Android Socket通讯实战，客户端与服务端双实现

初学者-Study 郭霖 2022-08-12 08:00 发表于江苏

/ 今日科技快讯 /

近日移动软件公司AppLovin宣布，将以每股58.85美元，总计175.4亿元的价格收购游戏引擎Unity Software，交易将以全股票的形式完成。AppLovin是一家来自加州帕洛阿尔托的移动软件开发商，致力于为应用程序开发人员推销软件平台，以帮助他们寻找客户，并带来收入。

/ 作者简介 /

又到了开心的周五了，祝大家周末愉快！

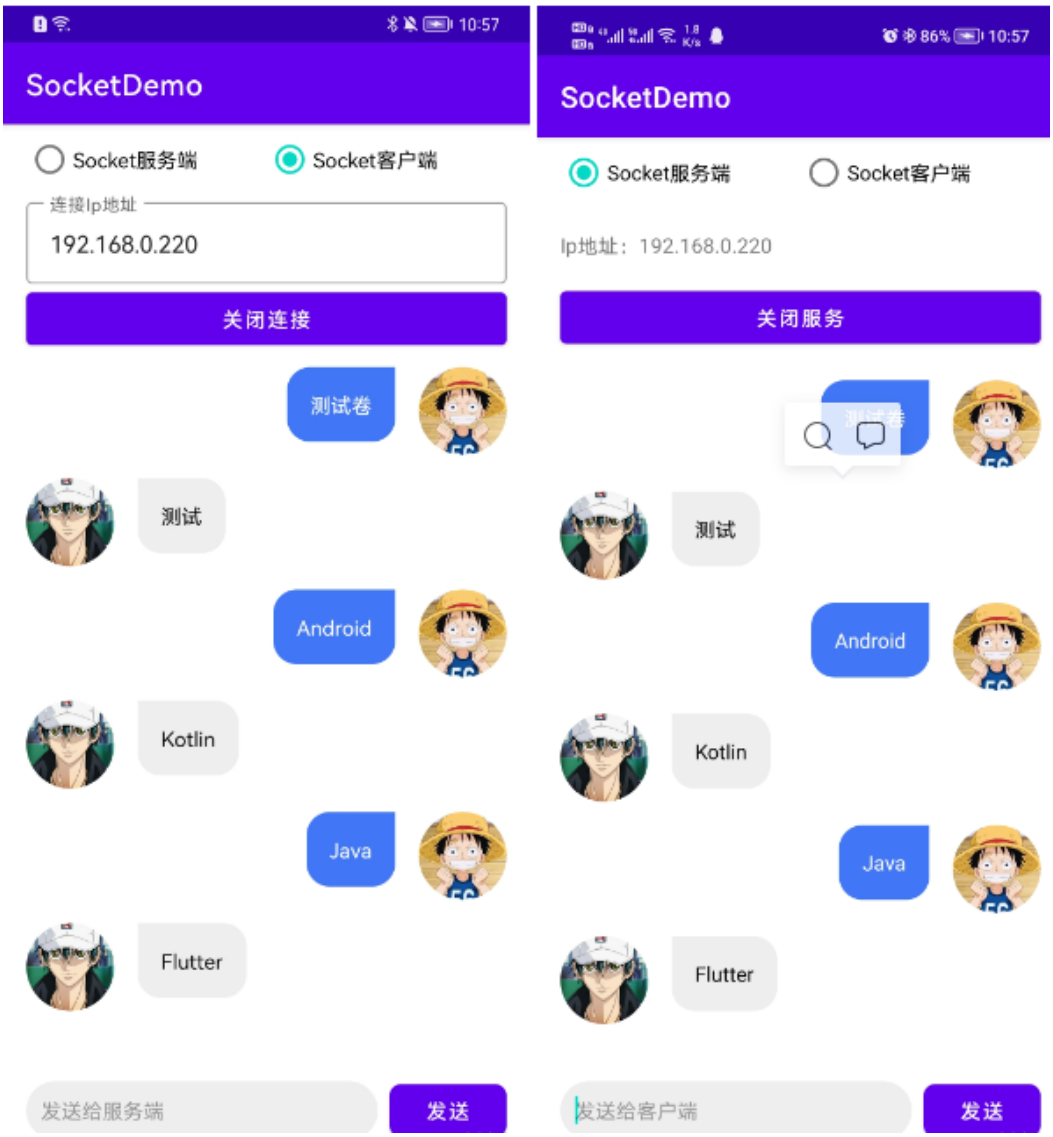
本篇文章来自初学者-Study的投稿，文章主要分享了Android Socket通讯的简单实现！同时也感谢作者贡献的精彩文章。

