Table of Contents

CoAP协议中文版

第一章 简介

在互联网上，使用web服务（web API）已经非常普遍，大多应用都在使用基于web的REST架构。

Constrained RESTful Environments（CoRE）的工作目标是采用恰当的方式在受限节点（如8位微控制器、较小RAM和ROM）和受限网络（例如6LoWPAN，[RFC4944]）上实现REST架构。6LoWPAN等受限网络支持把IPv6数据包分片成为小的链路层数据帧。然而，这导致数据发送成功率的下降。CoAP协议的设计目标之一是使数据包开销尽可能小，以减少分片的发生。

CoAP目标是设计一个通用的网络协议，满足受限环境的特殊需求，特别考虑了能源、楼宇自动化和其它M2M应用。CoAP不是对HTTP[RFC2616]协议的压缩，而是实现了REST的一个子集，并为M2M应用程序做了优化。尽管CoAP协议可以被当作一个HTTP压缩之后变得更紧凑的协议来使用，但更重要的是它还提供了M2M的一些特性，例如内置的资源发现、多播支持和异步消息交换。

本文档描述了CoAP协议，它可以很容易的与HTTP协议互相转换以集成到现有的web中，同时满足了许多特定的需求，如多播支持、非常小的开销和足够简单以适应受限环境和M2M应用。

1.1 特性

CoAP协议主要有如下特性：

- 它是一个满足受限环境下M2M需求的协议；
- 基于UDP[RFC0768]，拥有可选的可靠性保障，支持单播和多播请求；
- 异步消息交换；
- 轻量级的头部，且解析复杂度低；
- 支持URI和Content-Type；
- 能实现简单的数据缓存和代理；
- 无状态的HTTP映射，可以构建代理服务器，使CoAP资源可以用HTTP协议访问，也可以使HTTP接口实现于CoAP协议之上；
- 支持DTLS[RFC6347]协议。

1.2 术语解释

出现在本文档中的关键字“必须”、“必须不”、“要求”、“应该”、“应该不”、“应当”、“应当不”、“建议”、“不建议”、“或许”和“可选”，当加粗时，都在[RFC2119]中定义。这些词语也可以普通的形式出现，则不具备预先定义的含义。

本文档要求读者熟悉[RFC2616]中讨论的所有术语，包括resource , representation，cache和fresh。（对HTTP协议更新的RFC有7230到7235，但由于本文档的完成早于这些对HTTP协议更新的rfc，故本文档引用的是先前的HTTP协议版本：[RFC2616]）。此外，本文档定义了以下术语：

- 端(Endpoint)

  参与CoAP协议的一个实体。通俗的说，一个端指的是一个节点(node)，尽管Host这个词更符合互联网标准，与传输层的多路复用技术联系起来，可以包括一个UDP端口号。

- 发送者(Sender)

  消息的发起端。从交互角度来说，也称为“源端”（source endpoint）。

- 接收者(Recipient)

  消息的目的端。从交互角度来说，也称为“目的端”（destination endpoint）。

- 客户端(Client)

  消息请求的源端，消息响应的目的端。

- 服务端(Server)

  消息请求的目的端，消息响应的源端。

- 原始服务端(Origin Server)

  存储或创建一个给定的资源的服务端。

- 中间人(Intermediary)

  对于一个原始服务端（也可能是其它的中间人）来说，一个即是服务端又是客户端的CoAP端。常见的场景是一个代理。本文档中讨论了几种这样类型的代理。

- 代理(Proxy)

  代理是主要作用为转发请求和响应消息的中间人，有可能起到缓存，命名空间转换，或者协议转换的作用。与一般的中间人不同的是，代理通常不实现任何语义。在转发请求的场景下，根据定位的不同，主要分为两种：正向代理和反向代理。在某些情况下，一

个端有可能根据每一个请求的特性转换自己的角色，可以作为原始服务端、正向代理或反向代理。

- 正向代理(Forward-Proxy)

  即客户端的代理，通常是通过本地配置，用来代表客户端发出请求，必要时做一些转换。有些转换是很细微的，如代理“coap”开头的URI，然而有些请求则需要在其它应用层协议和coap协议间作转换。

- 反向代理(Reverse-Proxy)

  代替一个或多个服务端接收请求的端，必要时做一些转换。与正向代理不同的是，反向代理对客户端可能是完全透明的。反向代理接收请求，把它自己当成目标资源的原始服务端。

- CoAP到CoAP代理(CoAP-to-CoAP Proxy)

  把一个CoAP请求映射到另一个CoAP请求的代理。也就是说它的服务端部分和客户端部分都使用CoAP协议。与“跨协议代理”形成对比。

- 跨协议代理(Cross-Proxy)

  跨协议代理指的是在不同协议之间做转换的代理，例如一个从CoAP到HTTP的代理，或者HTTP到CoAP的代理。相对于CoAP到CoAP代理，跨协议代理有很多种。

- 需应答消息(Confirmable Message)

  要求ACK的消息称为需应答消息。当没有发生数据包丢失的时候，每个需应答消息必定会有一个类型为ACK或Reset的响应。后面简写为CON。

- 不需应答消息(Non-confirmable Message)

  不要求ACK的的消息称为不需应答消息。通常用于某些应用中周期性的重复发送数据的情形，例如不断的读取一个传感器的数据。后面简写成NON。

- ACK消息(Acknowledgement Message)

  ACK消息用于确认某个可靠消息已经到达。ACK消息自身并不代表这个请求处理的结果是成功还是失败。ACK消息有可能会同时为附带响应(Piggybacked Response)。

- 重置消息(Reset Message)

  Reset消息代表的是一个消息（需要应答或者不需要应答的消息）被收到了，但是由于缺少某些上下文信息而无法被正常的处理。这种情况通常是由于接收节点重启了，因而缺失了一些必要的信息，导致当前接收到的消息无法被处理。利用reset消息，也是一种低开销的检查端是否存活的方式（也称作CoAP ping，发送一个空的需应答消息）。后面简写成RST。

- 附带响应(Piggybacked Response)

  附带响应指的是，对于一个请求消息，它的ACK消息中包含了响应数据。

- 单独响应(Separate Response)

  当请求是一个需应答消息时，如果它的ACK是一个空消息（因为服务端对该请求产生对应结果需要一些时间），那么就需要一个单独的消息交换过程来完成对请求的响应（5.2.2节）。

- 空消息(Empty Message)

  空消息的code是0.00，有可能是请求，也可能是响应。空消息只有4个字节的header，没有body部分。

- 重要选项(Critical Option)

  指的是这样的选项：只有接收端正确的理解这个选项，那么这个请求才能被正确的处理（5.4.1节）。注意，这些选项的值通常都有一个范围，不支持的选项值会导致错误的响应消息或者拒绝这个消息。

- 非重要选项(Elective Option)

  非重要选项指的是如果接收端不理解这个选项，那么可以把它忽略。协议允许忽略这个选项而对消息进行处理（参见5.4.1节）。

- 非安全选项(Unsafe Option)

  非安全的选项指的是，代理必须理解这个选项才能正确的转发这个消息。并非所有的重要选项都是非安全选项。

- 转发安全选项(Safe-to-Forward Option)

  代理不理解这个选项，但也可以安全的转发这个消息。在不理解这个选项的情况下，也可以转发这个消息（参见5.4.2节）。

- 资源发现(Resource Discovery)

  资源发现指的是CoAP客户端获得服务端支持的所有资源列表的过程（参见第7章）。

- 内容格式(Content-Format)

  内容格式指的是互联网媒体类型和内容编码，用一个数值型标识符来标识。这个数值型标识符在“COAP 内容格式”中定义。当重点注意力不是在这个数值型的标识上，而是在资源表现本身时，就称作“表现格式”（REPRESENTATION FORMAT）。

更多关于受限节点和受限节点网络的术语，请参考[RFC7228]。

在本文档中，术语byte指的是通常的字节定义（即一个字节为8bit）。

本协议中所有长度超过一个字节的整型都采用网络字节序。

本文中使用的符号和C语言中类似，例外情况：运算符" ** "表示幂运算。

第二章 受限应用协议CoAP

CoAP协议的交互模型与HTTP的客户端/服务端模型类似。然而，在M2M的交互场景中，一个使用CoAP协议的设备通常既是客户端又是服务端。CoAP中的请求与HTTP协议中的请求相同，是由客户端发起的，请求一个位于服务端的资源（用URI标识），执行一个动作（用Method Code标识）。然后服务端发回一个响应，带有一个响应代码（Response Code），这个响应中有可能也包含一个资源的表现（附带响应格式）。

与HTTP协议不同的是，CoAP的交互是异步的，构建于面向数据报的传输协议，如UDP。交互是通过一个消息层来实现的，消息层提供了可选的可靠性支持（采用指数回退）。CoAP协议中定义了四种类型的消息： CON, NON, ACK和RST。这四种类型的消息中包含有请求和响应标识码，标识着这些消息是请求还是响应。请求可以包含在CON和NON两种类型中，而响应则除了可以包含在CON和NON之中，还可以包含在附带响应的ACK中。

从逻辑上，可以把CoAP协议划分为两层：消息层，用于处理UDP数据包和异步；请求/响应层，使用Method和Response Code，具体见图1。当然，CoAP是一个协议，消息和请求/响应仅仅是其头部特性。

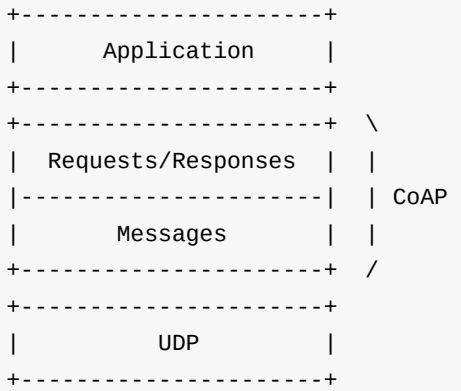

图1 CoAP中的抽象层次

2.1 消息模型

CoAP的消息模型是建立在UDP端到端通信的基础上的。

CoAP的头部为固定长度的（4个字节）二进制格式，其后是紧凑的二进制格式的选项部分，然后是数据部分(payload)，请求和响应都采用这种格式。CoAP的消息格式在第3章中详细讲述。每个消息都包含一个消息ID，用于检测重复提供传输可靠性。（这个消息ID是连续的，包含有16位，在默认的协议参数配置下，它允许每秒钟从一端到另一端传输大约250条消息）。

通过把消息标记为CON的，可以保障消息传输的可靠性。如图2所示，在收到一个CON消息之后，接收端会发送一个带有相同消息ID(Message ID)（在这个例子中是0x7d34）的ACK。如果在默认的超时时间之后没有收到带有相同消息ID的ACK，那么它将会被重传，如果仍然没有收到ACK，此后重传超时时间会以指数级递增。当接收端无法处理一个CON消息(也无法返回一个正常的错误响应)时，它将会回应一个RST消息，而不是ACK。

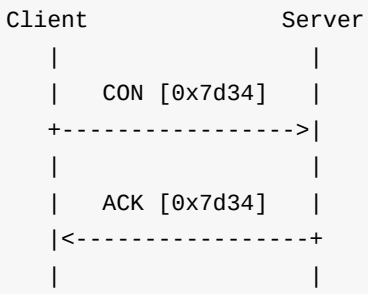

图2 可靠消息传输

当消息不需要可靠传输（例如持续不断的读取一个传感器数据）时，可以发送NON的消息。如图3所示，这些消息不需要应答，但它们仍然拥有消息ID，用于检测重复（在这个例子中0x01a0）。当接收端无法处理一个NON消息时，它有可能会返回一个RST消息。

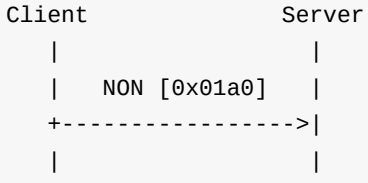

图3 非可靠消息传输

第4章详细讲述了CoAP消息的细节。

由于CoAP运行在UDP之上，它也支持目的ip为多播地址，可以实现多播CoAP请求。第8章讲述了正确的使用多播地址发送CoAP消息，和避免由此造成响应拥塞的预防措施。

第9章定义了CoAP的几个安全模型，从无安全的，到基于证书的安全机制。在本文档中，只指定DTLS作为协议的安全基础。文档[IPsec-CoAP]对在CoAP中使用IPsec进行了讨论。

2.2 请求响应模型

CoAP的请求和响应的语义都包含在CoAP消息中，请求和响应的消息分别带有方法码（Method Code）和响应码（Response Code）。可选的或者是默认的请求和响应信息，例如URI和数据的媒体类型等，都做为协议中的选项部分。CoAP使用一个Token来匹配请求对应的响应（见5.3节）。注意，这个Token和消息ID不同。

请求消息分为CON和NON两种。对于CON类型的请求，如果响应数据可以立即生成，那么对于请求消息的ACK就会同时携带响应数据。这就是附带响应，在5.2.1节中详细讲述。（不需要对附带响应再进行单独的应答，因为假如携带响应数据的ACK丢失，那么客户端会重传请求消息）。图4中展示了对于两个GET请求，服务端返回附带响应的例子，一个成功，一个导致了4.04（资源未找到）响应。

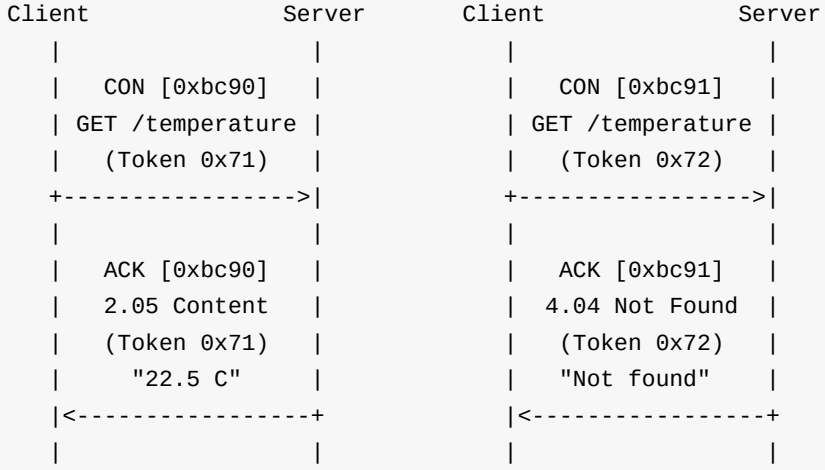

图4 对两个GET请求的附带响应

如果请求消息是一个CON类型的，而服务端无法立即响应，那么它就会立即发回一个空的ACK消息，以免客户端重传请求消息。当响应数据准备好了之后，服务器端就会把它组装成一个新的CON类型的消息（这需要客户端的ACK）。这种形式被称为“单独响应”，如图5所示，更多细节参见5.2.2节。

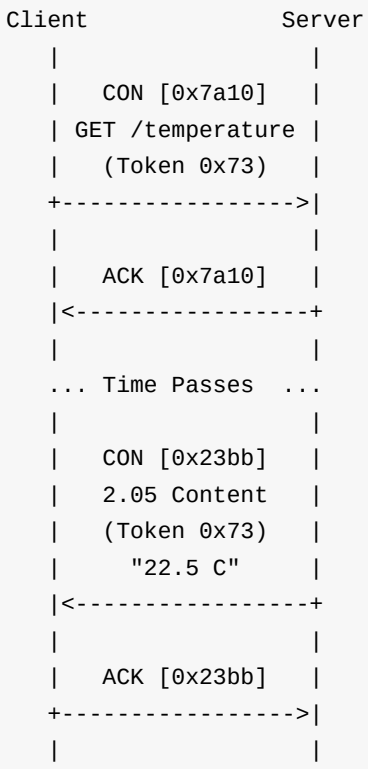

图5 GET请求和独立响应

如果一个请求是以NON类型的消息发送的，那么一般来说响应也将是一个NON类型的消息，但服务器也有可能发送一个CON类型的消息作为响应。这种交互如图6所示。

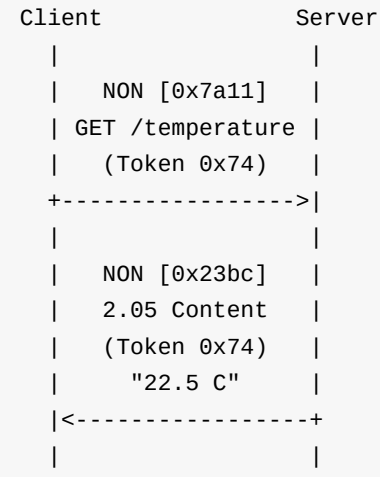

图6 NON类型消息的请求和响应

类似HTTP，CoAP协议也使用GET, PUT, POST和DELETE方法，具体的语义在5.8节中讲述。（注意，CoAP协议的这些方法的语义很像HTTP，但和HTTP不完全一样。读者对于HTTP协议的经验对理解CoAP是很有帮助的，但是二者之间还是有不少的区别，值得阅读本规范）。

在特定的情况下，可以增加这四种方法之外的方法。新的方法不一定要使用成对的请求/响应的形式。即使对于现有的方法，一个请求也有可能产生多个响应，例如一个多播请求（第8章），或者一个订阅(Observe)选项。

服务端对于URI的支持是很简单的，因为客户端已经把URI拆分为主机、端口、路径和参数，并可以使用默认值。响应代码是HTTP状态代码的一个子集，但增加了几个CoAP特有的响应码，在5.9节中讲述。

2.3 中间人和缓存

为了能够快速的对客户端的请求进行响应，CoAP协议支持将响应缓存。通过在CoAP响应消息中的时效性和有效性信息，可以启用简单的缓存。缓存可以位于端中，也可以位于中间代理。5.6节具体描述了缓存。

代理在受限网络中十分有用，它能够有效的减少网络传输、提高性能、获取正在休眠的设备的资源，并且能提高安全性。协议支持一个CoAP端代理另一个CoAP端的请求。当使用代理时，请求信息包含了资源的URI，目标IP地址则被设置成了代理的IP地址。关于代理功能的更多信息请阅读5.7节。

由于CoAP是根据REST架构设计的，因此表现的和HTTP协议很类似，很容易做从CoAP到HTTP的映射和从HTTP到CoAP的映射。这样的映射可用于使用CoAP协议实现一个HTTP REST接口，或者在HTTP与CoAP之间互转。这个互转可以由一个跨协议代理来实现，这个代理把请求方法、返回代码、媒体类型、选项转换为对应的HTTP中的特性。第10章讲述了更多关于HTTP映射的细节。

