第16讲 | 流媒体协议：如何在直播里看到美女帅哥？

2018-06-22 刘超

趣谈网络协议 进入课程 >

讲述：刘超
时长 18:28 大小 8.60M

最近直播比较火，很多人都喜欢看直播，那一个直播系统里面都有哪些组成部分，都使用了什么协议呢？

无论是直播还是点播，其实都是对于视频数据的传输。一提到视频，大家都爱看，但是一提到视频技术，大家都头疼，因为名词实在是太多了。

三个名词系列

我这里列三个名词系列，你先大致有个印象。

名词系列一：AVI、MPEG、RMVB、MP4、MOV、FLV、WebM、WMV、ASF、MKV。例如 RMVB 和 MP4，看着是不是很熟悉？

名词系列二：H.261、 H.262、H.263、H.264、H.265。这个是不是就没怎么听过了？别着急，你先记住，要重点关注 H.264。

名词系列三：MPEG-1、MPEG-2、MPEG-4、MPEG-7。MPEG 好像听说过，但是后面的数字是怎么回事？是不是又熟悉又陌生？

这里，我想问你个问题，视频是什么？我说，其实就是快速播放一连串连续的图片。

每一张图片，我们称为一**帧**。只要每秒钟帧的数据足够多，也即播放得足够快。比如每秒 30 帧，以人的眼睛的敏感程度，是看不出这是一张张独立的图片的，这就是我们常说的**帧率（FPS）**。

每一张图片，都是由**像素**组成的，假设为 1024*768（这个像素数不算多）。每个像素由 RGB 组成，每个 8 位，共 24 位。

我们来算一下，每秒钟的视频有多大？

30 帧 × 1024 × 768 × 24 = 566,231,040Bits = 70,778,880Bytes

如果一分钟呢？4,246,732,800Bytes，已经是 4 个 G 了。

是不是不算不知道，一算吓一跳？这个数据量实在是太大，根本没办法存储和传输。如果这样存储，你的硬盘很快就满了；如果这样传输，那多少带宽也不够用啊！

怎么办呢？人们想到了**编码**，就是看如何用尽量少的 Bit 数保存视频，使播放的时候画面看起来仍然很精美。**编码是一个压缩的过程。**

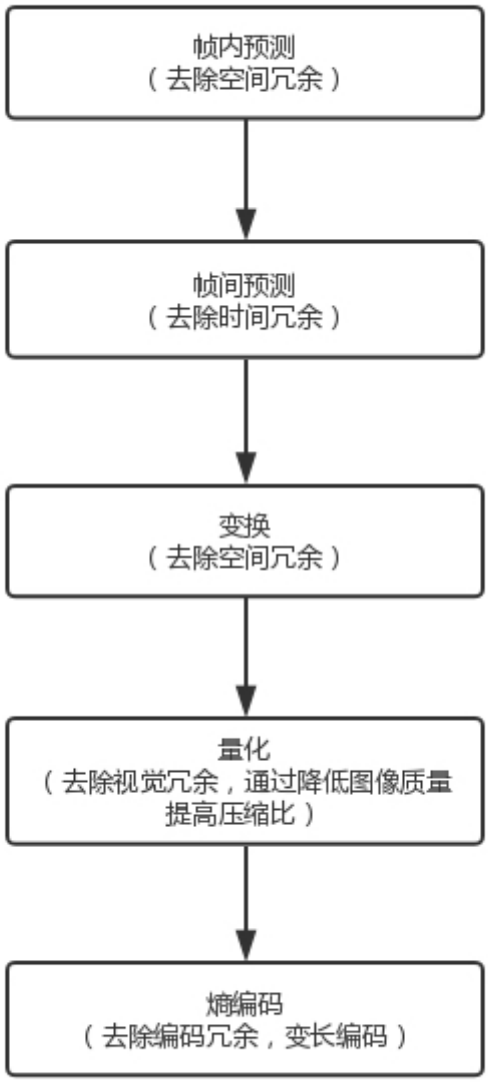
视频和图片的压缩过程有什么特点？

之所以能够对视频流中的图片进行压缩，因为视频和图片有这样一些特点。

1. **空间冗余**：图像的相邻像素之间有较强的相关性，一张图片相邻像素往往是渐变的，不是突变的，没必要每个像素都完整地保存，可以隔几个保存一个，中间的用算法计算出来。
2. **时间冗余**：视频序列的相邻图像之间内容相似。一个视频中连续出现的图片也不是突变的，可以根据已有的图片进行预测和推断。

3. **视觉冗余**：人的视觉系统对某些细节不敏感，因此不会每一个细节都注意到，可以允许丢失一些数据。
4. **编码冗余**：不同像素值出现的概率不同，概率高的用的字节少，概率低的用的字节多，类似霍夫曼编码（Huffman Coding）的思路。

总之，用于编码的算法非常复杂，而且多种多样，但是编码过程其实都是类似的。



视频编码的两大流派

能不能形成一定的标准呢？要不然开发视频播放的人得累死了。当然能，我这里就给你介绍，视频编码的两大流派。

流派一：ITU（International Telecommunications Union）的 VCEG（Video Coding Experts Group），这个称为**国际电联下的 VCEG**。既然是电信，可想而知，他们最初做视频编码，主要侧重传输。名词系列二，就是这个组织制定的标准。

