转载请注明链接: https://blog.csdn.net/feather_wch/article/details/79425955

> 总结归纳Binder连接池中的知识点以及实现方法

# Binder连接池实例

版本: 2019/3/14-1



Tips：通过AIDL生成Binder文件步骤

> 1. 在工程main中java目录的包内创建 `Book.java` -需要跨进程传输的类、单纯实现Parcelable接口
> 2. 在工程 `main` 目录下创建 `aidl` 文件夹，在其中创建 `Book.aidl` 文件-用于对应 `Book.java`
> 3. 在 `main/aidl` 目录下创建 `IBookManager.aidl` -内部编写实际需要Server远程处理的操作
> 4. 选择android studio的build中make project-系统就会自动生成对应java文件 `IBookManager` ， 位于目录 `app\build\generated\source\aidl\debug\` 包下

1、Binder连接池的实现思路

> 1. `main/aidl` 下创建 `IBinderPool.aidl`，并生成对应 `java`文件 (如上方法)
> 2. 实现 `BinderPool.java` -内部实现连接服务端和处理服务端返回对应功能Binder的逻辑
> 3. 实现 `BinderPoolService.java` -实现服务端
> 4. 实现 功能的`Binder` ， 如 `ICalculate`、`IBookManager` 等等。

2、Binder连接池-实例：

> 1、AIDL( `main/aidl/...` )

```
// IBinderPool.aidl-Binder连接池
package com.example.a6005001819.androiddeveloper;

interface IBinderPool {
    IBinder queryBinder(int binderCode);
}

// ICalculate.aidl-功能1：计算
package com.example.a6005001819.androiddeveloper;

interface ICalculate {
    int add(int first, int second);
    int sub(int first, int second);
}

//IBookManager.aidl-功能2：图书管理
package com.example.a6005001819.androiddeveloper;
//关键：导入Book.java
import com.example.a6005001819.androiddeveloper.Book;

interface IBookManager {
    List<Book> getBookList();
    void addBook(in Book book);
}

//Book.aidl-图书
package com.example.a6005001819.androiddeveloper;
parcelable Book;
```

> 2、线程池实现(BinderPool.java)

```
public class BinderPool {
    private Context mContext;
    private static BinderPool mInstance;
    //服务端返回的连接池的Binder对象
    private static IBinderPool mIBinderPool;
    //同步工具-连接成功(回调onServiceConnected)后，才会允许后续如queryBinder的操作
    private CountDownLatch mCountDownLatch;

    //功能Binder对应的code
    public static final int BINDER_NONE = -1;
    public static final int BINDER_CALCULATE= 1;
    public static final int BINDER_BOOKMANAGER= 2;

    /**
     * 1. 单例形式的BinderPool连接池
     */
    private BinderPool(Context context){
        mContext = context.getApplicationContext();
        connectBinderPoolService();
    }
    public static BinderPool getInstance(Context context){
        if(mIBinderPool == null){
            synchronized (BinderPool.class){
                if(mIBinderPool == null){
                    mInstance = new BinderPool(context);
                }
            }
        }
        return mInstance;
    }

    /**
     * 2. 查询Binder-利用服务端连接池的Binder
     */
    public IBinder queryBinder(int binderCode){
        IBinder binder = null;
        if(mIBinderPool != null){
            try {
                binder = mIBinderPool.queryBinder(binderCode);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        }
        return  binder;
    }

    /**================================================*
     * 3. 连接服务端Service
     *================================================*/
    private synchronized  void connectBinderPoolService(){
        mCountDownLatch = new CountDownLatch(1);
        //1. 连接服务端
        Intent intent = new Intent(mContext, BinderPoolService.class);
        mContext.bindService(intent, mBinderPoolConnection, Context.BIND_AUTO_CREATE);
```

```
        //2. 一直阻塞到onServiceConnected回调
        try {
            mCountDownLatch.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    private ServiceConnection mBinderPoolConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            //1. 获取到服务端的BinderPool
            mIBinderPool = IBinderPool.Stub.asInterface(service);
            //2. 设置死亡代理
            try {
                mIBinderPool.asBinder().linkToDeath(mBinderPoolDeathRecipient, 0);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
            //3. CountDownLatch计数-1，结果为0时唤醒await
            mCountDownLatch.countDown();
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            //* 可以在这里处理Binder意外死亡问题---会在客户端的UI线程中被回调
        }
    };

    /**
     * 死亡代理： Binder意外死亡时，binderDied()会在客户端的Binder线程池中被回调
     *   注意：禁止UI操作
     */
    private IBinder.DeathRecipient mBinderPoolDeathRecipient = new IBinder.DeathRecipient() {
        @Override
        public void binderDied() {
            //1. 移除先前注册的死亡通知
            mIBinderPool.asBinder().unlinkToDeath(mBinderPoolDeathRecipient, 0);
            mIBinderPool = null;
            //2. 连接服务端：会进行连接和设置死亡代理
            connectBinderPoolService();
        }
    };

    /**==================================================*
     * 4. 服务端Service的BinderPool的Binder对象
     *   作用于服务端，对于客户端这些代码没什么用
     *==================================================*/
    public static class BinderPoolImpl extends  IBinderPool.Stub{

        @Override
        public IBinder queryBinder(int binderCode) throws RemoteException {
            IBinder binder = null;
            switch (binderCode){
                case BINDER_CALCULATE:
```

```
                    binder = new ICalculateImpl();
                    break;
                case BINDER_BOOKMANAGER:
                    break;
                default:
                    break;
            }
            return binder;
        }
    }
}
```

