计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(10):118−123 [doi: 10.15888/j.cnki.csa.005989]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 WebRTC 的多媒体通信模块①

于　波[1], 缪红娣[1,2]

[1](中国科学院 沈阳计算技术研究所, 沈阳 110168)
[2](中国科学院大学, 北京 100049)

摘　要: 传统的 Web IM 要求通信双方在浏览器中安装插件 (如 Adobe Flash Player), 这不但降低了用户体验, 还增加了开发者对插件开发、更新及维护的繁琐工作. 此外, 传统的 Web IM 主要采用了定时访问服务器的方法 (即轮询方式) 实现浏览器与服务器之间的交互, 该方式降低了实时性且增加了对服务器资源的消耗. 针对上述问题, 本文采用 WebSocket 连接技术使得浏览器与服务器之间能够通过长连接方式进行数据交互, 该方式提高了实时性, 降低了对服务器的负载. 然后, 本文依据提供的 WebRTC API, 实现了一个具备音视频通信以及文件传输功能的多媒体通信模块. 通过 MVC 三层架构模式, 对该模块进行了具体划分与实现. 最后通过测试表明该设计能够满足用户的基本功能需求.

关键词: WebRTC; WebSocket; 视频通信; 文件传输

引用格式: 于波,缪红娣.基于 WebRTC 的多媒体通信模块.计算机系统应用,2017,26(10):118–123. http://www.c-s-a.org.cn/1003-3254/5989.html

Multimedia Communication Module Based on WebRTC

YU Bo[1], MIAO Hong-Di[1,2]

[1](Shenyang Institute of Computer Technology, Chinese Academy of Sciences, Shenyang 110168, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: The traditional Web IM requires both sides to install plug-ins (such as Adobe Flash Player) in the browser, which does not only reduce the user experience, but also increases the developers' tedious work for plug-in development, updating and maintenance. In addition, the traditional Web IM mainly adopts the method of the regular access server (the polling mode) to achieve the interaction between the browser and server, which reduces the real-time performance and increases the consumption of server resources. According to the problems above, this article implements the long connection between the browser and server and data interaction through WebSocket, which improves the real-time performance. Then on the basis of providing WebRTC API, we implement a module which performs the function of audio and video communication and file transfer. Through the MVC three-tier architecture model, we carry on the concrete division and implementation of the module. Finally, the test shows that the design can meet the basic functional requirements of the user.

Key words: WebRTC; WebSocket; video communication; file transport

传统的 Web IM 采用基于 Ajax 轮询或是 Comet 的信令传输方式实现浏览器与服务器的数据交互. 二者存在不同方面的缺陷. 通过 Ajax 方式可以更新局部界面, 减少了响应中传输的数据量, 但可能会导致大量请求的产生, 浪费了网络资源, 增加了服务器负载. 而在 Comet 方式中, 可以实时更新内容, 但为了保持响应, 一次连接的时间会增加, 为了保持浏览器与服务器之间连接的持久性需要消耗大量的资源.

① 收稿时间: 2017-01-17; 采用时间: 2017-02-15

本文采用 WebSocket 技术, 它通过浏览器与服务器之间建立持久连接的方式进行数据的双向通信, 减少了连接服务器产生的开销, 降低了对网络带宽的占用, 有效解决了 Ajax 和 Comet 造成的问题. 本文还结合 WebRTC(Web Real-Time Commmunication, Web 实时通信) 技术, 设计了一个多媒体通信模块, 实现了音视频通信以及文件传输的功能. 相比于传统的 Web IM, 其无需通信双方在浏览器中安装插件 (如 Adobe flash), 也不要求开发者对音视频数据进行处理, 只需要专注网页的 JavaScript 程序的开发.