初学者-Study的博客地址：

> https://llw-study.blog.csdn.net/?type=blog

/ 前言 /

Socket通讯在很多地方都会用到，Android上同样不例外，Socket不是一种协议，而是一个编程调用接口（API），属于传输层，通过Socket，我们才能在Andorid平台上通过 TCP/IP 协议进行开发。先看看效果图：



先说明一下流程：

- ① 准备两台Android手机（真机）。
- ② 连接同一个WIFI网络 。
- ③ 服务端开启服务。
- ④ 客户端连接服务。
- ⑤ 服务端与客户端进行消息发送接收。

那么根据这个流程我们开始写代码。

/ 创建项目 /

创建一个名为SocketDemo的项目，使用Kotlin。

因为涉及到网络通讯，所以需要在AndroidManifest.xml配置网络权限。

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
```

然后再配置一下app的build.gradle，在android{}闭包下添加：

```
buildFeatures {
    viewBinding true
}
```

这里开启项目的viewBinding，其他的就没啥好配置的了，进入正式的编码环节。

/ 构建主页面 /

创建项目会默认有一个MainActivity，这个页面既是服务端，又是客户端。修改一下activity_main.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <RadioGroup
        android:id="@+id/rg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:orientation="horizontal"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent">

        <RadioButton
            android:id="@+id/rb_server"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:checked="true"
```

```
        android:text="Socket服务端" />

    <RadioButton
        android:id="@+id/rb_client"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Socket客户端" />
</RadioGroup>

<LinearLayout
    android:id="@+id/lay_server"
    android:layout_width="match_parent"
    android:layout_height="110dp"
    android:orientation="vertical">

    <TextView
        android:id="@+id/tv_ip_address"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp"
        android:text="Ip地址：" />

    <Button
        android:id="@+id/btn_start_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:text="开启服务"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/radioGroup" />
</LinearLayout>

<LinearLayout
    android:id="@+id/lay_client"
    android:layout_width="match_parent"
    android:layout_height="110dp"
    android:orientation="vertical"
    android:visibility="gone">

    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/op_code_layout"
        style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp">
```

