下载原理

对于Android来说，其下载器的原理非常简单，仅仅是I/O流的实现而已，只要了解I/O流就能够写得出，下面这个是一个简单java项目的下载代码：

```
try {
    // strUrl 下载的网络地址
    URL url = new URL(strUrl);
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setConnectTimeout(10 * 1000);
    conn.setRequestMethod("GET");
    int code = conn.getResponseCode();
    if (code == 200) {
        System.out.println("下载开始");
        File file = new File("e:\" + strUrl.substring((strUrl.lastIndexOf("/") +
1)));
            long length = conn.getContentLength();
            if (length > 1024) {
                long size = length / (1024 * 1024);
                System.out.println("下载大小" + size + "mb");
            }
            InputStream is = conn.getInputStream();
            byte[] bt = new byte[1024];
            int len = 0;
            RandomAccessFile raf = new RandomAccessFile(file, "rwd");
            raf.setLength(length);
            while ((len = is.read(bt)) != -1) {
                raf.write(bt, 0, len);
            }
            System.out.println("下载完成");
            is.close();
            raf.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
```

代码很简单，用支持http协议的网络地址进行下载，然后使用I/O流下载，或许大家不熟悉的只有RandomAccessFile这个API了，这是一个支持任意位置下载的一个API，同时它有个setLength()方法，可以直接设置RandomAccessFile文件，的长度。还有个seek()方法，可以直接设定从文件的哪个位置开始写入文件。

RandomAccessFile是个很重要的API，对于断点下载而言。

直接下载可以了，那么如何断点下载呢?

所谓断点下载，就是在停止下载文件的时候记住停止时的下载位置，等下次继续下载的时候从这个位置继续下载。

这个时候我们只需设置一个停止位置，然后用RandomAccessFile的seek()方法读取这个位置就可以了

所以这时我们要分两步走

1. 初始化下载线程，获取文件的信息，如文件的大小等
2. 开始下载文件，如果文件信息已存在，则查询先前下载到哪一个位置。

代码断点续传代码如下：

```
private static void mutilDownload(String path) {
    HttpURLConnection conn = null;
    try {
        URL url = new URL(path);
        conn = (HttpURLConnection) url.openConnection();
        conn.setConnectTimeout(10 * 1000);
        conn.setRequestMethod("GET");
        conn.setReadTimeout(5 * 1000);
        int code = conn.getResponseCode();
        if (code == HttpURLConnection.HTTP_OK) {
            File file = new File("e:\\" + path.substring(path.lastIndexOf("/") +
1));
            long filelength = conn.getContentLength();
            RandomAccessFile randomFile = new RandomAccessFile(file, "rwd");
            randomFile.setLength(filelength);
            randomFile.close();
            long endposition = filelength;

            new newThreadDown(path, endposition).start();
        }

    } catch (Exception e) {

    } finally {
        conn.disconnect();
    }
}

public static class newThreadDown extends Thread {
    private String urlstr;
    private long lastPostion;
    private long endposition;
    public newThreadDown(String urlstr, long endposition) {
        this.urlstr = urlstr;
        this.endposition = endposition;
    }

    @Override
    public void run() {
        HttpURLConnection conn = null;
        try {
            URL url = new URL(urlstr);
            conn = (HttpURLConnection) url.openConnection();
            conn.setConnectTimeout(10 * 1000);
            conn.setRequestMethod("GET");
            conn.setReadTimeout(10 * 1000);
            long startposition = 0;
            // 创建记录缓存文件
            File tempfile = new File("e:\\" + 1 + ".txt");
            if (tempfile.exists()) {
                InputStreamReader isr = new InputStreamReader(new
FileInputStream(tempfile));
                BufferedReader br = new BufferedReader(isr);
                String lastStr = br.readLine();
                lastPostion = Integer.parseInt(lastStr);
```

```
                conn.setRequestProperty("Range", "bytes=" + lastPostion + "-" +
endposition);
                br.close();
            } else {
                lastPostion = startposition;
                conn.setRequestProperty("Range", "bytes=" + lastPostion + "-" +
endposition);
            }

            if (conn.getResponseCode() == HttpURLConnection.HTTP_PARTIAL) {
                System.out.println(206 + "请求成功");
                InputStream is = conn.getInputStream();
                RandomAccessFile accessFile = new RandomAccessFile(new
File("e:\\" + path.substring(path.lastIndexOf("/") + 1)),
                        "rwd");
                accessFile.seek(lastPostion);
                System.out.println("开始位置" + lastPostion);
                byte[] bt = new byte[1024 * 200];
                int len = 0;
                long total = 0;
                while ((len = is.read(bt)) != -1) {
                    total += len;
                    accessFile.write(bt, 0, len);
                    long currentposition = startposition + total;
                    File cachefile = new File("e:\\" + 1 + ".txt");
                    RandomAccessFile rf = new RandomAccessFile(cachefile,
"rwd");
                    rf.write(String.valueOf(currentposition).getBytes());
                    rf.close();
                }
                System.out.println("下载完毕");
                is.close();
                accessFile.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        super.run();
    }
}
```

