25 | 那些常见的流媒体服务器，你该选择谁？

2019-09-10 李超

从0打造音视频直播系统 进入课程 >

讲述：李超
时长 20:44 大小 14.25M

在上一篇文章中，我们详细讨论了三种类型（Mesh、MCU 和 SFU）的多方通信架构，并从中知道**SFU 方案是目前最优的一种多方通信架构方案**，而且这种方案目前已经有很多开源的实现了，比较流行的开源项目包括 Licode、Janus-gateway、MediaSoup、Medooze 等。

当然，你也可以自己实现 SFU 流媒体服务器，但自己实现流媒体服务器困难还是蛮多的，它里面至少要涉及到 DTLS 协议、ICE 协议、SRTP/SRTCP 协议等，光理解这些协议就要花不少的时间，更何况要实现它了。

而使用开源的流媒体服务器就可以大大地减少你的工作量，但这也带来一个问题：这么多开源流媒体服务器，究竟该选哪一个呢？这就涉及到一个选型的问题，你必须要对上面提到的开源项目有个清楚的认知，知道它们各自的优劣点，才能选出更适合的那款流媒体服务器。

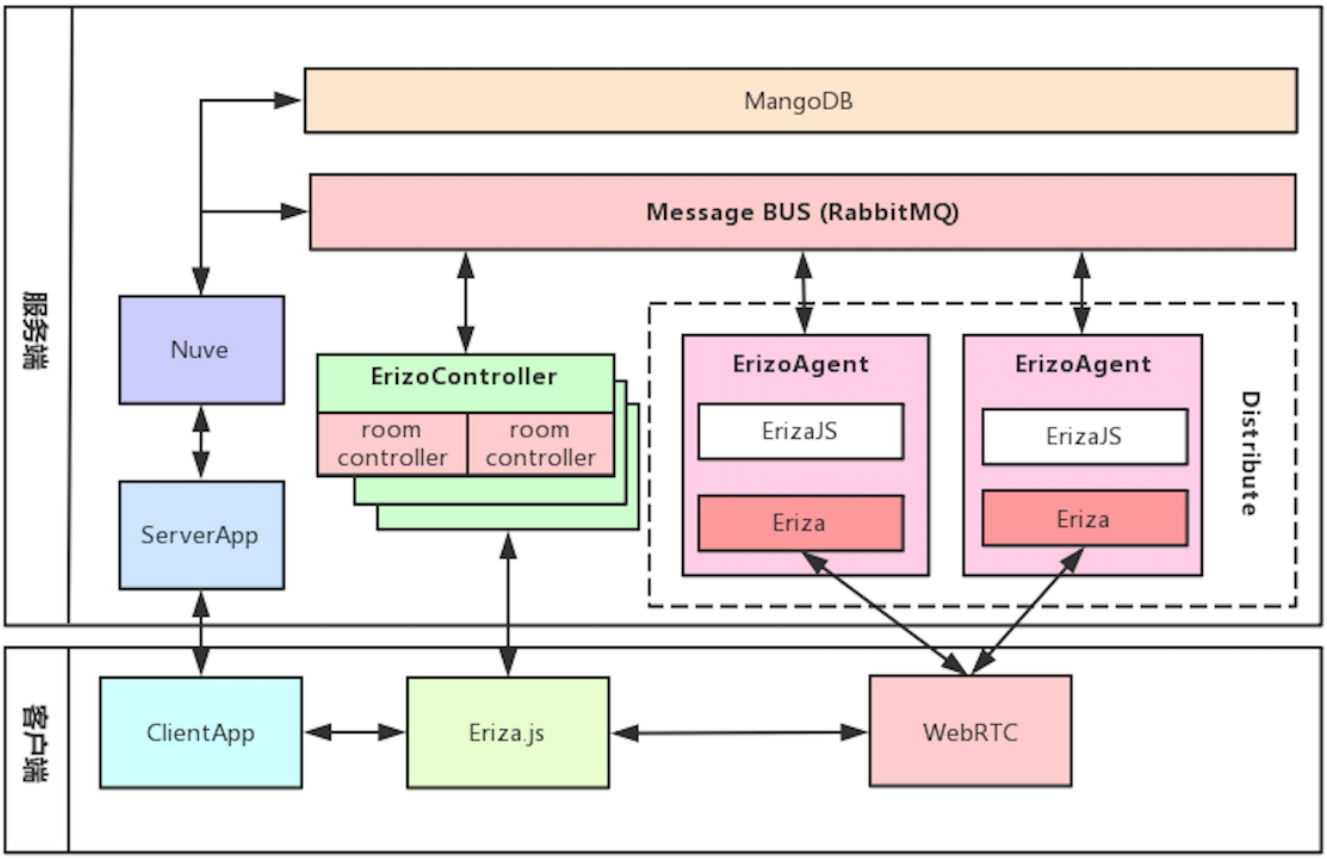
接下来我们就按顺序把这几个开源实现向你做下介绍，从而让你了解它们的优缺点，以便选出更适合你的那款流媒体服务器。

