WebRTC内置debug工具，详细参数解读 - 音视频通信 - SegmentFault

为了确保这篇文章所写内容尽可能的准确，我决定请来Philipp Hancke来作为此篇文章的共同作者。

当你想要找到你WebRTC产品中的问题时，webrtc-internals是一个非常棒的工具，因为你需要用它测试WebRTC以及debug，或者你需要对你的配置进行微调。

如何获得webrtc-internals的数据转储（stats dump）？

如果你对这个工具不熟悉的话，那么打开你Chrome浏览器里的WebRTC段，在这段里打开另一个表单并且将其指向这个内部（internal）URL：chrome://webrtc-internals/

webrtc-internals允许将轨道作为大型的JSON下载下来，这样你就可以一层一层地来看它了，但是当你这么做的时候，你会看到类似这样的东西：

查看webrtc-internals数据

人们通常问到的第一件事是—这些数字到底代表什么？一位我们自己的测试人员将这些值放入时序图表里并且将其输出出来。这就给了我们要比直接从webrtc-internals中取出的300×140的图片要大的多的图表。

这些图表是使用HighCharts库得到的，并且有很多十分方便的特性，比如隐藏线条，放大所需区域，或者停靠在特定点处并显示精确值。这比用JSON转储（像上面一样）要方便的多。

回到基础的webrtc-internals页中。在此页顶端，我们可以考到一系列的表单，一个是给getUserMedia调用的，剩下的两个分别给每个RTCPeerConnection。

在GetUserMedia请求表单中，我们可以看到每次的getUserMedia调用，以及相关约束。不幸的是，我们不能看到结果或者MediaStreams中有的ids。

RTCPeerConnection数据

对于每个peerconnection，我们可以在这里看到这四点：

1. RTCPeerConnection是如何配置的，也就是STUN和TURN服务器是如何被使用的，以及如何配置PeerConnection
2. API的轨迹被调用显示在左边。这些API轨迹展现了所有的RTCPeerConnection调用和他们的参数（例如createOffer），以及回调和类似于onicecandidate的事件触发器
3. 从getStats() API采集的数据在右侧被显示出来
4. 由getStats() API产生的图表在底部显示

RTCPeerConnection API轨迹是非常强大的工具，可以帮助你完成很多的事情，比如分析造成ICE失败的原因，或者帮你找到适合部署TURN服务器的地方。我们会在以后的博文中来谈这些。

webrtc-internals所给出的统计数据是Chrome的内部格式。这意味着其与目前的规范略有不同步，一些名称和结构体会有改变。在较高层，我们在webrtc-internals页上看到的与我们调用这个函数所得到的结果相近：

下面是RTCStatsReport对象的队列，其中有很多秘钥和数值，可以这样读取：

要记住的是在这些统计数据和规范之间有一些区别。这里面有一个经验法则，任意一个名称以“Id”结尾的秘钥都包含一个指向不同的报告，其id属性与秘钥的值对应。所以全部这些报告都是彼此相连的。还要注意，这些值都是字符型的，尽管它们看起来像布尔值那样的数字。

RTCStatsReport中最重要的属性是报告的种类，下面是其中的几种：

- googCertificate
- googComponent
- googCandidatePare
- localCandidate
- remoteCandidate
- ssrc
- VideoBWE

让我们来深入探讨一下这些报告型

googTrack与googLibjingleSession报告

googTrack和googLibjingleSession没包含什么信息，所以我们跳过它不做分析。

googCertificate报告

googCertificate报告包括了一些有关近端和对等端所使用的DTLS证书的信息，以及指纹和哈希算法。这些都在RTCCertificateStats字典中有详细说明。

googComponent报告

googComponent报告的作用就像是认证数据与连接之间的胶水。它包含了一个纸箱当前活跃的候选项对的指针，以及有关用语DTLS和SRTP加密的加密套接字。

googCandidatePair报告

googCandidatePair对一对ICE候选做了描述，也就是低层次的连接。从这个报告中，你可以得到这些信息：

- 发送和接收的数据包以及字节数总数（bytesSent，bytesReceived，packetsSent；因为不明原因丢失的packetsReceived）。这是一个包含RTP报头的UDP或者TCP字节。
- 如何判断这是否是一个活跃的连接，googActiveConnection的值是真则为活跃，否则为假。大多数时间你都会只对活跃的候选对感兴趣。对等的规范可以在这里找到。
- 被发送和接收的STUN请求和应答数量是计算在ICE进程中输入和输出的STUN请求数量。

- googRtt是最新的STUN请求的往返时间。这与ssrc报告上的googRtt是不一样的，我们稍后会说。
- localCandidateId和remoteCandidateId指向localCandidate型和remoteCandidate型。localCandidate和remoteCandidate描述了本地和远端的ICE候选项。你可以在googLocalAddress型上面找到绝大多数信息。
- googTr以及googLocalCandidateType的值。
- googTransportType规定了传输的类型。注意这些数据的值通常是“udp”的，即便是在TCP上的TURN被用于连接TURN服务器的情况下。只有当ICE-TCP被使用时，此值才会是“tcp”的。

从下面这张图上可以比较直观地看到一些数据，如发送和接收的字节数等等：

localCandidate和remoteCandidate报告

感谢上天localCandidate和remoteCandidate与规范中所描述的是一模一样的，告诉我们ip地址，端口号，以及候选项的类型。对于TURN候选来说，其会告诉我们候选被分配在哪个端口上了。

