FTP 应用软件说明书

Design By Leaves

目　录

摘 要

本作品设计了一款应用于 Android 移动设备的 FTP 应用软件，该软件改进了目前市场上存在的已有FTP软件的一些不足之处，同时也能满足多线程下载、断点续传的基本要求，实现了 FTP 上文件的移动传输，方便了移动办公。实验证实，该应用软件在满足以上功能的前提下也能够保证文件传输的可靠性，即应用软件能够根据已有的下载、上载情况保证所传输的文件不被损坏。

关键词：FTP，多线程，断点续传

第一部分 作品设计背景

1.1 作品设计动机

FTP（File Transfer Protocol）是 Internet 上用来传送文件的协议（文件传输协议）。它是为了我们能够在 Internet 上互相传送文件而制定的的文件传送标准，规定了 Internet 上文件如何传送。也就是说，通过 FTP 协议，我们就可以跟 Internet 上的 FTP 服务器进行文件的上传（Upload）或下载（Download）等动作。

FTP 应用软件在学校、公司、单位等工作学习中是必不可少的，它极大的方便了我们日常工作学习中的电子资料共享。目前 PC 端的 FTP 应用软件已经比较完善，功能相对齐全，已足够满足工作学习的需求，但是在移动端 FTP 应用软件还不够完善，例如不支持中文文件名显示、无法实现断点续传、界面操作复杂等一些问题。

为了解决以上的问题，本项目重新设计开发了 FTP 应用软件，通过第三方工具 Apache Commons Net 可以实现 FTP 上文件的可靠传输，并实现了文件的多线程下载、上传，断点续传，同时更接地气的实现了文件名中文解析。

1.2 作品的主要内容

该作品实现的功能内容主要包括如下几点：

① 有图形化操作界面客户端（Android）；
② 支持本地和远程系统的文件列表；
③ 支持文件的下载和上传；
④ 支持中文文件名解析；
⑤ 支持上传文件进度显示；
⑥ 支持断点续传；
⑦ 支持多线程传输。

第二部分 UI 界面设计

2.1 UI 设计原则

用户界面（UI）是一个应用程序的窗口，决定着用户对软件的喜好与否。在移动互联网的影响下，各种新的应用程序不断被开发者、企业等发布，这些应用程序如何吸引用户很大程度上靠得的是良好的 UI 设计，因为它能吸引用户甚至改变用户的操作习惯。所以该软件在 UI 设计上遵守了以下三个要点：一是要遵守谷歌公司发布的 Android UI 设计规范，按照规范的要求来进行 UI 的设计；二是要参考当前受用户喜欢的应用程序的 UI 设计，并结合自身软件的使用情况及功能来进行设计；三是要考虑到 Android 设备的碎片化带来的屏幕适配问题，进行了适配性设计。本软件最后经过不断的设计，并确定了该软件的最终 UI 设计方案。

在 Android 的世界中，有四大重要组件，包括 Activity，Service，Content Provider，BroadcastReceive。其中的 Activity 就是就是最基本也是最为常用的四大组件之一。Activity 生命周期的几个过程简单描述如下，其基本流程图如下图（图 2.1.1 Activity 生命周期）所示

启动 Activity：系统会先调用 onCreate 方法，然后调用 onStart 方法，最后调用 onResume，Activity 进入运行状态。

当前 Activity 被其他 Activity 覆盖其上或被锁屏：系统会调用 onPause 方法，暂停当前 Activity 的执行。

当前 Activity 由被覆盖状态回到前台或解锁屏：系统会调用 onResume 方法，再次进入运行状态。

当前 Activity 转到新的 Activity 界面或按 Home 键回到主屏，自身退居后台：系统会先调用 onPause 方法，然后调用 onStop 方法，进入停滞状态。

用户后退回到此 Activity：系统会先调用 onRestart 方法，然后调用 onStart 方法，最后调用 onResume 方法，再次进入运行状态。

当前 Activity 处于被覆盖状态或者后台不可见状态，即第 2 步和第 4 步，系统内存不足，杀死当前 Activity，而后用户退回当前 Activity：再次调用 onCreate 方法、onStart 方法、onResume 方法，进入运行状态。