Licode

Licode 既可以用作 SFU 类型的流媒体服务器，也可以用作 MCU 类型的流媒体服务器。一般情况下，它都被用于 SFU 类型的流媒体服务器。

Licode 不仅仅是一个流媒体通信服务器，而且还是一个包括了媒体通信层、业务层、用户管理等功能的完整系统，并且该系统还支持分布式部署。

Licode 是由 C++ 和 Node.js 语言实现。其中，媒体通信部分由 C++ 语言实现，而信令控制、用户管理、房间管理用 Node.js 实现。它的源码地址为：https://github.com/lynckia/licode 。下面这张图是 Licode 的整体架构图：



Licode 架构图

通过这张图你可以看出，Licode 从功能层面来讲分成三部分，即 Nuve 、ErizoController 和 ErizoAgent 三部分，它们之间通过消息队列进行通信。

Nuve 是一个 Web 服务，用于管理用户、房间、产生 token 以及房间的均衡负载等相关工作。它使用 MangoDB 存储房间和 token 信息，但不存储用户信息。

ErizoController，用于管理控制，信令和非音视频数据都通过它接收。它通过消息队列与 Nuve 进行通信，也就是说 Nuve 可以通过消息队列对 ErizoController 进行控制。

ErizoAgent，用于音视频流媒体数据的传输，可以分布式布署。ErizoAgent 与 ErizoController 的通信也是通过消息队列，信令消息通过 ErizoController 接收到后，再通过消息队列发给 ErizoAgent，从而实现对 ErizoAgent 进行控制。

通过上面的描述，你可以知道 Licode 不仅仅是一个 SFU 流媒体服务器，它还包括了与流媒体相关的业务管理系统、信令系统、流媒体服务器以及客户端 SDK 等等，可以说它是一个比较完善的产品。

如果你使用 Licode 作为流媒体服务器，基本上不需要做二次开发了，所以这样一套系统对于没有音视频积累的公司和个人具有非常大的诱惑力。目前 Intel CS 项目就是在 Licode 基础上研发出来的，已经为不少公司提供了服务。

