# Lanya

**MainActivity.java**

根据以上代码的结构和功能，我会将它们分为以下几部分：

1. 权限请求和检查
   * **requestPermissions()**方法
   * **checkLocationPermission()**方法
   * **onRequestPermissionsResult()**方法
2. 初始化和设置
   * **onCreate()**方法
   * **onStart()**方法
   * **onActivityResult()**方法
3. 蓝牙设备搜索和配对
   * **discoverDevices()**方法
4. 与列表交互
   * **onResume()**方法
   * **devicesListView.setOnItemClickListener()**方法

这个代码片段是一个名为**lanya**的Android项目的**MainActivity**类，它继承自**AppCompatActivity**。这个类主要用于显示已配对的蓝牙设备，并允许用户选择一个设备进行连接。以下是对代码主要部分的分析：

1. 导入所需的包和类。
2. 声明一些类成员变量，如**REQUEST\_CODE\_PERMISSIONS**、**REQUEST\_ENABLE\_BT**、**REQUEST\_LOCATION\_PERMISSION**，它们用于请求权限和启用蓝牙的结果代码。还有**BluetoothAdapter**、**ArrayAdapter**和**ArrayList**来存储蓝牙设备信息。
3. **onCreate**方法是活动创建时调用的方法。在此方法中，请求所需的权限，获取布局中的**ListView**并设置适配器。然后获取默认的蓝牙适配器，并检查设备是否支持蓝牙。最后，为**ListView**设置**OnItemClickListener**，当用户选择一个设备时，将启动**ConnectedActivity**并传递所选设备。
4. **onResume**方法在活动恢复时调用。在这里，如果蓝牙已启用并且位置权限已授权，则调用**discoverDevices**方法来获取已配对的设备列表。
5. **discoverDevices**方法用于搜索已配对的蓝牙设备。首先清空设备列表，然后检查蓝牙连接和扫描权限。如果已授权，获取已配对的设备并将其添加到**ArrayAdapter**和**ArrayList**中。
6. **requestPermissions**方法用于请求应用所需的权限。根据Android版本，请求蓝牙连接、扫描和位置权限。
7. **checkLocationPermission**方法用于检查位置权限是否已授权。
8. **onActivityResult**方法在启用蓝牙的请求结果返回时调用。如果用户允许启用蓝牙，则调用**discoverDevices**方法。否则，显示提示并结束活动。
9. **onRequestPermissionsResult**方法在请求权限的结果返回时调用。如果所有权限都被授权，则调用**discoverDevices**方法。否则，显示提示并结束活动。

**onResume()**方法是用来触发蓝牙设备搜索的入口，当需要请求权限时，会调用**requestPermissions()**方法。当请求权限结果返回时，会调用**onRequestPermissionsResult()**方法进行处理。当从系统蓝牙设置页面返回时，会调用**onActivityResult()**方法更新设备列表。**discoverDevices()**方法则是用于实际搜索和添加设备的核心逻辑。实现了**蓝牙设备发现和连接功能**

1. **onStart**方法在活动开始时调用。检查蓝牙是否启用，如果没有启用，则请求启用蓝牙。

当用户启动应用时，以下是这个程序的执行流程：

1. **onCreate**：应用启动时，**onCreate**方法会被调用。此方法主要完成以下任务：
   * 请求蓝牙和位置权限。
   * 初始化ListView并设置其适配器。
   * 获取默认的蓝牙适配器。如果设备不支持蓝牙，提示用户并结束活动。
   * 为ListView设置项目点击监听器。当用户点击列表项时，启动**ConnectedActivity**并传递选定的设备。
2. **onStart**：在活动进入“开始”状态时，**onStart**方法会被调用。此方法主要完成以下任务：
   * 检查蓝牙适配器是否为空并且未启用。如果满足条件，则根据权限情况请求启用蓝牙。
3. **onResume**：在活动进入“恢复”状态时，**onResume**方法会被调用。此方法主要完成以下任务：
   * 如果蓝牙适配器已启用，调用**discoverDevices**方法获取已配对设备列表。
4. **onActivityResult**：当启用蓝牙的请求结果返回时，**onActivityResult**方法会被调用。此方法主要完成以下任务：
   * 判断请求结果。如果蓝牙未启用，提示用户并结束活动。
5. **onRequestPermissionsResult**：当权限请求结果返回时，**onRequestPermissionsResult**方法会被调用。此方法主要完成以下任务：
   * 判断所有请求的权限是否已授权。如果是，则应用继续运行；否则，提示用户并结束活动。
6. **discoverDevices**：此方法用于发现并显示已配对的蓝牙设备。它会执行以下任务：
   * 清空设备适配器和已配对设备列表。
   * 检查是否具有蓝牙连接、扫描和位置权限。如果有权限，则获取已配对设备并将其添加到适配器和列表中；否则，提示用户需要权限。
7. **requestPermissions**：此方法用于请求蓝牙和位置权限。它会执行以下任务：
   * 检查当前是否具有所需权限。如果没有权限，则请求相应权限。
8. **hasBluetoothPermissions**：此方法用于检查是否具有蓝牙连接、扫描和位置权限。它会返回一个布尔值，表示是否具有所需权限。
9. 用户选择设备：当用户从设备列表中选择一个设备时，**onItemClick**方法会被调用。此方法主要完成以下任务：
   * 获取选定设备并启动**ConnectedActivity**，将选定的设备作为参数传递给新活动。