这些都是java项目，可以在eclipse中直接运行测试。

下载的原理已经梳理清楚了，剩下的只要把下载程序移植到Android项目中去就好了。

Android下载器实现

作为一个实战项目，我们要尽可能的完善，尽可能的使用Android中的控件，所以我们不做自己将上面的代码复制到项目中，然后用ProgressBar更新UI的事情，我们要尽可能的复制！

我们的口号是：不做简单活！

知识要点

- Android四大组件之Service
- Android四大组件之Broadcast
- 数据存储SQLiteDatabase
  现在让我们开始完善这个单线程的下载器吧

下载器的布局

做一个简单的界面，我们用到开始下载按键，停止下载按键，一个Progressbar，以及一个TextView显示文件名。

如此简单的布局就不写代码了，详情可以下载我的Github项目研究。

封装实体对象

在本项目中有个两个实体类对象，即FileInfo类和ThreadInfo类，他们之中的变量都拥有get和set方法，FileInfo类需要实现序列化，详细代码请查看项目地址FileInfo类代码（略）：

```
public class FileInfo implements Serializable {
    private int id;
    private String url;
    private String fileName;
    private int length;
    private int finished;
    public FileInfo() {
        super();
    }
    /**
 *
 * @param id    文件的ID
 * @param url   文件的下載地址
 * @param fileName   文件的名字
 * @param length   文件的總大小
 * @param finished   文件已經完成了多少
 */
    public FileInfo(int id, String url, String fileName, int length, int
finished) {
        super();
        this.id = id;
        this.url = url;
        this.fileName = fileName;
        this.length = length;
        this.finished = finished;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
}
```

ThreadInfo类代码（略）：

```
public class ThreadInfo {
```

```
    private int id;
    private String url;
    private int start;
    private int end;
    private int finished;
    public ThreadInfo() {
        super();
    }
    /**
 * @param id 綫程的ID
 * @param url 下載文件的網絡地址
 * @param start 綫程下載的開始位置
 * @param end 綫程下載的結束位置
 * @param finished  綫程已經下載到哪個位置
 */
    public ThreadInfo(int id, String url, int start, int end, int finished) {
        super();
        this.id = id;
        this.url = url;
        this.start = start;
        this.end = end;
        this.finished = finished;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
```

创建数据库

使用SQLiteDatabase，我们首先要实现一个数据库的帮助类，然后创建一个操作数据库的接口类，最后实现这个接口的数据库操作类。

使用数据库是用于保存ThreadInfo对象的信息，并且实时更新下载进度，但需要断点续传的时候从数据库中取出保存的信息，继续下载。

这里提示一下，保存断点信息可以不使用数据库，试用SharedPreference也是可以起到同样的作用，具体方法请读着自己摸索。

数据库帮助类代码如下：

```
public class DBHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "download.db";
    private static final int VERSION = 1;
    private static final String SQL_CREATE = "create table thread_info(_id
integer primary key autoincrement, "
            + "thread_id integer, url text, start integer, end integer, finished
integer)";
    private static final String SQL_DROP = "drop table if exists thread_info";
    public DBHelper(Context context) {
        super(context, DB_NAME, null, VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE);
```

```
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL(SQL_DROP);
        db.execSQL(SQL_CREATE);
    }
}
```

