聊天消息

一、功能概述

`聊天室模块(chat)` 提供聊天功能 Api，用于开发者集成聊天功能。

二、获取 Socket 实例

2.1 获取聊天室 socket 实例

Api 方法： `getSocket(check?: false): undefined | Socket`

参数说明：

- check：无需检查 socket 是否存在，`false` 类型，选传

返回值说明： `undefined | Socket` 类型

三、聊天室状态信息

3.1 获取聊天室信息

与用户、聊天室有关的状态均存储在聊天室模块中，并通过 `getChatInfo` 获取聊天室状态信息。

Api 方法： `getChatInfo(): ChatModuleInfo`

返回值说明： 聊天室状态信息，`ChatModuleInfo` 类型，详细类型说明如下

属性名	说明	类型
`chatExited`	聊天室是否被退出	`boolean`
`chatRoomIsClosed`	聊天室是否已关闭	`boolean`
`isKicked`	是否被踢出	`boolean`
`isShield`	是否被禁言	`boolean`

示例：

```
const chatInfo = watchCore.chat.getChatInfo();
console.log('房间是否被关闭', chatInfo.chatRoomIsClosed);
console.log('当前用户是否被禁言', chatInfo.isShield);
```

四、聊天消息来源

通过聊天室消息事件 `ChatEvents.ChatMessage`、聊天历史记录 Api `getChatHistory` 等获取的聊天消息类型均为 ChatMsgSource 类型 所有消息数据都有对应的消息来源 `msgSource` 字段，该字段为 `ChatMsgSource` 枚举，开发者可根据该字段显示相应的消息样式。

Enum 枚举： `ChatMsgSource`

常量	枚举成员	说明	消息类型	服务端消息
`'speak'`	`ChatMsgSource.Speak`	发言消息	`ChatMsgSpeakType`	✓
`'image'`	`ChatMsgSource.Image`	图片消息	`ChatMsgImageType`	✓
`'emotion'`	`ChatMsgSource.Emotion`	表情图片消息	`ChatMsgEmotionType`	✓
`'reward'`	`ChatMsgSource.Reward`	打赏消息	`ChatMsgRewardType`	✓
`'file'`	`ChatMsgSource.File`	文件分享消息	`ChatMsgFileType`	✓
`'redpaper'`	`ChatMsgSource.Redpaper`	红包消息	`ChatMsgRedpaperType`	✓
`'redpaperReceive'`	`ChatMsgSource.RedpaperReceive`	红包领取消息	`ChatMsgRedpaperReceiveType`	×
`'customerMessage'`	`ChatMsgSource.CustomerMessage`	自定义消息(服务端)	`ChatMsgCustomerMessageType`	✓
`'customMessage'`	`ChatMsgSource.CustomMessage`	自定义消息(客户端)	`ChatMsgCustomMessageType`	x

常量	枚举成员	说明	消息类型	服务端消息
`'system'`	`ChatMsgSource.System`	系统消息	`ChatMsgSystemType`	×
`'motivationLike'`	`ChatMsgSource.MotivationLike`	课堂激励点赞消息	`ChatMsgMotivationLikeType`	×
`'speakTop'`	`ChatMsgSource.SpeakTop`	评论上墙	-	-
`'speakCancelTop'`	`ChatMsgSource.SpeakCancelTop`	-	-	-
`'effect'`	`ChatMsgSource.Effect`	消息特效	-	×
`'iarCheckIn'`	`ChatMsgSource.IarCheckIn`	签到消息	`ChatMsgIarCheckInMessageType`	✓
`'iarAnswerCard'`	`ChatMsgSource.IarAnswerCard`	答题卡消息	`ChatMsgIarAnswerCardMessageType`	✓
`'iarQuestionnaire'`	`ChatMsgSource.IarQuestionnaire`	问卷消息	`ChatMsgIarQuestionnaireMessageType`	✓
`'unknown'`	`ChatMsgSource.Unknown`	未知的消息来源	-	×

