

中华人民共和国通信行业标准

YD/T 3241—2017

受限应用协议（CoAP）技术要求

Technical requirements for constrained application protocol (CoAP)

2017-04-12 发布

2017-07-01 实施

中华人民共和国工业和信息化部 发布

目 次

前言.....	III
1 范围.....	1
2 规范性引用文件.....	1
3 缩略语、术语和定义.....	1
3.1 缩略语.....	1
3.2 术语和定义.....	2
4 业务特征.....	4
4.1 概述.....	4
4.2 消息模型.....	4
4.3 请求/响应模型.....	5
4.4 中介与缓存.....	6
4.5 资源发现.....	6
5 消息格式.....	7
5.1 基本格式.....	7
5.2 可选项格式.....	8
5.3 可选项值格式.....	8
6 消息传送方式.....	9
6.1 基本方式.....	9
6.2 消息与端点.....	9
6.3 消息可靠传送.....	9
6.4 消息不可靠传送.....	10
6.5 消息相关性.....	11
6.6 数据重复.....	11
6.7 消息大小.....	11
6.8 拥塞控制.....	12
6.9 传送参数.....	12
7 请求/响应语义.....	14
7.1 概述.....	14
7.2 请求.....	14
7.3 响应.....	15
7.4 请求/响应匹配.....	16
7.5 可选项.....	17
7.6 负载与表示.....	20

7.7	缓存	21
7.8	代理	22
7.9	方法定义	24
7.10	响应代码定义	25
7.11	可选项定义	28
8	CoAP URI	32
8.1	CoAP URI 概述	32
8.2	CoAP URI 方案	32
8.3	coaps URI 方案	33
8.4	规范化和比对规则	33
8.5	将 URI 分解成可选项	33
8.6	将可选项组合成 URI	34
9	发现	35
9.1	服务发现	35
9.2	资源发现	35
10	组播	36
10.1	组播的意义	36
10.2	消息层	36
10.3	请求/响应层	36
11	安全	37
11.1	安全概述	37
11.2	DTLS-secured CoAP	38
12	CoAP 与 HTTP 协议转换代理	41
12.1	转换代理的意义	41
12.2	CoAP-HTTP 代理	42
12.3	HTTP-CoAP 代理	43

前 言

本标准是受限应用协议系列标准之一，该系列标准的名称及结构如下：

- 《受限应用协议（CoAP）技术要求》
- 《受限应用协议（CoAP）测试方法》

本标准按照 GB/T 1.1—2009 给出的规则起草。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

本标准由中国通信标准化协会提出并归口。

本标准起草单位：中国信息通信研究院、北京邮电大学。

本标准主要起草人：付国强、罗 松、黄小红。

受限应用协议（CoAP）技术要求

1 范围

本标准规定了受限应用协议技术要求，主要包括：受限应用协议概述和业务特征、受限应用协议的消息格式、受限应用协议的消息传送方式、请求响应语义和组播问题等。

本标准适用于使用受限应用协议的设备。

2 规范性引用文件

下列文件对于本文件的应用是必不可少的。凡是注日期的引用文件，仅所注日期的版本适用于本文件。凡是不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

IETF RFC 3986 统一资源标识符：通用语法（Uniform Resource Identifier: Generic Syntax）

IETF RFC 4492 椭圆曲线密码体制传输层安全加密套件（Elliptic Curve Cryptography Cipher Suites for Transport Layer Security）

IETF RFC 5246 传输层安全协议（The Transport Layer Security Protocol）

IETF RFC 6690 受限 RESTful 环境链接格式（Constrained RESTful Environments Link Format）

3 缩略语、术语和定义

3.1 缩略语

下列缩略语适用于本文件：

CoAP	受限应用协议	Constrained Application Protocol
CON	需确认消息	Confirmable message
DTLS	数据包传输层安全协议	Datagram Transport Layer Security
ECDSA	椭圆曲线数字签名算法	Elliptic Curve Digital Signature Algorithm
HTTP	超文本传输协议	HyperText Transfer Protocol
ICMP	因特网控制报文协议	Internet Control Message Protocol
ID	账号	Identification
IETF	互联网工程任务组	Internet Engineering Task Force
IP	因特网协议	Internet Protocol
IPv4	互联网协议第 4 版本	Internet Protocol Version 4

ITU	国际电信联盟	International Telecommunications Union
M2M	机器与机器	Machine to Machine
MTU	最大传输单元	Maximum Transmission Unit
NON	不需确认消息	Non-confirmable message
REST	表征性状态转移	Representational State Transfer
SCTP	流控制传输协议	Stream Control Transmission Protocol
SMS	短信息服务	Short Message Service
SNI	服务器名称指示	Server Name Indication
TCP	传输控制协议	Transmission Control Protocol
TLS	安全传输层协议	Transport Layer Security
UDP	用户数据包协议	User Datagram Protocol
URI	统一资源标识符	Uniform Resource Identifier
XMPP	可扩展消息和出席信息协议	The Extensible Messaging and Presence Protocol

3.2 术语和定义

下列术语和定义适用于本文件：

3.2.1

端点 endpoint
CoAP 中的参与实体。

3.2.2

令牌 token
用于匹配单一申请与其响应的字段。

3.2.3

发送者 sender
消息的源端点。

3.2.4

接收者 recipient
消息的目的端点。

3.2.5

客户端 client
请求消息的源端点，响应消息的目的端点。

3.2.6

服务器 server

请求消息的目的端点，响应消息的源端点。

3.2.7

源服务器 origin server

给定资源所在或产生的服务器。

3.2.8

中介 intermediary

主要用于转发请求和中继响应，可能有缓存、名字空间转换、协议转换的功能。

3.2.9

代理 proxy

不实现特定的应用语义。根据代理在消息转发结构中的位置，一般有两种形式的代理：转发代理和反转代理。

3.2.10

转发代理 forward-proxy

由客户端选取的端点，用来代表客户端执行请求，做相应的转换工作。

3.2.11

反转代理 reverse-proxy

一个或多个服务器之间的端点，满足这些服务器的请求执行要求。与转发代理不同，客户端可能不知道反转代理的存在。

3.2.12

CoAP-to-CoAP 代理 coAP-to-CoAP proxy

从 CoAP 请求映射到 CoAP 请求的代理。

3.2.13

协议转换代理 cross-proxy

在多个协议之间进行转换的代理。

3.2.14

捎带响应 piggy-backed response

包含在 CoAP ACK 消息当中。

3.2.15

资源发现 resource discovery
客户端向服务器请求其拥有的资源的过程。

4 业务特征

4.1 概述

CoAP 协议是一种满足受限环境特殊要求的通用 web 协议，其应用领域集中在能源、建筑和其他 M2M 应用。CoAP 协议并不是对 HTTP 协议的简单压缩，而是符合 REST 要求的 HTTP 协议的子集，该子集适用于 M2M 应用。尽管 CoAP 协议是简化了的 HTTP 协议，但其更重要的作用是提供 M2M 需要的内置发现、组播支持和异步通信的功能。

本标准定义了受限应用协议（CoAP），该协议可以直接转换为 HTTP 协议从而方便与现有 web 进行整合，该协议满足 M2M 对组播、低负载和精简的要求。

CoAP 的交互模型类似于 HTTP 协议的客户端/服务器模型。而 M2M 交互要求 CoAP 实体扮演客户端和服务端两种角色。CoAP 请求相当于 HTTP 请求，是由客户端发送给服务器的消息，该消息用于请求一个动作（方法编号）或者资源（URI）。服务器返回响应码，该响应可能包括资源状态转移。

CoAP 在处理异步交互时使用诸如 UDP 协议这类的面向数据包的传输方式。这种消息一般是依靠支持可靠传递（可选）的消息层来实现。CoAP 协议定义了四种消息：需确认消息（Confirmable），不需确认消息（Non-confirmable），ACK 消息（Acknowledgement），重置消息（Reset）；方法码和响应码包含在上述消息当中，携带请求或响应消息。这四种类型消息的基础交互与请求/响应交互正交，请求消息可以在需确认消息和不需确认消息中携带，而响应消息可以在 ACK 消息中捎带。

4.2 消息模型

CoAP 消息模型建立在端点之间基于 UDP 的消息传递的基础上。

CoAP 协议使用定长二进制头（4 字节），之后可以增加精简二进制可选项和负载。CoAP 消息格式在第五章中定义。每个消息都包含消息 ID，用于消息重复检测和可靠性选项。

可靠性保证是通过将消息标记为需确认消息（CON）来实现的。需确认消息在收到 ACK 消息之前按默认定时器重传，重传时使用相同的 MessageID。其流程如图 1 所示。

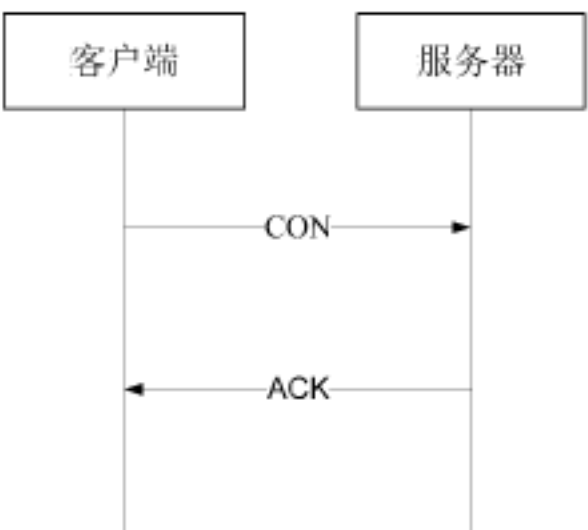


图 1 可靠消息传递

如果接收方无法处理需确认消息（甚至不能回复错误的 ACK 消息），则回复一个 RST 消息替代 ACK 消息。

当消息不需要可靠传递时，虽然不需要 ACK 消息，但是 MessageID 仍是必不可少的。其流程如图 2 所示。当接收方无法处理该消息时，则回复一个 RST 消息。

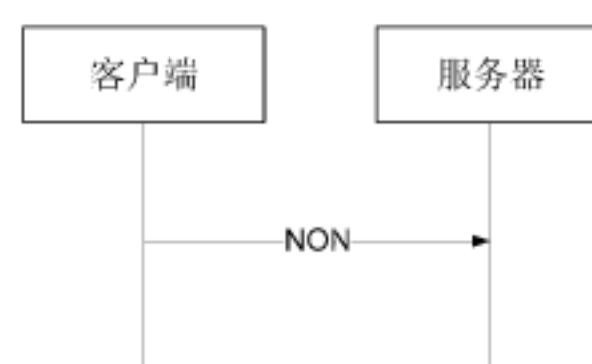


图 2 不可靠消息传递

CoAP 协议基于 UDP 协议，所以其支持组播。在第 10 章当中介绍具体的组播处理。

第 11 章当中定义了安全模型，范围从无安全保护到基于信用的安全保护。

4.3 请求/响应模型

CoAP 消息携带请求和响应消息语义，或者是方法代码或者是响应代码，其中可选的请求或响应信息，如 URI 和负载媒体类型都被看作是可选项。在底层消息中，单个请求消息与其响应很难做到一一对应，这时就引入了令牌（Token），令牌的作用是从底层消息中匹配请求和响应。令牌与 MessageID 不同，MessageID 是消息的唯一标识，而令牌则用于匹配请求与响应。

请求消息由需确认消息或者不需确认消息承载，对应的响应消息由结果 ACK 消息承载。这种形式称之为捎带消息，详见 7.3。（不需要独立的 ACK 消息来传递捎带消息，因为如果 ACK 消息丢失，客户端会重新发起一次请求。）以下是两个 Get 请求，以及捎带消息响应，流程如图 3 所示。

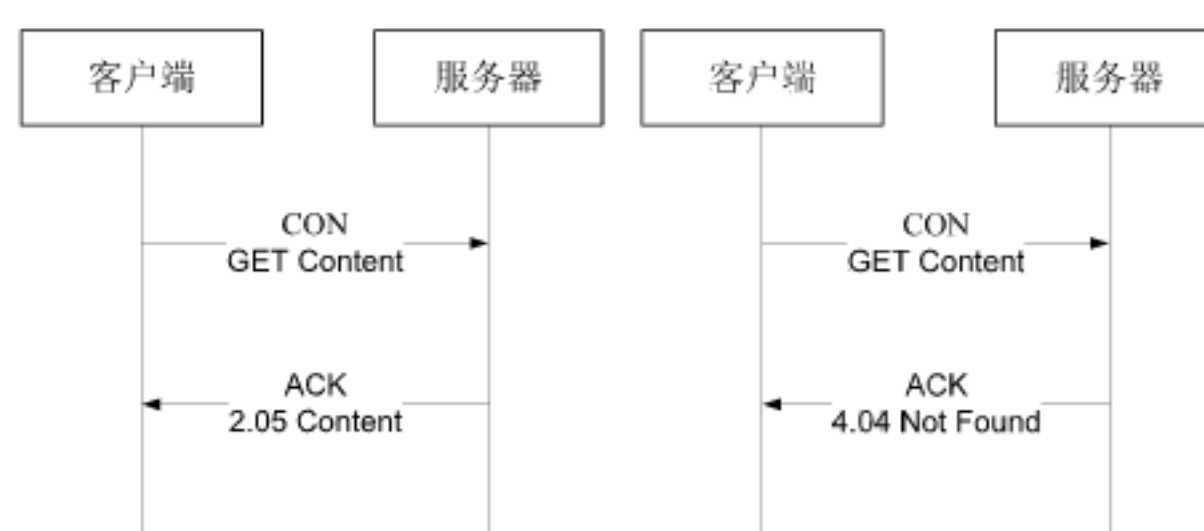


图 3 需确认消息中的捎带响应

如果服务器不能及时响应一个需确认消息请求，其只需要回复一个空的 ACK 消息，通过这种方式客户端可以停止发送该请求。响应消息处理完成之后，服务器会发起一个新的需确认消息（这时需要客户端回 ACK 消息）。一般称呼这种情况为分离消息（separate response），其流程如图 4 所示。详见 7.3。

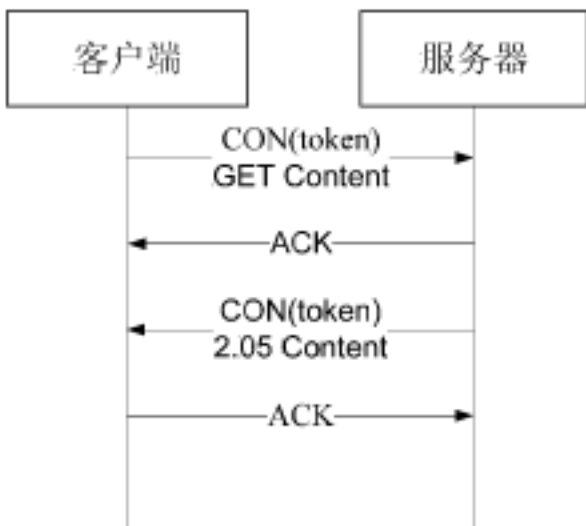


图 4 需确认消息中的分离消息

如果请求消息是以不需确认消息的形式发出的，其响应消息也要以不需确认消息来发送。流程如图 5 所示。

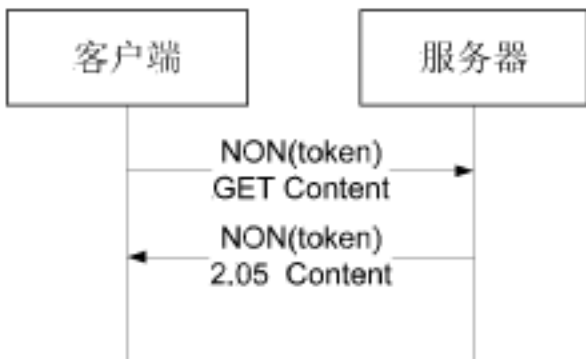


图 5 不需确认消息的请求与响应

CoAP 使用 GET，PUT，POST 和 DELETE 方法，其语义定义见 7.9。

在这四种基础方法上建立的其他方法可以添加到 CoAP 协议当中。新的方法不需要以请求、响应对的形式出现。单一的请求消息也可能有多个响应消息，比如组播请求。

URI 支持在服务器中简化，因为客户端已经解析了 URI 并且将其分解为主机、端口、路径以及查询组件，利用默认值来提高效率。响应代码由 HTTP 响应代码的子集和 CoAP 定义的响应代码组成，其定义见 7.10。

