[UEFI Practice] Network Boot

原创 jiangwei0512 Posted on 2020-06-14 23:36:33 Read 1.1w Collection 17 Likes 5

Category Column: UEFI Development Basics Article Tags: uefi

Brief description

UEFI implements several network boot methods, such as HTTP boot, PXE boot, etc. The corresponding defaults are as follows:

cpp AI generated projects 登录复制 run

```cpp
!if $(PXE_ENABLE) == TRUE
  NetworkPkg/UefiPxeBcDxe/UefiPxeBcDxe.inf
!endif
!if $(HTTP_BOOT_ENABLE) == TRUE
  NetworkPkg/DnsDxe/DnsDxe.inf
  NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesDxe.inf
  NetworkPkg/HttpDxe/HttpDxe.inf
  NetworkPkg/HttpBootDxe/HttpBootDxe.inf
!endif
```

From their macro definitions, we can see which codes correspond to HTTP boot and PXE boot. The following introduces these two network boot methods respectively.

PXE boot

PXE network boot is an older network boot method. The PXE introduction and instructions have detailed instructions on how to use it, but here we mainly introduce its implementation. First, check the NetworkPkg\UefiPxeBcDxe\UefiPxeBcDxe.inf file. From here, you can see that it actually implements two protocols to package the entire PXE action:

cpp AI generated projects 登录复制 run

```cpp
[Protocols]
  ## TO_START
  ## SOMETIMES_CONSUMES
  gEfiDevicePathProtocolGuid
  gEfiNetworkInterfaceIdentifierProtocolGuid_31        ## SOMETIMES_CONSUMES
  gEfiArpServiceBindingProtocolGuid                    ## TO_START
  gEfiArpProtocolGuid                                  ## TO_START
  gEfiIp4ServiceBindingProtocolGuid                    ## TO_START
  gEfiIp4ProtocolGuid                                  ## TO_START
  gEfiIp4Config2ProtocolGuid                           ## TO_START
  gEfiIp6ServiceBindingProtocolGuid                    ## TO_START
  gEfiIp6ProtocolGuid                                  ## TO_START
  gEfiIp6ConfigProtocolGuid                            ## TO_START
  gEfiUdp4ServiceBindingProtocolGuid                   ## TO_START
  gEfiUdp4ProtocolGuid                                 ## TO_START
  gEfiMtftp4ServiceBindingProtocolGuid                 ## TO_START
  gEfiMtftp4ProtocolGuid                               ## TO_START
  gEfiDhcp4ServiceBindingProtocolGuid                  ## TO_START
  gEfiDhcp4ProtocolGuid                                ## TO_START
  gEfiUdp6ServiceBindingProtocolGuid                   ## TO_START
  gEfiUdp6ProtocolGuid                                 ## TO_START
  gEfiMtftp6ServiceBindingProtocolGuid                 ## TO_START
  gEfiMtftp6ProtocolGuid                               ## TO_START
  gEfiDhcp6ServiceBindingProtocolGuid                  ## TO_START
  gEfiDhcp6ProtocolGuid                                ## TO_START
  gEfiDns6ServiceBindingProtocolGuid                   ## SOMETIMES_CONSUMES
  gEfiDns6ProtocolGuid                                 ## SOMETIMES_CONSUMES
  gEfiPxeBaseCodeCallbackProtocolGuid                  ## SOMETIMES_PRODUCES
  gEfiPxeBaseCodeProtocolGuid                          ## BY_START
  gEfiLoadFileProtocolGuid                             ## BY_START
  gEfiAdapterInformationProtocolGuid                   ## SOMETIMES_CONSUMES
```

