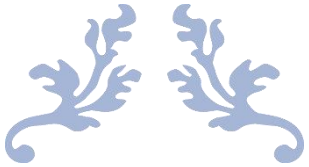


仿 airdrop 软件项目设计

龙语嫣 41911038
谢　融 41911030

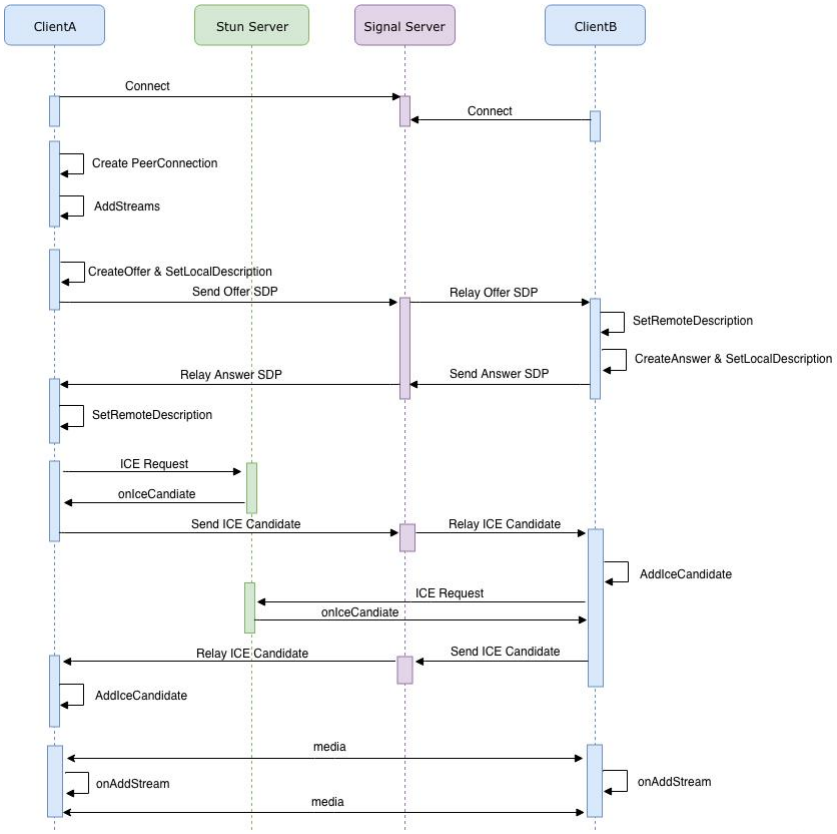
仿 airdrop 软件项目设计

1. 关键功能

1.1 设计框架

WebRTC，名称源自网页即时通信，是一个支持网页浏览器进实时语音对话或视频对话的 API。

WebRTC 的数据通道(DataChannel)支持两个节点之间的双向数据传输，如图是其一对一的通道链路：



为了实现多设备文件传输，我们必须把一对一 P2P 模式转换为多对多的模式，因此采用了 SFU 服务器。SFU 服务器最核心的特点是把自己“伪装”成了一个 WebRTC 的 Peer 客户端那么，此时 WebRTC 的其他客户端其实并不知道自

已通过 P2P 连接过去的是一台真实的客户端还是一台服务器，这种连接被叫做 P2S。除了“伪装”成一个 WebRTC 的 Peer 客户端外，SFU 服务器还有一个最重要的能力就是具备 one-to-many 的能力，即可以将一个 Client 端的数据转发到其他多个 Client 端。这种网络拓扑结构中，无论多少人同时进行数据传输，每个 WebRTC 的客户端只需要连接一个 SFU 服务器，上行一路数据即可，极大减少了多人数据传输场景下 Mesh 模型给客户端带来的上行带宽压力。

