

# 会话初始协议-SIP 协议中文版本

译者 : james.zhu

SIP 协议-james.zhu 中文版本-v1

发布时间: 2023 年

联系方式 :

个人邮箱 : 522137361@qq.com

网站:www.asterisk.org.cn www.sip.org.cn

微信公众号 : asterisk-cn

## RFC3261 规范-SIP 协议-james.zhu 中文版本发布说明

### 版权声明 :

此译作中文版版权归 james.zhu 个人所有, 任何组织和个人未经本人同意, 不得转载。

### 其他说明 :

1. 因个人能力有限, 所有翻译内容中文版本可能存在理解错误或者偏差, 在后期版本更新中会逐步优化, 使得中文含义更加通畅, 准确。

2. 本文档中可能存在一些拼写错误，将在后期版本中修正。
3. SIP 相关协议和使用细节场景等非规范内容将在后期版本中逐步更新添加。
4. 笔者添加了部分的非规范内容，希望进一步确认其内容的逻辑性，在此内容前做了说明，请勿理解为规范规定内容。
5. 此版本为非主流版本，是个人学习和以及对 SIP 相关协议规范理解输出的中文版本。
6. 关于内容的优化和其他问题，请通过以上联系方式联系作者本人。

#### 友情链接：

<https://www.rfc-editor.org/rfc/rfc3261>

Asterisk 开源社区：[www.asterisk.org.cn](http://www.asterisk.org.cn)

开源 SIP 协议及技术资料分享平台：[www.sip.org.cn](http://www.sip.org.cn)

深圳鼎信通达股份有限公司：[www.dinstar.cn](http://www.dinstar.cn)

---

## SIP: Session Initiation Protocol

#### Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current

edition of the "Internet Official Protocol Standards" (STD1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice Copyright (C) The Internet Society (2002). All Rights Reserved.

## SIP 协议摘要

本规范描述了会话初始协议 (SIP) , 一个应用层控制 (信令) 协议, 一方或者多方参与用户通过此协议用来创建、修改和终止会话。这些会话包括 Internet 电话呼叫、多媒体分发处理和多媒体会议。

SIP 邀请用来创建会话传输会话描述, 允许参与用户通过传输的会话描述进行协商来达成一致的媒体类型协商。SIP 利用被称之为代理服务器的网元帮助将 SIP 请求路由到用户的当前位置, 对用户进行认证并且授权用户访问所提供的服务, 实现服务提供商的呼叫路由策略, 并为用户提供业务功能服务。SIP 也提供注册功能, 注册功能允许用户上传它们的当前的位置信息, 以便让代理服务器使用。SIP 是可运行在多个不同的传输协议之上的协议。

## 1 Introduction

很多基于网络的应用软件都要求可以实现会话创建和管理, 这里的会话可以视为是关联多个参与方交换数据的方式。实际部署这些应用软件是非常复杂的过程 : 用户可能在几个终端之间移动切换, 用户也可能使用多个名字, 用户也可能使用不同的媒体, 有时还同时使用不同的媒体介质。目前, 有很多不同的协议被准许在网络上运行, 这些协议来传输各种形式的实时媒体会话数据, 例如语音视频, 文本信息。Session Initiation Protocol (SIP) 支持以上所说的这些功能描述和相关的协议, 它可以支持开启网络的用户代理来发现其他的终端, 准许其他终端的某些会话属性, 终端之间可以共享这些会话属性。

为了查询到期望的会话参与对象，和其他的功能，SIP 支持了网络主机设施创建（称为代理服务器），用户代理可以对会话发送注册，邀请和其他的请求。SIP 是一个敏捷，通用的工具，它用来创建，修改和结束会话，它可以不依赖于正在工作的传输协议，并且无需依赖于各种已建立的会话类型。

## 2 Overview of SIP Functionality

SIP 一种应用层的控制协议，它可以创建，修改和结束多媒体会话（会议），例如网络电话呼叫。SIP 也可以邀请参与对象加入到已存在的会话，例如多方广播会议。它可以从当前存在的会话中再加入媒体也可以移除媒体。SIP 可以透明支持名称映射，服务重新转发服务，这些服务功能支持个人移动能力[参考链接 27]-无论网络位置如何，用户可以在网络中保持一个对外单点可视的身份。

SIP 支持创建和结束媒体通信的五个方面的功能：

- 用户定位：端系统的决定来支持通信；
- 用户有效性：决定被呼叫方是否有意愿决定加入通信；
- 用户能力：决定用户可使用的媒体和其媒体参数；
- 会话创建：“ringing”，在呼叫方和被呼叫方之间创建会话参数；
- 会话管理：包括转发，结束会话，修改会话参数和调用服务。

SIP 不是一个单一，垂直集成度通信系统。SIP 而是一个模块，它可以用来和其他的 IETF 协议集成来构建一个完整的媒体架构。典型的架构如，和实时传输协议 (RTP(RFC1889[28])) 配合使用实现实时数据传输，提供 QoS 反馈，使用实时媒体协议 (RTSP(RFC2326[29])) 来控制媒体流和媒体的发送控制，媒体网关控制协议 (MEGACO) (RFC3015[30]) 来控制网关对 PSTN 网络的支持，和会话描述协议 (SDP) (RFC2327 [1]) 来描述媒体会话。因此，SIP 应该结合其他的协议一起使用对用户提供完整的服务。但是，基本的 SIP 功能和操作不会依赖于其他任何协议。

SIP 本身不提供服务。但是，SIP 提供基本的操作，这些操作可以支持部署不同的服务。例如，SIP 可以定位一个用户，并且对当前定位发送一个不透明的对象。如果此基本操作用来支持发送一个写入 SDP 的会话描述，终端可以同意会话中的参数。如果同样

的操作用来传递一张呼叫方的图片和此会话描述，那么就可以在早期部署一个“caller ID”服务。就像这个例子所展示的，一个单个基本操作往往被用来提供不同的服务。

SIP 不提供会议控制服务例如发言权控制和发言，它不能对会议发出命令控制如何管理会议。SIP 可以用来发起一个会话，这个会话可以用来支持一些会议控制协议。因为，SIP 创建的消息和会话可以传递到完全不同的网络中，SIP 不能也不会提供任何网络资源预设的支持能力。

SIP 所提供的服务的本质使得安全性特别重要。对于对端来说，SIP 提供了一个安全服务单元，这些服务单元包括拒绝攻击防止服务，认证（包括用户对用户，代理对用户），集成保护，加密和私有服务。

SIP 可以支持 IPv4 和 IPv6 两种网络环境。

### 3 Terminology

在本文档中，关键词 "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", 和 "OPTIONAL" 通过 BCP14, RFC2119 [2] 加以解释，并且说明了遵从 SIP 部署要求级别和严格程度。读者需要根据关键词的字面意思来区分这些规则的基本和宽泛程度，尽可能最大程度对应 SIP 协议的要求。很多时候，由于开发人员，特别是对英文协议了解不够，或者对协议的理解不同，所以导致一些兼容性问题或者功能不一致等问题。

### 4 Overview of Operation

此部分使用一个简单示例介绍了 SIP 的基本操作。它实际上是一个学习指导，没有包含任何正式的说明。

第一个示例显示了 SIP 的基本功能：终端定位，希望通信的意愿，创建会话参数的协商和创建会话后会话拆线。

图表 1 显示了一个典型的介于两个用户之间的 SIP 消息交互，两个用户分别是 Alice 和 Bob。（每个消息都通过一个带字母 F 的标签来标注，文本号码说明一个标注号码）。在

这个例子中，Alice 使用了一个在 PC 上运行的 SIP 应用程序（作为一个软电话）来呼叫 Bob，Bob 的电话是一个基于互联网的 SIP 电话。这个图例也同时显示了，这里有两个 SIP 代理服务器介于 Alice 和 Bob 之间来支持会话管理工作。在图例 1 中，这种典型的设置方式我们通常称之为“SIP 拓扑图”“SIP 框架”。

Alice 使用自己的 SIP 身份“呼叫”Bob，这种 SIP 身份是一种 URL 类型，我们这里称之为 SIP URL。SIP URLs 在第 19.1 章节中做了定义。它的格式和邮件的格式非常相似，一般都包括一个用户名和主机名称。在这个例子中，它就是 `sip:bob@biloxi.com`，这里 biloxi.com 是一个 Bob 的 SIP 服务提供商。Alice 可能也具有和 Bob 的 URL 同样的类型，或点击一个超链接后进入一个地址薄。SIP 同样也提供一个安全的 URL，被称之为 SIPS URL。安全 URL 的示例为 `sips:bob@biloxi.com`。通过 SIPS URL 发起的呼叫可以保证安全，加密的传输，它用来传输所有从呼叫方到被呼叫方域的所有 SIP 消息。从这里，开始，SIP 的请求消息安全地发送到被呼叫方，但是安全机制依赖于被呼叫方域的安全策略设置。

SIP 是基于一种类似于 HTTP-形式的请求/响应事务处理模式。每个事务处理包括一个启动了特别 method 方法的请求，或者一个功能，和至少一个来自于服务器端的响应构成。在这个例子中，事务处理是以 Alice 的软电话开始，软电话发送一个 INVITE 请求，携带了 Bob SIP URL 地址。这里，INVITE 就是一个 SIP method 方法，它指定了一个执行命令，请求方（Alice）想让服务器方（Bob）接收这个请求。这个 INVITE 请求中包含了几个 header fields-头字段。Header fields 被命名为属性值，这些属性值提供了关于消息的其他额外信息。在 INVITE 中的某些属性表示了呼叫的唯一身份，目的地地址，Alice 的地址，和 Alice 和 Bob 之间创建会话所期望的会话类型的信息。INVITE(F1 消息中) 可能类似于这样的流程：

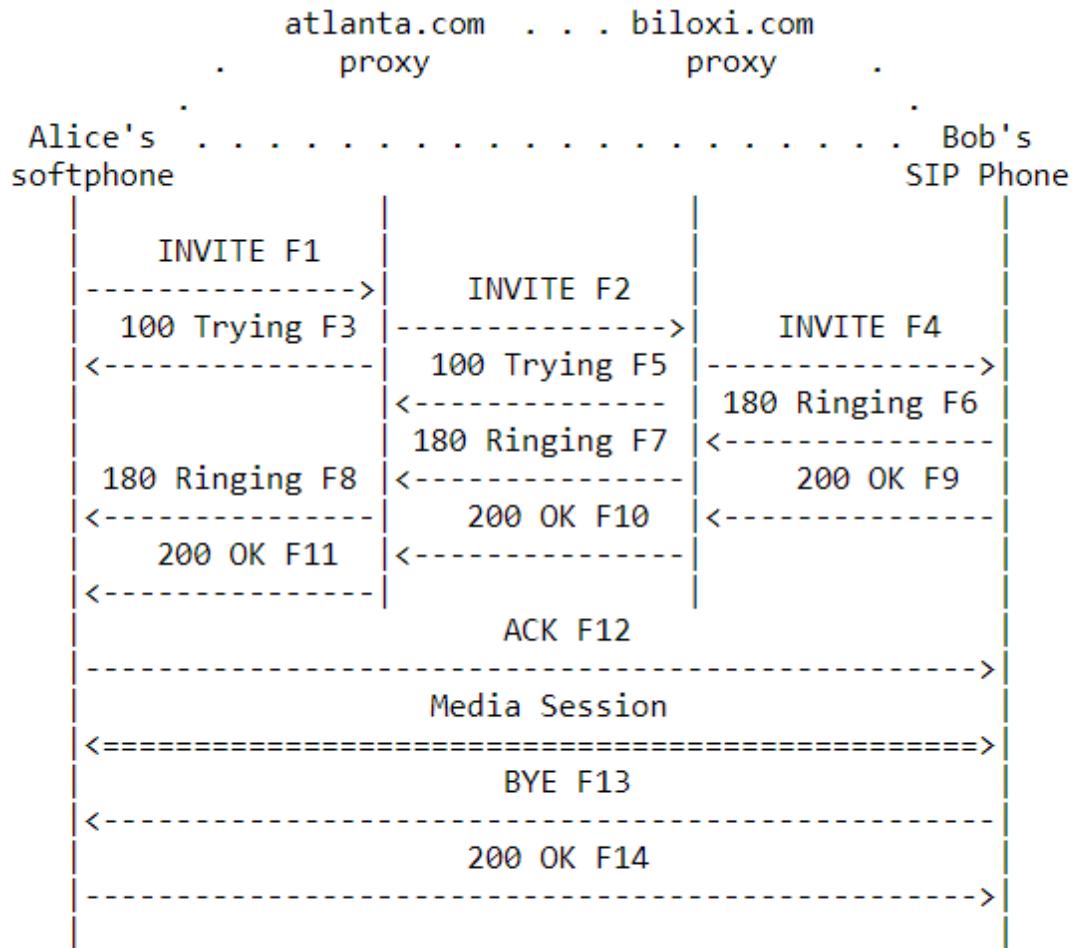


Figure 1: SIP session setup example with SIP trapezoid

```

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhs
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 的 SDP 消息不显示)

文本消息的第一行包含了一个方法名称 method (INVITE)。紧接着的是几行包含一个 header 值域的列表。在这个示例中，它们包含了最少的要求设置。这些头 header 简单描述如下：

Via 包含了一个地址(pc33.atlanta.com)，Alice 希望对这个请求获得响应的地址。它也包含了一个 branch 参数来确认这个事务处理。

To 包含了一个显示名称(Bob)和一个 SIP 或者 SIPS URL (sip:bob@biloxi.com)。Display names 在 RFC2822 [3]有介绍。

From 也包含一个显示名称 (Alice) 和一个 SIP 或 SIPS URI (sip:alice@atlanta.com) ，它表示这个请求的发起方。这个 header 同时还有一个一个标签 tag 参数，此标签包含了一个任意字符串 (1928301774) ，这个字符串被添加到了软电话的 URL。此标签也是为了确认身份的目的。

Call-ID 包含一个对这个呼叫的全局唯一确认信息，它是由一个任意字符串和软电话主机名称或 IP 地址组合而成。To tag 的组合，From tag 的组合，和 Call-ID 完整定义了一个 Alice 和 BoB 两者之间的点对点的 SIP 关系，这种关系可以看作一个 dialog 对话。

CSeq 或 Command Sequence 包含一个整数和一个 method 名称。这个 CSeq 是一个增长的数值，它是支持每一个在 dialog 里新的请求，并且是一个普通的序列号码。

Contact 包含一个 SIP 或者 SIPS URI，它用来表示一个直接的路由去联系 Alice，通常情况下，它由一个用户名以及它所在的全限定域名构成 (FQDN)。如果使用了全限定域名 (FQDN) 的话，许多终端系统没有已注册的域名，因此，这里 IP 地址是允许的。Via 头告诉其他参数往哪里发送响应消息，Contact 告诉其他参数往哪里发送将来的请求消息。

Max-Forwards 最大前转来限定一个请求到达目的地的最大跳跃 (hop) 数量。它是由一个整数数值构成，在经过一个跳转时会降低一个数值。

Content-Type 包含一个信息体的描述，这里忽略。

Content-Length 包含计算消息体长度的八位位组（字节）。

完整的 SIP 头字段集在 20 章节中有详细的定义。

会话的细节，例如媒体类型，编码，采样率等没有在 SIP 中进行描述。这些细节而是包含在了 SIP 消息体中，它们通过编码以后以各种协议的格式出现。其中一种协议格式就是 会话描述协议 -Session Description Protocol (SDP) (RFC2327 [1])。这个 SDP 消息（没有在这里显示）示例是通过 SIP 消息来传输，传输的方式类似于电子邮件中的附件方式来传输，或类似于通过 HTTP 消息传输网页页面内容的方式。

因为软电话不知道 Bob 的地址或 biloxi.com 域名中的 SIP 服务器地址，软电话发送一个 INVITE 到 SIP 服务器端，这个 SIP 服务器支持 Alice 的域，atlanta.com。atlanta.com SIP 服务器已经配置了 Alice 的软电话，或者通过 DHCP 发现了软电话地址信息。

这个 atlanta.com SIP 服务器是一个代理服务器。代理服务器接收请求，然后作为一个请求者转发这些请求。在这个实例中，代理服务器接收到了 INVITE 请求，然后发送了一个 100 (Trying) 响应给 Alice 的软电话。这个 100 (Trying) 响应表示这个 INVITE 已经被收到，代理正在通过路由设置路由这个 INVITE 到其目的地。在 SIP 中，响应消息使用一个三位数的响应码和描述短语作为回复消息。这个响应消息在 Via 中包括同样的 To, From, Call-ID, CSeq 和 branch parameter，这些参数和 INVITE 中的一样，这些参数允许 Alice 的软电话关联响应消息来发送 INVITE 消息。这个 atlanta.com 代理服务器定位到这个代理服务器在 biloxi.com，它可能执行一个特别的 DNS 查询来找到服务 biloxi.com 域的 SIP 服务器。这个部分的描述在[参考链接 4]中。因此，它获得 biloxi.com 代理服务器的 IP 地址，然后转发或者在这里代理其 INVITE 请求。在转发这个请求之前，这个 atlanta.com 代理服务器添加另外一个 Via 头字段，这个头字段包含自己的地址（这个 INVITE 已经在第一个 Via 包含了 Alice 的地址）。biloxi.com 代理服务器收到这个 INVITE 消息后，然后回复一个带 100 (Trying) 响应消息到 atlanta.com 代理服务器，表示它已经收到了这个 INVITE 消息，正在处理这个请求。代理服务器会查询一个定位服务器，我们称之为定位服务，定位服务包含当前 Bob 的 IP 地址。（我们将会在下一个部分看到如何实现数据库查询。）biloxi.com 代理服务器会添加另外一个 Via header，并且携带自己的 IP 地址，这个地址是针对这个 INVITE 请求的，代理转发这个请求到 Bob 的 SIP 软电话。

Bob 的软电话收到这个 INVITE 消息后，对 Bob 发出提示，告诉他有从 Alice 来的电话呼叫，Bob 决定是否应答这个呼叫，这里 Bob 的软电话会产生振铃提示。Bob 的软电话提示 180 振铃，这个响应消息会路由根据相反的方向回到两个代理服务器。每个代理使用 Via header 域值来决定发送响应的地址方向，并且从顶部路由记录中删除自己的地址。因此，尽管要求 DNS 和定位服务查询 路由这个初始的 INVITE 请求，180 (Ringing) 响应返回到呼叫方时可以没有查询消息或没有代理服务器中所保持的状态。

这样也获得了一个合理的响应属性，每个看到 INVITE 消息的代理服务器也可以看到所有对 INVITE 的响应消息。

当 Alice 的软电话收到这个 180 (Ringing) 响应后，它会传递这个信息给 Alice，传递过来的信息方式可以使用一个回铃音 (ringback tone) 或者在 Alice 终端屏幕显示一个消息。

在这个示例中，Bob 决定应答这个呼叫。当他拿起电话听筒时，他的 SIP 电话会发送一个 200 (OK) 响应消息来表示这个呼叫已经应答。这个 200 (OK) 包含了一个消息体，这个消息体带了这个呼叫会话的媒体描述类型，这个媒体描述中说明了 Bob 希望和 Alice 创建会话。因此，这里有一个两阶段的 SDP 消息交互过程：Alice 发送一个交互消息给 Bob，Bob 然后发送了一个交互消息给 Alice。这个两阶段的交互提供基本的协商能力，它是基于一个简单的 offer/answer SDP 交互模式来进行的。如果 Bob 不希望应答这个呼叫或者 Bob 电话可能被占线，他此时和其他人进行通话，那么 Bob 终端则会发送一个错误响应消息而不是 200 (ok)，这样就会导致没有媒体创建的情况。完整的 SIP 响应码在图例 1 中列出。Bob 发送的 200 (OK) (图例 1 中的消息 F9) 可能类似于这样：

```

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=1
92.0.2.2
Via: SIP/2.0/UDP
pc33.atlanta.com ;branch=z9hG4bK776asdhd ;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)

```

响应消息的第一行包含了响应码(200)和响应原因短语习语(OK)。其余其他行消息包含 header 值域消息。Via, To, From, Call-ID, 和 CSeq header 值域都是从 INVITE 请求中拷贝过来的。(注意, 这里有三个 Via 头值 -一个是 Alice 软电话添加的, 另外一个是 atlanta.com 代理服务器添加到, 还有一个是 biloxi.com 代理添加的)。Bob 的软电话已经在 header 中添加了一个 tag 标签参数 To。这个标签将会在两个终端的 dialog 中使用, 也将会在此呼叫后续的请求和响应使用。

Contact 头包含了一个 URL 地址, 这个 URL 就是 Bob 可以直接访问的软电话地址。Content-Type 和 Content-Length 参考消息体 (这里没有列出), 这个消息包含了 Bob SDP 媒体的信息。

另外, 在这个示例中, DNS 和定位查询显示代理服务器可以做一个灵活的路由决定, 它可以决定往哪里发送请求。例如, 如果 Bob 软电话返回一个 486 (Busy, 忙状态)响应的话, biloxi.com 代理服务器可以代理这个 INVITE 到 Bob 的语音邮箱服务器。代理

服务器也可以同时发送一个 INVITE 到多个地址。这种类型的并行查询方式称之为 forking (分叉复制) 处理方式。

在这个示例中，200 (OK) 消息通过两个代理服务器返回到 Alice，Alice 软电话收到响应消息，软电话停止回铃音，表示呼叫已应答。最后，Alice 软电话发送一个确认消息 ACK，这个消息发送到 Bob 软电话来确认最终响应 (200 (OK)) 已收到。在这个示例中，这个 ACK 是通过 Alice 软电话直接被发送到了 Bob 软电话，发送过程绕开了两个代理服务器。这样处理的原因就是因为两个终端已经通过互相学习知道对方的地址，双方地址是通过 INVITE/200 (OK) 交互时的 Contact 头获得，当然这个地址在初始时的 INVITE 是双方都不知道的。两个代理服务器的查询服务也不需要，因此，代理服务器则会退出这个呼叫流程。这个处理过程成功实现了 INVITE/200/ACK 三方握手，并且创建了 SIP 会话。完整的关于 SIP 会话创建的细节，请参考第 13 章节。

Alice 和 Bob 之间的媒体会话启动，他们之间的软电话开始发送媒体数据包，媒体数据包的格式是他们之间的 SDP 交互相同意支持的格式。一般来说，端对端的媒体数据通过不同的路径发送，而不是使用 SIP 信令消息的路径。

在这个会话过程中，Alice 或 Bob 任何一方都可以有权决定修改媒体会话的属性。修改会话属性是通过发送一个 re-INVITE 消息，在此消息中包含一个新的媒体描述来实现。这个 re-INVITE 涉及到了已存在的 dialog，因此其他的参与方知道这个消息是修改了现在的会话，而不是重新建立的新会话。其他方发送一个 200 (ok) 接受这个修改。请求方对 200 (ok) 发送一个 ACK。如果其他方不能接受这个修改的话，它会发生一个错误响应，例如 488 (Not Acceptable Here)，同样也接收一个 ACK 确认消息。但是，这个 re-INVITE 失败不会导致目前的呼叫失败-这个会话仍然会继续使用以前协商的属性。完整的话会修改细节，请阅读第 14 章节内容。

在呼叫结束后，Bob 首先挂机(hangs up)，并且生成一个 BYE 消息。这个 BYE 会直接路由返回到 Alice 的软电话，这里仍然绕过了代理。Alice 确认了 BYE 接收，发送一个 200 (ok)，结束这个会话和 BYE 消息事务。这里没有 ACK 发送-ACK 发送仅发生在对 INVITE 请求的响应中。对于对 INVITE 特别处理的原因会在后续的章节中讨论，这里涉及了 SIP 中的可靠性机制问题，振铃应答的时间程度和分叉处理等因素。因为这个原因，SIP 中的请求处理经常被划分为 INVITE 或者 non-INVITE 两个层面，除了

INVITE method 以外，还涉及其他的所有 methods。完整的关于会话结束的详情将在第 15 章节进行讨论。

第 24.2 章节描述了在 Figure1 的完整的消息构成。

在某些用例中，对于代理来说可能非常有用，在整个会话期间可以看到两个终端之间在 SIP 信令路径上的所有消息。例如，如果 biloxi.com 代理服务器希望保持除了初始 INVITE 以外的 SIP 消息，它对 INVITE 添加一个要求的路由 header 值，这个值我们称之为 Record-Route，用来解析主机或者代理的 IP 地址。因为 Bob 软电话（这个消息会通过 200 (ok) 传递给 Bob）和 Alice 软电话将会收到这个消息，在整个 dialog 过程中保存这个消息。biloxi.com 代理服务器将收到和对 BYE 代理转发这个 ACK, BYE 和 200 (OK)。每个代理可以独立决定收到的后续的消息，消息将被通过所有选择接收的代理发送，这些代理来接收这个消息。这个能力经常被代理使用，代理通过这个能力可以提供 mid-call 功能或者呼叫期间的控制功能。

注册是另外一个 SIP 常用的操作。注册是一种方法，它可以使得 biloxi.com 服务器获知当前 Bob 的位置。Bob 软电话基于初始化处理，在一定周期内 Bob 软电话对在 biloxi.com 的服务器发送 REGISTER 消息，我们称之为 SIP registrar 或者 SIP 注册。REGISTER 消息关联 Bob 的 SIP 软电话或者 SIPS URI (sip:bob@biloxi.com)，这个机器是当前 Bob 写入记录的地址（它在 Contact 头中传输 SIP 或者 SIP URL）。这个注册会写入此关联，也被称之为在数据库中的绑定或者定位服务，此定位服务可以使用在 biloxi.com 域的代理中。经常，对于一个域的注册服务器需要和这个域的代理协同工作。这里一定要注意，区分不同类型的 SIP 服务器功能概念是非常重要的，它们区别是在于逻辑处理的不同，而不是物理上，形体上的不同。

Bob 不仅仅局限于从一台设备注册。例如，Bob 的两个终端设备，一台在家里面，另外一台在办公室都发送注册消息。两台设备的消息都保存在定位服务中，允许代理执行各种对 Bob 终端的定位查询。

同样的道理，同一时间，多个用户可以注册到一台单个设备上。

定位服务仅是一个抽象概念。通常情况下，它包括一些必要的信息，它支持代理输入一个 URL，并且接收零个或多个 URL，这些 URL 告诉代理往什么地址发送请求。注册

是一种方式来创建这些信息，但也不仅仅是这一种方式。任意映射功能可以通过管理员自行决定。

最后，一定要注意，在 SIP 中注册是用来路由入局的 SIP 请求的，它不能充当出局的授权请求的角色。在 SIP 中，签权和身份认证的处理可以基于逐一请求的方式，使用 challenge/response 机制的方式来处理，或使用更低一层的方案来处理，这种方案的讨论在第 26 章节有所描述。

完整的关于 SIP 注册的消息细节示例在第 24.1 章节讨论。

其他的 SIP 操作，例如查询 SIP 服务器的能力或终端使用 OPTIONS，或使用 CANCEL 取消正在进行的请求等流程将会在后续之间进行讨论。

## 5 Structure of the Protocol

SIP 是按照一定的层级结构创建的协议，这表示它的行为是根据一系列各自相对独立的处理流程来实现，每个处理阶段之间是松耦合关系。协议行为描述为多个层级，这样是为了支持呈现的目的，支持标准的函数描述，这些描述涉及了单一环境的多个网元。它不能通过任何方式来决定部署。当我们说一个网元要“包含”一个层级时，我们的意思是它符合一系列在这个层级所定义的规范规则。

不是每个通过协议设定的网元都包含在每个层级。进一步说，在 SIP 协议中设定的网元是逻辑网元，不是物理形态的网元。一种物理实现可以选择不同的逻辑网元来执行，也许可以基于依次事务对事务的处理方式进行。

SIP 结构的最低层是语法和解码层。解码是通过增强的 Backus-Naur Form grammar (BNF) 语法来实现的。完整的 BNF 在第 25 章节有介绍。整个 SIP 消息结构的总览在第 7 章节介绍。

第二层是传输层。它定义了用户如何发送请求，如何接收响应和服务器如何通过网络接收请求和发送响应。所有 SIP 网元都包含一个传输层。传输层是在第 18 章节进行描述。

第三层是事务层。事务是 SIP 的基础核心模块。事务是一个由用户端事务对服务器端事务发送的请求，用户端使用传输层对服务器端发送事务请求，所有的服务器端事务

所携带响应消息返回到客户端。事务层处理应用层的重传，对请求响应的匹配和应用层超时管理。任何由用户代理（UAC）完成的任务通过使用一系列的事务来触发。具体的关于事务的讨论在第 17 章有讨论。用户代理包含了一个事务层，就像是一个状态代理。无状态代理没有包含事务层。事务层有一个用户端模块（称之为用户事务）和一个服务器端事务模块（称之为服务器端模块），每个模块通过各自的有限状态机来呈现，状态机来处理每个特别的请求。

在事务层上面的是事务用户（TU）。每个 SIP 实体，除了无状态代理都是一个事务用户。当一个 TU 希望发送一个请求时，它会创建一个用户事务实例，然后把这个实例传递给这个请求，并且携带目的地 IP 地址，端口和传输请求。一个创建了用户事务的 TU 也可以取消这个用户事务。当用户取消了一个事务时，它会请求服务器停止进一步的处理，变换到退出的状态，这个状态是这个事务初始化前的退出状态，并且生成对这个事务生成错误响应消息。这个处理过程是通过一个 CANCEL 请求来处理，它构成了属于自己的事务，但是仅针对这个被取消的事务（第 9 章）。

SIP 网元也就是用户代理用户侧，服务器，无状态代理，有状态代理和注册。SIP 网元包含了一个核心模块，这个核心模块来对各自其网元进行区别处理。在核心网元模块中，除了无状态代理以外，其他的网元都是事务用户。这里，UAC 和 UAS 的核心处理流程来自于 method。关于 methods 支持了多种规则和定义（第 8 章）。对于 UAS 来说，这些规则控制请求的结构；对于 UAS 来说，这些规则控制请求的流程和生成响应消息。因为，注册在 SIP 协议中扮演着一个非常重要的角色，一个处理注册的 UAS 会设定一个特别的名称注册。在第 10 章中描述了 UAC 和 UAS 核心的对 REGISTER method 的处理方式。第 11 章描述了 UAC 和 UAS 核心对 OPTIONS method 的处理方式，它决定 UA 的支持能力。

某些其他的请求是在 dialog 中发送。一个 dialog 是一个介于用户代理之间的 peer-to-peer SIP 关系，这种关系存在于一定时间内。这个 dialog 支持介于用户代理之间的消息的顺序传递和正确的请求路由。在这个细节规定中，INVITE method 是唯一的方法来创建 dialog。当一个 UAC 在 dialog 中发送一个请求时，它会遵守一般的 UAC 规则，这些规则在第 8 章加以讨论，它也会遵守 mid-dialog 请求时的规则。第 12 章讨论在 dialog 和表述它们的结构和维护流程。

在 SIP 协议中，最重要的 method 是 INVITE method，它用来创建参与方之间的会话。一个会话是参与方的汇集和它们之间通信的媒体流交互。第 13 章讨论了如何实现会话发起，这些导致了一个或者多个 SIP dialog 生成。第 14 章讨论了如何在一个 dialog 中通过 INVITE 用法来修改会话属性。最后，在第 15 章中讨论如何结束一个会话。

第章节 8, 10, 11, 12, 13, 14, 和 15 完整讨论了 UA core(第 9 章描述了取消流程，这个取消流程支持都支持 UA core 和 proxy core)。第 16 章讨论代理的网元，这些网元支持了介于两个用户代理之间的信息路由。

## 6 Definitions

以下定义对 SIP 协议非常重要。

Address-of-Record: 一个 address-of-record (AOR) 是一个 SIP 或者 SIPS URI 地址，它指到了一个域，同时它支持定位服务。定位服务可以映射这个 URL 到其他的 URL，其他的 URL 可能绑定了用户的有效性和可用性。典型的示例是，定位服务通过注册来实现。AOR 经常被认为是一个用户的“公开地址”。

Back-to-Back User Agent: 背靠背用户代理 (B2BUA) 是一个逻辑实体，它作为一个 UAS 来接收一个请求，处理这个请求。为了决定如何应答这个请求，它的工作方式又类似于一个 user agent client (UAC) 来生成请求。不像代理服务器，它会保持 dialog 状态，并且必须介入到整个它所创建的 dialogs 中发送的所有请求。因为，它自己本身就是一个 UAC 和 UAS 的结合体，本身并不需要特别明确的定义来定义它的行为。

Call: 呼叫是一个非正式的名称，它指的是介于终端之间的通信，通常情况下创建呼叫的目的是为了多媒体的沟通。

Call Leg: dialog 另外的名称[参考链接 31]；已不在此规定中使用。

Call Stateful: 一个代理是有状态呼叫，它具有这样的特征。如果它保持 dialog 整个状态，这个状态一直持续从初始化 INVITE 开始到 BYE 请求结束。一个 call stateful 代理总是一个事务状态，但是事务状态不一定是一个有状态呼叫代理。

Client: 终端用户是任何一个网络的网元，它发送 SIP 请求和接收 SIP 响应。用户端可能，或不可能直接和真人用户进行互动。用户代理终端和代理是终端。

Conference: 一个对媒体会话，它包含了多个参与方。

Core: Core 指定了某些功能，这些功能专门针对某些 SIP 实体的参与方类型。例如，具体指定了是一个状态或者非状态代理，用户代理或者注册。除了某些非状态代理以外，所有的 core 功能都是事务用户。

Dialog: dialog 是一种端对端的 SIP 关系，它介于两个 UA 之间，这两个 UA 在一定时间内维持着某种绑定关系。一个 dialog 的创建是通过 SIP 消息，例如对 INVITE 请求的 2xx 响应。一个 Dialog 是通过一个 call identifier, local tag, 和一个 remote tag 来定义的。Dialog 以前称之为一个 call leg, call leg 在 RFC2543 定义。

Downstream: 在事务内的一个消息前转的，它涉及到了一个请求流程，这个请求流程是从用户代理客户端到用户代理服务器端的处理方向。

Final Response: 是一个响应消息，它结束 SIP 事务，相反的，一个 provisional response 则不会结束事务。所有 All 2xx, 3xx, 4xx, 5xx 和 6xx responses 都是最终响应消息。

Header: 头是 SIP 消息的组件，它传递消息的信息。它由一系列的头字段值构成。

Header Field: 头是 SIP 消息的组件。一个头字段或者头字段可以表现为一个或多个头字段值。每一行头字段值包含头字段值名称和一个或者多个头字段值。如果有多个头字段值的话，可以通过逗号分开。一些头字段值仅有单行头字段，它总是以单行头字段出现。

Header Field Value: 一个头字段或者字段是一个单个数值；它由零个或多个头字段值构成。

Home Domain: 主机域对 SIP 用户提供服务。典型的解释是，这是一个域名，它出现在注册 AOR 的 URL 中。

Informational Response: 类似于一个临时响应。

Initiator, Calling Party, Caller: 一方发起一个会话（和 dialog），它带着一个 INVITE 请求。一个呼叫方始终保持一个角色，这个角色从它开始发送这个初始的 INVITE 开始计算，这个 INVITE 创建了一个 dialog，一直到结束这个 dialog。

Invitation: 一个 INVITE 请求。

Invitee, Invited User, Called Party, Callee: 一方接收一个 INVITE 请求，这个请求的目的是为了创建一个新的会话。被呼叫方始终保持这个角色，这个角色从它开始接收这个 INVITE 开始计算，一直到 dialog 结束，这个 dialog 是由那个 INVITE 创建。

Location Service: 定位服务用来支持一个 SIP 重定位或代理服务器来获得关于被呼叫方可能存在的地址信息。它包含一个绑定的 address-of-record 列表数值，这些从零个到多个 contact 地址。这个绑定关系可以通过多种方式来创建或者删除；此协议细节中定义了一个 REGISTER method 来更新绑定关系。

Loop: 一个已到达代理的请求，经过前转以后，后来又返回到同样的代理。当这个请求第二次到达代理时，这个请求的 Request-URI 确认是第一次的请求，并且其他影响代理操作的头字段值不会改变，因此代理会在这个请求中做出和第一次同样的处理决定。回环的请求是一种错误，流程会检测回环请求，通过协议本身来处理这种回环请求。

Loose Routing: 如果代理遵守本规范来处理路由头字段，代理会被告知是一个松散路由。这些流程从一系列的代理中分开了目的地请求（出现在 Request-URL 中），代理所遵守的机制被称之为松散路由。

Message: 消息是在 SIP 网元之间发送的数据，它是协议的一部分。SIP 消息可以是请求或者响应消息。

Method: method 是一个基本功能，一个请求在服务器端被激活。Method 在请求自己的消息中传输。常见的 methods 是 INVITE 和 BYE 请求。

Outbound Proxy: 它是一个代理，负责接收从客户端发出的请求，即使它可能不是一个通过 Request-URI 解析度服务器。通常情况下，一个 UA 可以通过 outbound proxy 手动配置，或通过自动配置协议进行学习。

Parallel Search: 在并行查询中，一个代理会对可能存在的用户位置发送几个请求，这些可能存在的用户位置用来接收请求。而且，并行查询也不是发送一个请求，然后等待收到这个请求的最后响应，然后发送接下来的请求。它不会等到前面的请求响应收到以后再发送下一个请求。

Provisional Response: 它是临时响应，临时响应表示服务器端的处理进程，但是临时响应不会结束 SIP 事务。1xx 是临时响应，其他的响应是最终响应。

Proxy, Proxy Server: 代理是一个中间实体，它的工作方式既是一个服务器端，又是一个客户端，作为客户端的作用是支持其他客户端发起请求。代理服务器基本功能是扮演路由的角色，它的工作就是确保请求被发送到比较接近目标用户的其他实体。代理也可以执行一些强制的策略（例如，确保用户被允许呼叫）。代理可以解析请求消息的部分消息内容，如果必要的话，在一个请求消息被前转之前，代理可以重写请求消息的部分参数内容。

Recursion: 递归处理。当用户在响应中的 Contact 头字段中产生一个或多个 URLs 的新请求时，用户会在 3xx 响应中产生递归。

Redirect Server: 重定向服务器是一个用户代理服务器，它会对接收的请求产生 3xx 响应，重新定向用户，让用户联系其他可选的 URL 列表中的 URI 地址。

Registrar: 注册服务是一个注册服务器，它用来接受 REGISTER 请求，负责把注册服务器接受的信息保存到定位服务所支持的 domain，这个 domain 是注册服务器负责。

Regular Transaction: 正常事务是任何带 method 的事务，带 INVITE, ACK, 或者 CANCEL 的 method 的除外。

Request: 请求是一个由用户端发送到服务器的 SIP 消息，请求的目的是触发一个特别的操作。

Response: 响应是一个由服务器端发送到用户端的 SIP 消息，其目的是说明请求发送后服务器端回复的状态。

Ringback: 回铃是一种信令音（回铃音），它是由呼叫方应用程序生成，用来表示被呼叫方已经被提示（被呼叫方正在振铃状态）。

Route Set: 路由集是一组有序 SIP 或者 SIPS URI 的集和，它用来表示当发送一个特别的请求时所经过的代理列表。路由集通过路由头，例如 Record-Route 或者经过配置后获得。

Server: 服务器是网络中的一个网元，它用来接收请求，然后对其进行服务支持，并且对其请求返回响应消息。服务器的实例包括代理，用户终端服务器，重定位服务器和注册服务器。

Sequential Search: 在顺序查询中，代理服务器按照顺序尝试查询每个 contact 地址，并且，只有当上一个查询返回最终响应后才进行下一个查询的新的处理。2xx 或者 6xx 的最终响应总是结束顺序查询处理。

Session: 在 SDP 规范中：“一个媒体会话是一系列媒体发送方和媒体接收方，以及从发送方到接收方之间的媒体数据流。一个媒体会议就是一个媒体会话的举例。”(RFC 2327 [1])（对 SDP 定义的会话来说，一个会话由一个或多个 RTP 会话构成）。就像定义中的那样，对于同一会话来说，一个被呼叫方可以被不同的呼叫方多次邀请。如果使用了 SDP，会话通过 SDP 用户名称，会话 ID，网络类型，地址类型 和在地址单元中的原始值域构成。

SIP Transaction: 一个 SIP 事务会在客户端和服务器端之间，它由第一个由客户端发出的请求开始到服务器端最终响应的所有消息构成（非 1xx 消息）。

如果请求是 INVITE，并且最终响应是一个非-2xx 消息的话，这个事务也会对这个响应包括一个 ACK。这个对于 200 OK (INVITE 的响应) 的 ACK 来说，它是一个独立的事务处理。

Spiral: spiral 是一种“螺旋式”处理方式，它是一个 SIP 请求，返回到代理，然后，代理再把这个请求前转到其他的终端，但是处理的流程不同，也导致和初始的 URL 不同。通常情况下，螺旋式处理方式表示请求中的 Request-URI 和上一次抵达的请求中的 URL 是不同的。注意，螺旋式的处理流程处理不是一个错误条件，它和 loop (回

环完全不同）。典型的使用场景是呼叫前转的处理。A 用户呼叫 joe@example.com。这里的 example.com 是一个代理，它会前转到用户 Joe 的电脑终端，接下来，joe 会把这个呼叫前转继续前转到 bob@example.com。这里的 请求其实又回到了同一代理 example.com。但是，这种处理方式不是 loop 环境。因为，这里的请求发生了变化，它触发了不同的呼叫，这里的 URL 是 bob@example.com，不是 joe@example.com。所以，它的处理是有效的处理流程，被认为是一种螺旋式的处理。而在 loop 中，它的处理流程和 Request-URI 是保持不变的，代理重复处理同样的流程，因此导致触发错误条件。

Stateful Proxy: 状态代理是一个逻辑实体，它按照规范中请求处理的流程保持用户端和服务器端之间的事务状态机的处理状态，也就是所谓的事务状态代理。状态代理的执行在第 16 章做了进一步的说明。状态代理（事务）和呼叫状态代理是不同的。

Stateless Proxy: 无状态代理是一个逻辑实体，它不会保持用户端和服务器端之间的事务状态机。无状态代理前转从下游收到的每个请求，前转从上游收到的每一个响应。

Strict Routing: 如果代理被称为严格代理表示这个代理遵守 RFC2543 的路由处理规则，和一些比较早的 RFC 版本规范。当 Router 头出现时，那个规范会引起代理破坏 Request-URI 的内容。严格路由的流程不在本规范中使用，本规范支持松散路由的处理。执行严格路由的代理也被称为严格路由器。

Target Refresh Request: 目标刷新请求是在一个 dialog 中发送，这个请求可以修改 dialog 中的远端目标。

Transaction User (TU): TU 是处理协议层，它存在于事务层。TU (事务用户) 包括 UAC core, UAS core 和 proxy core。

Upstream: 它表示在事务中的转发消息方向，针对的是从用户代理服务器端返回到用户代理客户端的响应流程。

URL-encoded: 一个通过规范 RFC2396, 章节 2.4 [5]解码的字符串。

User Agent Client (UAC): 用户代理客户端是一个逻辑实体，它创建了一个新的请求，并且使用客户端事务状态机发送请求。UAC 的角色是仅维持那个事务的时长。换句

话说，UAC 是一款软件，它发起一个请求，它以 UAC 的方式工作。如果它后续收到一个请求，它以用户代理服务器的方式来处理事务流程。

UAC Core: 一系列 UAC 的请求处理功能，它在事务层和传输层以上。

User Agent Server (UAS): 用户代理服务器是一个逻辑实体，它对 SIP 请求生成一个响应。响应接受，拒绝或转发请求。它的角色是仅维持事务时长。换句话说，它是一款软件来响应请求，它以 UAS 的方式工作。如果在后续状态中收到一个请求，它以用户代理客户端的方式来处理事务流程。

UAS Core: 一系列 UAS 的请求处理功能，它在事务层和传输层以上。

User Agent (UA): UA 是一个逻辑实体，它能以用户代理客户端或者用户代理服务器端的方式工作。

UAC 和 UAS 的角色，代理和转发服务器都是基于事务对事务的基础上定义的。例如，用户代理以 UAC 的方式发起一个呼叫时，发送请求时，它的工作方式是 UAC；当从被呼叫方收到一个 BYE 请求时，它的工作方式是 UAS。同样的道理，同样的软件，它可以以代理服务器的方式工作来处理请求，也可以以转发服务器的方式工作来处理下一个请求。

代理，定位服务器和注册服务器都是逻辑实体。在部署时，它们可以集成为一个单一的应用服务器。

## 7 SIP Messages

SIP 是基于文本的协议，使用的是 UTF-8 charset (RFC 2279 [7])。

一个 SIP 消息可以是从客户端到服务器端的请求消息，也可以是服务器端到客户端的响应消息。

虽然它们的语法规规范和字符设置不同，请求 (第 7.1 章节) 和响应 (第 7.2 章节) 消息都使用 RFC2822[3] 的基本格式来处理。(例如，SIP 允许支持头字段，这些字段对 RFC2822 来说是无效的头字段)。两种类型的消息由一个起始行，一个或者多个头，一个表示头结束的空行和一个可选的消息体表示。

```

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

```

起始行，每个消息头和空行都必须以换行符结尾。注意，即使没有消息体，空行也要显示。

除了上面字符的不同以外，很多 SIP 消息和 SIP 头语法都是遵守 HTTP/1.1 的语法。于其在这里重复语法和语义的定义，这里，我们建议使用 HTTP/1.1 规范的[HX.Y] 的部分作为参考 (RFC 2616 [8])。

但是，SIP 不是 HTTP 的拓展。

## 7.1 Requests

SIP 请求通过在起始行带一个 Request 行和其他的 method 加以区别。一个请求行包含 method 名称，一个 Request-URI，和由单空格字符分开的协议版本。

请求行以换行符 CRLF 结束。可以允许无回车或换行，除了在以换行符结束的序列中。不允许在任何网元中有任意数量的空白格 (LWS) 存在。

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

**Method:** 此规范定义了六个方法: REGISTER 支持注册联系消息，INVITE，ACK，和 CANCEL 支持会话创建，BYE 支持结束会话，OPTIONS 支持对服务器的能力查询。SIP 拓展中定义了其他的方法。

**Request-URI:** Request-URI 是一个 SIP 或 SIP URL 在第 19.1 章节介绍，它或者是一个标准的 URL(RFC2396[5])。它表示这个用户或这个服务被记录。Request-URI 不能包含非转义符空格或控制字符，不能以"<>"方式出现。

SIP 网元中可能支持 Request-URIs，不一定是 sip 或者 sips，也可能是其他的 URL schemes 形式，例如"tel"，这是 RFC2806 [9]的 URL schemes。SIP 网元可以在它们的处理过程中使用任何机制转译非 SIP，最终生成 SIP URI，或者其他 scheme。

SIP-Version: 请求和响应都包括在使用的 SIP 版本，并且遵守 [H3.1] (HTTP 替代了 SIP, HTTP/1.1 替代了 SIP/2.0)，这里涉及了版本顺序，遵从要求和版本更新数量。为了遵从此规范，应用程序发送到 SIP 消息必须包括 SIP 版本 "SIP/2.0"。此 SIP 版本字符串是大小写敏感，但是使用时必须发送大写字母。

不像 HTTP/1.1，SIP 把此版本号看作为一个一般字面字符串。在实际使用时，这应该没有什么不同。

## 7.2 Responses

SIP 响应消息和请求消息不同，响应消息包含一个 Status-Line 作为一个起始 start-line。在每个网元中，一个 Status-Line 由响应版本，然后跟随一个数字类型的状态码以及其关联的文本短语，通过一个单空格字符分开。

除了在最后的 CRLF 顺序中，可以允许无 CR (回车) 或者 LF (换行) 转义字符。

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

状态码是一个三位整数的结果代码，它表示一个测试输出的响应理解，满足请求的要求。原因短语的目的是对状态码给予一个短语解释。使用状态码的目的是为了系统的自动处理，而原因短语的目的是方便用户阅读理解状态原因，具有可读性。用户不要求检查或显示原因短语。

这里，此规范建议使用明确的用词来表示原因短语，部署使用时可使用其他的文本。例如，在请求中的 Accept-Language 头中的语言。

状态码的第一个数字定义了响应的级别。状态码后两位没有层级的设置。因为这个原因，任何状态码介于 100 和 199 之间的响应被看作是"1xx response"，任何状态码介于 200 和 299 的响应看作是一个"2xx response"响应，以此类推。以第一个数字为划分类别，SIP/2.0 支持了六个级别的状态响应码：

1xx: Provisional – 请求收到的响应码，表示是临时响应，会继续处理此请求；

2xx: Success – 成功收到处理流程，理解，接受了处理流程；

3xx: Redirection – 需要进一步的流程处理来完成此请求；

4xx: Client Error – 此请求中包含错误语法或不能满足服务器的要求；

5xx: Server Error – 服务器端不能满足一个明确有效请求；

6xx: Global Failure – 任何服务器都不能满足此请求流程。

第 21 章定义了这些级别和描述了其无效码。

### 7.3 Header Fields

在语法和语义方面，SIP 头和 HTTP 头非常相似。在实际应用环境中，SIP header 遵从 [H4.2] 对消息头的语法和对拓展头的规则。但是，后者通过 HTTP 定义，使用了隐藏的空格。此规范和 RFC 2234[10]是一致的，仅使用了明确的空格，并且看作为语法的一个部分。

[H4.2] 也定义了同一域名称的多个头的语法，这些值都以逗号隔离的列表，这些列表可以合并成一个头值。这个应用方式也可以支持 SIP，但是因为具体的规范有所不同。具体来说，任何 SIP 头都以下语法的形式表现

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

可以支持合并同一名称的头成为一个逗号隔离的列表。此 Contact header 支持逗号隔离的列表，除非这个头的值是“\*”。

#### 7.3.1 Header Field Format

头字段域遵从标准的头格式标准，在 RFC2822 第 2.2 章节 [3]定义。每个头由域名，然后冒号(":") 和域值构成。

`field-name: field-value`

消息头顶标准语法在第 25 章定义，然后紧跟一个任意数量的空格。但是，在部署使用时应该避免基于头字段和冒号之间的空格，在值域和冒号之间使用一个单空格。

<code>Subject:</code>	<code>lunch</code>
<code>Subject :</code>	<code>lunch</code>
<code>Subject</code>	<code>: lunch</code>
<code>Subject:</code>	<code>lunch</code>

因此，以上格式都是有效的，建议使用最后的格式。

Header 头字段可以扩展为多行，实现方式是通过在每一行前添加至少一个 SP 或 HT tab 键来实现。在下一行开始前的换行符和空格被看作是一个单 SP 政府。因此，以下几种格式表达的意思是相同的：

```
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!
```

带不同域值的头的相对顺序不是非常重要。但是，规范推荐，为了支持代理处理，这些头(例如，Via, Route, Record-Route, Proxy-Require, Max-Forwards, 和 Proxy-Authorization) 应该出现在消息体的顶部来支持代理的快速解析。头的相对顺序和其对应的名称是非常重要的。如果或只有如果那个头的域值定义为以逗号分割的列表时(第 7.3 章)，具有同样名称的多个头值可以出现在消息中。它必须可以支持多行头值可能合并为一对"field-name; field-value"的形式，而没有改变消息的语义，首先预设每一个接下来的头值，然后以逗号分开。这个规则对 WWW-Authenticate, Authorization, Proxy-Authenticate, 和 Proxy-Authorization 头是一个例外。

带它们名字的多头值域可能出现在消息中，但是，因为它们的语法没有遵从第 7.3 章节的标准格式，它们不允许合并为单头行域值。

使用时必须可以处理同样名称的多头值，无论是每行单值合并的头或是逗号分隔的方式。

以下各组头值是有效，相等的：

```
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>
```

每个组的值是有效的，但是又表达各自不同含义：

```
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>
```

头的名称格式是通过每个头名称来定义的。它总是是 UTF8 文本八位字节不确定度顺序出现或空格，标志符，分隔符和带引号的字符出现。许多存在的头会附加到通过标准规范值，通过分号的方式，分隔参数名称，参数值，具体格式为：

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

虽然任意数目的参数可以附加到头上，但是，任何已给定的参数名称不能出现第二次。

当对比头值时，头名称总是大小写不敏感的。要不然，这个头是一个指定的头，它已经声明了值域名称，参数名称和参数值是大小写不敏感的头。标记符总是大小写不敏感的字符。除非，这个标记符已经声明其属性，否则，被引号标注的字符值是大小写敏感的值。例如，

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

等同于

CONTACT: <sip:alice@atlanta.com>;ExPiReS=3600

和

Content-Disposition: session;handling=optional

等同于

content-disposition: Session;HANDLING=OPTIONAL

以下这两组是不相同的：

Warning: 370 devnull "Choose a bigger pipe"

Warning: 370 devnull "CHOOSE A BIGGER PIPE"

### 7.3.2 Header Field Classification

一些头字段仅在请求或者在响应中有一定的合理性。它们被称之为 request header fields 和 response header fields。如果一个头字段出现在消息体中，没有匹配任何头的层级（例如，请求的头出现在响应的消息体中），它则必须被忽略掉。第 20 章定义了头字段的各种层级类别。

### 7.3.3 Compact Form

SIP 提供了一种机制以压缩的形式来表达普通的头。当传输很大的消息体的消息内容时，这种方式也比较有用，例如当使用 UDP 传输时，如果内容数据超过 MTU 极限后，使用压缩的格式就可以满足最大 MTU 支持。压缩格式在第 20 章定义。压缩格式可以在任何时候在没有改变消息语义时替换为比较长的格式。

头字段值可以以比较长的格式或者压缩格式出现在同样的消息体中。在使用时每个头都必须支持比较长的格式和压缩格式。

## 7.4 Bodies

除非另外提示，Requests(请求)可能包括消息体，这种请求包括一个新请求，新请求在本规范的新拓展中定义。消息体解析依赖于请求方式 method。

对响应消息来说，请求方式和响应状态码决定消息体类型和消息解析。所有的响应可能包括在一个消息体。

### 7.4.1 Message Body Type

消息体的网络媒体类型必须通过 Content-Type 头给定。如果消息体已经处理过编码流程，例如压缩，那么必须通过 Content-Encoding 头声明；否则，必须忽略 Content-Encoding。如果可行的话，声明消息体字符串为 Content-Type 头的一个部分。

在 RFC 2046 [11] 定义的 "multipart" MIME 类型可以在消息体中使用。在使用中，如果远端部署方请求通过了一个 Accept 头，这个头没有包含 multipart，那么，发送的请求中包含多方消息体必须发送一个会话描述作为一个非 multipart 消息体。

SIP 消息可以包含二进制消息体或消息体的部分。当发送方没有提供明确的字符参数设置时，被定义的 text 的媒体子类型有一个默认字符设置值"UTF-8"。

#### 7.4.2 Message Body Length

以 bytes 为单位的消息体长度是由 Content-Length 头提供。第 20.14 章描述了头内容的具体细节。

HTTP/1.1 中的分块传输编码不能在 SIP 中使用。（注意：为了以一系列的传输来分块数据，分块传输编码修改了消息体，每一个块都有各自的大小指示）

### 7.5 Framing SIP Messages

不像 HTTP，SIP 部署使用了 UDP 或其他的不可信赖的数据包协议。每个数据包传输一个请求或者响应。参考第 18 章介绍了使用非可靠性传输的限定。

通过以数据流方式传输方式来处理 SIP 消息的机制必须在 start-line 之前忽略掉任何回车换行字符[H4.1]。

Content-Length header 头的值用来定位数据流中的每个 SIP 消息结束位置。当 SIP 消息是通过数据方式传输时，它总是出现在这里。

## 8 General User Agent Behavior

一个用户代理表示一个最终的系统架构。它包含一个用户代理客户端（UAC），用来产生请求，和一个用户代理服务器端，它用来对请求产生响应反馈。UAC 用户代理客户端具备产生请求的能力，UAC 产生请求是由外部刺激和驱动的流程而产生（例如，用户点击了一个按钮和 PSTN 线路上的一个信号），并且对响应进行处理。一个 UAS 代理客户端可以接收一个请求，并且基于用户输入，外部驱动刺激，程序执行结果或者其他机制所产生一个响应。

当一个 UAC 发送一个请求时，这个请求会经过几个代理服务器，这些代理服务器将前转这个请求到 UAS。当 UAS 生成响应时，这个响应会返回到 UAC。

UAC 和 UAS 的处理流程完全依赖于两个因素。首先，这个流程取决于这个请求或响应是否在 dialog 里面还是外面，其次，流程还取决于请求的 method。Dialogs 的讨论将会在第 12 章进行；它们表示一种介于用户代理之间的点对点的关系，这个关系是通过具体的 SIP methods 创建的，例如 INVITE。

在本部分内容中，我们讨论 UAC 和 UAS 的执行处理规则，这个规则是完全独立于 method 的，当处理请求时，这些请求是在 dialog 的外面。这里当然也包括请求自己创建的 dialog。

关于 dialog 外部的对请求和响应的安全处理流程的描述在第 26 章进行。具体来说，介于 UAS 和 UAC 存在的机制是互相验证的过程。通过消息体使用 S/MIME 加密的方式实现一系列私有功能支持。

## 8.1 UAC Behavior

这部分讨论 UAC 的外面 dialog 的运行状态。

### 8.1.1 Generating the Request

一个有效的被 UAC 规范化的 SIP 请求必须最低包括以下几个头字段：To, From, CSeq, Call-ID, Max-Forwards, 和 Via；对所有 SIP 请求来说，这些头字段是强制支持的。

这六个头字段是构建一个 SIP 消息的基础结构，因为它们联合起来为 SIP 通过了最基本的和最重要的路由服务，消息地址，响应路由，限定消息扩展，消息顺序和事务的唯一身份。

这些头字段另外包含了 method, Request-URI, 和 SIP version。

运行在 dialog 外面的请求发送示例包括了一个 INVITE，它用来创建一个会话(第 13 章)和一个 OPTIONS，它用来查询能力支持(第 11 章)。

#### 8.1.1.1 Request-URI

消息的初始 Request-URI 应该在 To 头中设置为 URL 的值。一个需要注意的例外就是 REGISTER method；REGISTER 的 Request-URI 设置方式在第 10 章中讨论。对于安全原因或便利性来说，它可能也不是太方便来设置这些值域为同样的值（特别是，如果在转换期间，初始的 UA 希望 Request-URI 可以被修改的环境中）。

在某些特定的环境中，一个已存在的 route 状态可以影响 Request-URI 的消息。一个已存在的路由系列是一系列有序 URIs，这些 URLs 确认服务器链，UAC 将会发送出去的请求，这些请求是 dialog 外部的请求。通常情况下，这些 URL 在 UA 端通过一个用户或服务商手动配置，或者通过其他的非 SIP 机制来配置。当服务商希望配置 UA 支持一个 outbound proxy 时，规范还是推荐需要提供一个已存在的路由系列，设置为一个单 URI 作为一个 outbound proxy。

当出现了一个预先存在的路由表时，如在 12.2.1.1 所描述的中，映射 Request-URL 和 Router 头值的处理流程必须被遵守（即使没有 dialog），使用所期望的 Request-URI 作为远端的目标 URL。

#### 8.1.1.2 To

首先 To 头也是最重要设定了期望的请求逻辑，或者用户的 address-of-record，或者是一个请求目标资源。这可能是或者不是最终请求接收方。To 头可能包含一个 SIP 或者 SIPS URL，但是，如果在其他所要求的场景中，它也可以使用其他的 URL schemes（例如，tel URL (RFC 2806 [9])）。所有的 SIP 部署必须支持此 SIP URI scheme。任何支持 TLS 部署的，必须也支持 SIPS URI scheme。To 头支持一个显示名称。

UAC 可以通过多种方式学习如何对一个特别的请求映射 To 头。通常情况下，用户建议通过人机界面输入 To 头，也许通过人工输入 URL 或从地址簿中选择其地址。很多情况下，用户没有输入完整的 URL 地址，而是输入一个数字字符串或者字母（例如，“Bob”）。这是 UA 的自定义的输入方式，用户自己解析这个输入结果。使用字符构建 SIP URL 的用户部分应用在 UA 期望名字可以被解析为一种域名格式，植入到 SIP URL 中的@符号前（例如，sip:bob@example.com）。使用字符构建 SIPS 的用户部分应用在用户希望通信在安全状态，名称可以被域名解析。右侧域名经常是请求者的主机名称，支持主机域处理出局的请求。对于某些功能来说非常有用，例如，“快速拨号功能”。快速拨号功能要求解析主机域名的用户部分内容。tel URL 可以使用在某

些环境中，UA 不需要设定域名，只是解析用户已输入的电话号码。更准确地说，每个请求通过的 domain 都会有这样的机会。举例，一个在机场的用户可能登录系统，通过一个 outbound proxy 发送请求。如果他输入号码是“411”的话（这个号码是美国当地号码查询系统），这个号码需要解析，然后通过在机场的 outbound proxy 做进一步处理，而不是用户的主机 domain 处理。这种情况下，tel:411 就是一个正确的选择路由。

一个在 dialog 外面的请求不能包含一个 To tag; 请求中的 To 来确认 dialog 的 peer。因为没有创建 dialog，因此也没有 tag 出现。

关于 To 头字段的进一步介绍，请参阅第 20.39 章节。

以下是一个有效的 To 头字段的示例：

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

#### 8.1.1.3 From

From 指示初始请求的逻辑实体，可能是用户 address-of-record 地址。就像 To 头值一样，它包含一个 URL 地址和可选显示名称。它被 SIP 网元用来决定一个请求所需要的处理规则（例如，自动拒绝呼叫）。这是非常重要的规则处理，在一个正在运行的 UA 中，From 头不能包含 IP 地址和这个主机的 FQDN，因为它们都不是逻辑名称。

From 头支持一个显示名称。除了正确的语法以外，一个 UAC 应该使用这个显示名称 "Anonymous"，如果客户实体是隐藏状态，则是一个无实际意义的 URL（例如，sip:thisis@anonymous.invalid）。

通常情况下，在一个指定 UA 生成的请求中，其 From 头的值是由用户或者用户本地域名管理员预设临时值。如果一个指定的 UA 用来支持多个用户的话，它可能带有一个可切换到属性设置，这个属性设置文件包括一个 URL，这个 URL 和其用户属性实体文件相对应。请求接收方能验证请求的发起方身份，以便确认它们在 From 报头的身份声明（第 22 章规范了更多关于验证的机制设定）。

From 报头必须包含一个由 UAC 选择的新的“tag”参数。具体选择细节查看第 19.3 章。

更多关于 From 头字段细节，参考第 20.20 章。

例如：

