

参考博客

<https://www.jianshu.com/p/ee465def07ed>

内存溢出参考博客

https://www.oschina.net/question/1015065_2150170?sort=time

泄漏代码

<https://www.jianshu.com/p/ee465def07ed>

内存抖动享学代码<https://www.jianshu.com/p/2db63db97023>

垃圾回收机制

<https://www.cnblogs.com/huangwentian/p/10375966.html>

Android堆内存

<https://www.cnblogs.com/killmyday/archive/2013/06/12/3132518.html>

java线程调度

https://blog.csdn.net/qq_41701956/article/details/81664921

JVM内存管理

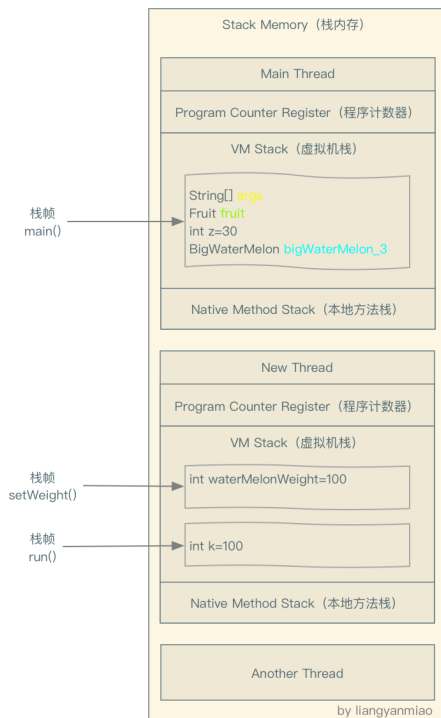
<https://blog.csdn.net/forezp/article/details/84503495>

深入理解垃圾回收机制

https://blog.csdn.net/yubujian_/article/details/80804708

内存分配

https://blog.csdn.net/jie_sil/article/details/90745324



堆内存溢出示例代码

```

1 public class Heap
2 {
3     public static void main(String[] args)
4     {
5         ArrayList list=new ArrayList();
6         while(true)
7         {
8             list.add(new Heap());
9         }
10    }
11 }

```

项目常见情况

1 生产者与消费者 速率不一致。没有控制队列中最大size大小，导致堆内存溢出

2 JSONObject.toJSON(object) 来处理Javabean的，这样处理简单的对象是没有问题的，但是对象如果复杂的话就会发生一些问题。

object对象过于复杂和大量时，用toJSON解析就会出现CPU、内存一直飙升，JVM一直执行GC操作，但是无法回收内存，最后会报

java.lang.OutOfMemoryError: GC overhead limit exceeded 错误。

内存抖动

优化内存抖动，核心就是防止频繁创建对象。常见的反面教材就是：循环中创建对象，大量调用的api中创建对象。而优化的主要手段，就是对象复用，常见的手段是：对象池，像是Handler的Message 单链表池，Glide的bitmap池等。

```
1  class Fruit {
2
3  static int x = 10;
4  static BigWaterMelon bigWaterMelon_1 = new BigWaterMelon(x);
5
6  int y = 20;
7  BigWaterMelon bigWaterMelon_2 = new BigWaterMelon(y);
8
9  public static void main(String[] args) {
10
11  final Fruit fruit = new Fruit();
12  int z = 30;
13  BigWaterMelon bigWaterMelon_3 = new BigWaterMelon(z);
14
15  new Thread() {
16  @Override
17  public void run() {
18  int k = 100;
19  setWeight(k);
20  }
21
22  void setWeight(int waterMelonWeight) {
23  fruit.bigWaterMelon_2.weight = waterMelonWeight;
24  }
25  }.start();
26  }
27  }
28
29  class BigWaterMelon {
30
31  public BigWaterMelon(int weight) {
32  this.weight = weight;
```

```
33  }
34
35  public int weight;
36  }
37
```

回收验证

```
1  package company;
2
3  public class Test1 {
4      public static Test1 test;
5      public void isAlive() {
6          System.out.println("I am alive :");
7      }
8      @Override
9      protected void finalize() throws Throwable {
10         super.finalize();
11         System.out.println("finalize method executed!");
12         test = this;
13     }
14     public static void main(String[] args) throws Exception {
15         test = new Test1();
16         test = null;
17         System.gc();
18         Thread.sleep(500);
19         if (test != null) {
20             test.isAlive();
21         } else {
22             System.out.println("no,I am dead :");
23         }
24         // 下面代码与上面完全一致，但是此次自救失败
25         test = null;
26         System.gc();
27         Thread.sleep(500);
28         if (test != null) {
29             test.isAlive();
30         } else {
31             System.out.println("no,I am dead :");
32         }
33     }
```

