作业回顾

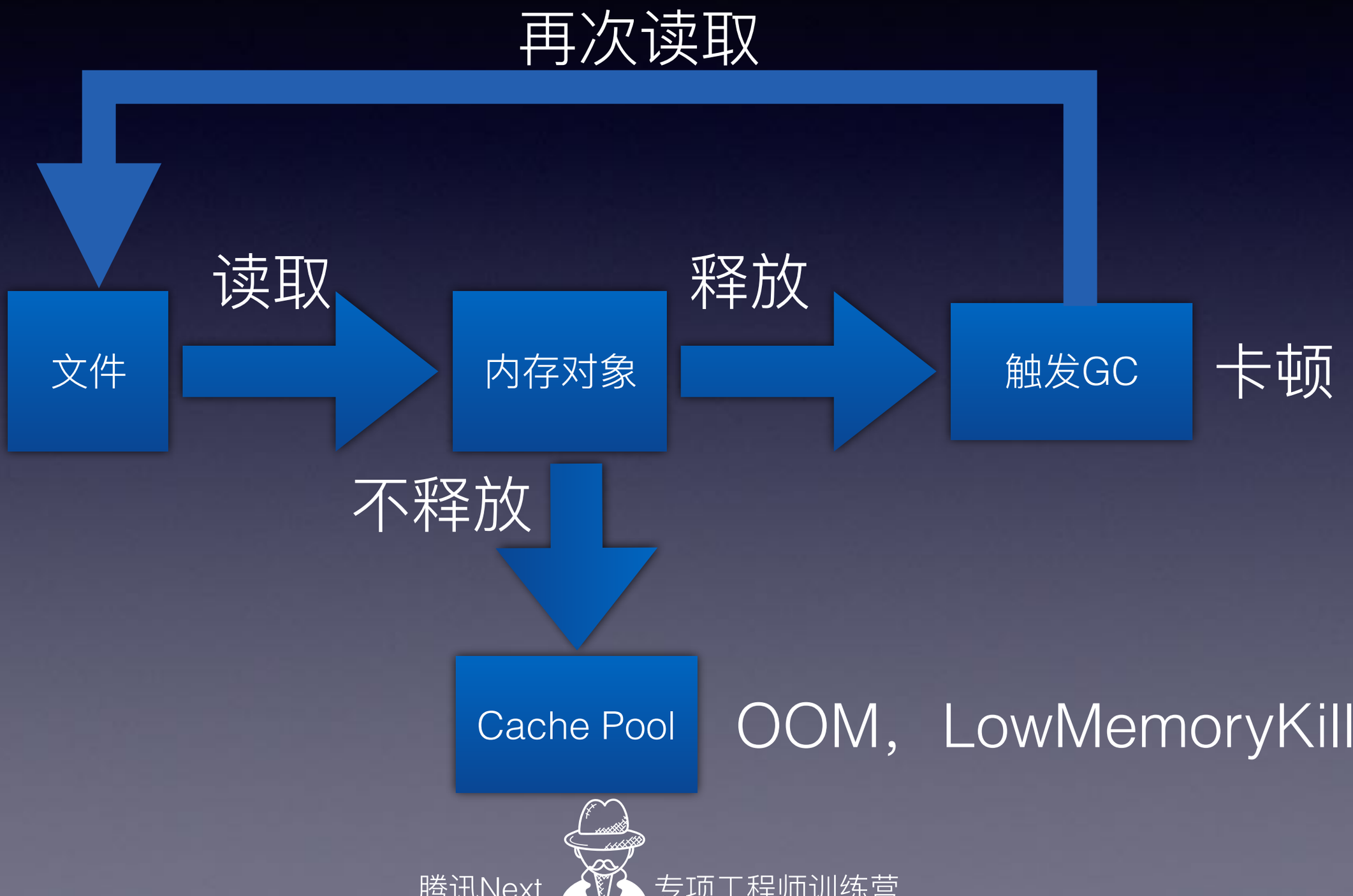
- Handler导致的泄漏
- 怎么解决?

