



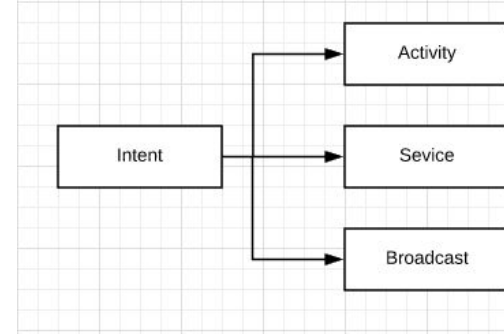
Android İşletim Sisteminde Uygulamalar Arası Haberleşme Mekanizmalarına Giriş



Bu Mekanizmalar Neler?

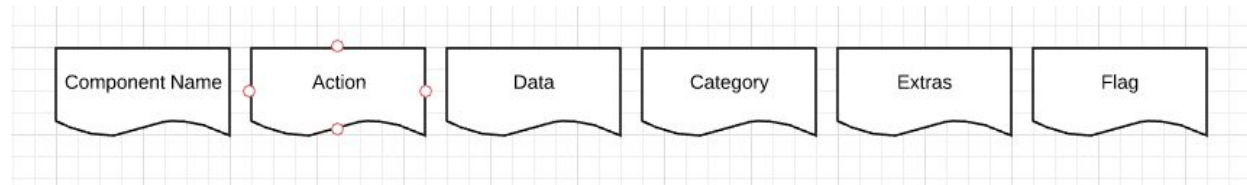
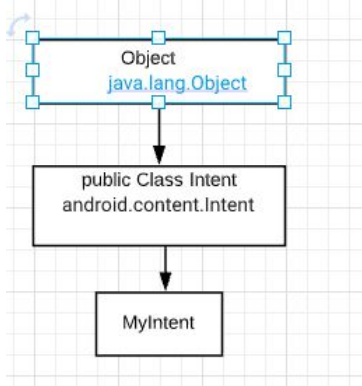
1. Intent
2. Content Provider
3. Broadcast
4. Local Broadcast
5. Bind Servis
6. EventBus
7. Live Data
8. RxJava

1. Intent



Bir uygulama bileşenleri arasında veri transferini ve iletişimini sağlayan haberleşme nesnesi.

- a. Explicit Intent
- b. Implicit Intent



CreateIntent [~/Development/GIT/Example/CreateIntent] - .../app/src/main/java/com/example/createintent/MainActivity.java [app] - Android Studio

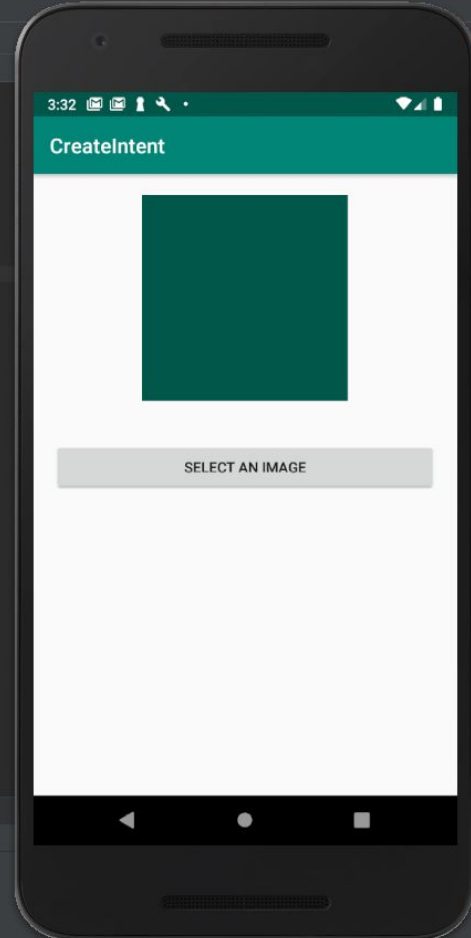
Window Help

createintent MainActivity

activity_main.xml MainActivity.java app

```
13 Button btn;
14 ImageView imageView;
15 Uri imageUri;
16 static final int SELECT_IMAGE=13;
17
18 @Override
19 protected void onCreate(Bundle savedInstanceState) {
20     super.onCreate(savedInstanceState);
21     setContentView(R.layout.activity_main);
22     getControlView();
23     setClickForView();
24 }
25 private void getControlView(){
26     btn=findViewById(R.id.btn);
27     imageView=findViewById(R.id.imageView);
28 }
29 private void setClickForView(){
30     btn.setOnClickListener(new View.OnClickListener() {
31         @Override
32         public void onClick(View view) {
33             selectImage();
34         }
35     });
36 }
37 public void selectImage(){
38     Intent intent =new Intent(Intent.ACTION_GET_CONTENT);
39     intent.setType("image/*");
40     if (intent.resolveActivity(getPackageManager()) != null){
41         startActivityForResult(intent,SELECT_IMAGE);
42     }
43 }
44 @Override
45 protected void onActivityResult(int requestCode, int resultCode, Intent data){
46     if(requestCode==SELECT_IMAGE && resultCode == RESULT_OK){
47         imageUri =data.getData();
48         imageView.setImageURI(imageUri);
49     }
50 }
51
52 }
53 }
```

