UTLog设计及使用介绍

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 修订者 | 修订内容 | 时间 |
| 1.0 | 陈航 | 文档初始化 | 2016.06.16 |
| 1.0 | 陈航 | 添加UTLog简介、初始化配置、使用介绍、模块设计 | 2016.06.16 |

目录

[1. UTLog简介 3](#_Toc454089274)

[为何需要重新设计Log接口 3](#_Toc454089275)

[UTLog的基本功能 3](#_Toc454089276)

[2. UTLog初始化配置 3](#_Toc454089277)

[1) 使用默认的配置 3](#_Toc454089278)

[2) 设置自定义Printer 4](#_Toc454089279)

[3) 设置自定义的log记录器 5](#_Toc454089280)

[4) 设置自定义的内存捕捉器 5](#_Toc454089281)

[5) 设置线程信息捕捉器 6](#_Toc454089282)

[6) 设置JsonFormatter 6](#_Toc454089283)

[7) 设置XmlFormatter 7](#_Toc454089284)

[8) 设置UTLog的Configuration 7](#_Toc454089285)

[3. UTLog的使用 8](#_Toc454089286)

[1) 输出不同级别的调试信息 8](#_Toc454089287)

[2) 查看当前调用栈 9](#_Toc454089288)

[3) 查看当前内存信息 9](#_Toc454089289)

[4) 查看当前线程信息 9](#_Toc454089290)

[5) 输出额外的title 9](#_Toc454089291)

[6) 组合输出信息 10](#_Toc454089292)

[4. UTLog模块设计 10](#_Toc454089293)

## UTLog简介

UTLog是一个为方便调试而设计的Log工具，建议Android工程中，把使用Android原生接口Log的地方，替换为统一的接口UTLog。UTLog的使用和原生接口的使用方式很相似，并做了一些功能增强。

### 为何需要重新设计Log接口

使用自定义的Log接口，可以方便开发者跟踪软件运行状况，如果用android原生的Log接口，在不能使用adb工具的情况下，如何查看软件运行情况将会变得很困难。使用自定义的Log接口后，可以解决这个问题，如把log保存在本地文件，方便查看，甚至可以把log信息上报给后台服务器，方便实时分析软件情况。

自定义的Log组件比原生的Log更为强大，可以做更多方便的操作，如设定软件输出log级别，在release的软件中，屏蔽debug信息等。

### UTLog的基本功能

UTLog包含的基本功能如下：

* 为每条log指定级别，分别有VERBOSE、DEBUG、INFO、WARN、ERROR、ASSERT五个输出级别，分别对应android的log级别。
* 全局打开/关闭log输出
* 设定显示的log级别，低级别的log信息会被过滤
* 查看log语句的调用栈
* 查看当前线程信息
* 查看内存状态
* 保存log到本地文件，方便跟踪软件运行情况
* 为log信息添加额外的title信息
* 可以定制比原生log更为美观的输出，如为json、xml等数据添加格式

## UTLog初始化配置

使用UTLog之前，请务必先执行初始化代码。一般可以把初始化代码放在Application类的onCreate()方法中。UTLog采用了非常灵活的配置方式，里面的组件使用者都可以重新定制。比如重新定制文件保存器AsyncFileWriter、重新定制log输出器Printer、重新定制JsonFormatter等。以下将会详细介绍如何初始化UTLog

### 使用默认的配置

如果没有特别的需求，UTLog已经有各组件的默认实现，只需要一行代码就可以初始化了。如：

**public void** easyUTLogSetup(Context context) {  
 UTLog.*setup*(**new** LoggerBuilder(context)); //使用默认的配置  
}

### 设置自定义Printer

**public void** setupUTLog(Context context) {  
  
 LoggerBuilder builder = **new** LoggerBuilder(context);  
  
 builder.setPrinter(**new** Printer() {  
 @Override  
 **public void** print(AdvancedLogInfo logInfo) {  
 Log.*d*(logInfo.getTag(), "-----------------------");  
 Log.*d*(logInfo.getTag(), logInfo.getMessage());  
 Log.*d*(logInfo.getTag(), "-----------------------");  
 }  
 });  
  
 UTLog.*setup*(builder);  
}

这样，每条输出的log就都会被两行分割线包裹在中间，通过这个方法可以定制各种美观的输出界面。UTLog已经内置了两种Printer实现，分别是：PrettyPrinter和NormalPrinter

