[UEFI Practice] JSON under BIOS

原创 jiangwei0512 Posted on 2023-03-19 23:27:17 Read 1.4k Collection 2 Likes 3

Copyright CC 4.0 BY-SA

Category Column: UEFI Development Basics Article Tags: uefi json

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

JSON in a nutshell

Description of JSON:

- JSON is a format for storing and transmitting data.
- JSON is often used when sending data from a server to a web page.
- JSON refers to JavaScript Object Notation.
- JSON is a lightweight data interchange format.
- JSON is language independent.
- JSON is "self-describing" and easy to understand.

When data is exchanged between the browser and the server, it can only be text. JSON is text, and we can convert any JavaScript object to JSON and then send JSON to the server. We can also convert any JSON received from the server to a JavaScript object.

JSON uses JavaScript syntax, but the JSON format is plain text. The text can be read and used as data by any programming language.

JSON and BIOS

Currently, JSON is already used in the open source code of EDK, which is reflected in RedfishPkg:

bash AI generated projects 登录复制

```bash
PS D:\Gitee\edk2-beni\edk2> git submodule
-b64af41c3276f97f0e181920400ee056b9c88037 ArmPkg/Library/ArmSoftFloatLib/berkeley-softfloat-3
-f4153a09f87cbb9c826d8fc12c74642bb2d879ea BaseTools/Source/C/BrotliCompress/brotli
-d82e959e621a3d597f1e0d50ff8c2d8b96915fd7 CryptoPkg/Library/OpensslLib/openssl
-f4153a09f87cbb9c826d8fc12c74642bb2d879ea MdeModulePkg/Library/BrotliCustomDecompressLib/brotli
-abfc8ff81df4067f309032467785e06975678f0d MdeModulePkg/Universal/RegularExpressionDxe/oniguruma
-e9ebfa7e77a6bee77df44e096b100e7131044059 RedfishPkg/Library/JsonLib/jansson  # RedfishPkg中包含的子模块
-1cc9cde3448cdd2e000886a26acf1caac2db7cf1 UnitTestFrameworkPkg/Library/CmockaLib/cmocka
-86add13493e5c881d7e4ba77fb91c1f57752b3a4 UnitTestFrameworkPkg/Library/GoogleTestLib/googletest
```

It is a C language implementation of JSON, and the corresponding open source library is https://github.com/akheron/jansson.git.

The reason why it is in RedfishPkg is that the most common way for Redfish to transmit data is JSON. In order to implement Redfish under BIOS (currently not included in the official EDK open source code), JSON needs to be processed. RedfishPkg provides a JSON operation interface, located in RedfishPkg\Include\Library\JsonLib.h. Since there are many functions involved, I will not list them all here.