收起 ^

BY_START here means to generate Protocol. You can find their installation code directly in NetworkPkg\UefiPxeBcDxe\PxeBcDriver.c:

cpp AI generated projects 登录复制 run

```cpp
  //
  // Create a new handle for IPv4 virtual nic,
  // and install PxeBaseCode, LoadFile and DevicePath protocols.
  //
  Status = gBS->InstallMultipleProtocolInterfaces (
                  &Private->Ip4Nic->Controller,
                  &gEfiDevicePathProtocolGuid,
                  Private->Ip4Nic->DevicePath,
                  &gEfiLoadFileProtocolGuid,
                  &Private->Ip4Nic->LoadFile,
                  &gEfiPxeBaseCodeProtocolGuid,
                  &Private->PxeBc,
                  NULL
                  );
```

收起 ^

In fact, it will be installed twice, corresponding to IPv4 and IPv6. Let's take IPv4 as an example. You can see that in addition to installing the two protocols mentioned above, the Device Path Protocol is also installed. It is installed as a general requirement and will not be specifically explained here.

cpp AI generated projects 登录复制 run

```cpp
///
/// The EFI_LOAD_FILE_PROTOCOL is a simple protocol used to obtain files from arbitrary devices.
///
struct _EFI_LOAD_FILE_PROTOCOL {
  EFI_LOAD_FILE LoadFile;
};
```

It is an interface for obtaining files. In fact, PXE is just a process of obtaining BootLoader through the network, so the interface here is also consistent. It should also be noted that in fact, HTTP boot also implements this interface, because it also only obtains BootLoader through the network.

The implementation of LoadFile is as follows (NetworkPkg\UefiPxeBcDxe\PxeBcImpl.c):

cpp AI generated projects 登录复制 run

```cpp
EFI_LOAD_FILE_PROTOCOL  gLoadFileProtocolTemplate = { EfiPxeLoadFile };
```

Its specific implementation is based on PxeBc .

The Protocol corresponding to gEfiPxeBaseCodeProtocolGuid is slightly more complicated (MdePkg\Include\Protocol\PxeBaseCode.h):

cpp AI generated projects 登录复制 run

```cpp
///
/// The EFI_PXE_BASE_CODE_PROTOCOL is used to control PXE-compatible devices.
/// An EFI_PXE_BASE_CODE_PROTOCOL will be layered on top of an
/// EFI_MANAGED_NETWORK_PROTOCOL protocol in order to perform packet level transactions.
/// The EFI_PXE_BASE_CODE_PROTOCOL handle also supports the
/// EFI_LOAD_FILE_PROTOCOL protocol. This provides a clean way to obtain control from the
/// boot manager if the boot path is from the remote device.
///
struct _EFI_PXE_BASE_CODE_PROTOCOL {
  ///
  ///  The revision of the EFI_PXE_BASE_CODE_PROTOCOL. All future revisions must
  ///  be backwards compatible. If a future version is not backwards compatible
  ///  it is not the same GUID.
  ///
  UINT64                          Revision;
  EFI_PXE_BASE_CODE_START         Start;
  EFI_PXE_BASE_CODE_STOP          Stop;
  EFI_PXE_BASE_CODE_DHCP          Dhcp;
  EFI_PXE_BASE_CODE_DISCOVER      Discover;
  EFI_PXE_BASE_CODE_MTFTP         Mtftp;
  EFI_PXE_BASE_CODE_UDP_WRITE     UdpWrite;
  EFI_PXE_BASE_CODE_UDP_READ      UdpRead;
  EFI_PXE_BASE_CODE_SET_IP_FILTER SetIpFilter;
  EFI_PXE_BASE_CODE_ARP           Arp;
  EFI_PXE_BASE_CODE_SET_PARAMETERS SetParameters;
  EFI_PXE_BASE_CODE_SET_STATION_IP SetStationIp;
  EFI_PXE_BASE_CODE_SET_PACKETS   SetPackets;
  ///
  /// The pointer to the EFI_PXE_BASE_CODE_MODE data for this device.
  ///
  EFI_PXE_BASE_CODE_MODE          *Mode;
};
```

