概要介绍

产品综述

网易作为国内最大最早的即时通信开发商之一，从最早的网易泡泡到后来的易信，已经有了超过十五年的通讯技术积累。现在我们整合了这些产品高稳定、高可靠即时通信能力，开发者能以很小的成本将即时通信功能集成到自己的 APP 中。

网易云信 SDK (NIM SDK) 为PC/移动Web应用提供完善的即时通信功能开发框架，屏蔽其内部复杂细节，对外提供较为简洁的 API 接口，方便第三方应用快速集成即时通信功能。SDK 兼容 IE9+(音视频部分为IE10及以上)、Edge、Chrome 58+、 Safari 10+、Firefox 54+等主流桌面版浏览器，兼容iPhone 5s以上机型(操作系统iOS 8.0+)的Safari浏览器及其内置微信浏览器、主流机型Android 5+系统的Chrome浏览器及其内置微信浏览器。

若需要支持IE8浏览器，请联系商务获取v5.0.0以下版本的web sdk

网易云信还开发了可供开发者们参考，如何使用该SDK的Web Demo：

Web Demo 源码导读

Web Demo (移动端H5)源码导读 (不包含音视频)

Web Demo (微信小程序 IM部分)源码

Web Demo (ReactNative IM部分)源码

业务介绍

点对点聊天

即一对一单聊，网易云信 SDK 提供文字、图片、语音、地理位置、文件、自

定义消息等多种能力，开发者可根据自身需求定制附件下载、未读、推送等多种消息行为。

群聊天

即多人聊天群组服务，内置普通群和高级群，普通群类似于没有权限的讨论组，适用于快速创建多人会话的场景；高级群在普通群的基础上拥有了更多的权限设置，包括邀请的验证方式，管理员权限，禁言等更高级的功能，适用于更复杂更定制化的场景。

群聊天有人数限制，人数在千级别以上推荐使用聊天室。

聊天室聊天

聊天室是群聊人数在万级甚至更多的聊天解决方案，可用于游戏直播，网络授课，远程医疗等业务场景上。聊天室没有人数限制，同时提供基本的权限服务。

聊天室在进入时需要重新建立链接，同时由于场景消息量通常很大， SDK不会做聊天室消息存储，需要上层自行实现。 在人数低于千级别时，推荐使用群组聊天。

资料托管

网易云通信提供了用户资料的可选托管，开发者可以根据喜好，将用户资料托管给网易云信或者 APP 应用服务器自行管理。

事件订阅

网易云信允许用户订阅监听其他用户产生的事件，产生的事件的方式分为两种：

用户主动发布的自定义事件。
由于用户的一些特定行为触发的内置系统事件。

开发者可以用事件订阅自定义用户的在线状态，如忙碌，隐身等等。

快速体验

云信SDK以方便、敏捷、稳定为宗旨，服务开发者，具备快速接入的特性，以下为最小应用示例：

```
    <h1>云信 Web SDK Online Test</h1>
    <div>
    <p>
        <span>APPKEY:</span>
    <input type="text" id="appkey" value="45c6af3c98409b18a84451215
d0bdd6e">
    </p>
    <p>
        <span>ACCOUNT:</span>
        <input type="text" id="account" value="greatcs4">
    </p>
    <p>
        <span>TOKEN:</span>
        <input type="text" id="token" value="e10adc3949ba59abbe56e0
57f20f883e">
    </p>
    <p>
        <button id="connect-sdk">连接SDK</button>
    </p>
    </div>
    <script src="http://yx-web.nos.netease.com/official/websdk/NIM_
Web_SDK_v4.8.0.js"></script>
```

```
function connectSDK () {
  var appkey = document.getElementById('appkey').value
  var account = document.getElementById('account').value
  var token = document.getElementById('token').value
  window.nim = SDK.NIM.getInstance({
    appKey: appkey,
    account: account,
    token: token,
    onconnect: function () {
      alert('SDK 连接成功')
      // 连接成功后才能发消息
      window.nim.sendText({
        scene: 'p2p',
        to: 'account',
        text: 'hello',
        done: function sendMsgDone (msg) {
        }
      })
    },
    ondisconnect: function (obj) {
      console.log('SDK 连接断开', obj)
    },
    onerror: function (error) {
      console.log('SDK 连接失败', error)
    }
  })
}

document.getElementById('connect-sdk').addEventListener('click', fu
nction () {
  connectSDK()
})
```

推荐一个免费的在线编辑工具，开发者也可以在上面编辑调试相应代码:
CodePen

集成方式

NIM SDK 需要您通过手动下载 SDK, 然后添加到您的项目中。

下载SDK

我们提供两个SDK获取方式。分别为：

官网 SDK 下载入口。

向客户经理咨询并索要尝鲜版Web功能

SDK 目录结构介绍

如果要在浏览器里面使用 SDK, 相应的 JS 文件都在 `js` 目录下.

引入SDK

SDK选择

如果要使用 IM 功能, 请引入 `NIM_Web_NIM_v.js`

请通过 `NIM` 来获取引用

如果要使用聊天室功能, 请引入 `NIM_Web_Chatroom_v.js`

请通过 `Chatroom` 来获取引用

如果同时使用 IM 和聊天室功能, 请引入 `NIM_Web_SDK_v.js`

如果通过 script 标签引入, 请通过 `SDK.NIM` 和 `SDK.Chatroom` 来获取 `NIM` 和 `Chatroom` 的引用, 下文中的 API 都是通过 `NIM` 和 `Chatroom` 来调用的

如果要使用插件版实时音视频功能, 请引入 `NIM_Web_Netcall_v.js`, 通过 `Netcall` 来获取引用, 调用 `NIM.use(Netcall)` 来加载实时音视频插件

如果要使用WebRTC实时音视频功能, 请引入 `NIM_Web_WebRTC_v.js`, 通过 `WebRTC` 来获取引用, 调用 `NIM.use(WebRTC)` 来加载实时音视频插件

SDK引入方式

script 标签方式引用
将所需的SDK脚本，通过传入script标签的src中
在下文中使用window对象属性即可获取对sdk的引用

```
<!-- 例如 -->
<script src="NIM_Web_SDK.js">
<script>
  var nim = SDK.NIM.getInstance({
    // ...
  })
</script>
```

import/require 方式引用
直接使用import/require方式获取引用
如果开发者选用 webpack/babel 来打包, 那么请使用 exclude 将 SDK 文件排除, 避免 babel 二次打包引起的错误

SDK在工程中调用示例：

```
  // 使用示例
  import SDK from 'NIM_Web_SDK.js'
  const nim = SDK.NIM.getInstance({
    // ...
  })
```

相对应Webpack配置

```
// Webpack 参考配置
module: {
  rules: [
    {
      test: /\.js$/,
      loader: 'babel-loader',
      exclude: /NIM_Web_SDK.*\.js/,
      query: {
        presets: [
          // ...
        ],
        // ...
      }
      // ...
    },
    // ...
  ],
  // ...
}
```

使用说明

实例调用方式

所有业务均通过 NIM SDK 单例调用

```
// 引入SDK类的引用以后，获取SDK对象实例
var nim = SDK.NIM.getInstance({
  // ...
})
```

以发送点对点消息为例：

```
var msg = nim.sendText({
  scene: 'p2p',
  to: 'account',
  text: 'hello',
  done: function sendMsgDone (error, msg) {
    // ...
  }
})
```

事件通知方式

SDK 通过两种方式通知上层 API 调用结果：回调(callback)和委托 (delegate)，两种方式都只在主线程触发(为保证浏览器兼容性，没有使用web worker)。

一般回调接口直接反映在对应接口的 `done` 参数上，调用时设置即可。
委托则需要开发者在合适时机在初始化异步监听函数上进行处理

```
// 委托通知示例
var nim = NIM.getInstance({
  // ... 此处省略其他配置
  onmsg: function (msg) {
    // 此处为委托消息事件，消息发送成功后，成功消息也在此处处理
  }
})
// 回调通知示例
var msg = nim.sendText({
  scene: 'p2p',
  to: 'account',
  text: 'hello',
  done: function sendMsgDone (error, msg) {
    // 此处为回调消息事件，仅仅通知开发者，消息是否发送成功
  }
})
```

浏览器兼容性

云信 Web SDK (不包含实时音视频)兼容到 IE8
- IE8/IE9 需要将项目部署在 HTTPS 环境下才能连接到云信服务器, 其它高级浏览器可以在 HTTP 或者 HTTPS 环境下连接到云信服务器

数据库兼容性

在支持数据库的浏览器上 SDK 会将数据缓存到数据库中, 后续同步都是增量更新, 加快初始化速度。

Web SDK在底层使用了indexedDb，作为数据库进行存储。由于浏览器兼容性及其本身设计上的限制/缺陷（如读写库性能问题、数据空间、没有数据库表级别锁、不能只修改对象单个字段、非阻塞等等问题），用户场景没有强烈需要本地数据存储的应用，**不建议开启数据库**。

是否支持数据库

```
// 通过此 `boolean` 值来查看 SDK 在某个浏览器上是否支持数据库
NIM.support.db
```

其原理是结合了JavaScript"鸭子类型"探测和UserAgent识别，做判断，所以非法改写的UserAgent可能会导致报错。

不使用数据库

如果开发者不想使用数据库, 那么可以设置初始化参数`db`为`false`来禁用数据库

```
var nim = NIM.getInstance({
    db: false
});
```

依赖说明

SDK 使用一系列开源库来更好的完成工作, 所有库均挂在 NIM 下面
SDK 使用 es5-shim 来让低版本浏览器兼容 ES5 的部分方法
SDK 使用 platform.js 来检测浏览器平台, 通过 `NIM.platform` 来获取此库的引用
SDK 使用 socket.io-client 0.9 来建立 Socket 连接, 通过 `NIM.io` 或 `window.io` 来获取此库的引用

初始化 SDK

请查阅集成方式#引入 SDK来下载并引入 SDK 文件

示例代码

`NIM.getInstance`接口为单例模式, 对于同一个账号, 永远返回同一份实例, 即只有第一次调用会初始化一个实例
后续调用此接口会直接返回初始化过的实例, 同时也会调用接口更新 IM 配置更新传入的配置
后续调用此接口时, 如果连接已断开, 会自动建立连接
当发生掉线时，SDK 会自动进行重连
开发者在收到`onconnect`回调之后代表链接已经建立(登录成功), 此时 SDK 会开始同步数据, 随后在收到`onsyncdone`回调之后表示 SDK 完成了数据同步工作, 此时开发者可以进行渲染 UI 等操作了。
这里的`data`代表数据, 在后面章节的示例代码中会多次用到这个对象
这里的`nim`代表 SDK, 在后面章节的示例代码中会多次用到这个对象.
这里的参数并不是所有的初始化参数, 请查阅其它章节的初始化参数
初始化 SDK
连接初始化参数
多端登录初始化参数
消息初始化参数
群组初始化参数
用户资料初始化参数
好友关系初始化参数
用户关系初始化参数
会话初始化参数
系统通知初始化参数
同步完成
完整的初始化代码

```
var data = {};
// 注意这里，引入的 SDK 文件不一样的话，你可能需要使用 SDK.NIM.getInstanc
e 来调用接口
var nim = NIM.getInstance({
  // debug: true,
  appKey: 'appKey',
  account: 'account',
  token: 'token',
  // privateConf: {}, // 私有化部署方案所需的配置
  onconnect: onConnect,
  onwillreconnect: onWillReconnect,
  ondisconnect: onDisconnect,
  onerror: onError
});
function onConnect() {
  console.log('连接成功');
}
function onWillReconnect(obj) {
  // 此时说明 SDK 已经断开连接，请开发者在界面上提示用户连接已断开，而且正在
重新建立连接
  console.log('即将重连');
  console.log(obj.retryCount);
  console.log(obj.duration);
}
function onDisconnect(error) {
  // 此时说明 SDK 处于断开状态，开发者此时应该根据错误码提示相应的错误信息，
并且跳转到登录页面
  console.log('丢失连接');
  console.log(error);
  if (error) {
    switch (error.code) {
      // 账号或者密码错误，请跳转到登录页面并提示错误
      case 302:
        break;
      // 重复登录，已经在其它端登录了，请跳转到登录页面并提示错误
      case 417:
        break;
      // 被踢，请提示错误后跳转到登录页面
      case 'kicked':
        break;
      default:
        break;
    }
  }
}
function onError(error) {
  console.log(error);
}
```

参数解释

debug: 是否开启日志, 开发者可以开启日志, 这样 SDK 会将关键操作的信息打印到控制台上, 便于调试
logFunc: 日志分析函数。默认情况下，开发者使用云信 SDK 会将日志打印到 console 中，但在诸如移动端、混合应用场景，对错误的跟踪与查询并不是很方便。SDK 提供了logFunc配置参数，可截获日志信息，输入到用户自定义的函数中（sdk 也提供了 plugin 供开发者参考）。辅助开发在在混合应用/PC 端/移动端浏览器上对信息做二次处理。
appKey: 在云信管理后台查看应用的 appKey
account: 帐号, 应用内唯一
token: 帐号的 token, 用于建立连接
transports: 用于建立长连接的协议数组，可不填，默认为['websocket', 'xhr-polling']
默认状态 sdk 优先使用 websocket 连接，如果浏览器不支持 websocket，则使用 xhr-polling
开发者可手动设置连接及顺序，可支持选项包括 websocket、xhr-polling、flashsocket
示例如： transports: ['websocket'、'xhr-polling'、'flashsocket']
privateConf: 私有化部署所需的配置，可通过管理后台配置并下载相应的js文件，将相应内容引入配置
onconnect: 连接建立后的回调(登录成功), 会传入一个对象, 包含登录的信息, 有以下字段
lastLoginDeviceId: 上次登录的设备的设备号
connectionId: 本次登录的连接号
ip: 客户端 IP
port: 客户端端口
country: 本次登录的国家
onwillreconnect: 即将重连的回调
此时说明 SDK 已经断开连接, 请开发者在界面上提示用户连接已断开, 而且正在重新建立连接
此回调会收到一个对象, 包含额外的信息, 有以下字段
duration: 距离下次重连的时间

retryCount: 重连尝试的次数
ondisconnect: 断开连接后的回调
此时说明 SDK 处于断开状态, 开发者此时应该根据错误码提示相应的错误信息, 并且跳转到登录页面
此回调会收到一个对象, 包含错误的信息, 有以下字段
code: 出错时的错误码, 可能为空
302: 账号或者密码错误, 请跳转到登录页面并提示错误
417: 重复登录, 已经在其它端登录了, 请跳转到登录页面并提示错误
'kicked': 被踢
onerror: 发生错误的回调, 会传入错误对象

同步完成

SDK 在同步完成之后会通知开发者, 开发者可以在此回调之后再初始化自己的界面, 以及进行其他操作, 同步的数据包括下面章节中的

黑名单, 对应回调 onblacklist, 请参考用户关系托管里面的初始化参数
静音列表, 对应回调 onmutelist, 请参考用户关系托管里面的初始化参数
好友, 对应回调 onfriends, 请参考好友关系托管里面的初始化参数
我的名片, 对应回调 onmyinfo, 请参考用户资料托管里面的初始化参数
好友的名片, 对应回调 onusers, 请参考用户资料托管里面的初始化参数
群, 对应回调 onteams, 请参考群组功能里面的初始化参数
会话, 对应回调 onsessions, 请参考最近会话里面的初始化参数
漫游消息, 对应回调 onroamingmsgs, 请参考消息收发里面的初始化参数
离线消息, 对应回调 onofflinemsgs, 请参考消息收发里面的初始化参数
离线系统通知, 对应回调 onofflinesysmsgs, 请参考系统通知里面的初始化参数
离线自定义系统通知, 对应回调 onofflinecustomsysmsgs, 请参考系统通知里面的初始化参数

示例代码

这里的参数并不是所有的初始化参数, 请查阅初始化 SDK, 以及其它章节的初始化参数
连接初始化参数

```
var nim = NIM.getInstance({
  onsyncdone: onSyncDone
});
function onSyncDone() {
  console.log('同步完成');
}
```

初始化配置(同步协议)

SDK 默认对以下参数进行同步，对其他参数不进行同步

`syncRelations` 同步用户关系

`syncFriends` 同步好友关系

`syncFriendUsers` 同步好友对应的用户名片列表

`syncRoamingMsgs` 同步漫游消息(离线消息自动下推，不需要开关)

`syncMsgReceipts` 同步消息已读回执

`syncTeams` 同步群

`syncExtraTeamInfo` 同步额外群资料

开发者可以通过开关来选择同步或不同步可配置参数的同步数据, 这些开关都是初始化参数

所有可配置同步参数如下：

`syncRelations`, 是否同步黑名单和静音列表, 默认`true`. 如果传`false`就收不到黑名单和静音列表, 即不会收到`onblacklist`回调和`onmutelist`回调, 开发者后续可以调用获取黑名单和静音列表来获取黑名单和静音列表。

syncFriends, 是否同步好友列表, 默认true. 如果传false就收不到onfriends回调, 开发者后续可以调用获取好友列表来获取好友列表。
syncFriendUsers, 是否同步好友对应的用户名片列表, 默认true, 如果传false就收不到onusers回调.
syncTeams, 是否同步群列表, 默认true. 如果传false就收不到群列表, 即不会收到onteams回调, 开发者后续可以调用获取群列表来获取群列表.
syncExtraTeamInfo, 是否同步额外的群信息, 默认true会同步额外的群信息, 目前包括
当前登录用户是否开启某个群的消息提醒 (SDK 只是存储了此信息, 具体用此信息来做什么事情完全由开发者控制)
调用接口修改自己的群属性来关闭/开启某个群的消息提醒
调用接口是否需要群消息通知来查询是否需要群消息通知
syncTeamMembers, Deprecated, 4.4.0 版本以后废弃，是否同步群成员, 4.2.0 版本及以前默认true，4.3.0 版本及以后默认false，只有在syncTeams=true的时候才起作用, 如果传false就不会同步群成员, 即不会收到onteammembers和onsyncteammembersdone回调, 开发者可以调用获取群成员来按需获取群成员.
syncRoamingMsgs, 是否同步漫游消息, 默认true. 如果传false就收不到漫游消息, 即不会收到onroamingmsgs回调.
syncMsgReceipts, 是否同步已读回执时间戳, 默认true. 如果传false就收不到已读回执时间戳.
syncBroadcastMsgs, 是否同步广播消息, 默认false.
syncSessionUnread, 是否同步会话未读数(开启数据库时有效，保证多端未读数相一致)

示例代码

这里的参数并不是所有的初始化参数, 请查阅初始化 SDK, 以及其它章节的初始化参数
连接初始化参数
多端登录初始化参数
消息初始化参数
群组初始化参数
用户资料初始化参数
好友关系初始化参数

```
var nim = NIM.getInstance({
  syncRelations: false,
  syncBroadcastMsgs: true,
  onsyncRelationsdone: function(err, obj) {
    // ....
  }
});
```

初始化配置(其它协议)

`debug` 是否开启日志，参数为：true/false，开发者可以开启日志, 这样 SDK 会将关键操作的信息打印到控制台上, 便于调试

`db` 是否开启本地数据库存储数据，参数为：true/false，使用 indexedDb

`uploadUrl/downloadUrl` 私有化部署方案对文件处理专用接口，参考预览图片通用方法

`enabledHttpsForMessage` 针对 iOS 对 https 链接的要求而设计，若设置为 true，则所有文件类型消息的链接转为 https，现在所有使用云信 sdk 进行上传的文件链接均为 https。开发者一般不需要关注此项

`needReconnect` 连接失败时是否需要自动重连

`autoMarkRead` 对收到的消息，是否自动标记为已读

`shouldIgnoreNotification` 该参数类型为函数(function)，表示是否要忽略某条通知类消息。该方法会将接收到的通知类消息对象，按照用户上层定义的逻辑进行过滤, 如果该方法返回 true，那么 SDK 将忽略此条通知类消息

`shouldCountNotifyUnread` 该参数类型为函数(function)，群消息通知是否加入未读数开关，有上层对回调参数对象做分析，如果返回 true，则计入未读数，否则不计入

`ntServerAddress` 连接统计上报地址，设为`null`则不上报，用户可填写自己的日志上报地址，不填则默认为云信服务器的统计地址

微信小程序

SDK 提供微信小程序相关的支持
在云信websdk 5.0.0以后 正式对微信小程序多条websocket连接做了支持，底层所采用的是小程序SocketTask任务做的连接，参见socket-task，开发者在使用websdk时，需要将小程序基础库升级至1.7.0及以上版本，低版本sdk不再兼容。
若开发者需要在微信公众平台后台配置相应白名单，小程序文档链接
相关配置列表如下：
request 合法域名：
https://lbs.netease.im
https://wlnimsc0.netease.im
https://wlnimsc1.netease.im
https://dr.netease.im
https://nosdn.127.net
https://nim.nos.netease.com
https://nos.netease.com
socket 合法域名:
wss://wlnimsc0.netease.im
wss://wlnimsc1.netease.im
uploadFile 合法域名:
https://nim.nos.netease.com
https://nos.netease.com
downloadFile 合法域名
https://nim.nos.netease.com
https://nos.netease.com

小程序聊天室地址请走工单流程/或联系技术支持

日志分析

SDK 提供初始化参数，辅助用户对日志信息做收集，并进一步分析。如在混合

应用中，用户甚至可以直接传入 js-bridge 的方法收集日志。
例子为移动端应用将日志上报给服务器做分析

```
// 客户端代码
// 日志统计插件
function LoggerPlugin(options) {
  var logLevelMap = {
    debug: 0,
    log: 1,
    info: 2,
    warn: 3,
    error: 4
  };
  var self = this;
  var postUrl = options.url || null;
  self.level = logLevelMap[options.level] || 0;
  self.logCache = [];
  self.logNum = 1;
  self.timeInterval = 5000;
  window.onerror = function(
    errorMessage,
    scriptURI,
    lineNumber,
    columnNumber,
    errorObj
  ) {
    self.error.call(self, errorObj);
  };
  setInterval(function() {
    if (self.logCache.length > 0 && postUrl) {
      self.postLogs(postUrl, self.logCache);
    }
  }, self.timeInterval);
}

LoggerPlugin.prototype.debug = function() {
  if (this.level > 0) {
    return;
  }
  console.debug.apply(this, arguments);
  this.cacheLogs.apply(this, ['[degbug]'].concat(arguments));
};

LoggerPlugin.prototype.log = function() {
  if (this.level > 1) {
    return;
  }
  console.log.apply(this, arguments);
  this.cacheLogs.apply(this, ['[log]'].concat(arguments));
};

LoggerPlugin.prototype.info = function() {
```

```
  if (this.level > 2) {
    return;
  }
  console.info.apply(this, arguments);
  this.cacheLogs.apply(this, ['[info]'].concat(arguments));
};

LoggerPlugin.prototype.warn = function() {
  if (this.level > 3) {
    return;
  }
  console.warn.apply(this, arguments);
  this.cacheLogs.apply(this, ['[warn]'].concat(arguments));
};

LoggerPlugin.prototype.error = function() {
  if (this.level > 4) {
    return;
  }
  console.error.apply(this, arguments);
  this.cacheLogs.apply(this, ['[error]'].concat(arguments));
};

LoggerPlugin.prototype.cacheLogs = function(logLevel, args) {
  var currentCache = [];
  for (var i = 0; i < args.length; i++) {
    var arg = args[i];
    if (typeof arg === 'object') {
      currentCache.push(JSON.stringify(arg));
    } else {
      currentCache.push(arg);
    }
  }
  var logStr = this.logNum++ + ' ' + logLevel + ' ' + currentCache.
join('; ');
  this.logCache.push(logStr.replace('%c', ''));
};

LoggerPlugin.prototype.postLogs = function(url, data) {
  var self = this;
  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function() {
    if (xhr.readyState === 4) {
      if (xhr.status === 200) {
        console.info('LoggerPlugin::日志上报完成');
        self.logCache = [];
        self.timeInterval = 5000;
      } else {
        self.timeInterval += 5000;
      }
    }
  };
  xhr.open('POST', url);
  xhr.setRequestHeader('Content-Type', 'plain/text;charset=utf-8');
```

```
  xhr.timeout = 360;
  xhr.send(data.join('\n'));
};
```

