Lập trình mạng với Android

Phát triển ứng dụng trên thiết bị di động nâng cao

Giới thiệu về lập trình mạng trên Android

- Khái niệm lập trình mạng
- Ứng dụng của lập trình mạng trong Android
- Các giao thức mạng phổ biến: HTTP, TCP, UDP

Khái niệm lập trình mạng

- Lập trình mạng là quá trình tạo ra các ứng dụng có khả năng giao tiếp và trao đổi dữ liệu qua mạng. Trong Android, lập trình mạng cho phép ứng dụng kết nối với các dịch vụ web, server hoặc các thiết bị khác để truyền tải dữ liệu.

Đặc điểm của lập trình mạng

- **Giao tiếp giữa các thiết bị**: Ứng dụng có thể gửi và nhận dữ liệu từ các nguồn khác nhau (máy chủ, dịch vụ đám mây, API RESTful, MQTT, WebSocket...).
- **Kết nối đa dạng**: Hỗ trợ nhiều giao thức như HTTP, TCP, UDP, WebSocket...
- **Bất đồng bộ (Asynchronous)**: Hạn chế việc chặn luồng chính của ứng dụng bằng cách sử dụng các kỹ thuật như AsyncTask, Coroutines (Kotlin), RxJava.
- **Bảo mật dữ liệu**: Thường sử dụng mã hóa SSL/TLS để bảo vệ dữ liệu khi truyền tải qua mạng.

Ứng dụng của lập trình mạng trên thiết bị di động?

Ứng dụng của lập trình mạng trong Android

- Lập trình mạng là thành phần cốt lõi trong nhiều ứng dụng di động hiện đại. Một số ứng dụng điển hình của lập trình mạng trong Android:
- **Ứng dụng mạng xã hội**: Gửi và nhận tin nhắn (Facebook, Zalo, WhatsApp).
- **Ứng dụng thương mại điện tử**: Truy cập dữ liệu sản phẩm, đặt hàng trực tuyến (Shopee, Tiki).
- **Ứng dụng streaming**: Xem video trực tuyến (YouTube, Netflix).
- **Ứng dụng IoT**: Điều khiển thiết bị từ xa, giám sát hệ thống.
- **Ứng dụng bản đồ và điều hướng**: Kết nối với API bản đồ để tìm đường đi, chia sẻ vị trí (Google Maps).
- **Ứng dụng tài chính**: Đồng bộ dữ liệu ngân hàng, ví điện tử.

Các giao thức mạng phổ biến trong Android

HTTP (Hypertext Transfer Protocol)

- Là giao thức truyền tải siêu văn bản, chủ yếu được sử dụng để giao tiếp giữa ứng dụng Android và các API RESTful trên web.
- **Thư viện phổ biến**: HttpURLConnection, OkHttp, Retrofit (Android).
- **Ưu điểm**:
  - Dễ sử dụng, hỗ trợ nhiều phương thức (GET, POST, PUT, DELETE).
  - Tích hợp sẵn trên Android.
- **Nhược điểm**:
  - Chậm hơn so với WebSocket khi cần cập nhật dữ liệu liên tục.

Ví dụ:
Gửi request GET để lấy dữ liệu từ API

```
public static String getDataFromUrl(String urlString) throws IOException {
    URL url = new URL(urlString);
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod("GET");
    BufferedReader reader = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    StringBuilder result = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) {
        result.append(line);
    }
    return result.toString();
}
```

Các giao thức mạng phổ biến trong Android

- **TCP (Transmission Control Protocol)**
- Là giao thức kết nối có kiểm soát, đảm bảo dữ liệu được gửi đi một cách đầy đủ và chính xác.
- **Thường sử dụng trong**: Chat, truyền file, ứng dụng yêu cầu độ tin cậy cao.
- **Ưu điểm**:
  - Đảm bảo toàn vẹn dữ liệu.
  - Gói tin không bị mất hoặc lặp lại.
- **Nhược điểm**:
  - Hiệu suất thấp hơn UDP do có cơ chế kiểm tra và xác nhận dữ liệu.

