文件编号：DSP2044

日期：2018 年 9 月 4 日

版本：1.0.4

Redfish 白皮书

文件分级：参考文件

文件状态：已发布

文件语言：英语（美国）

目录

什么是 Redfish 应用程序接口？

Redfish 是一个管理标准，在超媒体 RESTful 接口内部使用数据模型表述。模型通过标准的、机器可读的架构表示，以 JSON 作为消息的有效负载。协议本身利用 OData v4。由于这是一种超媒体应用程序接口，其能通过一致接口表述各类实施情况。Redfish 拥有机制，管理数据中心资源、处理事件、长期任务和发现。

1. 为什么是新接口？

各种各样的影响导致需要一个新的标准管理接口。

首先，市场从传统数据中心环境向扩展解决架构转移。扩展解决架构和超范围已经采用了大量的简单服务器应用，这些服务器聚集在一起以分布式的方式执行一组任务（与集中式布置相反）。这些环境中的可靠性通过专有软件或开源软件实现。为此，应用模型有别于传统的企业环境。一种类比情况是服务器被视为“牛”,而非“宠物”。这组客户需要一个在异构、多供应商环境内保持一致的标准接口。

扩展管理中缺乏功能性和同质界面。例如，智能平台管理接口（IPMI）特色应用局限于一组“最小公分母”的命令（例如电源开/关/重新启动、温度值、文本控制台）。结果是，希望使用带功能的这些客户却被迫使用一组缩减后的功能，因为供应商扩展并不是在所有平台上都通用的。这组新客户越来越多地开发自己的工具进行紧密集成，有时依赖带内软件进行管理，因为它们能够在该环境中开发一组通用的可管理性。随着原始设备制造商扩展的激增，平台管理规范的碎片化越来越严重，导致功能不能满足向外扩展的客户需求，因为规范已经碎片化了。通过引用特定的安全和加密需求，现有的管理解决架构将不再满足客户的安全需求。

如 SMASH 等其它标准，并没有达到人们所希望的普及程度。原因在于它的复杂性。CLP 最终在大多数硬件中实现，但没有使用一致的输出格式，因此解析结果数据是独立实现的。WS 管理仅在有限数量的带外环境中实现。它是一个复杂的、分层协议，在同构环境中表现最好，并因此从未满足过异构需求。此外，对协议、类属操作、架构和概要本身组合的复杂理解最终形成了一个解决架构，即花费多年来开发、更改和添加新功能，而客户则需要用几个月的时间来理解重要的资源承诺和专业知识，以便熟练掌握应用，同时还需要大量的操作来执行最简单的任务。因此，虽然它能代表可扩展的系统，但接口本身却带有一个不可扩展的 IO 模式。

硬件领域正在快速变化。将多个系统放在传统的“刀片服务器”上或将多个刀片服务器合成为一个单一系统的系统正变得越来越普遍。这些需要由同一个正在管理传统企业环境的软件来管理。然而，这些系统无法用位操作协议充分表示。它们无法代表存在于现代系统组件间的复杂结构关系。除此之外，系统集成点，如机架或外壳管理器，无法使用这些协议执行用

相同方法表示复杂性的功能。

最后，客户要求一个现代接口。他们希望应用程序接口使用在新兴云界面中普遍存在的协议、结构和安全模式。这些接口的优势在于客户已经对这些工具进行了投资用于加快开发。具体来说，客户（无任何供应商提示）要求用 JSON 格式表示数据的 RESTful 协议。

2. 为什么是 REST、JSON 和 OData？

REST 正在迅速取代 SOAP 成为当今特色协议。整个云生态系统都在采用，网站应用程序接口社区也在使用。它比 SOAP 更容易学。它具有简单性，能直接以数据模式（因为严格来说它不是一种协议）映射到 HTTP 操作上。

- 最受欢迎的网站应用程序接口使用协议

REST 语义很容易映射到 HTTP/HTTPS。因此，它很容易被大多数的系统管理员所理解。它具有众所周知的安全模式，所支持的网站配置设置便于理解。因为高中和大专院校已经开设了 RES 课程，对其使用方式的普及需求正在大大降低。网络服务器的开源实施比比皆是，编程语言辅助无处不在。

JSON 正在快速成为现代数据格式。它本质上是人类可读的，比 XML 更简洁明了，有大量的现代语言支持，是网络服务应用程序接口中增长最快的数据格式。它对嵌入式可管理环境还具有一个额外优势。大多数 BMCs 已经支持一个网络服务器，并能通过浏览器管理服务器，这已成为常例。通常通过一个 Java 脚本驱动接口完成。通过选择 JSON，网络浏览器可以直接使用网络浏览器内来自 Redfish 服务的数据，从而确保通过浏览器和程序接口的数据在语义和价值上是一致的。

