Redfish White Paper

Redfish 白皮书

目录

什么是 Redfish API?

Redfish 是一种管理标准，它使用超媒体 RESTful 接口的数据模型表示法。此模型以标准的机器可读模式表示，其消息负载以 JSON 来表示。协议本身利用了 OData v4 版本。Redfish API 是超媒体 API，可通过统一的接口来表示各种实现。Redfish API 提供数据中心资源管理、事件处理、长时间任务以及发现等机制。

1.为什么需要新的接口?

各种影响导致需要新的标准管理接口。

首先，市场正在从传统的数据中心环境向可扩展的解决方案转变。可扩展解决方案以及超大规模采取分布式方式(而非集中式)，即由大量简单的服务器共同执行一组任务。在这样的环境中，可靠性是通过自有软件或开源软件来实现的。因此，使用模型不同于传统的企业环境。一个形象的比喻就是把服务器作为 “牲畜”，而不是“宠物”对待。这种客户环境需要标准化的接口，以便在异构的多供应商环境中实现一致性。

目前的扩展管理缺少功能性和同构接口。例如，IPMI 功能的使用仅限于“最小公分母”命令(如 Power On/Off/Reboot、temperature value、text console)。因此，需要带外功能的客户无法使用完整的功能，因为供应商扩展未实现跨平台通用。此类新客户开始开发越来越多的自己的工具，以实现紧密集成，因为具备开发公共可管理功能的能力，他们有时会通过带内软件来实现管理功能。平台管理规范随着 OEM 扩展增多逐渐变得零散，致使管理功能无法满足零散客户的扩展性需求。另外，现有管理解决方案已无法通过引用特定的安全和加密需求来满足客户的安全需求。

其他标准（如 SMASH）无法实现通用性要求。这是由其复杂性决定的。CLP 最终得以在大多数硬件中实现，但由于不支持统一的输出格式，解析结果数据要依赖于实现方式。WS 管理只能在有限的带外环境中实现。WS 管理是复杂

的分层协议，最适用于同构环境，而无法满足异构要求。另外，此接口的实现非常复杂，涉及对协议的了解、通用操作、模式、概要文件本身等一系列问题，历时多年开发、调整并添加了新功能的解决方案即体现了这一复杂性。客户需要花几个月的时间来熟悉接口，了解熟练使用接口所需的重要资源约定和技术。即便是完成最简单的任务，也需要大量的操作。因此，虽然此接口支持可扩展系统，但它本身的 I/O 模式是不可扩展的。

硬件环境正在迅速变化。将多个系统置于一个传统“刀片”中，或者将多个刀片聚合到单个系统中，这些类型的应用正变得越来越普遍。这些需求由传统企业环境管理软件来管理。但是，此类新系统无法通过位操作协议来充分表示。他们无法展示现代系统组件之间复杂的架构关系。此外，系统聚合点（如机壳机箱管理器）无法使用这些协议通过同样的方法来表示系统的复杂性。

最后，客户需要更现代的接口。他们希望在新兴的云计算环境中，以 API 的形式使用协议、架构和安全模型。这些 API 的优势在于，客户已自行研制了加速开发的工具。客户(无需厂商推动)需要的就是以 JSON 格式表示数据的 RESTful 协议。

2.为什么采用 REST、JSON 和 OData?

REST 正在迅速取代 SOAP 成为一种主流协议。整个云生态系统以及 Web API 社区都在采用 REST 协议。REST 的学习比 SOAP 要快得多。REST 是可以直接映射到 HTTP 协议的简单数据架构（而不是严格意义上的协议）。

- Web API 最常用的协议

REST 语义很容易映射到 HTTP/HTTPS。因此，REST 有著名的安全模型和网络配置设置，大多数系统管理员很容易理解这一点。这是高中和 JR 学院教授的内容。因此，教授 REST 用法的需求大大降低了。以开源方式实现的网络服务器比比皆是，大量编程语言提供对 REST 的支持。

