LRH1993 android进程间通信方式 5540f64 on 2 Sep 2017

1 contributor

951 lines (763 sloc) 33.1 KB

Android 中的 IPC 方式

一、使用 Intent

1. Activity，Service，Receiver 都支持在 Intent 中传递 Bundle 数据，而 Bundle 实现了 Parcelable 接口，可以在不同的进程间进行传输。
2. 在一个进程中启动了另一个进程的 Activity，Service 和 Receiver ，可以在 Bundle 中附加要传递的数据通过 Intent 发送出去。

二、使用文件共享

1. Windows 上，一个文件如果被加了排斥锁会导致其他线程无法对其进行访问，包括读和写；而 Android 系统基于 Linux，使得其并发读取文件没有限制地进行，甚至允许两个线程同时对一个文件进行读写操作，尽管这样可能会出问题。
2. 可以在一个进程中序列化一个对象到文件系统中，在另一个进程中反序列化恢复这个对象（**注意**：并不是同一个对象，只是内容相同。）。
3. SharedPreferences 是个特例，系统对它的读 / 写有一定的缓存策略，即内存中会有一份 ShardPreferences 文件的缓存，系统对他的读 / 写就变得不可靠，当面对高并发的读写访问，SharedPreferences 有很多大的几率丢失数据。因此，IPC 不建议采用 SharedPreferences。

三、使用 Messenger

Messenger 是一种轻量级的 IPC 方案，它的底层实现是 AIDL ，可以在不同进程中传递 Message 对象，它一次只处理一个请求，在服务端不需要考虑线程同步的问题，服务端不存在并发执行的情形。

- 服务端进程：服务端创建一个 Service 来处理客户端请求，同时通过一个 Handler 对象来实例化一个 Messenger 对象，然后在 Service 的 onBind 中返回这个 Messenger 对象底层的 Binder 即可。

```
public class MessengerService extends Service {

    private static final String TAG = MessengerService.class.getSimpleName();

    private class MessengerHandler extends Handler {

        /**
         * @param msg
         */
        @Override
        public void handleMessage(Message msg) {

            switch (msg.what) {
                case Constants.MSG_FROM_CLIENT:
                    Log.d(TAG, "receive msg from client: msg = [" + msg.getData().getString(Constants.MSG_KEY) + "]")
                    Toast.makeText(MessengerService.this, "receive msg from client: msg = [" + msg.getData().getStrir
                    Messenger client = msg.replyTo;
                    Message replyMsg = Message.obtain(null, Constants.MSG_FROM_SERVICE);
                    Bundle bundle = new Bundle();
                    bundle.putString(Constants.MSG_KEY, "我已经收到你的消息，稍后回复你！");
                    replyMsg.setData(bundle);
                    try {
                        client.send(replyMsg);
                    } catch (RemoteException e) {
                        e.printStackTrace();
                    }
```

```
                    break;
                default:
                    super.handleMessage(msg);
            }
        }
    }

    private Messenger mMessenger = new Messenger(new MessengerHandler());


    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return mMessenger.getBinder();
    }
}
```

- 客户端进程：首先绑定服务端 Service，绑定成功之后用服务端的 IBinder 对象创建一个 Messenger，通过这个 Messenger 就可以向服务端发送消息了，消息类型是 Message 。如果需要服务端响应，则需要创建一个 Handler 并通过它来创建一个 Messenger（和服务端一样），并通过 Message 的 replyTo 参数传递给服务端。服务端通过 Message 的 replyTo 参数就可以回应客户端了。

```
public class MainActivity extends AppCompatActivity {
    private static final String TAG = MainActivity.class.getSimpleName();
    private Messenger mGetReplyMessenger = new Messenger(new MessageHandler());
    private Messenger mService;

    private class MessageHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case Constants.MSG_FROM_SERVICE:
                    Log.d(TAG, "received msg form service: msg = [" + msg.getData().getString(Constants.MSG_KEY) + "]
                    Toast.makeText(MainActivity.this, "received msg form service: msg = [" + msg.getData().getString(
                    break;
                default:
                    super.handleMessage(msg);
            }
        }
    }



    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }

    public void bindService(View v) {
        Intent mIntent = new Intent(this, MessengerService.class);
        bindService(mIntent, mServiceConnection, Context.BIND_AUTO_CREATE);
    }

    public void sendMessage(View v) {
        Message msg = Message.obtain(null,Constants.MSG_FROM_CLIENT);
        Bundle data = new Bundle();
        data.putString(Constants.MSG_KEY, "Hello! This is client.");
        msg.setData(data);
        msg.replyTo = mGetReplyMessenger;
        try {
            mService.send(msg);
        } catch (RemoteException e) {
            e.printStackTrace();
        }

    }

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