Using JSON in BIOS

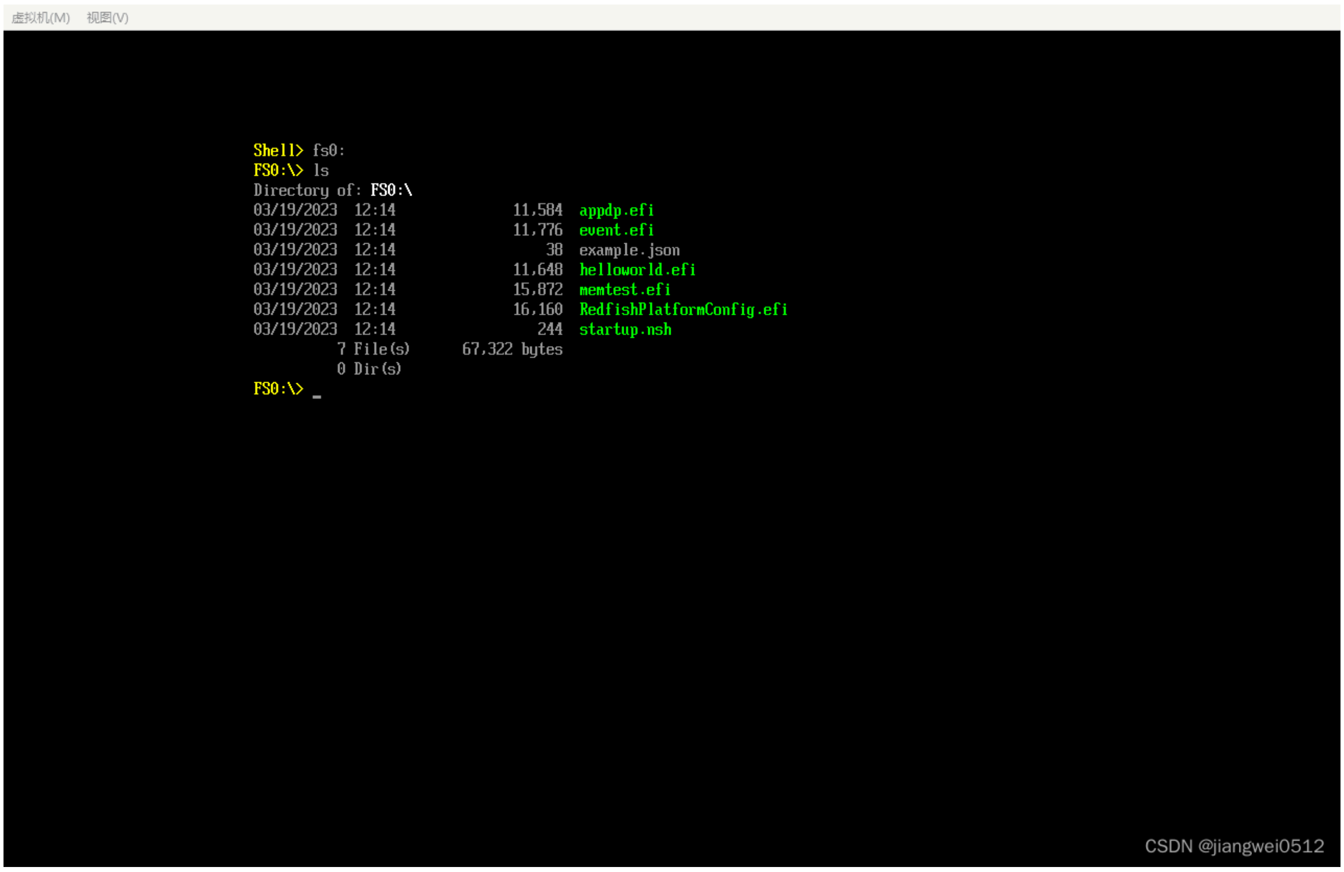
This section introduces an example of using JSON under BIOS. The final code can be found at https://gitee.com/jiangwei0512/edk2-beni.git.

1. First create a simple JSON file (example.json):

c AI generated projects 登录复制 run

```c
{
  "name": "BENI",
  "age": 18
}
```

We will operate it under Shell, so we need to put it in FileSystem, corresponding to FS0:



```
Shell> fs0:
FS0:\> ls
Directory of: FS0:\
03/19/2023  12:14              11,584  appdp.efi
03/19/2023  12:14              11,776  event.efi
03/19/2023  12:14                  38  example.json
03/19/2023  12:14              11,648  helloworld.efi
03/19/2023  12:14              15,872  memtest.efi
03/19/2023  12:14              16,160  RedfishPlatformConfig.efi
03/19/2023  12:14                 244  startup.nsh
          7 File(s)      67,322 bytes
          0 Dir(s)
FS0:\> _
```

2. Then include the JsonLib library under BIOS:

c AI generated projects 登录复制 run

```c
JsonLib|RedfishPkg/Library/JsonLib/JsonLib.inf
```

This library also depends on some other libraries, such as Ucs2Utf8Lib, RedfishCrtLib, etc., which also need to be included.

3. Then use a Shell command to test the library, corresponding to BeniPkg\DynamicCommand\TestDynamicCommand\TestDynamicCommand.inf.