4.4 中介与缓存

协议支持响应缓存以满足请求的效率要求。简单缓存使 CoAP 响应中携带的信息新鲜、有效。缓存可以在端点中，也可以在中介当中。缓存功能见 7.7。

代理在受限网络当中有很重要的意义，包括网络流量限制、性能增强、接入休眠设备等方面。协议支持由代理代表其他 CoAP 端点发送请求。当使用代理时，请求中包含请求资源的 URI，目的地址填写代理的地址。7.8 中有代理功能的详细描述。

CoAP 协议使用 REST 风格设计，因此其外在功能与 HTTP 相似，所以 CoAP 和 HTTP 的互相映射具有天然优势。这种映射可以用基于 CoAP 的 HTTP REST 接口实现，或者在 HTTP 与 CoAP 之间转换。这种转换可以利用协议转换代理实现，其功能是转换方法、响应代码、媒体类型及其他可选项到相应的 HTTP 特征。第 12 章介绍了 HTTP 映射。

4.5 资源发现

在 M2M 应用中，资源发现是很重要的功能。第 9 章中讨论这个问题。

5 消息格式

5.1 基本格式

CoAP 协议是建立在基于 UDP 的压缩消息交互基础上的。CoAP 应用了数据包传输层安全协议 (DTLS)。CoAP 也可以使用 SMS, TCP 或者 SCTP 来作为传输协议, 但这些应用超出了本文的范围。

CoAP 消息以简单二进制形式编码。其消息格式以定长 4 字节消息头开始。之后是 0~8 字节的可变长的令牌值。之后是一排 0 或者可选的类型、长度、值格式 (TLV, Type-Length-Value), 如有负载, 负载将占据数据包的其他部分。消息格式如图 6 所示。

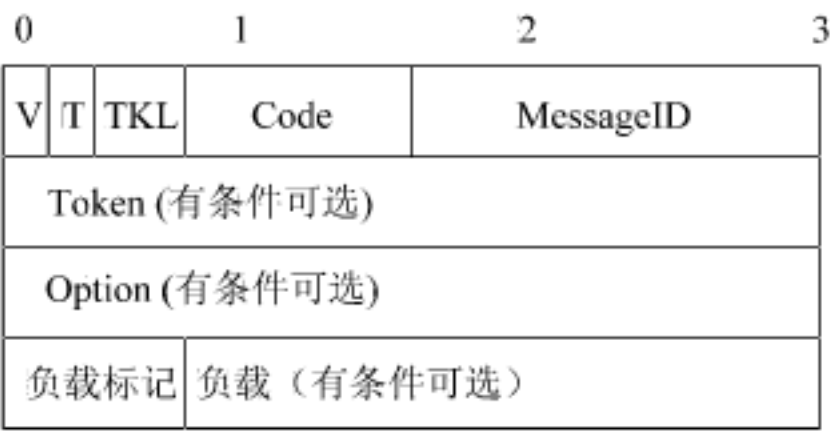


图 6 消息格式

图 6 所示的消息头定义如下：

版本 (V)：2bit 无符号整数。指出了 CoAP 的版本号。如果执行本标准，该参数应为 1。其他值为保留值。该字段不正确时，消息直接予以抛弃。

类型 (T)：2bit 无符号整数。消息如果为需确认消息该值为 0，如果为不需确认消息该值为 1，ACK 消息该值为 2，RST 消息该值为 3。消息类型的语义见第 7 章。

令牌长度 (TKL)：4bit 无符号整数。指出了可变长令牌的长度。其中 9~15 位为保留字段，不可发送，如果接收到按消息格式错误处理。

编码 (Code)：8bit 无符号整数，前三位为类型，后 5 位为内容。如记录入 c.dd 时，c 为前三位代表的 0~7 之中的一个，dd 为后五位代表的 00~31 中的一个。类型中，0 代表请求，1 代表成功响应，4 代表客户端失败响应，5 为服务器失败响应，其他值为保留值。特殊情况，0.00 为空消息。消息为请求消息时，其值为内容请求方法；消息为响应消息时，其值为响应代码。所有可能的值见 CoAP 代码注册表。请求、响应的语义见第 7 章。

Message ID：16bit 无符号整数，以网络顺序排序。用于检测消息是否重复，同时用来匹配需确认消息的 ACK 和不需确认消息的 RST 消息。产生与匹配规则见第 6 章。

消息头之后跟的是令牌值，0~8 字节长，由令牌长度字段定义。令牌值用来匹配请求、响应消息。产生于匹配方式见 7.4。

消息头、令牌之后可以为全 0 或者其他可选项。可选项之后可以为消息结尾或者一个负载标记及负载。

消息头、令牌、可选项之后就是可选项负载。如果存在非零负载，其由一个固定前缀标记，1 字节的负载标记 (0xFF)，意味着可选项的结束和负载的开始。负载数据从标记开始直到 UDP 报文结尾。无负载标记意味着负载长度为 0。有负载标记而无负载应按照格式错误消息执行。

5.2 可选项格式

CoAP 定义了一些可以被包含在消息当中的可选实例。消息中的每个可选实例都指定了 CoAP 可选项中定义的可选项数量，可选项长度和可选项本身。

与直接定义可选项数不同，实例必须以可选项数+增量的形式顺序出现：可选项数的计算方法是增量加上消息运行的实例的可选项数。消息中的第一个实例，其计算可选项数时增量为 0。用增量为 0 的方式同一个可选项可以包含多个实例。

可选项数保存在 CoAP 可选项数注册表当中。7.5 当中定义了其语义。可选项格式如图 7 所示。

Option 增量	Option 长度	1 byte
Option 增量（扩展）		0~2 byte
Option 长度（扩展）		0~2 byte
Option 值		0或多 byte

图 7 可选项格式

图 7 所示的可选项字段定义如下：

增量：4bit 无符号整数。其值为 0~12。其中三个值保留为特殊构造字段：

- 13：8bit 无符号整数。初始字节之后的 8bit 无符号整数，其增量减去 13；
- 14：16bit 无符号整数。初始字节之后的按网络字节顺序的 16bit 无符号整数，其增量减 269；
- 15：为负载标识保留。如果该值已设置但是非负载标识，应按照格式错误处理。

结果增量是本可选项的可选项数与之前可选项的不同之处，可选项数是前一个可选项数加上增量计算得来的。

可选项长度：4bit 无符号整数。可选项值的长度（字节），其值为 0~12。其中三个值保留为特殊构造字段：

- 13：8bit 无符号整数。在可选项值之前，表示其可选项长度减去 13；
- 14：16bit 无符号整数。在可选项值之前按网络顺序排列，表示其可选项长度减去 269；
- 15：保留位。如果该值已设置，应按照格式错误处理。

值：完全的可选项长度（字节）序列。可选项值的长度和格式取决于可选项本身，可以是不同的长度值。本标准应用格式见 4.3。其他标准定义的可选项可以用作其他可选项值格式。

5.3 可选项值格式

本标准中定义的可选项应用以下可选项值格式。

空：长度（字节）为 0 的序列。

不透明：未知长度（字节）的序列。

单元：代表用可选项长度字段值排序的非负整数。

可选项定义可以定义一组授权字节号码；如果可选，发送方应用尽可能少的字节来表示该号码，如，不带引导的 0 字节。例如，数字 0 表示空可选项值，数字 1 用一个数值为 1 的字节代表(00000001)。接收方应能够正确执行引导的 0 字节消息。

开发提示：在模板中用固定大小可选项的模板实现（硬件实现）时，会严格限制发送方异常行为。

字符串：Unicode 字符串以 Net-Unicode （见 IETF RFC 5198）格式用 UTF-8 编码（见 IETF RFC 3629）。

在 CoAP 协议中使用 UTF-8 时，字符串编码可以直接使用，可以比照 CoAP 协议不透明字符串实现。在 CoAP 实现中，不需要执行标准化（除非外部导入的未标准化的非 CoAP 协议 Unicode 编码）。ASCII 字符串（不带特殊控制字符）一般都是有效的 UTF-8 Net-Unicode 字符串。

6 消息传送方式

6.1 基本方式

CoAP 消息在 CoAP 端点间异步交互。用第 7 章中定义的请求和响应消息进行传输。

CoAP 基于不可靠的诸如 UDP 之类的协议，CoAP 消息可能出现乱序、重复或者丢失的情况。因此，CoAP 实现了一种轻量级的可靠性机制，出错时不需要像 TCP 那样重传整个消息。其拥有以下的一些特性：

- 需确认消息的简单的停-等可靠重传机制和指数退避机制。
- 需确认消息与不需确认消息的重复检测。

6.2 消息与端点

CoAP 端点是 CoAP 消息的起点或终点。端点的具体定义取决于 CoAP 使用的传输方式。本标准中使用的传输方式下，端点的身份由安全模式判断（第 11 章）：没有安全措施情况下，端点通过 IP 地址和 UDP 端口来判断。在其他安全模式下，身份由安全模式判断。

消息类型由 CoAP 消息头中的类型字段判断。

与消息类型相区别，消息可以承载请求、响应或者空消息。其通过消息头中的请求/响应码字段来发送信号，并与请求/响应模型有关。其可能的取值见 CoAP Code 注册表。

空消息的 Code 字段为 0.00。令牌长度应设置为 0 并且 Messageid 之后不可以有数据。如果有，则按照消息格式错误处理。

6.3 消息可靠传送

消息的可靠传输是依靠在 CoAP 消息头中将消息设置为需确认消息来实现的。需确认消息总是携带者请求或者响应消息，除非在有空消息引起的 RST 消息的情况下。接收方应用 ACK 消息来响应一个需确认消息，如果没有相应的上下文消息来进行处理（空消息，使用保留响应码 1、6、7，消息格式错误），则应予以拒绝；拒绝需确认消息可以发送匹配的 RST 消息或者直接予以拒绝。ACK 消息应使用需确认消息的 MessageID，且应携带响应消息或为空。RST 消息应使用需确认消息的 MessageID，且应为空。拒绝 ACK 消息或者 RST 消息采用直接忽略的方式。接收方不响应 ACK 消息或者 RST 消息。

发送方在未接到 ACK 消息（RST 消息）或重发定时器未超时时会一直重发消息，重发时间间隔以指数级增长。

在 CoAP 端点未收到 ACK 时，应跟踪重传其发送的需确认消息，这由两个参数控制，一个是定

时器，一个是重发计数器。对于少数需确认消息，其初始定时器为 `ACK_TIMEOUT` 和 $(\text{ACK_TIMEOUT} * \text{ACK_RANDOM_FACTOR})$ 之间的随机数，重发计数器设置为 0。当定时器被触发且计数器不超限时，会进行重发，这时计数器加 1 且重发间隔翻倍。如果重发计数器超限，端点会接收到一个 `RST` 消息，重发消息被取消，应用会进行报错。另一方面，如果端点按时接收到了 `ACK` 消息，则认为传输成功。

本标准对用于实现上述二进制指数退避算法的时钟精度没有做强制要求。在某些情况，端点因为休眠日程可能会延迟接到重传消息，但可以在再次重发时收到消息。尽管如此，其最小时间间隔仍然是 `ACK_TIMEOUT`，整个序列都应在 `MAX_TRANSMIT_SPAN` 范围内，尽管这可能使发送者丧失一次重传的机会。

发送需确认消息的 CoAP 端点可以不再等待 `ACK` 消息，就算在未达到 `MAX_RETRANSMIT` 计数器值之前：举例来说，应用取消了请求，所以其不再需要响应，或者通过其他途径获知需确认消息已经送达。特别的，CoAP 请求消息可以引出单个响应消息，这就意味着只有 `ACK` 消息丢失了，这时再重传请求消息毫无意义。尽管如此，响应者绝不能依靠请求者的这种越层行为。就算是需确认消息已经告知了请求者，如果需要的话也应保持向请求回复 `ACK` 的状态。

放弃重传的另一个原因可能是 `ICMP` 错误。如果打算考虑 `ICMP` 错误以防止欺骗攻击，则在实现时应在 `ICMP` 消息中仔细检查源数据包的信息，包括端口号和 CoAP 消息头信息如消息类型、消息码、`MessageID` 和令牌；如果由于 `UDP service API` 的限制使这种方式不可行，`ICMP` 错误应被忽略。除此之外，还应引入 `path MTU discovery`（见 IETF RFC 4821）算法。主机、网络、端口或协议不可达错误，在妥善检查之后也可以用来通知应用报错。

6.4 消息不可靠传送

一些消息不需要通知。这尤其适合那些周期性重复的应用需求，如重复读取一个传感器的数据时，个别消息是否送达无很大的意义。

作为一个更轻量的替代，在降低可靠性的条件下，一个消息可以标记为不需确认消息。不需确认消息或者是请求或者是响应，但绝不能为空消息。不需确认消息不需要由接收方确认。如果接收方缺少执行该消息的上下文，直接对该消息进行拒绝；拒绝方式是发送一个与收到消息匹配的 `RST` 消息，如果找不到 `RST` 消息对应的消息，则拒绝消息应予以忽略。

在 CoAP 层面，发送者无法得知不需确认消息送达与否。发送方可以选择在 `MAX_TRANSMIT_SPAN` 范围内多发几次消息，或者由网络重发。为了使接收者一个消息只执行一次，不需确认消息也设置了 `MessageID`。

总结下本节及上一节，可用的四种消息见表 1。表 1 中*意味着该组合一般操作中不常用，只用作 `RST` 消息。

表 1 消息类型的用途

	CON	NON	ACK	RST
REQUEST	X	X	—	—
RESPONSE	X	X	X	—
EMPTY	*	—	X	X

6.5 消息相关性

ACK 消息和 RST 消息与需确认消息和不需确认消息通过 MessageID 及附加相应端点地址信息产生相关性。MessageID 是 16 位无符号整数，位于需确认消息和不需确认消息的消息头中。消息接收方回复的 ACK 消息和 RST 消息的 MessageID 应与收到的消息一致。

相同 MessageID 在 EXCHANGE_LIFETIME 时间内不可以重用。

开发提示：产生 MessageID 可以使用多种策略。CoAP 端点可以保持一个 MessageID 变量，每次产生一个消息变量都会增加，不管消息发往哪里。处理消息量大的端点可以保持多个 MessageID 变量，通过字首或目的地址区分。强烈推荐其初始值随机产生，以免被断路攻击。对于 ACK 和 RST 消息，其消息应与之前的消息匹配。

6.6 数据重复

接收方可能会在一个 EXCHANGE_LIFETIME 周期内接收到多个一样的需确认消息（MessageID 与源端点一致），例如，当消息未超时发生 ACK 消息丢失或未送达消息发送方。接收方应用同样的 ACK 消息或者 RST 消息回复，但只执行一次请求或响应消息。在需确认消息传输幂等请求或者以幂等形式处理时该规则无效。重复消息无效示例如下：

- 服务器可以无视请求来回答所有的同意响应的幂等重传请求，因此其不需要记住任何 MessageID。例如，在执行成本低时，一个实现可能想当做独立请求来重复执行的 GET，PUT 和 DELETEA 请求。
- 功能不强的服务器可能忽略这种请求。例如，POST 请求结果只是服务器的临时状态，请求执行多次时成本要比跟踪器原有消息低。
- 接收方可能会在一个 NON_LIFETIME 周期内接收到多个一样的不需确认消息（MessageID 与源端点一致），作为一般规则这种消息可以被忽略，其请求或响应消息只应被执行一次。

