[UEFI Practice] UEFI Graphics Display (Character Output)

jiangwei0512 Modified on 2023-08-11 10:22:00 Read 2k Collection 6 Likes 3

Copyright CC 4.0 BY-SA

Category Column: UEFI Development Basics Article Tags: uefi

UEFI Development … This column includes this content 136 articles Subscribe to our column

HII Font

Next `EFI_HII_FONT_PROTOCOL` , it is introduced that it completes the conversion from characters to pixels in the UEFI `EFI_HII_FONT_PROTOCOL` code. This section mainly introduces this conversion relationship. Its implementation code is in edk2\MdeModulePkg\Universal\HiiDatabaseDxe\HiiDatabaseDxe.inf. In addition , this module also implements many other protocols , which will be introduced later when used. Therefore, the HiiDatabaseDxe.inf module is not only the basis of fonts, but also the basic module of the UEFI human-machine interface (Human Interface Infrastructure).

Structural organization

The following figure shows the font-related structure in the HII database foundation :

As can be seen from the figure, there are two types of fonts, one is the normal version and the other is the simplified version, which will be introduced separately in this section.

The structure on the far left `HII_DATABASE_PRIVATE_DATA` is the basic structure of the HII database module, corresponding to the font part:

c AI generated projects 登录复制 run

```c
HII_DATABASE_PRIVATE_DATA  mPrivate = {
  // 前略
  LIST_ENTRY                        DatabaseList;  // 所有HII资源都放在这个列表中，其中就包括字体和简单字体
  // 前略
  {
    HiiStringToImage,
    HiiStringIdToImage,
    HiiGetGlyph,
    HiiGetFontInfo
  },
  // 中略
  LIST_ENTRY                        FontInfoList; // global font info list
  // 后略
};
```

收起 ^

It contains a Protocol, corresponding to `EFI_HII_FONT_PROTOCOL` its interface:

c AI generated projects 登录复制 run

```c
///
/// The protocol provides the service to retrieve the font informations.
///
struct _EFI_HII_FONT_PROTOCOL {

```

```
  EFI_HII_STRING_TO_IMAGE       StringToImage;
  EFI_HII_STRING_ID_TO_IMAGE    StringIdToImage;
  EFI_HII_GET_GLYPH             GetGlyph;
  EFI_HII_GET_FONT_INFO         GetFontInfo;
};
```

This will be introduced later.

It also contains a database list `DatabaseList`, which contains all HII resources, including the registered font part. And a font information list `FontInfoList`, which points to an additional structure `HII_GLOBAL_FONT_INFO`, which contains the data needed by ordinary fonts, but not needed by simple fonts. Because it is a list, it means that UEFI can support multiple font information.

`HII_GLOBAL_FONT_INFO` The structure is as follows:

c AI generated projects 登录复制 run

```c
typedef struct _HII_GLOBAL_FONT_INFO {
  UINTN                         Signature;
  LIST_ENTRY                    Entry;
  HII_FONT_PACKAGE_INSTANCE     *FontPackage;
  UINTN                         FontInfoSize;
  EFI_FONT_INFO                 *FontInfo;
} HII_GLOBAL_FONT_INFO;
```

The important thing is the following three members, which represent two types of information, Font and `FontPackage` Glyph `FontInfo`. They contain all the information about fonts and glyphs. Special explanation is needed about fonts and glyphs:

- Font is the overall expression of all characters, such as Kaiti, Songti, etc., which represents an overall concept.
- Glyphs are what each character looks like, and actually involve how a character is drawn under UEFI.
- All glyphs make up a font style.

The following is a brief introduction to the structures related to font information. First is the structure that describes the font itself:

c AI generated projects 登录复制 run

```c
typedef struct {
  EFI_HII_FONT_STYLE    FontStyle;  ///< 一个枚举值，表示的是粗体、斜体等
  UINT16                FontSize;   ///< character cell height in pixels
  CHAR16                FontName[1];///< 字体名，这个在文本编辑器里面更容易看到
} EFI_FONT_INFO;
```