在上述程序执行流程中，我们可以进一步详细地解释应用的逻辑和功能：

1. **MainActivity**活动启动时，首先会调用**onCreate**方法。在这个方法中，应用首先请求蓝牙和位置权限，然后初始化列表视图（**ListView**），并为其设置适配器，用于显示已配对的蓝牙设备。
2. 接下来，应用获取系统默认的蓝牙适配器。如果设备不支持蓝牙，应用会向用户显示一个Toast提示消息，并结束活动。
3. **onCreate**方法还会为列表视图设置一个项目点击监听器。当用户点击某个蓝牙设备时，应用会启动**ConnectedActivity**活动并将选定的蓝牙设备传递给它。**ConnectedActivity**是用于处理蓝牙设备连接和数据传输的活动。
4. 在**onStart**方法中，应用会检查蓝牙适配器是否已启用。如果没有启用，应用会请求用户启用蓝牙。如果用户拒绝启用蓝牙，应用将显示一个Toast提示消息，并结束活动。
5. **onResume**方法在活动重新获取焦点时被调用。在这里，应用会调用**discoverDevices**方法，该方法用于发现已配对的蓝牙设备并将它们添加到列表视图中。这样，用户可以看到所有已配对的设备，并从中选择一个设备来建立连接。
6. 当应用收到权限请求结果时，**onRequestPermissionsResult**方法会被调用。如果用户授予了所有请求的权限，应用会继续运行并发现已配对的设备。如果有任何权限被拒绝，应用会显示一个Toast提示消息，并结束活动。
7. 当启用蓝牙的请求结果返回时，**onActivityResult**方法会被调用。如果用户同意启用蓝牙，应用会继续运行并发现已配对的设备。如果用户拒绝启用蓝牙，应用会显示一个Toast提示消息，并结束活动。
8. 用户从设备列表中选择一个设备时，**onItemClick**方法会被调用。这时，应用会启动**ConnectedActivity**并传递选定的设备。在**ConnectedActivity**中，应用将处理蓝牙连接和数据传输。

以上代码的程序流程：

1. 应用程序启动并加载主活动(MainActivity)。
2. 在**onCreate()**方法中设置布局并初始化ListView和适配器。
3. 检查设备是否支持蓝牙，如果不支持则提示用户并退出应用程序。
4. 在**onStart()**方法中检查蓝牙是否已启用。如果没有启用，检查是否授权了蓝牙和位置权限，如果已授权则请求用户启用蓝牙，否则请求权限。
5. 在**onResume()**方法中检查蓝牙和位置权限是否已授权，如果已授权，则开始搜索蓝牙设备并显示它们的名称和地址。
6. 如果用户点击设备列表中的项目，则会启动ConnectedActivity，并将选定的设备传递给它。
7. 在**onActivityResult()**方法中处理用户启用蓝牙的结果，如果启用成功，则开始搜索蓝牙设备。
8. 在**onRequestPermissionsResult()**方法中处理用户授权权限的结果，如果授权成功，则开始搜索蓝牙设备。
9. **discoverDevices()**方法用于搜索已配对的蓝牙设备并将它们添加到列表中。
10. 当用户授权蓝牙和位置权限时，**checkLocationPermission()**方法用于检查是否已授权位置权限。
11. 如果用户没有授权必需的权限，应用程序将提示用户并退出。

**ConnectedActivity.java**

这段代码是一个名为 **ConnectedActivity** 的 Android 活动。在这个活动中，应用程序连接到选定的蓝牙设备并允许用户发送文件。以下是代码的详细分析：

1. 在 **onCreate** 方法中，首先设置布局并初始化视图。然后，通过 Intent 从 **MainActivity** 获取选定的蓝牙设备。如果设备不支持蓝牙连接，显示警告。
2. 创建一个 RFCOMM（无线电频率通信）蓝牙套接字，用于连接到蓝牙设备。UUID 是蓝牙串行端口配置文件（SPP）的通用唯一标识符，用于设备之间的无线串行通信。
3. 尝试连接到蓝牙设备并获取输出流。如果连接成功，显示相应的消息；如果连接失败，显示错误信息。
4. 在 **sendFileButton** 上设置点击监听器。当用户点击按钮时，调用 **selectFile()** 方法，该方法使用 Intent 选择器允许用户从存储中选择一个文件。
5. 当用户选择一个文件并返回结果时，**onActivityResult** 方法被调用。这里，创建一个新线程并调用 **sendFile()** 方法发送文件。
6. 在 **sendFile** 方法中，通过蓝牙套接字的输出流发送文件。将文件内容读入缓冲区并通过输出流写入，直到文件结束。发送成功后，显示 Toast 提示；发送失败时，显示错误信息。
7. 在 **onDestroy** 方法中，关闭蓝牙套接字以释放资源。
8. **initView()** 方法用于初始化视图，例如 TextView。

总之，这个活动连接到选定的蓝牙设备并允许用户选择一个文件发送到设备。文件发送完成后，显示相应的成功或失败信息。

以下是 **ConnectedActivity** 的详细执行流程和数据流：

1. 当 **ConnectedActivity** 启动时，**onCreate()** 方法首先被调用。该方法设置活动的布局，初始化视图组件，并从 **MainActivity** 获取传递过来的选定蓝牙设备。
2. 接着，尝试使用设备的 UUID 创建一个蓝牙套接字并连接到蓝牙设备。UUID 是一个用于蓝牙串行端口配置文件（SPP）的通用唯一标识符，用于设备之间的无线串行通信。
3. 如果连接成功，应用程序会在 TextView 中显示连接成功的消息。如果连接失败，显示错误信息并退出。
4. 用户可以点击“发送文件”按钮。点击事件触发 **onClick()** 方法，该方法调用 **selectFile()** 方法。**selectFile()** 方法使用 Intent 选择器启动一个新活动，允许用户从设备存储中选择文件。
5. 用户选择文件后，系统会调用 **onActivityResult()** 方法。在此方法中，将选定文件的 Uri 传递给 **sendFile()** 方法，并在新线程中执行该方法。
6. **sendFile()** 方法负责通过蓝牙套接字的输出流发送文件。首先，将文件输入流与 Uri 关联，然后创建一个缓冲区用于读取文件内容。接下来，循环读取文件内容并将其写入蓝牙套接字的输出流，直到文件结束。在此过程中，数据流通过蓝牙连接从 Android 设备传输到另一个蓝牙设备。
7. 文件发送完成后，**sendFile()** 方法会更新 UI，显示发送成功或失败的消息。此操作在主线程上运行，因为 UI 更新不能在工作线程上完成。
8. 当用户结束 **ConnectedActivity** 时，**onDestroy()** 方法会被调用。在此方法中，应用程序关闭蓝牙套接字以释放资源。

综上所述，**ConnectedActivity** 的执行流程涉及到从 Android 设备选择文件，通过蓝牙连接将其发送到另一个蓝牙设备，并在发送过程中更新 UI 以显示成功或失败信息。数据流主要通过蓝牙套接字的输出流在两个蓝牙设备之间进行传输。