JSON 迅速成为一种现代数据格式。它本质上是人可读的，比 XML 更简洁，得到很多的现代语言支持，是 Web 服务 API 最快速增长的数据格式。附加的嵌入式管理环境是 JSON 的一大优势。大多数 BMC 已经支持 Web 服务器，而通过浏览器来管理服务器已很普遍。这通常是通过 JS 驱动的接口完成的。选择 JSON 后，Web 浏览器可以使用直接来自其 Redfish 服务的数据，确保通过浏览器和编程接口的数据具有相同的语义和值。

- 2014 年最流行的编码语言
- 新 API 中使用的数据格式

但下面的 RESTful 实践和 JSON 格式化的结果是不够的。RESTful 接口的种类几乎和应用程序的种类一样多，各种 RESTful 接口有其特定的资源类型、报头和查询选项以及结果表示法。虽然 JSON 提供简单易读的表示方法，但常见属性（如 ID、类型、链接等）的语义是会根据命名约定来定义的，不同服务有不同的命名约定。

OData 定义了一组常见的 RESTful 约定，提供 API 之间的互操作性。

采用常见的 OData 约定来描述 JSON 有效负载中的模式、URL 约定、通用属性的命名及结构，这样不仅集合了适用于传统和可扩展环境的 RESTful API 的最佳实践，而且使 Redfish 服务可用由通用客户端库、应用程序、工具组成的不断扩大的生态系统。

3.为什么采用超媒体 API?

客户需要能够支持海量计算平台的通用 API，而非多种接口或编程方式。他们需要的是适用于独立服务器、超级扩展和可分区甚至虚拟系统的通用 API。单个固定 URI API 将无法完全显示各种金属片、内部服务器以及相关管理器之间的各类包含关系。这意味着，一对多、多对多的关系是必须的。

此外，API 必须兼具易用性和灵活性，以同时适应简单系统和超级扩展系统。使用 URL 来表示相似资源的关联和集合，这在超媒体 API 中已得到证明。

我们可以通过一个使用中的实现方式来简单地了解 Redfish API，下面的章节将介绍一种实现方式，并通过此实现解释各种架构和概念。

您需要两件东西：浏览器和一种通过浏览器接入的实现方式。您也可以使用文件浏览器或管理器来浏览文件。

Redfish API 是基于 REST 和 JSON 的，所以您只需浏览器即可查看 Redfish 实现。建议您的浏览器支持 JSON 格式，以便读取数据。大多数浏览器可通过安装插件来支持 JSON 格式。

为了获得真正的“Redfish”体验，您可以为浏览器下载 RESTful 插件，如适用于 Chrome 的高级 REST 客户端。这样，您就可以设置头部，并看到 HTTP 代码等浏览器通常会隐藏的项目。您还可以方便地访问“真正的”Redfish 实现（如模拟器或供应商实现）。

4.访问实现

您可以直接查看标记，或通过 Web 服务器查看。为此，您只需访问一个实现。方法如下：

- 访问公共模型。在 Redfish.dmtf.com 网站上有一个运行 JSON 的 Web 服务器。您可以将浏览器指向该地址并访问数据。在这里，您了解了解模型和模式。
- 将数据复制到您的 Web 服务器。您可以访问 Redfish 模型，并将 Mockup 路径下的所有文件备份至您本地硬盘的/redfish/v1 路径下，服务器使用该路径提供 HTML 页面。例如，您可以下载 nginx，设置返回 JSON 格式，然后将模型文件加载到 HTML 路径下。

5.根

Redfish 是一种超媒体 API。这意味着，您可以通过其它资源返回的 URL 获取所有资源。由于存在一个众所周知的 URL，所有实现都有一个共同的起点。此 URI 即针对 Redfish 接口 V1 版本的“/redfish/v1”。

每个 URL 包含模式(HTTP://部分)、节点(如 www.dmtf.org 或 IP 地址，如 127.0.0.1)和资源部分。这三项共同构成了您的浏览器中的 URL。因此，如果您在自己的机器上使用 nginx 服务器，应在浏览器中输入“HTTP://127.0.0.1/redfish/v1/”来访问 Redfish 的根。

