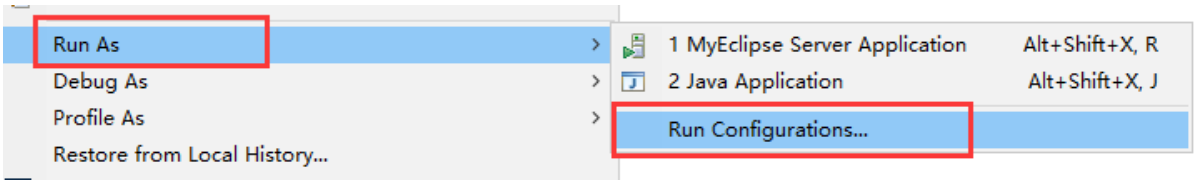
测试：

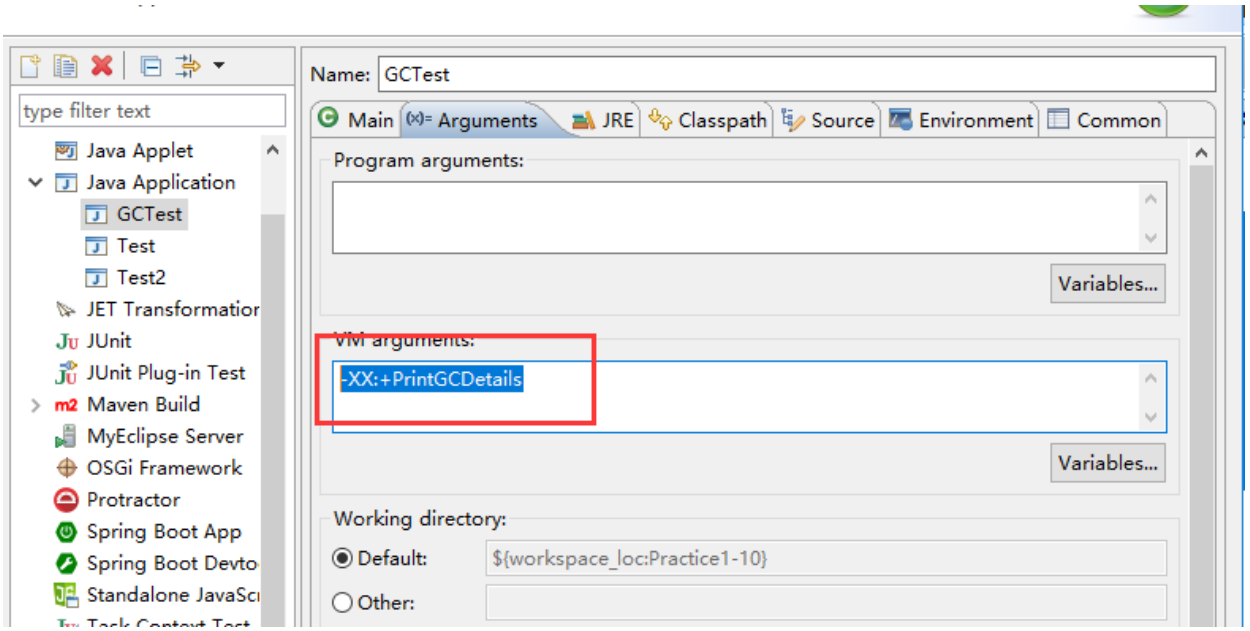
```
public class GCTest {

	public static void main(String[] args) {
		byte[] allocation1, allocation2;
		allocation1 = new byte[30900*1024];
		//allocation2 = new byte[900*1024];
	}
}
```

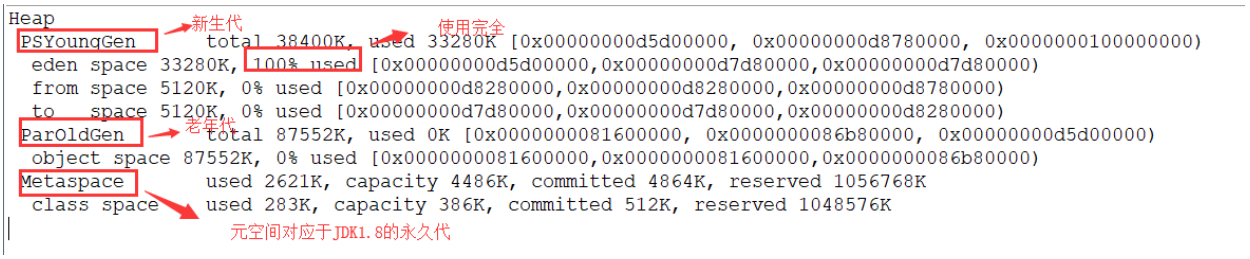
通过以下方式运行：



添加的参数：[redacted]



运行结果（红色字体描述有误，应该是对应于 JDK1.7 的永久代）：



从上图我们可以看出 eden 区内存几乎已经被分配完全（即使程序什么也不做，新生代也会使用 2000 多 k 内存）。假如我们再为 allocation2 分配内存会出现什么情况呢？

```
allocation2 = new byte[900*1024];
```

```
[GC (Allocation Failure) [PSYoungGen: 32897K->768K(38400K)] 32897K->31676K(125952K), 0.0229658 secs] [Times: us
Heap
 PSYoungGen      total 38400K, used 2001K [0x00000000d5d00000, 0x00000000da800000, 0x0000000100000000)
  eden space 33280K, 3% used [0x00000000d5d00000,0x00000000d5e344b8,0x00000000d7d80000)
  from space 5120K, 15% used [0x00000000d7d80000,0x00000000d7e40030,0x00000000d8280000)
  to   space 5120K, 0% used [0x00000000da300000,0x00000000da300000,0x00000000da800000)
 ParOldGen       total 87552K, used 30908K [0x0000000081600000, 0x0000000086b80000, 0x00000000d5d00000)
  object space 87552K, 35% used [0x0000000081600000,0x000000008342f010,0x0000000086b80000)
 Metaspace       used 2621K, capacity 4486K, committed 4864K, reserved 1056768K
  class space    used 283K, capacity 386K, committed 512K, reserved 1048576K
```

简单解释一下为什么会出现这种情况： 因为给 allocation2 分配内存的时候 eden 区内存几乎已经被分配完了，我们刚刚讲了当 Eden 区没有足够空间进行分配时，虚拟机将发起一次 Minor GC.GC 期间虚拟机又发现 allocation1 无法存入 Survivor 空间，所以只好通过 分配担保机制 把新生代的对象提前转移到老年代中去，老年代上的空间足够存放 allocation1，所以不会出现 Full GC。执行 Minor GC 后，后面分配的对象如果能够存在 eden 区的话，还是会在 eden 区分配内存。可以执行如下代码验证：

```
public class GCTest {

        public static void main(String[] args) {
                byte[] allocation1, allocation2,allocation3,allocation4,allocation5;
                allocation1 = new byte[32000*1024];
                allocation2 = new byte[1000*1024];
                allocation3 = new byte[1000*1024];
                allocation4 = new byte[1000*1024];
                allocation5 = new byte[1000*1024];
        }
}
```