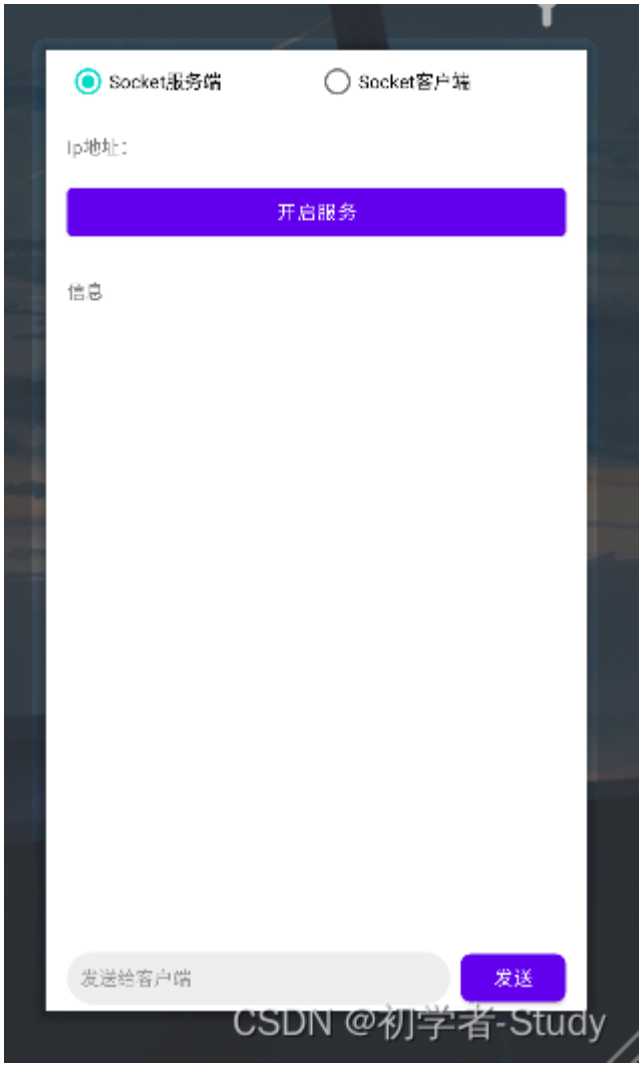
```
        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/et_ip_address"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="连接Ip地址"
            android:inputType="text"
            android:lines="1"
            android:singleLine="true" />
        </com.google.android.material.textfield.TextInputLayout>

        <Button
            android:id="@+id/btn_connect_service"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:layout_marginEnd="16dp"
            android:text="连接服务"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/btn_start_service" />
    </LinearLayout>

    <TextView
        android:id="@+id/tv_info"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:padding="16dp"
        android:text="信息" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:gravity="center_vertical"
        android:paddingStart="16dp"
        android:paddingEnd="16dp">

        <androidx.appcompat.widget.AppCompatEditText
            android:id="@+id/et_msg"
            android:layout_width="0dp"
            android:gravity="center_vertical"
            android:layout_height="40dp"
            android:hint="发送给客户端"
            android:textSize="14sp"
            android:layout_weight="1"
            android:background="@drawable/shape_et_bg"
            android:padding="10dp" />

        <com.google.android.material.button.MaterialButton
            android:id="@+id/btn_send_msg"
```

```
        android:layout_width="80dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:text="发送"
        app:cornerRadius="8dp" />
    </LinearLayout>
</LinearLayout>
```

这里面有一个输入框的背景样式，在drawable下新增shape_et_bg.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <corners android:radius="20dp" />
    <solid android:color="#EEE" />
</shape>
```

说明一下页面的内容，首先是RadioButton切换服务端和客户端，服务端则显示当前手机的IP地址和开启服务按钮，客户端则显示一个输入框，和连接服务按钮。中间主要内容就是服务端和客户端交互的信息，底部是一个输入框和发送消息按钮。预览的效果如图所示：



/ 服务端 /

在com.llw.socket包下新建一个server包，我们服务端的代码就写在这个server包下。新建一个ServerCallback接口，代码如下：

```
interface ServerCallback {
    //接收客户端的消息
    fun receiveClientMsg(success: Boolean, msg: String)
    //其他消息
    fun otherMsg(msg: String)
}
```

下面就是主要的服务端代码了，在server包下新建一个SocketServer类，代码如下：

```
object SocketServer {

    private val TAG = SocketServer::class.java.simpleName
```

```
private const val SOCKET_PORT = 9527

private var socket: Socket? = null
private var serverSocket: ServerSocket? = null

private lateinit var mCallback: ServerCallback

private lateinit var outputStream: OutputStream

var result = true
/**
 * 开启服务
 */
fun startServer(callback: ServerCallback): Boolean {
   mCallback = callback
   Thread {
      try {
         serverSocket = ServerSocket(SOCKET_PORT)
         while (result) {
            socket = serverSocket?.accept()
            mCallback.otherMsg("${socket?.inetAddress} to connected")
            ServerThread(socket!!, mCallback).start()
         }
      } catch (e: IOException) {
         e.printStackTrace()
         result = false
      }
   }.start()
   return result
}