Ví dụ:
Gửi tin nhắn TCP từ Android đến Server

```
Socket socket = new Socket("192.168.1.100", 8080);
OutputStream outputStream = socket.getOutputStream();
PrintWriter writer = new PrintWriter(outputStream, true);
writer.println("Hello, Server!");
```

Các giao thức mạng phổ biến trong Android

- **UDP (User Datagram Protocol)**
- Giao thức truyền tải dữ liệu không kết nối, nhanh hơn TCP nhưng không đảm bảo toàn vẹn dữ liệu.
- **Thường sử dụng trong**: Streaming video, game online, VoIP.
- **Ưu điểm**:
  - Tốc độ nhanh, không cần xác nhận dữ liệu.
  - Giảm tải tài nguyên hệ thống.
- **Nhược điểm**:
  - Dữ liệu có thể bị mất hoặc đến không theo thứ tự.

Ví dụ:
Gửi dữ liệu UDP từ Android

```
DatagramSocket socket = new DatagramSocket();
InetAddress serverAddress = InetAddress.getByName("192.168.1.100");
byte[] message = "Hello UDP".getBytes();
DatagramPacket packet = new DatagramPacket(message, message.length, serverAddress, 8080);
socket.send(packet);
socket.close();
```

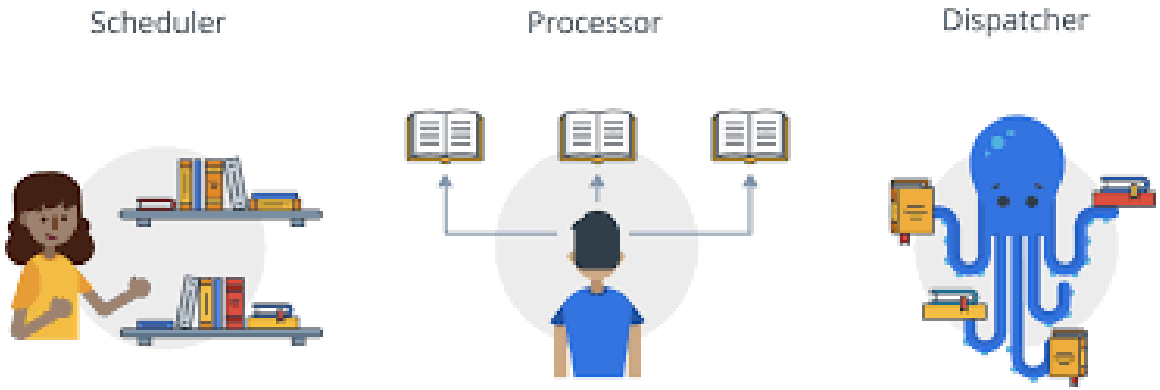
Các giao thức mạng phổ biến trong Android

Giao thức	Loại	Đặc điểm chính	Ứng dụng phổ biến
HTTP	Kết nối	Đảm bảo dữ liệu, dễ sử dụng	Giao tiếp với API RESTful
TCP	Kết nối	Độ tin cậy cao, tốc độ trung bình	Ứng dụng chat, truyền file
UDP	Không kết nối	Tốc độ nhanh, không đảm bảo dữ liệu	Streaming, game online

Lưu ý: Việc lựa chọn giao thức phụ thuộc vào yêu cầu của ứng dụng. Nếu cần đảm bảo dữ liệu, hãy sử dụng TCP. Nếu cần tốc độ cao, hãy sử dụng UDP.

Lập trình đa luồng trong Android

- Tại sao cần lập trình đa luồng?
- Các cách triển khai: Thread, AsyncTask, Handler, Executor
- Ví dụ: Tạo một tiến trình tải dữ liệu từ Internet bằng Thread



Tại sao cần lập trình đa luồng?

- Lập trình đa luồng (Multithreading) cho phép một ứng dụng thực hiện nhiều tác vụ cùng một lúc, giúp cải thiện hiệu suất và trải nghiệm người dùng.
- **Lợi ích của lập trình đa luồng:**
- **Giữ ứng dụng mượt mà**: Android có **Main Thread (UI Thread)** chịu trách nhiệm cập nhật giao diện. Nếu một tác vụ nặng (như tải dữ liệu từ mạng) chạy trên Main Thread, ứng dụng có thể bị **treo (ANR – Application Not Responding)**.
- **Tăng tốc xử lý**: Các tác vụ độc lập có thể chạy song song, giảm thời gian chờ đợi.
- **Cải thiện trải nghiệm người dùng**: Giúp ứng dụng phản hồi nhanh hơn khi thực hiện các tác vụ nền như tải hình ảnh, xử lý dữ liệu, hoặc tính toán phức tạp.

