# 云计算，雾计算，边缘计算

## 1云计算

随着物联网行业技术标准的完善以及关键技术上的不断突破，数据大爆炸时代将越走越近。如此海量的数据不及时处理利用起来，那么它们将很快变成数据垃圾。那么问题来了？我们不可能给每个终端装上一个计算机。如何解决海量数据的处理分析问题呢？们知道，每台服务器都有自己的CPU、内存，但分配到这些服务器的应用往往不能充分地利用这些资源。再者，为了确保服务的可靠性往往还要预留冗余的服务器、存储器、网络设备等，而很多时候，这些硬件资源往往处于空置状态，并没有得到充分的利用。最后，正确预测不同应用对服务器的计算能力和存储器的存储能力的需求又是困难的。因此，2006年Google的CEO埃里克·施密特首次提出了云计算的概念。

以下参考https://blog.csdn.net/gongxifacai\_believe/article/details/78655659

云计算是一种利用互联网实现随时随地、按需、便捷地使用共享计算设施、存储设备、应用程序等资源的计算模式。

云计算系统由云平台、云存储、云终端、云安全四个基本部分组成。云平台作为提供云计算服务的基础，管理着数量巨大的CPU、存储器、交换机等大量硬件资源，以虚拟化的技术来来整合一个数据中心或多个数据中心的资源，屏蔽不同底层设备的差异性，以一种透明的方式向用户提供计算环境、开发平台、软件应用等在内的多种服务。

通常情况下，云平台从用户的角度可分为公有云、私有云、混合云等。

公有云：第三方提供商为用户提供服务的云平台，用户可通过互联网访问公有云。

私有云：为一个用户单独使用而组建的，对数据存储量、处理量、安全性要求高。

混合云：是结合了公有云和私有云的优点而组建的。

通过从提供服务的层次可分为以下几个层次的服务：

基础设施即服务（IaaS）：消费者通过Internet可以从完善的计算机基础设施获得服务。例如：硬件服务器租用。比如说

平台即服务（PaaS）：PaaS实际上是指将软件研发的平台作为一种服务，以SaaS的模式提交给用户。因此，PaaS也是SaaS模式的一种应用。但是，PaaS的出现可以加快SaaS的发展，尤其是加快SaaS应用的开发速度。例如：软件的个性化定制开发。

软件即服务（SaaS）：它是一种通过Internet提供软件的模式，用户无需购买软件，而是向提供商租用基于Web的软件，来管理企业经营活动。例如：阳光云服务器。

云计算的特点：

https://baike.baidu.com/item/%E4%BA%91%E8%AE%A1%E7%AE%97/9969353?fr=aladdin#4

云计算是通过使计算分布在大量的分布式计算机上，而非本地计算机或远程服务器中，企业数据中心的运行将与互联网更相似。这使得企业能够将资源切换到需要的应用上，根据需求访问计算机和存储系统。

好比是从古老的单台发电机模式转向了电厂集中供电的模式。它意味着计算能力也可以作为一种商品进行流通，就像煤气、水电一样，取用方便，费用低廉。最大的不同在于，它是通过互联网进行传输的。

被普遍接受的云计算特点如下：

(1) 超大规模

“云”具有相当的规模，Google云计算已经拥有100多万台服务器， Amazon、IBM、微软、Yahoo等的“云”均拥有几十万台服务器。企业私有云一般拥有数百上千台服务器。“云”能赋予用户前所未有的计算能力。

(2) 虚拟化

云计算支持用户在任意位置、使用各种终端获取应用服务。所请求的资源来自“云”，而不是固定的有形的实体。应用在“云”中某处运行，但实际上用户无需了解、也不用担心应用运行的具体位置。只需要一台笔记本或者一个手机，就可以通过网络服务来实现我们需要的一切，甚至包括超级计算这样的任务。

(3) 高可靠性

“云”使用了数据多副本容错、计算节点同构可互换等措施来保障服务的高可靠性，使用云计算比使用本地计算机可靠。

(4) 通用性

云计算不针对特定的应用，在“云”的支撑下可以构造出千变万化的应用，同一个“云”可以同时支撑不同的应用运行。

(5) 高可扩展性

“云”的规模可以动态伸缩，满足应用和用户规模增长的需要。

(6) 按需服务

“云”是一个庞大的资源池，你按需购买；云可以像自来水，电，煤气那样计费。

(7) 极其廉价

由于“云”的特殊容错措施可以采用极其廉价的节点来构成云，“云”的自动化集中式管理使大量企业无需负担日益高昂的数据中心管理成本，“云”的通用性使资源的利用率较之传统系统大幅提升，因此用户可以充分享受“云”的低成本优势，经常只要花费几百美元、几天时间就能完成以前需要数万美元、数月时间才能完成的任务。

云计算可以彻底改变人们未来的生活，但同时也要重视环境问题，这样才能真正为人类进步做贡献,而不是简单的技术提升。

(8) 潜在的危险性

云计算服务除了提供计算服务外，还必然提供了存储服务。但是云计算服务当前垄断在私人机构（企业）手中，而他们仅仅能够提供商业信用。对于政府机构、商业机构（特别像银行这样持有敏感数据的商业机构）对于选择云计算服务应保持足够的警惕。一旦商业用户大规模使用私人机构提供的云计算服务，无论其技术优势有多强，都不可避免地让这些私人机构以“数据（信息）”的重要性挟制整个社会。对于信息社会而言，“信息”是至关重要的。另一方面，云计算中的数据对于数据所有者以外的其他用户云计算用户是保密的，但是对于提供云计算的商业机构而言确实毫无秘密可言。所有这些潜在的危险，是商业机构和政府机构选择云计算服务、特别是国外机构提供的云计算服务时，不得不考虑的一个重要的前提。

## 2 雾计算

我们知道，将数据从云端导入和导出实际上比人们想象的要更为复杂，由于接入设备越来越多，在传输数据、获取信息时，带宽就显得不够用了，这就为雾计算的产生提供了空间。

雾计算的概念在2011年被人提出，是由性能较弱、更为分散的各种功能计算机组成，渗入电器、工厂、汽车、街灯及人们生活中的各种物品。雾计算雾计算是一种对云计算概念的延伸它主要使用的是边缘网络中的设备，是半虚拟化的服务计算架构模型，强调数量，不管单个计算节点能力多么弱都要发挥作用。

特点：

雾计算有几个明显特征：低延时、位置感知、广泛的地理分布、适应移动性的应用，支持更多的边缘节点，安全。这些特征使得移动业务部署更加方便，满足更广泛的节点接入。

雾计算是一种对云计算概念的延伸它主要使用的是边缘网络中的设备，数据传递具有极低时延。

雾计算具有辽阔的地理分布，带有大量网络节点的大规模传感器网络。

雾计算移动性好，手机和其他移动设备可以互相之间直接通信，信号不必到云端甚至基站去绕一圈，支持很高的移动性。

云计算与雾计算的区别。

|  |  |  |
| --- | --- | --- |
|  | 云计算 | 雾计算 |
| 数据存储位置 | 云中（服务器） | 网络边缘的设备中（更依赖本地设备） |
| 架构 | 集中式 | 分布式 |
| 单个服务器的计算性能 | 要求高 | 比较低 |
| 移动性 | 好 | 更好  手机和其他移动设备之间可以直接通信，信号不用传到云端 |
| 安全性 |  | 更高 |
| 计算模式 | 网络中心 | 网络边缘 |

## 3边缘计算

边缘计算指在物理上靠近物或数据源头的网络边缘侧，融合网络、计算、存储、应用核心能力的开放平台，就近提供边缘智能服务，满足行业数字化在敏捷连接、实时业务、数据优化、应用智能、安全与隐私保护等方面的关键需求

有雾计算为什么还要用边缘计算？

边缘计算，进一步推进了雾计算的「LAN内的处理能力」的理念，处理能力更靠近数据源。不是在中央服务器里整理后实施处理，而是在网络内的各设备实施处理。这样，通过把传感器连接到可编程自动控制器(PAC)上，使处理和通信的把握成为可能。和雾计算相比的优点，根据它的性质单一的故障点比较少。各自的设备独立动作，可以判断什么数据保存在本地，什么数据发到云端。

有些数据不需要传到云上处理的时候，却传到了云上，这样在传输的过程中可能增加故障的可能性。

那么，边缘计算和云计算又有何区别？这两者都是处理大数据的计算运行方式。但不同的是，这一次，数据不用再传到遥远的云端，在边缘侧就能解决，更适合实时的数据分析和智能化处理，也更加高效而且安全。

边缘计算的优势：

1实时或更快速的数据处理和分析：数据处理更接近数据来源，而不是在外部数据中心或云端进行，因此可以减少迟延时间。

2较低的成本：企业在本地设备的数据管理解决方案上的花费比在云和数据中心网络上的花费要少。

3网络流量较少：随着物联网设备数量的增加，数据生成继续以创纪录的速度增加。因此，网络带宽变得更加有限，让云端不堪重负，造成更大的数据瓶颈。

4更高的应用程序运行效率：随着滞后减少，应用程序能够以更快的速度更高效地运行。

5削弱云端的角色也会降低发生单点故障的可能性。

6安全性高，由于边缘设备能够在收集和本地处理数据，数据不必传输到云端。因此，敏感信息不需要经由网络，这样要是云遭到网络攻击，影响也不会那么严重

边缘计算的另一个关键优势与安全性和合规性有关。随着政府越来越关注企业如何利用消费者的数据，这一点尤为重要

# EdgeX Foundry

# 1整体介绍

## 1.1 介绍

EdgeX Foundry是一系列松耦合、开源的微服务集合，位于网络的边缘，可以与设备、传感器、执行器和其他物联网对象的物理世界进行交互。EdgeX Foundry 旨在创造一个互操作性、即插即用、模块化的物联网边缘计算的生态系统。

南侧:在物理领域内的所有物联网对象，以及与这些设备、传感器、执行器和其他物联网对象直接通信并从中收集数据的网络边缘，统称为“南侧”。

北侧:将数据收集、存储、聚合、分析并转换为信息的云(或企业系统)，以及与云通信的网络部分称为网络的“北侧”。

EdgeX可以根据需要和指示发送“北”、“南”或“横向”数据。

因此，EdgeX使数据可以向北移动到云，也可以横向移动到其他网关，或返回到设备、传感器和执行器。

## 1.2 EdgeX Foundry包含的服务层

EdgeX Foundry是一个开源微服务的集合。这些微服务被组织成4个服务层和2个基础的增强系统服务。该微服务集合构成了四个微服务层及两个增强的基础系统服务，这四个微服务层包含了从物理域数据采集到信息域数据处理等一系列的服务，另外两个基础系统服务为该四个服务层提供支撑服务。

EdgeX铸造厂的4个服务层如下:

核心服务层

支持服务层

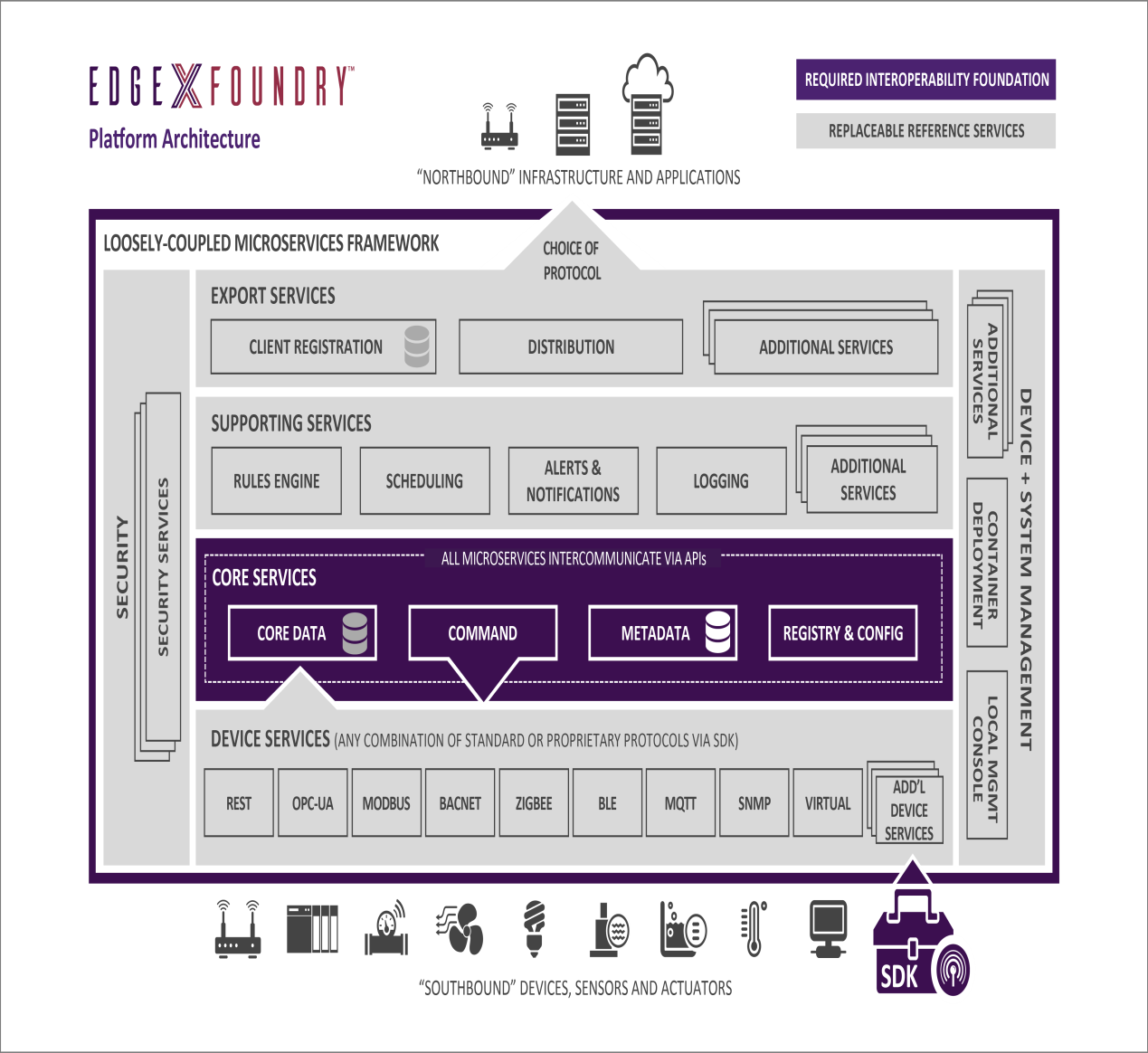
导出服务层

设备服务层

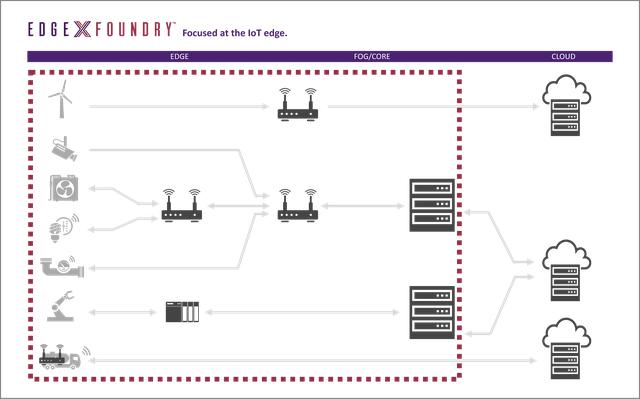
EdgeX铸造公司的两项基础系统服务如下:

安全

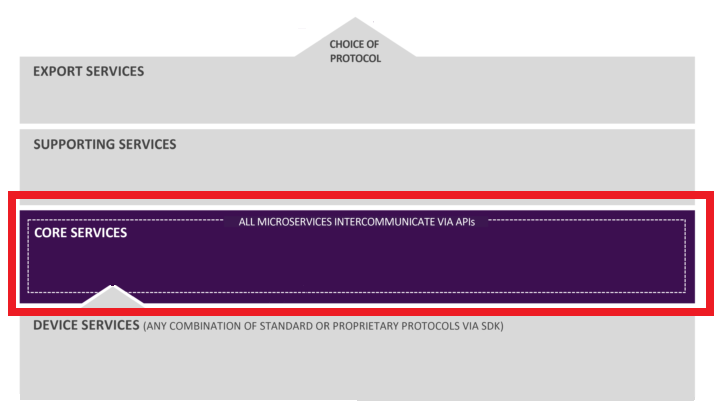
系统管理



从整体上来看，在最简单的实现中，单个网关设备具有运行在其上的EdgeX核心以及一组装置，支持和导出服务以及管理服务和安全服务。这成为边缘处理节点，在与云连接之前，与多个设备，传感器，机器等的交互在一起进行交互处理。



## 1.3核心服务层



核心服务(CS)层分隔了边缘的北侧和南侧层。核心服务包括以下组件:

