

# RTMP协议中文

blog:	<a href="https://github.com/adwpc">https://github.com/adwpc</a>
ver	0.2
info	参考官方文档和网上翻译整理

## 1. Introduction(简介)

Adobe 公司的实时消息传输协议 (RTMP) 通过一个可靠地流传输提供了一个双向多通道消息服务，比如 TCP [RFC0793]，意图在通信端之间传递带有时间信息的视频、音频和数据消息流。实现通常对不同类型的消息分配不同的优先级，当运载能力有限时，这会影响等待流传输的消息的次序。

本文档将对实时流传输协议 (Real Time Messaging Protocol) 的语法和操作进行描述。

### 1.1. Terminology(术语)

本文档中出现的关键字，"MUST"、"MUST NOT"、"REQUIRED"、"SHALL"、"SHALL NOT"、"SHOULD"、"SHOULD NOT"、"RECOMMENDED"、"NOT RECOMMENDED"、"MAY"、"OPTIONAL"，都将在 [RFC2119] 中进行解释。

## 2. Contributors(贡献者)

Rajesh Mallipeddi，Adobe Systems 原成员，起草了本文档原始规范，并提供大部分的原始内容。

Mohit Srivastava，Adobe Systems 成员，促成了本规范的开发。

## 3. Definitions(名词解释)

Payload (有效载荷)：

包含于一个数据包中的数据，例如音频采样或者压缩的视频数据。payload 的格式和解释，超出了本文档的范围。

**Packet (数据包) :**

一个数据包由一个固定头和有效载荷数据构成。一些个底层协议可能会要求对数据包定义封装。

**Port (端口) :**

"传输协议用以区分开指定一台主机的不同目的地的一个抽象。TCP/IP 使用小的正整数对端口进行标识。" OSI 传输层使用的运输选择器 (TSEL) 相当于端口。

**Transport address (传输地址) :**

用以识别传输层端点的网络地址和端口的组合，例如一个 IP 地址和一个 TCP 端口。  
数据包由一个源传输地址传送到一个目的传输地址。

**Message stream (消息流) :**

通信中消息流通的一个逻辑通道。

**Message stream ID (消息流 ID) :**

每个消息有一个关联的 ID，使用 ID 可以识别出流通中的消息流。

**Chunk (块) :**

消息的一段。消息在网络发送之前被拆分成很多小的部分。块可以确保端到端交付所有消息有序 timestamp，即使有很多不同的流。

**Chunk stream (块流) :**

通信中允许块流向一个特定方向的逻辑通道。块流可以从客户端流向服务器，也可以从服务器流向客户端。

**Chunk stream ID (块流 ID) :**

每个块有一个关联的 ID，使用 ID 可以识别出流通中的块流。

**Multiplexing (合成) :**

将独立的音频/视频数据合成为一个连续的音频/视频流的加工，这样可以同时发送几个视频和音频。

**DeMultiplexing (分解) :**

Multiplexing 的逆向处理，将交叉的音频和视频数据还原成原始音频和视频数据的格式。

Remote Procedure Call (RPC 远程方法调用) :

允许客户端或服务器调用对端的一个子程序或者程序的请求。

Metadata (元数据) :

关于数据的一个描述。一个电影的 metadata 包括电影标题、持续时间、创建时间等等。

Application Instance (应用实例) :

服务器上应用的实例，客户端可以连接这个实例并发送连接请求。

Action Message Format (AMF 动作消息格式协议) :

一个用于序列化 ActionScript 对象图的紧凑的二进制格式。AMF 有两个版本：AMF 0 [AMF0] 和 AMF 3 [AMF3]。

## 4. Byte Order, Alignment, and Time Format(字节序、对齐和时间格式)

所有整数型属性以网络字节顺序传输，字节 0 代表第一个字节，零位是一个单词或字段最常用的有效位。字节序通常是大端排序。关于传输顺序的更多细节描述参考 IP 协议 [RFC0791]。除非另外注明，本文档中的数值常量都是十进制的 (以 10 为基础)。

除非另有规定，RTMP 中的所有数据都是字节对准的；例如，一个十六位的属性可能会在一个奇字节偏移上。填充后，填充字节应该有零值。

RTMP 中的 Timestamps 以一个整数形式给出，表示一个未指明的时间点。典型地，每个流会以一个为 0 的 timestamp 起始，但这不是必须的，只要双端能够就时间点达成一致。注意这意味着任意不同流 (尤其是来自不同主机的) 的同步需要 RTMP 之外的机制。

因为 timestamp 的长度为 32 位，每隔 49 天 17 小时 2 分钟和 47.296 秒就要重来一次。因为允许流连续传输，有可能要多年，RTMP 应用在处理 timestamp 时应该使用序列码算法 [RFC1982]，并且能够处理无限循环。例如，一个应用假定所有相邻的 timestamp 都在  $2^{31} - 1$  毫秒之内，因此 10000 在 4000000000 之后，而 3000000000 在 4000000000 之前。

timestamp 也可以使用无符整数定义，相对于前面的 timestamp。timestamp 的长度可能会是 24 位或者 32 位。

## 5. RTMP Chunk Stream(RTMP 块流)

本节介绍实时消息传输协议的块流 (RTMP 块流)。它为上层多媒体流协议提供合并和打包的服务。

当设计 RTMP 块流使用实时消息传输协议时，它可以处理任何发送消息流的协议。每个消息包含 timestamp 和 payload 类型标识。RTMP 块流和 RTMP 一起适合各种音频-视频应用，从一对一和一对多直播到点播服务，到互动会议应用。

当使用可靠传输协议时，比如 TCP [RFC0793]，RTMP 块流能够对于多流提供所有消息可靠的 timestamp 有序端对端传输。RTMP 块流并不提供任何优先权或类似形式的控制，但是可以被上层协议用来提供这种优先级。例如，一个直播视频服务器可能会基于发送时间或者每个消息的确认时间丢弃一个传输缓慢的客户端的视频消息以确保及时获取其音频消息。

RTMP 块流包括其自身的带内协议控制信息，并且提供机制为上层协议植入用户控制消息。

### 5.1 Message Format(消息格式)

可以被分割为块以支持组合的消息的格式取决于上层协议。消息格式必须包含以下创建块所需的字段。

Timestamp :

消息的 timestamp。这个字段可以传输四个字节。

Length :

消息的有效负载长度。如果不能省略掉消息头，那它也被包括进这个长度。这个字段占用了块头的三个字节。

Type Id :

一些类型 ID 保留给协议控制消息使用。这些传播信息的消息由 RTMP 块流协议和上层协议共同处理。其他的所有类型 ID 可用于上层协议，它们被 RTMP 块流处理为不透明值。事实上，RTMP 块流中没有任何地方要把这些值当做类型使用；所有消息必须是同一类型，或者应用使用这一字段来区分同步跟踪，而不是类型。这一字段占用了块头的一个字节。

Message Stream ID :

消息流ID可以是任意值。合并到同一个块流的不同的消息流是根据各自的消息流 ID 进行分解。除此之外，对 RTMP 块流而言，这是一个不透明的值。这个字段以小端格式占用了块头的四个字节。

## 5.2 Handshake(握手)

一个 RTMP 连接以握手开始。RTMP 的握手不同于其他协议；RTMP 握手由三个固定长度的块组成，而不是像其他协议一样的带有报头的可变长度的块。

客户端 (发起连接请求的终端) 和服务器端各自发送相同的三块。便于演示，当发送自客户端时这些块被指定为 C0、C1 和 C2；当发送自服务器端时这些块分别被指定为 S0、S1 和 S2。

### 5.2.1. 握手顺序

握手以客户端发送 C0 和 C1 块开始。

客户端必须等待接收到 S1 才能发送 C2。

客户端必须等待接收到 S2 才能发送任何其他数据。

服务器端必须等待接收到 C0 才能发送 S0 和 S1，也可以等待接收到 C1 再发送 S0 和 S1。服务器端必须等待接收到 C1 才能发送 S2。服务器端必须等待接收到 C2 才能发送任何其他数据。

### 5.2.2. C0 和 S0 的格式

C0 和 S0 包都是一个单一的八位字节，以一个单独的八位整型域进行处理：

0	1	2	3	4	5	6	7
+-----+-----+-----+							
	version						
+-----+-----+-----+							

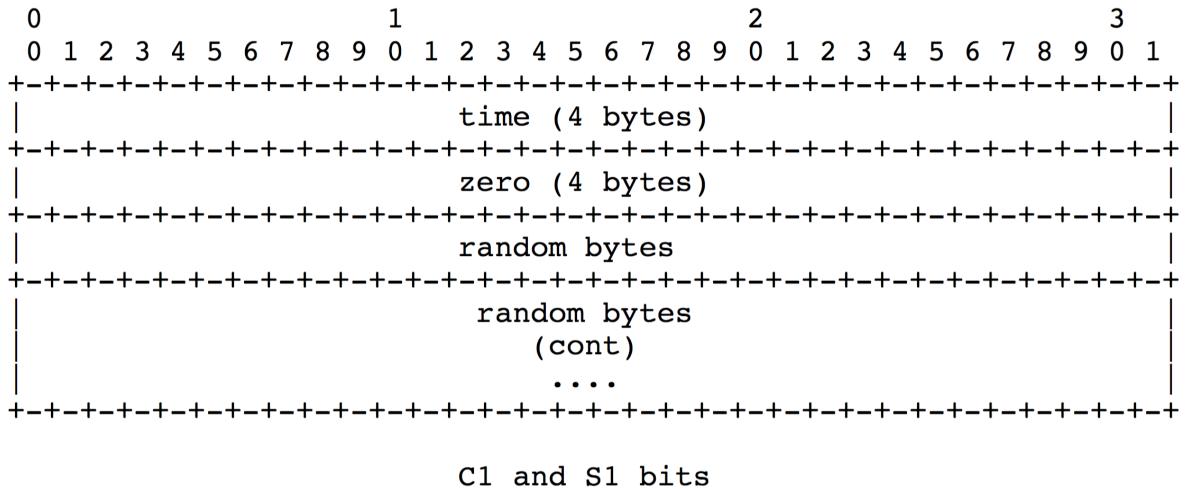
C0 and S0 bits

以下是 C0/S0 包中的字段：

版本号 (八位)：在 C0 中，这一字段指示出客户端要求的 RTMP 版本号。在 S0 中，这一字段指示出服务器端选择的 RTMP 版本号。本文档中规范的版本号为 3。0、1、2 三个值是由早期其他产品使用的，是废弃值；4 - 31 被保留为 RTMP 协议的未来实现版本使用；32 - 255 不允许使用 (以区分开 RTMP 和其他常以一个可打印字符开始的文本协议)。无法识别客户端所请求版本号的服务器应该以版本 3 响应，(收到响应的) 客户端可以选择降低到版本 3，或者放弃握手。

### 5.2.3. C1 和 S1 的格式

C1 和 S1 数据包的长度都是 1536 字节，包含以下字段：



Time (四个字节)：

这个字段包含一个 timestamp，用于本终端发送的所有后续块的时间起点。这个值可以是 0，或者一些任意值。要同步多个块流，终端可以发送其他块流当前的 timestamp 的值。