用户退出当前 Activity：系统先调用 onPause 方法，然后调用 onStop 方法，最后调用 onDestory 方法，结束当前 Activity。

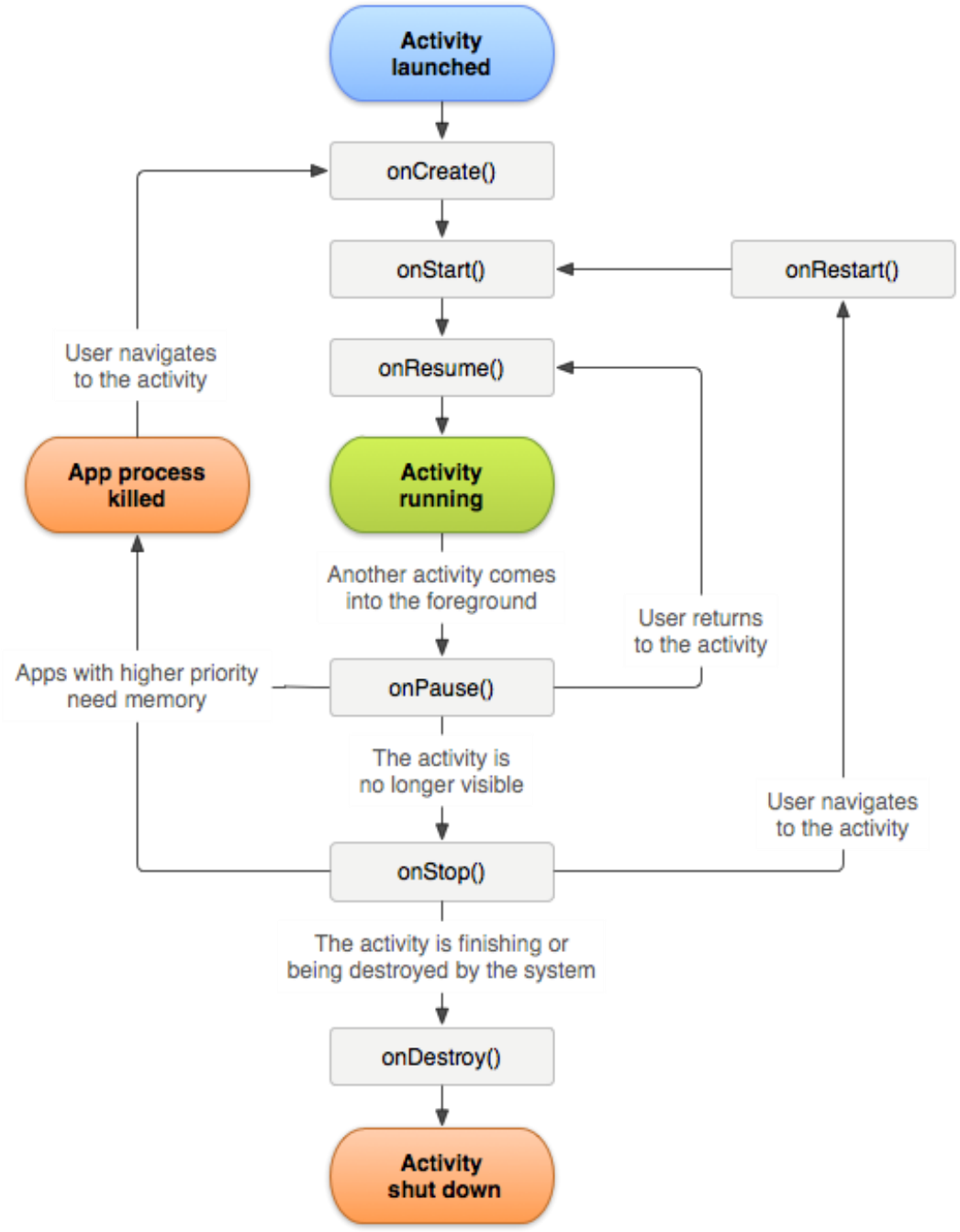

图 2.1.1 Activity 生命周期

2.2 主界面介绍

该软件主界面采用 Google 原生 Navigation Drawer Activity 设计，没有采用第三方 UI 设计框架，其主界面导航栏从屏幕的左侧滑出，显示应用导航的视图（如图 2.2.1 主界面导航栏），内容区域采用 ListView 控件进行内容展示（如图 2.2.2 主界面内容展示）。

图 2.2.1 主界面导航栏

图 2.2.2 主界面内容展示

可以通过主界面 ActionBar 上的菜单按钮添加 FTP 站点，也可通过长按 FTP 站点，对其进行修改或者删除。所有 FTP 站点数据信息都将保存到 SQLite 数据库中，便于数据持久化。

2.3 FTP 远程列表界面介绍

FTP 远程列表界面设计采用了文件资源管理器的样式（如图 2.3.1 FTP 远程列表界面），在布局上由图标、文件名、文件类型、返回上一级目录和返回根目录构成。另外顶部的 ActionBar 左侧加入了返回按钮，便于快速返回上一个 Activity，ActionBar 右侧加入了菜单选项，可以进入本地文件列表、下载列表和上传列表。

图 2.3.1 FTP 远程列表界面

2.4 本地文件列表界面介绍

本地文件列表界面设计同样采用了文件资源管理器的样式（如图 2.4.1 本地文件列表），在布局上由图标、文件名、文件类型、返回上一级目录和返回根目录构成。另外顶部的 ActionBar 左侧加入了返回按钮，便于快速返回上一个 Activity，ActionBar 右侧加入了菜单选项，可以进入本地文件列表、下载列表和上传列表。

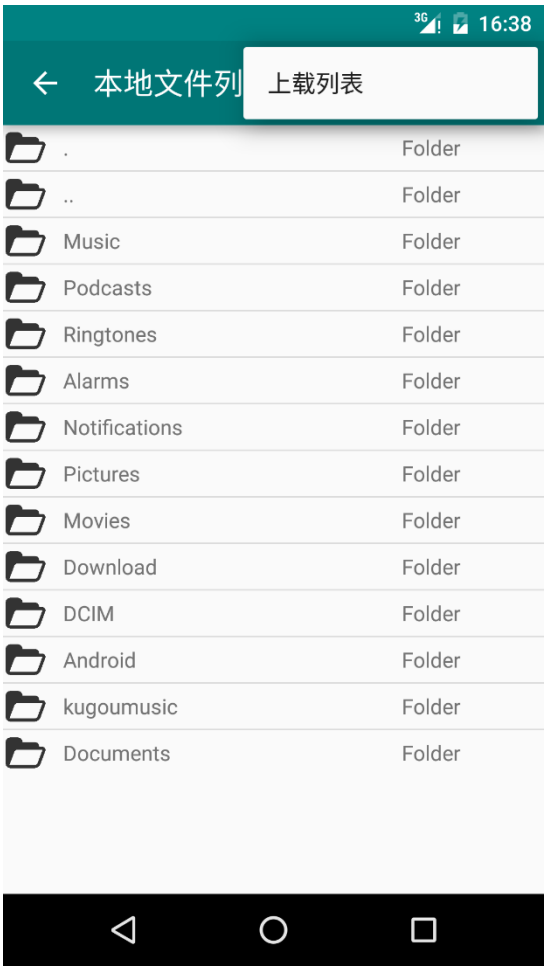

图 2.4.1 本地文件列表

2.5 本地文件浏览器界面介绍

本地文件浏览器界面相比本地文件列表界面比较简单，该界面主要用于对文件的浏览和查看，可以在这里打开你想要查看的文件，如果该格式文件系统不能自动识别，则可以选择一种打开方式打开（如图 2.5.1 本地文件浏览器）。