流派二：ISO（International Standards Organization）的 MPEG（Moving Picture Experts Group），这个是**ISO 旗下的 MPEG**，本来是做视频存储的。例如，编码后保存在 VCD 和 DVD 中。当然后来也慢慢侧重视频传输了。名词系列三，就是这个组织制定的标准。

后来，ITU-T（国际电信联盟电信标准化部门，ITU Telecommunication Standardization Sector）与 MPEG 联合制定了 H.264/MPEG-4 AVC，这才是我们这一节要重点关注的。

经过编码之后，生动活泼的一帧一帧的图像，就变成了一串串让人看不懂的二进制，这个二进制可以放在一个文件里面，按照一定的格式保存起来，这就是名词系列一。

其实这些就是视频保存成文件的格式。例如，前几个字节是什么意义，后几个字节是什么意义，然后是数据，数据中保存的就是编码好的结果。

如何在直播里看到帅哥美女？

当然，这个二进制也可以通过某种网络协议进行封装，放在互联网上传输，这个时候就可以进行网络直播了。

网络协议将**编码**好的视频流，从主播端推送到服务器，在服务器上有个运行了同样协议的服务端来接收这些网络包，从而得到里面的视频流，这个过程称为**接流**。

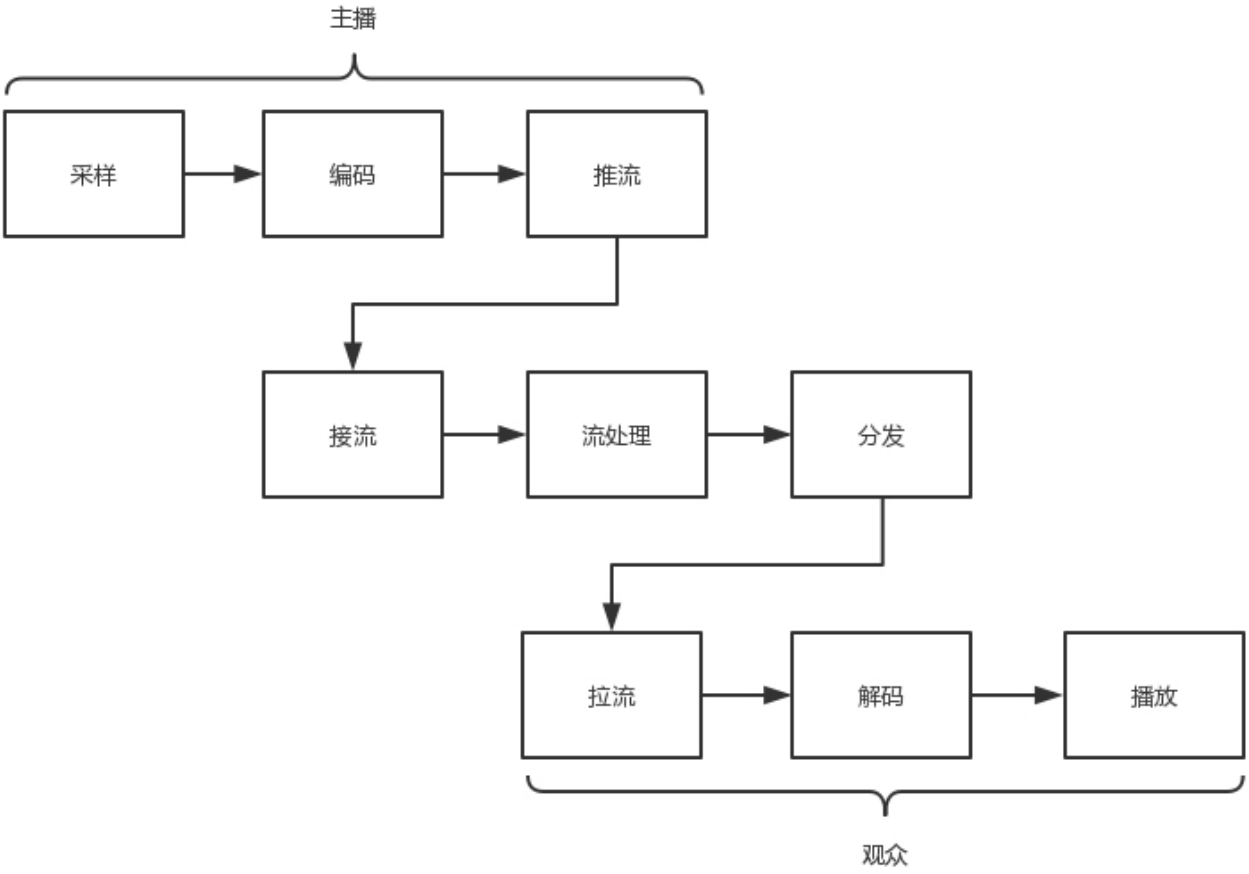
服务端接到视频流之后，可以对视频流进行一定的处理，例如**转码**，也即从一个编码格式，转成另一种格式。因为观众使用的客户端千差万别，要保证他们都能看到直播。

流处理完毕之后，就可以等待观众的客户端来请求这些视频流。观众的客户端请求的过程称为**拉流**。

如果有非常多的观众，同时看一个视频直播，那都从一个服务器上**拉流**，压力太大了，因而需要一个视频的**分发**网络，将视频预先加载到就近的边缘节点，这样大部分观众看的视频，是从边缘节点拉取的，就能降低服务器的压力。

