

28 | 让我们一起探索Medooze的具体实现吧（下）

2019-09-17 李超

从0打造音视频直播系统

[进入课程 >](#)



讲述：李超

时长 22:44 大小 20.83M



在[上一篇文章](#)中，我向你介绍了 Medooze 的 SFU 模型、录制回放模型以及推流模型，并且还向你展示了 Medooze 的架构，以及 Medooze 核心组件的基本功能等相关方面的知识。通过这些内容，你现在应该已经对 Medooze 有了一个初步了解了。

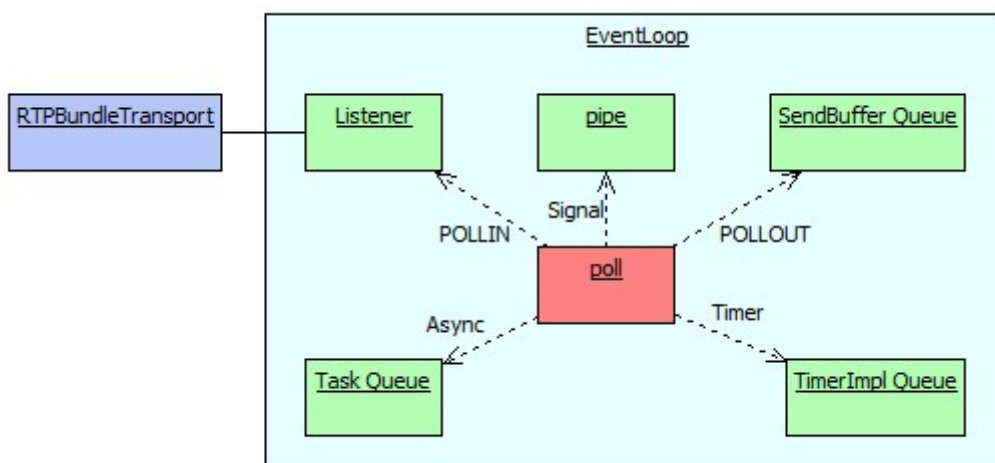
不过，那些知识我们是从静态的角度讲解的 Medooze，而本文我们再从动态的角度来分析一下它。讲解的主要内容包括：WebRTC 终端是如何与 Medooze 建立连接的、数据流是如何流转的，以及 Medooze 是如何进行异步 I/O 事件处理的。

异步 I/O 事件模型

异步 I/O 事件是什么概念呢？你可以把它理解为一个引擎或动力源，这里你可以类比汽车系统来更好地理解这个概念。

汽车的动力源是发动机，发动机供油后会产生动力，然后通过传动系统带动行车系统。同时，发动机会驱动发电机产生电能给蓄电池充电。汽车启动的时候，蓄电池会驱动起动机进行点火。

这里的汽车系统就是一种事件驱动模型，发动机是事件源，驱动整车运行。实际上，Medooze 的异步 I/O 事件模型与汽车系统的模型是很类似的。那接下来，我们就看一下 Medooze 中的异步 I/O 事件驱动机制：



Medooze 事件驱动机制图

上面这张图就是 Modooze 异步 I/O 事件处理机制的整体模型图，通过这张图你可以发现 **poll 就是整个系统的核心**。如果说 EventLoop 是 Medooze 的发动机，那么 EventLoop 中的 poll 就是汽缸，是动力的发源地。


下面我就向你介绍一下 **poll 的基本工作原理**。在 Linux 系统中，无论是对磁盘的操作还是对网络的操作，它们都统一使用文件描述符 fd 来表示，一个 fd 既可以是一个文件句柄，也可以是一个网络 socket 的句柄。换句话说，我们只要拿到这个句柄就可以对文件或 socket 进行操作了，比如往 fd 中写数据或者从 fd 中读数据。

对于 poll 来说，它可以对一组 fd 进行监听，监听这些 fd 是否有事件发生，实际上就是监听 fd 是否可**写数据**和**读数据**。当然，具体是监听读事件还是写事件或其他什么事件，是需要你告诉 poll 的，这样 poll 一旦收到对应的事件时，就会通知你该事件已经来了，你可以做你想要做的事儿了。