Ví dụ về vấn đề khi không dùng đa luồng

- Giả sử một ứng dụng Android tải dữ liệu từ Internet mà không sử dụng đa luồng, UI sẽ bị chặn cho đến khi dữ liệu được tải xong. Điều này làm giảm trải nghiệm người dùng vì họ không thể tương tác với ứng dụng trong lúc tải dữ liệu.

Các cách triển khai lập trình đa luồng trong Android

Thread (Luồng cơ bản)

•Thread là cách cơ bản nhất để tạo một luồng mới trong Java/Android.

•**Ưu điểm**: Đơn giản, dễ sử dụng cho các tác vụ nhỏ.

•**Nhược điểm**: Không thể cập nhật UI trực tiếp từ Thread phụ, cần sử dụng Handler.

Ví dụ:

```
class MyThread extends Thread {
    @Override
    public void run() {
        // Công việc chạy trên luồng mới
        Log.d("THREAD", "Luồng đang chạy...");
    }
}

// Khởi chạy luồng
MyThread thread = new MyThread();
thread.start();
```

Các cách triển khai lập trình đa luồng trong Android

AsyncTask (Đã lỗi thời, nhưng vẫn phổ biến trong các dự án cũ)

•AsyncTask được thiết kế để xử lý các tác vụ nền mà vẫn có thể cập nhật UI.

•**Ưu điểm**: Dễ sử dụng, hỗ trợ cập nhật UI (trong onPostExecute).

•**Nhược điểm**: Không thích hợp cho các tác vụ dài vì có thể gây rò rỉ bộ nhớ

Ví dụ

```
class DownloadTask extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... urls) {
        // Tải dữ liệu từ Internet
        return "Dữ liệu đã tải";
    }

    @Override
    protected void onPostExecute(String result) {
        // Cập nhật UI
        Log.d("ASYNCTASK", result);
    }
}

// Chạy AsyncTask
new DownloadTask().execute("https://example.com/data.json");
```

Lưu ý: AsyncTask đã bị loại bỏ từ Android API 30. Hãy sử dụng Executor hoặc Coroutines thay thế.

Các cách triển khai lập trình đa luồng trong Android

Handler (Quản lý luồng một cách linh hoạt hơn)

- Handler cho phép giao tiếp giữa các luồng bằng cách gửi và xử lý thông điệp (Message).
- **Ưu điểm**: Quản lý luồng tốt hơn, có thể giao tiếp với UI Thread.
- **Nhược điểm**: Cần nhiều code hơn so với AsyncTask.

Ví dụ

```
Handler handler = new Handler(Looper.getMainLooper());

new Thread(new Runnable() {
    @Override
    public void run() {
        String data = "Dữ liệu tải xong";

        // Gửi thông điệp đến Main Thread
        handler.post(new Runnable() {
            @Override
            public void run() {
                Log.d("HANDLER", data);
            }
        });
    }
}).start();
```

Các cách triển khai lập trình đa luồng trong Android

Executor (Giải pháp tốt hơn AsyncTask)

- Executor giúp quản lý các luồng một cách tối ưu hơn bằng cách sử dụng **Thread Pool**.
- **Ưu điểm**: Hiệu suất tốt hơn, có thể tái sử dụng luồng.
- **Nhược điểm**: Cần viết code nhiều hơn một chút.

Ví dụ

```
ExecutorService executor = Executors.newSingleThreadExecutor();
executor.execute(new Runnable() {
    @Override
    public void run() {
        Log.d("EXECUTOR", "Luồng đang chạy...");
    }
});
```

Các cách triển khai lập trình đa luồng trong Android

- Với các tác vụ đơn giản → Dùng Thread hoặc Handler.
- Với các tác vụ phức tạp, nhiều tiến trình chạy đồng thời → Dùng Executor.
- Với Android hiện đại → Dùng Coroutines trong Kotlin.