6.7 消息大小

因为使用了特殊的链路层，所以 CoAP 协议应保证其消息足够小以适应链路层的数据包，这对实现来说是一个挑战。CoAP 协议只规定了数据的最大上限。大小超过 IP 数据包的消息将导致不理想的碎片。CoAP 消息应保持小于单个 IP 包的大小，且能够放进独立的 IP 数据报文。如果目的地的 Path MTU 是未知的话，则 IP MTU 将假设为 1280 字节；如果消息头大小未知，比较好的消息大小上限为 1152 字节，其中负载的大小上限为 1024 字节。

开发提示：CoAP 中应用的消息大小格式适用于 IPv6 和当今绝大多数的 IPv4 路径。在受限网络环境中更重要的分块 fragmentation 类型发生在适配层；在接近 three-digit 消息大小时，这驱使开发者减少包的大小，转而使用 block-wise 传输。

在受限节点上的开发中，消息大小也是被考虑的重要因素。在许多实现方式中，需要先申请一块缓冲区来接收信息。如果应用的能力不足以提供这个缓冲区，可以使用下面的这种实现方式（不使用 DTLS）：当接收的消息大于已划定的缓冲区的时候，可以探测并抛弃消息的尾部，只接收消息的头部。因此，即便不是所有负载，至少 CoAP 消息头和可选项也会存储在缓冲区中。如果负载不全，则服务器可以中断并回复一个 4.13 的响应码。服务器发送一个幂等的请求并接收到一个超大小的响应，可以重新发起请求，在请求中调整到合适的 Block Option。

6.8 拥塞控制

CoAP 协议基本的拥塞控制是由指数退避机制实现的，其具体细节见 6.3。

为了避免拥塞，客户端（包括代理）应严格限制针对特定服务器的并发未完成交互数量（其数量小于 NSTART 值）。

未完成交互或者是一个在等待 ACK 的需确认消息，或者是收到 ACK 了还没收到响应消息。初始 NSTART 值为 1。

进一步的拥塞控制机制还需要进一步研究，就如 6.9 中提到的自动初始化 CoAP 参数传送一样，这种情况下，NSTART 值可以大于 1。

在 EXCHANGE_LIFETIME 时间内如果收不到 ACK 消息，客户端将不再等待需确认消息的响应。具体的停止等待方式算法不在本标准当中定义。除非额外的拥塞控制机制有其他调整，端点向另一个不回响应的端点的消息发送速率应不能超过 PROBING_RATE 的平均速率。

提示：CoAP 协议当中，拥塞控制主要由客户端完成。当然，客户端可能发生故障或被攻击，为了控制这种风险，服务器应在合理假设的基础上限制某些应用请求的响应速率。这种限制只针对行为异常的端点。

6.9 传送参数

6.9.1 参数表

消息传递的控制主要由表 2 中的参数完成。

表 2 CoAP 协议参数

名称	默认值
ACK_TIMEOUT	2s
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
NSTART	1
DEFAULT_LEISURE	5s
PROBING_RATE	1 字节/s

6.9.2 参数替换

ACK_TIMEOUT, ACK_RANDOM_FACTOR, MAX_RETRANSMIT, NSTART, DEFAULT_LEISURE 和 PROBING_RATE 的值可以由应用环境配置，但其配置方式不在本标准当中定义。建议应用环境使用固定值参数，采用不固定值的操作方式超出本标准范围。

在互联网中，传递参数用于拥塞控制。如果某架构想用不同的值，则这个架构需要保证拥塞控制属性不冲突。IETF RFC 5405 中规定，ACK_TIMEOUT 不能小于 1s。

CoAP 在设计上允许实现不使用 round-trip-time（RTT）方法。但仅在使用这种方式时，实现才可以明显的减少 ACK_TIMEOUT 或者增加 NSTART 值。

在不能保证拥塞控制时，配置不能够减少 ACK_TIMEOUT 或者增加 NSTART 值。ACK_RANDOM_FACTOR 不能够小于 1，为了提供同步效果其值应不同于 1。

MAX_RETRANSMIT 可以自由调整，但是太小的值将减少需确认消息的自动接收，但是太大的值又需要对时间值进行调整。

如果传递参数的选取导致了提取时间值的增加，那么这种配置方式应保证这个调整过的值用于所有其需要通信的端点。

6.9.3 参数传递中的时间值提取

ACK_TIMEOUT, ACK_RANDOM_FACTOR 和 MAX_RETRANSMIT 的组合会影响重传的时间，这就影响了实现中对消息信息的保存时间。为了能够无二义性地引用这些提取时间值，以下是一些值的产生方式描述如下：

- MAX_TRANSMIT_SPAN 是从需确认消息第一次发送到最后一次重传的最大时长。对于默认参数，其数值为 $(2+4+8+16) \times 1.5 = 45\text{s}$ ，其算法为：

$$\text{ACK_TIMEOUT} \times ((2^{\text{MAX_RETRANSMIT}}) - 1) \times \text{ACK_RANDOM_FACTOR}.$$

- MAX_TRANSMIT_WAIT 是从需确认消息第一次发送到发送者不再接收 ACK 或 RST 消息的最大时间间隔。对于默认传递参数，其值为 $(2+4+8+16+32) \times 1.5 = 93\text{s}$ ，其算法为：

$$\text{ACK_TIMEOUT} \times ((2^{\text{MAX_RETRANSMIT}+1}) - 1) \times \text{ACK_RANDOM_FACTOR}.$$

另外，需要对网络和节点的特征做一些假设。

MAX_LATENCY 是等待一个数据包从开始传输到消息送达的最大时间间隔。这个数值与 IETF RFC 0793 中定义的 MSL (Maximum Segment Lifetime) 相关，定义为 2min。注意，这个值没必要比 MAX_TRANSMIT_WAIT 小，因为 MAX_LATENCY 并不是描述协议工作正常时的状态，而是协议设想的最坏的情况。在这里，我们将该值定义为 100s。

PROCESSING_DELAY 是处理需确认消息产生 ACK 的时间。我们假设节点试图在消息超时前处理 ACK，因此保守估计将其值设置为 ACK_TIMEOUT。

MAX_RTT 是消息往返的最大时间，其算法为 $(2 \times \text{MAX_LATENCY}) + \text{PROCESSING_DELAY}$ 。

从这些值，我们可以提取出与协议操作有关的如下值：

EXCHANGE_LIFETIME 是从发送一个需确认消息到结束等待 ACK 的时间。EXCHANGE_LIFETIME 包含 MAX_TRANSMIT_SPAN，发送 MAX_LATENCY，PROCESSING_DELAY，接收 MAX_LATENCY。注意，如果配置已选取最后等待期为 $(\text{ACK_TIMEOUT} \times (2^{\text{MAX_RETRANSMIT}}))$ 或者 MAX_TRANSMIT_SPAN 和 MAX_TRANSMIT_WAIT 之间的差小于 MAX_LATENCY，则没必要考虑 MAX_TRANSMIT_WAIT。在这种情况下，EXCHANGE_LIFETIME 定义为：

$$\text{MAX_TRANSMIT_SPAN} + (2 \times \text{MAX_LATENCY}) + \text{PROCESSING_DELAY}.$$

默认情况下该值为 247s。

NON_LIFETIME 是从发送一个不需确认消息到该消息 MessageID 可以安全重用的时间。如果没用到多路传输，其值设置为 MAX_LATENCY，或者 100s。但为了实现组播，CoAP 发送者可能会发多个不需确认消息。这个周期不在本标准中定义，接收者判断消息是否重复的时间为 MAX_TRANSMIT_SPAN。

因此，可以使用值：

MAX_TRANSMIT_SPAN + MAX_LATENCY

在使用默认值时为 145s；如果实现只想用一个超时值来重用 messageID，则其 EXCHANGE_LIFETIME 需要调整的大一些。

表 3 为本条中介绍的参数默认值。

表 3 提取协议参数

名称	默认值
MAX_TRANSMIT_SPAN	45s
MAX_TRANSMIT_WAIT	93s
MAX_LATENCY	100s
PROCESSING_DELAY	2s
MAX_RTT	202s
EXCHANGE_LIFETIME	247s
NON_LIFETIME	145s

7 请求/响应语义

7.1 概述

CoAP 操作是一个类似 HTTP 操作的请求/响应模型：作为“客户端”的 CoAP 端点向“服务器”端点发送一个或者多个 CoAP 请求，“服务器”端点通过发送 CoAP 响应的方式对 CoAP 请求提供服务。与 HTTP 请求不同，请求和响应不是通过先前建立的连接发送的，而是通过 CoAP 消息异步交换的。

7.2 请求

CoAP 请求包括：应用到资源的方法、资源的标识、负载（payload）和网络媒体类型（如果有的话）以及关于该请求的可选的元数据。

CoAP 支持基本的方法：GET、POST、PUT 和 DELETE，这些方法很容易映射到 HTTP。CoAP 和 HTTP（见 IETF RFC 2616 的 12.2）有相同的安全属性（仅检索）和幂等（多次调用得到的效果相同）。GET 方法是安全的，因此不必对资源采取检索以外的操作。GET、PUT 和 DELETE 方法应以幂等的方式执行。POST 方法不是幂等的，因为它的效果由原始服务器决定并且依赖于目标资源，它通常会导致一个新的资源被创建或者目标资源被更新。

请求是通过设置可确认（CON）或不可确认（NON）信息的 CoAP 报头中的代码域和包含请求信息来进行初始化的。

请求中所使用的方法将在 7.9 中进行详细描述。

7.3 响应

7.3.1 响应代码格式

接收到请求并且进行解释之后，服务器返回一个 CoAP 响应消息，该响应消息通过客户端发送的请求中的令牌进行匹配（见 7.4，注意，这不同于用于可确认消息与其确认消息之间进行匹配的消息 ID）。

响应是通过 CoAP 报头中的代码域来确认的，该代码域将被设置为响应代码。类似于 HTTP 状态代码，CoAP 响应代码表明服务器试图理解并且满足请求。这些代码将在 7.10 中进行定义。CoAP 报头中的代码域所设置的响应代码值将由 CoAP 响应代码注册表进行维护。

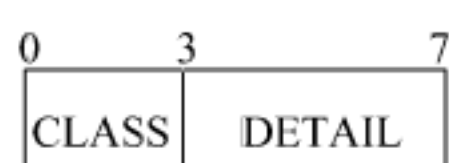


图 8 响应代码的结构

图 8 所示的 8 位响应代码中，高 3 位用于定义响应的分类，低 5 位不用于分类，而是针对整体分类给出更多的细节内容。

为了提高规范和协议诊断的可读性，包括响应代码在内的 CoAP 代码采用“c.dd”的格式表示，“c”是用于标识类别的一位十进制数，“dd”是用于标识细节信息的两位十进制数。例如，“Forbidden”表示为 4.03，4.03 转换成 8bit 值之后，十六进制值是 0x83 ($4 \times 0x20 + 3$)，十进制值是 131 ($4 \times 32 + 3$)。

CoAP 协议定义了 3 类的响应代码：

2-成功：请求已被收到、理解和接收。

4-客户端错误：请求的语法错误或无法实现。

5-服务器错误：服务器无法完成一个明显合法的请求。

响应代码的设计是可扩展的：如果响应代码中的客户端和服务器错误不能被端点识别的话，将被视为与该类型相同的通用响应代码的类（分别为 4.00 和 5.00）。然而，CoAP 中没有通用的响应代码来标识成功，所以未被端点识别的成功识别的响应代码类只能用于确定请求已经成功，但是没有任何进一步的细节。

可能的响应代码是在 7.10 节中详细描述。

响应可以通过多种方式发送，在下面的章节里会做进一步的定义。

7.3.2 捎带消息 (Piggy-backed)

在最基本的情况下，响应直接通过回复请求的确认消息携带（这要求请求是可确认的消息类型 CON）。这称为“捎带”响应。

通过确认消息携带响应的过程不依赖于该响应标识的是成功或失败。因此，响应通过回复请求的确认消息携带，不需要单独的消息来返回响应。

具体实现需要注意的事项：协议将是否采用捎带方式返回响应（即是否发送单独的响应消息）留给服务器做决定。客户端应做好接收的准备。为了保证实现的质量，强烈建议服务器尽可能实现捎带功能的代码，以节省网络资源和客户端以及服务器的资源。

7.3.3 分离消息 (Separate)

并不是所有的情况下都可以通过捎带方式返回响应。例如，为了避免客户端重复发送请求消息的风险，服务器可能需要比它可以等待响应消息时间更长的时间来获得请求响资源的表征（也可见 6.9.3 关于 PROCESSING_DELAY 的讨论）。针对不可确认消息中的请求的响应通常是单独发送的（因为其没有确认消息）。

服务器实现这一方法可以采取如下方式：服务器开始尝试获取资源的表征，在此过程中，服务器让确认计时器超时。或者采用另外一种方式：服务器立即发送确认消息，并且提前知道其不“捎带 (piggy-backed)”响应。在这两种情况下，确认消息有效的承诺，请求稍后会被执行。

服务器最终获取资源表征之后，将发送响应。如果其期望该消息不会丢失，服务器将向客户端发送一个可确认消息，并且客户端将回复一个确认消息，并发回服务器所选择的新的消息 ID。（服务器也可能发送不可确认消息。详见 7.3.4）

当服务器选择使用单独消息返回响应时，它以空消息的形式发送确认消息以回应客户端发送的可确认请求。一旦服务器返回空的确认消息，即使客户端重新发送另一个相同的请求，它也不能通过发送另一个确认消息来捎带响应。如果接收到另一个重发请求（可能因为原来的确认消息被延迟），服务器会发送另一个空的确认，并且响应应通过一个单独的消息进行发送。

之后，如果服务器发送了一个可确认的响应，客户端对这个响应的确认应是一个空消息（不携带请求和响应）。服务器收到相匹配的确认后（忽略任何响应代码或有效负载）应立即停止传输或重置消息。

具体实现需要注意的事项：因为底层数据报文传输可能不是按顺序保存的，携带响应的可确认消息可能先于或后于请求的确认消息到达；为了停止重传的顺序，可确认消息也可作为确认消息。还要注意，CoAP 协议本身在这里并不做任何具体的要求，但是从应用程序的观点来看，还是期望响应能在预期的时限内到达；因为没有提供保活机制的底层传输协议，请求者可能希望设置一个与 CoAP 的重传计时器无关的计时器，以避免服务器宕机或者无法发送响应。

7.3.4 不需确认消息 (Non-confirmable)

如果请求消息是不可确认的，响应也应该以一个不可确认消息的形式返回。然而，端点 (endpoint) 应准备接收用于回复可确认请求的不可确认响应（在一个空确认消息之前或之后）或者回复不可确认请求的可确认响应。

7.4 请求/响应匹配

7.4.1 令牌 (Token)

不管如何发送响应，响应是通过客户端在请求中包含的令牌和附加的通信端点的地址信息来匹配响应的。

令牌用来匹配响应和请求。令牌值是 0~8 个字节的序列。（注意，每条消息携带一个令牌，即使其长度为 0。）每个请求携带一个由客户端生成的令牌，服务器在发送响应的时候不允许对令牌做任何修改。

令牌的目的是用于客户端在不与服务器验证的情况下区分并发的请求（见 7.3）；令牌也可称作“请

求 ID”。

客户端生成令牌时应保证在当前使用的令牌对于一个给定的源/目标端点是独一无二的。（注意，如果客户端每次请求使用不同的端点，例如不同的源端口号，客户端实现可以使用同样的令牌。）空令牌值在如下情况是合适的，如：一个目的地址没有其他的令牌在使用，或者针对每个目的地址按串行的方式发送请求和接收“捎带”应答。然而完成这个目标存在多种可能的实现策略。