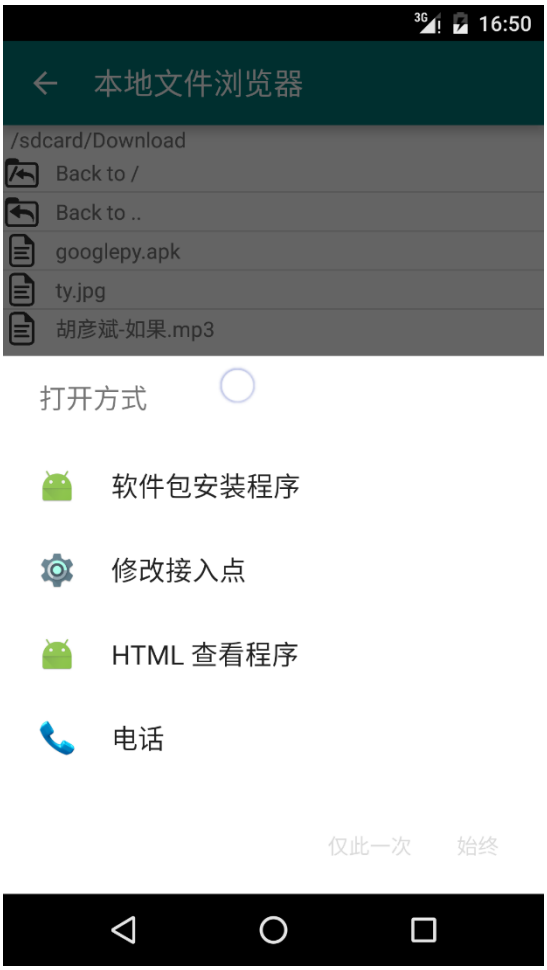

图 2.5.1 本地文件浏览器

2.6 下载界面介绍

下载列表主要用于控制文件的下载，其主界面主要由文本、进度条、图片按钮组成（如图 2.6.1 下载列表）。下载进度条会随着下载的进行而动态更新。另外顶部的 ActionBar 左侧加入了返回按钮，便于快速返回上一个 Activity。具体 XML 布局详见程序清单 2.6.1 下载界面 XML 布局。

程序清单 2.6.1 下载界面 XML 布局

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

```
android:orientation="vertical"
tools:context="com.slm.ftp.DownloadActivity">

<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp">

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="FileName"
            android:textSize="15sp"
            android:id="@+id/tvFileName"
            android:layout_weight="9" />
    </LinearLayout>

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_vertical">

        <LinearLayout
            android:orientation="horizontal"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="7"
            android:gravity="center_vertical">
```

```
    <ProgressBar
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:id="@+id/pbProgress"
        android:layout_below="@+id/textView"
        android:layout_toRightOf="@+id/textView"
        android:layout_toEndOf="@+id/textView"
        android:layout_weight="7.5" />

    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="0 %"
        android:id="@+id/tvProgress"
        android:layout_weight="2.5"
        android:gravity="right" />
</LinearLayout>

<LinearLayout
    android:orientation="horizontal"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:gravity="center_vertical|right"
    android:layout_weight="3">

    <ImageButton
        android:layout_width="30dp"
        android:layout_height="30dp"
        android:background="@drawable/down"
        android:id="@+id/btStart" />

    <ImageButton
        android:layout_width="30dp"
```

```
                android:layout_height="30dp"
                android:background="@drawable/pause"
                android:id="@+id/btStop"
                android:layout_marginLeft="7dp" />

            <ImageButton
                android:layout_width="25dp"
                android:layout_height="25dp"
                android:background="@drawable/delete"
                android:id="@+id/btCancel"
                android:layout_marginLeft="7dp" />
        </LinearLayout>
    </LinearLayout>
</LinearLayout>
</LinearLayout>
```

图 2.6.1 下载列表

2.7 上载界面介绍

上载列表主要用于控制文件的上载，其主界面和下载界面一样主要由文本、进度条、图片按钮组成（如图 2.7.1 上载列表）。下载进度条会随着下载的进行而动态更新。另外顶部的 ActionBar 左侧加入了返回按钮，便于快速返回上一个 Activity。

图 2.7.1 上载列表

2.8 FTP 信息界面介绍

FTP 信息界面主要用于添加更新 FTP 站点信息，其界面主要由文本、编辑框、按钮组成（如图 2.8.1 FTP 信息）。通过此界面添加保存 FTP 站点后就可以对添加的 FTP 进行访问，如果 FTP 给予用户读写权限就可以在 FTP 上下载文件或上载文件。


17:05
FTP信息
FTP站点名称：
Mine
域 名 / IP：
192.168.3.3
端 口 号：
21
用 户 名：
anonymous
密 码：
保 存

图 2.8.1 FTP 信息

第三部分 功能逻辑设计

3.1 代码软件包关系

本项目在程序结构设计时对各功能模块的职责进行了详细划分，按照各功能模块的职责将程序分为 5 个包（如图 3.1.1 程序包结构），分别为：com.slm.ftp，com.slm.adapter，com.slm.dbhelper，com.slm.method，com.slm.upDown。

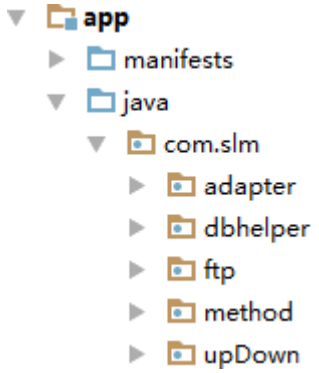

图 3.1.1 程序包结构

com.slm.ftp 包主要用于管理各个 Activity，该软件的所有界面实现都在该包中进行，对软件界面实行统一管理。

com.slm.adapter 包主要用于 FTP 远程列表、本地列表、下载列表、上载列表的 Adapter 管理。我们知道，在 Android 中 Adapter 是与 View 之间提供数据的桥梁（如图 3.1.2 Adapter 数据适配示意图），在本软件中，通过重写 BaseAdapter 使其为 FTP 远程列表、本地列表、下载列表、上载列表的数据显示服务，按照样式中预先设定好的显示格式显示数据。