```
    private ServiceConnection mServiceConnection = new ServiceConnection() {
        /**
         * @param name
         * @param service
         */
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            mService = new Messenger(service);
            Message msg = Message.obtain(null,Constants.MSG_FROM_CLIENT);
            Bundle data = new Bundle();
            data.putString(Constants.MSG_KEY, "Hello! This is client.");
            msg.setData(data);
            //
            msg.replyTo = mGetReplyMessenger;
            try {
                mService.send(msg);
            } catch (RemoteException e) {
                e.printStackTrace();
            }

        }

        /**
         * @param name
         */
        @Override
        public void onServiceDisconnected(ComponentName name) {


        }
    };
}
```

注意：客户端和服务端是通过拿到对方的 Messenger 来发送 Message 的。只不过客户端通过 bindService onServiceConnected 而服务端通过 message.replyTo 来获得对方的 Messenger 。Messenger 中有一个 Hanlder 以串行的方式处理队列中的消息。不存在并发执行，因此我们不用考虑线程同步的问题。

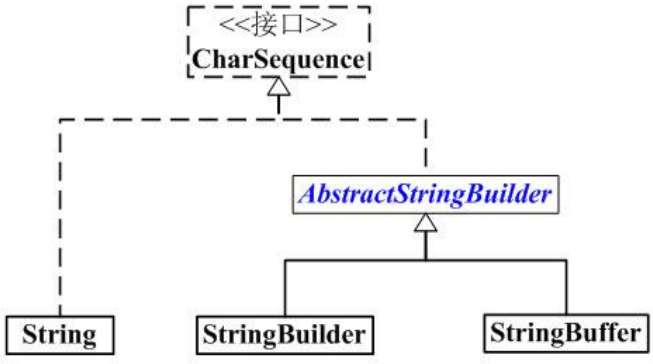
Markdown

四、使用 AIDL

Messenger 是以串行的方式处理客户端发来的消息，如果大量消息同时发送到服务端，服务端只能一个一个处理，所以大量并发请求就不适合用 Messenger ，而且 Messenger 只适合传递消息，不能跨进程调用服务端的方法。AIDL 可以解决并发和跨进程调用方法的问题，要知道 Messenger 本质上也是 AIDL ，只不过系统做了封装方便上层的调用而已。

AIDL 文件支持的数据类型

- *基本数据类型*;



- *String* 和 *CharSequence*
- *ArrayList* ， 里面的元素必须能够被 AIDL 支持;
- *HashMap* ， 里面的元素必须能够被 AIDL 支持;
- *Parcelable* ， 实现 Parcelable 接口的对象； **注意：如果 AIDL 文件中用到了自定义的 Parcelable 对象，必须新建一个和它同名的 AIDL 文件。**
- *AIDL* ， AIDL 接口本身也可以在 AIDL 文件中使用。

服务端

服务端创建一个 Service 用来监听客户端的连接请求，然后创建一个 AIDL 文件，将暴露给客户端的接口在这个 AIDL 文件中声明，最后在 Service 中实现这个 AIDL 接口即可。

客户端

绑定服务端的 Service ，绑定成功后，将服务端返回的 Binder 对象转成 AIDL 接口所属的类型，然后就可以调用 AIDL 中的方法了。客户端调用远程服务的方法，被调用的方法运行在服务端的 Binder 线程池中，同时客户端的线程会被挂起，如果服务端方法执行比较耗时，就会导致客户端线程长时间阻塞，导致 ANR 。客户端的 onServiceConnected 和 onServiceDisconnected 方法都在 UI 线程中。

服务端访问权限管理

- 使用 Permission 验证，在 manifest 中声明

```
<permission android:name="com.jc.ipc.ACCESS_BOOK_SERVICE"
    android:protectionLevel="normal"/>
<uses-permission android:name="com.jc.ipc.ACCESS_BOOK_SERVICE"/>
```

服务端 onBinder 方法中