Ssrc报告

ssrc报告是这里面最重要的报告之一。每一个音频或者视频轨道发送或接收都有一个ssrc报告。在旧版本的规范中，这些叫做MediaStreamTrackStats和RTPStreamStats。其内容决定于这是音频还是视频轨道，以及这是发送还是接收。让我们先来描述下一些其中基本的元素：

- mediaType表示我们在观察的是音频数据还是视频数据
- ssrc属性指定了媒体是发送ssrc还是接收
- googTrackId会识别这些数据描述的轨迹。这个id可以在SDP中，以及本地或远端媒体流轨道中被找到。事实上，这违背了以“Id”为后缀的命名法则，通常以“Id”结束的都是一个指向其他报告的指针。Google把goog stats给搞错了。
- googRtt表示的是往返时间。与之前说过的往返时间不同，这个往返时间是从RTCP测量的时间。
- transportId，是指向被用于传送RTP流的部分。通常用于音频和视频流的transportId是一样的。
- googCodecName规定了编译码器的名称。典型的音频编解码器是opus，对于视频来说，使用的是VP8，VP9或者使用H264。你还可以看到在codecImplementationName统计数据中使用的实施方案的有关信息。
- bytesSent，bytesReceived，packetsSent以及packetsReceived的值可以让你计算出总的字节数。这些数字是累加的，所以你需要在你最后一次询问getStats之后要将其按时间分开。规定中的示例代码写的还不错，单是要注意Chrome有事会将这些计数器重置，所以你有可能得到一个负数的速率。
- packetsLost让你知道有多少包在传输过程中丢失了。对于发送端来说，丢包来自RTCP，对于接收端来说，丢包是在本地测量的。当你在检查一个质量不好的通话时，这个参数可能是你想要查看的最直接的数据。

音频特性

对于音轨来说，我们有audioInputLevel和audioOutputLevel（在规范中叫做audioLevel）可以告诉我们音频信号是来与麦克风，还是通过扬声器播出的。这个特性可以用来探测Chrome里不受欢迎的音频bug。我们还可以通过googJitterReceived和googJitterBufferReceived得知有多少抖动被接收，以及jitter buffer的状态。

视频特性

对于视频轨道来说，我们有两大信息需要关注。第一个是被送入googNacksSent，googPLIsSent和googFirsSent中，NACK，PLI和FIR数据包的数量差别。这可以让我们知道丢包会如何影响视频质量。

更重要的是，我们得知了框架大小和速率是作为输入（googFrameWidthInput，googFrameHeightInput，googFrameRateInput）并且实时上是发送到网络之上（googFrameWidthSent，googFrameHeightSent，googFrameRateSent）。

相似的数据可以在接收端被收集到存在googFrameWidthReceived，googFrameHeightReceived中。对于框架速率来说我们甚至可以将其从googFrameRateReceived，googFrameRateDecoded和GOOGFrameRateOutput中分开来。

在编码端我们可以看到这些值之间的差别，还能知道为什么图片会被缩小。通常这些事情发生不是因为没有足够大的CPU，就是没有足够大的带宽来传送完整的图片。另外，想要降低框架速率（其可以从对比googFrameRateInput和googFrameRateSent之间的差距得到），我们需要得到额外的信息：分辨率是否因为CPU的问题而得到适应，以及是否是因为带宽不够使得googBandwidthLimitedResolution的值是真。无论是上述哪个情况发生了改变，googAdaptionChanges计数器都会增加。

我们可以从这张图表上看到这些变化：

这里的丢包是人为产生的。作为反应，Chrome在t=184时第一次尝试降低分辨率，这是绿线代表的googFrameWidthSent开始偏离黑线代表的googFrameWidthInput。接下来在t=186时，框架开始下降，输入框架速率（浅蓝色线条所示）大约是30fps，与发出的框架速率（蓝色线条所示）产生区别，后者几乎是0.

另外，Chrome在ssrc报告中公开了大量关于音频和视频堆栈的表现的数据。我们会在未来的博文中进行讨论。

VideoBWE报告

最后，但并不是不重要，我们来分析一下VideoBWE报告。就像它名字所表达的，它包括有关带宽估计的信息。但是还有一些其他的有用信息包含在这个报告里：

- googAvailableReceiveBandwidth—对于接收视频数据可用的带宽。
- googAvailableSendBandwidth—对于发送视频数据可用的带宽。
- googTargetEncBitrate—视频编码器的目标比特率。这项指标会尝试填满可用的带宽。
- googActualEncBitrate—视频编码器输出的比特率。通常这与目标比特率是匹配的。
- googTansmitBitrate—这个比特率是实际传输的比特率。如果此数值与实际编码比特率有较大的差别，那么可能是因为前向错误纠正造成的。
- googRetransmitBitrate—如果RTX被使用的话，这项允许测量重传的比特率。此数据通常代表丢包率。
- googBucketDelay—是Google为了处理大框架速率的策略表示。通常是很小的数值。

正如你看到的，这个报告会给你视频质量最重要的信息—可用带宽。查看发送和接收的可用带宽通常都是在深入分析ssrc报告之前做的最重要的事。因为有时你可能会发现这样的情况，这解释了用户所抱怨的“质量差”：

在这种情况下，“在所有时间里，带宽估计都在下降”是对质量问题的一个比较好的解释。

> *原作者：Levent-Levi*
> *翻译：刘通*
> *原文链接：http://testrtc.com/webrtc-int...*