2.3 资源发现

资源发现对于M2M交互来说是很重要的，因此被CoRE Link Format [RFC6690]所支持，如第7章所讲述。

第三章 消息格式

CoAP的消息格式是很紧凑的，默认运行在UDP上（每个CoAP消息都是UDP数据包中的数据部分）。CoAP也可以运行在DTLS协议上（见9.1节）和其它传输协议上，例如SMS，TCP或SCTP，这些不属于本文档的范畴（CoAP不支持UDP-lite[RFC3828]和UDP zero checksum[RFC6936]）。

CoAP消息用二进制格式进行编码。这个消息格式以一个固定4个字节的头部开始。此后是一个长度在0到8字节之间的Token。Token值之后是0个或多个Type-Length-Value(TLV)格式的选项(Option)。之后到整个数据报的结尾都是payload部分，payload可以为空。

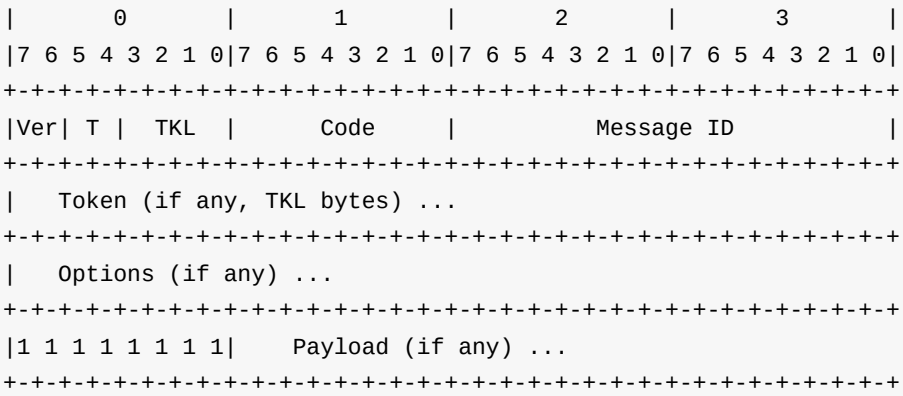

图7 消息格式

头部字段定义如下：

- 版本号(Ver)：2-bit无符号整型，代表CoAP版本号。本文档的实现必须设置这个字段为0b01。其它的值为今后其它版本保留。对于带有未知版本号的消息，必须忽略。
- 类型(T)：2-bit无符号整型。代表这个消息的类型是：CON(0), NON(1), ACK(2),或RST(3)。这些消息类型的语义在第4章进行定义。
- Token长度(TKL)：4-bit无符号整型。表示变长的Token字段(0-8字节）的长度。长度9-15是保留的，不能设置长度为9-15。如果设置了长度为9-15，必须被当作消息格式错误来处理。
- 代码(Code)：8-bit无符号整型。拆分为3-bit的分类信息和5-bit详细信息。写作”c.dd”。c是3-bit长，可以是一个从0到7的数字，dd是5-bit长，它一个两位的数字，从00到31。分类信息c可以代表是一个请求（0），一个成功的响应（2），一个客户端错误响应（4），或者一个服务端错误响应（5）。所有其它的值都是保留的。代码0.00是一个特

殊的情况，表示一个空的消息。当消息是一个请求时，Code字段表示请求方法。当响应时，Code字段代表响应代码。Code字段所有可取的值都在CoAP代码表（12.1节）中定义了。请求和响应的语义定义在第5章。

- 消息ID(Message ID)：16-bit无符号整型，网络字节序。用于检测消息重复以及匹配ACK/RST类型的消息和CON/NON类型的消息。生成消息ID和匹配消息的规则在第4章中讲述。

头部之后是Token值，可以有0到8个字节，由Token长度字段指定。这个Token值用于将某个请求和对应的响应关联。生成Token和关联请求与响应的规则在5.3.1节讲述。

头部和Token之后，是0个或多个选项（见3.1节）。一个选项之后，有可能是消息结束，也可能是另一个选项，也可能是payload标识符和payload部分。

在头部、token和选项之后，是payload部分（可以没有payload）。如果有payload，并且长度不为0，那么payload之前有一个固定长度为一个字节的payload标识符（0xFF），它标志着选项部分的结束和payload部分的开始。payload部分从标识符之后开始，一直到这个UDP数据报结束，也就是说，payload部分的长度可以根据UDP数据报的长度计算出来。如果没有payload标识符，那么就代表这是一个0长度的payload。如果存在payload标识符但其后跟随的是0长度的payload，那么必须当作消息格式错误处理。

实现注意：0xFF这个值有可能出现在一个选项的长度或选项的值中，所以简单的扫描0xFF来寻找payload标识符是不可行的。作为payload标识符的0xFF只可能出现在一个选项结束之后下一个选项有可能开始的地方。

3.1 Option格式

CoAP定义了许多option。消息中的每个option都有一个option编号，option值长度，和option值。 消息中的option号（TLV格式中的T）并不是直接指定option编号的。所有的option必须按实际option编号的递增排列，某一个option和上一个option之间的option编号差值为delta；每一个TLV格式的option号都是delta值（数据包中第一个option的delta即它的option编号）。同一个编号的option再次出现时，delta的值为0。

option编号由“CoAP option编号”表维护（见12.2节）。5.4讲述了本文档中定义的option的语义。

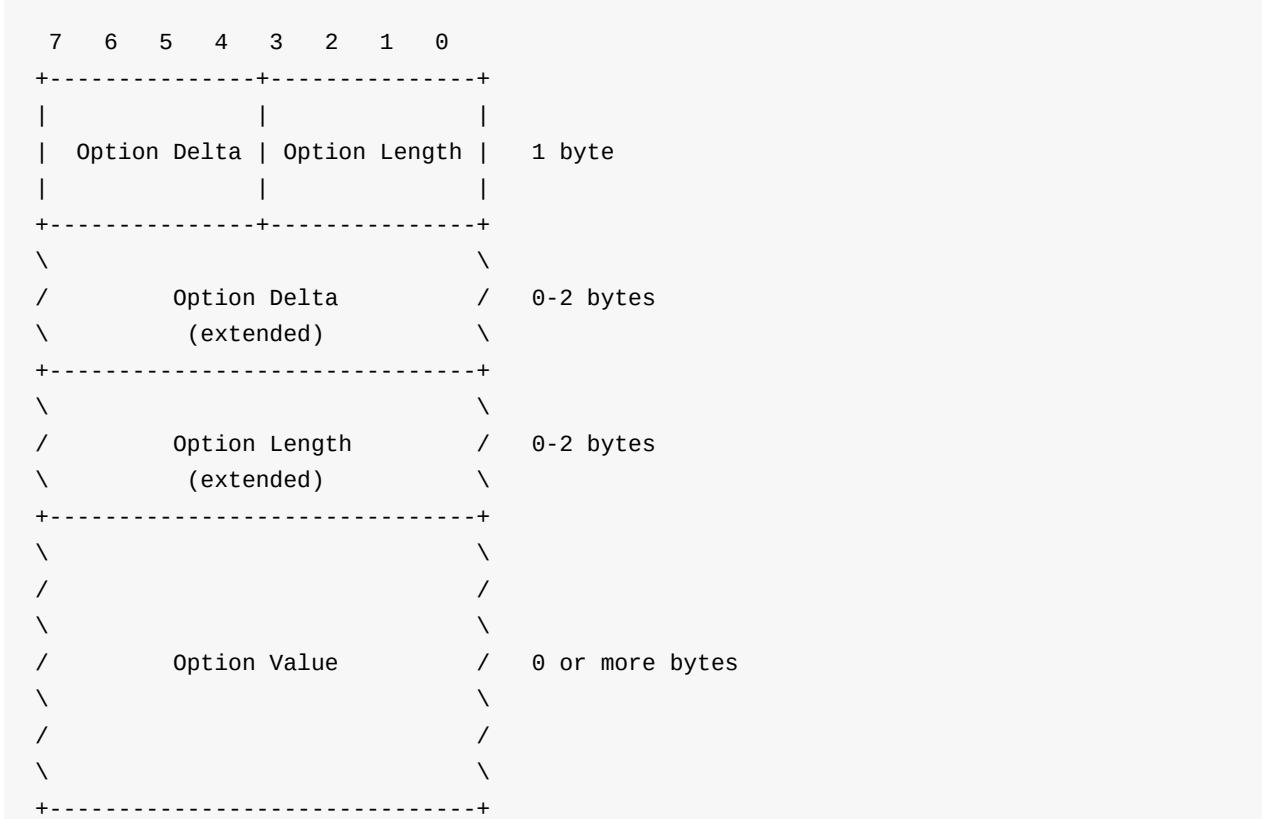

图8 option格式

一个option之中的各个字段的含义如下：

- Option Delta：4-bit无符号整型。值0-12代表option delta。其它3个值作为特殊情况保留：
- 当值为**13**：有一个8-bit无符号整型（extended）跟随在第一个字节之后，本option的实际delta是这个8-bit值加13。
    - 当值为**14**：有一个16-bit无符号整型（网络字节序）（extended）跟随在第一个字节之后，本option的实际delta是这个16-bit值加269。

    - 当值为**15**：为payload标识符而保留。如果这个字段被设置为值15，但这个字节不是payload标识符，那么必须当作消息格式错误来处理。

- Option Length：4-bit无符号整数。值0-12代表这个option值的长度，单位是字节。其它3个值是特殊保留的：
    - 当值为**13**：有一个8-bit无符号整型跟随在第一个字节之后，本option的实际长度是这个8-bit值加13。
    - 当值为**14**：一个16-bit无符号整型（网络字节序）跟随在第一个字节之后，本option的实际长度是这个16-bit值加269。
    - 当值为**15**：保留为将来使用。如果这个字段被设置为值15，必须当作消息格式错误来处理。
- Option Value：共option Length个字节。option值字段的长度和格式取决于具体的option，有可能定义变长的值。3.2节讲述了本文档所使用的option格式。其它文档中定义的option可能使用其它option值的格式。

3.2 Option Value格式

- Empty：长度为0。
- Opaque：一个（用户自定义的）字节序列。
- Uint：一个非负的整数，以网络字节序表现，由option Length字段指定其长度。

  option有可能指定字节数的范围。如果有选择的话，发送者应该用尽可能少的字节数表示这个整数，如省略开头为的0的字节。例如，数字0用空的option值来表示（0长度字节序列），数字1用一个单字节表示，MSB first（高位优先）下为0b00000001。接收端必须能够处理0开头的值。

  实现注意：在一些高受限的、模板化的实现（比如由硬件实现）下，允许发送者使用固定长度的option。

- String：Unicode字符串，使用UTF-8[RFC3629]编码，使用Net-Unicode格式[RFC5198]。

  注意，在CoAP协议中，所有使用UTF-8编码的字符串可以不需要经过标准化转换而被直接使用，或者作为用户自定义的字节序列比较，除非该Unicode字符串是从CoAP协议之外的资源中引入的。注意，所有的ASCII（不作为特殊控制字符使用）字符都是合法的UTF-8 Net-Unicode字符串。

第四章 消息传递

CoAP消息在端与端之间的交换是异步的。消息承载了CoAP的请求和响应，请求和响应的语义在第5章中定义。

由于CoAP运行在如UDP这种非可靠的传输协议上，因此CoAP消息也许是乱序到达的，或是重复的，甚至被丢失。出于这个原因，CoAP中实现了一个轻量级的可靠性机制，这个机制并不是试图重新实现TCP的所有特性。它有以下特性：

- 简单的停等(stop-and-wait)重传机制，对于需要应答的消息，每次的重传等待时间会指数级增长。
- 对于CON和NON类型的消息，都进行重复检测。

4.1 消息和端

一个CoAP端可能是CoAP消息的源端或者目的端。端具体的定义（即标识）由CoAP所使用的传输协议来决定。对于本文档所指定的传输层协议，端的标识取决于所使用的安全模式（参见第9章）：对于无安全的模式，端由一个IP地址和一个UDP端口号来标识。对于其它安全模式，端的标识由具体的安全模式来定义。

CoAP协议有许多不同的消息类型，由CoAP头部的Type字段标识。

消息为请求、响应还是为空，这和请求/响应模型相关，是由头部的请求/响应码字段（code段）定义的。这个字段允许的值在CoAP代码表中定义（见12.1节）。

空消息的code段被设为0.00。它的TKL字段必须设为0，且Message ID字段后必须没有数据，否则应当做消息格式错误处理。

4.2 可靠的消息传递

如果CoAP协议头部中T字段标识为CON类型，则说明其采用可靠传输。CON类型的消息一般携带请求或响应，除非它是想要发送空消息以引发RST。接收端接收到CON类型消息后，必须是以下两种情况之一：

- 通过返回一个ACK消息确认已收到。回复的ACK必须带有和CON消息相同的Message ID，并且必须携带有响应（附带响应格式）或者为空消息（单独响应格式）。
- 如果是因为该消息缺少上下文而无法被正确处理（例如消息为空、使用了保留的Code类别(1，6或7)，或者消息格式错误），拒绝这个消息。通过回复对应的RST消息，或者直接忽略，接收端可以拒绝CON类型的消息。回复RST消息必须带有和CON消息相同的Message ID，并且必须为空消息。参看5.2.1节和5.2.2节。

如果接收端想要拒绝一个ACK消息或者RST消息（例如ACK消息携带有请求或者有保留的code类别，RST消息不为空消息等），只需要忽略即可。一般的，ACK和RST消息的接收端必须不应答ACK或RST消息。

注意：发送端重传消息的间隔以指数增长，直到它收到一个ACK或者RST消息，或者达到最大重传次数。

重传由超时时间和重传计数控制。对于每一个CON类型的消息，发送端必须一直维护超时时间和重传计数,直到收到对应的ACK或者RST。对于一个新的CON消息，初始的超时时间被设置为介于ACK_TIMEOUT和ACK_TIMEOUT*ACK_RANDOM_FACTOR)之间（参见4.8节）的随机值（通常不是整数秒），重传计数被设置为0。当超时发生，且重传计数的值小于MAX_RETRANSMIT，消息被重传，重传计数增加，超时时间变为原来的两倍。如果在超时发生的时候重传计数达到了MAX_RETRANSMIT，或者收到了一个RST消息，那么就会放弃消息的传输，由应用程序来处理这个传输失败；如果在超时之前收到了ACK，那么传输就被认为成功了。

这一机制并不强制要求精准的时钟来实现上述指数回退算法。具体的说，某个端可能由于它周期性的休眠而没有赶上某一个重传时间点，但它却赶上了下一个。然而，两次重传之间的最小时间间隔是ACK_TIMEOUT，并且整个传输和重传的过程必须在MAX_TRANSMIT_SPAN（见4.8.2节）之内，尽管这意味着发送者有可能会错过传输的机会。

发送CON消息的端有可能在重传计数到达MAX_RETRANSMIT之前就放弃重传。例如，应用程序取消了这一请求，因为它已经不再需要响应；或者有其它证据表明消息已经到达，比如该请求消息导致接收端产生了单独响应，这种情况下就很明显，丢失的仅仅是请求消息的ACK而已，重传这个请求消息毫无意义。然而，响应端必须不能依赖请求端这种跨层的行为，它必须维护状态，为这个（可能重复的）请求回复ACK。如果需要的话，即使源请求端确认了CON响应（单独响应的第2步），接收端也应该回复（源请求端重传的请求）ACK。

另一个放弃重传的原因可能是收到了ICMP错误。如果希望处理ICMP错误以减轻潜在的ICMP欺骗攻击的影响，那么在实现上，应该仔细检查产生ICMP消息的原始数据，包括端口号和CoAP头部信息，如Message type, code, Message ID和Token；如果由于UDP提供的接口API的限制而无法检查，那么就应该忽略ICMP错误。如果遵循了第4.6节中的“实现注意”，那么正常情况下不应该发生数据包过大错误(IPv4的"fragmentation neede and DF set" [RFC0792])RFC4443]，因此应该被忽略。如果没有遵循，那么就应该进入一个路径MTU计算算法[RFC4821]。Source Quench和Time Exceeded类型的ICMP错误应该被忽略。主机，网络，端口或协议不可达错误和参数错误有可能经过适当的审查后用于通知应用程序，消息发送失败。

4.3 不可靠的消息传递

有的消息不需要应答。尤其是周期性的重复的消息，例如对传感器的数据的重复读取，并不需要每次都成功。

在没有可靠性保障的情况下传输，可以把消息标记为NON类型，作为更加轻量级的选择。NON消息总是承载着一个不为空的请求或者响应。接收端必须不应答NON消息。如果缺少必要的上下文而无法正确的处理（包括消息是空的，使用了保留类别的code(1,6或7)，或者消息格式错误）那么接收端必须丢弃这个NON消息。同时服务端有可能会忽略该消息或者响应一个对应的RST消息。

在CoAP层，发送者没有任何办法得知NON类型的消息是否被接收端收到。发送者可能在MAX_TRANSMIT_SPAN(在第4.7节的条款所限制，特别是PROBING_RATE，如果没有收到响应)之内发送多个副本，网络传输也有可能导致消息出现重复。为了使接收端能够只处理一次，NON消息也带有Message ID(和CON消息的Message ID共用数字池)。

总结第4.2和4.3节，四种类型的消息的使用如表1所示。“*”代表的是正常情况下不会这么使用这个组合，除非为了引起一个RST消息(也就是“CoAP ping”)。

	CON	NON	ACK	RST
Request	X	X	-	-
Response	X	X	X	-
Empty	*	-	X	X

表1： 消息类型的使用

4.4 消息之间的关联

一个ACK消息或RST消息必定和一个CON消息或NON消息关联起来。关联是通过Message ID和附加的接收端的地址来实现的。Message ID是一个16-bit的无符号整数，由CON或NON消息的发送端生成，包含在CoAP头部中。接收端发回的ACK或RST消息必须带有相同的Message ID。

在EXCHANGE_LIFETIME时间之内，相同的Message ID必须不能重复使用(即与同一个端通信)(见第4.8.2节)。

实现注意：有许多策略可以用来生成Message ID。最简单的情况是，通过一个变量来生成Message ID，每当发送了一个CON或NON消息，不论目标地址或端口是什么，都改变这个变量。如果一个端需要处理大量的连接，那么它也可以使用多个变量，例如为每个前缀或者目标地址使用一个变量。(注意，有些接收端可能无法区分发收到的数据包是单播还是多播的，所以生成Message ID的端必须要确保这种情况下不会重复)。强烈建议这个变量的初始值是随机的，这样可以降低off-path攻击成功的可能性。

为了使ACK或RST消息与CON或NON消息匹配，ACK(或RST)消息的Message ID和源地址必须和CON(或NON)消息的Message ID和目的地址一致。