那具体该怎么做呢？我们将要监听的 fd 设置好监听的事件后，交给 poll，同时还可以给它设置一个超时时间，poll 就开始工作了。当有事件发生的时候，poll 的调用线程就会被唤醒，poll 重新得到执行，并返回执行结果。此时，我们可以根据 poll 的返回值，做出相应的处理了。

需要说明的是，poll API 本身是一个同步系统调用，当没有要监听的事件（I/O 事件和 timer 事件）发生的时候，poll 的调用线程就会被系统阻塞住，并让它处于休眠状态，而它处理的 fd 是异步调用，这两者一定不要弄混了。

poll 的功能类似 select，其 API 原型如下：


 复制代码

```
1 include <poll.h>
2 int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

其各个参数及返回值含义如下表所示：

输入参数	说明	备注
fds	监听的 pollfd 类型的数组	需要创建一个 pollfd 类型的数组，将每一个要监听的 fd 赋值给相应的数组元素，必须要设置 events 参数。
nfds	fds 数组长度	
timeout	超时事件间隔	单位是毫秒(ms)。 <ol style="list-style-type: none"> 1. timeout > 0，表示 timeout 毫秒后，poll 会返回； 2. timeout == 0，表示 poll 立即返回； 3. timeout < 0，表示 poll 不受超时时间控制。
返回值	<ol style="list-style-type: none"> 1. 返回值 > 0，表示产生 revents 的 fd 个数； 2. 返回值 == 0，表示超时事件发生，没有 fd 产生 revents； 3. 返回值 < 0，表示 poll 调用发生了错误，具体错误原因要参考 errno 变量的值。 	

其中第一个 **fds**，它表示的是监听的 **pollfd** 类型的数组，pollfd 类型结构如下：

 复制代码