操作数据库的接口类代码：

```
public interface ThreadDAO {
    // 插入綫程
    public void insertThread(ThreadInfo info);
    // 删除綫程
    public void deleteThread(String url, int thread_id);
    // 更新綫程
    public void updateThread(String url, int thread_id, int finished);
    // 查詢綫程
    public List<ThreadInfo> queryThreads(String url);
    // 判斷綫程是否存在
    public Boolean isExists(String url, int threadId);
}
```

实现接口的数据库工具类：

```
public class ThreadDAOImple implements ThreadDAO {
    private DBHelper dbHelper = null;
    public ThreadDAOImple(Context context) {
        super();
        this.dbHelper = new DBHelper(context);
    }
    // 插入綫程
    @Override
    public void insertThread(ThreadInfo info) {
        SQLiteDatabase db = dbHelper.getReadableDatabase();
        ContentValues values = new ContentValues();
        values.put("thread_id", info.getId());
        values.put("url", info.getUrl());
        values.put("start", info.getStart());
        values.put("end", info.getEnd());
        values.put("finished", info.getFinished());
        db.insert("thread_info", null, values);
        db.close();
    }
    // 删除綫程
    @Override
    public void deleteThread(String url, int thread_id) {
        SQLiteDatabase db = dbHelper.getReadableDatabase();
        db.delete("thread_info", "url = ? and thread_id = ?", new String[] {
            url, thread_id + ""
        }
        );
        db.close();
    }
    // 更新綫程
    @Override
```

```
    public void updateThread(String url, int thread_id, int finished) {
        SQLiteDatabase db = dbHelper.getReadableDatabase();
        db.execSQL("update thread_info set finished = ? where url = ? and
thread_id = ?",
                    new Object[]{
            finished, url, thread_id
        }
        );
        db.close();
    }
    // 查詢綫程
    @Override
    public List<ThreadInfo> queryThreads(String url) {
        SQLiteDatabase db = dbHelper.getReadableDatabase();
        List<ThreadInfo> list = new ArrayList<ThreadInfo>();
        Cursor cursor = db.query("thread_info", null, "url = ?", new String[] {
url }, null, null, null);
        while (cursor.moveToNext()) {
            ThreadInfo thread = new ThreadInfo();
            thread.setId(cursor.getint(cursor.getColumnIndex("thread_id")));
            thread.setUrl(cursor.getString(cursor.getColumnIndex("url")));
            thread.setStart(cursor.getint(cursor.getColumnIndex("start")));
            thread.setEnd(cursor.getint(cursor.getColumnIndex("end")));

thread.setFinished(cursor.getint(cursor.getColumnIndex("finished")));
            list.add(thread);
        }
        cursor.close();
        db.close();
        return list;
    }
    // 判斷綫程是否爲空
    @Override
    public Boolean isExists(String url, int thread_id) {
        SQLiteDatabase db = dbHelper.getReadableDatabase();
        Cursor cursor = db.query("thread_info", null, "url = ? and thread_id =
?", new String[] { url, thread_id + "" },
                    null, null, null);
        Boolean exists = cursor.moveToNext();
        db.close();
        db.close();
        return exists;
    }
}
```

从Activity中向Service传参

很好，前期的准备工作已经做好了，需要从Activity中启动线程，并且将Activity中获得的关于下载文件的信息传递到Service中去，我们只需要用Intent便可以将FileInfo对象传递过去。在这里要注意的是如果FileInfo没有序列化，继承Serializable接口，那么Intent无法将FileInfo对象传送出去。

首先创建一个DownloadService服务类，继承自Service，定义ACITON_START和ACTION_STOP两个常量，重新onStartCommand方法，代码如下：

```
public class DownloadService extends Service {
    public static final String ACTION_START = "ACTION_START";
    public static final String ACTION_STOP = "ACTION_STOP";
```

```
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // 获得Activity穿来的参数
        if (ACTION_START.equals(intent.getAction())) {
            FileInfo fileInfo = (FileInfo)
intent.getSerializableExtra("fileInfo");
            Log.i("test", "START" + fileInfo.toString());
        } else if (ACTION_STOP.equals(intent.getAction())) {
            FileInfo fileInfo = (FileInfo)
intent.getSerializableExtra("fileInfo");
        }
        return super.onStartCommand(intent, flags, startId);
    }
}
```

