**远程控制应用开发指南**

**V1.0**

**2024年7月9日**

# 目 录

[目 录 I](#_Toc171433095)

[第1章 Android Studio工具介绍 1](#_Toc171433096)

[1.1 Android Studio下载安装 1](#_Toc171433097)

[1.2 设备连接 11](#_Toc171433098)

[1.2.1 连接实体设备 11](#_Toc171433099)

[1.2.2 创建虚拟设备 12](#_Toc171433100)

[第2章 远程控制app开发介绍 15](#_Toc171433101)

[2.1 项目结构 15](#_Toc171433102)

[2.2 Android页面布局组件介绍 15](#_Toc171433103)

[2.2.1 LinearLayout 17](#_Toc171433104)

[2.2.2 ImageView 18](#_Toc171433105)

[2.2.3 TextView 18](#_Toc171433106)

[2.2.4 ToggleButton 18](#_Toc171433107)

[2.2.5 RadioGroup 18](#_Toc171433108)

[2.3 Android控制程序 18](#_Toc171433109)

[2.4 MQTT协议介绍 20](#_Toc171433110)

[2.4.1 MQTT协议 20](#_Toc171433111)

[2.4.2 Android使用MQTT 21](#_Toc171433112)

[2.4.3 发送接收数据说明 23](#_Toc171433113)

[附 录 25](#_Toc171433114)

# 第1章 Android Studio工具介绍

## 1.1 Android Studio下载安装

Android Studio是开发Android应用程序的官方IDE，基于Intellij IDEA。可以到Android Studio官网下载最新版本：<https://developer.android.google.cn/studio?hl=zh-cn>。本手册基于版本**Android Studio Giraffe | 2022.3.1**测试通过，以下过程为该版本的安装说明，仅供参考：

