Часть 1. Работа с JSON

JSON (JavaScript Object Notation) — это легкий формат обмена данными, основанный на подмножестве языка JavaScript. JSON используется для передачи структурированных данных между клиентом и сервером. Он часто применяется в веб-разработке, мобильных приложениях и других системах, где требуется передача данных.

JSON состоит из пар ключ-значение, где ключи обычно строковые, а значения могут быть строками, числами, логическими значениями, массивами, объектами или null. Его легко читать и понимать как человеку, так и компьютеру, что делает его идеальным для обмена данными.

В Android разработке JSON часто используется для передачи данных между клиентом и сервером или для сохранения информации локально.

Для работы с форматом json нет встроенных средств, но есть куча библиотек и пакетов, которые можно использовать для данной цели. Одним из наиболее популярных из них является пакет **com.google.code.gson**.

Для облегчения работы с JSON объектами с использованием данного пакета необходимо дополнительно добавить библиотеку Gson в build.gradle.

```
    implementation 'com.google.code.gson:gson:2.8.8'
```

Теперь рассмотрим несколько примеров работы с JSON и классом User, состоящим из полей: имя, возраст, почта.

Создание JSON строки.

```
    public void parseJsonUsingGson() {
        String jsonStr = "{\"name\":\"John\", \"age\":30,
\"email\":\"john@example.com\"}";
        Gson gson = new Gson();

        User user = gson.fromJson(jsonStr, User.class);
```

```
        System.out.println("Name: " + user.name);
        System.out.println("Age: " + user.age);
        System.out.println("Email: " + user.email);
    }
```

Для работы с json создается объект Gson. Для десериализации выполняется метод **fromJson(),** в который передается объект класса User с сериализованными данными и тип, к которому надо десериализиовать данные.

Чтение одной JSON строки.

```
    public String createJsonUsingGson() {
        User user = new User();
        user.name = "Alice";
        user.age = 25;
        user.email = "alice@example.com";

        Gson gson = new Gson();
        return gson.toJson(user);
    }
```

Для сериализации данных в формат json у этого объекта вызывается метод **toJson(),** в который передаются сериализуемые данные. На выходе метод toJson() возвращает строку, которая затем сохраняется в текстовый файл.

Чтение JSON массива.

```
    public void parseJsonArrayUsingGson() {
        String jsonArrayStr = "[{\"name\":\"John\", \"age\":30,
\"email\":\"john@example.com\"}," +
                              "{\"name\":\"Alice\", \"age\":25,
\"email\":\"alice@example.com\"}]";
        Gson gson = new Gson();
        Type userListType = new TypeToken<List<User>>(){}.getType();

        List<User> users = gson.fromJson(jsonArrayStr, userListType);

        for (User user : users) {
            System.out.println("Name: " + user.name);
            System.out.println("Age: " + user.age);
```

```
            System.out.println("Email: " + user.email);
        }
    }
```

Часть 2. Провайдеры контента

Провайдеры контента (Content Providers) в Android представляют собой компоненты, которые позволяют приложениям делиться данными между собой. Провайдеры контента инкапсулируют данные и предоставляют механизмы для определения политик доступа к этим данным. Это особенно полезно в среде, где множество различных приложений пытаются работать с одним и тем же набором данных или когда данные должны быть предоставлены из одного приложения другим.

Зачем нужны провайдеры контента?

Провайдеры контента в Android предоставляют следующие преимущества:

1. Безопасный доступ к данным: Провайдеры контента могут ограничивать доступ к данным приложения, предоставляя только необходимые данные и скрывая внутреннюю структуру базы данных.
2. Стандартизация доступа к данным: Они предлагают унифицированный интерфейс для работы с данными, что упрощает обмен данными между приложениями.
3. Управление данными: Провайдеры контента упрощают управление данными, так как обработка запросов, вставка, удаление и обновление данных производятся через стандартизированные API.
4. Интеграция с Android системой: Провайдеры могут легко интегрироваться с другими Android компонентами, такими как загрузчики (Loaders) и система поиска.

Рассмотрим следующий пример с провайдером контента.