**文件发送的具体实现过程**

**<一>**在**ConnectedActivity**类中，文件发送主要通过**sendFile()**方法实现。以下是详细的步骤和分析：

1. **sendFile()**方法接收一个**fileUri**参数，该参数表示用户选择的文件在设备上的位置。通过**getContentResolver().openInputStream(fileUri)**方法，可以为该文件创建一个**InputStream**。
2. 蓝牙套接字连接成功后，可以通过**bluetoothSocket.getOutputStream()**获取蓝牙套接字的输出流。这个输出流用于将数据发送到另一个蓝牙设备。
3. 在**sendFile()**方法中，使用一个循环来读取文件的内容。每次循环，都从文件输入流读取最多1024字节的数据到缓冲区。然后，将缓冲区中的数据写入蓝牙套接字的输出流。这个过程会一直持续到文件结束。
4. 在文件发送完成后，关闭文件输入流，并在UI上显示发送成功或失败的消息。

**<二>ConnectedActivity** 实现了文件传输的主要流程。以下是主要步骤：

1. 在 **onCreate()** 方法中，首先初始化视图，并获取 **BluetoothDevice** 对象。
2. 检查 Bluetooth 连接权限，如果没有权限，则记录日志并返回。
3. 尝试使用 **device.createInsecureRfcommSocketToServiceRecord(uuid)** 方法建立到远程设备的连接，并将结果赋值给 **bluetoothSocket**。
4. 设置 **sendFileButton** 的点击事件监听器，当用户点击按钮时，调用 **selectFile()** 方法打开文件选择器。
5. 在 **selectFile()** 方法中，创建一个 Intent 以获取文件内容，并启动 Activity。
6. 当用户选择了一个文件，**onActivityResult()** 方法会被调用。在这个方法中，通过传入的 Intent 获取文件的 URI。
7. 在新线程中调用 **sendFile(Uri fileUri)** 方法，将文件发送到已连接的设备。
8. 在 **sendFile(Uri fileUri)** 方法中：
   * 获取 **bluetoothSocket** 的输出流。
   * 打开文件输入流。
   * 使用缓冲区读取文件，并将读取到的数据写入到输出流。
   * 关闭文件输入流。
   * 更新 UI，显示发送成功的提示。
9. 当 Activity 销毁时，在 **onDestroy()** 方法中，如果 **bluetoothSocket** 不为 null，则关闭蓝牙连接。

**什么是Intent对象？**

在Android应用程序中，Intent（意图）对象是一种用于在应用程序组件之间传递数据的机制。Intent可以用于启动Activity、Service和BroadcastReceiver组件，以及用于在应用程序内部和应用程序之间传递数据。

Intent包含了要执行的操作的描述（如启动Activity或Service、发送Broadcast等）以及要传递的数据（如字符串、整数、数组等）。它还可以包含额外的信息，如组件类名、URI等。

在Android中，可以使用显式Intent和隐式Intent。显式Intent用于启动应用程序中已知的组件，而隐式Intent用于启动应用程序中没有明确指定的组件，但满足Intent中描述的操作和数据类型的组件。

例如，在以下代码中，创建了一个Intent对象，它将数据和一个参数传递给另一个Activity：

Intent intent = new Intent(MainActivity.this, SecondActivity.class);

intent.putExtra("key", value);

startActivity(intent);

在这个例子中，我们创建了一个Intent对象，并将MainActivity作为上下文参数，以启动名为SecondActivity的Activity组件。我们还通过调用**putExtra()**方法将名为“key”的字符串和value的值作为额外的数据附加到Intent对象中。最后，我们使用**startActivity()**方法启动SecondActivity，并将Intent对象传递给它。这样，在SecondActivity中就可以通过调用**getIntent().getStringExtra("key")**方法获取数据。

**activity\_main.xml**

这是 MainActivity 的布局文件，它包含一个垂直方向的 LinearLayout，其内部包含一个 ListView。ListView 用于展示已配对的蓝牙设备列表。设备名称和地址将显示在列表中，用户可以通过点击列表中的设备来与其建立连接。

**activity\_connected.xml**

这是 ConnectedActivity 的布局文件。这个布局文件包含一个垂直方向的 LinearLayout，包括一个 Button 和一个 TextView。Button 用于触发选择文件的操作，当用户点击该按钮时，将打开系统的文件选择器。TextView 用于显示连接状态、文件发送状态等日志信息。

**AndroidManifest.xml**

这是应用程序的清单文件，其中声明了应用程序需要的权限、Activity 等信息。应用程序需要以下权限：

* BLUETOOTH：允许应用程序使用蓝牙功能。
* BLUETOOTH\_ADMIN：允许应用程序发现和配对蓝牙设备。
* BLUETOOTH\_CONNECT：允许应用程序建立蓝牙连接。
* BLUETOOTH\_SCAN：允许应用程序扫描附近的蓝牙设备。
* ACCESS\_FINE\_LOCATION 和 ACCESS\_COARSE\_LOCATION：这些权限用于蓝牙设备扫描，从 Android 6.0（API 级别 23）开始，蓝牙设备扫描需要定位权限。
* READ\_EXTERNAL\_STORAGE 和 WRITE\_EXTERNAL\_STORAGE：这些权限允许应用程序访问设备的外部存储，以便从外部存储中读取要发送的文件。

此外，清单文件还声明了 MainActivity 和 ConnectedActivity，它们分别用于展示设备列表和与设备进行通信。

重要细节：

1. 蓝牙连接使用了通用的 UUID（00001101-0000-1000-8000-00805F9B34FB），这个 UUID 适用于大多数设备上的串行端口通信。
2. 在 Android 12（API 级别 31）及以上版本，需要检查 BLUETOOTH\_CONNECT 权限是否已被授予，否则无法建立连接。
3. 文件发送过程中，每次将文件内容读取到缓冲区中，并通过 OutputStream 将其发送给蓝牙设备，直到文件读取完毕。发送的文件类型并没有限制。

# Lanya2

**MainActivity.java**

这是lanya2项目的MainActivity，它是一个Android应用程序的入口点。这个类继承自AppCompatActivity，因此它可以作为一个兼容Android各个版本的Activity。

在**onCreate**方法中：

1. 调用**super.onCreate(savedInstanceState)**以确保父类的**onCreate**方法被执行。
2. 使用**setContentView(R.layout.activity\_main)**设置MainActivity对应的布局文件，这里是**activity\_main.xml**。

