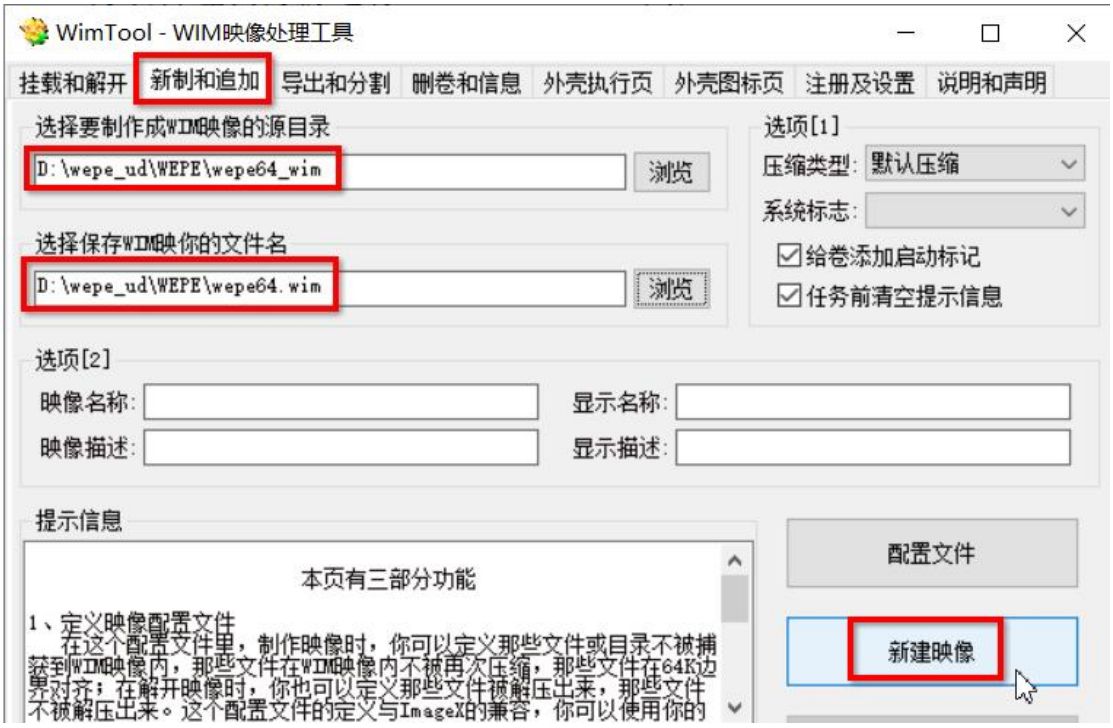


如上图，我们更新完工具软件，修改完配置文件，就可以保存配置文件，然后把这个 wepe64_wim 文件夹再打包成 wim 映像文件，用于替换原来的 wepe64.wim



先把原来的 wepe64.wim 文件备份一下，改名为 wepe64_bak.wim，

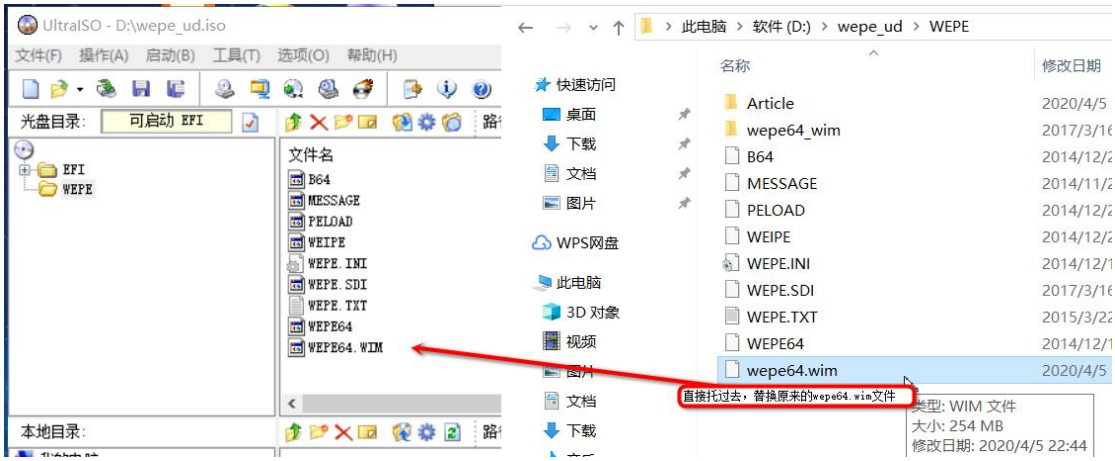
再以管理员身份运行 WimTool.exe 工具，

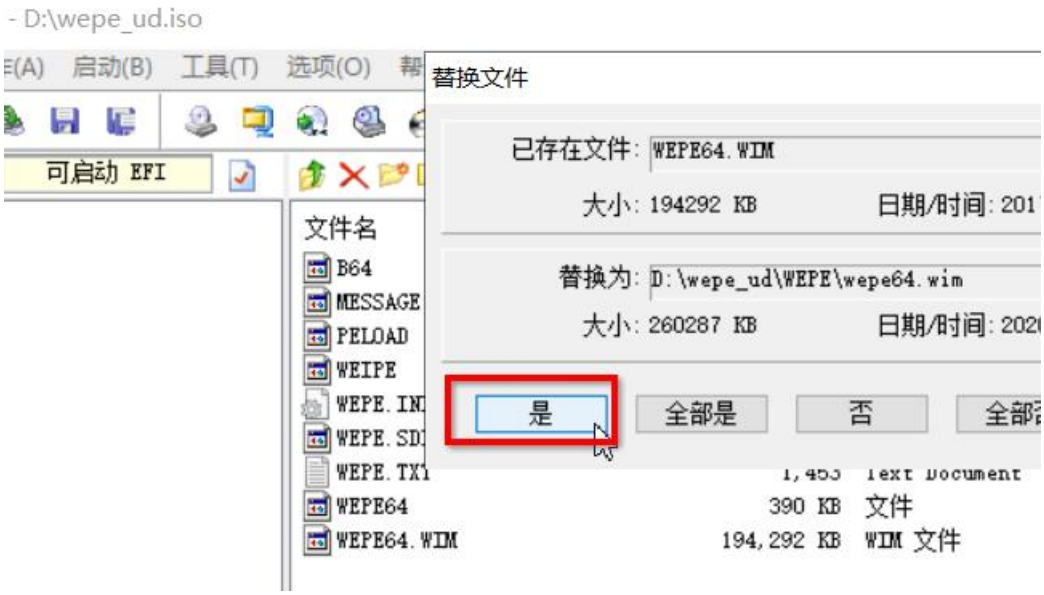


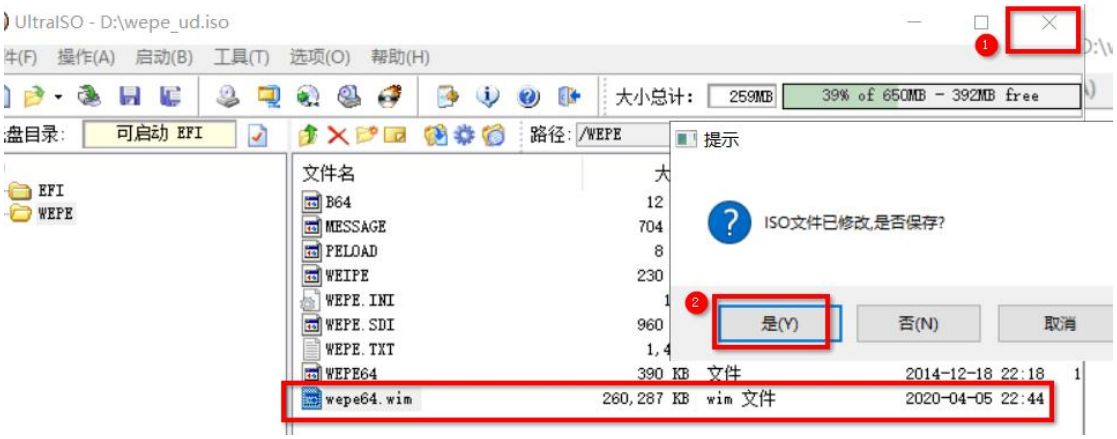
新的 wim 映像文件名要和原来的一致，点击“新建映像”

等到 100%完成即可，

然后用 UltraISO 打开 UD 区里的 wepe.iso 文件，用新的 wepe64.wim 替换 iso 文件里的\WEPE\WEPE64.wim

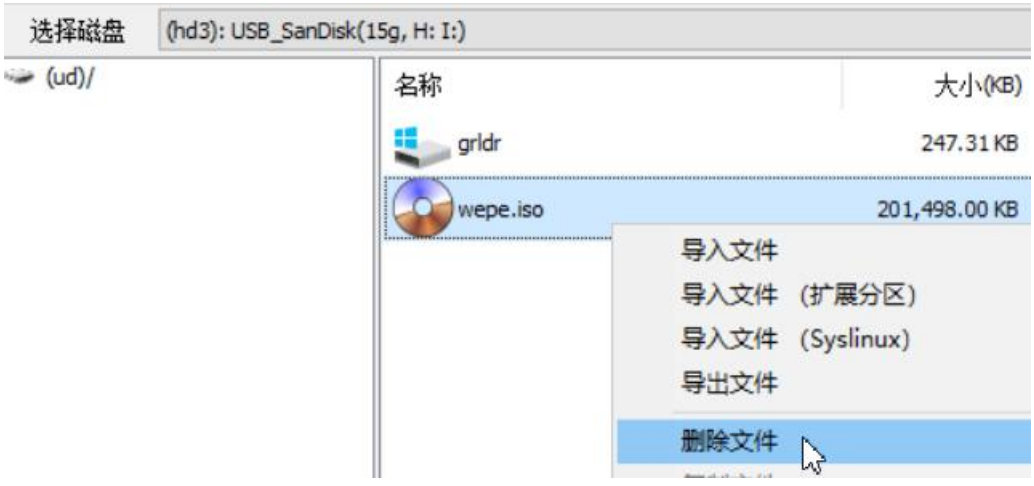


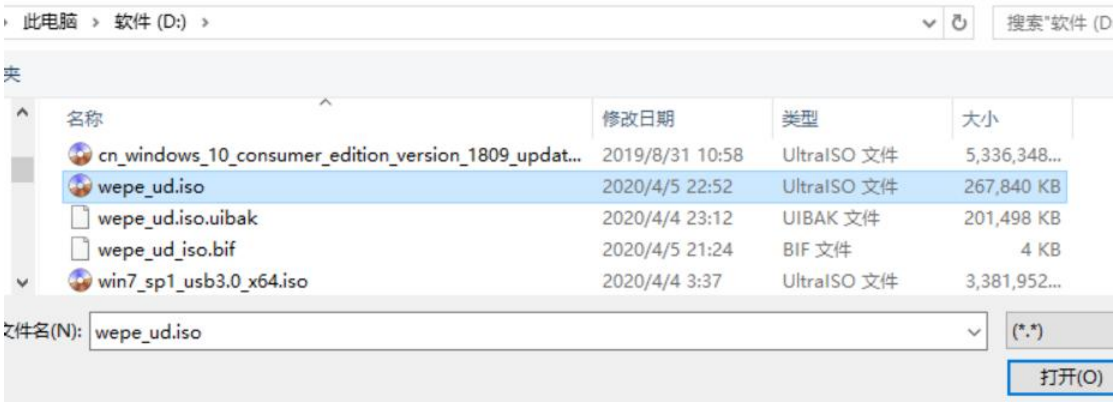




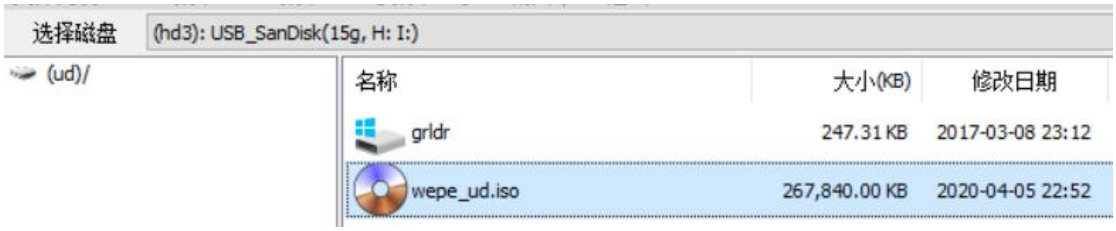
替换之后，过几秒再关闭 UltraISO，会弹出是否保存的提示，点击“是”

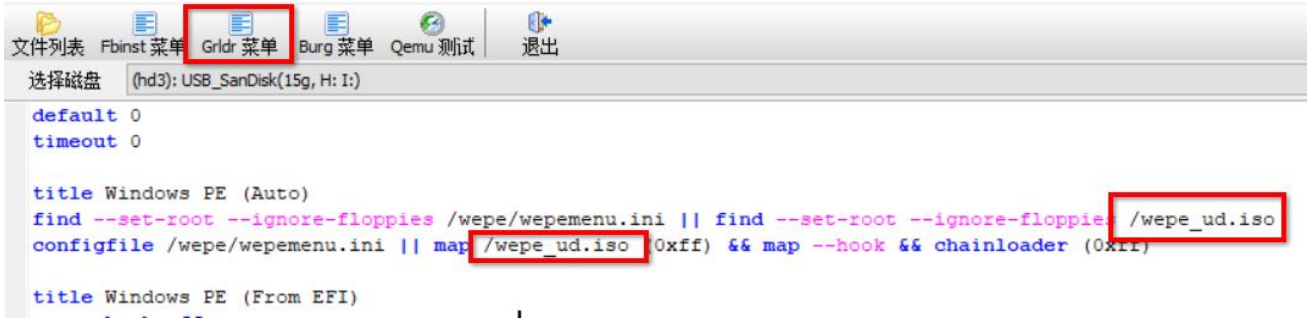
最后，插入 PE 优盘，打开 FbinstTool_1.6 工具，删除 UD 区原来的 wepe.iso，再导入新的 wepe_ud.iso 文件即可，文件名称和原来的不一样了，所以还要修改 grldr 菜单。



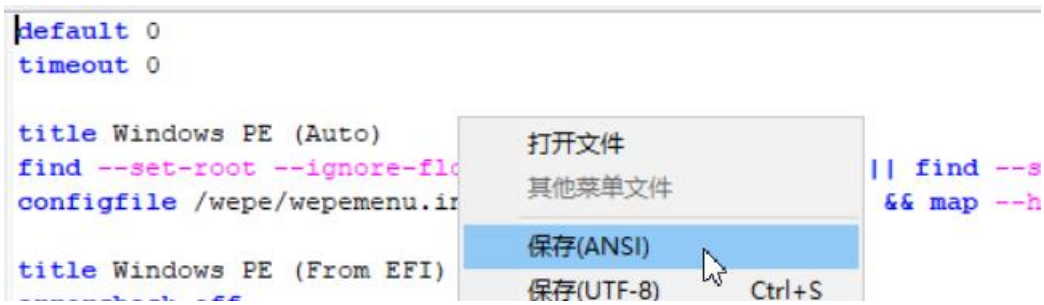


导入速度比较慢，待导入完成，就去修改 grldr 菜单





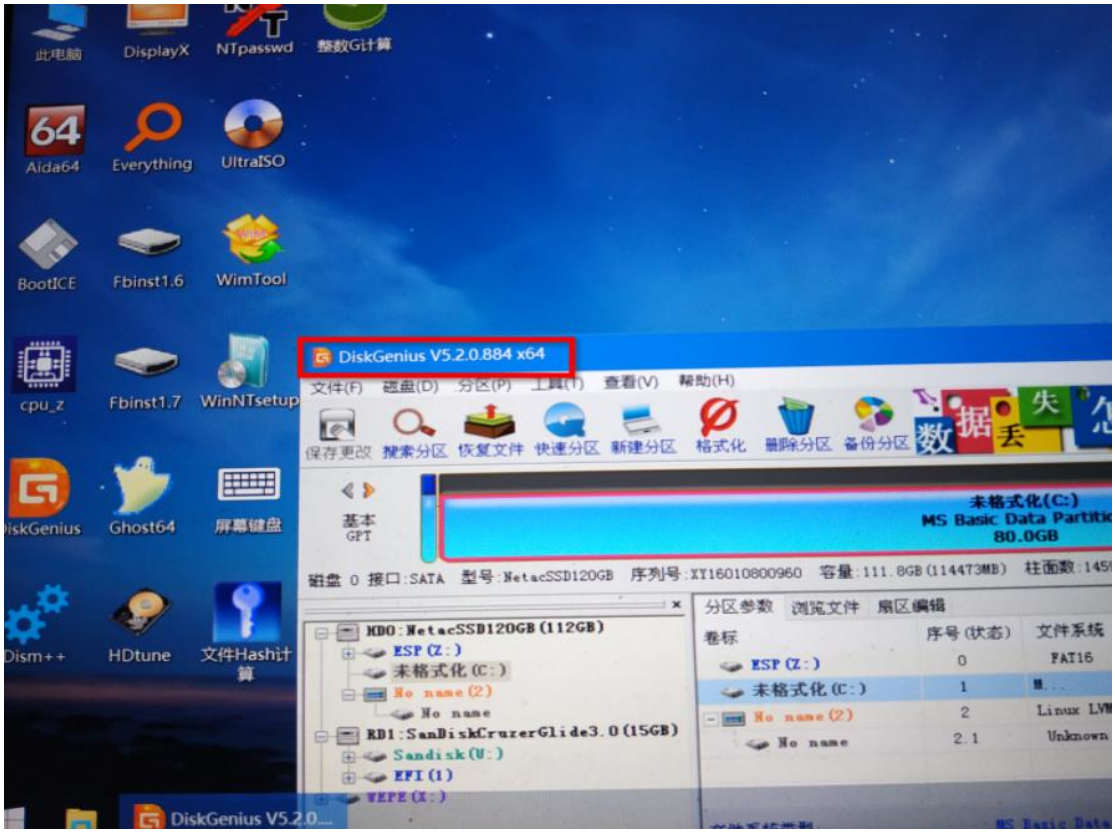
如上图，把 grldr 菜单里的 wepe.iso 都改为 wepe_ud.iso 就行，再右键菜单内容，点击“保存 ANSI”



保存后，就可以关闭 FbinstTool 了，

UD 区的文件替换后，再把 EFI 区的，里面的\wepe\wepe64.wim 也替换成刚刚修改过的 wepe64.wim，这样就大功告成了（EFI 分区空间不够，要先删除原来的，再复制新的过去）

修改替换后，我们验证一下是否成功了，把 U 盘插上计算机，启动时选择从 U 盘启动，进入 PE 系统看看，



可见已经成功了，工具软件已被替换成新的版本，桌面上也出现了我们刚刚添加的新工具。

第 21 章、PE 启动 U 盘非得是三分区格式吗？

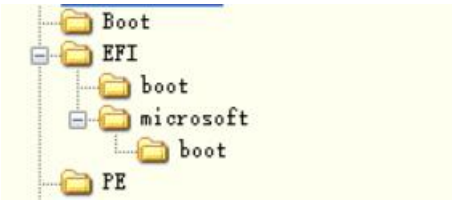
大多数的 PE 启动盘制作工具都是按全能三分区的方式去制作 PE 启动盘的。这样做有它的好处，不过也有些麻烦，因为 windows 10 能识别出那个 300MB 左右的卷标名为 EFI 的分区，它是隐藏的，一般 windows 10 以前的系统不会识别出来，但 win10 能识别出来，插上 U 盘后，总觉得弹出 2 个 U 盘分区不太美观。

重点是这 2 个分区都是激活的（活动的），有可能会报错，而且最为致命的是，当我们进入 PE 系统后，有可能 PE 系统只识别出了其中的一个分区（那个小的），没识别出大的那个（我们存放了系统 iso 镜像的那个盘符），这样进入了 PE 系统也没法正常安装系统了。

所以作者给出一个**只做一个分区的方案**，先把 PE 启动盘的 EFI 隐藏分区里的文件，导出到某个磁盘下，我们自己重新分区，重新制作 PE 启动盘，以后要是有新的 U 盘要制作，也可以自己制作了，不用再借助 PE 制作工具软件了。PE 启动盘的 EFI 分区里的文件，我们只要其中的 wim 映像文件和引导文件就行，就是那个 boot.wim 或 wepe64.wim 文件（PE 系统映像文件），还有\EFI 目录及\boot 目录，微 PE 的 EFI 分区里只有 EFI 的引导，没有 boot 引导目录，可以从自己的 windows 系统启动分区里复制出来，用磁盘精灵导出。

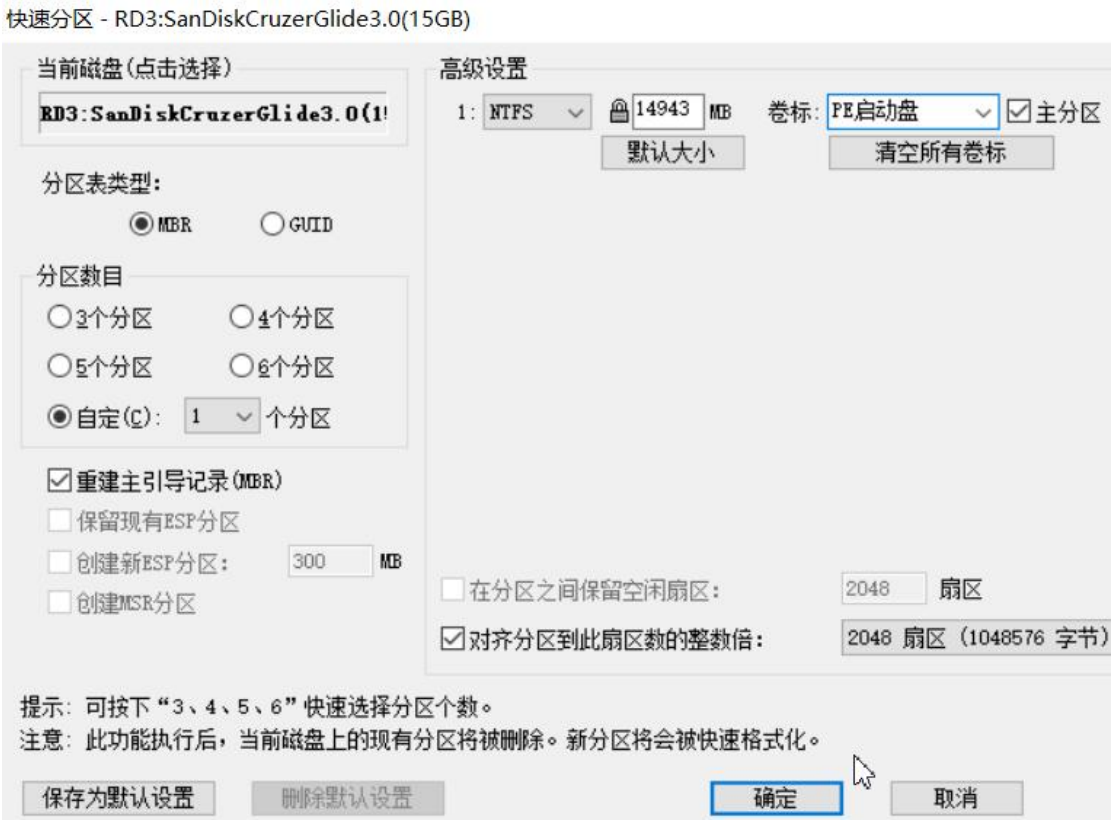
①准备启动文件

为了能在传统的 BIOS 模式下启动以及在 UEFI 模式下启动，所以 2 种模式下的启动文件都要有，
传统的 BIOS 模式下的启动文件在 Boot 目录下，有 boot.sdi 和 bcd 菜单文件
UEFI 模式下的启动文件在 EFI 目录下，有\boot\bootx64.efi 和\microsoft\boot\bcd
PE 目录是用来存放我们修改过的 PE 映像 wim 文件
bootmgr 是 BIOS 模式下要用到的文件，



目录结构如上图，总的来说就是要有 2 种启动模式的引导文件和 PE 系统映像文件，

②文件准备好了，就可以使用磁盘精灵去格式化 U 盘，要用磁盘工具去 U 盘上创建分区，只创建一个分区，

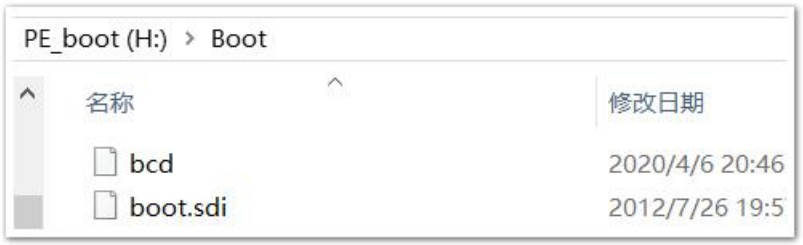


创建完分区后，再格式化分区为 NTFS 文件系统，

③再把准备的文件都复制到 U 盘里，U 盘根目录下结构如下：

bootmgr 和\Boot 目录是传统 BIOS 启动需要的

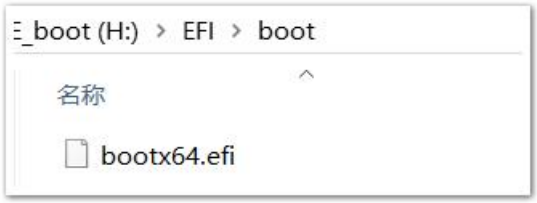
\Boot 目录里的文件如下：



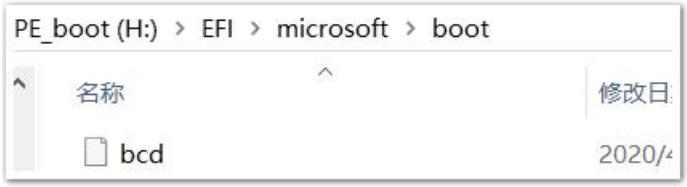
\EFI 目录里的文件（夹）如下：

\EFI 目录是 UEFI 模式启动时需要的！

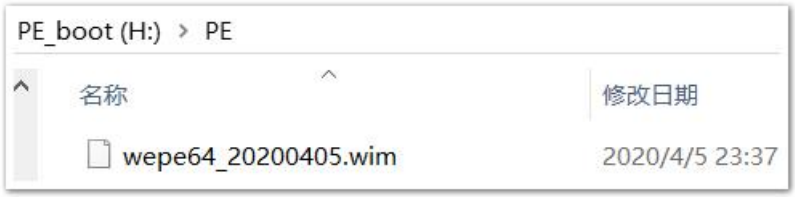
\EFI\boot 目录：



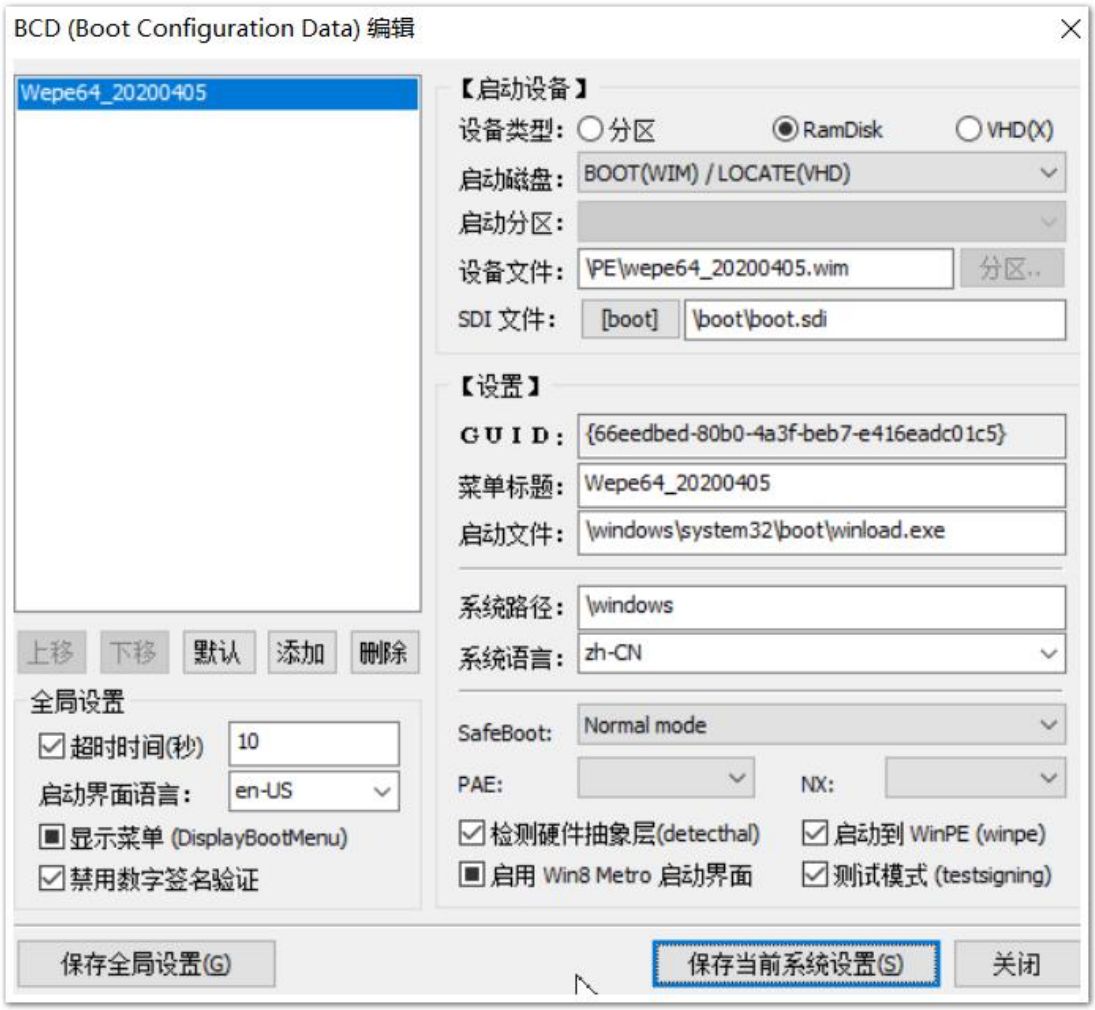
\EFI\microsoft\boot 目录：



\PE 目录：（存放修改过的 PE 系统映像文件）可有多个，名称可自定义

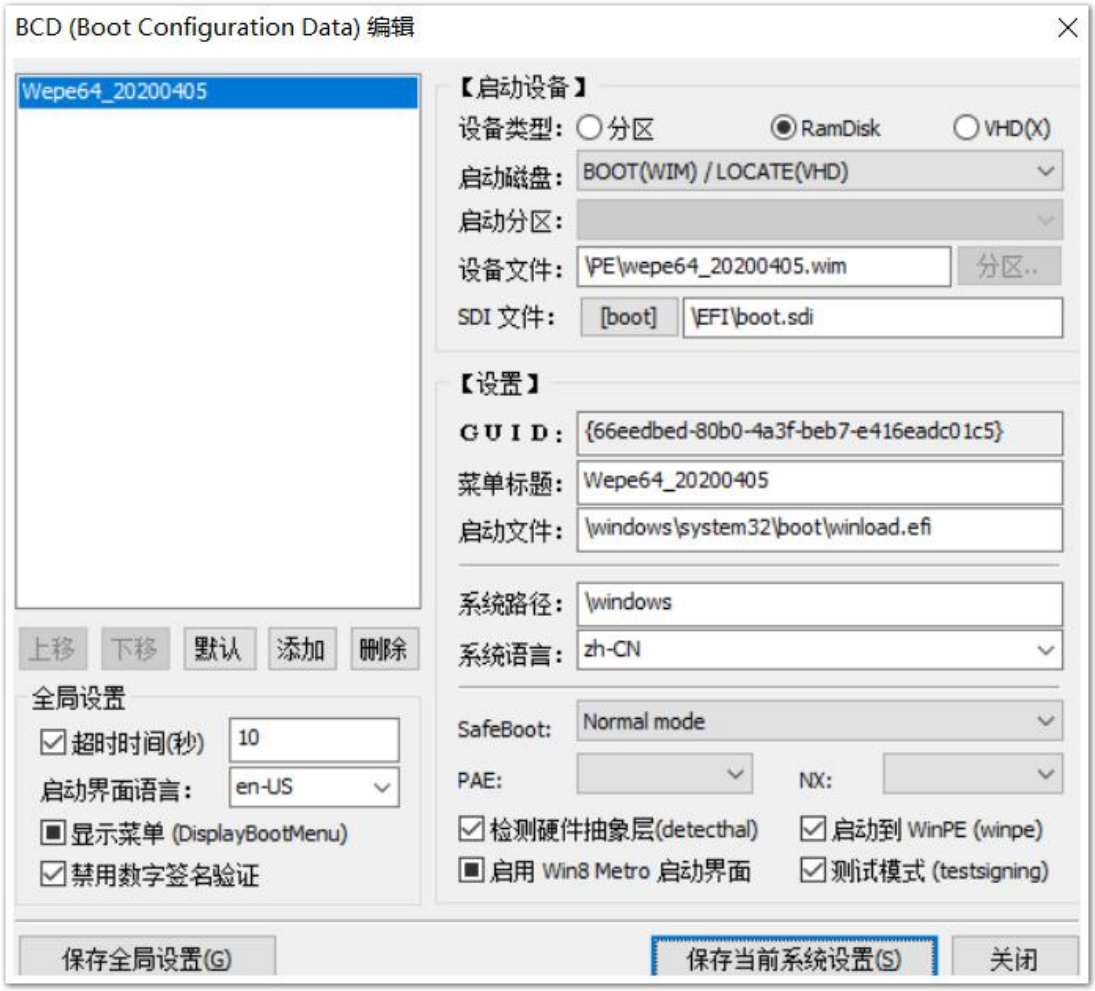


④修改启动菜单项 bcd 文件，2 种启动模式下的 bcd 文件都要修改
先用 BootICE 工具修改 Legacy 传统 BIOS 启动下的\boot\bcd 文件

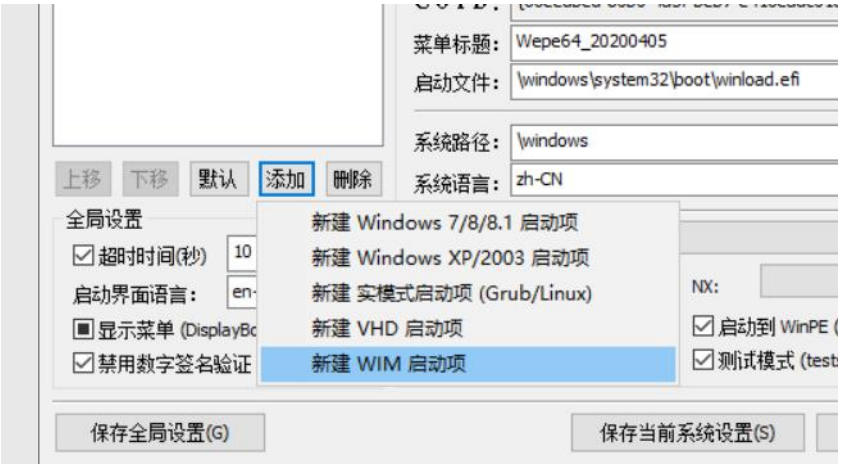


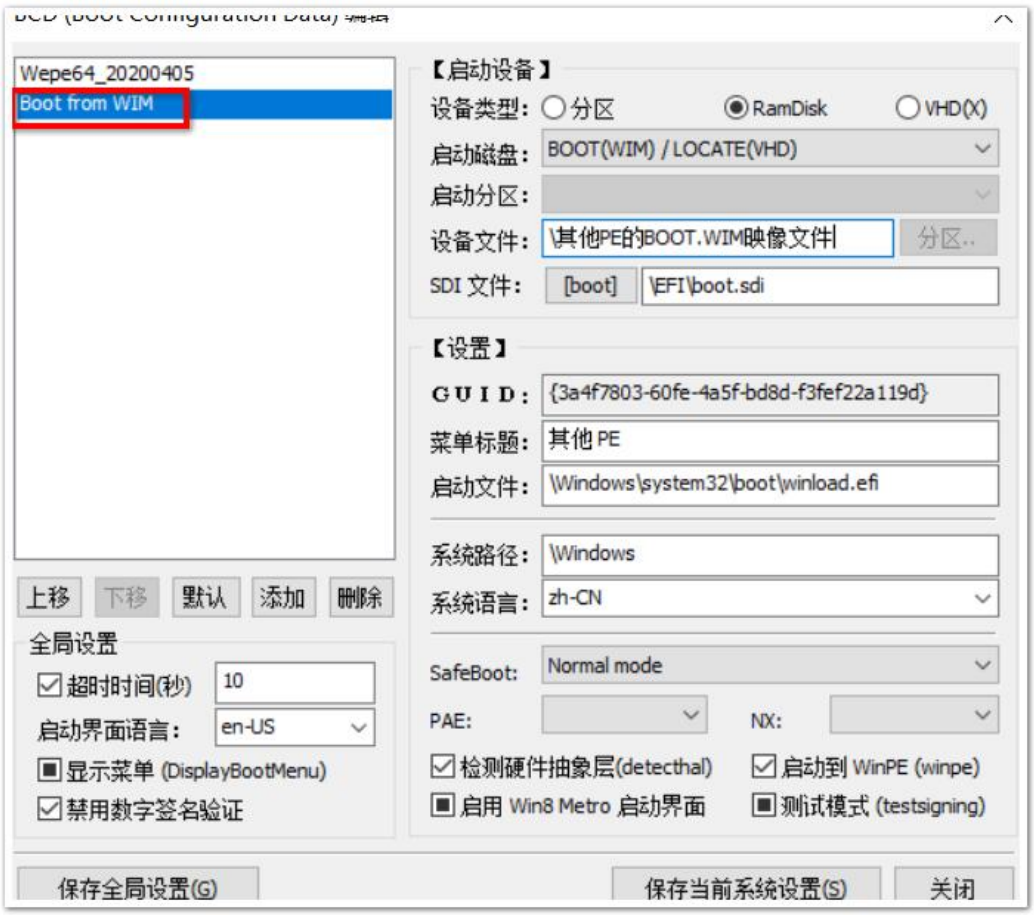
修改如上，设备类型为 RamDisk，SDI 文件为\boot\boot.sdi 文件，因为是 LegacyBIOS 启动，所以系统加载文件为 winload.exe，确定无误后，保存当前系统设置

再修改 UEFI 启动下的\EFI\microsoft\boot\bcd 文件



参数如上，要注意 UEFI 模式下，系统加载文件为 winload.efi，确认无误后，保存当前系统设置。
如果有多个 PE 系统的映像文件，我们可以再创建几个启动菜单





再保存即可。

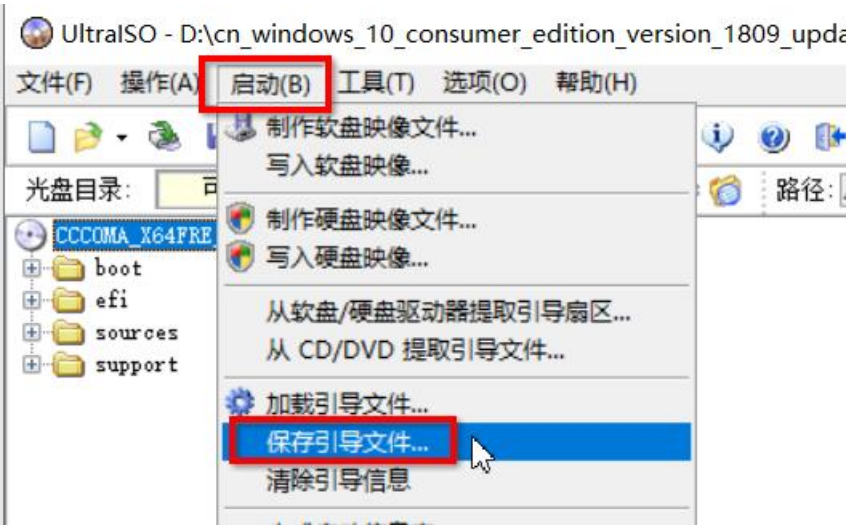
修改完 bcd 文件，就可以拔出 U 盘，插入目标计算机测试一下能否在 2 种启动模式下都正常启动，如果不能肯定是某些文件路径写错了，认真仔细检查一下再改改就行了。

第 22 章、封装 PE 系统镜像的注意点

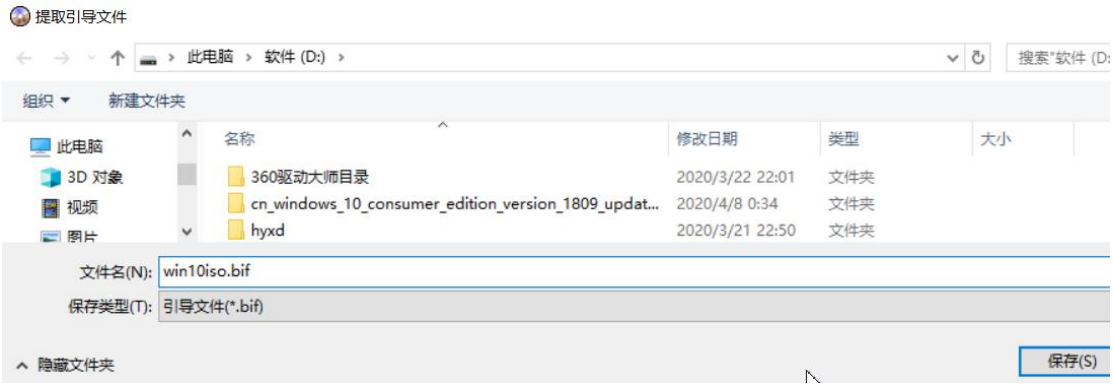
我们在修改添加新的工具软件到 PE 映像里后，要再次封装成 wim 文件，然后再把 wim 文件和引导文件封装成 iso 镜像文件。在这 2 个过程中有一些是要注意的，不然没法正常进入 PE 系统，

①比如在引导界面有时我们选择使用 iso 文件启动，结果不能启动而返回到引导菜单界面，这是因为 ISO 文件没有引导信息，我们需要添加。因为我们使用的 PE 是 windows PE，所以先从 windows 系统安装光盘镜像文件里提取引导信息。使用 UltraISO 工具打开 win10 光盘镜像文件，

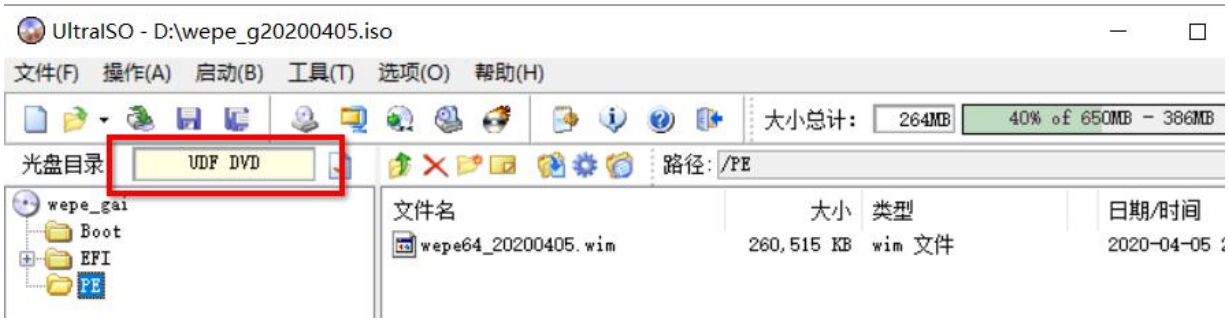
可见它是可启动的，我们要把它的引导文件导出来，



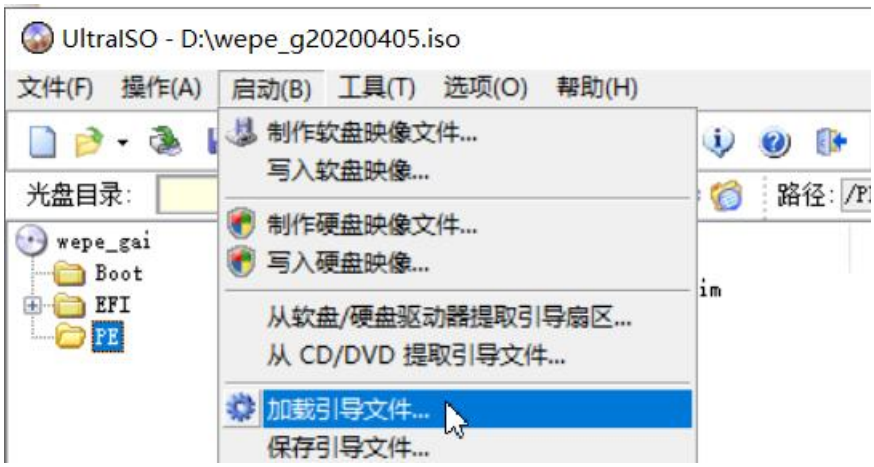
点击“启动”→“保存引导文件”



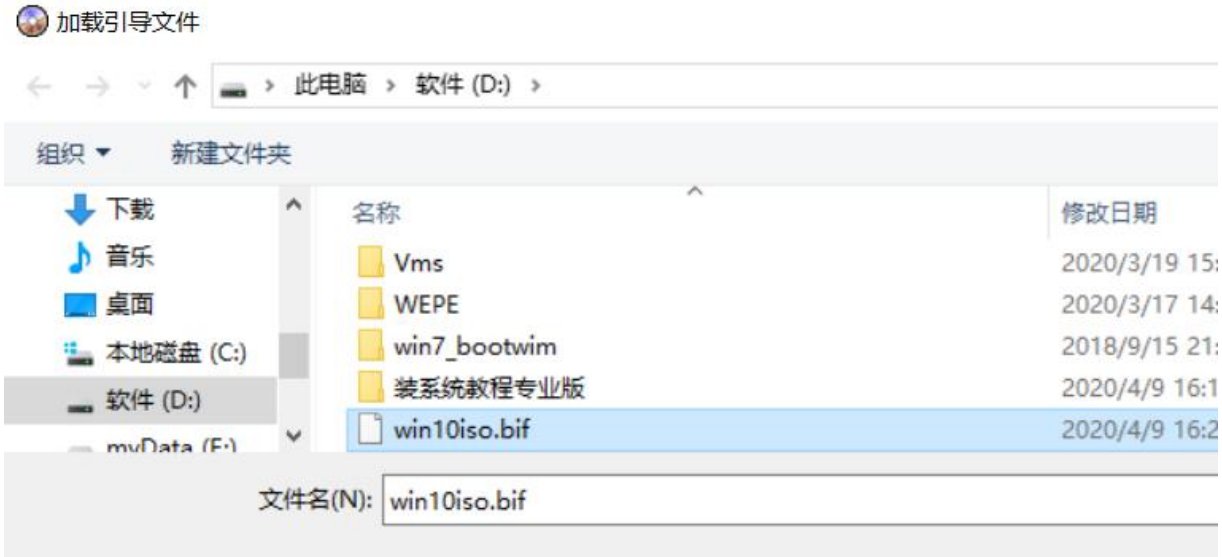
保存到 D 盘，文件名为 win10iso.bif



然后再打开我们的 PE 镜像，如上图，它没有显示“可启动”字样，所以需要我们把刚刚提取的 win10iso.bif 引导文件导入到这个 pe 的光盘镜像文件里，



点击“启动”→“加载引导文件”

