http://lovestblog.cn/blog/2014/06/28/hsdb-string/

通过 HSDB 来了解 String 值的真身在哪里

最近通过@RednaxelaFX的一篇文章得知了 HSDB，并好好研究了一下用法，对学习 jvm 的人来说绝对是一个利器，可以摆脱 GDB，直接图形化看内存结构布局，具体的用法我就不多说了，这篇文章介绍得很详细了，这次写文章主要是想通过这一利器来分析下 String 的值在 java 里的内存情况，不同场景下的 String 的值到底是在内存里的哪块区域，这里强调的是值，并不是对象，因为对象我们都知道是存在 heap 里的，我们看 java.lang.String 的源码会看到有一个 value 数组，这里才是真正的值，本文顺带也是 hsdb 用法的一个介绍，如此利器希望给大家带来不一样的乐趣。

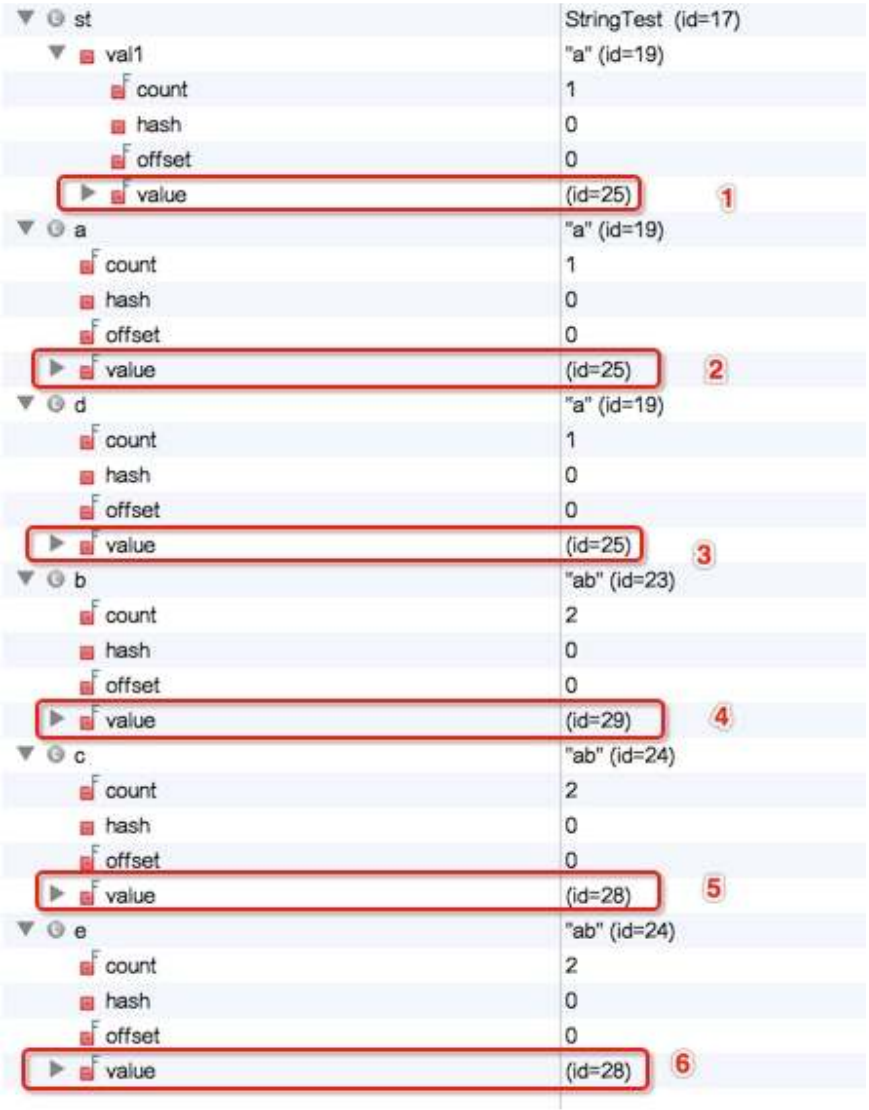
还是先看 demo

```
public class StringTest {
    private String val1="a";
    private static String val2=StringTest.class.getName()+"b";

    public static void main(String args[]){
        StringTest st=new StringTest();
        String a="a";
        String d="a";
        String b=a+"b";
        String c="a"+"b";
        String e="ab";
        System.out.println(a+b+c+d+e);
    }
}
```

