# Object的内存布

## 使用JavaAgent测试Object的大小

### 对象大小（64位机）

#### 观察虚拟机配置

java -XX:+PrintCommandLineFlags -version

```
C:\Users\Chase>java -XX:+PrintCommandLineFlags -version
-XX:InitialHeapSize=16777216 -XX:MaxHeapSize=268435456 -XX:+PrintCommandLineFlags -XX:-UseLargePagesIndividualAllocation

java version "1.8.0_221"
Java(TM) SE Runtime Environment (build 1.8.0_221-b11)
Java HotSpot(TM) Client VM (build 25.221-b11, mixed mode)

C:\Users\Chase>
```

### 对象的内存布局分为两种　普通对象和数组对象

#### 普通对象

1. 对象头：markword 8

   hospot 中叫 markword　 占8个字节

2. ClassPointer指针：-XX:+UseCompressedClassPointers 为4字节 不开启为8字节

   t = new T() 出来对象后　有一个指针 t , 这个指针指向 T.class

3. 实例数据

   1. long  int  String ....
   2. 引用类型：-XX:+UseCompressedOops 为4字节 不开启为8字节
      Oops Ordinary Object Pointers

4. Padding对齐，8的倍数

   读取时按照块来读取的 ，目的是为了提升效率 ，对齐后对象大小是8的倍数

#### 数组对象

1. 对象头：markword 8
2. ClassPointer指针同上
3. 数组长度：4字节
4. 数组数据

5. 对齐 8的倍数

## 如何观察Object大小?

原理：java不像其他语言可以动态获取大小，需要自己实现一个Agent （Java的一个机制）

> Agent：JVM在加载到Class之前，中间可以有一个Agent代理截获Class中的内容，也可以任意修改，通过这个机制获取Object的大小，如下是Agent实现的过程

## 实验

1. 新建项目ObjectSize （1.8）
2. 创建文件ObjectSizeAgent

```
import java.lang.instrument.Instrumentation;

public class ObjectSizeAgent {
    private static Instrumentation inst;
    // premain   格式是固定的
    // 这个方法是虚拟机自己调用的  JVM会传入一个 Instrumentation  保存到Agent里面
    public static void premain(String agentArgs, Instrumentation _inst) {
        inst = _inst;
    }

    // 通过保存JVM的 Instrumentation    调用 里面的 getObjectSize 方法
    public static long sizeOf(Object o) {
        return inst.getObjectSize(o);
    }
}
```

3. src目录下创建META-INF/MANIFEST.MF

```
Manifest-Version: 1.0
Created-By: chase.com
Premain-Class: com.chase.jvm.agent.ObjectSizeAgent  // main 方法运行之前 的一个Class
```

> 注意Premain-Class这行必须是新的一行（回车 + 换行），确认idea不能有任何错误提示

4. 打包jar文件
5. 在需要使用该Agent Jar的项目中引入该Jar包
   project structure - project settings - library 添加该jar包
6. 运行时需要该Agent Jar的类，加入参数：  指定Agent

```
-javaagent:C:\work\ijprojects\ObjectSize\out\artifacts\ObjectSize_jar\ObjectSize.jar
```

7. 如何使用该类：

```
package com.chase.jvm.c3_jmm;

import com.chase.jvm.agent.ObjectSizeAgent;

public class T03_SizeOfAnObject {
    public static void main(String[] args) {
        // 16 字节  对象头8  + class指针8压缩后成 4
        // 结果：8 + 4  + Padding对齐  = 16
        System.out.println(ObjectSizeAgent.sizeOf(new Object()));
        // 24字节   对象头8 + class指针8---4  + 数组长度 4  数组内容 没内容= 0字节
        // 结果：8 + 4 + 4 + 0 + Padding(4)   = 20
        System.out.println(ObjectSizeAgent.sizeOf(new int[] {}));
        // 8 头 + classPointer 4 + int 4 + String引用 4（压缩后）+ byte 1 + byte 1
        // + Object 4(压缩后) + byte 1
        // 结果：8+4+4+4+4+1+1+4+1 + Padding（1）   = 32
        System.out.println(ObjectSizeAgent.sizeOf(new P()));
    }

    // 一个Object 占用多少个字节
    // --- 运行配置
    // -XX:+UseCompressedClassPointers  对class指针压缩  默认开启
    // -XX:+UseCompressedOops  对普通象指针压缩  默认开启
    //     Oops = ordinary object pointers  普通对象指针
    private static class P {
                        //8 _markword     对象头 8 个字节
                        //4 _oop指针     class 指针 8  压缩后4
        int id;         //4
        String name;    //4  引用类型，正常是8个字节  压缩后变成4
        int age;        //4

        byte b1;        //1
        byte b2;        //1

        Object o;       //4
        byte b3;        //1

    }
}
```

## Hotspot开启内存压缩的规则（64位机）

1. 4G以下，直接砍掉高32位
2. 4G - 32G，默认开启内存压缩 ClassPointers Oops
3. 32G，压缩无效，使用64位
   内存并不是越大越好（^-^）

## IdentityHashCode的问题

当一个对象计算过identityHashCode之后，不能进入偏向锁状态

https://cloud.tencent.com/developer/article/1480590https://cloud.tencent.com/developer/article/1484167

https://cloud.tencent.com/developer/article/1485795

https://cloud.tencent.com/developer/article/1482500

## 对象定位

t 怎么找到  T() 和 T.class

- https://blog.csdn.net/clover_lily/article/details/80095580

T t = new T()

1. 句柄池　—- 优点：效率高

   有一个橘句柄池的东西　存了两个指针一个指向对象，一个指向 Class

2. 直接指针　——优点：GC 回收快

   t 指向对象，对象指向 Tlass

不同虚拟机实现的机制不一样 HosPot　是第二种