**[Моя практика логгирования в Android](http://megadarja.blogspot.ru/2012/11/androidlogwrapper.html)**

Я люблю логи, считаю их лучшим инструментом отладки, чем дебаггер и стараюсь использовать везде.

Логгирование в Андроиде меня удивило. С одной стороны, состав инструментов SDK намекает на него как главный инструмент отладки. Да и при тестировании без них никуда: андроидское сообщение "Приложение неожиданно остановилось" обычно совершенно ни о чем не говорит, а логи позволяют воссоздать картину преступления. С другой стороны, API для логгирования, мягко говоря, не блещет. Особенно по сравнению с такими монстрами, как log4j.

**Что мне не нравится**

Как известно, вызов лога выглядит примерно так:

Log.i(tag, message)

Возникают следующие вопросы/претензии:

1. Зачем такая искусственная сущность, как tag?
2. Почему бы не выводить имя класса и номер строки, где тег вызвался?
3. Почему в сигнатурах методов не предполагается никакого форматирования?

В общем, несмотря на наличие средств для логгирования, всё равно приходится писать свою обёртку

**Tag**

Этот самый tag вызывает массу вопросов. Что туда выводить?

Мне встречалось несколько вариантов:

* Одна константа на все приложение
* Имя класса, откуда вызывается лог
* Что попало (в зависимости от логической части программы)

С первыми двумя пунктами всё более-менее понятно. Третий подход используется, например, в Facebook SDK ([конкретный пример](https://github.com/facebook/facebook-android-sdk/blob/master/facebook/src/com/facebook/android/Facebook.java))

Какой подход выбрать? В зависимости от приложения. Если приложение небольшое и логов пишет немного, то сойдет и первый. Если приложение побольше и флудит в логи много, то имеет смысл использовать второй или третий подход. Единственное, лучше использовать для тегов общий префикс, чтобы иметь возможность просмотреть весь лог приложения. Просто для того, чтобы было удобно фильтровать логи в зависимости от той части, которую мы проверяем.

**Форматы**

Типичный вызов лога в андроиде выглядит так:

Log.d(TAG, "Detected location from = " + location.getProvider() + ", (" + location.getLatitude() + ", " + location.getLongitude() + ")");

А теперь посмотрим на [log4net](http://megadarja.blogspot.ru/2008/04/log4net.html):

Log.InfoFormat(

"CustomReports.GetReportData: the report type [{0}] is not supported as custom report",

reportInfo.ReportTypeId

);

Вызов второго вида кажется намного более приятным.

**Источник лога**

В log4net была возможность задать в конфигурации, чтобы со строчкой лога выводился класс, метод и номер строки, откуда лог был вызван.

По-моему, это мастхэв и жить без этого нельзя. Во-первых, так можно быстро отключать в коде логи, которые сильно мозолят глаза (иначе придется лазить поиском по всему проекту). Ну а во-вторых, это и отладку может сильно ускорить.

Кроме того, иногда хочется вывести не просто класс и номер строки, откуда лог был вызван, а еще и кусок стека вызовов.

**Что делаю**

Пишу разные обертки. Приведу пример наиболее универсальной из них.

* Методы v, d, i, w, e выводят логи с общим тегом.
* В методах vt, dt, it, wt, et можно указать свой тег. При этом желательно, чтобы у всех логов был общий префикс.
* Все методы выводят класс и строчку, где были вызваны.
* Методы vtrace, dtrace, itrace, wtrace, etrace выводят не просто класс и строчку, а ещё и кусок стека вызовов заданной длины. Ну хочется иногда.
* TAG\_WEB и TAG\_STORAGE даны просто для примера.
* ENABLED — для возможности отключить логи в проекте
* С помощью флага LOCATION\_ENABLED можно выключить вывод источника лога. Это если вдруг разовьется паранойя насчет производительности.