### 设置自定义的log记录器

**public void** setupUTLog(Context context) {  
  
 LoggerBuilder builder = **new** LoggerBuilder(context);  
  
 //实现你自己的消息记录器，在AdvancedLogInfo的write方法中把Log写入文件系统  
 builder.setLogWriter(**new** AsyncFileWriter<AdvancedLogInfo>() {  
 @Override  
 **public void** write(AdvancedLogInfo logInfo) {  
  
 }  
  
 @Override  
 **public void** close() **throws** IOException {  
  
 }  
 });  
  
 UTLog.*setup*(builder);  
}

UTLog已经内置了一个AsyncFileWriter的实现，若用户没有通过setLogWriter方法覆盖默认的Writer的话，log会默认写入/Android/data/[app\_package\_name]/cache/log/目录中，并为每次启动应用创建一个文件，如logcat\_20160616114650.txt

### 设置自定义的内存捕捉器

**public void** setupUTLog(Context context) {  
  
 LoggerBuilder builder = **new** LoggerBuilder(context);  
  
 //获取内存信息  
 builder.setMemoryGetter(**new** MemoryGetter() {  
 @Override  
 **public** MemoryState getMemory() {  
 MemoryState memoryState = **new** MemoryState();  
 // *TODO: 2016/6/16 设置内存信息* **return** memoryState;  
 }  
 });  
  
 UTLog.*setup*(builder);  
}

### 设置线程信息捕捉器

**public void** setupUTLog(Context context) {  
  
 LoggerBuilder builder = **new** LoggerBuilder(context);  
  
 //返回自定义的Thread信息,此回调会在打印语句所在线程调用  
 builder.setThreadGetter(**new** ThreadGetter() {  
 @Override  
 **public** String getThreadInfo() {  
 **return** "\*\*\* " + Thread.*currentThread*().getName() + " \*\*\*";  
 }  
 });  
  
 UTLog.*setup*(builder);  
}

### 设置JsonFormatter

**public void** setupUTLog(Context context) {  
  
 LoggerBuilder builder = **new** LoggerBuilder(context);  
  
 //格式化json字符串，让输出更美观  
 builder.setJsonFormatter(**new** StringFormatter() {  
 @Override  
 **public** String format(String string) **throws** Exception {  
 **return** "return your formatted json here";  
 }  
 });  
  
 UTLog.*setup*(builder);  
}

### 设置XmlFormatter

**public void** setupUTLog(Context context) {  
  
 LoggerBuilder builder = **new** LoggerBuilder(context);  
  
 //格式化xml字符串，让输出更美观  
 builder.setXmlFormatter(**new** StringFormatter() {  
 @Override  
 **public** String format(String string) **throws** Exception {  
 **return** "return your formatted xml here";  
 }  
 });  
 UTLog.*setup*(builder);  
}

### 设置UTLog的Configuration

**public void** setupUTLog(Context context) {  
  
 LoggerBuilder builder = **new** LoggerBuilder(context);  
  
 //配置Log  
 builder.setConfiguration(**new** LogConfiguration(**true**, LogLevel.*DEBUG*, "")); // 设置是否输出、打印级别、全局的tag前缀  
   
 UTLog.*setup*(builder);  
}

UTLog还为Configuration提供了额外的接口，方便在初始化之后更改其设置，配置接口如下：

**private void** changeConfiguration() {  
  
 //设置log开关  
 UTLog.*setDebugEnabled*(**true**);  
 UTLog.*setDebugEnabled*(**false**);  
  
 //设置log输出级别  
 UTLog.*setLogLevel*(LogLevel.*VERBOSE*);  
 UTLog.*setLogLevel*(LogLevel.*DEBUG*);  
 UTLog.*setLogLevel*(LogLevel.*INFO*);  
 UTLog.*setLogLevel*(LogLevel.*WARN*);  
 UTLog.*setLogLevel*(LogLevel.*ERROR*);  
 UTLog.*setLogLevel*(LogLevel.*ASSERT*);  
  
 //设置tag前缀  
 UTLog.*setTagPrefix*("My-Prefix");  
 UTLog.*setTagPrefix*("");  
}

## UTLog的使用

### 输出不同级别的调试信息

**private void** logDemo() {  
  
 UTLog.*v*("tag", "message");  
 UTLog.*d*("tag", "message");  
 UTLog.*i*("tag", "message");  
 UTLog.*w*("tag", "message");  
 UTLog.*e*("tag", "message");  
 UTLog.*wtf*("tag", "message");  
}

