**Логи а Android**

Логи - это текстовые сообщения, выводимые программистом из разных мест программы для отслеживания хода её выполнения. Они очень удобны для отладки приложений. Обычно их выводят в текстовые файлы, но также их можно выводить на экран, в консоль, да и вообще куда угодно.  
  
В Android существует специальный класс **Log**, который позволяет выводит логи из запущенного приложения в специальную консоль LogCat в среде разработки Eclipse или же в консоль на самом устройстве, откуда их можно читать специальным приложением. Но после бета-тестирования, при публикации приложения для всеобщего пользования, логи из программы рекомендуется убирать, чтобы они не тормозили работу программы и не перегружали процессор. А это довольно нудная задача.  
  
Кроме того, стандартный класс **Log**просто выводит вписанное программистом сообщение один к одному и его тег для фильтра. В сообщении программисту, чтобы не запутаться, приходится указывать не только то, что он хочет сообщить, но также и класс, и метод, к которому это сообщение отностся. Чтобы не писать лишних слов, программисты придумывают различные классы-обёртки для класса Log, автоматизирующие этот процесс.  
  
Сегодня, столкнувшись с необходимостью активного использования логов, я нашёл на просторах Интернета один из таких классов, немного его доработал, прокомментировал, и выставляю его на всеобщее обозрение. Класс просто добавляется в проект приложения, после чего из проекта убирается объявление импорта стандартного класса Log:*import android.util.Log;* В результате те же самые команды вывода логов, уже имеющиеся в тексте, начинают вызывать наш класс, и добавлять в вывод информацию о классе, методе и даже номере строки расположения команды! Помимо всего прочего, новый класс Log позволяет полностью отключать логирование путём установки константы logEnabled в состояние false.  
  
Те, кого беспокоит, что компилятор javac не удалит [dead-code elimination](http://ru.wikipedia.org/wiki/%D0%A3%D0%B4%D0%B0%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BC%D1%91%D1%80%D1%82%D0%B2%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BE%D0%B4%D0%B0), могут воспользоваться утилитой Proguard, которой можно вырезать класс целиком. Метод описан здесь: <http://megadarja.blogspot.ru/2012/07/android-disable-log.html>.

Код Класса

import android.text.TextUtils;

/\*\*

\* Отладочный класс, заменяющий стандартный, и автоматически выдающий в лог помимо сообщения - класс, метод и номер строки.

\* Основа взята отсюда: http://habrahabr.ru/post/116376/

\* Развитие основы: http://megadarja.blogspot.ru/2012/11/androidlogwrapper.html

\*/

public final class Log {

private static boolean logEnabled = true; // Установив константу в false мы отключим выдачу логов.

// ERROR

public static void e(String tag, String msg) {

if (logEnabled)

android.util.Log.e(tag, getLocation() + msg); // Вывести лог уровня ошибок

}

// WARN

public static void w(String tag, String msg) {

if (logEnabled)

android.util.Log.w(tag, getLocation() + msg); // Вывести лог уровня предупреждений

}

// INFO

public static void i(String tag, String msg) {

if (logEnabled)

android.util.Log.i(tag, getLocation() + msg); // Вывести лог уровня информирования

}

// DEBUG

public static void d(String tag, String msg) {

if (logEnabled)

android.util.Log.d(tag, getLocation() + msg); // Вывести лог уровня отладки

}

// VERBOSE

public static void v(String tag, String msg) {

if (logEnabled)

android.util.Log.v(tag, getLocation() + msg); // Вывести лог уровня подробностей

}

/\*\*

\* @return Текст, содержащий расположение лога в теле программы: класс, метод, номер строки расположения лога

\*/

private static String getLocation() {

final String className = Log.class.getName(); // Получить имя класса лога

final StackTraceElement[] traces = Thread.currentThread().getStackTrace(); // Получить стек вызовов текущего потока

boolean found = false; // Инициализировать переменную поиска класса

for (int i = 0; i < traces.length; i++) { // Перебрать все вызовы

StackTraceElement trace = traces[i]; // Забрать очередной класс

try {

// Если текущий класс в стеке вызовов найден

if (found) {

if (!trace.getClassName().startsWith(className)) { // Запросить следующий класс, и если он не класс лога

Class clazz = Class.forName(trace.getClassName()); // взять экземпляр класса (объект)

return "[" + getClassName(clazz) + ":" + trace.getMethodName() + ":" + trace.getLineNumber() + "]: "; // Заполнить и вернуть строку

}

}

// Найти текущий класс в стеке вызовов

else if (trace.getClassName().startsWith(className)) { // Если имя класса начинается с имени класса лога

found = true; // включить предыдущий блок

continue;

}

} catch (ClassNotFoundException e) {

}

}

return "[]: ";

}

// Получить имя класса из экземпляра класса (объекта).

// Если класс анонимный - вернуть имя внешнего класса.

private static String getClassName(Class clazz) {

if (clazz != null) { // Если объект существует

if (!TextUtils.isEmpty(clazz.getSimpleName())) { // Если его класс не анонимный (у него есть имя),

return clazz.getSimpleName(); // то вернуть это имя

}

return getClassName(clazz.getEnclosingClass()); // Иначе вернуть имя внешнего класса

}

return ""; // Если класса не существует, вернуть пустую строку

}

}