1 相关技术简介

1.1 WebSocket 与其他 Web 推送技术

WebSocket 是 HTML5 的一项新技术, 实现了浏览器与服务器之间的全双工通信. WebSocket 是建立在 TCP 协议之上的一个独立的协议, IETF 在 2011 年 12 月将其定义为标准. 它是建立在 HTTP 基础上的协议, 由客户端发起连接, 并使用 HTTP 协议的 101 状态码进行协议切换. 一旦 WebSocket 连接建立, 浏览器和服务器之间就创建了持久性的连接, 并允许数据进行双向传送. 由于其连接具有持久性, 不需要再进行连接的频繁创建, 节省了服务器资源, 也减少了网络带宽的占用. 同时, WebSocket 头部的信息量很小, 因此通讯的信息量也相应缩小, 流量的开销削减.

Flash XML Socket 技术: 用户在安装了 Flash 播放器之后, 根据 Flash 提供的相应的 XML Socket 类, 利用 JavaScript 直接调用 Flash 提供相应的接口与服务器端的套接口进行相关通讯. 服务器端返回的 XML 格式的传送信息, 之后在 web 端显示出 HTML 页面的内容. 采用该技术需要安装相应的插件.

Ajax 轮询: 客户端在处理完服务器端返回的相应信息后, 会再一次发出请求, 重新建立连接. 如果服务器端此时有新的数据达到, 服务器端将会对信息进行保存, 由客户端通过建立连接, 之后一次性的取回所有信息并进行处置. 该种方式不需要安装插件, 但存在多次连接的可能, 增加了服务器的负载.

基于 Iframe 的流方式: Iframe 是一种 HTML 标记, 通过在 HTML 页面中嵌入隐藏帧, 将该帧的 src 属性设置为对一个长连接的请求, 以此来完成客户端与服务器端之间的数据信息的传输.

WebSocket 技术与其他推送技术在是否需要插件、重连次数以及对浏览器的兼容性这几个方面的对比如表 1.

表 1 Web 推送技术对比

推送技术	插件	重连次数	兼容性
Flash XMLSocket	是	无	需插件
Ajax轮询	否	极多	IE6+
Iframe流	否	较少	IE6+
WebSocket	否	无	IE10+

1.2 WebRTC

WebRTC 技术是一个基于标准化的行业性开源项目, 旨在将实时通讯模块引入到所有浏览器中, 并通过标准的 HTML5 标签和 JavaScript API 将即时通讯功能提供给 Web 开发者.

WebRTC 系统架构主要分三层: 上层是面向第三方开发者的 WebRTC 标准 API(JavaScript), 方便开发者使用; 中间层主要包括相应的音频引擎、视频引擎、数据传输模块, 是 WebRTC 技术的核心实现; 底层包括音视频的采集和网络 IO, 由相应的浏览器厂商进行自主实现. 浏览器之间通过 PeerConnection API 对双向媒体流进行管理; 使用 JavaScript 会话建立协议 JSEP(JavaScript session establishment protocol) 对媒体参数进行协商. 在建立了 DataChannel 之后, 使用 SCTP 协议对媒体数据流进行可靠地传输.

2 核心模块设计

多媒体通信模块主要分为 UI 界面层; 业务逻辑层以及连接层. 其中业务逻辑层主要实现了文件传输以及音视频通信功能模块; 连接层实现了浏览器与服务器之间的双向通信, 并建立了对等连接, 实现了音视频以及文件媒体流的传输通道. 模块化分如图 1.

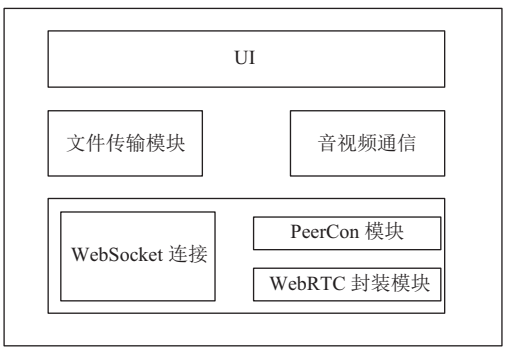

图 1 模块划分

2.1 连接模块

2.1.1 WebSocket 连接

WebSocket 连接负责浏览器与服务器之间的数据交互. 在 HTTP 连接建立后, 需要完成一次“握手”操作才能实现浏览器与服务器之间的持久连接. 该模块在与服务器建立连接时, 调用 connect 方法, 并对 onpen, onmessage, onerror 及 onclose 的回调函数进行相应的实现. 关于 WebSocket 的连接模块函数如表 2.

