Real Time Streaming Protocol(RTSP) 中文版

（前十章）

1 绪论

1.1 目的

实时流协议（RTSP）建立并控制一个或几个时间同步的连续流媒体。尽管连续媒体流与控制流有可能交叉，但 RTSP 本身通常并不发送连续媒体流。换言之，RTSP 充当多媒体服务器的网络远程控制。

表示描述(presentation description)定义了被控流，但本文并没有定义表示描述的格式。

这里没有使用 RTSP 连接的概念，而由 RTSP 会话(session)代替（每次服务由服务器端保持一个带标签的会话）。RTSP 会话没有绑定到传输层连接（如 TCP 连接）。因为虽然在 RTSP 会话期间，RTSP 客户端可打开或关闭多个对服务器端的可靠传输连接以发出 RTSP 请求。但此外，也可能使用无连接传输协议，比如用 UDP 发送 RTSP 请求。

RTSP 控制的流可能用到 RTP，但 RTSP 操作并不依赖用于携带连续媒体的传输机制。实时流协议在语法和操作上与 HTTP/1.1 类似，因此 HTTP 的扩展机制大都可加入 RTSP。尽管如此，RTSP 在很多方面还是和 HTTP 有很大的不同：

- RTSP 引入了很多新方法并且有不同的协议标识符。
- RTSP 服务器在大多数默认情况下需要维持一个状态，但 HTTP 是无状态协议。
- RTSP 客户机和服务器都可以发出请求。
- 数据由另一个协议传送（有一特例除外）。
- RTSP 使用 ISO 10646(UTF-8) 而不是 ISO 8859-1，以配合当前 HTML 的国际化。
- RTSP 使用 URI 请求时包含绝对 URI。而由于历史原因造成的向后兼容性问题，HTTP/1.1 只在请求中包含绝对路径，把主机名放入单独的标题域中。
  这使得“虚拟主机”实现更为简便，一个单独 IP 地址的主机可虚拟为几个文件树主机。

协议支持的操作如下：

- 从媒体服务器上检索媒体：
  用户可通过 HTTP 或其它方法请求一个表示描述。如表示是组播，表示描述就包含用于连续媒体的的组播地址和端口。如表示仅通过单播发送给用户，用户为了安全应提供目的地址。
- 媒体服务器邀请进入会议：
  媒体服务器可被邀请参加正进行的会议，或回放媒体，或记录其中一部分，或全部。这种模式在分布式教育应用上很有用，会议中几方可轮流按远程控制按钮。
- 将媒体加到现成讲座中：
  如服务器告诉用户可获得附加媒体内容，对现场讲座显得尤其有用。
  如 HTTP/1.1 中类似，RTSP 请求可由代理、通道与缓存处理。

1.2 要求

在本文档中的关键字“必须”，“一定不能”，“要求”，“会”，“不会”，“应该”，“不

应该”，“被推荐的”，“可以”，和“可选择的”都在 RFC2119 中解释。

1.3 术语

一些术语原由 HTTP/1.1 采用。在 HTTP/1.1 中定义的术语这里不再列举。

合控制：服务器使用单条时间线对多个流的控制。对音频/视频回馈来讲，这就意味着客户端仅需发送一条播放或者暂停消息就可同时控制音频和视频的回馈。

会议：多方参与的多媒体表示，这里的多方意味着大于或者等于一方。

客户端：指请求媒体服务器上连续流媒体数据的客户端。

连接：两个应用程序以通讯为目的在传输层建立虚拟电路。

容器文件：可以容纳多个共同播放时包含表示(presentation)的媒体流的文件。RTSP 服务器可以为这些容器文件提供合控制，但容器文件的概念本身并不是本协议内容。

连续媒体：接受器和数据源之间存在时序关系的数据。也就是说，接受器需要重新产生存在于源数据中的时序关系。最普通的连续媒体的例子是音频和动画视频。连续媒体可以是实时的（但是不交互的），它们在源和接受器之间是一种紧密的时序关系；或者是流的形式，这种关系就没有那么严格了。

实体：作为请求或者回应的有效负荷传输的信息。由以实体标题域（entity-header field）形式存在的元信息和以实体主体（entity body）形式存在的内容组成，如第八章所述。

媒体的初始化：数据类型/编码的具体初始化，这些包括时钟输率，颜色表等。用户请求媒体回放的任何独立传输信息，是在创建流时初始化媒体流相位时产生的。

媒体参数:针对回放前或回放过程中有可能改变的媒体类型而专门设定的参数。

媒体服务器：可对一个或多个媒体流提供回放和录制服务的服务器。同一个表示(presentation)中不同的媒体流可能来自于不同的媒体服务器。媒体服务器可以建立在作为传送请求表示(presentation)的 Web 服务器的主机上，也可以建立在不同的主机上。

媒体服务器重定向：重新定向媒体客户端到另外一个媒体服务器。

(媒体)流：单个媒体实例，比如，在应用中共用音频流或视频流。当使用 RTP 时，流包括由 RTP 会话(session)中源所创建的所有 RTP 和 RTCP 包。这和定义 DSM-CC 流时相同。

消息：RTSP 通讯的基本单元。由 15 章语法定义的一串八位位组组成，并通过连接或者无连接协议传送。

参与者：一个会议成员。参与者可以是机器，比如是媒体记录或回放服务器。

表示(presentation)：对用户请求所回馈的一组流，其使用下面的表示描述(presentation description)形式。在本文中的多数情况下，其意味着对流进行总体控制，但这并不是必须的。

表示描述(presentation description)：表示描述包含在表示(presentation)中一个或者多个媒体流的信息。比如，编码，网络地址和内容的信息。另外，其他 IETF 协议，如 SDP 协议使用会话(session)代替现场 presentation。表示描述可以采用包括会话描述(session description)SDP 在内的多种格式。

回应：RTSP 回应。如果能理解 HTTP 回应，就能清楚的理解 RTSP 回应。

请求:RTSP 请求。如果能理解 HTTP 请求，就能清楚的理解 RTSP 请求。

RTSP 会话(session)：RTSP 交互的全过程。比如，一个电影的观看过程。会话(session)包括由客户端建立连续媒体流传输机制(SETUP)，使用播放(PLAY)或录

制(RECORD)开始传送流，用停止(TEARDOWN)关闭流。
传输初始化：客户端和服务器端之间传输信息（端口号，传输协议等）的交互。

1.4 协议特点

RTSP 特性如下：

可扩展性：

RTSP 中很容易加入新方法和参数。

易解析：

RTSP 可由标准 HTTP 或 MIME 解析器解析。

安全：

RTSP 使用网页安全机制。所有 HTTP 授权机制如 basic 和 digest 授权都可直接使用。也可以传输层或网络层安全机制。

独立于传输：

RTSP 可使用不可靠数据报协议（UDP）、可靠数据报协议（RDP），如要实现应用级可靠，可使用可靠流协议。

多服务器支持：

表示(presentation)中的每个流可放在不同服务器上，用户端自动同不同服务器建立几个并发控制连接，媒体同步在传输层执行。

记录设备控制：

协议可控制记录和回放设备，也可控制可在记录和回放切换的设备。

流控与会议开始分离：

流控与邀请媒体服务器入会分离；仅要求会议初始化协议提供，或可用来创建唯一会议标识号。特殊情况下， SIP 或 H.323 可用来邀请服务器入会。

适合专业应用：

通过 SMPTE 时标，RTSP 支持帧级精度，允许远程数字编辑。

表示描述中立：

协议没强加特殊表示或元文件，可传达所用格式类型；然而，表示描述至少必须包含一个 RTSP URI。

代理与防火墙友好：

协议可由应用和传输层防火墙处理。防火墙需要理解 SETUP 方法，为 UDP 媒体流打开一个\"缺口\"。

HTTP 友好：

此处，RTSP 明智的采用 HTTP 观念，使现在结构都可重用。结构包括 Internet 内容选择平台（PICS）。由于在大多数情况下控制连续媒体需要服务器状态， RTSP 不仅仅向 HTTP 添加方法。

