Memory Leaks In Android

By 技术小黑屋

About me

- 技术小黑屋(droidyue.com)博主
- @Flipboard中国

GDG && Me

- It’s pure and with no ads.
- It’s full of ganhuo
- 3rd time to participate


Purjoy
纯享


Purjoy
纯享

微信扫一扫关注

What's Memory Leaks

A memory leak occurs when object references that are no longer needed are unnecessarily maintained.

Why We need to resolve leaks

- **Leaks put more pressure on the Heap since it reduces the available memory.**
- **It degrades system performance over time such as GC**
- **It may lead to the Out Of Memory Error**

concepts

- **Objects In Java**
- **What is GC**
- **How GC works**
- **Reference types in Java**

Objects In Java

- Objects are allocated in Java Heap when we use new instruction
- Objects could be referenced by Local variables, Instance variables and Class variables
- Usually objects referenced by Class variables lives the longest,Instance variables the second, Local variables the last.
- Dead objects will be collected by GC

What is GC

- **GC is short for Garbage Collection**
- **In CPP, we need to remove the objects manually.**
- **It runs in JVM to reclaim unused objects**
- **With GC, we don't need to manage the allocated memory**

How GC mark objects as garbage

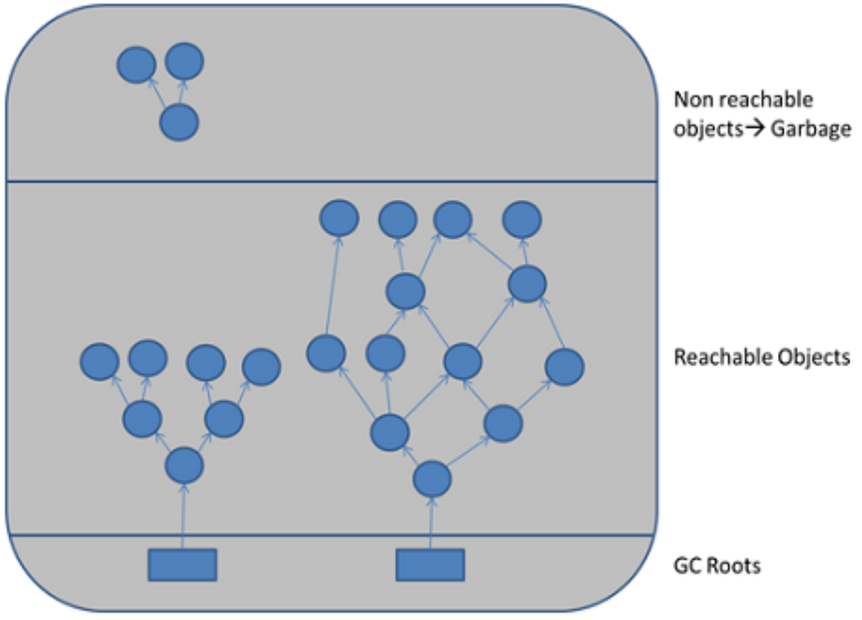
- **Reference Counting**
- **GC Roots Tracing**

Reference Counting

- **Every Object has a reference counter.**
- **When an object is referenced（assigned to variables, passed into a method）, the reference counter +1.**
- **When an object is no longer referenced(leave it's scope), the reference counter -1.**
- **Circular references could not be resolved by this collector.**
- **Reference Counting is seldom implemented in major JVMs**

GC Roots Tracing

- **The tracing starts from GC roots**
- **Every object strongly-reachable to GC roots will be marked as alive.**
- **Those unreachable objects should be garbage to be collected.**
- **This is the most popular implementation for modern JVMs.**


Non reachable
objects→ Garbage
Reachable Objects
GC Roots

GC roots

- **Classes loaded by system class loader**
- **living threads**
- **variables and parameters in java method stacks**
- **variables and parameters in native method stacks**
- **.etc**

Reference Types in Java

- **Strong Reference: the normal one. e.g.String s = "android";**
- **SoftReference: collected when the memory is tensive**
- **WeakReference: collected when the GC occurs.**
- **Phantom Reference: used with reference queue to tell whether the object is collected.**

