drivemode

Keishin Yokomaku / shibuya.apk #11 @ GMO Yours

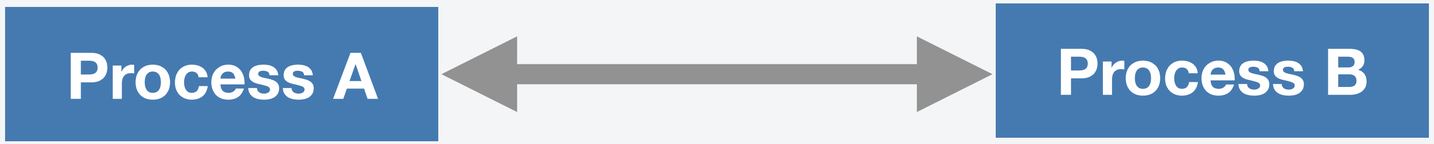
Practical IPC

About Me

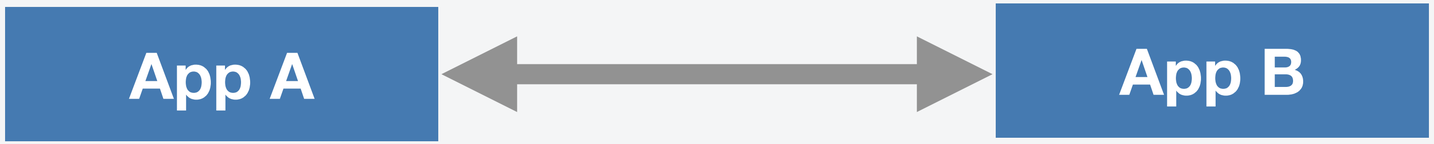
- Keishin Yokomaku
-  Drivemode, Inc. / Principal Engineer
-  KeithYokoma: GitHub / Twitter / Qiita / Tumblr / Stack Overflow
  - Books: Mobile App Dev Guide / Android Academia / Grimoire of Android
  - Fun: Gymnastics / Cycling / Photography / Motorsport
  - Today's Quote: "Power is everything."

Practical IPC

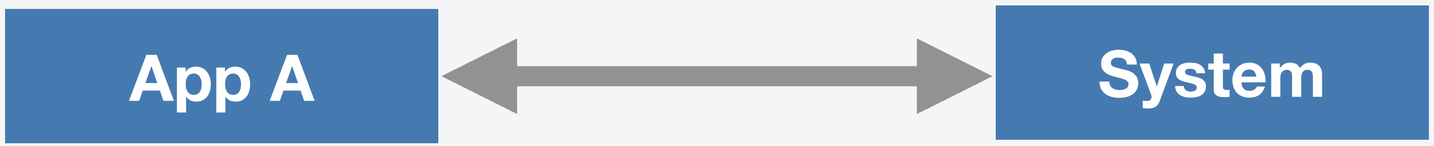
Inter-Process Communication (abbr. IPC)



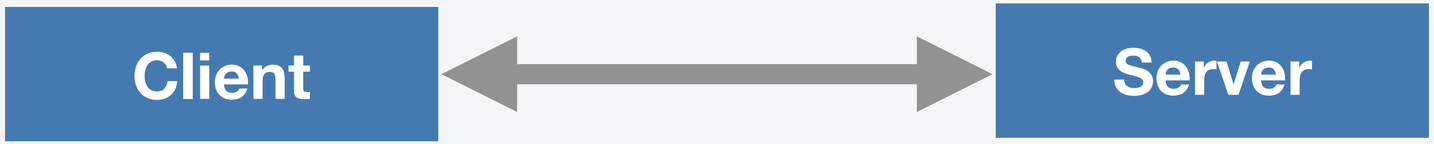
Inter-Process Communication (abbr. IPC)



Inter-Process Communication (abbr. IPC)



Inter-Process Communication (abbr. IPC)



