JsSIP 解析

简介

- the javascript SIP libray.
- Runs in the browser and Node.js.
- Audio/video calls (WebRTC) and instant messaging
- Lightweight!
- Works with OverSIP, Kamailio, Asterisk.
- Written by the authors of RFC 7118 "The WebSocket Protocol as a Transport for SIP" and OverSIP.
- 100% pure JavaScript built from the ground up.
- 截止2022-05-19，最新版本3.9.0。官方文档地址 github

SIP协议

SIP(Session Initiation Protocol)，RFC3261，建议阅读第四节快速了解SIP协议如何进行运作的。

这里还有一些中文资料可以帮助到你。

SIP协议第一版草案于2000年提出，在2002正式落地，后续又有诸多升级，你可以查看datatracker里查看更多信息。

使用

```
import { UA, WebSocketInterface } from 'jssip';
class Communicate {
    _ua: UA;
    id: 1001;
    host: '192.168.2.117';
    port: '8088';
    realm: 'YSAsterisk';
    ha1:'fb99c05ba206'; // 密码
    _session: any;
    constructor(){
        this._session = {};
        // 创建webscoket实例，用于传输SIP报文  RFC7118
        const socket = new WebSocketInterface(
            this.port ? `wss://${this.host}:${this.port}/ws` :
`wss://${this.host}/ws`
        );
        const uri = 'sip:' + this.id + '@' + this.host;
        const config = {
            sockets: [socket],
            uri: uri,
            contact_uri: `${uri};webclient`,
            authorization_user: this.name ? this.name : this.id,
            ha1: this.ha1,
            realm: this.realm ? this.realm : 'YSAsterisk',
            register: this.register,
            user_agent: 'WebClient',
```

```
            session_timers: true,
            register_expires: 1800,
            no_answer_timeout: 300,
        };
        // 创建UA实例
        this._ua = new UA(config);
        // 收到Invite包时触发该事件
        this._ua.on('newRTCSession',(data)=>{
            const sessionData = data;
            const rtc = sessionData.session;
            const id = sessionData.request.call_id;
            // 订阅事件RTCSession触发的事件
            rtc.on('progress',()=>{});
            rtc.on('icecandidate',()=>{});
            rtc.on('sdp',()=>{});
            rtc.on('confirmed',()=>{});
            rtc.on('accepted',()=>{});
            rtc.on('ended',()=>{});
            rtc.on('failed',()=>{});
            // 将通话相关信息及rtc对象存储起来
            this._session[id] = {
                rtc,
                _request: { call_id: id },
                _from_tag: sessionData.request.to_tag,
                _to_tag: sessionData.request.from_tag,
                _last_video_id: '',
                _remoteStream: null,
            };
        })
        this._ua.start();
    }
    // call 是UA的方法
    call(){
        this._ua.call()
    }
    // hold 是RTCSession的方法
    hold(id){
        this._session[id].rtc.hold();
    }
    //  mute 是RTCSession的方法
    mute(id){
        this._session[id].rtc.mute();
    }
}
```