- 2014 年最流行的编码语言
- 新应用程序接口使用的数据格式

但是，仅仅遵循 RESTful 操作以及将作为独立 JSON 的结果格式化是不够的。RESTful 接口与应用几乎一样多，它们暴露的资源、可用的标题和查询选项以及结果表达方式各不相同。类似的是，虽然 JSON 提供了一种易于阅读的表述，但通过命名规格强加的常用属性语义，诸如身份号、类型、连接等，在各个服务间都是不相同的。

OData 定义了一组通用的 RESTful 惯例，在应用程序接口间提供互通性。

采用常见的 OData 惯例描述 JSON 载内的架构、url 规格，命名和通用属性结构，不仅为 RESTful 应用程序接口提供了最佳操作，让其在传统和可扩展环境中使用，也让 Redfish 服务进一步被通用客户端库、应用程序和工具等正在成长的生态系统所使用。

3. 为什么是超媒体应用程序接口？

需要一个单一的应用程序接口与无数的计算平台相匹配。行业不会支持多个接口或编程风格。客户需要一个单一的接口，它能从一个独立的服务器市场向超范围的、可分割的甚至虚拟的系统市场跨越。一个固定的同一资源标识符的应用程序接口不足以证明各种形式的金属薄片间的各控制关系、它们内在的服务器以及与之关联的管理器。这说明一对多或多对多的关联基数是需要的。

除此之外，应用程序接口对简易系统必须要简单，对超大规模计算要灵活。用来代表相似资源关联和集成的统一资源标识符的应用已在超媒体应用程序接口内得以证实。

起步

了解 Redfish 应用程序接口的一个简易方法是查看使用中的实施情况，为此，下文将简单介绍实施，并通过实施过程对多种架构结构和理念进行解释。

您需要两样东西开始操作——浏览器以及利用浏览器实施访问。或者，您可以使用文件浏览器或管理器阅读该文件。

由于该文件是以 REST 和 JSON 为基础的，您需要一个浏览器，能够启动查看 Redfish 实施过程。为便于更容易读取数据，建议您使用能够查看 JSON 格式的浏览器。可通过插件为大多数浏览器增加 JSON 格式功能。为了更真实地体验 Redfish，您可以为您的浏览器下载一个 RESTful 插件，例如，Chrome 的高级 REST 客户端插件。这能让你设置标题、查看超文本传输协议代码和其它浏览器正常隐藏的条目。当访问真实的 Redfish 实施过程时，您还将因此获益，诸如模拟器或供应商实现。

4. 访问实施

您可以直接或通过网络服务器查看实物模型。您所要做的就是访问实施。有如下几种方法：

- 可在 DMTF 网站的“实物模型开发”页下查看公共实物模型。网站允许用户查看样本载用于各类资源。
- 将数据拷贝至您的网络服务器。在 DMTF 网站上的 Redfish”应用程序接口页发现的 DSP2043 从是一个压缩包文件，内含多组实物模型。您可将特定的实物模型路径拷贝至您当地网络服务器的“/redfish/v1”文件夹内。
- 将数据拷贝至“Redfish 实物模型服务器”，从而启动本地 web 服务器并将其指向一组模型文件。

5. 根

Redfish 是一款超媒体应用程序接口，意味着你可以通过其它资源返回的 URL 获取所有资源。

但是，仅有一个众所周知的 URLs，为此，每次实施都有一个共同的起点。第一版 Redfish 接口的统一资源标识符是“/redfish/V1”。

URLs 拥有一个架构（HTTP://部分）、一个节点（如 www.dmtf.org.cn 或 IP 地址，如 127.0.0.1）以及一个资源部分。将这些一起放在浏览器的 URL 内。因此，如果你在自己的机器上使用了 nginx 服务器，你可以在浏览器内输入“HTTP://127.0.0.1/redfishi/v1/”访问 Redfish 根。

请从上述已经建立的 URL 中获取服务根。

基本理念

每一个 URL 代表一个资源。它可以是服务、集合、实体或一些其它结构。但在 RESTful 条款中，这些 URIs 指资源以及与资源互动的客户端。当你看到“资源”这个术语时，你可以将其想象为你访问 RUI 时所取回的东西。