IPC が使われている場所

- PackageManager
  - e.g. PackageManager が動いているプロセスに対する問い合わせ
- Intent
  - e.g. ランチャーからアプリの起動
- android.media.session, android.media.browser
  - Android Auto 用のオーディオアプリ向けフレームワーク

自分で IPC してみる

- 作るもの
  - AIDL
  - Service

AIDL

- Android Interface Definition Language
  - Java に似た文法を持った特別な言語
  - インタフェースを定義するために使う
  - 実際の処理は Java で記述する
  - aidl 専用のディレクトリに入れておく

AIDL の記述例

```
package com.github.keithyokoma.aidl;

interface SampleAidl {
  String getSomething(int value);
}
```

コンパイルエラーになる AIDL の記述例 1

```
package com.github.keithyokoma.aidl;

interface SampleAidl {
  String getSomething(int value);

  String getSomething(int value, boolean flag);
}
```

オーバーロードはできない

コンパイルエラーになる AIDL の記述例 2

```
package com.github.keithyokoma.aidl;

interface SampleAidl {
  String getSomething(int value);

  void doSomething(CustomObject object);
}
```

プリミティブ型・String型・一部のコレクション型以外の型は特別に宣言しない限り使えない

独自クラスを IPC で使うための parcelable キーワード

```
package com.github.keithyokoma.aidl;

interface SampleAidl {
  String getSomething(int value);

  void doSomething(CustomObject object);
}

parcelable CustomObject;
```

CustomObject は Parcelable を実装シたクラスであることを明示

非同期処理のための AIDL の記法

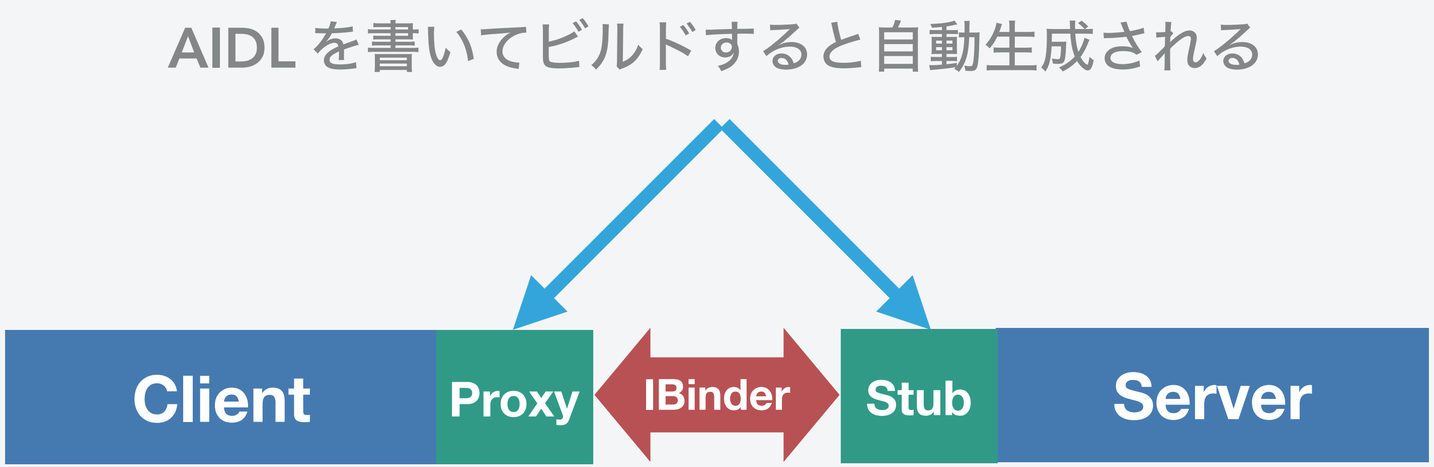
```
package com.github.keithyokoma.aidl;

oneway interface SampleAidl {
  String getSomething(int value);
}
```