Leaks In Android

- **It seems occurs more frequently in Android**
- **As the available memory for each App is restricted, it could result in OOMs more easily.**
- **Leaks will disappear as soon as the App exits(the process is terminated)**

Cases In Android

- **keeping a long-lived(static) reference to a (Activity)Context**
- **Forget to unregister the listeners**
- **Caused by Non-static Inner Class**

Keep a long-lived (static)reference to a Context(Activity)

- An Activity instance hold a lot of views. A View can hold many bitmaps.
- Activity should be destroyed as when it's not be needed any more.
- If an activity is leaked, a lot of memory won't not be reclaimed.

```
public class AppSettings {
    private Context mAppContext;
    private static AppSettings sInstance = new AppSettings();

    //some other codes
    public static AppSettings getInstance() {
      return sInstance;
    }

    public final void setup(Context context) {
        mAppContext = context;
    }
}
```

```
public class AppSettings {
    private Context mAppContext;
    private static AppSettings sInstance = new AppSettings();

    //some other codes
    public static AppSettings getInstance() {
      return sInstance;
    }

    public final void setup(Context context) {
        mAppContext = context.getApplicationContext();
    }
}
```

Leaks caused by forgetting unregistration

```
public class MainActivity extends AppCompatActivity implements OnNetworkChangedListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        NetworkManager.getInstance().registerListener(this);
    }

    @Override
    public void onNetworkUp() {

    }

    @Override
    public void onNetworkDown() {

    }
}
```

Leaks caused by forgetting unregistration

```
public class MainActivity extends AppCompatActivity implements OnNetworkChangedListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        NetworkManager.getInstance().registerListener(this);
    }

    @Override
    public void onNetworkUp() {

    }

    @Override
    public void onNetworkDown() {

    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        NetworkManager.getInstance().unregisterListener(this);
    }

}
```

Leaks caused by non-static inner class

```
public class SensorListenerActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        sensorManager.registerListener(new SensorListener() {
            @Override
            public void onSensorChanged(int sensor, float[] values) {

            }

            @Override
            public void onAccuracyChanged(int sensor, int accuracy) {

            }
        }, SensorManager.SENSOR_ALL);
    }
}
```

Memory Leaks Scenarios

- **Passing Activity as the Context of Singleton**
- **Use Activity.getSystemService() to obtain some services.(PowerManager)**
- **Resources(closeable objects) not closed.**
- **Handler could cause memory leaks.**
- **Delayed tasks may cause memory leaks.**
- **And other scenarios**

Ways to resolve

- **Remove the unnecessary reference relationships**
- **Use WeakReference or other variants**

Detect and Analyze

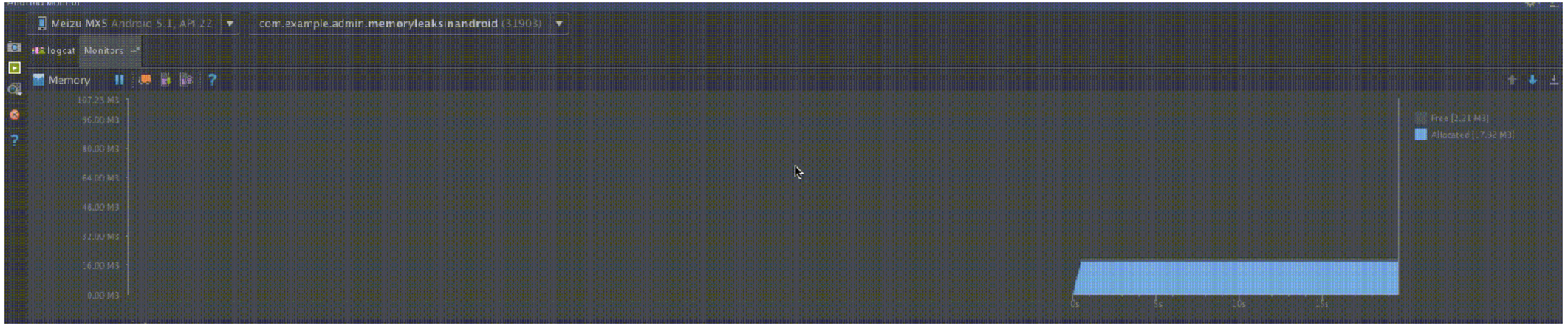
- **StrictMode**
- **Android Memory Monitors**
- **LeakCanary**
- **Heap Dump**
- **MAT**

