

android日志框架Logger的详细使用（目前最新版2.1.1）

📅 2018-07-27 | 📁 Android | 👁 14

前言

在开发过程中，log日志是每个人多会用上的。平时在开发大项目的时候，几乎每个类多需要加log，然而在发布之前，要求全部删除，真是累。因为系统log暂时没有统一管理的方法。还有就是，特别是在调试的时候，对于线程并发或者子线程的调试，还是加log调试比较精确，用debug工具调试经常和实际不符合。那么，如果使用系统log，调试完又得找出来删除，累++。

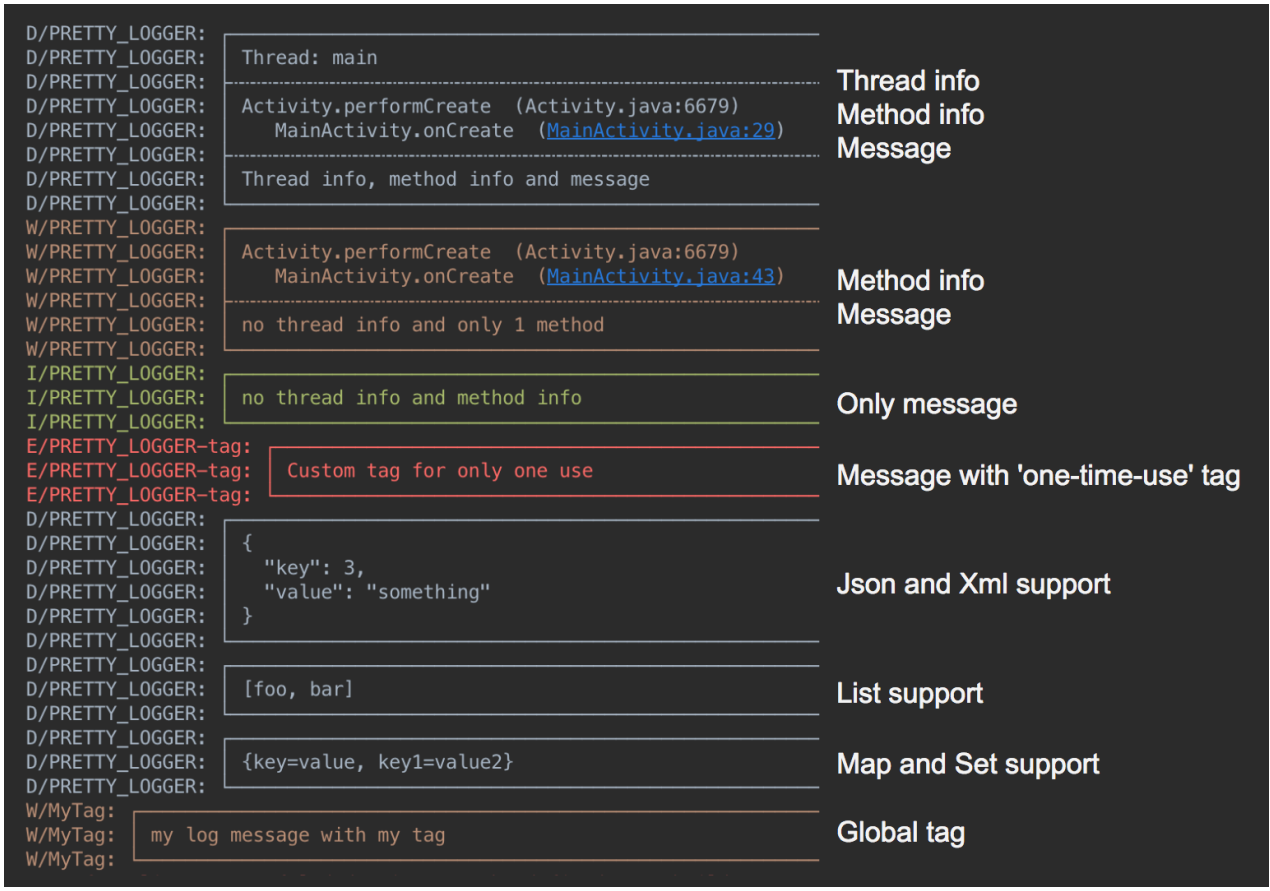
作为程序猿，这种笨方法肯定是会鄙视的。对于这种需求，我们自己封装一个简单的工具包。当然这里介绍的是，github上很多人用log框架：**logger**。