资源格式由架构定义。每个资源都有一个特殊的格式，该格式在 Redfish 该架构内有说明，客户端可用于确定关于资源的语义（尽管我们在尽可能地让事物直观）。Redfish 架构有两种定义格式：一种是 OData 格式，另一种是 JSON 架构格式。OData 架构格式定义（CSDL），通用的 OData 工具和应用可对其解释。在 JSON 架构格式中为其它环境定义，像台风脚本、JavaScript、和虚拟化。

打算将资源的结构属性用作 JavaScript 变形，加速应用并允许 JavaScript 网页以及应用能直接使用数据。URIs 在重新启动过程中会反复出现，但客户端希望在/redfish/v1/路径下开始 ，并从这发现 URIs。

一个常见错误是 URI 的安装。Redfish 是一个超媒体应用程序接口，因此其在实施间各不相同——甚至于来源于相同的卖方。当前状态结果可与想达到的状态结果分隔开来。

6. 操作

操作有 GET、PUT、PATCH、POST、DELETE 和 HEAD。GET 指在未使用高级 REST 插件时浏览器所将做出的行为。只有模拟器和真实的实施方能支持其他操作。

GET 检索数据。POST 用于创建资源或使用行动（稍后详述）。DELETE 则会删除资源，但目前仅有少量资源可被删除。PATCH 用于更改一个资源上的一个或多个属性，而 PUT 可完全替换资源（尽管仅有少量资源可被完全替换——稍后详述）。HEAD 类似 GET，无数据退回，利用程序访问 Redfish 实施计算 URI 结构。

7. 版本

Redfish 有两种版本，一是协议版本，另一个是资源架构版本。协议版本在 URI 内——这也是你需要在/redfish/v1/启动的原因，指正在访问 1 版协议。1 版是现在唯一可用的一个版本，

但是我们需要提供潜在的未来版本。启动 URI 说明实施符合版本 1 的 Redfish 规范要求。请注意这是以 OData v4 基础的，因此在实施还要求 OData 协议标题（OData——版本）带上数字 4。

每个资源都有资源类型定义。资源类型在版本命名空间内定义。每个资源实例都有使用 OData 类型注释“@odata.type”表示的类型。类型注释值为资源类型的 URI，包括版本命名空间。为此，当看到“@odata.type”：“#ServiceRoot.v1_0_0.ServiceRoot”时，说明你正在处理的资源遵守在 1.0.0 版的 ServiceRoot 架构中定义的 ServiceRoot 类型定义，对应的架构文件位于 Redfish 架构资源库内的/schema/v1/ServiceRoot。该类型的完整 URI 则是“/schema/v1/ServiceRoot#ServiceRoot.v1_0_0.ServiceRoot”。架构文件有可能包含资源类型定义使用的其他类型（例如，结构类型和枚举类型），它们有相同的资源通道，但是碎片描述的却是不同的类型定义，在相同的命名空间内的最为典型。

8. 引用

当看到有链接时，说明有其他资源可引用。

JSON 无本地引用类型来参考其他资源。Redfish 需要有一个完全连接的资源树，但不要求客户端引用架构（针对简易操作）。由此，本规范利用 OData 惯例为其他内部或外部资源提供参考。

说明引用其他资源的属性遵循 OData 惯例要求，用属性名加“@odata.id”后缀形式来确定。代表 URLs 引用其他类型的属性，例如外部帮助主题，在元数据中用注释确定为字符串属性，说明它们代表 URL。

URIs 既不是绝对的也不是相对的。绝对的 URIs 没有网络协议地址，但以/redfish/v1/开始。如果有类似 Chrome 高级 REST 客户端这样的插件，点击该插件填写 URI 进行下次 GET。

9. 主对象

“主”对象是系统、管理器和机架。它们都是合集（见下一个标题）。我们会即刻深入研究这些资源，但对它们有所了解是件好事。

系统指典型的服务器。从中央处理器访问的数据层内的任何内容均代表一个系统，它们都在一个系统合集内，拥有中央处理器、存储装置和其它装置。

管理器指磁泡记忆体控制器、外壳管理器或其它正在管理基础设施的部件。管理器不仅要处理各种各样的管理服务，还拥有自己的设备（如网络完整性控制系统（NICs））

机架指基础设施的物理方面和控制。基础设施内的机框、外壳、刀片等，所有这些及其它更多方面均属于机架。因此有规定，机架内的机架，指覆盖传感器、风扇和诸如此类的机架。

系统有一个或多个管理器（因为有些管理器是多余的），并在一个机架内。管理器存在于机架内，并能够管理多个系统。机架则覆盖多个系统和/或管理器。

这仅仅是概要，但在查看 ServiceRoot 对象时，便能立刻看到这些。同时也请注意这里还有许多服务，如会话控制（稍后详述）。

10. 合集

相似资源组即为合集。实物模型内包含的合集有系统、管理器、机架、logEntries、Sessions、EventSubscriptions 及更多。

选取系统、管理器或机架并进入，就会发现这是一个合集。

合集反应有一个被称为“Members@odata.count”的数属性以及一个包含成员清单或成员链接的“Members”对象。成员链接即是 JSON 对象，带有一个单一的“@odata.id”属性，包含成员的 URL 信息。