1.2 文件传输

（1）在前端页面触发传送文件的事件(单击)后：

调用 class PeerUI，文件选择和传输主要由_onFilesSelected()和 setProgress()两个函数实现。

（2）上传文件以后：

解析文件（class FileDigester）并将文件分块（class FileChunker）传输。

上传文件到文件队列，解析并返回文件摘要，按分区解析文件，同时可查看下载进程和文件传输情况。

1.3 文本信息传输（多设备共用剪贴板功能）

（1）在前端右键单击(PC 端)或长按(移动端)触发传送文本的事件后：

调用 class SendTextDialog extends Diolog，将文本信息发送到服务器。

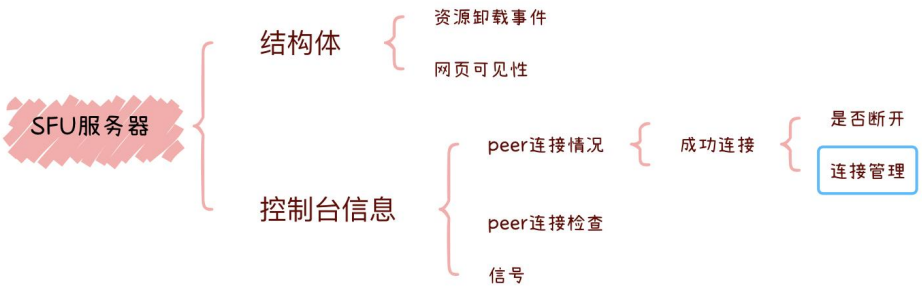
（2）接受文本信息以后：

转换文本编码，再转换文本格式。在接收到文本信息时，先解码再处理事件，处理完成以后，调用 class ReceiveTextDialog extends Diolog 将文本信息显示在前端页面上，并且设置有 copy 功能。

copy 功能主要由 clipboard.js 实现，采用的是 execCommand()方法。

2.关键代码

2.1 SFU 服务器



页面基础情况：

```
Events.on('beforeunload', e => this._disconnect());
//当窗口、文档及其资源即将被卸载时触发该事件
Events.on('pagehide', e => this._disconnect());
//当浏览器在呈现与会话历史不同的页面的过程中隐藏当前页面
//用于检测页面卸载事件
document.addEventListener('visibilitychange', e =>
this._onVisibilityChange());
//当用户导航到新页面、切换选项卡、关闭选项卡、最小化或关闭浏览器，
// 或者在移动设备上从浏览器切换到不同的应用程序时，此事件会触发
```

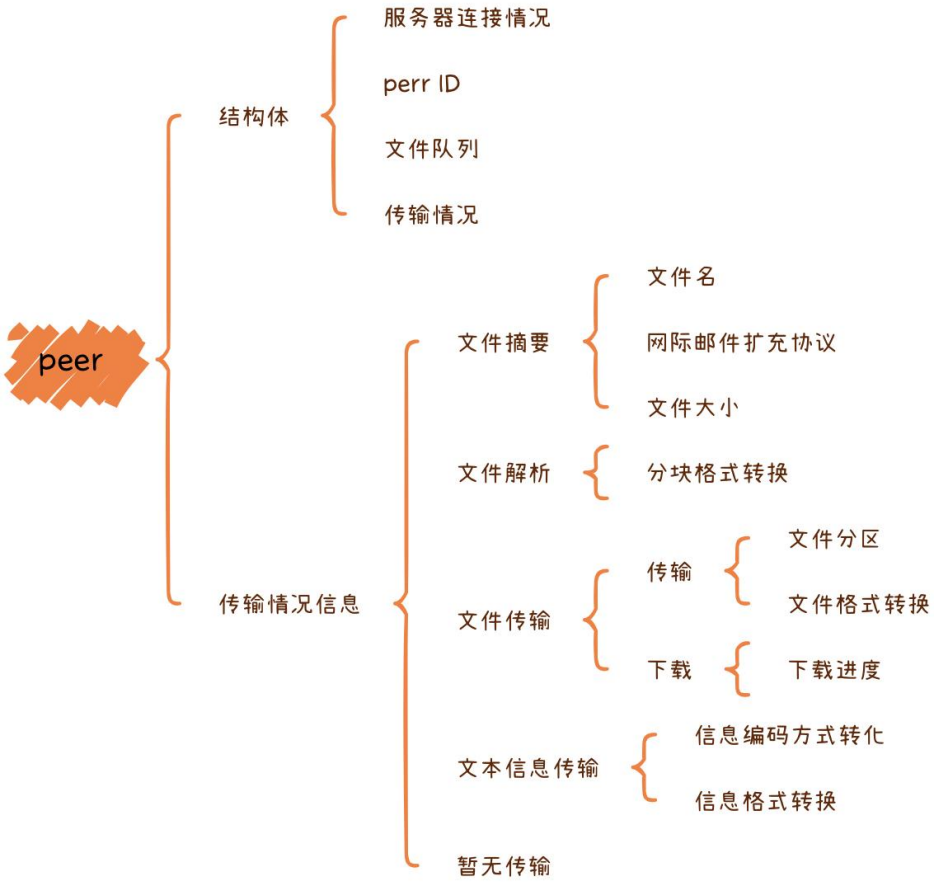
连接：

```
clearTimeout(this._reconnectTimer);//是否断开（重连倒计时）
if (this._isConnected() || this._isConnecting()) return;
//重连未响应
const ws = new WebSocket(this._endpoint());
//提供了用于创建和管理与服务器的WebSocket连接以及在连接上发送和接
收数据的 API
ws.binaryType = 'arraybuffer';//数据缓冲
ws.onopen = e => console.log('WS: server connected');//连接成功提示
ws.onmessage = e => this._onMessage(e.data);//控制台信息
ws.onclose = e => this._onDisconnect();//关闭连接
ws.onerror = e => console.error(e);
this._socket = ws;
```