```
1 struct pollfd {
2     int    fd;           / file descriptor /
3     short events;        / requested events /
4     short revents;       / returned events /
5 };
```

fd 是要监听的文件描述符。

events 是一个输入参数，表明需要监听的事件集合。

revents 是输出参数，表明函数返回后，在此 fd 上发生的事件的集合。

下表是常见事件的说明：

事件类型	说明	备注
POLLIN	表示 fd 有读事件发生	对于 Socket 来说，就是收到了对方的数据包。
POLLOUT	表示 fd 有可写事件发生	1. 对于 UDP Socket 来说，只要去监听 POLLOUT 事件，poll 马上会返回。 2. 对于 TCP Socket 来说，为了保证传输的可靠性，发送端会有 Send buffer，只有收到接收端的 ACK，发送的窗口才会滑动。如果因为网络原因导致了发送的数据包得不到 ACK，那么发送会出现 EAGAIN 现象，此时就不能发送，就没有 POLLOUT 事件。等到发送窗口滑动，发送端的 Send buffer 有空闲的时候，会产生 POLLOUT 事件。
POLLHUP	连接被对方关闭	这种关闭是正常关闭。调用 recv 会返回0。
POLLERR	Socket 产生了网络错误	比如连接异常断开。

了解了 poll API 各参数和相关事件的含义之后，接下来我们看一下当 poll 睡眠时，在哪些情况下是可以被唤醒的。poll 被唤醒的两个条件：

- 1. 某个文件描述符有事件发生时；
- 2. 超时事件发生。

以上就是异步 I/O 事件处理的原理以及 poll API 的讲解与使用。接下来我们再来看看 EventLoop 模块中其他几个类的功能及其说明：

pipe，好比是火花塞，用来唤醒 poll 的。比如 poll 线程被阻塞，我们又想让它随时可以唤起，就可以通过让 poll 监听一个 pipe 来实现。

Listener，数据包处理接口。只要实现了 Listener 接口，当 fd 接收到数据包以后，就可以调用此接口来处理。

SendBuffer，Queue 数据包发送队列。当有数据包需要发送给客户端的时候，首先会发送到 EventLoop 的发送队列，然后 EventLoop 就会根据 fd 的 POLLOUT 事件进行分发。

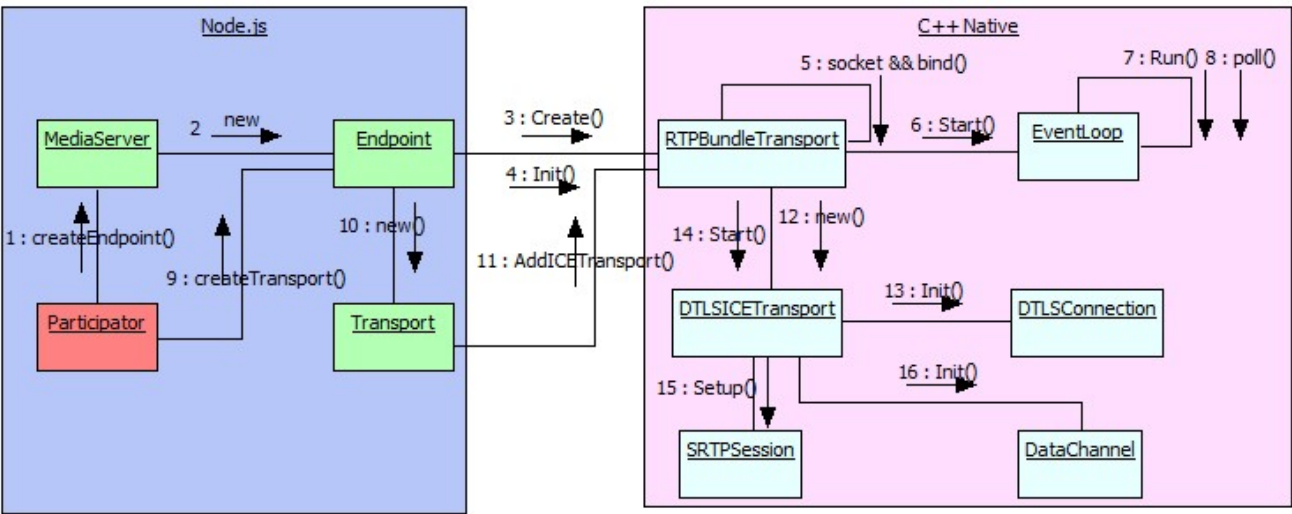
Task Queue，异步任务队列。当有逻辑需要 EventLoop 线程执行的的时候，通过调用 Async 函数将异步任务放入异步任务队列中，然后通过 Signal 唤醒 poll 去处理此异步任务。

TimerImpl Queue，定时器队列。需要定时器功能的时候，可以调用 CreateTimer/CancelTimer 去创建 / 取消一个定时器。

RTPBundleTransport，实现了 Listener 接口，所有接收到的 UDP 数据包都由此类处理。

ICE 连接建立过程

了解了 Medooze 的异步 I/O 事件处理机制后，接下来我们看一下 ICE 连接的建立。实际上，传输层各个组件的基本功能，我们在[上篇文章](#)中都已一一介绍过了，不过那都是静态的，本文我们从动态的角度，进一步了解一下 DTLS 连接的建立过程。



DTLS 连接建立过程图

当某个参与人 (Participator) 加入房间后, 需要建立一个 DTLS 连接使客户端与 Medooze 可以进行通信, 上图描述了各个对象实例之间的先后调用关系。其具体连接建立过程说明如下 (步骤较多, 建议对照上图来理解和学习) :

1. 当有参与人 (Participator) 加入时, 如果此时房间内只有一个人, 则该 Participator 对象会调用 MediaServer::createEndpoint 接口, 来创建一个接入点。
2. MediaServer 对象会创建一个 Endpoint 实例。
3. Endpoint 对象会创建一个 Native RTPBundleTransport 实例。