五、发送聊天消息

5.1 发送文本消息

用于观众进行聊天发言，调用过程中会触发 ChatEvents.ChatMessage 聊天消息事件。

Api 方法： `sendSpeakMsg(options: SendSpeakMsgOptions): Promise<ChatMsgSpeakType>`

参数说明：

- options：发言参数，`SendSpeakMsgOptions` 类型，必传，详细类型说明如下

参数名	说明	类型	必须	默认值
`forceSend`	强制发送，忽略禁言判断	`boolean`	否	`false`
`content`	发言内容	`string`	是	-
`onlyLocalMsg`	是否仅发送本地消息	`boolean`	否	`false`
`quoteMsg`	引用的消息	`ChatMsgQuoteOriginType`	否	-

返回值说明： 发送到服务器后的消息对象，`Promise<ChatMsgSpeakType>` 类型，详细类型说明如下

属性名	说明	类型
`id`	消息唯一标识	`string`
`msgSource`	消息来源	`Speak`
`time`	消息时间	`number`
`user`	用户信息	`ChatMessageUser<ChatUserType>`
`content`	发言内容	`string`
`quote`	回复内容	`ChatMsgQuoteType`
`isLocal`	是否本地发送的消息	`boolean`
`overLen`	是否已超出服务端文本长度	`boolean`
`isOverLength`	是否超长文本	`boolean`
`isSended`	是否已经发送完成	`boolean`
`isSendFailed`	是否发送失败	`boolean`

示例：

```
import { ChatMsgQuoteOriginType, ChatMsgSpeakType } from '@polyv/live-watch-sdk';
// 发送的文本
const content = '今天天气真好[呲牙][酷]';
// 被回复的消息
const currentQuoteMsg: ChatMsgSpeakType | undefined = undefined;
// 发送消息
watchCore.chat.sendSpeakMsg({
  content,
  quoteMsg: currentQuoteMsg,
});
```

5.2 发送图片消息

用于观众发送图片消息，调用过程中会触发 ChatEvents.ChatMessage 聊天消息事件。

Api 方法： `sendImageMsg(options: SendImageMsgOptions): Promise<ChatMsgImageType>`

参数说明：

- options：发送图片参数，`SendImageMsgOptions` 类型，必传，详细类型说明如下

参数名	说明	类型	必须	默认值
`forceSend`	强制发送，忽略禁言判断	`boolean`	否	`false`
`imageId`	图片 id，建议使用 uuid(v4) 生成	`string`	是	-
`imageUrl`	图片地址	`string`	是	-
`size`	图片尺寸	`Object`	否	-

返回值说明： 发送到服务器后的图片消息对象，`Promise<ChatMsgImageType>` 类型，详细类型说明如下

属性名	说明	类型
`id`	消息唯一标识	`string`
`msgSource`	消息来源	`Image`
`time`	消息时间	`number`
`user`	用户信息	`ChatMessageUser<ChatUserType>`

属性名	说明	类型
`imageId`	图片 id	`string`
`imageUrl`	图片地址	`string`
`size`	图片尺寸	`Object`
`isLocal`	是否本地发送的消息	`boolean`
`isSended`	是否已经发送完成	`boolean`
`localImageUrl`	本地发送消息的图片地址	`string`
`isIllegal`	是否违规	`boolean`
`isSendFailed`	是否发送失败	`boolean`

示例：

```
import { uuidV4 } from '@polyv/utils/string';
// 图片 id
const imageId = uuidV4();
// 图片地址
const imageUrl = '发送到图片地址，需要带有协议';
// 图片地址
const size = { width: 200, height: 100 };
// 发送图片消息
watchCore.chat.sendImageMsg({ imageId, imageUrl, size });
```

5.3 发送表情图片消息

通过 `getEmotionImages` 方法获取表情图片列表后，通过列表项的 `id`、`url` 发送表情图片消息，调用过程中会触发 ChatEvents.ChatMessage 聊天消息事件。

开发者可通过 getEmotionImages 获取表情图片列表。

Api 方法： `sendEmotionImageMsg(options: SendEmotionImageMsgOptions): Promise<ChatMsgEmotionType>`

参数说明：

- options：发送表情图片参数，`SendEmotionImageMsgOptions` 类型，必传，详细类型说明如下

参数名	说明	类型	必须	默认值
`forceSend`	强制发送，忽略禁言判断	`boolean`	否	`false`
`emotionId`	表情 id	`string`	是	-
`emotionUrl`	表情图片地址	`string`	是	-