收起 ^

It actually needs to complete network communication related actions, such as DHCP /TFPT, etc. The implementation of PxeBc is as follows (NetworkPkg\UefiPxeBcDxe\PxeBcImpl.c):

cpp AI generated projects 登录复制 run

```cpp
EFI_PXE_BASE_CODE_PROTOCOL  gPxeBcProtocolTemplate = {
  EFI_PXE_BASE_CODE_PROTOCOL_REVISION,
  EfiPxeBcStart,
  EfiPxeBcStop,
  EfiPxeBcDhcp,
  EfiPxeBcDiscover,
  EfiPxeBcMtftp,
  EfiPxeBcUdpWrite,
  EfiPxeBcUdpRead,
  EfiPxeBcSetIpFilter,
  EfiPxeBcArp,
  EfiPxeBcSetParameters,
  EfiPxeBcSetStationIP,
  EfiPxeBcSetPackets,
  NULL
};
```

收起 ^

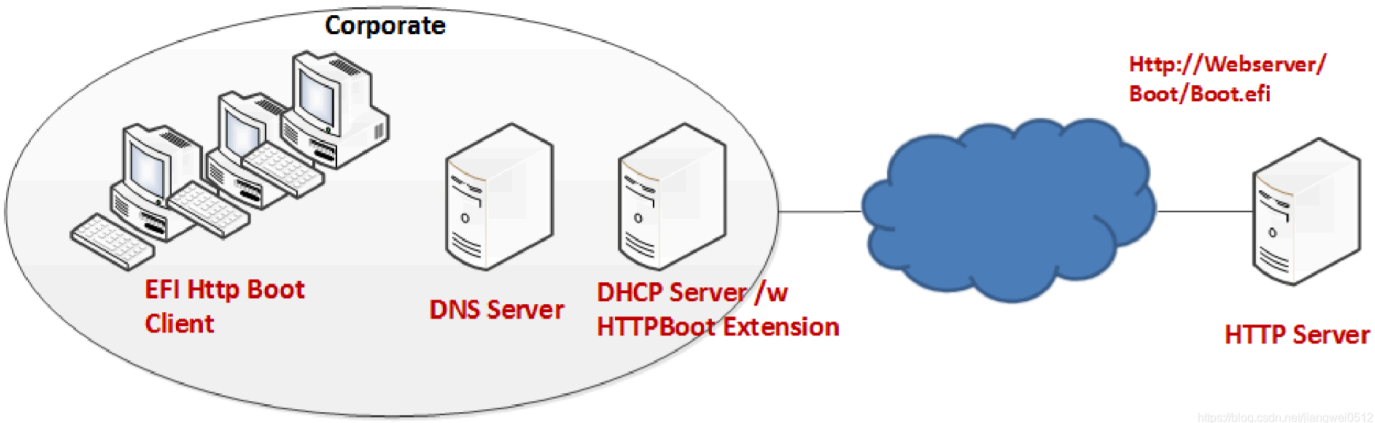
The relationship between PxeBc and LoadFile is connected through the following structure (NetworkPkg\UefiPxeBcDxe\PxeBcImpl.h) :

cpp AI generated projects 登录复制 run

```cpp
struct _PXEBC_VIRTUAL_NIC {
  UINT32                                Signature;
  EFI_HANDLE                            Controller;
  EFI_LOAD_FILE_PROTOCOL                LoadFile;
  EFI_DEVICE_PATH_PROTOCOL              *DevicePath;
  PXEBC_PRIVATE_DATA                    *Private;
};
```

HTTP Startup

Although HTTP itself is not a new thing, it is newer than PXE in network booting. Compared with PXE using TFTP to download Boot Loader, HTTP booting of course uses HTTP to download Boot Loader. Although there is no essential difference between the two, HTTP has a wider range of applicability. The following is the environment diagram of HTTP booting:



The BootLoader is located on the HTTP server, not the TFTP server for PXE booting. HTTP uses URI to access the server device, while PXE uses IP to access the server device.

