聊天重放-示例

此处提供一个`聊天室模块-聊天重放`的基础使用示例，为了实现该功能，该示例还包含播放器和聊天室的基础内容

更多内容请结合文档参考 观看页 SDK 及其配套的开源项目

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta
      name="viewport"
      content="width=device-width, initial-scale=1.0"
    />
    <title>Chat-Replay</title>
    <style>
      html,
      body {
        padding: 0;
        margin: 0;
      }

      .demo-intro {
        padding: 0 16px;
      }

      .demo-player-container {
        width: 640px;
        height: 360px;
        padding: 16px;
        margin-bottom: 80px;
      }

      #player {
        width: 100%;
        height: 100%;
        position: relative;
      }

      .demo-player-control {
        display: flex;
        margin-top: 24px;
        gap: 12px;
      }

      .demo-chat-container {
        min-width: 640px;
        max-width: 80vw;
        height: 360px;
        padding: 16px;
      }

      .demo-chat__list {
        width: 100%;
        height: 100%;
        overflow-y: auto;
      }

      .demo-chat-control {
        display: flex;
        margin-top: 24px;
```

```
        gap: 12px;
      }
    </style>
  </head>
  <body>
    <div class="demo-intro">
      <p>说明</p>
      <ul>
        <li>聊天重放是模拟直播场景，所以不建议在聊天重放的状态下支持观众发言</li>
        <li>
          获取聊天历史消息时，并不是真的去做分页查询，从而加载上一页数据和下一页数据，而是根据对应的一条聊天消息，加载它的上一批数据和下一批数据，所以此处不做完整示例，请自行根据"虚拟列表"的机制来实现上下滚动加载聊天记录的功能
        </li>
      </ul>
    </div>

    <div class="demo-player-container">
      <div id="player"></div>
      <div class="demo-player-control">
        <button class="demo-player-control__play-btn">播放</button>
        <button class="demo-player-control__pause-btn">暂停</button>
      </div>
    </div>

    <div class="demo-chat-container">
      <div class="demo-chat-input-control">
        <input
          class="demo-chat-input-control__input"
          type="text"
          placeholder="聊天重放不支持发送消息"
        />
        <button
          class="demo-chat-input-control__send-btn"
          disabled
        >
          发送普通文本消息
        </button>
        <button
          class="demo-chat-input-control__send-btn--custom"
          disabled
        >
          发送自定义消息
        </button>
        <button
          class="demo-chat-input-control__send-like"
          disabled
        >
          点赞:
          <span id="like-count">0</span>
        </button>
        <button
          class="demo-chat-input-control__load-prev-btn"
          disabled
        >
          加载上一页
        </button>
```

```
      <button
        class="demo-chat-input-control__load-next-btn"
        disabled
      >
        加载下一页
      </button>
      <br />
    </div>
    <ul class="demo-chat__list"></ul>
  </div>

  <!-- 此处使用在线JS链接演示，开发者可使用npm包形式安装 -->
  <script src="https://websdk.videocc.net/live-watch-sdk/latest/lib/polyv-live-watch-sdk.umd.js"></script>
  <script>
    const {
      createWatchCore,

      PolyvWatchCoreEvents,
      ChannelEvents,
      PlayerEvents,
      ChatEvents,

      LiveStatus,
      ChatMsgReplayStatus,
      ChatReplayRequestHistoryMode,
      ChatMsgSource,
    } = window.PolyvLiveWatchSDK;

    /** 全局变量 */
    var globalStore = {
      watchCore: null, // 观看页核心实例
      playerInited: false, // 播放器是否初始化完成
      playStatus: false, // 播放状态
      liveStatus: LiveStatus.End, // 直播流状态
      chatConnected: false, // 是否连接聊天室
      chatHistoryData: [], // 聊天历史数据
    };

    /**
     * 绑定播放器 UI 事件
     */
    function bindPlayerUIEvent(watchCore) {
      const $playBtn = document.querySelector('.demo-player-control__play-btn');
      const $pauseBtn = document.querySelector('.demo-player-control__pause-btn');

      $playBtn.addEventListener('click', () => {
        if (!globalStore.playerInited) return;

        watchCore.player.play();
      });

