Logcat



В Android SDK входит набор инструментов, предназначенных для отладки. Самый важный инструмент при отладке - это **LogCat** (очень красивое название, которое можно перевести как Логичный Кот). Он отображает сообщения логов (журнал логов), рассылаемые при помощи различных методов.

Рассмотрим на примере. Очень часто программисту нужно вывести куда-то промежуточные результаты, чтобы понять, почему программа не работает. Особо хитрые временно размещают на экране текстовую метку и выводят туда сообщение при помощи метода **textView.setText("Здесь был Васька")**. Но есть способ лучше. В Android есть специальный класс **android.util.Log** для подобных случаев.

Класс **android.util.Log** позволяет разбивать сообщения по категориям в зависимости от важности. Для разбивки по категориям используются специальные методы, которые легко запомнить по первым буквам, указывающие на категорию:

* **Log.e()** - ошибки (error)
* **Log.w()** - предупреждения (warning)
* **Log.i()** - информация (info)
* **Log.d()** - отладка (degub)
* **Log.v()** - подробности (verbose)
* **Log.wtf()** - очень серьезная ошибка! (What a Terrible Failure!, работает начиная с Android 2.2)
* **Log.meow()** - когда жрать дадут? (MEOW!) Недокументированный метод, используйте на свой страх и риск. Работает не на всех устройствах

В первом параметре метода используется строка, называемая тегом. Обычно принято объявлять глобальную статическую строковую переменную **TAG** в начале кода:

private static final String TAG = "MyApp";

Некоторые в сложных проектах используют следующий вариант, чтобы понимать, в каком классе происходит вызов:

private static final String TAG = this.getClass().getSimpleName();

Далее уже в любом месте вашей программы вы вызываете нужный метод журналирования с этим тегом:

Log.i(TAG, "Это мое сообщение для записи в журнале");

Также используется в исключениях:

try {

// ...

} catch (Exception exception) {

Log.e(TAG, "Получено исключение", exception);

}

Пользователи не видят этот журнал. Но, вы, как разработчик, можете увидеть его через программу **LogCat**, доступный через ADB или DDMS.

В **LogCat** вы можете отфильтровать сообщение по заданному тегу, чтобы видеть на экране только свои сообщения. Для этого выберите нужный тип тега из выпадающего списка **Log Level**.

Для отслеживания сообщений с заданным текстом введите в поле поиска нужную строку и нажмите Enter.

**LogCat** также можно запустить из командной строки:

adb logcat

Параметры командной строки смотрите в документации.

Быстрое отключение журналирования

Настоятельно рекомендуется удалять все вызовы **LogCat** в готовых приложениях. Если проект очень большой и вызовы журналирования разбросаны по всем местам кода, то ручное удаление (или комментирование) становится утомительным занятием. Многие разработчики используют следующую хитрость - создают обёртку вокруг вызова методов **LogCat**.

public static final boolean isDebug = false;

public final String TAG = "MyLogger";

public void MyLogger(String statement){

if (isDebug) {

Log.v(TAG, statement);

}

}

Теперь остаётся только присвоить нужное значение переменной **isDebug** перед созданием готового apk-файла для распространения.

Программисты, работающие в Goggle, используют похожий приём. Подробнее можно почитать [здесь](http://gmariotti.blogspot.ru/2013/01/android-logging.html).

Способ устарел. В 17-й версии Android Build Tools появился класс **BuildConfig**, содержащий статическое поле**DEBUG**. Можно проверить следующим образом:

if (BuildConfig.DEBUG) {

// Режим отладки, ведём логи

}

Способ для продвинутых ([отсюда](http://www.dimasokol.ru/to-log-or-not-to-log-logcat-output-for-debug/)) - например, требуется релиз с выводом в лог, или наоборот — debug с выключенным выводом. В этом случае можно создать собственный параметр и добавить его в секцию **buildType**gradle-файла:

buildTypes {

release {

// Здесь настройки релиза

buildConfigField "boolean", "USE\_LOG", "false"

}

debug {

buildConfigField "boolean", "USE\_LOG", "true"

}

// «Релиз с логами» наследуем от обычного релиза

releaseWithLog.initWith(buildTypes.release)

releaseWithLog {

buildConfigField "boolean", "USE\_LOG", "true"

}

}

В этом случае конфигурация **releaseWithLog** будет являться релизной сборкой с ведением логов. Естественно, в коде слегка поменяется проверка:

if (BuildConfig.USE\_LOG) {

// Здесь используем логи

}

LogCat на устройстве

Попался в сети пример для просмотра сообщений **LogCat** на устройстве. С примером не разбирался, оставлю здесь на память.