/**
 * 关闭服务
 */
fun stopServer() {
   socket?.apply {
      shutdownInput()
      shutdownOutput()
      close()
   }
   serverSocket?.close()
}

/**
 * 发送到客户端
 */
fun sendToClient(msg: String) {
   Thread {
      if (socket!!.isClosed) {
         Log.e(TAG, "sendToClient: Socket is closed")
```

```
            return@Thread
        }
        outputStream = socket!!.getOutputStream()
        try {
            outputStream.write(msg.toByteArray())
            outputStream.flush()
            mCallback.otherMsg("toClient: $msg")
            Log.d(TAG, "发送到客户端成功")
        } catch (e: IOException) {
            e.printStackTrace()
            Log.e(TAG, "向客户端发送消息失败")
        }
    }.start()
}

class ServerThread(private val socket: Socket, private val callback: ServerCallback) :
    Thread() {

    override fun run() {
        val inputStream: InputStream?
        try {
            inputStream = socket.getInputStream()
            val buffer = ByteArray(1024)
            var len: Int
            var receiveStr = ""
            if (inputStream.available() == 0) {
                Log.e(TAG, "inputStream.available() == 0")
            }
            while (inputStream.read(buffer).also { len = it } != -1) {
                receiveStr += String(buffer, 0, len, Charsets.UTF_8)
                if (len < 1024) {
                    callback.receiveClientMsg(true, receiveStr)
                    receiveStr = ""
                }
            }
        } catch (e: IOException) {
            e.printStackTrace()
            e.message?.let { Log.e("socket error", it) }
            callback.receiveClientMsg(false, "")
        }
    }
}
}
```

代码从上往下看，首先是初始化一些变量，然后就是startServer()函数，在这里进行回调接口的初始化然后开一个子线程进行ServerSocket的构建，构建成功之后会监听连接，得到一个socket，这个socket就是客户端，这里将连接客户端的地址显示出来。

然后再开启一个子线程去处理客户端发送过来的消息。这个地方服务端和客户端差不多，下面看ServerThread中的代码。Socket通讯，发送和接收对应的是输入流和输入流，通过socket.getInputStream()得到输入流，获取字节数据然后转成String，通过接口回调，最后重置变量。

关闭服务就没好说的，代码一目了然。最后就是发送到客户端的sendToClient()函数。接收发送字符串，开启子线程，获取输出流，写入字节数据然后刷新，最后回调到页面。

/ 客户端 /

在com.llw.socket包下新建一个client包，我们客户端的代码就写在这个client包下。新建一个ClientCallback接口，代码如下：

```
interface ClientCallback {
    //接收服务端的消息
    fun receiveServerMsg(msg: String)
    //其他消息
    fun otherMsg(msg: String)
}
```

下面就是主要的客户端代码了，在client包下新建一个SocketClient类，代码如下：

```
object SocketClient {

    private val TAG = SocketClient::class.java.simpleName

    private var socket: Socket? = null

    private var outputStream: OutputStream? = null

    private var inputStreamReader: InputStreamReader? = null

    private lateinit var mCallback: ClientCallback

    private const val SOCKET_PORT = 9527

    /**
     * 连接服务
     */
    fun connectServer(ipAddress: String, callback: ClientCallback) {
        mCallback = callback
        Thread {
```

```
        try {
            socket = Socket(ipAddress, SOCKET_PORT)
            ClientThread(socket!!, mCallback).start()
        } catch (e: IOException) {
            e.printStackTrace()
        }
    }.start()
}