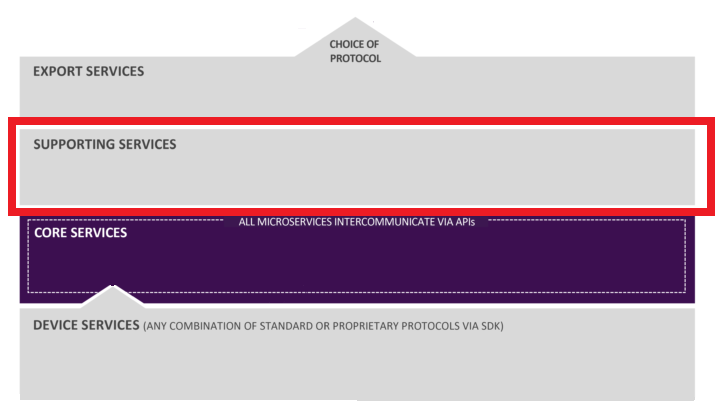
核心数据: 持久性存储库和从南侧对象收集的数据的相关管理服务。

命令: 处理北向应用发往南向设备的请求；当然该服务还会处理框架内其他微服务发往南向设备的请求，如本地的分析服务。

元数据:提供新设备的功能，并将其与拥有的设备服务进行配对。

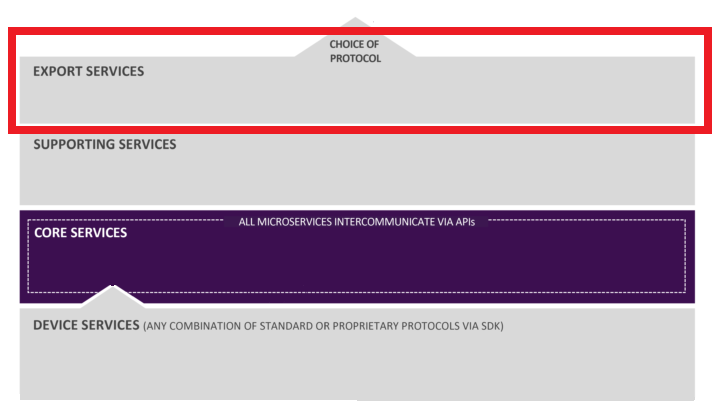
注册和配置: 为其他EdgeX Foundry微服务提供关于EdgeX Foundry内相关服务的信息，包括微服务配置属性。

## 1.4 支持服务层

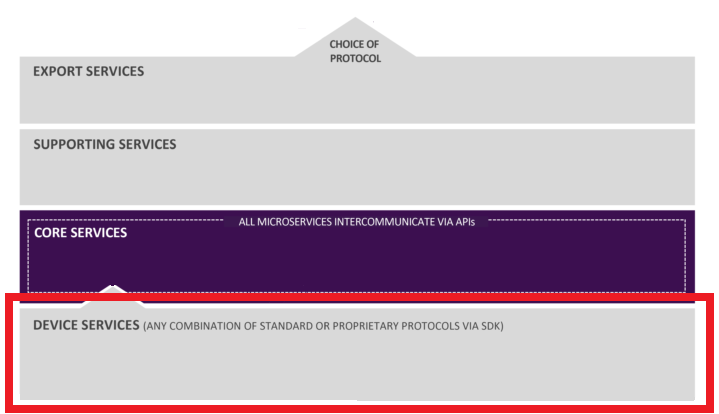


支持服务(SS)层包含广泛的微服务，该层微服务主要提供边缘分析服务和智能分析服务。 此外，该层还为EdgeX micorservices提供支持功能，如日志记录，调度和数据清理（清理）。规则引擎，警报和通知微服务 都在支持层内，因为它们在Core Services层上运行。 本地分析功能（目前仅以简单的规则引擎实现基本的分析功能）也位于此层。

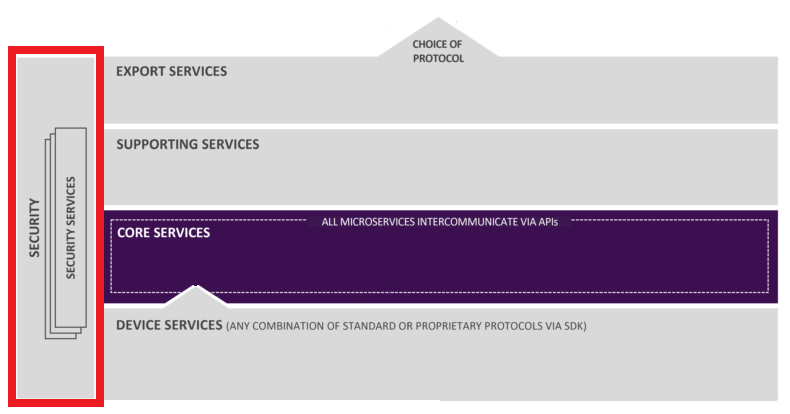
1.5 导出服务层

在必要情况下，EdgeX Foundry需要可以独立于其他系统运行。Edgex Foundry所依存的网关通常会在独立非联网环境下部署，同时监管一组传感器或设备。当网关在不联网环境下运行时，其监管的传感器及设备是不受外界环境监管或控制的。因此EdgeX Foundry在不连接北向应用的情况下，是可以长时间独立运行的。不过EdgeX Foundry收集的数据总归还是需要定期或不定期的传输给北向应用（通常为云端系统）。导出服务层就是为实现这个目的而存在的。导出服务层提供了一组微服务实现以下功能：  
北向应用可以在网关注册，并获取其感兴趣的南向设备的数据；  
通知数据何时被发往何地；  
通知数据传输格式；

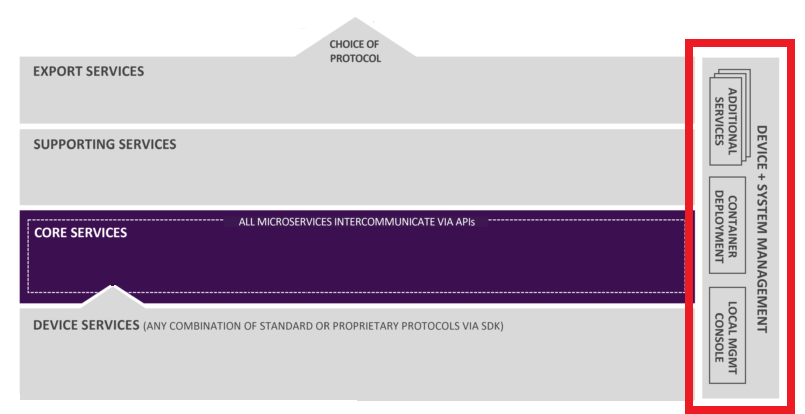
## 1.6 设备服务层

设备服务层负责与南向设备交互。  
设备服务是与南向设备或物联网对象交互的边缘连接器，包括但不限于：报警系统，家庭和办公楼中的暖气和空调系统，灯光，任何行业的机器，灌溉系统，无人驾驶飞机，目前自动化的运输，如一些铁路系统，自动化工厂，家用电器。未来，还可能包括无人驾驶汽车和卡车，交通信号灯，全自动快餐设施，全自动自助式杂货店，以及从病人身上读取健康数据的设备。  
设备服务可以同时服务于一个或多个设备（传感器，致动器等）。 设备服务层管理的“设备”可能不是简单的单一物理设备。它可能是EdgeX Foundry的另一个网关（以及该网关的所有设备），设备管理器或设备聚合器，充当设备或设备集合。  
设备服务层的微服务通过每个物联网对象本身的协议与设备，传感器，执行器和其他物联网对象进行通信。 设备服务层将由IoT（物联网）对象生成和传递的数据转换为常见的EdgeX Foundry数据结构，并将转换后的数据发送到核心服务层以及EdgeX Foundry其他层的其他微服务。

## 1.7 系统服务层



安全体系结构：EdgeX铸造厂内外的安全元件保护由EdgeX铸造厂管理的设备、传感器和其他物联网对象的数据和命令。



系统管理设施：提供EdgeX铸造微服务的安装、升级、启动、停止和监视，以及BIOS固件、操作系统和其他与网关相关的软件，还可以从离线的、基于企业的系统支持这些功能。

# 2 核心服务层具体介绍

## 2.1注册配置

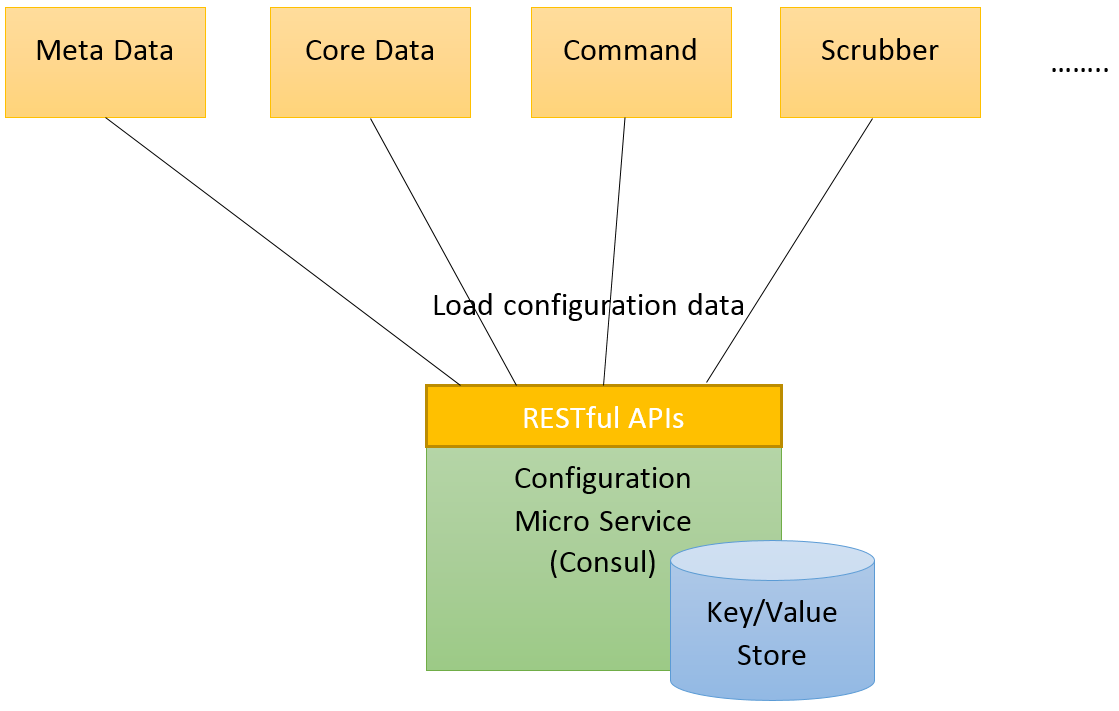
配置和注册中心微服务为EdgeX提供了集中式配置管理

其中包含了所有EdgeX微服务的配置和操作参数

作为EdgeX微服务注册中心，通过配置和注册微服务能够知道**所有EdgeX微服务**的位置和运行状态。当每一个EdgeX微服务启动时，都会将自己注册到配置和注册微服务。配置和注册微服务然后定期“ping”其他微服务，EdgeX铸造微服务可以在没有配置和注册微服务的情况下运行。因为没有配置和注册微服务，他只能通过本地（而非全局）提供初始化，对其他微服务的位置和操作状态进行假设。

### 2.1.1配置

对于配置，配置和注册表微服务集中并简化了不断增长的服务配置数据。配置微服务是作为它的基础的，开放源代码服务，提供了一个键和值存储，客户可以通过RESTful api访问EdgeX。



当每个微服务启动时，会出现一个特殊的“引导”阶段，以便加载自己的**配置数据**。在“引导”阶段，客户端微服务试图通过RESTful api访问配置微服务，并遵循命名约定检索自己的配置数据。此外，客户端微服务在本地准备默认配置文件。如果由于任何错误而无法到达配置微服务，客户端微服务将持续重试。当配置微服务不可用时，客户端微服务无法启动并报告错误消息。

在物理上，“配置”服务和“注册表”服务组合在一起并在同一个consul服务器节点上运行。

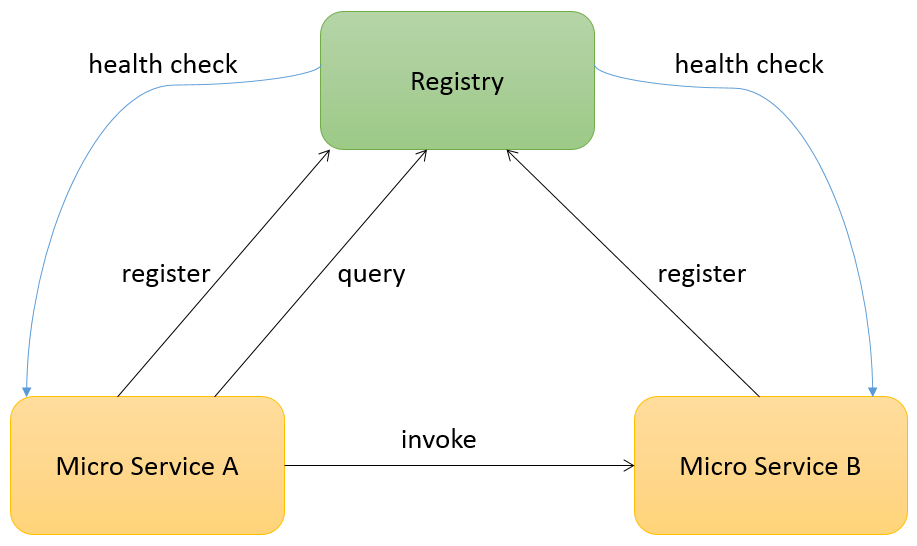
键/值存储

配置微服务利用consul提供的键/值存储来存储配置数据，每个键都包含一个由斜杠(“/”)分隔的名称空间，例如“/config/core-metadata/server”。端口"指定核心元数据微服务的端口号。

consul还提供了网络用户界面。用户可以通过web用户界面查看和设置键/值数据(可以设置acl)。

### 2.1.2注册

注册中心的目标是使微服务能够彼此发现和通信。当每个微服务启动时，它将自己注册到注册中心，注册中心将继续定期检查其可用性。当一个微服务需要连接到另一个微服务时，它连接到Registry以检索目标微服务的可用主机名和端口号，然后调用目标微服务。下图显示了基本流程。



注册中心也是使用consul作为其实现。consul提供服务注册、服务发现和健康检查的本地特性

Spring框架开发

由于Spring Boot在许多EdgeX铸造Java微服务中都使用，因此本节提供了一些指导方针和参考，用于使用consul技术开发Spring Boot应用程序。Spring有一个名为Spring Cloud consul的子项目，它通过自动配置和绑定Spring环境以及其他Spring编程模型习惯用法，为Spring引导应用程序提供consul集成。

Docker上运行

为了便于安装和更新，EdgeX铸造公司的微服务以Docker镜像的形式发布到Docker Hub，包括配置微服务:

<https://hub.docker.com/r/edgexfoundry/docker-core-consul/>

当Docker引擎准备好后，用户通过Docker pull command下载最新的consul镜像

在本地运行

要在本地运行consul，从consul的官方网站下载

## 2.3核心数据(重要)

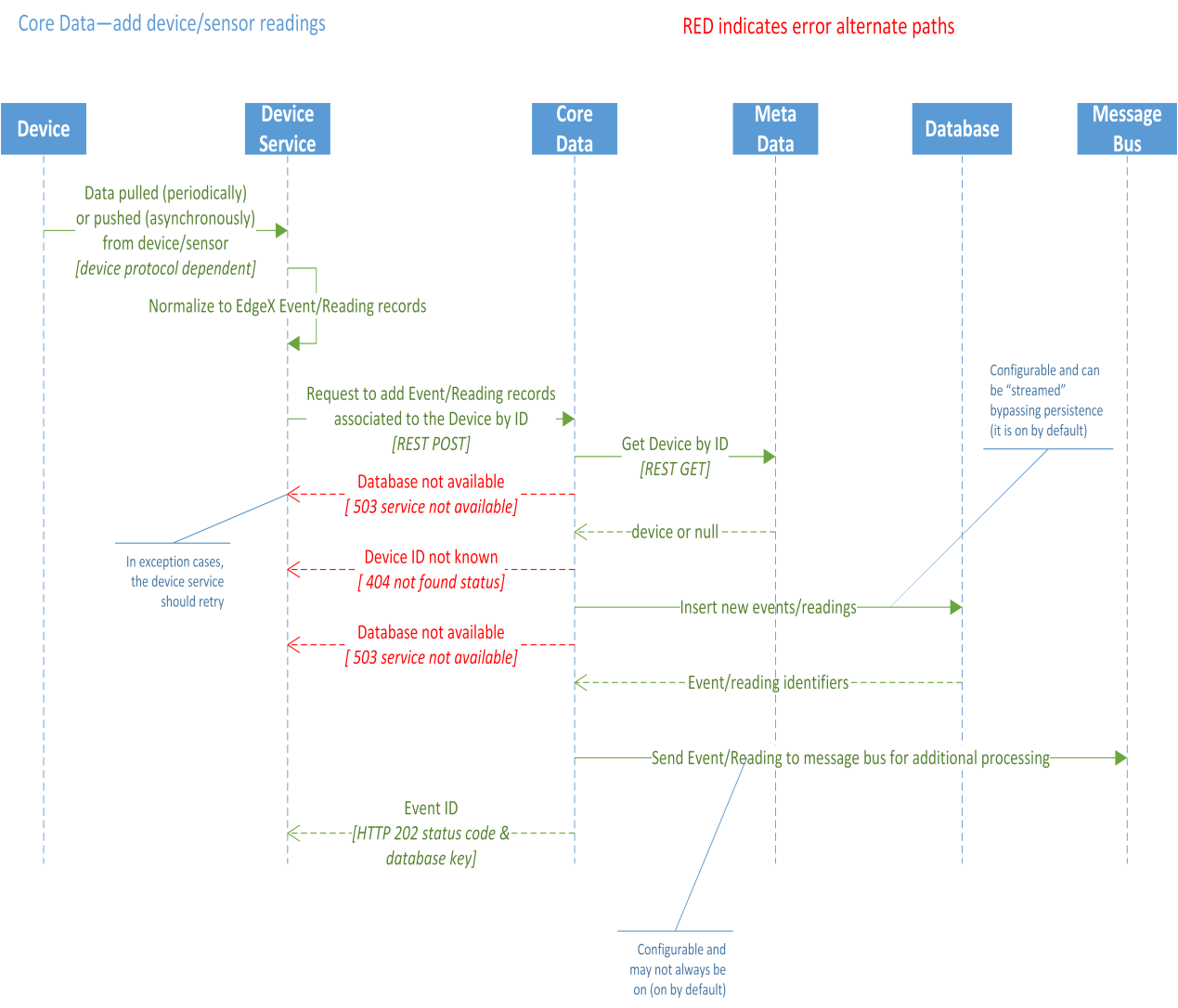
核心数据微服务为读取从设备和传感器收集来的数据提供了一个集中的持久性工具。核心数据微服务收集设备和传感器上的数据，将数据存储在边缘系统(比如网关)上，直到数据可以“向北”移动，然后导出到企业和云系统。