Currently supported font types (corresponding `FontStyle`):

c AI generated projects 登录复制 run

```c
//
// Value for font style
//
#define EFI_HII_FONT_STYLE_NORMAL     0x00000000
#define EFI_HII_FONT_STYLE_BOLD       0x00000001
#define EFI_HII_FONT_STYLE_ITALIC     0x00000002
#define EFI_HII_FONT_STYLE_EMBOSS     0x00010000
#define EFI_HII_FONT_STYLE_OUTLINE    0x00020000
#define EFI_HII_FONT_STYLE_SHADOW     0x00040000
#define EFI_HII_FONT_STYLE_UNDERLINE  0x00080000
#define EFI_HII_FONT_STYLE_DBL_UNDER  0x00100000
```

收起 ^

Then there is the structure of a single glyph:

c AI generated projects 登录复制 run

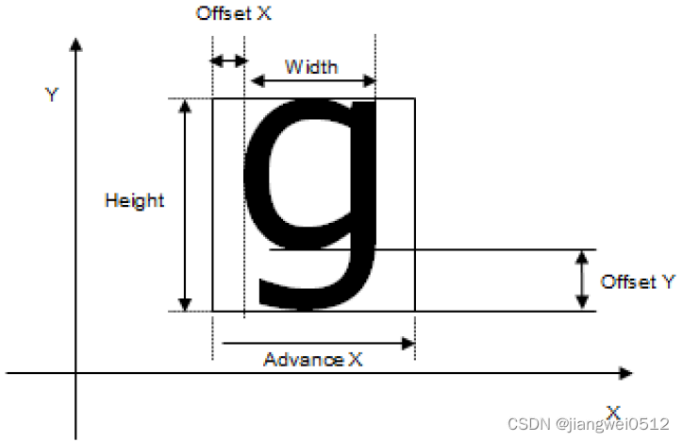
```c
typedef struct _HII_GLYPH_INFO {
  UINTN                 Signature;
  LIST_ENTRY            Entry;
  CHAR16                CharId;
  EFI_HII_GLYPH_INFO    Cell;
} HII_GLYPH_INFO;
```

There is a list here `Entry` because there need to be many glyphs, `CharId` which represent character encoding values. In `CHAR16` theory, a character that can cover all commonly used languages is used. The final `Cell` structure is as follows:

c AI generated projects 登录复制 run

```c
typedef struct _EFI_HII_GLYPH_INFO {
  UINT16    Width;
  UINT16    Height;
  INT16     OffsetX;
  INT16     OffsetY;
  INT16     AdvanceX;
} EFI_HII_GLYPH_INFO;
```

The corresponding relationship between these values and characters is shown below:



The remaining structures `EFI_HII_FONT_PACKAGE_HDR` are `HII_FONT_PACKAGE_INSTANCE` mainly descriptions of the organization of the above font and glyph structures.

It should be noted that so far only the structures related to fonts and glyphs have been described, and `EFI_GRAPHICS_OUTPUT_PROTOCOL` the data conversion model from characters to displayable graphics has not been actually described. This part of the content is hidden in a certain structure member in the front. Mainly `GlyphBlock` , as a structure member it is just a pointer, through which we can find the image display data corresponding to a certain character, so its design is more important, because we need to quickly find the image corresponding to the character.

In fact, due to usage restrictions, the application of fonts and glyphs is not very important under UEFI. UEFI only needs to be able to clearly represent characters. For this reason, there is a simplified version of the font. Its structure is quite simple. The important thing is the following structure:

c AI generated projects 登录复制 run

```c
///
/// A simplified font package consists of a font header
/// followed by a series of glyph structures.
///
typedef struct _EFI_HII_SIMPLE_FONT_PACKAGE_HDR {
  EFI_HII_PACKAGE_HEADER    Header;
  UINT16                    NumberOfNarrowGlyphs;
  UINT16                    NumberOfWideGlyphs;
  // EFI_NARROW_GLYPH       NarrowGlyphs[];
  // EFI_WIDE_GLYPH         WideGlyphs[];
} EFI_HII_SIMPLE_FONT_PACKAGE_HDR;
```

收起 ^

Although the structure contains the word HDR, it can be seen from the above code that it directly receives all the data, through which the characters can be directly converted into output graphics. The subsequent data contains two parts, corresponding to narrow characters and wide characters, which actually correspond to 8x19 and 16x19 forms. Taking narrow characters as an example, the data corresponding to a character is as follows:

c AI generated projects 登录复制 run

```c
typedef struct {
  CHAR16                UnicodeWeight;
  UINT8                 Attributes;
  UINT8                 GlyphCol1[EFI_GLYPH_HEIGHT];   // EFI_GLYPH_HEIGHT = 19
} EFI_NARROW_GLYPH;
```