/**
 * 关闭连接
 */
fun closeConnect() {
    inputStreamReader?.close()
    outputStream?.close()
    socket?.apply {
        shutdownInput()
        shutdownOutput()
        close()
    }
    Log.d(TAG, "关闭连接")
}

/**
 * 发送数据至服务器
 * @param msg 要发送至服务器的字符串
 */
fun sendToServer(msg: String) {
    Thread {
        if (socket!!.isClosed) {
            Log.e(TAG, "sendToServer: Socket is closed")
            return@Thread
        }
        outputStream = socket?.getOutputStream()
        try {
            outputStream?.write(msg.toByteArray())
            outputStream?.flush()
            mCallback.otherMsg("toServer: $msg")
        } catch (e: IOException) {
            e.printStackTrace()
            Log.e(TAG, "向服务端发送消息失败")
        }
    }.start()
}

class ClientThread(private val socket: Socket, private val callback: ClientCallback) : T
    override fun run() {
        val inputStream: InputStream?
        try {
            inputStream = socket.getInputStream()
```

```
            val buffer = ByteArray(1024)
            var len: Int
            var receiveStr = ""
            if (inputStream.available() == 0) {
                Log.e(TAG, "inputStream.available() == 0")
            }
            while (inputStream.read(buffer).also { len = it } != -1) {
                receiveStr += String(buffer, 0, len, Charsets.UTF_8)
                if (len < 1024) {
                    callback.receiveServerMsg(receiveStr)
                    receiveStr = ""
                }
            }
        } catch (e: IOException) {
            e.printStackTrace()
            e.message?.let { Log.e("socket error", it) }
            callback.receiveServerMsg( "")
        }
    }
}
}
```

客户端的代码和服务端其实很相似，这里我就简单说明一下，首先就是连接服务，需要输入服务端的ip地址，端口号则是写死的一个端口号，也可以动态去设置。其他的地方和服务端相似。

/ 业务交互 /

现在核心功能代码都写好了，下面怎么样让这些功能和页面串起来，这里因为涉及到用户交互所以会说明的多一点。

① 接口回调

还记得之前的ServerCallback和ClientCallback吗？这两个回调接口因为我们是服务端和客户端在一起的，所以在同一个Activity中去实现接口。

然后实现接口中的方法，在MainActivity中新增如下代码：

```
override fun receiveClientMsg(success: Boolean, msg: String) {

}

override fun otherMsg(msg: String) {

}

override fun receiveServerMsg(msg: String) {

}
```

这里的otherMsg()函数是服务端和客户端共用，因为函数名参数都一致，可以自行修改为不共用。这些函数里面后面会写代码，目前先不管，先实现页面的业务逻辑。

② 服务端和客户端切换

服务端和客户端的切换是会影响整个页面的，首先在MainActivity中定义变量，如下所示：

```
private val TAG = MainActivity::class.java.simpleName

private lateinit var binding: ActivityMainBinding

private val buffer = StringBuffer()

//当前是否为服务端
private var isServer = true

//Socket服务是否打开
private var openSocket = false

//Socket服务是否连接
private var connectSocket = false
```

然后修改一下onCreate()函数，代码如下：

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    initView()
}
```

这里就是实现ViewBinding，然后创建了一个initView()函数，代码如下：

```
private fun initView() {
    binding.tvIpAddress.text = "Ip地址：${getIp()}"
    //服务端和客户端切换
    binding.rg.setOnCheckedChangeListener { _, checkedId ->

    }
    //开启服务/关闭服务 服务端处理
    binding.btnStartService.setOnClickListener {

    }
    //连接服务/断开连接 客户端处理
    binding.btnConnectService.setOnClickListener {

    }
    //发送消息 给 服务端/客户端
    binding.btnSendMsg.setOnClickListener {

    }
}
```

