会话发起协议

Session Initiation Protocol

IETF RFC 3261

摘要：

本文档描述了会话发起协议（SIP），即有一个或多个参与者的用于创建、修改和终止会话的应用层控制（信令）协议。这些会话包括 Internet 电话呼叫、多媒体分发和多媒体会议。

用于创建会话的 SIP 邀请携带允许参与者进行兼容的媒体类型协商的会话描述。SIP 使用称作代理服务器的元素帮助将请求路由到用户的当前位置，认证并授权用户访问服务，实现提供商呼叫路由策略，并为用户提供功能。SIP 同时提供注册功能，允许用户上载其当前的位置信息，以便代理服务器使用。SIP 运行在多个不同的传输协议上。

1 介绍

Internet 上许多应用需要创建和管理会话，会话被认为是相关参与者之间的数据交换。这些应用的实现由于参与者的实践而复杂化：用户可能在端点间移动，他们可通过多个名称寻址，他们可以在多种不同的媒体上通信，甚至是同时的。已开发了多种用于携带各种实时多媒体会话数据（如语音、视频或文本消息）格式的协议。通过使 Internet 端点（称为用户代理）能够发现他人并协商可能共享的会话特征，会话发起协议（SIP）能与其它协议很好地协作。为了定位预期的会话参与者以及其他功能，SIP 允许创建网络主机（称为代理服务器）基础设施，用户代理能向代理服务器发送注册信息、会话邀请以及其它请求。SIP 是用于创建、修改和终止会话的灵活的、通用的工具。它独立于潜在的传输协议，并且不依赖于所建立的会话类型。

2 SIP 功能概要

SIP 是一个应用层控制协议，它能建立、修改和终止多媒体会话（会议），如 Internet 电话呼叫。SIP 也能邀请参与者到已建立的会话中，如多播会议。能从已建立的会议中增加（和删除）媒体。SIP 透明地支持名字映射和重定向服务，重定向服务支持个人移动性——用户可以维持一个外部可视的标识符，而不管其位置。

SIP 支持建立和终止多媒体通信的 5 个方面：

用户位置：确定用于通信的终端系统；

用户可用性：确定被呼叫者参加通信的意愿；

用户能力：确定使用的媒体和媒体参数；

会话建立：“响铃”，在呼叫和被呼叫者建立会话参数；

会话管理：包括传输和终止会话、修改会话参数以及调用服务。

SIP 不是一个垂直集成的通信系统。SIP 更像一个组件，它能与其他 IETF 协议一起使用，

创建一个完整的多媒体架构。典型的，该体系包括一些协议，如用于传输实时数据和提供QoS 反馈的实时传输协议（RTP）(RFC1889[28])、用于控制流媒体传输的实时流协议（RTSP）（RFC 2326[29])、用于控制公共交换电话网络（PSTN）网关的媒体网关控制协议（MEGACO）（RFC 3015[30]）和用于描述多媒体会话的会话描述协议（SDP）(RFC 2327[1])。因此，为向用户提供完整的服务，SIP 必须与其它协议一起使用。但是，SIP 的基本功能和操作不依赖于任何协议。

SIP 不提供服务，但是，SIP 提供能够实现不同服务的原语。例如，SIP 能够定位用户，并向其当前位置传输一个不透明对象。如果一个原语用来传输 SDP 格式的会话描述，例如，端点可以协商会话参数。如果同一原语用来传输呼叫者的图像和会话描述，就很容易实现“呼叫者 ID”服务。因此，典型的，一个原语可用来提供多种不同的服务。

SIP 不提供会议控制服务，如现场管制和投票，SIP 也没有描述如果管理会议。SIP 可以使用其它会议控制协议来初始化一个会话。建立的 SIP 消息和会话能在不同的网络上传输。因此 SIP 不提供任何网络资源预留能力。

提供服务的特性使得安全尤为重要。最后，SIP 提供安全服务套件，包括防止服务否认、认证（用户到用户的认证和代理到用户的认证）、完整性保护、加密和隐私服务。

SIP 能与 IPv4 和 IPv6 兼容。

3 术语

本文档中的关键词 MUST、MUST NOT、REQUIRED、SHALL、SHALL NOT、SHOULD、SHOULD NOT、RECOMMENDED、NOT RECOMMENDED、MAY 和 OPTIONAL 的解释与 BCP14 RFC2119[2]中的相同，指明了兼容的 SIP 实现的需求级别。

4 操作概要

本章用简单实例介绍了 SIP 的基本操作。本章仅作为指南，不包含任何标准化声明。

第一个实例说明了 SIP 的基本功能：端点的定位、想要通信的信号、为建立会话协商会话参数、拆除已建立的会话。

图 1 是在两个用户间（Alice 和 Bob）交换 SIP 消息的典型实例。(每个消息以文本形式的字母 F 和一个数字标记。)本例中，Alice 用 PC 上的 SIP 应用(称为软电话)呼叫 Internet 上 Bob 的 SIP 电话。图中有两个 SIP 代理服务器，它们分别代表 Alice 和 Bob 建立会话。

Alice 使用 Bob 的 SIP 身份，一个称为 SIP URI 的统一资源描述符（URI）“呼叫”Bob。URI 的定义见第 19.1 节。SIP URI 的格式与 email 地址类似，典型的包含一个用户名和一个主机名。本例中为 sip:bob@biloxi.com，其中 biloxi.com 是 Bob 的 SIP 服务提供商的域。Alice 的 SIP URI 为 sip:alice@atlanta.com。Alice 可能是键入 Bob 的 URI、或点击超级链接、或地址簿中的一个条目来发起呼叫。SIP 也提供安全的 URI，称为 SIPS URI，如 sips:bob@biloxi.com。对 SIPS URI 的呼叫保证了在呼叫者和被呼叫者域之间的所有 SIP 消息都是安全、加密传输的（即 TLS)。其中，请求安全地发送给被呼叫者，但是，安全机制依赖于被呼叫者的域策略。

SIP 是基于与 HTTP 类似的请求/响应事务模型。每个事务包含一个请求和至少一个响应，该请求调用服务器上特殊的方法和功能。本例中，事务开始于 Alice 用软电话向 Bob 的 SIP URI 发送一个 INVITE 请求。INVITE 是 SIP 的一个方法，它指出了请求者（Alice）希望服务

器（Bob）采取的行为。INVITE 请求包含许多头字段。头字段称为属性，它提供消息的附加信息。INVITE 中包括呼叫的唯一标识符、目的地址、Alice 的地址、Alice 希望与 Bob 建立会话类型的相关信息。

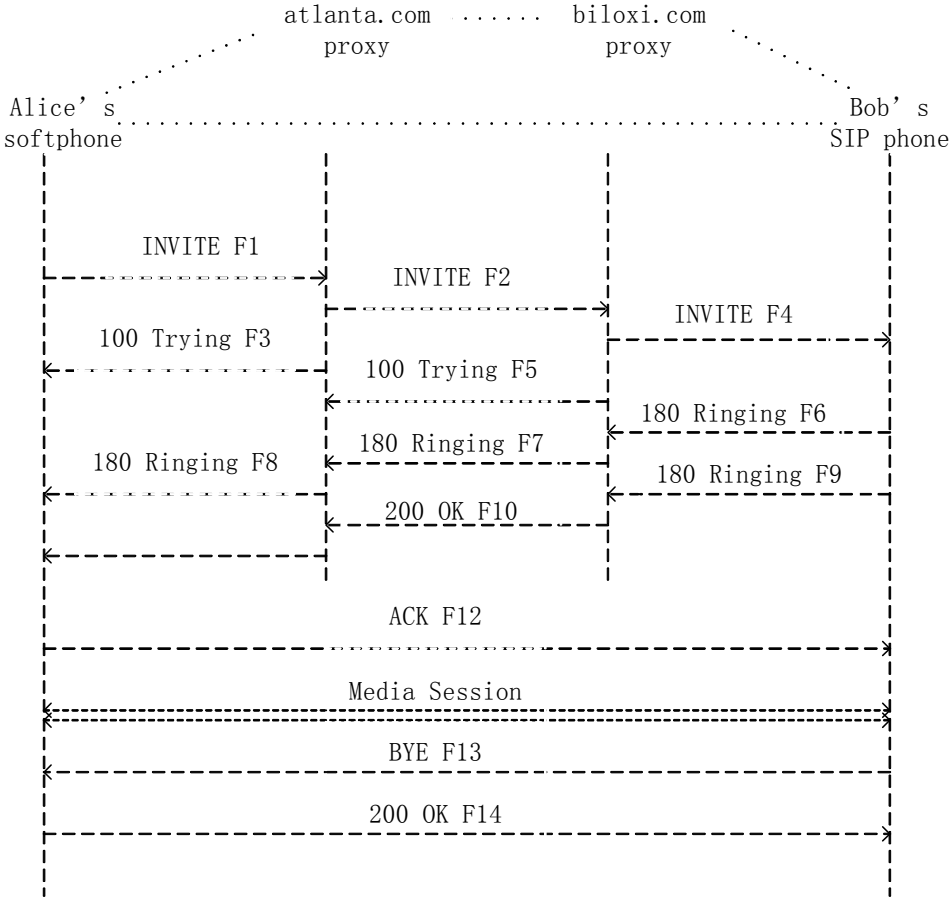

图 1：SIP 会话建立

INVITE（图 1 中的消息 F1）的内容可能如下：

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
(Alice’s SDP not shown)
```

以上文本消息的第一行包含方法名（INVITE）。第二行是头字段。本例是所需的最简设置。头字段简要描述如下：

Via 包含地址信息（pc33.atlanta.com），Alice 希望从该地址接收请求的响应。Via 也包含一个标识事务的分支参数。

To 包含一个显示名称（Bob）和 SIP 或 SIPS URI（sip:bob@biloxi.com），它是请求的目的地址。显示名称在 RFC2822[3]中描述。

From 也包含一个显示名称（Alice）和 SIP 或 SIPS URI（sip:alice@atlanta.com），它是请求的源地址。本头字段有个标签（tag）参数，它包含一个随机串，软电话将该串加到 URI 中。它用于身份鉴别。

Call-ID 包含呼叫的全球唯一标识符，由随机串、软电话主机名或 IP 地址连接而成。To 标签、From 标签和 Call-ID 一起定义 Alice 和 Bob 间对等的 SIP 关系，称为对话。

CSeq 或命令序列包含一个整数和一个方法名。在一个对话内，CSeq 号随着新请求增加，传统上它是一个序列号。

Contact 包含一个 SIP 或 SIPS URI，它表示联系 Alice 的直接路由，它由绝对域名称（FQDN）中的用户名组成。选用 FQDN 的时候，许多终端系统没有注册域名，因此允许 IP 地址。Via 头字段告诉其它元素向哪里发送响应，而 Contact 头字段告诉其它元素以后向哪里发送请求。

Max-Forwards 限制请求在达到其目的地过程中的跳跃点数。它包含一个整数，该整数在每一个跳跃点增加。

Content-Type 包含消息体的描述。

Content-Length 包含计算消息体长度的八位位组（字节）。

SIP 头字段的完整定义见第 20 章。

会话细节，如媒体类型、多媒体数字信号编码解码器或采样率没有使用 SIP 来定义。此外，SIP 消息体包含用其它协议格式编码的会话描述。会话描述协议（SDP）（RFC 2327[1]）就是这种协议格式之一。SIP 消息携带 SDP 的方式类似于 email 消息中携带文档附件，或 HTTP 消息中携带 Web 页面。

由于软电话不知道 Bob 和 biloxi.com 域中 SIP 服务器的位置，软电话向 Alice 域的 SIP 服务器发送 INVITE。可能已经在 Alice 的软电话中设置了 atlanta.com SIP 服务器地址，例如，该地址已被 DHCP 发现。

SIP 服务器是一个代理服务器。代理服务器接收 SIP 请求，并代替请求者转发请求。本例中，代理服务器接收 INVITE 请求，并向 Alice 的软电话返回一个 100（Trying）响应。该 100（Trying）响应表明代理服务器已接收到 INVITE，并向目的地址转发了 INVITE。SIP 的响应由三位数字编码＋描述短语组成。响应包含与 INVITE 中 Via 相同的 To、From、Call-ID、Cseq 和分支参数，从而允许 Alice 的软电话将该响应与已发送的 INVITE 相关联。atlanta.com 代理服务器定位 biloxi.com 代理服务器，其方法可能是通过执行特定类型的 DNS（域名服务）查询来查找 biloxi.com 域的 SIP 服务器。具体描述见[4]。然后，它获得 biloxi.com 代理服务器的 IP 地址，并向该服务器转发 INVITE 请求。在转发请求前，代理服务器增加包含自身地址（INVITE 的 Via 中已包含 Alice 的地址）的一个附加 Via 头字段值。biloxi.com 代理服务器接收到 INVITE 后，向 atlanta.com 代理服务器返回一个 100（Trying）响应，表明已接收到 INVITE，正在处理该请求。代理服务器查询称为位置服务的数据库，该数据库包含 Bob 的当前 IP 地址。（下一节中我们将看到如何操作此数据库）。biloxi.com 代理服务器向 INVITE 中增加一个带自身地址的 Via 头字段值，并将其发送到 Bob 的 SIP 电话。

Bob 的 SIP 电话接收到 INVITE，提醒 Bob 有来自 Alice 的电话，从而让 Bob 作出是否回话的决定，即 Bob 的电话响铃。Bob 的 SIP 电话在 180（Ringing）响应中表明这一点，该 180（Ringing）响应是通过两个代理在不同的方向路由返回。每个代理使用 Via 头字段决定向哪里发送响应，并从顶端移除其地址。尽管为了路由初始的 INVITE 需要查询 DNS 和位置服务，但在向呼叫者返回 180（Ringing）响应时既不需要查询，也不需要在代理中保存状态。这里也有期望的特性，即每个能看到 INVITE 的代理服务器也能看到该 INVITE 的所有响应。

当 Alice 的软电话接收到 180（Ringing）响应，软电话通过回铃音或在 Alice 屏幕上显示消息的方式将信息传递给 Alice。

本例中，Bob 决定回话。当 Bob 拿起手持设备时，他的 SIP 电话发送一个 200（OK）响应，表明电话已接。200（OK）包含一个 Bob 愿意与 Alice 建立的会话类型的 SDP 媒体描述的消息体。因此，这里有两个阶段的 SDP 消息交换：Alice 向 Bob 发送一个消息，Bob 向 Alice 返回一个消息。这两个阶段的交换提供了基本的协商能力，它是基于 SDP 交换的简单呼叫/应答模型。如果 Bob 不愿回话，或者正忙于另一个呼叫，那么发送的将不是 200（OK）而是一个错误响应，也就不会建立媒体会话。第 21 章描述了 SIP 响应代码的完整列表。200（OK）的形式如图 1 消息 F9：

SIP/2.0 200 OK

Via: SIP/2.0/UDP server10.biloxi.com

;branch=z9hG4bKnashds8;received=192.0.2.3

Via: SIP/2.0/UDP bigbox3.site3.atlanta.com

;branch=z9hG4bK77ef4c2312983.1;received=192.0.2.2

Via: SIP/2.0/UDP pc33.atlanta.com

;branch=z9hG4bK776asdhds ;received=192.0.2.1

To: Bob <sip:bob@biloxi.com>;tag=a6c85cf

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710@pc33.atlanta.com

CSeq: 314159 INVITE

Contact: <sip:bob@192.0.2.4>

Content-Type: application/sdp

Content-Length: 131

(Bob’s SDP not shown)

响应的第一行包含响应代码（200）和原因分析（OK）。剩余行包含头字段。Via、To、From、Call-ID 和 Cseq 头字段都是从 INVITE 请求中拷贝的。（这里有三个 Via 头字段值——一个是 Alice 的 SIP 电话增加的，一个是代理服务器增加的，一个是 biloxi.com 代理增加的。）Bob 的 SIP 电话向 To 头字段中增加了一个标签参数。对话的两个终端都会包含该标签，并且在本呼叫的后续请求和响应中也会包含该标签。Contact 头字段中包含 Bob 的 SIP 电话可直接到达的 URI。Content-Type 和 Content-Length 是指包含 Bob SDP 媒体信息的消息体。

除本例中 DNS 和位置服务查询外，代理服务器也可作出灵活的“路由选择决定”，决定向哪里发送请求。例如，如果 Bob 的 SIP 电话返回一个 486（Busy here）响应，biloxi.com

代理服务器可将 INVITE 转发到 Bob 的语音邮箱服务器。代理服务器也可同时向多个位置发送一个 INVITE。这种并行的搜索称为分发（forking）。

本例中，200（OK）通过两个代理服务器路由返回，并且 Alice 的软电话接收 200（OK），它停止回铃音，表明电话已被接通。最后，Alice 的软电话向 Bob 的 SIP 电话发送一个应答消息 ACK，确认接收到最后的响应（200（OK））。本例中，ACK 直接从 Alice 的软电话发送到 Bob 的 SIP 电话，而旁路两个代理。这是因为两个端点已从 INVITE 200（OK）交换的 Contact 头字段获知相互的地址，因而不再需要两个代理执行查询，两个代理退出呼叫流程。这样就结束了用于建立 SIP 会话的 INVITE/200/ACK 三方握手。会话建立的详细细节见第 13 章。

Call-ID: a84b4c76e66710@pc33.atlanta.com

CSeq: 314159 INVITE

Contact: <sip:bob@192.0.2.4>

Content-Type: application/sdp

Content-Length: 131

(Bob’s SDP not shown)

响应的第一行包含响应代码（200）和原因分析（OK）。剩余行包含头字段。Via、To、From、Call-ID 和 Cseq 头字段都是从 INVITE 请求中拷贝的。（这里有三个 Via 头字段值——一个是 Alice 的 SIP 电话增加的，一个是代理服务器增加的，一个是 biloxi.com 代理增加的。）Bob 的 SIP 电话向 To 头字段中增加了一个标签参数。对话的两个终端都会包含该标签，并且在本呼叫的后续请求和响应中也会包含该标签。Contact 头字段中包含 Bob 的 SIP 电话可直接到达的 URI。Content-Type 和 Content-Length 是指包含 Bob SDP 媒体信息的消息体。

除本例中 DNS 和位置服务查询外，代理服务器也可作出灵活的“路由选择决定”，决定向哪里发送请求。例如，如果 Bob 的 SIP 电话返回一个 486（Busy here）响应，biloxi.com 代理服务器可将 INVITE 转发到 Bob 的语音邮箱服务器。代理服务器也可同时向多个位置发送一个 INVITE。这种并行的搜索称为分发（forking）。

本例中，200（OK）通过两个代理服务器路由返回，并且 Alice 的软电话接收 200（OK），它停止回铃音，表明电话已被接通。最后，Alice 的软电话向 Bob 的 SIP 电话发送一个应答消息 ACK，确认接收到最后的响应（200（OK））。本例中，ACK 直接从 Alice 的软电话发送到 Bob 的 SIP 电话，而旁路两个代理。这是因为两个端点已从 INVITE 200（OK）交换的 Contact 头字段获知相互的地址，因而不再需要两个代理执行查询，两个代理退出呼叫流程。这样就结束了用于建立 SIP 会话的 INVITE/200/ACK 三方握手。会话建立的详细细节见第 13 章。

Alice 和 Bob 开始多媒体会话。他们使用在 SDP 交换中协商的格式发送媒体包。通常，端到端媒体包与 SIP 信令消息的路径不同。

在会话过程中，Alice 和 Bob 都可以决定更改媒体会话特征。更改可通过发送报告新媒体描述的 re-INVITE 实现。Re-INVITE 需要提及已建立的对话，以让另一方知道是修改现有的会话，而不是建立新会话。另一方发送 200（OK）接受改变。请求者用 ACK 响应 200（OK）。如果另一方不接受此改变，他就会发送一个错误响应，如 488（这里不接受），此时接收到的也是 ACK。但是，re-INVITE 的失败不会导致现有会话失败——使用先前协商的特征继续会话。关于会话修改的细节见第 14 章。

呼叫结束时，Bob 先断开连接（挂起），产生一个 BYE 消息。BYE 直接传送到 Alice 的软电话，而不通过代理。Alice 用 200（OK）响应确认接收到 BYE，从而终止会话和 BYE 事务。

这里不发送 ACK，ACK 仅作为 INVITE 请求的响应发送。关于 INVITE 特殊处理的原因后面将会讨论，但与 SIP 中的可靠性机制、回答响铃电话的时间长度和分发相关。因此，SIP 中的请求处理通常分为 INVITE 和非 INVITE（指除 INVITE 外的所有其它方法）。会话终止的细节见第 15 章。

第 24.2 节描述了图 1 中的消息。

某些情况下，SIP 信令路径中的代理能看见会话期间在端点间发送的消息是很有用的。例如，如果 biloxi.com 代理服务器希望在初始 INVITE 后继续留在 SIP 信令路径中，它将向 INVITE 添加一个所需路由选择头字段即 Record-Route，其包含解析代理主机名或 IP 地址的 URI。Bob 的 SIP 电话（根据在 200（OK）中传回的 Record-Route 头字段）和 Alice 的软电话将接收到此信息，并在整个对话期间保存该信息。此后，biloxi.com 代理服务器就会接收并转发 ACK、BYE 和 BYE 的 200（OK）。每个代理可独立地选择接收后来的消息，这些消息也会流过选择接收该消息的代理。这种能力通常用于提供呼叫中特征的代理。

注册是 SIP 中另一个通用操作。注册是 biloxi.com 服务器获知 Bob 当前位置的一种方式。在初始化时和在周期性间隔时，Bob 的 SIP 电话会向 biloxi.com 域中的服务器（即 SIP 注册员）发送 REGISTER 消息。REGISTER 消息将 Bob 的 SIP 或 SIPS URI（sip:bob@biloxi.com）与 Bob 当前登录的机器（在 Ccontact 头字段中为 SIP 或 SIPS URI）相关联。注册服务器将此关联（也称为绑定）写入一个数据库（称为位置服务）中，biloxi.com 域中的代理可使用该位置服务。通常，域注册服务器与该域的代理并置（co-located）。这里有一个重要的概念：不同类型的 SIP 服务器的区分是逻辑上的，而不是物理上的。

Bob 可注册多个设备。例如，Bob 家中的 SIP 电话和办公室的 SIP 电话都可以发送注册信息。这些信息都存储在位置服务中，并允许代理执行各种查询来定位 Bob。类似的，在一个设备上可同时注册多个用户。

位置服务是一个抽象的概念。它通常包含这样一些信息：允许代理输入 URI 和接收零或多个 URI（告诉代理向哪里发送请求）。注册是创建这种信息的一种方式，但不是唯一的方式。管理员可选择配置随机映射功能。

最后，值得注意的是在 SIP 中，注册用于转发输入的 SIP 请求，但不用于授权输出的请求。SIP 中的授权和认证是通过挑战/响应机制基于 request-by-request 处理的，或使用低层模式处理，具体讨论见第 26 章。

注册实例的 SIP 消息细节见第 24.1 节。

SIP 中其它操作将在后面的章节中介绍，如使用 OPTIONS 查询 SIP 服务器或客户端的能力，或使用 CANCEL 取消一个未决的请求。

5 协议结构

SIP 是一个分层协议，这就意味着其行为用相对独立的处理阶段集来描述，每个阶段间松耦合。为便于表述，协议的行为用层来描述，允许对功能的描述跨越元素。但是它没有规定实现。当我们说一个元素“包含”一层，我们的意思是说元素遵从该层定义的规则。

并非协议指定的每个元素都包含每一层。而且，SIP 所指的元素都是逻辑元素，而非物理元素。物理实现可作为不同逻辑元素，甚至是基于事务（transaction-by-transaction）的。

SIP 的最低层是语法和编码。其编码指定使用巴科斯范式（BNF）。完整的 BNF 见第 25 章。SIP 消息结构概要见第 7 章。

第二层是传输层。它定义在网络上客户端如何发送请求和接收响应，服务器如何接收请求和发送响应。所有 SIP 元素都包含传输层。传输层的描述见第 18 章。

第三层是事务层。事务是 SIP 的基础组件。事务是客户端事务（使用传输层）向服务器事务发送的请求，以及服务器事务向客户端发回的该请求的响应。事务层处理应用层转播、响应与请求的匹配以及应用层超时。用户代理客户端（UAC）完成任何任务都使用一系列事务。关于事务的讨论见第 17 章。用户代理包含一个事务层，如有状态代理。无状态代理不包含事务层。事务层有一个客户端组件(称为客户端事务)和服务器组件(称为服务器事务)，它们都用有限状态机表示，用来处理特殊请求。

事务层之上的层称为事务用户（TU)。每个 SIP 实体，除无状态代理外，都是事务用户。当事务用户想发送请求时，它就创建一个客户端事务实例，并将请求与目的 IP 地址、端口一起发送。创建客户端事务的 TU 也可以取消事务。客户端取消事务的时候，就要求服务器停止进一步处理，并恢复到初始化事务前的状态，然后返回该事务的一个错误响应。可通过 CANCEL 请求完成取消事务，CANCEL 请求包含自己的事务，同时也提及需要取消的事务（第 9 章)。

SIP 元素即用户代理客户端和服务器、无状态和有状态代理、注册服务器，包含区分这些元素的核心（Core)。除无状态代理外，核心是事务用户。UAC 和 UAS 核心的行为依赖于方法，所有方法有一些通用规则（见第 8 章)。对 UAC 而言，规则支配请求的结构；对 UAS 而言，规则管理请求的处理和响应的生成。由于注册在 SIP 中扮演很重要的角色，处理 REGISTER 的 UAS 有一个特殊的名称“注册员”。第 10 章描述了 REGISTER 方法的 UAC、UAS 核心行为。第 11 章描述了 OPTIONS 方法的 UAC、UAS 核心行为，用来确定 UA 的能力。

其它的请求都在对话中发送。对话是在两个用户代理间持续一定时间的对等 SIP 关系。对话促成两个代理间消息顺序和请求的正确路由。INVITE 方法是本规范中定义的用于建立对话的唯一方法。当 UAC 在对话的连接中发送一个请求时，它遵从第 8 章中讨论的通用 UAC 规则以及对话中请求规则。第 12 章讨论了对话，提出了构建和维护对话的过程，以及对话中请求的结构。

SIP 中最重要的方法是 INVITE 方法，它用于在参与者间建立会话。会话是参与者和参与者间通信的媒体流的集合。第 13 章讨论了如何初始化一个会话，以及产生的一个或多个 SIP 对话。第 14 章讨论了在对话中通过使用 INVITE 请求，修改会话特征。第 15 章讨论如何终止一个会话。

第 8、10、11、12、13、14 和 15 章讨论 UA 核心（第 9 章描述取消，它既适用于 UA 核心，也适用于代理核心)。第 16 章讨论代理元素，说明了用户代理间的消息路由。

6 定义

以下是 SIP 中一些重要的术语：

Address-of-Record：Address-of-Record（AOR）是一个 SIP 或 SIPS URI，它指向带位置服务的一个域，位置服务可以将一个 URI 与另一个 URI（可能找到用户的 URI）映射。典型的，通过注册来填写位置服务。通常认为 AOR 是用户的“公开地址”。

Back-to-Back User Agent：背对背用户代理（B2BUA）是一个逻辑实体，它接收请求，并作为用户代理服务器（UAS）处理该请求。为确定如何应答一个请求，它作为用户代理客户端（UAC）并生成请求。与代理服务器不同，B2BUA 保持对话状态，并参与其建立的对话中发送的所有请求。由于 B2BUA 是 UAC 和 UAS 间的连接，所以不需要明确定义其行为。

Call：呼叫是一个非正式术语，它是指对等实体间的通信，通常是建立多媒体对话。

Call Leg：对话[31]的另一个名称，本规范中没有使用。

Call Stateful：如果一个代理从初始 INVITE 到终止 BYE 请求都保留了对话的状态，那么该代理有呼叫状态。有呼叫状态代理通常是有状态事务。但反之则不一定成立。

Client：代理是发送 SIP 请求和接收 SIP 响应的任何网络元素。客户端可以与也可以不与用户直接交互。用户代理客户端和代理都是客户端。

Conference：包含多个参与者的多媒体会话。

Core：核心指明特殊 SIP 实体类型特有的功能，即有状态或无状态代理、用户代理和注册员特有的。所有核心，除无状态代理的核心外，都是事务用户。

Dialog:对话是两个UA间持续一段时间的对等SIP关系。对话由SIP消息建立,如INVITE请求的 2xx 响应。用呼叫标识符、本地标签和远程标签标识对话。对话即 RFC2543 中的呼叫腿。

Downstream：在一个事务中转发消息的方向，它是指请求从用户代理客户端流向用户代理服务器的方向。

Final Response：终止 SIP 事务的响应，与不终止 SIP 事务的临时响应相反，所有 2xx、3xx、4xx、5xx 和 6xx 响应都是最终响应。

Header：头是 SIP 消息的一个组件，它传递消息的信息。构造头字段作为头字段序列。

Header Field：头字段是 SIP 消息头的一个组件。一个头字段可以是一个或多个头字段行。头字段行由一个头字段名和零或多个头字段值组成。给定的头字段行中的多个头字段值用逗号隔开。某些头字段只能有一个头字段值，因此，它们通常只有一个头字段行。

Header Field Value：头字段值是一个值，头字段由零或多个都字段值组成。

Home Domain：此域为 SIP 用户提供服务。典型的，它通常是注册的记录地址中 URI 中出现的域。

Informational Response：与临时响应相同。

Initiator, Calling Party, Caller：用 INVITE 请求发起会话（和对话）的一方。从发送建立对话的初始 INVITE 请求到该对话终止，呼叫者保持此角色。

Invitation：一个 INVITE 请求。

Invitee, Invited User, Called Party, Callee：接收用于建立新会话的 INVITE 请求的一方。从接收用于建立对话的初始 INVITE 请求到该对话终止，被呼叫者保持此角色。

Location Service：SIP 重定向或代理服务器使用位置服务，以获得被呼叫者可能位置的信息。它包含记录地址与零或多个联系地址的绑定列表。有多种创建和删除这种绑定的方法。本规范定义了更新这种绑定的 REGISTER 方法。

Loop：一个请求到达代理，然后被转发，最后又返回到同一个代理。当请求第二次到达代理时，Request-URI 与第一次相同，其它影响代理操作的头字段也不变，因此代理能对该请求作出与第一次相同的处理决定。循环的请求是错误的，由协议描述检测和处理循环请求的程序。

Loose Routing：如果代理遵循本规范定义的处理路由头字段的程序，那么我们就称其为松散路由选择。该程序将请求的目的地址（在 Request-URI 中）与请求在路由中要访问的代理（在路由头字段中）相分离。符合此机制的代理即是松散路由器。

Message：协议中 SIP 元素间发送的数据。SIP 消息是请求或响应。

Method：方法是请求调用服务器上的基本功能。方法在请求消息中。方法的实例有INVITE 和 BYE 等。

Outbound Proxy：从客户端接收请求的代理，尽管它可能不是 Request-URI 解析的服务器。一般手动为 UA 配置一个出站代理，或通过自动配置协议获知一个代理。

Parallel Search：在并行搜索中，代理在接收到入站请求后，向可能的用户位置发布几个请求。在顺序搜索中，发送一个请求后，代理等待最终响应，而不是立即发送下一个请求；并行搜索在发送请求时，不等待先前请求的结果。

Provisional Response：服务器使用该响应指示进行的响应，该响应不终止 SIP 事务。1xx 响应是临时响应，其它响应都认为是最终响应。

Proxy, Proxy Server：一个媒介（中间）实体，它既作为服务器，也作为客户端（代替其它客户端发送请求）。代理服务器主要执行路由选择，即其职责是确保请求发送到一个离目标用户“更近”的实体。代理也可用于执行策略（例如，确定一个用户可以发起呼叫）。代理负责解释，如果需要的话，在转发请求前重写请求消息的特定部分。

Recursion：当客户端在响应的 Contact 头字段中产生一个或多个 URI 的请求时，客户端递归一个 3xx 响应。

Redirect Server：重定向服务器是用户代理服务器，它为接收到的请求生成 3xx 响应，告诉客户端联系一个可能的 URI 集。

Registrar：注册员是服务器，它接收 REGISTER 请求，并把从请求中接收的信息放入其所在的域的位置服务中。

Regular Transaction：正常事务是方法非 INVITE、ACK 和 CANCEL 的事务。

Request：为调用一个特殊的操作。客户端向服务器发送的 SIP 消息。

Response:为指明客户端向服务器发送的请求的状态,服务器向客户端发送的 SIP 消息。

Ringback：回铃是呼叫者的应用产生的信令音调（signaling tone），表明被呼叫者正在响铃。

Route Set：路由集是按顺序的 SIP 或 SIPS URI 的集合，它是发送特殊请求时必须遍历的代理的列表。路由集是可通过头如 Record-Route 识别的，或是可配置的。

Server：服务器是一个网络元素，它接收请求，并为这些请求发送响应。服务器的实例有代理、用户代理服务器、重定向服务器和注册员。

Sequential Search：在顺序搜索中，代理服务器按顺序尝试每个联系地址，在前一个产生最终响应后再开始尝试下一个。通常用 2xx 或 6xx 最终响应终止一个顺序搜索。

Session：在 SDP 规范中这样描述：“多媒体会话是多媒体发送者、接收者以及发送者和接收者间数据流的集合。多媒体会议是多媒体会话的实例。”（RFC2327[1]）（定义的 SDP 会话包含一个或多个 RTP 会话。）正如所定义的，在同一会话中，被呼叫者可以被不同的呼叫邀请多次。如果使用 SDP，会话由源字段中的 SDP 用户名、会话 ID、网络类型、地址类型和地址元素的串接定义。

SIP Transaction：SIP 事务在客户端和服务器之间发生，包含从客户端向服务器端发送的第一个请求到服务器端向客户端发送的最后一个响应（非 1xx）间的所有消息。如果请求是 INVITE 而最后的响应不是 2xx，那么事务也包含响应的 ACK。INVITE 请求的 2xx 响应的 ACK 是一个独立的事务。

Spiral：螺旋是发送到代理，继续转发，再次回到该代理的 SIP 请求，但是这次将做出

不同于原始请求的处理决定，通常表示请求的 Request-URI 与先前的不同。螺旋不是一个错误情况，与循环不同。典型的原因是呼叫转发。一个用户呼叫 joe@example.com.com，代理将其转发到 Joe 的 PC，PC 反之将其转发给 bob@example.com。该请求再次转发到.com 代理。但是，这不是循环。因为此时该请求的目标用户不同于上次，它被称为螺旋，这是一种有效的情况。

Stateful Proxy：本规范定义的在处理请求过程中，维持客户端和服务器事务状态机的逻辑实体，也称为事务有状态代理。有状态代理行为的详细定义见第 16 节。（事务）有状态代理不同于呼叫有状态代理。

Stateless Proxy：本规范定义的在处理请求过程中，不维持客户端和服务器事务状态机的逻辑实体。无状态代理转发它从下游接收到的每个请求和从上游接收的每个响应。

Strict Routing：如果代理遵循 RFC2543 以及本版本的先前工作中的路由处理规则，则称代理为严格路由。使用严格路由规则时，当出现 Route 头字段时，代理会损坏 Request-URI 的内容。本规范没有使用严格路由行为，而是使用松散路由行为。实现严格路由的代理称为严格路由器。

Target Refresh Request：将在对话中发送的目标更新请求定义为请求，它能修改对话的远程目标。

Transaction User (TU)：传输层上的协议处理层。事务用户包括 UAC 核心、UAS 核心和代理核心。

Upstream：事务中消息的转发方向，它是指响应从用户代理服务器流向用户代理客户端的方向。

URL-encoded：根据 RFC 2396 编码的字符串，见第 2.4[5]节。

User Agent Client (UAC)：用户代理客户端是创建新请求，然后使用客户端事务状态机发送请求的逻辑实体。UAC 角色仅存在事务的持续时间内。换言之，如果软件发起一个请求，它仅在该事务的持续时间内是一个 UAC。如果随后它接收到一个请求，在处理此事务时，它被假想成一个用户代理服务器角色。

UAC Core：事务和传输层上的 UAC 的处理功能集。

User Agent Server (UAS)：用户代理服务器是一个逻辑实体，它产生 SIP 请求的响应。响应接收、拒绝和重定向请求。该角色仅存在于事务期间。换言之，如果软件对请求作出响应，那么它在事务期间就是 UAS。如果随后它产生一个请求，那么在处理事务的期间我们就把它假想成用户代理客户端。

UAS Core：事务和传输层上 UAS 的处理功能集。

User Agent (UA)：既能作为用户代理客户端又能作为用户代理服务器的逻辑实体。

角色 UAC 和 UAS，以及代理和重定向服务器都是定义在事务的基础上的。例如，当用户发起一个呼叫发送初始 INVITE 请求时，它作为 UAC；当它从被呼叫者接收 BYE 请求时，作为 UAS。类似的，同一软件能作为一个请求的代理服务器和下一个请求的重定向服务器。

以上定义的代理、位置和注册服务器都是逻辑实体。实现时可能将它们结合到一个应用中。

7 SIP 消息

SIP 是一个基于文本的协议，它使用 UTF-8 字符集（RFC2279[6]）。

SIP 消息是从客户端到服务器的请求，或从服务器到客户端的响应。

尽管语法在字符集和语法细节上不同，请求（第 7.1 节）和响应（第 7.2 节）都使用基本的 RFC2822[3]格式。(SIP 允许头字段不是有效的 RFC2822 头字段。)两种类型的消息都由一个起始行、一个或多个头字段、一个标识头字段结束的空行和一个可选消息体组成。

```
generic-message = start-line
                  *message-header
                  CRLF
                  [ message-body ]
start-line      = Request-Line / Status-Line
```

起始行、每个消息头行和空行必须以回车换行序列（CRLF）终止。注意：即使没有消息体，也必须有空行。

除了上面所述的字符集不同外，SIP 消息和头字段语法大部分与 HTTP/1.1 相同。这里为了不重复语法和语义，我们用[HX.Y]指代当前 HTTP/1.1 规范（RFC2616[7]）的 X.Y 部分。

但是，SIP 不是 HTTP 的扩展。

7.1 请求

SIP 请求的起始行有 Request-Line，作为与其它消息的区分。Request-Line 包含一个方法名、一个 Request-URI 和由空格（SP）字符分开的协议版本。

Request-Line 以 CRLF 结束。除在终止行 CRLF 序列中外,其它的地方都不允许 CR 或 LF。在元素中不允许任意数量的空格 (LWS)。