但 Licode 也有以下一些缺点：

在 Linux 下目前只支持 Ubuntu 14.04 版本，在其他版本上很难编译通过。

Licode 不仅包括了 SFU，而且包括了 MCU，所以它的代码结构比较重，学习和掌握它要花不少的时间。

Licode 的性能一般， 如果你把流媒体服务器的性能排在第一位的话，那么 Licode 就不是特别理想的 SFU 流媒体服务器了。

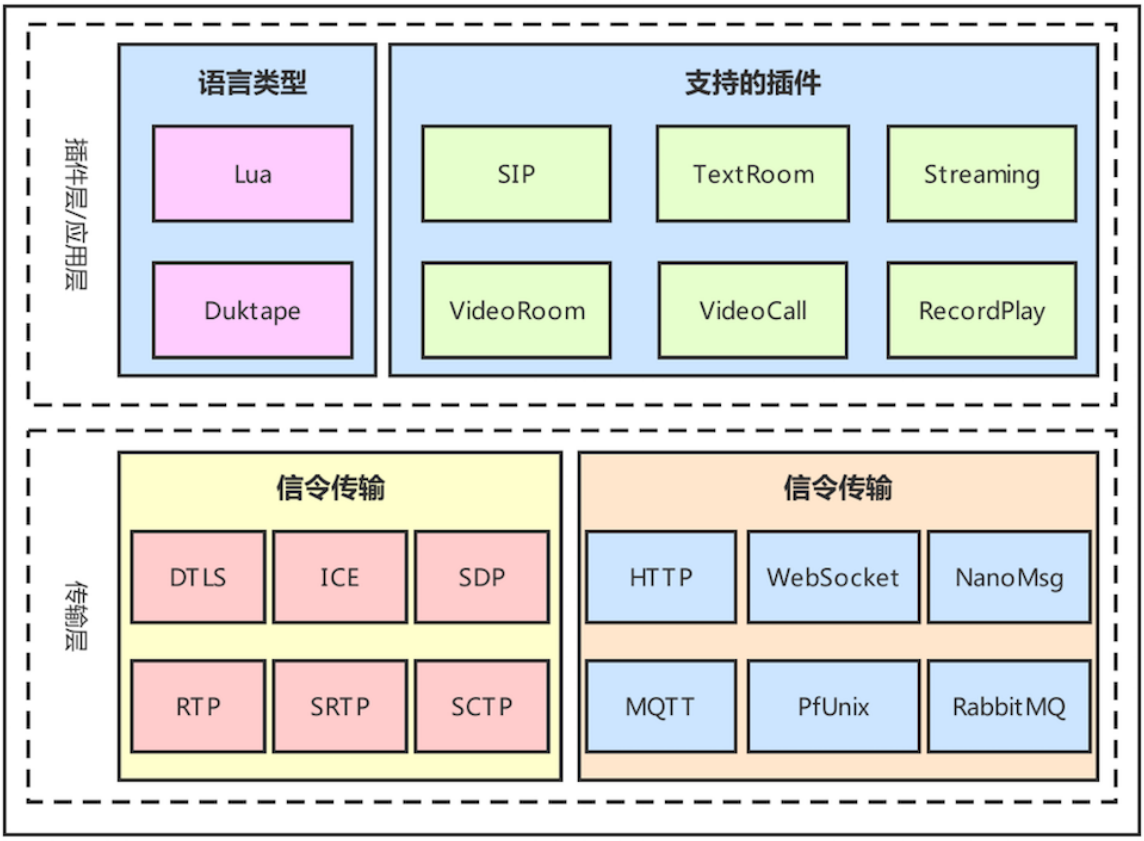
Janus-gateway

Janus 是一个非常有名的 WebRTC 流媒体服务器，它是以 Linux 风格编写的服务程序，采用 C 语言实现，支持 Linux/MacOS 下编译、部署，但不支持 Windows 环境。

它是一个开源项目，其源码的编译、安装非常简单，只要按 GitHub 上的说明操作即可。源码及编译手册的地址为：https://github.com/meetecho/janus-gateway 。

Janus 的部署也十分简单，具体步骤详见文档，地址为：https://janus.conf.meetecho.com/docs/deploy.html 。

下面我们来看一下 Janus 的整体架构，其架构如下图所示：



Janus 架构图

从上面的架构图中，你可以看出 Janus 分为两层，即**应用层**和**传输层**。

插件层又称为应用层，每个应用都是一个插件，可以根据用户的需要动态地加载或卸载掉某个应用。插件式架构方案是非常棒的一种设计方案，灵活、易扩展、容错性强，尤其适用于业务比较复杂的业务，但缺点是实现复杂，成本比较高。

在 Janus 中默认支持的插件包括以下几个。

SIP：这个插件使得 Janus 成了 SIP 用户的代理，从而允许 WebRTC 终端在 SIP 服务器（如 Asterisk）上注册，并向 SIP 服务器发送或接收音视频流。

TextRoom：该插件使用 DataChannel 实现了一个文本聊天室应用。

Streaming：它允许 WebRTC 终端观看 / 收听由其他工具生成的预先录制的文件或媒体。

VideoRoom：它实现了视频会议的 SFU 服务，实际就是一个音 / 视频路由器。

VideoCall：这是一个简单的视频呼叫的应用，允许两个 WebRTC 终端相互通信，它与 WebRTC 官网的例子相似（https://apprtc.appspot.com），不同点是这个插件要经过服务端进行音视频流中转，而 WebRTC 官网的例子走的是 P2P 直连。

RecordPlay：该插件有两个功能，一是将发送给 WebRTC 的数据录制下来，二是可以通过 WebRTC 进行回放。

传输层包括**媒体数据传输**和**信令传输。媒体数据传输层**主要实现了 WebRTC 中需要有流媒体协议及其相关协议，如 DTLS 协议、ICE 协议、SDP 协议、RTP 协议、SRTP 协议、SCTP 协议等。