客户端不使用传输层安全协议（见第 11 章）发送请求时应该使用一个复杂的、随机的令牌来防止欺骗的响应。令牌之所以能起保护性作用是因为它允许达到 8 字节的大小。令牌采用的实际大小取决于客户的安全需求和欺骗的响应造成的威胁程度。连接通用互联网的客户端至少应使用 32 位的随机数；请记住，如果不直接连接到互联网，客户端不一定非要能够防范欺骗。（注意，消息 ID 在安全方面的作用很小，因为它通常是按顺序分配的，即可推测的，可以通过欺骗单独的响应绕过消息 ID。）客户端如果希望优化令牌长度，则需要进一步检测正在进行的攻击的等级（例如，通过分析最近到达的消息中的令牌不匹配的情况）和适当地向上调整令牌的长度。IETF RFC 4086 讨论了安全的随机性需求。

端点如果接收到不是其产生的令牌，应把它视为不透明的，并且不解析其内容或结构。

7.4.2 请求/响应匹配规则（Request/Response Matching Rules）

准确的请求/响应匹配规则如下：

- 1) 响应的源端点应与原始请求的目标端点相同；
- 2) 在“捎带”响应中，可确认请求的消息ID和确认的消息ID、响应的令牌和原始请求的令牌应匹配。在采用单独方式发送的响应中，只要求响应的令牌和原始请求的令牌应匹配。

假设消息携带的响应不是客户端期待的（通过地址所识别的端点不是客户端所期待的，或者响应没有给定的令牌），响应将被拒绝（6.2、6.3）。

具体实现需要注意的事项：

在 CON 消息中接收到响应的客户端可能需要在发送 ACK 之后清理其消息状态。如果 ACK 消息丢失并且服务器重新发送 CON 消息，客户端可能不再有任何状态与这个响应有关联，并且将重传的消息作为异常消息处理；客户端将发送一个“Reset”消息，因此它不会接收更多的重传消息。这种行为是正常的且并不是一个错误的信号。（如果客户端不去积极优化内存，其内存中则仍然具有消息状态，因此客户端将会将第二次发送的 CON 消息作为重传消息。如果客户端期待服务器发送的更多的消息从（参见 IETF I-D.ietf-core-observe），其将一直保持这种状态。）

7.5 可选项

7.5.1 可选项的定义

请求和响应都可能包含一个或多个选项的列表。例如，请求的 URI 通过多个选项传输，HTTP 协议中 HTTP 头部携带的元数据也是作为选项提供的。

CoAP 定义了一组用于请求和响应的选项：

- 内容格式（Content-Format）；
- 标签（ETag）；
- 定位路径（Location-Path）；

- 位置查询 (Location-Query) ;
- 内容能够被缓存的时间 (Max-Age) ;
- 代理 Uri (Proxy-Uri) ;
- 代理模式 (Proxy-Scheme) ;
- Uri 主机 (Uri-Host) ;
- Uri 路径 (Uri-Path) ;
- Uri 端口 (Uri-Port) ;
- Uri 查询 (Uri-Query) ;
- 接受 (Accept) ;
- 是否匹配 (If-Match) ;
- 如果不匹配 (If-None-Match) ;
- 大小 (Size1) 。

这些选项的语义以及它们的属性在 7.11 中详细定义。

并不是定义的选项在所有方法和响应代码都要使用。方法和响应代码可能的选项在 7.9 和 7.10 分别有定义。假如一个选项不存在于现有方法或响应代码的定义中，发送者发送的数据一定不会包含这个选项，而且其一定会被接收者当做未识别选项处理。

7.5.2 关键的和可选的 (Critical/Elective)

所有选项可分为两类：“关键的”和“可选的”。这些选项之间的不同点是当这些选项未被端点识别时该如何处理。

- 接收时，未被识别的“可选的”选项应被静默忽略。
- 出现在可确认请求的未被识别的“关键的”选项应引起 4.02 (Bad Option) 响应的返回。这个响应应该包括一个诊断负载以描述未识别的选项。
- 出现在可确认响应或者确认的“捎带”消息中的未被识别的“关键的”选项应引发响应被拒绝（见 6.2）。
- 出现在不可确认消息中的未被识别的“关键的”选项一定会引发消息被拒绝（见 6.3）。

注意，无论是“关键的”还是“可选的”选项，选项都不是必须的（选项始终是可选的）：这些规则的定义目的是使得实现在不理解选项或者没有实现选项的时候能够停止对相应选项的处理。

关键的/可选的规则应用在没有代理的端点上。基于非安全/安全转发类的代理进程选项在 7.8 中定义。

7.5.3 代理非安全/安全转发和 NoCacheKey (Proxy Unsafe/Safe-to-Forward and NoCacheKey)

除了将选项分为关键的或可选的两类以外，选项也可以根据代理是如何处理选项的方式进行分类。为此，该选项可以被视为非安全转发 (Unsafe 被设置)，也可以被视为安全转发 (Unsafe 被清除)。

另外，对于一个被标记为安全转发的选项，在请求中的选项数值表明它是否要成为缓存密钥 (CacheKey) 的一部分（见 7.7）。如果无缓存密钥 (NoCacheKey) 的部分比特为 0，则选项是缓存密钥的一部分；如果无缓存密钥的所有比特均为 1，则选项不是缓存密钥的一部分。

注意：缓存密钥只与没有将给定的选项实现为请求选项、反而只依赖于非安全/安全转发的代理相

关。举例来说，对 ETag，实际上将请求选项作为缓存密钥的一部分是非常效率低下的，但如果代理没有实现 ETag，其是能选择的最好方法，因为响应将根据请求的选项的不同而有所不同。如果代理实现了 ETag 请求选项，将不使用 ETag 作为缓存密钥的一部分。

NoCacheKey 用 3bits 表示，因此 8 个代码（codepoints）中的 1 个可以用于标识 NoCacheKey，假设这是不太可能的情况。

关于这些分类的代理行为定义见 7.8。

7.5.4 长度

选项值被定义成一个特定的长度，通常采用上界和下界的形式。如果请求的选项值的长度超出定义的范围，该选项应被当作一个无法识别的选项（见 7.5.1）。

7.5.5 缺省值

选项可以定义为缺省值。如果该选项的值定义为缺省值，选项不应该包含在消息中。如果选项不存在，则应假定缺省值。

若关键选项有缺省值，缺省值通过如下方式选择：一条消息中的选项缺失可通过两种方式处理：一是无视关键选项进行实现；另一种是对这个选项的缺失解释为该选项的缺省值。

7.5.6 可重复的选项

一些选项的定义详细说明了这些选项是可重复的。一个可重复的选项可能在一个消息中包含一次或多次。一个不可重复的选项在一个消息中只能包含不超过一次。

如果消息包含一个比定义的选项出现次数的选项，每个额外选项出现，随后出现在消息中应被当作一个无法识别的选项（见 7.5.1）。

7.5.7 选项号码（Option Numbers）

选项是通过号码辨别的，选项号码还提供了一些额外的语义信息：例如，偶数表示的是可选的选项，而奇数则表示关键性选项。请注意，这不仅仅是一个惯例，它是一个协议的特点：选项是可选性的或关键性的完全由其选项号码是奇数还是偶数决定。

一般来说，选项号码使用 1 个比特掩码来表示选项是关键性的/可选的、非安全/安全转发以及在安全转发情况下的缓存密钥，如图 9 所示。接下来，比特掩码位可表示为一个单字节，该字节适用于选项号码的最低有效字节，该有效字节以无符号整数的形式表示。当第 7 位（最低有效位）是 1 时，选项是关键性的（同样，为 0 时是可选的）。当第 6 位是 1 时，选项是不安全的（同样，为 0 时是安全转发）。当第 6 位是 0 时，选项是 Unsafe，当且仅当 3~5 位都被置为 1 时，它不是一个缓存密钥（NoCacheKey）；所有其他的位组合则意味着它是 1 个缓存密钥。选项的这些分类将在下一节中解释。

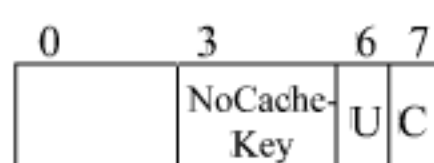


图 9 选项号码掩码（最低有效位）

端点可使用等价的 C 代码来导出选项号码“onum”的特性。

```
Critical = (onum & 1);
```

```
Unsafe = (onum & 2);
```

```
NoCacheKey = ((onum & 0x1e) == 0x1c);
```

7.6 负载与表示

请求和响应分别根据不同的方法或响应代码都可能包括有效负载。如果一个方法或响应代码没有定义负载，发送者不能包括负载，接收者应忽略它。

7.6.1 表征

请求或响应的用于指示成功的有效负载通常表示为一个资源(“资源表征”)或请求动作的结果(“操作结果”)。其格式是由互联网媒体类型指定的，内容编码则由 Content-Format 选项指定。如果没有选项，则不会指定缺省值并且格式需要由应用程序推断(例如，通过应用程序的上下文)。如果 Content-Format 没有给出，则需要尝试负载“嗅探”。

具体实现需要注意的事项：为了保证实现的质量，强烈期望在资源表征中尽可能给出 Content-Format。这不仅仅是一个“应该”级别的要求，因为它不是一个协议的要求，同时也很难描述究竟在什么情况下可以违反这种期望。

对于指示客户机或服务端错误的响应，只有给出 Content-Format 选项，负载才被认为是一个请求的动作的结果的表征。如果没有这个选项，负载是一个诊断负载(7.6.2)。

7.6.2 诊断负载

如果没有 Content-Format 选项，指示客户机或服务端错误的响应的负载是一种简洁的人类可读的诊断信息，该诊断信息解释了错误的情况。这种诊断消息应使用 utf-8 编码(见 IETF RFC 3629)，特别是使用 Net-Unicode 形式(见 IETF RFC 5198)。

该消息是类似一个 HTTP 状态行的原因描述。它的目的不是针对最终用户，而是针对软件工程师们，在调试过程中需要对其进行解析，因此不需要或不用提供任何语言标记。与往常的 HTTP 不同，如果没有超出响应代码的任何其他信息，有效负载应该是空的。

7.6.3 选择表征 (Selected Representation)

并非所有的响应都携带有效负载，用于表示请求所需要的资源表征。然而，它有时是有用，以便能够引用这样一个与响应相关的表征，而不依赖于其是否实际上是封闭的。

如果相应的请求是使用了 GET 方法，并排除任何条件请求选项，我们使用术语“选择表征”来指代在一个成功的响应中被选中的目标资源的当前表征(7.11.8)。

某些响应选项提供选择表征的元数据，这可能与包含在一些状态改变方法的响应消息中的表征不同。本标准中定义的响应选项中，只有 ETag 响应(7.11.6)选项被定义为已选择的表达元数据。

7.6.4 内容协商 (Content Negotiation)

服务器可能能够将多种表示格式之一提供为一个资源的表征。如果没有来自客户端的详细信息，

它将以它喜欢的格式提供表征。

使用请求的接收选项（7.9.4），客户端可以用于表示它愿意接受的内容格式。

7.7 缓存

7.7.1 缓存原则

CoAP 端点可以缓存响应，目的是减少将来的响应时间和网络带宽消耗，这也同样适用于请求。

在 CoAP 中进行缓存的目标是重新使用之前的应答消息来满足当前的请求。在某些情况下，一个已储存的响应可以在不需要网络请求的情况下被重新使用，从而减少了等待时间和网络来回传输；为此，“更新”机制可用于实现该目的（见 7.6.1）。甚至当一个新请求被需要时，经常可以重新使用之前的应答的有效载荷来满足请求，从而减少网络带宽使用；为此，“验证”机制可用于实现该目的（见 7.6.2）。

不像 HTTP，CoAP 响应的可缓存性不依赖于请求模式，而是响应码。每个响应码的可缓存性由 7.10 中的响应码定义所定义。每一个显示成功但是未被端点识别的响应码不会被缓存。

对于提出的请求，一个 CoAP 端点不能使用已储存的响应，除非：

- 提出的请求的方法和用来获得已储存的响应相匹配。
- 提出的请求的选择和用来获得储存响应的这些请求的选项相匹配（包括请求的 URI），除非选项被标记为 NoCacheKey（见 5.4）或被认为是缓存而按照缓存的行为进行解析，则不需要匹配（比如 ETag 选项，7.11.7，同样见 7.4.2）。
- 已储存的响应是新的或者被成功验证的，如以下定义。被用于匹配缓存入口请求选项的设置也同样被归类为“缓存密钥”。对于 URI 方案而不是 coap 和 coaps，这些组成请求 URI 的选项的匹配可能在 URI 方案特定的规则下执行。

7.7.2 新鲜度模型（Freshness Model）

当一个缓存中的响应是新的，它可以被用来满足随后的请求而不用连接源服务器，从而提高效率。

决定响应是否为新将由原服务器所定义的响应的时限来决定，这个时限可以通过 Max-Age 这个选项来定义（见 7.11.6）。Max-Age 选项表示的是如果响应存在的时间超过了特定的时限之后将不会被认为是新的响应。

Max-Age 选项缺省值是 60。因此，如果它没有出现在缓存响应中，在它的寿命超过 60s 之后，这个响应就会被认为不是新的。如果一个源服务器希望阻止缓存，它应明确的包括值为 0 的 Max-Age 选项。

如果一个客户端有了新的响应被存储，并且该客户端提出了新的请求且该请求匹配存储的响应所对应的请求，新的响应将使旧响应作废。

7.7.3 验证模型（Validation Model）

当一个端点有一个或多个存储的 GET 请求响应、但都无法使用时（例如因为他们不是新的），其可以使用 GET 请求中的 ETag 选项（7.11.7）给源服务器一个机会，让服务器选择一个存储的响应，并且更新该响应的时限。这个过程被称为对已存储的响应“验证”或“重新验证”。

当发送这样一个请求时，端点应该添加一个 ETag 选项来指明每个存储的响应的 entity-tag 是可用

的。

一个 2.03 (Valid) 响应表明其在更新之后可以被重新使用，该响应的 `entity-tag` 在响应的 `ETag` 选项中给出，如 7.11.1.3 所描述。

其他任何响应码则表明请求中提到的储存的响应则不可被重新使用。相反，响应应该满足请求而且可能要代替已存储的响应。

7.8 代理

7.8.1 代理原则

代理服务器可以用来代替 COAP 客户端执行请求。这种方法在某些情况下比较有效，例如，请求可能无法进行，或者为了减少响应时间和网络带宽或能量的消耗用缓存中的响应对请求进行回复。

在一个基于受限的 RESTful 环境的整体架构中，代理服务器可以为完全不同的用途提供服务。代理可以被客户端明确地选择，我们称之为“转发代理”。代理也可以代替源服务器，我们称之为“反向代理”。和这种分类相对应，一个代理可以将 COAP 的请求映射到一个 COAP 请求（COAP 到 COAP 代理）或实现 CoAP 协议和其他协议的翻译（“交叉代理”）。在 3.2 提供了这些术语的完整定义。

注意：在本标准的术语与在更广泛使用的 web 应用环境中使用的术语是兼容的，虽然不一定每一个细节都匹配（这可能与受限的 RESTful 环境没有关系）。没有太多的语义和这些术语的组成有关（如“前进”，“反向”或“交叉”）。

HTTP 代理，除了可作为 HTTP 代理，其还通常提供传输协议的代理功能（“CONNECT”），以通过代理实现终端到终端的传输层安全性。本说明书中，没有定义 COAP 到 COAP 代理的功能，因为在受限的 RESTful 环境下，UDP 数据包的转发不具有太多价值。