```
Request-Line = Method SP Request-URI SP SIP-Version CRLF
```

Method：本规范定义了六种方法：用于注册 Contact 信息的 REGISTER；用于建立会话的 INVITE、ACK 和 CANCEL；用于终止会话的 BYE 和用于查询服务器能力的 OPTIONS。SIP 扩展在标准跟踪 RFC 中可能有附加方法。

Request-URI：Request-URI 是 SIP 或 SIPS URI，描述见第 19.1 节，或是普通 URI（RFC2396[5]）。它指明请求的目的用户或服务。Request-URI 不能包含未保留空间（unescaped spaces）或控制字符，也不能包围在“<>”中。

SIP 元素可能支持“SIP”和“SIPS”以外的 Request-URI 模式,如 RFC2806[8]中的“tel”URI 模式。SIP 元素可使用任何机制将非 SIP URI 转换成 SIP URI、SIPS URI 或其它模式。

SIP-Version：请求和响应消息都包含所使用的 SIP 版本号，遵循[H3.1]（用 SIP 替换 HTTP、SIP/2.0 替换 HTTP/1.1）中关于版本次序、规范要求和版本号更新的描述。为遵从本规范，发送 SIP 消息的应用必须包含 SIP-Version “SIP/2.0”。 SIP-Version 字符串区分大小写，实现时必须发送大写形式的字符串。

与 HTTP/1.1 不同，SIP 将版本号视为字符串。而实际上，这是没有区别的。

7.2 响应

SIP 响应在起始行中有一个 Status-Line，作为与请求的区分。Status-Line 依次由协议版本号、数字 Status-Code 和以及相关的文本分析（textual phrase）组成，它们之间用

字符 SP 隔开。

除在最后的 CRLF 序列中，其他地方不允许有 CR 或 LF。

Status-Line = SIP-Version SP Status-Code SP Reason-Phrase CRLF

Status-Code 是一个三位整数的结果代码，它指明尝试理解和满足请求的结果。Reason-Phrase 是 Status-Code 的简短的文本描述。Status-Code 用于自动控制，Reason-Phrase 便于人理解。客户端不需要检查或显示 Reason-Phrase。

本规范建议了这些原因分析的一些特定用语，实现可以先用其它的文本，例如，在请求的 Accept-Language 头字段中指明的语言。

Status-Code 的第一个数字定义响应类型。后面的两位数字没有类别的意义。因此，状态代码在 100 到 199 之间的响应成为“1xx”响应，状态代码在 200 到 299 之间的响应成为“2xx”响应，依此类推。SIP/2.0 中第一个数字有六种值：

1xx:informational －已经收到请求、继续处理请求。

2xx:success －已经成功收到，理解和介绍行动。

3xx:Redirection － 为完成呼叫请求，还须采取进一步地动作。

4xx:Client Error － 请求有语法错误或服务器不能执行请求。

5xx: Server Error － 服务器出错，不能执行合法请求。

6xx: GLOBAL FAILURE － 任何服务器都不能执行请求。

第 21 节定义了这些类别，并描述了每个代码。

7.3 头字段

在语法和语义方面，SIP 头字段和 HTTP 头字段很相似。特别的，SIP 头字段遵循[H4.2]中消息头语法的定义，以及扩展多行头字段的规则。但是，后者在 HTTP 中有隐含的空白和折叠（whitespace and folding）。本规范遵从 RFC2234[9]，使用明确的空白和折叠作为语法的完整部分。

[H4.2]中指出其值用逗号分隔的、具有相同字段名的多个头字段能组合成一个头字段。这同样适用于 SIP，但对不同的语法有特殊的规则。特别的，SIP 头的语法格式如下：

header = header-name HCOLON header-value *(COMMA header-value)

它允许将名称相同的头字段组合成用逗号分隔的一列。除非头字段值为“*”，Contact 头字段允许用逗号分隔。

7.3.1 头字段格式

头字段遵循 RFC2822[3]的第 2.2 节中的通用头格式。每个头字段依次由：字段名、冒号“:”和字段值组成。

field-name: field-value

第 25 章中指定的消息头的正式语法允许冒号的两边有任意多个空格；但是，实现时应避免字段名和冒号间的空白，在冒号和字段值间使用一个空格（SP）。

Subject: lunch

Subject : lunch

Subject :lunch

Subject: lunch

因此，以上格式都是有效格式，且意义相同，但最后一个是首选格式。

头字段可扩展为多行，方法是在每个附加行前添加一个空格（SP）或横向制表（HT）。一行的结束以及下一行开始的空格都看作一个空格（SP）字符。因此，以下两种方式意义相同：

Subject: I know you’re there, pick up the phone and talk to me!

Subject: I know you’re there,

pick up the phone

and talk to me!

不同字段名的头字段的相对顺序并不重要。但推荐需要代理处理的头字段（例如，Via、Route、Record-Route、Proxy-Require、Max-Forwards 和 Proxy-Authorization）放在消息的头部，以便于快速解析。相同字段名的头字段行的相对顺序非常重要。带相同字段名的多个头字段行仅出现在这样的消息中：头字段的整个字段值的定义是用冒号分隔的一列（即遵循第7.3节定义的语法）。我们可以将多个头字段行组合成一个“field-name: fieldvalue”对，而不改变消息的语义，方法是依次将每个字段值添加到第一个后，它们之间用逗号分隔。WWW-Authenticate、 Authorization、Proxy-Authenticate 和 Proxy-Authorization 不符合本规则，字段名为上述项的多个头字段行可能在一个消息中同时出现，但由于它们不遵循第 7.3 节中的通用格式，因此不能将它们组合成一个头字段行。

实现应能以每行一值或逗号分隔值的格式处理相同名称的多个头字段行。

以下几组头字段行是合法的并且意义相同：

Route: <sip:alice@atlanta.com>

Subject: Lunch

Route: <sip:bob@biloxi.com>

Route: <sip:carol@chicago.com>

Route: <sip:alice@atlanta.com>, <sip:bob@biloxi.com>

Route: <sip:carol@chicago.com>

Subject: Lunch

Subject: Lunch

Route: <sip:alice@atlanta.com>, <sip:bob@biloxi.com>,

<sip:carol@chicago.com>

以下几组是合法的，但意义不同：

Route: <sip:alice@atlanta.com>

Route: <sip:bob@biloxi.com>

Route: <sip:carol@chicago.com>

Route: <sip:bob@biloxi.com>

Route: <sip:alice@atlanta.com>

Route: <sip:carol@chicago.com>

Route: <sip:alice@atlanta.com>,<sip:carol@chicago.com>,
<sip:bob@biloxi.com>

头字段值的格式按头名定义。它通常是一个 TEXT-UTF8 八位位组序列，或一个空格、标记、分隔符和引用串的组合。许多现用的头字段遵循这样的通用格式：字段值，随后是分号分隔的参数名、参数值对：

field-name: field-value *(;parameter-name=parameter-value)

尽管在头字段值后可附加任意多个参数对，但任何给定的参数名不能出现多次。

比较头字段时，字段名是不区分大小写的。除非在特定头字段的定义中有说明外，字段值、参数名和参数值都是不分大小写的。标记通常不区分大小写。除非特殊说明，引用串值要区分大小写。例如：

Contact: <sip:alice@atlanta.com>;expires=3600

与 CONTACT: <sip:alice@atlanta.com>;ExPiReS=3600 意义相同；

且 Content-Disposition: session;handling=optional 与

content-disposition: Session;HANDLING=OPTIONAL 意义相同。

以下两个头字段意义不同：

Warning: 370 devnull "Choose a bigger pipe"

Warning: 370 devnull "CHOOSE A BIGGER PIPE"

7.3.2 头字段分类

一些头字段只在请求中或响应中才有意义。它们分别称为请求头字段和响应头字段。如果消息中的头字段与其种类（例如响应中的请求头字段）不匹配，我们必须将其忽略。第 20 章定义了头字段的类别。

7.3.3 简约格式

SIP 提供了以简化格式表示通用头字段名的机制。当消息太长，无法传输时（例如，在使用 UDP 时消息超过最大传输单元(MTU)），这显得非常有用。第 20 章定义了一些简写格式。简写格式可在任何时候替代头字段名的长格式，而不需要改变消息语义。在一个消息中，头字段名可用短格式和长格式表示。实现必须能同时接受每个头名的长格式和短格式。

7.4 消息体

除非另有说明，请求（包含在规范的扩展中定义的新请求）可能包含消息体。对消息体的解释与请求方法有关。

对于响应消息，请求方法和响应状态代码决定消息体的类型和解释。所有的响应可能都包含消息体。

7.4.1 消息体类型

消息体的 Internet 媒体类型由 Content-Type 头字段指定。如果对体进行了任何编码，如压缩，那么必须在 Content-Encoding 头字段中指出。否则，必须省略 Content-Encoding。如果可行，消息体的字符集作为 Content-Type 头字段值的一部分。

可以在消息体中使用 RFC2046 [10]中定义的“多方”MIME 类型。如果远程实现请求，通过不包含多方的 Accept 头字段请求非多方消息体，那么发送包含多方消息体的请求的实现，必须发送一个会话描述，作为非多方消息体。

SIP 消息可能包含二进制体或体部分。如果发送者没有明确指定字符集参数，定义“文本”类型的媒体子类型有默认的字符集值“UTF-8”。

7.4.2 消息体长度

在 Content-Length 头字段中指定以字节计算的体长度。第 20.14 节详细描述了头字段的必须内容。

HTTP/1.1 的“分块”传输编码不能用于 SIP。（注意，为以分块序列传输消息，“分块”编码修改消息体，每一个都有大小指示器）

7.5 Framing SIP 消息

与 HTTP 不同，SIP 实现可使用 UDP 或其它不可靠数据报协议，每个数据报携带一个请求或响应。见第 18 章对使用不可靠传输的限制。

处理面向流传输的 SIP 消息的实现，必须忽略在起始行[H4.1]中出现的 CRLF。

Content-Length 头字段值用于定位流中每个 SIP 消息的结束。在面向流的传输上发送 SIP 消息时，通常会出现 Content-Length 头字段。

8 通用用户代理行为

用户代理表示终端系统。它包括用户代理客户端 (UAC)和用户代理服务器(UAS)，UAC 生成请求，UAS 响应请求。外部因素（用户点击按钮或 PSTN 线信号）促使 UAC 生成请求，并处理响应。UAS 接收请求，并且根据用户输入、外部因素、程序执行结果或者其它机制，生成响应。

当 UAC 发送请求时，请求通过代理服务器，可以将这些请求转发给 UAS。当 UAS 生成响应，响应将转发给 UAC。

UAC 和 UAS 过程取决于两个因素。第一个是，请求或响应是在对话的内部还是外部；第二个是，请求的方法。在第 12 章中讨论了对话；它们是通过特定 SIP 方法(如 INVITE)建立的，表示了用户代理之间的对等关系。

在本章，我们讨论了处理对话外的请求时，UAC 和 UAS 行为独立于方法的规则。这当然包括自己建立对话的请求。

在第 26 章中介绍了对话外的请求和响应的安全过程。特别介绍了 UAC 和 UAS 互相认证的机制。通过使用 S/MIME 对消息体加密，也可以支持有限的保密性。

8.1 UAC 行为

本章主要介绍了对话外的UAC行为。

8.1.1 生成请求

UAC制定的有效SIP请求，必须，至少包括以下头字段：To、From、Cseq、Call-ID、Max-Forwards和Via。在所有的SIP请求中，这些头字段都是必需的。这六个头字段是SIP消息基本的构件块，它们共同提供大部分关键性消息路由服务，包括消息的寻址、响应的路由、限制消息的传播、消息的排序和事务的唯一标识符。UAC制定的有效SIP请求除了包含这些头字段外，还有必需的请求行。这个请求行包含了方法、Request-URI和SIP版本。

在对话外发送的请求的实例包括INVITE建立会话（第13章）和OPTIONS查询能力(第11章)。

Request-URI 消息初始Request-URI应该设置成To字段的URI值。但应注意,REGISTER方法例外；第10章中给出了设置REGISTER的Request-URI的行为。保密性原因或者便于将这些字段设置成相同的值（特别是在传输过程中，原始UA期望改变Request-URI），可能不符合需要。

在一些特殊的情况中，预有的路由集合的存在可能影响消息的Request-URI。预有的路由集合可能是用来识别一系列服务器的URI有序集合，UAC将对话外的出站请求发送到其中一个服务器上。通常，预有的路由集合由用户或者服务提供商在UA上手动配置，或者通过一些其它的非SIP机制配置。当提供商希望用带外代理配置UA时，推荐通过为UA提供一个带外代理预有的带单一URI的路由集合来配置UA。

当预有的路由集合存在，必须遵循第12.2.1节中详细介绍的Request-URI和路由器头字段的填充过程（即使没有对话），使用想要的Request-URI作为远程目标URI。

To To头字段首先指明了想要的请求的“逻辑”接收者或者用户的记录地址或者作为请求目标的资源。这不一定是请求的最终接收者。To字段可能包含SIP或者SIPS URI，在适当的时候，它也可以使用其它URI模式（例如，tel URL (RFC 2806 [8])）。所有SIP执行必须支持SIPS URI模式。任何支持TLS的执行必须支持SIPS URI模式。To头字段考虑到了显示名称。

UAC可以知道怎样以多种方法为特定的请求填充To头字段。通常，用户建议To头字段通过用户界面填充——可能是手动输入URI或者从地址本中选择。通常，用户不必键入完整的URI，而是键入数字或字符串（如，“bob”）。这由UA来选择怎样解释此输入。使用字符串来形成用户SIPS URI的一部分，意味着UA希望此名字可以在SIPS URI注册的右边（RHS）进行域名解析(如，sip:bob@example.com)。使用字符串来形成用户SIPS URI的一部分，意味着UA希望可以安全地通信，同时，此名字可以在[@]的右边进行域名解析。右边通常是请求者的归属域，这考虑了处理出站请求的归属域。这种像“快速拨号”的特征很有用，它需要归属域的用户部分的解释。

当UA不希望指明应该解释用户输入的电话号码的域时，可以使用tel URL。相反，UA希望指明请求通过的每个域。例如，在飞机场的用户可能已经登机，通过飞机场的带外代理发送请求。如果键入“411”（美国本地目录帮助的电话号码），这需要通过飞机场——而不是用户归属域的带外代理解释并处理它。在这种情况中，tel:411将是正确的选择。

对话外的请求不能包含To标签，请求中的To字段标签表示对等对话。因为没有建立对话，所以不存在标签。

To 头字段的更多信息见第 20.39 节。下面是 To 头字段的实例：

To: Carol <sip:carol@chicago.com>

From From 头字段表示请求发起者的逻辑身份，有可能是用户的记录地址。和 To 字段一样，它包含了 URI 和显示名称，显示名称是可选的。SIP 元素用它来确定应用于请求的处理规则（如，自动呼叫拒绝）。同样地，因为没有逻辑名字，不包含 IP 地址或者 UA 运行主机的正式域名的 From URI 很重要。

From 头字段考虑了显示名称。如果客户端的身份是隐藏时，UAC 应该使用显示语法正确的名字“匿名的(anonymous)”,而不是无意义的 URI(像 sip:thisis@anonymous.invalid)。

通常，特殊 UA 产生的、在请求中填充到 From 头字段的值是用户或者用户本地域的管理员预先提供的。如果多个用户使用特殊 UA，那么，它具有可交换的描述，包括对应于描述用户身份的 URI。为了确定它们是 From 头字段所声称的人，请求的接收者可以认证请求的发起者（关于认证的详情见第 22 章）。

From 字段必须包含 UAC 选择的新标签参数。选择标签的详情见第 19.3 节。

关于 From 头字段的更多信息，见第 20.20 节。下面是 From 头字段的实例：

From: "Bob" <sips:bob@biloxi.com> ;tag=a48s

From: sip:+12125551212@phone2net.com;tag=887s

From: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8

Call-ID Call-ID 头字段作为集合一系列消息的唯一标识符。在对话中，每个 UA 发送的所有请求和响应中，Call-ID 必须是一样的。UA 的每个注册中，它应该是一样的。

在 UAC 创建的对话外的新请求中，如果不是特定方法行为覆盖的，UAC 选择的 Call-ID 头字段必须是在时间和空间上全球唯一的标识符。所有的 SIP UA 必须有一种方法来保证其他 UA 不会产生它们产生的 Call-ID 头字段。注意，当在特定的失效响应后，重发请求以修正请求时（如，认证挑战），重的请求将不作为新的请求，因此不需要新的 Call-ID 头字段；见第 8.1.3 节。

推荐在生成 Call-ID 时，使用密码学上的随机标识符（RFC 1750 [11]）。执行时可以使用这种格式“localid@host”。Call-ID 是区分大小写的，并且逐字节比较的。

使用密码学上的随机标识符提供了会话截获保护，并减少了 Call-ID 冲突的可能性。

对于选择请求的 Call-ID 头字段的值，不需要规章界面或用户界面。

关于 Call-ID 头字段的更多信息，见第 20.8 节。

下面是 Call-ID 头字段的实例：

Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@foo.bar.com

CSeq CSeq 头字段是用作识别和指示事务的。它由序列号和方法组成。此方法必须和请求相匹配。对于对话外的非 REGISTER 请求，此序列号是任意的。此序列号的值必须是值小于 $2^{\wedge 31}$ 的 32 位的无符号整数。只要遵循上述原则，客户端就可以随意地使用一种机制来选择 CSeq 头字段值。

第 12.2.1 节讨论了对话内的请求中 CSeq 的结构。

实例：

CSeq: 4711 INVITE

Max-Forwards Max-Forwards 头字段是用作限制请求传输到其目的地跳跃的点数。它

是一个整数，在每个跳跃点上减一。如果请求在到达其目的地之前，Max- Forwards 值到 0，将返回 483(太多跳跃点)错误响应，拒绝其请求。

UAC 必须在每个请求中插入 Max-Forwards 头字段，并赋初始值为 70。此数字足够长，可保证在没有环路时，请求不会在 SIP 网络中丢失；当存在环路时，此数字不会消耗代理太多的资源。要谨慎地使用更小的值，只有在 UA 知道网络拓扑时，才可以使用更小的值。

Via Via 头字段表示事务中使用的传输，并标识了响应发送的位置。仅在选择了要到达的下一个跳跃点后(这可能包括[4]中过程的使用)，才在传输中加上 Via 头字段的值。

当 UAC 创建请求时，它必须在请求中插入 Via。在头字段中的协议名和协议版本必须分别是 SIP 和 2.0。Via 头字段值必须包含分支参数（branch parameter）。此参数用来识别请求创建的事务。此参数同时用于客户端和服务器。

对于 UA 发送的所有请求，其分支参数的值必须在时间和空间上是唯一的。此规则的异常是 CANCEL 和非 2xx 响应的 ACK。

如下面讨论的，CANCEL 请求有与它取消的请求相同的分支值。如第 17.1.1 节讨论的，非 2xx 响应的 ACK 也有与它所应答的 INVITE 响应相同的分支 ID。

分支 ID 参数唯一的特性，有助于其作为事务 ID 使用，这不是 RFC2543 的一部分。

和本规范一致的元素插入的分支 ID 必须是以“z9hG4bk”字符开头。这七个字符用作 magic cookie（7 认为是足以确保旧 RFC 2543 执行不会选择这个值），以便于接收请求的服务器可以确定以本规范描述的格式（即，全球唯一）构造分支 ID。在本规范内，分支标签的精确格式是执行定义的(implementation-defined)。

当请求是通过传输层处理的（第 18 章），那么将发送 Via 头的 maddr、ttl 和 sent-by 组件。

在第 16.6 节的第 8 条和第 16.7 节的第 3 条介绍了代理的 Via 处理。

Contact Contact 头字段提供了 SIP 或者 SIPS URI，可用于为随后请求联系 UA 的具体实例。必须正确地表示 Contact 头字段，并包含任何请求中的 SIP 或者 SIPS URI，这将导致建立对话。在本规范中定义的方法，仅仅包含 INVITE 请求。对于这些请求，Contact 的范围是全局的。即 Contact 头字段值包括 UA 要接收请求的 URI，即使是在任何对话外的随后请求中使用，此 URI 都必须是有效的。

如果 Request-URI 或 top Route 头字段值包含 SIPS URI，Contact 头字段也必须包含 SIPS URI。

Contact 头字段的更多信息，见第 20.10 节。

Supported and Require 如果 UAC 支持 SIP 扩展，服务器可将此扩展用于响应，UAC 应该在请求中引用 Supported 头字段，列出这些扩展的可选标签（第 19.2 节）。

列出的可选标签必须引用标准协议栈 RFC 中定义的扩展。这就防止了服务器为了接收服务而坚持客户端执行非标准的、供应商定义的特性。因为，供应商也经常参考实验和情报的 RFC 中定义的扩展，定义自己的扩展，所以，它们显然不能用在请求中的 Supported 头字段。

如果 UAC 坚持要 UAS 理解，UAC 为处理请求而用于请求的扩展，它必须在请求中插入 Require 头字段，列出这些扩展的可选标签。如果 UAC 希望将扩展用于请求中，并坚持它所要经历的任何代理都可以理解这些扩展，它必须在请求中插入 Proxy-Require 头字段，列出这些扩展的可选标签。

和 Supported 头字段一起，Require 和 Proxy-Require 中头字段可选的标签必须仅仅引用标准协议栈 RFC 中定义的扩展。

Additional Message Components 在创建了新请求，并合理地构造上面所介绍的头字段后，可以添加任何其他的可选头字段，作为具体方法的头字段。

SIP 请求可能包含 MIME 编码的消息体。不管请求包含的消息体是什么类型，必须阐明特定的头字段来说明主题内容的特征。关于这些头字段的更多信息，见第 20.11 节到第 20.15 节。

8.1.2 发送请求

请求的目的地是可计算的。除非有本地的策略指明，目的地必须通过[4]中介绍的 DNS 过程才可以确定。如果路由集合中的第一个元素指示了严格的路由器（导致组成第 12.2.1 节中介绍的请求），那么必须将 DNS 过程用于请求的 Request-URI。否则，必须将 DNS 过程用于请求中的第一个 Route 头字段（如果存在），如果没有 Route 头字段，将 DNS 过程用于请求的 Request-URI。这些过程产生有序的地址集合、端口和传输。如果 Request-URI 指定 SIP 源，那么与过程[4]使用哪个 URI 无关；如果输入的 URI 是 SIPS URI，那么 UAC 必须遵循过程[4]。

本地策略可以指定可选的目的地。如果 Request-URI 包含 SIPS URI，那么，任何可选的目的地必须和 TLS 联系。除此以外，如果请求中没有包含 Route 头字段，那么，就不约束可选的目的地。这提供了简单的可选机制——为预有的路由集合指定带外代理。然而，不推荐配置带外代理，反而应该使用单一 URI 的预有的路由集合。如果请求包括路由头字段，那么请求应该发送到来自其最上面值的位置；但是也可以遵循本文档指定的路由和 Request-URI 策略发送至任何 UA 信任的服务器（与 RFC2543 中相反）。特别的是，带外代理配置的 UAC 应该试图发送请求给第一个 Route 头字段值指明的位置，而不是采用策略发送所有消息给带外代理。

这确保了没有添加 Record-Route 头字段值的带外代理将不参与随后请求的路径。它也允许不能解析第一个 Route URI 的终端，将此任务委派给带外代理。

对于有状态元素，UAC 应该遵循[4]中定义的过程——尝试每个地址，直到联系到服务器。每次尝试构成新的事务，因此，每次用新的分支参数携带不同的最上面的 Via 头字段值。此外，在 Via 头字段的传输值设置成每个目标服务器确定的传输。

8.1.3 处理响应

响应最初是在传输层处理，然后传输到处理层。处理层完成其处理，然后将响应传输到 TU。TU 的大多数响应处理是指定方法的。但是，还有一些通用的行为与方法无关。

Transaction Layer Errors 在很多情况中，处理层返回的响应不是 SIP 消息，而是处理层错误。当从处理层接收到超时错误时，必须视为接收到了 408（请求超时）状态代码。如果传输层报告了重大的传输错误（通常，由于 UDP 的重大 ICMP 错误或者 TCP 连接错误），这种情况必须视为 503（服务不可用）状态代码。

Unrecognized Responses UAC 必须将其不识别的最终响应视为对等于 x00 类的响应代码，并且，UAC 必须能够处理所有的 x00 响应代码。

例如，UAC 接收到了不识别的响应代码 431，它可以安全地设想此请求有问题，并将响应处理为接收到了 400（错误请求）响应代码。UAC 必须将不同于 100 的、不识别的临时响应视为 183（会话进行）。UAC 必须能够处理 100 和 183 响应。

Vias 如果在响应中存在不止一个 Via 头字段，那么 UAC 应该丢弃这些信息。

在请求发起者之前的其他 Via 头字段值的状态，暗示消息是错误指向的或者是不可靠的。

Processing 3xx Responses 在接收到重定向响应后（如，301 响应状态代码），客户端应该使用 Contact 头字段的 URI 来说明一个或者多个基于重定向请求的新请求。此过程类似于第 16.5 节和第 16.6 节中详细介绍的代理转发 3xx 响应。客户端从正确包含 URI 和原始请求 Request-URI 的初始目标集合开始。如果客户端希望阐明此请求的基于 3xx 类响应的新请求，它将此 URI 放在目标集合中。服从本规范的约束，客户端可以选择将哪个 Contact URI 放在目标集合中。随着代理递归，处理 3xx 类响应的客户端不能够再在目标集合中添加任何给定的 URI。如果原始请求有 Request-URI 中的 SIPS URI，那么客户端可以选择换成非 SIPS URI，但是应该通知用户不安全 URI 的重定向。

任何新请求都可能接收到包含其自己原始 URI 的 3xx 响应作为联系。两个位置可以互相配置成重定向。将任何给定的 URI 仅在目标集合放置一次可以防止无穷的重定向环路。

随着目标集合的增长，客户端可以以任何顺序生成 URI 新请求。通用的机制是按照 Contact 头字段值的'q'参数值排序。URI 请求可以是连续的，也可以是并行的。其中一种方法是，连续地处理递减的 q 值排序，并行地处理 URI 的每个 q 值。另一种方法是，仅仅连续地处理递减的 q 值排序，任意选择相等 q 值之间的联系。

如果联系列表中的地址造成错误，与在下一章所定义的一样，元素移到列表中的下一地址，直到列表用完为止。如果列表已经用完，那么此请求错误。

错误应该通过错误响应代码（比 399 大的代码）来检测；如果是网络错误，那么客户端事务将向事务用户报告传输层错误。注意，有些响应代码（详情见第 8.1.3.5 节）说明了重试的请求，重试的请求不应该认为是错误。

当接收到具体联系地址失败时，客户端应该尝试下一个连接地址。这包括创建新的客户端事务来传递新的请求。

为了在 3xx 响应中创建基于联系地址的请求,UAC 必须将目标集合中的整个 URI 复制到 Request-URI，除了 method-param 和 header URI 参数(参数定义见第 19.1.1 节)。它使用头参数来创建新请求的头字段值，重写相关重定向请求的头字段值是为了和第 19.1.5 节的原则一致。

注意，在一些实例中，在联系地址中通信的头字段可以改为添加到原始重定向请求的现有请求的头字段中。作为通用规则，如果头字段接受了逗号分隔的列标值，那么新的头字段可以添加到原始重定向请求的现有值中。如果头字段不能接受多个值，那么，可以用联系地址中通信的头字段值重写原始重定向请求。例如，如果与下列值一起返回联系地址：

sip:user@host?Subject=foo&Call-Info=<http://www.foo.com>

那么，将重写原始重定向请求的 Subject 头字段，HTTP URI 不过是附加在现有的 Call-Info 头字段值后面。

推荐 UAC 重用原始重定向请求中相同的 To、From 和 Call-ID，但是 UAC 可以选择更新新请求的 Call-ID 头字段值。

最后，一旦构造了新请求，那么，使用新的客户端事务来发送新请求，因此，必须在最上面 Via 字段中有新的分支 ID（见第 8.1.1 节）。

无论从其他哪个方面来看，在接收重定向响应基础上发送的请求应该重用原始请求的头字段和消息体。

在一些实例中，与接收到的状态代码和逾时间隔的状态有关，可以在 UAC 中临时或者永久地缓存 Contact 头字段值（见第 21.3.2 和 21.3.3 节）。

Processing 4xx Responses 具体的 4xx 响应代码与方法无关，需要具体的 UA 处理。

如果接收到 401（未经许可）或 407（需要代理认证）响应，那么 UAC 应该遵循第 22.2 节和第 22.3 节的认证过程，使用凭证来重试请求。

如果接收到 413（请求实体太大）响应（第 21.4.11 节），请求包含的消息体比 UAS 愿意接受的消息体长，如果可能的话，UAC 应该忽略消息体或者使用较短的消息体重试请求。

如果接收到 415(不支持的媒体类型)响应，UAS 不支持请求中包含的媒体类型。UAC 应该重试请求，此时仅使用响应中 Accept 头字段列出的类型，响应中 Accept-Encoding 头字段列出的编码和响应中 Accept-Language 头字段列出的语言。

如果接收到 416(不支持的 URI 模式)响应，服务器不支持 Request-URI 使用的 URI 模式。客户端应该重试请求，此时使用 SIPS URI。

如果接收到 420（错误的扩展）响应，此请求包含 Require 或者 Proxy- Require 头字段，列出了代理或者 UAS 不支持的可选标签特征。UAC 应该重试请求，此时，忽略响应中扩展列出的 Unsupported 头字段。

在上面所有的情况中，通过适当的修改创建新请求，重试请求。新请求组成新事务，应该有与原来请求相同的 Call-ID、To 和 From 值，但是 Cseq 应该包含比原来高的新序列号。

其它的 4xx 响应，包括仍然在定义的，重试可以与也可以不与方法和使用案例有关。

8.2 UAS 行为

当 UAS 处理对话外的请求时，与方法无关，要遵循一套处理规则。第 12 章给出了 UAS 怎样知道请求是在对话之内还是对话之外。

注意，请求处理是基本的。如果接受请求，所有与其相关的状态改变必须执行。如果拒绝请求，所有的状态改变都不能执行。

UAS 应该遵循本节中的步骤处理请求（即是以认证开头，随后检查方法、头字段以及本节的其它内容等等）。

8.2.1 方法检查

一旦请求通过了认证（或者跳过认证），UAS 必须检查请求的方法。如果 UAS 识别但是不支持请求的方法，它必须生成 405（方法不允许）响应。在第 8.2.6 节介绍了生成响应的过程。UAS 必须为 405（方法不允许）响应添加 Allow 头字段。Allow 头字段必须列出 UAS 生成消息支持的方法集合。

如果方法是服务器支持的一种，那么，将继续处理。

8.2.2 头检查

如果 UAS 不能理解请求中的头字段（即是没有在本规范或者其它支持的扩展中定义此头字段），那么服务器必须忽略此头字段，并继续处理消息。UAS 应该忽略任何在处理请求中非必需的、非格式化的头字段。

To and Request-URI To 头字段标识了原始请求中 From 字段中标识的用户。由于呼叫转移或者其它代理操作，原始接收者可以是也可以不是 UAS 处理此请求。当 To 头字段不是

UAS 身份时，UAS 可以应用任何策略来确定是否接受请求。然而，即使在 To 头字段中有它们不识别的 URI 模式（如，tel:URI），或者 To 头字段没有指明 UAS 已知的或者当前用户，推荐 UAS 接受请求。另一方面，如果 UAS 决定拒绝此请求，它应该生成响应 403（禁止）状态代码，并将其传送给服务器事务发送。

然而，Request-URI 识别处理请求的 UAS。如果 Request-URI 使用了 UAS 不支持的模式，它应该拒绝此请求，返回 416（不支持 URI 模式）响应。如果 Request-URI 不识别 UAS 将要接受请求的地址，它应该拒绝此请求，返回 404（没找到）响应。典型地，使用 REGISTER 方法来为具体的联系地址绑定其记录地址的 UA，将查找 Request-URI 与联系地址相等的请求。接收 Request-URI 的其它潜在资源包括，UA 发送的、建立或者更新对话的请求和响应的 Contact 头字段。

Merged Requests 如果请求在 To 头字段中没有标签，那么 UAS 核心必须检查正在进行事务的请求。如果 From 标签、Call-ID 和 Cseq 和事务精确匹配（基于第 17.2.3 节的匹配规则），那么 UAS 核心应该生成 482（发现环路）响应，并将其发送到服务器事务。

由于分发，相同的请求从不同的路径多次发送到 UAS。UAS 处理第一个接收到的这样的请求，对于第一个以外的请求，生成 482（发现环路）响应。

Require 假定 UAS 认为 Require 是处理请求的要素，那么，如果存在 Require 的话，它将检查 Require 头字段。

UAC 使用 Require 头字段告诉 UAS——为了正确地处理请求，UAC 希望 UAS 支持的 SIP 扩展。在第 20.32 节中描述了其格式。如果 UAS 不理解 Require 头字段中列出的可选标签，它必须返回状态代码 420（错误的扩展）。UAS 必须添加 Unsupported 头字段，并列出请求的 Require 头字段中它所不理解选项。

注意，在 SIP CANCEL 中不能使用 Require 和 Proxy-Require，或者发送 ACK 请求非 2xx 响应。如果在请求中出现了这些头字段，必须忽略它们。

ACK 请求 2xx 响应必须仅包含在初始请求中出现的 Require 和 Proxy-Require 值。

以下是 Require 的实例：

UAC -> AS: INVITE sip:watson@bell-telephone.com SIP/2.0

Require: 100rel

UAS -> UAC: SIP/2.0 420 Bad Extension

Unsupported: 100rel

当两边都理解了所有选项时，此行为确保了无延时地进行客户端-服务器交互；如果不理解选项，将减慢速度（如上例）。对于完好匹配的客户端-服务器对，节省了协商机制经常需要的来回路程，交互处理很快。

此外，当服务器不理解客户端请求的特征时，它可以模糊地移动。有些特征，如呼叫处理字段，仅是终端系统感兴趣的。

8.2.3 内容处理

假定 UAS 理解客户端需要的任何扩展，UAS 检查了消息体和描述它的头字段。如果不理解某一消息体的类型（Content-Type 指明的）、语言（Content- Language 指明的）或者编码（Content-Encoding 指明的），并且此消息体部分不是可选（Content-Disposition 字段指明的），那么 UAS 必须拒绝此请求，返回 415（不支持的媒体类型）响应。如果请求中包含了 UAS 不支持的消息体类型，响应必须包括 Accept 头字段，列出它所理解的所有消息体。

如果请求中包含了 UAS 不理解的编码，响应必须包括 Accept-Encoding 头字段，列出 UAS 所理解的编码。如果请求中包含了 UAS 不理解的语言，响应必须包括 Accept-Language 头字段，列出 UAS 所理解的语言。除了这些检查外，消息体处理还与方法及类型有关。关于处理具体内容头字段的更多信息，见第 7.4 节和第 20.11 到 20.15 节。

8.2.4 应用扩展

除非是请求的 Supported 头字段中指明了支持的扩展，不允许 UAS 生成响应时应用扩展。如果不支持想要的扩展，那么服务器应该依靠基准 SIP 和客户端支持的扩展。在少数情况中，没有扩展，服务器不处理请求，服务器可能发送 421（必需扩展）响应。此响应表示，没有支持指定的扩展，不能生成响应。此必需的扩展必须包含在响应的 Require 头字段中。不推荐使用此行为，因为它将破坏互操作性。

适用于非 421 响应的任何扩展必须在响应的 Require 头字段中列出来。当然，服务器不能应用响应的 Require 头字段没有列出来的扩展。结果，响应的 Require 头字段仅包含标准协议栈 RFC 中定义的可选标签。

8.2.5 处理请求

假定通过了前面子章节的所有检查，UAS 处理就成为面向具体方法的。第 10 章介绍了 REGISTER 请求，第 11 章介绍了 OPTIONS 请求，第 13 章介绍了 INVITE 请求，第 15 章介绍了 BYE 请求。

8.2.6 生成响应

当 UAS 希望为请求构造响应时，那么它遵循下面子章节介绍的通用过程。在本节没有详细介绍的，正在讨论的响应代码的其他行为，也是必需的。

一旦与创建响应相关的所有过程完成,UAS 就将这些响应发送给它所接收请求的服务器事务。

Sending a Provisional Response 生成响应大的、非面向具体方法的原则是，UAS 不应该发布非 INVITE 请求的临时响应；而是，应该尽可能地生成非 INVITE 请求的最终响应。

当生成 100（尝试）响应时，请求中的 Timestamp 头字段必须复制到 100（尝试）响应中。如果在生成响应时有延时，那么 UAS 应该将延时加到响应的 Timestamp 值中。此值必须包含以秒计算的、发送响应和接收请求的时间差。

Headers and Tags 响应的 From 字段必须和请求的 From 头字段相同，响应的 Call-ID 字段必须和请求的 Call-ID 头字段相同，响应的 CSeq 字段必须和请求的 CSeq 头字段相同，响应的 Via 字段必须和请求的 Via 头字段相同，并且必须保持相同的顺序。如果请求中包含请求的 To 标签，那么，响应中的 To 头字段必须和请求中的相同。然而，如果请求的 To 字段没有包含标签，那么，响应中的 To 头字段的 URI 必须和请求中的 To 头字段的 URI 相同，此外，UAS 必须在响应的 To 头字段中添加标签（除了 100（尝试）响应外，其中可能有标签）。这用于识别正在响应的 UAS，可能会生成对话 ID 的组件。相同的标签必须用于该请求的所有响应，包括最终和临时的（也包括 100（尝试））响应。生成标签的过程见第 19.3 节。

8.2.7 无状态 UAS 行为

无状态 UAS 是不能保持事务状态 UAS。它正常地回复请求，但是丢弃响应发送后的状态——通常 UAS 会保留其状态的。如果无状态 UAS 接收转发的请求，它将重新生成响应，并重新发送响应，就像它第一次收到请求一样。如果请求是一样的，除非请求处理的方法总是导

致相同的结果，否则 UAS 不能是无状态的。例如，此规则输出无状态注册服务器。无状态代理不使用处理层，它们直接从传输层接收到请求，并直接将响应发送给传输层。

无状态 UAS 角色主要是处理发布挑战响应的不需认证的请求。如果对不需认证的请求进行有状态的处理，那么，恶意的不需认证的请求可以创建大量的事务状态，可能会减慢或者挂掉 UAS 的呼叫处理，实际上，将造成拒绝服务；详情见第 26.5.1 节。

无状态 UAS 的大多数重要的行为如下：

- 无状态 UAS 不能够发送临时(1xx)响应。
- 无状态 UAS 不能够转发响应。
- 无状态 UAS 必须忽略 ACK 请求。
- 无状态 UAS 必须忽略 CANCEL 请求。
- 必须以无状态方式生成 To 头字段——为相同请求一贯地生成相同标签的方式，标签结构的信息见第 19.3 节。

在其它各个方面，无状态 UAS 和有状态 UAS 是一样的。对于每个新请求，UAS 可以以有状态或无状态的方式来操作。

8.3 重定向服务器

在很多架构中，需要减少负责路由请求的代理服务器的处理路径，提高信令路径的健壮性，这依靠重定向完成。

重定向允许服务器将请求的路由信息在响应中向后推给客户端，因此，为该事务它们自己跳出未来消息的环路，同时还要帮助定位请求目标，它们自己取出此事务的更多消息的环路。当请求的发起者接收到重定向时，它将发送基于其接收 URI 的新请求。将 URI 从网络核心层传播到其边界，重定向考虑到了相当多的网络伸缩性。

重定向服务器在逻辑上是由服务器处理层和访问定位服务的事务用户组成的（关于定位服务和注册服务器的更多信息见第 10 章）。定位服务实际上是数据库，它包含单个 URI 到 URI 目标可以找到的一个或多个可选地位集合的映射。

重定向服务器不发布自己的 SIP。在接收到不是 CANCEL 的请求，服务器拒绝请求；或者从定位服务中收集一系列可选地址，然后返回最终响应 3xx。对于完好格式的 CANCEL 请求，它应该返回 2xx 响应。此响应结束 SIP 处理。重定向服务器保持整个 SIP 事务的处理状态。在重定向服务器中检查向前的环路是客户端的责任。

当重定向服务器给请求返回 3xx 响应，它在 Contact 头字段填充（一个或多个）可选的地址。Contact 头字段值的 expires 参数也可以指出 Contact 数字的生命周期。

Contact 头字段包含 URI——提供要尝试的新地址和用户名，或者简单地指明其他传输参数。301（永久清除）或 302（暂时清除）响应也可以给出与初始请求目标相同的位置和用户名，但是还指明了其他传输参数，如，要尝试的不同服务器或多播地址，或 UDP 到 TCP 的 SIP 传输改变或者反之亦然——TCP 到 UDP 的 SIP 传输改变。

然而，重定向服务器不能将请求重定向到与 Request-URI 相等的 URI，相反的，如果 URI 不是指向它自己，那么服务器可以将请求代理到目的 URI，也可以用 404 响应拒绝请求。

如果客户端使用带外代理，并且此代理实际上重定向了请求，那么可能出现无限的重定向环路。

注意，Contact 头字段值可以指向与起始呼叫不同的资源。例如，SIP 与 PSTN 网关的呼叫连接可能需要发送具体的信息声明，如，“你所呼叫的号码已经改变”。

Contact 响应头字段可以包含适当的 URI,指示呼叫者可以到达的地方——不限于 SIPS URI。例如，它可以包含电话、传真、Internet 在线聊天系统（如果定义了）和 mail to:(RFC 2368 [32]) URL 的 URI。第 26.4.4 节讨论了将 SIPS URI 重定向到非 SIPS URI 的蕴含和限制。

Contact 头字段的 expires 参数表示 URI 有效的时间。此参数值是以秒表示的数字。如果不提供此参数，那么 expires 头字段的值就决定 URI 有效的时间。不规范的值应该视为等于 3600。

这提供了适度的向后兼容 RFC2543——在头字段中允许绝对时间。如果接收到绝对时间，那么将视为不规则，默认为 3600。

重定向服务器必须忽略不理解的特征（包括不识别的头字段，Require 和方法名中不可知的可选标签），并处理正在讨论的重定向请求。

9 取消请求

前面的章节已经为所有方法讨论了生成请求和处理请求响应的通用 UA 行为的所有方法。在本章，我们将讨论称为“CANCEL”的通用方法。

CANCEL 请求，如其名字所示，是用于取消客户端发送的前一请求。特别的是，它要求 UAS 停止处理请求，并为请求生成错误响应。对占用服务器很长时间才能响应的请求来说，CANCEL 请求是最有用的。因为这样，CANCEL 请求在占用服务器很长时间响应的地方很有用。为此，对需要很长时间生成响应的 INVITE 请求，CANCEL 是最好的。按照常例，接收到 INVITE 的 CANCEL 请求，但是没有发送最终响应的 UAS，可能会“停止响铃”，然后给 INVITE 发送特定的错误代码（a 487）。

代理和用户代理客户端都可以构造 CANCEL 请求。第 15 章讨论了在什么情况下，UAC 会取消 INVITE，第 16.10 节讨论了代理使用 CANCEL 的方式。

有状态代理对 CANCEL 做出响应，而不是简单地转发它从下游元素接收的响应。因此，因为每个有状态代理的跳跃点都响应它，CANCEL 被称为“逐跳”请求。

9.1 客户端行为

不应该发送 CANCEL 取消 INVITE 外的请求。

因为 INVITE 外的请求是立即响应的，为非 INVITE 发送 CANCEL 会创造竞态条件。

使用下列过程来构造 CANCEL 请求。CANCEL 请求中的 Request-URI、Call-ID、To、Cseq 的数字部分和 From 头字段，必须与要取消的请求中的一样，包括标签。客户端构造的 CANCEL 必须有单一的 Via 与要取消的请求中的最上面 Via 值匹配。这些头字段使用相同的值，允许 CANCEL 和它要取消的请求匹配（第 9.2 节指出了怎样发生这些匹配）。然而，此方法的 CSeq 头字段必须有 CANCEL 值。这允许它凭借自身的实力确定和处理为事务（见第 17 章）。

如果要取消的请求包括 Route 头字段，那么 CANCEL 请求必须包括此 Route 头字段值。

这是必需的以便于无状态代理可以正确地路由 CANCEL 请求。

CANCEL 请求不能包含任何 Require 和 Proxy-Require 头字段。

一旦构造了 CANCEL，客户端应该检查是否接收到要取消请求（这里指“原始请求”）的任何（临时或者永久）响应。

如果没有接收到临时响应，必须发送 CANCEL 请求，当然，客户端必须等待发送请求之前的临时响应。如果原始请求产生最终响应，不应该发送 CANCEL，因为它是有效的无操作，而 CANCEL 对已经生成最终响应的请求没有影响。

当客户端决定发送 CANCEL 时，它为 CANCEL 创建客户端事务，并将其和 CANCEL 请求以及目的地址、端口和传输一起发送。CANCEL 请求的目的地址、端口和传输必须和发送请求的一样。

在接收到之前请求的响应之前，如果允许发送 CANCEL，那么，服务器将在原始请求之前接收到 CANCEL。

注意，原始请求对应的事务和 CANCEL 事务是完全独立的。然而，UAC 取消请求不依靠接收原始请求的 487（请求超时）响应，因为与 RFC2543 兼容的 UAS 不会生成这样的响应。如果在 64*T1 秒内没有原始请求的最终响应（第 17.1.1 节定义了 T1），客户端应该考虑取消的原始事务，并且应该破坏客户端事务处理原始请求。

9.2 服务器行为

CANCEL 方法要求服务器边的 TU 取消挂起的事务。TU 确定使用 CANCEL 请求取消的事务，然后假定请求方法是 CANCLE 和 ACK 之外的任何方法，并应用第 17.2.3 节的事务匹配过程。匹配事务是要取消的一个。

服务器的 CANCEL 程序请求与服务器的类型有关。无状态代理将转发 CANCEL，有状态代理可以响应 CANCEL，并生成自己的 CANCEL 请求，同时，UAS 将响应 CANCEL。关于代理处理 CANCEL 的信息见第 16.10 节。

UAS 首先根据第 8.2 节介绍的通用 UAS 处理方法处理 CANCEL 请求。然而，因为 CANCEL 请求是逐跳的，并且不可以重新提交，为了在 Authorization 头字段得到正确的凭证，服务器不会向它们发出挑战。注意，CANCEL 请求也不包含 Require 头字段。

如果 UAS 没有找到与上面过程的 CANCEL 匹配的事务，那么它应该返回 481（呼叫腿/事务不存在）响应。如果原始请求的事务仍然存在，接收到 CANCEL 请求的 UAS 行为与它是否已经发送了原始请求的最终响应有关。如果它对会话状态和原始请求生成的响应没有影响，那么 CANCEL 请求对原始请求的处理没有影响。如果 UAS 没有发布原始请求的最终响应，那么，其行为和原始请求的方法有关。如果原始请求是 INVITE，UAS 应该立即响应 INVITE，返回 487（请求结束）。CANCEL 请求对本规范中定义的其它方法的事务处理没有影响。

不管原始请求的方法是什么，只要 CANCEL 可以匹配现有的事务，那么，UAS 回答 CANCEL 请求 200（OK）响应。遵循第 8.2.6 节介绍的过程构造此响应。注意，CANCEL 响应的 To 标签和原始请求响应的 To 标签应该是一样的。CANCEL 响应传送给服务器事务发送。

10 注册

10.1 概述

SIP 提供了发现机制。如果用户要发起和另一个用户的会话，SIP 必须发现可到达目的用户的当前主机。发现处理经常是 SIP 网络元素完成，比如代理服务器和重定向服务器——

它们负责接收请求，决定要发送请求的用户位置，然后将它发送到相应的位置。为了完成这些，SIP 网络元素查询了抽象服务——定位服务，这提供了特定域的地址绑定。这些地址绑定将输入的 SIP 和 SIPS URI（如，sip:bob@biloxi.com）映射到想要的用户“更近”的一个或多个 URI（如，sip:bob@engineering.biloxi.com）。最后，代理将查询定位服务，将接收到的 URI 映射到想要的接收者常驻的用户代理。

注册创建了特定域中定位服务的绑定，它将记录地址 URI 和一个或者多个联系地址相关联。因此，当域中的代理接收 Request-URI 和记录地址匹配的请求时，代理将请求转发给记录地址已注册的地址。通常，当请求记录地址路由到域中时，在域定位服务注册记录地址才有意义。在大多数情况中，这意味着注册的域需要与记录地址的 URI 域匹配。

建立定位服务内容有很多种方法。其中之一是管理。在上面的实例中，通过访问公司数据库可以知道 Bob 是工程部门的一员。但是，SIP 为 UA 提供了一种机制明确地创建绑定。这种机制就是注册。

注册必须发送 REGISTER 请求给特定类型的 UAS——即是注册服务器。注册服务器作为域中定位服务的前端，发送和写基于 REGISTER 内容的映射。随后主要是负责此域路由请求的代理服务器查询此定位服务。

图 2 说明了全部的注册过程。注意，在网络中，注册服务器和代理服务器可能是同一个设备所扮演的逻辑角色，为了清楚地说明其作用，本图中将它们分开了。同时也要注意，如果是两个分开的元素，那么 UA 可能通过代理服务器发送请求，以便于可以到达注册服务器。

SIP 不强制执行定位服务的具体机制。唯一的要求是域中的注册服务器必须能够在定位服务中读和写数据，域中的代理和重定向服务必须能够读取相同的数据。在同一域内，注册服务器与特定的 SIP 代理服务器可以在同一接点。

10.2 构造 REGISTER 请求

REGISTER 请求添加、删除和查询绑定。REGISTER 请求可以在记录地址和一个或多个联系地址之间添加新绑定。合适地通过认证的第三方可以完成代表特定记录地址的注册。客户端也可以删除以前的绑定或者查询绑定，确定记录地址绑定当前所在的位置。

除非另有说明，REGISTER 请求的构造和客户端发送 REGISTER 请求的行为与第 8.1 节和第 17.1 节中介绍的通用 UAC 行为是一样。

REGISTER 请求不建立对话。UAC 可以在 REGISTER 请求中包括基于第 8.1 节介绍的预有的路由集合的 Route 头字段。在 REGISTER 请求和响应中的 Record-Route 头字段没有意义，如果存在，必须忽略。特别的，UAC 不能根据 REGISTER 请求的任何响应中存在和缺少的 Record-Route 头字段创建新的路由集合。

下列头字段，除了 Contact 外，必须包含在 REGISTER 请求中。当然，也可以包括 Contact 头字段。

Request-URI: Request-URI 指定了注册服务器指明的定位服务域。（如 sip:chicago.com）。不能出现 SIPS URI 的 userinfo 和@组件。

To: To 头字段包括记录地址，可以创建、查询和修改其注册。To 头字段和 Request-URI 字段主要的不同是，前者包含用户名。此记录地址必须是 SIP 或者 SIPS URI。

From: From 头字段包含负责注册的人的记录地址。除非是第三方注册，此值和 To 头字段的值是一样的。

Call-ID: UAC 所有的注册应该使用与发送到注册服务器的注册相同的 Call-ID 头字段值。

如果相同的客户端使用不同的 Call-ID 值，那么注册服务器不能检测延时的 REGISTER 请求是否没有排序到达。

CSeq:CSeq 值保证 REGISTER 请求适当的排序。对于每个使用相同的 Call-ID 的 REGISTER 请求，UA 必须逐一增加 Cseq 值。

Contact: REGISTER 请求可能包括有一个或多个地址绑定值的 Contact 头字段。

直到它们接收到来自注册服务器的前一请求的最终响应,或之前的REGISTER请求超时，UA 才能发送新的注册（即是包含与转发相对的新 Contact 头字段值）。

在 REGISTER 请求中，下面的 Contact 头参数有特定的意义。

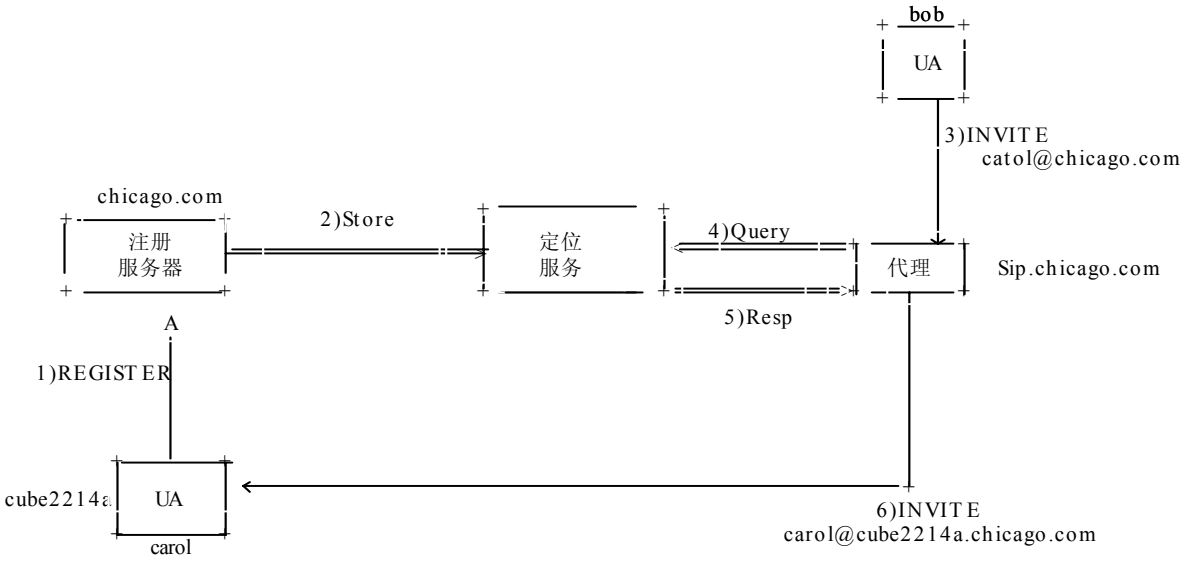

图 2：注册实例

action:RFC 2543 不赞成 action 参数。UAC 不应该使用 action 参数。

expires:expires 参数表示了 UA 绑定的有效时间。此参数值是表示秒的数字。如果不提供此参数，那么将使用 expires 头字段的值代替。不规范的值应该视为等于 3600。

10.2.1 添加绑定

发送给注册服务器的 REGISTER 请求包括 SIP 请求应该转发给记录地址的地址。记录地址包括在 REGISTER 请求的 To 头字段中。

请求的 Contact 头字段值主要是由 SIP 和 SIPS URI 组成的,它确定具体的 SIP 终端(如，“sip:carol@cube2214a.chicago.com”)，同时，也可以使用其它的 URI 模式。例如，SIP UA 可以选择注册电话号码（使用 tel URL，RFC 2806[8]）和 email 地址（使用 mail to URL，RFC 2368 [32]）作为的记录地址的联系人。

例如，Carol，使用记录地址“sip:carol@chicago.com”在 SIP 注册服务器的 chicago.com 域注册。chicago.com 域的代理服务器将使用 Carol 的注册，将 Carol 的记录地址请求路由到其 SIP 终端。

一旦客户端在注册服务器上建立了绑定，如果需要，它可以发送包含新绑定和修订现有绑定的随后注册。REGISTER 请求的 2xx 响应，将在 Contact 头字段中包含完整的、在此

注册服务器上注册记录地址的绑定列表。

如果 REGISTER 请求 To 头字段的记录地址是 SIPS URI，那么，请求的 Contact 头字段值也应该是 SIPS URI。当以其它方式来保证联系地址 表示的资源的安全时，客户端仅注册 SIP 记录地址下的非 SIPS URI。这可适用于调用不是 SIP 协议的 URI 和不是 TLS 协议保证其安全性的 SIP 设备。

注册不需要更新绑定。典型的是，UA 仅更新其自己的联系地址。

Setting the Expiration Interval of Contact Addresses 当客户端发送 REGISTER 请求时，它可以建议逾时间隔的状态，表示客户端注册有效的时间。（如第 10.3 节所描述的，注册服务器基于其本地策略选择实际的时间间隔。）

有两种方法可用于客户端建议绑定的逾时间隔：通过 Expires 头字段或者 expires Contact 头参数。当在单个 REGISTER 请求中给出多个绑定时，后者允许在预先绑定的基础上建议逾时间隔，但是前者建议的逾时间隔适用于不包含 expires 参数的所有 Contact 头字段值。

如果在 REGISTER 中，表示建议到期时间的两种机制都不存在，那么，客户端表示，它要服务进行选择。

Preferences among Contact Addresses 如果 REGISTER 请求发送多个 Contact，那么，注册 UA 要将所有的 Contact 头字段值的 URI 和 To 字段的记录地址相关联。此列表可以使用 Contact 头字段的‘q’参数排出优先级。Q 参数表示具体 Contact 头字段值与记录地址其它绑定的相关优先级。第 16.6 节介绍了代理服务器怎样使用此优先指示。

10.2.2 删除绑定

注册是软状态，除非是更新才到期，但是也可以明确地删除。客户端可以影响第 10.2.1 节介绍的注册服务器选择的逾时间隔。通过在 REGISTER 请求的联系地址指定“0”逾时间隔，UA 请求立即删除绑定。UA 应该支持这种机制，以便于可以在逾时间隔到期之前删除绑定。

REGISTER-specific Contact 头字段值“*”用于所有的注册，但是，如果 Expires 头字段不用值“0”表示，不能使用“*”。

使用“*”Contact 头字段值允许注册——在不知道精确值时，删除与记录地址相关的所有绑定。

10.2.3 提取绑定

不管请求是否包含有 Contact 头字段，REGISTER 请求成功的响应包含全部的现有绑定。如果 REGISTER 请求中没有 Contact 头字段，那么绑定列表左边不改变。

10.2.4 更新绑定

每个 UA 负责更新它之前建立的绑定。UA 不应该更新其它 UA 建立的绑定。

注册服务器的 200（OK）响应包含一系列的 Contact 头，列举了所有的当前绑定。使用第 19.1.4 节的比较规则，UA 比较每个联系地址，检查它是否创建了联系地址。如果是，根据 expires 参数——如果没有，就根据 Expires 字段值更新逾时间隔。UA 随后在逾时间隔结束之前，为其每个绑定发布 REGISTER 请求。它也可以在 REGISTER 请求中，组合几个更新。

在单个引导周期，UA 应该为所有的注册使用相同的 Call-ID。除非有重定向，注册更新应该发送到与原始注册相同的网络地址。

10.2.5 设置内部时钟

如果 REGISTER 请求的响应包含有 Data 头字段，那么客户端可以使用此头字段获取当前的时间，设置内部时钟。

10.2.6 发现注册服务器

UA 可以使用三种方法来确定发送注册的地址：通过配置、使用记录地址和多播。可以用注册服务器地址配置 UA，这种方法超出了本规范的范围。如果没有可配置的注册服务器地址，那么，UA 应该使用通用的 SIP 服务器定位机制，将主机部分的记录地址作为请求的 Request-URI 和地址。例如，UA 为用户“sip:carol@chicago.com”将 REGISTER 请求寻址到“sip:chicago.com”。

最后，UA 可以配置成多播。多播注册都编址为已知的“所有 SIP 服务器”多播地址“sip.mcast.net”(224.0.1.75 for IPv4)。现在没有分配已知的 Ipv6 的多播地址；当需要时，将单独说明这种分配。SIP UA 可以侦听此地址，并使用它来知道其它本地用户的当前位置（见[33]）；但是，它们并不响应请求。

在有些情况中，多播注册可能不适用，例如，如果多播事务共享相同的本地网络。

10.2.7 发送请求

一旦构造了 REGISTER 方法，并确定了消息的目的地，那么，UAS 遵循第 8.1.2 节介绍的过程，将 REGISTER 请求发送给处理层。如果因为 REGISTER 没有响应，处理层返回超时错误，UAS 不应该立即向相同的注册服务器再尝试注册。

立即再尝试有可能也超时。为造成超时的条件等待合理的时间间隔，可以减少不必要的网络负载。没有强制具体的时间间隔。

10.2.8 错误响应

如果 UA 接收到 423（时间间隔太短）响应，在使得 REGISTER 请求中的所有联系地址逾时间隔等于或大于 423（时间间隔太短）响应 Min-Expires 头字段的逾时间隔后，它可以再注册。

10.3 处理 REGISTER 请求

注册服务器是 UAS，在其管理域内，它响应 REGISTER 请求，并保留可以访问代理服务器和重定向服务器的一系列绑定。注册服务器遵循第 8.2 节和第 17.2 节处理响应，但是它只接受 REGISTER 请求。注册服务器不能生成 6xx 响应。

在适当的时候，注册服务器可以重定向 REGISTER 请求。通用的用法是，注册服务器侦听多播接口，用 302（暂时清除）响应将多播 REGISTER 请求重定向到其自己单播接口。

如果 Record-Route 包含在 REGISTER 请求中，那么，注册服务器必须忽略 Record-Route 头字段。注册服务器不能在 REGISTER 请求的响应中包含 Record-Route 头字段。

注册服务器可能接收到穿越代理的请求，它认为 REGISTER 是未知的请求，并添加 Record-Route 头字段值。

注册服务器必须知道（例如，通过配置）它所保留绑定的域。注册服务器必须按照其接收的顺序处理 REGISTER 请求。必须能够基本处理 REGISTER 请求，意味着具体的 REGISTER 请求可以完全处理，也可以一点都不处理。必须独立于其它注册和绑定改变，处理每个

REGISTER 请求。

当接收到 REGISTER 请求，注册服务器遵循以下步骤：

1、注册服务器检查 Request-URI，确定它是否可以访问 Request-URI 指定域的绑定。如果不能，并且，如果服务器也当作代理服务器，那么，服务器应该遵循第 16 章介绍的代理消息的通用行为，将请求转发给寻址域。

2.为了保证注册服务器支持任何必要的扩展,注册服务器必须像第 8.2.2 节介绍的 UAS 一样处理 Require 头字段。

3.注册服务器应该认证 UAC。第 22 章介绍了 SIP 用户代理的认证机制。注册服务器的行为绝对不会不考虑 SIP 通用认证框架。如果认证机制不可用，那么注册服务器可以将 From 地址作为请求发起者声明的身份。

4.注册服务器应该确定，认证用户是否有权修改记录地址的注册。例如，注册服务器可能查询授权数据库——它映射了用户名和用户有权修改的一系列记录地址。如果认证用户无权修改绑定，注册服务器必须返回 403（禁止），并跳过剩下的步骤。

在支持第三方注册的架构中，实体可以负责更新多个记录地址相关的注册。

5.注册服务器从请求的 To 头字段取出记录地址。如果记录地址不可用于 Request-URI 域，那么注册服务器发送 404（未找到）响应，并跳过剩下的步骤。URI 必须转换成规范的格式。为了实现这一点，必须删除所有的 URI 参数（包括 user-param），同时将所有的转义字符串转换成保留格式。此结果用作一系列绑定的索引。

6.注册服务器检查请求是否包含 Contact 头字段。如果没有，跳到最后一步。如果有 Contact 头字段，注册服务器检查，Contact 头字段是否包含了特殊值“*”和 Expires 字段。如果请求有其他的 Contact 头或者非零的到期时间，那么，请求是无效的，服务器返回 400（无效的请求）响应，并跳过剩下的步骤。如果没有，注册服务器检查，Call-ID 是否和每个绑定的存储值一致。如果没有，它必须删除这些绑定。如果一致，仅在请求的 Cseq 高于绑定的存储值时，删除绑定。否则，必须放弃更新，同时请求失败。

7.注册服务器现在依次处理 Contact 头字段的每个联系地址。对于每个地址，它按照下面的方法确定逾时间隔：

- 如果字段值有 expires 参数，此值必须当作请求的到期时间。
- 如果没有这样的参数，但是请求有 Expires 头字段，此值必须当作请求的到期时间。
- 如果都没有，本地配置的默认值必须当作请求的到期时间。

注册服务器可以选择小于请求的逾时间隔的到期时间。当且仅当请求的到期时间大于零，并且小于一个小时，同时小于注册服务器配置的最小值，注册服务器可以拒绝注册，并返回 423（时间间隔太短）响应。此响应必须包含 Min-Expires 头字段——说明注册服务器想要的最小逾时间隔。然后，它跳过剩下的步骤。

在限制需要保持的状态和减少可能的注册停滞的同时，允许注册服务器设置注册时间间隔，防止过于频繁的注册更新。注册的逾时间隔频繁地用于创建服务上。其中一个实例是，follow-me 服务，在这里，终端用户可能仅在一个很短的周期内可用。因此，注册服务器应该接受简短注册，如果时间间隔很短以至于更新会降低注册服务器的性能，那么，应该拒绝该请求。

对于每个地址，注册服务器随后使用 URI 比较规则搜索当前的绑定列表。如果绑定不存在，将暂时添加它。如果绑定存在，注册服务器检查 Call-ID 值。如果现有绑定的 Call-ID

值与请求中的 Call-ID 值不同，如果逾时间隔为零或者有其它更新，必须删除绑定。如果它们是相同的，那么注册服务器比较 CSeq 值。如果此值高于现有绑定的值，它必须更新或者删除绑定。否则，必须放弃更新，同时请求失败。

此算法确保了忽略相同 UA 无序的请求。

每个绑定记录记录了请求的 Call-ID 和 Cseq 值。

当且仅当，绑定更新和添加成功，必须提交绑定更新（即是使得代理和重定向服务器可见）。如果其中之一失败了（例如，因为后台数据库提交失败），那么，请求必须失败，返回 500（服务器错误）响应，同时必须删除所有尝试的绑定更新。

8.注册服务器返回 200（OK）响应。响应必须包含列出了所有当前绑定的 Contact 头字段值。每个 Contact 值必须对注册服务器选择的、说明其逾时间隔的“expires”参数起作用。此响应应该包含 Date 头字段。

11 查询能力

SIP 方法 OPTIONS 允许 UA 查询其它 UA 和代理服务器的能力。这就允许客户端不必“打电话”另一方，发现关于支持的方法、内容类型、扩展和编码等等的信息。例如，在客户端将 Require 头字段插入到 INVITE 列出的它所不确定目的 UAS 支持的选项时，客户端可以使用 OPTIONS 查询目的 UAS，检查 Supported 头字段是否返回此选项。所有的 UA 必须支持 OPTIONS 方法。

Request-URI 确定 OPTIONS 请求的目标，它可以识别其它的 UA 和 SIP 服务器。如果 OPTIONS 寻址到代理服务器，那么 Request-URI 设置为没有用户部分，和 REGISTER 请求的 Request-URI 设置一样。

换言之，服务器接收到 Max-Forwards 头字段为 0 的 OPTIONS 请求，可能不管 Request-URI 而直接响应请求。

此行为和 HTTP/1.1 相同。通过发送一系列有递增 Max-Forwards 值的 OPTIONS 请求，此行为可以用作“路由跟踪”功能，检查单个跳跃点服务器的能力。

作为通用 UA 的行为，如果 OPTIONS 没有响应，处理层可以返回超时错误。这可以说明目标不可到达，因此是无效的。

OPTIONS 请求可以作为建立对话的一部分发送，查询在对话中以后可能使用的对等物的能力。

11.1 构造 OPTIONS 请求

使用第 8.1.1 节讨论的 SIP 请求的标准规则来构造 OPTIONS 请求。

Contact 头字段可能出现在 OPTIONS 中。

应该包括 Accept 头字段，说明在响应中 UAC 想要接收到的消息体的类型。这主要用来设置一种用于描述 UA 媒体能力的格式，如 SDP（application/sdp）。

OPTIONS 请求的响应假定为是在原始请求 Request-URI 的范围内的。然而，仅当 OPTIONS 作为建立对话的一部分发送时，它才能保证生成 OPTIONS 响应的服务器可以接收到将来的请求。

以下是 OPTIONS 请求的实例：

OPTIONS sip:carol@chicago.com SIP/2.0

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKhjhs8ass877

Max-Forwards: 70

To: <sip:carol@chicago.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 63104 OPTIONS

Contact: <sip:alice@pc33.atlanta.com>

Accept: application/sdp

Content-Length: 0

11.2 处理 OPTIONS 请求

使用第 8.2.6 节讨论的 SIP 响应的标准规则来构造 OPTIONS 响应。选择的响应代码必须和 INVITE 请求已经选择的一样。即使，如果准备接受呼叫，返回 200 (OK)；如果 UAS 忙，返回 486（这儿正忙）等等。这允许 OPTIONS 请求用作确定 UAS 的基本状态，这可以是 UAS 是否接受 INVITE 请求的指示。

在对话中接收到 OPTIONS 请求生成 200 (OK)响应，和对话外构造的一样，对对话没有任何影响。

因为代理处理 OPTIONS 和 INVITE 请求的不同,OPTIONS 的使用有局限性。分发的 INVITE 可能返回多个 200(OK)响应，而分发的 OPTIONS 可能只返回一个 200(OK)响应，因为代理使用非 INVITE 处理机制处理的。标准细节见第 16.7 节。

如果代理服务器生成 OPTIONS 响应，那么，代理返回 200(OK)，列出服务器的能力。此响应不包括消息体。

Allow、Accept、Accept-Encoding、Accept-Language 和 Supported 头字段应该出现在 OPTIONS 请求的 200 (OK)响应中。如果代理产生此响应，因为代理不知道方法，Allow 是模糊的，所以代理应该忽略 Allow 头字段。Contact 头字段可能出现在 200 (OK)响应中，并和 3xx 响应有相同的语义。即是它们可能列出到达用户的一系列可选名字和方法。Warning 头字段也可能出现。

消息体可能发送，其类型由 OPTIONS 请求的 Accept 头字段确定（如果 Accept 头字段不存在，默认值为 application/sdp)。如果此类型包括可以描述媒体能力的类型，那么，UAS 应该为此目的在响应中包括消息体。在[12]中介绍了在 application/sdp 情况下，构造此消息体的详情。

以下是 UAS 创建的 OPTIONS 响应实例(和第 11.1 节的请求相对应):

SIP/2.0 200 OK

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKhjhs8ass877

;received=192.0.2.4

To: <sip:carol@chicago.com>;tag=93810874

From: Alice <sip:alice@atlanta.com>;tag=1928301774