LoggerPlugin 代码，sdk 也已集成，用户可直接使用`NIM.LoggerPlugin`做调用，文档中是希望用户可以参考代码做自己的一些高级开发

```
// 客户端代码
var nim = NIM.getInstance({
  debug: true,
  logFunc: new LoggerPlugin({
    url: '/getlogger',
    level: 'info'
  })
});
```

这里使用 `nodejs` + `express` 作为服务器，写了相关的服务器代码，仅供参考

```
// 服务端代码
const express = require('express');
const fs = require('fs');
const app = express();

app.post('/getlogger', (req, res) => {
  req.setEncoding('utf8');
  req.rawBody = '';
  req.on('data', chunk => {
    req.rawBody += chunk;
  });
  req.on('end', () => {
    let body = req.rawBody;
    fs.appendFile('sdklog.log', body, () => {
      res.end();
    });
  });
});
```

完整的初始化代码

请查阅其它章节的初始化参数

初始化 SDK

连接初始化参数

多端登录初始化参数

消息初始化参数
群组初始化参数
用户资料初始化参数
好友关系初始化参数
用户关系初始化参数
会话初始化参数
系统通知初始化参数
同步完成
完整的初始化代码

```
    var data = {};
    var nim = NIM.getInstance({
        // 初始化SDK
        // debug: true
        appKey: 'appKey',
        account: 'account',
        token: 'token',
        onconnect: onConnect,
        onerror: onError,
        onwillreconnect: onWillReconnect,
        ondisconnect: onDisconnect,
        // 多端登录
        onloginportschange: onLoginPortsChange,
        // 用户关系
        onblacklist: onBlacklist,
        onsyncmarkinblacklist: onMarkInBlacklist,
        onmutelist: onMutelist,
        onsyncmarkinmutelist: onMarkInMutelist,
        // 好友关系
        onfriends: onFriends,
        onsyncfriendaction: onSyncFriendAction,
        // 用户名片
        onmyinfo: onMyInfo,
        onupdatemyinfo: onUpdateMyInfo,
        onusers: onUsers,
        onupdateuser: onUpdateUser,
        // 机器人列表的回调
        onrobots: onRobots,
        // 群组
        onteams: onTeams,
        onsynccreateteam: onCreateTeam,
        onUpdateTeam: onUpdateTeam,
        onteammembers: onTeamMembers,
        // onsyncteammembersdone: onSyncTeamMembersDone,
        onupdateteammember: onUpdateTeamMember,
        // 群消息业务已读通知
        onTeamMsgReceipt: onTeamMsgReceipt,
        // 会话
```

```
        onsessions: onSessions,
        onupdatesession: onUpdateSession,
        // 消息
        onroamingmsgs: onRoamingMsgs,
        onofflinemsgs: onOfflineMsgs,
        onmsg: onMsg,
        // 系统通知
        onofflinesysmsgs: onOfflineSysMsgs,
        onsysmsg: onSysMsg,
        onupdatesysmsg: onUpdateSysMsg,
        onsysmsgunread: onSysMsgUnread,
        onupdatesysmsgunread: onUpdateSysMsgUnread,
        onofflinecustomsysmsgs: onOfflineCustomSysMsgs,
        oncustomsysmsg: onCustomSysMsg,
        // 收到广播消息
        onbroadcastmsg: onBroadcastMsg,
        onbroadcastmsgs: onBroadcastMsgs,
        // 同步完成
        onsyncdone: onSyncDone
    });

    function onConnect() {
        console.log('连接成功');
    }
    function onWillReconnect(obj) {
        // 此时说明 `SDK` 已经断开连接，请开发者在界面上提示用户连接已断开，而且正在重新建立连接
        console.log('即将重连');
        console.log(obj.retryCount);
        console.log(obj.duration);
    }
    function onDisconnect(error) {
        // 此时说明 `SDK` 处于断开状态，开发者此时应该根据错误码提示相应的错误信息，并且跳转到登录页面
        console.log('丢失连接');
        console.log(error);
        if (error) {
            switch (error.code) {
            // 账号或者密码错误，请跳转到登录页面并提示错误
            case 302:
                break;
            // 被踢，请提示错误后跳转到登录页面
            case 'kicked':
                break;
            default:
                break;
            }
        }
    }
    function onError(error) {
        console.log(error);
    }

    function onLoginPortsChange(loginPorts) {
```

```
        function onLoginPortsChange(loginPorts) {
            console.log('当前登录帐号在其它端的状态发生改变了', loginPorts);
        }

        function onBlacklist(blacklist) {
            console.log('收到黑名单', blacklist);
            data.blacklist = nim.mergeRelations(data.blacklist, blacklist);
            data.blacklist = nim.cutRelations(data.blacklist, blacklist.invalid);
            refreshBlacklistUI();
        }
        function onMarkInBlacklist(obj) {
            console.log(obj);
            console.log(obj.account + '被你在其它端' + (obj.isAdd ? '加入' : '移除') + '黑名单');
            if (obj.isAdd) {
                addToBlacklist(obj);
            } else {
                removeFromBlacklist(obj);
            }
        }
        function addToBlacklist(obj) {
            data.blacklist = nim.mergeRelations(data.blacklist, obj.record);
            refreshBlacklistUI();
        }
        function removeFromBlacklist(obj) {
            data.blacklist = nim.cutRelations(data.blacklist, obj.record);
            refreshBlacklistUI();
        }
        function refreshBlacklistUI() {
            // 刷新界面
        }
        function onMutelist(mutelist) {
            console.log('收到静音列表', mutelist);
            data.mutelist = nim.mergeRelations(data.mutelist, mutelist);
            data.mutelist = nim.cutRelations(data.mutelist, mutelist.invalid);
            refreshMutelistUI();
        }
        function onMarkInMutelist(obj) {
            console.log(obj);
            console.log(obj.account + '被你' + (obj.isAdd ? '加入' : '移除') + '静音列表');
            if (obj.isAdd) {
                addToMutelist(obj);
            } else {
                removeFromMutelist(obj);
            }
        }
        function addToMutelist(obj) {
            data.mutelist = nim.mergeRelations(data.mutelist, obj.record
```

```
        data.mutelist = nim.mergeRelations(data.mutelist, obj.record);
        refreshMutelistUI();
    }
    function removeFromMutelist(obj) {
        data.mutelist = nim.cutRelations(data.mutelist, obj.record);
        refreshMutelistUI();
    }
    function refreshMutelistUI() {
        // 刷新界面
    }

    function onFriends(friends) {
        console.log('收到好友列表', friends);
        data.friends = nim.mergeFriends(data.friends, friends);
        data.friends = nim.cutFriends(data.friends, friends.invalid);
        refreshFriendsUI();
    }
    function onSyncFriendAction(obj) {
        console.log(obj);
        switch (obj.type) {
        case 'addFriend':
            console.log('你在其它端直接加了一个好友' + obj.account + ', 附言' + obj.ps);
            onAddFriend(obj.friend);
            break;
        case 'applyFriend':
            console.log('你在其它端申请加了一个好友' + obj.account + ', 附言' + obj.ps);
            break;
        case 'passFriendApply':
            console.log('你在其它端通过了一个好友申请' + obj.account + ', 附言' + obj.ps);
            onAddFriend(obj.friend);
            break;
        case 'rejectFriendApply':
            console.log('你在其它端拒绝了一个好友申请' + obj.account + ', 附言' + obj.ps);
            break;
        case 'deleteFriend':
            console.log('你在其它端删了一个好友' + obj.account);
            onDeleteFriend(obj.account);
            break;
        case 'updateFriend':
            console.log('你在其它端更新了一个好友', obj.friend);
            onUpdateFriend(obj.friend);
            break;
        }
    }
    function onAddFriend(friend) {
        data.friends = nim.mergeFriends(data.friends, friend);
        refreshFriendsUI();
```

```
    }
    function onDeleteFriend(account) {
        data.friends = nim.cutFriendsByAccounts(data.friends, accou
nt);
        refreshFriendsUI();
    }
    function onUpdateFriend(friend) {
        data.friends = nim.mergeFriends(data.friends, friend);
        refreshFriendsUI();
    }
    function refreshFriendsUI() {
        // 刷新界面
    }

    function onMyInfo(user) {
        console.log('收到我的名片', user);
        data.myInfo = user;
        updateMyInfoUI();
    }
    function onUpdateMyInfo(user) {
        console.log('我的名片更新了', user);
        data.myInfo = NIM.util.merge(data.myInfo, user);
        updateMyInfoUI();
    }
    function updateMyInfoUI() {
        // 刷新界面
    }
    function onUsers(users) {
        console.log('收到用户名片列表', users);
        data.users = nim.mergeUsers(data.users, users);
    }
    function onUpdateUser(user) {
        console.log('用户名片更新了', user);
        data.users = nim.mergeUsers(data.users, user);
    }
    function onRobots (robots) {
        // 客户私有化方案不支持
        console.log('收到机器人列表', robots);
        data.robots = robots;
    }
    function onTeams(teams) {
        console.log('群列表', teams);
        data.teams = nim.mergeTeams(data.teams, teams);
        onInvalidTeams(teams.invalid);
    }
    function onInvalidTeams(teams) {
        data.teams = nim.cutTeams(data.teams, teams);
        data.invalidTeams = nim.mergeTeams(data.invalidTeams, teams
);
        refreshTeamsUI();
    }
    function onCreateTeam(team) {
        console.log('你创建了一个群', team);
        data.teams = nim.mergeTeams(data.teams, team);
```

```
        refreshTeamsUI();
        onTeamMembers({
            teamId: team.teamId,
            members: owner
        });
    }
    function refreshTeamsUI() {
        // 刷新界面
    }
    function onTeamMembers(teamId, members) {
        console.log('群id', teamId, '群成员', members);
        var teamId = obj.teamId;
        var members = obj.members;
        data.teamMembers = data.teamMembers || {};
        data.teamMembers[teamId] = nim.mergeTeamMembers(data.teamMe
mbers[teamId], members);
        data.teamMembers[teamId] = nim.cutTeamMembers(data.teamMemb
ers[teamId], members.invalid);
        refreshTeamMembersUI();
    }
    // function onSyncTeamMembersDone() {
    //     console.log('同步群列表完成');
    // }
    function onUpdateTeam (team) {
        console.log('群状态更新', team)
    }
    function onUpdateTeamMember(teamMember) {
        console.log('群成员信息更新了', teamMember);
        onTeamMembers({
            teamId: teamMember.teamId,
            members: teamMember
        });
    }
    function refreshTeamMembersUI() {
        // 刷新界面
    }
    function onTeamMsgReceipt (teamMsgReceipts) {
        console.log('群消息已读通知', teamMsgReceipts)
    }

    function onSessions(sessions) {
        console.log('收到会话列表', sessions);
        data.sessions = nim.mergeSessions(data.sessions, sessions);
        updateSessionsUI();
    }
    function onUpdateSession(session) {
        console.log('会话更新了', session);
        data.sessions = nim.mergeSessions(data.sessions, session);
        updateSessionsUI();
    }
    function updateSessionsUI() {
        // 刷新界面
    }
```

```
    function onRoamingMsgs(obj) {
        console.log('漫游消息', obj);
        pushMsg(obj.msgs);
    }
    function onOfflineMsgs(obj) {
        console.log('离线消息', obj);
        pushMsg(obj.msgs);
    }
    function onMsg(msg) {
        console.log('收到消息', msg.scene, msg.type, msg);
        pushMsg(msg);
    }
    function onBroadcastMsg(msg) {
        console.log('收到广播消息', msg);
    }
    function onBroadcastMsgs(msgs) {
        console.log('收到广播消息列表', msgs);
    }
    function pushMsg(msgs) {
        if (!Array.isArray(msgs)) { msgs = [msgs]; }
        var sessionId = msgs[0].sessionId;
        data.msgs = data.msgs || {};
        data.msgs[sessionId] = nim.mergeMsgs(data.msgs[sessionId],
msgs);
    }

    function onOfflineSysMsgs(sysMsgs) {
        console.log('收到离线系统通知', sysMsgs);
        pushSysMsgs(sysMsgs);
    }
    function onSysMsg(sysMsg) {
        console.log('收到系统通知', sysMsg)
        pushSysMsgs(sysMsg);
    }
    function onUpdateSysMsg(sysMsg) {
        pushSysMsgs(sysMsg);
    }
    function pushSysMsgs(sysMsgs) {
        data.sysMsgs = nim.mergeSysMsgs(data.sysMsgs, sysMsgs);
        refreshSysMsgsUI();
    }
    function onSysMsgUnread(obj) {
        console.log('收到系统通知未读数', obj);
        data.sysMsgUnread = obj;
        refreshSysMsgsUI();
    }
    function onUpdateSysMsgUnread(obj) {
        console.log('系统通知未读数更新了', obj);
        data.sysMsgUnread = obj;
        refreshSysMsgsUI();
    }
    function refreshSysMsgsUI() {
        // 刷新界面
    }
```

```
    function onOfflineCustomSysMsgs(sysMsgs) {
        console.log('收到离线自定义系统通知', sysMsgs);
    }
    function onCustomSysMsg(sysMsg) {
        console.log('收到自定义系统通知', sysMsg);
    }

    function onSyncDone() {
        console.log('同步完成');
    }
```

清除单例实例

web sdk 连接实例均为单例模式，但可以调用相应接口清除内存中记录的实例，即断开连接，清除内存消息记录及时间戳，方便开发者做到干净重连。

```
  var nim = NIM.getInstance({...})
  // 清除实例
  nim.destroy({
    done: function (err) {
      console.log('实例已被完全清除')
    }
  })
````

** 注意: **
- 手动调用 `nim.disconnect()` 方法或 `nim.destroy()` 方法后，会直接触发实例的 `ondisconnect` 方法，但此时websocket并没有真正意义上被销毁，只有在 `done` 回调的时候，才能保证socket是 `onclose` 状态的。
- 这点在微信小程序里尤其重要，因为小程序只有两条websocket可用，如果在前一条socket没有close的时候，会占用socket资源，导致反复重连或者悄悄被踢的错误。
登录与登出

登录登出逻辑

- SDK进行连接的过程，同时也会执行相应的登录逻辑，若登录成功，则会在初始化设置的`onconnect`回调中得到相应的登录信息。若登录失败，也将会在`ondisconnect`回调中获得相应的失败信息。
- 连接状态初始化可参见: [初始化SDK](http://dev.netease.im/docs/product/IM%E5%8D%B3%E6%97%B6%E9%80%9A%E8%AE%AF/SDK%E5%BC%80%E5%8F%91%E9%9B%86%E6%88%90/Web%E5%BC%80%E5%8F%91%E9%9B%86%E6%88%90/%E5%88%9D%E5%A7%8B%E5%8C%96)

登出 IM

- [初始化SDK](http://dev.netease.im/docs/product/IM%E5%8D%B3%E6%97%B6%E9%80%9A%E8%AE%AF/SDK%E5%BC%80%E5%8F%91%E9%9B%86%E6%88%90/Web%E5%BC%80%E5%8F%91%E9%9B%86%E6%88%90/%E5%88%9D%E5%A7%8B%E5%8C%96)之后，

```
SDK 会自动登录
- 在收到`onconnect`回调后可以调用`nim.disconnect();`来登出 SDK
- 登出 SDK 后可以调用`nim.connect();`来重新登入 SDK

## <span id="切换IM">切换 IM</soan>

如果需要切换 IM，则需要断开连接后再使用新账号重新连接。操作步骤如下
- 调用[登出IM](http://dev.netease.im/docs/product/IM%E5%8D%B3%E6%97%B6%E9%80%9A%E8%AE%AF/SDK%E5%BC%80%E5%8F%91%E9%9B%86%E6%88%90/Web%E5%BC%80%E5%8F%91%E9%9B%86%E6%88%90/%E7%99%BB%E5%BD%95%E7%99%BB%E5%87%BA#登出IM)来登出IM
- 调用[初始化SDK](http://dev.netease.im/docs/product/IM%E5%8D%B3%E6%97%B6%E9%80%9A%E8%AE%AF/SDK%E5%BC%80%E5%8F%91%E9%9B%86%E6%88%90/Web%E5%BC%80%E5%8F%91%E9%9B%86%E6%88%90/%E5%88%9D%E5%A7%8B%E5%8C%96)来初始化新的 IM

**若开发者有需求做无缝切换**，则可对不同账号的IM连接实例做保持，由上层管理不同账号间的处理逻辑

```javascript
    var nim1 = NIM.getInstance({
        account: 'nim1',
        // ...
    })
    var nim2 = NIM.getInstance({
        account: 'nim2',
        // ...
    })
    // nim1 nim2 ... nimN同时都会接收消息，都可发送消息及其它IM功能，请管理好相应实例方法的独立性，以及浏览器性能
```

更新 IM 配置

SDK 设计为单例模式, 如果需要更新当前 IM 的配置(部分配置不可更新，如用户账号), 那么可以通过两种方式完成：

再次调用NIM.getInstance方法，仅需传入需要更改配置的部分(不建议，因为使用不当，会出现实例引用逻辑混乱的问题)
对原有实例调用setOptions方法, 参数列表和格式跟NIM.getInstance保持一致, 以更新 token 为例
```

```
// 断开 IM
nim.disconnect();
// 更新 token
nim.setOptions({
    token: 'newToken'
});
// 重新连接
nim.connect();
```

多端登录

云信支持多端同时登录, 即用户可以同时在移动端和网页端登录同一账号

初始化参数

这里的参数并不是所有的初始化参数, 请查阅初始化SDK, 以及其它章节的初始化参数

连接初始化参数
多端登录初始化参数
消息初始化参数
群组初始化参数
用户资料初始化参数
好友关系初始化参数
用户关系初始化参数
会话初始化参数
系统通知初始化参数
同步完成
完整的初始化代码

示例代码

```
var nim = NIM.getInstance({
    onloginportschange: onLoginPortsChange
});
function onLoginPortsChange(loginPorts) {
    console.log('当前登录帐号在其它端的状态发生改变了', loginPorts);
}
```

参数解释

`onloginportschange`: 多端登录状态变化的回调, 会收到登录端列表, 以下情况会收到此回调
登录时其它端在线
登录后其它端上线或者下线

登录端

登录端代表登录在某个设备上的相关信息, 有如下字段
- `type`: 登录的设备类型
- `os`: 登录设备的操作系统
- `mac`: 登录设备的 mac 地址
- `deviceId`: 登录设备ID, uuid
- `account`: 登录的帐号
- `connectionId`: 登录设备分配的连接号
- `ip`: 登录的服务器 IP
- `time`: 登录时间
- `online`: 是否在线

设备类型

目前云信支持的登录端有以下几种类型
- `'Android'` (安卓)
- `'iOS'` (苹果)
- `'PC'` (桌面，这里默认指代windows)
- `'Web'` (浏览器)
- `'Mac'` (桌面)

踢其它端

示例代码

```
nim.kick({
    deviceIds: ['deviceId1'],
    done: onKick
});
function onKick(error, obj) {
    console.log('踢其它端' + (!error?'成功':'失败'));
    console.log(error);
    console.log(obj);
}
```

参数解释

其它登录端的设备号可以在`onloginportschange`回调里获取, 参考登录端对象

消息收发

消息说明

点对点或群组消息的收发，可分为消息、离线消息、漫游消息三类。
离线消息、漫游消息只能收不能发送，在初始化时onofflinemsgs/onroamingmsgs的回调中获取
消息可收可发，收消息在初始化时onmsg的回调中接收
离线消息：他人发给自己的消息，且自己的账号在任何客户端都未读过，则算离线消息；离线消息只要其它任何一端(包括自己)已读，则不会再收到对应消息。
漫游消息：他人发给自己的消息，在本客户端未读过，但在其他客户端如iOS/android/pc...读过，则算漫游消息；漫游消息只有本端已读，才不会再下推。

初始化参数

这里的参数并不是所有的初始化参数, 请查阅初始化SDK, 以及其它章节的初始化参数

连接初始化参数

多端登录初始化参数

消息初始化参数

群组初始化参数

用户资料初始化参数

好友关系初始化参数

用户关系初始化参数

会话初始化参数

系统通知初始化参数

同步完成

完整的初始化代码

示例代码

```
var nim = NIM.getInstance({
    onroamingmsgs: onRoamingMsgs,
    onofflinemsgs: onOfflineMsgs,
    onmsg: onMsg
});
function onRoamingMsgs(obj) {
    console.log('收到漫游消息', obj);
    pushMsg(obj.msgs);
}
function onOfflineMsgs(obj) {
    console.log('收到离线消息', obj);
    pushMsg(obj.msgs);
}
function onMsg(msg) {
    console.log('收到消息', msg.scene, msg.type, msg);
    pushMsg(msg);
    switch (msg.type) {
    case 'custom':
        onCustomMsg(msg);
        break;
    case 'notification':
        // 处理群通知消息
        onTeamNotificationMsg(msg);
        break;
    // 其它case
    default:
        break;
    }
}
function pushMsg(msgs) {
    if (!Array.isArray(msgs)) { msgs = [msgs]; }
    var sessionId = msg[0].scene + '-' + msgs[0].account;
    data.msgs = data.msgs || {};
    data.msgs[sessionId] = nim.mergeMsgs(data.msgs[sessionId], msgs
);
}
function onCustomMsg(msg) {
    // 处理自定义消息
}
```

参数解释

`shouldIgnoreNotification`, 该参数类型为函数(function)，表示是否要忽略某条通知类消息。该方法会将接收到的通知类消息对象，按照用户上层定义的逻辑进行过滤, 如果该方法返回 true，那么 SDK 将忽略此条通知类消息

`onroamingmsgs`, 同步漫游消息对象的回调, 每个会话对象对应一个回调, 会传入消息数组

`onofflinemsgs`, 同步离线消息对象的回调, 每个会话对象对应一个回调, 会传入

消息数组
在支持数据库时并且启用了多 tab 同时登录, 那么如果多个 tab 页同时断线重连之后, 只会有一个 tab 页负责存储漫游消息和离线消息, 即只会有一个 tab 页会收到 `onroamingmsgs` 和 `onofflinemsgs` 回调, 其它 tab 页在同步完成之后, 需要调用获取本地历史记录来从本地缓存中拉取消息记录
`onmsg`, 收到消息对象的回调
当前登录帐号在其它端发送消息之后也会收到此回调, 注意此时消息对象的`from`字段就是当前登录的帐号
可以调用nim.mergeMsgs来合并数据

消息对象

消息对象有以下字段
- `scene`: 消息场景
- `from`: 消息发送方, 帐号或群id
- `fromNick`: 消息发送方的昵称
- `fromClientType`: 发送方的设备类型
- `fromDeviceId`: 发送端设备id
- `to`: 消息接收方, 帐号或群id
- `time`: 时间戳
- `type`: 消息类型
- `sessionId`: 消息所属的会话对象的ID
- `target`: 聊天对象, 账号或者群id
- `flow`: 消息的流向
- 'in'表示此消息是收到的消息
- 'out'表示此消息是发出的消息
- `status`: 消息发送状态
- `'sending'` 发送中
- `'success'` 发送成功
- `'fail'` 发送失败
- `text`: 文本消息的文本内容, 请参考发送文本消息
- `file`: 文件消息的文件对象, 具体字段请参考图片对象、音频对象、视频对

象、文件对象, 请参考发送文件消息
- `geo`: 地理位置消息的地理位置对象, 请参考发送地理位置消息
- `tip`: 提醒消息的内容, 请参考发送提醒消息
- `content`: 自定义消息或机器人回复消息的消息内容, 开发者可以自行扩展, 建议封装成JSON格式字符串, 请参考发送自定义消息
- `attach`: 群通知消息的附加信息, 参考群通知消息来查看不同类型的群通知消息对应的附加信息
- `idClient`: SDK生成的消息id, 在发送消息之后会返回给开发者, 开发者可以在发送消息的回调里面根据这个ID来判断相应消息的发送状态, 到底是发送成功了还是发送失败了, 然后根据此状态来更新页面的UI。如果发送失败, 那么可以重发消息
- `isMuted`: 该消息在接收方是否应该被静音
- `resend`: 是否是重发的消息
- `custom`: 扩展字段
- 推荐使用`JSON`格式构建, 非`JSON`格式的话, Web端会正常接收, 但是会被其它端丢弃
- `pushContent`: 自定义推送文案
- `pushPayload`: 自定义的推送属性
- 推荐使用`JSON`格式构建, 非`JSON`格式的话, Web端会正常接收, 但是会被其它端丢弃
- `needPushNick`: 是否需要推送昵称
- `apns`: 特殊推送选项, 只在群会话中使用
- `apns.accounts`: 需要特殊推送的账号列表, 此字段不存在的话表示推送给当前会话内的所有用户
- `apns.content`: 需要特殊推送的文案
- `apns.forcePush`: 是否强制推送, true 表示即使推送列表中的用户屏蔽了当前会话（如静音）, 仍能够推送当前这条内容给相应用户
- `localCustom`: 本地自定义扩展字段
- 在支持数据库时可以调用更新本地消息来更新此字段, 此字段只会被更新到本地数据库, 不会被更新到服务器上
- `needMsgReceipt`: 是否需要业务已读（包含该字段即表示需要），只有设置了业务已读，才可以调用`getTeamMsgReads`, `getTeamMsgReadAccounts`等相关方法