```
public IBinder onBind(Intent intent) {
    //Permission 权限验证
    int check = checkCallingOrSelfPermission("com.jc.ipc.ACCESS_BOOK_SERVICE");
    if (check == PackageManager.PERMISSION_DENIED) {
        return null;
    }

    return mBinder;
}
```

- Pid Uid 验证

详细代码：

```
// Book.aidl
package com.jc.ipc.aidl;

parcelable Book;
```

```
// IBookManager.aidl
package com.jc.ipc.aidl;

import com.jc.ipc.aidl.Book;
import com.jc.ipc.aidl.INewBookArrivedListener;

// AIDL 接口中只支持方法，不支持静态常量，区别于传统的接口
interface IBookManager {
    List<Book> getBookList();

    // AIDL 中除了基本数据类型，其他数据类型必须标上方向,in,out 或者 inout
    // in 表示输入型参数
    // out 表示输出型参数
    // inout 表示输入输出型参数

    void addBook(in Book book);

    void registerListener(INewBookArrivedListener listener);
    void unregisterListener(INewBookArrivedListener listener);

}
```

```
// INewBookArrivedListener.aidl
package com.jc.ipc.aidl;
import com.jc.ipc.aidl.Book;

// 提醒客户端新书到来

interface INewBookArrivedListener {
```

```
    void onNewBookArrived(in Book newBook);
}
```

```
public class BookManagerActivity extends AppCompatActivity {
    private static final String TAG = BookManagerActivity.class.getSimpleName();
    private static final int MSG_NEW_BOOK_ARRIVED = 0x10;
    private Button getBookListBtn,addBookBtn;
    private TextView displayTextView;
    private IBookManager bookManager;
    private Handler mHandler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case MSG_NEW_BOOK_ARRIVED:
                    Log.d(TAG, "handleMessage: new book arrived " + msg.obj);
                    Toast.makeText(BookManagerActivity.this, "new book arrived " + msg.obj, Toast.LENGTH_SHORT).show(
                    break;
                default:
                    super.handleMessage(msg);
            }

        }
    };

    private ServiceConnection mServiceConn = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            bookManager = IBookManager.Stub.asInterface(service);
            try {
                bookManager.registerListener(listener);
            } catch (RemoteException e) {
                e.printStackTrace();
            }

        }

        @Override
        public void onServiceDisconnected(ComponentName name) {

        }
    };

    private INewBookArrivedListener listener = new INewBookArrivedListener.Stub() {
        @Override
        public void onNewBookArrived(Book newBook) throws RemoteException {
            mHandler.obtainMessage(MSG_NEW_BOOK_ARRIVED, newBook).sendToTarget();

        }
    };

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.book_manager);
        displayTextView = (TextView) findViewById(R.id.displayTextView);
        Intent intent = new Intent(this, BookManagerService.class);
        bindService(intent, mServiceConn, BIND_AUTO_CREATE);

    }


    public void getBookList(View view) {
        try {
            List<Book> list = bookManager.getBookList();
            Log.d(TAG, "getBookList: " + list.toString());
            displayTextView.setText(list.toString());
        } catch (RemoteException e) {
            e.printStackTrace();
        }

    }

    public void addBook(View view) {
        try {
            bookManager.addBook(new Book(3, "天龙八部"));
        } catch (RemoteException e) {
```

```
            e.printStackTrace();
        }
    }

    @Override
    protected void onDestroy() {
        if (bookManager != null && bookManager.asBinder().isBinderAlive()) {
            Log.d(TAG, "unregister listener " + listener);
            try {
                bookManager.unregisterListener(listener);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        }
        unbindService(mServiceConn);
        super.onDestroy();
    }
}
```

```
public class BookManagerService extends Service {
    private static final String TAG = BookManagerService.class.getSimpleName();

    // CopyOnWriteArrayList 支持并发读写，实现自动线程同步，他不是继承自 ArrayList
    private CopyOnWriteArrayList<Book> mBookList = new CopyOnWriteArrayList<Book>();
    //对象是不能跨进程传输的，对象的跨进程传输本质都是反序列化的过程，Binder 会把客户端传递过来的对象重新转化生成一个新的对象
    //RemoteCallbackList 是系统专门提供的用于删除系统跨进程 listener 的接口，利用底层的 Binder 对象是同一个
    //RemoteCallbackList 会在客户端进程终止后，自动溢出客户端注册的 listener ，内部自动实现了线程同步功能。
    private RemoteCallbackList<INewBookArrivedListener> mListeners = new RemoteCallbackList<>();
    private AtomicBoolean isServiceDestroied = new AtomicBoolean(false);


    private Binder mBinder = new IBookManager.Stub() {

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

        @Override
        public void addBook(Book book) throws RemoteException {
            Log.d(TAG, "addBook: " + book.toString());
            mBookList.add(book);