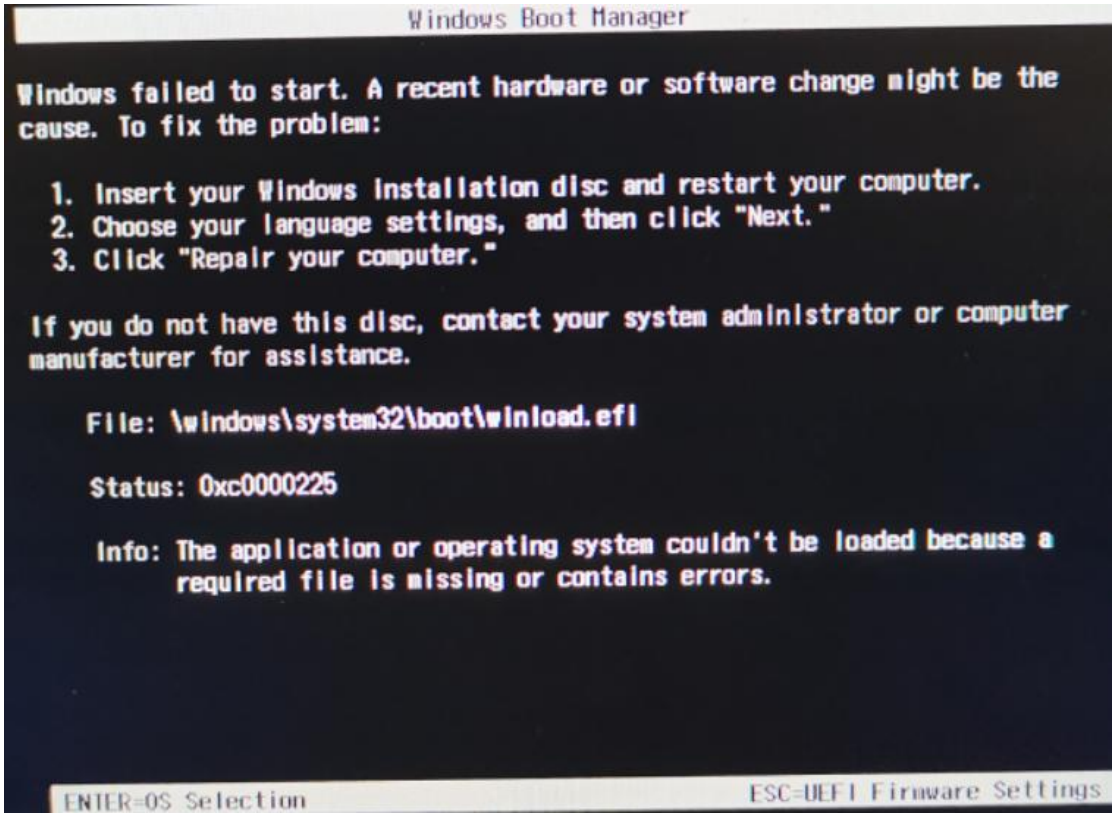


选择我们之前从 win10 光盘镜像文件里提取的引导文件，“打开”

然后就出现了“可启动”字样

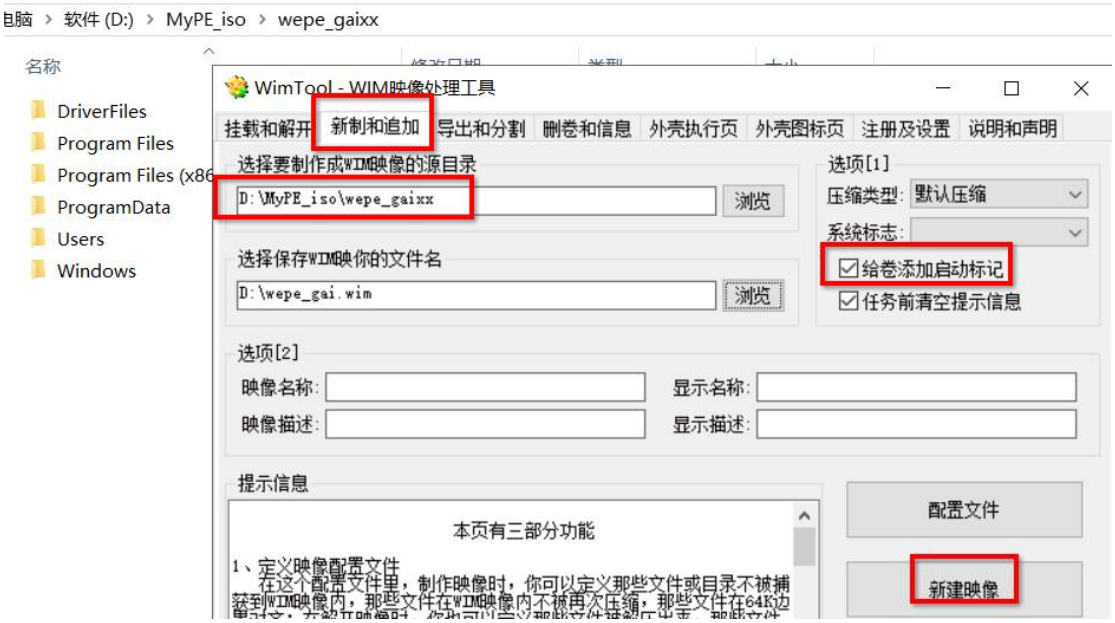
再点击工具栏的保存图标，就可以了。

②可以从 pe 的 iso 镜像文件启动了，结果进不了 PE 系统，出错提示如下：



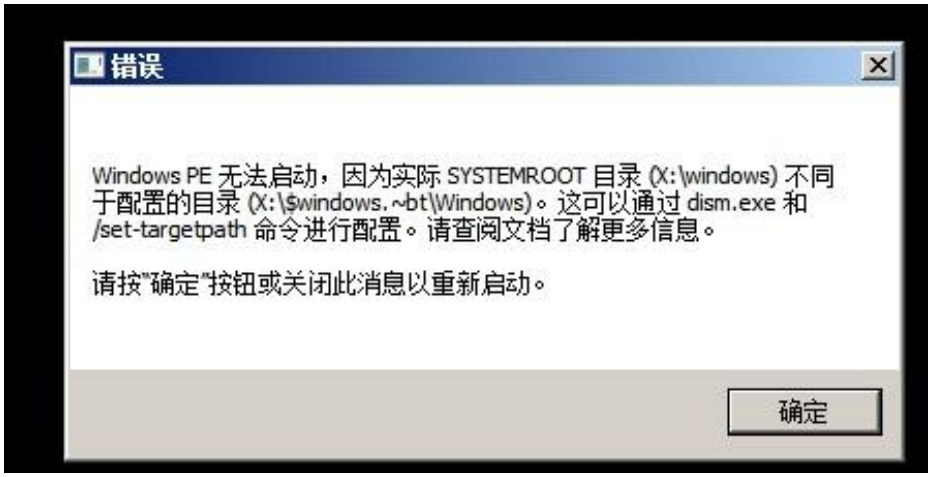
原因是 iso 镜像里的 pexx.wim 映像文件没有启动标记，是不可启动的 wim 文件。解决方法就是给它添加可启动标记，怎么操作呢？可以回炉再造，就是把 wim 文件解压到某目录，再用 WimTool 工具生成 wim 文件，在生成时添加可启动标记，

把 PE 光盘 ISO 镜像里的 wim 文件（本例中为 wepe 修改过的 wim）导出来，再用 WimTool 解压到 D:\MyPE_iso\wepe_gaixx 文件夹里



解压后，不用做修改，直接再进入“新制和追加”项，选中刚刚解压的目录，再记得勾选“给卷添加启动标记”，最后新建映像，生成了新的 wepe_gai.wim 文件，把这个新生成的文件替换原来 iso 光盘镜像里的 wim 文件就行了。

③这下重启，进入 PE 前没有错误提示了，结果就快打开 PE 系统桌面时，突然又出问题了，提示“Windows PE 无法启动，因为……”



根据它的提示可知，是系统设置的 systemroot 目录和我们 bcd 启动菜单里的不一样，可以用 dism.exe 修改，这个程序在 windows 系统里自带，我们以管理员身份运行 cmd 命令行，

输入以下命令：

dism /image:D:\wim 文件解压目录 /set-targetpath:X:\

```
C:\Windows\system32>dism /image:D:\MyPE_iso\wepe_gaixx  /Set-TargetPath:X:\

部署映像服务和管理工具
版本: 10.0.17763.1
```

```
目标路径: X:\

操作成功完成。
```

操作完成后，再重新用 WimTool 工具生成新的 wim 文件，替换 PE 光盘镜像文件里的 wim 文件即可。
因为 wim 映像在启动时，是装入内存中虚拟出来的磁盘中，该盘符一般为 X:盘，所以目标路径为 X:\

第 23 章、Centos 的启动过程

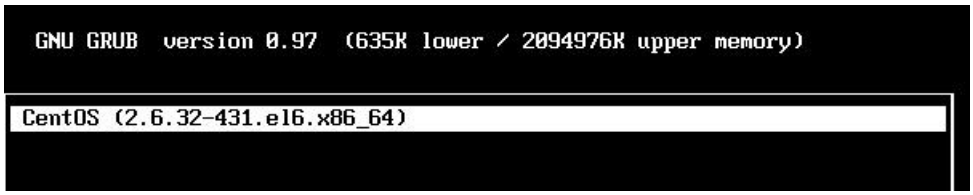
前面好几章都只讲了 windows 的相关知识，主要还是为了方便大家理解，由简入难。今天我们就开始学习 Linux 的相关知识。

Linux 的发行版本实在是太多了，作者也不可能一一讲解了，本章就以 Centos6 和 Centos7 为例讲一下其具体启动过程。

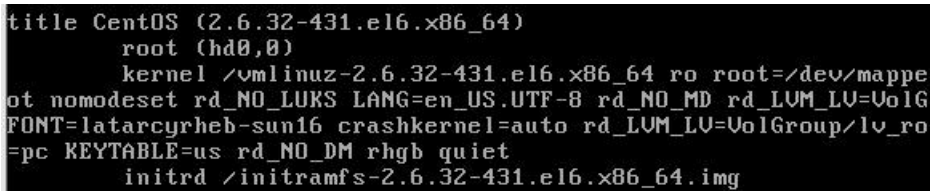
★Centos6 系统启动过程

Bios 启动模式下，BIOS 预启动程序先读取启动磁盘上的 MBR 扇区，确定分区表没问题后，再执行 mbr 扇区里的引导代码，里面的代码是 grub 引导代码。对，Centos6 使用 grub 引导去引导系统。

grub 引导在 bios 启动模式下，分三个阶段（stage），stage_1 的代码放在 MBR 扇区的 446 字节引导区里，主要作用是判断分区表的正确性和加载下一步的引导代码，第二步的引导代码称为 stage_1.5，位于 MBR 扇区之后的几个扇区里（mbr_gap），第二步的引导 stage_1.5 代码不多，大概不到 20KB，主要作用是识别分区里的文件系统，方便加载文件系统里的其他引导文件，比如加载/boot 分区里的 stage_2 引导程序和 grub.conf 配置文件，最终提供给用户几个启动菜单，

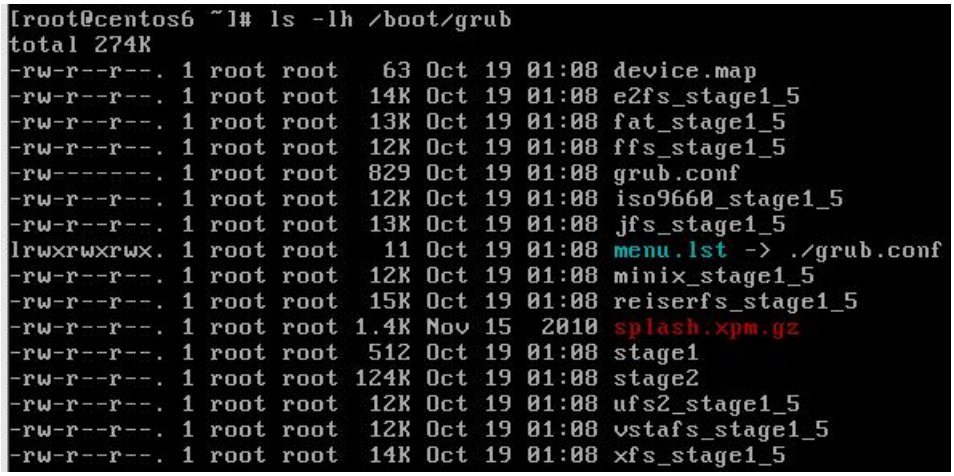


用户选中某菜单后，加载菜单里指定的系统内核和 initramfs 镜像。



最后由 initramfs 的 dracut 系统去找真正的系统的/根目录，再切换根目录到真正的根上，这样就完成了正式系统的启动。

下图可见 stage_2 文件位于/boot 分区的 grub 子目录下，名为 stage2，大小约 124KB



内核文件位于/boot 分区里，名为 vmlinuz-xxx
initramfs 文件和内核文件位于同一位置，名为 initramfs-xxx.img

```
[root@centos6 ~]# ls -lh /boot
total 22M
-rw-r--r--. 1 root root 103K Nov 22  2013 config-2.6.32-431.el6.x86_64
drwxr-xr-x. 3 root root 1.0K Oct 19 01:07 efi
drwxr-xr-x. 2 root root 1.0K Oct 19 01:08 grub
-rw-------. 1 root root  15M Oct 19 01:08 initramfs-2.6.32-431.el6.x86_64.img
drwx------. 2 root root  12K Oct 19 01:07 lost+found
-rw-r--r--. 1 root root 190K Nov 22  2013 symvers-2.6.32-431.el6.x86_64.gz
-rw-r--r--. 1 root root 2.5M Nov 22  2013 System.map-2.6.32-431.el6.x86_64
-rwxr-xr-x. 1 root root 4.0M Nov 22  2013 vmlinuz-2.6.32-431.el6.x86_64
[root@centos6 ~]#
```

Centos6 在 bios 启动模式，一般的分区结构如下：

```
[root@centos6 ~]# lsblk
NAME                        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                           8:0    0   20G  0 disk
├─sda1                        8:1    0  500M  0 part /boot
└─sda2                        8:2    0 19.5G  0 part
  ├─VolGroup-lv_root (dm-0) 253:0    0 17.5G  0 lvm  /
  └─VolGroup-lv_swap (dm-1) 253:1    0    2G  0 lvm  [SWAP]
```

```
[root@centos6 ~]# df -Th
Filesystem                    Type    Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup-lv_root ext4     18G  859M   16G   6% /
tmpfs                         tmpfs  931M     0  931M   0% /dev/shm
/dev/sda1                     ext4   485M   32M  428M   7% /boot
```

一般分 2 个分区，其中一个为标准分区，挂载在/boot 目录下，我们称之为/boot 分区，使用 ext4 文件系统，
而另一分区默认使用 lvm 逻辑卷去管理，默认卷组名 VolGroup 创建 lv_root 和 lv_swap　2 个逻辑卷，分别挂载在/根目录下和作为 swap 分区。

/boot 分区必须为标准分区，即直接在分区上使用 ext4 或 xfs 等文件系统，不能用 lvm 的逻辑分区，为什么呢？因为 centos6 是使用 grub 引导的，grub 的 stage_1 和 stage_1.5 的引导比较小，不能识别 lvm 逻辑卷，而 grub 还要到/boot 分区里去找 stage_2 及内核等文件，所以/boot 分区不能为 lvm 分区。
内核启动后，内核再加载 initramfs 里的某驱动和工具，就能识别 lvm 分区，所以/根分区可以为 Lvm 分区（当然也可以是标准分区）

内核的启动：

内核 vmlinuz 和 initramfs 文件被 grub 引导加载到内存后，grub 把引导权交给内核，内核名为 vmlinuz 时，末尾字母为 z 表示此内核文件是使用 gzip 压缩的，内核文件头部的代码会自动把此内核解压缩，内核解压后就开始做正式的工作了

主要是初始化系统中的各设备并做相关配置，如 cpu，I/O，存储等，配置过程也会加载一些内核里带有的部分设备驱动，调用这些驱动去初始化相关设备。内核启动后会创建一个 rootfs 文件系统。

内核文件本身比较小，带的东西不多，所以把其他具体系统中要用到的工具及驱动放到了 initramfs 文件里，内核把 initramfs 文件解压到 rootfs 里，然后执行 rootfs 里面的/sbin/init 程序，init 程序加载完最后的一些驱动后，会去寻找/根分区，把/根分区挂载到/sysroot 目录下，最后 init 会释放掉未使用的内存，并执行根切换（chroot　/sysroot）
切换到真实的/根（root）上面，同时运行根分区里的/sbin/init 程序，根分区里的/sbin/init 程序运行后，就是系统的 1 号进程，真正的系统就启动了，接下来就是系统的事情了。（比如执行/etc/inittab 及/etc/init.d/*等脚本文件，启动指定的服务）

小结：

启动过程	centos6 的
1	BIOS 预启动程序加载指定磁盘的 MBR 扇区里的引导：grub-stage1
2	stage1 加载位于 MBR 之后的 stage1.5 引导
3	stage1.5 去加载/boot 分区里的 grub 子目录下的 stage2 引导程序
4	stage2 读取同目录下的 menu.lst（即 grub.conf）配置菜单
5	当用户选择指定菜单项后，grub 引导指定的内核及 initramfs，若用户不选择，则启动默认的菜单项
6	内核文件解压后，初始化设备，加载驱动，解压 initramfs 到 rootfs 文件系统里，执行里面的/sbin/init
7	rootfs 里的/sbin/init 程序加载最后的驱动，寻找根分区，挂载到/sysroot 目录下，清理内存，把当前的根目录切换到/sysroot 下
8	系统切换到真正的根目录下后，执行真正根目录里的/sbin/init 程序，这也是系统里 pid 为 1 的程序，它再去完成系统最后的启动工作

UEFI 启动模式下，都差不多，就是 grub 引导的三个部分 stage1，stage1.5 和 stage2 都放在 EFI 分区下的 /EFI/BOOT/bootx64.efi 文件里，它们合并为一个文件了，uefi 预启动程序加载这个 efi 文件（在 x86_64pc 里名字为 bootx64.efi，它是 grub 引导）然后 grub 引导接下来的工作就和 bios 里的一样了。具体的放到其他章节里讲吧。

★Centos7 系统启动过程

centos7 使用 grub2 作为引导，BIOS 启动模式下，过程如下：

BIOS 预启动程序读取启动磁盘上的 MBR 扇区，执行里面的引导代码（grub2 的第一部分代码），mbr 里的代码再加载 mbr 扇区后的第二阶段的代码。第二阶段的代码能识别文件系统，所以它能加载/boot 分区里的某些模块及内核等文件比如加载/boot 分区的 grub2 子目录下的 i386-pc 目录里的其他模块，以及 fonts 文件等，再读取 grub.cfg 配置菜单，最后加载菜单项里指定的内核及 initramfs 文件，

```
[root@centos7 ~]# ls -lh /boot/grub2
total 32K
-rw-r--r--. 1 root root   84 May  5 21:16 device.map
drwxr-xr-x. 2 root root   25 May  5 21:16 fonts
-rw-r--r--. 1 root root 4.2K May  5 21:16 grub.cfg
-rw-r--r--. 1 root root 1.0K May  5 21:16 grubenv
drwxr-xr-x. 2 root root 8.0K May  5 21:16 i386-pc
drwxr-xr-x. 2 root root 4.0K May  5 21:16 locale
```

```
  CentOS Linux (3.10.0-1062.el7.x86_64) 7 (Core)
  CentOS Linux (0-rescue-03fcd77a2d354d17a8941622bd110b9b) 7 (Core)
```

内核及 initramfs 等文件位于/boot 分区下，

```
[root@centos7 ~]# ls -lh /boot
total 101M
-rw-r--r--. 1 root root 150K Aug  7  2019 config-3.10.0-1062.el7.x86_64
drwxr-xr-x. 3 root root   17 May  5 21:14 efi
drwxr-xr-x. 2 root root   27 May  5 21:14 grub
drwx------. 5 root root   97 May  5 21:16 grub2
-rw-------. 1 root root  55M May  5 21:15 initramfs-0-rescue-03fcd77a2d354d17a8941622bd110b9b.img
-rw-------. 1 root root  19M May  5 21:16 initramfs-3.10.0-1062.el7.x86_64.img
-rw-------. 1 root root  11M May  5 21:19 initramfs-3.10.0-1062.el7.x86_64kdump.img
-rw-r--r--. 1 root root 312K Aug  7  2019 symvers-3.10.0-1062.el7.x86_64.gz
-rw-------. 1 root root 3.5M Aug  7  2019 System.map-3.10.0-1062.el7.x86_64
-rwxr-xr-x. 1 root root 6.5M May  5 21:15 vmlinuz-0-rescue-03fcd77a2d354d17a8941622bd110b9b
-rwxr-xr-x. 1 root root 6.5M Aug  7  2019 vmlinuz-3.10.0-1062.el7.x86_64
```

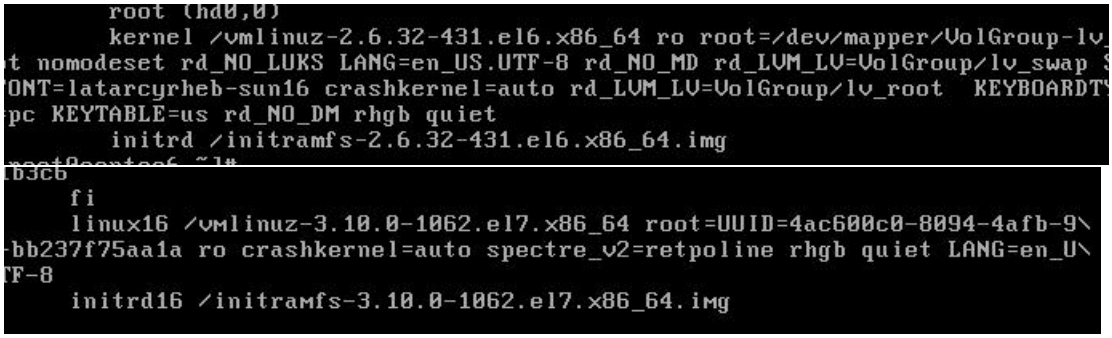
接下来的事情就和 centos6 的差不多了，只是启动的第一个程序不再是/sbin/init，而是/sbin/systemd 程序

小结：

启动过程	centos7 的
1	BIOS 预启动程序加载指定磁盘的 MBR 扇区里的引导：grub2
2	mbr 里的引导加载位于 MBR 之后的第二阶段引导
3	第二阶段引导去加载/boot 分区里的 grub2 子目录下的 i386-pc 子目录里的其他模块文件，grub2 正式加载完毕
4	grub2 读取同/boot/grub2 下的 grub.cfg 配置菜单
5	当用户选择指定菜单项后，grub2 引导指定的内核及 initramfs，若用户不选择，则启动默认的菜单项
6	内核文件解压后，初始化设备，加载驱动，解压 initramfs 到 rootfs 文件系统里，执行里面的/sbin/systemd
7	rootfs 里的/sbin/systemd 程序加载最后的驱动，寻找根分区，挂载到/sysroot 目录下，清理内存，把当前的根目录切换到/sysroot 下
8	系统切换到真正的根目录下后，执行根目录里的/sbin/systemd 程序，这也是系统里 pid 为 1 的程序，它再去完成系统最后的启动工作

第 24 章、传给内核的参数及 initramfs

grub 及 grub2 的启动菜单项如下：



grub 为：

kernel /vmlinuz-xxx 参数 1 参数 2 其他参数

initrd /initramfs-xxx.img

grub2 为：

linux16 /vmlinuz-xxx 参数 1 参数 2 其他参数

initrd16 /initramfs-xxx.img

在 BIOS 启动模式下：

grub 使用 kernel 命令来指定要加载的内核，在内核后可传递其他参数，

grub 使用 initrd 命令来指定要加载的 initramfs 文件

grub2 使用 linux16 命令来指定要加载的内核，在内核后可传递其他参数，

grub2 使用 initrd16 命令来指定要加载的 initramfs 文件

要传递给内核的参数，只和具体的内核版本有关，而跟 grub 的版本或其他引导无关，无论我们用 grub 去引导 centos6 还是 grub2 去引导 centos6，只要内核版本是一样的，其传入的参数就是同一个写法，与 grub 版本无关！

传给 centos6 内核（2.6.32）的参数如下：

root=xxx #表示真正的系统的根分区，可以使用 UUID，可以使用磁盘号，可以使用分区的 label，如 root=/dev/xxx，root=UUID=xxx，root=LABEL=xxx

ro 或 rw #加载真正的根分区时的读写属性，一般为 ro

rd_LVM_LV=xxx #要激活的 lvm 逻辑卷，如 rd_LVM_LV=/dev/mapper/VG_LV

LANG=xxx #设置系统的语言环境

rhgb #以图形界面方式启动系统

quiet #以文本方式启动系统，且禁止输出大多数的 log 信息

crashkernel=xxx #给 kdump 服务用的一个参数，值为 auto 时表示根据系统内存自动 reserve 一些内存给 kernelcrash 用

rd_NO_DM #不检测 raid 设备，DM 表示 Device Mapper Raid

rd_NO_MD #不检测 raid 设备，MD 表示 Multiple Device Raid

rd_NO_LUKS #不检测使用 LUKS 加密的磁盘

selinux=0 #不使用 selinux

nomodeset #告诉内核先不加载显卡而用 BIOS 模式显示，防止黑屏

xdriver=vesa #使用 vesa 接口驱动进入 XWindow 图形化界面窗口

SYSFONT=latarcyrheb-sun16 #设置系统默认的字体
method=nfs:x.x.x.x:/mnt #指定安装介质为 nfs 共享路径

传给 centos7 内核（3.10.0）的参数和上面的 centos6 的差不多，基本一样，就是 rd_LVM_LV 可以写成以下的形式：
rd.lvm.lv=xxx #要激活的 lvm 逻辑卷

在进入正式的 centos 系统后，可以查看/proc/cmdline 文件得知当前系统在启动时传给内核的参数

```
[root@localhost ~]# cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-3.10.0-1062.el7.x86_64 root=UUID=4ac600c0-8094-4afb-94b0-bb237f75aa1a ro
rnel=auto spectre_v2=retpoline rhgb quiet LANG=en_US.UTF-8
[root@localhost ~]#
```

其实 initrd 或 initrd16 命令后的 initramfs 文件也是传递给内核的参数，告诉内核要使用这个 **initramfs 文件**。

initramfs 可以在内核启动时提供一个用户态环境，借助它可以完成一些在内核启动阶段不易完成的工作，比如：

*加载第三方驱动，
*定制化启动过程，
*制作一个极简的 rescue shell 救援系统
*执行某些命令，比如/sbin/init 或 systemd 程序

从 centos6 开始不再使用原来的 mkinitrd 工具，而是使用 **dracut 工具**去生成 initramfs 文件，所以，当我们的 grub 配置菜单项写错了或磁盘分区的 uuid 变了，则在加载内核及 initramfs 后，很可能因为找不到真正的根分区或其他错误而停留在 dracut 的极简救援系统界面：

```
Warning: /dev/disk/by-uuid/4ac60ac0-8094-4afb-94b0-bb237f75aa1a does not exist

Generating "/run/initramfs/rdsosreport.txt"


Entering emergency mode. Exit the shell to continue.
Type "journalctl" to view system logs.
You might want to save "/run/initramfs/rdsosreport.txt" to a USB stick or /boot
after mounting them and attach it to a bug report.


dracut:/#
dracut:/#
dracut:/#
dracut:/# ls /
bin  dracut-state.sh  init  lib64  root  sbin      sys      tmp  var
dev  etc             lib   proc   run   shutdown  sysroot  usr
dracut:/# ls / --color=auto
bin  dracut-state.sh  init  lib64  root  sbin      sys      tmp  var
dev  etc             lib   proc   run   shutdown  sysroot  usr
```

```
dracut:/#
Display all 172 possibilities? (y or n)_
```

这个极简的系统里只有 100 多个命令，且根目录下的所有文件都是由 initramfs-xx.img 解压得来的，

dracut 工具会尝试尽可能少地将正式系统中的必备工具放入 initramfs 文件中，在 centos6 或 7 系统里，执行以下命令可以生成当前系统的 initramfs 文件：

dracut -f /某 dir/initramfs-$(uanme -r).img $(uname -r)

表示生成适配当前运行的内核的 initramfs 文件。当我们更新系统内核后，也会生成新的 initramfs 文件。所以内核的版本和 initramfs 的版本要一致。

```
[root@localhost ~]# ls /boot
config-3.10.0-1062.el7.x86_64
efi
grub
grub2
initramfs-0-rescue-03fcd77a2d354d17a8941622bd110b9b.img
initramfs-3.10.0-1062.el7.x86_64.img
initramfs-3.10.0-1062.el7.x86_64kdump.img
symvers-3.10.0-1062.el7.x86_64.gz
System.map-3.10.0-1062.el7.x86_64
vmlinuz-0-rescue-03fcd77a2d354d17a8941622bd110b9b
vmlinuz-3.10.0-1062.el7.x86_64
[root@localhost ~]#
```

因为 initramfs 文件是根据正式的系统去生成的，所以在我们安装系统时，最后一步就是生成 initramfs 文件。当我们升级系统后（比如从 centos6 升级到 7）如果正式的系统中缺少某些 lib 链接文件，则生成的 initramfs 文件里也会缺少这些链接文件，导致某些功能无法正常启动，比如 lvm 功能，这时，如果系统的根目录分区为 lvm 逻辑卷，而 initramfs 里的 lvm 工具又无法正常启动，则系统的根目录也无法被激活挂载，系统启动失败。

linux 内核对 initramfs-xxx.img 文件的处理流程：

首先内核先生成一个 rootfs 文件系统，rootfs 是内核里的一个特殊的文件系统，基于内存的文件系统，是内核启动的初始根文件系统。

内核再判断 initramfs 文件的格式，若为 cpio 格式的文件，则把 initramfs 的内容解压释放到由内核生成的 rootfs 中，最后执行其中的/sbin/init 或/sbin/systemd 程序。

第 25 章、Centos 系统的安装原理

这得先看一下 centos 的安装光盘镜像文件的结构：

（x86_64 类型的 iso 以 centos6 为例）

在 centos6 的安装光盘里就几个主要的目录：

/EFI 目录是给 UEFI 启动时准备的，里面主要是 grub 启动相关的文件

/images 目录里面有 pxeboot 子目录，里面有内核及 initrd 文件

/isolinux 目录下也有内核及 initramfs 文件，

/Packages 目录里是各 rpm 软件包， /repodata 目录里是关于 Packages 目录的 yum 仓库信息文件。

安装光盘里的 mbr 引导为 isolinux 引导。

当使用 BIOS 启动模式时，这个 isolinux 引导会去读取/isolinux 目录下的配置文件 isolinux.cfg ，并在用户选择 install centos 时加载内核 vmlinuz 及 initrd.img 文件。

如果是 UEFI 启动模式，会去寻找光盘里的/EFI 目录，然后找/EFI/BOOT/Bootx64.efi 引导，这个 bootx64.efi 引导是 grub 引导。加载引导后，grub 再读取配置文件 grub.conf，当用户选择要安装系统时，grub 会加载光盘里的/images/pxeboot/目录下的内核及 initrd 文件。

（/isolinux 目录下的内核与/images/pxeboot 目录下的内核其实是一样的，是同一个文件，initrd.img 文件也是一样的。只是目录不一样而已）

安装光盘里的 initrd 文件和正式系统中的 initramfs 文件原理差不多，只是功能不一样，或者说它们执行的程序不一样，正式的系统中的 initramfs 文件被加载后，会去找真正的系统的根分区，并挂载到/sysroot 下，最后切换根。

而安装光盘里的 initrd 文件在加载后，会执行 安装向导 程序。并在用户的操作下完成正式系统的安装。

安装向导会找到安装源，默认为当前的光盘，然后根据安装的版本（如最小化版本或桌面版）去安装指定数量的 rpm 软件包，软件包位于光盘的/Packages 目录下。安装完成后，再切换根到正式的系统下，生成 initramfs 文件。

aarch64 的 iso 以 rocky linux 8 为例：

我们发现 arm 版本的 iso 安装光盘里竟然没有/isolinux 目录，那引导放在哪里呢？

原来是 arm 的光盘不再使用 isolinux 引导了，直接使用 grub2 引导（BIOS 启动或 UEFI 启动时都是 grub2）

引导配置文件为 /EFI/BOOT/grub.cfg

第 26 章、Syslinux 引导

在进行 centos 系统的安装 之前，我们有必要先了解一下 Syslinux 引导。

Syslinux 是一个项目，官网为 syslinux.org，该项目包含以下启动引导程序：

引导名称	应用场景
Syslinux	安装于 MS-DOS 的 FAT 文件系统分区里，
Isolinux	用于光盘介质时的引导
Pxelinux	用于从网络启动
Extlinux	安装于 Ext 文件系统分区里
Memdisk	装载镜像文件到内存里再引导里面的系统

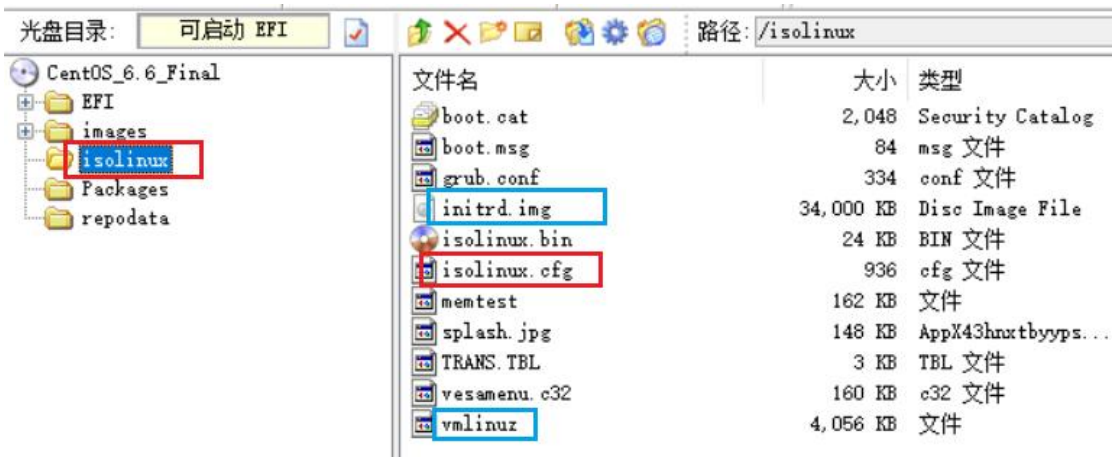
也就是说以上 5 种引导的区别是应用的场景不同，不同的文件系统或介质 使用不同的引导，它们都是属于 Syslinux 这个项目，由 Syslinux 团队开发的。

syslinux 引导可以引导 linux 系统，也可使用 chainloader 引导其他的 bootloader

配置文件为对应的引导名称加.cfg ，如 syslinux 引导的配置文件为 syslinux.cfg

isolinux 引导的配置文件为 isolinux.cfg。配置文件的内容和命令都差不多。

我们还是先看一下 centos6 的安装光盘文件结构吧：



安装光盘的/isolinux 目录为 isolinux 引导的默认工作目录，具体的文件及用途：

```
isolinux.cfg                #isolinux 引导的配置文件
vesamenu.c32                #窗口模块
splash.jpg                  #启动菜单背景图片
vmlinuz                     #centos6 的内核文件
initrd.img                  #initrd 文件，里面有安装向导程序
#其他的文件一般用不上
```

而光盘里的/EFI/BOOT 目录下的 bootx64.efi 引导为 grub 引导，配置文件为 bootx64.efi

BIOS 启动模式下使用的是 isolinux 引导，UEFI 启动模式下才使用 grub 引导。

isolinux.cfg 配置文件讲解：

```
default vesamenu.c32
#prompt 1
timeout 600

display boot.msg

menu background splash.jpg
menu title Welcome to CentOS 6.6!
menu color border 0 #ffffffff #00000000
menu color sel 7 #ffffffff #ff000000
menu color title 0 #ffffffff #00000000
menu color tabmsg 0 #ffffffff #00000000
menu color unsel 0 #ffffffff #00000000
menu color hotsel 0 #ff000000 #ffffffff
menu color hotkey 7 #ffffffff #ff000000
menu color scrollbar 0 #ffffffff #00000000

label linux
  menu label ^Install or upgrade an existing system
  menu default
  kernel vmlinuz
  append initrd=initrd.img
label vesa
  menu label Install system with ^basic video driver
  kernel vmlinuz
  append initrd=initrd.img xdriver=vesa nomodeset
label rescue
  menu label ^Rescue installed system
  kernel vmlinuz
  append initrd=initrd.img rescue
```

```
default              #命令指定使用的容器模块为 vesamenu.c32
timeout  600         #指定菜单停留时间，时间单位为 10 分之 1 秒，600 则表示为 60 秒
menu  xxx           #指定菜单的样式，如无特殊要求，可忽略

label  xxx              #定义一个菜单项，指定其 label
  menu  label  xxx        #指定该菜单项的显示名称
  menu  default           #指定该菜单项为默认选择的菜单
  kernel  vmlinuz         #指定要加载的 linux 内核，相对路径则表示和该配置文件相同的目录下
  append  xxx         #给该 Linux 内核传入的参数，包含了 initrd 文件
```

我们在使用刻录工具把 centos6 安装光盘 iso 文件刻入 U 盘时，会有什么变化呢？

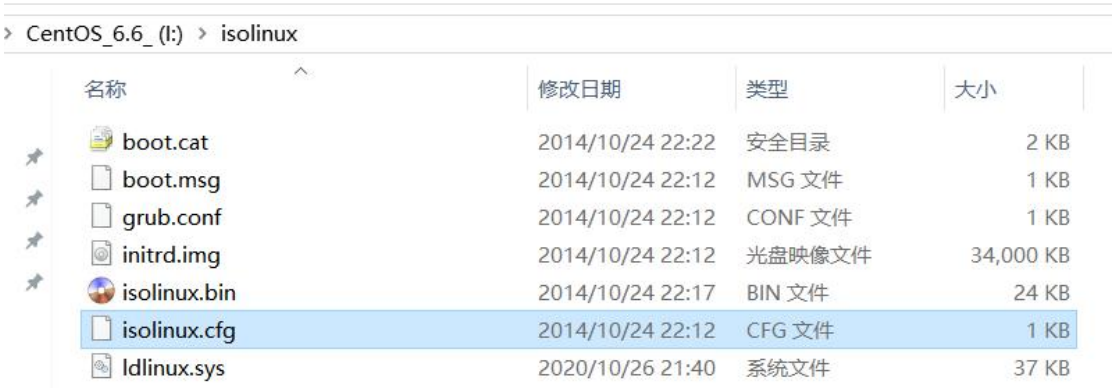
我们试试，使用 UltraIso 工具把 centos6.iso 刻录到 U 盘里，根据前面章节的知识我们知道，直接解压.iso 文件时，是只有里面的文件，而不包含引导的，如果刻入到 U 盘里，会带有引导，那么用 Ultraiso 写入 U 盘时，会是什么引导呢？

写入硬盘映像

通过刻录时的输出信息，我们大概可以猜到写入到 U 盘后，U 盘上使用的引导为 Syslinux 第 4 版，用 Bootice 工具验证一下

上图可见，U 盘的 MBR 主引导为 Ultraiso USB-HDD+的引导，然后此 mbr 引导再去引导位于分区上的 syslinux 4.07 的引导。

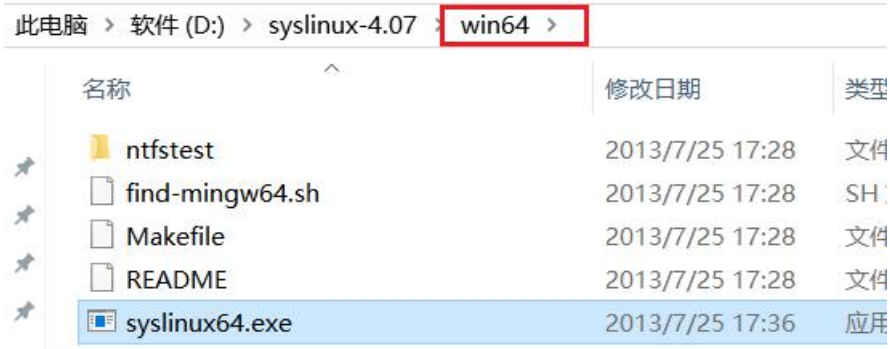
也就是说，直接使用光盘安装时，使用的是 isolinux 引导，刻录到 U 盘时，由 Ultraiso 工具把它转为了 syslinux 引导，然后在 U 盘的/isolinux 目录下多了一个 syslinux.cfg 配置文件，而不是光盘里的 isolinux.cfg



我们当然也可以不使用 Ultraiso 工具去刻录了，直接使用 syslinux 引导，把 syslinux 引导安装到 U 盘的 mbr 里或者分区里，再把 centos6 安装光盘文件解压到 u 盘上就行了。

怎么**手动安装 syslinux 引导**到 u 盘分区的 PBR 呢？
先把 centos6.iso 镜像文件解压到目标 U 盘里，本例中为 i 盘
下载 syslinux 工具 https://mirrors.edge.kernel.org/pub/linux/utils/boot/syslinux/
windows 版的下载.zip 的包，比如 **4.07 的版本**

然后解压到 windows 上的 D 盘下：



在解压目录里的 win64 或 win32 里有 syslinux64.exe 程序
以管理员身份运行 cmd 命令行，进入解压目录下的 win64 目录，执行以下命令

```
C:\Windows\system32>D:

D:\>cd syslinux-4.07\win64\

D:\syslinux-4.07\win64>syslinux64.exe -ma -d /isolinux  I:
```

> syslinux64.exe -fma -d /isolinux I: #表示把引导写入 I:盘，i 盘为目标 U 盘的盘符，配置文件的目录用-d 指出，本例为/isolinux 目录，

如果是 **6.x 版本**的 syslinux，则使用以下命令：

解压目录的\bios\win64 子目录下：

> syslinux64.exe --mbr --active --directory /isolinux/ --install I:

```
D:\syslinux-6.03\bios\win64>syslinux64.exe --mbr --active --directory /isolinux/ --install I:
```

然后要安装系统时，插入 U 盘，选择从目标 U 盘的分区启动就行了。不过，这个不保证在每台机器上都能正常引导。（一定要选择从目标 U 盘的分区启动，因为 syslinux 只能写入分区的 PBR，而不能写入 MBR，所以前面用 Ultraiso 刻录时，它是用了 Ultraiso 的 usb-hdd+的 MBR，而只把 syslinux 写入分区的 PBR）

最后的问题：

当我们使用使用 Ultraios 刻录镜像.iso 文件到 U 盘里后，该 U 盘能在 bios 模式下正常安装系统，而在 UEFI 启动时无法安装，无法进入到安装界面，这是为什么？

因为使用 Ultraios 工具刻录时，它只转换了 isolinux 引导为 syslinux，而没有修改/EFI/Boot/目录下的引导配置文件，导致这里出了问题，Ultraios 在刻录之前已经把目标 U 盘格式化为 FAT32 文件系统，而根据前面章节的知识，**FAT32 分区的卷标最多只有 11 个字符**

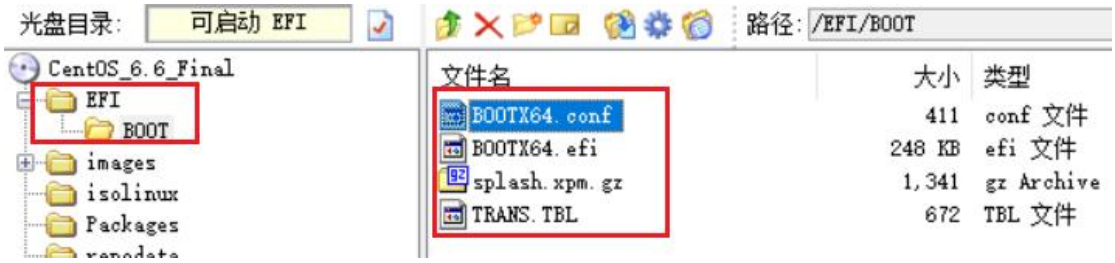
047	11	Volume label 卷标	Atxt	Centos 7

而如果原来的 grub 配置文件里如果使用的卷标名长度大于 11 字符时，就会有缺失，导致引导找不到目标 U 盘，无法正常进入安装界面。

第 27 章、初识 grub 引导

GRUB 全称 GRand Unified Bootloader，是由 GNU 组织相关人员开发的一个功能强大的引导程序，它支持多种文件系统，能引导多种操作系统。

centos6 使用的引导正是 grub，在 Centos6 的安装光盘里，grub 引导只在 UEFI 启动模式下才用到，装完系统后，正式的 Centos6 系统就只使用 grub 引导，而无论是哪种启动模式。



如上图，当使用 UEFI 启动时，预启动程序就去安装介质里找/EFI/BOOT/bootx64.efi 文件，这个文件在这里就是 grub 引导，它的配置文件为同目录下的 bootx64.conf

grub 的配置先不讲太多，主要配置如下：

```
#debug --graphics
default=0
splashimage=/EFI/BOOT/splash.xpm.gz
timeout 5
hiddenmenu
title CentOS 6.6
    kernel /images/pxeboot/vmlinuz
    initrd /images/pxeboot/initrd.img
title Install system with basic video driver
    kernel /images/pxeboot/vmlinuz xdriver=vesa nomodeset askmethod
    initrd /images/pxeboot/initrd.img
title rescue
    kernel /images/pxeboot/vmlinuz rescue askmethod
    initrd /images/pxeboot/initrd.img
```

```
default=0                  #表示默认选择第一个菜单项，从 0 开始编号
splashimage= xxx           #表示菜单使用的背景图片，图片格式为.xpm，用 gz 压缩的
timeout   5                #菜单停留的时间，单位为秒
hiddenmenu                 #隐藏菜单，只有当用户按下按键时才显示
title   xxx                #菜单项名称
  kernel   xxx             #要加载的内核文件，可带要传给内核的参数
  initrd   xxx             #要加载的 initrd 文件，功能和 initramfs 类似
```