然后修改MainActivity中的代码：

- 定义Intent常量
- 定义FileInfo常量
- 在onCreate方法中初始化两个常量：

```
fileInfo = new FileInfo(0, urlstr, getfileName(urlstr), 0, 0);
intent = new Intent(MainActivity.this, DownloadService.class);
```

设置点击事件：

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
    case R.id.start_button:
        // 开启服务
        fileName.setText(getfileName(urlstr));
        intent.setAction(DownloadService.ACTION_START);
        intent.putExtra("fileInfo", fileInfo);
        startService(intent);

        break;
    case R.id.stop_button:
        intent.setAction(DownloadService.ACTION_STOP);
        intent.putExtra("fileInfo", fileInfo);
        startService(intent);
        break;
    }
}
```

相亲我们已经可以启动服务了，点击按键启动服务之后，就能调用DownloadService中的onStartCommand方法，接收到从MainActivity传过来的fileInfo对象。

从DownloadService中初始化线程

在刚才从MainActivity传过来的fileInfo对象中只有下载的URL地址以及文件名，但是我们还不知道文件的长度，也没有设定好文件的保存位置等信息，初始化线程就是为了配置好这些信息。

从初始化线程中配置好fileInfo对象之后，需要将它传递给Handler，然后在Handler启动真正的下载任务，Handler代码如下：

```
// 從InitThread綫程中獲取FileInfo信息，然後開始下載任務
Handler mHandler = new Handler() {
    public void handleMessage(android.os.Message msg) {
        switch (msg.what) {
        case MSG_INIT:
            FileInfo fileInfo = (FileInfo) msg.obj;
            Log.i("test", "INIT:" + fileInfo.toString());
            // 獲取FileInfo對象，開始下載任務
            mTask = new DownloadTask(DownloadService.this, fileInfo);
            mTask.download();
            break;
        }
    };
};
```

InitThread内部类在完成初始化线程之后，将fileInfo传递给Handler，代码如下：

```
// 初始化下載綫程，獲得下載文件的信息
class InitThread extends Thread {
    private FileInfo mFileInfo = null;

    public InitThread(FileInfo mFileInfo) {
        super();
        this.mFileInfo = mFileInfo;
    }

    @Override
    public void run() {
        HttpURLConnection conn = null;
        RandomAccessFile raf = null;
        try {
            URL url = new URL(mFileInfo.getUrl());
            conn = (HttpURLConnection) url.openConnection();
            conn.setConnectTimeout(5 * 1000);
            conn.setRequestMethod("GET");
            int code = conn.getResponseCode();
            int length = -1;
            if (code == HttpURLConnection.HTTP_OK) {
                length = conn.getContentLength();
            }
            //如果文件长度为小于0，表示获取文件失败，直接返回
            if (length <= 0) {
                return;
            }
            // 判斷文件路徑是否存在，不存在這創建
            File dir = new File(DownloadPath);
            if (!dir.exists()) {
                dir.mkdir();
```

```
            }
            // 創建本地文件
            File file = new File(dir, mFileInfo.getFileName());
            raf = new RandomAccessFile(file, "rwd");
            raf.setLength(length);
            // 設置文件長度
            mFileInfo.setLength(length);
            // 將FileInfo對象傳遞給Handler
            Message msg = Message.obtain();
            msg.obj = mFileInfo;
            msg.what = MSG_INIT;
            mHandler.sendMessage(msg);
            msg.setTarget(mHandler);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            conn.disconnect();
            try {
                raf.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        super.run();
    }
}
```

然后修改onStartConnand方法，在点击开启服务的时候初始化线程

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    // 获得Activity穿来的参数
    if (ACTION_START.equals(intent.getAction())) {
        FileInfo fileInfo = (FileInfo) intent.getSerializableExtra("fileInfo");
        Log.i("test", "START" + fileInfo.toString());
        new InitThread(fileInfo).start();
    } else if (ACTION_STOP.equals(intent.getAction())) {
        FileInfo fileInfo = (FileInfo) intent.getSerializableExtra("fileInfo");
        Log.i("test", "STOP" + fileInfo.toString());
    }
    return super.onStartCommand(intent, flags, startId);
}
```