表 2 WebSocket 的连接模块函数说明

监听事件	说明
onopen	建立连接成功后, 对服务器端发送加入房间的信息.
onmessage	根据服务器返回的信息, 判断eventsName值, 并对其进行相应的处理. 其中eventsName的值存在四种可能: offer; answer; ice_candidate; join.
onerror	连接失败, 将调用该函数.
onclose	连接关闭时, 断开连接建立, 清空信息.

2.1.1 PeerCon 模块

本模块通过信令交换了对等连接两端的 Session 信息, 网络配置 (ice 候选项后) 实现了浏览器之间的对等连接, 为浏览器间的直接通信提供了条件. 并建立了数据通道 DataChannel, 经由此通道可以进行各种类型数据的传输. PeerCon 模块主要函数的说明如表 3.

表 3 PeerCon 模块主要函数说明

监听事件	说明
onopen	连接建立成功
onaddstream	添加本地媒体流
onicecandidate	将ICE候选发送到其余客户端
ondatachannel	数据通道建立成功后, 可以向其他用户发送数据

2.2 音视频通信模块

Alpha 和 Beta 进入同一个房间, 连入服务器进行通信. 当 Alpha(或 Beta) 点击浏览器调用该应用时, 便下载相关的 HTML 页以及 Javacscript 并通过 WebSocket 保持浏览器的持久连接.

2.2.1 呼叫流程

Alpha 点击网页上的呼叫按钮启动与 Beta 的通话, 先调用 getUserMedia 函数, 控制本机的音视频设备, 获取本地音视频流. 而后对 PeerConnection 对象进行实例化, 通过 addStream 将媒体流关联到该对象中. 设置本地 sdp, 并通过请求信令 (offer) 转发给 Beta, 在接收到 Beta 的应答 (answer) 后, 解析出 Beta 的 sdp 消息, 加载至浏览器内核, 完成呼叫建立.

其中信令类型为 Json 字符串, 其格式为

```
{
  “event”:”offer”, //应答的信令或请求的信令
  “data”:{
    “sdp”:sess_description, //会话描述相关信息
    “socketId”:socketId//远端用户的 socketId
    }
}
```

2.2.2 被呼叫流程

Beta 收到请求信令 (offer) 后, 从中解析出 Alpha 的 sdp, 并决定是否要接受通话请求. 若接受请求, 则实体化一个与 Alpha 请求相关的对等连接. Beta 的浏览器确认呼叫建立并且媒体流已经产生. 之后作为响应, 产生一个包含媒体信息和 ICE 候选的信令信息 answer, 并传回给 Alpha. 如图 2 所示为用户通信流程.

2.3 文件传输模块

WebRTC 不仅支持对等媒体的连接, 还提供了一种灵活可配置的数据通道. 基于此通道实现的文件与信息的传输有如下优势.

(1) DataChannel 支持流量大, 延迟低的连接, 既稳定可靠又不失灵活性.

(2) 无需服务器中转即可实现数据的交换.

(3) 无需插件, 即可实现文件的实时传输与共享.

在本模块中, 首先读取文件数据, 使用 PeerCon 模块在用户间创建一个对等连接, 并建立一个数据通道; 对所选择的文件进行分片, 并通过文件信令控制碎片传输. 最后对碎片进行组装并提供下载.

2.3.1 文件读取与分片

通过 HTML5 提供的 File API 规范使得用户能够与本地文件进行交互. 用户通过表单选择文件后, 触发了文件选择框的 change 事件, 通过 files 属性获取选定文件列表. 然后通过 FileReader 的接口异步读取文件内容为 DataURL.

如果要上传的文件很小, 则可以直接通过 File API 将其存储为一个 Blob 对象, 并通过数据通道进行可靠地传输.