- `isRoamingable`: 是否支持漫游
- `isSyncable`: 是否支持发送者多端同步
- `cc`: 是否支持抄送
- `isPushable`: 是否需要推送
- `isOfflinable`: 是否要存离线
- `isUnreadable`: 是否计入消息未读数
- `isLocal`: 是否是本地消息, 请查阅发送本地消息

消息场景

消息对象有一个字段`scene`来标明消息所属的场景, 具体场景如下
- `'p2p'` (点对点消息)
- `'team'` (群消息)

消息类型

消息对象有一个字段`type`来标明消息的类型, 具体类型如下
- `'text'` (文本消息)
- `'image'` (图片消息)
- `'audio'` (音频消息)
- `'video'` (视频消息)
- `'file'` (文件消息)
- `'geo'` (地理位置消息)
- `'custom'` (自定义消息)
- `'tip'` (提醒消息)
- 提醒消息用于会话内的状态提醒，如进入会话时出现的欢迎消息，或者会话命中敏感词后的提示消息等等.
- `'notification'` (群通知消息)
- 某些群操作后所有群成员会收到一条相应的群通知消息, 详细介绍请参考群通知消息
- 此类消息不会计入未读数

- 请参考消息对象、消息类型

不同类型的消息收发可参考：
- 发送消息相关文档
- 发送文本消息
- 预览文件
- 发送文件消息
- 发送地理位置消息
- 发送提醒消息
- 发送自定义消息

文本对象

当发送文本消息或收到文本消息时, 消息对象的`text`字段代表文本消息内容

图片对象

当发送图片消息或收到图片消息时, 消息对象的`file`字段代表图片对象, 包含以下属性:
- `name`: 名字
- `size`: 大小, 单位byte
- `md5`: md5
- `url`: url
- `ext`: 扩展名
- `w`: 宽, 单位px
- `h`: 高, 单位px

音频对象

当发送音频消息或收到音频消息时, 消息对象的`file`字段代表音频对象, 包含以下属性:
- `name`: 名字
- `size`: 大小, 单位byte

- `md5`: md5
- `url`: url
- `ext`: 扩展名
- `dur`: 长度, 单位ms

视频对象

当发送视频消息或收到视频消息时, 消息对象的`file`字段代表视频对象, 包含以下属性:
- `name`: 名字
- `size`: 大小, 单位byte
- `md5`: md5
- `url`: url
- `ext`: 扩展名
- `dur`: 长度, 单位ms
- `w`: 宽, 分辨率, 单位px
- `h`: 高, 分辨率, 单位px

视频对象取封面(首帧图片):
- 获取到的视频对象后加`vframe`即可，例如：原视频地址为`http://img-sample.nos-eastchina1.126.net/sample.wmv`，则封面(首帧)图片地址为
`http://img-sample.nos-eastchina1.126.net/sample.wmv?vframe`

文件对象

当发送文件消息或收到文件消息时, 消息对象的`file`字段代表文件对象, 包含以下属性:
- `name`: 名字
- `size`: 大小, 单位byte
- `md5`: md5
- `url`: url
- `ext`: 扩展名

地理位置对象

当发送地理位置消息或收到地理位置消息时, 消息对象的`geo`字段代表地理位置对象, 包含以下属性:
- `lng`: 经度
- `lat`: 纬度
- `title`: 地址描述

群通知消息

群通知消息是消息类型的一种
某些群操作后所有群成员会收到一条相应的群通知消息
群通知消息对应的消息对象有一个字段`attach`包含了额外的信息, `attach`有一个字段`type`来标识群通知消息的类型

`'updateTeam'` (更新群)
更新群后, 所有群成员会收到一条类型为`'updateTeam'`的群通知消息。此类群通知消息的`from`字段的值为更新群的人的帐号, `to`字段的值为对应的群ID, `attach`有一个字段`team`的值为被更新的群信息

`'addTeamMembers'` (拉人入群)
普通群, 拉人入群后, 所有群成员会收到一条类型为`'addTeamMembers'`的群通知消息。此类群通知消息的`from`字段的值为拉人的人的帐号, `to`字段的值为对应的群ID, `attach`有一个字段`team`的值为对应的群对象, `attach`有一个字段`accounts`的值为被拉的人的帐号列表, `attach`有一个字段`members`的值为被拉的群成员列表。

`'removeTeamMembers'` (踢人出群)
踢人出群后, 所有群成员会收到一条类型为`'removeTeamMembers'`的群通知消息。此类群通知消息的`from`字段的值为踢人的人的帐号, `to`字段的值为对应的群ID, `attach`有一个字段`team`的值为对应的群对象, `attach`有一个字段`accounts`的值为被踢的人的帐号列表。

`'acceptTeamInvite'` (接受入群邀请)
高级群的群主和管理员在邀请成员加入群（通过操作创建群或拉人入群）之后, 被邀请的人会收到一条类型为`'teamInvite'`的系统通知, 此类系统通知的`from`

字段的值为邀请方的帐号, to字段的值为对应的群ID, 此类系统通知的attach有一个字段team的值为被邀请进入的群, 被邀请的人可以选择接受邀请或者拒绝邀请。

如果接受邀请, 那么该群的所有群成员会收到一条类型为'acceptTeamInvite'的群通知消息, 此类群通知消息的from字段的值为接受入群邀请的人的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群对象, attach有一个字段members的值为接收入群邀请的群成员列表。

如果拒绝邀请, 那么邀请你的人会收到一条类型为'rejectTeamInvite'的系统通知, 此类系统通知的from字段的值为拒绝入群邀请的用户的帐号, to字段的值为对应的群ID。

'passTeamApply' (通过入群申请)

用户可以申请加入高级群, 目标群的群主和管理员会收到一条类型为'applyTeam'的系统通知, 此类系统通知的from字段的值为申请方的帐号, to字段的值为对应的群ID, 高级群的群主和管理员在收到入群申请后, 可以选择通过或者拒绝入群申请。

如果通过申请, 那么该群的所有群成员会收到一条类型为'passTeamApply'的群通知消息, 此类群通知消息的from字段的值为通过入群申请的人的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群对象, attach有一个字段account的值为申请方的帐号, attach有一个字段members的值为被通过申请的群成员列表。

如果拒绝申请, 那么申请人会收到一条类型为'rejectTeamApply'的系统通知, 此类系统通知的from字段的值为拒绝方的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群。

'addTeamManagers' (添加群管理员)

添加群管理员后, 所有群成员会收到一条类型为'addTeamManagers'的群通知消息。此类群通知消息的from字段的值为添加群管理员的人的帐号, to字段的值为对应的群ID, attach有一个字段accounts的值为被加为管理员的帐号列表, attach有一个字段members的值为被加为管理员的群成员列表

'removeTeamManagers' (移除群管理员)

移除群管理员后, 所有群成员会收到一条类型为'removeTeamManagers'的群通知消息。此类群通知消息的from字段的值为移除群管理员的人的帐号, to字段的值为对应的群ID, attach有一个字段accounts的值为被移除的管理员的帐号列表, attach有一个字段members的值为被移除管理员的群成员列表

'leaveTeam' (主动退群)
主动退群后, 所有群成员会收到一条类型为'leaveTeam'的群通知消息。此类群通知消息的from字段的值为退群的人的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群对象。
'dismissTeam' (解散群)
解散群后, 所有群成员会收到一条类型为'dismissTeam'的群通知消息。此类群通知消息的from字段为解散群的人的帐号, to字段的值为被对应的群ID。
'transferTeam' (转让群)
转让群后, 所有群成员会收到一条类型为'transferTeam'的群通知消息。此类群通知消息的from字段的值为转让群的人的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群对象, attach有一个字段account的值为为新群主的帐号, attach有一个字段members的值为包含新旧群主的群成员列表。
'updateTeamMute' (更新群成员禁言状态)
更新群成员禁言状态后, 所有群成员会收到一条类型为'updateTeamMute'的群通知消息。此类群通知消息的from字段的值为操作方, to字段的值为对应的群ID, attach有一个字段team的值为对应的群对象, attach有一个字段account的值为被禁言的帐号, attach有一个字段members的值为被禁言的群成员列表。
如果attach有account或者accounts字段, 那么attach的字段users包含这些账号对应的用户名片
更新群昵称不会收到群通知消息, 所有其它在线的群成员会收到初始化SDK时传入的onupdateteammember回调, 请参考修改自己的群属性和修改别人的群昵称

处理群通知消息

这一章节涉及到了群和群成员的处理, 要跟以下章节一起看
群组初始化参数里面的 onteams, onsynccreateteam, onteammembers, onupdateteammember
消息初始化参数里面的 onmsg
群通知消息

示例代码

```
function onTeamNotificationMsg(msg) {
    // 处理群通知消息
```

```
    var type = msg.attach.type,
        from = msg.from,
        teamId = msg.to,
        timetag = msg.time,
        team = msg.attach.team,
        account = msg.attach.account,
        accounts = msg.attach.accounts,
        members = msg.attach.members;
    switch (type) {
    case 'updateTeam':
        team.updateTime = timetag;
        onTeams(team);
        break;
    case 'addTeamMembers':
        onAddTeamMembers(team, accounts, members);
        break;
    case 'removeTeamMembers':
        onRemoveTeamMembers(team, teamId, accounts);
        break;
    case 'acceptTeamInvite':
        onAddTeamMembers(team, [from], members);
        break;
    case 'passTeamApply':
        onAddTeamMembers(team, [account], members);
        break;
    case 'addTeamManagers':
        updateTeamManagers(teamId, members);
        break;
    case 'removeTeamManagers':
        updateTeamManagers(teamId, members);
        break;
    case 'leaveTeam':
        onRemoveTeamMembers(team, teamId, [from]);
        break;
    case 'dismissTeam':
        dismissTeam(teamId);
        break;
    case 'transferTeam':
        transferTeam(team, members);
        break;
    }
}
function onAddTeamMembers(team, accounts, members) {
    var teamId = team.teamId;
    /*
    如果是别人被拉进来了，那么拼接群成员列表
    如果是自己被拉进来了，那么同步一次群成员列表
    */
    if (accounts.indexOf(data.account) === -1) {
        onTeamMembers({
            teamId: teamId,
            members: members
        });
```

```
    } else {
        nim.getTeamMembers({
            teamId: teamId,
            sync: true,
            done: function(error, obj) {
                if (!error) {
                    onTeamMembers(obj);
                }
            }
        });
    }
    onTeams(team);
}
function onRemoveTeamMembers(team, teamId, accounts) {
    /*
    如果是别人被踢了，那么移除群成员
    如果是自己被踢了，那么离开该群
    */
    if (accounts.indexOf(data.account) === -1) {
        if (team) {
            onTeams(team);
        }
        data.teamMembers[teamId] = nim.cutTeamMembersByAccounts(dat
a.teamMembers[teamId], teamId, accounts);
        refreshTeamMembersUI();
    } else {
        leaveTeam(teamId);
    }
}
function updateTeamManagers(teamId, members) {
    onTeamMembers({
        teamId: teamId,
        members: members
    });
};
function leaveTeam(teamId) {
    onInvalidTeams({
        teamId: teamId
    });
    removeAllTeamMembers(teamId);
}
function dismissTeam(teamId) {
    onInvalidTeams({
        teamId: teamId
    });
    removeAllTeamMembers(teamId);
}
function removeAllTeamMembers(teamId) {
    delete data.teamMembers[teamId];
    refreshTeamMembersUI();
}
function transferTeam(team, members) {
    var teamId = team.teamId;
    onTeamMembers({
```

```
        teamId: teamId,
        members: members
    });
    onTeams(team);
}
```

参数解释

这里面用到了nim.cutTeamMembersByAccounts来合并群成员

发送消息

跟发消息相关的接口有
发送文本消息
预览文件
发送文件消息
发送地理位置消息
发送提醒消息
发送自定义消息
发送消息的配置选项
发送本地消息
重发消息
转发消息
消息撤回
先解释几个所有发送消息的接口都用到的参数
scene参数用来指定发送消息的场景
to参数用来指定消息的接收方, 发送点对点消息时填帐号, 发送群消息时填群ID
发送消息的接口会返回SDK生成的ID, 对应为字段idClient, 有一个例外是直接发送文件消息是在beforesend回调里获取这个值
在done回调中可以根据error对象和消息对象的idClient字段来确定对应的消息的发送状态。
如果error为空, 那么表明idClient对应的消息发送成功
error不为空, 表明idClient对应的消息发送失败, error包含详细的错误信息
以下代码皆以发送点对点消息（scene为'p2p'）为例, 如需发送群消息, 请将sc

ene的值替换为'team', 将to的值替换为群ID

发送文本消息

文本消息是消息收发的一种

```
var msg = nim.sendText({
    scene: 'p2p',
    to: 'account',
    text: 'hello',
    done: sendMsgDone
});
console.log('正在发送p2p text消息, id=' + msg.idClient);
pushMsg(msg);
function sendMsgDone(error, msg) {
    console.log(error);
    console.log(msg);
    console.log('发送' + msg.scene + ' ' + msg.type + '消息' + (!err
or?'成功':'失败') + ', id=' + msg.idClient);
    pushMsg(msg);
}
```

预览文件

开发者可以预览文件, 支持以下几种场景
通过参数fileInput传入文件选择 dom 节点或者节点 ID
通过参数blob传入 Blob 对象
通过参数dataURL传入包含 MIME type 和 base64 数据的 data URL, 此用法需要浏览器支持 window.Blob
通过参数wxFilePath， 将废弃， 传入微信小程序file对象的path(微信小程序专用，5.1.0+版本推荐使用filePath)
通过参数filePath 传入文件路径，5.1.0版本支持小程序，未来支持nodejs、react-native的时候(包括微信小程序)，上传文件统一使用该参数
SDK会将文件上传到文件服务器, 然后将拿到的文件对象在done回调中传给开发者, 文件对象有以下几种
图片对象
音频对象
视频对象
文件对象

开发者在拿到文件对象之后, 可以调用发送文件消息来发送文件消息。
文件大小限制为最大100M
高级浏览器会在上传前就检测文件大小
IE8/IE9 会在上传完成后检测文件大小

```
nim.previewFile({
    type: 'image',
    fileInput: fileInput,
    uploadprogress: function(obj) {
        console.log('文件总大小: ' + obj.total + 'bytes');
        console.log('已经上传的大小: ' + obj.loaded + 'bytes');
        console.log('上传进度: ' + obj.percentage);
        console.log('上传进度文本: ' + obj.percentageText);
    },
    done: function(error, file) {
        console.log('上传image' + (!error?'成功':'失败'));
        // show file to the user
        if (!error) {
            var msg = nim.sendFile({
                scene: 'p2p',
                to: 'account',
                file: file,
                done: sendMsgDone
            });
            console.log('正在发送p2p image消息, id=' + msg.idClient);
            pushMsg(msg);
        }
    }
});
```

发送文件消息

文件消息是消息收发的一种
开发者可以直接发送文件消息
支持以下几种场景
通过参数`fileInput`传入文件选择 dom 节点或者节点 ID
通过参数`blob`传入 Blob 对象
通过参数`dataURL`传入包含 MIME type 和 base64 数据的 data URL, 此用法需要浏览器支持 window.Blob
通过参数`wxFilePath`传入微信小程序file对象的path(微信小程序专用，5.1.0+版本推荐使用`filePath`)
通过参数`filePath` 传入文件路径，5.1.0版本支持小程序，未来将会支持nodejs

、react-native
SDK会先将文件上传到文件服务器, 然后把拿到的文件对象在`uploaddone`回调中传给用户, 然后将其拼装成文件消息发送出去。
开发者也可以先预览文件来获取文件对象, 然后调用此接口发送文件消息。
直接发送文件消息的话会在`beforesend`回调里面传入SDK生成的`idClient`, 如果先预览文件再发送, 那么此接口会直接返回`idClient`
参数`type`指定了要发送的文件类型, 包括图片、音频、视频和普通文件, 对应的值分别为`'image'`、`'audio'`、`'video'`和`'file'`, 不传默认为`'file'`。
图片、音频、视频和普通文件的区别在于具体的文件信息不一样, 具体字段请参考
图片对象
音频对象
视频对象
文件对象
文件大小限制为最大100M
高级浏览器会在上传前就检测文件大小
IE8和IE9会在上传完成后检测文件大小

```
nim.sendFile({
    scene: 'p2p',
    to: 'account',
    type: 'image',
    fileInput: fileInput,
    beginupload: function(upload) {
        // - 如果开发者传入 fileInput，在此回调之前不能修改 fileInput
        // - 在此回调之后可以取消图片上传，此回调会接收一个参数 `upload`，调用 `upload.abort();` 来取消文件上传
    },
    uploadprogress: function(obj) {
        console.log('文件总大小：' + obj.total + 'bytes');
        console.log('已经上传的大小：' + obj.loaded + 'bytes');
        console.log('上传进度：' + obj.percentage);
        console.log('上传进度文本：' + obj.percentageText);
    },
    uploaddone: function(error, file) {
        console.log(error);
        console.log(file);
        console.log('上传' + (!error?'成功':'失败'));
    },
    beforesend: function(msg) {
        console.log('正在发送p2p image消息，id=' + msg.idClient);
        pushMsg(msg);
    },
    done: sendMsgDone
});
```

发送地理位置消息

地理位置消息是消息收发的一种，geo参数请参考地理位置对象

```
var msg = nim.sendGeo({
    scene: 'p2p',
    to: 'account',
    geo: {
        lng: '116.3833',
        lat: '39.9167',
        title: 'Beijing'
    },
    done: sendMsgDone
});
console.log('正在发送p2p geo消息，id=' + msg.idClient);
pushMsg(msg);
```

发送提醒消息

提醒消息是消息收发的一种
提醒消息用于会话内的状态提醒，如进入会话时出现的欢迎消息，或者会话命中敏感词后的提示消息等等.

```
var msg = nim.sendTipMsg({
    scene: 'p2p',
    to: 'account',
    tip: 'tip content',
    done: sendMsgDone
});
console.log('正在发送p2p提醒消息, id=' + msg.idClient);
pushMsg(msg);
```

发送自定义消息

自定义消息是消息收发的一种
在网易云信开放的web-demo源码中，type-1为[石头剪刀布]，type-2为[阅后即焚]，type-3为[贴图表情]，type-4为[白板教学]
下面的代码用自定义消息实现了石头剪刀布游戏

```
var value = Math.ceil(Math.random()*3);
var content = {
    type: 1,
    data: {
        value: value
    }
};
var msg = nim.sendCustomMsg({
    scene: 'p2p',
    to: 'account',
    content: JSON.stringify(content),
    done: sendMsgDone
});
console.log('正在发送p2p自定义消息, id=' + msg.idClient);
pushMsg(msg);
```

发送消息的配置选项

上面的各个发送消息的接口都可以配置额外的选项, 来满足开发者对服务器的自定义需求。
custom: 扩展字段
推荐使用JSON格式构建, 非JSON格式的话, Web端会正常接收, 但是会被其它端

丢弃

pushContent: 自定义推送文案

pushPayload: 自定义的推送属性

推荐使用JSON格式构建, 非JSON格式的话, Web端会正常接收, 但是会被其它端丢弃

needPushNick: 是否需要推送昵称

apns: 特殊推送选项, 只在群会话中使用

apns.accounts: 需要特殊推送的账号列表, 不填表示推送给当前会话内的所有用户

apns.content: 需要特殊推送的文案, 不填的话默认为 pushContent

apns.forcePush 是否强制推送, 不填的话默认 true. true 表示即使推送列表中的用户屏蔽了当前会话（如静音）, 仍能够推送当前这条内容给相应用户

isRoamingable: 是否支持漫游

isSyncable: 是否支持发送者多端同步

cc: 是否支持抄送

isPushable: 是否需要推送

isOfflinable: 是否要存离线

isUnreadable: 是否计入消息未读数

needMsgReceipt: 是否需要业务已读（包含该字段即表示需要），只有设置了业务已读，才可以调用getTeamMsgReads,getTeamMsgReadAccounts等相关方法

下面给一个发送文本消息的例子, 发送其它消息的接口类似

```
var msg = nim.sendText({
    scene: 'p2p',
    to: 'account',
    text: 'hello',
    custom: '{}',
    done: sendMsgDone
});
```

发送本地消息

发送消息时可以指定参数isLocal为true, 那么SDK并不会发送此条消息, 而是直接调用回调表示发送成功, 并更新对应的会话

```
var value = Math.ceil(Math.random()*3);
var content = {
    type: 1,
    data: {
        value: value
    }
};
var msg = nim.sendCustomMsg({
    scene: 'p2p',
    to: 'account',
    content: JSON.stringify(content),
    isLocal: true,
    done: sendMsgDone
});
console.log('正在发送p2p自定义消息, id=' + msg.idClient);
pushMsg(msg);
```

重发消息

如果消息发送失败, 那么可以重发消息

```
nim.resendMsg({
  msg: someMsg,
  done: sendMsgDone
})
console.log('正在重发消息', someMsg)
```

转发消息

`msg`: 待转发的消息

`scene`: 新的场景

`to`: 新的接收方, 对方帐号或者群id

```
nim.forwardMsg({
  msg: someMsg,
  scene: 'p2p',
  to: 'account',
  done: sendMsgDone
})
console.log('正在转发消息', someMsg)
```

消息撤回

在会话时，允许用户撤回一定时间内发送过的消息,这个时长可以由云信管理后台进行配置。
如果需要在撤回后显示一条已撤回的提示（见 Demo 交互），开发者可以自行构造一条提醒消息并插入本地数据库。
撤回消息后, 消息接收方会收到一条类型为'deleteMsg'的系统通知, 此类系统通知的 msg 为被删除的消息的部分字段。如果是群消息, 那么群里的所有人都会收到这条系统通知. 如果同时在多个端登录了同一个账号, 那么其它端也会收到这条系统通知.
msg: 待撤回的消息

```
nim.deleteMsg({
  msg: someMsg,
  done: deleteMsgDone
})
console.log('正在撤回消息', someMsg)
function deleteMsgDone (error) {
  console.log('撤回消息' + (!error?'成功':'失败'), error);
}
```

标记消息为已收到

先解释一下消息发送和接收的流程, A 发消息给 B, 实际的流程是:
A 将消息发送给服务器, 如果 B 在线, 服务器会将消息推给 B; 如果 B 不在线, 服务器会在 B 上线的时候将此消息作为离线消息推给 B
B 在收到在线消息和离线消息之后, 需要告诉服务器收到了这些消息, 这样 B 下次登录时服务器就不会再次推这些消息
如果 B 没有告诉服务器收到了这些消息, 那么 B 下次登录时, 服务器会再次将这些消息推给 B
默认情况下, SDK 在收到消息（包括在线消息和离线消息）之后就将消息标记为已收到, 这样下次登录时就不会再收到这些消息, 一般情况下开发者不需要关心此接口
在支持数据库时, SDK 会将消息存储于数据库中, 如果开发者发现会话的未读数大于收到的离线消息数, 那么需要从本地拉取未读取的消息.
在不支持数据库时, 如果开发者想控制标记消息为已收到的时机, 那么可以设置初始化参数autoMarkRead为false, 这样SDK就不会自动标记消息为已收到, 此

时需要开发者在适当的时机调用此接口来标记消息为已收到, 否则下次登录后还是会收到未标记为已收到的消息.