在以上 URL 中，在根服务中执行 GET 操作。

基本概念

每个 URL 代表一个资源。一个资源可以是一项服务、一个集合、一个实体或其他结构。但在 RESTful 术语中，URI 指向资源和与资源交互的客户端。所以，当您看到“资源”这个术语时，可以把它想象为访问某个 URI 时返回的内容。

资源格式由 Schema 来定义。Redfish Schema 中规定了每个资源的格式，客户端可使用此格式来确定资源的语义(虽然我们尽可能让事情直观一些)。Redfish Schema 以两种格式定义：OData-Schema 格式和 JSON Schema 格式。以 OData Schema 格式(CSDL)定义，是为了便于通用 OData 工具和应用程序解析。以 JSON Schema 格式定义，是为了应用于其他环境，如 Python 脚本、JavaScript 代码和可视化。

资源的结构属性可用作 JavaScript 变量。这样可加快采用，并允许 JavaScript Web 页面和已启用的应用直接使用数据。URI 在重新引导后不会改变，但客户端应从/redfish/v1/启动，并在此路径下发现 URI。

将URI固定化是一个常见的错误。Redfish是超媒体API，所以各实现（即使是同一厂商的实现）的URI是不同的。当前状态对象可以独立于理想状态对象。

6.操作

操作包括GET、PUT、PATCH、POST、DELETE 和 HEAD。在未安装高级REST插件时，您的浏览器会执行GET操作。只有模拟器和真正的实现才支持其他操作。

GET执行数据检索。POST用于创建资源或使用动作(详见下文)。DELETE用于删除资源，但目前只有少数资源可被删除。PATCH用于改变一个或多个资源的属性，PUT用于完全替换资源(只有少数资源可被完全替换，详见下文)。HEAD与GET类似，只是不返回主体数据，访问Redfish实现的程序可使用HEAD来获取URI结构。

7.版本控制

Redfish有两种版本，协议版本和资源模式版本。协议版本包含在URI中——这就是为什么您应该从/redfish/v1/开始。这表示您正在访问协议版本。V1版本是目前唯一的版本，但未来会有其它版本。以这种方式开头的URI表示实现方法符合Redfish规范V1版本。注意，因为Redfish基于OData v4，实现还要求OData协议头(OData-Version)的值为4。

每个资源都有资源类型定义。资源类型在带有版本的命名空间中定义。每个资源实例的类型由OData类型注释“@odata.type”来表示。类型注释的值是资源类型URI，包括带有版本的命名空间。所以，当您看到“@odata.type ”：“# ServiceRoot.1.0.0.ServiceRoot”时，您处理的是ServiceRoot模式1.0.0版本中定义的ServiceRoot类型的资源。相应的模式文件位于Redfish模式库的/schema /v1/ServiceRoot路径下。因此，此类型的完整URI为“/schema /v1/ServiceRoot#ServiceRoot.1.0.0.ServiceRoot”。模式文件可能包含资源类型定

义中的其他类型(如结构化类型和枚举)，这些类型有相同的资源路径，通常有相同的命名空间，但类型定义不同。

8.引用

链接部分包含对其他资源的引用。

JSON没有用于引用其它资源的本地引用类型。Redfish需要全连接资源树，无需客户端咨询模式(对于简单操作)。因此，Redfish根据OData约定来表示对另一内部资源或外部资源的引用。

表示根据OData约定引用其它资源的属性，其属性名后缀为“@odata.id”。表示其他类型引用的URL的属性（如外部帮助主题），在元数据中为字符串属性，注释为代表URL。

URI可以是绝对的或相对的。绝对URI没有IP地址，但从/redfish/v1/开始。如果您安装了适用于Chrome的高级REST客户端等插件，您可以单击此插件，输入下一个GET操作的URI。