控制台信息：

```
msg = JSON.parse(msg);//字符串转换
          console.log('WS:', msg);//发送到控制台
          switch (msg.type) {
               case 'peers':
                    Events.fire('peers', msg.peers);//连接 peer
                    break;
               case 'peer-joined':
                    Events.fire('peer-joined', msg.peer);
                    break;
               case 'peer-left':
                    Events.fire('peer-left', msg.peerId);//连接情况
                    break;
               case 'signal':
                    Events.fire('signal', msg);
                    break;
               case 'ping':
                    this.send({ type: 'pong' });//连接检查
                    break;
               case 'display-name':
                    Events.fire('display-name', msg);
                    break;
               default:
                    console.error('WS: unkown message type', msg);//错误：
未知信息类型
```

2.2 Peer



结构：

```
constructor(serverConnection, peerId) {
        this._server = serverConnection;//建立服务器链接
        this._peerId = peerId;
        this._filesQueue = [];//上传文件到队列
        this._busy = false;
    }
```

文件及文本信息传输：

```
if (typeof message !== 'string') {
            this._onChunkReceived(message);//一旦收到
            return;
        }
        message = JSON.parse(message);
        //解析 JSON 字符串，构造字符串描述的 JavaScript 值或对象。
        // 可以提供一个可选的 reviver 函数来在结果对象返回之前对其执行转换
        console.log('RTC:', message);
        //将消息输出到 Web 控制台。
        // 消息可以是单个字符串（带有可选的替换值），也可以是任何一个或多个 JavaScript 对象
        switch (message.type) {
            case 'header':
                this._onFileHeader(message);//返回文件摘要
                break;
            case 'partition':
                this._onReceivedPartitionEnd(message);//分区结果
                break;
            case 'partition-received':
                this._sendNextPartition();//发送下一个分区
                break;
            case 'progress':
                this._onDownloadProgress(message.progress);//下载进程
                break;
            case 'transfer-complete':
                this._onTransferCompleted();//传输结束
                break;
            case 'text':
                this._onTextReceived(message);//接受文本信息
                break;
        }
```

2.3 RTCPeer extends Peer

对 peer 增加监听器。

连接:

```
if (!this._conn) this._openConnection(peerId, isCaller);//开始连接

        if (isCaller) {
            this._openChannel();//打开连接通道
        } else {
            this._conn.ondatachannel = e => this._onChannelOpened(e);//开启完成
        }
```

2.4 前端关键功能代码

(1) 文件传输

```
_onFilesSelected(e) {
    const $input = e.target;
    const files = $input.files;
    Events.fire('files-selected', {
        files: files,
        to: this._peer.id
    });
    $input.value = null; // reset input
}  // 选择传输的文件
```

```
setProgress(progress) {
      if (progress > 0) {
            this.$el.setAttribute('transfer', '1');
      }
      if (progress > 0.5) {
            this.$progress.classList.add('over50');
      } else {
            this.$progress.classList.remove('over50');
      }
      const degrees = `rotate(${360 * progress}deg)`;
      this.$progress.style.setProperty('--progress',
degrees);
      if (progress >= 1) {
            this.setProgress(0);
            this.$el.removeAttribute('transfer');
      }
```

(2) 文件接收

```
if(this._autoDownload()){
      $a.click()
      return
}
if(file.mime.split('/')[0] === 'image'){
      console.log('the file is image');
      this.$el.querySelector('.preview').style.visibility = 'inherit';
      this.$el.querySelector("#img-preview").src = url;
}

this.$el.querySelector('#fileName').textContent = file.name;
this.$el.querySelector('#fileSize').textContent =
this._formatFileSize(file.size);
this.show();

if (window.isDownloadSupported) return;
$a.target = '_blank';
const reader = new FileReader();
reader.onload = e => $a.href = reader.result;
```