示例代码

```
var nim = NIM.getInstance({
    autoMarkRead: false
});
nim.markMsgRead(someMsg);
// or
nim.markMsgRead([someMsg]);
```

已读回执

会话对象加了一个属性`msgReceiptTime`表示消息已读回执时间戳, 如果有此字段, 说明此时间戳之前的所有消息对方均已读
目前仅对`'p2p'`会话起作用
此字段不一定有, 只有对方发送过已读回执之后才会有
调用接口发送消息已读回执来发送消息已读回执
调用接口查询消息是否被对方读过了来查询消息是否被对方读过了

发送消息已读回执

目前只支持`'p2p'`会话
如果没有传入消息, 则直接返回成功
如果已经发送过比传入的消息的时间戳大的已读回执, 那么直接返回成功
参数`msg`为要发送已读回执的会话的最后一条收到的消息, 可以直接通过session.lastMsg来获取此消息

```
nim.sendMsgReceipt({
    msg: session.lastMsg,
    done: sendMsgReceiptDone
});
function sendMsgReceiptDone(error, obj) {
    console.log('发送消息已读回执' + (!error?'成功':'失败'), error, ob
j);
}
```

查询消息是否被对方读过了

目前只支持'p2p'会话

```
var isRemoteRead = nim.isMsgRemoteRead(msg);
```

群组

群组功能概述

SDK 提供了普通群以及高级群两种形式的群功能。 高级群拥有更多的权限操作，两种群聊形式在共有操作上保持了接口一致。**推荐 APP 开发时只选择一种群类型进行开发。**普通群和高级群在原则上是不能互相转换的，他们的群类型在创建时就已经确定。在 `SDK 2.4.0` 版本后，高级群可以拥有普通群的全部功能，推荐使用高级群进行开发。

普通群 开发手册中所提及的普通群都等同于 Demo 中的讨论组。普通群没有权限操作，适用于快速创建多人会话的场景。每个普通群只有一个管理员。管理员可以对群进行增减员操作，普通成员只能对群进行增员操作。在添加新成员的时候，并不需要经过对方同意。
高级群 高级群在权限上有更多的限制，权限分为群主、管理员、以及群成员。

初始化参数

这里的参数并不是所有的初始化参数, 请查阅初始化SDK, 以及其它章节的初始化参数
连接初始化参数
多端登录初始化参数
消息初始化参数
群组初始化参数

用户资料初始化参数
好友关系初始化参数
用户关系初始化参数
会话初始化参数
系统通知初始化参数
同步完成

完整的初始化代码

请参考处理群通知消息

示例代码

```
var nim = NIM.getInstance({
  onteams: onTeams,
  onsynccreateteam: onCreateTeam,
  onteammembers: onTeamMembers,
  //onsyncteammembersdone: onSyncTeamMembersDone,
  onupdateteammember: onUpdateTeamMember,
  shouldCountNotifyUnread: function (msg) {
    // 根据msg的属性自己添加过滤器
    if (msg.something === someting) {
    return true
    }
  }
});
function onTeams(teams) {
  console.log('收到群列表', teams);
  data.teams = nim.mergeTeams(data.teams, teams);
  onInvalidTeams(teams.invalid);
}
function onInvalidTeams(teams) {
  data.teams = nim.cutTeams(data.teams, teams);
  data.invalidTeams = nim.mergeTeams(data.invalidTeams, teams);
  refreshTeamsUI();
}
function onCreateTeam(team) {
  console.log('你创建了一个群', team);
  data.teams = nim.mergeTeams(data.teams, team);
  refreshTeamsUI();
  onTeamMembers({
    teamId: team.teamId,
    members: owner
  });
}
function refreshTeamsUI() {
  // 刷新界面
}
```

```
function onTeamMembers(obj) {
  console.log('群id', teamId, '群成员', members);
  var teamId = obj.teamId;
  var members = obj.members;
  data.teamMembers = data.teamMembers || {};
  data.teamMembers[teamId] = nim.mergeTeamMembers(data.teamMembers[
teamId], members);
  data.teamMembers[teamId] = nim.cutTeamMembers(data.teamMembers[te
amId], members.invalid);
  refreshTeamMembersUI();
}
// function onSyncTeamMembersDone() {
//     console.log('同步群成员列表完成');
// }
function onUpdateTeamMember(teamMember) {
  console.log('群成员信息更新了', teamMember);
  onTeamMembers({
    teamId: teamMember.teamId,
    members: teamMember
  });
}
function refreshTeamMembersUI() {
  // 刷新界面
}
```

参数解释

onteams, 同步群列表的回调, 会传入群数组
teams的属性invalid包含退出的群
此回调是增量回调, 可以调用nim.mergeTeams和nim.cutTeams来合并数据
onsynccreateteam, 当前登录帐号在其它端创建群之后, 会收到此回调, 会传入群对象
onteammembers, 同步群成员的回调, 一个群对应一个回调, 会传入群成员数组
可以调用nim.mergeTeamMembers和nim.cutTeamMembers来合并数据
onsyncteammembersdone, Deprecated!, 当所有群的群成员同步结束时, 会调用此回调
onupdateteammember, 群成员信息更新后的回调, 会传入群成员对象, 不过此时的信息是不完整的, 只会包括被更新的字段。当前登录帐号在其它端修改自己的群属性时也会收到此回调。
onCreateTeam, 创建群的回调, 此方法接收一个参数, 包含群信息和群主信息
onUpdateTeam, 更新群的回调, 此方法接收一个参数, 更新后的群信息
onAddTeamMembers, 新成员入群的回调, 此方法接收一个参数, 包含群信息和群

成员信息

`onRemoveTeamMembers`, 有人出群的回调, 此方法接收一个参数, 包含群信息和群成员账号

`onUpdateTeamManagers`, 更新群管理员的回调, 此方法接收一个参数, 包含群信息和管理员信息

`onDismissTeam`, 解散群的回调, 此方法接收一个参数, 包含被解散的群id

`onTransferTeam`, 移交群的回调, 此方法接收一个参数, 包含群信息和新老群主信息

`onUpdateTeamMembersMute`, 更新群成员禁言状态的回调, 此方法接收一个参数, 包含群信息和禁言状态信息

`shouldCountNotifyUnread`, 群消息通知是否加入未读数开关，是一个函数，如果返回true，则计入未读数，否则不计入

群对象

群对象有如下字段

- `teamId`: 群Id
- `appId`: 群所属的app的id
- `type`: 群类型
- `name`: 群名字
- `avatar`: 群头像
- `intro`: 群简介
- `announcement`: 群公告
- `joinMode`: 群加入方式, 仅限高级群
- `beInviteMode`: 群被邀请模式, 仅限高级群
- `inviteMode`: 群邀请模式, 仅限高级群
- `updateTeamMode`: 群信息修改权限, 仅限高级群
- `updateCustomMode`: 群信息自定义字段修改权限, 仅限高级群
- `owner`: 群主
- `level`: 群人数上限
- `memberNum`: 群成员数量
- `memberUpdateTime`: 群成员最后更新时间

- `createTime`: 群创建时间
- `updateTime`: 群最后更新时间
- `custom`: 第三方扩展字段, 开发者可以自行扩展, 建议封装成JSON格式字符串
- `serverCustom`: 第三方服务器扩展字段, 开发者可以自行扩展, 建议封装成JSON格式字符串
- `valid`: 是否有效, 解散后该群无效
- `validToCurrentUser`: 该群是否对当前用户有效, 如果无效, 那么说明被踢了
- `mute`: 是否禁言, 禁言状态下成员不能发送消息
- `muteType`: 禁言类型
- `none`: 都不禁言
- `normal`: 普通成员禁言，即普通成员不能发消息
- `all`: 全体禁言，即所有成员均不能发消息

群类型

群对象有一个字段`type`来标明群类型, 具体类型如下
- `'normal'` (普通群)
- `'advanced'` (高级群)

群加入方式

群加入方式有以下几种
- `'noVerify'` (不需要验证)
- `'needVerify'` (需要验证)
- `'rejectAll'` (禁止任何人加入)

群被邀请模式

群被邀请模式有以下几种
- `'needVerify'` (需要邀请方同意)
- `'noVerify'` (不需要邀请方同意)

群邀请模式

群邀请模式有以下几种
- `'manager'` (只有管理员/群主可以邀请他人入群)
- `'all'` (所有人可以邀请他人入群)

群信息修改权限

群信息修改权限有以下几种
- `'manager'` (只有管理员/群主可以修改)
- `'all'` (所有人可以修改)

群信息自定义字段修改权限

群信息自定义字段修改权限有以下几种
- `'manager'` (只有管理员/群主可以修改)
- `'all'` (所有人可以修改)

群成员对象

群成员对象有如下字段
- `teamId`: 群ID
- `account`: 帐号
- `type`: 群成员类型
- `nickInTeam`: 在群里面的昵称

- `active`: 普通群拉人进来的时候, 被拉的人处于未激活状态, 未激活状态下看不到这个群, 当有人说话后自动转为激活状态, 能看到该群
- `joinTime`: 入群时间
- `updateTime`: 更新时间

群成员类型

`'normal'` (普通成员)
`'owner'` (群主)
`'manager'` (管理员)

创建群

普通群不可以设置群加入方式
高级群的群加入方式默认为 `'needVerify'`
高级群的群被邀请模式默认为`'needVerify'`
高级群的群邀请模式默认为`'manager'`
高级群的群信息修改权限默认为`'manager'`
高级群的群信息自定义字段修改权限默认为`'manager'`
普通群的创建者可以看到所有的群成员, 而被邀请的群成员在有人发消息之后才能看到该群, 而且会先收到一条类型为`'addTeamMembers'`的群通知消息, 然后会收到其它群消息。
高级群被邀请的群成员会收到一条类型为类型为`'teamInvite'`的系统通知, 群成员只有接受邀请之后才会出现在该群中。
接受邀请后, 所有群成员会收到一条类型为`'acceptTeamInvite'`的群通知消息。
拒绝邀请后, 群主会收到一条类型为`'rejectTeamInvite'`的系统通知。
`ps`: 附言, 选填, 开发者也可以使用JSON格式的字符串来扩展此内容

```
// 创建普通群
nim.createTeam({
  type: 'normal',
  name: '普通群',
  avatar: 'avatar',
  accounts: ['a1', 'a2'],
  ps: '我建了一个普通群',
  done: createTeamDone
});
// 创建高级群
nim.createTeam({
  type: 'advanced',
  name: '高级群',
  avatar: 'avatar',
  accounts: ['a1', 'a2'],
  intro: '群简介',
  announcement: '群公告',
  // joinMode: 'needVerify',
  // beInviteMode: 'needVerify',
  // inviteMode: 'manager',
  // updateTeamMode: 'manager',
  // updateCustomMode: 'manager',
  ps: '我建了一个高级群',
  custom: '群扩展字段，建议封装成JSON格式字符串',
  done: createTeamDone
});
function createTeamDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('创建' + obj.team.type + '群' + (!error?'成功':'失败'))
;
  if (!error) {
    onCreateTeam(obj.team, obj.owner);
  }
}
```

发送群消息

发送群消息时只需将上文中各个发送消息接口的`scene`替换为`'team'`，将`to`替换为`群ID`。

接收群消息

参考上文的接收消息

更新群

普通群不可以更新
群加入方式
群被邀请模式
群邀请模式
群信息修改权限
群信息自定义字段修改权限
更新群后, 所有群成员会收到一条类型为'updateTeam'的群通知消息。此类群通知消息的from字段的值为更新群的人的帐号, to字段的值为对应的群ID, attach有一个字段team的值为被更新的群信息

```
nim.updateTeam({
  teamId: 123,
  name: '群名字',
  avatar: 'avatar',
  intro: '群简介',
  announcement: '群公告',
  joinMode: 'needVerify',
  custom: '自定义字段',
  done: updateTeamDone
});
function updateTeamDone(error, team) {
  console.log(error);
  console.log(team);
  console.log('更新群' + (!error?'成功':'失败'));
}
```

拉人入群

普通群, 拉人入群后, 所有群成员会收到一条类型为'addTeamMembers'的群通知消息。此类群通知消息的from字段的值为拉人的人的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群对象, attach有一个字段accounts的值为被拉的人的帐号列表, attach有一个字段members的值为被拉的群成员列表。

被邀请的群成员在有人说话后才能看到该群, 而且会先收到一条类型为'addTeamMembers'的群通知消息, 然后会收到其它群消息。
高级群的群主和管理员在邀请成员加入群（通过操作创建群或拉人入群）之后, 被邀请的人会收到一条类型为'teamInvite'的[系统通知]/docs/product/IM即时通讯/SDK开发集成/Web开发集成/系统通知), 此类系统通知的from字段的值为邀请方的帐号, to字段的值为对应的群ID, 此类系统通知的attach有一个字段team的值为被邀请进入的群, 被邀请的人可以选择接受邀请或者拒绝邀请。
如果接受邀请, 那么该群的所有群成员会收到一条类型为'acceptTeamInvite'的群通知消息, 此类群通知消息的from字段的值为接受入群邀请的人的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群对象, attach有一个字段members的值为接收入群邀请的群成员列表。
如果拒绝邀请, 那么邀请你的人会收到一条类型为'rejectTeamInvite'的系统通知, 此类系统通知的from字段的值为拒绝入群邀请的用户的帐号, to字段的值为对应的群ID。
accounts: 因为群数量限制导致无法加入群的帐号列表。
ps: 附言, 选填, 开发者也可以使用JSON格式的字符串来扩展此内容

```
nim.addTeamMembers({
  teamId: 123,
  accounts: ['a3', 'a4'],
  ps: '加入我们的群吧',
  done: addTeamMembersDone
});
function addTeamMembersDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('入群邀请发送' + (!error?'成功':'失败'));
}
```

踢人出群

踢人出群后, 所有群成员会收到一条类型为'removeTeamMembers'的群通知消息。此类群通知消息的from字段的值为踢人的人的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群对象, attach有一个字段accounts的值为被踢的人的帐号列表。

```
nim.removeTeamMembers({
  teamId: 123,
  accounts: ['a3', 'a4'],
  done: removeTeamMembersDone
});
function removeTeamMembersDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('踢人出群' + (!error?'成功':'失败'));
}
```

接受入群邀请

高级群的群主和管理员在邀请成员加入群（通过操作创建群或拉人入群）之后，被邀请的人会收到一条类型为'teamInvite'的系统通知，此类系统通知的from字段的值为邀请方的帐号，to字段的值为对应的群ID，此类系统通知的attach有一个字段team的值为被邀请进入的群，被邀请的人可以选择接受邀请或者拒绝邀请。
如果接受邀请，那么该群的所有群成员会收到一条类型为'acceptTeamInvite'的群通知消息，此类群通知消息的from字段的值为接受入群邀请的人的帐号，to字段的值为对应的群ID，attach有一个字段team的值为对应的群对象，attach有一个字段members的值为接收入群邀请的群成员列表。
如果拒绝邀请，那么邀请你的人会收到一条类型为'rejectTeamInvite'的系统通知，此类系统通知的from字段的值为拒绝入群邀请的用户的帐号，to字段的值为对应的群ID。
参数from填邀请方的帐号

```
// 假设 sysMsg 是通过回调 `onsysmsg` 收到的系统通知
nim.acceptTeamInvite({
  idServer: sysMsg.idServer,
  teamId: 123,
  from: 'zyy1',
  done: acceptTeamInviteDone
});
function acceptTeamInviteDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('接受入群邀请' + (!error?'成功':'失败'));
}
```

拒绝入群邀请

高级群的群主和管理员在邀请成员加入群（通过操作创建群或拉人入群）之后，被邀请的人会收到一条类型为'teamInvite'的系统通知，此类系统通知的from字段的值为邀请方的帐号，to字段的值为对应的群ID，此类系统通知的attach有一个字段team的值为被邀请进入的群，被邀请的人可以选择接受邀请或者拒绝邀请。
如果接受邀请，那么该群的所有群成员会收到一条类型为'acceptTeamInvite'的群通知消息，此类群通知消息的from字段的值为接受入群邀请的人的帐号，to字段的值为对应的群ID，attach有一个字段team的值为对应的群对象，attach有一个字段members的值为接收入群邀请的群成员列表。
如果拒绝邀请，那么邀请你的人会收到一条类型为'rejectTeamInvite'的系统通知，此类系统通知的from字段的值为拒绝入群邀请的用户的帐号，to字段的值为对应的群ID。
参数from填邀请方的帐号
ps：附言，选填，开发者也可以使用JSON格式的字符串来扩展此内容

```
// 假设 sysMsg 是通过回调 `onsysmsg` 收到的系统通知
nim.rejectTeamInvite({
  idServer: sysMsg.idServer,
  teamId: 123,
  from: 'zyy1',
  ps: '就不',
  done: rejectTeamInviteDone
});
function rejectTeamInviteDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('拒绝入群邀请' + (!error?'成功':'失败'));
}
```

申请入群

用户可以申请加入高级群，目标群的群主和管理员会收到一条类型为'applyTeam'的系统通知，此类系统通知的from字段的值为申请方的帐号，to字段的值为对应的群ID，高级群的群主和管理员在收到入群申请后，可以选择通过或者拒绝入群申请。

如果通过申请, 那么该群的所有群成员会收到一条类型为'passTeamApply'的群通知消息, 此类群通知消息的from字段的值为通过入群申请的人的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群对象, attach有一个字段account的值为申请方的帐号, attach有一个字段members的值为被通过申请的群成员列表。
如果拒绝申请, 那么申请人会收到一条类型为'rejectTeamApply'的系统通知, 此类系统通知的from字段的值为拒绝方的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群。
ps: 附言, 选填, 开发者也可以使用JSON格式的字符串来扩展此内容

```
nim.applyTeam({
  teamId: 123,
  ps: '请加',
  done: applyTeamDone
});
function applyTeamDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('申请入群' + (!error?'成功':'失败'));
}
```

通过入群申请

用户可以申请加入高级群, 目标群的群主和管理员会收到一条类型为'applyTeam'的系统通知, 此类系统通知的from字段的值为申请方的帐号, to字段的值为对应的群ID, 高级群的群主和管理员在收到入群申请后, 可以选择通过或者拒绝入群申请。
如果通过申请, 那么该群的所有群成员会收到一条类型为'passTeamApply'的群通知消息, 此类群通知消息的from字段的值为通过入群申请的人的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群对象, attach有一个字段account的值为申请方的帐号, attach有一个字段members的值为被通过申请的群成员列表。
如果拒绝申请, 那么申请人会收到一条类型为'rejectTeamApply'的系统通知, 此类系统通知的from字段的值为拒绝方的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群。
参数from填申请方的帐号, 该参数的名字在v1.3.0版本中从account变为from

```
// 假设 sysMsg 是通过回调 `onsysmsg` 收到的系统通知
nim.passTeamApply({
  idServer: sysMsg.idServer,
  teamId: 123,
  from: 'a2',
  done: passTeamApplyDone
});
function passTeamApplyDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('通过入群申请' + (!error?'成功':'失败'));
}
```

拒绝入群申请

用户可以申请加入高级群, 目标群的群主和管理员会收到一条类型为'applyTeam'的系统通知, 此类系统通知的from字段的值为申请方的帐号, to字段的值为对应的群ID, 高级群的群主和管理员在收到入群申请后, 可以选择通过或者拒绝入群申请。

如果通过申请, 那么该群的所有群成员会收到一条类型为'passTeamApply'的群通知消息, 此类群通知消息的from字段的值为通过入群申请的人的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群对象, attach有一个字段account的值为申请方的帐号, attach有一个字段members的值为被通过申请的群成员列表。

如果拒绝申请, 那么申请人会收到一条类型为'rejectTeamApply'的系统通知, 此类系统通知的from字段的值为拒绝方的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群。

参数from填申请方的帐号, 该参数的名字在v1.3.0版本中从account变为from

ps: 附言, 选填, 开发者也可以使用JSON格式的字符串来扩展此内容

```
// 假设 sysMsg 是通过回调 `onsysmsg` 收到的系统通知
nim.rejectTeamApply({
  idServer: sysMsg.idServer,
  teamId: 123,
  from: 'a2',
  ps: '就不',
  done: rejectTeamApplyDone
});
function rejectTeamApplyDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('拒绝入群申请' + (!error?'成功':'失败'));
}
```

添加群管理员

添加群管理员后, 所有群成员会收到一条类型为'addTeamManagers'的群通知消息。此类群通知消息的from字段的值为添加群管理员的人的帐号, to字段的值为对应的群ID, attach有一个字段accounts的值为被加为管理员的帐号列表, attach有一个字段members的值为被加为管理员的群成员列表

```
nim.addTeamManagers({
  teamId: 123,
  accounts: ['a2', 'a3'],
  done: addTeamManagersDone
});
function addTeamManagersDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('添加群管理员' + (!error?'成功':'失败'));
}
```

移除群管理员

移除群管理员后, 所有群成员会收到一条类型为'removeTeamManagers'的群通知消息。此类群通知消息的from字段的值为移除群管理员的人的帐号, to字段的值为对应的群ID, attach有一个字段accounts的值为被移除的管理员的帐号列表, attach有一个字段members的值为被移除管理员的群成员列表

```
nim.removeTeamManagers({
  teamId: 123,
  accounts: ['a2', 'a3'],
  done: removeTeamManagersDone
});
function removeTeamManagersDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('移除群管理员' + (!error?'成功':'失败'));
}
```

主动退群

主动退群后, 所有群成员会收到一条类型为'leaveTeam'的群通知消息。此类群通知消息的from字段的值为退群的人的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群对象。

```
nim.leaveTeam({
  teamId: 123,
  done: leaveTeamDone
});
function leaveTeamDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('主动退群' + (!error?'成功':'失败'));
}
```

转让群

转让群后, 所有群成员会收到一条类型为'transferTeam'的群通知消息。此类群通知消息的from字段的值为转让群的人的帐号, to字段的值为对应的群ID, attach有一个字段team的值为对应的群对象, attach有一个字段account的值为为新群主的帐号, attach有一个字段members的值为包含新旧群主的群成员列表。如果转让群的同时离开群, 那么相当于调用主动退群来离开群, 所有群成员会再收到一条类型为'leaveTeam'的群通知消息。

```
nim.transferTeam({
  teamId: 123,
  account: 'zyy2',
  leave: false,
  done: transferOwnerDone
});
function transferOwnerDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('转让群' + (!error?'成功':'失败'));
}
```

解散群

解散群后, 所有群成员会收到一条类型为'dismissTeam'的群通知消息。此类群通知消息的from字段为解散群的人的帐号, to字段的值为被对应的群ID。

```
nim.dismissTeam({
  teamId: 123,
  done: dismissTeamDone
});
function dismissTeamDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('解散群' + (!error?'成功':'失败'));
}
```

修改自己的群属性

目前支持修改的属性有这些
- `nickInTeam`: 自己在群里面的群昵称
- 更新昵称后, 所有其它在线的群成员会收到初始化SDK时传入的`onupdateteammember`回调
- `muteTeam`: 是否关闭此群的消息提醒, `true`表示关闭提醒, 但是SDK仍然会收到这个群的消息, SDK只是记录这个设置, 具体根据这个设置要执行的操作由第三方APP决定, 设置之后可以调用接口是否需要群消息通知来查询是否需要群消息通知
- `muteNotiType`: 4.3.0版本添加，功能扩展了muteTeam属性，老版本无此

属性，若同时定义了muteTeam与muteNotiType，则以muteNotiType为准。0表示接收提醒，1表示关闭提醒，2表示仅接收管理员提醒
- `custom`: 第三方扩展字段, 开发者可以自行扩展, 建议封装成JSON格式字符串

```
nim.updateInfoInTeam({
  teamId: 123,
  // 此参数为可选参数
  // nickInTeam: '群昵称',
  // 静音群, 此参数为可选参数
  // muteTeam: true,
  // muteNotiType: 1,
  // 第三方扩展字段
  // custom: '{}'
  done: updateInfoInTeamDone
});
function updateInfoInTeamDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('修改自己的群属性' + (!error?'成功':'失败'));
}
```

修改别人的群昵称

所有其它在线的群成员会收到会收到初始化SDK时传入的`onupdateteammember`回调

```
nim.updateNickInTeam({
  teamId: 123,
  account: 'a2',
  nickInTeam: '群昵称',
  done: updateNickInTeamDone
});
function updateNickInTeamDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('修改自己的群属性' + (!error?'成功':'失败'));
}
```

