7 апреля в 19:14

Экономим память: Picasso vs UniversalImageLoader

из песочницы

Разработка под Android*

Привет, Android-разработчики!
Я думаю, каждый из нас сталкивается с загрузкой изображений по URL. Самый простой способ решения этой задачи: использовать готовую стороннюю библиотеку. Как правило, одним из таких готовых решений оказывается Universal Image Loader (UIL), Picasso. Когда я спрашиваю у разработчика, почему он выбрал ту или иную библиотеку, то, как правило, получаю разные ответы. Например, «у Picasso/UIL нет проблем с memory leaks», или «Square делают только правильные вещи», или просто «Да вот использую UIL, работает – и хорошо».
Так вот, мне стало интересно: какая из этих 2-х библиотек оптимально использует память? Я использую UIL и имею проблему с OutOfMemory на старых устройствах. Возможно, Picasso это лекарство?
Так появилась идея этого benchmark-а.

Цель тестирования: определить, какая из библиотек (UIL или Picasso) минимально использует память устройства.

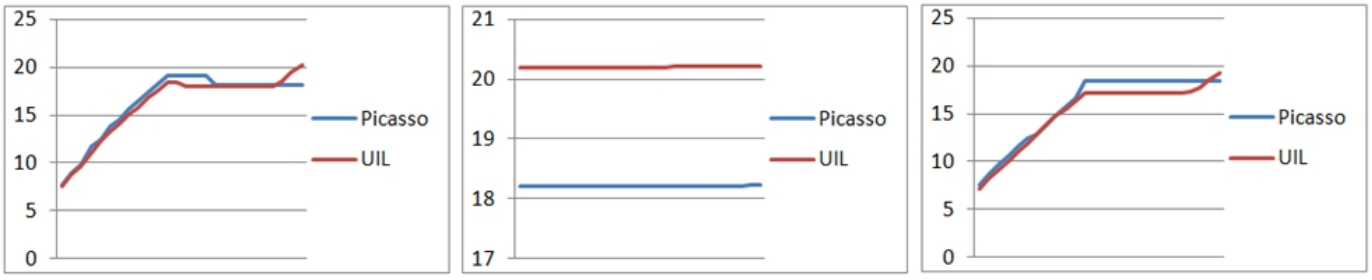
Тест кейсы:
— Загрузка маленьких изображений (240x240)
— Загрузка больших изображений (>400px по любому из габаритов)
— Загрузка больших изображений и преобразование их размера к габаритам ImageView
— Загрузка маленьких изображений и их показ в виде круглой картинки
— Загрузка больших изображений и показ их в конфигурации RGB565

Методика выполнения теста:
В качестве списка используем GridView шириной в 2 столбца. Адаптер настраивается отдельно под каждый тест кейс. В адаптер отдаем список заранее подготовленных URL, создавая, таким образом, одинаковые условия тестирования.
С периодом в 1 сек, список автоматически делает один проход вниз, а потом вверх с шагом в 4 изображения. По каждому шагу производится измерение памяти, использованной приложением.
Измеряем использованную память в 3 этапа для каждого тест кейса:
— первый запуск — с чистым кешем приложения;
— второй запуск: не закрывая приложение после первого прохода;
— третий запуск – после повторного открытия приложения без чистки кеша.
По окончанию выполнения тест кейса, я дополнительно записывал размер кеша, что тоже немаловажно для старых устройств.

Исходники Benchmark-а можно найти по ссылке
github.com/artemmanaenko/ImageLoadersTest. Проект собран под Gradle.

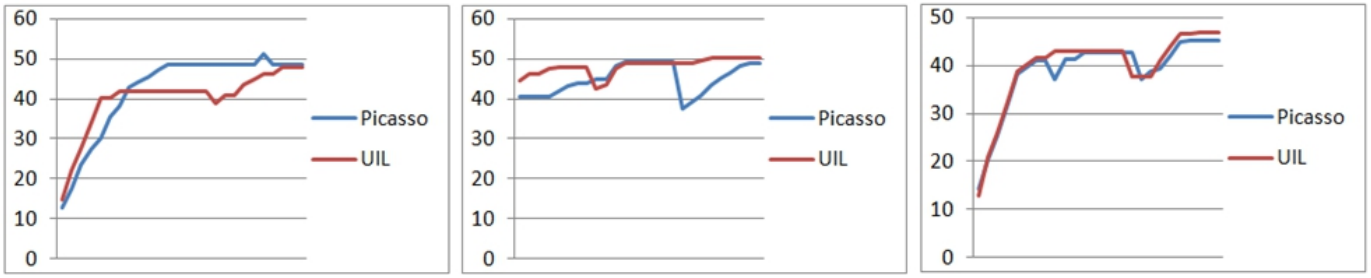
Итак, ниже результаты по каждому тест кейсу. Ось Y – используемая приложением память в Мб. Ось X – время проведения тест кейса.