这里给内核传递的参数好像和第 24 章讲的不一样啊，对，不一样，这里的内核及 initrd 文件主要功能是用来安装操作系统的，而不是正常的启动正式系统。所以参数不太一样。

第 28 章、使用 U 盘安装 Centos6

首先到网上下载 Centos6 的安装镜像.iso 文件，可以用以下地址：

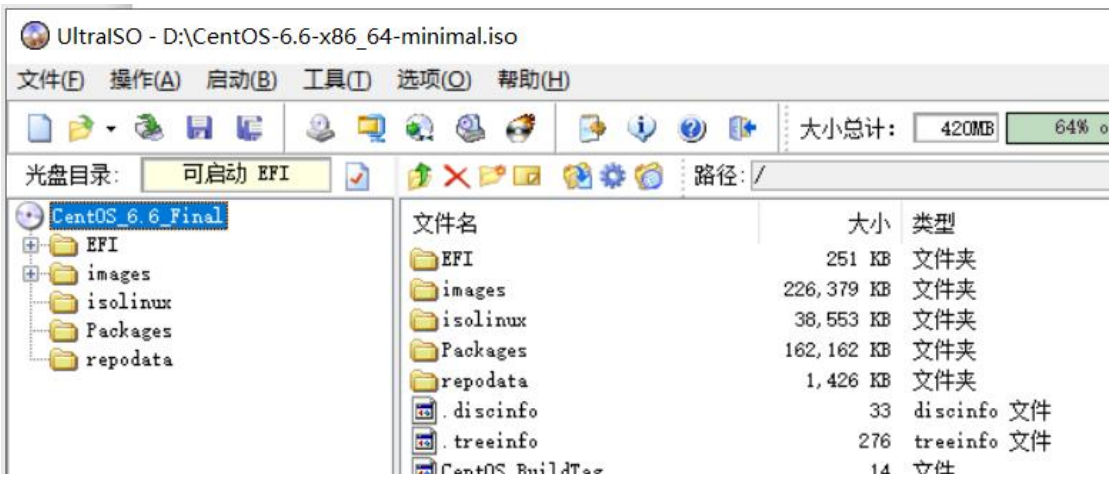
http://mirrors.163.com/centos/6.10/isos/x86_64/

http://mirrors.aliyun.com/centos/6/isos/x86_64/

http://vault.centos.org/6.9/isos/x86_64/

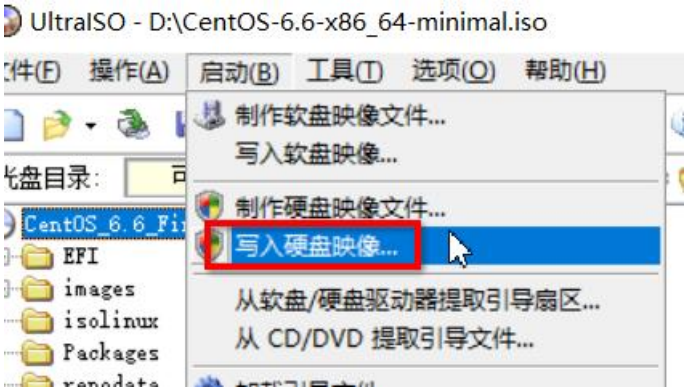
①使用 Ultraiso 工具刻录

使用 Ultraiso 工具打开 centos6 的.iso 文件



然后插入目标 U 盘，U 盘里的数据要先备份一下，

点击菜单栏的“启动”→“写入硬盘映像”



选择目标 U 盘，写入方式为“USB-HDD+”，点击“写入”

弹出提示，点击“是”

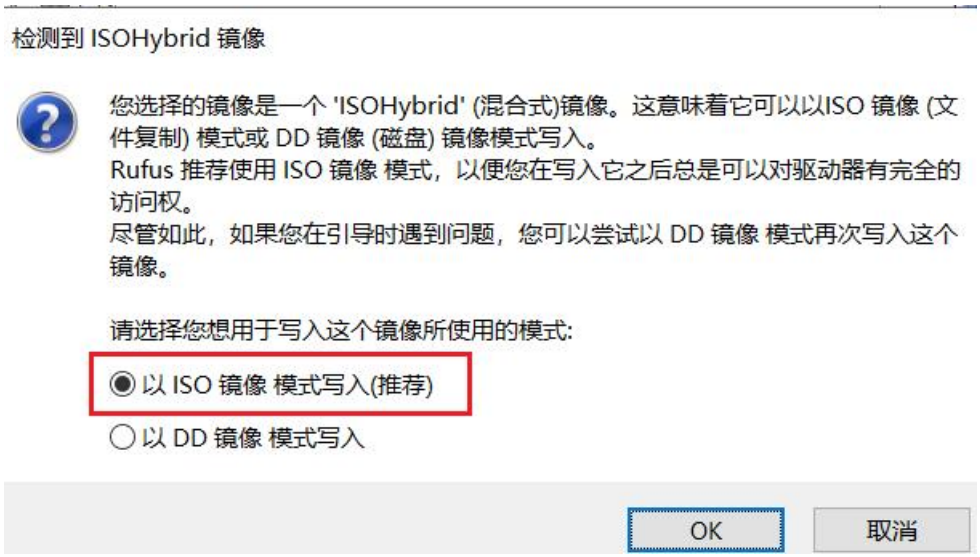
最后等待刻录完成就行了。

②使用 Rufus 工具刻录到 U 盘

官网： https://rufus.ie/en/

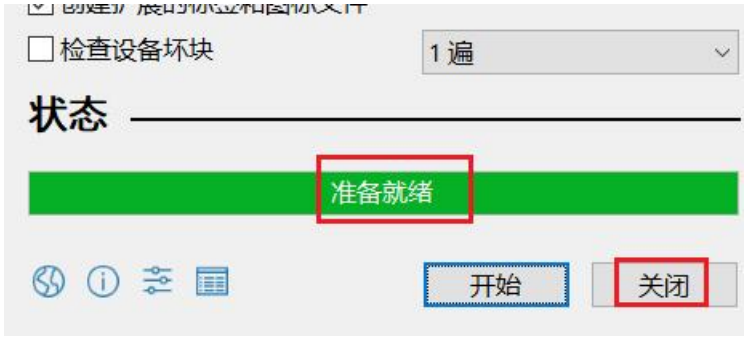
设备为目标 U 盘，引导类型选择为镜像文件，“选择”目标镜像 centos6.iso 文件

其他的就用默认设置，点击下方的“开始”



“以 iso 镜像模式写入”

“确定”

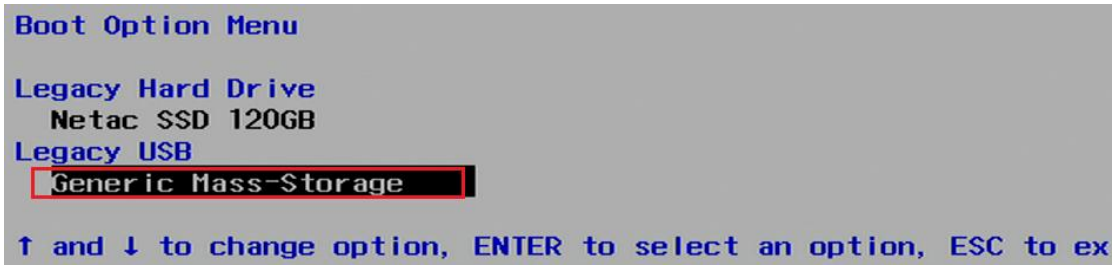


等刻录完成，状态为“准备就绪”，就可以关闭了。

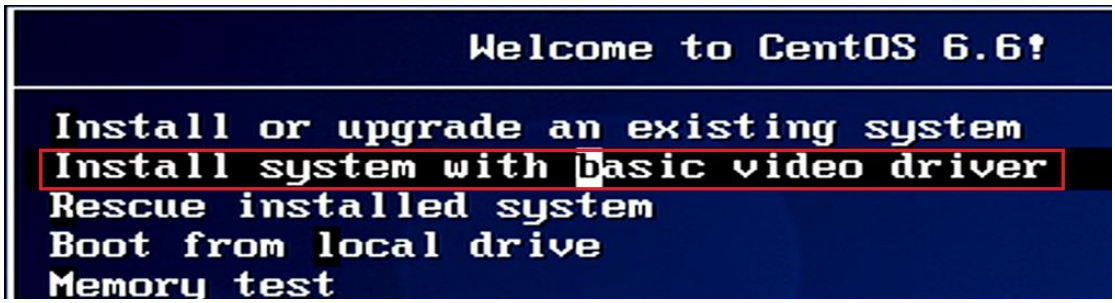
以上 2 种工具刻录随便选择一种，然后拔出 U 盘，插入目标计算机（要装 centos6 系统的计算机）

★centos6 不带有 XHCI 的驱动，较新机型无法通过 U 盘安装

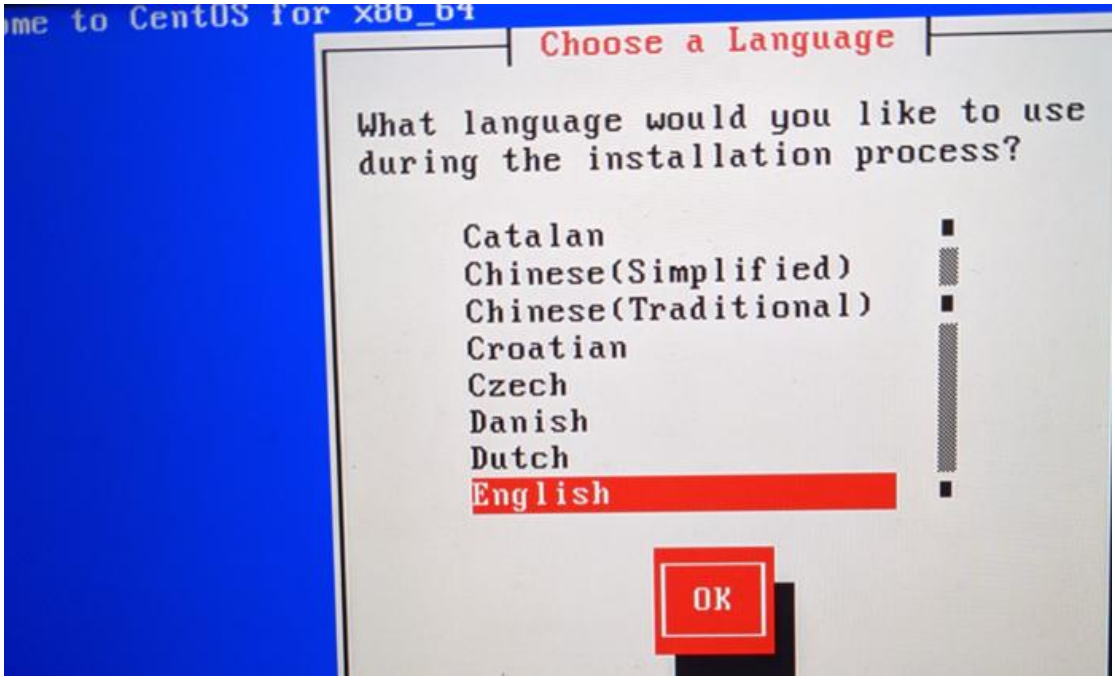
★BIOS 启动模式下



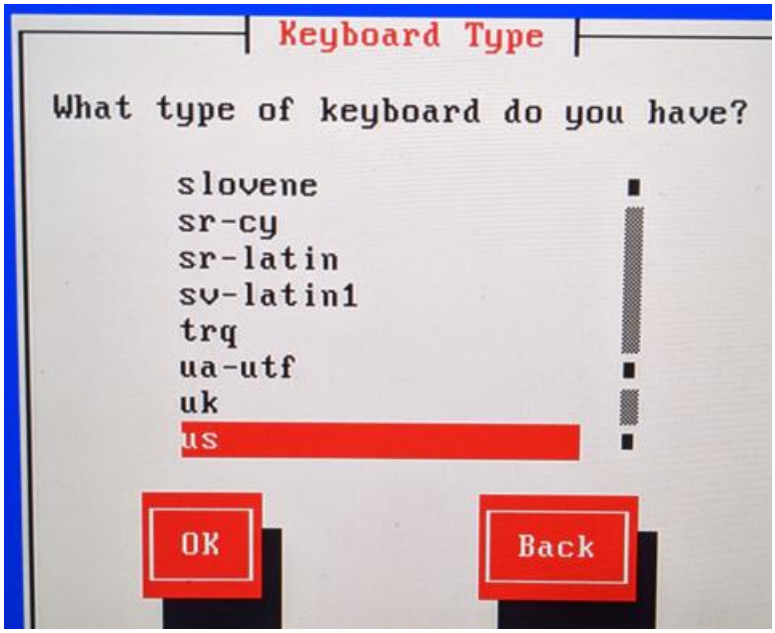
选择从目标 U 盘启动



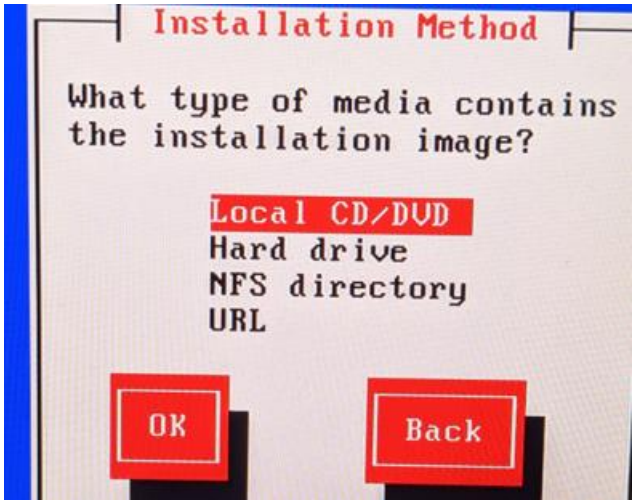
选择第二行，安装 centos6 系统，回车



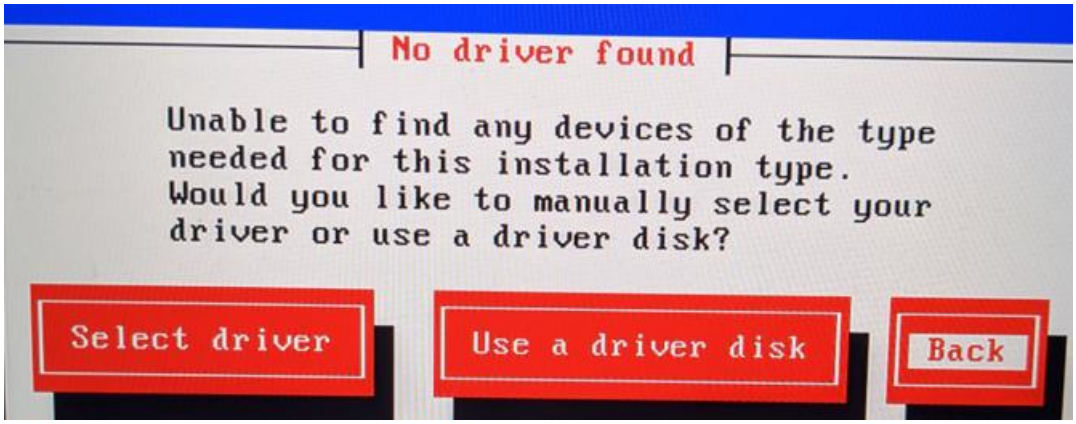
语言就用默认的 English



键盘就用默认的 us 键盘

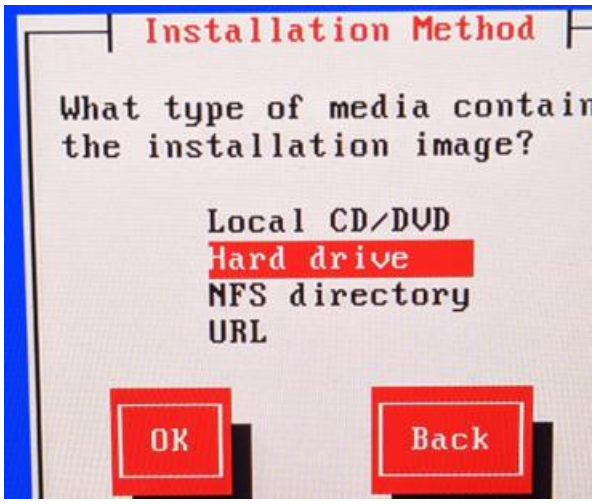


上图，问我们安装介质是哪个？如果是用 CD/DVD 光盘安装时，就选择 Local CD/DVD，我们先试试这个。

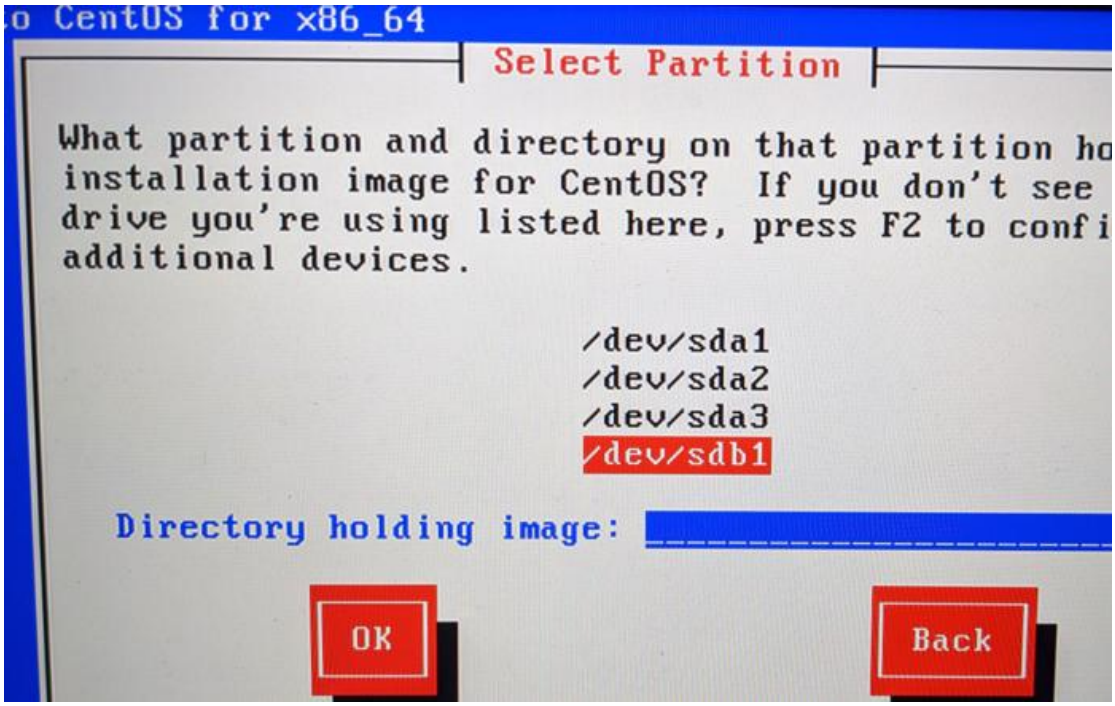


结果它说找不到安装的设备，说明安装介质不是 CD/DVD 了，那是什么？

当然是 U 盘了，我们刻录时是以 usb-hdd 的模式刻录的，所以这里的 U 盘被识别成了硬盘了。先 Back 退回上一步



选择 Hard drive，ok 回车

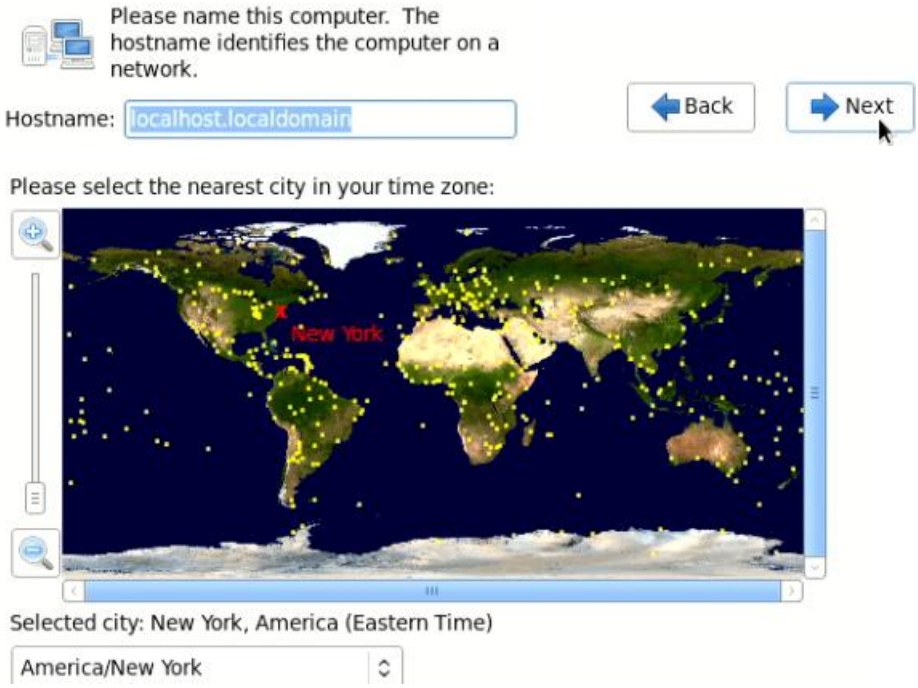


难题又来了，有 2 个设备，一个/dev/sda 一个/dev/sdb，哪个才是 U 盘呢？

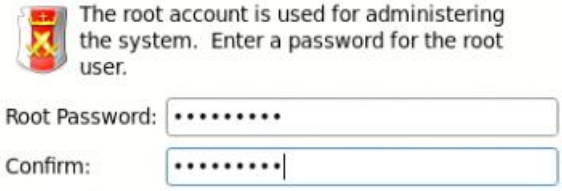
一般目标计算机里面就有一个硬盘，所以 U 盘一般为 sdb，而且上图中 sda 有 3 个分区，而 sdb 只有一个分区，因为我们在刻录时，U 盘就被格式化了，默认只有一个分区，所以我们判定/dev/sdb1 为 U 盘的分区，选中后，回车

果然可以了，上图就是 centos6 的安装向导，点击下一步

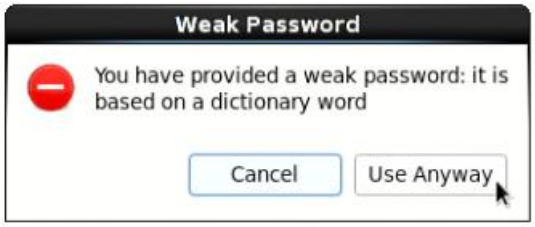
选择 Basic Storage Devices 设备类型，点击下一步



计算机名和时间时区等按实际情况填写

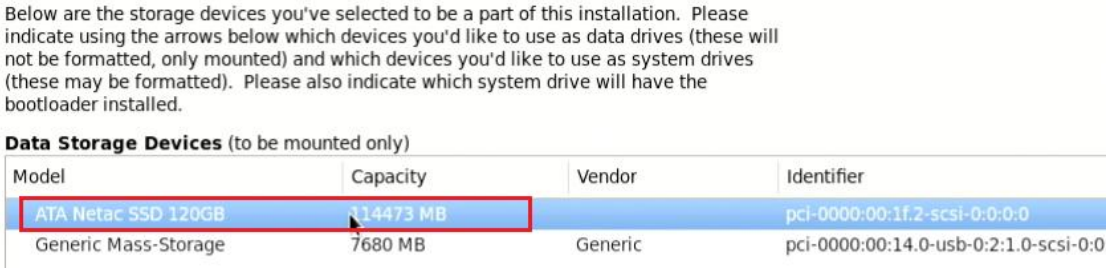


设置 root 用户的密码



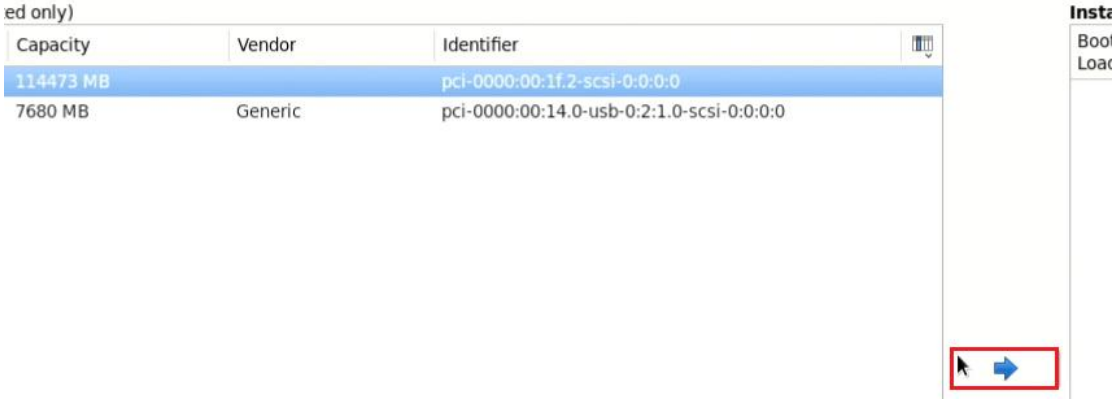
如果提示 Weak Password 密码强度不够，可以重新设置密码，也可以 Use Anyway

如果想完全重装系统，不考虑保留原计算机磁盘上的数据，则选择“Use All Space”使用整个磁盘的空间，点击下一步

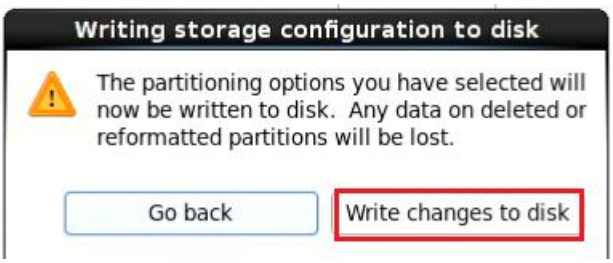


然后出现了上面这一步，因为此时计算机识别到 2 个磁盘，一个是原计算机上的硬盘（110GB 的），另一个是我们的 U 盘（8GB）

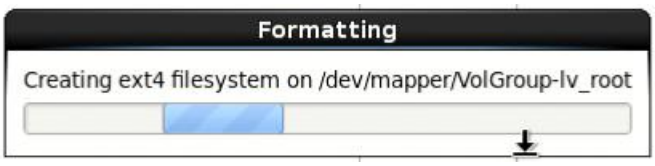
点击选中计算机的磁盘，再点击界面中央的右键，把计算机硬盘移到右边去，



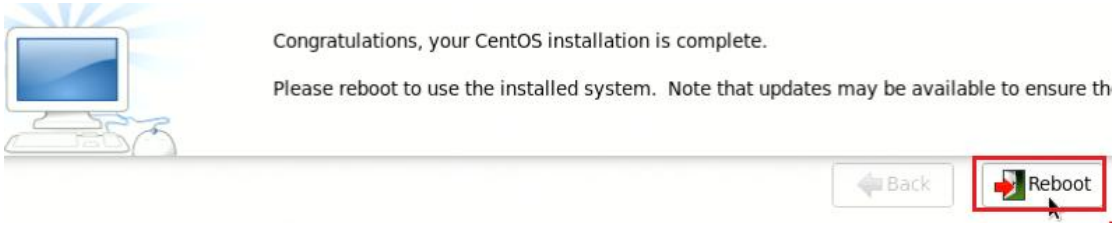
然后再勾选这个硬盘，让它左边的白圈中间出现黑点，再点击右下角的“下一步”



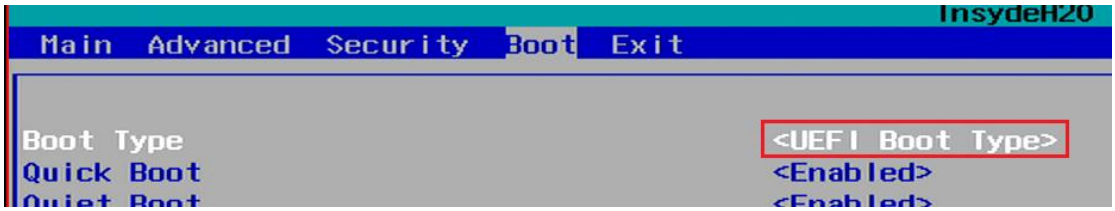
写入配置到磁盘里，使用的是默认的分区模式和大小



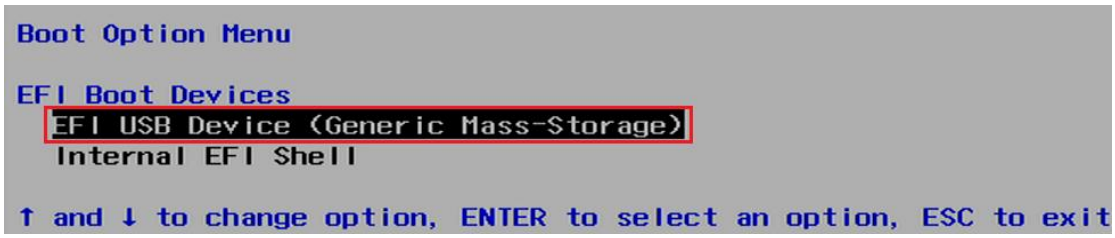
然后就开始安装系统到目标计算机磁盘里了。
等待安装完成，重启就行了。



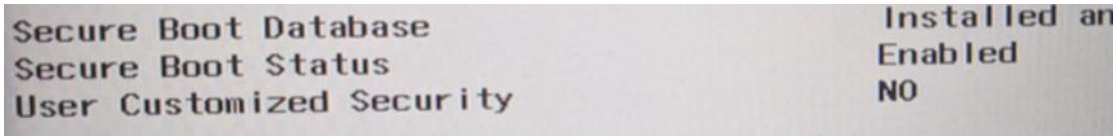
★UEFI 启动模式下



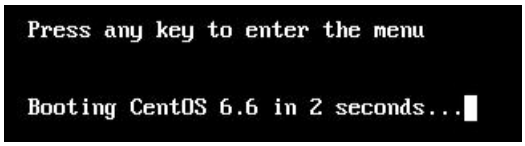
把目标计算机设置为 UEFI 启动模式，插入 U 盘，开机，选择从 U 盘启动



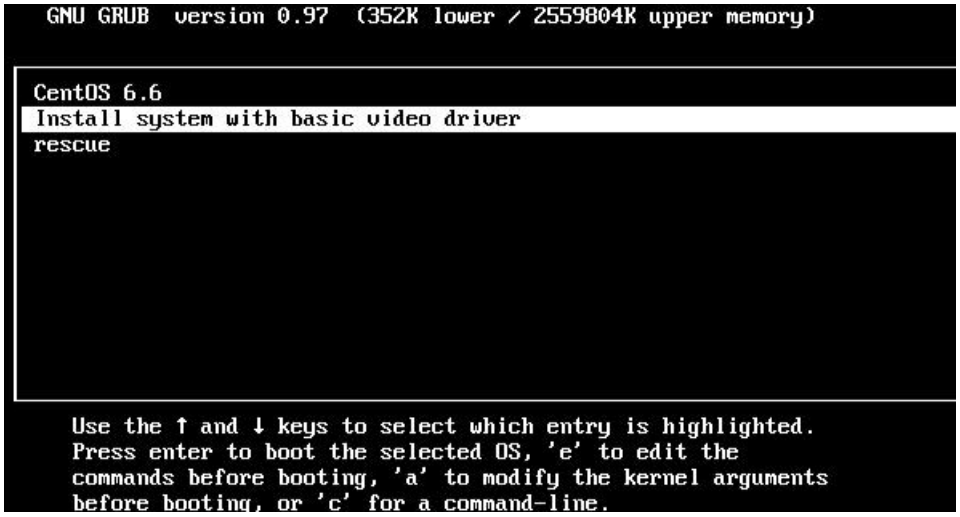
结果指示没法从 U 盘启动，因为什么当前的安装策略？哦，应该是开了 Secure Boot 安全启动，把它关了



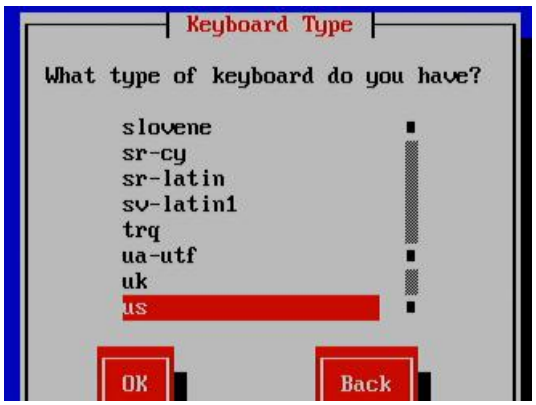
再重启，从 U 盘启动，就可以了

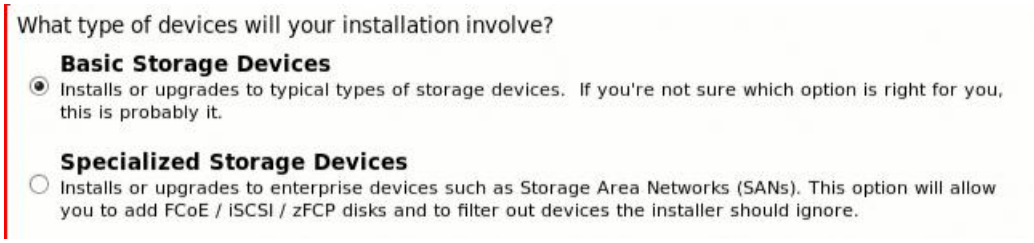


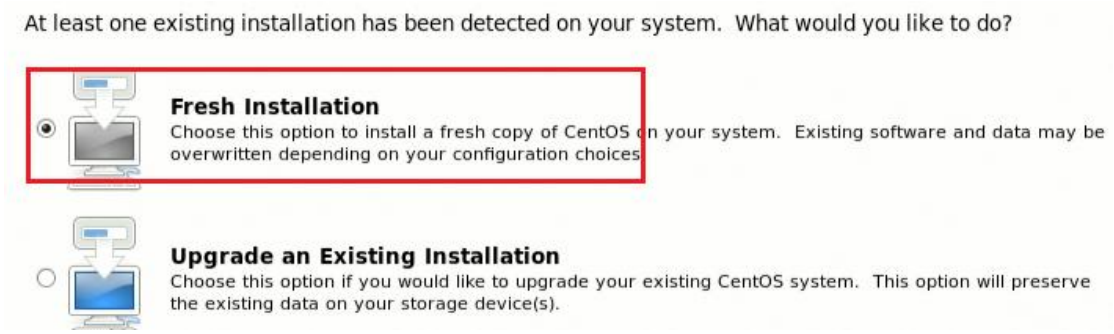
出现上图界面时，得按下空格键，不按也行，默认一般就是安装



选择安装系统，接下来就和前面的从 bios 启动时一样了，有少量的不同

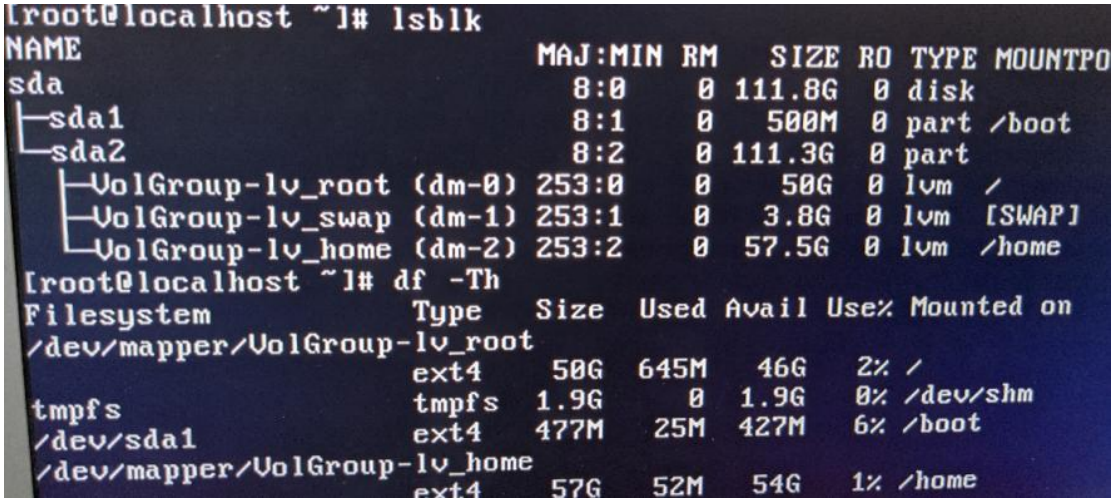






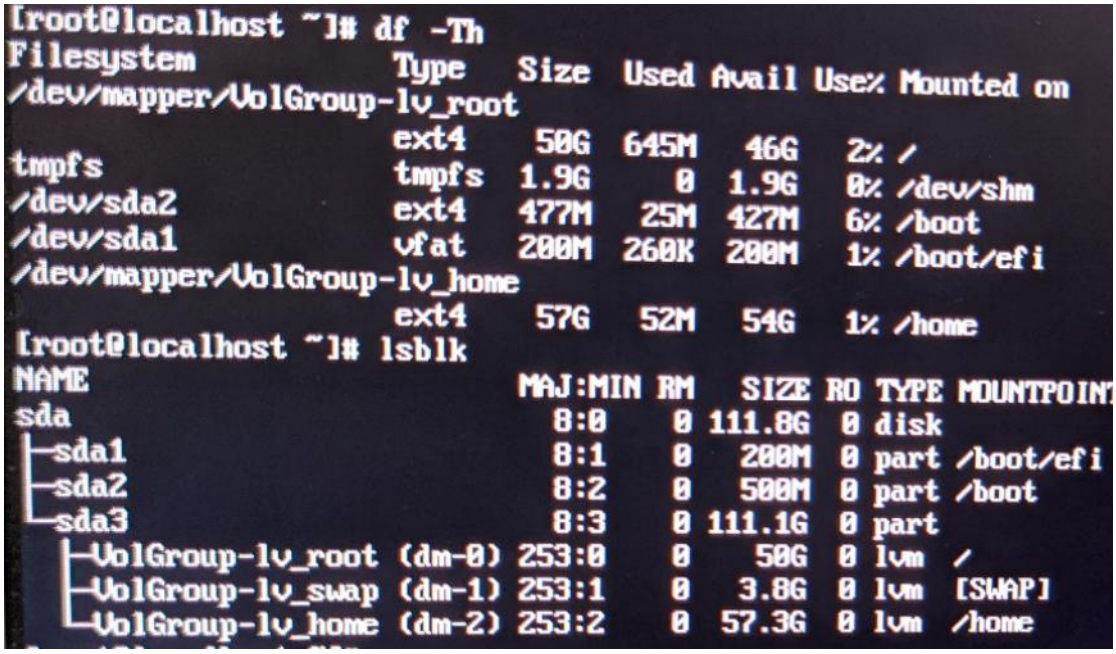
★2 种启动模式下安装系统后的差异

BIOS 启动模式下：

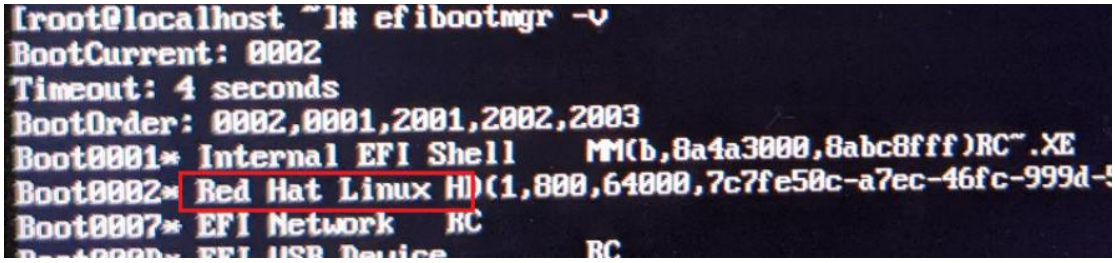


系统默认只划分 2 个主分区，
sda1 挂载到了/boot 目录下，分区大小 500MB
sda2 做成了 lvm 逻辑卷组，创建了 3 个逻辑卷：lv_root，lv_swap 和 lv_home
grub 引导是写入到了 MBR 扇区和 mbr 之后的扇区中，启动配置文件路径为：
/boot/grub/grub.conf

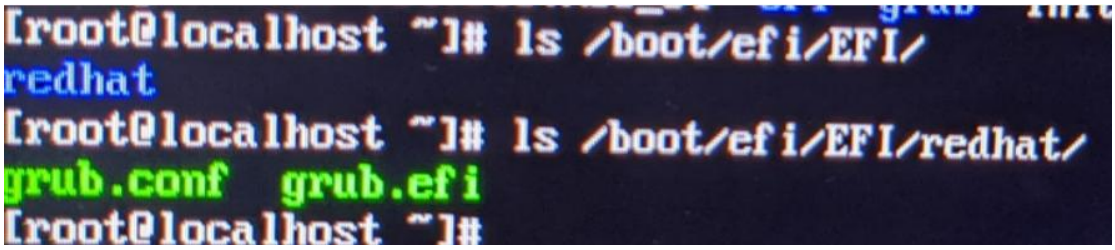
UEFI 启动模式下：



uefi 模式下系统默认划分了 3 个分区，
sda1 挂载到/boot/efi 目录，该分区有 200MB
sda2 挂载到/boot 目录，分区大小 500MB
sda3 也做成了 lvm 逻辑卷组，也是三个逻辑卷

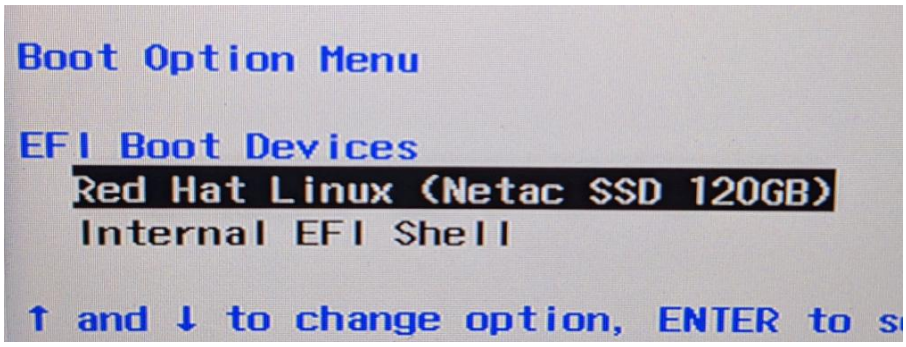


而且在 uefi 启动模式下，centos6 系统默认创建了一个 uefi 启动项，名称竟然还是 Red Hat Linux 的，可能是 centos 忘了改了，（^_^）



grub 引导是直接做为一个文件放到了 EFI 分区里，配置文件路径为：

/boot/efi/EFI/redhat/grub.conf



我们开启目标计算机时，选择的启动菜单名称是 Red Hat Linux

第 29 章、初识 grub2

Centos7 的安装光盘在 BIOS 模式下使用的是 isolinux 引导，刻录到 U 盘后，转为了 syslinux 引导。当使用 UEFI 模式启动时，默认使用的是/EFI/BOOT/下的 Bootx64.efi 引导，这个就是 grub2 引导，默认配置文件为/EFI/BOOT/grub.cfg

配置文件讲解：

```
set default="1"

function load_video {
  insmod efi_gop
  insmod efi_uga
  insmod video_bochs
  insmod video_cirrus
  insmod all_video
}

load_video
set gfxpayload=keep
insmod gzio
insmod part_gpt
insmod ext2

set timeout=60
### END /etc/grub.d/00_header ###

search --no-floppy --set=root -l 'CentOS 7 x86_64'

### BEGIN /etc/grub.d/10_linux ###
menuentry 'Install CentOS 7' --class fedora --class gnu-linux --class gnu --class os {
        linuxefi /images/pxeboot/vmlinuz inst.stage2=hd:LABEL=CentOS\x207\x20x86_64 quiet
        initrdefi /images/pxeboot/initrd.img
}
menuentry 'Test this media & install CentOS 7' --class fedora --class gnu-linux --class gnu --class os {
        linuxefi /images/pxeboot/vmlinuz inst.stage2=hd:LABEL=CentOS\x207\x20x86_64 rd.live.check quiet
        initrdefi /images/pxeboot/initrd.img
```

set default="1" #表示默认选择第 1 个菜单项，从 1 开始编号

function load_video { ... } #定义一个函数 load_video

load_video #使用前面定义的函数

set gfxpayload=keep #设置 gfxpayload 样式

insmod xxx #加载 grub2 的模块

#如果安装介质为 gpt 分区时，要加载 part_gpt 模块

#如果安装源文件所处文件系统为 ext2 则要加载 ext2 模块

set timeout=60 #设置菜单停留时间，单位秒

search --no-floppy --set=root -l 'Centos 7 x86_64'

-l 表示根据 label 搜索，搜索到 label 为'Centos 7 x86_64'的分区就设置为当前的 root，从这开始，下文的相对路径的文件就以此 root 为根目录

menuentry 'xxx' #声明一个菜单项，之后的定义用{}花括号括起来，菜单项声明后面的--class 表示菜单风格

{

linuxefi /xxx/vmlinuz 参数 xxx #加载 linux 内核及传参数给内核

initrdefi /xxx/initrd.img #加载 initrd.img 文件

}
#在文件路径前面如果不加上(hd0,1)之类的磁盘及分区信息，则表示使用上文 root 设置的分区

★传给内核的参数：

inst.stage2=hd:LABEL=CentOS\x207\x20x86_64　　#表示告诉内核，接下来要安装系统或其他操作的镜像源文件所在位置，使用 hd:LABEL=表示根据分区的 label 去找，grub2 的 label 不能含空格，所以当分区的 label 有空格时要用转义符号 \x20 表示，