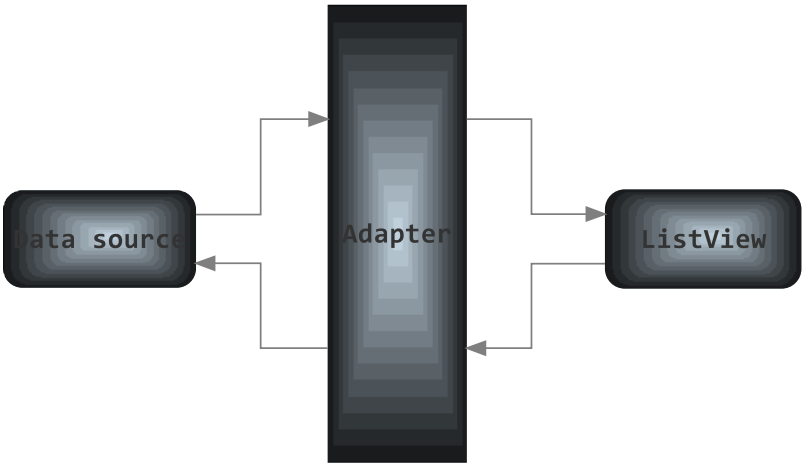

图 3.1.2 Adapter 数据适配示意图

com.slm.dbhelper 包主要用于管理数据库操作对象。把创建数据库以及数据库操作对象封装，这样可以防止在程序的其他地方出现数据库操作，降低了程度代码的耦合度，使程

序更加健壮。

com.slm.method 主要提供一些通用方法，如 FTP 信息对象、线程对象的类的封装，另外也用于如 FTP 远程列表中文件大小显示等。

com.slm.upDown 包主要用于文件下载、上载。其中包括两个服务类（DownloadService、UploadService）、两个任务类（DownloadTask、UploadTask）和一个 FTP 操作类。该软件核心功能实现均在这个包中完成。

3.2 数据库设计

该软件在程序中拥有一个数据库（SQLite），数据库中包含两张表，其中一张表为 ftp_info（表 3.2.1 ftp_info 表设计），另一张表为 thread_info（表 3.2.2 thread_info 表设计）。其中 ftp_info 表用于存储 FTP 站点信息，便于下载、上载时能够找到所需 ftp 连接信息，thread_info 表用于存储线程信息，即把每一个下载、上载的线程信息保存到数据库，主要用于下载进度显示和多线程断点续传。

表 3.2.1 ftp_info **表设计**

编号	字段名	字段类型	说明	备注
1	_id	integer	每一条记录 id	主键，自动增长
2	ftpName	text	FTP 站点名	
3	hostName	text	主机名	
4	port	integer	端口号	
5	userName	text	用户名	
6	password	text	密码	

表 3.2.2 thread_info **表设计**

编号	字段名	字段类型	说明	备注
1	_id	integer	每一条记录 id	主键，自动增长
2	ftp_id	integer	对应 ftp 连接信息 id	
3	remote_url	text	ftp 远程路径	
4	local_url	text	本地路径	
5	length	integer	下载或上载文件长度	
6	start	integer	开始位置	多线程断点续传
7	end	integer	结束为止	多线程断点续传
8	finished	integer	完成进度	
9	status	integer	完成状态	

3.3 各 Acitivity 逻辑关系

程序以 MyFTPActivity 为纽带可以启动几个基本 Activity，只有从主界面进入 FTP 远程列表界面需要向 FTPActivity 传递一个 FTP 信息对象，该对象用于后期下载、上载线程信息生成。由 FTPActivity 生成的线程信息都将被保存到线程信息数据库中，用于下载列表、上载列表显示。各 Activity 之间信息传递和联系如下图（图 3.3.1 各 Activity 之间信息传递）所示。

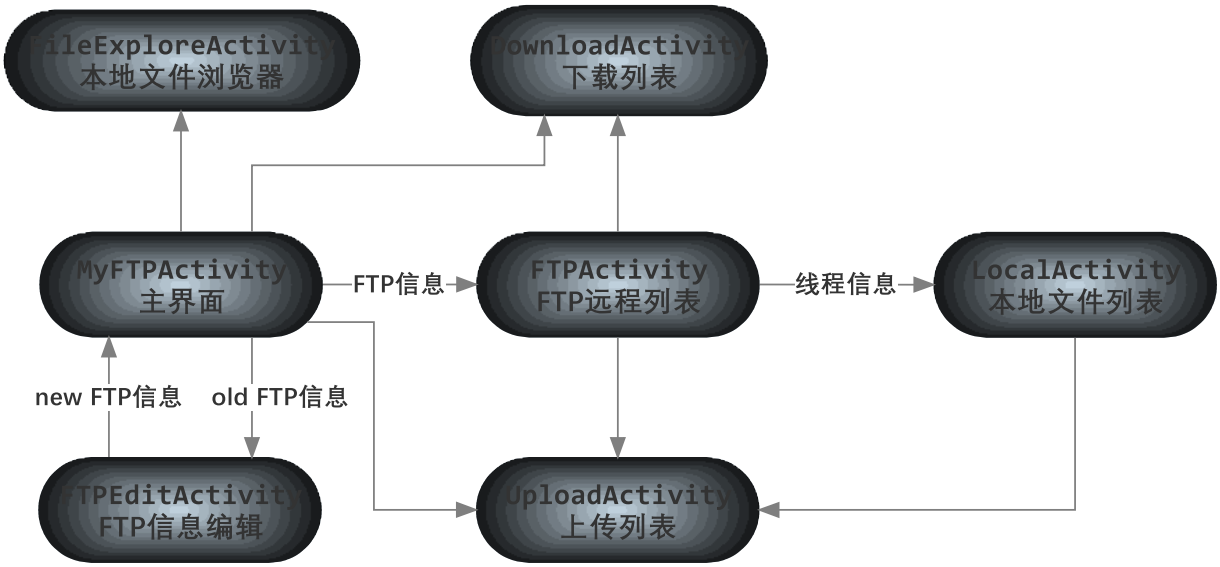

图 3.3.1 各 Activity 之间信息传递

3.4 界面线程与后台线程关系

以下载模块为例（上载模块原理相同），启动 DownloadActivity 后我们可以对下载列表中的下载任务进行操作。为了使程序进入后台之后仍能够正常下载，本程序把下载任务交给 Service 来执行，具体流程图详见图 3.4.1 基于 Service 的下载任务。

