Android性能优化

tag	author	date	history
Android	caizhenghe	2018-03-18	create doc

内存泄漏优化

> 内存泄漏的优化分为两方面，一方面是在开发过程中避免写出有内存泄漏的代码，另一方面是借助工具找出并解决潜在的内存泄漏。

泄漏场景

静态成员

静态变量生命周期比Activity长，如果一个静态变量持有Activity的引用，会导致Activity销毁后依然无法被释放，从而引发内存泄漏。

```
// MainActivity.java
private static Context sContext = this;
private static View sView = new View(this);
```

单例模式

类似于静态变量，Activity被一个单例模式的对象持有，一直无法被释放。

```
public class Single {
    public interface DataListner {
        void onDataChanged();
    }
    // Single内部维护着一个监听器列表，每个Listener都是一个接口
    private List<DataListner> mListeners = new ArrayList<DataListner>
();

    public void registerListener(DataListner listener) {
        // Activity实现Activity接口并注册到Single中。
        if (!mListeners.contains(listener))
            mListeners.add(listener);
    }
```

```
        public void unRegisterListener(DataListner listener) {
            // 反注册Activity
            mListeners.remove(listener);
        }

        private static class SingleHolder {
            // 实现lazy-loading单例模式
            public static final Single INSTANCE = new Single();
        }

        public static Single getInstance() {
            return SingleHolder.INSTANCE;
        }

    }
```

若Activity注册监听事件之后没有及时的反注册，就会被单例Single对象一直持有，最终导致泄漏。

- 定义SingleHolder的目的是实现单例模式的Lazy-loading
- 非静态内部类无法定义静态成员（final static修饰的基础数据类型除外），详情查看

属性动画

Handler（非静态内部类）

分析工具

LeakCanary

布局优化

merge标签

ViewStub

绘制优化

View的onDraw

响应速度优化

ANR日志分析

APP启动速度优化

ListView和Bitmap优化

ListView加载图片卡顿

Bitmap OOM

线程优化