返回值说明： 发送到服务器后的消息对象，`Promise<ChatMsgEmotionType>` 类型，详细类型说明如下

属性名	说明	类型
`id`	消息唯一标识	`string`
`msgSource`	消息来源	`Emotion`
`time`	消息时间	`number`
`user`	用户信息	`ChatMessageUser<ChatUserType>`
`emotionId`	表情 id	`string`
`emotionUrl`	表情图片地址	`string`
`size`	图片尺寸，socket 消息中的没有尺寸返回	`Object`
`isLocal`	是否本地发送的消息	`boolean`
`isSended`	是否已经发送完成	`boolean`
`isSendFailed`	是否发送失败	`boolean`

示例：

```
// 获取表情图片列表
const emotionImages = await watchCore.chat.getEmotionImages();
// 发送表情图片
const item = emotionImages[2];
watchCore.chat.sendEmotionImageMsg({ emotionId: item.id, emotionUrl: item.url });
```

5.4 发送系统消息

用于发送系统消息到聊天区，注意该消息并不会发送到服务端，调用过程中会触发 ChatEvents.ChatMessage 聊天消息事件。

Api 方法： `sendSystemMsg(content: string, type?: string): void`

参数说明：

- content：消息内容，`string` 类型，必传
- type：`string` 类型，选传

示例：

```
watchCore.chat.sendSystemMsg('聊天室已关闭');
```

5.5 发送自定义消息(客户端)

用于发送自定义消息(客户端)，调用过程中会触发 ChatEvents.ChatMessage 聊天消息事件，该事件返回的数据中会对应 ChatMsgSource.CustomMessage 的聊天消息数据

Api 方法： `sendCustomMessage(options: SendCustomMessageOptions<T>): Promise<string>`

> 从 v0.10.0 版本开始支持

参数说明：

- options：自定义消息选项，`SendCustomMessageOptions<T>` 类型，必传，详细类型说明如下

参数名	说明	类型	必须	默认值
`EVENT`	自定义事件，默认为 'client-custom'	`string`	否	-
`data`	自定义数据，但需要传入一个对象	`T`	是	-
`version`	自定义版本，默认为 1	`number`	否	-
`tip`	自定义提示，默认为空	`string`	否	-

参数名	说明	类型	必须	默认值
`joinHistoryList`	是否加入聊天历史数据, 默认为 true	`boolean`	否	-

返回值说明： 发送成功的自定义消息id， `Promise<string>` 类型

示例：

```
watchCore.chat.sendCustomMessage({ data: { test: '测试' } });
```

5.6 插入本地打赏消息

用于观众优先发送本地打赏消息，调用过程中会触发 ChatEvents.ChatMessage 聊天消息事件。

注意，插入本地打赏消息肯定为当前观众

Api 方法： `insertLocalRewardChatMsg(options: InsertLocalRewardChatMsgOptions): void`