其他服务，例如在EdgeX内部和EdgeX外部的调度服务，只能通过核心数据服务访问存储在网关上的设备和传感器数据。

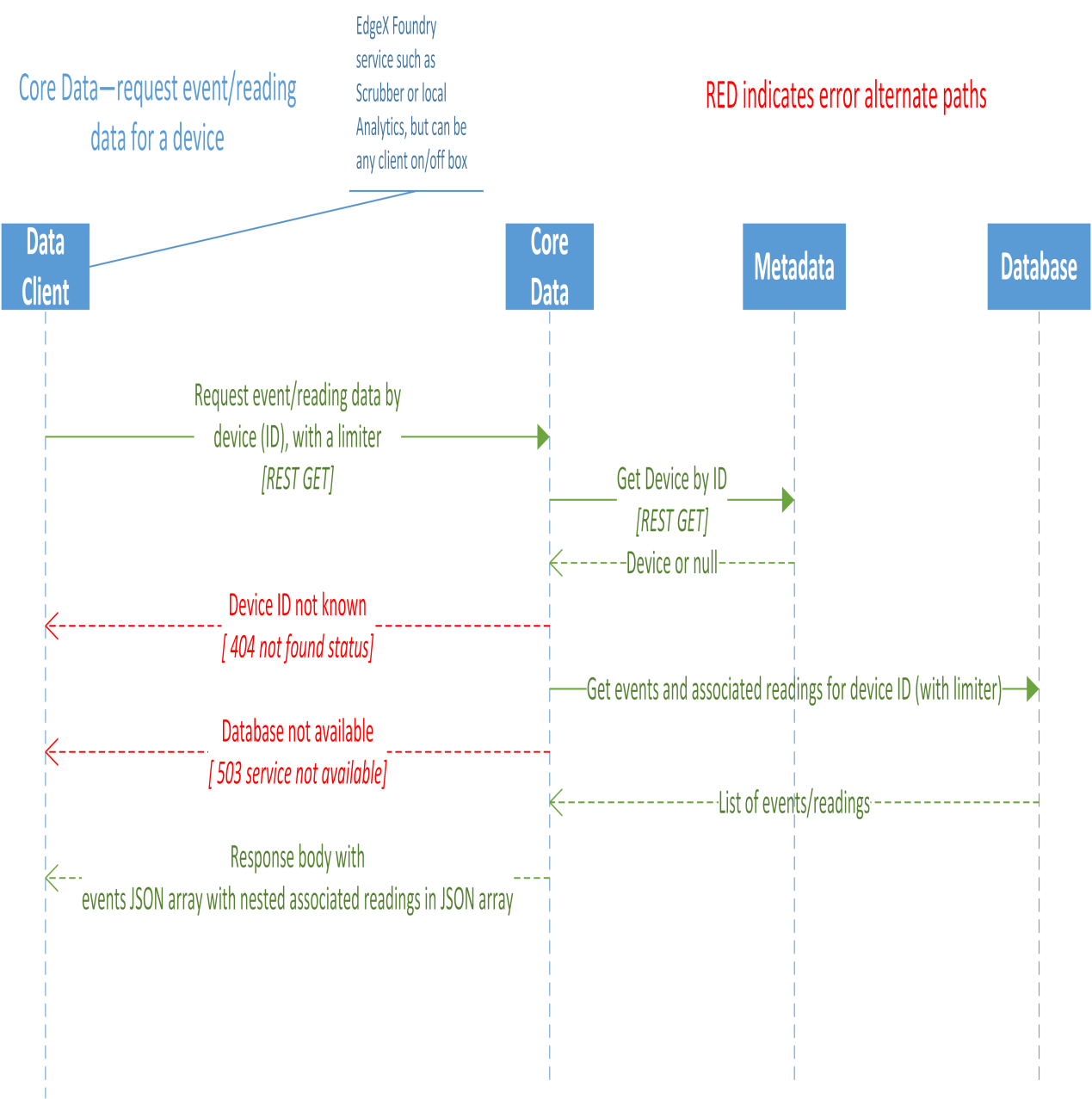
**Core Data使用一个REST API将数据移动到本地存储中或从本地存储中移出**。将来，微服务可以扩展到允许通过其他协议(如MQTT、AMQP等)访问数据。默认情况下，Core Data通过ZeroMQ将数据移动到导出服务层。Core Data的另一种配置允许数据通过MQTT分发到导出服务，但也需要安装代理（中间件），比如ActiveMQ。规则引擎microservice默认从导出分发微服务接收数据。

高级交互图

**Core Data 添加设备或传感器读数**



核心数据请求事件读取或设备数据



## 2.3 元数据（重要）

介绍

元数据微服务了解设备和传感器，以及如何与其他服务(如核心数据、命令等)使用的设备和传感器通信。

具体来说，元数据具有以下功能:

管理与EdgeX Foundry连接和操作的设备和传感器的信息

了解设备和传感器报告的数据类型和组织

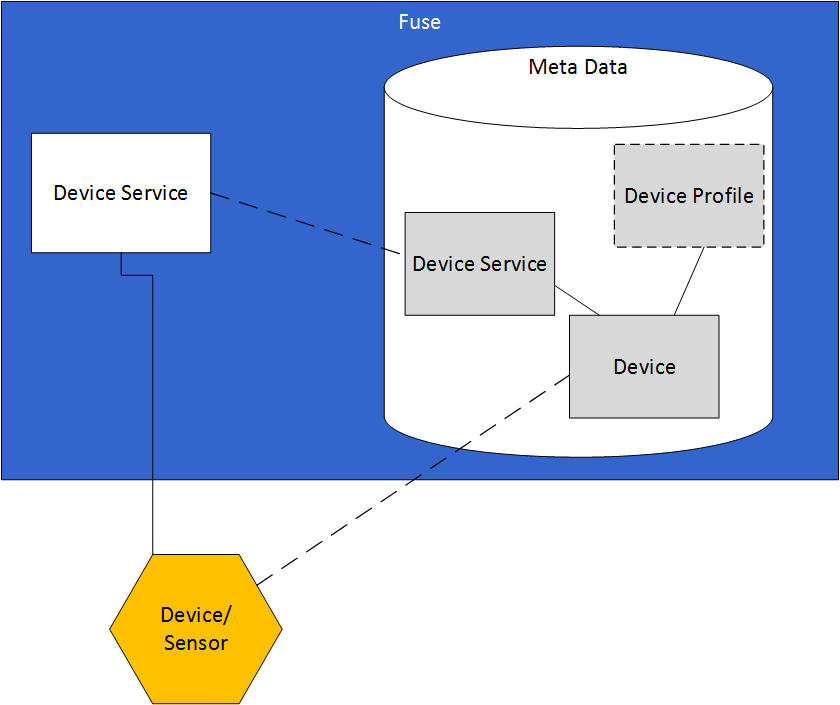
知道如何控制设备和传感器

元数据不做以下活动:

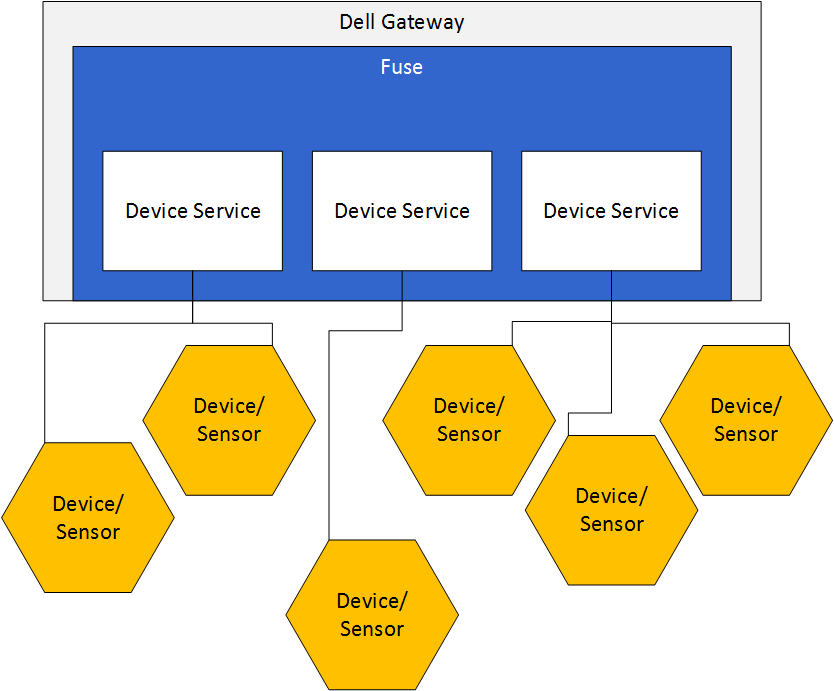
不做，也不负责从设备和传感器收集实际数据，而这些数据是由设备服务和核心数据执行的

不做，也不负责向设备和传感器发出命令，这是由命令和设备服务执行的

设备概要文件是元数据服务必须能够在本地持久性中存储或管理的第一个项目，并提供EdgeX 的其他服务。

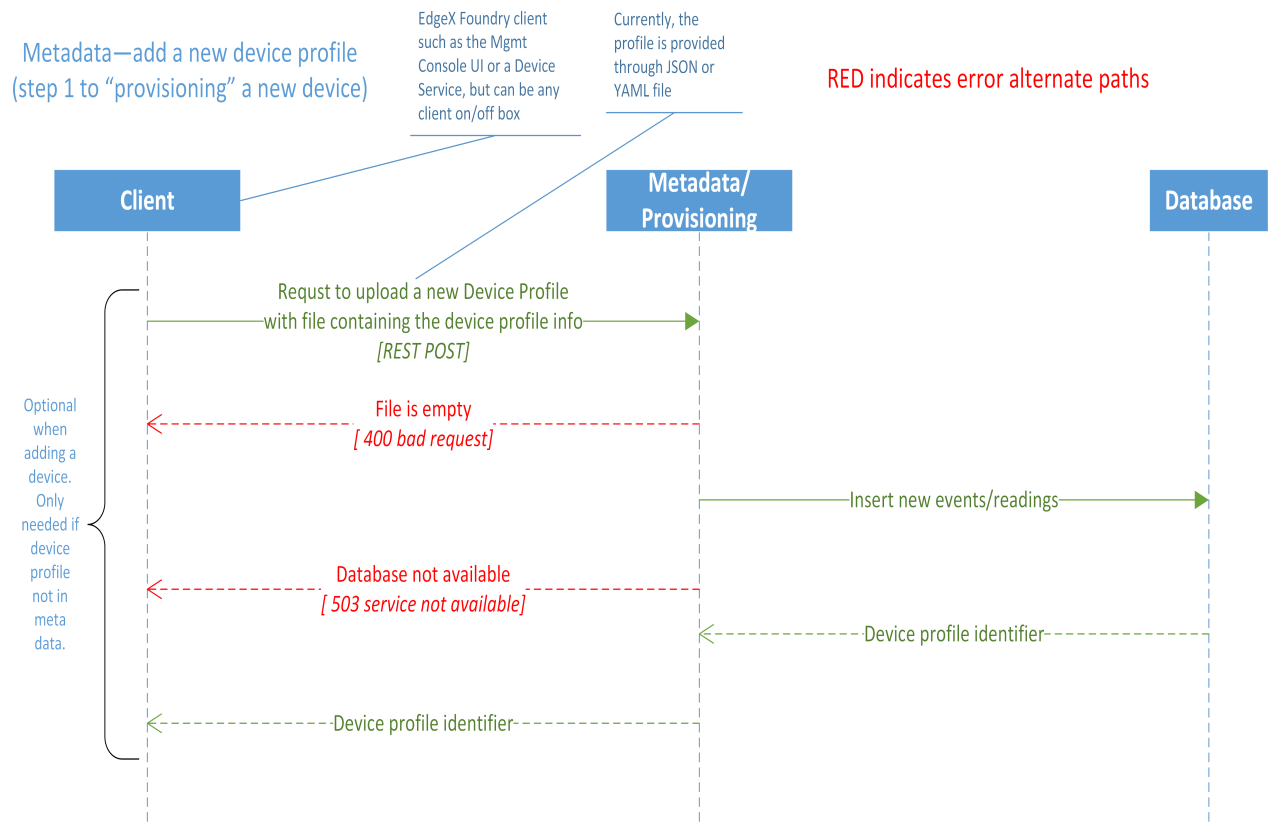


一个单一的设备服务促进了EdgeX Foundry和一个或多个实际设备或传感器之间的通信。通常，设备服务是通过特定协议与使用该协议的设备和传感器通信而构建的。例如，一个Modbus设备服务，它可以促进各种类型的Modbus设备之间的通信，例如电机控制器、近距离传感器、恒温器、电表等等。

