DMTF-related protocols (I): Redfish and SMBIOS

原创 dropevil Modified on 2025-02-17 19:17:20 Read 1.3k Collection 17 Likes 28
Category Column: DMTF Article Tags: Embedded Hardware

1. Redfish

1. Introduction

Redfish is a management standard expressed using a data model in a hypermedia RESTful interface . The model is expressed in a standard, machine-readable schema, and the payload of the message is expressed in JSON. The protocol itself leverages OData v4. Since it is a hypermedia API, it is able to express a variety of implementations through a consistent interface. It has mechanisms for managing data center resources, handling events, long-running tasks, and discovery.

2.root url

Redfish is a hypermedia API. This means that you can access all resources through URLs returned from other resources. But there is a well-known URL that gives every implementation a common starting point. For the first version of the Redfish interface, this URI is "/redfish/v1". The
URL consists of a scheme (the HTTP:// part), a node (such as www.dmtf.org or an IP address such as 127.0.0.1), and a resource part. You can combine these parts in a URL in a browser. So if you are using an nginx server on your machine, you should be able to type " HTTP://127.0.0.1/redfish/v1/ " in your browser to access the Redfish root directory.
Perform a GET operation on the service root directory from the URL you constructed above.

3. Basic properties

Properties that begin with "@" are annotations for the entire object, while properties with "@" in the middle of the property name are used to annotate individual properties. There are two types of annotation properties allowed: OData and DMTF annotations. OData annotations contain "@odata.". Examples of these properties are:
"@odata.type", which is used to look up the schema that defines this resource.
"@odata.id", which contains the URL for this resource. This is an href, but since Redfish is based on OData, this property is called "@odata.id" instead of "href".
"Members@odata.count", which defines the number of entries in the "Members" array.
DMTF annotations contain "@Redfish.". Examples of these properties are:
"@Redfish.Settings", which is used to indicate settings for a resource
Another common annotation is "@odata.context". This is really designed for generic OData v4 clients. This is clearly defined in the OData v4 specification, but basically @odata.context is used for a few different purposes:

1. Provides the location for metadata describing the payload.
2. Provides a root URL for resolving relative references.
   The structure of @odata.context is a URL pointing to a metadata document with a description of the data fragment.
   Technically, a metadata document only needs to define or reference any types it uses directly, and different payloads can reference different metadata documents. However, because @odata.context provides a root URL for resolving relative references (such as @odata.id), we must return a "canonical" metadata document. In addition, because our "@odata.type" annotations are written as fragments rather than full URLs, these fragments must be defined or referenced in this metadata document. And, because we use unversioned namespace aliases to qualify operations, these aliases must also be defined by reference in the referenced metadata document.
   For example, in the resource /redfish/v1/Systems/1, you will see the property "@odata.context" with the value "/redfish/v1/metadata#ComputerSystem.ComputerSystem". This tells the generic OData v4 client to find the ComputerSystem definition in the metadata, which references the definition of this entity.
   You will also see an annotation called "@Redfish.Copyright". This property is not returned by the implementation. It is only used as a copyright notice in the static sample responses.

4.Session

Normally, only the root user can access without establishing a session. However, if you are using a public mock or local web server, this is not necessary as there is no security. If you are running the emulator in secure mode, or are working on an actual implementation, you will need to establish a session.
First, you will need to know a valid username and password for your implementation.
The URI used to establish a session is also allowed for an unencrypted POST so that a session can be established. This URI is /redfish/v1/Sessions in most cases and can be determined by looking at the Sessions property under Links and finding the value of the @odata.id property in the service root (/redfish/v1/). A session
is created via an HTTP POST to the URI indicated by /redfish/v1#/Links/Sessions/@odata.id, including the following POST body:

AI generated projects 登录复制

```
POST /redfish/v1/SessionService/Sessions HTTP/1.1
Host: <host-path>
Content-Type: application/json; charset=utf-8
Content-Length: <computed-length>
Accept: application/json
OData-Version: 4.0
{
 "UserName": "<username>",
 "Password": "<password>"
}
```

收起

The response includes an X-Auth-Token header with the session token and a Location header. The
returned JSON body contains a representation of the newly created session object:

AI generated projects 登录复制

```
Location: /redfish/v1/SessionService/Sessions/1
X-Auth-Token: <session-auth-token>
{
 "@odata.context": "/redfish/v1/$metadata#SessionService/Sessions/$entity",
 "@odata.id": "/redfish/v1/SessionService/Sessions/1",
 "@odata.type": "#Session.v1_0_0.Session",
 "Id": "1",
 "Name": "User Session",
 "Description": "User Session",
 "UserName": "<username>"
}
```

收起

The token string will be used in the X-Auth-Token header of the response, and the same header will be used in all subsequent requests.
When you need to delete the session, you can perform a DELETE operation on the URL returned by @odata.id in the response (such as /redfish/v1/Sessions/Administrator1 in the above example).

