即时通信 IM

插件市场

产品文档

腾讯云

文档目录

插件市场
云端搜索插件
效果展示

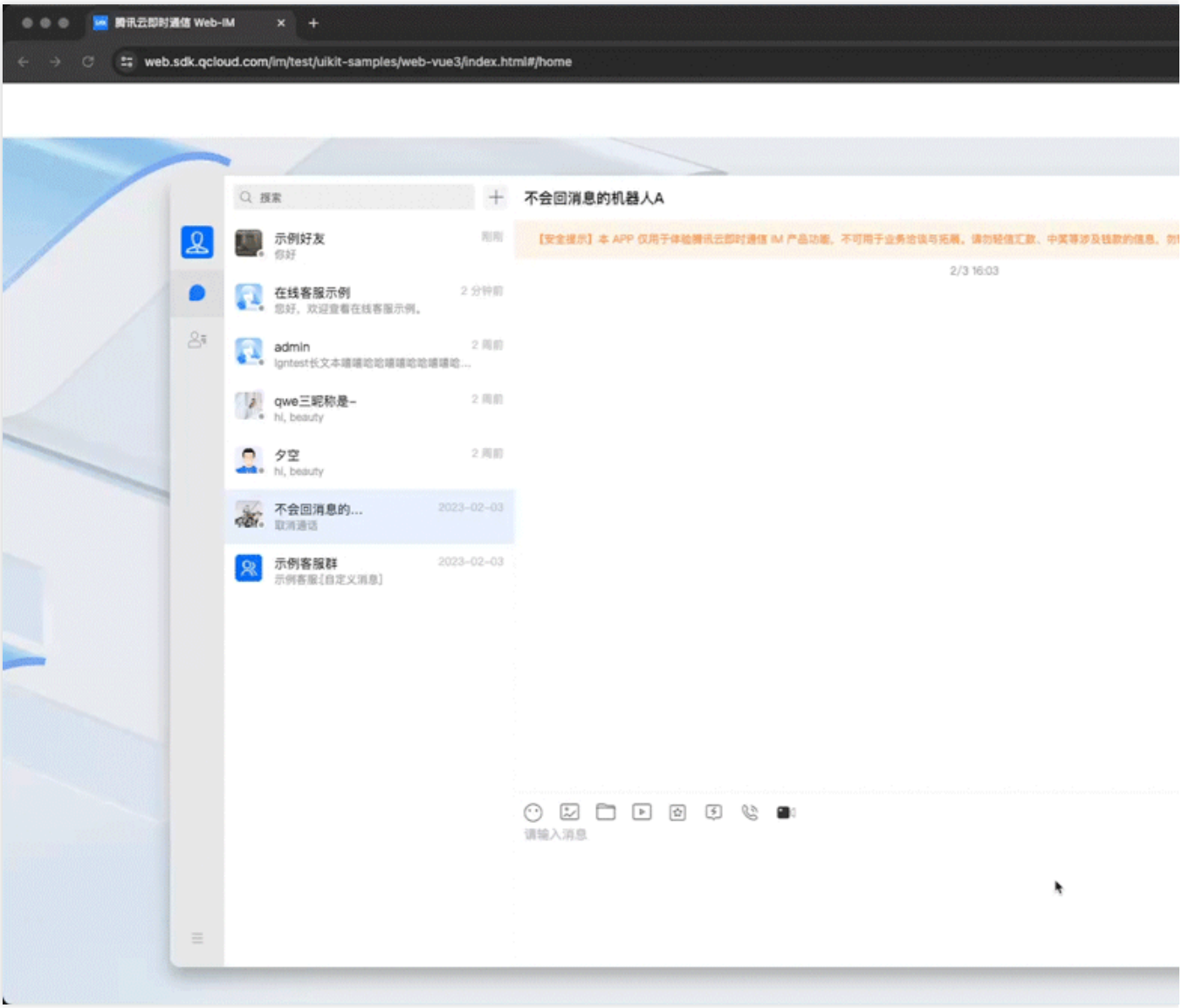
最近更新时间：2024-04-23 15:40:55

功能体验

使用场景

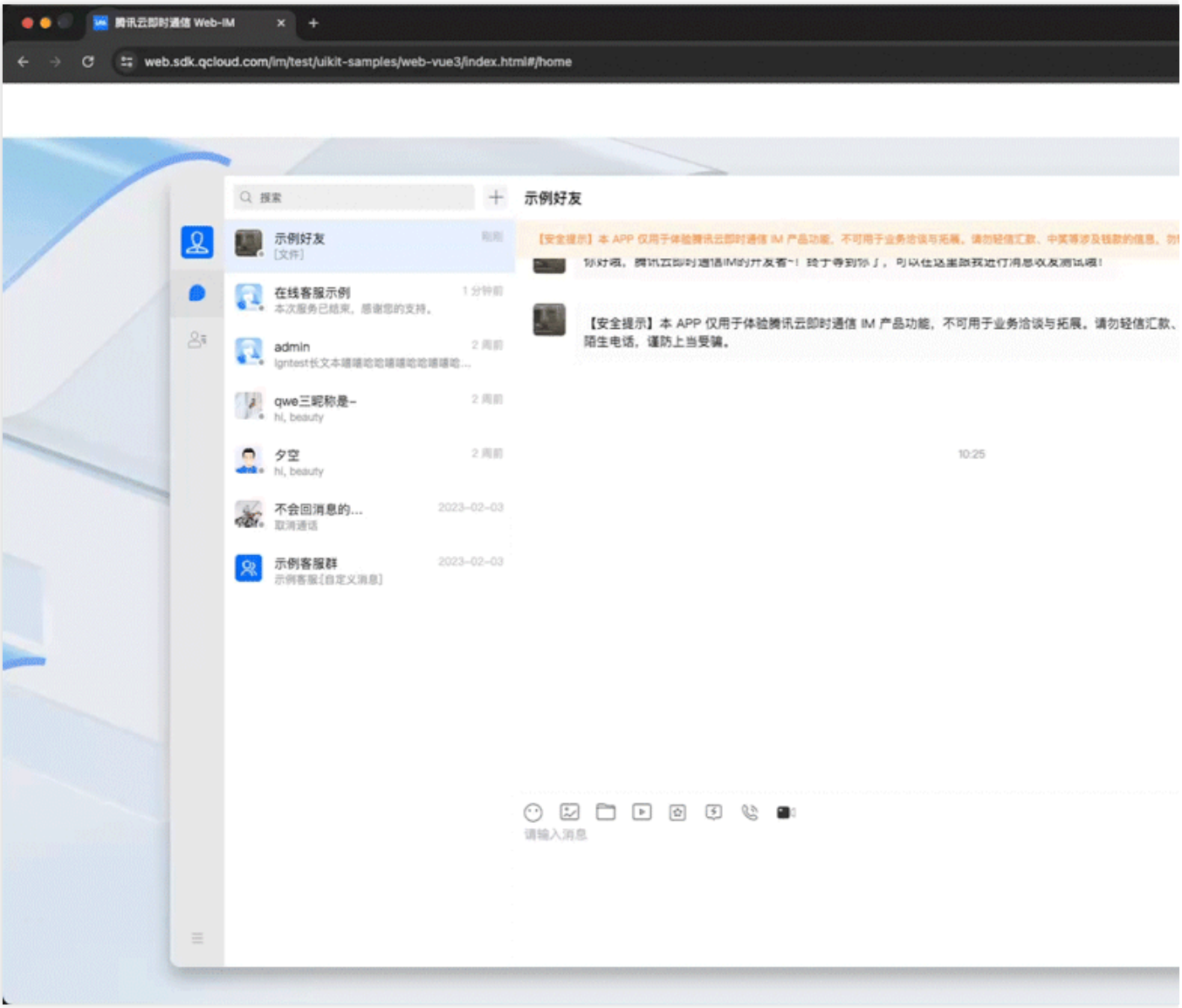
场景一：全局搜索

通过消息搜索功能，用户可全局搜索所有包含指定关键词的会话，还可在搜索栏中自由选择搜索的时间范围。用户单击搜索到的会话，可以直接定位至对应聊天位置，并高亮显示该消息。操作演示如下：



场景二：指定会话内搜索文件/图片/视频/其他

通过消息搜索功能，用户可直接搜索指定会话内的全部文件/图片/视频/其他消息，还可在指定的时间段内搜索。用户点击搜索到的文件/图片/视频/其他，同样可以直接定位至对应聊天位置，并高亮显示该文件/图片/视频/其他消息。以下演示以文件为例：



集成指引

Web&小程序&uni-app：集成指引

Android&iOS&Windows&Mac：集成指引

集成指引
Android&iOS&Windows&Mac

最近更新时间：2024-04-30 15:59:03

功能描述

消息云端搜索，提升 App 使用体验必备功能，可以帮助用户从纷繁复杂的信息中直接找到预期内容，快捷方便；也可以作为运营工具，增加相关内容的引导，简洁高效。

说明：

消息云端搜索功能仅 7.3.4358 及以上版本支持。

该功能为增值功能，详见价格说明。

如果您没有开通该服务，调用接口会返回 60020 错误码。

消息搜索类介绍

消息搜索参数类

消息搜索参数类为 `V2TIMMessageSearchParam`（Android / iOS & Mac / Windows）。搜索消息时，SDK 会按照该对象的设置，执行不同的搜索逻辑。

`V2TIMMessageSearchParam` 的参数说明如下：

参数	含义	说明
keywordList	关键字列表	最多支持 5 个。当消息发送者以及消息类型均未指定时，必须设置关键字列表；否则，关键字列表可以为空。
keywordListMatchType	指定关键字列表匹配类型	可设置为“或”关系搜索，或“与”关系搜索。取值分别为 `V2TIM_KEYWORD_LIST_MATCH_TYPE_OR` 和 `V2TIM_KEYWORD_LIST_MATCH_TYPE_AND`。默认为“或”关系搜索。
senderUserIDList	指定 userID 发送的消息	最多支持 5 个。
messageTypeList	指定搜索的消息类型集合	传空表示搜索支持的全部类型消息（`V2TIMFaceElem` 和 `V2TIMGroupTipsElem` 不支持搜索）。其他类型取值参考 `V2TIMElemType`（Android / iOS & Mac / Windows）。

conversationID	搜索 “全部会话” 还是搜索 “指定的会话”	`conversationID` 为空，搜索全部会话；`conversationID` 不为空，搜索指定会话。
searchTimePosition	搜索的起始时间点	默认为 0（从现在开始搜索）。UTC 时间戳，单位：秒。
searchTimePeriod	从起始时间点开始的过去时间范围	默认为 0（不限制时间范围）。24 x 60 x 60 代表过去一天，单位：秒。
searchCount	搜索的数量	搜索的数量，最大支持100。
searchCursor	搜索的游标	搜索的起始位置，第一次填写空字符串，续拉时填写上一次返回的 `V2TIMMessageSearchResult` 中的 `searchCursor` 。

消息搜索结果类

消息搜索结果类为 `V2TIMMessageSearchResult` （Android / iOS & Mac / Windows）。参数说明如下：

参数	含义	说明
totalCount	搜索结果总数	如果搜索指定会话，返回满足搜索条件的消息总数； 如果搜索全部会话，返回满足搜索条件的消息所在的所有会话总数量。
messageSearchResultItems	搜索结果列表	如果搜索指定会话，返回结果列表只包含该会话结果； 如果搜索全部会话，会对满足搜索条件的消息根据会话 ID 分组，分页返回分组结果。
searchCursor	续拉的游标	调用搜索接口续拉时需要填的游标

其中 `messageSearchResultItems` 是个列表，内含 `V2TIMMessageSearchResultItem` （Android / iOS & Mac / Windows）对象，参数说明如下：

参数	含义	说明
conversationID	会话 ID	-
messageCount	消息数量	当前会话一共搜索到了多少条符合要求的消息。
messageList	满足搜索条件的消息列表	如果搜索指定会话， `messageList` 中装载的是本会话中所有满足搜索条件的消息列表。 如果搜索全部会话， `messageList` 中装载的消息条数会有如下两种可能： 如果某个会话中匹配到的消息条数 > 1，则 `messageList` 为空，您可以在 UI 上显示 “{ `messageCount` } 条相关记录”。

		如果某个会话中匹配到的消息条数 = 1，则 `messageList` 为匹配到的那条消息，您可以在 UI 上显示之，并高亮匹配关键词。

搜索全部会话的消息

当用户在搜索框输入关键字搜索消息时，您可以调用 `searchCloudMessages` （Android / iOS & Mac / Windows）搜索消息。

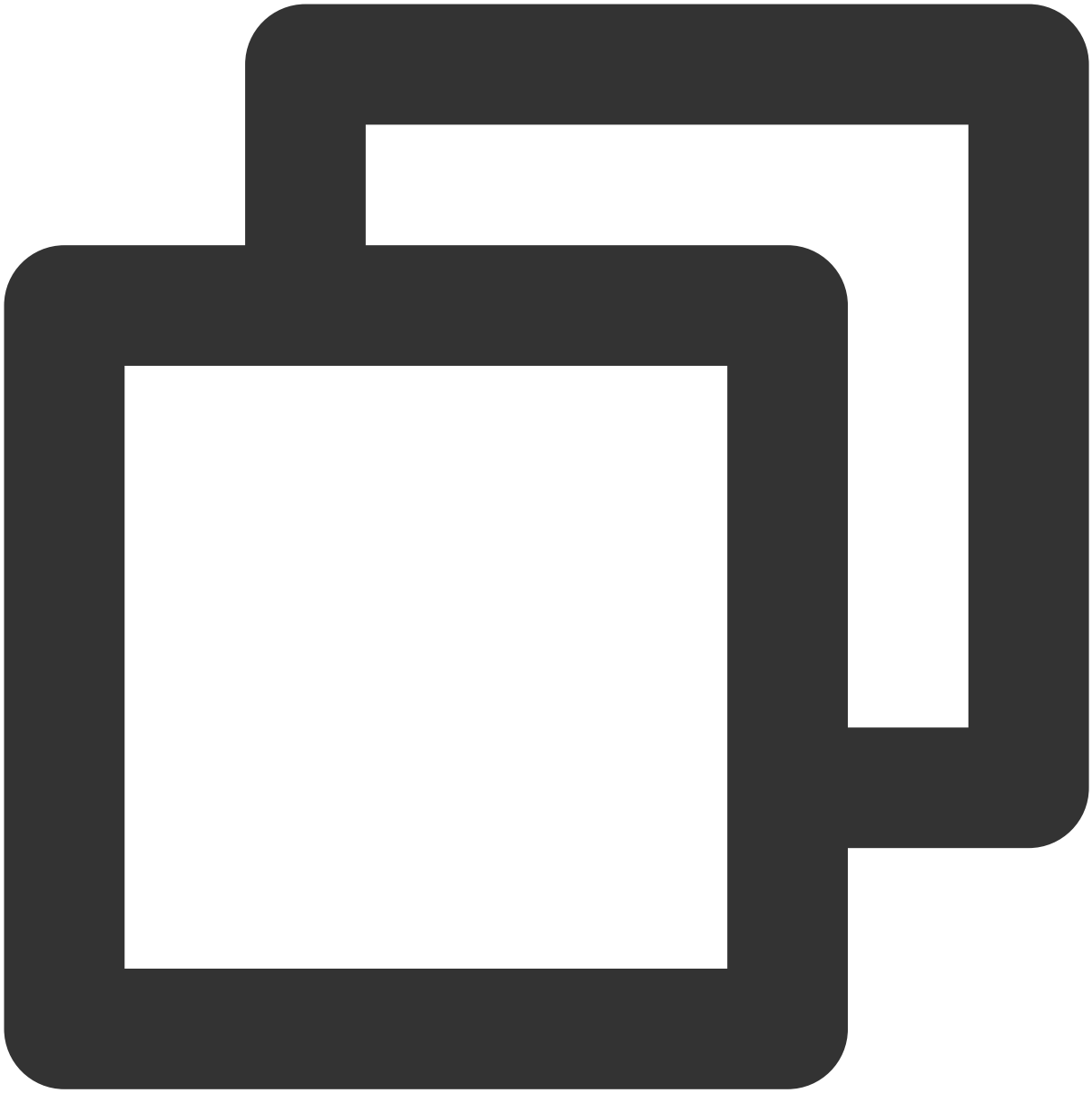
如果您希望在全部会话范围内搜索，只需要将 `V2TIMMessageSearchParam` 中的 `conversationID` 设置为空或者不设置即可。