看菜吃饭的资源类性能：磁盘

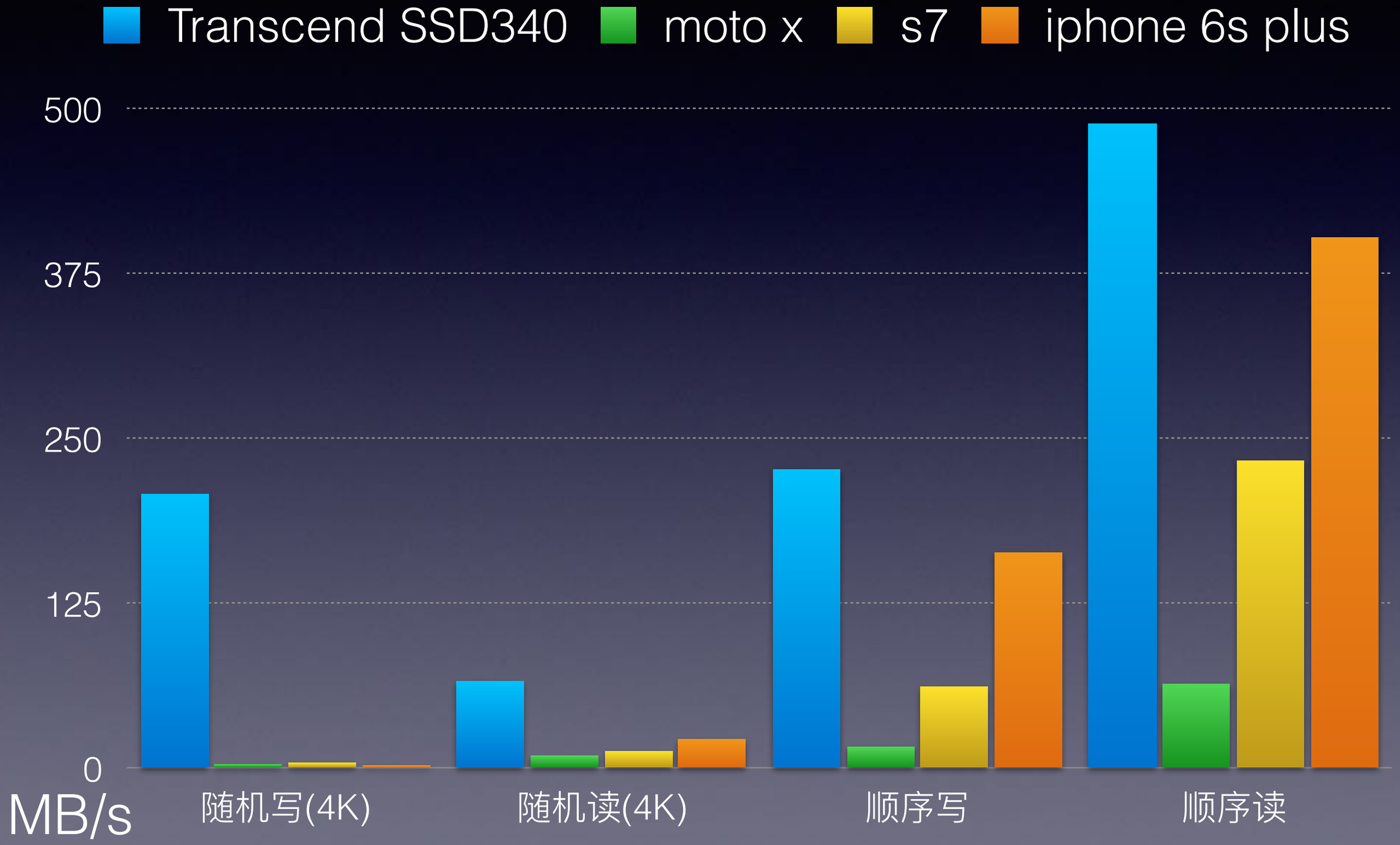
腾讯 victorhuang

腾讯Next 专项工程师训练营

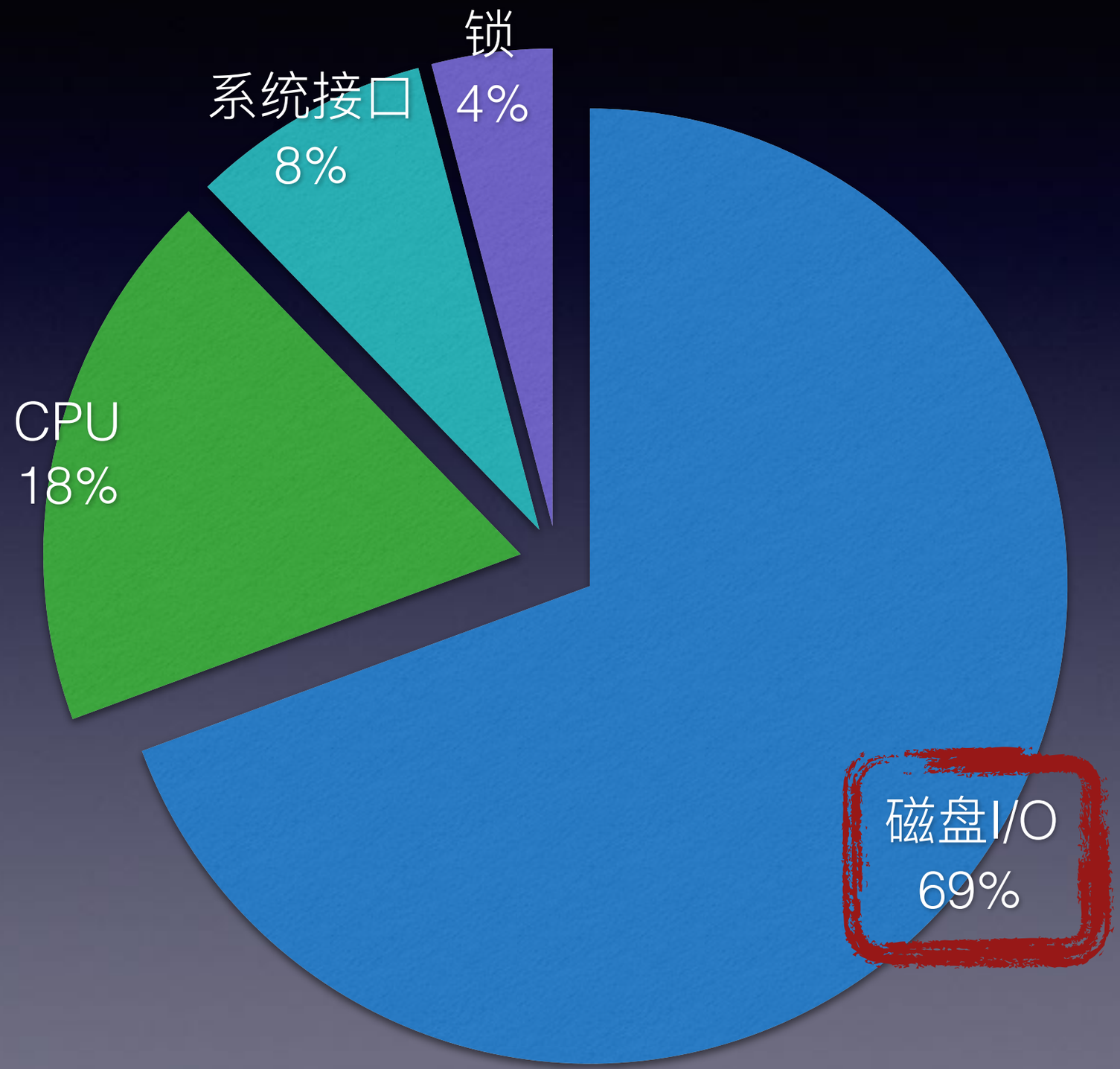
磁盘I/O与内存的关系



存储性能


Transcend SSD340
moto x
s7
iphone 6s plus
500
375
250
125
0
MB/s
随机写(4K)
随机读(4K)
顺序写
顺序读