4.5 消息去重

在EXCHANGE_LIFETIME时间之内，当ACK消息丢失或者在第一个超时时间之前没能到达原始服务端，接收端可能收到多次重复的CON消息(由Message ID和源端地址标识)。接收端应该对每一次收到的重复消息都回以相同的ACK或RST，但应该只处理一次。当CON消息传输的请求是幂等的时候见第5.1节)，或者可以以幂等的方式来处理时，这一规则可以放宽。消息去除重复规则被放宽的例子如下：

- 服务端对于幂等的请求的每一次重传都回以相同的响应（第4.2节），这样一来它就无需维护Message ID的状态，在此情况下可以放宽规则。例如，如果重复处理的过程的开销小于保留上一个响应的开销，实现中可能把GET,PUT,或DELETE请求的重传当作独立的请求来处理。
- 对于一些非幂等的请求，只要在应用层语义上这个取舍是值得的，一些资源受限的服务器也可能会放宽规则。例如，如果一个POST请求对服务端的数据状态的影响是很短暂的，那么可能重复处理请求的开销会小于保留上一次传输的相同请求处理状态的开销。

接收端可能在NON_LIFETIME（第4.8.2节）时间内收到重复的NON消息（由Message ID和源端地址标识）。接收端应该忽略掉重复的NON消息，只处理一次。这一规则根据应用程序的语义，有可能被放宽。

4.6 消息大小

为了提高实现的质量，应该尽量使CoAP消息小到可以在一个链路层数据包中传输（见第1章）。CoAP文档本身只限制了消息大小的上限。大于IP数据包大小的CoAP消息会造成分片。一个CoAP消息应该尽量包含在一个IP数据包之内（即避免IP分片）并且在UDP包的payload之中。如果目的地址的MTU大小是未知的，那么应该假定IP包的MTU大小为1280字节。如果无法从头部获知消息大小，那么应该设置消息最大为1152字节，payload最大为1024字节。

实现注意：CoAP消息大小的选择适用于IPv6和目前的大部分IPv4地址。（然而，对于IPv4，很难保障绝对不发生IP分片。如果需要支持运行在受限网络上的IPv4，那么协议的实现应该使用更为保守的IPv4数据报大小，例如576字节。按照[RFC0791]中所述，IPv4网络的MTU可以小到68字节，减去用于安全开销的字节数，可用于UDP payload的就只剩下40字节。如果要解决这个问题，也许应该设置IPv4的DF标志位，并且执行一些路径MTU探测算法[RFC4821]。然而在使用CoAP的一般场景中，没有必要采用这些策略）在许多受限网络中，一个重要的数据分片发生在适配层（例如6LoWPAN L2数据包最大只有127字节，还包括了各种开销在内）。这使得协议的实现应该尽可能减少数据包大小，当消息大小达到3位数的时候，应该使用块传输(block-wise transfer)。

在受限节点上，消息的大小很重要。许多实现都需要为接收消息创建缓冲区。如果一个实现由于资源过于受限而无法分配足够的缓冲区，那么对于不使用DTLS的消息，它可以使用以下策略：如果接收到一个数据报，但缓冲区太小不足以存储整个数据报，接收端通常能够判断出数据报的尾部是否被丢弃，并且能获得数据报的开头。一般来说，CoAP的头部和option部分很可能在缓冲区中。因此服务端可以正确理解这个请求，如果payload部分被截断了，可以返回一个4.13(请求数据过长，见第5.9.2.9节)的响应。当某个端发送一个幂等的请求，但接收到的响应大于它缓冲区大小，那么它可以为Block Option设置一个恰当的值，重复发送这个请求(见Bormann, C.和Z. Shelby著”Blockwise transfer in CoAP”)。

4.7 拥塞控制

CoAP的基本拥塞控制由指数回退机制提供，见4.2节 。

为了避免拥塞，客户端（包括代理）应该严格限制他们同时与指定的服务器（包括代理）维持的未完成交互的数量（即NSTART值）。一个未完成的交互可以是一个仍在等待ACK的CON消息；也可以是一个在等待响应消息或者ACK的请求（这两种情况可以同时出现，做为同一个未完成的交互）。在本协议里NSTART的默认值为1.

另外，在未来拥塞控制可能会进一步得到考虑和优化，CoAP传输参数(如4.8节中定义)可以自动初始化，因此可能允许NSTART值大于1。

当EXCHANGE_LIFETIME超时后，如果该CON请求还没有收到响应，客户端就会停止等待。客户端停止等待一个已经收到ACK的CON请求(单独响应情况)或者对一个NON请求的响应的策略还未被定义。除非有额外的拥塞控制优化，否则它向其他未响应端的平均发送速率必须不超过PROBING_RATE。

注意：CoAP协议中，拥塞控制主要由客户端实现。然而，客户端可能会出现故障(或者客户端实际上就是攻击者)，例如，在第11.3节中提到的放大攻击。为了将损失（网络带宽及能耗）降到最低，对合理的应用请求，服务器应该对响应限速。对于行为异常的端来说限速是有且最有效的办法了。

4.8 传输参数

信息的传输由以下参数控制

name	default value
ACK_TIMEOUT	2 seconds
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
NSTART	1
DEFAULT_LEISURE	5 seconds
PROBING_RATE	1 byte/second

表2： CoAP协议参数

4.8.1 改变参数

以上参数（包括动态调整值）一般需要在具体应用的环境中配置，但是配置方法超出本协议的讨论范围。本规范建议在应用环境中使用统一的参数。同样，对于配置不一致的参数值所产生的影响也超出本协议讨论范围。

如果在目前的拥塞情况下，所选择的传输参数在网络上是可以稳定运行，配置使用不同的参数值，可能会使得拥塞控制的性能受到影响。注意，ACK_TIMEOUT小于1秒不符合RFC5405([RTO-CONSIDER]也提供了一些额外的说明)。CoAP的设计可以保证那些不能维持RTT测量的应用运行。然而，要减小ACK_TIMEOUT或者增加NSTART，只有在能保持那个测试结果的时候安全进行。当没有一种机制来确保拥塞控制安全的时候，绝对不能减小ACK_TIMEOUT或者增加NSTART。因此，要么尽量限制这种配置，要么等到将来的文档中定义新的标准。

ACK_RANDOM_FACTOR（应答随机因子）绝对不能小于1.0，且最好有一个和1.0不同的值以保护同步。

MAX_RETRANSMIT（最大重传时间）可以自由的调整，但是如果太小的话将会减小收到CON包的概率，然而如果比这里给定的值要大的话，那些对时间有要求的参数要做更进一步的调整（参见4.8.2节）。

如果传输参数的选择导致衍生时间的增加（参见4.8.2小节），必须确保调整后的值对所有通信端都有效。

4.8.2 传输参数的衍生时间

ACK_TIMEOUT（应答超时时间）、ACK_RANDOM_FACTOR（应答随机因子）和MAX_RETRANSMIT（最大重传时间）这三个参数共同影响着重传时间，重传时间反过来也影响着这些参数需要保持的时间长度。为了能够对这些derived时间值有一个明确的参考，这里给出以下名称：

- 最大传输跨度（MAX_TRANSMIT_SPAN）：指的是从CON消息第一次发送到它的上一次重发之间的最大时间间隔。对默认的传输参数，它的值是(2+4+8+16)*1.5 = 45s，或者一般表示为：

  ```
  ACK_TIMEOUT * ((2 ** MAX_RETRANSMIT) - 1) * ACK_RANDOM_FACTOR
  ```

- 最大传输等待时间（MAX_TRANSMIT_WAIT）：是指从第一次传输CON消息到发送方放弃接收ACK者RST响应之间的时间间隔。默认的传输参数的值是93秒，或者表示为：

  ```
  ACK_TIMEOUT * ((2 ** (MAX_RETRANSMIT + 1)) - 1) *ACK_RANDOM_FACTOR
  ```

此外，我们还需要在一些网络和节点的特征上做些假设。

- 最大时延（MAX_LATENCY）：指的是数据包从开始发送到完全接收之间的最大时间。该常量与[RFC0793](协议中的MSL（最大段周期）相关，一般被设定为2分钟。注意，这并不一定比最大传输等待时间小，最大时延并不是想要描述当协议工作良好的情形，而是确保在最坏的情况下有保障。我们也可以很随意的定义最大时延为100秒。除了大量的配置与以前的TCP接近，这个值也允许Message ID存活时间定时器由8-bit数值表示（单位为秒）。在这些计算中，没有考虑传输方向的影响（假设网络是对称的）。如果不是这种情况，接下来计算将变得稍微复杂一些。

- 处理延时（PROCESSING_DELAY）：指的是CON消息得到ACK响应的时间。我们假设节点恰好在发送端超时之前发送ACK，那么这个时间就等于ACK_TIMEOUT。

- MAX_RTT：往返时间的最大值，或者是：

  ```
  （2 * MAX_LATENCY) + PROCESSING_DELAY
  ```

从这些值中，我们可以得到与协议操作相关的下列值：

- 交换周期(EXCHANGE_LIFETIME)：它是指从开始发送CON消息到不再接收ACK之间的时间，即该信息在消息层交换时可以被清除。交换周期包括MAX_TRANSMIT_SPAN、发送过程的MAX_LATENCY、PROCESSING_DELAY和接收过程的MAX_LATENCY。注意，如果最后等待周期(ACK_TIMEOUT *(2* * MAX_RETRANSMIT)或MAX_TRANSMIT和MAX_TRANSMIT_WAIT的差)小于MAX_LATENCY，这里就无需考虑MAX_TRANSMIT_WAIT(这种情况一般不可能出现)。在这种情况下，EXCHANGE_LIFETIME简化为：

  ```
  MAX_TRANSMIT_SPAN + (2 * MAX_LATENCY) + PROCESSING_DELAY
  ```

一般缺省值为247秒。

- 不需确认消息周期(NON_LIFETIME)：它指的是从发送NON消息到该Message ID可以被复用之间的时间。如NON消息没有多次发送，那么它的值是MAX_LATENCY或者是100秒。然而，尤其是在多播应用中，一个CoAP发送端可能发送NON消息很多次。Message ID重用超出本文档范围内。接收端希望在MAX_TRANSMIT_SPAN的时间内判断是否是重传包。基于这样的目的，使用缺省值145秒或使用下面的值将会更安全：

  `MAX_TRANSMIT_SPAN + MAX_LATENCY`

对于仅想通过一个超时时间来判断Message ID是否能够重用的应用来说，使用较大的EXCHANGE_LIFETIME更安全。

表3列举了上述一些参数的缺省值

name	default value
MAX_TRANSMIT_SPAN	45 s
MAX_TRANSMIT_WAIT	93 s
MAX_LATENCY	100 s
PROCESSING_DELAY	2 s
MAX_RTT	202 s
EXCHANGE_LIFETIME	247 s
NON_LIFETIME	145 s

表3 一些参数的缺省值

第五章 请求/响应的语义

CoAP模型的操作方式和HTTP的请求与响应模型类似：一个CoAP端作为客户端发送一个或者几个CoAP请求给服务端，服务端会发送CoAP响应来回应。与HTTP不同的是，HTTP的请求与响应是在一个优先建立好连接的基础上发送的，CoAP的请求与响应是直接通过CoAP消息进行异步交换。

5.1 请求

CoAP的请求包含了应用到资源上的方法，资源的识别，payload和网络媒体的类别，以及可选的元数据请求。

CoAP支持基本的GET,POST,PUT和DELETE方法，这些方法可以简单的映射到HTTP中。它们和HTTP（见[RFC2616]第9.1节）有相同的安全属性（仅检索）和幂等（多次调用有同样的效果）特性。GET的方法是安全的，因此，只能对这个资源进行检索，不能进行其他任何操作。GET,PUT和DELETE方法必须以幂等的方式执行。POST不支持幂等，因为它的效果是由源服务端和目标资源决定的，POST的结果是建立一个新资源或者是更新目标资源。

请求通过设置CoAP头部的字段来初始化一些信息，比如该消息是是CON还是NON的。

请求使用的方法在5.8节中有详细的描述。

5.2 响应

在接收到并解析了一个请求之后，服务端会回应一个CoAP响应，这个响应是通过客户端生成的token来匹配。注意，这和CON消息要与ACK通过Message ID匹配的机制不同。

一个响应是通过CoAP头部的字段来定义为一个响应码，与HTTP状态码类似，CoAP响应码显示尝试理解并满足请求的结果，这些codes在5.9节有完整的定义。响应码的编号是在CoAP头部的字段中设定的，并保存在CoAP响应码表中（12.1.2节）。

```
 0
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|class|  detail |
+-+-+-+-+-+-+-+-+
```

图9 Response Code结构

上面8-bit的响应码中的高3-bits定义响应的种类。低5-bits没有任何分类作用，他们仅提供一些额外的细节（图 3-3）。

CoAP响应码格式是“c.dd”，这里”c”是一个十进制数，”dd”是两位十进制数。举个例子，”Forbidden”被定义为4.03——这是一个8-bit的值,16进制为0x83（4*0c20+3*）或者十进制*131*（*4*32+3）。

有3种响应码：

- 2-success：代表成功收到请求，理解并接收。
- 4-client Error：客户端错误，代表请求包含了错误的语法或者服务端不能满足请求。
- 5-server error：服务端错误代表服务器未能响应请求。

响应码被设计成可扩展的：在客户端或者服务端错误中，如果某个端不能识别具体响应码，这个端会将之视为一般响应码（即4.00或5.00）。然而，对于success类，如果某个端不能识别具体响应码，那么这个端只能判断这个请求成功，而不能有更多的操作。在5.9节对响应码的做了详细的描述。响应能被通过多种方式发送，在接下来的小节中有定义。

5.2.1 附带响应

在大多数案例中，响应直接携带在ACK中（如果这个请求是CON)。这称为附带响应。

无论响应显示是成功还是失败，响应通过ACK消息回传。实际上，响应是附带在ACK中，不需要单独发送一个响应消息。

注意：规范将是否在ACK消息附带响应的决定权留给服务端。客户端必须同时做好接收这两种的准备。从实现层面考虑，强烈建议服务端只要有可能就采用附带响应——这种方式能节省客户端和服务端的网络资源。

5.2.2 单独响应

不是所有的消息都能将响应附带在ACK中回传。举个例子，一个服务端可能需要一段时间(比ACK超时时间更长)来获取资源表现，因此不需要冒险让客户端反复的重传请求信息（参见4.8.2节中PROCESSING_DELAY的讨论）。当请求是携带在NON中时，响应总是和ACK分离的（因为NON不需要ACK）。

服务端产生单独响应的时机是，服务端在获得资源表现过程中，ACK定时器超时。如果服务端提前知道不会有附带响应，服务端也可能马上发一个ACK。在这两个情况中，ACK都表示请求将马上执行。

当服务器最终获得了资源表现后，它发送这个响应。服务器希望消息不丢失，它会选择一个新的Message ID，发送CON到客户端，并需要客户端回复一个ACK（可能也会发送NON，参见5.2.3）。

当服务端选择使用单独响应，它会发送空的ACK给CON请求。只要服务端回复了空的ACK，即使客户端重传另一个同样的请求，它也不能再回复附带响应。如果收到一个重传的请求（可能是因为原有的ACK延迟了），服务端发送另一个空的ACK，所有的响应都必须发送单独响应。

如果这个服务端接着发了一个CON的单独响应，客户端回复这个响应的ACK也必须是空的信息（不携带请求和响应的消息）。收到ACK后，服务端在所有匹配的ACK(忽略任何响应码和payload)和RST消息中停止重传该响应。

实现注意：由于底层数据报传输可能是无序的，因此，单独响应的CON消息可能在请求的空ACK前到达客户端。为了避免重传，这个CON消息也作为ACK来处理。也要注意，虽然CoAP协议本身不做任何特殊的要求，但应用可能期望响应会在一个时间期限内到达。因为底层传输没有keep-alive机制，请求者可能要建立一个超时时间（与CoAP重传时间无关）以防服务端出问题或者不能响应。

5.2.3 无需确认消息NON

如果是NON请求，响应也应该通过NON回复。然而，端必须在发送CON请求时准备好接收一个NON响应（领先或者落后于一个空的ACK），在发送NON请求时准备好接收一个CON响应。

5.3 请求响应匹配

不管响应是怎么被发送的，响应和请求依靠包含在客户端的请求中的令牌（token），以及额外的相关端地址信息来匹配。

5.3.1 令牌（token）

token是用于匹配响应与请求的。token的值有0~8字节（注意，每个信息都携带token，即使其长度为零）。每个请求都携带由客户端生成的token，服务端在响应时必须复制（不能修改）这个token。

token用作client-local标示，用于区分并发请求（参见5.3节），也称为“request ID”。

客户端生成token时需要注意，当前使用的token对给定的源端和目的端应该都是独一无二的。（注意客户端在生成token时，如果要向不同的端（比如源端口号不同）中发送请求，可以使用同样的token）。当只向目的端产生一个token，或者向每个目的端发送的请求都是顺序的，且都是附带响应，token为空也是可行的。有多种策略实现。

如果客户端不使用传输层安全(TLS，见第9章）发送请求，就需要使用复杂的，随机的token来防止欺诈响应（见11.4节），起到保护功能，这也是token允许使用最多8个字节的原因。token中随机组件的实际长度取决于客户端的安全需求和欺诈响应造成的威胁程度。接入到互联网的客户端至少应该使用32位随机码，记住，没有直接连接互联网也不一定能有效防范欺诈。注意，Message ID几乎没有添加保护，因为它通常是顺序分配的，因此可能被猜测到，并通过欺诈响应绕过。客户端想要优化token长度，可能会向进一步检测正在进行的攻击等级（例如计算接收的token不匹配的消息个数）。[RFC4086]讨论对安全的随机性要求。

端接收一个不是它生成的token，必须把这个token当做不透明的，不能假设它的内容和结构。

5.3.2 请求/响应匹配规则

确切的匹配响应与请求的规则如下：

1. 响应的源端必须和原始请求的目的端一致。
2. 在附带响应中，CON请求和ACK的“Message ID”必须匹配，响应和原始请求的“token”必须匹配。在单独响应中，只需响应和原始请求的“token”匹配。万一信息携带异常的响应（不是认定的端，端地址、token和客户端的期望不匹配），这个响应必须被拒绝（见4.2和4.3）。

注意：客户端接收到CON响应之后，可能想在回复完ACK马上清除这个消息的状态。如果这个ACK丢失，且服务端重传这个CON消息，客户端可能不会再有任何与该响应相关联的状态，会导致这个重传成为异常消息；客户端可能会发送RST信息，这样它就不会再收到更多