示例代码如下：

Android

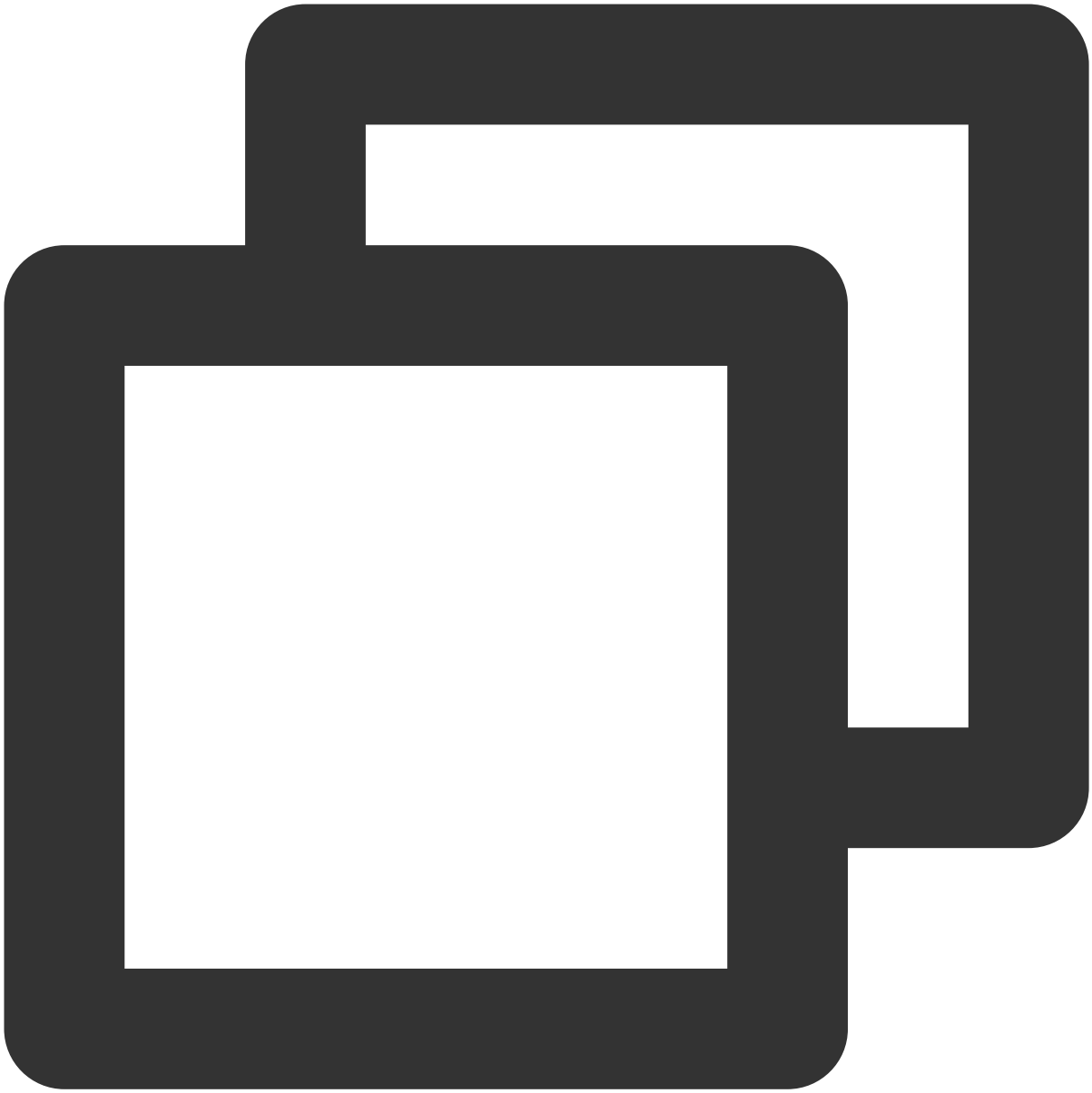
iOS & Mac

Windows



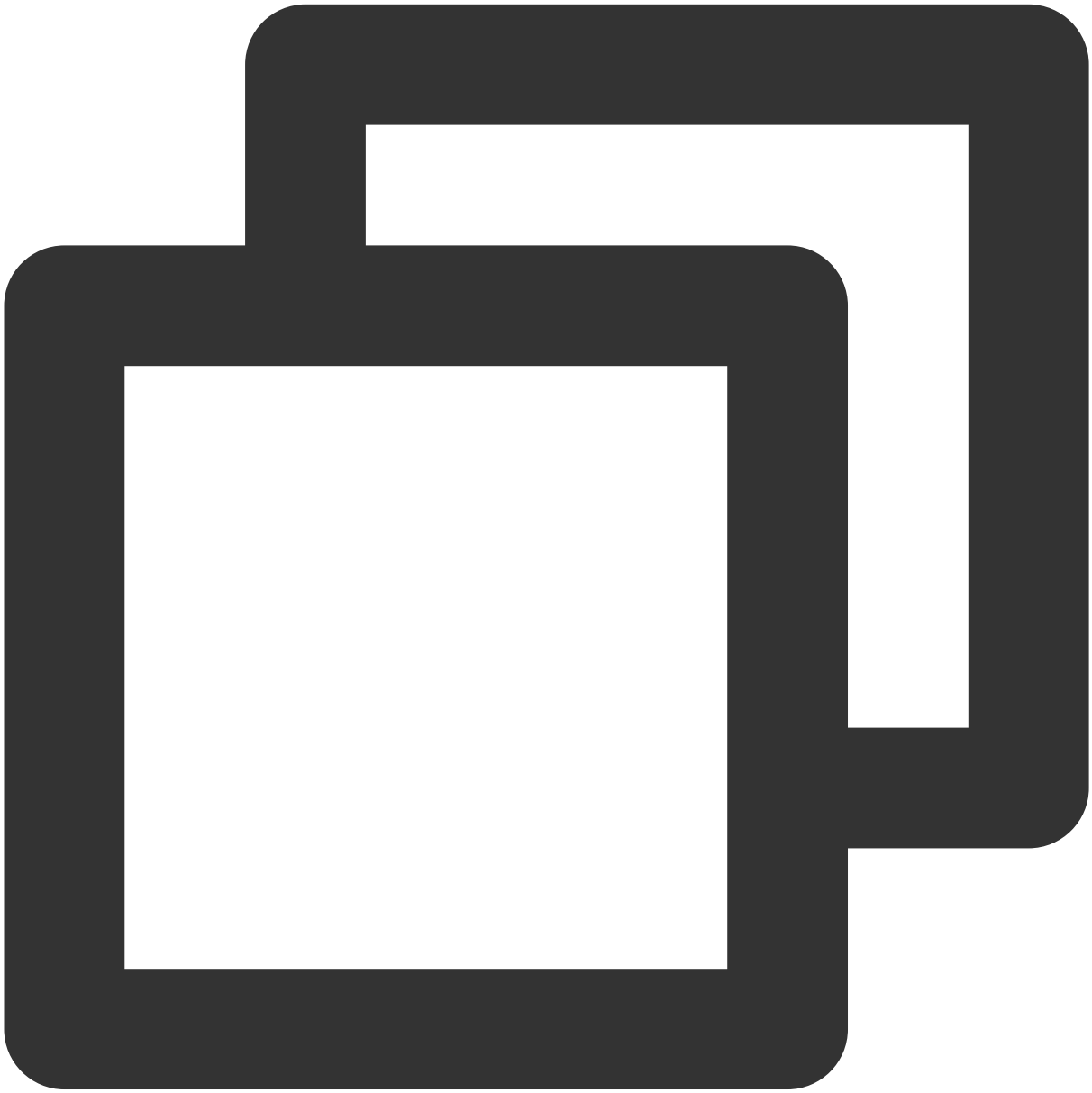
```
List<String> keywordList = new ArrayList<>();
keywordList.add("abc");
keywordList.add("123");
V2TIMMessageSearchParam searchParam = new V2TIMMessageSearchParam();
// 设置搜索关键字
searchParam.setKeywordList(keywordList);
// 搜索20条数据
searchParam.setSearchCount(20);
// 从最新的会话开始搜索
searchParam.setSearchCursor("");
// 从当前时间开始搜索
```

```
searchParam.setSearchTimePosition(0);
// 搜索10分钟之内的消息
searchParam.setSearchTimePeriod(600);
V2TIMManager.getMessageManager().searchCloudMessages(searchParam,newV2TIMValueCallb
    @Override
    public void onSuccess(V2TIMMessageSearchResult v2TIMMessageSearchResult) {
      // 搜索成功
    }

    @Override
    public void onError(int code, String desc) {
      // 搜索失败
    }
});
```



```
V2TIMMessageSearchParam *param = [[V2TIMMessageSearchParam alloc] init];
// 设置搜索关键字
param.keywordList = @[@"abc", @"123"];
param.messageTypeList = nil;
param.conversationID = nil;
param.searchTimePosition = 0;
param.searchTimePeriod = 0;
// 搜索20条数据
param.searchCount = 20;
// 从最新的会话开始搜索
param.searchCursor = @"";
```

```
[V2TIMManager.sharedInstance searchCloudMessages:param
succ:^(V2TIMMessageSearchResult *searchResult) {
    // 搜索成功，searchResult 中返回搜索结果
} fail:^(int code, NSString *desc) {
    // 搜索失败
}];
```



```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
    using SuccessCallback = std::function<void(const T&)>;
```

```
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
      success_callback_ = std::move(success_callback);
      error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
      if (success_callback_) {
        success_callback_(value);
      }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
      if (error_callback_) {
        error_callback_(error_code, error_message);
      }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

V2TIMMessageSearchParam searchParam;
// 设置搜索关键字
searchParam.keywordList.PushBack("abc");
searchParam.keywordList.PushBack("123");
// 搜索20条数据
searchParam.searchCount = 20;
// 从最新的会话开始搜索
searchParam.searchCursor = "";

auto callback = new ValueCallback<V2TIMMessageSearchResult>{};
callback->SetCallback(
    [=](const V2TIMMessageSearchResult& messageSearchResult) {
      // 搜索成功
      delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
      // 搜索失败
      delete callback;
    });

V2TIMManager::GetInstance()->GetMessageManager()->SearchCloudMessages(searchParam,
```

搜索指定会话的消息

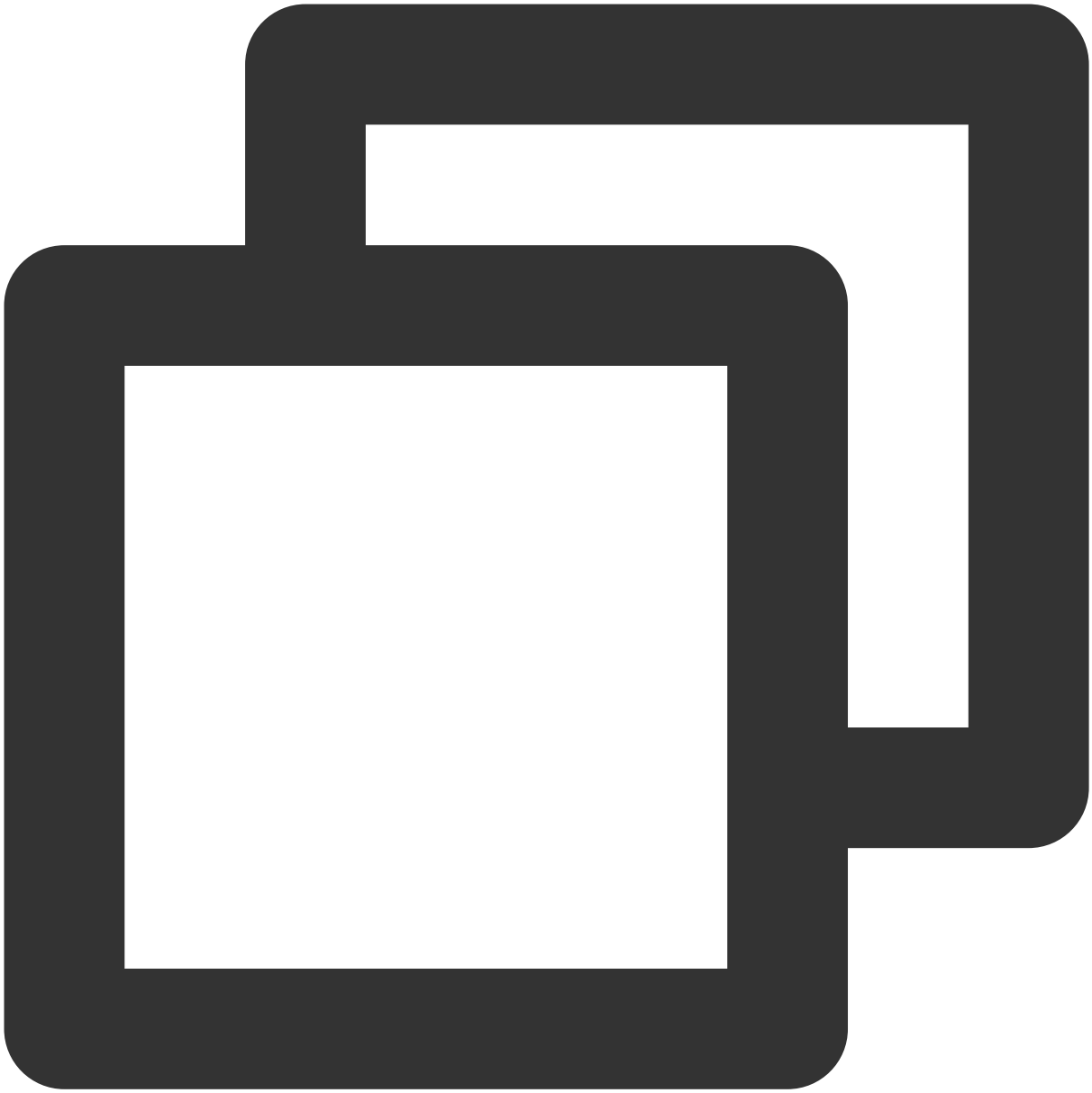
当用户在搜索框输入关键字搜索消息时，您可以调用 `searchCloudMessages` （Android / iOS & Mac / Windows）搜索消息。

示例代码如下：

Android

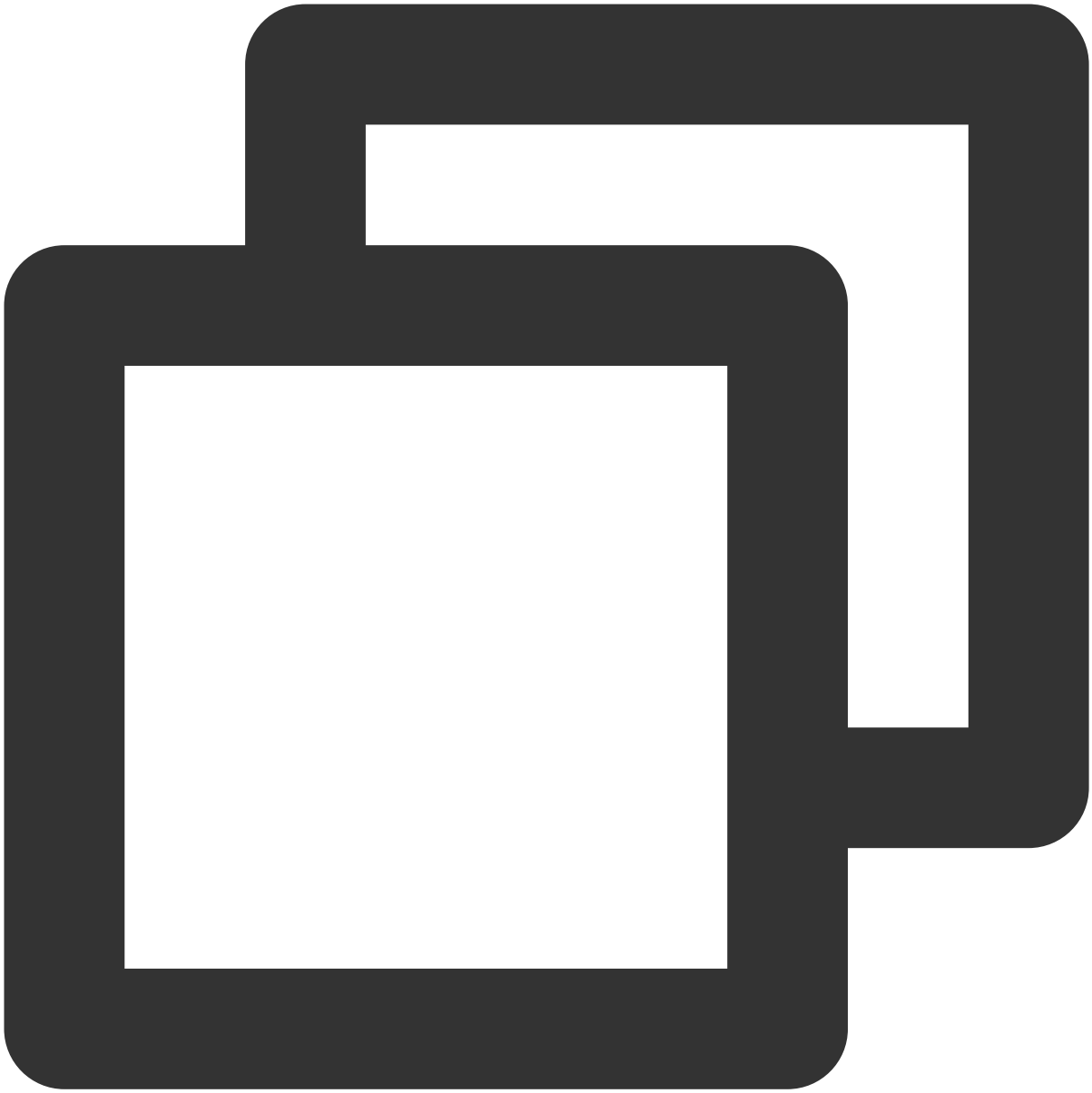
iOS & Mac

Windows



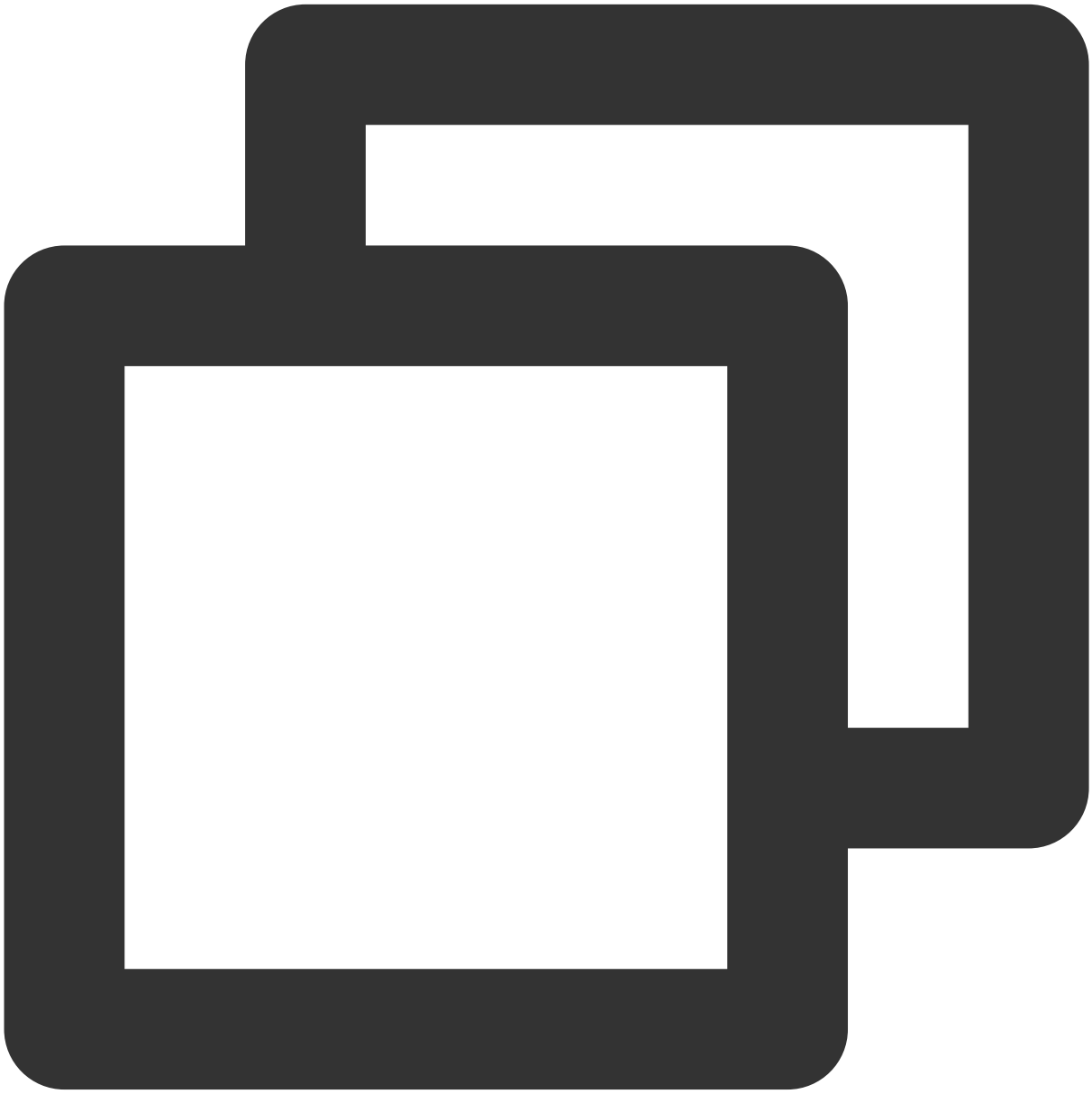
```
List<String> keywordList = new ArrayList<>();
keywordList.add("abc");
keywordList.add("123");
V2TIMMessageSearchParam searchParam = new V2TIMMessageSearchParam();
// 搜索与用户 user1 的单聊消息
searchParam.setConversationID("c2c_user1");
// 设置搜索关键字
searchParam.setKeywordList(keywordList);
// 搜索20条数据
searchParam.setSearchCount(20);
// 从最新的会话开始搜索
```

```
searchParam.setSearchCursor("");
// 从当前时间开始搜索
searchParam.setSearchTimePosition(0);
// 搜索10分钟之内的消息
searchParam.setSearchTimePeriod(600);
V2TIMManager.getMessageManager().searchCloudMessages(searchParam,newV2TIMValueCallb
    @Override
    public void onSuccess(V2TIMMessageSearchResult v2TIMMessageSearchResult) {
      // 搜索成功
    }

    @Override
    public void onError(int code, String desc) {
      // 搜索失败
    }
});
```



```
V2TIMMessageSearchParam *param = [[V2TIMMessageSearchParam alloc] init];
// 设置搜索关键字
param.keywordList = @[@"abc", @"123"];
param.messageTypeList = nil;
// 搜索与用户 user1 的单聊消息
param.conversationID = @"c2c_user1";
param.searchTimePosition = 0;
param.searchTimePeriod = 0;
// 搜索20条数据
param.searchCount = 20;
// 从最新的会话开始搜索
```

```
param.searchCursor = @"";
[V2TIMManager.sharedInstance searchCloudMessages:param
succ:^(V2TIMMessageSearchResult *searchResult) {
    // 搜索成功, searchResult 中返回搜索结果
} fail:^(int code, NSString *desc) {
    // 搜索失败
}];
```



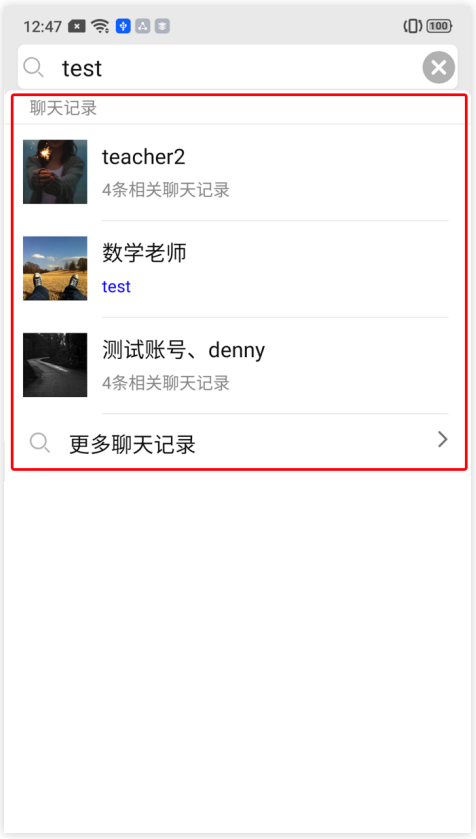
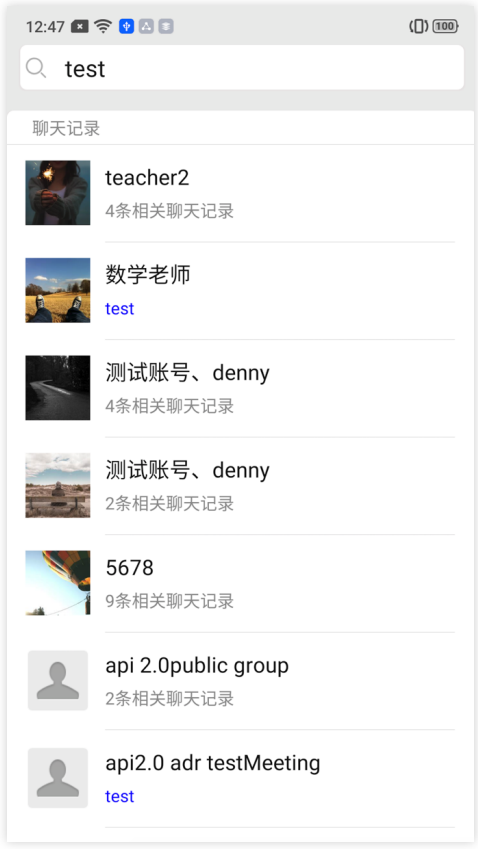
```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
```

```
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
      success_callback_ = std::move(success_callback);
      error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
      if (success_callback_) {
        success_callback_(value);
      }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
      if (error_callback_) {
        error_callback_(error_code, error_message);
      }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

V2TIMMessageSearchParam searchParam;
// 搜索与用户 user1 的单聊消息
searchParam.conversationID = "c2c_user1";
// 设置搜索关键字
searchParam.keywordList.PushBack("abc");
searchParam.keywordList.PushBack("123");
// 搜索20条数据
searchParam.searchCount = 20;
// 从最新的会话开始搜索
searchParam.searchCursor = "";

auto callback = new ValueCallback<V2TIMMessageSearchResult>{};
callback->SetCallback(
    [=](const V2TIMMessageSearchResult& messageSearchResult) {
      // 搜索成功
      delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
      // 搜索失败
      delete callback;
```

```
    });

V2TIMManager::GetInstance()->GetMessageManager()->SearchCloudMessages(searchParam,
```

搜索典型场景示例

普通的 IM 聊天软件，搜索界面的展示通常分这几种场景：

图 1：搜索聊天记录	图 2：搜索更多聊天记录	图
		

下文我们将依次向您展示如何利用 IM SDK 的搜索 API 实现上图的典型场景。

展示最近几个活跃的会话

如图 1 所示，最下方是搜索到的消息所属的最近 3 个会话列表，实现方式如下：

1. 设置搜索参数 `V2TIMMessageSearchParam`

`conversationID` 设置为 `null`，表示搜索所有会话的消息。

`searchCursor` 设置为 ""，表示搜索最新的数据。

`searchCount` 设置为 3，表示返回最近的会话数量，UI 上一般显示 3 条。

2. 处理搜索回调结果 `V2TIMMessageSearchResult`

`totalCount` 表示匹配到的消息所属的所有会话数量。

`messageSearchResultItems` 列表为最近 3（即入参 `searchCount`）个会话信息。其中元素 `V2TIMMessageSearchResultItem` 的 `messageCount` 表示当前会话搜索到的消息总数量；