Ví dụ: Tạo một tiến trình tải dữ liệu từ Internet bằng Thread

Mô tả

- Ứng dụng sẽ tải dữ liệu từ một URL.
- Sử dụng Thread để chạy tác vụ trên luồng nền.
- Cập nhật giao diện bằng Handler.

```java
public class MainActivity extends AppCompatActivity {
    private TextView textView;
    private Handler handler = new Handler(Looper.getMainLooper());

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        textView = findViewById(R.id.textView);

        // Bắt đầu tải dữ liệu
        new Thread(new Runnable() {
            @Override
            public void run() {
                final String data = loadDataFromInternet("https://example.com/data.json");

                // Cập nhật UI trên Main Thread
                handler.post(new Runnable() {
                    @Override
                    public void run() {
                        textView.setText(data);
                    }
                });
            }
        }).start();
    }
```

```java
private String loadDataFromInternet(String urlString) {
    try {
        URL url = new URL(urlString);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        BufferedReader reader = new BufferedReader(new InputStreamReader(conn.getInputStream()));
        StringBuilder result = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null) {
            result.append(line);
        }
        return result.toString();
    } catch (Exception e) {
        return "Lỗi tải dữ liệu";
    }
}
```

Các cách triển khai lập trình đa luồng trong Android

Phương pháp	Ưu điểm	Nhược điểm	Khi nào nên dùng?
Thread	Đơn giản, dễ dùng	Không cập nhật UI trực tiếp	Tác vụ nhỏ, đơn giản
AsyncTask (cũ)	Dễ cập nhật UI	Không phù hợp cho tác vụ dài	Dự án cũ, không khuyến khích
Handler	Gửi message giữa các luồng	Cần nhiều code hơn	Khi cần giao tiếp giữa UI Thread và Worker Thread
Executor	Quản lý luồng tốt, hiệu suất cao	Phức tạp hơn `Thread`	Khi cần quản lý nhiều luồng

Kết nối HTTP trong Android

- Giới thiệu về HttpURLConnection
- Cách gửi yêu cầu GET/POST
- Xử lý phản hồi từ server
- Ví dụ: Gửi yêu cầu GET đến API

Giới thiệu về HttpURLConnection

•HttpURLConnection là một lớp trong Java được sử dụng để giao tiếp với máy chủ thông qua giao thức HTTP.
•Hỗ trợ các phương thức HTTP như GET, POST, PUT, DELETE.
•Cho phép thiết lập header, gửi dữ liệu, đọc phản hồi từ server.
•Là một lựa chọn tốt khi cần giao tiếp mạng mà không muốn sử dụng thư viện bên ngoài.

Cách gửi yêu cầu GET/POST

Gửi yêu cầu GET

•Yêu cầu GET được sử dụng để lấy dữ liệu từ server mà không thay đổi dữ liệu trên máy chủ.

•Cú pháp cơ bản:

```
URL url = new URL("https://api.example.com/data");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");

int responseCode = conn.getResponseCode();
if (responseCode == HttpURLConnection.HTTP_OK) {
    BufferedReader reader = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    StringBuilder response = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) {
        response.append(line);
    }
    reader.close();
    System.out.println("Response: " + response.toString());
} else {
    System.out.println("GET request failed");
}
```

Cách gửi yêu cầu GET/POST

Gửi yêu cầu POST

- Dùng để gửi dữ liệu lên server, ví dụ như đăng nhập, đăng ký.
- Cú pháp:

```java
URL url = new URL("https://api.example.com/post");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("POST");
conn.setRequestProperty("Content-Type", "application/json");
conn.setDoOutput(true);

String jsonInputString = "{\"username\": \"test\", \"password\": \"1234\"}";
try (OutputStream os = conn.getOutputStream()) {
    byte[] input = jsonInputString.getBytes("utf-8");
    os.write(input, 0, input.length);
}

int responseCode = conn.getResponseCode();
if (responseCode == HttpURLConnection.HTTP_OK) {
    BufferedReader reader = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    StringBuilder response = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) {
        response.append(line);
    }
    reader.close();
    System.out.println("Response: " + response.toString());
} else {
    System.out.println("POST request failed");
}
```

Xử lý phản hồi từ server

- Khi gửi yêu cầu, server sẽ phản hồi mã trạng thái HTTP:
  - 200 OK – Thành công.
  - 400 Bad Request – Yêu cầu không hợp lệ.
  - 401 Unauthorized – Chưa đăng nhập.
  - 500 Internal Server Error – Lỗi server.
- Cần kiểm tra mã phản hồi (responseCode) trước khi đọc dữ liệu.