此外，代码创建了一个按钮**receiveFileButton**，并为其设置了一个点击事件监听器。当用户点击这个按钮时，将触发**onClick**方法，该方法执行以下操作：

1. 创建一个Intent，用于启动名为ReceiveFileActivity的Activity。
2. 使用**startActivity(intent)**方法启动ReceiveFileActivity。

这个MainActivity的主要作用是作为应用程序的入口点，并提供一个按钮，用户可以点击该按钮跳转到ReceiveFileActivity以接收文件。代码简洁且功能明确，设计合理。

**ReceiveFileActivity.java**

这是lanya2项目的ReceiveFileActivity类，这个Activity负责接收通过蓝牙发送过来的文件。这个类继承自AppCompatActivity，使其具有兼容性。

在**onCreate**方法中：

1. 调用**super.onCreate(savedInstanceState)**以确保父类的**onCreate**方法被执行。
2. 使用**setContentView(R.layout.activity\_receive\_file)**设置ReceiveFileActivity对应的布局文件，这里是**activity\_receive\_file.xml**。
3. 通过**findViewById**获取布局文件中的**received\_file\_content** TextView组件。
4. 获取蓝牙适配器（BluetoothAdapter）实例，如果设备不支持蓝牙，记录错误日志并结束Activity。
5. 调用**createServerSocket()**方法创建一个蓝牙服务器套接字（BluetoothServerSocket）并等待接收文件。

**createServerSocket()**方法：

1. 首先检查Android版本是否大于等于Build.VERSION\_CODES.S（Android 12），如果是，检查蓝牙连接权限。
2. 如果已经获得权限，或者版本低于Android 12，创建一个蓝牙服务器套接字并监听来自其他设备的连接请求。
3. 开启一个新线程等待接收连接，当有设备连接时，通过**readDataFromSocket()**方法读取数据。

**readDataFromSocket(BluetoothSocket socket)**方法：

1. 获取BluetoothSocket的输入流。
2. 创建一个byte数组作为缓冲区，用于存放读取到的数据。
3. 循环读取输入流中的数据，直到读取完毕。
4. 将读取到的数据转换为字符串，并追加到**receivedFileContentTextView**组件中。

在**onDestroy**方法中，如果serverSocket不为空，关闭serverSocket。

这个ReceiveFileActivity的主要作用是创建一个蓝牙服务器套接字，等待其他设备通过蓝牙发送文件，然后接收文件并显示文件内容。代码逻辑清晰，设计合理。但需要注意的是，由于它将接收到的数据直接转换为字符串并显示在TextView中，所以对于非文本文件（如图片、PDF等）显示会出现乱码。如果需要支持这些文件类型，需要对接收到的数据进行相应的处理。

首先，当应用启动时，MainActivity.java中的**onCreate**方法会被执行，这时它会加载activity\_main.xml布局并显示在屏幕上。在这个布局中，有一个按钮（receiveFileButton），当用户点击这个按钮时，会触发一个点击事件监听器，这个监听器会创建一个Intent，用于启动ReceiveFileActivity，并通过**startActivity(intent)**方法启动它。

当ReceiveFileActivity启动后，它的**onCreate**方法会被执行。在此方法中，它会加载activity\_receive\_file.xml布局并显示在屏幕上。同时，它会获取到布局中的receivedFileContentTextView组件，用于显示接收到的文件内容。然后，它会获取蓝牙适配器实例，并调用**createServerSocket()**方法创建一个蓝牙服务器套接字（BluetoothServerSocket）并等待接收文件。

**createServerSocket()**方法创建一个蓝牙服务器套接字并监听来自其他设备的连接请求。当有设备连接上后，它会开启一个新线程，通过**readDataFromSocket()**方法从BluetoothSocket的输入流中读取数据。

**readDataFromSocket()**方法将循环读取输入流中的数据，直到读取完毕。将读取到的数据转换为字符串，并追加到**receivedFileContentTextView**组件中，从而实现接收文件并在界面上显示文件内容的功能。

总结来说，MainActivity和ReceiveFileActivity的关系是：MainActivity作为应用的入口点，当用户点击MainActivity界面上的“receiveFileButton”按钮时，它会启动ReceiveFileActivity。ReceiveFileActivity负责接收通过蓝牙发送过来的文件，并将文件内容显示在界面上。数据流向是从发送方设备通过蓝牙连接传输给ReceiveFileActivity，然后将接收到的数据显示在receivedFileContentTextView组件中。

**文件发送的具体实现过程**

1. 首先，在**onCreate()**方法中，获取蓝牙适配器（BluetoothAdapter）。此适配器用于与其他蓝牙设备通信。然后调用**createServerSocket()**方法以监听来自其他设备的连接请求。
2. 在**createServerSocket()**方法中，根据应用名称（APP\_NAME）和UUID创建一个蓝牙服务器套接字（BluetoothServerSocket）。此套接字用于监听其他设备的连接请求，并在接收到请求时建立安全的蓝牙通信连接。
3. 接下来，启动一个新线程等待其他设备发起的连接请求。**serverSocket.accept()**方法将阻塞，直到另一个设备发起连接请求。一旦接受到连接请求，**accept()**方法将返回一个与远程设备关联的蓝牙套接字（BluetoothSocket）。
4. 随后，**readDataFromSocket()**方法将使用该蓝牙套接字从远程设备接收文件。此方法首先获取套接字的输入流。输入流用于从远程设备接收数据。
5. 然后创建一个1024字节的缓冲区（buffer）和一个字符串构建器（StringBuilder）。缓冲区用于暂存从输入流中读取的数据，字符串构建器用于存储接收到的文件内容。
6. 使用**inputStream.read(buffer)**方法从输入流中读取数据。这个方法会阻塞，直到有数据可读。每次读取的数据将存储在缓冲区中，方法返回实际读取的字节数。当没有更多数据可读时，方法返回-1。
7. 对于每次读取的数据，将其转换为字符串，然后追加到字符串构建器（**stringBuilder**）中。在此过程中，需要注意字符编码。在本示例中，使用UTF-8编码。
8. 为了实时更新接收到的文件内容视图（**receivedFileContentTextView**），使用**runOnUiThread()**方法在UI线程中设置视图的文本。将字符串构建器中的文本设置为视图的文本，以便用户可以查看接收到的文件内容。