适当的服务器控制：

如用户能启动一个流，它必须也能停止一个流。 服务器不能启动一个用户不能停止的流。

传输协调：

实际处理连续媒体流前，用户可协调传输方法。

性能协调：

如基本特征无效，必须有一些清理机制让用户决定那种方法没生效。这允许用户提出适合的用户界面。 例如，如果不允许寻找，用户界面必定能禁止位置条滑动。

以前要求 RTSP 必须能支持多用户，但现在得出一个更好的方法就是保证 RTSP 能很容

易扩展成支持多用户即可。因为流的标志可以被多个控制流使用，因此"远程通过"成为可能。协议不涉及到多个客户端如何协调入口，其由下层"社会协议"或其他下层控制机制提供。

1.5 RTSP 扩展

由于不是所有媒体服务器有着相同的功能，媒体服务器有必要支持不同请求集。例如：

- 服务器可能只须支持回放，因此不必不支持录制功能。
- 对于支持现场播放的服务器可能不支持寻找功能。
- 一些服务器可能不支持设置流参数，因此不支持 GET_PARAMETER 和 SET_PARAMETER。但服务器应该实现所有 12 章中要求的标题域。
- 表示描述(presentation description)应当保证不提出服务器不支持的功能，这种情形和 HTTP/1.1 中[H19.6]描述方法不支持 across server 的情形一致。
- RTSP 可以如下三种方式扩展，这里以改变大小排序：
  以新参数扩展。如用户需要拒绝通知，而方法扩展不支持，相应标记就加入要求的段中。
  加入新方法。如信息接收者不理解请求，返回 501 错误代码（还未实现），发送者不应再次尝试这种方法。用户可使用 OPTIONS 方法查询服务器支持的方法。服务器使用公共回应标题列出支持的方法。
  定义新版本协议，允许改变所有部分。（除了协议版本号位置）

1.6 操作模式

每个表示和媒体流可用 RTSP URL 识别。表示组成的整个表示与媒体属性由表示描述(presentation description)文件定义，表示描述格式不在本协议中定义。使用 HTTP 或其它途径用户可获得这个文件，它没有必要保存在媒体服务器上。

为了说明，假设表示描述(presentation description)描述了多个表示(presentation)，其中每个表示(presentation)维持了一个公共时间轴。为简化说明，且不失一般性，假定表示描述(presentation description)的确包含这样一个表示(presentation)。表示(presentation)可包含多个媒体流。

表示描述(presentation description)即组成表示的流的描述，包括它们的编码，语言和使用户可以选择最符合要求媒体的其他参数。在表示描述中，被 RTSP 分别控制的媒体流由 RTSP URL 表示。RTSP URL 指出了处理具体媒体流的服务器以及存在于该服务器上流的名字。多个媒体流可以放到不同的服务器上，比如音频和视频流可以分别放到不同服务器而负载共享。描述（description)还列出了服务器传输可使用的方法。

除媒体参数外，网络目标地址和端口也需要决定。下面区分几种操作模式：

单播：

以用户选择的端口号将媒体发送到 RTSP 请求源。

组播，服务器选择地址：

媒体服务器选择组播地址和端口，这是现场直播或准点播常用的方式。

组播，用户选择地址：

如服务器加入正在进行的组播会议，组播地址、端口和密匙由会议描述给出。

1.7 RTSP 状态

RTSP 控制通过单独协议发送的流，与控制通道无关。例如，RTSP 控制可通过 TCP 连接，而数据流通过 UDP。因此，即使媒体服务器没有收到请求，数据也会继续发送。在会话生命期，单个媒体流可通过不同 TCP 连接顺序发出请求来控制。所以，服务器需要维持能联系流与 RTSP 请求的会话状态。

RTSP 中很多方法与状态无关，但下列方法在定义服务器流资源的分配与应用上起着重要的用：

SETUP：

让服务器给流分配资源，启动 RTSP 会话。

PLAY 与 RECORD：

启动 SETUP 分配流的数据传输。

PAUSE：

临时停止流，而不释放服务器资源。

TEARDOWN：

释放流的资源，RTSP 会话停止。

标识状态的 RTSP 方法使用会话(session)标题域识别 RTSP 会话，为回应 SETUP 请求，服务器生成会话标识。

1.8 与其他协议关系

RTSP 在功能上与 HTTP 有重叠，与 HTTP 相互作用体现在与流内容的初始接触是通过网页的。目前的协议规范目的在于允许在网页服务器与实现 RTSP 媒体服务器之间存在不同传递点。例如，表示描述(presentation description)可通过 HTTP 和 RTSP 检索，这降低了浏览器的往返传递，也允许独立 RTSP 服务器与用户不全依靠 HTTP。

但是，RTSP 与 HTTP 的本质差别在于数据发送以不同协议进行。HTTP 是不对称协议，用户发出请求，服务器作出回应。RTSP 中，媒体用户和服务器都可发出请求，且其请求都是无状态的；在请求确认后很长时间内，仍可设置参数，控制媒体流。

重用 HTTP 功能至少在两个方面有好处，即安全和代理。要求非常接近，在缓存、代理和授权上采用 HTTP 功能是有价值的。

当大多数实时媒体使用 RTP 作为传输协议时，RTSP 没有绑定到 RTP。

RTSP 假设存在表示描述格式可表示包含几个媒体流的表示的静态与临时属性。

2 符号协定

既然很多定义和语法与 HTTP/1.1 中相同，这里仅指出它们在 HTTP/1.1 中定义的位置而并没有拷贝它们到本文档。为简便起见，本文档中[HX.Y]表示对应 HTTP/1.1（RFC 2068）中的 X.Y 部分。（[译者注：]为更方便学习 RTSP，本翻译文档将相关段落完全译出）

与[H2.1]类似，本文对所有机制的说明都是以散文和补充反馈的方式来描述的。除 RTSP 中以"1#"代替","为分隔符不同外，其余在 RFC 2234 中有详细的描述。简单说明补充反馈方式如下：

补充反馈方式（augmented BNF）包括下面的结构：

要解释的名词＝名词解释（name = definition）

规则的名字（name）就是它本身（不带任何尖括号，"<"，">"），后面跟个等号＝，

然后就是该规则的定义。如果规则需要用多个行来描述，利用空格进行缩进格式排版。某些基本的规则使用大写，如 SP, LWS, HT, CRLF, DIGIT, ALPHA,等等。定义
中还可以使用尖括号来帮助理解规则名的使用。
字面意思（"literal"）
文字的字面意思放在引号中间，除非特别指定，该段文字是大小写敏感的。
规则 1｜规则 2（rule1 | rule2）
“｜”表示其分隔的元素是可选的，比如，“是｜否”要选择‘是’或‘否’。
（规则 1 规则 2）（(rule1 rule2)）
在圆括号中的元素表明必选其一。如（元素 1（元素 2｜元素 3）元素 4）可表明两种意思，即“元素 1 元素 2 元素 4”和“元素 1 元素 3 元素 4”
*规则（*rule）
在元素前加星号“*”表示循环，其完整形式是“<n>*<m>元素”，表明元素最少产生<n>次，最多<m>次。缺省值是 0 到无限，例如，“1*元素”意思是至少有一个，而“1*2 元素”表明允许有 1 个或 2 个。
〔规则〕（[rule]）
方括号内是可选元素。如“〔元素 1 元素 2〕”与“*1（元素 1 元素 2）”是一回事。
N 规则（N rule）
表明循环的次数：“<n>（元素）”就是“<n>*<n>（元素）”，也就是精确指出<n>取值。因而，2DIGIT 就是 2 位数字，3ALPHA 就是由三个字母组成字符串。
＃规则（#rule）
“#”与“*”类似，用于定义元素列表。
完整形式是“<n>#<m>元素”表示至少有<n>个至多有<m>个元素，中间用“,”或任意数量的空格（LWS-linear whitespace）来分隔，这将使列表非常方便，如“（*LWS
元素 *（ *LWS "," *LWS 元素 ））”就等同于“1#元素”。
空元素在结构中可被任意使用，但不参与元素个数的计数。也就是说，“（元素 1),,（元素 2）”仅表示 2 个元素。但在结构中，应至少有一个非空的元素存在。缺省值是 0 到无限，即“#（元素）”表示可取任何数值，包括 0；而“1#元素”表示至少有 1 个；而“1#2 元素”表示有 1 个或 2 个。
；注释（; comment）
分号后面是注释，仅在单行使用。
隐含*LWS（implied *LWS）
本文的语法描述是基于单词的。除非另有指定，线性空格（LWS）可以两个邻近符号或分隔符（tspecials）之间任意使用，而不会对整句的意思造成影响。在两个符号之
间必须有至少一个分隔符，因为它们也要做为单独的符号来解释。实际上，应用程序在
产生 HTTP 结构时，应当试图遵照“通常方式”，因为现在的确有些实现方式在通常方式下无法正常工作。
在本备忘录中，我们用缩进的小型段落来提供动机和背景资料。这将使没有参与制定 RTSP 规范的读者更容易理解 RTSP 中各部分为什么要以该方式来实现。