> 从 v0.11.0 版本开始支持

参数说明：

- options：打赏消息选项， `InsertLocalRewardChatMsgOptions` 类型，必传

5.7 根据id 移除聊天消息

Api 方法： `removeChatMsg(id: string): void`

> 从 v1.2.0 版本开始支持

参数说明：

- id： `string` 类型，必传

5.8 根据id，替换对应的聊天内容

Api 方法： `replaceChatMsg(id: string, chatMsg: ChatMsgType): void`

> 从 v1.2.0 版本开始支持

参数说明：

- id：`string` 类型，必传
- chatMsg：`ChatMsgType` 类型，必传

六、监听聊天消息事件

当有观众发言、打赏等涉及聊天消息操作时，聊天室模块会回调 `ChatEvents.ChatMessage` 事件，开发者可通过监听该事件进行聊天消息的渲染。

由于本地发送消息时，在发送至服务端前就会回调 `ChatEvents.ChatMessage` 事件，此时回调的消息 id 即 `chatMsg.id` 为本地创建的 id，服务端回调后，通过 `ChatEvents.ReplaceChatMessage` 进行消息数据替换（包括图片违规等均通过该事件修改消息数据）。

```
import { ChatEvents, ChatMsgType } from "@polyv/live-watch-sdk";

// 聊天消息列表
const chatMsgList: ChatMsgType[] = [];

// 聊天消息事件
watchCore.chat.eventEmitter.on(ChatEvents.ChatMessage, (data) => {
  // 插入到聊天消息列表
  chatMsgList.push(data.chatMsg);
  // 渲染聊天消息...
});

// 替换聊天消息数据事件
watchCore.chat.eventEmitter.on(ChatEvents.ReplaceChatMessage, (data) => {
  // 需要被替换的消息 id
  const replaceId = data.id;
  // 新的消息对象
  const chatMsg = data.chatMsg;

  const index = chatMsgList.findIndex((item) => item.id === replaceId);
  if (index !== -1) {
    chatMsgList[index] = chatMsg;
  }
  // 将视图的消息节点替换...
});
```

七、获取聊天历史记录

7.1 设置聊天历史消息是否加密

仅 `getChatHistoryByTime` 方法支持加密返回

Api 方法： `setChatHistoryEncrypt(chatHistoryEncrypt: boolean): void`

> 从 v2.9.0 版本开始支持

参数说明：

- chatHistoryEncrypt：聊天历史消息是否加密，`boolean` 类型，必传

7.2 获取聊天历史消息

Api 方法： `getChatHistory(options?: GetChatHistoryOptions): Promise<ChatMsgType[]>`

> 从 v2.16.0 开始废弃 通过 `getChatHistory` 方法获取频道下的聊天历史消息。

参数说明：

- options：获取选项，`GetChatHistoryOptions` 类型，选传，默认 `{}`，详细类型说明如下

参数名	说明	类型	必须	默认值
`start`	消息起始索引	`number`	否	`0`
`end`	消息终止索引	`number`	否	`9`
`onlySpecialMsg`	是否仅获取特殊角色发言	`boolean`	否	`false`
`filterCustomMsg`	是否过滤自定义消息	`boolean`	否	`true`
`filterRedpaperMsg`	是否过滤红包信息	`boolean`	否	`false`

返回值说明： `Promise<ChatMsgType[]>` 类型

示例：

```
// 获取 0 ～ 19 条消息
const historyData = await watchCore.chat.getChatHistory({
  start: 0,
  end: 19,
});
// 返回数据示例，类型为：ChatMsgType[]
[{
  id: '5191c230-c6c4-11ed-8c31-23e8ced55946',
  time: 1679278200247,
  msgSource: 'speak',
  content: '今天天气真好[呲牙][酷]',
  user: {
    userId: '18012345678',
    nick: '小明',
    pic: '头像地址',
  },
}]
```

7.3 根据场次号分页获取聊天历史消息

通过 `getChatHistoryBySessionId` 方法获取直播场次中的聊天历史记录

Api 方法： `getChatHistoryBySessionId(options: GetChatHistoryBySessionIdOptions): Promise<PageContent<ChatMsgType>>`

参数说明：

- options：获取选项，`GetChatHistoryBySessionIdOptions` 类型，必传，详细类型说明如下

参数名	说明	类型	必须	默认值
`sessionId`	场次 Id	`string`	是	-
`pageNumber`	页数	`number`	否	`1`
`pageSize`	每页数量	`number`	否	`10`

返回值说明： `Promise<PageContent<ChatMsgType>>` 类型

示例：

```
// 指定的场次号
const sessionId = 'gksk8f2itb';
const result = await watchCore.chat.getChatHistoryBySessionId({
  sessionId,
});
console.log('当前页', result.pageNumber);
console.log('每页数量', result.pageSize);
console.log('总条目数', result.totalItems);
console.log('总页数', result.totalPages);
console.log('消息列表', result.contents); // ChatMsgType[]
```

7.4 根据时间戳获取聊天历史消息

Api 方法： `getChatHistoryByTime(options: GetChatHistoryByTimeOptions): Promise<ChatMsgType[]>`