合集可编页；带“@odata.nextlink”属性名的合集是不完整的，客户端会使用下一个链接属性的 URL 值从服务中检索该合集的下一部分。

11. 常用属性

当你遍览模型时，你会不断看到一些相同的属性。会在每一个资源内找到“名称”和“身份号”。“名称”和“身份号”是必须的。同时还会看到“状态”作为嵌入对象，其在所有应用中有相同的定义。所有这些其实都是架构的常见部分，并被其它架构引用。各个资源架构通过架构内的 odata“引用”元素引用常用属性，以便在任何地方使用同一定义。属性例子如下：

- “行动”，告知客户端哪个行动能被激发。更多信息，请见“行动”章节。
- “原始设备制造商”，对资源的标准定义具有特殊的供应商延展性。更多信息，请见“原始设备制造商”章节。

12. 常用注释

此外，还有注释属性。以“@”开头的属性是整个对象的注释，而“@”在属性名中间时被用以注释单个属性。允许有两种注释属性：OData 和 DMTF 注释。OData 注释中带有“@odata.”属性例子如下：

- “odata.type”用于查找定义资源的架构。
- “@odata.id”有资源 URL。这是一种超文本引用，但因 Redfish 是以 OData 为基础的，该属性便被称为“@odata.id”，而非“href”。

- “Members@odata.count”定义“Members”阵内的实体数量。

DMTF 注释中带有“@Redfish.”属性例子如下：

- “@Redfish.Settings”用来说明资源配置（稍后详述）。

另一个常用注释是“@odata.context”。这个实际上是针对通用 OData v4 客户端的。该属性在 OData v4 规范内定义，但基本上“@odata.context”能用于不同情况下：

1. 提供元数据位置，介绍载情况。
2. 提供根 URL，解决相对引用

@odata.context 结构是元数据文档的 URL，该文档带有描述数据的碎片。

技术上来说，元数据文档仅可定义或引用其直接使用的任一类型，不同的负载引用不同的元数据文档。然而，由于@odata.context 提供了一个根 URL 解决相对应用（诸如@odata.id 的），我们就必须返还“权威的”元数据文档。而且，“@odata.type”注释是按碎片而非完整 URLs 写的，元数据文档必须定义或引用这些碎片。另外，我们对带无版命名空间别名的行动进行了限定，在所引用的元数据文档中也必须通过引用对这些别名进行定义。

例如，在资源 /redfish/v1/Systems/1 中，你会看到带“/redfish/v1/$metadata#ComputerSystem.ComputerSystem”值的“@odata.context”属性。告知通用的 OData v4 客户端查找$元数据内的计算机系统定义，其对整个实体定义进行了引用。

你还会看到被称为“@Redfish.Copyright”的注释。实施不会返还属性。在这仅对在实物模型中使用的静态示例响应提出版权声明。

13. 行动

REST 并非万能，为此我们发明了“行动”。像系统上的“按钮”（依据设置情况重启或关闭系统）这类事物不能轻易在系统内体现，因为服务不清楚按键状态，从而无法按属性显示。另一个用途是针对长期操作的，长期操作作为一种原子动作更容易表达，为客户端提供便利——如固件更新或优雅关机等。

为此，我们创建“行动”代替要 PUT/PATCH 的隐藏属性或复杂状态机。利用 POSTs 资源完成行动（详情，见规范）。可通过查找“行动”属性得知资源内所支持的行动。

14. 架构

架构含有大量客户端能用的信息。建议客户端，尤其是编程客户端检查架构，确定属性周边语义。架构含有数据类型、枚举清单、信息描述、属性行为的标准信息和其它信息。

更多概念

是 Redfish 应用程序接口最简单的格式。你知道数据是可读的，但你可能还想了解安全性、架构内容、标题内容等等。

15. 会话控制

正常情况下，只有根可以在无需建立会话控制时被访问。但如果你正在使用公共实物模型或者本地的网络服务器，会话控制则毫无必要，因为没有安全性。如果你有一个在安全模式下运行的模拟器，或正在进行非模拟操作，则需要建立会话控制。