(3) 文本信息传送

```
_onRecipient(recipient) {
    this._recipient = recipient;
    this._handleShareTargetText();
    this.show();

    const range = document.createRange();
    const sel = window.getSelection();

    range.selectNodeContents(this.$text);
    sel.removeAllRanges();
    sel.addRange(range);
}
_handleShareTargetText() {
    if (!window.shareTargetText) return;
    this.$text.textContent =
window.shareTargetText;
    window.shareTargetText = '';
```

(4) 文本信息接收

```
_onText(e) {
    this.$text.innerHTML = '';
    const text = e.text;
    if (isURL(text)) {
        const $a = document.createElement('a');
        $a.href = text;
        $a.target = '_blank';
        $a.textContent = text;
        this.$text.appendChild($a);
    } else {
        this.$text.textContent = text;
    }
    this.show();
    window.blop.play();
}

async _onCopy() {
    await
navigator.clipboard.writeText(this.$text.textContent);
    Events.fire('notify-user', 'Copied to clipboard');
```

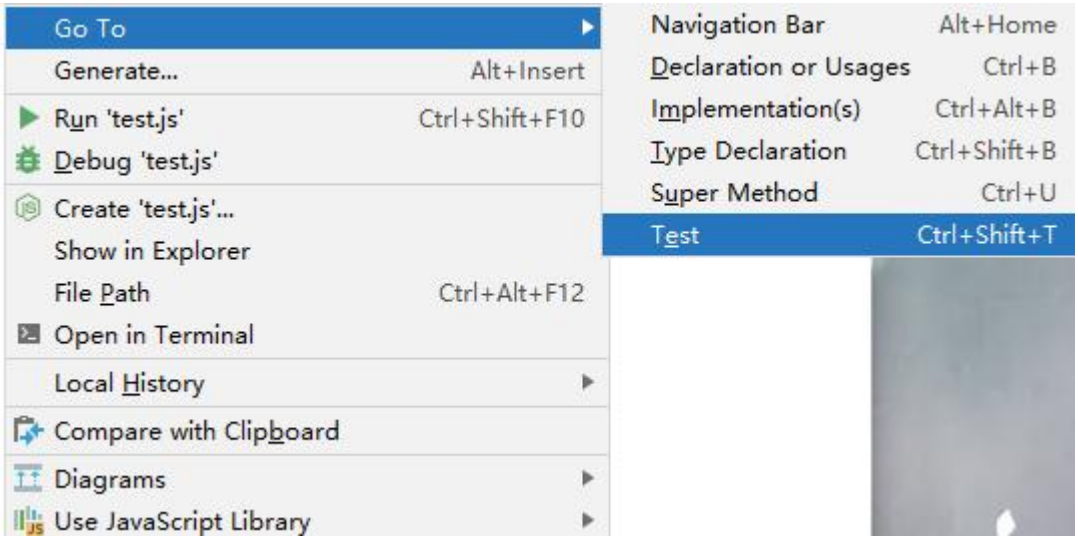
(5) 将接收的文本信息复制到剪贴板

```
const span = document.createElement('span');
span.textContent = text; // 返回文本内容
span.style.whiteSpace = 'pre'; // 保留空格和换行符
span.style.position = 'absolute';
span.style.left = '-9999px';
span.style.top = '-9999px';

// 创建一个窗口用于显示接受到的文本信息
const win = window;
const selection = win.getSelection(); // 创建一个 selection 对象
win.document.body.appendChild(span); // 向窗口中添加 span 为子类
const range = win.document.createRange(); // 创建一个 Range 对象
selection.removeAllRanges(); // 从当前 selection 对象中移除所有 range 对象
range.selectNode(span); // 使 range 对象包含 span
selection.addRange(range); // 将 range 添加到 selection 的 range 对象中

let success = false;
try {
    success = win.document.execCommand('copy');
// 调用 execCommand 方法复制文本(选中的文本进入剪贴板)
} catch (err) {
    return Promise.error();
}
```

3. 单元测试



4. 使用教程

4.1 操作系统

Mac；Windows；android；iOS

4.2 使用步骤

(1) 用户传输文本或使用剪贴板时，需要保证在同一局域网下；然后同时登陆到文件传输系统查看设备是否能够是否连接到；

(2) 右键点击鼠标，能够找出文本框使用剪贴板发送文本信息；

(3) 点击想要传输文件的设备，发送文件。