搜索到的消息条数 > 1，则 `messageList` 为空，您可以在 UI 上显示 “4 条相关聊天记录”，其中的 4 为 `messageCount`。

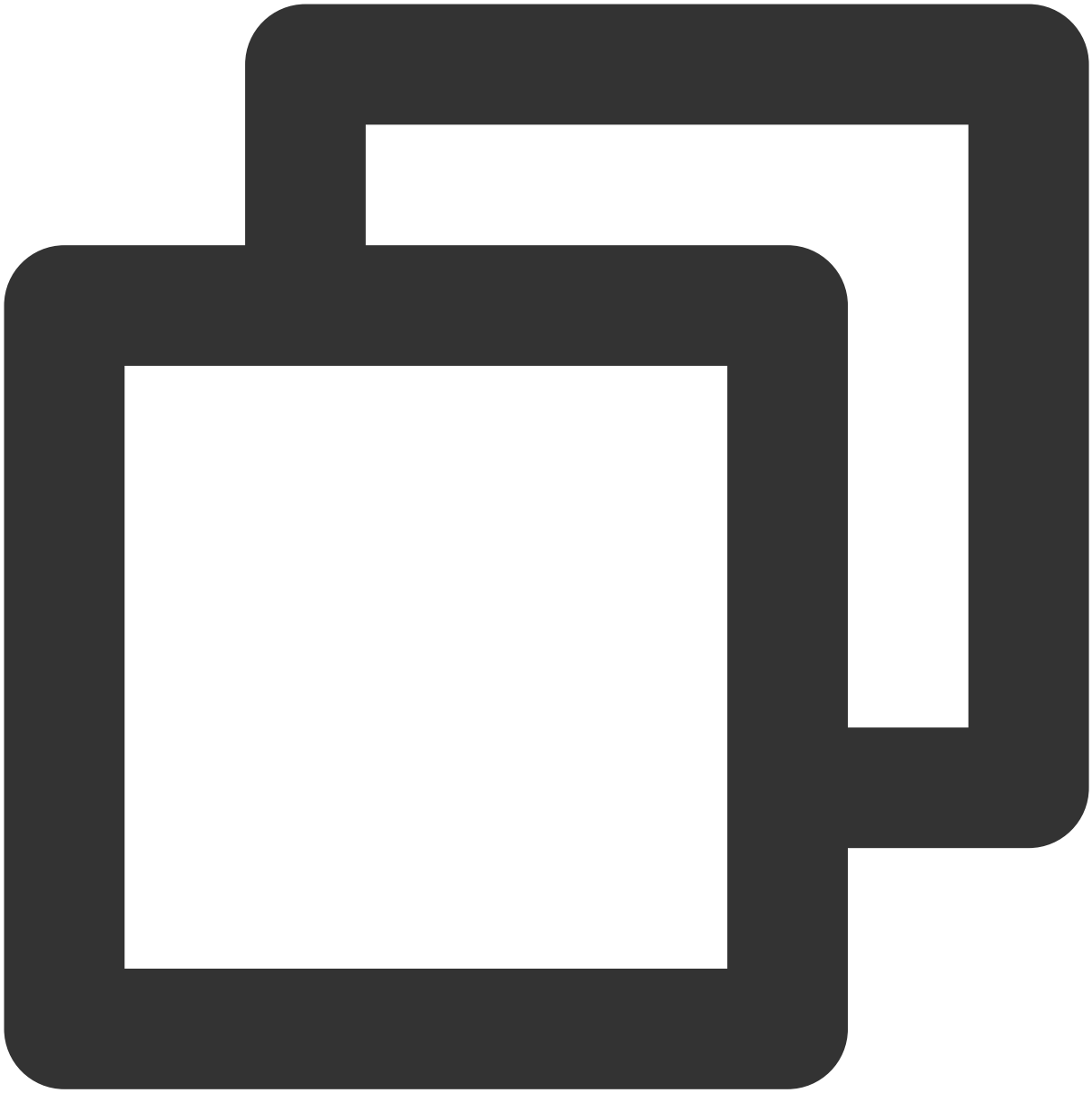
搜索到的消息条数 = 1，则 `messageList` 为匹配到的那条消息，您可以在 UI 上显示消息内容并高亮搜索关键词，例如搜索典型场景示例图中的 “test”。

示例代码如下：

Android

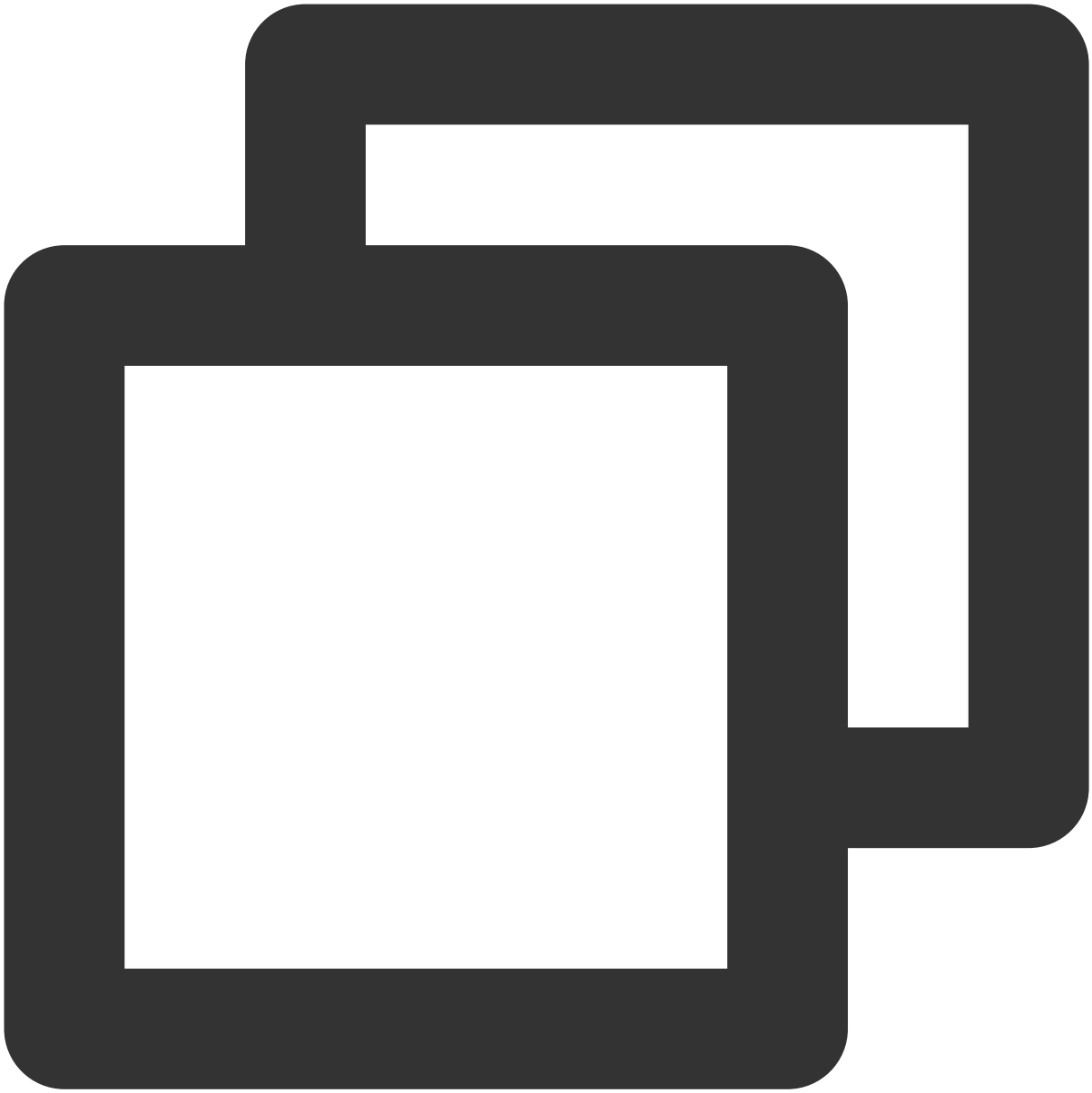
iOS & Mac

Windows



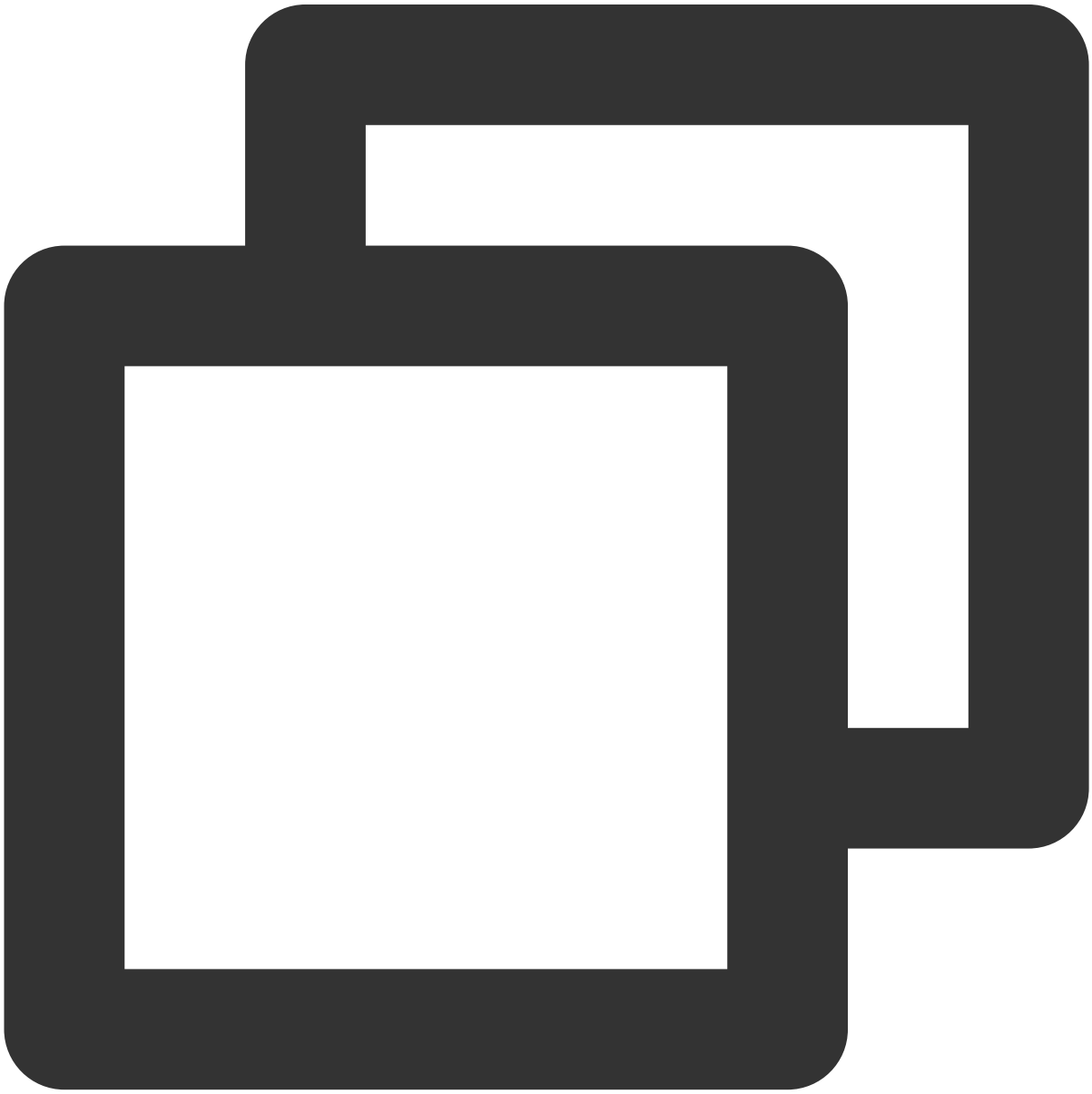
```
List<String> keywordList = new ArrayList<>();
keywordList.add("test");
V2TIMMessageSearchParam v2TIMMessageSearchParam = new V2TIMMessageSearchParam();
// conversationID 设置为 null 表示搜索所有会话中的消息，结果会按照会话分类
v2TIMMessageSearchParam.setConversationID(null);
v2TIMMessageSearchParam.setKeywordList(keywordList);
v2TIMMessageSearchParam.setSearchCursor("");
v2TIMMessageSearchParam.setSearchCount(3);
V2TIMManager.getMessageManager().searchCloudMessages(v2TIMMessageSearchParam, newV2
    @Override
    public void onSuccess(V2TIMMessageSearchResult v2TIMMessageSearchResult) {
```

```
        // 匹配到的消息所属的所有会话数量
        int totalCount = v2TIMMessageSearchResult.getTotalCount();
        // 最近3个根据消息会话分类的信息
        List<V2TIMMessageSearchResultItem> resultItemList = v2TIMMessageSearchResult.
        for (V2TIMMessageSearchResultItem resultItem : resultItemList) {
          // 会话 ID
          String conversationID = resultItem.getConversationID();
          // 该会话匹配到的所有消息数量
          int totalMessageCount = resultItem.getMessageCount();
          // 消息列表：如果 totalMessageCount > 1，该列表为空；如果 totalMessageCount = 1,
          List<V2TIMMessage> v2TIMMessageList = resultItem.getMessageList();
        }
      }

      @Override
      public void onError(int code, String desc) {
      }
});
```



```
V2TIMMessageSearchParam *param = [[V2TIMMessageSearchParam alloc] init];
param.keywordList = @[@"test"];
// conversationID 设置为 nil 表示搜索所有会话中的消息，结果会按照会话分类
param.conversationID = nil;
param.searchCursor = @"";
param.searchCount = 3;
[V2TIMManager.sharedInstance searchCloudMessages:param succ:^(V2TIMMessageSearchRes
    // 匹配到的消息所属的所有会话数量
    NSInteger totalCount = searchResult.totalCount;
    // 最近3个根据消息会话分类的信息
    NSArray<V2TIMMessageSearchResultItem *> *messageSearchResultItems = searchResul
```

```
    for (V2TIMMessageSearchResultItem *searchItem in messageSearchResultItems) {
      // 会话 ID
      NSString *conversationID = searchItem.conversationID;
      // 该会话匹配到的所有消息数量
      NSUInteger messageCount = searchItem.messageCount;
      // 消息列表
      NSArray<V2TIMMessage *> *messageList = searchItem.messageList ?: @[];
    }
} fail:^(int code, NSString *desc) {
    // fail
}];
```



```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
      success_callback_ = std::move(success_callback);
      error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
      if (success_callback_) {
      success_callback_(value);
      }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
      if (error_callback_) {
      error_callback_(error_code, error_message);
      }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

V2TIMMessageSearchParam searchParam;
// conversationID 设置为空表示搜索所有会话中的消息，结果会按照会话分类
searchParam.conversationID = "";
searchParam.keywordList.PushBack("test");
searchParam.searchCursor = "";
searchParam.searchCount = 3;

auto callback = new ValueCallback<V2TIMMessageSearchResult>{};
callback->SetCallback(
    [=](const V2TIMMessageSearchResult& messageSearchResult) {
      // 匹配到的消息所属的所有会话数量
      uint32_t totalCount = messageSearchResult.totalCount;
      // 最近3个根据消息会话分类的信息
      V2TIMMessageSearchResultItemVector messageSearchResultItems =
      messageSearchResult.messageSearchResultItems;
      for (size_t i = 0; i < messageSearchResultItems.Size(); ++i) {
```

```
        // 会话 ID
        V2TIMString conversationID = messageSearchResultItems[i].conversationID;
        // 该会话匹配到的所有消息数量
        uint32_t messageCount = messageSearchResultItems[i].messageCount;
        // 消息列表：如果 messageCount > 1，该列表为空；如果 messageCount = 1，该列表元素为
        V2TIMMessageVector messageList = messageSearchResultItems[i].messageList;
      }
      delete callback;
      },
      [=](int error_code, const V2TIMString& error_message) {
        // 搜索失败
        delete callback;
    });

V2TIMManager::GetInstance()->GetMessageManager()->SearchCloudMessages(searchParam,
```

展示所有搜索到的消息所属会话列表

点击搜索典型场景示例图 1 中的 “更多聊天记录”，会跳转到图 2，展示所有搜索到的消息所属的会话列表。搜索参数和搜索结果描述跟上文的场景类似。

为了防止内存膨胀，我们强烈建议您分页加载会话列表。 举个例子，分页加载，每页展示 10 条会话结果，搜索参数 `V2TIMMessageSearchParam` 可以参考如下设置：

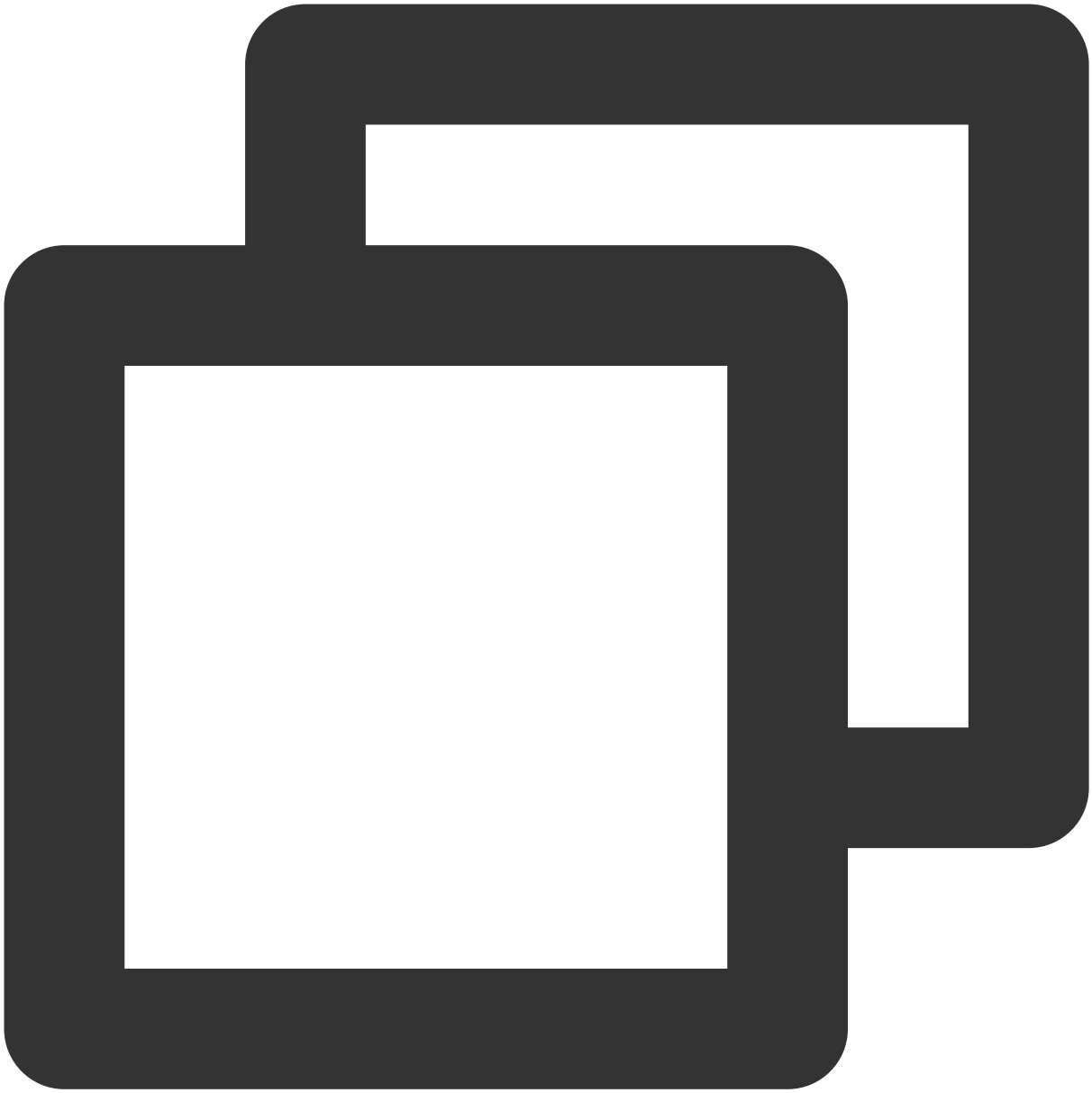
1. 首次调用：设置参数 `searchCount` = 10， `searchCursor` = ""。调用 `searchCloudMessages` 获取消息搜索结果，解析并展示到首页，并且从结果回调中获取会话总数量 `totalCount` 以及下次请求的游标 `searchCursor` 。
2. 当界面滑动快到底部后根据上一次请求结果中的游标 `searchCursor` 继续拉取下一页的数据 。

示例代码如下:

Android

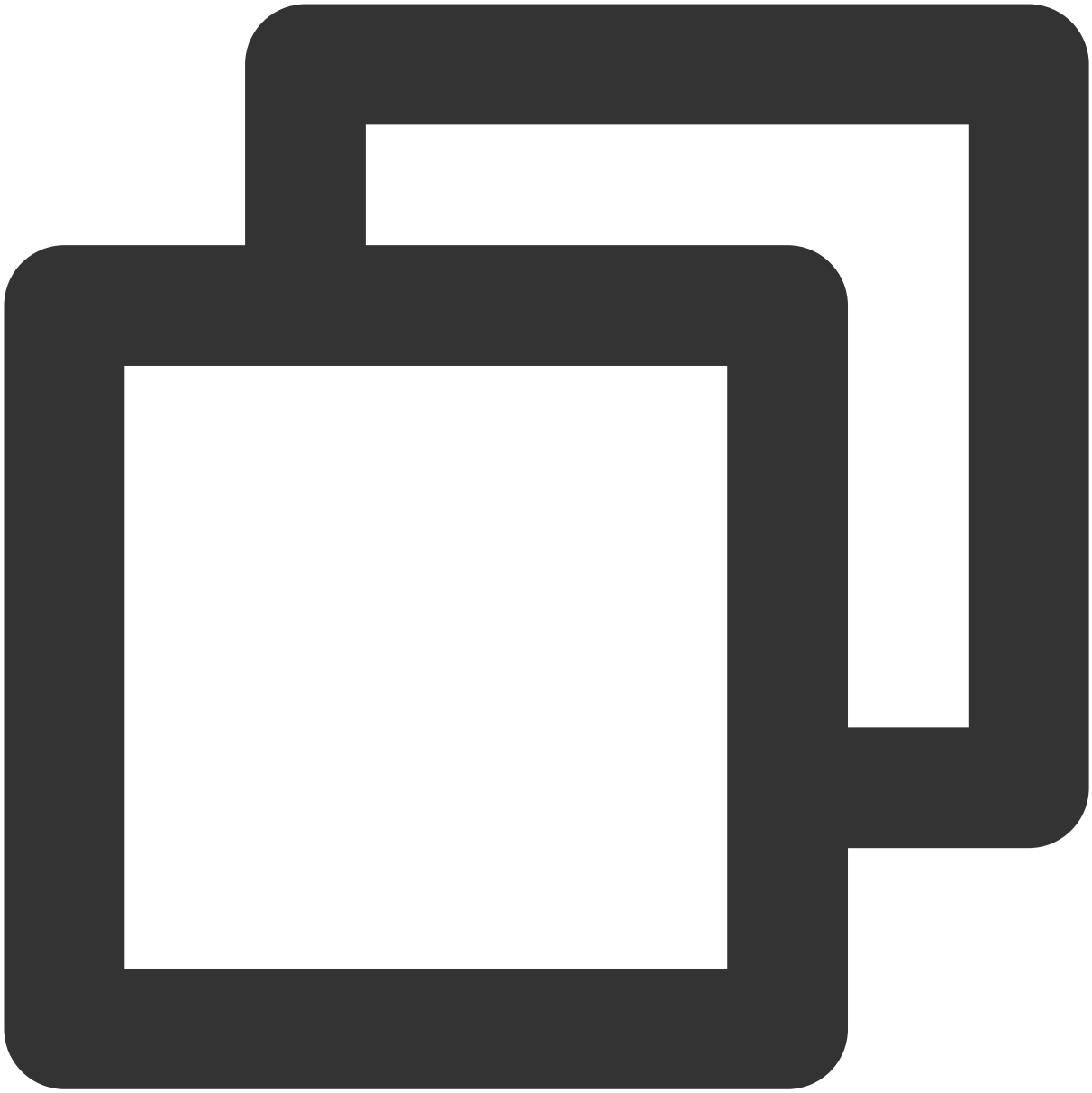
iOS & Mac

Windows



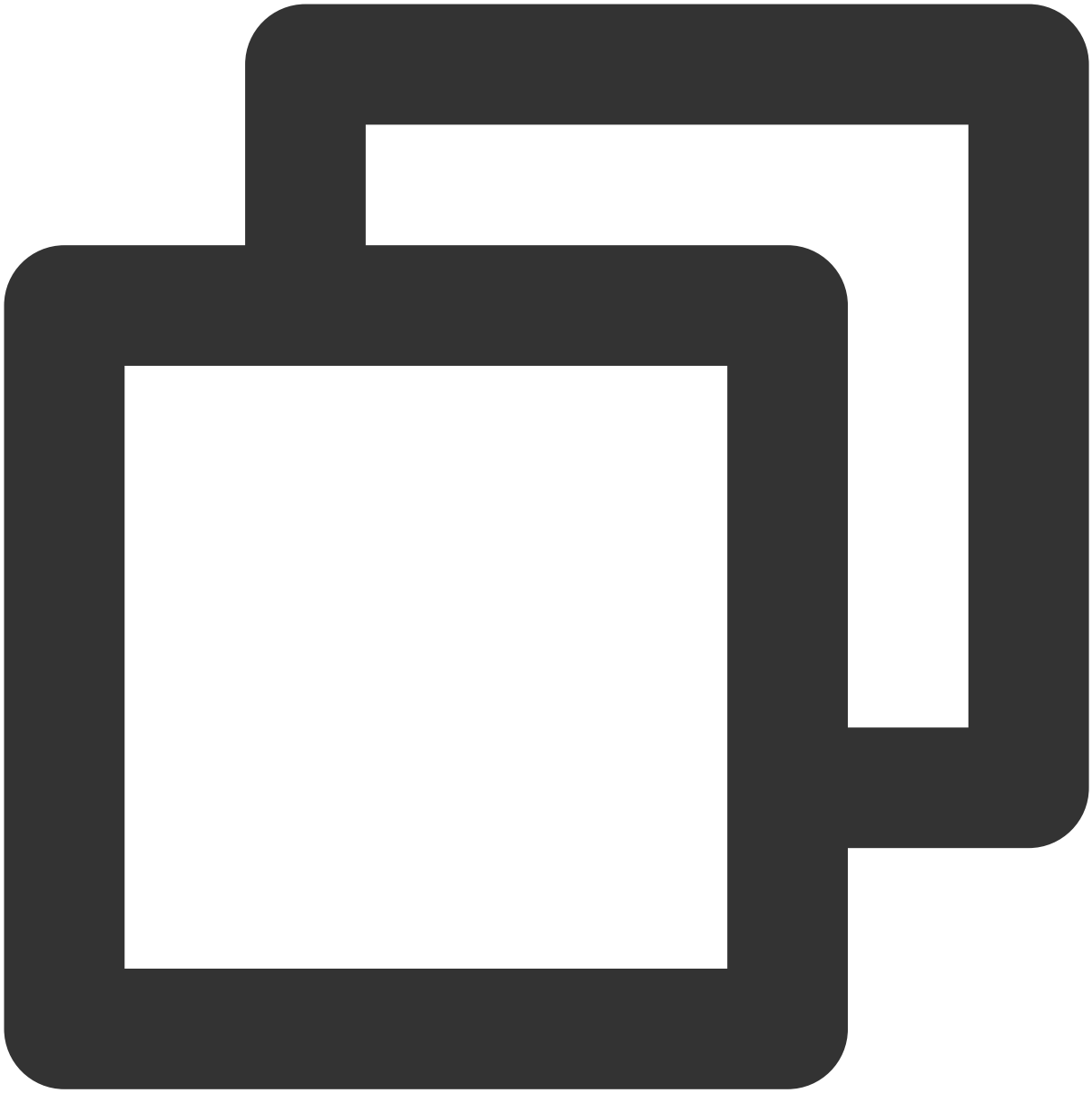
```
......
// 记录搜索游标
String searchCursor = "";
......

private void searchConversation(String cursor) {
    List<String> keywordList = new ArrayList<>();
    keywordList.add("test");
    V2TIMMessageSearchParam v2TIMMessageSearchParam = new V2TIMMessageSearchParam()
    v2TIMMessageSearchParam.setConversationID(null);
    v2TIMMessageSearchParam.setKeywordList(keywordList);
```

```
    v2TIMMessageSearchParam.setSearchCount(10);
    v2TIMMessageSearchParam.setSearchCursor(cursor);
    V2TIMManager.getMessageManager().searchCloudMessages(v2TIMMessageSearchParam, n
      @Override
      public void onSuccess(V2TIMMessageSearchResult v2TIMMessageSearchResult) {
        // 匹配到的消息所属的所有会话数量
        int totalCount = v2TIMMessageSearchResult.getTotalCount();
        // 下一页的游标
        searchCursor = v2TIMMessageSearchResult.getSearchCursor();
        // 该页的根据消息会话分类的信息
        List<V2TIMMessageSearchResultItem> resultItemList = v2TIMMessageSearchResul
        for (V2TIMMessageSearchResultItem resultItem : resultItemList) {
          // 会话 ID
          String conversationID = resultItem.getConversationID();
          // 该会话匹配到的所有消息数量
          int totalMessageCount = resultItem.getMessageCount();
          // 消息列表：如果 totalMessageCount > 1，该列表为空；如果 totalMessageCount =
          List<V2TIMMessage> v2TIMMessageList = resultItem.getMessageList();
        }
      }

      @Override
      public void onError(int code, String desc) {
      }
    });
  }

// 当需要加载下一页时
public void loadMore() {
    searchConversation(searchCursor);
}
```



```
......
// 记录搜索游标
NSString *searchCursor = @"";
......

- (void)searchConversation:(NSString *)cursor {
    V2TIMMessageSearchParam *param = [[V2TIMMessageSearchParam alloc] init];
    param.keywordList = @[@"test"];
    param.conversationID = nil;
    param.searchCursor = cursor;
    param.searchCount = 10;
```

```
    [V2TIMManager.sharedInstance searchCloudMessages:param succ:^(V2TIMMessageSearc
      // 匹配到的消息所属的所有会话数量
      NSUInteger totalCount = searchResult.totalCount;
      // 下一页的游标
      searchCursor = searchResult.searchCursor;
      // 该页的根据消息会话分类的信息
      NSArray<V2TIMMessageSearchResultItem *> *messageSearchResultItems = searchRes
      for (V2TIMMessageSearchResultItem *searchItem in messageSearchResultItems) {
        // 会话 ID
        NSString *conversationID = searchItem.conversationID;
        // 该会话匹配到的所有消息数量
        NSUInteger totalMessageCount = searchItem.messageCount;
        // 消息列表：如果 totalMessageCount > 1，该列表为空；如果 totalMessageCount = 1,
        NSArray<V2TIMMessage *> *messageList = searchItem.messageList ?: @[];
      }
    } fail:^(int code, NSString *desc) {
    // fail
    }];
}

// 当需要加载下一页时
- (void)loadMore {
    [self searchConversation:searchCursor];
}
```



```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
      success_callback_ = std::move(success_callback);
```

```
      error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
    if (success_callback_) {
      success_callback_(value);
      }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
    if (error_callback_) {
      error_callback_(error_code, error_message);
      }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

// 记录搜索游标
V2TIMString searchCursor = "";

void SearchConversation(V2TIMString cursor) {
    V2TIMMessageSearchParam searchParam;
    searchParam.keywordList.PushBack("test");
    searchParam.searchCursor = cursor;
    searchParam.searchCount = 10;

    auto callback = new ValueCallback<V2TIMMessageSearchResult>{};
      callback->SetCallback(
      [=](const V2TIMMessageSearchResult& messageSearchResult) {
        // 匹配到的消息所属的所有会话数量
        uint32_t totalCount = messageSearchResult.totalCount;
        // 下一页的游标
        searchCursor = messageSearchResult.searchCursor;
        // 该页的根据消息会话分类的信息
        V2TIMMessageSearchResultItemVector messageSearchResultItems =
        messageSearchResult.messageSearchResultItems;
        for (size_t i = 0; i < messageSearchResultItems.Size(); ++i) {
          // 会话 ID
          V2TIMString conversationID = messageSearchResultItems[i].conversationID;
          // 该会话匹配到的所有消息数量
          uint32_t messageCount = messageSearchResultItems[i].messageCount;
          // 消息列表：如果 messageCount > 1，该列表为空；如果 messageCount = 1，该列表元素
          V2TIMMessageVector messageList = messageSearchResultItems[i].messageList;
        }
        delete callback;
```

```
        },
        [=](int error_code, const V2TIMString& error_message) {
        // 搜索失败
        delete callback;
      });

    V2TIMManager::GetInstance()->GetMessageManager()->SearchLocalMessages(searchPar
}

// 当需要加载下一页时
void LoadMore() { SearchConversation(searchCursor); }
```

展示搜索指定会话的消息

与搜索典型场景示例图 2 展示会话列表不同的是，图 3 所示在指定会话中搜索到的消息列表。 为了防止内存膨胀，我们强烈建议您分页加载消息。举个例子，分页加载，每页展示 10 条消息结果：

1. 搜索参数 `V2TIMMessageSearchParam` 可以参考如下设置：

设置搜索参数 `V2TIMMessageSearchParam` 的 `conversationID` 为搜索的会话 ID。

首次调用：设置参数 `searchCount` = 10， `searchCursor` = ""。调用 `searchCloudMessages` 获取消息搜索结果，解析并展示到首页，并且从结果回调中获取会话总数量 `totalCount` 以及下一页的游标 `searchCursor` 。

再次调用：更新参数 `searchCursor` 的值为上一步调用结果中的返回值。

2. 处理搜索结果 `V2TIMMessageSearchResult` ：

`totalCount` 表示该会话匹配到的所有消息数量。

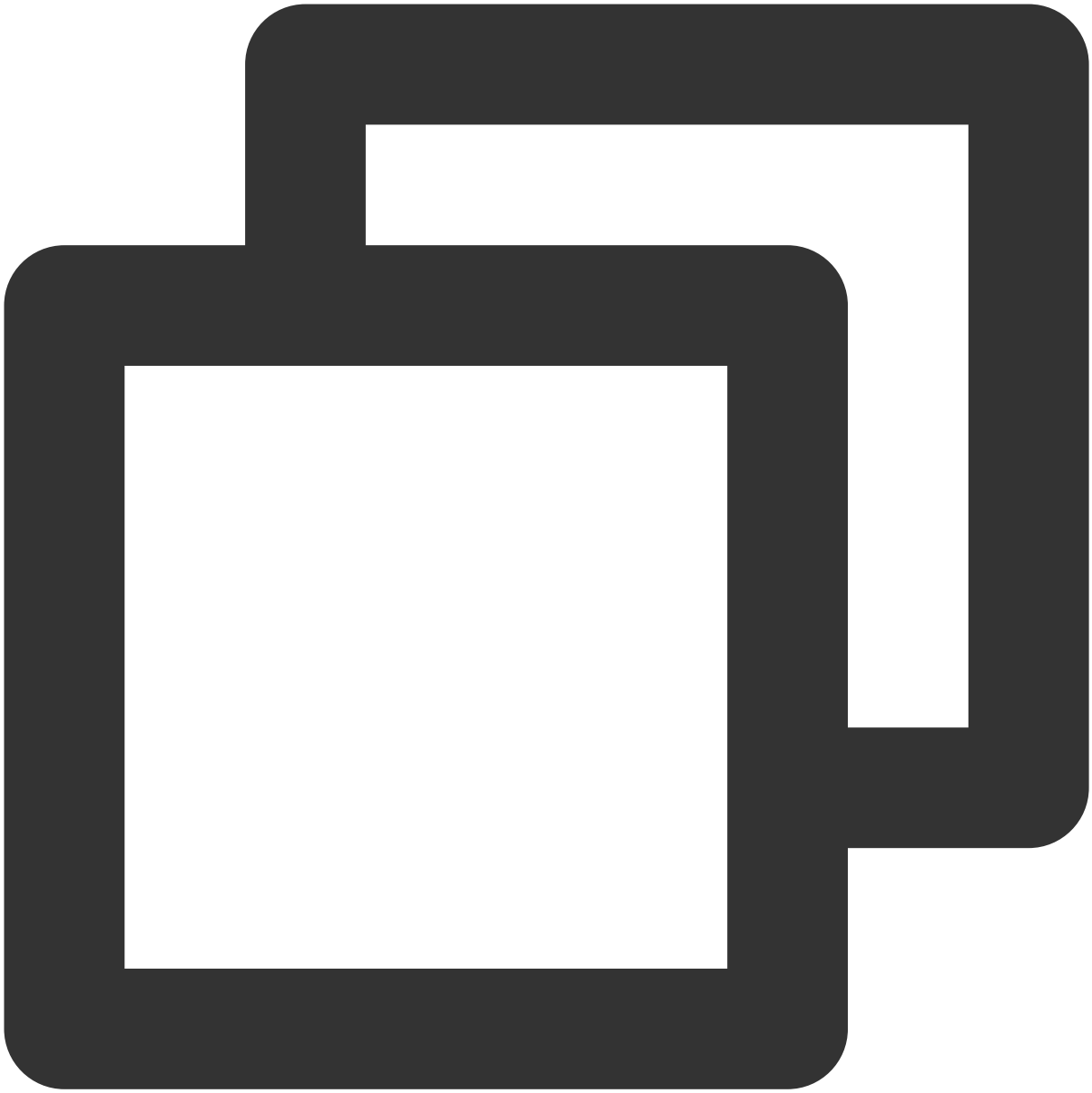
`messageSearchResultItems` 列表只有该会话的结果。列表中的元素 `V2TIMMessageSearchResultItem` 的 `messageCount` 为该分页的消息数量， `messageList` 为该分页的消息列表。

