4-SRS 4.0支持WebRTC一对一通话

音视频高级课程：https://ke.qq.com/course/468797?tuin=137bb271

1 SRS4.0一对一通话简介

SRS 使用的是外置信令：

- SRS负责媒体能力

- 外置信令服务器负责房间管理

官方文档参考地址：https://github.com/ossrs/srs/wiki/v4_CN_WebRTC#sfu-one-to-one

signaling（信令）和httpx-static（web访问）这两个项目，代码都放在了SRS的3rdparty目录，不依赖网络就可以编译（依赖Go环境）。

2 环境搭建

2.1 安装go语言环境

下载和解压Go安装包

在Go语言官网找到对应的安装包（https://golang.google.cn/dl/），但是先不要急着下载。
提示：阅读本节需要对 Linux 系统及常用的命令有一定的了解。

go1.16.3 ▾

File name	Kind	OS	Arch	Size	SHA256 Checksum
go1.16.3.src.tar.gz	**Source**			**20MB**	b298d29de9236ca47a023e382313bcc2d2eed31dfa706b60a04103ce83a71a25
go1.16.3.darwin-amd64.tar.gz	Archive	macOS	x86-64	124MB	6bb1cf421f8abc2a9a4e39140b7397cdae6aca3e8d36dcff39a1a77f4f1170ac
go1.16.3.darwin-amd64.pkg	**Installer**	**macOS**	**x86-64**	**125MB**	af29670bff9a1f9c078b1f3b027a8cbc006f6044baaafc7dd32416a374dd6248
go1.16.3.darwin-arm64.tar.gz	Archive	macOS	ARMv8	120MB	f4e96bbcd5d2d1942f5b55d9e4ab19564da4fad192012f6d7b0b9b055ba4208f
go1.16.3.darwin-arm64.pkg	**Installer**	**macOS**	**ARMv8**	**120MB**	ff26b39bd2a5ebb6416061eaa41e59b44f02199cbb2442f5e217c722ffe6db91
go1.16.3.linux-386.tar.gz	Archive	Linux	x86	98MB	48b2d1481db756c88c18b1f064dbfc3e265ce4a775a23177ca17e25d13a24c5d
go1.16.3.linux-amd64.tar.gz	**Archive**	**Linux**	**x86-64**	**123MB**	951a3c7c6ce4e56ad883f97d9db74d3d6d80d5fec77455c6ada6c1f7ac4776d2
go1.16.3.linux-arm64.tar.gz	Archive	Linux	ARMv8	95MB	566b1d6f17d2bc4ad5f81486f0df44f3088c3ed47a3bec4099d8ed9939e90d5d
go1.16.3.linux-armv6l.tar.gz	Archive	Linux	ARMv6	96MB	0dae30385e3564a557dac7f12a63eedc73543e6da0f6017990e214ce8cc8797c
go1.16.3.windows-386.zip	Archive	Windows	x86	112MB	a3c16e1531bf9726f47911c4a9ed7cb665a6207a51c44f10ebad4db63b4bcc5a
go1.16.3.windows-386.msi	Installer	Windows	x86	98MB	75c501ce9c7e542653da034db02d8d9dc6cecef2804df9fc9cc6f90708a02d8c
go1.16.3.windows-amd64.zip	Archive	Windows	x86-64	137MB	a4400345135b36cb7942e52bbaf978b66814738b855eeff8de879a09fd99de7f
go1.16.3.windows-amd64.msi	**Installer**	**Windows**	**x86-64**	**119MB**	850cf9e4b0ab0369ab2750c6ab25725b6a298490d09bf3fde61b08f770328f4c

注意：开发包有 32 位和 64 位两个版本，需要根据读者电脑的情况选择不同的版本。
接下来将带领大家一步步的完成安装过程。

```
cd /usr/local/
wget https://dl.google.com/go/go1.16.5.linux-amd64.tar.gz --no-check-certificate
tar -C /usr/local -xzf go1.16.5.linux-amd64.tar.gz
```