信令传输层用于处理 Janus 的各种信令，它支持的传输协议包括 HTTP/HTTPS、WebSocket/WebSockets、NanoMsg、MQTT、PfUnix、RabbitMQ。不过需要注意的是，有些协议是可以通过编译选项来控制是否安装的，也就是说这些协议并不是默认全部安装的。另外，Janus 所有信令的格式都是采用 Json 格式。

通过上面的分析，你可以知道，Janus 整体架构采用了插件的方案，这种架构方案非常优秀，用户可以根据自己的需要非常方便地在上面编写自己的应用程序。而且它目前支持的功能非常多，比如支持 SIP、 RTSP、音视频文件播放、录制等等，所以在与其他系统的融合性上有非常大的优势。另外，它底层的代码是由 C 语言编写的，性能也非常强劲。Janus 的开发、部署手册也非常完善，因此它是一个非常棒的开源项目。

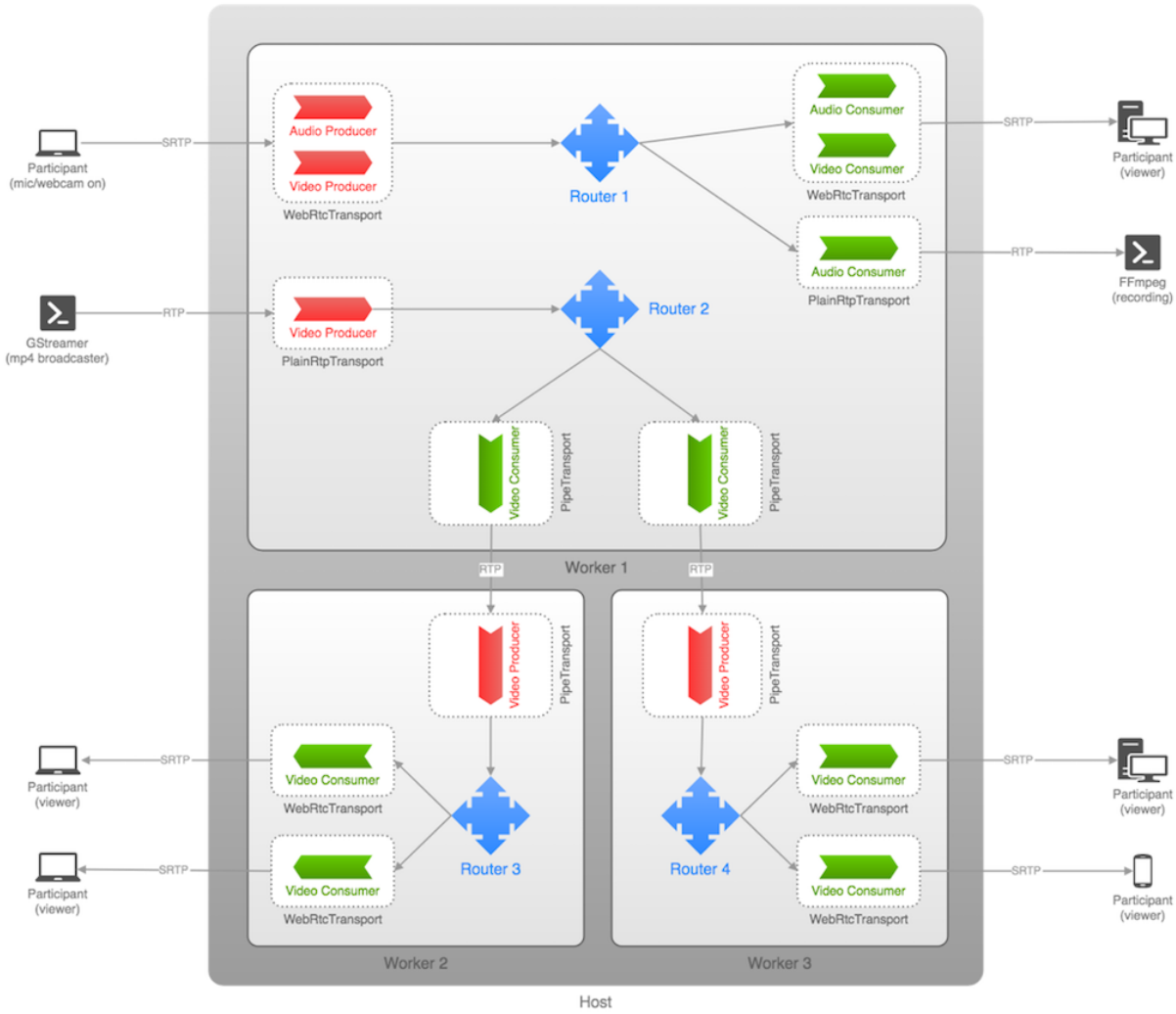
如果说 Janus 有什么缺点的话，那就是它的架构设计比较复杂，对于初学者来说难度较大。