oneway キーワードがつくと、メソッド呼び出しが即座に返却され結果は非同期に返ってくる

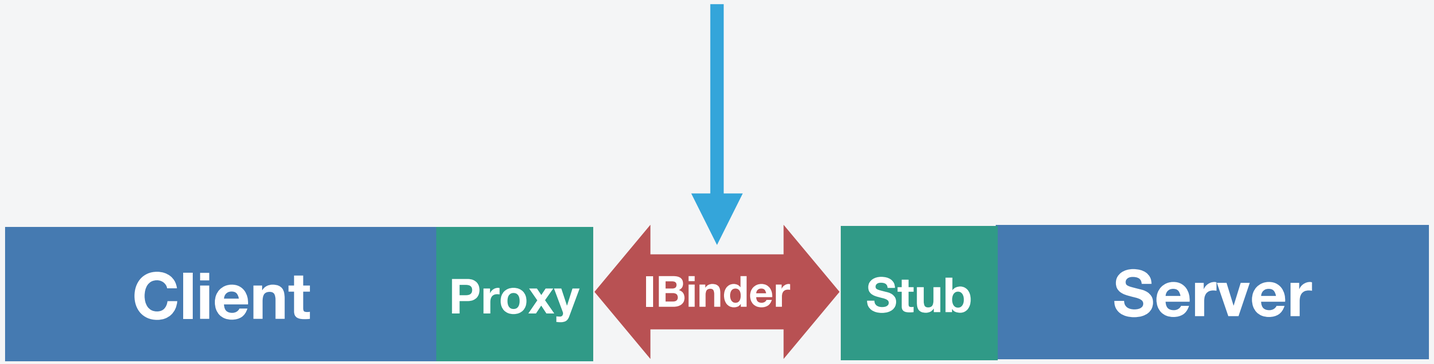
AIDL を使った IPC の図

AIDL を使った IPC の図



AIDL を使った IPC の図

実際にはlibbinder や binder モジュール等の処理が間にある



Service の実装1

```
public class MyService extends Service {
   private final SampleAidl.Stub mStub = new SampleAidl.Stub() {
      @Override
      public String getSomething(int value) {
         return “Hello World!”;
      }
   }

   @Nullable
   @Override
   public IBinder onBind(Intent intent) {
      return mStub;
   }
}
```

Service の実装1

```
public class MyService extends Service {
   private final SampleAidl.Stub mStub = new SampleAidl.Stub() {
      @Override
      public String getSomething(int value) {
         return “Hello World!”;
      }
   }

   @Nullable
   @Override
   public IBinder onBind(Intent intent) {
      return mStub;
   }
}
```

Service の実装1

```
public class MyService extends Service {
   private final SampleAidl.Stub mStub = new SampleAidl.Stub() {
      @Override
      public String getSomething(int value) {
         return “Hello World!”;
      }
   }

   @Nullable
   @Override
   public IBinder onBind(Intent intent) {
      return mStub;
   }
}
```

Service の実装2

```
<manifest>
  <application>
    <service
      android:name=".MyService"
      android:exported="true"
      android:enabled="true">
      <intent-filter>
        <action android:name="com.example.SampleAidl.ACTION_CONNECT" />
      </intent-filter>
    </service>
  </application>
</manifest>
```

Service の実装2

```
<manifest>
  <application>
    <service
      android:name=".MyService"
      android:exported="true"
      android:enabled="true">
      <intent-filter>
        <action android:name="com.example.SampleAidl.ACTION_CONNECT" />
      </intent-filter>
    </service>
  </application>
</manifest>
```

Service を呼び出す側の実装1

```
public class MyActivity extends Activity {
   private final ServiceConnection mConnection = new ServiceConnection() {
      @Override
      public void onServiceConnected(ComponentName name, IBinder service) {
         mService = SampleAidl.Stub.asInterface(service);
      }

      @Override
      public void onServiceDisconnected(ComponentName name) {
         mService = null;
      }
   }
   private SampleAidl mService;
}
```

Service を呼び出す側の実装1

```
public class MyActivity extends Activity {
  private final ServiceConnection mConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
      mService = SampleAidl.Stub.asInterface(service);
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
      mService = null;
    }
  }
  private SampleAidl mService;
}
```