本文想从上面的例子得出哪些结论呢？

1. 实例变量 val1 和局部变量 a,d 是否指向同一个内存地址
2. 局部变量 b,c,e 是否指向同一个内存地址
3. 局部变量 b 的值是在哪里分配的,stack?heap?perm?
4. 字符常量”a”,”ab”分配在哪里？
5. 静态变量 val2 的值又是分配在哪里？

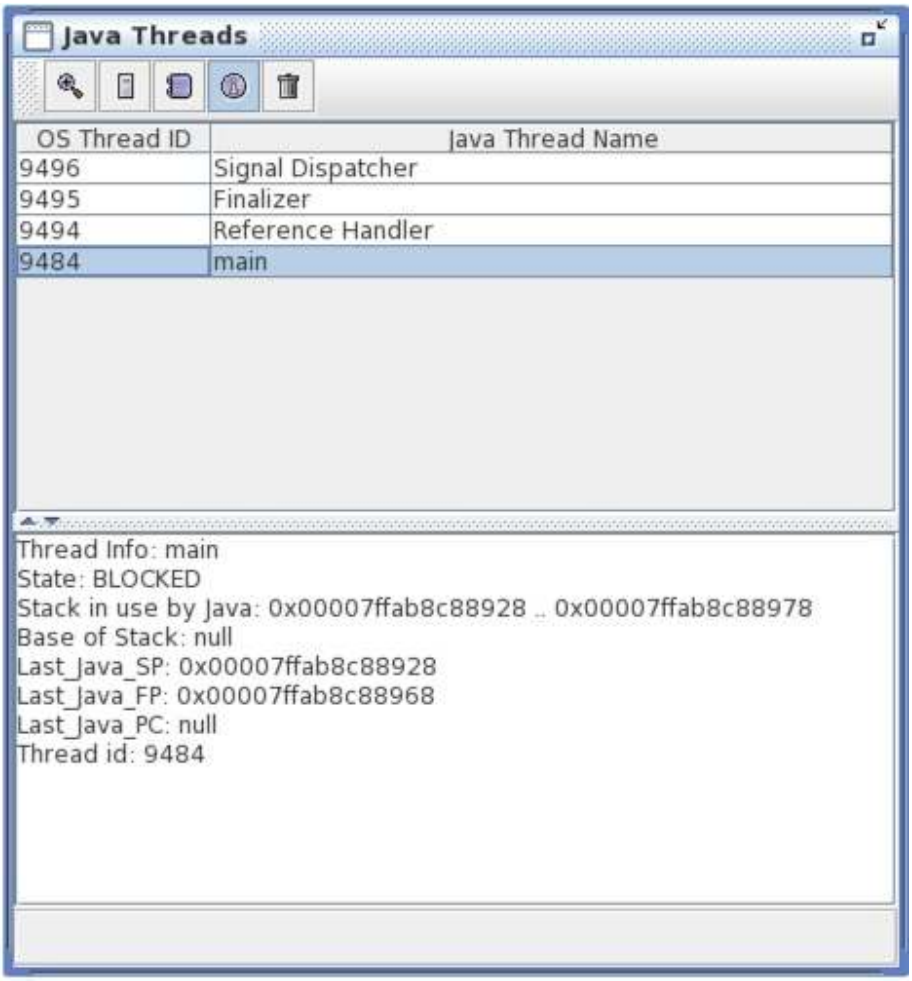
先看看我们通过 eclipse 调试能确定的结果，断点打在最后一行



得到初步结论：
1. 实例变量 val1 和局部变量 a,d 里的 value 值都是指向同一个 id 为 25 的值
2. 局部变量 c 和 e 指向了同一个 id 为 28 的值
3. 局部变量 b 和 c，e 不是指向同一个地方，有一个面值相同的值在另外一个内存区域

接下来我们通过 hsdb 来验证下上面的结论，以及解答剩下的疑惑
操作步骤如下：
1. 设置断点在`System.out.println(a+b+c+d+e);`这一行，vm 参数设置`-XX:+UseSerialGC -Xmx10m`
2. 通过 jps 命令获取对应的 pid
3. 然后通过如下命令打开 hsdb：`java -cp $JAVA_HOME/lib/sa-jdi.jar sun.jvm.hotspot.HSDB,(如果是 windows 请替换 $JAVA_HOME 为%JAVA_HOME%)`
4. 点击 File->Attach to...输入 pid

