- 技术栈
- 目录结构
- API接口
- 路由管理
- 接口返回状态码说明
    - 导入SDK
    - Im使用
        - 1、创建Im实例 `imApp.getInstance()`
        - 2、解密（消息历史记录是密文返回的） `imApp.decrypt()`
    - Im.socket相关操作
        - 1、websocket 绑定用户 `imApp.socket.bindUser()`
        - 2、websocket 获取当前连接的状态 `imApp.socket.getState()`
        - 3、websocket 关闭服务 `imApp.socket.close()`
        - 4、websocket 更新配置 `imApp.socket.updateConfig()`
    - 用户、商家应用场景
    - 客服应用场景

技术栈

`Node.js` `webpack` `Vue` `vuex` `vue-router` `axios` `ES6`

注：相关版本号可进入 `/package.json` 查看

目录结构

- `/src` *主开发目录*
  - `/assets` *存放静态文件*
  - `/components` *存放公用组件*
    - `/ChatList` *对话列表版块组件*
    - `/Editor` *输入版块组件*
    - `/Toast` *消息提示组件*
    - `/MemberList` *群成员列表版块组件*
  - `/http` *路由管理*
    - `api.js` *API接口列表*
    - `index.js` *http请求对象封装*
  - `/lib` *库*
    - `im-sdk.min.js` *SDK文件*
  - `/router` *路由管理*
  - `/store` *状态管理*
  - `/utils` *工具文件目录*
  - `/views` *视图组件*
    - `/groupConversation` *对话窗口*
    - `/createGroup` *模拟B2C创建群* `生产环境可删除`
    - `/login` *模拟B2C登录* `生产环境可删除`
  - `App.vue`
  - `main.js` * 入口文件*
- `/public` 存放静态文件(不打包)
  - `favicon.ico` *浏览器图标*
  - `index.html` *html模板文件*
  - `test.html` *模拟客服登录* `不用可删除`

API接口

注：`接口默认使用当前域名` 如果非当前域名，可进入`/src/http/index.js`中配置，如下：

```
...
axios.defaults.baseURL = ''
...
```

路由管理

** 说明 **

- 配置目录：`/src/router`

接口返回状态码说明

接口返回数据结构

```
{
    code: 0, // 状态码
    data: {}, // 数据体
    msg: '' // 提示消息
}
```

状态码code	说明
0	成功
1	失败
401	未授权，会自动跳转到登录

注：`401`未授权会自动跳转到登录，跳转地址可进入`/src/http/index.js`文件处理,示例如下：

```
// 添加响应拦截器
axios.interceptors.response.use(
    res => {
        // 请求成功后移除记录
        removePendingRequestRecord(res.config)
        // 公用返回码处理
        switch(res.data.code){
            case 401:
                // 未授权处理
                utils.removeSession('userInfo')
                router.push('/login/logout')
                return Promise.reject(res)
        }
    ...
}
```

导入SDK

SDK放在项目的`/src/lib/im-sdk.min.js`路径下

```
// 非 ESModule
const Im = require('src/lib/im-sdk.min.js').default
// ESModule
import Im from 'src/lib/im-sdk.min.js'
```

Im使用

以下的`imApp`都是页面加载时创建的唯一应用：`var imApp = new Im();`

1、创建Im实例 `imApp.getInstance()`

新建Im实例，传入相应的回调事件

```
imApp.getInstance({
    wsAddr: data.wsAddress,          // websocket地址
    dlgKey: data.dlgKey,             // 解密的secretKey
    dlgId: data.dlgId,               // 群ID
    user: {
        fromuid: data.userId, // b2c对应Im的userId
    },
    // 订阅事件
    handler: {
        onConnect: function(e){},          // 连接成功的回调
        onBindUserSuccess: function(e){}, // 绑定用户成功的回调
        onMessage: function(e){},          // 收消息的回调
        onClose: function(e){},            // 关闭连接的回调
        onError: function(e){},            // 连接出错的回调
        onReConnect: function(e){}         // 需要重连的回调
    }
})
```

使用详细可查看示例：`/src/views/groupConversation`

2、解密（消息历史记录是密文返回的）`imApp.decrypt()`

```
// content 密文
// dlgKey  群会话的key
// dlgId   群会话ID
imApp.decrypt(item.content, dlgId, dlgKey)
```

Im.socket相关操作

1、websocket 绑定用户 `imApp.socket.bindUser()`

`websocket` 连接成功后可进行绑定用户的操作

```
// 平台对应的Im账号的信息
let param = {
    avatar: avatar
    token: imToken,
    name: name,
    remark: remark,
    uid: userId
}
imApp.socket.bindUser(param)
```

2、websocket 获取当前连接的状态 `imApp.socket.getState()`

```
imApp.socket.getState() // 连接：true，未连接：false
```

3、websocket 关闭服务 `imApp.socket.close()`

```
if (imApp && imApp.socket && imApp.socket.getState()) {
    imApp.socket.close()
}
```

4、websocket 更新配置 `imApp.socket.updateConfig()`

```
imApp.socket.updateConfig({
    dlgKey: dlgKey, // 群会话的key
    dlgId: dlgId    // 群会话ID
})
```

用户、商家应用场景

操作流程

- 1、B2C平台拿到用户`Im`账号信息自动拼接一个会话链接
- 2、用户、商家点击链接直接进入群会话

连接示例及参数说明 `http://xxxx/groupConversation?groupId=xxx&userId=xxx`

名称	说明
userId	平台用户对应im账号的useId
groupId	群ID

注：模拟例子可查看`/src/views/createGroup`示例

客服应用场景

`注：`

- 进入客服页面，默认已经登录了平台的账号,并拿到了Im账号的信息
- 会话窗口通过`iframe`方式引入

```
<iframe id="myFrame" src="http://xxxx/groupConversation"
frameborder="0"></iframe>
```

- 模拟示例可以看项目的`/test.html`

操作流程如下：

1、以当前客服用户创建websocket会话，通过`window.postMessage`方式向iframe发送消息`{action, payload}`

```
let serviceInfo = {
    token,   // token
    userId,  // Im用户ID
    urole    // 用户角色
}
// 传递客服用户信息，以当前客服建立websocket会话
sendToIframe({
    action: 'transferUser',
    payload: serviceInfo
})
// 向iframe发消息公用函数
function sendToIframe({ action, payload }) {
    chatIfr = document.getElementById('myFrame')
    chatIfr.contentWindow.postMessage({
        action, payload
    }, '*')
}
```

2、iframe绑定用户建立会话后回调处理

```
// 监听iframe发过来的消息
window.addEventListener('message', receiveFromIframe, false)
// 绑定成功回调
function receiveFromIframe(event) {
    if (event.data && event.data.action) {
        switch(event.data.action) {
            case 'bindUserSuccess':
                // 绑定访客成功，需要的话做一些操作
                break
        }
    }
}
```

3、调用接口`getDlgPage`获取会话列表

4、点击对应的会话、通过`window.postMessage`方式向iframe发送消息`{action, payload}`

```
let selectedConversation = {
    dlgId,  // 群会话ID
    dlgKey, // 群会话Key
    name    // 群名称
}
// 选择群进行对话
sendToIframe({
    action: 'selectConversation',
    payload: selectedConversation
})
```