Service を呼び出す側の実装2

```
public class MyActivity extends Activity {
   private final ServiceConnection mConnection = // ……
   private SampleAidl mService;

   public void onClickStartConnection(View view) {
      Intent intent = new Intent("com.sample.SampleAidl.ACTION_CONNECT");
      bindService(intent, mConnection, BIND_AUTO_CREATE);
   }

   @Override
   protected void onDestroy() {
      unbindService(mConnection);
      super.onDestroy();
   }
}
```

Service を呼び出す側の実装2

```
public class MyActivity extends Activity {
   private final ServiceConnection mConnection = // ……
   private SampleAidl mService;

   public void onClickStartConnection(View view) {
      Intent intent = new Intent("com.sample.SampleAidl.ACTION_CONNECT");
      bindService(intent, mConnection, BIND_AUTO_CREATE);
   }

   @Override
   protected void onDestroy() {
      unbindService(mConnection);
      super.onDestroy();
   }
}
```

Service を呼び出す側の実装3

```
public class MyActivity extends Activity {
   private final ServiceConnection mConnection = // ……
   private SampleAidl mService;

   public void onClickGetSomething() {
      try {
         Toast.makeText(getApplicationContext(),
               mService.getSomething(1),
               Toast.LENGTH_SHORT).show();
      } catch (RemoteException e) {
         // something went wrong
      }
   }
}
```

Service を呼び出す側の実装3

```
public class MyActivity extends Activity {
  private final ServiceConnection mConnection = // ……
  private SampleAidl mService;

  public void onClickGetSomething() {
    try {
      Toast.makeText(getApplicationContext(),
          mService.getSomething(1),
          Toast.LENGTH_SHORT).show();
    } catch (RemoteException e) {
      // something went wrong
    }
  }
}
```

なぜ IPC を使うのか

- 他のアプリとバックグラウンドで連携したい
  - e.g. 他のアプリに直接データを送り込んで何らかの処理をしてもらいたい
  - e.g. 他のアプリにデータの取得を代理してもらいたい
  - e.g. 他のアプリが持っているコンテンツを操作したい
  - etc...

なぜ IPC を使うのか

- 良いところ
  - Activity のみならず、常駐型 Service でも使える
  - バインド以外で Intent (broadcast 含む) の仕組みを介さないので動作もはやい
- 辛いところ
  - 管理コストがかさむ(外部公開前提なのでバージョン管理が必要・後述)
  - 母艦アプリが無いと役に立たない

IPC の使い所

- SDK としてアプリの特定の機能・API を公開する
  - e.g. オーディオの再生コントロール機能を AIDL で定義する
  - e.g. AIDL 経由でアプリにアドオン機能を追加する
  - etc...

IPC を使うときの注意点

- IPC での通信相手の検証
- IPC で想定される例外
- AIDL のバージョン管理
- ProGuard の設定

IPC での通信相手のバリデーション

- 許可した通信相手とだけ IPC したい
  - 何もしないと誰とでも IPC できてしまう
    - セキュリティリスク
  - 接続時に誰が通信を試みているか検証する
    - ダメな相手とのコネクションは落とす

IPC での通信相手のバリデーション

- 検証に使う情報
  - パッケージ名
  - 署名のハッシュ
- Android Auto 向け API の MediaBrowserService が良い例
  - xml に許可するパッケージ名と署名のハッシュの一覧を持っておく
  - http://bit.ly/2gdkUJP

考慮しておくべき例外

- TransactionTooLargeException
- BadParcelableException
- DeadObjectException

TransactionTooLargeException

TransactionTooLargeException

- プロセス間通信で使えるメモリ空間の制限を超えた
  - 1プロセスに割り当てられるプロセス間通信用のメモリ空間は1mB
- 回避策
  - 同時にたくさんプロセス間通信しない
  - Bitmap など巨大なデータをプロセス間通信にのせない
  - どうしても Bitmap を使うときは、Bitmap のサイズを指定できるようにする