Zero (四个字节)：

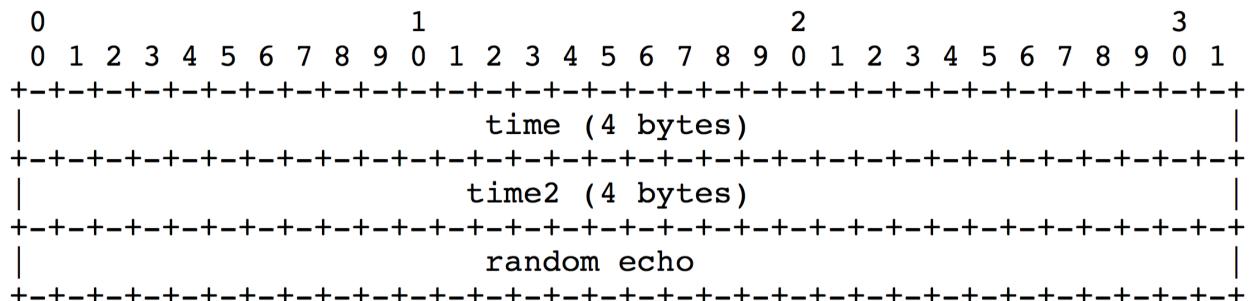
这个字段必须都是 0。

Random data (1528 个字节)：

这个字段可以包含任意值。终端需要区分出响应来自它发起的握手还是对端发起的握手，这个数据应该发送一些足够随机的数。这个不需要对随机数进行加密保护，也不需要动态值。

### 5.2.4. C2 和 S2 的格式

C2 和 S2 数据包长度都是 1536 字节，基本就是 S1 和 C1 的副本 (分别)，包含有以下字段：



Time (四个字节) :

这个字段必须包含终端在 S1 (给 C2) 或者 C1 (给 S2) 发的 timestamp。

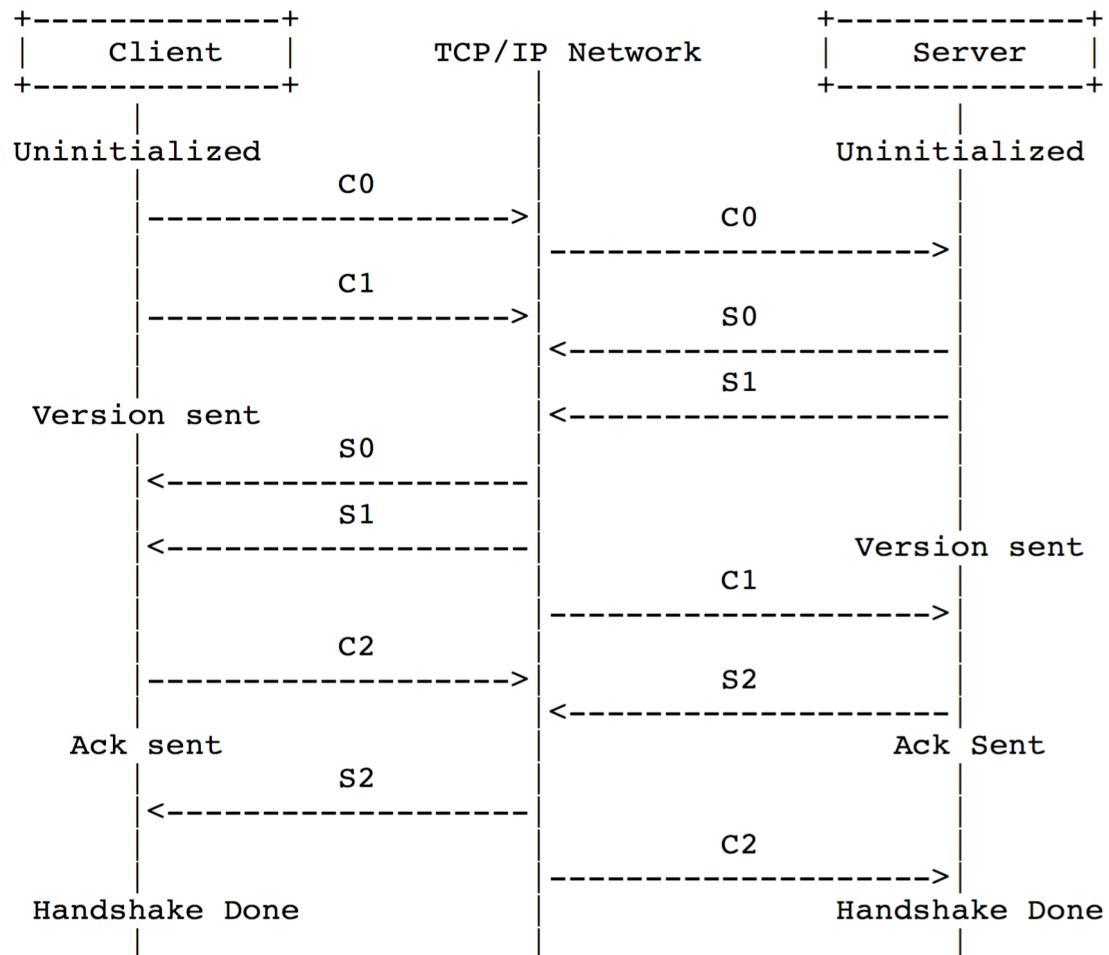
Time2 (四个字节) :

这个字段必须包含终端先前发出数据包 (s1 或者 c1) timestamp。

Random echo (1528 个字节) :

这个字段必须包含终端发的 S1 (给 C2) 或者 S2 (给 C1) 的随机数。两端都可以一起使用 time 和 time2 字段再加当前 timestamp 以快速估算带宽和/或者连接延迟，但这不太可能是有多大用处。

### 5.2.5. 握手示意图



下面描述了握手示意图中提到的状态：

**Uninitialized (未初始化) :**

协议的版本号在这个阶段被发送。客户端和服务器都是 uninitialized (未初始化) 状态。之后客户端在数据包 C0 中将协议版本号发出。如果服务器支持这个版本，它将在回应中发送 S0 和 S1。如果不支持呢，服务器会才去适当的行为进行响应。在 RTMP 协议中，这个行为就是终止连接。

**Version Sent (版本已发送) :**

在未初始化状态之后，客户端和服务器都进入 Version Sent (版本已发送) 状态。客户端会等待接收数据包 S1 而服务器在等待 C1。一旦拿到期待的包，客户端会发送数据包 C2 而服务器发送数据包 S2。(客户端和服务器各自的)状态随即变为 Ack Sent (确认已发送)。

**Ack Sent (确认已发送) :**

客户端和服务器分别等待 S2 和 C2。

**Handshake Done (握手结束) :**

客户端和服务器可以开始交换消息了。

### 5.3. Chunking(分块)

握手之后，连接开始对一个或多个块流进行合并。创建的每个块都有一个唯一 ID 对其进行关联，叫做 chunk stream ID (块流标识，CSID)。这些块通过网络进行传输。传递时，每个块必须被完全发送才可以发送下一块。在接收端，这些块被根据CSID 被组装成消息。

分块允许上层协议将大的消息分解为更小的消息，例如，防止体积大的但优先级小的消息 (比如视频) 阻碍体积较小但优先级高的消息 (比如音频或者控制命令)。

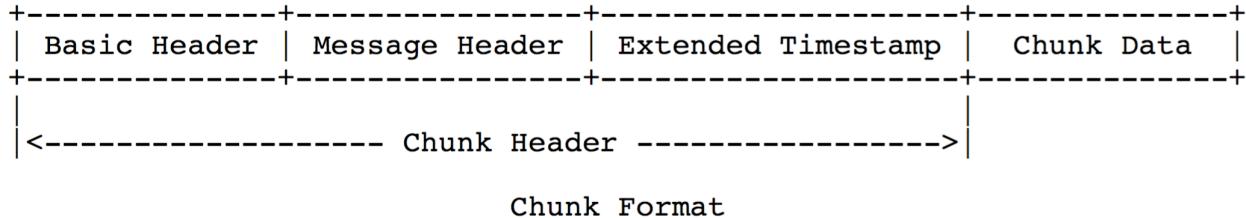
分块也让我们能够使用较小开销发送小消息，因为块头包含在消息内部的信息压缩提示。

块的大小是可以配置的。它可以使用一个设置块大小的控制消息进行设置 (参考 5.4.1)。更大的块大小可以降低 CPU 开销，但在低带宽连接时因为它的大量的写入也会延迟

其他内容的传递。更小的块不利于高比特率的流化。所以块的大小设置取决于具体情况。

### 5.3.1. Chunk Format(块格式)

每个块包含一个头和数据体。块头包含三个部分：



Basic Header (基本头，1 到 3 个字节)：

这个字段对CSID和fmt(块类型)进行编码。fmt(块类型)决定了message header(消息头)的编码格式。(这一字段的) 长度完全取决于CSID，因为CSID是一个可变长度的字段。

Message Header (消息头，0 , 3 , 7 , 或者 11 个字节)：

这一字段对正在发送的消息(不管是完整消息，还是只是一小部分)的信息进行编码。这一字段的长度可以使用basic header中定义的fmt(块类型)进行决定。

Extended Timestamp (扩展 timestamp , 0 或 4 字节)：

这一字段是否出现取决于message header(块消息头)中的 timestamp 或者 timestamp delta 字段。更多信息参考 5.3.1.3 节。

Chunk Data (有效大小)：

当前块的payload(有效负载)，相当于定义的最大块大小。

#### 5.3.1.1. Chunk Basic Header(块基本头)

basic header对CSID和fmt进行编码。basic header字段可能会有 1 , 2 或者 3 个字节，取决于CSID。

一个(RTMP) 实现应该使用能够容纳这个 ID 的最小的容量进行表示。

RTMP 协议最多支持 65597 个流，CSID范围 3 - 65599。0、1、2 被保留。

この値は、データの種類と、データの構造を示すためのマスクです。

1 値(0-5bit)表示三字节形式，并且 ID 范围为 64 - 65599 ((第三个字节) \* 256 + 第二个字节 + 64)。3 - 63 范围内的值表示整个流 ID。

2 值(0-5bit)的块流 ID 被保留，用于下层协议控制消息和命令。

CSID 2 - 63 可以编进这一字段的一字节版本中。块基本头中的 0 - 5 位 (最低位字节) 代表CSID。

0	1	2	3	4	5	6	7
+---+---+---+---+---+---+							
fmt		cs		id			
+---+---+---+---+---+---+							

Chunk basic header 1

CSID 64 - 319 可以以二字节的形式编码在头中。计算为 (第二个字节 + 64) :

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+															
fmt		0				cs		id		- 64					
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+															

Chunk basic header 2

CSID 64 - 65599 可以编码在这个字段的三字节版本中。计算为 ((第三个字节) \* 256 + (第二个字节) + 64)。

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+																							
fmt		1				cs		id		- 64													
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+																							

Chunk basic header 3

CSID(六位) :

这一字段包含有块流标识，值的范围是 2 - 63。0 和 1 用于指示这一字段是 2 或者 3 字节版本。

fmt (两位) :

这一字段指定 chunk message header(块消息头) 的结构格式。每种结构的

△ 块消息头 chunk message header (块消息头) 以单字节形式表示。单字节块消息头会在下一小节解释。

### CSID - 64 (8 或者 16 位) :

这一字段包含了CSID 减掉 64 后的值。例如，365 会以一个 1 进行表示，和这里的一个 16 位的 301 (CSID - 64)。

CSID 64 - 319 可以使用 2-byte 或者 3-byte 的形式在头中表示。

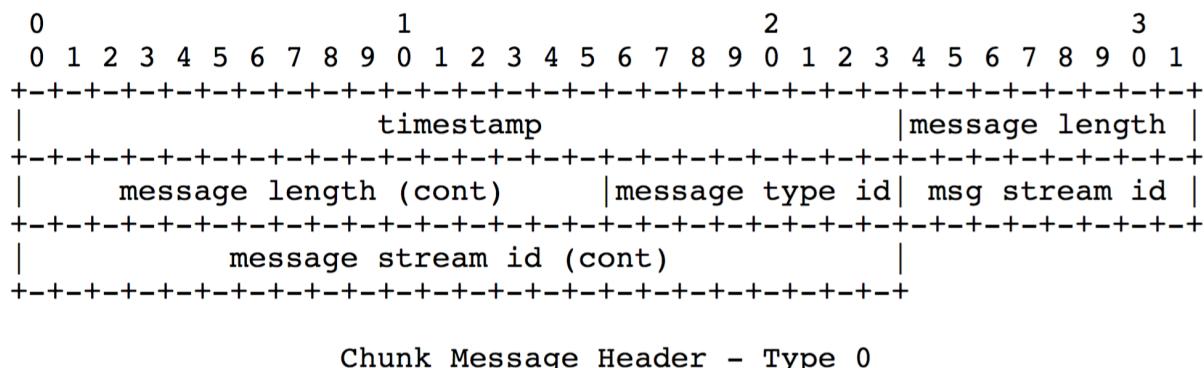
#### 5.3.1.2. Chunk Message Header(块消息头)