更新群成员禁言状态

更新群成员禁言状态后, 所有群成员会收到一条类型为'updateTeamMute'的群通知消息。此类群通知消息的from字段的值为操作方, to字段的值为对应的群ID, attach有一个字段team的值为对应的群对象, attach有一个字段account的值为被禁言的帐号, attach有一个字段members的值为被禁言的群成员列表。

```
nim.updateMuteStateInTeam({
  teamId: '123',
  account: 'a',
  mute: true,
  done: updateMuteStateInTeamDone
})
function updateMuteStateInTeamDone(error, obj) {
  console.log('更新群成员禁言状态' + (!error?'成功':'失败'), error, obj
);
}
```

获取群禁言成员列表

```
nim.getMutedTeamMembers({
  teamId: 'teamId',
  done: getMutedTeamMembersDone
})
function getMutedTeamMembersDone (error, obj) {
  console.log('获取群禁言成员列表' + (!error?'成功':'失败'))
  console.log(obj)
}
```

群整体禁言

只有群主有权限对群组整体禁言，一旦标记禁言，则组内相关人员均被禁言
请求参数：
teamId 待禁言群id
type 禁言类型
none 解除禁言
normal 普通群成员被禁言
all 全体禁言（包括管理员），暂不支持客户端调用，仅可通过server api处理
状态更新：

返回的群属性有两个字段供开发者判断：

`team.mute` true/false，群是否被禁言

`team.muteType` 群禁言类型 `none/normal/all`

```
  nim.muteTeamAll({
    teamId: 'teamId',
    type: 'none',
    done: muteTeamAllDone
  })
  function muteTeamAllDone (error, obj) {
    console.log('群禁言: ' + (!error?'成功':'失败'))
    console.log(obj)
  }
```

获取群

开发者可以调用此接口获取群资料

```
nim.getTeam({
  teamId: 123,
  done: getTeamDone
});
function getTeamDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('获取群' + (!error?'成功':'失败'));
}
```

获取群列表

如果开发者在初始化SDK的时候设置了`syncTeams`为`false`, 那么就收不到`onteams`回调, 可以调用此方法来获取群列表

```
nim.getTeams({
  done: getTeamsDone
});
function getTeamsDone(error, teams) {
  console.log(error);
  console.log(teams);
  console.log('获取群列表' + (!error?'成功':'失败'));
  if (!error) {
    onTeams(teams);
  }
}
```

获取群成员

如果开发者在初始化SDK时选择设置了`syncTeamMembers`为`false`, 那么就收不到`onteammembers`回调, 可以调用此方法来获取群成员列表
接受入群邀请之后调用此方法来获取群成员列表

```
nim.getTeamMembers({
  teamId: 123,
  done: getTeamMembersDone
});
function getTeamMembersDone(error, obj) {
  console.log(error);
  console.log(obj);
  console.log('获取群成员' + (!error?'成功':'失败'));
  if (!error) {
    onTeamMembers(obj);
  }
}
```

是否需要群消息通知

此接口用于查询是否需要群消息通知
成功时会附上一个 map, key 是群 ID, value 是一个布尔值, 表示该群是否需要群消息通知
调用接口修改自己的群属性来关闭/开启某个群的消息提醒

```
nim.notifyForNewTeamMsg({
  teamIds: ['123'],
  done: notifyForNewTeamMsgDone
})
function notifyForNewTeamMsgDone(error, map) {
  console.log(error);
  console.log(map);
  console.log('查询是否需要群消息通知' + (!error?'成功':'失败'));
}
```

群消息已读通知

设置了needMsgReceipt字段(即需要业务已读回执)的群消息，本端在线状态下，对方已读会收到相应的通知消息，通知消息格式如下：

```
  {
    teamMsgReceipts: [
      {
        account: "cs3",
        idClient: "5b77d3ff7eb06af5567f56647518694b",
        idServer: "68953284018340",
        read: "1",
        teamId: "1027484",
        unread: "1"
      }
    ]
```

发送消息需要业务已读参考消息类型

```
  NIM.getInstance({
    // ...
    onTeamMsgReceipt: onTeamMsgReceipt
  })
  function onTeamMsgReceipt (obj) {
    console.log(obj)
  }
```

标记群组消息已读

对应于群组消息发送时配置了needMsgReceipt字段的群组消息，接收方可以对消息发送已读回执

该接口可以发送多个群组、多条消息的已读回执
请求参数：
teamMsgReceipts 需要发送回执的消息配置列表
teamMsgReceipt.teamId 群组id
teamMsgReceipt.idServer 消息的serverid
done 回调函数
回调结果:
error 第一个参数，如果为null表示成功
obj 第二个参数，发送的参数(用于校验)
content 第三个参数
content.teamMsgReceipts 失败的账号列表

```
nim.sendTeamMsgReceipt({
  teamMsgReceipts: [{
    teamId: '1027484',
    idServer: '68953284018302'
  }],
  done: sendTeamMsgReceiptDone
})
function sendTeamMsgReceiptDone (error, obj, content) {
  console.log('标记群组消息已读' + (!error?'成功':'失败'));
}
```

查询群组消息已读、未读数量

对应于群组消息发送时配置了needMsgReceipt字段的群组消息，接收方可以对消息发送已读回执
该接口可以查询群组消息已读未读数量，可以同时发送多个群消息配置
请求参数：
teamMsgReceipts 需要发送回执的消息配置列表
teamMsgReceipt.teamId 群组id
teamMsgReceipt.idServer 消息的serverid
done 回调函数
回调结果:
error 第一个参数，如果为null表示成功
obj 第二个参数，发送的参数(用于校验)

content 第三个参数，失败的账号列表
如果是多个群消息配置的结果，则该字段的teamMsgReceipts 数组顺序与查询配置一致

```
  nim.getTeamMsgReads({
    teamMsgReceipts: [{
      teamId: '1027484',
      idServer: '68953284018302'
    }],
    done: getTeamMsgReadsDone
  })
  function getTeamMsgReadsDone (error, obj, content) {
    console.log('获取群组消息已读' + (!error?'成功':'失败'));
    /* content.teamMsgReceipts 为
      [ {
        idClient: "c7575fca32bf142787986e752fdeff6a"
        idServer: "68527276949899"
        read: "0"
        teamId: "1021136"
        unread: "187"
      } ]
    */
```

查询群组消息未读账号列表

对应于群组消息发送时配置了needMsgReceipt字段的群组消息，接收方可以对消息发送已读回执
该接口可以查询群组消息未读账号列表，发送配置
请求参数：
teamMsgReceipt 需要发送回执的消息配置
teamMsgReceipt.teamId 群组id
teamMsgReceipt.idServer 消息的serverid
done 回调函数
回调结果:
error 第一个参数，如果为null表示成功
obj 第二个参数，发送的参数(用于校验)
content 第三个参数，账号列表
idClient 消息idClient
readAccounts 已读帐号列表

`unreadAccounts` 未读帐号列表

```
nim.getTeamMsgReadAccounts({
  teamMsgReceipt: {
    teamId: '1027484',
    idServer: '68953284018302'
  },
  done: getTeamMsgReadAccountsDone
})
function getTeamMsgReadAccountsDone (error, obj, content) {
  console.log('获取群组消息已读' + (!error?'成功':'失败'));
  /* content.teamMsgReceipt 为
    idClient: "c7575fca32bf142787986e752fdeff6a",
    readAccounts: Array[],
    unreadAccounts: Array[]
  */
```

聊天室

请查阅集成方式来下载并引入 SDK 文件

聊天室功能概述

目前不支持通过 SDK 接口建立/解散聊天室。
进入聊天室时必须建立新的连接，退出聊天室或者被踢会断开连接，在聊天室中掉线会有自动重连，开发者需要监听聊天室连接状态来做出正确的界面表现。
支持聊天人数无上限。
聊天室只允许用户手动进入，无法进行邀请。
支持同时进入多个聊天室，会建立多个连接。
断开聊天室连接后，服务器不会再推送该聊天室的消息给此用户。
在进行一切操作之前，必须先进入聊天室。即必须先初始化好聊天室并且收到`onconnect`回调。
聊天室成员分固定成员和游客两种类型。

获取聊天室服务器地址

初始化聊天室之前要先获取聊天室服务器地址, 有两种方式

如果开发者有 NIM 的实例, 那么可以直接从 IM 连接上获取聊天室服务器地址, 示例代码如下

```
nim.getChatroomAddress({
  chatroomId: 'chatroomId',
  done: getChatroomAddressDone
});
function getChatroomAddressDone(error, obj) {
  console.log('获取聊天室地址' + (!error?'成功':'失败'), error, obj);
}
```

如果开发者没有 NIM 的实例, 那么需要从服务器获取聊天室服务器地址, 请参考 demo 来查看具体的做法

初始化聊天室

初始化聊天室之前，必须拿到聊天室服务器地址
此接口为单例模式, 对于同一个账号, 永远返回同一份实例, 即只有第一次调用会初始化一个实例
后续调用此接口会直接返回初始化过的实例, 同时也会调用接口更新聊天室配置更新传入的配置
后续调用此接口时, 如果连接已断开, 会自动建立连接
当发生掉线时，SDK会自动进行重连
在收到`onconnect`回调之后说明成功进入聊天室, 此时可以进行其他的聊天室操作了.
匿名登录聊天室
SDK支持用户以游客身份访问聊天室，即通过配置参数 `isAnonymous`: true实现
第一次使用匿名方式登录聊天室时（新建实例），无需填写account参数，但需要在登录以后从chatroom实例中获取SDK生成的该参数
使用匿名方式登录聊天室，必须填写用户昵称（非匿名方式为选填），建议填写头像

为防止聊天室不断的被创建新实例，建议用户需要更新聊天室配置时（匿名模式），在update或第二次getInstance时，将前一次获取的account通过参数传入

示例代码

```
// 注意这里，引入的 SDK 文件不一样的话，你可能需要使用 SDK.Chatroom.getInstance 来调用接口
// 非匿名方式登录
var chatroom = Chatroom.getInstance({
  appKey: 'appKey',
  account: 'account',
  token: 'token',
  chatroomId: 'chatroomId',
  chatroomAddresses: [
    'address1',
    'address2'
  ],
  onconnect: onChatroomConnect,
  onerror: onChatroomError,
  onwillreconnect: onChatroomWillReconnect,
  ondisconnect: onChatroomDisconnect,
  // 消息
  onmsgs: onChatroomMsgs
});
function onChatroomConnect(obj) {
  console.log('进入聊天室', obj);
  // 连接成功后才可以发消息
  var msg = chatroom.sendText({
    text: 'hello',
    done: function sendChatroomMsgDone (msgObj) {
    }
  })
}
function onChatroomWillReconnect(obj) {
  // 此时说明 `SDK` 已经断开连接，请开发者在界面上提示用户连接已断开，而且正在重新建立连接
  console.log('即将重连', obj);
}
function onChatroomDisconnect(error) {
  // 此时说明 `SDK` 处于断开状态，开发者此时应该根据错误码提示相应的错误信息，并且跳转到登录页面
  console.log('连接断开', error);
  if (error) {
    switch (error.code) {
    // 账号或者密码错误，请跳转到登录页面并提示错误
    case 302:
      break;
    // 被踢，请提示错误后跳转到登录页面
    case 'kicked':
      break;
```

```
      break;
    default:
      break;
    }
  }
}
function onChatroomError(error, obj) {
  console.log('发生错误', error, obj);
}
function onChatroomMsgs(msgs) {
  console.log('收到聊天室消息', msgs);
}
```

```
// 匿名方式登录
// 非匿名方式登录
var chatroom = Chatroom.getInstance({
  appKey: 'appKey',
  // account: 不需要填account
  // token: 不需要填account
  chatroomId: 'chatroomId',
  chatroomAddresses: [
    'address1',
    'address2'
  ],
  chatroomNick: 'chatroomNick',
  chatroomAvatar: 'chatroomAvatar',
  isAnonymous: true,
  onconnect: onChatroomConnect,
  // ...
});

function onChatroomConnect (obj) {
  // 该处chatroom为全局生成的实例
  window.account = chatroom.account
}
```

参数解释

appKey: 在云信管理后台查看应用的 appKey

account: 帐号, 应用内唯一

token: 帐号的 token, 用于建立连接

chatroomId: 聊天室 id

chatroomAddresses: 聊天室地址列表

chatroomNick: 进入聊天室后展示的昵称, 如果不设置并且托管了用户资料, 那么使用用户资料里面的昵称

chatroomAvatar: 进入聊天室后展示的头像, 如果不设置并且托管了用户资料, 那么使用用户资料里面的头像

chatroomCustom: 扩展字段, 设置了之后, 通过获取聊天室成员列表获取的聊天室成员信息会包含此字段
推荐使用JSON格式构建, 非JSON格式的话, Web端会正常接收, 但是会被其它端丢弃
chatroomEnterCustom: 扩展字段, 如果填了, 那么其它聊天室成员收到的聊天室通知消息的attach.custom的值为此字段
推荐使用JSON格式构建, 非JSON格式的话, Web端会正常接收, 但是会被其它端丢弃
onconnect: 连接建立后的回调, 会传入聊天室信息
onwillreconnect: 即将重连的回调
此时说明 SDK 已经断开连接, 请开发者在界面上提示用户连接已断开, 而且正在重新建立连接
此回调会收到一个对象, 包含额外的信息, 有以下字段
duration: 距离下次重连的时间
retryCount: 重连尝试的次数
ondisconnect: 断开连接后的回调
此时说明 SDK 处于断开状态, 开发者此时应该根据错误码提示相应的错误信息, 并且跳转到登录页面
此回调会收到一个对象, 包含错误的信息, 有以下字段
code: 出错时的错误码, 可能为空
302: 账号或者密码错误
'kicked': 被踢
当code为'kicked'的时候, 此对象会有以下字段
reason: 被踢的原因
chatroomClosed: 聊天室关闭了
managerKick: 被管理员踢出
samePlatformKick: 不允许同一个帐号重复登录同一个聊天室
message: 文字描述的被踢的原因
onerror: 发生错误的回调, 会传入错误对象
onmsgs: 收到消息的回调, 会传入聊天室消息对象数组

退出聊天室

初始化聊天室并收到onconnect回调之后, 表明进入了聊天室
在收到onconnect回调后可以调用chatroom.disconnect();来退出聊天室
退出聊天室后可以调用chatroom.connect();来重新进入聊天室

切换聊天室

如果需要切换聊天室, 操作步骤如下
- 调用退出聊天室来退出聊天室
- 调用初始化聊天室来初始化新的聊天室

更新聊天室配置

聊天室设计为单例模式, 如果需要更新当前聊天室的配置, 那么可以调用此接口, 参数列表和格式跟Chatroom.getInstance保持一致, 以更新 token 为例

```
// 断开聊天室
chatroom.disconnect()
// 更新 token
chatroom.setOptions({
  token: 'newToken'
});
// 重新连接
chatroom.connect()
```

清除聊天室实例

web sdk 连接实例均为单例模式，但可以调用相应接口清除内存中记录的实例，即断开连接，清除内存消息记录及时间戳，方便开发者做到干净重连。

```
  var chatroom = Chatroom.getInstance({...})
  // 清除实例
  chatroom.destroy({
    done: function (err) {
      console.log('实例已被完全清除')
    }
  })
````

聊天室信息对象

聊天室信息对象有以下字段
- `id`: 聊天室 id
- `name`: 聊天室名字
- `announcement`: 聊天室公告
- `broadcastUrl`: 直播地址
- `custom`: 第三方扩展字段
  - 推荐使用`JSON`格式构建, 非`JSON`格式的话, Web端会正常接收, 但是会被其它端丢弃
- `createTime`: 创建时间
- `updateTime`: 更新时间
- `creator`: 创建者账号
- `onlineMemberNum`: 当前在线人数
- `mute` 是否禁言, 禁言状态下普通成员不能发送消息, 创建者和管理员可以发送消息

获取聊天室信息

```javascript
chatroom.getChatroom({
  done: getChatroomDone
});
function getChatroomDone(error, obj) {
  console.log('获取聊天室信息' + (!error?'成功':'失败'), error, obj);
}
```

更新聊天室信息

当更新聊天室信息时, 所有聊天室成员会收到类型为'updateChatroom'的聊天室通知消息。

可更新的字段有
- `chatroom`: 聊天室自有属性
- `chatroom.name`: 聊天室名字
- `chatroom.announcement`: 聊天室公告
- `chatroom.broadcastUrl`: 直播地址

- `chatroom.custom`: 第三方扩展字段
- `chatroom.queuelevel` 队列管理权限：0:所有人都有权限变更队列，1:只有主播管理员才能操作变更

其他参数:
- `needNotify` 是否需要下发对应的通知消息
- `custom` 对应的通知消息的扩展字段
- `done` 更新操作完成的回调

```
  chatroom.updateChatroom({
    chatroom: {
      name: 'newName',
      announcement: 'newAnnouncement',
      broadcastUrl: 'newBroadcastUrl',
      custom: 'newCustom'
    },
    needNotify: true,
    custom: 'biu',
    done: updateChatroomDone
  })
  function updateChatroomDone () {
    console.log('更新聊天室信息' + (!error?'成功':'失败'), error, obj)
;
  }
```

更新自己在聊天室内的信息

当更新自己在聊天室内的信息时, 所有聊天室成员会收到类型为`'updateMemberInfo'`的聊天室通知消息。

可更新的字段有
- `'nick'` 聊天室内的昵称
- `'avatar'` 聊天室内的头像
- `'custom'`: 第三方扩展字段

```
  chatroom.updateMyChatroomMemberInfo({
    member: {
    nick: 'newNick',
    avatar: 'newAvatar',
    custom: 'newCustom',
    },
    needNotify: true,
    custom: 'biu',
    done: updateMyChatroomMemberInfoDone
  })

  function updateMyChatroomMemberInfoDone (error, obj) {
    console.log('更新自己在聊天室内的信息' + (!error?'成功':'失败'), err
or, obj);
  }
```

聊天室消息

聊天室消息对象

聊天室消息对象有以下字段
- `chatroomId`: 聊天室 ID
- `idClient`: SDK生成的消息id, 在发送消息之后会返回给开发者, 开发者可以在发送消息的结果回调里面根据这个ID来判断相应消息的发送状态, 到底是发送成功了还是发送失败了, 然后根据此状态来更新页面的UI。如果发送失败, 那么可以重新发送此消息
- `from`: 消息发送方, 帐号
- `fromNick`: 消息发送方的昵称
- `fromAvatar`: 消息发送方的头像
- `fromCustom`: 消息发送方的扩展字段
- `fromClientType`: 发送方的设备类型
- `type`: 聊天室消息类型
- `flow`: 消息的流向
- 'in'表示此消息是收到的消息
- 'out'表示此消息是发出的消息
- `text`: 文本消息的文本内容, 请参考发送聊天室文本消息
- `file`: 文件消息的文件对象, 具体字段请参考图片对象、音频对象、视频对

象、文件对象, 请参考发送聊天室文件消息
- `geo`: 地理位置消息的地理位置对象, 请参考发送聊天室地理位置消息
- `tip`: 提醒消息的内容, 请参考发送聊天室提醒消息
- `content`: 自定义消息的消息内容, 开发者可以自行扩展, 建议封装成JSON格式字符串, 请参考发送聊天室自定义消息
- `attach`: 聊天室通知消息的附加信息, 参考聊天室通知消息的类型来查看详细解释
- `custom`: 扩展字段
- 推荐使用`JSON`格式构建, 非`JSON`格式的话, Web端会正常接收, 但是会被其它端丢弃
- `resend`: 是否是重发的消息
- `time`: 时间戳

聊天室消息类型

`'text'` (文本)
`'image'` (图片)
`'audio'` (音频)
`'video'` (视频)
`'file'` (文件)
`'geo'` (地理位置)
`'custom'` (自定义消息)
`'tip'` (提醒消息)
提醒消息用于会话内的状态提醒，如进入会话时出现的欢迎消息，或者会话命中敏感词后的提示消息等等.
`'notification'` (聊天室通知消息)
某些聊天室操作后所有聊天室成员会收到一条相应的聊天室通知消息, 详细介绍请参考聊天室通知消息的类型

聊天室通知消息的类型

聊天室通知消息是聊天室消息的一种, 请参考聊天室消息类型, 某些聊天室操作后所有聊天室成员会收到一条相应的聊天室通知消息
聊天室通知消息有一个字段`attach`包含了额外的信息, `attach`有一个字段`type`

来标识聊天室通知消息的类型

`memberEnter`

当有人进入聊天室时, 所有聊天室成员会收到类型为`'memberEnter'`的聊天室通知消息。

`memberExit`

当有人退出聊天室时, 所有聊天室成员会收到类型为`'memberExit'`的聊天室通知消息。

`addManager`

当有人被加为管理员时, 所有聊天室成员会收到类型为`'addManager'`的聊天室通知消息。

`removeManager`

当有人被移除管理员时, 所有聊天室成员会收到类型为`'removeManager'`的聊天室通知消息。

`addCommon`

当有人被加为普通成员时, 所有聊天室成员会收到类型为`'addCommon'`的聊天室通知消息。

`removeCommon`

当有人被移除普通成员时, 所有聊天室成员会收到类型为`'removeCommon'`的聊天室通知消息。

`blackMember`

当有人被加入黑名单时, 所有聊天室成员会收到类型为`'blackMember'`的聊天室通知消息。

`unblackMember`

当有人被移除黑名单时, 所有聊天室成员会收到类型为`'blackMember'`的聊天室通知消息。

`gagMember`

当有人被加入禁言名单时, 所有聊天室成员会收到类型为`'gagMember'`的聊天室通知消息。

`ungagMember`

当有人被移除禁言名单时, 所有聊天室成员会收到类型为`'ungagMember'`的聊天室通知消息。

`kickMember`

当有人被踢出聊天室时, 所有聊天室成员会收到类型为`'kickMember'`的聊天室

通知消息。

updateChatroom
当更新聊天室信息时, 所有聊天室成员会收到类型为'updateChatroom'的聊天室通知消息。

updateMemberInfo
当更新自己在聊天室内的信息时, 所有聊天室成员会收到类型为'updateMemberInfo'的聊天室通知消息。

addTempMute

removeTempMute
当有人被设置聊天室临时禁言时，所有聊天室成员会收到类型为'addTempMute' or 'removeTempMute'的聊天室通知消息。
muteRoom 聊天室被禁言了,只有管理员可以发言,其他人都处于禁言状态
unmuteRoom 聊天室解除全体禁言状态
attach的字段from为操作方的账号, fromNick为操作方的昵称, to为被操作方的账号, toNick为被操作方的昵称
如果是addTempMute, attach的字段duration代表本次禁言的时长
如果是removeTempMute, attach的字段duration代表解禁提前的时长

发送聊天室消息

包括以下接口

发送聊天室文本消息
预览聊天室文件
发送聊天室文件消息
发送聊天室地理位置消息
发送聊天室提醒消息
发送聊天室自定义消息
发送聊天室消息的配置选项

发送聊天室文本消息

```
var msg = chatroom.sendText({
  text: 'hello',
  done: sendChatroomMsgDone
});
console.log('正在发送聊天室text消息, id=' + msg.idClient);
function sendChatroomMsgDone(error, msg) {
  console.log('发送聊天室' + msg.type + '消息' + (!error?'成功':'失败'
) + ', id=' + msg.idClient, error, msg);
}
```

预览聊天室文件

开发者可以预览文件, 支持以下几种场景
通过参数`fileInput`传入文件选择 dom 节点或者节点 ID
通过参数`blob`传入 Blob 对象
通过参数`dataURL`传入包含 MIME type 和 base64 数据的 data URL, 此用法需要浏览器支持 window.Blob
SDK会将文件上传到文件服务器, 然后将拿到的文件对象在`done`回调中传给开发者, 文件对象有以下几种
图片对象
音频对象
视频对象
文件对象
开发者在拿到文件对象之后, 可以调用发送聊天室文件消息来发送文件消息。
文件大小限制为最大100M
高级浏览器会在上传前就检测文件大小
IE8/IE9 会在上传完成后检测文件大小

```
chatroom.previewFile({
  type: 'image',
  fileInput: fileInput,
  uploadprogress: function(obj) {
    console.log('文件总大小: ' + obj.total + 'bytes');
    console.log('已经上传的大小: ' + obj.loaded + 'bytes');
    console.log('上传进度: ' + obj.percentage);
    console.log('上传进度文本: ' + obj.percentageText);
  },
  done: function(error, file) {
    console.log('上传image' + (!error?'成功':'失败'));
    // show file to the user
    if (!error) {
      var msg = chatroom.sendFile({
        scene: 'p2p',
        to: 'account',
        file: file,
        done: sendChatroomMsgDone
      });
      console.log('正在发送聊天室image消息, id=' + msg.idClient);
    }
  }
});
```

