

22 | 如何保证数据传输的安全（下）？

2019-09-03 李超

从0打造音视频直播系统

[进入课程 >](#)



讲述：李超

时长 17:48 大小 24.46M



在[上一篇文章](#)中，我向你介绍了保证数据传输安全的一些**基本概念**，如非对称加密、数字证书等等。那本文我们就结合这些基本概念，来一起看看 **WebRTC 是如何保障数据安全的**。

WebRTC 的一个重要应用就是在浏览器上实现音视频实时通信，正如[上一篇文章](#)中所讲到的，在浏览器上由于用户量巨大，所以对于音视频的通信必须要有一套机制来保证音视频数据的安全性。

实际上，在浏览器上对于音视频数据的安全最终还是由 WebRTC 库自己保证的。它是通过使用 SDP、STUN、DTLS、SRTP 等几个协议的结合来达到数据安全的。

WebRTC 数据安全机制

为了保障音频数据的安全，WebRTC 使用了一整套机制来进行保护，下面我们就来看一下 WebRTC 是如何保障数据安全的吧！

我们来假设一个场景，A 与 B 通信，为了保障数据的安全，我们只需要在 A 端发送数据前将要发送的数据进行加密，B 端收到数据之后再进行解密就好了，这样看起来保证数据的安全还是蛮简单的事情。

但这里有一个问题，**B 端是如何知道 A 端使用的哪种加密算法进行加密的呢？**另外，加密算法还分对称加密和非对称加密，我们应该选择哪个呢？实际上在[上一篇文章](#)中我已经向你做了介绍，对于加密来说，使用非对称加密是最安全的，因此**选择非对称加密是必然的选择。**

既然选择非对称加密，那么 A 端与 B 端就要交换各自的公钥，这样当 A 端使用私钥加密时，B 端才能用 A 的公钥进行解密。同样的道理，B 端使用自己的私钥进行加密时，A 端可以使用 B 端的公钥进行解密。

按照上面的描述，你会发现其逻辑上有个安全漏洞，即 A 与 B 交换公钥时，并没有进行任何防护。黑客完全可以用各种手段在 A 与 B 交换公钥时获取到这些公钥，这样他们就可以轻而易举地将传输的音视频数据进行解密了。

为了解决这个问题，WebRTC 引入了 **DTLS** (Datagram Transport Layer Security)，至于 DTLS 的实现细节，你暂时不用关心，后面我们会对它做详细的讲解。你现在只要知道**通过 DTLS 协议就可以有效地解决 A 与 B 之间交换公钥时可能被窃取的问题就好了。**

A 与 B 交换公钥时被窃取的问题解决后，是不是双方通信就安全了呢？

我们再来想像一个场景，还是 A 与 B 通信，但此时 B 并不是真正的 B，而是冒充的，这样 A 与 B 通信时，冒充的 B 就获得了 A 的重要信息。其实这种情况更多发生在会议系统或在线教育的小班课中，此时会议中有多人进行互动，如果黑客进入了会议中，他只需听别人说话，自己不发言，这样就将关键的信息窃取走了。所以现在的问题又来了，我们该**如何辨别对方的身份是否合法呢？**

看到辨别身份的问题是不是似曾相识？在[上一篇文章](#)中我向你介绍过通过数字签名的方式可以防止内容被篡改。WebRTC 也是使用的这种方式，它首先通过信令服务器交换 SDP，SDP 信息中包括了以下几个重要信息：