      $pauseBtn.addEventListener('click', () => {
        if (!globalStore.playerInited) return;

        watchCore.player.pause();
      });
```

```
}

/**
 * 绑定聊天室 UI 事件
 */
function bindChatUIEvent(watchCore) {
  const $input = document.querySelector('.demo-chat-input-control__input');
  const $sendBtn = document.querySelector('.demo-chat-input-control__send-btn');
  const $customSendBtn = document.querySelector('.demo-chat-input-control__send-btn--custom');
  const $sendLike = document.querySelector('.demo-chat-input-control__send-like');
  const $loadPrevBtn = document.querySelector('.demo-chat-input-control__load-prev-btn');
  const $loadNextBtn = document.querySelector('.demo-chat-input-control__load-next-btn');

  $sendBtn.addEventListener('click', () => {
    if (!globalStore.chatConnected) return;

    const content = $input.value.trim();
    if (!content) return;

    $input.value = '';
    // 发送文本消息
    watchCore.chat.sendSpeakMsg({
      content,
    });
  });

  $customSendBtn.addEventListener('click', () => {
    if (!globalStore.chatConnected) return;

    const content = $input.value.trim();
    if (!content) return;

    $input.value = '';
    // 发送自定义消息
    watchCore.chat.sendCustomMessage({
      data: { content },
    });
  });

  $sendLike.addEventListener('click', () => {
    if (!globalStore.chatConnected) return;

    // 发送点赞消息
    // 注意：此处仅作示例，真实场景请使用节流机制，避免高并发
    watchCore.chat.sendLike(1);
  });

  // 加载上一页数据
  $loadPrevBtn.addEventListener('click', loadPrevChatHistory);

  // 加载下一页数据
  $loadNextBtn.addEventListener('click', loadNextChatHistory);
}

/**
 * 绑定 UI 事件
```

```
 */
function bindUIEvent(watchCore) {
  bindPlayerUIEvent(watchCore);
  bindChatUIEvent(watchCore);
}

/**
 * 绑定 watch-sdk 中和 player-module 相关的事件
 */
function bindPlvPlayerModuleEvents(watchCore) {
  // 监听播放器-初始化事件
  watchCore.player.eventEmitter.on(PlayerEvents.PlayerInited, () => {
    const playerInfo = watchCore.player.getPlayerInfo();
    globalStore.playerInited = playerInfo.playerInited;

    console.warn('播放器初始化完成');
    console.log('当前是否使用无延迟播放器：', playerInfo.isLowLatency);
  });

  // 监听播放器-播放信息变化
  watchCore.player.eventEmitter.on(PlayerEvents.PlayerInfoChange, ({ playerInfo }) => {
    globalStore.playStatus = playerInfo.playStatus;
  });

  // 监听播放器-播放时间改变
  watchCore.player.eventEmitter.on(PlayerEvents.TimeUpdate, params => {
    if (getChatReplayEnabled(watchCore)) {
      // 参考文档：https://help.polyv.net/#/live/js/new_sdk/live_watch_sdk/articles/modules/chat/chat-
replay?id=_25-%e5%a4%84%e7%90%86%e8%81%8a%e5%a4%a9%e6%b6%88%e6%81%af%e9%87%8d%e6%94%be
      watchCore.chat.processChatMsgReplay(params);
    }
  });

  // 监听播放器-播放事件
  watchCore.player.eventEmitter.on(PlayerEvents.PlayerPlaying, () => {
    console.warn('播放回调，播放视频中');
  });

  // 监听播放器-暂停事件
  watchCore.player.eventEmitter.on(PlayerEvents.PlayerPause, () => {
    console.warn('暂停回调，已暂停视频播放');
  });
}

/**
 * 绑定 watch-sdk 中和 chat-module 相关的事件
 */
function bindPlvChatModuleEvents(watchCore) {
  // 监听聊天室-超出直播间最大在线人数
  watchCore.chat.eventEmitter.on(ChatEvents.OverMaxOnlineCount, () => {
    alert('已超出直播间最大在线人数 ！ ');
  });

  // 监听聊天室-点赞数回调
  watchCore.chat.eventEmitter.on(ChatEvents.ChatLikeCountChange, data => {
    console.log('实时点赞数：', data.realtimeLikes);
```

```
    $likeCount = document.querySelector('#like-count');
    $likeCount.innerText = data.realtimeLikes;
  });

  // 监听聊天室-聊天消息事件
  watchCore.chat.eventEmitter.on(ChatEvents.ChatMessage, ({ chatMsg }) => {
    if (
      chatMsg.msgSource === ChatMsgSource.CustomerMessage ||
      chatMsg.msgSource === ChatMsgSource.CustomMessage
    ) {
      console.warn('自定义聊天消息回调：', chatMsg);
    } else {
      console.warn('普通聊天消息回调：', chatMsg);
    }

    renderMsgList([chatMsg]);
  });