3 协议参数

3.1 RTSP 版本

同[H3.1]定义，仅用 RTSP 代替 HTTP 即可。如下：

RTSP 采用主从（<major>.<minor>）数字形式来表示版本。协议的版本政策倾向于让发送方表明其消息的格式及功能，而不仅仅为了获得通讯的特性，这样做的目的是为了与更高版本的 RTSP 实现通讯。只增加扩展域的值或增加了不影响通讯行为的消息组件都不会导致版本数据的变化。当协议消息的主要解析算法没变，而消息语法及发送方的隐含功能增加了，将会导致从版本号（<minor>）增加；当协议中消息的格式变化了，主版本号（<major>）也将发生改变。

RTSP 消息的版本由消息第一行中的 RTSP 版本域来表示。RTSP-Version = "RTSP" "/" 1*DIGIT "." 1*DIGIT 注意，主从版本应当被看作单独的整数，因为它们都有可能增加，从而超过一位整数。因而，RTSP/2.4 比 RTSP/2.13 版本低，而 RTSP/2.13 又比 RTSP/12.3 版本低。版本号前面的 0 将被接收方忽略，而在发送方处也不应产生。

本文档定义了 RTSP 协议的 1.0 版本。发送本规范定义的请求（Request）或回应（Response）消息的应用必须指明 RTSP 的版本为“RTSP/1.0”。使用该版本号意味着发送消息的应用至少有条件的遵循本规范。应用的 RTSP 版本即为应用至少能有条件遵循的 RTSP 版本中的最高版本。

当代理及网关收到与其自身版本不同的 RTSP 请求时，必须小心处理请求的推送，因为协议版本表明发送方的能力，代理或网关不应发出高于自身版本的消息。如果收到高版本的请求，代理或网关必须降低该请求的版本，并回应一个错误。而低版本的请求也应在被推送前升级。代理或网关回应请求时必须和请求的版本相同。

3.2 RTSP URL

“rtsp”和“rtspu”表示要通过 RTSP 协议来定位网络资源。本节详细定义了 RTSP URL 的语法和语义。

```
rtsp_URL =  ( "rtsp:" | "rtspu:" ) "//" host [ ":" port ] [ abs_path ]
host    =  <合法的 Internet 主机域名或 IP 地址(用十进制数及点组成), 见 RFC1123，2.1 节定义>
port    =  *DIGIT
```

abs_path 在 [H3.2.1]中定义如下：

```
abs_path = "/" rel_path
rel_path = [ path ] [ ";" params ] [ "?" query ]

path = fsegment *( "/" segment )
fsegment = 1*pchar
segment = *pchar

params = param *( ";" param )
param = *( pchar | "/" )

scheme = 1*( ALPHA | DIGIT | "+" | "-" | "." )
net_loc = *( pchar | ";" | "?" )
query = *( uchar | reserved )
```

```
fragment = *( uchar | reserved )

pchar = uchar | ":" | "@" | "&" | "=" | "+"
uchar = unreserved | escape
unreserved = ALPHA | DIGIT | safe | extra | national

escape = "%" HEX HEX
reserved = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+"
extra = "!" | "*" | "'" | "(" | ")" | ","
safe = "$" | "-" | "_" | "."
unsafe = CTL | SP | <"> | "#" | "%" | "<" | ">"
national = <any OCTET excluding ALPHA, DIGIT,
reserved, extra, safe, and unsafe>
```

权威的 URL 语法及语义信息请参见 RFC1738[4]和 RFC1808[9]。
[注意]：段(fragment)和询问(query)标识符在这时没有明确的定义，需要到 RTSP 服务器上解释。
rtsp 要求使用可靠协议（Internet 的 TCP 协议）发出命令，而 rtspu 则使用不可靠协议（Internet 的 UDP 协议）。如是端口为空或没指定，则缺省为 80 端口。对于 rtsp_URI 来说，拥有被请求的资源的服务器主机通过侦听该端口的 TCP 连接(rtsp)或 UDP 包(rtspu)来接收该 URI 请求。只要可能，应尽量避免的在 URL 中直接使用 IP 地址。（请参考 RFC1924）文本媒体标识符使用 URL 中的字符集及转义规则（参考 RFC1738）来标识一个表示(presentation)与单个流(stream)。URL 可以用于单个流或者多个流的集合，比如表示(presentation)。因此，在第十章所描述的请求(request)可以用于整个表示(presentation)或表示中的单个流。注意，有些请求方法仅能用于流而不能用于表示，反之亦然。

例如：RTSP URL:
rtsp://media.example.com:554/twister/audiotrack
标识了 twister 表示(presentation)中，可以通过 media.example.com554 端口的 TCP 连接发送 RTSP 请求来控制的音频流。

也可以是这样 RTSP URL：
rtsp://media.example.com:554/twister
标识了由音频和视频流组成的 twister 表示(presentation)。

这并没有给出 URL 中相关流的标准方法。表示描述定义了表示中的层次关系以及单独流的 URL。如一个表示描述可能将一个流命名为 a.mov，而将整个表示命名为 b.mov。RTSP URL 的路径组成对客户端来说不可见并且也并没有给出服务器的具体文件系统构。
只需进行简单替换后，表示描述同样可以用于非 RTSP 媒体控制协议。

3.3 会议标识

会议标识采用 URI 标准编码方法编码，并对 RTSP 来说是不可见的。它们能包含任一八位位组值。必须保证会议标识在全局中的唯一性。在 H.323 中，将用到会议的

标识值。

```
conference-id =  1*xchar
```

会议标识允许 RTSP 会话从媒体服务器参与的多媒体会议中获取参数。比如，可以要求媒体服务器用会议描述中的标识值来代替 RTSP 客户端以提供详细的传输信息。多媒体会议的建立不属于本协议内容，具体请参见 H.323 或 SIP 协议。

3.4 会话标识

会话标识符是不可见的任意长度的字符串。线性空格必须是 URL-escaped。会话标识符必须随机产生并且至少应有 8 个八位位组长以保证其难以被猜出。（详见 16 章）

```
session-id  =  1*( ALPHA | DIGIT | safe )
```

3.5 SMPTE 相对时间戳

SMPTE 相对时间戳表示相对于开始剪辑的时间。相对时间戳以 SMPTE 时间编码形式表示而可以达到帧级量级的精度。时间编码的格式为：时：分：秒；帧.子帧，并以剪辑开始为起点。缺省的 SMPTE 格式为“SMPTE 30 drop”格式，其帧速是 29.97 帧每秒。可通过选择使用不同“SMPTE time”来选择其他 SMPTE 编码格式（如“SMPTE 25”格式）。帧域的时间值在 0 到 29 之间。30 帧每秒和 29.97 帧每秒的不同之处在于后者除每第十分钟外的每分钟都要丢掉头两个帧（00 和 01）。忽略帧值为 0 的帧，子帧以百分之一帧为单位。

```
smpte-range =  smpte-type "=" smpte-time "-" [ smpte-time ]
 smpte-type  =  "smpte" | "smpte-30-drop" | "smpte-25"
                  ; 还可以加入其他时间编码
 smpte-time  =  1*2DIGIT ":" 1*2DIGIT ":" 1*2DIGIT [ ":" 1*2DIGIT ]
            [ "." 1*2DIGIT ]
```