的重传消息。这个行为是正常的，并不是一个错误（没有积极优化内存使用状态的客户端会将第二个CON认定为重发。客户端事实上期望从服务器[observe]得到更多消息，就必须在任何情况下保持状态）。

5.4 选项

请求和响应可能包含一个或多个option的列表。举个例子，请求消息里的URI是都在多个option中传输，在HTTP中元数据可能会携带在HTTP头部，也是作为option来提供的。

CoAP定义了一组用于请求和响应的选项。

- Content-Format
- ETag
- Location-Path
- Location-Query
- Max-Age
- Proxy-Uri
- Proxy-Scheme
- Uri-Host
- Uri-Path
- Uri-Port
- Uri-Query
- Accept
- If-Match
- If-None-Match
- Size1

这些选项的语义和他们相应的属性都在5.10节有详细定义。

并不是所有的选项都被定义可以使用所有方法和响应代码。方法和响应码的可能的选项都各自被定义在5.8和5.9节。如果一个选项没有定义方法或响应代码，，那它就禁止包含在发送内容里，并且必须被接收端当作未识别的选项。

5.4.1 重要选项/非重要选项Critical/Elective

选项分两个种类：重要“critical”或者非重要“elective”。这两者的不同之处是端如何处理一个不能识别的option。

- 根据接收情况，不能识别的非重要option必须忽略。
- 不能识别的非重要option出现在一个CON请求中，必须返回4.02（Bad Option）的响应。这个响应应该包含一个诊断的payload来描述这个不能识别的option（见5.5.2节）。
- 不能识别的重要option出现在一个CON响应中或者附带响应的ACK中，必须拒绝这个响应（4.2节）。
- 不能识别的“critical”option出现在一个NON消息中，必须拒绝这个消息。

注意，无论重要还是非重要，option永远不会强制(总是可选)：这些规则是为了实现停止处理它们没有理解的或没有执行的option。

重要/非重要规则不适用于代理。代理处理option基于Unsafe/Safe-to-Forward (定义于5.7节)。

5.4.2 Proxy Unsafe or Safe-to-Forward and NoCacheKey

一个option除了被标注为重要或者不重要之外，option同样会基于代理如何处理不能识别的option来分类。为此，option可以被视为unsafe to forward（unsafe标识被设置），或者safe-to-foreword（unsafe标识被清除）。

此外，在请求中对于被标记为safe-to-forward的option，option编号表明它是否成为cache-key（见5.6节）的一部分。只要有一位NoCacheKey bits是0，它就是cache-key的一部分；如果所有的NoCacheKey bits是1，它才不是（见5.4.6节）。

注意：Cache-Key只和依赖于Unsafe/Safe-to-Forward指示，而不是将给定的option执行为请求option的代理相关。举个例子，ETag，使用请求选项作为Cache-Key的一部分实际上是非常低效的，但如果ETag没有被代理执行，这就是你所能做的最好的事，因为响应将根据请求选择而变化。一个更有用的代理，不用执行ETag请求选项，就是不使用ETag作为Cache-Key的一部分。 NoCacheKey以3-bit表示，所以八个响应码只有一个是NoCacheKey，剩下七个响应码代表其他情况。与这些分类相关的代理行为在5.7节有定义。与这些分类相关的代理在5.7节中有定义。

5.4.3 长度

选项值定义有一个特定长度，通常的形式是一个上界和下界。如果在请求中option的长度值超过定义的范围，这个option必须当做一个不能识别的option处理（见5.4.1节）。

5.4.4 默认值

Option可能有被定义一个默认的值。如果一个option的值是默认值，这个option就不应该包含在消息中，如果这个option不存在，必须假定为默认值。

当一个重要option有一个默认值时，就会以这种方式被选择，消息里的option空缺可以被两种执行合理地处理，一种是没有意识到重要option的执行，另一种是将空缺解释为存在option默认值的执行。

5.4.5 可重复选项

一些option的定义指定这些option是可重复的。一个消息中可能包含一个或者几个可重复的option。一个不可重复的option，在一个消息中绝对不能出现超过一次。

如果消息里的option出现的次数比定义的option多，随后出现的多余option必须当做无法识别的option(见5.4.1节)。

5.4.6 选项编号

一个option由一个option编号所定义，这个编号也能提供一些额外的语义信息，比如，奇数编号代表一个重要option，偶数编号代表非重要option。注意，这不只是一个约定，也是协议的一个功能：option是否重要，取决于这个option编号是奇数还是偶数。

更通俗点说，一个option编号由一位掩码来确定这个option是否重要，是unsafe还是safe-to-forward，还有，如果是safe-to-forward，还提供一个cache-key，如下图所示。在下文中，这位掩码被表示成为一个字节，当LSB格式时，option编号表示成无符号整数。当bit7（最低有效位）是1，option是重要（同样，如果是0的话为非重要）。当bit6是1，option是unsafe（同样，如果是0的话为safe-to-forward）。当bit6是0的时候，也就是这个option不是unsafe。当且仅当bit 3-5全都设置为1的时候它不是一个cache-key（NoCacheKey），所有其他的bit的组合，代表它确实是一个Cache-Key。这些options的分类在下面章节中有阐述。

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|           | NoCacheKey| U | C |
+---+---+---+---+---+---+---+---+
图10：option number mask（LSB）
```

一个端可以使用一段等价的C代码（如图11），来得到这个option的编号“onum”的特性。

```
Critical = (onum & 1);
UnSafe = (onum & 2);
NoCacheKey = ((onum & 0x1e) == 0x1c);

Figure 11: 选项编号的确定特性
```

Option的编号都定义在CoAP选项编号文档中（见12.2节）。

5.5 Payload和表现

请求和响应依赖于各自的方法码或者返回码，因而可以包含payload。如果一个方法码或者返回码没有定义为包含payload，那么发送端一定不能包含payload，而且接收端必须要忽略它。

5.5.1 表现

请求或者响应的payload标明了成功是资源的表现（“resource representation”）或者请求动作的结果（“action result”）。它的格式由互联网媒体类型指定，内容编码由Content-Format选项决定。当缺乏这个选项时，没有假定的默认值，那么格式将由应用来决定（比如从应用的内容上）。如果没有指定的内容类型，才会尝试使用探测payload。

实现注意：在执行质量层面，强烈建议由资源表现来提供Content-Format的显示。因为这不是协议的要求，所以并不是一个强制要求，同时也很难概述出究竟在什么情况下会违反这种建议。

对于标示客户端或者服务端错误的响应，当且仅当内容格式(Content-Format)选项给出时，payload被认为是请求动作的结果的表示。当没有该option时，payload是诊断式的(见5.5.2)。

5.5.2 诊断式的payload

如果没有内容格式(Content-Format)的选项，响应中的payload使用简洁可读的诊断信息来指示一个客户端错误或者服务端错误，并解释错误的情况。该诊断信息一定要使用UTF-8格式编码[RFC3629]，更确切的说是使用Net-Unicode格式[RFC5198]。

这种消息和HTTP的状态行中的原因描述(Reason-Phrase)很类似。它不是为终端用户设计的，而是为软件工程师设计的，因为调试过程中需要在当前的上下文中用符合英语语言规范来解释它；因此，不需要提供语言标记的机制。与HTTP不同的是，如果在返回码(Response Code)外没有额外的信息，负载必须是空的。

5.5.3 经由选择的表现

不是所有的携带payload的响应都提供由请求寻址的资源的表现。然而，能够参考与响应相关的表示有时也是有用的，不依赖于它实际上是否封闭独立。

我们使用术语"selected representation"来引用被一个成功的响应选中的目标源的当前表示，如果这个相关的请求已经使用了GET方法，并且不包括任何有条件的请求选项。(5.10.8节)

已确定的响应选项提供关于选择表示的元数据，它可能不同于响应一些包含状态改变的方法的消息的表现。在该篇规范中定义的响应选项，只有ETag响应选项（5.10.6节）定义为关于选择表示的元数据。

5.5.4 内容协商

服务器可能会以多种表示格式来提供资源表示。如若没有来自客户端的更多信息，服务端会提供它偏爱的格式。

通过使用请求中的接收选项(5.10.4节)，客户端能够标示它偏爱接收的内容格式。

5.6 缓存

CoAP端点为了减少响应时间和网络带宽消耗，它可以缓存响应。

CoAP中缓存的目标是通过重利用先前的响应消息来应答当前的请求。在某些情况下：甚至无需网络请求，就能够重利用已经存储的响应，从而减少延时和网络回传；一种名为"freshness"的机制用于这个目标(参考5.6.1节)。甚至当有一个新请求时，通常可以重利用先前响应的payload来应答该请求，从而减少了网络带宽的消耗；一种名为"validation"的机制用于这个目标(参考5.6.2节)。

与HTTP不同，CoAP响应的缓存能力并不依赖于请求方法，而是依赖于返回码(Response Code)。在5.9节中返回码定义中列出了每种返回码代表的缓存能力。端点标示成功和无法识别的响应码必须不能被缓存。

对于已提出的请求，CoAP端点一定不能使用已存储的响应，除非：

- 已提出的请求方法和用于获取存储响应的方法相匹配
- 已提出的请求和那些用于获取存储回复的请求(包含请求URI)的所有选项相匹配，除了不需要匹配标记为NoCacheKey的任何请求选项(Section 5.4)或者能被缓存识别并且相对于缓存行为能够完全解释的选项匹配(比如在5.10.6中描述的ETag请求选项，也可以参考5.4.2)
- 已存储的回复是按照下面将要定义的更新或者成功验证。

用于匹配缓存入口的请求选项族可以全体称为“Cache-Key“。比起coap和coaps，在URI格式中，构成请求URI的选项匹配可以在URI格式下的特定规则中执行。

5.6.1 新鲜度模型(Freshness Model)

当缓存中的回复是“新鲜”的，不用联系原始服务端就能用于应答请求，因此提高了效率。

对起点服务器的测定新鲜度的机制通过使用Max-Age选项(5.10.5节)在未来提供一个明确的到期时间。Max-Age选项的意思是当响应的时间超过了指定的秒数后就被认定为“不新鲜”的。

Max-Age选项的默认值是60。所以，如果响应不在一个可缓存的响应中，那么在60秒后该响应就被认定是不新鲜的。如果最初的服务器希望禁用缓存，它就必须将Max-Age选项的值指定为0秒。

如果客户端有一个新鲜的已存储的响应，并且为已存储的响应生产了一个匹配的新请求，新的响应就会使得旧的响应失效。

5.6.2 校验模型

当端点对一个GET请求有一个或多个存储的响应，但又不能使用其中任意一个时(例如它们都不是新鲜的)，它能够使用GET请求中的ETag选项（5.10.6）给原始服务端一个选择存储的响应并且更新它的新鲜度的机会。这个过程称为验证或者重验证已经存储的响应。

当发送一个这样的请求，端点应当为每个适当的存储响应添加一个ETag选项来指定它们的entity-tag。

按照5.9.1.3中的描述，携带2.03(Valid)响应码的响应中，ETag选项中的entity-tag所标识的已存储响应，可以在完成更新后重新使用。

其它任何响应码都表明请求中的已存储响应都不适用。相反，响应应当用于应答请求并替代已存储的响应。

5.7 代理

代理是能够代表CoAP客户端执行请求的CoAP端。当客户端不能生成请求，或者需要减少响应时间、降低网络带宽或者功耗，因此需要cache响应时，代理相当有用。

在受限的RESTful环境中的整体架构中，代理可以实现完全不同的目的。客户端可以明确地选择代理，我们称为正向代理。代理也可以被插入来代替原始服务器，我们称为“反向代理”。从CoAP请求映射到CoAP请求(CoAP-to-CoAP)的代理或者转换不同的协议(跨协议)的代理，可以和正向代理、反向代理互相组合。在1.2中有这些术语的完整定义。

注意：这篇规范中的术语与互联网应用环境中的术语是兼容的，在各项细节中无需匹配它(甚至都与受限的RESTful环境无关)。没有太多的语义应该归属于术语的成分(例如正向，反向或者跨协议)。

HTTP代理，除了作为HTTP代理，通过提供传输层协议的代理功能来保证端对端通信的传输层安全。这篇规范中并没有在CoAP-to-CoAP的代理中定义这样的功能，因为在受限的RESTful环境中转发UDP包看起来没太多价值。可以参考10.2.7中的跨协议代理例子。

当客户端使用代理提出请求，会使用一个安全的URI方案(例如“coaps”或“https”)，只要在客户端和代理之间没有使用等效的底层安全机制，那么发往代理的请求必须使用DTLS。

5.7.1 代理操作

根据从客户端接收到的请求，代理通常需要一种为到目的端的请求分配可能的请求参数的方法。该方法完全由正向代理指定，但是也可以依赖于反向代理的特定配置。特别是，反向代理的客户端通常没有标示目的端的定位器，因此有必要在反向代理中有命名空间转换的格式。然而，代理操作的一些方面对于其各种形式是常见的。

如果代理没有使用缓存，那么它仅仅简单的往指定的目的端转发请求。否则，如果代理使用缓存但是没有与转化的请求相匹配的，且已存储的新鲜的响应，那么根据5.6节它需要更新缓存。代理识别请求的option，它应当知道该option是否能够当作在缓存值中查询的键值的一部分。举个例子，由于对于不同Uri路径值的请求指向不同的资源，Uri路径值通常当作Cache-Key的一部分，而token值从来都不能当作Cache-Key的而一部分。对于代理没能识别的但是在选项码中标记为Safe-to-Forward的选项，选项也标示了它是否包含在Cache-Key中(NoCacheKey没有完全设定或者完全设定)。（无法识别并且标记为Unsafe的选项就是4.02Bad Option）。

如果发往目的端的请求超时了，那么必须返回一个5.04（GateWay Timeout）响应。如果发往目的端的请求返回一个无法被代理处理的响应(比如无法识别的关键选项或者消息格式错误)，那么必须返回一个5.02（Bad Gateway）响应。否则代理向客户端返回响应。

如果响应在缓存中生成，生成的（或者隐含的）Max-Age选项一定不能超过最初由服务端设定的max-age，max-age表示在缓存中资源存活的时间。举个例子，对于每条响应可以按照下面的公式由代理计算出Max-Age选项：