配置环境变量

我们需要配置 2 个环境变量分别是 GOROOT 和 PATH。

- GOROOT 的值应该为Go语言的当前安装目录：
  export GOROOT=/usr/local/go
- PATH 为了方便使用Go语言命令和 Go 程序的可执行文件，需要追加其值：
  export PATH=$PATH:$GOROOT/bin:$GOBIN

为了方便以后的使用，需要把这几个环境变量添加 profile 文件中（~/.bash_profile 或 /etc/profile）。如果是单用户使用，可以将环境变量添加在 home 目录下的 bash_profile 文件中，如果是多用户使用，需要添加在 /etc/profile 文件。（推荐大家在 /etc/profile 文件中设置环境变量）

使用`vim /etc/profile`命令打开 profile 文件，并将环境变量添加到文件末尾。

添加完成后使用`:wq`命令保存并退出。

然后，使用`source /etc/profile`命令使配置文件生效，现在就可以在任意目录使用Go语言命令了。

2.2 编译和启动srs

```
git clone -b v4.0.123 https://gitee.com/winlinvip/srs.oschina.git srs.4.0.123
cd srs.4.0.123/trunk
./configure
make
./objs/srs -c conf/rtc.conf
```

2.3 编译和启动信令服务器

```
cd 3rdparty/signaling
make
./objs/signaling
```

使用端口：1989

2.4 编译和启动web服务器

这里需要server.crt和server.key，如果没有则用openssl生成。

编译和启动httpx-static for HTTPS/WSS

```
cd 3rdparty/httpx-static
make
./objs/httpx-static -http 80 -https 443 -ssk server.key -ssc server.crt \
    -proxy http://127.0.0.1:1989/sig -proxy http://127.0.0.1:1985/rtc \
    -proxy http://127.0.0.1:8080/
```

- 1935端口对应的是rtmp服务
- 1985对应的是http api服务，进一步学习：https://github.com/ossrs/srs/wiki/v4_CN_HTTPApi
- 8080对应的是http-flv、hls的服务器端口

生成 server.key和server.crt

生成 server.key

> openssl genrsa -out server.key 2048

生成 server.crt

> openssl req -new -x509 -key server.key -out server.crt -days 3650

进入测试页面

https://120.27.131.197/demos/

注意：不是 https://120.27.131.197/**demos**

打开demo地址 HTTPS or IP:

- http://localhost/demos/
- https://localhost/demos/
- https://192.168.3.6/demos/

3 逻辑分析

srs流媒体服务器

对应的请求端口1985

发布流

发布信令对应的data示例：


```
{    "api": "http://114.215.169.66:1985/rtc/v1/publish/",
    "tid": "1793820032d",
    "streamurl": "webrtc://114.215.169.66/live/livestream",
    "clientip": null,
    "sdp": "v=0\no=- 170307602475242460 2 IN IP4 127.0.0.1\ns=-\nt=0 0\na=group:BUNDLE 0..\n"
}
```

播放流

播放信令对应的data示例：

{ "api": "http://114.215.169.66:1985/rtc/v1/play/",

"tid": "17938243c61",

"streamurl": "webrtc://114.215.169.66/live/livestream",

"clientip": null,

"sdp": "v=0\no=– 8914052225280550591 2 IN IP4 127.0.0.1\ns=–\nt=0 0\na=group:BUNDLE... \n"

}

信令服务器

对应的请求端口1989，接口/sig/v1/rtc

信令：

- join 加入房间
- publish 发布
- control 控制

服务器结构体

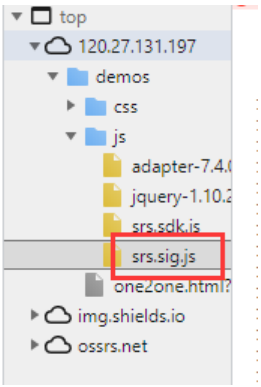
```
// 参与者
type Participant struct {
```

```
    Room       *Room       `json:"-"`           // 所属房间
    Display    string      `json:"display"`    // 显示
    Publishing bool        `json:"publishing"` // 显示名
    Out        chan []byte `json:"-"`           //
}
```

```
// 房间
type Room struct {
    Name         string         `json:"room"`         // 房间名
    Participants []*Participant `json:"participants"` // 所有参与者
    lock         sync.RWMutex   `json:"-"`            // 锁
}
```