开启下载任务

终于轮到真正的下载任务了，这就是最后一步了。

在之前的代码中，即使用Handler接收InitThread中传递过来的fileInfo对象时，有一段代码还没有实现，这段代码是正在的下载逻辑：

```
// 獲取FileInfo對象，開始下載任務
mTask = new DownloadTask(DownloadService.this, fileInfo);
mTask.download();
```

现在我们开始实现DownloadTask下载任务这个类吧。

DownloadTask类有以下成员变量：

```
private Context mComtext = null;
private FileInfo mFileInfo = null;
private ThreadDAO mDao = null;
private int mFinished = 0;
public Boolean mIsPause = false;
```

mComtext就不做介绍了，mFileInfo是封装了下载文件的信息对象；

mDAO是对数据库进行操作的工具类，它将会引用实现了它的接口的ThreadDAOImple类。

mFinished用于临时存储文件下载的进度。

mIsPause则用于判断文件是否在下载状态又或者停止状态。

设定好成员变量，在创建DownloadTask的构造函数，将成员变量初始化

```
public DownloadTask(Context comtext, FileInfo fileInfo) {
    super();
    this.mComtext = comtext;
    this.mFileInfo = fileInfo;
    this.mDao = new ThreadDAOImple(mComtext);
}
```

下面开始的便是下载线程的代码实现，将之前的代码原理搬过来，改一改就好了，这里还是展示给大家看吧，代码如下：

```
class DownloadThread extends Thread {
    private ThreadInfo threadInfo = null;

    public DownloadThread(ThreadInfo threadInfo) {
        super();
        this.threadInfo = threadInfo;
    }

    @Override
    public void run() {
        // 如果數據庫不存在下載信息，添加下載信息
        if (!mDao.isExists(threadInfo.getUrl(), threadInfo.getId())) {
            mDao.insertThread(threadInfo);
        }
        HttpURLConnection conn = null;
        RandomAccessFile raf = null;
        InputStream is = null;
        try {
            URL url = new URL(mFileInfo.getUrl());
            conn = (HttpURLConnection) url.openConnection();
            conn.setConnectTimeout(5 * 1000);
            conn.setRequestMethod("GET");


            int start = threadInfo.getStart() + threadInfo.getFinished();
            // 設置下載文件開始到結束的位置
            conn.setRequestProperty("Range", "bytes=" + start + "-" +
threadInfo.getEnd());
```

```
            File file = new File(DownloadService.DownloadPath,
mFileInfo.getFileName());
            raf = new RandomAccessFile(file, "rwd");
            raf.seek(start);
            mFinished += threadInfo.getFinished();
            int code = conn.getResponseCode();
            if (code == HttpURLConnection.HTTP_PARTIAL) {
                is = conn.getInputStream();
                byte[] bt = new byte[1024];
                int len = -1;
                // 定义UI刷新时间
                long time = System.currentTimeMillis();
                while ((len = is.read(bt)) != -1) {
                    raf.write(bt, 0, len);
                    mFinished += len;
                    // 設置爲500毫米更新一次
                    if (System.currentTimeMillis() - time > 500) {
                        time = System.currentTimeMillis();

                        Intent intent = new
Intent(DownloadService.ACTION_UPDATE);
                        intent.putExtra("finished", mFinished * 100 /
mFileInfo.getLength());
                        Log.i("test", mFinished * 100 / mFileInfo.getLength() +
"");
                        // 發送廣播給Activity
                        mComtext.sendBroadcast(intent);
                    }
                    if (mIsPause) {
                        mDao.updateThread(threadInfo.getUrl(),
threadInfo.getId(), mFinished);
                        return;
                    }
                }
            }
            // 下載完成后，删除數據庫信息
            mDao.deleteThread(threadInfo.getUrl(), threadInfo.getId());

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            conn.disconnect();
            try {
                is.close();
                raf.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        super.run();
    }
}
```