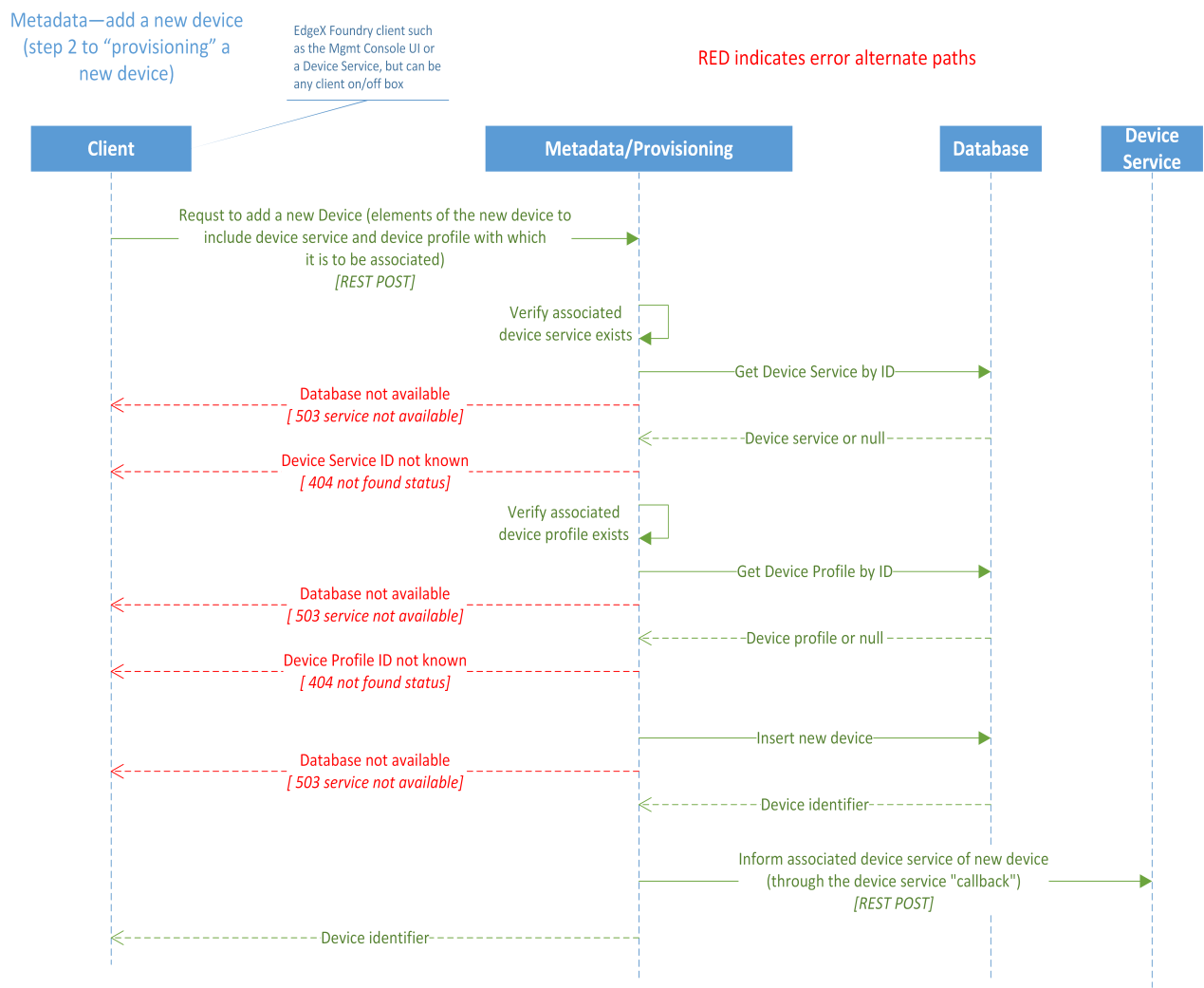


高级交互图

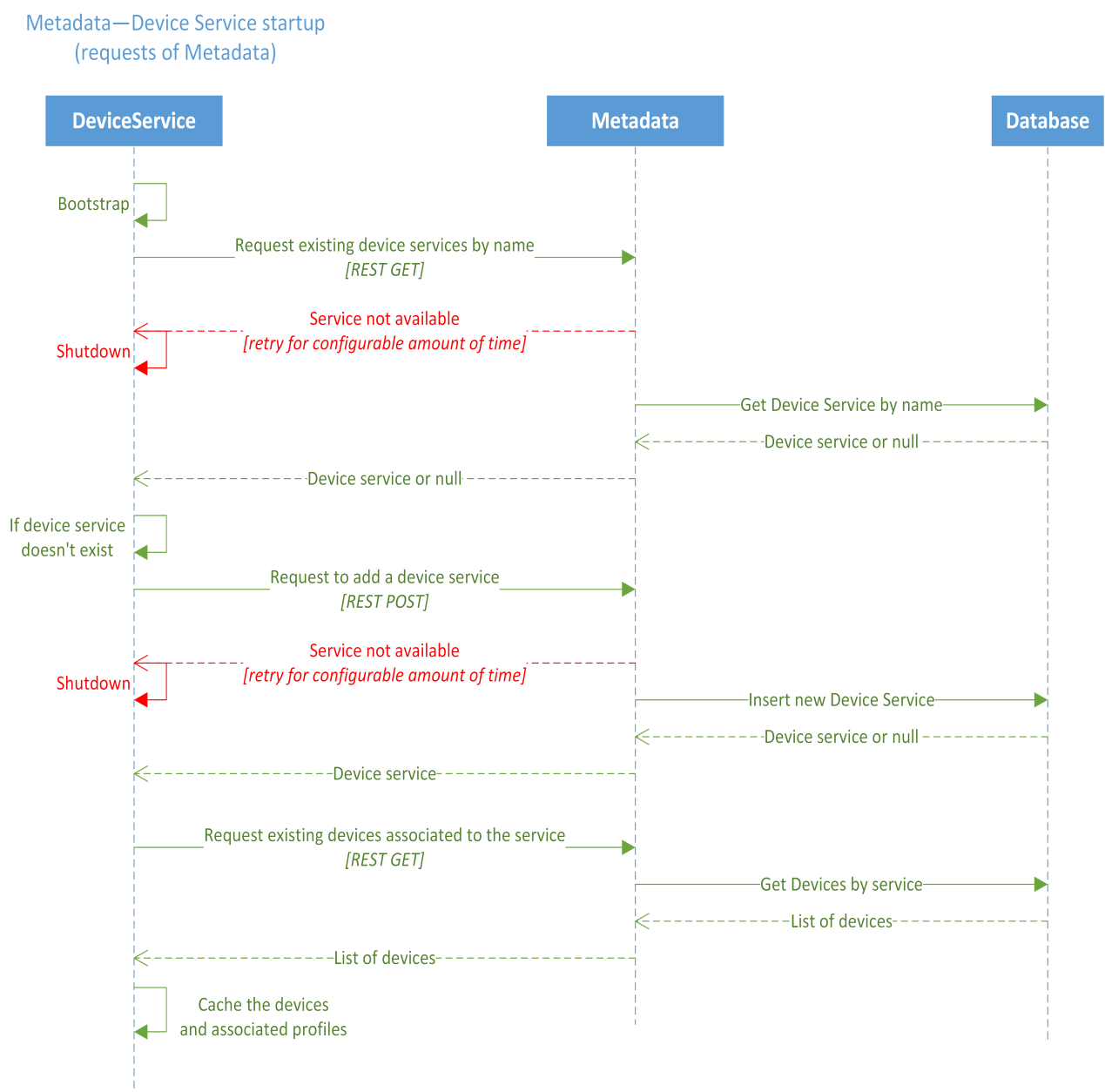
1添加新设备配置文件(配置新设备的第1步)



2添加新设备配置文件(配置新设备的第2步)



3 edgexfoundry元数据设备服务启动



## 2.4 命令（重要）

命令微服务(通常称为命令和控制微服务)允许代表**以下人员**向设备和传感器发出命令或动作:

EdgeX 内的其他微服务（例如，本地边缘分析或规则引擎微服务）

与EdgeX 相同系统上可能存在的其他应用程序(例如，需要关闭传感器的系统管理代理)

对于需要命令这些设备的任何外部系统(例如，基于云的应用程序，它确定需要修改设备集合上的设置)

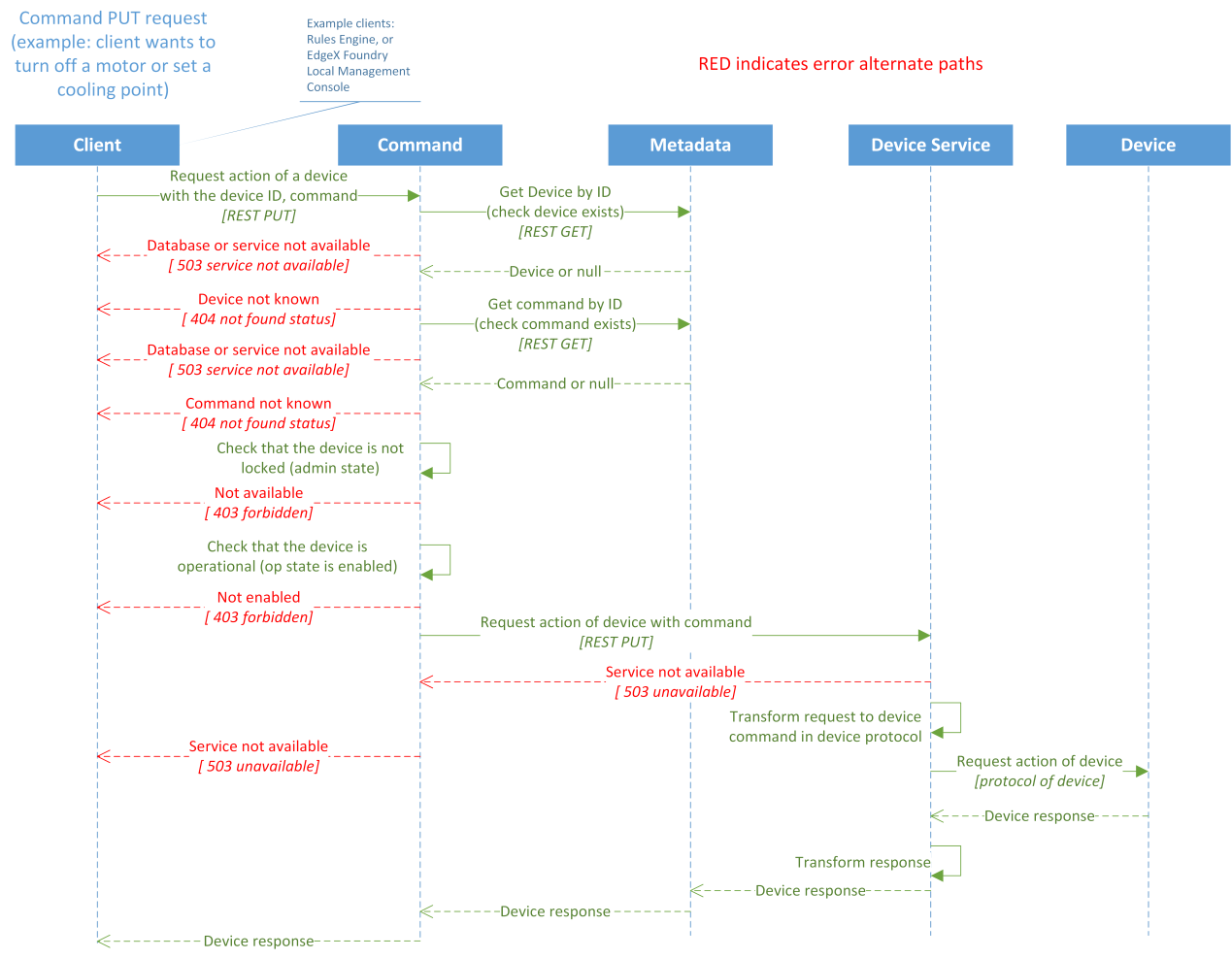
命令微服务以通用的规范化方式公开命令，以简化与设备的通信。包括命令获取、来自设备或传感器的数据请求、命令PUT、请求采取行动或从EdgeX获得新设置或数据。

命令微服务从元数据服务获得有关设备和传感器的信息。命令服务是通过设备服务将命令和动作传递给设备和传感器，而不是直接与设备或传感器通信。因此，命令微服务是来自edgex北侧的命令或动作请求的翻译器，如规则引擎和导出设施、edgex和网关的协议专用设备或传感器和相关设备服务侧。

命令服务在设备和传感器周围提供一层保护，不允许通过设备服务与设备和传感器进行不必要的交互

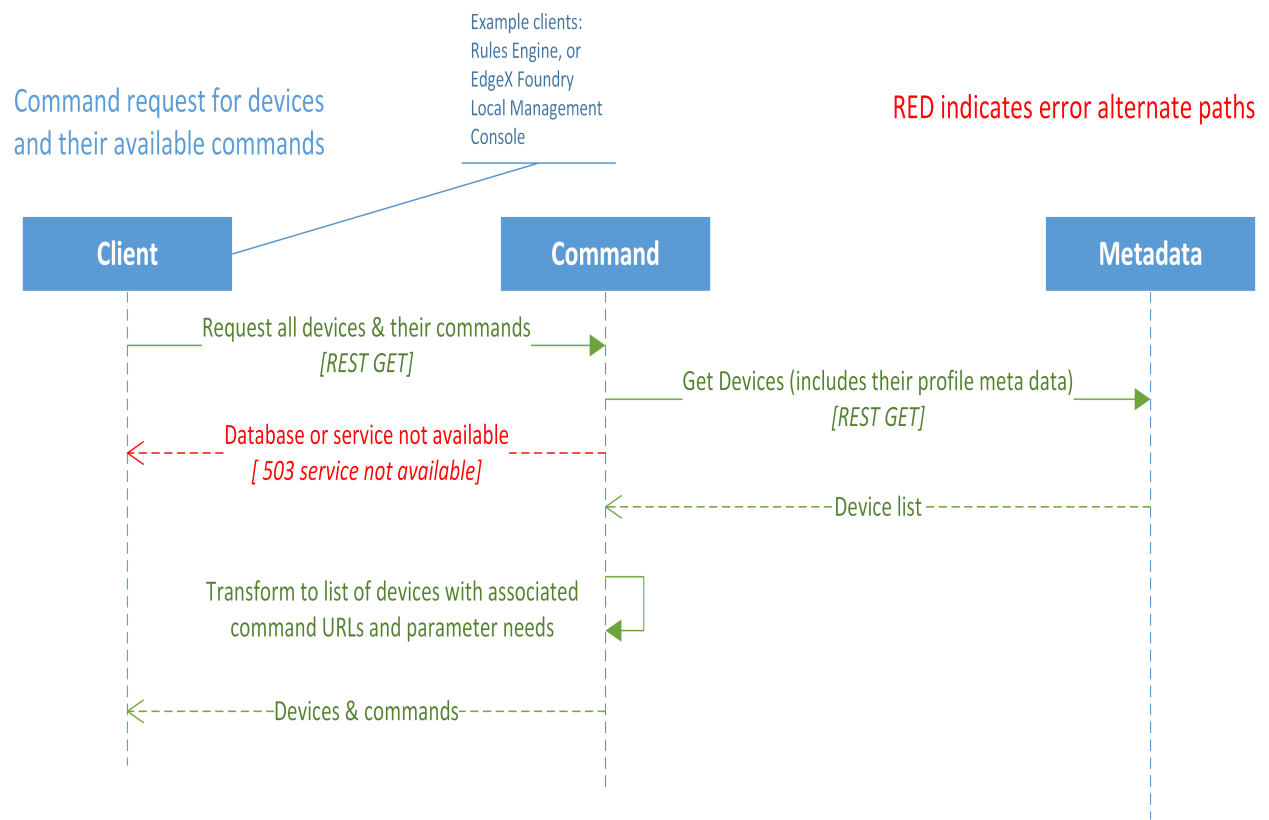
高级交互图

edgex命令PUT请求（同步）



Edgex command 请求设备及其可用命令

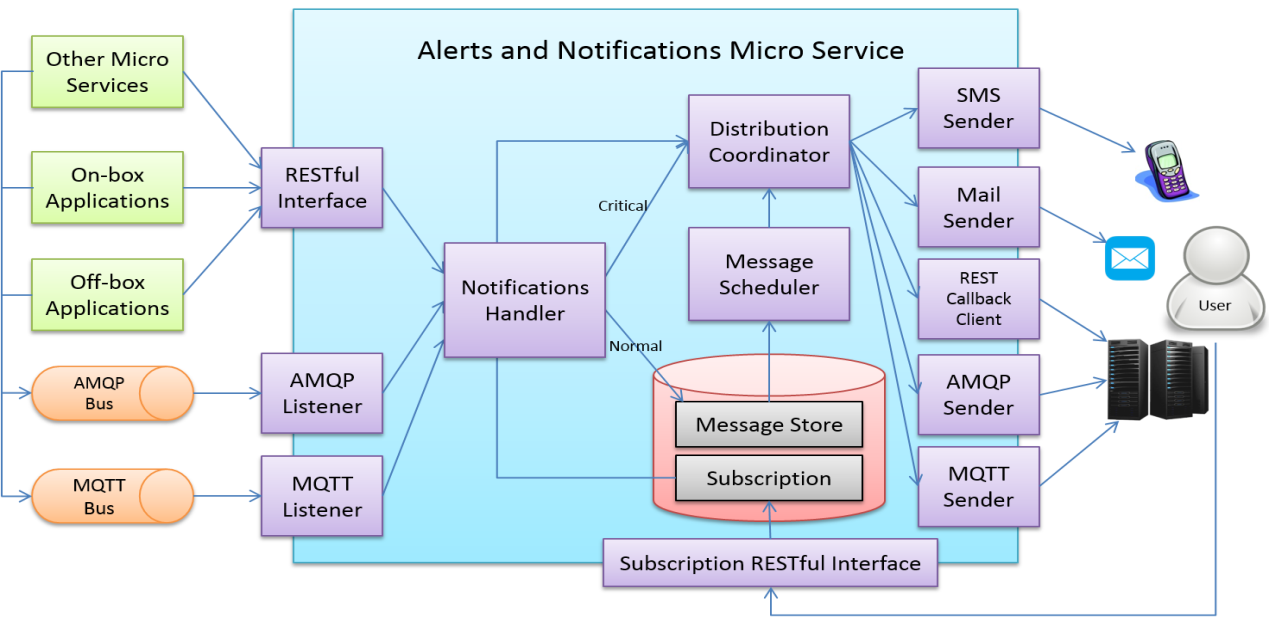
**command提供rest api接口，可以查看设备及其可访问的请求命令及参数**



# 3支持层

## 3.1警报通知

当需要向一个系统或人发出通知，以通知一个微服务在节点上发现了什么东西时，警报和通知微服务会传递该信息。其他服务可能需要广播的警报和通知的例子包括，在某些参数(通常由规则引擎服务检测到)或系统或服务故障(通常由系统管理服务检测到)之外检测到的传感器数据。



该图显示了警报和通知的高级体系结构。

左边为其他微服务、On -box应用程序和off-box应用程序，提供了RESTful接口。

在右侧，通知接收者可以是云中或服务器室内的人或应用程序系统。通过调用订阅RESTful接口来订阅特定类型的通知。当事件发生时，接收者通过定义的接收通道获得适当的通知。接收通道包括SMS消息、电子邮件、REST回调、AMQP、MQTT等。

当警报和通知从任何接口接收通知时，通知将在内部传递给通知处理程序。**通知处理程序首先持久化接收通知**，如果通知是关键的(severity = " critical ")，则立即将它们传递给分发协调器。对于正常通知(severity = " normal ")，它们等待消息调度器批量处理。

警报和通知是可伸缩的，可以扩展以添加更多的服务器，并设置相应的消息调度器来处理它们。例如，正常严重程度通知的消息调度器每两小时触发一次，轻微严重程度通知每24小时触发一次，每天晚上午夜。