结合代码分析：

1. 在**onCreate()**方法中，获取蓝牙适配器（BluetoothAdapter），并调用**createServerSocket()**方法。

bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

if (bluetoothAdapter == null) {

Log.e(TAG, "Bluetooth not supported");

finish();

}

createServerSocket();

1. 在**createServerSocket()**方法中，创建一个蓝牙服务器套接字（BluetoothServerSocket），用于监听来自其他设备的连接请求。

serverSocket=bluetoothAdapter.listenUsingInsecureRfcommWithServiceRecord(APP\_NAME, MY\_UUID);

1. 在**createServerSocket()**方法中，启动一个新线程，等待其他设备发起的连接请求。当接受到连接请求时，获取与远程设备关联的蓝牙套接字（BluetoothSocket）。

new Thread(() -> {

try {

BluetoothSocket socket = serverSocket.accept();

Log.i(TAG, "Connected to device: " + socket.getRemoteDevice().getName());

readDataFromSocket(socket);

} catch (IOException e) {

Log.e(TAG, "Error accepting connection", e);

runOnUiThread(new Runnable() {

@Override

public void run() {

receivedFileContentTextView.append("连接失败："+e.getMessage());

}

});

}

}).start();

1. 在**readDataFromSocket(BluetoothSocket socket)**方法中，使用蓝牙套接字获取输入流，并读取发送端发送的文件内容。

InputStream inputStream = socket.getInputStream();

byte[] buffer = new byte[1024];

int bytesRead;

StringBuilder stringBuilder = new StringBuilder();

1. 持续读取输入流中的数据，直到读取完毕。将读取到的数据追加到**stringBuilder**中，并在UI线程中更新接收到的文件内容视图（**receivedFileContentTextView**）。

while ((bytesRead = inputStream.read(buffer)) != -1) {

String text = new String(buffer, 0, bytesRead, StandardCharsets.UTF\_8);

stringBuilder.append(text);

runOnUiThread(new Runnable() {

@Override

public void run() {

receivedFileContentTextView.setText(stringBuilder.toString());

}

});

}

**发送图片，word之类的就会乱码，只能发送txt文件才不会乱码的原因:**

ReceiveFileActivity中的代码实际上可以接收任何类型的文件，包括图片、视频、PDF、Word等。问题在于，接收到的数据是如何被处理和显示的。

在ReceiveFileActivity中，接收到的文件数据被读取为字节数组，然后使用UTF-8编码将其转换为字符串（**String text = new String(buffer, 0, bytesRead, StandardCharsets.UTF\_8);**）。这种处理方法适用于文本文件（如TXT文件），因为它们是以字符形式存储的，可以直接转换为字符串并显示在TextView组件上。

然而，对于非文本文件（如图片、视频、PDF、Word等），这种处理方法是不合适的。这些文件包含二进制数据，不能直接转换为字符串。当试图将这些文件的二进制数据转换为字符串时，可能会出现乱码，因为UTF-8编码并不适用于非文本数据。因此，在接收并显示这些文件时，会出现乱码的问题。

要解决这个问题，需要根据接收到的文件类型进行相应的处理。例如，对于图片文件，您可以将字节数组转换为Bitmap对象，并使用ImageView组件来显示；对于PDF和Word文件，您可以使用相关的库或Android原生API来处理和显示这些文件。

**如何可以接收图片呢？做了一下修改**

1. 在**ReceiveFileActivity**类中添加一个新的**ImageView**成员变量：

private ImageView receivedFileImageView;

1. 在**onCreate()**方法中初始化**receivedFileImageView**：

receivedFileImageView=findViewById(R.id.received\_file\_image);

1. 修改**readDataFromSocket()**方法，以根据接收到的数据类型显示文本或图像：根据接收到的文件类型标识，修改该方法以处理不同类型的文件。这样做的目的是让接收端能够针对不同类型的文件执行相应的处理逻辑。
2. 添加一个辅助方法getFileType（）来确定接收到的数据的文件类型：在处理文本文件时，将接收到的数据追加到TextView中以显示文件内容。这样做的目的是为了在接收端正确显示文本文件内容。
3. 在布局XML文件中，同时添加TextView和ImageView，并设置它们的ID分别为received\_file\_content和received\_file\_image。默认情况下，将图像视图的可见性设置为gone。

**activity\_main.xml**

此文件是MainActivity的布局文件，它包含一个按钮，用户可以点击该按钮以接收文件。

**activity\_receive\_file.xml**

此文件是ReceiveFileActivity的布局文件，它包含一个TextView，用于显示接收到的文本文件内容，以及一个ImageView，用于显示接收到的图像文件。

**AndroidManifest.xml**

此文件描述了应用程序的组件及其相关元数据，例如权限、Activity声明等。

**只能发一次，发第二次就显示接收失败的原因：**

问题的原因在于，在**ReceiveFileActivity**的**readDataFromSocket**方法中，没有一个终止条件来结束**while**循环。这会导致在接收到第一个文件后，循环无法结束，使得后续的文件传输无法进行。

为了解决这个问题，可以在发送端的**ConnectedActivity**向接收端发送一个特殊的终止字符或者字符串，以表示文件传输的结束。然后在接收端的**ReceiveFileActivity**中检测这个特殊的终止字符或者字符串，从而结束循环。

首先，在**ConnectedActivity**的**sendFile**方法中，在文件发送完毕后，发送一个特殊的终止字符串，如"EOF"：

outputStream.write("EOF".getBytes(StandardCharsets.UTF\_8));

outputStream.flush();

然后，在**ReceiveFileActivity**的**readDataFromSocket**方法中，检测这个特殊的终止字符串，并在检测到后跳出循环：

if (text.contains("EOF")) {

stringBuilder.append(text.substring(0, text.indexOf("EOF")));

break;

}

这样修改后，**ReceiveFileActivity**在收到文件后，会检测到"EOF"字符串，并跳出循环，允许后续的文件传输正常进行。

# Lanya（发送端）和Lanya2（接收端）两个项目的交互

Lanya（发送端）和Lanya2（接收端）两个项目通过蓝牙进行交互。在这个过程中，一部手机上的lanya应用程序作为客户端，另一部手机上的lanya2应用程序作为服务器。以下是两个应用程序如何交互的详细分析：