```
Call-ID: a84b4c76e66710
CSeq: 63104 OPTIONS
Contact: <sip:carol@chicago.com>
Contact: <mailto:carol@chicago.com>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE
Accept: application/sdp
Accept-Encoding: gzip
Accept-Language: en
Supported: foo
Content-Type: application/sdp
Content-Length: 274
(SDP not shown)
```

12 对话

用户代理一个重要的概念是对话。对话表示两个用户代理之间持续存在的对等关系。对话有助于用户代理之间的序列消息和它们之间正确的路由信息。对话表示解释 SIP 消息的内容。第 8 章讨论了与 UA 处理无关的、对话外的请求和响应。本节将讨论怎样在对话内发送随后的请求和响应。

每个 UA 用对话 ID 标识对话，对话 ID 由 Call-ID 值、本地标签和远程标签组成。每个 UA 中包含在对话内的对话 ID 是不同的。特别的是， UA 的本地标签和其对等的 UA 的远程标签是一样的。标签是不透明的令牌，有助于生成唯一的对话 ID。

对话 ID 也与包含 To 字段标签的所有请求和响应相关联。计算消息对话 ID 的规则和 SIP 元素是 UAC 还是 UAS 有关。对于 UAC，对话 ID 的 Call-ID 值设置为消息的 Call-ID，远程标签设置成消息 To 字段的标签，本地标签设置成消息 From 字段的标签（这些规则适用于请求和响应）。当对话希望是 UAS，那么，其对话 ID 的 Call-ID 值设置为消息的 Call-ID，远程标签设置成消息 From 字段的标签，本地标签设置成消息 To 字段的标签。

对话包括对话内的下一消息转发所需的特定状态。此状态由对话 ID、本地序列号（用于排列 UA 到其对等物的请求）、远程序列号（用于排列其对等物到 UA 的请求）、本地 URI、远程 URI、远程目标、“安全”的布尔型标记和路由集合组成，这是 URI 的有序列表。路由集合是需要穿越服务器发送请求到对等物的列表。对话也可以是在“早期”状态，当它和临时响应一起创建时，出现此状态；当 2xx 最终响应到达时，转换成“确认的”状态。对于其他的响应，或者如果在对话中根本没有到达，那么，早期对话结束。

12.1 创建对话

通过使用具体的方法生成请求的非错误响应来创建对话。

在本规范内，仅当请求是 INVITE， 2xx 和 101-199 响应有 To 标签时，建立对话。请求的非最终响应建立的对话是“早期”状态，称为早期对话。在创建对话时，可以以其他方式定义扩展。第 13 章给出了具体 INVITE 方法的详细介绍。在这里我们仅介绍不是用此方法创建的对话状态。

UA 必须分配值给下面介绍的对话 ID 组件。

12.1.1 UAS 行为

当 UAS 用建立对话的响应（如，2xx to INVITE）响应请求时，UAS 必须将请求的所有 Record-Route 头字段值复制到响应中（包括，UA 已知或者未知的 URI、URI 参数和 Record-Route 头字段参数），并且必须保留这些值的顺序。UAS 必须在响应中添加 Contact 头字段。Contact 头字段包含地址，此地址是在对话中 UAS 要联系随后请求的地址（包括 INVITE 中 2xx 响应的 ACK）。通常，此 URI 的主机部分是 IP 地址或者主机的正式域名。Contact 头字段提供的 URI 必须是 SIP 或 SIPS URI。如果发起对话的请求包含 Request-URI 的 SIPS URI 或者 Record-Route 头字段最上面值——如果有的话，或者没有 Record-Route 头字段时，就是 Contact 头字段，响应的 Contact 头字段必须是 SIPS URI。URI 应该是全局范围（即是在对话外的消息使用相同的 URI）。同样的，INVITE 的 Contact 头字段的 URI 的范围也不限于此对话中。因此，即使是对话外，UAC 消息中也可以使用此 URI。

UAS 随后构造对话的状态。在整个对话期间，必须保留此状态。

如果请求通过 TLS，Request-URI 包含 SIPS URI，那么“安全”标签设置为 TRUE。

路由集合必须设置为一系列请求的 Record-Route 头字段 URI，按顺序取出，并保留所有的 URI 参数。如果在请求中没有 Record-Route 头字段，路由集合必须设置为空集合。此路由集合，即使是空的，也要覆盖现有的路由集合，用于对话中的下一请求。远程目标必须设置成请求的 Contact 头字段的 URI。

远程序列号必须设置成请求的 CSeq 头字段的序列号值。本地序列号值必须为空。对话 ID 的呼叫识别符组件必须设置成请求的 Call-ID 值。对话 ID 的本地标签组件必须设置成请求（通常包含标签）的响应中 To 字段的标签，对话 ID 的远程标签组件必须设置成请求中的 From 字段的标签。UAS 必须准备好接收 From 字段无标签的请求，在这种情况中，认为标签有空值。

这保持了与 RFC 2543 向后的兼容性， RFC 2543 没有强制 From 标签。

远程 URI 必须设置成 From 字段的 URI，本地 URI 必须设置成 To 字段的 URI。

12.1.2 UAC 行为

当 UAC 发送可以建立对话的请求时（如，INVITE），它必须在请求的头字段提供全局范围的 SIP 或 SIPS URI（即，相同的 SIPS URI 可用于对话之外）。如果请求有 Request-URI 或 Route 头字段最上面值有 SIPS URI，Contact 头字段必须包含 SIPS URI。

当 UAC 接收到建立对话的响应，它构造对话的状态。在整个对话期间，必须保留此状态。

如果请求通过 TLS，Request-URI 包含 SIPS URI，那么“安全”标签设置为 TRUE。

路由集合必须设置为一系列响应的 Record-Route 头字段 URI，按顺序取出，并保留所有的 URI 参数。如果在请求中没有 Record-Route 头字段，路由集合必须设置为空集合。此路由集合，即使是空的，也要覆盖现有的路由集合，用于对话中的下一请求。远程目标必须设置成响应的 Contact 头字段的 URI。

本地序列号必须设置成响应的 CSeq 头字段的序列号值。远程序列号值必须为空（当远程 UA 发送对话内的请求时，建立）。对话 ID 的呼叫识别符组件必须设置成请求的 Call-ID 值。对话 ID 的本地标签组件必须设置成请求（通常包含标签）的请求中 From 字段的标签，对话 ID 的远程标签组件必须设置成响应中的 To 字段的标签。UAC 必须准备好接收 To 字段无标签的响应，在这种情况中，认为标签有空值。

这保留了与 RFC 2543 向后的兼容性，RFC 2543 没有强制 To 标签。

远程 URI 必须设置成 To 字段的 URI，本地 URI 必须设置成 From 字段的 URI。

12.2 对话中的请求

一旦在两个 UA 之间已经建立了对话，在需要时，任何一方都可以发起新事务。发送请求的 UA 将起到事务的 UAC 的作用。接收请求的 UA 将起到 UAS 的作用。注意，这可能是在建立对话的事务期间，不同于 UA 所保留的不同角色。

对话内的请求可以包含 Record-Route 和 Contact 头字段。然而，虽然它们可以修改远程目标的 URI，这些请求不能修改对话的路由集合。特别的是，没有以更新请求为目标的请求不修改对话远程目标 URI，以更新请求为目标的请求修改对话远程目标 URI。对于用 INVITE 建立的对话，定义的目标更新请求是 re-INVITE（见第 14 章）。其他扩展可能定义以其他方式建立对话的不同目标更新请求。注意，ACK 不是目标更新请求。

目标更新请求仅更新对话远程目标 URI，并不更新 Record-Route 的路由集合。更新后者将引入与 RFC2543 兼容系统的向后兼容的严重问题。

12.2.1 UAC 行为

Generating the Request 使用存储在对话中的状态组件构造对话内的请求。

请求的 To 字段中的 URI 必须设置成对话状态的远程 URI。请求的 To 字段中的标签必须设置成对话 ID 的远程标签。请求的 From URI 必须设置成对话状态的本地 URI。请求的 From 头字段中的标签必须设置成对话 ID 的本地标签。如果远程或本地标签为 null，标签参数必须分别从 To 或者 From 头字段忽略。

在随后请求内使用原始请求的 To 或 From 字段的 URI 是和 RFC 2543 向后兼容的，这使用了对话识别符的 URI。在本规范中，此标签仅用于对话识别符。期望在本规范的下一版本中反对在对话中请求强制反映原始 To 和 From URI。

请求的 Call-ID 必须设置对话的 Call-ID。对话内的请求在每个间接寻址必须严格包含单调增加的连续的 Cseq 序列号（逐次增一）。（当然除了 ACK 和 CANCEL，其序列号等于确认或者取消请求的序列号。）因此，如果本地序列号不为空，本地序列号的值必须逐次增一，此值必须放在 Cseq 头字段中。如果本地序列号为空，必须使用第 8.1.1 节的原则选择此初始值。CSeq 头字段值的方法字段必须和请求的方法匹配。

在单个呼叫中，使用 32 位长度的序列号，客户端可以每秒产生一个需要 136 年才能绕回的请求。选择序列号的初始值，以便于在相同呼叫中的随后请求不会绕回。非零的初始值允许客户端使用基于时间的初始序列号。

客户端可以，例如，在 32 位秒时钟中选择 31 个有效位作为初始序列号。

UAC 使用远程目标和路由集合来建立请求的 Request-URI 和 Route 头字段。

如果路由集合为空，那么 UAC 必须将远程目标 URI 放在 Request-URI 中。UAC 不能在请求中添加 Route 头字段。

如果路由集合不为空，并且路由集合的第一个 URI 包含 lr 参数（见第 19.1.1 节），那么 UAC 必须将远程目标 URI 放在 Request-URI 中，同时依次引用 Route 头字段包含的路由集合，包括参数。

如果路由集合不为空，并且其第一个 URI 不包含 lr 参数（见第 19.1.1 节），那么 UAC 必须将远程目标 URI 放在 Request-URI 中，同时，去掉 Request-URI 中不允许的参数。UAC 必须依次引用 Route 头字段包含的剩余路由集合，包括参数。然后，UAC 必须将远程目标 URI 作为最后一个值放在 Route 头字段中。

例如，如果远程目标是 sip:user@remoteua，并且路由集合包含：

<sip:proxy1>,<sip:proxy2>,<sip:proxy3;lr>,<sip:proxy4>

将和下列 Request-URI 和 Route 头字段一起形成请求：

METHOD sip:proxy1

Route: <sip:proxy2>,<sip:proxy3;lr>,<sip:proxy4>,<sip:user@remoteua>

如果路由集合的第一个 URI 不包含 lr 参数，代理表明不理解本文档中介绍的路由机制，那么将使用 RFC2543 说明的机制，在转发消息时，用接收到的第一个 Route 头字段代替 Request-URI。将 Request-URI 放在 Route 头字段的尾部，防止了 Request-URI 中的信息穿越严格的路由器（当请求到达松散的路由器时，将返回到 Request-URI）。

UAC 应该在对话目标更新请求中包含 Contact 头字段，除非是需要改变它，URI 应该是和之前的对话内请求使用的一样。如果“安全”标志为 TRUE，那么，URI 必须为 SIPS URI。

如 12.2.2 介绍的那样，目标更新请求的 Contact 头字段更新远程目标 URI。这允许 UA 提供新的联系地址，在对话期间应该改变其地址。

然而，不是目标更新请求的请求对对话的远程目标 URI 没有影响。

在第 8.1.1 节介绍了请求剩余部分的组成。

一旦构造了请求，使用与对话外请求相同的过程（第 8.1.2 节），计算出服务器的地址，并发送请求。

如果没有 Route 头字段，那么第 8.1.2 节的过程通常会将请求发送到第一个 Route 头字段值或 Request-URI 说明的地址。按照具体约束，他们允许将请求发送到可选的地址（如在路由集合中没有说明的默认的带外代理）。

Processing the Responses UAC 将接收到来自处理层的请求的响应。如果客户端事务返回超时，那么，它可以认为是 408 响应（请求超时）。

对话内发送的请求接收到 3xx 响应的 UAC 行为和对话外发送请求是一样的。在第 8.1.3 节中介绍了此行为。

注意，虽然，当 UAC 尝试可选的地址时，它仍然使用对话的路由集合来建立请求的 Route 头。

当 UAC 接收到返回给目标更新请求的 2xx 响应，如果存在的话，它必须将响应中 Contact 头字段的 URI 替换对话的远程目标 URI。

如果对话内请求的响应是 481（呼叫/事务不存在）或者 408（请求超时），那么，UA 应该结束对话。如果请求根本没有接收到响应，那么，UAC 也应该结束对话（客户端事务将通知 TU 超时）。

对于 INVITE 发起的对话，发送 BYE，结束对话。

12.2.2 UAS 行为

对话内的请求，和其它请求一样，是基本的。如果 UAS 接受具体的请求，完成与此请求相关的所有状态改变。如果 UAS 不能接受具体的请求，不完成与此请求相关的所有状态改变。

注意，有些请求，如 INVITEs，影响到几个状态。

UAS 将接收来自处理层的请求。如果请求的 To 头字段有标签，那么，UAS 核心计算出请求对应的对话标识符，并与现有的对话相比较。如果可以匹配，这将是 mid-dialog 请求。此种情况，UAS 先将应用与对话外的请求相同的处理规则，见第 8.2 节。

如果请求的 To 头字段有标签，但是对话标识符不能和现有的对话匹配，那么，UAS 可能会崩溃，并重新启动；或者它将会接收到不同（可能是错误的）的 UAS 的请求（UAS 可以构造 To 标签，以便于 USA 可以识别 UAS 可能恢复的标签）。另一种可能性是，入站请求仅仅是简单的路由错误。基于 To 标签，UAS 可以接受或拒绝此请求。接受合格的 To 标签提供了健壮性，这样，即使是崩溃，对话仍然可以继续。UA 希望支持此能力必须考虑一些问题，如，选择单调增加的 Cseq 序列号（即使有重启），重新构造路由集合以及接受溢出的 RTP 时间戳和序列号。

如果 UAS 因为不想重建对话而拒绝请求，那么，它必须响应请求，返回 481(呼叫/事务不存在)状态代码，并将状态代码发送给服务器事务。

在对话内可能接收到不改变对话状态的请求（如，OPTIONS 请求）。将像对话外接收的请求一样处理这些请求。

如果远程序列号为空，那么，它必须设置成请求的 Cseq 头字段的序列号值。如果远程序列号不为空，但是，请求的序列号值比远程序列号值低，那么，请求是无序的，必须拒绝请求，返回 500（服务器内部错误）响应。如果远程序列号不为空，请求的序列号值比远程

序列号值高，那么，请求是有序的。Cseq 序列号可能比远程序列号高很多。这不是错误的情况，UAS 应该准备接收和处理 Cseq 值比之前接收的请求 Cseq 值高很多的请求。然后，UAS 必须将远程序列号设置成请求的 CSeq 头字段值的序列号值。

如果代理挑战 UAC 生成的请求，那么，UAC 应该重新提交带有凭证的请求。重新提交的请求将有新的 Cseq 号。UAS 将不留意第一个请求，这样，它注意到了 Cseq 实数空间的间隔。这样一个间隔不代表任何错误情况。

当 UAS 接收到目标更新请求，如果存在的话，它必须用请求的 Contact 头字段中的 URI 替换对话远程目标 URI。

12.3 结束对话

与方法无关，如果对话外的请求生成非 2xx 的最终响应，那么，通过响应此请求的临时响应创建的早期对话将结束。结束已确定的对话的机制是由方法确定的。在本规范中，BYE 方法结束会话以及与其相关的对话。详情见第 15 章。

13 发起会话

13.1 概述

当用户代理客户端想要发起会话（如，音频、视频或游戏），它明确地表达了 INVITE 请求。INVITE 请求请求服务器建立会话。代理可以转发此请求，最后到达一个或者多个可以接受此请求的 UAS。这些 UAS 通常会询问用户是否接受此邀请。不久之后，这些 UAS 可以接受这些邀请（意味着将要建立会话），返回 2xx 响应。如果不能接受响应，与拒绝的原因有关，返回 3xx、4xx、5xx、6xx 响应。在发送最终响应之前，UAS 也发送临时响应（1xx），建议 UAC 继续与呼叫用户联系。

可能在接收到一个或多个临时响应后，UAC 将获取一个或多个 2xx 响应或非 2xx 的最终响应。因为时间延迟，它可能接收到 INVITE 的最终响应，INVITE 事务的可靠性机制与其他的请求（如，OPTIONS）不同。一旦它接收到最终响应，UAC 需要发送 ACK 给它所接收到的每个最终响应。发送 ACK 的过程与响应的类型有关。对于 300 和 699 之间的最终响应，在处理层遵循一套规则（见第 17 节）完成 ACK 处理。对于 2xx 响应，UAC 核心生成 ACK。

响应 INVITE 的 2xx 响应建立会话，同时，它也创建发布 INVITE 的 UA 和生成 2xx 响应的 UA 之间的对话。因此，当从多个不同的远程 UA（因为 INVITE 分发）接受到多个 2xx 响应，每个 2xx 建立不同的对话。

所有这些对话都是相同的呼叫的一部分。

本节提供了使用 INVITE 建立会话的细节。支持 INVITE 的 UA 也支持 ACK、CANCEL 和 BYE。

13.2 UAC 处理

13.2.1 创建初始的 INVITE

因为初始 INVITE 表示对话外的请求，遵循第 8.1.1 节的过程建立它。对于具体的 INVITE，其他的处理是必需的。

在 INVITE 中应该有 Allow 头字段(第 20.5 节)。它指出了在对话期间，UA 发送 INVITE，

在对话内可以调用的方法。例如，可以接收对话内[34]INFO 请求的 UA，应该包括段列出 INFO 方法的 Allow 头字。

在 INVITE 中应该有 Supported(第 20.37 节)。它列举了 UAC 理解的所有扩展。

在 INVITE 中可以有 Accept (第 20.1 节)。它指出了 UA 在接收到的响应和 INVITE 建立的对话内随后发送给它的请求中——可以接受的 Content-Types。对于指出支持不同会话描述格式来说，Accept 头字段尤其有用。

UAC 可以添加 Expires 头字段（第 20.19 节）来限制邀请的有效期。如果到了 Expires 头字段中指出的时间，并且没有接收到 INVITE 的最终响应，那么，UAC 核心应该按照第 9 章所述为 INVITE 生成 CANCEL 请求。

UAC 也可以在其他头字段中找到一些有用的头字段，添加到 INVITE 中，如，Subject (第 20.36 节)，Organization (第 20.25 节)和 User-Agent (第 20.41 节)头字段。它们都包含有与 INVITE 相关的信息。

UAC 可以选择将消息体添加 INVITE 中。第 8.1.1 节涉及到了怎样在其它需要说明消息体的位置建构头字段——Content-Type。

消息体包含有会话描述——其对应的 Content- Disposition 是“session”，有一些具体规则用于此消息体。在 UA 发送称为“呼叫”(它包含了提议的会话描述）的会话描述的地方，SIP 使用呼叫/应答模式。呼叫指出想要的通信方式（音频、视频、游戏)、这些方式的参数（如，编码类型）和接收应答器媒体的地址。另一个 UA 用其他会话描述来响应，称为“应答”，它指出接受哪种通信方式、使用于这些方式的参数和接受呼叫者的媒体类型。呼叫/应答交换是在对话环境中的，这样，如果 SIP INVITE 引起了多个对话，每一个就是单独的呼叫/应答交换。呼叫/应答模式定义了何时可以生成呼叫和应答的约束（例如，当一个正在处理之中时，不能生成新的呼叫)。这引入了在 SIP 消息中什么地方可以出现呼叫和应答的约束。在本规范中，呼叫和应答仅出现在 INVITE 请求和响应以及 ACK 中。呼叫和应答的使用有进一步的约束。对于初始 INVITE 的事务，规则是：

- 必须是在 INVITE 中发起呼叫，如果不是，它必须是在 UAS 返回给 UAC 的第一个可信的非错误消息中。在本规范中，即是最终的 2xx 响应。
- 如果是在 INVITE 中发起呼叫，那么，应答必须是在 UAS 返回给与此 INVITE 相关的 UAC 的第一个可信的非错误消息中。在本规范中，仅是 INVITE 的最终的 2xx 响应。相同的精确应答也可以放在应答之前的临时响应中。UAC 必须将它接收到的第一个会话描述看成是应答，并且必须忽略初始 INVITE 随后的响应中的会话描述。
- 如果是在 UAS 返回给与 UAC 的第一个可信的非错误消息中发起呼叫，那么，应答必须是在此消息的确认中（在本规范中，2xx 响应确认)。
- 在发送或者接收第一个呼叫的应答后，基于此方法指定的规则，UAC 可以生成请求的并发呼叫，但是此仅限于它接收到之前的呼叫的应答，并且没有给它未获取应答的 UAC 发送任何呼叫。
- 一旦 UAS 已经发送或者接收到发起呼叫的应答，它不能在初始 INVITE 的响应中生成并发呼叫。这意味着，直到完成了发起事务，基于本规范的 UAS 不能生成并发请求。

具体地说，上面的规则指明了符合本规范的 UA 的交换——呼叫是 INVITE，应答是在 2xx 中（使用相同的值，也可以是 1xx)；或者呼叫是 2xx，应答是在 ACK 中。支持 INVITE 的所有用户代理必须支持这两种交换。

所有的用户代理必须支持会话描述协议（SDP）（RFC 2327 [1]），把它当成是描述会话的方法，必须遵循在[12]中定义的过程来构造呼叫和应答的用法。

刚才介绍的呼叫-应答模式仅用于 Content-Disposition 头字段值为“session”的消息体。因此，INVITE 和 ACK 可能包括消息体消息（例如，INVITE 携带一张照片（Content-Disposition: render）和 ACK 会话描述（Content-Disposition: session））。

如果 Content-Disposition 头字段丢失，Content-Type application/sdp 消息体意味着部署“session”，而其他的内容类型意味着“render”。

一旦创建了 INVITE，UAC 遵循第 8 章定义的发送对话外请求的过程。这引入了构造客户端事务，最终它会发送请求，并传递响应给 UAC。

13.2.2 处理 INVITE 响应

一旦 INVITE 已经传送给 INVITE 客户端事务，那么，UAC 将等待 INVITE 的响应。如果 INVITE 客户端事务返回超时而不是响应，那么，与 8.1.3 介绍的一样，TU 认为接收到 408（请求超时）响应。

1xx Responses 在接收到一个或多个最终响应之前，可以有任意个临时响应（零个或多个）。INVITE 请求的临时响应可以创建“早期对话”。如果临时响应的 To 字段有标签，并且响应的对话 ID 和现有的对话不匹配，那么，使用第 12.1.2 节定义的过程建立对话。

如果在初始 INVITE 事务完成之前，UAC 需要发送请求给其对等物，那么，仅需要早期对话。只要对话是早期状态，那么，在临时响应中出现的头字段是可用的（例如，只有是在早期状态中，临时响应的 Allow 头字段包含在对话中可用的方法。）。

3xx Responses 3xx 响应可以包含一个或者多个 Contact 头字段——提供可以到达被呼叫者的新地址。与 3xx 响应的状态代码有关（见第 21.3 节），UAC 可以选择尝试新地址。

4xx、 5xx and 6xx Responses INVITE 可能接收到单个的非 2xx 最终响应。4xx、5xx 和 6xx 响应可能包含 Contact 头字段值——指出可以找到错误的其他信息的位置。必须忽略随后的最终响应（可能是在错误环境到达的）。

在接收到非 2xx 最终响应后，认为所有的早期对话结束。

在接收到非 2xx 最终响应后，UAC 核心认为 INVITE 事务完成。

INVITE 客户端事务处理为响应生成的 ACK（见第 17 章）。

2xx Responses 因为分发代理，多个 2xx 响应可能到达单个 INVITE 请求的 UAC。用 To 头字段的标签参数来区分每个响应，每个响应用独有的对话标识符来表示独有的对话。

如果 2xx 响应的对话标识符和现有对话的对话标识符匹配，那么，对话必须转变成“确认的”状态，必须使用第 12.2.1 节的过程，以 2xx 响应为基础计算对话的路由集合。此外，必须使用第 12.2.1 节的过程建立“确认的”状态的新对话。

注意，唯一可以再计算出的状态是路由集合。其他状态，如对话内发送的最高序列号（远程和本地）是不可以再计算的。仅再计算出路由集合是和 RFC2543 向后兼容的。RFC 2543 不要求在 1xx 反映 Record-Route 头字段，仅在 2xx 中反映。可是，因为可能在早期的对话中发送 mid-dialog 请求，如修改序列号，我们不能更新对话的整个状态。

UAC 核心必须为接收到的来自处理层的每个 2xx 生成 ACK 请求。用与对话外发送请求相同的方法（见第 12 章）建立 ACK 的头字段——除了 Cseq 和与认证相关的头字段外。CSeq 头字段的序列号必须与已经确认的 INVITE 相同，同时，Cseq 方法必须是 ACK。ACK 必须包含与 INVITE 相同的凭证。如果 2xx 包含呼叫（基于上面的规则），ACK 必须在其消息体中携

带应答。如果 2xx 响应的呼叫是不可接受的，那么，UAC 核心必须在 ACK 中生成有效的应答，然后，立即发送 BYE。

一旦建立了 ACK，[4]的过程用于确定目的地址、端口和传输。然而，请求传送给处理层直接传输，而不是客户端事务。这是因为，UAC 核心处理 ACK 的重发，而不是处理层。每当 2xx 最终响应的重发触发了 ACK 的到达，ACK 都必须传送给客户端事务。

UAC 核心认为，在接收了第一个 2xx 响应后，INVITE 事务完成 64*T1 秒。在这个点，将结束所有的为转变成建立对话的早期对话。一旦 UAC 核心认为 INVITE 事务已经完成，那么，将不再有新的 2xx 响应到达。

如果在确认了 INVITE 的 2xx 响应后，UAC 不想继续会话，那么，如第 15 章所述，UAC 必须发送 BYE 请求，结束会话。

13.3 UAS 处理

13.3.1 处理 INVITE

UAS 核心将从处理层接收 INVITE 请求。它首先执行第 8.2 节的请求处理过程，这同时适用于对话外和对话内的请求。

假定完成这些处理状态，没有生成响应，那么，UAS 核心执行其他的处理步骤：

1、如果请求是包含 Expires 头字段的 INVITE，那么，UAS 核心设置头字段值以秒表示的计时器。当计时器溢出，就认为请求到期。如果在 UAS 生成了最终响应之前请求到期，那么应该生成 487（请求结束）响应。

2、如果请求是 mid-dialog 请求，那么，先用第 12 章介绍与方法无关的处理。它也可以修改会话；第 14 章有详细的介绍。

3、如果请求的 To 头字段有标签，但是对话标识符不能与现有的对话匹配，那么，UAS 可能会崩溃，并重新启动；或者它将会接收到不同（可能是错误的）的 UAS 的请求。第 12.2.2 节提供了在这种环境下，获取健壮行为的原则。

处理这里的转发，认为 INVITE 是对话外的，因此可用来建立新的会话。

INVITE 可以包括会话描述，在这种情况下，用会话中的呼叫来表示 UAS。即使 INVITE 在对话外，用户都有可能已经在特定的会话中。可能出现多个其它的参与者邀请用户参与相同的多播会议。例如，SDP 在原始字段包括会话 ID 和版本号。如果用户已经是会话的一员，在会话描述中的参数也没有改变，那么，UAS 可能会接收 INVITE（即是不提示用户发送 2xx 响应）。

如果没有包括会话描述，那么，将要求 UAS 参与会话，同时，UAC 要求 UAS 提供会话的呼叫。UAS 必须在其返回给 UAC 的第一个非错误的可信消息中提供呼叫。在本规范中，即是 INVITE 的 2xx 响应。

UAS 可以表示进行、接受、重定向和拒绝此邀请。在所有的这些情况中，它是用第 8.2.6 节介绍的过程，简要地说明响应。

Progress 如果 UAS 不能立即应答请求，它可以选择表示 UAC 进行的类型（如，表示电话正在响铃）。用 101 到 199 之间的临时响应来完成。遵循第 12.1.1 节和第 8.2.6 节的过程，这些临时响应建立早期的对话。当它愿意时，UAS 可以发送很多临时响应。每个响应必须指出相同的对话 ID。然而，这些响应不是可信地传输的。

如果要扩展应答 INVITE 的时间，那么，为了防止代理取消事务，它需要请求一个“扩

展”。当事务的响应之间有 3 分钟的间隙时，代理可以选择取消事务。为了防止取消，UAS 必须在每分钟都发送非 100 临时响应，处理可能丢失的临时响应。

当用户没挂断，或者与 PSTN 系统交互工作时（PSTN 系统允许通信在无应答呼叫的情况下发生），INVITE 事务继续扩展持续时间。后者普遍用在交互语音响应（IVR）系统中。

The INVITE is Redirected 如果 UAS 决定重定向呼叫，那么，发送 3xx 响应。300（多种选择）、301（永久清除）、302（暂时清除）响应应该包含 Contact 头字段——包含一个或多个要尝试新地址的 URI。此响应传送给 INVITE 服务器事务，它将处理转发。

The INVITE is Rejected 当被呼叫者在当前不愿意或者不能在此终端系统建立其他呼叫时，经常出现这种情况。在这种情况中，应该返回 486（这儿正忙）响应。如果 UAS 知道没有其他的终端系统可以接受此呼叫，它将发送 600（各处都忙）响应。但是，通常，UAS 不可能知道这个，因此，不经常使用此响应。此响应传送给 INVITE 服务器事务，它将处理转发。

UAS 拒绝包含在 INVITE 中的呼叫，应该返回 488（这里不能接受）响应。

这样一个响应应该包括 Warning 头字段值，解释为什么拒绝此呼叫。

The INVITE is Accepted UAS 核心生成 2xx 响应。遵循第 12.1.1 节和第 8.2.6 节的过程，此响应建立对话。

INVITE 的 2xx 响应应该包括 Allow 头字段和 Supported 头字段，也可以包括 Accept 头字段。包括这些头字段，允许 UAC 不需要探索，就确定在呼叫持续期间 UAS 支持的特性和扩展。

如果 INVITE 请求包含呼叫,UAS 还没有发送应答,那么,2xx 必须包含应答。如果 INVITE 请求不包含呼叫，UAS 还没有发送呼叫，那么，2xx 必须包含呼叫。

一旦建立此响应，它传送给 INVITE 服务器事务。然而，注意，只要它接收到此最终响应，并将它发送给传输，那么，将破坏 INVITE 服务器事务。因此，在 ACK 到达之前，有必要定期直接发送响应给传输。2xx 响应在一个时间间隔内传送给传输，该时间间隔以 T1 秒开始，并且每次重发时间间隔加倍，直到到达 T2 秒（第 17 章定义了 T1 和 T2）。当接收到响应的 ACK 请求，停止重发响应。这与用于发送响应的传输协议无关。

因为 2xx 是端对端重发的，在 UAS 和 UAC 之间的逐跳可能是 UDP。为了确保在这些逐跳之间可信的传输，即使在 UAS 的传输是可靠的，也要定期的重发响应。

如果在 64*T1 秒内没有接收 ACK，服务器重发 2xx 响应，确定对话，并将结束会话。如第 15 章所介绍的，通过 BYE 来结束会话。

14 修改现有的会话

成功的 INVITE 请求（见第 13 章）同时建立两个用户代理之间的对话和呼叫-应答模式的会话。第 12 章说明了怎样使用目标更新请求修改现有的对话（如，修改对话的远程目标 URI）。本章介绍了怎样修改当前会话。此修改包括，修改地址或端口、添加媒体流、删除媒体流，等等。通过在建立会话的相同对话中发送新的 INVITE 来完成。在现有对话中发送 INVITE 是 re-INVITE。

注意，单个的 re-INVITE 可以同时修改对话和会话的参数。

呼叫者和被呼叫者都可以修改现有的会话。

UA 检测媒体错误的行为是本地策略的事情。但是，当网络拥塞时，为了避免大量的网

络流量，不推荐自动生成 re-INVITE 和 BYE。在任何请情况中，如果要自动发送这些消息，应该在随机的间隔后发送这些消息。

注意，上面的章节提及了自动生成 re-INVITE 和 BYE。如果用户因为媒体错误而挂断，那么，UA 应该和平常一样发送 BYE 请求。

14.1 UAC 行为

在 INVITEs 中用于会话描述的呼叫-应答模式(第 12.3.1 节)也同样适用于 re-INVITE。结果，例如，想要添加媒体流的 UAC，将创建包含此媒体流的新呼叫，并在 INVITE 请求中将其发送给其对等物。重点要指出的是，要发送会话的全部描述，而不仅仅是改变。这支持在不同元素中的无状态处理，也支持自动恢复能力。当然，UAC 可以发送无状态描述的 re-INVITE，在这种情况中，re-INVITE 的第一个可信非错误的响应将包含呼叫（在本规范中，即是 2xx 响应）。

如果会话描述格式有版本号，那么，呼叫应该指出已经改变了会话描述的版本。

按照第 12 章介绍的，遵循与在现有对话内的规则请求相同的方法，设置 re-INVITE 的 To、From、Call-ID、Cseq 和 Request-URI。

因为，UAS 一般不会警告用户接受 re-INVITE，UAC 可以选择不为 re-INVITE 添加 Alert-Info 头字段和 Content-Disposition“Alert”消息体。

不像 INVITE 可以分发，re-INVITE 不会分发，因此，它仅能产生单一的最终响应。re-INVITE 不会分发的原因是，Request-URI 确定了目标是和它建立对话的 UA，而不是确定用户的记录地址。

注意，当正在任一方向进行其它的 INVITE 事务时，UAC 不能在对话内发起新的 INVITE 事务。

1、如果有正在进行的 INVITE 客户端事务，那么 TU 必须等到事务完成或结束状态，才可以发起新的 INVITE。

2、如果有正在进行的 INVITE 服务器事务，那么 TU 必须等到事务完成或结束状态，才可以发起新的 INVITE。

然而，当 INVITE 事务正在处理时，UA 可以发起普通的事务。当普通的事务正在处理时，UA 可以发起 INVITE 事务。

如果 UA 接收到 re-INVITE 的非 2xx 最终响应，那么，将像没有发布 re-INVITE 一样，不改变会话参数。注意，如 12.2.1 所述，如果非 200 最终响应是 481（呼叫/事务不存在）或者 408（请求超时），或者请求根本没有接收到 re-INVITE 的响应（即是 INVITE 客户端事务返回超时），那么，UA 将结束对话。

如果 UAC 接收到 re-INVITE 的 491 响应，那么，它应该启动有 T 值的计时器，T 选择如下：

1、如果对话 ID 有自己的 Call-ID（指它生成的值），那么，T 是以 10ms 为单位、在 2.1 到 4 秒内随机选择的值。

2、如果对话 ID 没有自己的 Call-ID，那么，T 是以 10ms 为单位、在 0 到 2 秒内随机选择的值。

当计时器溢出，如果它仍然要修改会话状态，那么，UAC 应该再次尝试 re-INVITE。例如，如果呼叫已经用 BYE 挂断，那么将不能出现 re-INVITE。

发送 re-INVITE 和为 re-INVITE 的 2xx 响应生成 ACK 的规则和初始 INVITE 的规则相同（第 13.2.1 节）。

14.2 UAS 行为

第 13.3.1 节介绍了从入站的初始 INVITE 中区分入站的 re-INVITE 以及处理现有对话中的 re-INVITE 的过程。

UAS 在发送第一个 INVITE 的最终响应之前，接收到相同会话的、低 Cseq 序列号的第二个 INVITE，UAS 必须对第二个 INVITE 返回 500 响应（服务器内部错误），并且必须包括在 0 到 10 秒之间随机选择值的 Retry-After 头字段。

UAS 正在处理会话已经发送的 INVITE 时，接收到 INVITE，它必须给接收到的 INVITE 返回 491 响应（请求待处理）。

如果 UA 接收到现有对话的 re-INVITE，它必须检查会话描述的版本标识符，如果没有版本标识符，认为会话描述的内容已经改变。

如果会话描述已经改变，那么，可能在要求用户确认后，UAS 必须调整会话参数。

会话描述的版本可以用来调节新到达会议的能力——添加删除媒体、从单点到多播会议。

如果不能接受新会话描述，UAS 可以通过对 re-INVITE 返回 488 响应（在此不能接受）拒绝它。此响应应该包含 Warning 头字段。

如果 UAS 生成 2xx 响应，但不能接受 ACK，那么，它应该生成 BYE 结束会话。

因为，UAC 一般不将此信息返回给用户，UAS 可以选择不为 re-INVITE 生成 180（响铃）响应。因为此原因，在响应 re-INVITE 时，UAS 可以选择不使用 Alert-Info 头字段和 Content-Disposition“Alert”消息体。

在 2xx 中提供呼叫（因为 INVITE 不包含呼叫）的 UAS 应该，和[12]中介绍的 SDP 情况一样，服从发送呼叫更新现有会话的约束，建立呼叫，就像 UAS 正在做出新的呼叫。特别的是，此方法应该包括 UA 愿意支持的媒体格式和媒体类型。UAS 必须确保会话描述与 UAS 先前的会话描述在对等物需要支持的媒体格式、传输和其它参数等方面重叠。这就避免了对等物拒绝会话描述。如果 UAC 不能接受它，UAC 应该生成带有有效会话描述的响应，然后发送 BYE，结束会话。

15 结束会话

本章介绍了结束 SIP 建立的会话的过程。会话的状态和对话的状态是紧密相关的。当用 INVITE 发起会话时，不同 UAS 产生的 1xx 和 2xx 响应都创建对话，如果此响应完成呼叫/应答交换，那么，它也创建会话。结果，每个会话都与创建它的对话相关联。如果初始的 INVITE 生成非 2xx 最终响应，那么，它将结束通过此请求的响应创建的所有会话（如果有）和对话（如果有）。由于完成了事务，非 2xx 最终响应也防止了 INVITE 创建更深层次的会话作为 INVITE 的结果。BYE 请求用来结束具体的会话和尚未建立的会话。在这种情况中，具体的会话和对话是一个 UA 与对话中另一个对等 UA 建立的。当在对话中接收到 BYE，应该结束和此对话相关的所有会话。UA 不能在对话外发送 BYE。呼叫者 UA 可以在确认的和早期的对话中发送 BYE；被呼叫者 UA 可以在确认的对话中发送 BYE，但是不能在早期的对话中发送 BYE。然而，在被呼叫者 UA 接收到其 2xx 响应的 ACK 和服务器事务超时之前，它不能在确认

的对话中发送 BYE。如果在此对话相关的应用层状态中没有定义 SIP 扩展，那么，BYE 也可以结束对话。

对话和会话中 INVITE 的非 2xx 最终响应的效果是使用 CANCEL attractive。CANCEL 尝试将非 2xx 最终响应强制发送给 INVITE（特别是，487）。因此，如果 UAC 希望完全放弃其呼叫尝试，它发送 CANCEL。如果 INVITE 导致了 INVITE 的 2xx 最终响应，这意味着正在处理 CANCEL 时，UAS 接受了此邀请。UAC 可以继续 2xx 响应建立的会话，也可以使用 BYE 结束会话。

在 SIP 中，没有很好的定义“挂断”的概念。挂断是很具体的细节，但是也是很通用的用户接口。一般，当用户挂断，它表示想要结束建立会话的尝试和结束已经创建的会话。对于呼叫者 UA，如果初始的 INVITE 没有创建最终响应，那么挂断意味着 CANCEL 请求；对于在最终响应后所有确认的对话，挂断意味着 BYE。对于被呼叫者 UA，一般来说，挂断意味着 BYE。

可能是，当用户拿起电话，生成 2xx，在接收到 ACK 后，挂断可能会导致 BYE。这并不是指，用户在接收到 ACK 前不能挂断电话，仅仅是指，为了正确的清除，用户电话的软件需要在一小段时间维持状态。如果特定的 UI 允许用户在其应答之前拒绝呼叫，403（禁止）是一个很好的表示方式。按上面的规则，可以发送 BYE。

15.1 使用 BYE 请求结束会话

15.1.1 UAC 行为

如第 12 章所介绍的，构造 BYE 请求和对话内其他请求一样。

一旦构造了 BYE，UAC 核心创建新的非 INVITE 客户端事务，并将其传送给 BYE 请求。BYE 请求一旦发送给客户端事务，UAC 必须认为会话结束了（并因此停止发送和侦听媒体）。如果 BYE 响应是 481（呼叫/事务不存在）或者 408（请求超时），或者请求根本没有接收到 BYE 的响应（即是 INVITE 客户端事务返回超时），那么，UAC 必须认为会话和对话结束了。

15.1.2 UAS 行为

UAS 首先遵循第 8.2 节介绍的通用 UAS 处理过程来处理 BYE 请求。

UAS 核心接收到 BYE 请求，检查它是否和现有对话匹配。如果 BYE 不能和现有对话匹配，那么，UAS 核心必须生成 481（呼叫/事务不存在）响应，并将其发送给服务器事务。

此规则意味着，将拒绝 UAC 发送的无标签 BYE。这是对 RFC2543 的修改——它允许无标签的 BYE。

UAS 核心接收现有对话的 BYE 请求，必须遵循第 12.2.2 节的过程处理请求。一旦完成，UAS 应该结束会话（并因此停止发送和侦听媒体）。它唯一不可以选择的情况是多播会话，在多播会话中，即使会话的其他参与者已经结束了其包含的会话，都有可能参与（participation）。不管它是否结束参与会话，UAS 核心必须对 BYE 生成 2xx 响应，并将其发送到服务器事务传输。

UAS 必须响应任何此对话中接收的挂起的请求。推荐对于这些挂起的请求生成 487（请求结束）响应。

16 代理行为

16.1 概述

SIP代理是将SIP请求路由到用户代理服务器和将SIP响应路由到用户代理客户端的元素。到达UAS的请求可以穿透其间的多个代理。每个代理都可以作出路由决定，并在将请求转发到下一个元素之前对其修改。响应将通过代理路由，该代理与逆序请求穿透的代理相同。

代理作为 SIP 元素的逻辑角色。当请求到达时，该元素在作为代理之前先确定是否需要响应其上的请求。例如，在充当代理之前，该请求可能难以理解或者该元素需要客户端的凭证。该元素可能用适当的错误代码作出响应。当直接响应请求时，该元素则扮演UAS的角色，并且其行为必须如第8.2节所述。

对于每个新的请求，代理可以以有状态或者无状态模式操作。当采用无状态模式时，代理仅充当一种简单转发元素。它将每个请求下游转发给某单个元素，该元素是基于请求由目标和路由决定确定的。它简单地转发它接收的每个上流响应。一旦请求被转发，无状态代理将丢弃该请求相关的所有状态信息。

有状态代理将保留每个入站请求的信息（特别是事务状态），并且它所发送的任何请求都作为入站请求处理结果。有状态代理使用入站请求信息来影响与该请求相关的下一步消息处理。有状态代理可以选择“分发”请求，并将它路由到多个目的地。转发给多个位置的请求都必须有状态处理。

在某些情况中，代理可能在未成为有状态事务时就使用有状态传输协议（如TCP）转发请求。例如，代理可以将请求从一个TCP连接转发到另一个无状态事务，只要它在消息里放有足够多的信息以将该响应向下转发给该请求所到达的相同连接上。在不同传输类型之间所转发的请求必须有状态的转发事务，这里代理的TU必须主动地确保在每一个传输上都是可靠传输。

在请求处理期间，只要有状态代理不作阻止其回到初始无状态的事情（例如，分发或生成一个100响应），它就可以转换为无状态操作模式。当完成这样的转换时，丢弃了所有状态。代理不应发起CANCEL请求。

无论请求是有状态还是无状态，相关的许多处理都是相同的。后面几个章节主要从有状态代理的角度来描述。而最后一章则介绍了无状态代理的不同行为。

16.2 有状态代理

当处于有状态模式时，代理则完全作为一个SIP事务处理引擎。在此按照第17章所定义的服务器和客户端事务模拟其行为。有状态代理通过高级代理处理组件（如图3，作为代理核心）将一个服务器事务与多个客户端事务联系在一起。入站请求由服务器事务来处理。服务器事务处理后的请求传送到代理核心。代理核心确定将该请求路由到何处，并选择一个或者多个下一跳位置。每个下一跳位置的出站请求都由其自身相关的客户端事务来处理。代理核心收集来自客户端事务的响应，然后使用客户端事务将响应发送到服务器事务。

有状态代理为每个所接收的新请求创建新的服务器事务。随后按第17章介绍的服务器事务，处理请求的转发。在发送及时的临时请求给第8.2.6节介绍的服务器事务（如，100（尝试））时，代理核心必须作为UAS。因此，有状态代理不应该对非INVITE请求生成100（尝试）响应。

这是代理行为的模型，而不是软件。可以自由地采用任何方法来复制该模型定义的外部行为。

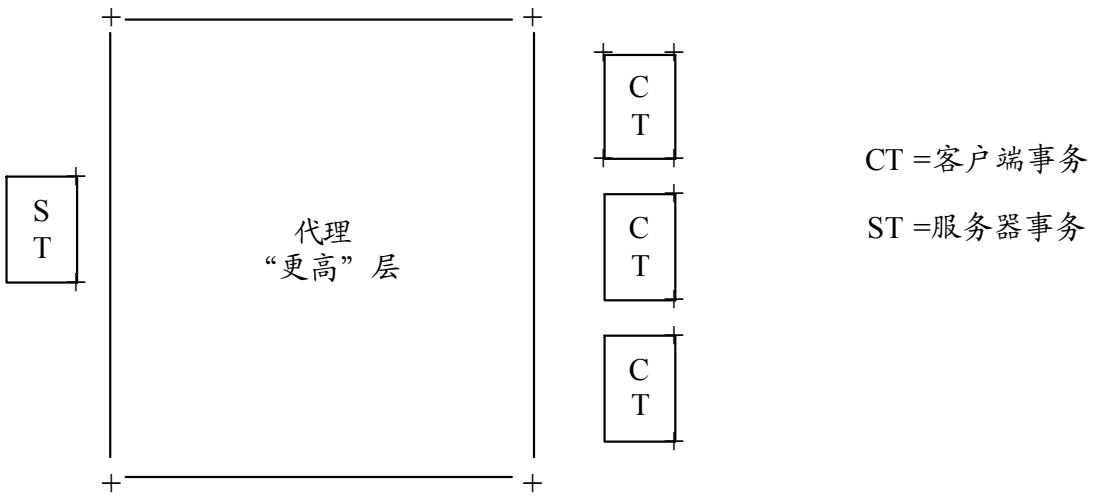

图 3：有状态模型

对于所有新的请求，包括未知方法、作为代理的元素，请求必须：

1. 确认请求(第 16.3 节)
2. 预处理路由信息(第 16.4 节)
3. 确定请求目标(第 16.5 节)
4. 向每个目标转发请求(第 16.6 节)
5. 处理所有响应(第 16.7 节)

16.3 请求确认

在元素代理请求之前，它必须确认消息的有效性。有效消息必须经过以下检查：

1. 合理语法
2. URI 模式
3. Max-Forwards
4. (可选)循环检测
5. Proxy-Require
6. Proxy-Authorization

如果其中任何检查失败，该元素必须作为用户代理服务器（参见第 8.2 节），然后响应错误代码。

注意代理不需要检测已合并的请求，它不应将已合并的请求作为错误的条件。接收请求的端点将解析合并，如第 8.2.2 节所述。

1、合理的语法检查

请求必须足够规格化，以便于与服务器事务一起处理。在剩余的这些请求确认步骤中

或者请求转发章节所涉及的组件必须是规格化组件。对于其它组件，则对规格化的要求不是很严格，当转发消息时，可以忽略此组件或者该组件保持不变。举个例子来说，由于 Date 头字段的格式不好，元素将不能接受请求。同样的，代理不应该在转发请求之前将格式不好的 Date 头字段删除。

本协议可用来扩展。新扩展可以随时定义一些新的方法和头字段。这样元素不能因为请求中包含它所不知道的方法和头字段而拒绝代理请求。

2、URI 模式检查

如果 Request-URI 的 URI 模式不为代理所理解，代理应拒绝该请求，并返回 416（不支持 URI 模式）响应。

3、Max-Forwards 检查

Max-Forwards 头字段（第 20.22 节）用于限制 SIP 请求可以穿透的元素数。

如果请求不包含 Max-Forwards 头字段，Max-Forwards 检查通过。

如果该请求包含 Max-Forwards 头字段，该字段是比零大的字段值，则 Max-Forwards 检查也通过。

如果该请求包含 Max-Forwards 头字段，该字段值为零，元素不需转发请求。如果该请求用于 OPTION，该元素可以作为最终接受者，并作出响应，如第 11 章所示。此外，元素必须返回 483（太多跳跃点）响应代码。

4、可选循环检测检查

在转发请求前，元素可以对转发循环作检查。如果请求包含了 Via 头字段，所发送的值等于代理预先放置在请求中的值，该请求先由该元素转发。请求通过该元素循环或者合理的螺旋。为了确定了请求是否循环，该元素可以完成第 16.6 节中第八步所描述的分支参数计算，并将它与 Via 头字段中所接收的参数进行比较。如果参数匹配，请求循环。如果它们不相同，则请求正螺旋，处理继续。如果探测出发生循环，则该元素返回 482（循环探测）响应。

5、Proxy-Require 检查

对协议作进一步扩展，则可以引入代理特殊处理的功能。端点在请求中加入使用这些特性的 Proxy-Require（代理需求）头字段，告知代理除非理解该特性，否则不处理该请求。

如果该请求包含 Proxy-Require 头字段（第 20.29 节），它带有一个或者多个该元素所不理解的可选标签，元素必须返回 420（错误扩展）响应。该响应必须包括 Unsupported（第 20.40 节）头字段，该字段列出了该元素不理解的可选标签。

6、Proxy-Authorization 代理认证检查

如果在转发请求之间，元素要求凭证，必须按照第 22.3 节所描述的要求对该请求检查。在第 22.3 节中也定义了如果检查失败，元素必须执行什么操作。

16.4 路由信息处理

代理必须检查请求的 Request-URI。如果请求的 Request-URI 包含了该代理预先放置在 Record-Route 头字段（参见 16.6 中第 4 条）值，代理必须使用 Route 头字段中的最后一个值代替请求中 Request-URI 的值，并且将该值从 Route 头字段中去掉。一旦代理接收到已修改的请求，它将继续处理。

这种情况只有在元素向作为严格路由器的代理（可以是一个端点）发送请求时发生。对所接收的请求重写必须具有这些元素的向后兼容性。它也允许遵循本规范的元素，通过严格路由代理保留 Request-URI（见 12.2.1）。

本规范不强制代理保持状态，以便于它检查先前放置在 Record-Route 头字段中的 URI。反过来说，代理只需要在这些 URI 中放置足够多的信息，它就可以在随后出现时可以识别其值。

如果 Request-URI 包含 maddr 参数，那么代理必须检查所配置代理负责的值是否在地址集或者域中。如果 Request-URI 包含带值的 maddr 参数，该值由代理负责，并且使用 Request-URI 中所指示（明确指示或者默认）的端口和传输来接收请求，那么，代理必须去除 maddr 参数和任何非默认端口或者传输参数，并且继续处理，就像这些值从未在请求中出现一样。

如果所接收请求带有与代理匹配的 maddr，但是该 maddr 参数与 URI 中所指示的端口或者传输不同。这样就需要使用所指示的端口和传输将请求转发给代理。

如果 Route 头字段中的第一个值指明这个代理，该代理必须将此值从请求中去掉。

16.5 确定请求目标

接下来，代理计算出请求的目标。目标集合可以由请求的内容预先确定，也可以从抽象定位服务中获得。集合中的每个目标都作为 URI。

如果请求的 Request-URI 包含 maddr 参数，Request-URI 必须作为唯一的目标 URI 放在目标集合中，并且，代理必须继续请求转发（第 16.6 节）。

如果 Request-URI 的域表示此域不是该元素负责的域，那么，Request-URI 必须作为唯一的目标 URI 放在目标集合中，同时，此元素必须继续请求转发的任务（第 16.6 节）。

在很多情况中，代理可能接收到其不负责的域的请求。防火墙代理处理出站呼叫（HTTP 代理处理出站请求的方式）是可能发生此情况的一个实例。

如果请求的目标集合没有如上所述预先确定，那么，它意味着元素是对 Request-URI 的域负责任的，同时，元素可以使用它想要的任何机制，确定发送请求的地址。这些机制可以作为访问抽象定位服务的模型。这可能由以下几个部分组成——获取 SIP 注册服务器创建的定位服务信息、读取数据库、查询状态服务器、使用其他协议和在 Request-URI 中简单地执行算法替代。当访问注册服务器构造的定位服务时，在将其用作索引之前，首先要按照第 10.3 节所述的方式将 Request-URI 规范化。这些机制的输出用作构造目标集合。

如果 Request-URI 没有为代理提供足够的信息来确定目标集合，那么，它应该返回 485（模糊）响应。此响应应该包含 Contact 头字段——包含已作尝试的新地址的 URI。如，对 sip:John.Smith@company.com 的 INVITE 可能有些含糊不清，其定位服务列出了多个 John Smiths。详情参见第 21.4.23 节。

在请求中或关于请求或元素当前环境的任何信息都可以用作构造目标集合。例如，依赖于头字段和消息体的内容和状态、请求间隔的延迟、请求到达的接口、之前请求的错误、甚至是元素的当前利用率，可以构造不同的目标集合。

当通过这些服务对潜在的目标定义时，将其 URI 添加到目标集合中。目标仅能在目标集合中放置一次。如果在目标集合中已经有目标 URI（以 URI 类型相同的定义为基础），那么，不能再添加目标 URI。

如果原始请求的 Request-URI 没有指出代理负责的资源，那么，代理不能在目标集合中添加其它的目标。

只有代理对此 URI 负责时，代理才可以在转发期间修改 Request-URI 的 URI。如果代理不对此 URI 负责，那么，它将不能使用下面介绍的 3xx 和 416 响应。

如果原始请求的 Request-URI 指出了代理负责的资源，那么，在开始请求转发后，代理可以在目标集合中添加目标。它可以使用在处理过程中获取的任何信息来确定新目标。例如，代理可以选择将重定向响应（3xx）中获取的联系信息合并到目标集合中。如果在建立目标集合的同时，代理使用一个动态信息资源（如，它查询 SIP 注册服务器），那么，在处理请求的过程中，它应对资源监控。一旦新的地址有效，应该将其添加到目标集合中。如上所述，任何给定的 URI 不能多次添加到目标集合中。

仅允许 URI 在目标集合中添加一次，可以减少不必要的网络流量，从重定向请求中获取的合并联系信息的情况，防止了无穷的网络递归。

例如，当目标 URI 和入站的请求 URI 相同时，通常的定位服务是“无操作”。为了进一步处理，请求发送到指定的下一跳代理。在第 16.6 节第 6 条所描述的请求转发期间，以 SIP 和 SIPS URI 表示的下一跳识别符作为最上面的 Route 头字段值插入到请求中。

如果 Request-URI 指出此代理上的资源不存在，那么，代理必须返回 404（没找到）响应。

如果在应用以上所有之后，目标集合仍然为空，代理必须返回 480（暂时不可用）错误响应。

16.6 转发请求