9.主要的对象

“主要”对象指系统、管理器和机壳。三者都是集合(见下一节)。稍后我们会详细介绍这些资源，下面先简单了解一下。

系统指典型服务器。系统可以是CPU访问的任何数据面对象，这些对象组成系统集合。系统中包含CPU、内存和其他设备。

管理器指BMC、机箱管理器或其他基础设施管理组件。管理器处理各种管理服务，但也有自己的设备(如网卡)。

机壳指基础设施的物理容器部分。机架、机箱、刀片等等都包含在机壳内。因此，有一种说法叫“机壳中的机壳”，即包含传感器、风扇等的机壳。

一个系统可以包含同一机壳内一个或多个管理器(因为有些管理器是冗余的)。一个管理器可以管理同一机壳内的多个系统。一个机壳可以容纳多个系统和/或管理器。

以上只是概述，但对于ServiceRoot对象，您可以立即看到以上对象。此外，您还会注意到会话等服务(详见后文)。

10.集合

一组类似的资源称为集合。模型中集合的例子包括系统、管理器、机壳、日志条目、会话、事件订阅等等。

选择系统、管理器或机壳，并进入所选项目。您会看到它是一个集合。

集合响应包含名为“Members@odata.count”的计数属性，以及一个“成员”对象，其中包含成员列表或到成员的链接。成员链接表示为带有“@odata.id”属性的JSON对象，属性中包含成员的URL。

集合可以分页；带有“@odata.nextLink”属性的集合是不完整集合，客户端可以使用下一个链接属性的URL值，在服务中检索集合的下一部分。

11.公共属性

当您浏览模型时，您会反复看到一些相同的属性。在每个资源中您都会发现“Name”和“Id”属性。“Name”和“Id”是必要属性。您还会看到嵌入式对象“Status”，它在所有使用中有相同的定义。实际上，这些属性属于Schema的公共部分，可以被其他Schema引用。公共属性通过模式中的OData“引用”元素被各资源的模式引用，确保各处使用的定义一致。这类属性包括：

- “Actions”，告知客户可以调用哪些操作。(详见动作章节)
- “Oem”，具体厂商针对标准资源定义的扩展。详见Oem章节。

12.常见注释

一些属性同时也是注释。这些属性以“@”开始，或包含“@”。注释属性有两种：OData注释和DMTF注释。OData注释包含“@odata.”或“@odata.”，例如：

- “@odata.type”，用于查找定义此资源的模式。
- “@odata.id”，包含资源的URL。此属性为HREF，但由于Redfish基于OData，该属性称为“@odata.id”而非“href”。
- “Members@odata.count”，定义了集合中资源的数量。

DMTF注释包含“@DMTF.”，例如：

- “@DMTF.Settings”，表示资源设置(详见后文)。

另一常见注释是“@odata.context”。这是通用OData v4客户端的专用注释。该注释由ODatav4规范定义，但往往用于其它用途：

1. 提供描述有效负载的元数据位置。
2. 提供用于解析相对引用的根URL。

@odata.context的结构为带有数据(通常根路径为顶层单一实例或集合)描述片段的元数据文档URL。

理论上，元数据文档只需定义或引用它直接使用的类型，不同的有效负载可以引用不同的元数据文档。然而，由于@odata.context提供根URL来解析相对引用(如@odata.id)，我们必须返回“规范的”元数据文档。另外，由于“@odata.type”注释为片段而非完整URL，这些片段必须在元数据文档中定义或引用。因为我们会解释包含不带版本的命名空间别名的动作，这些别名也必须在被引用的元数据文档中通过引用来定义。

例如，在资源/redfish/v1/Systems/1中，您会看到“@odata.context”属性的值为“/redfish /v1/$metadata#Systems/Links/Members/$entity”。这是告诉通用OData v4客户端，在$metadata中查找Systems的定义，查看Links属性定义，其中包含Members属性定义，该定义包含对此实体定义的引用。

13. 动作

使用REST并不能轻松完成所有操作。所以我们还会发出“动作”。对系统进行的“下压按钮”操作(根据设置不同，可能重置或关闭系统)在系统中不易表示，因为服务不知道按钮处于何种状态，因此不会将此操作视为属性。另一用法是针对长时间操作，长时间操作（如固件更新或正常关机）更易表示为方便客户端的原子操作。