所以 CentOS\x207\x20x86_64 就表示 CentOS 7 x86_64

quiet　　#表示以文本方式启动系统，且禁止输出大多数的 log 信息

rd.live.check　　#表示进入 live OS 系统并进行介质检查

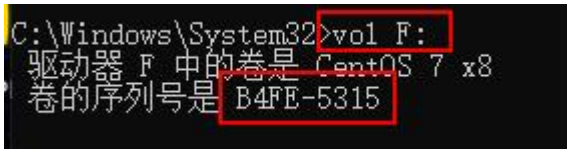
rescue　　#表示进入救援系统

注意：

inst.stage2=hd:LABEL=xxx　　#默认是使用 label 去找安装源文件所在分区

默认配置里写的是 Centos 7 x86_64 也就是安装光盘的 Label 名称，如果使用刻录工具将安装光盘.iso 文件刻录到 U 盘后，可能 Label 就变了或者有残缺，因为 U 盘上的分区文件系统默认是 FAT 的，而 FAT 文件系统允许的 Label 卷标最多只能有 **11 个字符**，而 Centos 7 x86_64 有 15 个字符，在 U 盘的 FAT 分区里就只显示出 Centos 7 x8 这几个字符，所以 grub2 会找不到目标分区，无法进入安装界面。

解决方法有：



windows 下 cmd 命令：　vol　F:　　#查看目标分区的卷标 UUID

再修改配置文件/EFI/BOOT/grub.cfg 里的内核参数

inst.stage2=hd:UUID=B4FE-5315　　#使用 UUID 去查找目标分区

```
search --no-floppy --set=root -l 'CentOS 7 x86_64'

### BEGIN /etc/grub.d/10_linux ###
menuentry 'Install CentOS 7' --class fedora --class gnu-linux --class gnu --class os {
	linuxefi /images/pxeboot/vmlinuz inst.stage2=hd:UUID=B4FE-5315 quiet
	initrdefi /images/pxeboot/initrd.img
}
menuentry 'Test this media & install CentOS 7' --class fedora --class gnu-linux --class
	linuxefi /images/pxeboot/vmlinuz inst.stage2=hd:UUID=B4FE-5315 rd.live.check quie
```

第 30 章、使用 U 盘安装 Centos7

首先下载 Centos7 的安装光盘镜像文件，下载地址：
http://mirrors.aliyun.com/centos/7/isos/x86_64/
http://mirrors.163.com/centos/7.8.2003/isos/x86_64/

然后使用刻录工具刻录到 U 盘里，比如用 Ultraiso

点击 Ultraiso 菜单栏的“启动”→“写入硬盘映像”

选择目标 U 盘，（U 盘里的数据要先备份一下），写入方式为 USB-HDD+
点击“写入”

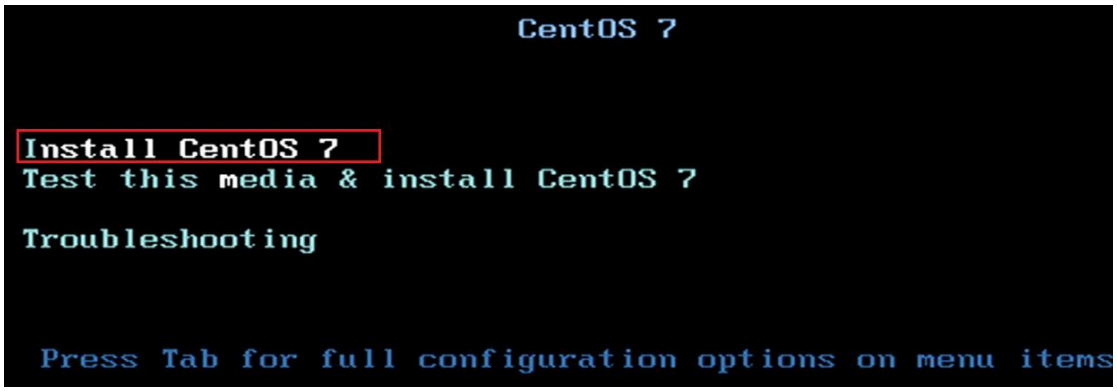
提示

等刻录完成，就可以退出 Ultraiso 了

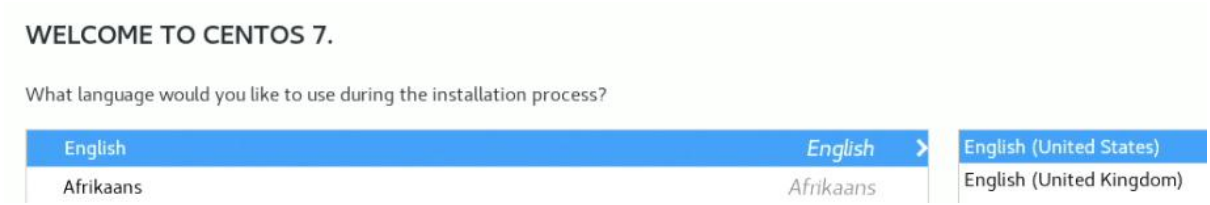
拔出 U 盘，插入目标计算机（要安装 centos7 系统的计算机）

★BIOS 启动模式下

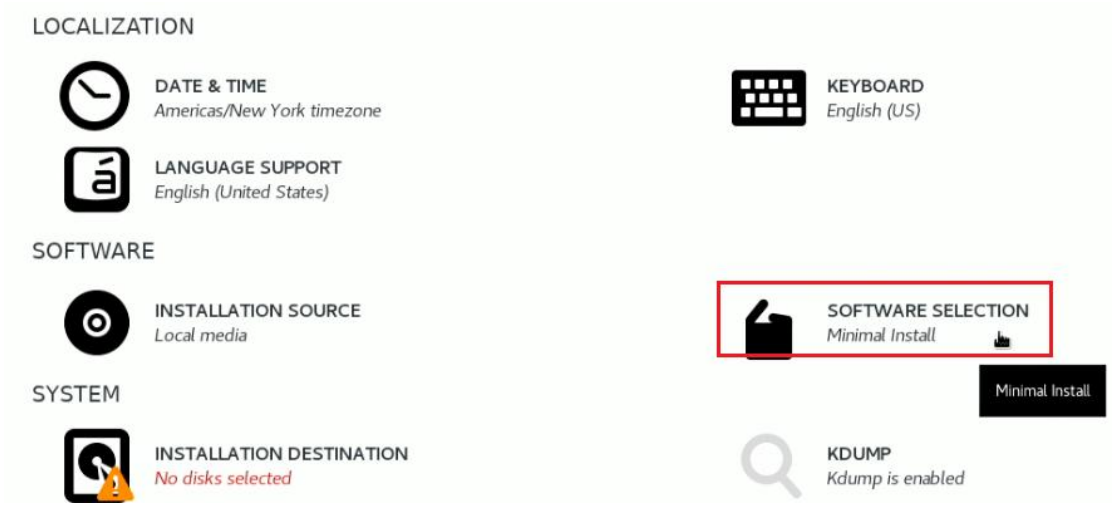
开机时按提示按下选择启动项的按键，比如 F11，选择从目标 U 盘启动（本例中 U 盘为 Generic Mass-Storage 1.11）



按上键头选中“Install　Centos 7”安装 centos7，回车

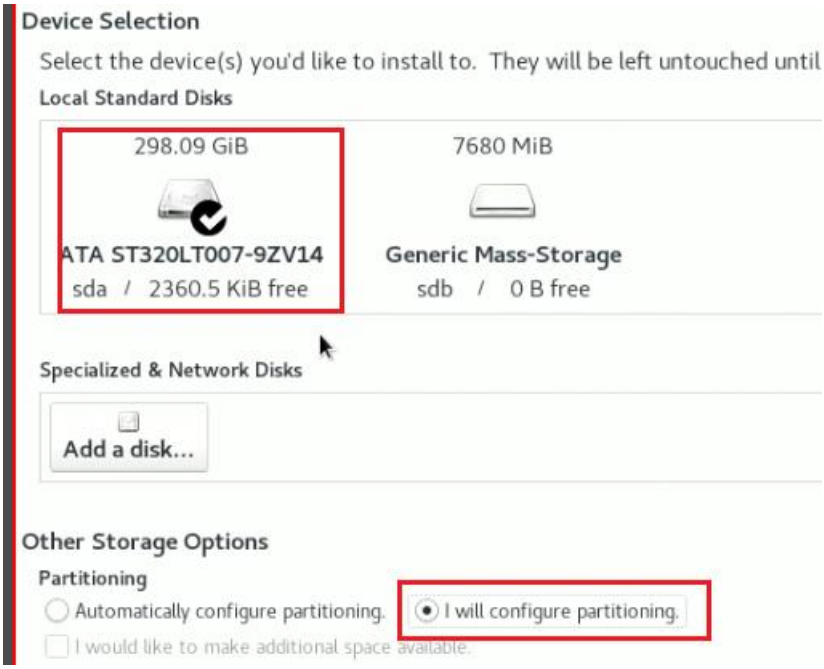


一般都能正常进入安装向导，如上图，首先要选择目标语言，然后点击下一步

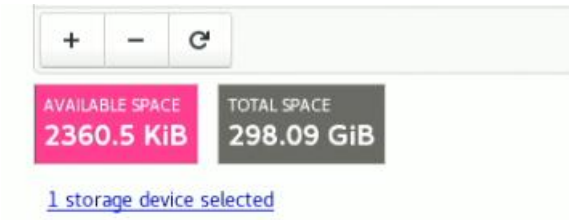


在“Software Selection”里选择要安装的版本，一般使用最小化安装版本

再点击“Installation Destination”安装目标

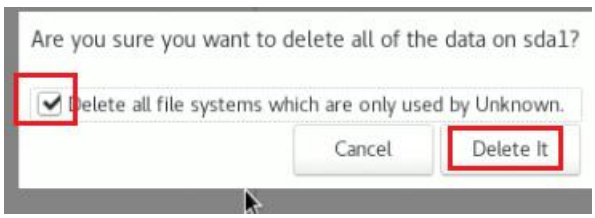


选中目标计算机的硬盘，再点击下方的“I will configure partitioning”进行分区配置，点击左上角的 Done

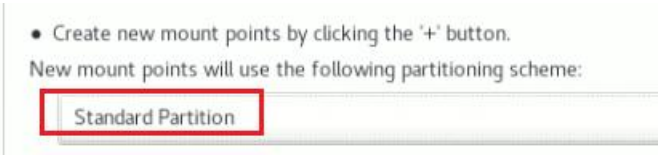


如果目标磁盘剩余空间不够，则一定是之前装有系统或格式化过，可以先把旧的分区删除了，前提是确定原来的数据已经备份了。

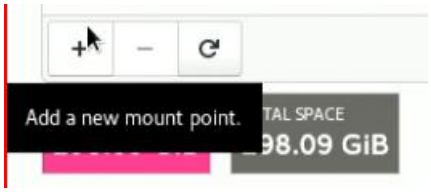
展开原来的系统的分区项，选中/boot 分区，再点击左下角的减号-进行删除



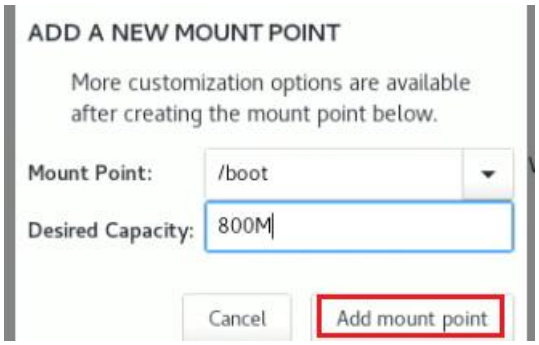
然后会弹出是否删除所有文件系统的对话框，直接勾选，点击 Delete it
前提是旧的分区里的数据已备份，如果不弹出这个对话框，则一个分区一个分区的删除



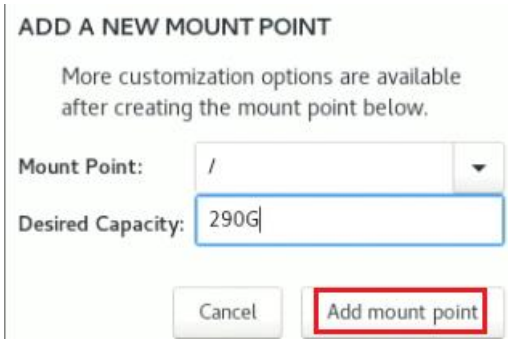
然后 New mount points will use the following partitioning scheme 下选择使用标准分区 Standard Partition



再点击左下角的加号+ ，新建一个分区，先创建/boot 分区



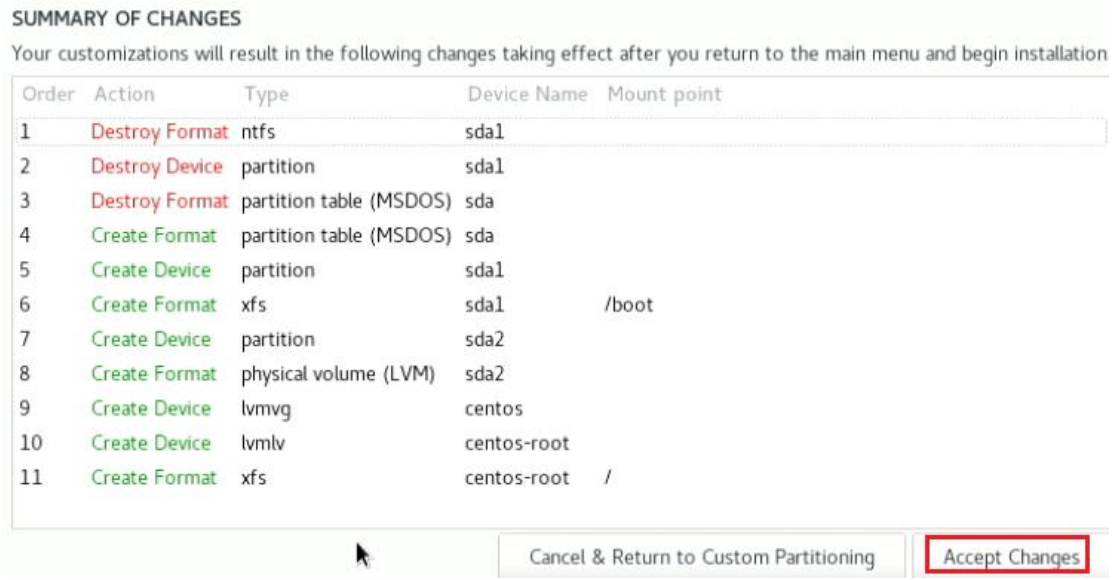
如果考虑以后要更新内核的话，可以设置 500M 以上大小，本例中使用 800M
然后点击 Add mount point，
再点击左下角的加号+ ，创建/根分区



如果不再创建其他分区，则把磁盘剩余的所有空间都划分给/根分区
分区的大小可以用 M,G 等单位，

因为之前使用的是标准分区，所以/根分区这里也是标准分区，不过实体机的话一般考虑使用 lvm 分区，方便以后增加磁盘，于是把根分区的 Standard Partition 类型改为 lvm

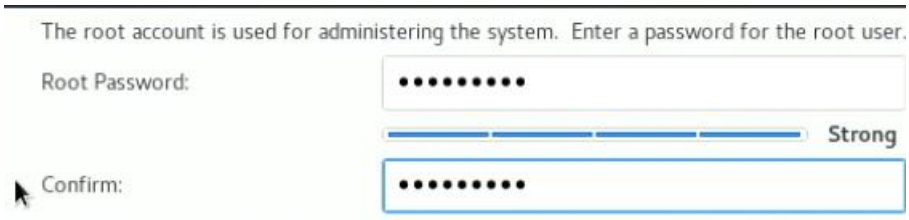
最后点击左上角的 Done，点击 2 次



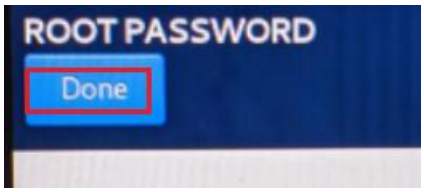
确认清单里点击右边的 Accept Changes 确认

再在安装向导里点击右下角的“Begin Installation”开始安装

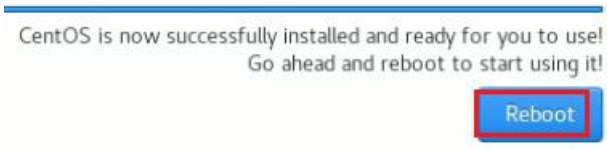
在安装过程中，我们也可以先创建 root 用户的密码，点击“Root Password”



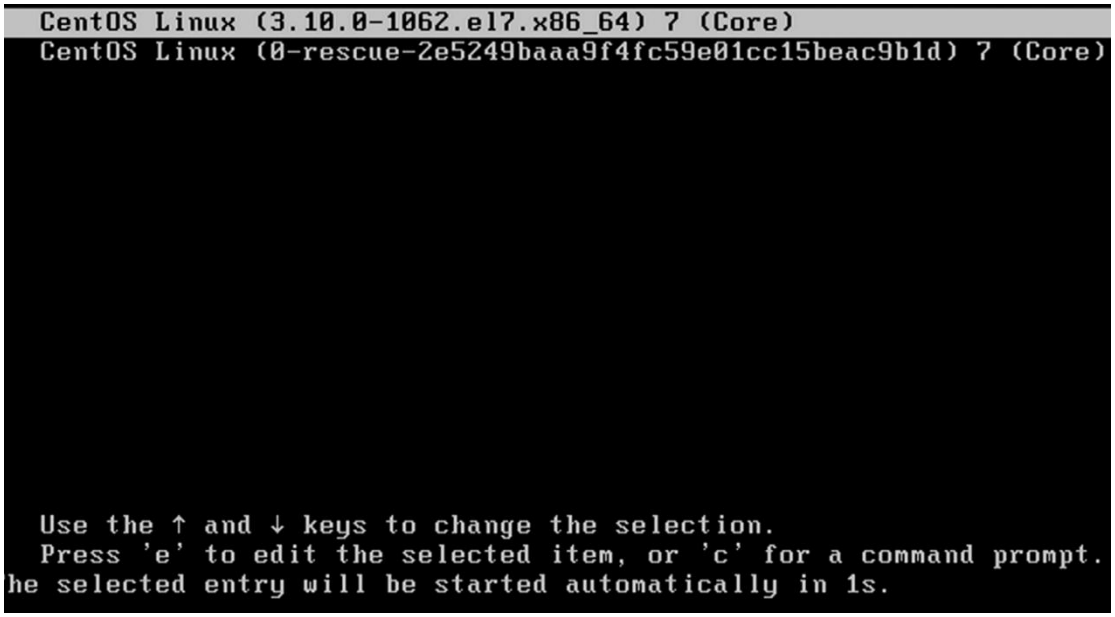
输入 root 密码



最后点击左上角的 Done ，如果密码强度不够，可能要点击 2 次

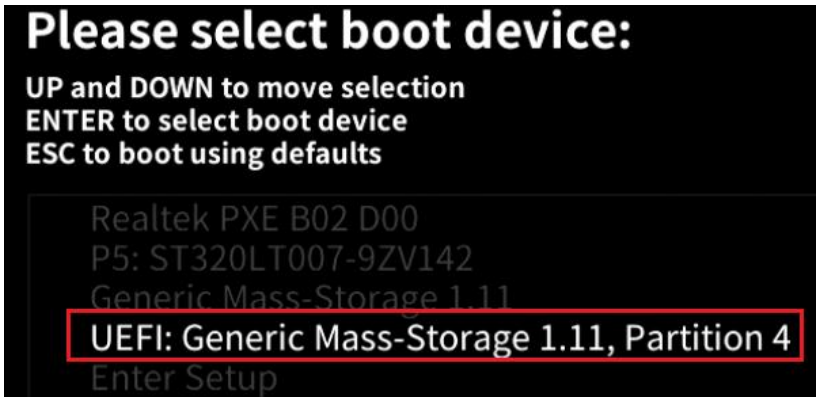


等待十几分钟，安装完毕，点击 Reboot 重启，ok 了

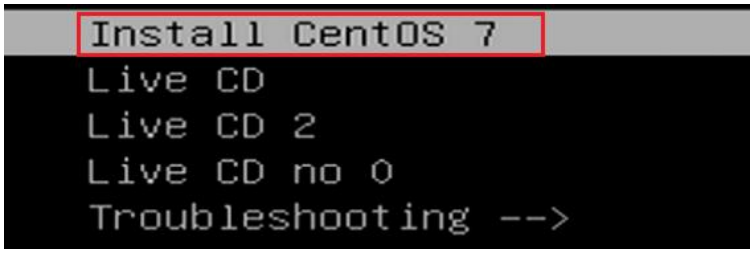



```
[root@localhost ~]# lsblk
NAME            MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda               8:0    0 298.1G  0 disk
├─sda1            8:1    0   800M  0 part /boot
└─sda2            8:2    0   290G  0 part
  └─centos-root 253:0    0   290G  0 lvm  /
```

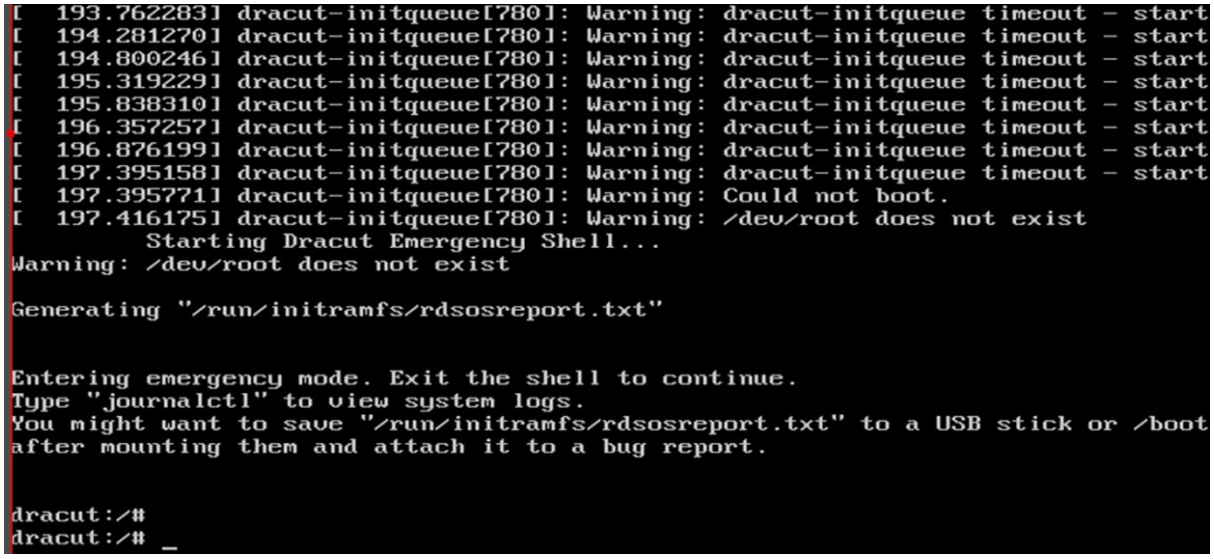
★UEFI 启动模式下



首先选择从目标 U 盘启动，



进入启动菜单后，按上键头选择“Install Centos7”安装 centos7



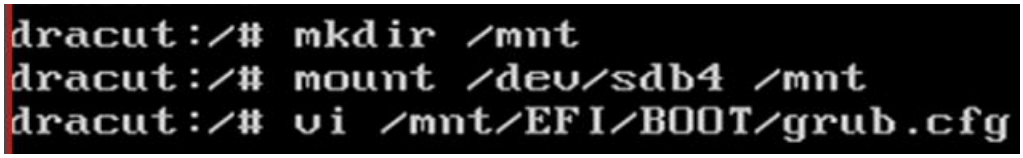
根据第 31 章的知识，应该是找不到安装源介质，因为我们使用刻录工具刻录时默认它只会生成 syslinux.cfg 配置文件，而不会帮我们改/EFI/BOOT/grub.cfg 配置，导致 grub2 找不到安装介质，这时候我们可以在 dracut 紧急模式下进行修改。

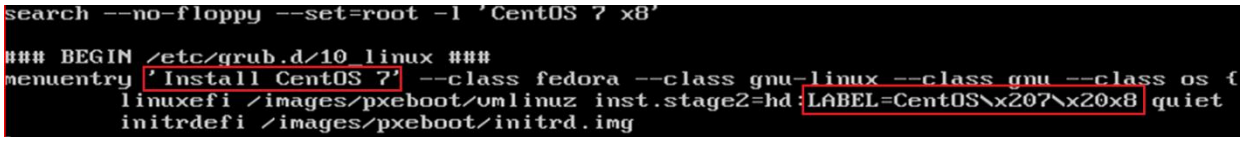
dracut# blkid #列出所有磁盘分区的相关信息

通过 blkid 查看到的信息，我们得知/dev/sdb4 为安装 U 盘，它的卷标为：
Centos 7 x8 ，而/EFI/BOOT/grub.cfg 文件里的 inst.stage2=hd:LABEL=的值却是：
Centos 7 x86_64，所以我们可以把它修改成使用 UUID=B4FE-5315，或者直接删掉几个字符，
改成 **Centos 7 x8** 就行了，这样还要简单些。

先挂载 u 盘到/mnt 目录下

```
dracut#   mkdir   /mnt                    #先创建挂载目录/mnt
dracut#   mount   /dev/sdb4   /mnt        #挂载 u 盘到/mnt 目录下
dracut#   vi   /mnt/EFI/BOOT/grub.cfg      #编辑 grub.cfg 配置文件
```

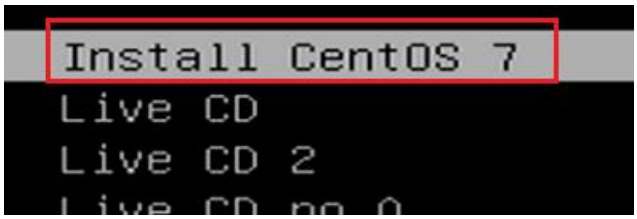


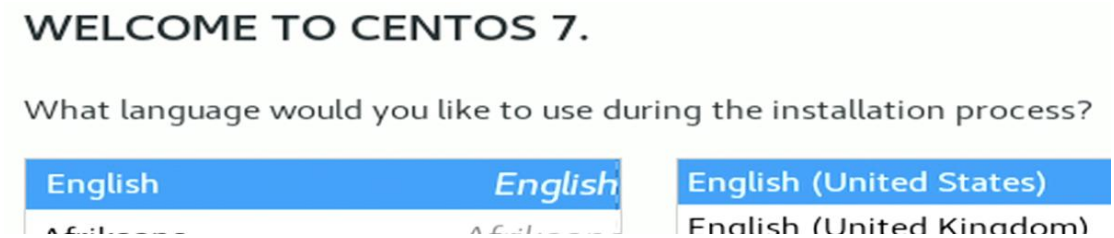


把里面的每个 menuentry 下的 inst.stage2=的值改为 CentOS\x207\x20x8
保存，退出（\x20 表示空格）

```
dracut#   init   6                    #重启，
```

再选择从 U 盘启动→“Install Centos7”

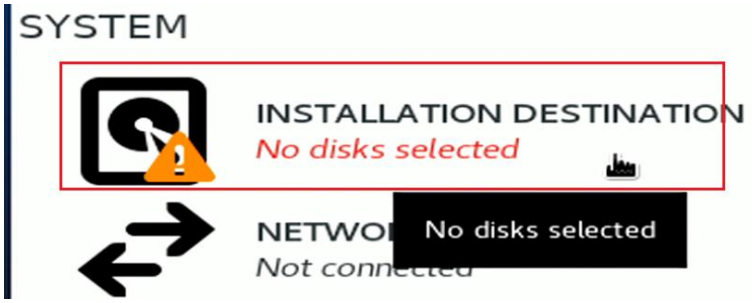




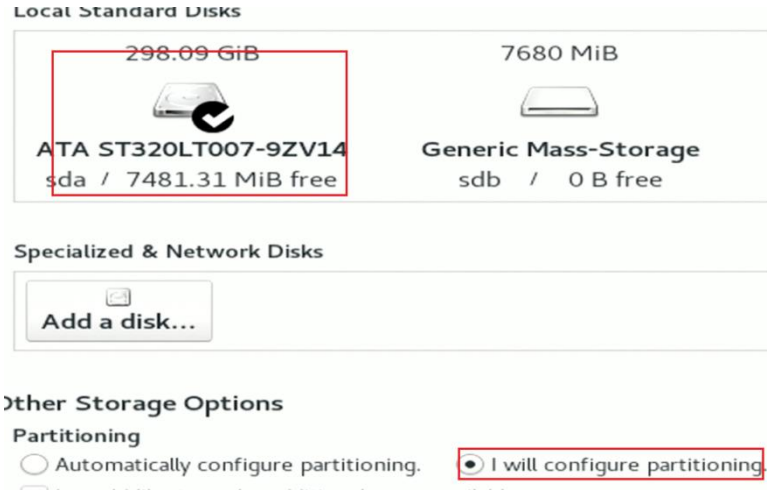
在安装向导里，选择目标语言，点击下一步

如上图，出现感叹号警告时，不要慌，正常现象，等一会儿就行了

在“Software Selection”里选择安装的版本，一般选择 Minimal Install 最小化版

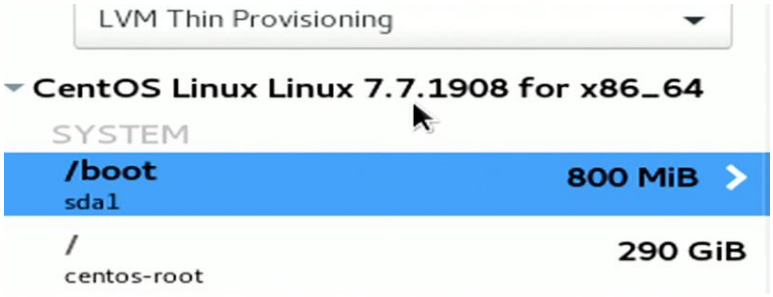


再点击“Installation Destination”，选择安装的目的磁盘，当然是目标计算机的硬盘了



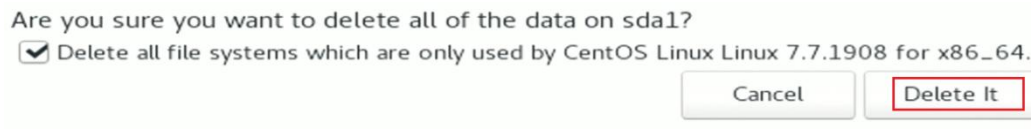
选中目标计算机的硬盘，再点击下方的“I will configure partitioning”进行分区配置，点击左上角的 Done

如果目标磁盘剩余空间不够，则一定是之前装有系统，可以先把旧的分区删除了，前提是确定原来的数据已经备份了。

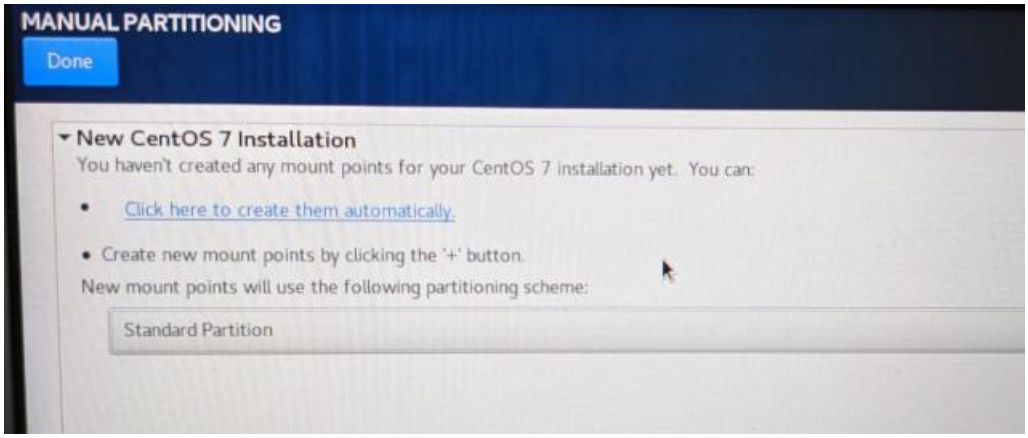


原来是之前装有过 centos 7 系统，展开它的分区项
删除/boot 分区

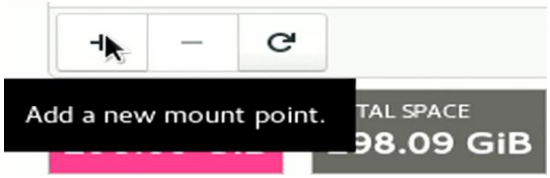
选中它的/boot 分区，点击左下角的减号- 进行删除



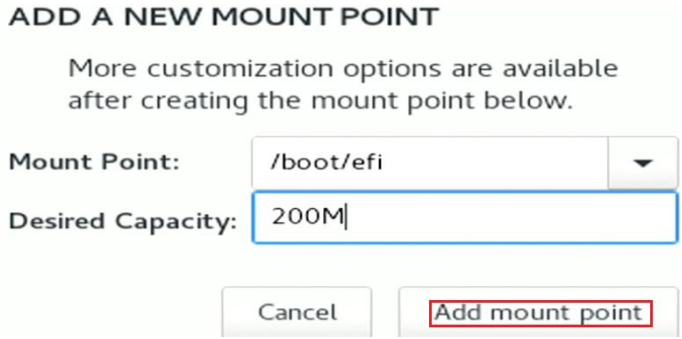
然后勾选“删除所有文件系统”，点击“Delete it”



然后 New mount points will use the following partitioning scheme 下选择使用标准分区 Standard Partition

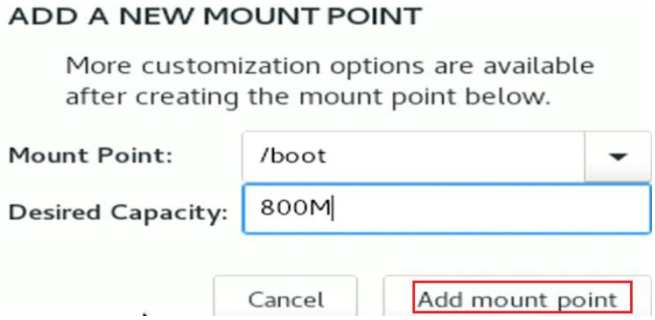


再点击左下角的加号+ ，新建一个分区，先创建/boot/efi 分区

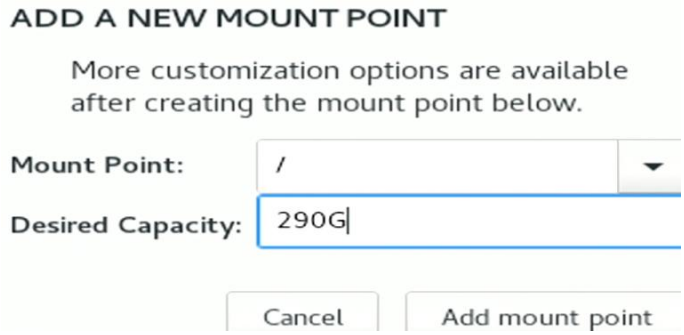


/boot/efi 分区一般 200M 就足够了，它只保存 grub2 引导相关的文件，用不了多少存储的。

再点击左下角的加号，添加/boot 分区

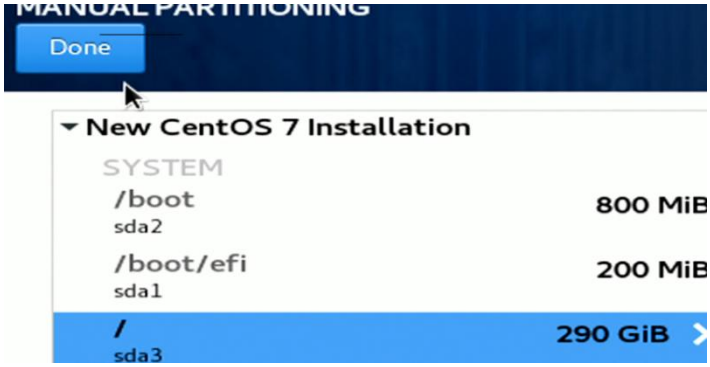


/boot 分区设置多大才合适呢？这得看后期想对目标系统安装几个内核文件，一般起始大小为 300M，以后打算增加一个内核，得加 100M 的大小，本例中设置 800M 也够了。

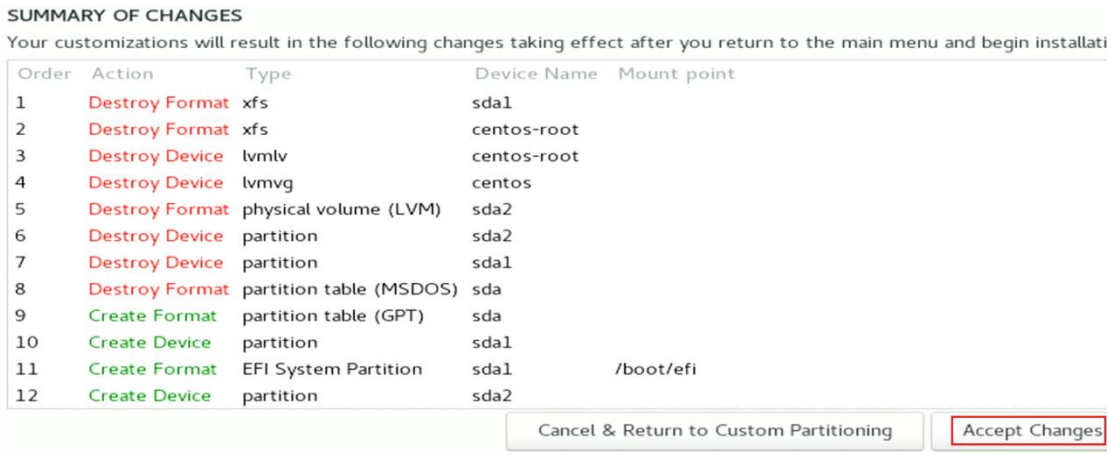


最后创建/根分区，把剩余的磁盘空间都划分给它

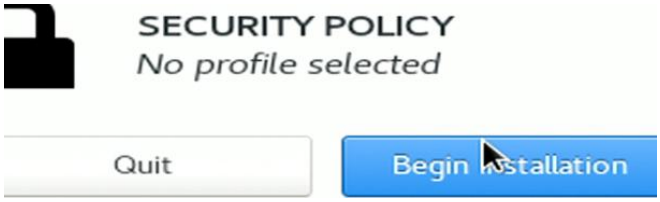
本例就一共只创建三个分区，如果有明确的需要，则要根据现实情况来划分分区



分区规划完成后，点击左上角的 Done，要点 2 次

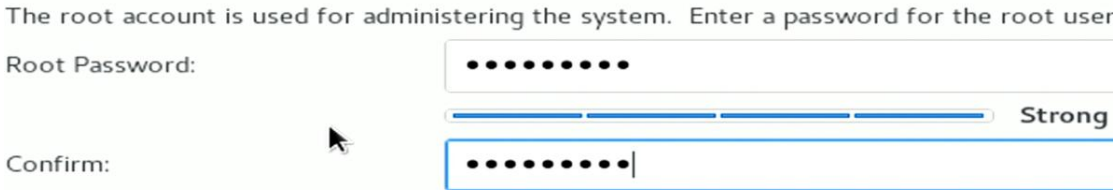


弹出一张总清单，点击右边的 Accept Changes



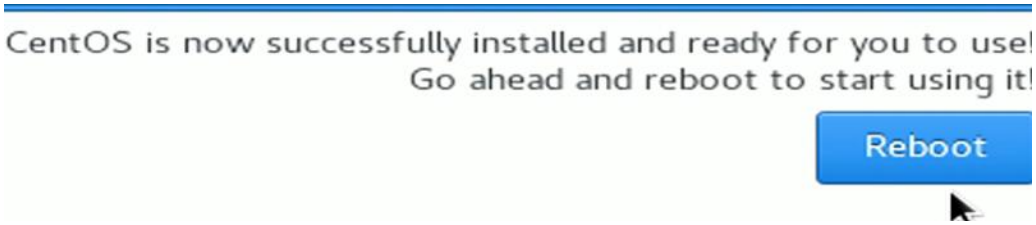
然后点击安装向导右下角的“Begin Installation”开始安装

在安装过程中，我们也可以先创建 root 用户的密码，点击“Root Password”



输入 root 密码

最后点击左上角的 Done ，如果密码强度不够，可能要点击 2 次



等待十几分钟，安装完毕，点击 Reboot 重启，ok 了

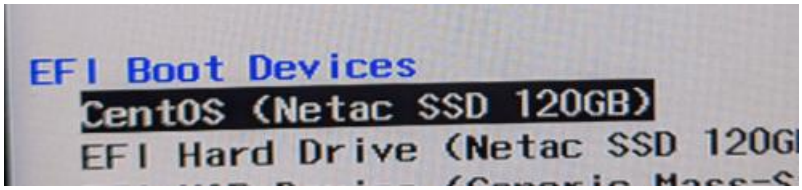
```
[root@localhost ~]# lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 298.1G  0 disk
├─sda1   8:1    0   200M  0 part /boot/efi
├─sda2   8:2    0   800M  0 part /boot
└─sda3   8:3    0   290G  0 part /
```

```
[root@localhost ~]# efibootmgr -v
BootCurrent: 0006
Timeout: 5 seconds
BootOrder: 0006,0000,0005,0002,0004
Boot0000* CentOS        HD(1,GPT,af957e56-8463-402e-88c0-090b0bb233d6,0x800,0x64000)/File(\EFI\CENTOS\SHIMX64.EFI)
Boot0002* Hard Drive    BBS(HD,,0x0)..GO..NO........o.S.T.3.2.0.L.T.0.0.7.-.9.Z.V.1.4.2..................A.....
....u.G.e.n.e.r.i.c. .M.a.s.s.-.S.t.o.r.a.g.e. .1...1.1..................A.....................H..Gd-.;.A..MQ..L.G.e.n.
Boot0004* Network Card  BBS(Network,,0x0)..GO..NO........k.R.e.a.l.t.e.k. .P.X.E. .B.0.2. .D.0.0.....................
Boot0005* UEFI: Generic Mass-Storage 1.11, Partition 4  /Pci(0x14,0x0)/USB(8,0)/HD(4,MBR,0xcad4ebea,0x100,0xefff00)..BO
Boot0006* CentOS        HD(1,GPT,af957e56-8463-402e-88c0-090b0bb233d6,0x800,0x64000)/File(\EFI\CENTOS\SHIM.EFI)..BO
```

```
[root@localhost ~]# efibootmgr
BootCurrent: 0006
Timeout: 5 seconds
BootOrder: 0006,0000,0005,0002,0004
Boot0000* CentOS
Boot0002* Hard Drive
Boot0004* Network Card
Boot0005* UEFI: Generic Mass-Storage 1.11, Partition 4
Boot0006* CentOS
```

uefi 启动模式下，安装完系统后，centos7 会自动添加一条 uefi 启动菜单，名为 CentOS 或 Centos Linux，从目标磁盘的 /EFI/centos/shimx64.efi 文件启动

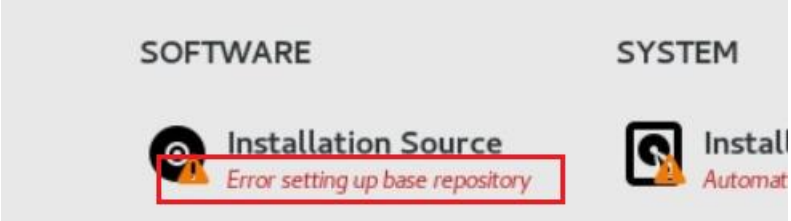
有的版本从/EFI/CENTOS/SHIM.EFI

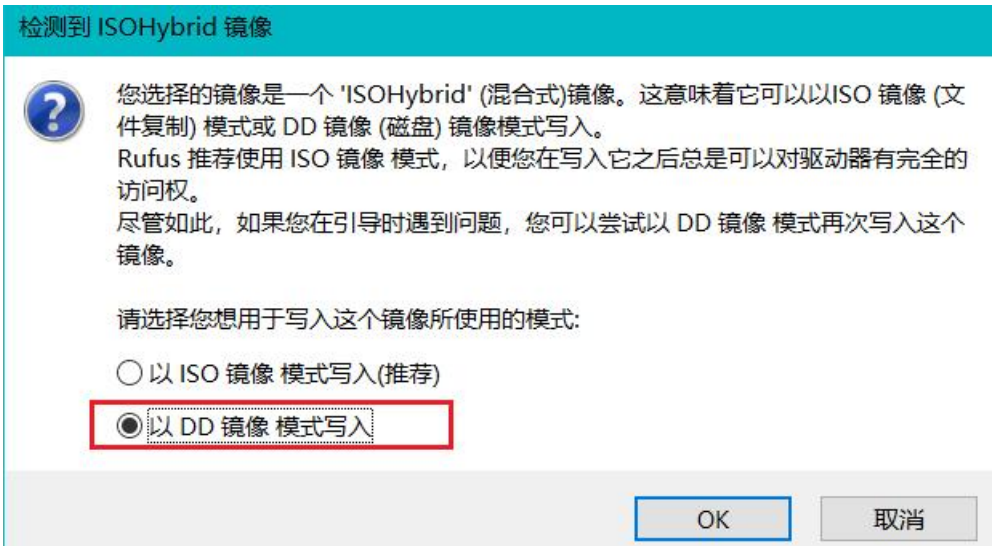


且设置为第一启动项。有关 UEFI 的具体情况请看第 31 章。

★U 盘安装 centos 报错：Error setting up base repository

原因未知，可以使用 rufus 工具进行刻录，选择“以 DD 镜像模式写入”





第 31 章、深入了解 UEFI

人们习惯于把硬件之上 到正式系统之下的这个中间过程用到的功能组件称为固件（Firmware），比如 BIOS 固件和 UEFI 固件，不过作者还是喜欢称之为预启动程序，进入正式系统之前除了预启动程序，还有引导程序（bootloader），这个引导程序是固件和正式系统之间的一层

在本文档中，在硬件之上到正式系统之下的中间过程用到的功能组件有：
预启动程序（计算机主板在出厂时固化在 ROM 芯片里的程序，即主板上的 firmware）
引导程序（使用工具刻录到启动磁盘的 MBR 或 PBR 里的程序及后续加载的其他引导文件，即 bootloader）

UEFI 预启动程序也不用把它想得太复杂太高深，我们就认为它是新版本的 BIOS，功能上比原来的要强大一些而已。本来所有的预启动程序都叫 BIOS（基本输入输出系统），不同厂商主板上的 BIOS 系统功能大同小异，后来出了个名为 UEFI 的 BIOS，它和传统的 BIOS 有很大的差别，所以原来的预启动程序就叫 Legacy BIOS 了，再后来人们为了方便称呼，就仍把传统的 Legacy BIOS 叫作 BIOS，把 UEFI BIOS 叫作 UEFI。也就是 UEFI 要与传统的预启动程序划清界线。