相信很多人多用过，这里给还没有尝试过的人简单介绍下，使用也超级简单。

logger作为调试框架，可以支持多种格式：

- 线程的信息
- 类、方法的信息
- 格式化将 json、xml 输出
- 支持字符串格式参数
- 各种集合输出，list、map、array、set等，（只支持debug输出）
- 支持从日志跳转到源码
-

如下图（引用原作者）：



可以看出，和系统对比，它最大的亮点是优雅的输出log信息，当然同时提供很多其他信息，一目了然，更清晰！

如果你每天被log折磨的生不如死，那么让logger来拯救你吧~

作者：Orhan Obut

github: <https://github.com/orhanobut/logger>

目前，将近7.5k个star让他位列调试框架第二名，屈居facebook的stetho之后,8.5k个star。但这2个调试工具应用场景不用，所以简单的调试，logger是最佳选择了。这里简单介绍下stetho，它是一款提供在Chrome开发者工具上调试Android app的开源框架，可以在Chrome查看数据库，不用想以前那样把数据库导出，然后在用工具查看；配合网络框架可以直接打印查看网络请求的数据，而不需要一个一个添加打印出来，省去很多繁琐的事情。有兴趣的可以了解：

作者：FaceBook

官网地址: <http://facebook.github.io/stetho/>

github <https://github.com/facebook/stetho>

使用Logger

使用框架，是比较简单的，如果可以看英文版，可以直接到<https://github.com/orhanobut/logger>了解。

logger目前的最新版本是2.1.1，网上也很多人使用1.15的版本，如果使用最新版，单独修改配置文件，是有报错的。因为，最新版在初始化这块和原来有了较大的改动。

Breaking changes

- Initialization is changed. No backward compatibility support. Use `Logger.addLogAdapter`
- LogLevel is removed. Use the new `isLoggable` approach

主要是初始化、和控制Log打印这2方面。这里介绍2.1.1的版本使用，所以想升级的也可以往下看看哦。

3步，只要3步，你就学会使用Logger，请：

1.导入依赖

```
compile 'com.orhanobut:logger:2.1.1'
```

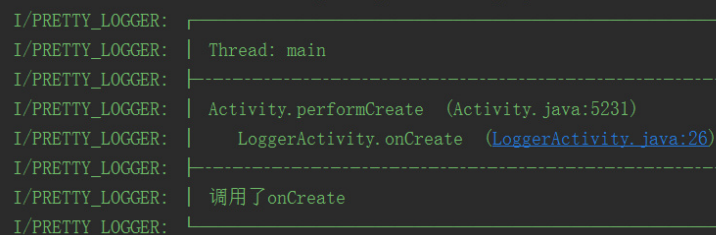
```
Logger.addLogAdapter(new AndroidLogAdapter());
```

注：如果在Application中初始化，记得修改 AndroidManifest.xml 中 application 标签属性，添加 android:name=".MyApplication”，不然不会打印。