`searchCursor` 表示下一页搜索的起始游标。

Android

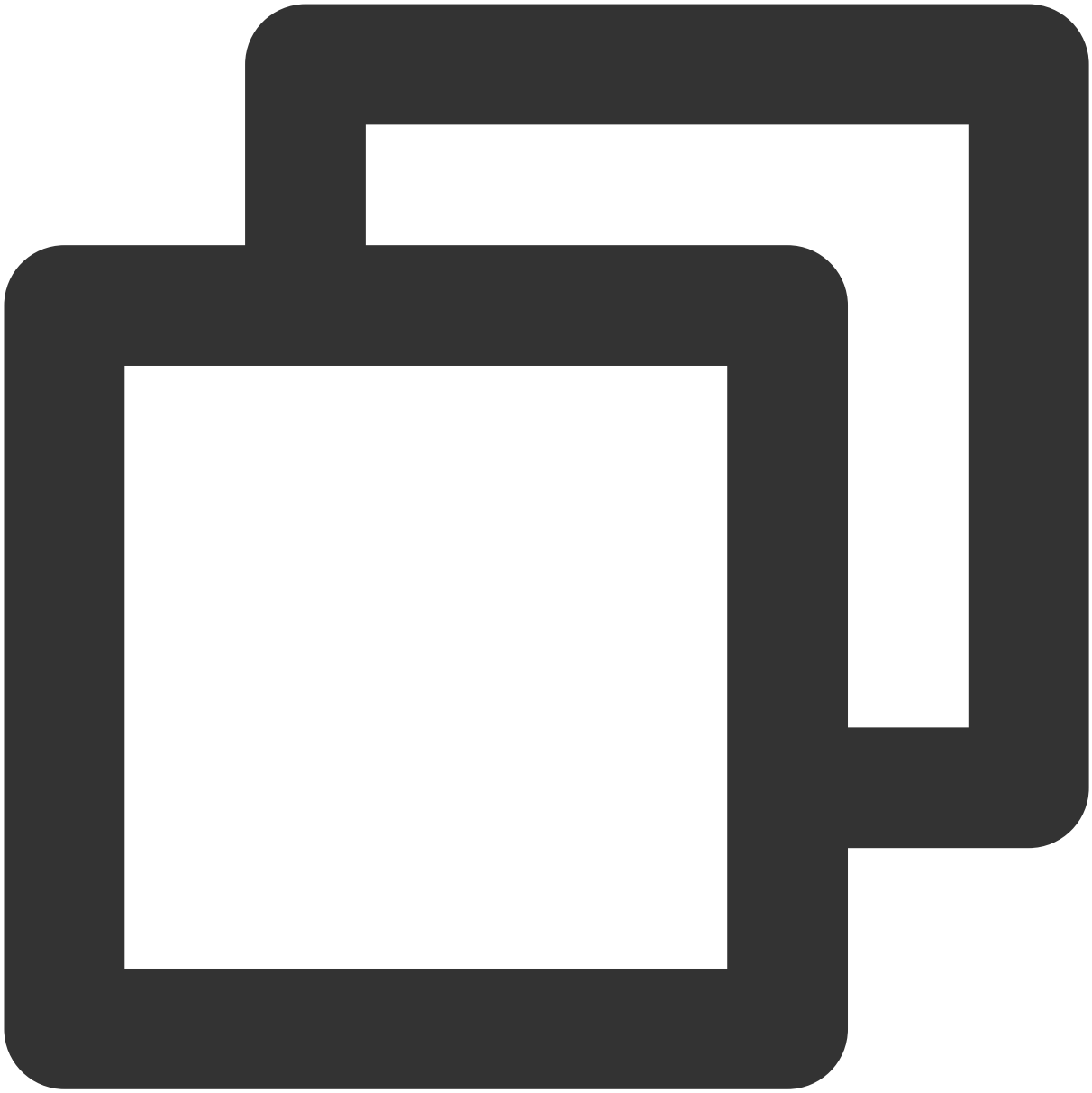
iOS & Mac

Windows



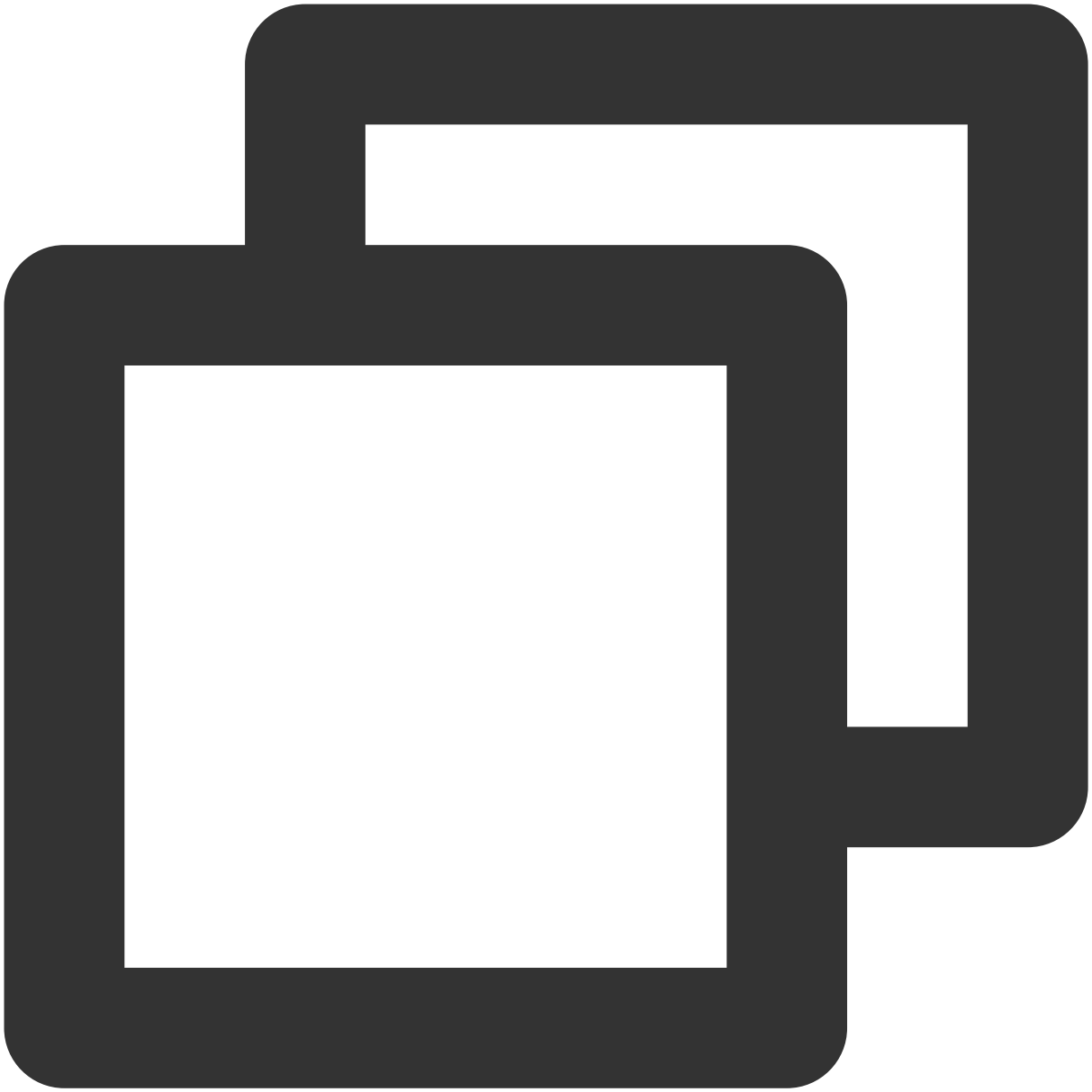
```
......
// 记录搜索游标
String searchCursor = "";
......

private void searchMessage(String cursor) {
    List<String> keywordList = new ArrayList<>();
    keywordList.add("test");
    V2TIMMessageSearchParam v2TIMMessageSearchParam = new V2TIMMessageSearchParam()
    v2TIMMessageSearchParam.setConversationID(conversationID);
    v2TIMMessageSearchParam.setKeywordList(keywordList);
```

```
    v2TIMMessageSearchParam.setSearchCount(10);
    v2TIMMessageSearchParam.setSearcuCursor(cursor);
    V2TIMManager.getMessageManager().searchCloudMessages(v2TIMMessageSearchParam,ne
      @Override
      public void onSuccess(V2TIMMessageSearchResult v2TIMMessageSearchResult) {
      // 该会话匹配到的所有消息数量
      int totalMessageCount = v2TIMMessageSearchResult.getTotalCount();
      // 下一页的游标
      searchCursor = v2TIMMessageSearchResult.getSearchCursor();
      // 该页消息信息
      List<V2TIMMessageSearchResultItem> resultItemList = v2TIMMessageSearchResult.
      for (V2TIMMessageSearchResultItem resultItem : resultItemList) {
        // 会话 ID
        String conversationID = resultItem.getConversationID();
        // 该页的消息数量
        int totalMessageCount = resultItem.getMessageCount();
        // 该页的消息数据列表
        List<V2TIMMessage> v2TIMMessageList = resultItem.getMessageList();
      }
    }

      @Override
      public void onError(int code, String desc) {
      }
    });
}
// 当需要加载下一页时
public void loadMore() {
    searchMessage(searchCursor);
}
```



```
......
// 记录搜索游标
NSString *searchCursor = @"";
......

- (void)searchMessage:(NSString *)cursor {
  V2TIMMessageSearchParam *param = [[V2TIMMessageSearchParam alloc] init];
  param.keywordList = @[@"test"];
  // conversationID 是要搜索的会话 ID
  param.conversationID = conversationID;
  param.searchCursor = cursor;
```

```
    param.searchCount = 10;
    [V2TIMManager.sharedInstance searchCloudMessages:param succ:^(V2TIMMessageSearchR
      // 该会话匹配到的所有消息数量
      NSUInteger totalMessageCount = searchResult.totalCount;
      // 下一页的游标
      searchCursor = searchResult.searchCursor;
      // 该页消息信息
      NSArray<V2TIMMessageSearchResultItem *> *messageSearchResultItems = searchResul
      for (V2TIMMessageSearchResultItem *searchItem in messageSearchResultItems) {
        // 会话 ID
        NSString *conversationID = searchItem.conversationID;
        // 该页的消息数量
        NSUInteger totalMessageCount = searchItem.messageCount;
        // 该页的消息数据列表
        NSArray<V2TIMMessage *> *messageList = searchItem.messageList ?: @[];
        }
      } fail:^(int code, NSString *desc) {
        // fail
      }];
}

// 当需要加载下一页时
- (void)loadMore {
    [self searchMessage:searchCursor];
}
```



```
template <class T>
class ValueCallback final : public V2TIMValueCallback<T> {
public:
    using SuccessCallback = std::function<void(const T&)>;
    using ErrorCallback = std::function<void(int, const V2TIMString&)>;

    ValueCallback() = default;
    ~ValueCallback() override = default;

    void SetCallback(SuccessCallback success_callback, ErrorCallback error_callback
      success_callback_ = std::move(success_callback);
```

```
      error_callback_ = std::move(error_callback);
    }

    void OnSuccess(const T& value) override {
      if (success_callback_) {
        success_callback_(value);
      }
    }
    void OnError(int error_code, const V2TIMString& error_message) override {
      if (error_callback_) {
        error_callback_(error_code, error_message);
      }
    }

private:
    SuccessCallback success_callback_;
    ErrorCallback error_callback_;
};

// 记录搜索游标
V2TIMString searchCursor = "";

void SearchConversation(V2TIMString cursor) {
    V2TIMMessageSearchParam searchParam;
    searchParam.conversationID = conversationID;
    searchParam.keywordList.PushBack("test");
    searchParam.searchCursor = cursor;
    searchParam.searchCount = 10;

    auto callback = new ValueCallback<V2TIMMessageSearchResult>{};
    callback->SetCallback(
      [=](const V2TIMMessageSearchResult& messageSearchResult) {
        // 该会话匹配到的所有消息数量
        uint32_t totalCount = messageSearchResult.totalCount;
        // 下一页的游标
        searchCursor = messageSearchResult.searchCursor;
        // 该页消息信息
        V2TIMMessageSearchResultItemVector messageSearchResultItems =
        messageSearchResult.messageSearchResultItems;
        for (size_t i = 0; i < messageSearchResultItems.Size(); ++i) {
          // 会话 ID
          V2TIMString conversationID = messageSearchResultItems[i].conversationID;
          // 该页的消息数量
          uint32_t messageCount = messageSearchResultItems[i].messageCount;
          // 该页的消息数据列表
          V2TIMMessageVector messageList = messageSearchResultItems[i].messageList;
        }
```

```
        delete callback;
    },
    [=](int error_code, const V2TIMString& error_message) {
        // 搜索失败
        delete callback;
    });

    V2TIMManager::GetInstance()->GetMessageManager()->SearchCloudMessages(searchPar
}

// 当需要加载下一页时
void LoadMore() { SearchConversation(SearchCursor); }
```

搜索自定义消息

通常情况下，如果您使用接口 `createCustomMessage(data)` （Android / iOS & Mac / Windows）创建自定义消息，该消息无法被搜到，因为 SDK 将该自定义消息保存为二进制数据流。
如果您希望自定义消息可以被搜到，需要使用接口 `createCustomMessage(data, description, extension)` （Android / iOS & Mac / Windows）来创建并发送自定义消息，把需要搜索的文本放到 `description` 参数中。
如果您配置了离线推送功能，设置参数 `description` 后，自定义消息也会有离线推送且通知栏展示该参数内容。 如果不需要离线推送可以用发消息接口 `sendMessage` （Android / iOS & Mac / Windows）的参数 `V2TIMOfflinePushInfo` 中的 `disablePush` 来控制。
如果推送的通知栏内容不想展示为被搜索的文本，可以用参数 `V2TIMOfflinePushInfo` 中的 `desc` 来另外设置推送内容。

搜索富媒体消息

富媒体消息包含文件、图片、语音、视频消息。
对于文件消息，界面通常显示文件名。如果调用 `createFileMessage` 创建文件消息时传入 `fileName` 参数， `fileName` 会作为文件消息被搜索的内容，与搜索关键词进行匹配。如果未设置 `fileName` ，SDK 则会自动从 `filePath` 提取文件名作为搜索内容。 `fileName` 和 `filePath` 信息会保存到本地和服务器，换设备拉取相关信息后均可搜索。
对于图片、语音、视频消息，并没有类似 `fileName` 这种名称，界面通常显示缩略图或时长，此时指定 `keywordList` 搜索无效。 如果您希望搜索出此类消息，可以指定 `messageTypeList` 为 `V2TIM_ELEM_TYPE_IMAGE` / `V2TIM_ELEM_TYPE_SOUND` / `V2TIM_ELEM_TYPE_VIDEO` 做分类搜索，此时会搜索出所有指定类型的消息。

交流与反馈

加入Telegram 技术交流群组或 WhatsApp 交流群，享有专业工程师的支持，解决您的难题。

Web & uni-app & 小程序

最近更新时间：2024-04-23 15:41:20

功能体验

含 UI 集成

快速集成消息云端搜索

Web&H5 Vue2&Vue3

Uniapp Vue2&Vue3

步骤1: 集成TUIKit

@tencentcloud/chat-uikit-vue ≥ 2.0.0，如未集成，请务必先根据 Vue2 & Vue3 TUIKit 快速集成指引 进行集成。

步骤2：控制台开通云端搜索插件

注意：

每个插件限免费试用 1 次，有效期 7 天，试用结束后将停服，请提前购买。试用时，仅支持搜索开通云端搜索功能后产生的消息内容，不支持历史消息搜索；购买插件后，将自动同步历史消息，支持历史消息搜索。

步骤3：搜索您的第一条消息

在完成 Vue2 & Vue3 TUIKit 快速集成指引 - 步骤6: 发送您的第一条消息后，搜索您刚才发送的消息。

步骤1: 集成TUIKit

@tencentcloud/chat-uikit-uniapp ≥ 2.0.6，如未集成，请请务必先根据 uniapp TUIKit 快速集成指引 进行集成。

步骤2：控制台开通云端搜索插件

注意：

每个插件限免费试用 1 次，有效期 7 天，试用结束后将停服，请提前购买。试用时，仅支持搜索开通云端搜索功能后产生的消息内容，不支持历史消息搜索；购买插件后，将自动同步历史消息，支持历史消息搜索。

步骤3：搜索你的第一条消息

在完成 Uniapp TUIKit 快速集成指引 - 步骤6: 发送您的第一条消息后，搜索您刚才发送的消息。

独立引入消息云端搜索

说明：

以上 快速集成消息云端搜索 中已默认包含消息云端搜索全部功能，无需重复引入。

如果您想独立引入 <TUISearch> 消息云端搜索，请参考以下教程。

Web&H5 Vue2&Vue3

Uniapp Vue2&Vue3

前提条件

@tencentcloud/chat-uikit-vue ≥ 2.0.0，如未集成，请务必先根据 Vue2 & Vue3 TUIKit 快速集成指引 进行集成。

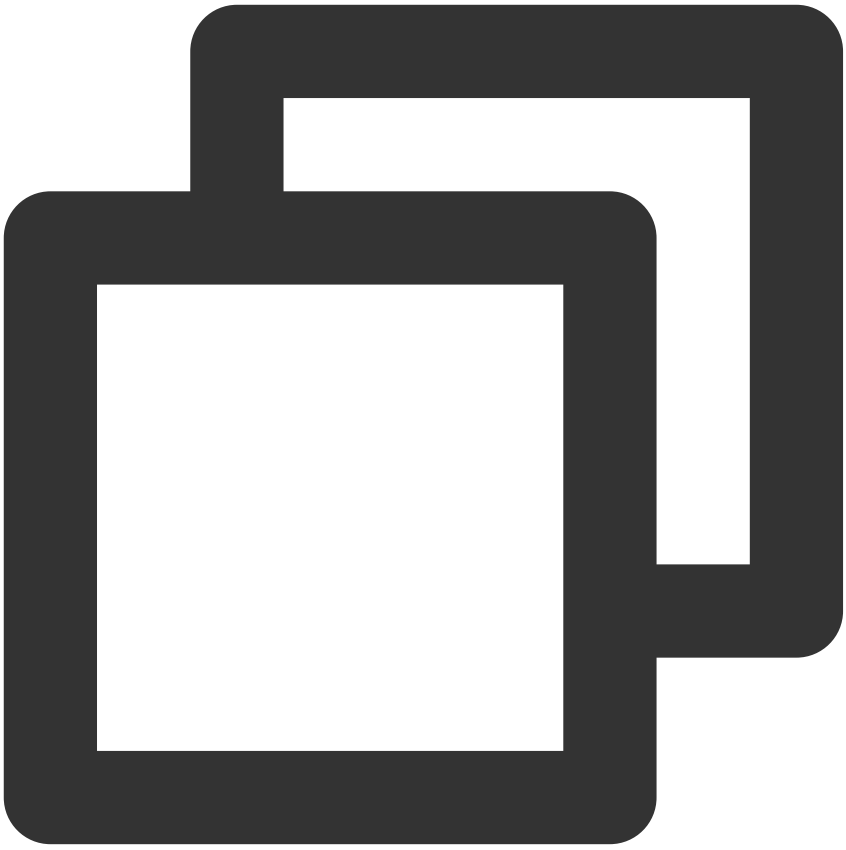
引入 <TUISearch>

在您所需要使用 **消息云端搜索** 功能的 `.vue` 界面，引入 **<TUISearch>** 。

<TUISearch> 参数说明

参数名	类型	说明
searchType	String	global：全局搜索（default）
		conversation：会话内搜索

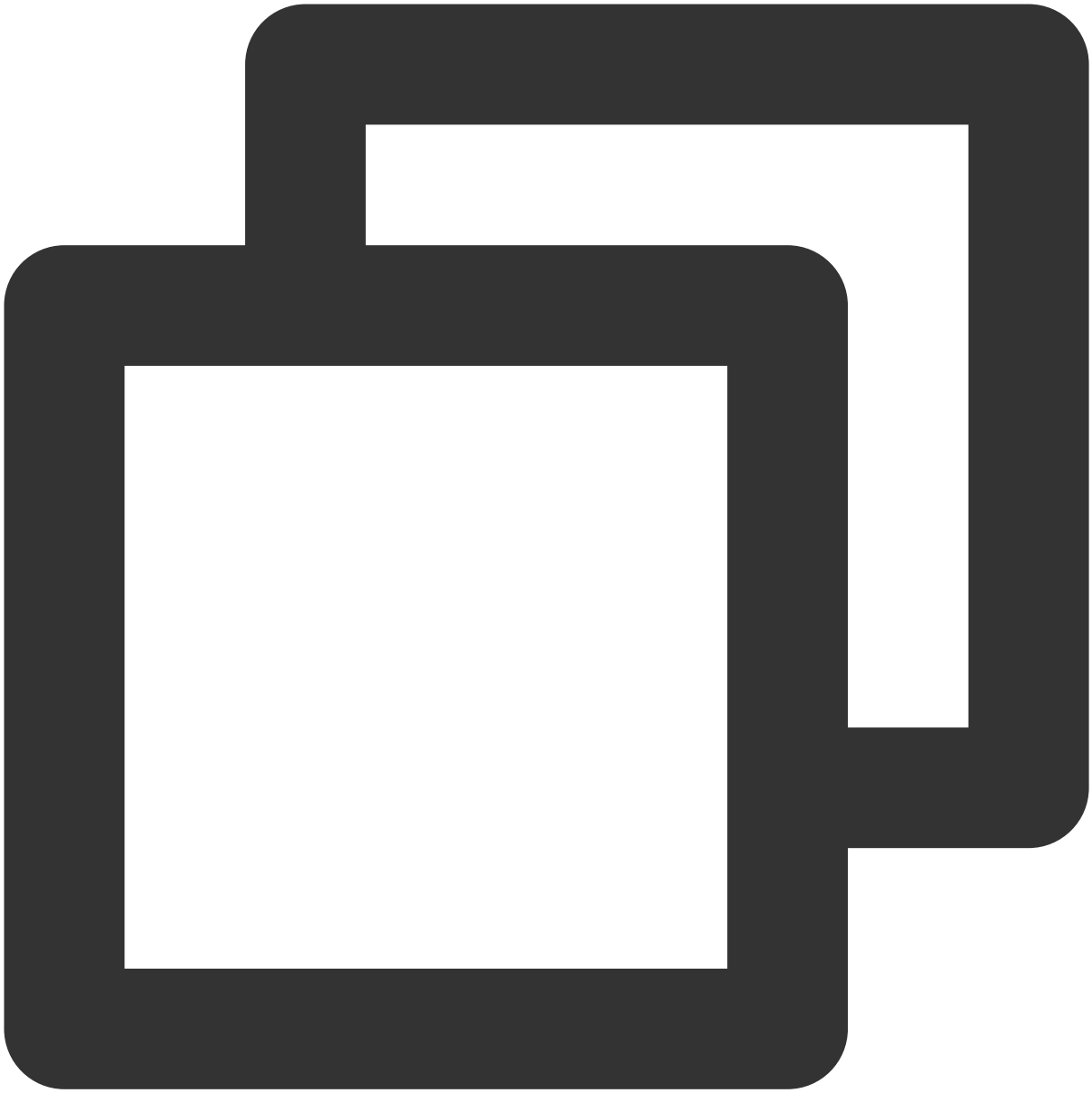
<TUISearch> 效果展示



```
<TUISearch searchType="global" />
```

```
<TUISearch searchTy
```

使用 TUISearch



```
import { TUISearch } from "@tencentcloud/chat-uikit-vue";
// 全局搜索
<TUISearch searchType="global" />
// 会话内搜索
<TUISearch searchType="conversation" />
```

删除默认引入的 TUISearch

TUIKit 中默认集成 `<TUISearch>`，如您不按照默认集成方式使用，可在 `TUIKit/index.vue` 中，注释掉 `<TUISearch>` 即可。

Uniapp TUISearch 支持两种方式引入：组件方式引入与界面方式引入。

前提条件

@tencentcloud/chat-uikit-uniapp ≥ 2.0.6，如未集成，请请务必先根据 uniapp TUIKit 快速集成指引 进行集成。

组件方式引入

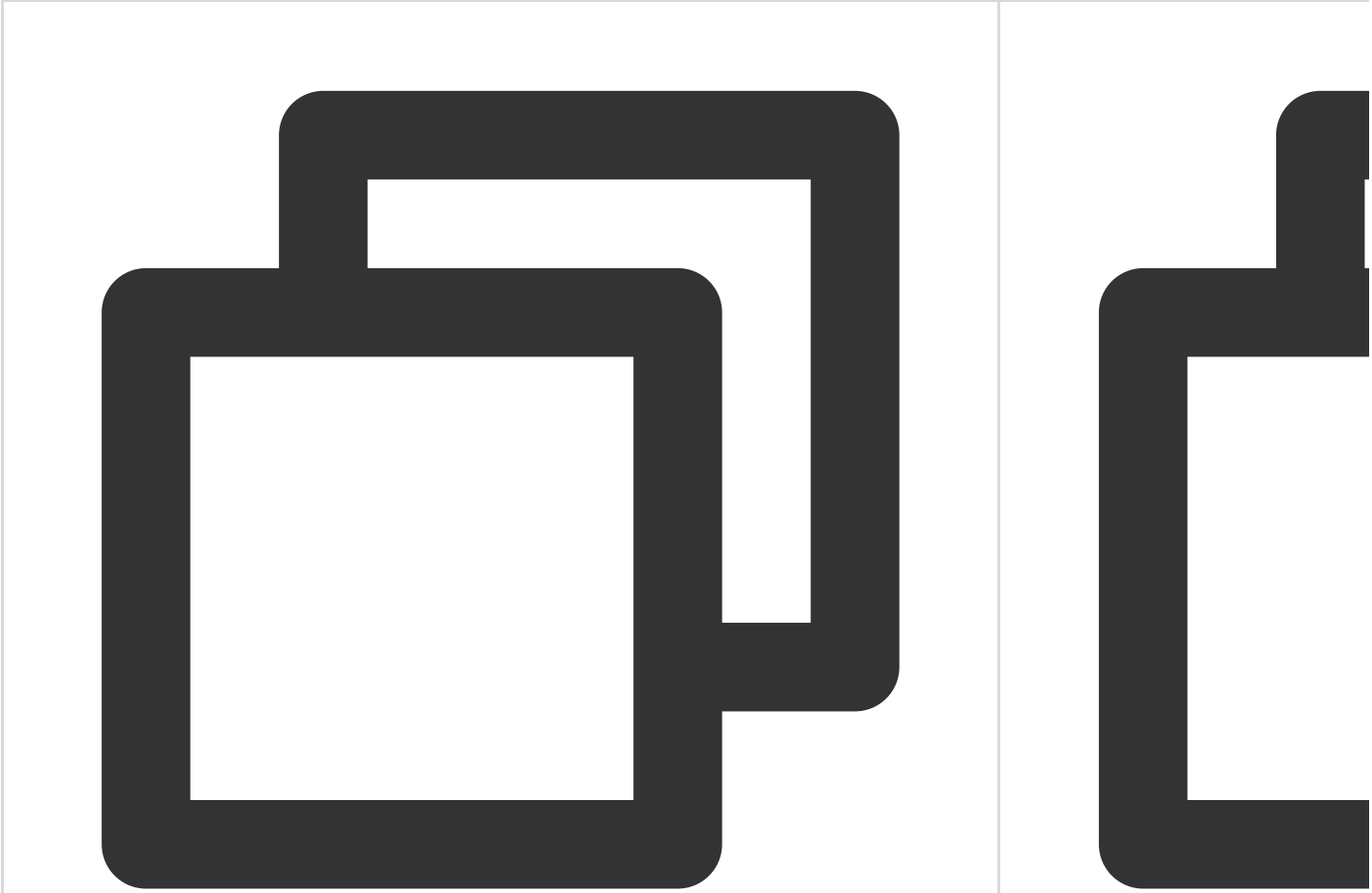
界面方式引入

在您所需要使用 **消息云端搜索** 功能的 `.vue` 界面，引入 **<TUISearch>** 。

<TUISearch> 参数说明

参数名	类型	说明
searchType	String	global：全局搜索
		conversation：会话内搜索（default）

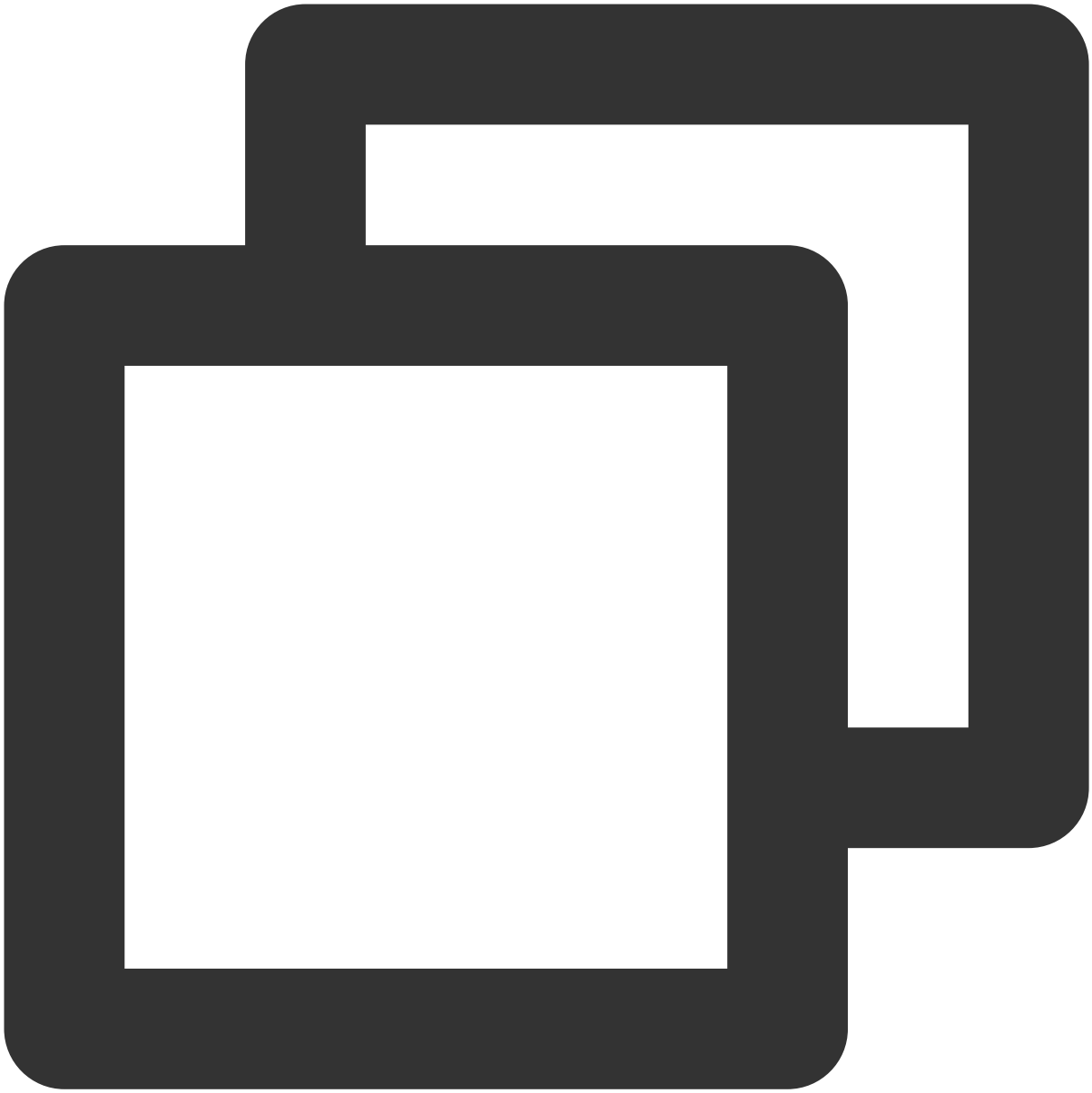
<TUISearch> 效果展示



```
<TUISearch searchType="global" />
```

```
<TUISearch searchTyp
```