@SuppressWarnings("ALL")

public class Log {

private static final String TAG = "MegaApp";

public static final String TAG\_WEB = tag("web");

public static final String TAG\_STORAGE = tag("storage");

private static boolean ENABLED = true;

private static final boolean LOCATION\_ENABLED = true;

public static String tag(String tag) {

return TAG + "-" + tag;

}

public static void vt(String tag, String format, Object... args) {

if (ENABLED) android.util.Log.v(tag, String.format(format, args) + getLocation());

}

public static void dt(String tag, String format, Object... args) {

if (ENABLED) android.util.Log.d(tag, String.format(format, args) + getLocation());

}

public static void it(String tag, String format, Object... args) {

if (ENABLED) android.util.Log.i(tag, String.format(format, args) + getLocation());

}

public static void wt(String tag, String format, Object... args) {

if (ENABLED) android.util.Log.w(tag, String.format(format, args) + getLocation());

}

public static void wt(String tag, String message, Throwable e) {

if (ENABLED) android.util.Log.w(tag, message + getLocation(), e);

}

public static void wt(String tag, Throwable e) {

if (ENABLED) android.util.Log.w(tag, e);

}

public static void et(String tag, Throwable e) {

if (ENABLED) android.util.Log.e(tag, e.getMessage() + getLocation(), e);

}

public static void et(String tag, String format, Object... args) {

if (ENABLED) android.util.Log.e(tag, String.format(format, args) + getLocation());

}

public static void v(String format, Object... args) {

vt(TAG, format, args);

}

public static void d(String format, Object... args) {

dt(TAG, format, args);

}

public static void i(String format, Object... args) {

it(TAG, format, args);

}

public static void w(String format, Object... args) {

wt(TAG, format, args);

}

public static void w(String message, Throwable e) {

wt(TAG, message, e);

}

public static void w(Throwable e) {

wt(TAG, e);

}

public static void e(String format, Object... args) {

et(TAG, format, args);

}

public static void e(Throwable e) {

et(TAG, e);

}

public static void vtrace(int traceLength, String format, Object... args) {

vtrace(TAG, traceLength, format, args);

}

public static void dtrace(int traceLength, String format, Object... args) {

dtrace(TAG, traceLength, format, args);

}

public static void itrace(int traceLength, String format, Object... args) {

itrace(TAG, traceLength, format, args);

}

public static void wtrace(int traceLength, String format, Object... args) {

wtrace(TAG, traceLength, format, args);

}

public static void etrace(int traceLength, String format, Object... args) {

etrace(TAG, traceLength, format, args);

}

public static void vtrace(String tag, int traceLength, String format, Object... args) {

if (ENABLED) android.util.Log.v(tag, String.format(format, args) + getTrace(traceLength));

}

public static void dtrace(String tag, int traceLength, String format, Object... args) {

if (ENABLED) android.util.Log.d(tag, String.format(format, args) + getTrace(traceLength));

}

public static void itrace(String tag, int traceLength, String format, Object... args) {

if (ENABLED) android.util.Log.i(tag, String.format(format, args) + getTrace(traceLength));

}

public static void wtrace(String tag, int traceLength, String format, Object... args) {

if (ENABLED) android.util.Log.w(tag, String.format(format, args) + getTrace(traceLength));

}

public static void etrace(String tag, int traceLength, String format, Object... args) {

if (ENABLED) android.util.Log.e(tag, String.format(format, args) + getTrace(traceLength));

}

private static String getTrace(int length) {

if (!LOCATION\_ENABLED)

return "";

final String logClassName = Log.class.getName();

final StackTraceElement[] traces = Thread.currentThread().getStackTrace();

int foundIndex = -1;

for (int i = 0; i < traces.length; i++) {

StackTraceElement trace = traces[i];

if (trace.getClassName().startsWith(logClassName)) {

foundIndex = i;

} else {

if (foundIndex > 0)

break;

}

}

StringBuilder sb = new StringBuilder();

sb.append("**\n**");

for (int i = foundIndex + 1; i < foundIndex + length + 1; ++i) {

if (i > traces.length)

break;