客户端函数

srs.sig.js



SrsRtcSignalingAsync：

- connect：连接信令服务器 wss://xxxx/sig/v1/rtc
- onmessage：处理接收到的消息
- send ： 把对象序列化后发送到服务器
- close： 关闭

SrsRtcSignalingParse：

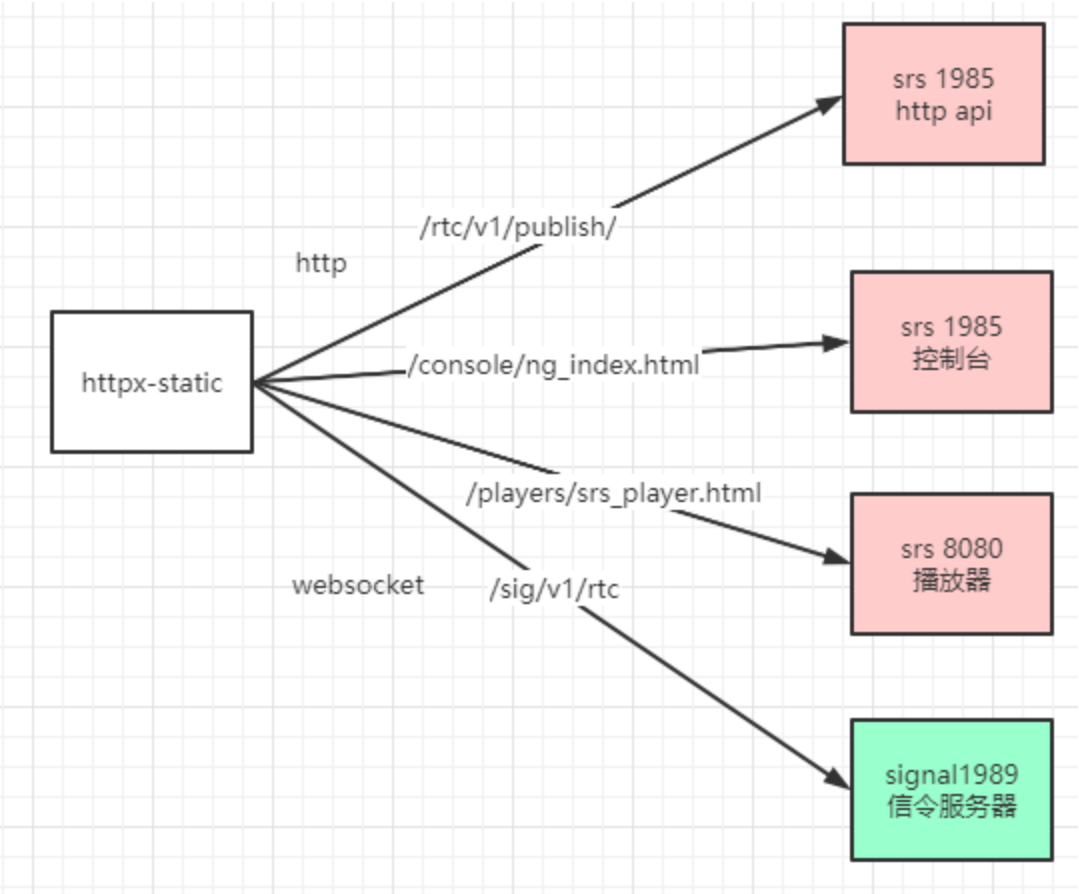
- 解析出连接信令服务器的地址
- 房间号：可填或随机
- 显示名：可填或随机
- 主机

比如下以下所示：


```
{    "query": "",
    "rawQuery": "?autostart=true&room=4a02515",
    "wsSchema": "wss",
    "wsHost": "120.27.131.197",
    "host": "120.27.131.197",
    "room": "4a02515",
    "display": "4186e4f",
    "autostart": true
}
```