HTTP boot is slightly more complex to implement than PXE boot, involving DNS/DHCP/URL etc. But the key is the NetworkPkg\HttpBootDxe\HttpBootDxe.inf module, which implements EFI_LOAD_FILE_PROTOCOL (NetworkPkg\HttpBootDxe\HttpBootDxe.c):

cpp AI generated projects 登录复制 run

```cpp
  //
  // Create a child handle for the HTTP boot and install DevPath and Load file protocol on it.
  //
  CopyMem (&Private->Ip4Nic->LoadFile, &gHttpBootDxeLoadFile, sizeof (EFI_LOAD_FILE_PROTOCOL));
  Status = gBS->InstallMultipleProtocolInterfaces (
                  &Private->Ip4Nic->Controller,
                  &gEfiLoadFileProtocolGuid,
                  &Private->Ip4Nic->LoadFile,
                  &gEfiDevicePathProtocolGuid,
                  Private->Ip4Nic->DevicePath,
                  NULL
                  );
  if (EFI_ERROR (Status)) {
    goto ON_ERROR;
  }
```

收起 ^

It is also divided into two types: IPv4 and IPv6. The corresponding LoadFile implementation (NetworkPkg\HttpBootDxe\HttpBootImpl.c):

cpp AI generated projects 登录复制 run

```cpp
///
/// Load File Protocol instance
///
GLOBAL_REMOVE_IF_UNREFERENCED
EFI_LOAD_FILE_PROTOCOL  gHttpBootDxeLoadFile = {
  HttpBootDxeLoadFile
};
```

There are also some protocols used to handle HTTP communication, such as:

cpp AI generated projects 登录复制 run

```cpp
///
/// EFI_HTTP_UTILITIES_PROTOCOL
/// designed to be used by EFI drivers and applications to parse HTTP
/// headers from a byte stream. This driver is neither dependent on
/// network connectivity, nor the existence of an underlying network
/// infrastructure.
///
struct _EFI_HTTP_UTILITIES_PROTOCOL {
  EFI_HTTP_UTILS_BUILD          Build;
  EFI_HTTP_UTILS_PARSE          Parse;
};
///
/// The EFI_DNS4_Protocol provides the function to get the host name and address
/// mapping, also provides pass through interface to retrieve arbitrary information
/// from DNS.
///
struct _EFI_DNS4_PROTOCOL {
  EFI_DNS4_GET_MODE_DATA        GetModeData;
  EFI_DNS4_CONFIGURE            Configure;
  EFI_DNS4_HOST_NAME_TO_IP      HostNameToIp;
  EFI_DNS4_IP_TO_HOST_NAME      IpToHostName;
  EFI_DNS4_GENERAL_LOOKUP       GeneralLookUp;
  EFI_DNS4_UPDATE_DNS_CACHE     UpdateDnsCache;
  EFI_DNS4_POLL                 Poll;
  EFI_DNS4_CANCEL               Cancel;
};
```

收起 ^

For HTTP startup, you can also refer to: https://github.com/tianocore/tianocore.github.io/wiki/HTTP-Boot .

Network startup items

After briefly introducing HTTP boot and PXE boot, the next thing to focus on is how to use these boot implementations. In fact, it has been introduced before, which is mainly done through EFI_LOAD_FILE_PROTOCOL . For network boot, the general code is as follows:

cpp AI generated projects 登录复制 run

```cpp
  Status = gBS->LocateHandleBuffer (ByProtocol, &gEfiLoadFileProtocolGuid, NULL, &HandleCount, &Handles);
  if (EFI_ERROR (Status)) {
    HandleCount = 0;
    Handles = NULL;
  }
  NextFullPath = NULL;
  GetNext = (BOOLEAN)(FullPath == NULL);
  for (Index = 0; Index < HandleCount; Index++) {
    NextFullPath = BmExpandLoadFile (Handles[Index], FilePath);
```