（1）双击安装exe文件，点击【Next】，如图1.1.1所示。

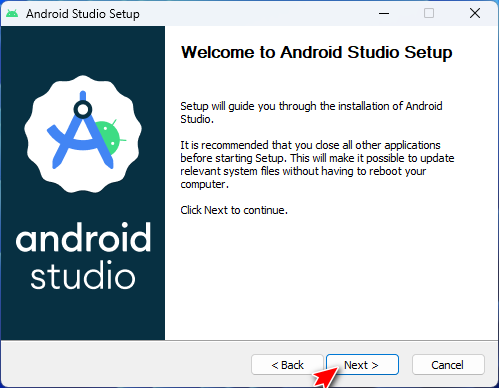


图1.1.1 Android Studio安装步骤1

（2）点击【Next】，如图1.1.2所示：

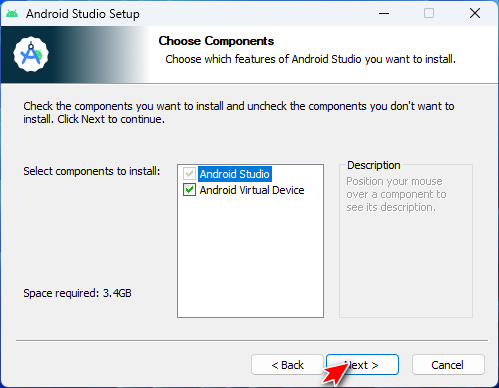


图1.1.2 Android Studio安装步骤2

（3）选择Android Studio安装路径，然后点击【Next】，如图1.1.3所示：

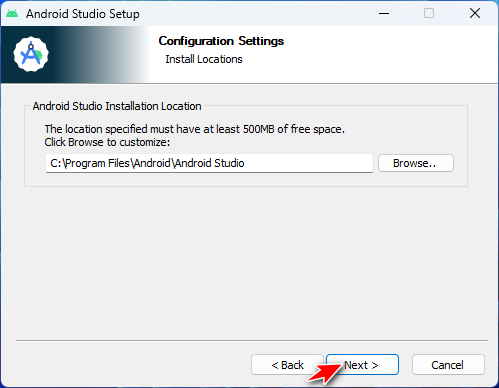


图1.1.3 Android Studio安装步骤3

（4）点击【Install】，等待安装完成，如图1.1.4所示

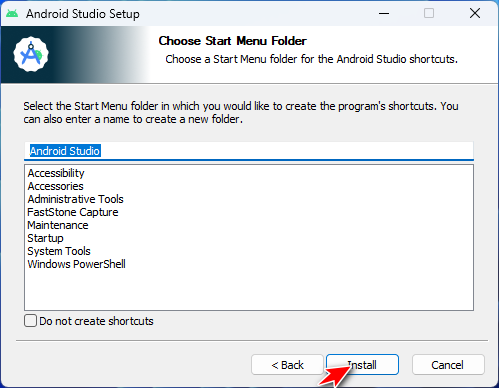


图1.1.4 Android Studio安装步骤4

（5）点击【Next】，如图1.1.5所示：

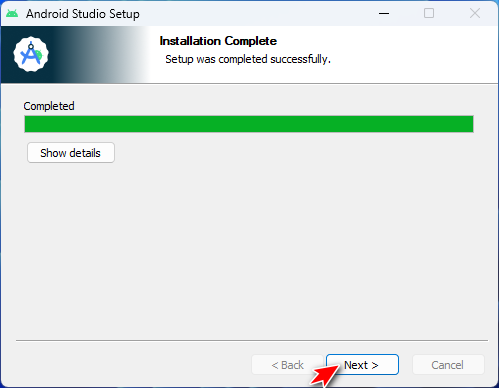


图1.1.5 Android Studio安装步骤5

（6）默认勾选【Start Android Studio】，点击【Finish】，如图1.1.6所示：

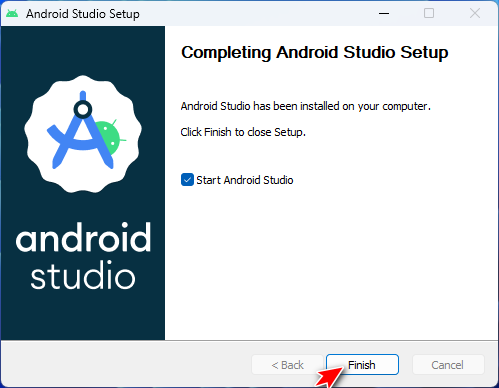


图1.1.6 Android Studio安装步骤6

（7）首次安装，不导入设置，选中【Do not import settings】，然后点击【OK】，如图1.1.7所示：

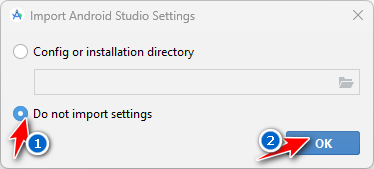


图1.1.7 Android Studio安装步骤7

（8）选择【Don’t send】，如图1.1.8所示：

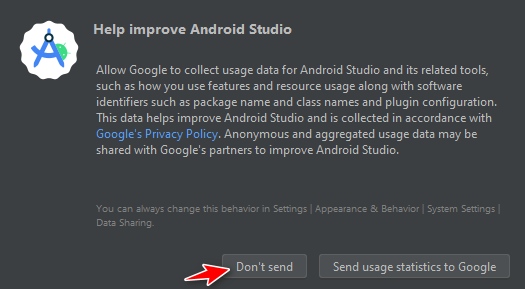


图1.1.8 Android Studio安装步骤8

（9）选择【Cancel】，如图1.1.9所示：

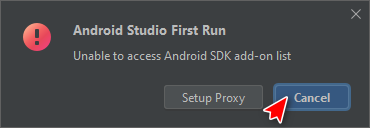


图1.1.9 Android Studio安装步骤9