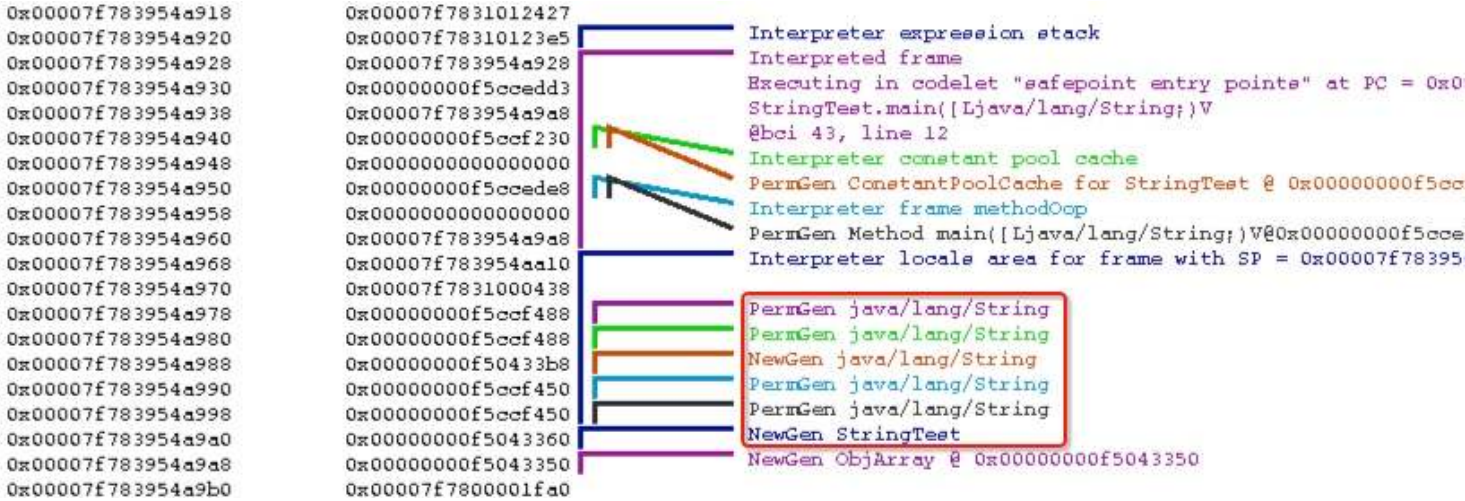
此时你会看到如下界面：



这是一个线程列表，我们选择 main 这个线程，也就是我们的主线程

5. 点击面板里第二个图标

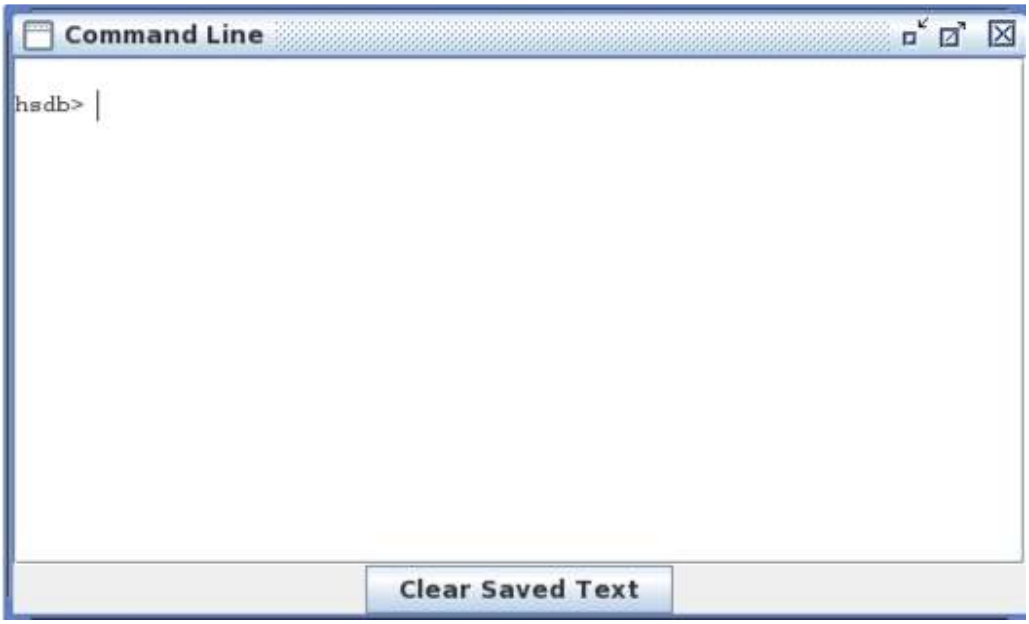
得到如下图



这个图其实是我们 main 方法的栈帧，因为我们目前只有一层调用，还在 main 方法里，看到我们圈起来的那部分内容，看到好多 String 对象既有在 PermGen 里的，也有在 NewGen 里的，那么每一个具体是什么值呢，是对应我们代码里的那些局部变量吗，如果是的话，哪个对应哪个呢

6. 点击大窗口里的 windows->Console