> 从 v0.9.0 版本开始支持

参数说明：

- options：请求选项，`GetChatHistoryByTimeOptions` 类型，必传，详细类型说明如下

参数名	说明	类型	必须	默认值
`timestamp`	时间戳	`number`	否	-
`count`	时间戳相对应的数量	`number`	否	`1`
`size`	获取的数量	`number`	否	`10`
`exclude`	排除的消息类型	`string[]`	否	-
`validateExcludeMsgSource`	是否校验"排除的消息类型"	`boolean`	否	`true`
`mode`	聊天请求历史数据模式	`ChatRequestHistoryMode`	否	-
`onlySpecialMsg`	是否只获取特殊身份消息	`boolean`	否	`false`

返回值说明： 消息列表，`Promise<ChatMsgType[]>` 类型

八、消息/评论上墙

8.1 获取评论上墙数据

Api 方法： `getSpeakTopInfo(): undefined | ChatMsgSpeakTopType`

> 从 v1.5.0 版本开始支持

返回值说明： `undefined | ChatMsgSpeakTopType` 类型

8.2 设置评论上墙数据

Api 方法： `setSpeakTop(data: ChatMsgSpeakTopType | SliceIdSpeakTop): void`

> 从 v1.5.0 版本开始支持

参数说明：

- data：`ChatMsgSpeakTopType | SliceIdSpeakTop` 类型，必传

8.3 取消评论上墙

Api 方法： `cancelSpeakTop(): void`

> 从 v1.5.0 版本开始支持

九、消息/文件

9.1 获取频道文件列表

Api 方法： `getChannelFileList(params: GetChannelFileListParams): Promise<GetChannelFileListResponse>`

> 从 v2.6.0 版本开始支持

参数说明：

- params：`GetChannelFileListParams` 类型，必传，详细类型说明如下

参数名	说明	类型	必须	默认值
`pageSize`	-	`number`	否	-
`pageNumber`	-	`number`	否	-

返回值说明： `Promise<GetChannelFileListResponse>` 类型，详细类型说明如下

属性名	说明	类型
`list`	-	`ChannelFileListItem[]`
`page`	-	`number`
`size`	-	`number`
`totalCount`	-	`number`
`totalPage`	-	`number`

十、其他

10.1 获取超长消息的完整文本

讲师可发送超过 2000 字的文本消息，该消息为超长文本消息（通过 `chatMsg.isOverLength === true` 判断），`chat` 提供消息文本只会返回前 1000 字的消息字符串，如需展示完整的消息文本，可调用该方法获取。

Api 方法： `getFullMessage(id: string, chatMsg?: ChatMsgSpeakType): Promise<string>`

参数说明：

- id：聊天消息 id，`string` 类型，必传
- chatMsg：聊天消息，在未超过服务端聊天文本长度或者请求异常时会使用聊天消息内容返回 [v0.11.0 新增]，`ChatMsgSpeakType` 类型，选传，详细类型说明如下

参数名	说明	类型	必须	默认值
`id`	消息唯一标识	`string`	是	-
`msgSource`	消息来源	`Speak`	是	-
`time`	消息时间	`number`	是	-
`user`	用户信息	`ChatMessageUser<ChatUserType>`	是	-
`content`	发言内容	`string`	是	-
`quote`	回复内容	`ChatMsgQuoteType`	否	-
`isLocal`	是否本地发送的消息	`boolean`	否	-
`overLen`	是否已超出服务端文本长度	`boolean`	否	-
`isOverLength`	是否超长文本	`boolean`	否	-
`isSended`	是否已经发送完成	`boolean`	否	-
`isSendFailed`	是否发送失败	`boolean`	否	-

返回值说明： `Promise<string>` 类型

示例：

```
import { ChatMsgSpeakType } from '@polyv/live-watch-sdk';

async function getFullMessageText(chatMsg: ChatMsgSpeakType): Promise<string> {
  if (!chatMsg.isOverLength) {
    throw new Error('该消息非超长文本');
  }

  const result = await watchCore.chat.getFullMessage(chatMsg.id, chatMsg);
  console.log('完整的文本', result);
  return result;
}
```