在initView()函数中，首先要做的就是显示当前的Ip地址，不管你是服务端还是客户端，我都会获取Ip地址，因为在你切换时并不会重新获取Ip地址，这里有一个getIp()函数，代码如下：

```
private fun getIp() =
    intToIp((applicationContext.getSystemService(WIFI_SERVICE) as WifiManager).connect

private fun intToIp(ip: Int) =
    "${(ip and 0xFF)}.${(ip shr 8 and 0xFF)}.${(ip shr 16 and 0xFF)}.${(ip shr 24 and 0xFF)
```

这里的WIFI_SERVICE就对应之前在AndroidManifest.xml中配置的WIFI状态读取权限。下面我们完成服务端和客户端切换对UI上的改变。代码如下：

```
    binding.rg.setOnCheckedChangeListener { _, checkedId ->
        isServer = when (checkedId) {
            R.id.rb_server -> true
            R.id.rb_client -> false
            else -> true
        }
        binding.layServer.visibility = if (isServer) View.VISIBLE else View.GONE
        binding.layClient.visibility = if (isServer) View.GONE else View.VISIBLE
```

```
        binding.etMsg.hint = if (isServer) "发送给客户端" else "发送给服务端"
    }
```

这里在对RadioGroup进行选中改变监听，点击RadioButton获取id设置是否为服务端，就是改变isServer的值，默认是服务端。然后就是根据isServer去设置服务端布局和客户端布局的显示状态，同时还需要设置底部输入框的提示文字。

③ 服务开启和关闭

如果当前是服务端，则会看到开启服务按钮，点击按钮的代码如下：

```
    binding.btnStartService.setOnClickListener {
        openSocket = if (openSocket) {
            SocketServer.stopServer();false
        } else SocketServer.startServer(this)
        //显示日志
        showInfo(if (openSocket) "开启服务" else "关闭服务")
        //改变按钮文字
        binding.btnStartService.text = if (openSocket) "关闭服务" else "开启服务"
    }
```

这里根据当前是否开启服务条件去控制是开启服务还是关闭服务，还有一些不严谨，再往下就是一个显示日志的方法和修改按钮显示文字，这里就是页面中部的那个TextView。showInfo()函数代码很简单，如下所示：

```
private fun showInfo(info: String) {
    buffer.append(info).append("\n")
    runOnUiThread { binding.tvInfo.text = buffer.toString() }
}
```

就是字符串拼接，然后显示出来。

④ 服务连接和断开

如果当前是客户端，则会看到连接服务按钮，点击按钮的代码如下：

```
    binding.btnConnectService.setOnClickListener {
        val ip = binding.etIpAddress.text.toString()
        if (ip.isEmpty()) {
            showMsg("请输入Ip地址");return@setOnClickListener
```

```
        }
        connectSocket = if (connectSocket) {
            SocketClient.closeConnect();false
        } else {
            SocketClient.connectServer(ip, this);true
        }
        showInfo(if (connectSocket) "连接服务" else "关闭连接")
        binding.btnConnectService.text = if (connectSocket) "关闭连接" else "连接服务"
    }
```

这里会先检查是否输入IP地址，没有就会提示一下，showMsg()函数代码如下：

```
private fun showMsg(msg: String) = Toast.makeText(this, msg, Toast.LENGTH_SHORT).show()
```

这里还缺少一步检查ip地址是否合规，我就先不做了。ip地址有了地址就会根据connectSocket状态得知当前点击按钮时执行连接还是断开。最后同样时显示日志和修改按钮文字。

⑤ 发送消息

终于到了底部的发送消息处理了，点击按钮的代码如下：

```
    binding.btnSendMsg.setOnClickListener {
        val msg = binding.etMsg.text.toString()
        if (msg.isEmpty()) {
            showMsg("请输入要发送的信息");return@setOnClickListener
        }
        //检查是否能发送消息
        val isSend = if (openSocket) openSocket  else if (connectSocket) connectSocket  else
        if (!isSend) {
            showMsg("当前未开启服务或连接服务");return@setOnClickListener
        }
        if (isServer) SocketServer.sendToClient(msg) else SocketClient.sendToServer(msg)
        binding.etMsg.setText("")
    }
```