当客户端使用代理发送采用一种安全的 URI 方案（例如，`coaps` 或 `https`）的请求时，发送到代理的请求应该使用 DTLS 安全协议，除非客户端与代理之间采用了等价的底层安全协议。

7.8.2 代理操作 (Proxy Operation)

基于代理接收到的请求，它通常需要一种方法来确定发送到目的地址的请求的潜在请求参数。这种方式为转发代理做了详细说明，但可能依赖于反向代理的具体配置。特别是，反向代理的客户端通常并不为目的地址指示一个定位器，迫使反向代理需要有某种形式的名称空间翻译。然而，代理服务器操作的许多方面具有共同的形式。

如果代理不采用缓存，则它只是向目标地址转发翻译的请求。否则，如果它采用缓存，但是找不到一个存储的响应与翻译的请求相匹配，则认为该请求是刚到达的，根据 7.7，那么就需要刷新其缓存。对于代理服务器可以识别的请求的可选项，它知道是否该选项要用于充当被用来查询缓存值的密钥的一部分。例如，因为不同的 URI 路径值的请求定位不同的资源，URI 路径值总是缓存密钥的一部分，同时，令牌值则不是缓存密钥的一部分。对于代理不能识别、但被标记为 `Safe-to-Forward` 的选项号码的选项，该选项也表明它是否包含在缓存密钥中（如果所有的 `NoCacheKey` 都没有设置，则包含在缓存密钥中；如果 `NoCacheKey` 都被设置了，则不包含在缓存密钥中）。（无法识别的并且标记为非安全的选项将导致 4.02 错误选项。）

如果到目的地址的请求超时，则应返回 5.04（网关超时）响应。如果到目的地址的请求无法被代

理服务器处理（例如，由于无法识别的关键选项，消息格式错误），则应返回 5.02（错误网关）响应。否则，该代理将响应发回给客户端。

如果响应是由缓存产生的，考虑到在高速缓存中的资源表征花费的时间，生成的（或隐含的）Max-Age 选项不可以扩展服务器预先设置的 Max-Age。例如，采用以下公式，代理可以对每个响应的 Max-Age 选项进行调整：

$$\text{proxy-max-age} = \text{original-max-age} - \text{cache-age}$$

例如，如果请求的是代理资源，该代理资源 20s 之前刷新的并且原始的 Max-Age 为 60s，那么资源的代理最大年龄现在是 40s。考虑到从源服务器的发送过来潜在的网络延迟，代理应稳妥地提供 Max-Age 值。

代理请求中的所有选项应在代理服务器进行处理。无法识别的代理请求中的不安全选项将导致由代理返回的 4.02（Bad Option）响应。COAP-to-COAP 代理应给原始服务器转发所有不能被识别的安全转发选项（Safe-to-Forward）。同样，不能被 COAP-to-COAP 代理服务器识别的响应中的非安全选项将导致 5.02（Bad Gateway）响应。再次说明，无法识别的安全转发选项应被转发。

COAP 和 HTTP 之间的跨协议代理的其他注意事项见第 12 章。

7.8.3 转发代理（Forward-Proxies）

CoAP 协议对请求是发送给原始服务器还是由转发代理进行发送的两种情况是区分对待的。发送到转发代理服务器的 CoAP 请求将作为正常的可确认或不可确认请求发送到转发代理端点，但是，将以不同的方式来定义请求的 URI：代理的请求 URI 是以字符串的形势写入 Proxy-Uri 选项的（见 7.11.3），而发送到原始服务器的请求的 URI 则被分成了 Uri-Host、Uri-Port 和 Uri-Query 选项（见 7.11.2）；或者代理请求的 URI 可以由 Proxy-Scheme 选项和以上提及的三种选项组装而成。

当一个代理请求发送到一个端点、并且该端点不愿意或不能充当请求 URI 的代理时，其应返回一个 5.05（代理不支持）的响应。如果 authority（主机和端口）被识别出是代理端点本身（见 7.11.3），则该请求应被视为一个本地（非代理）的请求。

除非代理的配置是将代理请求转发到另一个代理，否则它应按照以下方式翻译请求：请求 URI 的方案定义了输出协议及其细节（例如，在“CoAP”方案，CoAP 采用 UDP；在“coaps”方案，CoAP 采用 DTLS。）对于 COAP-to-COAP 代理，原始服务器的 IP 地址和端口是由请求 URI 的 authority 组件确定的，请求 URI 将被解码并将被分为 Uri-Host、Uri-Port 和 Uri-Query 选项。这会占用 Proxy-Uri 或 Proxy-Scheme 选项，因此不会被转发到原始服务器。

7.8.4 反向代理（Reverse-Proxies）

反向代理服务器不使用 Proxy-Uri 或 Proxy-Scheme 选项，但需要根据请求的信息和配置中的信息来确定请求的目的地址（下一跳）。例如，一个反向代理可能会提供它通过资源发现获取的已存在的各种资源，这些资源就像它本身的资源的一样。反向代理免费建立了标识这些资源的 URI 命名空间。反向代理也可以建立一个命名空间以给客户端提供监控请求去向的能力，如在所提供的资源的 URI 路径中嵌入主机标识和端口号。

在处理响应时，反向代理要谨慎处理不同来源的 ETag 选项值，在提供给客户端的同一资源上不要弄混淆。在许多情况下，ETag 可以不做改变而转发。如果从一个反向代理服务器提供的资源到众多原始服务器提供的资源的映射是不唯一的，反向代理可能需要生成一个新的 ETag，确保能够正确地保存这个选项的语义。

7.9 方法定义

7.9.1 GET

GET 方法根据当前由请求 URI 所标识的相应资源的信息去获取资源的表征。如果请求中包含 Accept 选项，则表示响应的优先文本格式。如果请求中包含一个 ETag 选项，GET 方法将要求 ETag 进行验证，并且只有当验证失败时 GET 请求的表征才会被发送。如果 GET 成功的话，响应中应包含一个 2.05 (Content) 或 2.03 (Valid) 响应码。

7.9.2 POST

POST 方法要求包含在该请求内的表征能被处理。POST 方法执行的实际功能由源服务端定义并且依赖于目标端的资源。这通常会创建一个新的资源或者更新目的端的资源。

如果在服务端创建了新的资源，那么服务端将返回包含响应码 2.01 (Create) 的响应，以及包含新资源的 URI，URI 由一个或多个 Location-Path 和 (或) Location—Query 选项 (见 7.11.9) 的序列组成。如果 POST 虽然成功却没有在服务器上创建一个新的资源，那么响应中需要包含一个响应码 2.04 (Changed)。如果 POST 成功并且目标资源被删除，则响应中需要包含响应码 2.02 (Deleted)。

POST 即不是安全的也不是幂等的。

7.9.3 PUT

PUT 方法要求对请求 URI 所标识的资源进行更新或根据 URI 附带的表征创建资源。表征的格式由媒体类型定义，内容编码则由 Content—Format 选项中定义。

如果一个资源存在于请求 URI 中，那么 URI 附带的表征应该被作为该资源的修改版本，并且需要返回一个响应码 2.04 (Changed)。如果资源不存在，服务端则会创建一个 URI 指向的新资源，并产生一个响应码 2.01 (Created)。如果资源没有被创建或修改，则会发送一个相应的错误响应码。

更多有关 PUT 的限制条件包含在请求中的 IF-Match (见 7.11.10) 或者 IF-None-Match (见 7.11.10) 选项中。

PUT 不是安全的，但是是幂等的。

7.9.4 DELETE

DELETE 方法请求删除 URI 所标识的资源。当成功删除或者在收到请求之前资源已经不存在时，需要使用响应码 2.02 (Deleted)。

DELETE 不是安全的，但是是幂等的。

7.10 响应代码定义

7.10.1 成功 2.xx

这类状态代码表示客户端的请求被成功接收、理解和接受。

7.10.1.1 2.01 创建

类似 HTTP 201 “创建”，但是在响应 POST 和 PUT 请求中使用。与响应一起返回的负载（如果有的话）是相应动作的结果的表征。

如果响应包括一个或多个 Location-Path 或 Location-Query 选项，这些选项的值表明了资源被创建的位置。否则，资源在请求 URI 所指向的位置中创建。接收到该响应的缓存应将所创建的资源的所有存储的响应标记为非新。

这类响应是不能被缓存的。

7.10.1.2 2.02 删除

类似 HTTP 204 “没有内容”，但是只在导致资源不能再被使用的请求的响应中使用，比如 DELETE 和某些情况下的 POST。与响应一起返回的负载（如果有的话）是相应动作的结果的表征。

这个响应是不可缓存的。然而，缓存应将删除的资源的所有存储的响应标记为非新。

7.10.1.3 2.03 有效

类似 HTTP 304 “未修改”，但只用于表示响应是有效的，该响应是由包含 ETag 选项的 entity-tag 标识的。因此，响应应包含一个 ETag 选项，并且不能包含负载。

当识别并处理 ETag 的响应选项的缓存收到 2.03（有效）的响应，它应通过更新响应中 Max-Age 选项值来更新存储的响应（显示的或者隐式的作为缺省值，见 7.7.2）。对于响应中每种类型的 Safe-to-Forward 选项，在存储的响应中的这种类型的选项集合（可能为空）应被所接收的响应中该类型的选项的集合替换。（根据选项的定义，非安全选项可能触发类似选项的特殊处理。）

7.10.1.4 2.04 改变

类似 HTTP 204 “无内容”，但只用在 POST 和 PUT 请求的响应中。与响应一起返回的负载（如果有的话）是相应动作的结果的表征。

这个响应是不可缓存的。然而，缓存应将所改变的资源的所有存储的响应标记为非新。

7.10.1.5 2.05

内容类似 HTTP 200 “OK”，但只用于响应 GET 请求。与响应一起返回的负载是目标资源的表征。

这个响应是可缓存的：缓存可以使用 max-age 的选项来确定新鲜度（见 7.7.1）并且使用 ETag 选项（如果存在的话）进行验证（见 7.7.2）。

7.10.2 客户端错误 4.xx

这类响应代码是应用于客户端似乎有错误的情况。这些响应码适用于任何请求方法。

在 7.6.2 中详述的条件下，服务器应该包含诊断负载。

这类的响应是可缓存：缓存可以使用 `max-age` 的选项来确定新鲜度（见第 7.6.1 节）。但是它们是无法验证的。

- 1) 4.00 错误请求：类似HTTP 400 “错误请求”。
- 2) 4.01 未授权：客户无权执行所请求的操作。客户端在没有更新其在服务器的认证状态的情况下，不能重复发送该请求。
- 3) 4.02 错误的选项：由于一个或多个无法识别或畸形选项，该请求无法被服务器理解。客户端如果不做任何修改，则不能重复发送请求。
- 4) 4.03 禁止：类似HTTP 403 “禁止”。
- 5) 4.04 未找到：类似HTTP 404 “未找到”。
- 6) 4.05 方法不允许：类似HTTP 405 “方法不允许”，但是没有相对应的“允许”头部字段。
- 7) 4.06 不可接受：类似HTTP 406 “不可接受”，但是没有响应实体。
- 8) 4.12 先决条件失败：类似HTTP 412 “先决条件失败”。
- 9) 4.13 请求实体太大：类似 HTTP 413 “请求实体太大”。响应应该包括Size1选项（见 7.11.11），Size1选项用以指示请求实体的服务器能够并且愿意处理的最大尺寸，除非服务器不能提供这些信息的状态。
- 10) 4.15 不受支持的内容格式：类似HTTP 415 “不受支持的媒体类型”。

7.10.3 服务器错误 5.xx

这类响应代码表示服务器检测到该请求中有错误或无法执行的案件。这些响应代码适用于任何请求方法。

在 7.6.2 中详述的条件下，服务器应该包含诊断负载。

这类的响应是可缓存：缓存可以使用 `max-age` 的选项来确定新鲜度（见 7.7.1）。但是它们是无法验证的。

- 1) 5.00 服务器内部错误：类似 HTTP 500 “服务器内部错误”。
- 2) 5.01 没有实现：类似 HTTP 501 “没有实现”。
- 3) 5.02 错误的网关：类似HTTP 502 “错误的网关”。
- 4) 5.03 服务不可用：类似HTTP 503 “服务不可用”，但使用Max-Age选项替代“Retry-After”头部字段以指示重试之后的秒数。
- 5) 5.04 网关超时：类似HTTP 504 “网关超时”。
- 6) 5.05 代理不支持：服务器不能或不愿给Proxy-Uri选项中指定URI或通过Proxy-Scheme指定的URI充当代理服务器的角色（见7.11.3）。

7.10.4 响应代码列表

当前定义的响应代码见表 4。

表 4 响应代码

类型	代码	名称	含义
成功	2.01	创建	类似 HTTP 201 “创建”，但是在响应 POST 和 PUT 请求中使用。与响应一起返回的负载（如果有的话）是相应动作的结果的表征
	2.02	删除	类似 HTTP 204 “没有内容”，但是只在导致资源不能再被使用的请求的响应中使用，比如 DELETE 和某些情况下的 POST。与响应一起返回的负载（如果有的话）是相应动作的结果的表征
	2.03	有效	类似 HTTP 304 “未修改”，但只用于表示响应是有效的，该响应是由包含 ETag 选项的 entity-tag 标识的。因此，响应必须包含一个 ETag 选项，并且不能包含负载
	2.04	改变	类似 HTTP 204 “无内容”，但只用在 POST 和 PUT 请求的响应中。与响应一起返回的负载（如果有的话）是相应动作的结果的表征
	2.05	200 OK	只用于响应 GET 请求。与响应一起返回的负载是目标资源的表征
客户端错误	4.00	错误请求	类似 HTTP 400 “错误请求”
	4.01	未授权	客户无权执行所请求的操作。客户端在没有更新其在服务器的认证状态的情况下，不能重复发送该请求
	4.02	错误的选项	由于一个或多个无法识别或畸形选项，该请求无法被服务器理解。客户端如果不做任何修改，则不能重复发送请求
	4.03	禁止	类似 HTTP 403 “禁止”
	4.04	未找到	类似 HTTP 404 “未找到”
	4.05	方法不允许	类似 HTTP 405 “方法不允许”，但是没有相对应的“允许”头部字段
	4.06	不可接受	类似 HTTP 406 “不可接受”，但是没有响应实体
	4.12	先决条件失败	类似 HTTP 412 “先决条件失败”
	4.13	请求实体太大	类似 HTTP 413 “请求实体太大”
服务器错误	4.15	不受支持的内容格式	类似 HTTP 415 “不受支持的媒体类型”
	5.00	服务器内部错误	类似 HTTP 500 “服务器内部错误”
	5.01	没有实现	类似 HTTP 501 “没有实现”
	5.02	错误的网关	类似 HTTP 502 “错误的网关”
	5.03	服务不可用	类似 HTTP 503 “服务不可用”，但使用 Max-Age 选项替代“Retry-After”头部字段以指示重试之后的秒数
	5.04	网关超时	类似 HTTP 504 “网关超时”
	5.05	代理不支持	服务器不能或不愿给 Proxy-Uri 选项中指定 URI 或通过 Proxy-Scheme 指定的 URI 充当代理服务器的角色

7.11 可选项定义

7.11.1 可选项概述

表 4 总结了 CoAP 选项，本节中也解释了每个选项的定义。

表 5 中，C、U 和 N 列分别表示 Critical、UnSafe 和 NoCacheKey 三个属性。由于 NoCacheKey 只含有一个 Safe-to-Forward 的选项含义（未被标记为不安全），因此不安全性的选项列里都以横杠填充。（目前的标准里没有定义任何 NoCacheKey 选项，但表格里仍然保留了这个选项，目的是为了便于将来标准的扩展。）

表 5 可选项

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)