3.使用

```
Logger.d("hello");
```

如图：



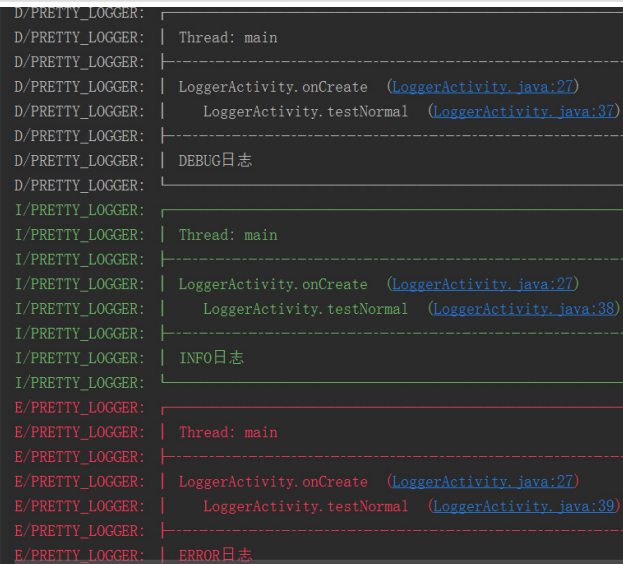
```
I/PRETTY_LOGGER: |
I/PRETTY_LOGGER: | Thread: main
I/PRETTY_LOGGER: |-----
I/PRETTY_LOGGER: | Activity.performCreate (Activity.java:5231)
I/PRETTY_LOGGER: |     LoggerActivity.onCreate (LoggerActivity.java:26)
I/PRETTY_LOGGER: |-----
I/PRETTY_LOGGER: | 调用了onCreate
I/PRETTY_LOGGER: |
```

当然，如果还有追求，继续往下看看它的其他功能。

支持的数据类型

普通类型

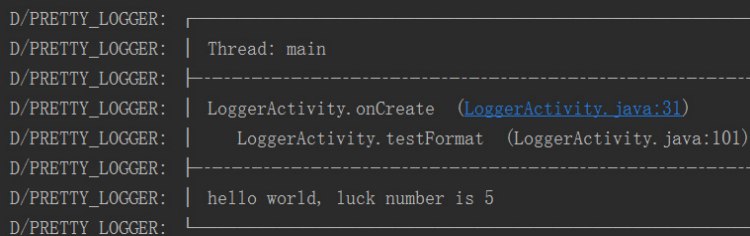
```
public void testNormal(){
    Logger.d("DEBUG日志");
    Logger.i("INFO日志");
    Logger.e("ERROR日志");
}
```



```
D/PRETTY_LOGGER: |
D/PRETTY_LOGGER: | Thread: main
D/PRETTY_LOGGER: |-----|
D/PRETTY_LOGGER: | LoggerActivity.onCreate (LoggerActivity.java:27)
D/PRETTY_LOGGER: |   LoggerActivity.testNormal (LoggerActivity.java:37)
D/PRETTY_LOGGER: |-----|
D/PRETTY_LOGGER: | DEBUG日志
D/PRETTY_LOGGER: |
I/PRETTY_LOGGER: |
I/PRETTY_LOGGER: | Thread: main
I/PRETTY_LOGGER: |-----|
I/PRETTY_LOGGER: | LoggerActivity.onCreate (LoggerActivity.java:27)
I/PRETTY_LOGGER: |   LoggerActivity.testNormal (LoggerActivity.java:38)
I/PRETTY_LOGGER: |-----|
I/PRETTY_LOGGER: | INFO日志
I/PRETTY_LOGGER: |
E/PRETTY_LOGGER: |
E/PRETTY_LOGGER: | Thread: main
E/PRETTY_LOGGER: |-----|
E/PRETTY_LOGGER: | LoggerActivity.onCreate (LoggerActivity.java:27)
E/PRETTY_LOGGER: |   LoggerActivity.testNormal (LoggerActivity.java:39)
E/PRETTY_LOGGER: |-----|
E/PRETTY_LOGGER: | ERROR日志
```

Format类型

```
public void testFormat(){
    Logger.d("hello %s, luck number is %d", "world", 5);
}
```



```
D/PRETTY_LOGGER: |
D/PRETTY_LOGGER: | Thread: main
D/PRETTY_LOGGER: |-----|
D/PRETTY_LOGGER: | LoggerActivity.onCreate (LoggerActivity.java:31)
D/PRETTY_LOGGER: |   LoggerActivity.testFormat (LoggerActivity.java:101)
D/PRETTY_LOGGER: |-----|
D/PRETTY_LOGGER: | hello world, luck number is 5
D/PRETTY_LOGGER: |
```