Mediasoup

Mediasoup 是推出时间不长的 WebRTC 流媒体服务器开源库，其地址为：https://github.com/versatica/mediasoup/ 。

Mediasoup 由**应用层**和**数据处理层**组成。应用层是通过 Node.js 实现的；数据处理层由 C++ 语言实现，包括 DTLS 协议实现、ICE 协议实现、SRTP/SRTCP 协议实现、路由转发等。

下面我们来看一下 Mediasoup 的架构图，如下所示：



Mediasoup 架构图

Mediasoup 把每个实例称为一个 Worker，在 Worker 内部有多个 Router，每个 Router 相当于一个房间。在每个房间里可以有多个用户或称为参与人，每个参与人在 Mediasoup 中由一个 Transport 代理。换句话说，对于房间（Router）来说，Transport 就相当于一个用户。

Transport 有三种类型，即 WebRtcTransport、PlainRtpTransport 和 PipeTransport。

WebRtcTransport 用于与 WebRTC 类型的客户端进行连接，如浏览器。

PlainRtpTransport 用于与传统的 RTP 类型的客户端连接，通过该 Transport 可以播放多媒体文件、FFmpeg 的推流等。

PipeTransport 用于 Router 之间的连接，也就是一个房间中的音视频流通过 PipeTransport 传到另一个房间。

在每个 Transport 中可以包括多个 Producer 和 Consumer。

> Producer 表示媒体流的共享者，它又分为两种类型，即音频的共享者和视频的共享者。
>
> Consumer 表示媒体流的消费者，它也分为两种类型，即音频的消费者和视频的消费者。

Mediasoup 的实现逻辑非常清晰，它不关心上层应用该如何做，只关心底层数据的传输，并将它做到极致。

Mediasoup 底层使用 C++ 开发，使用 libuv 作为其异步 IO 事件处理库，所以保证了其性能的高效性。同时它支持了几乎所有 WebRTC 为了实时传输做的各种优化，所以说它是一个特别优秀的 WebRTC SFU 流媒体服务器。

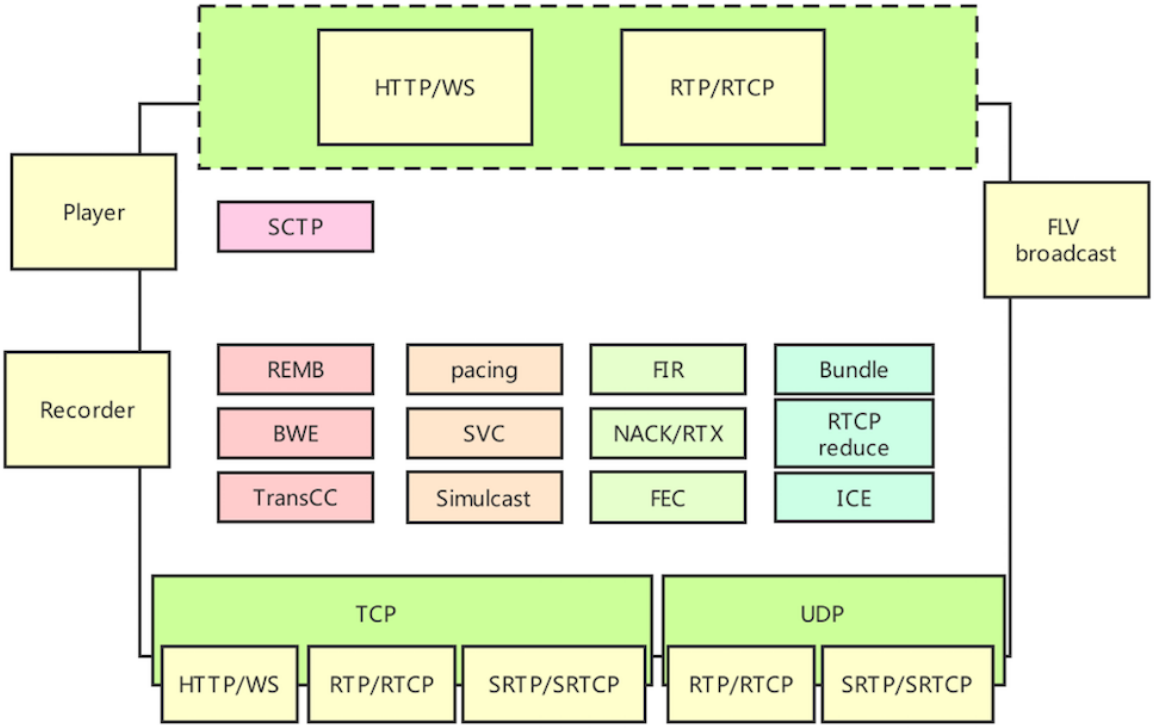
它与 Janus 相比，它更聚焦于数据传输的实时性、高效性、简洁性，而 Janus 相比 Mediasoup 做的事儿更多，架构和逻辑也更加复杂。所以**对于想学习 WebRTC 流媒体服务器源码的同学来说，Mediasoup 是一个非常不错的项目。**