块消息头有四种不同的格式，由 "fmt" 字段进行选择。

每个块消息头使用最紧凑表示方法。

##### 5.3.1.2.1. 类型 0

类型 0 块头的长度是 11 个字节。这一类型必须用在块流的起始位置，和流 timestamp 重来的时候 (比如，重置)。

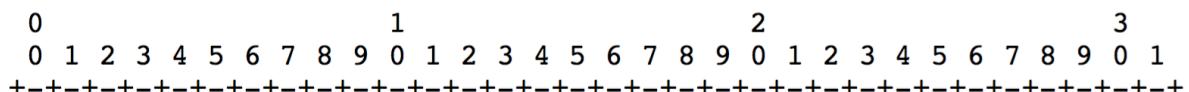


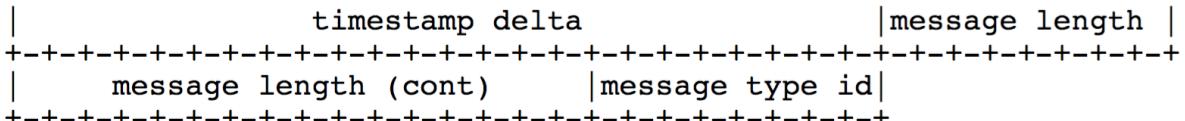
##### timestamp (三个字节) :

对于 type-0 块，当前消息的绝对 timestamp 在这里发送。如果 timestamp 大于或者等于 16777215 (十六进制 0xFFFFFFF)，这一字段必须是 16777215，表明有扩展 timestamp 字段来补充完整的 32 位 timestamp。否则的话，这一字段必须是整个的 timestamp。

##### 5.3.1.2.2. 类型 1

类型 1 块头长为 7 个字节。不包含消息流 ID；这一块使用前一块一样的流 ID。可变长度消息的流 (例如，一些视频格式) 应该在第一块之后使用这一格式表示之后的每个新消息。

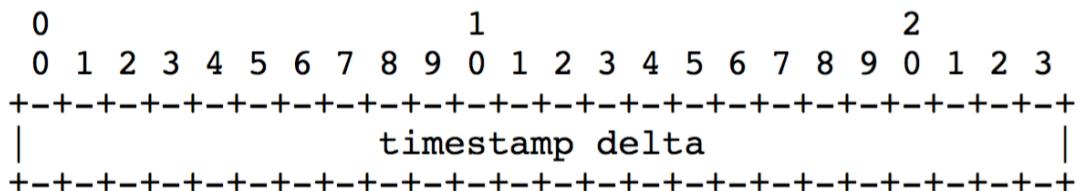




Chunk Message Header - Type 1

### 5.3.1.2.3. 类型 2

类型 2 块头长度为 3 个字节。既不包含流 ID 也不包含消息长度；这一块具有和前一块相同的流 ID 和消息长度。具有不变长度的消息（例如，一些音频和数据格式）应该在第一块之后使用这一格式表示之后的每个新消息。



Chunk Message Header - Type 2

### 5.3.1.2.4. 类型 3

类型 3 的块没有消息头。流 ID、消息长度以及 timestamp delta 等字段都不存在；这种类型的块使用前面块一样的 CSID。当单一个消息被分割为多块时，除了第一块的其他块都应该使用这种类型。参考例 2（5.3.2.2 小节）。组成流的消息具有同样的大小，流 ID 和时间间隔应该在类型 2 之后的所有块都使用这一类型。参考例 1（5.3.2.1 小节）。如果第一个消息和第二个消息之间的 delta 和第一个消息的 timestamp 一样的话，那么在类型 0 的块之后要紧跟一个类型 3 的块，因为无需再来一个类型 2 的块来注册 delta 了。如果一个类型 3 的块跟着一个类型 0 的块，那么这个类型 3 块的 timestamp delta 和类型 0 块的 timestamp 是一样的。

### 5.3.1.2.5. 通用头字段

块消息头中各字段的描述如下：

**timestamp delta**（三个字节）：

对于一个类型 1 或者类型 2 的块，前一块的 timestamp 和当前块的 timestamp 的区别在这里发送。如果 delta 大于或者等于 16777215（十六进制 0xFFFFF），那么这一字段必须是为 16777215，表示具有扩展 timestamp 字段来对整个 32 位 delta 进行编码。否则的话，这一字段应该是为具体 delta。

**message length**（三个字节）：

对于一个类型 0 或者类型 1 的块，消息长度在这里进行发送。注意这通常不同于块的有效载荷的长度。块的有效载荷代表所有的除了最后一块的最大块大小，以及剩余的

(也可能是小消息的整个长度) 最后一块。

message type id (消息类型 id , 一个字节) :

对于类型 0 或者类型 1 的块 , 消息的类型在这里发送。

message stream id (四个字节) :

对于一个类型为 0 的块 , 保存消息流 ID。消息流 ID 以小端格式保存。所有同一个块流下的消息都来自同一个消息流。当可以将不同的消息流组合进同一个块流时 , 这种方法比头压缩的做法要好。但是 , 当一个消息流被关闭而其他的随后另一个是打开着的 , 就没有理由将现有块流以发送一个新的类型 0 的块进行复用了。

### 5.3.1.3. 扩展 timestamp

扩展 timestamp 字段用于对大于 16777215 (0xFFFFFFF) 的 timestamp 或者 timestamp delta 进行编码 ; 也就是 , 对于不适合于在 24 位的类型 0、1 和 2 的块里的 timestamp 和 timestamp delta 编码。这一字段包含了整个 32 位的 timestamp 或者 timestamp delta 编码。可以通过设置类型 0 块的 timestamp 字段、类型 1 或者 2 块的 timestamp delta 字段 16777215 (0xFFFFFFF) 来启用这一字段。当最近的具有同一块流的类型 0、1 或 2 块指示扩展 timestamp 字段出现时 , 这一字段才会在类型为 3 的块中出现。

## 5.3.2. 例子

### 5.3.2.1. 例子 1

这个例子演示了一个简单地音频消息流。这个例子演示了信息的冗余。

	Message Stream ID	Message Type ID	Time	Length
Msg # 1	12345	8	1000	32
Msg # 2	12345	8	1020	32
Msg # 3	12345	8	1040	32
Msg # 4	12345	8	1060	32

Sample audio messages to be made into chunks

下一个表格演示了这个流所产生的块。从消息 3 起 , 数据传输得到了最佳化利用。每条消息的开销在这一点之后都只有一个字节。

Chunk Stream ID	Chunk Type	Header Data	No.of Bytes after	Total No.of Bytes in the

				Header	Chunk
Chunk#1	3	0	delta: 1000 length: 32, type: 8, stream ID: 12345 (11 bytes)	32	44
Chunk#2	3	2	20 (3 bytes)	32	36
Chunk#3	3	3	none (0 bytes)	32	33
Chunk#4	3	3	none (0 bytes)	32	33

Format of each of the chunks of audio messages

### 5.3.2.2. 例子 2

这一例子阐述了一条消息太大，无法装在一个 128 字节的块里，被分割为若干块。

	Message Stream ID	Message Type ID	Time	Length
Msg # 1	12346	9 (video)	1000	307

Sample Message to be broken to chunks

这是传输的块：

	Chunk Stream ID	Chunk Type	Header Data	No. of Bytes after Header	Total No. of bytes in the chunk
Chunk#1	4	0	delta: 1000 length: 307 type: 9, stream ID: 12346 (11 bytes)	128	140
Chunk#2	4	3	none (0 bytes)	128	129
Chunk#3	4	3	none (0 bytes)	51	52

Format of each of the chunks

块 1 的数据头说明了整个消息长度是为 30 / 1 个字节。

由以上俩例子可以得知，块类型 3 可以被用于两种不同的方式。第一种是用于定义一条消息的配置。第二种是定义一个可以从现有状态数据中派生出来的消息的起点。

## 5.4. 协议控制消息

RTMP 块流使用消息类型 ID 为 1、2、3、5 和 6 用于协议控制消息。这些消息包含有 RTMP 块流协议所需要的信息。

这些协议控制消息必须使用消息流 ID 0 (作为已知控制流) 并以流 ID 为 2 的块发送。协议控制消息一旦被接收到就立即生效；协议控制消息的 timestamp 被忽略。

### 5.4.1. 设置块类型 (1)

协议控制消息 1，设置块大小，以通知对端一个新的最大块大小。

默认的最大块大小是为 128 字节，但是客户端或者服务器可以改变这个大小，并使用这一消息对对端进行更新。例如，假定一个客户端想要发送一个 131 字节的音频数据，当前块大小是默认的 128。在这种情况下，客户端可以发送这种消息到服务器以通知它块大小现在是 131 字节了。这样客户端就可以在单一块中发送整个音频数据了。

最大块大小设置的话最少为 128 字节，包含内容最少要一个字节。最大块大小由每个方面 (服务器或者客户端) 自行维护。

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1			
+-----+			
0	chunk size (31 bits)		
+-----+			

Payload for the 'Set Chunk Size' protocol message

0：这个位必须为 0。

chunk size (块大小，31 位)：

这一字段保存新的最大块大小值，以字节为单位，这将用于之后发送者发送的块，直到有更多 (关于最大块大小的) 通知。有效值为 1 到 2147483647 (0x7FFFFFFF，1 和 2147483647 都可取)；但是所有大于 16777215 (0xFFFFFFF) 的大小值是等价的，因为没有一个块比一整个消息大，并且没有一个消息大于 16777215 字节。

### 5.4.2. 终止消息 (2)

协议控制消息 2，终止消息，用于通知对端，如果对端在等待去完成一个消息的块的话，然后抛弃一个块流中已接受到的部分消息。对端接收到块流 ID 作为当前协议消息的有效负载。一些程序可能会在关闭的时候使用这个消息以指示不需要进一步对这个消息的处理

了。

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
+---+	+---+	+---+	+---+
	chunk stream id (32 bits)		
+---+	+---+	+---+	+---+

Payload for the 'Abort Message' protocol message

chunk stream ID (块流 ID , 32 位) : 这一字段保存块流 ID , 该流的当前消息会被丢弃。

#### 5.4.3. 确认 (3)

客户端或者服务器在接收到等同于窗口大小的字节之后必须要发送给对端一个确认。窗口大小是指发送者在没有收到接收者确认之前发送的最大数量的字节。这个消息定义了序列号 , 也就是目前接收到的字节数。

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
+---+	+---+	+---+	+---+
	sequence number (4 bytes)		
+---+	+---+	+---+	+---+