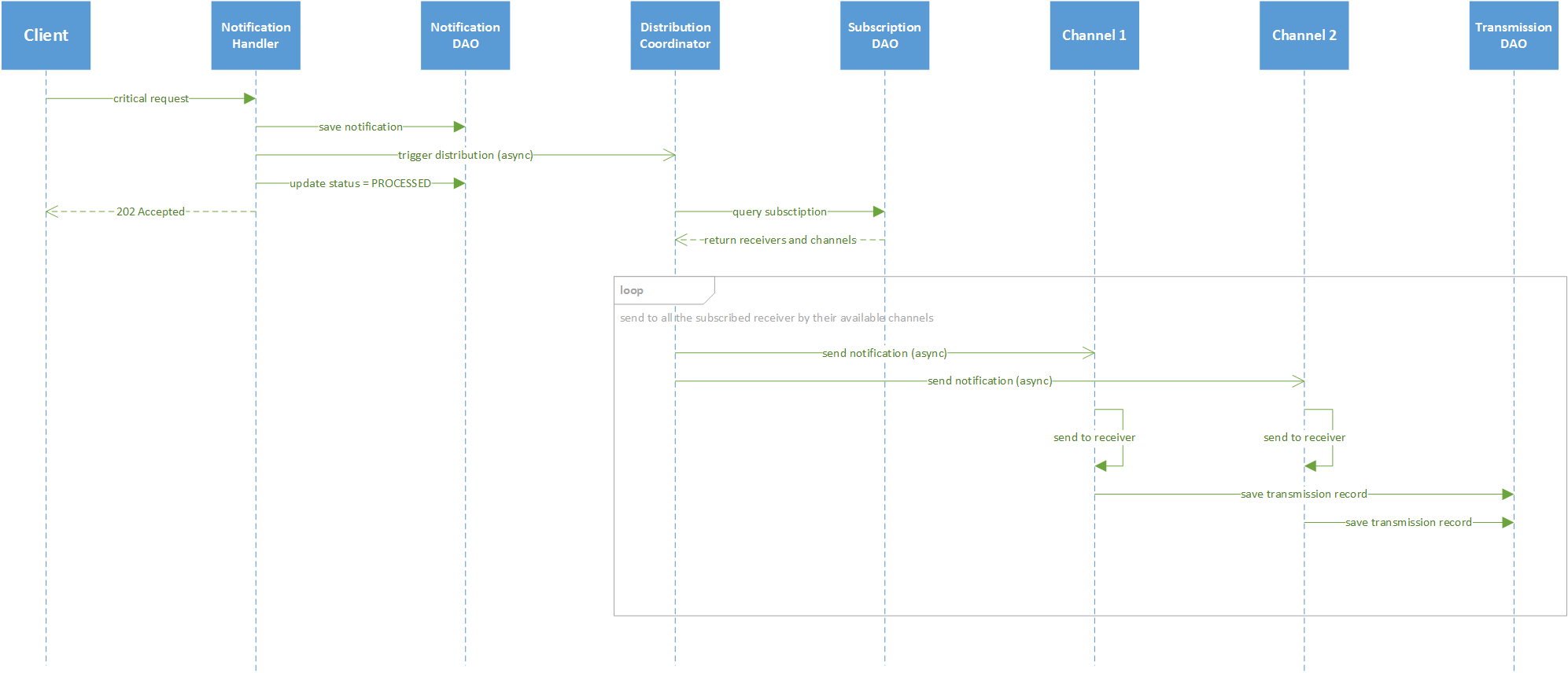
当分发协调器接收到通知时，它首先查询订阅以获取需要获取通知的接收者及其接收通道信息。根据通道信息，分发协调器将此通知传递给相应的通道发送者。然后，通道发送者将通知发送给订阅的接收者。

MongoDB被选择用于警报和通知的持久化，因此数据模型设计没有外键，基于文档结构的范式。

高级交互图

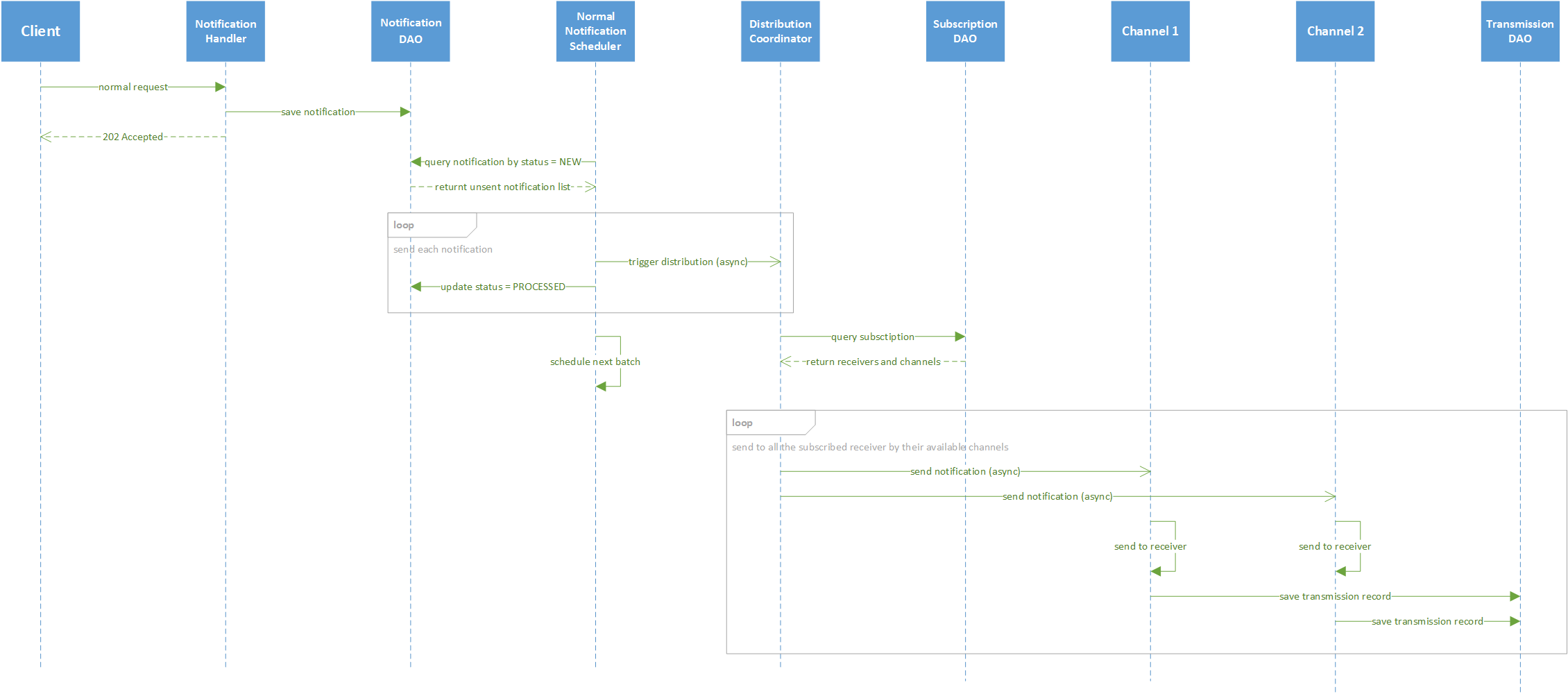
重要通知序列

当接收到关键通知(SEVERITY = " critical ")时，**它首先持久化，并立即触发分发过程**。在更新通知状态之后，警报和通知将响应客户机以指示通知已被接受。



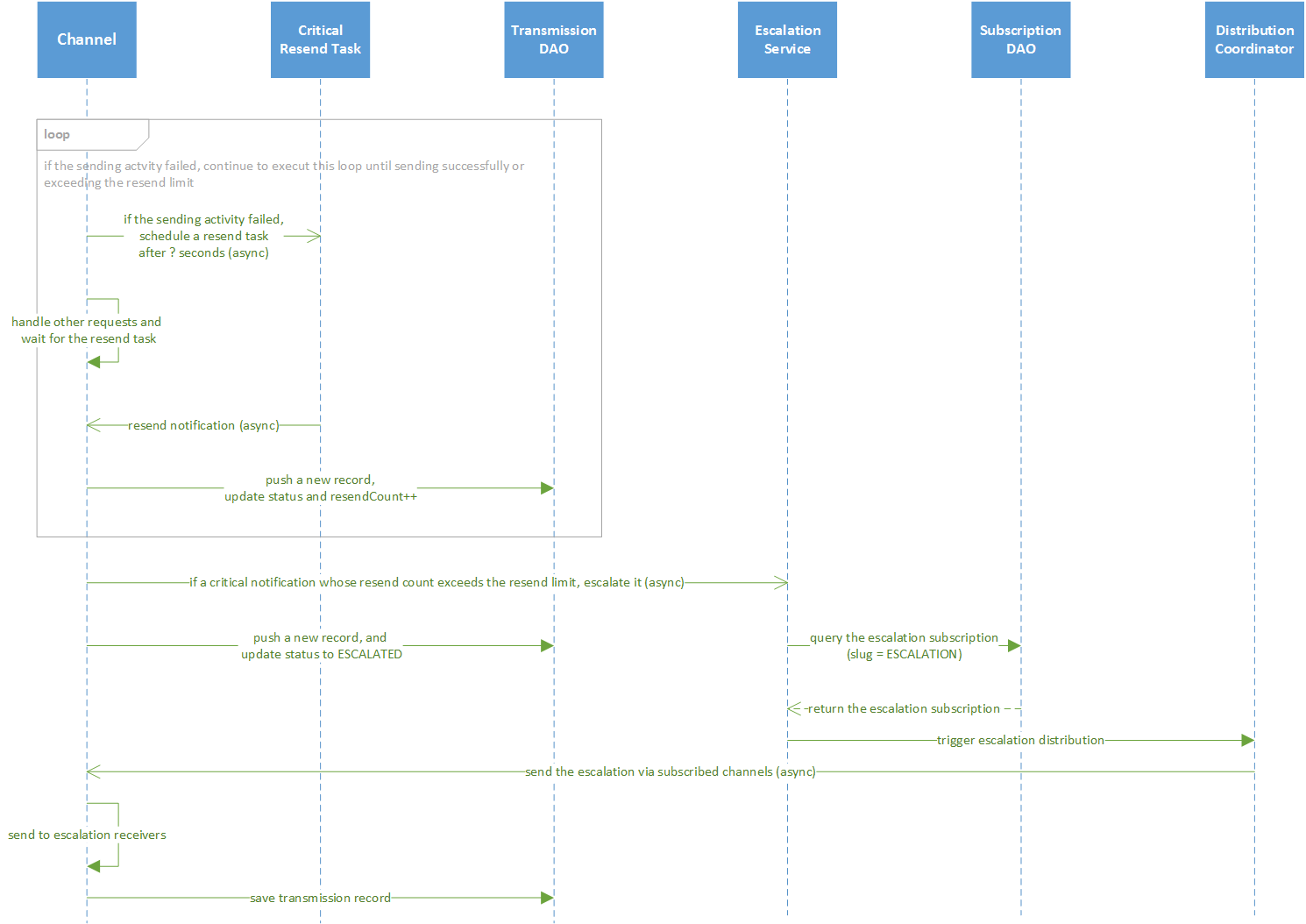
正常通知序列

当接收到正常通知(SEVERITY = " normal ")时，它首先持久化，并响应客户机以指示通知已经被立即接受。在可配置的持续时间之后，调度器将在批处理中触发分发过程。



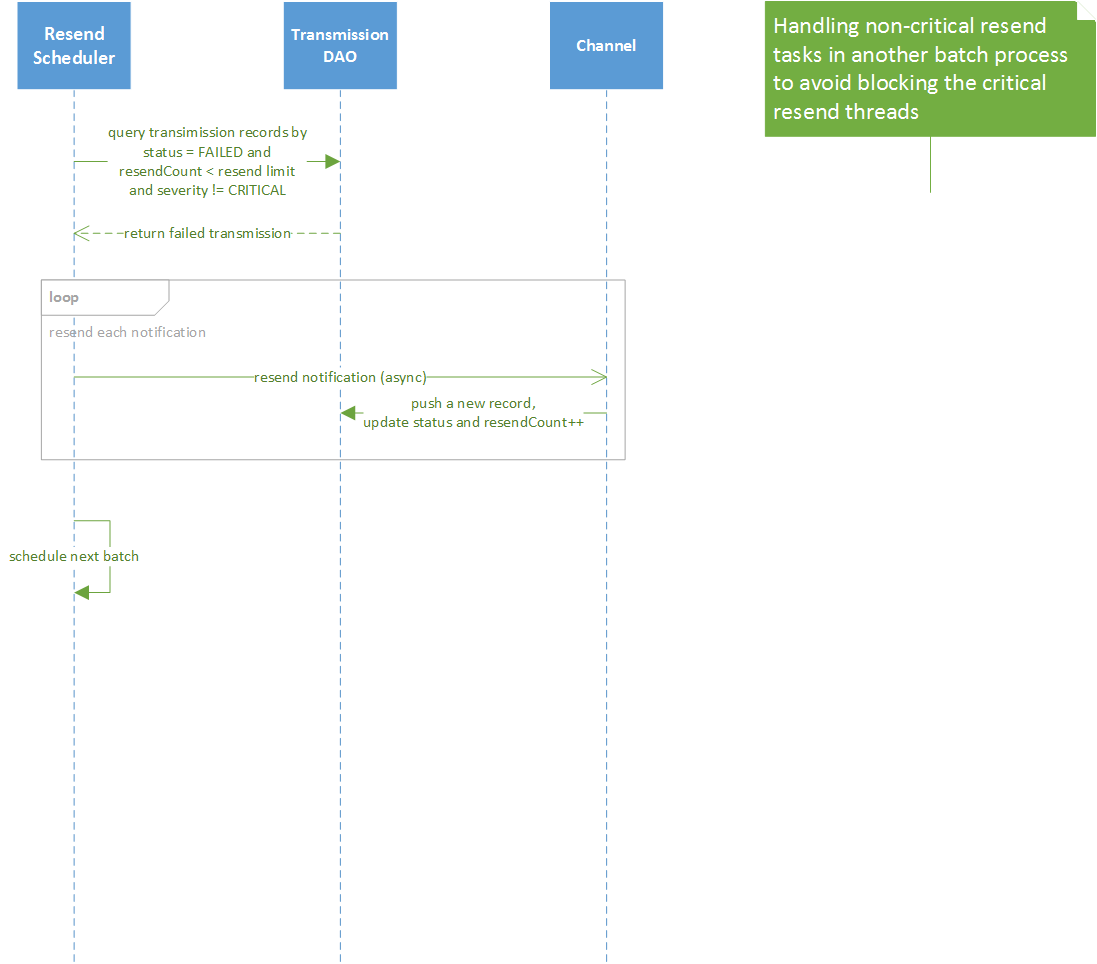
关键重新发送序列

当在发送关键通知时遇到任何错误时，将调度一个单独的重发任务，并保留每个传输记录。如果重发任务持续失败，重发计数超过可配置的限制，则触发升级过程。升级通知被发送到特定订阅(slug =“升级”)的特定接收方。



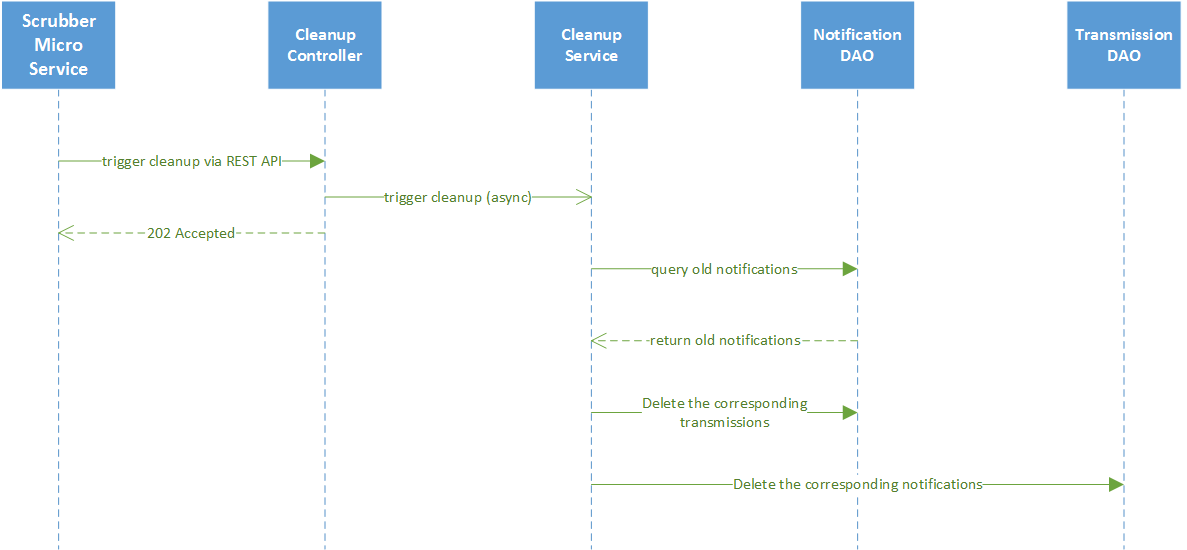
重新发送序列

对于其他非关键通知，重发操作由调度程序触发。



清理序列

清除服务删除旧的通知和传输记录



## 3.2 日志（了解）

日志记录对于所有现代软件应用程序来说都是至关重要的。正确的日志记录为用户提供了以下好处:

能够监控和理解系统在做什么

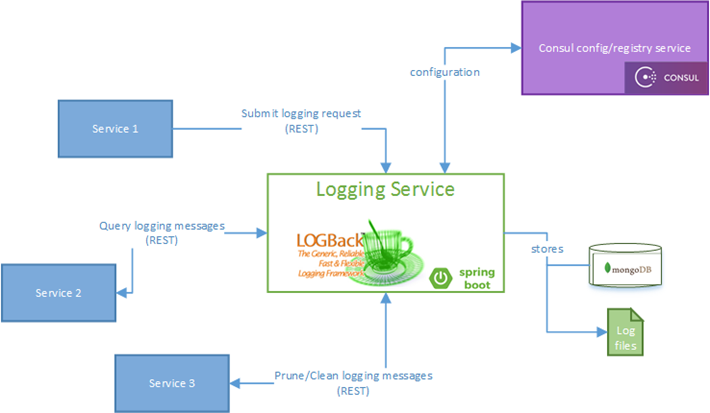
能够理解服务之间如何交互

问题被迅速发现并修复

性能得到了改进

为其他微服务提供RESTful API

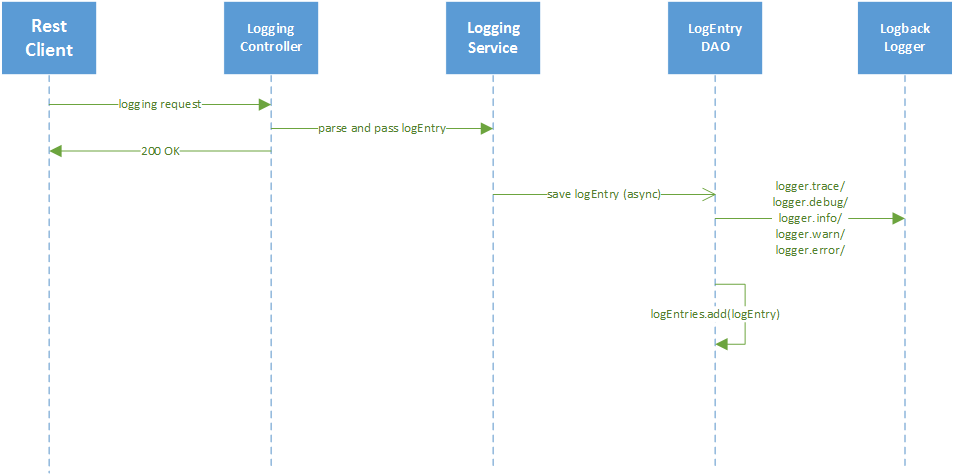
高级架构图



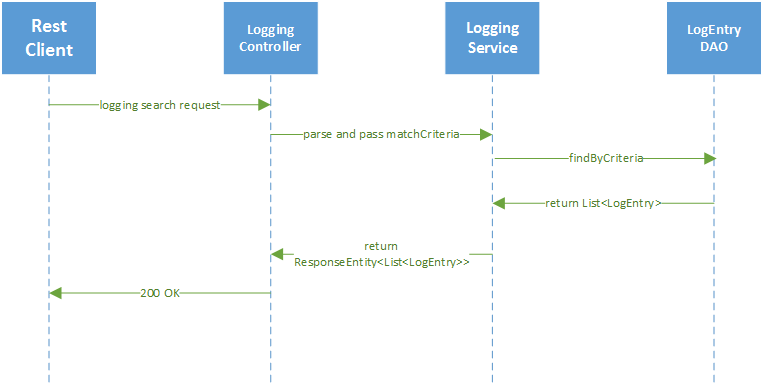
它使用Spring Boot Application框架。其他微服务通过RESTful api与EdgeX Foundry日志服务进行交互，以提交日志请求、查询历史日志记录，并删除历史记录。在内部，EdgeX Foundry日志服务利用LOGBack作为其底层日志框架。EdgeX Foundry日志服务支持两个可配置的持久性选项：文件或MongoDB。

高级交互图

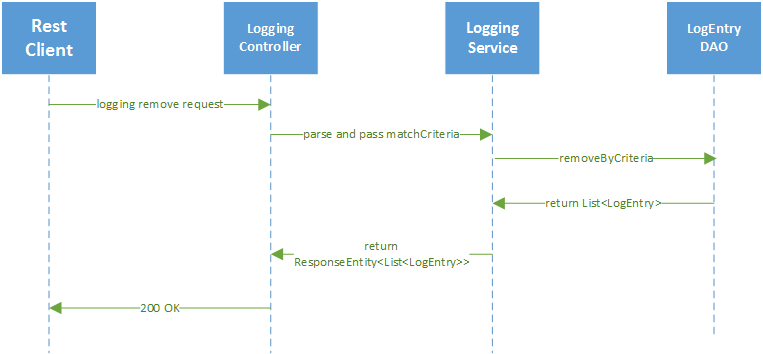
日志记录请求的序列图（logingservice底层是logback）



查询历史日志的序列图

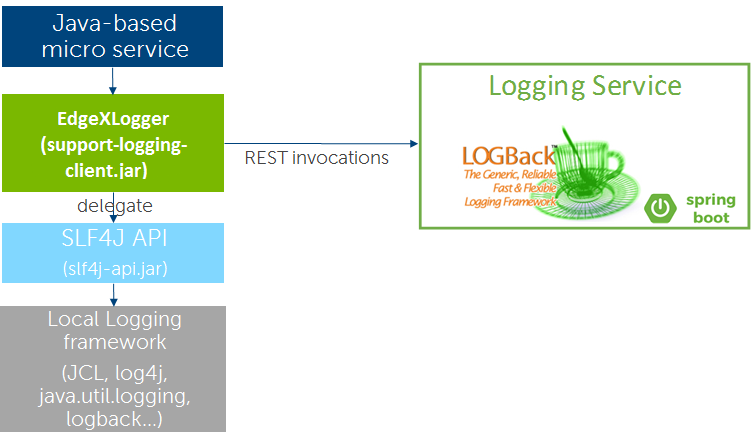


删除历史日志的序列图



用于java日志服务客户端库

由于大多数EdgeX铸造微服务都是在Java中实现的，所以我们为Java提供了一个客户端库，这样基于Java的微服务就可以直接切换它们的记录器来使用EdgeX铸造日志服务。下一个图形显示了Java客户机库的高级设计体系结构。



## 3.3 调度

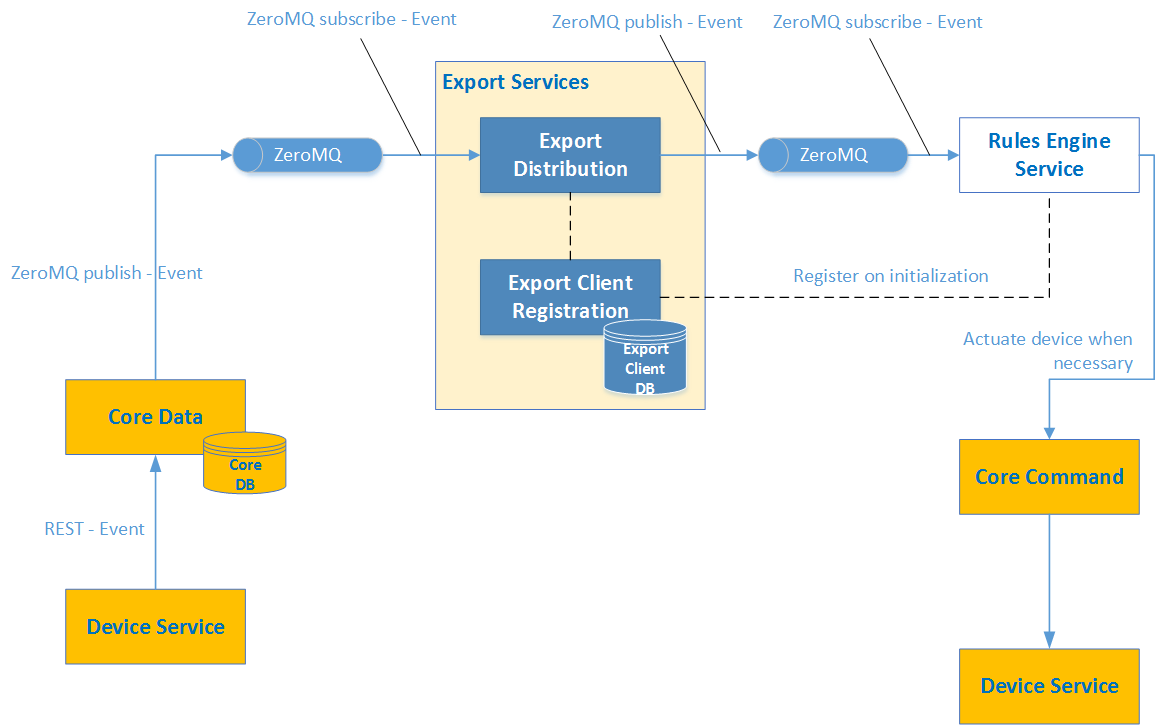
调度微服务包括清除事件和读取已经导出到网关的数据(对于核心数据)的擦洗器微服务。另外，还可以配置支持层微服务来删除未导出的陈旧事件/读取数据。删除陈旧的事件/读取数据使网关能够继续以静态存储量运行，而在出口设备不能运行或运行速度不足以跟上数据收集速度的情况下，系统继续从设备和传感器收集新数据。

将导出的记录和陈旧的记录删除在一个可配置的时间表上。默认情况下，洗涤器每30分钟清理一次数据。

洗涤器微服务并没有直接从EdgeX Foundry的持久存储中删除数据，而是调用Core Data来删除记录。Core Data用作访问持久事件/读取数据的单点。洗涤器微服务是一个独立的服务，没有任何客户端。也就是说，没有调用洗涤器的API。洗涤器按时间触发器工作。

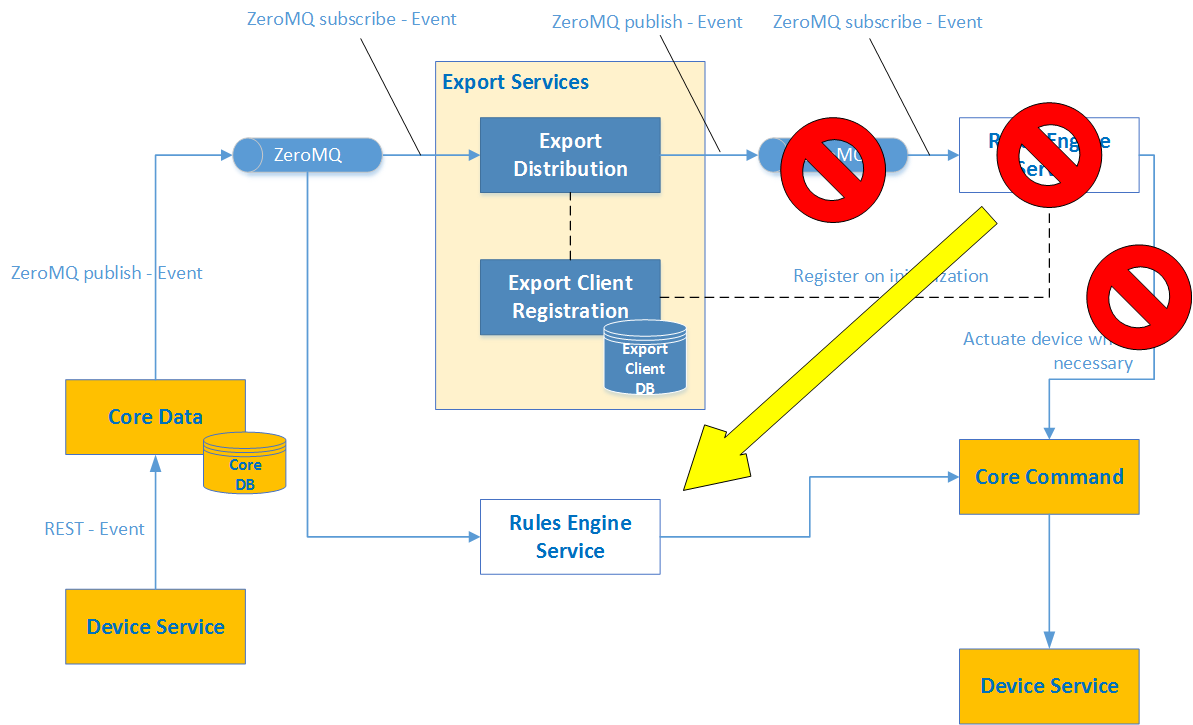
## 3.4 规则引擎

规则引擎微服务为edge事件触发机制提供了一个参考实现。规则引擎服务监视传入传感器或设备数据的读数在目标范围内，并触发即时设备驱动。因此，规则引擎为更快的响应时间提供网络边缘处或附近的“智能”。 规则引擎作为导出服务客户端，参考实现规则引擎是一个自动导出服务客户机。当服务启动时，它会自动调用导出客户端注册微服务来注册成为所有设备的客户端，以及来自核心数据的传感器读数。作为导出服务客户机，参考实现规则引擎通过导出分发微服务接收所有事件和读数。根据数据，指示参考实现规则引擎监视通过导出分发微服务接收到的每个事件和读取，规则引擎通过核心命令微服务触发对设备的任何动作(随后通过设备服务与实际设备通信)。



规则引擎直接连接到核心数据

在更多时间敏感的使用案例或环境中，如果连接的事物生成了大量的数据，可以将规则引擎微服务连接到直接从核心数据中获得的数据。也就是说，通过导出服务来激活基于规则的命令。

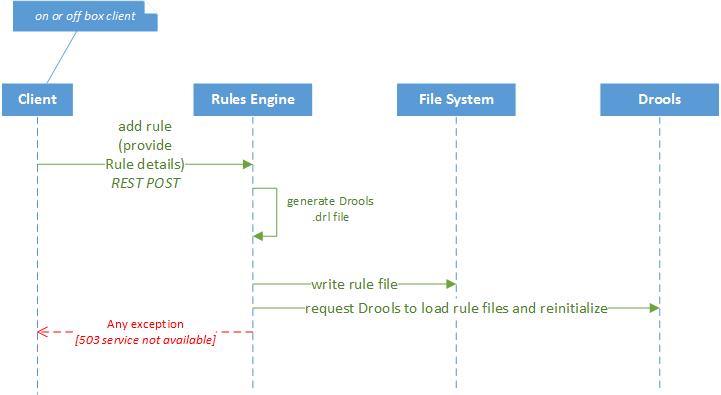


规则引擎已经为这个选项编写了程序。默认情况下，规则引擎微服务将自己注册为导出服务的客户端。这个自动注册可以关闭，并且规则引擎可以直接连接到核心数据之外的ZeroMQ发布数据。注意:由于核心数据的ZeroMQ管道是一种发布-订阅机制，它允许多个订阅者。当规则引擎作为订阅服务器连接时，Core Data实际上同时发布到两个客户机或订阅服务器:导出服务和规则引擎。为了断开规则引擎与导出服务(作为客户端)的连接，并将其直接连接到核心数据，必须更改相应规则引擎微服务配置参数(可在application.properties中找到)。

规则客户端和高级交互图

规则引擎微服务附带了一个RESTful服务，它允许添加和删除新的规则。RESTful API允许动态地将JSON中定义的新规则添加到规则引擎中(通过REST POST)。提供的JSON数据被转换为Drools规则文件(.drl文件)由microservice提供。每个规则都必须与用于标识规则和保存规则的Drool文件的惟一名称相关联。还可以通过名称要求删除规则。

注意：由于Drools中存在的问题，删除的规则只会清空内容。规则的名称(以及表示规则的文件)仍然在系统中。因此，在停止规则引擎微服务和删除空的Drool文件之前，不能重用规则名。



# 4 导出层

## 4.1客户端注册

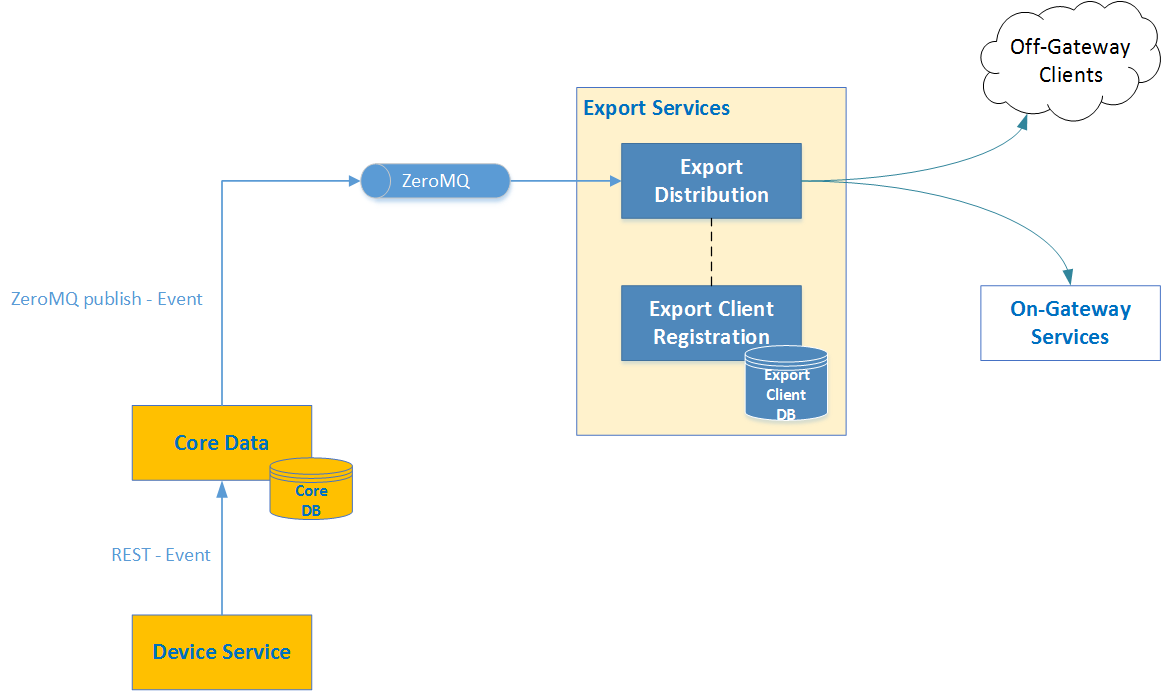
EdgeXFoundry导出客户端注册微服务允许客户端(无论它们是在网关上还是在网关外)注册为来自核心数据的数据接收者。