首先，你需要一个有效用户名和密码。

用于建立会话控制的 URI 也可允许用于无担保的 POST，以便建立会话控制。大多数情况下 URI 是/redfish/v1/Sessions，并可通过查看链接下的会话控制属性以及在服务根（/redfish/v1/）中查找@odata.id 属性值来确定。

会话控制通过 HTTP POST 在 URI 创建，用/redfish/v1#/Links/Sessions/@odata.id 说明的 URI 包含如下 POST：

```
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>"
}
```

退回包含有带会话控制标记的 X-Auth-Token 标题和位置标题。

JSON 退回含有新创建的会话控制对象。

```
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>"
}
```

在随后的所有服务请求中，在相同标题中的响应 X-Auth-Token 标题内使用标记字符串。当是时候删除会话控制时，在回复的@odata.id 处已退回的 URL 上进行删除操作（上述事例中的/redfish/v1/Sessions/Administrator1）

16. 冗余

返回至其中一个机架，跟着链接到“热”处看看风扇，你就会发现 Redfish 呈现冗余的方式。

你会注意到一个称为“冗余”的阵列，显示在设置内有两个风扇在关联性属性中使用相同的数值。冗余在 Redfish 内有一个常用的定义架构，内有其它属性证明冗余的其它重要特性。通过此，客户端能计算出哪类项目归属于那个冗余设置，因为@odata.id 值是冗余设置成员的指标。

“@odata.id”属性值不一定要注释整个资源。属性值有两种格式：JSON 指标或 OData 引用。

- 当为 JSON 指标时，其内会带#说明资源停留的地方和属性模式归属地。架构也会引用该属性。JSON 指标值示例“/redfish/v1/Chassis/1/Thermal#/Fans/0”。
- 当为 OData 引用时，其内无#。架构会有属性定义。JSON 指标值示例“/redfish/v1/Chassis/1/Thermal/Fans/0”。

17. 关联项

如果依然处于“热”资源内，你会看到温度阵元素内有一个带“@odata.id”的关联项属性。这是子资源的引用，温度传感器正在测量——有可能是处理器。类似冗余链接，该值指向子资源。为此，客户端可确定温度传感器正在测量的物体。

关联性具有常用架构定义，在使用时是环境独立的。然而，关联性常常用来证明两个不同资源或子资源间的关系。

与冗余一样，“@odata.id”属性值不一定指整个资源。

18. 服务

返回根，看一看一些常见的服务，包括任务、会话控制、事件服务和账户服务。可在规范中读取更多有关它们的内容，但其实从它们各自的命名方式即可了解它们提供的服务。任务包含有可能要启动的工作清单，通常作为行动结果。会话控制在上文中已叙述。账户服务指创建用户的方式，事件服务则在下文详细叙述。

19. 注册

信息注册允许管理处理器在活动及错误下保留一个简短的标示符以供使用，客户端可在注册中查找确定真实信息。以每条消息为基础，参数替换机制已就位。

通过将信息注册为单独实体，便于支持国际化，这是因为管理处理器无需带翻译。相反地是，注册本身能被翻译，客户端能够显示所需要的翻译信息。

20. 标题

Redfish 服务支持常用 HTTP 标题的子设置。规范内有完整标题清单，但最为两关的两个则是 Accept 标题（必须设置为“application/json”）和先前叙述的 X-Auth-Token。然而，从实

际实施中，你能够得到更多的标题信息，但是在静态的实物模型中是看不到的。

21. ETags

浏览器使用 ETags 优化 IO（缓存）。ETags 会进行 If-Match 操作，如果匹配，则不会麻烦拖拽数据。我们用它们来决定是否对象有变更。因此，如果有进行 PUT 操作或工具就位（如操纵台的 BIOS 配置），每个 ETag 都会变化。要时常注意 Redfish 与非 Redfish 客户端（以及其它 Redfish 客户端）之间的 PUTs 的竞争条件。

问题是这个一个实物模型，而非真实的网络服务，为此你看不到 ETags 工作。

21.1. 处理客户端竞争条件

有竞争性的客户端有可能试图设置设定数据相互覆盖取代。该情况可使用 ETags 和 If-Match 处理。

客户端读/更改/写循环包括：

1. 获取资源，包括当前 ETag
2. 变更资源属性。
3. 生成新的 ETag。
4. 对已变更的包含有 If-Match 的资源进行 PATCH，涵盖上一条已知的 ETag。
5. 如果供应的 ETag 与对象不再匹配（读后有变更），服务会退回，提示 412 先决条件失败。客户端要重复这些步骤恢复。
6. 当更新设定数据时，客户端应包含 ETag，使能供应商更有效的检测变化情况，防止多客户端竞争条件。