检查是否有消息输入，然后是根据当前是否为服务端进行消息发送，发送后清空输入框。

⑥ 显示消息内容

在服务端和客户端连接之后，服务端发送消息之后，客户端收到，客户端发送消息之后，服务端收到。在①中我们实现了接口，现在只要将接口返回的消息显示出来就行了。

```
override fun receiveClientMsg(success: Boolean, msg: String) {
    showInfo("ClientMsg: $msg")
}

override fun otherMsg(msg: String) {
    showInfo(msg)
}

override fun receiveServerMsg(msg: String) {
    showInfo("ServerMsg: $msg")
}
```

那么现在所有的代码都写完了，因为页面的底部是一个输入框，当点击之后会弹出软键盘，此时页面会被顶上去，为了避免这样的问题，修改修改一下我们运行看看效果。

/ UI优化 /

既然现在消息通讯已经可以了，那么我们可不可以做成类似聊天的UI风格呢？当然可以。首先要改变一下UI，先把activity_main.xml中id为tv_info的控件控件删掉，换成RecyclerView，代码如下：

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/rv_msg"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1" />
```

现在MainActivity中肯定会有报错，不过我们先不管它，先写列表适配器的代码。

① 列表适配器

做适配器的话要考虑服务端和客户端的关系，因此和传统的聊天是有区别的。首先在layout下创建一个item_rv_msg.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.androi
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:paddingTop="8dp"
    android:paddingBottom="8dp"
    android:paddingStart="16dp"
    android:paddingEnd="16dp">

    <com.google.android.material.imageview.ShapeableImageView
        android:id="@+id/iv_server"
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:src="@drawable/icon_server"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:shapeAppearanceOverlay="@style/circleImageStyle" />

    <TextView
        android:id="@+id/tv_server_msg"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_weight="1"
        android:background="@drawable/shape_left_msg_bg"
        android:text="123"
        android:textColor="@color/black"
        app:layout_constraintStart_toEndOf="@+id/iv_server"
        app:layout_constraintTop_toTopOf="@+id/iv_server" />

    <TextView
```

```
      android:id="@+id/tv_client_msg"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_marginEnd="16dp"
      android:layout_weight="1"
      android:background="@drawable/shape_right_msg_bg"
      android:text="123"
      android:textColor="@color/white"
      app:layout_constraintEnd_toStartOf="@+id/iv_client"
      app:layout_constraintTop_toTopOf="@+id/iv_client" />

   <com.google.android.material.imageview.ShapeableImageView
      android:id="@+id/iv_client"
      android:layout_width="60dp"
      android:layout_height="60dp"
      android:src="@drawable/icon_client"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintTop_toTopOf="parent"
      app:shapeAppearanceOverlay="@style/circleImageStyle" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

这里用到了两个头像图片。

可以直接去我源码里面拿，同时为了设置圆形头像，我在themes.xml中增加了一个样式，代码如下：

```
   <style name="circleImageStyle">
      <item name="cornerFamily">rounded</item>
      <item name="cornerSize">50%</item>
   </style>
```

然后就是消息的背景样式了，因为是要区分服务端和客户端的，服务端在左，客户端在右。在drawable中新增shape_left_msg_bg.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="#EEE" />
    <corners
        android:bottomLeftRadius="16dp"
        android:bottomRightRadius="16dp"
        android:topRightRadius="16dp" />
    <padding
        android:bottom="16dp"
        android:left="16dp"
        android:right="16dp"
        android:top="16dp" />
</shape>
```

shape_right_msg_bg.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="#4177F6" />
    <corners
        android:bottomLeftRadius="16dp"
        android:bottomRightRadius="16dp"
        android:topLeftRadius="16dp" />
    <padding
        android:bottom="16dp"
        android:left="16dp"
        android:right="16dp"
        android:top="16dp" />
</shape>
```

适配器是需要数据的，因为我们在com.llw.socket包下新增一个数据类，代码如下：