集合类型

首先看看List:

```
List list = new ArrayList();
list.add("hello");
list.add("world");
Logger.d(list);
```

```
D/PRETTY_LOGGER: |
D/PRETTY_LOGGER: | Thread: main
D/PRETTY_LOGGER: | -----
D/PRETTY_LOGGER: | LoggerActivity.onCreate (LoggerActivity.java:30)
D/PRETTY_LOGGER: |     LoggerActivity.testCollections (LoggerActivity.java:80)
D/PRETTY_LOGGER: | -----
D/PRETTY_LOGGER: | [hello, world]
D/PRETTY_LOGGER: |
```

```
Log.d(TAG, list.toString());
```

首先是调用：

```
@Override public void d(Object object) {
    log(DEBUG, null, Utils.toString(object));
}

public static String toString(Object object) {
    if (object == null) {
        return "null";
    }

    if (!object.getClass().isArray()) {
        return object.toString();
    }

    if (object instanceof boolean[]) {
        return Arrays.toString((boolean[]) object);
    }

    if (object instanceof byte[]) {
        return Arrays.toString((byte[]) object);
    }

    if (object instanceof char[]) {
        return Arrays.toString((char[]) object);
    }

    if (object instanceof short[]) {
        return Arrays.toString((short[]) object);
    }

    if (object instanceof int[]) {
        return Arrays.toString((int[]) object);
    }

    if (object instanceof long[]) {
        return Arrays.toString((long[]) object);
    }

    if (object instanceof float[]) {
```

```
        return Arrays.toString((float[]) object);
    }

    if (object instanceof double[]) {
        return Arrays.toString((double[]) object);
    }

    if (object instanceof Object[]) {
        return Arrays.deepToString((Object[]) object);
    }

    return "Couldn't find a correct type for the object";
}
```

这样就不需要我们自己调用toString(), 直接可以打印了。

其他的数据结构也是一样的, 看图就可以了:

```
public void testCollections(){

    /**
     * Map
     */
    Map map = new HashMap();
    map.put("map1", "hello");
    map.put("map2", "world");
    Logger.d(map);

    /**
     * Set
     */
    Set set = new HashSet();
    set.add("hello");
    set.add("world");
    Logger.d(set);

    /**
     * Array
     */
    String [] strs = {"hello", "world"};
    Logger.d(strs);
}
```

```

D/PRETTY_LOGGER: |
D/PRETTY_LOGGER: | Thread: main
D/PRETTY_LOGGER: |-----
D/PRETTY_LOGGER: | LoggerActivity.onCreate (LoggerActivity.java:30)
D/PRETTY_LOGGER: |   LoggerActivity.testCollections (LoggerActivity.java:86)
D/PRETTY_LOGGER: |-----
D/PRETTY_LOGGER: | {map2=world, map1=hello}
D/PRETTY_LOGGER: |
D/PRETTY_LOGGER: |
D/PRETTY_LOGGER: | Thread: main
D/PRETTY_LOGGER: |-----
D/PRETTY_LOGGER: | LoggerActivity.onCreate (LoggerActivity.java:30)
D/PRETTY_LOGGER: |   LoggerActivity.testCollections (LoggerActivity.java:91)
D/PRETTY_LOGGER: |-----
D/PRETTY_LOGGER: | [hello, world]
D/PRETTY_LOGGER: |
D/PRETTY_LOGGER: |
D/PRETTY_LOGGER: | Thread: main
D/PRETTY_LOGGER: |-----
D/PRETTY_LOGGER: | LoggerActivity.onCreate (LoggerActivity.java:30)
D/PRETTY_LOGGER: |   LoggerActivity.testCollections (LoggerActivity.java:94)
D/PRETTY_LOGGER: |-----
D/PRETTY_LOGGER: | [hello, world]

```

Map类型

Set类型

Array类型

注：打印集合的时候，特别是数组时，前面不要添加其他字符，如：Logger.d(“数组集合”+str);这样会把整体当做一个参数，导致判断错误，不能正常输出数组类型。其次是，作者提示，打印集合时，使用DEBUG等级。

JSON数据

```

public void testJson(){

    String jsonDate = "{ \"id\":859, \"channelnumber\":681, \"bilingual\":0, \"name\":\"beIN Sports 2 English 600K H265\", \"audiotype\":\"MPEG1\", \"videotype\":\"MPEG2\", \"language\":\"EN\", \"country\":\"cn\" }";
    Logger.json(jsonDate);
    Log.d(TAG, jsonDate);
}