22. 更新资源

简易的 GET、PUT/PATCH 方法是一种模型，在此客户端可能获取当前配置，PUT 或 PTACH 一个新的预期配置至相同的资源 URI。通过 HTTP 状态代码和相应机制确定 PUT 或 PATCH 是否成功。

这样是为了与内在提供商，如 Redfish 服务的自有数据进行交互。固件可处理并适当响应操作和配置变更。

一旦 PUT 或 PATCH，HTTP 响应会包含一个 HTTP 状态代码，若出现失败，则会包含一个作为响应体的扩展错误信息结构。

22.1. PUT vs PATCH

为什么是同时 PUT 和 PATCH 呢？在设计本规范时，我们在何时进行完整资源替换代替部分替换上遇到了语义问题。有些资源既允许部分属性替换，也允许完整的资源替换，因此我们需要一种方法来确定何时清理资源内的现有属性，像设定数据。事情开始变得非常复杂了，已针对此实施了特定（非自然的）规则，并尝试使用单一操作。

PATCH 解决了这个问题。PUT 往往是完整替换。PATCH 一直是部分替换。客户端或服务器无需复杂逻辑就能实现服务决定行为。

PATCH 在行业中获得了广泛应用，且已获得了开放栈和其它应用程序接口的支持，并在多个壳网络服务器中可用。

22.2. 当前配置 vs 设置

Redfish 内基本上有两种对象——当前配置和设定。大部分对象说明了任一指定资源的当前状态。在一个资源内，偶尔会看到一个名为“@Redfish.Setting”的属性。该注释有链接，会告诉你在何处为配置进行 PUTs 和 PATCHs 操作。它说明了资源的未来状态。

有些资源能即刻处理变更，其他的则需要重启系统或服务。“@Redfish.Setting”用于让客户端了解资源类型以及在何处做变更处理。

如果你看见“@Redfish.Setting”属性，它带有一个资源链接，方便在下一次选中时进行变更，像重置或重启。如果看不到设置链接，要立刻对对象进行 PATCH 操作（在衍生任务缺失情况下）。

如果设置能适用于“直播”，还会显示上一次设置适用于资源的时间信息——时间、ETag 和任何已退回的信息。

有可能需要设置的资源是类似 NICs、存储器和 BIOS 等的设备。

22.3. 确定能更新的属性

架构和元数据均能定义可更新的属性。标有“OData.Permissions/Read”或仅为只读的属性说明这些属性永不可更新。相反地，“OData.Permissions/ReadWrite”或只读=错误说明属性是可写的。长描述（LongDescription）（标准文本）说明了在何种条件下是可写的属性。请注意，无“Permissions”注释的元数据或无“只读=正确”注释的架构默认为是可写的。

请注意，即使它们标记为可写，这并不说明该属性是可写的，因为实施只允许属性可读。

复杂类型属性上的许可是复杂类型内所有属性的默认许可，无“Permissions”注释。例如，IPv4Address 复杂类型内的 SubnetMask 属性无“Permissions”注释。EthernetInterface 内的 IPv4Address 属性是 IPv4Address 类型。如果 IPv4Address 带有“OData.Permission/Read”注

释，SubnetMask 会变为只读。如果 IPv4Address 带有“OData.Permission/ReadWrite”注释，SubnetMask 会变为读写。为明确，最佳操作是为复杂类型内的所有属性定义“Permissions”注释。

Redfish PUT/PATCH 的语义是试图更新一个不可写的属性不会被视为错误。Redfish 服务的实施会退回，提示“200 带扩展错误信息”，说明属性不能被升级更新。

23. 扩展错误响应

单独的 HTTP 错误语义不会为用户或应用在了解起因以及采取正确措施方面提供足够信息参考，从而引发一个扩展错误响应结构概念。扩展错误响应也支持注册，并容许客户端存在更多有意义的错误语言。

例如，假设客户端想要立刻更改大量属性。然而其中一个属性是无效的。如果实施退回提示“400”，就无法帮助客户端了解是哪个属性存在错误以及错误原因。或者是如果退回提示“200”，但其中一个属性不存在，则说明这是一个无意义的响应。

如果响应伴随有带扩展错误响应的 JSON 体，该结构不仅包含代码被退回的原因，还有属性名和更多的信息。因为扩展错误有一个信息阵，如果出现有多个问题时，多个信息可被退回。