Payload for the 'Acknowledgement' protocol message

sequence number (序列号 , 32 位) : 这一字段保存有目前接收到的字节数。

#### 5.4.4. 窗口确认大小 (5)

客户端或者服务器端发送这条消息来通知对端发送和应答之间的窗口大小。发送者在发送完窗口大小字节之后期待对端的确认。接收端在上次确认发送后接收到的指示数值后 , 或者会话建立之后尚未发送确认 , 必须发送一个确认 (5.4.3 小节)。

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
+---+	+---+	+---+	+---+
	Acknowledgement Window size (4 bytes)		
+---+	+---+	+---+	+---+

Payload for the 'Window Acknowledgement Size' protocol message

#### 5.4.5. 设置对端带宽 (6)

客户端或者服务器端发送这一消息来限制其对端的输出带宽。对端接收到这一消息后 , 将通过限制这一消息中窗口大小指出的已发送但未被答复的数据的数量以限制其输出带宽。接收到这一消息的对端应该回复一个窗口确认大小消息 , 如果这个窗口大小不同于其发送给 (设置对端带宽) 发送者的最后一条消息。

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
Acknowledgement Window size			
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
Limit Type			
+---+---+---+---+---+			

Payload for the 'Set Peer Bandwidth' protocol message

限制类型取以下值之一：

0 - Hard :

对端应该限制其输出带宽窗口大小。

1 - Soft :

对端应该限制其输出带宽窗口大小，或者已经有限制的话就取两者之间的较小值。

2 - Dynamic :

如果先前的限制类型为 Hard，处理这个消息就好像它被标记为 Hard，否则的话忽略这个消息。

## 6. RTMP Message Format(RTMP 消息格式)

这一节定义了使用下层传输层(比如 RTMP 块流协议)传输的 RTMP 消息的格式。

RTMP 协议设计使用 RTMP 块流，可以使用其他任意传输协议对消息进行发送。RTMP 块流和 RTMP 一起适用于多种音频 - 视频应用，从一对一和一对多直播到点播服务，再到互动会议应用。

### 6.1. RTMP 消息格式

服务器端和客户端通过网络发送 RTMP 消息来进行彼此通信。消息可以包含音频、视频、数据，或者其他消息。

RTMP 消息有两部分：header(头)和payload(有效载荷)。

#### 6.1.1. 消息头

消息头包含以下：

Message Type (消息类型)：

一个字节的字段来表示消息类型。类型 ID 1 - 6 被保留用于协议控制消息。

Length (长度) :

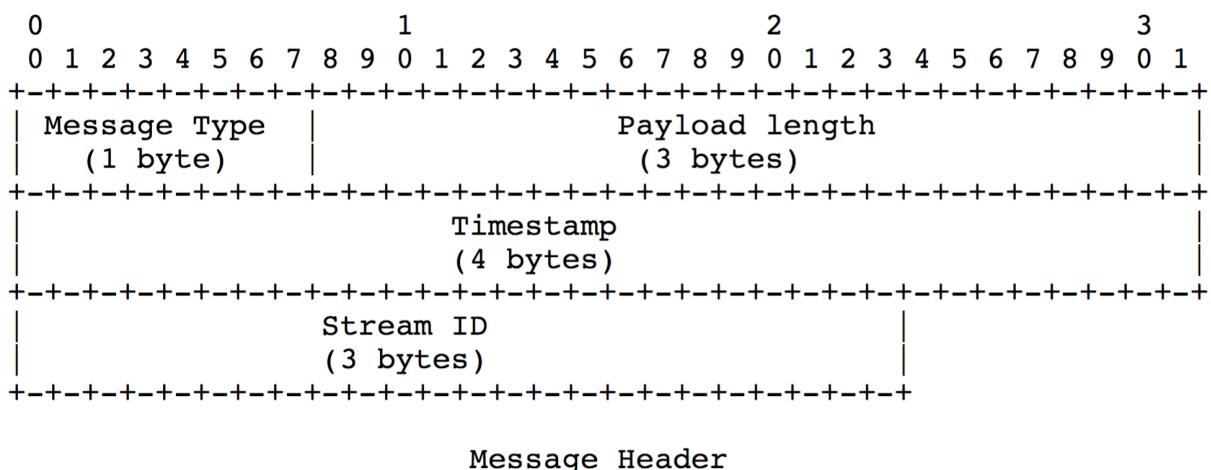
三个字节的字段来表示有效负载的字节数。以大端格式保存。

Timestamp :

四个字节的字段包含了当前消息的 timestamp。四个字节也以大端格式保存。

Message Stream Id (消息流 ID) :

三个字节的字段以指示出当前消息的流。这三个字节以大端格式保存。



### 6.1.2. 消息有效载荷

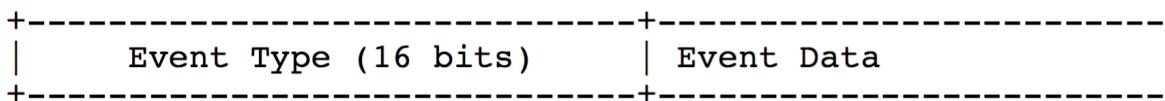
消息的另一个部分就是有效负载，这是这个消息所包含的实际内容。例如，它可以是一些音频样本或者压缩的视频数据。有效载荷格式和解释不在本文档范围之内。

## 6.2. 用户控制消息 (4)

RTMP 使用消息类型 ID 4 表示用户控制消息。这些消息包含 RTMP 流传输层所使用的信息。RTMP 块流协议使用 ID 为 1、2、3、5 和 6 (5.4 节介绍)。

用户控制消息应该使用消息流 ID 0 (以被认为是控制流)，并且以 RTMP 块流发送时以块流 ID 为 2。用户控制消息一旦被接收立马生效；它们的 timestamp 是被忽略的。

客户端或者服务器端发送这个消息来通知对端用户操作事件。这一消息携带有事件类型和事件数据。



## Payload for the ‘User Control’ protocol message

消息数据的前两个字节用于指示事件类型。事件类型被事件数据紧随。事件数据字段的大小是可变的。但是，如果消息必须通过 RTMP 块流层传输时，最大块大小 (5.4.1 节) 应该足够大以允许这些消息填充在一个单一块中。

事件类型和事件数据格式将在 7.1.7 小节列出。

# 7. RTMP Command Messages(RTMP 命令消息)

这一节描述了在服务器端和客户端彼此通信交换的消息和命令的不同的类型。

服务器端和客户端交换的不同消息类型包括用于发送音频数据的音频消息、用于发送视频数据的视频消息、用于发送任意用户数据的数据消息、共享对象消息以及命令消息。共享对象消息提供了一个通用的方法来管理多用户和一台服务器之间的分布式数据。命令消息在客户端和服务器端传输 AMF 编码的命令。客户端或者服务器端可以通过使用命令消息和对端通信的流请求远程方法调用 (RPC)。

## 7.1. 消息的类型

服务器端和客户端通过在网络中发送消息来进行彼此通信。消息可以是任何类型，包含音频消息，视频消息，命令消息，共享对象消息，数据消息，以及用户控制消息。

### 7.1.1. 命令消息 (20, 17)

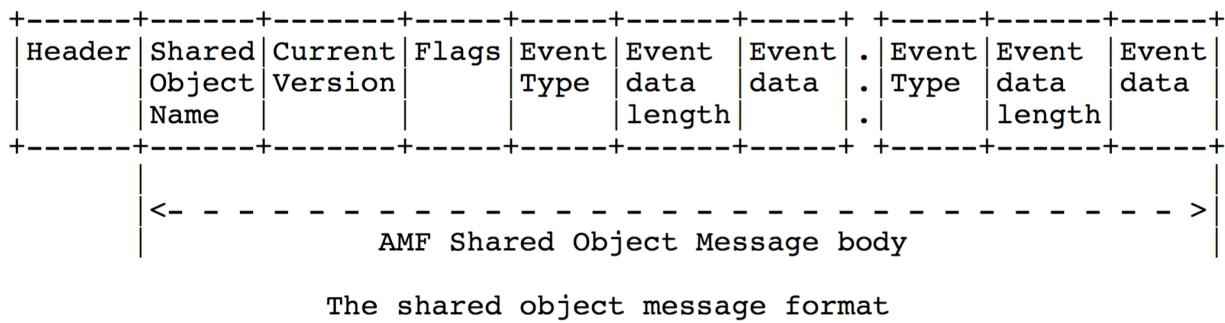
命令消息在客户端和服务器端传递 AMF 编码的命令。这些消息被分配以消息类型值为 20 以进行 AMF0 编码，消息类型值为 17 以进行 AMF3 编码。这些消息发送以进行一些操作，比如，连接，创建流，发布，播放，对端暂停。命令消息，像 onstatus、result 等等，用于通知发送者请求的命令的状态。一个命令消息由命令名、事务 ID 和包含相关参数的命令对象组成。一个客户端或者一个服务器端可以通过和对端通信的流使用这些命令消息请求远程调用 (RPC)。

### 7.1.2. 数据消息 (18, 15)

客户端或者服务器端通过发送这些消息以发送元数据或者任何用户数据到对端。元数据包括数据 (音频，视频等等) 的详细信息，比如创建时间，时长，主题等等。这些消息被分配以消息类型为 18 以进行 AMF0 编码和消息类型 15 以进行 AMF3 编码。

### 7.1.3. 共享对象消息 (19, 16)

所谓共享对象其实是一个 Flash 对象（一个名值对的集合），这个对象在多个不同客户端、应用实例中保持同步。消息类型 19 用于 AMF0 编码、16 用于 AMF3 编码都被为共享对象事件保留。每个消息可以包含有不同事件。



支持以下事件类型：

事件	描述
Use(=1)	客户端发送这一事件以通知服务器端一个已命名的共享对象已创建。
Release(=2)	当共享对象在客户端被删除时客户端发送这一事件到服务器端。
Request Change (=3)	客户端发送给服务器端这一事件以请求共享对象的已命名的参数所关联到的值的改变。
Change (=4)	服务器端发送这一事件已通知发起这一请求之外的所有客户端，一个已命名参数的值的改变。
Success (=5)	如果请求被接受，服务器端发送这一事件给请求的客户端，以作为 RequestChange 事件的响应。
SendMessage (=6)	客户端发送这一事件到服务器端以广播一条消息。一旦接收到来这一事件，服务器端将会给所有的客户端广播这一消息，包括这一消息的发起者。
Status (=7)	服务器端发送这一事件以通知客户端异常情况。
Clear (=8)	服务器端发送这一消息到客户端以清理一个共享对象。服务器端也会对客户端发送的 Use 事件使用这一事件进行响应。
Remove (=9)	服务器端发送这一事件有客户端删除一个 slot。

Request Remove (=10)	客户端发送这一事件有客户端删除一个 slot。
Use Success (=11)	服务器端发送给客户端这一事件表示连接成功。

#### 7.1.4. 音频消息 (8)

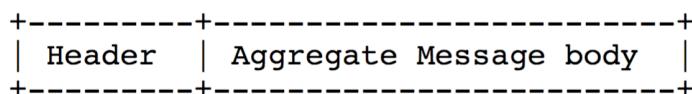
客户端或者服务器端发送这一消息以发送音频数据到对端。消息类型 8 为音频消息保留。

#### 7.1.5. 视频消息 (9)

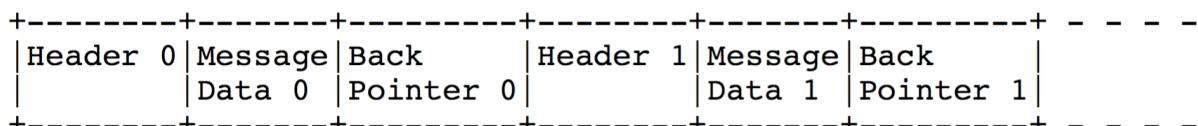
客户端或者服务器发送这一消息以发送视频数据到对端。消息类型 9 为视频消息保留。