外网卡顿情况分析



阶段性优化效果显著

落地以来，接入9个项目，820多个BUG，解决率85%

10%

30%

23%

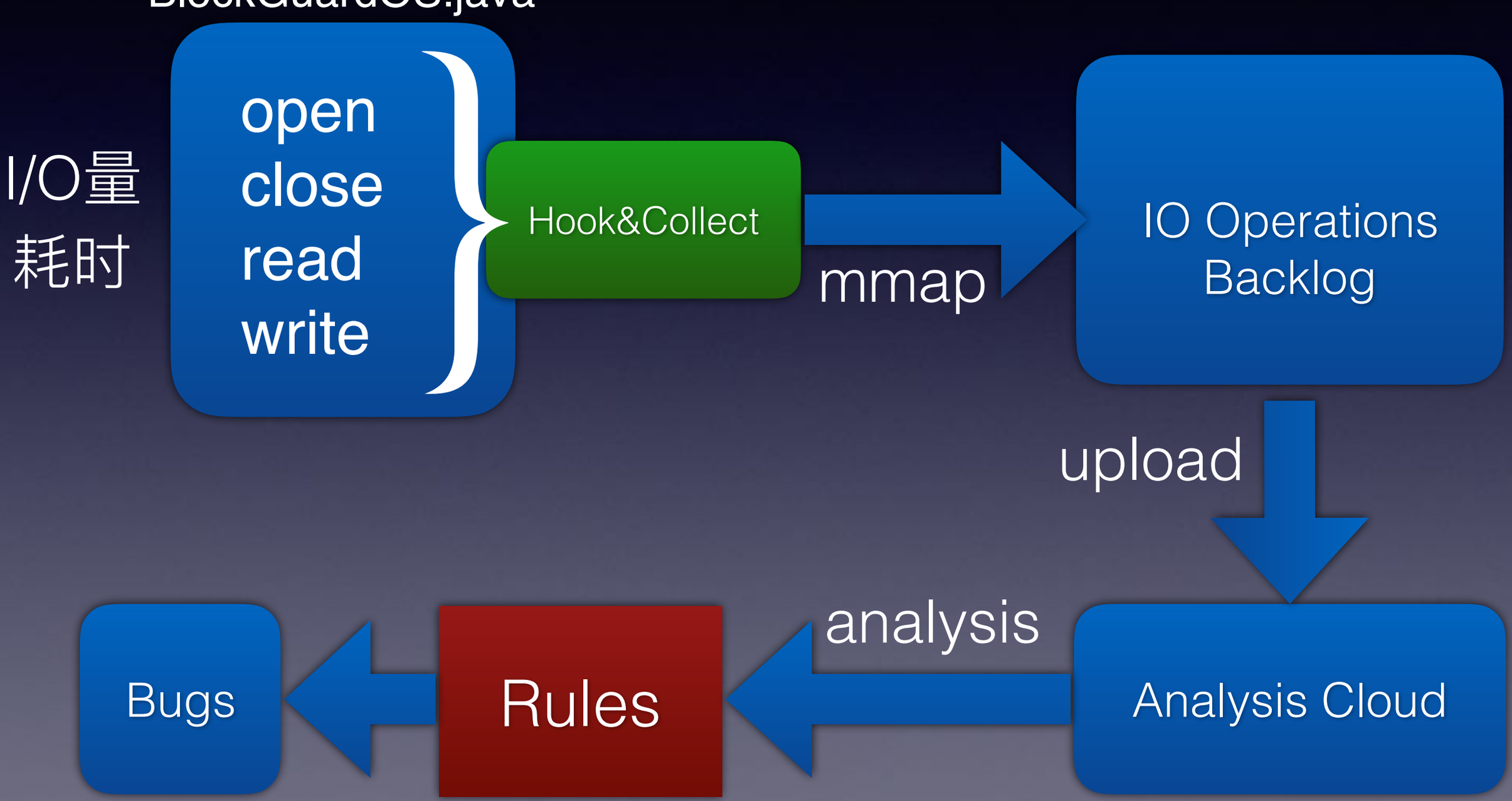
下面从三个方面来说说I/O性能

提供发现问题的全新角度

提供定位问题的全新角度

提供解决问题的解决方案

提供一个发现问题的全新角度



案例：Bitmap.decodeFile

文件格式：PNG
耗时：49ms

会有什么问题呢?

I/O操作大小：12KB

I/O操作次数：108次

案例：Bitmap.decodeFile

	调用方法	耗时(ms)	读磁盘次数
i9300 (android 4.3)	decodeStream	25	2
	decodeFile	26	1
MI2S (android 5.0)	decodeStream	18	2
	decodeFile	49	108

小结：前面没有说过的思路

通过hook系统方法的接口，来增加帮助发现问题的信息

案例：读取8条群消息

场景：群聊天窗口有时卡一下，有时卡几下

分析：

用StrictMode来排查，没有主线程I/O

用TraceView来排查，主线程没有耗时特别长的函数

I/O操作大小：486MB

这是个数据库I/O问题，参考上面的思路

有什么办法增加我们分析的信息？

提供一个定位问题的全新角度

- SQLiteDatabase接口调用信息
- 数据库page类型及对应的数据量
- SQL语句对应的执行计划

SQLiteDatabase接口调用信息

Hook SqlDatabase.java

executeUpdateDelete()
executeInsert()
execute()
cursor.fillWindow()
openDatabase()
beginTransaction()
endTransaction()
enableWAL()

