<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](http://rednaxelafx.iteye.com/blog/1847971)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

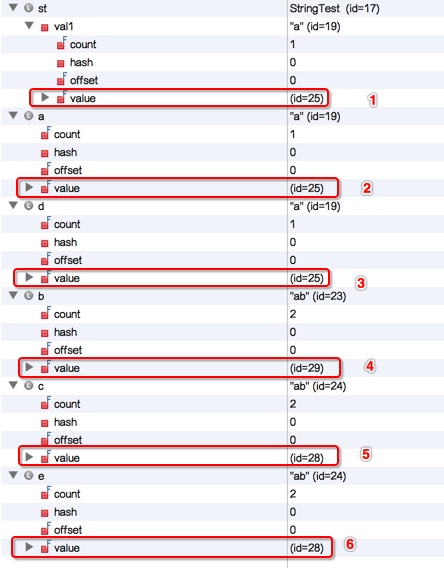
2. 局部变量b,c,e是否指向同一个内存地址

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：

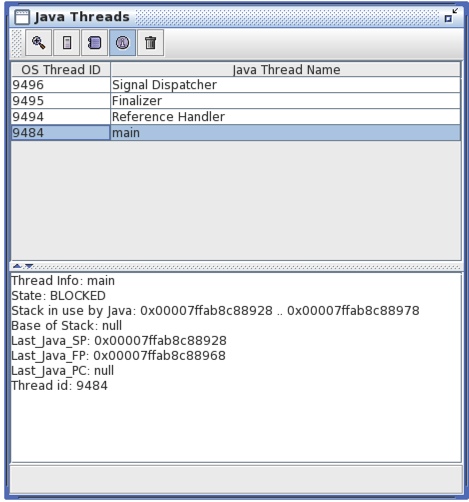
1. 设置断点在`System.out.println(a+b+c+d+e);`这一行，vm参数设置`-XX:+UseSerialGC -Xmx10m`

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：

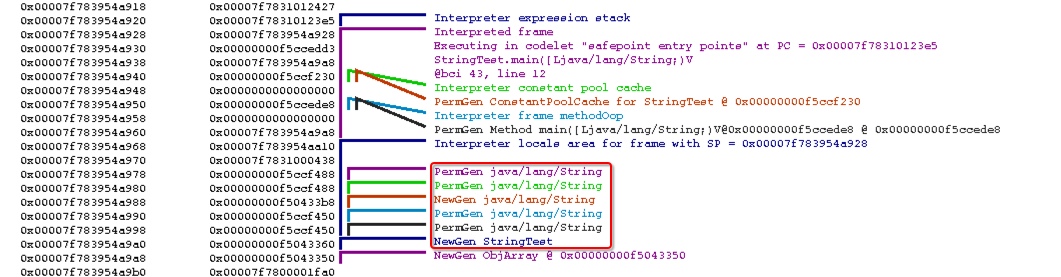


这是一个线程列表，我们选择main这个线程，也就是我们的主线程

5. 点击面板里第二个图标

http://lovestblog.cn/images/2014/06/hsdb_2.jpg

得到如下图

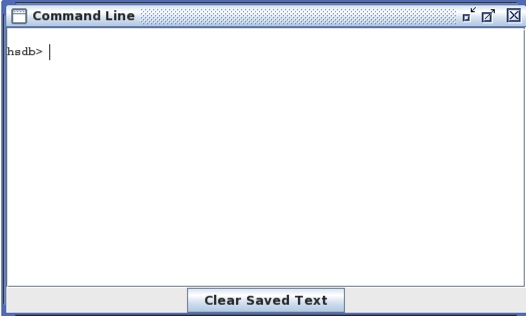


这个图其实是我们main方法的栈帧，因为我们目前只有一层调用，还在main方法里，看到我们圈起来的那部分内容，看到好多String对象既有在PermGen里的，也有在NewGen里的，那么每一个具体是什么值呢，是对应我们代码里的那些局部变量吗，如果是的话，哪个对应哪个呢