UEFI 只是制定了统一的规范，并不是一个程序，而是规范，由不同的主板厂商去开发符合这个规范的预启动程序。
UEFI 有哪些规范呢？
1.支持 CSM 兼容模式
2.支持 FAT 文件系统
3.UEFI 启动菜单能从指定的 efi 文件启动
4.安全启动

①CSM 兼容模式
大多数厂商的 UEFI 预启动程序都实现了 csm 兼容模式，就是可以从 使用 MBR 引导 的磁盘启动，并给这些引导提供传统 BIOS 的环境，从而让这些引导能正常运行，引导系统。当开启 CSM 时，uefi 可以从使用 MBR 分区的磁盘的 FAT 文件系统里的/EFI/BOOT/BOOT*xxx*.EFI 文件启动。

②FAT 文件系统
UEFI 要求必须能够识别并读取 FAT 系列的文件系统，从而能从 FAT 分区里加载 uefi 引导文件，FAT 系列文件系统有：FAT12、FAT16、FAT32、vFAT
但并未表明只能从 FAT 系列的分区启动，所以 Apple 公司就在他们的 Mac 电脑上使用 HFS+的分区作为 EFI 分区

③UEFI 启动菜单
传统的 bios 启动菜单只能选择从哪个磁盘或分区启动，而 UEFI 的启动菜单除了可以从磁盘/分区启动外，还多了一种，可以从文件启动，即由正式的操作系统在安装完毕后创建一条 uefi 启动菜单，指明从某磁盘的某 efi 分区的某个 xxx.efi 文件启动，这样一来就省去了让 uefi 预启动程序自己去目标磁盘上找可启动的 efi 分区（ESP）里的/EFI/BOOT/目录下的对应平台的 bootxxx.efi 文件，为什么要这样做呢，或者说为什么要有“从文件启动”的菜单呢？
UEFI 规范是这样说的：

当启动时指定从某磁盘启动，它就会去找目标磁盘上的第一个 EFI 分区，再判断当前计算机平台（如 x86, x86_64, arm, arm64 等），再去找这个 efi 分区里的/EFI/BOOT/目录下的对应的 efi 程序文件。根据不同的平台，这个 efi 文件名称也不一样，具体如下：

计算机平台	目标 efi 程序文件
x86_32 （IA32）	BOOTia32.EFI
x86_64 （amd64）	BOOTx64.EFI
Itanium （IA64）	BOOTia64.EFI
ARM （AArch32）	BOOTarm.EFI
AA64 （AArch64，arm64）	BOOTaa64.EFI

如果我们用的电脑是 x86_64 的平台，则见到的启动文件为 EFI 分区下的/EFI/BOOT/BOOTx64.EFI
因为 EFI 分区一般为 FAT 系列的文件系统，windows 系统里不区分大小写，在 Linux 正式系统里还是要区分大小写的，有的 UEFI 预启动程序也不区分大小写。

如果按照上面的说法，一个 EFI 分区就只能引导一个操作系统了，因为不管 EFI 分区里有多少个 efi 引导程序，它都只会找其中一个 BOOTxxx.EFI，而 UEFI 又想能直接从不同的 BOOTxxx.EFI 文件去启动，或者从用户指定的位于 EFI 分区里的任何目录下的任意名字的 xxx.efi 文件启动。所以，UEFI 又设计了一种“从文件启动”的菜单项（为了能自定义要加载的引导程序文件）

还有一个问题，刚刚说是正式的系统在安装完毕后才创建一条 UEFI 启动菜单，从文件启动。我们在装系统时，在进入安装向导时，就已经错过了配置 uefi 预启动程序的配置界面了（开机后按下 Del 或 F2 进入的那个设置 uefi 的界面）
那么正式的操作系统是如何能在 非 uefi 配置界面 进行 写入这条从文件启动的 uefi 启动菜单呢？

因为 UEFI 规范允许用户在进入正式的操作系统后，也能配置 uefi 启动菜单项，这在传统的 bios 启动下的操作系统里是无法完成的事，而 uefi 就可以。
在正式的系统里可以借助 efibootmgr 工具或 EasyUEFI 工具去查看并修改或添加删除 某条 uefi 启动菜单。
（linux 系统用 efibootmgr，windows 可用 EasyUEFI）


```
[root@localhost ~]# efibootmgr
BootCurrent: 0006
Timeout: 5 seconds
BootOrder: 0006,0000,0005,0002,0004
Boot0000* CentOS
Boot0002* Hard Drive
Boot0004* Network Card
Boot0005* UEFI: Generic Mass-Storage 1.11, Partition 4
Boot0006* CentOS
```

linux 使用 efibootmgr -v 查看时，可见当前的启动菜单项 BootCurrent 为 0006，然后下面列出了若干条启动菜单项，每项前面用 Bootxxxx*表示编号，后面为启动菜单项的名称

可以删除某条启动菜单项：

```
# efibootmgr  -b  0002  -B                    #删除 Boot0002 的启动菜单项
```

或者添加一个从某文件启动的菜单项：

```
# efibootmgr  -c  -w  -L  "label name"  -d  /dev/sdc  -p  1  -l  \\EFI\\BOOT\\grubx64.efi
#表示从/dev/sdc 的第 1 个分区里的/EFI/BOOT/grubx64.efi 文件启动，菜单项名称为-L 指定的 label name，文件路径分隔符要用双反斜杠\\表示
```

④安全启动（secure boot）

UEFI 安全启动是利用数字签名机制来确认 efi 引导程序/驱动程序是否是受信任的，当开启了安全启动时，只有受信任的 efi 引导程序/驱动程序才能被加载并运行。类似于浏览器在访问网站时的那个 ssl 证书的安全性。

也就是说，主板厂商会把一些常见的 CA 证书预存在他们的主板上，当开启安全启动功能后，uefi 预启动程序会先检查要加载的 efi 引导程序是否有数字签名，没有则不加载它，有则用预存的 CA 证书去验证它的数字签名是否为可信的，是可信的则加载，不是则不加载。

所以当开启了安全启动功能后：

我们自己写的 efi 引导程序得签名，去哪儿签名呢？得看主板里存放了哪些 CA 证书，比如有微软的 CA 证书时，我们就去找微软签名我们的程序，有其他 CA 机构的证书时，就去找其他 CA 机构签名。问题来了，主板里预存的 ca 证书有哪些 CA 机构的？怎么找它们签名？签名要钱吗？

答案是：大多数主板厂家只预存了微软的 CA 证书，而且找微软签名有可能是要钱的。所以当启动 windows 系统时能启动，而启动其他系统时，可能其他系统没有找微软签名，导致引导程序“不可信”而无法启动。这也就是为什么 当我们安装非 windows 系统时，要关闭 uefi 安全启动 的原因。

那为什么 Linux 等系统不让主板厂商预装他们的 CA 证书呢？因为商业上的原因及开源世界对自由使用软件的原则，所以他们没有给自己的系统做 uefi 的签名。

不过后来也有某些 linux 发行版本开始找微软签名了，比如 Ubuntu（ubuntu 有了签名后，在它的 ubuntu lts 16.04 系统强制要求使用安全启动，真是妙啊）

微软和 Ubuntu 一起搞了个 shim 引导（中间层引导），就是使用微软的 CA 证书去给这个 shim 引导签名，这样这个 shim 引导就能在开启了安全启动的计算机上加载并运行了，shim 只是个引导程序，由它再去引导其他的 efi 引导，最终引导正式的操作系统。

好消息就是 shim 自己也可以给其他的 efi 引导程序和内核程序签名，这样有了 shim 以后，就可以不再找微软要签名了，直接找 shim 就可以了（shim 当成了中间 CA 证书）

在 Ubuntu 新版本系统里，启动时的加载顺序为：

①uefi 预启动程序从文件启动时（ubuntu 系统的 uefi 启动菜单项里指定的 efi 文件）

先用主板上的 CA 证书验证启动文件（如/EFI/ubuntu/shimx64.efi）

验证通过就加载它，并把引导权交给 shimx64.efi

②shim 再去验证并加载同目录下的 grubx64.efi 引导程序，

③grub2 启动后再回调 shimx64.efi 程序去验证内核文件，验证通过后就加载内核，剩下的就和平常一样了。

告诉大家一个秘密，centos7 也是这样启动的，它也用了 shim，所以 centos7 也是能在开启了安全启动的时候正常启动。它的引导加载流程如下：

/EFI/centos/shimx64.efi → /EFI/centos/grubx64.efi → 回调 shimx64.efi 验证 linux 内核 → 加载内核启动系统

当我们选择从磁盘启动时，uefi 预启动程序会去找/EFI/BOOT/BOOTx64.efi，这时的 BOOTx64.efi 不是 grub2 引导，而是 shim，它和/EFI/centos/shimx64.efi 是同一个文件，可以用 hash 校验查看得知。然后这个/EFI/BOOT/Bootx64.efi 会再去加载/EFI/centos/grubx64.efi

第 32 章、深入了解 grub

在 centos6 等其他旧版的系统上是使用 grub 做引导的，为了方便维护这些系统，有必要了解一下 grub 的安装。

在 windows 系统上暂没找到可以直接安装 grub 到目标磁盘的工具，不过有个叫 grub4dos 的，本章先不讲这个。所以我们直接用 centos6 系统去操作，或者直接使用 centos6 的安装光盘去启动进入 Live 系统，在 Live 系统里使用 grub 工具去安装 grub 引导到目标磁盘。

★在 centos6 里安装 grub 到目标 U 盘（BIOS 模式）

此 centos6 系统是在传统 BIOS 启动模式下进入的系统，所以我们的 grub 引导也是在传统 BIOS 模式下的引导，首先插入 U 盘到 centos6 计算机上

```
[root@centos6 ~]# lsblk
NAME                        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0                          11:0    1  408M  0 rom
sda                           8:0    0   20G  0 disk
├─sda1                        8:1    0  500M  0 part /boot
└─sda2                        8:2    0 19.5G  0 part
  ├─VolGroup-lv_root (dm-0) 253:0    0 17.5G  0 lvm  /
  └─VolGroup-lv_swap (dm-1) 253:1    0    2G  0 lvm  [SWAP]
sdb                           8:16   1  7.5G  0 disk
└─sdb1                        8:17   1  7.5G  0 part
```

lsblk　　#查看当前系统的磁盘及分区情况，本例中 sdb 是我们的 U 盘，只有一个分区 sdb1

```
[root@centos6 ~]# mount /dev/sdb1 /mnt
[root@centos6 ~]# ls /mnt
autorun.ico       isolinux                  RPM-GPG-KEY-CentOS-Debug-6
autorun.inf       ldlinux.sys               RPM-GPG-KEY-CentOS-Security-6
CentOS_BuildTag   LOST.DIR                  RPM-GPG-KEY-CentOS-Testing-6
EFI               Packages                  syslinux.cfg
EULA              RELEASE-NOTES-en-US.html  System Volume Information
GPL               repodata                  TRANS.TBL
images            RPM-GPG-KEY-CentOS-6      WhatsApp_2.20.202.apk
```

mount　/dev/sdb1　/mnt　　#把 sdb1 挂载到/mnt 目录下

ls　/mnt

#可见 sdb1 分区里的文件，这里可以确定是我们的 U 盘，因为这个 U 盘上面有我之前刻录的 centos6 安装源文件，如果不是则要重新找一下再挂载。若我们插入的 U 盘没有分区或分区不识别，则要提前备份 U 盘里的文件，再在 centos6 里重新创建分区，再创建文件系统，grub 能识别的文件系统比较多，一般可以有 ext2，ext3，ext4，还有 FAT 文件系统，本例中作者的 U 盘已有一个分区，是 FAT32 的文件系统。

```
[root@centos6 ~]# grub-install --root-directory=/mnt --no-floppy /dev/sdb
Probing devices to guess BIOS drives. This may take a long time.
Installation finished. No error reported.
This is the contents of the device map /mnt/boot/grub/device.map.
Check if this is correct or not. If any of the lines is incorrect,
fix it and re-run the script `grub-install'.

(hd0)   /dev/sda
(hd1)   /dev/sdb
[root@centos6 ~]#
```

grub-install　--root-directory=/mnt　--no-floppy　/dev/sdb

#安装 grub 引导到 u 盘，末尾的/dev/sdb 如果不加数字 1，2 之类的，则表示安装 MBR 里，而不是安装到分区里，这样就可以直接从此 U 盘启动，

--no-floppy　　#表示启动时不查找软盘

--root-directory=/mnt　　#表示要把 grub 相关的文件复制到这个目录下的 boot/grub 子目录，比如 stage1，stage1_5，stage2 文件

这些 stage 文件从哪来的？当然是 centos6 系统上有的，centos6 在安装时默认就装有 grub 这个工具，这个工具在安装时就会把这些 stage 文件放在/usr/share/grub 目录下的对应平台子目录里

```
[root@centos6 ~]# ls /usr/share/grub
x86_64-redhat
[root@centos6 ~]# ls /usr/share/grub/x86_64-redhat/
e2fs_stage1_5      jfs_stage1_5       stage2           xfs_stage1_5
fat_stage1_5       minix_stage1_5     stage2_eltorito
ffs_stage1_5       reiserfs_stage1_5  ufs2_stage1_5
iso9660_stage1_5   stage1             vstafs_stage1_5
```

grub 在安装到磁盘时，会把 stage1 文件（即第一阶段引导代码）复制到目标磁盘的 MBR 扇区里，如果是要安装到分区上，则复制到分区的 PBR 里。

再判断--root-directory=指定的目录 所对应分区的文件系统类型，把对应分区类型的 stage1_5 文件复制到 MBR 扇区之后的几个扇区里，（如果是安装到分区，则把 stage1_5 文件复制到 PBR 扇区之后的几个扇区里）

本例中，目标 U 盘的分区为 FAT32 文件系统，所以 grub 在安装时会把 fat_stage1_5 文件写到 MBR 扇区之后的扇区里。然后 stage1_5 默认是会去该分区里的/boot/grub/目录下找 stage2 引导文件和 grub.conf 配置文件。

```
[root@centos6 ~]# ls /mnt --color=none
autorun.ico       isolinux                    RPM-GPG-KEY-CentOS-
autorun.inf       ldlinux.sys                 RPM-GPG-KEY-CentOS-
boot              LOST.DIR                    syslinux.cfg
CentOS_BuildTag   Packages                    System Volume Infor
EFI               RELEASE-NOTES-en-US.html    TRANS.TBL
```

```
[root@centos6 ~]# cd /mnt/boot
[root@centos6 boot]# ls
grub
[root@centos6 boot]# cd grub
[root@centos6 grub]# ls
device.map      iso9660_stage1_5   stage1           xfs_stage1_5
e2fs_stage1_5   jfs_stage1_5       stage2
fat_stage1_5    minix_stage1_5     ufs2_stage1_5
ffs_stage1_5    reiserfs_stage1_5  vstafs_stage1_5
[root@centos6 grub]#
```

在目标磁盘的启动分区里已经创建了/boot/grub 目录，grub 在安装时也已经把 stage1,1_5,2 文件复制到此目录下了，默认没有 grub.conf 配置文件，要我们自己创建。

```
[root@centos6 grub]# pwd
/mnt/boot/grub
[root@centos6 grub]# vi grub.conf _
```

创建 grub.conf 文件（位于 sdb1 分区里的 boot/grub 子目录下）内容如下：

```
default=0
splashimage=/EFI/BOOT/splash.xpm.gz
timeout  5
title  CentOS 6.6
    kernel  /images/pxeboot/vmlinuz
    initrd  /images/pxeboot/initrd.img
title  Install  system  with  basic  video  driver
    kernel  /images/pxeboot/vmlinuz  xdriver=vesa  nomodeset  askmethod
    initrd  /images/pxeboot/initrd.img
title  rescue
    kernel  /images/pxeboot/vmlinuz  rescue  askmethod
    initrd  /images/pxeboot/initrd.img
```

#这个内容可以先随便写，本例主要目录是安装 grub，能进入 grub 菜单就算成功了

```
[root@centos6 grub]# cat grub.conf
default=0
splashimage=/EFI/BOOT/splash.xpm.gz
timeout 5
title CentOS 6.6
        kernel /images/pxeboot/vmlinuz
        initrd /images/pxeboot/initrd.img
title Install system with basic video driver
        kernel /images/pxeboot/vmlinuz xdriver=vesa nomodeset askmethod
        initrd /images/pxeboot/initrd.img
title rescue
        kernel /images/pxeboot/vmlinuz rescue askmethod
        initrd /images/pxeboot/initrd.img
[root@centos6 grub]#
```

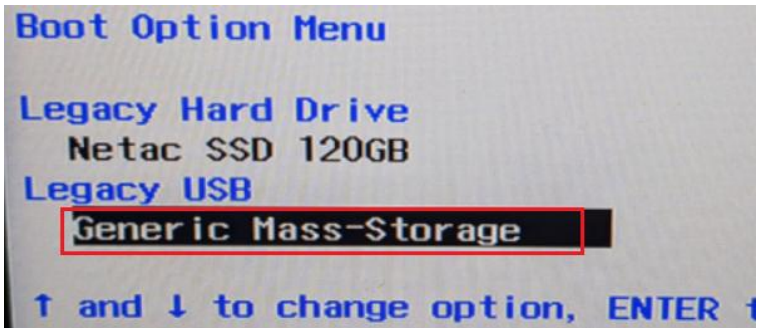
保存，

```
[root@centos6 grub]# cd /root
[root@centos6 ~]# umount /mnt
[root@centos6 ~]#
```

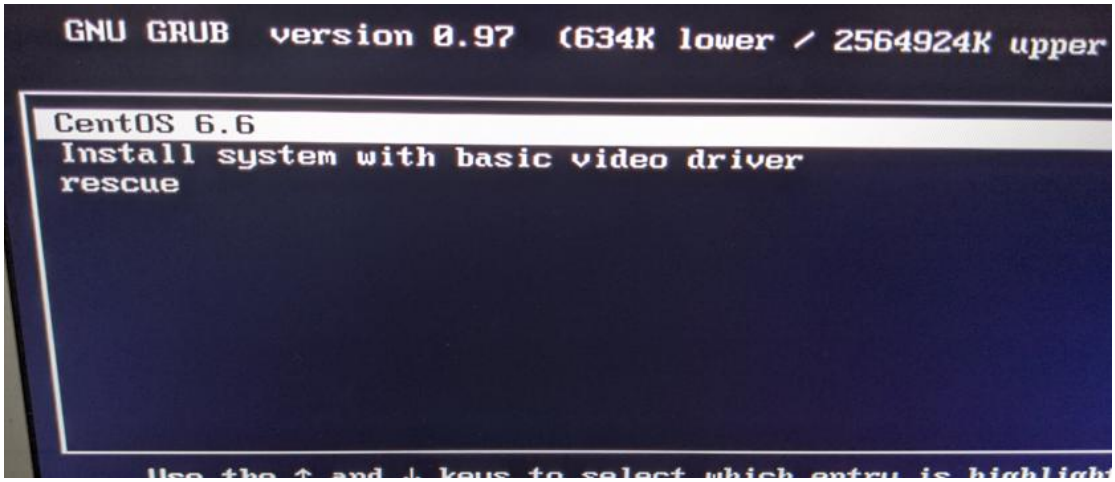
退出/mnt 目录，卸载/mnt 目录

umount /mnt

卸载成功后再拔出 U 盘



把 U 盘插入目标计算机，启动计算机时选择从此 U 盘启动



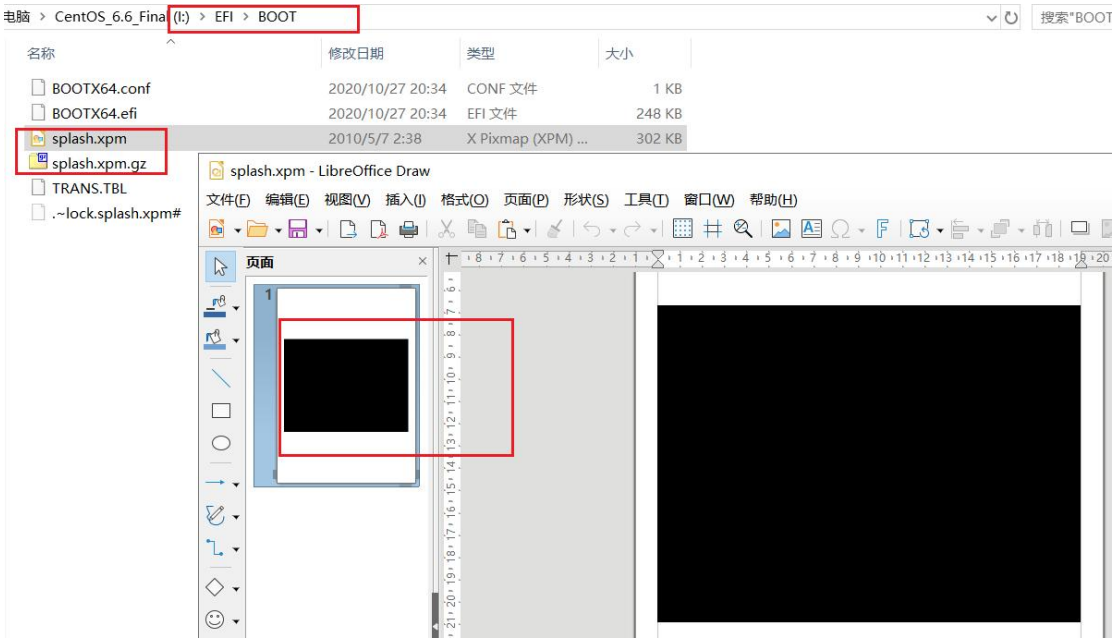
如上图，已经成功从 U 盘启动，也显示 grub 菜单，ok 了，grub 引导安装成功。

补充：

在目标磁盘的启动分区的/boot/grub 目录下有若干 stage 文件，其实只有 stage2 是用得到的，其他的在安装 grub 到磁盘上时就已经嵌入到磁盘的 MBR 及 MBR_gap 扇区里了，从此磁盘启动时，已经用不到此目录下的 stage1 和 stage1_5 文件了。只留 stage2 和 grub.conf 两个文件就行了。

那为什么安装 grub 时还要把这么多 stage1 及 stage1_5 文件复制到此目录下呢？
是为了方便以后的维护，以后在用这个 U 盘启动时，可以再用此 U 盘上的 grub 程序去安装 grub 引导到其他的磁盘上，而其他的磁盘分区的文件系统可能各种各样，所以最好还是带上这么多的 stage1_5 文件，以便应对不同的文件系统。这些文件本身不大，留着也不占存储。

还有，细心的朋友可能已经发现了，我在写 grub.conf 配置文件时，已经指定了背景图片为 U 盘启动分区里的/EFI/BOOT/splash.xpm.gz 文件，而且我的 U 盘之前正好是刻录了 centos6 的安装光盘文件，所以也有这个 splash.xpm.gz 文件，那么，刚刚在启动时，为什么背景还是黑色的？
我们进入 U 盘的/EFI/BOOT/目录，把 splash.xpm.gz 文件解压后就得到一个 splash.xpm 文件，用能打开.xpm 格式图片的看图工具打开来看一看就明白了，如下图：



原来这个背景图片本来就是全黑的，@_@，我们也可以换成自定义的背景图片，具体的请看之后的章节。

★也可进入 grub 交互界面进行安装

mount /dev/sdb1 /mnt
mkdir /mnt/grub
#在 U 盘启动分区上创建一个名为 grub 的目录，目录名字可以随便取

首先查看目标分区的相关目录里是否有 stage1，stage1_5，stage2 文件，如果没有则先复制（这里当然没有，因为是刚刚创建的目录，但有时我们会去修复现有系统的 grub 引导，所以要查看原/boot 分区的 grub 目录里是否有引导文件），可以从 centos6 系统里的/usr/share/grub/x86_64-redhat/目录去复制，复制里面的所有文件到 U 盘目标目录里

cp /usr/share/grub/x86_64-redhat/* /mnt/grub/

```
[root@centos6 ~]# mkdir /mnt/grub
[root@centos6 ~]# cp /usr/share/grub/x86_64-redhat/*  /mnt/grub/
[root@centos6 ~]#
[root@centos6 ~]# ls /mnt/grub
e2fs_stage1_5     jfs_stage1_5       stage2           xfs_stage1_5
fat_stage1_5      minix_stage1_5     stage2_eltorito
ffs_stage1_5      reiserfs_stage1_5  ufs2_stage1_5
iso9660_stage1_5  stage1             vstafs_stage1_5
```

接着再在/mnt 目录下（即 U 盘的启动分区根目录下）创建一个**文件**，名字随便取，最好不要和已有的文件名相同

touch /mnt/myudisk

然后就可以进入 grub 交互界面进行操作了

命令行里直接输入 grub 命令就进入交互界面了，交互界面以 grub>为提示符

要确认在 grub 里，目标 U 盘的设备及分区名称，可以用 find 命令查找刚刚在 U 盘的启动分区根目录下创建的 myudisk 文件，然后得知它的设备及分区号（创建/mnt/myudisk 文件的作用就在于此）

grub 的设备命名规则：磁盘是从 0 开始编号，分区从 0 开始编号

例如第 1 块硬盘的第 1 个分区表示为 (hd0,0)

```
[root@centos6 ~]# grub
Probing devices to guess BIOS drives. This may take a long time.


    GNU GRUB  version 0.97  (640K lower / 3072K upper memory)

 [ Minimal BASH-like line editing is supported.  For the first word, TAB
   lists possible command completions.  Anywhere else TAB lists the possible
   completions of a device/filename.]
grub> find /myudisk
find /myudisk
 (hd1,0)
grub>
```

> find /myudisk #交互界面里查找哪个磁盘/分区下有 myudisk 文件，本例中为(hd1,0)

> root (hd1,0) #表示设置当前工作分区为 U 盘的启动分区

> setup (hd1) #安装 grub 引导到 hd1 磁盘，即我们的 U 盘

```
grub> root (hd1,0)
root (hd1,0)
 Filesystem type is fat, partition type 0xc
grub> setup (hd1)
setup (hd1)
 Checking if "/boot/grub/stage1" exists... yes
 Checking if "/boot/grub/stage2" exists... yes
 Checking if "/boot/grub/fat_stage1_5" exists... yes
 Running "embed /boot/grub/fat_stage1_5 (hd1)"...  25 sectors are embedded.
succeeded
 Running "install /boot/grub/stage1 (hd1) (hd1)1+25 p (hd1,0)/boot/grub/stage2
boot/grub/grub.conf"... succeeded
Done.
grub> _
```

默认是会去找此 U 盘的启动分区里的/grub 目录，并判断是否有 stage 相关文件，有则能成功安装，安装成功后，退出 grub 就行了，最后在/grub 目录下手动创建 grub.conf 配置文件即可。

如果我们要自定义存放 stage 文件的目录及配置文件名，怎么办呢？
可以在交互界面里使用 install 命令

```
grub> root (hd1,0)
root (hd1,0)
 Filesystem type is fat, partition type 0xc
grub>

grub> install /boot/grub/stage1 (hd1) /boot/grub/stage2 p /boot/grub/grub.conf
install /boot/grub/stage1 (hd1) /boot/grub/stage2 p /boot/grub/grub.conf
grub>
```

> root (hd1,0) #同样是要先指定工作分区

> install /boot/grub/stage1 (hd1) /boot/grub/stage2 p /boot/grub/grub.conf

#指明启动第一阶段的文件路径，然后是要安装到的磁盘为 hd1，接着是启动第二阶段的文件路径，最后用 p 指明配置文件

> quit #退出 grub 交互界面

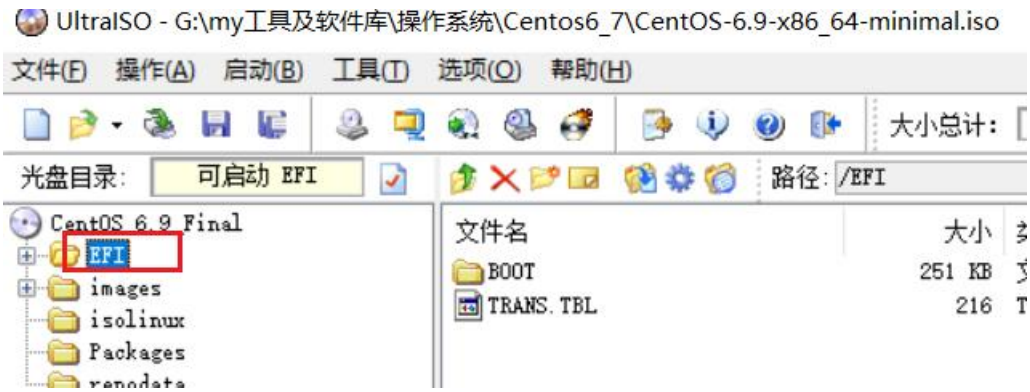
```
grub> quit
quit
```

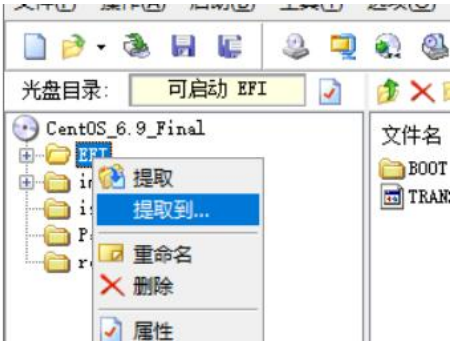
安装引导完成后，卸载 U 盘即可

★安装 grub 到目标 U 盘（UEFI 模式）

本小节就不进入 centos6 系统了，直接在 windows 上操作吧，
因为大多数的主板 UEFI 都支持 CSM 兼容模式，所以我们的 U 盘也可以仍旧使用 MBR 分区表，
把目标 U 盘格式化为 FAT32 文件系统，本例中 U 盘卷标为 **CENTOS_6_6_**

然后用 Ultraiso 工具或其他能打开.iso 文件的工具将 centos6.x 的安装光盘镜像文件打开，将里面的 EFI 文件夹整个导出到目标 U 盘根目录下





提取到目标 U 盘后，进入 U 盘看看，

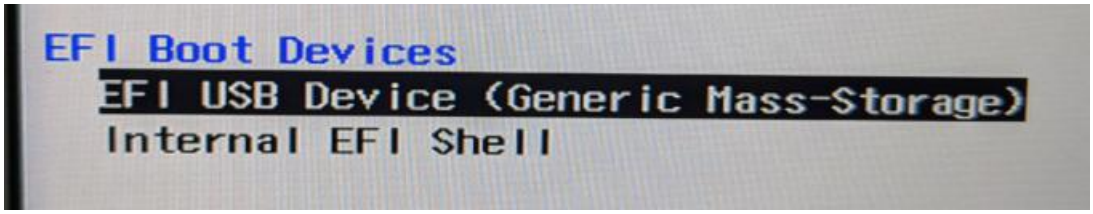
U 盘里面的/EFI/BOOT/目录下的 BOOTX64.efi 就是 grub 引导程序，是 uefi 启动模式下的 grub 引导，使用的是 efi 字节码。这一个文件里已经包含 stage1，所有的 stage1_5 及 stage2 文件。

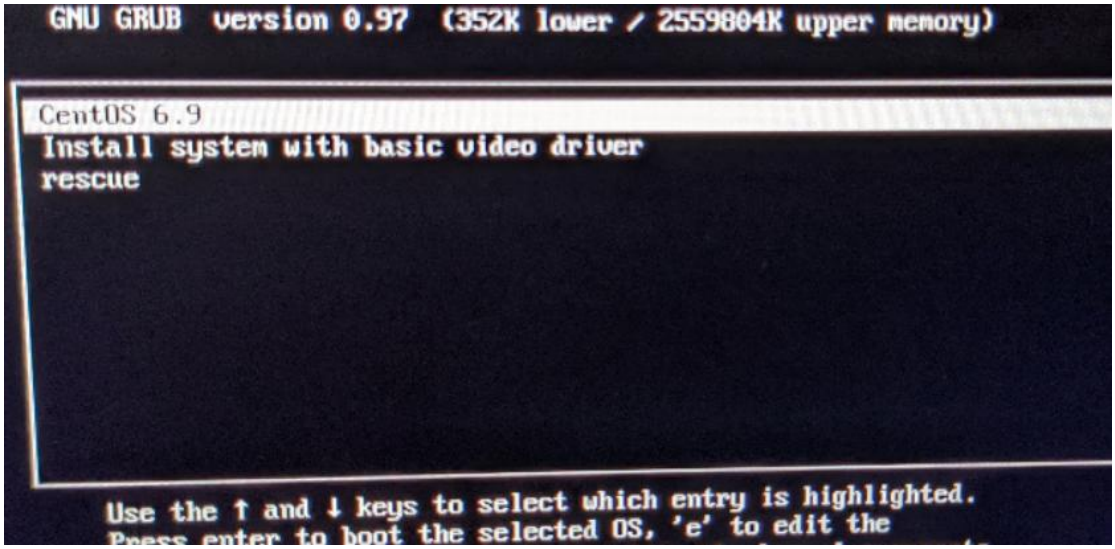
那个 BOOTx64.conf 就是 grub 的配置文件，内容如下：

```
#debug --graphics
default=0
splashimage=/EFI/BOOT/splash.xpm.gz
timeout 5
#hiddenmenu
title CentOS 6.9
    kernel /images/pxeboot/vmlinuz
    initrd /images/pxeboot/initrd.img
title Install system with basic video driver
    kernel /images/pxeboot/vmlinuz nomodeset askmethod
    initrd /images/pxeboot/initrd.img
title rescue
    kernel /images/pxeboot/vmlinuz rescue askmethod
    initrd /images/pxeboot/initrd.img
```

把配置文件里的 hiddenmenu 注释掉，前面加个#号注释，保存，然后拔出 U 盘

把目标 U 盘插入计算机，开启计算机时选择从目标 U 盘启动（确保目标计算机是使用 UEFI 启动模式，且开启了 CSM 兼容模式）



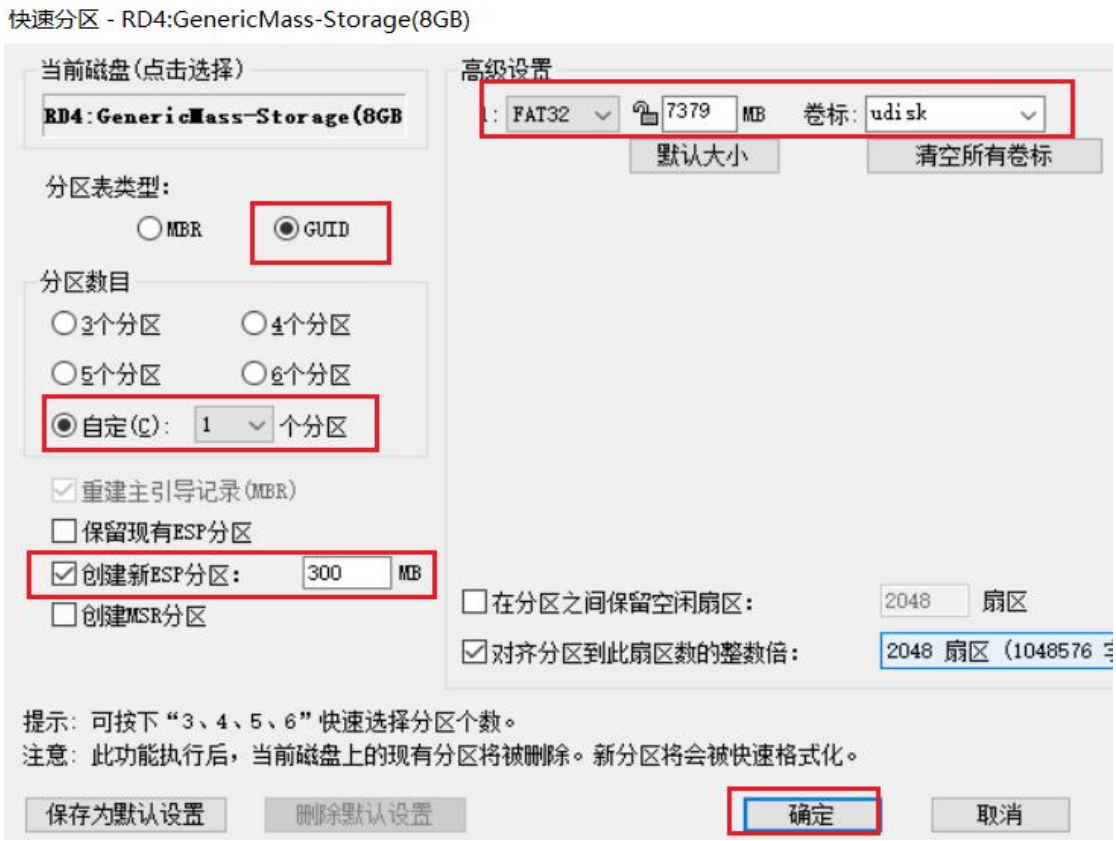


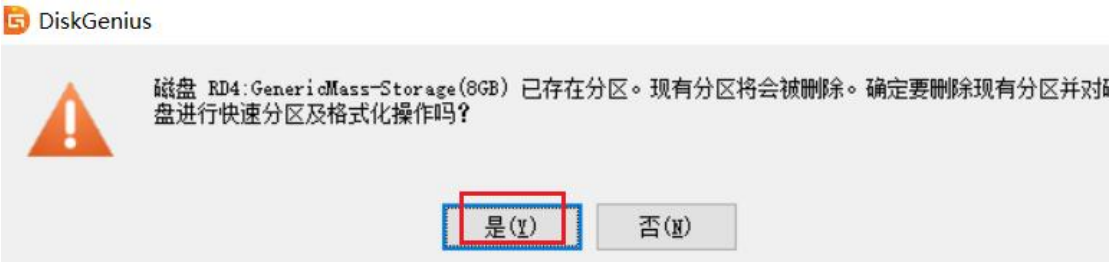
如上图，可见已经成功启动了 grub 引导，且显示了 grub 菜单界面，说明安装成功，uefi 模式下的 grub 引导就是这样安装的，不用写入 MBR 或 PBR 扇区。

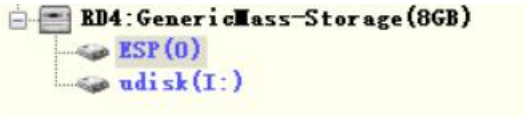
磁盘分区里只要有/EFI/BOOT/bootx64.efi 文件就行

问题来了，有的主板不支持 CSM 兼容模式，无法从使用了 MBR 分区表的 U 盘启动，怎么办？

可以使用磁盘工具把目标 U 盘格式化为 GPT 分区表，并创建 EFI 分区（ESP 分区）使用管理员身份运行 Diskgenius 工具，把目标 U 盘改为 gtp 分区并创建 ESP 分区

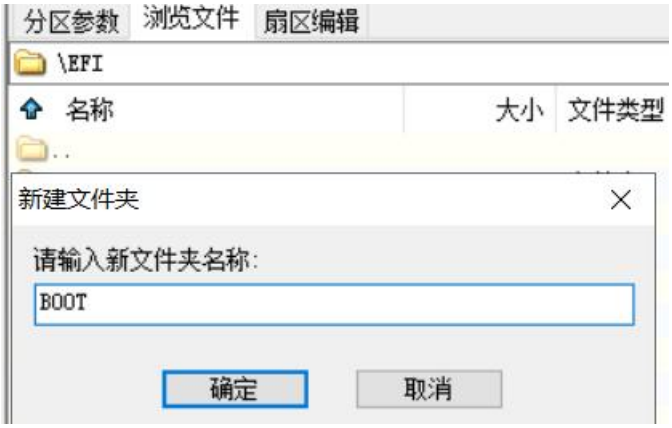




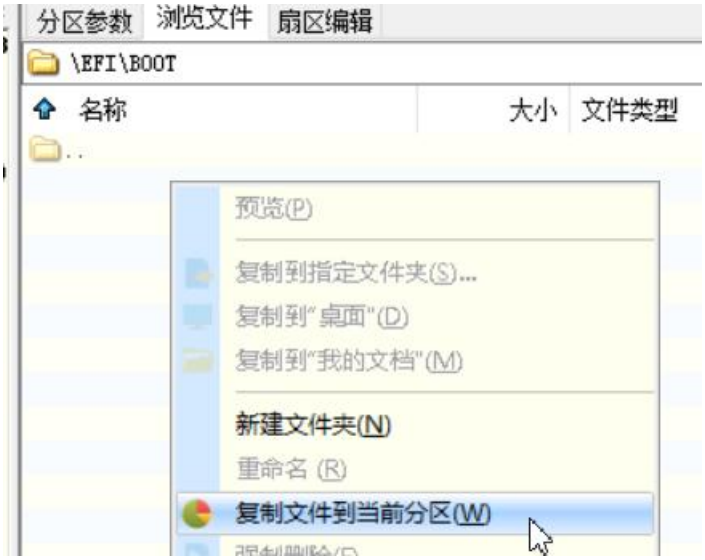


然后该目标 U 盘就有 2 个分区了，一个 ESP 分区，另一个为 FAT32 分区，一般在 windows 里看不到这个 ESP 分区，怎么把引导文件复制到此分区下呢？

可以先把 centos6 安装光盘镜像文件里的 EFI 目录提取到电脑的其他磁盘上，比如 D 盘，然后在 DiskGenius 里进入目标 U 盘的 ESP 分区，选择浏览文件，可见文件为空，可以点击右键，新建文件夹，先创建 EFI 文件夹，进入 EFI 文件夹后再创建 BOOT 文件夹，



再进入\EFI\BOOT 文件夹，点击右键，“复制文件到当前分区”



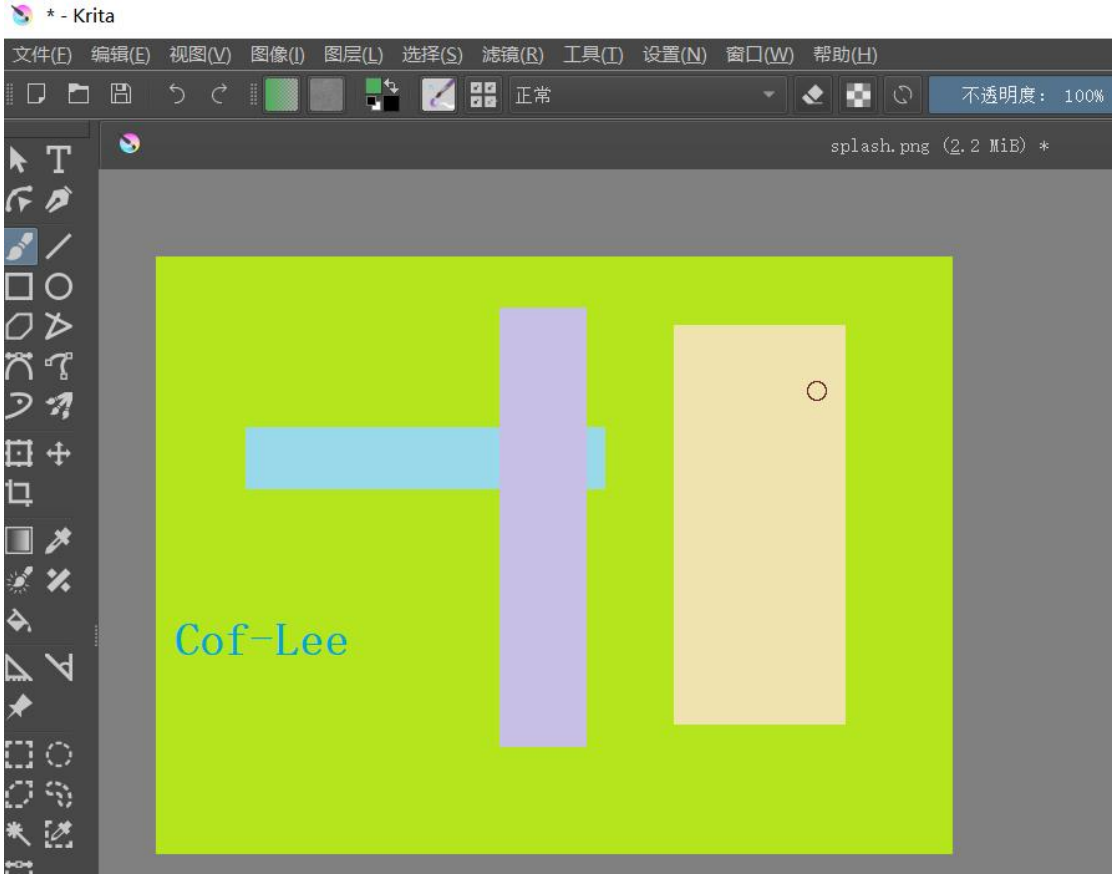
然后选择复制 D 盘里的\EFI\BOOT 目录下的所有文件到 U 盘 ESP 分区的\EFI\BOOT 目录下，

ok 了，拔出 U 盘，插入目标计算机，启动看看
不出意外是可以进入 grub 菜单界面的。

★制作.xpm 图片

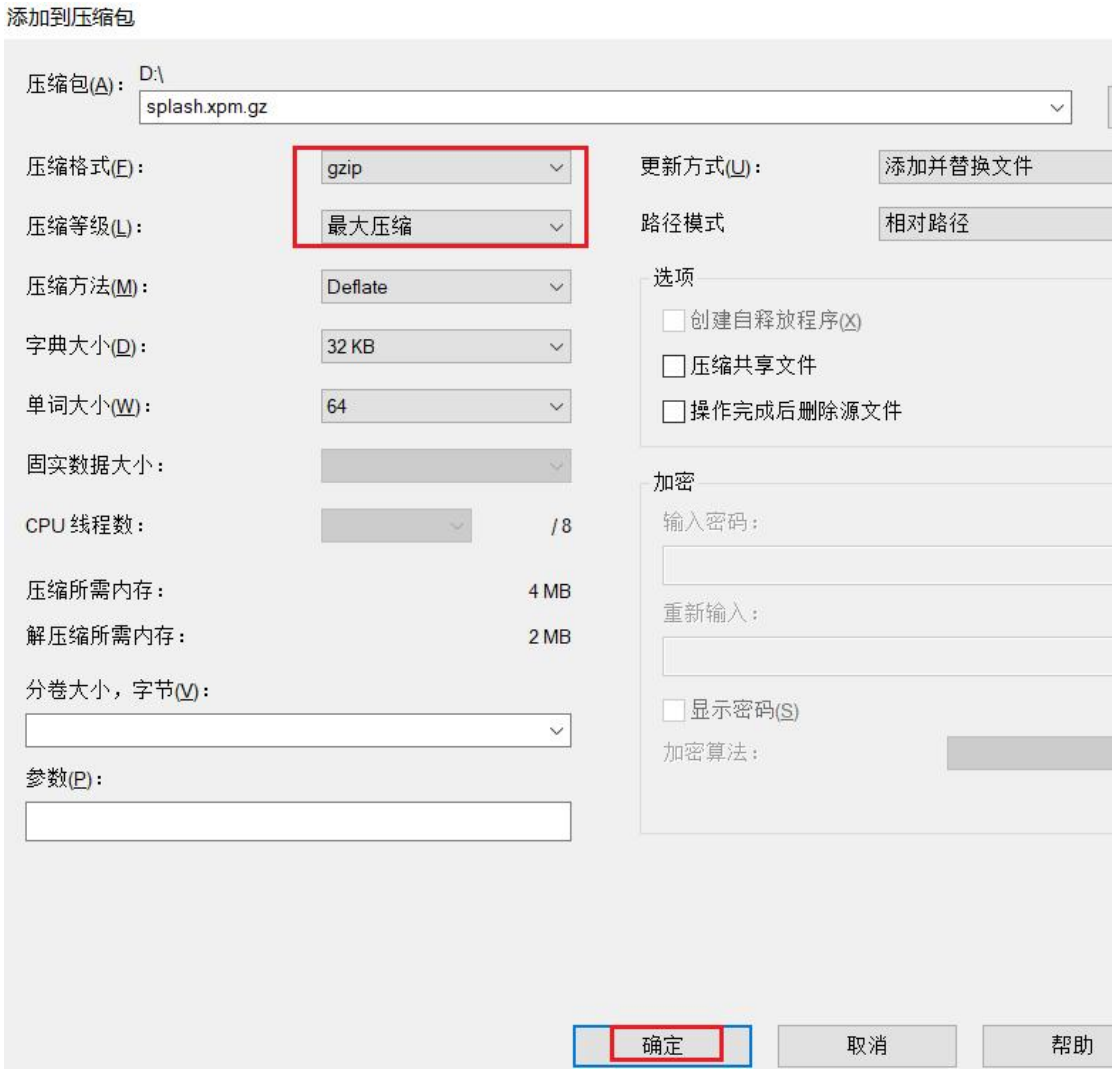
.xpm 格式的图片就是用字符去描述各像素点的信息的文件，一般不常见。如何把常见的 jpg 或 png 转成 xpm 格式呢？
可以在线转换：https://anyconv.com/png-to-xpm-converter/
也可以用程序转换：使用 krita 工具，
https://krita.org/en/download/krita-desktop/