因此，我们创造了“动作”，以替代可以PUT/PATCH的隐藏属性或复杂的状态机。动作是通过对资源执行POST操作来完成的(详见规范)。您可以通过查找“Actions”属性了解某资源支持哪些动作。

14. 模式

模式包扩客户端可使用的大量信息。建议客户端，特别是编程客户端，通过检查模式来确定属性的语义。模式包含数据类型、枚举列表、属性信息描述、属性行为的规范信息以及其他信息。

更多概念

以上是最简单的Redfish API。您会发现数据简单易读，但可能想进一步了解安全、模式、报头等方面的内容。

15.会话

通常，在没有建立会话时，只有根路径是可以访问的。但如果您使用的是公共模型或本地Web服务器，则无需建立会话，因为没有安全机制。如果您的模拟器运行在安全模式下，或者您在使用真正的实现，则需要建立会话。

首先，您需要知道实例的有效用户名和密码。

建立会话使用的URI也可用于不安全POST。在大多数情况下，此URI为/redfish/v1/Sessions，您通过查看Links下的Sessions属性，并在服务根路径(/redfish/v1/)下找到@odata.id属性值来确定。

HTTP POST创建会话，URI为/Redfish/v1#/Links/Sessions/@odata.id，POST正文如下：

```
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>"

}
```

返回信息包含带有会话令牌和位置头的X-Auth-Token报头。

返回的JSON实体包括新创建的会话对象的表示：

```
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.1.0.0.Session",

"Id": "1",

"Name": "User Session",

"Description": "User Session",

"UserName": "<username>"

}
```

响应X-Auth-Token报头中的令牌字符串可用于所有后续服务请求的相同报头中。当删除会话时，您可在对响应的@odata.id中返回的URL（在以上示例中为/redfish/v1/Sessions/Administrator1）执行DELETE操作。

冗余

回到其中一个机壳，根据“热量(Thermal)”链接查看风扇，您将看到Redfish 如何显示冗余。

你会发现一个称作“冗余”的数组。它显示了同一个集合中的两个风扇在相关项目属性中使用相同的值。冗余在Redfish中有一个共同的模式定义，除了成员，还有其他属性用以展示其余关于冗余的重要属性。既然@odata值指向冗余集合成员，这就是客户端如何找出属于冗余组的项目。

然而，"@odata.id"属性的值不需要是一个整体的资源。这个属性的值有两种格式:JSON指针或OData引用。

- 如果是JSON指针，会有一个#表明资源停止的地方和属性模式开始的地方。模式也引用属性。JSON指针值的一个示例可能是“/redfish/v1/Chassis/1/Thermal#/Fans/0”。
- 如果是OData引用，不会出现#。模式有一个属性的定义。JSON指针值的一个示例可能是“/redfish/v1/Chassis/1/Thermal/Fans/0”。

17.相关项目

如果你仍然处于“热量”的资源中，你可以看到温度数组元素有个带有“@odata.id”的相关项目属性。这为子资源提供参考，这个温度传感器在测量——也许是一个处理器。就像冗余链接，可能的值是一个子资源。因此，客户端可以决定这温度传感器测量的是什么。

拥有一个共同模式定义的相关项目，在使用中是依赖情境的。但它总是用来显示两个不同的资源或子资源的关系。

和冗余一样， "@odata.id"属性不一定是一个整体的资源。

18.服务

我们回到根看看一些常见的服务。任务、会话、事件服务(EventService)和账号服务(AccountService)都是公共服务。你可以在规范中阅读更多详细信息，但他们所提供的服务的名字应该是人人皆知的。任务包含一系列可能已经

