CodeChatSDK代码示例

目录

初始化工作

```
//获取实例
Client client = Client.Instance;

//配置客户端信息
client.ServerHost = "39.97.250.50:6061";
client.SetHttpApi("http://39.97.250.50:6060",
"AQEAAAABAAD_rAp4DJh05a1HAwFT3A6K");

//绑定事件
client.LoginSuccessEvent += LoginSuccess;
client.DisconnectedEvent += Disconnected;
client.LoginFailedEvent += LoginFailed;
client.RegisterSuccessEvent += RegisterSuccess;
client.RegisterFailedEvent += RegisterFailed;
client.AddMessageEvent += AddeMessage;
client.RemoveMessageEvent += RemoveMessage;
client.AddTopicEvent += AddTopic;
client.RemoveTopicEvent += RemoveTopic;
client.AddSubscriberEvent += AddSubscriber;
client.RemoveSubscriberEvent += RemoveSubscriber;
client.SubscriberStateChangedEvent += SubscriberStateChange;
//其他事件绑定省略
```

用户

登陆

用户对象与用户控制器对象可保存为界面类的私有变量，供后续操作使用。

```
//获取用户单例
Account account = AccountController.Instance;

//设置用户名信息
account.Username = username;
account.Password = password;

//获取或创建用户文件夹（以UWP文件操作为例）
StorageFolder folder = ApplicationData.Current.LocalFolder;
StorageFolder databaseFolder = null;
try
{
    //存在则获取文件夹
    databaseFolder = await folder.GetFolderAsync(account.Username);
}
catch
{
    //不存在则创建文件夹
    databaseFolder = await folder.CreateFolderAsync(account.Username);
}

//获取数据库存储路径
string databasePath = databaseFolder.Path;

//连接或创建数据库
SqliteAccountRepository database = new SqliteAccountRepository(databasePath);

//创建用户控制器
AccountController accountController = new AccountController();

//设置数据库
await accountController.SetDatabase(database);

//登陆
accountController.Login();
```

注册

```
//获取用户单例
Account account = AccountController.Instance;
```

```
//设置用户信息
account.Username = username;
account.Password = password;
account.FormattedName = formattedName;
account.Email = email;

//获取或创建用户文件夹（以UWP文件操作为例）
StorageFolder folder = ApplicationData.Current.LocalFolder;
StorageFolder databaseFolder = null;
try
{
    //存在则获取文件夹
    databaseFolder = await folder.GetFolderAsync(account.Username);
}
catch
{
    //不存在则创建文件夹
    databaseFolder = await folder.CreateFolderAsync(account.Username);
}

//获取数据库存储路径
string databasePath = databaseFolder.Path;

//连接或创建数据库
SqliteAccountRepository database = new SqliteAccountRepository(databasePath);

//创建用户控制器
AccountController accountController = new AccountController();

//设置数据库
await accountController.SetDatabase(database);

//注册
accountController.Register();
```

发送注册验证码

验证码发送操作建议在RegisterSuccessEvent的绑定函数中执行

```
//验证码
string code = "123456";

//发送注册验证码
accountController.SendVerificationCode(code);
```

修改密码

```
//修改密码
string password = "new_password";
accountController.ChangePassword(password);
```

设置用户名

```
//设置用户名
string formattedName = "new_formatted_name";
accountController.SetFormattedName(formattedName);
```

设置头像

```
//打开文件选取器（以UWP文件操作为例）
FileOpenPicker picker = new FileOpenPicker();
picker.CommitButtonText = "Set";
picker.ViewMode = PickerViewMode.Thumbnail;
picker.SuggestedStartLocation = PickerLocationId.ComputerFolder;
picker.FileTypeFilter.Add("*");

//获取选中文件
var file = await picker.PickSingleFileAsync();
if (file != null)
{
    //获取文件属性
    BasicProperties property = await file.GetBasicPropertiesAsync();

    //转换成Bytes数组
    IBuffer buffer = await FileIO.ReadBufferAsync(file);
    byte[] bytes = buffer.ToArray();

    //设置头像
    accountController.SetAvator(file, property.Size, bytes);
}
```

添加标签

```
//添加标签
string tag = "new_tag";
accountController.AddTag(tag);
```

移除标签

```
//移除标签
string tag = "existed_tag";
accountController.RemoveTag(tag);
```