（10）选择【Next】，如图1.1.10所示：

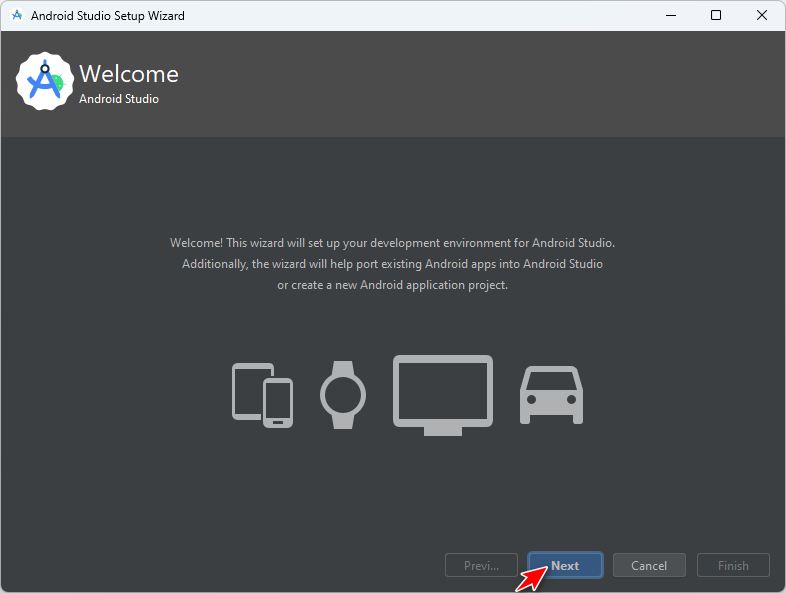


图1.1.10 Android Studio安装步骤10

（11）使用Standard安装，默认选中【Standard】，点击【Next】，如图1.1.11所示：

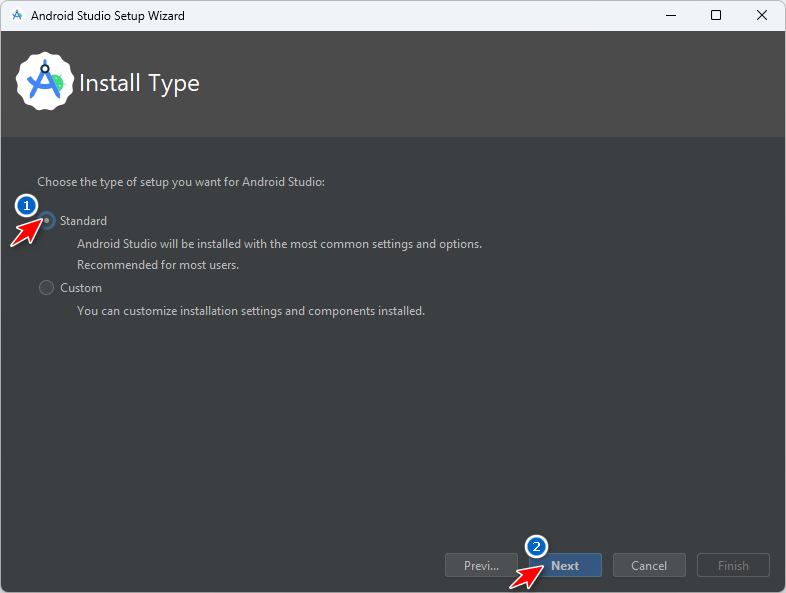


图1.1.11 Android Studio安装步骤11

（12）选择UI主题，暗色或者亮色，然后点击【Next】，如图1.1.12所示

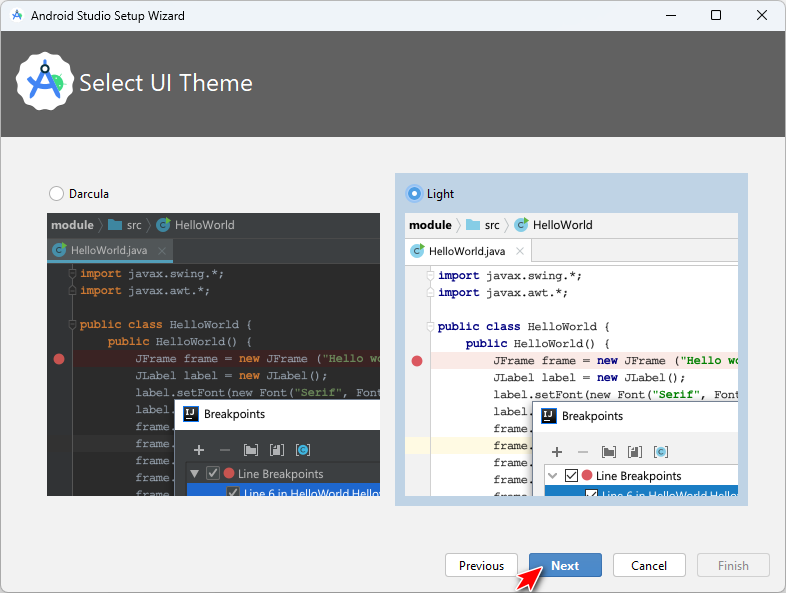


图1.1.12 Android Studio安装步骤12

（13）选择【Next】，如图1.1.13所示：

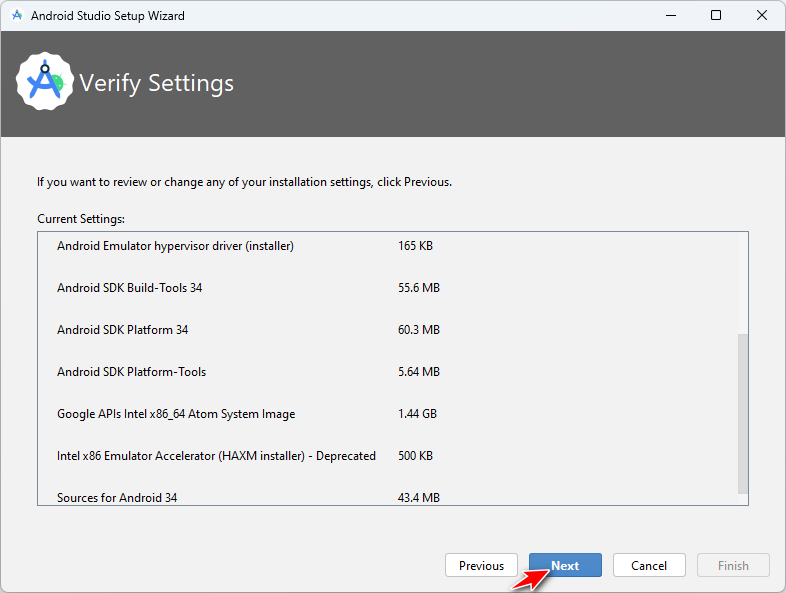


图1.1.13 Android Studio安装步骤13

（14）选择【android-sdk-license】，选中【Accept】，如图1.1.14所示：

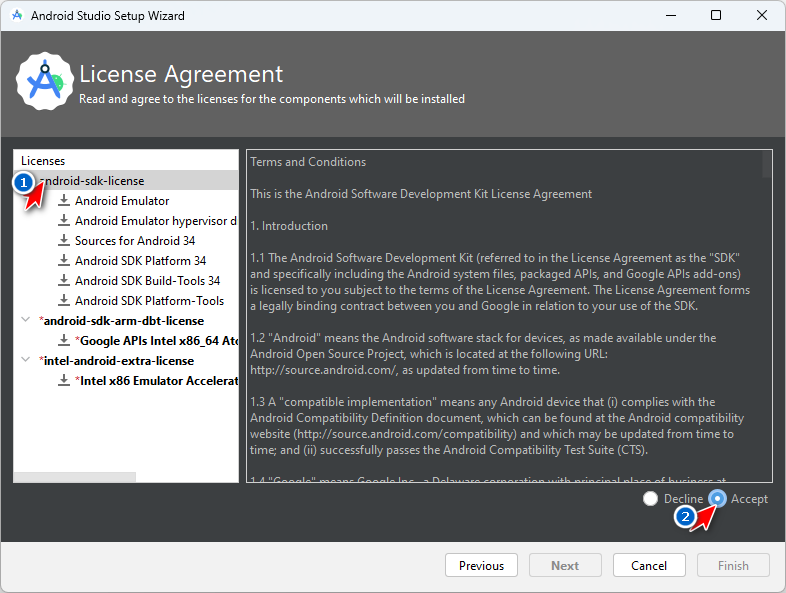


图1.1.14 Android Studio安装步骤14

（15）选择【android-sdk-arm-dbt-license】，选中【Accept】，如图1.1.15所示：

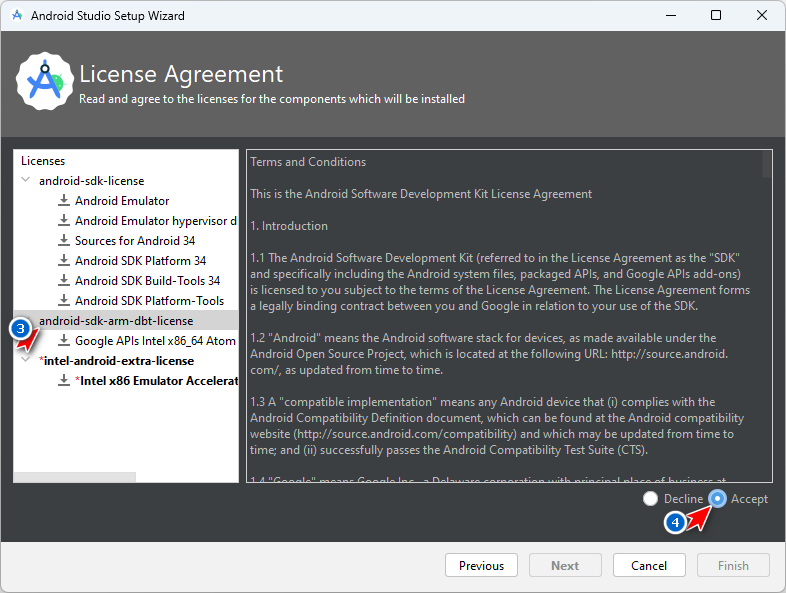


图1.1.15 Android Studio安装步骤15

（16）选择【intel-android-extra-license】，选中【Accept】，然后选择【Finish】，如图1.1.16所示：

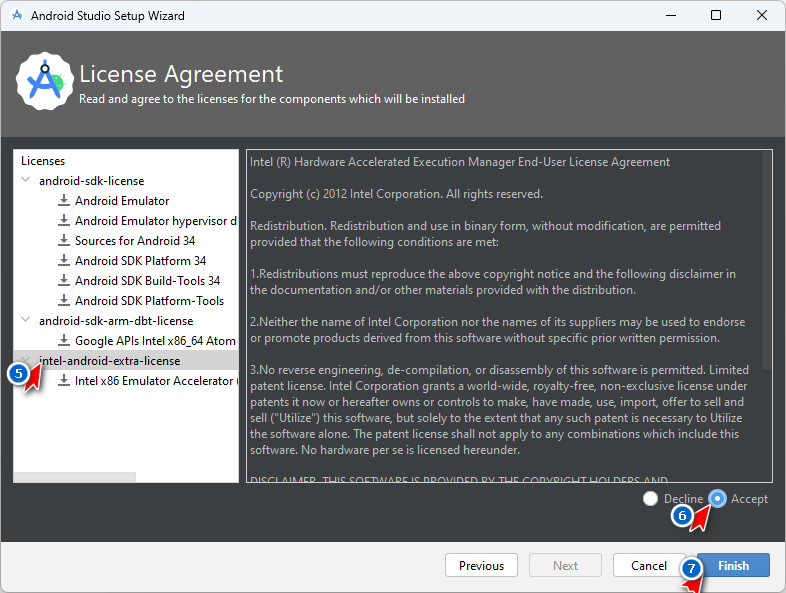


图1.1.16Android Studio安装步骤16

（17）等待配置完成，点击【Finish】，进入Android Studio欢迎页，如图1.1.17所示：

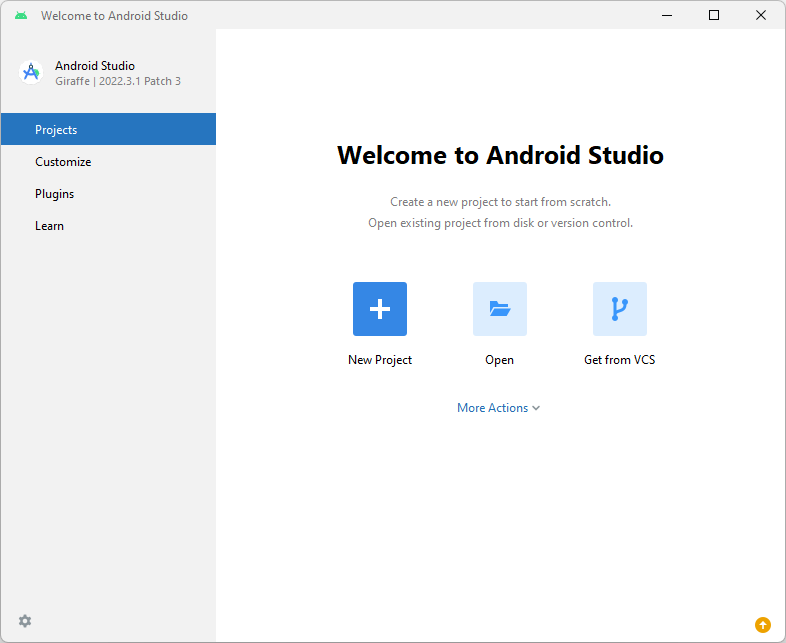
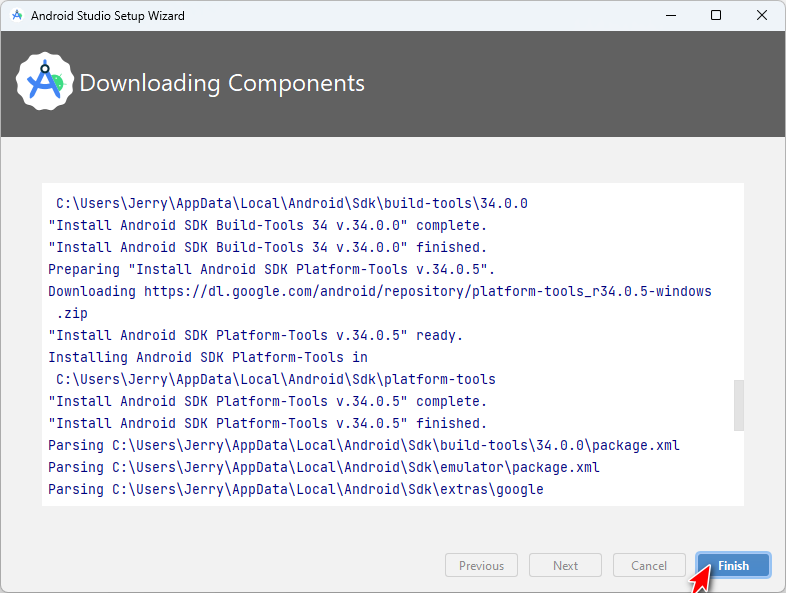