发送聊天室文件消息

文件消息是聊天室消息的一种
开发者可以直接发送文件消息
支持以下几种场景
通过参数`fileInput`传入文件选择 dom 节点或者节点 ID
通过参数`blob`传入 Blob 对象
通过参数`dataURL`传入包含 MIME type 和 base64 数据的 data URL, 此用法需要浏览器支持 window.Blob
SDK会先将文件上传到文件服务器, 然后把拿到的文件对象在`uploaddone`回调中传给用户, 然后将其拼装成文件消息发送出去。
开发者也可以先预览聊天室文件来获取文件对象, 然后调用此接口发送文件消息。
直接发送文件消息的话会在`beforesend`回调里面传入SDK生成的`idClient`, 如果先预览文件再发送, 那么此接口会直接返回`idClient`
参数`type`指定了要发送的文件类型, 包括图片、音频、视频和普通文件, 对应的值分别为`'image'`、`'audio'`、`'video'`和`'file'`, 不传默认为`'file'`。

图片、音频、视频和普通文件的区别在于具体的文件信息不一样, 具体字段请参考
图片对象
音频对象
视频对象
文件对象
文件大小限制为最大100M
高级浏览器会在上传前就检测文件大小
IE8和IE9会在上传完成后检测文件大小

```
chatroom.sendFile({
  type: 'image',
  fileInput: fileInput,
  uploadprogress: function(obj) {
    console.log('文件总大小: ' + obj.total + 'bytes');
    console.log('已经上传的大小: ' + obj.loaded + 'bytes');
    console.log('上传进度: ' + obj.percentage);
    console.log('上传进度文本: ' + obj.percentageText);
  },
  uploaddone: function(error, file) {
    console.log('上传' + (!error?'成功':'失败'), error, file);
  },
  beforesend: function(msg) {
    console.log('正在发送聊天室image消息, id=' + msg.idClient);
  },
  done: sendChatroomMsgDone
});
```

发送聊天室地理位置消息

地理位置消息是聊天室消息的一种, geo参数请参考地理位置对象

```
var msg = chatroom.sendGeo({
  scene: 'p2p',
  to: 'account',
  geo: {
    lng: '116.3833',
    lat: '39.9167',
    title: 'Beijing'
  },
  done: sendChatroomMsgDone
});
console.log('正在发送聊天室geo消息, id=' + msg.idClient);
```

发送聊天室提醒消息

提醒消息是聊天室消息的一种
提醒消息用于会话内的状态提醒，如进入会话时出现的欢迎消息，或者会话命中敏感词后的提示消息等等.

```
var msg = chatroom.sendTipMsg({
  scene: 'p2p',
  to: 'account',
  tip: 'tip content',
  done: sendChatroomMsgDone
});
console.log('正在发送聊天室提醒消息, id=' + msg.idClient);
```

发送聊天室自定义消息

```
var value = Math.ceil(Math.random()*3);
var content = {
  type: 1,
  data: {
    value: value
  }
};
var msg = chatroom.sendCustomMsg({
  content: JSON.stringify(content),
  done: sendChatroomMsgDone
});
console.log('正在发送聊天室自定义消息, id=' + msg.idClient);
```

发送聊天室消息的配置选项

上面的各个发送消息的接口都可以配置额外的选项, 来满足开发者对服务器的自定义需求。
custom: 扩展字段
推荐使用JSON格式构建, 非JSON格式的话, Web端会正常接收, 但是会被其它端丢弃
下面给一个发送文本消息的例子, 发送其它消息的接口类似

```
var msg = chatroom.sendText({
  text: 'hello',
  custom: '{}',
  done: sendChatroomMsgDone
});
console.log('正在发送聊天室text消息, id=' + msg.idClient);
```

custom 过滤筛选自定义消息

```
  chatroom.getHistoryMsgs({
    timetag: 1451393192478,
    limit: 100,
    msgTypes: ['text', 'image'],
    done: getHistoryMsgsDone
  })

  function getHistoryMsgsDone(error, obj) {
    console.log('获取聊天室历史' + (!error?'成功':'失败'), error, obj.
msgs);
  }
```

聊天室成员

聊天室成员对象

聊天室成员对象有以下字段

chatroomId: 聊天室 ID
account: 账号
nick: 聊天室内的昵称
avatar: 聊天室内的头像
type: 聊天室成员类型
guest 是否是游客
blacked 是否被拉黑
gaged 是否被禁言
level: 级别
online: 是否在线, 只有固定成员才能离线, 对游客而言只能是在线
enterTime: 进入聊天室的时间, 如果离线, 无该字段

custom: 扩展字段
推荐使用JSON格式构建, 非JSON格式的话, Web端会正常接收, 但是会被其它端丢弃
updateTime: 更新时间
tempMuted: 是否被临时禁言
tempMuteDuration: 临时禁言剩余时长

聊天室成员类型

聊天室成员分为固定成员和游客两种。固定成员又分为房主、管理员、普通成员和受限成员四种。禁言用户和拉黑用户都属于受限用户。

'owner' (房主)
'manager' (管理员)
'restricted' (受限制, 被拉黑或者禁言)
'common' (普通成员)
'guest' (游客)

获取聊天室成员列表

guest: true表示获取游客, false表示获取非游客成员
游客列表按照游客进入聊天室的时间倒序排列
非游客（即固定成员）列表按照成为固定成员的时间倒序排列
当设置guest=false来获取非游客成员时, 默认会获取所有的固定成员, 包括不在线的, 可以设置onlyOnline=true来只获取在线的固定成员
time 分页用, 查找该时间戳之前的成员
默认 0 代表当前服务器时间
获取游客时, 此字段填上次获取的最后一个游客的enterTime
获取非游客时, 此字段填上次获取的最后一个非游客的updateTime
limit 分页用, 默认 100

```
chatroom.getChatroomMembers({
  guest: false,
  limit: 100,
  done: getChatroomMembersDone
});
function getChatroomMembersDone(error, obj) {
  console.log('获取聊天室成员' + (!error?'成功':'失败'), error, obj.me
mbers);
}
```

获取聊天室成员信息

accounts: 待查询的账号列表, 每次最多20个

```
chatroom.getChatroomMembersInfo({
  accounts: ['account1', 'account2'],
  done: getChatroomMembersInfoDone
});
function getChatroomMembersInfoDone(error, obj) {
  console.log('获取聊天室成员信息' + (!error?'成功':'失败'), error, obj
);
}
```

管理聊天室成员

包括以下接口

设置聊天室管理员
设置聊天室普通成员
设置聊天室黑名单
设置聊天室禁言名单
设置聊天室临时禁言

设置聊天室管理员

管理员可以设置聊天室普通成员, 设置聊天室黑名单, 设置聊天室禁言名单, 踢聊天室成员

account: 待设置的账号
isAdd: true表示添加, false表示移除
当有人被加为管理员时, 所有聊天室成员会收到类型为'addManager'的聊天室通知消息。
当有人被移除管理员时, 所有聊天室成员会收到类型为'removeManager'的聊天室通知消息。
custom: 扩展字段, 如果填了, 那么其它聊天室成员收到的聊天室通知消息的attach.custom的值为此字段
推荐使用JSON格式构建, 非JSON格式的话, Web端会正常接收, 但是会被其它端丢弃

```
chatroom.markChatroomManager({
  account: 'account',
  isAdd: true,
  done: markChatroomManagerDone
});
function markChatroomManagerDone(error, obj) {
  console.log('添加聊天室管理员' + (!error?'成功':'失败'), error, obj.
member);
}
```

设置聊天室普通成员

account: 待设置的账号
isAdd: 是否加为普通成员
当有人被加为普通成员时, 所有聊天室成员会收到类型为'addCommon'的聊天室通知消息。
当有人被移除普通成员时, 所有聊天室成员会收到类型为'removeCommon'的聊天室通知消息。
level: 等级
custom: 扩展字段, 如果填了, 那么其它聊天室成员收到的聊天室通知消息的attach.custom的值为此字段
推荐使用JSON格式构建, 非JSON格式的话, Web端会正常接收, 但是会被其它端丢弃

```
chatroom.markChatroomCommonMember({
  account: 'account',
  level: 1,
  done: markChatroomCommonMemberDone
});
function markChatroomCommonMemberDone(error) {
  console.log('设置聊天室普通成员' + (!error?'成功':'失败'), error);
}
```

设置聊天室黑名单

被加入黑名单的人将不能进入此聊天室

account: 待设置的账号

isAdd: true表示添加, false表示移除

当有人被加入黑名单时, 所有聊天室成员会收到类型为'blackMember'的聊天室通知消息。

当有人被移除黑名单时, 所有聊天室成员会收到类型为'blackMember'的聊天室通知消息。

custom: 扩展字段, 如果填了, 那么其它聊天室成员收到的聊天室通知消息的attach.custom的值为此字段

推荐使用JSON格式构建, 非JSON格式的话, Web端会正常接收, 但是会被其它端丢弃

```
chatroom.markChatroomBlacklist({
  account: 'account',
  isAdd: true,
  done: markChatroomBlacklistDone
});
function markChatroomBlacklistDone(error, obj) {
  console.log('添加聊天室黑名单' + (!error?'成功':'失败'), error, obj.
member);
}
```

设置聊天室禁言名单

被加入禁言名单的人将不能在该聊天室发送消息

account: 待设置的账号

isAdd: true表示添加, false表示移除

当有人被加入禁言名单时, 所有聊天室成员会收到类型为'gagMember'的聊天室

通知消息。
当有人被移除禁言名单时, 所有聊天室成员会收到类型为'ungagMember'的聊天室通知消息。
custom: 扩展字段, 如果填了, 那么其它聊天室成员收到的聊天室通知消息的attach.custom的值为此字段
推荐使用JSON格式构建, 非JSON格式的话, Web端会正常接收, 但是会被其它端丢弃

```
chatroom.markChatroomGaglist({
  account: 'account',
  isAdd: true,
  done: markChatroomGaglistDone
});
function markChatroomGaglistDone(error, obj) {
  console.log('添加聊天室禁言名单' + (!error?'成功':'失败'), error, obj.member);
}
```

设置聊天室临时禁言

当有人被设置聊天室临时禁言时，所有聊天室成员会收到类型为'addTempMute' or 'removeTempMute'的聊天室通知消息。
account: 帐号
duration: 禁言时长，单位秒，传0表示解除禁言
needNotify: 是否需要下发对应的通知消息
custom: 对应的通知消息的扩展字段

```
chatroom.updateChatroomMemberTempMute({
  account: 'account',
  duration: 60,
  needNotify: true,
  custom: 'biu',
  done: updateChatroomMemberTempMuteDone
})
function updateChatroomMemberTempMuteDone(error, obj) {
  console.log('设置聊天室临时禁言' + (!error?'成功':'失败'), error, obj);
}
```

踢聊天室成员

account: 待踢的账号
custom: 扩展字段, 如果填了, 那么其它聊天室成员收到的聊天室通知消息的attach.custom的值为此字段, 被踢的人收到的ondisconnect回调接收的参数的custom的值为此字段
推荐使用JSON格式构建, 非JSON格式的话, Web端会正常接收, 但是会被其它端丢弃
当有人被踢出聊天室时, 所有聊天室成员会收到类型为'kickMember'的聊天室通知消息。

```
chatroom.kickChatroomMember({
  account: 'account',
  done: kickChatroomMemberDone
});
function kickChatroomMemberDone(error, obj) {
  console.log('踢人' + (!error?'成功':'失败'), error, obj);
}
```

聊天室队列服务

聊天室队列中添加元素

方法名：queueOffer
参数：
elementKey: string类型，元素键名
elementValue: string类型，元素内容
transient: boolean类型，帐号从聊天室中离开/掉线后，其添加的元素是否被删除
done: 执行完成的回调函数，第一个参数为error
示例：

```
chatroom.queueOffer({
  elementKey: `account`,
  elementValue: JSON.stringify({
    nick: `nickname`,
    webrtc: 1
  }),
  transient: true,
  done (err, obj, content) {
    if (err) {
      console.error(err)
    }
  }
})
```

聊天室队列中删除元素

方法名：`queuePoll`，传空取第一个元素
参数：
`elementKey`: string类型，需要删除的元素键名
`done`: 执行完成的回调函数，第一个参数为`error`
示例：

```
chatroom.queuePoll({
  elementKey: `account`,
  done (err, obj, content) {
    if (err) {
      console.error(err)
    }
  }
})
```

聊天室队列中获取列表

方法名：`queueList`
参数：
`done`: 执行完成的回调函数，第一个参数为`error`，第三个参数为返回的结果
示例：

```
  chatroom.queueList({
    done (err, obj, content) {
    if (err) {
      console.error(err)
    }
    console.log(content)
    if (content && content.queueList) {
      queueCount = 0
      for (let i = 0; i < content.queueList.length; i++) {
        let queue = content.queueList[i]
        console.log(queue)
        queueCount++
      }
    }
    }
  })
```

聊天室队列中查看第一个元素

方法名：`peak`

参数：

`done`: 执行完成的回调函数，第一个参数为`error`，第三个参数为返回的结果

示例：

```
  chatroom.peak({
    done (err, obj, content) {
    if (err) {
      console.error(err)
    }
    console.log(content)
    }
  })
```

清除聊天室队列

方法名：`drop`

参数：

`done`: 执行完成的回调函数，第一个参数为`error`

示例：

```
chatroom.drop({
  done (err, obj, content) {
  if (err) {
  console.error(err)
  }
  }
})
```

聊天室队列通知

聊天室队列的变更会在聊天室通知消息中下发

```
function onChatroomMsgs (msgs) {
let self = this
msgs.forEach(msg => {
  if (msg.type === 'notification') {
  let attach = msg.attach
  let qc = attach.queueChange || {}
  switch (attach.type) {
  case 'updateQueue':
    if (qc.type === 'OFFER') {
    console.log(qc)
    } else if (qc.type === 'POLL') {
    console.log(qc)
    } else if (qc.type === 'DROP') {
    console.log(qc)
    } else if (qc.type === 'PARTCLEAR') {
    console.log(qc)
    }
    break
  case 'batchUpdateQueue':
    console.log(qc)
  }
  break
  }
})
}
```

用户资料托管

SDK 提供用户资料托管

初始化参数

这里的参数并不是所有的初始化参数, 请查阅初始化SDK, 以及其它章节的初始化参数

连接初始化参数
多端登录初始化参数
消息初始化参数
群组初始化参数
用户资料初始化参数
好友关系初始化参数
用户关系初始化参数
会话初始化参数
系统通知初始化参数
同步完成
完整的初始化代码

示例代码

```
var nim = NIM.getInstance({
    onmyinfo: onMyInfo,
    onupdatemyinfo: onUpdateMyInfo,
    onusers: onUsers,
    onupdateuser: onUpdateUser
});
function onMyInfo(user) {
    console.log('收到我的名片', user);
    data.myInfo = user;
    updateMyInfoUI();
}
function onUpdateMyInfo(user) {
    console.log('我的名片更新了', user);
    data.myInfo = NIM.util.merge(data.myInfo, user);
    updateMyInfoUI();
}
function updateMyInfoUI() {
    // 刷新界面
}
function onUsers(users) {
    console.log('收到用户资料列表', users);
    data.users = nim.mergeUsers(data.users, users);
}
function onUpdateUser(user) {
    console.log('用户资料更新了', user);
    data.users = nim.mergeUsers(data.users, user);
}
```

参数解释

`onmyinfo`: 同步登录用户资料的回调, 会传入用户资料
`onupdatemyinfo`: 当前登录用户在其它端修改自己的个人名片之后的回调, 会传入用户资料
`onusers`: 同步好友用户资料的回调, 会传入用户资料数组
此回调是增量回调, 可以调用nim.mergeUsers来合并数据
`onupdateuser`: 用户资料更新后的回调, 会传入用户资料，请参考用户资料更新时机

用户资料对象

用户资料对象有以下字段：
- `account`: 账号

- `nick`: 昵称
- `avatar`: 头像
- `sign`: 签名
- `gender`: 性别
- `email`: 邮箱
- `birth`: 生日
- `tel`: 电话号码
- `custom`: 扩展字段
- 推荐使用`JSON`格式构建, 非`JSON`格式的话, Web端会正常接收, 但是会被其它端丢弃
- `createTime`: 创建时间
- `updateTime`: 更新时间

性别

`'unknown'` (未知)
`'male'` (男)
`'female'` (女)

更新我的资料

```
nim.updateMyInfo({
    nick: 'newNick',
    avatar: 'http://newAvatar',
    sign: 'newSign',
    gender: 'male',
    email: 'new@email.com',
    birth: '1900-01-01',
    tel: '13523578129',
    custom: '{type: "newCustom", value: "new"}',
    done: updateMyInfoDone
});
function updateMyInfoDone(error, user) {
    console.log('更新我的名片' + (!error?'成功':'失败'));
    console.log(error);
    console.log(user);
    if (!error) {
        onUpdateMyInfo(user);
    }
}
```

用户资料更新时机

用户资料除自己之外，不保证其他用户资料实时更新，其他用户资料更新时机为
收到此用户发来的消息
每次同步会同步好友对应的用户资料
如果想手动刷新用户资料，请参考获取用户资料和获取用户资料数组

获取用户资料

请参考用户资料更新时机
可以传入参数`sync=true`来强制从服务器获取最新的数据

```
nim.getUser({
    account: 'account',
    done: getUserDone
});
function getUserDone(error, user) {
    console.log(error);
    console.log(user);
    console.log('获取用户资料' + (!error?'成功':'失败'));
    if (!error) {
        onUsers(user);
    }
}
```

获取用户资料数组

请参考用户资料更新时机
可以传入参数`sync=true`来强制从服务器获取最新的数据
每次最多 150 个

```
nim.getUsers({
    accounts: ['account1', 'account2'],
    done: getUsersDone
});
function getUsersDone(error, users) {
    console.log(error);
    console.log(users);
    console.log('获取用户资料数组' + (!error?'成功':'失败'));
    if (!error) {
        onUsers(users);
    }
}
```

好友关系托管

SDK 提供好友关系托管

初始化参数

这里的参数并不是所有的初始化参数, 请查阅初始化SDK, 以及其它章节的初始化参数

连接初始化参数
多端登录初始化参数
消息初始化参数
群组初始化参数
用户资料初始化参数
好友关系初始化参数
用户关系初始化参数
会话初始化参数
系统通知初始化参数
同步完成
完整的初始化代码

请参考处理系统通知里面的跟好友相关的逻辑

示例代码

```
var nim = NIM.getInstance({
    onfriends: onFriends,
    onsyncfriendaction: onSyncFriendAction
});
function onFriends(friends) {
    console.log('收到好友列表', friends);
    data.friends = nim.mergeFriends(data.friends, friends);
    data.friends = nim.cutFriends(data.friends, friends.invalid);
    refreshFriendsUI();
}
function onSyncFriendAction(obj) {
    console.log(obj);
    switch (obj.type) {
    case 'addFriend':
        console.log('你在其它端直接加了一个好友' + obj.account + ', 附言' + obj.ps);
        onAddFriend(obj.friend);
        break;
    case 'applyFriend':
        console.log('你在其它端申请加了一个好友' + obj.account + ', 附言' + obj.ps);
        break;
    case 'passFriendApply':
        console.log('你在其它端通过了一个好友申请' + obj.account + ',
```

```
附言' + obj.ps);
            onAddFriend(obj.friend);
            break;
        case 'rejectFriendApply':
            console.log('你在其它端拒绝了一个好友申请' + obj.account + ',
附言' + obj.ps);
            break;
        case 'deleteFriend':
            console.log('你在其它端删了一个好友' + obj.account);
            onDeleteFriend(obj.account);
            break;
        case 'updateFriend':
            console.log('你在其它端更新了一个好友', obj.friend);
            onUpdateFriend(obj.friend);
            break;
        }
}
function onAddFriend(friend) {
    data.friends = nim.mergeFriends(data.friends, friend);
    refreshFriendsUI();
}
function onDeleteFriend(account) {
    data.friends = nim.cutFriendsByAccounts(data.friends, account);
    refreshFriendsUI();
}
function onUpdateFriend(friend) {
    data.friends = nim.mergeFriends(data.friends, friend);
    refreshFriendsUI();
}
function refreshFriendsUI() {
    // 刷新界面
}
```

参数解释

`onfriends`, 同步好友列表的回调, 会传入好友列表`friends`

`friends`的属性invalid包含被删除的好友列表

此回调是增量回调, 可以调用nim.mergeFriends和nim.cutFriends来合并数据

`onsyncfriendaction`, 当前登录用户在其它端进行好友相关的操作后的回调

操作包括

直接加为好友

申请加为好友

通过好友申请

拒绝好友申请

删除好友

更新好友

此回调会收到一个参数obj, 它有一个字段type的值为操作的类型, 具体类型如下：

'addFriend' (直接加为好友), 此时obj的字段如下:

account的值为被直接加为好友的账号

friend为被直接加为好友的好友对象

ps为附言

'applyFriend' (申请加为好友), 此时obj的字段如下:

account的值为被申请加为好友的账号

ps为附言

'passFriendApply' (通过好友申请), 此时obj的字段如下:

account的值为被通过好友申请的账号

friend为被通过好友申请的好友对象

ps为附言

'rejectFriendApply' (拒绝好友申请), 此时obj的字段如下:

account的值为被拒绝好友申请的账号

ps为附言

'deleteFriend' (删除好友), 此时obj的字段如下:

account的值为被删除好友的账号

'updateFriend' (更新好友), 此时obj的字段如下:

friend的值为被更新的好友对象

可以调用nim.mergeFriends和nim.cutFriendsByAccounts来合并数据

好友对象

好友对象有以下字段:

- `account`: 账号
- `alias`: 昵称
- `custom`: 扩展字段, 开发者可以自行扩展, 建议封装成JSON格式字符串
- `createTime`: 成为好友的时间
- `updateTime`: 更新时间

直接加为好友

直接加某个用户为好友后, 对方不需要确认, 直接成为当前登录用户的好友
ps: 附言, 选填, 开发者也可以使用JSON格式的字符串来扩展此内容
对方会收到一条类型为'addFriend'的系统通知, 此类系统通知的from字段的值为申请方的帐号, to字段的值为接收方的账号。

```
nim.addFriend({
    account: 'account',
    ps: 'ps',
    done: addFriendDone
});
function addFriendDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('直接加为好友' + (!error?'成功':'失败'));
    if (!error) {
        onAddFriend(obj.friend);
    }
}
```

申请加为好友

申请加某个用户为好友后, 对方会收到一条类型为'applyFriend'的系统通知, 此类系统通知的from字段的值为申请方的帐号, to字段的值为接收方的账号, 用户在收到好友申请后, 可以选择通过或者拒绝好友申请。
如果通过好友申请, 那么申请方会收到一条类型为'passFriendApply'的系统通知, 此类系统通知的from字段的值为通过方的帐号, to字段的值为申请方的账号。
如果拒绝好友申请, 那么申请方会收到一条类型为'rejectFriendApply'的系统通知, 此类系统通知的from字段的值为拒绝方的帐号, to字段的值为申请方的账号。
ps: 附言, 选填, 开发者也可以使用JSON格式的字符串来扩展此内容

```
nim.applyFriend({
    account: 'account',
    ps: 'ps',
    done: applyFriendDone
});
function applyFriendDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('申请加为好友' + (!error?'成功':'失败'));
}
```

通过好友申请

申请加某个用户为好友后, 对方会收到一条类型为'applyFriend'的系统通知, 此类系统通知的from字段的值为申请方的帐号, to字段的值为接收方的账号, 用户在收到好友申请后, 可以选择通过或者拒绝好友申请。
如果通过好友申请, 那么申请方会收到一条类型为'passFriendApply'的系统通知, 此类系统通知的from字段的值为通过方的帐号, to字段的值为申请方的账号。
如果拒绝好友申请, 那么申请方会收到一条类型为'rejectFriendApply'的系统通知, 此类系统通知的from字段的值为拒绝方的帐号, to字段的值为申请方的账号。
ps: 附言, 选填, 开发者也可以使用JSON格式的字符串来扩展此内容

```
// 假设 sysMsg 是通过回调 `onsysmsg` 收到的系统通知
nim.passFriendApply({
    idServer: sysMsg.idServer,
    account: 'account',
    ps: 'ps',
    done: passFriendApplyDone
});
function passFriendApplyDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('通过好友申请' + (!error?'成功':'失败'));
    if (!error) {
        onAddFriend(obj.friend);
    }
}
```