10.2 获取聊天室设置

从 v1.2.0 开始可以通过 `PlvChatModule.generateDefaultChatSetting()` 来获取默认配置

用于获取管理后台的聊天室设置信息。

Api 方法： `getChatSetting(): ChatSetting`

返回值说明： 聊天室设置信息，`ChatSetting` 类型，详细类型说明如下

属性名	说明	类型
`watchChatEnabled`	聊天室是否长时间未使用	`boolean`
`showCustomMessageEnabled`	是否显示自定义消息	`boolean`
`quoteReplyEnabled`	聊天引用回复开关	`boolean`
`chatTranslateEnabled`	翻译开关	`boolean`
`chatRobotEnabled`	虚拟人数开关	`boolean`
`restrictChatEnabled`	聊天室并发人数限制开关	`boolean`
`likeEnabled`	点赞开关	`boolean`
`likeSingleClickEnabled`	单击点赞开关	`boolean`
`likeHoldEnabled`	长按点赞开关	`boolean`
`filterManagerMsgEnabled`	是否只看主持人信息	`boolean`
`viewerSendImgEnabled`	发送图片开关	`boolean`
`welcomeEnabled`	欢迎语开关	`boolean`
`emotionalFeedbackEnabled`	情绪反馈开关	`boolean`
`chatOnlineNumberEnable`	聊天室在线人数开关	`boolean`
`faceEmotionEnabled`	黄脸表情开关	`boolean`
`imageEmotionEnabled`	图片表情开关	`boolean`
`getChatHistoryByTimestampEnabled`	根据时间戳获取聊天历史记录	`boolean`
`chatMessageLayout`	聊天消息布局	`ChatMessageLayout`
`portraitChatMessageLayout`	竖屏聊天消息布局	`ChatMessageLayout`

属性名	说明	类型
`chatMessageAvatarDisplay`	聊天消息头像是否显示	`boolean`
`portraitChatMessageAvatarDisplay`	竖屏聊天消息头像是否显示	`boolean`
`chatUIVersion`	聊天室 UI 版本	`'v1' | 'v2'`
`likeIconUrl`	点赞图标	`string`
`likeIconType`	点赞图标类型	`string`
`likeEffectType`	点赞特效类型	`LikeEffectType`
`likeEffectIcons`	点赞特效配置	`string[]`
`likeCountEnabled`	是否显示点赞数	`boolean`
`customViewerLabelUrl`	自定义观众标签地址	`string`

示例：

```
const setting = watchCore.chat.getChatSetting();
console.log('观看页聊天室开关', setting.watchChatEnabled);
console.log('是否显示翻译功能', setting.chatTranslateEnabled);
```

10.3 转换发言内容

通过 `parseSpeakContent` 将观众发言中的表情、链接转换成 html 元素。

转换顺序：parseLink > removeEmotion > parseEmotion > parseLineBreak

Api 方法： `parseSpeakContent(content: string, options?: ParseOptions): string`

参数说明：

- content：发言内容，`string` 类型，必传
- options：转换选项，`ParseOptions` 类型，选传，详细类型说明如下

参数名	说明	类型	必须	默认值
`parseLink`	是否转换链接	`boolean`	否	`false`

参数名	说明	类型	必须	默认值
`removeEmotion`	移除表情内容	`boolean`	否	`false`
`parseEmotion`	是否转换表情	`boolean`	否	`false`
`parseLineBreak`	是否转换换行符	`boolean`	否	`false`

返回值说明： 转换后的 html 字符

示例：

```
// 转换链接
watchCore.chat.parseSpeakContent('这是我们的官网地址：https://www.polyv.net/', { parseLink: true });
// 转换后的字符串：这是我们的官网地址：<a target="_blank" rel="noopener"
href="https://www.polyv.net/">https://www.polyv.net/</a>

// 移除表情
watchCore.chat.parseSpeakContent('今天天气真好[呲牙]', { removeEmotion: true });
// 转换后的字符串：今天天气真好

// 转换表情
watchCore.chat.parseSpeakContent('今天天气真好[呲牙]', { parseEmotion: true });
// 转换后的字符串：今天天气真好<img src="黄脸表情图片地址" alt="呲牙" class="plv-emotion-img" title="呲牙" />

// 将换行符转成 <br />
watchCore.chat.parseSpeakContent('这是一段文字\n这是另一段文字', { parseLineBreak: true });
// 转换后的字符串：这是一段文字<br />这是另一段文字
```