封装xmppManager类

```
#import <Foundation/Foundation.h>
#import "XMPPFramework.h"
// 该类主要封装了xmpp的常用方法
@interface XMPPManager : NSObject<XMPPStreamDelegate>
//通信管道，输入输出流@property(nonatomic,strong)XMPPStream
xmppStream;
//单例方法+(XMPPManager )defaultManager;
//登录的方法
-(void)loginwithName:(NSString )userName andPassword:
(NSString )password;
//注册-(void)registerWithName:(NSString )userName andPassword:
(NSString )password;
-(void)logout;
@end

#pragma mark 单例方法的实现
+(XMPPManager )defaultManager{
static XMPPManager manager = nil;
static dispatch_once_t onceToken;
dispatch_once(&onceToken, ^{
manager = [[XMPPManager alloc]init];
});
return manager;
}

-(instancetype)init{
 if ([super init]){
//1.初始化xmppStream，登录和注册的时候都会用到它
self.xmppStream = [[XMPPStream alloc]init];
//设置服务器地址,这里用的是本地地址（可换成公司具体地址）
 self.xmppStream.hostName = @"127.0.0.1";
//    设置端口号
self.xmppStream.hostPort = 5222;
```

```
//    设置代理
[self.xmppStream addDelegate:self
delegateQueue:dispatch_get_main_queue()];
 }
 return self;
}

-(void)logout{
//表示离线不可用
 XMPPPresence *presence = [XMPPPresence
presenceWithType:@"unavailable"];
//向服务器发送离线消息
[self.xmppStream sendElement:presence];
//断开链接
[self.xmppStream disconnect];
}

/*
 1.初始化一个xmppStream
 2.连接服务器（成功或者失败）
 3.成功的基础上，服务器验证（成功或者失败）
 4.成功的基础上，发送上线消息
*/
//
@property(nonatomic)ConnectServerPurpose
connectServerPurposeType;//用来标记连接服务器目的的属性
-(void)loginwithName:
```

```
(NSString *)userName
andPassword:(NSString
*)password
{
    //标记连接服务器的目的

self.connectServerPurpos
eType =
ConnectServerPurposeLogi
n;
    //这里记录用户输入的密
码，在登录（注册）的方法里面使
用
    self.password =
password;
    /**
     *  1.初始化一个
xmppStream
          2.连接服务器（成功或
者失败）
```

```
            3.成功的基础上，服务器验证（成功或者失败）
            4.成功的基础上，发送上线消息
        */


    // *  创建xmppjid（用户）
    // *  @param NSString 用户名，域名，登录服务器的方式（苹果，安卓等）

    XMPPJID *jid = [XMPPJID jidWithUser:userName domain:@"jackwong.local" resource:@"iPhone8"];

self.xmppStream.myJID =
```

```
jid;
    //连接到服务器
    [self
connectToServer];

    //有可能成功或者失败，所以有相对应的代理方法

}
#pragma mark 连接到服务器的方法
-(void)connectToServer{
    //如果已经存在一个连接，需要将当前的连接断开，然后再开始新的连接
    if ([self.xmppStream
isConnected]) {
        [self logout];
    }
    NSError *error =
```

```
nil;
    [self.xmppStream
connectWithTimeout:30.0f
error:&error];
    if (error) {
        NSLog(@"error =
%@",error);
    }
}
#pragma mark 注销方法的实现
-(void)logout{
    //表示离线不可用
    XMPPPresence
*presence =
[XMPPPresence
presenceWithType:@"unava
ilable"];
//    向服务器发送离线消息
    [self.xmppStream
sendElement:presence];
```

```
        //断开链接
        [self.xmppStream
disconnect];
}
#pragma mark xmppStream的
代理方法
//连接服务器失败的方法
-
(void)xmppStreamConnectD
idTimeout:(XMPPStream
*)sender{
    NSLog(@"连接服务器失败
的方法，请检查网络是否正常");
}
//连接服务器成功的方法
-
(void)xmppStreamDidConne
ct:(XMPPStream *)sender
{
    NSLog(@"连接服务器成功
```

```
的方法");
//登录
    if
(self.connectServerPurpo
seType ==
ConnectServerPurposeLogi
n) {
        NSError *error =
nil;
//          向服务器发送密码验
证 //验证可能失败或者成功
//          if ([sender
supportsDeprecatedPlainA
uthentication]) {
                [sender
authenticateWithPassword
:self.password
error:&error];
//          }
//          NSLog(@"-----
```

```
%@",self.password);
    }
//注册
    else{
        //向服务器发送一个密码注册（成功或者失败）
        [sender registerWithPassword:self.password error:nil];
    }
}


//验证成功的方法
-(void)xmppStreamDidAuthenticate:(XMPPStream *)sender
{
    NSLog(@"验证成功的方
```

```
法");
    /**
     *  unavailable 离线
       available  上线
       away  离开
       do not disturb 忙
碌
     */
    XMPPPresence
*presence =
[XMPPPresence
presenceWithType:@"avail
able"];
    [self.xmppStream
sendElement:presence];
}
//验证失败的方法
-(void)xmppStream:
(XMPPStream *)sender
didNotAuthenticate:
```

```
(DDXMLElement *)error
{
    NSLog(@"验证失败的方法,请检查你的用户名或密码是否正确,%@",error);
}


#pragma mark 注册
-(void)registerWithName:(NSString *)userName andPassword:(NSString *)password{
    self.password = password;
    //0.标记连接服务器的目的

self.connectServerPurposeType = ConnectServerPurposeRegi
```

```
ster;
    //1. 创建一个jid
    XMPPJID *jid =
[XMPPJID
jidWithUser:userName
domain:@"jackwong.local"
resource:@"iPhone8"];
    //2.将jid绑定到
xmppStream

self.xmppStream.myJID =
jid;
    //3.连接到服务器
    [self
connectToServer];

}

#pragma mark 注册成功的方法
-
```

```
(void)xmppStreamDidRegis
ter:(XMPPStream
*)sender{
    NSLog(@"注册成功的方
法");

}
#pragma mark 注册失败的方法
-(void)xmppStream:
(XMPPStream *)sender
didNotRegister:
(DDXMLElement *)error
{
    NSLog(@"注册失败执行的
方法");
}

//好友管理
@property(nonatomic,stro
ng)XMPPRoster
```

```
*xmppRoster;
//信息归档的上下文
@property(nonatomic,stro
ng)NSManagedObjectContex
t
*messageArchivingContext
;

//2.好友管理
    //获得一个存储好友的
CoreData仓库，用来数据持久化

XMPPRosterCoreDataStorag
e *rosterCoreDataStorage
=
[XMPPRosterCoreDataStora
ge sharedInstance];
    //初始化xmppRoster
    self.xmppRoster =
[[XMPPRoster
```

```
alloc]initWithRosterStorage:rosterCoreDataStorage
dispatchQueue:dispatch_get_main_queue()];
    //激活
    [self.xmppRoster activate:self.xmppStream];
    //设置代理
    [self.xmppRoster addDelegate:self delegateQueue:dispatch_get_main_queue()];

    //3.保存聊天记录
    //初始化一个仓库

XMPPMessageArchivingCoreDataStorage
```

```
*messageStorage =
[XMPPMessageArchivingCor
eDataStorage
sharedInstance];
    //创建一个消息归档对象

self.xmppMessageArchivin
g =
[[XMPPMessageArchiving
alloc]initWithMessageArc
hivingStorage:messageSto
rage
dispatchQueue:dispatch_g
et_main_queue()];
    //激活

[self.xmppMessageArchivi
ng
activate:self.xmppStream
];
```

```
    //上下文

self.messageArchivingCon
text =
messageStorage.mainThrea
dManagedObjectContext;


// 收到好友请求执行的方法
-(void)xmppRoster:
(XMPPRoster *)sender
didReceivePresenceSubscr
iptionRequest:
(XMPPPresence
*)presence{
//    self.fromJid =
presence.from;
    UIAlertView *alert =
[[UIAlertView
alloc]initWithTitle:@"提
```

```
示:有人添加你"
message:presence.from.us
er  delegate:self
cancelButtonTitle:@"拒绝"
otherButtonTitles:@"OK",
nil];
    [alert show];
}


#pragma mark 开始检索好友列
表的方法
-
(void)xmppRosterDidBegin
Populating:(XMPPRoster
*)sender {
    NSLog(@"开始检索好友列
表");
}
```

```
#pragma mark 正在检索好友列表的方法
-(void)xmppRoster:(XMPPRoster *)sender didRecieveRosterItem:(DDXMLElement *)item{    NSLog(@"每一个好友都会走一次这个方法");
    //获得item的属性里的jid字符串，再通过它获得jid对象
    NSString *jidStr = [[item attributeForName:@"jid"] stringValue];    XMPPJID *jid = [XMPPJID jidWithString:jidStr];
    //是否已经添加
    if ([self.rosterJids containsObject:jid])
{        return;
```

```
        }
        //将好友添加到数组中去
        [self.rosterJids
addObject:jid];
        //添加完数据要更新UI（表
视图更新）
        NSIndexPath
*indexPath =
[NSIndexPath
indexPathForRow:self.ros
terJids.count-1
inSection:0];
        [self.tableView
insertRowsAtIndexPaths:@
[indexPath]
withRowAnimation:UITable
ViewRowAnimationAutomati
c];
}
```

```
#pragma mark 删除好友执行的方法
-(void)tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath *)indexPath{
    if (editingStyle==UITableViewCellEditingStyleDelete) {
        //找到要删除的人
        XMPPJID *jid = self.rosterJids[indexPath.row];
        //从数组中删除
        [self.rosterJids
```

```
removeObjectAtIndex:inde
xPath.row];
            //从Ui单元格删除
            [tableView
deleteRowsAtIndexPaths:@
[indexPath]
withRowAnimation:UITable
ViewRowAnimationAutomati
c             ];
            //从服务器删除
            [[XMPPManager
defaultManager].xmppRost
er removeUser:jid];     }
}
```