`UnicodeWeight` Corresponds to the computer representation of the character, `Attributes` indicating the properties of the character. Currently supported values are:

c AI generated projects 登录复制 run

```c
///
/// Contents of EFI_NARROW_GLYPH.Attributes.
///@{
#define EFI_GLYPH_NON_SPACING  0x01
#define EFI_GLYPH_WIDE         0x02
#define EFI_GLYPH_HEIGHT       19
#define EFI_GLYPH_WIDTH        8
///@}
```

`GlyphCol1` It is actually a bitmap, and each bit indicates whether the pixel needs to be drawn. In the code example of [UEFI Actual Combat] UEFI Graphics Display (From Pixels to Characters) , a two-dimensional array is used to indicate whether the corresponding pixel needs to be drawn:

c AI generated projects 登录复制 run

```c
  UINT8 BltIndex[NARROW_HEIGHT * NARROW_WIDTH] = {
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 1, 0, 0, 0, 0,
    0, 0, 1, 1, 1, 0, 0, 0,
    0, 1, 1, 0, 1, 1, 0, 0,
    1, 1, 0, 0, 0, 1, 1, 0,
    1, 1, 0, 0, 0, 1, 1, 0,
    1, 1, 0, 0, 0, 1, 1, 0,
    1, 1, 0, 0, 0, 1, 1, 0,
    1, 1, 1, 1, 1, 1, 1, 0,
    1, 1, 0, 0, 0, 1, 1, 0,
    1, 1, 0, 0, 0, 1, 1, 0,
    1, 1, 0, 0, 0, 1, 1, 0,
    1, 1, 0, 0, 0, 1, 1, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0
  };
```

收起 ^

Obviously the method here is simpler and more compact than the above code.

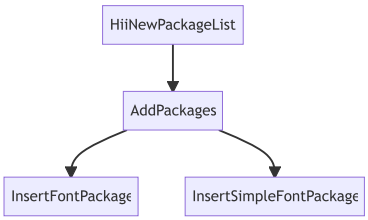
The open source EDK code uses this simplified font form to represent characters.

Character to glyph representation

Registering character description data to glyphs involves an interface of HiiDataBase:

```c
///
/// Database manager for HII-related data structures.
///
struct _EFI_HII_DATABASE_PROTOCOL {
  EFI_HII_DATABASE_NEW_PACK            NewPackageList;
  // 后略
};
```

Here we will not write its function prototype, but directly focus on the most important part of its implementation for fonts. The calling process is as follows:



The location of the font registration data is in the GraphicsConsoleDxe module introduced earlier:

```c
  //
  // Add 4 bytes to the header for entire length for HiiAddPackages use only.
  //
  //    +--------------------------------+ <-- Package
  //    |                                |
  //    |    PackageLength(4 bytes)      |
  //    |                                |
  //    |--------------------------------| <-- SimplifiedFont
  //    |                                |
  //    |EFI_HII_SIMPLE_FONT_PACKAGE_HDR |
  //    |                                |
  //    |--------------------------------| <-- Location
  //    |                                |
  //    |     gUsStdNarrowGlyphData      |
  //    |                                |
  //    +--------------------------------+

  PackageLength = sizeof (EFI_HII_SIMPLE_FONT_PACKAGE_HDR) + mNarrowFontSize + 4;
  Package       = AllocateZeroPool (PackageLength);
  ASSERT (Package != NULL);
twen
  WriteUnaligned32 ((UINT32 *)Package, PackageLength);
  SimplifiedFont                       = (EFI_HII_SIMPLE_FONT_PACKAGE_HDR *)(Package + 4);
  SimplifiedFont->Header.Length        = (UINT32)(PackageLength - 4);
  SimplifiedFont->Header.Type          = EFI_HII_PACKAGE_SIMPLE_FONTS;
  SimplifiedFont->NumberOfNarrowGlyphs = (UINT16)(mNarrowFontSize / sizeof (EFI_NARROW_GLYPH));

  Location = (UINT8 *)(&SimplifiedFont->NumberOfWideGlyphs + 1);
  CopyMem (Location, gUsStdNarrowGlyphData, mNarrowFontSize);

  //
  // Add this simplified font package to a package list then install it.
  //
  mHiiHandle = HiiAddPackages (
                 &mFontPackageListGuid,
                 NULL,
                 Package,
                 NULL
                 );
  ASSERT (mHiiHandle != NULL);
  FreePool (Package);
```