7.11.2 Uri-Host, Uri-Port, Uri-Path 和 Uri-Query

Uri-Host、Uri-Port、Uri-Path 和 Uri-Query 选项被用来指定发往 CoAP 源服务器的请求的目标资源。选项将请求 URI 的不同组件进行编码，使得选项值的 percent-encoding 是不可见的，这样任意参与的端点都可以对完整的 URI 进行重构。CoAP URI 的语法定义见章节 8。

将 URI 解析为选项的步骤在 8.5 中定义。通过解析步骤可得到请求中包含的零个或多个 Uri-Host, Uri-Port, Uri-Path 和 Uri-Query 选项，这些选项中包含下列值：

- Uri-Host 选项指的是被请求资源的 Internet 主机；
- Uri-Port 选项指的是被请求资源的传输层端口；
- 每个 Uri-Path 选项指的是访问资源的绝对路径片段；
- 每个 Uri-Query 选项指的是参数化资源的一个参数。

注意：段（见 IETF RFC 3986）并不是请求 URI 的一部分，因此它不会在 CoAP 请求中被发送。

Uri-Host 选项的缺省值是请求信息的目标 IP 地址。同样地，Uri-Port 选项的缺省值是请求信息的目标 UDP 端口。Uri-Host 和 Uri-Port 选项的缺省值对发往大多数的目标服务器来说都是足够的。只有在目的主机上有多个虚拟服务器的情况下才需要在 Uri-Host 和 Uri-Port 选项中给出明确的值。

Uri-Path 和 Uri-Query 选项可以包含任何字符序列，且不采用 percent-encoding。Uri-Path 选项的值一定不能是 “.” 或 “..”（因为在解析选项之前，请求 URI 必须被解析）。

8.5 给出了如何从选项中构造请求 URI 的步骤。注意具体的实现并不需要构造 URI；它可以简单地通过观察独立选项来检索目标资源。

附录 B 中提供了几个例子。

7.11.3 Proxy-Uri and Proxy-Scheme

Proxy-Uri 选项被用来将请求转发到转发代理（见 7.8）。转发代理对请求进行转发或从根据有效的缓存提供服务，并返回一个响应消息。

选项值是一个绝对 URI（见 IETF RFC 3986）。

注意转发代理可以将请求转发至另外的代理，或者直接转发到绝对 URI 指定的服务器。为了避免请求循环，代理应能识别它所有服务器的名称，包括所有的别名、本地变量和 IP 地址。

如果一个通信端点接收到一条带有 Proxy-Uri 选项的请求，但是其不能或不愿意担任该请求的转发代理，那么该端点必返回 5.05（不支持代理）响应。

Proxy-Uri 选项应优先于 Uri-Host, Uri-Port, Uri-Path 或 Uri-Query 选项（这些选项不能与 Proxy-Uri 同时出现在一个请求中）。

作为简化多数代理客户端功能的特殊例子，绝对 URI 可根据 Uri-*选项构造。当 Proxy-Scheme 选项出现时，绝对 URI 构造如下所示：一个 CoAP URI 可根据 Uri-*选项进行构造，如 8.6 的定义。在结果 URI 中，最初的 scheme 将被 Proxy-Scheme 选项的内容所代替，但不包括后续的冒号。注意，该例子仅在 URI 的组件而不是 scheme 的组件可以用 Uri-*选项表示的情况下适用。例如：要表示在 authority 有 userinfo 组件的 URI，只能使用 Proxy-Uri。

7.11.4 Content-Format

Content-Format 选项用来表示信息负载的表示格式。具体的表示格式可用一个数字化的内容格式标识符来进行标识。如果没有该选项时，也不会给定任何缺省值，即任何信息负载的表示格式都是不确定的。

7.11.5 Accept

CoAP Accept 选项可用于表示哪一种 Content-Format 是可被客户端接受。具体的表示格式可用一个数字化的内容格式标识符来进行标识。如果没有给出 Accept 选项，客户端就不表示某个偏好（因此

不会设定缺省值)。客户端希望服务器能够根据 Content-Format 所定义的内容格式进行返回。如果可能,服务器将返回客户端所偏好的内容格式。如果服务器无法返回偏好的内容格式,那么应返回一条 4.06 “Not Acceptable” 响应,除非另一个错误代码拥有更高的优先权。

7.11.6 Max-Age

Max-Age 选项表示一条响应信息被判定为失活之前所能允许的在缓存中的最大时间。

选项值是 0 到 $2^{32}-1$ 之间表示秒的整型数(约 136.1 年)。当响应消息的该选项中为空时,它的缺省值为 60 秒。

Max-Age 的值从响应开始传输时进行计算。对于提供资源的服务器,如果对 Max-Age 的容忍度有着严格的定义时,应该在每一次传输前更新该值。

实体标签可用作本地资源的标识符,以区别随时间变化的相同资源的不同表征。它由提供资源的服务器生成,服务器可采用任意的方法来生成该实体标签,包括版本、校验和、哈希或时间。接收实体标签的端点应将其当做不透明的,并且不能对它的内容和结构做任何假设。(生成实体标签的端点最好尽可能使用最紧凑的表征,特别是客户端和中间节点可能要存储多个 ETag 值。)

ETag 作为响应选项时,返回消息中的 ETag 选项给出了“标签化表征”的当前值(即请求被处理后)。如果没有 Location-* 选项,标签化的表征就是所选目标资源的表征。如果出现了一个或多个 Location-* 选项,这样就显示了 URI 的位置,那么标签化表征就可以通过 GET 请求来进行获取。

一个 ETag 响应选项可包含在任何带有标签化表征的响应消息中(比如它在 4.04 或 4.00 响应中没有意义)。ETag 选项在一条响应消息中一定不能多次出现。

ETag 选项不存在缺省值;如果它没有出现在响应消息中,那么服务器就不会对标签化表示的实体标签进行声明。

7.11.7 ETag 作为响应选项

一个端点之前如果从资源中获得过 1 个或多个表征,并且通过这些表征获取了 ETag 响应选项,可以在一个 GET 请求中指明 1 个或多个缓存响应的 ETag 选项。

如果给出的 ETag 之中有一个是当前表征获的实体标签,即是有效的,那么服务器可以发送一条 2.03 有效响应来代替 2.05 内容响应;这个 2.03 有效响应将在返回的响应选项中返回这个特定的 ETag。

实际上,客户端可以决定任意存储的表征是否是当前的而无需再次传输。

在一个请求中 ETag 选项可能出现零次,一次或多次。

7.11.8 Location-Path and Location-Query

Location-Path 和 Location-Query 选项共同表示一个包含了一个绝对路径或一条查询字符串或包含了两者的 URI。这两个选项的组合将包含在 2.01 (Created) 响应中,用以表示根据一个 POST 请求所创建的资源的位置。该位置可以根据请求 URI 进行解析。

如果一个携带了一个或多个 Location-Path 和/或 Location-Query 选项的响应进入缓存,缓存能够进行识别且 URI 指向了一个或多个已经存在的缓存响应,那些已有的缓存响应应该被标记为非新。

每个 Location-Path 选项指定了一个资源的绝对路径片段,每个 Location-Query 选项指定了一个描述资源的参数。Location-Path 和 Location-Query 选项可包含任意字符序列,且不采用任何

percent-encoding。Location-Path 选项值不能是 “.” 或 “..”。

从选项中构造位置 URI 的步骤与 8.6 中类似，除了前面五步被跳过且结果是一个相对 URI 引用，该引用可以解释为与请求 URI 相关。注意用这种方式构造的相对 URI 引用包括一个绝对路径（例如，如果省略 Location-Path 但有 Location-Query 选项表示 URI 中的路径组件为 “/”）。

可用来计算相对 URI 引用的选项统称为 Location-*选项。除了 Location-Path 和 Location-Query，将来可能定义更多的 Location-*选项，标准里为其预留了选项编号 128、132、137 和 140。如果 Location-Path 和/或 Location-Query 以外的任何预留选项出现在消息中且并不被支持，那么就必须返回一条 4.02（Bad Option）错误。

7.11.9 Conditional Request Options

Conditional Request Options 使客户端能够请求服务器当前仅当选项指定的某个条件被满足时对其请求进行处理。

对于这些选项，如果给出的条件无法满足，那么服务器必须不处理该请求。相反，服务器必须回应 4.12（Precondition Failed）响应码。

如果条件满足，服务器将处理请求方法，就如 Conditional Request Options 选项没有出现过一样。

如果请求里导致了 2.xx 或 4.12 响应码，且没有 Conditional Request Options，那么任何 2.xx 或 4.12 响应码选项都有可能被忽视。

7.11.9.1 If-Match

If-Match 选项可以用于定义发送一个请求的条件，即一个或多个目标资源的表征存在或其具体的 ETag 值。If-Match 对于资源的更新请求如 PUT 请求是有用的，它可以防止多客户端在同一资源上同时执行操作时可能发生的意外重写（即“丢失更新”问题）。

If-Match 选项值可以是一个 ETag 或者空串。If-Match 选项的 ETag 值和表征的 ETag 相匹配。而 If-Match 选项的 ETag 值为空则可以和任意存在的表征相匹配（即，其前置条件是目标资源的任意现有表征都存在）。

If-Match 选项可出现多次。如果任意一个选项匹配了，则条件满足。

如果存在 1 个或多个 If-Match 选项，但没有任何选项可匹配，则条件不满足。

7.11.9.2 If-None-Match

If-None-Match 选项可以用于定义发送一个请求的条件，即目标资源不存在。If-None-Match 对于资源的更新请求如 PUT 请求是有用的，它可以防止多客户端在同一资源上同时执行操作时可能发生的意外重写。If-None-Match 选项不带任何值。

如果目标资源确实存在，则条件不满足。

（在请求中将 If-Match 和 If-None-Match 选项组合起来并不是很有用，因为这样一来条件将永远无法满足。）

7.11.10 Size1 Option

Size1 选项用于提供请求中资源表征的大小信息。选项值是字节的整数倍。它的主要作用是与

block-wise 传输一起使用（参见 IETF I-D.ietf-core-block）。在本标准中，它可用于 4.13 响应中（7.10），以表示服务器能操作的最大请求实体的大小。

8 CoAP URI

8.1 CoAP URI 概述

CoAP 使用 “coap” 和 “coaps” URI 方案来识别 CoAP 资源，并提供定位资源的一种手段。资源是分层组织的，并且由一个 CoAP 源服务器进行管理，该服务器在给定 UDP 端口上侦听 CoAP 请求（“coap”）或 DTLS-secured CoAP 请求（“coaps”）。该 CoAP 服务器通过通用语法的 authority 组件来标识，该组件包括一个主机组件和可选的 UDP 端口号。URI 的剩余部分可用于识别资源，且该资源可以被 CoAP 协议定义的方法操作。“coap” 和 “coaps” URI 方案分别类似 “http” 和 “https” 方案。

本条中的 “coap” 和 “coaps” URI 方案的语法在 Augmented Backus-Naur Form (ABNF)（见 IETF RFC 5234）中有详细描述。“host”、“port”、“path-abempty”、“query”、“segment”、“IP-literal”、“IPv4address” 和 “reg-name” 的定义采用 IETF RFC 3986 中的定义。

具体实现需注意的事项：

不幸的是，随着时间的推移，URI 格式的复杂性显著提升。鼓励实现者仔细研究 IETF RFC 3986。例如，IPv6 地址的 ABNF 比预期可能更为复杂。此外，在从 URI 到其解码组件（或者反过来），实现者应该严格地执行 percent decoding/encoding 的过程。percent decoding 对数据透明性至关重要，但是可能会导致异常的结果，例如，路径组成部分的斜线（“/”）。

8.2 CoAP URI 方案

coap-URI = "coap:" "/" host [":" port] path-abempty ["?" query]

如果主机组件以 IP-literal 或 IPv4 地址的形式呈现，那么 CoAP 服务器可以通过这个 IP 地址进行访问。如果主机是注册名称，则该名称被认为是一种间接的标识符，端点可能会使用名称解析服务（如 DNS）找到该主机的地址。主机不能为空，如果收到的 URI 缺少 authority 或者主机为空，那么它必须被视为无效。端口子组件表示 CoAP 服务器所监听的 UDP 端口。如果它为空或没有给出，那么默认的端口为 5683。

路径用于在主机和端口的范围内识别资源。它由多个路径段按序组成，路径段用斜线符（U+002F SOLIDUS “/”）隔开。

该查询可对访问的资源进一步参数化。它由用 “&” 符号（U0026 AMPERSAND “&”）分隔的参数序列组成。参数采用 “key=value” 对的形式。

“CoAP” URI 方案支持 “/.well-known/” 路径前缀，用于标识主机命名空间中的 “well-known locations”，该路径前缀在 IETF RFC 5785 中有定义。这可用于发现主机的策略或其他信息的发现（“站点范围内的元数据”），如管理的资源（见第 9 章）。

建议应用程序设计人员使用简短的但可描述的 URI。因为 CoAP 所应用的环境通常受限于带宽和

能量，在简短和描述性之间进行权衡，应该更倾向于简略，但是也不能忽略描述性。

8.3 coaps URI 方案

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

所有上述的“coap”方案中列出的要求也是“coaps”方案的要求，不同之处在于，如果端口子组件为空或没有给出，[IANA_TBD_PORT] 的 UDP 端口是不同的，并且 UDP 数据包应是通过使用 11.2 中描述的 DTLS 来保密。

通过“coaps”方案访问的资源与“CoAP”方案访问的资源不共享 authority，即使他们的资源标识符指向了同一 authority（同一台主机监听相同的 UDP 端口）。它们采用的是不同的名字空间，并且被认为是不同的源服务器。

8.4 规范化和比对规则

因为“coap”和“coaps”方案符合 URI 通用语法的，所以这些 URI 是规范化的，并且可以根据 IETF RFC 3986 的第 6 章定义的算法对上述每种方案的默认值进行比对。

如果端口等于所使用方案的默认端口，正常形式是忽略端口子组件。同样，空路径组件等效于“/”绝对路径，所以正常形式是提供“/”路径。该方案和主机是不区分大小写的，通常情况下以小写的形式提供；IP-literals 采用 IETF RFC 5952 推荐的格式；所有其他组件按区分大小写的形式进行比对。字符（除了保留字符集）等同于它们的 percent-encoded 字节（见 IETF RFC 3986，2.1）：正常的形式是不对它们进行编码。

例如，以下三个 URI 是等价的，并且 CoAP 消息中具有相同的选项和选项值：

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

8.5 将 URI 分解成可选项

从字符串|url|中解析请求的选项的步骤如下。解析结果是 0 或者是多个 Uri-Host、Uri-Port、Uri-Path 和 Uri-Query 选项，或者失败。

- 1) 如果|url|字符串不是一个绝对 URI，则使这个算法失败。
- 2) 使用 IETF RFC 3986 定义的参考解析方案解析|url|字符串。在这个阶段，URL 为 ASCII 编码，即使解码组件在步骤 5)、8) 和 9) 之后将被解译为 UTF-8（参见 IETF RFC 3629）。

注：将其解析为与什么相关都没有关系，因为此时我们已经知道它是绝对 URL。

- 3) 如果|url|的<scheme>组件在转换为 ASCII 小写字母时，其值不为“coap”或“coaps”，则使这个算法失败。

- 4) 如果|url|有<fragment>组件，则使这个算法失败。

- 5) 如果|url|的<host>组件没有采用 IP-literal 或 IPv4address 的形式来描述目标 IP 地址，但其包括了一个 Uri-Host 选项，则采用选项的值作为|url|的<host>组件的值，并转换为 ASCII 小写字母，然后把所有 percent-encodings（“%”后跟两个十六进制数字）转换为相应的字符。