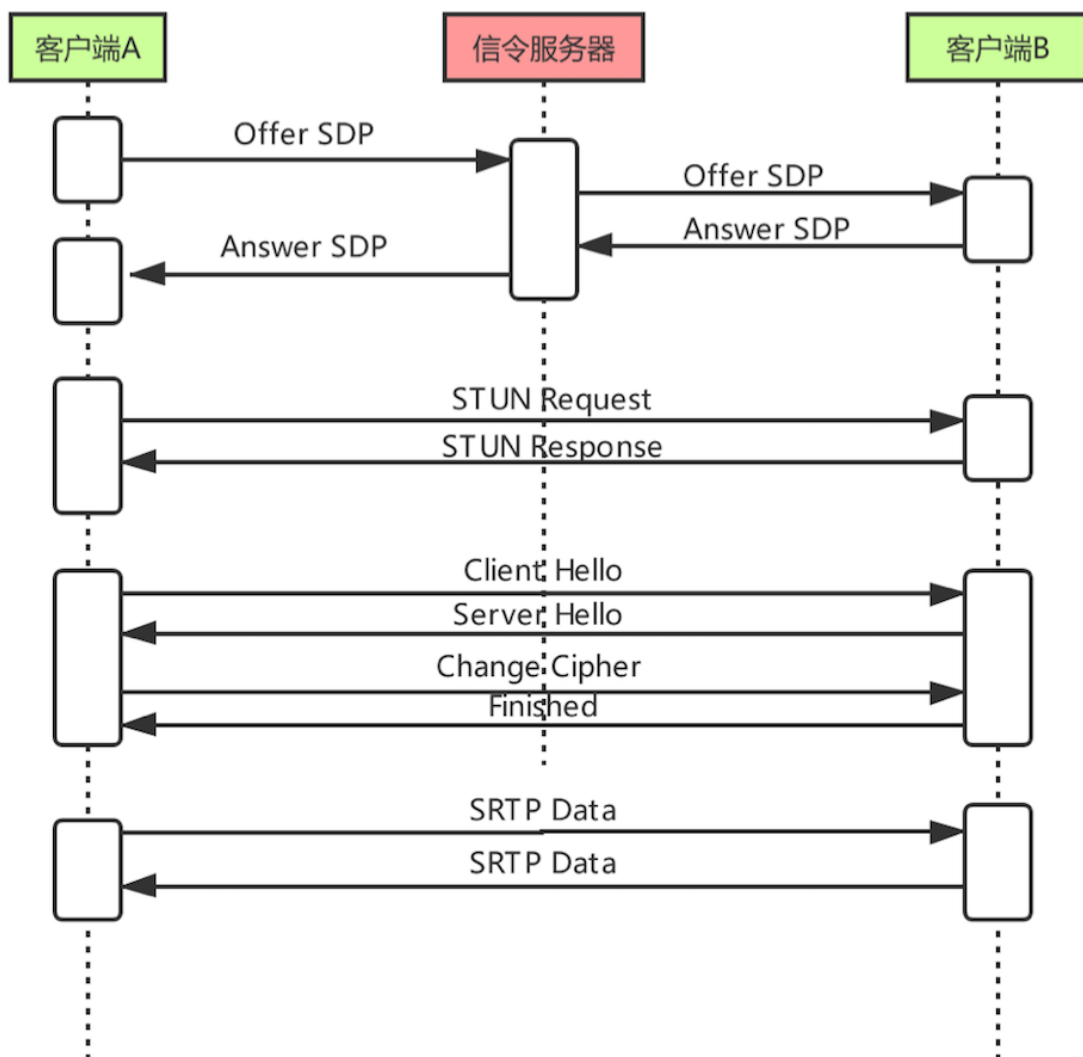
```
1 ...  
2 a=ice-ufrag:khLS  
3 a=ice-pwd:cxlzteJaJBou3DspNaPsJh1Q  
4 a=fingerprint:sha-256 FA:14:42:3B:C7:97:1B:E8:AE:0C2:71:03:05:05:16:8F:B9:C7:98:E9:60:4:  
5 ...
```

SDP 交换完成后，A 与 B 都获取到了对方的 ice-ufrag、ice-pwd 和 fingerprint 信息，有了这些信息后，就可验证对方是否是一个合法用户了。

其中，**ice-ufrag 和 ice-pwd 是用户名和密码**。当 A 与 B 建立连接时，A 要带着它的用户名和密码过来，此时 B 端就可以通过验证 A 带来的用户名和密码与 SDP 中的用户名和密码是否一致的，来判断 A 是否是一个合法用户了。

除此之外，**fingerprint**也是验证合法性的关键一步，它是存放公钥证书的**指纹（或叫信息摘要）**，在通过 ice-ufrag 和 ice-pwd 验证用户的合法性之余，还要对它发送的证书做验证，看看证书在传输的过程中是否被篡改了。