1. Lanya（发送端）： a. 当用户启动Lanya应用程序时，应用程序会请求蓝牙和位置权限，然后显示已配对的设备列表。 b. 用户从设备列表中选择一个设备，应用程序会尝试与选定的设备建立蓝牙连接（使用不安全的RFCOMM套接字和预定义的UUID）。 c. 连接成功后，用户被引导到ConnectedActivity，这里有一个“发送文件”按钮。 d. 用户点击“发送文件”按钮，选择一个要发送的文件，然后应用程序会创建一个新线程来发送文件。 e. 在这个新线程中，应用程序会通过已建立的蓝牙连接的输出流将文件的内容发送给接收端。
2. Lanya2（接收端）： a. 当用户启动Lanya2应用程序时，应用程序会请求蓝牙权限，然后创建一个蓝牙服务器套接字。 b. 应用程序在一个新线程中等待客户端的连接请求。 c. 当有客户端连接时，应用程序会接受连接并创建一个蓝牙套接字。此时，发送端和接收端已经建立了连接。 d. 接收端会读取来自发送端的数据（文件内容），并显示在屏幕上。

在整个过程中，Lanya（发送端）和Lanya2（接收端）通过建立蓝牙连接并使用RFCOMM套接字进行交互。发送端通过输出流发送文件，而接收端通过输入流接收文件。UUID在这里起到了识别特定蓝牙服务的作用。在这个例子中，UUID是预定义的，需要确保两个应用程序使用相同的UUID以实现正确的连接。

在Android开发中，涉及到的蓝牙知识主要包括以下几个方面：

1. 蓝牙设备的发现和连接：Android设备通过蓝牙进行通信时，需要先搜索周围的蓝牙设备，并与指定设备建立连接。在Android中，可以通过**BluetoothAdapter**类进行蓝牙设备的搜索和连接操作。
2. 蓝牙通信的基础：蓝牙通信是通过建立RFCOMM通道进行数据传输的，RFCOMM通道是一种基于串行端口的虚拟通信端口，通过这个通道可以实现可靠的点对点数据传输。在Android中，可以通过**BluetoothSocket**类创建RFCOMM通道。
3. 蓝牙数据传输：蓝牙数据传输可以使用输入输出流进行读写操作。在Android中，可以通过**BluetoothSocket**类的**getInputStream()**和**getOutputStream()**方法获取输入输出流进行数据传输。
4. 蓝牙权限：在Android中，蓝牙相关的操作需要在清单文件中声明相应的权限，例如**BLUETOOTH**、**BLUETOOTH\_ADMIN**、**ACCESS\_COARSE\_LOCATION**、**ACCESS\_FINE\_LOCATION**等。
5. 蓝牙配对：在蓝牙设备之间进行数据传输之前，需要进行配对操作。在Android中，可以通过**BluetoothDevice**类的**createBond()**方法进行配对操作。
6. 蓝牙广播：蓝牙设备之间通过广播进行通信。在Android中，可以通过**BroadcastReceiver**类接收蓝牙广播消息，例如蓝牙设备的连接状态、搜索状态等。

本次实验的核心编程思想是通过客户端-服务器模型，利用蓝牙通信实现两个Android项目间的文件传输。首先，在发送文件的客户端项目中，实现蓝牙设备发现与连接功能，并允许用户选择要发送的文件。然后，通过建立的蓝牙连接，将文件内容写入蓝牙套接字。在接收文件的服务器项目中，创建一个蓝牙服务器套接字来监听并接受客户端的连接请求。随后，从蓝牙套接字中读取文件内容并将其显示在界面上。在整个过程中，还需处理权限请求与管理，确保应用程序在各种设备上正常运行。最终，这两个项目共同实现了在两台设备之间通过蓝牙传输文件的功能。

本实验成功实现了两部手机之间通过蓝牙进行文件传输的功能。通过对代码的分析，我们可以得出以下体会：

1. 权限管理：在进行蓝牙操作之前，需要确保应用程序具有相关权限。本实验涉及到的权限包括：BLUETOOTH\_CONNECT、BLUETOOTH\_SCAN和ACCESS\_FINE\_LOCATION。对于Android S及以上版本，需要在运行时请求这些权限。
2. 蓝牙连接与通信：在建立蓝牙连接时，选择UUID非常重要，因为它决定了设备之间的通信协议。在本实验中，我们采用了通用串行总线(UUID) "00001101-0000-1000-8000-00805F9B34FB"，这是一个用于串行通信的通用UUID。
3. 线程管理：在进行蓝牙通信时，为了避免阻塞UI线程，我们需要将耗时操作放在子线程中进行。在本实验中，文件的发送和接收都是在子线程中完成的。此外，在更新UI时，需要切换回主线程。
4. 数据传输：在发送文件时，我们采用了逐字节的方式将文件内容写入输出流。在接收文件时，我们从输入流中读取数据，并将其显示在TextView中。这种方法适用于各种类型的文件传输。
5. 错误处理：在实验过程中，我们需要注意异常的捕获和处理。例如，当建立蓝牙连接、读取文件或关闭socket时，可能会出现IOException。我们需要合理处理这些异常，以确保程序的稳定运行。

心路历程：

1. 初步了解需求：在开始实验前，我们首先对实验需求进行了分析，明确了实验目标，即在两部手机之间通过蓝牙实现文件传输。
2. 学习蓝牙知识：为了完成这个实验，我们花时间学习了蓝牙通信的基本概念和原理，了解了蓝牙设备之间如何建立连接以及如何传输数据。
3. 设计项目架构：在了解了实验需求和蓝牙知识后，我们设计了两个项目的架构，分别是lanya和lanya2。这两个项目分别负责发送和接收文件，确保了代码的模块化和复用性。
4. 编写并测试代码：根据项目架构，我们编写了相应的代码，并在两台手机上进行了测试，确保实现了文件传输的功能。

出现权限没有处理好，建立连接没有成功，发送文件没有发完就断开了连接这些问题，具体可以怎么来解决？

1. 权限处理：

在Android 6.0及以上版本中，需要在运行时请求敏感权限。首先，在AndroidManifest.xml中声明所需的权限，例如蓝牙相关权限：

<uses-permission android:name="android.permission.BLUETOOTH" />

<uses-permission android:name="android.permission.BLUETOOTH\_ADMIN" />

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

然后，在代码中检查并请求权限：

private void checkAndRequestPermissions() {

if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS\_FINE\_LOCATION) != PackageManager.PERMISSION\_GRANTED) {

ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.ACCESS\_FINE\_LOCATION}, REQUEST\_LOCATION\_PERMISSION);

}

}

最后，处理权限请求结果：

@Override

public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {

if (requestCode == REQUEST\_LOCATION\_PERMISSION) {

if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION\_GRANTED) {

// 权限已获得，可以继续执行蓝牙操作

} else {

// 权限被拒绝，提示用户

Toast.makeText(this, "Location permission is required for Bluetooth", Toast.LENGTH\_SHORT).show();

}

}

}

1. 建立连接： 在尝试建立连接时，确保使用正确的UUID，并在连接失败时处理异常。可以考虑使用**createRfcommSocketToServiceRecord**（安全连接）或**createInsecureRfcommSocketToServiceRecord**（不安全连接）方法。如果连接失败，可以尝试关闭并重新创建套接字。此外，可以设置一个超时机制，防止无限期等待连接。
2. 发送文件过程中的断开连接： 发送文件时，确保在发送过程中使用适当的缓冲区大小，并在写入数据后及时调用**flush()**方法。此外，在发送完所有数据后，可以使用**outputStream.close()**和**inputStream.close()**关闭输入输出流，再使用**bluetoothSocket.close()**关闭蓝牙套接字。在异常处理中，也应确保正确关闭资源。可以考虑添加重试机制，以便在出现问题时重新发送文件。

我的代码中是否处理了所有可能出现的异常情况

1. 在**ConnectedActivity**类的**onCreate()**方法中，处理**bluetoothSocket.connect()**可能抛出的**IOException**。这里您已经处理了该异常，但可以进一步完善，例如，添加重试机制或在捕获异常后关闭蓝牙套接字。
2. 在**ConnectedActivity**类的**sendFile()**方法中，处理**getContentResolver().openInputStream(fileUri)**可能抛出的**FileNotFoundException**。此外，**outputStream.write()**和**outputStream.flush()**也可能抛出**IOException**。您已经处理了这些异常，但可以考虑在出现问题时重新发送文件。
3. 在**ReceiveFileActivity**类的**createServerSocket()**方法中，处理**bluetoothAdapter.listenUsingInsecureRfcommWithServiceRecord()**可能抛出的**IOException**。您已经处理了这个异常，但在异常处理代码中应该结束当前活动，避免程序继续运行。您可以使用**finish()**方法。
4. 在**ReceiveFileActivity**类的**readDataFromSocket()**方法中，处理**socket.getInputStream()**可能抛出的**IOException**。同时，**inputStream.read()**也可能抛出**IOException**。您已经处理了这些异常，但可以考虑在出现问题时关闭套接字和输入流。

什么叫抛出的IOException

"抛出的IOException"是指在代码执行过程中可能发生的一种异常（错误）类型。IOException（输入/输出异常）是Java中表示与输入、输出操作相关的错误的一个类。当程序在执行涉及输入或输出的操作（如文件读写、网络通信等）时，如果遇到问题，通常会抛出这种类型的异常。

在Java中，当程序执行过程中遇到异常情况时，通常会创建一个表示异常的对象（如**IOException**），然后抛出（throw）这个对象。抛出异常的目的是通知调用者（调用了可能出现异常的方法的代码）发生了异常，并让调用者有机会处理这个异常。如果调用者没有捕获（catch）并处理这个异常，异常会继续向上抛出，直到被处理或导致程序终止。

为了避免程序因为未处理的异常而意外终止，我们需要使用**try**-**catch**语句来捕获可能抛出的异常。**try**块包含可能抛出异常的代码，**catch**块用于捕获和处理异常。当**try**块中的代码抛出异常时，程序会跳到与该异常类型匹配的**catch**块中执行。这样，我们就可以在**catch**块中编写处理异常的代码，例如记录错误信息、重试操作或通知用户。

java有多少异常对象，除了IOException，还有什么

Java中有很多异常类，它们都继承自**java.lang.Throwable**基类。这些异常类可以分为两大类：**Error**和**Exception**。

1. **Error**：这是Java程序中无法处理的严重问题，例如系统内存不足、虚拟机错误等。程序通常无法从这些错误中恢复，因此我们不需要关注它们。
2. **Exception**：这是程序中可能发生的问题，可以通过捕获和处理来解决。**Exception**类有很多子类，代表各种不同类型的异常。它们可以细分为两类：
   * 受检异常（Checked Exceptions）：需要在代码中显式处理（使用**try**-**catch**语句捕获）的异常。如果方法内部抛出受检异常，必须在方法签名中声明该异常（使用**throws**关键字）。例如：**FileNotFoundException**、**ParseException**等。
   * 非受检异常（Unchecked Exceptions）：不强制在代码中显式处理的异常，通常是程序员的错误导致的，例如**NullPointerException**、**IndexOutOfBoundsException**、**ArithmeticException**等。

除了**IOException**之外，还有很多其他异常类。以下是一些常见的异常类：

* **NullPointerException**：当尝试访问一个空引用（null）的对象或方法时抛出。
* **NumberFormatException**：当尝试将一个不合适的字符串转换为数字时抛出。
* **ArrayIndexOutOfBoundsException**：当访问数组时使用了非法索引（负数或超出数组长度）抛出。
* **FileNotFoundException**：当尝试打开一个不存在的文件时抛出。
* **ClassCastException**：当尝试将对象强制转换为不兼容类型时抛出。
* **IllegalArgumentException**：当方法接收到非法参数时抛出。
* **MalformedURLException**：当尝试创建一个不正确的URL时抛出。
* **InterruptedException**：当一个线程在等待、休眠或占用时被另一个线程中断时抛出。