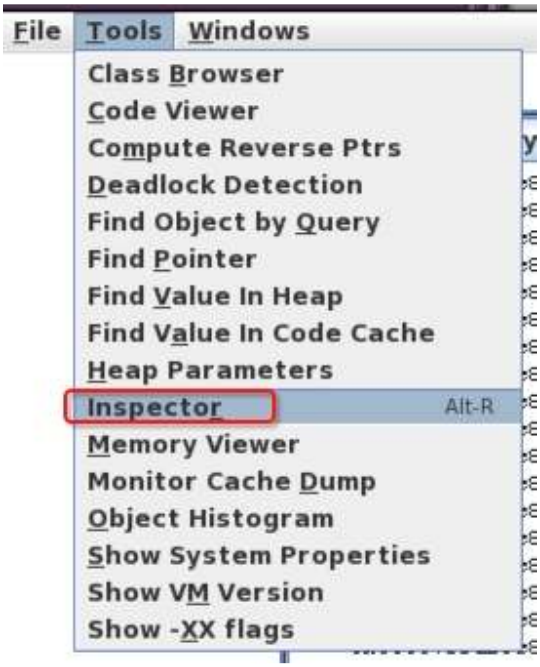
得到命令行控制台窗口，在窗口内敲回车，会看到如下界面



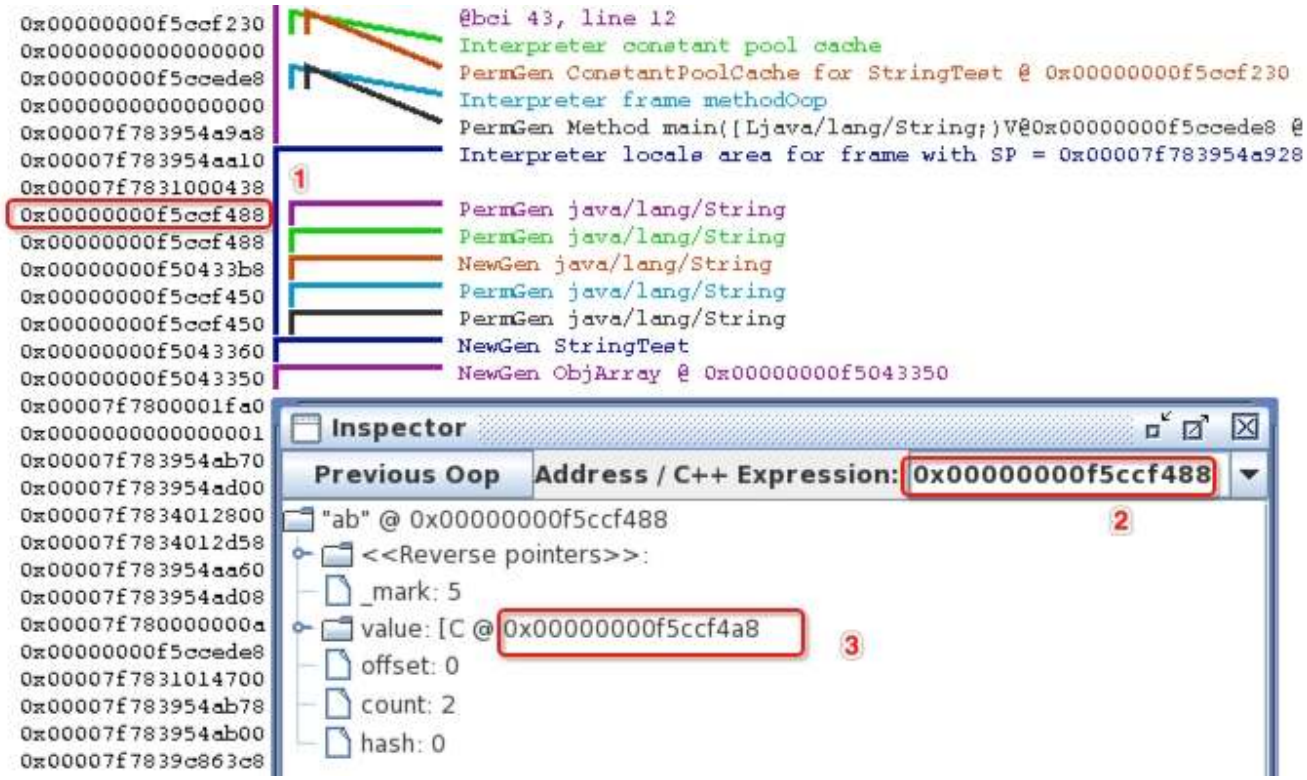
7. 在命令行窗口里输入 universe，先得到每个分区的内存范围，由于格式的问题，我就直接 copy 出来了

```
Gen 0:   eden [0x00000000f5000000,0x00000000f5050f78,0x00000000f52b0000) space capacity = 2818048, 11.7684
  from [0x00000000f52b0000,0x00000000f52b0000,0x00000000f5300000) space capacity = 327680, 0.0 used
  to   [0x00000000f5300000,0x00000000f5300000,0x00000000f5350000) space capacity = 327680, 0.0 usedInvocat

Gen 1:   old  [0x00000000f5350000,0x00000000f5350000,0x00000000f5a00000) space capacity = 7012352, 0.0 use

  perm [0x00000000f5a00000,0x00000000f5ccf4c0,0x00000000f6ec0000) space
```