通过上面的描述你就可以知道 WebRTC 在数据安全方面做了非常多少努力了。下面的序列图就清楚地表述了我上面所讲述的内容。



WebRTC 安全序列图

从这张图中你可以看到，A 与 B 在传输数据之前，需要经历如下几个步骤。

首先通过信令服务器交换 SDP 信息，也就是进行媒体协商。在 SDP 中记录了用户的用户名、密码和指纹，有了这些信息就可以对用户的身份进行确认了。

紧接着，A 通过 STUN 协议（底层使用 UDP 协议）进行身份认证。如果 STUN 消息中的用户名和密码与交换的 SDP 中的用户名和密码一致，则说明是合法用户。

确认用户为合法用户后，则需要进行 DTLS 协商，交换公钥证书并协商密码相关的信息。同时还要通过 fingerprint 对证书进行验证，确认其没有在传输中被篡改。

最后，再使用协商后的密码信息和公钥对数据进行加密，开始传输音视频数据。

以上就是 WebRTC 保证数据安全的整套机制。

前面我们说了 WebRTC 是通过使用 DTLS、SRTP 等几个协议的结合来达到数据安全的，那接下来我们就来分别看一下这几个协议是如何实现的。

DTLS 协议

说到网络上的数据安全你可能首先想到的是 HTTPS，你也可以简单地将 HTTPS 理解为“HTTP 协议 + 数据加密”，当然实际上它要复杂得多。HTTPS 的底层最初是使用 SSL (Secure Sockets Layer, 安全套接层) 协议对数据加密。当 SSL 更新到 3.0 时，IETF 对 SSL 3.0 进行了标准化，并增加了一些新的功能，不过基本与 SSL 3.0 没什么区别，标准化后的 SSL 更名为 TLS 1.0 (Transport Layer Security, 安全传输层协议)，所以可以说 TLS 1.0 就是 SSL 的 3.1 版本。