首先程序启动一个 Service 并向其发出动作指令，告诉 Service 我们即将进行的任务是什么。

当发出“下载开始”指令后，Service 会启动一个下载任务（DowloadTask），下载任务会在子线程中运行并随时存储下载进度，下载任务通过广播（Broadcast）告诉 DownloadActivity 下载进度，DownloadActivity 收到进度更新广播（BroadcastReceiver）后进行下载进度更新。

当发出“下载暂停”指令后，Service 会给 DowloadTask 发出一个下载中断指令，DowloadTask 保存下载进度后其子线程就会正常结束。DownloadActivity 也不会再收到进度更新广播。

当发出“取消下载”指令后，Service 就会把当前的下载线程信息从数据库中删除，该

下载任务将不会被启动。

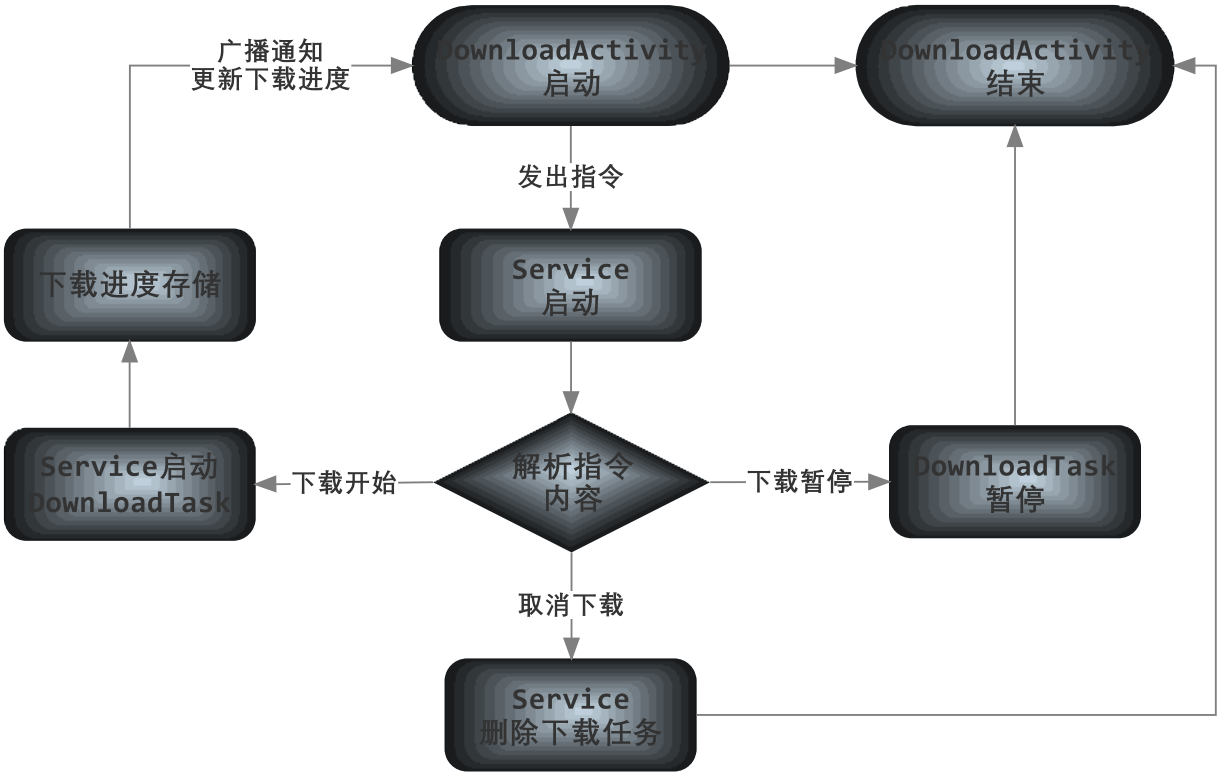

图 3.4.1 基于 Service 的下载任务

第四部分 关键技术实现

4.1 单例模式实现数据库安全访问

数据库同时被多个线程访问操作，常规的代码结构会造成线程不安全。在这种情况下就要引入单例模式，保证这个数据库操作类永远只有一个实例，同时在数据库操作函数中也要加入关键字 synchronized。

实现单例模式的第一步是把 DBHelper 类的构造函数私有化，保证其他类不能实例化 DBHelper 类。具体代码见程序清单 4.1.1 DBHelper 类的构造函数私有化。

程序清单 4.1.1 DBHelper 类的构造函数私有化

```
private DBHelper(Context context) {
    super(context, DB_NAME, null, VERSION);
}
```

第二步是定义一个静态引用，获得 DBHelper 类的实例化对象，这样可以保证 DBHelper 类只能被实例化一次，保证不同线程得到的 DBHelper 引用都是同一个对象。具体代码见程序清单 4.1.2 获得类的对象。

程序清单 4.1.2 获得类的对象

```
public static DBHelper getInstance(Context context) {
    if (null == dbHelper) {
        dbHelper = new DBHelper(context);
    }
    return dbHelper;
}
```

第三为了保证线程安全，需要在对数据库进行插入、删除和修改操作的方法中加入关键字 synchronized，保证某一时刻只有一个线程可以进行一个操作，其它线程必须等待前一个线程完成操作。