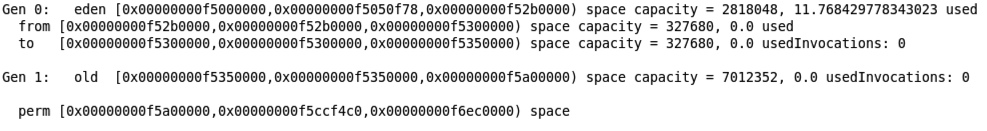
6. 点击大窗口里的windows->Console



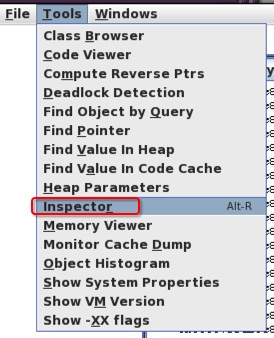
得到命令行控制台窗口，在窗口内敲回车，会看到如下界面



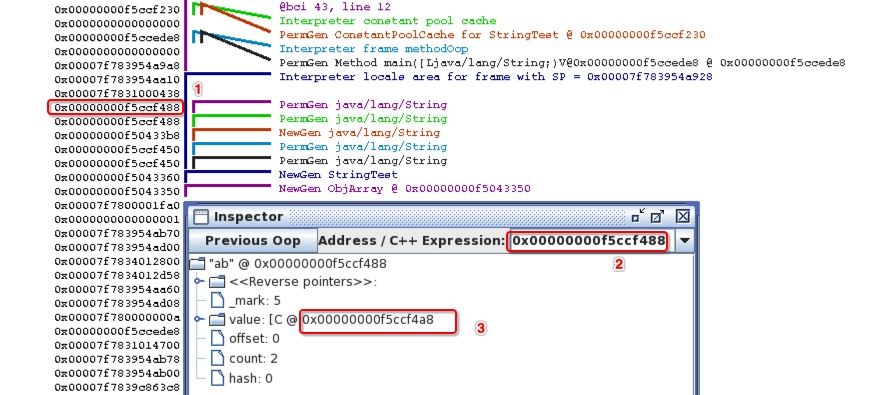
7. 在命令行窗口里输入universe，先得到每个分区的内存范围，由于格式的问题，我就直接copy出来了



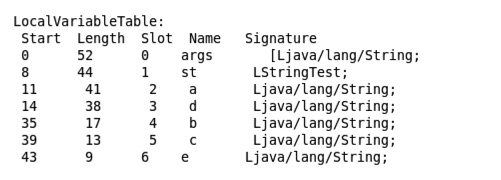
8. 从上面的main方法栈帧里我们分别取查看那些String对象在内存里的位置（第二列地址就是对象的地址，第一列是栈帧里每部分的内存地址），先按照如下菜单调出查看内存结构的窗口



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

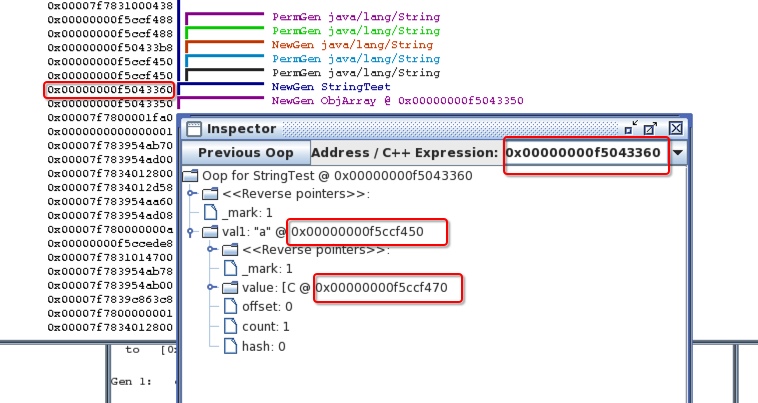


其实我们看到的那几个String对象的顺序是对应我们声明的局部变量的逆序，也就是e，c，b，d，a，最后那个StringTest就是局部变量st，后面的ObjArray其实是我们main方法传进的字符数组，这个其实我们通过javap -verbose StringTest可以查到



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

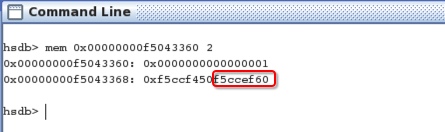
\* 局部变量a,d,c,e是在perm区的

\* 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

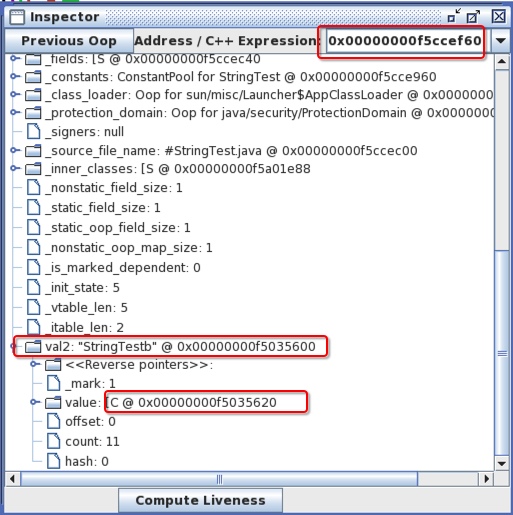
\* 同时能验证上面一开始得出的三个结论

\* 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.