StackTraceElement trace = traces[i];

sb.append(String.format(" at %s.%s:%s**\n**", trace.getClassName(), trace.getMethodName(), trace.getLineNumber()));

}

sb.delete(sb.length() - 1, sb.length());

return "**\n**" + sb.toString();

}

private static String getLocation() {

if (!LOCATION\_ENABLED)

return "";

final String logClassName = Log.class.getName();

final StackTraceElement[] traces = Thread.currentThread().getStackTrace();

boolean found = false;

for (int i = 0; i < traces.length; i++) {

StackTraceElement trace = traces[i];

try {

if (found) {

if (!trace.getClassName().startsWith(logClassName)) {

Class<?> clazz = Class.forName(trace.getClassName());

String clazzName = clazz.getSimpleName();

if (TextUtils.isEmpty(clazzName))

clazzName = clazz.getName();

return String.format(" [%s.%s:%d]", clazzName, trace.getMethodName(), trace.getLineNumber());

}

}

else if (trace.getClassName().startsWith(logClassName)) {

found = true;

}

}

catch (ClassNotFoundException e) {

}

}

return " []";

}

}

**И что получилось**

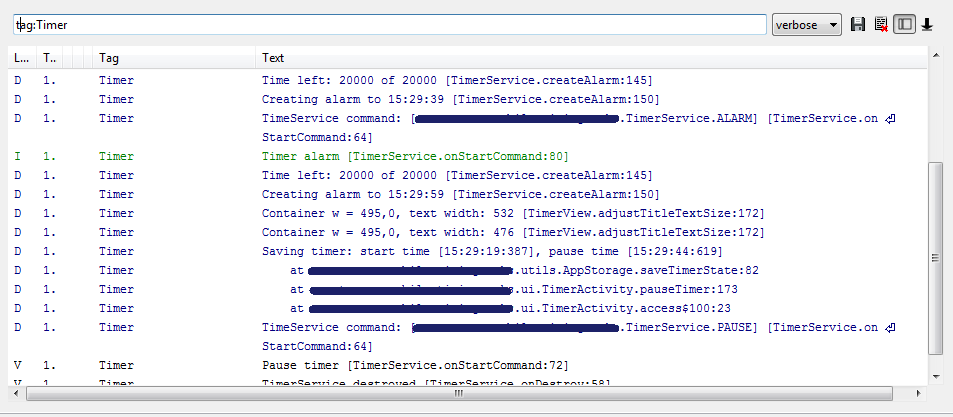
Примеры вызова лога:

Log.d("Saving timer: start time [%1$tT:%1$tL], pause time [%2$tT:%2$tL]", startTime, pauseTime);

Log.d("Container w = [%.1f], text [%s] width: [%d]", mContainerWidth, text, bounds.width());

(Небольшое замечание. Я обычно заключаю значения в квадратные скобки. Это, во-первых, для того, чтобы отделить их от всего остального текста. А во-вторых, в значении может быть пустая строка, которую я не замечу. А [] замечу точно)

А вот, что видим в ddms при использовании этой обертки:

[](http://2.bp.blogspot.com/-py8lq_5NO5s/ULEbViUA2BI/AAAAAAAADIA/OoV2nZYrwuE/s1600/log_example.png)

**Заключение**

Кто-нибудь может сказать, что String.format и определение источника — долгие операции, а поэтому так писать не надо. Могу возразить, что при выпуске продакшен-версии вызовы логов можно (и нужно) отключать (с помощью флага ENABLED или [с помощью Proguard](http://megadarja.blogspot.ru/2012/07/android-disable-log.html)), так что на производительность они не повлияют.

Ещё есть мысль, что в андроиде вполне юзабелен log4j. Возможно. Но добавление библиотеки, функциональность которой никак не пригодится пользователю, — более чем сомнительное решение. А логи Андроида с описанной доработкой вполне себе ничего.