如果文件较大, 则需要一个分片机制: 将文件分成多个碎片 (chunk), 然后进行分片发送. 在发送过程中, 数据包大小应该控制在 1200 字节, 且文件块上应该附加一些便于对方识别的元数据, 如块的 ID. 若文件的分片过小, 分片数量将会增加, 会导致传输延时的增加, 此时可以对 chunk 进行进一步地封装, 将多个 chunk 封装成一个 block 块, 如此便可节省很多的带宽和 CPU.

2.3.2 文件传输与碎片组装

对于文件的传输有两种方式, 支持 SCTP 的可靠传输方式, 其内置了流量控制, 可以进行可靠的传输. 以及不可靠的传输方式, 虽然可以提升传输速率, 但文件可能丢失, 无法得到完整文件. 因此需要进行动态的流量控制和重传机制.

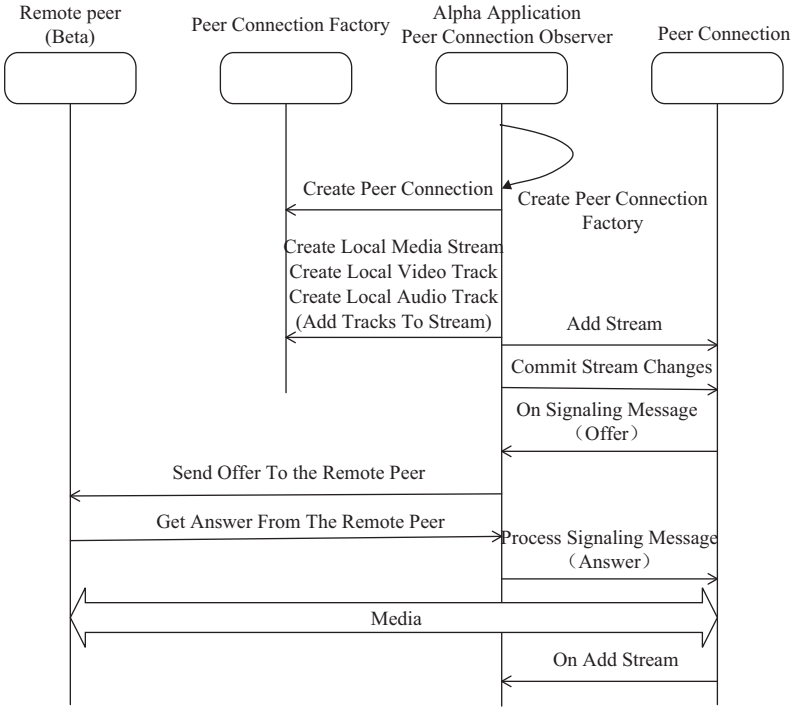

图 2 用户通信流程

为了保证接收到完整的文件以及所有的文件块. 在不可靠的传输方式上, 接收端将请求它没有的 chunk, 并监控这些 chunk, 防止请求端没有接收到回应. 设置一个数据结构: PendingChunks, 根据它来判断哪些 chunk 没有被请求到或者还没有到达. 并通过监控它来设置一个定时机制, 决定一个请求何时被抛弃以及如何处理.

此外, 可以通过动态调整数据片量的方式来控制流量. 通过设置 SDP 中参数字段 b, 定义最大宽带. 若所有请求发送成功, 我们将窗口大小加一. 若有一个失败, 则将窗口大小减去 2 *(丢弃的 chunk), 且窗口大小不超过窗口的一半. 如图 3 所示为文件发送流程.