### 查看当前调用栈

**private void** logDemo() {  
  
 UTLog.*watchStack*().d("tag", "message");  
}

### 查看当前内存信息

**private void** logDemo() {  
  
 UTLog.*watchMemory*().d("tag", "message");  
}

### 查看当前线程信息

**private void** logDemo() {  
  
 UTLog.*watchStack*().d("tag", "message");  
}

### 输出额外的title

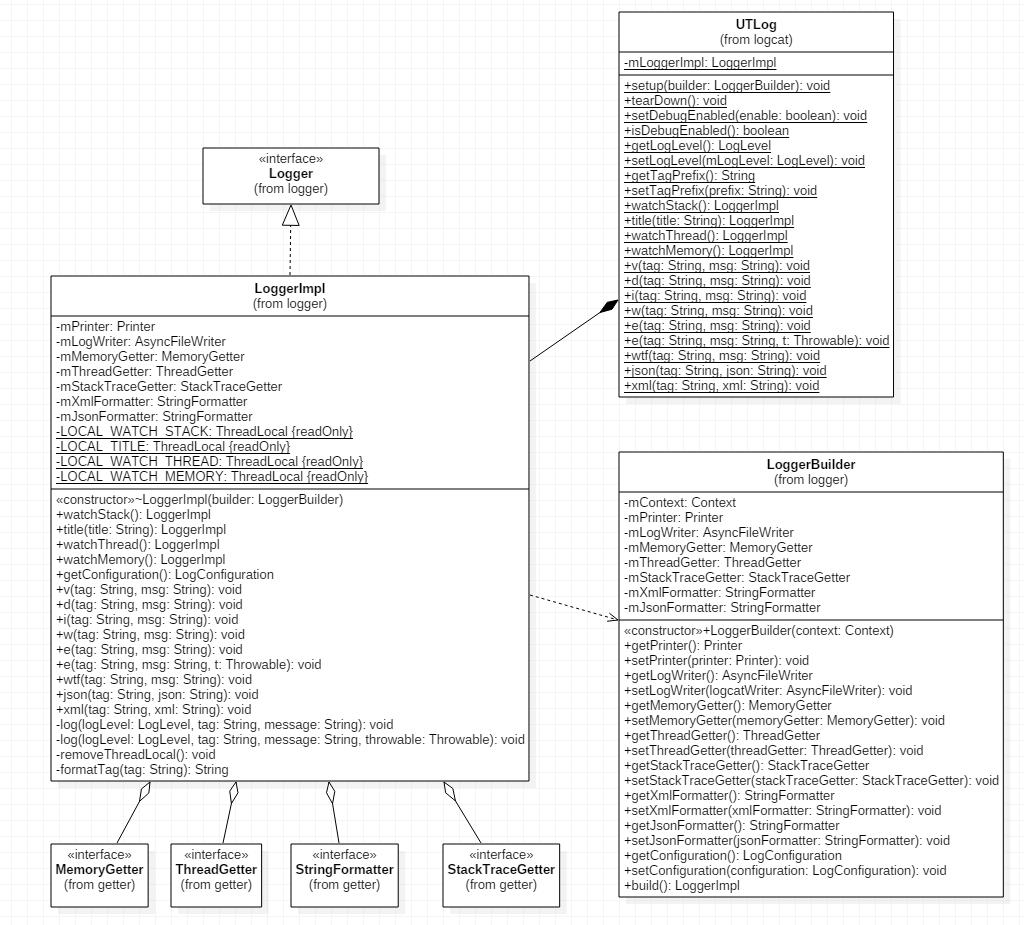
**private void** logDemo() {  
  
 UTLog.*title*("YourTitle").d("tag", "message");  
}

### 组合输出信息

**private void** logDemo() {  
  
 UTLog.*watchStack*().watchThread().watchMemory().title(("YourTitle")).d("tag", "message");  
}

## UTLog模块设计

如图：



## Logcat捕捉

Logcat mLogcat = **new** Logcat();  
mLogcat.start(**new** Logcat.Callback() {  
 @Override  
 **public void** onLog(LogInfo log) {  
 // 在这里处理接收到的log，注意，这个回调是运行在子线程的，不能操作UI  
 }  
});

这样，就能用logcat命令接收到Android的logcat输出。主要是用到了“logcat –v time”命令，效果如下：