另外，对于开发能力比较强的公司来说，根据自己的业务需要在 Mediasoup 上做二次开发也是非常值得推荐的技术方案。

Medooze

Medooze 是一款综合流媒体服务器，它不仅支持 WebRTC 协议栈，还支持很多其他协议，如 RTP、RTMP 等。其源码地址为：https://github.com/medooze/media-server 。

下面我们来看一下 Medooze 的架构图：



Medooze 架构图

从大的方面来讲，Medooze 支持 RTP/RTCP、SRTP/SRCP 等相关协议，从而可以实现与 WebRTC 终端进行互联。除此之外，Medooze 还可以接入 RTP 流、RTMP 流等，因此你可以使用 GStreamer/FFmpeg 向 Medooze 推流，这样进入到同一个房间的其他 WebRTC 终端就可以看到 / 听到由 GStream/FFmpeg 推送上来的音视频流了。另外，Medooze 还支持录制功能，即上图中的 Recorder 模块的作用，可以通过它将房间内的音视频流录制下来，以便后期回放。

为了提高多方通信的质量，Medooze 在音视频的内容上以及网络传输的质量上都做了大量优化。关于这些细节我们这里就不展开了，因为在后面的文章中我们还会对 Medooze 作进一步的讲解。

以上我们介绍的是 Medooze 的核心层，下面我们再来看看 Medooze 的控制逻辑层。Medooze 的控制逻辑层是通过 Node.js 实现的，Medooze 通过 Node.js 对外提供了完整的控制逻辑操作相关的 API，通过这些 API 你可以很容易的控制 Medooze 的行为了。

通过上面的介绍，我们可以知道 Medooze 与 Mediasoup 相比，两者在核心层实现的功能都差不多，但 Medooze 的功能更强大，包括了录制、推 RTMP 流、播放 FLV 文件等相关的操作，而 Mediasoup 则没有这些功能。

不过 Medooze 也有一些缺点，尽管 Medooze 也是 C++ 开发的流媒体服务务器，使用了异步 IO 事件处理机制，但它使用的异步 IO 事件处理的 API 是 poll，poll 在处理异步 IO 事件时，与 Linux 下最强劲的异步 IO 事件 API epoll 相比要逊色不少，这导致它在接收 / 发送音视频包时性能比 Mediasoup 要稍差一些。

如何选择 SFU

上面我们对 Licode、Janus、Mediasoup、Medooze 这几个开源的 WebRTC 流媒体服务器做了比较，从中你应该对它们的架构和特性有了一个基本了解。整体上看这些流媒体服务器各有优缺点，对于我们来说到底该选择那个项目作为我们的流媒体服务器呢?