订阅者

获取订阅者列表

```
//获取订阅者列表
List<Subscriber> subscribers = account.SubscriberList;
```

获取订阅者控制器

```
//通过AccountController获得，未指定特定订阅者，只可用于本地搜索订阅者
SubscriberController subscriberController =
accountController.GetSubscriberController();
```

本地搜索订阅者

```
//返回所有结果
//搜索条件
string condition = "hello";

//搜索范围为所有订阅者
List<Subscriber> subscribers = await
subscriberController.SearchSubscriber(condition);


//分页返回结果
//搜索条件
string condition = "hello";
```

```
//分页页码，从1开始
int pageIndex = 1;
//页面大小
int pageSize = 2;
//页面数量，执行函数后将赋值为页面数量
int pageCount = 0;

//搜索范围为所有订阅者
List<Subscriber> subscribers =
subscriberController.SearchSubscriber(condition, pageIndex, pageSize, ref
pageCount);
```

添加订阅者

```
//添加订阅者
bool result = await accountController.AddSubscriber(subscriber);

//判断结果
if(result == true)
{
    //添加成功操作
}
else
{
    //添加失败操作
    //失败原因为重复添加
}
```

移除订阅者

```
//移除订阅者
bool result = await accountController.RemoveSubscriber(subscriber);

//判断结果
if(result == true)
{
    //移除成功操作
}
else
{
    //移除失败操作
    //失败原因为不存在该订阅者
}
```

在线搜索订阅者

```
//返回所有结果
//搜索条件
string condition = "hello";

//搜索范围为整个服务器
List<Subscriber> subscribers =
accountController.SearchSubscriberOnline(condition);


//分页返回结果
//搜索条件
string condition = "hello";

//分页页码，从1开始
int pageIndex = 1;
//页面大小
int pageSize = 2;
//页面数量，执行函数后将赋值为页面数量
int pageCount = 0;

//搜索范围为整个服务器
List<Subscriber> subscribers =
accountController.SearchSubscriberOnline(condition, pageIndex, pageSize, ref
pageCount);
```

话题

获取话题列表

```
//获取话题列表
List<Topic> topics = account.TopicList;
```

刷新话题列表

```
//刷新话题列表
accountController.RefreshTopicList();
```

获取话题

```
//获取当前话题
//来源于ListView的点击项
Topic currentTopic = itemClickEventArgs.ClickedItem as Topic;

//由话题名寻找话题
Topic currentTopic = accountController.GetTopicByName(topicName);

//由话题列表索引获取话题
Topic currentTopic = accountController.GetTopicAt(index);
```

获取话题控制器

对任何话题进行操作（如移除话题、发送消息、移除消息、设置话题备注）时，都需要通过话题控制器。

```
//通过AccountController获得，未指定特定话题，只可用于本地搜索话题
TopicController topicController = accountController.GetTopicController();

//获取当前话题
Topic currentTopic = accountController.GetTopicByName(topic);

//根据话题获取话题控制器
TopicController topicController = await
accountController.GetTopicController(currentTopic);

//根据话题名获取话题控制器
TopicController topicController = await
accountController.GetTopicControllerByName(topic);
```

本地搜索话题列表

```
//返回所有结果
//搜索条件
string condition = "hello";

//搜索范围为所有话题
List<Topic> topics = await topicController.SearchTopic(condition);


//分页返回结果
//搜索条件
```

```
string condition = "hello";

//分页页码，从1开始
int pageIndex = 1;
//页面大小
int pageSize = 2;
//页面数量，执行函数后将赋值为页面数量
int pageCount = 0;

//搜索范围为所有话题
List<Topic> topics = topicController.SearchTopic(condition, pageIndex,
pageSize, ref pageCount);
```

订阅话题

根据Tinode规定，对话题进行任何操作之前都需要订阅该话题。当通过话题控制器控制话题时，在获取控制器时就已经订阅话题，不需要再手动订阅。

移除话题

```
//移除话题
bool result = await accountController.RemoveTopic(currentTopic);

//判断结果
if(result == true)
{
    //移除成功操作
}
else
{
    //移除失败操作
    //失败原因为不存在该话题
}
```