StrictMode

- **StrictMode is a developer tool which detects Thread and VM violations.**
- **The detectAll or detectActivityLeaks could detect the leaks of Activities.**
- **By using setClassInstanceLimit, we could detect memory leaks of other objects.**
- **StrictMode could only detects phenomenon, but could not provide more details.**

Android Memory Monitors

- **It's bundled in Android Studio**
- **We can see the changes of heap as time passes**
- **If the heap grows and never goes down, there will be much chance that there are memory leaks.**


Meizu MX5 Android 5.1, API 22
com.example.admin.memoryleaksinandroid (31903)
logcat
Monitors
Memory
Free [2.21 MB]
Allocated [7.32 MB]

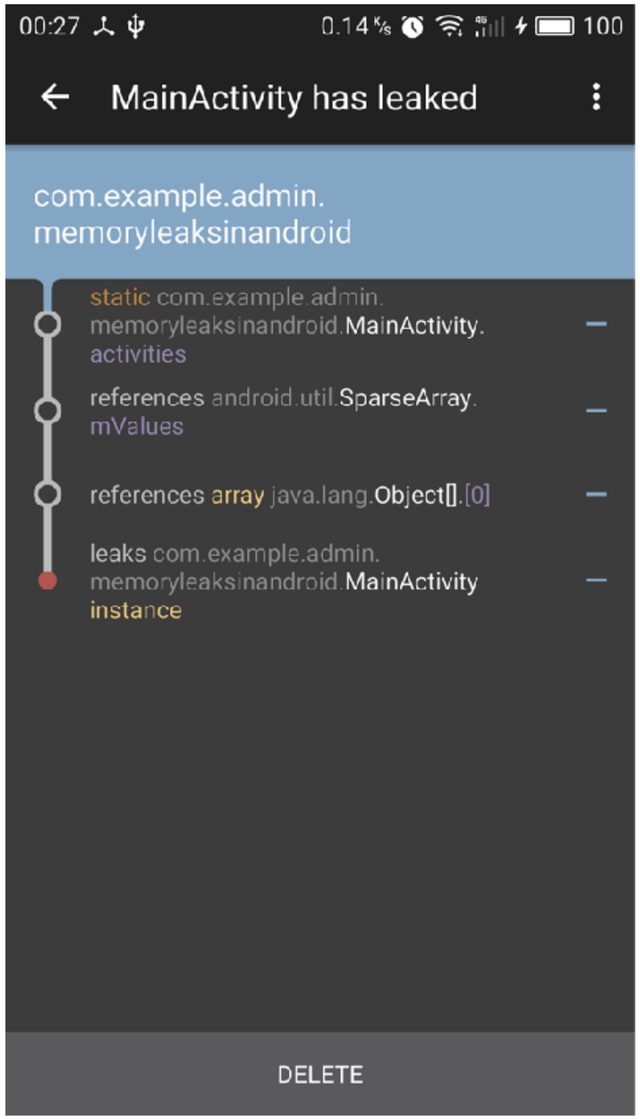
LeakCanary

- **A memory leak detection library for Android and Java.**
- **It’s powered by Square**
- **LeakCanary will automatically show a notification when an activity memory leak is detected in your debug build.**

```
dependencies {
   debugCompile 'com.squareup.leakcanary:leakcanary-android:1.5'
   releaseCompile 'com.squareup.leakcanary:leakcanary-android-no-op:1.5'
   testCompile 'com.squareup.leakcanary:leakcanary-android-no-op:1.5'
 }
```

```
public class ExampleApplication extends Application {

  @Override public void onCreate() {
    super.onCreate();
    if (LeakCanary.isInAnalyzerProcess(this)) {
      // This process is dedicated to LeakCanary for heap analysis.
      // You should not init your app in this process.
      return;
    }
    LeakCanary.install(this);
    // Normal app init code...
  }
}
```