当观众的客户端将视频流拉下来之后，就需要进行**解码**，也即通过上述过程的逆过程，将一串串看不懂的二进制，再转变成一帧帧生动的图片，在客户端**播放**出来，这样你就能看到美女帅哥啦。

整个直播过程，可以用这个的图来描述。



接下来，我们依次来看一下每个过程。

编码：如何将丰富多彩的图片变成二进制流？

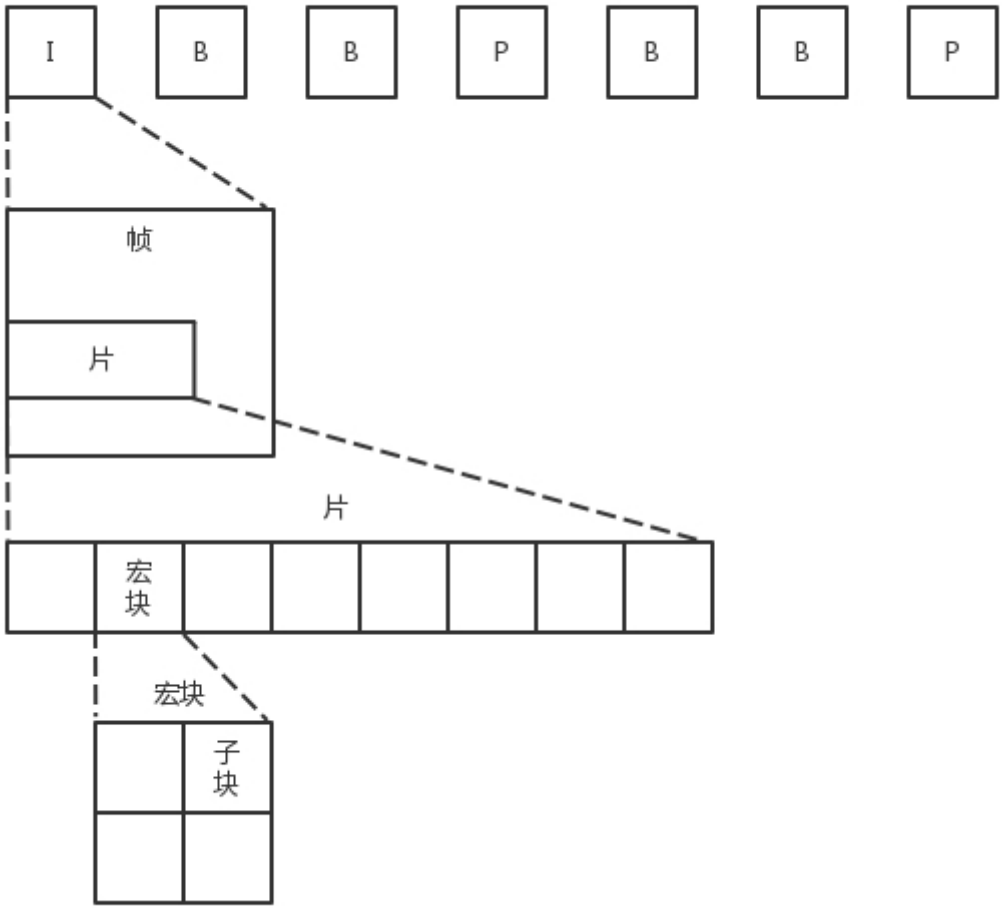
虽然我们说视频是一张张图片的序列，但是如果每张图片都完整，就太大了，因而会将视频序列分成三种帧。

I 帧，也称关键帧。里面是完整的图片，只需要本帧数据，就可以完成解码。

P 帧，前向预测编码帧。P 帧表示的是这一帧跟之前的一个关键帧（或 P 帧）的差别，解码时需要用之前缓存的画面，叠加上和本帧定义的差别，生成最终画面。

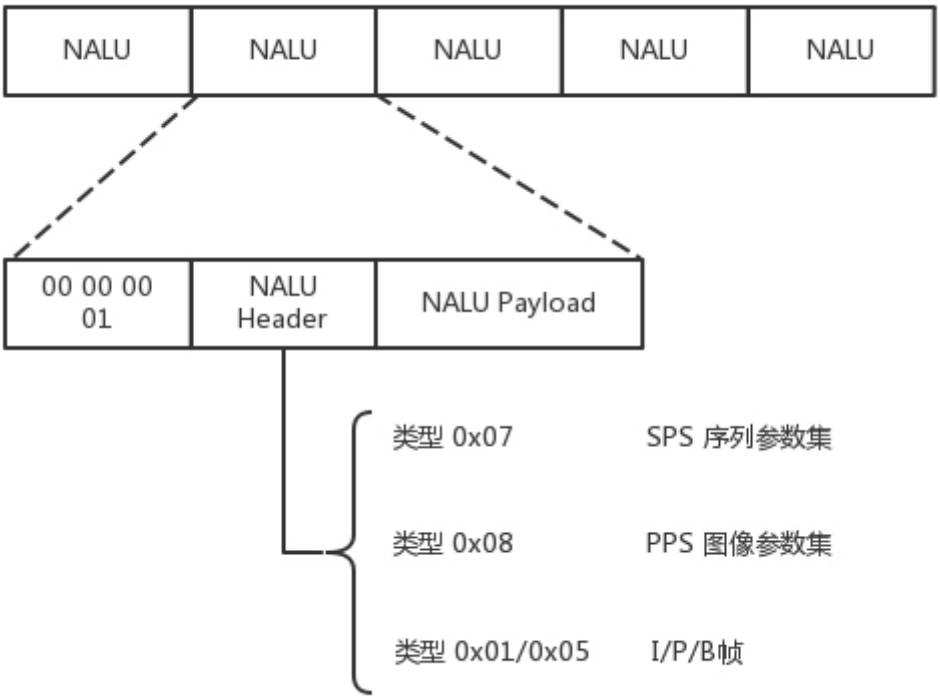
B 帧，双向预测内插编码帧。B 帧记录的是本帧与前后帧的差别。要解码 B 帧，不仅要取得之前的缓存画面，还要解码之后的画面，通过前后画面的数据与本帧数据的叠加，取得最终的画面。