```

```

D/PRETTY_LOGGER: |
D/PRETTY_LOGGER: | Thread: main
D/PRETTY_LOGGER: |-----
D/PRETTY_LOGGER: | LoggerActivity.onCreate (LoggerActivity.java:29)
D/PRETTY_LOGGER: |   LoggerActivity.testJson (LoggerActivity.java:61)
D/PRETTY_LOGGER: |-----
D/PRETTY_LOGGER: | {
D/PRETTY_LOGGER: |   "id": 859,
D/PRETTY_LOGGER: |   "audiotype": "MPEG1",
D/PRETTY_LOGGER: |   "name": "beIN Sports 2 English 600K H265",
D/PRETTY_LOGGER: |   "callsign": "beIN Sports 2 English 600K H265",
D/PRETTY_LOGGER: |   "videotype": "MPEG2",
D/PRETTY_LOGGER: |   "language": "EN",
D/PRETTY_LOGGER: |   "bilingual": 0,
D/PRETTY_LOGGER: |   "channelnumber": 681,
D/PRETTY_LOGGER: |   "country": "cn"
D/PRETTY_LOGGER: | }
D/PRETTY_LOGGER: |
D/Log: {"id":859,"channelnumber":681,"bilingual":0,"name":"beIN Sports 2 English 600K H265",
      "language":"EN",
      "videotype":"MPEG2","audiotype":"MPEG1","callsign":"beIN Sports 2 English 600K H265","country":"cn"};

```

Logger框架打印

系统Log打印

XML数据

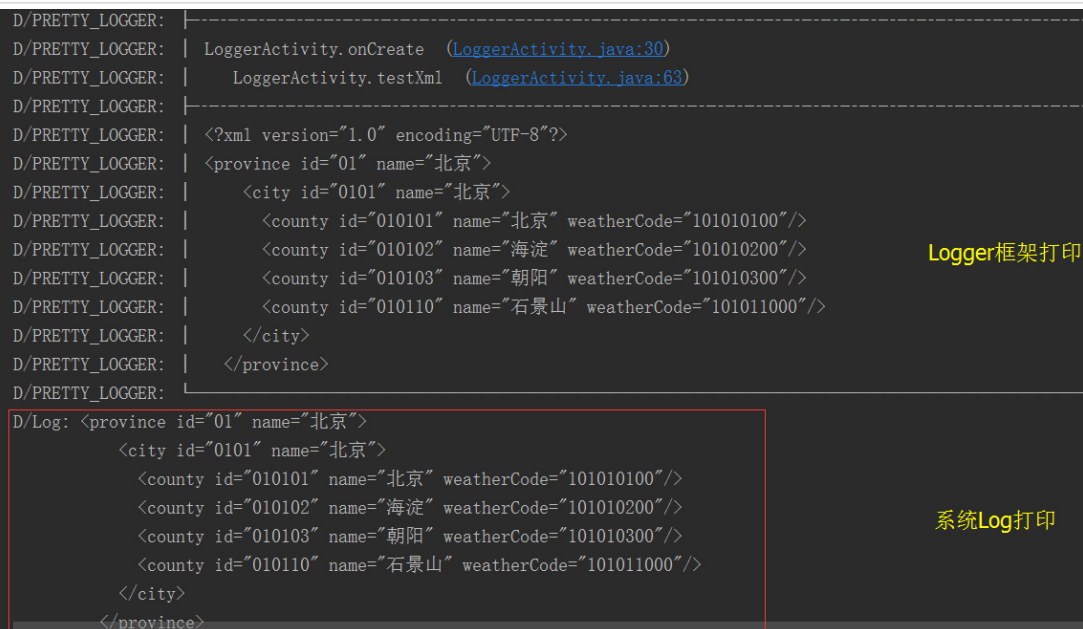
```

public void testXml() {

    String xmlDate = "<province id=\"01\" name=\"北京\"> \n" +
        "    <city id=\"0101\" name=\"北京\"> \n" +
        "        <county id=\"010101\" name=\"北京\" weatherCode=\"101010100\"> \n" +
        "        <county id=\"010102\" name=\"海淀\" weatherCode=\"101010200\"> \n" +
        "        <county id=\"010103\" name=\"朝阳\" weatherCode=\"101010300\"> \n" +
        "        <county id=\"010110\" name=\"石景山\" weatherCode=\"101011000\"> \n" +
        "    </city> \n" +
        "</province> ";

    Logger.xml(xmlDate);
    Log.d(TAG, xmlDate);
}