```
proxy-max-age = original-max-age - cache-age
```

举个例子，如果代理资源在20秒之前更新，并且最初的Max-Age是60秒，那么代理中该资源的max-age就是40秒。考虑到初始服务器的潜在网络延时，代理生成的响应的max-age值最好比该值偏小。

出现在代理请求里的所有选项都必须被处理。请求中无法被代理识别的Unsafe选项一定会导致代理返回一个4.02（Bad Option）响应。CoAP-to-CoAP代理必须往原始服务端转发不能识别的所有Safe-to-Forward选项。类似的，在响应中无法被CoAP-to-CoAP代理服务端识别的Unsafe选项会导致一个5.02(Bad Gateway)的响应。此外，不被识别的Safe-to-Forward选项必须被转发。

在第十章中详细讨论CoAP和HTTP的跨协议代理。

5.7.2 正向代理

CoAP区分原始服务端的请求和正向代理的请求。CoAP向正向代理的发出请求是普通的CON和NON请求，但是它们用不同的方式来指定请求URI：代理请求中的请求URI指定为Proxy-Uri选项中的字符串（参考5.10.2），而原始服务端中的请求分为Uri-Host，Uri-Port，Uri-Path和Uri-Query选项（参考5.10.1）。作为另一种选择，代理请求中的URI也可以由Proxy-Scheme选项和刚才提到的分开的选项组合而成。

当端点收到一个代理请求，而端点不想或者不能当作代理来处理这个请求URI，那么它必须返回一个5.05（Proxying Not Supported）的响应。如果授权(host and port)当作代理端点本身（5.10.2），那么请求必须当作一个本地的（non-proxied）请求。

一般来说，代理必须按照下面来解释请求：请求URI的设计定义输出的协议和它的细节(例如coap设计编码CoAP是在UDP之上的，而coaps设计编码是在DTLS之上的)。对于一个CoAP-to-CoAP的代理，初始服务端的IP地址和端口是由请求URI的授权分量决定的，请求URI可以被解码分为Uri-Host，Uri-Port，Uri-Path和Uri-Query选项。如果代理被配置为将代理请求转发到另一个代理，上面的解释方法会使其占用Proxy-Uri或Proxy-Scheme选项，导致其无法转发到原始服务端。

5.7.3 反向代理

反向代理不会利用Proxy-Uri或者Proxy-Scheme选项，但是需要从请求信息和配置信息中确定请求的目的端（下一跳）。例如，反向代理在通过资源侦测获知资源的存在后，反向代理能提供各种资源，好像它们就是自己的资源一样。反向代理可以自由的为识别这些资源的URIs

建立命名空间。

反向代理也可以构建一个命名空间，可以让客户端更好的控制请求路径，比如，将主机标识符和端口编号嵌入到资源的URI路径。

在响应处理中，反向代理必须谨慎处理：不同资源的ETag选项值不能和客户端提供的资源混合起来。在很多情况下，ETag能够在不修改的条件下被转发。如果从反向代理提供的资源到各种初始服务器提供的资源的映射不是唯一的，反向代理需要生成一个新的ETag，来保证该选项的语义是正确的。

5.8 方法定义

在本小节中，每个方法及其行为都有定义。带有无法识别或者不支持的方法码的请求必须生成一个4.05(Method Not Allowed)的附带响应。

5.8.1 GET

GET方法根据请求URI定位资源，从相符合的信息中获取对应的表现。如果请求包含Accept选项，就表明了首选的响应内容格式。如果请求包含ETag选项，GET方法要求验证ETag并且只有当验证失败的时候才会传输表现。当验证成功，响应中应当包含2.05（Content）或者2.03（Valid）的响应码。

GET方法是安全而且幂等的。

5.8.2 POST

POST方法要求处理包含在请求中的表现。POST方法执行的实际功能由原始服务端决定，并依赖于目标资源。它通常的结果是创建新资源或者更新目标资源。

如果服务端响应请求，创建了资源，那么返回的响应码为2.01（Created），并应当在一系列Location-Path和/或Location-Query选项（5.10.7）中包含新资源的URI。如果POST方法成功但不是由服务端创建新资源引起的，那么响应中应当含有2.04（已修改）的响应码。如果POST方法成功并且是由删除目标资源引起的，那么响应中应当包含2.02（已删除）的响应码。

POST方法既不安全也不幂等。

5.8.3 PUT

PUT方法要求更新或创建由请求URI定位的资源。表现的格式由媒体类型和Content-Format选项中的内容编码制定。

如果请求URI中的资源已存在，那么封闭表现应当看做是该资源修改后的版本，并且应当返回一个2.04（Changed）的响应码。如果没有对应的资源，那么服务端可能根据URI创建一个新的资源，返回一个2.01（Created）的响应码。如果既不能创建资源，又不能修改资源，那么应当发送一个恰当的错误响应码。

在If-Match（5.10.8.1）或If-None-Match（5.10.8.2）中包含更多对于PUT方法的限制规定。

PUT方法不安全但是幂等的。

5.8.4 DELETE

DELETE方法要求删除由请求URI定位的资源。如果成功或者该资源不存在，必须返回2.02（已删除）的响应码。

DELETE方法不安全但是幂等的。

5.9 返回码定义

在下面将要说明的返回码中，包括了所有请求可能导致的响应。同时，某些地方会说明该返回码对应的HTTP[RFC2616]中的返回码，但这并不意味着会修改第10章中CoAP和HTTP的映射。

5.9.1 成功 2.xx

这类返回码表明客户端的请求被成功的接收、理解并接受。

2.01 Created

类似HTTP的返回码201（“Created”），但该返回码仅用于对POST和PUT请求的响应。响应包中如果同时带有Payload，则表明了服务端处理的结果。

如果响应包中包含一个或多个Location-Path 与/或 Location-Query的选项，这些选项的值说明这个被创建资源的路径。否则，这个资源就在请求的URI下创建。一个缓存端如果收到这个响应必须将所有这个被创建资源的响应缓存标记为未刷新。

这种响应不能被缓存。

2.02 Deleted

这个响应码和HTTP的响应码204（“No Content”）类似，但仅仅用于响应那些导致该资源无效的请求，如DELETE，或者某些情况下的POST。响应包中如果同时带有Payload，表明了服务端处理的结果。

这个响应不能被缓存，一个缓存端必须将所有为这个被删除资源的响应缓存标记为未刷新。

2.03 Valid

这个响应码和HTTP的响应码304（“Not Modified”）相关，但仅用于当携带了ETag选项时，说明（请求）中的entity tag是合法的。相应的，响应也必须携带ETag选项且必须不能包含payload。

如果一个缓存端能够认出并且处理携带ETag option的响应报文，当它收到2.03响应时，它必须将其缓存的响应中的Max-Age option的值修改为和该响应的值一致（可能为明确的值或者是一个默认值，参见5.6.2节）。对于响应中每个转发安全选项，所有缓存的响应中的对应选项中的值应该更新为该响应的值。非安全选项可能会触发类似的选项自定义的特有的操作。

2.04 Changed

这个响应码类似HTTP的响应码204（“No Content”），但仅用于POST和PUT请求的响应。如果有的话，Payload和响应一起返回，携带该操作的结果。

这个响应是不能被缓存的。因此，一个缓存端必须将针对这个被改变资源的缓存响应标记为未刷新。

2.05 Content

这个响应码类似HTTP的响应码200（“OK”），但仅用于GET请求的响应。

Payload和响应一起返回，携带目标资源的内容。

这个响应是可以被缓存的，缓存端能够使用Max-Age选项来决定刷新时间（参见5.6.1节），如果有ETag选项，也可以将其用于数据确认（是否有更新）的检查。

5.9.2 客户端错误 4.xx

这个响应码集合里包含了客户端一些可能出错的情况。这些响应码适用于所有请求方法。

服务端在如5.5.2节的情况下，应该返回一个包含诊断信息（diagnostic）的payload。

这个集合里的响应是可以被缓存的，缓存端能够使用Max-Age选项来决定刷新时间（参见5.6.1节），但不能用于确认检查。

4.00 Bad Request

这个响应码类似HTTP的响应码400（“Bad Request”）。

4.01 Unauthorized

客户端并未得到执行该操作的授权。客户端不应该重复发送这个请求，除非改变了自己的授权状态。当然，有一些特殊的机制在这种情况下能被使用（超出了本文档的范围），参见第9章。

4.02 Bad Option

这个请求里面的一个或者更多的选项不能被服务端理解，或者有错误。客户端不应该重复发送这个请求，直到修改那些选项。

4.03 Forbidden

这个响应码类似HTTP的响应码403（“Forbidden”）。

4.04 Not Found

这个响应码类似HTTP的响应码404（“Not Found”）。

4.05 Method Not Allowed

这个响应码类似HTTP的响应码405（“Method Not Allowed”），但并没有类似的Allow头部。

4.06 Not Acceptable

这个响应码类似HTTP的响应码406（“Not Acceptable”），但没有响应实体。

4.12 Precondition Failed

这个响应码类似HTTP的响应码412（“Precondition Failed”）。

4.13 Request Entity Too Large

这个响应码类似HTTP的响应码413（“Request Entity Too Large”）。

除非服务端无法提供其最大接收的数据大小，否则应该提供一个携带Size1选项（参见5.10.9章）的响应。

4.15 Unsupported Content-Format

这个响应码类似HTTP的响应码415（“Unsupported Media Type”）。

5.9.3 服务端错误 5.xx

这个响应码集合里包含的异常情况为：服务端出错或者无法处理这个请求。这些响应码适用于所有请求方法。

服务端在如5.5.2节的情况下，应该返回一个包含诊断信息（diagnostic）的payload。

这个集合里的响应是可以被缓存的，缓存端能够使用Max-Age选项来决定刷新时间（参见5.6.1节），但不能用于确认检查。

5.00 Internal Server Error

这个响应码类似HTTP的响应码500（“Internal Server Error”）。

5.01 Not Implemented

这个响应码类似HTTP的响应码501（“Not Implemented”）。

5.02 Bad Gateway

这个响应码类似HTTP的响应码502（“Bad Gateway”）。

5.03 Service Unavailable

这个响应码类似HTTP的响应码503（“Service Unavailable”），但是用Max-Age选项取代了HTTP的“Retry-After”头部，用来表明下次重试需要等待多少秒。

5.04 Gateway Timeout

这个响应码类似HTTP的响应码504（“Gateway Timeout”）。

5.05 Proxying Not Supported

请求中包含Proxy-Uri选项或者使用Proxy-Scheme（参见5.10.2节），但是服务端无法（或者不愿意）为其指定的URI做正向代理。

5.1 Option定义

CoAP的那些选项在表5-1中总结，并且在后面的子章节中描述。

在表中，C，U和N这几列分别代表临界（Critical），不安全（UnSafe）和不缓存（NoCacheKey）。由于NoCacheKey仅仅用于安全转发（Safe-to-Forward）选项（没有标记为Unsafe），这一列被填满了横杠。

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
17	x				Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)
60			x		Size1	uint	0-4	(none)

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

表4 选项表

5.10.1 Uri-Host，Uri-Port，Uri-Path，Uri-Query

Uri-Host，Uri-Port，Uri-Path，Uri-Query选项都用来定位一个向原始服务端请求的目标资源。选项将请求URI的不同组件进行编码，在这些选项的value值中，看不到百分号编码（也称URL编码），且完整的URI能够在任何有关的端被重建。CoAP URI的语义在第六章定义。

将URI解析成选项的步骤定义在6.4节。这些步骤是由请求中的0个或者多个Uri-Host，Uri-Port，Uri-Path，Uri-Query选项决定，每个选项包含以下信息：

- Uri-Host选项定义了被请求资源的网络主机；
- Uri-Port选项定义了资源在传输层的端口号；
- 每个Uri-Path选项定义了一段资源的绝对路径；
- 每个Uri-Query选项定义了一个资源的参数。

注意：Fragments（[RFC3986]，见3.5节）并不是URI请求的一部分，因此在CoAP请求中不会被传输。

Uri-Host选项的默认值是请求消息的目的IP地址。同样地，Uri-Host选项的默认值即UDP的端口号。Uri-Host和Uri-Port选项的默认值对于大部分服务端来说已经足够了。当一个端有多个虚拟服务端时才会使用明确的Uri-Host和Uri-Port选项。

Uri-Path和Uri-Query选项能够被任何字符顺序编码。不会采用百分号编码。Uri-Path选项的值必须不可以是“.”或者“..”（因此URI请求必须在将填充选项前将它们解析替换掉）。

例子可以在附录B找到。

5.10.2 Proxy-Uri和Proxy-Scheme

Proxy-Uri选项被用于生成一个向正向代理（5.7节）发送的请求。用来请求正向代理将该请求转发给服务端或者从合法的缓存中返回一个响应。

该选项的值是一个绝对URI（[RFC3986]，4.3节）。

注意，正向代理可能将该请求转发给另一个代理，或者直接给服务端，由absolute-URI决定。为了避免请求循环，代理必须能够识别所有它面对的服务端名字，包括任何别名，局部变化，以及IP地址。

一个Proxy-Uri选项优先于任何Uri-Host、Uri-Port、Uri-Path或者Uri-Query选项（它们每一个都必须不能出现在一个包含Proxy-Uri选项的请求中）。

作为一个简化一些代理客户端的特例，绝对URI可以从Uri-*选项中构建。当Proxy-Scheme选项存在，绝对URI能够被构造如下：一个CoAP的URI按照6.5节的定义被构建。在所得的URI中，初始的scheme的冒号之后，会被Proxy-Scheme选项的内容替换。注意，这只适用于在所需的URI组件中，除了scheme组件，其他要素实际上可以使用Uri-*选项表示的情况。例如，通过一个userinfo组件来认证URI的话，只能使用Proxy-Uri组件。

5.10.3 Content-Format

Content-Format选项即消息的payload段。Content-Format的不同格式通过在12.3节中的CoAP Content-Format表中定义，通过数字来索引。该选项不存在默认值，即任何无该选项的payload都是未定义的（见5.5节）。

5.10.4 Accept

CoAP的Accept选项用于表明哪些Content-Format能够被客户端接受。它的表示格式同样也是在12.3节中的CoAP Content-Format表中定义。如果没有Accept选项，表明客户端可以接收所有格式（也即该选项没有默认值）。客户端接受服务端返回它指定的格式。如果服务端无法

提供客户端指定的格式，服务端必须返回一个4.06“Not Acceptable”，除非另外的错误码比这个返回码优先。

5.10.5 Max-Age

Max-Age选项定义了一个响应在它被标记为未刷新前，最多能缓存的时间（见5.6.1）。

该选项的值是一个整型的秒数，从0到2**32-1（大约136.1年）。如响应中没有定义这个选项，它的默认值是60秒。

这个值是传输时开始计时的，那些对Max-Age时间要求严格的服务器必须在每次重传前更新这个值（见5.7.1）。

5.10.6 ETag

实体标识作为本地资源标识符，用来区分该资源是否已经随着时间推移而变化了。它由提供资源的服务端生成，可能通过版本、checksum、哈希或者时间等方式来生成。一个收到实体标识的端必须将该标识视为不透明的且不能假定它的内容或结构。生成实体标识的端被提倡使用尽量压缩的表达，因为客户端和中间人可能保存多个ETag值。

作为一个响应选项的**ETag**

响应中的ETag选项提供了它的当前值（例如，当一个请求被处理后），作为代表标签（“tagged representation”）。如果没有Location- 选项存在，代表标签就是目标资源的选定表现*(5.5.3节)*。如果一个或者多个*Location-*\选项存在，且因此一个URI地址被指明，代表标签表示客户端需要通过给定的URI地址重新获取资源。

一个ETag选项能够包含在任何带有代表标签的响应中（当然，如果在4.04或者4.00回复中，没什么意义）。ETag选项必须不能在一次响应中包含多次。

ETag选项没有默认值，如果它不存在响应中，服务端就没有代表标签的实体标识。

作为请求选项的**ETag**

在GET请求中，一个端如果已经从资源中获取一个或者多个表现，且从响应获得了它们的ETag，就能够为一个或者多个这些已保存的响应指定ETag选项。

服务器可以发出2.03有效响应(5.9.1.3节)代替2.05内容响应，如果给定的etags中的一个当前表现为实体标记，即是有效的；然后2.03有效响应在响应选项里回应这个特定的ETag。

事实上，客户端能够确定目前所存储的表现是不是最新的（见5.6.2节），而不需要重新将他们传输。

ETag选项可能在一个请求中发生0次，1次或者多次。

5.10.7 Location-Path和Location-Query

Location-Path和Location-Query选项定义由一个绝对路径、一个请求字符串，或者二者一起组成的相对URI。

这些选项的组合包含在2.01（Created）响应中，来定位这个被创建资源的位置，而这个响应是由POST请求（5.8.2节）引起的。这个地址是由请求URI分解得出。

如果响应携带一个或者多个Location-Path和/或Location-Query选项，该响应通过了一个解释这些选项的缓存，且其中的某些响应缓存隐含了这个URI，这些响应缓存必须被标记为未刷新。

每个Location-Path选项对应该资源绝对路径的一个段，每个Location-Query选项对应该资源的一个参数。Location-Path和Location-Query选项能够包含任何字符顺序。不执行百分号编码。Location-Path选项的值必须不能是“.”或“..”。

从选项中构造一个URI地址可以参考6.5节，跳过前5步，且该结果是一个在请求URI基础上的相对URI引用。注意一个通过这种方式构造的相对URI引用经常会包含一个绝对路径，例如，不携带Location-Path但是提供Location-Query意味着在这个URI中从“/”开始。

用来计算相对URI引用的选项统一被称为Location-*。不仅是Location-Path和Location-Query，更多的Location-*选项可能在未来被定义，且已经在option的数字标记中替它们保留了128,132,136和140。如果这些这些保留的选项编号出现在Location-Path和/或Location-Query之外且不被支持，必须返回4.02（Bad Option）。

5.10.8 条件请求选项

条件请求选项允许客户端通知服务端，当这些在选项中包含的条件被满足时才执行请求。

对于每个条件请求选项，如果给定的条件没有满足，服务端必须不能执行它所请求的方法，而必须返回4.12（Precondition Failed）。

如果条件被满足，服务端执行它请求的方法，和条件请求选项不存在时相同。

如果一个请求可能没有条件请求选项，收到除了2.xx或者4.xx的响应码，那么任何条件请求选项可能被忽略。

If-Match

If-Match选项可能用来为当前实体或者为一个或多个目标资源的表现的ETag值生成一个条件请求。If-Match在资源更新的请求上很有用，比如PUT请求，用于保护多个客户端在同一资源下进行类似操作时意外覆盖（比如“lost update”问题）。

If-Match选项的值是一个ETag或者是一个空白字符串。一个带有ETa的If-Match选项需要和对应表现的ETag完全相同。值为空的If-Match选项和所有存在的表现匹配（如，把目标资源任何当前表现的实体作为前置）。

If-Match选项可能多次出现。如果任意一个条件匹配，那么这个条件成立。

如果存在一个或多个If-Match选项，但没有一个选项匹配，那么这个条件不成立。

If-None-Match

If-None-Match选项可能被用于当目标资源不存在时，生成一个条件请求。If-None-Match对于创建请求，比如PUT请求来说很有用，能够保护多个客户端在同一资源下进行类似操作时意外覆盖。If-None-Match选项没有value值。

如果目标资源存在，这个条件不成立。

注意，将If-Match和If-None-Match选项放在一个请求中不太好，因为这会导致条件永不成立。

5.10.9 Size1选项

Size1选项提供在一个请求中资源表现的长度信息。选项的值是一个整型，表示字节数。它主要用于块传输[BLOCK]。在目前标准中，它用于响应码码4.13（参见5.9.2.9），定义服务端能够处理的请求实体的最大长度。

第六章 CoAP URI

CoAP中使用“coap”和“coaps”的URI scheme来标识CoAP资源和提供资源定位。CoAP资源由潜在的在某个UDP端口监听CoAP请求（“coap”）或DTLS加密的CoAP请求（“coaps”）的CoAP源服务器组织和分层。CoAP服务器是通过通用语法中的authority组件，即host组件（即IP地址或域名）和一个可选的UDP端口号来区分的。URI中剩余部分则标识了一个能够被CoAP协议中定义的方法操作的资源。“coap”和“coaps”的URI可以分别和“http”以及“https”的URI做类比。

“coap”和“coaps” URI scheme的语法在本章被定义，采用ABNF格式（Augmented-Bacus-Naur Form）[RFC5234]。关于“host”、“port”、“path-abempty”、“query”、“segment”、“IP-literal”、“IPvaddress”和“reg-name”的定义请参考[RFC3986]。

实现注意：不幸的是，到目前为止URI格式已经非常复杂。建议开发者们仔细查看[RFC3986]。例如，IPv6地址上的ABNF就比预期的更复杂。同样的，开发者们需要小心处理URI的百分比解码或百分比编码的处理，在从一个URI和它的解码组件之间只执行一次。百分号编码对数据透明相当重要，但是处理不好可能会导致未定义的结果，例如在path组件中的斜线。

6.1 CoAP URI scheme

```
coap-URI = “coap:” “//” host [ “:” port ] path-abempty [ “?” query ]
```

如果host组件提供的是一个ip如ipv4地址（例如192.168.1.1），那么CoAP服务器可以通过该ip地址访问。如果host是一个域名，由于域名是一个间接的标识，因此端需要使用域名解析服务如DNS，来发现host的真正地址。host必须不能是空；如果收到一个authority或者host为空的URI，那么必须认为这是一个非法的URI。port组件则代表CoAP服务端可以在UDP的哪个端口被访问。如果为空或者未提供，则使用默认的5683端口。

path则在一个host和port范围内定义了资源。它由一系列被“/”（U+002F）分隔开的路径段组成。

请求用来对资源进行进一步优化。它们由“&”（U+0026）分隔。一个参数经常由“key=value”格式组成。

“coap”URI还支持路径前缀“/.well-known”，它在[RFC5785]中定义为host命名空间里的“well-known locations”。它允许对一个host的policy或其他信息（“site-wide metadata”）的发现，例如hosted资源（第7章）。

建议应用开发者们采用尽量简短而清晰的URI。由于CoAP经常使受到带宽和功耗的限制，所以应该优先考虑简短，但又不忽略清晰性。

6.2 CoAPs URI scheme

```
coaps-URI = “coaps:” “//” host [ “:” port] path-abempty [ “?” query]
```

在上一节所列举的“coap”的要求也适用于“coaps”，不同的在于，“coaps”默认端口为5684，且UDP数据报必须通过DTLS（见9.1节）加密。

“coaps”请求的缓存回复在第11.2节讨论。

在“coaps”下获取的有效资源和“coap”下不是相同的，即使“coap”和“coaps”可能采用相同的host和端口，它们命名空间相互独立，且可视为独立的原始服务器。

6.3 标准化和比较规则

由于“coap”和“coaps”符合URI通用语法，它们的URI的标准化和比较规则采用上述默认描述，符合在[RFC3986]中的算法，参见第6章。

如果端口和默认的端口一致，通常可以省略端口子组件。类似的，path组件为空等同于绝对路径“/”，因此通常做法是将path的值替换为“/”。scheme和host对大小写不敏感（即EXaMpLE等价于example），但通常采用小写；ip字段则采用[RFC5952]的方式；其他所有组件则是大小写敏感的。除了保留的字符以外，都等价于其对应的百分号编码（参见[RFC3986]，2.1节）；通常做法是不对它们编码。

举个例子，下面的几个CoAP消息的URI是等价的：