One thing to note here is that RedfishCrtLib, which JsonLib depends on, uses another library, BaseSortLib, which is also needed by Shell applications (SortLib is actually used, but the two correspond to the same interface, but the implementation is different). The two use different code implementations, which leads to the need for Shell commands to be used by `StringNoCaseCompare()` ASSERT, resulting in usage exceptions. The solution here is to modify the implementation of RedfishCrtLib (corresponding to RedfishPkg\PrivateLibrary\RedfishCrtLib\RedfishCrtLib.inf):

bash AI generated projects 登录复制

```bash
[LibraryClasses]
  BaseLib
  SortLib    # 使用SortLib而不是BaseSortLib
  DebugLib
  MemoryAllocationLib
  UefiRuntimeServicesTableLib
```

4. Then comes the specific implementation of the code.

Here we first get the file example.json:

c AI generated projects 登录复制 run

```c
Status = ShellOpenFileByName (
          FileName,
          &FileHandle,
          EFI_FILE_MODE_READ,
          0
          );
if (EFI_ERROR (Status)) {
  DEBUG ((EFI_D_ERROR, "[%a][%d] Failed. - %r\n", __FUNCTION__, __LINE__, Status));
  goto DONE;
}

Status = gEfiShellProtocol->GetFileSize (FileHandle, &FileSize);
if (EFI_ERROR (Status)) {
  DEBUG ((EFI_D_ERROR, "[%a][%d] Failed. - %r\n", __FUNCTION__, __LINE__, Status));
  goto DONE;
}

Data = AllocateZeroPool (FileSize);
if (NULL == Data) {
  DEBUG ((EFI_D_ERROR, "[%a][%d] Out of memory\n", __FUNCTION__, __LINE__));
  goto DONE;
}

Status = ShellReadFile (FileHandle, &FileSize, Data);
if (EFI_ERROR (Status)) {
  DEBUG ((EFI_D_ERROR, "[%a][%d] Failed. - %r\n", __FUNCTION__, __LINE__, Status));
  return;
}
```

收起

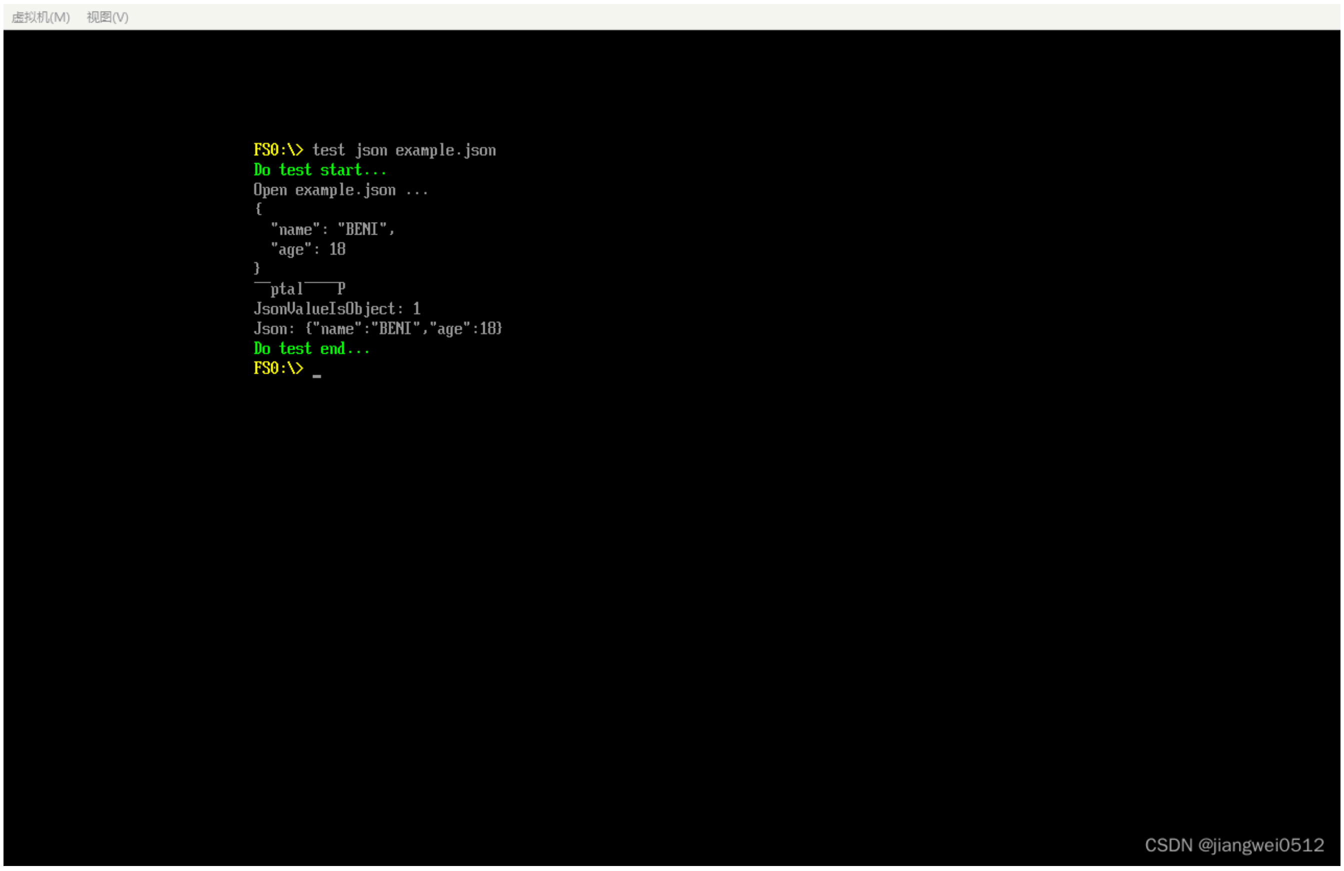
This will get the data of the JSON file. The next step is to convert the data into JSON, which requires the use of `JsonLoadBuffer()` functions in JsonLib:

c AI generated projects 登录复制 run

```c
Json = JsonLoadBuffer (Data, FileSize, 0x4, &JsonError);
if (NULL == Json) {
  DEBUG ((EFI_D_ERROR, "[%a][%d] Failed. - %r\n", __FUNCTION__, __LINE__, Status));
  goto DONE;
}

Print (L"JsonValueIsObject: %d\r\n", JsonValueIsObject (Json));
Print (L"Json: %a\r\n", JsonDumpString (Json, EDKII_JSON_COMPACT));
```

The test results are as follows:



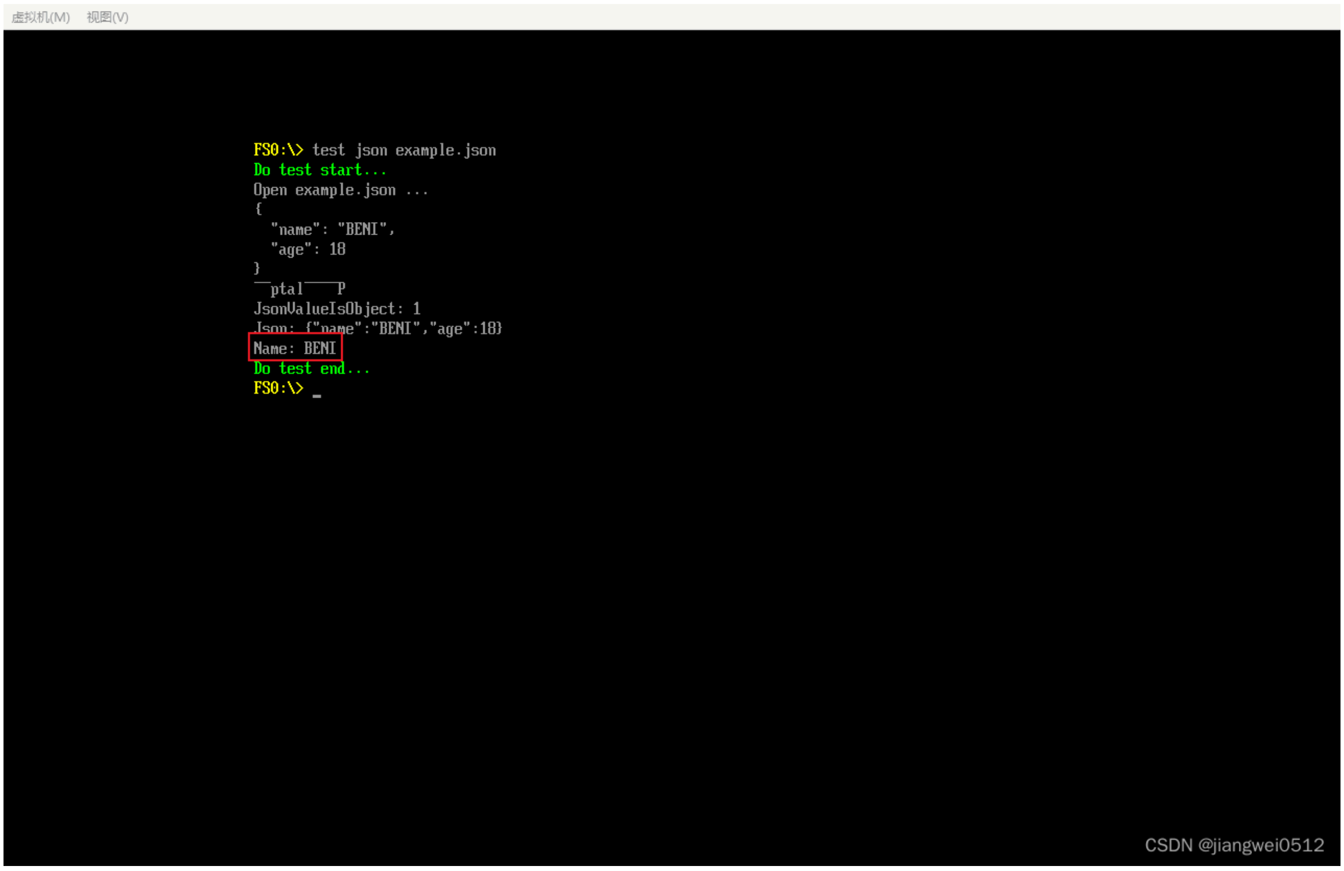
```
FS0:\> test json example.json
Do test start...
Open example.json ...
{
  "name": "BENI",
  "age": 18
}
¯¯ptal¯¯¯P
JsonValueIsObject: 1
Json: {"name":"BENI","age":18}
Do test end...
FS0:\> _
```