一旦目标集非空时，代理便开始转发请求。有状态代理可以以任何顺序处理这个目标集合。它可以串行处理多个目标，即允许客户端事务在下一个事务开始前完成。它也可以采用每个目标并行处理的方式启动客户端事务。它还可以将目标集合任意分组，对这些组串行处理，或者对每个组中的目标并行处理。

常用的排序机制是使用目标的 q 值参数，这些参数可从 Contact 头字段（参见第 20.10 节）中获取。目标的处理顺序是从最高 q 值到最低 q 值。具有相同 q 值的目标可并行处理。

当接收响应时，有状态代理必须具有维护对象集合的机制，然后将对每个转发请求的响应与原始请求相关联。对于该模型而言，这种机制就是由代理层在转发第一个请求之前创建的“响应上下文”。

对于每个目标，代理都遵循以下步骤发送请求：

1、 复制所接收的请求

2、 更新 Request-URI

3、 更新 Max-Forwards 头字段

4、 添加 Record-Route 头字段值（可选的）

5、 添加其它头字段（可选）

6、 后处理路由信息

7、 确定下一跳的地址、端口和传输协议

8、 添加一个 Via 头字段值

9、 如果需要，添加一个 Content-Length 头字段

10、 转发新请求

11、 设置计时器 C

每个步骤详述如下：

1、 复制请求

代理首先将接收到的请求复制。这个副本最初必须包含所接收到的请求的所有头字段。在一下描述中没有明确说明的字段不可以去掉。副本应与所接收的请求保持相同的头字段顺序。代理不能用通用字段名（请参见第 7.3.1 节）对字段值重新排序。代理也不能添加、修改或删除消息体。

在实际执行中有时并不需要完成复制请求;但是主要前提条件是每个下一跳的处理都以相同的请求开始。

2、 Request-URI

必须用目标的 URI 代替位于副本起始行的 Request-URI。如果这个目标的 URI 含有 Request-URI 中不允许的任何参数，那么必须删除它们。

这就是代理角色的本质。代理会通过这种机制将请求路由到目的地。

在某些情况下，接收到的 Request-URI 未经修改便放到了目标集合中。对于该目标而言，无需执行以上替换操作。

3、 Max-Forwards

如果副本包含 Max-Forwards 头字段，那么代理必须将其值减 1。

如果副本不含 Max-Forwards 头字段，那么代理必须添加一个 Max-Forwards 字段值，其值为 70。

现有的某些 UA 在请求中没有提供 Max-Forwards 字段。

4、 Record-Route

如果这个代理希望保留在该请求所创建的对话中后续请求的路径上(假定这个请求会创建一个对话),那么即使已经存在 Route 头字段,它也必须在副本中任何已有的 Record-Route 头字段值之前插入一个 Record-Route 头字段值。

建立对话的请求可以包含一个预先加载的 Route 头字段。

如果这个请求已经是对话的一部分，那么，如果代理希望保留在对话中后续请求的路径上，它就应当插入一个 Record-Route 头字段值。在通常端点操作（如第 12 章所述）的情况下，这些 Record-Route 头字段值不会对端点使用的路由集合产生任何影响。

如果代理选择不向已经是对话的一部分的请求中添加 Record-Route 头字段值，那么代理将会保留在该路径上。但是当失败端点重新构建对话时，将会从这个路径上去掉这个代理。

代理会向任何请求添加 Record-Route 头字段值。如果请求不启动对话，那么端点将忽略这个值。欲了解端点如何使用 Record-Route 头字段值构建 Route 头字段的详情，请参阅第 12 章。

请求路径中的每个代理都会单独选择是否添加 Record-Route 头字段值——请求中 Record-Route 头字段的状态并不迫使该代理添加一个值。

放入 Record-Route 头字段值中的 URI 必须是 SIP 或 SIPS URI。这个 URI 必须包含一个 lr 参数（请参阅第 19.1.1 节）。这个 URI 可以与请求转发到的每个目的地不同。除非代理

知道（如在私有网络中）下一个将在后续请求路径中的下游元素支持此传输协议，URI 不应包含传输参数。

这个代理提供的 URI 将被其它元素用来作路由决策。一般说来，这个代理无法知道其它元素的性能，因此它必须将其本身限制为 SIP 执行的强制元素：SIP URI 和 TCP/UDP 传输。

当对放置在 Record-Route 头字段的 URI 应用服务器定位的第四步时，这个 URI 必须对插入此 URI 的元素解析，这样后续请求就可以访问相同的 SIP 元素。如果 Request-URI 包含 SIPS URI，或顶端 Route 头字段值（在步骤 6 的后处理后）包含 SIPS URI，那么放置在 Record-Route 头字段的 URI 必须是 SIPS URI。此外，如果没有在 TLS 之上接收到这个请求，代理就必须插入一个 Record-Route 头字段。与之类似的情况：代理在 TLS 上接收请求，但是生成的请求没有在 Request-URI 或顶端 Route 头字段值中（在步骤 6 的后处理后）的 SIPS URI，那么代理必须插入一个非 SIPS URI 的 Record-Route 头字段。

在整个对话期间，位于安全边界的代理必须保留此边界。

当放置在 Record-Route 头字段中的 URI 以响应的形式回传时，如果它需要重写，那么这个 URI 必须与众不同，以便在此时可以定位。（这个请求可通过此代理螺旋，导致了添加多个 Record-Route 头字段值）。第 16.7 节第 8 条推荐了一种使 URI 与众不同的机制。

代理可在 Record-Route 头字段值中包含参数。这些参数值将在对该请求的某些响应中得到回应，这些响应如：对 INVITE 的 200（OK）响应。这样的参数对于在消息中而不是在代理中保持状态会很有用。

如果某代理需要存在于任何类型的对话（如一用户要跨越防火墙）路径中，那么它应当用一个它不理解具有对话语义的方法向每个请求添加一个 Record-Route 头字段值。

代理放置在 Record-Route 头字段中的 URI，仅仅在出现 URI 的事务所创建的任何对话的生命周期内有效。例如，对话有状态代理可在对话终止后可以拒绝接受在 Record-Route 中有该值的后续请求。当然，非对话有状态协议没有对话何时终止的概念，但它们可以对值中信息编码，以将其与后续请求的对话标识符相比较，然后会拒绝与那些信息不匹配的请求。端点不能在提供它的对话外使用从 Record-Route 头字段获得的 URI。欲了解更多有关 Record-Route 头字段端点的使用问题，请参阅第 12 章。

代理需要观察对话中所有消息的某些服务可能需要 Record-Routing。但是，Record-Routing 会减慢处理过程，并降低可扩展性，使得代理只在特定服务需要的情况下进行 Record-Route。

Record-Route 过程旨在为启动对话的任何 SIP 请求工作。INVITE 是本规范中唯一属于这类的请求，但是这个协议的扩展可以定义其它请求。

5、 添加其他头字段

在这一点上，代理可以向副本添加任何其它适当的头字段。

6、 对路由信息进行后处理

代理可以设置本地策略，要求请求在传送到目的地之前访问特定代理集合。代理必须确保所有这些代理都是松散路由。一般说来，如果这些代理位于同一管理域中，这点可能比较容易明白。该代理集合由 URI 集合（每个 URI 都包含 lr 参数）表示。该代理集合必须放入副本中的 Route 头字段中，并且位于任何现有的值之前，如果这些值存在的话。

如果代理可以设置本地策略，要求请求访问一个特定代理，那么将 Route 值放入 Route 头字段中的可选办法是跳过下面描述的第 10 步的转发步骤，而仅向那个特定代理的地址、端口以及传输协议确定的目的地发送请求。如果这个请求有 Route 头字段，那么除非知道下

一跳代理是松散路由，否则不能使用这种可选方法。此外，虽然这个方法也可用，但对于操作的健壮性、灵活性、通用性和连贯性来说，Route 插入机制是更可取的。而且，如果 Request-URI 包含 SIPS URI，那么必须使用 TLS 与该代理进行通信。

如果副本包含一个 Route 头字段，那么代理对其第一个值中 URI 检查。如果此 URI 不含有 lr 参数，这个代理必须对该副本做如下修改：

- 这个代理必须将 Request-URI 作为最后一个值放入 Route 头字段中。
- 然后该代理必须将第一个 Route 头字段值放在 Request-URI 中，并将第一个 Route 头字段值从 Route 头字段中删除。

将 Request-URI 附加在 Route 头字段中，是通过严格路由元素传送 Request-URI 中的信息的机制的一部分。将第一个 Route 头字段值加到 Request-URI 中会将消息格式化，严格路由元素期望以这种格式来接收这条消息（其自身的 URI 在 Request-URI 中，下一个要访问的位置在第一个 Route 头字段值中）。

7、 确定下一跳的地址、端口和传输协议

代理可设置本地策略将请求发送到特定 IP 地址、端口和传输协议确定的目的地，而这个目的地与 Route 值和 Request-URI 值无关。如果代理不确定这个 IP 地址、端口和传输协议是否与作为松散路由器的服务器对应，那么不可以使用此策略。但是，对于通过特定下一跳发送请求的这种机制，我们并不推荐；对于这种情况，可以使用 Route 头字段。

在没有这种替换机制情况下，代理应按照第四条中所列出的步骤确定往何处发送请求，如下描述。如果代理已对请求重新格式化以将其发送到前面第六步描述的严格路由元素，代理应将这些步骤应用到请求的 Request-URI。否则，如果 Route 头字段的第一个值存在，也就是 Request-URI 存在，代理必须对其应用这些步骤。这些步骤将产生一个有序的数据集（地址、端口、传输协议）。不管使用哪个 URI 作为第四条步骤的输入，如果 Request-URI 指定一个 SIPS 资源，代理就必须将输入 URI 当作 SIPS URI，按照第四条步骤的规定执行。

正如第四条所描述的，代理必须不断尝试将消息传送到元组集的第一个元组，然后依次传送元组集的第二个元组、第三个元组，直到传送成功为止。

对于准备接收消息的元组，代理必须为将消息格式化为适合于该元组的消息，并使用第八条和第 10 条所描述的新的客户端事务发送请求。

由于每一次尝试都使用一个新客户端事务，因此每一次尝试都代表一个新的分支。因而，由第 8 步中插入的 Via 头字段提供的分支参数必须互不相同。

如果客户端事务报告发送请求失败或者状态机超时，该代理将继续发送到有序元组系列的下一个地址。如果到了这个有序系列的末尾，请求便无法转发到目标集合的这个元素中。代理不需要在响应上下文中放入任何东西，但是它的行为就像是这个目标集合的元素返回 408（请求超时）最终响应。

8、 添加一个 Via 头字段值

代理必须在副本中现有的 Via 头字段值之前插入一个 Via 头字段值。这个值的构造遵循第 8.1.1 节相同的规定。这意味着代理将计算自己的分支参数，分支参数对于其对应的分支来说要具有全局唯一性，并且包含必备的 magic cookie。请注意，这意味着对于由代理产生的螺旋请求或者循环请求的不同实例来说，分支参数应有所区别。

选择检测循环的代理在它们用来构造分支参数的值上还有其它限制。选择检测循环的代理应当创建一个分支参数，这个参数由应用分为两部分。第一部分必须满足上述第 8.1.1 节所述的限制。第二部分用来完成循环检测并区别循环请求与螺旋请求。

当请求返回给代理时，通过验证那些对请求处理产生影响的字段没有改变，完成循环检测。分支参数这一部分中的值应当反映所有字段，包括任何 Route 头字段、Proxy-Require 头字段和 Proxy-Authorization 头字段。这是为了确保如果请求路由回代理，并且这些字段中的其中一个发生更改，那么它将被视为螺旋请求而不是循环请求（请参阅第 16.3 节）。创建此值的通用方法是对以下内容作加密哈希运算：所接收请求（在转换之前）的 TO 标记、From 标记、Call-Id 头字段、Request-URI、最上面的 Via 头、Cseq 头字段的序列号，并且如果 Proxy-Require 和 Proxy-Authorization 头字段出现时，还包括它们。用于计算哈希的算法是与执行程序有关，然而以 16 进制表示的 MD5，是一个不错的选择。（对于标记来说，不允许使用 Base64。）

如果代理希望对循环进行检测，那么它提供的 branch 参数必须依赖于所有影响请求处理的信息，包括入站的 Request-URI 和任何影响该请求的进入或路由的头字段。对于区分循环请求与那些在返回给服务器之前其路由参数发生更改的请求，这是必需的。

请求方法不能包含在分支参数的运算中。特别是 CANCEL 请求和 ACK 请求（用于 non-2xx 响应）必须与它们取消或确认的对应的请求有相同的分支参数值。在服务器端处理分支参数的那些相关的请求中使用这些分支参数。（请参阅第 17.3 节和第 9.2 节）。

9、 如果需要，添加一个 Content-Length 头字段

如果使用流传输协议将请求发送到下一跳，并且副本不包含 Content-Length 头字段，那么这个代理必须插入一个带有请求主体的正确值的 Content-Length 头字段（请参阅 20.14 节）。

10、 转发请求

第 17.7 节提到过，有状态代理必须为这个请求创建新的客户端事务，并指导这个事务使用第 7 步确定的地址、端口和传输协议发送请求。

11、 设置计时器 C

为了处理 INVITE 请求从不生成最终响应这种情况，TU 使用一个称为计时器 C 的计时器。当 INVITE 请求得到代理时，必须为每个客户端事务设置计时器 C。必须将计时器设置为大于三分钟。第 16.7 节第 2 条将讨论如何用临时响应更新这个计时器，第 16.8 节将讨论触发该计时器时的处理情况。

16.7 响应处理

当某元素接收到响应时，它首先会试图对匹配该响应的客户端事务定位（第 17.1.3 节）。如果没有找到匹配的客户端事务，这个元素必须将这个响应（即使这个响应是一个信息响应）作为无状态代理（如下所述）处理。如果找到了匹配的客户端事务，那么这个响应便会传递到客户端事务。

在没有客户端事务（或更普遍地说，知道发送了一个相关请求）的情况下转发响应会提高健壮性。特别是它确保将对 INVITE 请求的“延迟的”2xx 响应恰当地转发。

客户端事务将响应传送到代理层时，必须经过以下的处理步骤：

1. 找到合适的响应上下文
2. 为临时响应更新计时器 C
3. 去掉最上面的 Via
4. 对响应上下文添加响应

5. 检查响应是否应当立即转发

6. 当需要的时候，从响应上下文中选择最佳最终响应

如果在每个与响应上下文相关联的客 户端事务终止后没有最终响应得到转发，那么代理必须选择并转发它目前所见到的“最合适”的响应。

所转发的每一个响应必须执行以下处理。每个请求可能有多个响应得到转发：至少每个请求有一个临时响应和一个最终响应。

7. 如果需要，集合授权头字段值

8. 选择性重写 Record-Route 头字段值

9. 转发这个响应

10. 生成任何所需的 CANCEL 请求

上面的每个步骤详述如下：

1. 查找上下文

在使用第 16.6 节描述的方法转发原先的请求之前，代理会对其创建的“响应上下文”进行定位。余下的处理步骤将在这个上下文中进行。

2. 为临时响应更新计时器 C

对于 INVITE 事务来说，如果该响应是一个临时响应，其状态代码为包含 101 和 199 在内，从 101 至 199（即除 100 以外）的任意值，那么代理必须对那个客户端事务的计时器 C 进行重新设置。可以给计时器重新设置一个不同的值，但是这个值必须大于 3 分钟。

3. Via

代理从响应中删除最上面 Via 头字段值。

如果响应中没有保留 Via 头字段值，那么这个响应就是这个元素的响应，不必转发。本节所描述的剩余处理工作不将对该消息执行，而是遵循第 8.1.3 节描述的 UAC 处理规则（已经发生了传输层处理）。

例如，当元素生成第 10 章所述的 CANCEL 请求时，这种情况会发生。

4. 向上下文添加响应

在与上下文相关联的服务器事务上生成最终响应之前，所接收到的最终响应存储在响应上下文中。这个响应可能是那个服务器事务上所返回的候选最佳最终响应。即使这个响应没有被选中，构成最佳响应可能也需要这个响应中的信息。

如果代理选择通过向目标集合添加联系人来在 3xx 响应中对联系人进行递归操作，那么它必须在将这个响应添加到响应上下文之前将这些联系人从响应中删除。但是，如果原先请求的 Request-URI 是 SIPS URI，那么代理不应当对非 SIPS URI 进行递归操作。如果代理对 3xx 响应中的所有联系人都进行递归操作，那么代理不应当向响应上下文添加所产生的无联系人响应。

在向响应上下文添加响应之前去掉联系人可以避免下一个上游元素重新尝试该代理已经尝试过的位置。

3xx 响应既可包含 SIP，也包含 SIPS 和非 SIP URI。代理可以选择递归 SIP 和 SIPS URI，并将其余部分放在将要返回的响应上下文，有可能是在最终响应中。

如果代理接收到了一个请求的 416（不支持的 URI 模式）响应，该请求的 Request-URI 模式不是 SIP，但是原始接收请求中的模式是 SIP 或者 SIPS（即在代理该请求时，代理将模

式从 SIP 或 SIPS 更改到了其它模式），那么该代理应当向目标集合添加一个新的 URI。这个 URI 应当是刚刚尝试过的非 SIP URI 的 SIP URI 版本。在 tel URL 的情况下，这一步通过将 tel URL 的电话订阅用户部分放在 SIP URI 的用户部分中，并将主机部分设置为优先请求所发送的区域。欲了解如何从 tel URL 形成 SIP URI，请参阅 19.1.6 节。

就像 3xx 响应一样，如果代理通过尝试 SIP 或 SIPS URI 来“递归”416，416 不应添加到响应上下文中。

5. 为转发检查响应

直到最终响应在服务器事务发送之前，必须立即转发以下响应：

- 除 100（Trying）以外的任何临时响应
- 任何 2xx 响应

如果接收到一个 6xx 响应，那么它不会被立即转发，但是有状态代理应当取消所有的客户端待处理事务（如第 10 章所述），而且它不能在该上下文中创建任何新的分支。

这是对 RFC2543 所做的更改，RFC2543 要求代理将 6xx 响应立即转发。对于 INVITE 事务来说，该方法有个问题：2xx 响应能够到达另一个分支，而在这种情况下，代理就必须转发 2xx。结果是 UAC 可以在接收到 2xx 响应之前接收到 6xx 响应，而这是不允许发生的。根据新规则，在接收 6xx 之前，代理将发布一条 CANCEL 请求，通常这条请求会从所有的未决客户端事务中产生 487 响应，然后此刻 6xx 便会上游转发。

在服务器事务上发送一条最终响应之后，必须立即转发以下响应：

- 对 INVITE 请求的任何 2xx 响应

有状态代理不必立即转发任何其它响应。特别是有状态代理不可以转发任何 100（Trying）响应。那些后来有可能作为“最佳”响应转发的响应已经集中在一起，如“向上下文添加响应”这一步骤所述。

任何被选择用来立即转发的响应，必须按照从“集合授权头字段值”步骤到“Record-Route”步骤中所描述的内容处理。

这一步骤与下一个步骤会确保有状态代理能向非 INVITE 请求准确转发一个最终响应，或者能向 INVITE 请求准确转发一个非 2xx 响应或者或者多个 2xx 响应。

6. 选择最佳响应

如果没有最终响应按照上述规则得到传送，并且该响应上下文中的所有客户端事务都已经终止，那么有状态代理必须向响应上下文的服务器事务发送一个最终响应。

有状态代理必须从那些所接收到的以及在响应上下文中存储的响应中选择“最佳”最终响应。

如果在上下文中没有最终响应，那么代理必须向服务器事务发送一个 408（响应超时）响应。

否则，代理必须转发来自响应上下文中所存储的响应中的一个响应。如果 6xx 类响应存在于该响应上下文中，代理必须从 6xx 类响应中选择。如果目前没有 6xx 类响应，代理应当从存储在响应上下文中的最低级响应类中选择。代理可以在所选类中任意选择响应。对于可以提供影响请求重新提交的信息的响应，代理应给予优先选择权。例如，如果选择 4xx 类，应首先选择的响应为 401、407、415、420 和 484。

一个接收 503（服务不可用）响应的代理不应该将该响应上游转发，除非该代理可以确定它代理的任何后续请求也可以生成 503 响应。换句话说，转发 503 意味着这个代理明白它

无法为任何请求提供服务，而不仅仅是不能为生成 503 的请求中的 Request-URI 提供服务。如果所接收到的唯一响应是 503，那么这个代理应当生成一个 500 响应并将其上游转发。

所传送的响应必须按照从“集合授权头字段值”步骤到“Record-Route”步骤中所描述内容处理。

例如，如果代理将一个请求转发到 4 个位置，然后接收到 503、407、501 和 404 这几个响应，那么它可以选择传送 407（需要代理认证）响应。

在建立对话过程中可能会涉及到 1xx 响应和 2xx 响应。当请求不包含 To 标记时，UAC 使用响应中的 To 标记来区分由一个创建请求的对话的多个响应。如果请求不包含标记，代理不能向 1xx 和 2xx 的 To 头字段中插入一个标记。代理也不允许修改 1xx 或 2xx 响应的 To 头字段中的标记。

由于代理不可以向不包含标记的请求的 1xx 响应的 To 头字段插入标记，那么它自身无法发布非 100 临时响应。但是它可以将请求分支给一个与代理共享相同元素的 UAS。这个 UAS 可以返回其自身临时响应，进入一个带有该请求启动程序的早期对话中。UAS 不需要成为代理的谨慎处理。它可以是用与代理相同的代码空间实现的虚拟 UAS。

3-6xx 响应是逐跳传送的。当发布一个 3-6xx 响应时，元素会有效地充当 UAS，通常根据从下游元素接收到的响应来发送自己的响应。当一个元素只是简单地将一个 3-6xx 响应转发到一个不包含 To 标记的请求时，该元素应当保存 To 标记。

代理不能对包含 To 标记的请求所转发的任何响应中的 To 标记进行修改。

如果代理替换了所转发的 3-6xx 响应中的 To 标记，对于上游元素来说没有什么不同，但保存原来的标记可以有助于调试。

当代理将几个响应的消息集合在一起时，可以任意从它们当中选择一个 To 标记，但是生成新的 To 标记可能会使调试容易一些。例如，当把 401（未授权）和 407（需要代理认证）结合在一起，或把从未加密的 3xx 响应的 Contact 值和未授权的 3xx 响应的 Contact 值结合在一起时，这种情况会发生。

7. 集合 Authorization 头字段值

如果选择的响应是 401（未授权）或 407（需要代理验证），那么代理必须从这个响应上下文目前接收到的所有的其它 401（未授权）响应和 407（需要代理认证）响应中，收集所有的 WWW-Authenticate 头字段值和 Proxy-Authenticate 头字段值，并在转发之前将它们不加修改地添加到这个响应中。产生的 401（未授权）响应或 407（需要代理认证）响应可能会有几个 WWW-Authenticate 头字段值和 Proxy-Authenticate 头字段值。

这是必需的，因为请求转发到的某些或者所有目的地，可能具有已请求的凭证。客户端需要解决所有的这些难题，并在它重新尝试请求时为它们中的每一个提供凭证。第 26 章说明了这个行为的目的。

8. Record-Route

如果选中的响应包含一个原先由该代理提供的 Record-Route 头字段值，那么该代理可在转发这个响应之前选择重写该值。这使得代理可以下一个上游元素或下游元素提供与其不同的 URI。代理可以以任何理由选择使用本机制。例如，本机制对于多宿主主机来说颇为有用。

如果代理通过 TLS 接收到请求，并通过非 TLS 连接发送这个请求，那么代理必须将 Record-Route 头字段中的 URI 重写为 SIPS URI。如果代理通过非 TLS 连接接收到请求，但将其通过 TLS 发送，那么代理必须将 Record-Route 头字段中的 URI 重写为 SIP URI。

由代理提供的新的URI所满足的条件必须与对请求中Record-Route的URI的约束相同（请参阅16.6节的第4步），并且要做以下修改：

该URI不应包含传输参数，除非代理知道在后续请求的路径中的下一个上游（与下游相对）元素支持此传输协议。

当代理决定修改响应中的Record-Route头字段值时，它执行的一个操作是定位它已经插入的 Record-Route 值。如果请求螺旋，并且这个代理在每个螺旋的迭代中插入了Record-Route 值，那么在响应（它必须以逆时针方向进行适当的迭代）中定位正确的值便比较棘手。以上规则建议希望重写Record-Route头字段值的代理会将足够明显的URI插入到Record-Route头字段中，以便可以选择正确的URI来进行重写操作。达到这一目标的一个推荐的机制是，对于代理来说，要向URI的用户部分附加一个唯一的用于代理实例的标识符。

当响应到达时，代理会对标识符与代理实例相匹配的第一个Record-Route进行修改。修改的结果是，没有这部分数据的URI会附加到URI的用户部分。进行了下一个迭代之后，相同的算法（即用参数查找顶端Record-Route头字段值）将会正确提取那个代理插入的下一个Record-Route头字段值。

并不是请求的每个响应都包含 Record-Route 头字段，代理向该请求添加了Record-Route头字段值。如果响应含有Record-Route头字段，它将包含代理添加的值。

9. 转发响应

执行了“集合授权头字段值”步骤到“Record-Route”步骤的处理后，代理可对选中的响应执行任何特定于功能的操作。代理不能添加、修改或删除消息体。除非另指明，否则代理不能删除除第16.7节第3条讨论的Via头字段值以外的任何头字段值。特别是，代理在处理与该响应相关的请求时，不能删除任何它已经添加到下一个Via头字段值中所接收的参数。代理必须将这个响应传送到与该响应上下文相关联的服务器事务。这将导致响应被发送到目前位于最上面Via头字段值中的位置。如果服务器事务不可以再处理传输，那么该元素必须通过将响应发送到服务器传输来将其无状态转发。该服务器事务可指示发送响应失败或状态机器中信令超时。这些错误可以作为诊断使用，但是协议并不需要代理进行补救。

直到所有与其相关的事务都终止为止，即使在转发最终响应之后，代理都必须对响应上下文进行维护。

10. 生成CANCEL

如果所转发的响应是一个最终响应，那么代理必须为所有与该响应上下文相关联的、待处理的客户端事务生成CANCEL请求。代理也应当在接收到一个6xx响应时为所有与该响应上下文相关联的、待处理的客户端事务生成CANCEL请求。待处理的客户端事务是这样的：它已收到临时响应，但未收到最终响应（即该事务处于进行状态），并且尚未具有为其产生的相关的CANCEL请求。第9.1节中描述了生成CANCEL请求这个问题。

转发最终请求时对CANCEL待处理客户端事务的需求不能确保端点不会收到INVITE的多个200(OK)响应。多个分支上的200(OK)响应可在CANCEL请求得到发送并处理之前生成。此外，将来的扩展有望不考虑发布CANCEL请求的需求。

16.8 处理计时器C

如果计时器C应当触发，那么代理必须用任何它所选择的值来重置这个计时器，或者终止客户端事务。如果客户端事务接收到临时响应，那么代理必须生成一个与该事务相匹配的

CANCEL 请求。如果客户端事务未收到临时响应，那么代理的行为与事务接收到一个 408（请求超时）响应相同。

允许代理可以重置计时器使得代理在计时器触发时可以根据当前情况（如利用率）对事务的生命周期进行动态扩展。

16.9 处理传输错误

如果传输层在代理试图转发一个请求（请参阅第 18.4 节）时通知它出现了错误，那么这个代理的行为与所转发请求接收到了一个 503（服务不可用）响应相同。

如果传输层在代理转发响应时通知有错误产生，那么它将会停止转发这个响应。代理不应当因为这个通知而取消任何与这个响应上下文相关联的尚未处理的客户端事务。

如果代理取消了其尚未处理的客户端事务，那么单一的恶意或不正常的客户端可能会导致所有的事务因其 Via 头字段而失败。

16.10 CANCEL 处理

有状态代理可能会向它在任何时间生成的任何其它请求（由接收那条请求的临时响应决定，如第 9.1 节所示）生成 CANCEL。当代理接收到一个匹配的 CANCEL 请求时，它必须取消任何与这个响应上下文相关的尚未处理的客户端事务。

根据 INVITE 的 Expires 头字段中指定的消逝的时间段，有状态代理可以为尚未处理的 INVITE 客户端事务生成 CANCEL 请求。但是，这通常是不需要的，因为所涉及到的端点会注意对事务的端点发信号。

当 CANCEL 请求由其自身的服务器事务在有状态代理中处理时，新的响应上下文不会为其产生。但代理层会为处理与这个 CANCEL 相关的请求的服务器事务搜索其现有的响应上下文。如果找到了匹配的响应上下文，那么这个元素必须立即向 CANCEL 请求返回一条 200（OK）响应。在这种情况下，该元素的作用如同第 8.2 节中定义的用户代理服务器。此外，该元素必须为这个上下文中所有的待处理的客户端事务生成 CANCEL 请求（如第 16.7 节第 10 步所述）。

如果没有找到响应上下文，那么这个元素没有可以请求应用 CANCEL 的任何信息。它必须将这个 CANCEL 请求无状态转发（它可能先前已经无状态转发了相关的请求）。

16.11 无状态代理

当进行无状态操作时，代理是一个简单的消息转发装置。无状态操作的多数处理与有状态操作相同。下面详细描述了两者的不同之处。

无状态代理没有事务的任何概念，也没有用来描述有状态代理行为的响应上下文的概念。但是无状态代理会从传输层（参阅第 18 章）直接接收消息，包括请求和响应。结果是，无状态代理不能在其自身重传消息。但是它们却会将接收到的所有重传的消息转发（它们没有区分原始请求与重传请求的能力）。此外，当对无状态处理一条请求时，元素不能生成其自己的 100（尝试）响应或任何其它临时响应。

如第 16.3 节所述，无状态代理必须对请求进行验证。

无状态代理必须遵循第 16.4 至第 16.5 节描述的请求处理步骤，以下情况除外：

- 无状态代理必须从目标集合中选择一个唯一的目标。这个选择必须只取决于消息中的字段和服务器的时间不变量属性。特别是每次处理重传的请求时，必须将其转发到相同的目的地。而且，CANCEL 和非路由 ACK 请求必须生成与和它们相关的 INVITE 相同的选择。

无状态代理必须遵循第 16.6 节描述的请求处理步骤，以下情况除外：

- 跨时空的对唯一分支 ID 的需求也可应用到无状态代理中。但是无状态代理不能简单地使用随机数生成器来计算分支 ID 的第一个组件(如第 16.6 节第 8 条所述)。这是因为，请求的重传需要具有相同的值，而无状态代理无法辨别原始请求和重传请求。因此，每次转发重传请求时，使其具有唯一性的分支参数的组件必须相同。所以对无状态代理而言，分支参数必须作为在重传过程中不变的消息参数的组合功能进行计算。

无状态代理可以使用任何它乐意使用的技术来保证其分支 ID 在事务中的唯一性。但是我们推荐以下步骤。代理对接收到的响应的顶端 Via 头字段中的分支 ID 进行检查。如果它以 magic cookie 开始，那么出站请求的分支 ID 的第一个组件可用接收到的分支 ID 的哈希来计算。否则，分支 ID 的第一个组件使用所接收的请求的顶端 Via、To 头字段中的标记、From 头字段中的标记、Call-ID 头字段、Cseq 号（但不是方法）以及 Request-URI 的哈希来计算。在通过两个不同的事务时，这些字段中的一个会改变。

- 第 16.6 节指定的所有其它消息转化都必须使得重传请求能得到相同的转化。特别是，如果代理向 Route 头字段插入 Record-Route 值或将在 Record-Route 头字段中加入 URI，该代理必须在重传请求中设置相同的值。至于 Via 分支参数，这意味着这种转变的根据必须是时间不变量配置或请求的重传不变量属性。
- 无状态代理将确定从哪里转发请求，如第 16.6 节第 10 条对有状态代理的描述。这个请求将被直接发送到传输层，而非经过客户端事务发送。

由于无状态代理必须将重传的请求转发到相同的目的地，还要向这些请求中的每一条添加相同的分支参数，所以它只能使用消息本身的信息和时间不变量配置数据来进行计算。如果配置状态不是时间不变量（例如，如果更新了一个路由表），那么变化影响到的任何请求可能在相当于变化前或变化后的事务超时窗口的时间间隔中不会被无状态传送。在这个间隔中处理受到影响的请求的方法是确定一种解决方案转发请求。一般的解决方案是将这些请求有状态地转发到事务。

无状态代理不能对 CANCEL 请求执行特别的处理。与许多其它请求一样，它们由上述规则处理。特别是无状态代理将相同的 Route 头字段的处理方法应用到了 CANCEL 请求中，而无状态代理也将这种方法应用到了任何其它请求中。

第 16.7 节描述的响应处理不会应用到具有无状态行为的代理上。当响应到达无状态代理时，该代理必须检查第一个（顶端）Via 头字段值中的 sent-by 值。如果这个地址与代理相匹配（这个值与该代理插入到前一个请求中的值相等），那么这个代理必须从响应中将这个头字段值删除，并将结果转发到下一个 Via 头字段值中表示的位置。代理不能添加、修改或删除消息体。除非另行指定，否则代理不能删除任何其它头字段值。如果这个地址与代理不匹配，那么必须丢弃该消息。

16.12 代理路由处理总结

相反，在没有本地策略的情况下，代理对包含 Route 头字段的请求所执行的处理可以归纳为以下几个步骤：

1. 代理将对 Request-URI 进行检查。如果它表示的是一个该代理所拥有的资源，那么代理将用运行定位服务的结果来代替它。否则，代理不会对 Request-URI 进行更改。

2. 代理将对顶端 Route 头字段值中的 URI 进行检查。如果该 URI 表示这个代理，那么代理将把它从 Route 头字段中删除（因为已经遍历了这个路由节点）。
3. 如果没有 Route 头字段值，代理将把这个请求转发到顶端 Route 头字段值或 Request-URI 中的 URI 表示的资源。当代理对此 URI 应用步骤 4 中所描述的要求时，代理可确定要使用的地址、端口和传输协议。

如果在请求路径中没有遇到严格路由的元素，那么 Request-URI 将总会表示请求的目标。

16.12.1 实例

Basic SIP Trapezoid 这个方案是基本的 SIP 四边形，U1→P1→P2→U2，都代理 record-routing。以下是流程：

U1 将：

INVITE sip:callee@domain.com SIP/2.0

Contact: sip:caller@u1.example.com

发送到 P1。P1 是一个出站代理。P1 不对 domain.com 负责，因此，它会在 DNS 中对 domain.com 进行查找，并在 DNS 中将其发送。P1 还会添加一个 Record-Route 头字段值：

INVITE sip:callee@domain.com SIP/2.0

Contact: sip:caller@u1.example.com

Record-Route: <sip:p1.example.com;lr>

P2 接收到这个值。P2 对 domain.com 负责，因此它会运行定位服务并重写 Request-URI。它还会添加一个 Record-Route 头字段值。由于没有 Route 头字段值，因此，它会对新的 Request-URI 进行解析，以确定在哪里发送请求：

INVITE sip:callee@u2.domain.com SIP/2.0

Contact: sip:caller@u1.example.com

Record-Route: <sip:p2.domain.com;lr>

Record-Route: <sip:p1.example.com;lr>

在 U2.domain.com 的被叫方接收到这些内容以 200 OK 作出响应：

SIP/2.0 200 OK

Contact: sip:callee@u2.domain.com

Record-Route: <sip:p2.domain.com;lr>

Record-Route: <sip:p1.example.com;lr>

在 U2 的被叫方还会将其对话状态的远程目标 URI 设置为 sip:caller@u1.example.com，并将其路由集设置为：

(<sip:p2.domain.com;lr>,<sip:p1.example.com;lr>)

这是由 P2 正常转发到 P1 再到 U1 的。现在，U1 将其对话状态的远程目标 URI 设置为 sip:callee@u2.domain.com，并将其路由集设置为：

(<sip:p1.example.com;lr>,<sip:p2.domain.com;lr>)

由于所有的路由集元素都包含 lr 参数，所以 U1 将构造以下 BYE 请求：

BYE sip:callee@u2.domain.com SIP/2.0

Route: <sip:p1.example.com;lr>,<sip:p2.domain.com;lr>

和所有其它元素（包括代理）一样，U1 也会使用 DNS 来对顶端 Route 头字段值中的 URI 进行解析，从而确定从何处发送请求。这里是从 P1 发送。P1 注意到它不对 Request-URI 中指示的资源负责，因此，它不会对资源进行更改。P1 明白这个资源是 Route 头字段中的第一个值，因此它会删除这个值，并将请求转发到 P2：

BYE sip:callee@u2.domain.com SIP/2.0

Route: <sip:p2.domain.com;lr>

P2 也注意到它不对 Request-URI 指示的资源负责（它对 domain.com 负责，但不对 u2.domain.com 负责），因此，它不会对资源进行更改。它也会明白资源在第一个 Route 头字段值中，因此，它会将其删除并根据 DNS 对 Request-URI 的查找情况将以下内容转发给 u2.domain.com：

BYE sip:callee@u2.domain.com SIP/2.0

遍历严格路由代理 在这个方案中，跨越四个代理建立了一个对话，每个代理都添加了 Record-Route 头字段值。第三个代理将实现 RFC2543 中指定的严格路由步骤，处理过程：U1→P1→P2→P3→P4→U2

到达 U2 的 INVITE 包含有：

INVITE sip:callee@u2.domain.com SIP/2.0

Contact: sip:caller@u1.example.com

Record-Route: <sip:p4.domain.com;lr>

Record-Route: <sip:p3.middle.com>

Record-Route: <sip:p2.example.com;lr>

Record-Route: <sip:p1.example.com;lr>

其中 U2 会用 200 OK 作出响应。随后，U2 会根据第一个 Route 头字段值将以下 BYE 请求发送到 P4。

BYE sip:caller@u1.example.com SIP/2.0

Route: <sip:p4.domain.com;lr>

Route: <sip:p3.middle.com>

Route: <sip:p2.example.com;lr>

Route: <sip:p1.example.com;lr>

P4 对 Request-URI 中指示的资源并不负责，因此，P4 将会对其不予理会。P4 注意到这个资源是第一个 Route 头字段值中的元素，因此会将其删除。然后它会准备根据 sip:p3.middle.com 目前处于第一的 Route 头字段值发送请求，但是它注意到这个 URI 并不包含 lr 参数，因此在发送之前，它会将消息重新格式化为：

BYE sip:p3.middle.com SIP/2.0

Route: <sip:p2.example.com;lr>

Route: <sip:p1.example.com;lr>

Route: <sip:caller@u1.example.com>

由于 P3 是严格路由器，所以它会将以下内容转发到 P2：

BYE sip:p2.example.com;lr SIP/2.0

Route: <sip:p1.example.com;lr>

Route: <sip:caller@u1.example.com>

P2 明白这个 request-URI 是它放入 Record-Route 头字段中的值，因此在做进一步处理之前，它会将这个请求重写为：

BYE sip:caller@u1.example.com SIP/2.0

Route: <sip:p1.example.com;lr>

P2 不对 u1.example.com 负责，因此，它将根据 Route 头字段值的解析向 P1 发送请求。

P1 注意到资源位于顶端 Route 头字段值中，因此会将其删除。因而：

BYE sip:caller@u1.example.com SIP/2.0

由于 P1 不对 u1.example.com 负责，并且没有 Route 头字段，因此 P1 将根据 Request-URI 将请求转发到 u1.example.com。

重写 Record-Route 头字段值 在这个方案中，U1 和 U2 位于不同的私有名称空间中，它们通过代理 P1 进入对话，该代理在这里充当名称空间之间的网关。U1→P1→U2

U1 发送：

INVITE sip:callee@gateway.leftprivatespace.com SIP/2.0

Contact: <sip:caller@u1.leftprivatespace.com>

P1 使用其定位服务并向 U2 发送以下内容：

INVITE sip:callee@rightprivatespace.com SIP/2.0

Contact: <sip:caller@u1.leftprivatespace.com>

Record-Route: <sip:gateway.rightprivatespace.com;lr>

U2 将这个 200（OK）返还给 P1：

SIP/2.0 200 OK

Contact: <sip:callee@u2.rightprivatespace.com>

Record-Route: <sip:gateway.rightprivatespace.com;lr>

P1 重写其 Record-Route 头参数，以提供一个 U1 认为有用的值，并将以下内容发送到 U1：

SIP/2.0 200 OK

Contact: <sip:callee@u2.rightprivatespace.com>

Record-Route: <sip:gateway.leftprivatespace.com;lr>

随后，U1 会向 P1 发送以下 BYE 请求：

BYE sip:callee@u2.rightprivatespace.com SIP/2.0

Route: <sip:gateway.leftprivatespace.com;lr>

P1 将其以以下形式转发到 U2：

BYE sip:callee@u2.rightprivatespace.com SIP/2.0

17 事务

SIP 是一个事务协议：组件间的交互发生在一系列独立的消息交换中。具体来说，SIP 事务由一个单一的请求和对这个请求的任何响应(包括零或多个临时响应以及一个或多个最终响应）组成。事务中的请求是 INVITE（称为 INVITE 事务）时，只有最终响应不是 2xx 响应时这个事务才包括 ACK。如果最终响应是 2xx，那么不应将 ACK 视为事务的一部分。

这种区分的根源在于向 UAC 传递 INVITE 的所有 200（OK）响应非常重要。为了将这些响应传递到 UAC，UAS 要负责重传它们（参阅第 13.1.1 节），并且 UAC 要负责用 ACK 对它们进行确认（请参阅第 13.2.2 节）。由于这个 ACK 只是由 UAC 重传的，因此可以有效地将其视为自身的事务。

事务分为客户端与服务器端。客户端称为客户端事务，服务器端称为服务器事务。客户端事务发送请求，服务器事务发送响应。客户端事务与服务器事务都是嵌入任何元素中的逻辑功能。具体来说，它们存在于用户代理和有状态代理服务器中。考虑第 4 章的实例。在这个实例中，UAC 执行客户端事务，其出站代理执行服务器事务。出站代理还会执行一个客户端事务，该事务会将请求发送到入站代理的服务器事务。这个入站代理也会执行一个客户端事务，这个事务将请求发送给 UAS 的服务器事务中。如图 4 所示。

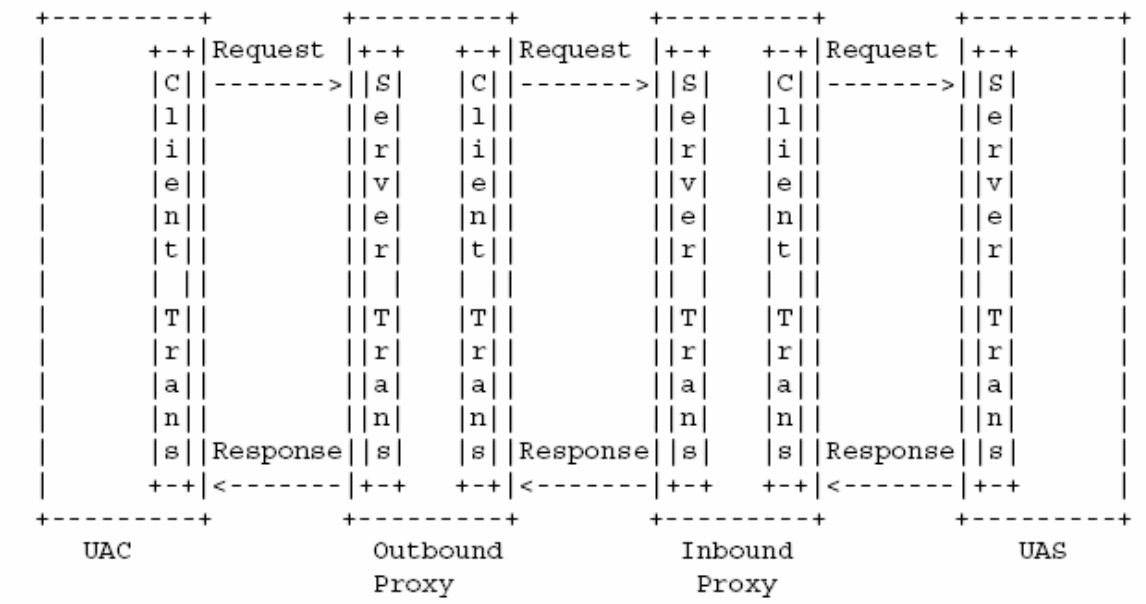

图 4：事务关系

无状态代理不包含客户端事务或服务器事务。事务存在于一端的 UA（或有状态代理）与另一端的 UA（或有状态代理）之间。就 SIP 事务而言，无状态事务是透明的。客户端事务从嵌入客户端的元素（这个元素称为“事务用户”或 TU；它可以是 UA，也可以是有状态代理）接收请求，并将请求可靠地传递给服务器事务。客户端事务还负责接收响应并将这些响应传递给 TU，同时将任何重传响应或不允许的响应（如对 ACK 的响应）过滤。此外，在 INVITE 请求的情况下，客户端事务负责为任何接收 2xx 响应的最终响应生成 ACK 请求。

与此相似，服务器事务从传输层接收请求并将请求交付给 TU。服务器事务会过滤来自网络的任何重传请求。服务器事务接受 TU 的响应并将它们交付给传输层，以在网络上进行传输。在 INVITE 事务中，它为任何最终响应（除 2xx 响应之外）接收 ACK 请求。

2xx 响应及其 ACK 采用特殊的处理方法。这个响应只能由 UAS 重传，并且其 ACK 只由 UAC 生成。这种端到端的处理方法是必要的，以便呼叫方知道接收呼叫的用户集。由于使用了这

种特殊的处理方法，2xx 响应由 UA 核心进行重传，而不是事务层。与此相似，UA 核心也会为 2xx 生成 ACK。沿路径的每个代理仅转发 INVITE 的 2xx 响应以及相应的 ACK。

17.1 客户端事务

客户端事务通过对状态机的维护来提供其功能。

TU 通过一个简单的接口与客户端事务进行通信。当 TU 想发起一个新事务时，这个接口会创建一个客户端事务，并把将要发送的 SIP 请求的目的地的 IP 地址、端口和传输协议传递给客户端。客户端事务开始执行其状态机。有效的响应会从客户端事务上传到 TU。

就 TU 传递请求的方法而言，有两种客户端事务状态机。其中一种用于处理 INVITE 请求客户端事务。这种机器称 INVITE 客户端事务。另一种用于处理除 INVITE 和 ACK 以外的所有请求的客户端事务 。这种机器称非 INVITE 客户端事务。ACK 没有客户端事务。如果 TU 希望发送 ACK，那么它可以直接将 ACK 传送到传输层进行传输。

INVITE 事务因其扩展的持续时间而与其它方法的事务不同。通常地，为了对 INVITE 进行响应，需要用户输入内容。发送响应所需的长时间的延时赞成一个三向握手。另一方面，其它方法的请求有望迅速完成。由于非 INVITE 事务依靠双向握手，TU 应对非 INVITE 请求立即做出响应。

17.1.1 INVITE 客户端事务

INVITE 事务概述：INVITE 事务由一个三向握手组成。客户端事务发送 INVITE，服务器事务发送响应，客户端事务还要发送一个 ACK。对不可靠的传输（如 UDP），客户端事务会在时间间隔 T1 后重传请求，每次重传后该时间翻倍。T1 是对往返时间（RTT）的估计，其缺省值为 500ms。这里描述的几乎所有的事务计时器都可以用 T1 测量，并通过更改 T1 来调整它们的值。可靠传输上的请求不会重传。接收到一个 1xx 响应后，任何重传都停止，客户端会等待进一步的响应。服务器事务会发送额外的 1xx 响应，这些 1xx 响应不会通过服务器事务进行可靠的传送。服务器事务最终会决定发送一个最终响应。对不可靠的传输来说，响应周期性地重传，而对可靠的传输来说，响应则只会发送一次。对客户端事务接收到的每一个最终响应，客户端事务都会发送一个 ACK，其目的是结束响应的重传。

正式描述 图 5 显示了 INVITE 客户端事务的状态机。当 TU 用 INVITE 请求发起一个新的客户端事务时，事务必须进入起始状态，即“正在呼叫中”。客户端事务必须将请求传送到传输层以进行传输（请参阅第 18 章）。如果使用了不可靠的传输，那么客户端事务必须启动值为 T1 的计时器 A。如果使用了可靠的传输，那么客户端事务则不应启动计时器 A（计时器 A 对请求的重传进行控制）。对于任何传输，客户端事务都必须启动具有 64＊T1 秒值的计时器 B（计时器 B 对事务的超时进行控制）。

当计时器 A 触发时，客户端事务必须将请求重传到传输层，并且必须将计时器的值重置为 2＊T1。在传输层上下文中重传的正式定义是：取出先前发送给传输层的消息，再次将其发送到传输层。

当计时器 A 在 2＊T1 秒后触发时，必须重传请求（假定客户端事务仍然处于这种状态）。这一过程必须继续，以便每次传输后在翻倍的时间间隔内重传请求。只有在客户端事务处于“正在呼叫中”状态时，才能完成重传。

T1 的缺省值为 500ms。T1 是客户端事务与服务器事务间 RTT 的估计值。元素可以在不允许连接 Internet 的关闭的私有网络中使用较小的 T1 值（虽然我们不推荐）。如果事先知道（如在高延迟访问链接上）RTT 较大，那么推荐将 T1 的值选得大一些。不管 T1 的值是什

么，都必须使用本章描述的重传的指数退避。

如果当计时器 B 触发时客户端事务仍处于“正在呼叫中”状态，那么客户端事务应当通知 TU 发生了超时。客户端事务不能生成 ACK。64＊T1 的值与在不可靠传输下发送七个请求所需的时间总和相等。

如果客户端事务处于“正在呼叫中”状态时接收到一个临时响应，那么它将转换到“进行”状态。在“进行”状态中，客户端事务不再重传请求。而且，临时响应必须传送到 TU。当客户端事务处于“进行”状态时，任何临时响应都必须上传到 TU。

当处于“正在呼叫中”状态或“进行”状态时，接收到状态码为 300－699 的响应会导致客户端事务转换到“完成”状态。客户端事务必须将接收到的响应上传到 TU，还必须生成一个 ACK 请求，即使在传输是可靠的情况下也是如此（第 17.1.1 节给出了从响应构造 ACK 的指导原则），然后将 ACK 请求传送到传输层进行传输。ACK 发送到的地址、端口和传输协议必须与原始请求发送到的地址、端口和传输协议相同。客户端事务应当在进入“完成”状态时启动计时器 D，对不可靠传输它的值至少应为 32 秒，而对可靠的传输其值应为零。计时器 D 反映了当使用不可靠传输时服务器事务可以停留在“完成”状态的时间总量。这与 INVITE 服务器事务中的计时器 H（其缺省值为 64＊T1）相等。但是，客户端事务不知道服务器事务使用的 T1 值，因此要使用 32s 这个绝对最小值，而不是根据计时器 D 来设定 T1 值。

在“完成”状态时，接收到的最终响应的任何重传都会导致向传输层重传 ACK，以进行重传，但是不能将新接收到的响应上传到 TU。重传响应定义为：根据第 17.1.3 节的规则与同一客户端事务相匹配的任何响应。

如果计时器 D 在客户端事务处于“完成”状态时触发，那么客户端事务必须转换到终止状态。

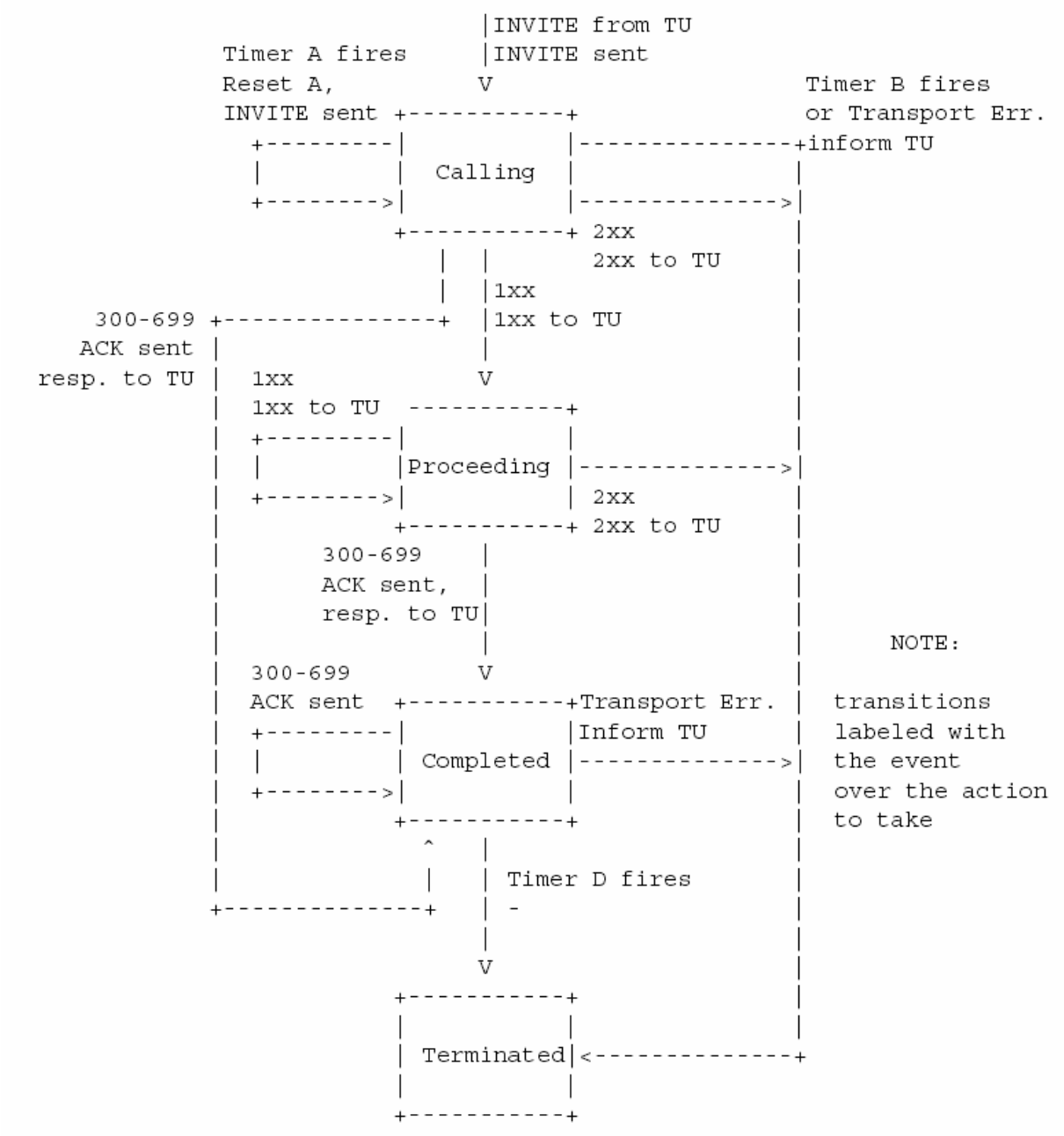

图 5：INVITE 客户端事务

当处于“正在呼叫中”或“进行”状态时，接收 2xx 响应必定导致客户端事务进入“终止”状态，并且响应必须要上传到 TU。对这个响应的处理取决于 TU 是代理核心还是 UAC 核心。UAC 核心将为这个响应生成 ACK，而代理核心则总是将 200（OK）转发到上游。由于对 200（OK）的处理并不是发生在事务层，因此代理与 UAC 会对它有不同的处理。

在客户端事务进入“终止”状态时它本身必须被销毁。实际上这对保证准确的操作是必须的。原因是对 INVITE 的 2xx 响应的处理是不同的：每个响应都是由代理转发的，并且在 UAC 中的 ACK 处理也是不同的。因此，每个 2xx 都需要传送到代理核心（以便可以转发）和 UAC 核心（以便它可以得到确认）。在事务层中没有进行处理操作。在传输接收到响应的任何时候，如果传输层（使用第 17.1.3 节中的规则）没有找到匹配的客户端事务，响应将直接传送到 UAC 核心。由于第一个 2xx 已经销毁了匹配的客户端事务，后续的 2xx 会发现没有匹配的客户端事务，所以才会直接传送到核心。

ACK 请求的构造 这一章指定了在客户端事务中发送的 ACK 请求的构造。为 2xx 生成 ACK 的 UAC 核心必须遵循第 13 章描述的规则。

由客户端事务构造的 ACK 请求必须包含 Call-ID、From 和 Request-URI 的值，这些值与客户端事务传送到传输层的请求（称为“原始请求”）中的那些头字段的值相等。ACK 中的 To 头字段必须与正在被确认的响应中的 To 头字段相等，但由于它添加了标记参数，因此它常常与原始请求中的 To 头字段不同。ACK 必须包含一个单一的 Via 头字段，它必须与原始请求的顶端 Via 头字段值相等。ACK 中的 Cseq 头字段包含的序列号值必须与原先请求中的相同，但是方法参数必须与“ACK”相等。

如果其响应正在被确认的 INVITE 请求有 Route 头字段，那么这些头字段必须在 ACK 中出现。这是为了确保可以通过任何下游无状态代理正确地路由 ACK。

虽然任何请求都可以包含一个消息体，但是 ACK 中的消息体比较特殊，因为如果没有理解消息体就无法拒绝请求。因此，不推荐在非 2xx 的 ACK 中放置消息体，但是如果已经放置了消息体，假定 INVITE 的响应不是 415，那么消息体类型要限制在任何在 INVITE 中出现的类型。如果 INVITE 的响应是 415，那么 ACK 中的消息体可以是 415 中 Accept 头字段中列出的任何类型。

例如，请考虑以下请求：

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKkjshdyff
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=88sja8x
Max-Forwards: 70
Call-ID: 987asjd97y7atg
CSeq: 986759 INVITE
```