在接收方接收到文件块之后, 先对数据包进行解析, 判断其是否为最后一个碎片. 如果不是, 则将碎片保存在离线存储中, 并等待下一个碎片. 如果所有的碎片都接收完毕, 则对碎片进行组合, 将其拼合为一个完整的文件, 并进行下载, 方法如下:

function downLoadFile(blob){

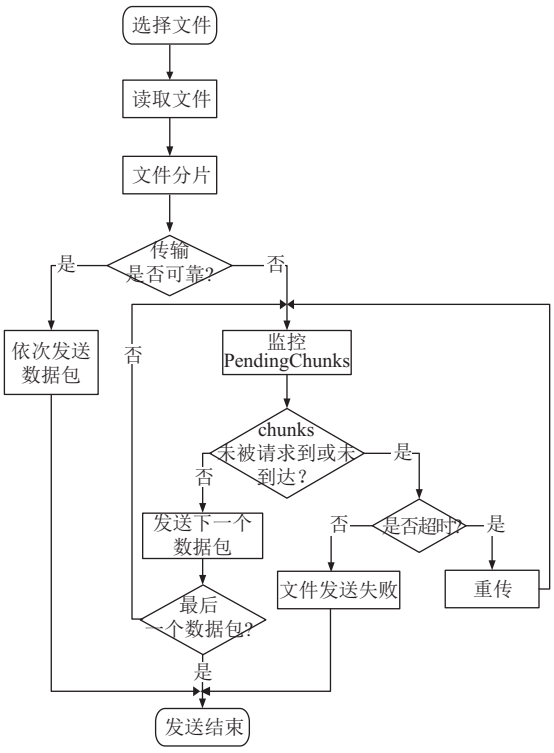

图 3 文件发送流程

```
//创建一个 a 标签
var wrapper= document.createElement('a');
//创建一个新的对象 URL, 并赋值给 href 属性
wrapper.href=window.URL.createObjectURL(blob);
//指定浏览器下载时采用新的文件名称
wrapper.download = 'test';
wrapper.click();};
```

如图 4 所示为文件接收流程.

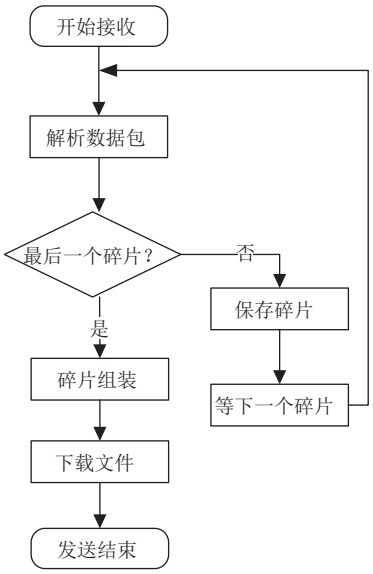

图 4 文件接收流程

2.3.3 文件信令控制

在文件传输的过程中, 通过解析 DataChannel 上的文件类型包来确定文件信令的类型. 主要的信令类型有请求发送 request; 文件碎片 chunk; 文件接收 accept; 文件拒绝 refuse; 文件确认 ask. 其中文件类型包定义为 Json 字符串, 定义的格式为:

```
{
  type:"confile",
  signal:"request", //信号类型
  sendId:sender_id, //发送者 id
  name:fsendFile.file.name, //发送文件的名称
  size: fsendFile.file.size, //文件大小
};
```

3 性能测试

测试环境为使用 Node.js 编写的 WebRTC 服务器端, 通过在 3000 端口监听 WebSocket 连接请求. 根据分片的大小不同, 对于传输 1MB 的文件, 所需要的时间也会有所不同, 如下图所示. 当分片过小时, 分片数据量增大, 导致传输中延迟增加, 文件传输总时间也会增加. 当传输的分片增大, 总传输延迟减小, 总传输时间就会趋于一个平稳的值. 如图 5 所示为文件分片传输结果.

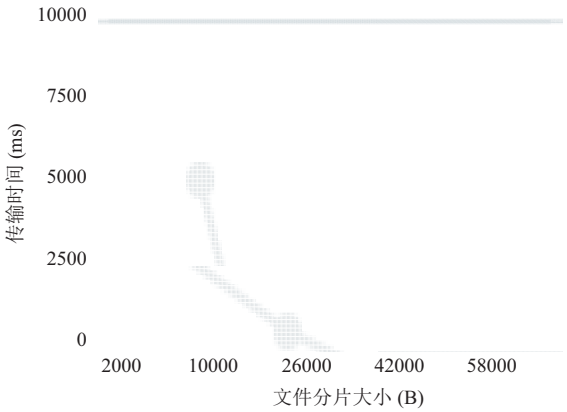

图 5 文件分片传输

在分片大小固定为 40000B 的情况下, 对于传输不同大小的文件所需时间进行测试. 从测试结果中可以看出, 传输时间基本趋于线性的增长. 如图 6 所示.

