对象的实例化内存布局与访问定位

对象的实例化

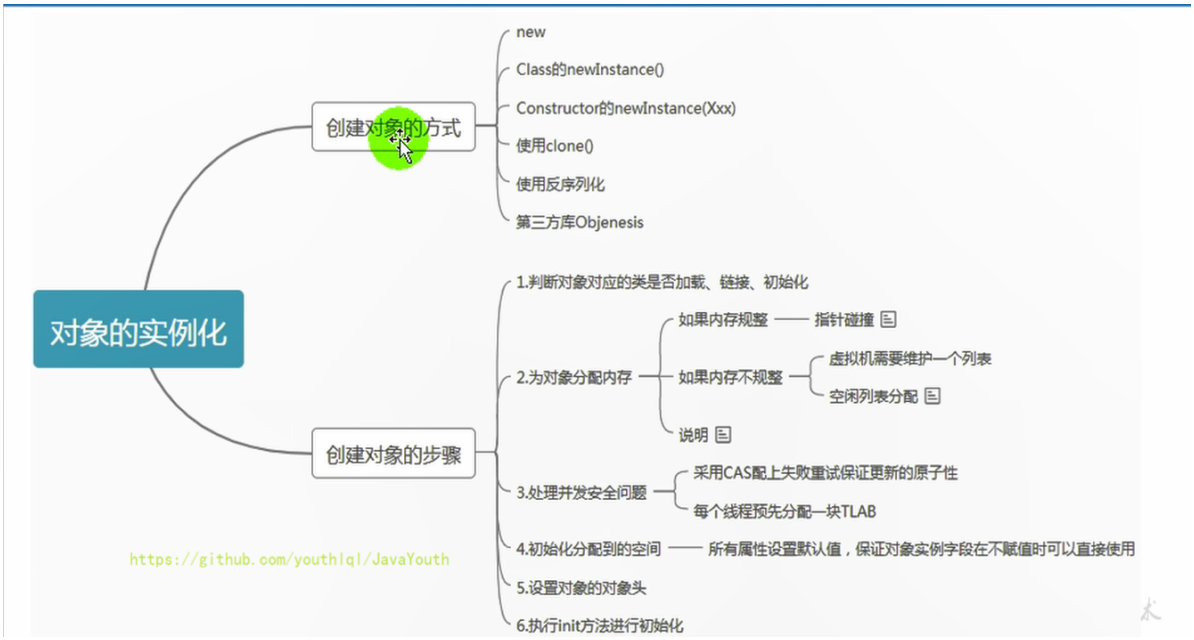
大厂面试题

美团：

1. 对象在`JVM`中是怎么存储的？
2. 对象头信息里面有哪些东西？

蚂蚁金服：

二面：`java`对象头里有什么



对象创建的方式

1. new：最常见的方式、单例类中调用getInstance的静态类方法，XXXFactory的静态方法
2. Class的newInstance方法：在JDK9里面被标记为过时的方法，因为只能调用空参构造器，并且权限必须为 public
3. Constructor的newInstance(Xxxx)：反射的方式，可以调用空参的，或者带参的构造器
4. 使用clone()：不调用任何的构造器，要求当前的类需要实现Cloneable接口中的clone方法
5. 使用序列化：从文件中，从网络中获取一个对象的二进制流，序列化一般用于Socket的网络传输
6. 第三方库 Objenesis

对象创建的步骤

> 从字节码看待对象的创建过程

```
public class ObjectTest {
    public static void main(String[] args) {
        Object obj = new Object();
    }
}
```

```
 public static void main(java.lang.String[]);
    descriptor: ([Ljava/lang/String;)V
    flags: ACC_PUBLIC, ACC_STATIC
    Code:
      stack=2, locals=2, args_size=1
         0: new           #2                  // class java/lang/Object
         3: dup
         4: invokespecial #1                  // Method java/lang/Object."
<init>":()V
         7: astore_1
         8: return
      LineNumberTable:
        line 9: 0
        line 10: 8
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0       9     0  args   [Ljava/lang/String;
            8       1     1   obj   Ljava/lang/Object;
}
```