        }

        @Override
        public void registerListener(INewBookArrivedListener listener) throws RemoteException {
            mListeners.register(listener);
        }

        @Override
        public void unregisterListener(INewBookArrivedListener listener) throws RemoteException {
            mListeners.unregister(listener);
        }
    };

    @Override
    public void onCreate() {
        super.onCreate();
        mBookList.add(new Book(1, "老人与海"));
        mBookList.add(new Book(2, "哈姆雷特"));
        new Thread(new ServiceWorker()).start();
    }

    private void onNewBookArrived(Book book) throws RemoteException {
        mBookList.add(book);

        int count = mListeners.beginBroadcast();

        for (int i = 0; i < count; i++) {
            INewBookArrivedListener listener = mListeners.getBroadcastItem(i);
            if (listener != null) {
                listener.onNewBookArrived(book);
            }
        }
```

```
        mListeners.finishBroadcast();

    }

    private class ServiceWorker implements Runnable {
        @Override
        public void run() {
            while (!isServiceDestroied.get()) {
                try {
                    Thread.sleep(5000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                int bookId = mBookList.size() +1;
                Book newBook = new Book(bookId, "new book # " + bookId);
                try {
                    onNewBookArrived(newBook);
                } catch (RemoteException e) {
                    e.printStackTrace();
                }
            }

        }
    }


    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        //Permission 权限验证
        int check = checkCallingOrSelfPermission("com.jc.ipc.ACCESS_BOOK_SERVICE");
        if (check == PackageManager.PERMISSION_DENIED) {
            return null;
        }

        return mBinder;
    }

    @Override
    public void onDestroy() {
        isServiceDestroied.set(true);
        super.onDestroy();
    }
}
```

五、使用 ContentProvider

用于不同应用间数据共享，和 Messenger 底层实现同样是 Binder 和 AIDL，系统做了封装，使用简单。 系统预置了许多 ContentProvider ，如通讯录、日程表，需要跨进程访问。 使用方法：继承 ContentProvider 类实现 6 个抽象方法，这六个方法均运行在 ContentProvider 进程中，除 onCreate 运行在主线程里，其他五个方法均由外界回调运行在 Binder 线程池中。

ContentProvider 的底层数据，可以是 SQLite 数据库，可以是文件，也可以是内存中的数据。

详见代码：

```
public class BookProvider extends ContentProvider {
    private static final String TAG = "BookProvider";
    public static final String AUTHORITY = "com.jc.ipc.Book.Provider";

    public static final Uri BOOK_CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/book");
    public static final Uri USER_CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/user");

    public static final int BOOK_URI_CODE = 0;
    public static final int USER_URI_CODE = 1;
    private static final UriMatcher sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);

    static {
        sUriMatcher.addURI(AUTHORITY, "book", BOOK_URI_CODE);
        sUriMatcher.addURI(AUTHORITY, "user", USER_URI_CODE);
    }

    private Context mContext;
```

```
    private SQLiteDatabase mDB;

    @Override
    public boolean onCreate() {
        mContext = getContext();
        initProviderData();

        return true;
    }

    private void initProviderData() {
        //不建议在 UI 线程中执行耗时操作
        mDB = new DBOpenHelper(mContext).getWritableDatabase();
        mDB.execSQL("delete from " + DBOpenHelper.BOOK_TABLE_NAME);
        mDB.execSQL("delete from " + DBOpenHelper.USER_TABLE_NAME);
        mDB.execSQL("insert into book values(3,'Android');");
        mDB.execSQL("insert into book values(4,'iOS');");
        mDB.execSQL("insert into book values(5,'Html5');");
        mDB.execSQL("insert into user values(1,'haohao',1);");
        mDB.execSQL("insert into user values(2,'nannan',0);");