24. 事件

我们需要多种事件机制以满足 SNMP、IPMI 和其它协议要求。机制在 BMCs 内是已经存在的。选取的方法论是基于订阅的推送（PUSH）机制。BMCs 更喜欢 PUSH 事件，原因是一旦事件发送，它不会在存储器内反复出现。

客户端首先需要确定 URI 需要事件发送的对象。通过向 EventSubscription 合集提交 POST 请求，可以建立会话控制。

Redfish 支持两类不同的事件：生命循环和警报事件。可通过实施，查看 EventService 了解所支持事件的类型。

本规范介绍了更多有关事件信息。

25. 创建用户账号和其它资源

类似事件和会话控制，通过 POST 操作创建资源。这通常是通过 POST 到合集。EventSubscription、会话控制和账户资源均有合集。这些合集的 URI 与其它资源一样可在超媒体应用程序接口内被发现。

架构对名为“RequiredOnCreate”的注释进行了定义，客户端可据此确定 POST 内必须要提供的属性。

26. 讯息对齐

架构师在 Redfish 内尝试的一件事是对齐所有信息格式。事件信息、扩展错误响应讯息、日志访问和信息注册均已对齐。当供应商特定格式和其它格式能在一个实施内保存时，会有一个信息格式对齐通道，不仅能使信息国际化，还能优化存储，并有望将客户端和访问及呈现数据的服务任务进行统一。

27. 原始设备制造商

在实物模型及可能的实施中，你会看到一个名为“Oem”的架构。该架构会在 3 处地方出现：作为资源的基础属性，在“链接”章节中或在“行动”章节中。Redfish 对机制进行了定义，以便供应商通过在“Oem”对象内增加自身对象囊括自有扩展。Oem 必须要有“odata.type”属性，以便客户端查到架构文件。除此之外，标准对要求不做主张，但它要是一个机制，多供应商环境能使用相同的架构潜在布置扩展作为标准资源。这样避免了对多个 IOs 的需求，并允许可扩展性。希望供应商随着时间推移能够引入特性，使其成为标准的一部分。

28. 幂等性

RESTful 服务的操作有 GET、PUT/PATCH、POST 和 DELETE。

幂等性是 Redfish 简化的重要概念。在第一次之后，幂等性操作能被反复激发，不会产生额外影响。例如，DVD 遥控器上的“STOP”键就是一个幂等，按下开关一次，电影停放，按开关多次，电影仍处于停放状态。换句话说，暂停键不是幂等。如果按暂停键两次，电影会再次播放。非幂等性操作以上一次状态而定。

幂等性操作有 HTTP PUT 和 PATCH。如果客户端第二次 PUTs 相同的数据，资源无变化。PATCH 替换属性，所以重演的 PATCH 不会再次更改资源，因为自第一次 PATCH 后就已经在预期状态内了。

资源 PUTting 有可能是最简单的配置变更表达，意指“用资源 Y 替换资源 X”。其它互动模式均要比它复杂。PATCH 稍微复杂点，因为它要求对资源进行差异化更新。能够通过幂等 PATCH 操作完全配置的服务就是交易。

Redfish 使用基础 REST 操作定义了 3 个基本的互动模式。

28.1. 幂等更改：GET/PUT/PATCH

资源 PATCHING 是指用已有属性的新数值替换资源内容。当服务能简单的消耗预期配置并产生结果状态时，就需要使用 PATCHing。大概情况是客户端有可能获取一个资源，更改属性值，然后 PATCH 提交更改。

资源 PUTting 是用新值替换资源内容。当客户端能简单将资源设置为预期状态时，则可使用 PUTting 操作。

由于多个客户端能访问一个资源，读-修改-写循环会导致竞争条件。通过在资源上使用 ETag 标题以及在随后的 PATCH 或 PUT 请求中在 If-Match 或 If-None-Match 内提供 ETag 值，客户端可避免此类情况。当更新时，服务也会更新 ETag。当第二个客户端尝试用旧的 ETag 值修改资源时，服务有能力驳回 PATCH 或 PUT 的重合请求。

28.2 创建、使用、删除：POST/GET/DELETE（非幂等）

资源 POSTing 是指创建一个子资源。在数据模型中应用较典型，数据模型内的条目必须创建及删除。创建示例包括新用户账号创建、用户账号删除、新日志访问、增加或移除授权等。这些条目也支持删除。通过 POSTing 一个 URI 内容完成创建。新创建的资源是 URI 的下一代，可通过 DELETE 新 URI 进行删除。