You can see that the JSON object under BIOS has been generated. If you want to get a specific item:

c AI generated projects 登录复制 run

```c
JsonData = JsonObjectGetValue (Json, "name");
if (JsonValueIsString (JsonData)) {
  Print (L"Name: %a\r\n", JsonValueGetAsciiString (JsonData));
}
```

This will print `name` the value corresponding to the key:



```
FS0:\> test json example.json
Do test start...
Open example.json ...
{
  "name": "BENI",
  "age": 18
}
¯¯ptal¯¯¯P
JsonValueIsObject: 1
Json: {"name":"BENI","age":18}
Name: BENI
Do test end...
FS0:\> _
```

In addition to reading key values, you can also modify the corresponding values through keys, such as here `name` :

c AI generated projects 登录复制 run

```c
// Create new JSON object for string.
NewData = JsonValueInitAsciiString ("JIANGWEI");
if (NULL == NewData) {
  DEBUG ((EFI_D_ERROR, "[%a][%d] Failed. - %r\n", __FUNCTION__, __LINE__, Status));
  goto DONE;
}
// Set JSON value.
Status = JsonObjectSetValue (Json, "name", NewData);
if (EFI_ERROR (Status)) {
  DEBUG ((EFI_D_ERROR, "[%a][%d] Failed. - %r\n", __FUNCTION__, __LINE__, Status));
  goto DONE;
}
```

收起 ^

Here the JSON object has been modified. If you want to save the modified value to a file, there seems to be no corresponding function in the current JsonLib, but you can still write the data to a file through some processing:

c AI generated projects 登录复制 run

```c
FileSize = AsciiStrLen (NewJson);
FileData = AllocateZeroPool (FileSize + 2);
CopyMem (FileData, JsonDumpString (Json, EDKII_JSON_ENSURE_ASCII), FileSize);
FileData[FileSize] = 0xD;
FileData[FileSize + 1] = 0xA;
FileSize = FileSize + 2;
BeniDumpHex (2, 0, FileSize, FileData);
Status = ShellWriteFile (FileHandle, &FileSize, FileData);
```

Here the string is converted into file content and then written to the file. The difference is that the string ends with '\0', while the file ends with a newline character (which doesn't seem right?).