置顶话题

```
//置顶话题
bool result = await accountController.PinTopic(currentTopic);

//判断结果
if(result == true)
{
    //置顶成功操作
}
else
{
    //置顶失败操作
    //失败原因为不存在该话题或话题已置顶
}
```

取消置顶话题

```
//取消置顶话题
bool result = await accountController.UnpinTopic(currentTopic);

//判断结果
if(result == true)
{
    //取消置顶成功操作
}
else
{
    //取消置顶失败操作
    //失败原因为不存在该话题或话题未置顶
}
```

设置话题备注

```
//设置话题备注
string comment = "new_comment";
await topicController.SetPrivateComment(comment);
```

消息

获取消息列表

```
//获取消息列表
List<ChatMessage> ChatMessageList = currentTopic.MessageList;
```

获取消息控制器

```
//通过AccountController获得，未指定特定话题，只可用于本地搜索全部消息
MessageController messageController =
accountController.GetMessageController();
```

本地搜索消息列表

```
//返回所有结果
//搜索条件
string condition = "hello";

//搜索范围为所有消息
List<ChatMessage> messages = await messageController.SearchMessage(condition);

//搜索范围为当前话题
List<ChatMessage> messages = await
messageController.SearchMessage(currentTopic, condition);


//分页返回结果
//搜索条件
string condition = "hello";
//分页页码，从1开始
int pageIndex = 1;
//页面大小
int pageSize = 2;
//页面数量，执行函数后将赋值为页面数量
int pageCount = 0;

//搜索范围为所有消息
List<ChatMessage> messages = messageController.SearchMessage(condition,
pageIndex, pageSize, ref pageCount);

//搜索范围为当前话题
List<ChatMessage> messages =
messageController.SearchMessage(currentTopic,condition, pageIndex, pageSize,
ref pageCount);
```

发送消息

发送普通消息

```
//消息主体
string messsge = "This is a message from code chat.";

//创建消息对象
ChatMessage chatMessage = new ChatMessage() { Text = message, IsPlainText =
true };

//发送消息
topicController.SendMessage(chatMessage);
```

发送代码

```
//代码类型
CodeType codeType = CodeType.JAVA;
//代码主体
string code = "printf(\"HelloWorld!\n\");";

//通过消息构造器构造消息
ChatMessage message = ChatMessageBuilder.BuildCodeMessage(codeType, code);

//发送消息
topicController.SendMessage(message);
```

发送附件

```
//打开文件选取器（以UWP文件操作为例）
FileOpenPicker picker = new FileOpenPicker();
picker.CommitButtonText = "Send";
picker.ViewMode = PickerViewMode.Thumbnail;
picker.SuggestedStartLocation = PickerLocationId.ComputerFolder;
picker.FileTypeFilter.Add("*");

//获取选中文件
var file = await picker.PickSingleFileAsync();
if (file != null)
{
    //获取文件属性
    BasicProperties property = await file.GetBasicPropertiesAsync();

    //转换成Bytes数组
```

```
    IBuffer buffer = await FileIO.ReadBufferAsync(file);
    byte[] bytes = buffer.ToArray();

    //试上传
    UploadedAttachmentInfo uploadedAttachmentInfo = await client.Upload(file,
property.Size, bytes);

    //判断上传是否成功
    if (uploadedAttachmentInfo != null)
    {
        //附件消息说明（可为空）
        string optionalMessage = "This is an attachment.";

        //运用消息构造器构造消息
        ChatMessage chatMessage =
MessageBuilder.BuildAttachmentMessage(uploadedAttachmentInfo,
optionalMessage);

        //发送消息
      topicController.SendMessage(chatMessage);
  }
  else
  {
        //创建发送消息对象
      ChatMessage chatMessage = new ChatMessage() { Text = "Fail to send.",
IsPlainText = true };

        //发送消息
        topicController.SendMessage(chatMessage);
    }
}
```

移除消息

```
//移除消息
bool result = await topicController.RemoveMessage(currentMessage);

//判断结果
if(result == true)
{
    //移除成功操作
}
else
{
    //移除失败操作
    //失败原因为不存在该消息
}
```

获取历史消息

```
//获取历史消息
topicController.LoadMessage();
```

消息标记为已读

```
//消息标记为已读
topicController.NoteRead(currentMessage);
```

解析消息

普通消息

```
string text = currentMessage.Text;
```

代码

```
//判断是否为代码消息
if (currentMessage.IsCode == true)
{
  CodeType codeType = currentMessage.CodeType;
  string code = currentMessage.Text;
}
```