#### 7.1.6. 聚合消息 (22)

聚合消息是一个单一的包含一系列的使用 6.1 节描述的 RTMP 子消息的消息。消息类型 22 用于统计消息。



The Aggregate Message format



The Aggregate Message body format

聚合消息的消息流 ID 覆盖了统计中子消息的消息流 ID。

聚合消息里的 timestamp 和第一个子消息的 timestamp 的不同点在于子消息的 timestamp 被相对流时间标调整了偏移。每个子消息的 timestamp 被加入偏移以达到一个统一时间。第一个子消息的 timestamp 应该和统计消息的 timestamp 一样，所以这个偏移量应该为 0。

反向指针包含有前一个消息的大小 (包含前一个消息的头)。这样子匹配了 FLV 文件的格式，用于反向查找。

使用聚会消息具有以下性能优势：

- 子消息可以在内存中连续存储。在网络中系统调用发送这些数据时更高效。
- 块流可以在一个块中以至多一个单一完整的消息发送。因此，增加块大小并减少了发

### 7.1.7. 用户控制消息事件

客户端或者服务器端发送这一消息来通知对端用户控制事件。关于这个的消息格式参考 6.2 节。

支持以下用户控制事件类型：

事件	描述
Stream Begin (=0)	服务器发送这个事件来通知客户端一个流已就绪并可以用来通信。默认情况下，这一事件在成功接收到客户端的应用连接命令之后以 ID 0 发送。这一事件数据为 4 字节，代表了已就绪流的流 ID。
Stream EOF (=1)	服务器端发送这一事件来通知客户端请求的流的回放数据已经结束。在发送额外的命令之前不再发送任何数据。客户端将丢弃接收到的这个流的消息。这一事件数据为 4 字节，代表了回放已结束的流的流 ID。
StreamDry (=2)	服务器端发送这一事件来通知客户端当前流中已没有数据。当服务器端在一段时间内没有检测到任何消息，它可以通知相关客户端当前流已经没数据了。这一事件数据为 4 字节，代表了已没数据的流的流 ID。
SetBuffer Length (=3)	客户端发送这一事件来通知服务器端用于缓存流中任何数据的缓存大小 (以毫秒为单位)。这一事件在服务器端开始处理流之前就发送。这一事件数据的前 4 个字节代表了流 ID 后 4 个字节代表了以毫秒为单位的缓存的长度。
StreamIs Recorded (=4)	服务器端发送这一事件来通知客户端当前流是一个录制流。这一事件数据为 4 字节，代表了录制流的流 ID。
PingRequest (=6)	服务器端发送这一事件用于测试是否能够送达客户端。时间数据是一个 4 字节的 timestamp，代表了服务器端发送这一命令时的服务器本地时间。客户端在接收到这一消息后会立即发送 PingResponse 回复。
PingResponse (=7)	客户端作为对 ping 请求的回复发送这一事件到服务器端。这一事件数据是为一个 4 字节的 timestamp，就是

## 7.2. 命令类型

客户端和服务器端交换 AMF 编码的命令。服务器端发送一个命令消息，这个命令消息由命令名、事务 ID 以及包含有相关参数的命令对象组成。例如，包含有 'app' 参数的连接命令，这个命令说明了客户端连接到的服务器端的应用名。接收者处理这一命令并回发一个同样事务 ID 的响应。回复字符串可以是 \_result、\_error 或者一个方法名的任意一个，比如，verifyClient 或者 contactExternalServer。

命令字符串 \_result 或者 \_error 是响应信号。事务 ID 指示出响应所指向的命令。这和 AMAP 和其他一些协议的标签一样。命令字符串中的方法名表示发送者试图执行接收者一端的一个方法。

以下类的对象用于发送不同的命令：

NetConnection 代表上层的服务器端和客户端之间连接的一个对象。

NetStream 一个代表发送音频流、视频流和其他相关数据的通道的对象。当然，我们也会发送控制数据流的命令，诸如 play、pause 等等。

### 7.2.1. NetConnection 命令

NetConnection 管理着一个客户端应用和服务器端之间的双向连接。此外，它还提供远程方法的异步调用。

NetConnection 可以发送以下命令：

- connect
- call
- close
- createStream

#### 7.2.1.1. connect 命令

客户端发送 connect 命令到服务器端来请求连接到一个服务器应用的实例。

由客户端发送到服务器端的 connect 命令结构如下：

字段名	类型	描述
Command Name	字符串	命令的名字。设置为"connect"。
Transaction ID	数字	总是设置为 1.
Command Object	对象	具有名值对的命令信息对象。
Optional User Arguments	对象	任意可选信息。

以下是为 connect 命令中使用name-value pairs对象的描述。

属性	类型	描述	范例
app	字符串	客户端连接到的服务器端应用的名字。	testapp
flashver	字符串	Flash Player 版本号。 和ApplicationScript getversion() 方法返回的是同一个字符串。	FMSc/1.0
swfUrl	字符串	进行当前连接的 SWF 文件源地址。	<a href="file:///C:/FlvPlayer.swf">file:///C:/FlvPlayer.swf</a>
2017年1月11日	字符串	服务器 URL。具有以下格式： protocol://serverName:port/appName/appInstance	rtmp://localhost:1935/testapp/instance1
fpad	布尔	如果使用了代理就是 true。	true 或者 false。
audioCodecs	数字	表明客户端所支持的音频编码。	SUPPORT_SND_MP3
videoCodecs	数字	表明支持的视频编码。	SUPPORT_VID_SORENSEN
videoFunction	数字	表明所支持的特殊视频方法。	SUPPORT_VID_CLIENT_SEEK
pageUrl	字符串	SWF 文件所加载的网页 URL。	<a href="http://somehost/sample.html">http://somehost/sample.html</a>

audioCodecs 属性的标识值：

Codec Flag	Usage	Value
SUPPORT_SND_NONE	Raw sound, no compression	0x0001
SUPPORT_SND_ADPCM	ADPCM compression	0x0002
SUPPORT_SND_MP3	mp3 compression	0x0004
SUPPORT_SND_INTEL	Not used	0x0008
SUPPORT_SND_UNUSED	Not used	0x0010
SUPPORT_SND_NELLY8	NellyMoser at 8-kHz compression	0x0020
SUPPORT_SND_NELLY	NellyMoser compression (5, 11, 22, and 44 kHz)	0x0040
SUPPORT_SND_G711A	G711A sound compression (Flash Media Server only)	0x0080
SUPPORT_SND_G711U	G711U sound compression (Flash Media Server only)	0x0100
SUPPORT_SND_NELLY16	NellyMouser at 16-kHz compression	0x0200
SUPPORT_SND_AAC	Advanced audio coding (AAC) codec	0x0400
SUPPORT_SND_SPEEX	Speex Audio	0x0800
SUPPORT_SND_ALL	All RTMP-supported audio codecs	0xFFFF

videoCodecs 属性的标识值：

Codec Flag	Usage	Value
SUPPORT_VID_UNUSED	Obsolete value	0x0001
SUPPORT_VID_JPEG	Obsolete value	0x0002
SUPPORT_VID_SORENSEN	Sorenson Flash video	0x0004
SUPPORT_VID_HOMEBREW	V1 screen sharing	0x0008
SUPPORT_VID_VP6 (On2)	On2 video (Flash 8+)	0x0010
SUPPORT_VID_VP6ALPHA (On2 with alpha channel)	On2 video with alpha channel	0x0020

SUPPORT_VID_HOMEBREWV (screensharing v2)	Screen sharing version 2 (Flash 8+)	0x0040
SUPPORT_VID_H264	H264 video	0x0080
SUPPORT_VID_ALL	All RTMP-supported video codecs	0x00FF

videoFunction 属性的标识值：

Function Flag	Usage	Value
SUPPORT_VID_CLIENT_SEEK	Indicates that the client can perform frame-accurate seeks.	1

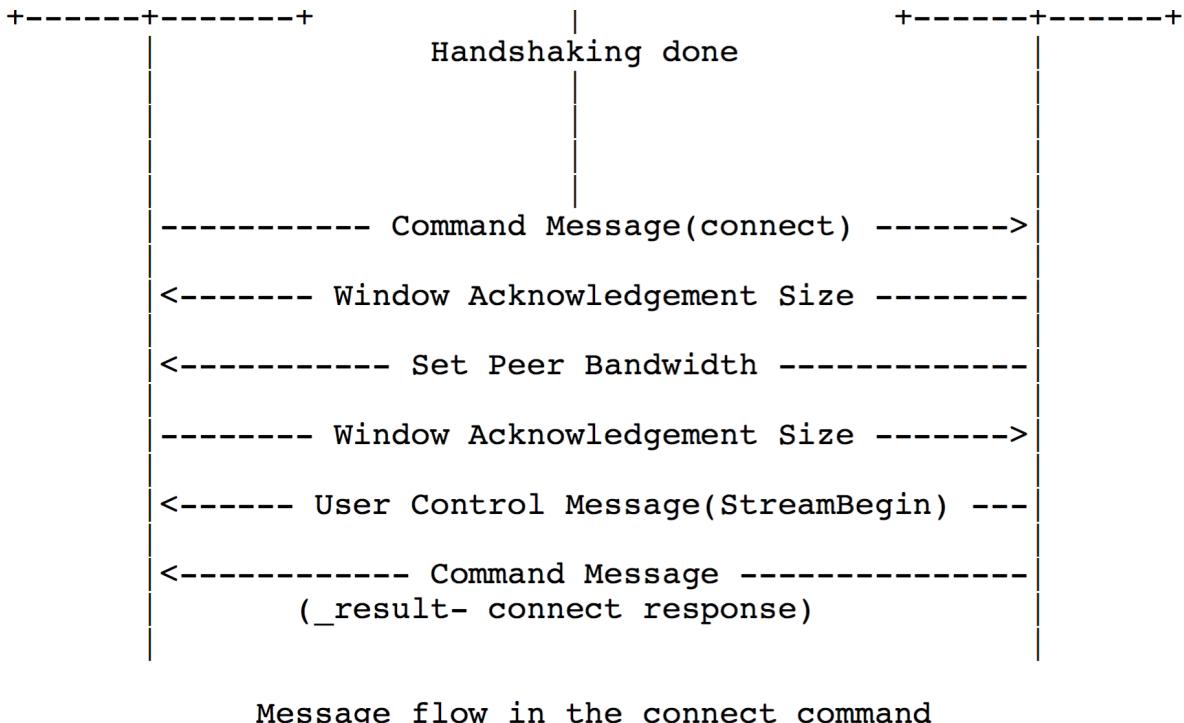
encoding 属性值：

Encoding Type	Usage	Value
AMF0	AMF0 object encoding supported by Flash 6 and later	0
AMF3	AMF3 encoding from Flash 9 (AS3)	3

服务器端到客户端的命令的结构如下：

Field Name	Type	Description
Command Name	String	_result or _error; indicates whether the response is result or error.
Transaction ID	Number	Transaction ID is 1 for connect responses
Properties	Object	Name-value pairs that describe the properties(fmsver etc.) of the connection.
Information	Object	Name-value pairs that describe the response from the server. 'code', 'level', 'description' are names of few among such information.