比如：

```
 smpte=10:12:33:20-
 smpte=10:07:33-
 smpte=10:07:00-10:07:33:05.01
 smpte-25=10:07:00-10:07:33:05.01
```

3.6 正常播放时间

正常播放时间(NPT)指出了流相对于表示(presentation)开始时的绝对位置。时间戳由一个十进制小数组成，以秒为单位，小数点左边可以直接以秒表示或者以小时：分：秒的形式表示。表示开始时对应 0.0 秒。负值没有意义。特殊的常数 now 定义为现场事件当前瞬间。它只能用于现场事件。在 DSM-CC 中，正常播放时间（NPT）是这样定义的：“直观地讲，NPT 是用户和程序联系的时钟。它经常作为数字显示在 VCR 上。当处于普通播放模式(scale = 1)时，NPT 正常前进。当处于快进扫描模式时(scale 率为大于 1 的正数)，NPT 快速前进。当处于反向扫描模式(scale 率小于 -1)时，NPT 快速后退。当处于暂停模式时，NPT 停止。NPT（逻辑上）等同于 SMPTE 时间编码。

```
npt-range  =  ( npt-time "-" [ npt-time ] ) | ( "-" npt-time )
```

```
npt-time   =  "now" | npt-sec | npt-hhmmss
npt-sec    =  1*DIGIT [ "." *DIGIT ]
npt-hhmmss  =  npt-hh ":" npt-mm ":" npt-ss [ "." *DIGIT ]
npt-hh     =  1*DIGIT   ; any positive number
npt-mm     =  1*2DIGIT  ; 0-59
npt-ss     =  1*2DIGIT  ; 0-59
```

比如：

```
npt=123.45-125
npt=12:05:35.3-
npt=now-
```

语法遵循 ISO 8601 规则。npt-sec 标志法便于自动产生， ntp-hhmmss 标志法便于人工使用。“now”常数允许客户端请求接收实时反馈而不是存储或者延时的版本。因为对于这种情况而言，既没有绝对时间，也没有 0 时间，所以需要该参数。

3.7 绝对时间

绝对时间表示为 ISO 8601 时间戳，使用 UTC(GMT)小数法表示。

```
utc-range  =  "clock" "=" utc-time "-" [ utc-time ]
utc-time   =  utc-date "T" utc-time "Z"
utc-date   =  8DIGIT          ; < YYYYMMDD >
utc-time   =  6DIGIT [ "." fraction ]  ; < HHMMSS.fraction >
```

比如，1996 年 11 月 8 日 14 点 37 分 20.25 秒 UTC 时间为：

```
19961108T143720.25Z
```

3.8 选择标签

选择标签是用来指定 RTSP 新选择的唯一标识符。这些标签用于要求(Require)(12.32 节)和代理要求(Proxy Require)(12.27 节)标题域中。

语法：

```
option-tag  =  1*xchar
```

建立新的 RTSP 选择可以通过在选择前加入相反域名的前缀（如：对于能访问到 foo.com 则 com.foo.mynewfeature" 是个合适的名字)或者在英特网权威数字分派委员会注册（IANA）新的选择。

3.8.1 用 IANA 注册新的选择标签

当注册新 RTSP 选择标签的时候，应该提供以下信息：

? 选择的名字和描述。名字长度不限，但是应该不少于 20 字符。名字不得包含任何空格，控制符或句点。

? 指出谁拥有选择的改变控制权（例如，IETF，国际标准化组织，国际电信联盟-T，其他的国际标准化体，一个团体，一个公司，或者一组公司)。

? 描述更为详细的参考文档（如果有)，比如，RFC，发表论文，专利文档，技术报告，源代码，或者计算机手册。

? 选择的所有权，以及联系地址（邮编及电子信件地址)。

4 RTSP 消息

RTSP 是基于文本的协议，采用 ISO 10646 字符集，使用 UTF-8 编码方案。行以 CRLF 中断，但接收者本身可将 CR 和 LF 解释成行终止符。基于文本的协议使以自描述方式增加可选参数更容易。由于参数的数量和命令的频率出现较低，处理效率没引起注意。如仔细研究，文本协议很容易以脚本语言（如：Tcl、Visual Basic 与 Perl）实现研究原型。

10646 字符集避免敏感字符集切换，但对应用来说不可见。RTCP 也采用这种编码方案。带有重要意义位的 ISO 8859-1 字符表示如 100001x 10xxxxxx.。RTSP 信息可通过任何低层传输协议携带。

请求包括方法、方法作用于其上的对象和进一步描述方法的参数。方法也可设计为在服务器端只需要少量或不需要状态维护。当信息体包含在信息中，信息体长度有如下因素决定：

不管实体标题域是否出现在信息中，不包括信息体的的回应信息总以标题域后第一和空行结束。

如出现内容长度标题域，其值以字节计，表示信息体长度。如未出现标题域，其值为零。

服务器关闭连接。

注意：RTSP 目前并不支持 HTTP/1.1\"块\"传输编码，需要有内容长度头。假如返回适度表示描述长度，即使动态产生，使块传输编码没有必要，服务器也应该能决定其长度。如有实体，即使必须有内容长度，且长度没显式给出，规则可确保行为合理。

从用户到服务器端的请求信息在第一行内包括源采用的方法、源标识和所用协议版本。RTSP 定义了附加状态代码，而没有定义任何 HTTP 代码。

4.1 消息类型

见[H4.1]。如下：

RTSP 消息由客户端到服务器的请求和由服务器到客户端的回应组成。

```
RTSP -message = Request | Response ; RTSP /1.0 messages
```

请求（Request）和回应（Response）消息都使用 RFC822 中实体传输部分规定（作为消息中的有效载荷）的消息格式。两者的消息都可能包括一起始行，一个或多个标题域（headers）、一行表示标题域结束的空行（即 CRLF 前没有内容的行），和一个消息主体（message-body,可选）。

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

为了健壮性考虑，服务器应该忽略任何在期望收到请求行时收到的空行。换句话说，如果服务器正在读协议流，在一个消息开始时如果首先收到了 CRLF，这个 CRLF 符应被忽略。

4.2 消息标题

见[H4.2]。
RTSP 标题域，包括主标题（General-Header,4.3 节）、请求标题（Request-Header ,5.2 节）、回应标题（Response-Header ,6.2 节）及实体标题（Entity-Header,7.1 节），都遵照 RFC822-3.1 节[7]给出的通用格式定义。每个标题域由后紧跟冒号的名字，单空格（SP），字符及域值组成。域名是大小写敏感的。虽然不提倡，标题域还是可以扩展成多行使用，只要这些行以一个以上的 SP 或 HT 开头就行。

```
RTSP-header = field-name ":" [ field-value ] CRLF
field-name = token
field-value = *( field-content | LWS )
field-content = <the OCTETs make up the field-value
and consisting of either *TEXT or combinations
of token, tspecials, and quoted-string>
```

标题域接收的顺序并不重要，但良好的习惯是，先发送主标题，然后是请求标题或回应
标题，最后是实体标题。当且仅当标题域的全部域值都用逗号分隔的列表示时（即，#（值)），多个有相同域名的 RTSP 标题域才可以表示在一个消息里。而且必须能在不改变消息语法的前提下，将并发的域值加到第一个值后面，之间用逗号分隔，最终能将多个标题域结合成“域名：域值”对。

4.3 消息主体

见[H4.3]。
RTSP 消息的消息主体（如果有）用来携带请求或回应的主体。仅在使用传输编码(Transfer-Encoding)时消息主体和实体主体才有所不同，这种情况在传输编码标题域中有详细说明。（见[H14.40]）

```
message-body = entity-body
| <entity-body encoded as per Transfer-Encoding>
```

传输编码必须能解释所有保证传输安全和正确的应用程序的传输编码。传输编码是消息而不是实体的一个属性，因此可以由任一应用程序随着请求/回应链添加或者删除。什么时候允许消息带消息体的规则在请求和回应两种情况下有所不同。在请求中有无消息主体的标志是是否包含内容长度或请求消息标题域中的传输编码标题域。只有当请求方法允许有实体主体的时候才能在请求中包含消息主体。