使用 TUISearch

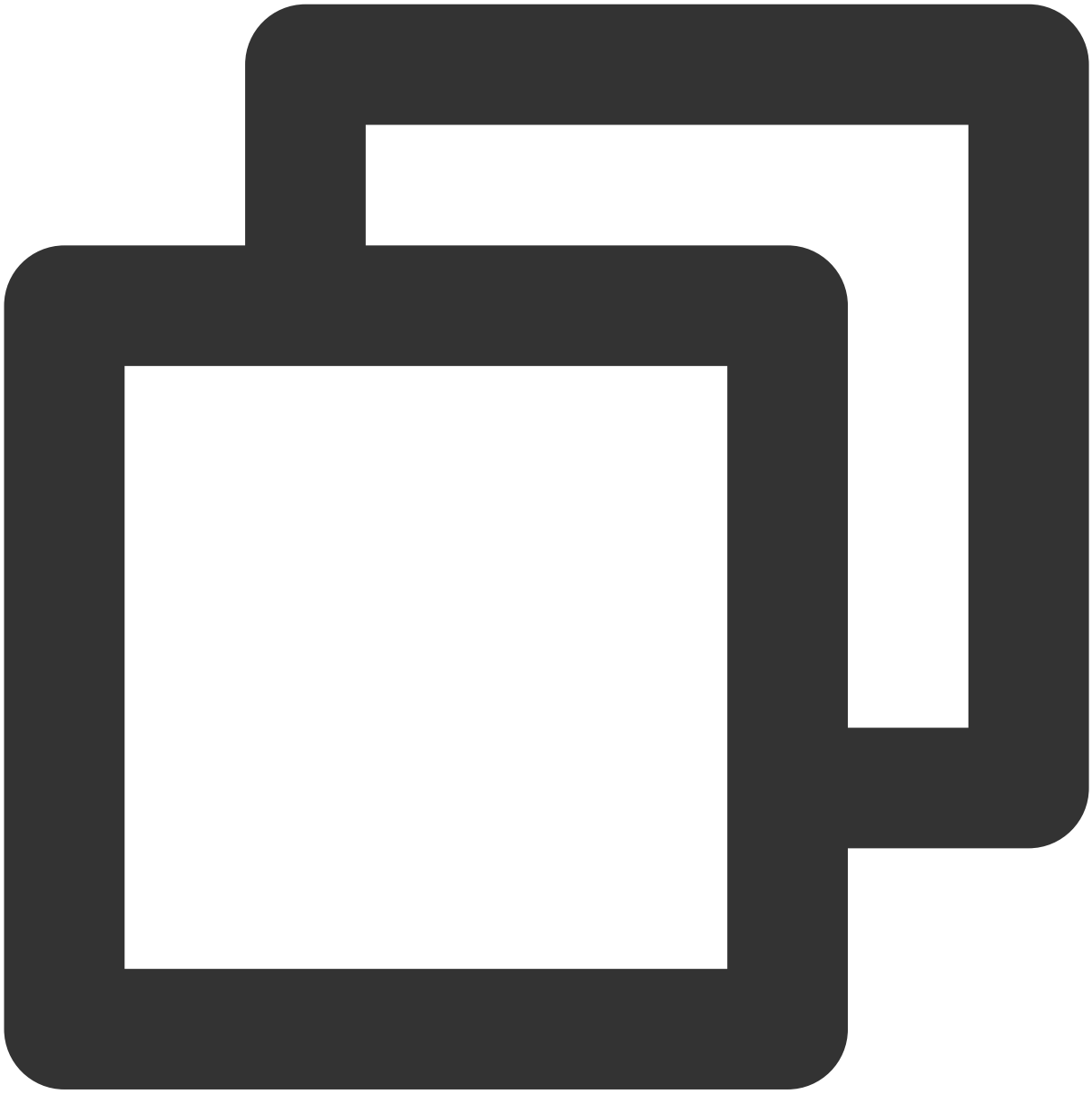


```
// 以下路径仅为示例路径，请根据您项目自身路径进行调整
import { TUISearch } from "/TUIKit/components/TUISearch/index.vue";
// 全局搜索
<TUISearch searchType="global" />
// 会话内搜索
<TUISearch searchType="conversation" />
```

删除默认引入的 TUISearch

TUIKit 中默认集成 `<TUISearch>` ， 如您不按照默认集成方式使用，可在 `TUIKit/components/TUIConversation/index.vue` 中，注释掉 `<TUISearch>` 即可。

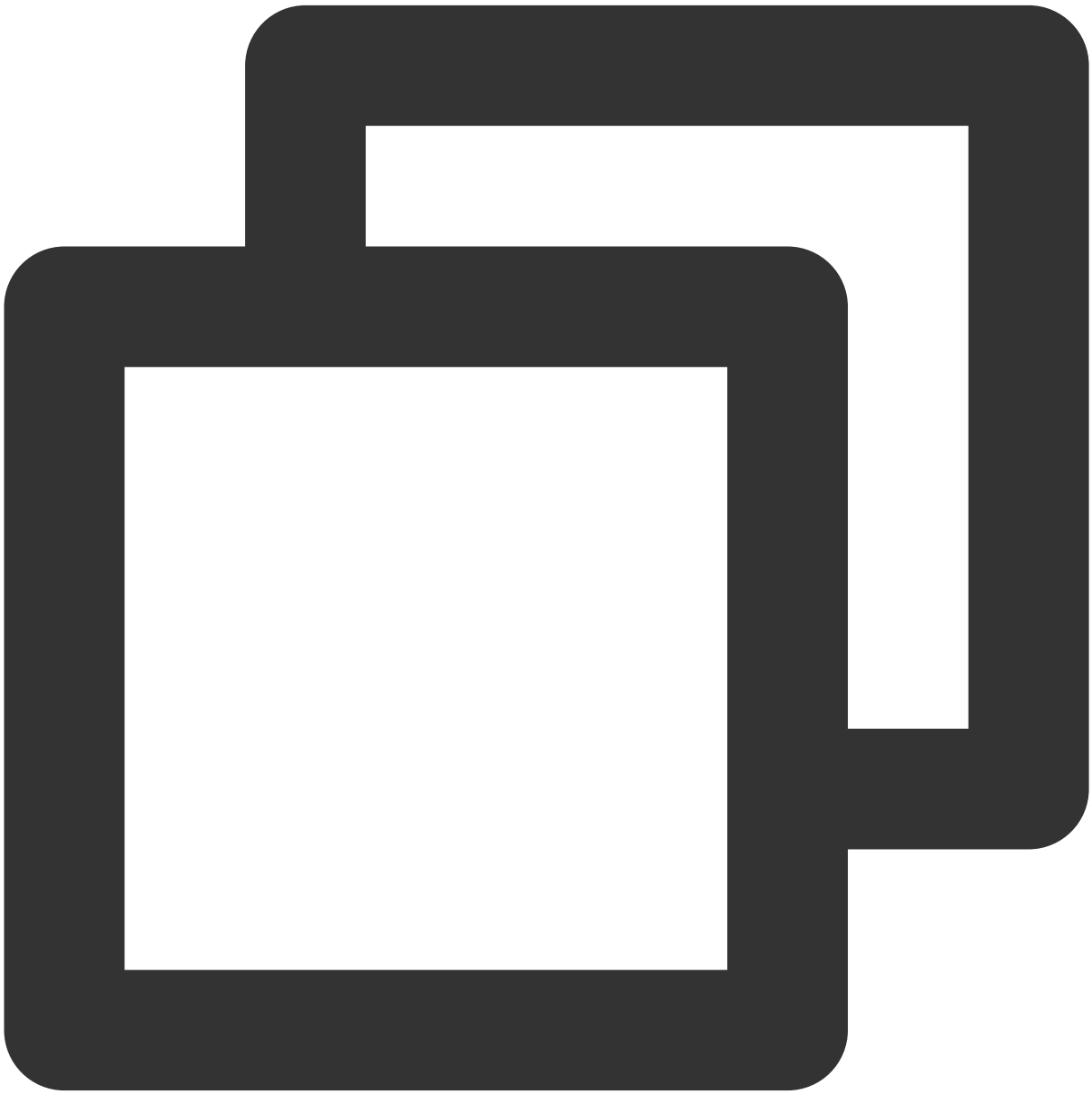
在 pages.json 新增 TUISearch 页面



```
{
  "pages": [
    ...,
    {
      "path": "TUIKit/components/TUISearch/index",
      "style": {
```

```
          "navigationBarTitleText": "聊天记录"
        }
      }
    ],
    ...
}
```

跳转到 TUISearch 界面



```
uni.navigateTo({
    url: "/TUIKit/components/TUISearch/index",
```

```
});
```

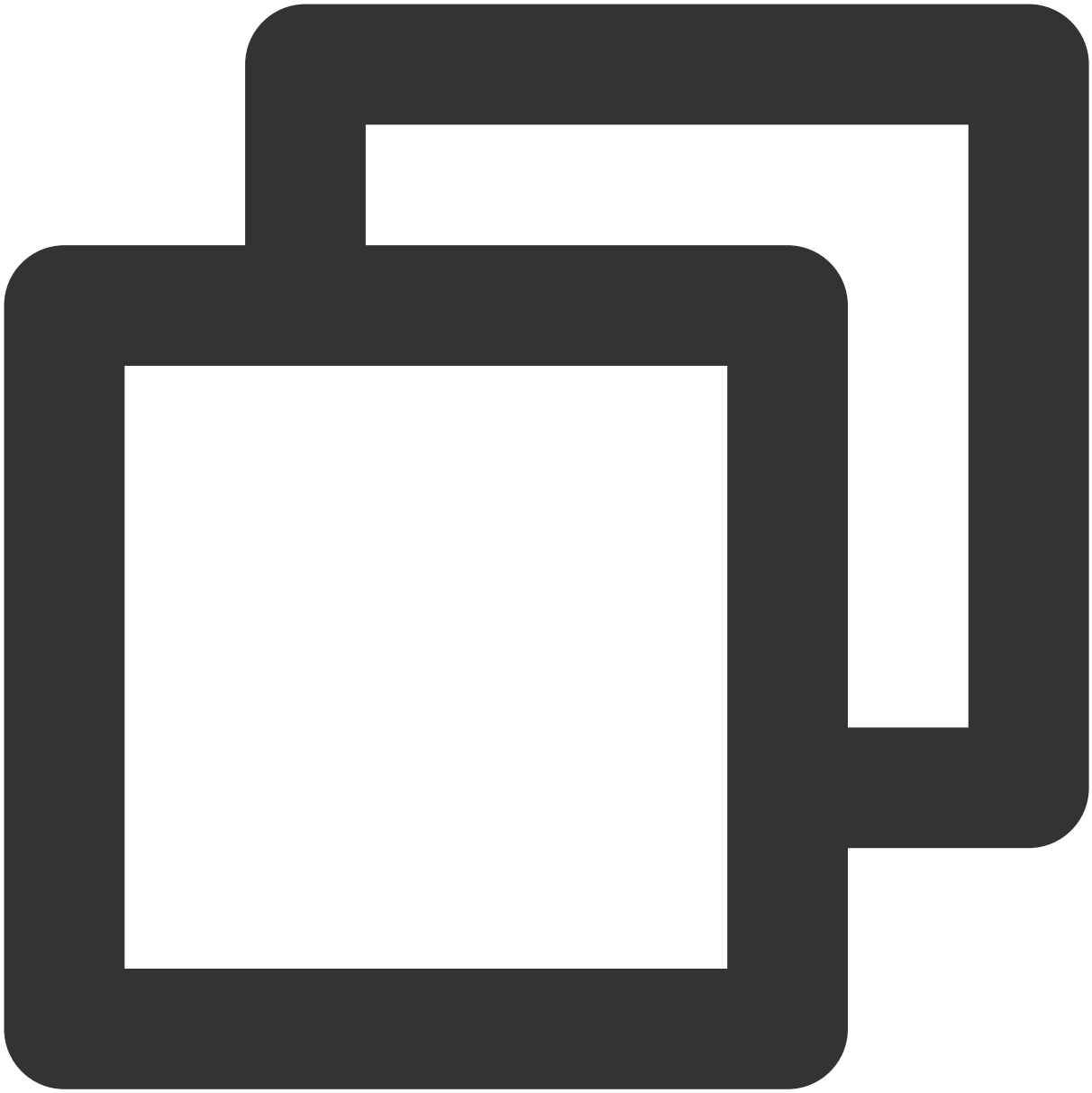
高级指引

新增搜索消息类型

原“全局搜索”消息类型列表	新增后“全局搜索”消息
	

目录位置： `src/TUIKit/components/TUISearch/search-type-list.ts`

`searchMessageTypeList` 中包含了所有“搜索消息类型” Tab 定义，如需新增 `searchMessageTypeList` 未定义的搜索消息类型，请按照以下结构在 `searchMessageTypeList` 中进行新增：

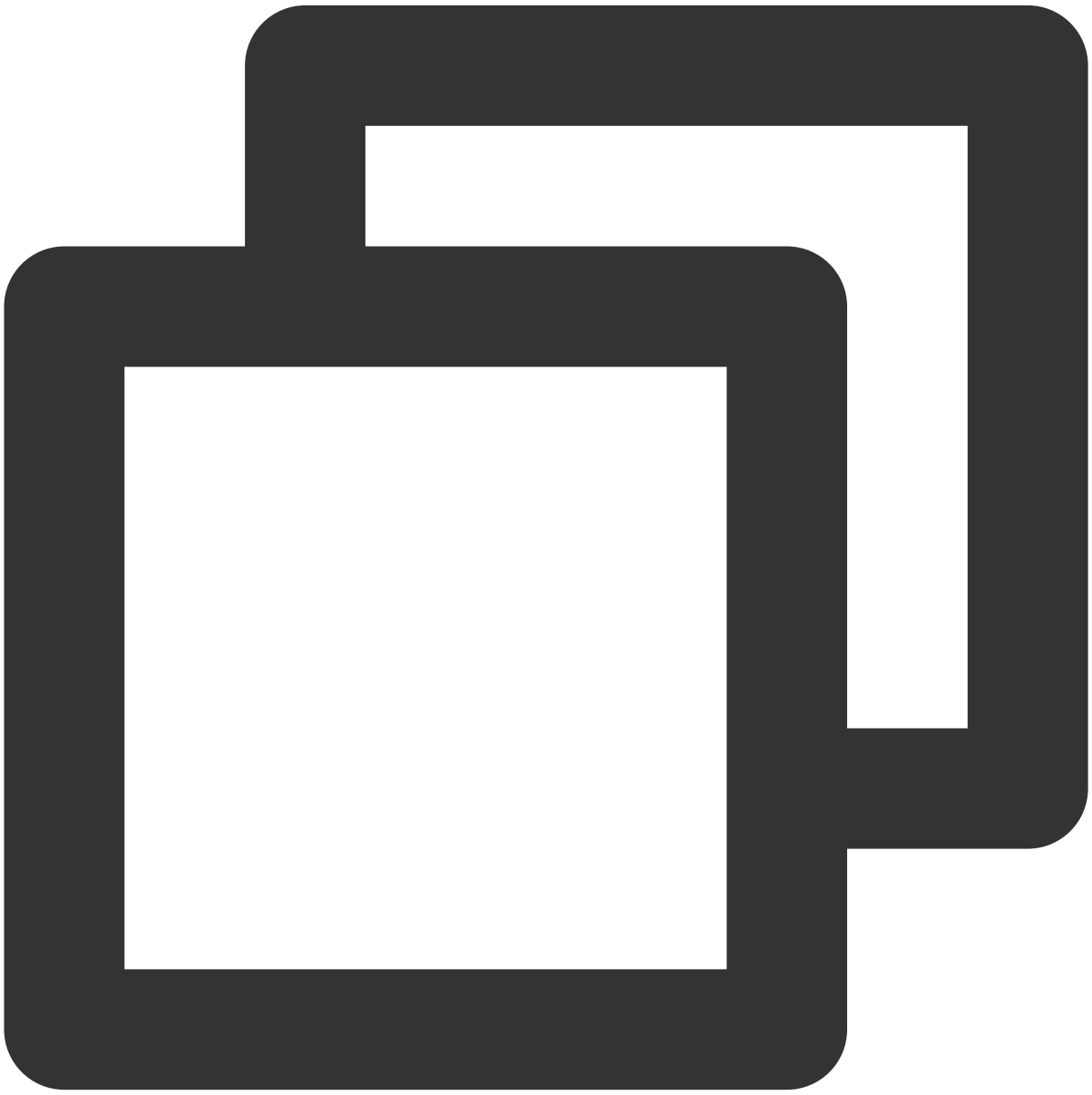


```
[keyName: string]: {
    key: string;// 消息搜索类型 key，请保持唯一性
    label: string;// 消息搜索类型渲染 label
    value: Array<string>;// 消息搜索类型实际搜索列表
 };

 // 例如，定义搜索自定义类型消息
export const searchMessageTypeList = {
    ...
    customMessage: {
      key: "customMessage",// 消息搜索类型 key，请保持唯一性
```

```
        label: "自定义",// 消息搜索类型渲染 label
        value: [TUIChatEngine.TYPES.MSG_CUSTOM],// 消息搜索类型实际搜索列表
      }
  };
```

因为 TUIKit 使用 i18next 支持国际化，如您声明新的 label，请在 `src/TUIKit/locales/zh_cn/TUISearch.ts` 以及 `src/TUIKit/locales/en/TUISearch.ts` 增加相应的国际化词条进行翻译。

如需将已定义的 `searchMessageTypeList` 中某类型增加到**全局搜索类型列表**或者**会话内搜索类型列表**，仅需将其 `key` 填入 `globalSearchTypeKeys` 或 `conversationSearchTypeKeys` 即可。