    }

    @Nullable
    @Override
    public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {
        Log.d(TAG, "query, current thread"+ Thread.currentThread());
        String table = getTableName(uri);
        if (table == null) {
            throw new IllegalArgumentException("Unsupported URI" + uri);
        }

        return mDB.query(table, projection, selection, selectionArgs, null, null, sortOrder, null);
    }

    @Nullable
    @Override
    public String getType(Uri uri) {
        Log.d(TAG, "getType");
        return null;
    }

    @Nullable
    @Override
    public Uri insert(Uri uri, ContentValues values) {
        Log.d(TAG, "insert");
        String table = getTableName(uri);
        if (table == null) {
            throw new IllegalArgumentException("Unsupported URI" + uri);
        }
        mDB.insert(table, null, values);
        // 通知外界 ContentProvider 中的数据发生变化
        mContext.getContentResolver().notifyChange(uri, null);
        return uri;
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        Log.d(TAG, "delete");
        String table = getTableName(uri);
        if (table == null) {
            throw new IllegalArgumentException("Unsupported URI" + uri);
        }
        int count = mDB.delete(table, selection, selectionArgs);
        if (count > 0) {
            mContext.getContentResolver().notifyChange(uri, null);
        }

        return count;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) {
        Log.d(TAG, "update");
        String table = getTableName(uri);
        if (table == null) {
            throw new IllegalArgumentException("Unsupported URI" + uri);
        }
```

```
        int row = mDB.update(table, values, selection, selectionArgs);
        if (row > 0) {
            getContext().getContentResolver().notifyChange(uri, null);
        }
        return row;
    }

    private String getTableName(Uri uri) {
        String tableName = null;
        switch (sUriMatcher.match(uri)) {
            case BOOK_URI_CODE:
                tableName = DBOpenHelper.BOOK_TABLE_NAME;
                break;
            case USER_URI_CODE:
                tableName = DBOpenHelper.USER_TABLE_NAME;
                break;
            default:
                break;
        }

        return tableName;

    }
}
```

```
public class DBOpenHelper extends SQLiteOpenHelper {

    private static final String DB_NAME = "book_provider.db";
    public static final String BOOK_TABLE_NAME = "book";
    public static final String USER_TABLE_NAME = "user";

    private static final int DB_VERSION = 1;

    private String CREATE_BOOK_TABLE = "CREATE TABLE IF NOT EXISTS "
            + BOOK_TABLE_NAME + "(_id INTEGER PRIMARY KEY," + "name TEXT)";

    private String CREATE_USER_TABLE = "CREATE TABLE IF NOT EXISTS "
            + USER_TABLE_NAME + "(_id INTEGER PRIMARY KEY," + "name TEXT,"
            + "sex INT)";



    public DBOpenHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_BOOK_TABLE);
        db.execSQL(CREATE_USER_TABLE);

    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }
}
```

```
public class ProviderActivity extends AppCompatActivity {
    private static final String TAG = ProviderActivity.class.getSimpleName();
    private TextView displayTextView;
    private Handler mHandler;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_provider);
        displayTextView = (TextView) findViewById(R.id.displayTextView);
        mHandler = new Handler();

        getContentResolver().registerContentObserver(BookProvider.BOOK_CONTENT_URI, true, new ContentObserver(mHandle
            @Override
            public boolean deliverSelfNotifications() {
```

```
                return super.deliverSelfNotifications();
            }

            @Override
            public void onChange(boolean selfChange) {
                super.onChange(selfChange);
            }

            @Override
            public void onChange(boolean selfChange, Uri uri) {
                Toast.makeText(ProviderActivity.this, uri.toString(), Toast.LENGTH_SHORT).show();
                super.onChange(selfChange, uri);
            }
        });




    }

    public void insert(View v) {
        ContentValues values = new ContentValues();
        values.put("_id",1123);
        values.put("name", "三国演义");
        getContentResolver().insert(BookProvider.BOOK_CONTENT_URI, values);

    }
    public void delete(View v) {
        getContentResolver().delete(BookProvider.BOOK_CONTENT_URI, "_id = 4", null);


    }
    public void update(View v) {
        ContentValues values = new ContentValues();
        values.put("_id",1123);
        values.put("name", "三国演义新版");
        getContentResolver().update(BookProvider.BOOK_CONTENT_URI, values , "_id = 1123", null);


