10 Мережеві можливості, доступ до веб-сервісів

10.1 Основи мережевої взаємодії. Синхронні і асинхронні запити

На практиці домінуючою моделлю мережевої взаємодії є стек протоколів TCP/IP, який в цілому відповідає моделі OSI, однак не регламентує обов'язкову наявність усіх рівнів в ній. Як випливає з назви, обов'язковими протоколами в ній є TCP (або його альтернатива UDP), а також IP, які реалізують транспортний і мережевий рівень моделі OSI.

Рівні TCP/IP стека:

· 4й - Прикладний рівень (Process/Application) — відповідає трьом верхнім рівням моделі OSI (проте, не обов'язково реалізує функціональність їх всіх)

· 3й - Транспортний рівень (Transport) — відповідає транспортному рівню моделі OSI

· 2й - Міжмережевий рівень (Internet) — відповідає мережному рівню моделі OSI

· 1й - Рівень мережевого доступу (Network Access) — відповідає двом нижнім рівням моделі OSI

У цій моделі верхній і нижній рівні включають в себе декілька рівнів моделі OSI і в різних випадках вони можуть бути реалізовані як одним протоколом взаємодії, так і декількома (відповідними окремим рівням). Наприклад, протокол HTTP реалізує рівні прикладної та представлення, а протокол TLS — сеансовий і представлення, а в поєднанні між собою вони можуть покрити всі 3 верхніх рівня. При цьому протокол HTTP працює і самостійно, і в цьому випадку, оскільки він не реалізує сеансовий рівень, HTTP-з'єднання називають "stateless", тобто не мають стану.

Модель TCP/IP також називають пісочним годинником, оскільки посередині в ній знаходиться один протокол, IP, а протоколи під ним і над ним є дуже різноманітними і покривають різні сценарії використання. Стандартизація протоколу посередині дає велику гнучкість низькорівневим протоколам (яка потрібна через наявність різних способів з'єднання) і високорівневим (потрібну через наявність різних сценаріїв роботи).

Сокет - це кінцева точка двостороннього з'єднання між двома системами, що працюють в мережі. Коли два або більше процеси взаємодіють через мережу,

вони взаємодіють використовуючи сокети. Порт є беззнаковое ціле число, яке унікально ідентифікує процес, шо виконується в мережі.

Підтримуються обидва, синхронне і асинхронне з'єднання при використанні сокетів. Також відомі, як блокований і не блокований режим роботи. Існують досить тонкі відмінності між двома цими методами. Коли працюємо в синхронному режимі, виклик методу блокує сам себе до тих пір поки операція не буде повністю закінчена. В іншому режимі роботи, тобто асинхронному режимі, метод повертається ще до того як час циклу обробки закінчилося. Під часом циклу обробки мається на увазі загальний час витрачений потоком до повного його завершення.

У синхронному режимі взаємодії, серверний додаток слухає конкретний порт на предмет отримання даних від клієнта. Поступаючи так, серверний додаток блокується (для інших клієнтів) до тих пір поки не отримає дані від клієнтського додатку. З іншого боку, під час роботи в асинхронному режимі, сервер може обробляти безліч клієнтських запитів одночасно. Зверніть увагу, що асинхронні команди використовують сокети зазвичай застосовуються для завдань які вимагають великих витрат часу. Типові прикладами таких задач є відкриття великих файлів, відправка запитів базі даних з великим обсягом даних, підключення до віддаленого комп'ютера, віддалений доступ до ресурсів вимагають великих тимчасових витрат.

Так само врахуйте, що асинхронні виклики насправді працюють в роздільних потоках.

Зазвичай програма має два види потоків, програмний потік і робочий потік. Програмний потік - це основний потік програми; робочий потік - це потік, що працює у фоновому режимі, для забезпечення асинхронних операцій.

Алгоритм роботи взаємодії розділяється на алгоритм сервера і клієнта.

Алгоритм клієнта:

1. **create** - створення сокета
2. **connect** - під’єднання сокета до віддаленої TCP/IP-адреси сервера
3. **read/write** - читання і запис даних з/на сервер.
4. **close** - розірвання з’єднання

Алгоритм сервера:

1. **create** - створення сокета
2. **bind** - прив'язування сокета до локальної IP - адреси.
3. **listen —** переведення сокета в режим прослуховування

4 **accept** — підтвердження запиту клієнта, повернення сокета клієнта для передачі даних
5 **read/write** — обмін даними
6 **close** — розривання з'єднання

На сьогоднішній день якщо не всі, більшість Android-пристроїв мають доступ до мережі інтернет. А велика кількість мобільних додатків так чи інакше взаємодіють із середовищем інтернету: завантажують файли, авторизуються та отримують інформацію із зовнішніх веб-сервісів тощо. Розглянемо, як ми можемо використовувати у своєму додатку доступ до Інтернету.