该请求的非 2xx 最终响应的 ACK 请求如下所示:

```
ACK sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKkjshdyff
To: Bob <sip:bob@biloxi.com>;tag=99sa0xk
From: Alice <sip:alice@atlanta.com>;tag=88sja8x
Max-Forwards: 70
Call-ID: 987asjd97y7atg
CSeq: 986759 ACK
```

17.1.2 非 INVITE 客户端事务

非 INVITE 事务概述 非 INVITE 事务不使用 ACK。它们是简单的请求--响应交互。对于不可靠的传输来说，请求在一个时间间隔（时间间隔起始值为 T1，在到 T2 前该时间间隔每次翻倍）内进行重传。如果接收到一个临时响应，那么对不可靠传输而言，将继续重传（但时间间隔在 T2 内）。服务器事务只有在接收到重传请求时，才会重传它发送的最后一个响应，这个响应可以是临时响应，也可以是最终响应。这就是即使在临时响应之后，仍需继续重传请求的原因：它们是为了确保可靠的传输最终响应。

与 INVITE 事务不同，非 INVITE 事务对 2xx 响应并不做特殊的处理。结果是，只向 UAC 传送了非 INVITE 的一个 2xx 响应。

正式描述 图 6 显示了非 INVITE 客户端事务的状态机。它与 INVITE 状态机非常相似。

当 TU 用请求发起新的客户端事务时，事务会进入“尝试”状态。进入这个状态时，客户端事务应当将计时器 F 设置为在 64＊T1 秒内触发。这个请求必须传送到传输层进行传输。如果使用的是不可靠的传输，那么客户端事务必须将计时器 E 设置为在 T1 秒之内触发。如果计时器 E 在客户端事务仍处于尝试状态就触发，那么计时器将会被重置，但是这次重置的值为 MIN（2＊T1，T2）。当计时器再一次触发时，它会被重置为 MIN（4＊T1，T2）。这一过程将会继续，以便在一个以指数递增的时间间隔（最大值为 T2）内发生重传。T2 的缺省值为 4 秒，如果服务器事务未立即响应的话，T2 代表非 INVITE 服务器事务将对请求发出响应所用的时间总和。对于 T1 和 T2 的缺省值来说，这将会产生 500ms、1s、2s、4s、4s、4s 的时间间隔。

如果计时器 F 在客户端事务仍处于“尝试”状态时触发，那么客户端事务应当向 TU 报告超时，然后应进入“终止”状态。如果在“尝试”状态下接收到了一个临时响应，那么这个响应必须传送到 TU，然后客户端事务应转换到“进行”状态。如果在“尝试”状态接收到一个最终响应（状态码为 200－699），那么这个响应必须要传送到 TU，客户端事务则必须转换到“完成”状态。

如果计时器 E 在客户端事务处于“进行”状态时触发，那么请求必须传送到传输层进行重传，计时器 E 的值也必须重置为 T2 秒。如果计时器 F 在“进行”状态时触发， TU 必须得到超时通知，并且客户端事务必须转换到终止状态。如果在“进行”状态时接收到一个最终响应（状态码为 200－699），那么这个响应必须要传送到 TU，而客户端事务则必须转换到“完成”状态。

一旦客户端事务进入“完成”状态，对不可靠传输，它必须将计时器 K 设置在 T4 秒之内触发；对可靠传输，计时器应设置为零。“完成”状态的存在是为了对可能接收到的任何额外重传响应起到缓冲作用（这也是客户端事务只对于不可靠的传输才保留的原因）。T4 表示网络清除客户端事务和服务器事务间消息所需的时间总和。T4 的缺省值为 5s。当响应使用第 17.1.3 节中指定的规则与同一事务匹配时，它便是重传响应。如果计时器 K 在这个状态触发，那么客户端事务则必须转换到“终止”状态。

一旦事务处于终止状态，必须立即销毁它。

17.1.3 将响应与客户端事务匹配

当客户端的传输层接收到响应时，它必须确定哪一个客户端事务来处理这个响应，以进行第 17.1.1 节和第 17.1.2 节所描述的处理。为此，需要使用顶端 Via 头字段中的分支参数。在两种情况下响应会与客户端事务匹配：

1. 响应在顶端 Via 头字段中的分支参数值与创建事务的请求的顶端 Via 头字段的分支参数值相同；
2. Cseq 头字段中的方法参数与创建事务的请求的方法匹配。由于 CANCEL 请求组成了共享分支参数值的不同事务，因此需要这个方法。

如果请求通过多播发送，那么它可能会从不同的服务器生成多个响应。这些响应在顶部 Via 中有相同的分支参数值，但在 To 标记中它们的分支参数值却不同。接收到的第一个响应将会使用上面的规则，而其它响应将视为重传响应。这并非出现了错误；多播 SIP 只提供根本的“single-hop-discovery-like”服务，这个服务只限于处理单一的响应。细节请参

阅第 18.1.1 节。

17.1.4 处理传输错误

当客户端事务向传输层发送请求时，如果传输层发生错误，可以遵循以下步骤。

客户端事务应当通知 TU 发生了传输错误，并直接转换到“终止”状态。TU 将处理[4]中描述的错误转移机制。

17.2 服务器事务

服务器事务负责将请求传递到 TU 以及对响应进行可靠的传输。它通过状态机来完成这些任务。当收到请求时，并且请求需要事务处理时（实际情况并非总是如此），核心创建服务器事务。

与客户端事务相同，服务器事务的状态机取决于接收到的请求是否是 INVITE 请求。

17.2.1 INVITE 服务器事务

图 7 是 INVITE 服务器事务的状态图。

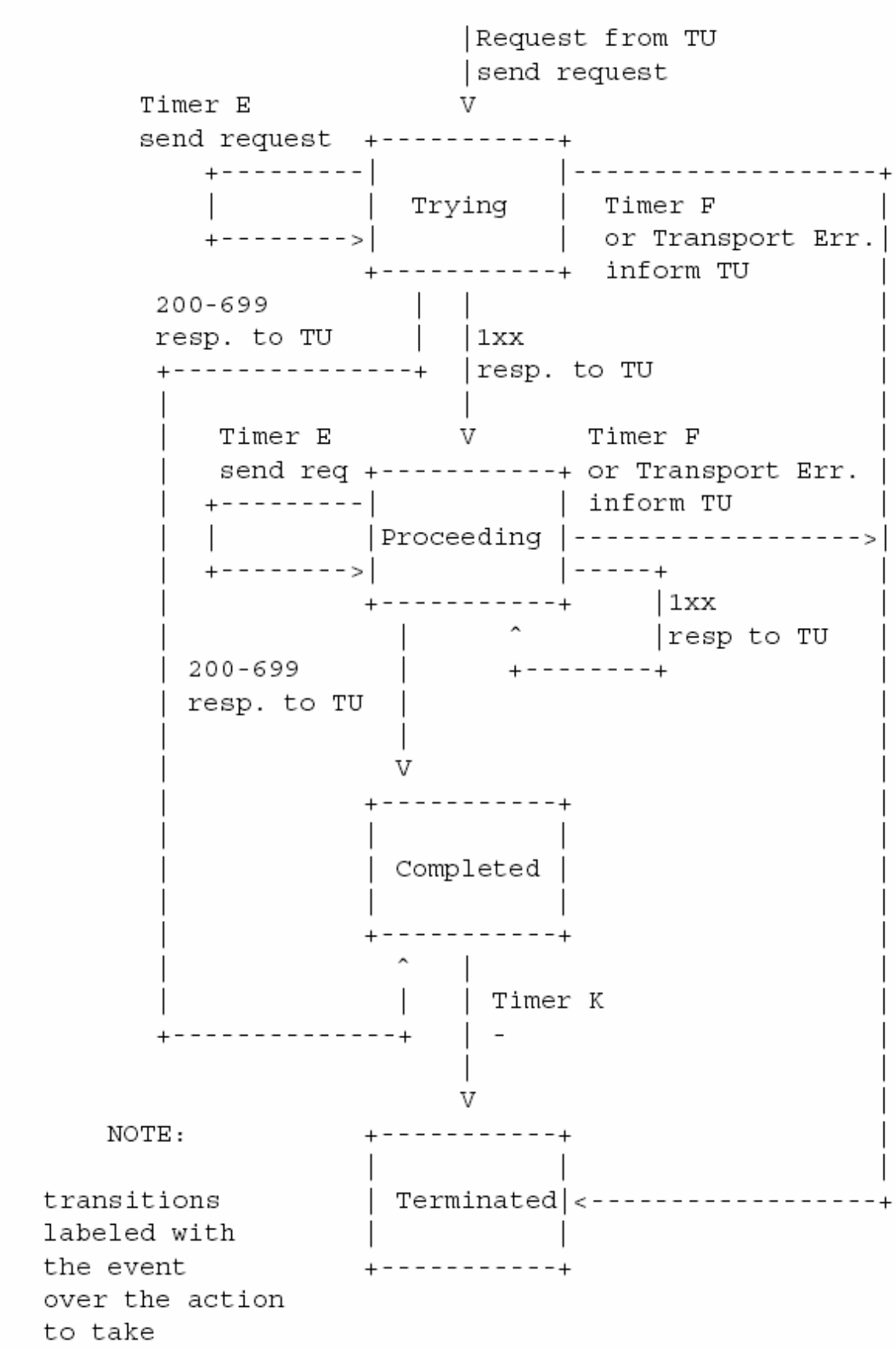

图 6：非 INVITE 客户端事务

当为请求构造服务器事务时，服务器事务便进入了“进行”状态。服务器事务必须生成

一个 100（尝试）响应，除非它知道 TU 将在 200ms 内生成一个临时或最终响应，在这种情况下，它可能会生成一个 100（尝试）响应。为了避免网络拥塞，需要临时响应迅速结束重传请求。除了把在响应的 To 头字段中插入标记（当请求的 To 头字段中没有标记时）这一点从 MAY 降为 SHOULD NOT 以外，100（尝试）响应是根据第 8.2.6 节的步骤构造的。这个请求必须传送到 TU。

TU 将任何数量的临时响应传送到服务器事务。只要服务器事务处于“进行”状态，所有这些响应都要传送到传输层进行传输。这些响应不是由事务层可靠发送的（它们不是由事务层重传的），因此不会导致服务器事务的状态改变。如果在“进行”状态中收到了重传请求，那么 TU 接收到的最新的临时响应必须要传送到传输层以进行重传。如果一个请求根据第 17.2.3 节的规则与同一服务器事务相匹配，那么这个请求便是重传请求。

当处于“进行”状态时，如果 TU 将一个 2xx 响应传送到服务器事务，那么服务器事务必须将这个响应传送到传输层进行传输。2xx 响应不是由服务器事务重传的，而是由 TU 重传的。然后服务器事务必须转换到“终止”状态。

当处于“进行”状态时，如果 TU 向服务器事务传送一个状态码为 300－699 的响应，那么这个响应必须要传送到传输层以进行传输，并且状态机必须进入“完成”状态。对于不可靠的传输来说，要将计时器 G 设置为在 T1 秒内触发，而对于可靠的传输则无需设置触发时间。

在 RFC 2543 中，即使对于可靠的传输，响应也总是要重传的。本规范在这一点上对 RFC 2543 进行了改动。

当进入“完成”状态时，对所有的传输，计时器 H 必须设置为在 64＊T1 秒触发。计时器 H 确定服务器事务何时不再重传响应。它的值应与计时器 B 相同，等于客户端事务继续重发请求的时间总和。如果计时器 G 触发，那么响应便会再次发送到传输层进行重传，计时器 G 设置为在 MIN（2＊T1，T2）秒内触发。从这个时间开始，当计时器 G 触发时，响应便会再次传送到传输层进行传输，而计时器 G 的值将会重置为先前的两倍，直到值超过 T2，此时，应将计时器 G 的值重置为 T2。这与非 INVITE 客户端事务处于“尝试”状态时对请求进行重传的行为如出一辙。此外，当处于“完成”状态时，如果接收到重传请求，那么服务器应当将响应传送到传输层进行重传。

如果服务器事务在“完成”状态时接收到 ACK，那么它必须转换到“确认”状态。由于在这个状态可以忽略计时器 G，因此，所有的响应都将停止重传。

如果在“完成”状态中计时器 H 触发，这表明服务器事务从未接收到 ACK。在这种情况下，服务器事务必须转换到“终止”状态，并且必须向 TU 指明发生了事务错误。

“确认”状态是为了接收任何到达的附加的 ACK 消息，该消息是由重传的最终响应触发的。当进入这种状态时，对于不可靠传输计时器 I 设置为在 T4 秒内触发；而对于可靠传输则设置为零。计时器 I 一旦触发，服务器必须转换到“终止”状态。

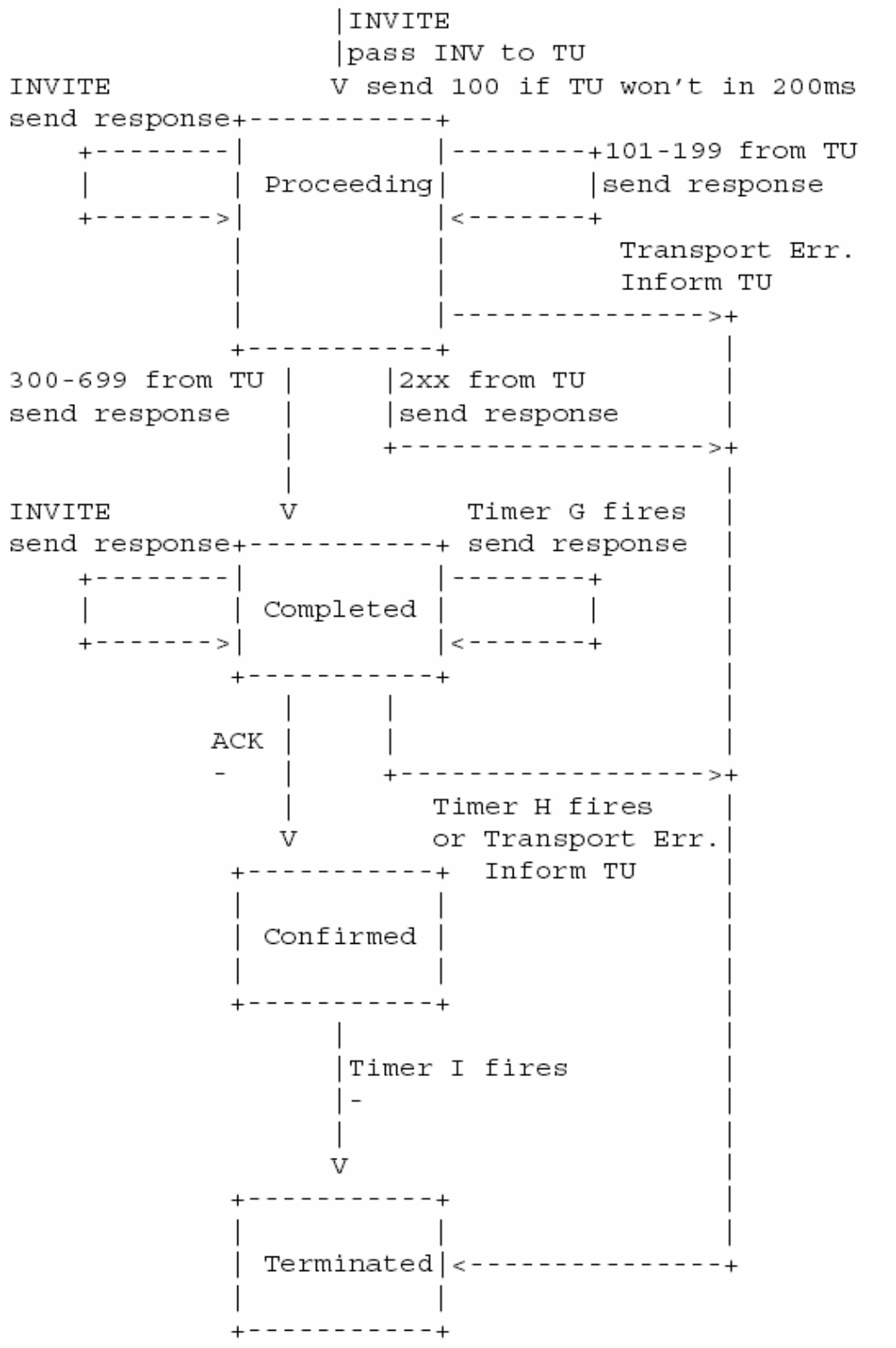

图 7：INVITE 服务器事务

一旦事务处于“终止”状态，必须要立即销毁它。与客户端事务相似，这是为了确保 INVITE 的 2xx 响应的可靠性。

17.2.2 非 INVITE 服务器事务

图 8 给出了非 INVITE 服务器事务的状态机。

这个状态机在“尝试”状态时初始化，初始化时会向状态机发送一个除 INVITE 或 ACK 之外的请求。这个请求将上传到 TU。一旦处于“尝试”状态，任何进一步的重传请求都会被丢弃。如果请求使用第 17.2.3 节中的规则与同一服务器事务相匹配，那么这个请求便是重传请求。

当处于“尝试”状态时，如果 TU 将一个临时响应传送到了服务器事务，那么服务器事务必须进入“进行”状态。这个响应必须要传送到传输层传输。在“进行”状态时从 TU 接收到的任何进一步的临时响应都必须传送到传输层进行传输。假若在“进行”状态接收到重传请求，那么最新发送的临时响应必须传送到传输层重传。如果在“进行”状态中 TU 将最终响应（状态码为 200－699）传送到服务器，那么事务必须进入“完成”状态，这个响应则必须传送到传输层进行传输。

当服务器事务进入“完成”状态时，对不可靠传输，它必须将计时器 J 设置为在 64＊T1 秒内触发；对可靠传输，设置为零秒触发。当处于“完成”状态时，在接收到重传响应的任何时候，服务器事务都必须将最终响应传送到传输层重传。在“完成”状态时，任何由 TU 传送到服务器事务的其它最终响应都必须被丢弃。服务器事务将保持这种状态，直到计时器 J 触发，这时服务器事务必须转换到“终止”状态。

当服务器事务进入“终止”状态时，必须要销毁它。

17.2.3 将请求与服务器事务匹配

当服务器从网络接收到请求时，该请求必须与现有的事务相匹配。匹配按以下方式进行：

首先要检查请求顶端 Via 头字段中的分支参数。如果有分支参数，并且以 magic cookie “z9Hg4BK”开头，那么这个请求便是由符合本规范的客户端事务生成的。因此，客户端发送的所有事务中的各个分支参数都应当不同。如果满足以下条件，则请求与事务相匹配：

1. 请求中的分支参数与创建事务的请求的顶端 Via 头字段中的分支参数相等；
2. 请求的顶端 Via 头字段中的 sent-by 值与创建事务的请求的顶端 Via 头字段中的 sent-by 值相等；
3. 请求的方法与创建事务的方法相同（除 ACK 以外，它的创建事务的请求方法为 INVITE）。

以上匹配规则适用于 INVITE 事务和非 INVITE 事务。

Sent-by 值是作为匹配过程的一部分使用的，因为不同客户端可能会对分支参数进行意外或恶意的复制。

如果顶端 Via 头字段中没有分支参数，或者分支参数不包含 magic cookie，那么可以使用以下步骤。这些步骤用来处理与遵从 RFC 2543 规范的实现的向后兼容问题。

如果一个 INVITE 请求的 Request-URI、To 标记、From 标记、Call-ID、Cseq 以及顶端 Via 头字段与创建事务的 INVITE 请求的对应内容相匹配，那么它与这个事务是匹配的。在这种情况下，INVITE 是创建这个事务的原始请求的重传请求。如果 ACK 请求的 Request-URI、From 标记、Call-ID、Cseq 号（不是方法）以及顶端 Via 头字段与创建事务的 INVITE 请求的对应内容相匹配，并且这个 ACK 的 To 标记与服务器事务发送的响应的 To 标记相匹配，那么这个 ACK 请求就与这个事务相匹配。匹配是根据为每个头字段定义的匹配规则进行的。在 ACK 匹配过程中在 To 头字段中包括一个标记有助于区分 2xx 的 ACK 与代理上其它响应的 ACK，

代理有可能将两种响应一起转发（在异常情况下，这是有可能发生的。具体说来，当代理分发一个请求，然后崩溃，响应可能会传递到另一个代理，这个代理可能会终止向上游转发多个响应）。可以认为，如果一个 ACK 请求与和前一个 ACK 匹配的 INVITE 事务相匹配，那么它是先前 ACK 请求的重传。

对于所有其它请求方法而言，如果一个请求的 Request-URI、To 标记、From 标记、Call-ID、Cseq（包括方法）以及顶端 Via 头字段与创建事务的请求的对应内容相匹配的话，那么该请求便与这个事务相匹配。匹配是根据为每个头字段定义的匹配规则进行的。当一个非 INVITE 请求与一个现有的事务相匹配时，它是创建这个事务的请求的重传。

由于匹配规则包括 Request-URI，因此服务器无法将响应与事务相匹配。当 TU 将一个响应传送到服务器事务时，它必须将其传递到这个响应指向的特定的服务器事务。

17.2.4 处理传输错误

当服务器事务向传输层发送一个响应时，如果传输层显示错误，则需遵循以下步骤：

首先要遵循[4]中的步骤，这些步骤试图将响应传递到备份。如果这些步骤全部失败，根据[4]中对错误的定义，服务器事务应当通知 TU 发生了错误，并应当转换到终止状态。

18 传输

传输层负责在网络上实际传输请求与响应。这包括在面向连接的传输中确定请求或响应要使用的连接。

传输层还负责为传输协议如 TCP、SCTP 或在这些协议上的 TLS（包括那些向传输层开放的协议）管理持久的连接。这包括由客户端或服务器传输层开放的连接，以便在客户端和服务器传输功能之间共享这些连接。这些连接可用连接的另一端的地址、端口和传输协议构成的元组来索引。当连接由传输层打开时，这个索引设置为目的 IP、端口号和传输协议。当传输层接受此连接时，这个索引则会设置为源 IP 地址、端口号和传输协议。请注意，由于源端口经常是短暂的，但是我们无法知道它是短暂的还是通过[4]中的步骤选择的，传输层接受的连接通常不会被重用。结果是使用面向连接的传输协议的、处于“对等”关系的两个代理通常会使用两个连接，其中每个连接用于在一个方向上初始化的事务。

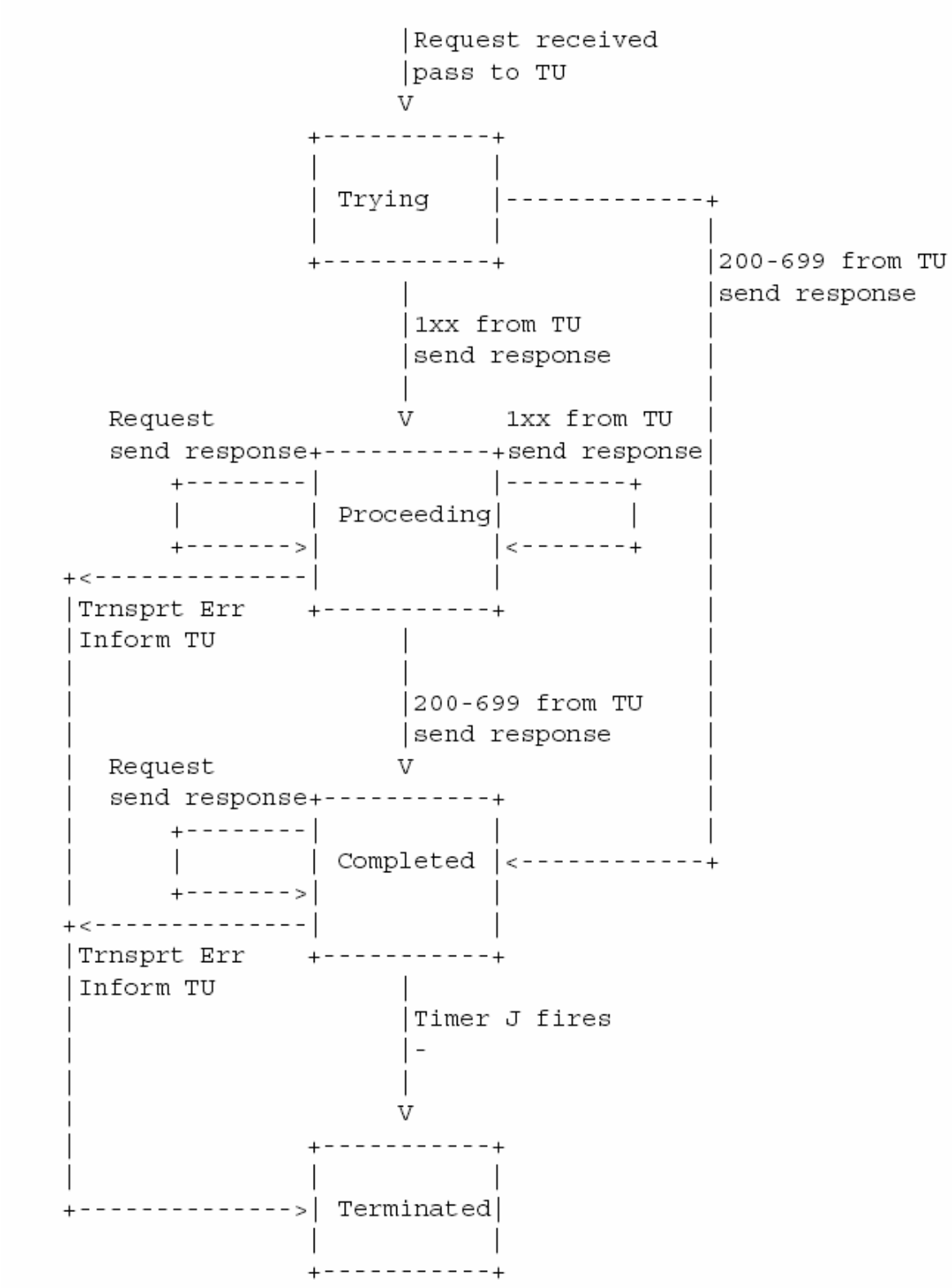

图 8：非 INVITE 服务器事务

对于一些由实现定义的持续时间而言，在该连接上发送或接收最后一条消息后，推荐将这个连接保持为开放状态。为保证一个事务从初始化到终止状态，这个持续时间至少应当与元素需要的最长时间相等。这使得事务可能在启动它们（如请求、响应以及非 2xx 响应的 INVITE 和 ACK)的同一个连接上完成。这通常意味着至少是 64*T1(T1 的定义请参阅第 17.1.1

节)。但是，如在一个具有对计时器 C 使用较大值的 TU 的元素中，这个值可以大一些（请参阅第 16.6 节第 11 条)。

所有 SIP 元素必须实现 UDP 和 TCP。SIP 元素可能实现其它协议。

本规范对 RFC254 进行了很大的改动，在本规范中，TCP 对于 UA 是必须的。它可以处理更大的消息，这些消息必须要使用 TCP，这一点下面将讨论到。即使一个元素从未发送过大的消息，它也会接收到这样的消息，并要求能够对其进行处理。

18.1 客户端

18.1.1 发送请求

传输层的客户端负责发送请求和接收响应。传输层的用户将请求传送到客户端传输，请求的内容包括 IP 地址、端口、传输协议和用于多播目的地的可能的 TTL。

如果一个请求在路径 MTU 的 200 字节内，或者大于 1300 字节并且路径 MTU 是不可知的，那么必须使用 RFC2914[36]拥塞控制传输协议（如 TCP）进行发送。如果这导致与顶端 Via 中所指的传输协议不同，那么就必须更改顶部 Via 中的值。这避免了 UDP 上的消息分段，并对较大的消息进行拥塞控制。但是，实现必须能够处理那些具有最大数据报包大小的消息。对于 UDP 来说，这个最大消息为 65,535 字节（包括 IP 头和 UDP 头在内)。

消息大小与 MTU 之间的 200 字节的“缓冲”适应了这样一个事实：SIP 中的响应可以比请求大。例如，这是由于在 INVITE 的响应中添加了 Record-Route 头字段值。有了这个额外的缓冲，响应可以比请求大 170 字节，并且在 Ipv4 上仍可以不分段(假设没有 IPSec，IP/UDP 需占用 30 字节)。当路径 MTU 未知时，根据对 1500 字节以太网 MTU 的假设，选择 1300 字节。

如果由于消息长度的限制，元素没使用 UDP 而用 TCP 发送请求；如果建立连接的尝试产生的结果是不支持 ICMP 协议或 TCP 复位，那么元素应当使用 UDP 重试这个请求。这只是提供了与不支持 TCP 而遵从 RFC2543 的实现的向后兼容性。据预测，本规范的以后版本将不推荐这个行为。

发送请求到多播地址的客户端必须向其包含目的多播地址的 Via 头字段值中添加 maddr 参数，对于 Ipv4 来说，则添加值为 1 的 ttl 参数。本规范没有定义 Ipv6 多播的用法，如果此需求增加，未来的标准化版本可能会对其进行阐述。

这些规则导致对 SIP 中的多播的限制。其主要功能是提供“single-hop-discovery-like”服务，即向一组同类的服务器发送请求，其中它只需要处理从这些服务器中的任何一台发出的响应。这一功能对于注册最为重要。实际上，根据第 17.1.3 节所述的事务处理规则，客户端事务将接收第一个响应，并将其它响应作为重传来查看，因为所有这些响应都包含了相同的 Via 分支标识符。

发送请求之前，客户端传输必须向 Via 头字段中插入一个 sent-by 字段值。这个字段包含一个 IP 地址或主机名称以及端口。我们推荐使用 FQDN。在某些情况下（如下所述），这个字段用来发送响应。如果没有端口，那么缺省值则取决于传输。对于 UDP、TCP 和 SCTP 来说，缺省值为 5060，对于 TLS 则是 5061。

对于可靠的传输而言，通常会在接收请求的同一连接上发送响应。因此，客户端传输必须在发送请求的同一连接上做好接收响应的准备。在错误的情况下，服务器可能会试图打开一个新连接发送这个响应。在这种情况下，传输层必须做好接收入站的连接的准备，该连接来自发送请求的源 IP 地址和 sent-by 字段中的端口号。它还必须准备好在任何地址和端口接收入站的连接，该地址和端口是一个服务器根据[4]中第 5 部分描述的过程选择的。

对于不可靠的单播而言，客户端传输必须做好准备从发送请求的源 IP 地址（因为响应会发回源地址）和 semt-by 字段的端口号接收响应。而且，与可靠的传输相同，在某些情况下，响应将发送到其它地方。客户端必须准备好在任何地址和端口接收响应，该地址和端口是一个服务器根据[4]中第 5 部分描述的过程选择的。

对于多播来说，客户端传输必须准备好从请求所发送的相同多播组和端口（即，需要将请求发送到多播组成员）接收响应。

如果一个请求发往一个现有连接已开放的 IP 地址、端口和传输协议，那么我们推荐这个现有连接用来发送请求，同时可开放和使用另一个连接。

如果使用多播发送请求，那么该请求发送到由传输用户提供的组地址、端口和 TTL。如果使用单播不可靠传输协议发送请求，那么它便发送到由传输用户提供的 IP 地址和端口。

18.1.2 接收响应

接收到一个响应时，客户端传输便会对顶端 Via 头字段值进行检查。如果头字段值中的 semt-by 参数值与配置客户端传输时插入到请求中的值不匹配，那么必须丢弃此响应。

如果存在任何客户端事务，那么客户端传输便会使用第 17.1.3 节中的匹配过程尝试将这个响应与现有的事务相匹配。如果找到了匹配，那么必须将响应传送给事务。如果没有找到匹配，则必须把这个响应传送到核心（不管它是无状态代理、有状态代理或 UA）以进行进一步的处理。对这些“偏离的”响应的处理取决于核心（例如，代理会转发这些响应，而 UA 则丢弃它们）。

18.2 服务器

18.2.1 接收请求

服务器应当准备在任何 IP 地址、端口和传输协议的联合上接收请求，该联合是 DNS 对分发的、用来与服务器进行通信的 SIP 或 SIPS URI[4]查找的结果。此时，“分发”包括将一个 URI 放入 RGDISTER 请求或重定向响应的 Contact 头字段中，或放入一个请求或响应的 Record-Route 头字段中。还可以通过将 URI 放在 web 页面或名片上做到分发。还推荐一个服务器在所有公共接口的默认 SIP 端口（TCP 和 UDP 的默认 SIP 端口为 5060， TCP 上的 TLS 的默认 SIP 端口为 5061）侦听消息。典型的例外情况就是私有网络，或多个服务器实例在同一主机上运行。当服务器侦听任何端口和接口上的 UDP 时，它必须同时侦听该端口和接口上的 TCP。这是因为消息过长时，会使用 TCP 发送消息，而不使用 UDP。反之则不正确。如果一个服务器在一个特殊的地址和端口上侦听 TCP，那么它就不需要在相同的地址和端口上侦听 UDP。当然，服务器为什么需要在特殊的地址和端口上侦听 UDP，可能有其它的原因。

当服务器在任何传输协议上接收到请求时，它必须检查顶端 Via 头字段值中的 sent-by 参数值。如果 sent-by 参数的主机部分包含一个域名，或它包含一个与包源地址不同的 IP 地址，那么服务器必须向 Via 头字段值中添加 received 参数。这个参数必须包含接收包的源地址。这是为了帮助服务器传输层发送响应，因为必须将响应发到发送请求的源 IP 地址。

考虑一个实例，服务器传输接收到一个请求，下面是该请求的一部分：

INVITE sip:bob@Biloxi.com SIP/2.0

Via: SIP/2.0/UDP bobspc.biloxi.com:5060

服务器传输从源 IP 地址 192.0.2.4 接收到该请求。在将其上传之前，传输会添加一个 received 参数，因此，请求的一部分的形式如下：

INVITE sip:bob@Biloxi.com SIP/2.0

Via: SIP/2.0/UDP bobspc.biloxi.com:5060;received=192.0.2.4

接下来，服务器传输将尝试将这个请求与一个服务器事务相匹配。该匹配按照第 17.2.3 节中描述的匹配规则进行。如果找到了匹配的服务器事务，那么这个请求发送到该事务以进行处理。如果没有找到匹配的服务器事务，这个请求将会发送到核心，核心会决定为这个请求构造一个新的服务器事务。注意，如果 UAS 核心向 INVITE 发送 2xx 响应，那么服务器事务就被破坏了。这表明当接收到 ACK 时，便不会有匹配的服务器事务，基于这一规则，将 ACK 发送到处理请求的 UAS 核心。

18.2.2 发送响应

为了确定在哪里发送响应，服务器传输会使用顶端 Via 头字段值。它必须遵循以下步骤：

- 如果“发送协议”是可靠的传输协议，如 TCP 或 SCTP，或者是在这两者之上的 TLS，那么必须使用现有的连接将响应发送到创建事务的原始请求的源地址（如果该连接仍然是开放状态）。这需要服务器传输在服务器事务和传输连接之间维持联系。如果连接不再开放，那么服务器应使用 sent-by 值中的端口（如果出现），或在没有指定端口时使用传输的默认端口，打开一个 received 参数中的 IP 地址的连接。如果连接尝试失败，那么为了确定打开连接和发送响应的 IP 地址和端口，服务器应使用[4]中的过程。
- 此外，如果 Via 头字段值包含一个 maddr 参数，那么必须使用 sent-by 中指明的端口或端口 5060（没有指定时）将响应转发到此参数中列出的地址。如果地址是多播地址，那么应使用 ttl 参数中指明的 TTL 发送响应，或者如果没有这个参数，可以使用值为 1 的 TTL 发送响应。
- 另外（对于不可靠单播传输），如果顶端 Via 有一个 received 参数，那么应使用 sent-by 值中指定的端口，如果没有明确指定则可使用端口 5060，将响应发送到 received 参数中的地址。如果发送失败，例如，得到一个 ICMP“端口不可到达”的响应，那么就应使用[4]的第 5 部分的过程来确定向哪里发送响应。
- 此外，如果响应不是由接收方标记的，那么这个响应必须使用[4]中第 5 部分的过程发送到 sent-by 值指明的地址。

18.3 帧

在面向消息的传输（如 UDP）中，如果消息有一个 Content-Length 头字段，那么就假定消息体包含对应长度的字节。如果传输包中有超过消息体长度的字节，那么必须将这些多余的字节丢弃。如果传输包在消息体之前就结束，那么便认为产生了错误。如果这个消息是个响应，则必须丢弃。如果这个消息是个请求，则元素应生成一个 400（坏请求）响应。如果消息没有 Content-Length 头字段，那么就假定消息体在传输包结束时结束。

在面向流的传输的情况下，如 TCP，Content-Length 头字段表示消息体的大小。Content-Length 头字段必须与面向流的传输一起使用。

18.4 错误处理

错误处理与消息是请求还是响应没有关系。

如果传输层用户要求在不可靠传输协议上发送消息，并且结果是 ICMP 错误，那么这个

行为取决于 ICMP 错误的类型。主机、网络、端口或协议不可访问错误或参数问题错误都致使传输层通知传输层用户在发送过程中发生了错误。应当忽略源站抑制和 TTL 溢出 ICMP 错误。

如果传输层用户要求在可靠传输协议上发送请求，并且结果是连接失败，那么传输层应通知传输用户在发送过程中发生了错误。

19 通用消息组件

在 SIP 消息(有时在 SIP 消息外)的各个地方有一些 SIP 消息组件，它们值得分别讨论。

19.1 SIP 和 SIPS 统一资源指示器

SIP 或 SIPS URI 用来标识通信资源。与所有的 URI 相同，SIP 和 SIPS URI 也可以放在 web 页面、电子邮件消息或印刷品中。它们包含足够的信息，用以发起和维护与资源通信的会话。

通信资源的实例包括以下内容：

- 在线服务的用户
- 多线电话的外观
- 消息系统的邮箱
- 网关服务的 PSTN 号
- 一个组织中的一个小组（如“销售部”或“问询台”）

SIPS URI 指明应安全地访问资源。特别地，这表明应在 UAC 和拥有 URI 的域间使用 TLS。安全通信用来从这里到达用户，其中特定的安全机制取决于域的策略。如果需要与资源进行安全的通信，任何由 SIP URI 描述的资源都可以通过更改模式“升级”到 SIPS URI。

19.1.1 SIP 和 SIPS URI 组件

“sip:”和“sips:”模式遵循 RFC2396[5]中的指导原则。它们使用一种与发邮件到 URL 类似的方式，允许指定 SIP 请求头字段和 SIP 消息体。这使得可以指定通过使用 web 页面或电子邮件信息中的 URI 发起的主题、媒体类型或会话的紧急性。第 25 章给出了 SIP 或 SIPS URI 的正式句法。SIP URI 的通用形式为：

sip:user:password@host:port;uri-parameters?headers

SIPS URI 的格式与此相同，除了模式是“sips”而非 sip 这一点不同以外。这些标记以及它们的扩展标记意义如下：

User:正在访问的主机上特殊资源的标识符。术语“主机”在这个上下文中常指一个域。URI 的 Userinfo 由这个 user 字段、password 字段以及跟随在它们后面的@标记组成。URI 的 Userinfo 部分是可选的，当目的主机不知道用户或当主机本身是正在被鉴别的资源时，它可以不出现。如果 SIP 或 SIPS URI 中有@标记，那么 user 字段不能为空。

如果所访问的主机可以处理电话号码，例如 Internet 电话网关，那么 RFC2806 中定义的电话用户字段可用来填充 user 字段。SIP 和 SIPS URI 中对电话用户字段编码的特定转义规则在第 19.1.2 节有述。

Password:密码与用户息息相关。虽然 SIP 和 SIPS URI 句法中存在这个字段，但是不推

荐使用这个字段。因为我们已经证明，几乎在每一种使用它的情况下，用明文（如 URI）传递认证信息实在是对安全的一大冒险。例如，在这个字段中传输 PIN 号便公开了 PIN。

请注意，password 字段仅仅是 user 部分的扩展。不希望特别重视字段 password 部分的实现可以简单地将“user:password”作为单一的字符串来处理。

Host:提供 SIP 资源的主机。主机部分包括一个完全限定域名或许多 Ipv4 或 Ipv6 地址。在任何可能的情况下，都推荐使用完全限定域名形式。

Port：将要发送请求的端口号。

URI 参数：影响 URI 构造的请求的参数。

URI 参数添加在主机端口组件之后，用分号隔开。

URI 参数的形式为：

parameter-name = parameter-value

即使 URI 中可以包含任意数量的 URI 参数，但是任何给定的参数名称不能出现二次。

可扩展的机制包括 transport、maddr、ttl、user、method 和 lr 参数。

传输参数确定用于发送 SIP 消息的传输机制，如在[4]中说明的那样。SIP 可以使用任何网络传输协议。定义了 UDP（RFC768[13]）、TCP（RFC761[14]）和 SCTP（RFC2960[15]）的参数名称。对 SIPS URI 而言，transport 参数必须表示一个可靠的传输。

Maddr 参数表示这个用户要联系的服务器的地址，它覆盖任何从 host 字段派生的地址。当有 maddr 参数时，URI 的 port 和 transport 组件将应用到 maddr 参数值表示的地址中。[4]给出了为了获得用于发送请求的目标地址、端口和传输协议的 transport、maddr 和 hostport 的适当的解释。

Maddr 字段已经用作松散源路由的简单形式。它允许 URI 指定在到达目的地的路径中必须经历的代理。不建议继续以这种方式使用 maddr 参数（反对使用支持它的机制）。而实现则应使用本文中描述的 Route 机制，如果需要的话，要建立一个已有的路由集（见第 8.1.1 节）。这里提供了一个完整的 URI，用于描述将会经历的节点。

Ttl 参数确定 UDP 多播包的使用期限值，并且必须只用在 maddr 是多播地址且传输协议是 UDP 的情况下。例如，要指定对 alice@atlanta.com 的呼叫，就要使用 ttl 值为 15 的 239.255.255.1 多播，并使用以下 URI：

sip:alice@atlanta.com;maddr=239.255.255.1;ttl=15

有效的 telephone-subscriber 字符串是有效用户字符串的子集。用户 URI 参数用于区别电话号码与那些看起来像电话号码的用户名。如果用户字符串包含一个格式为 telephone-subscriber 的电话号码，那么应当有 user 参数值“phone”。即便没有这个参数，如果用户名称空间的本地限制允许的话，SIP 与 SIPS URI 的接收方可以将@前面的部分解释为电话号码。

从 URI 构造 SIP 请求的方法可用 method 参数指定。

lr 参数（当存在的时候），表示负责资源的元素实现了本文档中指定的路由机制。这个参数将用在放在 Record-Route 头字段值中的 URI 代理中，并可能在已有的路由集中的 URI 中显示。

这个参数用于实现与这样一些系统的向后兼容性：这些系统实现了 RFC2543 中和到 bis-05 的 rfc2543bis 草案中的严格路由机制。准备发送一个基于 URI（不包含 lr 参数）的请求的元素可以假定接收元素实现了严格路由并重新格式化消息，以保存 Request-URI 中信

息。

由于 uri 参数机制是可扩展的，因此 SIP 元素必须忽略任何它们不懂的 uri 参数。

Header：头字段包含在用 URI 构造的请求中

SIP 请求中的头字段可以在 URI 中用“？”机制来指定。头的名称和值在 ampersand separated hname = hvalue pairs 中编码。特别的 hname“体”表示相联系的 hvalue 是 SIP 请求的消息体。

根据 URI 所在的上下文，表一总结了 SIP 和 SIPS URI 组件的用法。外面的一列描述了显示在 SIP 消息外任何地方（如在 web 页面或名片上）的 URI。以“m”为标记的项是必需的，以“o”为标记的项是可选的，而以“－”为标记的则是不允许的。处理 URI 的元素应当忽略即使存在但不允许的任何组件。第二列表示可选元素的缺省值（如果其值不存在的话)。“－”表示这个元素要么是不可选的，要么没有缺省值。

根据头字段所在的上下文，Contact 头字段中的 URI 具有不同的限制。其中一组适用于建立并维护对话（INVITE 及其 200（OK）响应）的消息。另一组适用于消息（REGISTER，其 200（OK）响应和对任何方法的 3xx 类响应）的注册与重定向。

					dialog reg./redir.	Contact/
	default	Req.-URI	To	From	Contact	R-R/Route	external
user	--	o	o	o	o	o	o
password	--	o	o	o	o	o	o
host	--	m	m	m	m	m	m
port	(1)	o	-	-	o	o	o
user-param	ip	o	o	o	o	o	o
method	INVITE	-	-	-	-	-	o
maddr-param	--	o	-	-	o	o	o
ttl-param	1	o	-	-	o	-	o
transp.-param	(2)	o	-	-	o	o	o
lr-param	--	o	-	-	-	o	o
other-param	--	o	o	o	o	o	o
headers	--	-	-	-	o	-	o

(1)：缺省端口值与传输和模式有关。使用 UDP、TCP 或 SCTP 时，Sip：的缺省值为 5060。使用 TCP 上的 TLS 时，sip：的缺省值为 5061；使用 TCP 上 TLS 时 sips:的缺省值为 5061。

(2)：缺省传输协议与模式有关。sip:的缺省传输协议是 UDP；sips:的缺省传输协议是 TCP。

表 1：SIP 头字段值、Request-URI 和引用的 URI 组件的用法与缺省值。

定义 SIP URI 中必须转义的字符集时，SIP 遵循 RFC 2396 [5]的需求和指导原则并使用

其“escaping”。以下内容出自 RFC 2396 [5]：

实际上在任何给定的 URI 组件中保留的字符集是由这个组件来定义的。一般说来，如果一个字符由其转义 US-ASCII 编码[5]替代了，使 URI 的语义改变，那么就保留这个字符。外部 US-ASCII 字符（RFC 2396 [5]），如空格和控制字符以及用作 URI 定界符的字符，也必须要转义。URI 不能包含未转义的空格和控制字符。

对于每个组件来说，有效的 BNF 扩展集明确定义了哪些字符可以显示为未转义。所有其他的字符必须要转义。

例如，“@”不是用户组件的字符集中的字符，因此用户“j@s0n”必须至少将@符号编码，如在“j 中。

对第 25 章中的 hname 和 hvalue 标记的扩展说明了头字段名称和值中所有 URI 保留的字符都必须转义。

用户组件的电话用户子集具有特定的转义注意事项。在 RFC 2806 [8]的电话用户描述中没有保留的字符集包含各种语法元素中的许多字符，这些字符在 SIP URI 中使用时要进行转义。出现在电话用户中而没有出现在用户规则的 BNF 扩展中的字符必须转义。

请注意，在 SIP 或 SIPS URI 的主机组件中不允许进行字符转义（这种情况在将来国际域名定稿后可能会改变）。当前的实现不能试图通过将主机组件中接收到的转义符作为它们的未转义的副本处理以提高健壮性。满足 IDN 需求的行为会有很大的不同。

19.1.2 SIP 和 SIP URI 实例

```
sip:alice@atlanta.com

sip:alice:secretword@atlanta.com;transport=tcp

sips:alice@atlanta.com?subject=project%20x&priority=urgent

sip:+1-212-555-1212:1234@gateway.com;user=phone

sips:1212@gateway.com

sip:alice@192.0.2.4

sip:atlanta.com;method=REGISTER?to=alice%40atlanta.com

sip:alice;day=tuesday@atlanta.com
```

上述最后一个实例 URI 有一个“alice;day=Tuesday”的用户字段值。上面定义的转义规则允许在这个字段中出现未转义的分号。对本协议而言，这个字段并没有解释清楚。这个字段值的结构仅仅对负责资源的 SIP 元素有用。

19.1.3 URI 比较

本规范中的某些操作需要确定两个 SIP 或 SIPS URI 是否相等。本规范中，注册服务器需要比较 REGISTER 请求的 Contact URI 中的绑定（请参阅第 10.3 节）。根据以下规则对 SIP 和 SIP URI 进行等同性比较：

- SIP 和 SIP URI 从不相等。
- 对 SIP 和 SIPS URI 的 userinfo 的比较是区分大小写的。包括包含密码的 userinfo 或作为电话用户进行格式化的用户信息。除非另有明确定义，URI 中所有其它组件的比较都是区分大小写的。
- 参数与头字段的顺序在比较 SIP 和 SIPS URI 时无足轻重。

- 除“保留”字符集（请参阅 RFC 2396 [5]）外，字符集与其“编码”相等。
- 主机名的 DNS 查找的结果 IP 地址与这个主机名不匹配。
- 欲使两个 URI 相等，用户、密码和端口组件必须要匹配。

省略用户组件的 URI 不会与一个包含用户组件的 URI 相匹配。省略密码组件的 URI 不会与包含密码组件的 URI 匹配。

省略任何具有缺省值的组件的 URI 将与明确包含这个带有缺省值的组件的 URI 不匹配。例如，一个省略可选端口组件的 URI 与一个明确声明端口为 5060 的 URI 不匹配。对于 transport 参数、ttl 参数、user 参数和方法组件来说，情况也是如此。

定义 sip:user@host 与 sip:user@host:5060 不相等，这是本规范对 RFC2543 的一个改变。当从 URI 派生地址时，相等的地址可能从相等的 URI 派生出来。URI sip:user@host:5060 总是对端口 5060 进行解析。URI sip:user@host 可能会通过[4]中详述的 DNS SRV 机制对其它端口进行解析。

以下是对 URI 的 uri 参数的比较：

- 两个 URI 中出现的任何 uri 参数都必须要匹配。
- 只在一个 URI 中出现的 User、ttl 或 uri 参数不会与其它内容匹配，即使它包含缺省值也不会匹配。
- 包含 maddr 参数的 URI 与不包含 maddr 参数的 URI 不匹配。
- 在比较 URI 时，可以忽略只在一个 URI 中出现的所有其它 uri 参数。
- 从不忽略 URI 头组件。任何现有的头组件必须在 URI 和将要与这些 URI 匹配的 URI 中。第 20 章定义了每个头字段的匹配规则。

下面每组中的 URI 都是相等的：

sip:%61lice@atlanta.com;transport=TCP

sip:alice@AtLanTa.CoM;Transport=tcp

sip:carol@chicago.com

sip:carol@chicago.com;newparam=5

sip:carol@chicago.com;security=on

sip:biloxi.com;transport=tcp;method=REGISTER?to=sip:bob%40biloxi.com

sip:biloxi.com;method=REGISTER;transport=tcp?to=sip:bob%40biloxi.com

sip:alice@atlanta.com?subject=project%20x&priority=urgent

sip:alice@atlanta.com?priority=urgent&subject=project%20x

下面每组中的 URI 是不相等的：

SIP:ALICE@AtLanTa.CoM;Transport=udp （用户名不同）

sip:alice@AtLanTa.CoM;Transport=UDP

sip:bob@biloxi.com （能解析成不同的端口）

sip:bob@biloxi.com:5060

sip:bob@biloxi.com （能解析成不同的传输协议）

sip:bob@biloxi.com;transport=udp

sip:bob@biloxi.com （能解析成不同的端口和传输协议）

sip:bob@biloxi.com:6000;transport=tcp

sip:carol@chicago.com （不同的头组件）

sip:carol@chicago.com?Subject=next%20meeting

sip:bob@phone21.boxesbybob.com （即使这是 phone21.box

sip:bob@192.0.2.4 esbybob.com 进行的解析）

请注意等同性是不可传递的：

- sip:carol@chicago.com

与 sip:carol@chicago.com;security=on 相等

- sip:carol@chicago.com

与 sip:carol@chicago.com;security=off 相等

- sip:carol@chicago.com;security=on

与 sip:carol@chicago.com;security=off 不相等

19.1.4 从 URI 构造请求

实现需要确定什么时候直接从 URI 构造请求。名片的 URI、web 页面的 URI 甚至协议中的资源（如已注册的联系）的 URI 都可能包含不恰当的头字段或体部分。

实现必须在构造（honor）的请求的 Request-URI 中包含任何提供的 transport、maddr、ttl 或 user 参数。如果这个 URI 包含 method 参数，那么其值必须作为这个请求的方法。method 参数不能放在 Request-URI 中。未知的 URI 参数必须要放入消息的 Request-URI 中。

实现应当使 URI 中的任何头或体部分的状态都包含在消息中，并选择基于每一个组件来构造请求。

实现不应当构造这些明显危险的头字段：From、Call-ID、Cseq、Via 和 Record-Route。

为了不在恶意攻击中被用作不知情的代理，实现不应当构造任何所请求的 Route 头字段值。

实现不应当构造包含这样一些头字段的请求：这些头字段可能错误地公开其位置或功能。这些头字段包括：Accept（接受）、Accept-Encoding（接受编码）、Accept-Language（接受语言）、Allow（允许）、Contact（联系人）（在其对话的使用中）、Organization（组织）、 Supported（受到支持）和 User-Agent（用户代理）。

实现应当验证任何所请求的描述性头字段的准确性，这些头字段包括：Content-Disposition（内容处理）、Content-Encoding（内容编码）、Content-Language（内容语言）、Content-Length（内容长度）、Content-Type（内容类型）、Date（日期）、Mime-Version 和 Timestamp（时间戳）。

如果通过从给定的 URI 构造消息来构造的请求不是有效的 SIP 请求，那么这个 URI 也是无效的。实现不应继续传输这个请求。它应当遵循无效 URI 的上下文中的行为准则。

所构造的请求的无效体现在许多方面，包括（但不限于）头字段中的句法错误、URI 参

数的无效组合或对消息体的不正确描述。

发送一个从给定的URI构造的请求可能需要一些实现无法提供的功能。例如，这个URI可能指出怎样使用未实现的传输或扩展。实现应当拒绝发送这些请求，而不能修改它们以匹配它们的功能。如果实现不支持一个请求需要的扩展，那么它不能发送这个请求。

例如,这样一个请求可以通过Require头参数的状态或带有未知或明确不支持的值的方法URI参数构成。

19.1.5 将SIP URI和tel URL相关联

当一个tel URL（RFC 2806 [8]）转化为SIP或SIPS URI时，tel URL的整个电话用户部分（包括任何参数）可以放入SIP或SIPS URI的userinfo部分。

因此，tel:+358-555-1234567;postd=pp22 变为

sip:+358-555-1234567;postd=pp22@foo.com;user=phone

或sips:+358-555-1234567;postd=pp22@foo.com;user=phone

而不是sip:+358-555-1234567@foo.com;postd=pp22;user=phone

或

sips:+358-555-1234567@foo.com;postd=pp22;user=phone

通常情况下，以这种方式将相等的“tel”URL 转换成SIP或SIPS URI可以不产生相等的SIP或SIPS URI。SIP和SIPS URI的userinfo是作为区分大小写的字符串来比较的。Tel URL的不区分大小写部分的变化与tel URL参数的重排序不影响tel URL的等同性，但是会影响从它们构造SIP URI的等同性。

例如：

tel:+358-555-1234567;postd=pp22

tel:+358-555-1234567;POSTD=PP22

是相等的，而

sip:+358-555-1234567;postd=pp22@foo.com;user=phone

sip:+358-555-1234567;POSTD=PP22@foo.com;user=phone

却不相等。

同样，

tel:+358-555-1234567;postd=pp22;isub=1411

tel:+358-555-1234567;isub=1411;postd=pp22

是相等的，而

sip:+358-555-1234567;postd=pp22;isub=1411@foo.com;user=phone

sip:+358-555-1234567;isub=1411;postd=pp22@foo.com;user=phone

是不相等的。

要解决这个问题，构造电话用户字段以将其放入SIP或SIPS URI的userinfo部分的元素应当将电话用户的不区分大小写的部分转化为小写,并通过参数名称将电话用户参数按词典序排序，除了在此顺序里最先出现的isdn-subaddress和post-dial以外。（除以后的扩展参数外，tel URL的所有组件都定义成可不区分大小写的比较。）