### 3、服务端实现(BinderPoolService.java)

```
public class BinderPoolService extends Service {

    private Binder mBinderPool = new BinderPool.BinderPoolImpl();

    public BinderPoolService() {
    }

    @Override
    public IBinder onBind(Intent intent) {
        return mBinderPool;
    }
}


//AndroidManifest.xml:
<service
    android:name=".BinderPoolService"
    android:enabled="true"
    android:exported="true"
    android:process=":remote"></service>
```

### 4、功能的实现(计算、图书管理等)- main/src

```
//功能1：计算-ICalculateImpl.java
public class ICalculateImpl extends ICalculate.Stub{
    @Override
    public int add(int a, int b) throws RemoteException {
        return a + b;
    }
}
```

```
//功能2：图书馆管理
//图书-Book.java
public class Book implements Parcelable {
    public int bookId;

    public Book(int bookId){
        this.bookId = bookId;
    }
    private Book(Parcel in){
        bookId = in.readInt();
    }

    public static final Creator<Book> CREATOR = new Creator<Book>() {
        @Override
        public Book createFromParcel(Parcel in) {
            return new Book(in);
        }

        @Override
        public Book[] newArray(int size) {
            return new Book[size];
        }
    };

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel parcel, int i) {
        parcel.writeInt(bookId);
    }
}
//图书管理-IBookManagerImpl.java:
public class IBookManagerImpl extends IBookManager.Stub{

    private CopyOnWriteArrayList<Book> mBookList = new CopyOnWriteArrayList<>();

    @Override
    public List<Book> getBookList() throws RemoteException {
        return mBookList;
    }

    @Override
    public void addBook(Book book) throws RemoteException {
        mBookList.add(book);
    }
}
```

> 5、客户端使用BinderPool进行操作

```
//0. 使用线程池
ExecutorService cachedThreadPool = Executors.newCachedThreadPool();
cachedThreadPool.execute(new Runnable() {
    @Override
    public void run() {
        //1. 获取Binder连接池
        BinderPool mBinderPool = BinderPool.getInstance(MainActivity.this);
        //2. 获取需要的功能
        ICalculate  mCalculate = ICalculateImpl.asInterface(mBinderPool.queryBinder(Bi
        try {
            //3.进行操作：计算
            Log.i("feather", "result=" + mCalculate.add(10, 20));
        } catch (RemoteException e) {
            e.printStackTrace();
        }
        //4. 进行操作：图书管理
        IBookManager mBookManager = IBookManagerImpl.asInterface(mBinderPool.queryBinde
        try {
            mBookManager.addBook(new Book(1));
            Log.i("feather", "bookId=" + mBookManager.getBookList().get(0).bookId);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
});
```