19 декабря 2009 в 18:45

Как реализовать загрузку изображений в список в отдельном потоке на Android

Android*

По просьбам трудящихся, статья о методе загрузки изображений в список в отдельном потоке на Android.

Задача:

Реализовать механизм загрузки изображений из Интернета и отображения их в списке. При этом загрузка изображений должна быть реализована в отдельном потоке, во избежания «зависания» UI приложения.

Реализация:

Для реализации поставленной задачи использованы стандартный виджет ListView и адаптер — ArrayAdapter. Для работы с изображениями создан helper-класс ImageManager, который имеет два метода downloadImage() и fetchImage(). Первый загружает изображений из Интернета. Второй — вызывает загрузку изображений в отдельном потоке и устанавливает результат в ImageView.

Пример испольования:

Реализацию поставленной задачи будем расматривать на примере моего проекта. И на его код я буду ссылаться в статье.
Исходники: fileshare.in.ua/3053597
APK: fileshare.in.ua/3053596

Описание реализации:

Давайте рассмотрим поподробнее каждый из методов ImageManager'а:

```
1.  package com.rudenko.android.ListIconFetching;
2.
3.  import java.io.BufferedInputStream;
4.  import java.io.IOException;
5.  import java.net.HttpURLConnection;
6.  import java.net.MalformedURLException;
7.  import java.net.URL;
8.
9.  import android.graphics.Bitmap;
10. import android.graphics.BitmapFactory;
11. import android.os.Handler;
12. import android.os.Message;
13. import android.util.Log;
14. import android.widget.ImageView;
15.
16. public class ImageManager {
17.   private final static String TAG = "ImageManager";
18.
19.   /** Private constructor prevents instantiation from other classes */
20.   private ImageManager () {}
21.
22.   public static void fetchImage(final String iUrl, final ImageView iView) {
23.     if ( iUrl == null || iView == null )
24.       return;
25.
26.     final Handler handler = new Handler() {
27.       @Override
28.       public void handleMessage(Message message) {
29.         final Bitmap image = (Bitmap) message.obj;
30.         iView.setImageBitmap(image);
31.       }
32.     };
33.
34.     final Thread thread = new Thread() {
35.       @Override
36.       public void run() {
37.         final Bitmap image = downloadImage(iUrl);
38.         if ( image != null ) {
39.           Log.v(TAG, "Got image by URL: " + iUrl);
40.           final Message message = handler.obtainMessage(1, image);
41.           handler.sendMessage(message);
42.         }
```

```
43.         }
44.       };
45.       iView.setImageResource(R.drawable.icon);
46.       thread.setPriority(3);
47.       thread.start();
48.     }
49.
50.     public static Bitmap downloadImage(String iUrl) {
51.       Bitmap bitmap = null;
52.       HttpURLConnection conn = null;
53.       BufferedInputStream buf_stream = null;
54.       try {
55.         Log.v(TAG, "Starting loading image by URL: " + iUrl);
56.         conn = (HttpURLConnection) new URL(iUrl).openConnection();
57.         conn.setDoInput(true);
58.         conn.setRequestProperty("Connection", "Keep-Alive");
59.         conn.connect();
60.         buf_stream = new BufferedInputStream(conn.getInputStream(), 8192);
61.         bitmap = BitmapFactory.decodeStream(buf_stream);
62.         buf_stream.close();
63.         conn.disconnect();
64.         buf_stream = null;
65.         conn = null;
66.       } catch (MalformedURLException ex) {
67.         Log.e(TAG, "Url parsing was failed: " + iUrl);
68.       } catch (IOException ex) {
69.         Log.d(TAG, iUrl + " does not exists");
70.       } catch (OutOfMemoryError e) {
71.         Log.w(TAG, "Out of memory!!!");
72.         return null;
73.       } finally {
74.         if ( buf_stream != null )
75.           try { buf_stream.close(); } catch (IOException ex) {}
76.         if ( conn != null )
77.           conn.disconnect();
78.       }
79.       return bitmap;
80.     }
81.   }
 * This source code was highlighted with Source Code Highlighter.
```

Метод fetchImage():

```
public static void fetchImage(final String iUrl, final ImageView iView);
```

Входные параметры:

iUrl — URL к изображению для загрузки
iView — ссылка на виджет ImageView, которому будет назначено изображение после загрузки.
Оба параметра являются обязательными.

Результат:

Функция не возвращает ничего.

Краткое описание:

Функция создает поток для загрузки изображения. На время загрузки во входной ImageView устанавливается стандартное изображение. После завершения загрузки, изображение входного ImageView обновляется загруженным.

Несколько слов о потоке: в строке 46 происходит понижение приоритета потока. Это сделано для того, что бы данный поток не забирал ресурсы необходимые для корректной работы приложения.

Метод downloadImage():

```
public static Bitmap downloadImage(String iUrl);
```

Входные параметры:

iUrl — URL к изображению для загрузки

Результат:

Изображение загруженное из интернета, либо Null — если операция не была выполнена успешно.

Краткое описание:

В функции создается соединение с сервером, где находится изображение. Происходит получение входного потока, который далее передается в BitmapFactory для создания изображения.