  // 监听聊天室-聊天消息重放-重放状态变更
  watchCore.chat.eventEmitter.on(ChatEvents.ChatMsgReplayStatusChange, params => {
    console.log('聊天重放状态变更：', params.status);
    if (params.status === ChatMsgReplayStatus.Wait) {
      console.warn('聊天重放数据准备中');
    }

    if (params.status === ChatMsgReplayStatus.InitSuccess) {
      console.warn('正在聊天重放中');
    }
  });

  //监听聊天室-聊天消息重放-重新加载
  watchCore.chat.eventEmitter.on(ChatEvents.ChatMsgReplayReload, () => {
    console.warn('聊天消息重放-重新加载');
    // 重新渲染聊天消息列表
    initChatMsgRender();
  });
}

/**
 * 绑定 watch-sdk 事件
 */
function bindWatchSdkEvent(watchCore) {
  // 监听核心 - setup 钩子
  watchCore.eventEmitter.on(PolyvWatchCoreEvents.WatchCoreSetuped, () => {
    console.info('观看页核心实例安装完成');
  });

  // 监听核心 - 连接聊天室
  watchCore.eventEmitter.on(PolyvWatchCoreEvents.WatchCoreConnected, () => {
    console.info('观看页核心实例连接聊天室成功');
    globalStore.chatConnected = true;
  });

  // 监听直播流变化
  watchCore.channel.eventEmitter.on(ChannelEvents.LiveStatusChange, ({ liveStatus }) => {
    if (globalStore.liveStatus === liveStatus) return;
```

```
    globalStore.liveStatus = liveStatus;
    console.warn('流状态变化', liveStatus);

    if (liveStatus === LiveStatus.Live) {
      console.warn('开始直播，请自行改成获取正常聊天历史记录的逻辑后，再重新加载播放器和聊天室');
      return;
    }

    // 直播流变化时重新创建播放器
    if (globalStore.playerInited) {
      globalStore.playerInited = false;
      createPlayer(watchCore);
    }
  });

  bindPlvPlayerModuleEvents(watchCore);
  bindPlvChatModuleEvents(watchCore);
}

/**
 * 绑定事件
 */
function bindEvent(watchCore) {
  bindUIEvent(watchCore);
  bindWatchSdkEvent(watchCore);
}

/**
 * 安装核心实例和授权
 */
async function setupWatchCore(watchCore) {
  await watchCore.setup();

  if (!watchCore.auth.isAuthorized()) {
    // 观众点击观看页入口或执行观看条件授权，如：无条件授权
    const result = await watchCore.auth.verifyNoneAuth();
    if (!result.success) {
      console.error('授权失败了！！', result.failReason);
      return;
    }

    // 当观众执行观看条件授权成功，需要重新安装 watchCore
    await watchCore.setup();
  }
}

/**
 * 获取后台是否启用聊天回放
 */
function getChatReplayEnabled(watchCore) {
  const playbackTarget = watchCore.playback.getCurrentPlaybackTarget();
  if (!playbackTarget) return false;

  return playbackTarget.chatReplayEnabled;
}
```

```
      /**
       * 创建播放器
       */
      async function createPlayer(watchCore) {
        // 获取回放选项，此次仅以单个回放为例
        function __getPlayerPlaybackOptions() {
          const liveStatus = watchCore.channel.getLiveStatus();

          if (liveStatus !== LiveStatus.Playback) return;

          const playbackTarget = watchCore.playback.getCurrentPlaybackTarget();
          if (!playbackTarget) return;

          return playbackTarget.playbackOptions;
        }

        // 具体文档参数见：
https://help.polyv.net/index.html#/live/miniprogram/live_watch_miniprogram_sdk/articles/modules/player/player?
id=classmethoddoc_playermodule_setupplayer
        await watchCore.player.setupPlayer({
          container: '#player',
          lowLatency: false, // 是否无延迟播放
          showController: true, // 是否显示播放器控制栏
          playbackOptions: __getPlayerPlaybackOptions(), // 回放参数
        });
      }

