用户绑定、解绑消息处理都是走的 **BindUserHandler** 类

```
//ConnectionServer#init()
messageDispatcher.register(Command.BIND, () -> new BindUserHandler(mPushServer));
messageDispatcher.register(Command.UNBIND, () -> new BindUserHandler(mPushServer));
```

```
//BindUserHandler.java
@Override
public void handle(BindUserMessage message) {
 if (message.getPacket().cmd == Command.BIND.cmd) {
 bind(message);
 } else {
 unbind(message);
 }
}
```

用户绑定：

```
private void bind(BindUserMessage message) {
 //如果没有userId, 返回ErrorMessage给客户端
 if (Strings.isNullOrEmpty(message.userId)) {
 ErrorMessage.from(message).setReason("invalid param").close();
 Logs.CONN.error("bind user failure for invalid param, conn={}", message.getConnection());
 return;
 }
 //1.绑定用户时先看下是否握手成功
 SessionContext context = message.getConnection().getSessionContext();
 if (context.handshakeOk()) {
 //处理重复绑定问题
 if (context.userId != null) {
 //如果上下文中已经存在绑定的相同userId, 则发送OkMessage绑定成功消息给客户端
 if (message.userId.equals(context.userId)) {
 context.tags = message.tags;
 OkMessage.from(message).setData("bind success").sendRaw();
 Logs.CONN.info("rebind user success, userId={}, session={}", message.userId, context);
 return;
 } else {
```

```
20   //如果与上下文中的userId不相同，则解绑，把已经绑定的用户挤下去，然后绑定当前用
户
21   unbind(message);
22   }
23   }
24
25   //验证用户身份
26   boolean success = bindValidator.validate(message.userId, message.data);
27   if (success) {
28   //2.如果握手成功，就把用户链接信息注册到路由中心，本地和远程各一份
29   success = routerCenter.register(message.userId,
message.getConnection());
30   }
31
32   if (success) {
33   context.userId = message.userId;
34   context.tags = message.tags;
35   EventBus.post(new UserOnlineEvent(message.getConnection(), message.user
Id));
36   OkMessage.from(message).setData("bind success").sendRaw();
37   Logs.CONN.info("bind user success, userId={}, session={}", message.user
Id, context);
38   } else {
39   //3.注册失败再处理下，防止本地注册成功，远程注册失败的情况，只有都成功了才叫成
功
40   routerCenter.unRegister(message.userId, context.getClientType());
41   ErrorMessage.from(message).setReason("bind failed").close();
42   Logs.CONN.info("bind user failure, userId={}, session={}", message.user
Id, context);
43   }
44   } else {
45   ErrorMessage.from(message).setReason("not handshake").close();
46   Logs.CONN.error("bind user failure not handshake, userId={}, conn={}",
message.userId, message.getConnection());
47   }
48  }
```

验证用户身份

```
1  @Spi(order = 1)
2  public static class DefaultBindValidatorFactory implements BindValidatorF
actory {
3   private final BindValidator validator = (userId, data) -> true;
```

```
4
5   @Override
6   public BindValidator get() {
7   return validator;
8   }
9  }
```

用户解绑：

```
1  /**
2   * 目前是以用户维度来存储路由信息的，所以在删除路由信息时要判断下是否是同一个设备
3   * 后续可以修改为按设备来存储路由信息。
4   *
5   * @param message
6   */
7  private void unbind(BindUserMessage message) {
8   if (Strings.isNullOrEmpty(message.userId)) {
9   ErrorMessage.from(message).setReason("invalid param").close();
10   Logs.CONN.error("unbind user failure invalid param, session={}", message.getConnection().getSessionContext());
11   return;
12   }
13   //1.解绑用户时先看下是否握手成功
14   SessionContext context = message.getConnection().getSessionContext();
15   if (context.handshakeOk()) {
16   //2.先删除远程路由，必须是同一个设备才允许解绑
17   boolean unRegisterSuccess = true;
18   int clientType = context.getClientType();
19   String userId = context.userId;
20   RemoteRouterManager remoteRouterManager = routerCenter.getRemoteRouterManager();
21   RemoteRouter remoteRouter = remoteRouterManager.lookup(userId, clientType);
22   if (remoteRouter != null) {
23   String deviceId = remoteRouter.getRouteValue().getDeviceId();
24   if (context.deviceId.equals(deviceId)) {//判断是否是同一个设备
25   unRegisterSuccess = remoteRouterManager.unRegister(userId, clientType);
26   }
27   }
28   //3.删除本地路由信息
```

```
29  LocalRouterManager localRouterManager = routerCenter.getLocalRouterMana
ger();
30  LocalRouter localRouter = localRouterManager.lookup(userId,
clientType);
31  if (localRouter != null) {
32  String deviceId = localRouter.getRouteValue().getSessionContext().devic
eId;
33  if (context.deviceId.equals(deviceId)) {//判断是否是同一个设备
34  unRegisterSuccess = localRouterManager.unRegister(userId, clientType) &
& unRegisterSuccess;
35  }
36  }
37
38  //4.路由删除成功，广播用户下线事件
39  if (unRegisterSuccess) {
40  context.userId = null;
41  context.tags = null;
42  EventBus.post(new UserOfflineEvent(message.getConnection(), userId));
43  OkMessage.from(message).setData("unbind success").sendRaw();
44  Logs.CONN.info("unbind user success, userId={}, session={}", userId, co
ntext);
45  } else {
46  ErrorMessage.from(message).setReason("unbind failed").sendRaw();
47  Logs.CONN.error("unbind user failure, unRegister router failure, userId
={}, session={}", userId, context);
48  }
49  } else {
50  ErrorMessage.from(message).setReason("not handshake").close();
51  Logs.CONN.error("unbind user failure not handshake, userId={}, session=
{}", message.userId, context);
52  }
53 }
```

用户在线、离线事件发布：

利用redis的PUB/SUB 在线事件，离线事件；

```
1 public final class UserEventConsumer extends EventConsumer {
2
3  private final MQClient mqClient = MQClientFactory.create();
4
5  private final UserManager userManager;
6
7  public UserEventConsumer(RemoteRouterManager remoteRouterManager) {
```

```
 this.userManager = new UserManager(remoteRouterManager);
 }

  @Subscribe
  @AllowConcurrentEvents
  void on(UserOnlineEvent event) {
  userManager.addToOnlineList(event.getUserId());
  mqClient.publish(ONLINE_CHANNEL, event.getUserId());
  }

  @Subscribe
  @AllowConcurrentEvents
  void on(UserOfflineEvent event) {
  userManager.remFormOnlineList(event.getUserId());
  mqClient.publish(OFFLINE_CHANNEL, event.getUserId());
  }

  public UserManager getUserManager() {
  return userManager;
  }
}
```

```
public interface Topics {
  String ONLINE_CHANNEL = "/mpush/online/";
  String OFFLINE_CHANNEL = "/mpush/offline/";
}
```

用户在线、离线事件订阅：

```
//这个类貌似没有被使用到
public class UserStatusChangeListener implements MQMessageReceiver {
  private static final Logger LOGGER = LoggerFactory.getLogger(UserStatusC
hangeListener.class);
  //只需要一台机器注册online、offline 消息通道
  public UserStatusChangeListener() {
  MQClientFactory.create().subscribe(ONLINE_CHANNEL, this);
  MQClientFactory.create().subscribe(OFFLINE_CHANNEL, this);
  }
  //可以参考RouterChangeListener#receive()方法
  @Override
  public void receive(String channel, Object message) {
```

```
12
13    }
14  }
```