```
data class Message(val type:Int, val msg:String)
```

最后我们写一个适配器，在com.llw.socket包下新增一个MsgAdapter，代码如下：

```
class MsgAdapter(private val messages: ArrayList<Message>) :
    RecyclerView.Adapter<MsgAdapter.ViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int) =
        ViewHolder(ItemRvMsgBinding.inflate(LayoutInflater.from(parent.context), parent, fa
```

```
    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val message = messages[position]
        if (message.type == 1) {
            holder.mView.tvServerMsg.text = message.msg
        } else {
            holder.mView.tvClientMsg.text = message.msg
        }

        holder.mView.ivServer.visibility = if (message.type == 1) View.VISIBLE else View.INVI
        holder.mView.ivClient.visibility = if (message.type == 1) View.INVISIBLE else View.VI
        holder.mView.tvServerMsg.visibility = if (message.type == 1) View.VISIBLE else View.G
        holder.mView.tvClientMsg.visibility = if (message.type == 1) View.GONE else View.VISI
    }

    override fun getItemCount() = messages.size

    class ViewHolder(itemView: ItemRvMsgBinding) : RecyclerView.ViewHolder(itemView.root)
        var mView: ItemRvMsgBinding
        init {
            mView = itemView
        }
    }
}
```

这里就是RecyclerView+ViewBinding的使用方式。根据不同的消息类型设置控件状态就可以了。

② 修改页面逻辑

首先要将适配器和RV绑定起来，在MainActivity中新增如下代码：

```
    //消息列表
    private val messages = ArrayList<Message>()
    //消息适配器
    private lateinit var msgAdapter: MsgAdapter
```

然后在initView()函数中初始化，代码如下：

```
    //初始化列表
    msgAdapter = MsgAdapter(messages)
    binding.rvMsg.apply {
        layoutManager = LinearLayoutManager(this@MainActivity)
```

```
    adapter = msgAdapter
}
```

代码添加位置如下图所示：

然后在MainActivity中新增一个updateList()函数，代码如下：

```
/**
 * 更新列表
 */
private fun updateList(type: Int, msg: String) {
    messages.add(Message(type, msg))
    runOnUiThread {
        (if (messages.size == 0) 0 else messages.size - 1).apply {
            msgAdapter.notifyItemChanged(this)
            binding.rvMsg.smoothScrollToPosition(this)
        }
    }
}
```

添加数据到列表，然后刷新位置，滑动到新增数据位置，将showInfo()函数代码先去掉，然后将btnStartService和btnConnectService按钮的点击事件中的showInfo修改为

showMsg。

然后修改一下这三个回调函数，代码如下：

```
override fun receiveClientMsg(success: Boolean, msg: String) = updateList(2, msg)

override fun receiveServerMsg(msg: String) = updateList(1, msg)

override fun otherMsg(msg: String) {
    Log.d(TAG, msg)
}
```

之前都是调用的showInfo()函数，现在都改了，为了让我们发送消息也能更新列表，在btnSendMsg按钮点击事件中，最后一行增加如下代码：

```
updateList(if (isServer) 1 else 2, msg)
```

那么现在代码就写完了，看看运行的效果。

源码地址：https://github.com/lilongweidev/SocketDemo

推荐阅读：

我的新书，《第一行代码 第3版》已出版！

安卓13来了，快！扶起我来！

模仿Android微信小程序，实现小程序独立任务视图的效果

欢迎关注我的公众号

学习技术或投稿

▼

长按上图，识别图中二维码即可关注

阅读原文

喜欢此内容的人还喜欢

10个优秀的日志分析工具

HACK之道

SpringBoot使用@Async的总结!

后端元宇宙

震惊！Vue和React的组件可以互相使用了？

前端修炼师