TLS 协议由**TLS 记录协议**和**TLS 握手协议**组成：

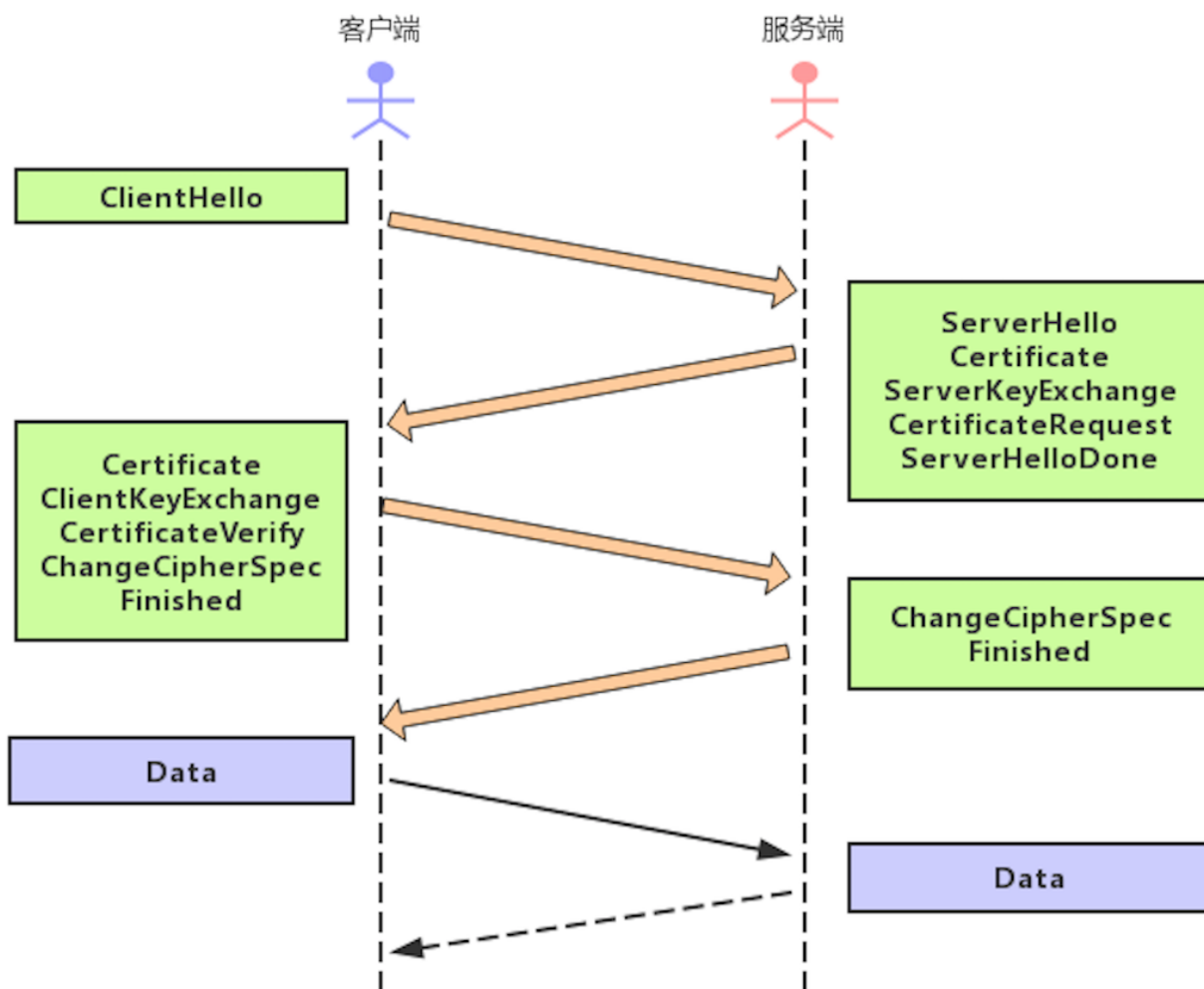
TLS 记录协议，用于数据的加密、数据完整性检测等；

TLS 握手协议，主要用于密钥的交换与身份的确认。

由于 TLS 底层是基于 TCP 协议的，而 WebRTC 音视频数据的传输主要基于 UDP 协议，因此 WebRTC 对数据的保护无法直接使用 TLS 协议。但 TLS 协议在数据安全方面做得确实非常完善，所以人们就想到是否可以将 TLS 协议移植到 UDP 协议上呢？因此 DTLS 就应运而生了。

所以你可以认为**DTLS 就是运行在 UDP 协议之上的简化版本的 TLS**，它使用的安全机制与 TLS 几乎一模一样。

在 DTLS 协议中，最关键的是它的**握手协议**，正如下图所展示的这样：



DTLS 握手协议示意图

在 WebRTC 中为了更有效地保护音视频数据，所以需要使用 DTLS 协议交换公钥证书，并确认使用的密码算法，这个过程在 DTLS 协议中称为**握手协议**。

DTLS 的握手过程如下：

首先 DTLS 协议采用 C/S 模式进行通信，其中发起请求的一端为客户端，接收请求的为服务端。

客户端向服务端发送 `ClientHello` 消息，服务端收到请求后，回 `ServerHello` 消息，并将自己的证书发送给客户端，同时请求客户端证书。

客户端收到证书后，将自己的证书发给服务端，并让服务端确认加密算法。

服务端确认加密算法后，发送 `Finished` 消息，至此握手结束。

DTLS 握手结束之后，通信双方就可以开始相互发送音视频数据了。

OpenSSL 库

讲到数据安全就不得不提 OpenSSL 库，通过它的名字你也基本可以知道它是做什么的。OpenSSL 是一个开源的 SSL 实现，正如我们上面说到的，SSL 是 TLS 早期的名字，实际上 OpenSSL 实现了整个 TLS 协议。

不仅如此，OpenSSL 还实现了 DTLS 协议，由于其代码开源，实现得又特别高效，所以现在大部分需要数据安全的应用程序基本上都是使用 OpenSSL 来实现的。

关于 OpenSSL 库，有以下几个基本概念你一定要清楚。

SSL_CTX: SSL 上下文，主要指明你要使用的 SSL 版本是多少。


SSL: 代表一个 SSL 连接，你可以把它看作是一个句柄，一般还要与一个具体的 socket 进行绑定。

SSL_Write: 用于向 SSL 连接写数据。

SSL_Read: 用于从 SSL 连接读数据。


那 OpenSSL 到底该如何使用呢？其实整体还是蛮简单的，下面我们就来看一下如何使用 OpenSSL，具体步骤可阐述为如下。

第一步，初始化 SSL。在这一步调用 `SSL_library_init()` 初始化 OpenSSL 库，然后加载 OpenSSL 支持的所有算法，以及相关的错误信息。