JsSip目录结构

```
| Config.js                 # 配置对象

| Constants.d.ts
| Constants.js              #常量定义：错误码、状态码、方法名等
| Dialog.js                 #sip dialog可见RFC3261 12.1，中文资料SIP中的dialog、call、session、transaction
```

```
│  DigestAuthentication.js        #数字签名
│  Exceptions.d.ts
│  Exceptions.js                  # 异常处理
│  Grammar.d.ts
│  Grammar.js                     # 语法转换器，用于转换SIP报文。该转换器基于peg.js生成的 peg.js基础使用   peg.js官网
│  Grammar.pegjs                  # 转换器规则
│  JsSIP.d.ts
│  JsSIP.js                       # 入口
│  Logger.js                      # 日志输出
│  Message.d.ts
│  Message.js                     # 消息类，多用于M消息
│  NameAddrHeader.d.ts
│  NameAddrHeader.js              #
│  Options.js                     # 操作类，有接听、拒接等方法
│  Parser.js                      # 工具方法，将消息头转换为对象
│  Registrator.d.ts
│  Registrator.js                 # 注册类
│  RequestSender.js               # 请求发送类
│  RTCSession.d.ts
│  RTCSession.js                  # * RTC会话类，通话中的answer、hold、mute等方法都在此。核心模块。
│  sanityCheck.js
│  SIPMessage.d.ts
│  SIPMessage.js                  # 包含四个类OutgoingRequest，InitialOutgoingInviteRequest，IncomingRequest, IncomingResponse
│  Socket.js
│  Timers.js
│  Transactions.js                # SIP事务，相关解释见上方dialog中文资料
│  Transport.d.ts
│  Transport.js                   # webscoket 传输对象
│  UA.d.ts
│  UA.js                          # * 用户注册、连接服务器等功能。核心模块
│  URI.d.ts
│  URI.js                         # SIP URI
│  Utils.d.ts
│  Utils.js
│  WebSocketInterface.d.ts
│  WebSocketInterface.js  # 提供webscoket 连接、发送消息等方法，此类会暴露给外部程序
│
├─Dialog
│      RequestSender.js
│
└─RTCSession
        DTMF.js
        Info.js
        ReferNotifier.js    # refer 方法使用
        ReferSubscriber.js # refer 方法使用
```

UA & RTCSession

> 本节将从UA实例化过程，呼出，呼入三个方面剖析JsSIP，代码基于3.9.0。

> ps：下文中的链接会跳转到代码中具体行位置。
>
> 跳转到git仓库后可以将github改为github1s，快速进入网页编辑器模式。如：
>
> https://github.com/versatica/JsSIP/blob/3.9.0/lib/UA.js 改为 https://github1s.com/versatica/JsSIP/blob/3.9.0/lib/UA.js

- UA实例化

  UA实例化构造方法主要干了两件事：1.加载配置；2.实例化Registrator;

  1. 加载配置

     加载配置调用了_loadConfig方法，在这个方法中会实例化Transport对象，并订阅onconnecting、onconnect、ondisconnect、ondata事件。

     其中onconnect绑定的onTransportConnect 方法中调用了Registrator对象的register()方法。所以UA注册时机是在web scoket连接后。

  2. 实例化Registrator

     实例化Registrator会被this._registrator接收，在UA的register()、unregister()等有关注册的方法都会调用该对象。

- 呼出

  UA的call()方法。

  该方法实例化了RTCSession，并调用RTCSession的connect()方法。

  进入RTCSession，在connect方法中调用了_createRTCConnection()方法，在这个方法中实例化了webRTC的RTCPeerConnection对象，并发布`peerconnection`事件。

  在调用_createRTCConnection()方法后调用了__newRTCSession()，在这个方法中调用了UA对象的newRTCSession()方法，在这个方法里发布了`newRTCSession`事件。它们两调用关系如下方代码：

```
// UA.js
newRTCSession(session, data)
{
    this._sessions[session.id] = session;
    this.emit('newRTCSession', data);
}

// RTCSession.js
_newRTCSession(originator, request)
{
  logger.debug('newRTCSession()');
  this._ua.newRTCSession(this, {
    originator,
    session : this,
    request
  });
}
```

  目光再回到RTCSession的connect方法上，connect方法最后一行调用了_sendInitialRequest()方法，这个方法实例化了RequestSender，在实例化参数中订阅了`onRequestTimeout`、`onTransportError`、`onAuthenticated`、`onReceiveResponse`四个事件，先记住onReceiveResponse事件注册的回调_receiveInviteResponse方法。