Имеется приложение по работе с коллекцией книг, хранящейся в базе данных SQLite и вы хотите предоставить другим приложениям доступ к информации о книгах пользователя.

Для начала необходимо определить URI (Uniform Resource Identifier) — уникальный идентификатор, используемый для идентификации данных в провайдере

контента. В Android URI для провайдера контента обычно включает в себя authority, который является символьным идентификатором провайдера.

```
    public static final Uri CONTENT_URI = Uri.parse("content://com.example.app.provider/books");
```

Далее необходимо реализовать класс провайдера и добавить в него базу данных.

Для создания собственного контент-провайдера нужно унаследоваться от абстрактного класса **ContentProvider.**

```
    public class BookProvider extends ContentProvider {
        private SQLiteDatabase db;

        @Override
        public boolean onCreate() {
            DBHelper dbHelper = new DBHelper(getContext());
            db = dbHelper.getWritableDatabase();
            return (db != null);
        }

        @Override
        public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {
            return db.query("books", projection, selection, selectionArgs, null, null, sortOrder);
        }

        @Override
        public Uri insert(Uri uri, ContentValues values) {
            long rowID = db.insert("books", "", values);
            Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
            getContext().getContentResolver().notifyChange(_uri, null);
            return _uri;
        }

        @Override
        public int delete(Uri uri, String selection, String[] selectionArgs) {
```

```
            int count = db.delete("books", selection, selectionArgs);
            getContext().getContentResolver().notifyChange(uri, null);
            return count;
        }

        @Override
        public int update(Uri uri, ContentValues values, String
selection, String[] selectionArgs) {
            int count = db.update("books", values, selection,
selectionArgs);
            getContext().getContentResolver().notifyChange(uri, null);
            return count;
        }
        @Override
        public String getType(Uri uri) {
            return "vnd.android.cursor.dir/vnd.example.books";
        }
    }
```

Последним шагом станет добавление провайдера в Манифесте и заполнение базы данных информацией о книгах.

```
    <provider
        android:name=".BookProvider"
        android:authorities="com.example.app.provider"
        android:exported="true"
        android:permission="android.permission.READ_PROVIDER" />
```

Далее, в Манифесте другого приложения даем разрешение на чтение данных из провайдера.

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

Затем обрабатываем URI и запрашиваем данные из другого приложения.

```
    Uri                          contentUri                          =
Uri.parse("content://com.example.app.provider/books");
    ContentResolver resolver = getContentResolver();
    Cursor cursor = resolver.query(contentUri, new String[] {"_id",
"title", "author"}, null, null, "title ASC");
    if (cursor != null) {
```

```
        try {
            int idColumn = cursor.getColumnIndex("_id");
            int titleColumn = cursor.getColumnIndex("title");
            int authorColumn = cursor.getColumnIndex("author");

            while (cursor.moveToNext()) {
                int id = cursor.getInt(idColumn);
                String title = cursor.getString(titleColumn);
                String author = cursor.getString(authorColumn);

                System.out.println("Book ID: " + id);
                System.out.println("Book Title: " + title);
                System.out.println("Author: " + author);
            }
        } finally {
            cursor.close(); // Важно закрыть курсор после использования
        }
    }
```

Таким образом, другое приложение получит данные о книгах пользователя из другого приложения.

Задание

1. Реализовать передачу данных в другое приложение через провайдер контента

2. Выполнить преобразование данных в JSON формат и сохранение в отдельный файл. Выполнить преобразование данных из файла JSON в различные поля.

Источники

1) https://metanit.com/java/android/13.3.php?ysclid=lvfld4qy1s640071326
2) https://abhiandroid.com/programming/json#gsc.tab=0
3) https://devcolibri.com/unit/урок-14-знакомство-с-форматом-json-парсинг-jsono/?ysclid=lvfm5oxjjb195070701
4) https://metanit.com/java/android/20.1.php?ysclid=lvflot2oin576330620
5) https://developer.alexanderklimov.ru/android/theory/contentprovider.php?ysclid=lvfm68x3gg106849498
6) https://developer.android.com/guide/topics/providers/content-providers
7) https://www.geeksforgeeks.org/content-providers-in-android-with-example/