      /**
       * 渲染聊天列表
       * @warn 此处简单的示例，真实场景请做好虚拟列表来处理高并发场景
       */
      function renderMsgList(data, reset = false) {
        const $msgList = document.querySelector('.demo-chat__list');
        const fragment = document.createDocumentFragment();
        data
          // 此处只处理了发言类型的消息
          .filter(chatMsg => chatMsg.msgSource === ChatMsgSource.Speak)
          .forEach(chatMsg => {
            const li = document.createElement('li');
            li.textContent = `${chatMsg.user.nick}： ${chatMsg.content || ''}`;
            fragment.appendChild(li);
          });

        if (reset) {
          $msgList.innerHTML = '';
        }

        $msgList.appendChild(fragment);
      }

      /**
       * 请求聊天重放历史记录
       */
      async function requestChatHistory(reqMode) {
        const { watchCore, chatHistoryData } = globalStore;
```

```
  // 此处仅作示例
  // 如果要获取下一批历史记录，一般是获取当前历史记录的最后一条消息，然后请求下一批数据
  // 如果要获取上一批历史记录，一般是获取当前历史记录的第一条消息，然后请求上一批数据
  const reMsg = (() => {
    const historyData = chatHistoryData;
    if (reqMode === ChatReplayRequestHistoryMode.Next) {
      return historyData[historyData.length - 1];
    }

    if (reqMode === ChatReplayRequestHistoryMode.Pre) {
      return historyData[0];
    }

    return undefined;
  })();

  const data = await watchCore.chat.getChatMsgReplayHistoryList({
    mode: reqMode,
    replayMsg: reMsg,
    count: 20,
  });

  return data;
}

/**
 * 加载上一页历史聊天消息
 */
async function loadPrevChatHistory() {
  const historyLength = globalStore.chatHistoryData.length;
  const watchCore = globalStore.watchCore;
  if (historyLength === 0 || !watchCore) return;

  const result = await requestChatHistory(ChatReplayRequestHistoryMode.Pre);

  console.warn('加载上一页历史聊天消息：', result);

  // globalStore.chatHistoryData = result.concat(globalStore.chatHistoryData);
  // renderMsgList(globalStore.chatHistoryData, true);
}

/**
 * 加载下一页历史聊天消息
 */
async function loadNextChatHistory() {
  const historyLength = globalStore.chatHistoryData.length;
  const watchCore = globalStore.watchCore;
  if (historyLength === 0 || !watchCore) return;

  const result = await requestChatHistory(ChatReplayRequestHistoryMode.Next);

  console.warn('加载下一页历史聊天消息：', result);

  // globalStore.chatHistoryData = globalStore.chatHistoryData.concat(result);
  // renderMsgList(result);
```

```
}

/**
 * 初始化聊天消息渲染
 */
async function initChatMsgRender() {
  // 获取聊天重放中的历史记录
  const historyData = await requestChatHistory(ChatReplayRequestHistoryMode.Init);

  globalStore.chatHistoryData = historyData;

  renderMsgList(historyData, true);
}

/**
 * 创建聊天列表
 */
async function createChatList(watchCore) {
  if (!getChatReplayEnabled(watchCore)) {
    console.warn('请使用一个开启聊天重放的频道');
    return;
  }

  console.warn('createChatList');
  await watchCore.chat.initChatMsgReplay();
  await initChatMsgRender();
}

/**
 * 设置当前回放对象
 */
function setCurrentPlaybackTarget(watchCore) {
  const liveStatus = watchCore.channel.getLiveStatus();
  if (liveStatus !== LiveStatus.Playback) return;

  const playbackTarget = watchCore.playback.getSinglePlaybackTarget();
  watchCore.playback.setCurrentPlaybackTarget(playbackTarget);
}

/**
 * 入口函数
 */
async function main() {
  // 创建核心，文档见：
https://help.polyv.net/index.html#/live/miniprogram/live_watch_miniprogram_sdk/articles/core/sdk-core
  const watchCore = createWatchCore({
    channelId: 'xxx', // 请使用一个"公开观看"并开启"聊天重放"的频道
    userInfo: {
      // 用户信息
      userId: 'polyv-123456',
      nick: 'polyv-测试昵称',
      actor: 'polyv-测试头衔',
    },
  });

  globalStore.watchCore = watchCore;
```

```
        // 绑定相关事件
        bindEvent(watchCore);
        // 安装核心
        await setupWatchCore(watchCore);

        // 连接聊天室
        watchCore.connect();

        // 设置当前回放对象，注意，在创建播放器和聊天列表前需要先设置当前的回放对象，确保后面流程正常
        setCurrentPlaybackTarget(watchCore);
        // 创建播放器
        createPlayer(watchCore);
        // 创建聊天列表
        createChatList(watchCore);
      }

      main();
    </script>
  </body>
</html>
```