而对于回应消息来说，无论消息中是否存在消息主体都与请求方法和回应状态编码无关。所有回应标题请求方法的消息都不能包含消息主体，尽管有时会因为存在实体标题域而使人产生误解。所有 1××（信息），204（无内容），304（未修改）回应都不包含消息主体。而其他回应则都包含主体，尽管其长度有可能长度为零。

4.4 消息长度

当消息包含消息主体时，消息主体的长度由以下规则来决定（按优先级高低顺序排列）：
1. 任何回应消息都不包含消息主体（如 1××，204 和 304 回应），并且不管消息中是否存在实体标题域都以消息标题域后的第一行空行表示结束。

2．如果内容长度标题域存在，它在字节中的值就是消息主体的长度。如果内容标题域不存在，则假设值为零。
3．服务器关闭连接时。（关闭连接没有用来表明请求主体结束，否则可能导致服务器不能回应。
注意，RTSP 不支持（至少现在）HTTP/1.1 的块传输编码（详见[H3.6]）并且要求有内容长度标题域。尽管表示描述长度动态产生，但由于可获得了表示描述返回长度，使得服务器总是能决定表示描述长度而不需使用块传输编码方式。只要有实体主体就必须有内容长度项，这些规则保证了即使没有给出明确长度也能做出合理的操作。

5 普通标题域

有几种标题域是请求与回应都要使用的，但并不用于被传输的实体。这些标题只用于被
传输的消息。

```
General-Header = Date ; Section 10.6
| Pragma ; Section 10.12
```

普通标题域名称只有在与协议版本的变化结合起来后，才能进行可靠的扩展。实际上，新的或实验中的标题域只要能被通讯各方识别，其语法就可使用，而无法识别的标题域都将被视为实体域。

6 请求

从客户端到服务器端的请求消息包括，消息首行中，对资源的请求方法、资源的标识符
及使用的协议。

```
Request = Request-Line ; 6.1 节
*( general-header ; 5 章
| request-header ; 6.2 节
| entity-header ) ; 8.1 节
CRLF
[ message-body ] ; 4.3 节
```

6.1 请求队列

```
Request-Line = Method SP Request-URI SP RTSP-Version CRLF
Method = "DESCRIBE" ; Section 10.2
| "ANNOUNCE" ; Section 10.3
| "GET_PARAMETER" ; Section 10.8
| "OPTIONS" ; Section 10.1
| "PAUSE" ; Section 10.6
| "PLAY" ; Section 10.5
| "RECORD" ; Section 10.11
| "REDIRECT" ; Section 10.10
| "SETUP" ; Section 10.4
```

```
| "SET_PARAMETER" ; Section 10.9
| "TEARDOWN" ; Section 10.7
| extension-method
extension-method = token
Request-URI = "*" | absolute_URI
RTSP-Version = "RTSP" "/" 1*DIGIT "." 1*DIGIT
```

6.2 请求标题域

```
request-header = Accept ; Section 12.1
| Accept-Encoding ; Section 12.2
| Accept-Language ; Section 12.3
| Authorization ; Section 12.5
| From ; Section 12.20
| If-Modified-Since ; Section 12.23
| Range ; Section 12.29
| Referer ; Section 12.30
| User-Agent ; Section 12.41
```

注意：相对于 HTTP/1.1 而言，RTSP 请求要求绝对路径（并包括 rtsp 或 rtspu 方案，主机，端口号）。HTTP/1.1 要求服务器理解绝对 URL，但是客户端需要假设为主机请求标题域。这样做完全是为了 HTTP/1.0 服务器端向后兼容性，因此 RTSP 并不需要这样做。在请求 URI 中星号“*”表示此请求不用于其他资源，只用于服务器本身，并且它只能在使用的方法不要求应用于资源时才能使用。比如：OPTIONS * RTSP/1.0。

7 回应

7.1 状态行

完整回应消息的第一行就是状态行，它依次由协议版本、数字形式的状态代码、及相应
的词语文本组成，各元素间以空格（SP）分隔，除了结尾的 CRLF 外，不允许出现单独的
CR 或 LF 符。

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

7.1.1 状态代码和原因分析

状态代码（Status-Code）由 3 位数字组成，表示请求是否被理解或被满足。原因分析是
用简短的文字来描述状态代码产生的原因。状态代码用来支持自动操作，原因分析是为人类用户准备的。客户端不需要检查或显示原因分析。

状态代码的第一位数字定义了回应的类别，后面两位数字没有具体分类。首位数字有 5 种取值可能：

o 1xx:：保留，将来使用。

o 2xx：成功 — 操作被接收、理解、接受（received, understood, accepted）。
o 3xx：重定向（Redirection）— 要完成请求必须进行进一步操作。
o 4xx：客户端出错 — 请求有语法错误或无法实现。
o 5xx：服务器端出错 — 服务器无法实现合法的请求。
HTTP/1.0 的状态代码、原因解释在下面给出。下面的原因解释只是建议采用，可任意
更改，而不会对协议造成影响。完整的代码定义在第 9 节。

```
Status-Code = "100" ; Continue
| "200" ; OK
| "201" ; Created
| "250" ; Low on Storage Space
| "300" ; Multiple Choices
| "301" ; Moved Permanently
| "302" ; Moved Temporarily
| "303" ; See Other
| "304" ; Not Modified
| "305" ; Use Proxy
| "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 Time-out
| "410" ; Gone
| "411" ; Length Required
| "412" ; Precondition Failed
| "413" ; Request Entity Too Large
| "414" ; Request-URI Too Large
| "415" ; Unsupported Media Type
| "451" ; Parameter Not Understood
| "452" ; Conference Not Found
| "453" ; Not Enough Bandwidth
| "454" ; Session Not Found
| "455" ; Method Not Valid in This State
| "456" ; Header Field Not Valid for Resource
| "457" ; Invalid Range
| "458" ; Parameter Is Read-Only
| "459" ; Aggregate operation not allowed
| "460" ; Only aggregate operation allowed
| "461" ; Unsupported transport
```

```
| "462" ; Destination unreachable
| "500" ; Internal Server Error
| "501" ; Not Implemented
| "502" ; Bad Gateway
| "503" ; Service Unavailable
| "504" ; Gateway Time-out
| "505" ; RTSP Version not supported
| "551" ; Option not supported
| extension-code
extension-code = 3DIGIT
Reason-Phrase = *<TEXT, excluding CR, LF>
```

HTTP 状态代码是可扩展的，而只有上述代码才可以被当前全部的应用所识别。HTTP 应用不要求了解全部注册的状态代码，当然，如果了解了更好。实际上，应用程序必须理解任何一种状态代码，如果碰到不识别的情况，可根据其首位数字来判断其类型并处理。另外，不要缓存无法识别的回应。

例如，如果客户端收到一个无法识别的状态码 431，可以安全地假定是请求出了问题，可认为回应的状态码就是 400。在这种情况下，用户代理应当在回应消息的实体中通知用户，因为实体中可以包括一些人类可以识别的非正常状态的描述信息。

```
Code reason
100 Continue all
200 OK all
201 Created RECORD
250 Low on Storage Space RECORD
300 Multiple Choices all
301 Moved Permanently all
302 Moved Temporarily all
303 See Other all
305 Use Proxy all
400 Bad Request all
401 Unauthorized all
402 Payment Required all
403 Forbidden all
404 Not Found all
405 Method Not Allowed all
406 Not Acceptable all
407 Proxy Authentication Required all
408 Request Timeout all
410 Gone all
411 Length Required all
412 Precondition Failed DESCRIBE, SETUP
413 Request Entity Too Large all
```