4. Native RTPBundleTransport 实例创建好后, Endpoint 再调用 RTPBundleTransport 实例的 init() 方法对 RTPBundleTransport 实例进行初始化。
5. 在 RTPBundleTransport 初始化过程中, 会创建 UDP socket, 动态绑定一个端口。这个动态端口的范围, 用户是可以指定的。
6. RTPBundleTransport 中包含了一个 EventLoop 实例, 所以会创建事件循环实例, 初始化 pipe。
7. EventLoop 会创建一个事件循环线程。
8. 事件循环线程中会执行 poll, 进入事件循环逻辑。**此时, Endpoint 的初始过程算是完成了。**
9. Participator 需要创建一个 **DTLS 连接**, DTLS 连接是由 Endpoint::createTransport 函数来创建的。
10. Endpoint 创建一个 Transport 实例, 此实例是 Native 的一个 wrapper。
11. Transport 调用 RTPBundleTransport::AddICETransport 接口, 准备创建 Native DTLS 实例。
12. RTPBundleTransport 创建一个 DTLSICETransport 实例。
13. DTLSICETransport 聚合了 DTLSConnection 实例, 对其进行初始化。主要工作就是证书的生成、证书的应用、ssl 连接实例的创建, 以及 DTLS 的握手过程。
14. DTLSICETransport 的初始化工作, 包括 SRTPSession 的初始化和 DataChannel 的初始化。

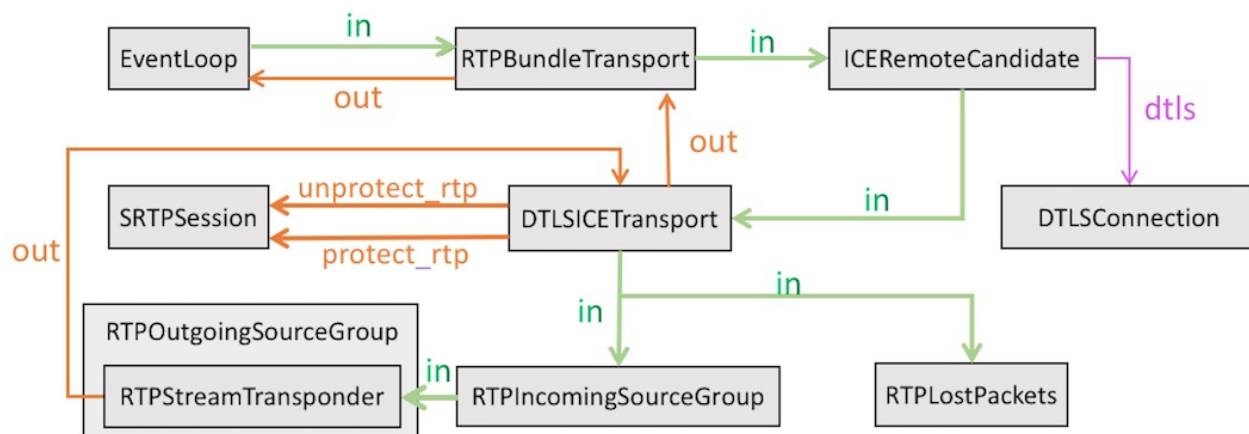
经历这么一个过程, ICE 连接就建立好了, 剩下就是媒体流的转发了。

SFU 数据包收发过程

当终端和 Medooze 建立好 DTLSICETransport 连接以后, 对于需要共享媒体的终端来说, Medooze 会在服务端为它建立一个与之对应的 IncomingStream 实例, 这样终端发送来的音视频流就进入到 IncomingStream 中。

如果其他终端想观看共享终端的媒体流时，Medooze 会为观看终端创建一个 OutgoingStream 实例，并且将此实例与共享者的 IncomingStream 实例绑定到一起，这样从 IncomingStream 中收到的音视频流就被源源不断输送给了 OutgoingStream。然后 OutgoingStream 又会将音视频流通过 DTLSICETransport 对象发送给观看终端，这样就实现了媒体流的转发。

RTP 数据包在 Medooze 中的详细流转过程大致如下图所示：



SFU 数据转发图

图中带有 in 标签的绿色线条表示共享者数据包流入的方向，带有 out 标签的深红色线条表示经过 Medooze 分发以后转发给观看者的数据流出方向，带有 protect_rtp/unprotect_rtp 标签的红色线条表示 RTP 数据包的加密、解密过程，带有 dtls 标签的紫色线条表示 DTLS 协议的协商过程以及 DTLS 消息的相关处理逻辑。

上面的内容有以下四点需要你注意一下：

如果采用 DTLS 协议进行网络传输，那么在连接建立的过程中会有一个 DTLS 握手的过程，这也是需要数据包收发处理的。所以，我们上图中的数据包流转不仅是音视频数据，还包括了 DTLS 握手数据包。

DTLSConnection 是专门处理 DTLS 协议握手过程的，由于协议复杂，这里就不做详细介绍了。

SRTPSession 是对音视频数据进行加密、解密的，这是 API 的调用，所以图里并没有流入、流出的过程说明。

RTPLostPackets 主要是进行网络丢包统计。

接下来，我们就对数据包的流转过程做一个简要的说明（步骤较多，建议对照上图来理解和学习）：

共享终端的音视频数据包到达 Medooze 服务器以后，是被 EventLoop 模块接收，接收到数据包不做处理，直接传给 RTPBundleTransport 模块。需要说明的是，数据包的收发最终都是落入到了 EventLoop 模块，因为数据包的收发都需要处理网络事件。