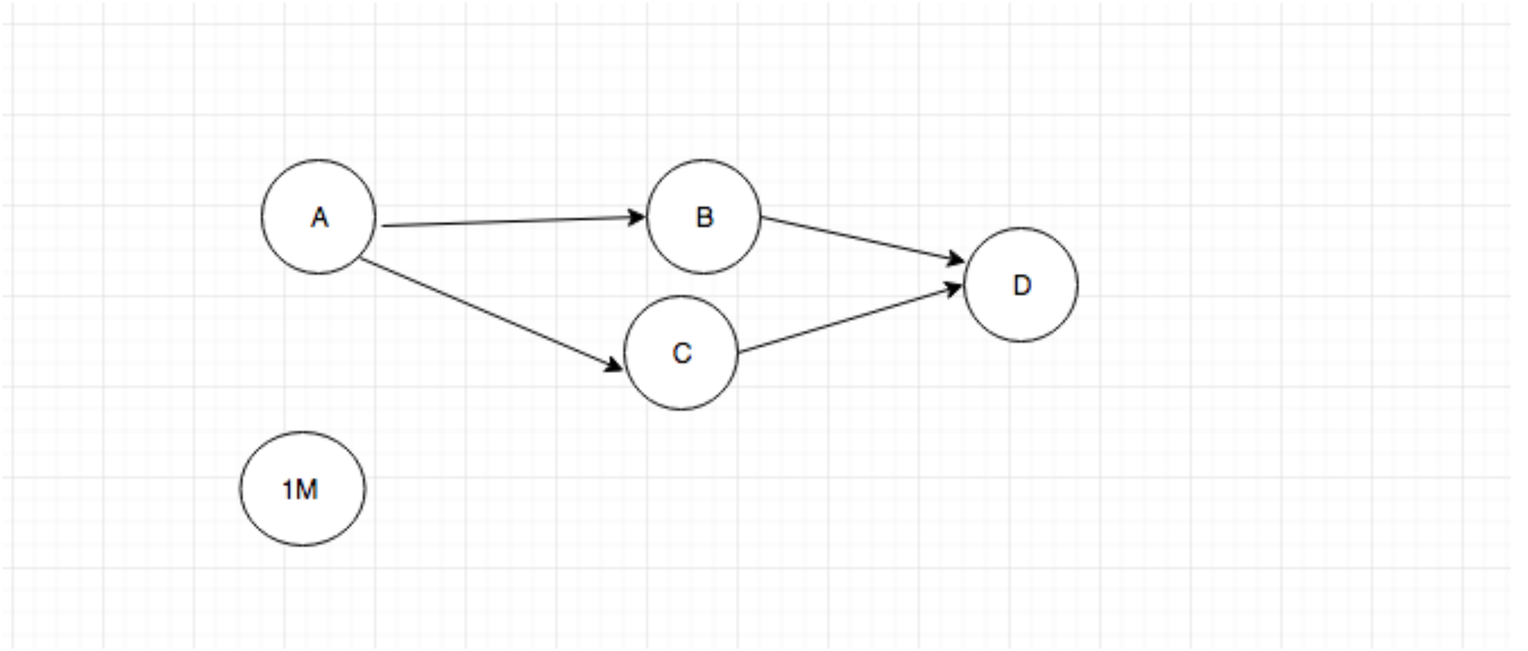

00:27
0.14
100
MainActivity has leaked
com.example.admin.
memoryleaksinandroid
static com.example.admin.
memoryleaksinandroid.MainActivity.
activities
references android.util.SparseArray.
mValues
references array java.lang.Object[].[0]
leaks com.example.admin.
memoryleaksinandroid.MainActivity
instance
DELETE

