22 апреля в 01:11

ImageLoaders: Продолжение

Разработка под Android*

В предыдущей статье я сравнивал библиотеки UniversalImageLoader (UIL) и Picasso на предмет оптимального использования памяти. Среди комментариев были некоторые тонкости применения библиотек и просьбы протестировать другие загрузчики. Эти вопросы дали начало второй части benchmark-а с участием 5 библиотек.
Результаты:
Итак, в benchmark-е будут принимать участие библиотеки:
UIL, Picasso, Glide, Query, Volley

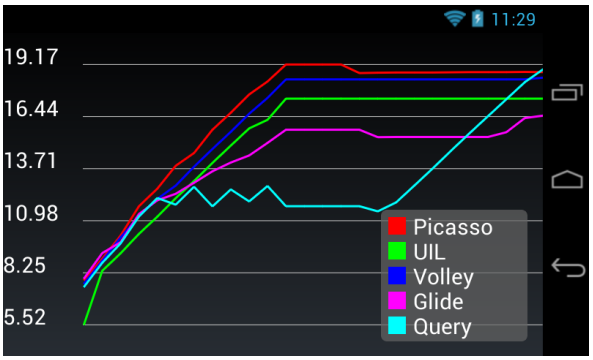
Исходники можно найти тут

В **первой** части статьи я проведу эксперименты с измерением используемой памяти с участием пяти библиотек.
Во **второй** части я сравню библиотеки по возможностям их настройки и визуальным характеристикам.

Используемая память

Если кому-то интересно ознакомиться с деталями эксперимента — их можно найти в предыдущей статье.
Скажу вкратце: я буду в равных условиях тестировать библиотеки для загрузки изображений и измерять используемую ими память. Графики на девайсе рисовались с помощью библиотеки GraphView.

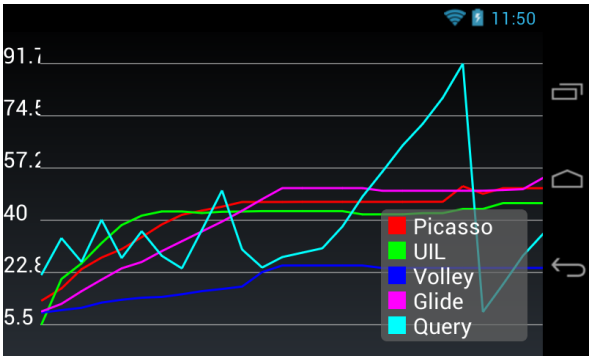
Загрузка небольших изображений 240x240



В этом эксперименте можно отметить снижение расхода памяти у **UIL** благодаря хаку от автора библиотеки:

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

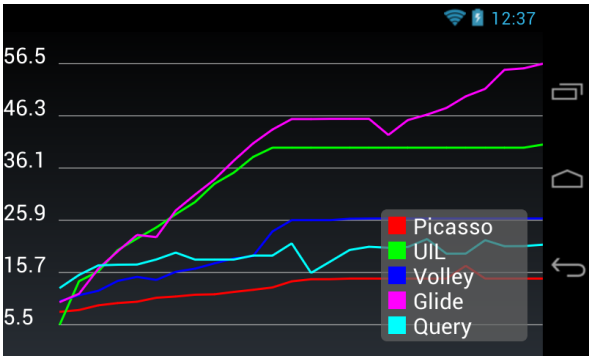
Загрузка больших изображений без конвертации формата



Весьма интересно ведет себя библиотека **Query**. Я не лазил в исходники, но у меня есть предположение, что кеш основан на списке, а не множестве. Иначе я не знаю как объяснить дублирование изображений на обратном пролистывании грида.
Так же неприятно удивила **Volley**. Очень часто не подгружается картинка в ImageView. Дебаг указал на то, что иногда производятся попытки установить recycled Bitmap. Вероятно, есть какая-то проблема с выбором подлежащих очистке картинок.

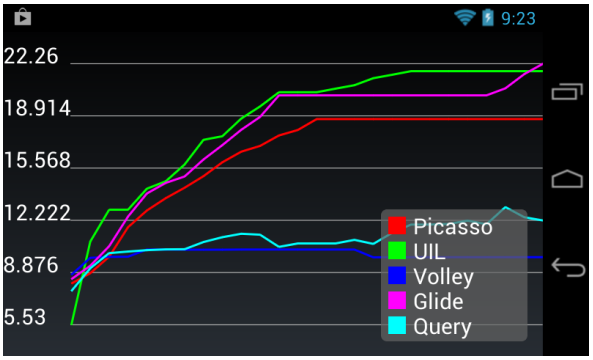
Загрузка больших изображений с конвертацией в формат RGB565 и подгонкой изображения к размеру ImageView

Как правило, от больших картинок не требуется 24-битный формат, в большинстве случаев достаточно и RGB565.
Наиболее просто сконвертировать формат позволяет **UIL**. Достаточно в опциях загрузки изображения указать необходимый Bitmap.Config.
Для остальных библиотек конвертацию пришлось писать.