414 Request-URI Too Long all
415 Unsupported Media Type all
451 Invalid parameter SETUP
452 Illegal Conference Identifier SETUP
453 Not Enough Bandwidth SETUP
454 Session Not Found all
455 Method Not Valid In This State all
456 Header Field Not Valid all
457 Invalid Range PLAY
458 Parameter Is Read-Only SET_PARAMETER
459 Aggregate Operation Not Allowed all
460 Only Aggregate Operation Allowed all
461 Unsupported Transport all
462 Destination Unreachable all
500 Internal Server Error all
501 Not Implemented all
502 Bad Gateway all
503 Service Unavailable all
504 Gateway Timeout all
505 RTSP Version Not Supported all
551 Option not support all
Table 1: Status codes and their usage with RTSP methods

7.1.2 回应标题域

回应标题域中包括不能放在状态行中的附加回应信息。该域还可以存放与服务器相关的
信息，以及在对请求 URI 所指定资源进行访问的下一步信息。

```
response-header = Location ; Section 12.25
| Proxy-Authenticate ; Section 12.26
| Public ; Section 12.28
| Retry-After ; Section 12.31
| Server ; Section 12.36
| Vary ; Section 12.42
| WWW-Authenticate ; Section 12.44
```

回应标题域名只有在与协议版本的变化结合起来后，才能进行可靠的扩展。实际上，新
的或实验中的标题域只要能被通讯各方识别，其语法就可使用，而无法识别的标题域都将被视为实体域。

8 实体

如不受请求方法或回应状态编码限制，请求和回应消息可传输实体，实体由实体

标题域和实体主体组成，有些回应仅包括实体头。在此，根据谁发送实体、谁接收实体，发送者和接收者可分别指用户和服务器。

8.1 实体标题域

实体标题定义实体主体可选元信息，如没有实体主体，指请求标识的资源。

```
entity-header = Allow ; Section 12.4
| Content-Base ; Section 12.11
| Content-Encoding ; Section 12.12
| Content-Language ; Section 12.13
| Content-Length ; Section 12.14
| Content-Location ; Section 12.15
| Content-Type ; Section 12.16
| Expires ; Section 12.19
| Last-Modified ; Section 12.24
| extension-header
extension-header = message-header
```

扩展头机制允许定义附加实体标题域，而不用改变协议，但这些段不能假定接收者能识别。不可识别标题域应被接收者忽略，而让代理转发。

8.2 实体主体见[H7.2]

与 RTSP 请求或回应一起发送的实体主体的格式和编码信息都在实体标题域（Entity-Header）中定义。

```
Entity-Body = *OCTET
```

实体主体只在请求方法有要求时才会被放在请求消息中。请求消息标题域处的内容长度
标题域（Content-Length header field）的标志将指明请求中的实体主体是否存在。包含实体主体的 RTSP/1.0 请求必须包含合法的内容长度标题域。

对回应消息来说，消息中是否包含实体主体取决于请求方法和回应代码。所有的 HEAD 请求方法的回应都不应包括主体，即便是实体标题域中指明有主体也一样。在主体中不应包括这些回应信息，全部 1xx（信息）、204（无内容）和 304（未修改）。而其它的回应必须包括实体主体或其内容长度标题（Content-Length header）域的定义值为 0。

9 连接

RTSP 请求可以几种不同方式传送：

1、持久传输连接，用于多个请求/回应传输。

2、每个请求/回应传输一个连接。

3、无连接模式。

传输连接类型由 RTSP URI 来定义。对 \"rtsp\" 方案，需要持续连接；而 \"rtspu\"方案，调用 RTSP 请求发送，而不用建立连接。

不象 HTTP，RTSP 允许媒体服务器给媒体用户发送请求。然而，这仅在持久连接时才支持，否则媒体服务器没有可靠途径到达用户，这也是请求通过防火墙从媒体服务器传到用户的唯一途径。

9.1 流水线操作

支持持久连接或无连接的客户端可能给其请求排队。服务器必须以收到请求的同样顺序发出回应。

9.2 可靠性及确认

如果请求不是发送给组播组，接收者就确认请求，如没有确认信息，发送者可在超过一个来回时间（RTT）后重发同一信息。 RTT 在 TCP 中估计，初始值为 500 ms。应用缓存最后所测量的 RTT，作为将来连接的初始值。如使用一个可靠传输协议传输 RTSP，请求不允许重发，RTSP 应用反过来依赖低层传输提供可靠性。如两个低层可靠传输（如 TCP 和 RTSP）应用重发请求，有可能每个包损失导致两次重传。由于传输栈在第一次尝试到达接收着者前不会发送应用层重传，接收者也不能充分利用应用层重传。如包损失由阻塞引起，不同层的重发将使阻塞进一步恶化。时标头用来避免重发模糊性问题，避免对圆锥算法的依赖。每个请求在 CSeq 头中携带一个系列号，每发送一个不同请求，它就加一。如由于没有确认而重发请求，请求必须携带初始系列号。

实现 RTSP 的系统必须支持通过 TCP 传输 RTSP ，并支持 UDP。对 UDP 和 TCP，RTSP 服务器的缺省端口都是 554。许多目的一致的 RTSP 包被打包成单个低层 PDU 或 TCP 流。RTSP 数据可与 RTP 和 RTCP 包交叉。不象 HTTP，RTSP 信息必须包含一个内容长度头，无论信息何时包含负载。否则，RTSP 包以空行结束，后跟最后一个信息头。

10 方法定义（method token）

表示了对请求统一资源标志符（Request-URI）识别的资源所执行的操作。方法名区分大小写。将来可能定义新的方法。方法名可能不以美元符'$'（十进制数 24）开头，但必须具有表征意义(must be a token)。

表格 2 是对方法的一个小结。

method	direction	object	requirement
DESCRIBE	C -> S	P,S	recommended
ANNOUNCE	C -> S,S ->C	P,S	optional
GET PARAMETER	C -> S,S ->C	P,S	optional
OPTIONS	C -> S,S ->C	P,S	required(S ! C: optional)
PAUSE	C -> S	P,S	recommended
PLAY	C -> S	P,S	required
RECORD	C -> S	P,S	optional
REDIRECT	S ->C	P,S	optional
SETUP	C -> S	S	required
SET PARAMETER	C -> S,S ->C	P,S	optional

TEARDOWN C -> S P,S required

表 2：对 RTSP 方法，和其操作方向及所操作对象（P: 表示, S: 媒体流）的一个概览

注意：PAUSE 方法是推荐的，但在构建一个全功能的服务器（fully functional server）
时可能不支持此方法，这时就不需要它，比如对于 live feeds。如果服务器不支持某个特殊
方法，它必将返回"501 Not Implemented"，并且客户端应该不再向该服务器请求该方法。
（注：Presentation 是一个以完整的 media feed 呈现给 client 的一个或多个媒体流的集合，
暂且翻译成表示）

10.1 OPTIONS

其行为与[H9.2]中描述的等同。OPTIONS 请求可能在任何时候发出，例如客户端将要发
出一个非标准的请求时。它不影响服务器状态。
示例：

```
C->S: OPTIONS * RTSP/1.0
CSeq: 1
Require: implicit-play
Proxy-Require: gzipped-messages

S->C: RTSP/1.0 200 OK
CSeq: 1
Public: DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE
```

注意：这些都是必要的构造特征（necessarily fictional features）。(你可能不希望我
们去有意忽略那些实际上有用的特征，因此在这一部分中我们将给出一个详细的例子)。

10.2 DESCRIBE

DESCRIBE 方法从服务器检索表示的描述或媒体对象，这些资源通过请求统一资源定位符（the request URL）识别。此方法可能结合使用 Accept 首部域来指定客户端理解的描述格式。服务器端用被请求资源的描述对客户端作出响应。DESCRIBE 的答复-响应对（reply-response pair）组成了 RTSP 的媒体初始化阶段。
示例：

```
C->S: DESCRIBE rtsp://server.example.com/fizzle/foo RTSP/1.0
CSeq: 312
Accept: application/sdp, application/rtsl, application/mheg

S->C: RTSP/1.0 200 OK
```

```
CSeq: 312
Date: 23 Jan 1997 15:35:06 GMT
Content-Type: application/sdp
Content-Length: 376

v=0
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
e=mjh@isi.edu (Mark Handley)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 3456 RTP/AVP 0
m=video 2232 RTP/AVP 31
m=whiteboard 32416 UDP WB
a=orient:portrait
```