    }
    public void query(View v) {
        Cursor bookCursor = getContentResolver().query(BookProvider.BOOK_CONTENT_URI, new String[]{"_id", "name"}, nu
        StringBuilder sb = new StringBuilder();
        while (bookCursor.moveToNext()) {
            Book book = new Book(bookCursor.getInt(0),bookCursor.getString(1));
            sb.append(book.toString()).append("\n");
        }
        sb.append("--------------------------------").append("\n");
        bookCursor.close();

        Cursor userCursor = getContentResolver().query(BookProvider.USER_CONTENT_URI, new String[]{"_id", "name", "se
        while (userCursor.moveToNext()) {
            sb.append(userCursor.getInt(0))
                    .append(userCursor.getString(1)).append(" ,")
                    .append(userCursor.getInt(2)).append(" ,")
                    .append("\n");
        }
        sb.append("--------------------------------");
        userCursor.close();
        displayTextView.setText(sb.toString());
    }
}
```

六、使用 Socket

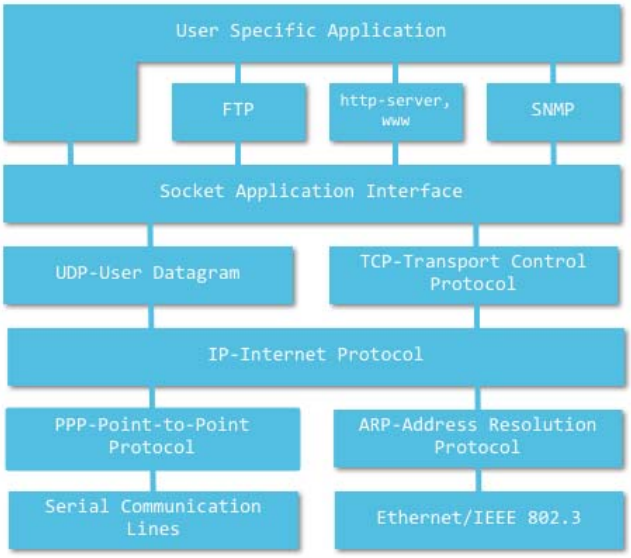
> Socket起源于 Unix，而 Unix 基本哲学之一就是"一切皆文件"，都可以用"打开 open –读写 write/read –关闭 close "模式来操作。Socket 就是该模式的一个实现，网络的 Socket 数据传输是一种特殊的 I/O，Socket 也是一种文件描述符。Socket 也具有一个类似于打开文件的函数调用： Socket()，该函数返回一个整型的Socket 描述符，随后的连接建立、数据传输等操作都是通过该 Socket 实现的。
>
> 常用的 Socket 类型有两种：流式 Socket（SOCK_STREAM）和数据报式 Socket（SOCK_DGRAM）。流式是一种面向连接的 Socket，针对于面向连接的 TCP 服务应用；数据报式 Socket 是一种无连接的 Socket ，对应于无连接的 UDP 服务应用。

Socket 本身可以传输任意字节流。

谈到Socket，就必须要说一说 TCP/IP 五层网络模型：

- 应用层：规定应用程序的数据格式，主要的协议 HTTP，FTP，WebSocket，POP3 等；
- 传输层：建立"端口到端口" 的通信，主要的协议：TCP，UDP；
- 网络层：建立"主机到主机"的通信，主要的协议：IP，ARP ，IP 协议的主要作用：一个是为每一台计算机分配 IP 地址，另一个是确定哪些地址在同一子网；
- 数据链路层：确定电信号的分组方式，主要的协议：以太网协议；
- 物理层：负责电信号的传输。

Socket 是连接应用层与传输层之间接口（API）。



只实现 TCP Socket 。

Client 端代码：

```
public class TCPClientActivity extends AppCompatActivity implements View.OnClickListener{

    private static final String TAG = "TCPClientActivity";
    public static final int MSG_RECEIVED = 0x10;
    public static final int MSG_READY = 0x11;
    private EditText editText;
    private TextView textView;
    private PrintWriter mPrintWriter;
    private Socket mClientSocket;
    private Button sendBtn;
    private StringBuilder stringBuilder;
    private Handler mHandler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case MSG_READY:
                    sendBtn.setEnabled(true);
                    break;
                case MSG_RECEIVED:
                    stringBuilder.append(msg.obj).append("\n");
                    textView.setText(stringBuilder.toString());
                    break;
                default:
                    super.handleMessage(msg);
            }