```
From: "Bob" <sips:biloxi.com> ;tag=a48s
From: sip:+12125551212@phone2net.com;tag=887s
From: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8
```

#### 8.1.1.4 Call-ID

Call-ID 头字段的工作方式类似于一个唯一的标识符，它用来成组一系列的消息。在一个 dialog 处理过程中，任何一方 UA 发送的所有请求和响应都必须包含相同的 Call-ID。每个 UA 注册中的 Call-ID 应该是相同的。

在一个外部 dialog 由 UAC 创建的请求中，Call-ID 头必须由 UAC 选择，在整个处理和时间段上，它可以作为一个全局的唯一标识，除非其他设定的 methods 处理流程修改它。所有 SIP UA 必须有其含义来确保这个它们生成的 Call-ID 头不会被其他 UA 不经意生成一个新的 Call-ID。注意，当获取到请求时，对于某些失败响应处理时，这些失败响应针对此请求要求一个重新修正（例如，认证流程），这些获取到的请求不会认为是一个新的请求，因此，它们不需要一个新的 Call-ID。

具体细节规范请参考第 8.1.3.5 章。

规范推荐使用 cryptographically random identifiers(RFC1750 [12]) 来生成 Call-ID。部署格式可以使用此格式"localid@host"。Call-ID 是大小写敏感的，可以进行一比特一比特的简单对比。

使用 cryptographically random identifiers 提供了对会话的保护，防止被黑客篡改，同时也降低了唯一 Call-ID 的相似度冲突。

对于请求来说，不能通过配置或者界面来提供 Call-ID 头选项选择。

关于更多 Call-ID 头的说明，参考第 20.8 章。

示例：

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

#### 8.1.1.5 CSeq

CSeq 头的目的是对事务确认和排序。它由一个序列号和一个 method 构成。这个 method 必须匹配请求的 method。对于 dialog 之外的非-注册请求，此序列号码是一个任意值。这个序列号码必须是一个可表达的值，此值是一个 32-bit unsigned 整数，并且它必须少于  $2^{31}$ 。只要它遵守以上指南，客户端可以使用任意机制选择 CSeq 头。

第 12.2.1.1 章节讨论了在 dialog 中 CSeq 的构成方式。

例如：

CSeq: 4711 INVITE

#### 8.1.1.6 Max-Forwards

Max-Forwards 头支持一个有限的跃点数，此跃点数是一个请求从此路径开始的初始点到传输到最终目的地经过的跃点。它有一个整数构成，每经过一个跃点，跃点数会自动减少一个数字。如果这个 Max-Forwards 值在抵达请求的最终目的地前降低到 0，它将被拒绝，同时返回一个 483(Too Many Hops) 错误响应。

UAC 必须在每个请求中插入一个 Max-Forwards 头，发起的请求中初始的这个值应该是 70。这个数值已经足够大，可以保证在一个 SIP 网络环境中没有环路时请求不会被丢弃，但是有时环路发生的时候可能也没有消耗很多的代理资源。用户可以选择比较低的值设置，但是一定要注意，UA 需要了解此网络拓扑环境。

#### 8.1.1.7 Via

Via 头值表示一个传输方式，这个传输方式实际上是响应消息发送到地址，这个地址是针对事务和确认来说的。只有下一跳的传输选择以后，Via 头才能被添加（参考使用流程[参考链接 4]）。

当 UAC 创建一个请求后，它必须在请求中插入一个 Via。协议名称和协议版本必须是 SIP 和 2.0。Via 头必须包含一个 branch 参数。这个参数用来确认被这个请求创建的事务。这个参数支持客户端和服务器端。

无论是从空间和时间角度来看，branch 参数在这个 UA 发送的所有请求中具有唯一性。这个规则对 CANCEL 和 non-2xx 响应的 ACK 是例外。就像我们在下面讨论的一样，CANCEL 请求的 branch 参数和这个请求被取消的参数是一样的。同样，在 17.1.1.3 章节的讨论中，一个对 non-2xx 响应的 ACK 响应也有同样的 branch ID，这个 ID 和 INVITE 响应它的一样。

branch ID 参数的唯一属性帮助它作为事务 ID 来使用，它不是 RFC2543 的一个部分。

branch ID 必须按照规范的格式来处理，它必须以字符“z9hG4bK”开头。这七个字符是比较神奇的处理方式（7 被认为可以支持足够的资源，以便保证和旧规范 RFC2543 兼容，旧规范没有选择这个数值，所以不会导致冲突），因此，收到这个请求的服务器端可以决定通过这种方式来构建 branch ID。

Via 头的 maddr, ttl, 和其他请求将在传输层处理（参考第 18 章）。

对于代理来说，Via 处理方式在 6.6 章节的 Item 8 和 6.7 章节 Item 3 说明。

#### 8.1.1.8 Contact

Contact 头提供一个 SIP 或 SIPS URL，这个 URL 用来联系指定的 UA 示例的后续的请求。这个头必须是现存状态，并且在请求中包含完整的 SIP 或者 SIPS URL，可以支持 dialog 创建。在此规范中定义的 methods，它们仅包括 INVITE 请求。对于这些请求来说，Contact 的范围是全局的。这也表示，Contact 头值包含一个 URL，UA 通过这个 URL 接收请求，并且这个 URL 必须是有效的，甚至可以使用在后续的请求中，这些请求已经不在 dialog 范围内的请求。

如果 Request-URI 或最顶部的 Route 头值中包含了一个 SIPS URL，这个 Contact 也必须包含一个 SIPS。

关于更多 Contact 头字段的说明，参考第 20.10 章节。

#### 8.1.1.9 Supported and Require

如果 UAC 支持拓展功能的话，服务器端可以支持对此的响应，UAC 应该在请求中列出一个可选标签 tags 来表示可支持的拓展功能，可选择并且参考（第 19.2 章节）。

列出的可选标签必须来源于在标准规范 RFC 中定义的拓展。这样做的目的是服务器端防止客户端强制使用非标准的，或厂家定义的功能接收服务。在一个请求中，测试类的和信息类的 RFC 拓展明确说明不能使用在 Supported 头中，因此，我们经常看到使用由厂家定义的拓展。

如果 UAC 希望坚持让服务器理解这个拓展功能，UAC 坚持使用这个拓展的话，UAC 必须在请求中插入一个 Require 头，这个头在可选标签中列出来表示需要服务器端支持这个拓展。

如果 UAC 希望在代理中坚持使用这个拓展功能的话，并且需要在代理路径理解这个拓展的话，

UAC 必须在请求可选标签列表中插入一个 Proxy-Require 头表示需要代理支持的拓展。

就像刚才在 Supported 头使用说明的一样，在 Require 和 Proxy-Require 头中的可选标签所支持的拓展必须来自于标准 RFC 定义的拓展。

#### 8.1.1.10 Additional Message Components

在一个新的请求创建以后，前面提到的那些头字段已经被构建。添加任何额外的可选头字段，需要指定具体的 method。

SIP 请求可以包含一个 MIME 解码的信息体。无论在请求中包含什么样的消息体，某些头字段必须进行规范化处理，进行内容中的字符整合。更多关于这些头字段的说明，参考章节从 20.11 到 20.15。

#### 8.1.2 Sending the Request

请求的目的地是通过计算获得的。除非，在发送请求的路径存在一个逻辑策略强制操作，否则请求目的地必须是通过 DNS 处理流程来处理，具体处理描述参考 [4]。如果在 route set 的第一个网元表示是严格路由的话（导致重构请求，具体描述在 12.2.1.1 中讨论），这个处理流程也必须使用在请求的 Request-URI 中。否则，这个流程使用在请求中的第一个 Route 头中（如果存在的话）或如果当前没有 Route 的时候，使用

在请求的 Request-URI。这些流程产生了按序的设置组，包括了地址，端口和参数方式。在处理流程中，URL 作为输入数据，他们的处理流程[4]不依赖于 URL 本身，如果 Request-URI 设置了一个 SIPS 的源，UAC 必须遵从处理流程 [4]，输入的 URL 是一个 SIPS URI。

本地策略可以设定一系列可选的目的地地址。如果 Request-URI 包含一个 SIPS URI，任何可选目的地地址必须支持 TLS 连接。除此之外，如果请求中没有包含 Route 头的话，对可选目的地没有任何限定设置。对于已存在的 route set 来说，通过这样的方式，它可以提供一个简单可选 方式来设定一个 outbound proxy 代理。但是，不推荐使用那种方式来设置一个 outbound proxy；应该通过一个单 URL 使用一个已存在的 route set 替代设置方式。如果一个请求中包含一个 Route 头的话，这个请求应该被发送到最顶部的 Route 地址，但是这个请求也可以被发送到任何服务器，只要 UA 认可其身份，其身份设置是通过 Route 和 Request-URI 策略设定的。具体的策略设定在此规范中（相反的规范设置 RFC 2543）。尤其是一个 UAC 配置了 outbound proxy，它应该尝试发送请求到一个地址，这个地址应该是第一个 Route 头字段地址，而不是调整发送策略，这个策略发送所有消息到这个 outbound proxy。

这样做的目的可以确保 outbound proxies 不添加 Record-Route 头字段值，这些头值将会被丢出后续的请求路径。它允许不能解析第一个 Route URI 的终端对 outbound proxy 代表执行任务。

UAC 应该遵从对 stateful 网元定义的处理流程，这个流程在[参考链接 4]有具体的定义，UAC 应该一直尝试每个地址直到连接到一个服务器地址。每个尝试连接都构成一个新的事务，并且因此每个携带最顶部 Via 头值的传输都会有一个新的 branch 参数值。进一步说，在 Via 头中的传输值被设置为传输方式设定的值。无论这个值怎么设置，这个值是传输方式针对目的地服务器决定的。

#### 8.1.3 Processing Responses

响应消息首先是通过传输层进行处理，然后在传输到事务层。事务层执行自己的处理流程，然后再传递回 TU 处理。在 TU 中的大部分响应处理流程是和具体的 method 相关的。但是，一些基本的处理方式不依赖于 methods 本身。

##### 8.1.3.1 Transaction Layer Errors

在一些情况下，一些由事务层返回的响应消息不是 SIP 消息，是一个事务层错误。当从事务层收到一个超时错误时，它必须被作为一个 408 错误。如果由传输层报告了一个致命的传输错误（通常情况下，是因为一个 UDP 中的致命 ICMP 错误或 TCP 的连接错误），这种状态必须被视为一个 503 错误代码（服务不可用）。

#### 8.1.3.2 Unrecognized Responses

UAC 必须处理任何无类别等级的最终响应消息，并且 UAC 也必须可以处理任何 x00 类别的响应消息。例如，如果一个 UAC 收到一个无类别响应代码是 431 的响应消息，此 UAC 可安全地认为可能在请求中有什么错误发生。一个 UAC 必须视临时响应消息是不同于 100 响应的，它也不会被视为 183 响应消息。UAC 必须能够处理 100 响应和 183 响应消息。

#### 8.1.3.3 Vias

如果在响应消息中出现了一个以上的 Via 头字段值，此 UAC 应该丢弃这个消息。

多个出现的 Via 头字段值是请求发起方置入的值，这些消息是错误设置或者配置文件损害导致。

#### 8.1.4 Processing 3xx Responses

对于转发协议的处理上（例如，301 协议状态码），客户端应该基于转发请求，在 Contact 头中使用 URL (s) 重新构建一个或多个新的请求。这个过程类似于代理对 3xx 类别响应的递归处理，具体的细节参考第 16.5 和第 16.6 章节。客户端启动时携带初始的目标地址列表，其中包含完整的 URL。这是初始请求的 Request-URI。如果客户端希望基于 3xx 重构新的请求，它会置入这些 URLs 在目标列表中。在此规范中，对象的限制是，一个客户端可以选择哪个 URLs 可以置入到目标组设置。当代理递归发生时，客户端处理 3xx 类别响应时，它一定不能再次添加任何已给定的 URL 到目标组中。如果初始的请求已经在 Request-URL 中有一个 SIPS URL，客户端可以选择递归到一个非-SIPS URI，但是应该通知转发用户，这是一个不安全的 URL。

任何新请求可以接收 3xx 响应，这些响应自己包含初始的 URL，这些 URL 作为 contact。可以配置两个地址互相转发。在目标地址组置入一个给定的 URL，其目的是防止无限转发环路发生。

当目的地组设置增加时，客户端可以以任何顺序对 URLs 生成新的请求。一般的机制是在 Contact 头中设置一个“q”参数值来表示顺序。对 URL 生成的请求可以是并行方式或连续生成方式。一种方式是通过连续方式处理递减的 q-值组，并且以并行方式处理在每个 q-值组的 URL。另外一种方式是按照递减的 q-值顺序，仅执行连续处理，在相同 q-值的 contacts 之间任意选择一个值。

如果连接在列表中的 contact 失败，继续连接列表中的下一个地址，直到列表地址连接全部失败。如果地址连接全部失败的话，那么这个请求就已经失败。

失败结果应该通过失败响应码来检测（响应码高于 399）；对网络错误来说，客户端事务层将会对事务层用户报告传输层所发生的错误。注意，一些响应码（详情参见 8.1.3.5）表示请求可被重新获取；重新发送到请求不应该被认为是失败响应。

当收到针对某个特定 contact 地址的失败时，客户端应该尝试下一个 contact 地址。这样就会导致针对发送的新请求创建一个新客户事务。

为了在 3xx 响应中基于一个 contact 地址创建一个请求，除了“method-param”和“header” URI（参考 19.1.1 章节对参数的定义）以外，UAC 必须从目标组中拷贝所有 URL 到 Request-URI。它使用“header”参数为新请求创建 header 头值，覆盖和转发请求中相关的头字段值，具体操作规范参考 19.1.5 章节。

注意，在一些例子中，在 contact 地址中，已经构成通信关系的头可以替换追加到已存在的请求的头中，这些请求的头是在初始转发请求中的头。作为一个一般规则，如果头字段可以接受以逗号隔离的域值列表，那么新的头值可以追加到初始转发到请求中。如果头字段不能接受支持追加多个值的话，初始转发请求中的值可以被在 contact 地址中已经构成通信关系的头字段值覆盖。例如，如果返回的 contact 地址携带了如下值的话：

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

那么，在初始转发请求中的 Subject 头可以被覆盖，但是这个 HTTP URL 很少被追加到任何已存在的 Call-Info 头值中。

规范推荐 UAC 重用在初始转发请求中同样的 To, From 和 Call-ID, 但是 UAC 例如也可以选择更新 Call-ID 支持新的请求。

最后, 一旦一个新的请求生成以后, 新请求使用一个新客户端事务发送这个请求, 因此, 它必须在最顶部的 Via 头中生成一个新的 branch ID。关于这一讨论, 参考 8.1.1.7 章节。

从其他角度所期望的, 转发响应接收方发送到请求应该重用初始请求的头字段和消息体。

在一些例子中, Contact 头字段值可能在 UAC 端被临时或永久缓冲保存, 这取决于收到的状态码和内部超时设置状态。查看 21.3.2 和 21.3.3 章节介绍。

#### 8.1.3.5 Processing 4xx Responses

某些 4xx 响应码要求 UA 有特定的处理流程, 这取决于 method 本身。

如果收到一个 401 (Unauthorized) 或 407 (Proxy Authentication Required) 响应时, UAC 应该遵从认证部分的处理流程第 22.2 章节和第 22.3 章节重新通过请求获取安全消息。

如果收到一个 413 (Request Entity Too Large) 响应 (21.4.11), 这个请求包含的消息体大于 UAS 愿意接收的消息体时。如果可能的话, UAC 应该重发这个请求, 忽略这个消息体或者重发一个小一点的消息体。

如果收到 415 (Unsupported Media Type) 响应 (第 21.4.13 章), 这个请求中包含一个 UAS 不支持的媒体类型。UAC 应该重发这个请求, 这次仅使用在响应消息中列表支持的 content 类型, 这些列出的支持类型在 Accept-Encoding 头中, 或者在 Accept-Language 的 languages 列表中。

如果收到一个 416 (Unsupported URI Scheme) 响应 (第 21.4.14 章), 表示服务器端不支持此 Request-URI 使用的 URI scheme。客户端应该重发请求, 这次的请求使用 SIP URI。

如果收到一个 420 (Bad Extension) 响应 (第 21.4.15 章), 表示这个请求在 option-tag 标签所支持的功能中包含了一个 Require 或者 Proxy-Require 头值, 这个标签所支持

的功能是 proxy 或 UAS 不能支持的。UAC 应该重发这个请求，在响应中忽略任何不支持的拓展头字段。

在以上的例子中，请求重发都是通过创建一个新的请求，在新请求中需要做一定的必要的修改才能满足协商机制。这个新请求重构了一个新的事务，并且也应该和前面的请求具有同样的 Call-ID 和 To 头值，但是 CSeq 应该包含一个新的序列号码，这个新的序列号码高于前面的号码。

对于其他的 4xx 响应，还有其他没有被定义的响应，重发请求可能或者也不可能发生，这依赖于使用的 method 和用户使用场景。

## 8.2 UAS Behavior

当 UAS 处理一个请求处于 dialog 外部的请求 (outside of a dialog) 情况下的请求，规范规定了一系列的处理流程，这些流程独立于 method。第 12 章给出了一个指导，指导说明了 UAS 如何通知请求是一个内部的还是 outside of a dialog。

注意，这里的请求处理是非常恒定的。具体来说，如果接受了这样的请求，所有和此请求绑定的状态修改必须执行。如果拒绝了此请求，所有和此请求绑定的状态修改不能执行。

UASs 应该根据以下步骤处理请求（开始认证处理，检查 method，头字段和以及本章节其他部分处理）。

### 8.2.1 Method Inspection

一旦请求完成认证流程（或者跳过认证），UAS 必须检查请求的 method。如果 UAS 已经识别到 method，但是不能支持请求的 method 的话，它必须生成一个 405 (Method Not Allowed) 响应消息。生成此响应消息的处理流程在第 8.2.6 章节有介绍。UAS 也必须对这个 405 响应消息增加一个 Allow 头。这个 Allow 头必须增加一个列表来表示 UAS 能够支持的 methods 列表。Allow 头的讨论在第 20.5 章节有讨论。

如果服务器端可以支持其中一个 method，则处理流程继续进行。

### 8.2.2 Header Inspection

如果 UAS 不能理解请求中的一个头的话（规范中没有定义这个头字段或规范不支持这个拓展头），服务器必须忽略这个头，继续处理消息。如果出现异常的头字段，UAS 应该忽略异常的头字段值，这些头值对于进一步处理请求是没有必要的。

#### 8.2.2.1 To and Request-URI

To 头字段定义请求的初始接收方，这里的请求是由在 From 头字段中定义的用户发起。因为可能有呼叫前转或其他代理的操作，初始接收方可能是或不是正在处理此请求的 UAS。当这里的 To 头字段不是 UAS 的确认身份时，UAS 可以使用任何策略来决定它是否接受请求。

但是，规范推荐，UAS 接受请求，即使它们不能识别 To 头字段中的 URI scheme（例如，一个 tel:URL），或如果 To 头字段不能处理这个 UAS 的已知的或当前用户。如果，在另一方面，UAS 决定拒绝这个请求，UAS 应该生成一个响应消息和其响应状态码 403 (Forbidden)，并且返回这个响应码到服务器事务层的传输。

如果，Request-URI 定义这个 UAS，它来处理这个请求。如果 Request-URI 使用的一个 scheme 不是这个 UAS 所支持的 scheme，它应该拒绝这个请求，并且返回一个 416 (Unsupported URI Scheme) 响应消息。如果 Request-URI 没有定义一个地址，这个地址是 UAS 愿意为这个请求所接受的地址，它应该拒绝这个请求，并且返回一个 404 (Not Found) 响应消息。典型环境下，一个 UA 会使用 REGISTER method 绑定它自己的 address-of-record (aor) 到一个具体的 contact 地址上，contact 地址可以是多个地址形式。UA 将会看到请求中的 Request-URI 地址和 contact 地址相同。接收 Request-URIs 的其他潜在地址源包括请求的 Contact 头和由 UA 发送到响应地址源，这个响应地址源是创建或刷新 dialogs 的地址。

#### 8.2.2.2 Merged Requests

如果请求中的 To 头字段中没有 tag 标签，UAS core 必须对比检查请求的将要处理的事务。如果 From tag，Call-ID，和 CSeq 完全和将要处理的事务所关联的匹配，但是请求事务的话（匹配规则参见第 17.2.3 章节），UAS core 应该生成一个 482 (Loop Detected) 响应消息，然后把这个响应传递给这个服务器的事务层。

如果同样的请求，这些请求抵达 UAS 多于一次以上的话，这些请求是来自于不同的路径的话，原因可能是进行了分叉 forking 处理。这里的 UAS 处理第一个这样的请求，然后对其他请求返回响应 482 (Loop Detected)。

#### 8.2.2.3 Require

假设 UAS 决定处理请求中的符合规则的参数网元，如果 Require 头出现在当前的消息中，它会检查这个 Require 头字段。

Require 头的作用是 UAC 用来通知 UAS 关于 SIP 拓展的消息，UAC 期望 UAS 支持这些 SIP 拓展，UAS 能够正确处理这些请求中的 SIP 拓展。Require 头的格式在第 20.32 章节中有进一步的描述。如果 UAS 不能理解 Require 头中列出的 option-tag 列表的话，UAS 必须返回一个生成的响应状态码 420 (Bad Extension)。UAS 必须添加一个 Unsupported 头，在这个 Unsupported 头中列出 UAS 不能支持的拓展，这些拓展是请求中的 Require 头所列出的拓展。

注意，Require 和 Proxy-Require 不能使用在 SIP CANCEL 请求中或 ACK 请求，这里的 ACK 请求是发送给 non-2xx 响应消息的。如果这些头值出现在这些请求中时，它们必须要被忽略掉。

一个针对 2xx 响应的 ACK 请求必须仅包含那些出现在初始请求中的 Require 和 Proxy-Require 值。

例如：

UAC->UAS: INVITE sip:watson@bell-telephone.com SIP/2.0  
Require: 100rel

UAS->UAC: SIP/2.0 420 Bad Extension  
Unsupported: 100rel

客户端和服务器端能够互相理解双方所有可选参数值，规范所定义的流程可以确保客户端和服务器端的交互将会快速处理，无任何时延。如果双方参数中，一方不能理解对方的拓展时，处理流程放缓，例如上面的示例。因此，对于客户端和服务器端所支持的拓展都能完全匹配的场景中，交互处理流程会相对较快。如果需要保存一个双向处理，通常需要协商机制来完成。另外，当客户端需要支持的功能，但是服务器端不能理解此拓展功能的话，此头可以移除一些带歧义的拓展

功能支持，例如呼叫处理方面的功能，这些功能可能仅是呼叫流程末端系统感兴趣的。

### 8.2.3 Content Processing

假设 UAS 理解所有客户端请求的拓展功能，然后 UAS 检查消息体的内容和头字段。如果其中任何消息的类型（由 Content-Type 表示），语言（由 Content-Language 表示）或者 编码（由 Content-Encoding 表示）不能被支持，并且 body 部分不是一个可选的值（由 Content-Disposition 头表示），UAS 必须拒绝这个请求，返回错误状态响应码 415（Unsupported Media Type）响应。这个响应必须包含一个 Accept 头的列表，列表表示 UAS 可以理解的消息体类型，在事件中包含 UAS 不能理解的消息体类型。如果 UAS 不能理解请求做包含的内容解码，UAS 响应中必须包含一个 UAS 可接受的 Accept-Encoding 头列表，列表中列出 UAS 所支持的解码方式。如果 UAS 不能理解请求的头中列出的支持的内容语言，响应中必须包含一个 Accept-Language 头，这个头列出 UAS 所支持的语言。除了检查以上这些类型以外，消息体处理还依赖于 method 和类型。更多关于具体内容头的处理，参考第 7.4 章节，还有从第 20.11 到 20.15。

### 8.2.4 Applying Extensions

当生成响应消息时，UAS 不能直接期望使用拓展功能，除非在请求中的头 Supported 头中已经表示支持了这个拓展。如果所希望的拓展不能被支持的话，服务器应该仅依赖基本的 SIP 和其他客户端所支持的拓展来处理。在极少情况下，服务器没有拓展的话就不能处理请求，这个服务器可以发送一个 421（Extension Required）响应消息。这个响应消息表示，如果没有具体的拓展功能，服务器端不能生成一个规范的响应。这些服务器端所需要支持的拓展必须包括在响应消息中的 Require 头中。规范不推荐这种操作方式，因为，一般情况下，因为它会破坏流程的兼容性处理。

任何在 non-421 响应中列出的拓展功能必须包含在响应消息的 Require 头列表中。

当然，服务器端也不能使用没有在请求的 Supported 头中列出的拓展。因此，响应消息中的 Require 头就会只能包含在标准 RFCs 中多定义的可选标签。

### 8.2.5 Processing the Request

假设前面讨论的所有子章节都通过的话，UAS 的处理就会进入到和 method 相关的处理流程。第 10 章节涵盖 REGISTER 请求，第 11 章节涵盖 OPTIONS 请求，第 13 涵盖 INVITE 请求，最后，第 15 章节涵盖 BYE 请求。

### 8.2.6 Generating the Response

当 UAS 希望对请求构建一个响应时，UAS 必须遵守一般的处理流程。这些处理流程会在下面的子章节中进行说明。另外，对于一些非常具体的响应码的处理问题，这里没有规范具体的细节，也可不做要求。

一旦所有和创建响应消息所关联的流程完成以后，UAS 负责返回到服务器事务层，这里是它收到请求的地址。

#### 8.2.6.1 Sending a Provisional Response

对生成响应来说，一个主要的不具体到某个 method 的原则是，UASs 对非 INVITE 不应该发送临时响应消息。相反，UASs 应该尽快对非 INVITE 请求生成一个最终响应消息。

当生成 100 (Trying) 响应时，重新在在请求中的任何 Timestamp 头必须拷贝到这个 100 (Trying) 响应中。如果在生成响应时有延迟，UAS 应该在 Timestamp 头中添加一个延迟数值。这个延迟数值必须包含响应发送时间值和接收请求时间值，此值以秒为单位。

#### 8.2.6.2 Headers and Tags

响应消息中的 From 必须和请求中的 From 头相同。响应中的 Call-ID 头必须和请求中的 Call-ID 相同。响应中的 CSeq 必须和请求中的 CSeq 相同。响应中的 Via 头必须和请求中的 Via 头相同，而且保持相同的顺序。

如果在请求中包含了一个 To tag 标签，响应中的 To 必须和请求中的 To 头相同。但是，如果在请求中没有包含 To 头值，在响应中回复中，To 头中的 URL 必须和请求中 To 头的 URL 相同；另外，在响应回复中，UAS 必须在 To 标签中增加一个标签（支持 100 (trying) 异常响应）。这样处理的目的是确认 UAS 正在响应处理，也可能因为这个异常响应会生成一个 dialog ID 组件。同样的标签使用在此请求的所有响应中，

包括最终响应和临时响应（除了 100 (trying 以外)）。对此标签生成的流程在中第 19.3 章定义。

### 8.2.7 Stateless UAS Behavior

无状态 UAS 是一种不保存事务状态的 UAS。它通常会转发请求，而且协议发送后会丢弃 UAS 的状态消息。如果一个无状态 UAS 收到请求重发，此 UAS 会重新生成响应，重新发送响应，就像它对第一个请求回复一样。一个 UAS 不能是无状态模式，除非这个 method 的请求处理总导致同样的响应（如果请求是确认的）。无状态注册不遵守此规则。无状态 UASs 不涉及事务层；UASs 直接收到传输层请求后，直接对传输层返回响应。

无状态 UAS 的基本功能是处理无需验证的请求，这些请求面对响应问题。如果无验证请求是通过有状态 UAS 来处理的话，那么就会导致这些无验证请求产生大量的事务状态，这些事务状态数据会导致在 UAS 侧呼叫处理速度放慢，影响 UAS 处理性能，可能立刻生成了拒绝攻击服务的条件。

更多关于拒绝攻击服务的内容，查阅第 26.1.5 章。

无状态 UAS 的最重要处理方式包括以下几个方面：

- 无状态 UAS 一定不能发送临时响应(1xx)。
- 无状态 UAS 一定不能重回响应消息。
- 无状态 UAS 必须忽略 ACK 请求。
- 无状态 UAS 必须忽略 CANCEL 请求。
- 响应中的 To 头字段值必须以一种无状态的方式生成，这种生成方式对同样的请求生成。
- 同样的标签，此方式的目的是保持标签的一致性。更多关于标签构成，参考第 19.3 章。Section 19.3

关于其他方面的处理规范，无状态 UAS 和有状态 UAS 是一样的。对每个新请求来说，UAS 可以以有状态方式或无状态方式操作。

### 8.3 Redirect Servers

在一些技术架构中，代理服务器的目的是为了降低处理负载，这些代理服务器可能是负责处理路由请求和优化信令路径的强健性，都是通过重定向方式进行转发处理。

重定向处理允许服务器端对请求在响应中推送路由消息，返回给客户端，因此，重定向会把自己踢出此环路的事务处理中，定位到请求的目标地址。当请求发起方收到了重新定位响应以后，发起方会基于收到的 URL 地址重新发送一个新的请求。通过从网络核心传输 URLs 到其网络边界，重定向允许相关网络拓展性。

重定向服务器逻辑上由一个服务器事务层和一个事务用户构成。事务用户可以访问某些定位服务（参考第 10 章节获得更多注册和定位服务详情）。定位服务实际上是一个数据库，数据库映射单个 URL 地址和一个或多个可选地址，这些地址是 URL 的目标地址。

重定向服务器自己不能发起任何属于自己的 SIP 请求。收到了一个请求以后（除了 CANCEL 请求以外），服务器可以拒绝此请求或从定位服务收集可选地址，然后返回一个最终响应 3xx。

对于格式非常规范的 CANCEL 请求，重定位服务器应该返回一个 2xx 响应。这个响应表示结束 SIP 事务处理。重定位服务器保持一个完整 SIP 事务的状态。它是客户端的责任，用来检测介于重定向服务器之间的前转环路。

当重定向服务器对请求返回一个 3xx 响应时，定向服务器会在 Contact 头中插入查询到的定位地址列表。同时，在 Contact 头中增加一个“expires”参数值，此值表示 Contact 数据中地址的生命周期。

Contact 头中包含 URLs，并且提供新的地址和用户名来尝试，或者简单提供指定的额外传输参数。301（Moved Permanently）或 302（Moved Temporarily）响应可能也提供同样地址和用户名，这个用户名是初始请求的目标地址，但是设定了额外的传输参数值，例如尝试不同的服务器或组播地址，或者传输方式的相关，例如从 UDP 传输修改为 TCP 传输，或者相反处理。

不管怎样，重定位服务器一定不能重定位一个请求到一个 URL，这个 URL 和一个在 Request-URI 的 URL 相同；相反，服务器可以代理转发这个请求到目的地 URL，或者拒绝此请求，返回一个 404 响应。

如果客户端正在使用一个 outbound proxy，并且实际上重新定位此请求，这里可能产生一个潜在的无限重定位环路。

注意，一个 Contact 头可能涉及不同的源地址，而不是初始呼叫的源地址。例如，一个 SIP 呼叫连接到 PSTN 网关，网关可能需要提供一个特别的语音说明（例如，您拨打的号码已经被修改）。

一个 Contact 响应消息头可以包含任何恰当的 URL 值，这个值表示被呼叫方已经被连接，也不仅局限于 SIP URLs 地址。例如，它可以包含电话的 URLs 地址，传真或 irc（如果有定义的话）或一个 mailto: (RFC 2368 [32]) 邮件地址。第 26.4.4 章节讨论了重定位处理中 SIPs URL 到一个 non-SIPS URI 的影响和局限性。

Contact 头中的 “expires” 参数表示 URL 的生命周期。此值以秒为单位计算。如果没有提供此参数的话，以 Expires 头中的数值决定 URL 的生命周期。如果出现异常值的话，此值应该视为等同于 3600。

这种方式提供了一种最大程度的可能，保证了和 RFC2534 向后兼容，这也支持了一个在这个头中的绝对时间值。如果收到了一个绝对时间的话，它将被视为异常值，默认的时间为 3600。

重定位服务器一定要忽略某些功能（包括无法识别的头字段格式，任何在 Require 中的未知可选标签，甚至于未知的 method 名称），和某些存在问题的重定位请求。

## 9 Canceling a Request

在前面的章节中，我们已经讨论了关于标准 UA 的处理流程，包括 method 的请求所产生的请求和处理响应流程。在这个章节，我们讨论 CANCEL，它是一个目的性比较强的 method。

CANCEL 请求和它的名字一样，它是用来取消前面由客户端发送的请求。具体来说，它要求 UAS 退出前面的请求，并且针对那个请求生成一个错误响应。如果 UAS 已经针对一个请求生成了最终响应回复，这时，CANCEL 再次针对这个请求发送取消的话是无效的。因为这个原因，大部分的 CANCEL 请求都需要服务器端耗费比较长的时间做出响应回复。因此，对于 INVITE 来说，使用 CANCEL 是最好的办法，这样就可以通过一个比较长的时间生成响应回复。在以上使用环境中，接收 CANCEL 请求的 UAS，在还没有发送最终响应时，UAS 将会处理一个“stop ringing”，然后再针对这个 INVITE 发送一个特别错误响应码（一个 487）。

代理服务器和用户代理客户端都可以创建和发送 CANCEL 响应。第 15 章节讨论了 UAC 取消 INVITE 请求的条件，第 16.10 章节讨论了代理使用 CANCEL 的细节。

有状态代理响应一个 CANCEL 请求，而不是简单前转一个从下游网元中收到的响应。因为那个原因，CANCEL 被看作是一个“hop-by-hop”请求，因为它被回复是在每个有状态代理 hop 的节点上的。

### 9.1 Client Behavior

一个 CANCEL 请求只能用来取消一个 INVITE 请求，不应该发送去取消其他的请求。

因为除了 INVITE 请求以外，其他请求会马上响应这个请求，因此，发送一个 CANCEL 请求到一个非 INVITE 请求总会创建一个互相竞争的条件。

以下流程用来构建一个 CANCEL 请求。在 CANCEL 请求中的 Request-URI, Call-ID, To, CSeq 的数字部分，和 From 头字段值必须是确认的，这些参数，包括标签是在被取消的请求中。通过客户端构建的 CANCEL 中必须只有一个 Via 头值匹配被取消的请求中的最顶部的 Via 头字段值。使用这些头字段中同样的值允许此 CANCEL 匹配它将要取消的请求。（第 9.2 章节说明了匹配是如何发生的）。但是，此 CSeq 头字段中的 method 部分必须含有一个 CANCEL 的值。通过这样的处理流程，作为一个事务，在它的具有权限范围内，CANCEL 才可以被完整确认和处理，（参考 第 17 章节）。

如果这个被正在取消的请求中包含了一个 Route 头字段值，此 CANCEL 请求必须包括那个 Route 头字段的值。

这个要求是一个必须条件，通过这样的处理要求，无状态代理才能正确地路由此 CANCEL 请求。

CANCEL 请求中一定不能包含任何 Require 或 Proxy-Require 头。

一旦 CANCEL 被创建以后，客户端应该检查针对正在被取消的请求，它这里是否收到任何响应消息（临时响应或最终响应）（因此，这里的请求是一个原始请求）。

如果客户端没有收到任何临时响应，CANCEL 一定不能被发送；相反，客户端必须在发送请求之前等待临时响应到达。如果初始的请求已经生成了一个最终响应，这个请求就不应该被发送出去了，这是一个无效操作，因为这个 CANCEL 请求已经对这个初始请求没有任何作用，这个请求已经生成了最终响应。当客户端决定发送 CANCEL 时，它针对这个 CANCEL 创建一个客户端事务，然后传输这个事务，包括这个 CANCEL 请求关联的目的地地址，端口和传输方式内容。针对这个 CANCEL 请求定义的目的地地址，端口和传输方式必须和发送初始请求的互相确认。

在接收前一个请求的响应之前，如果被允许发送 CANCEL，在初始请求之前，服务器可以接收这个 CANCEL。

注意，两种事务-相对于初始请求的事务和 CANCEL 事务，它们都是独立完成的。但是，一个正在处理取消请求的 UAC 不能依赖于针对初始请求的响应-487（请求结束），因为需要保持和 RFC 2543 的一致性，UAS 将不会生成一个这样的响应。如果在  $64*T1$  秒内（ $T1$  在第 17.1.1.1 章节中定义），针对初始请求没有收到最终响应的话，这个客户端应该可以认为这个初始事务可以被取消，并且应该销毁这个正在负责初始请求的客户端事务。

## 9.2 Server Behavior

CANCEL method 要求在服务器端的事务用户（TU）取消待处理的事务。TU 通过执行 CANCEL 请求决定事务是否取消，而且假设请求 method 是任何一种 method，但是 CANCEL 或者 ACK 可以应用事务匹配流程，具体匹配流程在第 17.2.3 章节中有所讨论。这个匹配的事务是其中一个将被取消的事务。

在服务器端关于执行 CANCEL 请求的流程完全取决于服务器类型。一个无状态代理会前转这个取消请求，一个有状态代理可能会有响应消息，生成它自己的一些 CANCEL

请求，然后 UAS 回复这些响应。查阅第 16.10 章节获得关于 CANCEL 的代理处理方式。

根据标准 UAS 的处理方式，UAS 首先处理 CANCEL 请求，具体的处理方式描述在第 8.2 章节有更多介绍。但是，因为 CANCEL 请求是 hop-by-hop 的处理方式，因此它不能被重新提交。服务器端也不会验证其在 Authorization 头中获得安全验证消息。注意，CANCEL 请求也不包含 Require 头字段。

根据以上所说的处理流程，如果 UAS 没有发现一个针对 CANCEL 的匹配事务的话，它应该对这个 CANCEL 响应一个 481 错误码（Call Leg/Transaction Does Not Exist）。如果关联初始请求的事务仍然存在的话，收到 CANCEL 请求的 UAC 的执行方式取决于这个 UAC 是否已经针对关联的初始请求已经发送了最终响应。如果已经发送了最终响应，那么这个 CANCEL 请求对初始请求处理没有任何影响，对任何会话状态没有任何影响，对初始请求生成的响应没有任何影响。如果这个 UAS 没有发送最终响应的话，这个 UAS 的处理流程则取决于初始请求的 method。进一步说，如果初始请求是一个 INVITE method，这个 UAS 应该对这个 INVITE 马上回复一个 487 错误码（Request Terminated）。CANCEL 请求不会对本规范中所定义的其他 method 所产生的事务有影响，仅对自己的 method 所产生的事务有影响。

无论初始请求的 method 是何种 method，只要 CANCEL 匹配了一个现存的事务，这个 UAS 应该自己应答这个 CANCEL，并且返回一个 200(OK) 响应。这个响应消息是通过以下处理流程来创建的，具体创建流程查阅，这里需要注意，对于 CANCEL 来说，这个响应中的这个 To 标签和初始请求中的响应中的 To 标签应该相同。这个 CANCEL 的回复响应被传递到这个服务器的事务处理进行传输。

## 10 Registrations

### 10.1 Overview

SIP 提供一种查询能力。如果一个用户想和其他用户发起一个会话的话，SIP 必须查找当前其他用户的目的地地址是否是可达状态。这个查询处理过程经常是基于用户地址信息，通过 SIP 网络的核心网元单元来完成，例如代理服务器和重定位服务器（负责接收请求）来决定发送其发送地址，然后发送这个用户到其地址。为了实现这个处理

流程，SIP 网络的网元单元会查询一个抽象服务，我们称之为定位服务。这个服务为具体的域提供地址绑定。这些地址绑定映射一个正抵达的 SIP 或者 SIPS URL 地址，例如，`sip:bob@biloxi.com`，可能会匹配一个或者多个这样相似的 URLs 地址，例如 `sip:bob@engineering.biloxi.com`。最终，代理会查询一个定位服务，定位服务会映射收到的 URL 地址到用户代理，这个用户是已经定位的，这个会话期望抵达的用户。

注册实际上是在定位服务中为指定的域创建了一个绑定关系，这个指定的域通过一个或多个 contact 地址关联了一个 address-of-record (AoR) URI 地址。因此，当这个域的代理收到一个请求，这个请求中的 Request-URI 匹配了 address-of-record (AOR) 时，代理就会转发这个请求到这个 contact 地址，这个地址已经注册到了那个 address-of-record (AOR) 上。一般情况下，只有这样的处理方式是合理的，当注册请求的 AOR 路由到那个域的时候，在域的定位服务上注册添加一个 address-of-record (AOR) 地址。大部分情况下，这里表示的意思是，注册的域将需要匹配在 AOR 地址的域。

有多种方式创建定位服务的内容。一种方式是通过管理方式创建。在以上的示例中，Bob 被看作是一名技术部门的成员之一，他有权访问公司数据库。具体操作过程中，SIP 协议对 UA 提供了一种机制，可以明确创建一种绑定关系。这种机制被称之为注册。

注册需要对指定的 UAS 类型对象发生一个 REGISTER 请求，这种类型的对象称之为注册服务或者注册服务器。注册服务工作方式类似于置于域定位服务的前端一个角色，注册服务负责读写映射 REGISTER 请求中的内容。定位服务然后通过解析出的结果查询代理服务器，代理服务器负责针对那个域的绑定结果路由那个请求。

以上所有关于注册流程的解释在 Figure 2 有完整的说明。注意，注册服务和代理服务器都是逻辑对象，它们可以在网络中部署为一个单一的逻辑对象。

为了说明其功能，这里都把它们分开说明。同时也要注意，如果注册服务和代理服务是独立的对象，为了让 UAs 那个抵达注册服务，UAs 可以通过代理服务器发送请求服务。

SIP 不能对一个特别的机制授权来支持定位服务。只有一个要求是必须满足的，那就是针对某些域的注册服务必须可以读写到定位服务中，并且针对那个域的代理或者转发服务必须能够读那些数据消息。注册服务针对同一域，它可以和一个指定的 SIP 代理服务器部署在同一服务器中。

## 10.2 Constructing the REGISTER Request

注册可以请求添加、移除和查询绑定。一个注册请求可以在一个 address-of-record 和一个或多个 contact 地址之间添加一个新的绑定。通过一个授权的、合适的第三方执行一个指定的 address-of-record 的注册。客户端也可以移除前面的绑定或通过查询来决定哪个绑定是支持的 address-of-record。

这里有一个特别说明，注册请求构建和客户发送注册请求的处理是通过基本的 UAC 处理方式来规范的，具体的规范说明，请参阅第 8.1 章节和第 17.1 章节。

一个注册请求不能创建 dialog。一个 UAC 可以在注册请求中包括一个 Route 头，这里的注册请求是基于一个前面存在的 route set 列表，具体的描述参阅第 8.1 章节。在注册请求或响应中的这个 Record-Route 头没有任何含义，如果出现的话，它必须被忽略。特别强调，这个 UAC 一定不能在注册请求的任何响应中，基于当前的或缺省的 Record-Route 创建一个新的 route set。

除了 Contact 以外，以下头字段必须包括在注册请求中。Contact 头也可以包括进去：

Request-URI: Request-URI 命名定位服务的域名（例如，“sip:chicago.com”）。

SIP URL 的“userinfo”和“@”部分一定不能出现。

To: To 头包含记录地址，其注册流程可以被创建、查询和修改。To 头和 Request-URI 是不同的，因为前者包含一个用户名称。这个记录地址必须是一个 SIP URL 或者 SIPS URL。

From: From 头包含一个人的地址记录，它负责注册。除非请求是一个第三方的注册，否则，这个值和 To 头字段值相同。

Call-ID: 所有从 UAC 发送到特定注册服务的注册使用同一 Call-ID 头字段。

如果同样的客户端使用不同的 Call-ID 值，注册服务不能检测是否是一个延迟的注册请求，这个请求可能没有按照顺序抵达注册服务。

CSeq: CSeq 值用来保证正确的注册请求顺序处理。UA 必须对每个带同样 Call-ID 的注册请求递增一个数值来保证其实现的正确性。

Contact: 注册请求可以包含一个 Contact 头，它包含零个或者多个绑定的地址。

UAs 在收到注册的最终响应之前或者没有收到前面的注册请求超时响应，它们一定不能发送新的注册（和重传相反，它包含一个新的 Contact 地址头）。

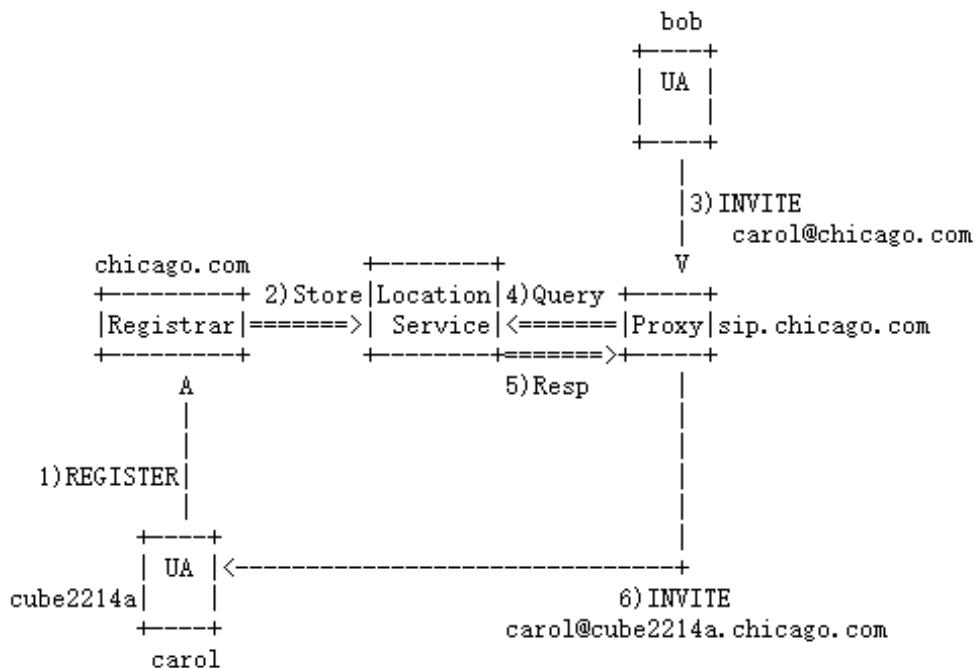


Figure 2: 注册示例

以下 Contact 头参数在注册请求中具有特别的含义：

action: 在 RFC2543 的规范中使用的参数"action" 已经停止使用。UACs 不应该再使用"action" 参数。

expires: "expires" 参数表示 UA 绑定的有效时长。参数值以秒为单位。如果没有提供这个参数的话，将使用 Expires 头的值来代替。部署时可以，如果此值大于

$2^{32}-1$  (4294967295 seconds 或 136years)可以看作是等于  $2^{32}-1$ 。如果是异常的值的话，异常的值应该被视为等于 3600。

#### 10.2.1 Adding Bindings

REGISTER 请求发送的一个注册服务中包含 contact address(es)地址，这个地址是针对 SIP 请求的 address-of-record 地址，应该被处理的转发处理的地址。address-of-record 地址包含在注册请求的 To 头中。

典型的请求中的 Contact 头值由 SIP 或者 SIPS URLs 组成，这些 URL 地址用来确认具体的 SIP 终端（例如，“sip:carol@cube2214a.chicago.com”），但是也可以使用其他的 URL 格式。举例，一个 SIP UA 可以选择使用电话号码的方式注册（使用 tel URL, RFC2806 [9]）或者邮件地址的格式注册（使用一个 mailto URL, RFC2368[32]），这些地址格式可以作为 Contacts 的 address-of-record 地址。

例如，Carol 使用这个 AOR 地址“sip:carol@chicago.com”注册服务，她的注册服务使用的是代理服务器在 chicago.com，通过这个代理服务器来路由 Carol 的 address-of-record，然后到其具体的 SIP 终端。

一旦客户端作为注册服务，并且创建了绑定关系的话，它可能在接下来的注册中包含新的绑定或者如有必要，它修改现存的绑定关系。在返回的 2xx 注册请求响应码中的 Contact 头中将包含一个完整的针对 AOR (address-of-record) 的已注册的绑定列表。

如果在注册请求中 To 头的 AOR (address-of-record) 使用的是 SIPS URL 格式，那么任何在请求中的 Contact 头也应该使用 SIPS URI 格式。当安全资源是以 contact 头出现，这个安全资源能够通过其他手段得到保证时，客户端应该只能在一个 SIPS 的 AOR 下使用非-SIPS URLs。这种方式也许是可行的，终端的 URLs 采用了其他的协议而不是 SIP 协议，或者 SIP 设备的加密协议使用的是其他协议不是 TLS 协议。

支持无需更新所有的绑定。通常来说，一个 UA 仅更新自己的 contact 地址。

##### 10.2.1.1 Setting the Expiration Interval of Contact Addresses

当客户端发送一个注册请求时，客户端可能会建议设置一个超时周期，这个超时周期设置表示客户端注册的时长有效期。(在第 10.3 章节中有描述，注册服务会基于本地策略选择一个实际时间周期)。

为了绑定注册，本规范在客户端提供了两种方式可以建议设置超时周期：通过一个 Expires 头字段或者一个 Contact 头中的"expires" 参数来设置。当在一个注册请求环境下支持了多个绑定时，后者允许基于每个绑定来设置超时周期，前者则建议对不包含 "expires" 参数的所有 Contact 头设置超时周期。

如果以上所有的设置方式没有出现在注册请求中时，那说明客户端希望服务器端来为客户端设置超时设置。

#### 10.2.1.2 Preferences among Contact Addresses

如果在注册请求中，有一个以上的 Contact 被发送出去的话，那么说明正在注册的 UA 试图使用 To 头中的 AOR (address-of-record) 地址关联 Contact 头字段中的所有 URLs 地址。这个 URL 列表的优先级根据 Contact 头中的 "q" 参数来确定。参数 "q" 表示针对这个特定的 Contact 头值来说，对比其他的绑定的 AOR 地址，"q" 设置了一个相对偏好。在第 16.6 章节中描述了代理服务器如何使用这个偏好指示。

#### 10.2.2 Removing Bindings

注册绑定是一种软状态，除非此状态被刷新，否则，它会终止注册状态，但是也可以直接解除绑定关系。客户端可以尝试去设置超时周期来解除绑定，超时周期有注册服务来设置，具体的描述在第 10.2.1 章节中介绍。一个 UA 可以通过在注册请求中的 Contact 地址中设置超时周期为 0 来立即解除绑定。在超时周期超时之前，UAs 应该支持绑定解除的机制。

REGISTER-specific Contact 头字段 "\*" 的值应用在所有的注册中，但是它一定不能被使用，除非 Expires 以设置为 "0" 的方式出现。

使用 Contact 头的 "\*" 允许一个正在注册的 UA 移除所有和 address-of-record 绑定的关系，无需知道其准确数值。

#### 10.2.3 Fetching Bindings

无论请求是否包含一个 Contact 头，对任何注册请求来说，一个成功的协议包含完整的存在的绑定列表。如果没有 Contact 头出现在当前的注册请求中，绑定列表不会修改。

#### 10.2.4 Refreshing Bindings

每个 UA 负责对已创建的绑定进行刷新。一个 UA 的绑定不应该由其他的 UA 执行绑定刷新。

从注册服务获得的 200 (OK) 响应码包含了一个 Contact 域的列表，这个列表模拟当前的绑定状态。如果 UA 通过第 19.1.4 章节中的对比规则，已经创建了这个 contact 地址，UA 会对比每个 contact 地址来对照检查。如果是这样的话，这个 UA 会根据超时参数更新超时周期设置，或者，如果缺省了这个参数的话，则根据这个 Expires 域来设置超时设置。此 UA 在超时周期时间到期之前，然后对它的每个绑定执行一个 REGISTER 请求。它可能在一个 REGISTER 请求中合并几个更新。

在单个启动循环中，一个 UA 应该使用同一 Call-ID 支持所有的注册。除非是一个转发服务，作为初始注册，注册刷新应该被发送到同样的网络地址中。

#### 10.2.5 Setting the Internal Clock

如果注册请求中的响应包含一个 Date 头，客户端可以使用这个日期头来学习当前的时间，以便使用此时间设置任何内部时钟。

#### 10.2.6 Discovering a Registrar

UA 能够使用三种方式来决定注册的地址：通过配置的方式，使用 address-of-record 地址，使用 multicast 广播方式。通过注册服务地址，UA 能够被设置。如果没有设置注册服务的地址的话，UA 应该使用 address-of-record 的主机地址作为 Request-URI，并且使用正常的 SIP 定位服务机制[4]对此地址发送请求。例如，对于这个 UA 的用户 "sip:carol@chicago.com" 来说，其注册地址请求应该发送到 "sip:chicago.com"。

最后，一个 UA 可以使用广播的方式来进行设置。广播注册被称之为 "all SIP servers" 广播地址 "sip.mcast.net" (224.0.1.75 是 IPv4 地址)。没有分配到没有非常熟知的 IPv6 广播地址；这种广播地址分配目前没有独立说明。SIP UA 可以监听那个地址，使用此

地址，并且让其他当地用户也可以获悉本 UA 的地址状态（参考 [33]）；但是，本地用户不会对此请求回复响应消息。

广播注册方式可能不适合使用在某些环境中，例如，多个业务共享同一内网。

#### **10.2.7 Transmitting a Request**

一旦 REGISTER method 构建成功以后，并且消息目的地确认以后，UACs 会根据在第 8.1.2 章中所描述的流程让传输层来进行下一步处理。

因为 REGISTER 无响应生成，如果传输层返回超时错误，UAC 不应该马上重新注册到同样的注册服务。

一个马上重新注册的处理流程也可能和超时一样。在一个合理的时间周期范围内等待网络环境的修正，降低网络负载，排查网络设备故障。这里，无特别具体的时间周期设置。

#### **10.2.8 Error Responses**

如果一个 UA 收到了 423 响应码（Interval Too Brief），它可以重新注册。但是，这里有一个必要条件-注册请求中的超时周期后才能重新注册。具体来说，注册流程中需要耗费一定的时间，使得在注册请求中所有 contact 地址中的超时周期等于或大于 423 响应中的 Min-Expires 头中的超时周期，它才可以重新注册。

### **10.3 Processing REGISTER Requests**

注册服务是一个 UAS 端，UAS 端对注册请求进行响应，并且维持一个绑定列表。在管理员域范围内，这个绑定列表对代理服务器和重转发服务器来说是可访问的。根据第 8.2 章节和第 17.2 章节的规定，注册服务处理请求，它也仅接受注册请求。注册服务一定不能生成 6xx 响应消息。

注册服务可以重新转发注册请求，这是可以接受的。一个比较常见的使用场景就是注册服务监听一个组播接口来转发组播注册请求，组播注册请求携带一个 302 (Moved Temporarily) 临时响应发送到自己的单播接口。

如果在注册请求中携带了 Record-Route 头的话，注册服务必须忽略 Record-Route 头。注册服务一定不能在针对注册请求的任何响应的消息中携带 Record-Route 头。

注册服务可能会收到这样的请求，这个请求经过了一个代理服务器节点，代理服务器把注册视为一个未知请求，代理服务器添加了一个 Record-Route 头字段值。

一个注册服务需要知道（通过配置文件）域的列表来维持绑定关系。注册服务按照收到注册请求的顺序来处理注册请求。注册请求必须是通过完全自动处理方式对请求进行处理。每个注册消息必须独立处理或者独立绑定修改。

当收到一个注册请求时，注册服务需要经过以下几个步骤：

1. 注册服务会检查这个 Request-URI 地址来决定是否它可以访问这个地址绑定在 Request-URI 所定义的域。如果不能的话，如果这个服务器可以作为一个代理服务器的话，服务器应该转发此请求到已标识地址的域，然后根据代理信息的一般流程来处理，具体的代理信息描述在第 16 章。
2. 为了保证注册服务可以支持任何必要的拓展功能，注册服务必须处理 Require header 头字段值。具体对 UAs 的 Require 头描述在第 8.2.2 章。
3. 注册服务应该 UAC 进行签权检查。针对 SIP 用户代理请求检查机制在第 22 章有介绍。注册流程绝不能覆盖 SIP 的基本请求架构。如果没有签权机制支持的话，注册服务可以提取 From 地址作为请求发起方已确认的身份。
4. 如果已签权的用户被授权修改注册来支持 address-of-record，注册服务应该可以决定此授权。例如，注册服务可能会查询授权数据库来映射用户名称和 address-of-record 列表匹配，注册服务然后决定此用户是否有权修改绑定关系。如果签权用户没有被授权修改绑定关系的话，注册服务必须返回一个 403 (Forbidden) 错误协议码，并且忽略其余的步骤。

在支持第三方注册的架构中，其中一个实体可能负责更新注册绑定，通过多个 addresses-of-record 关联注册绑定。

5. 注册服务从请求的 To 头中提取 address-of-record 地址。如果针对在 Request-URI 地址中的域来说，address-of-record 不是有效地址的话，注册服务必须发送一个 404 (Not Found) 响应码，并且忽略其余步骤。这个 URL 必须被转换成一个标准的格式。为了实现这个要求，所有的 URI 参数必须被移除（包括 user-param），并且任何转义字符必须转换成非转义格式。然后把结果设置为绑定列表的索引。
6. 注册服务检查是否请求中包含 Contact 头。如果没有的话，它会直接跳到最后步骤。如果包含一个 Contact 头字段值的话，注册服务检查这个头字段值包含一个特殊标识符 “\*” 和一个 Expires 域值。如果这个请求还有其他

Contact 域值或一个非零的超时时间设置，那么这个请求是一个无效的请求，服务器端必须返回一个 400 (Invalid Request) 无效请求的响应码，并且忽略其余步骤。如果没有其他的 Contact 地址的话，注册服务检查是否这个 Call-ID 和存储在绑定数据库中的每个绑定中的值一致。如果两个值不一致，注册服务必须移除这个绑定。如果注册服务同意的话，注册服务必须移除这个绑定，仅保留请求中 CSeq 的值高于存储的绑定值的部分绑定关系。否则，更新必须中断，这个请求失败。

7. 现在，注册服务开始依次处理 Contact 头中的每个 contact 地址。对于每个地址来说，超时周期设置通过以下步骤来决定：

- 如果 Contact 头中有一个 “expires” 参数，此参数必须被视作请求超时参数。
- 如果头中没有这样的参数，但是请求中包含了一个 Expires 头的话，此值必须被视为请求超时参数。
- 如果以上两个参数都没有，本地配置的默认值必须被视为请求超时参数。

注册服务可能选择一个超时设置，这个设置小于请求中的超时周期设置。如果并且仅如果请求的超时周期大于零并且少于一小时而且小于注册服务配置的最低设置，注册服务可以拒绝这个注册请求，并且返回 423 响应码 (Interval Too Brief)。此响应必须包含一个 Min-Expires 头，此值用来声明注册服务那个接受的最小超时周期。然后注册服务忽略其余处理步骤。

允许注册服务设置注册周期保护来维持注册服务的稳定性，注册周期保护可以应对超负荷的注册刷新同时能够维持注册状态，使得注册状态处于最新状态。注册的超时周期经常使用在服务创建中。分机随行服务就是一个比较常用的例子，用户在终端侧，终端状态短时间有效。因此，注册服务应该接受比较短的注册；如果注册周期过短的话，请求应该被拒绝，太短周期设置导致刷新过于频繁，最后降低了注册服务的性能。

对每个地址来说，注册服务使用 URL 对比规则来查询当前的绑定列表。如果绑定列表不存在的话，注册服务会直接添加列表。如果绑定列表存在的话，注册服务将检查 Call-ID 值。如果在当前的绑定中的 Call-ID 值不同于请求中的 Call-ID 值，超时时间为零并且更新后也为零，绑定必须被移除。如果两个 Call-ID 相同，注册服务就会对比 CSeq 值。如果此值大于当前绑定的值，

注册服务必须更新或者移除以上绑定。如果不能更新的话，更新操作中断，此请求失败。

这个机制保证从同一 UA 发送的那些异常的请求可以被忽略。

每个绑定记录记录了从请求中获得的 all-ID 和 CSeq 值。

如果并且仅如果所有绑定更新和其他的都是成功的，绑定更新才能记录存储（此更新对代理服务器或者转发服务器是可见状态）。如果它们其中一个失败（例如，后台数据库更新失败），请求一定是失败的，并且返回一个 500 协议错误码（服务器错误），并且所有直接绑定更新必须被移除。

8. 注册服务返回一个 200 (OK) 响应。这个响应必须包含 Contact 头字段值，这些值枚举所有当前绑定。每个 Contact 值必须支持一个“expires”参数值，这个值用来表示注册服务的超时周期。响应消息中应该包含一个 Date 头字段。

## 11 Querying for Capabilities

SIP method OPTIONS 支持一个 UA 查询其他 UA 或者代理服务器相关的支持能力。这种方式允许客户端发现 supported methods, content types, extensions, codecs, 等相关能力支持。无需对第三方“振铃”。例如，客户端在 INVITE 中插入一个 Require 头选项，客户端不能确定这个选项是否被目的地 UAS 所支持的话，客户端可以通过 OPTION 选项查询目的地 UAS 来是否支持此选项，UAS 将会返回 option 结果，通过一个 Supported 头来表示查询结果。所有的 UA 必须支持 OPTIONS method。

OPTIONS 请求目的地通过 Request-URI 来确认，此目的地确认消息可定位另外 UA 或一个 SIP 服务器端。如果此 OPTIONS 标识地址到一个代理服务器，此 Request-URI 设置为不带用户部分信息，这种处理方式和注册请求中的 Request-URI 相似。

或者，服务器收到一个 OPTIONS 请求，携带的 Max-Forwards 头字段值为零的选项，不管 Request-URI，服务器端可以响应这个请求。

这种处理方式在 HTTP/1.1 中非常普遍。这种处理方式可以作为一种“traceroute”功能来检查个体跳点服务器支持能力，跳点服务器会发送一系列的 OPTIONS 请求，并且携带递增的 Max-Forwards 头字段值。

就像一般的 UA 处理场景例子，如果 OPTIONS 没有产生响应消息，事务层会返回一个超时错误。这也表示目的地是不可达的地址，因此是一个无效的地址。

OPTIONS 请求可以作为一个已创建的 dialog 的部分消息来发送，它可以查询对端的支持能力，这个支持能力可以使用在后续的 dialog 中。

### 11.1 Construction of OPTIONS Request

OPTIONS 请求使用标准的 SIP 请求规则来构建的，具体规则参考第 8.1.1 章节。

Contact 头字段可能出现在 OPTIONS 请求中。

Accept 头应该包括在 OPTIONS 请求中，此头用来表示此消息体类型，UAC 希望在回复的响应中收到此消息体类型。通常情况下，这是一种消息格式，这种格式用来描述一个 UA 的媒体能力，例如 SDP (application/sdp)。

针对 OPTIONS 请求的响应假设限定在原始请求的 Request-URI 中。可是，仅当 OPTIONS 被作为已创建的 dialog 部分消息发送时，保证生成 OPTIONS 响应的服务器能够收到后续的 OPTIONS 请求。

OPTIONS 请求示例：