```
coap://example.com:5683/~sensors/temp.xml
coap://EXAMPLE.com/%7Esensors/temp.xml
coap://EXAMPLE.com/%7esensors/temp.xml
```

6.4 将URI解码为选项

将请求的URI解析成option的步骤如下所述。经过这些步骤后，要么生成包含0个或多个Uri-Host、Uri-Port、Uri-Path和Uri-Query选项的请求，要么会失败。

1. 如果URI不是一个绝对URI（[RFC3986]），那么解析失败。
2. 采用[RFC3986]中的解决方案来处理URL。在这一步，即使在第5、8、9步后会被解码成UTF-8[RFC3629]，URL是ASCII编码[RFC0020]。

   注意：不用关心这个URL是和谁相关的，因为我们知道此时它是一个绝对URL。

3. 如果被转化成为小写ASCII字符后的URL不含有值为“coap”或者“coaps”的scheme组件，那么解析失败。
4. 如果URL有一个fragment组件，那么解析失败。
5. 如果URL中的host组件不是以ip字符串格式组成的请求端目的ip地址，那么要包含一个Uri-Host选项，并且让选项的值和host组件的值相同，转化为ASCII小写字母，然后将所有百分号编码转化为为对应的字符。

   注意：通常来说请求的目的ip地址是从host获取，它保证了Uri-Host选项仅用于host组件，格式为reg-name。

6. 如果URL有port组件，那么将port的值解析成十进制整型，否则，port采用默认值。
7. 如果port的值和请求的目的UDP端口不一致，增加Uri-Port选项并将其值置为port的值。
8. 如果URL中的path组件为空或者只有一个“/”（U+002F），则进行下一步；否则，针对path组件中的每个段，都需要包含一个Uri-Path选项，且将选项的值（将百分号编码转换为对应的字符后）置为这个段的值（不含分隔符）。
9. 如果URL有query组件，那么，针对每个query组件中的参数，都需要包含一个Uri-Query选项，并且让选项的值（将百分号编码转换为对应的字符后）为该参数的值（不包含“？”和“&”）。

   注意，这些规则可以完全解析任何百分比编码。

第七章 发现

7.1 服务发现

作为CoAP服务端提供的发现服务的一部分，客户端需要对服务端有所了解。

客户端发现服务端，是客户端通过从URI中获取或学习服务端命名空间中的资源来做到的。另外，客户端能够使用多播CoAP（第8章）和“All CoAP Nodes”多播地址来查找CoAP服务端。

除非“coap”或者“coaps”URI中指定了服务端的UDP 端口，否则服务端默认是能够通过默认端口连接的。

提供资源发现（见7.2节）的服务端必须支持，提供其他资源的服务端应该支持CoAP默认端口号5683。对于DTLS加密类型的CoAP，提供资源发现和提供其他资源的服务端可能支持默认5684端口。此外，其他端可能采用另外的端口，例如，端口动态分配情况下。

实现注意：当一个CoAP服务端是由6LowPAN节点提供，当端口使用[RFC4944]和[RFC6282]定义的UDP端口压缩方式，采用61616-61632之间的端口号时，头部的压缩会更好。注意，如果某个服务端的UDP端口和默认端口不同，可以将它们（采用非默认端口的和采用默认端口的）视为两个不同的端。

7.2 资源发现

CoAP端提供的资源发现在M2M（machine-to-machine）应用中相当有用，因为在其中没有人的干预，而固定的接口又会导致连接的难复用性和较差的鲁棒性。为了最大化CoRE环境的互用性，一个CoAP端应该支持[RFC6690]描述的可发现资源的CoRE连接格式，除非要求完全手动配置。由服务端来决定哪些资源能够被发现（如果有的话）。

7.2.1 'ct' 特性

这节定义了一种采用[RFC6690]的新的web连接（Web Linking [RFC5988]）特性。Content-Format码"ct"特性提供了该资源会返回哪种Content-Format的提示。注意，由于这仅仅是一个暗示，并不会取代针对某个资源表现的请求的回复中的Content-Format选项。"ct"特性的值是采用CoAP编码格式的十进制整型ASCII码，而且必须在0-65535范围内。举个例子，"application/xml"被定义成"ct=41"。如果Content-Format特性不存在，那么该类型无意义。Content-Format码特性可能包含一个用空格分隔开的Content-Format码序列，表明多个content-format可用。这个特性的值的语法在下面被总结，其中"cardinal"、"SP"、"DQUOTE"在[RFC6690]中定义。

```
ct-value = cardinal
/ DQUOTE cardinal *( 1*SP cardinal ) DQUOTE
```

第八章 多播CoAP

CoAP支持在IP多播组中发送请求，这相当于连续的单播CoAP。更多关于多播组内CoAP的交互讨论在[GROUPCOMM]。

那些希望其他端能够发现的CoAP端，采用多播服务发现，加入一个或多个适当的all-CoAP-node的多播地址（12.8节）并且监听默认CoAP端口（注意，这些端可能会收到其他多播地址的多播请求，包含all-nodes的IPV6地址或者IPV4广播）。因此一个端必须做好接收这些消息的准备，但如果没有提供多播发现服务的话，可以忽略它们。

8.1 消息层

多播请求的特点是目的地址由具体的CoAP端地址变成了IP多播地址，多播请求必须是不需应答消息。

服务端应该能够识别出这些通过多播发来的请求，例如，尽量使用现有的API，如IPV6_RECVPKTINFO [RFC3542].

为了避免错误的响应，当服务端发现该请求是通过多播接收的，必须不能返回RST消息给NON。如果没有发现（为多播请求），服务端可能会返回一个RST消息给NON。由于这种RST消息看起来和发送者发送的单播消息相同，发送者必须避免使用一个可能依然在其他接收到多播消息的端中依旧使用的Message ID。在撰写本文档的时侯，多播消息只能用于不含DTLS的CoAP。因此意味着在本文档中为CoAP定义的安全模式并不适用于多播。

8.2 请求响应层

服务端发现某个多播请求，它可能会一直忽略这个请求，特别是当它没有任何有用的回复（比如说，它只有一个空的payload或者一个错误响应）。这个判断是由应用决定的（例如，在请求过滤 [RFC6690]中，当过滤条件不满足时，服务端不会响应多播请求。更多例子见[GROUPCOMM]）。

如果服务端决定响应一个多播请求，它不应该立即响应。相反它应该会等待一段时间才进行响应。我们把这段时间称为空闲（Leisure）时间。空闲时间的值可能由应用决定，也可能由下面的描述中得到。服务端应该在选取的空闲时间中的某个随机时间发送一个单播响应给该多播请求。如果再次收到同一个多播组的请求，一个新的空闲时间最早需在上一个空闲时间结束后才能开始。

为了计算出一个空闲时间，服务端必须有一个组大小估计G，一个目标数据传输率R（二者都必须谨慎的选择），一个估计的回复大小S，一个大致的空闲时间计算公式如下：

```
lb_Leisure = S * G / R
```

举个例子，一个在2.4G频段，采用IEEE 802.15.4（6LoWPAN）的本地网络中，G可能被设置成100，S为100字节，目标速率R为8kbits/s = 1kB/s。那么空闲时间为100 * 100 / 1000 = 10秒。

如果一个CoAP端不能获得足够的数据来计算空闲时间，它可能采用DEFAULT_LEISURE。

当匹配一个多播请求的回复时，仅仅token必须要匹配。回复的源端不需要（可能也不会）和原始请求的目的端匹配。

为了解释表现中的Location-*选项和任何嵌入的链接，请求URI（例如，响应对应的基本URI）通过将原始请求URI的host组件中的多播地址替换成实际响应的端的ip地址。

8.2.1 Caching

当客户端发出多播请求，它经常生成一个新的请求给多播组（因为组中可能有新成员，或者没有收到之前的请求）。它可能将接收到的回复的缓存更新。然后，它同时采用cached-still-fresh和新的响应作为请求的响应。

向一个多播组发送的GET请求的响应，可能被用于满足相关的单播请求URI的后续请求。单播请求URI中的authority部分即回复包中的传输层源地址。

通过发送一个get请求给涉及到的单播请求URI，一个响应回复可能重新生效。

向多播组发送的get请求必须不能包含ETag选项。抑制客户端已有的响应包的机制仍有待进一步研究。

8.2.2 代理

当一个正向代理收到一个带有Proxy-Uri或从Proxy-Scheme中构建的URI，且其为多播地址，代理如上所述，会获得一个响应集合，并且会发送所有的所有的响应包（包括cached-still-fresh的和新的）给原始客户端。

协议并没有提供对被修改的单播请求URI（基本URI）的返回方式，因此转发。转发多播请求在[GROUPCOMM]中有更仔细的讨论，一种定位基本URI的方案可在[COAP-MISC]第3章中找到。

第九章 安全CoAP

本章节定义CoAP中的DTLS绑定。 在配置阶段，要为CoAP设备提供它需要的安全信息，包括密钥卡片和访问控制表。本规范在9.1.3.2.1中定义了RawPublicKey模式下的配置。在配置的最后阶段，设备会处于下面四种安全模式下的一种，并带有模式的附属信息。本规范中NoSec和RawPublicKey模式都是强制执行的。

- NoSec：没有协议层的安全(禁用DTLS)。合适的情况下，应当使用其他技术来提供底层的安全机制。在[IPsec-CoAP]中讨论IPsec的使用。某些使用受限节点的链路层也提供链路层安全机制，它可能适用于合适的密钥管理。
- PreSharedKey：DTLS被启用，有一个预先共享的密钥列表[RFC4279]，而且每个密钥都包括节点列表(参见9.1.3.1节)。极端情况下，一个密钥对应一个节点(1:1 node/key比例)。如果两个以上的实体共享一个特定的预先共享密钥，那么该密钥只把实体认证为那个组的成员，而不是单独的一个对象。
- RawPublicKey：DTLS被启用，设备有一个非对称密匙对，但是没有证书（一个原始公钥），它是使用out-of-band机制[RFC7250]来验证的，如9.1.3.2节中所述。该设备也有一个从公钥计算来的身份和一个可以沟通的节点的身份列表。
- Certificate:DTLS被启用，设备有一个带有X.509证书[RFC5280]的非对称密钥对，与它的目录绑定，并由一些常见的受信任的根证书颁发机构颁发，如9.1.3.3节所述。设备也有根信任锚的列表，可用于验证证书。

在“NoSec”模式下，系统只需向某个IP和端口发送带"coap" scheme头的UDP数据包即可。但只有当攻击者不能利用CoAP节点收发包时才是安全的；在11.5节中有关于这个问题的附加描述。

另外3种安全模式都利用DTLS实现，并且由“coaps” scheme和DTLS下的CoAP默认端口标识。这是是一个能用于认证(带有安全模型的限制)的安全关联，基于这个认证，可以授权其他的通信端。CoAP本身不提供用于认证或者授权的协议原语；当需要的时候，由通信安全(比如IPSec或DTLS)或者对象安全(带有负载)提供。在特定操作需要认证的设备通常使用这两种安全方式。在涉及中间人的地方，只有当中间人是信任体系中的一部分时才能保证通信安全。CoAP并没有提供一种方法来转发不同级别的授权，客户端可能与其他中间人或原始服务器有这些授权，因此可能需要在第一个中间人的时候就要执行所有授权。

9.1 DTLS-Secured CoAP

如同HTTP在TCP中使用TLS来保证安全一样，CoAP在UDP之上使用DTLS[RFC6347]来保证安全(图13)。本节定义了CoAP绑定到DTLS和适合受限环境的必须强制执行最低的配置。绑定是通过一系列的单播CoAP增量来定义的。实际上，DTLS就是TLS和附加功能来处理UDP传输的不可靠特性。

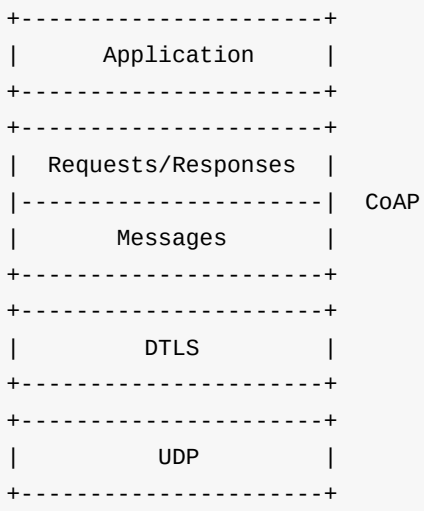

图 13: DTLS-Secured CoAP的抽象分层

在一些受限节点(有限的flash和RAM)和网络(受限带宽或者高可扩展性要求)中，依赖于正在使用的特定密码组合，DTLS的所有模式可能是不可用的。一些DTLS密码组合在设置安全关联时，可能增加一些复杂的重要实现以及一些所需的初始信号交换开销。一旦完成最初的握手，DTLS添加一个大约13字节的有限的per-datafgram开销，不包括任何的初始化向量/随机数(例如8个字节的TLS_PSK_WITH_AES_128_CCM_8[RFC6655]),完整性检查值(例如8个字节的TLS_PSK_WITH_AES_128_CCM_8[RFC6655]),和一些密码组合要求的填充值。考虑到可能适用的特定密码组，应该仔细权衡DTLS的给定模式是否适用于以CoAP为基础的应用程序，会话维护是否与应用流程兼容，用于约束节点和额外的网络开销的资源是否足够。(对于一些使用DTLS的模式，本规范确定了一个强制执行的密钥组。当这些密钥组的确合适时，能实现最大化的互通性。应用的特定安全策略可以决定使用的实际密钥组的实际设置)。DTLS不适用于组密钥(多播通信)；然而，它可能是未来组密钥管理协议中的一部分。

9.1.1 消息层

一个端能够作为CoAP客户端也应该能够作为DTLS客户端。它应该在合适的端口向服务端发起会话。当DTLS的握手结束时，客户端可能发起第一个CoAP请求。所有的CoAP消息必须当做DTLS"应用数据"发送。

为了将ACK消息或RST消息匹配到CON消息，或者RST消息匹配到NCON消息，附加上下面的规则：DTLS会话必须一致，时间段必须一致。

当消息在发送时有一致的DTLS会话、一致的时间段和相同的消息ID，那么消息就是一致的。

注意：当重传一个CON消息，尽管CoAP的消息ID一致，但每次尝试都会使用一个新的DTLS序列号。因此接收者必须按照4.5节中描述的去执行重复数据删除。重传不能跨时间段执行。

在RawPublicKey和认证模式中的DTLS连接被设置为使用相互验证，因此它们能在两个方向上维持连接并重用于未来的消息交换。当设备需要恢复资源时，设备可以关闭一个DTLS连接，但是通常它们必须尽可能保持长连接。在每次CoAP消息交换后关闭DTLS连接是很低效的。

9.1.2 请求/响应层

为了将响应匹配到请求上，加上下面的规则：DTLS会话必须一致，时间段必须一致。

这意味着对于一个DTLS安全请求的响应必须一直使用相同的安全会话和时间段。任何想提供一个DTLS请求的NoSec响应的尝试并不匹配这个请求，因此必须被拒绝（除非它匹配一个无关的NoSec请求）。

9.1.3 端点身份

按照[RFC6066]的第3节的定义，设备应当支持服务器名称指示(SNI)来指示在SNI主机名称域中的授权。有了这个特性，当一个作为多个权威的虚拟服务器的主机接收到新的DTLS连接时，它知道这个DTLS会话要使用哪个密钥。

9.1.3.1，Pre-Shared Keys

当形成一个新节点的连接时，系统根据它试图到达哪些节点来选择一个适当的密钥，然后使用DTLS PSK（预共享密钥）形成一个DTLS会话。在这些模式中的执行必须支持强制执行密码组TLS_PSK_WITH_AES_128_CCM_8，如[RFC6655]中所规定的。

根据调试模型，应用可能需要为身份提示定义应用规范(参见[RFC4279]的5.2节)，以支持PSK身份提示的使用。

应用了[RFC4279]中第7节的安全性考虑。应用应当仔细权衡它是否需要完全正向保密(PFS)，并且选择一个合适的密钥组([RFC4279]的7.1节).PSK的信息熵必须足够应付强力攻击(PSK不是随机选择的而是人为选择的)和字典式攻击([RFC4279]的7.2节)。客户端身份的明文通信可能会泄露数据或者隐私([RFC4279]的7.3节)。

原始公钥证书

在这种模式下，设备有个非对称密钥对但是没有X.509证书(原始公钥)；例如，非对称秘钥对是由厂商生成并安装在设备上(参见11.6节)。一个设备可能配置了多个原始公钥。原始公钥的类型和长度取决于所使用的密钥组。在RawPublicKey模式中执行必须支持强制执行的密码组TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8，如[RFC7251]，[RFC5246]和[RFC4492]中所规定的。使用的秘钥必须带有ECDSA。curve secp256r1必须支持[RFC4492]；这和NIST P-256 curve相等。这个哈希算法是SHA-256。实现必须使用Supported Elliptic Curves和被支持的点格式扩展[RFC4492]；必须支持未压缩的点格式；[RFC6090]可以作为一个实现方法。一些有关实现这个密钥组的指导可以在[W3CXMLSEC]中找到。使用原始公钥与TLS的机制在[RFC7250]中有规定。

实现注意：这意味着在图14中列出的带有至少一个值的扩展将出现在DTLS握手中。

```
Extension: elliptic_curves
Type: elliptic_curves (0x000a)
Length: 4
Elliptic Curves Length: 2
Elliptic curves (1 curve)
Elliptic curve: secp256r1 (0x0017)

Extension: ec_point_formats
Type: ec_point_formats (0x000b)
Length: 2
EC point formats Length: 1
Elliptic curves point formats (1)
EC point format: uncompressed (0)