    }
    };

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.tcp_client_activity);
        editText = (EditText) findViewById(R.id.editText);
```

```
        textView = (TextView) findViewById(R.id.displayTextView);
        sendBtn = (Button) findViewById(R.id.sendBtn);
        sendBtn.setOnClickListener(this);
        sendBtn.setEnabled(false);
        stringBuilder = new StringBuilder();

        Intent intent = new Intent(TCPClientActivity.this, TCPServerService.class);
        startService(intent);

        new Thread(){
            @Override
            public void run() {
                connectTcpServer();
            }
        }.start();
    }


    private String formatDateTime(long time) {
        return new SimpleDateFormat("(HH:mm:ss)").format(new Date(time));
    }

    private void connectTcpServer() {
        Socket socket = null;
        while (socket == null) {
            try {
                socket = new Socket("localhost", 8888);
                mClientSocket = socket;
                mPrintWriter = new PrintWriter(new BufferedWriter(
                        new OutputStreamWriter(socket.getOutputStream())
                ), true);
                mHandler.sendEmptyMessage(MSG_READY);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        // receive message
        BufferedReader bufferedReader = null;
        try {
            bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        } catch (IOException e) {
            e.printStackTrace();
        }
        while (!isFinishing()) {
            try {
                String msg = bufferedReader.readLine();
                if (msg != null) {
                    String time = formatDateTime(System.currentTimeMillis());
                    String showedMsg = "server " + time + ":" + msg
                            + "\n";
                    mHandler.obtainMessage(MSG_RECEIVED, showedMsg).sendToTarget();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }

        }
    }

    @Override
    public void onClick(View v) {
        if (mPrintWriter != null) {
            String msg = editText.getText().toString();
            mPrintWriter.println(msg);
            editText.setText("");
            String time = formatDateTime(System.currentTimeMillis());
            String showedMsg = "self " + time + ":" + msg + "\n";
            stringBuilder.append(showedMsg);

        }

    }

    @Override
    protected void onDestroy() {
        if (mClientSocket != null) {
            try {
```

```
                mClientSocket.shutdownInput();
                mClientSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        super.onDestroy();
    }
}
```

Server 端代码：

```
public class TCPServerService extends Service {
    private static final String TAG = "TCPServerService";
    private boolean isServiceDestroyed = false;
    private String[] mMessages = new String[]{
            "Hello! Body!",
            "用户不在线！请稍后再联系！",
            "请问你叫什么名字呀？",
            "厉害了，我的哥！",
            "Google 不需要科学上网是真的吗？",
            "扎心了，老铁！！！"
    };


    @Override
    public void onCreate() {
        new Thread(new TCPServer()).start();
        super.onCreate();
    }

    @Override
    public void onDestroy() {
        isServiceDestroyed = true;
        super.onDestroy();
    }

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

    private class TCPServer implements Runnable {

        @Override
        public void run() {
            ServerSocket serverSocket = null;
            try {
                serverSocket = new ServerSocket(8888);
            } catch (IOException e) {
                e.printStackTrace();
                return;
            }
            while (!isServiceDestroyed) {
                // receive request from client
                try {
                    final Socket client = serverSocket.accept();
                    Log.d(TAG, "================ accept ==================");
                    new Thread(){
                        @Override
                        public void run() {
                            try {
                                responseClient(client);
                            } catch (IOException e) {
                                e.printStackTrace();
                            }
                        }
                    }.start();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

        }
    }
```

```
    private void responseClient(Socket client) throws IOException {
        //receive message
        BufferedReader in = new BufferedReader(
                new InputStreamReader(client.getInputStream()));
        //send message
        PrintWriter out = new PrintWriter(
                new BufferedWriter(
                        new OutputStreamWriter(
                                client.getOutputStream())),true);
        out.println("欢迎来到聊天室！");

        while (!isServiceDestroyed) {
            String str = in.readLine();
            Log.d(TAG, "message from client: " + str);
            if (str == null) {
                return;
            }
            Random random = new Random();
            int index = random.nextInt(mMessages.length);
            String msg = mMessages[index];
            out.println(msg);
            Log.d(TAG, "send Message: " + msg);
        }
        out.close();
        in.close();
        client.close();

    }
}
```

演示：

UDP Socket 可以自己尝试着实现。