 复制代码


```
1 SSL_library_init()
2 OpenSSL_add_all_algorithms() /* 载入 openssl 所支持的算法 */
3 SSL_load_error_strings() /* 载入 openssl 的相关错误信息 */
```

第二步，创建 SSL 上下文。在这一步可以指定使用的 SSL 协议是哪个版本的。

 复制代码


```
1 SSL_CTX * ctx = SSL_CTX_new(SSLv23_server_method())
```


第三步，加载证书。如下所示，该函数的第一个参数是 SSL 上下文，第二个参数是要加载的证书，第三个参数为证书类型。

 复制代码


```
1 SSL_CTX_use_certificate_file(ctx, certificate.crt, SSL_FILETYPE_PEM)
```

第四步，加载私钥。在这一步可以先将私钥加载进来，然后再检测私钥是否正确。

 复制代码

```
1 SSL_CTX_use_PrivateKey_file(ctx, prikey.pem, SSL_FILETYPE_PEM)
2 SSL_CTX_check_private_key(ctx) /* 检查私钥是否正确 */
```

第五步，建立 SSL 并与 Socket 绑定。在这一步，首先通过 SSL 上下文创建 SSL 对象；然后，将 SSL 对象与已经创建好的 socket 进行绑定；最后是建立 SSL 连接。

 复制代码

```
1 SSL *ssl = SSL_new(ctx) /* 创建 SSL 用于通信 */
2 SSL_set_fd(ssl, socket_fd) /* 与 socket 绑定 */
3 SSL_accept(ssl) /* 建立 SSL 连接 */
```

第六步，使用 SSL 进行数据通信。主要通过 SSL_write 和 SSL_read 发送和接收数据。

 复制代码

```
1 SSL_write(ssl, buf, strlen(buf)) /* 向 ssl 发数据，消息通过 SSL 加密 */
2 SSL_read(ssl, buf, MAXBUF) /* 从 ssl 接收消息 */
```

第七步，释放资源。当 SSL 使用完后，需要将占用的资源全部释放掉，怎么实现呢？首先将 SSL 连接关掉，然后释放 SSL 对象，最后释放 SSL 上下文。


```
1 SSL_shutdown(ssl) /* 关闭 SSL 连接 */
2 SSL_free(ssl) /* 释放 SSL */
3 SSL_CTX_free(ctx) /* 释放 CTX */
```

以上这七步就是使用 OpenSSL 的基本步骤。为了更好地理解和应用 OpenSSL，你熟悉完这每一个步骤后，最好可以自己再将它们串联起来，形成自己的知识体系。

SRTP/SRTCP 协议

在《[06 | WebRTC 中的 RTP 及 RTCP 详解](#)》一文中我向你详细介绍了 RTP/RTCP 协议，通过该文你可以了解到，RTP/RTCP 协议并没有对它的负载数据进行任何保护。因此，如果你通过抓包工具，如 Wireshark，将音视频数据抓取到后，通过该工具就可以直接将音视频流播放出来，这是非常恐怖的事情。

在 WebRTC 中，为了防止这类事情发生，没有直接使用 RTP/RTCP 协议，而是使用了 SRTP/SRTCP 协议，即安全的 RTP/RTCP 协议。

WebRTC 使用了非常有名的 libsrtp 库将原来的 RTP/RTCP 协议数据转换成 SRTP/SRTCP 协议数据。libsrtp 的使用非常简单，具体步骤可总结为如下。

