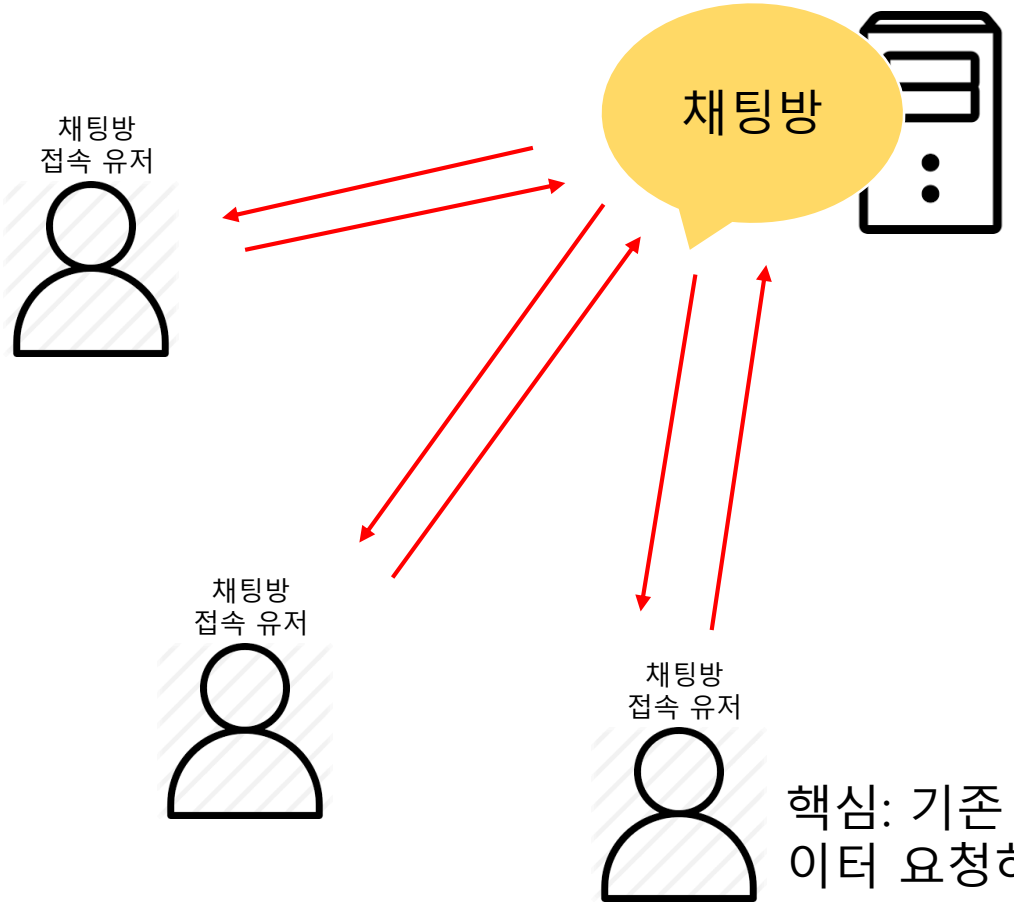
소켓 통신



핵심: 기존 API 요청은 단방향. 유저가 서버로 데이터 요청하고 응답으로 데이터가 옴.

소켓: 유저가 서버로, 서버가 유저에게 이벤트를 알림 → 양방향 통신

실시간 채팅 흐름

1. 유저가 채팅방에 입장
   - 유저가 서버로 join 이벤트 emit → 소켓 통신 시작
   - GET: /{{chatroomId}}/{{userId}}/enter API 요청
     - 읽지 않은 메시지 수 1 감소. (카톡 1 사라지는 것)
     - 서버에서 유저로 join 이벤트 emit과 함께 1 줄어든 메시지 리스트 전송
       - 각 유저들은 이 이벤트와 함께 메시지를 수신하여 UI 갱신
     - 이 API의 최종 응답으로 입장한 유저도 위처럼 1 줄어든 메시지 리스트 수신
       - 입장 유저는 이 응답으로 수신한 메시지로 UI 갱신