Client	Server
--------	--------



Message flow in the connect command

命令执行时消息流动如下：

1. 客户端发送 connect 命令到服务器端以请求对服务器端应用实例的连接。
2. 收到 connect 命令后，服务器端发送协议消息 '窗口确认大小' 到客户端。服务器端也会连接到 connect 命令中提到的应用。
3. 服务器端发送协议消息 '设置对端带宽' 到客户端。
4. 在处理完协议消息 '设置对端带宽' 之后客户端发送协议消息 '窗口确认大小' 到服务器端。
5. 服务器端发送另一个用户控制消息 (StreamBegin) 类型的协议消息到客户端。
6. 服务器端发送结果命令消息告知客户端连接状态 (success/fail)。这一命令定义了事务 ID (常常为 connect 命令设置为 1)。这一消息也定义了一些属性，比如 FMS 服务器版本 (字符串)。此外，它还定义了其他连接关联到的信息，比如 level (字符串)、code (字符串)、description (字符串)、objectencoding (数字) 等等。

### 7.2.1.2. call 方法

NetConnection 对象的 call 方法执行接收端远程方法的调用 (PRC)。被调用的 PRC 名字作为一个参数传给调用命令。

发送端发送给接收端的命令结构如下：

字段名	类型	描述
Procedure Name	字符串	调用的远程方法的名字。

Transaction ID	数字	如果期望回复我们要给一个事务 ID。否则我们传 0 值即可。
Command Object	对象	如果存在一些命令信息要设置这个对象，否则置空。
Optional Arguments	对象	任意要提供的可选参数。

回复的命令结构如下：

字段名	类型	描述
Command Name	字符串	命令的名字。
Transaction ID	数字	响应所属的命令的 ID。
Command Object	对象	如果存在一些命令信息要设置这个对象，否则置空。
Response	对象	调用方法的回复。

#### 7.2.1.3. createStream 命令

客户端发送这一命令到服务器端以为消息连接创建一个逻辑通道。音频、视频和元数据使用 createStream 命令创建的流通道传输。

NetConnection 是默认的通信通道，流 ID 为 0。协议和一些命令消息，包括 createStream，使用默认的通信通道。

客户端发送给服务器端的命令结构如下：

字段名	类型	描述
Command Name	字符串	命令名。设置给 "createStream"。
Transaction ID	数字	命令的事务 ID。
Command Object	对象	如果存在一些命令信息要设置这个对象，否则置空。

服务器端发送给客户端的命令结构如下：

字段名	类型	描述
Command Name	字符串	_result 或者 _error；表明回复是一个结果还是错误。
Transaction ID	数字	响应所属的命令的 ID。

Command Object	对象	如果存在一些命令信息要设置这个对象，否则置空。
Stream ID	数字	返回值要么是一个流 ID 要么是一个错误信息对象。

### 7.2.2. NetStream 命令

NetStream 定义了传输通道，通过这个通道，音频流、视频流以及数据消息流可以通过连接客户端到服务端的 NetConnection 传输。

以下命令可以由客户端使用 NetStream 往服务器端发送：

- play
- play2
- deleteStream
- closeStream
- receiveAudio
- receiveVideo
- publish
- seek
- pause

服务器端使用 "onStatus" 命令向客户端发送 NetStream 状态：

字段名	类型	描述
Command Name	字符串	命令名 "onStatus"。
Transaction ID	数字	事务 ID 设置为 0。
Command Object	Null	onStatus 消息没有命令对象。
Info Object	对象	一个 AMF 对象至少要有以下三个属性。"level" (字符串)：这一消息的等级，"warning"、"status"、"error" 中的某个值；"code" (字符串)：消息码，例如 "NetStream.Play.Start"；"description" (字符串)：关于这个消息人类可读描述。

### 1.2.2.1. play 命令

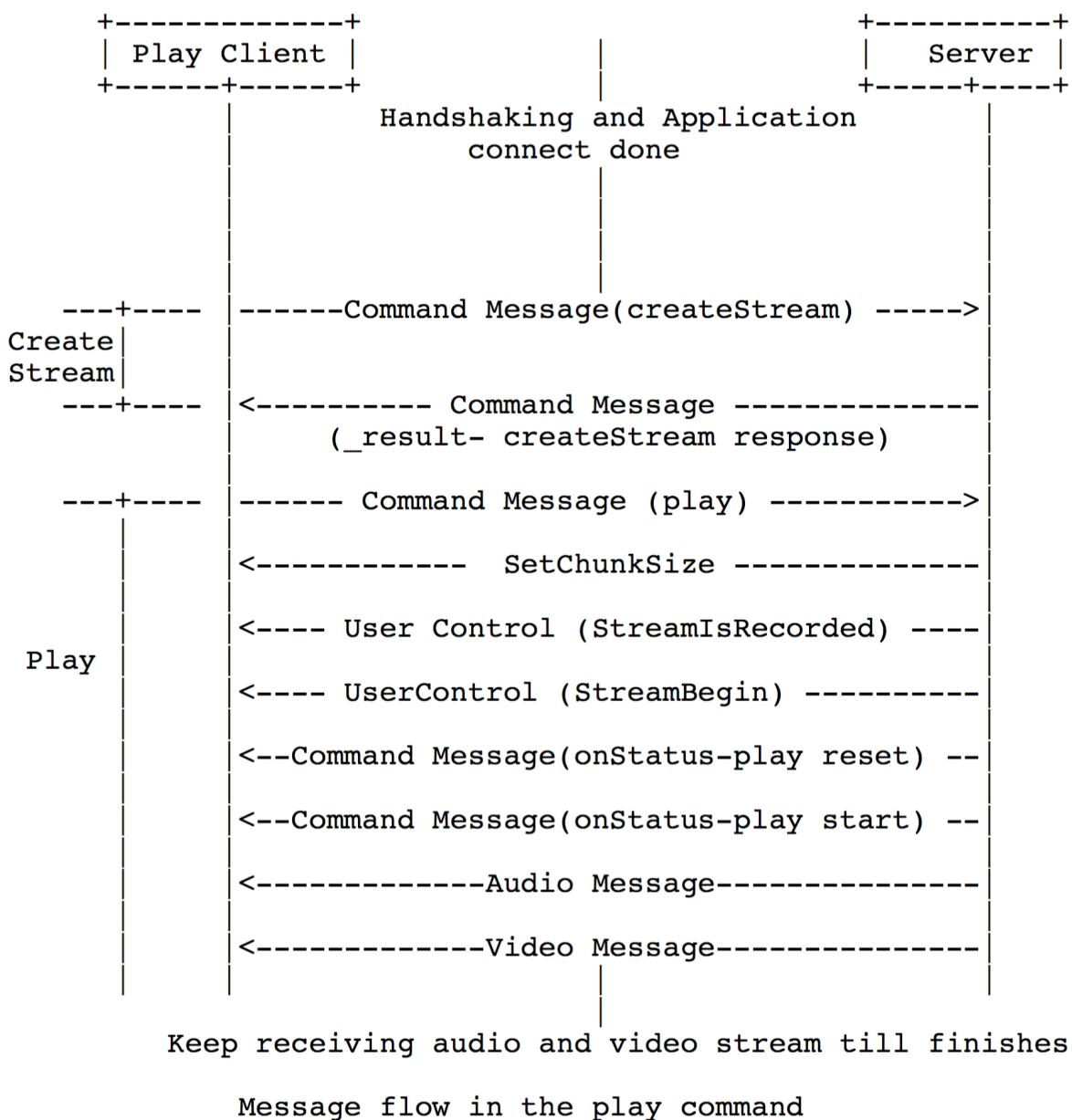
客户端发送这一命令到服务器端以播放流。也可以多次使用这一命令以创建一个播放列表。

如果你想要创建一个动态的播放列表这一可以在不同的直播流或者录制流之间进行切换播放的话，多次调用 play 方法，并在每次调用时传递重置为 false。相反的，如果你想要立即播放指定流，将其他等待播放的流清空，并为重置设为 true。

客户端发送到服务器端的命令结构如下：

字段名	类型	描述
Command Name	字符串	命令名。设为 "play"。
Transaction ID	数字	事务 ID 设为 0。
Command Object	Null	命令信息不存在。设为 null 类型。
Stream Name	字符串	要播放流的名字。要播放视频 (FLV) 文件，使用没有文件扩展名的名字对流名进行定义 (例如，"sample")。要重播 MP3 或者 ID3，你必须在流名前加上 mp3：例如，"mp3:sample"。要播放 H.264/AAC 文件，你必须在流名前加上 mp4：并指定文件扩展名。例如，要播放 sample.m4v 文件，定义 "mp4:sample.m4v"。
Start	数字	一个可选的参数，以秒为单位定义开始时间。默认值为 -2，表示用户首先尝试播放流名字段中定义的直播流。如果那个名字的直播流没有找到，它将播放同名的录制流。如果没有那个名字的录制流，客户端将等待一个新的那个名字的直播流，并当其有效时进行播放。如果你在 Start 字段中传递 -1，那么就只播放流名中定义的那个名字的直播流。如果你在 Start 字段中传递 0 或一个整数，那么将从 Start 字段定义的时间开始播放流名中定义的那个录制流。如果没有找到录制流，那么将播放播放列表中的下一项。
Duration	数字	一个可选的参数，以秒为单位定义了回放的持续时间。默认值为 -1。-1 值意味着一个直播流会一直播放直到它不再可用或者一个录制流一直播放直到结束。如果你传递 0 值，它将只播放单一帧，因为播放时间已经在录制流的开始的 Start 字段指定了。假定定义在 Start 字段中的值大于或者等于 0。如果你传递一个正数，将

		播放 Duration 字段定义的一段直播流。之后，变为可播放状态，或者播放 Duration 字段定义的一段录制流。(如果流在 Duration 字段定义的时间段内结束，那么流结束时回放结束)。如果你在 Duration 字段中传递一个 -1 以外的负数的话，它将把你给的值当做 -1 处理。
Reset	布尔	一个可选的布尔值或者数字定义了是否对以前的播放列表进行 flush。



命令执行时的消息流动是为：

1. 当客户端从服务器端接收到 createStream 命令的结果是为 success 时，发送 play 命令。

2. 一旦接收到 play 命令，服务器端发送一个协议消息来设置块大小。
  3. 服务器端发送另一个协议消息 (用户控制)，这个消息中定义了 'StreamIsRecorded' 事件和流 ID。消息在前两个字节中保存事件类型，在后四个字节中保存流 ID。
  4. 服务器端发送另一个协议消息 (用户控制)，这一消息包含 'StreamBegin' 事件，来指示发送给客户端的流的起点。
  5. 如果客户端发送的 play 命令成功，服务器端发送一个 onStatus 命令消息 NetStream.Play.Start & NetStream.Play.Reset。只有当客户端发送的 play 命令设置了 reset 时服务器端才会发送 NetStream.Play.Reset。如果要播放的流没有找到，服务器端发送 onStatus 消息 NetStream.Play.StreamNotFound。
- 之后，服务器端发送视频和音频数据，客户端对其进行播放。

#### 7.2.2.2. play2

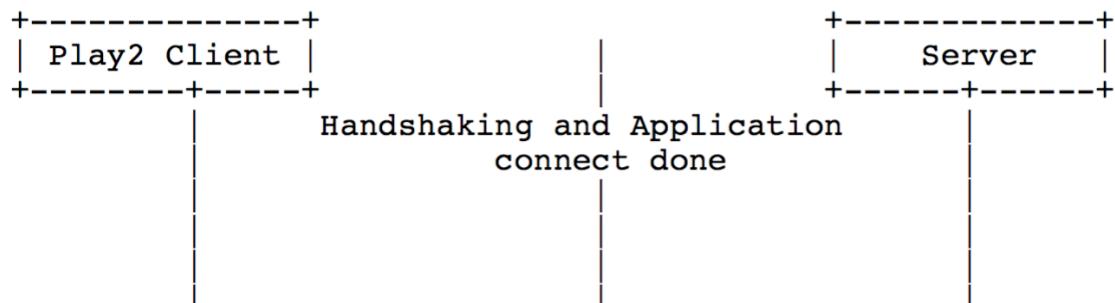
不同于 play 命令的是，play2 可以在不改变播放内容时间轴的情况下切换到不同的比特率。服务器端为客户端可以在 play2 中请求所有支持的码率维护了不同的字段。

