Performance

Performance

- With more and more apps being developed on android devices performance is starting to matter to users
  - Even though the latest devices have much more horsepower available to them.
  - In this lecture we will explore some things that can improve the performance of your applications
  - To make them run more smoothly

Why performance matters

- Performance matters for two main reasons
  - Higher performance means a smoother more instantanious application that responds quickly to user input
  - Apps that exhibit higher performance on the same tasks will complete in a shorter time meaning there is less use of the battery
  - Thus it is in your interest to make your applications perform well

The two basic rules of performance

- There are two basic rules to improving performance
  - Don't do work that you don't need to do
  - Avoid memory allocation if you can
  - By observing these rules you will create an application that shows high performance and will place little drain on the battery

Where most of the following advice comes from

- Most if not all of the advice that you will see here comes from the Android documentation on the topic of performance
  - http://developer.android.com/training/articles/perf-tips.html
  - Advice that comes directly from the android developers themselves
  - Thus it is advice worth considering

How the performance numbers are obtained

- Most of the performance numbers that you will see here have come from experimentation on android code
  - And also through the use of a code profiler, which is a tool for monitoring the execution time of code.
  - A profiler will produce a range of statistics that will show where a program spends the majority of its time.
  - Gives an indication as to where the code can be improved to produce extra performance

The goal that is to be achieved when performing optimisation

- When performing optimisation you must remember that you will run into the law of diminishing returns
  - Each optimisation you do after previous operations will result in a smaller performance gain
  - The goal therefore is to only do optimisation to the point where performance is percieved as being good enough
  - This is because you don't remove bottlenecks in applications they just move elsewhere

Mobile is more complicated compared to desktop

- You also have an additional issue in that your code has to support multiple CPU architectures
  - Thus CPU specific optimisations may improve performance for one CPU type but may kill performance on others
  - Whereas on a desktop you are mainly limited to x86 based architectures
  - To complicate maters further there is a JIT (Just In Time) compiler that is also trying to optimise your code at run time

Try to avoid allocating memory

- Memory allocation/deallocation are expensive operations in any OS
  - Require a switch from user mode to kernel mode to allocate the memory and another switch back to continue running the application
  - Also the more you allocate memory the more the garbage collector will get involved and will reduce your application performance
  - There are a couple of examples involving string buffers and parallel single dimension arrays that can help reduce allocation

Examples with string buffers

- If you have a method that returns a string that is to be used immediately with a StringBuffer or StringBuilder you will be better just appending the string to the StringBuilder directly
  - Prevents the need to allocate and deallocate a temporary string on each method call.
- Another option when working with small portions of strings is to use a substring instead of making a copy
  - Instead of copying the data across the substring will use the same shared character data, thus saving the need to copy values

Parallel single dimension arrays

- If you have an array of objects that are simply containers you will be better splitting them up into separate single dimension arrays
  - E.g. if you have an array of (Foo, Bar) objects it would be quicker to process them as two parallel arrays Foo[] and Bar[]
  - Prevents the need to allocate and deallocate memory for the container objects
  - Generally this is reserved for internal code as its not a good idea to expose this in an API

Using static methods

- Use static methods where possible as they are much quicker than virtual methods
  - Particularly if you don't need to access an object's fields
  - Invocations on static methods are about 15 to 20% quicker than object methods
  - Difficult to find such opportunities but take them where possible

How static final helps

- Using the static final keywords as applied to constants will improve the lookup and reference times of these constants
  - The constants will be computed once and stored in the dex homogenous pools
  - Where it can be referenced and looked up relatively inexpensively
  - Precomputed data like this removes computations from your program before it even runs.

Avoiding internal getters and setters

- Normally in object oriented code partiularly in native languages like C++ you will usually use getters and setters internally in a class
  - While most compilers will inline access to these methods (replacing the method call with the actual code of the method) this is not done in android
    - According to the documentation
  - To avoid the slowdown associated with the method calls you should try to inline the code yourself. However be aware that inlining only works for 1 or 2 line methods.