1、判断对象对应的类是否加载、链接、初始化

1. 虚拟机遇到一条new指令，首先去检查这个指令的参数能否在Metaspace的常量池中定位到一个类的符号引用，并且检查这个符号引用代表的类是否已经被加载，解析和初始化。（即判断类元信息是否存在）。
2. 如果该类没有加载，那么在双亲委派模式下，使用当前类加载器以ClassLoader + 包名 + 类名为key进行查找对应的.class文件，如果没有找到文件，则抛出ClassNotFoundException异常，如果找到，则进行类加载，并生成对应的Class对象。

2、为对象分配内存

1. 首先计算对象占用空间的大小，接着在堆中划分一块内存给新对象。如果实例成员变量是引用变量，仅分配引用变量空间即可，即4个字节大小
2. 如果内存规整：采用指针碰撞分配内存
    - 如果内存是规整的，那么虚拟机将采用的是指针碰撞法（Bump The Point）来为对象分配内存。
    - 意思是所有用过的内存在一边，空闲的内存放另外一边，中间放着一个指针作为分界点的指示器，分配内存就仅仅是把指针往空闲内存那边挪动一段与对象大小相等的距离罢了。

    - 如果垃圾收集器选择的是Serial，ParNew这种基于压缩算法的，虚拟机采用这种分配方式。一般使用带Compact（整理）过程的收集器时，使用指针碰撞。
    - 标记压缩（整理）算法会整理内存碎片，堆内存一存对象，另一边为空闲区域
3. 如果内存不规整
    - 如果内存不是规整的，已使用的内存和未使用的内存相互交错，那么虚拟机将采用的是空闲列表来为对象分配内存。
    - 意思是虚拟机维护了一个列表，记录上哪些内存块是可用的，再分配的时候从列表中找到一块足够大的空间划分给对象实例，并更新列表上的内容。这种分配方式成为了“空闲列表（Free List）”
    - 选择哪种分配方式由Java堆是否规整所决定，而Java堆是否规整又由所采用的垃圾收集器是否带有压缩整理功能决定
    - 标记清除算法清理过后的堆内存，就会存在很多内存碎片。

3、处理并发问题

1. 采用CAS+失败重试保证更新的原子性
2. 每个线程预先分配TLAB - 通过设置 -XX:+UseTLAB参数来设置（区域加锁机制）
3. 在Eden区给每个线程分配一块区域

4、初始化分配到的空间

- 所有属性设置默认值，保证对象实例字段在不赋值可以直接使用
- 给对象属性赋值的顺序：

1. 属性的默认值初始化
2. 显示初始化/代码块初始化（并列关系，谁先谁后看代码编写的顺序）
3. 构造器初始化

5、设置对象的对象头

将对象的所属类（即类的元数据信息）、对象的HashCode和对象的GC信息、锁信息等数据存储在对象的对象头中。这个过程的具体设置方式取决于JVM实现。

6、执行init方法进行初始化

1. 在Java程序的视角看来，初始化才正式开始。初始化成员变量，执行实例化代码块，调用类的构造方法，并把堆内对象的首地址赋值给引用变量
2. 因此一般来说（由字节码中跟随invokespecial指令所决定），new指令之后会接着就是执行init方法，把对象按照程序员的意愿进行初始化，这样一个真正可用的对象才算完成创建出来。

> **从字节码角度看 init 方法**

```
/**
 * 测试对象实例化的过程
 *  ① 加载类元信息 - ② 为对象分配内存 - ③ 处理并发问题  - ④ 属性的默认初始化（零值初始化）
 *  - ⑤ 设置对象头的信息 - ⑥ 属性的显式初始化、代码块中初始化、构造器中初始化
 *
```