MainActivity





2. Content Provider

Yapılandırılmış veri yapılarına erişim işlemlerini yönetmek için kullanılan uygulama bileşenleridir. Temel amacı diğer uygulamaların verilere erişebilmesini sağlayan yapılar geliştirmektir. Özellikle, veri paylaşmak, farklı uygulamalar için ortak veri yapıları oluşturmak için kullanılır.

Bir içerik sağlayıcısından verilere erişmek için `ContentResolver` sınıfı kullanılır.

`content://com.mas.createcontentprovider.books.lbook/2`



Bir Content Provider Oluşturma Aşamaları

1. ContentProvider ile extend edilen bir sınıf oluşturuyoruz.
2. Content Provider'e erişmek için URI adresleri tanımlıyoruz.
3. Veri tabanı oluşturuyoruz.
4. Veritabanları üzerinde işlem yapmak için sorgular oluşturuyoruz.
5. Son olarak da oluşturulan ContentProvider'i manifest dosyasına bildiriyoruz.




3. Broadcast

Android sistem tarafından, sistem genelinde yapılan duyuruları alan bir bileşendir. Birçok yayın sistem tarafından yapılır.

Bir broadcast kullanmak için projemize BroadcastReceiver extends etmemiz gerekiyor.

Örnek verirsek, mesaj gelmesi, batarya gücünün az olması gibi durumlarda sistem genelinde bir yayın , yani duyuru yapılır.



Öncelikle Broadcast Receiver'ı sisteme bildirmek zorundayız, bu işlem **dinamik** ve **statik** olmak üzere iki farklı yolla yapılmaktadır.

- AndroidManifest.xml dosyasında application etiketleri arasında receiver'ı belirtmek (Statik)
- registerReceiver() metodu ile kayıt edilmesi. (Dinamik)

Broadcast Kullanımı

```
public class MainActivity extends AppCompatActivity {
    FirstBroadcastReceivers fbr;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        fbr = new FirstBroadcastReceivers();
    }

    @Override
    protected void onResume() {
        super.onResume();
        registerReceiver(fbr, new IntentFilter( "FirstBroadcastReceivers"));
        Log.i( tag: "Receiver", msg: "Receiver is register");
    }

    @Override
    protected void onPause() {
        super.onPause();
        unregisterReceiver(fbr);
        Log.i( tag: "Receiver", msg: "Receiver is unregister.");
    }

    public void sendIntent(View view){
        Intent intent=new Intent( action: "FirstBroadcastReceivers");
        intent.putExtra( name: "value", value: "Broadcast Receiver");
        sendBroadcast(intent);
    }
}
```

adcast() metodu ile mesaj yayınlayacağız.
bu mesajı alıp gösterme.

sendBro
Sonra

```
public class FirstBroadcastReceivers extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String value=intent.getStringExtra( name: "value");
        Toast.makeText(context, text: "Intent: "+value,Toast.LENGTH_SHORT).show();
    }
}
```



4. Local Broadcast

Aynı uygulama içerisinde farklı componentler arasında iletişim sağlayabiliyoruz. Bunlar iki activity arasında, activity servis arasında, activity ve broadcast arasında veya servis ile broadcast arasında olabilir. Bir broadcastı local olarak kullanabilmek için LocalBroadcastManager sınıfını kullanırız.

Kullanılmasının en önemli sebebi daha fazla güvenlik sağlamasıdır. Çünkü local olarak çalışır broadcast intenti uygulama dışına çıkmaz.