1. SQL语句
2. 是否使用事务
3. 是否打开WAL
4. 是否重复连接数据库

数据库page类型及对应的数据量

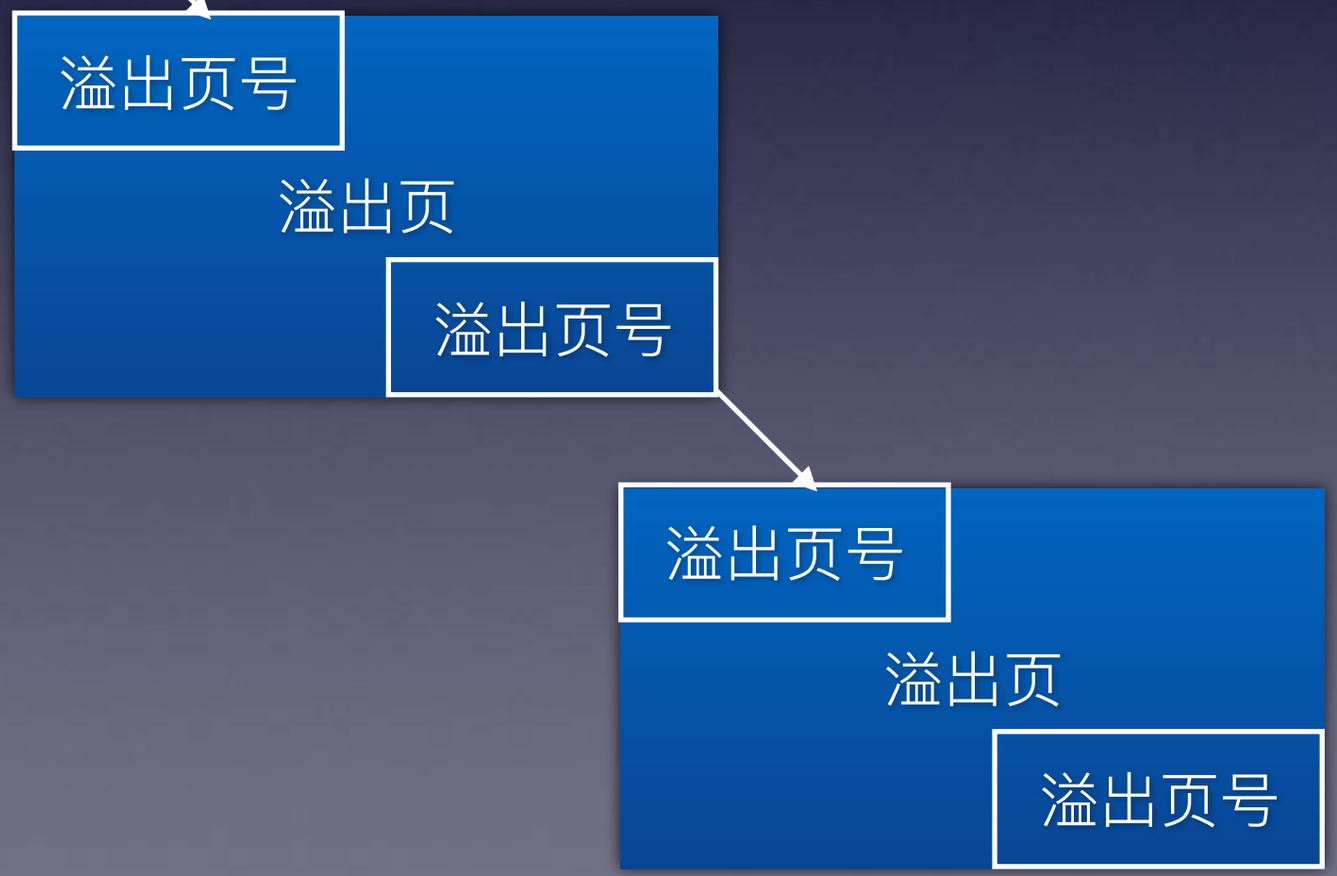
0x0d表示b+tree叶子页

0x05是b+tree内部页

0x0a是b-tree叶子页

0x02是b-tree内部页

参考：https://www.cnblogs.com/jiangcsu/p/6569045.html



案例：查看8条群消息
487MB

/data/user/0/com.tencent.mobileqq/databases/2872971611.db	(	select * from (select _id# extraflag# frienduin#	1626	487708