8. 从上面的 main 方法栈帧里我们分别取查看那些 String 对象在内存里的位置（第二列地址就是对象的地址，第一列是栈帧里每部分的内存地址），先按照如下菜单调出查看内存结构的窗口



弹出窗口之后类似下面的操作，在 2 处输入 1 的地址，1 处圈起来的是紫色标注的 String 对象的内存地址，细心的读者可能发现了，1 处的地址在上面的每个分区内存块的 PSPermGen 里，这说明这个值为 ab 的 String 对象是在 perm 区的，这个对象的 char 数组的地址，也就是下面的标注 3 处的

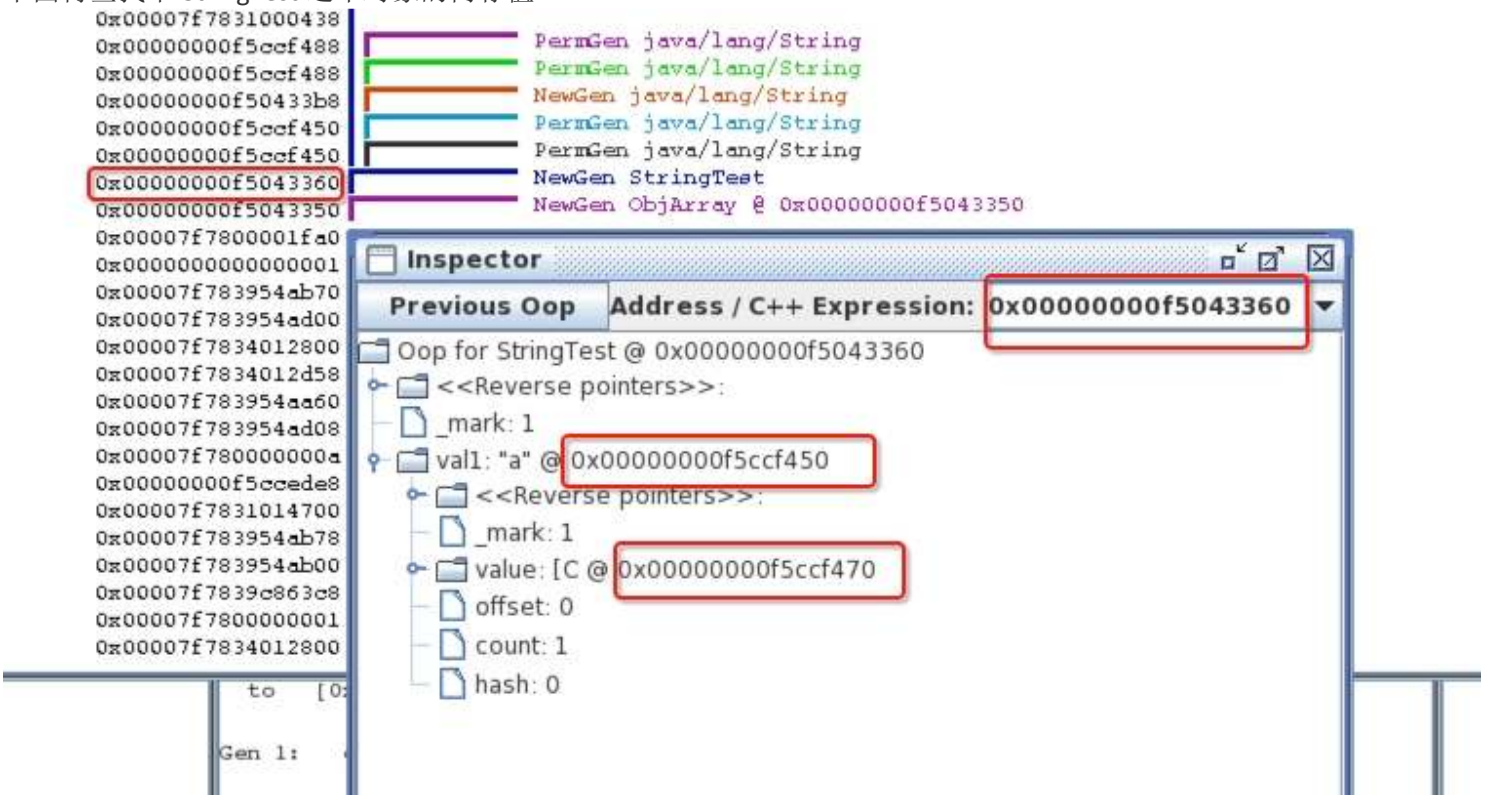


其实我们看到的那几个 String 对象的顺序是对应我们声明的局部变量的逆序，也就是 e，c，b，d，a，最后那个 StringTest 就是局部变量 st，后面的 ObjArray 其实是我们 main 方法传进的字符数组，这个其实我们通过 javap -verbose StringTest 可以查到

```
LocalVariableTable:
 Start  Length  Slot  Name   Signature
 0      52      0    args       [Ljava/lang/String;
 8      44      1    st       LStringTest;
 11      41      2    a       Ljava/lang/String;
 14      38      3    d       Ljava/lang/String;
 35      17      4    b       Ljava/lang/String;
 39      13      5    c       Ljava/lang/String;
 43      9      6    e       Ljava/lang/String;
```

