

Rajdip Sanyal (06/09/2024)
sanyalrajdip864@gmail.com

Android Framework with Java

Broadcast

Android development, a broadcast is a way to send or receive messages across different components of an application or between different applications. It allows for decoupled communication, meaning that the sender of a broadcast does not need to know about the receivers and vice versa.

Key Concepts of Broadcasts in Android:

Broadcasts: They are essentially messages or announcements sent by an application or system to inform other components about some event or change. For instance, the system can broadcast a message when the battery is low or when the device is connected to a Wi-Fi network.

Broadcast Receiver: This is a component that listens for and responds to broadcast messages. You can define a BroadcastReceiver in your app to handle specific broadcasts that are relevant to your application.

Sending a Broadcast: You can send a broadcast using the `sendBroadcast()` method for general broadcasts, `sendOrderedBroadcast()` for ordered broadcasts (where the receivers are executed in a specific order), and `sendStickyBroadcast()` for sticky broadcasts (where the broadcast remains available to future broadcasts even after it has been sent).

Registering a Broadcast Receiver: Broadcast receivers can be registered either statically (in the `AndroidManifest.xml`) or dynamically (at runtime using `registerReceiver()`).

```

private myBroadcastReceiver receiver = new myBroadcastReceiver(); 1 usage
protected void onStart()
{
    super.onStart();
    IntentFilter filter=new IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED);
    registerReceiver(receiver,filter);
    bgService service=new bgService();

    try {
        service.onHandleIntent(Intent.getIntent(Intent.ACTION_ANSWER));
    }
    catch(Exception e)
    {

    }
}

```

Intents

In android development, an Intent is a messaging object used to request an action from another app component. Intents are fundamental for interacting with other components within an app or even between different apps. They help to start activities, services, and broadcast receivers.

Here's a breakdown of how Intents are used and their key components:

Types of Intents:

Explicit Intents: Specify the target component by its class name. This is often used when you know exactly which component you want to start. For example, starting a new activity within the same app.

Implicit Intents: Do not specify the target component explicitly. Instead, they declare a general action to be performed and the system figures out which

component can handle that action. For instance, opening a webpage or picking a photo.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Implicit

    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse(uriString: "https://github.com/aryan1403"));

    //Explicit

    Intent intent1=new Intent(getApplicationContext(),SecondActivity.class);

    setContentView(R.layout.activity_main);
    startActivity(intent1);
}
```

ArrayAdapter

ArrayAdapter is a class used in Android development to provide a way to manage and display a list of data in a ListView or Spinner (among other UI elements). It acts as a bridge between the data and the views that display the data.

Here's a breakdown of how it works:

Key Concepts: Data Source: ArrayAdapter requires a data source, typically an array or a list of objects. For example, you might use an array of strings or a list of custom objects.

Layout: You need to provide a layout resource that defines how each item in the list should look. This layout resource is used to create the view for each item in the list.

Context: The adapter needs a Context, which is usually the current activity or application context. This context is used to access resources and layout inflation.

```
public void onClick(View v) {
    if (currentIndex < fetchedItems.size()) {
        // Add the next item to the textList and update the ListView
        textList.add(fetchedItems.get(currentIndex));
        currentIndex++; // Move to the next item

        // Update ListView with the new item
        ArrayAdapter<String> adapter = new ArrayAdapter<>(context: MainActivity.this,
        listView.setAdapter(adapter);
    } else {
        Toast.makeText(context: MainActivity.this, text: "No more items to load", Toas
    }
}
});
```

XML parsing

XML (Extensible Markup Language) parsing refers to the process of reading and interpreting XML documents to extract meaningful information. XML is a flexible text format used to store and transport structured data. Parsing XML involves interpreting the XML document according to its structure, which is defined by a set of rules and a schema(if applicable).

Applications:

1. Data Integration: XML is commonly used in data interchange between different systems and applications.
2. Configuration Files: XML is often used for configuration files in various software applications.
3. Web Services: XML is used in web services protocols such as SOAP.