4.2 多线程后台上下载

要实现程序能够后台上下载，那就要把相关线程放到 Service 中运行，这样可以保证在 UI 线程结束时，上下载线程不被结束。以下载为例（上载同理），当 DownloadService 被启动时，并接收到 ACTION_START（下载开始）指令后，程序首先通过定义好的初始化子线程把下载任务需要用到的信息初始化。当一切初始化工作完成时，初始化子线程通过

Handler 向 DownloadService 发出一个空消息，意思是告诉 DownloadService 我的工作完成了，你可以启动下载任务了。DownloadService 中的 Handler 接收到初始化子线程发来的消息就开始创建一个下载任务对象，同时将每个下载任务存入下载任务集合 tasks 中，详见程序清单 4.2.1 Handler 接收消息并启动下载任务。

程序清单 4.2.1 Handler **接收消息并启动下载任务**

```
Handler handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        DownloadTask downloadTask = new
            DownloadTask(DownloadService.this,
            threadInfo, ftpClient, ftpFile);
        downloadTask.start();
        tasks.put(threadInfo.getThread_id(), downloadTask);
    }
};
```

将每个下载任务存储起来的目的是为了实现断点续传，当出于某些需求时，我们需要暂停下载任务，这时程序就可以从下载集合映射中取出相关下载任务进行中断下载操作。

4.3 多线程断点续传

实现多线程断点续传的原理就是在每一个下载任务中定义一个全局公有的变量，本程序中该变量为 isPause。当 DownloadService（以下载为例，上载同理）收到 ACTION_STOP（下载暂停）指令后，程序就从下载任务集合取出对应下载任务，并将下载任务中的 isPause 值设为 true，表示程序需要停止该下载任务。因为每一个下载任务都是在一个子线程中运行，对于正确结束一个线程，我们不能用 stop 方法。在下载任务中下载文件是通过 while 循环不断读取一个文件流。因此要正常结束一个线程，就只要让这个线程退出 while 循环。因为下载任务中设置了变量 isPause，每一次循环都会判断 isPause 是否为真值，一旦 isPause 为真，就将现在的进度保存到数据库中，并使用 return 方法结束 while 循环，这样线程就正常结束了。一旦程序需要再次启动该下载任务时，只要从数据库中读取下载进度，并设置文件流的读取开始位置，这样就能继续该文件的下载，直至完成。

第五部分 作品使用说明及功能测试

5.1 系统运行环境要求

本程序最低 Android 版本要求为 4.0，需要获得网络权限、读写存储器权限。

5.2 软件使用说明

首先通过主界面的选项菜单，添加一个 FTP 站点，如图 5.2.1 添加 FTP 站点。

图 5.2.1 添加 FTP 站点

输入 FTP 连接相关信息，如图 5.2.2 编辑 FTP 连接信息，输入完成后点击保存，添加的新的 FTP 站点就在主界面显示，如图 5.2.3 FTP 添加成功显示站点信息。

图 5.2.2 编辑 FTP 连接信息

图 5.2.3 FTP 添加成功显示站点信息

点击对应站点，可浏览 FTP 远程列表中的相关文件信息，如图 5.2.4 FTP 远程列表。

图 5.2.4 FTP 远程列表

长按某一文件，会弹出上下文菜单，可选择“快速下载”和“加入下载列表”两项，如图 5.2.5 下载上下文菜单，选择“快速下载”，系统会在后台静默开启下载，直至结束，选择“加入下载列表”，则由下载列表统一管理下载任务。

图 5.2.5 下载上下文菜单

在 FTP 远程列表的选项菜单中（图 5.2.6 FTP 远程列表的选项菜单），选择“本地文件列表”，可进入手机的本地文件列表（图 5.2.7 本地文件列表）。

图 5.2.6 FTP 远程列表的选项菜单

图 5.2.7 本地文件列表

选择想要上传的文件，长按该文件，弹出上下文菜单，如图 5.2.8 上载上下文菜单，加入上载列表后，则由上载列表统一管理上载任务，默认上载路径为进入本地文件列表时的 FTP 当前路径。

图 5.2.8 上载上下文菜单

在下载列表中，可以开始、暂停和取消下载任务，如图 5.2.9 开始、暂停下载任务，上载同理。

图 5.2.9 开始、暂停下载任务

最后为了方便操作，在主界面的左侧有一个抽屉式导航栏，如图 5.2.10 抽屉式导航栏，在这里可以方便地访问几个主要操作的界面

图 5.2.10 抽屉式导航栏

5.3 系统功能测试

为了确保软件的可靠性，本项目通过软件工程学方法对本软件进行如下测试：

表 5.3.1 FTP 连接模块测试

输入数据	预期结果	实验结果	结果	结论
IP：192.168.3.3 Port:21 用户名：anonymous 密码：空字符串	连接成功	连接成功	正确	
IP：10.132.252.208 Port：21 用户名：android 密码：android	连接成功	连接成功	正确	
IP：10.132.252.208	连接失败	连接失败	正确	对于有用户名

Port：21 用户名：android 密码：空字符串	和密码的 FTP，必须正确输入密码才能连接

表 5.3.2 FTP 远程列表测试

输入数据	预期结果	实验结果	结果	结论
IP 为 192.168.3.3 的 FTP 被点击	FTP 列表正确加载	FTP 列表正确加载	正确	
IP 为 10.132.252.208 的 FTP 被点击	FTP 列表正确加载	FTP 列表正确加载	正确	
点击相应 FTP 列表中的文件夹	进入该文件夹列表	进入该文件夹列表	正确	

表 5.3.3 本地文件列表测试

输入数据	预期结果	实验结果	结果	结论
点击本地文件列表	列出根目录文件列表	列出根目录文件列表	正确	
点击相应列表中的文件夹	进入该文件夹列表	进入该文件夹列表	正确	

表 5.3.4 下载测试

输入数据	预期结果	实验结果	结果	结论
点击下载按钮	下载开始	下载开始	正确	
点击暂停按钮	下载暂停	下载暂停	正确	
再次点击下载按钮	获得上次下载的进度继续开始下载	获得上次下载的进度继续开始下载	正确	该软件支持断点续传
点击取消按钮	任务取消	任务取消	正确	