附件

```
//判断是否为普通消息
if (currentMessage.IsAttachment == true)
{
    //附件说明信息
    string text = currentMessage.Text;

  //通过消息解析器获取URL列表
  List<string> urls =
ChatMessageParser.ParseUrl(currentMessage,client.ApiBaseUrl);

    foreach(string url in urls)
    {
        //遍历URL列表进行操作
    }
}
```

图片Base64

```
//判断是否为普通消息
if (currentMessage.IsPlainText == false)
{
    List<string> base64s = ChatMessageParser.ParseImageBase64(currentMessage);

  foreach(string base64 in base64s)
  {
    //遍历base64列表进行操作
          //可用Converter中提供的方法将base64转为Bitmap对象
          Bitmap image = Converter.ConvertBase64ToImage(base64);
  }

}
```

事件

登陆成功事件

登陆成功后执行该事件，可以通过该事件进行界面的更新，但要注意界面更新的线程安全问题。

```
private async void LoginSuccess(object o, LoginSuccessEventArgs args)
{
    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
() =>
    {
        //执行登陆成功后界面变化相关操作
    });
}
```

登陆失败事件

```
private async void LoginFailed(object sender, LoginFailedEventArgs args)
{

    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
() =>
    {
        //args.Exception.Message可获得失败原因
        //执行登陆失败后界面变化相关操作
    });
}
```

注册成功事件

```
private async void RegisterSuccess(object sendero, RegisterSuccessEventArgs
args)
{
    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
() =>
    {
      //执行注册成功后界面变化相关操作
        //建议跳转至验证码输入界面
  string code = "123456";
  accountController.SendVerificationCode(code);
    });
}
```

注册失败事件

```
private async void RegisterFailed(object sendero, RegisterFailedEventArgs
args)
{
    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
() =>
    {
        //args.Exception.Message可获得失败原因
        //执行注册失败后界面变化相关操作
    });
}
```

添加话题事件

```
private async void AddTopic(object sender, AddTopicEventArgs args)
{
    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
() =>
    {
        //args.Topic可获得添加的话题
        //执行添加话题后界面变化相关操作
    });
}
```

移除话题事件

```
private async void RemoveTopic(object sender, RemoveTopicEventArgs args)
{
    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
() =>
    {
        //args.Topic可获得添加的话题
        //执行移除话题后界面变化相关操作
    });
}
```

添加订阅者事件

```
private async void AddSubscriber(object sender, AddSubscriberEventArgs args)
{
    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
() =>
    {
        //args.Subscriber可获得添加的订阅者
        //执行添加订阅者后界面变化相关操作
    });
}
```

移除订阅者事件

```
private async void RemoveSubscriber(object sender, RemoveSubscriberEventArgs
args)
{
    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
() =>
    {
        //args.Subscriber可获得移除的订阅者
        //执行移除订阅者后界面变化相关操作
    });
}
```

订阅者状态变化事件

```
private async void SubscriberStateChange(object sender,
SubscriberStateChangedEventArgs args)
{
    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
() =>
    {
      //args.IsOnline可获得状态
        if (args.IsOnline)
        {
            //args.Subscriber可获得变化的订阅者
            //上线界面变化操作
        }
        else
        {
            //args.Subscriber可获得变化的订阅者
            //下线界面变化操作
        }
    });
}
```

添加消息事件

```
private async void AddeMessage(object sender, AddMessageEventArgs args)
{
    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
() =>
    {
        //args.Message可获得添加的消息
        //执行添加消息后界面变化相关操作
    });
}
```

移除消息事件

```
private async void RemoveMessage(object sender, RemoveMessageEventArgs args)
{
    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
() =>
    {
        //args.Message可获得移除的消息
        //执行移除消息后界面变化相关操作
    });
}
```

未连接服务器事件

```
private void Disconnected(object sender, DisconnectedEventArgs args)
{
  await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, ()
=>
    {
        //args.Exception.Message可获得未连接原因
        //执行未连接服务器后界面变化相关操作
    });
}
```

个人资料更新事件

当获取到当前用户的UserId与Tags时调用，界面可不绑定。