Android 日志保存策略

一、更详细的日志信息

既然决定自定义一个 log，那我们就可以让它显示更多的信息，如线程信息：threadId，threadName 等：

```
private String getFunctionName() {
        StackTraceElement[] sts = Thread.currentThread().getStackTrace();
        if (sts == null) {
            return null;
        }

        for (StackTraceElement st : sts) {
            if (st.isNativeMethod()) {
                continue;
            }
            if (st.getClassName().equals(Thread.class.getName())) {
                continue;
            }
            if (st.getClassName().equals(this.getClass().getName())) {
                continue;
            }

            Thread t = Thread.currentThread();
            return "[T(id:" + t.getId() +
                    ", name:" + t.getName() +
                    ", priority:" + t.getPriority() +
                    ", groupName:" + t.getThreadGroup().getName() +
                    "): " + st.getFileName() + ":"
                    + st.getLineNumber() + " " + st.getMethodName() + " ]";
        }
        return "";
    }
```

StackTrace（堆栈轨迹）存放的就是方法调用栈的信息，我们从中获取方法执行的线程相关的信息，以及执行的方法名称等。这些信息能帮助我们更好的查找问题之所在。

```
private void logPrint(int logLevel, Object msg) {
        if (isDebug) {
            String name = getFunctionName();
            customTag = TextUtils.isEmpty(customTag) ? defaultTag : customTag;
```

```
            Log.println(logLevel, customTag, name + " - " + msg);
        }
    }
```

使用 Log.println 方法打印相关信息即可。

二、日志保存策略

后端的人在测试的时候会遇到 BUG，有时候不知道到底是前端出了问题还是后端的问题，为了更好更快速的定位，后端应该知道前端的日志保存在哪里。这就需要我们制定一个日志保存策略。（即使要上传日志，也应该先保存成文件再上传文件，不然每一条日志调用一次接口，接口的压力会很大，很不合理）

由于保存日志的过程是个耗时过程，我们需要开启线程去保存。但是日志产生的频率可能很高，又不能采用一般的线程去处理，太多的线程也会损耗性能。所以我们应该考虑队列的形式保存日志，然后一条一条的去保存。

```
    public void initSaveStrategy(Context context) {
        if (saveLogStrategy != null || !isDebug) {
            return;
        }
        final int MAX_BYTES = 1024 * 1024;
        String diskPath = Environment.getExternalStorageDirectory().getAbsolutePath();
        File cacheFile = context.getCacheDir();
        if (cacheFile != null) {
            diskPath = cacheFile.getAbsolutePath();
        }
        String folder = diskPath + File.separatorChar + "log";
        HandlerThread ht = new HandlerThread("SohuLiveLogger." + folder);
        ht.start();
        Handler handler = new SaveLogStrategy.WriteHandler(ht.getLooper(), folder,
MAX_BYTES);
        saveLogStrategy = new SaveLogStrategy(handler);
    }
   public static class WriteHandler extends Handler {

        private final String folder;
        private final int maxFileSize;

        WriteHandler(@NonNull Looper looper, @NonNull String folder, int maxFileSize) {
            super(checkNotNull(looper));
            this.folder = checkNotNull(folder);
            this.maxFileSize = maxFileSize;
        }
```

```
        @Override
        public void handleMessage(@NonNull Message msg) {
            String content = (String) msg.obj;
            FileWriter fileWriter = null;
            File logFile = getLogFile(folder, "logs");
            try {
                fileWriter = new FileWriter(logFile, true);
                writeLog(fileWriter, content);
                fileWriter.flush();
                fileWriter.close();
            } catch (IOException e) {
                if (fileWriter != null) {
                    try {
                        fileWriter.flush();
                        fileWriter.close();
                    } catch (IOException e1) {

                    }
                }
            }
        }
```

我们使用 HandlerThread 来处理这个任务。HandlerThread 是一个可以使用 handler 的 Thread。当我们把消息保存到消息队列中去之后会在线程中去处理，又能保证不会产生很多线程。其实这里也可以使用 instentservice 实现，这个服务适合量大而不太耗时的任务。

最后在一个方法中统一打印和保存即可：

```
    private void logPrint(int logLevel, Object msg) {
        if (isDebug) {
            String name = getFunctionName();
            if (saveLogStrategy != null) {
                saveLogStrategy.log(Log.ERROR, customTag, name + " - " + msg);
            }
            Log.println(logLevel, customTag, name + " - " + msg);
        }
    }
```