```
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 Processing of OPTIONS Request

对 OPTIONS 的 SIP 响应的构建是通过标准规则来实现的，具体的讨论在第 8.2.6 章节。响应码的选择必须和 INVITE 请求中的请求统一。如果 UAS 准备接受呼叫，那么将会

返回 200 (OK) , 如果 UAS 处于示忙状态, 那么 486(Busy Here) 将会被返回, 等等。这样就会支持 OPTIONS 请求来决定 UAS 的基本状态, 可以用来表示 UAS 是否会接受一个 INVITE 请求。

在 dialog 中收到 OPTIONS 请求, 此 OPTIONS 请求生成一个 200 (OK) 响应, 这个响应等同于一个已创建的, dialog 之外的请求, 这个请求不会对此 dialog 有任何影响。

因为代理对 OPTIONS 的处理方式和 INVITE 请求处理方式的不同, OPTIONS 有其局限性。分叉的 INVITE 会导致返回多个 200 (OK) 响应, 一个分叉的 OPTIONS 将仅导致一个单个的 200 (OK) 响应, 因为它被视为代理使用了 non-INVITE 处理方式。具体详细介绍参考第 16.7 章节。

如果 OPTIONS 的响应是由代理服务器生成的话, 代理返回一个 200 (OK) , 列出服务器的支持能力。响应消息中不包含消息体内容。

Allow, Accept, Accept-Encoding, Accept-Language, 和 Supported 头应该出现在 OPTIONS 请求的 200 (OK) 的响应中。如果响应消息是由代理生成的话, Allow 头应该被省略, 它是不规范的, 代理对 method 具有不可知性。Contact 头可能出现在 200 (OK) 的响应中, 和 3xx 响应的语义相同。在这种语义环境中, 响应消息中可能列出可达用户的一系列可选名称和 methods。告警头也可能出现在响应消息中。

消息体也可能被发送, 发送何种类型的消息取决于 OPTIONS 请求中的 Accept 头字段 (如果没有出现此 Accept 头的话, 默认发送 application/sdp) 。如果类型中包含了一种消息类型的话, 这个类型描述了媒体能力的话, UAS 应该在响应中包含一个消息体来说明媒体支持能力。关于这样的 application/sdp 构建方式, 参见[参考链接 13]。

根据在第 11.1 章节中的一个 UAS 请求生成的 OPTIONS 响应示例 :

```

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 Dialogs

对用户代理来说，一个重要的概念是 dialog。一个 dialog 表示两个用户代理之间在特定时间内保持的一个点对点关系。Dialog 用来支持用户代理之间一系列的消息，并且在它们之间提供正确的请求路由。此 dialog 通过 SIP 消息解析，表示为文本内容。第 8 章节讨论过针对外部 dialog 的请求和响应环境中对于 method 独立的 UA 场景的处理流程。此章节讨论如何使用请求和响应来构建一个 dialog，以及如何在 dialog 中发送后续的请求和响应。

在每个 UA 中，dialog 是通过 dialog ID 来确认的，它由一个 Call-ID 值，一个本地标签和远端标签构成。在此 dialog 中，每个 UA 涉及的 dialog ID 是不同的。具体来说，在一端 UA 的本地标签等同于在对端点 UA 的远端标签。此标签是一个不透明度标识符号，此标识符号支持唯一的 dialog IDs 生成。

dialog ID 也不但关联所有的响应消息，并且关联任何请求，此请求在 To 头中含有一个标签。一个消息的 dialog ID 计算规则取决于 SIP 网元是一个 UAC 还是 UAS。对于 UAC 来说，dialog ID 的 Call-ID 设置为此消息的 Call-ID，远端标签设置为消息中 To 头中的这个标签，本地标签设置为消息中 From 头中的标签（此规则适用于请求和响应中）。

对于 UAS 来说，dialog ID 中的 Call-ID 值设置为消息的 Call-ID，远端标签设置为消息中 From 头中的标签，本地标签设置为消息中 To 头中的标签。

一个 dialog 包含某些特别状态消息，这些消息用来支持在此 dialog 中的后续的消息传输。这个状态消息由这个 dialog ID，一个本地序列号（用来支持从 UA 发送到对端的请求的顺序），一个远端序列号（用来支持从远端到 UA 请求的顺序），一个本地 URL，一个远端 URL，远端目的地，一个命名为“secure”的布尔 flag，和一个路由组，此路由组是一个按续排列的 URL 列表。路由组是一个服务器列表，请求需要通过此列表服务器路径发送到对端。

Dialog 可以是一种“早期”状态，这种早期状态发生在当它创建时，它携带了一个临时响应，当收到一个 2xx 最终响应消息后，此状态会转化为一个“确认”状态。在上面的 dialog 中，对于其他响应或者完全没有收到任何响应，这个早期 dialog 就会结束。

## 12.1 Creation of a Dialog

Dialogs 是通过具体的 methods 来创建的，由一系列对请求的非失败响应生成。在此规范中，仅 2xx 和 101-199 响应携带 To 标签的，请求是 INVITE 请求的响应将会创建 dialog。对于由非最终响应创建的 dialog 来说，这种 dialog 是处于“早期”状态，称之为一个早期 dialog。拓展可定义其他含义支持创建 dialogs。第 13 章节列出了更多细节，这些细节提供了针对 INVITE method 的说明。这里，我们讨论 dialog 状态创建的流程，不依赖于这个 method。

UAs 必须对 dialog ID 组件赋值。这些组件将会在下面章节进行讨论。

### 12.1.1 UAS behavior

当 UAS 对请求返回响应时，响应消息中携带了创建 dialog 消息（例如，INVITE 响应的 2xx），UAS 必须从请求中拷贝所有 Record-Route 头值到响应消息中（包括 URIs，URI 参数和任何 Record-Route 头参数，无论这些参数对 UAS 是已知还是未知参数），而且必须保持这些参数的顺序。此 UAS 必须对响应添加一个 Contact 头，这个 Contact 头包含一个地址，UAS 将会在 dialog（包括 INVITE 中 ACK 的 2xx 响应）中的后续请

求联系此地址。一般来说，此 URL 的主机内容是此 IP 地址，或者主机的 FQDN。在 Contact 头中提供的 URI 必须是一个 SIP 或者 SIPS URL。

如果在初始化了 dialog 的请求中的 Request-URI 或者 top Record-Route 头中的值域中包含 SIPS URI，如果没有 Record-Route 头字段，如果有任何值或者 Contact 头的话，响应中的 Contact 头必须是一个 SIP URL。此 URL 应该支持一个全局范围（也就是说，在消息中，同样的 URL 可以使用在此 dialog 外部）。同样的方式，在 INVITE 中的 Contact 头字段中的 URL 使用范围也不能被局限于此 dialog 中。因此，它可以针对 UAC 的消息中，甚至于也可以使用在此 dialog 外部。

UAS 然后构建 dialog 状态。在 dialog 生命周期内，此状态必须被持续维护。

如果请求是通过 TLS 发送过来的，并且 Request-URI 包含一个 SIPS URI，“secure”设置为 TRUE。

路由组必须设置到请求的 Record-Route 头的 URL 列表中，按照顺序处理，并且保留所有的 URL 参数值。如果在请求中没有出现 Record-Route 头，路由组必须设置为空。这个路由组甚至是空的路由组将会在 dialog 的后续请求中覆盖任何已存在的路由组设置。远端目的地地址必须设置为从此请求的 Contact 头获得的 URL 地址。

远端序列号必须设置为请求中 CSeq 的序列号。本地序列号必须为空。Dialog ID 中的呼叫身份组件必须设置为请求中的 Call-ID 值。Dialog ID 中的本地标签组件必须设置为此请求的相应响应中的 TO 域中的标签值（总是要包含一个 tag 标签），dialog ID 中的远端标签组件必须设置为从请求中 From 域获得的标签值。UAS 必须准备接收一个在 From 域中无 tag 标签的请求，这样的环境中，此标签 tag 被认为是一个空值的标签。

这样的处理方式为了支持向后兼容，兼容 RFC 2543 规范，在 RFC 2543 中，tags 不是强制使用的。

远端的 URL 必须设置为从 From 获得的 URI，本地 URL 必须设置为从 To 中获得的 URL。

#### 12.1.2 UAC Behavior

当 UAC 发送了一个请求，此请求能够创建 dialog（例如发送的 INVITE），UAC 必须在请求的 Contact 头中提供一个支持全局范围的 SIP 或 SIPS URL（同样的 SIP URL 可以使用在 dialog 的外部环境中）。如果请求中含有 Request-URI 值或路由中的最顶部的 Route 头中带一个 SIPS URI，那么 Contact 头必须包含一个 SIPS URI。

当 UAC 收到了一个响应，此响应创建一个 dialog，它构建了这个 dialog 的状态。dialog 状态必须被维持在 dialog 生命周期内。

如果此请求是通过 TLS 发送，并且 Request-URI 包含一个 SIPS URI，“secure” Flag 设置为 TRUE。

路由组必须设置到响应的 Record-Route 头的 URL 列表中，按照顺序处理，并且保留所有的 URL 参数值。如果在响应中没有出现 Record-Route 头，路由组必须设置为空。这个路由组甚至是空的路由组将会在 dialog 的后续请求中覆盖任何已存在的路由组设置。远端目的地地址必须设置为从此请求的 Contact 头获得的 URL 地址。

本地序列号必须设置为请求中 CSeq 的序列号。远端序列号必须为空（当远端 UA 在 dialog 中发送一个请求时，远端序列号才能被创建）。Dialog ID 中的呼叫身份组件必须设置为请求中的 Call-ID 值。Dialog ID 中的本地标签组件必须设置为此请求的相应响应中的 From 域中的标签值（总是要包含一个 tag 标签），dialog ID 中的远端标签组件必须设置为从响应中 To 域获得的标签值。UAC 必须准备接收一个在 To 域中无 tag 标签的响应，这样的环境中，此标签 tag 被认为是一个空值的标签。

这样的处理方式为了支持向后兼容，兼容 RFC2543 规范，在 RFC2543 中，tags 不是强制使用的。

远端的 URL 必须设置为从 To 获得的 URI，本地 URL 必须设置为从 From 中获得的 URL。

## 12.2 Requests within a Dialog

一旦介于两个 UA 之间的 dialog 创建以后，如有必要，其中之一 UA 可以在 dialog 中发起新的事务。发送请求的 UA 将会在事务中充当 UAC 的角色。接收请求的 UA 将会

在事务中充当 UAS 的角色。注意，在 UA 执行事务期间，这些事务创建了不同的 dialog，它们的角色可能是不同的。

在 dialog 中，请求可以包含 Record-Route 和 Contact 头值。尽管这些请求可能修改远端目的地 URL，但是，这些请求不能导致 dialog 中路由组修改。

具体来说，一些请求中，不刷新目的地请求的这些请求不修改 dialog 的远端目的地 URL，刷新目的地请求的可以修改。在使用一个 INVITE 创建的 dialog 中，只有 re-INVITE 是一个被定义的目的地刷新请求。

参考（第 14 章节）获得更多讨论。其他拓展可以通过不同的方式在 dialog 中定义其他的目的地刷新请求。

注意，一个 ACK 不是一个目的地刷新请求。

目的地刷新请求仅更新 dialog 的远端目的地 URL，不更新从 Record-Route 构建的路由组。更新后者将会引起与 RFC2543 向后兼容的问题。

#### 12.2.1 UAC Behavior

##### 12.2.1.1 Generating the Request

Dialog 中的请求是通过此状态多种组件构成，此状态被存为 dialog 的一个部分。

请求中 To 头字段中的 URL 必须设置为远端 URL（从 dialog 状态中获得）。在请求中 To 头字段中的标签 tag 必须设置为 dialog ID 的远端标签 tag。请求的 From URL 必须设置为本地 URL 地址（dialog 状态中获得的）。请求中 From 头中的 tag 标签必须设置为 dialog ID 的本地 tag 标签。如果远端或者本地标签 tags 值为空，标签参数必须从各自的 To 或者 From 头中忽略。

初始请求中的 TO 头和 From 头使用 URL 方式以及在此后续请求中的 URL 使用方式是通过 RFC2543 的向后兼容性来完成的，RFC2543 中使用 URL 来支持 dialog 的身份确认。在本规范中，仅使用 tags 标签来确认 dialog 的身份。预计，在 mid-dialog 中的初始 To 和 From 头 URL 强制映射处理方式将会在此规范的后续重审中被废弃。

此请求的 Call-ID 必须设置为 dialog 的 Call-ID。在每个方向上（当然，除了 ACK 和 CANCEL 以外，这些请求中的号码等于此请求被确认或者取消的号码），一个 dialog 请求必须包含严格单调增加和持续的 CSeq 序列号（每次递增一个数值）。因此，如果本地序列号不是空值，本地序列号必须递增一，并且此值必须存储在 CSeq 头中。如果本地序列号为空，必须选择一个初始的值，根据第 8.1.1.5 章节中的指导来选择。CSeq 头中的 method 必须匹配请求中 method。

CSeq 使用 32 bits 长度的数字串，在一个单呼叫中，一个客户端能够生成一个请求，一秒内可能大概需要 136 年才需包含这样的数字。选择了序列号的初始值以便在同样呼叫中的后续请求将不会在包含此序列号数字。非零的值允许客户使用一个基于时间的初始序列号。例如，客户端可以选择最有效率的 31 bits 长度作为初始序列号（32bits 秒级为标准）。

UAC 使用此远端目的地和路由组来创建请求中的 Request-URI 和 Route 头字段。

如果路由组设置为空，UAC 必须把远端目的地 URL 置于 Request-URI 中。UAC 一定不能给请求添加 Route 头值。

如果路由组不为空，在路由组的第一个 URL 中包含了 lr 参数（参考第 19.1.1 章节）的话，UAC 必须把远端目的地 URL 置入到 Request-URI，并且必须包含一个 Route 头，此 Route 按续包含路由组值和所有参数。

如果路由组不为空，并且它的第一个不包含 lr 参数，此 UAC 必须把从路由组中的第一个 URL 置入到 Request-URI，去除任何 Request-URI 不支持的参数。此 UAC 必须添加一个 Route 头，此头值按续包含剩余的路由组值。然后，作为最后的值，此 UAC 必须把远端目的地 URL 置入到 Route 头值中。

例如，如果远端目的地是 sip:user@remoteua 的话，并且路由组 route set 包含：

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

此请求的构成需要以下 Request-URI 和 Route 头字段值：

METHOD sip:proxy1

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

如果路由组中的第一个 URL 不包含 lr 参数值，此 proxy 表示不理解此规范中的路由机制，使用 RFC 2543 的规范来执行，使用第一个 Route 头替换这个 Request-URI，第一个 Route 头是当进行消息转发时收到的路由头。处理消息过程中，通过（严格路由）strict router 时，把路由头中结尾的 Request-URI 保存到那个 Request-URI 中。（当此请求抵达一个松散-路由时 loose-router，它将被返回到此 Request-URI 中）。

在一个 dialog 中，UAC 应该在任何目的地刷新请求中包含一个 Contact 头，而且，除非处理过程中有一个需求需要修改它，URL 应该和此 dialog 中前面请求所使用的 URL 相同。如果“secure”flag 为 true 的话，那个 URL 必须是一个 SIPS URI 格式。就像在第 12.2.2 章节讨论的一样，在目的地刷新请求中的 Contact 头将会更新远端目的地 URL。这样就会支持 UA 提供一个新的 contact 地址，在此 dialog 生命周期内，它的地址也会发生改变。

但是，请求不是目的地刷新请求的话，此请求不会影响此 dialog 中的远端目的地 URL 地址。

其他请求构成方式在第 8.1.1 章节中有更多介绍。

一旦请求构建完成后，服务器的地址会被处理，使用同样的外部 dialog 请求处理方式发生此请求（第 8.1.2 章）。

在第 8.1.2 章中规定的处理流程中，如果没有 Route 头的话，此流程将会导致请求被发送到一个地址，这个地址标识在 topmost Route 头中或者 Request-URI 中。受限于某些限制，它们允许此请求被发送到其他可选目的地地址（例如，默认的 outbound proxy 没有出现在路由组中）。

### 12.2.1.2 Processing the Responses

UAC 将会从事务层收到一个请求的响应消息。如果客户端事务返回一个超时的话，此错误看作是一个 408（请求超时）响应。

UAC 处理方式需要注意。如果此 UAC 收到了一个针对一个请求返回的 3xx 响应，此请求是在一个 dialog 中发送到请求，那么，此 UAC 处理方式和 dialog 外部发送到请求处理方式相同。具体的讨论参考第 8.1.3.4 章节。

注意，但是，当 UAC 尝试可选地址时，它仍然使用路由组支持此 dialog 创建请求的 Route 头。

当 UAC 收到一个针对目的地刷新请求的 2xx 响应时，如果出现了远端目的地 URL 的话，UAC 必须用此响应中的 Contact 头的 URL 替换 dialog 中的远端目的地（remote target URI）。

如果在 dialog 中的请求响应是一个 481 错误（Call/Transaction Does Not Exist）或者一个 408（Request Timeout）的话，UAC 应该结束此 dialog。UAC 也应该结束完全无请求响应回复的 dialog（此客户端事务层将会通知 TU 超时的结果）。

对于以 INVITE 发起的 dialogs，通过发送一个 BYE 消息来结束 dialog。

#### 12.2.2 UAS Behavior

在 dialog 中发送的请求就像其他请求一样，它是一个原子的核心请求。如果 UAS 接受了一个特别的请求，所有和它关联的状态修改都要被执行。如果此请求被拒绝的话，不执行任何状态修改的流程。

注意，一些请求，例如，INVITE 请求，它们会影响一些状态改变。

UAS 将会从事务层收到此请求。如果此请求在 To 头中包含一个 tag 标签，UAS core 会处理 dialog 身份确认，和此请求保持一致，然后通过现有的 dialogs 和此 dialog 进行对比。如果对比匹配的话，那将确认这是一个 mid-dialog 请求。在此情况下，UAS 首先使用同样的针对外部 dialog 请求的处理规则来执行处理流程，具体的讨论在第 8.2 章节中。

如果此请求的 To 头中包含了一个并且 tag，但是，dialog 身份确认不能匹配当前存在的任何 dialog 时，UAS 可能已经出现系统崩溃然后重新启动，或者维持现有状态，这里，此 UAS 可能已经收到一个请求，这个请求可能是支持了不同的（失败的）UAS（这些 UAS 有能力构建 To 标签，因此那个 UAS 可以通过此 tag 标签来确认请求，这些标签是用来支持 UAS 提供失败恢复状态的标签）。另外一种可能是，收到的请求可能已经执行了错误路由。基于 To tag 标签，UAS 可能接受或者拒绝此请求。接受此请求可以为 To tag 标签提供健壮性，因此 dialog 可以维持持续性，甚至 UAS 崩溃。希望支持这种能力的 UA 必须考虑前面的这一点。在处理过程中可能出现一些问题，例

如 UA 重启时选择了严格递增 CSeq 序列号，重构路由组和接受超出范围的 RTP 时间戳和序列号码。

如果 UAS 想解决此请求的，因为 UAS 不想重新创建此 dialog，UAS 必须针对其请求回复一个 481 响应 (Call/Transaction Does Not Exist) 状态响应码，并且发送此响应到服务器事务层。

如果此请求没有通过任何方式修改 dialog 状态的话，此请求可能就会在 dialog 中收到（例如，OPTIONS 请求）。对它们的处理方式就像外部 dialog 中收到的请求一样。

如果远端序列号为空，此序列号必须设置为在请求中 CSeq 头的序列号值。如果远端序列号不为空，但是请求中的序列号值低于远端序列号的话，此请求已经排序异常，必须返回一个 500 (Server Internal Error) 错误响应码。如果远端序列号不为空，并且请求的序列号大于远端的序列号值，此请求是按续处理的。本地 CSeq 序列号大于远端 CSeq 序列号一位数是可能存在的。这本身不是一个错误状态，UAS 应该准备好接收处理类似的请求，这样的请求中携带的 CSeq 值会高于前一个接收的请求的 CSeq 值。

如果代理对一个由 UAC 生成的请求进行验证的话，此 UAC 需要重新提交携带安全消息的请求。重新提交的请求将会生成一个新的 CSeq 序列号。因为 UAS 从来没有看到过第一个请求，因此，UA 会在 CSeq 序列号位置提示中断，这样的中断不代表任何错误状态消息。

当 UAS 收到一个目的地刷新请求时，它必须使用此请求中 Contact 头中的 URL 替换 dialog 中的远端目的地 URL 地址（如果存在的话）。

### 12.3 Termination of a Dialog

Dialog 的结束和此 method 的使用环境是独立的，如果一个外部 dialog 请求生成了一个非 2xx 最终响应，任何通过以前请求响应创建的历史 dialogs 将会结束。结束确认 dialogs 的机制是依赖于具体的 method。在此规范中，BYE method 将会结束和它关联的会话和此 dialog。具体细节参考第 15 章节的内容。

# 13 Initiating a Session

## 13.1 Overview

当一个用户代理客户端期望发起一个会话（例如，语音，视频或者游戏）时，它会发送一个 INVITE 请求。这个 INVITE 请求将会询问服务器端来创建一个会话。这个请求可能会通过代理进行转发，最后抵达一个或者多个 UAS 端，此 UAS 端是最终接受此请求的服务器端。这些 UAS 端将需要定期查询此用户端，确认是否接受此邀请请求。

一定时间后，那些 UAS 端返回一个 2xx 响应表示能够接受此邀请（表示此会话已被创建）。如果此邀请没有被接受的话，UAS 将会对用户代理发送一个 3xx, 4xx, 5xx 或者 6xx 响应，状态错误码发送取决于被拒绝的理由。在 UAS 发送最终响应之前，UAS 也能发送一个临时响应（1xx），通知对方正在联系被呼叫方来处理 UAC 流程。

在收到一个或多个临时响应后，UAC 将会获得一个或者多个 2xx 响应，或者一个非-2xx 最终响应。因为需要耗费一定的时间等待接收邀请的最终响应消息，邀请（Invite）事务的可靠性机制处理方式和其他的请求（OPTIONS）有所不同。一旦 UAC 收到一个最终响应，此 UAC 需要对每个它收到的最终响应发送一个 ACK 确认消息。发送 ACK 的流程处理取决于响应的类型。对于介于 300 和 699 之间的最终响应，ACK 的处理是在事务层来完成对，并且需要遵从一系列规则（具体规则参考第 17 章节内容）。对于 2xx 响应，ACK 是由 UAC core 来生成。

由 INVITE 收到的 2xx 响应创建一个会话，同时它也在 UA 之间创建了一个 dialog。一个 UA 是发起此 INVITE 请求的，一个 UA 是生成 2xx 响应的。因此，当从不同远端 UA 收到多个 2xx 响应（因为 INVITE 分叉），每个 2xx 创建一个不同的 dialog。所有这些 dialog 都属于同一呼叫。

此章节提供了一个使用 INVITE 创建会话的细节。支持 INVITE 的 UA 也必须支持 ACK, CANCEL 和 BYE。

## 13.2 UAC Processing

### 13.2.1 Creating the Initial INVITE

因为初始请求表示一个 dialog 外部请求，它的构建过程遵从第 8.1.1 章节的处理流程。对于此 INVITE 具体情况来说，需要增加额外的步骤。

任何 Allow 头字段（第 20.5 章节）应该出现在此 INVITE 中。它表示在一个 dialog 生命周期内，UA 发送此 INVITE 时援引了何种 methods。例如，在 dialog 中，UA 接收 INFO 请求的能力应该[34]包含一个 Allow 头，此 Allow 头列出此 NFO method。

任何 Supported 头字段（第 20.37 章节）应该出现在此 INVITE 中。它枚举了 UAC 可以理解的所有拓展。

任何 Accept 头字段（第 20.1 章节）也可能出现在 INVITE 中。它表示 UA 可以接受何种 Content-Types，接受 Content-Types 的方向不仅是 UA 接收响应侧，而且还在由此 INVITE 创建的 dialog 中的后续请求侧。Accept 头字段非常有用，它原来表示各种会话描述格式的支持能力。

UAC 可以增加一个 Expires 头字段（第 20.19 章节）来限制请求的有效性。如果在 Expires 头中的时间设置超时，没有收到此 INVITE 的最后应答响应，UAC core 应该对此 INVITE 生成一个 CANCEL 请求，参考第 9 章节。

UAC 也可以发现其他有用的头字段添加到头字段中，其中包括 Subject（第 20.36 章节），Organization（第 20.25 章节）和 User-Agent（第 20.41 章节）头字段。所有这些头字段包含和 INVITE 相关的信息。

UAC 可以对此 INVITE 添加一个消息体。第 8.1.1.10 章节具体描述了如何构建头字段--Content-Type，和需要说明消息体内容。

针对包含会话描述的消息体，规范有一些特别的规则，消息体相应的 Content-Disposition 是一个“会话”。SIP 使用 offer/answer 模式支持 UA 发送一个会话描述，称之为 offer 端，offer 端包含了此会话的推荐的描述。此 offer 端指示它所期望的通信方式（语音，视频或者游戏），通信方式所支持的参数（例如编码类型）和从应答方接收媒体的地址。对端 UA 则携带另外一个会话描述做出响应，称之为 answer，它指示何种媒体方式可以被接受，所支持媒体方式的参数和从提供方接收媒体的地址。offer/answer 交互是在 dialog 的 context 中进行，因此，如果 SIP INVITE 导致了多个 dialog 的话，每个 dialog 就是一个独立的 offer/answer 交互。当发生 offer 和 answer 时，此 offer/answer 描述规定了限制条件（例如，当一个 offer 正在处理时，用户不能创建一个新的 offer）。通过这样的处理方式，在 SIP 消息中的 offer 和 answer 双方能够体现这样的限制措施。在此规范中，offers 和 answers 仅能够出现在 INVITE 请求，

响应和 ACK 中。这里，关于 offers 和 answers 模式有进一步的限定。对于初始 INVITE 事务，规则包括：

- 初始 offer 必须是在一个 INVITE 中，或没有在 INVITE 中，如果没有的话，初始 offer 是在从 UAS 返回 UAC 的第一个可靠的非失败消息中。在此规范中是 2xx 最终响应。
- 如果初始响应在 INVITE 中，应答必须是在从 UAS 返回到 UAC 的一个可靠非失败消息中，UAC 和那个 INVITE 有关联关系。对于此规范来说，仅表示为此 INVITE 的 2xx 最终响应。同样相似的应答也可以被置于任何临时响应中，这些临时响应是发送到前面应答方的响应。UAC 必须把它收到的第一个会话描述视为此应答方，并且必须忽略任何在针对此初始 INVITE 的后续响应中的会话描述。
- 如果此初始 offer 是在从 UAS 返回到 UAC 的第一个可靠的非失败消息中，从 answer 必须是对此消息的确认消息中（在此规范中，ACK 支持的 2xx 响应中）。
- 对第一个 offer 来说，如果 UAC 已发送或已收到一个应答后，此 UAC 可能在请求中生成后续的 offer，这些请求的处理是基于那个 method 指定的规则来进行的，但是，此处理方式仅支持两种状态，第一种是如果 UAC 已经收到了前面 offer 的应答后的状态，和如果 UAC 没有获得应答，它不发送任何 offer 的状态。
- 对初始 offer 来说，一旦 UAS 发出或收到了应答，此 UAS 一定不能在对初始请求的响应中生成后续的 offer。这表示，直到此初始事务完成前，基于此规范的 UAS 永远不能单独生成后续的 offer。

具体来说，以上规则中，对此规范单独指定了两个交互来支持 UA 的法则-offer 是在 INVITE 中， answer 是在 2xx 响应中（也可能在 1xx 响应中）或者 offer 是在 2xx 响应中， answer 是在 ACK 中。所有支持 INVITE 的代理必须支持它们的这两个交互。

所有用户代理必须支持 Session Description Protocol (SDP) (RFC 2327 [参考链接 1])作为一种手段来描述会话，它们构建 offer 和 answer 的方式必须遵从此流程，这个流程在定义在[13]章节。

此 offer-answer 模式的限制所描述的仅应用在消息体中，此消息体的 Content-Disposition 头值是一个“会话”。因此，INVITE 和 ACK 中包含一个消息体是可能的，例如，一个 INVITE 中包含一张图片 (Content-Disposition: render)，并且从 ACK 是一个会话描述 (Content-Disposition: session)。

如果此 Content-Disposition 头字段值丢失的话，Content-Type application/sdp 的消息体会说明这个 disposition "session"，其他的内容类型会说明"render"值。

一旦 INVITE 创建后，UAC 遵从一个处理机制，这个机制在第八章的 dialog 外部发送请求的章节中定义。这样会导致一个客户端的事务构建，此事务构建最终发送此请求并且对 UAC 返回响应。

#### 13.2.2 Processing INVITE Responses

一旦 INVITE 传递到 INVITE 的客户端事务，UAC 将会等待此 INVITE 的响应。如果此 INVITE 的客户端事务返回一个超时而不是一个响应，此响应是 TU 如果已收到了一个 408 (Request Timeout) 响应的话，它充当一个响应，此响应就像在第 8.1.3 章节讨论的一样。

##### 13.2.2.1 1xx Responses

在收到一个或者多个最终响应之前，可能抵达零，一个或者多个临时响应。对一个 INVITE 请求来说，临时响应能够创建“early dialogs”。如果在临时响应的 TO 中有一个 tag 标签的话，并且如果此响应的 dialog ID 不能匹配已存在的 dialog，将创建一个新的 dialog，创建流程在第 12.1.2 章节中定义。

仅需要 early dialog 的状态是，初始 INVITE 事务完成之前，如果 UAC 需要在其 dialog 中对其 peer 发送一个请求，UAC 才需要 early dialog。只要此 dialog 在早期状态的话，临时响应中出现 header 头字段是可以接受的，例如，在临时响应中的 Allow 头包含 methods，当处理状态在早期状态时，这些 methods 可以用在此 dialog 中。

##### 13.2.2.2 3xx Responses

一个 3xx 响应可以包含一个或者多个 Contact 头，这些 contact 头提供新的地址，这些地址是被呼叫方可达地址。根据 3xx 状态码的不同（第 21.3 章节），UAC 可能选择去尝试这些不同的新地址。

##### 13.2.2.3 4xx, 5xx and 6xx Responses

针对 INVITE 消息，可能收到一个单个非-2xx 最终响应。4xx, 5xx 和 6xx 响应中可能包含一个 Contact 头字段值，此值表示一个位置，此处发现关于错误的其他信息。后续最终响应（在错误条件下仅抵达的）必须被忽略。

在收到非-2xx 最终响应的回复以后，所有早期 dialogs 将会结束。

已经收到非-2xx 最终响应后，UAC core 认为 INVITE 事务已完成。此 INVITE 客户端事务处理会针对此响应来处理 ACKs 生成（第 17 章节）。

#### 13.2.2.4 2xx Responses

因为分叉代理的原因，一个单 INVITE 请求的多个 2xx 响应会抵达 UAC 端。每个响应通过 To 头中的标签参数加以区别，每个响应代表一个不同的 dialog，这些 dialog 具有不同的 dialog identifier 身份确认消息。

如果在 2xx 响应中断 dialog identifier 匹配了当前存在的 dialog 的 dialog identifier，此 dialog 必须被迁移到"confirmed" 确认状态，并且，在此 dialog 的路由组必须被重新计算，其算法基于 2xx 响应，按照的第 12.2.1.2 章节流程来处理。否则，在"confirmed" 状态中的一个新 dialog 必须重构，构建流程按照第 12.1.2 章节处理。

注意，只有一小部分的状态需要重新计算，这部分状态是 route set。在此 dialog 中其他状态部分例如最高序列号的部分（远端或者本地的）无需重新计算。

route set 部分的计算仅为了支持向后兼容。RFC 2543 没有在 1xx 响应中强制执行 Record-Route 头的检查，只有在 2xx 支持。, 因为，可能在早期的 dialog 中，mid-dialog 请求已经被发送出去，并且可能执行了其他的修改，例如修改了序列号，因此，我们不能更新整个 dialog 的状态。

UAC core 必须对每个从事务层收到的 2xx 生成一个 ACK 请求。除了认证需要的 CSeq 和头字段，Ack 请求的头字段构建方式和任何在 dialog 中的一样（第 12 章）。CSeq 头的序列号必须和 INVITE 确认的一样，但是 CSeq method 必须是 ACK。就像 INVITE 一样，ACK 必须包含同样的安全信息。如果 2xx 包含一个 offer 消息（基于上面的规则），此 ACK 必须在其消息体中传递一个 answer 消息。如果在 2xx 响应中的 offer 没有被接受，UAC core 必须在 ACK 中生成一个有效的 answer 消息，并且马上发送一个 BYE。

一旦 ACK 构建好以后，使用[参考链接 4]中的处理流程来决定目的地地址，端口和传输方式。但是，为了传输流程，请求将会直接传递到传输层，而不是传输到客户事务层。这是因为 UAC core 处理 ACK 的重传，不是事务层处理。每次 ACK 抵达触发 2xx 最终响应的重新传输 ACK 必须被传递到客户传输层。

第一个 2xx 响应收到以后，UAC core 认为 INVITE 事务完成了  $64*T1$  秒的定时。在这一时间点，所有没有切换到已创建状态的早期 dialog 都将结束。一旦认为 INVITE 事务被 UAC core 完成，则无更多新的 2xx 响应会到达。

如果，对 INVITE 确认了任何 2xx 响应，UAC core 不想继续那个 dialog，然后，UAC 必须发送一个 BYE 请求结束此 dialog，处理方式在第 15 章讨论。

### 13.3 UAS Processing

#### 13.3.1 Processing of the INVITE

UAS core 将会从事务层收到 INVITE 请求。它首先执行的是请求处理流程，参考第 8.2 章，这个流程适用于内部请求和 dialog 的外部请求。

假设那些流程完成执行步骤没有生成响应，此 UAS core 还会执行其他的步骤：

- 如果此请求是一个 INVITE 请求，这个 INVITE 请求包含一个 Expires header 的话，UAS core 设置一个定时器，并且定时器时长表示了 header 的值。当触发定时器以后，这个请求会被认为超时。如果这个请求超时是在 UAS 已生成最终响应之前，487 (Request Terminated) 响应应该被生成。
- 如果此请求是一个 mid-dialog 请求的话，需要根据各自 method 独立处理流程描述的来处理，具体的处理流程首先采用第 12.2.2 章节它也可能修改会话，第 14 章节提供了更多细节。
- 如果请求在 To header 中有一个 tag 标签，但是 dialog identifier 不能匹配任何已存 dialogs，UAS 可能已经崩溃，已重启，或者不同的 UAS 可能收到了请求（可能是失败的 UAS）。在这种情况下，第 12.2.2 章节提供了一个指南确保获得一个稳定的处理流程。

从这里开始的流程和后续的流程假设此 INVITE 是一个 dialog 外部的请求，并且其目的是为了创建一个新会话。

此 INVITE 可能包含一个会话描述，这种情况是 UAS 出现在一个这个会话的 offer 消息中。这是非常可能的，用户已经是那个会话中的一个参与方，甚至于此 INVITE 是一个 dialog 外部的 INVITE。这种情况可以发生在多播会议，用户被其他参与方邀请加入会议中。如果需要的话，此 UAS 可以在会话描述中使用 identifiers 来检测重复身份。

例如，SDP 包含一个会话 ID 和在 origin (o) 行的版本号。如果此用户已经是会话中的一员，并且包含了在会话描述的会话参数没有任何改变，UAS 可以接受这个 INVITE 请求（发送一个 2xx 响应，无需提示此用户）。

如果此 INVITE 没有包含任何的会话描述，UAS 最终被邀请加入一个会话中，并且 UAC 已经被要求由 UAS 提供会话 offer 消息。此 INVITE 必须在它的第一个非失败可靠性消息中提供此 offer 返回到此 UAC 端。在此规范中，就是一个对此 INVITE 的 2xx 响应消息。

UAS 能够指示处理状态，接受，转发和拒绝此邀请的能力。在这些所有场景中，UAS 通过流程来构建一个响应消息，流程的处理方式参考第 8.2.6 章节。

#### 13.3.1.1 Progress

如果 UAS 不能马上应答请求的话，它可以选择对 UAC 指示一些呼叫状态（例如，指示电话正在振铃）。这个指示状态通过发送一些临时响应来实现，临时响应取值介于 101 和 199 之间。这些临时响应创建早期的 dialogs，因此需要遵从第 12.1.1 章节的内容，和第 8.2.6 章节的内容。一个 UAS 只要它愿意，它可以发送多个临时响应。多个临时响应中的每个临时响应必须表示同一 dialog ID。不过，这些响应不是通过可靠传输实现的。

如果 UAS 期望一个延迟时间来应答这个 INVITE，它将需要请求一个拓展时间防止代理取消这个事务。当在一个事务中，响应之间的时间间隔超过三分钟，代理有取消事务的选择权。为了防止代理权限事务，UAS 必须每分钟发送一个非 100 的临时响应来应对丢失临时响应的可能性。

当用户被执行了呼叫停靠或者配合 PSTN 网络工作时（例如没有应答呼叫，进入了 IVR 流程），INVITE 事务可以在这个拓展的时间段继续执行下去。

#### 13.3.1.2 The INVITE is Redirected

如果 UAS 决定转发此呼叫时，需要发送一个 3xx 响应。一个 300 (Multiple Choices), 301 (Moved Permanently) 或 302 (Moved Temporarily) 应该包含一个 Contact 头，这个 contact 头包含一个或者多个新地址的 URLs，这些新地址将会在转发呼叫中使用。这个响应被传输到 INVITE 服务器事务层，事务层处理它的重传。

#### 13.3.1.3 The INVITE is Rejected

经常看到的场景是系统呼叫方不能启动或不能再进行额外的呼叫。在这种状态下，应该符合一个 486 (Busy Here)。如果 UAS 知道，无任何系统端资源来接受呼叫时，一个 600 (Busy Everywhere) 响应应该被返回。但是，好像 UAS 知道这种情况，因此通常不会使用响应。响应被传递到 INVITE 服务器端事务层，事务层将处理重传流程。

UAS 应该返回一个 488 (Not Acceptable Here) 响应，实际上，UAS 通过一个拒绝的 offer 来实现，这个 offer 包含在 INVITE 中。这样的响应应该包括一个告警头字段值，这个值解释 offer 被拒绝的原因。

#### 13.3.1.4 The INVITE is Accepted

UAS core 产生一个 2xx 响应消息。这个响应创建了一个 dialog，因此按照第 12.1.1 章的处理流程来执行，另外还有第 8.2.6 章流程。

对 INVITE 的一个 2xx 响应应该包含 Allow 头和 Supported 头，并且可能包含 Accept 头。包括这些头的话，无需侦测 UAS 功能，在呼叫期间允许 UAC 决定 UAS 所提供的功能和其拓展。

如果 INVITE 请求包含一个 offer 消息，并且 UAS 还没有发送 answer 消息的话，2xx 响应必须包含一个 answer 消息。如果此 INVITE 没有包含 offer 消息的话，如果 UAS 还没有发送 offer 的话，2xx 必须包含一个 offer 消息。

一旦响应构建后，响应会被传递到 INVITE 服务器事务层。注意，INVITE 服务器事务收到最终响应，传递到传输层后，INVITE 服务器事务将被销毁。因此，直到 ACK 抵

达之前，事务层定期直接发送响应消息到传输层是非常必要的。2xx 响应被传递到传输层，同时携带一个定时器，定时器从 T1 秒开始计算，然后针对每个重传定时器时间翻倍计算，直到 T1 定时器时间设置到了 T2 秒。（T1 和 T2 定义在第 17 章）。当收到针对此响应的 ACK 请求后，响应重传退出。这里，如何退出不取决于发送响应所使用的传输协议。

因为 2xx 通过端对端被重传，因此，在 UAS 和 UAC 之间可能有多个 hops，这些 hops 支持的 UDP。为了保证经过这些 hops 的可靠传输，尽管在 UAS 的传输是可靠性，响应消息也需要定期重传。

如果在  $64 \times T_1$  秒内，服务器端没有收到 ACK，服务器重发这个 2xx 响应，此 dialog 是被确认的，但是，此会话应该结束。结束会话使用 BYE 请求消息，具体描述在第 15 章节。

## 14 Modifying an Existing Session

一个成功的请求(参考第 13 章)不仅创建了两用户之间的 dialog，并且创建还创建了一个基于 offer-answer 模式的会话。第 12 章解释了如何使用目标刷新请求修改现存 dialog (例如，修改 dialog 的远端目标 URL)。这个部分介绍如何修改实际会话。这样的会话修改会涉及到地址修改和端口修改，增加媒体，删除媒体等细节。会话修改的方式是在同一 dialog (已创建了会话) 中发送一个新的 INVITE 请求来实现。在现存 dialog 中发送一个 INVITE 请求，称之为 re-INVITE。

注意，一个单个的 re-INVITE 能够同时修改此 dialog 本身和它的会话参数。

呼叫方和被呼叫方都可以修改现存会话。

媒体失败检测处理中，UA 的表现是本地策略负责的事情。但是，自动生成 re-INVITE 或 BYE 是不推荐的。不推荐的原因是为了避免洪水攻击。因为，当有网络流量时，这样处理方式可能导致洪水攻击从而造成网络拥塞。在如何场景中，如果消息被自动发送的话，这些消息应该在一个设定超时后发送。

注意，关于以上段落中是针对自动生成 BYEs 和 re-INVITEs 的。像正常处理流程一样，如果因为媒体失败用户挂机，UA 将会发送一个 BYE 请求消息。

## 14.1 UAC Behavior

同样的在 INVITE 中的 offer-answer 交互模式（第 13.2.1 章节）也应用在了 re-INVITEs 的处理流程中。这样处理的结果是，对于想增加媒体流的 UAC 来说，此 UAC 将要创建一个新的 offer，这个 offer 包含这个新的媒体流，通过一个 INVITE 请求发送此媒体流到对端。需要注意的是，会话全描述不仅仅被修改，它也被发送。在各种不同的场景中，这样的处理方式支持了无状态会话的处理流程，并且也支持了逃生和重新恢复功能。当然，一个 UAC 可以在不携带会话描述时发送 re-INVITE 请求，这种情况下，对 re-INVITE 的第一个可靠非失败响应来说，这个响应将会包含此 offer 信息（在此规范中，这个响应消息就是 2xx 响应）。

如果会话模式格式有版本号的话，发起方 offerer 应该指示此会话描述版本已经被修改。

在现存的 dialog 中，To, From, Call-ID, CSeq 和 re-INVITE 的 Request-URI 的设置规则和正常请求的所使用的规则一样，具体规则设置参考第 12 章。

UAC 可以选择不添加 Alert-Info 头或消息体，此消息体通过 Content-Disposition 来提示 re-INVITE，因为 UASs 不特意在接收 re-INVITE 时提醒用户。

不像 INVITE 请求，它可以进行分叉处理，一个 re-INVITE 从来不能进行分叉处理，因此，永远只能生成一个单个最终响应。re-INVITE 永远不会进行分叉处理的原因是使用 Request-URI 确定了目标，这个目标就是一个 UA 实例，它已经和 dialog 工作，而不是使用 address-of-record 来确定用户。

注意，当其他 INVITE 事务在同一方向正在处理时，UAC 一定不能在同一个 dialog 中发起一个新的 INVITE 事务。具体来说：

1. 如果已经存在一个出局的 INVITE 客户端事务，发起一个新的 INVITE 之前，TU 必须等待直到这个事务完成或结束状态。
2. 如果已经存在一个出局 INVITE 服务器端事务，发起一个新的 INVITE 之前，TU 必须等待直到事务完成或结束状态。

但是，一个事务正在处理时，一个 UA 可以发起一个正常的事务。当正常事务正在处理时，UA 也可以发起一个 INVITE。

如果一个 UA 收到了一个针对 re-INVITE 的非-2xx 的最终响应，这个会话参数必须保持原生状态，这些参数不能被修改，就像没有发送 re-INVITE 一样。注意，在章节 12.2.1.2 中说明的一样，针对 re-INVITE，如果收到的这个非-2xx 最终响应是一个 481 (Call/Transaction Does Not Exist) 或一个 408 (Request Timeout) 或者完全无响应 (INVITE 客户端事务返回超时)，UAC 将会结束这个 dialog。

针对一个 re-INVITE，如果一个 UAC 收到一个 491 响应的话，它应该启动一个定时器，按照以下规则启用定时器 T：

1. 如果这个 UAC 是 Call-ID 的 dialog ID 的所有者（表示它生成的值），T 可以在 10 毫秒单元内任意选择一个值，这个时间值介于 2.1 和 4 秒之间。
2. 如果这个 UAC 不是 Call-ID 的 dialog ID 的所有者（表示不是它生成的），T 值可以在 10 毫秒单元任意取值，取值范围介于 0 到 2 秒钟内。

当触发了这个定时器以后，如果 UAC 仍然期望再进行会话修改的话，此 UAC 应该尝试再发起这个 re-INVITE。例如，如果呼叫已经通过 BYE 消息被挂机，re-INVITE 就不会发生。

重传 re-INVITE 的规则和针对 re-INVITE 的 2xx 响应的 ACK 生成的规则和初始 INVITE 的相同(第 13.2.1 章)。

## 14.2 UAS Behavior

第 13.3.1 描述了处理流程，包括针对进入初始的 INVITES 区分和针对一个现存 dialog 中 re-INVITE 处理。

如果一个 UAS 收到了第二个 INVITE 请求，它发送针对第一个 INVITE 请求的最终响应之前，并且第一个 INVITE 在同一 dialog 中携带一个比较低的 CSeq 序列号的话，此 UAS 必须对第二个 INVITE 返回一个 500 (Server Internal Error) 响应，必须在此响应中包含一个 Retry-After 头，这个头携带的值是一个任意选择的值，取值范围在 0 和 10 秒之间。

如果一个 UAS 在一个 dialog 中收到了一个 INVITE 请求，同时此 UAS 在此 dialog 中发送了一个 INVITE 请求，这个 INVITE 是在处理状态的话，此 UAS 必须对收到的 INVITE 请求返回一个 491 (Request Pending) 响应。

如果 UA 针对现存的 dialog 收到一个 re-INVITE, 它必须在会话描述中检查是否有任何版本确认, 或者如果没有版本确认的话, 检查会话描述的内容是否被修改。如果会话描述被修改, 可能经过询问用户确认以后, UAS 必须相应调整会话参数。

会话描述版本可以在会议中用来调解新入会成员的能力, 增加或移除媒体, 或者从单播修改为多播会议状态。

如果新的会话描述没有被接受, UAS 能够拒绝此会话描述, 对这个 re-INVITE 返回一个 488 (Not Acceptable Here) 响应。这个响应中应该包含一个告警头字段。

如果 UAS 生成了一个 2xx 响应, 并且从来没有收到一个 ACK, 此 UAS 应该生成一个 BYE 消息来结束这个 dialog。

针对一个 re-INVITE, UAS 可以选择不生成 180 (Ringing) 响应, 因为 UAS 通常不会对此用户返回此消息。因为一些其他原因, 针对此 re-invite, UASs 可以选择不使用 Alert-Info header field 或者消息体, 在响应中断消息体携带 Content-Disposition "alert"。

如果 UAS 正在执行一个新的呼叫, 受到发送的 offer 的限定, 这个 offer 更新现存会话, 这种场景参考 SDP 示例[13], 因此, 这种 UAS, 具体来说, 在 2xx 响应中提供 offer 的 UAS (因为此 INVITE 不确认此 offer) 应该构建此 offer 消息, 构建规则应该遵从发送 offer 的限定规则。具体来说, 其含义是 offer 中应该尽可能多包括多个媒体格式和媒体类型, 这些媒体格式和媒体类型是 UA 将要支持的格式和类型。此 UAS 必须确保在会话描述中重叠覆盖全面会话描述中, 对端 peer 所支持的媒体格式, 传输或者支持的参数。这样可以避免对端 peer 拒绝会话描述。但是, 如果这些新会话或者参数对 UAC 是不能接受的话, UAC 应该生成一个 answer 消息, 携带一个有效的会话描述, 然后发送一个 BYE 消息结束此会话。

## 15 Terminating a Session

这个部分描述如何结束由 SIP 创建的会话流程。会话状态和 dialog 状态有非常紧密的关系。当通过 INVITE 创建了一个会话以后, 每个从不同 UAS 创建 1xx 或 2xx 响应, 并且, 如果那个响应完成了 offer/answer 交互的话, 它也创建了一个会话。因此, 每个会话被“关联”到了一个单个的 single 中-其中一个 dialog 会最终导致会话创建。如果一个初始的 INVITE 生成一个非 2xx 最终响应, 此响应结束通过响应到此请求所有

会话（如果有的话）和所有 dialogs（如果有的话）。因为成功完成事务处理，一个非 2xx 最终响应也会防止请求创建的更多会话。BYE 请求是用来结束一个特定的会话或未处理的会话。在这种使用场景中，此特定会话是一种特殊会话，它支持了对端 dialog 的 peer UA。当从一个 dialog 中收到 BYE 请求以后，任何和此 dialog 关联的会话都应该结束。在此 dialog 之外，UA 一定不能再发送 BYE 请求。呼叫方的 UA 可以为确认的 dialog 或者早期的 dialog 中发送 BYE，被呼叫方 UA 可以在确认的 dialog 中发送 BYE，但是一定不能在早期的 dialog 中发送 BYE。

但是，这里需要说明。直到被呼叫方 UA 收到其 2xx 响应的 ACK 或者服务器端事务超时之前，被呼叫方 UA 一定不能在确认的 dialog 中发送 BYE。如果没有已定义的 SIP 拓展应用层状态关联了此 dialog 的话，此 BYE 也会结束此 dialog。

在 dialog 中的针对 INVITE 非 2xx 最终响应的影响和会话使用了 CANCEL 的优点。CANCEL 会尝试强制 INVITE 的非 2xx 响应处理（特别是 487 状态码处理）。因此，如果一个 UAC 希望完全放弃其呼叫意愿，它可以发送一个 CANCEL。如果此 INVITE 导致一个对其 INVITE 的 2xx 最终响应，这说明在 CANCEL 正在处理过程中，UAS 接受了此邀请。此 UAC 可以通过任何 2xx 响应创建的会话继续执行或者使用 BYE 结束会话。

在 SIP 中，“hanging up”概念没有定义的非常完整。虽然它是非常常用的用户接口，它仍然需要在非常具体的流程中进行处理。一般来说，当用户挂机时，用户会指示一个期望表示结束一个尝试（试图创建会话），并且结束任何已创建的会话。如果初始 INVITE 还没有生成最终响应，并且最终响应以后，还没有对所有确认的 dialog 生成一个 BYE 消息，对于呼叫方 UA 来说，结束会话将会使用一个 CANCEL 请求。对于被呼叫方 UA 来说，它一般将会使用一个 BYE 消息；一般情况下，当用户接听电话时，会生成一个 2xx 响应消息，并且，挂机后，ACK 收到后会导致生成一个 BYE 消息产生。这里的意思不是说用户在收到 ACK 前不能挂机，它仅表示电话的软件系统需要维持一个短暂的状态来完成整个流程的完整处理。如果一个特别的 UI 允许此用户在接听之前拒绝呼叫，返回一个 403 是比较好的处理方式。对于以上每个规则来说，这里，BYE 消息不会被发送。

## 15.1 Terminating a Session with a BYE Request

### 15.1.1 UAC Behavior

BYE 请求的构建方式和其他在 dialog 中的请求的构建方式是一样的，读者可参考第 12 章。

一旦 BYE 消息构建以后，UAC core 创建一个新的非 INVITE 客户端事务，并且，传递给客户端事务这个 BYE 请求。UAC 必须考虑，只要 BYE 请求传递到客户端事务时，会话就要结束（还要停止发送和监听媒体）。如果 BYE 请求的回复的响应是 481 (Call/Transaction Does Not Exist) 或一个 408 (Request Timeout) 或对此 BYE 请求完全无响应（客户端事务返回一个超时），此 UAC 必须视为会话和 dialog 已结束。

#### 15.1.2 UAS Behavior

UAS 首先处理 BYE 请求，处理流程根据一般的 UAS 处理流程，具体过程描述参考第 8.2 章。收到了 BYE 请求的 UAS 检查是否此 BYE 请求匹配现存的 dialog。如果此 BYE 请求不能匹配现存的 dialog，UAS core 应该生成一个 481 响应 (Call/Transaction Does Not Exist)，并且传递此响应到服务器端事务端。

这个规则的含义是由 UAC 发送的 BYE 请求，没有标签的话将会被拒绝。这个修改来自于 RFC 2543，此规范中允许 BYE 无标签。

从一个现存 dialog 中收到 BYE 请求的 UAS core 必须遵从此规则来处理请求，规则参考第 12.2.2 章节。一旦完成处理，UAS 应该结束此会话（并且停止发送和监听媒体）。仅有一种情况需要注意，UAS core 能够决定不进行多播会话，多播会话中一个参与方是可能继续存在的，即使其他在 dialog 中的参与方已经结束其会话参与过程。无论是否由 UAS core 结束其在会话中的参与方，此 UAS core 必须对 BYE 请求生成一个 2xx 响应，并且必须传递此响应到服务器端事务来进行传输。

UAS 必须仍然对 dialog 中正在处理的请求进行响应。规范推荐对正在处理的请求生成一个 487 (Request Terminated) 响应。

## 16 Proxy Behavior

### 16.1 Overview

SIP 代理是 SIP 协议中的核心构件，它把 SIP 请求路由到用户代理服务器，并且把 SIP 响应路由到用户代理客户端。一个请求可能通过自己的方式穿越多个代理服务器到 UAS 端。每个代理将会做它们的路由决定，在转发到下一个网络构件前修改此请求。响应消息将会通过同样的代理请求穿越的路径，以相反的顺序路由到初始 UAC 端。

代理是 SIP 网络网元扮演的一个逻辑角色。当一个请求到达时，扮演代理角色的网元构件首先决定代理是否通过自己去做出响应。例如，此请求可能是一个错误的请求或在作为代理执行之前，客户端网络网元需要安全机制验证。SIP 网元可以通过任何合适的响应码来回复。

当直接对请求进行响应时，此网元扮演 UAS 的角色，执行方式参考第 8.2 章节内容。

针对每个新的请求，代理可以以状态模式或无状态模式工作。当代理以无状态模式工作时，代理工作方式就是一个简单的网络转发网元。它转发每个请求到一个单点网络网元，此网络网元是基于请求做出的目的地和路由决定来最终决定的。它仅简单转发从它上游收到的响应。无状态代理一旦转发了一个消息以后，它会丢弃此消息。有状态代理会记录每个呼入的请求(特别是事务状态)和其他呼入到此代理，经过代理处理的结果。有状态代理使用这些记录的消息来影响和此请求关联的未来消息的处理流程。有状态代理可以选择复制一个请求，并且路由这些复制的请求到多个目的地地址。任何请求，它可以转发到一个以上地址的话，它必须由状态代理来进行处理。

在某些环境中，代理可以使用有状态传输（例如 TCP）来前转请求，而无需是一个有状态事务。例如，代理可以从一个 TCP 连接的传输前转请求到另外一个无状态事务，只要此代理在消息中写入足够消息可以保证响应回复到同样的连接，此连接是那个请求抵达的节点连接。在介于不同传输类型的请求前转中，而且传输节点中代理的 TU 必须充当一个活动角色来确保传输可靠性，这样的请求前转必须是有一个前转的状态事务。

只要有状态代理没有执行任何处理，有状态代理可以在任何处理请求的时间内切换为一个无状态代理；否则，有状态代理最初就禁止切换为无状态代理（例如，复制或者生成 100 响应等）。当执行这样的切换时，所有的状态消息都会被丢弃。此代理不应该初始化 CANCEL 请求。

当针对一个请求，作为无状态代理或有状态代理会涉及很多的处理流程。在后续章节中会从有状态代理的角度做进一步的详解。最后一部分是说明无状态代理的处理过程。它们两者的处理完全不同。

## 16.2 Stateful Proxy

当代理是一个有状态代理时，此代理完全是一个 SIP 事务处理的引擎。它的执行方式通过服务器端和客户端的各种条件进行规范化处理，具体的执行方式在第 17 章中定义。有状态代理有服务器事务，服务器端事务通过更高级的代理处理模块关联一个或多个客户端事务，这个模块称之为 代理 core 模块（参考 figure3）。呼入的请求通过服务器端事务进行处理。

请求从服务器端事务传递到代理 core 模块（proxy core）。Proxy core 决定路由请求的目的地，选择一个或多个下一跳地址。每个下一跳地址的呼出请求通过它们自己关联的客户端事务来处理。Proxy core 从客户端事务收集响应消息，然后使用客户端事务发送响应消息到服务器端事务端。

有状态代理会对每个收到的新请求创建一个新的服务器端事务。请求重传由服务器端事务进行处理，具体处理方式参考第 17 章。代理的 core 模块必须像 UAS 一样在服务器端事务发送一个直接临时响应（例如 100 trying），执行方式在第 8.2.6 章节有描述。因此，有状态代理不应该对非 INVITE 请求生成 100 (Trying)。

这是一个代理工作的模式流程，不是一个软件。部署方式是自由的，用户可以使用任何方式按照代理工作模式复制一个外部处理流程。

对于所有新的请求，包括任何未知的，一个想要代理请求的网络网元必须：

1. 验证此请求 (第 16.3 章节)
2. 预处理路由信息 (第 16.4 章节)
3. 决定此请求的目的地 (第 16.5 章节)

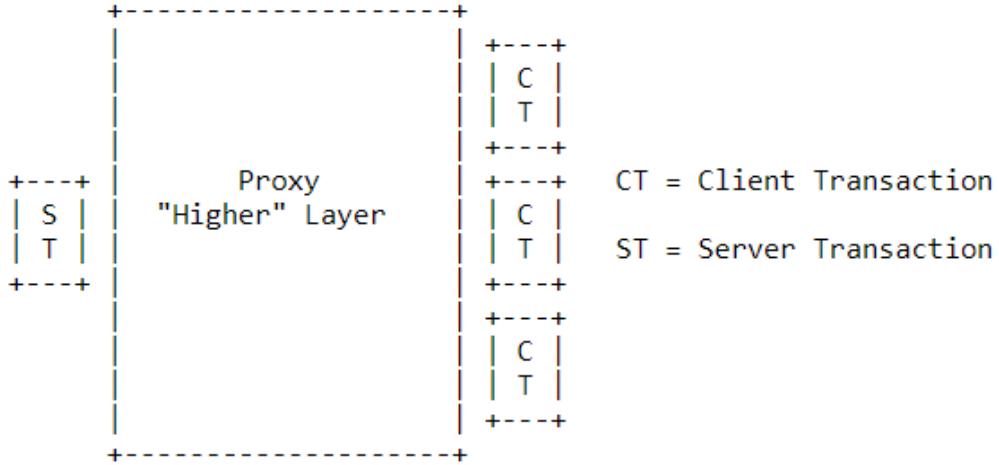


Figure 3: Stateful Proxy Model

4. 转发请求到每个目的地 (第 16.6 章节)
5. 处理所有响应 (第 16.7 章节)

### 16.3 Request Validation

在网络网元代理一个请求之前，网络网元必须验证此消息的有效性。一个有效的信息一定通过以下检查：

1. 正确的语法
2. URI scheme
3. Max-Forwards
4. (可选) Loop Detection
5. Proxy-Require
6. Proxy-Authorization

如果其中任何一个检查失败的话，网络网元必须作为用户代理服务器来执行（参考第 8.2 章节）并且返回错误码。

注意，代理不要求检查合并的请求，代理也一定不能视合并的请求为一个错误条件。收到此请求的终端将解决此合并情况，具体合并处理参考第 8.2.2 章节。

#### 1 正确语法检查

请求必须具备完好的语法结构来处理服务器端事务。任何涉及请求验证的其他步骤或请求前转部分必须具备完整的语法结构。当消息前转时，任何语法结构完整或不完整的都应该被忽略，其余部分保持不变。例如，因为请求中包含了一个异形的日期头值，网元将拒绝此请求。同样地，一个代理在前转请求时，它也不会移除这个异形的 Date 头值。

本协议的设计是可以支持可拓展的。未来的拓展可以在任何时候定义新的 methods 和新的头字段值。因为请求中包含一个 method 或一个头字段值，网元不能理解这些值得话，网元也一定不能拒绝代理一个请求。

## 2 URI scheme 检查

如果此 Request-URI 有一个 URL，代理不能理解它的语法结构，代理应该拒绝此请求，并且回复一个带状态码 416 (Unsupported URI Scheme) 的响应。

## 3 Max-Forwards 检查

Max-Forwards 头字段（第 20.22 章节）是用来限定 SIP 请求可穿越的网元节点数量。

如果此请求中不包含一个 Max-Forwards 头的话，请求检查将会被通过。

如果此请求包含一个 Max-Forwards 头字段值，头字段值大于零的话，检查将会被通过。

如果此请求包含一个 Max-Forwards 头字段值，其值为零，此网络网元一定不能前转此请求。如果此请求是一个可选请求，此网元可以作为最终接收方，并且通过一定的响应流程，具体流程参考第 11 章节。否则，网络网元一定要返回一个 483 (Too many hops)响应消息。

## 4 可选 Loop Detection 检查

网元在前转请求前可以对其请求进行前转回环检查。如果此请求包含一个 Via 头字段值，携带了一个 sent-by 值，其值等于代理置于前面请求中的值的话，那么，此请求已经在以前被网元前转。此请求要么已经是回环状态，或者正在合法通过此网络网元节点中。如果要决定此请求是否是回环状态，网元可以执行消息中的 branch 参数的计算，计算步骤描述在第 16.6 章节的第八步，然后和在 Via 头中收到的参数进行对比。

如果参数是匹配的，则说明请求是回环状态。如果对比参数的匹配价格是不同的，则说明请求是变化的，流程进行进行。如果检测到一个 loop 回环的话，网络网元可以返回一个 482 (Loop Detected) 响应。

## 5 Proxy-Require 检查

此协议未来的拓展可以引入新的功能，这些新功能要求代理做特别的处理。终端将在请求中包括一个 Proxy-Required 头来表示将使用这些功能，通知代理不要处理此请求，除非代理理解此头中的功能。

如果此请求中包含一个 Proxy-Require 头字段（第 20.29 章），并且包含一个或者多个代理不能理解的可选标签的话，网络网元必须返回 420 (Bad Extension) 响应。此响应必须包括一个不支持的头字段列表，列表中列出网络网元不能理解的可选标签（第 20.40 章节）。

## 6 Proxy-Authorization 检查

如果网元在前转请求前要求安全处理的话，此请求必须被进行安全检测，检测流程在第 22.3 章节。那个章节同时也定义了如果检测失败以后必须处理的流程。

### 16.4 Route Information Preprocessing

代理必须检查请求的 Request-URI。如果请求的 Request-URI 包含了一个值，此值已经被代理置入到了 Record-Route 头字段中（参考 16.6 章节 item 4），此代理必须替换这个请求中的 Request-URI 值，替换使用的值是来自于 Route 头字段值，并且从 Route 头字段中移除此值。然后代理必须按照它收到这个修改的请求那样去处理。

这个情况仅发生在当网络网元是一个严格路由器时，它发送请求到此代理处（也许已经是一个终端）。这样对接收的重写入是必要的，可以帮助和那些网络网元的向后兼容。通过严格路由的代理时，它也允许遵从此规范的网络网元维护 Request-URI（第 12.2.1.1 章节）。

此要求不会迫使代理来维持一个状态以便检测 URLs，这些 URLs 是代理前面置于 Record-Route 头字段中的 URLs。相反，代理仅需要在那些 URLs 中置于足够的信息作为参考值，使用以后出现这些 URLs 时，使用这些参考值来识别这些 URLs。

如果 Request-URI 包含一个 maddr 参数的话，此代理必须检查一系列地址中或者 domain 中的 maddr 的值，代理需要针对这些值进行配置对其负责。如果此 Request-URI 有一个 maddr 参数，其值是代理需要支持的，并且收到的请求使用了在 Request-URI 中明确地或者默认指示了这个端口和传输方式，此代理必须截取这个 maddr 和任何非默认端口或传输参数，并且好像那些值没有出现在请求中一样继续流程处理。

请求可以抵达时携带一个 maddr 地址匹配这个 proxy，但是可以和 URL 指示的端口或参数不同。这样的请求需要使用指示的端口和传输模式前转到这个代理。

如果在 Router 域中的第一个值指示这个代理的话，这个代理必须从请求中移除这个值。

#### 16.5 Determining Request Targets

下一步，代理计算请求的目的地。一组目的地将可能会是由请求消息预设的目的地或者可能会是通过抽象位置服务获得的目的地。在组中的目的地以 URL 的方式来呈现。

如果请求的 Request-URI 中包含一个 maddr 参数的话，此 Request-URI 仅作为目的地 URL 必须被置于目的地组，代理必须按照第 16.6 章节的流程来处理。

如果 Request-URI 的域指示了一个域，此网元不对此域负责的话，Request-URI 必须仅作为目的地被置于目的地组，并且此网元必须处理请求前转的任务，处理流程按照第 16.6 章节来处理。

很多情况下，代理可能收到了多一个域的请求，这个域不对此请求负责。处理呼出呼叫的防火墙代理（HTTP 代理负责呼出请求）是一个比较典型的示例。

如果请求的目的地组就像上面所描述的还没有被预设，这表示此网络网元需要对 Request-URI 中的域（domain）负责，并且此网络网元可以使用任何的网络网元期望的处理机制来决定发送此请求的目的地。作为访问抽象位置服务，这些机制中任何处理方式可以被经过模型化处理。这样的处理方式可能由几种方式来实现，包括从位置服务（由 SIP 注册服务创建）获得信息，读取一个数据库，查询在线服务器，利用其他协议，或者在 Request-URI 中执行简单的算法替换等方式。当访问由注册服务创建的位置服务时，Request-URI 必须首先 被进行规范化格式处理，处理流程按照第 10.3

章节进行，处理以后才能作为一个指示来使用。这些处理机制的结果用来构建目的地组。

如果此 Request-URI 不能提供足够的信息让代理决定目的地组的话，代理一个返回错误状态码 485 (Ambiguous) 响应。此响应中应该包含一个 Contact 头，此头值应该包含需要尝试的新的地址的 URL 消息。例如，一个 INVITE 请求中的 sip:John.Smith@company.com，在代理处可能不是非常确定的，此代理服务器的位置服务中可能包含多个 John Smiths 地址，如何处理类似的流程，需要按照第 21.4.23 章节来进行处理。

在请求中的任何信息或者请求相关的消息，或者当前网络网元环境的消息可以用来构建目的地组。例如，不同的目的地组可以依赖于消息头的内容，呈现内容或者依赖于消息体的呈现内容，请求抵达时间日期，请求通过何种接口抵达，历史请求失败记录或者甚至于网络网元当前使用层级等信息。

当潜在的目的地通过这些服务被定位以后，他们的 URL 会被添加到目的地组。目的地仅被置入到目的地组一次。如果目的地 URL 已经出现在了此组中（基于 URL 联系相同的定义），此目的地 URL 不能再次添加到目的地组中。

如果原始请求的 Request-URI 不指示代理对网络资源负责的话，此代理一定不能对目的地组添加额外的目的地。

在前转过程中，如果代理负责此 URL 的话，代理仅修改请求的 Request-URI。如果代理不对此 URL 负责的话，就像下面描述的，代理将不递归处理 3xx 或者 416 响应。

如果原始请求的 Request-URI 指示了一个代理所负责的网络资源，开始请求前转后，代理可以继续对目的地组添加一个目的地。代理可以使用任何在处理决定目的地流程中的信息。例如，代理可以选择并入在重转发响应 (3xx) 中获得的 contacts 到目的地组。当代理构建目的地组时，它使用了信息的动态资源（例如，它查询了 SIP 注册服务），代理应该为处理请求时段来监控那个资源。如果有新的有效地址出现以后，新地址应该添加到目的地组中。就像前面介绍的，如何已给定的目的地 URL 一定不能再次添加到目的地组。

仅允许一次 URL 添加到组会决定无必要的网络流量，并且万一合并从转发请求过来的 contacts 的话也可以防止无限循环的极端情况发生。

例如，一个正常的定位服务是一个 "no-op"，定位服务的目的地 URL 和呼入的请求 URL 相同。此请求被发送到一个特定的下一跳代理做进一步处理。在请求前转期间（按照处理，第六条的规范中，下一跳的身份作为 SIP 或者 SIPS URL，它被插入到请求的 SIP 头中，作为 top-most Route 头字段的值。

如果 Request-URI 表示的一个在此代理端的资源不存在，代理必须返回一个 404 (Not Found) 响应。

如果目的地组按照以上规则进行了处理以后，目的地组仍然为空，代理必须返回错误响应，这里应该返回一个 480 (Temporarily Unavailable) 临时响应。

## 16.6 Request Forwarding

只要目的地组是非空状态，代理可以开始前转此请求。有状态代理可以以任何顺序处理目的地组。代理可以连续处理，它可以允许每个客户端事务完成前启动下一个处理。代理也可以并行启动客户端事务支持每个目的地组队处理。代理也可以任意划分目的地组，让目的地组成为不同的小组，连续处理这些小组，并行处理每个小组中的目的地地址。

常用的顺序机制是使用目的地的参数 (qvalue) 作为处理顺序机制的参考，这个参数值从 Contact 头中获得（参考第 20.10 章节）。目的地处理顺序按照从最高值到最低值的顺序来处理。目的地值中的 qvalues 相同的话，则执行并行处理。

当有状态代理收到响应后，有状态代理启用原始请求，使用前转请求关联这个响应，有状态代理必须具备一个机制来维护目的地组。对于此模式的作用来说，这个机制是一个 "response context"，在前转第一个请求之前，它是由代理层创建。

对于每个目的地来说，代理前转请求需要根据以下步骤来实现：

1. 对收到的请求进行拷贝

2. 更新此 Request-URI
3. 更新此 Max-Forwards 头字段
4. 可选择地添加一个 Record-route 头字段值
5. 可选择性地添加其他的头字段值
6. 后处理路由信息
7. 决定此下一跳地址，端口和传输
8. 添加一个 Via 头字段值
9. 如有必要，添加一个 Content-Length 头字段
10. 前转此新请求
11. 设置定时器 C

以上每个步骤的细节如下：

### 1 拷贝请求

代理以拷贝收到的请求为启动步骤。此拷贝必须一开始就包含从收到请求中获得的所有头字段值。一些在处理过程中无描述细节的头字段一定不能被移除。此拷贝应该包含头字段的顺序，头字段的顺序是接收的请求中头字段的顺序。代理一定不能使用一般的字段（第 7.3.1 章节）重新对这些头字段进行排序。代理一定不能添加，修改或者移除消息体。

在实际使用中无需执行拷贝；最基本的要求是对每一个下一跳处理开始时使用同样的请求。

### 2 Request-URI

在拷贝中起始行的 Request-URI 必须使用此目的地的 URL 替换。如果此 URL 中包含任何参数，此 Request-URI 不支持这些参数的话，这些参数必须被移除。

这是一个代理的角色的本质特征。这是一个处理机制，请求通过代理路由其请求前转到它的目的地。

在某些环境中，收到的 Request-URI 会被置入到目的地组，无需任何修改。对于这样的目的地，以上替换实际上没有执行任何操作。

### 3 Max-Forwards

如果拷贝中包含了一个 Max-Forwards 头字段，代理必须递减其值，递减一。

如果此拷贝中没有包含 Max-Forwards 头字段，代理必须添加一个这样的头字段，设置的值应该是 70。

目前一些现存的 UAs 将在请求中不提供 Max-Forwards 头字段。

### 4 Record-Route

如果代理希望维持在 dialog（假设是由此请求创建的）中的将来的请求路径，即使一个 Route 头字段已经出现，它必须在拷贝中插入一个 Record-Route 头字段，这个 Record-Router 插入在任何现存 Record-Router 头之前。

请求创建 dialog 的流程可能包含一个预加载的 Route 头字段。

如果此请求已经是一个 dialog 的一个部分，如果代理希望在 dialog 中保持将来请求的路径，代理应该插入一个 Record-Route 头字段值。在正常的终端操作流程中，参考第 12 章节，这些 Record-Route 头字段值将不会对由此终端创建的路由组有任何影响。

如果此代理选择不在请求中插入一个 Record-Route 头字段值，而此请求已经是 dialog 的一个部分的话，此代理将维持此路径。但是，当失败的终端重新创建此 dialog 时，代理将移除此路径。

代理可以在任何请求中插入一个 Record-Route 头字段值。如果此请求没有初始化一个 dialog 的话，此终端将会忽略这个 Record-Route 值。参考第 12 章获得终端如何使用 Record-Route 头字段值构建 Route 头字段的细节。

每个在请求路径的代理选择是否独立添加一个 Record-Route 头字段值-在请求的 Record-Route 头字段的出现不强迫此代理添加一个新的值。

被置入到此 Record-Route 头字段的 URL 必须是一个 SIP 或者 SIPS URL。此 URL 必须包含一个 lr 参数（参考第 19.1.1 章节）。此 URL 可以针对不同目的地，此目的地是一

个请求前转的目的地地址。此 URL 不应该包含传输参数，除非此代理已经确认（例如，在私有网络）下一个下游网络网元，此网络网元将在候选请求的路径中支持此传输。

此代理提供的 URL 将被一些其他的网络网元使用作路由决定。通常来说，此代理没有办法知道网络网元的能力，因此，它必须限定自己为 SIP 部署中的强制网元：SIP URLs 和 TCP 或者 UDP 传输。

当此服务器定位流程[参考链接 4]执行时，置入到 Record-Route 头字段的 URL 必须解析到网络网元，网络网元插入此 URL，因此后续的请求抵达同样的 SIP 网络网元中。如果 Request-URI 包含一个 SIPS URI 或者最顶部的路由域值（post processing 项目步骤 6 后）包含一个 SIPS URI，置入到 Record-Route 头字段的 URL 必须是一个 SIPS URL。进一步说，如果此请求不是通过 TLS 收到的，此代理必须插入一个 Record-Route 头字段值。在相同的工作方式中，通过 TLS 收到请求的代理，但是它生成一个请求，在请求的 Request-URI 或者顶部路由头字段中没有 SIPS URL（post processing 项目步骤 6 后），此代理必须插入一个不是 SIPS URL 的 Record-Route 头字段。

在安全边界的代理贯穿整个 dialog，它必须维持其边界安全。

如果一个 URL 通过响应返回后，这个被置入在 Record-Route 头中的 URL 需要被重写，这个 URL 必须在那个时间被明确定位（请求通过此代理时会急增导致多个 Record-Route 头被添加）。第 16.7 章节的 item 8 中推荐了一个机制使得 URL 可以明确区别于其他的 URL。

此代理可以在 Record-Route 头字段中包括参数值。这些参数值将被一些请求的响应回复，比如 INVITE 请求的 200 (OK) 响应。这样的参数可能非常有用，例如可以支持在消息中保持状态，而不是在此代理中保持状态。

如果一个代理需要在 dialog 的任何类型的路径的话（例如穿越防火墙），此代理应该使用 method 对每个请求添加一个 Record-Route，代理不理解此 method，因为此 method 可能包含 dialog 语法规则。

被代理置于 Record-Route 头中的 URI 仅对任何 dialog 的生命周期有效，当事务产生时，dialog 由事务来创建。例如，dialog 结束以后，一个有状态的 dialog 代理可以拒绝接受任何后续请求中的 Request-URI 携带的值。对于非状态代理来说，它当然没有

概念何时结束 dialog，但是，这些非状态代理可以对 Request-URI 携带的值进行解码来获得足够消息，然后和后续的请求中的 dialog 身份进行对比，并且这些代理可以拒绝消息不匹配的请求。终端一定不能使用从 Record-Route 获得的 URL，Record-Route 是一个终端提供的外部 dialog 的 Record-Route。参考第 12 章节获得更多关于终端使用 Record-Route 头的消息说明。

某些服务要求 Record-routing 支持，在这些服务中，代理需要检测 dialog 中所有的消息。但是，这样会降低处理流程的速度，并且会削弱服务的拓展性，因此，如果一个特殊服务要求 record-route 时，代理则应该添加一个 record-route。

Record-Route 处理流程可以和发起 dialog 的任何 SIP 请求工作。在此规范中，INVITE 就是这样的一个请求，但是此协议的拓展可以定义其他的请求。

## 5 添加额外的头字段

代理可以在此处理点所对头字段拷贝添加合适的头字段。

## 6 后处理路由信息

代理可以支持一个本地策略，在发送路由请求到目的地之前，这个策略强制一个请求访问一系列指定的代理。代理必须确保所有这样的代理是 loose routers（松散路由/静态路由）。通常来说，如果这些代理都在同一管理域环境中的话，代理运行模式是可以确定的。这里的代理组通过一系列 URLs 来表示（每个代理包含它的 lr 参数）。如果存在这样的组设置的话，这样的组设置必须推送到当前已存在的 Router 头拷贝之前。如果缺省了 Route 头，必须添加这个 Route 头，并且包含一个 URLs 的列表。

如果此代理支持一个本地策略，这个策略强制请求访问一个指定的代理，相对于把 Route 值推送到 Route 头来说，策略会使用旁路分流的方式，通过以下条目 10 中的前转逻辑来处理请求，并且仅针对此指定的代理发送请求到此地址，端口和传输方式。如果此请求有一个 Route 头，除非知道下一跳是一个 loose 路由（静态路由），一定不能使用这种可选方式。除此之外，这种处理方式也可以被使用，但是推荐使用前面说提到的路由插入的机制来保证其健壮性，灵活性，通用性和运行的一致性。进一步来说，如果此 Request-URI 包含一个 SIPS URI 的话，必须使用 TLS 和那个代理进行通信。

如果此拷贝中包含了一个 Route 头字段，此代理必须首先检查其第一个值中的 URL。如果那个 URL 没有包含一个 lr 参数的话，代理必须按照以下流程修改此拷贝：

- ✓ 代理必须在 Route 头中置入 Request-URI，此 Request-URI 作为最后的一个值。
- ✓ 然后代理必须在 Request-URI 中置入第一个 Route 头字段值，并且从 Route 头字段中移除此值。

增补到 Route 头字段的这个 Request-URI 是一个机制的部分网元，这个机制用来通过严格路由网元在那个 Request-URL 中传递信息。分离第一个 Route 头字段值置入到 Request-URI 的处理方式构建了这个消息，严格路由网元希望通过这种方式接收此信息（信息中的 Request-URL 带自己的 URL 和在第一个 Route 头字段值中的下一个将要访问的地址）

## 7 决定下一跳地址，端口和传输方式

代理可以不依赖于 Route 和 Request-URI 的值，它有自己的本地策略来发送请求到一个指定的 IP 地址，端口和传输方式。如果此代理没有包含对应服务器的 IP 地址，端口和传输方式，这个服务器是一个 loose router 的话，这个本地策略一定不能被使用。无论以何种方式处理，通过一个指定的下一跳节点发送请求的这种机制是本规范不推荐的方式；相反，应该使用 Route 头字段值来满足上面所描述的处理目的。

在缺乏这种重要机制的情况下，代理使用在[参考链接 4]列表中的处理流程进行操作处理来决定此请求发送到地址。如果代理已经重新构建了请求，将请求发送到一个严格路由的网元时（上面第六步所描述的），这个代理必须使用那些流程来处理此请求中的 Request-URL。否则，如果 Route 头中有第一个值，这个代理必须使用此流程处理 Route 头字段中的第一个值，没有的话，处理 Request-URL 值。这个处理流程将会生成一个按序排列的（地址，端口和传输方式）的元组。对于流程[参考链接 4]来说，它不依赖于哪个 URL 作为输入处理的值，如果这个 Request-URI 指定了一个 SIPS 资源，输入的 URL 是一个 SIPS URL，代理必须遵从处理流程[参考链接 4]。

按照[参考链接 4]所描述的那样，代理必须尝试对元组中的第一组发送消息，然后按序处理其他元组地址，直到发送尝试全部成功完成。

对于每个尝试发送到元组，代理必须根据元组值的情况为此元组构建恰当的消息，使用一个新的客户端事务来发送此请求，具体新客户端事务的细节在第八步到第十步有描述。

因为每个尝试都使用了一个新的客户端事务，它代表了一个新的 branch。因此，通过 Via 头值（在第八步插入的）提供了 branch 参数必须区别于每个其他的尝试所使用的参数。

如果客户端事务对发送请求报告了一个失败，此代理会继续有序元组中的下一个地址。如果这个有序元组调用完成后，请求不能被前转到目的地组的网络网元。此代理无需在响应内容中置入任何其他内容，然而在其他方面，如果目的地组中的这个网络网元返回一个 408 (Request Timeout/请求超时) 最终响应的话，它也应该那样工作。

## 8 添加一个 Via 头字段值

代理必须在现存的 Via 头字段值之前插入一个 Via 头字段值。构建此值的流程遵守第 8.1.1.7 章节的指导规范。代理使用此指导计算自己的 branch 参数，这样的部署计算对那个 branch 来说是全局唯一的，并且包含必不可少的梦幻饼的值（magic cookie）。注意，这样的处理方式针对 branch 参数会产生不同的结果，经过代理的螺旋式请求或者回环请求的实例的 branch 参数会不同。

选择检测回环的代理还有一个额外的限制的值，这个值是这些代理用来构建 branch 参数的值。一个选择检测回环的代理应该由此部署方式创建一个可分离的 branch 参数，branch 参数分离为两个部分。第一部分必须满足第 8.1.1.7 章节中上面描述的限定。第二部分用来执行回环检测和用于从螺旋式请求区分回环请求的处理。

回环检测的验证是通过验证请求返回的一些头来最终确认是否是回环流程。具体来说，当一个请求返回到一个代理时，一些影响此请求处理的头字段值并没有发生改变。置入到 branch 参数部分的值应该反映所有的这些头字段（包括任何 Route, Proxy-Require 和 Proxy-Authorization 头字段值）。这是确保如果请求被路由回到此代理，并且这些值中的其中一个值是已经改变的，此请求就会被看作是一个螺旋式请求，而不是一个回环请求（参考第 16.3 章节）。创建此值的通常的做法是为了计算 To 标签，From 标签，Call-ID 头字段，收到的请求的 Request-URL（转译前），topmost Via 头，

CSeq 头字段的序列号，另外还有可能出现的 Proxy-Require 和 Proxy-Authorization 头字段值的安全哈希计算。

用来计算哈希的算法是和实现方式相关的，但是使用 MD5 算法 ((RFC1321 [35])，以 hexadecimal 十六进制的方式表达，这是一个恰当的选择（对于 token 来说，Base64 是不允许的）。

如果代理要检测回环，代理所提供的"branch" 参数必须依赖于所有影响此请求的信息，包括入局的 Request-URI 和任何其他头字段值，这些头字段值影响此请求的权限和路由策略。这是非常有必要的一个步骤，通过这个步骤可以此请求中区分这个请求是否是一个回环的请求，主要检测请求的路由参数在返回到此服务器之前是否已经发生了改变。

此请求的 method 一定不能包含在此请求 branch 参数的计算中。特别是 CANCEL 和 ACK 请求（对非-2xx 响应），它们必须具有同样的 branch 参数值，这些参数值就是它们对应的请求 branch 参数，这里的请求是它们所取消的或者确认的请求。此 branch 参数用在服务器端处理那些请求的流程中，用来关联那些相关请求（参考第 17.2.3 章节和第 9.2 章节内容）。

## 9 如有必要，添加一个 Content-Length 头字段

如果此请求将使用基于流传输方式被发送到下一跳，并且拷贝中包含一个无 Content-Length 的头字段，此代理必须为这个请求的消息体插入一个正确的值（具体操作参考第 17.14 章节）。

## 10 前转请求

有状态代理必须为此请求创建一个新的客户端事务，具体创建客户端事务流程请参考第 17.1 章节并且代理命令事务使用地址，端口和传输方式（第七步）来发送此请求。

## 11. 设置定时器 C

为了处理 INVITE 请求从来没有生成一个最终响应这种场景，TU 使用了一个定时器-称之为定时器 C。当请求被代理以后，必须为每个客户端事务设置定时器。此定时器必

须大于 3 分钟。第 16.7 章节第二部分讨论了如何使用临时响应更新定时器，当定时器被触发以后，第 16.8 章节讨论了如何进行处理的流程。

### 16.7 Response Processing

当网元收到一个响应以后，它会首先尝试配置一个客户端事务（第 17.1.3 章节）匹配此响应。如果没有发现任何匹配，网元作为一个无状态代理必须处理此响应（即使此响应是一个消息类响应）。如果发现匹配，客户端事务处理此响应。

对于没有发现客户端事务场景（或更多针对任何已发送关联请求的确认消息）来说，前转响应提高了系统的健壮性。尤其它确保了 INVITE 请求的“稍晚”的 2xx 响应可以恰当地进行转发处理。

当客户端事务传输响应到代理层以后，必须经过以下处理流程：

1. 找到合适的响应消息内容
2. 为临时响应更新定时器 C
3. 移除最顶部的 Via
4. 对此响应消息内容添加响应
5. 查看如果响应应该被马上快速前转
6. 当必要时，从响应消息内容中选择最佳的最后响应

通过响应消息内容和每个客户端事务关联结束后，如果无最终响应被前转，此代理必须从它处理过的响应中选择并且前转这个“最佳”响应。

以下处理流程必须在前转的响应中执行。它更像是对每个请求的多个响应的前转处理：至少每个临时响应和一个最终响应。

7. 如有必要，聚合授权头字段值
8. 有选择地重写 Record-Route 头字段值
9. 前转此响应
10. 生成任何必要的 CANCEL 请求

以上每个步骤的细节如下：

1. 找到 Context

代理使用在第 16.6 章节中描述的方式前转初始请求之前，它要定位自己创建的此“response context”。其余的处理步骤发生在此 context 中。

2. 为临时响应更新定时器 C

对一个 INVITE 事务来说，如果响应是一个临时响应，它携带的状态码是包括在 101 到 199，此代理必须为那个客户端事务重新设置定时器 C。这个定时器可以重设为不同的值，但是，这个值必须大于 3 分钟。

3. Via

代理从响应中移除最顶部的 Via 头值。

如果在响应中没有保留 Via 头字段值，此响应是本网络网元的，并且响应一定不能被前转。在章节中所描述的其余部分流程就不会在此响应中继续执行，在第 8.1.3 中所描述的 UAC 处理流程会替代以上部分的处理来继续执行（传输层流程已发生）。

当网元生成 CANCEL 请求就像第 10 章节所描述的那样时，这些就会发生。

4. 在 context 增加响应

直到最终响应在服务器端事务中生成，最终响应关联了 context 后，收到的最终响应将被保存在响应 context 中。这个响应可能是一个候选响应，在服务器端事务中，它将作为最佳最终响应被返回。即使没有选择此响应，也可能需要响应中的信息来构建最佳响应。

如果代理选择在 3xx 响应中通过添加 contacts 到目的地组的方式递归任何 contacts，在添加响应到响应 context 中之前，代理必须从响应中移除 contacts。但是，如果原始请求的 Request-URL 已是一个 SIPS URL 的话，代理不应该递归到一个非 SIPS URL 地址。

如果代理在 3xx 响应中递归所有的 contacts，代理不应该在响应 context 添加随之生成的无联系的响应。

添加响应到 context 之前要移除 contact，其目的是防止下一个网络网元向上游重新获取一个此代理已经尝试的地址。

3xx 响应可以包含一个 SIP, SIPS 和非 SIP URLs 的混合体。代理可以选择递归处理 SIP 和 SIPS URL，并且把其余部分置于响应的 context 作为返回的 context，可能在最终响应中返回。

如果代理针对请求收到一个 416 (不支持的 URL Scheme) 响应的话，此请求的 Request-URI scheme 不是 SIP，但是在最初的 scheme 中，它收到的请求是 SIP 或者 SIPS (当代理处理请求时，这是代理已从 SIP 或者 SIPS 修改到了其他的类型)，这个代理应该对目的地组添加一个新的 URL。这个新的 URL 应该是一个已尝试的非 SIP URL 版本的一个 SIP URL 版本。在 tel URL 的场景中，通过把 tel URL 的 telephone-subscriber 部分置于 SIP URL 的用户部分来完成，把主机部分设置为 domain，这里的 domain 是前面发送的请求。参考第 19.1.6 章节获得更多关于从 tel URL 构建 SIP URL 的详情。

和处理 3xx 响应一样，如果代理通过递归方式对 416 处理，它是通过尝试使用一个 SIP 或者 SIP URL 的话，这个 416 响应不应该被添加到此响应的 context 中。

## 5. 为前转检查响应

直到一个最终响应在服务器端事务发送已被之前，以下响应必须马上进行前转：

- ✓ 除了 100 (Trying) 以外的其他临时响应
- ✓ 任何 2xx 响应

如果收到一个 6xx 响应，此响应不会被马上前转，但是此有状态代理应该取消所有客户端待处理的事务，具体描述在第 10 章，并且它一定不能创建在此 context 中创建任何新的 branches。

此修改来自于 RFC 2543，此修改强制代理马上前转 6xx 响应。对于一个 INVITE 事务来说，这个处理方式已存在一个问题，在代理需要前转 2xx 响应时，这种处理方式可能引起 2xx 响应可以抵达其他的 branch 分支。

结果是 UAC 可能收到一个 6xx 响应，随后又收到一个 2xx 响应。这种情况是不应该发生的。在新的规则中，收到一个 6xx 响应以后，代理将发送一个 CANCEL 请求，这样将会导致从所有未处理的客户端事务产生 487 响应，然后在此业务点上，6xx 前转到上游网元。

在服务器端事务中所有的最终响应被发送以后，以下响应必须被尽快处理：

- ✓ 请求的任何 2xx 响应消息

一个有状态代理一定不能马上前转其他的响应消息。具体来说，有状态代理一定不能前转任何 100 (Trying) 响应。那些响应是候选响应作为稍晚前转使用，那些响应在“Add Response to Context”的步骤中最为最佳响应会被收集到候选响应中。具体描述在“Add Response to Context”中。

任何被选为快速前转的响应必须通过一定的处理，处理的步骤在“Aggregate Authorization Header Field Values” through “Record-Route” 有描述。

此步骤结合了下一个步骤，确保一个有状态代理将正确前转一个最终响应到非 INVITE 请求，并且也确保一个非 2xx 响应或者一个和多个 2xx 响应能够正确前转到一个 INVITE 请求。

## 6. 选择最佳响应

如果没有通过以上规则马上被前转的最终响应和所有在此响应 context 中结束的客户端事务，有状态代理必须对响应 context 的服务器端事务发送一个最终响应。

这个有状态代理必须在所有已收到的和已存储在响应 context 中的响应中选择一个最佳最终响应。

如果在 context 中没有最终响应，此代理必须对服务器端事务发送一个 408 (请求超时)。

否则，代理必须从存储的 context 中前转一个响应。如果在 context 存在任何 6xx 类别的响应，它必须选择 6xx 类别的响应前转。如果在 context 中没有存储任何 6xx 的响应类别的话，此代理应该从存储的 context 中选择一个最低类别的响应前转。此代理可以从已选的类别中选择任何响应。

如果选择了 4xx 类别的响应，此代理应该给某些提供响应消息，这些响应消息会对重新发送请求有影响的响应提供一个偏好取值，例如，401, 407, 415, 420 和 484。

除非代理可以决定它代理的任何后续请求也将会生成一个 503 响应（Service Unavailable），否则，收到 503 响应的代理不应该前转其响应到上游网元中。

换句话说，前转一个 503 响应表示此代理知道它不能继续对任何请求进行处理，不仅是针对一个请求中的 Request-URI 产生了这个 503 响应。如果收到的响应仅是 503 响应，此代理应该生成一个 500 响应，然后前转到那个上游网元。

前转响应必须根据步骤“Aggregate Authorization Header Field Values” through “Record-Route” 所描述的流程进行处理。

例如，如果代理前转一个请求到 4 个地址，并且分别收到了 503, 407, 501 和 404 响应，它可以选择对 407 (Proxy Authentication Required) 响应前转。

1xx 和 2xx 响应可能涉及到 dialog 的创建。当请求中不包含 To tag 标签时，对一个正在创建请求的 dialog 来说，UAC 用响应中的 To tag 区分多个响应。如果请求中不包含一个 tag，代理一定不能在 1xx 或 2xx 响应的 To 头中插入一个 tag 标签。代理一定不能修改 1xx 或 2xx 响应的 To 头中的标签。

因为代理针对一个不包含 tag 标签的请求可以不在其 1xx 响应 To 头中插入 tag 标签，所以，代理自己不能发送一个非-100 的临时响应。但是，作为代理，它可以对一个共享同一网络网元 UAS 生成以上此请求。这个 UAS 能够返回自己的临时响应，携带请求的初始者进入到一个早期的 dialog 状态。UAS 无需一定是一个从此代理来的严密处理流程。它可以是一个虚拟的 UAS 作为代理执行在同样的代码逻辑中。

3-6xx 响应消息是通过 hop-by-hop 方式来传输的。当签发了一个 3-6xx 响应时，网络网元会马上作为一个 UAS 的角色来工作，通常基于从下游网元收到的响应来签发自己的响应消息。当对不包含 To 标签的请求执行简单前转 3-6xx 响应时，网络网元应该预留这个 To 标签。

如果前转的原始请求中没有包含 To 标签，代理一定不能在前转响应中修改 To 标签。

如果在前转 3-6xx 响应中替换了 To tag 的代理预留了初始 tag，这些 tag 来协助排查问题，这样的状态下，对代理的上游网络网元来说，它并没有什么不同。

当代理正在从几个响应中合成信息时，从这些响应中选择的 To tag 是一个任意选择的 To tag，并且生成了一个新的 To tag 的话，这样可以使得代理的 debug 排查变得更加简单一些。这样的情况可能会发生，例如，当合并 401 (Unauthorized) 和 407 (Proxy Authentication Required) 的验证或者合并从未加密的和未验证的 3xx 响应来的 Contact 值。

## 7. 合并 Authorization 头字段值

如果已选择的响应是一个 401 (Unauthorized) 或者 407 (Proxy Authentication Required)，代理必须从目前收到的所有其他 401 (Unauthorized) 和 407 (Proxy Authentication Required) 响应 context 中选择任何 WWW-Authenticate 和 Proxy-Authenticate 头值，并且，在前转前不做任何修改，在这个响应中添加这些头值。这个导致的 401 (Unauthorized) 或 407 (Proxy Authentication Required) 响应可以带由几个 WWW-Authenticate 和 Proxy-Authenticate 头字段值。

这样操作的流程是必要的，因为任何或者所有请求前转过去的目的地可以要求请求安全验证。客户端需要收到这些请求的所有验证，并且当收到请求后，对每个请求验证提供安全验证信息。进一步的处理流程，读者参考第 26 章。

## 8. Record-Route

如果已选择的响应中包含了一个开始时由此代理提供的 Record-Route 头字段值，在前转此响应前，这个代理可以选择重写这个值。这样的操作允许代理为自己对下游和上游网络网元提供不同的 URLs。代理可以因为任何路由选择此机制。例如，这种机制可以使用在多宿主机网络环境。

如果代理通过 TLS 收到收到此请求，并且通过非 TLS 连接将请求发送出去的话，此代理必须重写在 Record-Route 域中的 URL，重写为一个 SIPS URL 格式。如果代理通过 TSL 收到此请求，并且通过 TLS 连接将请求发送出去，代理必须重写在 Record-Route 域中的 URL，重写为一个 SIP URL 格式。

由此代理提供的新 URL 必须满足同样的限制，此限制来自于请求中置于 Record-Route 头中的 URLs（参考第 16.6 章节的第四步），并且需要遵守以下修改规则：

除非此代理已确认下一个上游网络网元将会在后续请求的路径中，并且此后续请求支持此传输方式，否则此 URL 不应该包含传输参数。

当代理确认决定修改响应中的 Record-Route 头时，其中的一个操作是定位这个代理已插入的 Record-Route 值。如果这个请求是一个螺旋式（spiraled）的请求，而且此代理在螺旋式请求的每个循环中已插入了 Record-Route 值，对响应中（必须是正确的循环对应相反路径）的正确值进行定位是一个难以处理的流程。以上规则推荐，试图重写 Record-Route 头值的代理要在 Record-Route 插入一个有效的比较明显的 URLs，以便可以选择一个正确的值进行重写。为了取得这个目标的一个推荐的机制是为了对 URL 的用户部分此代理实例追加一个唯一的标识。

当响应抵达后，此代理修改第一个 Record-Route（其标识匹配代理实例）。这样的修改导致一个 URL 没有数据追加到 URL 的用户部分。在下一个请求螺旋循环处理时，使用同样的算法（使用参数找到最顶部的 topmost Record-Route 头）会正确提取被代理插入的下一个 Record-Route 头字段值。

对于一个代理添加了 Record-Route header 的请求来说，不是每个请求的的响应将包含一个 Record-Route 头字段值。如果响应中没有包含 Record-Route 头字段的话，响应中将包含代理添加的值。

## 9. 前转响应

执行了所描述的过程 - "Aggregate Authorization Header Field Values" through "Record-Route" 后，代理可以执行在已选响应中执行任何功能特定的操作流程。此代理一定不能添加添加，修改或者移除此信息体。除了在第 16.7 章节的 Item 3 讨论的 Via 头字段以外，除非有具体的指定，代理一定不能移除移除任何头字段值。具体来说，代理一定不能移除任何“已收到”的参数，当处理请求相关的响应时，代理可能已经添加到了下一个头字段中的这些参数。

此代理必须传递响应消息时携带响应 context 到服务器端事务。这样会导致正在被发送到此地址的响应现已指示在最顶部的 Via 头字段中。如果服务器端事务不再是有效状态来处理传输，网络网元通过发送此响应到服务器传输实现无状态前转。服务器事务可能在它的状态机中指示发送响应失败的错误或者发送一个信号表示超时。这些错误将视为恰当的诊断错误，但是协议要求在此代理中没有重新处理媒体的流程。

从前转一个最终响应以后，直到所有代理关联的事务结束，代理都必须维持响应的 context。

## 10. Generate CANCELs

如果前转响应是一个最终响应，此代理必须为所有和此响应 context 关联的待处理的客户端事务生成一个 CANCEL 请求。当代理收到了一个 6xx 响应时，代理也应该为和此响应 context 关联的所有待处理客户端事务生成一个 CANCEL 请求。待处理的客户端事务表示它收到了一个临时响应，但是没有收到最终响应（此事务在处理状态中），并且还没有一个和此事务关联的 CANCEL 生成。关于生成 CANCEL 请求的流程在第 9.1 章节有具体介绍。

对于基于最终响应的待处理客户端事务的 CANCEL 的要求来说，这个要求并不能保证对一个 INVITE 来说，终端将不会收到多个 200 (OK) 响应。在 CANCEL 请求被发送和处理之前，在多于一个 branch 的状态下可能生成多个 200 (OK) 响应。进一步来说，希望将来的扩展中涵盖这个要求来支持发送 CANCEL 请求。

### 16.8 Processing Timer C

如果触发了定时器 C，此代理必须使用任何其代理选择的定时器值来重置定时器，或者结束客户端事务。如果客户端事务已收到了一个临时响应，此代理必须生成一个 CANCEL 请求来匹配这个事务。如果客户端事务还没有收到一个临时响应，此代理必须按照此事务收到一个 408 响应（请求超时）一样的流程来进行处理。

当定时器触发后，允许代理重置定时器的话实际上就允许此代理在当前环境中动态扩展事务的生命周期。

#### 16.9 Handling Transport Errors

当代理前转一个请求时（参考第 18.4 章节），如果传输层提示一个代理错误，此代理的处理方式必须像前转请求收到一个 503 响应那样去处理。

当前转响应时，如果此代理收到了一个错误提示，代理要丢弃此响应。因为这个提示，代理不应该取消任何和此响应 context 关联的未完成的客户端事务。

如果代理取消了它的未完成的客户端事务，通过其 Via 头字段值，单个恶意的或者处理异常的客户端能够引起所有事务失败。

#### 16.10 CANCEL Processing

一个有状态代理可以对由此代理已生成的任何其他请求生成一个 CANCEL（取决于那个请求收到的一个临时响应，具体描述在第 9.1 章节）。当代理收到一个匹配的 CANCEL 请求时，此代理必须取消任何和响应内容绑定的待处理的客户端事务。

一个有状态的代理可以对待处理的 INVITE 客户端事务生成 CANCEL 请求，此待处理的 INVITE 客户端事务基于在 INVITE 的超时头值指定的时间段。不过，一般来说，这种处理方式是没有必要的，因为，此处理流程中所涉及的终端将会使用信令处理事务结束。

当一个 CANCEL 请求在有状态代理中由此代理自己的服务器端事务来负责处理时，不会为此请求创建一个新的响应内容。相反，代理层会为此服务器端事务中正在处理的请求所关联的 CANCEL 内容查询它自己现存响应内容。如果匹配了响应的内容，网络网元必须对 CANCEL 马上返回一个 200 (OK)。在这种情况下，网元被看作是一个用

户代理服务器，具体定义参考第 8.2 章节。另外，网络网元必须为所有在此内容中的待处理客户端事务生成一个 CANCEL 请求，具体定义参考 第 16.7 章节中的第十步。

如果没有找到响应内容，网元产生任何请求确认，也不会应用到此 CANCEL 请求中。代理必须对此 CANCEL 请求执行无状态前转（代理可能也会对其他以前关联的请求执行无状态前转）。

### 16.11 Stateless Proxy

当作为一个无状态代理时，代理是一个简单的消息转发者。当作为一个无状态代理时，很多的处理流程和有状态代理的处理流程相同。这里说明它们之间的不同点。

无状态代理不包含任何用来描述有状态代理行为的事务或响应内容的概念。相反，此无状态代理将之间从传输层获取请求和响应的消息（参考第 18 章节）。因此，无状态代理不会重传自己的消息。它们却会前转所有它们收到的消息（它们也没有能力从原始消息中区分重传消息）。进一步来说，当代理正在无状态处理一个请求时，网络网元一定不能生成它自己的 100 (Trying) 或其他临时响应消息。

无状态代理一定要通过第 16.3 章节的描述来验证请求。

一个无状态代理必须遵从第 16.4 到 16.5 的规范结合以下特例来处理请求：

无状态代理必须从目标组中选择一个或仅选择一个目标目的地。这个选择必须依赖于消息体中参数和服务器端时间恒定的属性参数。具体来说，代理处理时，重传请求必须被前转到同样的目的地地址。进一步说，CANCEL 和未经过路由的 ACK 请求必须生成同样的选择设置，其选择设置就像它们所关联的 INVITE 请求一样。

除了以下特例以外，一个无状态代理必须按照第 16.6 章节所描述的请求处理流程来进行处理：

- 对于唯一的 branch IDs 来说，要求此 IDs 贯穿整个处理流程空间和无状态代理所执行的时间范围内。但是，就像在第 16.6 bullet 8 中所描述的那样，一个无状态代理不能简单使用一个任意号码生成流程来计算 branch ID 的第一个组件。这是因为一个请求的重传需要同样的值，并且无状态代理不能从原始请求通知重传。因此，branch 参数的组件使得变成唯一标识，这样保证每次重传请求前转必须执行实现

的属性。对于无状态代理来说，branch 参数必须按照消息参数功能的组合来计算，这些信息参数在重传中是恒定的参数。

- 无状态代理可以使用任何它愿意使用的技术在涵盖事务处理的过程中保证它的 branch IDs 的唯一性。但是，以下流程是推荐的技术流程。代理从收到的请求的 Via 头中的最顶部的 Via 头中检查 branch ID。如果 branch ID 开始是以 magic cookie 开始，发出的请求 branch ID 的第一个模块是经过计算的，作为收到的 branch ID 的一个哈希值。否则，branch ID 的第一个模块经过计算作为最顶部 Via 头的哈希值，To 头中的 tag 标签，From 头中的 tag 标签，Call-ID 头，CSeq 号（但是是非 method），和收到的请求中的 Request-URI。以上这些头中的其中之一经过两个不同的事务时总是要发生变化。
- 所有其他在第 16.6 章节中指定的消息转换必须和重传中的消息转换相同。具体来说，如果代理在 Route 头中插入一个 Record-Route 值或者推送了 URLs 的话，代理必须把这些同样的值置入到重传请求中。就 Via branch 参数来说，这种处理方式应用于消息转换必须是基于时间恒定的配置环境或者请求中的重传恒定的变量属性的配置环境中。
- 就像第 16.6 第十条描述的应用，一个无状态代理决定请求前转的方向。请求会被直接发送到传输层而不是通过一个客户端事务来测试。
- 因为无状态代理必须重传请求到同样目的地地址，并且对这些请求的每个重传请求添加同样的 branch 参数值，代理仅自己能使用消息中的信息和时间恒定的配置数据支持请求的计算。如果配置状态不是时间恒定的（例如，路由表是更新的），在周期设置等于事务超时窗口之前或者周期设置等于事务超时窗口之后，任何因为变化而受影响的请求可能不被无状态地进行前转。在此周期期间，这种影响请求的处理方式是一个部署的决定。一般的解决办法是对这些请求进行有状态转发。
- 无状态代理一定不能执行针对 CANCEL 请求的特别处理。CANCEL 请求的处理方式使用以上规则进行处理，和其他请求相同。具体来说，无状态代理对 CANCEL 请求应用了和其他请求一样的处理流程。

响应处理流程就像在第 16.7 章节所描述的，它不应用在代理无状态的表现中。当响应抵达到一个无状态的响应时，此代理必须在最顶部的 Via 头字段中插入这个 sent-by 值。如果那个地址不能匹配此代理的话，（它等于一个此代理已经在前面的请求中插入的值），此代理必须从响应中移除那个头字段值，并且前转这个结果到这个地址，这个地址在下一个 Via 头字段中指示。代理一定不能对添加、修改，或者移除消息体。除非有特别指定，否则此代理一定不能移除任何其他的头字段值。如果此地址不能匹配这个代理，此信息一定要被通过静静地丢弃方式处理（非规范说明，建议参考 RFC2865 关于关键词说明）。

### **16.12 Summary of Proxy Route Processing**

相反地，在缺少本地策略设置的环境中，处理代理流程时，对包含 Route 头字段的请求的处理时可以总结为以下几个步骤。

1. 代理将检查 Request-URI。如果它指示了一个属于代理自己的资源，代理将使用正在运行的定位服务结果替换这个资源。否则，代理将不会修改这个 Request-URI。
2. 代理将会检查在最顶端路由头字段中的 URL 值。如果此 URL 值指示了此代理，代理将会从路由头字段中移除这个 URL 值（已经抵达此路由节点）。
3. 代理将会前转此请求到一个资源，这个资源是在最顶部路由头字段中表示的 URL 或者是 Request-URI 中的资源地址（如果路由头没有出现的话）。当前转请求，对 URL 使用了[参考链接 4]的流程时，代理决定资源的地址，端口和传输方式。

如果在请求路径上没有遇到严格路由 (strict-routing) Request-URI 将总是表示请求的目的地地址。

#### **16.12.1 Examples**

##### **16.12.1.1 Basic SIP Trapezoid**

此流程示例是一个基本的 SIP 呼叫图例，U1->P1->P2 ->U2，都支持了代理 record-routing。以下是呼叫流程。

U1 发送：