打开 krita 程序，打开目标图片，再导出为.xpm 文件即可

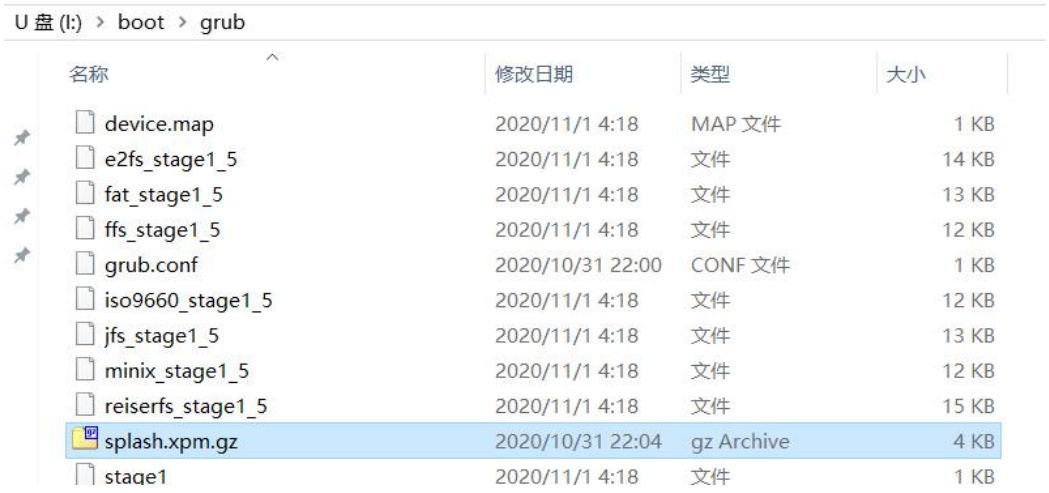


保存到 splash.xpm 文件

转换完毕后，再用压缩工具使用 gzip 压缩将 slpash.xpm 压缩，不压缩的话也行



将 splash.xpm.gz 或 splash.xpm 复制到 grub 的工作目录下



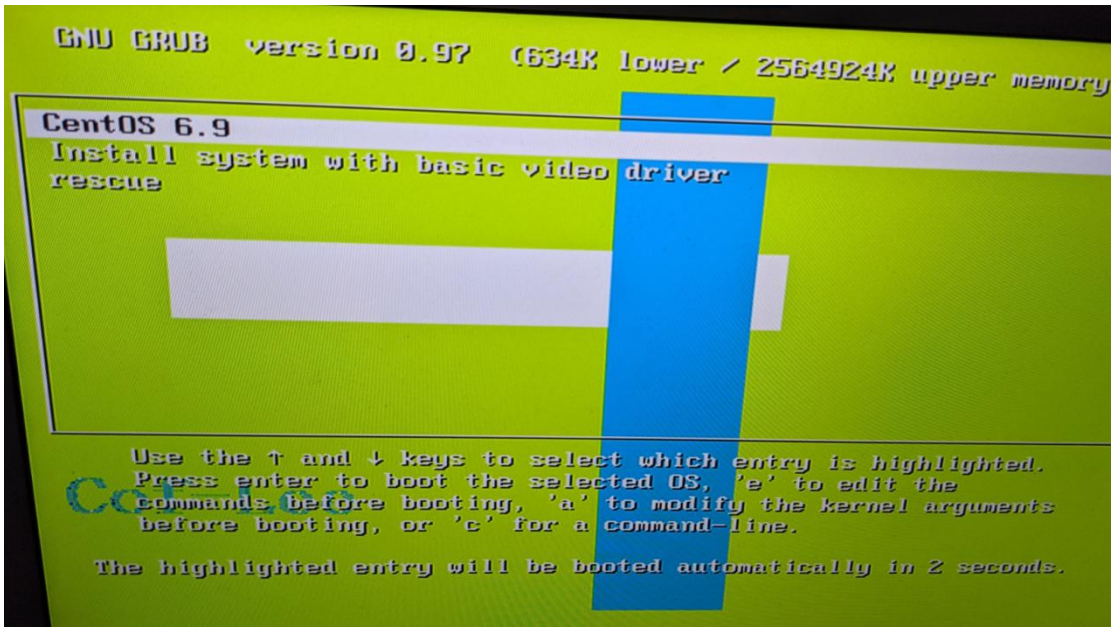
然后看一下 grub.conf 配置，

```
#debug --graphics
default=0
splashimage=/EFI/BOOT/splash.xpm.gz
timeout 5
#hiddenmenu
```

显示的是 splash.xpm.gz，如果不压缩，就改成 splash.xpm 文件也行，

不过，这个 xpm 文件原图不能有太多的颜色，grub 一般只用 14 色的图片，像素为 640x480
所以要想办法把图片的色彩降低，或者自己用画图工具只画纯色图，然后根据启动菜单的效果去调整图片。

把 U 盘插入目标计算机，开机，从 U 盘启动，就能看到有背景图片了：



★使用 grub 引导多个系统

未完成！

grub-mkimage -d x86_64-efi -p /grub2 -c bootconfigpxe.cfg -o grub2x64pxe.efi -O x86_64-efi blocklist boot chain configfile disk echo efi_gop efi_uga efiload exfat extcmd fat halt iso9660 ls minicmd normal ntfs part_gpt part_msdos progress reboot search terminal udf map font gfxterm linux linux16 vhd loopback regexp probe wimboot multiboot multiboot2 test help ext2 net tftp http efinet efi_netfs cat cpio

内置菜单 bootconfigpxe.cfg 内容如下，为了保持文件夹的分类清晰，我放在\boot\grub2\目录下。

```
#用于 pxe 启动的 grub2 内置菜单：
if search --no-floppy -f --set=root /boot/grub2/grub.cfg; then
configfile (tftp)/grub2/grub2pxe.cfg
fi
```

\boot\grub2\grub2pxe.cfg 是 pxe 启动菜单，grub2x64pxe.efi 会自动搜索\boot\grub2\grub2pxe.cfg 菜单文件，也就是说客户机启动流程是接收 http 服务器传来的 grub2x64pxe.efi→加载内置菜单 bootconfigpxe.cfg（已编译进入 grub2x64pxe.efi）→查找外置菜单(tftp)/grub2/grub2pxe.cfg，这里 http 没有初始化，似乎只能用(tftp)，

```
net_bootp
set net_default_server=10.7.20.78
set root='(http)'

menuentry "Win10_16299_PE_x86_x64_10.31.iso" "Win10_16299_PE_x86_x64_10.31.iso" {
map --mem --type=CD (http)/imgs/Win10_16299_PE_x86_x64_10.31.iso
}


menuentry "/imgs/SXWIN10PEX64_17763_NET20191205/boot.wim"
"/imgs/SXWIN10PEX64_17763_NET20191205/boot.wim" {
wimboot @:bootmgfw.efi:(http)/ms/EFI/boot/bootx64.efi              @:bcd:(http)/grub2/wimboot/bcd
@:boot.sdi:(http)/boot.sdi              @:boot.wim:(http)/imgs/SXWIN10PEX64_17763_NET20191205/boot.wim
}


menuentry "1.grubfm_iso_wim_img" --hotkey=1 {
        export grub_file=/boot/imgs
        configfile /boot/grub2/grubfm.cfg
}

menuentry "2.Reboot" --hotkey=2 {reboot}
```

```
menuentry "3.Halt" --hotkey=3 {halt}
```

第 33 章、深入了解 grub2

★grub2 模块

grub2 官网：　　https://www.gnu.org/software/grub/index.html
grub2 下载地址：　https://ftp.gnu.org/gnu/grub/

grub2 的模块有 200 多个，centos7 安装光盘自带的 grub2 为 2.02 版本，安装 grub2-pc-modules.noarch 及 grub2-efi-x64-modules.noarch 后，就会在/usr/lib/grub 目录下生成以下 2 个子目录：

/usr/lib/grub/i386-pc　　#bios 模式下的 grub2 模块
/usr/lib/grub/x86_64-efi　　#uefi 模式下的 grub2 模块

目录里有 grub2 的模块文件及模块相关的描述文件，grub2 模块大致可分为以下几类：

command.lst	命令模块	类似 linux 命令，提供各种不同的基础功能	cat　echo　help　ls　boot　date
crypto.lst	加密模块	提供了各种数据完整性校验与密码算法支持	gcry_crc　crc64　gcry_rsa　gcry_sha1
fs.lst	文件系统模块	提供了访问各种文件系统的功能	btrfs　cpio　exfat　ext2　fat　hfs　iso9660　ntfs　tar　udf　xfs　zfs
partmap.lst	分区模块	提供了识别各种分区格式的功能	part_bsd　part_gpt　part_msdos
terminal.lst	终端模块	提供了各种不同终端的支持	serial　gfxterm　vga_text　at_keyboard
video.lst	视频模块	提供了各种不同的视频模式支持	efi_gop　vga　video_bochs　video_cirrus
moddep.lst			

★grub2 救援模式

grub2 在 BIOS 模式下的启动过程：

①BIOS 预启动程序读取目标磁盘上的 MBR 扇区，执行里面的引导代码（grub2 的第一部分代码：boot.img）
②MBR 里的 grub2 第一部分代码再加载 MBR 扇区后（MBR_gap 或 BPR_gap）的第二阶段的代码：core.img
③第二阶段的代码设置 prefix,root,cmdpath 三个环境变量，识别文件系统并加载/boot 分区里的 normal.mod 模块（及其他依赖的模块）
④执行 normal　$prefix/grub.cfg 命令加载配置菜单

grub2 在 UEFI 模式下的启动过程：

①UEFI 预启动程序读取目标磁盘上的第 1 个 esp 分区里的/EFI/BOOT/BOOTxxx.EFI 文件（core.img）
②BOOTxxx.EFI 程序设置 prefix,root,cmdpath 三个环境变量，加载 normal.mod 模块（及其他依赖的模块）
③执行 normal　$prefix/grub.cfg 命令加载配置菜单

如果以上步骤都成功，则进入普通模式，显示$prefix/grub.cfg 里的菜单项，或者直接进入 GRUB SHELL（未找到'$prefix/grub.cfg'时）

在普通模式中，命令模块 command.lst 与加密模块 crypto.lst 会被自动按需载入（无需使用 insmod 命令），并且可使用完整的 GRUB 脚本功能。其他模块则需要明确使用 insmod 命令来载入。

如果在加载 normal.mod 模块这一步出现故障，则会进入 GRUB2 的救援模式
在救援模式中，GRUB 只自动设置了 cmdpath prefix root 三个环境变量，并且只能使用 insmod ls set unset 四个命令。
只有当额外的模块被加载之后，才可以使用一些其它的命令，进入救援模式可能是因为 GRUB2 没有正确安装。

★grub2 常用命令

在 grub 启动界面按下字母 c 即可进入命令模式

按下 Tab 键可列出所有已加载的命令

```
grub>
Possible commands are:

  . acpi appleloader authenticate badram boot break cat chainloader clear
configfile continue cutmem dump echo exit export extract_entries_configfile
extract_entries_source file gettext halt help initrd initrd16 initrdefi insmod
linux linux16 linuxefi ls lsmod menuentry module multiboot net_add_addr
net_add_dns net_add_route net_bootp net_bootp6 net_del_addr net_del_dns
net_del_route net_get_dhcp_option net_ipv6_autoconf net_ls_addr net_ls_cards
net_ls_dns net_ls_routes net_nslookup normal normal_exit reboot return rmmod
search.file set setparams shift source submenu terminal_input terminal_output
unset
grub>
```

insmod xxx #加载模块，默认从 $prefix/<platform> 这个路径下加载

set 变量名=值 #变量名可有 root,cmdpath,prefix 等

```
grub> insmod zfs
grub>
grub> echo $root
hd0,gpt1
grub>
grub> echo $prefix
(hd0,gpt1)/EFI/grub
grub>
```

ls #列出磁盘设备

ls -l **#列出磁盘设备详细信息；grub2 的磁盘从 0 开始计数，分区从 1 开始计数**

```
grub> ls
(hd0) (hd0,gpt1) (fd0) (fd1) (cd0)
grub>
grub> ls -l
Device hd0: No known filesystem detected - Sector size 512B - Total size
7864320KiB
        Partition hd0,gpt1: Filesystem type fat, UUID ACB3-F02B - Partition
start at 1024KiB - Total size 204800KiB
Device fd0: No known filesystem detected - Sector size 512B - Total size
1440KiB
Device fd1: No known filesystem detected - Sector size 512B - Total size
1440KiB
Device cd0: No known filesystem detected - Sector size 2048B - Total size 2KiB

grub>
```

ls -lh / #列出/根目录下的文件（当前 root 设备的）

ls -lh /efi #列出/efi 目录下的文件

```
grub> ls  -lh  /
DIR         efi/
1.50K       NvVars

grub> ls -lh  /efi
DIR         boot/
DIR         grub/
DIR         grub2/
```

ls (hd0,gpt1)/ #列出目标分区下的文件

```
grub> ls  (hd0,gpt1)/
efi/ NvVars
grub>
grub> ls  (hd0,gpt1)/efi
boot/ grub/ grub2/
```

search.file /filename #查找目标文件所在分区

```
grub> search.file /EFI/grub/grub.cfg
 hd0,gpt1
grub>
```

search.file /filename root #查找目标文件所在分区并设置为当前 root 设备

```
grub> search.file /EFI/grub/grub.cfg  root
grub> echo $root
hd0,gpt1
```

set pager=1 #设置分页，输出内容超过一屏时会暂停输出，按空格后翻下一页，设置 0 表示禁止分页显示

chainloader (hd0,1)+1 #调用目标磁盘分区的第一个扇区内的引导；扇区从 0 开始计数

(hd0,1)+1 完整写法为 (hd0,1)0+1 表示(第 1 个磁盘的第 1 个分区)的 0 扇区开始，+1 表示长度为 1 的连续块

★grub2 设置背景图片及字体

#grub2 目前支持 png,jpeg 格式的图片，需要加载相应图片模块及 gfxterm 相关模块；命令如下

```
insmod  all_video
insmod  gfxterm
insmod  gfxterm_background
insmod  png
insmod  jpeg
set  gfxmode=480x320
terminal_output  gfxterm
background_image  /EFI/grub2/myipxe.jpg
```

#效果如下：

★设置字体

将 pf2 字体文件放到$prefix/fonts/目录下，比如 unicode.pf2

grub 命令：

```
set  gfxterm_font=unicode
insmod  gfxterm
terminal_output  gfxterm
loadfont  unicode
```

★创建 pf2 字体

```
# grub2-mkfont  -i0  -n  serif  -o  serif.pf2  -s24  -v  serif.ttf        #将 ttf 字体转为 pf2；源字体可为 ttc,ttf,otf
```

★安装 grub2 到磁盘上（BIOS 模式）

```
# yum   install   grub2-pc-modules.noarch                #在 centos7 的 DVD 光盘上有；默认已安装
#安装 grub2-pc-modules 软件包后就有 /usr/lib/grub/i386-pc 这个目录

# mkdir   /myboot
# mount   /dev/sdc1   /myboot                          #sdc1 为目标磁盘的 boot 分区
# grub2-install   --directory=/usr/lib/grub/i386-pc   --target=i386-pc   --boot-directory=/myboot   /dev/sdc
```

--directory　　指定源模块所在目录，默认为/usr/lib/grub/<platform>
--target　　指定目标平台，默认是 x86_64-efi，GRUB2 支持的平台有 arm64-efi、i386-pc，x86_64-efi 等
--boot-directory　　指定引导目录，即 boot 分区所挂载的目录
--bootloader-id　　the ID of bootloader.　This option is only available on EFI and Macs.
--efi-directory　　仅当--target 参数设置为 efi 时才需要该参数，指定 EFI 系统分区所挂载的目录
--removable　　仅当--target 参数设置为 efi 时才需要该参数，指定将 EFI 引导程序安装到可移动存储介质上
命令最后的 /dev/sdc 为目标磁盘

#安装完成后，会在 boot 分区创建 grub2 子目录，grub2 下再创建一些必要的文件及子目录

```
[root@localhost ~]# ll /myboot/
total 4
drwxr-xr-x. 5 root root 4096 Oct  5 19:33 grub2
[root@localhost ~]#
[root@localhost ~]# ll /myboot/grub2/
total 32
drwxr-xr-x. 2 root root  4096 Oct  5 19:33 fonts
-rwxr-xr-x. 1 root root  1024 Oct  5 19:33 grubenv
drwxr-xr-x. 2 root root 20480 Oct  5 19:33 i386-pc
drwxr-xr-x. 2 root root  4096 Oct  5 19:33 locale
```

BIOS 模式下，选择从刚刚安装了 grub2 引导的磁盘启动，grub2 会去查找 boot 分区下的 grub2/grub.cfg 配置文件并加载，如果找不到 grub2/grub.cfg 菜单配置文件，则会进入 grub2 的交互界面

```
      Minimal BASH-like line editing is supported. For the first word,
      TAB lists possible command completions. Anywhere else TAB lists
      possible device or file completions.


grub>
grub> _
```

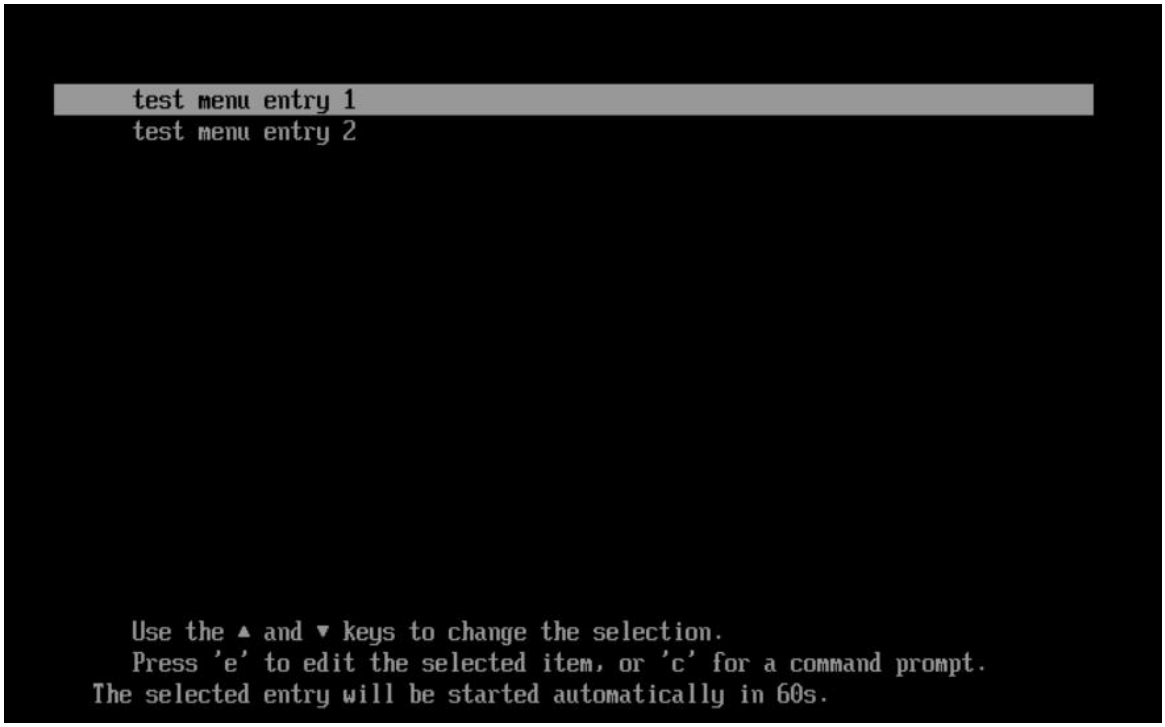
★安装 grub2 到磁盘上（UEFI 模式）

```
# yum  install  grub2-efi-x64-modules.noarch                #在 centos7 的 Everything 光盘上有
#安装 grub2-efi-x64-modules 软件包后就有 /usr/lib/grub/x86_64-efi 这个目录

# yum  install  dosfstools                          #安装此软件包后就有 vfat,fat 等文件系统工具
#将目标磁盘创建一个 efi 分区并格式化为 vfat 文件系统（本例中为/dev/sdc1）
# mkdir  /myefi
# mount  /dev/sdc1  /myefi                          #sdc1 为目标磁盘的 efi 分区
# mkdir  -p  /myefi/EFI/BOOT
# mkdir  -p  /myefi/EFI/grub

#编译 efi 模式下的 grub2 引导程序
# grub2-mkimage  -d  /usr/lib/grub/x86_64-efi  -O  x86_64-efi  -p  /EFI/grub  -o  ./grub2-BOOTx64.EFI  \
part_gpt  part_msdos  part_bsd  part_apple  part_sun  part_sunpc  \
btrfs  cpio  cpio_be  fat  exfat  ext2  ntfs  xfs  iso9660  hfs  hfsplus  udf  ufs1  ufs2  reiserfs  jfs  zfs  \
efi_gop  efi_uga  video_bochs  video_cirrus  gfxterm  gfxterm_menu  gfxterm_background  gfxmenu  \
disk  scsi  appleldr  normal  file  search_fs_file  configfile  chain  linux  multiboot  \
boot  echo  cat  ls  halt  reboot  minicmd  help

-d       指定模块所在目录，默认为/usr/lib/grub/<platform>
-O       定义 Target
-p       prefix，定义配置文件和 mod 文件的父目录，在此目录下有 grub.cfg 文件及 x86_64-efi 模块目录
-c       指定配置文件，这个配置文件会嵌入 efi 文件内；比如 -c  grub2-BOOTx64.cfg
-o       定义编译后生成的引导文件名字

# cp  ./grub2-BOOTx64.EFI  /myefi/EFI/BOOT/BOOTx64.EFI
# cp  -r  /usr/lib/grub/x86_64-efi  /myefi/EFI/grub

# vi  /myefi/EFI/grub/grub.cfg                      #创建菜单配置文件（仅作测试）内容如下
set  timeout=60
menuentry "test menu entry 1" {
        set root='(hd0,gpt2)'
        chainloader /EFI/microsoft/BOOT/bootmgfw.efi
}
menuentry "test menu entry 2" {
        set root='(hd0,gpt2)'
        chainloader /EFI/microsoft/BOOT/bootmgfw.efi
}

# vi  grub2-BOOTx64.cfg             #编写嵌入 grub 引导程序里的菜单文件（会去查找外部配置），内容如下
search.file  /EFI/grub/grub.cfg  root
set  prefix=($root)/EFI/grub
configfile  ($root)/EFI/grub/grub.cfg
#本例中未嵌入此菜单文件
```

tree /myefi/ #查看下 efi 分区里的目录结构

```
[root@localhost ~]# tree /myefi/
/myefi/
└── EFI
    ├── BOOT
    │   └── BOOTx64.EFI
    └── grub
        ├── grub.cfg
        └── x86_64-efi
            ├── acpi.mod
            ├── adler32.mod
            ├── affs.mod
            ├── afs.mod
            ├── ahci.mod
            └── all_video.mod
```

最后将目标磁盘 efi 分区取消挂载，再从目标磁盘 efi 分区启动，效果如下：



第 34 章、PXE 网络启动

请注意 PXE 和 PE 不是一回事。PXE（Preboot eXecution Environment）名为 预启动执行环境，是使用网络接口启动计算机的方式，可以通过网络获取启动文件并执行此启动文件从而进行进一步的引导，所以 PXE 也叫网络启动，它是不用从本地的磁盘获取启动文件的。

PXE 网络启动要用到 2 个角色，一个服务端，一个客户端，服务端是用来存放启动文件的，客户端是要启动的目标计算机，客户端要通过 pxe 相关协议去获取服务端的启动文件并执行该文件。PXE 用到的网络协议主要有 DHCP 和 TFTP

PXE 的大概运作流程是：

启动目标计算机，选择使用 PXE 启动，网络接口通过 DHCP 协议获取 IP 地址和引导文件名，再向指定的服务器获取引导文件，获取引导文件后，把控制权交给它，这样 PXE 的任务就完成了。

因为现在计算机的预启动程序有 2 种，所以通过 PXE 获取到的启动文件也要是适配的，比如在 Legacy BIOS 传统启动模式下，获取到的启动文件是要能在 Legacy BIOS 环境下执行的。EFI 启动模式下获取到的启动文件也是要能在 EFI 环境下执行的。

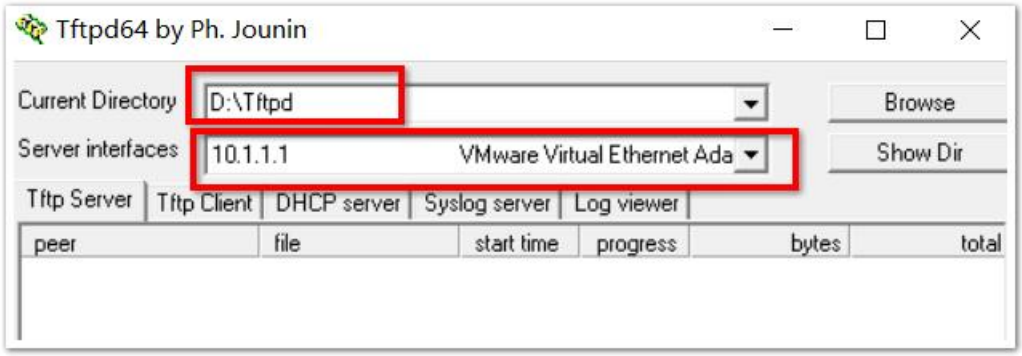
因为是 PXE 网络启动，它至少要用到 2 台计算机，一台做服务端，另一台做客户端，如果没有 2 台电脑的，做实验时可以在自己电脑上安装虚拟机，创建新的虚拟机时，网络模式为“仅主机”模式，且不要使用 CD/DVD 文件，这样它默认就是从网络启动了。

①服务端配置（搭建 DHCP 和 TFTP 服务器）

为了方便，一般 DHCP 和 TFTP 服务器都放在同一台计算机上运行了，本章节使用 Tftpd 软件做实验，官网 https://pjo2.github.io/tftpd64/

下载地址 http://tftpd32.jounin.net/tftpd32_download.html

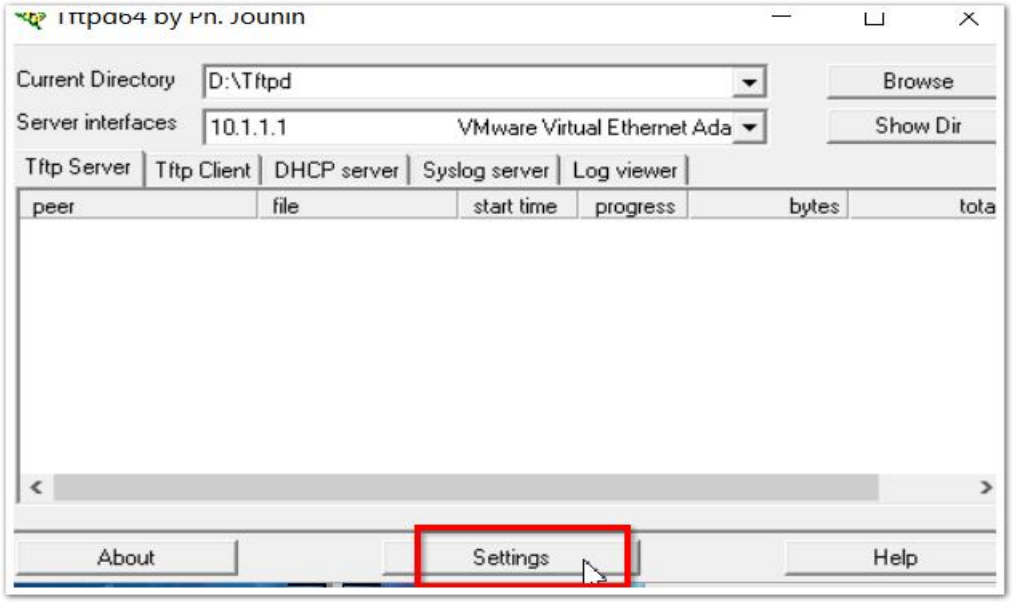
以管理员身份运行 Tftpd64.exe，



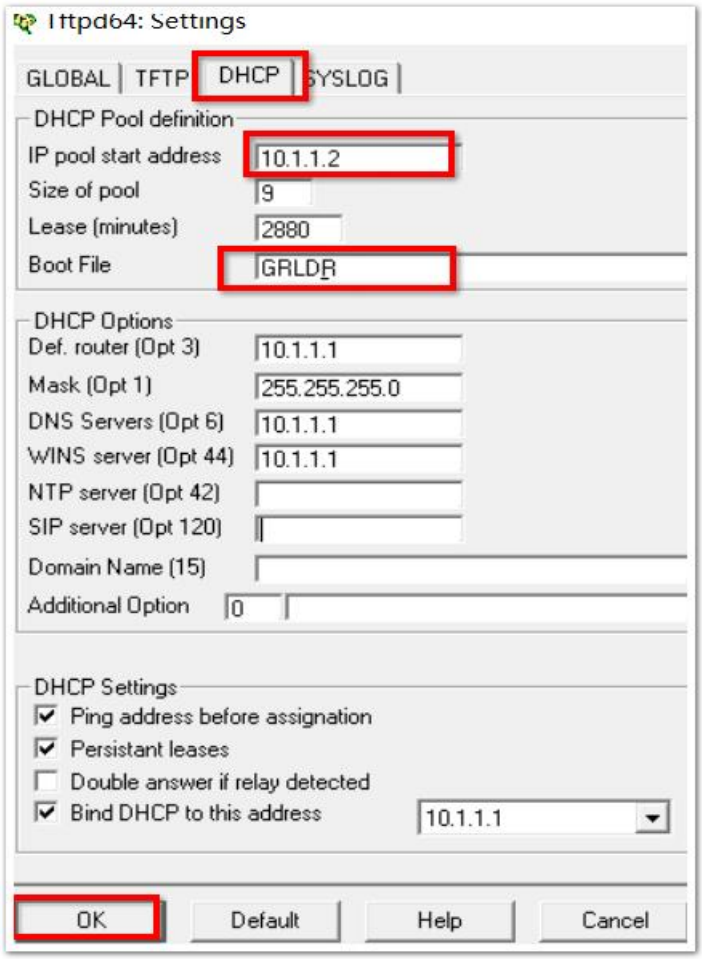
默认的主界面是 TFTP 服务器的界面，设置只有 2 个：

Current Directory 当前目录为 Tftp 服务器的根目录，里面存放启动引导等文件，

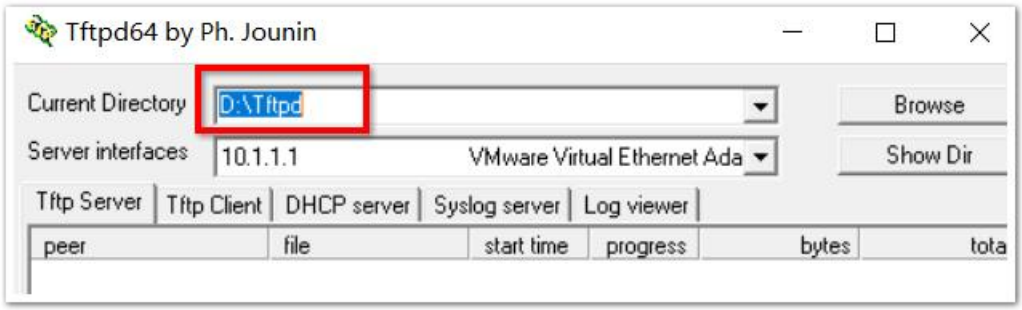
Server interfaces 服务网络接口，因为要使用虚拟机做客户端，所以绑定虚拟机使用到的那个 VMware Virtual 1 网卡（仅主机模式下用到的），该网卡是要自己手动配置 IP 的，比如 10.1.1.1



然后，点击下面的“Settings”设置，进入 DHCP 设置界面：

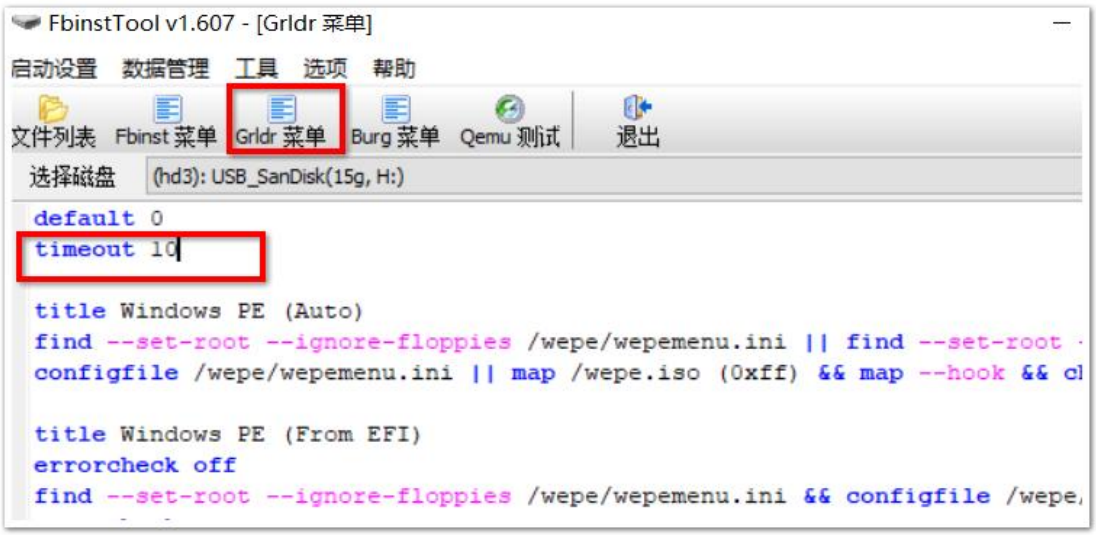


主要是 ip pool 地址池要和绑定的网卡处于同一网段，然后 Boot File 为 PXE 客户端要使用到的启动文件，我们先放个简单的引导文件上去，比如 GRLDR 文件，仅做演示，然后把 GRLDR 文件放到 TFTP 的根目录下（D:\Tftpd 目录）
配置完 DHCP 后，点击左下角的“OK”，返回到主界面，

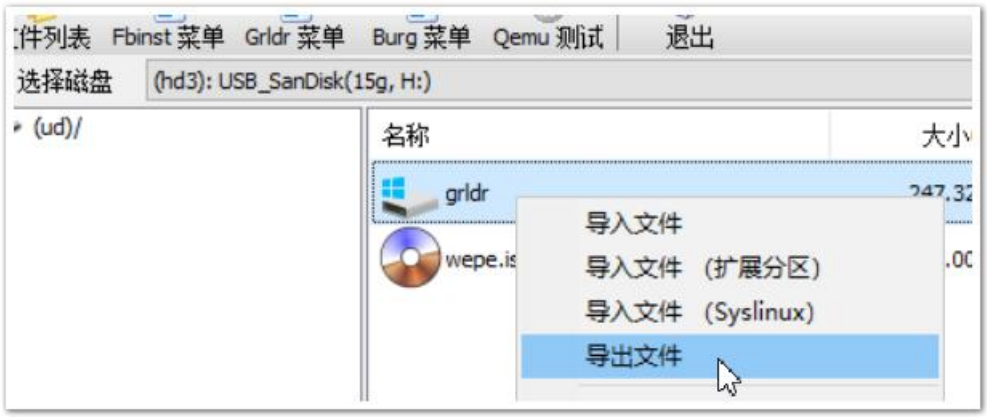


返回到 TFTP 的主界面后，Current Directory **目录会变**，要重设回 D:\Tftpd
然后电脑的**防火墙**要放行 TFTP 服务和 DHCP 服务，不会配置的，可以**暂时关闭**防火墙。

最后记得把 Grub 的引导文件 GRLDR 放到 D:\Tftpd，这个 GRLDR 文件从哪儿来？可以从 PE 启动盘（使用 wepe 或老毛桃制作的启动盘）里用 FbinstTool 导出，也可从 grub4dos 里复制。



先修改 Grldr 菜单，把 timeout 默认时间改大一点（比如 10 或 20 秒）方便在 PXE 启动时查看效果。修改完，保存，再导出 grldr 文件到 D:\Tftpd

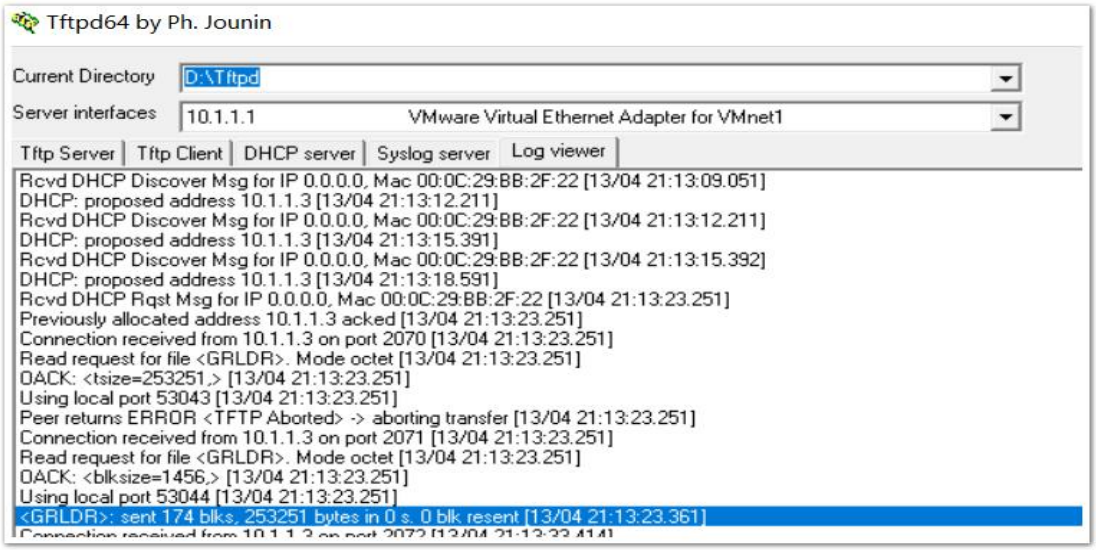


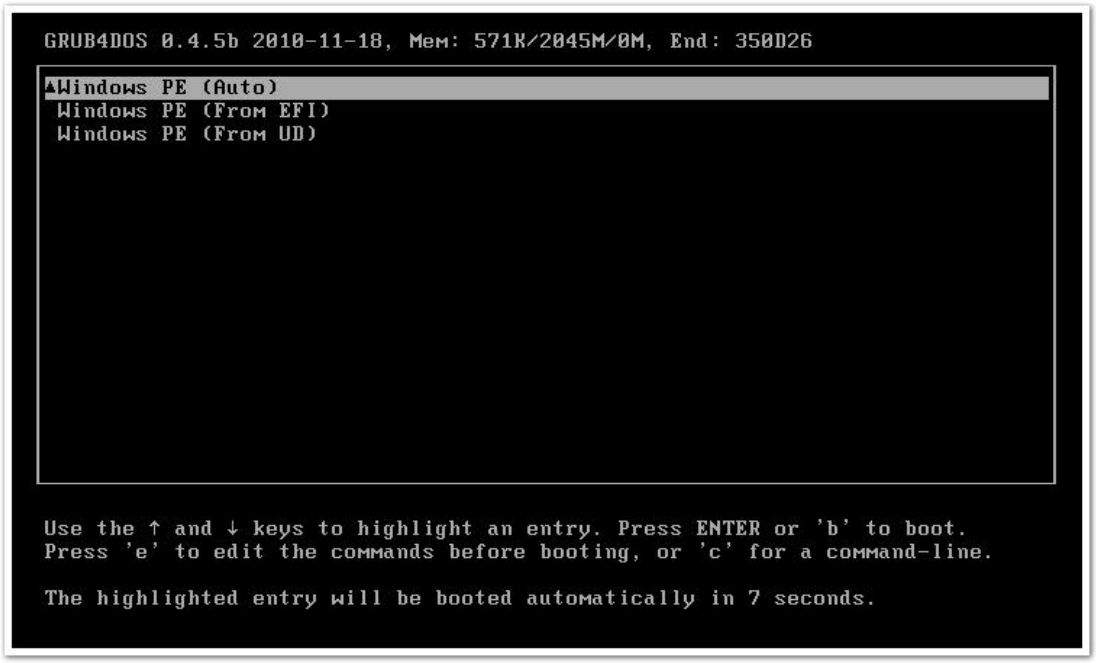
导出文件名为 GRLDR，保存到 D:\Tftpd

②客户端的创建（使用 vmware 虚拟）

虚拟机配置如上，没有使用 CD/DVD，网络模式为仅主机模式（和 vmware net 1 在同一网络环境下），所以物理机上的 DHCP 服务器能给这个虚拟机分配 IP，在没有 CD/DVD 光盘，新的磁盘里又没有启动的系统的情况下，虚拟机只能从 PXE 网络启动了，我们开启此虚拟机看看

上图出现 DHCP..-x 字样，说明它在使用 PXE 了，先是启用 DHCP 客户端获取相关配置，过几秒，查看 Tftpd 的 log viewer 日志，可见客户端获取到 ip 地址（10.1.1.3）和 GRLDR 引导文件了，





虚拟机执行此引导文件后，出现了 grub 引导菜单界面，如上图，说明 pxe 启动成功，pxe 的任务完成了，

过了 timeout 默认时间，会出现上图的界面，因为 GRLDR 引导获取不到默认菜单指定的文件，这个没关系，这是 grldr 的任务，跟 PXE 无关了。

小结：

通过本章的实验，初步了解了 PXE 网络启动的原理及过程。PXE 网络启动就是在目标计算机启动时，选择从 PXE 网络启动，然后目标计算机的网卡会发出 DHCP 请求，获取 ip 地址及引导文件等相关配置，然后去向指定的服务器获取指定的引导文件，

服务端要提供相关的文件，并通过 tftp 服务把文件传送给客户端，客户端获取到引导文件后，把控制权交给引导文件，PXE 的任务就完成了，接下来的工作就是那个引导文件的事了。

要注意的是网络上这 2 台计算机要能二层互通。相关的网络知识（tftp 和 dhcp）可以去深入地学习一下。

第 35 章、PXE 网络启动之 ipxe

首先要明白 PXE 网络启动和 BIOS 及 UEFI 不是并列的关系，PXE 启动和 U 盘启动/光盘启动/硬盘启动等才是并列的关系。PXE 只是获取引导文件的一种途径。

在 Legacy BIOS 模式下，无论是从 U 盘/光盘/PXE 网络/硬盘启动都要求其引导文件能在此环境（BIOS）下运行，在 UEFI 模式下，也是同样的道理，要求引导文件能在此环境（UEFI）下运行。

使用 PXE 网络启动，我们可以通过网络获取引导文件，当目标计算机的硬盘上的引导文件丢失时，可以通过 PXE 救急，把引导文件传输过去，那么为什么不用 U 盘呢？可能有时不方便用 U 盘，或者 U 盘不在身边，所以用网络去传输也是一种不错的方法。当然 PXE 网络启动也可以搭建无盘计算机，网络中所有的客户机都不用硬盘，通过网络启动获取引导及系统文件（比如网吧的无盘系统），也可以用 PXE 进行批量安装系统，这样就不用在每台计算机上插入安装介质了。

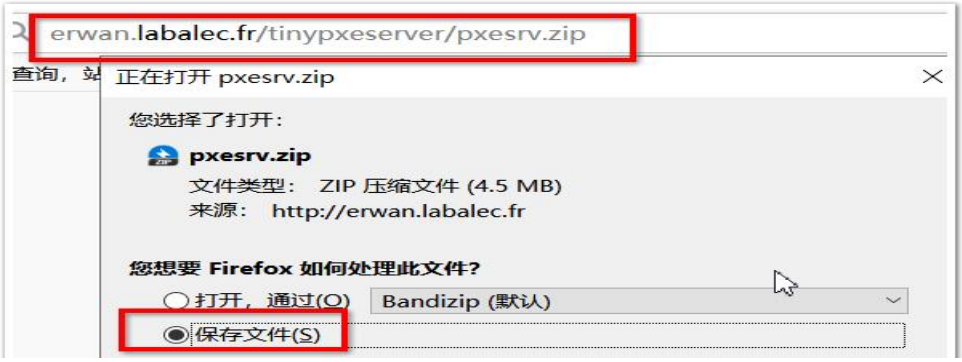
在 PXE 网络启动时，我们传输过去的引导文件要求能够支持网络传输文件的功能，这个怎么理解呢？首先，引导文件是通过 PXE 网络传输过去的，当 PXE 把控制权交给引导文件时，它的任务就完成了，这时引导文件要再想引导其他系统，就得获取其他系统的文件，怎么获取呢？通过 PXE？不是的，PXE 的任务完成了，它不管这个引导了，所以传输后续文件的任务得由那个引导文件去完成，所以这个引导文件（或者说引导程序）也得支持网络传输的功能。