Серед стандартних елементів нам доступний віджет WebView, який може завантажувати контент із певної URL-адреси. Але цим можливості роботи з мережею Android не обмежуються. Для отримання даних із певного інтернет-ресурсу ми можемо використовувати класи **HttpUrlConnection** (для протоколу HTTP) та **HttpsUrlConnection** (для протоколу HTTPS) із стандартної бібліотеки Java.

Отже, створимо новий проект із порожньою MainActivity. Насамперед для роботи з мережею нам потрібно встановити у файлі маніфесту **AndroidManifest.xml** відповідний дозвіл:

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

У файлі **activity_main.xml** , який представляє розмітку MainActivity, визначимо наступний код:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent" >
7
8      <Button
9          android:id="@+id/downloadBtn"
10         android:layout_width="0dp"
11         android:layout_height="wrap_content"
12         android:text="Загрузка"
13         app:layout_constraintLeft_toLeftOf="parent"
14         app:layout_constraintRight_toRightOf="parent"
15         app:layout_constraintTop_toTopOf="parent" />
16     <WebView
17         android:id="@+id/webView"
```

```
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/downloadBtn"
        app:layout_constraintBottom_toTopOf="@id/scrollView" />
    <ScrollView
        android:id="@+id/scrollView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/webView"
        app:layout_constraintBottom_toBottomOf="parent">

        <TextView android:id="@+id/content"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
    </ScrollView>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Тут визначено кнопку для завантаження даних, а самі дані для прикладу завантажуються одночасно у вигляді рядка в текстове поле і елемент WebView. Так як даних може бути дуже багато, текстове поле поміщене в елемент ScrollView.

Оскільки завантаження даних може зайняти деякий час, то звернення до інтернет-ресурсу визначимо в окремому потоці і для цього змінимо код **MainActivity** таким чином:

```
package com.example.httpapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.webkit.WebView;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
```

```
import java.net.URL;
import javax.net.ssl.HttpsURLConnection;

public class MainActivity extends AppCompatActivity {

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

        TextView contentView = findViewById(R.id.content);
        WebView webView = findViewById(R.id.webView);
        webView.getSettings().setJavaScriptEnabled(true);
        Button btnFetch = findViewById(R.id.downloadBtn);
        btnFetch.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                contentView.setText("Загрузка...");
                new Thread(new Runnable() {
                    public void run() {
                        try{
                            String content =
getContent("https://stackoverflow.com/");
                            webView.post(new Runnable() {
                                public void run() {
                                    webView.loadDataWithBaseURL("https://stackover
flow.com/",content, "text/html", "UTF-8", "https://stackoverflow.com/");
                                    Toast.makeText(getApplicationContext(),
"Данные загружены", Toast.LENGTH_SHORT).show();
                                }
                            });
                            contentView.post(new Runnable() {
                                public void run() {
                                    contentView.setText(content);
                                }
                            });
                        }
                        catch (IOException ex){
                            contentView.post(new Runnable() {
                                public void run() {
                                    contentView.setText("Ошибка: " +
ex.getMessage());
                                    Toast.makeText(getApplicationContext(),
"Ошибка", Toast.LENGTH_SHORT).show();
                                }
```

```
                        });
                    }
                }
            }).start();
        }
    });
}
private String getContent(String path) throws IOException {
    BufferedReader reader=null;
    InputStream stream = null;
    HttpsURLConnection connection = null;
    try {
        URL url=new URL(path);
        connection =(HttpsURLConnection)url.openConnection();
        connection.setRequestMethod("GET");
        connection.setReadTimeout(10000);
        connection.connect();
        stream = connection.getInputStream();
        reader= new BufferedReader(new InputStreamReader(stream));
        StringBuilder buf=new StringBuilder();
        String line;
        while ((line=reader.readLine()) != null) {
            buf.append(line).append("\n");
        }
        return(buf.toString());
    }
    finally {
        if (reader != null) {
            reader.close();
        }
        if (stream != null) {
            stream.close();
        }
        if (connection != null) {
            connection.disconnect();
        }
    }
}
}
```

Безпосередньо для самого завантаження визначено метод `getContent()`, який завантажуватиме веб-сторінку за допомогою класу **HttpsURLConnection** і повертатиме код завантаженої сторінки у вигляді рядка.