```
INVITE sip:callee@domain.com SIP/2.0  
Contact: sip:caller@u1.example.com
```

到 P1。P1 是一个呼出代理。P1 不对域 domain.com 负责，因此，它将查询 DNS，发送请求到此域。它同时添加一个 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 获得以上消息以后。它负责域 domain.com，所以它运行定位查询服务，重写这个 Request-URI。它也添加了一个 Record-Route 头字段值。这里没有 Route 头字段，因此它解析这个新的 Request-URI 来决定往哪里发送请求的目的地：

```
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 域的被呼叫方获得以上消息以后，返回响应消息，携带 200 OK 响应返回：

```
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 的被呼叫方也设置了它的 dialog 状态的远端目的地 URI，设置此 URL 为 sip:caller@u1.example.com，并且设置了其路由表（route set）到：

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

正常情况下，以上路由通过 P2 前转到 P1，然后到 U1。现在，U1 设置了其 dialog 状态远端目的地 URL 到此地址 `sip:callee@u2.domain.com`，并且设置路由表为：

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

因为所有这些路由表网元中包含了 lr 参数，U1 创建以下 BYR 请求：

```
BYE sip:callee@u2.domain.com SIP/2.0
Route: <sip:p1.example.com;lr>,<sip:p2.domain.com;lr>
```

当任何其他网元（包括代理）进行处理时，它使用 DNS 解析路由头中最顶部的 URL 地址，然后决定请求发送目的地。这样就会发送到 P1。P1 注意到它不负责在 Request-URI 的资源地址，所以，不会修改此地址。它将会看到它自己是路由头字段中的第一个地址，它会移除那个值，然后前转此请求到 P2：

```
BYE sip:callee@u2.domain.com SIP/2.0
Route: <sip:p2.domain.com;lr>
```

P2 也注意到，它不会对由 Request-URI 指示的资源负责（它对 domain.com 负责，不会对 u2.domain.com），因此，它不会修改这个资源地址。它确实看到自己是路由头字段中的第一个值，因此，它会移除自己的值，通过 DNS 查询 Request-URI，然后前转到以下地址 u2.domain.com：

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

#### 16.12.1.2 Traversing a Strict-Routing Proxy

在这个流程中，创建一个 dialog 穿越了 4 个代理，每个代理会添加 Record-Route 头字段值。第三个代理应用了严格路由处理流程，此处理流程在 RFC 2543 规范中定义，很多工作在处理中。

U1->P1->P2->P3->P4->U2

抵达 U2 的请求包含：

```
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>
```

U2 将对此请求返回响应，携带一个 200 OK。稍晚后，U2 根据第一个 Route 头字段值发送以下 BYE 请求到 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 不负责在 Request-URI 表示的资源地址，因此，P4 将单独保留此资源地址。P4 注意到自己是一个网络网元，并且出现在了第一个路由头值中，因此，自己需要被移除。然后，P4 准备发送此请求，发送请求是基于 sip:p3.middle.com 的第一个路由头字段值地址。但是，它注意到，这个 URL 不包含 lr 参数，在发送请求前，P4 重新进行格式化处理，重新格式化后的请求是这样的：

```
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>
```

P3 是一个严格路由方式，因此，它前转以下消息到 P2:

```
BYE sip:p2.example.com;lr SIP/2.0
Route: <sip:p1.example.com;lr>
Route: <sip:caller@u1.example.com>
```

P2 看到了在 request-URI 的值，自己被置于 Record-Route 头字段中。因此，P2 在做进一步处理之前，P2 重写此请求，重新格式化后的请求是这样的：

```
BYE sip:caller@u1.example.com SIP/2.0
Route: <sip:p1.example.com;lr>
```

P2 不会对此 u1.example.com 资源地址负责，因此，它会对 P1 发送此请求，发送请求的地址是基于路由头字段中解析的结果。

P1 注意到自己是最顶部的 Route 路由头字段值，因此，它因此自己的地址，最后变成以下格式继续进行处理：

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

因为 P1 不对 u1.example.com 负责，并且也没有任何 Route 头字段值，P1 将会根据 Request-URI 前转此请求到 u1.example.com 地址。

#### 16.12.1.3 Rewriting Record-Route Header Field Values

在这个场景中，U1 和 U2 在不同的私有命名空间，并且它们通过一个代理 P1 进入到了一个 dialog 中，代理作为一个网关在命名空间之间工作。

U1->P1->U2

U1 发送：

```
INVITE sip:callee@gateway.leftprivatespace.com SIP/2.0
Contact: <sip:caller@u1.leftprivatespace.com>
```

P1 使用了其定位服务，并且发送以下消息到 U2：

```
INVITE sip:callee@rightprivatespace.com SIP/2.0
Contact: <sip:caller@u1.leftprivatespace.com>
Record-Route: <sip:gateway.rightprivatespace.com;lr>
```

U2 发送此 200 (OK) 返回到 P1：

```
SIP/2.0 200 OK
Contact: <sip:callee@u2.rightprivatespace.com>
Record-Route: <sip:gateway.rightprivatespace.com;lr>
```

P1 重写它的 Record-Route 头字段参数提供一个值，U1 发现这个值是有用的值，并且发送以下消息到 U1：

```
SIP/2.0 200 OK
Contact: <sip:callee@u2.rightprivatespace.com>
Record-Route: <sip:gateway.leftprivatespace.com;lr>
```

稍晚，U1 发送以下 BYE 请求到 P1：