Avoid using floating point numbers if possible

- Floating point calculations take twice as long on android devices as the same integer calculations.
  - Thus wherever possible try and use integer calculation instead
  - An example of such a trick is with applications that deal with monetary values. Here we would represent 100.00 euros in floating point but it can be represented just as well as 10,000 cent
  - Thus all floating point calculations are replaced by integer operations

Know your libraries

- Try to use standard library code where possible as it is generally reliable and is also likely to have much higher performance than equivalents you write
  - Lots of libraries use either inlined intrinsics or hand written and tuned assembly code for certain functions that will run at native speed
  - For example the System.arraycopy() method on a Nexus one will run 9 times faster than a hand coded loop passed through a JIT
    - Because it uses hand tuned assembly to do the job quickly

Optimise your layout hierarchies

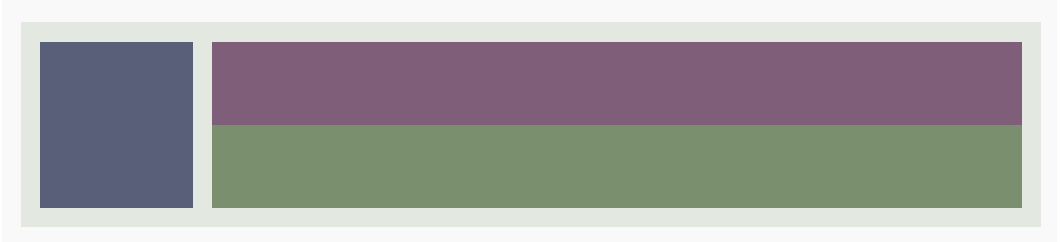
- Layouts that have many different levels will take longer to build.
  - Also they will take longer to resize
  - Thus it is a good idea to observe the hierarchies of your layouts and see if they can be flattened in any way
  - This is one of the reasons why the relative layout is used frequently in android. As flattened layouts take the least amount of time to construct and resize

The hierarchy viewer

- The Android SDK tools provide a hierarchy viewer tool that enables you to analyse your layouts while your application is running
  - It will provide you with running stats of your layout measure, layout and draw performance
  - Uses a traffic light system to identify potential bottlenecks
  - The next slide shows an example layout with issues in laying out textviews

An Example hierarchy

- For the purposes of an example we will use a couple of linear layouts to represent the item that we have shown below
  - We will use a vertical linear layout for the textviews on the right
  - And that will be contained in a horizontal linear layout that will have an image on the left