Загрузка маленьких изображений



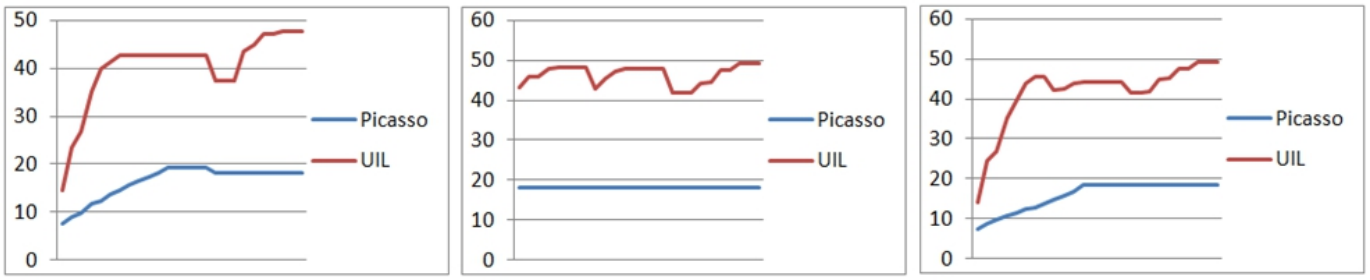
Размер кеша: Picasso=1.39 Мб, UIL=1.17 Мб

Загрузка больших изображений



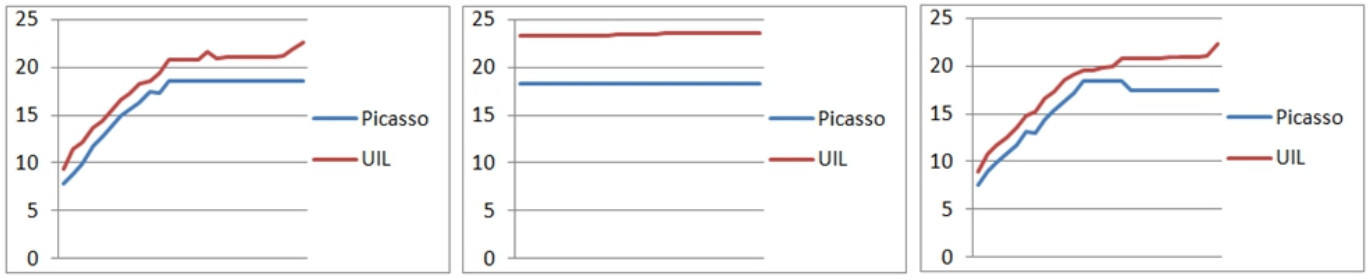
Размер кеша: Picasso=3,67 Мб, UIL=5,44 Мб

Загрузка больших изображений с преобразованием до размера ImageView



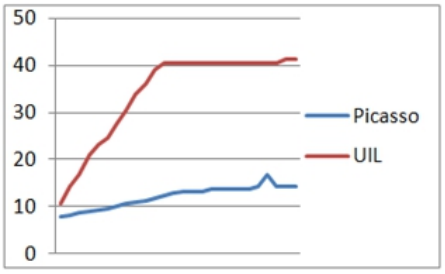
Размер кеша: Picasso=3,67 Мб, UIL=5,44 Мб

Загрузка маленьких изображений и их обрезка до круглой картинки

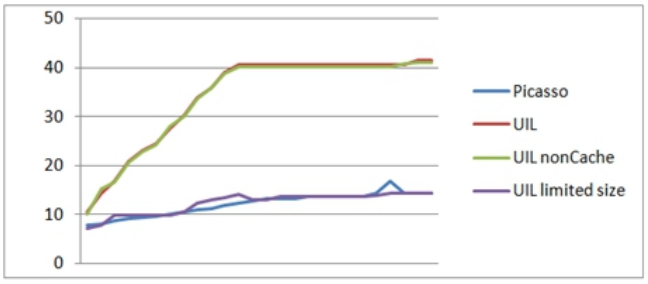


Размер кеша: Picasso=1.39 Мб, UIL=1.17 Мб

Загрузка больших изображений и показ их в конфигурации RGB565



Результаты экспериментов с большими картинками меня впечатлили, и я решил, что стоит попробовать настроить конфигурацию UIL. Чтобы не сильно загружать память кешем – я попробовал отключить у UIL кеш в RAM. И, как вариант, установить кешируемый габарит картинки – не более, чем в половину экрана.