遵循这个建议，

tel:+358-555-1234567;postd=pp22 和

tel:+358-555-1234567;POSTD=PP22

都将成为

sip:+358-555-1234567;postd=pp22@foo.com;user=phone

而

tel:+358-555-1234567;tsp=a.b;phone-context=5 和

tel:+358-555-1234567;phone-context=5;tsp=a.b

都将成为

sip:+358-555-1234567;phone-context=5;tsp=a.b@foo.com;user=phone

19.2 可选标记

可选标记是在 SIP 中指定新选项（扩展）的一些独特的标识符。这些标记用于 Require（第 20.32 节）、Proxy-Require（第 20.29 节）、Supported（第 20.37 节）和 Unsupported（第 20.40 节）的头字段中。请注意这些选项是作为参数以 option-tag = token 的形式出现在那些头字段中的。（欲了解标记的定义，请参阅第 25 章）

在标准跟踪 RFC 中定义了可选标记。这是对过去实践的一大改变，定义它们是为了确保连续的多厂商互操作性（请参阅第 20.32 节与第 20.37 节的讨论）。将可选标记注册到 IANA 可确保轻而易举地对其进行引用。

19.3 标记

在 SIP 消息的 To 和 From 头字段中使用 tag 参数。它是鉴别对话的通用机制，这个机制是将 Call-ID 与两个标记（每个标记来自对话的一个参与方）的组合。当 UA 在对话外发送请求时，它只包含一个 From 标记，并提供对话 ID 的“一半”。这个对话在收到各个响应后完成，每个响应都组成了 To 头字段的另一半。分发 SIP 请求意味着多个对话可以从一个单一的请求建立。这也解释了双方对话标识符的必要性，如果没有接收方的作用，发送方无法去除从一个请求建立多个对话的歧义。

当 UA 产生一个标记并插入到请求或响应中时，该标记必须是全球唯一的，并且必须至少使用 32 位随机数进行随机加密。这个选择需求的特性是指 UA 把一个不同的标记放入 INVITE 的 From 头中，而非把该标记放入相同 INVITE 的响应的 To 头字段中。为了使 UA 邀请其自身加入会话（这是在 PSTN 网关中对呼叫进行组合的一个普通实例），这是必需的。同样，不同呼叫的两个 INVITE 会具有不同的 From 标记，不同呼叫的两个响应则具有不同的 To 标记。

除了全球唯一性需求外，生成标记的算法是实现特有的。在容错系统中，在备用服务器上恢复失败的对话时，标记会起到帮助作用。通过 UAS 选择标记的这种方式，备份服务器能识别出请求是失败的服务器上的对话的一部分，因此，UAS 能确定应尝试恢复对话和与对话相关的任何其它状态。

20 头字段

已经在第 7.3 节说明了头字段通用的语法。本章列出了所有头字段及其语法的注释、意义和用法。本章中，我们使用[HX.Y]代表当前 HTTP/1.1 标准 RFC2616[7]的 X.Y 节。给出了每个头字段的实例。

在表 2 和表 3 中概括了与方法和代理处理有关的头字段的信息。

“where”列描述了在其中使用头字段的请求和响应类型。该列的值如下：

R：仅仅可以出现在请求中的头字段；

r：仅仅可以出现在响应中的头字段；

2xx,4xx,etc.:多个值和范围指明与头字段一起使用的响应代码；

c：从请求中复制到响应的头字段

- 在“where”列中的空项表示头字段可以在所有请求和响应中出现。

“proxy”列描述了代理可以在头字段上执行的操作：

a：如果不存在，代理能够添加或者连接头字段。

m：代理能够修改现有的头字段的值。

d：代理能够删除头字段的值。

r：代理必须能够读取头字段的值，因此该头字段的值不能加密。

下面的 6 列是与方法中的头字段的存在有关：

c：有条件的；在头字段的需求与消息的环境有关。

m：此头字段是强制的，必须有的。

m*：应该发送此头字段，但是，在没有该头字段的情况下，客户端/服务器需要准备接收消息。

o：此头字段是可选的。

t：应该发送此头字段，但是，在没有该头字段的情况下，客户端/服务器需要准备接收消息。

如果使用基于流的协议（比如 TCP）作为传输协议，那么必须发送此头字段。

*：如果消息体非空，此头字段是必需的。详情参见第 20.14、第 20.15 和第 7.4 节。

-：头字段是不可用的。

“Optional”的意思是在请求或者响应中元素可以包括头字段，并且第如果在请求或者响应中出现，UA 可以忽略该头字段（在第 20.32 节中讨论的 Require 头字段是该规则的特例）。“mandatory”头字段必须在请求中出现，并且接收请求的 UAS 必须理解该“mandatory”头字段。mandatory 响应头字段必须在响应中出现，并且处理响应的 UAC 必须理解该头字段。“Not applicable”的意思是，头字段不能在请求中出现。如果一个头字段错误地出现在请求中，接收请求的 UAS 必须忽略它。同样的，为响应标记了“not applicable”的头字段，表示 UAS 不能在响应中设置该头字段，UAC 必须在响应中忽略该头字段。

UA 将忽略不理解的扩展的头参数。

还定义了一些通用头字段名称的简写形式，用于当消息的大小成为问题的时候。

Contact、From 和 To 头字段包含了 URI。如果 URI 包含了逗号、问号或者分号，必须在

尖括号中包含该 URI（<and>）。在这些括号中包含任何 URI 参数。如果 URI 没有包括在尖括号中，任何以分号分隔的参数是头参数，而不是 URI 参数。

20.1 Accept

Accept 头字段遵循在[H14.1]中定义的语法。语义也是这样，除非不存在 Accept 头字段，服务器将假定 application/sdp 的默认值。

空 Accept 头字段表示没有可接受的格式。

实例：

Accept：application/sdp;level=1,application/x-private,text/html

20.2 Accept-Encoding

Accept-Encoding 头字段同 Accept 头字段一样，但是限制了在响应中可以接受的内容-代码[H3.5]。参见[H14.3]。在 SIP 中的语义与那些在[H14.3]中定义的语义一样。允许空的 Accept-Encoding 头字段。这等效于 Accept-Encoding：identity，也就是 identity encoding，表示没有编码。

如果没有 Accept-Encoding 头字段出现，服务器将假定 identity 的默认值。

Header	field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Accept	R	-	o	-	o	m*	o		
Accept	2xx	-	-	-	o	m*	o		
Accept	415	-	c	-	c	c	c		
Accept-Encoding	R	-	o	-	o	o	o		
Accept-Encoding	2xx	-	-	-	o	m*	o		
Accept-Encoding	415	-	c	-	c	c	c		
Accept-Language	R	-	o	-	o	o	o		
Accept-Language	2xx	-	-	-	o	m*	o		
Accept-Language	415	-	c	-	c	c	c		
Alert-Info	R	ar	-	-	-	o	-	-	
Alert-Info	180	ar	-	-	-	o	-	-	
Allow	R	-	o	-	o	o	o		
Allow	2xx	-	o	-	m*	m*	o		

Allow	r	-	o	-	o	o	o		
Allow	405	-	m	-	m	m	m		
Authentication-Info	2xx	-	o	-	o	o	o		
Authorization	R	o	o	o	o	o	o		
Call-ID	c	r	m	m	m	m	m	m	
Call-Info	ar	-	-	-	o	o	o		
Contact	R	o	-	-	m	o	o		
Contact	1xx	-	-	-	o	-	-		
Contact	2xx	-	-	-	m	o	o		
Contact	3xx	d	-	o	-	o	o	o	
Contact	485	-	o	-	o	o	o		
Content-Disposition	o	o	-	o	o	o			
Content-Encoding	o	o	-	o	o	o			
Content-Language	o	o	-	o	o	o			
Content-Length	ar	t	t	t	t	t	t		
Content-Type	*	*	-	*	*	*			
CSeq	c	r	m	m	m	m	m	m	
Date	a	o	o	o	o	o	o		
Error-Info	300-699	a	-	o	o	o	o	o	
Expires	-	-	-	o	-	o			
From	c	r	m	m	m	m	m	m	
In-Reply-To	R	-	-	-	o	-	-		
Max-Forwards	R	amr	m	m	m	m	m	m	
Min-Expires	423	-	-	-	-	-	m		
MIME-Version	o	o	-	o	o	o			
Organization	ar	-	-	-	o	o	o		

表 2：头字段概述，A－O

Header	field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Priority	R	ar	-	-	-	o	-	-	
Proxy-Authenticate	407	ar	-	m	-	m	m	m	
Proxy-Authenticate	401	ar	-	o	o	o	o	o	
Proxy-Authorization	R	dr	o	o	-	o	o	o	
Proxy-Require	R	ar	-	o	-	o	o	o	
Record-Route	R	ar	o	o	o	o	o	-	
Record-Route	2xx,18x	mr	-	o	o	o	o	-	
Reply-To	-	-	-	o	-	-			
Require	ar	-	c	-	c	c	c		
Retry-After	404,413,480,486	-	o	o	o	o	o		
500,503	-	o	o	o	o	o			
600,603	-	o	o	o	o	o			
Route	R	adr	c	c	c	c	c	c	
Server	r	-	o	o	o	o	o		
Subject	R	-	-	-	o	-	-		
Supported	R	-	o	o	m*	o	o		
Supported	2xx	-	o	o	m*	m*	o		
Timestamp	o	o	o	o	o	o			
To	c(1)	r	m	m	m	m	m	m	
Unsupported	420	-	m	-	m	m	m		
User-Agent	o	o	o	o	o	o			
Via	R	amr	m	m	m	m	m	m	
Via	rc	dr	m	m	m	m	m	m	
Warning	r	-	o	o	o	o	o		
WWW-Authenticate	401	ar	-	m	-	m	m	m	

WWW-Authenticate	407	ar	-	o	-	o	o	o	

表 3：头字段概述，P－Z；(1)：从其它可能的标签中复制

这同 HTTP 定义稍有不同，HTTP 定义指定当不存在 Accept-Encoding 头字段时，能够使用任何编码，但是首选 identity 编码。

实例:

Accept-Encoding：gzip

20.3 Accept-Language

在请求中使用 Accept-Language 头字段指定首选语言，该首选语言用于原因分析、会话描述或者在响应中携带的作为消息体的状态响应。如果不存在 Accept-Language 头字段，服务器将假定客户端接受所有语言。

Accept-Language 头字段遵循在[H14.4]中定义的语法。语言排序的规则是基于用于 SIP 的“q”参数。

实例：

Accept-Language：da,en-gb;q=0.8,en;q=0.7

20.4 Alert-Info

当在 INVITE 请求中出现时,Alert-Info 头字段为 UAS 指定了一个替换响铃音。当在 180（响铃）响应中出现时，Alert-Info 头字段为 UAC 指定了一个替换响铃音。该头字段的典型用法是代理插入该头字段以提供与众不同的响铃特征。Alert-Info 头字段会引入安全风险。这些风险和处理它们的方式将在第 20.9 讨论，因为面临同样的风险，在该节中还讨论了 Call-Info 头字段。

另外，用户应该能够选择禁用该特征。

这将有助于阻止来自非信任元素使用该字段造成的中断。

实例：

Alert-Info：<http://www.example.com/sounds/moo.wav>

20.5 Allow

Allow 头字段列出了一系列 UA 产生消息所支持的方法。

所有被 UA 理解的方法，包括 ACK 和 CANCEL，当出现时，必须包含在 Allow 头字段的方法列表中。没有 Allow 头字段不应该解释为 UA 发送的消息不支持方法。相反的，它的意思是 UA 没有提供任何支持方法的信息。

实例：

Allow：INVITE,ACK,OPTIONS,CANCEL,BYE

20.6 Authentication-Info

Authentication-Info 头字段规定了带 HTTP 摘要的相互认证。UAS 可以在请求的 2xx 响应中包括该头字段，使用基于 Authorization 头字段的摘要成功认证了该头字段。

语法和语义遵循在 RFC2617[16]中的规定。

实例：

Authentication-Info：nextnonce="47364c23432d2e131a5fb210812c"

20.7 Authorization

Authorization 头字段包含了 UA 的认证凭证。第 22.2 节概述了 Authorization 头字段的用法，并且在第 22.4 节中描述了它和 HTTP 认证一起使用时的语法和语义。

该头字段，与 Proxy-Authrization 一起，打破了多头字段值的通用规则。尽管不是逗号分隔的列表，该头字段名称可以多次出现，但是不能使用在第 7.3 节中描述的通用规则将其组成单头行。

在下面的实例中，没有引用 Digest 参数。

Authorization：Digestusername="Alice",realm="atlanta.com",

nonce="84a4cc6f3082121f32b42a2187831a9e",

response="7587245234b3434cc3412213e5f113a5432"

20.8 Call-ID

Call-ID 头字段唯一地识别具体的邀请或者所有具体的客户端的注册。单一多媒体会议能够用不同的 Call-ID 发起若干呼叫，例如，如果用户多次邀请单用户到同一（远程）会议。Call-ID 是区分大小写并且逐个字节进行简单比较。

Call-ID 头字段的简写形式是 I。

实例：

Call-ID：f81d4fae-7dec-11d0-a765-00a0c91e6bf6@biloxi.com

i:f81d4fae-7dec-11d0-a765-00a0c91e6bf6@192.0.2.4

20.9 Call-Info

根据在请求或者响应中是否发现 Call-Info 头字段，Call-Info 头字段提供有关呼叫方和被呼叫方的附加信息。Purpose 参数描述 URI 的目的。icon 参数指定图像，该图像合适作为呼叫方和被呼叫方的形象代表。info 参数通常描述了呼叫方和被呼叫方，例如，通过网页。Card 参数提供了名片，例如，以 vCard[37]或者 LDIF[38]格式。使用在第 27 章讲到的 IANA 和处理器，能够注册附加标记。

使用 Call-Info 头字段会引入安全风险。如果被呼叫方收到恶意呼叫方提供的 URI，被呼叫方可能会有显示不正确的或者进攻性质的内容、危险或者是不合法的内容等等这样的风险。因此，如果 UA 能够验证发起头字段的元素的认证并且信任该元素，推荐 UA 仅仅在 Call-Info 头字段中提供信息。这不需要是对等的 UA；代理可以在请求中插入该头字段。

实例：

Call-Info：<http://wwww.example.com/alice/photo.jpg>

;purpose=icon,

<http://www.example.com/alice/>;purpose=info

20.10 Contact

Contact 头字段值提供了 URI，该 URI 的意义基于它所在的请求或者响应的类型。

Contact 头字段值包含了显示名称、带 URI 参数的 URI 和头参数。

本文定义了 Contact 参数 q 和 expires。这些参数仅用于当 Contact 出现在 REGISTER 请求或者响应中，或者在 3xx 响应中。其它的参数可以在其它规范中定义。

当头字段值包含了显示名称时，包含所有 URI 参数的 URI 会包括在“<”和“>”中。如果不存在“<”和“>”，所有在 URI 之后的参数是头参数，不是 URI 参数。如果希望能够标记显示名称或者引用字符串，那么需要大量的字符集合。

即使 display-name 是空，如果 addr-spec 包含了逗号、分号或者问号，必须使用 name-addr 的形式。在 display-name 和“<”之间可以是也不可以是 LWS。

这些解析显示名称、URI 和 URI 参数和头参数的规则，还用于 TO 和 From 头字段。

Contact 头字段与 HTTP 中定位头字段有相同的作用。而 HTTP 头字段仅仅允许一个未引用的地址。因为 URI 能够包含作为保留字符串的逗号和分号，会误认为它们是头或者参数分隔符。

Contact 头字段的简写形式是 m(“moved”)。

实例：

Contact：″Mr.Watson″<sip:watson@worcester.bell-telephone.com>

;q=0.7;expires=3600,

″Mr.Watson″<mailto:watson@bell-telephone.com>;q=0.1

m：<sips:bob@192.0.2.4>;expires=60

20.11 Content-Disposition

Content-Disposition 头字段描述了，UAC 或者 UAS 是怎样解释消息体或者一个消息体部分（对于多部分消息）。该 SIP 头字段扩展了 MIME Content-Type(RFC 2183 [17])。

SIP 定义了 Content-Disposition 头的若干新的 disposition-Types。session 值说明，消息体部分为呼叫或者早期（预呼叫）媒体描述了一个会话。render 值说明，将显示消息体部分或者否则向用户提交消息体部分。注意，使用 render 值而不是 inline 的目的是，避免显示 MIME 消息体作为整个消息提交的部分这一含义(因为 SIP 消息的 MIME 消息体通常不向用户显示)。为向后兼容，如果 Content-Disposition 头字段丢失，服务器将假定 Content-Typeapplication/sdp 的消息体是配置 session，而其它内容类型是 render。配置类型 icon 说明，消息体部分包含了适合作为呼叫方或者被呼叫方的形象代表的图像，当接收消息或者持续对话时，用户代理将以信息的方式发送该图像。Alert 的值说明，消息体部分包含了比如音频复制文件这样的信息，当用户代理试图警告用户接收请求，通常是发起对

话的请求时，用户代理将发送该信息；能够发送该警告消息体，比如发送该警告消息体可作为发送 180（响铃）临时响应后的电话呼叫的铃音。

仅仅在消息已经完全认证时，才处理向用户发送内容的任何带 disposition-type 的 MIME 消息体。

处理参数、处理-参量描述了，如果接收了没有理解其内容类型或者配置类型的消息体，UAS 将怎样反应。参数定义了 optional 和 required 值。如果处理参数丢失，将假定 required 值。处理参数将在 RFC 3204[18]中描述。

如果头字段丢失，MIME 类型确定默认的内容配置。如果没有，就假定为“render”。

实例：

Content-Disposition：session

20.12 Content-Encoding

使用 Content-Encoding 头字段作为 media-type 的修饰符。当存在时，它指明要对实体主体采用的附加内容译码方式，因而要获得 Content-Type 头字段中提及的媒体类型，必须采用与译码方式一致的解码机制。Content-Encoding 内容编码主要用来，在没有丢失它的基础媒体类型身份的情况下压缩消息体。

如果实体主体中应用了多种编码，内容编码必须按照应用它们的顺序列出。

所有的内容编码值是不区分大小写的。IANA 的作用是注册内容编码值标记。内容-代码的语法定义参见[H3.5]。

客户端可以将内容编码应用于请求中的消息体。服务器可以将内容编码应用于响应中的消息体。服务器只能使用在请求的 Accept-Encoding 头字段中列出的编码。

Content-Encoding 的简写形式是 e。实例：

Content-Encoding：gzip

e：tar

20.13 Content-Language

参见[H14.12]。实例：

Content-Language：fr

20.14 Content-Length

Content-Length 头字段表明消息-消息体的大小，以八位字节十进制数形式，向接收者发送。应用将使用该字段表示传送的消息-消息体的大小，而不管实体的媒体类型。如果基于流的协议（如 TCP）作为传输协议来使用，必须使用该头字段。

消息-消息体的大小不包括 CRLF 分隔的头字段和消息体。任何大于或者等于 0 的 Content-Length 都是有效的。如果在消息中不存在消息体，那么，Content-Length 头字段值必须设置为 0。

忽略 Content-Length 的能力简化了动态产生响应的、像 cgi 的脚本的创建。

该头字段的简写形式是 l。

实例：

Content-Length：349

l：173

20.15 Content-Type

Content-Type 头字段表示向接收者发送的消息体的媒体类型。在[H3.7]定义了“媒体类型”元素。如果消息体不是空的，必须有 Content-Type 头字段。如果消息体是空的，并且也有 Content-Type 头字段，则表示特定类型的消息体的长度是 0（例如，空的音频文件）。

该头字段的简写形式是 c。

实例：

Content-Type：application/sdp

c：text/html;charset=ISO-8859-4

20.16 Cseq

在请求中的 Cseq 头字段包括了一个单十进制序列号和请求方法。该序列号必须表示为 32 位无符号整数。Cseq 的方法部分是区分大小写的。Cseq 头字段作为在对话中的事务排序，以提供一种方法来唯一识别事务以及区分新的请求和请求重发。如果序列号和请求方法是一样的，则我们认为两个 Cseq 头字段是相等的。

实例：

CSeq：4711 INVITE

20.17 Date

Date 头字段包含了日期和时间。与 HTTP/1.1 不同，SIP 仅支持最新的 RFC1123[19]日期格式。正如在[H3.3]中，SIP 限制了在 SIP 日期的时区为“GMT”，而 RFC3261 允许任何时区。RFC1123 日期是区分大小写的。

Date 头字段反映了第一次发送请求或者响应的时间。

在没有电池供电的时钟情况下，简单终端系统可用 Date 头字段获取当前的时间概念。而，在它的 GMT 格式中，它需要客户端知道它们对 GMT 的偏移量。

实例：

Date：Sat,13 Nov 2010 23:29:00 GMT

20.18 Error-Info

Error-Info 头字段提供了指向错误状态响应这一附加信息的指针。

SIP UAC 拥有大容量的用户接口：从弹出式窗口和 PC 软客户端音频到音频“black”电话或者通过网关连接的终端。不是强迫服务器产生错误选择是发送带详细原因分析的错误状态代码还是播放音频记录，Error-Info 头字段允许同时发送两者。然后，UAC 选择向呼叫方发送哪一个错误指针。

UAC 可以在 Error-Info 头字段中处理 SIP 或者 SIPS URI，就好像它是重定向中的 Contact，并且产生了新的 INVITE，该 INVITE 的结果是建立了一个记录的通告会话。可以向用户传送非 SIP URI。

实例：

SIP/2.0 404 The number you have dialed is not in service

Error-Info：<sip:not-in-service-recording@atlanta.com>

20.19 Expires

Expires 头字段提供了有关的时间，在此时间之后，消息（或者内容）结束。

它的精确含义与方法有关。

INVITE 中的终止时间不影响由邀请引起的实际会话的持续时间。而会话描述协议可以表示会话持续时间的时间界限。

该字段的值是整数（十进制）秒，在 0 和（2**32）-1 之间，从接收请求开始计算。

实例：

Expires：5

20.20 From

From 头字段表示了请求的发起者。这可以与对话的发起者不同。在 From 头字段中，被呼叫方向呼叫方发送的请求使用被呼叫方的地址。

通过用户界面交付可选的 display-name。如果客户端的身份是隐藏的，系统将使用显示名称“Anomymous”。甚至如果 display-name 是空的，且 addr-spec 包含了逗号、问号或者分号，必须使用 name-addr 形式。将在第 7.3.1 节讨论语法问题。

如果两个 From 头字段的 URI 相匹配，并且它们的参数也匹配，则它们是等效的。为了比较，忽略在一个头字段中扩展的，在另一个头字段不存在的参数。这就表示了显示名称以及有或者没有尖括号都不会影响匹配。

显示名称、URI 和 URI 参数、和头字段参数的分析规则参见第 20.10 节。

From 头字段的简写形式是 f。

实例：

From：＂A.G.Bell＂<sip:agb@bell-telephone.com>;tag=a48s

From：sip:+12125551212@server.phone2net.com;tag=887s

f：Anonymous<sip:c8oqz84zk7z@privacy.org>;tag=hyh8

20.21 In-Reply-To

In-Reply-To 头字段列举了呼叫引用或者返回的 Call-ID。客户端可以存储这些 Call-ID，然后在返回呼叫的 In-Reply-To 头字段中包含这些 Call-ID。

它允许自动呼叫分配系统向第一个呼叫的发起者路由返回的呼叫。它还允许被呼叫方筛选呼叫，因此，将仅仅接受它们发起呼叫的返回呼叫。该字段不能代替请求认证。

实例：

In-Reply-To：70710@saturn.bell-tel.com,17320@saturn.bell-tel.com

20.22 Max-Forwards

Max-Forwards 头字段必须与 SIP 方法一起使用，以限制能够向下一个下游服务器转发请求的代理或者网关的数量。当客户端试图跟踪出现错误或者链中循环的请求链时，它也是非常有用的。

Max-Forwards 值是在 0 到 225 之间的整数，它表示允许转发该请求消息的剩余次数。转发请求的每个服务器减少该数字。推荐初始值为 70。

不能保证循环检测的元素应该插入该头字段。例如，B2BUA 应该插入 Max-Forwards 头字段。

实例：

Max-Forwards：6

20.23 Min-Expires

Min-Expires 头字段传送支持服务器管理的软状态元素的最小刷新间隔。它包含了注册服务器存储的 Contact 头字段。该头字段包含了从 0 到（2**32）-1 之间的十进制整数秒。在第 10.2.8、第 10.3 和第 21.4.17 节中介绍了该头字段在 423（时间间隔太短）响应中的用法。

实例：

Min-Expires：60

20.24 MIME-Version

参见[H19.4.1]。

实例：

MIME-Version：1.0

20.25 Organization

属于某组织的 SIP 元素发起请求或者响应，Organization 头字段传送该组织的名称。

客户端软件可以使用该字段筛选呼叫。

实例：

Organization：Boxes by Bob

20.26 Priority

Priority 头字段表示当客户端发现请求时请求的迫切性。Priority 头字段描述了 SIP 请求应有的向人或者代理发送的优先权。例如，它可以成为呼叫路由和呼叫接受的一个决定

因素。对于这些决定因素，规定没有包含 Priority 头字段的消息的 Priority 是 normal。Priority 头字段不影响使用通信资源，比如在路由器中分组转发的优先权或者在 PSTN 网关中的接入电路。该头字段值可以为 non-urgent、normal、urgent 和 emergency，但是在别的地方定义了字段的其它值。推荐仅在生命、身体或者财产即将受到威胁的时候使用 emergency 值。否则，没有为该字段定义语义。

除了 RFC2076[39]值外，附加了“emergency”。

实例：

Subject：A tornado is heading our way!

Priority：emergency

or

Subject：Weekend plans

Priority：non-urgent

20.27 Proxy-Authenticate

Proxy-Authenticate 头字段值包含了认证挑战。

在[H14.33]中定义了该头字段的用法。它的更加详细的用法参见第 22.3 节。

实例：

Proxy-Authenticate：Digestrealm="atlanta.com",

domain="sip:ss1.carrier.com",qop="auth",

nonce="f84f1cec41e6cbe5aea9c8e88d359",

opaque="",stale=FALSE,algorithm=MD5

20.28 Proxy-Authorization

Proxy-Authorization 头字段允许客户端向需要认证的代理（或者其用户）识别自己。Proxy-Authorization 头字段值由包含用户代理的认证信息的凭证组成，这些用户代理的认证信息用于代理和/或被请求资源的域。

该头字段用法的定义参见第 22.3 节。

该头字段与 Authorization 一起，打破了多种头字段名称的通用规则。尽管没有逗号分隔列表，该头字段名称可以出现多次，但是不能使用第 7.3.1 节中描述的通用规则组成单个头行。

实例：

Proxy-Authorization：Digestusername="Alice",realm="atlanta.com",

nonce="c60f3082ee1212b402a21831ae",

response="245f23415f11432b3434341c022"

20.29 Proxy-Require

使用 Proxy-Require 头字段表示代理必须支持的代理敏感的特征。该消息的更加详细机

制和用法实例，参见第 20.32 节。

实例：

Proxy-Require：foo

20.30 Record-Route

代理将 Record-Route 头字段插入到请求中，强迫在对话中的未来的请求都通过代理路由。

在第 16.12.1 节中介绍了该头字段的用法实例和 Route 头字段。

实例：

Record-Route: <sip:server10.biloxi.com;lr>,

<sip:bigbox3.site3.atlanta.com;lr>

20.31 Reply-To

Reply-To 头字段包含了与 From 头字段不同的逻辑返回 URI。例如，可以使用 URI 返回丢失的呼叫或者未建立的会话。如果用户希望保持匿名，将在请求中忽略该头字段，或者用不显示任何隐私信息的方法填充该头字段。

即使如果 display-name 是空的，且 addr-spec 包含了逗号、问号或者分号，必须使用 name-addr 形式。在第 7.3.1 节中讨论了语法问题。

实例：

Reply-To: Bob <sip:bob@biloxi.com>

20.32 Require

UAC 使用 Require 头字段通知 UAS——有关 UAC 为处理请求期望 UAS 支持的选项。尽管是一个可选的头字段，如果存在，就不能忽略 Require。

第 19.2 节中介绍了，Require 头字段包含可选标签的列表。每个可选标签定义了处理请求必须理解的 SIP 扩展。常常，这用来表示需要理解的具体的扩展头字段。遵循本规范的 UAC 必须仅仅包括符合标准跟踪 RFC 的选项标签。

实例：

Require: 100rel

20.33 Retry-After

Retry-After 头字段可以与 500（服务器内部错误）或者 530（不能提供服务）响应一起使用，以说明对请求客户端不能提供的服务预计持续多长时间，并且该头字段还可以与 404（没有发现）、413（请求实体太大）、480（暂时不可用）、486（现在正忙）、600（忙碌）或者 603（拒绝）响应一起使用，以说明能够重新访问被呼叫方的时间。该头字段值是响应时间之后的正整数秒(十进制)。

可选注释可以用来说明回叫时间的附加信息。可选的 duration 参数说明从可用的初始

时间开始到到达被呼叫方的持续时间。如果没有提供持续时间这一参数，就假定可无限期地使用服务。

实例：

Retry-After: 18000;duration=3600

Retry-After: 120 (I’ m in a meeting)

20.34 Route

Route 头字段用于强迫通过列出的代理路由请求。Route 头字段用法的实例参见第16.12.1 节。

实例：

Route: <sip:bigbox3.site3.atlanta.com;lr>,

<sip:server10.biloxi.com;lr>

20.35 Server

Server 头字段包含了 UAS 为处理请求而使用的软件的信息。

显示服务器的特定软件版本可能使服务器更加容易受到有安全漏洞的软件的攻击。实现者应该使 Server 头字段成为可控制的选项。

实例：

Server: HomeServer v2

20.36 Subject

Subject 头字段提供了摘要或者说明了呼叫的种类，在没有分析会话描述的情况下，允许筛选呼叫。会话描述不需要使用与邀请相同的主体标志。

Subject 头字段的简写形式是 s。

实例：

Subject: Need more boxes

s: Tech Support

20.37 Supported

Supported 头字段列举了 UAC 和 UAS 支持的所有扩展。

第 19.2 节中介绍了，Supported 头字段包含 UAC 或者 UAS 理解的可选标签列表。遵循本规范的 UA 必须仅包含符合标准跟踪 RFC 的可选标签。如果为空，表示不支持扩展。

Supported 头字段的简写形式是 k。

实例：

Supported: 100rel

20.38 Timestamp

Timestamp 头字段描述了 UAC 向 UAS 发送请求的时间。

有关怎样对包含该头字段的请求产生响应的细节，参见第 8.2.6 节。尽管在这里没有定义使用头字段的标准行为，它允许扩展或者 SIP 应用获得 RTT 评估。

实例：

Timestamp: 54

20.39 To

To 头字段规定了请求的逻辑接收者。

可选的 display-name 必须由用户界面交付。Tag 参数作为对话标识符的通用机制。

Tag 参数的详情参见第 19.3 节。

To 头字段等式的比较与 From 头字段的比较是相同的。显示名称、URI 和 URI 参数以及头字段参数的分析规则，参见第 20.10 节。

To 头字段的简写形式是 t。

下面是有效的 To 头字段的实例：

To: The Operator <sip:operator@cs.columbia.edu>;tag=287447

t: sip:+12125551212@server.phone2net.com

20.40 Unsupported

Unsupported 头字段列出了 UAS 不支持的特征。动机参见第 20.32 节。

实例：

Unsupported: foo

20.41 User-Agent

User-Agent 头字段包含了 UAC 发起请求的信息。该头字段的语义在[H14.43]中定义。

显示用户代理的特定软件版本可能使用户代理更加容易受到有安全漏洞的软件的攻击。实现者应该使 User-Agent 头字段成为可控制的选项。

实例：

User-Agent: Softphone Beta1.5

20.42 Via

Via 头字段说明到目前为止请求选择的路径，并且说明了在路由响应中应该遵循的路径。Via 头字段值中的分支 ID 参数作为事务标识符，并且代理使用该参数检测循环。

Via 头字段值包含了发送消息使用的传输层协议、客户端主机的名称或者网络地址和希望收到响应的可能端口号。Via 头字段值还包括了如 maddr、ttl、received 和 branch 参数，

在其它章节中介绍了它们的含义和用法。为使实现遵循本规范，分支参数的值必须初始化为 magic cookie “z9hG4bK”，如在第 8.1.1 节中讨论的那样。

在这里定义的传输层协议是 UDP、TCP、TLS 和 SCTP。TLS 是在 TCP 之上的 TLS。当向 SIPS URI 发送请求时，协议仍然标识为“SIP”，并且传输层协议是 TLS。

Via: SIP/2.0/UDP erlang.bell-telephone.com:5060;branch=z9hG4bK87asdks7

Via: SIP/2.0/UDP 192.0.2.1:5060 ;received=192.0.2.207

;branch=z9hG4bK77asjd

Via 头字段的简写形式是 v。

在该实例中，多个家用主机用两个地址发起消息，192.0.2.1 和 192.0.2.207。发送者错误地猜测了将使用哪一个网络接口。Erlang.bell-telephone.com 发现了这个不匹配，并且向以前跳跃点的 Via 头字段值添加包含该包实际源地址的参数。

主机或者网络地址和端口号不需要遵循 SIP URI 语法。特别地，允许在 LWS 两边使用“:”或者“/”，如下所示：

Via: SIP / 2.0 / UDP first.example.com: 4000;ttl=16

;maddr=224.2.0.1 ;branch=z9hG4bKa7c6a8dlze.1

尽管本规范强制规定分支参数存在于所有请求中，但该头字段的 BNF 说明分支参数是可选的。这就允许与 RFC 2543 元素的互操作，而不需要必须插入分支参数。

如果两个 Via 头字段的发送协议和由谁发送（sent-by）字段是相等的，并且它们有相同的参数，所有参数值也是相等的，则这两个头字段是相等的。

20.43 Warning

使用 Warning 头字段携带有关响应状态的附加信息。Warning 头字段值和响应一起发送，并且包含了三位的警告代码、主机名称和警告文本。

“警告文本”应该是最可能被接收响应的人所理解的自然语言。该决定是基于任何可用的知识，比如用户的位置、请求中的 Accept-Language 字段、或者响应中的 Content-Language 字段。默认语言是 i-default [20]。

在下面列出了当前定义的“Warn-code”以及推荐的英文警告文本和它们的含义说明。这些警告描述了会话描述导致的错误。警告代码的第一位以“3”开始表示指定用于 SIP 的警告。保留 300 到 329 的警告用作在会话描述说明关键字问题，从 330 到 339 的警告是与在会话描述中请求的基础网络服务相关的警告，从 370 到 379 的警告是与在会话描述中请求的 QoS 参数数量相关的警告，从 390 到 399 的警告是以上分类中没有包含的各种警告。

300 不相容的网络协议：在会话描述中包含的一个或者多个网络协议是不可用的。

301 不相容的网络地址格式：在会话描述中包含的一个或者多个网络地址格式是不可用的。

302 不相容的传输协议：在会话描述中描述的一个或者多个传输协议是不可用的。

303 不相容的带宽单位：在会话描述中包含的一个或者多个带宽测量单位是不可理解的。

304 不可用的媒体类型：在会话描述中包含的一个或者多个媒体类型是不可用的。

305 不相容的媒体格式：在会话描述中包含的一个或者多个媒体格式是不可用的。

306 不可理解的属性：在会话描述中不支持一个或者多个媒体属性。

307 不可理解的会话描述参数：以上列出的参数之外的参数是不可理解的。

330 不可用的多播：用户所在的地点不支持多播。

331 不可用的单播：用户所在的地点不支持单播通信（通常是因为防火墙的状态）。

370 不足的带宽：在会话描述中指定的或者媒体定义的带宽超过已知可用的带宽。

399 各种警告：警告文本包括了向人提供的或者记录的任何信息。收到该警告的系统不能采取任何自动行动。

HTTP/1.1 采用 1xx 和 2xx。

附加的“warn-code”能够通过 IANA 定义，就像在第 27.2 节中定义的。

实例：

Warning: 307 isi.edu "Session parameter ’ foo’ not understood"

Warning: 301 isi.edu "Incompatible network address type ’ E.164’ "

20.44 WWW-Authenticate

WWW-Authenticate 头字段值包含了认证挑战。有关它的更加详细的用法，参见第 22.2 节。

实例：

WWW-Authenticate: Digest realm="atlanta.com",

domain="sip:boxesbybob.com", qop="auth",

nonce="f84f1cec41e6cbe5aea9c8e88d359",

opaque="", stale=FALSE, algorithm=MD5

21 响应代码

响应代码遵循并且扩展了 HTTP/1.1 响应代码。不是所有的 HTTP/1.1 响应代码都合适，以下仅提供了那些合适的 HTTP/1.1 响应代码。其它的 HTTP/1.1 响应代码不能使用。而且，SIP 定义了新的类，6xx。

21.1 临时 1xx

临时响应，也称为信息的响应，说明连接的服务器正在执行某些深层次的操作并且还没有最终的响应。如果服务器希望占用多于 200 毫秒以获得最终的响应，则发送 1xx 响应。注意，1xx 响应并不是可靠的传送。它们从不会造成客户端发送 ACK。临时（1xx）响应可以包含带有会话描述的消息体。

21.1.1 100 尝试

该响应表示请求已经被下一个跳跃点的服务器接收，并且正在执行某些未指定的操作代表该呼叫（例如，正在查询数据库）。该响应，与其它所有的临时响应一样，通过 UAC 终止重发 INVITE。100（尝试）响应不用于其它临时响应，因为它从不通过状态代理向上流转发。

21.1.2 180 响铃

接收 INVITE 的 UA 正在试图警告用户。可用该响应发起本地回铃。

21.1.3 181 呼叫正被转发

服务器可以使用该状态代码表示呼叫正在向不同的目的地转发。

21.1.4 182 排队

被呼叫方暂时不能访问，但是服务器确定将该呼叫放入等待队列而不是拒绝它。当被呼叫方可以访问时，它会返回相应的最终状态响应。原因解析可以提供有关呼叫状态更加详细的情况，比如，“等待队列中有 5 个呼叫；预期等待时间是 15 分钟”。服务器可以发布几个 182（排队）响应以更新呼叫方的有关呼叫等待队列的状态。

21.1.5 183 会话处理中

使用 183（会话处理中）响应传达有关未分类的呼叫处理的信息。可以使用原因解析、头字段或者消息体传送有关呼叫处理更加详细的情况。

21.2 成功的 2xx

请求成功。

21.2.1 200 OK

请求成功。响应返回的信息是与请求中使用的方法有关。

21.3 重定向的 3xx

3xx 响应提供有关用户新位置，或者有关能够满足呼叫的可替代服务的信息。

21.3.1 300 多种选择

在请求中的地址分解为若干选择，每一个选择都有自己的特殊的地址，用户（或者 UA）能够选择喜欢的通信终端，并将请求重定向到该位置。

如果 Accept 请求头字段允许，响应可以包括消息体，该消息体包含了资源特征以及用户或者 UA 能选择的最合适的位置的列表。然而，没有为该消息体定义 MIME 类型。

选择还可以作为 Contact 字段（第 20.10 节）列出。与 HTTP 不同，SIP 响应可以包含若干 Contact 字段或者在 Contact 字段中的地址列表。UA 可以使用 Contact 头字段值用于自动重定向或者向用户挑战要求确认选择。然而，本规范没有为自动选择定义任何标准。

如果能够从不同的位置到达被呼叫方，并且服务器不能或者不推荐代理请求，该状态响应适用。

21.3.2 301 永久清除

不能够在 Request-URI 地址中找到用户，并且请求客户端将用 Contact 头字段（第 20.10 节）提供的新地址重试请求。请求者将用新值更新地址目录、通讯录和用户位置缓存，并向列出的地址重定向未来请求。

21.3.3 302 临时清除

请求客户端将用 Contact 头字段（第 20.10 节）提供的新地址重试请求。在响应中，新请求的 Request-URI 使用响应中的 Contact 头字段值。

Contact URI 有效期的持续时间可以通过 Expires（第 20.19 节）头字段或者 Contact 头字段的 expires 参数来说明。代理和 UA 可以在终止的持续时间内缓存该 URI。如果没有明确的终止时间，用于递归的地址仅仅一次有效，并且不必缓存用于未来的处理。

如果来自 Contact 头字段的 URI 缓存出错，可以再试一次来自重定向请求的 Request-URI。

临时的 URI 的过期时间可能比终止时间早，那么，新的临时 URI 可用。

21.3.4 305 使用代理

必须通过 Contact 字段给出的代理访问请求的资源。Contact 字段给出代理的 URI。接收者应该通过代理重复该单请求。仅只有 UAS 才能产生 305（使用代理）响应。

21.3.5 380 可选择服务

呼叫没有成功，但是可选择的服务可能存在。

在响应的消息体中描述可选择的服务。在这里没有定义这种消息体的格式，并且该格式将成为未来标准化的主体。

21.4 请求错误 4xx

4xx 响应定义了来自具体服务器的错误响应。客户端将不再重试未经过修改（比如，添加相应的授权）的请求。然而，向其它的服务器发送同样的请求，可能会成功。

21.4.1 400 错误的请求

因为不正确的语法，不能理解请求。原因分析将详细识别语法问题，比如，“丢失 Call-ID 头字段”。

21.4.2 401 未授权

请求需要用户认证。UAS 和注册服务器发起该响应，同时，代理服务器使用 407（需要代理认证）。

21.4.3 402 需要报酬

为未来使用预定。

21.4.4 403 禁止

服务器理解请求，但是拒绝实现它。Authorization 不会有帮助，并且不会重复请求。

21.4.5 404 没找到

服务器有确定的信息，Request-URI 规定的域中不存在用户。如果 Request-URI 中的域与请求的接收者处理的任何域不匹配，也返回该状态。

21.4.6 405 方法不允许

理解 Request-Line 中规定的方法，但是不允许 Request-URI 识别地址。

响应必须包括 Allow 头字段，该头字段包含了指定地址有效方法的列表。

21.4.7 406 不可接受

请求识别的资源仅仅能够产生响应实体，该实体有根据请求发送的 Accept 头字段不可接受的内容特征。

21.4.8 407 需要代理认证

该代码与 401（未授权）相同，但是表明了客户端必须首先向代理自认证。SIP 访问认证将在第 26 章和第 22.3 节中解释。

访问通信信道的应用（例如，电话网关）可以使用该状态代码，而不是被呼叫者需要认证。

21.4.9 408 请求超时

服务器不能在相应的时间范围内产生响应，例如，它不能及时地确定用户的位置。稍后，客户端可以在不作修改的情况下重复该请求。

21.4.10 410 离开

请求的资源在服务器端不再可用，并且不知道任何转发地址。认为该状态是永久不变的。如果服务器不知道，或者没有设备以确定状态是否是永恒的，将改用状态代码 404（没找到）。

21.4.11 413 请求的实体太大

因为请求实体超过服务器愿意或者能够处理的范围，服务器拒绝处理请求。服务器可以结束连接以阻止客户端继续请求。

如果状态是暂时的，服务器将包括 Retry-After 头字段表示它是暂时的，在此之后客户端可以重试。

21.4.12 414 Request-URI 太长

因为 Request-URI 的长度超过服务器愿意解释的范围，服务器拒绝服务请求。

21.4.13 415 不支持的媒体类型

因为请求方法的服务器不支持请求的消息体的格式，服务器拒绝服务请求。服务器必须返回可接受的格式列表，根据具体问题和内容，使用 Accept、Accept-Encoding 或者 Accept-Language 头字段。在第 8.1.3 节中介绍了 UAC 处理该响应。

21.4.14 416 不支持的 URI 模式

因为服务器不知道 Request-URI 中 URI 的模式，服务器不能处理请求。在 8.1.3 节中介绍了客户端处理该响应。

21.4.15 420 错误的扩展

服务器不理解 Proxy-Require（第 20.29 节）或者 Require（第 20.32 节）头字段中规定的协议的扩展。在响应中，服务器必须在 Unsupported 头字段中包括不支持的扩展的列表。

在第 8.1.3 节中介绍了 UAC 处理该响应。

21.4.16 421 需要扩展

UAS 需要特殊的扩展以处理请求，但是在请求中，该扩展没有在 Supported 头字段中列出。带该状态代码的响应必须包含 Require 头字段，在该头字段中列出了需要的扩展。

UAS 不应该使用该响应，除非它确实不能向客户端提供任何有用的服务。相反的，如果 Supported 头字段中没有列出希望的扩展，通过使用基准 SIP 性能和客户端支持的扩展，服务器将处理请求。

21.4.17 423 时间间隔太短

因为请求刷新资源的终止时间太短，服务器拒绝请求。注册服务器能够使用该响应拒绝 Contact 头字段终止时间太短的注册。在第 10.2.8 节、第 10.3 节和第 20.23 节中介绍了该响应和相关的 Min-Expires 头字段的用法。

21.4.18 480 暂时不可用

成功联系被呼叫者的终端系统，但是当前不可访问被呼叫者（例如，没有登录、登录但是处于不能与他通信的状态、或者启用了“不要打扰”功能）。响应可以在 Retry-After 头字段中说明更合适的呼叫时间。也可以在别处访问用户（该服务器不知道）。原因分析将说明为什么被呼叫者不可访问的更加详细的原因。UA 可以设置该值。可以使用状态 486（这里忙）以更详细地说明呼叫失败的具体原因。

重定向或者代理服务器也返回该状态，重定向或者代理服务器认可 Request-URI 识别的用户，但是该状态当前没有该用户的有效转发位置。

21.4.19 481 呼叫/事务不存在

该状态表示 UAS 接收了与任何现有的对话或者事务不匹配的请求。

21.4.20 482 检测到循环

服务器已经检测到循环（第 16.3 节第 4 项）。

21.4.21 483 太多跳跃点

服务器接收了包含值为 0 的 Max-Forwards 头字段(第 20.22 节)的请求。

21.4.22 484 地址不完整

服务器接收了带不完整的 Request-URI 的请求。附加信息将在原因分析中提供。

该状态代码允许重叠拨号。通过重叠拨号，客户端不知道拨号字符串的长度。它发送长度增加的字符串，提示用户不断输入，直到不再接收 484（地址不完整）状态响应。

21.4.23 485 不明确

Request-URI 是不明确的。响应可以在 Contact 头字段中包含可能明确的地址列表。显示可选择的地址会破坏用户或者组织的隐私。必须配置服务器用状态 404（没找到）响应或者取消不明确的 Request-URI 的可能的选择列表。

带 Request-URI sip: lee@example.com 请求的响应实例：

SIP/2.0 485 Ambiguous

Contact: Carol Lee <sip:carol.lee@example.com>

Contact: Ping Lee <sip:p.lee@example.com>

Contact: Lee M. Foote <sips:lee.foote@example.com>

一些电子邮件和语音邮件系统提供该功能。因为语义不同，使用与 3xx 分离的状态代码：对于 300，假定提供的选择将到达相同的用户或者服务。同时自动选择或者顺序查找使 3xx 响应有意义，485（不明确）响应需要用户干预。

21.4.24 486 这里忙

成功联系被呼叫者的终端系统，但是被呼叫者当前不愿意或者不可以在该终端系统接受附加的呼叫。响应可以在 Retry-After 头字段中说明更合适的呼叫时间。可以在别处访问用户，比如通过语音邮件服务。如果客户端知道其它终端系统都不能接受该呼叫，将使用状态 600（各处忙）。

21.4.25 487 请求结束

BYE 或者 CANCEL 请求结束请求。CANCEL 请求本身从不返回该响应。

21.4.26 488 这里不能接受

该响应与 606（不能接受）有同样的意思，但是仅仅应用于 Request-URI 访问规定的资源，并且请求可以在任何地方成功。

在响应中可以存在包括媒体性能描述的消息体，该消息体根据 INVITE（或者 application/sdp，如果不存在 INVITE）中 Accept 头字段格式化，与对 OPTIONS 请求的 200（OK）响应中的消息体相同。

21.4.27 491 请求挂起

UAS 接收请求，在相同的对话中，UAS 有挂起的请求。第 14.2 节中描述了怎样解决这种“glare”的情况。

21.4.28 493 无法识别的

UAS 接收的请求包含加密 MIME 体，接收者没有处理该 MIME 体或者不向该体提供正确的解密密钥。该响应可以有包含了正确的公钥的单消息体，该公钥可用于向 UA 发送的加密的 MIME 体。将在第 23.2 节中讲到该响应代码的用法。

21.5 服务器错误 5xx

5xx 响应是当服务器本身发生错误时提供的错误响应。

21.5.1 500 服务器内部错误

服务器遇到没有预料的状态，阻止它实现请求。客户端可以显示具体的错误状态，并且可以在若干秒后重试请求。

如果状态是暂时的，使用 Retry-After 头字段，服务器可以说明客户端可重试请求的时间。

21.5.2 501 没有实现

服务器不支持实现请求所需的功能。它是当 UAS 不认可请求的方法以及不能为任何用户支持请求的方法时的相应的响应。（代理转发所有的请求，而不考虑方法。）

注意，当服务器认可请求方法，但是不允许或者不支持该方法时，发送 405（方法不允许）。

21.5.3 502 坏的网关

在尝试实现请求过程中，服务器接收了来自它访问的下游服务器的无效响应，此时，该服务器相当于网关或者代理。

21.5.4 503 服务不能提供

因为暂时的过载或者服务器维修，服务器暂时不能处理请求。在 Retry-After 头字段中，服务器可以说明客户端可以重试请求的时间。如果没有提供 Retry-After 头字段，客户端必须当作好像它接收了 500 响应（Server 内部错误）。

接收了 503（服务不能提供）的客户端（代理或者 UAC）将尝试向可选服务器转发请求。如果 Retry-After 头字段存在，在 Retry-After 头字段中规定的时间间隔内，该客户端（代理或者 UAC）不应该向可选服务器转发任何其它的请求。

服务器可以拒绝连接或者撤销请求，而不是用 503（服务不能提供）响应。

21.5.5 504 服务器超时

在尝试处理请求过程中，服务器没有从它访问的外部服务器接收及时的响应。如果在来自上游服务器的 Expires 头字段规定的时间范围内，没有响应，改用 408（请求超时）。

21.5.6 505 版本不支持

服务器不支持或者拒绝支持请求中使用的 SIP 协议的版本。除了此错误消息外，服务器指出它不能够或者不愿意使用与客户端相同的主版本完成请求。

21.5.7 513 消息太大

因为消息长度超过服务器的容量，该服务器不能处理请求。

21.6 全局错误 6xx

6xx 响应说明服务器有具体用户的确定信息，不仅仅是 Request-URI 中指定的特殊实例。

21.6.1 600 各处忙

成功联系被呼叫者的终端系统，但是被呼叫者正在忙碌并且不愿意在该时间接受呼叫。响应可以在 Retry-After 头字段中说明更合适的呼叫时间。如果被呼叫者不希望显示拒绝呼叫的原因，被呼叫者改为使用状态代码 603（拒绝）。仅仅在客户端知道其它终端（如，语音邮件系统）都不会回复请求的情况下，返回该状态响应。否则，返回 486（这儿忙）。

21.6.2 603 拒绝

成功联系被呼叫者的机器，但是用户明确地不希望或者不能够参加。响应可以在 Retry-After 头字段中说明更好的呼叫时间。仅仅在客户端知道其它终端都不会回复请求的

情况下，返回该状态代码。

21.6.3 604 不存在

服务器有权威信息，在 Request-URI 中指定的用户在任何地方都不存在。

21.6.4 606 不能接受

成功访问用户代理，但是不接受会话描述的某些方面，比如，被请求的媒体、带宽或者寻址方式。

606 响应（不能接受）的意思是，用户希望通信，但是不能充分地支持会话描述。在 Warning 头字段中，606 响应（不能接受）可以包含描述为什么不支持会话描述的原因列表。在第 20.43 节中列出了 Warning 原因代码。

在响应中可以存在包含媒体性能描述的消息体，该消息体根据 INVITE（或者 application/sdp，如果不存在 INVITE）中的 Accept 头字段格式化，与对 OPTIONS 请求的 200（OK）响应中的消息体相同。

希望不是经常需要协商，并且当邀请新的用户参加现有的会议时，不可能协商。邀请发起人决定是否对 606（不能接受）响应起作用。

仅仅在客户端知道其它终端都不会回复请求的情况下，返回该状态响应。

22 HTTP 认证的用法

SIP 为认证提供了无状态的、基于挑战的机制，该机制以 HTTP 中的认证为基础。无论代理服务器或者 UA 何时接收到请求（第 22.1 节中提到的情况除外），它都要求请求发起者保证其身份。一旦确定了发起者，请求的接收者将确定此用户是否有权发起的请求。本文中，没有推荐和讨论授权系统。

本章中描述的“摘要”认证机制，仅提供了消息认证和重放保护，不提供消息的完整性和机密性。为阻止主动攻击者修改 SIP 请求和响应，需要采用除摘要以外的其它保护措施。

注意，由于“基本”认证的弱安全性，因此不支持“基本”认证的用法。服务器不能接受使用“基本”授权模式的凭证，并且服务器还不能用“基本”挑战。这是对 RFC 2543 的修改。

22.1 架构

SIP 认证的架构与 HTTP（RFC 2617[16]）的架构非常相似。尤其是认证模式、认证参数、挑战、域、域值和凭证的 BNF 是一样的（尽管不允许使用“基本”作为模式）。SIP 中，UAS 使用 401 响应（未授权）挑战 UAC 的身份。此外，注册服务器和重定向服务器可以使用 401 响应（未授权）认证，但是代理不能，它可以改用 407 响应（需要代理认证）。在各种消息中需要包含的 Proxy-Authenticate、Proxy-Authorization、WWW-Authenticate 和 Authorization， 与 RFC 2617[16]中描述的一样。

因为 SIP 没有规范根 URL 的概念，SIP 中的空间保护的概念解释是不相同的。域字符串单独定义了保护域。这一点与 RFC 2543 不同。在 RFC 2543 中，Request-URI 和域共同定义保护域。

因为 UAC 发送的 Request-URI 和挑战服务器接收的 Request-URI 可能不同，同时，UAC 可能不知道 Request-URI 真正的最终形式，所以先前保护域的定义产生了一些混淆。而且，

以前的定义是基于 Request-URI 中 SIP URI 的状态，似乎排除了可选的 URI 模式（例如，tel URL）。

将对所接收的请求认证的用户代理和代理服务器的操作员必须遵循以下为其服务器创建域字符串的规定：

- 域字符串必须全局唯一。推荐域字符串包含主机名或者域名，遵循 RFC 2617[16] 中 3.2.1 节的推荐。
- 域字符串应该提供可向用户交付的、使用者可理解的标识符。

例如：

INVITE sip:bob@biloxi.com SIP/2.0

Authorization: Digest realm="biloxi.com", <...>

通常，SIP 认证对于特定域、保护域是有意义的。因此，对于摘要认证，每个这样的保护域有它自己的用户名和密码。如果服务器不需要认证具体请求，它可以接受默认用户名“anonymous”和没有密码（密码为“”）。类似地，代表许多用户的 UAC，如 PSTN 网关，可以有它们自己指定设备的用户名和密码，而不是说明具体的用户。

