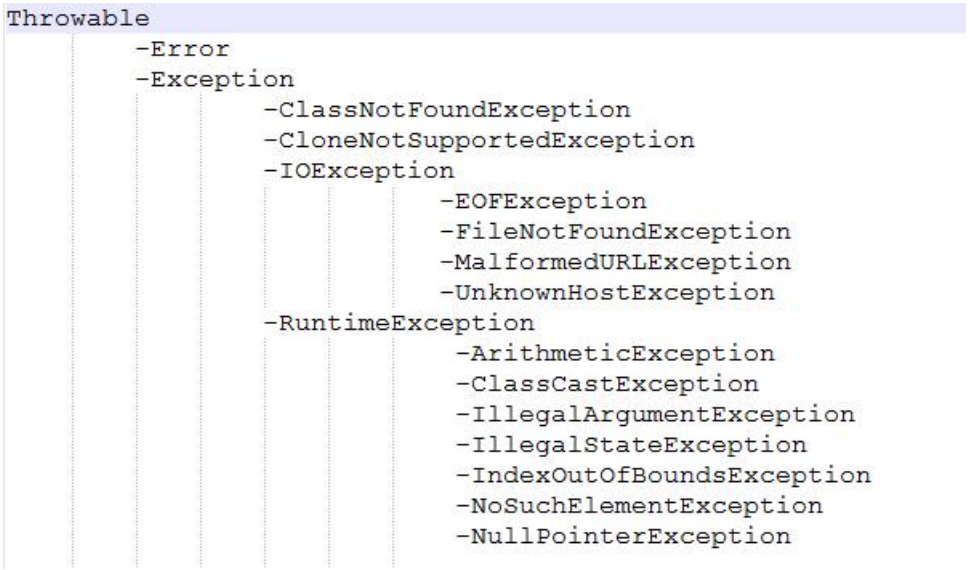
Java中Error和Exception

Java 异常类继承关系图：



（一）Throwable

Throwable 类是 Java 语言中所有错误或异常的超类。只有当对象是此类或其子类之一的实例时，才能通过 Java 虚拟机或者 Java throw 语句抛出，才可以是 catch 子句中的参数类型。 Throwable 类及其子类有两个构造方法，一个不带参数，另一个带有 String 参数，此参数可用于生成详细消息。 Throwable 包含了其线程创建时线程执行堆栈的快照。它还包含了给出有关错误更多信息的消息字符串。 Java将可抛出(Throwable)的结构分为三种类型： 1. 错误(Error) 2. 运行时异常(RuntimeException) 3. 被检查的异常(Checked Exception)

1.**Error** Error 是 Throwable 的子类，用于指示合理的应用程序不应该试图捕获的严重问题。大多数这样的错误都是异常条件。和RuntimeException一样， 编译器也不会检查Error。 当资源不足、约束失败、或是其它程序无法继续运行的条件发生时，就产生错误，程序本身无法修复这些错误的。 2.**Exception** Exception 类及其子类是 Throwable 的一种形式，它指出了合理的应用程序想要捕获的条件。 对于可以恢复的条件使用**被检查异常**（Exception的子类中除了RuntimeException之外的其它子类），对于程序错误使用运行时异常。 2.1 ClassNotFoundException 当应用程序试图使用以下方法通过字符串名加载类时： Class 类中的 forName 方法。 ClassLoader 类中的 findSystemClass 方法。 ClassLoader 类中的 loadClass 方法。 但是没有找到具有指定名称的类的定义，抛出该异常。

2.2 CloneNotSupportedException

当调用 Object 类中的 clone 方法复制对象，但该对象的类无法实现 Cloneable 接口时，抛出该异常。重写 clone 方法的应用程序也可能抛出此异常，指示不能或不应复制一个对象。

2.3 IOException 当发生某种 I/O 异常时，抛出此异常。此类是失败或中断的 I/O 操作生成的异常的通用类。

- EOFException

  当输入过程中意外到达文件或流的末尾时，抛出此异常。 此异常主要被**数据输入流**用来表明到达流的末尾。 注意：其他许多输入操作返回一个特殊值表示到达流的末尾，而不是抛出异常。 - FileNotFoundException