这只是异常类的一部分，Java标准库中还有很多其他异常类。在实际编程中，你可能会遇到这些以及其他异常类。

整体项目重点部分

1. 权限检查：在安卓应用中，需要确保已经获得了蓝牙连接和读写文件的权限。使用**ActivityCompat.checkSelfPermission()**方法和**PackageManager.PERMISSION\_GRANTED**常量来检查权限。
2. 蓝牙设备连接：使用**BluetoothAdapter**来获取设备列表，**BluetoothDevice**来表示单个设备，以及**BluetoothSocket**来建立与设备的连接。在**ConnectedActivity**中，通过调用**device.createInsecureRfcommSocketToServiceRecord(uuid)**创建蓝牙连接。
3. 文件选择和传输：在**ConnectedActivity**中，使用**Intent.ACTION\_GET\_CONTENT**选择要发送的文件，并使用**getContentResolver().openInputStream(fileUri)**获取该文件的输入流。将文件内容读取到缓冲区中，并通过蓝牙连接的输出流发送给接收方。
4. 文件接收：在**ReceiveFileActivity**中，创建一个**BluetoothServerSocket**来监听来自发送方的连接请求。一旦连接建立，使用**socket.getInputStream()**获取输入流，并从输入流中读取文件内容。这里将文件内容显示在屏幕上，但在实际应用中，可以将接收到的内容保存到本地文件或进行其他处理。
5. 异常处理：在代码中，你已经添加了一些异常处理来处理可能出现的问题，例如蓝牙连接失败、文件读写错误等。这有助于提高代码的健壮性和稳定性。
6. 资源清理：在**onDestroy()**方法中，确保关闭蓝牙连接和相关资源，以避免内存泄漏或其他问题。
7. UUID的使用：在**ConnectedActivity**和**ReceiveFileActivity**中，UUID被用于表示蓝牙服务。这是一个标准的蓝牙串口服务UUID，用于在两个设备之间建立RFCOMM连接。
8. 视图初始化：在**ConnectedActivity**中，使用**initView()**方法来初始化视图。这种将视图初始化从**onCreate()**方法中抽离出来的方式有助于提高代码的可读性和可维护性。
9. 蓝牙连接权限检查：在**ConnectedActivity**中，针对安卓12及以上版本，检查**Manifest.permission.BLUETOOTH\_CONNECT**权限。这是在新版本中引入的权限，需要注意适配。
10. onActivityResult的实现：在**ConnectedActivity**中，通过**onActivityResult()**方法处理文件选择请求。根据请求码（**SELECT\_FILE\_REQUEST**）和结果码（**RESULT\_OK**），调用**sendFile()**方法进行文件传输。
11. 使用**BluetoothAdapter**获取设备：在**DeviceListActivity**中，通过**bluetoothAdapter.getBondedDevices()**获取已配对的设备列表，然后使用适配器将设备显示在列表中。
12. 设备列表点击事件处理：在**DeviceListActivity**中，为列表项添加点击事件处理，通过**Intent**将选中的设备对象传递给**ConnectedActivity**。
13. 在**ReceiveFileActivity**中创建**BluetoothServerSocket**：通过**bluetoothAdapter.listenUsingInsecureRfcommWithServiceRecord()**创建**BluetoothServerSocket**，并在新线程中等待来自发送方的连接请求。
14. 文件内容接收与显示：在**ReceiveFileActivity**中，使用**StringBuilder**来将接收到的文件内容拼接成字符串，并在UI线程中将其显示在**receivedFileContentTextView**中。
15. onRequestPermissionsResult的实现：在**ReceiveFileActivity**中，处理请求蓝牙连接权限的结果。如果权限被授予，则调用**createServerSocket()**方法继续进行蓝牙连接。
16. 蓝牙Socket的关闭：在**ConnectedActivity**和**ReceiveFileActivity**的**onDestroy()**方法中，确保蓝牙Socket在活动销毁时被正确关闭。
17. 异常处理：在发送和接收文件的过程中，都有使用**try-catch**语句处理可能抛出的**IOException**，并在UI线程中更新视图以显示相关错误信息。
18. 主线程与子线程交互：在文件传输和接收过程中，涉及到主线程与子线程之间的交互。通过**runOnUiThread()**方法在UI线程中更新视图。
19. 使用BluetoothDevice.createInsecureRfcommSocketToServiceRecord()方法建立到远程设备的连接。此方法允许创建一个不安全的RFCOMM套接字，用于连接到远程设备的蓝牙串口服务。
20. 在ConnectedActivity中，使用FileInputStream读取选定文件的内容。将文件内容读入缓冲区并发送到连接的蓝牙设备。
21. 在ReceiveFileActivity中，使用InputStream从连接的蓝牙设备接收文件内容。读取数据并将其添加到StringBuilder中，用于显示接收到的文件内容。
22. 在ConnectedActivity和ReceiveFileActivity中，为连接的蓝牙设备创建新线程。新线程用于处理文件传输和接收，避免阻塞主线程。
23. 在ConnectedActivity中，通过sendFile()方法发送文件。将文件名和文件内容作为参数传递给该方法，然后将数据写入蓝牙连接的输出流。
24. 在DeviceListActivity中，使用ArrayAdapter将已配对的蓝牙设备显示在列表中。通过适配器将设备列表数据与视图绑定，以便在屏幕上呈现设备列表。
25. 在DeviceListActivity中，使用Intent传递选中设备的信息。将选中的设备作为Parcelable对象放入Intent，然后将其传递给ConnectedActivity。
26. 在ReceiveFileActivity中，检查设备是否支持蓝牙。如果BluetoothAdapter.getDefaultAdapter()返回null，则表示设备不支持蓝牙，结束活动。
27. 在ConnectedActivity和ReceiveFileActivity中，遇到异常时在UI线程中更新视图以显示错误信息。使用runOnUiThread()方法确保在主线程中更新视图。
28. 在MainActivity中，使用OnClickListener监听按钮点击事件。当点击按钮时，启动ReceiveFileActivity。
29. 在整个项目中，通过Android Studio的资源管理器，确保所有资源（如布局文件和字符串资源）都正确地引用和使用。

**总结如下**

1. 使用BluetoothAdapter检测设备是否支持蓝牙并管理蓝牙连接。
2. 实现DeviceListActivity以显示已配对的蓝牙设备列表。
3. 使用UUID创建唯一的服务标识符，以在设备间建立稳定的蓝牙连接。
4. 在ConnectedActivity中创建蓝牙套接字并建立连接。
5. 在ReceiveFileActivity中创建蓝牙服务器套接字并等待接收连接。
6. 处理Android版本和蓝牙权限的变化，确保应用在各种设备和Android版本上正常运行。
7. 使用新线程处理蓝牙连接和文件传输，避免阻塞主线程。
8. 在ConnectedActivity中使用FileInputStream读取文件内容并通过蓝牙输出流发送。
9. 在ReceiveFileActivity中使用InputStream接收文件内容并显示在TextView上。