表 5.3.5 上载测试

输入数据	预期结果	实验结果	结果	结论
点击上载按钮	上载开始	上载开始	正确	
点击暂停按钮	上载暂停	上载暂停	正确	

再次点击上载按钮	获得上次上载的进度继续开始上载	获得上次上载的进度继续开始上载	正确	该软件支持断点续传
点击取消按钮	任务取消	任务取消	正确	

附件 关键程序代码

本报告附下载相关代码 DownloadService、DownloadTask、ThreadInfo（因上载原理相同，不在重复放置代码）：

程序清单 5.3.1 DownloadService

```
package com.slm.upDown;

import android.app.Service;
import android.content.Intent;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.support.annotation.Nullable;
import android.util.Log;

import com.slm.dbhelper.FtpDAO;
import com.slm.dbhelper.ThreadDAO;
import com.slm.method.FTPInfo;
import com.slm.method.ThreadInfo;

import org.apache.commons.net.ftp.FTPClient;
import org.apache.commons.net.ftp.FTPFile;

import java.io.IOException;
import java.util.LinkedHashMap;
import java.util.Map;

/**
 * Created by Leaves on 2016/6/4.
 */
public class DownloadService extends Service {

    public static final String ACTION_START = "ACTION_START"; // 开始下载
    public static final String ACTION_STOP = "ACTION_STOP"; // 暂停下载
    public static final String ACTION_CANCEL = "ACTION_CANCEL"; // 暂停下载
```

```
    public static final String ACTION_UPDATE = "ACTION_UPDATE"; // 更新 UI
    public static final String ACTION_FINISHED = "ACTION_FINISHED"; // 下载结束

    private FtpDAO ftpDAO = null;
    private ThreadDAO threadDAO = null;

    private FTP ftp = null;
    private FTPClient ftpClient = null;
    private ThreadInfo threadInfo = null;
    private FTPInfo ftpInfo = null;
    private FTPFile ftpFile = null;

    private Map<Integer, DownloadTask> tasks = new LinkedHashMap<Integer,
DownloadTask>(); // 下载任务集合

    Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            DownloadTask downloadTask = new DownloadTask(DownloadService.this,
threadInfo, ftpClient, ftpFile);
            downloadTask.start();
            tasks.put(threadInfo.getThread_id(), downloadTask);
        }
    };

    /**
     * 启动 Service 自动调用 onStartCommand
     *
     * @param intent
     * @param flags
     * @param startId
     * @return
     */
    @Override
```

```
    public int onStartCommand(Intent intent, int flags, int startId) {
        if (intent.getAction().equals(ACTION_START)) {
            threadInfo = (ThreadInfo) intent.getSerializableExtra("threadInfo");
            String                              fileName                              =
threadInfo.getRemote_url().substring(threadInfo.getRemote_url().lastIndexOf("/")
+ 1);
            // 下载任务初始化
            InitThread initThread = new InitThread();
            initThread.start();
            Log.i("ACTION_START ---->", fileName + " Start");

        } else if (intent.getAction().equals(ACTION_STOP)) {
            threadInfo = (ThreadInfo) intent.getSerializableExtra("threadInfo");
            String                              fileName                              =
threadInfo.getRemote_url().substring(threadInfo.getRemote_url().lastIndexOf("/")
+ 1);
            // 取出下载任务
            DownloadTask downloadTask = tasks.get(threadInfo.getThread_id());
            if (downloadTask != null) {
                downloadTask.isPause = true;
            }
            Log.i("ACTION_STOP ---->", fileName + " Stop");

        } else if (intent.getAction().equals(ACTION_CANCEL)) {
            threadInfo = (ThreadInfo) intent.getSerializableExtra("threadInfo");
            threadDAO = new ThreadDAO(this);
            threadDAO.deleteThread(threadInfo);
            // 广播通知下载任务取消
            Intent intent1 = new Intent();
            intent1.setAction(ACTION_CANCEL);
            intent1.putExtra("threadInfo", threadInfo);
            this.sendBroadcast(intent1);
            Log.i("ACTION_CANCEL ---->", "Cancel");
        }
```

```
        return super.onStartCommand(intent, flags, startId);
    }

    /**
     * 初始化下载任务
     */
    class InitThread extends Thread {
        @Override
        public void run() {
            super.run();
            try {
                ftpDAO = new FtpDAO(DownloadService.this);
                ftpInfo = ftpDAO.getFTPInfo(threadInfo.getFtp_id());
                ftp = new FTP(ftpInfo);
                ftp.openConnect();
                ftpClient = ftp.getFtpClient();
                ftpFile = ftp.getFTPFile(threadInfo.getRemote_url());
            } catch (IOException e) {
                e.printStackTrace();
            }
            handler.sendEmptyMessage(0);
        }
    }

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

}
```