На основе эксперимента я сделал следующие выводы:

- Если ваши списки работают с маленькими изображениями (сравнимыми с размером ImageView) – выбор библиотеки для вас не принципиален. Picasso создает чуть больший кеш на диске, при этом используя меньше RAM примерно на тот же размер.
- Picasso показала потрясающие результаты по управлению памятью, работая с большими изображениями. UIL, по всей видимости, хранит оригинал изображения в памяти. Picasso хранит уже преобразованный размер картинки. Потому и кеш на диске у Picasso значительно меньше.
- UIL может работать с той же эффективностью, что и Picasso, если его дополнительно настроить. Например, ограничить размер кеша в памяти. Или, как в одном из тестов – ограничить вручную размер кешируемых фотографий. Второй способ может быть непригоден для использования, поскольку устанавливает глобальную конфигурацию ImageLoader-а.
- Работа с круглыми аватарами обходится «дешевле» через Picasso. Но, опять же, за счет того, что я вручную вызывал recycle() у оригинального Bitmap-а. Такое же действие можно выполнить и в UIL, устанавливая переопределенный BitmapDisplayer.
- Picasso предельно проста в использовании и уже «с коробки» работает с памятью эффективно. Так выглядит инициализация и выполнение загрузки для библиотек:
  - Picasso
  - UIL
- Есть у Picasso и минус: трансформации изображений и приведение к RGB565 необходимо делать в самописных классах.
  - RoundTransformation

Config565Transformation

Для себя я сделал вывод: проекты необходимо переводить на Picasso. В моем случае это решит проблему с перерасходом памяти. Надеюсь, этот пост будет полезен и Вам!

Universal Image Loader, Picasso, android, android development, image loaders, разработка под android, загрузка изображений, память

Похожие публикации

DevConf 2014: Разработка под Android Wear (Google Glass, фитнес-трекеры, умные часы) 2 июня в 14:35

[ОПРОС] Какую ОС вы хотели бы использовать для разработки под Tizen? 26 мая в 15:03

Обновлены средства разработки под android, поддержка java 7 5 марта в 01:43

JavaScript to APK. Подводные камни разработки под Android для тех, кого задолбал PhoneGap. Построение мостов из Java в JavaScript 4 марта в 15:09

Разработка под Android в NetBeans IDE без плагинов. Часть 2 1 февраля в 05:25

Четыре метода загрузки изображений с веб-сайта с помощью Python 25 января в 00:24

Разработка под Android в NetBeans IDE без плагинов. Часть 1 5 января в 00:01

Пользователи Delphi о мобильной разработке под Android 9 декабря 2013 в 17:16

Про мой опыт разработки под Android или тренируемся на Крестиках-Ноликах 4 ноября 2013 в 19:48

Разработка под Blackberry 10. Первые впечатления 25 марта 2013 в 12:56

Комментарии (28)

отслеживать новые: в почте в трекере

izeberq, 8 апреля 2014 в 10:39 # +1

Отличное сравнение.
Для загрузки картинок под Android есть еще библиотека glide (https://github.com/bumptech/glide).
Интересно, насколько она эффективна по сравнению с Picasso.

recompileme, 8 апреля 2014 в 11:59 # +1

Рекомендую code.google.com/p/android-query/wiki/ImageLoading

Artem_Manaenko, 8 апреля 2014 в 12:19 # +1

Допишу на выходных тесты этих 2-х либ. Зафейворите репозиторий — выложу отчет туда.

bigfatbrowncat, 8 апреля 2014 в 15:17 # 0

А как у перечисленных библиотек с такими функциям/проблемами:

1. Картинка грузится с закрытого сервера, требуется авторизация, кроме того с запросом передаются специальные HTTP-заголовки, также как и с ответом — и всё это надо гибко обработать

2. Картинка грузится медленно, приложение должно быть возможно свернуть/восстановить (короче, есть ли там работа в режиме фонового service-а)?

Artem_Manaenko, 8 апреля 2014 в 15:38 (комментарий был изменён) # +1

1. С UIL-ом такая комбинация точно работает. В конфигурации можно установить кастомный ImageDownloader, который экстендит библиотечный. Там и можно настроить хедеры и выполнить авторизацию перед запросом. Я использую в одном проекте. А вот с Picasso — не уверен, нужно смотреть.
2. Нужно тестировать. Но, поскольку, библиотека использует пул потоков для загрузки — то скорее всего Threads никуда не деваются при сворачивании Activity. А если нужно остановить\восстановить запросы по onStart()/onPause() — обе библиотеки поддерживают такую возможность.