注：通常情况下请求的目标 IP 地址从 host 部分解析而来，这可以确保 Uri-Host 选项只用于表单 reg-name 的<host>

组件。

6) 如果|url|有<port>组件, 则将|port|等于该组件的十进制整数的值, 否则, 令|port|等于该方案的默认端口。

7) 如果|port|不等于请求的目的 UDP 端口, 但包含 Uri-Port 选项, 令|port|等于该选项的值。

8) 如果|url|的<path>组件的值是空的或由一个斜杠字符 (U+002F SOLIDUS “/”) 组成, 则移动到下一个步骤。

否则, 对于组件的<path>每个段, 包含 Uri-Path Option 选择, 则让每个段采用选项的值 (不包括分隔斜杠字符), 并将每个 percent-encoding (“%” 后跟两个十六进制数字) 转换为相应的字节。

9) 如果|url|具有的<query>组件, 对于<query>组件中的每个参数, 包括一个 Uri-Query 选项, 则每个参数采用选项的值 (不包括问号和分隔符号字符), 并将每个 percent-encoding 转换为相应的字节。

注: 这些步骤可以完全解析任何 percent-encoding。

8.6 将可选项组合成 URI

从请求的选项中构建一个 URI 的步骤如下。这些步骤或者生成一个 URI, 或者失败。在这些步骤中, 对字符进行 percent-encoding 意味着 “%” 字符后跟两个 16 进制数字替代每个字符 (UTF-8 编码), 数字 A-F 是大写字母 (见 IETF RFC 3986 中 2.1 定义; 为了减少可变性, 用在 CoAP 的 URI 的 percent-encoding 的十六进制表示法应使用大写字母)。“unreserved”和“sub-delims”的定义见 IETF RFC 3986。

1) 如果请求使用 DTLS 进行保密, 令|URL|为字符串 “coaps://”。否则, 令|url|为字符串 “coap://”。

2) 如果请求包含 Uri-Host 选项, 令|host|为该选项的值, 其中的任何非 ASCII 字符由它们相应的 percent-encoding 代替。如果|host|是不是一个有效的 reg-name 或 IP-literal 或 IPv4address, 算法失败。如果请求不包含 Uri-Host 选项, 令|host|等于请求的目标 IP 地址的 IP-literal 或 IPv4address 表示。

3) 将 |host| 添加到 |url|。

4) 如果请求包含 Uri-Port 选项, 令|port|为选项的值。另外, 令|port|为请求目的端的 UDP 端口。

5) 如果|port|不是该方案的默认端口, 则添加一个 U+003A COLON 符号 (:), 后面紧跟十进制表示的|port|, 然后将|port|添加到|url|。

6) 令|resource name|是空字符串。对于请求中每个 Uri-Path 选项, 将所有不属于 “unreserved” 集、“sub-delims” 集、U+003A COLON (:) 还是 U+0040 COMMERCIAL AT (@) 字符的其他全部字符转换成 percent-encoding, 然后在其前面添加一个字符 U+002F SOLIDUS (/), 然后添加到|resource name|。

7) 如果|resource name|是空字符串, 将其设置为单个字符 U+002F SOLIDUS (/)。

8) 对于请求中的每个 Uri-Query 选项, 将所有不属于 “unreserved” 集、“sub-delims” 集 (除了 U+0026 AMPERSAND (&))、U+003A COLON (:)、U+0040 COMMERCIAL AT (@)、U+002F SOLIDUS (/) 还是 U+003F QUESTION MARK (?) 字符的其他全部字符转换成 percent-encoding, 然后在其前面添加一个字符 QUESTION MARK (?) (对于第一个选项) 或 U+0026 AMPERSAND (&) (对于后续选项), 然后添加到|resource name|。

9) 添加 |resource name| 到 |url|。

10) 返回 `|url|`。

注：设计这些步骤用来以规范的生成 URI（见 8.3）。

9 发现

9.1 服务发现

作为 CoAP 服务器发现服务的一部分，客户端应了解服务器所使用的端点。

客户端通过（了解或）学习指向服务器命名空间里的一个资源的 URI 来发现服务器。或者，客户端可以使用组播 COAP（见第 10 章）和采用“所有 CoAP 节点”组播地址来查找 COAP 服务器。

除非“coap”或“coaps”URI 的端口子组件定义了 CoAP 服务器所监听的 UDP 端口，否则，服务器默认采用缺省端口。

提供资源的服务器应支持 CoAP 的缺省端口号 5683，用于资源发现（见 9.2），也应该为访问其他资源提供支持。对于 DTLS-secured COAP 的默认端口号，服务器也可能支持，用于资源的发现和其他资源的访问。此外其他通信端点可以采用其他端口，例如采用动态端口空间。

具体实现需要注意的事项：当 CoAP 服务器由 6LoWPAN 的节点托管时，当其支持 IETF RFC 4944 中定义的 61616-61631 压缩 UDP 端口空间时，报头压缩效率会提高（注意，因为它的 UDP 端口不同于默认端口，所以它与服务器的默认端口是不一样的通信端点）。

9.2 资源发现

由 CoAP 端点提供的资源发现在 M2M 应用中是极为重要的，M2M 的应用中没有人类参与并且采用的静态接口比较脆弱。为了在 CoRE 环境中最大限度地提高互操作性，如 IETF RFC 6690 中描述的 CoAP 端点应支持可发现资源的 CoRE 链接格式（CoRE Link Format），除非完全要进行手动配置。哪些资源（如果有的话）是可被发现的是由服务器决定的。

本条定义了一个新的 web 链接（见 IETF RFC 5988）属性，用于和 IETF RFC 6690 一起使用。内容格式（Content-Format）的代码“ct”属性为此资源返回的内容格式（Content-Format）提供了提示。注意，这只是一个提示，不会替代实际请求资源表征时获得的 CoAP 响应中的 Content-format 选项。该值是十进制的 ASCII 整数，采用 CoAP 标识符代码的格式，取值范围应在 0~65535（16 位无符号整数）的范围内。例如应用 application/xml 将被显示为“ct=41”。如果没有 Content-Format 代码属性存在，则无法对其类型进行假设。Content-Format 代码属性可能包括多个 Content-Format 代码，中间用空格分隔，这表明多种有内容格式（content-formats）可供选择。该属性值的语法如下，其中 cardinal、SP 和 DQUOTE 在 IETF RFC 6690 中被定义。

$$\text{ct-value} = \text{cardinal} \\ / \text{ DQUOTE cardinal } * (1 * \text{SP cardinal }) \text{ DQUOTE}$$

10 组播

10.1 组播的意义

CoAP 支持把请求发送到一个 IP 组播组。这是由一系列增量到单播 CoAP 定义的。

CoAP 终端，为它们想要其他终端能够找到并使用组播服务发现提供服务，他们加入一个或多个合适的全 CoAP 节点组播地址，并监听 CoAP 默认端口。需要注意的是一个终端可能会收到其他组播地址的组播请求，包括全节点的 IPv6 地址（或通过 IPv4 的广播地址）；因此终端必须准备好接收这样的信息，但如果组播服务发现并不是期望的，也可以忽略他们。

10.2 消息层

组播请求的特征是将 CoAP 消息发送到 IP 组播地址而不是 CoAP 终端。这样的组播请求应是不可确认的。

服务器应该知道请求是通过组播到达的，例如，如果可以的话，利用现有的 API（如，IPV6_RECVPKTINFO（见 IETF RFC 3542））。

为了避免错误响应的信息爆炸，当服务器意识到请求通过组播到达，它不能返回一个 RST 答复 NON。如果服务器不知道，它会像平常那样返回一个 RST 答复 NON。因为这样的 Reset 消息会看起来和自来发送方的单播消息的 RST 相同，消息 ID 仍然活跃于可能接收到组播消息的单播端点，发送方应避免使用这样的消息 ID。

在写的时候，组播消息只能在 UDP 中传输，而不能在 DTLS 中。这意味着，这个文件中定义的 COAP 的安全模式并不适用于组播。

10.3 请求/响应层

当一台服务器意识到请求通过组播到达时，服务器可能总是忽略这个请求，尤其是当它没有任何有效的响应时（例如，如果它只有一个空负载或错误响应）。这样的决定可能依赖于应用程序。

如果一台服务器确实决定响应组播请求，它不应该立即响应。相反，它应该挑选一个它打算响应的持续时间段。为了阐述，我们称这个时间的长度为空闲期（Leisure）。空闲期的具体值可以取决于具体的应用，或如下文所述导出。服务器应该在选择的空闲期内挑选一个随机的时间点发回单播响应给组播请求。如果接下来的响应需要在同一个组播地址的成员中发送，新的空闲期可以在前一个完成后最早启动。

为了计算空闲期的值，服务器应该有一个群体规模估值 G ，一个目标数据传输速率 R （这两者都应谨慎选择）和一个估计的响应大小 S ，粗略的空闲期的下限值可以这样计算：

$$lb_Leisure = S \times G / R$$

例如，对于一个在符合 2.4 GHz IEEE 802.15.4（6LoWPAN）标准网络的本地链路范围内的组播请求， G 可以设置为 100（保守估计）， S 设置为 100 字节，目标传输速率设置为 8 kbit/s=1 kB/s。由此产生的空闲期的最小值为 10s。

如果 CoAP 终端没有合适的数据来计算空闲期的值，它可以采用默认值 DEFAULT_LEISURE。

在匹配组播请求的响应时，只有令牌应匹配。响应的源端点不需要与原始请求的目标端点一样。

为了解释 `Location-*` 选项和任何其他表征中嵌入的链接，和响应相关的请求 URI（基本 URI）的 Host 组件的组播地址应该由实际响应的端点的 IP 地址所替换。

10.3.1 缓存

当客户端发送组播请求时，它总是发出一个新的请求到组播组（因为有可能同时有新的组成员加入或者有成员还没有收到之前的请求）。客户端可以用接收响应更新缓存。然后，它使用 `cached-still-fresh` 和 `'new'` 的响应作为请求的结果。

一个回复发送到组播组的 GET 请求的响应可以用来满足相关的单播请求 URI 的后续请求。单播请求的 URI 是通过用响应消息的传输层源地址替换请求 URI 的权限部分获得的。

缓存可以通过向相关单播请求 URI 发送 GET 请求使响应重新生效。

一个到组播组的 GET 请求不得包含 ETag 选项。一种抑制客户端已有响应的机制有待进一步研究。

10.3.2 代理

当转发代理服务器接收到一个表示组播地址的 `Proxy-Uri` 或根据 `Proxy-Scheme` 组成的 URI 的请求时，这个代理服务器获得一组上述的响应，并发送所有响应（包括 `cached-still-fresh` 和 `new`）回原始客户端。

11 安全

11.1 安全概述

本章定义了 CoAP 绑定 DTLS。

在准备阶段，CoAP 设备提供了其需要的安全性信息，包括密钥材料和访问控制列表。11.2.4.3 `RawPublicKey` 模式定义了配置。在准备阶段结束时，设备将会根据如下给定的模式的信息采用四种安全模式中的一种。对于本标准，`NOSEC` 模式和 `RawPublicKey` 模式是强制实现的。

NOSEC: 没有协议级安全（DTLS 被禁用）。在适当的时候，采用替代技术来提供底层的安全。IPsec 的使用超出了本标准的范围。受限节点采用的确定的链路层也提供链路层安全，在有适当的密钥管理的情况下，这种方式是合适的。

PreSharedKey: DTLS 被启用，并且有一个预共享密钥的列表（见 IETF RFC 4279），如 11.2.4.1 中所述，每个密钥包括一个使用它可以进行通信的节点的列表。在极端情况下，可能有一个密钥对应一个与 CoAP 节点通信的节点（1:1 节点/密钥比）。反之，如果两个以上的实体共享一个特定的预共享密钥，这个密钥只能使实体作为该组的成员来认证，而不是一个特定的对等体。

RawPublicKey: DTLS 被启用，该设备具有不包含证书（原始公钥）的非对称密钥对，如 11.2.4.2 所述，证书使用 `out-of-hand` 机制验证。该设备还具有由公用密钥计算得来的标识和一个它能与之通信的节点的标志的列表。

Certificate（证书）: DTLS 被启用，该设备具有使用 X.509 证书（参见 IETF RFC 5280）的非对称密钥对，证书将其绑定到权限名称（`Authority Name`），并且由通用的信任根签名。该装置还具有根信

任锚节点的列表，可用于验证证书。

在“ NOSEC ”模式下，系统只是通过 IP 协议上的标准的 UDP 协议发送数据包，并且 “CoAP” 方案和 CoAP 默认端口会指示该系统。该系统仅仅通过阻止攻击者从 CoAP 节点网络发送或者接受数据包来保证系统安全。

其他三种安全模式下使用 DTLS 实现，由 “ coaps ” 方案和 DTLS-secured CoAP 默认端口表示。其结果是用于认证的安全关联（在安全模式的范围内），并且基于此认证，授权通信伙伴。CoAP 本身不提供协议原语进行认证或授权；当必须进行认证时，它可以是由通信的安全提供（即 IPsec 或 DTLS ）或是由对象安全提供（在负载中）。需要授权的特定操作设备需要这两种形式的安全模式之一。必要地，如果涉及到中间中介，只有当中间中介是信任关系时，通信安全才会生效；CoAP 没有提供一种转发不同的授权级别的方式，客户端有该权限与中间中介到更深一层中间中介或者原始服务器进行通信。因此，可能需要在第一个中间中介执行所有授权。

11.2 DTLS-secured CoAP

11.2.1 DTLS 的使用方式

就像 HTTP 协议基于 TCP 协议使用传输层安全(TLS)保证安全，CoAP 基于 UDP 协议使用 Datagram TLS（DTLS）（参见 IETF RFC 6347）保证安全（如图 10 所示）。本节定义 CoAP 绑定 DTLS，同时，定义了适用于受限环境的最小的托管实现配置。绑定是由单播 CoAP 的一系列增量定义的。实际上，DTLS 是 TLS 与其处理 UDP 传输中的不可靠性的附加功能组成的。

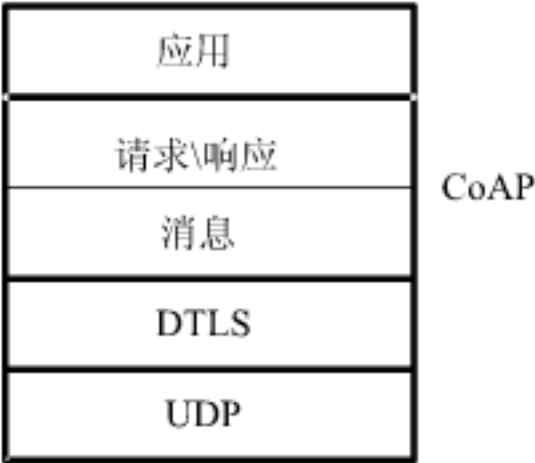


图 10 响应代码的结构

在一些受限的节点（闪存和/或 RAM 受限）和网络（带宽或高可扩展性的要求受限）中，根据特定的密码组合的使用，并不是所有的 DTLS 模式都适用。有些 DTLS 密码组合可能显著的增加实现的复杂性，以及增加建立安全关联时需要的一些初始握手的开销。一旦初始握手完成后，DTLS 的每个数据包增加了约 13 个字节的开销，其中不包括任何初始化向量/随机数（例如，8 字节的 TLS_PSK_WITH_AES_128_CCM_8（参见 IETF RFC 6655）），完整性校验值（例如，8 字节的 TLS_PSK_WITH_AES_128_CCM_8（参见 IETF RFC 6655））和密码组合所需的填充。

对于基于 CoAP 的应用来说，是否选择 DTLS，或者选择哪一种模式的 DTLS 是合适的，需要谨慎地权衡，权衡时要考虑到可用的特定的密码组合，以及会话维护是否使得它和应用流程相匹配，对于受限节点和增加的网络开销，是否有充足资源可用。（对于某些使用 DTLS 的模式，本标准确定了强制实现的密码组合。在这些密码组合确实合适的情况下，这是一个最大化互操作性的实现需求。应用程序的特定的安全策略可以决定实际使用的密码组合（一套）。DTLS 是不适合组密钥的（组播通信），但是，它可能是在将来的组密钥管理协议的一部分。