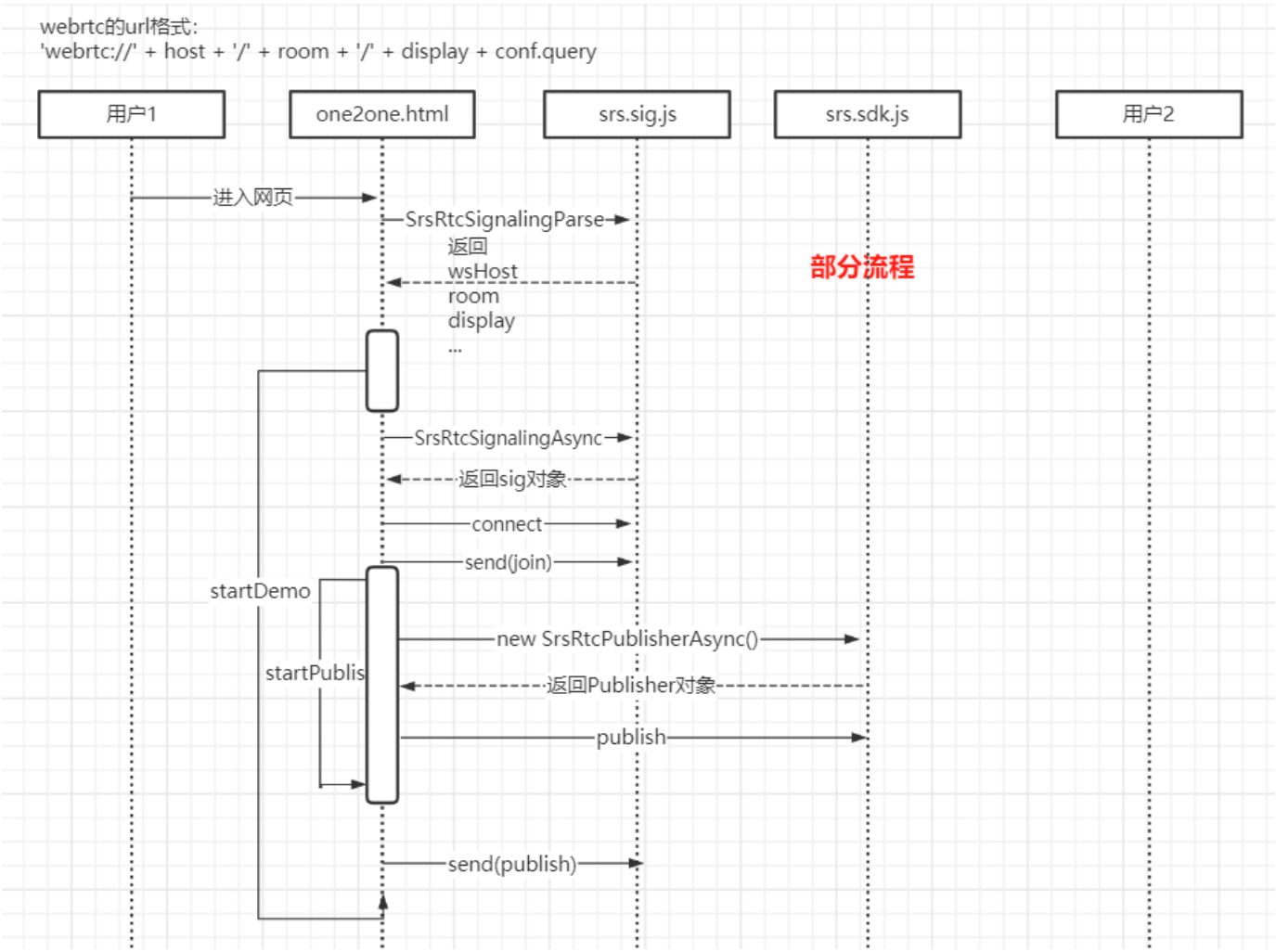

web服务器

httpx-static做了代理



客户端函数

见图分析


webrtc的url格式:
'webrtc://' + host + '/' + room + '/' + display + conf.query
用户1
one2one.html
srs.sig.js
srs.sdk.js
用户2
进入网页
SrsRtcSignalingParse
返回
wsHost
room
display
...
部分流程
SrsRtcSignalingAsync
返回sig对象
connect
send(join)
startDemo
new SrsRtcPublisherAsync()
startPublis
返回Publisher对象
publish
send(publish)