可以看出，I 帧最完整，B 帧压缩率最高，而压缩后帧的序列，应该是在 IBBP 的间隔出现的。这就是**通过时序进行编码。**



在一帧中，分成多个片，每个片中分成多个宏块，每个宏块分成多个子块，这样将一张大的图分解成一个个小块，可以方便进行**空间上的编码。**

尽管时空非常立体的组成了一个序列，但是总归还是要压缩成一个二进制流。这个流是有结构的，是一个个的**网络提取层单元**（**NALU**，**Network Abstraction Layer Unit**）。变成这种格式就是为了传输，因为网络上的传输，默认的是一个个的包，因而这里也就分成了一个个的单元。



每一个 NALU 首先是一个起始标识符，用于标识 NALU 之间的间隔；然后是 NALU 的头，里面主要配置了 NALU 的类型；最终 Payload 里面是 NALU 承载的数据。

在 NALU 头里面，主要的内容是类型**NAL Type**。

0x07 表示 SPS，是序列参数集， 包括一个图像序列的所有信息，如图像尺寸、视频格式等。

0x08 表示 PPS，是图像参数集，包括一个图像的所有分片的所有相关信息，包括图像类型、序列号等。

在传输视频流之前，必须要传输这两类参数，不然无法解码。为了保证容错性，每一个 I 帧前面，都会传一遍这两个参数集合。

如果 NALU Header 里面的表示类型是 SPS 或者 PPS，则 Payload 中就是真正的参数集的内容。

如果类型是帧，则 Payload 中才是正的视频数据，当然也是一帧一帧存放的，前面说了，一帧的内容还是挺多的，因而每一个 NALU 里面保存的是一片。对于每一片，到底是 I

帧，还是 P 帧，还是 B 帧，在片结构里面也有个 Header，这里面有个类型，然后是片的内容。

这样，整个格式就出来了，**一个视频，可以拆分成一系列的帧，每一帧拆分成一系列的片，每一片都放在一个 NALU 里面，NALU 之间都是通过特殊的起始标识符分隔，在每一个 I 帧的第一片前面，要插入单独保存 SPS 和 PPS 的 NALU，最终形成一个长长的 NALU 序列**。

推流：如何把数据流打包传输到对端？

那这个格式是不是就能够直接在网上传输到对端，开始直播了呢？其实还不是，还需要将这个二进制的流打包成网络包进行发送，这里我们使用**RTMP 协议**。这就进入了第二个过程，**推流**。

RTMP 是基于 TCP 的，因而肯定需要双方建立一个 TCP 的连接。在有 TCP 的连接的基础上，还需要建立一个 RTMP 的连接，也即在程序里面，你需要调用 RTMP 类库的 Connect 函数，显示创建一个连接。

RTMP 为什么需要建立一个单独的连接呢？

因为它们需要商量一些事情，保证以后的传输能正常进行。主要就是两个事情，一个是**版本号**，如果客户端、服务器的版本号不一致，则不能工作。另一个就是**时间戳**，视频播放中，时间是很重要的，后面的数据流互通的时候，经常要带上时间戳的差值，因而一开始双方就要知道对方的时间戳。