Спочатку створюється елемент `HttpsURLConnection`:

```
1   URL url=new URL(path);
```

```
connection =(HttpsURLConnection)url.openConnection();
connection.setRequestMethod("GET"); // установка метода получения данных -GET
connection.setReadTimeout(10000); // установка таймаута перед выполнением - 10 000
миллисекунд
connection.connect(); // подключаемся к ресурсу
```

Після підключення відбувається зчитування із вхідного потоку:

```
stream = connection.getInputStream();
reader= new BufferedReader(new InputStreamReader(stream));
```

Використовуючи вхідний потік, ми можемо рахувати його в рядок.

Цей метод `getContent()` буде викликатися в обробнику натискання кнопки:

```
Button btnFetch = (Button)findViewById(R.id.downloadBtn);
btnFetch.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        contentView.setText("Загрузка...");
        new Thread(new Runnable() {
            public void run() {
                try{
                    String content = getContent("https://stackoverflow.com/");
```

Оскільки завантаження може зайняти тривалий час, то метод `getContent()` в окремому потоці за допомогою об'єктів Thread та Runnable. Наприклад, у цьому випадку звернення йде до ресурсу "https://stackoverflow.com/". Запустимо програму та натиснемо на кнопку. І за наявності інтернету програма завантажить гравну сторінку з "https://stackoverflow.com/" і відобразить її у WebView та TextView.

Звичайно, цей спосіб навряд чи підходить для перегляду інтернет-сторінок, проте таким чином ми можемо отримувати будь-які дані (не інтернет-сторінки) від різних веб-сервісів, наприклад, у форматі xml або json (наприклад, різні курси валют, показники погоди), використовуючи спеціальні api, а потім після обробки показувати їх користувачев

10.2 Використання запитів GET, POST.

Давайте почнемо зі створення дуже простого серверного додатка на стороні Android. Відкрийте Android Studio та створіть новий проект із порожньою активністю. Потім додайте новий клас Java для сервера.

Android використовує звичайні Java ServerSockets , тому ми можемо дуже легко створити базовий сервер. Єдине, що варто згадати, це те, що ми маємо запустити сокет в окремому потоці, тому що Android не дозволяє будь-який мережевий код у головному потоці, і він викидає виняток, якщо ви спробуєте.

У основній діяльності програми нам потрібно лише створити новий об’єкт Server.

```
package
com.pingpoli.servertest;
                        import androidx.appcompat.app.AppCompatActivity;
                        import android.os.Bundle;
                        public class MainActivity extends AppCompatActivity
                        {
                            private Server server;
                            @Override
                            protected void onCreate( Bundle savedInstanceState )
                            {
                                super.onCreate( savedInstanceState );
                                setContentView( R.layout.activity_main );
                                this.server = new Server();
                            }
                        }
```

Нарешті, нам потрібно надати нашому додатку доступ до Інтернету, що можна зробити, додавши ці рядки до файлу AndroidManifest.xml.

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

Простий клієнт

Нам також потрібен простий клієнт TCP, щоб перевірити, чи можемо ми підключитися до нашого сервера. Я використовую власну мережеву бібліотеку C++, але для цього підручника давайте створимо простий клієнт на Java:

```
package
test;
       import java.io.BufferedInputStream;
       import java.io.BufferedOutputStream;
       import java.io.DataInputStream;
       import java.io.DataOutputStream;
       import java.io.IOException;
       import java.net.Socket;
       public class TestClient
       {
           private Socket socket;
           private DataInputStream dataInputStream;
           private DataOutputStream dataOutputStream;

           public static void main( String[] args )
```

```
    {
        TestClient client = new TestClient();
    }
    public TestClient()
    {
        try
        {
            // create new socket and connect to the server
            this.socket = new Socket( "127.0.0.1" , 12345 );
        }
        catch( IOException e )
        {
            System.out.println( "failed to create socket" );
            e.printStackTrace();
        }

        System.out.println( "connected" );
        try
        {
            this.dataInputStream = new DataInputStream( new BufferedInputStream(
this.socket.getInputStream() ) );
            this.dataOutputStream = new DataOutputStream( new BufferedOutputStream(
this.socket.getOutputStream() ) );
        }
        catch ( IOException e )
        {
            System.out.println( "failed to create streams" );
            e.printStackTrace();
        }

        while ( true )
        {
            try
            {
                int test = this.dataInputStream.readInt();
                System.out.println( "int received: "+test );
                if ( test == 42 ) break;
            }
            catch ( IOException e )
            {
                System.out.println( "failed to read data" );
                e.printStackTrace();
                break;
            }
        }
    }
}
```