Как использовать ImageManager:

Рассмотрим на примере к статье. В методе FetchImageAdapter.getView(), для подгрузки изображений в ImageView строки списка, используется следующая строка:

```
ImageManager.fetchImage(android.image, holder.ib_logo);
```

где android.image — URL к изображению, а holder.ib_logo — ImageView строки списка.

Заключение:

Данный механизм подходит для загрузки изображения из Интернета в параллельном потоке, для любого виджета ImageView Android. Т. е. данный механизм можно использовать не только для конкретно поставленной задачи.

P. S. пользуйтесь на здоровья.

android, development, images, listview

Похожие публикации

Портирование любимой игры под Android 20 августа в 17:01

25+ видеоуроков по Android для начинающих 21 июля в 13:29

Android Development Tutorial. Часть 2/? 8 марта 2011 в 23:20

Android Development Tutorial. Часть 1/? 7 марта 2011 в 00:03

В Украине начался HTC Android Developers Contest 2.0 25 января 2011 в 15:44

Стань судьей Android Developer Challange 2. На Android Market появилось программа для судей-волонтеров 26 сентября 2009 в 14:22

Российские программисты одни из победителей Android Developer Challenge 10 сентября 2008 в 09:22

Победители Android Developer Challenge 31 августа 2008 в 07:29

Закончился первый этап Android Developers Challenge 22 апреля 2008 в 19:24

SDK, Android Developer Challenge, сообщество, блог, видеоканал 12 ноября 2007 в 19:50

Комментарии (13)

meesix, 19 декабря 2009 в 19:49 # +2

Спасибо за вторую уже публикацию о разработке под Андроид :) очень интересно, надеюсь что вы будете продолжать дальше.

Boiler, 20 декабря 2009 в 02:09 # +1

Почему вы не используете класс AsyncTask для разделения потоков?

rude, 20 декабря 2009 в 13:09 # ↑ 0

использовать потоки напрямую — привычка из C++

i_home, 20 декабря 2009 в 05:19 # 0

А что произойдет если в момент загрузки пользователь повернет девайс, (сменит ориентацию экрана)?

rude, 20 декабря 2009 в 13:11 # ↑ 0

в моем примере не произойдет ничего, т. к. ориентация задана постоянная — portrait.
для случая когда нужно что бы работал авто-поворот (sensor), добавляем кеширование изображений и используем изображения повторно, из памяти ;)

troorl1985, 20 декабря 2009 в 14:11 # 0

Спасибо. То, что доктор прописал.

Dyn, 22 декабря 2009 в 14:53 # 0

Понижение приоритета потока интересная мысль, надо будет воспользоваться :)

Исходники скачать не получается, поэтому поинтересуюсь тут:
а при быстром скроллинге картинки в один View загружаться не будут?

Dyn, 22 декабря 2009 в 15:02 # ↑ 0

скачал сорцы:
Вот в чем вопрос заключался:
при прокрутке метод getView у FetchImageAdapter будет вызывать некоторое количество раз.
При этом объект ViewHolder, передаваемый в качестве параметра, может быть одинаковым для разных элементов коллекции. тут все ок (для оптимизации).
И это означает, что в один и тот же объект с разной периодичностью, будет из отдельных потоков загружаться некоторое количество картинок, принадлежащих разным объектам. Последствия думаю очевидны.

зы. Поправьте меня, если где ошибся.

rude, 22 декабря 2009 в 15:05 # ↑ 0

вы не ошибаетесь :)
смотрите ответ ниже

rude, 22 декабря 2009 в 15:04 # ↑ 0

в данном примере будут, если список будет достаточно длинный. но этого можно избежать, созданием HashMap ключами которого будут ImageView а значениями, — URL. тогда, если пришел запрос на новое изображения для уже существующего в HashMap ImageView, обновляем значение, а изображение по пердыдущему URL не присваевать ImageView. Надеюсь понятно изложил.
Опять-таки, советую создать кеш скаченных изображений, что бы не скачивать их снова. Но это уже тема для другого поста ;)

Dyn, 22 декабря 2009 в 15:14 # ↑ 0

Идея более-менее понятна, хотя на мой вкус проще поток обрывать.
Иначе при прокрутке память потоками быстро засрется.

Кеширование картинок в памяти тоже легко может упереться на OutOfMemory

зы. отталкиваюсь от опыта своего приложения, где средняя длина списка около 200, а размер картинки около 150x150

rude, 22 декабря 2009 в 15:20 # ↑ 0

1) кешируйте и складывайте картинки на диск, и делайте recycle() ненужным, тогда OutOfMemory не будет. а если в кеше нет изображения, востанавливаем его с диска.
2) обрывать поток означает то, что нужно будет создавать его опять, если пользователь будет скролить список вверх (назад), что не есть хорошо, ИМХО.

tot_ra, 18 октября 2010 в 13:27 # 0

Спасибо, пробовал до этого сделать как в статье Tom van Zummeren'a
blog.jteam.nl/2009/09/17/exploring-the-world-of-android-part-2/

но что-то у меня с руками и потоками и не сложилось, хотя он и делал кеширование. Самое главное что шаблон отрисовывается а картинки потом подгружаются