数据由传感器和设备发送到网络边缘，进入EdgeX。然后，数据通过EdgeX的设备服务层传输到核心服务层的核心数据，然后数据通过消息队列传送到导出设施。

导出设施包括两项微型服务:

一个微服务，使客户能够注册他们感兴趣的数据

一个微服务，可以根据需要过滤和转换数据，然后将数据物理地获取到注册客户端，称为Export发行版微服务



**网关上的客户端**，包括本地分析服务、关键事件处理器、规则引擎服务等等。这些本地客户端通常需要通过**命令微服务**根据所检查的数据返回传感器/设备，因为这些数据是通过导出设施接收的。

**非网关客户端**通常是基于企业和云的系统或聚合网关系统。这些系统通常是**数据的历史存储**库，也可能提供更深入的分析功能。

客户机注册微服务提供了一个REST API，使客户机能够建立新的请求、更新现有请求和删除对核心数据的请求。虽然没有与此服务直接关联的用户界面，但是UI控制台和其他管理系统可以使用提供的api提供这种功能。

客户注册详情

客户机注册服务允许客户机提供关于“什么”、“如何”和“在哪里”需要数据的一些详细信息。

1“什么”数据

“**什么”指定数据的过滤器:过滤并只向客户端发送感兴趣的数据**。默认情况下，通过核心数据的网关收集的所有数据都被发送到每个客户端。然而，今天的客户可以建立两个过滤器。

**根据设备ID或设备名称进行筛选**。通过客户端注册指定的设备id或设备名称集合，客户端只能从这些设备和传感器获取数据。

**根据值描述符ID或值描述符名称进行过滤**。值描述符描述数据的类型和内容。例如，一个名为“温度”的值描述符将描述温控器或其他温度读数数据。使用客户端注册指定的值描述符id或名称的集合使客户端能够只获取带有这些值描述符标记的数据。

当注册没有提供过滤器时，所有数据都被转发给客户端。提供注册的一个或两个过滤器，将发送到客户机的数据限制为与设备初始化或数据的值描述符标记匹配的数据。

2如何格式化

数据如何传递给客户端还取决于客户端注册。客户机可以请求以特定格式发送数据，对数据进行加密、压缩或加密和压缩。

支持格式- JSON和XML选项

加密-不能指定加密或AES

压缩-没有压缩、GZIP或ZIP选项可用

3“在哪里”交货

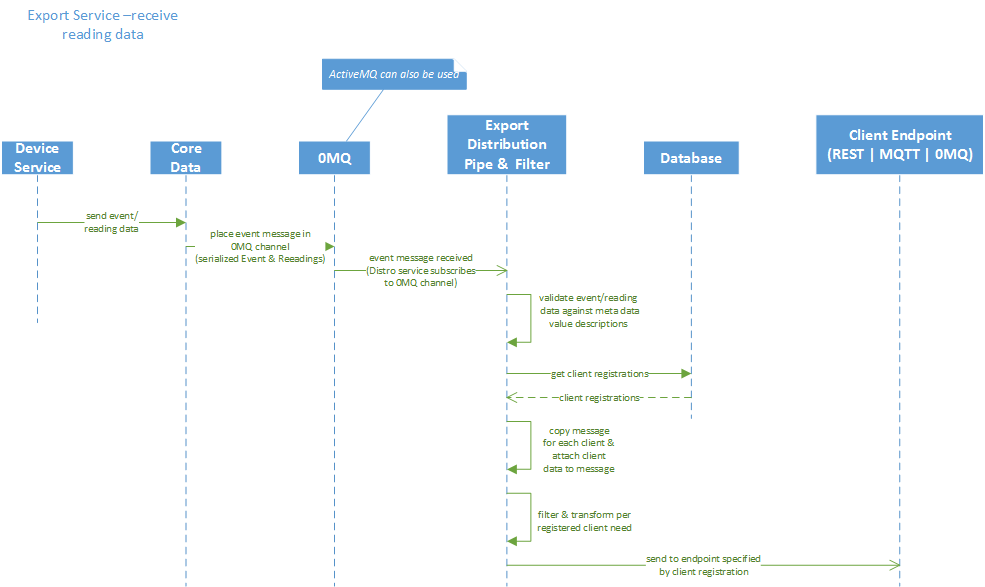
EdgeX客户端可以请求将数据推送到他们选择的端点。注册时需要提供端点(URL、凭证等)的详细信息。**EdgeX Foundry支持两种类型的目的地:推送到REST端点，或者发布到指定的MQTT代理主题。**

REST端点交付必须提供客户端注册。

## 4.2分发

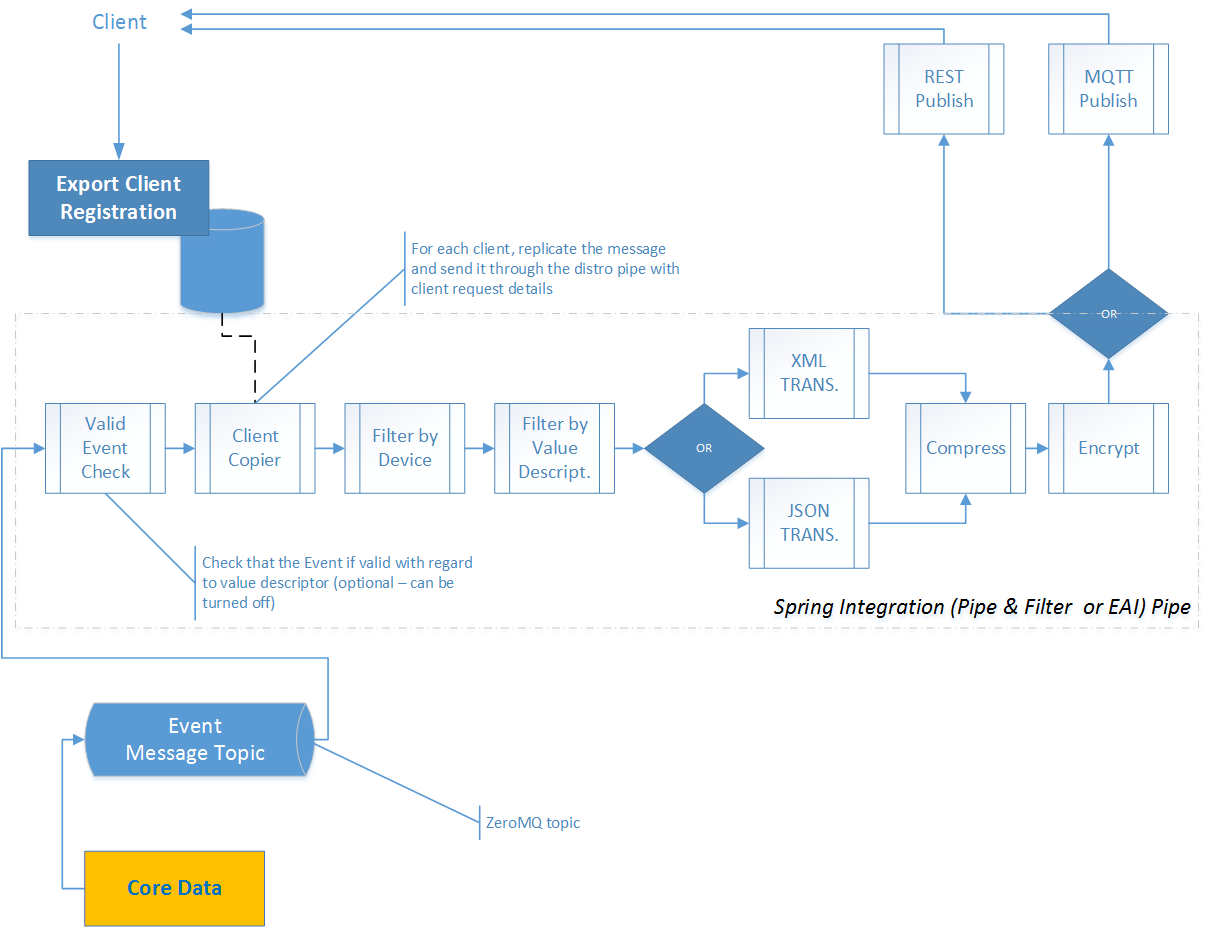
通过消息队列接收来自核心数据的数据，然后过滤、转换和格式化每个客户机请求的数据，然后通过REST、MQTT或ZeroMQ将其分发到选择的客户机端点。导出分发服务是基于企业应用程序集成模式的管道和过滤器体系结构。数据通过消息队列到达管道的前端。默认情况下，核心数据异步地将每个设备或传感器事件(带有读数)**推入**ZeroMQ。作为替代配置，Core Data可以将每个设备或传感器事件与读数一起发布到网关ActiveMQ代理中。导出发行版服务侦听传入事件。当接收到一个事件时，该事件被视为一条消息，并通过导出发行版管道和过滤器发送到注册客户端。

高级交互图



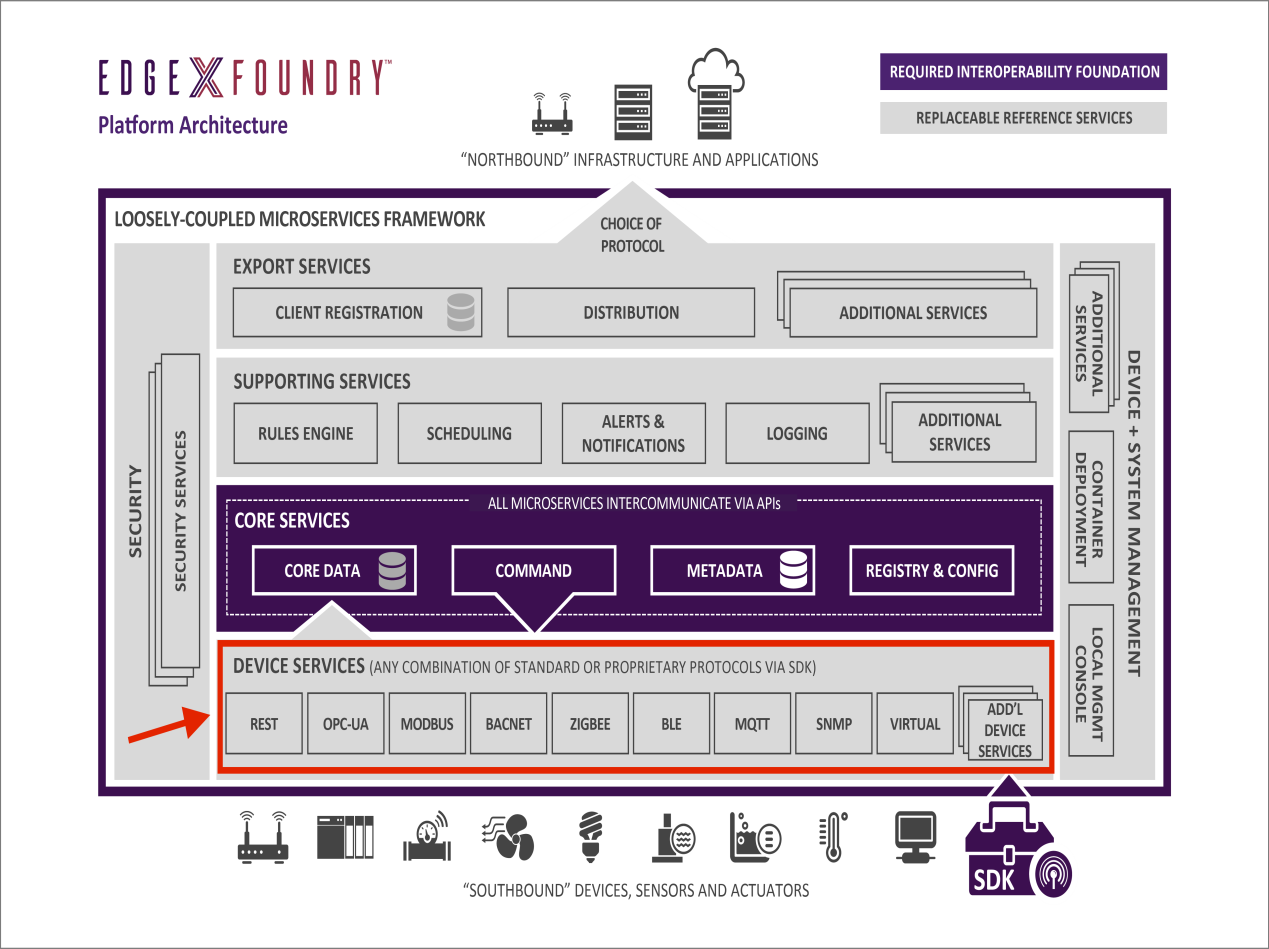
管道和过滤器

在导出分发服务中，一个Spring集成组件集，根据每个客户端注册的首选项过滤和转换事件数据。此图显示了分布服务管道中包含的组件和操作的列表。



检查事件是否对值描述符有效

对于每个客户端，复制消息并根据客户请求细节通过分发管道发送消息



设备服务层与设备服务交互。

设备服务(DS)是与设备或物联网对象交互的边缘连接器，这些设备或物联网对象包括但不限于:家用电器、报警系统、HVAC设备、照明、任何行业的机器、灌溉系统、无人机、交通信号、自动化运输等等。

设备服务可以同时服务一个或多个设备，包括传感器、执行器等等。DS管理的“设备”可以不是简单的单一物理设备，也可以是另一个网关以及该网关的所有设备、设备管理器或设备聚合器(作为EdgeXFoundry的设备或设备集合)。

设备服务层的微服务通过**原生的物联网对象协议**与设备、传感器、执行器和其他物联网对象进行通信。DS层将由IoT对象生成和通信的数据转换为通用的EdgeXFoundry数据结构，并将转换后的数据发送到核心服务层，以及EdgeXFoundry其他层的其他微服务。

目前，edgexdry设备服务层包括以下微服务:

APIs–Device Services–Virtual Device Service

设备服务需求

下面提供了对设备服务要求。这些要求被用于定义通过任何设备服务SDK提供的功能，以生成设备服务脚手架代码。它们还可以帮助读者理解设备服务的职责和角色。

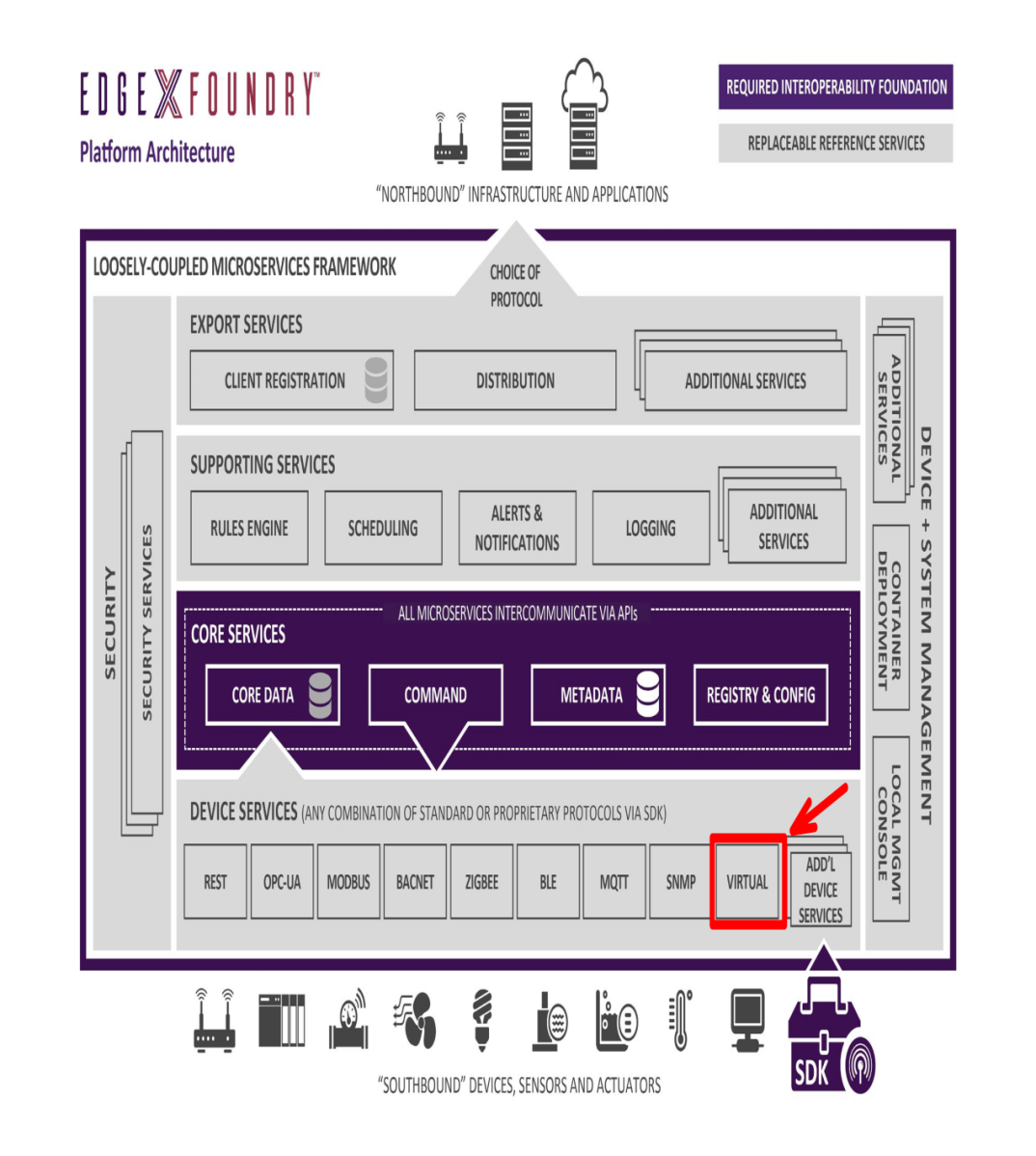
[DS-SDK Requirements](https://wiki.edgexfoundry.org/display/FA/Architecture--Device+Services+Microservices?preview=/328052/4587926/DS-SDK-Requirements-v3.xlsx)