第一步，初始化 libsrtp。

```
1 srtp_init();
```

第二步，创建 Session。创建 Session 要略微复杂一些，这过程中需要指定创建的策略，比如使用哪种算法进行内容的完整性检测，解码时的公钥是什么，等等。


```
1 ...
2 srtp_policy_t policy;
3
4 // Set all policy fields to 0.
5 std::memset(&policy, 0, sizeof(srtp_policy_t)); // 清空结构体
```

```

6
7 // 指定用哪种算法进行内容的完整性检测
8 switch (profile){
9     case Profile::AES_CM_128_HMAC_SHA1_80:
10         srtp_crypto_policy_set_aes_cm_128_hmac_sha1_80(&policy.rtp);          srtp_crypto_po
11         break;
12
13     case Profile::AES_CM_128_HMAC_SHA1_32:
14         srtp_crypto_policy_set_aes_cm_128_hmac_sha1_32(&policy.rtp);
15         srtp_crypto_policy_set_aes_cm_128_hmac_sha1_80(&policy.rtcp); // NOTE: Must be 80 f
16         break;
17
18 default:
19     ...
20 }
21
22 policy.ssrc.value = 0;
23 policy.key        = key; // 指定解码时的公钥
24
25 // Required for sending RTP retransmission without RTX.
26 policy.allow_repeat_tx = 1;    // 使用 RTX 进行 RTP 包重传
27 policy.window_size    = 1024; // 窗口缓冲区大小
28 policy.next           = nullptr;
29
30 // Set the SRTP session.
31 srtp_err_status_t err = srtp_create(&this->session, &policy);
32 ...

```

第三步，对 RTP 包加密。如下面示例代码所示：

 复制代码


```

1 srtp_err_status_t err = srtp_protect(this->session,
2                                     (void*)EncryptBuffer,
3                                     reinterpret_cast<int*>(len));

```


第一个参数是 Session，第二个参数是要加密的数据，第三个参数是被加密数据的长度。需要注意的是，加密后的数据也存放在 EncryptBuffer，即输入数据与输出数据共用同一块内容。

第四步，对 SRTP 包解密。这同上面的第三步类似，只不过操作相反，这里变成了解密。

 复制代码

```
1 srtp_err_status_t err = srtp_unprotect(this->session,  
2                                     (void*)data,  
3                                     reinterpret_cast<int*>(len));
```

第五步，也就是最后一步，是释放资源。

 复制代码

```
1 srtp_shutdown();
```

以上这五步就是使用 libsrtp 对 RTP 数据加密 / 解密的基本步骤。

小结

本文首先向你介绍了 WebRTC 为了保护音视频数据的安全性，提供的是怎样一整套安全机制，然后又对这一套安全机制中的几个重点概念做了详细讲解，如 DTLS 协议、OpenSSL 库的使用以及如何将 RTP/RTCP 数据转成 SRTP/SRTCP 数据。

在介绍 DTLS 协议时，我们重点讲解了它是如何进行握手操作的，这里你只需要了解 DTLS 握手的基本原理即可，不用对细节进行特别的追究，因为握手的所有操作实际上都是由 OpenSSL 库实现的，你只需要调用它的 API 即可。

然后，我们又详细介绍了使用 OpenSSL 库的基本步骤，一共分成七大步。但这里所讲的七大步只是使用 OpenSSL 的一个基本步骤，对于 OpenSSL 的 DTLS 协议握手的 API 我这里并没有进行讲解，但有了这个基础我相信你自己一定可能自行学习 OpenSSL 的其他相关知识了。

最后，我们还讲解了 libsrtp 库的使用，这个库使用起来非常简单，关键点还是你对[上篇文章](#)中安全相关的概念是否清楚，如果清楚的话，就很容易理解 libsrtp 的使用方法了。所以说，那些安全相关的概念虽很基础，但真的很重要。

通过上面的描述，我想现在你应该已经对 WebRTC 如何保护数据的安全有了一个清楚的认知了吧！

思考时间

今天留给你的思考题是：RTP 包被加密为 SRTP 包，这个过程是对整个包进行的加密，还是对 RTP 包的 Payload 部分加的密？

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 21 | 如何保证数据传输的安全（上）？

下一篇 23 | 实战演练：通过WebRTC实现一个1对1音视频实时直播系统

精选留言 (2)

写留言



重生

2019-09-08

老师，有一点疑惑，SDP中交互用户名和密码是明文，这样黑客不是可以获取到吗？或者模仿对应得用户名和密码机制，这样无法提供安全保护吧？

展开 ∨



XE.COM

2019-09-03

老师，能不能介绍一些关于WebRTC源码分析方面的资料，谢谢

作者回复: 以后会专门出这方面的课程