```
BYE sip:callee@u2.rightprivatespace.com SIP/2.0
Route: <sip:gateway.leftprivatespace.com;lr>
```

P1 前转消息到 U2，就像以下消息：

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

## 17 Transactions

SIP 是一种事务协议：介于模块构件之间的交互发生在一系列的互相独立的消息交互过程中。具体来说，一个 SIP 事务由一个单请求和任何针对此请求的响应构成，这些响应包含零个或者多个临时响应，和一个或者多个最终响应。在这种事务示例中，请求是一个 INVITE（称之为一个 INVITE 事务），此事务中也包含此 ACK（只有如果最终响应不是一个 2xx 响应时）。如果响应是一个 2xx 响应，此 ACK 不会被看作是此事务的一个部分。

这样分开的根本原因在于针对一个 INVITE 到 UAC 传输所有 200 (OK) 响应的重要性。为了确保所有的响应能够传输到 UAC，UAS 需要各自承担自己的责任，保证能够重传其响应。参考第 13.3.1.4 章节），并且 UAC 各自承担自己的责任，通过 ACK 获得确认（参考第 13.2.2.4 章节）。因为此 ACK 仅通过 UAC 重传，因此，它实际上被看作是有自己的事务。

事务有一个客户端和一个服务器端。客户端被称之为一个客户事务，服务器端被称之为服务器端事务。客户端发送请求，服务器端发送响应。客户端和服务器端都是一个逻辑功能，此逻辑功能植入可以植入到任何数量的网元中。具体来说，它们存在于用户代理和有状态 proxy 服务器。回顾一下在第 4 章节中的例子，在这个例子中，UAC 执行一个客户端事务，它的出局 outbound proxy 执行服务器端事务。这个 outbound proxy 也执行一个客户端事务，在 inbound proxy 中，通过客户端事务发送请求到服务器端事务。那个 proxy 也执行一个客户端事务，客户端事务按照顺序在 UAS 中发送请求到服务器端事务中。以下是 Figure 4 示例。

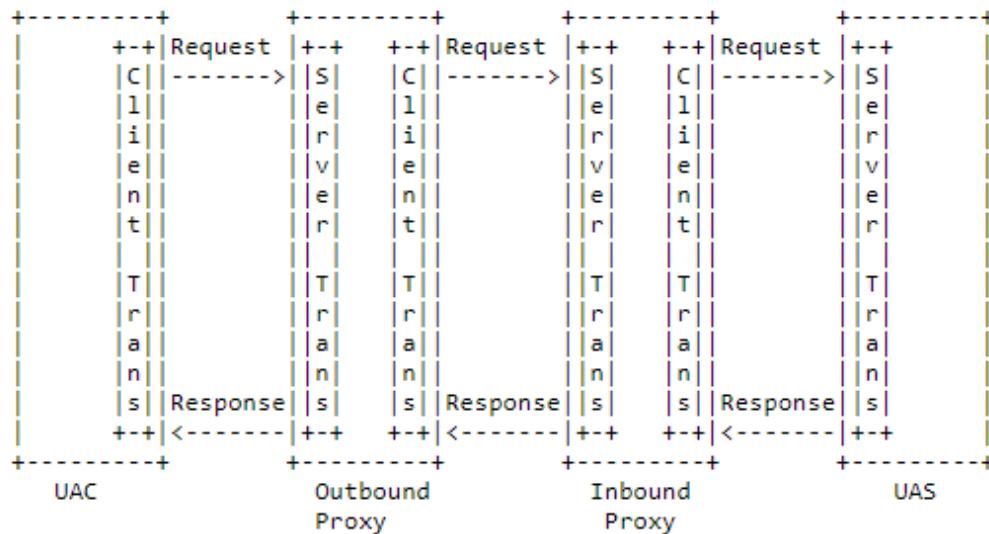


Figure 4: Transaction relationships

一个无状态 proxy 不包含客户端或者服务器端的事务。事务存在于一侧的 UA 或者有状态 proxy，和另外一侧的 UA 或者有状态 proxy。只要发起 SIP 事务，无状态代理实际上已经非常清楚。客户端事务的主要目的是接收从植入到客户端网元来的请求（我们称此网元为 "Transaction User" 或者 TU；它可以是一个 UA 或有状态 proxy），并且为此请求提供可靠传输到服务器端事务。

客户端事务同时也负责接收响应消息，然后传输这些响应消息到 TU，过滤任何重传或者未经允许的响应（例如，对 ACK 的响应）。另外，在 INVITE 请求使用中，客户端事务负责生成 ACK 请求的任何最终响应（除了 2xx 响应）。

同样的道理，服务器端事务负责接收从传输层来的请求，并且传输这些请求到 TU 侧。服务器端事务过滤任何网络重传请求。服务器端事务接受从 TU 来的响应，并且传输这些响应到传输层支持通过网络传输。在请求事务使用中，除了 2xx 响应以外，服务器端事务会吸纳针对任何最终响应的 ACK 请求。

2xx 响应和它接收的 ACK 是一种特殊的处理。这个响应仅被 UAS 重传，并且仅由 UAC 生成其 ACK。这种端对端的处理方式是需要的，以便呼叫方获知已接收呼叫方完整的用户信息。因为这种特殊的处理方式，2xx 响应的重传是由 UA core 来负责处理，而不是事务层负责处理。同样的，针对 2xx 的 ACK 生成也是由 UA core 负责处理。每个传输流程路径的 proxy 仅前转每个 2xx 响应到 INVITE，和其相应的 ACK 响应消息。

## 17.1 Client Transaction

客户端事务的功能是通过状态机维护来实现。

TU 和客户端事务的通信是通过一个简单的接口来实现。当 TU 希望发起一个新的事务时，它创建一个新的客户端事务，传递客户端事务一个 SIP 请求，还有 IP 地址，端口和传输方式。客户端事务开始状态机执行。有效响应从客户端事务发送到 TU 侧。

有两种不同的客户端事务状态机类型，两种方式取决于 TU 传输的请求方式。一种是处理 INVITE 请求的状态客户端事务状态机。这种类型的状态机被视为 INVITE 客户端事务状态机。另外一种是所有请求的状态机，除了 INVITE 和 ACK methods。这一类被视为非 INVITE 客户端事务状态机。没有支持 ACK 的客户端状态机。如果 TU 希望发送一个 ACK 的话，TU 直接对传输层传递 ACK 来实现传输。

因为 INVITE method 拓展时间处理，INVITE 事务和其他的 methods 的事务状态机不同。通常情况下，呼叫用户输入其要求的结果来满足其 INVITE 要求。执行的稍长延迟来发送一个三次握手的流程。另外一方面，其他 methods 的请求希望快速完成。因为，非 INVITE 事务的信任机制是基于两次握手决定的，因此，TU 应该马上响应非 INVITE 请求。

### 17.1.1 INVITE Client Transaction

#### 17.1.1.1 Overview of INVITE Transaction

INVITE 事务由一个三次握手构成。客户端事务发送一个 INVITE，服务器端事务发送一个响应，客户端事务发送一个 ACK。对于不可靠传输，例如使用 UDP，客户端事务在一定时间周期内重传请求，这个周期定时器以 T1 秒开始启动，每个重传后翻倍计算这个时间。T1 是一个 RTT (round-trip time) 的估值，其默认值为 500ms。这里介绍的几乎所有的事务定时器取值都和 T1 相关。通过 T1 来调整它们的值。请求不会通过可靠传输重传。收到一个 1xx 响应以后，任何重传都会一起退出，客户端等待进一步的响应。服务器端事务不能进行可靠传输的话，服务器端事务能够增加其余的 1xx 响应来支持。最终，服务器端事务来决定发送一个最终响应。对于非可靠传输，响应是通过周期性重传来实现，对于可靠传输来说，其响应被发送一次。对于每个客户端事务收到的最终响应，客户端事务发送一个 ACK，其目的是处理响应所针对的重传停止流程。

#### 17.1.1.2 Formal Description

Figure 5 显示了 INVITE 客户端事务的状态机构成。当 TU 创建了一个新的客户端事务，发起 INVITE 时，初始状态—"calling"必须是已进入状态。客户端事务必须传递这个请求到传输层来进行传输（参考第 18 章）。如果使用的是非可靠性传输，客户端事务必须启用 A 定时器，并且使用 T1 的值来计算。如果使用的是可靠性传输，客户端事务应该不能启动 A 定时器。A 定时器控制请求重传。对于任何传输来说，客户端事务必须启动 B 定时器，计时使用  $64*T1$  秒。定时器 B 控制事务超时。

当定时器 A 被触发以后，客户端事务必须重传此请求，传递请求到传输层，并且，必须重置 A 定时器，使用的计算值为  $2*T1$ 。

在传输层内容中的重传规范定义是通过提取前面发送到传输层的消息，然后再次发送到传输层。

当定时器 A 触发了  $2*T1$  秒后，此请求必须被重传（假设客户端事务仍然在此状态）。此处理流程必须继续进行，以便请求在一定周期内被重传，每个重传需要翻倍计算周期时间。这些重传应该仅在客户端事务为"calling"状态时完成。

$T1$  的默认值为 500 毫秒。 $T1$  是介于客户端和服务器端事务之间的 RTT 的预估值。在封闭网络网元可以更小的  $T1$  值（不推荐），封闭网络或者私有网络不允许进行外网访问。 $T1$  可以现在比较大的数值，如果提前知道有路径延迟（RTT 的值比较大的话）， $T1$  选择比较大的值也是推荐的一种方式。无论  $T1$  值怎么样设置，在重传中的指数补偿必须使用本章节说描述的方式进行处理。

当定时器 B 触发时，如果客户端事务仍然在"Calling"状态的话，客户端事务应该通知此 TU，超时已发生。此客户端一定不能生成 ACK。 $64*T1$  的值等于在非可靠传输环境中发送七次请求的时间总和。

如果客户端事务在"Calling"状态时收到一个临时响应，它会切换到"Proceeding"状态。在"Proceeding"状态，客户端事务不应该继续重传此请求。进一步说，当客户端事务在"Proceeding"状态时，临时响应必须被传递到 TU。

当在"Calling"或者"Proceeding"状态，带状态响应码（300-699）的响应接收方必须要求客户端事务切换为"Completed"状态。客户端事务必须传递接收到的响应到 TU。即时传输是可靠传输（关于构建 ACK 的指导参考-第 17.1.1.3 章节），客户端事务必须生成一个 ACK 请求，并且传递此 ACK 到传输层来实现传输。必须根据初始请求发送的地址，端口，传输方式，此 ACK 也必须被发送到同样的地址，端口和传输方式。当客户端事务进入到"Completed"状态时，客户端事务应该启动定时器 D，定时器值设置为至少 32 秒来支持非可靠传输（例如 UDP），设置零秒来支持可靠传输（例如 TCP）。

定时器 D 反应一个总时间，此时间是当使用了非可靠传输时，服务器端事务在 "Completed" 状态保持的时间。此时间等于 INVITE 服务器事务中的定时器 H 的时间，定时器 H 的默认时间是  $64 \times T_1$ 。但是，客户端事务不知道服务器端事务中使用的  $T_1$  时间值，因此，使用绝对最小值 32s 秒，而不是依赖于  $T_1$  中的定时器 D 的时间值。

当在 "Completed" 状态时，客户端所谓收到的任何最终响应的重传必须要求此 ACK 重传递到传输层来支持重传，但是，新收到的响应一定不能传递到 TU。此响应的一个重传定义为任何响应，它将匹配同样的客户端事务，匹配规则基于第 17.1.3 章节所定义的规则。

www.sip.org.cn

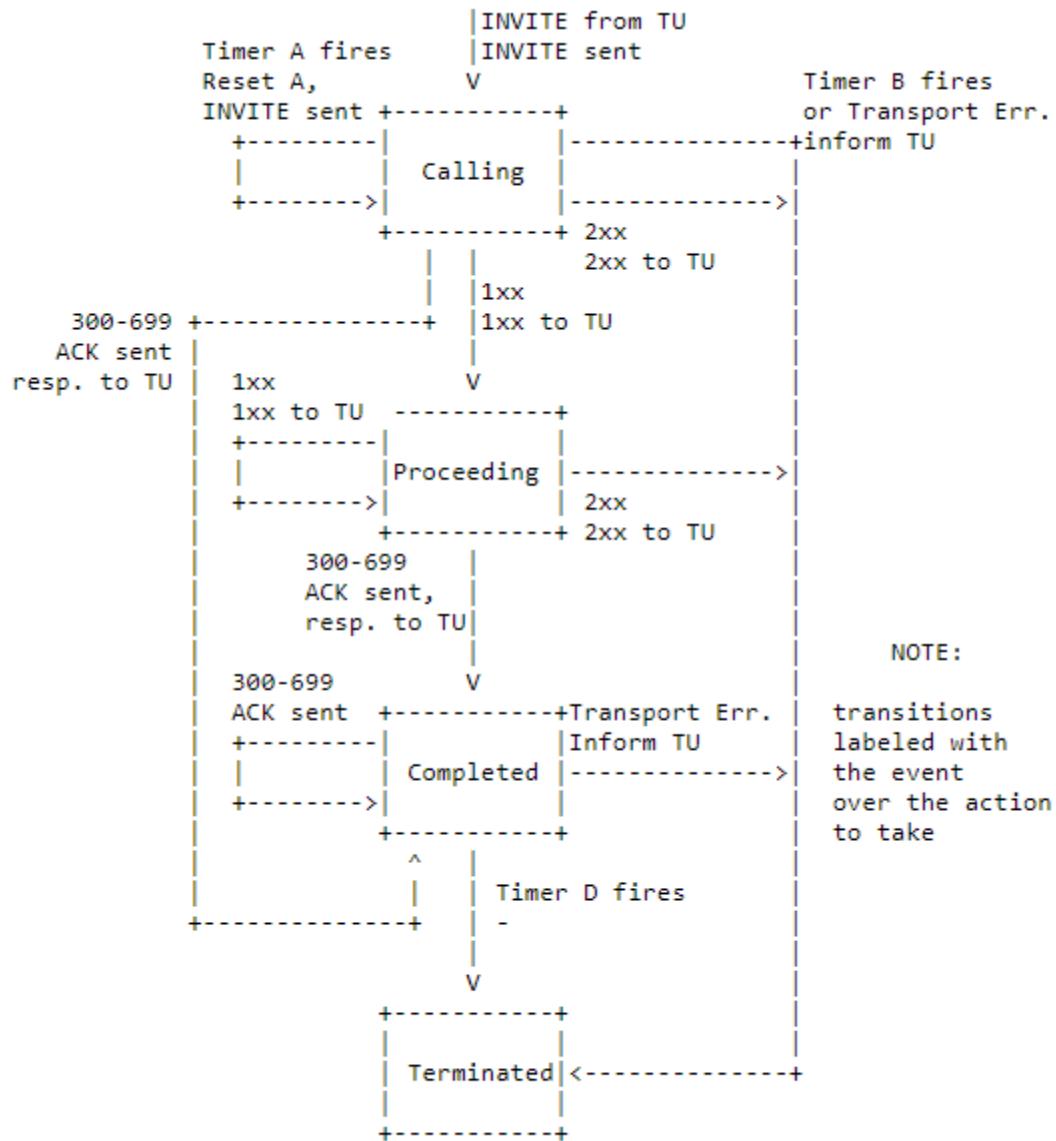


Figure 5: INVITE client transaction

当客户端事务在"Completed"状态时，如果定时器 D 被触发，客户端事务必须迁移到结束状态。

当客户端事务在"Calling" 或者 "Proceeding"状态时，2xx 响应的接收方必须发起一个客户端事务进入到"Terminated"状态，响应也必须传递到 TU。此响应的处理依赖于是否 TU 是一个 proxy core 还是一个 UAC core。

UAC core 将处理此响应的 ACK 生成，而 proxy core 将会前转此 200 (OK) 到上游节点。介于 proxy 和 UAC 之间的 200 (OK) 的区别处理上有原因的，因为 200 (OK) 的处理没有发生在事务层。

客户端事务必须销毁进入到"Terminated"状态的实例。实际上这也是非常必要的，以便保证客户端事务正确运行。其原因是，一个 INVITE 的 2xx 响应是不同的。每个 2xx 通过 proxies 进行了前转，并且在 UAC 中 ACK 处理也是不同的。因此，每个 2xx 都需要传递到一个 proxy core (这样，proxy core 才能进行转发处理)，并且也需要每个 2xx 传递到一个 UAC core (这样才能让 UAC core 确认状态)，无事务层处理发生。但是，如果传输层收到一个响应，如果传输层没有发现任何可匹配的客户端事务（使用规则第 17.1.3 章），此响应会直接传递到 core。因为第一个 2xx 就会销毁匹配客户端事务，后续的 2xx 将不会发现匹配，因此会传递到 core 中。

#### 17.1.1.3 Construction of the ACK Request

此部分内容详解在客户端事务中 ACK 请求的构建。为 2xx 生成 ACK 的 UAC core 必须根据第 13 章所描述的规则进行处理。

由客户端事务创建的 ACK 请求必须包含 Call-ID, From, 和 Request-URI 的值，这些值等于请求中的头字段值，此请求是通过客户端事务传输层的（称之为“初始请求”）。ACK 中的 To header 必须等于被确认的响应中的 To，因此，可以通过其他的 tag 标签传输区别初始请求中的 To header。此 ACK 必须包含一个单个 Via header，并且，此 Via header 必须等于初始请求中的 top Via 头。在 ACK 中的 CSeq header 必须包含同样的序列号值，就像出现在初始请求的值，但是 method 参数必须等于"ACK"。

如果一个已被确认的 INVITE 请求，请求中已有 Route header 值，这些 Route fields 必须出现在 ACK 中。这样做的目的是确保此 ACK 可以通过下游无状态代理实现正确路由处理。

尽管任何请求都可能包含一个消息体，因为如果消息体不能被解析或者理解，此请求也不能被拒绝，因此在 ACK 中的消息体是非常特别的。因此，对于非 2xx 来说，替换 ACK 中

的消息体是不推荐的，但是，如果想替换的话，假设一个 INVITE 的响应不是 415，消息体类型可以通过出现在 INVITE 中加以限定。如果是 415 的话，ACK 中的消息体可以包含任何列表类型，这个列表类型在响应 415 的 Accept header 列出。

例如如下请求示例：

```
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
```

针对此请求，ACK 请求对非-2xx 的最后响应就像这样：

```
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 Non-INVITE Client Transaction

### 17.1.2.1 Overview of the non-INVITE Transaction

非 INVITE 事务涉及 ACK 使用。它们是简单的请求-响应交互流程。对于非可靠传输来说，请求通过一定的周期进行重传，周期以 T1 开始计算，然后翻倍计算，直到 T1 达到 T2 定时器值。如果收到一个临时响应，对非可靠传输来说，重传仍然继续，但是周期是以 T2 来计算。仅当收到请求重传时，服务器端事务重传的最后响应（服务器端事务发送的）可以为一个临时或者最终响应。这也是为什么尽管是临时响应后，请求重传需要继续执行，它们可以确保最终响应的可靠性传递。

不像 INVITE 事务，一个非 INVITE 事务无任何针对 2xx 响应的特别处理。其结果是对一个非 INVITE 来说，仅单个 2xx 响应不断传输到 UAC 端。

#### 17.1.2.2 Formal Description

非 INVITE 客户端事务的状态机在 Figure 6 中显示。其处理流程和 INVITE 的状态机非常相似。

当 TU 初始化一个新的客户端事务，这个客户端事务是一个请求时，客户端事务会进入到一个 "Trying" 状态。当进入到此状态以后，此客户端事务应该在  $64*T1$  时间内设置定时器 F 等待触发。此请求必须被传递到传输层进行传输。如果使用的是非可靠传输，客户端事务必须是在定时器 E，定时器 E 在  $T1$  秒内触发。如果定时器 E 在此状态触发，定时器重新设置，但是时间设置为  $\text{MIN}(2*T1, T2)$  的最小值。当定时器再次触发后，定时器重新设置为  $\text{MIN}(4*T1, T2)$ 。这个过程会继续执行支持重传发生，其发生周期是通过指数级增加的周期来计算，计算以  $T2$  为基准。如果没有马上响应的话，默认的  $T2$  的值是 4s，它代表非 INVITE 服务器端事务对请求生成响应所耗费的时间。对于默认的  $T1$  和  $T2$  值来说，这样的处理方式会导致定时器是 500ms, 1 s, 2 s, 4 s, 4 s, 4 s, 等等。

当客户端事务在 "Trying" 状态时，如果定时器 F 被触发，客户端事务应该通知 TU 涉及的超时，然后客户端事务进入到 "Terminated" 状态。当客户端事务在 "Trying" 状态时，如果收到一个临时响应，此响应必须要被传递到 TU，然后，客户端事务应该被转移到 "Proceeding" 状态。当客户端事务在 "Trying" 状态时，它收到一个最终响应（状态码是 200-699），此响应必须被传递到 TU，并且，客户端事务必须切换到 "Completed" 状态。

如果在 "Proceeding" 状态时，定时器 Timer E 被触发，此请求必须传递到传输层进行重传处理，并且，定时器 E 必须重新设置，使用  $T2$  值进行重置。如果客户端事务在 "Proceeding" 状态时，如果定时器 F 被触发，必须通知 TU 超时。客户端事务必须强化到结束状态。如果在 "Proceeding" 状态收到一个最终响应，状态代码为 200-699，响应必须传递到 TU 侧，客户端事务必须强化到 "Completed" 状态。

一旦客户端事务进入到"Completed"状态，针对非可靠传输，它必须在 T4 秒内设置定时器 K，针对可靠传输设置为零秒。"Completed"状态存在的目的是缓存其他可能收到的响应重传（这也是为什么客户端事务仅为非可靠传输进行保持处理）。

T4 定时器表示网络清理客户端和服务器端之间事务信息所需时间。默认的 T4 值为 5s。当一个响应匹配了同一事务时，此响应是一个重传响应，具体判断规则详解参考第 17.1.3 章节。当事务在"Completed"状态，如果定时器 K 被触发，客户端事务必须切换到"Terminated"状态。

一旦事务在结束状态，它必须被马上销毁。

### 17.1.3 Matching Responses to Client Transactions

当客户端的传输层收到一个响应，传输层需要决定哪个客户端事务来处理此响应，这样可以进入到第 17.1.1 和第 17.1.2 章节所描述的处理流程。其中，在 top Via 的头中的 branch 参数可以用来帮助进入到处理流程。响应消息匹配客户端事务需要满足两个条件：

1. 如果此响应在 top Via 头中有同样的 branch 参数值，这个值和创建了此事务的请求中 top Via 头中的 branch 参数一样。
2. 如果 CSeg 头中的 method 参数匹配了创建此事务的请求中的 matches 参数。因为一个 CANCEL 请求构成不同的事务，但是可以共享同一 branch 参数，因此，需要 method 来支持。

如果请求是通过多播方式发送的，可能会从不同的服务器生成不同的响应消息。这些响应都会在最顶端的 Via 中带一个同样的 branch 参数值，但是在 To 标签中可能略有不同。基于以上条件收到第一个响应，将使用此响应做其他进一步处理，其余响应被视为重传响应，这不是真正的错误；多播 SIP 仅提供一个基本的"single-hop-discovery-like"服务，这种查询服务提供一种单跳查询，通过地址，DNS 等相关消息查询最近服务支持，这种服务仅支持单个响应。具体细节通过第 18.1.1 章节查看。

### 17.1.4 Handling Transport Errors

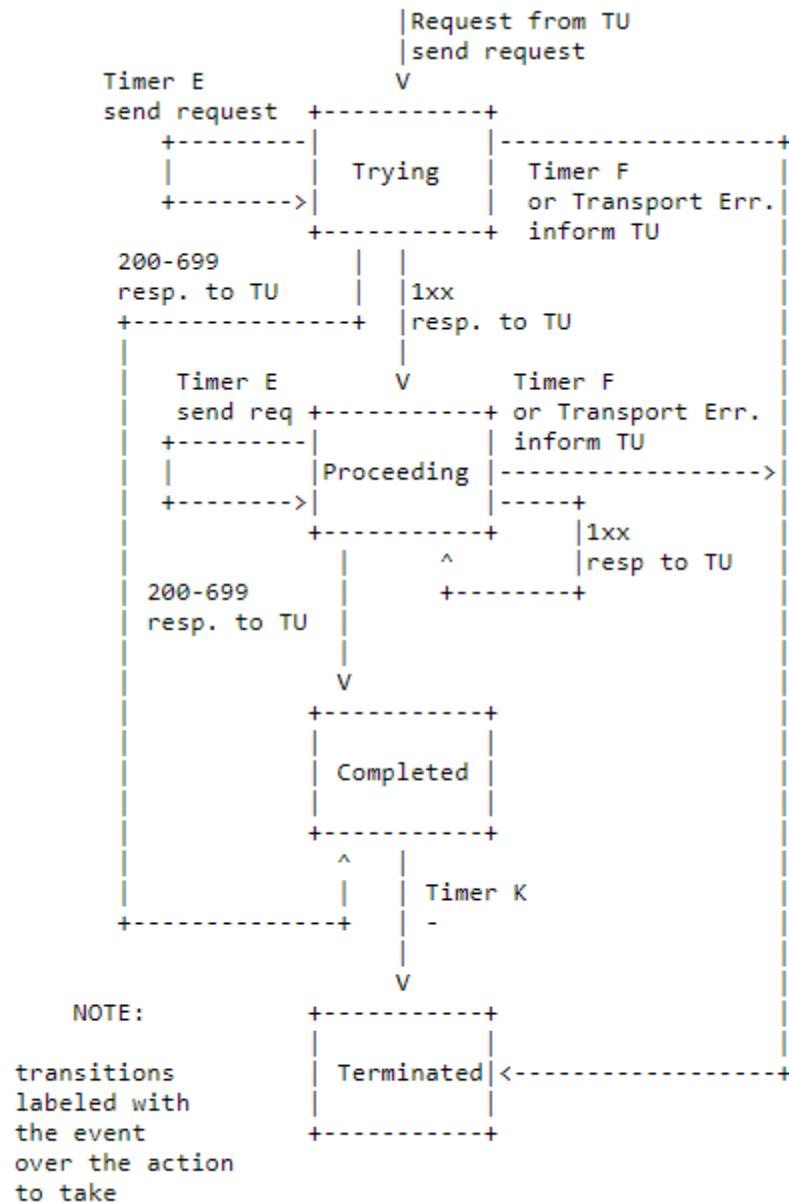


Figure 6: non-INVITE client transaction

当客户端事务对传输层发送一个请求，如果传输层表示一个失败状态时，必须根据以下规则来进行处理。

客户端事务应该通知 TU，已经发生传输失败，并且，客户端事务应该直接切换到“Terminated”状态。TU 将通过失效机制来对后续状态进行处理。关于失效机制的处理过程参考[参考链接 4]。

## 17.2 Server Transaction

服务器端事务负责把请求发送到 TU，同时负责响应的可靠性传输。它通过状态机来实现传输。当收到客户端请求后，对此请求的事务处理准备好以后，core 创建一个服务器端事务。

和客户端事务一样，状态机取决于收到的请求是否是一个 INVITE 请求。

### 17.2.1 INVITE Server Transaction

针对 INVITE 服务器端事务的状态图例在 Figure 7 中展示。

当构建一个服务器端事务来支持请求时，服务器端事务会进入到“Proceeding”状态。此服务器端事务必须生成一个 100 (Trying) 响应，除非它知道在 200 毫秒内，TU 将会生成一个临时响应或最终响应（在一些场景中它可能生成一个 100 (trying)）。临时响应也是需要的，它可以快速缓解请求重传以避免网络拥塞。除了在协议的 To 头中插入标签从“可以”被降级到“不应该”以外（无任何头出现在请求中），100 (Trying) 响应的构建根据处理第 8.2.6 章节的流程来实现。此请求必须传递到此 TU 端。

此 TU 传递任意数量的临时响应到服务器端事务。只要服务器端事务进入到“Proceeding”状态，每个临时响应必须被传递到传输层进行传输。临时响应不通过事务层进行可靠传输（临时响应不会通过事务层进行重传），临时响应不会引起服务器端事务的状态修改。当服务器端事务在“Proceeding”状态时，它收到一个请求重传的话，它从 TU 收到的最近的临时响应必须被传递到传输层来进行重传。如果请求匹配了同样的服务器端事务，这个匹配规则是根据第 17.2.3 章节来执行的话，这个请求是一个重传请求。

如果服务器端事务在"Proceeding"状态时， TU 传递一个 2xx 响应到服务器端事务，此服务器端事务必须传递此响应到传输层来进行传输。这不是一个由服务器端事务发起的重传流程。

2xx 响应的重传是由 TU 来负责处理。服务器端事务必须切换到"Terminated"状态。

当服务器端事务在"Proceeding"状态时，如果 TU 传递了一个响应消息，对服务器端事务返回的状态码是 300 到 699，此响应必须传递到传输层进行传输，状态机必须进入到"Completed"状态。对于非可靠传输来说，定时器 G 设置在 T1 秒内启动，此定时器不设置启动可靠传输。

此修改是来自于 RFC2543，这里响应总是被重传，甚至于看传输也会进行重传。

当服务器端事务进入到 "Completed" 状态时，定时器 H 必须设置在  $64*T1$  内启动来支持可靠传输和非可靠传输。定时器 H 决定何时服务器端事务停止重传响应。它的值等同于定时器 B，这个时间周期是客户端事务继续重试发送一个请求的时间。如果定时器 G 触发以后，响应被多次传递到传输层进行重传，并且，定时器设置在  $\text{MIN}(2*T1, T2)$  秒内启动。从那开始起计算，当定时器 G 被触发以后，此响应会被再次传输到传输层进行重新传输，并且，定时器 G 重新设置，设置的值为以前值的 2 倍。在一些场景中，定时器 G 需要重新设置，这个值超过 T2，那么定时器 G 就会设置为 T2 值。这个处理和非-INVITE 客户端事务的 "Trying" 状态中的请求处理完全相同。进一步说，当服务器端事务在 "Completed" 状态时，如果它收到一个重传请求，此服务器应该传递此响应到传输层来进行重新传输。

当服务器端事务在 "Completed" 状态时，它收到一个 ACK 的话，服务器端事务必须切换到 "Confirmed" 状态。因为定时器 G 已经在此状态下被忽略，响应的任何重传都会被停止。

当服务器端事务在“Completed”状态时，定时器 H 被触发的话，服务器端事务会说明，从来没有收到 ACK。在这种情况下，服务器端事务必须切换到“Terminated”状态，并且必须对 TU 指示已发生一个事务处理失败。

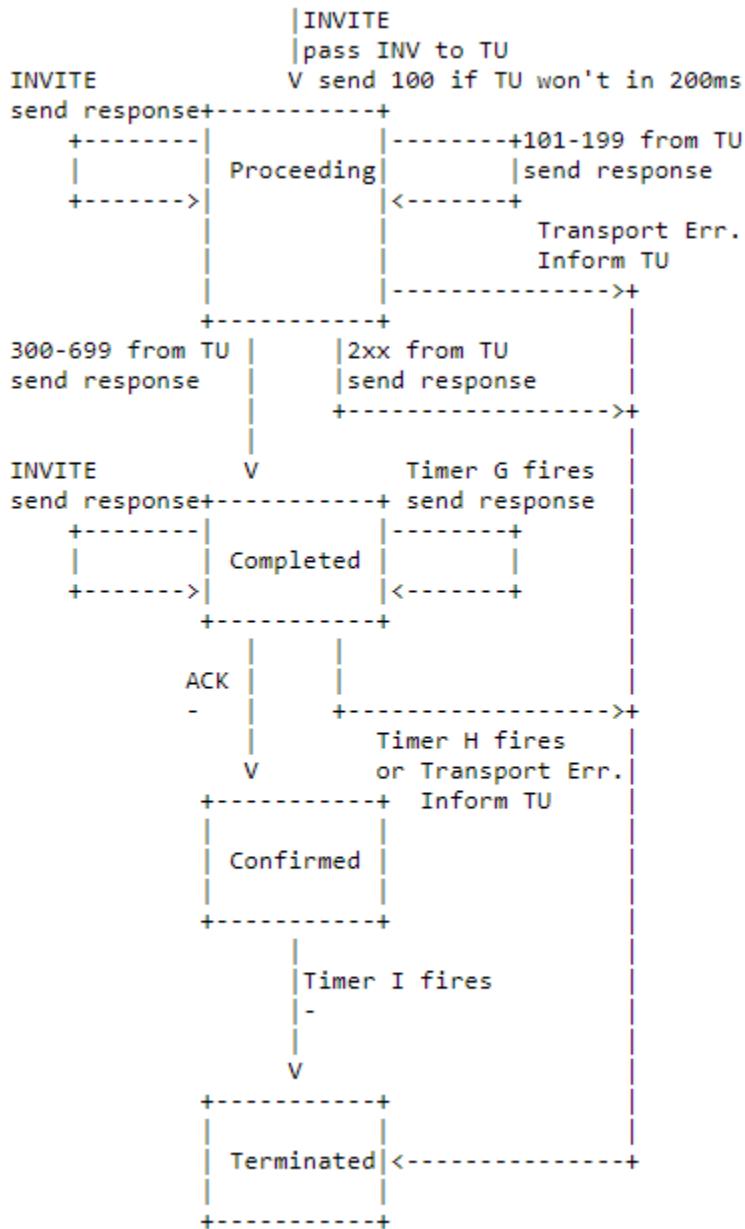


Figure 7: INVITE server transaction

“Confirmed” 状态的目的是处理任何其他额外的 ACK 消息，这些消息是由最终响应重传发送过来的或由最终响应重传所触发的信息。当服务器端事务进入到这个状态时，定时器 I 设置为 T4 秒内等待启动，支持非可靠性传输，并且设置零秒为可靠传输。一旦定时器 I 被触发以后，服务器必须切换到 “Terminated” 状态。

一旦此服务器端事务进入到 “Terminated” 状态，它必须马上被销毁。和客户端事务一样，同样也需要确保 INVITE 的 2xx 响应的可靠性。

### 17.2.2 Non-INVITE Server Transaction

非-INVITE 服务器端事务的状态机在图例 8 显示。

状态机在 “Trying” 状态下被初始化，当被执行初始化时，状态机传递一个请求（除了 INVITE 或者 ACK 以外）。这个请求传递到 TU 端。一旦进入到 “Trying” 状态的话，其他更多请求重传就会被丢弃。如果此请求匹配了同样的服务器端事务（根据第 17.2.3 规则匹配）的话，这个请求就是一个重传请求。

当状态机在 “Trying” 状态时，如果 TU 对服务器端事务传递了一个临时响应的话，服务器端事务必须进入到 “Proceeding” 状态。此响应必须传递到传输层进行传输。当在 “Proceeding” 状态时，任何从 TU 收到的更多临时响应必须被传递到传输层进行传输。当在 “Proceeding” 状态时，如果收到一个请求重传的话，最近被发送的临时响应必须传递到传输层进行重传。当在 “Proceeding” 状态时，如果 TU 传递了一个最终响应（状态码是 200-699），此事务必须进入到 “Completed” 状态，响应消息必须被传输到传输层进行传输。

当服务器端事务进入到 “Completed” 状态时，服务器端事务必须为非可靠性传输在  $64*T1$  秒内设置定时器 J 来等待启动，为可靠性传输设置定时器 J 为零秒启动。当在 “Completed” 状态时，无论何时收到一个重传的请求，服务器端事务必须传递最终响应到传输层进行重传。当在 “Completed” 状态时，必须丢弃任何由 TU 传递到服务器端事

务的其他最终响应。服务器端事务一直保持这个状态，直到定时器 J 被触发，在这个触发时间点，服务器端事务必须切换到“Terminated”状态。

一旦服务器端事务进入到“Terminated”状态以后，它必须销毁这个工作示例。

### 17.2.3 Matching Requests to Server Transactions

当从网络收到一个请求以后，请求响应匹配现存的事务。匹配处理过程按照以下方式进行处理。

在请求最顶端的 Via 头的 branch 参数要进行检查。如果它是出现状态，并且是以这个“z9hG4bK”开始，这个由客户端事务生成的请求是符合此规范的。因此，branch 参数将是一个唯一的标识，涵盖所有由那个客户端发送的所有事务。请求匹配所有的事务，如果：

1. 请求中的 branch 参数等于此请求（此请求创建了事务）Via 头最顶端的一个参数，并且
2. 在请求中的顶端的 Via 中的 sent-by 值等于请求中的一个值，此请求创建了事务，并且
3. 请求的 method 匹配一个 method，这个 method 创建了此事务（除了 ACK 以外），  
    创建 事务的请求的 method 是 INVITE。

这个匹配规则同样可适用于 INVITE 和非-INVITE 事务。

sent-by 的值可用来作为匹配流程的一个部分，因为可能由很多不同的客户发送的 branch 参数，这些参数可能是重复的，也可能是恶意生成的参数。

如果在请求最顶端的 Via 头的 branch 参数没有出现的话，或者不包含“z9hG4bK”值，则需要根据以下规则来进行处理。这些规则和 RFC2543 兼容。

如果 Request-URI, To tag, From tag, Call-ID, CSeq, 和顶端的 Via 头匹配了 INVITE 请求（这个 INVITE 请求创建了事务）中的这些头值，则 INVITE 请求匹配了事务。在这种情况下，此 INVITE 是一个初始 INVITE（创建此事务）的重传。如果，Request-URI, From tag, Call-ID, CSeq number（不是 method），和顶端 Via 头值匹配了 INVITE 请求（这个请求创建了事务）中的这些头值，并且 ACK 中的 To 标签匹配了由服务器端事务发送过来的 To 标签值，这个 ACK 请求则匹配了事务。匹配流程是通过匹配规则（针对每个头都有其定义）来处理的。ACK 匹配过程中的 To 头中所包含的标签帮助消除针对在代理不同其他响应的 2xx 响应的 ACK，很多时候，代理端口你已经转发了所有的响应消息。这种情况虽然不是经常发生，但是有这种情况存在。例如，proxy 可能做了一个请求的分叉处理，然后，可能这个代理就出现了故障，响应消息可能被发送到其他的代理服务器，这样其他多个响应消息就会停止发送消息到上游服务器地址。

一个匹配了 INVITE 事务（已匹配前面的 ACK）的 ACK 请求被认为是一个前面 ACK 请求的重传处理。

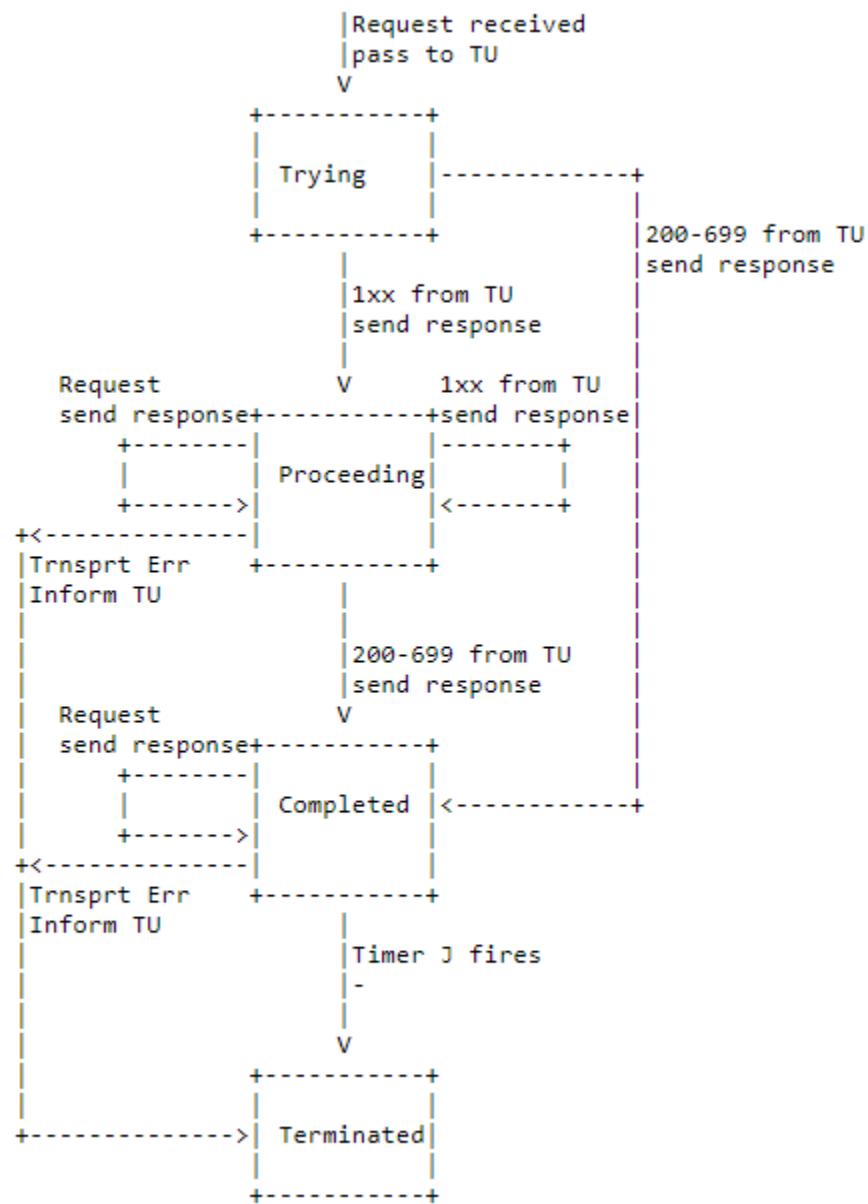


Figure 8: non-INVITE server transaction

对于其他的请求 methods 来说，如果 Request-URI, To tag, From tag, Call-ID, CSeq (包括此 method) 和顶端的 top Via 头匹配了创建事务的请求中的参数值，一个请求匹配了一个事务。匹配过程是通过对每个头的定义的规则来进行的。

当一个非 INVITE 请求匹配了一个现存的事务，它则是一个创建事务的请求的重传处理。

因为匹配规则包括了一个 Request-URI，服务器不能对事务匹配一个响应。当此 TU 传递一个响应到这个服务器端事务时，TU 必须传递 TU 到一个具体的服务器端事务，这个具体的服务器端事务是响应的目的地地址。

#### 17.2.4 Handling Transport Errors

当服务器端事务对传输层发送了一个响应，这个响应将要从传输层发送出去，如果传输层指示了一个失败结果时，需要根据以下规则进行错误处理。

首先，需要按照[参考路径 4]中的步骤进行处理，这个步骤尝试发送响应到一个备份地址。如果这些处理失败的话，按照[参考链接 4]说明的失败的定义，服务器端事务应该通知 TU 已发生一个失败流程，服务器端事务应该把状态机切换到结束状态。

## 18 Transport

传输层负责通过网络传输层进行请求和响应的实际传输。这也包括连接的确定，如果发生面向连接时，使用何种连接来支持请求或响应。

传输层负责针对传输协议，例如 TCP 和 SCTP 或 TLS 等，用来管理其连接的可持续性，包括这些对传输层开放的端口。这也包括对客户端传输和服务器端传输开启的连接，因此，这个连接由客户端和服务器端共享传输功能。这些连接以索引的方式进行排列，由三组不同数据构成，包括远端连接的地址，端口和传输协议。当连接对传输层开启以后，此索引会设置目的地地址，端口和传输方式。当传输层接受了连接以后，此索引被设置为源 IP 地址，端口号和传输方式。注意，因为源端口经常是瞬间开放的，端口不会被获悉是瞬间开放或通过流程[4]选择的端口，被传输层接受的端口将不会被经常重复使用。其结果是，使用面向连接的两个处于“端绑定”关系的代理 经常有两个连接，其中一个是在每个方向上支持事务初始化处理的。

本规范推荐，某些场景使用此连接发送最后的消息发送，或者接收最后消息以后，针对某些定义的部署场景时间段连接保持开放状态。这个时间段应该至少等于网络网元传输所需要的时间总和，这些网络网元需要一定的时间引入一个事务一直到结束事务状态。这样的可以保证通过同样发起的连接，事务能够成功完成（例如，请求和响应，并且在 INVITE 场景中，ACK 是一个非 2xx 响应）。这样的处理过程表示至少需要设置  $64*T1$ （参考 第 17.1.1.1 章节对 T1 的定义）。但是，这个定时器值可以是一个比较大的值，例如，一些网络网元使用了一个 TU，这个 TU 使用了一个比定时器 C 更大的取值（参考 16.6 章节）。

所有 SIP 网元必须部署支持 UDP 和 TCP。SIP 网元也可以使用其他的响应。

对 UA 强制使用 TCP 是 RFC2543 的重大修改。很多应用产生了大量的消息传输，这些大批量的消息传输必须使用 TCP。在下面章节将讨论为什么必须使用 TCP。因此，即使网络网元从来不发送内容很大的消息，这些网络网元也可能接收这些消息，并且需要具备一定的能力来处理这些大内容消息。

## 18.1 Clients

### 18.1.1 Sending Requests

传输层的客户端侧负责发送请求和接收响应。传输层用户传递客户传输请求，一个 IP 地址，端口，传输方式和可能支持一个 TTL 的多播目的地。

如果一个请求在路径 MTU 的 200 bytes 之内，或者此请求大于 1300 bytes 并且此路径 MTU 是未知的，此请求必须使用 RFC2914 [43] 拥塞控制响应来完成传输，例如 TCP 协议。如果这种情况引起了在 top Via 中指示的传输协议修改，top Via 中的值也必须修改。这样做的目的是防止通过 UDP 传输的消息碎片，并且为大数据包提供拥塞控制。但是，部署方式必须能够处理消息数据包，支持到最大的数据包大小。对于 UDP 来说，数据包大小是 65,535 bytes，包括 IP 和 UDP 头。

介于消息大小和 MTU 之间的这个 200 byte “缓冲” 调节一个实际事实，SIP 中的响应数据会大于此响应的请求的数据。例如，发生这样的情况是因为对此请求在响应中增加了 Record-Route 头的其他部分。如果支持了这个

多余的缓冲，响应消息可以是大概 170 bytes，这样的话，响应数据就会大于请求的数据，并且仍然不会在 IPv4 的网络环境中导致消息碎片（假设没有 IPSec，大概 30 bytes 会被 IP/UDP 占用）。

基于 1500 bytes 的以太网 MTU 的假设，当路径 MTU 是未知状态时，1300 bytes 会被选择使用。

因为一个请求的消息大小的限定，如果网元通过 TCP 发送请求，此请求使用 TCP 发送，否则使用 UDP 发送。如果此尝试创建来连接，要么连接生成了一个不支持的 ICMP 协议，要么导致 TCP 重设，网元应该使用 UDP 重试此请求。这样的流程仅提供一个向后兼容支持来支持 RFC2543，此规范不支持 TCP 协议。这也是可预期的，这种处理方式在此官方的未来版本中将会被废弃。

客户端发送一个请求到多播地址，这个客户端必须在其 Via 头中添加一个“maddr”参数，在 Via 头字段值中包含目的地多播地址，并且如果是 IPv4 地址的话，头字段值中应该添加一个参数“ttl”，此参数设置为 1。没有在本规范中定义使用 IPv6 多播地址的参数值，如果有需求时，此定义是将来规范的题目。

这些规则导致了在 SIP 中多播功能目的性的局限。SIP 的基本功能是提供一个“类似于单跳发现”服务，传输一个请求到一个同种类的服务器，要求这些服务器中其中一个服务器处理其请求。这个功能对于注册来说非常有用。事实上，在基于事务处理的流程规则中，客户端事务将接受第一个响应，并且因为他们所有响应都包含同样的 Via branch identifier，客户端事务把其它响应看作重传。

在发送请求之前，客户端传输必须在 Via 头字段中插入一个“sent-by”域值。这个域值包含一个 IP 地址或者主机名，和端口。规范推荐使用一个 FQDN 值。在某些环境中，此域值用来发送响应消息。具体描述如下。如果缺省了端口，在传输中使用默认设置端口。默认端口 UDP 使用 5060，TCP 和 SCTP 支持的 TLS 使用 5061。

对于可靠性传输来说，响应通常在连接状态（请求收到时状态）时被发送。因此，客户端传输必须准备好在接收响应的同时，使用同一连接发送请求。在错误发生的环境中，服务

器可以尝试重新创建一个新的连接来发送响应。为了处理这样的情况，传输层必须准备接收一个接入方向的连接，这个连接来自于通过“sent-by”域值发送请求的源地址和端口。

客户端传输也必须准备接收一个进入方向的连接，这个进入方向的连接发出地址是任何地址和端口，这个地址和端口是服务器端通过一定处理规则选定的，关于如何选定地址和端口按照章节 5 中的[4]部分执行。

对于非可靠单播传输来说，客户端传输必须准备接收来自于源 IP 地址的响应，这个源 IP 地址是“sent-by”值域中的请求发送地址和端口号。另外，和可靠性传输一样，在某些环境中，响应将会发送到其他的地址。客户端必须准备接收来自于任何地址和端口的响应消息，这些地址和端口都是服务器端按照规则选定的地址和端口，关于如何选定地址和端口，按照章节 5 中的[4]部分执行。

对于多播来说，客户端传输必须准备在同样的多播组和端口接收响应消息，多播组和端口是请求发送地址组（那也就是说，这个组是发送请求的多播组成员）。

如果一个请求被预设为 IP 地址，端口和传输方式（当前存在的连接已经开启了此传输），规范建议使用此传输发送请求，但是也可以开启其他的连接和使用其他连接。

如果使用多播地址发送一个请求，这个请求要发送到此组地址，端口和传输层用户提供的 TTL。如果使用单播非可靠传输发送一个请求，这个请求要发送到此 IP 地址和传输层用户提供的端口。

#### 18.1.2 Receiving Responses

当收到一个响应后，客户端传输检查顶部的 Via 头字段值。如果在头中的“sent-by”参数值不能符合这个值，此值是这个客户端传输已配置要插入请求的值，响应必须丢弃此值。如果存在任何客户端事务，客户端传输采用第 17.1.3 章节的匹配流程尝试匹配响应和已存在的事务。如果有一个匹配存在，此响应必须被传递到那个事务。否则，此响应必须被传

递到 core 中（是否是无状态代理，状态代理，代理或者 UA）做进一步的处理。这些“特殊”响应的处理取决于 core 本身（代理将会前转这些响应，而 UA 可能丢弃这些响应）。

## 18.2 Servers

### 18.2.1 Receiving Requests

服务器应该准备接收任何由 IP 地址，端口和传输构成的组合请求，这个组合可以是在 SIP 或者 SIPS URL 端的 DNS 结果，SIPS URL 分发以后实现和服务器通信的目的。这里的“分发”包括在 REGISTER 请求中或者转发响应中插入一个 Contact 头值。在实际生活中，URI 也可以添加在网页上分发出去，也可以通过名片方式分发出去。本规范推荐服务器在所有公开接口监听 SIP 默认的端口（5006 端口支持 TCP 和 UDP，5061 端口监听 TCP 的 TLS）。典型特例是私网环境或者多个服务器示例运行在同一主机。对于服务器用来监听 UDP 的任何接口和端口，服务器也必须使用这些接口和端口监听 TCP。这是因为消息如果消息太大，消息可能需要通过 TCP 传输，而不是 UDP 传输。因此，相反的要求不一定是正确的。服务器需要不针对 UDP 监听一个指定的地址和端口，仅因为监听了同样的地址和端口。这里也可能存在其他更好的原因，为什么服务器需要为 UDP 监听一个指定的地址和端口。

当服务器传输通过任何传输收到一个请求时，它必须检查在顶部的 Via 头值中的“sent-by”参数。如果“sent-by”参数中的 host 主机部分包含一个 domain 名称 或者如果它包含一个 IP 地址，这个地址不同于数据包源地址，此服务器必须在 Via 头字段中添加一个“received”参数。这个参数必须包含一个从数据包收到的源地址。这样做的目的是帮助服务器端传输发送响应，因此，它必须源 IP 地址，这个地址是从请求方来的地址。

现在看一下服务器传输收到的一个请求，它大概的部分数据格式如下：

```
INVITE sip:bob@Biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060
```

这个请求是通过 192.0.2.4 这个源地址收到的。在进一步传输这个请求之前，传输添加了一个“received”参数，因此，这个请求会变成这样：

```
INVITE sip:bob@Biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060;received=192.0.2.4
```

下一步，服务器传输尝试和服务器端事务匹配。匹配流程按照第 17.2.3 章节来执行。如果发现了一个服务器事务的匹配，请求会传递到服务器事务进行处理。如果没有发现匹配，此请求会传递到 core，core 决定为此请求重构一个新的服务器事务。注意，当 UAS core 对 INVITE 发送一个 2xx 响应，服务器事务将被销毁。其含义表示，当 ACK 抵达时，没有发现匹配的服务器事务，并且基于此规则，ACK 要被传递到 UAS core，在 UAS core 中做处理。

### 18.2.2 Sending Responses

服务器传输使用最顶部的 Via 头来决定发送响应的地址。它必须按照以下流程处理：

- 如果“发送协议”是一个可靠传输协议，例如 TCP 或者 SCTP 或者 TLS 的话，如果那个连接仍然是开启状态，响应必须使用现存连接，使用此现存连接来连接到初始请求（它创建了此事务）的源地址。这样就要求服务器端传输保持一个服务器端事务和传输连接的关联关系。如果那个连接不再开启的话，服务器应该对“received”参数中的 IP 地址开启一个连接，如果连接存在，使用在“sent-by”端口，或者如果没有设定端口，使用传输的默认端口。如果尝试连接失败，服务器应该使用在 [4] 的流程决定开启连接的 IP 地址和端口和并且对此 IP 地址和端口返回响应。
- 另外，如果 Via 头字段值包含一个“maddr”参数，响应必须前转到那里列表中的地址，使用在“sent-by”指示的端口，或者如果缺省的话使用 5060 端口。如果这个地址是一个多播地址，此响应应该使用在“ttl”参数中的 TTL 发送，或者如果缺省 ttl 的话，通过 TTL 为 1 发送。
- 另外，（对于非可靠性单播传输），如果顶部 Via 有一个“received”参数，此响应必须发送到在“received”参数中指示的地址，使用在“sent-by”值中指示灯的端口，或者如果无任何明示说明的话，使用端口 5060。如果端口选择失败，例如，出现了一

个 ICMP “端口不可达” 响应，应该使用[参考链接 4]中的章节 5 (RFC3263 规范) 的流程决定发往响应的地址。

- 另外，如果它不是已打标签的接收方，此响应必须返回到 “sent-by” 指示的地址，处理流程按照[参考链接 4]的第五章节执行。

### 18.3 Framing

在以消息初始化的传输（例如， UDP）场景中，如果消息中有一个 Content-Length 头字段，消息体假设包含了多个 bytes 数据。如果在传输数据包中有其他的数据包，超出了消息体本身，这些多余的数据必须丢弃。如果传输数据包在消息体结束前完成，将被视为错误。如果消息是一个响应，必须将它丢弃。如果消息是一个请求，网元应该生成一个 400 (Bad Request) 响应。如果消息体没有 Content-Length 头字段值，消息体在传输数据结束处完成。

在以流初始化的传输（例如， TCP）中，这个 Content-Length 头指示消息体的大小。此 Content-Length 头字段必须使用流初始化传输。

### 18.4 Error Handling

错误处理是独立执行的，它和消息是请求或者响应无关。

如果传输用户要求通过非可靠传输发送消息，导致了一个 ICMP 错误，此处理方式依赖于 ICMP 的类型。主机，网络，端口或者协议的非可靠错误，或者参数问题所导致的错误应该会造成传输层在发送中通知一个传输用户失败。源结束和 TTL 超过 ICMP 错误应该忽略。

如果传输用户要求通过可靠传输发送一个请求，导致了一个连接失败，传输层应该在发送中通知一个传输用户失败。

## 19 Common Message Components

SIP 消息中有各种模块构件，这些模块构件会出现在 SIP 消息的不同的地方（有时可能不在 SIP 消息中），这些消息值得分别讨论。

### 19.1 SIP and SIPS Uniform Resource Indicators

一个 SIP 或 SIPS URI 定义了一个通信的资源。像其他所有的 URLs 一样，SIP 和 SIPS URLs 可以置入到网页中，邮件消息中或者打印出的资料中。它们包含了充足的信息配合资源来初始化和维护会话。

通信资源示例包含以下内容：

- 一个用户的最新服务
- 多线路电话呈现
- 消息系统中的邮箱
- 网关服务的电话号码
- 组织中的一个组（例如，“销售”，或者“客服”）

一个 SIPS URI 设定安全通信的资源。其含义特别表示，TLS 用于 UAC 和它 URL 所属的域名之间。从这一点来看，在它们之间，根据域名设定的具体的安全机制策略，使用安全通信可以抵达用户。任何由 SIP URL 描述的资源可以通过修改结构升级为一个 SIPS URI。

#### 19.1.1 SIP and SIPS URI Components

“sip:” 和 “sips:” 架构按照 RFC2396 的[5]章节进行处理。它们使用的格式和 mail 对的 URL 相似，遵从 SIP 请求头字段和 SIP 消息体细节进行处理。这样就可能设定主题，媒体类型或者，或者设定页面中的 URL，邮件中的初始化的紧急会话。SIP 或者 SIPS URL 的格式语法在第 25 章节中介绍。一般的 SIP URL 格式为：

```
sip:user:password@host:port;uri-parameters?headers
```

SIPS URL 的格式也是类似的，除了使用结构 “sips” 替代了 sip 以外。它的标识和部分标识符号的解释如下：

user: 带主机地址的参与者资源身份。这里的 “host” 经常被看作为一个域名地址。URL 中的由用户值域，密码域和跟随它们的符号@标识构成。userinfo 是 URL 的可选选项，当目的地主机没有此用户标识或当主机自己被看作一个资源被确认时，这个 userinfo 部分可以缺省。如果@符号出现在 SIP 或者 SIPS URI 中时，user 域值部分一定不能为空。

如果被标识的主机地址能够处理电话号码，例如，一个网络电话网关地址，在 RFC 2806 [9] 定义的电话定义值域用来填入此 user 值域。在 19.1.2 中描述了多个转译规则对 SIP 或者 SIPS URL 中的电话订阅值进行编码。

password: 密码和用户关联。虽然 SIP 和 SIPS URL 语法允许此域值出现，但是这种使用方式是不推荐的，因为通过明文（URLS）传递认证信息在几乎所有的场景中已经被证明存在安全风险。例如，在此值域中传输 PIN 号码的话，将会暴露 PIN 号码。

注意，密码域仅是用户部分的扩展。对于密码部分如果不希望给予过多特别重要关注的话，密码部分可以把"user:password"作为一个单字符串来处理。

host: 主机提供 SIP 资源。主机是全限定类型的域名或者 IPv4 或 IPv6 地址。在任何时候只要可能，使用全限定域名格式是推荐方式。

port: 端口号是请求被发送的端口号。

URI parameters: 来自于 URL 的参数，影响请求构建。

URL 参数添加到主机后，通过冒号分开。

URI 参数使用的格式如下：

parameter-name "=" parameter-value

尽管可以在 URL 参数中可以添加任意数量的参数，但是，已给定的参数名称只能在 URL 中出现一次。

此扩展机制包括传输，maddr，ttl，user，method 和 lr 参数值。

传输传输决定何种传输机制发送 SIP 消息，在[4]中定义。SIP 可以使用任何传输协议。参数名称分别使用 UDP (RFC 768 [14]), TCP (RFC 761 [15]), 和 SCTP(RFC 2960 [16])。对于 SIPS URL 参数来说，传输必须指示为可靠性传输。

maddr 参数指示此用户将要联系的服务器地址，它是从主机域获取的最重要地址。当出现了 maddr 参数时，在参数值中的 URL 的端口和传输模块会使用此地址。[4] 描述了传输，maddr 和主机端口正确兼容性处理以便获得目的地地址，端口和传输来发送请求。

Maddr 域值已经被用来作为一种简单的松散源路由形式。它允许 URL 指定一个代理，必须经过此代理路径才能到达目的地地址。继续使用 maddr 参数这种方式是强烈不推荐的（启用这种机制的处理方式已经被废止）。在此官方中，部署使用 Route 机制是推荐的方式，如有必要，它创建了一个已存在路由表（参考第 8.1.1.1 章节）。这样的方式提供了一个完整的 URL 来描述穿越到节点。

ttl 参数值决定 UDP 多播数据包的存活时长，并且仅必须使用在如果 maddr 是一个多播地址和传输协议是 UDP 的情况下。例如，设定了一个呼叫为 alice@atlanta.com，它使用的多播地址为 239.255.255.1，支持的 ttl 值为 15 的话，那么以下 URL 应该被使用：

sip:alice@atlanta.com;maddr=239.255.255.1;ttl=15

有效的电话订阅用户字符串集是有效用户字符串的子集。用户的 URI 参数存在的目的是从用户名称（经常发生的情况是用户名称类似于电话号码）中区别于电话号码用户。如果用户字符串包含了一个电话号码，此电话号码格式化为 telephone-subscriber，此用户参数值的 “phone” 应该出现。甚至没有此参数的话，如果对于用户名称的名称空间的本地限定允许那样的话，SIP 和 SIPS 的接收方可以解析此 pre-@ 部分作为一个电话号码。

从 URL 构建的 SIP 请求的 method 可以通过 method 参数来设定。

当 lr 参数出现时，它表示此网元对此资源负责执行本规范中设定的路由机制。此参数将被使用在 URLs 代理中，置于 Record-Route 头字段值中，并且可能出现在预设的路由表中的 URLs 中。

此参数用来获得系统的向后兼容性，兼容一些系统部署了 RFC2543 规范的 strict-routing（严格路由）机制，和 rfc2543bis 草案一直到 bis-05 的版本。一个准备发送请求，请求是基于 URL 并且 URL 不包含此参数的网元可以假设接收方网元部署了 strict-routing（严格路由），并且重新格式化信息继续维持在 Request-URL 中的信息。

因为 uri 参数机制是可拓展的，SIP 参数网元必须默默的忽略掉它们不理解的参数值。

Headers: 头字段值需要包含在从 URL 构建的请求中。

在 SIP 请求中的头字段可以通过在 URL 中带“？”机制来定义。头的名称和其值被解码为一对用等于号隔开的名称=值的形式。这个特别的名称消息体指示关联的 hvalue 是 SIP 请求的消息体。

Table1 汇总了基于 URL 呈现内容的 SIP 使用和 SIPS URI 的构件。External 行描述了 URLs，这些 URL 出现在了 SIP 消息以外的任何地方，例如，页面中或者名片中的 URL 地址。入口标识“m”是一个强制要求，那些标识“o”是可选标识，那些标识“-”是不被允许的标识。如果出现了未允许的构件的话，正在处理 URLs 的网元应该忽略任何未允许的构件。如果可选网元没有出现的话，第二行指示可选网元的默认值。“—”指示此网元既不是可选的或这里无默认值。

在 Contacts 中的 URIs 头字段中有不同的限制，这个限制取决于头字段中出现的内容值。一组是应用在创建和维护 dialogs 的消息中（INVITE 和它的 200 (OK)响应中）。其他的应用在注册和重转消息中（REGISTER 和其 200 (OK)响应，和对任何 method 的 3xx 分类响应中）。