28.3 行动：用“行动”属性 POST（非幂等）

创建资源行动属性中被退回的行动会导致定制行动的执行。该模式非幂等，是因为相同内容的再次创建会导致额外工作的执行。发送一个试验事件或清空日志均是定制行动。

29. 继承和多态

你看不到的条目之一是继承和多态。

从一个单一主对象处继承的对象被称为资源。除此之外，Redfish 不使用继承。同样地，Redfish 也不会采用常见的属性并将它们放入父对象。因此，在任何架构内没有子对象可详细说明。你永远看不到子类型与父类型一起退回到同一合集。

30. 拷贝继承

Redfish 采用常见定义。如果定义不在资源内，作者会从其它架构剪切并粘贴所要的对象，并将它们包含在该对象中。这样给作者带来了一个保持事情同步的负担，但同时也防止读者通过引用挖掘另一个引用只为查出特定对象的语义。

31. 多态联盟

Redfish 采用有相同语义的资源，并将所有定义放入同一资源。通过这种方法，相似的资源就会有相似的语义。实施方无需使用不受支持的属性。例如，如果有 3 个不同的设备，它们有 85%的相似度，每个设备 15%的差异就会涵盖在常见的语义定义内。实施方然后可使用资源所需的 85%这部分以及将 15%的定义用于正在实施的设备。这个方法会产生较大的架构，但也会产生较小的架构。当看到这个方法时，在架构和资源内通常还会看到一个额外的“资源类型”风格属性，帮助客户端告知设备特点。

结论

Redfish 应用程序接口为信息技术提供了一种新的编程风格，能够以一致的方式管理系统，从超范围到刀片再到独立服务器。我们认为 Redfish 是一种自然演变，即适应用户需求又与这些用户正在使用的工具匹配。

常用案例

这里又一些使用案例，帮助你了解架构并开始客户端代码。

32. 查找系统温度传感器

应用代码常常在根启动：/redfish/v1/

1. 在根内对象是一个被称为“系统”的属性
    1. 查找“@odata.id”值。它是系统合集 URI。
    2. 在实物模型中，URI 是/redfish/v1/Systems
    3. 在 URI 上进行 GET 操作
2. 在“Members”阵处查看“链接”对象
    1. 查找讨论中的系统“@odata.id”（有可能要求在各系统上进行 GETs 操作，并查看系统内属性确定正确的系统）
    2. 在实物模型中，URI 是/redfish/v1/Systems/1
    3. 在 URI 上进行 GET 操作
3. 在“链接”对象内查找被称为“机架”的对象
    1. 查找“@odata.id”值。它是内含系统的机架
    2. 在实物模型中，URI 是/redfish/v1/Chassis/1
    3. 在 URI 上进行 GET 操作
4. 在“链接”部分查找被称为“热”的对象
    1. 查找“@odata.id”值。它内有温度传感器
    2. 在实物模型中，URI 是/redfish/v1/Chassis/1/Thermal
    3. 在资源上进行 GET 操作
5. 在“温度”阵中查找——它是系统温度传感器
    1. 查看“关联项”，明确找出各温度传感器监控的部件。

33. 机架内的机架

应用代码常常在根启动：/redfish/v1/

1. 在根内对象是一个被称为“系统”的属性
    1. 查找“@odata.id”值。它是机架合集 URI。
    2. 在实物模型中，URI 是/redfish/v1/Chassis
    3. 在 URI 上进行 GET 操作
2. 在“Members”阵处查看“链接”对象
    1. 查找讨论中的机架“@odata.id”（有可能要求在各系统上进行 GETs 操作，并查看机架内属性确定正确的系统）
    2. 在实物模型中，URI 是/redfish/v1/Systems/Enc1
    3. 在 URI 上进行 GET 操作
3. 在“链接”对象内查找被称为“包涵”的对象
    1. 查找“@odata.id”值。它是机架内的机架
4. 在“链接”部分查找被称为“被包涵”的对象
    1. 查找“@odata.id”值。它是机架内的机架。

版本历史记录

版本	日期	描述
1.0.0	2015 年 8 月	首次发布
1.0.1	2015 年 8 月	将@DMTF 更正为@Redfish
1.0.2	2016 年 4 月	增加实物模型内的版权注释声明
1.0.3	2017 年 8 月	更新《访问实施》章节内的正确信息
		固定《常用注释》章节内的注释文件
		更新《常用注释》章节内的@odata.context 文档，应用简易格式
		更新《幂等更改》章节，提及客户端竞争条件
1.0.4	2018 年 9 月	当 If-Match 标题内有不正确的 etag 时，更正 PATCH 的 HTTP 状态代码，