图1.1.17 Android Studio安装步骤17

（18）打开项目前需要配置sdk，点击【More Actions】，选择【SDK Manager】，如图1.1.18所示：

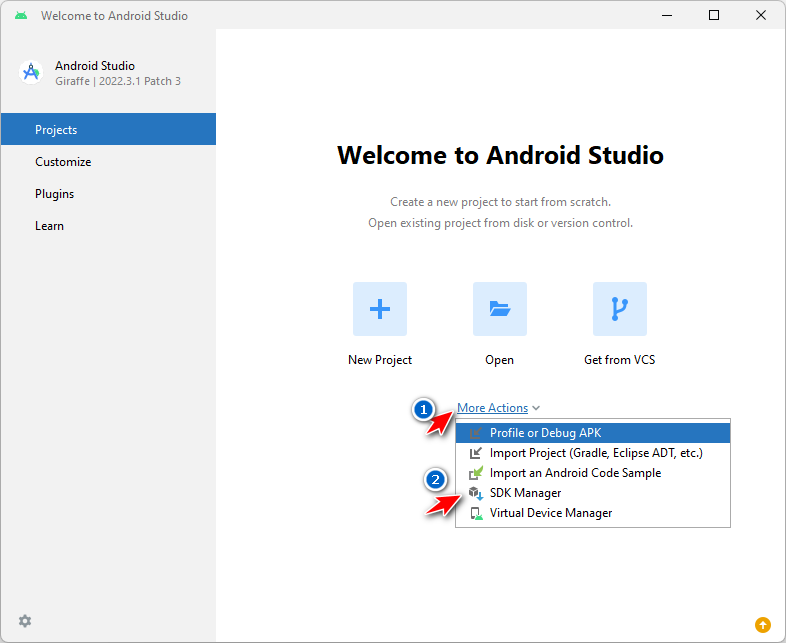


图1.1.18 Android Studio安装步骤18

（19）配置SDK保存路径以及下载SDK。实验需要API 29，选中Android 10.0（Q），然后点击OK，开始下载SDK，如图1.1.19所示。

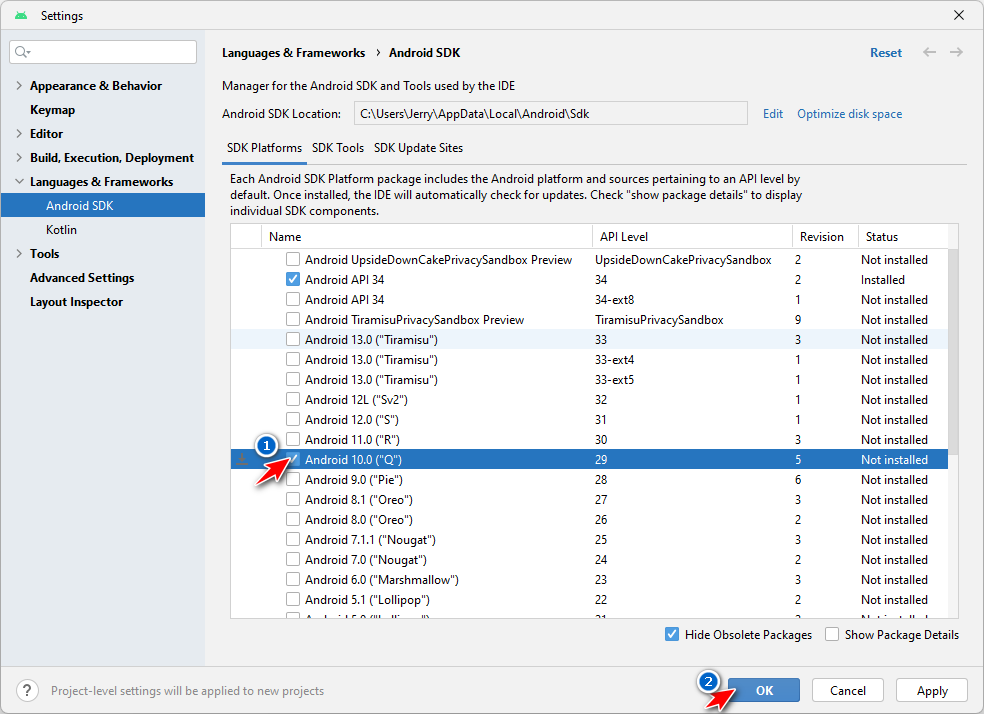


图1.1.19 Android Studio安装步骤19

（20）点击【OK】，如图1.1.20所示：

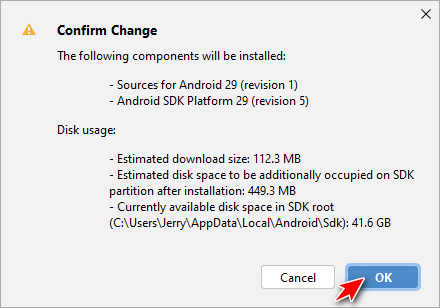


图1.1.20 Android Studio安装步骤20

（21）等待下载完成，点击【Finish】，回到Android Studio启动页，如图1.1.21所示：

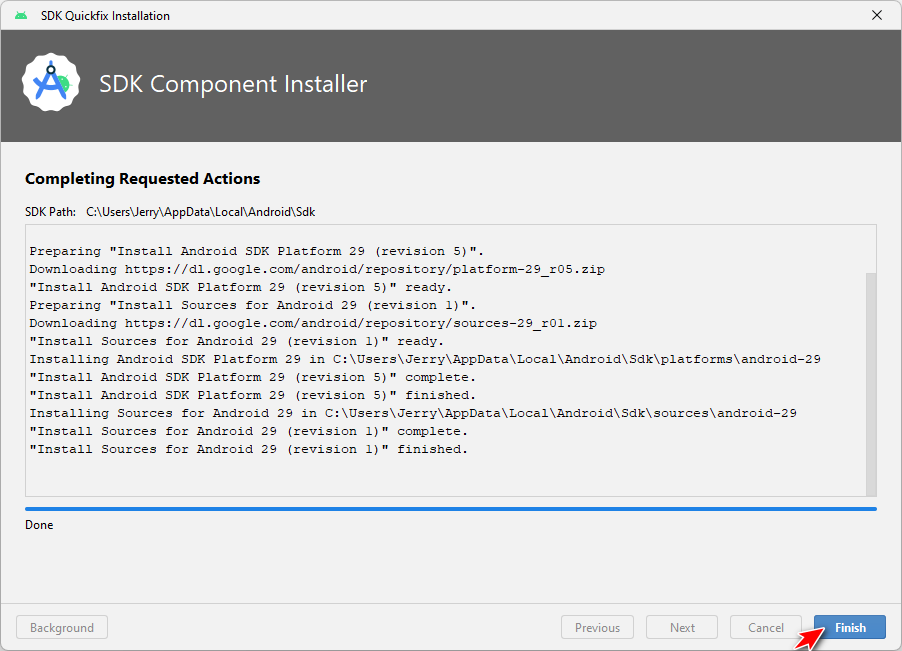


图1.2.21 Android Studio安装步骤21

Android Studio启动页点击【Open】，选择项目所在的文件夹，点击【OK】，即可打开Android项目，如图1.1.22所示：

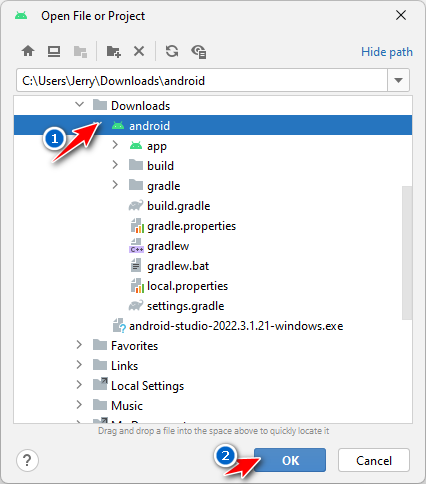
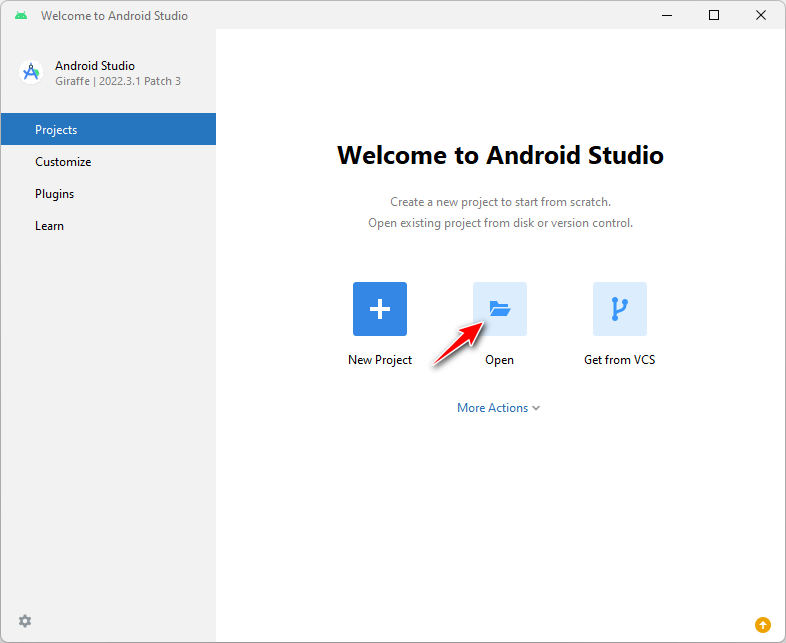


图1.1.22 Android Studio安装步骤22

（23）同步SDK设置，点击【OK】，如图1.1.23所示：

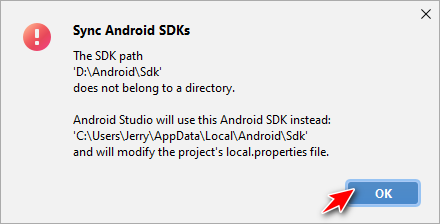


图1.1.23 Android Studio安装步骤23

（24）点击左上角【Android】，下拉选择【Project】，即可查看项目所有文件，如图1.1.24所示：

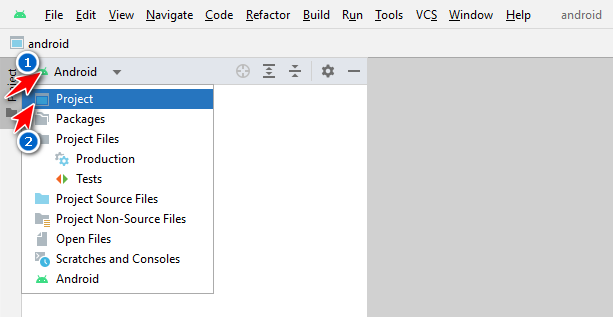
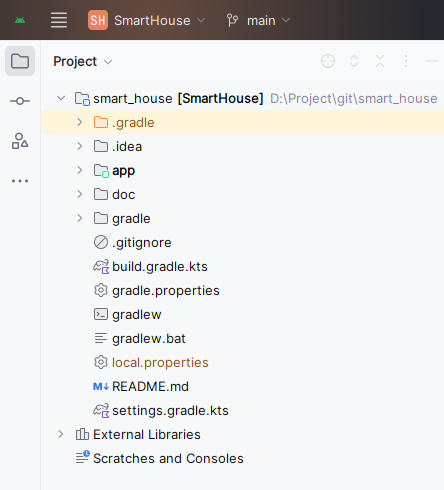
 

图1.1.24 Android Studio安装步骤24

## 1.2 设备连接

调试程序时，可以直接连接实体设备（Android手机），也可以在Android Studio中创建虚拟设备。

### 1.2.1 连接实体设备

连接实体手机前需要在手机中开启USB调试模式。“USB调试”开关位于开发者选项页面，但开发者选项的入口默认隐藏，不同的设备， 开发者选项界面所在的位置或所用的名称可能有所不同。但是方法都是一样的。

首先，进入“设置-关于手机”页面

其次，找到设备的版本信息、或是安卓版本号、或是软件版本号

然后，快速点击多次，就可以看到“已开启开发者模式”的提示

最后，返回“系统管理”，或是设置页面，就可以看到“开发者选项”，一般都在页面偏下的位置。

以下一部分品牌的开发者模式对应开启位置：

|  |  |
| --- | --- |
| 设备 | 设置 |
| Google Pixel | 设置 > 关于手机 > build号 |
| Samsung Galaxy S8及更高版本 | 设置 > 关于手机 > 软件信息 > 版本号 |
| LG G6及更高版本 | 设置 > 关于手机 > 软件信息 > 版本号 |
| HTC U11及更高版本 | 设置 > 关于 > 软件信息 > 更多 > 版本号 或 设置 > 系统 > 关于手机 > 软件信息 > 更多 > 版本号 |
| 一加5T及更高版本 | 设置 > 关于手机 > build号 |
| 小米 | 设置 > 我的设备> 全部参数 > MIUI版本 |
| vivo | 设置 >系统管理 > 关于手机 > 版本信息 > 软件版本号 |
| 华为 | 设置 > 关于手机 > 版本号 |

进入开发者选项，开启开发者选项开关，找到USB调试开关，并开启。将设备通过USB连接电脑后，可以在设备上看到“允许USB调试”的弹框。为了之后调试方便，这里勾选“一律允许使用这台计算机进行调试”，并点击确定。打开电脑命令行，输入指令“adb devices -l”，查看输出结果，就可以看到设备是否连接成功。

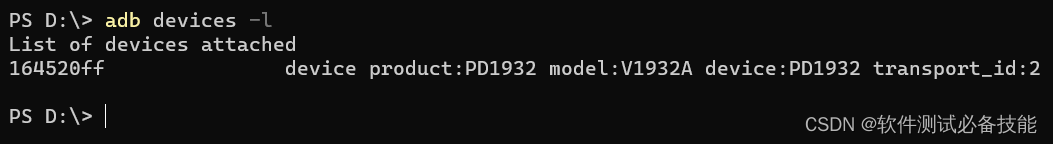
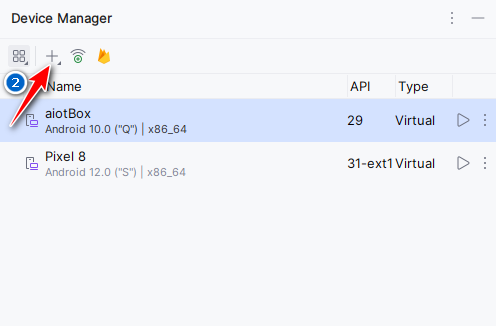
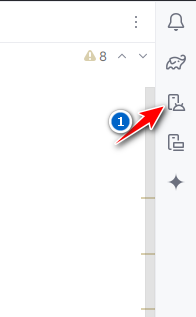


图1.2.1 连接设备

### 1.2.2 创建虚拟设备

如果暂时没有Android手机，可以在Android Studio中创建虚拟设备。如图1.2.2所示，点击对应图标打开设备管理器，然后依次点击添加图标，点击Create Virtual Device打开虚拟设备配置窗口，如图1.2.3所示:



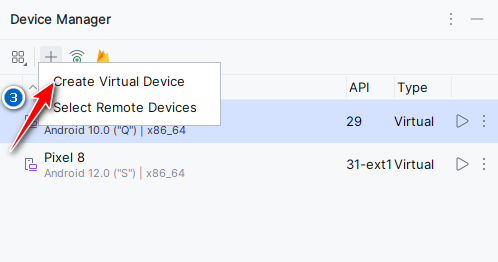


图1.2.2 创建虚拟设备

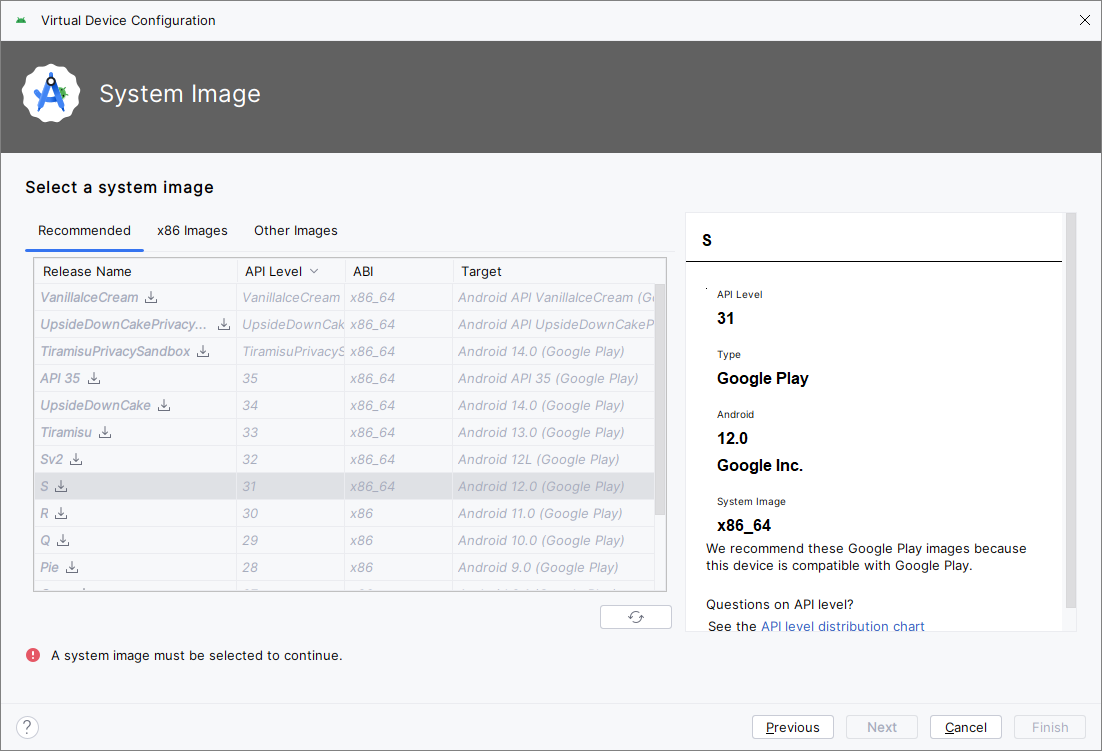
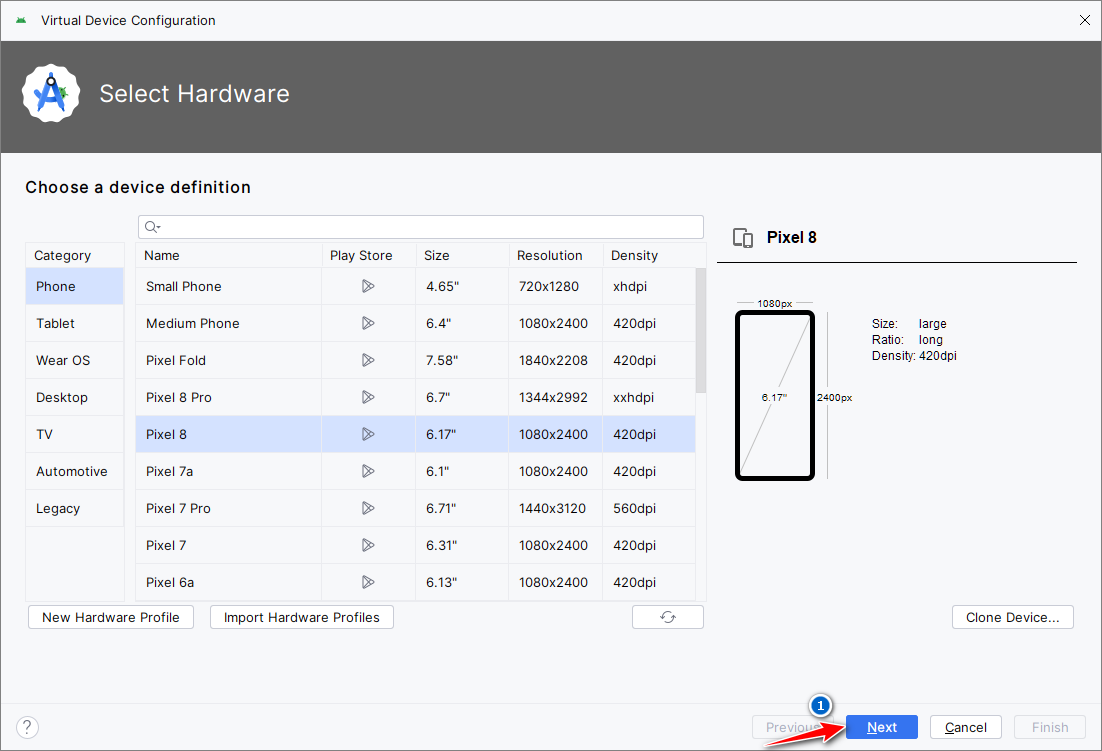


图1.2.3 虚拟设备配置

然后点击Next，进入系统镜像选择窗口，可以自行选择想要运行的系统镜像，由于程序中设置的SDK为Android Q，镜像需要选择Android Q及以上的版本，然后点击版本后面的下载图标，会进入SDK镜像下载窗口，如图1.2.4所示。

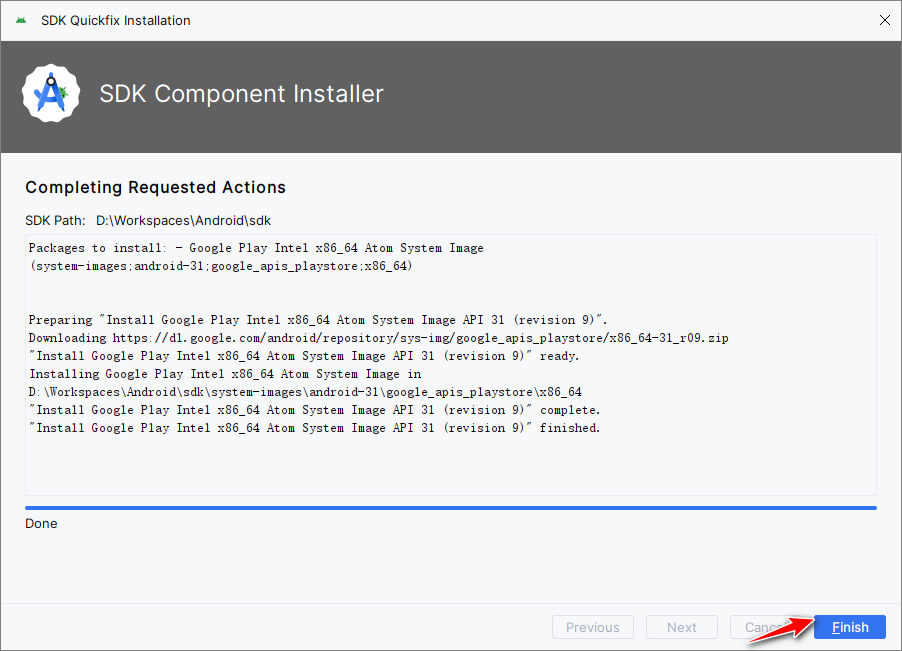
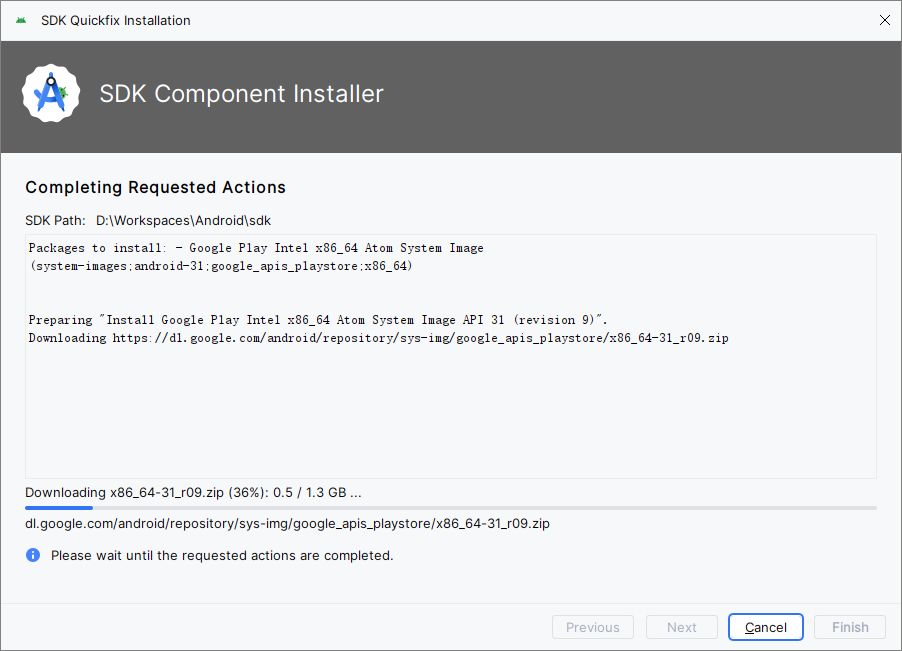