未来沟通这些事情，需要发送六条消息：客户端发送 C0、C1、 C2，服务器发送 S0、S1、 S2。

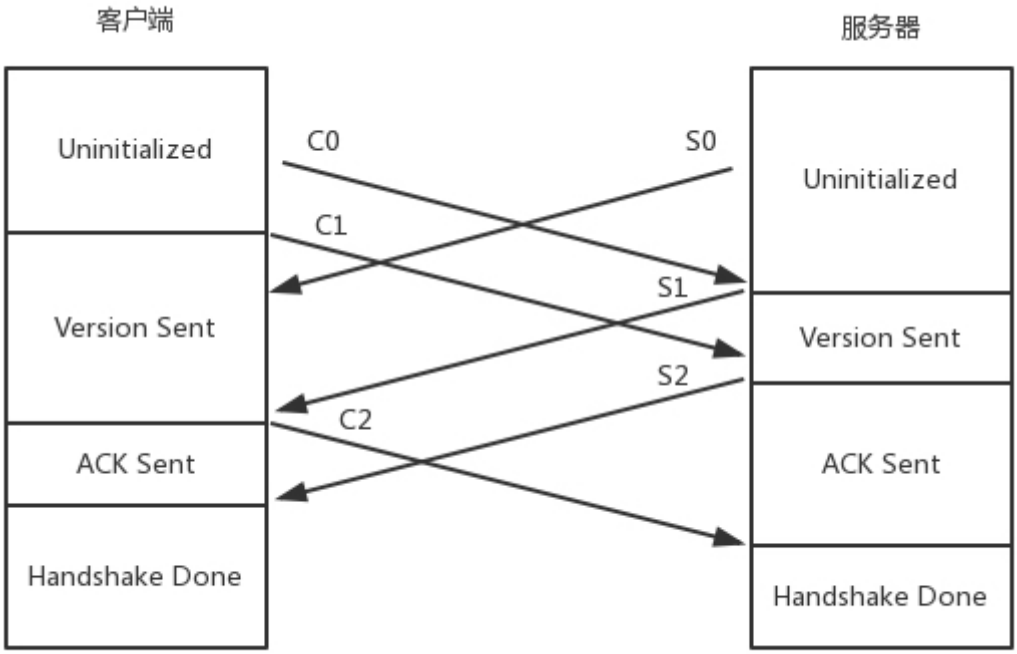
首先，客户端发送 C0 表示自己的版本号，不必等对方的回复，然后发送 C1 表示自己的时间戳。

服务器只有在收到 C0 的时候，才能返回 S0，表明自己的版本号，如果版本不匹配，可以断开连接。

服务器发送完 S0 后，也不用等什么，就直接发送自己的时间戳 S1。客户端收到 S1 的时候，发一个知道了对方时间戳的 ACK C2。同理服务器收到 C1 的时候，发一个知道了对方

时间戳的 ACK S2。

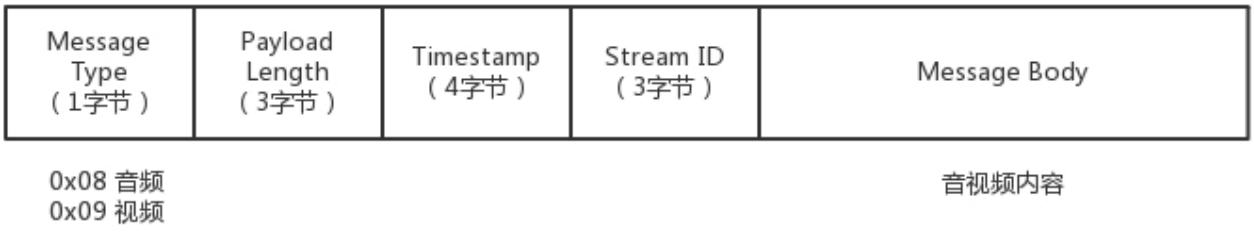
于是，握手完成。



握手之后，双方需要互相传递一些控制信息，例如 Chunk 块的大小、窗口大小等。

真正传输数据的时候，还是需要创建一个流 Stream，然后通过这个 Stream 来推流 publish。

推流的过程，就是将 NALU 放在 Message 里面发送，这个也称为**RTMP Packet 包**。Message 的格式就像这样。



发送的时候，去掉 NALU 的起始标识符。因为这部分对于 RTMP 协议来讲没有用。接下来，将 SPS 和 PPS 参数集封装成一个 RTMP 包发送，然后发送一个个片的 NALU。

RTMP 在收发数据的时候并不是以 Message 为单位的，而是把 Message 拆分成 Chunk 发送，而且必须在一个 Chunk 发送完成之后，才能开始发送下一个 Chunk。每个 Chunk 中都带有 Message ID，表示属于哪个 Message，接收端也会按照这个 ID 将 Chunk 组装成 Message。

前面连接的时候，设置的 Chunk 块大小就是指这个 Chunk。将大的消息变为小的块再发送，可以在低带宽的情况下，减少网络拥塞。

这有一个分块的例子，你可以看一下。

假设一个视频的消息长度为 307，但是 Chunk 大小约定为 128，于是会拆分为三个 Chunk。

第一个 Chunk 的 Type = 0，表示 Chunk 头是完整的；头里面 Timestamp 为 1000，总长度 Length 为 307，类型为 9，是个视频，Stream ID 为 12346，正文部分承担 128 个字节的 Data。