Однак якщо ви спробуєте підключитися до сервера, який працює на емуляторі, або підключеного телефону, це поки що не спрацює. Ви, мабуть, помітили в цьому рядку `this.socket = new Socket( "127.0.0.1" , 12345 );`, що ми підключаємося до localhost, що трохи дивно, враховуючи, що ми хочемо підключитися до іншого пристрою. Але на це є причина:

ADB або **Android Debug Bridge** є офіційним інструментом для розробки Android. Він використовується для зв'язку з емуляторами та пристроями, що є саме тим, що ми хочемо зробити. По-перше, нам потрібно завантажити інструменти платформи ADB/SDK і розпакувати їх десь. Тоді ми можемо відкрити термінал у витягнутій папці (порада Windows: клацніть у навігаційному рядку провідника та введіть «*cmd*», натисніть ENTER, і відкриється інтерфейс командного рядка в поточній папці). Там нам потрібно ввести таку команду:

```
adb forward tcp:12345 tcp:12345
```

Це перенаправлятиме трафік TCP з локального хосту вашого комп'ютера на порт 12345 на ваш пристрій на цьому ж порту. Після цього клієнт, запущений на вашому комп'ютері, зможе підключитися до сервера, запущеного на вашому пристрої Android.

Важлива примітка

Кожного разу, коли ви перемикаєтеся між емулятором і смартфоном, підключеним через USB, потрібно знову вводити команду adb.

Інша альтернатива — викликати команду безпосередньо з клієнтської програми перед підключенням:

```
C++: system( "PATH_TO/adb.exe forward tcp:12345 tcp:12345" );
```

```
Java: Runtime.getRuntime().exec( "PATH_TO/adb.exe forward tcp:12345
tcp:12345" );
```

Підключити емулятор Android або підключений смартфон до комп'ютера за допомогою роз'ємів TCP не надто складно, якщо ви знаєте, як це зробити. Мені знадобився деякий час, щоб зрозуміти це, але, сподіваюся, цей посібник допоможе вам змусити це працювати набагато швидше.

10.3 Використання сторонніх бібліотек Volley, Retrofit.

Використовувати Retrofit :

- коли потрібно стандартна реалізація REST API для парсингу JSON
- реалізація без кастомних запитів, запитів із пріоритетом, кешування, повторів etc.

Використовувати Volley :

- незвичайні вимоги до запитів
- можливо, буде необхідність розширення функціональності вашого REST API надалі

NB: Якщо потрібно завантажувати великі файли або потоки, то не слід використовувати ні те, ні інше. Використовуйте Download Manager

Retrofit

Властивості:

- made in Square
- не підтримує обробку зображень (добре використовувати для цього Picasso (див. посилання Picasso vs Glide нижче))
- використовує пул потоків, паралельні запити та бібліотеку OkHttp
- пріоритезація, скасування, повторні запити - вручну (у Retrofit 2 трохи зручніше)
- повна підтримка POST-запитів та multipart завантажень файлів
- Retofit 2 працює з RxJava простіше (з Observable types)
- HTTP-запити описуються через анотації, синхронні та асинхронні виклики REST-методів, дані можуть передаватися як JSON, XML, Protobuf. З цією бібліотекою на пару вміє працювати RoboSpice .

Отже, Retrofit – бібліотека REST-клієнт для роботи в Android.

Підключення:

```
compile 'com.squareup.retrofit2:retrofit:2.3.0'
```

Volley

Властивості:

- made in Google
- підтримує мінімальний функціонал обробки зображень (для більш повного функціоналу, можна використовувати Glide , + про Glide див. тут , + відмінності Glide від Picasso тут чи тут чи наочно тут)
- використовує пул потоків, паралельні запити та бібліотеку OkHttp
- Гнучкий механізм кешування (який добре підходить для Glide)

- підтримує виставлення пріоритетів, відміну запитів, повторні запити - використовуючи для цього трохи коду
- підтримує POST-запити (але необхідно конвертувати Java-об'єкти в JSON-об'єкти використовуючи, наприклад, бібліотеку Gson), також підтримує multipart завантаження, але, надаючи більшу гнучкість, api не такий відполірований, як у Retrofit'a