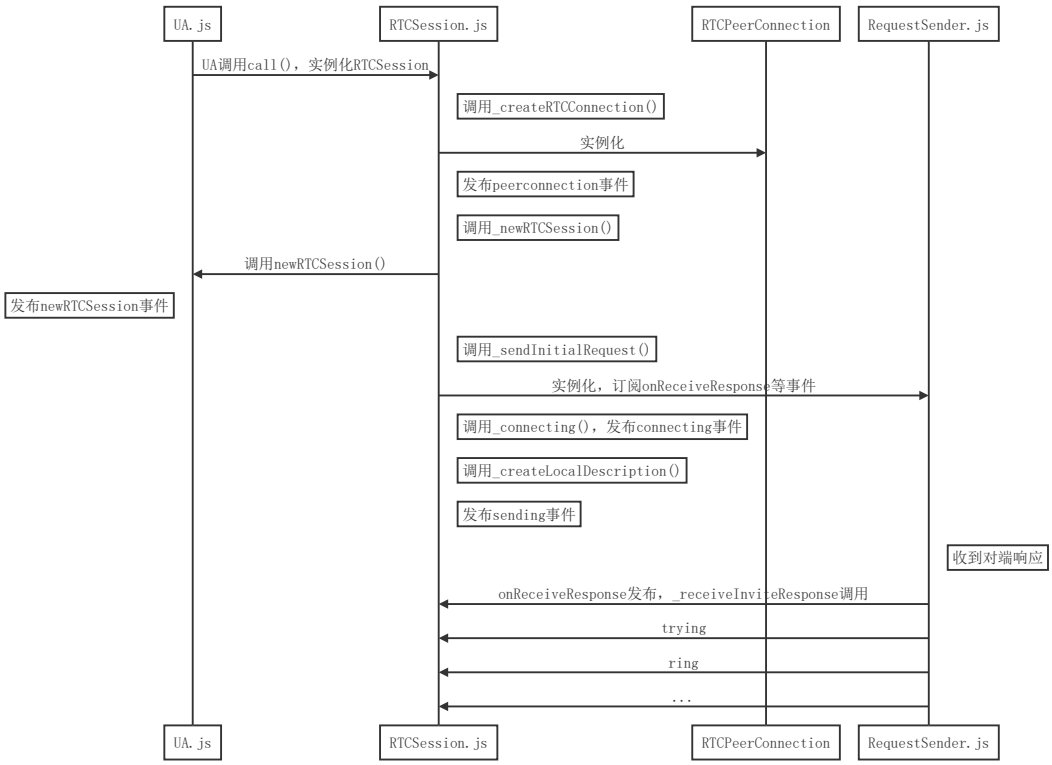
实例化RequestSender后会调用_connecting()，该方法中发布`connecting`事件。随后调用_createLocalDescription()，在这个方法里创建了Offer，并订阅了许多有关sdp、peerconnection、icecandidate的事件，当这些事件触发，对应的将事件继续发布给外部程序。

_createLocalDescription()调用之后会发布`sending`事件，并发送请求。

至此一通电话就送出去了，当对端响应了我们的邀请，onReceiveResponse就被触发并且调用_receiveInviteResponse()，这个方法里处理了对端trying、ring、accept等事件，RTCPeerConnection的setRemoteDescription也是在这里设置的。

让我们来屡一下事件发生顺序，首先是UA的`newRTCSession`最早被发布，随后是`peerconnection`，然后是`connecting`，`sending`，`icecandidate`，`sdp`，最后是`accept`等其他需要操作才能触发的事件。

下图是方法调用顺序，不代表事件发生顺序:



- 呼入

  在实例化UA时调用的_loadConfig方法中订阅了web scoket的ondata事件，订阅的方法为onTransportData()，message如果是SIPMessage.IncomingRequest的实例，会调用UA对象的receiveRequest()方法。