哪些引导程序支持网络驱动并通过网络获取文件呢？一般在 windows 和 linux 系统安装光盘里有支持 pxe 的引导，在服务端的 DHCP 参数里设置引导文件名称，然后把安装光盘解压到 Tftpd 根目录下，这样目标计算机启动时能获取到引导文件，而这个引导文件又自带网络驱动，并传输接下来要用到的安装文件，完成操作系统的安装。不过，一般这些引导程序仍然是使用 tftp 协议去传输文件，安装操作系统要用到的文件非常多，小则几百 MB，大则几个 GB，所以用 Tftp 协议去传输，恐怕得等好几个小时，有时还会出错，这就麻烦了。所以最好是设置传输文件的方式为 FTP 或者 http 之类较快的传输协议。

本章节先不讲使用安装光盘里自带的支持网络传输的引导程序了，推荐一个其他的开源的支持使用 http 协议快速传输文件的引导工具：ipxe

因为 ipxe 支持 http 协议去获取其他文件，所以我们在服务端计算机上要安装 http 服务器，又要有之前 pxe 用到的 dhcp 和 tftp，可以安装 dhcp，tftp，http 三个服务（初学者安装这 3 个服务可能有一定的难度），本章我们为了方便，不必安装 3 种服务器，可以使用 Tiny PXE server 这个服务端软件（开源免费的），它融合了 tftp 和 dhcp 及 http 等服务。且 Tiny PXe 也自带了 ipxe 这个引导程序。

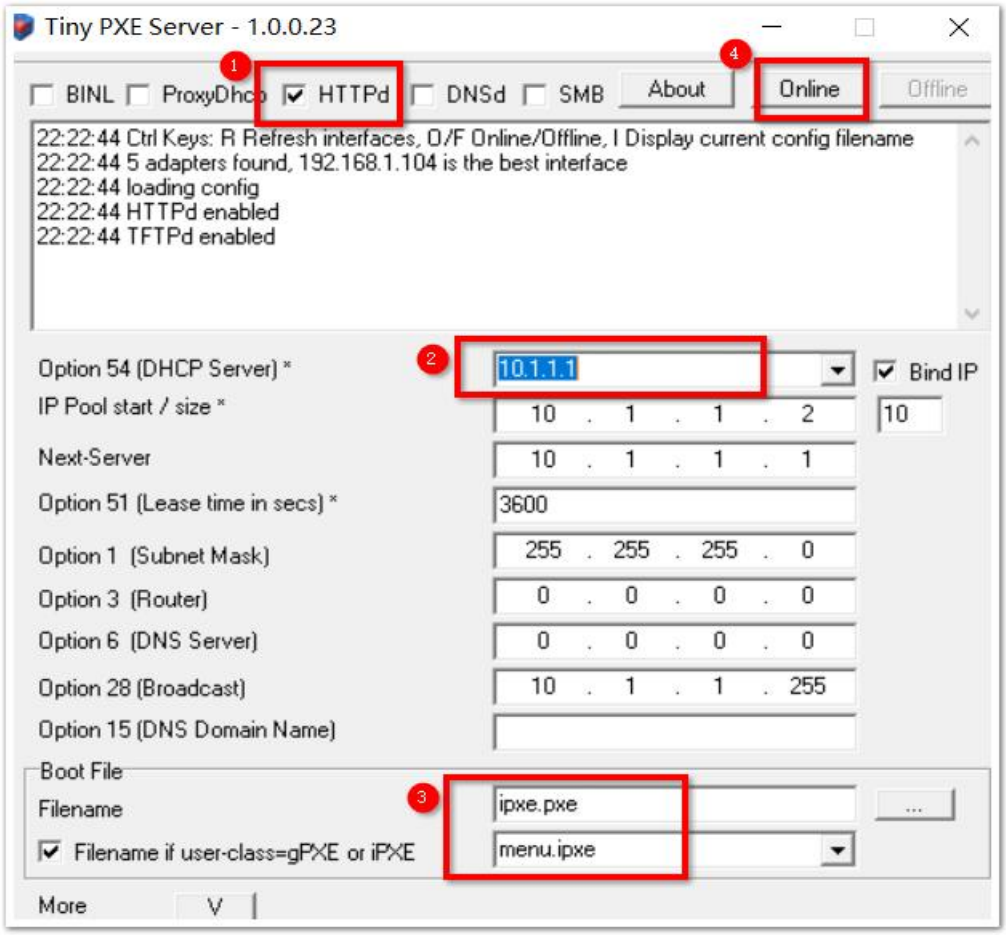
Tiny PXE server 下载地址：http://erwan.labalec.fr/tinypxeserver/pxesrv.zip



下载到 D 盘，然后解压到 TinyPXEserv1.0.0.23 文件夹里，里面那个 pxesrv.exe 就是 Tiny PXE server 程序了，以管理员身份运行它

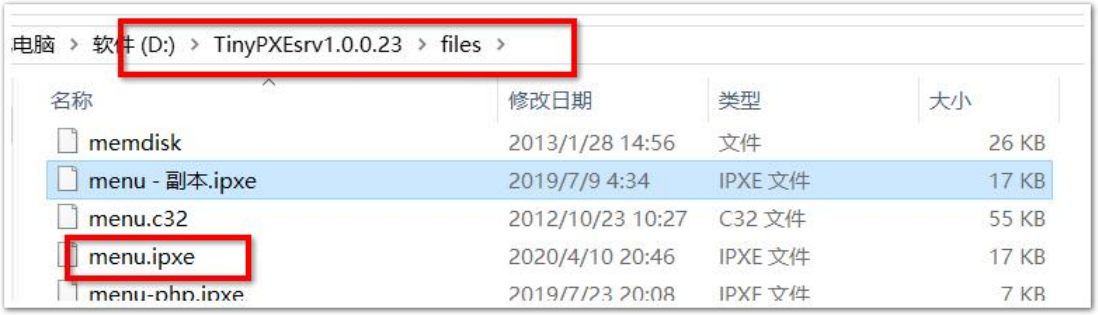
我们可以通过 PXE 启动获取到 ipxe 引导，再由 ipxe 去做进一步的工作，比如通过网络获取 windows PE 系统文件并进入此 PE 系统。

以管理员身份运行 pxesrv.exe，它的 tftp 以及 http 服务的根目录为 D:\files 文件夹，



首先绑定和客户端位于同一网络的网卡 IP，再设置启动文件名为 ipxe.exe，（注意，Tiny PXE Server 提供的 tftp/http 的根目录就变为指定的启动文件 ipxe.exe 所处的目录了，本例中为 D:TinyPXEsrv1.0.0.23\files\）最后点击主界面右上角的 Online 按钮。这样服务就启动了，

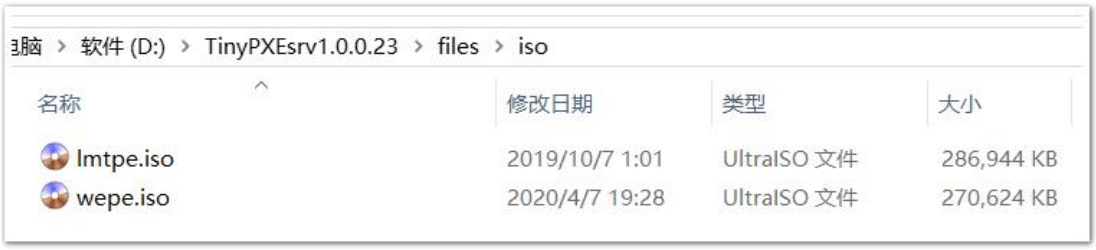
我们要先修改一下 ipxe 的菜单文件，只写一个简单的菜单，比如：获取 wepe.iso 文件，并启动此 PE 系统，



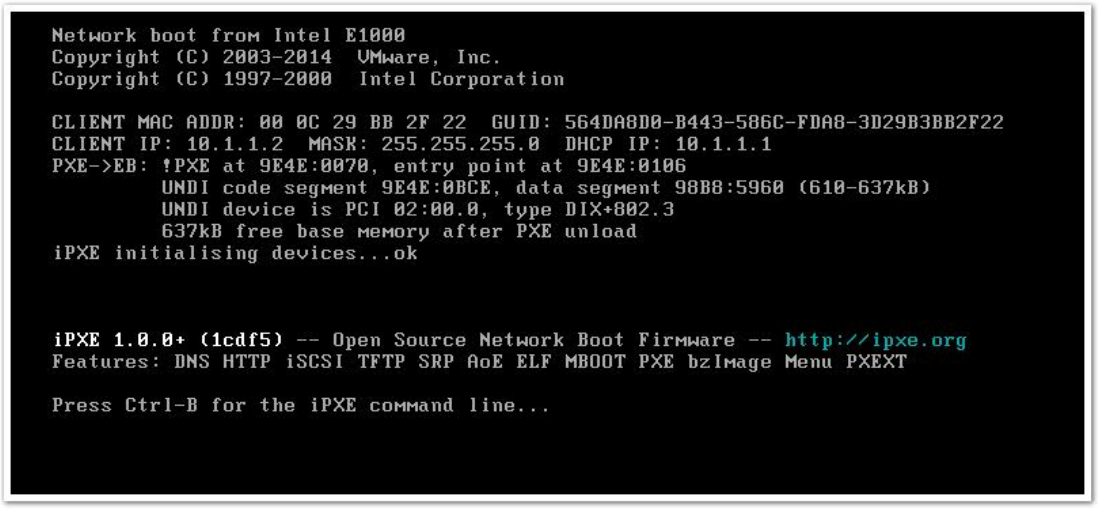
ipxe 的菜单文件名为 menu.ipxe，先备份一份，再修改它，把它里面的配置全删了，按照下面的写：

```
#!ipxe
set boot-url http://${next-server}
set menu-timeout 5000

:start
menu Welcome to iPXE's Boot Menu
item --gap -- ------------------------- Utilities autoexec.ipxe ---------------------
item wepe wepe_iso grub boot
item lmtpe lmtpe_iso grub boot

item --gap -- ------------------------------ Advanced -------------------------------
item reboot         Reboot
item exit           Exit (boot local disk)
choose --default exit --timeout 30000 target && goto ${target}

:wepe
initrd http://${next-server}/iso/wepe.iso
chain http://${next-server}/grub.exe --config-file="map (rd)+1 (0xff);map --hook;chainloader (0xff)"

:lmtpe
initrd http://${next-server}/iso/lmtpe.iso
chain http://${next-server}/grub.exe --config-file="map (rd)+1 (0xff);map --hook;chainloader (0xff)"

:reboot
reboot

:exit
exit
```

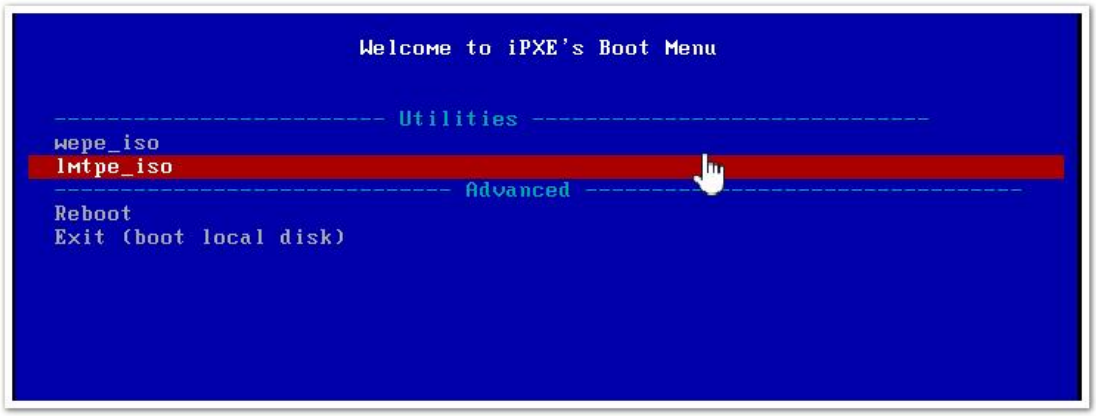
然后把 wepe.iso 文件和 lmtpe.iso 文件复制一份到 Tinypxe 的服务根目录 D:TinyPXEsrv1.0.0.23\files\iso 下，如果没有 iso 文件夹，可以先创建，



最后可以启动客户机了，选择从 PXE 网络启动（bios 模式）



如上图，说明 PXE 已经获取到 ipxe.exe 文件并执行此程序了，上图为 ipxe.exe 的界面，它启动后再启动网络驱动并通过 http 协议去获取 menu.ipxe 菜单文件，展示菜单如下图：

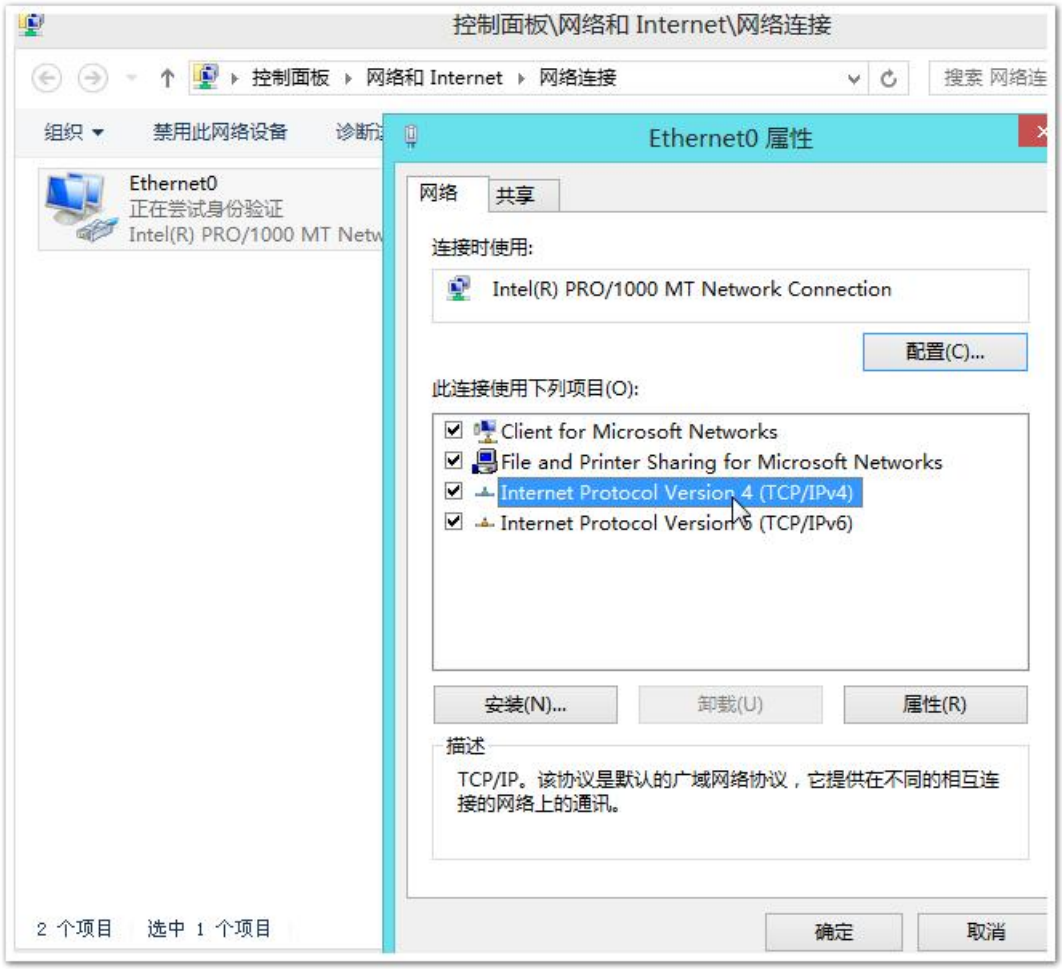


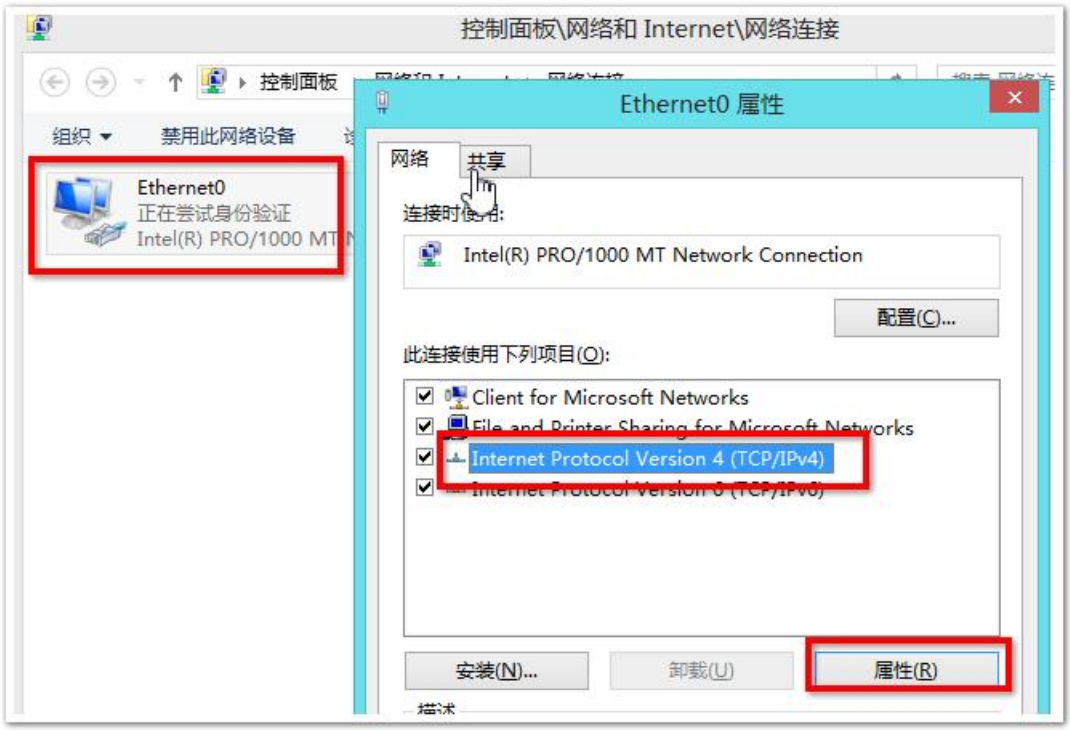
我们选择 lmtpe.iso 这个菜单，回车，ipxe 程序再通过 http 协议去下载此 pe 的镜像文件，如下图，速度比 tftp 快多了，

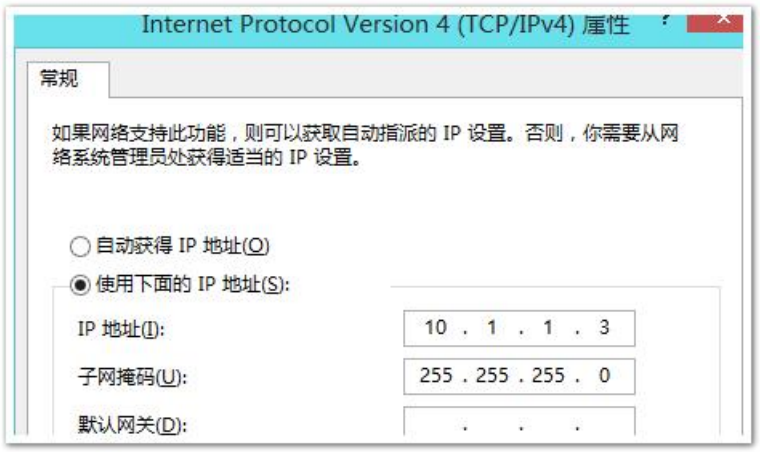
等下载完成后，它根据菜单里的指示，用 grub.exe 去运行此 lmtpe.iso 镜像文件，从而进入 PE 系统，建议选择使用支持网络的 PE 系统，

老毛桃 PE 是支持网络的，用它就可以再进行其他的操作了，比如使用 DiskGenius 磁盘精灵去完成目标计算机的硬盘的分区，以及使用 winNTsetup.exe 去安装 windows 系统，安装的 windows iso 镜像文件可以通过网络去传输。

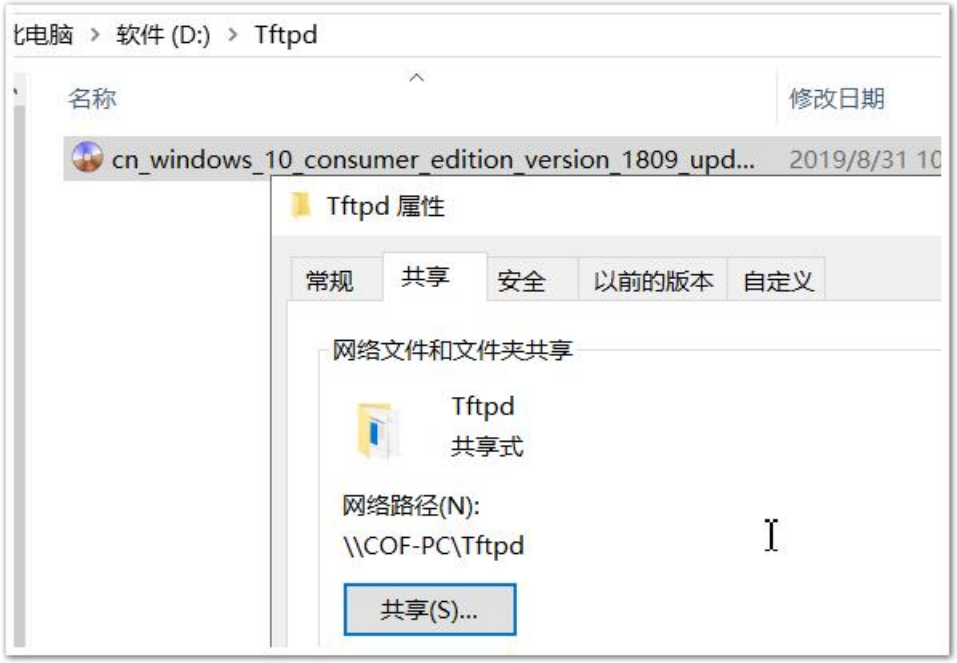
先配置老毛桃 PE 系统的 IP 地址，再把服务端的共享目录（存放有 winodws 安装光盘文件的）挂载到 K 盘，


控制面板\网络和 Internet\网络连接
控制面板 ▸ 网络和 Internet ▸ 网络连接
搜索 网络连
组织
禁用此网络设备
Ethernet0
正在尝试身份验证
Intel(R) PRO/1000 MT Netw
Ethernet0 属性
网络
共享
连接时使用:
Intel(R) PRO/1000 MT Network Connection
配置(C)...
此连接使用下列项目(O):
Client for Microsoft Networks
File and Printer Sharing for Microsoft Networks
Internet Protocol Version 4 (TCP/IPv4)
Internet Protocol Version 6 (TCP/IPv6)
安装(N)...
卸载(U)
属性(R)
描述
TCP/IP。该协议是默认的广域网络协议，它提供在不同的相互连接的网络上的通讯。
2 个项目
选中 1 个项目
确定
取消

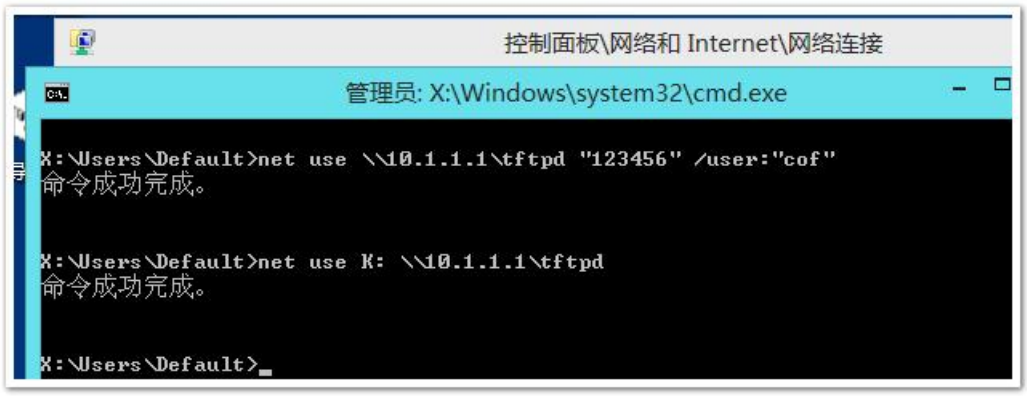

控制面板\网络和 Internet\网络连接
控制面板
组织
禁用此网络设备
Ethernet0
正在尝试身份验证
Intel(R) PRO/1000 MT
Ethernet0 属性
网络
共享
Intel(R) PRO/1000 MT Network Connection
配置(C)...
此连接使用下列项目(O):
Client for Microsoft Networks
Internet Protocol Version 4 (TCP/IPv4)
安装(N)...
卸载(U)
属性(R)

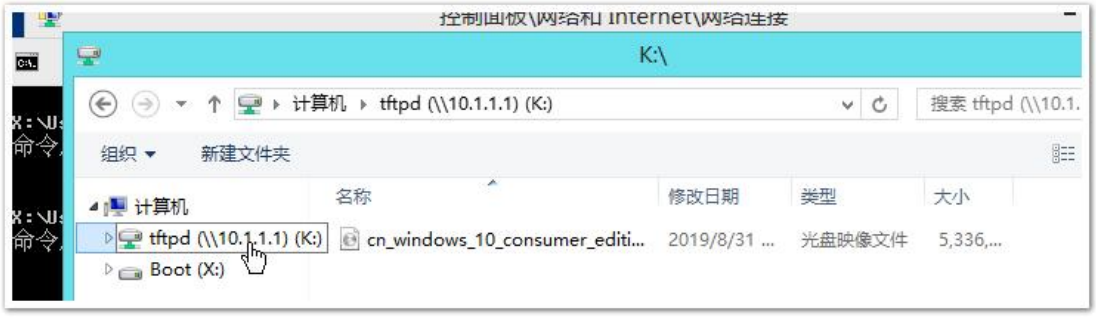


为什么要手动配置 PE 系统的 IP 地址，因为可能网络中有其他的 dhcp 服务器的干扰，手动确认一下比较好。
配置和 dhcp 服务（Tiny PXE server）同一网段的 IP 就行了，再使用命令挂载服务端计算机上的共享目录，共享目录如下：



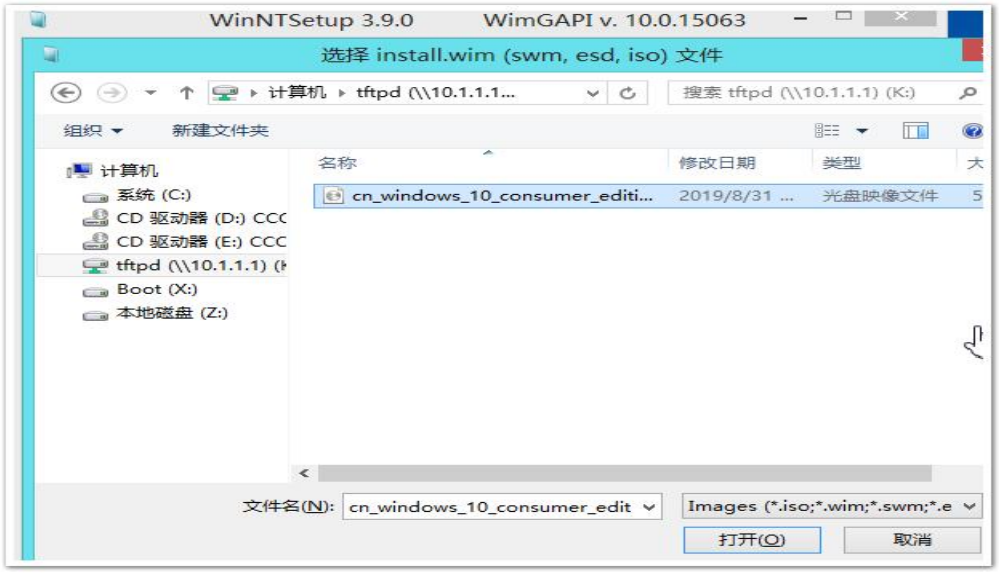
在 PE 系统上操作如下：（cof 为服务端计算机的用户名，密码为 123456）





然后在 PE 里打开文件管理器，可见已经映射共享目录到 PE 系统里了，盘符为 K:

接下来分区完成后，就可以使用 winNTsetup.exe 安装系统了，详细步骤请参考前面章节



小结：

PXE 网络启动时，在服务端使用 Tiny PXE server 服务器，它只是 tftp, dhcp, http 等服务的集合体程序，由它完成引导文件的传输，在 dhcp 设置里指定使用 ipxe.exe 这个引导程序。

客户端以 bios 模式启动后通过 PXE 启动，获取 dhcp 配置并根据配置里的 boot file，使用 tftp 协议获取 ipxe.exe 文件并执行它，ipxe 再根据它自己的配置去获取下一步的引导文件。ipxe 是支持网络驱动以及使用 http 协议传送文件的，所以速率会快很多，再通过 ipxe 的菜单，指定要传送 xxPE.iso 文件，再启动此 PE 系统，接下来就可以在 PE 系统里完成其他任务。

第 36 章、深入了解 ipxe

ipxe 是一个开源的引导程序，支持从网络加载其他文件进行下一步的引导，
官网 https://ipxe.org/
如果目标计算机支持从 PXE 网络启动，则直接使用 pxe 加载 ipxe 引导
bios 模式使用 http://boot.ipxe.org/undionly.kpxe 文件
uefi 模式使用 http://boot.ipxe.org/ipxe.efi 文件
如果需要获取源码，进行自定义编译，则源码获取地址： https://github.com/ipxe/ipxe

★ipxe 启动流程：
①pxe 目标设备启动后，选择从网络启动，目标网卡通过 dhcp 获取 ip 及启动文件等配置信息
②目标设备通过 tftp 下载 dhcp 指定的 ipxe 引导文件（pxe 默认使用 tftp 下载文件），并将控制权交给 ipxe
③ipxe 会再次初始化网卡，重新执行 dhcp 客户端流程，在进行 dhcp 请求时会带上 option 93（Client-Arch，字段类型为 uint16）和 option 77（User-Class，字段类型为字符串）
option 93 = 0x0000 表示 IA x86 PC
option 93 = 0x0007 表示 EFI x64 （值为网络字节序）
Client-Arch 的值在 ipxe 里显示为 pcbios 或 efi；User-Class 值固定为"iPXE"
④dhcp server 发现请求中带有 User-Class 参数且值为 iPXE 时，返回特制的启动文件（文件名可自由指定）
⑤ipxe 通过 tftp 获取特制的启动文件（比如 menu.ipxe 菜单配置文件），再读取此菜单配置文件，展示菜单项，最后根据用户选择的菜单项进行下一步引导

如果 dhcp server 不识别 option 77 User-Class，则启动文件一直是默认的 ipxe 引导文件，而不是菜单文件，导致 ipxe 过程进入几次循环，最后 ipxe 启动失败，设备重启

★使用 ipxe 引导有 2 种方法：
一种是要在 dhcp server 做 option 77（user-class）配置，指定 ipxe 的启动菜单文件
另一种是获取 ipxe 源代码，编译时内嵌菜单文件，内置的菜单文件里可二次链接到外部菜单文件

①option 77 user-class

/etc/dhcp/dhcpd.conf 配置部分内容如下（官方指导链接 https://ipxe.org/howto/dhcpd）

option client-architecture code 93 = unsigned integer 16;	定义 client-architecture 为 dhcp option 93，且数据类型为 uint16 ；要写在 subnet {}这段之外
subnet 10.99.1.0 netmask 255.255.255.0 {	
option routers 10.99.1.1;	分配给客户端的网关
option subnet-mask 255.255.255.0;	分配给客户端的子网掩码
option domain-name-servers 8.8.8.8,114.114.114.114;	给客户端的 dns 地址
option domain-name "cof-lee.com";	
range dynamic-bootp 10.99.1.100 10.99.1.200;	分配给客户端的 ip 范围
next-server 10.99.1.1;	tftp 及 dhcp 服务器地址，启动文件从这个地址获取
if exists user-class and option user-class = "iPXE" {	
filename "http://192.168.1.11/menu.ipxe";	如果匹配到有 user-class 选项且值为 iPXE，则使用此文件（这个文件是 iPXE 的菜单配置文件）
} elsif option client-architecture = 00:00 {	当 dhcp option 93 的值为 00:00 时，表示 pcbios

filename "undionly.kpxe";	如果客户端平台为 pcbios，则使用此引导文件（bios 模式下的 ipxe 引导文件）
} else {	client-architecture != 00:00 表示 efi
filename "ipxe.efi";	使用此引导文件（efi 模式下的 ipxe 引导文件）
}	
}	

②重新编译 ipxe（内嵌菜单配置）

源码获取地址： https://github.com/ipxe/ipxe

#以下是在 centos7 x86 系统下的编译

yum install -y gcc mtools binutils make perl xz-devel xorriso syslinux unzip

unzip ipxe-master.zip #解压源码包

cd ipxe-master/src/ #进入解压目录的 src/子目录下

vi default-bios.ipxe #创建一个菜单配置文件，名称自定义

```
#!ipxe
dhcp
chain tftp://${next-server}/menu-bios.ipxe
```

make bin/undionly.kpxe EMBED=default-bios.ipxe #编译 ipxe 引导文件，并将菜单配置文件内嵌进去

#默认生成的引导文件为此路径下的 bin/undionly.kpxe，可复制到其他地方并重命名

cp bin/undionly.kpxe ./undionly-bios-tftp.kpxe

vi default-uefi.ipxe

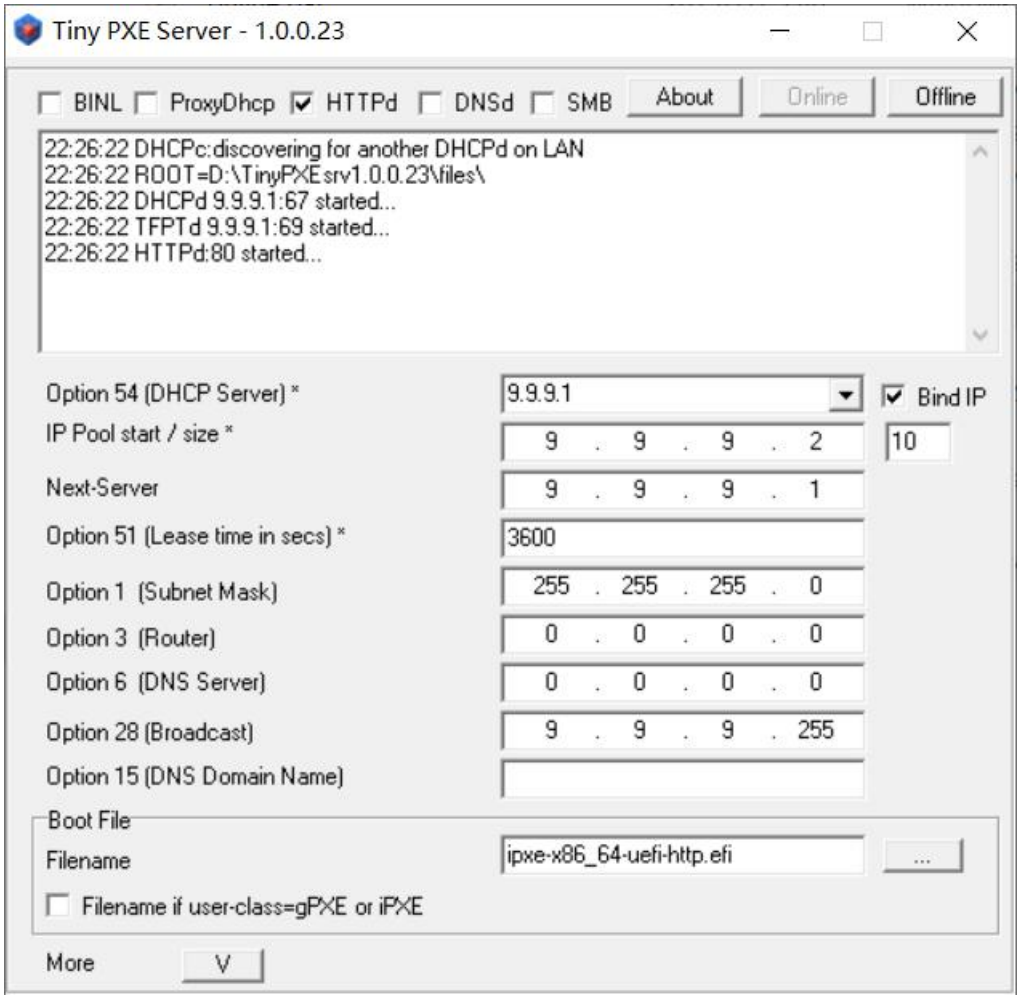
```
#!ipxe
dhcp
chain http://${next-server}/menu-uefi.ipxe
```

make bin-x86_64-efi/ipxe.efi EMBED=default-uefi.ipxe #编译 ipxe 引导文件（efi 模式下的）

#默认生成的引导文件为此路径下的 bin-x86_64-efi/ipxe.efi，可复制到其他地方并重命名

cp bin-x86_64-efi/ipxe.efi ./ipxe-x86_64-uefi-http.efi

再在 dhcp server 里指定 bootfile 为刚刚编译的文件



 # uefi 启动下的日志：

22:26:22 DHCPc:discovering for another DHCPd on LAN
22:26:22 ROOT=D:\TinyPXEsrv1.0.0.23\files\
22:26:22 DHCPd 9.9.9.1:67 started...
22:26:22 TFPTd 9.9.9.1:69 started...
22:26:22 HTTPd:80 started...

22:30:01 DHCPd:DISCOVER received, MAC:00-E0-70-CD-F5-73, XID:8836F425
22:30:01 DHCPd:OFFER sent, IP:9.9.9.2, XID:8836F425
22:30:04 DHCPd:REQUEST received, MAC:00-E0-70-CD-F5-73, XID:8836F425
22:30:04 DHCPd:ACK sent, IP:9.9.9.2, XID:8836F425
22:30:04 TFTPd:DoReadFile:ipxe-x86_64-uefi-http.efi B:1468 T:0
22:30:04 TFTPd:DoReadFile:autoexec.ipxe B:0 T:887
22:30:04 TFTPd:DoReadFile:autoexec.ipxe B:0 T:0
22:30:09 DHCPd:DISCOVER received, MAC:00-E0-70-CD-F5-73, XID:D7F194D
22:30:10 DHCPd:OFFER sent, IP:9.9.9.3, XID:D7F194D
22:30:11 DHCPd:REQUEST received, MAC:00-E0-70-CD-F5-73, XID:D7F194D
22:30:11 DHCPd:ACK sent, IP:9.9.9.3, XID:D7F194D
22:30:12 HTTPd:Connect: 9.9.9.3, TID=9652

22:30:12 HTTPd:Client: 9.9.9.3 [GET] /menu-uefi.ipxe
22:30:12 HTTPd:Server : Returning /menu-uefi.ipxe
22:30:18 HTTPd:Client: 9.9.9.3 [GET] /iso/wepe.iso
22:30:18 HTTPd:Server : Returning /iso/wepe.iso
22:30:21 HTTPd:Client: 9.9.9.3 [GET] /grub.exe
22:30:21 HTTPd:Server : Returning /grub.exe
#从上面的日志可知，一开始目标主机使用 pxe 启动，获取到 9.9.9.2 的 ip 及启动文件 ipxe-x86_64-uefi-http.efi，启动文件执行后默认下载 autoexec.ipxe（服务器上没有此文件也不影响）
由于 ipxe-x86_64-uefi-http.efi 已内置菜单，所以在 ipxe 重新初始化 dhcp 后，直接按内置菜单指定的从 http 下载 menu-uefi.ipxe 文件

 # bios 启动下的日志：
22:54:41 DHCPc:discovering for another DHCPd on LAN
22:54:41 ROOT=D:\TinyPXEsrv1.0.0.23\files\
22:54:41 DHCPd 10.99.1.1:67 started...
22:54:41 TFPTd 10.99.1.1:69 started...
22:54:41 HTTPd:80 started...