自定义的 log 策略还是比较简单，主要就是这个思想：打印日志信息详细，保存要采用队列的形式。一下是全部代码：

```
public class Logger {
```

```
    public final static String tag = "";
    private static SaveLogStrategy saveLogStrategy;
    private final static boolean logFlag = true;
    private static Logger logger;
    private int logLevel = Log.VERBOSE;
    private static boolean isDebug = BuildConfig.DEBUG;
    private String customTag = null;

    private Logger(String customTag) {
        this.customTag = customTag;
    }

    public void initSaveStrategy(Context context) {
        if (saveLogStrategy != null || !isDebug) {
            return;
        }
        final int MAX_BYTES = 1024 * 1024;
        String diskPath = Environment.getExternalStorageDirectory().getAbsolutePath();
        File cacheFile = context.getCacheDir();
        if (cacheFile != null) {
            diskPath = cacheFile.getAbsolutePath();
        }
        String folder = diskPath + File.separatorChar + "log";
        HandlerThread ht = new HandlerThread("Logger." + folder);
        ht.start();
        Handler handler = new SaveLogStrategy.WriteHandler(ht.getLooper(), folder,
MAX_BYTES);
        saveLogStrategy = new SaveLogStrategy(handler);
    }

    public static Logger getLogger(String tag) {
        if (logger == null) {
            logger = new Logger(tag);
        }
        return logger;
    }

    public static Logger getLogger() {
        if (logger == null) {
            logger = new Logger(tag);
        }
        return logger;
    }
```

```
/**
 * Verbose(2) 级别日志
 *
 * @param str String
 */
public void v(Object str) {
        logLevel = Log.VERBOSE;
        logPrint(logLevel, str);
}

/**
 * Debug(3) 级别日志
 *
 * @param str String
 */
public void d(Object str) {
        logLevel = Log.DEBUG;
        logPrint(logLevel, str);
}

/**
 * Info(4) 级别日志
 *
 * @param str String
 */
public void i(Object str) {
        logLevel = Log.INFO;
        logPrint(logLevel, str);
}

/**
 * Warn(5) 级别日志
 *
 * @param str String
 */
public void w(Object str) {
        logLevel = Log.WARN;
        logPrint(logLevel, str);
}

/**
 * Error(6) 级别日志
 *
 * @param str String
```

```
 */
public void e(Object str) {
        logLevel = Log.ERROR;
        logPrint(logLevel, str);
}

private void logPrint(int logLevel, Object msg) {
        if (isDebug) {
                String name = getFunctionName();
                if (saveLogStrategy != null) {
                        saveLogStrategy.log(Log.ERROR, customTag, name + " - " + msg);
                }
                Log.println(logLevel, customTag, name + " - " + msg);
        }
}

/**
 * 获取当前方法名
 *
 * @return 方法名
 */
private String getFunctionName() {
        StackTraceElement[] sts = Thread.currentThread().getStackTrace();
        if (sts == null) {
                return null;
        }

        for (StackTraceElement st : sts) {
                if (st.isNativeMethod()) {
                        continue;
                }
                if (st.getClassName().equals(Thread.class.getName())) {
                        continue;
                }
                if (st.getClassName().equals(this.getClass().getName())) {
                        continue;
                }

                Thread t = Thread.currentThread();
                return "[Thread(id:" + t.getId() +
                                ", name:" + t.getName() +
                                ", priority:" + t.getPriority() +
                                ", groupName:" + t.getThreadGroup().getName() +
                                "): " + st.getFileName() + ":"
```

```
                + st.getLineNumber() + " " + st.getMethodName() + " ]";
        }
        return "";
    }
}

public class SaveLogStrategy {

    @NonNull
    private final Handler handler;

    public SaveLogStrategy(@NonNull Handler handler) {
        this.handler = checkNotNull(handler);
    }

    public void log(int level, @Nullable String tag, @NonNull String message) {
        checkNotNull(message);
        handler.sendMessage(handler.obtainMessage(level, message));
    }

    static class WriteHandler extends Handler {

        private final String folder;
        private final int maxFileSize;

        WriteHandler(@NonNull Looper looper, @NonNull String folder, int maxFileSize) {
            super(checkNotNull(looper));
            this.folder = checkNotNull(folder);
            this.maxFileSize = maxFileSize;
        }

        @SuppressWarnings("checkstyle:emptyblock")
        @Override
        public void handleMessage(@NonNull Message msg) {
            String content = (String) msg.obj;
            FileWriter fileWriter = null;
            File logFile = getLogFile(folder, "logs");

            try {
                fileWriter = new FileWriter(logFile, true);

                writeLog(fileWriter, content);

                fileWriter.flush();
```

```
                fileWriter.close();
            } catch (IOException e) {
                if (fileWriter != null) {
                    try {
                        fileWriter.flush();
                        fileWriter.close();
                    } catch (IOException e1) {

                    }
                }
            }
        }

        private void writeLog(@NonNull FileWriter fileWriter, @NonNull String content) throws
IOException {
            checkNotNull(fileWriter);
            checkNotNull(content);
            fileWriter.append("\n").append(content);
        }

        private File getLogFile(@NonNull String folderName, @NonNull String fileName) {
            checkNotNull(folderName);
            checkNotNull(fileName);

            File folder = new File(folderName);
            if (!folder.exists()) {
                if (!folder.mkdirs()) {
                    Log.println(Log.ERROR, "saveLog", "文件未创建成功，可能是读写权限
没给");
                }
            }

            int newFileCount = 0;
            File newFile;
            File existingFile = null;

            newFile = new File(folder, String.format("%s_%s.txt", fileName, newFileCount));
            while (newFile.exists()) {
                existingFile = newFile;
                newFileCount++;
                newFile = new File(folder, String.format("%s_%s.txt", fileName,
newFileCount));
            }
```

```
                if (existingFile != null) {
                    if (existingFile.length() >= maxFileSize) {
                        return newFile;
                    }
                    return existingFile;
                }

                return newFile;
            }
        }
    }
```