11.2.2 消息层

端点可作为 CoAP 客户端，也可作为 DTLS 的客户端。它应当向服务器适当的端口发起会话。当 DTLS 握手完成后，客户端可以初始化第一个 COAP 请求。所有 COAP 信息必须被作为 DTLS “应用程序数据” 发送。

下面的规则用于映射一个 ACK 或 RST 到 CON 消息或者一个 RST 到 NON 消息：DTLS 会话应相同，且 epoch 应相同。

当消息在同一个 DTLS 会话和 epoch 中发送时，它们是相同的，有着同样的消息 ID。

注意：当一个可确认消息被重发，每一次尝试重发都会使用一个新的 DTLS 序列号，即使 CoAP 消息 ID 保持不变。因此，接收端仍不得不如 6.6 所述执行重复数据删除。重传不能跨过 epoch 执行。

在 RawPublicKey 和证书模式下的 DTLS 连接设置成使用互相认证，使它们能够在将来任何一个方向的消息交换中能够保持并重用。当设备需要恢复资源的时候，它们可以断开 DTLS 连接，但是通常来说，设备应该尽可能长的保持连接状态。在每个 CoAP 消息交换后断开 DTLS 连接是非常低效的。

11.2.3 请求/响应层

下面的规则用来响应一个请求：DTLS 会话应相同，且 epoch 应相同。

这意味着对于 DTLS 安全请求的响应，应总是 DTLS 安全的，即使用相同的安全会话和 epoch。任何试图向 DTLS 请求作出 NoSec 响应的尝试都被简单地认为请求不匹配（除非它确实匹配一个无关的 NoSec 请求），因此应被拒绝。

11.2.4 端点认证

11.2.4.1 端点命名

设备应支持 SNI，以表示 SNI 主机名字段中的认证名。这是必要的，这样当一台主机作为多个认证方的虚拟服务器接收新的 DTLS 连接时，它知道哪些密钥用于 DTLS 会话。

11.2.4.2 预共享密钥

当产生一个到新节点的连接时，系统将基于它尝试到达的节点，选择适当的密钥，然后用 DTLS 的 PSK（预共享密钥）模式产生一个 DTLS 会话。在这些模式下的实现应强制性实现 IETF RFC 6655 中说明的密码组合 TLS_PSK_WITH_AES_128_CCM_8。

11.2.4.3 原始公钥证书

在这种模式下，设备拥有非对称密钥对，但是没有 X.509 证书（称为原始公钥）；例如，非对称密钥对是由制造商提供，并安装在设备上。一台设备可能用多个原始公钥进行配置。原始公钥的类型和长度取决于所使用的密码组合。在 RawPublicKey 模式下的实现应强制性实现 IETF I-D.mcgreww-tls-aes-ccm-ecc、IETF RFC 5246 和 IETF RFC 4492 中说明的加密套件 TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8。所使用的密钥应是能够 ECDSA 的。曲线 secp256r1 应支持 IETF RFC 4492。该曲线是等效于 NIST P-256 曲线。哈希算法是 SHA-256。其实现应使用支持椭圆曲线的延伸和点支式格式的扩展 IETF RFC 4492；应支持未压缩点格式；IETF RFC 6090 可以作为一种实现方法。与该密码组合的实现的相关的一些指导可以在 W3C XMLSEC 中找到。使用 TLS 和原始公钥的机制在 IETF

I-D.ietf-tls-oob-pubkey 中说明。

实现注意事项：具体而言，这意味着下面列出的扩展与其列出的值将会呈现在 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)

RawPublicKey 模式被设计成在 M2M 部署中易于供应的形式。假定每个设备都有安装了适当的非对称公钥对。一个标识符由 IETF RFC 6920 的第 2 章中所述的公开密钥的端点计算。支持检查 RawPublicKey 标识的全部实现应至少支持 SHA-256-120 模式（SHA-256 截断为 120 位）。实现应该也支持更长的长度标识，并且可以支持更短的长度。请注意，较短的长度提供的抵抗攻击的安全性更弱，不推荐使用。

根据标识符如何提供给验证它们的系统，对 URI、二进制、和/或人工朗读格式 IETF RFC 6920 的支持需要被实现。所有的实现应该支持二进制模式，并且拥有用户界面的实现还应支持人工朗读格式。

在配置中，采集每个节点的标识符，例如通过读取设备外部的条形码或通过获得标识符的预编译列表。这些标识符然后被安装在相应的端点上，例如一个 M2M 数据采集服务器。该标识符被用于两个目的，关联端点与底层设备信息与执行访问控制。在（初始和后续）配置中，标识符的访问控制列表，设备可能通过该列表发起 DTLS 会话，也应该被安装和维护。

11.2.4.4 X.509 证书

证书模式的实现应支持实现 IETF I-D.mcgregor-tls-aes-ccm-ecc、IETF RFC 5246 和 IETF RFC 4492 中指定的密码组合 TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8。也就是说，该证书包括一个

SubjectPublicKeyInfo, 用于指示拥有命名曲线 secp256r1 IETF RFC 5480 的 id-ecPublicKey 的算法; 哈希算法是 SHA-256; 如果包含密钥的用法, 扩展名用来表示数字签名。证书应由使用 secp256r1 的 ECDSA 署名, 并且署名应使用 SHA-256 算法。使用的密钥应是能够 ECDSA 的。曲线 secp256r1 应支持 IETF RFC 4992; 该曲线等效于 NIST P-256 曲线。哈希算法是 SHA-256。实现应使用支持椭圆曲线的延伸和点支式格式的扩展 IETF RFC 4492; 未压缩的点格式应支持; IETF RFC 6090 可以作为一种实现方法。

证书中的认证名称将被构造成设备的一个很长的唯一标示符。认证名称也能基于 FQDN, 它被用作 CoAP URI 的主机部分。然而, 设备的 IP 地址不应该典型地被用作认证名称, 因为它将随着时间而改变。系统中使用的发现进程将建立给定设备的 IP 地址到每个设备认证名称之间的映射。有一些设备可能拥有多个认证名称, 因此可能需要多个证书。

当产生一个新的连接时, 来自远程设备的证书需要被认证。如果 CoAP 节点拥有绝对时间源, 那么该节点应该检查证书的有效日期是否在范围之内。证书应采用功能等同于 IETF RFC 5280 第 6 章规定的算法进行验证以适合于安全性要求。如果证书中包含 SubjectAltName, 那么认证名称应至少匹配任何 COAP URI 中的 URI 类型的字段发现在 SubjectAltName 组的认证名称中的一个。如果在证书中没有的 SubjectAltName, 那么认证名称应与证书中找到的使用 IETF RFC 2818 中定义的匹配规则的 CN, 除非使用通配符证书是不允许的。

CoRE 证书状态检测支持需要进一步的研究。由于 OCSP (见 IETF RFC 2560) 到 CoAP 的映射目前尚未定义, 并且 OCSP 可能也不容易适用于各种环境, 另一种更好的办法可以使用 TLS 证书状态请求扩展 (见 IETF RFC 6066 第 8 章, 也被称为 “OCSP stapling”), 或者如果有的话, 最好是多证书状态扩展。

如果系统除了证书还拥有共享密钥, 那么包含共享密钥的加密组合, 如 TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA (见 IETF RFC 5489), 应该被使用。

12 CoAP 与 HTTP 协议转换代理

12.1 转换代理的意义

CoAP 协议支持 HTTP 功能的有限子集, 因此 HTTP 跨协议代理是比较直接的。CoAP 与 HTTP 之间的代理有很多原因, 比如当设计一个适用于两者的 web 接口。同样的, CoAP 也可以通过代理的方式转换为 XMPP 和 SIP 协议, 这些不在本标准中定义。

通过前向代理访问资源可以有两种方向:

CoAP-HTTP 代理: 让 CoAP 客户端通过代理来访问 HTTP 服务器上的资源。实现方式是, 在消息中包含 Proxy-Uri 和 Proxy-Scheme 或者在向 CoAP-HTTP 代理发送的请求中使用 “http” 或 “https” 的 URI 可选项。

HTTP-CoAP 代理: 让 HTTP 客户端通过代理来访问 CoAP 服务器上的资源。实现方式是, 在向 HTTP-CoAP 代理发送的 HTTP 请求的 Request-Line 中定义 “coap” 或者 “coaps” URI。

只有使用请求/响应模型的 CoAP 协议可以与 HTTP 协议映射。需确认消息和不需确认消息模型对

于代理功能是透明的。接下来的部分描述了前向代理如何处理请求。未定义反向代理因为代理功能对客户端透明，代理扮演源服务器的角色。即便如此，反向代理和前向代理仍需要考虑类似的问题，一般来讲希望反向代理像前向代理一样处理问题。作为开发提示，HTTP 客户端函数库不能提供在 HTTP 请求行中增加 CoAP URI 的方式，这使与 HTTP-CoAP 前向代理互动很难完成；反向代理因此应用更为广范。另一标准中约定了 URI 在这种 HTTP-CoAP 反向代理中的操作方式。

12.2 CoAP-HTTP 代理

12.2.1 定义

如果请求中包含使用 ‘http’ 或者 ‘https’ URI 的 Proxy-Uri 或 Proxy-Scheme 可选项，那么要求接受消息的 CoAP 端点执行由 HTTP 资源指定的方法提供的操作方式，并将结果回送给客户端。

本条定义了对于任何 CoAP 请求，代理都应给予 CoAP 响应。如何满足请求的需要是实现细节，最理想的是代理翻译并向 HTTP 源服务器前传请求。

因为 HTTP 和 CoAP 共享基本请求方法集，在 HTTP 资源上执行 CoAP 请求与在 CoAP 资源上执行并无太大不同。在本条的子条中介绍了在 HTTP 资源上执行单个 CoAP 方法的意义。

如果代理不能或不想处理 HTTP URI 的请求，可以向客户端回复响应代码为 5.05 的响应。如果代理需要第三方处理响应，且无法在合理时间片内得到结果，可以向客户端回复响应代码为 5.04 的响应；得到结果但无法识别，则回复响应代码为 5.02 的响应。

12.2.2 GET

GET 方法请求代理返回该请求所标识的 URI 的 HTTP 资源的表示。

如果成功，则返回响应代码 2.05 的响应。响应的负载应代表目标 HTTP 资源，Content-Format 可选项也要做相应的设置。响应应指定 Max-Age 值，该值不大于资源表示的刷新时间。如果 HTTP 实体有实体标签，代理应在响应中增加 ETag 可选项，执行 ETag 可选项的请求将在接下来描述。

通过如下的可选项，客户端可以顺畅的执行 GET 请求：

- Accept: 请求可包含 Accept 可选项，指出首选响应 content-format。
- ETag: 请求中应包含一个或多个 ETag 可选项，指出客户端存储的响应。这要求代理发送响应代码为 2.03 的响应，只有在请求中有实体标签时其会发送带的响应代码 2.05 响应。注意 CoAP ETag 一直是 HTTP 场景中的强 Etag；CoAP 不等效于 HTTP 弱 Etag，其没有在代理中应用这些好的方式。

12.2.3 PUT

PUT 方法要求代理更新或创建 HTTP 资源，该资源由请求 URI 来封闭表示。

如果请求 URI 创建了新资源，应向客户端返回响应代码 2.01 的响应。如果已存在的资源调整，则应回复成功完成的响应。

12.2.4 DELETE

DELETE 方法要求代理删除 HTTP 资源，该资源由 HTTP 源服务器的请求 URI 辨别。

如果执行成功或者该资源在请求时已经不存在，则应回复响应代码 2.02 的响应。

12.2.5 POST

POST 方法要求代理代表封闭在 HTTP 源服务器上执行的请求。POST 方法执行的实际功能由源服务器辨别并取决于由请求 URI 辨别的资源。

如果 POST 方法执行的动作并不影响资源，则应回复响应代码 2.04 的响应。如果资源在源服务器上创建，则应回复响应代码 2.01 的响应。

12.3 HTTP-CoAP 代理

12.3.1 定义

如果 HTTP 请求的请求行包含 ‘coap’ 或者 ‘coaps’ URI，则要求接收 HTTP 端点执行由 CoAP 资源请求的方法定义的操作，并向客户端返回结果。

本条定义了对于任何 HTTP 请求，代理都应给予 HTTP 响应。除非其他标准规定，所有语句都是推荐行为；一些高度受限的实现可能需要使用捷径。如何满足请求的需要是实现细节，最理想的是代理翻译并向 CoAP 源服务器前传请求。在本节的子节中介绍了在 CoAP 资源上执行单个 CoAP 方法的意义。

如果代理不能或不想处理 CoAP URI 的请求，可以向客户端回复响应代码为 505 的响应。如果代理需要第三方处理响应，且无法在合理时间片内得到结果，可以向客户端回复响应代码为 504 的响应；得到结果但无法识别，则回复响应代码为 502 的响应。

12.3.2 OPTION 和 TRACE

因为 CoAP 协议不支持 OPTION 和 TRACE 方法，所以应向客户端回复 501 的错误响应。

12.3.3 GET

GET 方法要求代理代办 CoAP 资源回复消息。

如果成功，则回复 200 的响应。响应负载应代表目标 CoAP 资源，其消息头中的 Content-Type 和 Content-Encoding 也要做相应设置。响应应指定 Max-Age 值，该值不大于资源表示的刷新时间。如果 CoAP 中有 ETag 可选项，代理应在消息头中包含 ETag。

通过下面的可选项，客户端可以影响 GET 请求的执行：

- Accept: 请求中 HTTP Accept 头字段最优先的 Media-type 映射为 CoAP Accept 可选项。HTTP Accept Media-type 范围，参数和扩展在 CoAP Accept 可选项没有得到支持。如果代理不能发送可处理的响应，则需回复 406 的响应。代理可以根据 HTTP Accept 头字段的 Media-type 重发请求。
- Conditional GET: 包含 “If-Match” 和 “If-None-Match” 请求头字段的 Conditional HTTP GET 请求可以映射为 CoAP 请求。“If-Modified-Since” 和 “If-Unmodified-Since” 请求头字段并不被 CoAP 直接支持，而是由本地代理缓存方式实现的。

12.3.4 HEAD

HEAD 方法和 GET 类似，除了服务器在响应中应不回复消息体。

在 CoAP 协议中并没有与 HTTP HEAD 方法等效的处理方式，HTTP-CoAP 代理为 CoAP 资源响应

HEAD 请求，并只回复 HTTP 头。

开发提示：HTTP-CoAP 代理可以使用 block-wise 传输可选项来使传输数据最少，但这需要做好源服务器不支持 block-wise 传输的准备。

12.3.5 POST

POST 方法要求代理代表 CoAP 源服务器执行的封闭请求。POST 方法执行的实际功能由源服务器辨别并取决于由请求 URI 辨别的资源。

如果 POST 方法执行的动作并不影响资源，则应回复响应代码 200 或 204 的响应。如果资源在源服务器上创建，则应回复响应代码 201 的响应。

如果 CoAP 响应中包含 Location-*可选项，Location 头字段由返回的可选项构成。

12.3.6 PUT

PUT 方法要求代理更新或创建 CoAP 资源，该资源由请求 URI 来封闭表示。

如果请求行创建了新的资源，需向客户端回复 201 的响应。如果已存在的资源修改，则需要回复 200 或者 204 的响应，指出请求已成功。

12.3.7 DELETE

DELETE 方法要求代理删除源服务器上的特定 CoAP 资源。

如果响应包含实体描述状态或者代表动作已执行的 204 的响应，则回复 200 OK 的成功响应。

12.3.8 CONNECT

这种方法目前不被 HTTP-CoAP 代理支持，因为 TLS 到 DTLS 隧道还没被定义。目前，需向客户端回复 501 的响应。