Ví dụ: Gửi yêu cầu GET đến API

Dưới đây là ví dụ gửi yêu cầu GET đến API công khai:

```
public class MainActivity extends AppCompatActivity {
    private TextView textView;
    private Handler handler = new Handler(Looper.getMainLooper());

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        textView = findViewById(R.id.textView);

        new Thread(new Runnable() {
            @Override
            public void run() {
                final String data = fetchAPIData("https://jsonplaceholder.typicode.com/posts/1");

                handler.post(new Runnable() {
                    @Override
                    public void run() {
                        textView.setText(data);
                    }
                });
            }
        }).start();
    }
```

```
    private String fetchAPIData(String urlString) {
        try {
            URL url = new URL(urlString);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");

            int responseCode = conn.getResponseCode();
            if (responseCode == HttpURLConnection.HTTP_OK) {
                BufferedReader reader = new BufferedReader(new InputStreamReader(conn.getInputStream()))
                StringBuilder result = new StringBuilder();
                String line;
                while ((line = reader.readLine()) != null) {
                    result.append(line);
                }
                reader.close();
                return result.toString();
            } else {
                return "Lỗi: " + responseCode;
            }
        } catch (Exception e) {
            return "Lỗi tải dữ liệu";
        }
    }
}
```

Lập trình Socket trên Android

- Giới thiệu lập trình hướng kết nối (TCP)
- Giới thiệu lập trình không kết nối (UDP)
- Gửi/Nhận dữ liệu từ client Android đến server PC
- Ví dụ: Gửi tin nhắn TCP từ Android đến server

LẬP TRÌNH SOCKET TRÊN ANDROID

Giới thiệu về Lập trình Socket

Lập trình Socket là một kỹ thuật giúp các thiết bị có thể giao tiếp với nhau thông qua mạng. Socket có thể hoạt động trên nhiều giao thức khác nhau, phổ biến nhất là:

- **TCP (Transmission Control Protocol):** Kết nối tin cậy, đảm bảo dữ liệu được truyền chính xác.
- **UDP (User Datagram Protocol):** Kết nối không tin cậy, tốc độ nhanh hơn nhưng không đảm bảo toàn vẹn dữ liệu.

Ứng dụng của Lập trình Socket trong Android

Lập trình Socket trên Android có nhiều ứng dụng thực tế, bao gồm:

- **Ứng dụng Chat:** Nhắn tin thời gian thực giữa các thiết bị.
- **Điều khiển thiết bị IoT:** Điều khiển thiết bị thông minh như đèn, camera giám sát.
- **Truy xuất dữ liệu từ Server:** Gửi yêu cầu đến máy chủ và nhận phản hồi.
- **Game Online:** Đồng bộ dữ liệu giữa các người chơi trong thời gian thực.

Kiến trúc Client - Server trong Lập trình Socket

Lập trình Socket thường sử dụng mô hình **Client - Server**, trong đó:

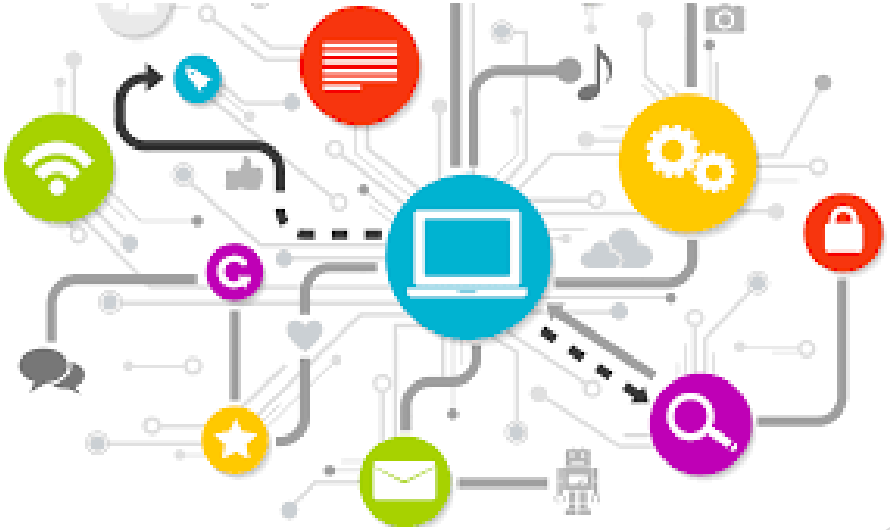
- **Client (Ứng dụng Android):** Gửi yêu cầu đến Server và nhận phản hồi.
- **Server (Máy chủ PC hoặc Cloud):** Xử lý yêu cầu và gửi dữ liệu về cho Client.