```



```

D/PRETTY_LOGGER: |-----|
D/PRETTY_LOGGER: | LoggerActivity.onCreate (LoggerActivity.java:30)
D/PRETTY_LOGGER: | LoggerActivity.testXml (LoggerActivity.java:63)
D/PRETTY_LOGGER: |-----|
D/PRETTY_LOGGER: | <?xml version="1.0" encoding="UTF-8"?>
D/PRETTY_LOGGER: | <province id="01" name="北京">
D/PRETTY_LOGGER: |     <city id="0101" name="北京">
D/PRETTY_LOGGER: |         <county id="010101" name="北京" weatherCode="101010100"/>
D/PRETTY_LOGGER: |         <county id="010102" name="海淀" weatherCode="101010200"/>
D/PRETTY_LOGGER: |         <county id="010103" name="朝阳" weatherCode="101010300"/>
D/PRETTY_LOGGER: |         <county id="010110" name="石景山" weatherCode="101011000"/>
D/PRETTY_LOGGER: |     </city>
D/PRETTY_LOGGER: | </province>
D/PRETTY_LOGGER: |-----|
D/Log: <province id="01" name="北京">
    <city id="0101" name="北京">
        <county id="010101" name="北京" weatherCode="101010100"/>
        <county id="010102" name="海淀" weatherCode="101010200"/>
        <county id="010103" name="朝阳" weatherCode="101010300"/>
        <county id="010110" name="石景山" weatherCode="101011000"/>
    </city>
</province>

```

修改默认配置

最新版修改配置，和旧版也是有不同，如下：

```

FormatStrategy formatStrategy = PrettyFormatStrategy.newBuilder()
    .showThreadInfo(false)    // (可选) 是否显示线程信息。 默认值为true
    .methodCount(2)          // (可选) 要显示的方法行数。 默认2
    .methodOffset(7)         // (可选) 设置调用堆栈的函数偏移值，0的话则从打印该Log
    .logStrategy(customLog)  // (可选) 更改要打印的日志策略。 默认LogCat
    .tag("MyTAG")             // (可选) 每个日志的全局标记。 默认PRETTY_LOGGER (如上)
    .build();
Logger.addLogAdapter(new AndroidLogAdapter(formatStrategy));

```

具体配置大家可以自行去实践下。一般默认就可以了，这里说下，修改全局的TAG，和局部的TAG。

1.修改全局的TAG，在初始化的application中修改，如：


```

FormatStrategy formatStrategy = PrettyFormatStrategy.newBuilder()
    .tag("MyTAG")
    .build();
Logger.addLogAdapter(new AndroidLogAdapter(formatStrategy));

```

The screenshot shows the logcat interface with a red box highlighting the 'MyTAG' logs. The logs are as follows:

```

D/MyTAG: |
D/MyTAG: | Thread: main
D/MyTAG: | -----
D/MyTAG: | LoggerActivity.onCreate (LoggerActivity.java:27)
D/MyTAG: |   LoggerActivity.testNormal (LoggerActivity.java:37)
D/MyTAG: | -----
D/MyTAG: | DEBUG日志
D/MyTAG: |
I/MyTAG: |
I/MyTAG: | Thread: main
I/MyTAG: | -----
I/MyTAG: | LoggerActivity.onCreate (LoggerActivity.java:27)
I/MyTAG: |   LoggerActivity.testNormal (LoggerActivity.java:38)
I/MyTAG: | -----
I/MyTAG: | INFO日志
I/MyTAG: |

```

2.修改局部的TAG，哪里打印就在哪里修改，只对当前打印有效，如：

```

public void testNormal(){
    Logger.t("hello").d("DEBUG日志");
    Logger.i("INFO日志");
}