Don’t

```
interface RemoteService {
  void setBitmap(in Bitmap image);
}
```

Do

```
interface RemoteService {
  void setBitmap(in Uri imageUri);
}
```

BadParcelableException

BadParcelableException

- 互換性のない Parcelable を受け取った
  - クラスローダの違いを考慮に入れよう
  - フィールドの追加・削除は慎重に（Serializableよりはやさしい）
- 回避策
  - クラスローダを置き換える

DeadObjectException

DeadObjectException

- 死んでいるプロセスに通信を試みた
  - 死体蹴り、ダメ絶対
- 回避策
  - IBinder.DeathRecipient や ServiceConnection で死活監視
- SDK としてラップしたクラスを提供する場合、死んだことをコールバックするインタフェースを別途用意するとよい

DeadObjectException

```
private RemoteService mService;
private boolean mIsBound;

private ServiceConnection mConnection = new ServiceConnection() {
   @Override
   public void onServiceConnected(ComponentName component, IBinder service) {
      mService = ((RemoteService.Binder) service).getService();
      mIsBound = true;
   }

   @Override
   public void onServiceDisconnected() {
      mIsBound = false;
   }
}
```

DeadObjectException

```
public static abstract class Callback implements IBinder.DeathRecipient {
   @Override
   public void binderDied() {
     onDestroy();
   }
   public abstract void onDestroy();
}
private IBinder mBinder;

public void registerCallback(Callback callback) {
   mBinder.asBinder().linkToDeath(callback, 0);
}

public void unregisterCallback(Callback callback) {
   mBinder.asBinder().unlinkToDeath(callback, 0);
}
```

AIDL のバージョン管理

- メソッドをオーバーロードしたい
  - 不可能なので別名にするしかない
  - バッドノウハウを使う(後述)
- Parcelable を変更したい
  - フィールドの追加のとき、Parcel からの復帰や書き込みの順番は最後に追加

AIDL のバージョン管理におけるバッドノウハウ

- オーバーロードできない問題
  - 実は Service の実装にメソッドがありさえすればオーバーロードはできる
  - e.g. 呼び出し先の AIDL で hoge() メソッドを hoge(int) に変更、ただし Service の実装には hoge() も残してある => 呼び出し元のアプリの AIDL には hoge() しかなくても hoge() はちゃんと呼べる

ProGuard

- メソッド名の難読化は必ず回避する
  - メソッド名が解決できなくなって死んでしまう
- AIDL を含めた SDK やライブラリを配布するなら、ProGuard の設定も同梱する

まとめ

まとめ

- AIDL の使い所
  - 他のアプリに自分のアプリのもつ機能を公開する
    - アドオン追加、データの取得・送信、コンテンツの操作など
  - SDK として配布
    - Intent のような中間レイヤがなくなるので動作は速い
    - コールバック等を用意して開発者フレンドリーにする

まとめ

- AIDL の制約
  - オーバーロード不可
  - プリミティブ型、`String`型、一部コレクション型以外は`Parcelable`を実装
- Parcelable の制約
  - 巨大なオブジェクトはメモリの制約を考えること
  - オブジェクトの互換性に気をつける

まとめ

- ProGuard
  - IPC に関わる部分は難読化を回避する
- プロセスの死活監視
  - 死んだプロセスに通信をしない
- 通信相手のバリデーション
  - 誰にでも公開していい機能かどうかで制約を決める

drivemode

Keishin Yokomaku / shibuya.apk #11 @ GMO Yours

Practical IPC

Reference

- Deep Dive into Android IPC/Binder Framework http://bit.ly/2dsFzaI
- Android Binder-IPC http://bit.ly/2gFKWcl
- Binder のはじめの一歩と Android のプロセス間通信 http://bit.ly/2fY9lWi
- Binder のはじめの一歩 http://bit.ly/2fNs35k
- Open Binder http://bit.ly/2fNlP5h
- Intent の概要 http://bit.ly/2fANsvs

Registration has started! http://bit.ly/2gFFK8z