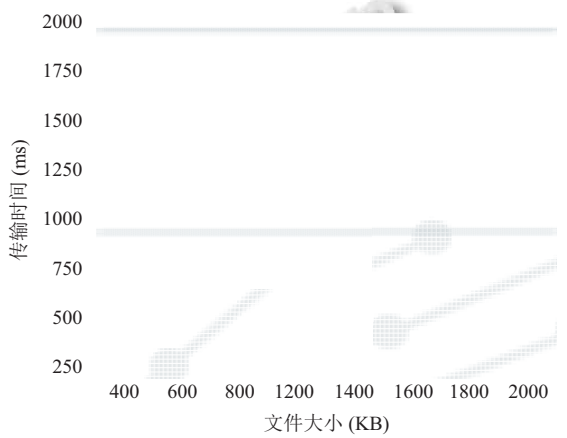

图 6 文件传输时延

4 结语

WebRTC 不但给我们带来了基于浏览器的音视频体验, 还给我们提供了无需插件, 无需服务器中转即可以实现文件传输的条件. 本文通过 WebSocket 连接模块构建起服务器与浏览器之间的数据交互. 通过 PeerCon 模块实现浏览器间的连接建立, 为浏览器间音视频通

信提供基础; 同时为文件传输建立起数据通道 DataChannel. 在本文中, 文件通过分片以及重组的方式进行传输, 在不可靠传输方式时为其提供动态流量控制以及重传机制保证文件的完整性. 从最后的测试结果可以得出, 该模块实现了实时的音视频聊天以及文件传输的功能.

参考文献

1 屈振华, 李慧云, 张海涛, 等. WebRTC 技术初探. 电信科学, 2012, 28(10): 106–110. [doi: 10.3969/j.issn.1000-0801.2012.10.018]

2 付斌, 杨鑫, 王松, 等. WebRTC 技术研究及其应用. 电信科学, 2013, 29(9): 108–112.

3 林鸿, 王松, 杨鑫, 等. 基于 WebRTC 技术的应用及平台技术开发与设计. 电信科学, 2013, 29(9): 20–25, 36.

4 才鑫. 基于 WebRTC 的多方多媒体通信系统的设计与实现[硕士学位论文]. 北京: 北京邮电大学, 2015.

5 王令宇. 基于 SaaS 模式的即时通信平台的设计与实现[硕士学位论文]. 北京: 北京邮电大学, 2015.

6 Schaefer C, HO C, Harrop R. WebSocket. In: Sv C, HO C, Harrop R, eds. Pro Spring. 4th ed. New York: Apress, 2014: 645–661.

7 Johnston AB, Burnett DC. WebRTC: APIs and RTCWEB protocols of the HTML5 real-time web. St. Louis: Digital Codex LLC, 2012.

8 Nurminen JK, Meyn AJR, Jalonen E, *et al*. P2P media streaming with HTML5 and WebRTC. Proc. of 2013 IEEE Conference on Computer Communications Workshops. Turin, Italy. 2013. 63–64.

9 Sredojev B, Samardzija D, Posarat D. WebRTC technology overview and signaling solution design and implementation. Proc. of the 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics. Opatija, Croatia. 2015. 1006–1009.

10 Janczukowicz E, Braud A, Tuffin S, *et al*. Evaluation of network solutions for improving WebRTC quality. Proc. of the 24th International Conference on Software, Telecommunications and Computer Networks. Split, Yugoslavia. 2016. 1–6.

11 Johnston AB, Buenett DC. Web WebRTC 权威指南. 声网 Agora.io, 译. 北京: 机械工业出版社, 2016.