而服务器可以合理地挑战大多数的 SIP 请求，本文定义了两个需要认证特殊处理的请求：ACK 和 CANCEL。

认证模式使用响应携带用于计算随机数的值（如摘要），在这样的认证模式下，没有响应的请求会产生一些问题，包括 ACK。因此，ACK 服务器必须接受 INVITE 中服务器接受的任何凭证。

创建 ACK 消息的 UAC 将复制 ACK 对应的 INVITE 中出现的所有 Authorization 和 Proxy-Authorization 头字段值。服务器不能试图挑战 ACK。

尽管 CANCEL 方法确实产生了一个响应（2xx），因为不能重提交这些请求，服务器不能试图挑战 CANCEL 请求，因为不能重提交这些请求。通常，如果 CANCEL 请求与发送的已取消的请求来自相同的跳跃点（如果 26.2.1 节所描述的某种传输层或者网络层安全关联在适当的位置），服务器应该接受该 CANCEL 请求。

当 UAC 接收了一个挑战，如果 UAC 设备不知道所述域的凭证，它将向用户交付挑战中 realm 参数的内容（在 WWW-Authenticate 或者 Proxy-Authenticate 头字段中）。为域预配置带凭证的 UA 的服务供应商应该知道，当在预配置设备处挑战时，用户不会有机会为该域提供它们自己的凭证。

最后，请注意，即使 UAC 能够定位与正确的域结合的凭证，可能会存在这样的情况：这些凭证不再是有效的，或者无论什么原因挑战服务器将不接受这些凭证（尤其是当提交的是“anonymous”和没有密码）。在这样的实例中，服务器可以重复它的挑战，或者用 403 禁止响应。UAC 不能用刚刚被拒绝的凭证重试请求（尽管可以在随机数已经失效的情况下重试请求）。

22.2 用户之间认证

当 UAS 接收了来自 UAC 的请求，UAS 可以在处理请求之前认证发起者。如果在请求中没有提供凭证（在 Authorization 头字段中），用 401（未授权）状态代码拒绝请求，UAS 可以要求发起者提供凭证。

必须在 401（未授权）响应消息中包括 WWW-Authenticate 响应头字段。字段值至少由一个挑战组成，该挑战说明了域可用的认证模式和参数。

在 401 挑战中 WWW- Authenticate 头字段的实例：

```
WWW-Authenticate: Digest
        realm="biloxi.com",
        qop="auth,auth-int",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

当源 UAC 接收 401（未授权），如果可以，它将用正确的凭证重发请求。在处理前，UAC 需要源用户输入。

一旦提供了认证凭证（用户直接提供，或者在内部密钥环中发现），UA 将为给定值的 To 头字段和 realm 缓存凭证，并且为目的地的下一请求尝试重用这些值。UA 可以用任何一种方式缓存凭证。

如果不能为域定位凭证，UAC 可以用用户名“anonymous”和没有密码（密码为“”）尝试重试请求。

一旦定位了凭证，任何希望向 UAS 或者注册服务器自认证的 UA——通常但不是必须在接收 401（未授权）响应后，可以通过在请求中包含 Authorization 头字段实现自认证。Authorization 字段值由包含 UA 认证信息的凭证组成，该认证信息用于被请求资源的域和需要支持认证和重放保护的参数。

Authorization 头字段实例：

```
Authorization: Digest username="bob",
realm="biloxi.com",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
uri="sip:bob@biloxi.com",
qop=auth,
nc=00000001,
cnonce="0a4f113b",
response="6629fae49393a05397450978507c4ef1",
opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

在接收 401（未授权）或者 407（需要代理认证）响应后，当 UAC 用它的凭证再次提交请求时，它必须递增 Cseq 头字段值——正如发送更新的请求时它通常会做的那样。

22.3 代理对用户的认证

同样的，当 UAC 向代理服务器发送请求时，代理服务器可以在请求处理之前认证发起者。在请求中，如果没有提供凭证（在 Proxy-Authorization 头字段中），代理用 407（需要代理认证）状态代码拒绝请求，要求发起者提供凭证。代理必须用请求资源的代理可用的 Proxy-Authentication 头字段值填写 407（需要代理认证）消息。

[16]中描述了 Proxy-Authentication 和 Proxy-Authorization 的对比用法，并说明了其不同点。代理不能向 Proxy-Authorization 头字段添加值。遵循其它响应的过程，必须向上游 UAC 转发所有的 407（需要代理认证）响应。UAC 负责添加包含凭证的 Proxy-Authorization 头字段值，该凭证用于要求认证的代理的域。

如果代理重提交添加了 Proxy-Authorization 头字段值的请求，它需要在新请求中递增 Cseq。然而，因为 Cseq 值不同，造成提交初始请求的 UAC 放弃来自 UAS 的响应。

当源 UAC 接收了 407（需要代理认证），如果可以，它将用正确的凭证重发起请求。它将遵循与显示响应 401 的 realm 参数相同的过程。

如果不能为域定位凭证，UAC 可以尝试用用户名“anonymous”和没有密码（密码“”）重试请求。

UAC 还将缓存在重发请求中使用的凭证。

推荐使用以下规则缓存代理凭证：

如果 UA 在带特定 Call-ID 的请求的 401/407 响应中接收了 Proxy-Authenticate 头字段值，UA 将在所有包含相同 Call-ID 的后续请求中，为域合并凭证。不应该通过对话缓存凭证；然而，如果用 UA 本地带外代理的域配置了该 UA，如果存在的话，那么 UA 可以通过对话为该域缓存凭证。

注意，这就意味着对话的未来请求可以包含 Route 头路径中的代理不需要的凭证。

希望向代理服务器自认证的任何 UA——通常但不是必须在接收 407（需要代理认证）响应后，可以通过在请求中包含 Proxy-Authorization 头字段实现自认证。Proxy-Authorization 请求头字段允许客户端向需要认证的代理识别自己（或者其用户）。Proxy-Authorization 头字段值由包含 UA 认证信息的凭证组成，该认证信息用于代理和/或被请求资源的域。

Proxy-Authorization 头字段值仅用于 realm 参数中识别的域的代理（该代理先前可能已经使用 Proxy-Authenticate 字段请求认证）。

在一条链中使用多个代理时，任何与 Proxy-Authorization 头字段值规定的 realm 参数不匹配的代理都不能消耗 Proxy-Authorization 头字段值。

注意，如果在 Proxy-Authorization 头字段中使用了不支持域的认证模式，代理服务器必须尝试解析所有的 Proxy-Authorization 头字段值，以确定它们中的一个是否有代理服务器认为有效的凭证。因为在大型网络中，这样可能会非常耗时，所以代理服务器应该在 Proxy-Authorization 头字段中使用一种支持的域认证模式。

如果分发请求（正如第 16.7 节中所描述的），各种代理服务器和/或 UA 可能希望挑战 UAC。在这种情况下，分发代理服务器负责将这些挑战聚集在一个响应中。在 UA 的分发代理发送的单一响应中，必须放入在分发请求的响应中接收的每个 WWW-Authenticate 和 Proxy-Authenticate 值；这些头字段值的顺序不重要。

当代理服务器发起挑战响应请求时，直到 UAC 用有效的凭证重试请求，它才代理请求。分发代理可以同时向需要认证的多个代理服务器转发请求，直到源 UAC 在它们各自的域中自认证为止，每个代理服务器才转发请求。如果 UAC 没有为每个挑战提供凭证，发起挑战的代理服务器将不会向目的用户所在的 UA 转发请求，因此，并未充分体现分发的优点。

当在对 401（未授权）或者 407（需要代理认证）的响应中重提交包含多个挑战的请求时，UAC 可以在每个 WWW-Authenticate 值中包括 Authorization 值，在每个 Proxy-Authenticate 值中包含 Proxy-Authorization 值，用于希望 UAC 提供凭证的那个挑战。正如以上注意事项所述，realm 参数将区分在请求中的多个凭证。

相同的 401（未授权）或者 407（需要代理认证）中可能会出现与相同的域相关的多个挑战。例如，当分发请求到达使用公共域的、在相同管理域中的多个代理时，可能发生这种情况。当它重试请求时，UAC 可以因此在 Authorization 或者 Proxy-Authorization 头字段中用相同的 realm 参数值提供多个凭证。相同的域应该使用相同的凭证。

22.4 摘要认证模式

本节描述了SIP应用HTTP摘要认证模式所需的修改和澄清。SIP模式的用法与HTTP[16]的用法几乎完全相同。

因为RFC 2543是基于RFC 2069[40]中定义的HTTP摘要，支持RFC 2617的SIP服务器必须保证它们向后兼容RFC 2069。在RFC 2617中规定了向后兼容过程。注意，SIP服务器不能接受或者请求基本认证。

摘要认证规则遵循[16]中的定义，用“SIP/2.0”代替了 “HTTP/1.1”，除了以下几点区别：

1. 挑战中包括的URI有以下的BNF：

   URI＝SIP-URI/SIP-URI

2. 在RFC 2617中，BNF有一个错误，HTTP摘要认证的授权头字段的‘uri’参数，没有引号。（RFC 2617第3.5节中的实例是正确的。）对于SIP，‘uri’必须有引号。

3. digest-uri-value的BNF是：

   digest-uri-value = Request-URI ; 正如在第25章中定义的。

4. 根据Etag选择随机数的过程不适用于SIP。

5. 在RFC 2617[16]中有关缓存操作的文本不适用于SIP。

6. RFC 2617需要服务器检查请求行中的URI与Authorization头字段中的URI是否指向同一个资源。在SIP环境中，因为在某些代理处转发，这两个URI可以代表不同的用户。 因此，在SIP中，服务器可以检查Authorization头字段值中的Request-URI与用户的一致性，服务器愿意为该用户接受转发或直接请求，但是，如果这两个字段不相等，并不一定失败。

7. 在摘要认证模式中，作为消息完整性保证的A2值计算的澄清，当实体为空时（即，当SIP消息没有消息体时），实现者应该假定实体的哈希解析为空字符串的MD5哈希，或者：

   H(entity-body) = MD5("") = "d41d8cd98f00b204e9800998ecf8427e"

8. RFC 2617指出，如果没有发送qop指令，不能在Authorization（和扩展的Proxy-Authorization）头字段中发送cnonce值。因此，任何与conoce（包括“MD5-Sess”)有关的算法，需要发送qop指令。为了使RFC2617向后兼容RFC 2069，在RFC 2617中，可选择使用qop参数；因为RFC 2543是基于RFC 2069，为了客户端和服务器能够接收，qop参数必须保持可选。然而，服务器必须经常在WWW-Authenticate和Proxy-Authenticate头字段值中发送qop参数。如果客户端在挑战头字段中接收了qop参数，它必须在任何结果授权头字段中发送qop参数。

RFC 2543不允许使用Authentication-Info头字段（它有效地使用rfc2069）。然而，目前，我们允许使用该头字段，因为它提供了在消息体上的完整性检查，并且提供了相互认证。RFC 2617[16]定义了在请求中使用qop属性的向后兼容机制。服务器必须使用这些机制以确定客户端是否支持在RFC 2069中没有规定的、RFC 2617中的新的机制。

23 S/MIME

SIP 消息携带 MIME 消息体,MIME 标准包括了保护 MIME 内容以保证完整性和机密性的机制(包括‘multipart/signed’和‘application/pkcs7-mime’MIME 类型,参见 RFC 1847[21]、RFC 2630[22]和 RFC 2633[23])。然而,实现者应该注意,基于观察或者修改 SIP 消息体(尤其是 SDP)的网络中间媒介(非典型的代理服务器)非常少见,安全的 MIME 可能会阻止这些中间媒介运行。

这些特别应用于某些类型的防火墙。

反对使用 RFC 2543 中描述的加密头字段和 SIP 消息体的 PGP 机制。

23.1 S/MIME 证书

S/MIME 识别终端用户使用的证书与服务器使用的证书有一个重要的不同点——这些证书声称用终端用户地址鉴别持有者,而不是声称持有者的身份与特定主机相符合。该地址由 userinfo 加上“@”以及 SIP 或者 SIPS URI 的 domainname 部分连接组成(换言之,电子邮件地址的形式,如 bob@biloxi.com),通常与用户的记录地址一致。

这些证书还与用于签名或者加密 SIP 消息体的密钥有关。用发送者(他可以把它们的公钥适当地包含在消息中)的私钥对消息体签名,而用期望接收者的公钥加密消息体。显而易见,发送者必须预知接收者的公钥以加密消息体。在虚拟密钥环上,公钥能够在 UA 中存储。

每个支持 S/MIME 的用户代理必须包含特定用于终端用户证书的密钥环。该密钥环应该在记录地址和相应的证书之间映射。当用户用相同的记录地址填写信令的源 URI(From 头字段)时,用户将使用相同的证书。

因为目前实际上没有为终端用户应用提供证书的统一的权威,所以与现有终端用户证书有关的任何机制都有很大的限制。无论如何,用户都应该获取已知公共认证中心的证书。作为一种可选的方法,用户可以创建自签名证书。将在 26.4.2 节中深入讲解自签名证书的含义。实现还可以在部署中使用预配置证书,该部署中的所有 SIP 实体之间存在先前信任关系。

除了获取终端用户证书的问题,有少量众所周知的分配终端用户证书的集中目录。然而,证书的持有者应该适当地在任何公共目录中发布它们的证书。同样的,UAC 应该支持输入(手动地或者自动地)证书的机制,该证书可以在对应 SIP 请求的目标 URI 的公共目录中发现的。

23.2 S/MIME 密钥交换

用以下的方式,SIP 本身还可以用作分配公钥的方法。

无论何时在 SIP 的 S/MIME 中使用了 CMS SignedData 消息,该消息都必须包含验证签名所必需的公钥证书。

当 UAC 发送了包含 S/MIME 消息体的请求时(S/MIME 消息体可发起对话,或者在对话环境之外发送非 INVITE 请求),UAC 将构造消息体作为 S/MIME ‘multipart/signed’ 的 CMS SignedDate 消息体。如果期望的 CMS 服务是 EnvelopedData(并且已知目标用户的公钥),UAC 将发送封装在 SignedData 消息中的 EnvelopedData 消息。

当 UAS 接收了包含带有证书的 S/MIME CMS 消息体的请求时,如果可能的话,UAS 将首先用证书中心的可用根证书验证证书。UAS 还应该确定证书的主体(对于 S/MIME,SubjectAltName 将包含相应的身份),将该值与请求的 From 头字段比较。如果因为证书是自签名或者未知权威签名,而不能验证证书,或者如果证书是可验证的,但是它的主体不符

合请求的 From 头字段，那么，UAS 必须在继续前通知其用户证书的状态（包括证书的主体、它的签名者和密钥指纹信息）和明确允许的请求。如果成功验证了证书，并且证书的主体与 SIP 请求的 From 头字段一致，或者用户（通知之后）明确授权使用该证书，那么，UAS 应该将证书添加到本地密钥环中，该密钥环通过证书持有者的记录地址检索。

当 UAS 发送包含 S/MIME 消息体的响应，该 S/MIME 消息体应答对话中的第一个请求，或者 UAS 向对话环境之外的非 INVITE 请求发送响应时，UAS 将构造消息体作为 S/MIME ‘multipart/signed’ CMS SignedData 消息体。如果期望的 CMS 服务是 EnvelopedData，UAS 将发送封装在 SignedData 消息中的 EnvelopedData 消息。

当 UAC 接收了包含 S/MIME 消息体的响应时，该 S/MIME 消息体包括了证书，如果可能的话，UAC 将首先用合适的根证书验证证书。UAC 还将确定证书的主体，将该值与响应的 To 字段比较；尽管这两者可能有很大区别，但这并不一定是安全漏洞。如果因为证书是自签名或者未知权威签名，而不能验证证书，那么，UAC 必须在继续进行前通知其其用户证书的状态（包括证书的主体、它的签名者和密钥指纹信息）和明确允许的请求。如果成功验证了证书，并且证书的主体与响应中 To 头字段一致，或者如果用户（在通知之后）明确授权使用该证书，那么，UAC 应该将证书添加到本地密钥环中，该密钥环通过证书持有者的记录地址检索。如果 UAC 没有在先前的事务中向 UAS 发送它自己的证书，UAC 将为它的下一个请求或者响应使用 CMS SignedData 消息体。

有时，当 UA 接收了包括 From 头字段的请求或者响应时，该 From 头字段与 UA 密钥环中的值一致，UA 将比较在这些消息中提供的证书和在它的密钥环中现有的证书。如果两者有差异，那么 UA 必须在继续处理信令前，通知其用户有关证书的改变（换句话说，预示这是一种潜在的安全漏洞）并且获得用户的允许。如果用户授权该证书，将把该证书与该记录地址的先前值添加到密钥环。

然而，当使用自签名证书或者未知权威签名的证书时，该密钥交换机制不能保证密钥的安全交换，它非常容易受到攻击。然而，作者认为，事实上，与广泛使用的 SSH 应用相比较，该密钥交换机制提供的安全性比没有安全性好。第 26.4.2 节中更加详细地解释了这些限制。

如果 UA 接收了用接收者未知的公钥加密的 S/MIME 消息体，UA 必须用 493（无法识别的）响应拒绝请求。该响应应该在 MIME 消息体中，通过‘certs-only’的“smime-type”参数，为应答者包含有效证书（如果可能，与在拒绝请求的 To 头字段中提供的记录地址一致）。

在没有证书的情况下，发送 493（无法识别的），表示应答者不能使用 S/MIME 加密消息，尽管它们仍旧支持 S/MIME 签名。

注意，如果 MIME 类型不可理解，那么，接收了包含不可选的 S/MIME 消息体请求（用 Content-Disposition 头所需的 handling 参数）的用户代理必须用 415 不支持的媒体类型响应拒绝请求。当发送 S/MIME 时，接收这种响应的用户代理应该通知其用户，远程设备不支持 S/MIME，如果可以，随后在没有 S/MIME 的情况下，重发送请求；然而，该 415 响应可能造成降级攻击。

如果用户代理在请求中发送 S/MIME 消息体，并接收了包含不安全的 MIME 消息体的响应，UAC 将通知其用户，会话是不安全的。然而，如果支持 S/MIME 的用户代理接收了带不安全消息体的请求，它不应该用安全的消息体响应，但是如果它期望发送者的 S/MIME（例如，因为发送者的 From 头字段值与它的密钥链上的身份一致），UAS 将通知其用户，会话是不安全的。

当发生匿名证书管理事件时，在通知用户的先前文本呼叫中会出现很多情况。用户可以询问在这些情况下他们该做什么。

首先，证书中的意料之外的改变，或者在需要安全性时缺乏安全性，会造成警告，但不一定预示攻击正在进行。用户可以中止任何连接尝试或者拒绝它们接收的连接请求；用电话学的说法，它们可以挂起和回叫。用户希望寻找可选方法，以联系其它方并且确定它们已经合理地改变了密钥。注意，有时强迫用户改变它们的证书，例如，当它们怀疑它们私钥的安全性受到威胁时。当他们的私钥不再是秘密时，用户必须合理地产生新的密钥，并且与拥有它们旧密钥的用户重建立信任。

最后，如果在对话期间，UA 接收了 CMS SignedData 消息中的证书，该证书不符合先前在对话中交换的证书，UA 必须通知其用户这一改变，换句话说，预示这是一个潜在的安全漏洞。

23.3 安全的 MIME 消息体

对 SIP 有意义的安全的 MIME 消息体有两种类型：这些消息体的使用将遵循 S/MIME 规范[23]，但有少许变化。

- “multipart/signed”必须仅与CMS分离的签名一起使用。

这允许与不遵循 S/MIME 的接收者的向后兼容。

- S/MIME消息体应该有Content-Disposition头字段，应该需要处理参数值。
- 如果UAC在其密钥环上没有与要发送请求的记录地址有关的证书，那么，UAC将不能发送加密的“application/pkcs7-time”MIME消息。UAC可以用CMS分离的签名发送初始请求，比如OPTION消息，请求远端的证书（签名应该在23.4节中描述的类型的“message/sip”消息体上）。

注意，未来的 S/MIME 标准化工作可以定义不基于证书的密钥。

- S/MIME 消息体的发送者应该使用“SMIMECapabilities”属性（参见[23]的 2.5.2 节），为进一步的通信表示它们的能力和偏爱。尤其注意发送者可以使用“preferSignedData”能力，鼓励接收者用 CMS SignedData 消息响应（例如，当发送上述 OPTIONS 请求时）。
- S/MIME 实现至少必须支持 SHA1 数字签名算法以及 3DES 加密算法。可能支持所有其它的签名和加密算法。实现能够用“SMIMECapabilities”属性协商对这些算法的支持。
- 应该仅用一个证书对 SIP 消息中的 S/MIME 消息体签名。如果 UA 接收了带多个签名的消息，最外层的签名将作为该消息体的唯一的证书。而不使用其它的签名。

以下是在 SIP 消息中加密 S/MIME SDP 消息体的一个实例：

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
      name=smime.p7m
```

```
Content-Disposition: attachment; filename=smime.p7m
    handling=required
**************************************************
* Content-Type: application/sdp                  *
*                                                *
* v=0                                            *
* o=alice 53655765 2353687637 IN IP4 pc33.atlanta.com       *
* s=-                                            *
* t=0 0                                          *
* c=IN IP4 pc33.atlanta.com                      *
* m=audio 3456 RTP/AVP 0 1 3 99                  *
* a=rtpmap:0 PCMU/8000                           *
**************************************************
```

23.4 使用 S/MIME 的 SIP 头的保密性和完整性：隧道 SIP

作为一定程度提供端对端认证、SIP 头字段的完整性和机密性的方法，S/MIME 可以在“message/sip”类型的 MIME 消息体中封装整个 SIP 消息，并且通过与典型 SIP 消息体相同的方法在这些消息体中应用 MIME 安全。这些封装的 SIP 请求和响应不组成单独的对话或者信息，它们是用来验证完整性或提供其它信息的“outer”消息的副本。

如果 UAS 接收了包含隧道“message/sip”的 S/MIME 消息体的请求，它应该在响应中用相同类型的 smime 包含隧道“message/sip”消息体。

任何传统的 MIME 消息体(比如 SDP)应该放在“inner”消息中，因此它们也可以从 S/MIME 安全中受益。注意，如果任何非安全的 MIME 类型也在请求中发送，“message/sip”消息体可以作为 MIME 的“multipart/mixed”消息体的一部分发送。

23.4.1 SIP 头的完整性和机密性

当使用了 S/MIME 完整性或机密性机制时，“inner”消息中的值与“outer”消息中的值可能会有差异。在本节中，将讲述处理这种差异的规则，该规则适用于本文中描述的所有头字段。

注意，为释放时间戳，所有的隧道“message/sip”SIP 消息应该在“inner”和“outer”头中包含 Date 头。

完整性 无论何时执行完整性检查，通过使用第 20 章中描述的 SIP 的比较规则，使签名消息体中的头字段值匹配“outer”消息中的头字段值，以确定头字段的完整性。

代理服务器可以合理地修改的头字段是：Request-URI、Via、Record-Route、Route、Max-Forwards 和 Proxy-Authorization。即使这些头字段不是完整的端对端，实现不应该认为这是一个安全漏洞。本文中定义的任何其它头字段的改变都破坏完整性；必须将区别告诉用户。

机密性 当加密消息时，可以在加密的消息体中包括“outer”消息中未存在的头字段。

因为一些头字段是在请求和响应中所需的头字段，它们常常是明文，这些头字段是：To、From、Call-ID、Cseq 和 Contact。对 Call-ID、Cseq 或者 Contact 加密可能是没有用的，而允许对“outer”消息的 To 或者 From 加密。注意，加密的消息体中的值不是用于识别事

务或者对话——它们仅仅是信息。如果加密的消息体中的 From 头字段与“outer”消息中的值不一样，向用户显示加密的消息体中的值，但是该值不应该在未来消息的“outer”头字段中使用。

主要地，用户代理希望加密有端对端语义的头字段，包括：Subject、Reply-To、Organization、Accept、Accept-Encoding、Accept-Language、Alert-Info、Error-Info、Authentication-Info、Expires、In-Reply-To、Require、Supported、Unsupported、Retry-After、User-Agent、Server 和 Warning。如果这些头字段的任何一个在加密的消息体中出现，无论是否需要向用户显示头字段值或者在 UA 中设置内部状态，将改用这些头字段，而不是“outer”头字段。然而，不应该在未来消息的“outer”头字段中使用这些头字段。

如果存在，Date 头字段必须和“inner”、“outer”头中的一样。

因为 MIME 消息体附加了“inner”消息，通常实现将加密 MIME 规定（MIME-specific）的头字段，包括：MIME-Version、Content-Type、Content-Length、Content-Language、Content-Encoding 和 Content-Disposition。“outer”消息将有合适的 S/MIME 消息体的 MIME 头字段。这些头字段（和头字段开始的 MIME 消息体）将视为 SIP 消息中接收的标准 MIME 头字段和消息体。

加密以下的头字段不是特别的有用：Min-Expires、Timestamp、Authorization、Priority 和 WWW-Authenticate。该范畴还包括那些代理服务器可以改变的头字段（前面章节所描述的）。如果“outer”消息中没有包括这些头字段，UA 将不能在“inner”消息中包括它们。UA 在加密的消息体中接收了这些头字段中的任何一个，将忽略其加密值。

注意，SIP 的扩展可以定义附加的头字段；这些扩展的程序设计者应该描述这种头字段的完整性和机密性。如果 SIP UA 遇到了破坏完整性的未知头字段，它必须忽略该头字段。

23.4.2 隧道完整性和认证

如果在用 CMS 分离的签名签署的“message/sip”MIME 消息体中复制发送者希望保证安全的头字段，那么 S/MIME 消息体中的隧道 SIP 消息就可以为 SIP 头字段提供完整性。

如果“message/sip”消息体至少包含了基本对话标识符(To、From、Call-ID 和 CSeq)，那么签名的 MIME 消息体就可以提供有限的认证。至少，如果接收者不知道对消息体签名所使用的证书，不能验证该证书，那么，签名用来确定发起对话的相同证书持有者在对话中随后发送的请求。如果该签名的 MIME 消息体的接收者可以完全信任该证书(接收者能够验证该证书，接收者从可信任的储存库获取该证书，或者接收者经常使用该证书)，那么签名可看作是证书主体身份的一个更有力的声明。

为了减少加减整个头字段可能造成的混乱，发送者应该在签名的消息体中复制请求的所有头字段。任何需要完整性保护的消息体必须附加“inner”消息。

如果带签名消息体的消息中有 Date 头，如果可行，接收者将比较头字段值和它自己的内部时钟。如果检测到明显的时间差异(大约一小时或者更久)，用户代理将警告用户这一异常，并且注意这是一个潜在的安全漏洞。

如果接收者检测到消息中的完整性被破坏，假若该消息是请求，用 403(禁止)响应拒绝它，或者结束现有的对话。UA 将通知用户这一情况，并且请求明确指导怎样处理。

以下是隧道“message/sip”消息体的应用实例：

```
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
```

```
Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: multipart/signed;
protocol="application/pkcs7-signature";
micalg=sha1; boundary=boundary42
Content-Length: 568

--boundary42
Content-Type: message/sip

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <bob@biloxi.com>
From: Alice <alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 147

v=0
o=UserA 2890844526 2890844526 IN IP4 here.com
s=Session SDP
c=IN IP4 pc33.atlanta.com
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s;
handling=required

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42-
```

23.4.3 隧道加密

期望使用该机制加密 CMS EncelopedData 消息 S/MIME 消息体的“message/sip”MIME 消息体，但实际上，大多数头字段至少对网络有某些用处；用 S/MIME 加密的通常用法是，保证象 SDP 那样的消息体安全，而不是消息头。一些消息的头字段可能可以保证端对端的安全性，比如 Subject 和 Organization。未来 SIP 应用定义的头可能也需要模糊处理。

加密头字段另一个可能的应用是有选择的匿名。可以用不包含个人信息的 From 头字段创建请求(例如，sip:anonymous@anonymizer.invalid)。然而，另一个包含发起人真实记录地址的 From 头字段可以在“message/sip”MIME 消息体中加密，在该消息体中，它仅对对话的终端可见。

注意，如果匿名使用该加密机制，消息的接收者将不再使用 From 头字段作为获取与发送者有关的适当的 S/MIME 密钥的证书密钥链的索引。必须首先解密该消息，并且使用“inner”的 From 头字段作为索引。

为了提供端对端的完整性，发送者应该对加密的“message/sip”MIME 消息体签名。这会创建一个“multipart/signed”的 MIME 消息体，该消息体包含“application/pkcs7-mime”类型的加密消息体和签名。

以下是加密和签名的消息的实例，用“*”框住的文本被加密了：

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Anonymous <sip:anonymous@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:pc33.atlanta.com>
Content-Type: multipart/signed;
  protocol="application/pkcs7-signature";
  micalg=sha1; boundary=boundary42
Content-Length: 568

--boundary42
Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
    name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m
  handling=required
Content-Length: 231

**********************************************
* Content-Type: message/sip                  *
*                                            *
```

24 实例

下面的实例中，为简洁，通常忽略了消息体、相应的 Content-Length 和 Content-Type 头字段。

24.1 注册

Bob 开始注册。消息流如图 9 所示。为简化，这里没有显示注册所需要的认证。

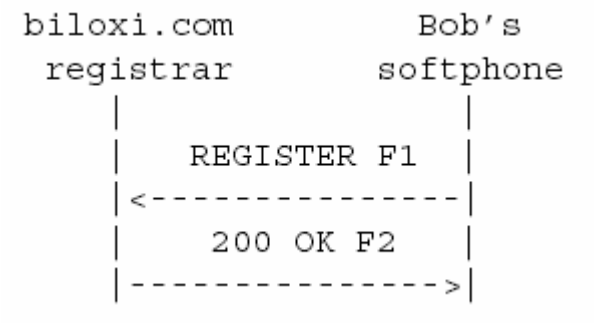

图 9: SIP 注册实例

F1 REGISTER Bob→Registrar

REGISTER sip:registrar.biloxi.com SIP/2.0

Via: SIP/2.0/UDP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7

Max-Forwards: 70

To: Bob <sip:bob@biloxi.com>

From: Bob <sip:bob@biloxi.com>;tag=456248

Call-ID: 843817637684230@998sdasdh09

CSeq: 1826 REGISTER

Contact: <sip:bob@192.0.2.4>

Expires: 7200

Content-Length: 0

注册两小时后过期。注册员用 200 OK 响应。

F2 200 OK Registrar→Bob

SIP/2.0 200 OK

Via: SIP/2.0/UDP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7

; received=192.0.2.4

To: Bob <sip:bob@biloxi.com>;tag=2493k59kd

From: Bob <sip:bob@biloxi.com>;tag=456248

Call-ID: 843817637684230@998sdasdh09

CSeq: 1826 REGISTER

Contact: <sip:bob@192.0.2.4>

Expires: 7200

Content-Length: 0

24.2 会话建立

本实例包含第 4 章中会话建立的全部细节。消息流如图 1 所示。注意这些消息流只显示了所需的头字段的最小集——其它头字段如 Allow 和 Supported 通常都会出现。

F1 INVITE Alice→proxy

INVITE sip:bob@biloxi.com SIP/2.0

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

Max-Forwards: 70

To: Bob <sip:bob@biloxi.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Contact: <sip:alice@pc33.atlanta.com>

Content-Type: application/sdp

Content-Length: 142

(Alice 担 SDP not shown)

F2 100 Trying proxy →Alice

SIP/2.0 100 Trying

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

;received=192.0.2.1

To: Bob <sip:bob@biloxi.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Content-Length: 0

F3 INVITE proxy →biloxi.com proxy

INVITE sip:bob@biloxi.com SIP/2.0

Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

;received=192.0.2.1

Max-Forwards: 69

To: Bob <sip:bob@biloxi.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Contact: <sip:alice@pc33.atlanta.com>

Content-Type: application/sdp

Content-Length: 142

(Alice担 SDP not shown)

F4 100 Trying biloxi.com proxy→proxy

SIP/2.0 100 Trying

Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1

;received=192.0.2.2

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

;received=192.0.2.1

To: Bob <sip:bob@biloxi.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Content-Length: 0

F5 INVITE biloxi.com proxy →Bob

INVITE sip:bob@192.0.2.4 SIP/2.0

Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1

Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1

;received=192.0.2.2

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8

;received=192.0.2.1

Max-Forwards: 68

To: Bob <sip:bob@biloxi.com>

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 INVITE

Contact: <sip:alice@pc33.atlanta.com>

Content-Type: application/sdp

Content-Length: 142

(Alice担 SDP not shown)

F6 180 Ringing Bob→biloxi.com proxy

SIP/2.0 180 Ringing

Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1

;received=192.0.2.3

Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0

F7 180 Ringing Biloxi .company →proxy

SIP/2.0 180 Ringing
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0

F8 180 Ringing proxy →Alice

SIP/2.0 180 Ringing
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0

F9 200 OK Bob→biloxi.com proxy

SIP/2.0 200 OK
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
;received=192.0.2.3

Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131
(Bob' SDP not shown)

F10 200 OK biloxi.com proxy →atlanta.com proxy

SIP/2.0 200 OK
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131
(Bob's SDP not shown)

F11 200 OK proxy →Alice

SIP/2.0 200 OK
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp

Content-Length: 131

(Bob' SDP not shown)

F12 ACK Alice →Bob

ACK sip:bob@192.0.2.4 SIP/2.0

Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds9

Max-Forwards: 70

To: Bob <sip:bob@biloxi.com>;tag=a6c85cf

From: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 314159 ACK

Content-Length: 0

这样就建立了 Alice 和 Bob 间的媒体会话。

Bob 先挂机，注意 Bob 的 SIP 电话维持其 Cseq 编号空间，本例中编号空间的起始值为 231。由于是 Bob 发起请求，所以交换了 To URI 和 From URI 以及标签。

F13 BYE Bob→Alice

BYE sip:alice@pc33.atlanta.com SIP/2.0

Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10

Max-Forwards: 70

From: Bob <sip:bob@biloxi.com>;tag=a6c85cf

To: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 231 BYE

Content-Length: 0

F14 200 OK Alice →Bob

SIP/2.0 200 OK

Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10

From: Bob <sip:bob@biloxi.com>;tag=a6c85cf

To: Alice <sip:alice@atlanta.com>;tag=1928301774

Call-ID: a84b4c76e66710

CSeq: 231 BYE

Content-Length: 0

SIP 呼叫流程文档[41]包含 SIP 消息的更详细的实例。

25 SIP 协议的扩展 BNF

本文档所指定的所有机制都用散文和扩展 BNF 描述，扩展的 BNF 的定义见 RFC2234[9]。RFC2234 第 6 章定义了本规范使用的核心规则，这里不再重述。为理解本规范，实现者需熟

悉 RFC2234 中的符号和内容。一些基本规则用大写字母，如 SP、LWS、HTAB、CRLF、DIGIT、ALPHA 等。在定义中使用了角括号，以阐述规则名的使用。

使用方括号的地方在语法上是多余的。它作为一种语义提示：表示可选择使用特定参数。

25.1 基本规则

本规范使用了以下规则描述基本解析结构。US-ASCII 编码字符集由 ANSI X3.4-1986 定义。

```
Alphanum     = ALPHA / DIGIT
```

有几个规则是从 RFC 2396 [5]中引入的，但为遵循 RFC 2234 [9]，这里对它们进行了更新。这些规则包括：

```
reserved     = ";" / "/" / "?" / ":" / "@" / "&" / "=" / "+"
                    / "$" / ","
unreserved   = alphanum / mark
mark         = "-" / "_" / "." / "→" / "~ " / "*" / "'  "/ "(" / ")"
escaped = "%" HEXDIG HEXDIG
```

如果连续以空格或水平制表开头，SIP 头字段值可分为多行。所有的线性空格，包括折叠的语义都与 SP 相同。接收者在解释字段值或向下游转发消息前，可以用一个 SP 替代任何线性空格。这一点与 RFC 2616[7]中描述的 HTTP/1.1 相同。当线性空格可选时，使用 SWS 结构，线性空格通常在标记和分隔符之间。

```
LWS          = [*WSP CRLF] 1*WSP ; linear whitespace
SWS          = [LWS] ; sep whitespace
```

这里使用冒号将头名和值分隔，根据前面的规则，允许在冒号前出现空格，但在冒号后不允许分行和空格，包括 linebreak。HCOLON 的结构如下：

```
HCOLON       = *( SP / HTAB ) ":" SWS
```

TEXT-UTF8 规则仅用于描述字段内容和值，不用于消息解析器的解释。*TEXT-UTF8 包含 UTF-8 字符集（RFC 2279 [6]）的字符。TEXT-UTF8-TRIM 规则用于描述性字段内容，该内容不是引用串，其中第一和最后一位 LWS 无意义。因此，SIP 不同于 HTTP，HTTP 使用 ISO 8859-1 字符集。

```
TEXT-UTF8-TRIM  = 1*TEXT-UTF8char *(*LWS TEXT-UTF8char)
TEXT-UTF8char   = %x21-7E / UTF8-NONASCII
UTF8-NONASCII   = %xC0-DF 1UTF8-CONT
                        / %xE0-EF 2UTF8-CONT
                        / %xF0-F7 3UTF8-CONT
                        / %xF8-Fb 4UTF8-CONT
                        / %xFC-FD 5UTF8-CONT
UTF8-CONT       = %x80-BF
```

在 TEXT-UTF8-TRIM 中，CRLF 作为头字段的附加部分。在解释 TEXT-UTF8- TRIM 值前，希望用一个 SP 替代折叠 LWS。

在几个协议元素中使用了十六进制数字字符。某些元素（认证）要求十六进制的首字母小写。

```
LHE          = DIGIT / %x61-66 ;                lowercase        a-f
```

许多 SIP 头字段值由 LWS 或特殊字符分隔的词组成。除非有特殊说明，token 的字母不区分大小写。这些特殊字符是参数值中的引用串。Word 结构在 Call-ID 中使用，允许使用很多分隔符。

```
Token            = 1*(alphanum / "-" / "." / "→" / "%" / "*"
                       / "_" / "+" / " ‘" / "’ " / "~" )
```

```
separators  = "(" / ")" / "<" / ">" / "@" /
                "," / ";" / ":" / "\\" / DQUOTE /
                "/" / "[" / "]" / "?" / "=" /
                "{" / "}" / SP / HTAB
word        = 1*(alphanum / "-" / "." / "→" / "%" / "*" /
                "_" / "+" / "? / "? / "? /
                "(" / ")" / "<" / ">" /
                ":" / "\\" / DQUOTE /
                "/" / "[" / "]" / "?" /
                "{" / "}" )
```

在元素间使用标记或分隔符时，在这些字符前后允许空格。

```
these characters:
STAR        = SWS "*" SWS ; asterisk
SLASH   = SWS "/" SWS ; slash
EQUAL   = SWS "=" SWS ; equal
LPAREN  = SWS "(" SWS ; left parenthesis
RPAREN  = SWS ")" SWS ; right parenthesis
RAQUOT  = ">" SWS ; right angle quote
LAQUOT  = SWS "<"; left angle quote
COMMA   = SWS "," SWS ; comma
SEMI        = SWS ";" SWS ; semicolon
COLON   = SWS ":" SWS ; colon
LDQUOT  = SWS DQUOTE; open double quotation mark
RDQUOT  = DQUOTE SWS ; close double quotation mark
```

SIP 头字段中可包含一些 Commnet，方法是用小括号将注释文本括起来。

只在字段值定义中包含 *comment* 的字段才允许有注释。在其它字段中，小括号都作为字段值的一部分。

```
user-unreserved     = "&" / "=" / "+" / "$" / "," / ";" / "?" / "/"
Comment             = LPAREN *(ctext / quoted-pair / comment) RPAREN
Ctext               = %x21-27 / %x2A-5B / %x5D-7E / UTF8-NONASCII
                    / LWS
```

ctext 包括除左括号、右括号和反斜杠之外的所有字符。如果文本串是用双引号标记的引用，那么该文本串作一个词解析。在引用串中，应忽略引号（“）和反斜杠（\）。

```
quoted-string       = SWS DQUOTE *(qdtext / quoted-pair ) DQUOTE
qdtext              = LWS / %x21 / %x23-5B / %x5D-7E
                    / UTF8-NONASCII
```

反斜杠（\）可作为引用串和注释结构中的单字符引用机制。与 HTTP/1.1 不同，字符 CR 和 LF 不能被忽略，以免与行折叠和头分隔相冲突。

```
quoted-pair          = "\\" (%x00-09 / %x0B-0C
                     / %x0E-7F)
SIP-URI                = "sip:" [ userinfo ] hostport
                     uri-parameters [ headers ]
SIPS-URI               = "sip:" [ userinfo ] hostport
                     uri-parameters [ headers ]
userinfo               = ( user / telephone-subscriber ) [ ":" password ] "@"
user                   = 1*( unreserved / escaped / user-unreserved )
user-unreserved      = "&" / "=" / "+" / "$" / "," / ";" / "?" / "/"
password               = *( unreserved / escaped /
                     "&" / "=" / "+" / "$" / "," )
hostport           = host [ ":" port ]
host               = hostname / IPv4address / IPv6reference
hostname           = *( domainlabel "." ) toplabel [ "." ]
domainlabe         = alphanum
                          / alphanum *( alphanum / "-" ) alphanum
toplabel           = ALPHA / ALPHA *( alphanum / "-" ) alphanum
IPv4address        =1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference          = "[" IPv6address "]"
IPv6address        = hexpart [ ":" IPv4address ]
Hexpart      = hexseq / hexseq "::" [ hexseq ] / "::" [ hexseq ]
Hexseq             = hex4 *( ":" hex4)
hex4               = 1*4HEXDIG
port               = 1*DIGIT
```

电话订阅用户的 BNF 见 RFC 2806 [8]。注意，在电话订阅用户的 BNF 中允许而在 SIP URI 的用户部分不允许的字符必须忽略。

```
uri-parameter          = *( ";" uri-parameter)
uri-parameter          = transport-param / user-param / method-param
                         / ttl-param / maddr-param / lr-param / other-param
transport-param = "transport="
                         ( "udp" / "tcp" / "sctp" / "tls"
                         / other-transport)
                         other-transport    = token
user-param         = "user=" ( "phone" / "ip" / other-user)
```

```
other-user      = token
method-param    = "method=" Method
ttl-param       = "ttl=" ttl
maddr-param     = "maddr=" host
lr-param        = "lr"
other-param     = pname [ "=" pvalue ]
pname           = 1*paramchar
pvalue          = 1*paramchar
paramchar       = param-unreserved / unreserved / escaped
param-unreserved    = "[" / "]" / "/" / ":" / "&" / "+" / "$"

headers         = "?" header *( "&" header )
header          = hname "=" hvalue
hname           = 1*( hnv-unreserved / unreserved / escaped )
hvalue          = *( hnv-unreserved / unreserved / escaped )
hnv-unreserved  = "[" / "]" / "/" / "?" / ":" / "+" / "$"

SIP-message     = Request / Response
Request     = Request-Line
                    *( message-header )
                    CRLF
                    [ message-body ]
Request-line        = Method SP Request-URI SP SIP-Version CRLF
Request-URI     = SIP-URI / SIPS-URI / absoluteURI
AbsoluteURI     = scheme ":" ( hier-part / opaque-part )
hier-part       = ( net-path / abs-path ) [ "?" query ]
net-path        = "//" authority [ abs-path ]
abs-path        = "/" path-segments
opaque-part     = uric-no-slash *uric
uric            = reserved / unreserved / escaped
uric-no-slash       = unreserved / escaped / ";" / "?" / ":" / "@"
                    / "&" / "=" / "+" / "$" / ","
path-segments   = segment *( "/" segment )
segment     = *pchar *( ";" param )
param           = *pchar
pchar           = unreserved / escaped /
```

```
                    ":" / "@" / "&" / "=" / "+" / "$" / ","
scheme          = ALPHA *( ALPHA / DIGIT / "+" / "-" / "." )
authority       = srvr / reg-name
srvr            = [ [ userinfo "@" ] hostport ]
reg-name        = 1*( unreserved / escaped / "$" / ","
                    / ";" / ":" / "@" / "&" / "=" / "+" )
query           = *uric
SIP-Version     = "SIP" "/" 1*DIGIT "." 1*DIGIT
message-header  = (Accept
                    / Accept-Encoding
                    / Accept-Language
                    / Alert-Info
                    / Allow
                    / Authentication-Info
                    / Authorization
                    / Call-ID
                    / Call-Info
                    / Contact
                    / Content-Disposition
                    / Content-Encoding
                    / Content-Language
                    / Content-Length
                    / Content-Type
                    / CSeq
                    / Date
                    / Error-Info
                    / Expires
                    / From
                    / In-Reply-To
                    / Max-Forwards
                    / MIME-Version
                    / Min-Expires
                    / Organization
                    / Priority
                    / Proxy-Authenticate
                    / Proxy-Authorization
```

```
                        / Proxy-Require
                        / Record-Route
                        / Reply-To
                        / Require
                        / Retry-After
                        / Route
                        / Server
                        / Subject
                        / Supported
                        / Timestamp
                        / To
                        / Unsupported
                        / User-Agent
                        / Via
                        / Warning
                        / WWW-Authenticate
                        / extension-header) CRLF

INVITEm     = %x49.4E.56.49.54.45 ; INVITE in caps
ACKm            = %x41.43.4B ; ACK in caps
OPTIONSm        = %x4F.50.54.49.4F.4E.53 ; OPTIONS in caps
BYEm            = %x42.59.45 ; BYE in caps
CANCELm     = %x43.41.4E.43.45.4C ; CANCEL in caps
REGISTERm       = %x52.45.47.49.53.54.45.52 ; REGISTER in caps
Method          = INVITEm / ACKm / OPTIONSm / BYEm
                        / CANCELm / REGISTERm
                        / extension-method
extension-method    = token
Response        =Status-Line
                        *( message-header )
                        CRLF
                        [ message-body ]

Status-Line             = SIP-Version SP Status-Code SP Reason-Phrase CRLF
Status-Code             = Informational
                        /                       Redirection
```

```
                    /                       Success
                    /                       Client-Error
                    /                       Server-Error
                    /                       Global-Failure
                    /                        extension-code
extension-code  = 3DIGIT
Reason-Phrase   = *(reserved / unreserved / escaped
                    /UTF8-NONASCII/UTF8-CONT/SP/HTAB)

Informational          = "100" ; Trying
                    / "180" ; Ringing
                    / "181" ; Call Is Being Forwarded
                    / "182" ; Queued
                    / "183" ; Session Progress

Success     = "200" ; OK

Redirection     ="300" ; Multiple Choices
                    / "301" ; Moved Permanently
                    / "302" ; Moved Temporarily
                    / "305" ; Use Proxy
                    / "380" ; Alternative Service

Client-Error        = "400" ; Bad Request
                    / "401" ; Unauthorized
                    /             "402" ; Payment Required
                    /             "403" ; Forbidden
                    /             "404" ; Not Found
                    /              "405" ; Method Not Allowed
                    /             "406" ; Not Acceptable
                    /             "407" ; Proxy Authentication Required
                    /              "408" ; Request Timeout
                    /              "410" ; Gone
                    /              "413" ; Request Entity Too Large
                    /              "414" ; Request-URI Too Large
                    /              "415" ; Unsupported Media Type
```

```
                    /              "416" ; Unsupported URI Scheme
                    /              "420" ; Bad Extension
                    /              "421" ; Extension Required
                    /              "423" ; Interval Too Brief
                    /               "480" ; Temporarily not available
                    /               "481" ; Call Leg/Transaction Does Not
               Exist
                    /               "482" ; Loop Detected
                    /               "483" ; Too Many Hops
                    /               "484" ; Address Incomplete
                    /               "485" ; Ambiguous
                    /               "486" ; Busy Here
                    /               "487" ; Request Terminated
                    /               "488" ; Not Acceptable Here
                    /               "491" ; Request Pending
                    /               "493" ; Undecipherable
Server-Error        = "500" ; Internal Server Error
                    / "501" ; Not Implemented
                    / "502" ; Bad Gateway
                    / "503" ; Service Unavailable
                    / "504" ; Server Time-out
                    / "505" ; SIP Version not supported
                    / "513" ; Message Too Large

Global-Failure  = "600" ; Busy Everywhere
                    / "603" ; Decline
                    / "604" ; Does not exist anywhere
                    /"606" ; Not Acceptable

Accept          = "Accept" HCOLON
                        [ accept-range *(COMMA accept-range) ]
accept-range        = media-range *(SEMI accept-param)
media-range     = ( "*/*"
                    / ( m-type SLASH "*" )
                    / ( m-type SLASH m-subtype )
                    ) *( SEMI m-parameter )
```

```
accept-param        = ("q" EQUAL qvalue) / generic-param
qvalue          = ( "0" [ "." 0*3DIGIT ] )
                    / ( "1" [ "." 0*3("0") ] )
generic-param   = token [ EQUAL gen-value ]
gen-value       = token / host / quoted-string
Accept-Encoding = "Accept-Encoding" HCOLON
Encoding        = codings *(SEMI accept-param)
Codings     = content-coding / "*"
content-coding  = token

Accept-Language         = "Accept-Language" HCOLON
                     [ language *(COMMA language) ]
language        = language-range *(SEMI accept-param)
language-range  = ( ( 1*8ALPHA *( "-" 1*8ALPHA ) ) / "*" )

Alert-Info      = "Alert-Info" HCOLON alert-param *(COMMA alert-param)
alert-param     = LAQUOT absoluteURI RAQUOT *( SEMI generic-param )

Allow           = "Allow" HCOLON [Method *(COMMA Method)]

Authorization       = "Authorization" HCOLON credentials
Credentials     = ("Digest" LWS digest-response)
                    / other-response
digest-response = dig-resp *(COMMA dig-resp)
dig-resp        = username / realm / nonce / digest-uri
                            / dresponse / algorithm / cnonce
                            / opaque / message-qop
                            / nonce-count / auth-param
username        = "username" EQUAL username-value
username-value  = quoted-string
digest-uri      = "uri" EQUAL LDQUOT digest-uri-value RDQUOT
digest-uri-value    = rquest-uri ; Equal to request-uri as specified
                     by HTTP/1.1
message-qop     = "qop" EQUAL qop-value
cnonce          = "cnonce" EQUAL cnonce-value
cnonce-value        = nonce-value
```

```
nonce-count     ="nc" EQUAL nc-value
nc-value        = 8LHEX
dresponse       ="response" EQUAL request-digest
request-digest      = LDQUOT 32LHEX RDQUOT
auth-param      = auth-param-name EQUAL
                            ( token / quoted-string )
auth-param-name = token
other-response  = auth-scheme LWS auth-param
                      *(COMMA auth-param)
auth-scheme     = token

Authentication-Info = "Authentication-Info" HCOLON ainfo
                      *(COMMA ainfo)
ainfo           = nextnonce / message-qop
                          / response-auth / cnonce
                          / nonce-count
nextnonce       = "nextnonce" EQUAL nonce-value
response-auth       = "rspauth" EQUAL response-digest
response-digest = LDQUOT *LHEX RDQUOT
Call-ID         = ( "Call-ID" / "i" ) HCOLON callid
Called          = word [ "@" word ]

Call-Info       = ( "Call-ID" / "i" ) HCOLON callid
Info            = LAQUOT absoluteURI RAQUOT *( SEMI info-param)
info-param      = ( "purpose" EQUAL ( "icon" / "info"
                       / "card" / token ) ) / generic-param

Contact         = ("Contact" / "m" ) HCOLON
                     ( STAR / (contact-param *(COMMA contact-param)))
contact-param   = (name-addr / addr-spec) *(SEMI contact-params)
name-addr       = [ display-name ] LAQUOT addr-spec RAQUOT
addr-spec       = SIP-URI / SIPS-URI / absoluteURI
display-name        = *(token LWS)/ quoted-string

contact-params  = c-p-q / c-p-expires
                     / contact-extension
```

```
c-p-q           = "q" EQUAL qvalue
c-p-expires     = "expires" EQUAL delta-seconds
contact-extension   = generic-param
delta-seconds       = 1*DIGIT

Content-Disposition = "Content-Disposition" HCOLON
                        disp-type *( SEMI disp-param )
disp-type       = "render" / "session" / "icon" / "alert"
                        / disp-extension-token
disp-param      = handling-param / generic-param
handling-param  = "handling" EQUAL
                        ( "optional" / "required"
                        / other-handling )
other-handling  = token
disp-extension-token= token

content-Encoding    = ( "Content-Encoding" / "e" ) HCOLON
                      content-coding *(COMMA content-coding)

content-Language    = "Content-Language" HCOLON
                      language-tag *(COMMA language-tag)
language-tag        = primary-tag *( "-" subtag )
primary-tag     = 1*8ALPHA
subtag          = 1*8ALPHA

Content-Length  = ( "Content-Length" / "l" ) HCOLON 1*DIGIT
Content-Type        = ( "Content-Type" / "c" ) HCOLON media-type
media-type      = m-type SLASH m-subtype *(SEMI m-parameter)
m-type          = discrete-type / composite-type
discrete-type       = "text" / "image" / "audio" / "video"
                      / "application" / extension-token
composite-type  = "message" / "multipart" / extension-token
extension-token = ietf-token / x-token
ietf-token      = token
x-token         = "x-" token
m-subtype       = extension-token / iana-token
```

```
iana-token      = token
m-parameter     = m-attribute EQUAL m-value
m-attribute     = token
m-value     = token / quoted-string

Cseq            = "CSeq" HCOLON 1*DIGIT LWS Method

Date            = "Date" HCOLON SIP-date
SIP-date        = rfc1123-date
rfc1123-date        = wkday "," SP date1 SP time SP "GMT"
date1           = 2DIGIT SP month SP 4DIGIT
                      ; day month year (e.g., 02 Jun 1982)
time            = 2DIGIT ":" 2DIGIT ":" 2DIGIT
                      ; 00:00:00 - 23:59:59
wkday           = "Mon" / "Tue" / "Wed"
                      / "Thu" / "Fri" / "Sat" / "Sun"
month           = "Jan" / "Feb" / "Mar" / "Apr"
                      / "May" / "Jun" / "Jul" / "Aug"
                      / "Sep" / "Oct" / "Nov" / "Dec"

Error-Info      = "Error-Info" HCOLON error-uri *(COMMA error-uri)
error-uri       = LAQUOT absoluteURI RAQUOT *( SEMI generic-param )
Expires         = "Expires" HCOLON delta-seconds
From            = ( "From" / "f" ) HCOLON from-spec
from-spec       = ( name-addr / addr-spec )
                     *( SEMI from-param )
from-param      = tag-param / generic-param
tag-param       = "tag" EQUAL token

In-Reply-To     = "In-Reply-To" HCOLON callid *(COMMA callid)

Max-Forwards    = "Max-Forwards" HCOLON 1*DIGIT

MIME-Version    = "MIME-Version" HCOLON 1*DIGIT "." 1*DIGIT

Min-Expires     = "Min-Expires" HCOLON delta-seconds
```

```
Organization        = "Organization" HCOLON [TEXT-UTF8-TRIM]

Priority            = "Priority" HCOLON priority-value
priority-value      = "emergency" / "urgent" / "normal"
                      / "non-urgent" / other-priority
other-priority      = token

Proxy-Authenticate  = "Proxy-Authenticate" HCOLON challenge
Challenge       = ("Digest" LWS digest-cln *(COMMA digest-cln))
                      /other-challenge
other-challenge = auth-scheme LWS auth-param
                      *(COMMA auth-param)
digest-cln      = realm / domain / nonce
                      / opaque / stale / algorithm
                      / qop-options / auth-param
realm           = "realm" EQUAL realm-value
realm-value     = quoted-string
domain          = "domain" EQUAL LDQUOT URI
                      *( 1*SP URI ) RDQUOT
URI         = absoluteURI / abs-path
Nonce           = "nonce" EQUAL nonce-value
nonce-value     = quoted-string
opaque          = "opaque" EQUAL quoted-string
stale           = "stale" EQUAL ( "true" / "false" )
algorithm       = "algorithm" EQUAL ( "MD5" / "MD5-sess"
                      / token )
qop-options     = "qop" EQUAL LDQUOT qop-value
                      *("," qop-value) RDQUOT
qop-value       = "auth" / "auth-int" / token
proxy-Authorization = "Proxy-Authorization" HCOLON credentials

proxy-Require   = "Proxy-Require" HCOLON option-tag
                      *(COMMA option-tag)
option-tag      = token
```

```
record-Route        = "Record-Route" HCOLON rec-route *(COMMA rec-route)
rec-route       = name-addr *( SEMI rr-param )
rr-param        = generic-param

Reply-To        = "Reply-To" HCOLON rplyto-spec
rplyto-spec     = ( name-addr / addr-spec )
                    *( SEMI rplyto-param )
rplyto-param        = generic-param
Require     = "Require" HCOLON option-tag *(COMMA option-tag)

Retry-After     = "Retry-After" HCOLON delta-seconds
                    [ comment ] *( SEMI retry-param )

retry-param     = ("duration" EQUAL delta-seconds)
                    / generic-param

Route           = "Route" HCOLON route-param *(COMMA route-param)
route-param     = name-addr *( SEMI rr-param )

Server          = "Server" HCOLON server-val *(LWS server-val)
server-val      = product / comment
product         = token [SLASH product-version]
product-version = token

Subject         = ( "Subject" / "s" ) HCOLON [TEXT-UTF8-TRIM]
Supported       = ( "Supported" / "k" ) HCOLON
                    [option-tag *(COMMA option-tag)]

Timestamp       = "Timestamp" HCOLON 1*(DIGIT)
                    [ "." *(DIGIT) ] [ LWS delay ]
delay           = *(DIGIT) [ "." *(DIGIT) ]

To          = ( "To" / "t" ) HCOLON ( name-addr
                    / addr-spec ) *( SEMI to-param )
to-param        = tag-param / generic-param
```

```
Unsupported     = "Unsupported" HCOLON option-tag *(COMMA option-tag)
User-Agent      = "User-Agent" HCOLON server-val *(LWS server-val)

Via         = ( "Via" / "v" ) HCOLON via-parm *(COMMA via-parm)
via-parm        = sent-protocol LWS sent-by *( SEMI via-params )
via-params      = via-ttl / via-maddr
                    / via-received / via-branch
                    / via-extension
via-ttl         = "ttl" EQUAL ttl
via-maddr       = "maddr" EQUAL host
via-received        = "received" EQUAL (IPv4address / IPv6address)
via-branch      = "branch" EQUAL token
via-extension       = generic-param
sent-protocol       = protocol-name SLASH protocol-version
                     SLASH transport
protocol-name   = "SIP" / token
protocol-version    = token
transport       = "UDP" / "TCP" / "TLS" / "SCTP"
                    / other-transport
sent-by         = host [ COLON port ]
ttl         = 1*3DIGIT ; 0 to 255

Warning     = "Warning" HCOLON warning-value *(COMMA warning-value)
warning-value   = warn-code SP warn-agent SP warn-text
warn-code       = 3DIGIT
warn-agent      = hostport / pseudonym
                    ; the name or pseudonym of the server adding
                    ; the Warning header, for use in debugging
warn-text       =quoted-string
pseudonym       = token

WWW-Authenticate    = "WWW-Authenticate" HCOLON challenge

extension-header    = header-name HCOLON header-value
header-name     = token
header-value        = *(TEXT-UTF8char / UTF8-CONT / LWS)
```

message-body = *OCTET

26 安全考虑:威胁模型和安全用法推荐

SIP 不是一个很容易实现安全的协议。媒介的使用、多面的信任关系，在完全不信任的元素间的期望用法以及用户之间的操作，都使得安全问题非常重要。今天，在广泛变化的环境和用法中，需要在不必大量协调的情况下部署安全解决方案。为了满足这些不同的需要，需要一些适用于 SIP 不同方面和用法的特殊机制。