Mecid, 8 апреля 2014 в 17:04 (комментарий был изменён) # 0

Я использую UIL для загрузки картинок в твиттер клиенте Robird.
Каждый твит имеет как минимум 1 картинку — аватарку пользователя и может еще отображать превью картинок с разных сервисов.
UIL работает очень плавно и быстро подгружает картинки. Никаких поддергиваний при скроле. Вот моя конфигурация:

```
        Executor downloadExecutor = Executors.newFixedThreadPool(5);
        Executor cachedExecutor = Executors.newSingleThreadExecutor();
```

```
ActivityManager am = (ActivityManager) getSystemService(Context.ACTIVITY_SERVICE);
int memClass = am.getMemoryClass();
final int memoryCacheSize = 1024 * 1024 * memClass / 8;

DisplayImageOptions options = new DisplayImageOptions.Builder()
        .showStubImage(android.R.color.transparent)
        .bitmapConfig(Bitmap.Config.RGB_565)
        .imageScaleType(ImageScaleType.IN_SAMPLE_INT)
        .cacheInMemory(true)
        .cacheOnDisc(true)
        .build();

File cacheDir = StorageUtils.getCacheDirectory(this);
ImageLoaderConfiguration config = new ImageLoaderConfiguration.Builder(getApplicationContext())
        .taskExecutor(downloadExecutor)
        .taskExecutorForCachedImages(cachedExecutor)
        .memoryCache(new UsingFreqLimitedMemoryCache(memoryCacheSize)) // 2 Mb
        .discCache(new TotalSizeLimitedDiscCache(cacheDir, 52428800))
        .imageDownloader(new BaseImageDownloader(this, 5 * 1000, 30 * 1000)) // connectTimeout (5 s), readTimeout (30 s)
        .defaultDisplayImageOptions(options)
        .build();
```

К сожалению при использовании Picasso у меня были проблемы со скоростью загрузки картинок. Правда это было в первых версия библиотеки. Как она сейчас работает в плане скорости загрузки и отображения картинок?

Artem_Manaenko, 8 апреля 2014 в 20:21 (комментарий был изменён) # 0

А почему именно такой объем памяти?

```
int memClass = am.getMemoryClass();
final int memoryCacheSize = 1024 * 1024 * memClass / 8;
```

memClass / 8 — почему именно этот коэффициент?
Всегда волновал вопрос, как правильнее высчитать размер кеша.

Mecid, 8 апреля 2014 в 20:42 # 0

Коэффициент был выбран опытным путем. Picasso использует коэффициент равный 7. Вроде и у них и у меня все ок с этим)

vovkab, 8 апреля 2014 в 21:33 # 0

developer.android.com/training/displaying-bitmaps/cache-bitmap.html#memory-cache

> **Note:** In this example, one eighth of the application memory is allocated for our cache. On a normal/hdpi device this is a minimum of around 4MB (32/8). A full screen GridView filled with images on a device with 800x480 resolution would use around 1.5MB (800*480*4 bytes), so this would cache a minimum of around 2.5 pages of images in memory.

Artem_Manaenko, 8 апреля 2014 в 20:23 # 0

Есть идея, как кодом вычислить «подергивания»? Если что — реализую, и будет ещё один Benchmark.

Mecid, 8 апреля 2014 в 20:43 # 0

Поддергивания только в реальном использовании можно заметить. Бенчмаркам не поддается))

У вас в реальных проектах picasso используется? есть списки с картинками?

Artem_Manaenko, 8 апреля 2014 в 22:01 # 0

Пока что UIL. На тест Picasso подтолкнула постоянная проблема с бесконечной сборкой мусора и OutOfMemory на старых устройствах. Но планирую пробу заинтегрить Picasso.

Mecid, 8 апреля 2014 в 22:20 # 0

я планирую отказаться от UIL из-за крешей последней версии на android 4.3 при попытке загрузить много картинок.
Хотелось бы еще оптимизировать использование памяти, вот присматриваюсь к Picasso и Glide.

Mecid, 9 апреля 2014 в 14:40 # 0

Еще как я понял загрузку картинок при скроле нельзя тормозить в picasso?

AChep, 12 апреля 2014 в 19:58 # 0

можно вот так: Настройки > Для разработчиков > Запись времени работы GPU

r_ii, 8 апреля 2014 в 20:31 # 0

Еще было бы интересно увидеть в тестировании volley

Artem_Manaenko, 8 апреля 2014 в 22:01 # 0

На выходных добавятся ещё 3 библиотеки. Выпущу benchmark v2.

vovkab, 8 апреля 2014 в 21:12 (комментарий был изменён) # +1

В UIL есть еще баг который приводит к кешированию картинок 2 раза.