开始的工作，通常是行动的结果。会话在上面已经讨论。账号服务用于如何创建用户。下文将对事件服务进行更多讨论。

19.注册表

消息注册允许管理处理器保留一个简短的用于事件和错误的标识符，然后客户可以查阅注册表来确定实际的消息。基于消息，有机制可实现参数替代。

通过将消息注册作为单独的实体，很容易支持国际化，因为管理处理器不需要包含翻译。相反，可以翻译注册表本身，客户端可以显示任何需要的翻译。

20.报文头

Redfish服务支持常见HTTP头的子集，其完整列表包含在规范中，但最相关的两个报文头是Accept头(必须设置为application/json)和前面所讨论的X-Auth-Token。但你会从实际的实现中得到更多信息，而这些信息你不会从从静态模型中看到。

21.ETag

Etags 由浏览器使用来优化IO(缓存)。他们执行一个If-Match语句，如果匹配，就不用拖拉数据。我们使用这些数据来确定某个对象是否已经发生了改变。因此，如果PUT完成或者工具(如BIOS的配置控制台)发生变化，每个ETag将发生改变。因此，人们总是会注意到Put中Redfish和非Redfish客户端（以及其他Redish客户端）之间的任何竞态条件。

问题是，这是个模型并不是真正的web服务，所以你看不到ETag工作。

21.1 处理客户端竞态条件

竞争的客户尝试设置数据相互覆盖是有可能的。这个条件是使用Etag和If- Match。

客户端读取/修改/写周期包括:

- 获取资源，包括当前的Etag
- 修改资源属性
- 生成一个新的Etag
- 修补修改的资源，包括包含最后一个已知的Etag的If-Match头
- 如果提供的Etag不再匹配对象(读取之后有些已经改变)，服务会返回“304 Not Modified”。客户端应该通过重复这些步骤进行恢复。
- 客户端在更新设置数据时*应该*包含一个Etag，以便提供者更有效地检测变化并防止多客户端竞态条件。

22.更新资源

在简单的GET、PUT/PATCH方法的模型中，客户端可能会GET当前配置，并将期望的配置PUT或PATCH到同一个资源URI中。PUT或PATCH操作的成功是由HTTP状态码和响应实体决定的。

更新资源用于与内在服务商的交互，例如Redfish服务提供商的数据。对于操作和配置的更改，固件可以进行适当的处理和响应。

当执行PUT或PATCH操作后，HTTP响应包含一个HTTP状态码，在失败的情况下，扩展的错误信息结构作为响应体。

22.1 PUT vs PATCH

为什么PUT和PATCH操作二者都采用？在设计本规范时，当要执行资源完全更换而非部分更换时，我们遇到语义问题。一些资源允许更换某些属性或完全更换资源。我们需要一种方法来确定何时清理如SettingData等资源中的现有属性。事情开始变得非常复杂，特殊(即非自然)规则是为了尝试使用单个操作。

PATCH解决了这个问题。PUT总是执行完全更换。PATCH总是执行部分更换。实现了针对服务的确定行为，而对客户端或服务器没有产生复杂的逻辑。

PATCH在行业中获得了广泛应用。已经得到了 Open Stack 和许多其他 API 的支持，并在许多现成的web服务器中可获得。

22.2 当前的配置 vs 设置

在Redfish中，基本上有两种类型的对象-当前的配置和设置。大多数对象代表任何给定资源的当前状态。偶尔，在某个资源中，你会看到一个称为“@DMTF.Settings”的属性。这个注释提供一个链接告诉你在哪里执行用作配置的PUT和PATCH操作。它代表了资源的未来状态。

一些资源可以立即处理变化，其他的可能需要重新开始/重新启动系统或服务。“@DMTF.Settings”用来让客户知道这是什么类型的资源，在哪里进行更改。

如果你看到“@DMTF.Settings”属性，该属性有一个到某个资源的链接，以便进行更改，该更改会在下次机会如重置或重启后被采纳。如果您没有看到设置链接，对象的所有PATCH操作应该立刻执行(在没有衍生任务的情况下)。

我们还提供有关设置最近一次应用到资源是时间的信息-如果设置能够成功应用，时间，ETag和任何消息本应都会返回的。

可能需要设置操作的资源的例子是像网卡、存储以及BIOS设备。

22.3 确定可以被更新的属性

模式和元数据都定义了哪种属性可以被更新。标有“OData.Permissions/Read”或有只读权限的属性，表明这些属性不能被更新。

相反，“OData.Permissions/ReadWrite”或“read-only= false”表示该属性可能是可写的。LongDescription(规范性文本)可能表明哪些属性是可写的并在什么条件下可写。请注意，默认情况下，没有“权限”注释的元数据或没有“readonly = true”的模式可以认为是可写的。