其实，对流媒体服务器的选择是一个 Balance，没有最好，只有最合适。每个开源实现都有其各自的特点，都可以应用到实际产品中，只不过作为开发人员都有自己独特的技术背景，**你需要根据自身特点以及项目特点选一个最合适的。**接下来，我就介绍一下我是如何对这些开源项目进行评判和选择的。

在选择之前我会像上面一样对要选择的项目有个大体的了解，然后通过以下几个方面来对项目进行打分，从而选出最适合我的项目。

首先是实现语言。在一个团队中肯定会选择一种大家都比较熟悉的语言作为项目开发的语言，所以我们在选择开源项目时，就要选择使用这种语言开发的开源项目。比如阿里系基本都用 Java 语言进行开发，所以它们在选择开源项目时，基本都会选择 Java 开发的开源项目；而做音视频流媒体服务的开发人员，为了追求性能，所以一般都选择 C/C++ 语言开发的开源项目。

Meooze、Mediasoup、Licode 这三个流媒体服务器的媒体通信部分都是由 C++ 实现的，而控制逻辑是通过 Node.js 实现，因此如果你是 C++ 开发人员，且有 JavaScript 技术背景，那么你就应该在这三种流媒体服务器之间选择，因为这样更容易入门。而 Janus-gateway 是完全通过 C 语言实现的，服务部署是传统的 Linux 风格，因此如果你是 Linux/C 开发者，则应该选择 Janus 作为你的流媒体服务器。

其次是系统特点。像 Licode 是一个完整的系统，支持分布式集群部署，所以系统相对复杂，学习周期要长一些。它可以直接布署在生产环境，但是二次开发的灵活性不够。Janus-gateway 是一个独立的服务，支持的信令协议很丰富，而且支持插件开发，易扩展，对于 Linux/C 背景的开发者是很不错的选择。Medooze 和 Mediasoup 都是流媒体服务器库，对于需要将流媒体服务器集成到自己产品中的开发者来说，应该选择它们。

另外，从性能方面讲，Licode、Meooze、Mediasoup、Janus-gateway 单台服务都可以支持 500 方参会人，所以它们的性能都还是不错的。相对来说，Licode 的性能与其他流媒体服务器相比要低一些；Medooze 由于没有使用 epoll 来处理异步 IO 事件，所以性能也受到一些影响。不过总的来说，它们在 500 方的容量下，视频质量都可以得到很好的保证，延迟在 100ms 左右。

小结

本文我向你介绍了 Licode、Meooze、Mediasoup、Janus-gateway 四个开源 SFU 的架构及其特点。

从 SFU 模型来看，它们的实现基本都类似的，只是概念叫法不同，它们之间的差异主要体现在系统对外提供的接口上。对于想要研究 SFU 基本原理的开发者来说，只要选择其中一个项目仔细研读即可。但我个人还是建议初学者选择 Mediasoup 作为研究 SFU 原理的对象，因为它结构简单，代码量少，很容易入门。当然如果你是一个纯 C 技术背景的开发者，可以选择 Janus-gateway 作为研究对象。

如果说还需要研究录制回放、MCU 的开发者，可以去研究 Medooze。Medooze 结构比较完整，功能齐全，代码量也不大。而如果你想要让自己的媒体服务器支持集群部署，可以参考 Licode。

思考时间

如果让你自己实现一个 SFU 流媒体服务器，你会怎么设计？你认为都有哪些问题是实现 SFU 的关键呢？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

上一篇　24 | 多人音视频实时通讯是怎样的架构?

下一篇　26 | 为什么编译Medooze Server这么难?

精选留言 (4)

写留言

三角形小于零
2019-09-11

老师，后面会讲 webrtc 服务端的压测相关的内容吗

展开

作者回复: 这个专栏不会讲，压测这个会在专门讲服务器的课中才会讲

三角形小于零
2019-09-11

老师，后面会讲 turn server吗

展开

作者回复: turn服务是 1对1的，那个没啥要讲的呀！你是想实现多对多吧？如果是这样的话，后面几篇文章是专门讲这个的。

牛景波
2019-09-10

sfu核心就是简单和快，转发上用共享内存，无锁队列，消息队列是不能用了；收发包可以用dpdk加速，降低cpu消耗

展开

作者回复: 不错！在 Licode中虽然你看到它用了 RabbitMQ, 但它是信令消息才会用，对于流媒体数据它是不会使用 MQ的哈

周伟民
2019-09-10

老师，你好。RTSP协议在安防监控领域应用很广泛，而且RTSP协议用UDP负载RTP包，实时性比较好。与RTMP/HLS相比，为什么RTSP协议在互联网领域应用很少呢？

作者回复: 因为 RTMP/HLS 底层用的是 TCP 协议，这样就不用考虑丢包情况了。

RTSP 数据是用的 UDP，控制是它自己的一套，实际上与 SIP协议很类似的。但现在大家更多的是使用 WebRTC， RTSP 和 SIP 用的都比较少，只是在专有领域才会用。