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.

```java
package com.example.xmlparsing;

import ...

public class MainActivity extends AppCompatActivity {
        private ListView lv;  2 usages
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            ArrayList<user> userList = new ArrayList<>();
            user u = null;
            lv = findViewById(R.id.userList);
            try {
                InputStream is = getAssets().open( fileName: "userdetails.xml");
                XmlPullParser xmlPullParser = XmlPullParserFactory.newInstance().newPullParser(
                xmlPullParser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES,  b: false);
                xmlPullParser.setInput(is,  s: null);
                String tag = "";
                String text = "";
                int event = xmlPullParser.getEventType();
                while (event != XmlPullParser.END_DOCUMENT)
                {
                    tag = xmlPullParser.getName();
                    switch (event)
                    {
```

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.