收起 ^

The specific data is all stored `gUsStdNarrowGlyphData` in an array, which is located in a separate file edk2\MdeModulePkg\Universal\Console\GraphicsConsoleDxe\LaffStd.c:

```c
EFI_NARROW_GLYPH  gUsStdNarrowGlyphData[] = {
  //
  // Unicode glyphs from 0x20 to 0x7e are the same as ASCII characters 0x20 to 0x7e
  //
  { 0x0020,                                 0x00, { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
  },
  { 0x0021,                                 0x00, { 0x00, 0x00, 0x00, 0x18, 0x3C, 0x3C, 0x3C, 0x18, 0x18, 0x18, 0x18, 0x18, 0x00, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00 }
  },
  // 后面还有很多的数据
```

收起 ^

In fact, since English only needs to support ASCII, plus some specific graphics, the final data will not be too much.

EFI_HII_FONT_PROTOCOL

The following describes the font operation function, which corresponds to `EFI_HII_FONT_PROTOCOL` the following form:

```c
///
/// The protocol provides the service to retrieve the font informations.
///
struct _EFI_HII_FONT_PROTOCOL {
  EFI_HII_STRING_TO_IMAGE       StringToImage;
  EFI_HII_STRING_ID_TO_IMAGE    StringIdToImage;
  EFI_HII_GET_GLYPH             GetGlyph;
  EFI_HII_GET_FONT_INFO         GetFontInfo;
};
```

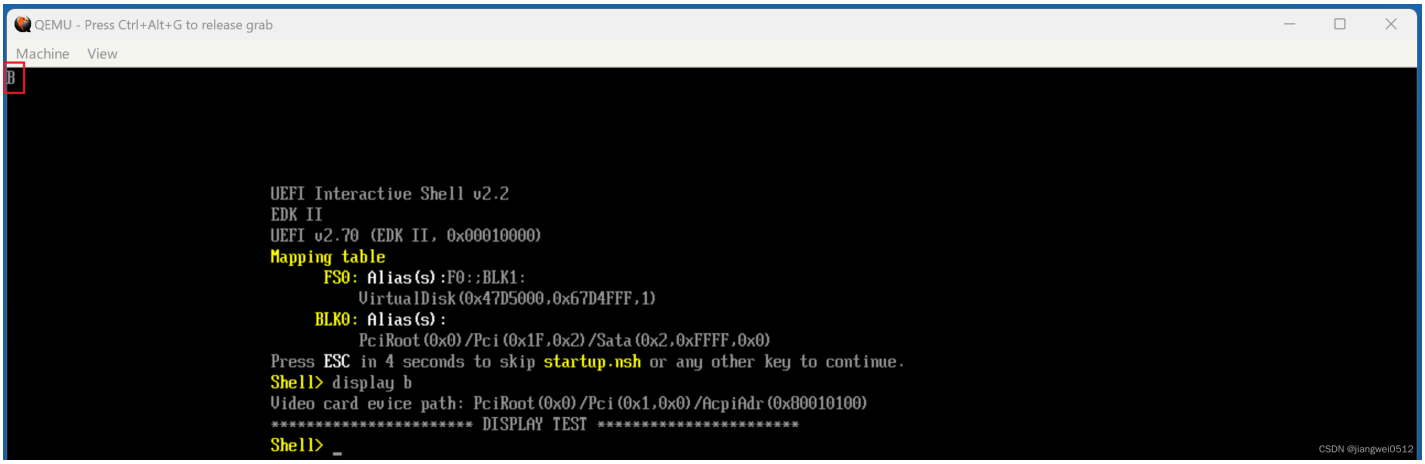
The first two are direct display functions, the difference is that `StringToImage` they directly output strings, and `StringIdToImage` according to the data created by the uni file, `StringId` find the string and output it. The last two functions get the font and glyph information. [UEFI Actual Combat] In the example of UEFI graphic display (from pixels to characters), an A was written by manually building glyphs, but here you can `GetGlyph()` get the glyph and output it by, no longer needing to build it manually. Here is an example:

```c
VOID
ShowB (
  IN  EFI_HII_FONT_PROTOCOL         *HiiFont,
  IN  EFI_GRAPHICS_OUTPUT_PROTOCOL  *Gop
  )
{
  EFI_STATUS        Status = EFI_ABORTED;
  EFI_IMAGE_OUTPUT  *Blt = NULL;

  Status = HiiFont->GetGlyph (
                      HiiFont,
                      L'B',
                      NULL,
                      &Blt,
                      NULL
                      );
  if (EFI_ERROR (Status)) {
    DEBUG ((EFI_D_ERROR, "[%a][%d] Failed. - %r\n", __FUNCTION__, __LINE__, Status));
  } else {
    Gop->Blt (
          Gop,
          Blt->Image.Bitmap,
          EfiBltBufferToVideo,
          0,
          0,
          0,
          0,
          Blt->Width,
          Blt->Height,
          0
          );
  }
}
```