```

The screenshot shows the logcat interface with a red box highlighting the 'PRETTY_LOGGER' logs. The logs are as follows:

```

D/PRETTY_LOGGER-hello: |
D/PRETTY_LOGGER-hello: | Thread: main
D/PRETTY_LOGGER-hello: | -----
D/PRETTY_LOGGER-hello: | LoggerActivity.onCreate (LoggerActivity.java:27)
D/PRETTY_LOGGER-hello: |   LoggerActivity.testNormal (LoggerActivity.java:36)
D/PRETTY_LOGGER-hello: | -----
D/PRETTY_LOGGER-hello: | DEBUG日志
D/PRETTY_LOGGER-hello: |
I/PRETTY_LOGGER: |
I/PRETTY_LOGGER: | Thread: main
I/PRETTY_LOGGER: | -----
I/PRETTY_LOGGER: | LoggerActivity.onCreate (LoggerActivity.java:27)
I/PRETTY_LOGGER: |   LoggerActivity.testNormal (LoggerActivity.java:38)
I/PRETTY_LOGGER: | -----
I/PRETTY_LOGGER: | INFO日志
I/PRETTY_LOGGER: |

```

注：如果尝试用局部的方法，在application初始化时修改。当然，这是有效果的，但只能用一次，其他打印还是会用全局的，可自行测试。

控制打印开关

这个功能是最喜欢的了，当我们发布软件，或者太多的log影响性能，那么就可以关闭logger打印，在application初始化的地方：

```

Logger.addLogAdapter(new AndroidLogAdapter() {

```

```
@Override public boolean isLoggable(int priority, String tag) {
    return BuildConfig.DEBUG;
}

});
```

通过适配器控制打印，只要覆盖isLoggable()方法，返回BuildConfig.DEBUG即可。这样子，log就不再打印出来了。

注：返回值导入的包是：import com.orhanobut.logger.BuildConfig; 有时顺手导入：com.gotechcn.frameworks，那就没有效果。

保存log到文件

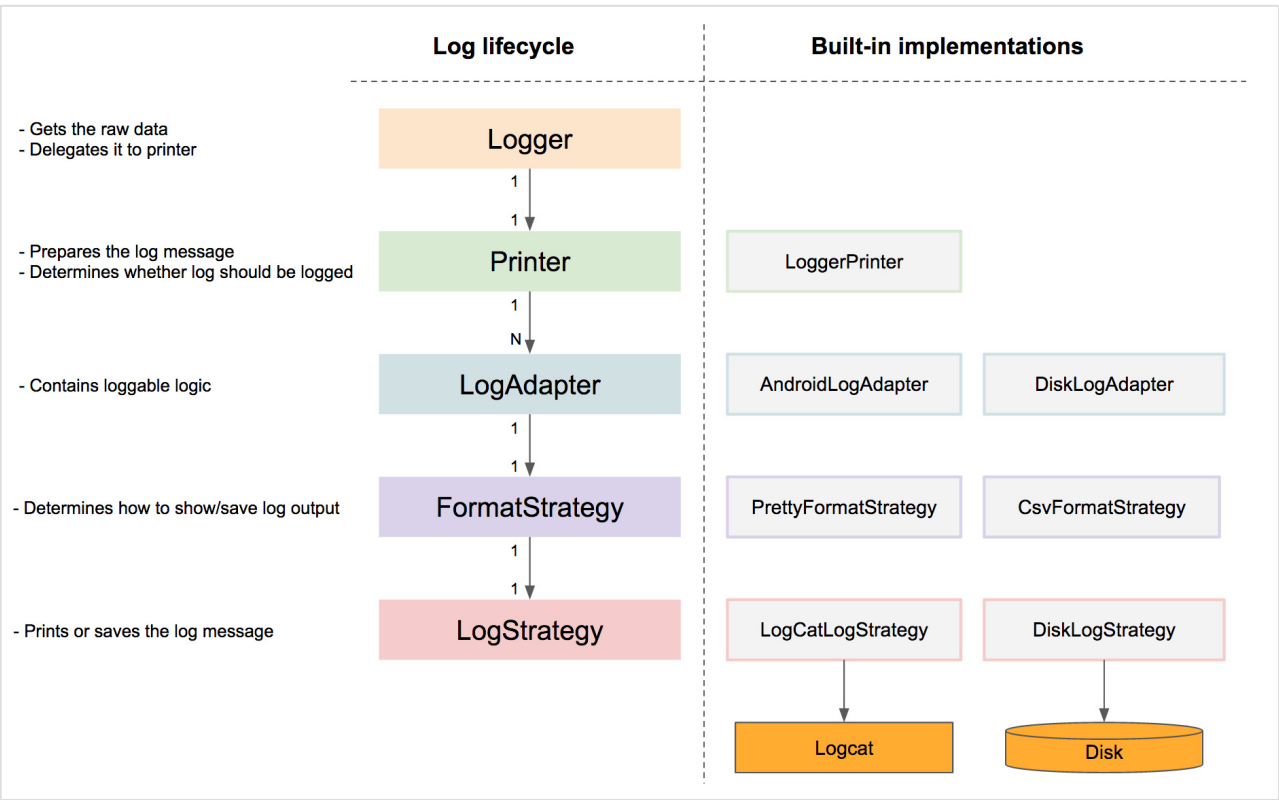
```
Logger.addLogAdapter(new DiskLogAdapter());
```