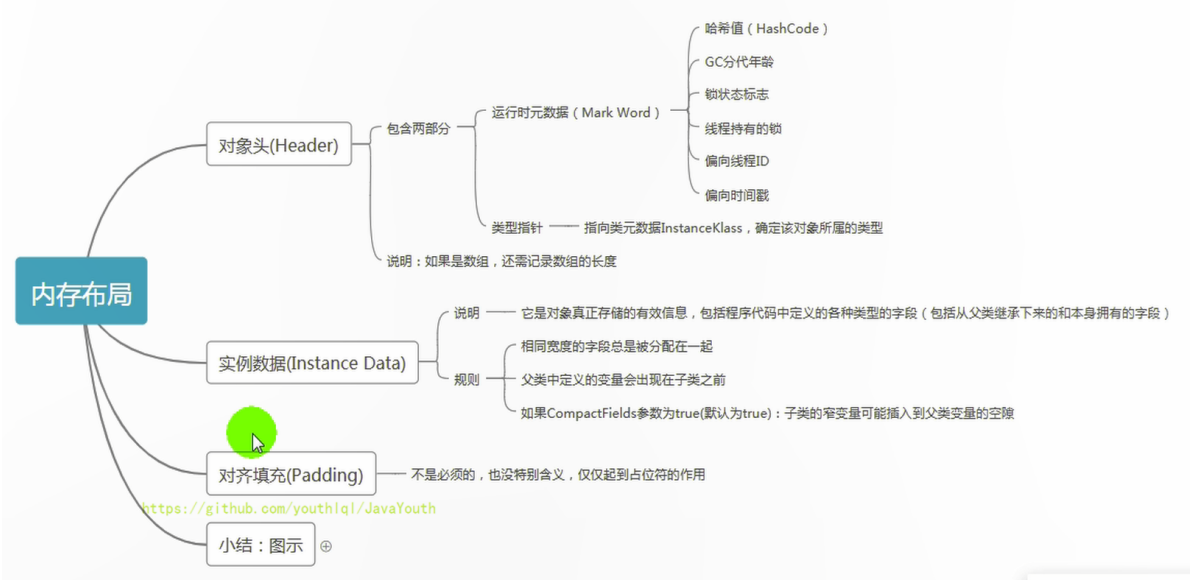
```
 *
 *  给对象的属性赋值的操作：
 *  ① 属性的默认初始化 - ② 显式初始化 / ③ 代码块中初始化 - ④ 构造器中初始化
 */

public class Customer{
    int id = 1001;
    String name;
    Account acct;

    {
        name = "匿名客户";
    }
    public Customer(){
        acct = new Account();
    }

}
class Account{

}
```

Customer类的字节码

```
 0 aload_0
 1 invokespecial #1 <java/lang/Object.<init>>
 4 aload_0
 5 sipush 1001
 8 putfield #2 <com/atguigu/java/Customer.id>
11 aload_0
12 ldc #3 <匿名客户>
14 putfield #4 <com/atguigu/java/Customer.name>
17 aload_0
18 new #5 <com/atguigu/java/Account>
21 dup
22 invokespecial #6 <com/atguigu/java/Account.<init>>
25 putfield #7 <com/atguigu/java/Customer.acct>
28 return
```

- init() 方法的字节码指令：
  - 属性的默认值初始化：`id = 1001;`
  - 显示初始化/代码块初始化：`name = "匿名客户";`
  - 构造器初始化：`acct = new Account();`