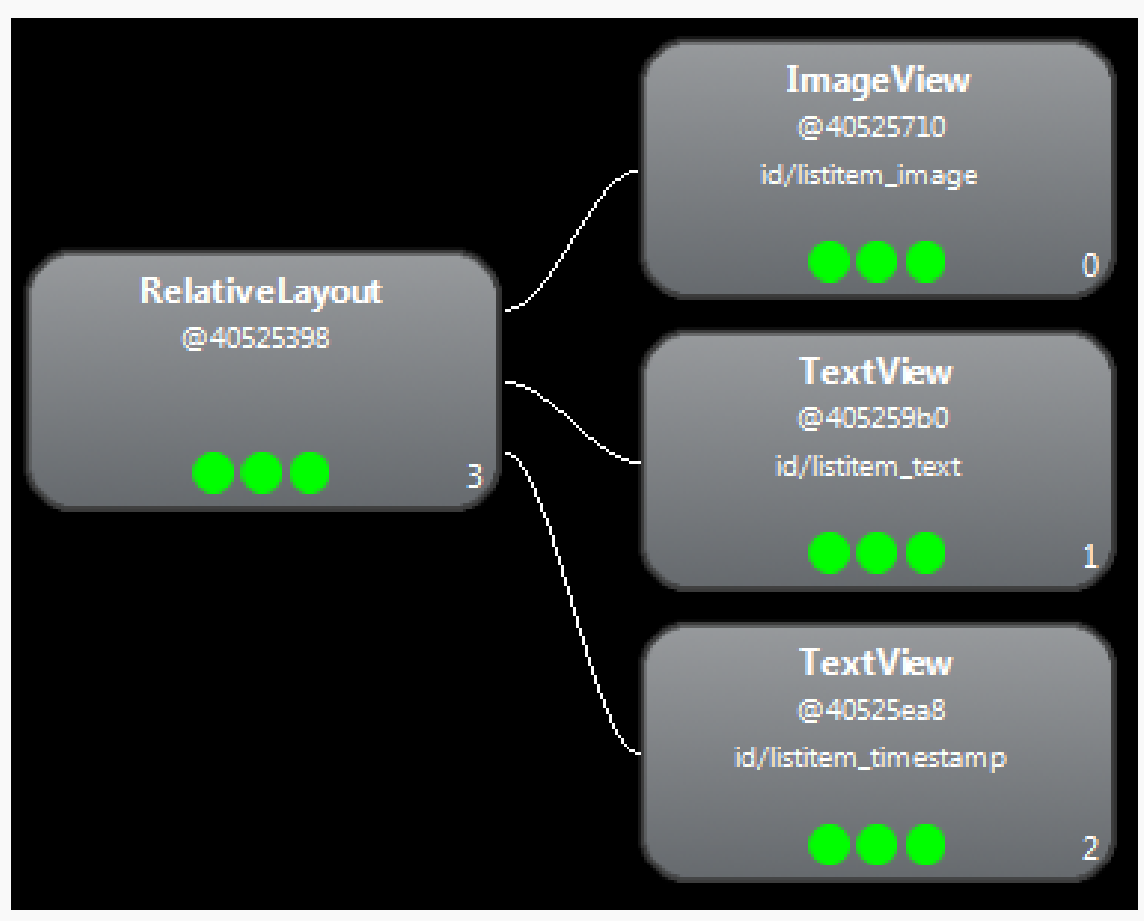


An example hierarchy

Flattening the layout

- A hierarchy like this takes some time to build
  - This will add up significantly if something like this is used in a list view
  - Thus flattening this layout to two levels with a relative layout instead of a three level linear layout hierarchy would be benefical
  - An example of the revised layout is shown on the next slide

Flattening the layout



Performance difference

- The difference between the two layouts in the measure, layout and draw times is at least 20% quicker with the two level relative layout over the three level linear layouts
  - While this may seem like a small saving, that saving is multiplied when used in a list view of many items
  - Thus redesigning layouts can save you time in your applications.
    - Particularly if you have an application that will load and construct layouts frequently

Using the include and merge tags

- Very simple trick here to reduce the size of your code. If you identify combinations of components that are frequently reused together you are better off splitting them into seperate files
  - And including them in each layout by using the <include> tag
  - Each reused layout should have the <merge> tags as the root
  - This way they will be included into the layout without any additional items in it's place

Optimising for battery

- In general your applications should try to use as little battery of a device as is possible
  - Remember these devices generally last a day at best
  - Thus applications that consume battery will be removed quickly
  - It is possible however to modify the behaviour of your application in response to changes in charging state or battery level.

Determining and monitoring charging state

- By registering an intent filter that looks for the ACTION_BATTERY_CHANGED intent you can determine information about the charging state of the device
  - Will enable you to not only determine if the device is charging
  - But also if it is charging via a USB port or from AC power
  - For example you might ramp up performance when connected to AC than when you are on battery power. Also used to ramp up brightness

Determining and monitoring battery level

- Using another intent filter it is possible to determine if the battery level is low or if it is ok
  - Generally when a battery level goes low it is a good idea to reduce your use of resources in your application
    - See examples on next slide
  - Enables the device to use as little power as possible when the battery level is 15% or less

Options in response to battery levels and charging states

- Somethings applications frequently do when they have been informed that the battery level of the device is low are
  - Reduce the sampling rate on sensors to consume less battery power
  - Disable some functionallity entirely. e.g. disabling access to flash with low power
  - If the application is not critical to the system then disable all functionality to save power.

Final bits of advice

- If you must implement an algorithm that is not found in the standard libararies try to implement it in the most efficient way possible
  - More efficiency == less battery drain
- If you know what operations your application is expected to mostly do then try and use data structures that provide fast implementations of your common operations.
  - Again will reduce the amount of work your application has to do.

Final bits of advice

- When drawing custom views keep all memory allocation in the shared init() method instead of the onDraw() method
  - As stated before onDraw may be called many times a second and the allocation routines will definitely slow things down here
- Finally avoid feature creep and software bloat
  - Remember your application should do a single job and should do it well. Resist the urge to add more and more features as this will slow down your application performance