通过打印，找到保存的路径： /storage/emulated/0

但手机里面就是没有这个文件夹，不知道什么原因，真机和模拟机多没有，知道可以留言噢，感谢！

工作流程原理图

logg框架的整体流程如图（作者原图）：



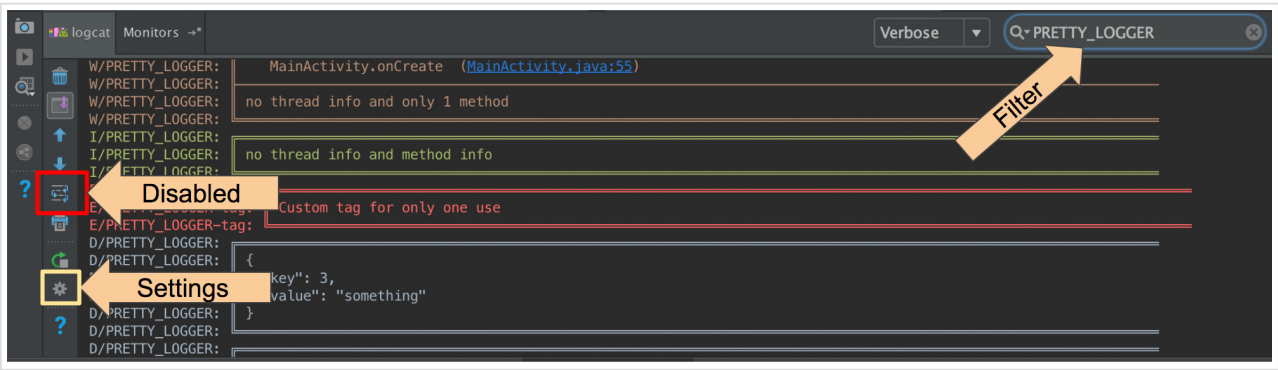
其他

对Log信息的设置与过滤：

Filter：编辑默认或者自定义的标签，过滤标签以外的信息；

Disable：设置user soft wraps，即设置换行

Setting：设置log头部的信息，比如时间、线程PID、包名、TAG等开关设置



好了，对于Logger框架的介绍就到这里了。有时会更新，记得使用时看看GitHub升级了没有，同时在使用是否和以前兼容。对于一些其他的方法，可以直接看源码，具体是干什么用的。

坚持原创技术分享，您的支持将鼓励我继续创作！

打赏

[# blog](#) [# markdown](#) [# Android](#)

◀ Lottie Android For Animation

0条评论 zhangmiao

zm

推荐 分享

评分最高



开始讨论...

来做第一个留言的人吧！

订阅 在您的网站上使用 Disqus添加 Disqus添加 Disqus 隐私政策隐私政策隐私

© 2018 ZhangMiao

由 [Hexo](#) 强力驱动 | 主题 — [NexT.Pisces](#) v5.1.4