LocalBroadcast Kullanımı

```
public class MainActivity extends AppCompatActivity {  
    private LocalBroadcastManager manager;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        manager = LocalBroadcastManager.getInstance(this);  
    }  
  
    public void sendNormalBroadcast(View view) {  
        Intent intent = new Intent( packageContext: this, MyReceiver.class);  
  
        intent.putExtra( name: "a", value: 15);  
        intent.putExtra( name: "b", value: 20);  
  
        sendBroadcast(intent);  
    }  
  
    @Override  
    protected void onResume() {  
        super.onResume();  
        IntentFilter filter = new IntentFilter();  
        filter.addAction("my.result.receiver");  
        manager.registerReceiver(receiver, filter);  
    }  
  
    @Override  
    protected void onPause() {  
        super.onPause();  
        manager.unregisterReceiver(receiver);  
    }  
  
    private BroadcastReceiver receiver = new BroadcastReceiver() {  
        @Override  
        public void onReceive(Context context, Intent intent) {  
            int sum = intent.getIntExtra( name: "Sum", defaultValue: 0);  
            Toast.makeText(context, text: "Sum : " + sum, Toast.LENGTH_LONG).show();  
        }  
    };  
}
```

```
public class MyReceiver extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
  
        int fistValue=intent.getIntExtra( name: "a", defaultValue: 0);  
        int secondValue=intent.getIntExtra( name: "b", defaultValue: 0);  
        int result= fistValue + secondValue;  
  
        LocalBroadcastManager manager=LocalBroadcastManager.getInstance(context);  
  
        Intent returningIntent=new Intent( action: "my.result.receiver");  
        returningIntent.putExtra( name: "Sum", result);  
        manager.sendBroadcast(returningIntent);  
    }  
}
```

- Bir broadcastı local olarak kullanabilmek için LocalBroadcastManager sınıfını kullanırsınız.
- Öncelikle LocalBroadcastManager manager=LocalBroadcastManager.getInstance(context); diyoruz. Bu nesneyi tanımlıyoruz. Daha sonra bu nesneyi kullanarak sendBroadcast methodu sayesinde alıcılarımıza mesaj gönderebiliriz.
- Yine bu nesne sayesinde receiverı dinamik olarak register ve unregister edebiliriz. Bu sayede mesaj göndermemiş ve de alıcılarımızın dışarıdan bir mesaj almamasını sağlamış oluruz.

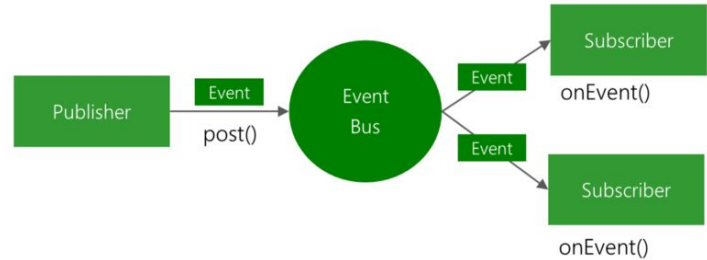


5. Bind Servis

İstemci-Sunucu arayüzü içerisinde bulunan server dediğimiz bir kavrama karşılık gelir. Başka bileşenlerin servise bağlanmasına, istek göndermesine, sonuç almasına imkan tanır. Bind servise bağlı component kalmadığında kendini yok eder. Main thread üzerinde çalışır.

6. Eventbus

Android için activityler, fragmentler, threadler, servisler bileşenler arasındaki iletişimi daha az kod ile çok daha basit hale getirmek için optimize edilmiş [greenrobot](#) bir kütüphanedir.





EventBus Kullanmanın Faydaları

- Bileşenler arası iletişimi kolaylaştırır.
- Eventların göndericilerini ve alıcılarını ayırır.
- Hızlı ve daha iyi performans için optimize edilmiş.



7. Live Data

- Android Jetpack'ın bir parçası.
- LiveData yaşam döngüsü bilincine sahiptir,yani etkinlikler, fragmanlar veya servisler gibi diğer uygulama bileşenlerinin yaşam döngüsüne uyumludur.

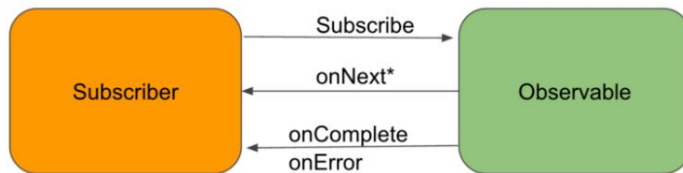


Live Data Kullanmanın Avantajları

- Kullanıcı arayüzü ile veri durumunun eşleşmesini garanti eder
- Memory Leak yoktur.
- Stopped durumdaki Activityler de crashleri önler
- Her zaman güncel veri vardır.

8. RxJava

RxJava, JVM (Java Virtual Machine) için geliştirilmiş bir reactive programlama kütüphanesidir.





Kaynakça

<https://developer.android.com>

<http://greenrobot.org/eventbus/>