拒绝好友申请

申请加某个用户为好友后, 对方会收到一条类型为'applyFriend'的系统通知, 此类系统通知的from字段的值为申请方的帐号, to字段的值为接收方的账号, 用户在收到好友申请后, 可以选择通过或者拒绝好友申请。
如果通过好友申请, 那么申请方会收到一条类型为'passFriendApply'的系统通知, 此类系统通知的from字段的值为通过方的帐号, to字段的值为申请方的账号。
如果拒绝好友申请, 那么申请方会收到一条类型为'rejectFriendApply'的系统通知, 此类系统通知的from字段的值为拒绝方的帐号, to字段的值为申请方的账号。
ps: 附言, 选填, 开发者也可以使用JSON格式的字符串来扩展此内容

```
// 假设 sysMsg 是通过回调 `onsysmsg` 收到的系统通知
nim.rejectFriendApply({
    idServer: sysMsg.idServer,
    account: 'account',
    ps: 'ps',
    done: rejectFriendApplyDone
});
function rejectFriendApplyDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('拒绝好友申请' + (!error?'成功':'失败'));
}
```

删除好友

删除好友后, 被删除的人会收到一条类型为'deleteFriend'的系统通知, 此类系统通知的from字段的值为删除方的帐号, to字段的值为被删除方的账号。

```
nim.deleteFriend({
    account: 'account',
    done: deleteFriendDone
});
function deleteFriendDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('删除好友' + (!error?'成功':'失败'));
    if (!error) {
        onDeleteFriend(obj.account);
    }
}
```

更新好友

开发者可以用此接口来更新好友的备注
开发者也可以使用JSON格式的扩展字段来进行扩展

```
nim.updateFriend({
    account: 'account',
    alias: 'alias',
    custom: 'custom',
    done: updateFriendDone
});
function updateFriendDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('更新好友' + (!error?'成功':'失败'));
    if (!error) {
        onUpdateFriend(obj);
    }
}
```

获取好友列表

如果开发者在初始化SDK的时候设置了`syncFriends`为`false`, 那么就收不到`onfriends`回调, 可以调用此接口来获取好友列表。

```
nim.getFriends({
    done: getFriendsDone
});
function getFriendsDone(error, friends) {
    console.log('获取好友列表' + (!error?'成功':'失败'), error, friend
s);
    if (!error) {
        onFriends(friends);
    }
}
```

用户关系托管

SDK 提供了用户关系托管, 包括黑名单和静音列表

黑名单

如果一个用户被加入了黑名单, 那么就不再会收到此用户发送的消息
如果一个用户被从黑名单移除, 那么会重新收到此用户发送的消息

静音列表

SDK只负责维护静音列表, 具体根据静音列表要进行的操作由开发者决定

初始化参数

这里的参数并不是所有的初始化参数, 请查阅初始化SDK, 以及其它章节的初始化参数
连接初始化参数
多端登录初始化参数

示例代码

```
var nim = NIM.getInstance({
    onblacklist: onBlacklist,
    onsyncmarkinblacklist: onMarkInBlacklist,
    onmutelist: onMutelist,
    onsyncmarkinmutelist: onMarkInMutelist
});
function onBlacklist(blacklist) {
    console.log('收到黑名单', blacklist);
    data.blacklist = nim.mergeRelations(data.blacklist, blacklist);
    data.blacklist = nim.cutRelations(data.blacklist, blacklist.inv
alid);
    refreshBlacklistUI();
}
function onMarkInBlacklist(obj) {
    console.log(obj);
    console.log(obj.account + '被你' + (obj.isAdd ? '加入' : '移除')
+ '黑名单');
    if (obj.isAdd) {
        addToBlacklist(obj);
    } else {
        removeFromBlacklist(obj);
    }
}
function addToBlacklist(obj) {
    data.blacklist = nim.mergeRelations(data.blacklist, obj.record)
;
    refreshBlacklistUI();
}
function removeFromBlacklist(obj) {
    data.blacklist = nim.cutRelations(data.blacklist, obj.record);
    refreshBlacklistUI();
}
function refreshBlacklistUI() {
    // 刷新界面
}
```

```
function onMutelist(mutelist) {
    console.log('收到静音列表', mutelist);
    data.mutelist = nim.mergeRelations(data.mutelist, mutelist);
    data.mutelist = nim.cutRelations(data.mutelist, mutelist.invali
d);
    refreshMutelistUI();
}
function onMarkInMutelist(obj) {
    console.log(obj);
    console.log(obj.account + '被你' + (obj.isAdd ? '加入' : '移除')
+ '静音列表');
    if (obj.isAdd) {
        addToMutelist(obj);
    } else {
        removeFromMutelist(obj);
    }
}
function addToMutelist(obj) {
    data.mutelist = nim.mergeRelations(data.mutelist, obj.record);
    refreshMutelistUI();
}
function removeFromMutelist(obj) {
    data.mutelist = nim.cutRelations(data.mutelist, obj.record);
    refreshMutelistUI();
}
function refreshMutelistUI() {
    // 刷新界面
}
```

参数解释

`onblacklist`: 同步黑名单的回调, 会传入黑名单列表`blacklist`

`blacklist`的属性`invalid`包含被删除的黑名单列表

此回调是增量回调, 可以调用nim.mergeRelations和nim.cutRelations来合并数据

`onsyncmarkinblacklist`: 当前登录用户在其它端加入黑名单/从黑名单移除后的回调, 会传入一个参数, 包含两个字段

`account`: 要加入黑名单/从黑名单移除的账号

`isAdd`: `true`表示加入黑名单, `false`表示从黑名单移除

`reocrd`, 拼装好的对象

`onmutelist`: 同步静音列表的回调, 会传入静音列表`mutelist`

`mutelist`的属性`invalid`包含被删除的静音列表

此回调是增量回调, 可以调用nim.mergeRelations和nim.cutRelations来合并数据

onsyncmarkinmutelist: 当前登录用户在其它端加入静音列表/从静音列表移除后的回调, 会传入一个参数, 包含两个字段
account: 要加入静音列表/从静音列表移除的账号
isAdd: true表示加入静音列表, false表示从静音列表移除
reocrd, 拼装好的对象

加入黑名单/从黑名单移除

此接口可以完成以下两个功能, 通过参数isAdd来决定实际的功能
isAdd为true时, 会将account加入黑名单
如果一个用户被加入了黑名单, 那么就不再会收到此用户发送的消息
isAdd为false时, 会将account从黑名单移除
如果一个用户被从黑名单移除, 那么会重新收到此用户发送的消息
每个功能SDK都提供了相应的独立接口

```
nim.markInBlacklist({
    account: 'account',
    // `true`表示加入黑名单, `false`表示从黑名单移除
    isAdd: true,
    done: markInBlacklistDone
});
function markInBlacklistDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('将' + obj.account + (isAdd ? '加入黑名单' : '从黑名单移除') + (!error?'成功':'失败'));
    if (!error) {
        onMarkInBlacklist(obj);
    }
}
```

加入黑名单

如果一个用户被加入了黑名单, 那么就不再会收到此用户发送的消息
SDK内部调用nim.markInBlacklist来完成实际工作

```
nim.addToBlacklist({
    account: 'account',
    done: addToBlacklistDone
});
function addToBlacklistDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('加入黑名单' + (!error?'成功':'失败'));
    if (!error) {
        addToBlacklist(obj);
    }
}
```

从黑名单移除

如果一个用户被从黑名单移除, 那么会重新收到此用户发送的消息
SDK内部调用nim.markInBlacklist来完成实际工作

```
nim.removeFromBlacklist({
    account: 'account',
    done: removeFromBlacklistDone
});
function removeFromBlacklistDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('从黑名单移除' + (!error?'成功':'失败'));
    if (!error) {
        removeFromBlacklist(obj);
    }
}
```

加入静音列表/从静音列表移除

此接口可以完成以下两个功能, 通过参数`isAdd`来决定实际的功能
`isAdd`为`true`时, 会将`account`加入静音列表
`isAdd`为`false`时, 会将`account`从静音列表移除
每个功能SDK都提供了相应的独立接口

```
nim.markInMutelist({
    account: 'account',
    // `true`表示加入静音列表, `false`表示从静音列表移除
    isAdd: 'true',
    done: markInMutelistDone
});
function markInMutelistDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('将' + obj.account + (isAdd ? '加入静音列表' : '从静音列表移除') + (!error?'成功':'失败'));
    if (!error) {
        onMarkInMutelist(obj);
    }
}
```

加入静音列表

SDK只负责维护静音列表, 具体要根据静音列表进行的操作由开发者决定
SDK内部调用nim.markInMutelist来完成实际工作

```
nim.addToMutelist({
    account: 'account',
    done: addToMutelistDone
});
function addToMutelistDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('加入静音列表' + (!error?'成功':'失败'));
    if (!error) {
        addToMutelist(obj);
    }
}
```

从静音列表移除

SDK只负责维护静音列表, 具体要根据静音列表进行的操作由开发者决定
SDK内部调用nim.markInMutelist来完成实际工作

```
nim.removeFromMutelist({
    account: 'account',
    done: removeFromMutelistDone
});
function removeFromMutelistDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('从静音列表移除' + (!error?'成功':'失败'));
    if (!error) {
        removeFromMutelist(obj);
    }
}
```

获取黑名单和静音列表

如果开发者在初始化SDK的时候设置了`syncRelations`为`false`, 那么就收不到`onblacklist`和`onmutelist`回调, 可以调用此接口来获取黑名单和静音列表。

```
nim.getRelations({
    done: getRelationsDone
});
function getRelationsDone(error, obj) {
    console.log('获取静音列表' + (!error?'成功':'失败'), error, obj);
    if (!error) {
        onBlacklist(obj.blacklist);
        onMutelist(obj.mutelist);
    }
}
```

最近会话

生成规则

SDK 会根据漫游消息和离线消息来生成初始会话列表, 在收到消息和发送消息之后 SDK 会更新会话列表

初始化参数

这里的参数并不是所有的初始化参数, 请查阅初始化SDK, 以及其它章节的初始化参数

示例代码

```
var nim = NIM.getInstance({
    onsessions: onSessions,
    onupdatesession: onUpdateSession
});
function onSessions(sessions) {
    console.log('收到会话列表', sessions);
    data.sessions = nim.mergeSessions(data.sessions, sessions);
    updateSessionsUI();
}
function onUpdateSession(session) {
    console.log('会话更新了', session);
    data.sessions = nim.mergeSessions(data.sessions, session);
    updateSessionsUI();
}
function updateSessionsUI() {
    // 刷新界面
}
```

参数解释

`syncSessionUnread`, 是否同步会话的未读数, 默认不同步
如果选择同步

那么在一个端读过的会话在其它端也会被标记为已读
在调用设置当前会话的时候 SDK 会自动同步一次未读数, 此后如果收到当前会话的消息, 需要手动调用重置会话未读数来同步未读数
`onsessions`, 同步最近会话列表回调, 会传入会话列表, 按时间正序排列, 即最近聊过天的放在列表的最后面
此回调是增量回调, 可以调用nim.mergeSessions来合并数据
`onupdatesession`, 更新会话的回调, 会传入会话对象, 以下情况会收到此回调
收到消息
发送消息
设置当前会话
重置会话未读数

会话对象

会话对象有以下字段:

`id`: 会话ID
`scene`: 场景
`to`: 聊天对象, 账号或群ID
`updateTime`: 会话更新的时间
`unread`: 未读数
`lastMsg`: 此会话的最后一条消息
`msgReceiptTime`: 消息已读回执时间戳, 如果有此字段, 说明此时间戳之前的所有消息对方均已读
目前仅对`'p2p'`会话起作用
此字段不一定有, 只有对方发送过已读回执之后才会有
调用接口发送消息已读回执来发送消息已读回执
调用接口查询消息是否被对方读过了来查询消息是否被对方读过了
`localCustom`: 本地自定义扩展字段
在支持数据库时可以调用更新本地会话来更新此字段, 此字段只会被更新到本地数据库, 不会被更新到服务器上

未读数

SDK 会自动管理会话的未读数, 会话对象的`unread`的值为会话的未读数, 如果开发者发现会话的未读数大于收到的离线消息数, 那么需要从本地拉取未读取的消息

会话未读数的初始化在不同的配置环境下，会有不同的计算规则：
- 开启数据库： db = true
- 开启同步会话未读数： syncSessionUnread = true
- 此时会话未读数通过服务器下推的Ack或本地存储的Ack时间戳，与本地数据库中对应会话的本地历史记录做比较，晚于该Ack且不是自己发的消息的数量，为未读数
- 参见会话初始化参数
- 未开启会话未读数： syncSessionUnread = false
- 此时会话未读数通过从本地数据库上次所记录的未读数中取得，如果有离线消息且消息属性标记为isUnreadable，则会在原来的未读数上增加计数
- 不开启数据库： db = false
- 不开启自动标记消息已读： autoMarkRead = false
- 此时服务器下推的所有离线消息算未读，漫游消息算已读
- 开启自动标记消息已读： autoMarkRead = true
- 此时每次收到离线消息，均会告知服务器该消息已读，下一次登录，服务器就不会下推离线消息，而将这些消息标记为漫游消息。没有离线消息，未读数在表现上均为0
- 参见标记消息为已收到

设置当前会话

如果是已经存在的会话记录, 会将此会话未读数置为 0, 开发者会收到`onupdatesession`回调
之后此会话在收到消息之后不会更新未读数

```
nim.setCurrSession('sessionId')
```

重置会话未读数

如果是已经存在的会话记录, 会将此会话未读数置为 0, 那么会收到`onupdatesession`回调
之后此会话在收到消息之后依然会更新未读数

```
nim.resetSessionUnread('sessionId')
```

重置所有会话未读数(内存中的)

如果是内存中已经存在的会话记录, 会将该内存中的所有会话记录的未读数均清零, 那么会多次收到`onupdatesession`回调
之后此会话在收到消息之后依然会更新未读数

```
nim.resetAllSessionUnread()
```

重置当前回话

重置当前会话后, 所有会话在收到消息之后会更新未读数

```
nim.resetCurrSession();
```

获取本地会话列表

在支持数据库时, SDK 会将会话存储于数据库中, 并且在初始化时通过回调`onsessions`将会话列表返回给开发者, 不过此列表最多 100 条记录
如果想获取更多会话记录, 可以调用此方法来获取更多本地会话记录
`lastSessionId`为上次查询的最后一条会话的`id`, 第一次不填
`limit`为本次查询的会话数量限制, 最多 100 条, 默认 100 条
默认从最近的会话开始往前查找本地会话, 可以传参数`reverse=true`来从第一条会话开始往后查找本地会话

```
nim.getLocalSessions({
    lastSessionId: lastSessionId,
    limit: 100,
    done: getLocalSessionsDone
});
function getLocalSessionsDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('获取本地会话列表' + (!error?'成功':'失败'));
    if (!error) {
        onSessions(obj.sessions);
    }
}
```

插入一条本地会话记录

开发者可以插入一条本地会话记录, 在支持数据库时, SDK 会将此会话存储于本地数据库, 反之, 数据仅存于内存里面
SDK 会设置一个比当前所有会话更新时间大的一个时间为此会话的更新时间, 或者开发者可以传入参数`updateTime`来指定更新时间
在回调里面, 开发者需要保存生成的会话

```
nim.insertLocalSession({
    scene: 'p2p',
    to: 'account',
    done: insertLocalSessionDone
});
function insertLocalSessionDone(error, obj) {
    console.log('插入本地会话记录' + (!error?'成功':'失败'), error, ob
j);
    if (!error) {
        onSessions(obj.session);
    }
}
```

更新本地会话

更新 `id` 对应的本地会话
如果不支持数据库, 算成功
如果对应的会话不存在, 算成功, 返回 null

这些字段只会被更新到本地数据库, 不会被更新到服务器上
目前只允许更新 `localCustom`

```
nim.updateLocalSession({
    id: 'p2p-account',
    localCustom: '{"key","value"}',
    done: updateLocalSessionDone
});
function updateLocalSessionDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('更新本地会话' + (!error?'成功':'失败'));
}
```

删除本地会话

在支持数据库时, 删了本地会话之后, 下次同步就同步不到对应的会话
如果不支持数据库, 算成功
如果对应的会话不存在, 算成功
参数 id 为会话 id 或 id 数组

```
nim.deleteLocalSession({
    id: 'p2p-account',
    done: deleteLocalSessionDone
});
function deleteLocalSessionDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('删除本地会话' + (!error?'成功':'失败'));
}
```

删除服务器上的会话

删了服务器上的会话之后, 在不支持数据库时, 下次同步就同步不到对应的会话以及会话对应的漫游消息; 此外, 在新设备上也同步不到对应的会话以及会话对应的漫游消息
`scene`请参考消息场景
`to` 为对方账号或群ID

```
nim.deleteSession({
    scene: 'p2p',
    to: 'account',
    done: deleteSessionDone
});
function deleteSessionDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('删除服务器上的会话' + (!error?'成功':'失败'));
}
```

批量删除服务器上的会话

删了服务器上的会话之后, 在不支持数据库时, 下次同步就同步不到对应的会话以及会话对应的漫游消息; 此外, 在新设备上也同步不到对应的会话以及会话对应的漫游消息

`scene`请参考消息场景

`to` 为对方账号或群ID

```
nim.deleteSessions({
    sessions: {[
        scene: 'p2p',
        to: 'account'
    ], [
        scene: 'p2p',
        to: 'account1'
    ]},
    done: deleteSessionsDone
});
function deleteSessionsDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('批量删除会话' + (!error?'成功':'失败'));
}
```

系统通知

初始化参数

这里的参数并不是所有的初始化参数, 请查阅初始化SDK, 以及其它章节的初始化参数

连接初始化参数
多端登录初始化参数
消息初始化参数
群组初始化参数
用户资料初始化参数
好友关系初始化参数
用户关系初始化参数
会话初始化参数
系统通知初始化参数
同步完成
完整的初始化代码

示例代码

```
var nim = NIM.getInstance({
    onofflinesysmsgs: onOfflineSysMsgs,
    onsysmsg: onSysMsg,
    onupdatesysmsg: onUpdateSysMsg,
    onsysmsgunread: onSysMsgUnread,
    onupdatesysmsgunread: onUpdateSysMsgUnread,
    onofflinecustomsysmsgs: onOfflineCustomSysMsgs,
    oncustomsysmsg: onCustomSysMsg,
    onbroadcastmsg: onBroadcastMsg,
    onbroadcastmsgs: onBroadcastMsgs,
});
function onOfflineSysMsgs(sysMsgs) {
    console.log('收到离线系统通知', sysMsgs);
    pushSysMsgs(sysMsgs);
}
function onSysMsg(sysMsg) {
    console.log('收到系统通知', sysMsg)
    pushSysMsgs(sysMsg);
}
function onUpdateSysMsg(sysMsg) {
    pushSysMsgs(sysMsg);
}
function pushSysMsgs(sysMsgs) {
    data.sysMsgs = nim.mergeSysMsgs(data.sysMsgs, sysMsgs);
    refreshSysMsgsUI();
}
function onSysMsgUnread(obj) {
    console.log('收到系统通知未读数', obj);
    data.sysMsgUnread = obj;
    refreshSysMsgsUI();
}
function onUpdateSysMsgUnread(obj) {
    console.log('系统通知未读数更新了', obj);
    data.sysMsgUnread = obj;
    refreshSysMsgsUI();
}
function refreshSysMsgsUI() {
    // 刷新界面
}
function onOfflineCustomSysMsgs(sysMsgs) {
    console.log('收到离线自定义系统通知', sysMsgs);
}
function onCustomSysMsg(sysMsg) {
    console.log('收到自定义系统通知', sysMsg);
}
function onBroadcastMsg(msg) {
    console.log('收到广播消息', msg);
}
function onBroadcastMsgs(msgs) {
    console.log('收到广播消息', msgs);
}
```

参数解释

`onofflinesysmsgs`, 同步离线系统通知的回调, 会传入系统通知数组
在支持数据库时并且启用了多 tab 同时登录, 那么如果多个 tab 页同时断线重连之后, 只会有一个 tab 页负责存储离线系统通知, 即只会有一个 tab 页会收到 `onofflinesysmsgs`回调, 其它 tab 页在同步完成之后, 需要调用获取本地系统通知来从本地缓存中拉取系统通知
`onsysmsg`, 收到系统通知的回调, 会传入系统通知
收到系统通知后需要调用标记系统通知为已读状态来将系统通知标记为已读状态
`onupdatesysmsg`, 更新系统通知后的回调, 会传入{@link SystemMessage|系统通知}
以下情况会收到此回调
通过好友申请
拒绝好友申请
接受入群邀请
拒绝入群邀请
通过入群申请
拒绝入群申请
这些操作的发起方会收到此回调, 接收被更新的系统通知, 根据操作的类型系统通知会被更新为下面两种状态
`'passed'`: 已通过
`'rejected'`: 已拒绝
`onsysmsgunread`: 收到系统通知未读数的回调
SDK 会管理内建系统通知的未读数, 此回调接收的对象包括以下字段
`friend`: 所有跟好友相关的系统通知的未读数
`addFriend`: 直接加为好友的未读数
`applyFriend`: 申请加为好友的未读数
`passFriendApply`: 通过好友申请的未读数
`rejectFriendApply`: 拒绝好友申请的未读数
`deleteFriend`: 删除好友的未读数
`team`: 所有跟群相关的系统通知的未读数
`teamInvite`: 入群邀请的未读数
`rejectTeamInvite`: 接受入群邀请的未读数

`applyTeam`: 入群申请的未读数
`rejectTeamApply`: 拒绝入群申请的未读数
`deleteMsg`: 撤回消息的未读数
`onupdatesysmsgunread`: 更新系统通知未读数的回调
`onofflinecustomsysmsgs`, 同步离线自定义系统通知的回调, 会传入系统通知数组
`oncustomsysmsg`, 收到自定义系统通知的回调, 会传入系统通知
`onbroadcastmsg`, 收到广播消息的回调，一条
`onbroadcastmsgs`, 收到广播消息的回调，多条

系统通知分为两种
- 内建系统通知
- 目前所有的内建系统通知都是与高级群相关的通知, 某些群操作后相关的群成员会收到相应的系统通知。
- 内建系统通知与群通知消息的区别是系统通知是发给单人的通知, 群通知消息是发给所有群成员的消息。
- 自定义系统通知

系统通知对象

系统通知对象有以下字段
- `time`: 时间戳
- `type`: 系统通知类型, 自定义系统通知无此字段
- `from`: 系统通知的来源, 账号或者群ID
- `to`: 系统通知的目标, 账号或者群ID
- `idServer`: 内建系统通知的 idServer
- `read`: 内建系统通知是否已读
- `category`: 内建系统通知种类
- `state`: 内建系统通知状态
- `error`: 内建系统通知的状态为 `'error'` 时, 此字段包含错误的信息
- `localCustom`: 内建系统通知的本地自定义扩展字段
- 在支持数据库时可以调用更新本地会话来更新此字段, 此字段只会被更新到本

地数据库, 不会被更新到服务器上
- `ps`: 内建系统通知的附言
- `attach`: 内建系统通知的附加信息, 参考系统通知类型来查看不同类型的系统通知对应的附加信息
- `scene`: 自定义系系统通知的场景, 参考消息场景
- `content`: 自定义系统通知的内容
- `isPushable`: 是否需要推送
- `apnsText`: 自定义系统通知的apns推送文案, 仅对接收方为iOS设备有效
- `pushPayload`: 自定义系统通知的推送属性
- 推荐使用`JSON`格式构建, 非`JSON`格式的话, Web端会正常接收, 但是会被其它端丢弃
- `needPushNick`: 是否需要推送昵称
- `sendToOnlineUsersOnly`: 自定义系统通知是否只发送给在线用户。
- `true`时只发送给在线用户, 适合发送即时通知, 比如正在输入。
- `false`时假如目标用户或群不在线, 会在其上线后推送过去。
- 该参数只对点对点自定义系统通知有效, 对群自定义系统通知无效, 群自定义系统通知只会发给在线的群成员, 不会存离线。
- `cc`: 自定义系统通知是否抄送