前面讲过，RTPBundleTransport 模块就像一个容器，管理着所有参会终端的网络连接实例。当数据包到达 RTPBundleTransport 模块以后，它会根据数据包的来源，将数据包分发给相应的 ICERemoteCandidate 实例。ICERemoteCandidate 是针对每一个通信终端建立的一个逻辑连接。同样，收发也都需要经过 RTPBundleTransport 模块，为此模块绑定了 UDP socket。

ICERemoteCandidate 会对数据包的类型进行检查，如果是 DTLS 数据包，就会转给 DTLSConnection 模块，处理 DTLS 协商过程；如果是音视频数据，那么会转给 DTLSICETransport 模块。

在 DTLSICETransport 处理模块中，它首先会调用 SRTPSession 的 unprotect_rtp 函数进行解密；然后，调用 RTPLostPackets 的 API 处理丢包逻辑；最后，将数据包传给 RTPIncomingSourceGroup 模块。

RTPIncomingSourceGroup 会遍历所有订阅此流的观看终端，然后将数据流转发给相应的 RTPOutgoingSourceGroup 实例。

RTPOutgoingSourceGroup 模块对于下行流的处理，是通过 RTPStreamTransponder 模块进行 RTCP 相关逻辑处理，然后传给 DTLSICETransport 模块。

DTLSICETransport 模块会调用 SRTPSession 模块的 protect_rtp 函数，对下行数据包进行加密，然后传给 RTPBundleTransport 模块。

RTPBundleTransport 模块没有任何逻辑处理，直接将数据包传给了 EventLoop 模块。EventLoop 是提供了一个下行发送队列，其实是将数据保存到发送队列中。

EventLoop 从发送队列中获取待发送的数据包并发送给观看端。

小结

本文我们重点介绍了事件驱动机制，它像人的心脏一样，在现代服务器中起着至关重要的作用。Medooze 服务器中的具体实现模块就是 EventLoop 模块。通过本文的学习，我相信

你对事件驱动机制有了一定的了解。接下来，就需要结合代码做进一步的学习，最好能进行相应的项目实践，这样对你继续研读 Medooze 源码会有很大的帮助。

后半部分，我们还分别介绍了 DTLS 连接建立过程和数据包的流转过程。通过本文以及上一篇文章的学习，我相信你对 Medooze 的基础架构以及 SFU 的实现细节已经有了深刻的理解。

思考时间

今天你的思考题是：异步 I/O 事件中，什么进候会触发写事件？

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。



从 0 打造音视频直播系统

手把手教你打造实时互动音视频直播系统

李超

新东方音视频直播技术专家
前沪江音视频架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 27 | 让我们一起探索Medooze的具体实现吧（上）

下一篇 29 | 如何使用Medooze 实现多方视频会议？

精选留言 (4)

写留言



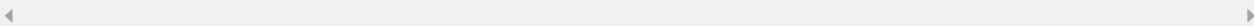
Joseph

2019-09-18

老师介绍的非常仔细，谢谢老师！

展开 ▾

作者回复: 谢谢！



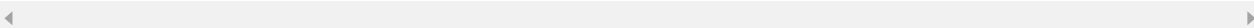
frank

2019-09-17

重点是介绍libmediaserver.a? mcu不介绍吗？

展开 ▾

作者回复: 这是medooze的核心，你可以用它实现SFU,也可以用它来实现 MCU，现在MCU应用的地方很少

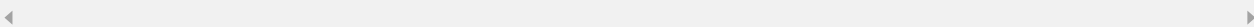


周龙亭

2019-09-17

DTLS连接建立过程中，WebRTC客户端是怎么参与的？

作者回复: 在第 30 篇文章中有介绍



良师益友

2019-09-17

为什么不用epoll呢？

展开 ▾

作者回复: 这是需要优化的点！