```

1 package com.example.xmlparsing;
2
3 > import ...
20
21 public class MainActivity extends AppCompatActivity {
22     private ListView lv; 2 usages
23     @Override
24     protected void onCreate(Bundle savedInstanceState) {
25         super.onCreate(savedInstanceState);
26         setContentView(R.layout.activity_main);
27         ArrayList<user> userList = new ArrayList<>();
28         user u = null;
29         lv = findViewById(R.id.userList);
30         try {
31             InputStream is = getAssets().open(fileName: "userdetails.xml");
32             XmlPullParser xmlPullParser = XmlPullParserFactory.newInstance().newPullParser(
33             xmlPullParser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, b: false);
34             xmlPullParser.setInput(is, s: null);
35             String tag = "";
36             String text = "";
37             int event = xmlPullParser.getEventType();
38             while (event != XmlPullParser.END_DOCUMENT)
39             {
40                 tag = xmlPullParser.getName();
41                 switch (event)
42                 {

```

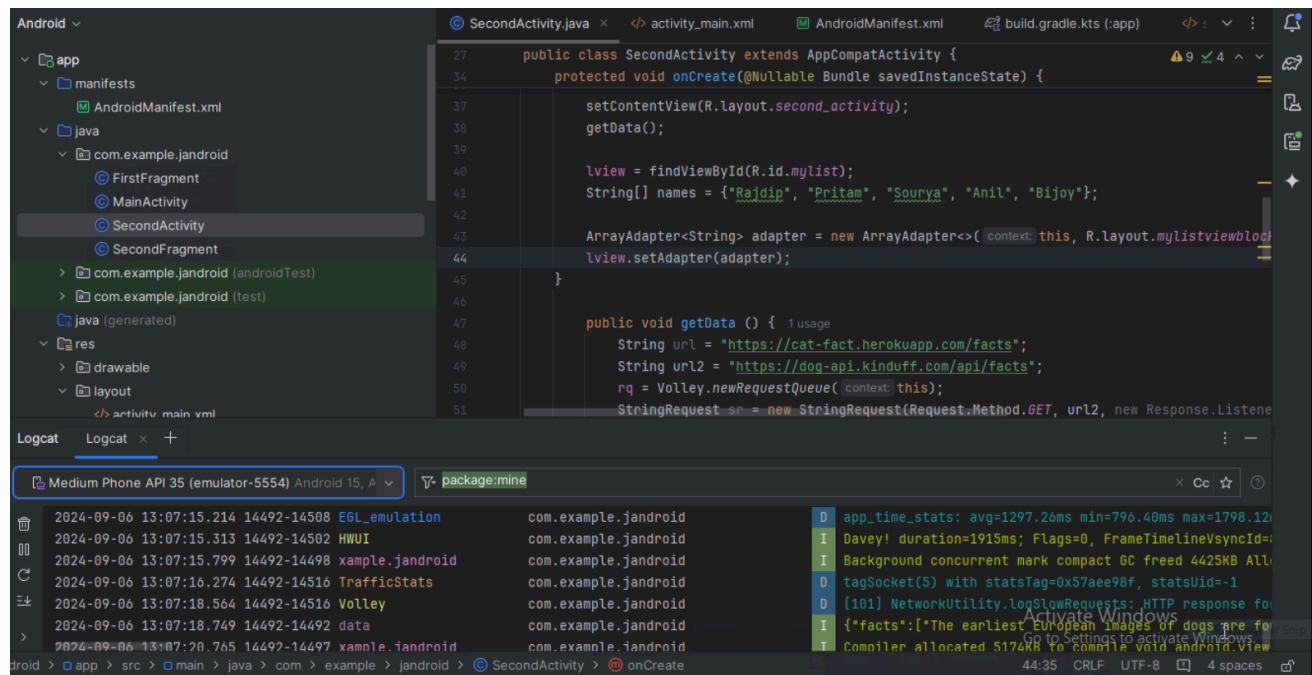
Volley

Volley is a library in Android development for managing network requests and responses. It simplifies the process of sending and receiving network requests, handling image loading, and caching responses, making it easier to build an efficient network communication in android apps.

It simplifies network operations and makes it easier to manage requests, responses, and caching. However, for more complex needs or if you require advanced features, you might also want to look into other libraries such as Retrofit or OkHttp.

Fetch API in Android

In Android development, the Fetch API is not a part of the standard Android SDK. The Fetch API is a web standard used in web development to make network requests. It provides a modern, promise-based approach to handling HTTP requests and responses in JavaScript.



The screenshot displays the Android Studio IDE. The top toolbar shows icons for running, debugging, and other development tools. The left sidebar contains a project tree with the following structure:

- app
 - manifests
 - AndroidManifest.xml
 - java
 - com.example.jandroid
 - FirstFragment
 - MainActivity
 - SecondActivity (selected)
 - SecondFragment
 - com.example.jandroid (androidTest)
 - com.example.jandroid (test)
 - java (generated)
 - res
 - drawable
 - layout

The main editor shows the `SecondActivity.java` file with the following code:

```
27 public class SecondActivity extends AppCompatActivity {
34     protected void onCreate(@Nullable Bundle savedInstanceState) {
37         setContentView(R.layout.second_activity);
38         getData();
39
40         lvview = findViewById(R.id.mylist);
41         String[] names = {"Rajdip", "Pritam", "Sourya", "Anil", "Bijoy"};
42
43         ArrayAdapter<String> adapter = new ArrayAdapter<>(context, this, R.layout.mylistviewblock, names);
44         lvview.setAdapter(adapter);
45     }
46
47     public void getData () { 1 usage
48         String url = "https://cat-fact.herokuapp.com/facts";
49         String url2 = "https://dog-api.kinduff.com/api/facts";
50         rq = Volley.newRequestQueue(context, this);
51         StringRequest sr = new StringRequest(Request.Method.GET, url2, new Response.Listener<String>() {
52             @Override
53             public void onResponse(String response) {
54                 // TODO: Handle the response
55             }
56         });
57         sr.setRetryPolicy(new DefaultRetryPolicy());
58         sr.addHeader("Accept", "application/json");
59         sr.addHeader("Accept-Encoding", "gzip");
60         sr.addHeader("User-Agent", "com.example.jandroid");
61         rq.add(sr);
62         rq.start();
63     }
64 }
```

The bottom Logcat window shows the following log entries:

Time	Level	Category	Message
2024-09-06 13:07:15.214	INFO	EGL_emulation	com.example.jandroid
2024-09-06 13:07:15.313	INFO	HWUI	com.example.jandroid
2024-09-06 13:07:15.799	INFO	xample.jandroid	com.example.jandroid
2024-09-06 13:07:16.274	INFO	TrafficStats	com.example.jandroid
2024-09-06 13:07:18.564	INFO	Volley	com.example.jandroid
2024-09-06 13:07:18.749	INFO	data	com.example.jandroid
2024-09-06 13:07:20.765	INFO	xample.jandroid	com.example.jandroid

The Logcat window also shows a warning message: "app_time_stats: avg=1297.26ms min=796.40ms max=1798.12ms".