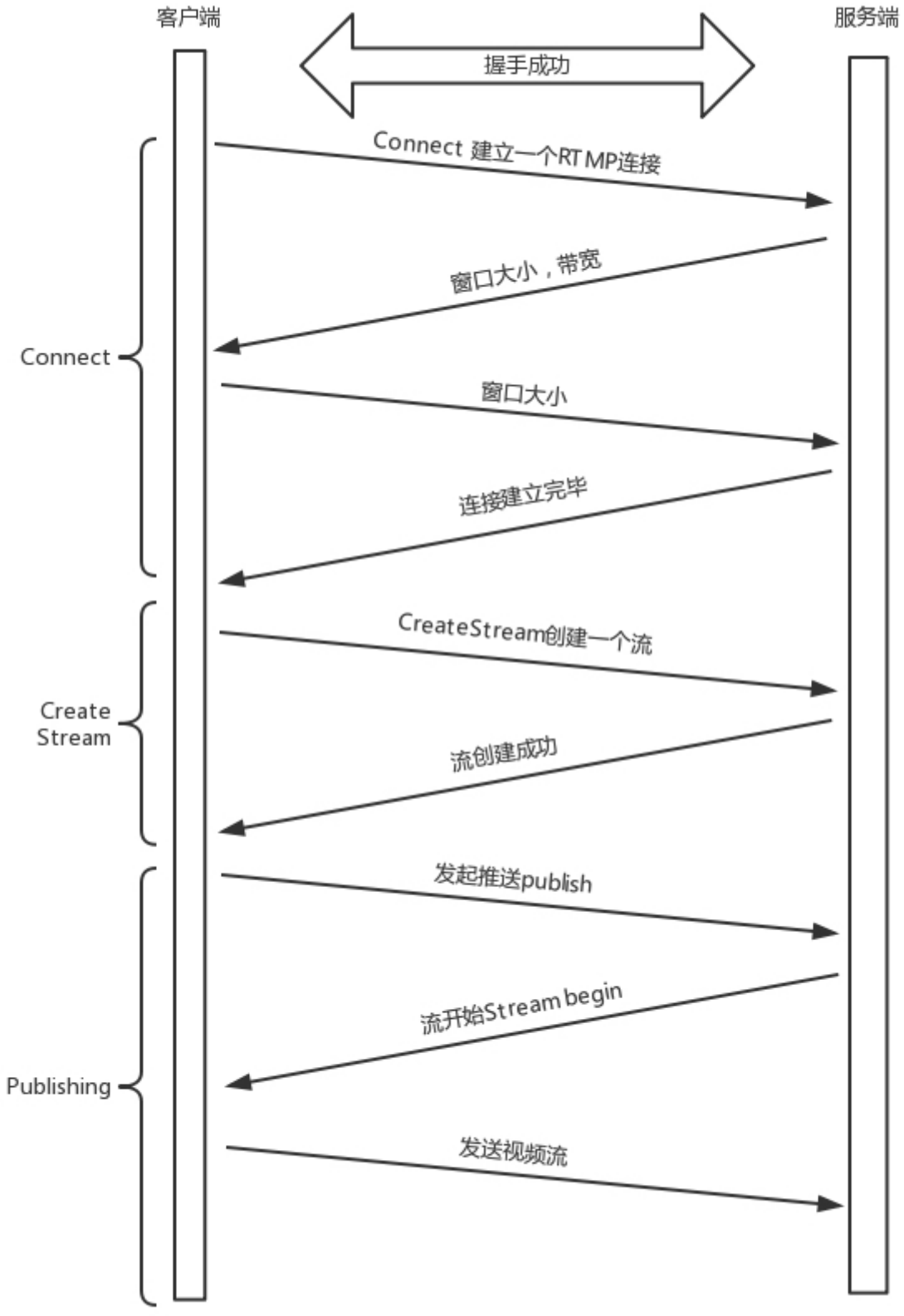
第二个 Chunk 也要发送 128 个字节，Chunk 头由于和第一个 Chunk 一样，因此采用 Chunk Type = 3，表示头一样就不再发送了。

第三个 Chunk 要发送的 Data 的长度为 307-128-128=51 个字节，还是采用 Type = 3。

Message Type（1字节）	Payload Length（3字节）	Timestamp（4字节）	Stream ID（3字节）	Message Body
0x09 视频	307	1000	12346	307个字节的视频数据

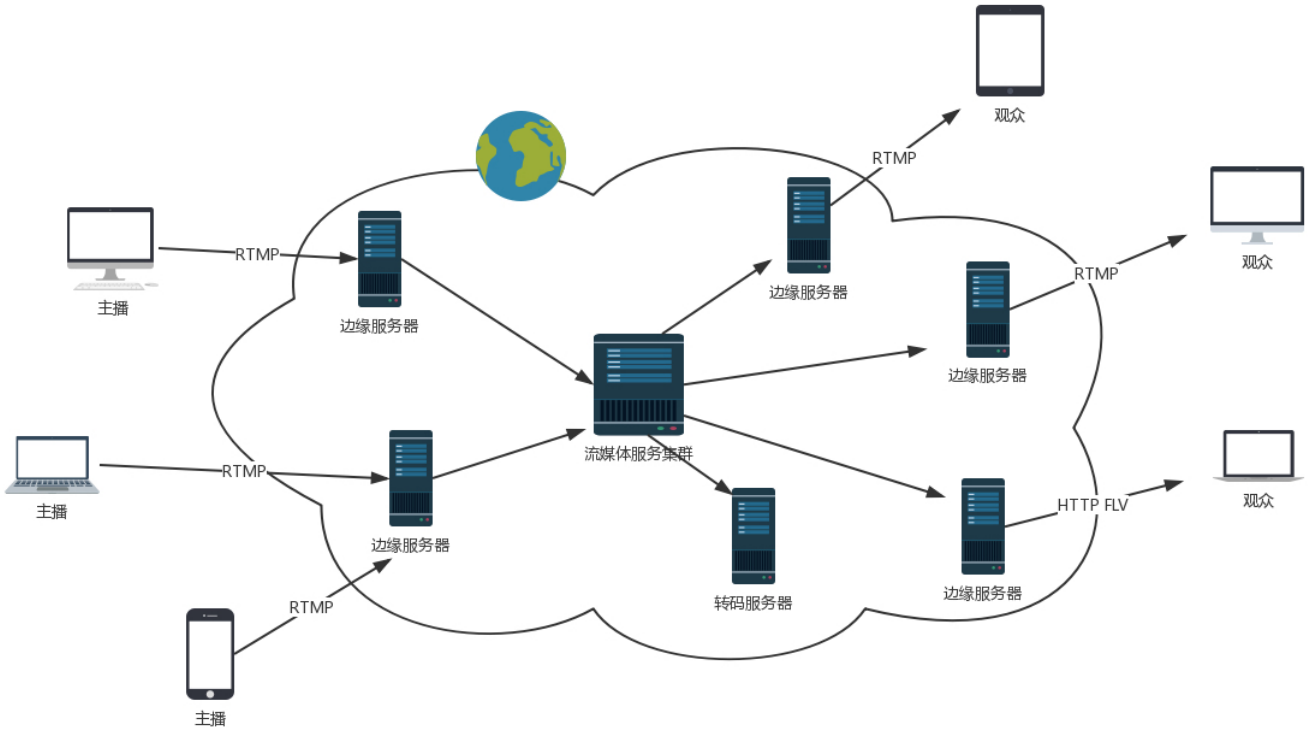
Chunk Stream ID	Chunk Type	Chunk Header	
4	0	delta: 1000 length: 307 type: 9 stream id: 12346	128字节视频数据
4	3	128字节视频数据	
4	3	51字节视频数据	