Все потому, что используют размер вью в качестве ключа для кеша, и если у вас ImageView, например, match_parent/wrap_content, то в первый проход размер не будет известен, а будет только во второй, как результат, получаем:
1. дублирующиеся картинки в памяти (и на диске?)
2. и бонусом мигание картинок при прокрутке.

p.s. баг тикет на github issues/376

NOSTRA, 10 апреля 2014 в 00:24 # +1

Дублированиения на диске нет. Дублирование в памяти можно предотвратить с помощью опции конфигурации ImageLoaderConfiguration.denyCacheImageMultipleSizesInMemory().

А проблема с первым проходом при match_parent/wrap_content — да, существует. На данный момент выход — это вызывать ImageLoader через post():

```
imageView.post(new Runnable() {
    @Override
    public void run() {
        imageLoader.displayImage(...);
    }
});
```

vovkab, 10 апреля 2014 в 01:53 # 0

Я для себя сделал грязный хак, который решает все эти проблемы разом:

/* com.nostra13.universalimageloader.core.ImageLoader:194 */
String memoryCacheKey = uri; // MemoryCacheUtil.generateKey(uri, targetSize);

Не очень красиво, за то эффективно. Но будет здорово, если можно будет это решить как то в конфигурации по умолчанию, без лишних движений.

NOSTRA, 10 апреля 2014 в 23:00 # 0

Да, возможность конфигурить извне memoryCacheKey (типа `MemoryCacheKeyGenarator`) — это в ближайших планах.

vovkab, 10 апреля 2014 в 06:44 # 0

по поводу post runnable

Ну вот у нас есть сетка с картинками, видно сразу 20+ картинок, то есть мне нужно запостить 20 runnable просто что бы подождать пока будет известен размер для ImageView? А потом мне что нужно хранить флаг, и больше это не делать, или каждый раз при прокрутке пулять 20 + runnable?
Как то это все не правильно.

NOSTRA, 10 апреля 2014 в 23:02 # 0

Да, пока это такой workaround. Хорошего решения, встроенного в текущую логику библиотеки, я пока не придумал.

artemgapchenko, 8 апреля 2014 в 22:42 # 0

В документации по UIL написано, что порядок определения размеров Bitmap следующий:

- Get actual measured width and height of ImageView
- Get android:layout_width and android:layout_height parameters
- Get android:maxWidth and/or android:maxHeight parameters
- Get maximum width and/or height parameters from configuration (memoryCacheExtraOptions(int, int) option)
- Get width and/or height of device screen

Соответственно, если размеры ImageView, в который нужно положить изображение, никак не ограничить (match_parent/wrap_content не в счет), произойдет загрузка изображения, смасштабированного под ширину/высоту устройства. Я для себя проблему расчета размера Bitmap'ов при загрузке оных в GridView решил установкой параметров maxWidth/maxHeight, принадлежащих отдельным ImageView, в нужные мне значения.

vovkab, 8 апреля 2014 в 23:00 # +2

Вообще это какой то не правильный подход. Зачем ограничивать размеры вью? Что бы потом изобретать велосипед с ручным расчетом на разных размерах экранов? Для этого и придуман wrap_content/match_parent/ layout_weight.

Все что написано вами выше, должно быть опционально и выключено по умолчанию. Потому что, большинство рест апи позволяет:
1. тянуть картинки разного качества, а это значит что нам не нужно дополнительно, что либо еще делать на клиентах. Экономим память, диск, сеть, батарейку и тд.
2. ImageView, без труда может и сам масштабировать картинку, если нужно.

Итого, самый популярный способ показать картинки: GridView с автоматическим расположением картинок (количество столбцов, размер и тд), и uil тут явно не ваш друг.

artemgapchenko, 9 апреля 2014 в 11:28 # ☆ ⑂ ↑ 0

У меня опыта разработки под Android всего два месяца, поэтому могу порой принимать странные решения. :) Спасибо за объяснение, посмотрю в сторону Picasso и других библиотек.

NOSTRA, 10 апреля 2014 в 23:05 # ☆ 0

Кстати, какие версии библиотек использовались в бенчмарке? Это важно.

Artem_Manaenko, 12 апреля 2014 в 19:49 # ☆ ⑂ ↑ 0

из build.gradle:
compile 'com.squareup.picasso:picasso:2.1.1@jar'
compile 'com.nostra13.universalimageloader:universal-image-loader:1.9.0@jar'