收起

The output is:



Chinese character output

We have previously introduced how to output English. Here we will briefly explain how to use Chinese characters under UEFI. The principle itself is relatively simple. English output uses narrow fonts, which is 8x19 pixels. This is obviously not enough for Chinese characters, so the UEFI specification also defines wide fonts:

```c
///
/// The EFI_WIDE_GLYPH has a preferred dimension (w x h) of 16 x 19 pixels, which is large enough
/// to accommodate logographic characters.
///
typedef struct {
  ///
  /// The Unicode representation of the glyph. The term weight is the
  /// technical term for a character code.
  ///
  CHAR16    UnicodeWeight;
  ///
  /// The data element containing the glyph definitions.
  ///
  UINT8     Attributes;
  ///
  /// The column major glyph representation of the character. Bits
```

```
  /// with values of one indicate that the corresponding pixel is to be
  /// on when normally displayed; those with zero are off.
  ///
  UINT8     GlyphCol1[EFI_GLYPH_HEIGHT];
  ///
  /// The column major glyph representation of the character. Bits
  /// with values of one indicate that the corresponding pixel is to be
  /// on when normally displayed; those with zero are off.
  ///
  UINT8     GlyphCol2[EFI_GLYPH_HEIGHT];
  ///
  /// Ensures that sizeof (EFI_WIDE_GLYPH) is twice the
  /// sizeof (EFI_NARROW_GLYPH). The contents of Pad must
  /// be zero.
  ///
  UINT8     Pad[3];
} EFI_WIDE_GLYPH;
```

收起 ^

Compared with narrow-body fonts, the key point is to use two 8x19 pixels, each of which outputs half a Chinese character, to finally form a complete Chinese character. The focus afterwards is how to construct the pixels of a Chinese character. This has been provided in uefi-programming/book/GUIbasics/font/SimpleFont/createdata.html at master · zhenghuadai/uefi-programming · GitHub. Tools can be used to create a complete `EFI_WIDE_GLYPH` array to represent all Chinese characters (since many fonts are copyrighted, you need to be careful when using them, and it is best to refer to the free font construction yourself). Here, the tool is used directly to get the array, and then the Chinese characters are registered in the same way as the narrow-body fonts. The corresponding code is:

c AI generated projects 登录复制 run

```c
  //
  // Reference:
  // edk2\MdeModulePkg\Universal\Console\GraphicsConsoleDxe\GraphicsConsole.c
  //
  PackageLength = sizeof (EFI_HII_SIMPLE_FONT_PACKAGE_HDR) + mWideFontSize + 4;
  Package       = AllocateZeroPool (PackageLength);
  if (NULL == Package) {
    DEBUG ((EFI_D_ERROR, "[%a][%d] Out of memory\n", __FUNCTION__, __LINE__));
    return EFI_OUT_OF_RESOURCES;
  }

  //
  // Without this code, system will hang.
  //
  WriteUnaligned32 ((UINT32 *)Package, PackageLength);

  SimplifiedFont                      = (EFI_HII_SIMPLE_FONT_PACKAGE_HDR *)(Package + 4);
  SimplifiedFont->Header.Length       = (UINT32)(PackageLength - 4);
  SimplifiedFont->Header.Type         = EFI_HII_PACKAGE_SIMPLE_FONTS;
  SimplifiedFont->NumberOfNarrowGlyphs = 0;
  SimplifiedFont->NumberOfWideGlyphs = (UINT16) (mWideFontSize / sizeof (EFI_WIDE_GLYPH));

  Location = (UINT8 *)(&SimplifiedFont->NumberOfWideGlyphs + 1);
  CopyMem (Location, gWideGlyphData, mWideFontSize);

  mHiiHandle = HiiAddPackages (
                &mFontPackageListGuid,
                NULL,
                Package,
                NULL
                );
  if (NULL == mHiiHandle) {
    DEBUG ((EFI_D_ERROR, "[%a][%d] NULL == mHiiHandle\n", __FUNCTION__, __LINE__));
    Status = EFI_NOT_READY;
  } else {
    DEBUG ((EFI_D_ERROR, "HanFont added\n"));
  }

  FreePool (Package);
```

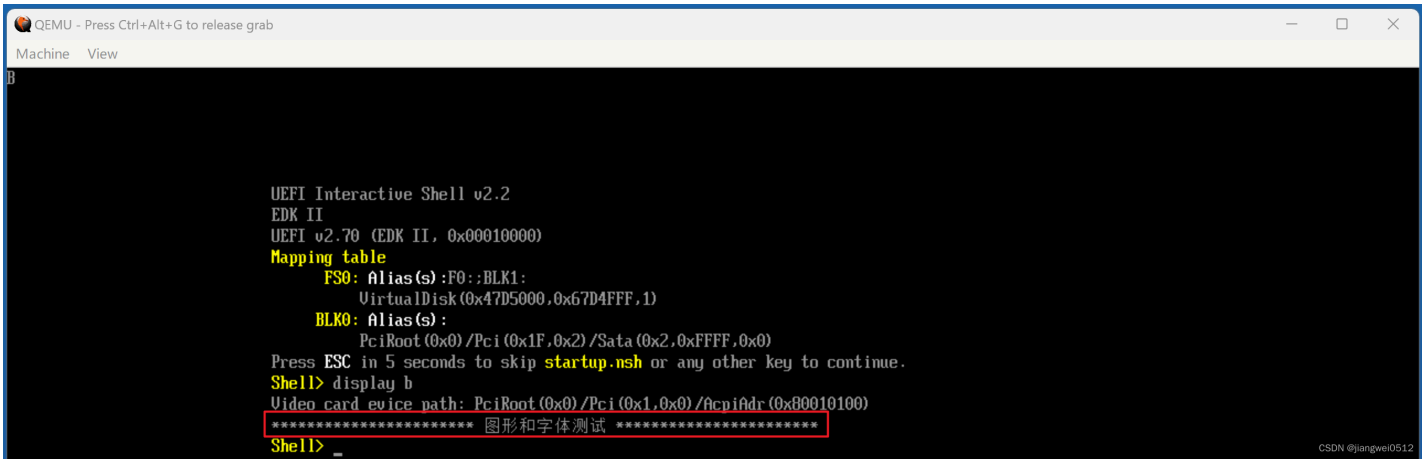
收起 ^

`gWideGlyphData` All Chinese character glyphs are stored in it. Since there is too much data, it will not be listed here. You can use it directly after registration. The following is a code example:

c AI generated projects 登录复制 run

```c
Print (L"********************** 图形和字体测试 **********************\r\n");
```

The result is:



However, you need to pay attention to ensure that the corresponding c file uses the GBK (or other Chinese character encoding format) encoding format when compiling, otherwise the compilation will fail:

```
SearchString: Error while processing file
```

about Us Careers Business Cooperation Seeking coverage 400-660-0108 kefu@csdn.net Online Customer Service Working hours 8:30-22:00
Public Security Registration Number 11010502030143 Beijing ICP No. 19004658 Beijing Internet Publishing House [2020] No. 1039-165
Commercial website registration information Beijing Internet Illegal and Harmful Information Reporting Center Parental Control
Online 110 Alarm Service China Internet Reporting Center Chrome Store Download Account Management Specifications
Copyright and Disclaimer Copyright Complaints Publication License Business license
©1999-2025 Beijing Innovation Lezhi Network Technology Co., Ltd.