注意，即使它们标记为可写，也并不一定表明属性可写，因为实现可以允许属性是只读的。

Redfish PUT/PATCH的语义是，试图更新非可写属性并不视为一个错误。Redfish服务的一个实现将返回带有扩展错误信息的200，此信息表明该属性不能被更新。

23.扩展的错误响应

HTTP错误语义本身没有足够的信息为用户或应用程序理解原因并采取纠正措施，导致错误响应构造的概念。它还支持注册，允许为客户提供更有意义的错误语义。

例如，假设一个客户想要一次性更改几个属性。但其中一个是无效的。如果实现返回“400”，其不会帮助客户知道哪个属性发生错误以及原因。或者如果返回“200”但是其中一个属性不存在，这不是一个有意义的响应。

如果响应伴随着一个带有扩展错误响应的JSON体，结构不仅包含代码返回的原因，还包含属性名称，甚至更多的信息。因为扩展错误有一组消息，如果遇到超过一个问题，可以返回不止一条消息。

24.事件

我们需要某种类型的事件机制来满足与SNMP，IPMI和其他协议竞争。BMC中已经有机制了。所选择的方法是一个基于订阅的PUSH机制。PUSH事件为BMC首选，因为这意味着事件一旦发送出去，事件不会在内存中持久。

客户首先需要确定希望事件发送到的URI。通过向EventSubscription集合提交POST请求，可建立一个会话。

Redfish支持两种不同类别的事件：生命周期和警报事件。你可以查看EventService以查找实现支持的事件的类型。

有关事件的详细信息在规范中进行描述。

25.创建用户帐户和其他资源

类似于事件和会话，资源通过POST操作创建。该过程通常经历一个POST操作到集合。事件订阅，会话和帐户服务都有集合。类似于超媒体API中的其他资源，这些集合的URI也会在该超媒体API中被发现。

模式定义了一个注释“RequiredOnCreate”，客户端可以用其来确定哪些属性必须在POST中提供。

26.消息对齐

Redfish的架构师尝试使所有消息格式对齐。事件消息，扩展的错误响应消息，日志条目和消息注册都是对齐的。而供应商特定的和其他格式可以保存在一个实现中，消息格式对齐的路径转发不仅可以实现消息国际化，而且可以优化存储和统一客户端和服务任务，这些任务可以访问和呈现这些数据。

27.Oem

在模型和可能的实现中，您将看到一个称作“Oem”的结构。这种结构可以用在三个地方：作为资源的基础属性、在“链接”段或者在“行动”的对象。Redfish定义了一个机制，以便供应商可以通过在“Oem”对象中添加自己的对象，以包括自己的扩展。它必须有一个“@odata.type”属性，这样客户可以找到模式文件。另外，标准没有给出相关的需求，但它是这样一种机制，多供应

商环境可以潜在地放置扩展，以同一构造作为标准资源。这防止需要更多的IO并允许可扩展性。随着时间的推移，希望供应商提供功能以成为标准的一部分。

28.幂等(Idempotency)

任何RESTful服务的RESTful操作都是GET、PUT/PATCH、POST和DELETE。

幂等是Redfish简化的一个重要概念。在第一次调用幂等操作后可以多次调用，并且没有附加影响。例如：DVD遥控器上的停止按钮是幂等的，按一次，电影就停止，按多次，这部电影仍然处于停止状态。相反，暂停按钮，不是幂等的。如果你按两次，又开始播放。非幂等操作取决于先前的状态。

幂等操作是HTTP PUT和PATCH。如果客户端第二次PUT相同的数据，资源不会改变。PATCH替换属性，所以第二次PATCH操作不会再次改变资源，因为它已经在第一次PATCH操作时处于期待的状态。

PUT一资源是改变配置的最简单表达。它意味着“用Y替代资源X”。其他交互模式比这更复杂。PATCH操作稍微更复杂的，因为其需要差异化更新资源。通过幂等PATCH操作可以完全配置的服务是比较理想的。

Redfish使用基本的REST操作定义了三个基本的交互模式.