В этом эксперименте особо выделилась **Glide**, заняв памяти больше чем **Volley**, **Query** и **Picasso** вместе взятые. Сразу хочу уточнить: можно было ограничить размер кеша, но в эксперименте я хотел увидеть как библиотеки работают с памятью в базовой конфигурации. Каждая глобальная чистка по лимиту кеша — это заметный затык в работе UI-потока. Поэтому от библиотеки хотелось бы какой-то постепенной очистки, если так можно выразиться.

Загрузка маленьких изображений (240x240) с последующим скруглением



К сожалению, столь успешные результаты **Volley** учитывать нельзя, поскольку на обратном проходе я не увидел картинок. Все они оказались recycled, даже с учетом, что я не вызывал recycle() у оригинала при трансформации. **Не годится.**

Glide, судя по всему, выполняет преобразование всех картинок в RGB565, поскольку у картинок оказался черный фон вместо прозрачного. **Не годится.**

Наиболее просто делать трансформацию с помощью **UIL** и **Query**. В **UIL** есть встроенный класс RoundBitmapDisplayer, который можно подключить к опциям загрузки.
Для **Query** достаточно в опциях указать радиус скругления картинки.
Для **Picasso** и **Glide** включение трансформации в загрузку выглядит как один из этапов Builder-а.

```
Glide.load(url).centerCrop().transform(roundTransformation).into(imageView);
Picasso.with(context).load(url).transform(picassoRoundTransformation).noFade().into(imageView);
```

Для **Volley** пришлось переопределять ImageView и делать конвертацию Bitmap-а в методе setImageBitmap().
Если у кого есть решение лучше — предлагайте свои варианты.

Визуальные характеристики.

Picasso уступает остальным библиотекам в скорости загрузки изображений. Не знаю как это объяснить, но факт: картинки через Picasso появляются чуть позже.

Glide, как было сказано выше, преобразовывает картинки в RGB565, чем может доставить неприятности при трансформации изображений.

UIL визуально быстрее загружает картинки. Но нужно учесть одну особенность: при использовании FadeInBitmapDisplayer для плавного показа картинок, лучше использовать расширенный конструктор.

```
new FadeInBitmapDisplayer(duration, true, true, false)
```

В такой конфигурации картинка будет анимироваться только в первый раз (пока картика жива в кеше), и при пролистывании не будет напрягать взгляд излишней анимацией. Это ИМХО.
В **Picasso** эффект «fade in» работает по-определению именно в таком режиме.

Volley в свою очередь часто не показывает преобразованные изображения из-за того, что они уже recycled(). Жду в комментариях ваши предложения по исправлению этого бага.

Query визуально ничем не удивил. Загрузка картинок выглядит как-то прерывисто. Редко соблюдается тот порядок, в котором картинки были запрошены.

Настройка загрузчика

UIL — безусловный лидер по возможностям тонкой настройки параметров загрузки. Благодаря Builder-паттерну формирования настроек можно легко ознакомиться с возможностями библиотеки. В принципе, можно обойтись и без мануала. UIL так же позволяет подключить

OnScollListener для приостановки запросов на время скролла списка.

Glide и **Picasso** — самые простые в интеграции библиотеки. **Picasso** имеет преимущество в наличии maven-репозитория. Если нужно быстро интегрировать загрузку изображений, при этом получив оптимальные настройки и экономный расход памяти — выбор за **Picasso**.

Picasso и **UIL** предоставляют интерфейсы, реализуя которые можно загрузить картинку куда угодно.

Volley требует долгой подготовки к использованию. Нужно самому создавать и конфигурировать экземпляр кеша и очередь загрузки. В разметке необходимо использовать виджет NetworkImageView или же его наследники. Для меня — это значительное неудобство. Единственный приемлемый вариант использования этой библиотеки для загрузки изображений — это если Network вашего проекта построен на Volley.

Query, в принципе, не тяжела в настройке. Но предлагаемый авторами вариант использования в адаптерах меня немного смущает. Для загрузки каждой картинки нужно создавать экземпляр класса AQuery, который сам по себе является довольно тяжелым. Это не лучшая библиотека, если хочется сэкономить процессорное время на сборке мусора.

▸ Пример использования Query

Нередко встречается необходимость использования своего HTTPClient-а для загрузки картинок. **Volley** и **UIL** предоставляют инструменты для настройки соединения. На просторах интернета я также встречал интеграцию кастомного HttpClient в **Picasso**. Но насколько я понял, стандартными средствами это сделать не получится.

Выводы

Чем больше библиотек — тем сложнее стало сделать какие-либо выводы. Попробую в одном предложении охарактеризовать каждую библиотеку.