5. Etag

[The core role of ETag]
ETag (entity tag) is a mechanism used for resource version control in the HTTP protocol. In the Redfish standard, ETag is designed to:

1. Cache optimization - The browser/client can avoid retransmitting unchanged data by sending an If-Match request header with an ETag
2. Change Detection - Any resource modification through a PUT request or through other means (such as BIOS console configuration tools) will trigger an ETag value change
3. Concurrency control - detects write race conditions between Redfish clients and non-Redfish clients (or other Redfish clients)

Note: In a real production environment, the ETag mechanism can effectively prevent the "lost update" problem, ensuring that when two clients modify a resource at the same time, the later request will be rejected due to ETag mismatch.

Client operation process

1. **Resource reading phase**

    1. GET request to obtain the current status of the resource
    2. Record the ETag value (version identifier) in the response header
2. **Local modification stage**
    1. Modify resource attributes (such as BIOS configuration parameters)
    2. Generate a new ETag (usually a hash or version number)
3. **Submit update phase**

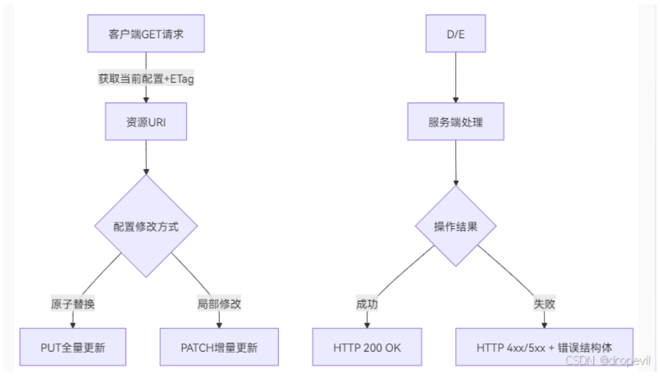
   http

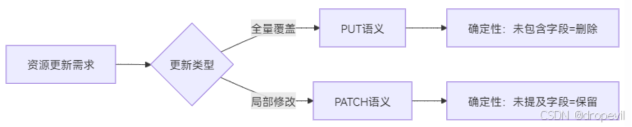
   AI generated projects 登录复制

```
PATCH /redfish/v1/Systems/1/Bios/Settings
If-Match: "a18c7e5d"  // 携带原始ETag
{
  "Attribute": "NewValue"
}
```

4. **Conflict Detection Mechanism**
    1. The server verifies the If-Match header and the current resource ETag
    2. Match successful → Apply changes and update ETag (HTTP 200)
    3. Matching failed → Return 412 Precondition Failed error
5. **Client recovery strategy**
    1. After receiving a 412 error, you need to re-execute the entire operation chain
    2. It is recommended to use an exponential backoff retry mechanism

5.PUT or PATCH





Dimensions	PUT	PATCH
Data Integrity	A complete resource representation must be submitted	Only the changed fields need to be submitted
Idempotence	Yes (multiple submissions will result in the same result)	Yes (based on RFC 5789)
Transmission efficiency	O(n) full transfer	O(Δ) difference transmission
Server processing	Replace the entire resource tree	Apply the JSON Merge-Patch algorithm
Typical scenarios	Configuration template import/restore default	Real-time tuning/hot update

SMBIOS

The SMBIOS specification addresses how motherboard and system vendors present management information for their products in a standard format through platform firmware. This information is intended to allow generic instrumentation to pass this data to management applications using CIM (WBEM data model) or direct access, and eliminate error-prone operations such as probing system hardware for presence detection.

support processor

- IA-32 (x86)
- x64 (x86-64, Intel64, AMD64, EM64T)
- Intel® Itanium® architecture
- 32-bit ARM (Aarch32)
- 64-bit ARM (Aarch64)
- RISC-V 32 (RV32)
- RISC-V 64 (RV64)
- RISC-V 128 (RV128)
- 32-bit LoongArch (LoongArch32)
- 64-bit LoongArch (LoongArch64)

UUID

1. Core attributes

    1. Identifier properties

- Uniqueness guarantee: dual uniqueness design in time and space
- Encoding length: 128 bits fixed length (16 bytes)
- Registration mechanism: No central registration agency required

2. Data structure specification (RFC4122 standard)

Offset	Field Name	length	describe
00h	time_low	DWORD (4 bytes)	Timestamp low segment
04h	time_mid	WORD (2 bytes)	Timestamp median segment
06h	time_hi_and_version	WORD (2 bytes)	High-order segment of timestamp (including version number)
08h	clock_seq_hi_and_reserved	BYTE (1 byte)	Clock sequence high bit (including variant identification)
09h	clock_seq_low	BYTE (1 byte)	Clock sequence low
0Ah	Node	6 bytes	Spatially unique node identifier

3. Byte order specification (SMBIOS implementation requirements)

1. Coding conflict points

- RFC4122 recommends: Network byte order (big endian)
- PC industry practice: Little endian (including ACPI/UEFI/SMBIOS)
  → Applicable fields: time_low/time_mid/time_hi_and_version

1. Transmission format requirements

- Must use wire format (network transmission format)
- Example conversion:
  UUID {00112233-4455-6677-8899-AABBCCDDEEFF}
  → Byte sequence: 33 22 11 00 55 44 77 66 88 99 AA BB CC DD EE FF
- Note: The UUID field content in the SMBIOS specification is considered opaque data, and the focus is only on the correct handling of byte order. When developing and implementing, special attention should be paid to the differences between the little-endian encoding conventions of the PC industry and the RFC standard.