Here we get all the implemented EFI_LOAD_FILE_PROTOCOL , and for each Protocol, call its LoadFile to get the file. However, it should be noted that because the addresses used for HTTP boot and PXE boot are different, the corresponding Device Path is also different:

```cpp
/**
  Causes the driver to load a specified file.
  @param  This       Protocol instance pointer.
  @param  FilePath   The device specific path of the file to load.
  @param  BootPolicy If TRUE, indicates that the request originates from the
                     boot manager is attempting to load FilePath as a boot
                     selection. If FALSE, then FilePath must match as exact file
                     to be loaded.
  @param  BufferSize On input the size of Buffer in bytes. On output with a return
                     code of EFI_SUCCESS, the amount of data transferred to
                     Buffer. On output with a return code of EFI_BUFFER_TOO_SMALL,
                     the size of Buffer required to retrieve the requested file.
  @param  Buffer     The memory buffer to transfer the file to. IF Buffer is NULL,
                     then the size of the requested file is returned in
                     BufferSize.

  @retval EFI_SUCCESS           The file was loaded.
  @retval EFI_UNSUPPORTED       The device does not support the provided BootPolicy
  @retval EFI_INVALID_PARAMETER FilePath is not a valid device path, or
                                BufferSize is NULL.
  @retval EFI_NO_MEDIA          No medium was present to load the file.
```

```cpp
  @retval EFI_DEVICE_ERROR      The file was not loaded due to a device error.
  @retval EFI_NO_RESPONSE       The remote system did not respond.
  @retval EFI_NOT_FOUND         The file was not found.
  @retval EFI_ABORTED           The file load process was manually cancelled.
  @retval EFI_WARN_FILE_SYSTEM  The resulting Buffer contains UEFI-compliant file system.
**/

typedef
EFI_STATUS
(EFIAPI *EFI_LOAD_FILE)(
  IN EFI_LOAD_FILE_PROTOCOL           *This,
  IN EFI_DEVICE_PATH_PROTOCOL         *FilePath,
  IN BOOLEAN                          BootPolicy,
  IN OUT UINTN                        *BufferSize,
  IN VOID                             *Buffer OPTIONAL
  );
```

收起 ⌃

That is, the FilePath here has different values. The Device Path structure corresponding to HTTP is:

cpp AI generated projects 登录复制 run

```cpp
///
/// Uniform Resource Identifiers (URI) Device Path SubType
///
#define MSG_URI_DP               0x18
typedef struct {
  EFI_DEVICE_PATH_PROTOCOL        Header;
  ///
  /// Instance of the URI pursuant to RFC 3986.
  ///
  CHAR8                           Uri[];
} URI_DEVICE_PATH;
```

收起 ⌃

Device Path will be used in the implementation of HttpBootDxeLoadFile :

cpp AI generated projects 登录复制 run

```cpp
EFI_STATUS
EFIAPI
HttpBootDxeLoadFile (
  IN EFI_LOAD_FILE_PROTOCOL           *This,
  IN EFI_DEVICE_PATH_PROTOCOL         *FilePath,
  IN BOOLEAN                          BootPolicy,
  IN OUT UINTN                        *BufferSize,
  IN VOID                             *Buffer OPTIONAL
```

展开 ⌄

PXE boot does not require a special Device Path:

cpp AI generated projects 登录复制 run

```cpp
EFI_STATUS
EFIAPI
EfiPxeLoadFile (
  IN     EFI_LOAD_FILE_PROTOCOL           *This,
  IN     EFI_DEVICE_PATH_PROTOCOL         *FilePath,
  IN     BOOLEAN                          BootPolicy,
  IN OUT UINTN                            *BufferSize,
  IN     VOID                             *Buffer       OPTIONAL
  )
{
  PXEBC_PRIVATE_DATA          *Private;
  PXEBC_VIRTUAL_NIC           *VirtualNic;
  EFI_PXE_BASE_CODE_PROTOCOL  *PxeBc;
  BOOLEAN                     UsingIpv6;
  EFI_STATUS                  Status;
  BOOLEAN                     MediaPresent;

  if (FilePath == NULL || !IsDevicePathEnd (FilePath)) {
    return EFI_INVALID_PARAMETER;
  }
```

收起 ⌃

Here, only a judgment is made, but it is not actually used. Except for the Device Path, there is no big difference between the two calls to LoadFile .