Heap Dump

- **A heap dump is a snapshot of memory at a given point in time.**
- **It contains information on the Java objects and classes in memory at the time the snapshot was taken.**
- **We can do a heap dump via Heap Monitor, LeakCanary**
- **It will generate a hprof file**
- **Usually the hprof file should be converted before to be opened using MAT**

Shallow Heap vs Retained Heap

- **Shallow heap is the memory consumed by one object**
- **Retained set of X is the set of objects which would be removed by GC when X is garbage collected**
- **Retained heap of X is the sum of shallow sizes of all objects in the retained set of X**

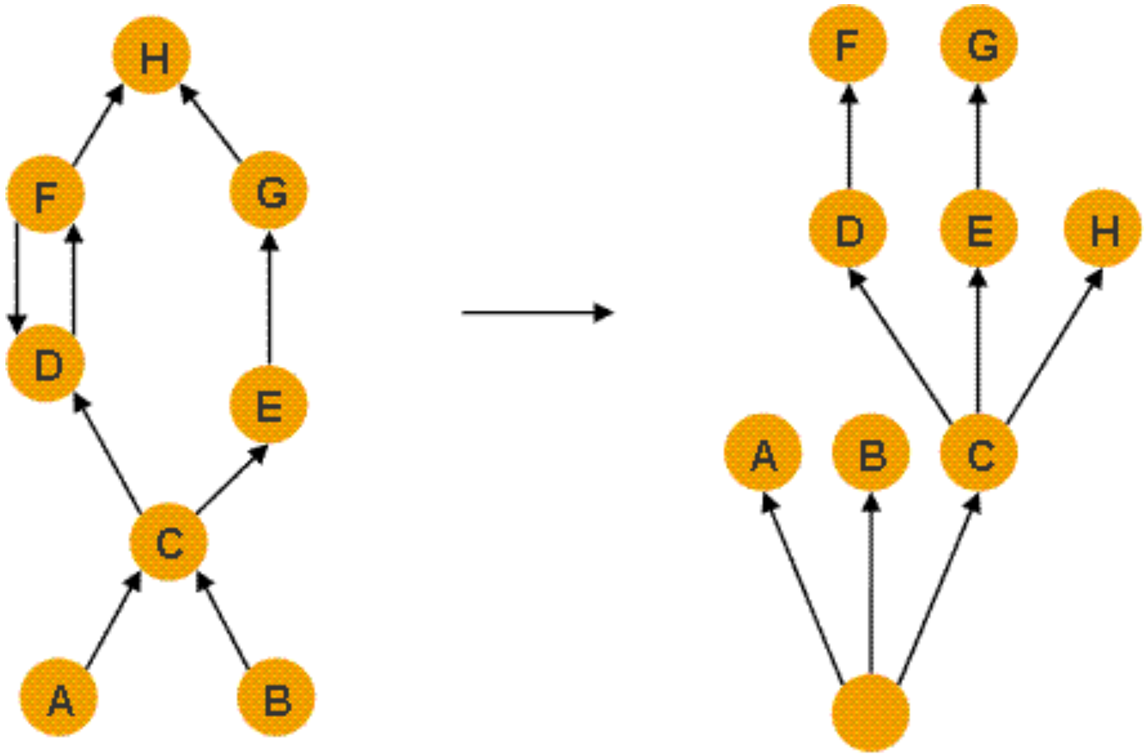

A
B
C
D
1M

How to calculate the size ?

Dominator Tree

- **An object x dominates an object y if every path in the object graph from the start (or the root) node to y must go through x.**

- **The immediate dominator x of some object y is the dominator closest to the object y.**


H
F
G
D
E
C
A
B
F
G
D
E
H
A
B
C

Memory Leaks Vs OOM

- **OOM is thrown when there is no enough space to create any objects and the heap could not expanded any more.**
- **OOM is a common indication of Memory Leaks**
- **Not all the OOMs are caused by Memory Leaks**
- **Memory Leaks could cause OOMs, but this is not inevitable.**

Questions and Answers

View droidyue.com to get more