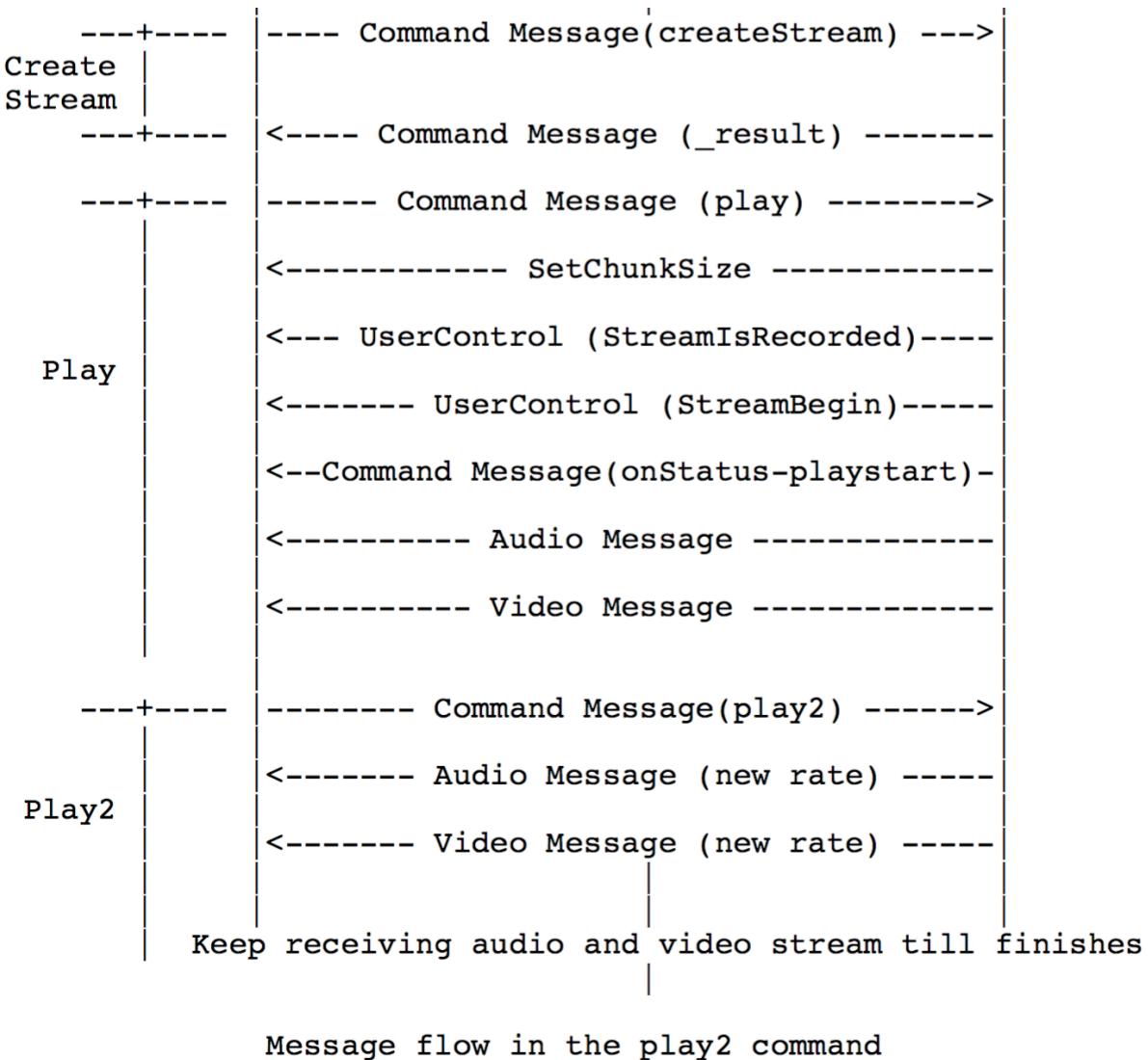
客户端发送给服务器端的命令结构如下：

字段名	类型	描述
Command Name	字符串	命令名，设置为 "play2"。
Transaction ID	数字	事务 ID 设置为 0。
Command Object	Null	命令信息不存在，设置为 null 类型。
Parameters	对象	一个 AMF 编码的对象，该对象的属性是为公开的 flash.net.NetStreamPlayOptions ActionScript 对象所描述的属性。

NetStreamPlayOptions 对象的公开属性在 ActionScript 3 语言指南中 [AS3] 有所描述。

命令执行时的消息流动如下图所示：





### 7.2.2.3. deleteStream 命令

当 NetStream 对象消亡时 NetStream 发送 deleteStream 命令。

客户端发送给服务器端的命令结构如下：

字段名	类型	描述
Command Name	字符串	命令名，设置为 "deleteStream"。
Transaction ID	数字	事务 ID 设置为 0。
Command Object	Null	命令信息对象不存在，设为 null 类型。
Stream ID	数字	服务器端消亡的流 ID。

服务器端不再发送任何回复。

### 7.2.2.4. receiveAudio 命令

NetStream 通过发送 receiveAudio 消息来通知服务器端是否发送音频到客户端。

客户端发送给服务器端的命令结构如下：

字段名	类型	描述
Command Name	字符串	命令名，设置为 "receiveAudio"。
Transaction ID	数字	事务 ID 设置为 0。
Command Object	Null	命令信息对象不存在，设置为 null 类型。
Bool Flag	布尔	true 或者 false 以表明是否接受音频。

如果发送来的 receiveAudio 命令布尔字段被设为 false 时服务器端不发送任何回复。

如果这一标识被设为 true，服务器端以状态消息 NetStream.Seek.Notify 和

NetStream.Play.Start 进行回复。

#### 7.2.2.5. receiveVideo 命令

NetStream 通过发送 receiveVideo 消息来通知服务器端是否发送视频到客户端。

客户端发送给服务器端的命令结构如下：

字段名	类型	描述
Command Name	字符串	命令名，设置为 "receiveVideo"。
Transaction ID	数字	事务 ID 设置为 0。
Command Object	Null	命令信息对象不存在，设置为 null 类型。
Bool Flag	布尔	true 或者 false 以表明是否接受视频。

如果发送来的 receiveVideo 命令布尔字段被设为 false 时服务器端不发送任何回复。

如果这一标识被设为 true，服务器端以状态消息 NetStream.Seek.Notify 和

NetStream.Play.Start 进行回复。

#### 7.2.2.6. publish 命令

客户端发送给服务器端这一命令以发布一个已命名的流。使用这个名字，任意客户端都可以播放这个流，并接受发布的音频、视频以及数据消息。

客户端发送给服务器端的命令结构如下：

字段名	类型	描述
Command Name	字符串	命令名，设置为 "publish"。
Transaction ID	数字	事务 ID 设置为 0。
Command Object	Null	命令信息对象不存在，设置为 null 类型。
Publishing Name	字符串	发布的流的名字。
Publishing Type	字符串	发布类型。可以设置为 "live"、"record" 或者 "append"。 record : 流被发布，数据被录制到一个新的文件。新文件被存储在服务器上包含服务应用目录的子路径。如果文件已存在，将重写。 append : 流被发布，数据被添加到一个文件。如果该文件没找着，将新建一个。 live : 直播数据只被发布，并不对其进行录制。

服务器端回复 onStatus 命令以标注发布的起始位置。

#### 7.2.2.7. seek 命令

客户端发送 seek 命令以查找一个多媒体文件或一个播放列表的偏移量 (以毫秒为单位)。

客户端发送到服务器端的命令结构如下：

字段名	类型	描述
Command Name	字符串	命令的名字，设为 "seek"。
Transaction ID	数字	事务 ID 设为 0。
Command Object	Null	没有命令信息对象，设置为 null 类型。
milliSeconds	数字	播放列表查找的毫秒数。

seek 命令执行成功时服务器会发送一个状态消息 NetStream.Seek.Notify。失败的话，服务器端返回一个 \_error 消息。

#### 7.2.2.8. pause 命令

客户端发送 pause 命令以告知服务器端是暂停还是开始播放。

客户端发送给服务器端的命令结构如下：

字段名	类型	描述
Command Name	字符串	命令名，设为 "pause"。
Transaction ID	数字	没有这一命令的事务 ID，设为 0。
Command Object	Null	命令信息对象不存在，设为 null 类型。
Pause/Unpause Flag	布尔	true 或者 false，来指示暂停或者重新播放。
milliSeconds	数字	流暂停或者重新开始所在的毫秒数。这个是客户端暂停的当前流时间。当回放已恢复时，服务器端值发送带有比这个值大的 timestamp 消息。

当流暂停时，服务器端发送一个状态消息

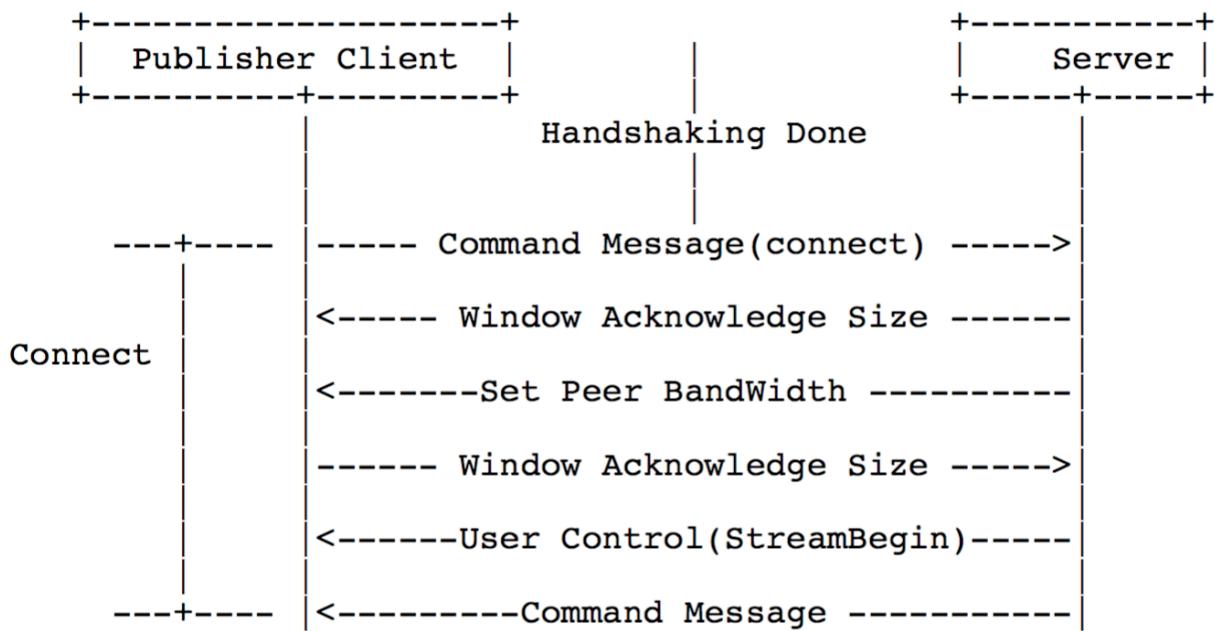
NetStream.Pause.Notify。NetStream.Unpause.Notify 只有针对没有暂停的流进行发放。失败的话，服务器端返回一个 \_error 消息。