哪个 solt 对应哪个局部变量都有写的，要想看到这个必须在编译的时候要加上-g 参数才行

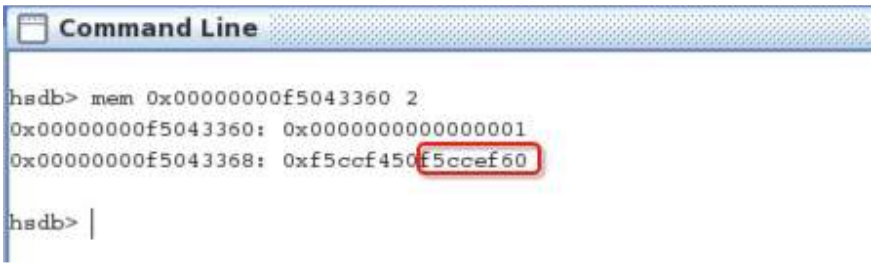
下面再查找下 StringTest 这个对象的内存值



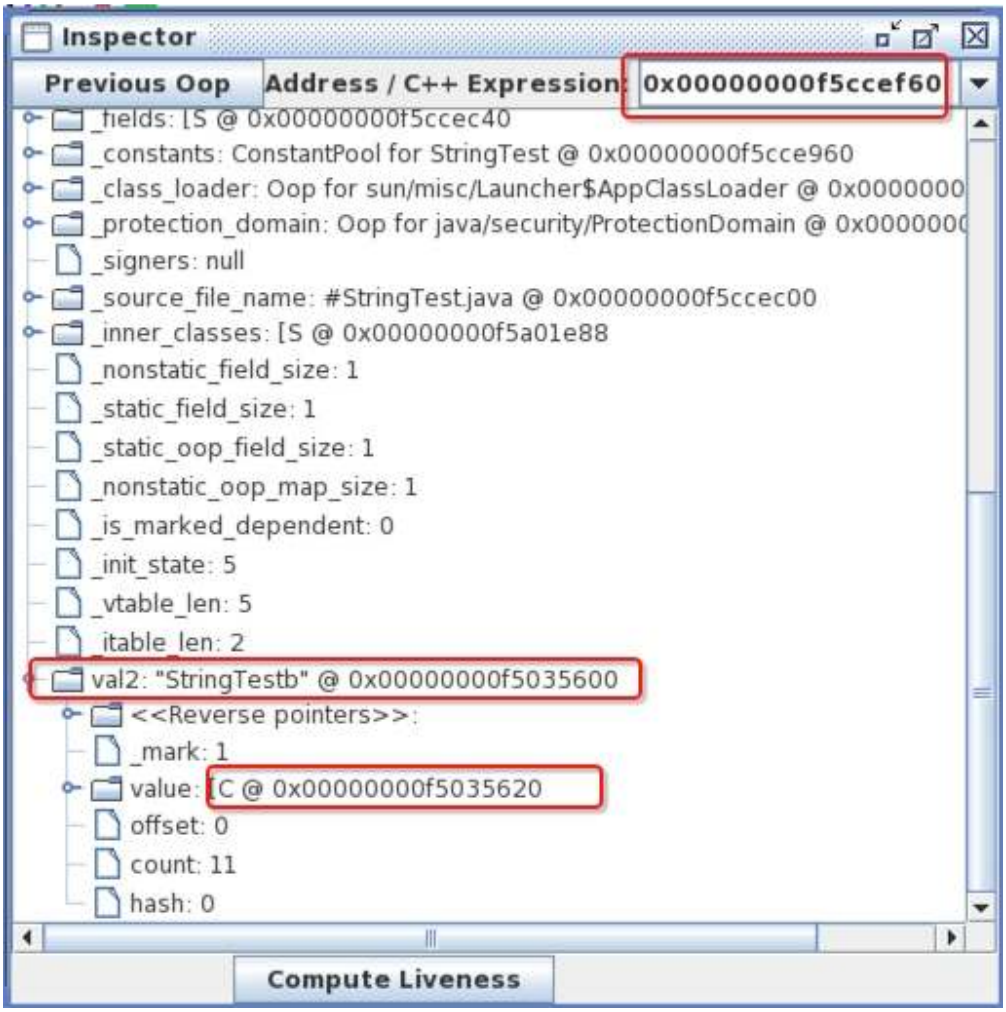
通过分别对比每个 String 对象和 StringTest 对象的内存地址和每个区的内存地址范围，我们能得出的结论是

* 局部变量 a,d,c,e 是在 perm 区的
* 局部变量 b 和 st 是在 eden 区的，但是 st 的 val1 的值又是在 perm 区的
* 同时能验证上面一开始得出的三个结论
* 我们也没看到有对象在栈上分配，只看到栈上持有对象的引用，因此当栈回收的时候只是将引用给回收了，具体的对象值还是在内存里

9. 接下来是要找到静态变量 val2，在命令行中输入`mem 0x00000000f5043360 2`，因为 val2 作为静态变量是和 class 关联的，因此要找到对象的 class，如果了解 java 对象的内存结构的话我们知道每个 oop 都有一个 head，这个 head 由两部分组成，一个是 mark,另一个是_klass，因此通过 mem 对 oop 的内存地址取连续的两个字宽，第二个字宽就是我们要的 klass



这里估计是一个 bug，不能全取，我们只能取后面的 8 位才行，也就是 0xf5ccef60，然后按照第 8 步的方式输入上面的内存地址，在最后我们看到 val2



对比内存分代我们得到这个地址是在 eden 区的，也就是在 heap 里分配的,另外如果你加一个赋值常量的静态变量，你会发现居然是在 perm 区的，这个就大家自己去验证吧

注：以上结论都是在 centos 系统 jdk6 上进行验证的，jdk7 可能有所不一样.