就这样数据就源源不断到达流媒体服务器，整个过程就像这样。



这个时候，大量观看直播的观众就可以通过 RTMP 协议从流媒体服务器上拉取，但是这么多的用户量，都去同一个地方拉取，服务器压力会很大，而且用户分布在全国甚至全球，如果都去统一的一个地方下载，也会时延比较长，需要有分发网络。

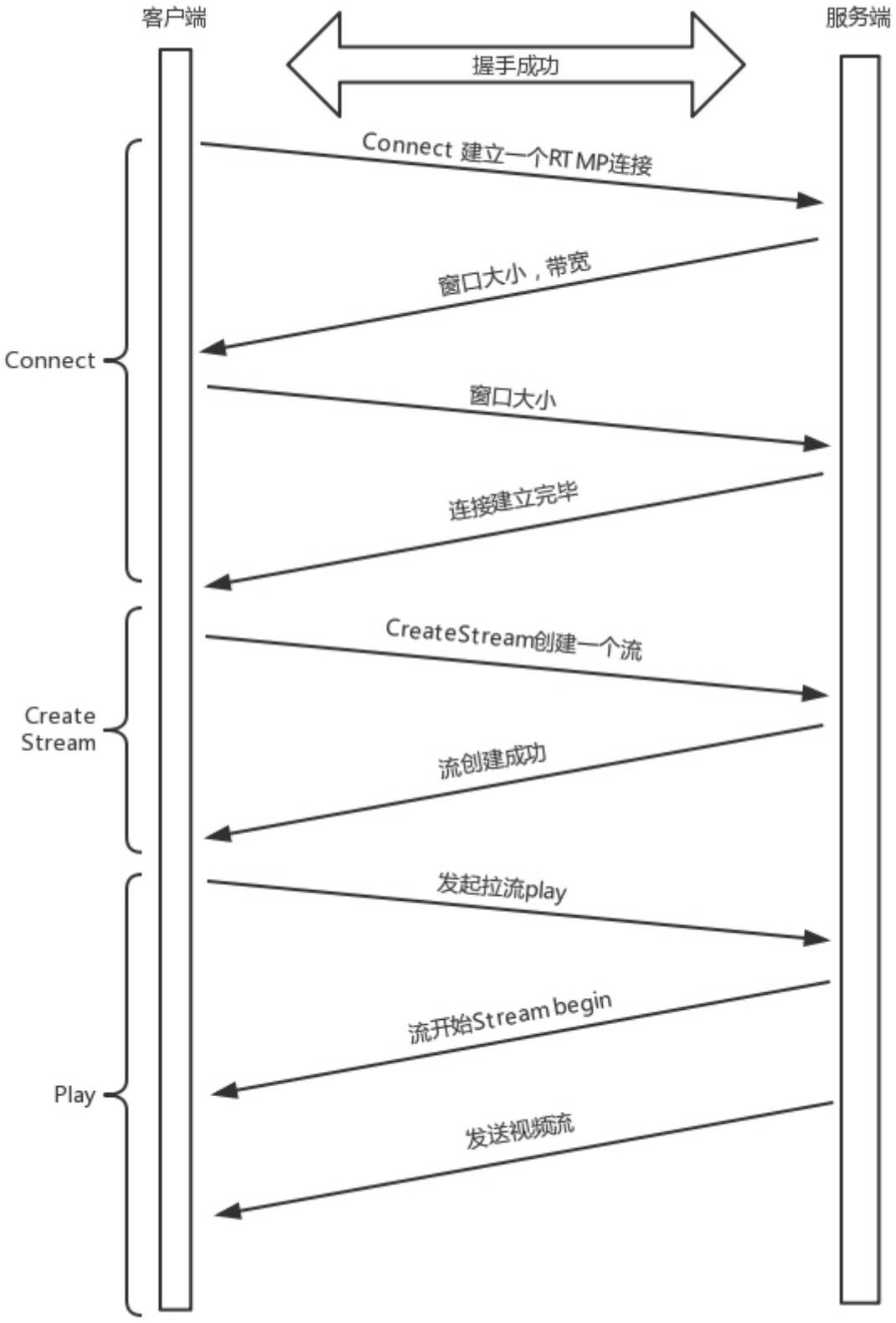
分发网络分为**中心**和**边缘**两层。边缘层服务器部署在全国各地及横跨各大运营商里，和用户距离很近。中心层是流媒体服务集群，负责内容的转发。智能负载均衡系统，根据用户的地理位置信息，就近选择边缘服务器，为用户提供推 / 拉流服务。中心层也负责转码服务，例如，把 RTMP 协议的码流转换为 HLS 码流。