Разметка

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:orientation="vertical" >

<ListView

android:id="@android:id/list"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content" />

</LinearLayout>

Разметка для элемента списка:

<?xml version="1.0" encoding="utf-8"?>

<TextView xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/txtLogString"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content" />

Добавляем разрешение в манифест:

<uses-permission android:name="android.permission.READ\_LOGS" />

Код для класса:

package ru.alexanderklimov.test;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.List;

import android.app.AlertDialog;

import android.app.ListActivity;

import android.content.Context;

import android.graphics.Color;

import android.os.AsyncTask;

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.ArrayAdapter;

import android.widget.ListView;

import android.widget.TextView;

public class TestActivity extends ListActivity {

private LogStringadapter adapter = null;

private ArrayList<String> logarray = null;

private LogReaderTask logReaderTask = null;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_test);

logarray = new ArrayList<String>();

adapter = new LogStringadapter(this, R.id.txtLogString, logarray);

setListAdapter(adapter);

logReaderTask = new LogReaderTask();

logReaderTask.execute();

}

@Override

protected void onDestroy() {

logReaderTask.stopTask();

super.onDestroy();

}

@Override

protected void onListItemClick(ListView l, View v, int position, long id) {

super.onListItemClick(l, v, position, id);

final AlertDialog.Builder builder = new AlertDialog.Builder(

TestActivity.this);

String text = ((String) ((TextView) v).getText());

builder.setMessage(text);

builder.show();

}

private int getLogColor(String type) {

int color = Color.BLUE;

if (type.equals("D")) {

color = Color.rgb(0, 0, 200);

} else if (type.equals("W")) {

color = Color.rgb(128, 0, 0);

} else if (type.equals("E")) {

color = Color.rgb(255, 0, 0);

;

} else if (type.equals("I")) {

color = Color.rgb(0, 128, 0);

;

}

return color;

}

private class LogStringadapter extends ArrayAdapter<String> {

private List<String> objects = null;

public LogStringadapter(Context context, int textviewid,

List<String> objects) {

super(context, textviewid, objects);

this.objects = objects;

}

@Override

public int getCount() {

return ((null != objects) ? objects.size() : 0);

}

@Override

public long getItemId(int position) {

return position;

}

@Override

public String getItem(int position) {

return ((null != objects) ? objects.get(position) : null);

}

public View getView(int position, View convertView, ViewGroup parent) {

View view = convertView;

if (null == view) {

LayoutInflater vi = (LayoutInflater) TestActivity.this

.getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);

view = vi.inflate(R.layout.logitem, null);

}

String data = objects.get(position);

if (null != data) {

TextView textview = (TextView) view

.findViewById(R.id.txtLogString);

String type = data.substring(0, 1);

String line = data.substring(2);

textview.setText(line);

textview.setTextColor(getLogColor(type));

}

return view;

}

}

private class LogReaderTask extends AsyncTask<Void, String, Void> {

private final String[] LOGCAT\_CMD = new String[] { "logcat" };

private final int BUFFER\_SIZE = 1024;

private boolean isRunning = true;

private Process logprocess = null;

private BufferedReader reader = null;

private String[] line = null;

@Override

protected Void doInBackground(Void... params) {

try {

logprocess = Runtime.getRuntime().exec(LOGCAT\_CMD);

} catch (IOException e) {

e.printStackTrace();

isRunning = false;

}

try {

reader = new BufferedReader(new InputStreamReader(

logprocess.getInputStream()), BUFFER\_SIZE);

} catch (IllegalArgumentException e) {

e.printStackTrace();

isRunning = false;

}

line = new String[1];

try {

while (isRunning) {

line[0] = reader.readLine();

publishProgress(line);

}

} catch (IOException e) {

e.printStackTrace();

isRunning = false;

}

return null;

}

@Override

protected void onCancelled() {

super.onCancelled();

}

@Override

protected void onPreExecute() {

super.onPreExecute();

}

@Override

protected void onPostExecute(Void result) {

super.onPostExecute(result);

}

@Override

protected void onProgressUpdate(String... values) {

super.onProgressUpdate(values);

adapter.add(values[0]);

}

public void stopTask() {

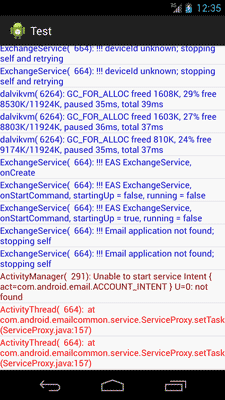
isRunning = false;

logprocess.destroy();

}

}

}



Дополнительное чтение

[Меняем цвет для сообщений в LogCat](http://developer.alexanderklimov.ru/android/tips-studio.php" \l "logcat)