Extension: signature_algorithms
Type: signature_algorithms (0x000d)
Length: 4
Data (4 bytes): 00 02 04 03
HashAlgorithm: sha256 (4)
SignatureAlgorithm: ecdsa (3)
```

图14: TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 的DTLS扩展表示

配置

RawPublicKey模式被设计成可轻易配置在M2M部署里。假定每个设备有一个已经安装好的非对称公钥对。在[RFC6920]的第2节中描述了端点从公钥中计算出标识符。所有支持检查RawPublicKey标识符的执行必须至少支持sha-256-120模式(SHA-256截断为120位).实现也应当支持更长的长度标识符和可以支持更短的长度。请注意更短的长度在面对攻击时提供更少的安全保护，因此不推荐使用。 通过URI、二进制、与/或人可识别的格式，标识符被给到验证它们的系统。 [RFC6920]。所有的实现应当支持二进制模式，并且包含用户接口的实现也要支持人可识别的格式。

在配置期间，收集每个节点的标识符，例如通过读取设备外部的条形码或者获取标识符的预编译列表。这些标识符安装在相关的节点中，例如一个M2M的数据收集服务端。标识符用于两个目的，与端点连接更多的设备信息和执行访问控制。在（最初的和进行的）配置期间，标识符的访问控制列表也应该安装和维护，用这些标识符，设备可以开始DTLS会话。

X.509证书

在证书模式下实现必须支持强制实施的密码组TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8如[RFC7251]，[RFC5246]和[RFC4492]中所规定。即证书包括SubjectPublicKeyInfo，它会显示带有danamedCurves secp256r1的id-ecPublicKey的算法 [RFC5480]；公钥格式是未压缩的[RFC5480]；哈希算法是SHA-256；如果包括的话，密匙使用扩展显示数字签名。证书必须使用secp256r1签署ECDSA，而且签名必须使用SHA-256。使用的密匙必须是ECDSA capable。必须支持curve secp256r1[RFC4492]；这个曲线和NIST P-256曲线是等效的。哈希算法是SHA-256。实现必须使用Supported Elliptic Curves和Supported Point Formats Extensions[RFC4492];必须支持未压缩的点格式；[RFC6090]可以作为一个实现方法。

证书的主语会从该设备长期的惟一标识符中建立，例如 EUI-64 [EUI64]。这个主语也可以基于完全限定域名(FQDN)，它是被用作CoAP URI的主机部分。然而，设备的IP地址一般不能当做主语，因为它是可变的。系统中的发现过程会建立给定设备的IP地址和每个设备主语之间的映射。一些含有多于一个主语的设备也必须包含多个证书。

当生成一个新的连接，远程设备的证书就需要验证。如果CoAP节点有一个绝对时间的来源，那么节点应当检查证书的有效时间在范围之内。为了达到安全要求，证书必须被验证为适合于安全需求，使用的功能相当于[RFC5280]第6节中指定的算法。如果认证包含一个SubjectAltName，那么请求URI的授权必须匹配至少一个在SubjectAltName族中URI类型域中的任何CoAP URI的授权。如果在证书中没有SubjectAltName，那么除了有通配符的证书不被允许以外，请求URI的授权必须匹配证书中的通用名(CN)，使用http://tools.ietf.org/pdf/rfc3280中定义的匹配规则。

证书状态检查的核心支持需要进一步研究。由于在线证书状态协议(OCSP)[RFC6960]到CoAP的映射当前没有被定义，而且OCSP也可能不是很容易适用于所有环境，所以另一种方法可能是使用TLS证书状态请求扩展（[RFC6066]第8节；也被称为“OCSP装订”）或最好是多个证书状态扩展（[RFC6961]），如果可用的话。

如果系统除了证书外还有一个共享密钥，那么应该使用一个包括共享密钥（如TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA[RFC5489]）的密码组。

第十章 CoAP和HTTP的跨协议代理

CoAP支持HTTP功能的有限子集，因此从CoAP代理到HTTP是很简单的。在CoAP和HTTP之间采用代理可能有几个原因，例如，在设计一个这两种协议都可以使用的网站界面，或在实现CoAP-HTTP代理时。同样地，CoAP也可以代理到诸如XMPP[RFC6120]或者SIP[RFC3264]等协议；这些代理机制的定义超出本规范的范围。

通过一个正向代理来访问资源有两个可能的方向：

CoAP-HTTP代理：通过一个中间人使得CoAP客户端访问HTTP服务端的资源。这是通过在CoAP-HTTP代理的CoAP请求里包含带有“http”或“https”URI的Proxy-Uri或Proxy-Scheme选项发起的。

HTTP-CoAP代理：通过一个中间人使得HTTP客户端访问CoAP服务端的资源。这是通过在HTTP-CoAP代理的HTTP请求的Request-Line 中指定“coap”或“coaps”URI发起的。

无论哪种方式，只有CoAP的请求/响应模型被映射到HTTP。CON或者NON消息等模型应该是透明的，对代理功能没有影响。下面的章节描述对正向代理的请求的处理。没有提及反向代理，因为代理功能对客户端是透明的，就相当于原始服务器一样。然而，对反向代理的考虑和对正向代理的考虑应该是一样的，而且通常会期望反向代理以与正向代理类似的方式运行。实现中需注意，HTTP客户端函数库没有提供一种方法将CoAP URI放在HTTP的 request-Line中，使得操作HTTP-CoAP正向代理变得困难；反向代理可能因此有更好的适用性。另外一份规范会定义例如HTTP-CoAP反向代理的的URIs操作的规定[MAPPING]。

10.1 CoAP-HTTP代理

如果‘http’或‘https’URI中的请求包含Proxy-Uri或者Proxy-Scheme选项[RFC2616]，那么接收的CoAP端点(今后称为“代理”)要求对指明的HTTP资源执行请求方法中指定的操作，并向客户端返回结果。(可参考5.7节，如何生成包含安全要求的代理请求)

这一节为所有CoAP 请求指定了代理应该返回到客户端的CoAP响应。代理实际如何响应请求是一个实现细节，期望的典型情况是代理转发请求到HTTP源服务端。

由于HTTP和CoAP共享基本的请求方法集，因此在HTTP资源上执行CoAP请求与在CoAP资源上执行它并没有什么不同。本节的下面的小节中将解释在HTTP资源上执行的每个CoAP方法的含义。

如果代理不能或者不愿服务带有HTTP URI的请求，那么向客户端返回5.05(Proxying Not Supported)响应。如果代理通过与第三方交互(例如HTTP原始服务端)来服务请求，并且无法再合理的时间内获得结果，返回一个5.04(GateWay Timeout)的响应；如果可以获取结果但是不能解释该结果，返回5.02(Bad Gateway)的响应。

10.1.1 GET

GET方法请求代理返回一个由请求URI定位的HTTP资源表现。

一旦成功了，必须返回2.05(Content)响应码。响应的payload必须是目标HTTP资源的表现，而且必须设定相应的Content-Format选项。响应必须携带Max-Age值，这个值不大于该资源的刷新剩余时间。如果HTTP实体有一个实体标记，代理必须在响应中包含ETag选项，并且按照下面的描述处理请求中的ETag选项。

客户端可以通过包含下面的选项来影响GET请求的流程：

- Accept：请求可以包含一个Accept选项，标识优先的响应内容格式。
- ETag：请求可以包含一个或多个ETag选项，标识客户端存储的响应。这就要求当代理需要发送2.03(Valid)响应时，转而发送带有请求的实体标记的2.05(Content)响应。请注意CoAP ETags是HTTP看来的强ETags；CoAP没有HTTP的弱ETags，并且在跨协议代理中没有好的方式来使用它们。

10.1.2 PUT

PUT方法要求代理按照请求URI定位的路径来更新或者创建HTTP资源。

如果按照请求URI创建了一个新的资源，那么必须向客户端发送2.01(Created)的响应。如果修改了一个已经存在的资源，那么必须返回一个2.04(Changed)响应来标示请求的成功完成。

10.1.3 DELETE

DELETE方法请求代理在HTTP源服务端按照请求的URI删除HTTP资源。

如果成功了或者在请求时不存在该资源，那么向客户端发送一个2.02(Deleted)响应。

10.1.4 POST

POST方法要求代理将包含在请求中的表现更新到HTTP原始服务端。POST方法的实际执行结果由原始服务端决定，并且依赖于请求URI定位的资源。

如果通过POST方法执行的动作不会产生一个可以由URI来标识的资源，那么必须向客户端回复2.04(Changed)的响应。如果资源在源服务端被创建，那么必须返回2.01(Created)的响应。

10.1 HTTP-CoAP代理

如果一个HTTP请求包含带有“coap”或者“coaps”URI的Request-URI，那么接收的HTTP端(今后称为“代理”)要求对指明的CoAP资源执行请求方法中指定的操作，并向客户端返回结果。本小节定义了针对任何HTTP请求，代理应该向客户端返回的响应。除非另有说明，所有的申明都是推荐的行为，在一些特别受限的实现需要使用简化方式。代理如何响应请求是一个执行细节，然而期望的典型情况是代理转发请求到CoAP原始服务端。在下面的小节中将解释在CoAP资源上执行的每个HTTP方法的含义。

如果代理不能或者不想服务带有CoAP URI的请求，将会向客户端返回501(Not Implemented)响应。如果代理通过与第三方交互(比如CoAP原始服务端)来服务请求，并且无法再合理的时间内获得结果，返回一个5.04(GateWay Timeout)的响应；如果可以获取结果但是不能解释该结果，返回5.02(Bad Gateway)的响应。

10.2.1 OPTIONS and TRACE

由于CoAP不支持OPTIONS和TRACE方法，必须向客户端返回501(Not Implemented)错误。

10.2.2 GET

GET方法要求代理返回由Request-URI定位的CoAP资源的表现。

一旦成功，返回200(OK)响应码。响应的payload必须是目标CoAP资源的表现，而且必须设定相应的Content-Type和Content-Encoding域。响应必须携带Max-Age值，这个值不大于该资源的刷新剩余时间。如果CoAP响应有一个ETag选项，代理必须在响应中包括一个ETag头。

客户端可以通过包含下面的选项来影响GET请求的流程：

- Accept：在请求中的HTTP Accept头的优先采用的媒体类型被映射到一个CoAP Accept选项。HTTP Accept 媒体类型的范围，参数和扩展都不被CoAP的Accept选项支持。如果代理不能找到双方（服务端和客户端）同时可以接受的的Accept域，那么代理将发送406响应。代理可能使用从HTTP Accept头的其他媒体类型来重新发起请求。
- Conditional GETS：有条件的HTTP GET要求能够被映射到对应CoAP请求的的“If-Match”或“If-None-Match”请求头域。“If-Modified-Since”和“If-Unmodified-Since”请求头域不能直接被CoAP支持，但是能够被缓存代理在本地执行。

10.2.3 HEAD

除了服务端必须不能在响应中返回一个消息体，HEAD方法与GET是一样的。

尽管在CoAP中没有与HTTP的HEAD方法直接等效的方法，HTTP-CoAP代理能够对CoAP资源的HEAD请求作出响应，并且返回不带消息体的HTTP头。

实现注意：HTTP-CoAP代理可能试着使用块传输选项来减小每次传输的数据量，但是需要注意原始服务端是否支持块传输。

10.2.4 POST

POST方法要求代理将包含在请求中的表现更新到CoAP原始服务端。POST方法实际执行的功能由原始服务端决定，并且依赖于请求URI指定的资源。

如果通过POST方法执行的动作不会产生一个可以由URI来标识的资源，那么必须向客户端返回一个200(OK)或者204(No Content)的响应。如果在源服务端已经创建了资源，那么必须返回201(Created)响应码。

如果在CoAP响应中有任何的Location-*选项，将返回由这些选项的值构造的Location header域。

10.2.5 PUT

PUT方法要求代理更新或者创建由请求URI定位的CoAP资源。

如果根据请求URI创建了一个新的资源，那么向客户端返回201(Created)响应码。如果修改了存在的资源，发送200(OK)响应码或者204(No Content)响应码来标示请求的成功完成。

10.2.6 DELETE

DELETE方法要求代理在CoAP原始服务端删除由请求URI定位的CoAP资源。

如果操作成功，那么向客户端发送200(OK)的响应；如果请求时不存在该资源，那么向客户端返回204(No Content)响应。

10.2.7 CONNECT

由于没有TLS到DTLS的通道，当前HTTP-CoAP代理功能不支持本方法。因此，会向客户端返回501(Not Implemented)错误。

第十一章 安全事项

本节分析协议面临的可能漏洞，并告知协议和应用开发者在本文档中CoAP的安全限制。因为CoAP实现了HTTP/1.1属性中的一个子集，在[RFC2616]的第15节中的安全考虑同样与CoAP相关。本节集中描述CoAP特有的局限性。

11.1 解析协议和处理URIs

网络解析的应用可以显示收包处理逻辑的漏洞。复杂的解析器可能是很多漏洞的根源，比如能够远程的破坏一个节点，或者甚至在节点上执行任意的代码。CoAP尝试通过降低解析器的复杂性、限定可编码的值一个范围、主动降低在多个代表相同事件的表现上请求不常用的方法的复杂度，来减少引入这类漏洞的可能性。许多URI处理都放在客户端，更进一步减少了往服务端引入漏洞的可能性。即便这样，CoAP执行中的URI处理代码仍然可能存在许多漏洞，因此需要谨慎处理。CoAP的访问控制的实现必须保证不因为URI和通过URI定位的资源代码的差异而引入漏洞。最复杂的解析器可能是CoRE Link Format，尽管它也可能是按照减少执行复杂度的目标设计的[RFC6690]，也可以参考[RFC2616]的15.2节。

11.2 代理和缓存

按照[RFC2616]中的15.7节提到的，代理由于其本身的中间人性质,可能会破坏直接CoAP消息交换的IPsec或者DTLS保护。因此，它们是CoAP消息交换中最可能破坏机密性或者完整性的。按照[RFC2616]中提到的，它们也可能破坏可用性。

当代理采用缓存的时候，会加大请求/响应数据的机密性和完整性的威胁。注意，CoAP没有定义任何能够为HTTP/1.1提供更好的敏感数据保护的cache-suppressing Cache-Control选项。

对于包含cache的情况，访问控制的实现需要注意，如果请求有对应的cache条目，对应的cache数据也需要应用控制规则。这对实现多个安全域的客户端以及服务于多个客户端的代理很有意义。同样，有缓存的代理必须不能转发缓存数据给那些传输安全性更低的代理。
与"coap"机制不同，对“coaps”认定的请求的响应不是“public”的，因此不能重用于共享的缓存，除非缓存能够对缓存条目进行等效的访问控制。如果CoAP中默认缓存了消息，它们能在私有的缓存中被重新利用。

最终，给多个原始请求端服务的代理可能发送单独响应(和附带响应相反)来提供额外的增幅(参考11.3)。

11.3 增幅的风险

CoAP服务端通常用响应包来回复请求包。响应包可能比请求包大许多。网络攻击者可能利用CoAP节点将一个小的攻击包变为一个大的攻击包，这个过程称为增幅。这就存在CoAP节点利用协议的增幅特性产生一个DoS攻击的风险：由于网络限制，攻击者占据的带宽有限，但通过扩大特性，能够使攻击者突破带宽限制。

由于UDP协议不提供验证请求包中的源地址的方法，因此在使能NoSec访问的节点中，网络攻击者能够访问它，并且能够访问潜在的被攻击者，这就存在风险。网络攻击者只需要将被攻击者的IP地址放在合适的请求包中的源地址中，就能产生定向到被攻击者的更大的包。

作为降低影响的方法，很多受限网络只能产生很小的通信量，这样使得CoAP节点在攻击中不容易被注意。然而，受限网络的有限带宽使得网络本身容易成为增幅攻击中的受害者。

因此，如果请求没有认证，在响应中不能有大的增幅因子。CoAP服务端通过使用CoAP的slicing/blocking模式和大的资源表现采用小的分配来减少为攻击者提供的增幅量。举个例子，对于一个1000字节的资源，一个10字节的请求引起一个80字节的响应(64字节的块)，而不是1016字节的响应，这样就相当的减少了提供的增幅。

CoAP也能在请求中支持多播IP地址的使用，这在M2M中是一项重要的要求。多播CoAP请求也许是事故或者DoS攻击的源头，尤其在受限网络中。本规范试图在响应返回时加一些限制来减少多播请求的增幅效果。为了减少恶意使用的可能性，CoAP服务端不能接受没被认证的多播请求，也对潜在的源加上一些多播的边界限制。可能的话，CoAP服务端应当限制对特定资源的多播请求的支持。

在提供POSIX接口API[IEEE1003.1]的通用操作系统中，很难查出一个包是否指向多播地址。很多实现能够知道自己是否已经加入多播组，采用FF0x::1格式指向多播地址的包会产生问题，这种包会被所有IPV6节点接收。实现时必须使用诸如IPV6_RECVPKTINFO[RFC3542]的新的API。

11.4 地址欺骗攻击

由于UDP中没有握手机制，一个欺诈端点能够自由的读写受限网络中的消息(例如NoSec或PreSharedKey中的nodes/key比率大于1：1)，这样就能轻易的使用下面手段来攻击单个端点，一组端点，甚至整个网络：

1. 在对CON消息或者NONCON消息的响应中回复一个虚假的复位消息，这样使得端点“deaf”
2. 在对CON消息的响应中回复一个虚假的ACK，这样可能会阻止CON消息的发送者重传，并让实际的响应失效。
3. 利用伪造的payload/options来欺骗整个响应(有几个不同层级的影响：从单个响应的破坏到配套基础设施的攻击，例如破坏代理缓存或者在资源目录中欺骗validation/lookup接口，更普遍的，任何储存网络状态的组件和利用CoAP为来处理和更新状态的通信设备体都是一个潜在的攻击目标。
4. 对目标节点多播欺骗请求，这会导致网络拥塞或崩溃，DoS攻击或者从强行唤醒。
5. 欺骗observe消息。

尽管没有实现基于随机token询问的安全传输层,不明来源的回应攻击可以被检测和缓和。[RFC4086]讨论了随机询问的安全机制。

特别的，使用CON消息的CoAP能侦测到其他类型的欺骗，因为从被欺骗的端点中有攻击源发来的ACK或者RST消息。但是欺骗保持message ID的追踪是很难的，另一方面在被攻击之后，利用此对探测欺骗非常有用。这些攻击可以利用安全模式而不是NoSec来防范。

无论有没有源地址欺骗，客户端能试图往服务端发送复杂的请求来加重服务端的负载；地址欺骗使得回溯，阻塞和攻击更加困难。由于CON请求的花销很小，所以很容易执行这种攻击。在这种攻击下，带有受限电源的受限节点会比预期更快的耗尽能量(电量耗尽攻击)。如果客户端利用CON消息和服务器利用分离的CON回应一个不会响应的地址(可能是欺骗)，服务器就需要为其分配内存和对每一个未回应的节点进行回应这回耗尽服务为的MAX_TRANSMIT_SPAN，使服务器没有资源处理合法的交互。后面的问题可以可通过限制回应的速率以减缓，见4.7.攻击者可以伪装合法客户端的地址进行欺骗这可能引起服务端阻塞针对客户端的合法的回应，因为NSTART=1,。攻击是针对非安全模式的,这些攻击在安全模式被避免，而不是NoSec。

11.5 跨协议攻击

CoAP端点向虚假的源地址发送包不仅能用于增幅，也能用于对一个监听给定地址(IP地址和端口)的受害者进行跨协议攻击。按照下面步骤就会发生：

攻击者往CoAP端点发送一个带有将虚假源地址当做给定地址的消息。

CoAP端点向给定的源地址发送响应消息。

在给定地址接收UDP包的受害者就会按照不同协议的规则进行解析。

它可以用于绕过阻止从攻击者向受害者通信的防火墙规则，但是将允许从CoAP端点(在其他协议中也担当有效角色)到受害者通信。

另外，CoAP端点可能成为由其他基于UDP协议(譬如DNS)的端点发起的跨协议攻击的受害者。在这些情况下，如果端点的安全属性依赖于检查IP地址(和使用虚假IP地址来切断外界攻击的防火墙)，就有可能遭受攻击。通常，由于基于UDP的协议缺乏上下文，因此它们更容易成为跨协议攻击的目标。

最后，由其他方式传输的CoAP URI能够用于使得客户端往其他协议的端点发送消息。

一个减轻跨协议攻击的措施是严格检查接收包的语法和语法中的足够多的差异。举个例子，如果很难使得DNS服务器向CoAP端点发送一个传递检查的DNS响应，那么就会起作用。可惜的是，DNS回复的前两个字节是能被攻击者选中的ID，并且映射为CoAP头部的关键部分，后面两个字节被当做CoAP的消息ID(任何值都可行)。DNS的计数字可以当做一个(不存在但是可选的)CoAP选项0的多重实例，或者当做一个Token。攻击者利用重复的查询来在CoAP端点上达到一个预期效果；服务端增加的响应(如果有的话)将被当做额外的负载。

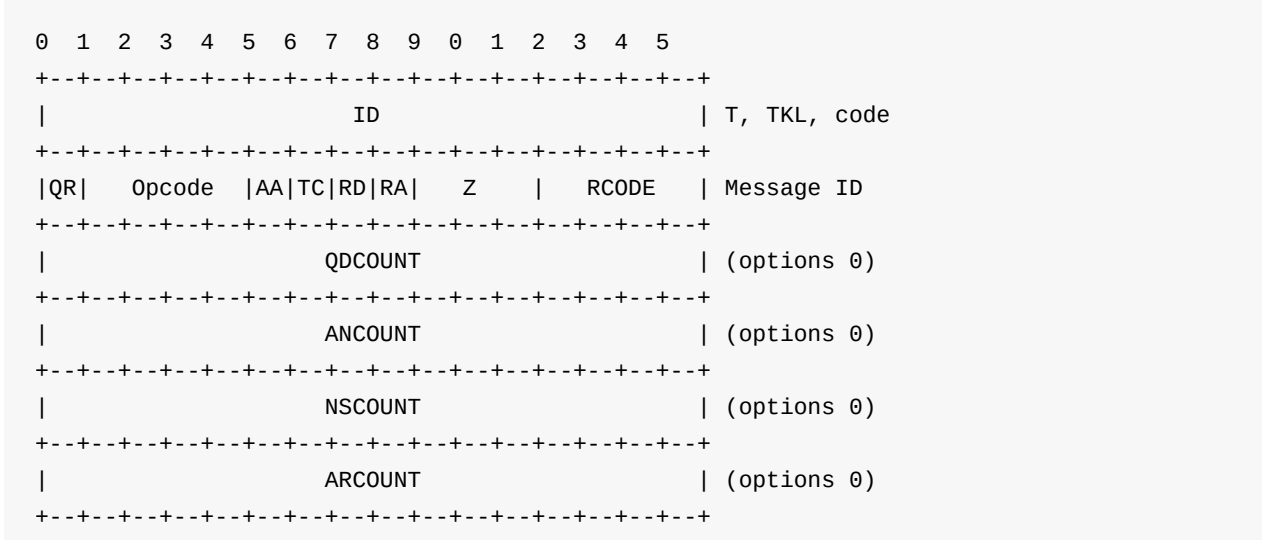

Figure 15: DNS头部 ([RFC1035]的4.1.1节[http://tools.ietf.org/pdf/rfc1035#section-4.1.1]) 和 CoAP消息

一般来说，对于任何一对协议，协议之一很容易设计为使得攻击者生成类似另一种协议的消息的回复。比起保证或者证明不存在可实行的攻击，生成可能未完全启动的但可能被二次开发的攻击的实例通常更加困难。如果端点依据受信任包的源IP地址而没有授权攻击者，那么跨协议攻击才能完全的减轻。相反的，完全依赖防火墙的NoSec环境下的CoAP安全不仅需要防火墙来切断CoAP的端点，并且使得所有其它的端点使用一些基于UDP的协议来往CoAP端点发送UDP消息。

除了上面的考量，也要考虑跨协议攻击的DTLS安全。举例，如果相同的DTLS安全连接("connection")用于传输多种协议的数据，那么DTLS就不对这些协议提供应对跨协议攻击的保护。

11.6 受限节点的注意事项

受限节点上的实现经常发现它们没有好的信息熵来源[RFC4086]
[http://tools.ietf.org/pdf/rfc4086]。在这种情况下，节点**必须**不能用于对信息熵要求高的处理过程，比如说key的生成。另外，在制造或者运行中可以生成keys并添加到设备中。

由于受限节点的低处理能力，它们最容易受到时序攻击的影响。在实现密码原语时需多加注意。

在暴露环境中安装大量的受限节点会使得它们对窜改没有抵抗能力，包括密匙内容的恢复。当定义分配给它们的认证信息时需要考虑到这一点。尤其是，当向一组节点分配一个共享的key可能使得任何一个单一的受限节点成为破换整个组的目标。

第十二章 互联网地址分配注意事项

这篇文档定义了两个sub-registries，给CoAP头部代码字段的值包含“Constrained RESTful Environments (CoRE) Parameters”注册表。将来参考“CoRE参数”注册表。

这两个sub-registries都是8位值，可以用三个十进制的符号表示为“c.dd”,用一个句号将第一位和第二位数字分离开；第一位数字c为0~7，表示代码种类；第二个和第三个数字dd为00~31的十进制数，表示细节。

所有的代码值都按照sub-registries，按照如下范围安排：