图1.2.4 SDK镜像下载窗口

下载完成后，点击Finish回到镜像选择窗口，选中下载的版本后点击Next进入虚拟设备配置窗口，如图1.2.5所示。

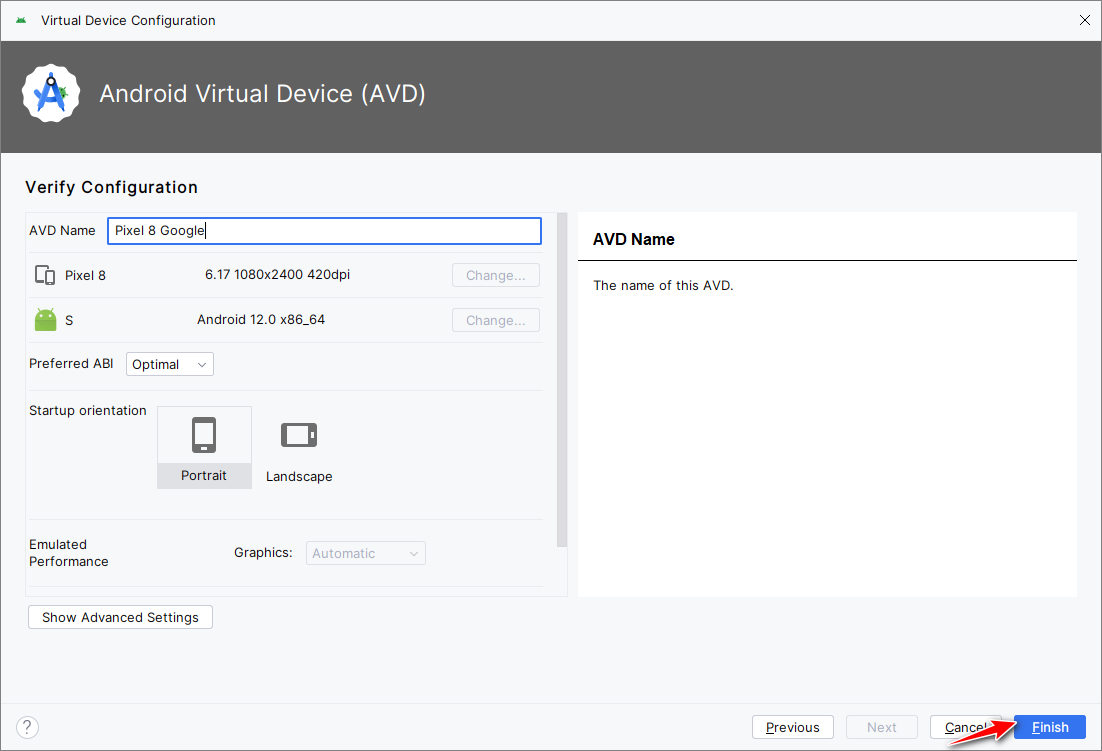
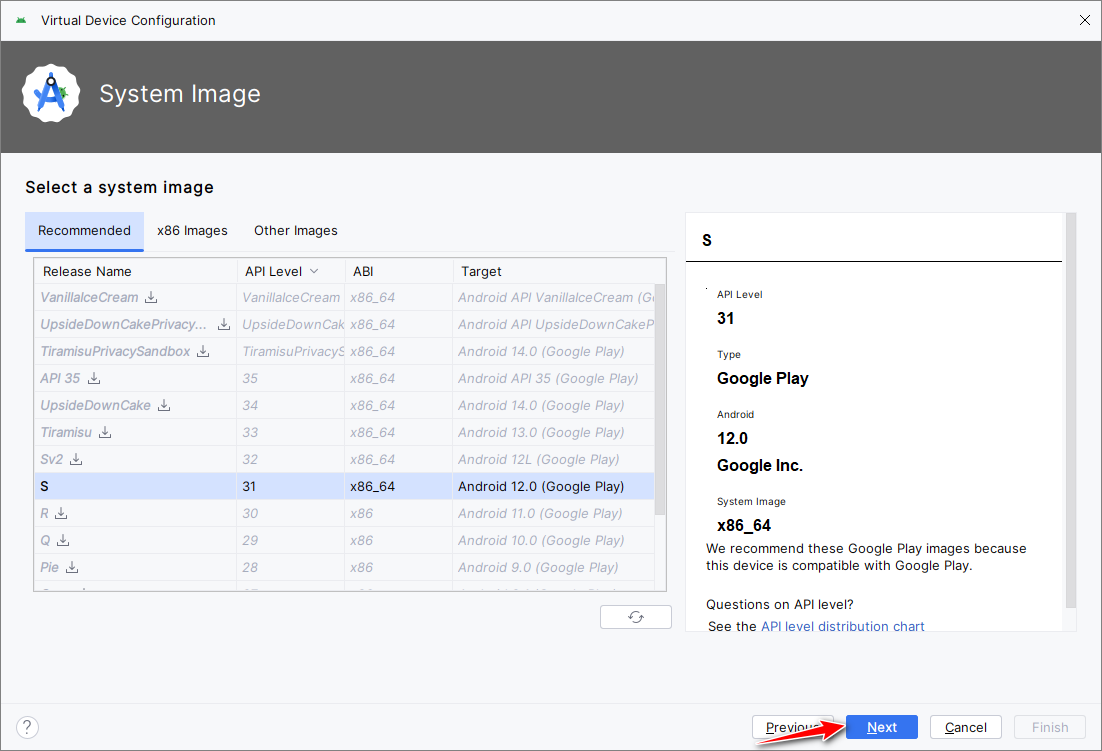


图1.2.5 虚拟机配置窗口

修改虚拟设备名称后点击Next，虚拟设备创建完成，然后在设备管理器中点击设备后面的运行图标即可开启虚拟设备，如图1.2.6所示，设备开启后，Android Studio顶部会显示已连接的设备。