这套机制在后面的 DNS、HTTPDNS、CDN 的章节会更有详细的描述。

拉流：观众的客户端如何看到视频？

接下来，我们再来看观众的客户端通过 RTMP 拉流的过程。



先读到的是 H.264 的解码参数，例如 SPS 和 PPS，然后对收到的 NALU 组成的一个个帧，进行解码，交给播发器播放，一个绚丽多彩的视频画面就出来了。

小结

好了，今天的内容就到这里了，我们来总结一下：

> 视频名词比较多，编码两大流派达成了一致，都是通过时间、空间的各种算法来压缩数据；
>
> 压缩好的数据，为了传输组成一系列 NALU，按照帧和片依次排列；
>
> 排列好的 NALU，在网络传输的时候，要按照 RTMP 包的格式进行包装，RTMP 的包会拆分成 Chunk 进行传输；
>
> 推送到流媒体集群的视频流经过转码和分发，可以被客户端通过 RTMP 协议拉取，然后组合为 NALU，解码成视频格式进行播放。

最后，给你留两个思考题：

1. 你觉得基于 RTMP 的视频流传输的机制存在什么问题？如何进行优化？
2. 在线看视频之前，大家都是把电影下载下来看的，电影这么大，你知道如何快速下载吗？

欢迎你留言和我讨论。趣谈网络协议，我们下期见！

上一篇　第15讲 | HTTPS协议：点外卖的过程原来这么复杂

下一篇　第17讲 | P2P协议：我下小电影，99%急死你

精选留言 (30)

写留言

Jason
2018-06-22

31

写的特别清楚明白，超哥绝对适合当老师。问题1，rtmp的问题是基于tcp连接的，不适合做实时流传输？改进方法，那就是把UDP引进？

展开

作者回复: 赞

fang
2018-06-27

14

RTMP握手的图是不是有点问题？服务器收到客户端c0发来的版本号才回s0，图上服务器啥也没收到就发s0，它怎么知道给谁发？

展开

Jealone
2018-06-22

9

优化方式就是QUIC协议

展开

咖啡猫口里...
2018-06-22

5

直播用TCP延迟太高了吧，，，他的保证有序，拥塞机制，，导致current等待之前的包

作者回复: 是的，赞

wahaha
2018-06-23

4

一般说的看视频其实包含了音频，不然成了哑巴剧了，请老师讲下音频。

作者回复: 有机会补充一下

anzaikk
2018-09-22

3

一开始看不懂，看了五六遍感觉很清晰，赞

展开

小白
2018-07-28

3

前面看得很顺，这篇开始就开始懵了，主要是没有做过相互的业务，老师，有什么方法来理解谢谢吗

Jason
2018-07-12

3

再度一遍，还是很牛，以rtmp为例很系统的讲述了流媒体服务的原理，其实rtsp、hls也是类似的原理，只是交互形式的不同。每个视频从业者都应该拿去读读。

展开

吃鳄鱼的猫
2018-06-23

2

现在好像都是http-flv, rtmp和hls延迟太高

展开

夏洛克的救...
2018-06-22

2

老师 是不是忘记说视频混流了

展开

作者回复: 没说音频

新征程
2018-10-09

1

rtmp与dash hls. dash与mp4这些是啥关系

展开

Malcolm
2018-09-11

1

感谢老师！讲的很透彻！

展开

July
2018-08-22

1

RTMP 沟通的版本号是指什么的版本号呢？

展开

作者回复: 对于版本号的定义：当前rtmp协议的版本号一致为“3”，0、1、2是旧版本号，已经弃用。

byte
2018-07-06

1

希望补充在线视频网站的使用协议以及音频的协议。

展开

orangleliu
2018-07-04

1

1 tcp拥塞控制 优化的话 quic kcp这种基于udp的协议

2 p2p 网络 bt或者电驴这种

展开

Hulk 1

2018-07-01

讲解清晰，大赞

展开

小新是也 1

2018-06-24

视频是一帧一帧的图片，那么音频呢？它又是怎么压缩的？

作者回复: 是的，没说音频

yan华建

2019-05-09

思考题

2.在线看视频之前，大家都是把电影下载下来看的，电影这么大，你知道如何快速下载吗？

采用p2p的方式，

yan华建

2019-05-09

思考题

1.由于RTMP采用TCP，所以TCP存在的问题，RTMP也都存在。可以采用UDP，QUIC,视频传输过程中丢了一部分的帧，对人来说没有太大差异，更加注重实时性。

jztong

2019-05-08

老师，每个NALU是一片吗？不应该是几个宏块或者子块吗？