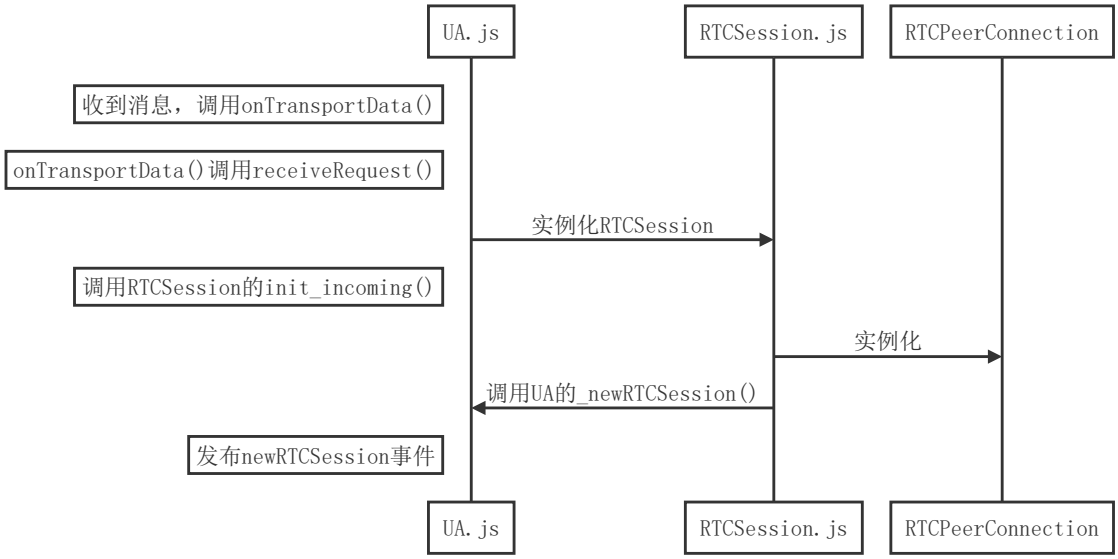
  在receiveRequest()方法中如果是INVITE包会新建一个事务，随后在后面的判断中实例化RTCSession对象,并调用init_incoming()方法.

  在RTCSession的init_incoming()方法会调用_newRTCSession方法，最终发布UA的`newRTCSession`事件。

  最后调用_progress方法，发布`progress`事件。

  这里省略中间协商消息...

  当协商完成后，外部程序就可调用answer方法接听该通来电。



如何拓展JsSIP

在https://github.com/versatica/JsSIP/issues/708 这个issue中提出JsSIP中没有实现rfc6665中的SUBSCRIBE/NOTIFY方法，于是问题提出者自己在JsSIP基础上实现了它。

下面跟随他的代码和JsSIP作者José在这个issue中提到的拓展JsSIP应遵循的代码规范了解如何拓展一个模块。

仓库地址：https://github.com/ikq/JsSIP/tree/subscribe_support

José提出的要求总结如下：

- 遵循现有的代码格式
- 需要根据rfc文档进行拓展
- 事件发布订阅使用`EventEmitter`，与其他模块保持一致
- 新增的模块使用独立的JS文件

代码实现，在与UA.js同级的目录下可以找到Subscriber.js和Notifier.js。

在UA中新增两个方法subscribe和notify，分别实例化Subscribe和Notifer对象，代码如下：

```
/**
 * Create subscriber instance
 */
subscribe(target, eventName, accept, options)
{
  logger.debug('subscribe()');

  return new Subscriber(this, target, eventName, accept, options);
}

/**
 * Create notifier instance
 */
notify(subscribe, contentType, options)
{
  logger.debug('notify()');

  return new Notifier(this, subscribe, contentType, options);
}
```

注意，这两个方法会将实例化后的Subcriber和Notify对象return，需要外部程序接收，实例不会挂载到UA上（当然你也可以自行改造），代码示例如下：

```
// 使用notifier变量接收Notify实例
const notifier = jssipUA.notify(subscribe, contentType, { pending });
```

小技巧，通过.d.ts文件可以快速了解这Subcriber和Notify对外暴露的方法有哪些，代码如下：

```
// Subscriber.d.ts
export class Subscriber extends EventEmitter {
  subscribe(body?: string): void;   // 发送SUBSCRIBE请求
  terminate(body?: string): void;   // 停止订阅
  get state(): string;
  get id(): string;
  static get C(): typeof SubscriberTerminatedCode;
  get C(): typeof SubscriberTerminatedCode;
}

// Notifier.d.ts
export class Notifier extends EventEmitter {
  start(): void;  // 根据代码注释，这个方法主要是订阅事件，接收发来的通知
  setActiveState(): void; // 切换active和pending状态
  notify(body?: string): void;  // 发送通知
  terminate(body?: string, reason?: string): void; // 发送最终通知
  get state(): string;
  get id(): string;
  static get C(): typeof NotifierTerminatedCode;
  get C(): typeof NotifierTerminatedCode;
}
```

Subcriber和Notify使用方法可以参考https://github1s.com/ikq/subscribe_notify_test/blob/HEAD/test.js 。