Các Bước Triển Khai Lập trình Socket trong Android

Bước 1: Xây dựng Server xử lý kết nối (Java/Python/PHP).
Bước 2: Viết Client trên Android để gửi và nhận dữ liệu.
Bước 3: Kiểm thử và xử lý lỗi để đảm bảo kết nối ổn định.

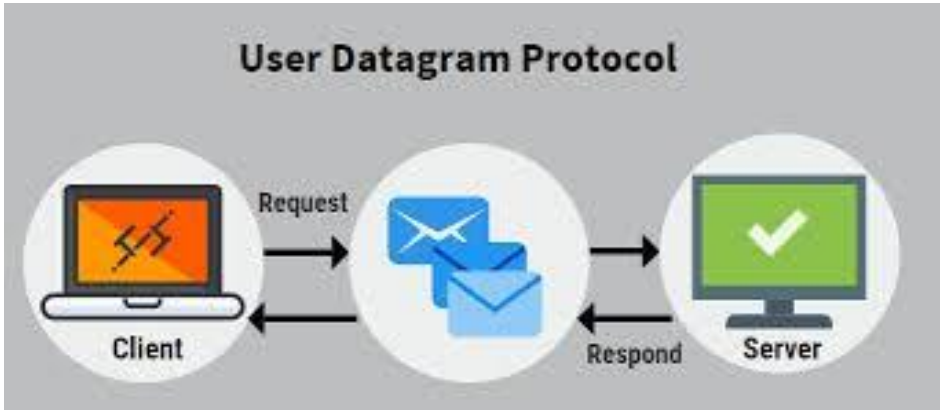
Lập trình hướng kết nối (TCP)

- TCP là giao thức kết nối, đảm bảo dữ liệu được truyền chính xác
- Dùng trong chat, truyền file, đồng bộ dữ liệu



Lập trình không kết nối (UDP)

- UDP là giao thức không kết nối, nhanh nhưng không đảm bảo dữ liệu
- Dùng trong game online, streaming, VoIP



Gửi/Nhận dữ liệu từ Android đến Server

- Android gửi yêu cầu qua socket
- Server lắng nghe và xử lý dữ liệu
- Trả kết quả về client

Ví dụ: Gửi tin nhắn TCP từ Android

- **Mô tả hoạt động**

1. **Server trên PC** lắng nghe kết nối từ client và nhận tin nhắn.
2. **Client Android** kết nối đến server qua địa chỉ IP và cổng.
3. **Client gửi tin nhắn**, server nhận và phản hồi lại.
4. **Client hiển thị phản hồi từ server**.

1. Server (Chạy trên PC - Java)

```java
import java.io.*;
import java.net.*;

public class TCPServer {
    public static void main(String[] args) {
        int port = 12345; // Cổng lắng nghe của server
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Server đang lắng nghe trên cổng " + port);
            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("Kết nối từ: " + socket.getInetAddress());

                // Đọc dữ liệu từ client
                BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                String message = in.readLine();
                System.out.println("Tin nhắn nhận được: " + message);

                // Phản hồi client
                PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
                out.println("Server nhận được: " + message);

                socket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
```

2. Client (Ứng dụng Android - Java)

- **Thêm quyền mạng vào AndroidManifest.xml**

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

Gửi tin nhắn từ Android đến server

```
import android.os.AsyncTask;
import android.util.Log;
import java.io.*;
import java.net.*;

public class TCPClient extends AsyncTask<String, Void, String> {
    private static final String SERVER_IP = "192.168.1.100"; // Địa chỉ IP của server (cần thay đổi)
    private static final int SERVER_PORT = 12345;

    @Override
    protected String doInBackground(String... params) {
        String message = params[0];
        try (Socket socket = new Socket(SERVER_IP, SERVER_PORT);
             PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {

            out.println(message);
            return in.readLine(); // Nhận phản hồi từ server
        } catch (IOException e) {
            Log.e("TCPClient", "Lỗi kết nối", e);
            return null;
        }
    }
```

```
    @Override
    protected void onPostExecute(String response) {
        if (response != null) {
            Log.d("TCPClient", "Phản hồi từ server: " + response);
        }
    }
}
```

- Sử dụng TCPClient trong Activity

```
public void sendMessage(View view) {
    String message = "Xin chào, server!";
    new TCPClient().execute(message);
}
```

Bài tập thực hành

Bài 1: Kiểm tra xem điện thoại có kết nối mạng không. Mở một kết nối đến một trang web (ví dụ: https://www.google.com).