883ms

SQL语句信息：
select * from (select from table1 order by A) left join table2 on B where C order by D limit 6288,8

240ms

select * from (select from table1 where C order by A limit 6288,8) left join table2;

8条

案例 好友动态Feeds读取

- 数据库page类型及对应的数据量
- I/O Read：72MB

stackTrac	fileName	tableInte	tableLeaf	indexInte	indexLeaf	overFlow	readSize	writeSize
com.qzone	/data/dat	0	821	0	420	0	152	4812
com.qzone	/data/dat	1	45	0	0	236	1128	0
com.qzone	/data/dat	0	924	0	0	15433	65428	0
com.qzone	/data/dat	0	0	0	1	0	4	0
com.qzone	/data/dat				1			0

924 叶子节点

？？？？？？？

15433个溢出页

分页不合理

案例 好友动态Feeds读取

SQLiteDatabase接口调用信息

SELECT * FROM A_0_3_1575000559 ORDER BY B DESC, C DESC LIMIT 266,10

这个SQL语句什么问题?

执行计划

SCAN TABLE A_0_3_1575000559 (~1000000 rows)

USE TEMP B-TREE FOR ORDER BY

创建B与C的联合索引:
336ms -> 34ms
76MB -> 6.5MB

小结

- 提供发现问题的全新角度
- 提供定位问题的全新角度
- 提供同一世界的全新方案

- I/O数据量,I/O操作次数
- SQLiteDatabase调用信息
- 数据库page类型及数据量
- 执行计划

提供全新的解決方案

那些函数搬运工？

BUG+解决方案

问题	解决方案
主线程I/O	SharePreference尽量用apply，少用commit
I/O操作次数过多	ObjectOutputStream，ZipOutputStream需要配合BufferOutputStream使用; 尽量使用decodeStream
I/O操作量过大	合理使用索引；利用内存缓存，避免重复读写
溢出页过多	setPageSize + db.execSQL("VACUUM;")才能重建分页
减少多表操作	尽量不要使用AutoIncreate

最后

- 真实压力场景缺失
  多纬度**定位信息**直达问题原因
- 问题随机出现
  通过不随机的**性能消耗**来发现随机的耗时问题
- 解决方法单一
  通过同一世界的**解决方案**，让开发不再是函数搬运工

推荐文章

- https://www.sqlite.org/pragma.html#pragma_cache_size
- http://codificar.com.br/blog/sqlite-optimization-faq/
- http://www.mobile-open.com/2016/926123.html
- http://fanli7.net/a/bianchengyuyan/C__/20121111/251406.html
- http://blog.csdn.net/littletigerat/article/details/5412572
- http://blog.devart.com/increasing-sqlite-performance.html

作业

优化copyFile函数
并提交代码diff信息，并证明性能更佳

```
    private static App mInstance;
+--  9 lines: @Override----------------
         *  You should not init your a
        if (LeakCanary.isInAnalyzerPro
            return;
        }
        LeakCanary.install(this);
```

```
    private static App mInstance;
+ +--  9 lines: @Override-----------------------------------------
         *  You should not init your app in this process. */
        if (LeakCanary.isInAnalyzerProcess(this)) {
            return;
        }
        LeakCanary.install(this);

        /**
         * Bug 1

        try {
            Thread.sleep(5*1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
         */
        //Log.d("abis", Arrays.toString(Build.SUPPORTED_ABIS));
        QAPM.setProperty(QAPM.PropertyKeyAppInstance, this);
        QAPM.setProperty(QAPM.PropertyKeyAppId, "33e15431-1024").se
        QAPM.setProperty(QAPM.PropertyKeyUserId, "11223344");
        QAPM.setProperty(QAPM.PropertyKeyLogLevel, QAPM.LevelDebug)
        QAPM.beginScene(QAPM.SCENE_ALL, QAPM.ModeAll);
```

diff信息

优化前： 20ms
优化后： 10ms

举证性能更佳
附加图或者表