28.1 幂等修改：GET/PUT/PATCH

PATCH资源是用所提供的属性新值替代资源的内容。该操作用于这样的场景：服务可以简单地使用配置并产生期望结果状态。很有可能，一个客户端可能会GET一个资源，修改属性值，然后通过执行PATCH来提交修改。

PUT资源是用资源的新值代替资源的内容。该操作用于这样的场景：客户端可简单地将资源设为期望的状态。

28.2 创建、使用、删除：POST/GET/DELETE(非幂等性)

对一个资源执行POST可以创建子资源的一个新实例。这通常用于那些能够创建和删除项目的数据模型。例子包括创建新的用户帐户、删除用户帐户、

新日志条目，添加或删除许可证，等等。这些项目可能也支持删除。通过将内容POST到一个URI完成创建。新创建的资源是一个URI的子资源，可以使用DELETE删除新的URI。

28.3 做动作：用 “Action”属性(非幂等性) POST

对一个资源的Action属性返回的动作做POST可以产生执行自定义动作。这种模式不是幂等，因为再POST相同的内容将导致执行额外的工作。例如，一个自定义的动作可能会发送一个测试事件或清除日志。

结论

Redfish API 代表着一种新的 IT 编程风格，能够以一致的方式管理服务器系统，从超大规模服务器到刀片服务器，再到独立式服务器。我们认为它代表了自然进化，可适合客户需求以及客户所使用的工具。

常见用例

这里有一些用例将帮助您了解架构，开始您的客户端代码。

29. 查找系统的温度传感器

应用程序代码应该总是从根开始：/redfish/v1/

1. 在根部，对象是一个被称为“系统”的属性。
   a) 找到“@odata.id”的值。这是系统集合的 URI。
   b) 在模型中，这个 URI 是/Redfish/v1/Systems。
   c) 在此 URI 执行 GET.
2. 在“成员”数组中查看“链接”对象。
   a) 找到有问题系统的“@odata.id” (这可能需要对每个系统执行 GET，查看系统的属性从而确定正确的系统)。

b） 在模型中，这个 URI 是/Redfish/v1/Systems/1。

c） 对此 URI 执行 GET。

3. 在“链接”对象中查看称作“Chassis”的对象。

a） 找到“@odata.id”值。这是系统所在的机壳。

b） 在模型中，这个 URI 是/redfish/v1/Chassis/1。

c） 对此 URI 执行 GET

4. 在“链接”部分查看称作“热量”的对象。

a） 找到"@odata.id"的值。这是其中的温度传感器。

b） 在模型中，这个 URI 是/redfish/v1/chassis/1/Thermal。

c） 对那个资源上执行 GET。

5. 查看“温度”数组——这是系统的温度传感器。

a） 查看“相关条目”找出具体每个温度传感器监控哪些组件。

30.壳中壳

应用程序代码应该从根开始：/redfish/v1/

1） 在根部，对象是一个被称为“机壳”的属性。

a） 找到“@odata.id”的值。这是机壳集合的 URI。

b） 在模型中，这个 URI 是/redfish/v1/Chassis。

c） 对那个 URI 执行 GET。

2） 在“成员”数组中查看“链接”对象

a) 找到有问题的机壳底架的“@odata.id”（这可能需要对每个系统执行GET，查看机壳的属性从而确定正确的系统）。

b) 在模型中，这个URI是/redfish/v1/Chassis/Enc1。

c) 对那个URI执行GET。

3) 在“链接”对象中，查看称作“容器”的对象。

a) 找到“@odata.id”值。这是机壳所在的机壳。

4) 在“链接”部分查看称作对“被包含”的对象。

a) 找到"@odata.id"的值。这是壳中壳。