系统通知类型

系统通知对象有一个字段`type`来标明系统通知的类型, 自定义系统通知无此字段, 具体类型如下
- `'teamInvite'` (入群邀请)
- 高级群的群主和管理员在邀请成员加入群（通过操作创建群或拉人入群）之后, 被邀请的人会收到一条类型为`'teamInvite'`的系统通知, 此类系统通知的`from`字段的值为邀请方的帐号, `to`字段的值为对应的群ID, 此类系统通知的`attach`有一个字段`team`的值为被邀请进入的群, 被邀请的人可以选择接受邀请或者拒绝邀请。
- 如果接受邀请, 那么该群的所有群成员会收到一条类型为`'acceptTeamInvite'`的群通知消息, 此类群通知消息的`from`字段的值为接受入群邀请的人的帐号, `to`字段的值为对应的群ID, `attach`有一个字段

`team`的值为对应的群对象, `attach`有一个字段`members`的值为接收入群邀请的群成员列表。
- 如果拒绝邀请, 那么邀请你的人会收到一条类型为`'rejectTeamInvite'`的系统通知, 此类系统通知的`from`字段的值为拒绝入群邀请的用户的帐号, `to`字段的值为对应的群ID。
- `'rejectTeamInvite'` (拒绝入群邀请)
- 见`'teamInvite'`
- `'applyTeam'` (入群申请)
- 用户可以申请加入高级群, 目标群的群主和管理员会收到一条类型为`'applyTeam'`的系统通知, 此类系统通知的`from`字段的值为申请方的帐号, `to`字段的值为对应的群ID, 高级群的群主和管理员在收到入群申请后, 可以选择通过或者拒绝入群申请。
- 如果通过申请, 那么该群的所有群成员会收到一条类型为`'passTeamApply'`的群通知消息, 此类群通知消息的`from`字段的值为通过入群申请的人的帐号, `to`字段的值为对应的群ID, `attach`有一个字段`team`的值为对应的群对象, `attach`有一个字段`account`的值为申请方的帐号, `attach`有一个字段`members`的值为被通过申请的群成员列表。
- 如果拒绝申请, 那么申请人会收到一条类型为`'rejectTeamApply'`的系统通知, 此类系统通知的`from`字段的值为拒绝方的帐号, `to`字段的值为对应的群ID, `attach`有一个字段`team`的值为对应的群。
- `'rejectTeamApply'` (拒绝入群申请)
- 见`'applyTeam'`
- `'addFriend'`
- 直接加某个用户为好友后, 对方不需要确认, 直接成为当前登录用户的好友
- 对方会收到一条类型为`'addFriend'`的系统通知, 此类系统通知的`from`字段的值为申请方的帐号, `to`字段的值为接收方的账号。
- `'applyFriend'`
- 申请加某个用户为好友后, 对方会收到一条类型为`'applyFriend'`的系统通知, 此类系统通知的`from`字段的值为申请方的帐号, `to`字段的值为接收方的账号, 用户在收到好友申请后, 可以选择通过或者拒绝好友申请。
- 如果通过好友申请, 那么申请方会收到一条类型为`'passFriendApply'`的系统通知, 此类系统通知的`from`字段的值为通过方的帐号, `to`字段的值为申请方的账号。

- 如果拒绝好友申请, 那么申请方会收到一条类型为`'rejectFriendApply'`的系统通知, 此类系统通知的`from`字段的值为拒绝方的帐号, `to`字段的值为申请方的账号。
- `'passFriendApply'`
- 见`'applyFriend'`
- `'rejectFriendApply'`
- 见`'applyFriend'`
- `'deleteFriend'`
- 删除好友后, 被删除的人会收到一条类型为`'deleteFriend'`的系统通知, 此类系统通知的`from`字段的值为删除方的帐号, `to`字段的值为被删除方的账号。
- `'deleteMsg'`
- 撤回消息后, 消息接收方会收到一条类型为`'deleteMsg'`的系统通知, 此类系统通知的`msg`为被删除的消息的部分字段。如果是群消息, 那么群里的所有人都会收到这条系统通知. 如果同时在多个端登录了同一个账号, 那么其它端也会收到这条系统通知.
- `'custom'`
- 自定义系统通知

内建系统通知种类

上文中的系统通知类型除了`'custom'`之外的其它类型都属于内建系统通知, 这些类型归为两大种类

`'team'`

`'friend'`

内建系统通知状态

`'init'`: 未处理状态

`'passed'`: 已通过

`'rejected'`: 已拒绝

'error': 错误

处理系统通知

这里涉及到了好友的处理, 请跟好友关系托管合在一起看

```
function handleSysMsgs(sysMsgs) {
    if (!Array.isArray(sysMsgs)) {sysMsgs=[sysMsgs];}
    sysMsgs.forEach(function(sysMsg) {
        var idServer = sysMsg.idServer;
        switch (sysMsg.type) {
        case 'addFriend':
            onAddFriend(sysMsg.friend);
            break;
        case 'applyFriend':
            break;
        case 'passFriendApply':
            onAddFriend(sysMsg.friend);
            break;
        case 'rejectFriendApply':
            break;
        case 'deleteFriend':
            onDeleteFriend(sysMsg.from);
            break;
        case 'applyTeam':
            break;
        case 'rejectTeamApply':
            break;
        case 'teamInvite':
            break;
        case 'rejectTeamInvite':
            break;
        default:
            break;
        }
    });
}
```

标记系统通知为已读状态

SDK 在收到系统通知后会更新系统通知未读数, 开发者需要调用此接口来通知 SDK 将某条系统通知标记为已读状态, 标记后会触发onupdatesysmsgunread回

调

`sysMsgs`为通过`onofflinesysmsgs`或者`onsysmsg`接收到的系统通知或者系统通知数组

```
nim.markSysMsgRead({
    sysMsgs: someSysMsg, // or [someSysMsg]
    done: markSysMsgReadDone
});
function markSysMsgReadDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('标记系统通知为已读状态' + (!error?'成功':'失败'));
}
```

获取本地系统通知

在支持数据库的时候, SDK 会将内建系统通知存储于数据库中
当开发者发现系统通知的未读数大于系统通知数量时, 说明有未读系统通知存储于数据库里面, 需要从本地拉取这部分系统通知
默认获取所有种类的系统通知, 可以传入参数`category`来限制系统通知种类
默认获取所有类型的系统通知, 可以传入参数`type`来限制系统通知类型
默认获取所有已读和未读的系统通知, 可以传入参数`read`来限制已读状态
如果不传, 默认获取所有已读和未读的系统通知
如果传 `true`, 那么只获取已读的系统通知
如果传 `false`, 那么只获取未读的系统通知
`lastIdServer`为上次查询的最后一条系统通知的`idServer`, 第一次不填
`limit`为本次查询的消息数量限制, 最多 100 条, 默认 100 条
默认从最近的系统通知开始往前查找本地系统通知, 可以传入参数`reverse=true`来从第一条系统通知开始往后查找本地系统通知

```
nim.getLocalSysMsgs({
    lastIdServer: 'lastIdServer',
    limit: 100,
    done: getLocalSysMsgsDone
});
function getLocalSysMsgsDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('获取本地系统通知' + (!error?'成功':'失败'));
    if (!error) {
        console.log(obj.sysMsgs);
    }
}
```

更新本地系统通知

更新 `idServer` 对应的本地系统通知
如果不支持数据库, 算成功
如果对应的系统通知不存在, 算成功, 返回 null
这些字段只会被更新到本地数据库, 不会被更新到服务器上

```
nim.updateLocalSysMsg({
    idServer: '1234',
    status: 'bingo',
    localCustom: '{"key","value"}',
    done: updateLocalSysMsgDone
});
function updateLocalSysMsgDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('更新本地系统通知' + (!error?'成功':'失败'));
}
```

删除本地系统通知

删除 `idServer` 对应的本地系统通知
如果不支持数据库, 算成功
如果对应的系统通知不存在, 算成功

```
nim.deleteLocalSysMsg({
    idServer: '1234',
    done: deleteLocalSysMsgDone
});
function deleteLocalSysMsgDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('删除本地系统通知' + (!error?'成功':'失败'));
}
```

删除所有本地系统通知

如果不支持数据库, 算成功
此方法同时会清空系统通知未读数, 开发者会收到onupdatesysmsgunread

```
nim.deleteAllLocalSysMsgs({
    done: deleteAllLocalSysMsgsDone
});
function deleteAllLocalSysMsgsDone(error, obj) {
    console.log(error);
    console.log(obj);
    console.log('删除所有本地系统通知' + (!error?'成功':'失败'));
}
```

自定义系统通知

开发者可以向其他用户或群发送自定义系统通知, 默认只发给在线用户, 如果需要发送给离线用户, 那么需要设置参数sendToOnlineUsersOnly=false, 请参考下面的示例代码
自定义系统通知和自定义消息的区别如下
自定义消息属于消息, 会存储在云信的消息数据库中, 需要跟其他消息一同展现给用户。
自定义系统通知属于系统通知, 用于第三方通知自己, 不会存储在云信的数据库中, SDK不会解析这些通知, SDK仅仅负责传递这些通知。
SDK 不存储自定义系统通知, 不管理自定义系统通知的未读数

```
var content = {
    type: 'type',
    value: 'value'
};
content = JSON.stringify(content);
var msgId = nim.sendCustomSysMsg({
    scene: 'p2p',
    to: 'account',
    content: content,
    sendToOnlineUsersOnly: false,
    apnsText: content,
    done: sendCustomSysMsgDone
});
console.log('正在发送p2p自定义系统通知, id=' + msgId);
function sendCustomSysMsgDone(error, msg) {
    console.log('发送' + msg.scene + '自定义系统通知' + (!error?'成功':'失败') + ', id=' + msg.idClient);
    console.log(error);
    console.log(msg);
}
```

广播消息

用户可以通过应用服务器发送广播消息，应用内的所有指定的在线用户都会收到广播包，此外广播消息支持离线存储，并设置有效期，最长7天，单个应用最多离线存储最近的100条广播通知

广播消息参数：

`broadcastId`: 广播消息id

`body`: 广播消息体

`fromAccid`: 发送该广播消息的账号

`time`: 广播消息发送时间戳

工具方法

图片操作

使用预览文件和发送文件消息拿到图片 url 之后，可以调用 SDK 提供的图片操作来处理图片, 所有的操作在 NIM 和 Chatroom 上都提供, 下文仅以 NIM 为例给出使用方法, 图片操作分为两大类

- 一类是通过 url 拼接的方式来处理图片, 此类接口是同步的, 一般用于展示图片, 通过此类方式生成的 url 仅可用于 UI 展示, 调用其它非图片处理接口的时候不能传入此类 url
- 预览图片通用方法
- 预览去除图片元信息
- 预览图片质量
- 预览interlace图片
- 预览旋转图片
- 预览高斯模糊图片
- 预览裁剪图片
- 预览生成缩略图
- 一类是通过服务器来处理图片, 此类接口是异步的, 一般用于生成新的 url, 调用其它非图片处理接口的时候可以传入此类 url
- 去除图片元信息
- 修改图片质量
- interlace图片
- 旋转图片
- 高斯模糊图片
- 裁剪图片
- 生成缩略图
- 处理图片

预览图片通用方法

只支持通过预览文件或发送文件消息拿到的图片 url, 或者经过其他图片操作后拿到的图片 url

即将以下常用的图片处理方法合并到一个接口中

预览去除图片元信息

预览图片质量

预览interlace图片

预览旋转图片
预览生成缩略图
代码示例

```
  var url = 'http://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=';
  var newImageUrl = nim.viewImageSync({
    url: url, // 必填
    strip: true, // 去除图片元信息 true or false 可选填
    quality: 80, // 图片质量 0 - 100 可选填
    interlace: true, // 渐变清晰， 可选填
    rotate: 90, // 旋转角度，顺时针，可选填
    thumbnail: { // 生成缩略图， 可选填
      width: 80,
      height: 20,
      mode: cover
    }
  });
```

newImageUrl形如： http://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ2OTUyMDc1NzAyNl8wMTA0NmIyMi0yNTQzLTQ2OTYtOWE0OC0zYjk0YjIyMWQyMjA=?imageView&stripmeta=1&quality=80&interlace=1&rotate=90&thumbnail=80z20

预览去除图片元信息

只支持通过预览文件或发送文件消息拿到的图片 url, 或者经过其他图片操作后拿到的图片 url
去除后的图片将不包含 EXIF 信息

```
var url = 'http://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=';
var stripMetaUrl = nim.viewImageStripMeta({
  url: url,
  strip: true
});
```

预览图片质量

只支持通过预览文件或发送文件消息拿到的图片 url, 或者经过其他图片操作后

拿到的图片 url
默认图片质量为100，开发者可以降低图片质量来省流量

```
var url = 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=';
var qualityUrl = nim.viewImageQuality({
    url: url,
    quality: 20
});
// 预览图片质量后的图片 url 如下
// qualityUrl === 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&quality=20'
// 开发者在浏览器中打开上面的链接之后，可以直接修改 url 里面的数字来观察不同的预览图片质量的结果
```

预览interlace图片

只支持通过预览文件或发送文件消息拿到的图片 url, 或者经过其他图片操作后拿到的图片 url
在网络环境较差时, interlace 后的图片会以从模糊到清晰的方式呈现给用户

```
var url = 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=';
var interlaceUrl = nim.viewImageInterlace({
    url: url
});
// interlace 后的图片 url 如下
// interlaceUrl === 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&interlace=1'
```

预览旋转图片

只支持通过预览文件或发送文件消息拿到的图片 url, 或者经过其他图片操作后拿到的图片 url

```
var url = 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=';
var rotateUrl = nim.viewImageRotate({
    url: url,
    angle: 90
});
// 旋转后的图片的 url 如下
// rotateUrl === 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&rotate=90'
// 开发者在浏览器中打开上面的链接之后，可以直接修改 url 里面的数字来观察不同的旋转结果
```

预览高斯模糊图片

只支持通过预览文件或发送文件消息拿到的图片 url, 或者经过其他图片操作后拿到的图片 url

```
var url = 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=';
var blurUrl = nim.viewImageBlur({
    url: url,
    radius: 5,
    sigma: 3
});
// 高斯模糊后的图片 url 如下
// blurUrl === 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&blur=5x3'
// 开发者在浏览器中打开上面的链接之后，可以直接修改 url 里面的数字来观察不同的高斯模糊后的结果
```

预览裁剪图片

只支持通过预览文件或发送文件消息拿到的图片 url, 或者经过其他图片操作后拿到的图片 url
从坐标 (x, y) 处截取尺寸为 width*height 的图片，(0, 0) 代表左上角
width/height 不能小于0, 如果 width/height 大于图片的原始宽度/高度, 那么将被替换为图片的原始宽度/高度
举个栗子, 假如说之前通过预览文件拿到的 url 为

https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=

传入 x/y/width/height 为 100/0/250/250 得到的裁剪图片的 url 为
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&crop=100_0_250_250

开发者在浏览器中打开上面的链接之后, 可以直接修改 url 里面的数字来观察不同的裁剪结果

```
var url = 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=';
var cropUrl = nim.viewImageCrop({
    url: url,
    x: 100,
    y: 0,
    width: 250,
    height: 250
});
// 裁剪后的图片的 url 如下
// cropUrl === 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&crop=100_0_250_250'
// 开发者在浏览器中打开上面的链接之后, 可以直接修改 url 里面的数字来观察不同的裁剪结果
```

预览生成缩略图

只支持通过预览文件或发送文件消息拿到的图片 url, 或者经过其他图片操作后拿到的图片 url
width/height 限制了缩略图的尺寸
width/height 必须大于等于 0, 不能同时为 0, 必须小于 4096
不同模式下生成的缩略图是不一样的, 目前支持以下三种模式
`'cover'`: 原图片等比缩略, 缩略图一边等于请求的尺寸, 另一边大于请求的尺寸, 即缩略图刚好能覆盖住尺寸为 width*height 的矩形
举个栗子, 假如说之前通过预览文件拿到的 url 为
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA

=
此模式下传入 80*100 的尺寸得到的缩略图 url 为
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80z100
开发者在浏览器中打开上面的链接之后, 可以直接修改 url 里面的数字来观察不同尺寸得到的缩略图
`'contain'`: 原图片等比缩略, 缩略图一边等于请求的尺寸, 另一边大于请求的尺寸, 即尺寸为 width*height 的矩形刚好能覆盖住缩略图
还是拿上面的 url 为例, 传入 80*100 的尺寸得到的缩略图 RUL 为
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80x100
开发者在浏览器中打开上面的链接之后, 可以直接修改 url 里面的数字来观察不同尺寸得到的缩略图
`'crop'`: 先等比缩略原图片, 使得一边等于请求的尺寸, 另一边大于请求的尺寸, 然后对大于请求尺寸的那条边进行裁剪, 使得最终的图片大小刚好等于请求的尺寸
还是拿上面的 url 为例, 传入 80*100 的尺寸得到的缩略图 url 为
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80y100
开发者在浏览器中打开上面的链接之后, 可以直接修改 url 里面的数字来观察不同尺寸得到的缩略图
如果缩略图尺寸大于图片尺寸，默认情况下图片不会被放大，可以传入参数`enlarge=true`来放大图片
举个栗子, 假如说之前通过预览文件拿到的 url 为
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=
此 url 对应的图片尺寸为 512-256, 如果使用 `'cover'` 模式来裁剪, 传入尺寸 1024-512, 得到的缩略图 url 为
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0Mz

E0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=1024z512

会发现图片尺寸并没有放大, 如果再传入参数 `enlarge=true`, 得到的缩略图 url 为

https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=1024z512&enlarge=1

会发现图片被放大了

`'crop'` 模式下可以传入参数 axis.x 或 axis.y 来控制最后一步裁剪的位置

x/y 必须为整数, 取值范围为 0-10, 此方法内部使用 Math.round 来格式化 x/y

x 为 0 时表示裁取最左端, x 为 10 时表示裁取最右端

y 为 0 时表示裁取最上端, y 为 10 时表示裁取最下端

x/y 默认值均为 5, 即裁取正中间

拿上面的 url 为例, 传入 80*100 的尺寸得到的缩略图 url 为

https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80y100

依次传入 x=0,1,2,3,4,5,6,7,8,9,10 得到的缩略图 url 为

https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80y100&axis=0_5

https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80y100&axis=1_5

https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80y100&axis=2_5

https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80y100&axis=3_5

https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80y100&axis=4_5

https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80y100&axis=5_5
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80y100&axis=6_5
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80y100&axis=7_5
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80y100&axis=8_5
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80y100&axis=9_5
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80y100&axis=10_5

拿上面的 url 为例, 传入 200*50 的尺寸得到的缩略图 RUL 为

https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=200y50

依次传入 y=0,2,4,6,8,10 得到的缩略图 url 为

https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=200y50&axis=5_0
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=200y50&axis=5_1
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=200y50&axis=5_2
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0Mz

E0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=200y50&axis=5_3
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=200y50&axis=5_4
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=200y50&axis=5_5
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=200y50&axis=5_6
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=200y50&axis=5_7
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=200y50&axis=5_8
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=200y50&axis=5_9
https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=200y50&axis=5_10
*

```
var url = 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=';
var thumbnailUrl = nim.viewImageThumbnail({
    url: url,
    mode: 'cover',
    width: 80,
    height: 100
});
// 缩略后的图片的 url 如下
// thumbnailUrl === 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltYV8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVlZjZlNzZjMzA=?imageView&thumbnail=80z100'
// 开发者在浏览器中打开上面的链接之后，可以直接修改 url 里面的数字来观察不同
```

```
的裁剪结果
 *
thumbnailUrl = nim.viewImageThumbnail({
    url: url,
    mode: 'contain',
    width: 80,
    height: 100
});
// 缩略后的图片的 url 如下
// thumbnailUrl === 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltY
V8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVl
ZjZlNzZjMzA=?imageView&thumbnail=80x100'
// 开发者在浏览器中打开上面的链接之后，可以直接修改 url 里面的数字来观察不同
的裁剪结果
 *
thumbnailUrl = nim.viewImageThumbnail({
    url: url,
    mode: 'contain',
    width: 80,
    height: 100
});
// 缩略后的图片的 url 如下
// thumbnailUrl === 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltY
V8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVl
ZjZlNzZjMzA=?imageView&thumbnail=80y100'
// 开发者在浏览器中打开上面的链接之后，可以直接修改 url 里面的数字来观察不同
的裁剪结果

thumbnailUrl = nim.viewImageThumbnail({
    url: url,
    mode: 'contain',
    width: 80,
    height: 100,
    axis: {
        // x 可取的值请参考上文描述
        x: 0
    }
});
// 缩略后的图片的 url 如下
// thumbnailUrl === 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltY
V8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVl
ZjZlNzZjMzA=?imageView&thumbnail=80y100&axis=0_5'
// 开发者在浏览器中打开上面的链接之后，可以直接修改 url 里面的数字来观察不同
的裁剪结果

thumbnailUrl = nim.viewImageThumbnail({
    url: url,
    mode: 'contain',
    width: 80,
    height: 100,
    axis: {
        // y 可取的值请参考上文描述
        y: 0
    }
```

```
});
// 缩略后的图片的 url 如下
// thumbnailUrl === 'https://nim.nos.netease.com/MTAxMTAwMg==/bmltY
V8xNDc5OTNfMTQ0MzE0NTgyNDI0M184YjFkYTMwMS02NjcxLTRiYjktYTUwZC04ZTVl
ZjZlNzZjMzA=?imageView&thumbnail=80y100&axis=5_0'
// 开发者在浏览器中打开上面的链接之后, 可以直接修改 url 里面的数字来观察不同
的裁剪结果
```

数据转BLOB

将包含 MIME type 和 base64 数据的 data URL 转换为 Blob 对象

```
var dataURL = '
wBDAAgGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8UHRofHh0aHBwgJC4nICIsIxwcKDcp
LDAxNDQ0Hyc5PTgyPC4zNDL/2wBDAQkJCQwLDBgNDRgyIRwhMjIyMjIyMjIyMjIyMjI
yMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjL/wAARCAAyADIDASIAAh
EBAxEB/8QAGgABAAMBAQEAAAAAAAAAAAAAAAIEBgUDAf/EACwQAAEEAQMCBQMFAQAAA
AAAAAEAAgMRBAUSISIxBhNBUWEUMnEjkaGxstH/xAAaAQACAwEBAAAAAAAAAAAAAAAA
AQIDBAUG/8QAIREAAgIBAwUBAAAAAAAAAAAAAAECERIEFCExM0FxodH/2gAMAwEAAhE
DEQA/ANkiKtm5gxI203fK87Y2XVnv+w7lXNpK2ebLKhHMyUuDHbtpokdr9lksjPzM18
7pMowYEBIfKw7TIR3ArsL4vuquJNlYenYub9TPE2U1I272lx4cGngjtYPp8rPuoWLI3
SLlaTqr8pz8XLa2PNiHW1vZw9CPhdVXxkpK0MIiKQBZzXcgxZGVJz+hh20D3cTf+QtE
40LDS4+gaLJXE13FfFqNSxyRvkg8qTpI2HktPNe7gVRqe2x4tqzh5OP5PhAxMaDULXH
n1sElWtQo+HJdvb6cEbvwP5VbLex3hzJxnSmSXHjDJKFciq49vlT1SYjw8xrTuknYyN
u1v3E1dD8WuZza9kPwSymAaRnMNzF7GFx4LmubyCtoOyxpx/q9UwtMi3mPGAllcfgU3
n3WyW/SJ4EkERFrA+EWORahkxszJnS5QEz3MEZMguwBX9eq9FFzA4EEWCk0n1HbqjN5
GmQyajKxkhDiwxu08dba+1xruOVX1DCZp2JFPNlDysZrY4uN5DjxurgE178LRnToDN5
vXd9r4UpNPxZQBLE14BsB3ItZ9tF3Y4OKmnNWvPNfTx0zAxsKHdj9Zl63yu5dIT6n/i
vqLWNY3a0ABSWhJJUiIRETAIiIiIAIiIAIiIAIiIA//Z';

var blob = NIM.blob.fromDataURL(dataURL);

// blob instanceof Blob === true;
```