#### 19.1.2 Character Escaping Requirements

	default	Req.-URI	To	From	Contact	reg./redir.	Contact/	dialog
						R-R/Route	external	
user	--	o	o	o	o	o	o	o
password	--	o	o	o	o	o	o	o
host	--	m	m	m	m	m	m	m
port	(1)	o	-	-	o	o	o	o
user-param	ip	o	o	o	o	o	o	o
method	INVITE	-	-	-	-	-	-	o
maddr-param	--	o	-	-	o	o	o	o
ttl-param	1	o	-	-	o	-	-	o
transp.-param	(2)	o	-	-	o	o	o	o
lr-param	--	o	-	-	-	o	o	o
other-param	--	o	o	o	o	o	o	o
headers	--	-	-	-	o	-	-	o

- (1) : 默认端口是传输端口，并且是依赖于技术体系。对于 SIP 来说，默认端口是 5060，可支持的传输包括 UDP，TCP，或者 SCTP。端口 5061 是 SIP 传输使用 TCP 通过 TLS 加密传输和 sips 通过 TCP 传输的端口。
- (2) : 默认的传输方式是取决于技术体系。对于 sip 来说，默认的是 UDP。对于 sips 来说，默认的是 TCP。

Table 1: 对 SIP 头字段值，Request-URL 和参考的使用和 URL 构件的默认值

当定义了一些在 SIP URL 中必须溢出（规避）的字符时，SIP 需要遵守 RFC2396[5]的要求和指南，并且使用它的“%HEX HEX”机制来溢出（规避）这些字符。从 RFC2396[5]的规定来看：

构件定义了在构件中任意给定的 URL 构件，这些构件中实际上预留了字符串。通常来说，对于一个预留字符来说，如果此字符通过溢出（规避）US-ASCII 编码[]的方式被替换尽管 URL 的语义发生了改变，这个字符仍然是一个预留的字符。这里不包括 US-ASCII 字符(RFC 2396[5])，例如空格和控制字符，以及使用在 URL 结束符中的字符，这些字符也必须要规避。URLs 一定不能包括任何未规避的空格和控制字符。

对于每个构件来说，一系列有效的 BNF 扩展准确定义了字符可能出现的未规避，未被处理的可能性。所有其他字符必须被规避处理。

例如， "@"不是一个用户构件中集合的字符，所以， 用户"j@s0n" 必须至少有一个解码的 @ 标识， 例如："j%40s0n"。

在第 25 章的扩展 hname 和 hvalue 令牌中显示所有在头名称和值中的预留 URI 字符必须被规避处理。

用户构件中的 telephone-subscriber 子类有特别规避的考虑因素。 在 RFC2806[9]中关于 telephone-subscriber 描述中一些字符未预留字符，未预留的字符中包含了部分字符，这些字符存在于各种语法要素中。当使用 SIP URLs 时，这些存在于各种语法中的字符需要被规避。任何出现在 telephone-subscriber 中的字符，这些字符没有出现在针对用户规则的 BNF 扩展中，那么这些字符必须被规避。

注意，任何要规避的字符不允许存在于一个 SIP 或者 SIPS URL 中的主机构件的 host 中（% 字符在扩展中是一个无效字符）。作为一个对国际域名定案的要求，这个说明可能要被修改。在当前的部署方式中，如果尝试提高处理的健壮性的话，一定不要试图通过这样的方式处理，把在主机构件收到的已经规避的字符从字面意思看作是对应的未规避的字符来处理。这种处理方式它需要符合 IDN 要求，可能在处理方式方面非常不同。

#### 19.1.3 SIP and SIPS URIs

```

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

```

在上面最后一个 URI 示例中，它含有一个用户域值 - " alice;day=Tuesday" 。以上这个定义的规避规则允许一个分号出现在未规避的域中。为了协议的设计目的，这个域是不透明度。此值的结构仅对 SIP 网元响应所需要的资源有用。

#### 19.1.4 URI Comparison

在此官方中的某些操作要求决定两个 SIP 或者 SIPS URLs 是否相等。此规范中，注册需要对比在注册请求中的 contact 的绑定关系（参考第 10.3 章节），SIP 和 SIPS URLs 的对比是否相等需要按照以下规则来进行：

- 一个 SIP 和 SIPS URI 从来不相等。
- SIP 和 SIPS URL 的用户信息对比是大小写敏感的。这包括用户信息就像电话订阅所包含的密码和格式。除非公开说明，URL 的其他所有构件的对比不是大小写敏感的。
- 在 SIP 和 SIPS URL 对比中，参数和头字段的顺序不重要。
  
- 除了保留字符（参考 RFC2396 [5]），其他字符等同于它们的 "%HEX HEX%" 编码。
- 通过主机名 DNS 查询获得的结果 IP 地址不匹配那个主机名。
- 对于两个相等的 URIs 来说，用户名，密码和主机名称，端口构件必须是匹配的。

URI 遗漏了用户构件部分的话将不能匹配包含用户构件部分的 URL。URL 遗漏了密码构件部分的话，它也不能匹配包含密码构件的 URL。

URI 遗漏了任何带默认参数构件的话，它不能匹配已公开声明的带默认参数的 URL。例如，一个 URL 遗漏了可选端口构件的话，它不能匹配一个公开声明的带端口 5060 的 URL。这个规则也同样适用于传输参数，ttl 参数，用户参数和 method 构件。

定义 `sip:user@host` 格式不等同于 `sip:user@host:5060`，这是一个来自于 RFC 2543 的改变。当从 URL 数据源中获取地址时，相同的地址期望从相同的 URLs 地址获得。此 URL `sip:user@host:5060` 将总是解析端口 5060。此 URL `sip:user@host` 可能通过在[4]的 DNS SRV 机制来解析。

- o URI 中的 url 参数构件按照以下规则对比：

- 任何出现在对比双方的 URLs 中的 url 参数必须匹配。
- 虽然 url 中包含默认的参数，一个用户，ttl 或者 method 的 url 参数仅在 URL 中出现一 次，那么它们之间也从不匹配。
- 一个包含 maddr 地址参数的 URL 不能匹配未包含 maddr 参数的 URL。
- 当对比 URL 时，所有仅出现过一次的其他所有 uri 参数将被忽略。

- o URI 头构件从来不能被忽略。任何出现的头构件必须出现在双方 URLs 中来支持 URL 匹配。针对每个头的匹配规则参考 20 章节。

以下每个组中的 URL 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.co
m
sip:biloxi.com;method=REGISTER;transport=tcp?to=sip:bob%40biloxi.co
m

sip:alice@atlanta.com?subject=project%20x&priority=urgent
sip:alice@atlanta.com?priority=urgent&subject=project%20x

```

以下每个组中的 URL 是不相等的：

SIP:ALICE@AtLanTa.CoM;Transport=udp	(不同用户名称)
sip:alice@AtLanTa.CoM;Transport=UDP	
Sip:bob@biloxi.com	(可解析为不同端口)
sip:bob@biloxi.com:5060	
sip:bob@biloxi.com	(可解析为不同传输协议)
sip:bob@biloxi.com;transport=udp	
sip:bob@biloxi.com	(可解析为不同端口和传输协议)
sip:bob@biloxi.com:6000;transport=tcp	
sip:carol@chicago.com	(不同头字段组件)
sip:carol@chicago.com?Subject=next%20meeting	
sip:bob@phone21.boxesbybob.com	(虽然
sip:bob@192.0.2.4	phone21.boxesbybob.com 可解析到此 IP)

注意，等同性是不可传递的：

- o sip:carol@chicago.com 和 sip:carol@chicago.com;security=on 是相同的
- o sip:carol@chicago.com 和 sip:carol@chicago.com;security=off 是相同的
- o sip:carol@chicago.com;security=on 和 sip:carol@chicago.com;security=off 是不同的

#### 19.1.5 Forming Requests from a URI

当直接从一个 URL 中构建请求时，部署方式需要注意。URLs 来自于名片信息，网页信息，并且甚至于来自协议本身的原来已注册的 contacts，此 URLs 可以包含非恰当的头字段或者消息体。

部署方式必须在已构建的请求的 Request-URI 中包括任何已提供的传输，maddr，ttl 或者用户参数。如果此 URL 包含了一个 method 参数，它的值必须作为请求 method 使用。这个 method 参数一定不能置于 Request-URI 中。未知的 URL 参数必须置于消息的 Request-URI。

部署方式应该把在 URL 出现的任何头消息或者部分消息体作为一个预期，把这些消息包含在消息中，并且选择这些消息来服务基于每个组件的请求。

部署方式不应该对非常危险的头字段提供服务支持，包括：From，Call-ID，CSeq，Via，和 Record-Route。

一个部署方式不应该支持任何已请求的 Route 头字段值，这样做的目的是为了不被作为一个在恶意攻击中的未知情的代理使用。

一个部署方式不应该支持这些请求，请求中包括可能引起错误地展现其位置和能力的头字段。这些头字段包括：Accept，Accept-Encoding，Accept-Language，Allow，Contact（在其 dialog 使用），Organization，Supported 和 User-Agent。

一个部署方式应该验证任何已请求的可描述头字段的准确性，这些头字段包括：Content-Disposition，Content-Encoding，Content-Language，Content-Length，Content-Type，Date，Mime-Version 和 Timestamp。

如果一个请求是无效的 SIP 请求，这个请求构建的信息是从一个给定的 URL 获得，那么这个 URL 也是无效的。实施方式一定不能处理此传输中请求。因为在 context 中含有无效的 URL，发生了这种情况，它应该跟踪此行为的原因。

构建的请求可以是无效的，无效的请求通过不同的方式来表达。它们包括，但是不仅限于，头字段中的语法错误，URL 参数的无效组合，或者消息体的错误描述。

发送一个从给定的 URL 构建的请求可以要求一个能力，对此部署方式来说，这个能力可以是一个无效的能力。此 URL 可能指示一个未部署的传输使用方式或者扩展方式，例如，一个部署方式应该拒绝发送这些请求，而不是修改请求来匹配支持能力。部署方式一定不能发送一个请求，这个请求要求一个扩展，这个扩展是它本身不能支持的扩展。

例如，这样一个请求，它是以这样的方式构建的。它通过一个 Require 头参数或者一个 method URL 呈现的，呈现数据中带有一个未知的或者明确不支持的值。

#### 19.1.6 Relating SIP URIs and tel URLs

当一个 tel URL(RFC2806[9])被转换成一个 SIP 或者 SIPS URI 地址时，tel URL 的整个 telephone-subscriber 部分和其包括的任何参数将被置于 SIP 或者 SIPS URL 的 userinfo 部分中。

因此，`tel:+358-555-1234567;postd=pp22` 转换成  
`sip:+358-555-1234567;postd=pp22@foo.com;user=phone`

或者

`sips:+358-555-1234567;postd=pp22@foo.com;user=phone`

而不是

`sip:+358-555-1234567@foo.com;postd=pp22;user=phone`

或者

`sips:+358-555-1234567@foo.com;postd=pp22;user=phone`

一般来说，已转换成 SIP 或者 SIPS URLs 等同的"tel" URLs，以这种方式处理的话，可能不会生成等同的 SIP 或者 SIPS URIs。SIP 或者 SIPS URLs 的 userinfo 部分数值是以大小写敏感的方式进行对比的。Tel URLs 中大小写不敏感的变量和 tel URL 的参数重新排序不会影响 tel URL 的等同关系，但是，它确实影响 SIP URLs 等同关系，这里的这些 SIP URLs 是通过 tel URLs 中构建的。

例如，

```
tel:+358-555-1234567;postd=pp22  
tel:+358-555-1234567;POSTD=PP22
```

等同于，

```
sip:+358-555-1234567;postd=pp22@foo.com;user=phone sip:+358-555-  
1234567;POSTD=PP22@foo.com;user=phone
```

同样的，

```
tel:+358-555-1234567;postd=pp22;isub=1411 tel:+358-555-  
1234567;isub=1411;postd=pp22
```

等同于，

```
sip:+358-555-1234567;postd=pp22;isub=1411@foo.com;user=phone sip:+358-555-  
1234567;isub=1411;postd=pp22@foo.com;user=phone
```

为了改善或解决这个问题，针对构建 telephone-subscriber 的要素参数，这些参数被转换置于 SIP 或者 SIPS URL 的 userinfo 部分时，这些参数应该通过封装处理 telephone-subscriber 中任何对大小写不敏感的部分数据，将这些大小写不敏感的字符转换为小写字符，并且对 telephone-subscriber 参数名称在词法上按照参数名称进行顺序。注意，这里除了 isdn-subaddress 和 post-dial 以外，这些参数会首先按照顺序发生（所有 tel URL 组件除了未来扩展到参数以外，它们都被定义，并且以大小写不敏感的方式进行对比）。

按照以下建议，所有的

```
tel:+358-555-1234567;postd=pp22  
tel:+358-555-1234567;POSTD=PP22
```

转换为

```
sip:+358-555-1234567;postd=pp22@foo.com;user=phone
```

并且，所有的

```
tel:+358-555-1234567;tsp=a.b;phone-context=5  
tel:+358-555-1234567;phone-context=5;tsp=a.b
```

转换为

```
sip:+358-555-1234567;phone-context=5;tsp=a.b@foo.com;user=phone
```

## 19.2 Option Tags

Option tags 是一个唯一标志，用来指明 SIP 中的新 options（扩展）的。这些 tags 在 Require( 第 20.32 章 )， Proxy-Require( 第 20.29 章 )， Supported( 第 20.37 章 ) 和 Unsupported( 第 20.40 章 ) 头字段中使用。注意这些 options 是以 option-tag= 的形式作为这些头字段的参数存在的（第 25 章有关定义符号）。

Option tags 是根据标准的 RFC 扩展定义的。这和过去的部署方式有所不同，这是规范组织为了保证多个厂商之间能够持续互相协作兼容（第 20.32 章、第 20.37 章的讨论）。option tags 的 IANA 注册记录可以保证用户方便参照。

## 19.3 Tags

“tag” 参数用于 SIP 消息中的 To 和 From 头字段。它作为一个通用的机制一部分来唯一标志一个对话，这个机制用 Call-ID 和两个从对话参与者的 tag 来标志一个对话。当 UA 在

对话外发出一个请求时，它只包含了 From tag, 提供了对话 ID 的“一半”。对话根据应答创建完成，这个应答在 To 头字段中提供了对话 ID 的另一半。SIP 请求的分支意味着一个单个请求可以创建多个对话。这个也解释了为何需要对话两方的标志；如果没有被叫方的标志，呼叫方不能分辨和消除由单个请求创建的多个对话。

当 UA 产生一个 tag 并且增加进一个请求或者应答的时候，它必须是一个全局唯一的，并且是密码随机数起码是 32 位的随机数。这个要求是为了让 UA 能够在同一个 INVITE 请求中，在给这个 INVITE 的应答中，在 To 头字段产生一个不同的 tag，和原始 INVITE 请求在 From 头字段中产生的 tag 不同。这是因为 UA 可以邀请自己到一个会话，常见的是在 PSTN 网关端实现的“hairpinning”（发夹）呼叫。类似的，对不同呼叫的两个 INVITE 也有不同的 From tag，并且给这两个呼叫的两个应答也有不同的 To tag。

在全局唯一要求之外，产生 tag 的算法是实现相关的。Tag 对于容错系统比较有用，在容错系统中，当主服务器出现故障的时候，对话会在另外一个服务器上进行恢复。UAS 可以产生一个 tag，让备用服务器能够识别到这个请求是在故障服务器上的对话，并且能够决定是否恢复对话和对话相关状态。

## 20 Header Fields

头字段或者头字段的语法描述在 7.3 节。本节列出了头字段的全部列表，包括了语法注释，含义，和用法。通过本节，我们使用[HX.Y]指当前 HTTP/1.1 的 RFC2616[8]的规范的 X.Y 节。每个头字段或者头字段都给出了示例。

关于与方法和 proxy 处理有关的头字段字段在表 2 和表 3 中有处理。

“where” 列描述了在头字段中能够使用的请求和应答的类型。这列的值是：

- R: 头字段只能在请求中出现；
- r: 头字段只能在应答中出现；
- 2xx, 4xx, 等等：一个数字的值区间表示头字段能够使用的应答代码；
- c: 头字段是从请求拷贝到应答的。

如果” where” 栏目是空白，表示头字段可以在所有的请求和应答中出现。

“proxy” 列描述了 proxy 在头字段上的操作

- a : 如果头字段不存在， proxy 可以增加或者连接头字段。
- m : proxy 可以修改现存的头字段值。
- d : proxy 可以删除头字段值。
- r : proxy 必须能读取这个头字段，因此这个头字段不能加密。

接下来 6 个栏目与在某一个方法中出现的头字段有关：

- c : 条件；对头字段的要求依赖于消息的内容。
- m : 头字段是强制要有的。
- m\* : 头字段应当被发送，但是客户端/服务端都需要准备接收没有这个头字段的消息。
- o : 头字段是可选的。
- t : 头字段应当被发送，但是客户端/服务端都需要准备接收没有这个头字段的消息。客户端/服务端都需要准备接收没有这个头字段的消息。如果通讯的协议是基于面向流的协议（比如 TCP），那么头字段值必须被发送。
- \* : 如果消息体不为空，那么头字段值就需要的。（细节请参见第 20.14,20.15 和第 7.4 章节）。
- :这个头字段是不适用的。

“Optional” 意味着这个网元可以在请求或者应答中包含这个头字段，并且 UA 可以忽略在请求或者应答中存在的这个头字段（这条规则有一个例外，就是 Require 头字段，在第 20.32 章节有描述）。” mandatory” （强制）头字段是必须在请求中存在的头字段，并且也必须是 UAS 接收到一个请求时能够理解的头字段。一个强制头字段必须也在应答中出现，并且 UAC 也能处理这个头字段。” Not applicable” （不适用）意味着头字段不能在请求中出现。如果一个 UAC 错误的把这个头字段放在请求中，在 UAS 收到的时候必须被忽略。同样的，如果应答中的” 不适用”的头字段，也就是说 UAS 不能在应答中放置的头字段，如果出现了，那么 UAC 也必须在应答中忽略掉这个头字段。

一个 UA 必须忽略他们所不能处理的扩展的头参数。

当整个消息大小是一个争议时，本规范也定义了常用的头字段或者头字段名的缩写。

在 Contact, From, To 头字段中都包含一个 URI。如果这个 URI 包含一个逗号, 问号或者分号, 那么这个 URI 必须使用尖括号括起来 (<和>)。所有的 URI 参数都必须在这些括号内。如果 URI 并非用尖括号括起来的, 那么用分号分开的参数将被视同与 header 参数而不是 URI 参数。

## 20.1 Accept

Accept 头字段的语义定义遵从[H14.1]。除了如果没有 Accept 头字段, 服务器应当认为 Accept 缺省值是 application/sdp 以外, 语义也是和 HTTP/1.1 类似的语义。  
空的 Accept 头字段意味着不接受任何格式。

例如:

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	
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	
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	
Content-Type		*	*	-	*	*	*	
CSeq	c	r	m	m	m	m	m	
Date		a	o	o	o	o	o	
Error-Info	300-699	a	-	o	o	o	o	
Expires		-	-	-	o	-	o	
From	c	r	m	m	m	m	m	
In-Reply-To	R	-	-	-	o	-	-	
Max-Forwards	R	amr	m	m	m	m	m	
Min-Expires	423	-	-	-	-	-	m	
MIME-Version		o	o	-	o	o	o	
Organization		ar	-	-	o	o	o	

Table 2: Summary of header fields, A--O

Header field	where		proxy	ACK	BYE	CAN	INV	OPT	REG
Priority	R	ar	-	-	-	o	-	-	-
Proxy-Authenticate	407	ar	-	m	-	m	m	m	m
Proxy-Authenticate	401	ar	-	o	o	o	o	o	o
Proxy-Authorization	R	dr	o	o	-	o	o	o	o
Proxy-Require	R	ar	-	o	-	o	o	o	o
Record-Route	R	ar	o	o	o	o	o	o	-
Record-Route	2xx, 18x	mr	-	o	o	o	o	-	-
Reply-To			-	-	-	o	-	-	-
Require		ar	-	c	-	c	c	c	c
Retry-After	404, 413, 480, 486 500, 503 600, 603		-	o	o	o	o	o	o
Route	R	adr	c	c	c	c	c	c	c
Server	r		-	o	o	o	o	o	o
Subject	R		-	-	-	o	-	-	-
Supported	R		-	o	o	m*	o	o	o
Supported	2xx		-	o	o	m*	m*	o	o
Timestamp			o	o	o	o	o	o	o
To	c(1)	r	m	m	m	m	m	m	m
Unsupported	420		-	m	-	m	m	m	m
User-Agent			o	o	o	o	o	o	o
Via	R	amr	m	m	m	m	m	m	m
Via	rc	dr	m	m	m	m	m	m	m
Warning	r		-	o	o	o	o	o	o
WWW-Authenticate	401	ar	-	m	-	m	m	m	m
WWW-Authenticate	407	ar	-	o	-	o	o	o	o

Table 3: Summary of header fields, P--Z; (1): copied with possible addition of tag

Accept: application/sdp;level=1, application/x-private, text/html

例子：

Accept: application/sdp;level=1,application/x-private,text/html

## 20.2 Accept-Encoding

Accept-Encoding 头字段类似 Accept，但是限定了接收应答中的内容的编码[H3.5]。参见 [H1 4.3]。在 SIP 中的语义和在[H14.3]中的定义是一致的。

规范允许一个空的 Accept-Encoding 头字段。它等同于 Accept-Encoding:identity，这就是说，仅只有 identity 解码，无实际解码，这种情况也是允许的。

如果没有 Accept-Encoding 头字段出现，那么服务端应当假设使用缺省值：identity。

这个和 HTTP 的定义略有不同，HTTP 提示如果本头字段不存在，那么可以使用任何编码形式，仅只是推荐 identity 编码而已。

例如：

Accept-Encoding:gzip

### 20.3 Accept-Language

Accept-Language 头字段用在请求中指定首选的的语言支持，这个首选语言使用在请求中的推荐语言，用来支持原因短语分析，会话描述，或者响应状态中传输的消息体内容。如果没有 Accept-Language 出现，那么服务端应当假设客户端会接受所有的语言。

Accept-Language 头字段遵从[H14.4]节定义的语法。对于 SIP 来说，也同样支持对语言排序，支持通过“q”参数来进行排序。

例如：

```
Accept-Language: da, en-gb; q= 0.8, en;q=0.7
```

### 20.4 Alert-Info

当 INVITE 请求有一个 Alert-Info 头字段的时候，Alert-Info 头字段就包含的是给 UAS 的一个额外的信息。当在 180 (Ringing) 应答中出现的时候，Alter-Info 头字段给出了 UAC 一个额外的回铃信息。这个头字段的一个典型用法就是让 proxy 增加这个头字段用来提供一个有差异的振铃功能。

Alter-Info 头字段可能会带来潜在的安全隐患。这个隐患以及相应的处理在第 20.9 章节有讲述，这个隐患和 Call-Info 头字段的隐患是相同的。

另外，用户应当可以有选择的屏蔽这个特定。

这个可以保护用户不因为使用了未受信任节点发送过来的这个头字段而导致的破坏。

例如：

Alter-Info: <http://www.example.com/sounds/moo.wav>

#### 20.5 Allow

Allow 头字段列出了 UA 生成的方法列表。

当 Allow 头字段支持出现时，所有只有 UA 支持的方法，包括 ACK 和 CANCEL 都必须列在这个 Allow 头字段中。如果没有 Allow 头字段缺省时，一定不能解析为发送消息的 UA 何种方法都不支持。准确的说，发送这个消息的 UA 并没有通知对端它支持何种方法。

在应答中提供 Allow 头字段支持不在 OPTIONS 提供支持支持头字段，这样会减小协商所需要的的消息数量。

例如：

Allow: INVITE, ACK, OPTIONS, CANCEL, BYE

#### 20.6 Authentication-Info

通过 Authentication-Info 域提供和 HTTP 相同的认证方法。UAS 可以在一个请求的带 2xx 的响应中包含一个头字段，这个请求是使用 digest 基于 Authorization 头字段一个成功认证的请求。

这个头字段的语法和语义遵循 RFC2617[17]的规范。

例如：

Authentication-Info: nextnonce=" 47364c23432d2e131a5fb210812c"

## 20.7 Authorization

Authorization 头字段包含了 UA 进行认证的信任书。第 22.2 章节介绍了对 Authorization 头字段的用法，第 22.4 章节介绍了和 HTTP 认证一起使用的时候的语法和语义。

这个头字段连同 Proxy-Authorization 并不遵循通常的多头字段值的规则。虽然它不是由逗号分割的列表，这个头字段名可以重复出现多次，但是一定不能使用第 7.3 章节的规则合并这些头成为单个头字段。

在下边的例子中，在 Digest 参数两边没有使用引号括起来。

```
Authorization:Digest username="Alice", realm="atlanta.com", nonce  
= "84a4cc6f3082121f32b42a2187831a94",  
response="7587245234b3434cc3412213e5f113a5432"
```

## 20.8 Call-ID

Call-ID 头字段用来唯一区别一个特定的邀请或者一个特定客户端的所有注册。单个多媒体会议可以生成多个不同 Call-ID 的呼叫，例如，当一个用户多次邀请单个个体加入同一个会议的场景。Call-ID 是大小写敏感，并且是以字节/字节进行比较的。

Call-ID 头字段的简写就是 i

Examples:

Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@biloxi.com i:f81d4fae-7dec-11d0-a765-00a0c91e6bf6@192.0.2.4

## 20.9 Call-Info

Call-Info 头字段提供了对呼叫方或者被叫方的附加信息，取决于此头是在请求中被发现还是在响应中被发现。如果出现在应答中则是被叫方的。URL 的目的是通过“ purpose” 参

数加以说明。”icon”参数包含了一个呼叫方或者被叫方的图标。”info”参数描述了简要的呼叫方或者被叫方的基本信息，例如，置入一个网页对身份进行说明等。”card”参数提供了一个名片，例如，基于 vCard[36]或者 LDIF[37]格式生成的名片。如果附加新的标记，可以通过第 27 节描述的步骤通过在 IANA 注册来附加支持。

使用 Call-Info 可能会带来一些安全风险。如果一个被叫方接到一个恶意呼叫方所提供的 URI，被叫方可能会显示一个不合适的内容，或者危险的甚至于一些非法的内容，等等。因此，规范推荐 UA 仅显示那些它自己能够检验并且信任的发送方身份的 Call-Info 头字段中的内容。这个需求可能对于对方 UA 来说并不需要。代理服务器可以在请求中加入这个头字段。

例如：

Call-Info: http://www.example.com/alice/photo.jpg;purpose=icon,  
http://www.example.com/alice/;purpose=info

#### 20.10 Contact

Contact 头字段提供了一个 URI，这个 URI 的含义取决于这个 contact 是在请求还是在应答中。

Contact 头字段包含了一个显示的名字，一个 URL 以及包含的参数和 header 参数。

本规范定义了一个 Contact 参数”q” 和” expires”。这些参数只有当 Contact 头字段在 REGISTER 的请求或者应答，或者 3xx 的应答中出现时才有效。其他参数可能在其他的规范中定义。当头字段值包含一个显示的名字，那么包含参数的 URI 应该用”<” 和”>”括起来。如果没有”<”，”>” 括起来，所有 URI 以后的参数都将视为 header 参数，而不是 URI 参数。如果需要设置比较长的名称的话，显示姓名可以是符号，或者引号引起起来的字符串。

即使” display-name” 为空，如果” addr-spec” 包含一个逗号或者分号，或者问号的话，也必须使用” name-addr” 的格式。这在 display-name 和” <” 之间可以有也可以没有 LWS(线性空白字符)。关于 LWS，参考：<https://www.rfc-editor.org/std/std68.txt>

对于针对显示名字，URI 和 URI 的参数，header 参数的解析规则同样也适用于 To 和 From 头字段。

Contact 头字段的工作原理类似于 HTTP 中的 Location 头字段。但是 HTTP 头字段只允许 1 个地址，未作为引用数据。作为预留字符，由于 URI 中可以包含逗号和分号，所以它们在各自的 header 或者参数分隔符上出现的话就是错误的。

Contact 头字段的缩写是 m(“moved”)

例如：

```
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 头字段描述了消息体基本消息，或者对于对方消息来说，一个消息体部分是如何被 UAC 或者 UAS 解析的。这个 SIP 字段扩展了 MIME Content-Type(RFC2183[18])支持。

SIP 定义了 Content-Disposition 中的多个新的” disposition-types” 。如果取值是” session” 表示消息体表示的是一个会话，针对的是呼叫 (calls) 或者早期 (pre-call) 媒体流。如果取值是” render” ，表示消息体可是被显示否则渲染展示给用户。注意，使用” render” 而不使用比” inline” ，渲染更适用于避免内涵内容的展现，MIME 消息体作为一个大的消息的一部分做渲染（由于 SIP 消息的 MIME 消息体经常被过滤，不能渲染展示给用户）。因为向后兼容的考虑，如果 Content-Disposition 头字段丢失，服务器应当假设 Content-

Type application/sdp 的消息体是 disposition” session”， 其他内容部分为 “render” 渲染的部分。

disposition 类型是 “icon” ， 表示消息体部分包含了一个用于表示呼叫者或者被叫者的 icon 图像， 当 UA 收到这个消息， 就可以渲染处理， 或者在对话过程中渲染展示。” alert” 表示消息体部分包含了信息， 比如是一段声音， 应当由 UA 展示给用户， 提示用户这个请求， 通常是请求初始化一个对话； 这个 altering 消息体可以是一个在 180 Ringing 临时应答发出后的一个铃声指示音。

任何携带了” disposition-type” 的 MIME 消息体，在这个消息经过了适当的安全认证后渲染显示到 UA 端。

处理参数-handling-param 描述了 UAS 在接收到这个内容类型或者部属类型以后， 如果 UAS 不能理解这些内容消息体时， UAS 应当如何处理。这个参数定义了” optional” 和” required” 两个值。如果处理参数丢失， 那么这个处理参数缺省值就是” required” 。处理参数在 RFC3204[19]中定义和描述。

如果这个头字段不存在， 那么 MIME 的类型决定了缺省的内容处理。如果没有任何 MIME 的类型， 那么假设缺省值就是” render” 。

例如：

Content-Disposition: session

## 20.12 Content-Encoding

Content-Encoding 头字段是对” media-type” (媒体类型)的一个修正。当存在这个头字段的时候， 它的值就是对包体内容编码的附加说明，并且因此必须根据本字段应用正确的解码机制， 这样才能得到正确的 Content-Type 头字段指出的媒体类型的解码。Content-Encoding 首要应用于在不丢失媒体类型标记的情况下对消息体进行压缩处理。

如果包体应用了多个编码，那么包体编码必须按顺序在这个字段中进行列出。

所有的 Content-Encoding 的值都是大小写不敏感的。IANA 是这个编码方式的注册机构。  
参见[H3.5]获得 Content-coding 的语法定义。

客户端可以在请求中进行包体的内容编码。服务端也可以在应答中进行内容编码。服务端必须只能应用客户端在请求中的 Accept-Encoding 头字段中列出的编码类型。

Content-Encoding 简写是 e。

示例：

Content-Encoding:gzip  
e: tar

### 21.13 Content-Language

参见[H14.12]。例如：

Content-Language: fr

### 20.14 Content-Length

Content-Length 头字段表示了消息体的大小，对消息的接受者指示消息体大小，以 10 进制单位数字表示。应用程序应该使用这个字段标识传输的消息体的大小，无需关心消息体的媒体类型。如果是基于流的通讯协议（比如 TCP）作为传输协议，必须使用本头字段。

消息的大小并不包含 CRLF 分开的头字段和包体。任何消息体，其大于或者等于 0 的 Content-Length 都是有效的值。如果消息中不包含包体，那么 Content-Length 必须设置为 0。

忽略 Content-Length 的能力可以简化类似 cgi 脚本创建的流程，这些 cgi 脚本一样的程序可以动态生成响应消息。

这个头字段的简写是 l。

例如：

Content – Length : 349  
l : 173

#### 20.15 Content-Type

Content-Type 头字段标识了发给对方的消息体媒体类型。” media-type” 是在[H3.7]中定义。如果消息体不为空，那么 Content-Type 头字段就必须存在。如果消息体是空的，而且本头字段是存在的，就表示了特定类型的媒体的包体长度是(比如空的音频文件)。

本头字段的简写是 c。

例如：

Content-Type: application/sdp  
c: text/html; charset=ISO-8859-4

#### 20.16 Cseq

请求中的 Cseq 头字段包含了一个单个的数字序列号和请求的方法。此序列号必须是以 32 位为单位的无符号整数。在 Cseq 的请求中，方法部分是大小写敏感的。Cseq 头字段的目的是为了在对话中对相关事务进行排序的，提供事务的唯一标志，并且用来区分新的请求和请求的重发流程。如果序列号相等，并且请求的方法相同，那么两个 Cseq 头字段就是相等的。

例如：

Cseq:4711 INVITE

#### 20.17 Date

Date 头字段包含了日期和时间。和 HTTP/1.1 不同的是，SIP 只支持最近的 RFC1123[20]格式的日期。就像在[H3.3]中，SIP 限制了在 SIP-date 中的时区是” GMT” ,但是在 RFC1123 中支持任意的时区。RFC1123 的日期是大小写敏感的。Date 头字段反应的时间是请求或者应答被发送的首次发生的时间。

Date 头字段可以用来简化没有后备电池提供时钟的终端系统，让他们能够获得当前的时间。但是因为是 GMT 格式的，所以，它要求客户端获悉，终端和 GMT 的偏移差。

例如：

Date : Date,13 Nov 2010 23:29:00 GMT

#### 20.18 Error-Info

Error-Info 头字段提供了错误状态响应应答的其他附加信息。

SIP UAC 具备多接口的能力，从 Windows 弹窗工具和 PC 端语音支持到通过网关接入的传统电话机或者其他终端。环境的具有从弹出的窗口 PC 界面，到仅支持语音的传统电话机或者网关连接支持的终端。强制服务器产生一个错误消息，与其选择发送一个带错误码和应答原因详情状态，并且对对端播放一段已录制语音，不如使用 Error-Info 头字段支持把以上两个关联信息都发送到对端。这样，UAC 就具备了一定的选择能力，然后 UAC 自己决定采用何种形式对呼叫方进行渲染处理。

UAC 可以把在 Error-Info 头中的一个 SIP 或者 SIPS URI 视为是在转发的一个 Contact 地址，并且据此生成一个新的 INVITE，这样可以创建了一个预录的声明会话。如果是非 SIP URI 地址，那么也可以渲染给用户。

例如：

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 头字段设置了消息（或者内容）超时后的相对时间。这个字段的准确定义是和 method 有关联关系。在 INVITE 中超时不会影响由此 INVITE 引起的实际会话时长。不过，会话描述协议可以提供一个能力支持，在一个会话期间对时长进行限定。

这个头字段的值是一个以秒计数整数，从 0 到  $(2^{32}) - 1$ ，从收到请求开始计数。

例如：

Expires:5

## 20.20 From

From 头字段指示请求初始方。这个地址可能和对话的初始方地址不同。被叫方对呼叫方发送的请求会使用被呼叫方在 From 头字段的地址。

选项” display-name” 是对人机界面提供的渲染展示。如果客户标身份是隐藏状态，那么系统应当使用” An onymous” 作为显示名字。即使是” display-name” 是空的，如果” addr-spec” 包含一个逗号，问号或者分号，那么就必须使用” name-addr” 格式。相关的格式讨论在第 7.3.1 节有具体描述。

如果 URI 相同，并且参数也相同，那么这两个 From 头字段就是相同的。如果扩展参数在一个头字段中存在，但是在另外一个头字段中不存在，当这两个头字段做比较时，这个参数将被忽略。这意味着显示名字，存在状态或者缺乏括弧都不会影响对比结果。

参见第 20.10 章节解析显示名称，URI 和 URI 参数，以及头字段参数的规则。

From 头字段的简写是 f。

例子：

```
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 头字段列举了和本次呼叫相关的或者返回的 Call-ID。这些 Call-ID 可以让客户端做缓存处理，然后包含在返回的呼叫头字段中。

这样的操作允许自动呼叫分发系统来路由返回到第一个呼叫的原始请求地点。这样的操作也支持了被叫方过滤呼叫，只有是从呼叫发起方的呼叫所返回的呼叫才能被接受。这个字段不是替代对请求验证。

例如：

In-Reply-To: 70710@saturn.bell-tel.com, 17320@saturn.bell-tel.com

### 20.22 Max-Forwards

Max-Forwards 头字段必须使用在任何一个 SIP 请求中使用，此头字段的目的是限制代理或者网关转发此请求到下游节点的代理或者或网管的数量。这个头字段非常有用，当客户端希望跟踪一个请求链路，这个请求链路出现的路由失败或者在中间链中出现循环的状态。

Max-Forwards 是一个 0-255 之间的整数，表示了在这个请求消息中允许被转发的剩余次数。每当服务器转发这个请求一次，按照计数就减一。建议的初始值是 70。当不能确定是否有循环路由的时候，必须在网元的头字段中增加此头字段。比如，一个 B2BUA-背靠背代理环境时就应该增加这个头字段。

例如：

Max-Forwards:6

### 20.23 Min-Expires

Min-Expires 头字段传递了一个最小的刷新时间，这个时间由那个服务器所控制的（soft-state）网元实现支持。这个包括被登记服务器所登记的 Contact 头字段。这个头字段包含

了一个以秒计数的整数，从 0 到 $(2^{32})-1$ 。关于这个头使用在 423(Interval Too Brief) 应答中，本头字段的用法在第 10.28, 10.3, 和 21.4.17 章节中有描述。

例如：

Min-Expires:60

非规范的其他补充，关于 soft-state 和 hard-state 对比说明，读者可查阅：

论文：A Comparison of Hard-state and Soft-state Signaling Protocols

<https://conferences.sigcomm.org/sigcomm/2003/papers/p251-ji.pdf>

#### 20.24 MIME-Version

参见[H19.4.1]

例如：

MIME-Version: 1.0

#### 20.25 Organization

Organization 头字段传输 SIP 请求签发或者应答的网元网元所属的组织名字。

这个字段可以用来支持客户端软件过滤此呼叫。

例如：

Organization: Boxes by Bob

#### 20.26 Priority

Priority 头字段标识了客户端收到请求的紧急程度或者优先级。Priority 头字段描述了 SIP 请求应当优先处理人工请求或者其代理发来的请求。举例来说，通过优先级处理，可能可以将一个决定分解为呼叫转发和呼叫接受。对于这些决定来说，如果消息没有指定包含 Priority 字段，那么处理此呼叫时应当将此呼叫视为“normal”优先级来处理。Priority 头字段不影响通讯资源使用的优先顺序，比如路由器上的包转发的优先级或者访问 PSTN 网

关电路的优先级。此头字段支持“non-urgent”，“normal”，“urgent”和“emergency”取值，另外的取值可以在其他处定义。规范建议“emergency”只用于影响到生命、身体、或者财产危急时候才使用。否则在其他情况下，本头字段没有定义额外的语义。

在 RFC2076[38]中，定义了“emergency”。

例如：

Subject: A tornado is heading our way!

Priority: emergency

或者

Subject: Weekend plans

Priority: non-urgent.

#### 20.27 Proxy-Authenticate

Proxy-Authenticate 头字段用来进行认证使用的。这个头字段的具体用法在[H14.33]中定义。参见第 22.3 章节关于本字段的细节讨论。

例如：

```
Proxy-Authenticate: Digest realm="atlanta.com",
domain="sip:ssl.carrier.com", qop="auth",
nonce="f84f1cec41e6cbe5aea9c8e88d359",
opaque="", stale=FALSE, algorithm=MD5
```

#### 20.28 Proxy-Authorization

Proxy-Authorization 头字段或者头字段允许客户端向一个要求认证的 proxy 代理确认自己（或者确认它的用户）的身份。一个 Proxy-Authorization 头字段消息由与 UA 安全消息构成，包含对代理认证的用户代理信息和/或本请求资源的 realm 域值。

参见第 22.3 章节关于这个头字段/域的定义。

此头字段，连同 Authorization 头字段并不遵循常用的多头字段名（多个相同头字段名的合并）的规则。虽然没有用以逗号分割的列表，这个头字段名仍然可以多次出现，并且一定不能用第 7.3.1 章节描述的通常规则将这些头字段合并成为一个头字段。

例如：

```
Proxy-Authorization: Digest username="Alice", realm="atlanta.com",
                     nonce="c60f3082ee1212b402a21831ae",
                     response="245f23415f11432b3434341c022"
```

#### 20.29 Proxy-Require

Proxy-Require 头字段/域用来表示 proxy 代理对某些敏感功能支持，要求一定支持的相关功能特性。参考 20.32 关于这个头字段的使用机制说明。

例如：

```
Proxy-Require: foo
```

#### 20.30 Record-Route

Record-Route 头字段/域是 proxy 在请求中插入的头，用来强制在 dialog 对话中的后续请求必须经过此代理呼叫路径。本头字段/域的用法示例在第 16.12.1 章节中有描述。

例子：

```
Record-Route: <sip:server10.biloxi.com;lr>,
               <sip:bigbox3.site3.atlanta.com;lr>
```

#### 20.31 Reply-To

Reply-To 头字段包含了逻辑上返回 URI，这个可以和 From 头字段不同。例如，URI 可以用来表示一个未接电话或者未建立的会话。如果用户希望保留匿名，那么这个头字段应当从请求中过滤或者使用其他方式来填充避免暴露任何隐私信息。

即使” display-name” 为空，如果” addr-spec” 包含了逗号、问号、或者分号，就必须使用” name-addr” 的格式来填写。这个语法在第 7.3.1 章节中定义。

例如：

```
Reply-To: Bob <sip:bob@biloxi.com>
```

### 20.32 Require

Require 头字段/域是 UAC 用于通知 UAS 相关支持选项，这些选项是 UAC 通知 UAS 需要支持的选项以进行后续的请求处理。虽然这是一个可选的头字段，但是如果 Require 头字段存在，那就一定不能忽略此头字段。

头字段包含一个 option tag 的列表，这个列表在第 19.2 章节中描述。每一个 option tag 定义了一个 SIP 扩展，这些扩展必须理解和处理请求。通常情况下，此定义表示了一个需要支持的扩展头字段的集合。遵从本规范相应的 UAC 规范必须仅包含 option tag 相应的 RFC 扩展。

例如：

```
Require: 100rel
```

### 20.33 Retry-After

Retry-After 头字段/域可以和 500 (Server Internal Error) 或者 503 (Service Unavailable) 响应表示对正在请求的客户端来说大概时长本服务仍会处于不可用状态，并且可以和 404(Not Found), 413(Request Entity Too Large), 480(Temporarily Unavailable), 486(Busy Here), 600 (Busy), 或者 603(Decline) 需要使用来表示何时被叫方会恢复可用状态。这个字段的值是以秒为单位的正整数（十进制），是从响应后开始计数的一个正整数值。

可选说明可以用额外消息指示一个回呼时间。可选参数”duration”参数表示了被叫方从开始初始可达状态恢复到可达状态的时间长度。如果没有定义可选时长，那么此服务被视为是永远有效。

例如：

```
Retry-After: 18000;duration=3600  
Retry-After: 120 (I'm in a meeting)
```

#### 20.34 Route

Route 头字段/域用于强制请求经过一个 proxy 代理的路由列表路径。Route 头字段的使用示例在第 16.12.1 章节中定义：

例如：

```
Route: <sip:bigbox3.site3.atlanta.com;lr>,  
<sip:server10.biloxi.com;lr>
```

#### 20.35 Server

Server 头字段包含了关于 UAS 处理请求所使用的软件信息。

暴露服务器的具体软件版本可能会导致服务器由于某个特定软件安全漏洞导致服务器受到安全攻击。用户部署时应该支持 Server 头字段是一个可配置的选项。

例如：

```
Server: HomeServer v2
```

#### 20.36 Subject

Subject 头字段对呼叫属性提供了一个汇总或指示说明，允许呼叫实现过滤而不用解析会话描述内容。作为一个邀请，会话描述并不需要使用同样的主题标识。

Subject 的缩写是 s。

例如：

Subject: Need more boxes  
s: Tech Support

#### **20.37 Supported**

Supported 头字段枚举了 UAC 或者 UAS 所支持的扩展。

Supported 头字段包含了一个 option tag 的列表，在第 19.2 章节描述了 option tag, UAS 或者 UAC 可以支持这些 tag。遵循本规范的 UA 必须仅包含标准 RFC 扩展的 option tag。如果本字段为空，这表示无任何扩展支持。

Supported 头字段的缩写是 k。

例如：

Supported: 100rel

#### **20.38 Timestamp**

Timestamp 头字段/域描述了当 UAC 对 UAC 发送请求时的时间戳。

参见第 8.2.6 章节中关于如何为请求生成一个包含此头字段的应答。虽然没有定义本字段的标准流程，生成响应的流程允许对扩展应用或者 SIP 应用获得 RTT 预估时间。

例如：

Timestamp:54

### 20.39 To

To 头字段定义了请求的逻辑接收者。选项” display-name” 用来表示渲染到客户端界面的信息。” tag” 参数提供了一个基本的对话确认身份机制。参见 19.3 节关于” tag” 参数的描述。

To 头字段的对比和对 From 头字段的对比是完全相同的。关于对 display name, URI 和 URI 参数, 以及头字段的参数解析规则, 参见第 20.10 章节说明。

To 头字段的缩写是 t。

以下是一个有效 To 头字段的举例：

```
To: The Operator <sip:operator@cs.columbia.edu>;tag=287447  
t: sip:+12125551212@server.phone2net.com
```

### 20.40 Unsupported

Unsupported 头字段/域列出了 UAS 不支持的功能列表。参见第 20.32 章节。

例如：

```
Unsupported:foo
```

### 20.41 User-Agent

User-Agent 头字段包含了发起请求的 UAC 信息。本头字段的语义在[H14.43]做了描述。

暴露了 UA 所使用的版本号可能会导致由于这个版本的安全漏洞而受到攻击。所以在部署时应该让 User-Agent 头字段支持可配置选项。

例如：

```
User-Agent: Softphone Beta1.5
```

## 20.42 Via

Via 头字段是用来指示请求当前经过的处理路径，并且表示了响应所应该经过的路径。在 Via 头字段的 branch ID 参数提供了事务标识符，并且 proxy 代理用 branch ID 来检查是否存在循环路由。

Via 头字段包含了用于发送消息的传输协议，客户端主机名或者网络地址，可能包含了接收应答所使用的端口号。Via 头字段还可以包含其他参数，例如，参数“maddr”，“ttl”，“received” 和“branch”，这些定义和使用方式在其他节中进行描述。对于遵循本规范的实现方式来说，这个 branch 参数的值必须以“magic cookie”“z9hG4b K”开头(第 8.1.1.7 章节)。

这里定义的传输协议是“UDP”，“TCP”，“TLS”，和“SCTP”，“TLS”意思是基于 TCP 的 TLS。当请求发送到一个 SIPS URI 地址时，协议仍然标记为“SIP”，但是传输协议是 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 头字段的缩写格式是 v。

在这个例子中，消息从多主机地址(multi-homed)发出，有两个地址，192.0.2.1 和 192.0.2.207。发送方使用了错误的网络接口。Erlang.belltelephone.com 发现了这个错误匹配，并且给上一个节点的 Via 增加了一个参数，包含了数据包实际来源地址。

在 SIP URI 语法中，并不要求填写主机名或者网络地址和端口号。特别是，允许在“：“或者“/”两边的 LWS。

例如：

```
Via: SIP / 2.0 / UDP first.example.com: 4000;ttl=16  
;maddr=224.2.0.1 ;branch=z9hG4bKa7c6a8dlze.1
```

即使本规范要求所有的请求中都包含 branch 参数，在本头字段的 BNF 描述中，branch 参数是可选的。这样就确保了 RFC2543 网元可以互相兼容，因为 RFC2543 没有插入 branch 参数。

如果发送所使用的协议和发送的域相同，并且都有相同的参数集合，并且参数都相同，那么所有参数值就是相同的。

#### 20.43 Warning

Warning 头字段/域用来传输响应状态的其他附加消息。Warning 头字段值是响应中携带此头字段值，并且包含了一个三位数的告警代码，主机名，和告警文本消息。

“warn-text” 应当是一种自然语言，为用户接收应答时候提供响应。此决定可以基于任何可用消息来决定，例如，用户定位信息，Accept-Language 域，或响应中的 Content-Language 等。默认语言是 i/default [21]。

以下列出了当前定义的“ warn-code”，并且提供了以英文描述方式所推荐的 warn-text 内容信息。这些告警信息描述了会话描述中的各种可能的失败状态。第一个 warn-code 中的数字是“ 3” 表示这是一个 SIP 规范的告警信息。告警信息 300 到 329 是预留的信息用于表示在会话描述中的错误关键词，330 到 339 是会话描述中和基本网络服务相关告警信息，370 到 379 是关于会话描述中关于 QoS 参数数值相关的告警，390 到 399 是以上未指示的杂项警告信息。

300 Incompatible networkprotocol: (不兼容的网络协议)，在会话描述中存在的一个或者多个不适用的网络协议。

301 Incompatible networkaddress formats(不兼容的网络地址格式) : 在会话描述中存在一个或者多个非法的网络地址。

302 Incompatible transportportocol(不兼容的传输协议) : 在会话描述中存在一个或者多个不兼容的传输协议。

303 : Incompatible bandwidth units (不兼容的计量单元) : 在会话描述中存在一个或者多个不支持的计量单元。

304 Media type not available (媒体类型不可用) : 在会话描述中存在一个或者多个不可用的媒体类型。

305 Incompatible media format (媒体格式不兼容) : 在会话描述中存在一个或者多个不兼容的媒体格式。

306 Attribute not understood (媒体属性不支持) : 在会话描述中存在一个或者多个不支持的媒体属性。

307 Session description parameter not understood (会话描述参数不支持) : 不支持列表列出的会话描述参数。

308 Multicast not available (组播不可用) : 目前用户所处节点不支持组播。

309 Unicast not available (单播不可用) : 目前用户所处节点不支持单播通信(通常是因为防火墙的原因)。

370 Insufficient bandwidth(带宽不足): 会话描述的带宽要求或者媒体所要求的带宽超过限制设置）。

399 Miscellaneous warning (杂项告警) : 此告警信息可以包含任意信息，这些信息将展现给用户或作为日志记录。接收告警信息的系统一定不能执行任何自动操作。

1xx 和 2xx 消息是 HTTP/1.1 的消息。

其他的“ warn-code” 通过 IANA 定义，在第 27.2 章节有额外说明。

例如：

```
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 头字段包含了认证验证信息，参见 第 22.2 章节相关详细说明。

例如：

```
WWW-Authenticate: Digest realm="atlanta.com",
domain="sip:boxesbybob.com",qop="auth",
nonce="f84f1cec41e6cbe5aea9c8e88d359",
opaque="", stale=FALSE, algorithm=MD5
```

# 21 Response Codes

响应代码和 HTTP/1.1 的响应码是一致的，并且扩展了 HTTP/1.1 的响应码。不是所有的 HTTP/1.1 响应码都可以适用于此规范，只有在这里列出的才是可适用的。不应当使用其他的 HTTP/1.1 响应码。而且，SIP 也定义了新的级别的响应码系列，6xx。

## 21.1 Provisional 1xx

临时响应，也称为消息响应，表示连接的服务器正在执行进一步的处理，并且还没有确认的响应。如果服务器希望耗用 200 毫秒以上时间来获得最终的响应，则服务器发送 1xx 响应。注意，1xx 响应并不是可靠的传送。这些临时响应从不会引起客户端发送 ACK。临时（1xx）响应可以包含带有会话描述的消息体。

### 21.1.1 100 trying

该响应表示此请求已经被下一个跳点的服务器接收，并且正在处理此呼叫中的某些未指定的执行流程(例如，正在查询数据库流程)。此响应与其它所有的临时响应一样，由 UAC 停止重发 INVITE 请求。100 trying 响应和其他的临时响应不同，此临时响应从不通过有状态代理向上游前转。

#### 21.1.2 180 Ringing

正在接收 INVITE 的 UA 正在对用户进行提示。此响应可用于初始化本地回铃音。

#### 21.1.3 181 Call Is Being Forwarded

服务器可以使用该状态代码表示呼叫正在被转发到不同的目的地组集。

#### 21.1.4 182 Queued

被呼叫方暂时不可用，但是服务器已确定将该呼叫转入等待队列，而不是直接拒绝此呼叫。当被呼叫方为可用状态后，它会返回恰当的相应最终状态响应。原因解析可以提供关于此呼叫状态细节，比如，“等待队列中有 5 个呼叫；预计等待时间是 15 分钟”。服务器可以签发多个 182(队列) 响应来更新关于呼叫方的有关呼叫等待队列的状态。

#### 21.1.5 183 Session Progress

使用 183(会话处理中) 响应传输有关未分类的呼叫处理信息。可以使用原因解析短语、头字段或者消息体传输有关呼叫进展的更加详细的状态。

### 21.2 Successful 2xx

请求是成功的。

#### 21.2.1 200 OK

请求已成功。在响应中的返回消息取决于请求的 methods。

## 21.3 Redirection 3xx

3xx 响应提供有关用户新位置信息，或者相关的能够满足呼叫的可选服务信息。

### 21.3.1 300 Multiple Choices

请求中的地址解析为若干选择，每一个选择都有自己具体的地址，用户（或者 UA）能够选择偏好的终端进行通信，并将此请求重定向到此位置。

响应中可以包括消息体，此消息体包含了资源属性列表和定位信息，如果 Accept 请求头字段允许支持这些资源的话，此用户或者 UA 能够从列表中选择的最合适的资源。但是，没有为此消息体定义 MIME 类型。

这些选择应该在 Contact 字段(第 20.10 节) 作为列表被列出。不像 HTTP，SIP 响应可以包含多个 Contact 字段或者在 Contact 字段中包含一个地址列表。UA 可使用此 Contact 头字段值实现自动重定向或者要求用户确认此选择。但是，本规范没有定义任何关于自动选择的标准。

如果能够从不同的位置抵达被呼叫方，并且服务器不能或者不推荐代理此请求，此状态响应也适用。

### 21.3.2 301 Moved Permanently

没有在 Request-URI 地址中找到用户，而且正在请求的客户端将用 Contact 头字段（第 20.10 章节）提供的新地址重试请求。请求方将使用新的值更新本地地址目录、通讯录和用户位置缓存，并且向列表中的地址重定向后续请求。

### 21.3.3 302 Moved Temporarily

正在请求的客户端将使用 Contact 头字段(第 20.10 章节) 提供的新地址重试请求。新请求的 Request-URI 将使用响应中的 Contact 头字段值。

Contact URI 有效时长可以通过 Expires(第 20.19 章节) 头字段或者 Contact 头字段的 expires 参数表示。代理和 UA 都可以缓存此 URI 来支持超时时长。如果没有明确说明超时时长，用于递归的地址仅一次有效，而且一定不能被缓存用于未来的事务处理。

如果来自 Contact 头字段的 URI 缓存失败，可以单独再使用重定向请求中的 Request-URI 尝试。

临时 URI 的过期时间超时可能早于超时时间，并且新的临时 URI 可能可用。

#### **21.3.4 305 Use Proxy**

必须通过 Contact 字段提供的代理访问已请求的资源。Contact 字段提供了代理的 URI。接收方希望通过代理重复该单次请求。只有 UAS 才能生成 305 (Use Proxy) 响应。

#### **21.3.5 380 Alternative Service**

呼叫没有成功，但是可能存在可选服务。响应的消息体中描述了可选服务。这里没有定义这样的消息体格式，此消息体格式可能称为未来的规范讨论主题。

### **21.4 Request Failure 4xx**

4xx 响应定义了来自具体服务器的错误响应。客户端将不应该重试未经过修改的请求(例如，添加相应的签权信息)，但是，同样的请求对其他服务器发送的话，请求可能会成功。

#### **21.4.1 400 Bad Request**

因为不正确的语法格式，此请求不能被正确解析理解。原因短语将详细说明语法问题，例如，“Missing Call-ID header field”。

#### **21.4.2 401 Unauthorized**

请求要求用户认证。该响应是由 UAS 和注册服务器签发，这里，代理服务器使用 407(Proxy Authentication Required)。

#### **21.4.3 402 Payment Required**

预留为未来使用。

#### **21.4.4 403 Forbidden**

服务器端理解此请求，但是拒绝执行此请求。Authorization 授权无任何帮助，并且此请求不应该被重复发送。

#### **21.4.5 404 Not Found**

服务器已有明确信息，该用户不存在于 Request-URI 指定的域中。如果 Request-URI 中的域不匹配请求的接收方所管理的任何域，那么也会返回该状态。

#### **21.4.6 405 Method Not Allowed**

理解 Request-Line 中设定的 method，但是 Request-URI 所确定的地址未被允许。响应必须包括 Allow 头字段，该头字段包含对已指示地址可用 methods 的列表。

#### **21.4.7 406 Not Acceptable**

被请求确认的资源仅能够产生响应实体的能力，该实体的内容属性不能被已发请求中 Accept 头字段规定的属性接受。

#### 21.4.8 407 Proxy Authentication Required

该代码与 401 (未授权) 相似，但是指示了客户端必须首先向代理自我认证。SIP 访问认证将在第 26 章和第 22.3 章中说明。

此状态代码是在访问通信信道的应用环境中使用（例如，电话网关），而不是使用在被呼叫方要求认证的环境中。

#### 21.4.9 408 Request Timeout

服务器不能在合适的时间范围内产生响应，例如，它不能及时决定用户位置。稍晚，客户端可在不作修改的情况下重复该请求。

#### 21.4.10 410 Gone

服务器端已请求的资源不可用，并且不知道转发地址。该状态希望是一个永久的。如果服务器不知道，或者没有相应设施来确定此状态是否是永久状态，那么应该使用状态代码 404 ((Not Found) 来替代。

非规范补充说明：

关于 411 响应代码说明：

<https://mailarchive.ietf.org/arch/msg/siz4R6b53eRX1kKt3ePGNXWsD60/>

关于 412 在其他扩展中支持

<https://www.rfc-editor.org/rfc/rfc3903#section-11.2>

#### 21.4.11 413 Request Entity Too Large

因为请求实体超过服务器愿意或能够处理的大小范围，服务器拒绝处理请求。服务器可以关闭连接防止客户端继续发送请求。

如果此状态是暂时的，服务器将包含 `Retry-After` 头字段表示此状态是暂时状态，在此后客户端可以重试。

#### **21.4.12 414 Request-URI Too Long**

因为 Request-URI 的长度超过服务器可以解析的范围，因此服务器拒绝服务此请求。

#### **21.4.13 415 Unsupported Media Type**

服务器拒绝对此请求进行服务，因为服务器不理解已请求的 `method` 中的消息体格式。服务器必须返回可接受的消息体格式列表，取决于具体问题所呈现的内容，格式使用 `Accept`、`Accept-Encoding` 或者 `Accept-Language` 头字段标识。在第 8.1.3.5 章节中介绍了 UAC 处理该响应的流程。

#### **21.4.14 416 Unsupported URI Scheme**

因为服务器不理解请求中的 Request-URI 的 URI 的模式，服务器不能处理请求。在 8.1.3.5 章节中介绍了客户端处理该响应的流程。

#### **21.4.15 420 Bad Extension**

服务器不理解在 `Proxy-Require`（第 20.29 节）或者 `Require`（第 20.32 节）头字段中规定的协议扩展。在响应中，服务器必须在 `Unsupported` 头字段中包括不支持的扩展列表。在第 8.1.3.5 章节中介绍了 UAC 处理该响应的流程。

#### **21.4.16 421 Extension Required**

UAS 需要特殊的扩展来处理此请求，但是在请求中的 `Supported` 头字段未列出该扩展。带该状态代码的响应必须包含一个 `Require` 头字段，在该头字段中列出所需要支持的扩展。

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 头字段中没有列出所期望的扩展，服务器端应该处理此请求，通过基准 SIP 处理能力和客户端所支持的任何扩展来进行。

#### 21.4.17 423 Interval Too Brief

因为请求刷新资源的超时时间太短，服务器拒绝了此请求。注册服务器可以使用该响应拒绝某些注册，这些注册中的 Contact 头字段超时时间设置太短。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 Temporarily Unavailable