  当试图打开指定路径名表示的文件失败时，抛出此异常。在不存在具有指定路径名的文件时，此异常将由 FileInputStream、FileOutputStream 和 RandomAccessFile 构造方法抛出。如果该文件存在，但是由于某些原因不可访问，比如试图打开一个只读文件进行写入，则此时这些构造方法仍然会抛出该异常。

- MalformedURLException

  抛出这一异常指示出现了错误的 URL。或者在规范字符串中找不到任何合法协议，或者无法解析字符串。 — UnknownHostException 指示主机 IP 地址无法确定而抛出的异常。

2.4 RuntimeException 是那些可能在 Java 虚拟机正常运行期间抛出的异常的超类。可能在执行方法期间抛出但未被捕获的 RuntimeException 的任何子类都无需在 throws 子句中进行声明。 Java编译器不会检查它。当程序中可能出现这类异常时，还是会编译通过。 虽然Java编译器不会检查运行时异常，但是我们也可以通过throws进行声明抛出，也可以通过try-catch对它进行捕

获处理。

- ArithmeticException

  当出现异常的运算条件时，抛出此异常。例如，一个整数“除以零”时，抛出此类的一个实例。

- ClassCastException

  当试图将对象强制转换为不是实例的子类时，抛出该异常。 例如：Object x = new Integer(0);

- LllegalArgumentException

  抛出的异常表明向方法传递了一个不合法或不正确的参数。

- IllegalStateException

  在非法或不适当的时间调用方法时产生的信号。换句话说，即 Java 环境或 Java 应用程序没有处于请求操作所要求的适当状态下。

- IndexOutOfBoundsException

  指示某排序索引（例如对数组、字符串或向量的排序）超出范围时抛出。 应用程序可以为这个类创建子类，以指示类似的异常。

- NoSuchElementException

  由 Enumeration 的 nextElement 方法抛出，表明枚举中没有更多的元素。

- NullPointerException

  当应用程序试图在需要对象的地方使用 null 时，抛出该异常。这种情况包括： 调用 null 对象的实例方法。 访问或修改 null 对象的字段。 将 null 作为一个数组，获得其长度。 将 null 作为一个数组，访问或修改其时间片。 将 null 作为 Throwable 值抛出。 应用程序应该抛出该类的实例，指示其他对 null 对象的非法使用。

（二） SOF （堆栈溢出 StackOverflow）

> StackOverflowError 的定义： 当应用程序递归太深而发生堆栈溢出时，抛出该错误。
>
> 因为栈一般默认为1-2m，一旦出现死循环或者是大量的递归调用，在不断的压栈过程中，造成栈容量超过1m而导致溢出。

栈溢出的原因：

1. 递归调用
2. 大量循环或死循环
3. 全局变量是否过多
4. 数组、List、map数据过大

（三）Android的ＯＯＭ（Out Of Memory）

当内存占有量超过了虚拟机的分配的最大值时就会产生内存溢出（VM里面分配不出更多的page）。　　一般出现情况：加载的图片太多或图片过大时、分配特大的数组、内存相应资源过多没有来不及释放。

解决方法：

①在内存引用上做处理

软引用是主要用于内存敏感的高速缓存。在jvm报告内存不足之前会清除所有的软引用，这样以来gc就有可能收集软可及的对象，可能解决内存吃紧问题，避免内存溢出。什么时候会被收集取决于gc的算法和gc运行时可用内存的大小。

②对图片做边界压缩，配合软引用使用　③显示的调用GC来回收内存

```
if(bitmapObject.isRecycled()==false) //如果没有回收
         bitmapObject.recycle();
```

④优化Dalvik虚拟机的堆内存分配　　1.增强程序堆内存的处理效率

```
//在程序onCreate时就可以调用 即可
private final static floatTARGET_HEAP_UTILIZATION = 0.75f;

VMRuntime.getRuntime().setTargetHeapUtilization(TARGET_HEAP_UTILIZATION);
```

2.设置缓存大小

```
private final static int CWJ_HEAP_SIZE = 6* 1024* 1024 ;
 //设置最小heap内存为6MB大小
VMRuntime.getRuntime().setMinimumHeapSize(CWJ_HEAP_SIZE);
```

⑤ 用LruCache 和 AsyncTask<>解决

从cache中去取Bitmap，如果取到Bitmap，就直接把这个Bitmap设置到ImageView上面。　　如果缓存中不存在，那么启动一个task去加载（可能从文件来，也可能从网络）。