|  |  |
| --- | --- |
|  |  |

图1.2.6 开启虚拟设备

设备连接后，点击运行图标，即可将应用安装到设备上运行。

# 第2章 远程控制app开发介绍

## 2.1 项目结构

远程控制APP程序项目结构如下图所示：

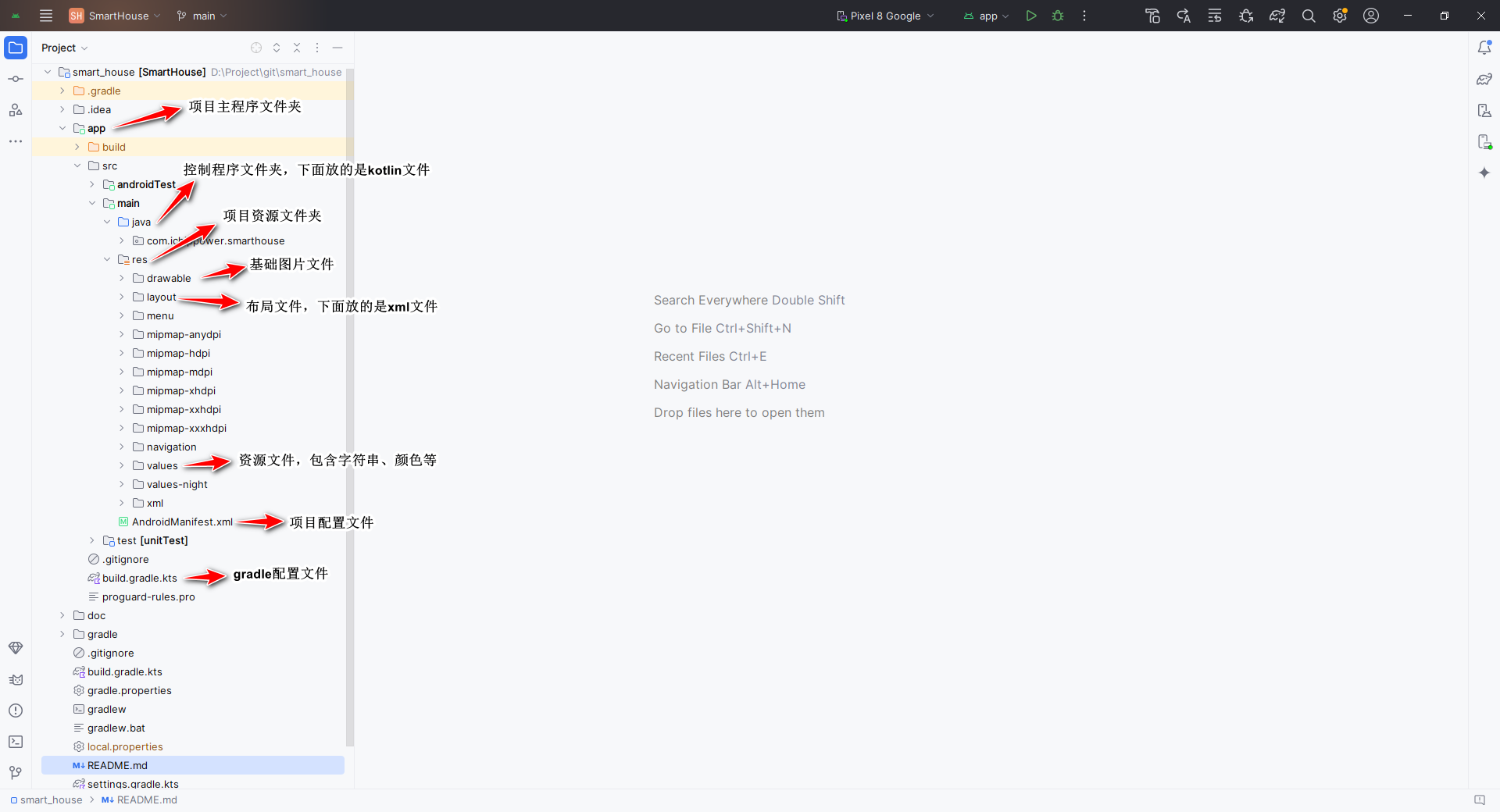


图2.1.1 项目结构

## 2.2 Android页面布局组件介绍

下面是fragment\_living.xml中的部分代码，以这部分代码简单介绍Android页面的布局和组件。

|  |
| --- |
| <LinearLayout  android:id="@+id/firstRow"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:baselineAligned="false"  app:layout\_constraintBottom\_toBottomOf="@id/contentRow">  <androidx.constraintlayout.widget.ConstraintLayout  android:layout\_width="0dp"  android:layout\_height="match\_parent"  android:layout\_weight="1"  android:background="@drawable/linear\_border"  android:visibility="visible"  tools:visibility="visible">  <ImageView  android:layout\_width="100dp"  android:layout\_height="150dp"  android:layout\_marginStart="5dp"  android:layout\_marginTop="20dp"  android:layout\_marginBottom="20dp"  app:layout\_constraintBottom\_toBottomOf="parent"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toTopOf="parent"  app:srcCompat="@drawable/door" />  <LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:layout\_marginStart="100dp"  android:gravity="center"  android:orientation="vertical"  app:layout\_constraintStart\_toStartOf="parent"  app:layout\_constraintTop\_toTopOf="parent">  <TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:fontFamily="sans-serif"  android:text="@string/door\_name"  android:textAlignment="center"  android:textColor="@color/black"  android:textSize="22sp"  android:textStyle="bold"  android:visibility="visible" />  <ToggleButton  android:id="@+id/switchDoor"  android:layout\_width="90dp"  android:layout\_height="48dp"  android:layout\_marginTop="25dp"  android:background="@drawable/togglebtn\_check"  android:checked="false"  android:disabledAlpha="1"  android:textOff=""  android:textOn="" />  </LinearLayout>  </androidx.constraintlayout.widget.ConstraintLayout>  </LinearLayout> |

客厅页面如图2.2.1所示。



图2.2.1 远程控制app客厅页面

### 2.2.1 LinearLayout

LinearLayout又称作线性布局，是一种非常常用的布局。正如它的名字所描述的一样，这个布局会将它所包含的控件在线性方向上依次排列。通过android:orientation属性指定了排列方向是vertical，控件就会在垂直方向上排列，如果指定的是horizontal，控件就会在水平方向上排列了。android:layout\_width和android:layout\_height设置布局大小，match\_parent指与父组件大小一致，wrap\_content是设置按照内容自动调节大小。

需要注意，如果LinearLayout的排列方向是horizontal，内部的控件就绝对不能将宽度指定为match\_parent，否则，单独一个控件就会将整个水平方向占满，其他的控件就没有可放置的位置了。同样的道理，如果LinearLayout的排列方向是vertical，内部的控件就不能将高度指定为match\_parent。

### 2.2.2 ImageView

ImageView是用于在界面上展示图片的一个控件，它可以让我们的程序界面变得更加丰富多彩。图片通常是放在以drawable目录下的，可以使用android:src属性给ImageView指定一张图片。

### 2.2.3 TextView

TextView用于在界面上展示文字。我们使用android:gravity来指定文字的对齐方式，可选值有top、bottom、start、end、center等，可以用“|”来同时指定多个值，这里我们指定的是"center"，效果等同于"center\_vertical|center\_horizontal"，表示文字在垂直和水平方向都居中对齐。通过android:textColor属性可以指定文字的颜色，通过android:textSize属性可以指定文字的大小。文字大小要使用sp作为单位。

### 2.2.4 ToggleButton

Button按钮是程序用于和用户进行交互的一个重要控件，ToggleButton是一个可以打开或关闭的按钮，可以通过点击按钮在两种状态之间切换。ToggleButton中可供我们设置的属性：

android:disabledAlpha：设置按钮在禁用时的透明度

android:textOff：按钮没有被选中时显示的文字

android:textOn：按钮被选中时显示的文字，另外，除了这个我们还可以自己写个selector，然后设置下background属性即可

### 2.2.5 RadioGroup

RadioGroup可以提供几个选项供用户选择，但只能选择其中的一个。其下面可以横着或者竖着挂几个RadioButton，也可以挂载其他控件（如TextView）。如果想改变RadioGroup里RadioButton的排列方式，可以使用属性android:orientation。RadioButton可以设置android:button="@null"然后设置android:background自定义RadioButton样式。注意要为每个RadioButton添加一个id，不然单选功能会失效。

## 2.3 Android控制程序

同样使用客厅页面控制程序的一部分为例介绍控制程序开发。

|  |
| --- |
| val noiseView: TextView = binding.noiseText  livingViewModel.noise\_text.observe(viewLifecycleOwner) {  noiseView.text = it  }  noiseView.setOnClickListener {  Thread {  run {  MqttService.publishData()  }  }.start()  }  val doorButton = binding.switchDoor  doorButton.setOnCheckedChangeListener { \_, isChecked ->  if (isChecked) {  Thread {  run {  val requestParam =  RequestParam(  HouseActivity.houseNumber,  "mode",  "客厅的门",  "on",  "",  ""  )  MqttService.publishMode(requestParam)  }  }.start()  } else {  Thread {  run {  val requestParam =  RequestParam(  HouseActivity.houseNumber,  "mode",  "客厅的门",  "off",  "",  ""  )  MqttService.publishMode(requestParam)  }  }.start()  }  } |

这里在数据TextView设置了点击事件SetOnClickListener，点击后获取小屋传感器数据。在控制门的ToggleButton上设置setOnCheckedChangeListener，isChecked参数为开关的状态True/False。当状态为True时发送开门的消息，状态为False时发送关门的消息。关于MQTT发送和接收消息的方式参考下一节。

|  |
| --- |
| class LivingViewModel : ViewModel() {  var noise\_text: LiveData<String> = MutableLiveData<String>().apply {  value = "30 DB"  }  fun update(noise: String) {  noise\_text = MutableLiveData<String>().apply {  value = "$noise DB"  }  }  } |

在Android应用中使用ViewModel可以有效地解决界面数据管理问题。ViewModel是一个负责准备和管理Activity或Fragment中UI所需数据的类，它存在的目的是使数据能够在配置变化时生存下来，例如屏幕旋转。这种机制简化了数据处理流程，并提高了应用的健壮性。在Activity和Fragment中通过ViewModelProvider来获取ViewModel的实例。这样做确保了配置更改时，您获得的ViewModel实例与之前的是相同的，从而数据得以保留。update()函数用来更新数据。LiveData是一个可观察的数据持有者类，它遵循观察者模式，允许数据变化可以通知到观察者组件。与ViewModel结合使用时，LiveData提供了一种响应式方式来更新UI，这样可以确保UI组件总是展示最新的数据。

## 2.4 MQTT协议介绍

### 2.4.1 MQTT协议

MQTT（Message Queuing Telemetry Transport）是一种轻量级、基于发布-订阅模式的消息传输协议，适用于资源受限的设备和低带宽、高延迟或不稳定的网络环境。它在物联网应用中广受欢迎，能够实现传感器、执行器和其它设备之间的高效通信。