程序清单 5.3.2 DownloadTask

```
package com.slm.upDown;
```

```
import android.content.Context;
import android.content.Intent;
import android.util.Log;

import com.slm.dbhelper.ThreadDAO;
import com.slm.method.ThreadInfo;

import org.apache.commons.net.ftp.FTPClient;
import org.apache.commons.net.ftp.FTPFile;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;

/**
 * Created by Leaves on 2016/6/4.
 */
public class DownloadTask {

    private Context context = null;

    private ThreadDAO threadDAO = null;
    private ThreadInfo threadInfo = null;

    private FTPFile ftpFile = null;
    private FTPClient ftpClient = null;

    private String remotePath = null; // 带文件名路径
    private String localPath = null; // 带文件名路径

    public boolean isPause = false;
    public static final int BROADCAST_TIME = 1000; // 广播时间间隔

    public DownloadTask(Context context, ThreadInfo threadInfo, FTPClient
```

```
ftpClient, FTPFile ftpFile) {
        this.context = context;
        this.threadInfo = threadInfo;
        this.ftpClient = ftpClient;
        this.ftpFile = ftpFile;
        threadDAO = new ThreadDAO(context);
    }

    public void start() {
        this.localPath = threadInfo.getLocal_url() + ftpFile.getName();
        this.remotePath = threadInfo.getRemote_url();
        DownloadThread downloadThread = new DownloadThread();
        downloadThread.start();
    }

    /**
     * 下载线程类
     */
    private class DownloadThread extends Thread {
        @Override
        public void run() {
            super.run();
            download();
        }
    }

    /**
     * 从 FTP 服务器上下载文件,支持断点续传，上传百分比汇报
     * @return 上传的状态
     * @throws IOException
     */
    private void download() {
        FileOutputStream fileOutputStream = null;
        InputStream inputStream = null;
```

```
        try {
            // 设置被动模式
            ftpClient.enterLocalPassiveMode();
            // 设置以二进制方式传输
            ftpClient.setFileType(org.apache.commons.net.ftp.FTP.BINARY_FILE_TYPE);

            long ftpFileSize = ftpFile.getSize();
            Log.i("ftpFileSize --->", ftpFile.getSize() + "");
            File localFile = new File(localPath);
            long localSize = 0L;

            if (localFile.exists()) {
                localSize = localFile.length();
                //判断本地文件大小是否大于远程文件大小
                if (localSize >= ftpFileSize) {
                    Log.i("下载进度 --->", "已下载完成，下载中止");
                }
                // 进行断点续传，并记录状态
                fileOutputStream = new FileOutputStream(localFile, true); // 文件追
加模式
                ftpClient.setRestartOffset(localSize);
                threadInfo.setFinished((int) localSize);
            } else {
                fileOutputStream = new FileOutputStream(localFile); // 新建文件
            }

            // 更新下载进度
            Intent intent = new Intent();
            intent.setAction(DownloadService.ACTION_UPDATE);

            // 传输开始
            inputStream   =   ftpClient.retrieveFileStream(remotePath);   //   new
String(remotePath.getBytes("GBK"), "iso-8859-1") 会报错
            long process = (long) (localSize * 1.0 / ftpFileSize * 100);
            byte[] bytes = new byte[1024];
```

```
int len = -1;
long time = System.currentTimeMillis();
while ((len = inputStream.read(bytes)) != -1) {
    // 写入文件
    fileOutputStream.write(bytes, 0, len);
    // 累加每个线程完成的进度
    threadInfo.setFinished(threadInfo.getFinished() + len);
    localSize += len;
    // 下载进度控制台显示
    long nowProcess = (long) (localSize * 1.0 / ftpFileSize * 100);
    if (nowProcess > process) {
        process = nowProcess;
        Log.i("下载进度 --->", ftpFile.getName() + " " + process);
    }

    // 每 BROADCAST_TIMEms 发送一次广播
    if (System.currentTimeMillis() - time > BROADCAST_TIME) {
        time = System.currentTimeMillis();
        // 把下载进度通过广播发送给 Activity
        intent.putExtra("threadInfo", threadInfo);
        context.sendBroadcast(intent);
    }

    if (isPause) {
        // 存储下载进度
        threadDAO.updateThread(threadInfo);
        return;
    }
}

threadInfo.setStatus(1);
threadDAO.updateThread(threadInfo);
// 发送广播通知 UI 下载任务结束
intent.putExtra("threadInfo", threadInfo);
context.sendBroadcast(intent);
```

```
            Log.i("下载完成 --->", "");

        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (fileOutputStream != null) {
                    fileOutputStream.close();
                }
                if (inputStream != null) {
                    inputStream.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

程序清单 5.3.3 ThreadInfo

```
package com.slm.method;

import java.io.Serializable;

/**
 * Created by Leaves on 2016/6/4.
 */
public class ThreadInfo implements Serializable {

    /**
     * 线程 id
     */
    private int thread_id;

    /**
```

```
 * ftp id
 */
private int ftp_id;

/**
 * 线程对应下载链接
 */
private String remote_url;

/**
 * 线程对应本地位置
 */
private String local_url;

/**
 * 该线程载文件总长度长度
 */
private int length = 0;

/**
 * 下载开始位置
 */
private int start = 0;

/**
 * 下载结束位置
 */
private int end = 0;

/**
 * 下载进度
 */
private int finished = 0;

/**
```

```
     * 改任务是否已下载完成 0 未完成 1 完成
     */
    private int status = 0;

    /**
     * 备用，非数据库字段
     */
    private int Remark = -1;

    public ThreadInfo() { }

    public ThreadInfo(int ftp_id, String remote_url, String local_url, int length,
int start, int end, int finished, int status) {
        this.ftp_id = ftp_id;
        this.remote_url = remote_url;
        this.local_url = local_url;
        this.length = length;
        this.start = start;
        this.end = end;
        this.finished = finished;
        this.status = status;
    }

    public int getThread_id() {
        return thread_id;
    }

    public void setThread_id(int thread_id) {
        this.thread_id = thread_id;
    }

    public int getFtp_id() {
        return ftp_id;
    }
```

```
public void setFtp_id(int ftp_id) {
    this.ftp_id = ftp_id;
}

public String getRemote_url() {
    return remote_url;
}

public void setRemote_url(String remote_url) {
    this.remote_url = remote_url;
}

public String getLocal_url() {
    return local_url;
}

public void setLocal_url(String local_url) {
    this.local_url = local_url;
}

public int getLength() {
    return length;
}

public void setLength(int length) {
    this.length = length;
}

public int getStart() {
    return start;
}

public void setStart(int start) {
    this.start = start;
}
```

```
public int getEnd() {
    return end;
}

public void setEnd(int end) {
    this.end = end;
}

public int getFinished() {
    return finished;
}

public void setFinished(int finished) {
    this.finished = finished;
}

public int getStatus() {
    return status;
}

public void setStatus(int status) {
    this.status = status;
}

public int getRemark() {
    return Remark;
}

public void setRemark(int remark) {
    Remark = remark;
}
```