Picasso — если нужно быстро интегрировать и не беспокоиться об используемой памяти
UIL — если Ваш проект требует специфическую настройку загрузки, но в то же время позволяет выполнить эту настройку быстро и понятно
Glide — некий клон **Picasso** (возможно, наоборот), но расходует немного больше памяти.
Volley — использование оправдано, если ваш Network построен на **Volley**, а соединение требует специфической настройки HttpClient-а для всего проекта
Query — сложная к пониманию и интеграции библиотека со слабой документацией, тем не менее показывающая отличные результаты по экономии памяти устройства.

Жду Ваших комментариев и полезных советов!

А какие ImageLoader-ы используете Вы?

55% (84)	UniversalImageLoader
30% (46)	Picasso
12% (18)	Volley
1% (1)	Glide
3% (5)	Query
19% (30)	Самописный
3% (5)	Другой (отпишите в комментариях)

Проголосовало 154 человека. Воздержалось 74 человека.

android, image loaders, андроид, загрузка изображений, benchmark, память, сравнение

Похожие публикации

Четыре метода загрузки изображений с веб-сайта с помощью Python 25 января в 00:24
Управление загрузкой изображений 13 августа 2013 в 12:08
PHPShop PriceLoader: пакетная обработка прайс-листов и загрузка изображений в интернет-магазин. Часть 2 16 июля 2013 в 10:33
Пакетная обработка и загрузка изображений товара в интернет-магазин 13 сентября 2012 в 11:53

Реализации прогрессивной загрузки изображений 23 мая 2012 в 18:50

Синхронная и асинхронная загрузка изображения из сети с последующей обработкой 2 марта 2012 в 18:49

Автоматизация загрузки изображений для товаров в интернет-каталогах 16 апреля 2010 в 09:04

Как реализовать загрузку изображений в список в отдельном потоке на Android 19 декабря 2009 в 18:45

Загрузка изображений на ImageShack в один клик 18 августа 2009 в 00:20

Безопасная загрузка изображений на сервер. Часть вторая 14 ноября 2008 в 16:06

Комментарии (16)

отслеживать новые: в почте в трекере

03uk, 22 апреля 2014 в 10:01 # +1

Жаль вы не добавили в сравнения Ion библиотеку

Artem_Manaenko, 22 апреля 2014 в 10:13 # +3

Честно говоря, не увидел этой библиотеки в комментариях, потому проигнорировал.
Я за неделю добавлю тест и этой библиотеки и выложу результат на GitHub. О готовности отпишусь в комментариях к этой статье.

Evgenij_Popovich, 22 апреля 2014 в 12:15 # +2

Использую bitmapfun developer.android.com/training/displaying-bitmaps/index.html. В некоторых местах пришлось допиливать под свои нужды, но со временем основные недостатки удалось побороть.

Mecid, 22 апреля 2014 в 17:44 # 0

В первом посте вы сделали вывод, что нужно переводить проекта на Picasso. Сейчас ваше мнение изменилось?

Artem_Manaenko, 22 апреля 2014 в 18:16 # ↑ 0

Нет, не изменилось. Те проекты, которые не нуждаются в кастомном HTTP-клиенте — уже почти переведены.

Mecid, 22 апреля 2014 в 20:41 # ↑ 0

UIL последний апдейт что-то сломал, на некоторых картинках падает с native exception. В итоге думаю перейти на Picasso, но ваш отзыв о скорости работы смутил немного.

Artem_Manaenko, 22 апреля 2014 в 21:22 # ↑ 0

Picasso интегрится предельно просто и быстро, потому попробовать нужно. Хотя бы просто для опыта. Возможно, работа библиотек как-то разнится от количества уже занятой приложением памяти. Или с GC они «дружат» по-разному. Я тестил на пустышке, а в вашем случае Picasso может справиться лучше остальных.

Mecid, 23 апреля 2014 в 00:17 # ↑ 0

скорость загрузки один из самых важных аспектов для меня, в Robird не хотелось бы экспериментировать с этим

03uk, 24 апреля 2014 в 08:51 # ↑ 0

Picasso по загрузке работает очень быстро и стабильно различия в мс, UIL классная штука, но Picasso мне показалось круче) NOSTRA простите

NOSTRA, 22 апреля 2014 в 23:22 # ↑ 0

Последний — это 1.9.1?

Mecid, 23 апреля 2014 в 00:16 # ↑ 0

именно

NOSTRA, 23 апреля 2014 в 00:34 # ↑ 0

Не могли бы вы скинуть лог этого native exception'а в личку?

Mecid, 23 апреля 2014 в 01:24 # ↑ 0

отправил

yanchenko, 22 апреля 2014 в 18:37 # +1

Буду благодарен, если добавите в сравнение мой *ImageFetcher*: droidparts.org/image_fetcher.html

Artem_Manaenko, 22 апреля 2014 в 19:21 # ↑ +1

Добавлю. Результат будет тоже на GitHub. Через недельку где-то.

 **recompileme,** 22 апреля 2014 в 22:07 # –3

За то время, которое Вы потратили на сравнение библиотек, можно было написать свой, ультрабыстрый имаджлоадер.