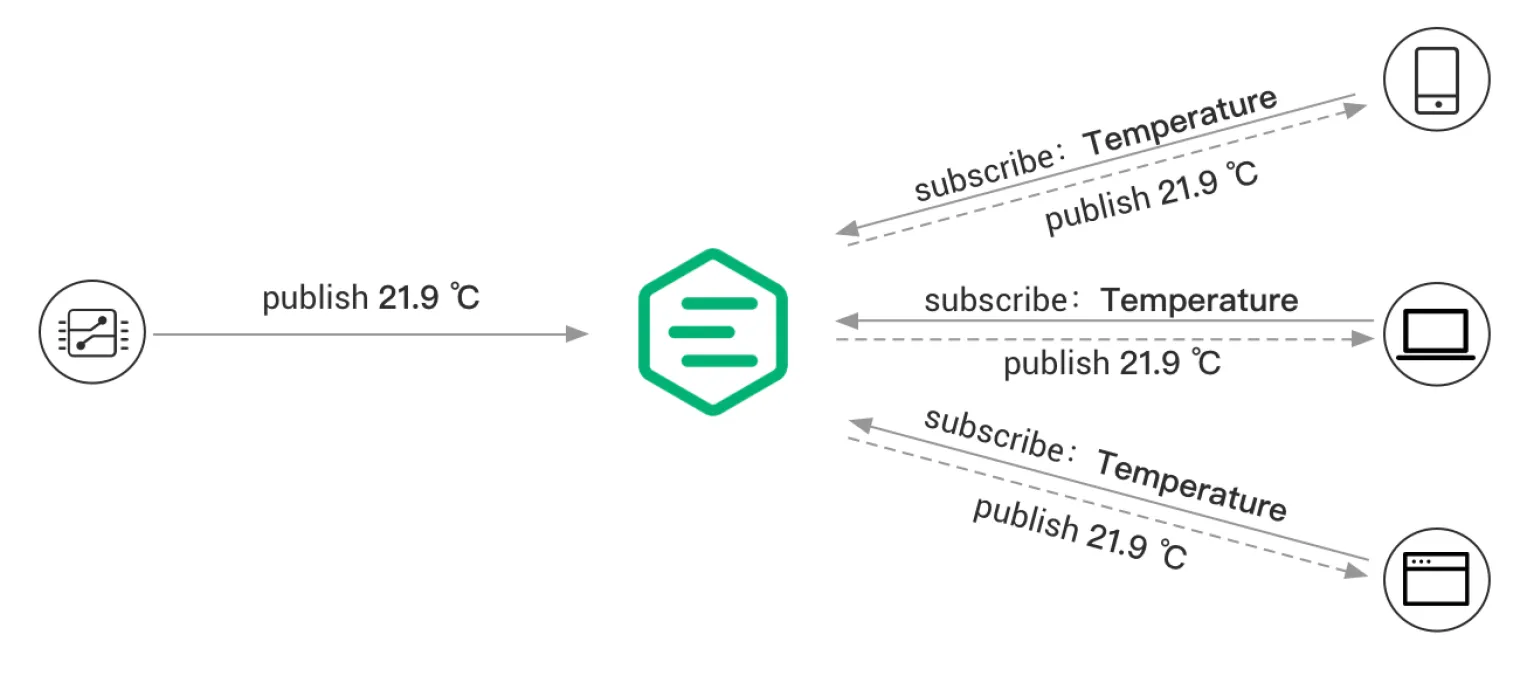


图2.4.1 MQTT通信

发布-订阅模式与客户端-服务器模式的不同之处在于，它将发送消息的客户端（发布者）和接收消息的客户端（订阅者）进行了解耦。发布者和订阅者之间无需建立直接连接，而是通过MQTT Broker来负责消息的路由和分发。图2.4.1展示了MQTT发布/订阅过程。温度传感器作为客户端连接到MQTT Broker，并通过发布操作将温度数据发布到一个特定主题（例如Temperature）。MQTT Broker接收到该消息后会负责将其转发给订阅了相应主题（Temperature）的订阅者客户端。

MQTT的工作流程如下：

* 客户端使用TCP/IP协议与Broker建立连接，可以选择使用TLS/SSL加密来实现安全通信。客户端提供认证信息，并指定会话类型（Clean Session或Persistent Session）。
* 客户端既可以向特定主题发布消息，也可以订阅主题以接收消息。当客户端发布消息时，它会将消息发送给MQTT Broker；而当客户端订阅消息时，它会接收与订阅主题相关的消息。
* MQTT Broker接收发布的消息，并将这些消息转发给订阅了对应主题的客户端。它根据QoS等级确保消息可靠传递，并根据会话类型为断开连接的客户端存储消息。

### 2.4.2 Android使用MQTT

（1）建立MQTT连接

|  |
| --- |
| mqttAndroidClient = MqttAndroidClient(  this, HOST, CLIENTID, Ack.AUTO\_ACK  ) |

context：当前Activity或者Fragment上下文

HOST：MQTT服务器地址

CLIENTID：客户端ID

|  |
| --- |
| mqttAndroidClient!!.setCallback(mqttCallback) //设置监听订阅消息的回调  mMqttConnectOptions = MqttConnectOptions()  mMqttConnectOptions!!.isCleanSession = true //设置是否清除缓存  mMqttConnectOptions!!.connectionTimeout = 10 //设置超时时间，单位：秒  mMqttConnectOptions!!.keepAliveInterval = 20 //设置心跳包发送间隔，单位：秒  mMqttConnectOptions!!.userName = USERNAME //设置用户名  mMqttConnectOptions!!.password = PASSWORD.toCharArray() //设置密码  /\*\*  \* 连接MQTT服务器  \*/  private fun doClientConnection() {  if (!mqttAndroidClient!!.isConnected && isConnectIsNormal) {  mqttAndroidClient!!.connect(  mMqttConnectOptions!!, null, iMqttActionListener  )  }  } |

（2）发送消息

|  |
| --- |
| /\*\*  \* 获取数据  \*/  fun publishData() {  //等待客户端连接  while (!mqttAndroidClient?.isConnected!!) {  continue  }  val qos = 2  val retained = false  val request = RequestParam(HouseActivity.houseNumber, "data", "", "", "", "")  val message = Gson().toJson(request)  Log.d(TAG, "publish message: $message")  //参数分别为：主题、消息的字节数组、服务质量、是否在服务器保留断开连接后的最后一条消息  **mqttAndroidClient!!.publish(publishTopic, message.toByteArray(), qos, retained)**  }  /\*\*  \* 修改模式  \*  \* @param requestParam 请求参数  \*/  fun publishMode(requestParam: RequestParam) {  //等待客户端连接  while (!mqttAndroidClient?.isConnected!!) {  continue  }  val qos = 2  val retained = false  val message = Gson().toJson(requestParam)  Log.d(TAG, "publish message: $message")  //参数分别为：主题、消息的字节数组、服务质量、是否在服务器保留断开连接后的最后一条消息  **mqttAndroidClient!!.publish(publishTopic, message.toByteArray(), qos, retained)**  } |

mqttAndroidClient!!.publish(publishTopic, message.toByteArray(), qos, retained)

publishTopic：发送主题

message：发送的消息

qos：QoS0，发送就不管了，最多一次；QoS1，发送之后依赖MQTT规范，是否启动重传消息，所以至少一次；QoS2，发送之后依赖MQTT消息机制，确保只有一次。

retained：是否保留消息。重新连接MQTT服务时，不需要接收该主题最新消息，设置retained为false；重新连接MQTT服务时，需要接收该主题最新消息，设置retained为true。

（3）订阅消息和接收消息

|  |
| --- |
| /\*\*  \* 订阅主题  \*/  fun subscribe() {  //等待客户端连接  while (!mqttAndroidClient?.isConnected!!) {  continue  }  **mqttAndroidClient!!.subscribe(subscribeTopic, 2) //订阅主题，参数：主题、服务质量（QoS）**  Log.d(TAG, "subscribe to $subscribeTopic")  }  /\*\*  \* 停止订阅主题  \*/  fun unsubscribe() {  //等待客户端连接  while (!mqttAndroidClient?.isConnected!!) {  continue  }  **mqttAndroidClient!!.unsubscribe(subscribeTopic)**  Log.d(TAG, "unsubscribe from $subscribeTopic")  }  //订阅主题的回调  private val mqttCallback: MqttCallback = object : MqttCallback {  override fun messageArrived(topic: String, message: MqttMessage) {  Log.i(TAG, "收到消息： " + String(message.payload))  //收到消息后，处理数据  response(String(message.payload))  }  override fun deliveryComplete(arg0: IMqttDeliveryToken) {}  override fun connectionLost(arg0: Throwable) {  Log.i(TAG, "连接断开 ")  doClientConnection() //连接断开，重连  }  } |

### 2.4.3 发送接收数据说明

（1）发送数据对象

|  |
| --- |
| /\*\*  \* houseNumber: 智慧小屋编号  \* commandType: 命令类型："data"：获取数据（device，switch，color和mode无效），"mode"：切换模式  \* device：设备。客厅：门，灯，灯带；卧室：灯；厨房：灯，抽油烟机，火焰报警，燃气报警；浴室：浴霸，热水器，排气扇，灯，窗帘，自动关闭窗帘，自动打开窗帘  \* （例：客厅的灯，如果设备的类型只有一个，省略房间）  \* switch：on/off，切换开关（color和mode无效）  \* color：yellow/blue/red/purple，客厅的灯切换颜色，其他设备无效，单独修改color，mode要设为""  \* mode：模式一/模式二/模式三，客厅的灯切换模式，其他设备无效，单独修改mode，color要设为""  \*/  class RequestParam(  private var houseNumber: String,  private var commandType: String,  private var device: String,  private var switch: String,  private var color: String,  private var mode: String  ) |

（2）接收数据对象

|  |
| --- |
| /\*\*  \* temperature：温度  \* humidity：湿度  \* illumination：光照强度  \* pressure：气压  \* noise：噪声强度  \* concentration：一氧化碳浓度  \* alarm：是否报警，在发送数据时设置为false  \* alarmType：报警类型，火焰报警/燃气报警，alarm为false时无效  \*/  @Serializable  data class ResponseParam(  private var temperature: String,  private var humidity: String,  private var illumination: String,  private var pressure: String,  private var noise: String,  private var concentration: String,  private var alarm: String,  private var alarmType: String  ) { …… } |

# 附 录

1. Kotlin语言开发指南

对于Kotlin开发可参考以下教程，中文版教程比较旧，有能力可参考英文版。

中文版教程：《[Kotlin实战](Kotlin实战.pdf)》

英文版教程：《[Kotlin in Action 2nd Edition](kotlin-action-2nd-edition.pdf)》

2. Android应用开发指南

对于Android开发可参考以下教程。

《[第一行代码 Android 第3版](第一行代码%20Android%20第3版.pdf)》