在这段代码中我们还要在下载的时候发送广播给Activity，用于更新Progressbar的进度，但更新不易过频，以免影响UI效果，所以设置为每500毫米更新一下，根据系统时间设定。

我们在下载逻辑中，还要判断当前下载的文件是否在数据库中存在，如果不存在就添加，如果存在就要从数据库中获取当前下载位置，然后继续下载，所以增加以下方法：

```
public void download(){
    // 從數據庫中獲取到下載的信息
    List<ThreadInfo> list = mDao.queryThreads(mFileInfo.getUrl());
    ThreadInfo info = null;
    if (list.size() == 0) {
        info = new ThreadInfo(0, mFileInfo.getUrl(), 0, mFileInfo.getLength(),
0);
    }else{
        info= list.get(0);
    }
    new DownloadThread(info).start();
}
```

好了，整个的下载任务类已经完成了，下面我们继续完善我们的代码吧。

完善Service和MainActivity代码

在之前，我们虽然把初始化下载线程InitThread写好了，然后通过初始化线程获取FileInfo对象，将其传递给Handler，在Handler中开启真正的下载任务。但是当时并没有调用这个InitThread类，现在再次修改DownloadService中的onStartCommand方法来启动InitThread任务吧

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    // 获得Activity穿来的参数
    if (ACTION_START.equals(intent.getAction())) {
        FileInfo fileInfo = (FileInfo) intent.getSerializableExtra("fileInfo");
        Log.i("test", "START" + fileInfo.toString());
        new InitThread(fileInfo).start();
    } else if (ACTION_STOP.equals(intent.getAction())) {
        FileInfo fileInfo = (FileInfo) intent.getSerializableExtra("fileInfo");
        Log.i("test", "STOP" + fileInfo.toString());
        if (mTask != null) {
            mTask.mIsPause = true;
        }
    }
    return super.onStartCommand(intent, flags, startId);
}
```

我们通过MainActivity的点击事件开启服务，如果Intent中传过来的值为ACTION_START的时候，开启初始化线程，获得FileInfo对象，然后将其传递给Handler启动下载任务。

如果MainActivity传递过来的值为ACTION_STOP，就判断当前是否有下载任务，如果有下载任务，就将DownloadTask中的成员变量mIsPause设置为true，这时就更新数据库中的下载进度了。

然后我们在修改MainActivity中代码，添加一个广播接收者的内部类，它接收从DownloadTask中传过来的广播--下载进度，然后实时更新ProgressBar。代码如下：

```
// 從DownloadTadk中獲取廣播信息，更新進度條
class UIRecive extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        if (DownloadService.ACTION_UPDATE.equals(intent.getAction())) {
            int finished = intent.getIntExtra("finished", 0);
            downloadProgress.setProgress(finished);

        }
    }
}
```

这是一个自己手动撰写的广播，因此需要动态注册，在MainActivity中的创建一个成员变量广播接收者对象mRecive，在onCreate方法注册广播接收者：

```
// 從DownloadTadk中獲取廣播信息，更新進度條
class UIRecive extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (DownloadService.ACTION_UPDATE.equals(intent.getAction())) {
            int finished = intent.getIntExtra("finished", 0);
            downloadProgress.setProgress(finished);
        }
    }
}
```

最后不要忘记在Activity的onDestry方法中注销广播！

最后的最后就是千万记得在Androidmainfest中获取网络、存储卡读取/写入权限等等。

总结

这是一个单线程的断点续传下载，也仅仅是一个下载DEMO，但是它包含了Android中各种组件的混合使用，对于Android新手全面的了解Android项目很有好处。

但是要注意的是这个项目仍然有许多BUG等着大家自己去修复，如在下载的时候再次点击下载，你会发现又多了一个新的下载线程，导致进度条跳来跳去。解决这个问题也简单，只需要在开启之前判断一下是否已经有这个文件了，如果有就直接跳。

其次还有个bug，那就是在本项目中存储进度的数据类型是int类型，如果你说下载的文件过大，如超过30M的时候，你会发现你的进度条下载到一半就消失了。这是因为下载数据超过int的数据范围，导致内存泄漏。这个问题只需要将数据类型修改为long类型就好了。