Bài 2: Đếm số bằng Thread

- Tạo một ứng dụng có nút Start để bắt đầu đếm số từ 1 đến 10.
- Số sẽ hiển thị trên TextView, mỗi giây tăng 1 đơn vị.
- Khi đếm xong, hiển thị "Hoàn thành!".

Bài 3: Ứng dụng gồm Client và Server, hai điện thoại có thể gửi tin nhắn qua WiFi hoặc cùng một mạng LAN.

- Điện thoại Server sẽ lắng nghe kết nối từ Client.
- Điện thoại Client có thể gửi tin nhắn đến Server.
- Tin nhắn hiển thị trên giao diện cả hai điện thoại.

Bài tập luyện tập

- Bài 4: Tạo ứng dụng tải dữ liệu JSON từ API và hiển thị trên RecyclerView
- Bài 5: Viết chương trình kiểm tra tốc độ mạng bằng HTTP request
- Bài 6: Tạo một ứng dụng gửi tin nhắn giữa hai thiết bị Android qua TCP
- Bài 7: Xây dựng ứng dụng đo độ trễ (ping) đến một server

Bài 4: Tạo ứng dụng tải dữ liệu JSON từ API và hiển thị trên RecyclerView

Bước 1: Thêm quyền Internet vào AndroidManifest.xml

xml

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

Bước 2: Thêm thư viện Retrofit vào build.gradle

implementation 'com.squareup.retrofit2:retrofit:2.9.0' implementation 'com.squareup.retrofit2:converter-gson:2.9.0'

Bước 3: Tạo interface API

public interface ApiService { @GET("posts") // Đường dẫn API (ví dụ: https://jsonplaceholder.typicode.com/posts) Call<List<Post>> getPosts(); }

Bước 4: Hiển thị dữ liệu trên RecyclerView

- Tạo Post model
- Viết PostAdapter
- Sử dụng RecyclerView trong Activity

Bài 5: Viết chương trình kiểm tra tốc độ mạng bằng HTTP request

Bước 1: Gửi yêu cầu HTTP bằng OkHttp

```
OkHttpClient client = new OkHttpClient();
long startTime = System.currentTimeMillis();
Request request = new
Request.Builder().url("https://www.google.com").build();
Response response = client.newCall(request).execute();
long endTime = System.currentTimeMillis();
long speed = endTime - startTime;
```

Bước 2: Hiển thị kết quả lên UI

- Dùng TextView để hiển thị thời gian phản hồi
- Tạo button để bắt đầu đo

Bài 6: Tạo một ứng dụng gửi tin nhắn giữa hai thiết bị Android qua TCP

Bước 1: Cấu hình server trên Android

```
ServerSocket serverSocket = new ServerSocket(12345);
Socket client = serverSocket.accept();
BufferedReader in = new BufferedReader(
        new InputStreamReader(client.getInputStream()));
String receivedMsg = in.readLine();
```

Bước 2: Client gửi tin nhắn

```
Socket socket = new Socket("192.168.1.101", 12345);
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
out.println("Hello from Client!");
```

Bước 3: Xây dựng giao diện người dùng

- Dùng RecyclerView hiển thị tin nhắn
- Button gửi tin nhắn

Bài 7: Xây dựng ứng dụng đo độ trễ (ping) đến một server

Bước 1: Thực hiện ping bằng Process

```
Process process = Runtime.getRuntime().exec
                                        ("ping -c 1 google.com");
BufferedReader reader = new BufferedReader(
        new InputStreamReader(process.getInputStream()));
String line;
while ((line = reader.readLine()) != null) {
        Log.d("Ping", line);
}
```

Bước 2: Hiển thị kết quả trên UI

- Button để bắt đầu đo ping
- TextView hiển thị kết quả