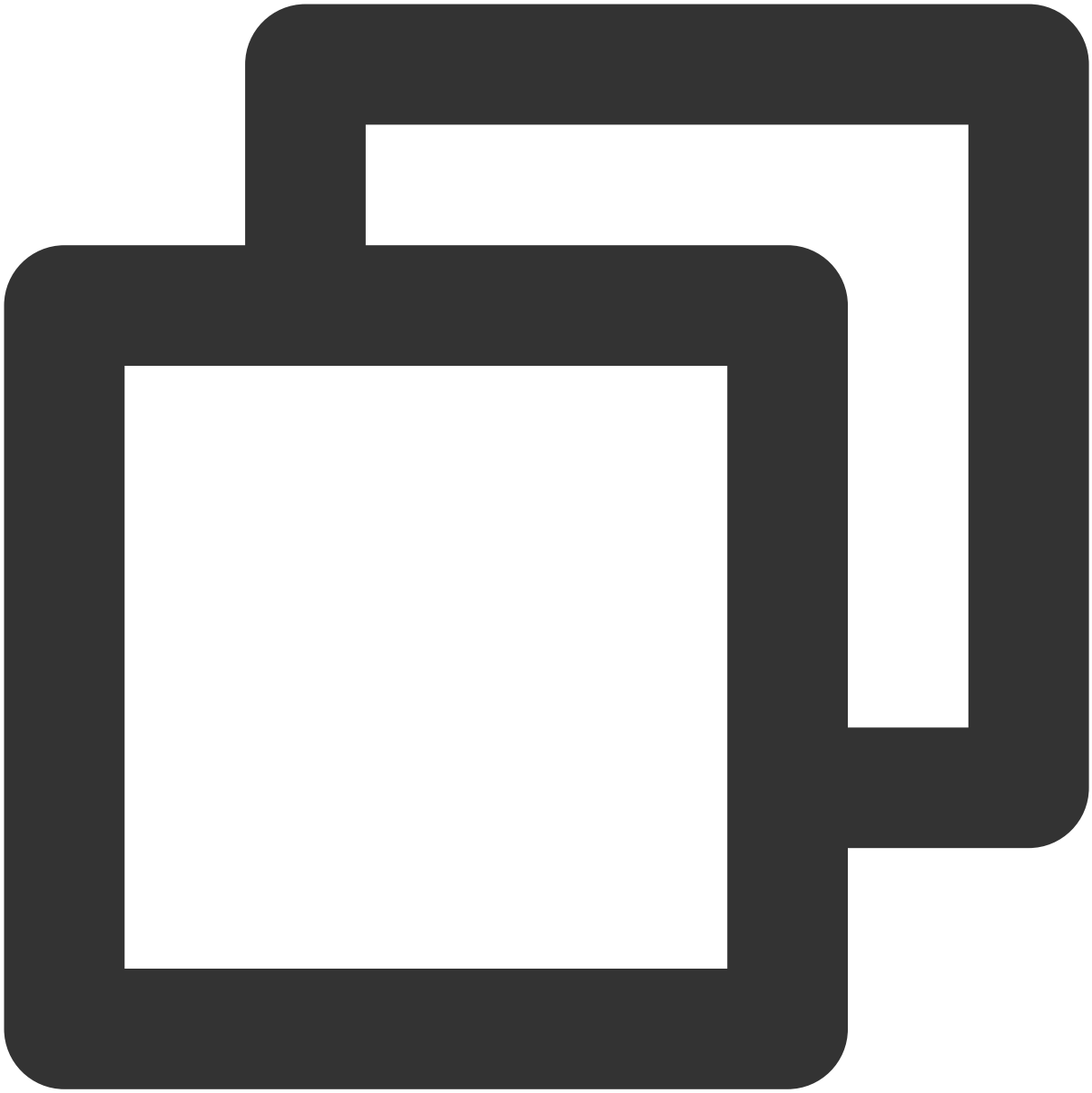
```
// 例如，将以上新增的 自定义消息 customMessage 应用到“全局搜索”消息类型列表
export const globalSearchTypeKeys = [..., "customMessage"];
// 例如，将以上新增的 自定义消息 customMessage 应用到“会话内搜索”消息类型列表
export const conversationSearchTypeKeys = [..., "customMessage"];
```

新增消息云端搜索时间范围

原“全局搜索”时间范围列表	新增后“全局搜索”时间
	

目录位置： `src/TUIKit/components/TUISearch/search-time-list.ts`

`searchMessageTimeList` 中包含了所有“搜索时间范围” Tab 定义，如需新增 `searchMessageTimeList` 未定义的搜索时间范围类型，请按照以下结构在 `searchMessageTimeList` 中进行新增：



```
[keyName: string]: {
    key: string;// 消息搜索时间范围 key，请保持唯一性
    label: string;// 消息搜索时间范围渲染 label
    value: {
      timePosition: number; // 消息搜索时间范围起始位置，默认为 0，从当前时间开始搜索
      timePeriod: number; // 从 timePosition 向前搜索的时间范围
    };
};

 // 例如，定义搜索“近两天“时间范围
export const searchMessageTimeList = {
```

```
    ...
    twoDay: {
       key: "twoDay",// 消息搜索时间范围 key，请保持唯一性
       label: "近两天",// 消息搜索时间范围渲染 label
       value: {
         timePosition: 0,// 消息搜索时间范围起始位置，默认为 0，从当前时间开始搜索
         timePeriod: 2 * oneDay,// 从 timePosition 向前搜索的时间范围
       },
   },
};
```

因为 TUIKit 使用 i18next 支持国际化，如您声明新的 label，请在 `src/TUIKit/locales/zh_cn/TUISearch.ts` 以及 `src/TUIKit/locales/en/TUISearch.ts` 增加相应的国际化词条进行翻译。

无 UI 集成

以下为**消息云端搜索**无 UI 集成方案：

功能描述

搜索云端消息，提升 IM SDK 使用体验，可以帮助用户从纷繁复杂的信息中直接找到预期内容，快捷方便；也可以作为运营工具，增加相关内容的引导，简洁高效。

注意：

搜索云端消息功能 v3.1.0 起支持。

此接口本地限频 2 次/秒。

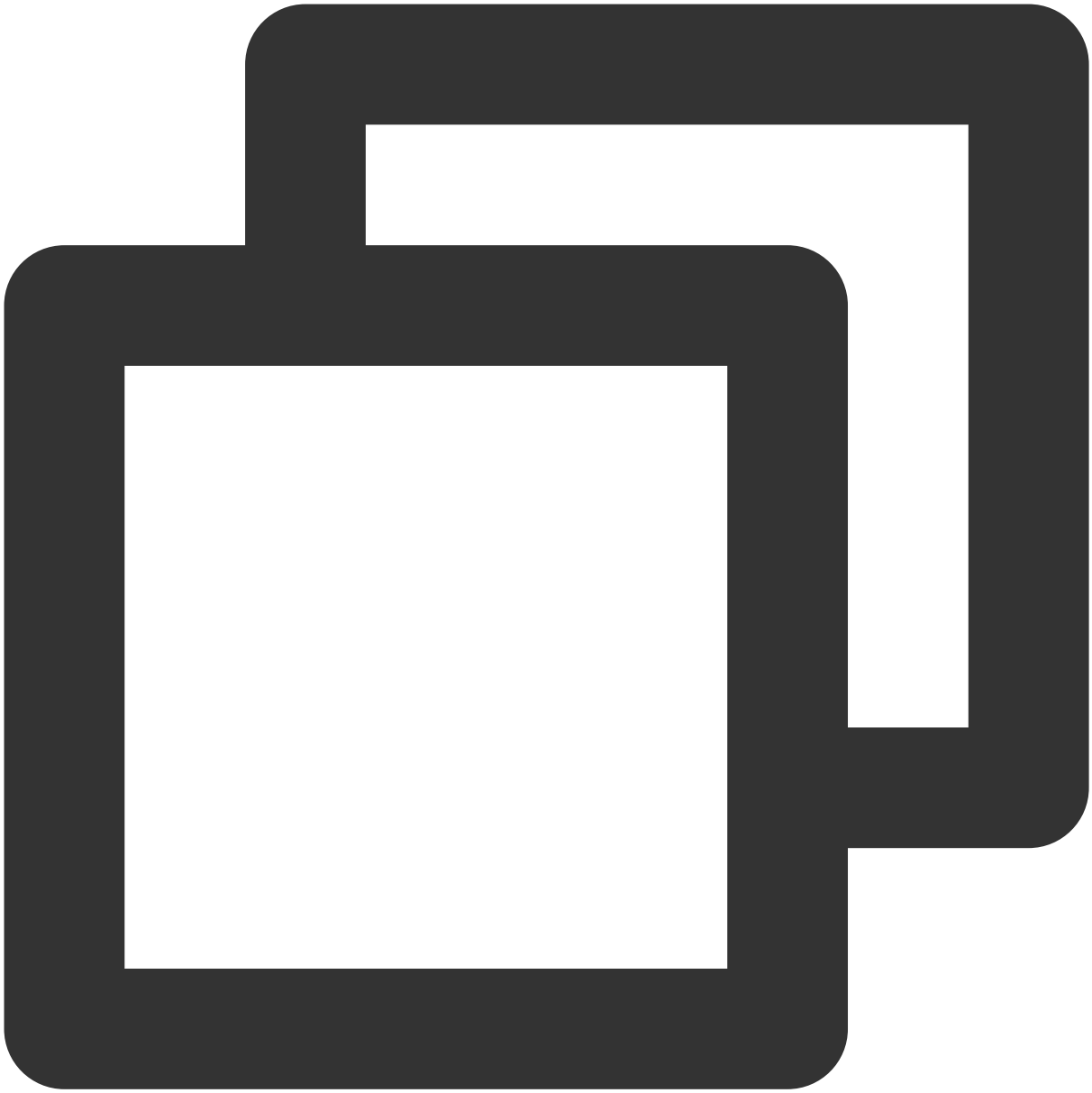
搜索**全部会话**的消息，如果匹配到的消息数量 messageCount > 1，则接口返回的 `messageList` 为 []，您可以在 UI 上展示 【 `${messageCount}` 条相关记录】。如果您想高亮展示匹配到的消息，请参考 指定搜索，将接口返回的 `messageList` 高亮展示。

搜索**全部会话**的消息，如果某个会话中匹配到的消息条数 = 1，则 `messageList` 为匹配到的那条消息。

社群、topic、直播群，不支持搜索云端消息。

搜索云端消息

接口



```
chat.searchCloudMessages(options);
```

参数

参数 options 为 Object 类型，包含的属性值如下：

Name	Type	Description
keywordList	Array | undefined	关键字列表，最多支持 5 个。 **注意：** 当消息发送者以及消息类型均未指定时，关键字列表必须非空；否则，关

keywordListMatchType	String | undefined	关键字列表匹配类型 or：“或”关系搜索（默认） and：“与”关系搜索
senderUserIDList	Array | undefined	指定 userID 发送的消息，最多支持 5 个。
messageTypeList	Array | undefined	指定搜索的消息类型集合，默认搜索全部类型。 不传入时，表示搜索支持的全部类型消息（`TencentCloudChat.TYPES.MSG_FACE`、`TencentCloudChat` 和 `TencentCloudChat.TYPES.MSG_GRP_SYS_NOTICE` 不支持） 传值时，取值详见 TencentCloudChat.TYPES。
conversationID	String | undefined	搜索“全部会话”还是搜索“指定的会话”，不传入时，表示全部会话，默认方式： `C2C${userID}`（单聊） `GROUP${groupID}`（群聊） 社群、topic、直播群，不支持搜索云端消息
timePosition	Number | undefined	搜索的起始时间点。默认为 0 即代表从现在开始搜索，单位：秒。
timePeriod	Number | undefined	从起始时间点开始的过去时间范围，单位秒。默认为 0 即代表不限制时间表过去一天。
cursor	String | undefined	每次云端搜索的起始位置。第一次搜索时不要传入 `cursor`，继续搜索 `searchCloudMessages` 接口返回的 `cursor` 的值 **注意：** 全量搜索时，cursor 的有效期为 2 分钟。

返回值

`Promise`

Name	Type	Description
totalCount	Number	如果搜索指定会话，返回满足搜索条件的消息总数。 如果搜索全部会话，返回满足搜索条件的消息所在的所有会话总数量。
searchResultList	Array	满足搜索条件的消息根据会话 ID 分组。
cursor	String	调用搜索接口续拉时需要填的游标。

其中 `searchResultList` 是个列表，内含搜索结果对象，参数说明如下：

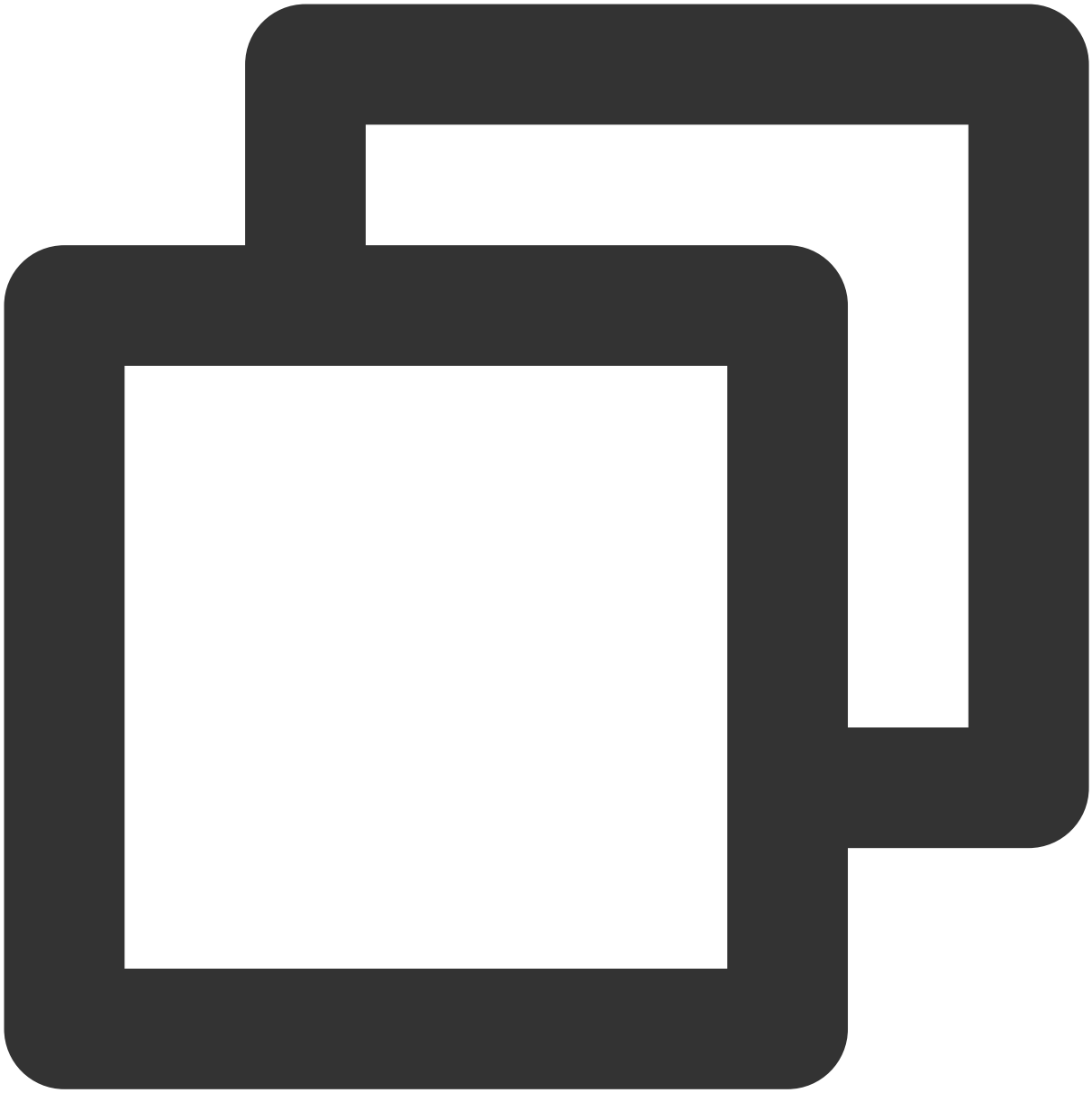
Name	Type	Description

conversationID	String	会话 ID。
messageCount	Number	当前会话一共搜索到了多少条符合要求的消息。
messageList	Array	当前会话中所有满足搜索条件的消息列表 **注意：** 1. 如果搜索指定会话，`messageList` 是本会话中所有满足搜索条件的消息列表。 2. 如果搜索全部会话，`messageList` 中装载的消息条数会有如下两种可能： 如果匹配到的消息数量 `messageCount` > 1，则接口返回的 `messageList` 为 []，您可以在 UI 上展示【${messageCount} 条相关记录】。如果您想高亮展示匹配到的消息，请参考 **指定搜索**，将接口返回的 `messageList` 高亮展示。 如果某个会话中匹配到的消息条数 = 1，则 `messageList` 为匹配到的那条消息。

搜索全部会话的消息

不指定 `conversationID` ，指定关键字搜索。

示例



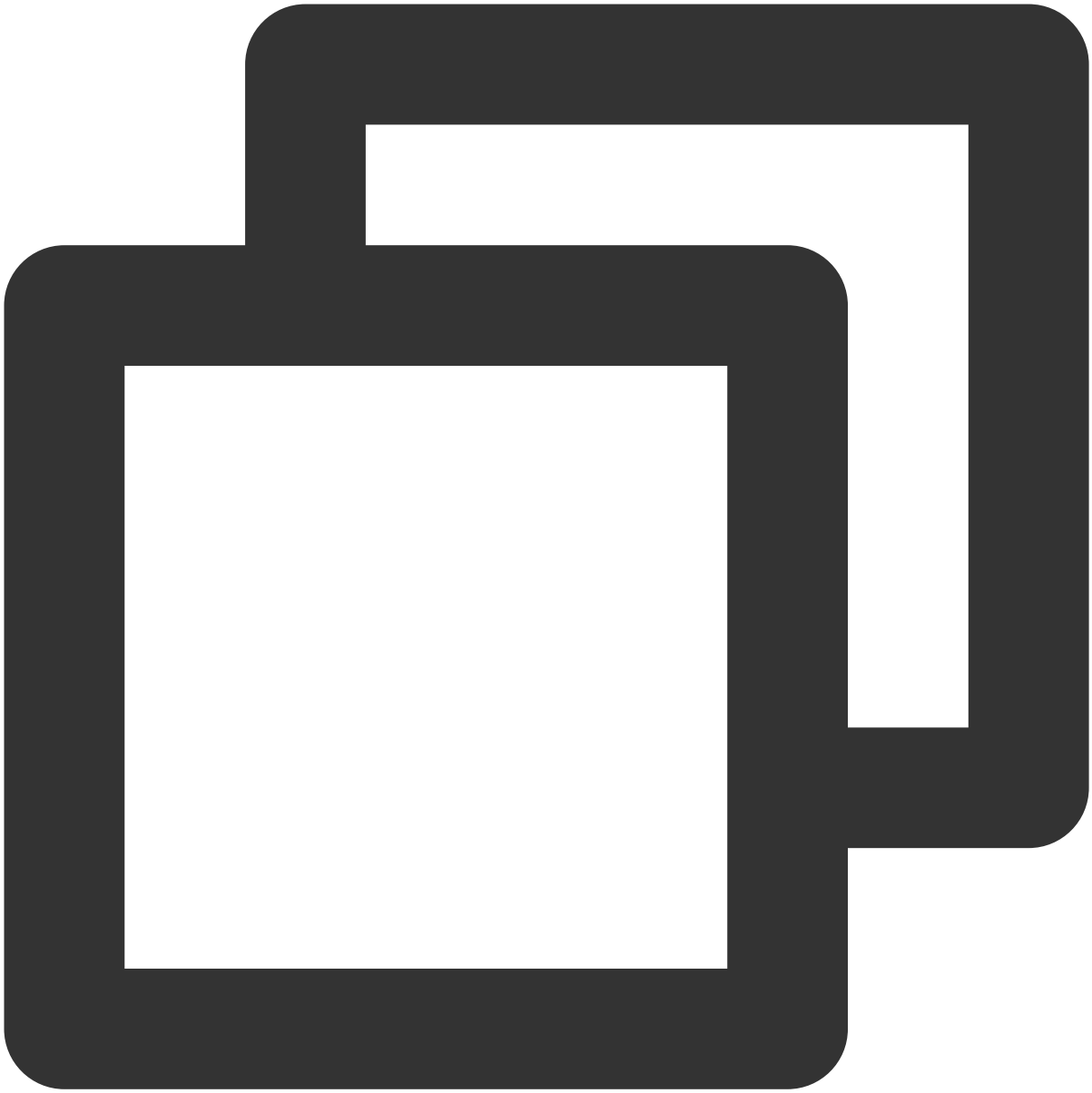
```
// 全量搜索，指定关键字
// - 搜索消息里出现 ‘你好’ 或 ‘在哪里’
let promise = chat.searchCloudMessages({
   keywordList: ['你好', '在哪里']
});
promise.then(function(imResponse) {
 // 搜索消息成功
   const { totalCount, cursor, searchResultList } = imResponse.data;
   console.log(totalCount); // 满足搜索条件的消息所在的所有会话总数量
   console.log(cursor); // 下一次云端搜索的起始位置，如果没有表示搜索结果拉取完成
   console.log(searchResultList); // 满足搜索条件的消息根据会话 ID 分组，分页返回分组结果
```

```
    for (let i = 0; i < searchResultList.length; i++) {
       const searchResultItem = searchResultList[i];
       const { conversationID, messageCount, messageList } = searchResultItem;
       console.log(conversationID); // 会话 ID
       console.log(messageCount); // 当前会话一共搜索到了多少条符合要求的消息
       // 本次搜索【全部会话】，那么 messageList 中装载的消息条数会有如下两种可能：
       // - 如果某个会话中匹配到的消息条数 > 1，则 messageList 为空，您可以在 UI 上显示“ mess
       // - 如果某个会话中匹配到的消息条数 = 1，则 messageList 为匹配到的那条消息，
       // 您可以在 UI 上显示之，并高亮匹配关键词。
       console.log(messageList);
     }
}).catch(function(imError) {
   console.error(imError); // 搜索消息失败
});
```