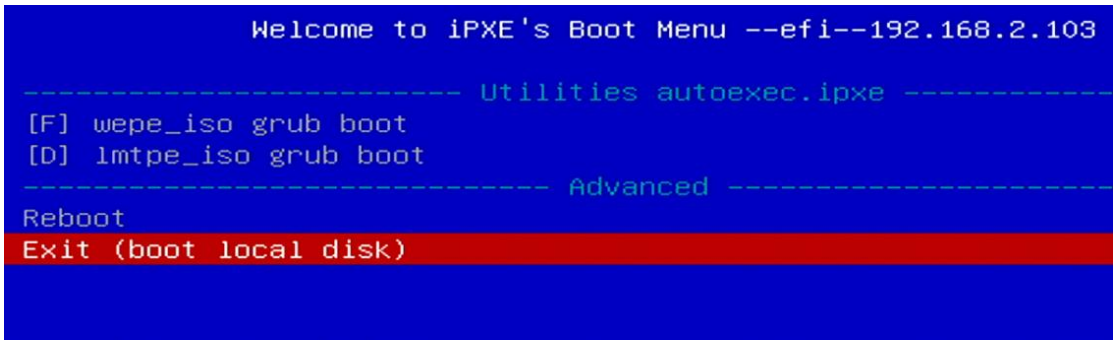
22:54:46 DHCPd:DISCOVER received, MAC:00-0C-29-69-21-01, XID:2A692101
22:54:47 DHCPd:OFFER sent, IP:10.99.1.2, XID:2A692101
22:54:49 DHCPd:REQUEST received, MAC:00-0C-29-69-21-01, XID:2A692101
22:54:49 DHCPd:ACK sent, IP:10.99.1.2, XID:2A692101
22:54:49 TFTPd:DoReadFile:undionly-bios-http.kpxe B:1456 T:0
22:54:49 DHCPd:DISCOVER received, MAC:00-0C-29-69-21-01, XID:C7CB6911
22:54:49 DHCPd:OFFER sent, IP:10.99.1.3, XID:C7CB6911
22:54:50 DHCPd:DISCOVER received, MAC:00-0C-29-69-21-01, XID:C7CB6911
22:54:50 DHCPd:OFFER sent, IP:10.99.1.4, XID:C7CB6911
22:54:52 DHCPd:REQUEST received, MAC:00-0C-29-69-21-01, XID:C7CB6911
22:54:52 DHCPd:ACK sent, IP:10.99.1.4, XID:C7CB6911
22:54:52 TFTPd:DoReadFile:menu-bios.ipxe B:1432 T:884
22:54:59 HTTPd:Connect: 10.99.1.4, TID=9928
22:54:59 HTTPd:Client: 10.99.1.4 [GET] /iso/wepe.iso
22:54:59 HTTPd:Server : Returning /iso/wepe.iso
22:55:23 HTTPd:Client: 10.99.1.4 [GET] /grub.exe
22:55:23 HTTPd:Server : Returning /grub.exe

#bios 启动的 ipxe 流程和 uefi 一样，只是它不会默认下载 autoexec.ipxe 文件

★ipxe 菜单文件示例

#!ipxe	#注释
set boot-url http://${next-server}	设置变量，变量 boot-url 表示为后面的值
set menu-timeout 15000	设置变量，变量名为 menu-timeout，值为 15000
:start	菜单开始标记
menu Welcome to iPXE's Boot Menu --${platform}--${ip}	显示 menu 标题，并显示平台及 ip 地址
item	菜单列表项，一个 item 一行，只写 item 表示空行
item --gap -- ---------- Utilities autoexec.ipxe ----------	--gap --表示这一项为分割线而已，不算一行菜单
item --key f wepe [F] wepe_iso grub boot	--key 指定快捷键，按下 f 选择名为 wepe 的菜单项
item --key d lmtpe [D] lmtpe_iso grub boot	按下字母 d，选择名为 lmtpe 的菜单项
item --gap -- ---------- Advanced ----------	--gap --后面的字符是会显示在启动界面的
item reboot Reboot	item 后面第一个单词为启动项名称，不会显示
item exit Exit (boot local disk)	item 后面第 2 个单词至行尾表示提示名称，会显示的
choose --default exit --timeout ${menu-timeout} target && goto ${target}	等待超时，选择默认启动项（exit 这个启动项），单位毫秒，15000 表示 15 秒
:wepe	具体启动项的定义，以下内容不会在启动界面显示
initrd http://${next-server}/iso/wepe.iso	
chain http://${next-server}/grub.exe --config-file="map (rd)+1 (0xff);map --hook;chainloader (0xff)"	
:lmtpe	具体启动项的定义
initrd http://${next-server}/iso/lmtpe.iso	
chain http://${next-server}/grub.exe --config-file="map (rd)+1 (0xff);map --hook;chainloader (0xff)"	
:reboot	具体启动项的定义
reboot	
:exit	具体启动项的定义
exit	

显示效果如下：



这个背景太单调了，能否指定背景图片呢？当然可以，请看下一小节

★ipxe 指定背景图片

可用 console 命令指定背景图片，默认下载的 ipxe 引导程序是不支持此命令的，需要使用源码编译（修改一下源码）
源码获取地址： https://github.com/ipxe/ipxe
#以下是在 centos7 x86 系统下的编译

```
# yum install -y gcc mtools binutils make perl xz-devel xorriso syslinux unzip

# unzip ipxe-master.zip                    #解压源码包
# cd ipxe-master/src/                      #进入解压目录的 src/子目录下
# vi config/general.h                      #将以下 2 行取消注释（删除最前面的//双斜线，define 前的#号得留着，C 语言中#号是特殊字符，不是注释，注释为//，下面已删除了双斜//注释）
#define CONSOLE_CMD
#define IMAGE_PNG

# vi config/console.h                      #将以下 1 行取消注释（删除最前面的//双斜线）
#define CONSOLE_FRAMEBUFFER

#This build option enables the commands console, colour, and cpair.

# make bin/undionly.kpxe                   #编译 bios 启动模式的 ipxe 引导程序
# cp bin/undionly.kpxe /var/lib/tftpboot/  #复制到 tftp 根目录下

# make bin-x86_64-efi/ipxe.efi             #编译 uefi 启动模式的 ipxe 引导程序
# cp bin-x86_64-efi/ipxe.efi /var/lib/tftpboot/  #复制到 tftp 根目录下
```

编写菜单文件：

```
# vi /var/lib/tftpboot/menu.ipxe           #内容如下（只支持 png 格式的图片）
#!ipxe
# console --x 480 --y 320                  #单独设置 console 大小是可以的，但如果下面设置了背景图片，则窗口大小就失效了，需要在指定背景图片的同时设置窗口大小，同一行里设置，如下面这行：
console --picture http://${next-server}/myipxe.png --x 480 --y 320
set menu-timeout 30000
set base http://${next-server}/CentOS-7-x86_64-DVD-2009
:start
menu Welcome to iPXE's Boot Menu --${platform}--${ip}
item
item --gap -- ---------- Utilities autoexec.ipxe ------------
item ks install centos7 with kickstart
item reboot Reboot
item exit Exit
choose --default exit --timeout ${menu-timeout} target && goto ${target}
:ks
kernel ${base}/images/pxeboot/vmlinuz initrd=initrd.img ks=http://${next-server}/centos7.ks
```

```
initrd    ${base}/images/pxeboot/initrd.img
boot
:reboot
reboot
:exit
exit
```

启动效果如下：

★ipxe 引导 kickstart 安装 centos7

首先还是要先准备好 3 个服务：dhcp，tftp，http；可以使用 Tiny PXE server，也可以自行安装这 3 个服务
然后将 CentOS-7-x86_64-DVD-2009.iso 镜像文件解压到 http 的/CentOS-7-x86_64-DVD-2009 目录下
再准备 ipxe 引导文件及编写相应的 ipxe 菜单配置文件，将这些文件放到 tftp 的/根目录下
dhcp 匹配 option 77 = iPXE 时要指定 filename 为 ipxe 的菜单配置文件（比如 menu.ipxe）

①tftp 的/根目录下的文件：

```
undionly.kpxe                #bios 启动模式的 ipxe 引导文件
ipxe.efi                     #uefi 启动模式的 ipxe 引导文件
menu.ipxe                    #ipxe 的菜单配置文件
```

②http 的/根目录下的文件及子目录

```
CentOS-7-x86_64-DVD-2009/    #centos7 的安装光盘镜像解压目录，即 centos7 的 yum-repo 仓库源
centos7.ks                   #适配目标主机的用于安装 centos7 的 kickstart 无人值守自动应答文件
```

③dhcp 的配置（部分）

```
option client-architecture code 93 = unsigned integer 16;
if exists user-class and option user-class = "iPXE" {
  filename "menu.ipxe";
} elsif option client-architecture = 00:00 {
  filename "undionly.kpxe";
} else {
  filename "ipxe.efi";
}
```

④ipxe 的菜单配置文件内容（menu.ipxe）

```
#!ipxe
set base http://${next-server}/CentOS-7-x86_64-DVD-2009
kernel ${base}/images/pxeboot/vmlinuz initrd=initrd.img ks=http://${next-server}/centos7.ks
initrd ${base}/images/pxeboot/initrd.img
boot
```

⑤kickstart 自动应答文件内容（centos7.ks）

```
#version=DEVEL
# 注意：要确认目标设备的网卡名称及磁盘名称，比如 ens33，/dev/sda，磁盘有数据时一定要先备份
# 适用系统 centos7

#配置安装源，ip 地址为 http 服务的地址（Use network installation），也可用 cdrom 表示使用安装光盘
url --url="http://10.99.1.2/CentOS-7-x86_64-DVD-2009"
```

```
# System authorization information
auth --enableshadow --passalgo=sha512

# License agreement
eula --agreed

#使用图形界面安装向导 （Use graphical install）
#graphical

#使用文字界面安装向导 （Use text mode install）
text

# Do not configure the X Window System
skipx

# Run the Setup Agent on first boot
firstboot --enable
ignoredisk --only-use=sda

#设置键盘 （Keyboard layouts）
keyboard --vckeymap=us --xlayouts='us'

#设置系统语言 （System language）
lang en_US.UTF-8

#配置网络 （Network information），要提前确认网络接口名称
network --bootproto=static --device=ens33 --ip=10.99.1.66 --gateway=10.99.1.1 --nameserver=223.5.5.5
--netmask=255.255.255.0 --activate --hostname=localhost.localdomain --onboot=yes
#network --bootproto=dhcp --device=ens33 --onboot=yes --ipv6=auto --activate --hostname=localhost.localdomain

#设置 root 用户密码 （Root password），可用以下命令生成密码 hash
# /usr/libexec/platform-python -c 'import crypt; print ( crypt.crypt("passwdxx", "$6$saltxxyy$" ) )'
rootpw --iscrypted $6$saltxxyy$9GvvDX71RCTAeeEP193El.ZWqhXsevvQ4NYUmVxhY.hD0TiI6bdiEe.CqK8cIxRQe75iQa8IyuZsQFbFUbDnV.

# System services
services --enabled="chronyd"

#设置系统时区 （System timezone）
timezone Asia/Shanghai

# System bootloader configuration
bootloader --append=" crashkernel=auto" --location=mbr --boot-drive=sda
```

```
#清除硬盘上的主引导记录 （Clear the Master Boot Record）
zerombr

#清除硬盘上的分区表信息 （Partition clearing information）
clearpart  --drives=sda  --initlabel

#分区设置，创建分区 （Disk partitioning information）
# /boot/efi 实际是 vfat 文件系统
part  /boot/efi  --fstype="efi"  --ondisk=sda  --size=200
--fsoptions="defaults,uid=0,gid=0,umask=0077,shortname=winnt"
part  /boot  --fstype="xfs"  --ondisk=sda  --size=512
#  part  swap  --asprimary  --fstype="swap"  --size=2048
#  part  /  --asprimary  --fstype="xfs"  --size=4096  --grow
#使用 lvm，配置如下：
part  pv.01  --fstype="lvmpv"  --ondisk=sda  --size=1  --grow
volgroup  centos  --pesize=4096  pv.01
logvol  swap  --fstype="swap"  --name=swap  --vgname=centos  --size 2048
logvol  /  --fstype="xfs"  --name=root  --vgname=centos  --size=4096  --grow

#安装软件，最小化，核心包，chrony 软件（适用于 centos7）
#只写@core 核心包，则安装 313 个左右数量的软件包（默认含有 chrony）
#额外写了@base 基础包，则安装 490 个左右数量的软件包
#默认安装的最小化包如下：
%packages
@^minimal
@core
chrony
kexec-tools
%end

%addon com_redhat_kdump --enable --reserve-mb='auto'
%end

%anaconda
pwpolicy root --minlen=6 --minquality=1 --notstrict --nochanges --notempty
pwpolicy user --minlen=6 --minquality=1 --notstrict --nochanges --emptyok
pwpolicy luks --minlen=6 --minquality=1 --notstrict --nochanges --notempty
%end

#安装后执行的脚本
%post
cat  >> /etc/ssh/sshd_config <<EOF
GSSAPIAuthentication  no
PermitRootLogin  yes
```

```
UseDNS    no
EOF
%end

#安装完成后，自动重启系统 （Reboot after installation）
reboot

#如果报错：Warning: /dev/root does not exist 说明内存太小了，要 2GB 及以上
#不要用 7zip 去解压.iso 镜像文件，会有解压错误，不完整的文件名
```

★附 tftp,http,dhcp 三个服务安装及配置脚本（centos7 的）

```
#!/bin/bash
yum    install    xinetd    tftp    tftp-server    -y
systemctl    enable    xinetd
cat    > /etc/xinetd.d/tftp    <<EOF
service tftp
{
            socket_type                 = dgram
            protocol                    = udp
            wait                         = yes
            user                         = root
            server                      = /usr/sbin/in.tftpd
            server_args                 = -s /var/lib/tftpboot
            disable                     = no
            per_source                   = 11
            cps                          = 100 2
            flags                       = IPv4
}
EOF
chmod    755    /var/lib/tftpboot
systemctl    restart    xinetd
firewall-cmd --add-service=tftp
firewall-cmd    --runtime-to-permanent
#需要手动复制 undionly.kpxe    ipxe.efi    menu.ipxe 三个文件到 /var/lib/tftpboot 目录下，给读权限


yum    install    httpd    -y
systemctl    enable    httpd
systemctl    start    httpd
firewall-cmd    --add-service=http
firewall-cmd    --runtime-to-permanent
#手动复制镜像解压目录 CentOS-7-x86_64-DVD-2009    及 centos7.ks 文件到 /var/www/html 目录下


yum    install    dhcp    -y
systemctl    enable    dhcpd
cat    >    /etc/dhcp/dhcpd.conf    <<EOF
default-lease-time    43200;
max-lease-time    86400;
option    client-architecture    code 93 = unsigned integer 16;
subnet    10.99.1.0    netmask    255.255.255.0    {
      option    routers    10.99.1.1;
      option    subnet-mask    255.255.255.0;
```

```
    option domain-name-servers 8.8.8.8,114.114.114.114;
    option domain-name "cof-lee.com";
    range dynamic-bootp 10.99.1.100 10.99.1.200;
    next-server 10.99.1.2;              # tftp 服务器的地址
    if exists user-class and option user-class = "iPXE" {
      filename "menu.ipxe";
    } elsif option client-architecture = 00:00 {
      filename "undionly.kpxe";
    } else {
      filename "ipxe.efi";
    }
}
EOF
systemctl start dhcpd
firewall-cmd --add-service=dhcp
firewall-cmd --runtime-to-permanent
```

⑥最后，启动目标计算机，选择从网络启动，即可进入 ipxe 界面，并自动安装 centos7 系统

⑦过程及效果如下：

★服务端查看 tftp,dhcp,http 日志：

```
tail -f /var/log/messages
tail -f /var/lib/dhcpd/dhcpd.leases
tail -f /var/log/httpd/access_log
```

★bios 启动模式下：

dhcp 及 tftp 日志：

```
Oct  5 18:53:31 localhost dhcpd: DHCPDISCOVER from 00:0c:29:58:2f:fc via ens33
Oct  5 18:53:32 localhost dhcpd: DHCPOFFER on 10.99.1.101 to 00:0c:29:58:2f:fc via ens33
Oct  5 18:53:33 localhost dhcpd: DHCPREQUEST for 10.99.1.101 (10.99.1.2) from 00:0c:29:58:2f:fc via ens33
Oct  5 18:53:33 localhost dhcpd: DHCPACK on 10.99.1.101 to 00:0c:29:58:2f:fc via ens33        #首先是 PXE 阶段，分配 10.99.1.101 的地址给客户机
Oct  5 18:53:33 localhost xinetd[2192]: START: tftp pid=2521 from=10.99.1.101
Oct  5 18:53:33 localhost in.tftpd[2522]: Error code 0: TFTP Aborted
Oct  5 18:53:33 localhost in.tftpd[2523]: Client 10.99.1.101 finished undionly.kpxe          #客户端使用 10.99.1.101 从 tftp 下载了 undionly.kpxe 引导文件
Oct  5 18:53:35 localhost dhcpd: DHCPDISCOVER from 00:0c:29:58:2f:fc via ens33              #ipxe 阶段，重新进行 dhcp 获取新的地址
Oct  5 18:53:36 localhost dhcpd: DHCPOFFER on 10.99.1.100 to 00:0c:29:58:2f:fc via ens33
Oct  5 18:53:36 localhost dhcpd: DHCPDISCOVER from 00:0c:29:58:2f:fc via ens33
Oct  5 18:53:36 localhost dhcpd: DHCPOFFER on 10.99.1.100 to 00:0c:29:58:2f:fc via ens33
Oct  5 18:53:38 localhost dhcpd: DHCPREQUEST for 10.99.1.100 (10.99.1.2) from 00:0c:29:58:2f:fc via ens33
Oct  5 18:53:38 localhost dhcpd: DHCPACK on 10.99.1.100 to 00:0c:29:58:2f:fc via ens33        #ipxe 获取新地址为 10.99.1.100
Oct  5 18:53:38 localhost in.tftpd[2524]: Client 10.99.1.100 finished menu.ipxe             #客户端使用 10.99.1.100 从 tftp 下载了菜单配置文件
Oct  5 18:53:54 localhost dhcpd: DHCPDISCOVER from 00:0c:29:58:2f:fc via ens33              #之后又进行了一次 dhcp 获取地址，这个是 vmlinuz 内核的操作
Oct  5 18:53:54 localhost dhcpd: DHCPOFFER on 10.99.1.101 to 00:0c:29:58:2f:fc via ens33
Oct  5 18:53:54 localhost dhcpd: DHCPREQUEST for 10.99.1.101 (10.99.1.2) from 00:0c:29:58:2f:fc via ens33
Oct  5 18:53:54 localhost dhcpd: DHCPACK on 10.99.1.101 to 00:0c:29:58:2f:fc via ens33
```

http 日志：

#以下 2 行是使用 10.99.1.100 的地址去下载的 vmlinuz 及 initrd.img 文件（这是 ipxe 菜单项里的配置，说明是 ipxe 去下载的）

10.99.1.100 - - [05/Oct/2023:18:53:38 +0800] "GET /CentOS-7-x86_64-DVD-2009/images/pxeboot/vmlinuz HTTP/1.1" 200 6769256 "-" "iPXE/1.21.1+ (gcad1)"

10.99.1.100 - - [05/Oct/2023:18:53:39 +0800] "GET /CentOS-7-x86_64-DVD-2009/images/pxeboot/initrd.img HTTP/1.1" 200 55129656 "-" "iPXE/1.21.1+ (gcad1)"

#以下获取 kickstart 自动应答文件，说明已经到了内核的操作阶段了，内核又重新获取 dhcp 配置，10.99.1.101 的地址

10.99.1.101 - - [05/Oct/2023:18:53:57 +0800] "GET /centos7.ks HTTP/1.1" 200 3495 "-" "curl/7.29.0"

10.99.1.101 - - [05/Oct/2023:18:53:58 +0800] "GET /CentOS-7-x86_64-DVD-2009/.treeinfo HTTP/1.1" 200 354 "-" "curl/7.29.0"

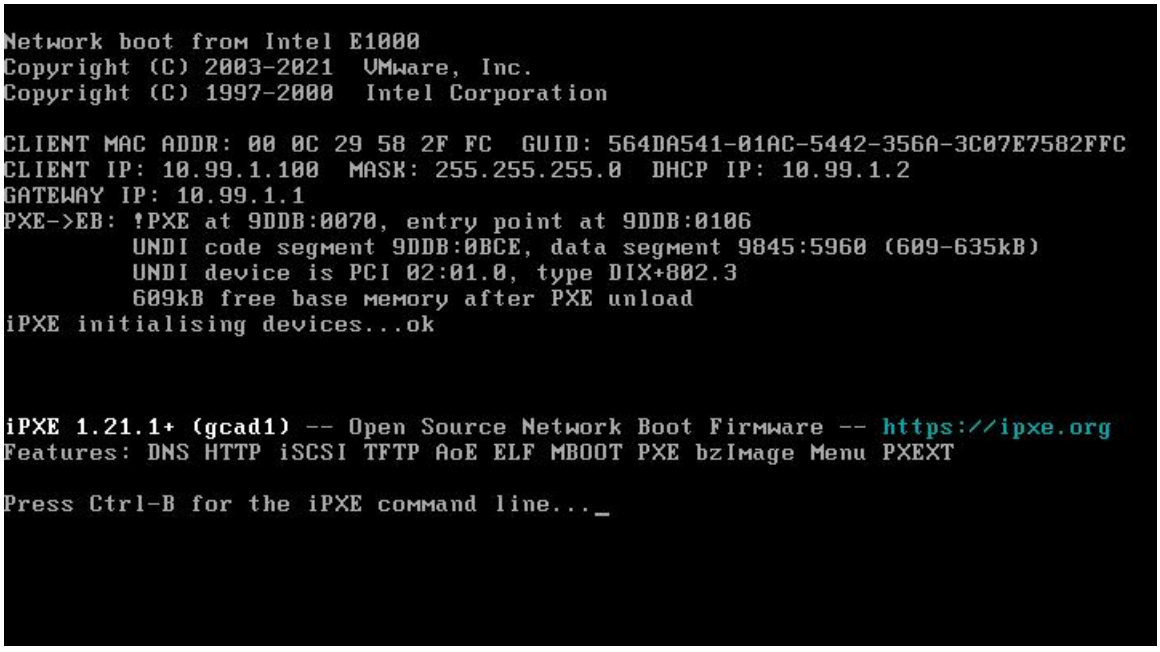
10.99.1.101 - - [05/Oct/2023:18:53:58 +0800] "GET /CentOS-7-x86_64-DVD-2009/LiveOS/squashfs.img HTTP/1.1" 200 521617408 "-" "curl/7.29.0"

10.99.1.101 - - [05/Oct/2023:18:54:02 +0800] "GET /CentOS-7-x86_64-DVD-2009/images/updates.img HTTP/1.1" 404 241 "-" "curl/7.29.0"

10.99.1.101 - - [05/Oct/2023:18:54:02 +0800] "GET /CentOS-7-x86_64-DVD-2009/images/product.img HTTP/1.1" 404 241 "-" "curl/7.29.0"

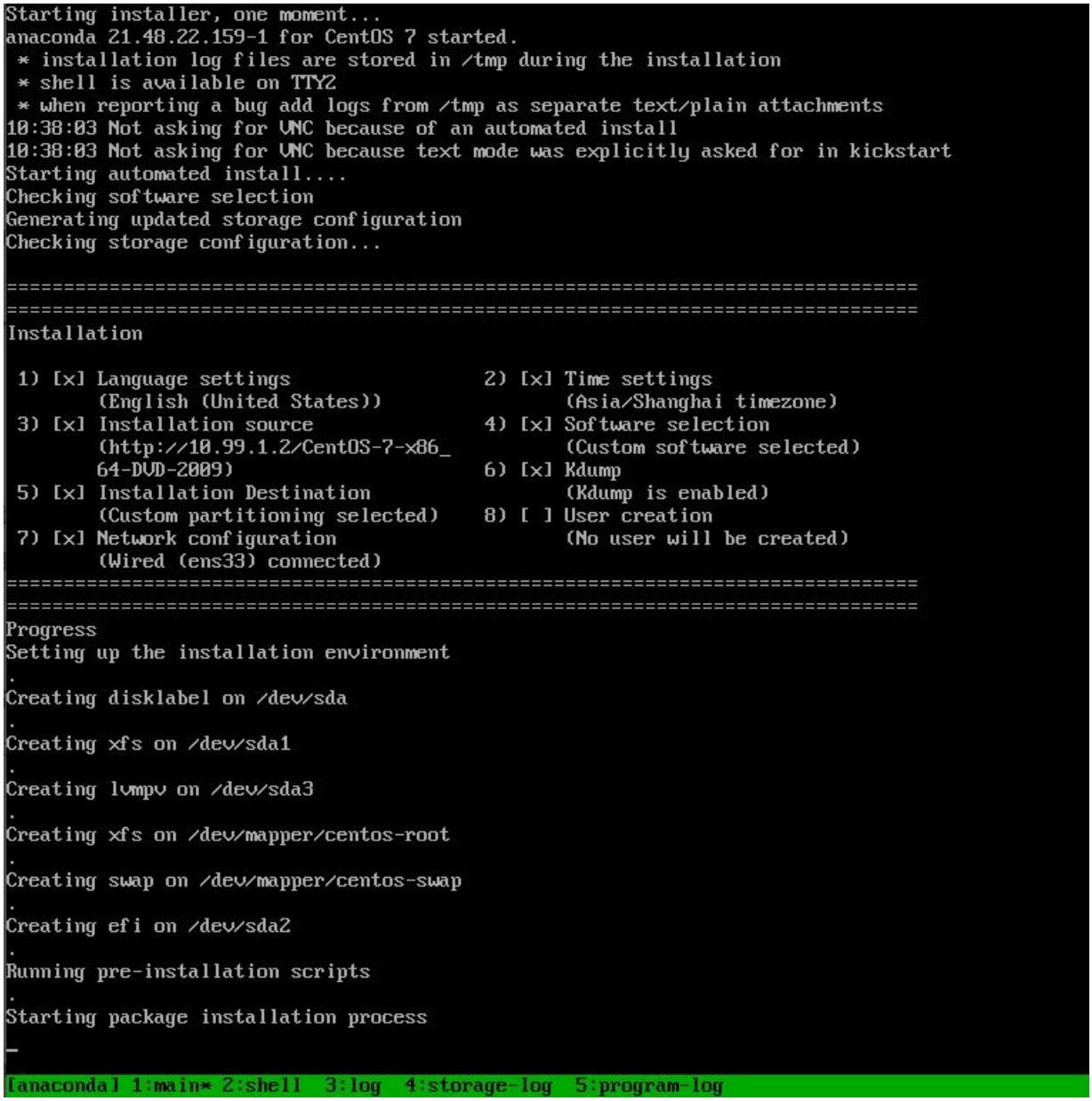
过程截图：

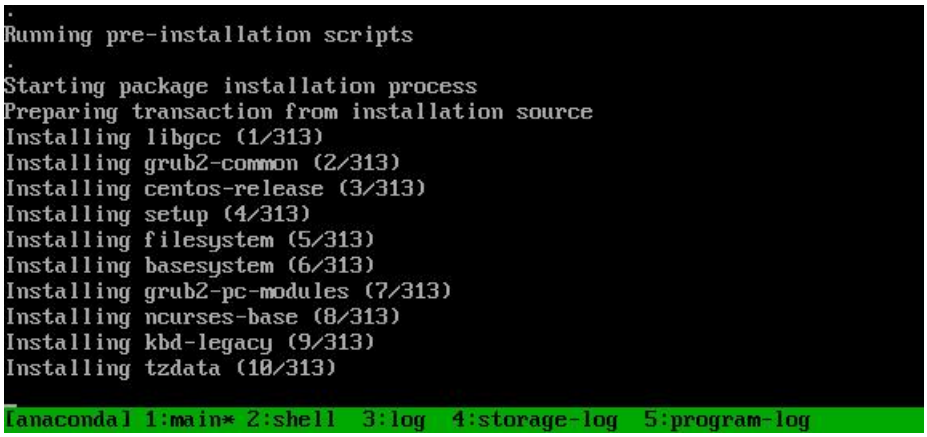
首先是 PXE 启动，从 tftp 获取 ipxe 引导文件后，将控制权交给了 ipxe



ipxe 先是通过 tftp 下载菜单文件 menu.ipxe，应用菜单项，通过 http 去下载内核文件 vmlinuz 及 initrd.img

内核文件解压，再下载 kickstart 文件，进行自动化安装系统





安装完成后，自动重启系统，进入安装后的 centos7 系统

```
CentOS Linux 7 (Core)
Kernel 3.10.0-1160.el7.x86_64 on an x86_64

localhost login: root
Password:
Last login: Thu Oct  5 10:46:38 from 10.99.1.1
[root@localhost ~]#
[root@localhost ~]# ip add
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:58:2f:fc brd ff:ff:ff:ff:ff:ff
    inet 10.99.1.66/24 brd 10.99.1.255 scope global noprefixroute ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe58:2ffc/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
[root@localhost ~]#
[root@localhost ~]#
```

★UEFI 启动模式下

dhcp 及 tftp 日志：

Oct 5 19:14:15 localhost dhcpd: DHCPDISCOVER from 00:0c:29:58:2f:fc via ens33

Oct 5 19:14:16 localhost dhcpd: DHCPOFFER on 10.99.1.101 to 00:0c:29:58:2f:fc via ens33

Oct 5 19:14:16 localhost dhcpd: DHCPREQUEST for 10.99.1.101 (10.99.1.2) from 00:0c:29:58:2f:fc via ens33

Oct 5 19:14:16 localhost dhcpd: DHCPACK on 10.99.1.101 to 00:0c:29:58:2f:fc via ens33

Oct 5 19:14:16 localhost in.tftpd[2604]: Error code 8: User aborted the transfer

Oct 5 19:14:16 localhost in.tftpd[2605]: Client 10.99.1.101 finished ipxe.efi #下载的是 ipxe.efi 引导，和 bios 模式下的不同

Oct 5 19:14:18 localhost dhcpd: DHCPDISCOVER from 00:0c:29:58:2f:fc via ens33

Oct 5 19:14:19 localhost dhcpd: DHCPOFFER on 10.99.1.100 to 00:0c:29:58:2f:fc via ens33

Oct 5 19:14:19 localhost dhcpd: DHCPREQUEST for 10.99.1.100 (10.99.1.2) from 00:0c:29:58:2f:fc via ens33

Oct 5 19:14:19 localhost dhcpd: DHCPACK on 10.99.1.100 to 00:0c:29:58:2f:fc via ens33

Oct 5 19:14:20 localhost in.tftpd[2607]: Client 10.99.1.100 finished menu.ipxe #下载菜单文件

Oct 5 19:14:35 localhost dhcpd: DHCPDISCOVER from 00:0c:29:58:2f:fc via ens33

Oct 5 19:14:35 localhost dhcpd: DHCPOFFER on 10.99.1.101 to 00:0c:29:58:2f:fc via ens33

Oct 5 19:14:35 localhost dhcpd: DHCPREQUEST for 10.99.1.101 (10.99.1.2) from 00:0c:29:58:2f:fc via ens33

Oct 5 19:14:35 localhost dhcpd: DHCPACK on 10.99.1.101 to 00:0c:29:58:2f:fc via ens33

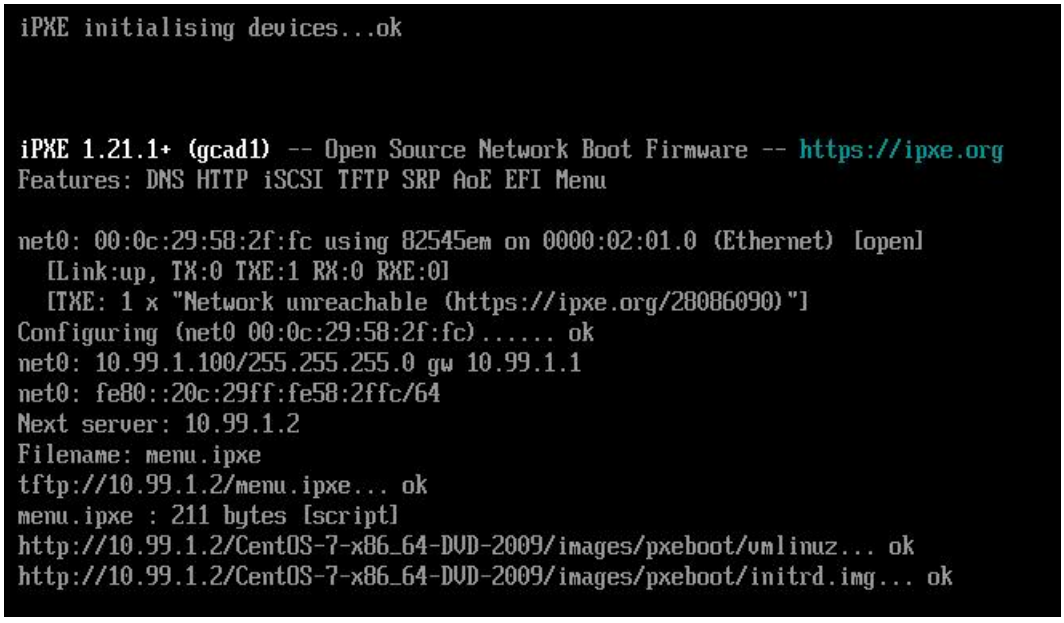
http 日志：

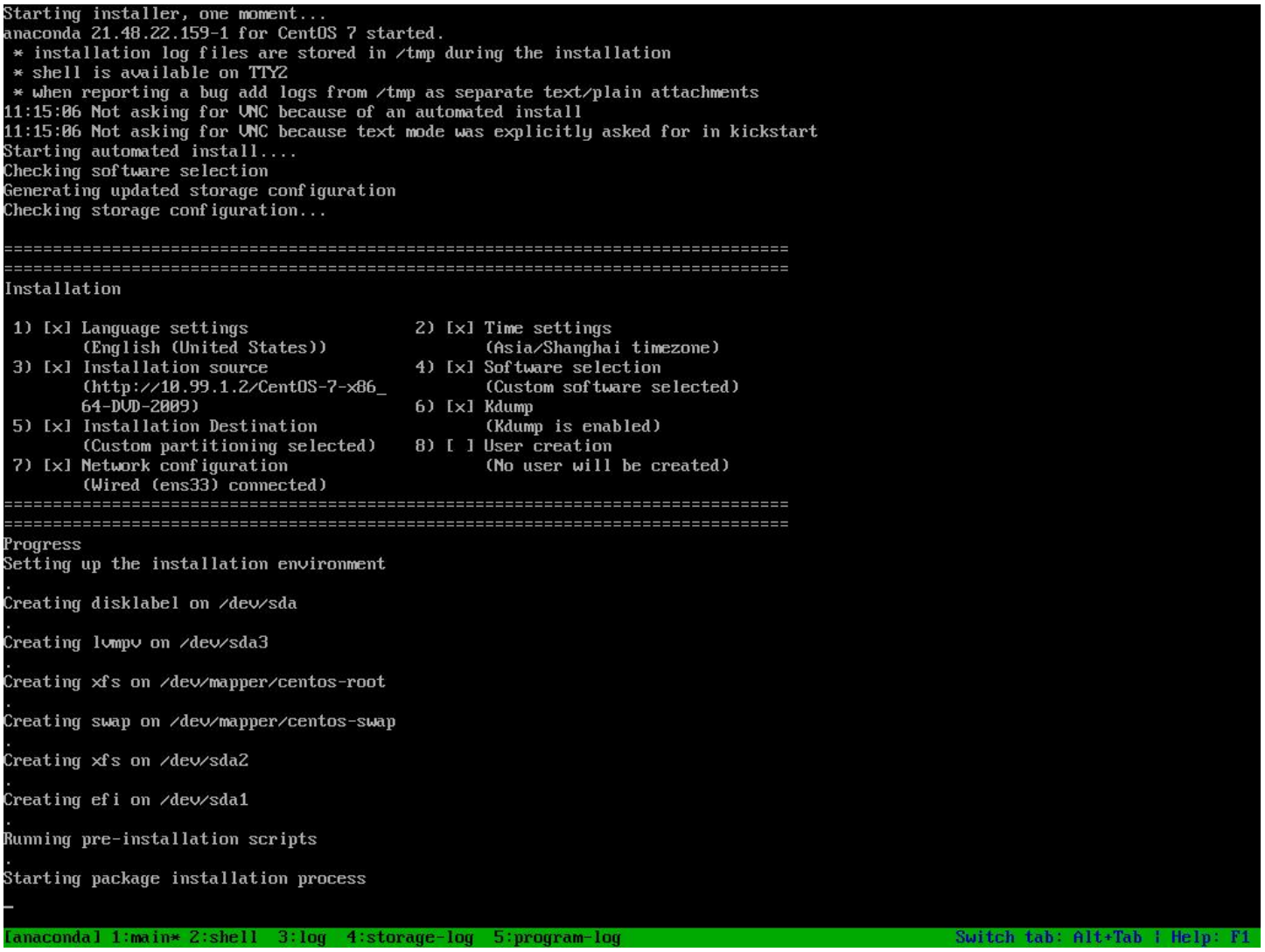
10.99.1.100 - - [05/Oct/2023:19:14:20 +0800] "GET /CentOS-7-x86_64-DVD-2009/images/pxeboot/vmlinuz HTTP/1.1" 200 6769256 "-" "iPXE/1.21.1+ (gcad1)"

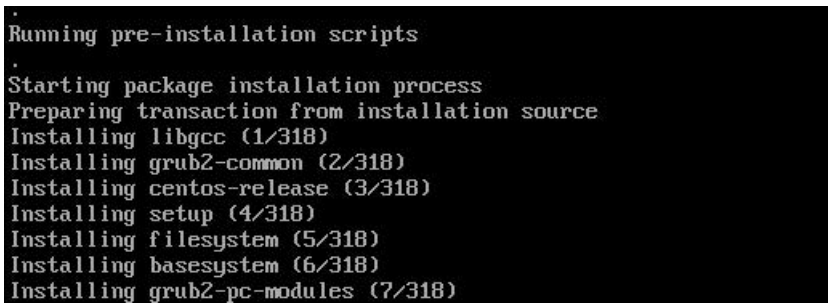
10.99.1.100 - - [05/Oct/2023:19:14:20 +0800] "GET /CentOS-7-x86_64-DVD-2009/images/pxeboot/initrd.img HTTP/1.1" 200 55129656 "-" "iPXE/1.21.1+ (gcad1)"

10.99.1.101 - - [05/Oct/2023:19:14:38 +0800] "GET /centos7.ks HTTP/1.1" 200 3495 "-" "curl/7.29.0"


iPXE initialising devices...ok
iPXE 1.21.1+ (gcad1) -- Open Source Network Boot Firmware -- https://ipxe.org
Features: DNS HTTP iSCSI TFTP SRP AoE EFI Menu
Press Ctrl-B for the iPXE command line...


iPXE initialising devices...ok
iPXE 1.21.1+ (gcad1) -- Open Source Network Boot Firmware -- https://ipxe.org
Features: DNS HTTP iSCSI TFTP SRP AoE EFI Menu
net0: 00:0c:29:58:2f:fc using 82545em on 0000:02:01.0 (Ethernet) [open]
[Link:up, TX:0 TXE:1 RX:0 RXE:0]
[TXE: 1 x "Network unreachable (https://ipxe.org/28086090)"]
Configuring (net0 00:0c:29:58:2f:fc)..... ok
net0: 10.99.1.100/255.255.255.0 gw 10.99.1.1
net0: fe80::20c:29ff:fe58:2ffc/64
Next server: 10.99.1.2
Filename: menu.ipxe
tftp://10.99.1.2/menu.ipxe... ok
menu.ipxe : 211 bytes [script]
http://10.99.1.2/CentOS-7-x86_64-DVD-2009/images/pxeboot/vmlinuz... ok
http://10.99.1.2/CentOS-7-x86_64-DVD-2009/images/pxeboot/initrd.img... ok


Starting installer, one moment...
anaconda 21.48.22.159-1 for CentOS 7 started.
* installation log files are stored in /tmp during the installation
* shell is available on TTY2
* when reporting a bug add logs from /tmp as separate text/plain attachments
11:15:06 Not asking for VNC because of an automated install
11:15:06 Not asking for VNC because text mode was explicitly asked for in kickstart
Starting automated install....
Checking software selection
Generating updated storage configuration
Checking storage configuration...
Installation
1) [x] Language settings
(English (United States))
2) [x] Time settings
(Asia/Shanghai timezone)
3) [x] Installation source
(http://10.99.1.2/CentOS-7-x86_
64-DVD-2009)
4) [x] Software selection
(Custom software selected)
5) [x] Installation Destination
(Custom partitioning selected)
6) [x] Kdump
(Kdump is enabled)
7) [x] Network configuration
(Wired (ens33) connected)
8) [ ] User creation
(No user will be created)
Progress
Setting up the installation environment
Creating disklabel on /dev/sda
Creating lvmpv on /dev/sda3
Creating xfs on /dev/mapper/centos-root
Creating swap on /dev/mapper/centos-swap
Creating xfs on /dev/sda2
Creating efi on /dev/sda1
Running pre-installation scripts
Starting package installation process
[anaconda] 1:main* 2:shell 3:log 4:storage-log 5:program-log
Switch tab: Alt+Tab | Help: F1


Running pre-installation scripts
Starting package installation process
Preparing transaction from installation source
Installing libgcc (1/318)
Installing grub2-common (2/318)
Installing centos-release (3/318)
Installing setup (4/318)
Installing filesystem (5/318)
Installing basesystem (6/318)
Installing grub2-pc-modules (7/318)

```
CentOS Linux 7 (Core)
Kernel 3.10.0-1160.el7.x86_64 on an x86_64

localhost login: root
Password:
[root@localhost ~]#
[root@localhost ~]# ip add
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:58:2f:fc brd ff:ff:ff:ff:ff:ff
    inet 10.99.1.66/24 brd 10.99.1.255 scope global noprefixroute ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe58:2ffc/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
[root@localhost ~]#
[root@localhost ~]# efibootmgr
BootCurrent: 0004
BootOrder: 0004,0000,0001,0002,0003
Boot0000* EFI VMware Virtual SCSI Hard Drive (0.0)
Boot0001* EFI VMware Virtual IDE CDROM Drive (IDE 1:0)
Boot0002* EFI Network
Boot0003* EFI Internal Shell (Unsupported option)
Boot0004* CentOS
[root@localhost ~]#
[root@localhost ~]#
```

★ubuntu　ipxe 网络安装规则：

未完成！

```
#!ipxe
set mirror https://mirrors.aliyun.com/
set release bionic
set arch amd64
set base-url ${mirror}/ubuntu/dists/${release}/main/installer-${arch}/current/images/netboot/ubuntu-installer/${arch}
kernel ${base-url}/linux auto=true url=http://www.haiyun.me/ubuntu.cfg keymap=us domain= hostname=ubuntu-server
interface=auto netcfg/get_ipaddress=192.168.168.4 netcfg/get_netmask=255.255.255.0 netcfg/get_gateway=192.168.168.1
netcfg/get_nameservers=192.168.168.1 netcfg/disable_dhcp=true
initrd ${base-url}/initrd.gz
boot
```

preseed 文件：

```
d-i debian-installer/locale string en_US
d-i console-setup/ask_detect boolean false
d-i keyboard-configuration/xkb-keymap select us
#使用静态 IP
#d-i netcfg/choose_interface select auto
#d-i netcfg/disable_autoconfig boolean true
#d-i netcfg/dhcp_failed note
#d-i netcfg/dhcp_options select Configure network manually
#d-i netcfg/get_ipaddress string 192.168.168.4
#d-i netcfg/get_netmask string 255.255.255.0
#d-i netcfg/get_gateway string 192.168.168.1
#d-i netcfg/get_nameservers string 192.168.168.1
#d-i netcfg/confirm_static boolean true
#d-i netcfg/get_hostname string ubuntu-server
#d-i netcfg/get_f - U = 7 ; s ( 2domain string
d-i hw-detect/load_firmware boolean true
d-i mirror/country string manual
#d-i mirror/http/hostname string mirrors.aliyun.com
d-i mirror/http/hostname string archive.ubuntu.com
d-i mirror/http/directory string /ubuntu
d-i mirror/http/proxy string
d-i passwd/root-login boolean ture
d-i passwd/make-user boolean false
#d-i passwd/root-password password 123456
#d-i passwd/root-password-again password 123456
#echo 'import crypt,getpass; print crypt.crypt(getpass.getpass(), "$6$16_CHARACTER_SALT_HERE")' | python -
d-i passwd/root-password-crypted password $6$16_CHARACTER_SAL$CIIxSZip5SHbUDtMlgweMCsEMqUsp4kGjo
d-i user-setup/allow-password-weak boolean true
```

```
d-i user-setup/encrypt-home boolean false
d-i clock-setup/utc boolean false
d-i time/zone string Asia/Shanghai
d-i clock-setup/ntp boolean true
#d-i partman-auto/disk string /dev/sda
d-i partman/early_command string debconf-set partman-auto/disk "$(list-devices disk | head -n1)"
d-i partman-lvm/device_remove_lvm boolean true
d-i partman-md/device_remove_md boolean true
d-i partman-partitioning/confirm_write_new_label boolean true
d-i partman/choose_partition select finish
d-i partman/confirm boolean true
d-i partman/confirm_nooverwrite boolean true
d-i partman/default_filesystem string ext4
d-i partman/mount_style select uuid
d-i partman-auto/choose_recipe select boot-root
d-i partman-auto/method string regular
d-i partman-auto/expert_recipe string                         \
boot-root ::                                                  \
5120 1 5120 ext4                                              \
$primary{ } $bootable{ }                                      \
method{ format } format{ }                                    \
use_filesystem{ } filesystem{ ext4 }                          \
mountpoint{ / }                                               \
.                                                             \
1 3 -1 ext4                                                   \
$primary{ }                                                   \
method{ format } format{ }                                    \
use_filesystem{ } filesystem{ ext4 }
mountpoint{ /home }                                           \
.                                                             \
1024 2 1024 linux-swap                                        \
$primary{ }                                                   \
method{ swap } format{ }                                      \
.
#d-i partman-auto/choose_recipe select boot-root
#d-i partman-auto/method string regular
#d-i partman-auto/expert_recipe string                         \
#       boot-root ::                                          \
#               1 2 -1 ext4                                   \
#                       $primary{ }                           \
#                       method{ format } format{ }            \
#                       use_filesystem{ } filesystem{ ext4 }  \
#                       mountpoint{ / }                       \
#               .                                             \
```

```
#                       128 1 128 linux-swap                                    \
#                               $primary{ }                                     \
#                               method{ swap } format{ }                        \
#                       .
#d-i partman-auto/choose_recipe select boot-lvm
#d-i partman-auto/method string lvm
#d-i partman-auto-lvm/guided_size string 100%
#d-i partman-auto-lvm/new_vg_name string vg00
#d-i partman-lvm/confirm boolean true
#d-i partman-lvm/confirm_nooverwrite boolean true
#d-i partman-auto/expert_recipe string                                          \
#           boot-lvm ::                                                         \
#                       1024 1 1024 ext4
#                               $primary{ } $bootable{ }                        \
#                               method{ format } format{ }                      \
#                               use_filesystem{ } filesystem{ ext4 }        \
#                               mountpoint{ /boot }                             \
#                       .                                                       \
#                       1 2 -1 ext4                                             \
#                               $primary{ }                                     \
#                               $defaultignore{ }                               \
#                               method{ lvm }                                   \
#                               device{ /dev/sda }                              \
#                               vg_name{ vg00 }                                 \
#                       .                                                       \
#                       1024 3 1024 swap                                        \
#                               $lvmok{ } lv_name{ lv_swap } in_vg{ vg00 } \
#                               method{ swap } format{ }                        \
#                   .                                                               \
#                       1 4 -1 ext4                                             \
#                               $lvmok{} lv_name{ lv_root } in_vg{ vg00 }    \
#                               method{ format } format{ }                      \
#                               use_filesystem{ } filesystem{ ext4 }            \
#                               mountpoint{ / }                                 \
#                   .
tasksel tasksel/first multiselect minimal
d-i pkgsel/update-policy select none
d-i pkgsel/include string openssh-server vim wget tmux net-tools software-properties-common
d-i pkgsel/upgrade select none
d-i grub-installer/only_debian boolean true
d-i grub-installer/bootdev string default
d-i finish-install/reboot_in_progress note
d-i debian-installer/exit/reboot boolean true
d-i preseed/late_command string cd /target/;\
```

```
echo 'UseDNS no' >> etc/ssh/sshd_config;\
echo 'AddressFamily inet' >;> etc/ssh/sshd_config;\
echo 'PermitRootLogin yes' >> etc/ssh/sshd_config;
#d-i anna/choose_modules string network-console
#d-i network-console/password password 123456
#d-i network-console/password-again password 123456
```

第 37 章、封装 ks 自动应答脚本到 iso 镜像

有时需要安装系统时希望能自动划分分区及执行后续脚本，可以先解压 iso 光盘文件到某个主目录下，提前写好 kickstart 应答脚本，然后在 iso 光盘引导配置文件里指定使用 ks 脚本，最后再封装为 iso 光盘文件

x86_64 类型的 iso 安装光盘，BIOS 启动时一般由 isolinux 引导，有/isolinux 目录
aarch64 类型的 iso 安装光盘，只由 grub2 引导，没有/isolinux 目录

/isolinux/isolinux.cfg 启动项使用 ks 配置：

```
label linux
  menu label ^Install Rocky Linux 8.10
  kernel  vmlinuz
  append  initrd=initrd.img  inst.stage2=hd:LABEL=rocky8  inst.repo=hd:LABEL=rocky8  inst.ks=hd:LABEL=rocky8:/ks.cfg
quiet
```

/EFI/BOOT/grub.cfg 启动项使用 ks 配置：

```
menuentry 'Install Rocky Linux 8.10' --class red --class gnu-linux --class gnu --class os {
  linux  /images/pxeboot/vmlinuz  inst.stage2=hd:LABEL=rocky8  inst.repo=hd:LABEL=rocky8
inst.ks=hd:LABEL=rocky8:/ks.cfg  quiet
  initrd  /images/pxeboot/initrd.img
}
```

然后将指定的 ks.cfg 文件放到光盘根目录下

★使用 xorriso 创建 iso 光盘文件（x86_64 可启动）

```
# cd  iso 解压目录下
# mkisofs  -joliet-long  -iso-level  3  -v  -r  -J  -R  -T  -V  "rocky8"  \
-b  isolinux/isolinux.bin  -c  isolinux/boot.cat  -no-emul-boot  -boot-load-size  4 \
-boot-info-table  -eltorito-alt-boot  -e  images/efiboot.img  -no-emul-boot  \
./  -o  /root/rocky-linux-8-x86_64-myKS.iso
```

★使用 xorriso 创建 iso 光盘文件（aarch64 可启动）

```
# cd  iso 解压目录下
# mkisofs  -joliet-long  -iso-level  3  -v  -r  -J  -R  -T  -V  "rocky8"  \
-eltorito-alt-boot  -e  images/efiboot.img  -no-emul-boot  \
./  -o  /root/rocky-linux-8-aarch64-myKS.iso
```

参数说明：

-r	将目标目录创建为 iso 镜像文件
-o	指定生成的镜像文件名称

-b 指定在制作可开机光盘时所需的开机映像文件
-c 制作可开机光盘时，会将开机映像文件中的 no-eltorito-catalog 全部内容作成一个文件
-no-emul-boot 非模拟模式启动
-iso-level 3 设置 ISO 兼容级别
-boot-load-size 4 设置引导加载器大小
-boot-info-table 创建引导信息表
-joliet-long 使用 joliet 格式的目录与文件名称，长文件名支持
-R 或 -rock 使用 Rock RidgeExtensions
-J 或 -joliet 使用 Joliet 格式的目录与文件名称
-v 或 -verbose 执行时显示详细的信息
-T 或-translation-table 建立文件名的转换表，适用于不支持 Rock Ridge Extensions 的系统
-V 指定光盘卷标，要和启动菜单里的 hd:LABEL=指定的值保持一致
-joliet-long 在 iso 文件系统中使用 Joliet 扩展，以便在 windows 系统中显示长文件名和 unicode 字符
-eltorito-boot 指定启动加载器（boot loader）
-eltorito-alt-boot -e images/efiboot.img -no-emul-boot 支持 EFI 启动
-as mkisofs 指定使用 mkisofs 参数集来创建 ISO 文件

★注意：

windows 安装镜像用的不是 isolinux 引导，直接用 windows 10 安装光盘改，把里面内容删除，换上 pe 系统的内容，ultraiso 工具就可以实现（根目录下要有 bootmgr.efi 文件）