设备服务设计

序列图，概述每个需求的对象和过程。

[Design for Requirements #1-4](https://wiki.edgexfoundry.org/display/FA/Architecture--Device+Services+Microservices?preview=/328052/4588068/DS-SDK-Design-SequenceDiagrams.vsdx)

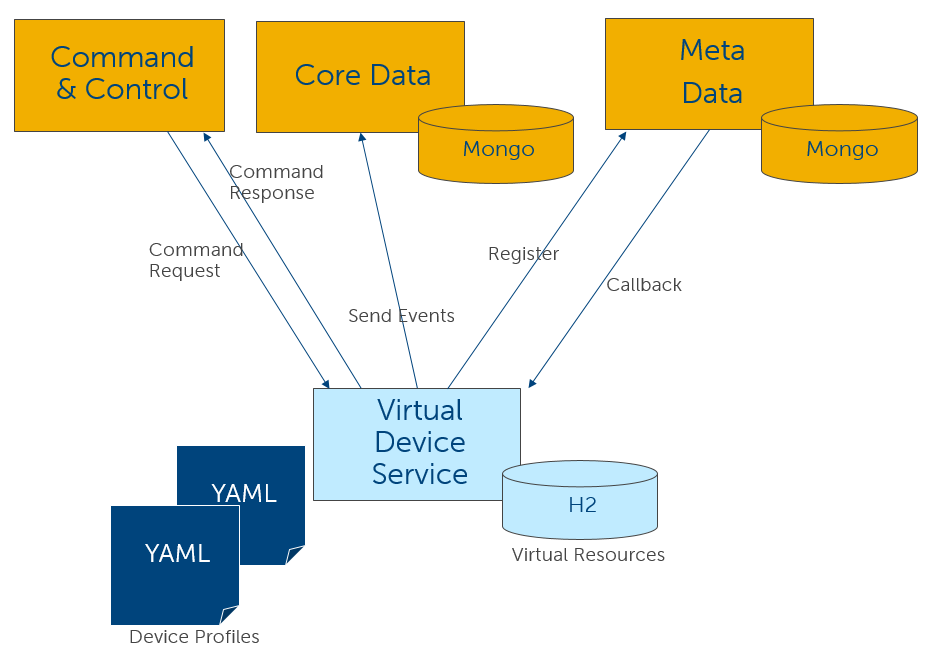
# 5虚拟设备



## 5.1 介绍

虚拟设备服务模拟不同类型的设备，为核心数据微服务生成事件和读数，用户通过命令和控制微服务（command&control）发送命令和获取响应。在执行功能或性能测试时，虚拟设备服务的这些特性在没有任何实际设备的情况下非常有用。

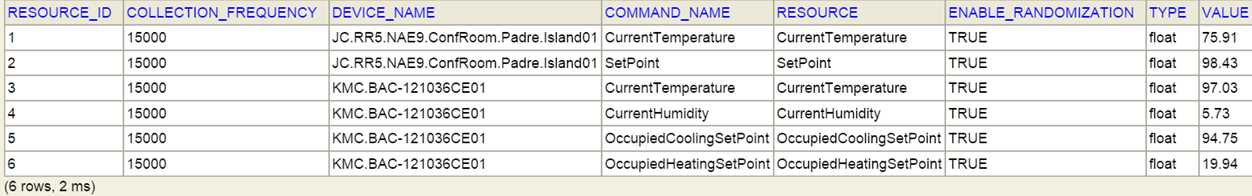
虚拟设备服务总体流程



**虚拟设备服务依赖于核心数据和元数据微服务**，因为初始化过程需要检查或注册配置文件、设备和值描述符。因此，核心数据和元数据微服务必须在虚拟设备服务初始化之前完全启动。在虚拟设备服务初始化过程开始时，流程向核心数据和元数据微服务发送ping请求，以验证它们的状态，直到它们都完全启动为止。固定的时间限制是600秒。600秒后，如果Core Data和Metadata Microservices没有完全启动，初始化过程失败。

当虚拟设备服务启动时，初始化过程从YAML文件中加载ValueDescriptor、DeviceService和DeviceProfile的定义，并在Metadata Microservice中创建它们。一些默认的YAML文件存在，用户可以创建自己的文件。此外，虚拟设备服务为元数据微服务提供回调api来管理设备实例。根据DeviceProfile定义中的GET命令，在虚拟设备中有一个H2数据库(内存中)存储资源，称为“虚拟资源”。

## 5.2 虚拟资源示例



资源id 收集频率 设备名字 命令名字 资源 是否随机 类型 值

虚拟设备服务启动后，虚拟设备服务定期(默认频率是15秒，可以从配置文件中修改)读取H2数据库，并将读取的当前值作为事件event发送到Core Data微服务。(虚拟资源的当前值在每个收集周期之前以随机值重新生成，因此在每个事件中读取的值应该是不同的。)

虚拟设备服务为命令和控制微服务(command&control)提供api，读取H2数据库，并返回GET命令的当前值。此时，虚拟设备服务支持两个特殊的PUT命令:enableRandomization”和“collectionFrequency”。这些命令修改了H2数据库中特定虚拟资源的ENABLE\_RANDOMIZATION和COLLECTION\_FREQUENCY列。

**总结虚拟服务启动的过程：**

**1虚拟设备服务启动时先向core data和core metadata 发送ping命令，看他们是否正常启动**

**2从yaml文件中读取ValueDescriptor、DeviceService和DeviceProfile的定义，并在metadata中创建**

**3服务启动之后，虚拟设备服务定期的从H2数据库中读取虚拟资源，并将最新读取的值最为event发送到core data。**

**5.3特殊的配置**

虚拟设备微服务包含一些独特的配置属性，下面列出了在运行虚拟设备服务之前应该调查的独特属性，以便更好地了解它在您的环境中是如何工作的:  
1）用于定位设备配置文件YAML文件的路径，这个文件用于定义由设备服务管理的虚拟设备。

application.device-profile-paths=./bacnet\_sample\_profiles,./modbus\_sample\_profiles

2）指向上述目录中YAML配置文件的配置设备的虚拟设备服务指示符，当设置为true时自动为每个配置文件创建一个设备。

application.auto-create-device=true

3）在开发过程中，每当服务启动时，让虚拟设备服务重新启动，并使用一个干净的Metadata数据库，通常是有利的。下面的属性指示设备服务在关闭时是否应该清除数据库中的任何现有虚拟设备，以便在服务启动备份时提供干净的环境。

application.auto-cleanup=true

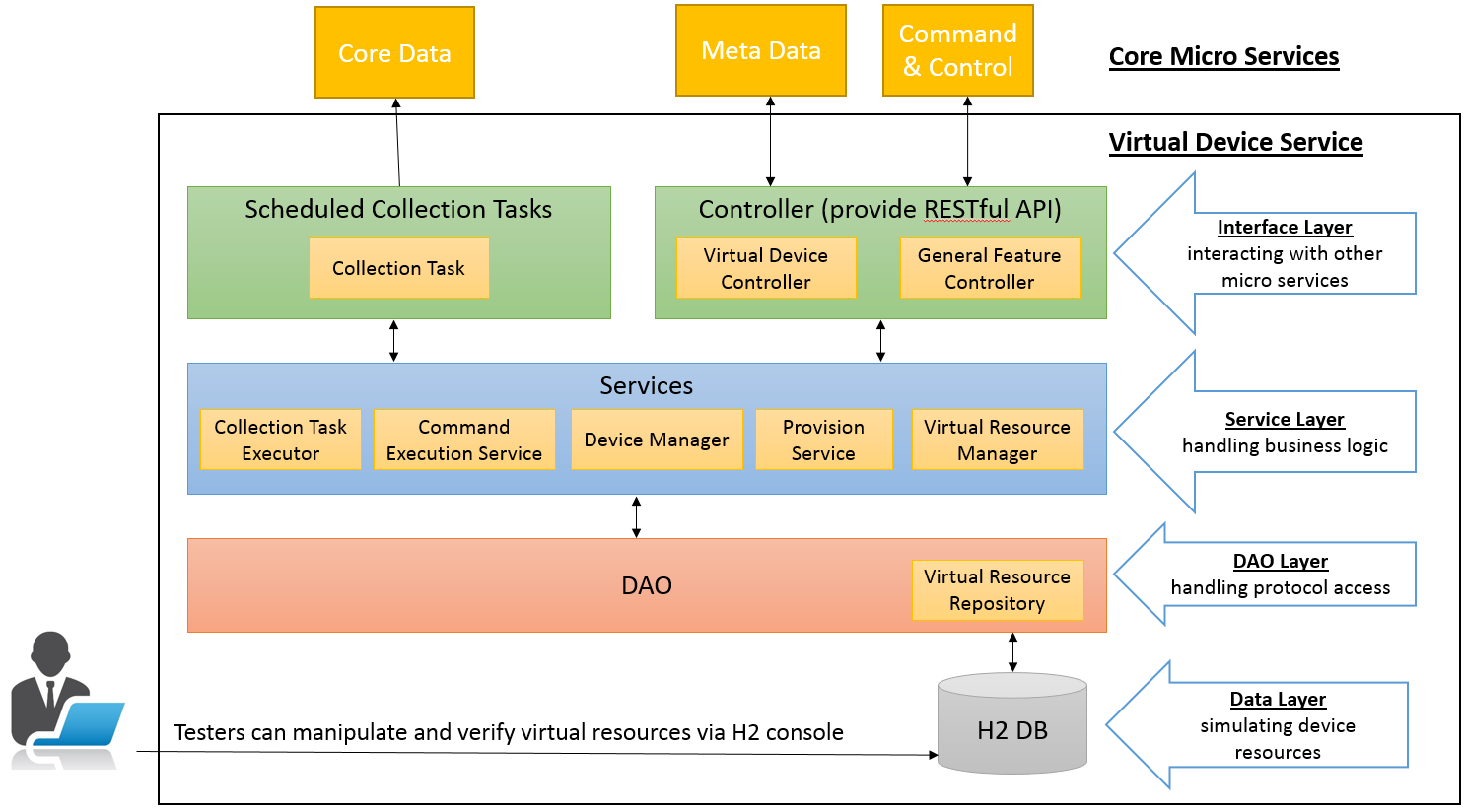
4）虚拟设备服务的调度程序应该几秒钟从虚拟设备中收集数据。

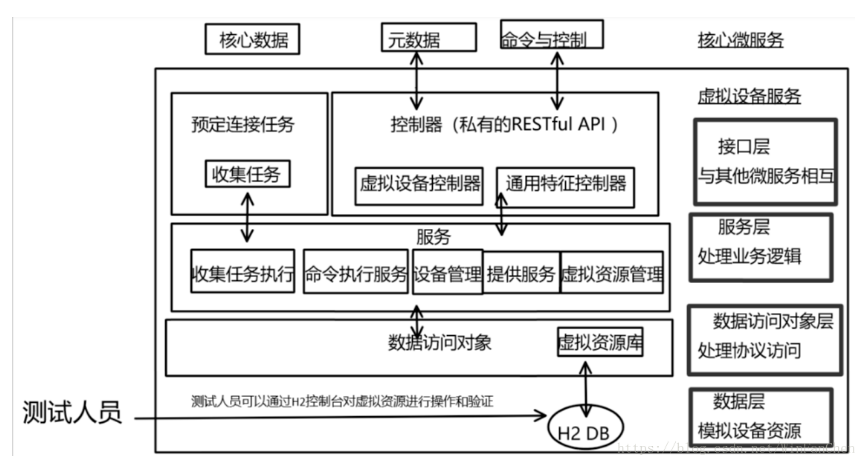
application.collection-frequency=15

## 5.4 虚拟设备服务系统结构

虚拟设备服务采用常规的MVC设计模式，将逻辑划分为不同的层。

系统结构图





**1接口层**—与其他微服务交互。

controller提供了RESTful API。该实现位于org.edgexfoundry.device.virtual.controller包中。

Scheduled Collection Tasks是一组异步任务，定期执行，为每个虚拟资源(GET命令)创建它们。详细的实现请参阅org.edgexfoundry. device.virtualal.scheduling package。此外调度程序从元数据微服务中读取调度schedule和ScheduleEvent，并安排收集任务。

任务执行逻辑位于org.edgexfoundry.device.virtual.service.impl.CollectionTaskExecutorImpl中，

任务创建行为位于org.edgexfoundry.device.virtual.service.impl.VirtualResourceManagerImpl.createDefaultRecords（）中

**2 服务层-**处理业务逻辑，如执行集合任务和命令、管理概要和设备等。

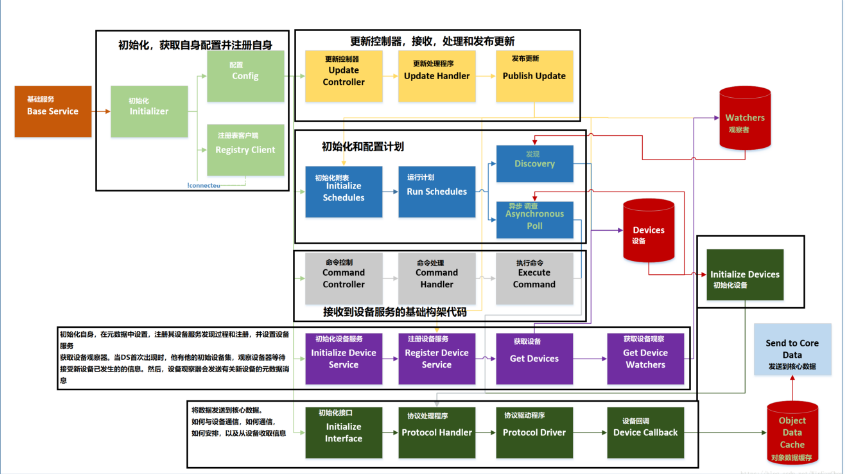
有关详细信息，请参阅org.edgexfoundry.device.virtual.service.impl包。

**3 DAO层-**处理协议访问/数据访问对象层。

对于虚拟设备服务，org.edgexfoundry.device.virtual.dao包中的Spring Data JPA接口。Spring框架将处理访问H2 DB的通信工作。

**4 数据层**-一个用来模拟设备资源H2 DB。

5.5 SDK的体系结构



5.6设备概要文件：

如果结构符合“设备和设备概要模型”(如下图显示的结构)，用户可以用YAML格式定义任何虚拟设备概要文件。通过将文件路径分配给应用程序属性“application. Device -profile-paths”，虚拟设备服务加载该文件夹下的所有YAML文件，该属性接受用逗号(，)分隔的多个值。

例如，以下设置导致虚拟设备服务在./bacnet\_sample\_profiles和./modbus\_sample\_profiles文件夹下加载所有YAML文件。

https://nexus.edgexfoundry.org/content/sites/docs/staging/master/docs/_build/html/_images/EdgeX_VirtualDeviceProfile.png

除了概要文件定义之外，在概要文件定义的“device.properties”文件中还定义了ValueDescriptors。ValueDescriptors的结构需要符合“设备概要文件”中的属性，在设备创建回调过程中创建ValueDescriptors并发送给Core Data 微服务。

通过application.properties文件设置auto-create-device= true(默认值为true)，虚拟设备服务在启动时自动为每个概要文件创建设备实例，设备实例开始向Core Data微服务发送事件和读数。