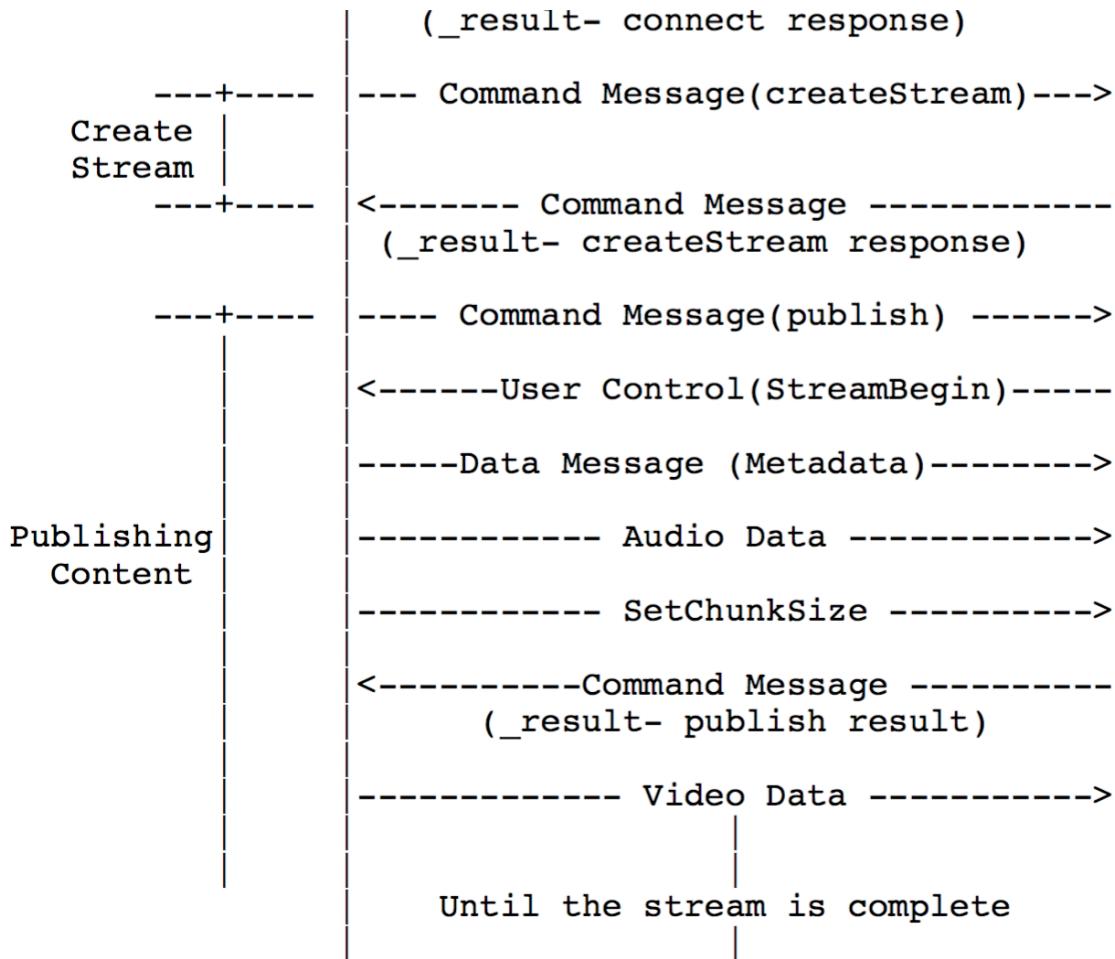
### 7.3. 消息交换例子

这里有几个解释使用 RTMP 交换消息的例子。

#### 7.3.1. 发布录制视频

这个例子说明了一个客户端是如何能够发布一个直播流然后传递视频流到服务器的。然后其他客户端可以对发布的流进行订阅并播放视频。

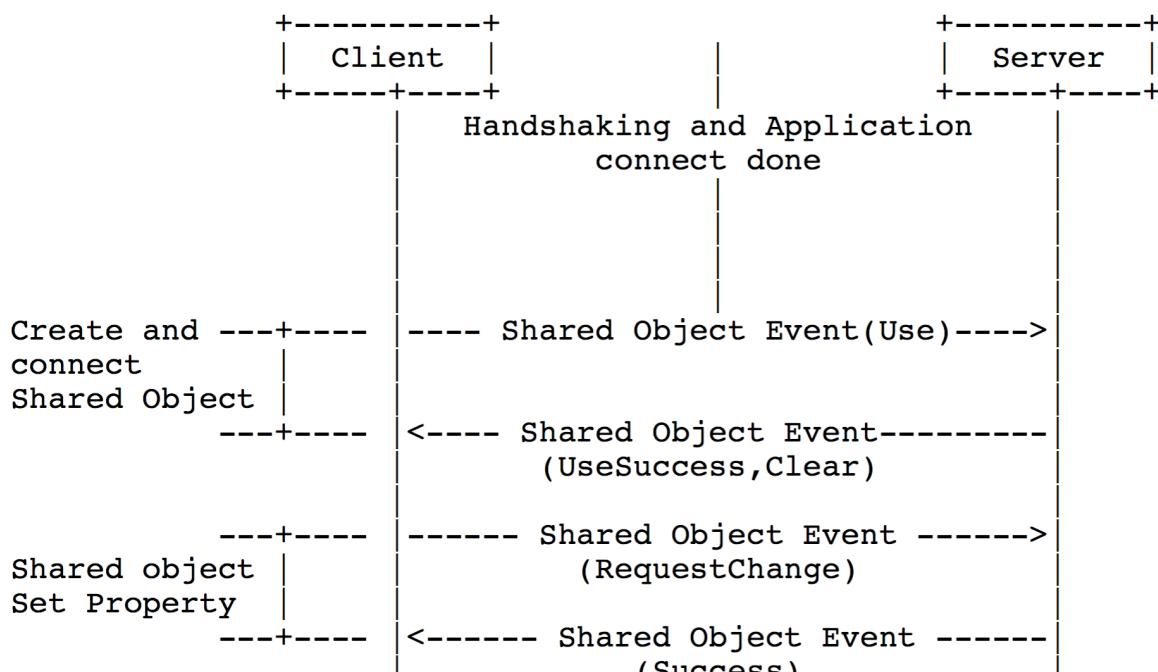


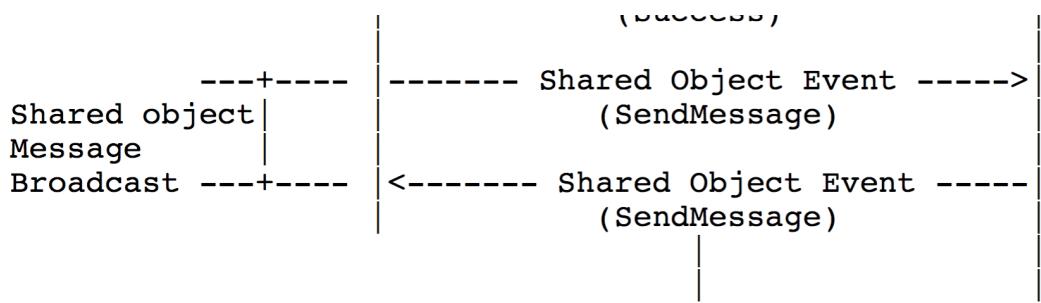


Message flow in publishing a video stream

### 7.3.2. 广播一个共享对象消息

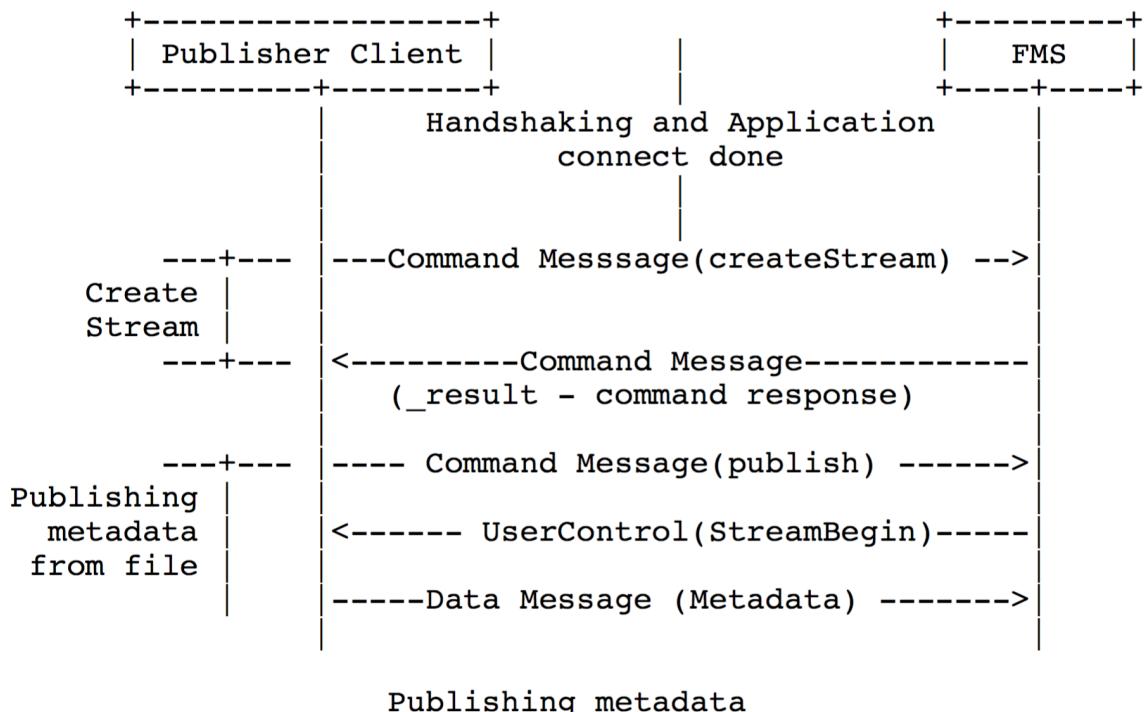
这个例子说明了在一个共享对象的创建和改变期间交换消息的变化。它也说明了共享对象消息广播的处理过程。





### 7.3.3. 发布来自录制流的元数据

这个例子描述了用于发布元数据的消息交换。



## 8. 参考文献

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[AS3] Adobe Systems, Inc., "ActionScript 3.0 Reference for the Adobe Flash Platform", 2011,  
[<http://www.adobe.com/devnet/actionscript/documentation.html>](http://www.adobe.com/devnet/actionscript/documentation.html).

[AMF0] Adobe Systems, Inc., "Action Message Format -- AMF 0", December 2007, <[http://opensource.adobe.com/wiki/download/attachments/1114283/amf0\\_spec\\_121207.pdf](http://opensource.adobe.com/wiki/download/attachments/1114283/amf0_spec_121207.pdf)>.

[AMF3] Adobe Systems, Inc., "Action Message Format -- AMF 3", May 2008, <[http://opensource.adobe.com/wiki/download/attachments/1114283/amf3\\_spec\\_05\\_05\\_08.pdf](http://opensource.adobe.com/wiki/download/attachments/1114283/amf3_spec_05_05_08.pdf)>.

#### 作者地址

Hardeep Singh Parmar (editor)

Adobe Systems Incorporated

345 Park Ave

San Jose, CA 95110-2704

US

Phone: +1 408 536 6000

Email: [hparm@adobe.com](mailto:hparm@adobe.com)

URI: <http://www.adobe.com/>

Michael C. Thornburgh (editor)

Adobe Systems Incorporated

345 Park Ave

San Jose, CA 95110-2704

US

Phone: +1 408 536 6000

Email: [mthornbu@adobe.com](mailto:mthornbu@adobe.com)

URI: <http://www.adobe.com/>

原文链接：

[http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/rtmp/pdf/rtmp\\_specification\\_1.0.pdf](http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf)。