```
0.00          表示一个空的消息（见4.1章）
0.01-0.31      表示一个请求。这个范围的值是根据“CoAP Method Codes”的sub-registry分配的（见12.1
.1节）
1.00-1.31      保留
2.00-5.31      表示一个响应。这个范围的值是根据“CoAP Response Codes”的sub-registry分配的（见12
.1.2节）
6.00-7.31      保留
```

12.1 CoAP代码注册

这篇文档定义了两个sub-registries，给CoAP头部代码字段的值包含“Constrained RESTful Environments (CoRE) Parameters”注册表。将来参考“CoRE参数”注册表。

这两个sub-registries都是8位值，可以用三个十进制的符号表示为“c.dd”,用一个句号将第一位和第二位数字分离开；第一位数字c为0~7，表示代码种类；第二个和第三个数字dd为00~31的十进制数，表示细节。

所有的代码值都按照sub-registries，按照如下范围安排：

```
0.00          表示一个空的消息（见4.1章）
0.01-0.31      表示一个请求。这个范围的值是根据“CoAP Method Codes”的sub-registry分配的（见12.1
.1节）
1.00-1.31      保留
2.00-5.31      表示一个响应。这个范围的值是根据“CoAP Response Codes”的sub-registry分配的（见12
.1.2节）
6.00-7.31      保留
```

12.1.1 方法码(Method Codes)

这个sub-registry是“CoAP Method Codes”

每次进入这个sub-registry都必须包含在0.01-0.31范围内的Method Code，方法的名字，方法文档的参考。

初始化进入这个sub-registry如下：

```
+------+--------+-----------+
| Code | Name   | Reference |
+------+--------+-----------+
| 0.01 | GET    | [RFC7252] |
| 0.02 | POST   | [RFC7252] |
| 0.03 | PUT    | [RFC7252] |
| 0.04 | DELETE | [RFC7252] |
+------+--------+-----------+
    表 5: CoAP Method Codes
```

其他Method Codes没有安排。

互联网号码分配政策在未来为这个sub-registry额外定义描述在“IETF Review or IESG Approval” [RFC5226].

方法代码的文档需要指定这个请求的语义，包含如下属性：

这个方法的响应码成功返回。
这个方法是否幂等，安全或两者都满足。

12.1.2 响应码

这个sub-registry的名字是“CoAP Response Codes”

每个sub-registry必须包含在2.00-5.31范围内的响应码，响应码的描述，响应码的文档参考。

初始化进入这个sub-registry如下：

Code	Description	Reference
2.01	Created	[RFC7252]
2.02	Deleted	[RFC7252]
2.03	Valid	[RFC7252]
2.04	Changed	[RFC7252]
2.05	Content	[RFC7252]
4.00	Bad Request	[RFC7252]
4.01	Unauthorized	[RFC7252]
4.02	Bad Option	[RFC7252]
4.03	Forbidden	[RFC7252]
4.04	Not Found	[RFC7252]
4.05	Method Not Allowed	[RFC7252]
4.06	Not Acceptable	[RFC7252]
4.12	Precondition Failed	[RFC7252]
4.13	Request Entity Too Large	[RFC7252]
4.15	Unsupported Content-Format	[RFC7252]
5.00	Internal Server Error	[RFC7252]
5.01	Not Implemented	[RFC7252]
5.02	Bad Gateway	[RFC7252]
5.03	Service Unavailable	[RFC7252]
5.04	Gateway Timeout	[RFC7252]
5.05	Proxying Not Supported	[RFC7252]

表 6: CoAP Response Codes

响应码3.00-3.31是预留给将来使用。所有其他响应码都没有被安排。

互联网号码分配政策为这个sub-registry以后额外的定义描述在“IETF Review or IESG Approval”[RFC5226].

响应码的文档需要指定这个响应的语义，包含如下属性：

- 响应码应用方式
- 是否需要携带payload，option。
- payload的语义。举个例子，2.05（内容）响应的payload是目标资源的展示；payload在

错误的响应中是可读和诊断的。

- payload的格式。举个例子，这个格式在2.05（内容）响应是通过内容格式选项表示；payload的格式在一个错误的响应中总是Net-Unicode文本
- 响应是否可以缓冲，取决于freshness model
- 响应是否通过合法性检查，取决于validation model
- 响应是否导致一个cache来标志响应已经存储，表明这个请求的URI不是最新的。

12.2 CoAP Option Number Registry

这篇文档为CoAP options中“CoRE Parameters”注册表中的option编号定义了一个sub-registry。这个sub-registry的名字是“CoAP Option Number”。

每个sub-registry必须包含这个option编号，option的名称，还有option文档参考。

初始化进入这个sub-registry如下：

Number	Name	Reference
0	(Reserved)	[RFC7252]
1	If-Match	[RFC7252]
3	Uri-Host	[RFC7252]
4	ETag	[RFC7252]
5	If-None-Match	[RFC7252]
7	Uri-Port	[RFC7252]
8	Location-Path	[RFC7252]
11	Uri-Path	[RFC7252]
12	Content-Format	[RFC7252]
14	Max-Age	[RFC7252]
15	Uri-Query	[RFC7252]
17	Accept	[RFC7252]
20	Location-Query	[RFC7252]
35	Proxy-Uri	[RFC7252]
39	Proxy-Scheme	[RFC7252]
60	Size1	[RFC7252]
128	(Reserved)	[RFC7252]
132	(Reserved)	[RFC7252]
136	(Reserved)	[RFC7252]
140	(Reserved)	[RFC7252]

表 7: CoAP Option Numbers

互联网号码分配政策为这个sub-registry以后额外的定义分为了如下3层。0..255是为option保留，在IETF有被定义（IETF Review or IESG Approval）。256..2047是为普通使用的包含公开规格（Specification Required）的options保留的。2048..64999是为了所有其他options，包含私人的或者赞助商规格的，这些需要经过一个特定的专家审核来确定这个option语法是定义正确的。在6500和65535之间的Option编号是保留下来用于实验。他们不是给赞助商使用，而且是禁止用于操作部署。

```
+-------------+----------------------------------------+
   |       Range | Registration Procedures                |
   +-------------+----------------------------------------+
   |       0-255 | IETF Review or IESG Approval           |
   |    256-2047 | Specification Required                 |
|  2048-64999 | Expert Review                           |
| 65000-65535 | Experimental use (no operational use) |
+-------------+----------------------------------------+
```

表 8: CoAP Option Numbers: Registration Procedures

这个option编号的文档应该指定这个option和它的编号，包含以下属性：

- option在请求中的意义。
- option在响应中的意义。
- 这个option是critical还是elective，由option编号决定。
- 这个option是否是safe-to-forward，如果是，是否是cache-key的一部分，这些由option编号决定（见5.4.2节）
- option值的格式和长度。
- 是否option只一次出现，还是它能出现很多次。
- 默认值。对于一个有默认值的critical option，存在这样的讨论，默认值如何能处理通过实施，而这个实施又不支持critical option（见5.4.4节）

12.3 CoAP Content-Formats Registry

互联网媒体类型被定义成一个字符串，例如“application/xml”[RFC2046]。为了最小化使用这些媒体类型表示格式所带来的payload开销，这篇文档为互联网媒体类型子集定义了一个sub-registry，这些子集在CoAP中使用，并且每个都分配好，和content-coding合并，有数字标识。这个sub-registry的名字是“CoAP Content-Formats”，在“CoRE Parameters”表里面。

每次进入到sub-registry必须包含这些IANA注册过的媒体类型，用于定义CoAP中的媒体类型数字的范围是0-65535，cotent-coding和这些定义相关，一个文档参考描述了payload和媒体类型的表达语义。

CoAP不包括用分离的方式来转移content-encoding信息和请求或响应，因为content-encoding每一个标识符也是特定的。如果多种content-encodings和媒体类型一起使用，然后每一个分离的content-format标识符将会被注册。相似的，其他参数和互联网媒体类型关联，比如level，也能被定义为CoAP Content-Format条目。

初始化进入这个sub-registry如下：

Media type	Encoding	ID	Reference
text/plain; charset=utf-8	-	0	[RFC2046] [RFC3676] [RFC5147]
application/link-format	-	40	[RFC6690]
application/xml	-	41	[RFC3023]
application/octet-stream	-	42	[RFC2045] [RFC2046]
application/exi	-	47	[REC-exi-20140211]
application/json	-	50	[RFC7159]

表 9: CoAP Content-Formats

65000~65535之间所包含的标识符是预留用于实验。他们不是给赞助商使用，也禁止用于操作部署。256~9999之间的标识符预留是用于未来IETF规格（IETF Review or IESG Approval）。所有其他标识符都尚未安排。

由于一个字节标识符的命名空间非常小，IANA政策对于sub-registry未来额外的定义在0~255范围，是“Expert Review”描述在[RFC5226]。IANA政策对于额外的定义在1000-64999包含了“First Come First Served”，描述在[RFC5226]。总结在下表中。

Range	Registration Procedures
0-255	Expert Review
256-9999	IETF Review or IESG Approval
10000-64999	First Come First Served
65000-65535	Experimental use (no operational use)

表 10: CoAP Content-Formats: Registration Procedures

在M2M的应用中，不希望这些常用的媒体类型，比如text/plain,application/xml或者application/octet-stream在实际应用中长期有效。推荐M2M的应用中使用CoAP请求新的互联网媒体类型，这些媒体类型来自于IANA表明的语义，该语义信息规定了如何创建和理解一个payload。举个例子，一个智能的payload携带的一个XML，可能请求一个更特定的类型，比如application/se+xml或者application/se-exi。

12.4 URI Scheme Registry

这篇文档中包含注册符合CoAP规范的URI请求。这个请求遵从于[RFC4395]

- URI 名字.

```
coap
```

- Status.

```
Permanent.
```

- URI 规范语法.
  在[RFC7252]的6.1 中定义.
- URI 规范的语义.
  CoAP URI规范提供了一种能在CoAP协议上定位和访问资源的的方法。通过CoAP服务器可以定义和操作资源。该规范是参考Http的URI见RFC2616
- 编码注意项(Encoding considerations).
  该规范的编码规则遵循已近制定的[RFC3986]，例如其互联网和资源章节表明UTF-8-based percent-encoding.
- Applications/protocols that use this URI scheme name.
  CoAP端点利用该规范访问CoAP资源
- 互操作注意项(Interoperability considerations).
  None.
- 安全注意项(Security considerations).
  See Section 11.1 of [RFC7252].
- 联系方式.
  IETF Chair chair@ietf.org
  Author/Change controller. IESG iesg@ietf.org
- References.
  [RFC7252]

12.5 安全URI规范注册表

该文档包含:

- URI 规范名称.
  coaps Status.
- Permanent.
  URI scheme syntax.
  Defined in Section 6.2 of [RFC7252].
- URI 规范语义.
  CoAP URI规范提供了一种能在CoAP协议上标识资源的的方法，该方法把DTLS作为其安全传输层。通过CoAP服务器可以定义和操作资源。该规范是参考Http的URI [RFC2616] 和 [RFC7252]。
- 编码注意项(Encoding considerations).
  该规范的编码规则遵循已近制定的[RFC3986]，例如其互联网和资源章节表明UTF-8-based percent-encoding.
- Applications/protocols that use this URI scheme name.
  CoAP端点利用该规范通过DTLS访问CoAP资源
- 互操作注意项(Interoperability considerations).
  None.
- 安全注意项(Security considerations).
  See Section 11.1 of [RFC7252].
- 联系方式.
  IETF Chair chair@ietf.org
  Author/Change controller. IESG iesg@ietf.org

12.6 安全服务名称和端口号表

- 服务名称.
  coaps
- 传输协议.
  udp
- 代理.
  IESG iesg@ietf.org
- 联系方式.
  IETF Chair chair@ietf.org
- 描述.
  DTLS-secured CoAP
- 参考.
  [RFC7252]
- 端口号. 5684

12.7 多播地址表

第8章，“多播CoAP”，定义了多播的使用。IANA已经安排了如下被用于CoAP节点多播地址：

- IPv4 --“All CoAP Nodes”地址为224.0.1.187，源自“IPv4 Multicast Address Space Registry”。当这个地址用于发现
- IPv6 -- "All CoAP Nodes" address FF0X::FD, from the "IPv6 Multicast Address Space Registry", in the "Variable Scope Multicast Addresses" space (RFC 3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to; CoAP needs the Link-Local and Site-Local scopes only.