搜索指定会话的消息

指定 `conversationID` ，指定关键字搜索。

示例



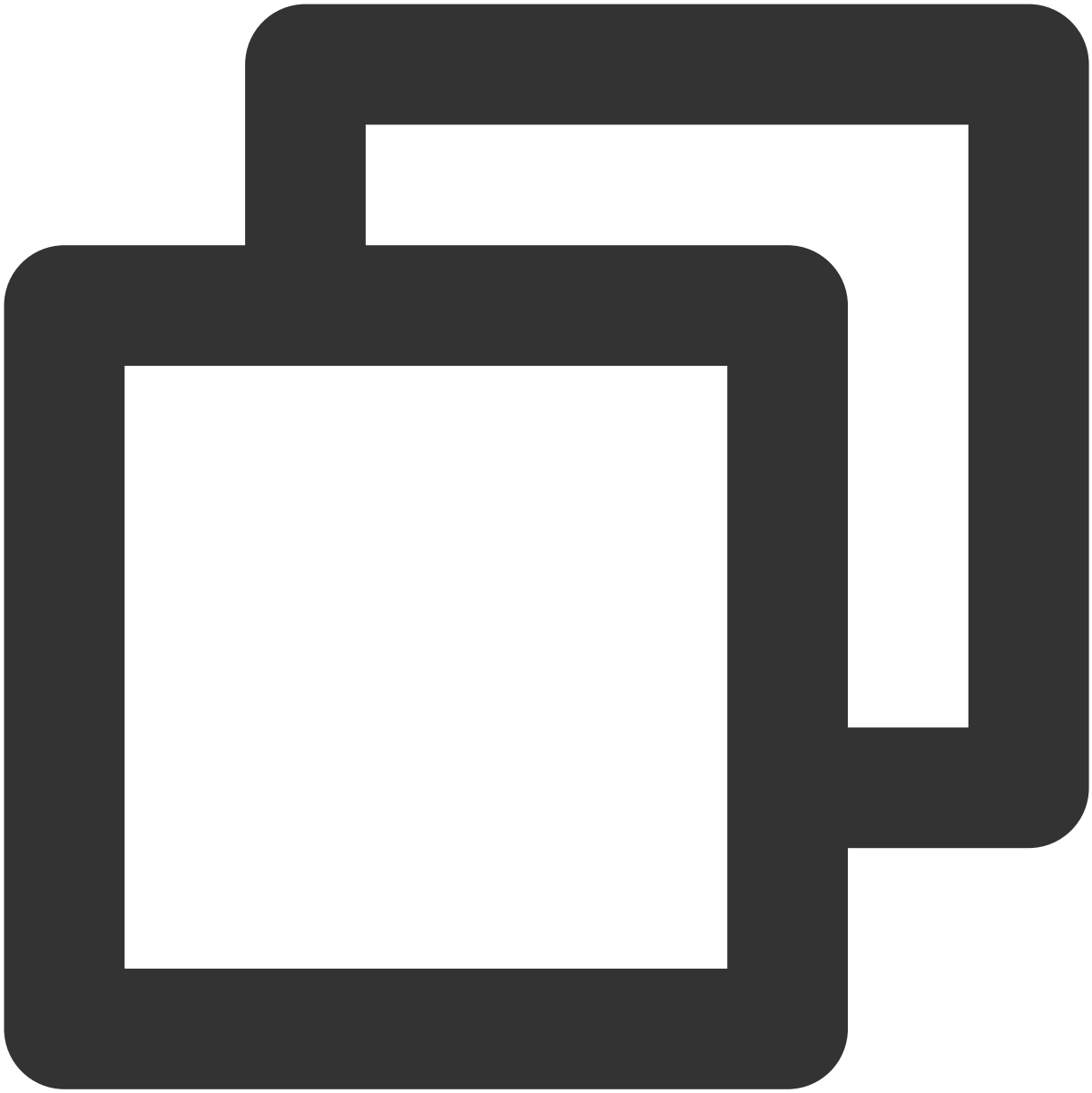
```
// 指定会话，指定关键字搜索
// - 搜索在 'GROUPPublic001' 会话中，消息里出现 ‘你好’ 或 ‘在哪里’ 的消息。
let promise = chat.searchCloudMessages({
   keywordList: ['你好', '在哪里'],
   conversationID: 'GROUPPublic001'
});
promise.then(function(imResponse) {
   // 搜索消息成功
   const { totalCount, cursor, searchResultList } = imResponse.data;
   console.log(totalCount); // 满足搜索条件的消息总数量
   console.log(cursor); // 下一次云端搜索的起始位置，如果没有表示搜索结果拉取完成
```

```
    console.log(searchResultList); // 当前会话搜索的消息结果
    const { conversationID, messageCount, messageList } = searchResultList[0];
    console.log(conversationID); // 会话ID
    console.log(messageCount); // 当前会话一共搜索到了多少条符合要求的消息
    console.log(messageList); // 本会话中所有满足搜索条件的消息列表
}).catch(function(imError); {
    console.error(imError); // 搜索消息失败
});
```

搜索自定义消息

1. 使用接口 createCustomMessage 来创建自定义消息时，需要把搜索的文本放到 `description` 参数中。支持关键词与 `description` 进行匹配。
2. 指定 `messageTypeList` 为 `TencentCloudChat.TYPES.MSG_CUSTOM` 做分类搜索，此时会搜索出所有自定义消息。

示例



```
// 全量搜索，指定关键字和消息类型搜索
// - 搜索消息类型为‘自定义消息’，且消息的 description 里出现了‘你好’或‘在哪里’的消息
let promise = chat.searchCloudMessages({
   keywordList: ['你好', '在哪里'],
   messageTypeList: [TencentCloudChat.TYPES.MSG_CUSTOM],
});
promise.then(function(imResponse) {
 // 搜索消息成功
   const { totalCount, cursor, searchResultList } = imResponse.data;
   console.log(totalCount); // 满足搜索条件的消息所在的所有会话总数量
   console.log(cursor); // 下一次云端搜索的起始位置，如果没有表示搜索结果拉取完成
```

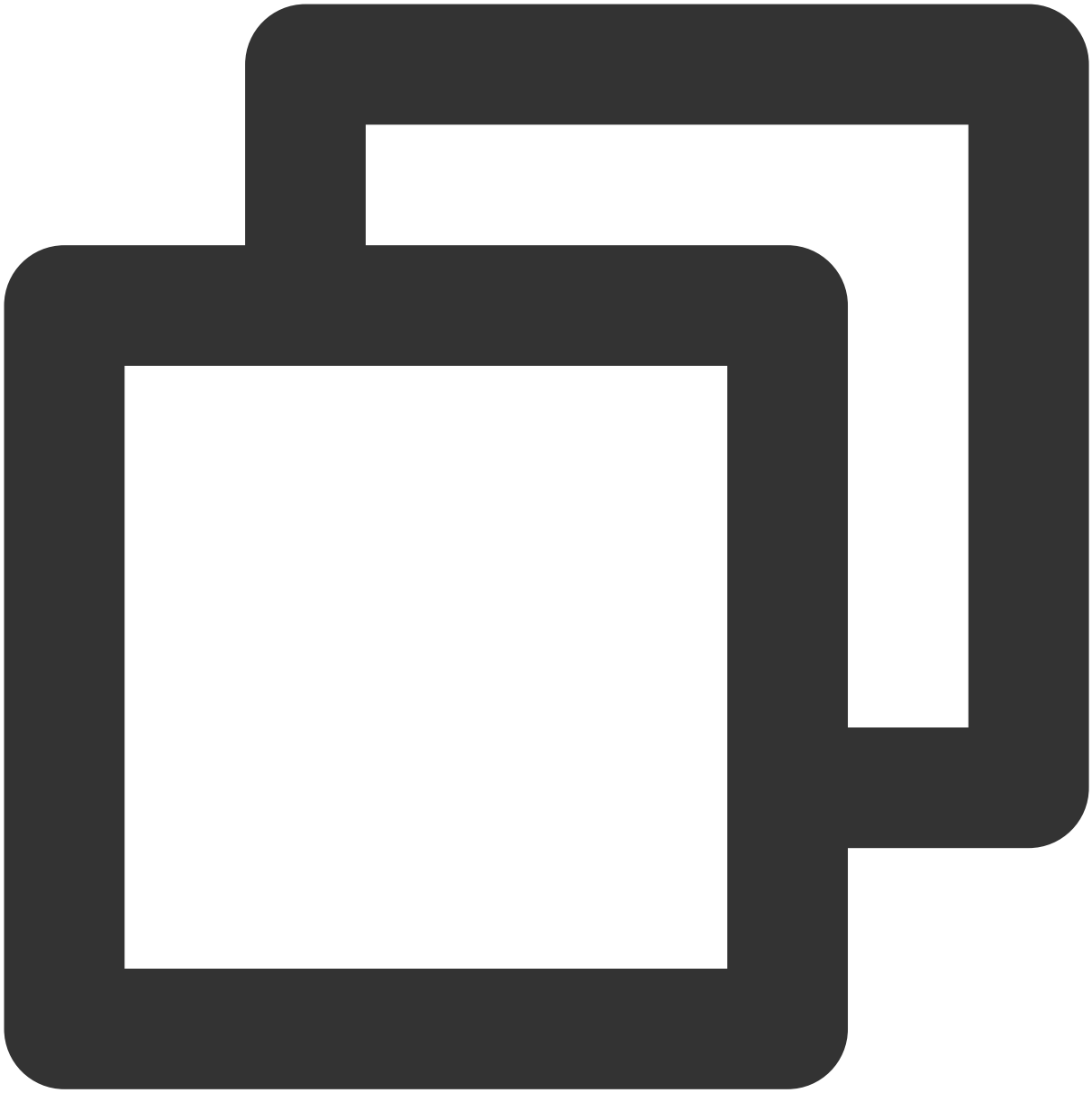
```
    console.log(searchResultList); // 满足搜索条件的消息根据会话 ID 分组，分页返回分组结果
    for (let i = 0; i < searchResultList.length; i++) {
       const searchResultItem = searchResultList[i];
       const { conversationID, messageCount, messageList } = searchResultItem;
       console.log(conversationID); // 会话 ID
       console.log(messageCount); // 当前会话一共搜索到了多少条符合要求的消息
       // 本次搜索【全部会话】，那么 messageList 中装载的消息条数会有如下两种可能：
       // - 如果某个会话中匹配到的消息条数 > 1，则 messageList 为空，您可以在 UI 上显示“ mess
       // - 如果某个会话中匹配到的消息条数 = 1，则 messageList 为匹配到的那条消息，
       // 您可以在 UI 上显示之，并高亮匹配关键词。
       console.log(messageList);
     }
}).catch(function(imError) {
    console.error(imError); // 搜索消息失败
});
```

搜索富媒体消息

富媒体消息包含文件、图片、语音、视频消息。

1. 对于文件消息，界面通常显示文件名。如果调用 `createFileMessage` 创建文件消息时传入 `fileName` 参数， `fileName` 会作为文件消息被搜索的内容，与搜索关键词进行匹配。
2. 对于图片、语音、视频消息，并没有类似 `fileName` 这种名称，界面通常显示缩略图或时长，此时指定 `keywordList` 搜索无效。
3. 可以指定 `messageTypeList` 为 `TencentCloudChat.TYPES.MSG_IMAGE` / `TencentCloudChat.TYPES.MSG_AUDIO` / `TencentCloudChat.TYPES.MSG_FILE` 做分类搜索，此时会搜索出所有指定类型的消息。

示例



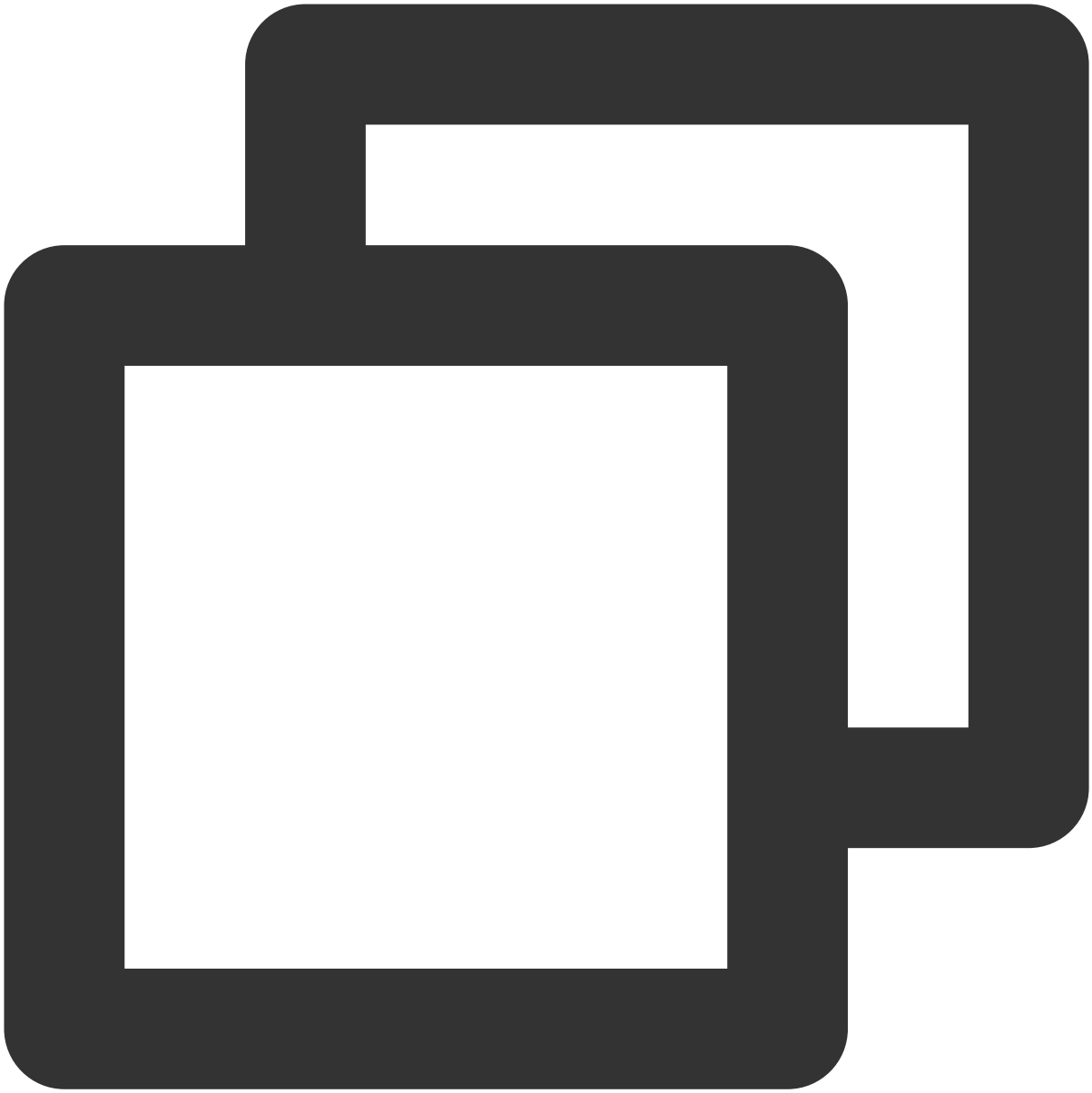
```
// 全量搜索，只指定消息发送者和消息类型搜索（当指定发送者和消息类型时，关键字列表可以为空）
// - 搜索消息发送者为 ‘user1’ 或 ‘user2’，且消息类型为 ‘图片消息’ 、‘语音消息’ 或 ‘文件消息’
let promise = chat.searchCloudMessages({
   senderUserIDList: ['user1', 'user2'],
   messageTypeList: [
      TencentCloudChat.TYPES.MSG_IMAGE,
      TencentCloudChat.TYPES.MSG_AUDIO,
      TencentCloudChat.TYPES.MSG_FILE
   ],
});
promise.then(function(imResponse) {
```

```
  // 搜索消息成功
    const { totalCount, cursor, searchResultList } = imResponse.data;
    console.log(totalCount); // 满足搜索条件的消息所在的所有会话总数量
    console.log(cursor); // 下一次云端搜索的起始位置，如果没有表示搜索结果拉取完成
    console.log(searchResultList); // 满足搜索条件的消息根据会话 ID 分组，分页返回分组结果
    for (let i = 0; i < searchResultList.length; i++) {
       const searchResultItem = searchResultList[i];
       const { conversationID, messageCount, messageList } = searchResultItem;
       console.log(conversationID); // 会话 ID
       console.log(messageCount); // 当前会话一共搜索到了多少条符合要求的消息
       // 本次搜索【全部会话】，那么 messageList 中装载的消息条数会有如下两种可能：
       // - 如果某个会话中匹配到的消息条数 > 1，则 messageList 为空，您可以在 UI 上显示“ mess
       // - 如果某个会话中匹配到的消息条数 = 1，则 messageList 为匹配到的那条消息，
       // 您可以在 UI 上显示之，并高亮匹配关键词。
       console.log(messageList);
     }
}).catch(function(imError) {
    console.error(imError); // 搜索消息失败
});
```

搜索地理位置消息

1. 支持 `latitude` 、 `longitude` 、 `description` 与关键词进行匹配。
2. 指定 `messageTypeList` 为 `TencentCloudChat.TYPES.MSG_LOCATION` 做分类搜索，此时会搜索出所有地理位置消息。

示例



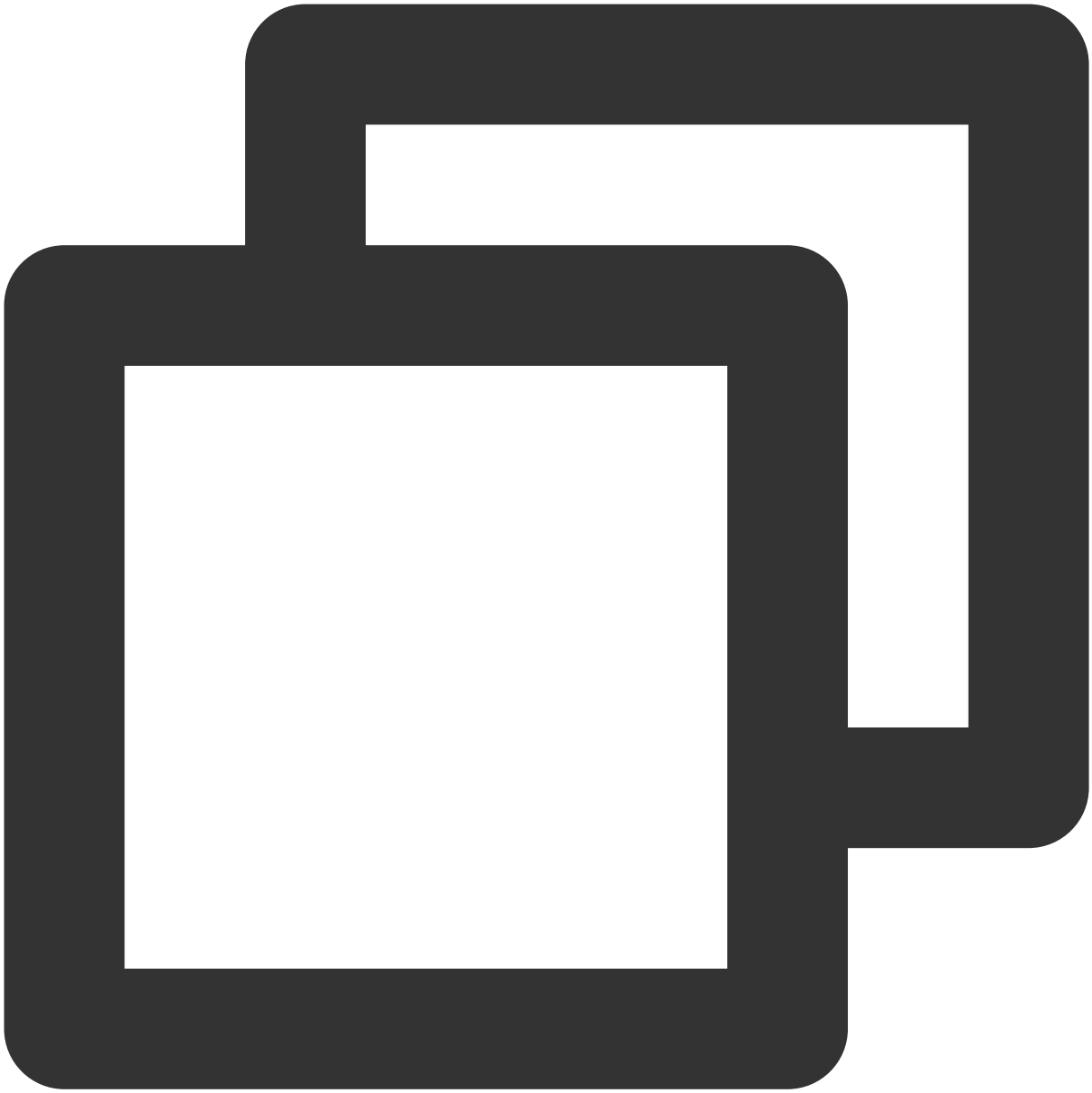
```
// 全量搜索，指定关键字和消息类型搜索
// - 搜索消息类型为‘地理位置’，且消息的 latitude、longitude、description 里出现了‘你好’或
let promise = chat.searchCloudMessages({
   keywordList: ['你好', '在哪里'],
   messageTypeList: [TencentCloudChat.TYPES.MSG_LOCATION],
});
promise.then(function(imResponse) {
 // 搜索消息成功
   const { totalCount, cursor, searchResultList } = imResponse.data;
   console.log(totalCount); // 满足搜索条件的消息所在的所有会话总数量
   console.log(cursor); // 下一次云端搜索的起始位置，如果没有表示搜索结果拉取完成
```

```
    console.log(searchResultList); // 满足搜索条件的消息根据会话 ID 分组，分页返回分组结果
    for (let i = 0; i < searchResultList.length; i++) {
       const searchResultItem = searchResultList[i];
       const { conversationID, messageCount, messageList } = searchResultItem;
       console.log(conversationID); // 会话 ID
       console.log(messageCount); // 当前会话一共搜索到了多少条符合要求的消息
       // 本次搜索【全部会话】，那么 messageList 中装载的消息条数会有如下两种可能：
       // - 如果某个会话中匹配到的消息条数 > 1，则 messageList 为空，您可以在 UI 上显示“ mess
       // - 如果某个会话中匹配到的消息条数 = 1，则 messageList 为匹配到的那条消息，
       // 您可以在 UI 上显示之，并高亮匹配关键词。
       console.log(messageList);
     }
}).catch(function(imError) {
    console.error(imError); // 搜索消息失败
});
```

搜索合并消息

1. 使用接口 createMergerMessage 来创建合并消息时，需要把搜索的文本放到 `title` 或 `abstractList` 参数中，支持 `title` 、 `abstractList` 与关键词进行匹配。
2. 指定 `messageTypeList` 为 `TencentCloudChat.TYPES.MSG_MERGER` 做分类搜索，此时会搜索出所有合并消息。

示例



```
// 全量搜索，指定关键字和消息类型搜索
// - 搜索消息类型为‘合并消息’，且消息的 title 或 abstractList 里出现了‘你好’或‘在哪里’的
let promise = chat.searchCloudMessages({
   keywordList: ['你好', '在哪里'],
   messageTypeList: [TencentCloudChat.TYPES.MSG_MERGER],
});
promise.then(function(imResponse) {
 // 搜索消息成功
   const { totalCount, cursor, searchResultList } = imResponse.data;
   console.log(totalCount); // 满足搜索条件的消息所在的所有会话总数量
   console.log(cursor); // 下一次云端搜索的起始位置，如果没有表示搜索结果拉取完成
```

```
    console.log(searchResultList); // 满足搜索条件的消息根据会话 ID 分组，分页返回分组结果
    for (let i = 0; i < searchResultList.length; i++) {
       const searchResultItem = searchResultList[i];
       const { conversationID, messageCount, messageList } = searchResultItem;
       console.log(conversationID); // 会话 ID
       console.log(messageCount); // 当前会话一共搜索到了多少条符合要求的消息
       // 本次搜索【全部会话】，那么 messageList 中装载的消息条数会有如下两种可能：
       // - 如果某个会话中匹配到的消息条数 > 1，则 messageList 为空，您可以在 UI 上显示“ mess
       // - 如果某个会话中匹配到的消息条数 = 1，则 messageList 为匹配到的那条消息，
       // 您可以在 UI 上显示之，并高亮匹配关键词。
       console.log(messageList);
     }
}).catch(function(imError) {
    console.error(imError); // 搜索消息失败
});
```