DESCRIBE 响应必须包含它所描述资源的所有媒体初始化信息。如果媒体客户端从一个数据源获得表示描述，而非通过 DESCRIBE，并且该描述包含了一个媒体初始化参数的全集，那么客户端就应该使用这些参数，而不是再通过 RTSP 请求相同媒体的描述。

再有，服务器不应该（SHOULD NOT）使用 DESCRIBE 响应作为 media indirection 的方法。

需要建立基本的规则，使得客户端有明确的方法了解何时通过 DESCRIBE 请求媒体初始化信息，何时不请求。强制 DESCRIBE 响应包含它所描述媒体流集合的所有初始化信息，不鼓励将 DESCRIBE 用作 media indirection 的方法，通过这两点避免了使用其他方法可能会引起的循环问题（looping problems）

媒体初始化是任何基于 RTSP 系统的必要条件，但 RTSP 规范并没有规定它必须通过 DESCRIBE 方法完成。RTSP 客户端可以通过 3 种方法来接收媒体初始化信息：

. DESCRIBE 方法；

.其它一些协议（HTTP，email 附件，等）；

.命令行或标准输入（同一个 SDP 或其它媒体初始化格式的文件一起启动，工作方式类似于浏览器的帮助程序）。

为了实际协同工作，严重（）推荐最精简的服务器也支持 DESCRIBE 方法，最精简的客户端也支持从标准输入，命令行和/或其它对于客户端操作环境合适的方法来接收媒体初始化文件的能力。

10.3 ANNOUNCE

ANNOUNCE 方法有两个用途：

当客户端向服务器发送时，ANNOUNCE 将通过请求 URL 识别的表示描述或者媒体对象提交给服务器；

当服务器相客户端发送时，ANNOUNCE 实时更新会话描述。

如果有新的媒体流加到表示中（比如在一个现场表示中），整个表示描述应该重发；而不只是增加组件，如果这样做的话，组件也可以被删除了。

示例：

```
C->S: ANNOUNCE rtsp://server.example.com/fizzle/foo RTSP/1.0
CSeq: 312
Date: 23 Jan 1997 15:35:06 GMT
Session: 47112344
Content-Type: application/sdp
Content-Length: 332

v=0
o=mhandley 2890844526 2890845468 IN IP4 126.16.64.4
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
e=mjh@isi.edu (Mark Handley)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 3456 RTP/AVP 0
m=video 2232 RTP/AVP 31

S->C: RTSP/1.0 200 OK
CSeq: 312
```

10.4 SETUP

SETUP 请求为 URI 指定流式媒体的传输机制。客户端能够发出一个 SETUP 请求为正在播放的媒体流改变传输参数，服务器可能同意这些参数的改变。若是不同意，它必须响应错误"455 Method Not Valid In This State"。 为了尽量绕开防火墙干涉，即使它不会影响参数，客户端也必须指出传输参数，例如，指出服务器向外发布的固定的广播地址。

由于 SETUP 包括了所有传输初始化信息，防火墙和其他中间的网络设备（它们需要这些信息）分让了解析 DESCRIBE 响应的繁琐任务，这些任务留给了媒体初始化。

Transport 首部域指定了客户端数据传输时可接受的传输参数；响应包含了由服务器选出的传输参数。

```
C->S: SETUP rtsp://example.com/foo/bar/baz.rm RTSP/1.0
CSeq: 302
Transport: RTP/AVP;unicast;client_port=4588-4589

S->C: RTSP/1.0 200 OK
CSeq: 302
Date: 23 Jan 1997 15:35:06 GMT
Session: 47112344
```

```
Transport: RTP/AVP;unicast;
client_port=4588-4589;server_port=6256-6257
```

作为对 SETUP 请求的响应，服务器产生了会话标志符。如果对服务器的请求中包含了会话标志符，服务器必须将此 setup 请求捆绑到一个存在的会话，或者返回"459 Aggregate Operation Not Allowed"。

10.5 PLAY

PLAY 方法告知服务器通过 SETUP 中指定的机制开始发送数据 。在尚未收到 SETUP 请求的成功应答之前，客户端不可以发出 PLAY 请求。PLAY 请求将正常播放时间（normal play time）定位到指定范围的起始处，并且传输数据流直到播放范围结束。PLAY 请求可能被管道化（pipelined），即放入队列中（queued）；服务器必须将 PLAY 请求放到队列中有序执行。也就是说，后一个 PLAY 请求需要等待前一个 PLAY 请求完成才能得到执行。

比如，在下例中，不管到达的两个 PLAY 请求之间有多紧凑，服务器首先 play 第 10 到 15 秒，然后立即第 20 到 25 秒，最后是第 30 秒直到结束。

```
C->S: PLAY rtsp://audio.example.com/audio RTSP/1.0
CSeq: 835
Session: 12345678
Range: npt=10-15

C->S: PLAY rtsp://audio.example.com/audio RTSP/1.0
CSeq: 836
Session: 12345678
Range: npt=20-25

C->S: PLAY rtsp://audio.example.com/audio RTSP/1.0
CSeq: 837
Session: 12345678
Range: npt=30-
```

结合 PAUSE 请求的描述，看更深一层的示例。

不含 Range 首部域的 PLAY 请求也是合法的。它从媒体流开头开始播放，直到媒体流被暂停。如果媒体流通过 PAUSE 暂停，媒体流传输将在暂停点（the pause point）重新开始。

如果媒体流正在播放，那么这样一个 PLAY 请求将不起更多的作用，只是客户端可以用此来测试服务器是否存活。

Range 首部域可能包含一个时间参数。该参数以 UTC 格式指定了播放（palayback）开始的时间。如果在这个指定时间后收到消息，那么播放立即开始。时间参数可能用来帮助同步从不同数据源获取的数据流。

对于一个点播（On-demand）媒体流，服务器用播放（play back）的实际范围答复请求。This
may differ from the requested range if alignment of the requested range to valid frame boundaries is required for the media source.

如果在请求中没有指定范围，当前位置将在答复中返回。答复中播放范围的单位与请求中相同。在播放完被要求的范围后，表示将自动暂停，就好像发出了一个 PAUSE 请求。

下面的示例在 play 整个表示时从 SMPTE 时间 0:10:20 直到剪辑（clip）结束。播放开始于 1997 年 1 月 23 号，15 点 36 分

```
C->S: PLAY rtsp://audio.example.com/twister.en RTSP/1.0
CSeq: 833
Session: 12345678
Range: smpte=0:10:20-;time=19970123T153600Z

S->C: RTSP/1.0 200 OK
CSeq: 833
Date: 23 Jan 1997 15:35:06 GMT
Range: smpte=0:10:22-;time=19970123T153600Z
For playing back a recording of a live presentation, it may be
desirable to use clock
units:

C->S: PLAY rtsp://audio.example.com/meeting.en RTSP/1.0
CSeq: 835
Session: 12345678
Range: clock=19961108T142300Z-19961108T143520Z

S->C: RTSP/1.0 200 OK
CSeq: 835
Date: 23 Jan 1997 15:35:06 GMT
```

只有播放的媒体服务器必须支持 npt 时间格式，可能支持 clock 和 smpte 格式。

10.6 PAUSE

PAUSE 请求引起媒体流传输的暂时中断。如果请求 URL 中指定了具体的媒体流，那么只有该媒体流的播放和记录被暂停（halt）。比如，指定暂停音频，播放将会无声。如果请求 URL 指定了一个表示或者媒体流已成组，那么在该表示或组中的所有当前活动流的传输将被暂停。在重启播放或记录后，必须维护不同媒体轨迹（track）的同步。尽管服务器可能在暂停后，在 timeout 的时间内关闭会话，释放资源，但是任何资源都必须保存，其中 timeout 参数位于 SETUP 消息的会话头中。

示例：

```
C->S: PAUSE rtsp://example.com/fizzle/foo RTSP/1.0
CSeq: 834
Session: 12345678

S->C: RTSP/1.0 200 OK
CSeq: 834
Date: 23 Jan 1997 15:35:06 GMT
```