成功联系到了被呼叫方所属系统，但是被呼叫方处于不可用状态（例如，被呼叫方没有登录、被呼叫方已登录但是处于不能与之通信的状态、或者被呼叫方启用了“DND-免打扰”功能）。响应可以在 Retry-After 头字段中指示一个更恰当的呼叫时间。用户也可能在别处有效(可能该服务器不知道)。原因短语应该指示更准确的说明来解释为什么此被呼叫方不可用。UA 可以设置该值。可以使用状态 486 (Busy Here) 来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，一个重定向或代理服务器通过 Request-URI 识别 用户，但是该状态码不能含有对此用户的当前有效转发地址。

#### 21.4.19 481 Call/Transaction Does Not Exist

该状态码指示 UAS 已收到了一个请求，此请求不能匹配现有的 dialog 或者事务。

#### 21.4.20 482 Loop Detected

服务器已经检测到网络回环（参见第 16.3 节第 4 项）。

#### 21.4.21 483 Too Many Hops

服务器收到了一个请求，在此请求中的 Max-Forwards 头字段值为 0(参见第 20.22 章节)。

#### 21.4.22 484 Address Incomplete

服务器收到了一个请求，此请求中携带了不完整的 Request-URI 信息。附加信息应该在原因短语中提供。

该状态代码允许重叠拨号。使用重叠拨号，客户端不知道拨号字符串的长度。它发送正在增加长度的字符串，提示用户不断输入字符串，直到不再接收 484(地址不完整)状态响应。

#### 21.4.23 485 Ambiguous

Request-URI 不明确。此响应可以在 Contact 头字段中包含可能的比较明确的地址列表。显示可选择地址会侵犯用户或者组织的隐私。服务器端必须支持可配置的选项，响应时使用状态码 404 (Not Found) 响应或者阻止对不明确的 Request-URI 提供可能的选项列表。

响应示例，针对带 Request-URI sip: lee@example.com 请求：

```
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箱系统提供主要的功能。因为其语义不同，使用了与 3xx 分离的状态代码：对于 300 来说，假设提供的选择将到达相同的用户或者服务。同时自动选择或者按序查找对 3xx 响应来说也是合理的，485 (Ambiguous) 响应则需要用户干预。

#### 21.4.24 486 Busy Here

成功联系到了被呼叫者的终端系统，但是被呼叫者当前不愿意或者不能在该终端系统接受额外的呼叫。此响应可以在 Retry-After 头字段中指示更合适的时间进行呼叫。可以在别处访问该用户，比如通过语音邮箱服务。如果客户端知道其它终端系统都不能接受该呼叫的话，可以使用效应状态 600 (Busy Everywhere) 替代。

#### 21.4.25 487 Request Terminated

通过 BYE 或者 CANCEL 请求结束该请求。CANCEL 请求本身从不返回该响应。

#### 21.4.26 488 Not Acceptable Here

该响应与 606 (Not Acceptable) 有同样的含义，但是仅适用于 Request-URI 访问的具体的资源，并且此请求可以在其他地方成功。

包含媒体能力描述的消息体可能出现在响应中，该消息体根据 INVITE 中 Accept 头字段格式化后的结果（如果不存在 INVITE 的话，或者 application/sdp 结果），这个消息体和 OPTIONS 请求返回的 200(OK)响应中的消息体相同。

#### 21.4.27 491 Request Pending

UAS 接收了此请求，在相同 dialog 中，UAS 已有待处理的请求。第 14.2 节中描述了如何解决这种特别的情况。

#### 21.4.28 493 Undecipherable

UAS 接收的请求中包含加密 MIME 体，接收方没有能力对此 MIME 体进行处理或不能提供正确的解密密钥。该响应可以包含单个消息体，在此消息体中提供正确的公钥，该公钥用于向 UA 发送的 MIME 体加密使用。第 23.2 章节介绍了此响应码的用法。

## 21.5 Server Failure 5xx

5xx 响应码是当服务器本身发生错误时提供的错误响应代码。

#### 21.5.1 500 Server Internal Error

服务器面临意外状态，此状态阻止它完成请求。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。

如果此状态是临时状态，服务器指示客户端在请求中使用 Retry-After 支持说明何时重试请求。

#### 21.5.2 501 Not Implemented

服务器不支持实现请求所需的功能。此响应也适用于当 UAS 不能识别请求 method，并且也不能为任何用户提供请求 method 能力的支持。(不管 method，代理转发所有的请求)。

注意，当服务器识别了 method，但是不允许或者不支持该 method 时，则发送 405(Method Not Allowed)。

#### 21.5.3 502 Bad Gateway

当服务器此时作为网关或者代理服务器工作时，尝试实现请求过程中，服务器收到了来自下游服务器的无效响应。

#### 21.5.4 503 Service Unavailable

因为服务器目前处于临时过载或者维护状态，暂时不能处理此请求。在 Retry-After 头字段中，服务器可以指示客户端何时重试请求。如果没有提供 Retry-After 头字段，客户端必须视为客户端接收了 500 响应(Server Internal Error)。

正在接收 503 响应 (Service Unavailable) 的客户端 (代理或 UAC) 应该尝试向备选服务器转发此请求。如果 Retry-After 头字段存在，在 Retry-After 头字段中规定的时间段内，此客户端 (代理或者 UAC) 不应该向备选服务器转发任何其它的请求。

服务器可以拒绝连接或者丢弃此请求，而不是用 503(Service Unavailable)响应码响应。

#### **21.5.5 504 Server Time-out**

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及时地接收到响应。如果在上游服务器的 Expires 头字段规定的时间段内没有收到响应，则使用 408 (Request Timeout) 替代。

#### **21.5.6 505 Version Not Supported**

服务器不支持或者拒绝支持请求中使用的 SIP 协议版本。除了此错误消息之外，服务器正在指示服务器端不能够或不愿意使用与客户端相同的主版本来完成此请求。

#### **21.5.7 513 Message Too Large**

请求中的消息长度超过了服务器本身的处理能力，服务器不能处理此请求。

### **21.6 Global Failures 6xx**

6xx 响应指示服务器端已有针对具体用户的非常明确的信息，不仅仅是已在 Request-URI 中指示的具体实例。

#### **21.6.1 600 Busy Everywhere**

已成功联系到了被呼叫方的终端系统，但是被呼叫方正在忙状态，并且不希望在此时间接听此呼叫。回复的响应可以在 Retry-After 头字段中指示呼叫合适时间。如果被呼叫方不希望显示拒绝呼叫的原因，被呼叫方可使用状态代码 603 (Decline) 替代。此状态码仅适

用于客户端知道其它终端（例如，语音邮箱系统）都不会回复此请求的情况下，返回此状态响应。否则，应该返回 486 (Busy Here)。

#### 21.6.2 603 Decline

已成功联系到被呼叫方设备机器，但是用户明确表示不希望或者不能够介入呼叫。回复的响应可以在 Retry-After 头字段中指示更合适的呼叫时间。此状态码仅适用于客户端知道其它终端都不会回复此请求的情况下，返回此状态码。

#### 21.6.3 604 Does Not Exist Anywhere

服务器有确认信息，确认消息说明，在 Request-URI 中指示的用户在任何其他地方都不存在。

#### 21.6.4 606 Not Acceptable

已成功访问用户代理，但是会话描述的某些特征不能被接受，例如，已请求的媒体、带宽或者地址方式。

606 响应 (Not Acceptable) 的意思是，用户希望通过通信，但是不能完全地支持会话描述。606 响应 (Not Acceptable) 可以在 Warning 头字段中包含一个原因列表来描述不支持会话描述的原因。告警原因码在第 20.43 节中有介绍。

在响应中可以出现一个消息体，消息体中包含媒体能力支持的描述，该消息体根据 INVITE (或者 application/sdp，如果 INVITE 不存在) 中的 Accept 头字段格式化处理，和 OPTIONS 请求的 200 (OK) 响应中的消息体相同。

系统部署中希望不要经常进行协商，并且当邀请新的用户加入现有的会议时，再进行协商是不可能的。这取决于邀请发起人，发起人决定是否对 606 (Not Acceptable) 响应做出响应执行。

此状态码仅适用于客户端知道其它终端都不会回复此请求的情况下，则返回此状态响应。

## 22 Usage of HTTP Authentication-HTTP

SIP 为身份验证提供了无状态，基于挑战的机制，该机制以 HTTP 中的验证为基础。任何时候，代理服务器或者 UA 接收到请求（第 22.1 节中涉及的情况除外），代理服务器或者 UA 都可以挑战请求发起方提供身份确认。一旦发起方身份确认以后，请求接收方将探知此正在被挑战的用户是否被授权发起请求。在本规范中，无可推荐和被讨论的授权系统。

本章中描述的 ““Digest” 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

和安全性保护。为防止主动攻击者修改 SIP 请求和响应，需要采用除 Digest 以外的其它保护措施来实现。

注意，由于其弱安全性特征，因此 “Basic” 认证的用法已经废止。服务器不能接受使用 “Basic” 授权模式的安全认证，而且服务器也一定采用 “Basic” 挑战机制。此修改来自于 RFC2543。

### 22.1 Framework

SIP 认证的架构和 HTTP (RFC2617[17]) 的架构非常相似。具体来说，对认证模式、认证参数、挑战、网络域、域值和安全的 BNF 是一样的（尽管不允许使用 “Basic” 作为模式）。在 SIP 中，UAS 使用 401 响应 (Unauthorized) 来挑战 UAC 的身份。另外，注册服务和重定向服务器可以使用 401 响应 (Unauthorized) 进行认证，但是代理一定不能 401 响应来进行认证，它可以使用 407 响应 (Proxy Authentication Required) 来替代。对于在各种消息中所包含的 Proxy-Authenticate、Proxy-Authorization、WWW-Authenticate 和 Authorization 的要求，和 RFC2617[17]中描述的是一样的。

因为 SIP 没有典型的根 URL 的概念，因此，在 SIP 中，保护机制的解释是不相同的。realm 网络域字符串单独定义了保护域的机制。这一点和 RFC2543 是不同。在 RFC2543 中，Request-URI 和域共同定义了保护域的机制。

因为 UAC 发送的 Request-URI 和挑战服务器接收的 Request-URI 可能是不同的，甚至于，UAC 可能也不知道 Request-URI 的最终格式，因此前面讨论的保护域的定义会产生一些混淆。另外，前面讨论的定义是取决于 Request-URI 中 SIP URI 的状态，此状态看起来排除了可选的 URI 模式（例如，tel URL）。

接收了请求，将进行认证的用户代理和代理服务器的操作者必须遵守以下指导，此指导是为其服务器网络域字符串创建的指导规定：

- 域字符串必须全局唯一的。本规范推荐域字符串包含主机名或者域名，遵循 RFC 2617[17]中 3.2.1 节的推荐。
- 网络域字符串应该是以用户可读的身份呈现，此呈现内容对于用户来说是可提交的。

例如：

```
INVITE sip:bob@biloxi.com SIP/2.0
Authorization: Digest realm="biloxi.com", <...>
```

通常来说，SIP 认证机制对于具体的网络域、保护域是有含义的。因此，对于 Digest 认证机制来说，每个这样的保护域有它自己的用户名和密码。如果服务器针对一个具体请求不需要认证，它可以接受默认用户名“anonymous”和无密码（password of “”）。同样，UAC 代表多用户，例如 PSTN 网关，这些 UAC 可以有它们自己指定的设备用户名和密码，而不用针对它们的网络域的具体用户账户。

当服务器可以合法地对大部分 SIP 请求挑战时，本规范定义了两个要求认证特殊处理的请求：ACK 和 CANCEL。

在一种认证模式中，认证模式使用响应消息来传输用于计算 nonces 值（例如，Digest）。如果某些请求没有收到响应的话，这样的请求就会产生一些问题，包括 ACK。因为这个原

因，任何服务器端接受了 INVITE 中的安全凭证，此服务器一定要接受此安全凭证的 ACK。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。服务器一定不能尝试挑战 ACK。

虽然 CANCEL method 确实产生了一个响应 (2xx) , 因为不能重提交这些请求，服务器一定不能尝试挑战 CANCEL 请求，因为不能重提交这些请求。一般来说，如果 CANCEL 请求和发送的已被取消的请求来自相同的跳点（提供了某种传输层或者网络层安全关联，具体描述参考第 26.2.1 章节内容），服务器应该接受此 CANCEL 请求。

当 UAC 收到了一个挑战，如果 UAC 设备不知道这个有待确认的所属网络域的安全信息，它应该向用户提交挑战中 realm 参数的内容（出现在 WWW-Authenticate 或 Proxy-Authenticate 头字段中）。当挑战预设设备时，为 UA 提供此网络域安全预设的服务供应商应该意识到用户将无机会为该网络域提供它们自己的安全信息。

最后，请注意，即使 UAC 能够通过恰当的安全消息确定了网络域的关联，仍然存在潜在问题：这些安全信息不再是有效的，或者挑战服务器将不接受这些安全信息，无论是什么原因（尤其是当提交的是“anonymous”和没有密码的情况下）。在这样的示例中，服务器可以重复它的挑战，或者使用 403 禁止响应替代。UAC 一定不能用刚被拒绝的安全凭证来重试请求（不过如果 nonce 失效的话，请求可以重试）。

## 22.2 User-to-User Authentication

当 UAS 收到了来自 UAC 的请求时，UAS 可以在处理请求之前对发起方进行认证。如果在请求中没有提供安全凭证（在 Authorization 头字段中），UAS 可以对发起方挑战验证，要求提供安全凭证，使用 401 (Unauthorized) 状态响应码拒绝此请求。

WWW-Authenticate 响应头字段必须包含在 401 (Unauthorized) 响应消息中。字段值至少由一个挑战和可使用的网络域参数组成，该挑战指示网络域可用的认证模式。

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"
```

当初始 UAC 收到了 401 (Unauthorized) 后，它应该，如果可以的话，使用恰当的安全凭证重新初始化请求。处理之前，UAC 可以要求来自于初始化用户的输入。一旦提供了认证凭证（或者用户直接提供，或在内部密钥环中发现此安全凭证），UA 将为已给定值的 To 头字段和 realm 缓存此安全凭证，并且为那个目的地的下一请求尝试重用这些安全凭证值。UA 可以用任何一种喜欢的方式缓存安全凭证。

如果不能为网络域确定安全凭证，UAC 可以使用用户名 “anonymous” 和无密码 (password of "") 尝试请求。一旦启动了安全凭证，任何希望向 UAS 或者注册服务器自认证的 UA--通常情况下，但不是必要，UA 在收到了 401 (Unauthorized) 响应后，可以通过在请求中包含 Authorization 头字段实现自认证。Authorization 字段值由安全凭证和认证支持所要求的参数和重放保护组成，安全凭证包含 UA 认证信息需要的安全凭证，该认证信息用于被请求资源网络域。

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"
```

在收到了 401 (Unauthorized) 或者 407 (Proxy Authentication Required) 响应后，当 UAC 使用它的安全凭证重新提交请求时，UAC 必须递增 Cseq 头字段值--就像当发送更新的请求时的操作。

### 22.3 Proxy-to-User Authentication

同样的处理方式，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对发起方进行认证。如果在在请求中没有提供安全凭证（在 Proxy-Authorization 头字段中），代理服务器可以对认证发起方进行挑战并要求提供安全凭证，通过拒绝此请求并且使用 407 (Proxy Authentication Required) 状态码指示认证要求。代理服务器必须使用代理可用的 Proxy-Authentication 头字段值针对已请求资源填写 407 填写 (Proxy Authentication Required) 消息。

[参考链接 17]中描述了 Proxy-Authentication 和 Proxy-Authorization 平行用法，其中有一点不同。代理一定不能向 Proxy-Authorization 头字段添加值。所有的 407 (Proxy Authentication Required) 响应必须被转发到上游，上游是 UAC 正在遵守的为其他资源服务的流程。它是 UAC 的责任，负责添加 Proxy-Authorization 头字段值，此头字段中包含被要求认证的代理网络域的安全凭证。

如果代理重提交请求，在新请求中增加 Proxy-Authorization 头字段值的话，将要求在新请求中递增 Cseq 值。但是，因为两个 Cseq 值不同，这样会造成已发初始请求的 UAC 会丢弃来自 UAS 的响应的问题。

当初始 UAC 收到了 407 (Proxy Authentication Required)，它应该，如果可以的话，使用恰当的安全凭证重发起请求。它应该遵守和为 “realm” 参数显示同样的流程，“realm” 参数给定用来响应 401。

如果不能确定网络域的安全凭证，UAC 可以使用用户名 “anonymous” 和无密码 (Password of "") 重试请求。

UAC 也应该将重发请求中使用的安全凭证执行缓存保存。

本规范推荐使用以下规则执行代理安全凭证缓存：

如果一个 UA 在 401/407 响应中收到了 Proxy-Authenticate 头字段值，并且和这个响应对应的请求中携带了特定的 Call-ID 值，此 UA 应该为所有包含相同 Call-ID 的后续请求处理此网络域安全凭证。这些跨越 dialog 的安全凭证一定不能被执行缓存处理；但是，如果用 UA 的本地 outbound 代理网络域配置了此 UA，当存在一个这样的配置的话，那么 UA 可以为此网络域（跨越 dialogs）执行缓存保存此安全凭证。注意，这表示 dialogs 中的未来请求可以包含安全凭证，此 Route 头路径中的代理不需要此安全凭证。

任何希望向代理服务器自认证的 UA-- 通常来说，但不是必要，在收到 407 (Proxy Authentication Required) 响应后，它可以通过在请求中包含 Proxy-Authorization 头字段来完成自认证流程。Proxy-Authorization 请求头字段允许客户端向要求认证的代理确认自己（或其用户）的身份。Proxy-Authorization 头字段值由安全凭证和/或所要求的资源网络域组成，安全凭证中包含对代理的 UA 认证信息。

Proxy-Authorization 头字段值仅适用于这样的代理，这个代理的 realm 通过 realm 参数来确定(此代理以前可能已经使用 Proxy-Authenticate 字段请求认证)。在一条代理链上使用了多个代理时，此链上的任何代理，它的 realm 值不能匹配 “realm” 参数中指定的值的话，此代理不能占用 Proxy-Authorization 头字段值。

注意，如果在 Proxy-Authorization 头字段中使用了不支持网络域的认证模式，代理服务器必须尝试解析所有的 Proxy-Authorization 头字段值来确定它们中是否存在代理服务器确认有效的安全凭证。在大型网络中，这样的操作可能会消耗大量时间，因此代理服务器应该使用一种认证模式，此认证模式在 Proxy-Authorization 头字段中指示支持网络域。

如果请求做了分叉处理（就像第 16.7 节中所描述的），各个代理服务器和/或 UA 可以希望挑战 UAC。在这种情况下，分叉代理服务器负责将这些挑战聚合在一个响应中。对于分叉代理请求所对应的响应收到的每个 WWW-Authenticate 和 Proxy-Authenticate 值来说，每个值必须被置入到单一响应中（此响应是由分叉代理发送到 UA 的响应）。这些头字段值的顺序是并不重要。

当代理服务器在响应中对请求发起挑战时，直到 UAC 已使用有效的安全凭证重试请求，代理服务器才代理此请求。分叉处理的代理可以同时向要求认证的多个代理服务器转发请求，直到初始 UAC 在它们各自的网络域中完成自认证为止，每个代理服务器才依次转发请求。如果 UAC 没有为每个挑战提供安全凭证，发起挑战的代理服务器将不会向可能是目的用户所在的 UA 转发此请求，因此，分叉处理的优势也大打折扣。

当在对 401 (Unauthorized) 或者 407 (Proxy Authentication Required) 的响应中重提交，这些响应中包含多个挑战时，一个 UAC 可以在每个 WWW-Authenticate 值中包含一个 Authorization 值，在每个 Proxy-Authenticate 值中包含一个 Proxy-Authorization 值，这些值用于 UAC 希望对挑战所提供的安全凭证。就像以上注意事项所讨论的那样，在请求中的多个安全凭证可以通过 realm 参数来加以区分。

这里存在一定的可能性，关联了同样 realm 网络域的多个挑战出现在了相同的 401 (Unauthorized) 或者 407 (Proxy Authentication Required) 中。这种情况是可能发生的。例如，当分叉请求抵达了共有网络域时，在同一管理域的多个代理使用此共有网络域。当它重试请求时，UAC 因此也可以在 Authorization 或者 Proxy-Authorization 头字段中使用相同 realm 参数值提供来多个安全凭证。相同的网络域应该使用相同的安全凭证。

#### 22.4 The Digest Authentication Scheme

本章节描述了 SIP 在应用 HTTP Digest 认证模式所需的修改和说明。SIP 模式的用法与 HTTP[参考链接 17] 的用法几乎完全相同。因为 RFC2543 是基于 RFC2069[39]中定义的 HTTP Digest 模式，支持 RFC2617 的 SIP 服务器必须保证它们向后兼容 RFC2069。在 RFC2617 中规定了向后兼容流程。注意，SIP 服务器不能接受或者请求 Basic 认证模式。

Digest 认证规则遵循[参考链接 17]中的定义，用 “SIP/2.0” 代替了 “HTTP/1.1”，另外还有以下不同：

- 挑战中包含的 URI 有以下的 BNF 格式：

```
URI = SIP-URI / SIPS-URI
```

- 在 RFC2617 中的 BNF 有一个错误，HTTP Digest 认证的 Authorization 头字段的 ‘uri’ 参数没有引号。（在 RFC2617 第 3.5 节中的示例格式是正确的。）对于 SIP 来说 ‘uri’ 必须有引号。
- digest-uri-value 的 BNF 格式是：

```
digest-uri-value = Request-URI ; 如在第 25 章中定义。
```

- 基于 Etag 选择 nonce 的示例流程不适用于 SIP。
- 在 RFC2617[参考链接 17]中关于缓存操作的文本不适用于 SIP。
- RFC2617[参考链接 17]要求服务器检查请求行中的 URI 和 Authorization 头字段中的 URI 是否指向同一个资源。在 SIP 文本中，因为在一些代理处转发，这两个 URI 可以用来区别不同的用户。因此，在 SIP 网络环境中，服务器可以检查 Authorization 头字段值中的 Request-URI 对应一个用户，此用户是服务器愿意为此用户接受前转或指引此请求，但是，如果这两个字段不相等，并不一定是失败的。
- 作为一个对 A2 值计算的阐述说明支持在 Digest 认证机制中的消息完整性保证，部署用户应该假设，当实体为空时（即当 SIP 消息中没有消息体时），实体的哈希解析为空字符串的 MD5 哈希，或：

```
H(entity-body) = MD5("") =
"d41d8cd98f00b204e9800998ecf8427e"
```

- RFC2617 规范提醒，如果没有发送 qop 指向时，不能在 Authorization（和扩展的 Proxy-Authorization）头字段中发送 cnonce 值。因此，任何依赖于 conoce（包括 “MD5-Sess” ）的算法，都要求发送 qop 指向。为了保持 RFC2617 向后兼容 RFC2069，在 RFC2617 中，qop 参数是可选参数；因为 RFC2543 是基于 RFC2069 发布的，为了支持客户端和服务器能够接收，qop 参数必须为可选状态。但是，服务器必须一直在 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，客户端必须在任何由此产生的 authorization 头字段中发送 qop 参数。

RFC2543 不允许使用 Authentication-Info 头字段（但它有效地使用在了 RFC2069 中）。因为 Authentication-Info 提供了在消息体上的完整性检查，并且提供了相互认证，我们现在允许使用该头字段。RFC 2617[参考链接 17]定义了在请求中使用 qop 属性的向后兼容机制。服务器端必须使用这些机制来确定客户端是否支持在 RFC2617 中的新的机制，这些新机制是没有在 RFC2069 中指定的。

## 23 S/MIME-MIME

SIP 消息传输 MIME 消息体，MIME 标准包括了一个机制用来保护 MIME 内容，确保其内容的完整性和安全性（包括 ‘multipart/signed’ 和 ‘application/pkcs7-mime’ MIME 类型，参见 RFC1847[参考链接 22]、RFC2630[参考链接 23]和 RFC2633[参考链接 24]）。但是，部署用户应该注意，网络环境中有一些少见的网络中间媒介（非典型代理服务器），这些网络中介需要查看或者修改 SIP 消息体（特别是 SDP），部署用户也应该注意，为了确保 MIME 的安全，防止这些中间媒介运行。

这样的应用特别针对某些类型的防火墙。

在 RFC2543 中定义的针对头字段和 SIP 消息体加密的 PGP 机制已经废弃。

### 23.1 S/MIME Certificates

用于确定终端用户的 S/MIME 使用的证书和服务器使用的证书有一个重要的不同点--这些证书声明持有者地址来确定证书持有者，而不是声明证书持有者的身份对应一个具体主机名称。该地址由 userinfo “@” 和 SIP 或者 SIPS URI 中的 domainname 部分构成（换言之，类似于电子邮件地址的格式，如 bob@biloxi.com），通常与用户的 address-of-record 一致。

这些证书还和用于签名或者对 SIP 消息体加密的密钥关联。消息体签名附带了发送方的私钥（发送方可以把它们的公钥使用恰当的方式包含在消息中），但是，使用计划接收消息的接收方的公钥对消息体加密。显然，发送者必须提前获知接收方的公钥来执行对消息体的加密处理。公钥能够存储在虚拟密钥环的 UA。

每个支持 S/MIME 的用户代理必须包含一个特定的适用于终端用户证书的密钥环。此密钥环应该在记录地址和相应的证书之间映射。随着时间的推移，当用户使用相同的记录地址计算信令的初始 URI (From 头字段) 时，用户应该使用相同的证书。

任何依赖于现存终端用户证书需要严格地限定在一个现实状态，目前实际上没有统一的权威组织能够为终端应用提供证书。但是，用户确实要求从熟知的证书机构获得证书。作为一种可选的方法，用户可以创建自签名证书。关于自签名证书使用将在第 26.4.2 章节中做进一步讨论。部署实施也可以在预设证书实现，部署中的所有 SIP 实体之间已在先前建立了信任关系。

除了上述讨论的获取终端用户证书的问题，还有一些集中目录来分发终端用户证书。但是，证书持有者应该以适当方式在任何公共目录发布它们的证书。同样，UAC 应该支持一种机制，这种机制可以导入（通过手动地或者自动方式）证书，在公共目录中找到的证书对应 SIP 请求的目的地 URI 地址。

### 23.2 S/MIME Key Exchange

SIP 本身也可以作为一种分发公钥的手段，通过以下方式来实现。

无论何时在 SIP 的 S/MIME 中使用了 CMS SignedData 消息，该消息必须包含证书，它用来对公钥进行验证。

当 UAC 发送了包含 S/MIME 消息体的请求时，这个 S/MIME 消息体初始化了 dialog 或者在 dialog 内容之外发送了一个非 INVITE 请求），UAC 应该构建消息体作为一个 S/MIME ‘multipart/signed’ 的 CMS SignedData 消息体。如果期望的 CMS 服务是

EnvelopedData（并且已知目标用户的公钥），此 UAC 将发送封装在 SignedData 消息中的 EnvelopedData 消息。

当 UAS 收到了一个包含证书的 S/MIME CMS 消息体的请求时，UAS 应该首验证此证书，如果可能的话，使用证书机构的有效根证书验证。UAS 还应该决定证书的主题（对于 S/MIME，SubjectAltName 将包含相应的身份），并且将该值与请求的 From 头字段进行比较。如果不能验证证书的话，其原因可能是证书是自签名证书或者未知证书机构签发的签名，或者如果证书是可验证的，但是它的主题不对应请求的 From 头字段，在处理之前，UAS 必须通知其用户证书的状态（包括证书的主题、它的签名者和密钥指纹信息）和明确的权限请求。如果证书验证成功，并且证书的主题和 SIP 请求的 From 头字段对应，或者如果用户（提示后）明确授权使用该证书，UAS 应该将证书添加到本地密钥环中，通过证书持有者的 address-of-record 来检索。

当 UAS 发送了一个包含 S/MIME 消息体的响应，该 S/MIME 消息体应答 dialog 中的第一个请求，或者 UAS 向 dialog 内容之外的非 INVITE 请求发送响应时，UAS 应该构建消息体作为 S/MIME 的 ‘multipart/signed’ CMS SignedData 消息体。如果期望的 CMS 服务是 EnvelopedData，UAS 应该发送封装在 SignedData 消息中的 EnvelopedData 消息。

当 UAC 收到了一个包含 S/MIME 消息体的响应时，该 S/MIME 消息体包括了证书，UAC 应该首先验证证书，如果可能的话，UAC 将使用恰当的根证书进行验证。UAC 应该决定证书的主题，将该值和响应中的 To 字段进行对比；虽然这两个值可能存在很大区别，但这并不一定说明是安全漏洞。如果证书不能验证成功，原因可能是证书是自签名证书或者未知机构签发的证书，则 UAC 必须在继续处理之前，通知其用户证书的状态（包括证书的主题、它的签名者和密钥指纹信息）和明确的许可请求。如果证书验证成功，并且证书的主题和响应中 To 头字段对应，或者如果用户（在通知后）明确授权使用此该证书，UAC 应该将证书添加到本地密钥环中，通过证书持有者的 address-of-record 来检索。

如果 UAC 没有在任何以前的事务中向 UAS 发送它自己的证书，UAC 应该使用 CMS SignedData 消息体支持它的后续请求或者响应。

在将来的一些场景中，当 UA 收到请求或者响应时，请求或者响应中包含的 From 头字段与 UA 密钥环中的值对应，UA 应该对比这些消息中所提供的证书和在它的密钥环中现有的证书。如果两者不一致，那么 UA 必须在继续处理信令之前，通知其用户关于相关证书的改变（换句话说，这可能说明存在潜在的安全漏洞）并且要求获得用户允许。如果用户授权了该证书，将把该证书和其 address-of-record 相关的前值添加到密钥环。

这里一定要注意，当使用自签名证书或者未知证书机构签发的证书时（它们会非常容易被攻击），该密钥交换机制不能保证密钥的安全交换。但是，作者观点认为，事实上，和被已经广泛使用的 SSH 应用相比较，该密钥的交换机制所提供的安全性比没有任何安全保障要好。第 26.4.2 章节中更加详细地解释了这些限制。

如果 UA 收到了 S/MIME 消息体，对于接收方来说，此消息体是通过公网未知公钥加密的，UA 必须使用 493 (Undecipherable) 响应拒绝请求。此响应应该在 MIME 消息体中，通过 ‘certs-only’ 的 “smime-type” 参数包含一个有效证书（如果可能，和在拒绝请求的 To 头字段中所提供的 address-of-record 对应）。

在发送的 493 (Undecipherable) 响应中没有证书则表示尽管它们仍支持 S/MIME 签名，应答者不能或者将不使用 S/MIME 加密消息。

注意，如果用户代理收到一个请求，请求中包含了一个非可选的 S/MIME 类型（携带了一个 Content-Disposition 头的“required”参数 handling），如果用户代理不能解析 MIME 消息体时，此用户代理必须拒绝此请求，返回响应码使用 415 不支持的媒体类型。当发送 S/MIME 时，接收这种响应的用户代理应该通知其远端用户，远程设备不支持 S/MIME，并且在随后如果合适的话，重发送请求时可能会发送没有 S/MIME 的请求；可是，此 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，在收到的响应中包含不安全的 MIME 消息，UAC 将通知其用户，会话是不安全的。但是，如果支持 S/MIME 的用户代理收到了一个带不安全消息体的请求，此用户代理不应该用安全的消息体来响应，但是如果它期望发送方

的 S/MIME (例如, 因为发送方的 From 字段值和它的密钥链上的身份对应) , UAS 应该将通知其用户, 此会话是不安全的。

当发生异常证书管理事件时, 针对用户提示来说, 多种情况会出现在以前的文本中。用户可以询问在这些状态下, 用户应该怎么处理。首先也是最重要的, 需要安全管理的情况下, 证书中意外修改, 或者在需要安全缺失都会引起警惕。但这样的警觉不一定表示安全攻击正在发生。用户可以中止任何连接尝试或者拒绝它们正在接收的连接请求; 用通信领域的俗语来说, 它们可以对此呼叫进行挂机或者回呼。用户期望找到一个可选方式来联系其他对端, 并且确定它们已经合法修改了它们的密钥。注意, 有时用户被迫修改它们的证书, 例如, 当它们被怀疑它们私钥被泄露。当它们的私钥不再是秘密状态时, 用户必须合法地产生新的密钥, 并且和拥有它们的旧密钥的用户重新建立信任关系。

最后, 如果在 dialog 进行过程中, UA 在 CMS SignedData 消息中收到了一个证书, 此证书不能对应在以前 dialog 中交换的证书, UA 必须通知其用户此修改状态, 这表示, 可能这是一个潜在的安全漏洞。

### 23.3 Securing MIME bodies

和 SIP 相关的关于 MIME 安全支持的消息体有两种类型: 这些消息体的使用将遵循 S/MIME 规范[24], 带有某些变化。

- “multipart/signed” 必须和 CMS 独立签名一起使用。  
这样的变化允许与未遵循 S/MIME 的接收方的向后兼容。
- S/MIME 消息体应该包含 Content-Disposition 头字段, “handling”参数值应该是 “required”的。
- 如果 UAC 在其密钥环上没有证书存在, 此密钥所关联的 AOR 地址是 UAC 计划发送请求的地址, 则 UACUAC 不能发送加密的 “application/pkcs7-time” MIME 消息。UAC 可以发送一个带有 CMS 独立的签名初始请求, 例如, OPTION 消息, 来索取远端证书 (签名应该包含在第 23.4 章节中描述的 “message/sip” 消息体类型中) 。

注意, 未来关于 S/MIME 标准化工作可以定义为基于密钥的非证书形式。

- 为了进一步通信，S/MIME 消息体的发送方应该采用“SMIMECapabilities”的属性（参见[参考链接 24]的 2.5.2 章节）来表示其服务能力和推荐偏好。特别注意，发送方也可以使用“preferSignedData”服务能力来支持接收方使用 CMS SignedData 消息响应请求（例如，当发送上述类似 OPTIONS 请求时）。
- S/MIME 部署至少必须支持 SHA1 作为数字签名算法和 3DES 作为加密算法。其他数字签名算法和加密算法也可以获得支持。部署方式能够通过“SMIMECapabilities”属性支持的协商机制实现对这些算法的支持。
- 在 SIP 消息中的每个 S/MIME 消息体使用签名仅获得一个证书。如果 UA 收到了带多个签名的消息，最外层的签名将视为此消息体的单一证书，不使用其他相应签名。以下是在 SIP 消息中加密 S/MIME SDP 消息体的一个示例：

```

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 SIP Header Privacy and Integrity using S/MIME: Tunneling SIP

针对 SIP 头安全的完整性来说，作为一种在某种程度提供提供端对端认证的手段，S/MIME 可以在 “message/sip” 类型的 MIME 消息体中封装整个 SIP 消息，并且可以使用同样的方法将 MIME 安全应用于这些消息体中，这些消息体作为典型的的消息体。这些封装的 SIP 请求和响应不构成分离的 dialog 或者事务，它们是 “outer” 消息的副本，用来验证安全的完整性或提供其它信息。

如果 UAS 接收了一个请求，这个请求中包含了一个隧道 “message/sip” 的 S/MIME 消息体，它应该在响应中使用相同 smime-type 来包含一个隧道 “message/sip” 消息体。

任何常用的 MIME 消息体（比如，SDP）应该附加在 “inner” 消息中，以便它们也可以从 S/MIME 安全中获得保障。

注意，如果任何非安全的 MIME 类型也应该在请求中发送的话，“message/sip” 消息体也可以 MIME 的 “multipart/mixed” 消息体的一个部分来发送。

#### 23.4.1 Integrity and Confidentiality Properties of SIP Headers

当使用了 S/MIME 完整性或安全性机制时，“inner” 消息中的值与 “outer” 消息中的值可能会有一定的差异。在本节中我们将介绍如何处理这些差异所遵守的规则，此规则仅适用于本规范中描述的所有头字段。

注意，为了实现释放时间戳的目的，所有经过隧道处理的 “message/sip” SIP 消息应该在 “inner” 和 “outer” 头中包含一个 Date 头。

##### 23.4.1.1 Integrity

无论何时执行完整性检查，决定头字段完整性是通过在签名消息体中的头字段值来匹配对比 “outer” 消息中的头字段，匹配对比规则在第 20 章中有相关描述。

被代理服务器可以合法修改的头字段是：Request-URI、Via、Record-Route、Route、Max-Forwards 和 Proxy-Authorization。如果这些头字段不是完整的端对端，部署方案不应该认为这是一个安全漏洞。本规范中定义的任何其它头字段的改变都算作是完整性的破坏；这个差异必须通知用户。

### 23.4.1.2 Confidentiality

当消息加密时，可以在已加密的消息体中包含一个头字段，这个头字段未出现在“outer”消息中。

因为一些头字段必须是以明文的形式出现，这些头字段要求出现在请求和响应中，这些字段包括：To、From、Call-ID、Cseq 和 Contact。虽然对 Call-ID、Cseq 或 Contact 来说，加密可选方式可能不是非常有用，但是对一些在“outer”信息中的 To 或者 From 头字段来说，对 To 或者 From 来说提供一个可选方式是允许的。注意，加密的消息体中的值不是用于识别事务或者 dialog 的--它们仅仅是一些信息。如果加密的消息体中的 From 头字段与“outer”消息中的 From 字段的值不同，加密的消息体中的值应该显示给用户，但是此值一定不能在未来消息的“outer”头字段中使用。

基本上来说，用户代理希望对一些包含端对端语义的头字段加密，这些头字段包括：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 字段。如果以上的这些头字段出现在加密的消息体中，应该使用这些头字段替代“outer”头字段，无论是否涉及对用户显示头字段值或涉及设置在 UA 中其内部状态。无论如何，这些头字段不应该使用在未来消息的“outer”头字段中。

如果出现的话，此 Date 头字段必须总是和出现在“inner”和“outer”中的 Date 头字段值相同。

笔者非规范补充：这里的出现或者存在是指谁存在或者出现，对照此规范相同章节。需要进行更详细的补充说明说明。

因为 MIME 消息体附加在了“inner”消息中，部署方案通常对具体 MIME 中的头字段加密，加密的字段包括：MIME-Version、Content-Type、Content-Length、Content-Language、Content-Encoding 和 Content-Disposition。在“outer”消息中，对 S/MIME

消息体来说，将包含特定的 MIME 头字段。这些头字段（和这些头字段开始使用的任何 MIME 消息体）将看作在接收的 SIP 消息中的正常的 MIME 头字段和消息体。

对于以下的头字段的加密不是非常有用，这些字段包括：Min-Expires、Timestamp、Authorization、Priority 和 WWW-Authenticate。此类别的头字段还包括那些代理服务器可改变的头字段（前面章节所描述的字段）。UA 从来都不应该包括这些字段。具体来说，如果“outer”消息中没有包括那些头字段，UA 也将不能在“inner”消息中包括那些字段。UA 收到了一个加密的消息体，此消息体中没有包含任何以上头字段的话，UA 应该其加密值。

注意，SIP 的扩展可以定义附加的头字段；这些扩展的发布者应该说明关于完整性和机密性属性中的这样的头字段。如果 SIP UA 遇到了一个未知的头字段，这些头字段违反了安全的完整性机制，UA 必须忽略此头字段。

#### 23.4.2 Tunneling Integrity and Authentication

如果发送方希望加密头字段，这个头字段被复制在“message/sip”MIME 消息体中，此消息体使用了 CMS 分离的签名签署，那么带此 S/MIME 消息体的隧道 SIP 消息就可以为此 SIP 头字段提供安全完整性。

如果隧道 SIP 消息已提供了“message/sip”消息体，这个消息体至少包含基础 dialog 标识符(To、From、Call-ID 和 CSeq)的标识，那么签名的 MIME 消息体就可以提供有限的认证努力。最起码，对接收方来说，如果接收方不知道对消息体签名所使用的证书，并且证书不能被验证的话，那么，签名可用来确认初始 dialog 的证书持有者是否和在此 dialog 中后续发送请求的证书持有者是否是同一证书持有者。如果此签名的 MIME 消息体的接收方有非常强烈的激励机制信任此证书(接收方能够验证此证书，接收者从可信任的仓库获取该证书，或者接收方经常使用此证书)，那么此签名可视为是证书主体身份的一个有力声明。

为了减少因为添加或减少整个头字段处理可能造成的困惑，发送方从在签名的消息体的请求中复制所有的头字段。任何需要完整性保护的消息体必须附加“inner”消息。

如果一个 Date 头字段出现在了带签名消息体的消息中，如果可行的话，接收方应该将此 Date 头字段值和它自己的内部时钟值进行比较。如果检测到明显的时间差异(大约一小时或更久时间)，用户代理应该对用户发出异常告警，并且提醒这是一个潜在的安全漏洞。

如果接收方检测到消息中的完整性被破坏，如果这是一个请求，返回响应使用 403(FORBIDDEN)，或结束现有的 dialog。UA 应该通知用户情况，并且请求明确说明处理的流程。

以下是隧道 “message/sip” 消息体的使用示例：

```
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
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
```

#### 23.4.3 Tunneling Encryption

隧道加密也可以使用此机制来进行描述，使用此机制在 CMS EncelopedData 消息 S/MIME 消息体中对 “message/sip” MIME 消息体加密，但是，在实际工作环境中，至少大多数头字段会使用在网络中；使用 S/MIME 加密的一般使用可以用于类似于 SDP 那样的消息体安全，而不用于消息头安全。一些关于信息的头字段，例如，Subject 和 Organization 可能可以保证端对端的安全。未来在 SIP 应用中定义的头可能也需要模糊处理。

对加密头字段另一个可能的应用方式是可选匿名方式。可以 From 头字段来构建一个请求，请求中的 From 头字段可以不包含任何个人信息(例，sip:anonymous@anonymizer.invalid)。但是，第二个 From 头字段包含发起方的真正的 address-of-record (AOR) 地址信息，此 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 dialog 的终端可见。

注意，如果匿名使用此加密机制，From 头字段将不再为消息的接收方使用，不能作为证书密钥链索引来获取与发送者相关的正确 S/MIME 密钥。消息首先需要解密，并且使用 “inner” 的 From 头字段作为索引。

为了提供端对端的安全完整性，应该由发送方对加密的 “message/sip” MIME 消息体签名。这样会创建一个 “multipart/signed” 的 MIME 消息体，此消息体包含 “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示例，用 Asterisk “\*” 星号框起来的文本是被已经加密的文本：

```

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 *
*
* 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 *
*
* v=0 *
* o=alice 53655765 2353687637 IN IP4 pc33.atlanta.com *

```

## 24 Examples

下面的实例中，为了表述相对简洁一些，我们通常忽略了消息体和其相应的 Content-Length 和 Content-Type 头字段。

### 24.1 Registration

Bob 在开始阶段注册。消息流程如图 9 所示。为简化流程，这里没有显示注册所需要的认证机制。

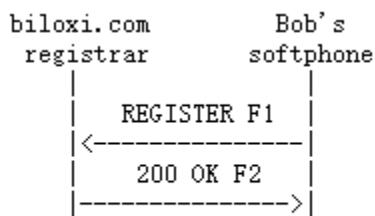


Figure 9: SIP Registration Example

```

F1 REGISTER Bob -> Registrar
REGISTER sip:registrar.biloxi.com SIP/2.0
Via: SIP/2.0/UDP
bobsipc.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@998sdasd09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0
  
```

注册在 2 小时后失效，注册服务器返回一个带 200 OK 的响应。

```
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@998sdasd09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0
```

## 24.2 Session Setup

本示例包含了第 4 章中会话建立所需要的全部细节。消息流在图 1 所示。注意这些消息流只显示了所需的最少要求的头字段--其它头字段如 Allow 和 Supported 通常都会出现在消息中。

F1 INVITE Alice -> atlanta.com 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's SDP not shown)

F2 100 Trying atlanta.com 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 atlanta.com 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
```

流程到此为止，在 Alice 和 Bob 之间媒体会话创建。

Bob 首先挂机，注意 Bob 的 SIP 电话维持自己的 Cseq 序列号范围，在此示例中，其范围取值从 231 开始。因为是由 Bob 发起的请求，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[参考资料链接 40]包含了 SIP 消息更详细的示例。

## 25 Augmented BNF for the SIP Protocol

在本规范中所指定的所有机制都以普通文章和扩展 BNF 的方式进行描述，关于扩展的 BNF 的定义参见 RFC2234[参考链接 10]。在 RFC2234 第 6 章定义了一组在本规范使用的核心规则，这里不再重述。为了完整理解此规范，部署者需要熟悉 RFC2234 中的标识符号和内容。一些基本规则使用大写字母表述，如 SP、LWS、HTAB、CRLF、DIGIT、ALPHA 等。尖括号使用在定义中用来说明规则名称的使用方式。

使用方括号是语法上的一种备用方式。它作为一种语义提示来表示特定参数是可选使用参数。

## 25.1 Basic Rules

本规范使用了以下规则描述基本的解析结构。US-ASCII 编码字符集由 ANSI X3.4-1986 定义。

```
alphanum = ALPHA / DIGIT
```

规范中有一些规则是从 RFC2396 [参考链接 5]中引入的，但为遵循 RFC 2234 [参考链接 10]，这里对它们进行了更新。这些规则包括：

```
reserved   = ":" / "/" / "?" / ":" / "@" / "&" / "=" / "+"
            / "$" / ","
unreserved = alphanum / mark
mark      = "-" / "." / ":" / "!" / "^" / "*" / ","
escaped   = "%" HEXDIG HEXDIG
```

SIP 头字段值可分为多行，如果续行的话，可以连续以空格或换码符'\t'开始。所有的线性空格，包括叠层，这些空格有同样的语义，类似于 SP。接收发在解析字段值或向下游转发消息前，可以用一个单个的 SP 替代任何线性空格。这一点与 RFC2616[参考链接 8]中描述的 HTTP/1.1 完全相同。当线性空格是可选时，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
```

为了将头名称从其它值中分离出来，根据前面的规则需要使用一个冒号，允许冒号前出现在空格，但不允许冒号后出现换行和空格，包括一个换行 linebreak。HCOLON 定义的如下：

```
HCOLON = *( SP / HTAB ) ":" SWS
```

TEXT-UTF8 规则仅用于可描述字段内容和值，这些值不用于消息解析器的解析。\*TEXT-UTF8 的单词包含 UTF-8 字符集，它们来自于 (RFC2279[7]) 的字符。TEXT-UTF8-TRIM 规则用于描述性字段内容，该内容不是引用的字符串，在引用的字符串中，其中第一和尾部的一位 LWS 无任何意义。从这一点来说，SIP 区别于 HTTP，HTTP 使用的是 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 定义中，CRLF 允许作为头字段的附加部分使用。在解析 TEXT-UTF8-TRIM 值前，希望用一个 SP 替代叠层的 LWS。

在几个协议网元中使用了十六进制数字字符。某些网元 (authentication) 强制十六进制的首字母为小写字母。

```
LHEX = DIGIT / %x61-66 :lowercase a-f
```

许多 SIP 头字段值由通过 LWS 或特殊字符分隔处理后的单词组成。除非有特殊说明，token 的字母对大小写不敏感。这些特殊字符必须为引用字符串，使用在参数值中。Word 结构使用在 Call-ID 中，允许使用多个分隔符。

```

token      = 1*(alphanum / "-" / "." / "!" / "%" / "*"
           / "-" / "+" / "^" / ";" / "~")
separators = "(" / ")" / "<" / ">" / "@" /
             "," / ":" / ";" / "\\" / DQUOTE /
             "/" / "[" / "]" / "?" / "=" /
             "{" / "}" / SP / HTAB
word       = 1*(alphanum / "-" / "." / "!" / "%" / "*"
           / "-" / "+" / "^" / ";" / "~" /
           "(" / ")" / "<" / ">" /
           ":" / "\\" / DQUOTE /
           "/" / "[" / "]" / "?" / =
           "{" / "}" )

```

当标记或分隔符使用在要素之间时，在这些字符前后允许使用空格。

```

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 "<" 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

```

一些 Commet 注释可以包含在一些 SIP 头字段中，其方法是用小括号将注释文本括起来。仅在字段值定义中包含 “comment” 的字段才允许包含注释。在其它字段中，小括号都作为字段值的一个部分。

```

comment  = LPAREN *(ctext / quoted-pair / comment) RPAREN
ctext    = %x21-27 / %x2A-5B / %x5D-7E / UTF8-NONASCII
           / LWS

```

ctext 包除了左括号、右括号和反斜杠之外的所有字符。如果文本字符串是以双引号标记的引用，那么该文本字符串解析为一个单个词语。在引用的字符串中忽略引号 ( " ) 和反斜杠 ( \ ) 。

```

quoted-string = SWS DQUOTE *(qdtext / quoted-pair ) DQUOTE
qdtext        = LWS / %x21 / %x23-5B / %x5D-7E
               / UTF8-NONASCII

```

反斜杠 (\) 可作为在引用字符串和注释结构中的一个单字符引用机制使用。不像 HTTP/1.1，在这个机制在，字符 CR 和 LF 不能被忽略，以免与行叠层和头分隔冲突。

```

quoted-pair = "\" (%x00-09 / %x0B-0C
              / %x0E-7F)

SIP-URI      = "sip:" [ userinfo ] hostport
                uri-parameters [ headers ]
SIPS-URI     = "sips:" [ 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 [ "." ]
domainlabel   = 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