注意，SIP 信令本身的安全性和与 SIP 一起使用的协议（比如 RTP）的安全性无关，也与 SIP 可能携带的任何消息体的安全隐含无关（尽管 MIME 安全在保护 SIP 中起到重要的作用）。任何与会话相关的媒体都能独立于任何相关的 SIP 信令而进行端对端加密。本文档没有描述媒体加密。

首先，检查一些传统威胁模式的考虑是，明确地确定 SIP 安全的需求。然后，通过解释可用于提供这些服务的安全机制，为安全服务详细说明这些威胁。随后，列举了对 SIP 实现者的需求，并列举了一些典型的部署——在这些部署中的安全机制可用来提高 SIP 的安全。本章最后说明了保密性。

26.1 攻击和威胁模型

本章详细描述了多数 SIP 部署的一些常见威胁。我们用这些威胁来解释 SIP 所需的每个安全服务。

下面例子并不能详尽地列出 SIP 受到的所有威胁，但这些“传统”的威胁论证了需要特殊安全服务，它们能潜在地阻止各种威胁。

这里假想一个环境：在这个环境里，攻击者可潜在地在网络上读取任何信息包——可想象在公网上频繁的使用 SIP。攻击者还可以更改包中的内容（可能在一些泄密的媒介上）。攻击者可能希望偷取服务、窃听通信或破坏会话。

26.1.1 注册欺骗

SIP 注册机制允许用户代理向注册服务器识别自己，注册服务器是用于定位用户（由记录地址指定）的一个设备。注册服务器评估 REGISTER 消息的 From 头字段中声称的身份，确定请求是否能修改 To 头字段中与记录地址相关联的联系地址。如果这两个字段相同，就会有许多有效部署，第三方就能代表用户注册联系。

UA 的所有者可任意修改 SIP 请求的头字段，这就为恶意注册开了后门。攻击者可以成功地模仿成授权更改与记录地址相关的 contact 的一方，例如，攻击者可以取消一个 URI 的所有现有联系的注册，然后注册自己的设备作为相应的联系地址，从而将被攻击用户的所有请求直接发送到攻击者的设备。

此威胁是威胁家族的一种，该威胁家族依赖于请求发起者缺乏密码担保。代表有价值服务（例如，使 SIP 请求与传统电话呼叫互通的网关）的任何 SIP UAS，都可以通过对接收的请求进行认证的方式，控制对其资源的访问。甚至终端用户 UA，例如 SIP 电话都希望能确定请求发起者的身份。

这种威胁说明了对安全服务的需求：使 SIP 实体能够认证请求的发起者。

26.1.2 冒充服务器

通常在 Request-URI 中指定请求的目的域。为发送请求，UA 通常直接与域中的服务器联系。然而，也可能有攻击者冒充远程服务器，这样 UA 的请求就会被其它方截取。

例如，考虑这样一种情况，在 chicago.com 域的重定向服务器冒充 biloxi.com 域的重定向服务器。如果用户代理向 biloxi.com 发送请求，但是在 chicago.com 域的重定向服务器用一个伪造的响应应答，该伪造的响应具有 biloxi.com 域响应的相应的 SIP 头字段。重定向响应中伪造的联系地址会使源 UA 指向一个不适当或不安全的资源，或阻止 biloxi.com 随后的请求。

这种威胁家族拥有大量的成员，许多都是非常危险的。与注册黑客威胁相反，考虑这样一种情况， chicago.com 截获向 biloxi.com 发送的注册信息，chicago.com 用伪造的 301（永久删除）响应，应答截获的注册。该应答看起来像是来自 biloxi.com，实际是来自于 chicago.com。以后来自 UA 的所有 REGISTER 请求都会发到 chicago.com。

为防止这种威胁，需要 UA 能对接收其请求的服务器认证。

26.1.3 篡改消息体

当然，SIP UA 通过可信的代理服务器发送请求。不考虑这种信任是如何建立的（本节的其它部分会讨论代理认证），UA 可以信任代理服务器以发送请求，但是并不检查或更改包含在请求中的消息体。

考虑这样一种情况，UA 用 SIP 消息体为媒体会话交换会话加密密钥。尽管 UA 信任其传递信令的域的代理服务器，但它不希望这个域的管理员解密随后的媒体会话。更糟的是如果代理服务器是恶意的，它就会修改会话密钥，发起拦截式攻击或改变源 UA 请求安全特性。

威胁家族不仅对会话密钥构成威胁，还对大多数 SIP 端对端携带的可想象的内容格式构成威胁。包括可能传递给用户的 MIME 消息体、SDP 或封装的电话信号等。攻击者会试图修改 SDP 消息体，例如，为窃听随后的语音通信，而将 RTP 媒体流指向一个窃听设备。

还要注意到，SIP 中的一些头字段意味着端对端，如 Subject。UA 可能会保护这些头字段和消息体（例如，可以使一个重要的请求看上去好像垃圾邮件一样）。然而，在发送请求时，代理服务器已对许多头字段进行了合理的检查和更改，所以并非所有头字段都需要端对端的安全保护。

因此，UA 可能希望端对端地保护 SIP 消息体，有时候还希望保护头字段。消息体需要的安全服务有机密性、完整性和认证。这些端对端的服务应与为确保与媒介（如代理服务器）交互所使用的安全方式无关。

26.1.4 中断会话

一旦通过最初的消息建立起对话，就可以发送修改对话和/或会话状态的请求。确保这些请求不被攻击者伪造非常重要。这就需要保证请求不被攻击者伪造篡改。

考虑这样一种情况，第三方为了获取会话参数（To 标签、 From 标签等）而捕获到了两方对话的最初信息，然后在会话中插入 BYE 请求。攻击者可以选择假冒两方中的一方，使得发送的请求看起来像是它们中的一方发送的。一旦其目标接收到 BYE 请求，会话就会提前结束。

类似的会话中威胁还有传送更改会话的伪造 re-INVITE（作为窃听攻击的一部分，还包括降低会话的安全或媒体流的重定向）。

威胁家族不仅对会话密钥构成威胁，还对大多数 SIP 端对端携带的可想象的内容格式构成威胁。包括可能传递给用户的 MIME 消息体、SDP 或封装的电话信号等。攻击者会试图修改 SDP 消息体，例如，为窃听随后的语音通信，而将 RTP 媒体流指向一个窃听设备。

还要注意到，SIP 中的一些头字段意味着端对端，如 Subject。UA 可能会保护这些头字段和消息体（例如，可以使一个重要的请求看上去好像垃圾邮件一样）。然而，在发送请求时，代理服务器已对许多头字段进行了合理的检查和更改，所以并非所有头字段都需要端对端的安全保护。

因此，UA 可能希望端对端地保护 SIP 消息体，有时候还希望保护头字段。消息体需要的安全服务有机密性、完整性和认证。这些端对端的服务应与为确保与媒介（如代理服务器）交互所使用的安全方式无关。

26.1.5 中断会话

一旦通过最初的消息建立起对话，就可以发送修改对话和/或会话状态的请求。确保这些请求不被攻击者伪造非常重要。这就需要保证请求不被攻击者伪造篡改。

考虑这样一种情况，第三方为了获取会话参数（To 标签、 From 标签等）而捕获到了两方对话的最初信息，然后在会话中插入 BYE 请求。攻击者可以选择假冒两方中的一方，使得发送的请求看起来像是它们中的一方发送的。一旦其目标接收到 BYE 请求，会话就会提前结束。

类似的会话中威胁还有传送更改会话的伪造 re-INVITE（作为窃听攻击的一部分，还包括降低会话的安全或媒体流的重定向）。

如果注册服务器没有对 REGISTER 请求进行正确的认证和授权，就会产生许多拒绝服务攻击。攻击者可以取消一个管理域内一些或所有用户的注册，这样就阻止了新会话对这些用户的邀请。攻击者还可以注册大量 contact，为一个给定的记录地址指定同一主机，以便在拒绝服务攻击中把注册服务器和相关的代理服务器用作放大器。攻击者还可以通过注册大量的绑定，来耗尽注册服务器的可用内存和硬盘资源。

使用多播方式传送 SIP 请求极大地增加了拒绝服务攻击的机会。

以上问题说明了定义减少拒绝服务威胁的架构的需求，并且安全机制应对这一类型的攻击引起重视。

26.2 安全机制

从上述威胁来看，我们总结出，SIP 协议所需的基本安全服务有：保护消息的机密性和完整性，防止重放攻击和消息欺骗，在会话中为参与者提供认证和保密，并防止拒绝服务攻击。某些 SIP 消息体需要机密性、完整性和认证等一些安全服务。

不用为 SIP 专门定义新的安全机制，SIP 无论在什么情况下都可以重用 HTTP 和 SMTP 空间可能的现有安全模型。

全消息加密是保护信令机密性的最好方式——它同时可确保任何恶意媒介不会修改消息。然而，不能对整个 SIP 请求和响应进行简单的端对端加密，因为在多数网络架构中，代理对消息字段如 Request-URI、Route 和 Via 必须是可见的，这样才能够正确发送 SIP 请求。同时，注意，代理服务器需要修改消息的一些特征（例如，增加 Via 头字段值）。在某种程度上，SIP UA 必须信任代理服务器。因此，建议 SIP 使用低层的安全机制，该机制在线路上以逐跳为单位对整个 SIP 请求或响应加密，并且，允许终端验证接收其请求的代理服务器

的身份。

SIP 实体还需要以一种安全的方式相互认证。当一个 SIP 端点向对等 UA 或代理服务器声称用户的身份时，该身份应该是可认证的。在 SIP 中提供一种密码认证机制来阐述这种需求。

SIP 消息体的独立的安全机制提供了端对端的相互认证的可选方式，也提供了用户代理必须信任媒介的限度。

26.2.1 传输和网络层安全

传送层和网络层安全对信令流加密，确保消息的机密性和完整性。

通常在建立低层安全时使用证书，在许多架构中，这些证书也可用来提供一种认证方式。

用于在网络层和传输层提供安全的两种可选方式分别是 TLS[24]和 IPSec[25]。

IPSec 是网络层协议工具集，可用作传统 IP（Internet 协议）的安全替代。IPSec 通常用在这样的架构中：主机和管理域间存在相互的信任关系。通常在主机的操作系统级实现 IPSec，或在安全网关上实现，该安全网关为它接收的来自特殊接口的所有流量提供机密性和完整性。或者在一个安全网关上为从专门的接口接收到信息提供机密性和完整性。同样可以基于逐跳使用 IPSec。

在多数架构中不需要将 IPSec 与 SIP 集成；IPSec 也许是最合适这种配置的，在这种配置中直接向 SIP 主机增加安全是困难的。与第一跳代理服务器有预先共享的键控关系的 UA 也非常适合使用 IPSec。SIP 的任何 IPSec 布署都需要一个 IPSec 概要，该概要描述了保护 SIP 安全所需的协议工具。本文档没给出这样的概要。

TLS 在面向连接的协议之上提供了传输层安全（如本文档中的 TCP）；可以规定“tls”作为在 via 头字段值或者 SIP-URI 中的期望的传输层协议。“tls”（表示 TCP 上的 TLS）最适合在这样的架构中使用：在主机间没有预有的信任关系，但在主机间需要逐跳安全。例如，Alice 信任其本地代理服务器，在交换证书后，Alice 的本地代理服务器确定信任 Bob 信任的本地代理服务器，因此，Bob 和 Alice 就可以安全地进行通信了。

TLS 必须与 SIP 应用紧密耦合。注意在 SIP 中基于逐跳规定传输机制。因此，在 TLS 上向代理服务器发送请求的 UA 不能保证端对端使用 TLS。

当 SIP 应用中使用 TLS 时，实现者必须最少支持 TLS_RSA_WITH_AES_128_CBC_SHA 密码组[26]。为向后兼容，代理服务器、重定向服务器和注册服务器必须支持 TLS_RSA_WITH_3DES_EDE_CBC_SHA。实现者也可以支持任何其他密码组。

26.2.2 SIPS URI 模式

尽管模式字符串是“sips”而不是“sip”，SIPS URI 模式仍然遵从 SIP URI 的语法（见第 19 章）。SIPS 的语义与 SIP URI 不同。SIPS 允许资源说明它们可以安全到达。

SIPS URI 可用作具体用户的记录地址——用户知道该 URI（在它们的名片上、在请求的 From 头字段中、在 REGISTER 请求的 To 头字段中）。当用作请求的 Request-URI 时，SIPS 模式表示从转发请求的每个跳跃点，到请求到达的负责 Request-URI 域部分的 SIP 实体，都必须用 TLS 保护其安全。当消息到达预期的域时，就会根据本地安全和路由策略对其进行处理，很可能在到达 UAS 的最后一跳中使用 TLS。当请求的发起者使用 SIPS URI 时（在目标的记录地址中配置了 SIPS URI 的情况），SIPS 指定到目标域的整个请求路径都要实现安全。

其他许多方式可以应用 SIPS 模式，除用于 Request-URI 外，目前还可以在 SIP 中使用 SIP URI，包括记录地址、联系地址（Contact 头内容，包括 REGISTER 方法）和 Route 头。

在每种情况下，SIPS URI 模式允许这些现有的字段指定安全资源。在任何关联中，废除 SIPS URI 的方式有其自身的安全属性，详细描述见[4]。

使用 SIPS 特别要求部署 TLS 的相互认证，就像需要密码组 TLS-RSA-WITH-AES-128-CBC-SHA 一样。在认证过程中接收的证书须由客户端持有的根证书进行验证，倘若证书认证失败，那么请求就会失败。

请注意，在 SIPS URI 模式中，传送与 TLS 协议无关，因此 sips:alice@atlanta.com;transport=tcp 和 sips:alice@atlanta.com;transport=sctp 是有效的，（尽管 UDP 对于 SIPS 来说不是一个有效的传送协议）。不赞成使用 transport=tls，部分原因是它是特定于请求的一跳。这是对 RFC2543 的一个改变。

分发 SIPS URI 作为记录地址的用户，可以选择运行一些设备，这些设备拒绝不安全传送的请求。

26.2.3 HTTP 认证

基于 HTTP 认证，SIP 提供了挑战能力，HTTP 认证取决于 401 和 407 响应代码以及携带挑战和凭证的头字段。不需要进行较大的改动，在 SIP 内重用 HTTP 摘要验证模式考虑了重放保护和单向认证。

22 章详细介绍了 SIP 摘要验证的用法。

26.2.4 S/MIME

如上所述，为了保密性而端对端地对 SIP 消息完全加密的方式是不适当的，因为网络媒介（例如代理服务器）需要查看某些头字段以正确地转发消息，并且，如果把这些媒介排除在安全相关考虑之外，那么 SIP 消息实质上是不可路由的。

然而，S/MIME 允许 SIP UA 在 SIP 内对 MIME 消息体进行加密，保证这些消息体端对端地安全，而不影响消息头。S/MME 不仅可以为消息体提供端对端的完整性和保密性，也可以提供相互认证。也可以使用 S/MME 通过 SIP 消息隧道为 SIP 头字段提供完整性和机密性。

23 章详细介绍了 SIP 中 S/MIME 的用法。

26.3 实现安全机制

26.3.1 对 SIP 实现者的要求

代理服务器、重定向服务器和注册服务器都必须实现 TLS，并且都必须支持相互认证和单向认证。强烈推荐 UA 可以启动 TLS；UA 也可以作为 TLS 服务器。代理服务器、重定向服务器和注册服务器应该持有站点证书——其主体与服务器规范的主机名一致。UA 可以有其自己和 TLS 相互认证的证书，在本文档没有阐明其使用。所有支持 TLS 的 SIP 元素必须有一种机制来确认在 TLS 协商中的证书。这就要求证书中心发布一个或多个根证书（尤其是可与发布网站浏览器根证书相比的众所周知的站点证书的分布）。

所有支持 TLS 的 SIP 元素必须支持 SIPS URI 模式。

代理服务器、重定向服务器、注册服务器和 UA 同样可以实现 IPSec 或者其它低层安全协议。

当 UA 试图联系代理服务器、重定向服务器或者注册服务器时，UAC 应该启动一个 TLS 连接，并通过此连接发送 SIP 消息。在一些架构中，UAS 还可以通过这样的 TLS 连接接收请

求。

代理服务器、重定向服务器、注册服务器和 UA 必须围绕第 22 章要求的所有方面实现摘要验证。代理服务器、重定向服务器和注册服务器至少要配置一个摘要域，同时给定的服务器支持的 realm 字符串至少应该有一个字符串与服务器的主机名或域名一致。

UA 可支持 MIME 消息体的签名和加密以及第 23 章所介绍的带 S/MIME 的凭证转移。如果 UA 为了确认 TLS 或 IPSec 的证书，而持有一个或多个认证中心的根证书，那么，在适当的时候，它应该可以重用这些证书来验证 S/MIME 证书。UA 可以持有特定用来验证 S/MIME 证书的根证书。

注意，可以预料的是，将来的安全扩展可以升级到与 S/MIME 相关联的标准长度，这样当 S/MIME 实现出现时，问题空间就变得易于理解。

26.3.2 安全解决方案

在某种程度上，这些安全机制的操作都可参照已有的 web 和 email 安全模型。在较高级别上，UA 会使用摘要的用户名和密码向服务器（代理服务器、重定向服务器和注册服务器）进行自验证，服务器使用 TLS 传递的站点证书向 UA 单跳离开或另外一个服务器单跳离开(反之亦然）进行自验证。

在对等的级别，通常，UA 相信网络会对另一方进行验证，但是，如果网络没有验证，或者网络本身就不可信，那么，S/MIME 可以用来提供直接的验证。

下面是一个说明性的实例，在此实例中，不同的 UA 和服务器使用这些安全机制来防止第 26.1 节所描述的那种威胁。当然，实现者和网络管理员可以按照本章的剩余部分给出的标准指南，这只作为实现的实例。

Registration 当 UA 在线，在其本地管理域注册时，就可以与它的注册服务器建立 TLS 连接（第 10 章介绍了 UA 怎样到达其注册服务器)。注册服务器为 UA 提供证书，并且，此证书标识的站点必须与 UA 要进行注册的域相符合；例如，如果 UA 想要注册的记录地址为 alice@atlanta.com，那么站点证书必须标识域中的主机（如 sip.atlanta.com)。当它接收到 TLS 证书消息时，UA 就会验证证书，并检查证书标识的站点。如果证书无效、吊销或者不能标识合适的当事人，那么 UA 就不能发送 REGISTER 消息，否则将继续注册。

当注册服务器提供有效证书时，UA 知道该注册服务器不是对 UA 进行重定向、盗窃口令或试图一些类似攻击的攻击者。

随后 UA 创建一个 REGISTER 请求，应该说明注册服务器接收的站点证书对应的 Request-URI。当 UA 在已有的 TLS 连接上发送 REGISTER 请求时，注册服务器应该挑战请求，返回 401(需要代理认证)响应。响应的 Proxy-Authenticate 头字段的 realm 参数应该与前面站点证书给出的域一致。当 UAC 接收到此挑战，它就应该提示用户出示凭证，或者从挑战的 realm 参数对应的密钥环中取出适当的凭证。证书的用户名应该和 REGISTER 请求 To 头字段 URI 的 USERINFO 部分一致。一旦摘要凭证插入到合适的 Proxy-Authenticate 头字段，REGISTER 就应该再提交给注册服务器。

由于注册服务器要求用户代理对其进行认证，所以这使攻击者难于伪造用户记录地址的 REGISTER 请求。要注意的是，由于 REGISTER 通过机密的 TLS 连接发送，所以攻击者不能够截获 REGISTER，记录凭证用于可能的重放攻击。

如果注册服务器也作为代理服务器，那么，一旦注册服务器接受了注册，UA 就应该将 TLS 连接开放给代理服务器，此管理域的用户可以向它发送请求。为了给刚完成注册的 UA 传递入站的请求，将会重用已有的 TLS 连接。

由于 UA 已经在 TLS 连接的另一边对服务器进行认证，通过此连接的所有请求通过代理服务器、为其所知地传输，所以，攻击者不能创建通过该代理服务器发送的欺骗请求。

Interdomain Requests 现在我们说，Alice 的 UA 想发起与在远程管理域的、名为 bob@biloxi.com 的用户的会话。我们就可以说本地管理域（atlanta.com）具有一个本地的带外代理。

处理管理域带内请求的代理服务器也可以作为本地带外代理。为了简单，我们认为情况如此（否则，用户代理将为此点的服务器发起新的 TLS 连接）。假定用户完成了上述的注册过程后，当它发送 INVITE 请求给其它用户时，它可以重用本地代理服务器的 TLS。UA 应该重用 INVITE 中缓存的凭证，避免对用户不必要的提示。

当本地带外代理确认了 UA 在 INVITE 中表示的凭证，它应该检查 Request-URI，以确定转发消息的方式（见[4]）。如果 Request-URI 的 domainname 部分与本地域(Atlanta.com)而不是 biloxi.com 一致，那么代理服务器就会咨询其定位服务，确定怎样最好地到达请求的用户。

假若 alice@atlanta.com 试图联系 alex@atlanta.com，那么本地代理将该请求代理到 Alex 注册时与注册服务器建立的 TLS 连接上。由于 Alex 通过他的认证渠道接收该请求，所以，他确信 Alice 的请求已经通过当地管理域的代理服务器进行过验证。

然而，在本例中，Request-URI 指定了一个远程域。本地带外代理服务器应该和远程代理服务器（在 biloxi.xom 上）建立一个 TLS 连接。连接的每一方都要验证和检查对方的证书，注意在证书中出现的域名，将其与 SIP 消息头字段进行比较。例如，代理服务器在这个阶段就应验证从 biloxi.xom 域的远端接收到的证书。一旦完成了验证，并查询两个代理间的安全信道完成 TLS 协商，那么，代理服务器可以向 biloxi.com 转发 INVITE 请求。

biloxi.com 域的代理服务器应该依次检查代理服务器的证书，并将证书中声明的域与 INVITE 请求的 From 头字段的 domainname 部分相比较。Biloxi 代理服务器有非常严格的安全策略，要求它对不是它所代理的管理域发来的请求一概拒绝。

建立一些这样的安全策略，以防止产生垃圾邮件的 SMTP 开路延迟的 SIP 对等物。

然而，此策略仅保证来自归于它自己域的请求；它不允许 bioxi.com 了解怎样认证 Alice。仅当 bioxi.com 有其它的方式获知认证策略时，它才可能了解 Alice 怎样证明自己的身份。bioxi.com 可能建立更严格的策略，禁止来自未知域的请求和 bioxi.com 共享同一认证策略。

一旦 bioxi 的代理批准了 INVITE，代理服务器应该识别与此请求的目标用户（在此是 bob@biloxi.com）相关的、已有的 TLS 信道（如果有的话）。因为请求是通过 biloxi 代理已经认证的 TLS 连接接收的，尽管信任 Alice 的身份不是必要的，Bob 知道没有篡改 From 头字段，并且 atlanta.com 已经确认了 Alice。

在它们转发请求之前，代理服务器都应该在请求中添加一个 Record-Route 头字段，以便于此对话中所有将来请求都通过代理服务器。这样，代理服务器就可以继续在此对话的生命周期中提供安全服务。如果代理服务器没有自己添加 Record-Route 头字段，那么，将来消息将没有任何安全服务直接在 Alice 和 Bob 之间端对端地通过(除非双方同意在一些独立的端对端安全上，如 S/MIME)。在这方面，梯形模型可以提供一个很好的结构，在这里，站点代理之间的协商约定可以在 Alice 和 Bob 之间提供一个合理的安全信道。

例如，欲捕获该架构的攻击者不能伪造一个 BYE 请求，并将其插入到 Bob 和 Alice 之间的信令流中。因为攻击者无法探知到会话参数，完整性机制传递性地保护了 Bob 和 Alice 之间的流量。

Pear-to-Pear Requests 另外，考虑到UA声称身份carol@chicago.com没有本地带外代理。当Carol希望发送INVITE给bob@biooxi.com，其UA应该直接发起与biloxi代理的TLS连接（使用[4]中描述的机制来确定怎样最好地到达给定的Request-URI）。当Carol的UA接收到biloxi代理的证书，在她通过TLS连接传送INVITE时，通常应该验证证书。然而，Carol没有办法向biloxi代理证明自己的身份，但是，她在INVITE的“message/sip”消息体上有一个CMS分离的签名。因为她没有正式与biloxi.com联系，所以，在此例中，Carol不可能有biloxi.com域的凭证。Biloxi代理也可能有一个严格的策略：在From头字段的domainname域没有biloxi.com的请求，它就不挑战请求——它认为这些用户是未认证的。

Biloxi代理对Bob有一个策略，所有未认证的请求应该重定向到在Bob@biloxi.com上注册的、合适的联系地址，即<sip:bob@192.0.2.4>。Carol通过其与biloxi代理服务器建立的TLS连接来接收重定向响应，所以她信任此联系地址的准确性。

随后，Carol应该与指定的地址建立TCP连接，发送带有Request-URI的新INVITE，包含要接收的联系地址（当请求准备好了，重新计算体中的签名）。Bob 在一个非安全的接口上接收此INVITE，但是在这种情况下，其UA会检查并识别请求的From头字段，然后将本地缓存的证书与INVITE消息体签名中的证书相比较。它以类似的方式回答，向Carol 自验证，这样，安全的会话就开始了。

有时，在管理域中的防火墙或NAT阻碍了TCP直接与UA建立连接。在这种情况下，代理服务器可能以不信任隐含的方法（如当地策略指令）将请求转发给 UA（例如，丢弃现有的TLS连接，并以明文TCP转发请求）。

Dos Protection 为了在使用了这些安全解决方案的架构中，将拒绝服务攻击威胁减小到最少，实现者应该注意下列的原则。

当运行SIP代理服务器的主机是从公网转发而来时，应该在管理域内用防御操作策略部署它（阻塞源路由信息流，优先过滤ping流量）。TLS 和IPSec都可以在参与安全联盟的管理域的边缘使用服务主机，以聚集安全隧道和套接字。这些服务主机可以承受拒绝服务攻击，确保管理域内的SIP主机不被过多的消息堵塞。

无论配置什么样的安全解决方案，直接发送到代理服务器的泛滥消息流都会锁上代理服务器上的资源，并且阻止预期的消息流到达目标地。在代理服务器内有一个与处理SIP事务相关的可计算的耗费，有状态代理服务器的耗费比无状态代理服务器高很多。因此，有状态代理比无状态代理服务器更容易受到泛滥的影响。

UA和代理服务器用401（未认证的）或407（需要代理认证）来挑战有问题的请求，丢弃一般的响应转发算法，因此，无状态地对未认证的请求起作用。

转发401（未认证的）或407（要求代理认证）状态响应放大了攻击者使用伪造的头字段值（例如Via）将信息流直接转给第三方的问题。

总之，代理服务器通过这种（如TLS）机制相互认证，极大地减少了欺诈媒介引入伪造的可以造成拒绝服务的请求或响应的可能性。这在一定程度上使得攻击者更难将无辜的SIP节点作为放大代理。

26.4 限制

虽然谨慎的使用这些安全机制能够阻止许多威胁，但是实现者和网络操作员必须理解这些机制的一些限制。

26.4.1 HTTP 摘要

在SIP中使用HTTP摘要的主要限制之一就是摘要的完整性机制不能很好地为SIP服务。特别的，它们对Request-URI和消息方法提供保护，但是对UA希望保护的一些头字段却不提供保护。

RFC2617中所描述的已有重播保护机制对SIP也有些限制。例如，下一代随机机制就不支持管道请求。随机数机制将用于重播保护。

HTTP 摘要的另一个限制就是域的范围。当用户希望以它们已有的关联向资源进行自验证时，摘要是有价值的，如服务提供商，用户是该服务提供商的一个客户（这只是一个非常普通的案例，因此该摘要提供一种非常有用的功能）。通过对比，TLS 的范围是域间或者多域，因为证书常常是全球可验证的，所以UA能够在没有已有关联的情况下认证服务器。

26.4.2 S/MIME

S/MIME 机制最显著的缺陷就是缺乏终端用户普遍的公钥基础设施。如果使用自签名证书（或者在对话中的一个参与方不能验证的证书），那么第23.2节所描述的基于SIP的密钥交换机制易受拦截式攻击，通过这种攻击，攻击者能够潜在地检查和修改S/MIME消息体。攻击者需要截取会话双方的第一次密钥交换，删除请求和响应现有的CMS分离的签名，并且插入不同的CMS分离的签名，该签名包含了攻击者提供的证书（但是该证书看似正确的记录地址的证书）。对话的双方将认为它们已经同另一方交换了密钥，实际上是每一方持有了攻击者的公共密钥。

注意，攻击者只能够利用对话双方的第一次密钥交换的弱点——随后，UA 将注意密钥的改变。攻击者过长时间（可能是几天、几周或几年）保持在对话双方所有未来对话的路径中是困难的。

SSH在第一次密钥交换中，易受到相同的拦截式攻击；然而，众所周知，SSH确实提高连接的安全，但它并不是完善的。使用密钥指纹能够向SIP提供帮助，正如它向SSH所提供的帮助一样。例如，如果双方使用SIP建立语音通信会话，那么每一方能够读取它们从对方接收的密钥指纹，该密钥指纹可以同最初的密钥指纹相比较。与中间拦截模仿参与者的信令相比，中间拦截模仿参与者的语音更难（这个实践是与Clipper基于芯片的安全电话一起使用的）。

如果UA在它们的密钥环上持有目的记录地址的证书，S/MIME机制则允许UA直接发送加密请求。然而，这就有可能使任何记录地址注册的具体设备将不能持有证书，该证书是设备的当前用户先前使用的，因此该设备不能正确地处理加密请求，这就导致一些可避免的错误信令。特别是当分发加密请求时。

当与具体用户（记录地址）关联而不是与设备（UA）关联时，与S/MIME关联的密钥是最有用的。当用户在设备间移动时，很难安全地在UA间传送私钥。本文档没有描述设备怎样获得这种密钥。

另外，S/MIME机制更大难点是它会导致大的消息，特别是使用第23.4节描述的SIP隧道机制时。鉴于该原因，当使用S/MIME隧道时，推荐使用TCP作为传输协议。

26.4.3 TLS

TLS的一个主要特点是它不能在UDP上运行；TLS需要面向连接的基础传输协议，在本文中指的是TCP。

本地带外代理服务器和/或注册服务器维护与大量UA同步的、许多长期的TLS连接也是

很困难的。这就引入了一些有效可伸缩的考虑，特别是对密集密码组的考虑。维护长期 TLS 连接的冗余是麻烦的，特别是当 UA 独自负责它们的建立时。

TLS 只允许 SIP 实体对与其相邻的服务器进行验证；TLS 确实提供了逐跳的安全。TLS 和本文本文指定的其他任何机制都不允许客户端认证它们不能形成直接 TCP 连接的代理服务器。

26.4.4 SIPS URI

实际上，在请求路径每一个分段上使用 TLS 都要求能够通过 TLS 到达终端 UAS（可能是以 SIPS URI 注册作为联系地址）。这是 SIPS 的首选用法。虽然，许多有效的架构使用 TLS 保证请求路径的安全，但是，UAS 的最后一跳的安全基于一些其他机制。因此，SIPS 不能保证真正地端对端使用 TLS。注意，许多 UA 不接受输入的 TLS 连接，甚至需要支持 TLS 的 UA 保持持久的 TLS 连接（如在 TLS 限制部分所述），作为 UAS，接收 TLS 上的请求。

不需要定位服务提供 SIPS Request-URI 的 SIPS 绑定。尽管通常是用户注册填充定位服务（如第 10.2.1 节所述），其他各种协议和接口能够可靠地为 AOR 提供联系地址，并且在适当的时候，这些工具可以自由地在 SIPS URI 与 SIP URI 之间映射。当查询绑定时，不管它是否接收到带有 SIPS Request-URI 的请求，定位服务都返回它的联系地址。如果重定向服务器正在访问位置服务，由处理重定向的 Contact 头字段的实体负责确定联系地址的属性。

确保 TLS 可以用于到目标域之间的所有请求分段有些复杂。不遵循或者折衷的密码认证代理服务器可以选择忽略与 SIPS 相关的转发规则（和普通转发规则，在第 16.6 节中描述）。比如，这种恶意的媒介能够将请求从 SIPS URI 重定向为 SIP URI，企图将安全级别降低。

另外，媒介可以合法地将请求从 SIP 重定向到 SIPS URI。请求接收者的 Request-URI 使用 SIPS URI 模式，因此不能仅基于 Request-URI 假定 SIPS 用于整个请求路径（从客户端起）。

为阐述这些问题，推荐请求接收者（其 Request-URI 包括 SIP 或者 SIPS URI）检查 To 头字段值，以确定它是否包含了 SIPS URI（注意，即使该 URI 与 To 头字段中的 URI 有相同的模式但不相等，这并不是一个安全漏洞）。尽管客户端可以选择不同的方法填充请求的 Request-URI 和 To 头字段，当使用了 SIPS 时，该差异可以解释为可能的安全破坏，请求的接收者可以因此拒绝请求。在到达本地管理域之前，接收者也可以检查 Via 头字段链，以复查整个请求路径是否使用了 TLS。源 UAC 也可以使用 S/MIME，以确保 To 头字段的初始形式可以端对端携带。

如果 UAS 有理由认为在传输过程中不正确地修改了 Request-URI 的模式，UA 将通知其用户这一潜在的安全漏洞。

作为阻止降级攻击的更好的方法，仅接受 SIPS 请求的实体也可以拒绝不安全的端口的连接。

毫无疑问，终端用户将鉴别 SIPS 与 SIP URI 之间的区别，并且终端用户能够在响应中手动将它们编辑成 Stimuli。这可能对安全有利，也可能降低安全性。例如，如果攻击者插入伪造的记录集合——有效地删除所有代理服务器的 SIPS 记录，破坏 DNS 高速缓存，那么任何穿透该代理服务器的 SIPS 请求可能失败。然而，当用户发现向 SIPS AOR 的重复呼叫失败时，它们可以在一些设备上手动将 SIPS 转换到 SIP 模式，并重试。当然，有一些保护措施（如果目的 UA 真的不能正常工作，它拒绝所有的非 SIPS 请求），但它是值得注意的限制。在好的一方面，甚至是它们仅和 SIP URI 一起出现，用户仍可以预言 SIPS 是有效的。

26.5 保密性

SIP 消息经常包含有关消息发送者的敏感信息——不仅仅是发送者说话的内容，还包括发送者通信的人、它们通信的时间和通信持续的时间以及参与会话的地点。许多应用及其用户需要向不必知道它的任何一方隐藏这些保密信息。

注意，还有一些不直接的方法能使保密信息泄漏。如果用户或者服务选择可以从人名和组织的从属关系（它描述了大部分的记录地址）猜出的、可到达自己的某一地址，那么，使用未入册“电话号码”保证保密性的传统方法是泄密的。通过泄密它们具体的呼叫方的行踪，用户定位服务可以破坏会话邀请接收者的隐私；因此，将以每个用户为基础、向某类呼叫方提供的位置类型和可用的信息来限制实现。这是希望在正在进行的 SIP 工作的进一步研究中解决的一类问题。

在某些情况下，用户可能希望在传送身份的头字段中隐藏个人信息。它不仅适用于 From 和代表请求发起者的相关头，还适用于 To——它不适合向最终目的发送快速拨号的别名或一组目标的未扩展标识符。当请求路由时，这两者都从 Request-URI 中删除；但是，如果两者最初是相同的，那么，在 To 头字段中不应该改变这两者。因此，鉴于保密性原因，期望创建与 Request-URI 不同的 To 头字段。

27 IANA 考虑

SIP 应用中所使用的所有方法名、头字段名、状态代码和选项标签都使用 IANA 注册，主要运用的是 RFC 中 LANA 考虑章节的指令。

本规范指导 IANA 创建四个子注册项，这些子注册项在 http://www.iana.org/assignments/sipparameters 上给出：选项标签、警告代码、方法和响应代码，这些子注册项被添加到现有的头字段的子注册项中。

27.1 选项标签

本规范建立了选项标签子注册项，该子注册项在 http://www.iana.org/assignments/sip-parameters 有描述。

选项标签在如 Require、Supported、Proxy-Require 和 Unsupport 之类的头字段中使用，这些头字段用于支持 SIP 扩展兼容机制（参看第 19.2 节）。选项标签本身是一个与特殊 SIP 选项相关联的字符串(也就是一个扩展)。它可以确认到 SIP 端点的选项。

当选项标签在标准跟踪 RFC 中公布时，由 IANA 对其注册。RFC 的 IANA 考虑章节包含以下信息，这些信息连同 RFC 的出版号一起出现在 IANA 注册处。

- 选项标签的名称。这个名称长度可任意，但最长不能超过 20 个字符。名称只能由字母或数字（第 25 章）组成。
- 描述扩展的描述文本。

27.2 警告代码

本规范建立了警告代码子注册项，该子注册项在 http://www.iana.org/assignments/sip-parameters 有描述，所有警告代码在第 20.43 节列出。附加的警告代码由 RFC 出版物注册。

警告代码表的描述文本如下：

当事务失败由会话描述协议(SDP)(RFC2327 [1])问题所产生，警告代码向 SIP 响应消息中的状态代码提供附加信息。

警告代码由三个数字组成。第一位数是“3”表明是 SIP 的警告。除非在以后的版本中描述使用除 3XX 之外的警告代码，只能注册 3XX 警告代码。

警告代码 300 至 329 保留，用于指明会话描述中关键字的问题；警告代码 330 至 339 对与会话描述中所请求的基础网络服务相关联的问题作出警告;370 至 379 对与会话描述中所请求定量 QOS 参数相关联的问题作出警告；390 至 399 是其他警告，不属于以上任何一种范畴。

27.3 头字段名

这里没有采用 IANA 在 http://www.iana.org/assignments/sipparameters 下所描述的有关头子注册项的说明。

为了注册一个新的头字段，RFC 出版物中需要提供以下信息：

- RFC 编号，在该编号中头已注册；
- 已注册的头字段名称；
- 如果定义了头字段，则需要该头字段的压缩形式版本。

一些通常广泛使用的头字段被分配一个字母的压缩形式(参看第 7.3.3 节)。在 SIP 工作组审阅后，并且在 RFC 发布后，才可以分配压缩形式。

27.4 方法和响应代码

本规范建立了方法和响应代码子注册项，这些子注册项在 http://www.iana.org/assignments/sip-parameters 下有描述，并且按照下述形式对其初始化。初始化方法表如下：

INVITE [RFC3261]

ACK [RFC3261]

BYE [RFC3261]

CANCEL [RFC3261]

REGISTER [RFC3261]

OPTIONS [RFC3261]

INFO [RFC2976]

响应代码表从第 21 章开始介绍，这一部分对 Informational、成功、重定向、客户端错误、服务器错误和全局错误分类标注。表格式如下：

类型 (例如, Informational)

Number Default Reason Phrase [RFC3261]

为注册一个新的响应代码或方法，需要在 RFC 出版物中提供以下信息：

- RFC 编号，在 RFC 编号已注册了方法和响应代码

- 已注册的响应代码编码或者方法名
- 如果可用，那么也需提供响应代码的默认原因分析

27.5 “message/sip”MIME 类型

为了允许 SIP 消息作为 SIP 的体而通过，本文档注册了“message/sip”MIME 媒体类型，旨在端到端的安全性考虑。以下信息定义了该媒体类型：

Media type name: message

Media subtype name: sip

Required parameters: none

可选参数：版本

版本：消息的 SIP 版本编号(如“2.0”)。如果不存在，默认版本号为“2.0”。

编码模式：SIP 消息由一个 8 位报头组成，其后跟一个二进制 MIME 数据对象（可选的）。因此，必须将 SIP 消息作为二进制看待。在正常环境下，SIP 消息可以采用二进制传输，而不需要特殊编码。

安全考虑：如下

作为一个安全机制，该用法的动机和实例与第 23.4 节中所给出的 S/MIME 一致。

27.6 新 Content-Disposition 参数注册

本文档注册了 4 个新 Content-Dispositio 报头 disposition-type：警报、图标、会话和提交。本文档的作者要求在 Content-Disposition 的 IANA 注册中记录这些值。

Content-Disposition 的描述，包括动机和实例见第 20.11 节。IAMA 注册可简短描述为：

alert	消息体是提醒用户的自定义铃声
icon	消息体是显示给用户的图标
render	消息体应显示给用户
session	消息体描述一个通信会话，例如 RFC2327 SDP 消息体

28 对 RFC 2543 的改变

RFC 3261 是 RFC 2543 的修正版。它大部分向后兼容 RFC 2543。这里描述的改变是指修改了在 RFC 2543 中发现的许多错误，并提供了 RFC 2543 中没有详细描述的案例的信息。这样协议就是一种更清晰的分层模型。

RFC 3261 的功能行为不同于 RFC 2543，这是对 RFC 2543 的一个实质性改变，这样就对某些情形下的互操作性或正确操作有影响。虽然功能行为不同于 RFC 2543 但这不是互用性问题的一个潜在根源。还有许多地方需要说明，这里就不再一一说明了。

28.1 主要功能改变

- 如果 UAC 希望在呼叫接收到应答前终止呼叫，它就发送 CANCEL 请求。如果最初的 INVITE 仍然返回 2xx，UAC 随后就会发送 BYE 请求。仅仅能在运行的呼叫腿上

发送 BYE（在 RFC3261 中称为会话），尽管在 RFC 2543 中可以在任何时间发送它。

- 转换 SIP BNF 以符合 RFC 2234。
- SIP URL BNF 更加通用，允许用户端使用更多的字符集。而且，比较规则简化了，基本不再区分大小写，并且对出现在参数中的详细的比较处理进行了描述。最实质的改变是带默认参数值的 URI 不能和未带参数的 URI 匹配。
- 删除 Via 隐藏。由于它依靠下一跳来执行模糊处理，所以它有很强的信任度。取而代之的，Via 隐藏可在有状态代理中作为一个本地实现选择，因而这里不再说明。
- 在 RFC 2543 中，CANCEL 和 INVITE 事务是混合使用的。在 RFC 3261 中它们分开使用。当用户先发送 INVITE 请求然后再发送 CANCEL 请求，INVITE 方法仍然正常终止。UAS 需要用 487 响应来应答先前的 INVITE 请求。
- 同样的，在 RFC 2543 中，CANCEL 和 BYE 事务也是混合使用的；RFC 2543 允许 UAS 在收到 BYE 请求后可以不去响应 INVITE 请求。RFC 3261 中不允许这样。原来发送的 INVITE 请求需要得到响应。
- RFC 2543 中，UA 仅仅只需支持 UDP。在 RFC 3261 中，UA 需支持 UDP 和 TCP。
- RFC 2543 中，在多个挑战情况下，分发代理仅仅从下游的元素上传一个挑战。在 RFC 3261 中，代理要接收所有的挑战并把它们放到转发响应中。
- 在摘要凭证中，需要引用 URI；这一点在 RFC 2617 和 RFC 2069 里是不清楚的，并且两者在这一点上是不一致的。
- SDP 处理在一个单独的规范[12]中进行了说明；它被更详细的指定成一个正式的呼叫/应答交换处理，该处理可通过 sip 有效地开辟隧道。对基线 SIP 实现，在 INVITE/200 和 200/ACK 中允许 SDP；RFC 2543 提到了在单个事务的 INVITE、200 和 ACK 中使用 SDP 的能力，但这一点没有很好的详细说明。在扩展中允许更复杂的 SDP 用法。
- 在 URI 和 Via 头字段里为 IPv6 增加了全面的支持。在 Via 中支持 Ipv6 要求 Via 的头字段参数允许带方括号和冒号。这些符号之前是不允许使用的。理论上，这样就会引起与先前实现的互操作性问题。然而，我们已发现大多数实现在这些参数里接受任何非控制 ASCII 码。
- DNS SRV 程序在一个单独的规范[4]中说明。这个程序使用了 SRV 和 NAPTR 的资源记录，并且如 RFC 2543 里描述的那样不再结合 SRV 记录的数据。
- 循环检测是可选的，它被 Max-Forwards 的一个强制用法代替。RFC 2543 里的循环检测程序有一个严重的 bug，这个 bug 会把“螺旋”报告成错误情况，实际上“螺旋”不是错误情况。可选循环检测程序在 RFC 3261 里得到更充分和正确的详细说明。
- RFC 3261 中标签的使用是强制性的(在 RFC 2543 里标签的使用是随意的)，因为在 RFC 3261 里标签是会话鉴别的基本构造模块。
- 增加的 Supported 头字段允许客户端指出服务器支持什么扩展，服务器能将这些扩展应用到响应中，并且在响应中用 Require 说明它们的用法。
- 几个头字段的 BNF 扩展参数已丢失，在这里添加了这些参数。
- Route 和 Record-Route 的结构处理在 RFC 2543 里没有很详细的规定，也没有正

确的方法。这种处理在 RFC 3261 中作了充分的改写(并且非常的简化)，这是有说服力的最大的改变。为不使用“预装载路由”的部署提供了向后兼容，这里最初的请求有以某种方式从外部 Record-Route 获得的 Route 头字段值集。在这些情况下，这个新的机制是不能互操作的。

- RFC 2543 中的消息行可用 CR、LF 或 CRLF 终止。本规范仅仅允许使用 CRLF 终止消息行。
- CANCEL 和 ACK 的 Route 用法在 RFC 2543 中没有明确定义。在 RFC 3261 里它有详细的说明；如果请求带有 Route 头字段，那么请求的非 2xx 响应 的 CANCEL 和 ACK 应携带相同的 Route 头字段值。2xx 响应的 ACK 使用的 Route 字段值是从 2xx 响应的 Record-Route 字段值中传递过来的。
- RFC 2543 允许在单个的 UDP 包中有多个请求。此用法在 RFC 3261 里被取消。
- 在 Expires 的头字段和参数中使用绝对时间已取消。因为这样将引起时间不同步的元素的互操作性问题。在 RFC 3261 中使用相对时间。
- 强制所有元素使用 Via 头字段值的分支参数。在 RFC 3261 里该分支参数是唯一的事务标识符。这样就避免了 RFC 2543 的复杂的故障甚多的事务鉴别规则。在参数值中使用 Magic cookies 是为了确定前一个跳跃点是否生成了全球唯一的参数，并且当它未给出时可以返回和原先的规则进行比较。 这样，就保证了互操作性。
- RFC 2543 中，TCP 连接的关闭和 CANCEL 请求产生相同的结果。代理间的 TCP 连接几乎是不可能实现（且是错误的）这一点的。这个功能在 RFC 3261 里被取消，所以 TCP 连接状态和 SIP 处理之间就不存在耦合。
- 在 RFC2543 中没有写明当一个 UA 的对等实体正在处理时，UA 是否可以向其对等实体初始化新事务。在这里特别指明了这一点，RFC3261 允许非 INVITE 请求，不允许 INVITE 请求。
- RFC3261 去掉了 PGP。由于它没有描述得很清晰，而且它与更完整的 PGP MIME 不兼容。在 RFC3261 中 PGP 被 S/MIME 所取代。
- RFC3261 为端到端 TLS 添加了一种“sips”URI 模式。这种模式不能向后与 RFC2543 兼容。当现有元素接收到在 Request-URI 带 SIPS URI 模式的请求时，它很可能拒绝该请求。这实际是一个特征：它确保了只有路径上的所有跳跃点都是安全的，才传送对 SIPS URI 的呼叫。
- RFC3261 通过 TLS 添加了安全特性，而且专门设置了一章，用较大的篇幅对完整的安全性考虑进行描述。
- 在 RFC2543 中，不要求代理转发 101～199 范围内的临时响应。在 RFC3261 中要求代理必须转发。这点非常重要，因为许多随后的特性都依赖于 101 到 199 范围内的所有临时响应的传送。
- 在 RFC2543 中，对 503 响应代码的描述很少。我们可以发现，在实际使用中它常用于指示代理中的失败或者负载条件。这就需要我们对它给予特殊的对待了。特别是，503 的接收会触发对 DNS SRV 查询结果的下一个元素联系的尝试。也就是说，只有在特定条件下，代理才会向上游转发 503 响应。
- 在 RFC2543 中定义了 UA 对服务器的认证机制，但并没有描述清楚。在 RFC3261 去掉了这一部分。取而代之的，允许 RFC2617 中相互认证过程。

- 在 UA 接收到初始 INVITE 的 ACK 后，它才能发送呼叫的 BYE。这在 RFC2543 中是允许的，但是这可能会导致竞争状态出现。
- 在 UA 或者代理获取请求的临时响应后，它才能向事务发送 CANCEL。这在 RFC2543 中是允许的，但是这可能会导致竞争状态出现。
- RFC3261 中不允许在注册中有行为参数。因为对于某些有用的服务而言，这是不足够的，而且在代理中使用应用处理时，还会造成冲突。
- RFC2543 具有许多多播的特殊案例。例如，特定响应的删除，计时器的调整等等。而在 RFC3261 中，多播则受到更多限制，使用多播时，不会对协议操作带来影响。对多播的这种限制已经在文档中写明。
- 在 RFC3261 中完全删掉了基本认证，并且禁止其使用。
- 在 RFC3261 中，代理在接收 6xx 时不再立即将其转发，而是立即取消待处理分支。这就避免了可能的竞态状态出现，这种竞态状态会导致 UAC 在接收到 2xx 之后接收到 6xx。而在竞态状态以外的所有案例中，结果都是相同的——6xx 上游转发。
- RFC2543 中没有对请求合并问题作过多描述。在代理处分发一个请求，随后这个请求会在一个元素处再合并，这就是请求合并。只有 UA 才能处理合并，而且，定义了拒绝除第一个请求外的所有请求的程序。

28.2 次要功能的变化

- 为用户的可选内容表达添加了 Alert-Inf、Error-Info 和 Call-Info 的头字段。
- 添加 Content-Language、Content-Disposition 和 MIME-Version 头字段。
- 添加了“双占用处理”机制，用来处理双方同时给对方发送 re-INVITE。它使用新的 491（未决的请求）错误代码。
- 为了支持丢失的呼叫或者消息在稍后返回，添加了 IN-Reply-To 和 Reply-To 的头字段。
- 添加 TLS 和 SCTP 作为有效的 SIP 传输机制。
- 在 RFC2543 中，在呼叫期间处理失败状态有多种机制，在 RFC3261 中，这些机制被统一了，即发送 BYE 终止呼叫。
- 在 RFC2543 中，要求在 TCP 上对 INVITE 响应重新传送，但是应注意的是，实际上只在 2xx 响应时需要转发 INVITE。这是因为协议层不完备。在 RFC3261 中，定义了多个相关的传输层，所以不再需要 RFC2543 中的规定。只需要在 TCP 上转发 INVITE 的 2xx 响应。
- 现在客户端和服务器事务机由超时驱动，而不是由转发数所驱动。这样就允许为 TCP 和 UDP 指定合适的状态机。
- 在 REGISTER 响应里使用 Date 头字段，以提供用户代理中日期自动配置的简单方法。
- 允许注册员拒绝因持续时间太短而期满的注册。为此，定义了 423 响应代码和 Min-Expires。