PAUSE 请求中可能包含一个 Range 首部域用来指定何时媒体流或表示暂停，我们称这个时刻为暂停点（pause point）。该首部域必须包含一个精确的值，而不是一个时间范围。媒体流的正常播放时间设置成暂停点。当服务器遇到在任何当前挂起（pending）的 PLAY 请求中指定的时间点后，暂停请求生效。如果 Range 首部域指定了一个时间超出了任何一个当前挂起的 PLAY 请求，将返回错误"457 Invalid Range" 。如果一个媒体单元（比如一个音频或视频禎）正好在一个暂停点开始，那么表示将不会被播放或记录。如果 Range 首部域缺失，那么在收到暂停消息后媒体流传输立即中断，并且暂停点设置成当前正常播放时间。

利用 PAUSE 请求可忽视所有排队的 PLAY 请求，但必须维护媒体流中的暂停点。不带 Range 首部域的后继 PLAY 请求从暂停点重启播放。

比如，如果服务器有两个挂起的播放请求，播放范围（range）分别是 10 到 15 和 20 到 29，这时收到一个暂停请求，暂停点是 NPT21，那么它将会开始播放第二个范围，并且在 NPT21 处停止。如果服务器正在服务第一个请求播放到 NPT13 位置，收到暂停请求，暂停点 NPT12，那么它将立即停止。如果请求在 NPT16 暂停，那么服务器在完成第一个播放请求后停止，放弃了第二个播放请求。

再如，服务器收到播放请求，播放范围从 10 到 15 和 13 到 20（即之间有重叠），PAUSE 暂停点是 NPT14，则当服务器播放第一段范围时，PAUSE 请求将生效，而第二个 PLAY 请求会被忽略重叠部分，就好像服务器在开始播放第二段前收到 PAUSE 请求。不管 PAUSE 请求何时到达，它总是设置 NPT 到 14。

如果服务器已经在 Range 首部域指定的时间外发送了数据，后继的 PLAY 仍会在暂停点及时重启，因为它认为客户端会丢弃在暂停点后收到的数据。这就确保了连续、无隙的暂停/播放循环。

10.7 TEARDOWN

TEARDOWN 请求终止了给定 URI 的媒体流传输，并释放了与该媒体流相关的资源。如果该 URI 是对此表示的表示 URI，那么任何与此会话相关的任何 RTSP 会话标志符将不再有效。除非所有传输参数由会话描述符定义，否则 SETUP 请求必须在会话能被再次播放之前发出。

示例：

```
C->S: TEARDOWN rtsp://example.com/fizzle/foo RTSP/1.0
CSeq: 892
Session: 12345678

S->C: RTSP/1.0 200 OK
CSeq: 892
```

10.8 GET PARAMETER

GET PARAMETER 请求检索 URI 指定的表示或媒体流的参数值。答复和响应的内容留给了实现。不带实体主体的 GET PARAMETER 可用来测试客户端或服务器是否存活（"Ping"）。

示例：

```
S->C: GET_PARAMETER rtsp://example.com/fizzle/foo RTSP/1.0
```

```
CSeq: 431
Content-Type: text/parameters
Session: 12345678
Content-Length: 15
packets_received
jitter

C->S: RTSP/1.0 200 OK
CSeq: 431
Content-Length: 46
Content-Type: text/parameters
packets_received: 10
jitter: 0.3838
```

"text/parameters"段只是参数类型的一个例子。对此方法有意的进行了松散的定义，对于答复和响应的内容将在更深一层的实验中给出定义。

10.9 SET PARAMETER

此方法给 URI 指定的表示或媒体流设置参数值。

帮助客户端检查某个特殊的请求为何失败的请求（晕~）应该只附带一个参数。当请求附带
多个参数时，服务器只有在这些参数全都设置正确时才作出响应。服务器必须允许某个参数被
重复设置成相同的值，但可能不允许改变参数值。
注意：必须只能使用 SETUP 命令来给媒体流设置传输参数。 限制只有 SETUP 能设置传输参数有利于防火墙设计。
示例：

```
C->S: SET_PARAMETER rtsp://example.com/fizzle/foo RTSP/1.0
CSeq: 421
Content-length: 20
Content-type: text/parameters
barparam: barstuff

S->C: RTSP/1.0 451 Invalid Parameter
CSeq: 421
Content-length: 10
Content-type: text/parameters
barparam
```

"text/parameters"段只是参数类型的一个例子。对此方法有意的进行了松散的定义，对于答复和响应的内容将在更深一层的实验中给出定义。

10.10 REDIRECT

REDIRECT 请求告知客户端连接到另一个服务器位置。它包含首部域 Location，

该域指出了客户端应该发出请求的 URL。它可能包含参数 Range，在重定向生效时，该域指明了媒体流的范围。如果客户端希望继续发送或接收其 URI 指定的媒体，它必须发出一个 TEARDOWN 请求来关闭当前会话，并向委派的主机发送 SETUP 以建立新的会话。

本例中，在给定的播放时间将 URI 请求重定向到新的服务器：

```
S->C: REDIRECT rtsp://example.com/fizzle/foo RTSP/1.0
CSeq: 732
Location: rtsp://bigserver.com:8001
Range: clock=19960213T143205Z-
```

10.11 RECORD

此方法根据表示描述开始记录媒体数据。时间戳（timestamp）表现了起始和结束时间（UTC）。

如果没有给定时间范围，就使用表示描述中提供的开始和结束时间。如果会话已经开启，立即开始记录。由服务器来决定是否存储记录的数据到请求 URI 下或者其它 URI 下。如果服务器没有使用请求 URI，那么响应代码应该是 201（创建），并且包含一个实体，该实体描述了请求的状态，并通过 Location 首部域指向新资源。

允许记录现场表示（live presentations）的媒体服务器必须支持时钟范围格式（the clock
range format），smpte 格式对此无用。

在本示例中，媒体服务器被邀请到指定的会议

```
C->S: RECORD rtsp://example.com/meeting/audio.en RTSP/1.0
CSeq: 954
Session: 12345678
Conference: 128.16.64.19/32492374
```

10.12 Embedded (Interleaved) Binary Data

可能某些防火墙设计和环境会强制服务器交叉 RTSP 方法和媒体流数据。这种交叉增加了客户端和服务器操作的复杂性，带来了额外的开销，因此通常情况下应该避免；除非必须交叉。只有 RTSP 在 TCP 上运载时，交叉的二进制数据才能使用。

媒体流数据，如 RTP 包，被封装成下列形式：ASCII 的美元符（十进制数 24），一个字节的通道标志符（channel identifier），被封装的二进制数据的长度，以网络字节顺序编码的 2 字节
整数。紧接着的是上层的协议头。每个＄块都正确地包含了一个上层协议数据单元，比如一个 RTP 包。

通道标志符使用交叉参数定义在传输头。

当使用实时传输协议传输时，RTP 和 RTCP 消息也会在 TCP 连接上相互交叉。默认情况下，RTCP 包会在第一个可用的通道上发送，高于 RTP 通道。而客户端可能在另一个通道显式地请求 RTCP 包。在传输头的交叉参数中指定两个通道可解决此问题。

```
C->S: SETUP rtsp://foo.com/bar.file RTSP/1.0
CSeq: 2
Transport: RTP/AVP/TCP;interleaved=0-1

S->C: RTSP/1.0 200 OK
```

```
CSeq: 2
Date: 05 Jun 1997 18:57:18 GMT
Transport: RTP/AVP/TCP;interleaved=0-1
Session: 12345678

C->S: PLAY rtsp://foo.com/bar.file RTSP/1.0
CSeq: 3
Session: 12345678

S->C: RTSP/1.0 200 OK
CSeq: 3
Session: 12345678
Date: 05 Jun 1997 18:59:15 GMT
RTP-Info: url=rtsp://foo.com/bar.file;
seq=232433;rtptime=972948234
S->C: $\000{2 byte length}{"length" bytes data, w/RTP header}
S->C: $\000{2 byte length}{"length" bytes data, w/RTP header}
S->C: $\001{2 byte length}{"length" bytes RTCP packet}
```