실시간 채팅 흐름

2. 유저가 메시지를 전송
   - POST: /{{chatroomId}}/chat API 요청
     - 실시간 소통은 참여하지 않지만 채팅방에 존재하는 유저들은 FCM 수신
     - 서버에서 전송자를 포함한 모든 실시간 참여 유저에게 전송한 메시지와 ‘new message’ 이벤트를 emit
       - 각 유저들은 이 이벤트와 메시지를 수신하여 UI를 갱신. (원래 이 과정에서 다시 유저에서 서버로 메시지를 읽었다는 이벤트를 emit 하여 카운트를 갱신해야 하지만 그러면 너무 복잡해져서 새 메시지 전송할 때 이미 실시간 참여 유저들은 메시지 읽은 것으로 간주하고 보냅니다.)
     - 메시지 전동한 유저는 최종 응답으로 전송 결과 true, false를 받음.

실시간 채팅 흐름

3. 유저가 실시간 채팅을 중단
   - DELETE: /{{chatroomId}}/{{userId}}/leave API 요청
   - socket.disconnect();

클라이언트 측 구현 예시

```
override fun onResume(){
    super.onResume()
    mSocket.on("new message", onNewMessage)
    mSocket.on("join", onJoin)
    mSocket.connect()
    mSocket.emit("join", data)//방 입장
}
override fun onPause() {
    super.onPause()
    api.leaveRoom(chatroomId, userId).enqueue(object:Callback<Void>{
        override fun onResponse(call: Call<Void>, response: Response<Void>) {}
        override fun onFailure(call: Call<Void>, t: Throwable) {}
    })
    adapter.setList(ArrayList())
    mSocket.disconnect()
    mSocket.off("new message", onNewMessage)
    mSocket.off("join", onJoin)

}
```

//채팅방 입장 시
//new message 수신 리스너 등록: 서버->유저로 new message 이벤트가 발생할 시 해당 리스너가 수행됩니다
//join 수신 리스너 등록: 서버->유저로 join 이벤트가 발생할 시 해당 리스너가 수행됩니다
//connect: 소켓 연결
//유저에서 서버로 join 이벤트를 emit 합니다. data에는 userId와 chatroomId 정보 존재

//채팅방 나갈 시

//API 요청

//소켓 통신 종료

//리스너 수신 종료

//리스너 수신 종료

클라이언트 측 구현 예시

```
private val onNewMessage = Emitter.Listener { args ->
    ioScope.launch {
        val item:JSONObject = args[0] as JSONObject;
        val chatId = item.getString("id")
        val sendUserId= item.getString("userId")
        val name = item.getString("name")
        val image = item.getString("image")
        val message = item.getString("message")
        val time = item.getString("createdAt")
        val count = item.getInt("count")

        val chatItem = ChatItem(chatId,sendUserId,name,image,message,count,time)
        if(sendUserId == userId)
            chatItem.viewType= ChatAdapter.MY_CHAT
        else
            chatItem.viewType= ChatAdapter.OTHER_CHAT
        adapter.addItem(chatItem)
        recyclerView.scrollToPosition(adapter.itemCount - 1)
    }
}
```

"new message" 리스너 구현 예시
→ 전달받은 메시지로 UI 갱신

"join" 리스너 구현 예시
→ count가 갱신된 채팅 리스트를 UI 갱신

```
private val onJoin = Emitter.Listener { args ->
    ioScope.launch {
        val chatList = ArrayList<ChatItem>()
        val data :JSONArray?= args[0] as? JSONArray
        var chatId: String
        var sendUserId: String
        var name: String
        var image: String
        var message: String
        var time: String
        var count: Int

        for(i in 0 until data!!.length()){
            val item = data.getJSONObject(i)
            chatId = item.getString("id")
            sendUserId = item.getString("userId")
            name = item.getString("name")
            image = item.getString("image")
            message = item.getString("message")
            time = item.getString("createdAt")
            count = item.getInt("count")
            Log.d("count", "count:"+count)
            val chatItem = ChatItem(chatId,sendUserId,name,image,message,count,time)
            if(sendUserId == userId)
                chatItem.viewType= ChatAdapter.MY_CHAT
            else
                chatItem.viewType= ChatAdapter.OTHER_CHAT
            chatList.add(chatItem)
        }
        adapter.setList(chatList)
    }
}
```