```

电话用户的关于 BNF，参见 RFC2806[参考链接 9]。注意，但是在电话用户的语法 BNF 中允许任何字符，而 SIP URI 的用户部分不允许的话，这些字符必须要被忽略。

```

uri-parameters      = *( ";" 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

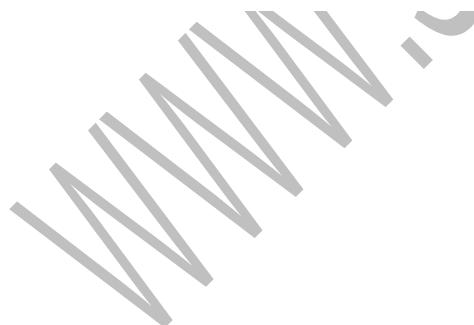
```

```
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)

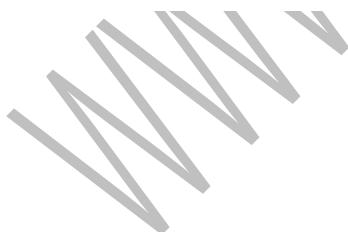
Informational   = "100" : Trying
               / "180" : Ringing
               / "181" : Call Is Being Forwarded
               / "182" : Queued
               / "183" : Session Progress
```



```
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
```



```
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 *(COMMA encoding) ]
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
callid           = word [ "@" word ]

Call-Info         = "Call-Info" HCOLON info *(COMMA info)
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

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 Security Considerations: Threat Model and Security Usage Recommendations

SIP 协议不是一个很容易实现安全管理的协议。媒体中介的 SIP 使用、SIP 的多层次的信任关系，在完全不信任的网元之间中所期望的用法和 SIP 用户之间的操作，这些因素都让安全问题变得非常琐碎，安全也显得更加重要。在缺乏广泛协同，并且用户环境变化很大和使用的不同中，今天，可部署的安全解决方案是需要的。为了满足这些不同的产业化的需要变化，用户需要一些完全不同机制来适用于不同的用户场景。

注意，SIP 信令本身自己的安全性和与 SIP 配合的概念所使用的协议安全性无关，例如，RTP 协议，或与具体的 SIP 消息体安全部署传输（虽然 MIME 的安全在 SIP 安全方面扮演角色）。任何与一个会话关联的媒体可以独立地进行端对端加密，这些对媒体加密独立于其任何关联的 SIP 信令。媒体加密的讨论超出了本规范讨论的范围。

关于安全讨论，主要有两个方面需要考虑。按照这两个方面的讨论，首先审查一组传统安全威胁模式，这些传统的安全威胁模式比较宽泛地定义了 SIP 安全的需求。一组安全服务详细阐述了安全威胁的细节，然后介绍了提供安全服务所使用的几个安全机制概念解释。接下来，列举了对 SIP 部署的要求，并且还列举了几个比较典型的部署使用方式--这些部署方式所使用的安全机制可以用来提高 SIP 的安全性。在此章节议定出针对隐私安全的一些提示。

### 26.1 Attacks and Threat Models

本章详细描述了在 SIP 部署环境中出现的一些常见安全威胁。选择这些特别的安全威胁解释 SIP 所需的每个安全服务。

下面例子并不能详尽地列出所有 SIP 受到的安全威胁；确切的说，这些“传统”的安全威胁论证了针对特别安全服务的需求，这些安全服务能够防止各种潜在威胁发生。

这些安全攻击假设了一个这样的环境：在这个环境里，攻击者可能读取网络上任何数据信息包——我们可以预知在公网上频繁使用 SIP 环境。网络上的攻击者可以修改数据包中的内容（这些数据包可能在一些暴露的中间媒体）。攻击者可能希望盗取服务、对通信服务进行监听或破坏会话。

#### 26.1.1 Registration Hijacking

注册劫持是 SIP 安全威胁中的其中一种攻击方式。SIP 注册机制允许用户代理向注册服务器自识别用户身份，用户代理（由 AOR 地址选定的）通过终端实现自己的定位，向注册服务器进行注册。注册服务器评估 REGISTER 消息的 From 头字段中所声明的身份，决定此请求是否能修改 To 头字段中 contact 地址，这个 contact 地址关联 To 头字段中的 address-of-record (AOR) 地址。如果这两个字段经常是相同的，那就会有许多有效部署，在这些部署环境中，软硬件第三方可以代表用户注册这些 contact。

UA 的所有者能够任意修改 SIP 请求的 From 头字段，因此，这样就为恶意注册打开了后门。成功地冒充已被授权第三方授权的攻击者可以修改和 AOR 地址关联的 contact 地址，例如，攻击者可以注销针对一个 URI 的所有现存 contacts，然后使用盗用的 contact 地址注册自己所有的设备，从而将所有向被感染的用户发出的请求直接发送到攻击者自己控制的设备上，最终实现注册劫持。

这种威胁是威胁分类中的其中一种威胁方式，此安全威胁分类依赖于请求发起方缺乏密码保障。任何充当有价值服务（例如，网关设备，实现 SIP 请求与传统电话互通业务服务）的 SIP UAS 可以通过接收认证请求方式来控制对其资源的访问。即使，终端用户 UA，例如 SIP 电话都有这样的意愿，SIP 电话希望能确定请求发起方的身份。

这种安全威胁说明了对安全服务的需求，这种需求使得 SIP 实体能够对请求的发起方进行认证。

### 26.1.2 Impersonating a Server

服务器冒充或服务器伪装是攻击者在 SIP 网络安全威胁使用的另外一种攻击手段。请求要抵达的网络域通常在 Request-URI 中指定。为传输此请求，UA 通常与在此域的服务器直接联系。但是，这里存在着一个可能性，攻击者可以冒充为一个远程服务器，和 UA 所发送的请求就会被第三方服务器拦截。

例如，讨论一种这样的情况，重定向服务器在一个网络域上，这个域是 chicago.com，使用这个域冒充另外一个重定向服务器域，此重定向服务器在 biloxi.com 上。用户代理向 biloxi.com 域发送请求，但是在 chicago.com 域的重定向服务器应答了这个请求，在回复中使用了一个伪造的响应，此伪造的响应包含来自于 biloxi.com 域的响应所需要的 SIP 头字段。在重定向响应中的伪装的 contact 地址会指引初始方 UA 指向一个不恰当的或不安全的资源，也可能简单地阻止后续请求继续向 biloxi.com 域发送请求。

这种威胁分类具有很多子类成员，许多子类的安全威胁是非常危险的。与注册劫持相反，我们考虑这样一种情况，注册消息是应该发送到 biloxi.com 的，但是此注册请求被 chicago.com 这个域截获，chicago.com 使用一个伪造的 301 (Moved Permanently) 做的响应回复。这样的响应看起来像是来自指定的 biloxi.com 域，作为一个合理的注册服务，经过伪造以后，这个响应实际上是真正来自于 chicago.com 这个域。后续来自初始 UA 的所有 REGISTER 请求都会发到经过伪造的 chicago.com 域上。

为防止这种安全威胁发生，需要 UA 对它们发送请求的服务器认证。

### 26.1.3 Tampering with Message Bodies

篡改消息体也是 SIP 中的一种安全威胁。作为一种常规操作，SIP UA 通过可信任的代理服务器路由请求。不管这种信任是如何创建的（代理认证已经在其它章节讨论），一个 UA 可以信任代理服务器路由请求，但是并不检查或可能修改请求中的消息体。

我们现在讨论这样一种关于 UA 情况，UA 使用 SIP 消息体为媒体会话进行会话加密密钥的通信。虽然 UA 信任其代理服务器的域，使用此域正在正常传输信令，但 UA 不希望这个域的管理员具有对任何后续媒体会话的解密能力。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。更糟糕的是，如果代理服务器是活跃恶意的代理服务器，它可以修改会话密钥，或作为一个中间人，或可能修改初始方 UA 已请求安全特性。

此威胁分类不仅使用在会话密钥中，对会话密钥构成威胁，还对大多数在 SIP 端对端传输中可想到的内容格式构成威胁。这些内容可能包括为用户呈现的 MIME 消息体、SDP 或封装的电话信号和其它的内容。攻击者可能试图修改 SDP 消息体，例如，将 RTP 媒体流指向一个窃听设备，这样就可以让窃听设备对后续的语音通信进行窃听操作。

我们还要注意，在 SIP 中的一些头字段是有含义的端对端头字段，例如 Subject。UA 可能会保护这些头字段和消息体（例如，恶意中间媒体改变 Subject 这个头字段，通过这样的修改，可以使得一个重要的请求看上去像一条垃圾信息）。但是，因为在路由请求时，代理服务器已对许多头字段进行了合法检查和修改，所以，不是所有头字段都应该是安全的端对端头。

因此，针对以上原因，UA 可能希望以端对端方式进行保护，保护 SIP 消息体，并且有时可能在某些有限环境中对一些头字段进行保护。消息体所要求的安全服务包括机密性、完整性和认证。这些端对端的服务独立于用于中间媒体（如代理服务器）交互所使用的安全手段，这些中间媒体的安全交互和端对端安全服务无关。

#### 26.1.4 Tearing Down Sessions

在 SIP 中拆除会话是威胁分类中的其中一种安全威胁。dialog 一旦通过初始的消息建立以后，通过发送后续请求就可以修改 dialog 状态，和/或会话状态。对于此安全分类来说，非常重要的是，需要确保会话的原则，那就是这些请求不能被攻击者伪造。

我们考虑这样一种情况，第三方攻击者捕获在 dialog 中的一些初始消息，这些初始消息是 dialog 中双方共享的消息，攻击者利用这些消息学习会话参数（To 标签、From 标签等），然后在会话中插入攻击者所要求的 BYE 请求。攻击者可以选择伪造请求，这样就使得这些请求看起来是来自于 dialog 参与方的其中一方。一旦其目标用户接收到 BYE 请求，曾会话就会被提前拆除。

同样的，中期会话（mid-session）威胁包括了伪造 re-INVITE 传输，通过这个伪造的 re-INVITE 来修改会话（可能会降低会话安全或对媒体流进行重定向来作为窃听攻击的一个部分）。

对于这种安全威胁最有效的反制措施是对 BYE 请求发送方认证。在这种实例中，接收方仅需要获知 BYE 来自于同样的发送方，此发送方具有对应的已创建 dialog（与之相反的是探测发送方的绝对身份）。另外，因为某些安全设置的原因，如果攻击者不能获知会话参数的话，伪造 BYE 请求也是不可能的。但是，一些中间媒体（例如代理服务器）需要在会话创建时插入那些会话参数。

#### 26.1.5 Denial of Service and Amplification

拒绝服务攻击或者数据放大是威胁分类中的一种安全威胁。拒绝服务攻击主要针对网络环境进行攻击，表现为让网络中特定网元设备或者软件不可用，通常使用的攻击方式是对某些网元网络接口转发大量的网络数据。一个分布式的拒绝服务攻击允许一个网络用户通过多个网络宿主机对一个目标地址发送大量的网络数据来进行洪水攻击服务。

在许多 SIP 网络架构中，SIP 代理服务器面对公网方便接受来自全世界部署的 IP 终端。SIP 为分布式拒绝攻击制造了很多潜在机会，SIP 系统部署用户或者 SIP 系统运营用户必须认识和应对分布式拒绝攻击的威胁。

攻击者可以创建一个伪造的请求，在伪造的请求中包含了已篡改的源 IP 地址和一个对应的 Via 头字段，Via 头字段定位目标宿主机。攻击者使用这些篡改的消息作为请求初始方

对 SIP 网络网元发送大量的请求。根据以上根据方式，因此攻击者可利用一个运气比较差的实体 SIP UA 或者代理针对目标地址生成大量的拒绝服务攻击请求流量。

同样的思路，攻击者也可利用定位目标地址的请求进行攻击。攻击者使用此请求中已篡改的 Route 头字段对复制分叉处理的代理发送消息，然后复制分叉处理的代理再对目标地址发送大量类似消息。

攻击者也可以使用 Record-Route 实现同样的效果。当攻击者已经确定通过请求初始的 SIP dialog 将会导致大量在向后方向初始产生的事务。

如果注册服务不能对 REGISTER 请求正确认证和授权的话，拒绝服务攻击数量会出现。攻击者可在管理员域对部分或所有用户撤销注册，从而阻止用户加入到新的会话中。攻击者也可以注册大量的 contacts，这些 contacts 选定同一主机支持给定的 AOR 地址（address-of-record），以便攻击者在拒绝服务攻击中将这个注册服务，和任何关联的代理服务器作为数据放大器来使用。攻击者也可能通过使用大量注册绑定消息来尝试耗尽注册服务器的有效内存和存储资源。

使用多播传输 SIP 请求将大幅增加拒绝服务的可能性。

以上这些问题阐述了基本的需求，这些需求定义一个最小化拒绝服务风险的技术架构，并且在针对此类威胁的安全机制的推荐中也需要考虑到此需求。

## 26.2 Security Mechanisms

通过以上关于安全威胁的描述，我们获知 SIP 协议所要求的基本的安全服务是：维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私密机制，和防止拒绝服务攻击。某些在 SIP 消息中的消息体要求独立的机密性、完整性和认证安全服务。

SIP 协议尽可能重用了从 HTTP 和 SMTP 衍生出来的，现存安全模式，而不是专门为 SIP 协议定义新的安全机制。

消息全加密提供了维护信令的机密性的最好方式，同时消息全加密方式也可以保证消息不会被任何恶意中间媒介修改。但是，在整个 SIP 处理流程中，不能对 SIP 的请求和其响应进行简单的端对端加密，因为在绝大部分的网络架构中，对代理来说，某些消息字段，例如：Request-URI、Route 和 Via 必须是可见的，这样才能保证经过多个代理以后，SIP 请求能够被路由到正确的目标目的地。注意，代理服务器也需要修改某些消息的特性（增加一个 Via 头字段）来实现 SIP 功能。因此，从某种程度上来说，代理服务器对 SIP UA 来说，它们必须是可信任状态。为了此目的的实现，规范建议 SIP 使用较低层的安全机制，实现方式是基于逐跳对流程中的整个 SIP 请求或响进行加密，并且允许终端验证代理服务器的身份，确保此服务器是终端对其发送请求的代理服务器。

SIP 实体还需要以一种安全的方式对另外一个实体进行确认。当一个 SIP 终端向对端 UA 或代理服务器声明了自己的用户身份时，此身份应该能够以某种方式进行验证。在 SIP 中提供一种密码认证机制来应对这种要求。

一个独立的安全机制为 SIP 消息体提供了一种除端对端的相互认证以外的备选方式，同时，在用户代理必须信任中间媒体的情况下，这种独立的安全机制也提供了一个用户代理对中间媒体的信任限度。

#### 26.2.1 Transport and Network Layer Security

传输层或网络层安全对信令数据流量加密，确保消息的机密性和完整性。

在很多场合中，证书用在较低层安全创建环境中，这些证书也可以在很多网络架构中提供一种认证方式。

用于在传输层和网络层提供安全的两种受欢迎的可选方式分别是 TLS[参考链接 25]和 IPsec[参考链接 26]。

IPSec 是一个网络层协议工具集，它可整体用来作为一个传统 IP（Internet 协议）的安全替代。IPSec 通常使用在这样的架构中，在这样的技术架构中，主机集或者管理域集它们有一个现存的相互信任的关系。通常情况下，IPSec 部署在主机的操作系统级上，或部署在安全网关上，此安全网关针对它从特定接口（例如，在 VPN 架构中）接收的所有流量提供消息的机密性和完整性。IPSec 也可以部署在基于逐跳的网络环境中。

在许多网络架构中，IPSec 不要求和 SIP 应用集成；IPSec 也许是最合适部署在这样的环境中，对 SIP 主机增加安全机制异常困难，也许比较合适的方式就是部署 IPSec。UA 有共享密钥，此密钥和第一跳代理服务器存在共享密钥关系，这样的 UA 也非常适合使用 IPSec。针对 SIP 的任何 IPSec 布署要求一个 IPSec 概述，此概述描述实现 SIP 安全所要求的协议工具。本规范没有提供这样的概述文档。

TLS 在面向连接的协议层之上提供了传输层安全（本规范中是指 TCP 协议）；通过在 Via 头字段值或 SIP-URI 中设定“tls”（表示通过 TCP 的 TLS），它可作为一个理想的传输协议。最适合部署 TLS 的架构是，主机间需要逐跳安全保护，主机之间无预设的信任关联。例如，Alice 信任她自己本地代理服务器，经过交换证书后，Alice 的本地代理服务器决定信任 Bob 的本地代理服务器，当然，Bob 自己也信任自己的本地代理服务器。因此，Bob 和 Alice 就可以安全地进行通信。

TLS 必须与 SIP 的应用紧密配合使用。注意，在 SIP 中，传输机制是基于逐跳实现的，因此，通过 TLS 对代理服务器发送请求的 UA 并不能保证端对端使用了 TLS。

当 SIP 应用中使用 TLS 时，部署者必须最少支持 TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA 密码套件 [参考链接 6]。为了实现向后兼容的目的，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。部署者也可以支持任何其他密码套件。

#### 26.2.2 SIPS URI Scheme

虽然此结构字符串是“sips”而不是“sip”，SIPS URI 模式仍然遵守 SIP URI 的语法结构（见第 19 章描述）。但是，SIPS 的语义与 SIP URI 非常不同。SIPS 允许资源指定它们可以安全到达。

SIPS URI 可为一个具体用户用作 address-of-record (AOR) 使用--用户非常清楚此 URI (在用户的名片、在请求的 From 头字段、在 REGISTER 请求的 To 头字段)。当使用在请求的 Request-URI 时，SIPS 结构模式表示，从转发请求的每个跳跃点，一直到请求到达的 SIP 实体，此 SIP 实体负责 Request-URI 域部分，这些都必须用 TLS 进行安全保护。一旦请求到达一个有安全疑问域时，此请求就会根据本地安全和路由策略进行处理，相当有可能在到达 UAS 的最后一跳中使用 TLS。当请求的发起方使用 SIPS URI 发送请求时（可能会有一种情况，如果使用了一个 SIPS URL 作为目的地的 address-of-record (AOR) 地址），SIPS 要求对目标域上的整个请求路径都要进行安全保护。

今天，除了 Request-URI 以外，在 SIP 中，其他许多使用 SIP URL 的方式也可以应用 SIPS 结构模式，包括 address-of-record (AOR)，Contact 地址 (Contact 头内容，还包括它们的 REGISTER method) 和 Route 头。在每个实例中，SIPS URI 结构模式允许现有字段指定安全资源。可以在任何 SIPS URL 的内容中取消 SIPS URL 引用关联，取消引用关联的方式有其自身的安全属性，关于其自身安全属性的描述参见参考链接[4]。

使用 SIPS 尤其要求应该部署 TLS 的相互认证，就像应该需要密码套件 TLS-RSA-WITH-AES-128-CBC-SHA 一样。在认证过程中收到的证书须用客户端所持有的根证书进行有效验证；证书认证失败，请求也会失败。

注意，在 SIPS URI 的结构模式中，传输独立于 TLS 协议，因此，`sips:alice@atlanta.com;transport=tcp` 和 `sips:alice@atlanta.com;transport=sctp` 都是有效的，（尽管，注意 UDP 对 SIPS 来说不是一个有效的传输协议）。`transport=tls` 使用因此已被废弃，部分原因是它特定于请求的单个一跳。这是从 RFC2543 以后的一个修改。

用户分发了 SIPS URI 作为 address-of-record (AOR) , 这些用户可以选择运行一些设备, 让这些设备拒绝通过非安全模式下传输的请求。

#### **26.2.3 HTTP Authentication**

SIP 提供了基于 HTTP 认证的挑战能力, 这种挑战能力依赖于 401 和 407 响应代码和头字段, 头字段传输挑战和安全信息。此挑战能力没有进行重大修改, 在 SIP 中重用 HTTP 摘要验证结构可以允许重放保护和单向认证。

在第 22 章详细介绍了关于 SIP 摘要验证的用法。

#### **26.2.4 S/MIME**

根据我们上面的介绍的内容来看, 为了保密性的目的而对整个 SIP 消息进行端对端加密是不合适的, 因为网络中间媒体 (例如, 代理服务器) 需要查看某些头字段来确保正确地路由 SIP 消息, 并且, 如果把这些网络中间媒体排除在和安全关联之外, SIP 消息是根本不能路由的。

但是, S/MIME 允许 SIPUA 在 SIP 中对 MIME 消息体进行加密, 对这些消息体进行端对端加密并不影响消息头工作。S/MME 能够为消息体提供端对端的完整性和保密性, 也能够提供相互认证。还有一种可能是, 通过 SIP 消息隧道, 使用 S/MME 为 SIP 头字段提供完整性和机密性的格式。

在第 23 章详细介绍了关于 SIP 中 S/MIME 的使用方法。

### **26.3 Implementing Security Mechanisms**

#### **26.3.1 Requirements for Implementers of SIP**

代理服务器、重定向服务器和注册服务器都必须支持 TLS 部署, 并且都必须支持相互认证和单向认证。规范强烈推荐 UA 有能力初始 TLS ; UA 也可以作为 TLS 服务器使用。代理服

务器、重定向服务器和注册服务器应该拥有站点证书--其主题和服务器规范的主机名对应。UA 可以拥有自己的证书，使用这些证书使用 TLS 进行相互认证，在规范中没有其使用说明。所有支持 TLS 的 SIP 网元必须有一种机制来验证在 TLS 协商中收到的证书。这就要求证书授权机构签发一个或多个根证书（最好是非常知名的站点证书发布方，这些权威发布方可与网站浏览器根证书签发的发布方相比）。

所有支持 TLS 的 SIP 网元必须支持 SIPS URI 模式结构。

代理服务器、重定向服务器、注册服务器和 UA 也可以部署 IPSec 或其它较低层安全协议。

当 UA 尝试联系代理服务器、重定向服务器或者注册服务器时，UAC 应该通过它将发送 SIP 消息的连接来初始启动一个 TLS 连接。在一些技术架构中，UAS 也可以通过这样的 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 必须围绕着第 22 章要求的所有方面部署摘要授权。代理服务器、重定向服务器和注册服务器应该至少要配置一个摘要 realm 域，并且由给定服务器支持的至少一个 realm 字符串应该和此服务器主机名或域名对应。

UA 可支持 MIME 消息体的签名和加密和带 S/MIME 的凭证转移，凭证转移在本规范第 23 章进行说明。如果一个 UA 为了验证 TLS 或 IPSec 的证书，而持有一个或多个认证机构的根证书，那么，在适当时，它应该能够重用这些证书来验证 S/MIME 证书。UA 可以持有一个根证书专门用于验证 S/MIME 证书。

注意，可预期的是，需要 S/MIME 部署时，将来的安全扩展可以提升和 S/MIME 相关联的标准强度，问题空间就变得容易理解。

### 26.3.2 Security Solutions

在某种程度上，这些安全机制运行需要一致，安全机制运行都可参照已有的 web 和 email 安全模式。在较高层级中，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) 进行自我验证; 服务器使用由 TLS 传输的站点证书向 UA 单跃点或另外一个服务器单跃点(反之亦然)进行自验证。

在点对点的层级中, 通常来说, UA 信任网络, 让此网络对另一方进行验证。但是, 如果该网络没有验证, 或者网络本身是不可信网络, 那么, S/MIME 可以用来提供直接的验证。

下面是一个说明的实例, 在此实例中, 不同的 UA 和服务器使用这些安全机制防止安全威胁, 这些威胁讨论在第 26.1 章节中有具体描述。部署方和网络管理员可按照本章的后续部分给出的标准指南来实现, 这些实例仅为部署的示例。

#### 26.3.2.1 Registration

当 UA 处于在线状态, 通过本地管理域注册时, UA 一个和它的注册服务器创建一个 TLS 连接(第 10 章描述了 UA 如何到达其注册服务器)。注册服务器应该为 UA 提供一个证书, 而且, 此证书识别的站点必须对应 UA 要进行注册的域; 例如, 如果 UA 想要注册的 address-of-record (AOR) 的记录地址为 alice@atlanta.com, 那么站点证书必须识别域 atlanta.com 上的主机(如 sip.atlanta.com)。当 UA 接收到 TLS 证书消息时, UA 应该验证证书, 并且检查证书识别的站点。如果证书无效、被撤销或者不能识别正确的对端实体, UA 就不能发送 REGISTER 消息, 否则将继续注册。

当注册服务器已提供了有效证书, UA 知道该注册服务器不是一个对 UA 进行重定向、 盗窃密码或尝试类似攻击的攻击者。

然后 UA 创建一个 REGISTER 请求, 在请求中说明 Request-URI 对应从注册服务器接收的站点证书中的 Request-URI。当 UA 通过现存 TLS 连接发送 REGISTER 请求时, 注册服务器应该挑战请求, 返回 401(Proxy Authentication Required)响应码。响应码中的 Proxy-Authenticate 头字段的 realm 参数应该与前面站点证书给出的域对应。当 UAC 接收到此挑战, 它应该提示用户提供凭证, 或者从挑战信息中的的 realm 参数所对应的密钥环中提取正确的凭证。凭证的用户名应该和 REGISTER 请求中的 To 头字段 URI 的 USERINFO 一致。

一旦摘要凭证插入到正确的 Proxy-Authenticate 头字段中，应该对注册服务器重新提交 REGISTER 请求。

因为注册服务器要求用户代理对其身份进行认证，这将使得攻击者伪造 REGISTER 请求中的用户 address-of-record (AOR) 地址变得非常困难。也要注意的是，由于 REGISTER 请求是通过安全的 TLS 连接发送，所以攻击者不可能截获 REGISTER 请求，通过记录的注册请求凭证发动任何可能的重放攻击。

一旦注册服务器接受了注册请求，UA 应该将 TLS 连接开放给注册服务器来作为代理服务器使用，在此管理域的用户可以向此代理服务器发送请求。为了对刚完成注册的 UA 传输入局请求，将重用现存的 TLS 连接。

因为 UA 已经对在 TLS 连接的另一边的服务器进行了认证，通过此连接的所有请求已被获知经过了此代理服务器-因此，攻击者不能通过此代理服务器发送欺骗请求。

#### 26.3.2.2 Interdomain Requests

现在，我们假设 Alice 的 UA 想对一个位于远端管理域 bob@biloxi.com 的用户发起会话。我们假设本地管理域 (atlanta.com) 支持了一个本地的 outbound proxy 出局代理。

一个负责管理域的 inbound 请求的代理服务器也可以作为一个本地的 outbound proxy 代理；为了方便起见，对 atlanta.com 来说，我们假设这样一个场景（否则，在这一点上，用户代理将为独立的服务器发起一个新的 TLS 连接）。假设，用户完成了上述的注册流程后，当它对其它用户发送 INVITE 请求时，它应该重用本地代理服务器的 TLS。UA 应该重用 INVITE 中缓存的安全凭证，避免对用户再次提出不必要的提示。

当本地 outbound proxy 出局代理验证了 UA 在 INVITE 中出示的安全凭证，此本地出局代理服务器应该检查 Request-URI 值来决定如何路由消息（见参考链接[4]）。如果 Request-URI 中的 “domainname” 和本地域(Atlanta.com)一致，而不是和 biloxi.com 一致，那么代理服务器就会查询其定位服务器来决定如何以最优方式抵达请求的用户。

假设 alice@atlanta.com 已试图联系了 alex@atlanta.com，那么本地代理服务器将把此请求代理到一个 TLS 连接上，这个连接是当 Alex 注册后，Alex 和注册服务器之间已创建的 TLS 连接。因为 Alex 已通过他的认证通道接收到了此请求，因此，Alex 确信 Alice 的请求已经通过本地管理域下的代理服务器的授权。

但是，在本示例中，Request-UR 指定了一个远程域。在 atlanta.com 域的本地 outbound proxy 出局代理服务器应该和远端代理服务器（在 biloxi.xom 域上代理服务器）建立一个 TLS 连接。因为，在此 TLS 连接中的双方都是服务器方，它们都拥有站点证书，应该进行证书互相认证。连接的对端都应该验证和检查对端的证书，注意，出现在证书中的域名，将和 SIP 消息中的头字段进行比较。例如，atlanta.com 代理服务器应该在从 biloxi.xom 域对应的远端收到证书的这个阶段开始验证证书。一旦完成了这样的操作验证，并且成功完成 TLS 协商，在两个代理服务器之间就会生成一个安全通道。那么，atlanta.com 代理服务器就可以向 biloxi.com 转发 INVITE 请求。

反过来，在 biloxi.com 域的代理服务器也应该检查在 atlanta.com 域的代理服务器的证书，并且将证书中声明的域和 INVITE 请求中的 From 头字段中的“domainname”部分进行对比。Biloxi 域的代理服务器可以有非常严格的安全策略，在安全策略中，它可以拒绝请求，这些发出请求的管理域不能匹配它已代理的管理域。

建立这样的安全策略可防止在 SIP 中出现和 SMTP 中的‘开放转发’类似的场景发生，利用开放转发类似的业务会生成垃圾邮件一样的垃圾信息。

但是，此策略只保证来自于所属它自己域的请求的安全；此策略不允许 biloxi.com 探测 atlanta.com 是如何认证 Alice 的。仅如果 biloxi.com 已有其它的方式获知 atlanta.com 的认证策略时，biloxi.com 它才可能探测 Alice 如何证明自己的身份。biloxi.com 可以建立更严格的安全策略，禁止来自未知管理域的请求和 biloxi.com 分享一个共有的认证策略。

一旦 INVITE 请求被 biloxi 的代理确认，如果有任何和用户关联，此用户是由请求发出的目标用户（这里是 bob@biloxi.com），代理服务器应该识别现存的 TLS 通道。此请求应该

通过这个通道代理到 Bob。因为，此请求是通过一个就像代理服务器一样已认证的 TLS 连接接收的请求。虽然没有必要是否信任 Alice 的身份，但是，Bob 知道 From 头字段没有被篡改，而且 atlanta.com 已验证了 Alice 的身份。

在代理转发请求之前，所有代理服务器都应该在请求中添加一个 Record-Route 头字段，以便在此 dialog 中的后续请求都能通过代理服务器。因此，代理服务器就可以继续在此 dialog 的生命周期中仍然提供安全服务。如果代理服务器没有把自己添加到 Record-Route 头字段，那么，后续消息将直接在 Alice 和 Bob 之间以端对端方式进行传输而不经过任何安全服务（除非双方同意在一些独立的端对端安全上进行，例如，S/MIME）。从 SIP 的角度来看，梯形模型（不规则四边形模式-两个代理服务器，两个终端的示例模型）可以提供一个非常好的架构，这个模型规范了在两个站点代理之间的协商约定，通过此约定为 Alice 和 Bob 之间提供一个合理的安全通道。

例如，作为一个攻击者，试图在此架构捕获某些漏洞或者进行攻击时，攻击者不可能伪造一个 BYE 请求，也不可能将攻击者插入到 Bob 和 Alice 之间的信令流中。因为攻击者无法探知到此会话的参数，也因为完整性机制通过可传递的方式保护了 Alice 和 Bob 之间的流量。

#### 26.3.2.3 Peer-to-Peer Requests

换个角度来说，我们考虑这样一个场景，UA 声明了在 carol@chicago.com 的身份，此身份不支持本地出局代理。当 Carol 希望对 bob@biloxi.com 地址发送 INVITE 请求时，她的 UA 应该直接对 biloxi 代理发起一个 TLS 连接（使用[参考链接 4]中描述的机制决定如何以最佳方式抵达指定的 Request-URI）。当 Carol 的 UA 接收到从 biloxi 代理发送的证书，通常情况下，它的 UA 应该先验证其证书，然后 UA 通过其验证以后，使用通过的 TLS 连接传输 INVITE 请求。但是，Carol 无法向 biloxi 代理确认自己的身份，她可以在 INVITE 中的 “message/sip” 消息体上增加一个 CMS 独立的签名来确认自己的身份。因为她和 biloxi.com 没有正式的关联关系，所以，在此例中，Carol 也不可能有 biloxi.com 域的安全凭证。Biloxi 代理也可以创建一个严格的安全策略，通过这种严格的安全策略来排除代理

对挑战请求的干扰，在这些请求中的 From 头字段 “domainname” 域没有 biloxi.com 的值的话--代理将认为这些用户是未认证用户。

Biloxi 代理针对 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、正确的 contact 地址，命名为<sip:bob@192.0.2.4>。Carol 通过她已和 biloxi 代理服务器创建的 TLS 连接来接收重定向响应，因此，Carol 信任此 contact 地址的真实性。

然后，Carol 应该使用指定的地址创建一个 TCP 连接，发送一个带有 Request-URI 的新 INVITE，Request-URI 中要包含收到的 contact 地址（当请求已经准备好以后，重新计算消息体中的签名）。Bob 在一个非安全的接口收到此 INVITE，但是 Bob 的 UA 会检查并识别请求中的 From 头字段，然后将本地缓存的证书与 INVITE 消息体中出现的签名进行对比。Bob 以同样的方式回复，向 Carol 进行自验证，然后开始安全 dialog。

有时，在管理域中的防火墙或 NAT 阻止了 TCP 直接和 UA 连接的创建。在这些环境中，以一个不信任暗示的方式作为本地策略的规则，代理服务器也可以通过潜在转发方式对 UA 转发请求（例如，废弃现存的 TLS 连接，并且通过明文 TCP 转发请求）。

#### 26.3.2.4 DoS Protection

为了将针对网络架构的拒绝服务攻击风险减少到最小，使用这些安全解决方案时，部署者应该注意以下几个方面的指定原则。

当主机是在一个正在运行 SIP 代理服务器的主机，此主机是一个可从公网路由，此主机应该部署在一个带防御操作策略管理域上（过滤源路由的流量，最好优先过滤 ping 流量）。TLS 和 IPsec 也都可以在管理域边缘使用堡垒机，在边缘处参与安全关联来聚合安全隧道和套接字。这些部署在边缘处的堡垒机可以承受拒绝服务攻击，保护管理域内的 SIP 主机不被过多的消息阻塞。

无论部署何种安全解决方案，直接发送到代理服务器的消息洪水会锁定代理服务器资源，而且阻止预期的消息流到达目的地地址。在代理服务器端有一个和处理 SIP 事务相关的运

算支出，作为有状态代理服务器时对 SIP 事务处理的费用支出大于作为无状态代理服务器时对 SIP 事务处理的费用。因此，有状态代理比无状态代理服务器更容易受到洪水攻击的影响。

UA 和代理服务器仅使用 401 (Unauthorized) 或 407 (Proxy Authentication Required) 对有疑问的请求进行挑战，丢弃正常的响应传输算法，并且，对未认证的请求执行无状态处理。

重传 401 (Unauthorized) 或 407 (Proxy Authentication Required) 状态响应放大了攻击者问题，攻击者使用篡改的头字段值（例如，Via）将流量转发到一个第三方。

总之，通过机制，例如 TLS，代理服务器之间实现了证书相互认证，这样极大地减少了流氓中间媒体入侵的可能性，这些中间媒体引入篡改的请求或者响应，这些被篡改的请求和响应可以造成拒绝服务。这样也让攻击者试图将合法的 SIP 节点作为放大的代理的操作变得更加困难。

## 26.4 Limitations

当用户以合理的方式使用这些安全机制后，尽管这些安全机制能够防止很多安全威胁发生，但是部署者和网络运营人员仍然必须理解这些安全机制的局限性。

### 26.4.1 HTTP Digest

在 SIP 中使用 HTTP 摘要其中一个主要的局限性，其局限性是摘要的完整性机制不能很好配合 SIP 工作。具体来说，HTTP 摘要提供了对 Request-URI 和消息 methods 的保护，但是 HTTP 摘要不能对任何 UA 所期望提供安全机制的头字段提供保护。

在 RFC2617 中所描述的现存的重放保护机制，也对 SIP 有一定的限制。例如，下一代随机数机制不支持基于管道的请求。应该使用随机数计数机制用于重放保护。

在 SIP 中使用 HTTP 摘要的另一个限制就是 realm 域的范围。摘要是否有效取决于应用环境。当用户们期望向它们的预存关联资源进行自验证时，摘要是有效的，如服务提供商的客户，这里，用户是此服务提供商的一个客户（这只是一个非常常见的使用场景，此摘要提供一种非常有用的功能）。对比来看，TLS 的使用范围是在域间或者多网域环境中使用，因为证书部署经常是全局可验证的，所以 UA 能够在无预存关联的情况下认证服务器。

#### 26.4.2 S/MIME

使用 S/MIME 机制有一个最显著的缺陷，对终端用户来说，证书机构缺乏广布的公钥基础设施。如果使用自签证书（或者在 dialog 中的其中一方参与方不能验证证书），那么基于 SIP 的密钥交换机制，在第 23.2 节所描述的这种机制就容易受到自己人式的攻击，攻击者极可能检查和修改 S/MIME 消息体。攻击者需要非法获取通信双方在 dialog 中的第一次密钥交换，从请求和响应移除现存的 CMS 独立签名，并且插入一个攻击者自己的不同 CMS 独立的签名，并且在此签名中包含了攻击者所提供的证书（而且此证书看似是一个带正确 AOR 记录地址的证书）。事实上，当每一方都持有了攻击者的公共密钥后，在 dialog 中的每一方将认为它们已经和对方交换了密钥。

需要特别注意的是，攻击者仅能够利用 dialog 中双方的第一次密钥交换的漏洞-在后续的场合中，密钥的改变会向 UA 发出提示。攻击者长时间停留在双方后续的 dialog 的路径上是非常困难的（可能需要停留几天、几周或几年）。

SSH 在第一次密钥交换中也容易受到同样的自己人攻击；而且，大家都认可，虽然 SSH 不完美，但是，SSH 也确实提升了连接的安全性。就像为 SSH 所提供协助一样，使用指纹也能够向 SIP 提供协助。例如，如果双方使用 SIP 创建了语音通信的会话，那么每一方能够读取到它们从对方收到的密钥指纹，该密钥指纹可以和初始的密钥指纹相进行对比。这样可以确信的是，对中间人攻击来说，模拟会话参与者的语音比模拟会话参与者的信令更难以实现（一种通常实践方式-基于 Clipper 芯片的安全电话）。

如果 UA 在它们的密钥环上持有对目的地的 AOR 的证书，S/MIME 机制则允许 UA 无需提前声明而直接发送加密的请求。但是，也有一种可能，使用了 AOR 记录地址注册的某个

特定终端设备不持有这个证书，此证书是设备当前用户以前部署使用的证书，因此，此设备将不能正确地处理加密请求，这样就会导致一些可避免的错误信令。特别是，当加密请求经过分叉处理时很可能出现这样的情况。

当密钥关联的是一个具体用户（AOR）而不是和设备（一个 UA）关联时，和 S/MIME 关联的密钥也是非常有用的。当用户在几台设备之间移动使用时，在 UA 之间安全传输私钥是非常困难的。在本规范中没有说明设备如何获得这种私钥。

另外，使用 S/MIME 机制更大难题是它会导致大量消息，特别是当使用 SIP 隧道机制时，具体 SIP 隧道机制的使用参考第 23.4 章描述。因为这个原因，当部署 S/MIME 隧道时，规范推荐使用 TCP 作为传输协议。

#### 26.4.3 TLS

对于 TLS 来说，最受关注的是，TLS 不能在 UDP 上运行；TLS 要求面向连接为基础传输协议，在本规范中特指 TCP。

对于本地出局代理服务器和/或注册服务器来说，它们需要与大量的 UA 同时维护长 TLS 连接，这种操作也是非常困难的。这样就产生了一些有效可扩展的担忧，特别是使用了大量的密码套件。维护长 TLS 连接的冗余是繁琐的，尤其是，当 UA 独自负责 TLS 长连接创建时显得更加繁琐。

TLS 只允许 SIP 实体对和其在网络上相邻的服务器进行验证；TLS 提供了严格的逐跳的安全。TLS 和在本规范指定的其他任何机制都不允许客户端对那些它们不能创建一个直接 TCP 连接的实体认证代理服务器。

#### 26.4.4 SIPS URIs

实际上，在请求路径每一个分段上使用 TLS 意味着通过 TLS 连接的终端 UAS 必须是可达的（可能以 SIPS URI 注册作为 contact 地址）。这是 SIPS 推荐用法。尽管，许多有效的技术架构使用 TLS 保证请求路径的安全，但是，UAS 的最后一跳的安全依赖于一些其他机

制。因此，SIPS 不能保证 TLS 的使用方式是真正的端对端方式。注意，因为很多 UA 不接受入局方向的 TLS 连接，甚至于这些支持 TLS 的 UA 可能要求维护持续的 TLS 连接，关于 TLS 连接在 TLS 局限性在以上部分有具体介绍，作为 UAS，通过这个持续的 TLS 的连接来接收请求。

定位服务不要求提供 SIPS 和 SIPS Request-URI 的绑定关系。尽管，通常情况下定位服务是通过注册服务器计算（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 contact 地址，并且这些工具可以视情况自由地在 SIPS URI 和 SIP URI 之间映射。当定位服务器查询绑定关系时，无论定位服务器是否接收到带 SIPS Request-URI 的请求，定位服务都返回它的 contact 地址。如果重定向服务器正在访问定位服务，由处理重定向的 Contact 头字段的实体决定 contact 地址的属性。

确保将 TLS 用于所有请求分段到目标域是非常复杂的。有一种可能，经过密码处理的认证的代理服务器，这些代理服务器是非规范或者非协定的代理服务器，按照一种方式，这些代理服务器可以选择丢弃和 SIPS 相关的转发规则（一般转发规则，参考第 16.6 节中的描述）。这样的恶意的中间媒体能够将请求从 SIPS URI 方式重定向为 SIP URI 方式，试图降低安全级别。

此外，一个中间媒体可以合法地将一个请求从 SIP 重定向为 SIPS URI。请求接收者的 Request-URI 使用了 SIPS URI 结构模式，因此不能基于 Request-URI 前提下单独假设 SIPS 被使用在整个请求路径上（从客户端以后）。

为了处理这些顾虑，规范推荐请求接收方，并且这些接收方的 Request-URI 中包括 SIP 或者 SIPS URI，请求接收方检查 To 头字段值查看它是否包含了 SIPS URI（不过要注意，如果此 URI 和 To 头字段中的 URI 有相同的结构模式，但是并不相等的话，这也并能构成一个安全漏洞）。虽然客户端可以选择通过不同的方法计算请求中的 Request-URI 和 To 头字段，但是，当使用了 SIPS 时，这种不同的计算方法的差别可能被解释为潜在的安全侵犯，于是，请求的接收方可以因此拒绝这个请求。在请求到达本地管理域之前，接收者也可以检查 Via 头字段链来重复确认整个请求路径是否使用了 TLS。初始的 UAC 也可以使用 S/MIME 体以确保 To 头字段的初始格式可以以端对端方式传输。

如果 UAS 有理由认为，在传输过程中 Request-URI 的结构模式被违规修改，此 UA 应该通知其用户这个潜在的安全漏洞。

作为防止降级攻击的改进的方法，仅接受 SIPS 请求的实体也可以拒绝来自于不安全端口的连接。

毫无疑问，终端用户将识别 SIPS 和 SIP URI 之间的区别，并且终端用户可以手动修改它们来应对刺激。这种操作可能对安全有利，也可能降低安全性。例如，如果攻击者破坏了 DNS 缓存，插入了一个伪造的记录集合-这个记录集合实际上删除了所有对代理服务器的 SIPS 记录，那么，任何穿越此代理服务器的 SIPS 请求可能会失败。但是，当用户看到对 SIPS AOR 的地址重复呼叫失败时，它们可以在一些设备上手动将 SIPS 转换到 SIP 模式，然后重试。当然，有一些保护措施保护安全机制（如果目的地 UA 确实不能正常工作，它会拒绝所有的非 SIPS 请求），但是，它仍是一个值得关注的限制。好的一个方面是，当它们仅和 SIP URI 一起出现，用户仍可以预测“SIPS ‘将是有效的。

## 26.5 Privacy

SIP 消息中经常包含和发送方相关的敏感信息--不仅仅是发送方谈话的内容，还包括和发送方通信的人、他们通信的时间和通信时长以及参与会话的地址。许多应用及其用户要求对无需获知对端的用户隐藏这些私密信息。

注意，还有一些非直接的方法能够使得私密信息被泄漏。如果用户或者服务选择了可达的地址，这个地址可以从人名和组织的隶属关系（它描述了大部分的 AOR 记录地址）猜出的话，通过未入册“电话号码”来确保私密性的传统方法可能会造成安全损害。通过对呼叫方泄密接收方的具体的呼叫方的轨迹，用户定位服务可以侵犯会话邀请接收方的隐私；因此，部署方案可以对基于每个用户，定位类型进行限定，并且根据给出的有效信息来确定呼叫方的类别。这是问题的整个类型，希望在未来继续进行的 SIP 工作中做进一步的研究。

在某些场景中，用户可能希望在传输身份的头字段中隐藏个人信息。这不仅适用于 From 字段和代表请求发起方的相关头字段，也适用于 To--它可能不适合向最终目的地表示快速拨号的别名或对一组目标表示未扩展的身份标识。当请求被路由，这两者都会从 Request-URI 中删除；但是，如果两者最初是相同的，那么，不会在 To 头字段中被修改。因此，因为私密的原因，期望创建一个和 Request-URI 不同的 To 头字段。

## 27 IANA Considerations

SIP 应用中使用的所有 methods 名称、头字段名称、状态代码和选项标签都使用 IANA 注册，通过在 RFC 中的 IANA Considerations 章节的指令执行。本规范指导 IANA 在 <http://www.iana.org/assignments/sip-parameters> 创建了四个新的子注册：选项标签、警告代码、Methods 和响应代码，这些子注册已经被添加到现有的头字段的子注册中。

### 27.1 Option Tags

本规范创建了选项标签 -Option Tags 子注册，该子注册项在 <http://www.iana.org/assignments/sip-parameters> 有具体描述。

选项标签使用在一些头字段中，例如 Require、Supported、Proxy-Require 和 Unsupport 这类的头字段中，这些头字段用于支持对扩展的 SIP 兼容机制（参考第 19.2 节）。选项标签本身是一个字符串，它与一个特定的 SIP 选项关联（这里就是一个扩展）。它可以确认 SIP 端的选择。

当选项标签在标准路线 RFC 中发布时，通过 IANA 对其进行注册。RFC 的 IANA Consideration 章节包含以下信息，这些信息连同 RFC 的发布号一起出现在 IANA 注册。

- 选项标签的名称。这个名称长度可任意长度，但应该不能超过 20 个字符的长度。名称必须仅由字母或数字（参考第 25 章）构成。
- 描述扩展的描述文本。

## 27.2 Warn-Codes

本规范创建了警告代码子注册，该子注册项在 <http://www.iana.org/assignments/sip-parameters> 有具体描述，最初所有警告代码在第 20.43 节列出。附加的警告代码由 RFC 发布注册。

警告代码表的描述文本如下：

当事务失败的原因是来自于会话描述协议(SDP)(RFC2327[参考链接 1] )问题，警告代码在 SIP 响应中对状态代码提供消息补充。

警告代码由三个数字组成。第一位数是 “3” 表示是特指 SIP 的警告。除非在将来的规范中 描述使用除 3XX 以外的警告代码，仅可以注册 3XX 警告代码。

警告代码 300 至 329 是预留的，用于指示会话描述中关键字的问题；警告代码 330 至 339 对会话描述中要求的基础网络服务作出警告；370 至 379 是对在会话描述中所要求的量化的 QOS 参数相关的问题作出警告；最后，390 至 399 是其他杂项警告，此类警告不属于以上任何一种类别。

## 27.3 Header Field Names

此选项废弃了关于头子注册项在 <http://www.iana.org/assignments/sip-parameters> 的 IANA 指令。

为了注册一个新的头字段名称，在 RFC 发布中需要提供以下信息：

- RFC 编号，头已在此编号中注册；
- 已注册的头字段名称；
- 如果已定义了头字段，则需要提供该头字段的压缩格式的版本；

一些通常被广泛使用的头字段分配了一个单字母的压缩形式(参考第 7.3.3 节)。分配头字段的压缩形式需要经过 SIP 工作组审阅，遵守 RFC 发布才可以分配。

## 27.4 Method and Response Codes

本规范创建了 method 和其响应代码的子注册项，这些子注册项在 <http://www.iana.org/assignments/sip-parameters> 下有具体描述，并且按照下述形式对其进行初始化处理。初始的 method 表如下：

INVITE	<a href="#">[RFC3261]</a>
ACK	<a href="#">[RFC3261]</a>
BYE	<a href="#">[RFC3261]</a>
CANCEL	<a href="#">[RFC3261]</a>
REGISTER	<a href="#">[RFC3261]</a>
OPTIONS	<a href="#">[RFC3261]</a>
INFO	<a href="#">[RFC2976]</a>

响应代码表最初是从第 21 章开始引入，这一部分内容标注了消息、成功、重定向、客户端错误、服务器错误和全局失败内容。表的格式如下：

Type (e.g., Informational)		
Number	Default Reason Phrase	<a href="#">[RFC3261]</a>

为了注册一个新的响应代码或 method 名称，在 RFC 发布中需要提供以下信息：

- RFC 编号，响应代码或 method 已在此编号中注册；
- 已注册的响应代码编号或者 method 名称；
- 如果可行，需要提供响应代码对应的默认原因短语；

## 27.5 The "message/sip" MIME type

本规范注册了 “message/sip” MIME 媒体类型，以便允许让所有在 SIP 中的 SIP 消息作为隧道整体来处理，其基本目的是为了端到端的安全。以下信息定义了此媒体的类型：

```

Media type name: message
Media subtype name: sip
Required parameters: none

```

Optional parameters: version

版本：消息的 SIP 版本编号(例如，“2.0” )。如果版本号不存在，则默认版本号为“2.0”。

编码模式：SIP 消息是由一个 8bit 头组成，其后（可选）跟一个二进制 MIME 数据象。因此，必须将 SIP 消息视为二进制来看待。在正常环境下，SIP 消息的传输可以通过二进制能力传输，无需特殊编码。

安全考虑：见以下说明

作为一个安全机制，该用法的目的和示例与第 23.4 节中所描述的 S/MIME 细节一致。

## 27.6 New Content-Disposition Parameter Registrations

本规范注册了四个新 Content-Disposition 报头 “disposition-type”，它们分别是：警报、图标、会话和渲染。本规范作者要求在 IANA 注册中为 Content-Disposition 记录这些值。

这些“disposition-types”的描述，包括目的和示例，参考第 20.11 章节。对 IANA 注册合适的简短描述如下：

alert	消息体是一个自定义振铃音来提醒用户
icon	消息体作为图标显示给用户
render	消息体应该对用户显示
session	消息体描述了一个通信会话，例如，就像 RFC2327 的 SDP 消息体

## 28 Changes From RFC 2543

此 RFC 是 RFC2543 的修订版。它支持大部分的向后和 RFC2543 兼容。这里所描述的修改是修正了在 RFC2543 中发现的许多错误，并提供了在 RFC2543 中没有详细描述的场景信息。这里，此协议已通过更清晰的层级模型来表现。

我们把这些不同分拆为不同的功能处理流程，这些处理流程和 RFC2543 相比有很大不同，这样的修正影响了某些场景中的互操作性或正确操作，而且功能处理流程不同于 RFC2543，但这不是互操作性问题的一个潜在源头。还有许多修正需要澄清，在本文档中就不再逐一说明。

## 28.1 Major Functional Changes

- 如果 UAC 希望在呼叫应答前终止呼叫，它就发送 CANCEL 请求。如果最初的 INVITE 仍然返回 2xx，UAC 然后就会发送 BYE 请求。仅能在现存的呼叫腿上发送 BYE（在此规范中则为在呼叫的 dialog 中），然而，在 RFC2543 中可以在任何时间发送 BYE。
- 转换 SIP BNF 以满足 RFC2234 规范。
- SIP URL BNF 变得更加通用，允许在用户部分使用更多的字符集。另外，比较规则已简化，基本上已经对字母大小不敏感，并且对出现在参数中的对比处理描述进行了完善。最大的修改是带默认参数值的 URI 不能和无参数的 URI 进行匹配。
- 移除了 Via 隐藏。它已有一系列的信任问题，因为它依靠下一跳来执行这个模糊流程。相反，Via 隐藏可以在有状态代理中作为一个本地部署选择来完成，因此，这里不再说明。
- 在 RFC2543 中，CANCEL 和 INVITE 事务是混合在一起的。在本规范中它们分开的。当用户先发送 INVITE 请求，然后再发送 CANCEL 请求，INVITE 事务仍然正常终止。UAS 需要用 487 响应来应答以前的 INVITE 请求。
- 同样的，在 RFC2543 中，CANCEL 和 BYE 事务也是混合；在 RFC2543 中，允许当 UAS 收到 BYE 请求后，它可以不对 INVITE 请求发送响应。在本规范中则不允许这样的操作。初始 INVITE 请求需要响应。
- RFC2543 中，UA 仅需要支持 UDP。在本规范中，UA 需要支持 UDP 和 TCP。
- RFC2543 中，在多个挑战事件中，分叉代理仅仅从下游的网元中传递一个挑战。在本规范中，代理应该收集所有的挑战，并且把收集的挑战置于转发响应中。

- 在摘要安全凭证中，需要用引号引用 URL；这一点在 RFC2617 和 RFC2069 里未明确此用法，并且这两个规范在这一点上并不一致。
- SDP 处理流程已经分解为单独的规范[参考链接 13]，各自处理在其规范中进行了说明，更全面地指定为一个正式的 offer/answer 交互处理流程，该处理流程可通过 sip 有效地开辟新的通道。SDP 被允许在 INVITE/200 和 200/ACK 来支持基础 SIP 部署；RFC2543 略微提及了在单个事务中，INVITE、200 和 ACK 中使用 SDP 的能力，但这一点没有很好制定出细节流程。更复杂的 SDP 使用在扩展中支持。
- 在 URI 和 Via 头字段里对 IPv6 增加了全面的支持。在 Via 中支持 IPv6 要求 Via 的头字段参数带方括号和冒号字符。这些符号在之前版本中是不被允许使用的。从理论上来讲，这会引起与先前部署的互操作性问题。但是，我们已注意到大多数部署接受在参数里的任何非控制 ASCII 字符。
- DNS SRV 处理流程在分开的规范[参考链接 4]中进行了说明。这个流程使用了 SRV 和 NAPTR 的资源记录，并且就像 RFC2543 里所描述的那样，不再从跨 SRV 记录中合并数据。
- 回环检测是可选的，它被 Max-Forwards 的一个强制用法代替。RFC2543 中的回环检测流程有一个严重的 bug，当“螺旋”不是错误状态，这个 bug 会将“spirals-螺旋”报告为一个错误状态。在本规范中，可选回环检测流程得到了完整地和正确地说明。
- 在本规范中标签的使用是强制性的(在 RFC2543 里标签的使用是可选的)，因为在本规范中，标签是 dialog 确认的基本构造块。
- 增加的 Supported 头字段允许客户端向服务器端指示支持的扩展，并且服务器端在响应中应用这些扩展，在响应中通过 Require 指示这些扩展的用法。
- 几个头字段的扩展参数已从 BNF 中丢失，本规范中增加了这些参数。

- 在 RFC2543 中没有详细规定 Route 和 Record-Route 的结构处理，也没有正确的思路。这些问题在本规范作了重新梳理(而且非常的简化)，这可以说是一个最大的修改。考虑到不使用“预装载路由”的部署环境，本规范提供了向后兼容，这种环境中初始的请求有 Route 头字段值集，这些字段值集是通过某种方式从外部 Record-Route 获得。在这些情况下，新的机制是不能实现互操作的。
- RFC2543 中的消息行可用 CR、LF 或 CRLF 终止。本规范仅允许使用 CRLF 来终止消息行。
- 关于 CANCEL 和 ACK 的 Route 使用在 RFC2543 中没有被明确定义。在本规范中它有详细的说明；如果一个请求已有 Route 头字段，那么它的针对请求的非-2xx 响应的 CANCEL 和 ACK 需要携带相同的 Route 头字段值。它的对 2xx 响应的 ACK 则使用 Route 字段值，此值是从 2xx 响应的 Record-Route 字段值中获知的。
- RFC2543 规范中允许在一个单 UDP 数据包中支持多个请求。此用法在本规范中被移除。
- 在 Expires 的头字段和参数中使用绝对时间已被移除。因为这将引起时间不同步的网元之间的互操作性问题，网元设备之间的时间不同步会经常发生。在本规范中使用相对时间代替。
- 强制所有网元使用 Via 头字段值的分支参数。在本规范中，此分支参数充当唯一事务标识符的角色。这样就避免了在 RFC2543 中复杂的，并且容易引入 bug 的事务确认规则。在参数值中使用 magic cookie 是决定如果前一个跳跃点是否已成为了全局唯一参数，并且当它未出现时，它可以回落到原来的规则进行对比。这样，就保证了互操作性。
- RFC2543 中，一个 TCP 连接的关闭等同于 CANCEL。几乎不可能在代理之间部署 TCP 连接（并且是错误的）。此使用方式已经在本规范中被淘汰，因此 TCP 连接状态和 SIP 处理之间就不存在耦合。

- 当一个 UA 的对端正处于一个处理状态时，UA 是否可以向此对端初始化一个新的事务，这一点在 RFC2543 中没有提及。在本规范中特别说明了这一点，本规范允许发送一个非 -INVITE 请求，不允许发送 INVITE 请求。
- 本规范移除了 PGP。因为它没有被清楚说明，并且它与更完整的 PGP MIME 不兼容。在本规范中，PGP 被 S/MIME 取代。
- 本规范为端到端 TLS 增加了“sips” URI 结构模式。这种结构体不能向后和 RFC2543 兼容。当现有网元接收到了一个请求，在请求的 Request-URI 携带了 SIPS URI 结构模式的话，网元很可能拒绝此请求。这实际是一个特性：这样的功能确保了只有呼叫路径上的所有跳跃点都是安全的，才传输对 SIPS URI 的呼叫。
- 本规范通过 TLS 添加了额外的安全特性，而且在此规范中有大量的描述，并且提供了对完整的安全性考量的章节。
- 在 RFC2543 中，不要求代理对上游转发从 101 到 199 范围内的临时响应。在本规范中要求代理必须转发。这一点非常重要，因为许多后续的特性都依赖于从 101 到 199 范围内的所有临时响应的传输。
- 在 RFC2543 中，很少提及对 503 响应代码的描述。发现它经常使用在代理中，用于指示失败或者过载环境。这要求我们对它予以特殊的处理。特别是，503 接收会触发联系在 DNS SRV 查询结果中得到的下一个网元。另外，在某些环境中，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，但没有做太多描述。在本规范中移除了这一部分，替代它的是 RFC2617，允许 RFC2617 中相互认证流程。
- 直到在 UA 已收到初始 INVITE 的 ACK 以后，UA 才能发送呼叫的 BYE 请求。这在 RFC2543 规范中是被允许的，但是这可能会导致极端情况出现。

- 直到在 UA 或者代理获得了请求的临时响应后，UA 或者代理才能向事务发送 CANCEL。这在 RFC2543 规范中是被允许的，但是这可能会导致极端情况出现。
- 本规范废弃了在注册中的 action 参数。对于某些有用的服务而言这还是不够的，并且当在代理中使用应用程序处理时，它会引起冲突。
- RFC2543 有很多关于多播的特殊用例。例如，某些响应被删除，定时器被调整等等。而在本规范中，多播则扮演着非常有限的角色，并且和单播相反，使用多播时则不会对协议操作带来影响。对多播的结果限制已经在规范中说明。
- 在本规范中完全删掉了基本认证，并且其使用方式已被禁用。
- 在本规范中，代理在接收到 6xx 时不再立即将其转发，相反，代理会马上取消待处理的分支。这样就避免了可能的出现的极端状态，这种极端状态会导致 UAC 在收到一个 2xx 响应之后马上再收到一个 6xx 响应。除了极端状态以外的所有用例中，其结果都是相同的--向上游转发 6xx 响应。
- RFC2543 中没有解决请求消息合并的问题。这种情况是可能发生的，当请求在一个代理处执行了分叉复制处理，随后这个请求会在一个网元处再次进行合并处理，这就是请求合并。合并处理仅发生在 UA 端，而且定义了拒绝呼叫的流程，此流程拒绝除了第一个请求之外的所有呼叫请求。

## 28.2 Minor Functional Changes

- 对用户的可选内容呈现添加了 Alert-Info、Error-Info 和 Call-Info 的头字段。
- 添加了 Content-Language、Content-Disposition 和 MIME-Version 的头字段。
- 添加了“glare handling-异常再协同处理”机制，用来处理这种情况，当双方同时向对方发送 re-INVITE 请求。本规范使用新的 491 (Request Pending) 错误代码。

- 添加了 In-Reply-To 和 Reply-To 的头字段来支持未接呼叫或者消息在稍后时间的返回。
- 添加了 TLS 和 SCTP 作为有效的 SIP 传输机制。
- 在 RFC2543 规范中，有多种机制描述了在任何呼叫期间处理失败的流程，在本规范中，这些机制被整合，即通过发送 BYE 请求来终止呼叫。
- 在 RFC2543 中，要求了一个通过 TCP 上对 INVITE 的响应重传，但是需要注意的是，实际上仅对 2xx 响应时才需要这个 INVITE 重传。那是因为不完善的协议层结构。在本规范中，定义了多个连贯的事务层，所以不再需要 RFC2543 中的规定机制。仅需要通过 TCP 对 INVITE 的 2xx 响应进行重传。
- 现在客户端和服务器事务机是通过超时驱动的，而不是由重传计数来驱动的。这样就允许为 TCP 和 UDP 连接来指定合适的状态机。
- 在 REGISTER 的响应里使用 Date 头字段，为用户代理提供一种简单方法实现日期的自动配置。
- 本规范允许注册服务器拒绝用户注册，因为其用户注册端设置了过短的超时时间。为此目的，本规范定义了 423 响应代码和 Min-Expires 字段。

## 29 Normative References

- [1] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", [RFC 2327](#), April 1998.
- [2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [3] Resnick, P., "Internet Message Format", [RFC 2822](#), April 2001.
- [4] Rosenberg, J. and H. Schulzrinne, "SIP: Locating SIP Servers", [RFC 3263](#), June 2002.
- [5] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", [RFC 2396](#), August 1998.
- [6] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", [RFC 3268](#), June 2002.
- [7] Yergeau, F., "UTF-8, a transformation format of ISO 10646", [RFC 2279](#), January 1998.
- [8] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", [RFC 2616](#), June 1999.
- [9] Vaha-Sipila, A., "URLs for Telephone Calls", [RFC 2806](#), April 2000.
- [10] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", [RFC 2234](#), November 1997.
- [11] Freed, F. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", [RFC 2046](#), November 1996.
- [12] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", [RFC 1750](#), December 1994.
- [13] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with SDP", [RFC 3264](#), June 2002.
- [14] Postel, J., "User Datagram Protocol", STD 6, [RFC 768](#), August 1980.
- [15] Postel, J., "DoD Standard Transmission Control Protocol", [RFC 761](#), January 1980.

- [16] Stewart, R., Xie, Q., Morneau, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson, "Stream Control Transmission Protocol", [RFC 2960](#), October 2000.
- [17] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and L. Stewart, "HTTP authentication: Basic and Digest Access Authentication", [RFC 2617](#), June 1999.
- [18] Troost, R., Dorner, S. and K. Moore, "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field", [RFC 2183](#), August 1997.
- [19] Zimmerer, E., Peterson, J., Vemuri, A., Ong, L., Audet, F., Watson, M. and M. Zonoun, "MIME media types for ISUP and QSIG Objects", [RFC 3204](#), December 2001.
- [20] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, [RFC 1123](#), October 1989.
- [21] Alvestrand, H., "IETF Policy on Character Sets and Languages", [BCP 18](#), [RFC 2277](#), January 1998.
- [22] Galvin, J., Murphy, S., Crocker, S. and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", [RFC 1847](#), October 1995.
- [23] Housley, R., "Cryptographic Message Syntax", [RFC 2630](#), June 1999.
- [24] Ramsdell B., "S/MIME Version 3 Message Specification", [RFC 2633](#), June 1999.
- [25] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", [RFC 2246](#), January 1999.  
▼
- [26] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", [RFC 2401](#), November 1998.

## 30 Informative References

- [27] R. Pandya, "Emerging mobile and personal communication systems," IEEE Communications Magazine, Vol. 33, pp. 44--52, June 1995.
- [28] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", [RFC 1889](#), January 1996.

www.sip.org.cn

- [29] Schulzrinne, H., Rao, R. and R. Lanphier, "Real Time Streaming Protocol (RTSP)", [RFC 2326](#), April 1998.
- [30] Cuervo, F., Greene, N., Rayhan, A., Huitema, C., Rosen, B. and J. Segers, "Megaco Protocol Version 1.0", [RFC 3015](#), November 2000.
- [31] Handley, M., Schulzrinne, H., Schooler, E. and J. Rosenberg, "SIP: Session Initiation Protocol", [RFC 2543](#), March 1999.
- [32] Hoffman, P., Masinter, L. and J. Zawinski, "The mailto URL scheme", [RFC 2368](#), July 1998.
- [33] E. M. Schooler, "A multicast user directory service for synchronous rendezvous," Master's Thesis CS-TR-96-18, Department of Computer Science, California Institute of Technology, Pasadena, California, Aug. 1996.
- [34] Donovan, S., "The SIP INFO Method", [RFC 2976](#), October 2000.
- [35] Rivest, R., "The MD5 Message-Digest Algorithm", [RFC 1321](#), April 1992.
- [36] Dawson, F. and T. Howes, "vCard MIME Directory Profile", [RFC 2426](#), September 1998.
- [37] Good, G., "The LDAP Data Interchange Format (LDIF) - Technical Specification", [RFC 2849](#), June 2000.
- [38] Palme, J., "Common Internet Message Headers", [RFC 2076](#), February 1997.
- [39] Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P., Luotonen, A., Sink, E. and L. Stewart, "An Extension to HTTP: Digest Access Authentication", [RFC 2069](#), January 1997.
- [40] Johnston, A., Donovan, S., Sparks, R., Cunningham, C., Willis, D., Rosenberg, J., Summers, K. and H. Schulzrinne, "SIP Call Flow Examples", Work in Progress.
- [41] E. M. Schooler, "Case study: multimedia conference control in a packet-switched teleconferencing system," Journal of Internetworking: Research and Experience, Vol. 4, pp. 99--120, June 1993. ISI reprint series ISI/RS-93-359.

- [42] H. Schulzrinne, "Personal mobility for multimedia services in the Internet," in European Workshop on Interactive Distributed Multimedia Systems and Services (IDMS), (Berlin, Germany), Mar. 1996.
- [43] Floyd, S., "Congestion Control Principles", [RFC 2914](#), September 2000.

## A Table of Timer Values

表四汇总了本规范中使用的各种定时器含义和默认值设置。

定时器	值	所在章节	含义
T1	500 ms 默认	第 17.1.1.1	预估的 RTT 时间
T2	4s	第 17.1.2.2	对非-INVITE 请求和 INVITE 的响应的最大重传时间间隔
T4	5s	第 17.1.2.2	消息在网络保存的最长时间
Timer A	初始 T1 值	第 17.1.1.2	INVITE 请求重传间隔, 仅 UDP
Timer B	$64 \times T1$	第 17.1.1.2	INVITE 事务超时定时器
Timer C	> 3 min	第 16.6, bullet 11	代理 INVITE 事务超时
Timer D	UDP 是 > 32s, 0s 是 TCP/SCTP	第 17.1.1.2	响应重传的等待时间
Timer E	初始的 T1 值	第 17.1.2.2	非-INVITE 请求重传间隔, 仅对 UDP。
Timer F	$64 \times T1$	第 17.1.2.2	非-INVITE 事务超时定时器
Timer G	初始 T1 值	第 17.2.1	INVITE 响应重传间隔
Timer H	$64 \times T1$	第 17.2.1	接收 ACK 的等待时间
Timer I	UDP 为 T4, TCP/SCTP 为 0s	第 17.2.1	ACK 重传等待时间
Timer J	UDP 是 $64 \times T1$ , TCP/SCTP 是 0s	第 17.2.2	非-INVITE 请求重传等待时间
Timer K	UDP 为 T4, TCP/SCTP 为 0s	第 17.1.2.2	响应重传等待时间

Table 4: Summary of timers

## Acknowledgments

## Acknowledgments

We wish to thank the members of the IETF MMUSIC and SIP WGs for their comments and suggestions. Detailed comments were provided by Ofir Arkin, Brian Bidulock, Jim Buller, Neil Deason, Dave Devanathan, Keith Drage, Bill Fenner, Cedric Fluckiger, Yaron Goland, John Hearty, Bernie Hoeneisen, Jo Hornsby, Phil Hoffer, Christian Huitema, Hisham Khartabil, Jean Jervis, Gadi Karmi, Peter Kjellerstedt, Anders Kristensen, Jonathan Lennox, Gethin Liddell, Allison Mankin, William Marshall, Rohan Mahy, Keith Moore, Vern Paxson, Bob Penfield, Moshe J. Sambol, Chip Sharp, Igor Slepchin, Eric Tremblay, and Rick Workman.

Brian Rosen provided the compiled BNF.

Jean Mahoney provided technical writing assistance.

This work is based, *inter alia*, on [[41](#),[42](#)].



## Authors' Addresses

Authors addresses are listed alphabetically for the editors, the writers, and then the original authors of [RFC 2543](#). All listed authors actively contributed large amounts of text to this document.

Jonathan Rosenberg  
dynamicsoft  
72 Eagle Rock Ave  
East Hanover, NJ 07936  
USA

EMail: [jdrosen@dynamicsoft.com](mailto:jdrosen@dynamicsoft.com)

Henning Schulzrinne  
Dept. of Computer Science  
Columbia University  
1214 Amsterdam Avenue  
New York, NY 10027  
USA

EMail: [schulzrinne@cs.columbia.edu](mailto:schulzrinne@cs.columbia.edu)

Gonzalo Camarillo  
Ericsson  
Advanced Signalling Research Lab.  
FIN-02420 Jorvas  
Finland

EMail: [Gonzalo.Camarillo@ericsson.com](mailto:Gonzalo.Camarillo@ericsson.com)

Alan Johnston  
WorldCom  
100 South 4th Street  
St. Louis, MO 63102  
USA

EMail: [alan.johnston@wcom.com](mailto:alan.johnston@wcom.com)

Jon Peterson  
NeuStar, Inc  
1800 Sutter Street, Suite 570  
Concord, CA 94520  
USA

EMail: [jon.peterson@neustar.com](mailto:jon.peterson@neustar.com)

Robert Sparks  
dynamicsoft, Inc.  
5100 Tennyson Parkway  
Suite 1200  
Plano, Texas 75024  
USA

EMail: [rsparks@dynamicsoft.com](mailto:rsparks@dynamicsoft.com)

Mark Handley  
International Computer Science Institute  
1947 Center St, Suite 600  
Berkeley, CA 94704  
USA

EMail: [mjh@icir.org](mailto:mjh@icir.org)

Eve Schooler  
AT&T Labs-Research  
75 Willow Road  
Menlo Park, CA 94025  
USA

EMail: [schooler@research.att.com](mailto:schooler@research.att.com)

### Full Copyright Statement

Copyright (C) The Internet Society (2002). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

### Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