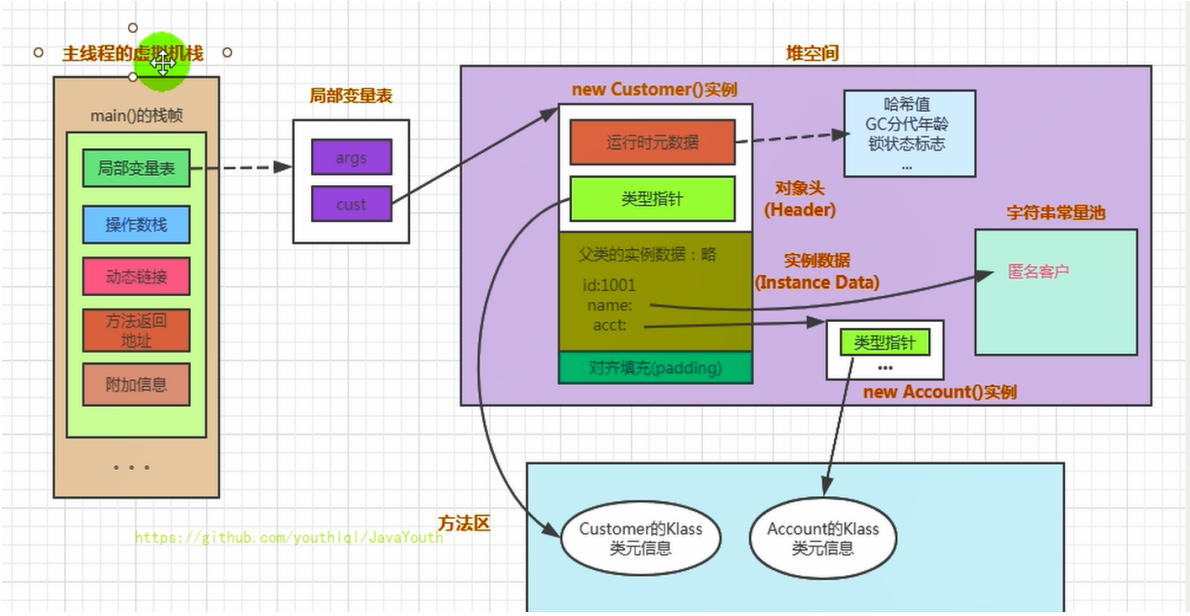
对象的内存布局



内存布局总结

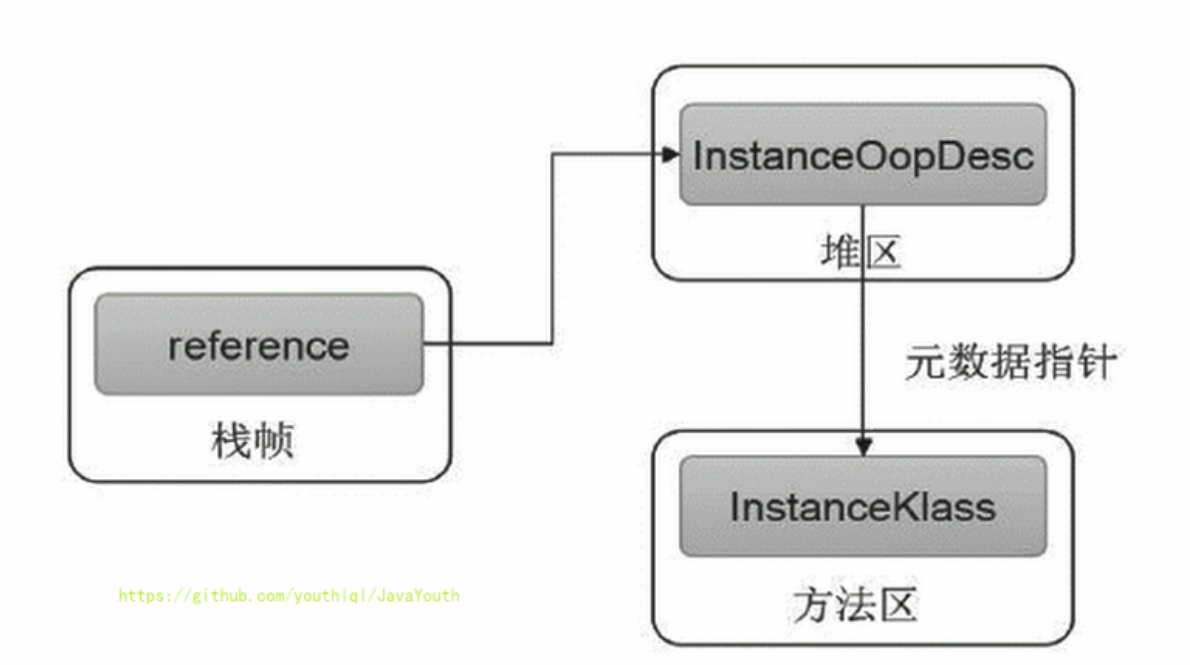
```
public class Customer{
    int id = 1001;
    String name;
    Account acct;

    {
        name = "匿名客户";
    }
    public Customer(){
        acct = new Account();
    }
    public static void main(String[] args) {
        Customer cust = new Customer();
    }
}
class Account{

}
```

图解内存布局



对象的访问定位

JVM是如何通过栈帧中的对象引用访问到其内部的对象实例呢?

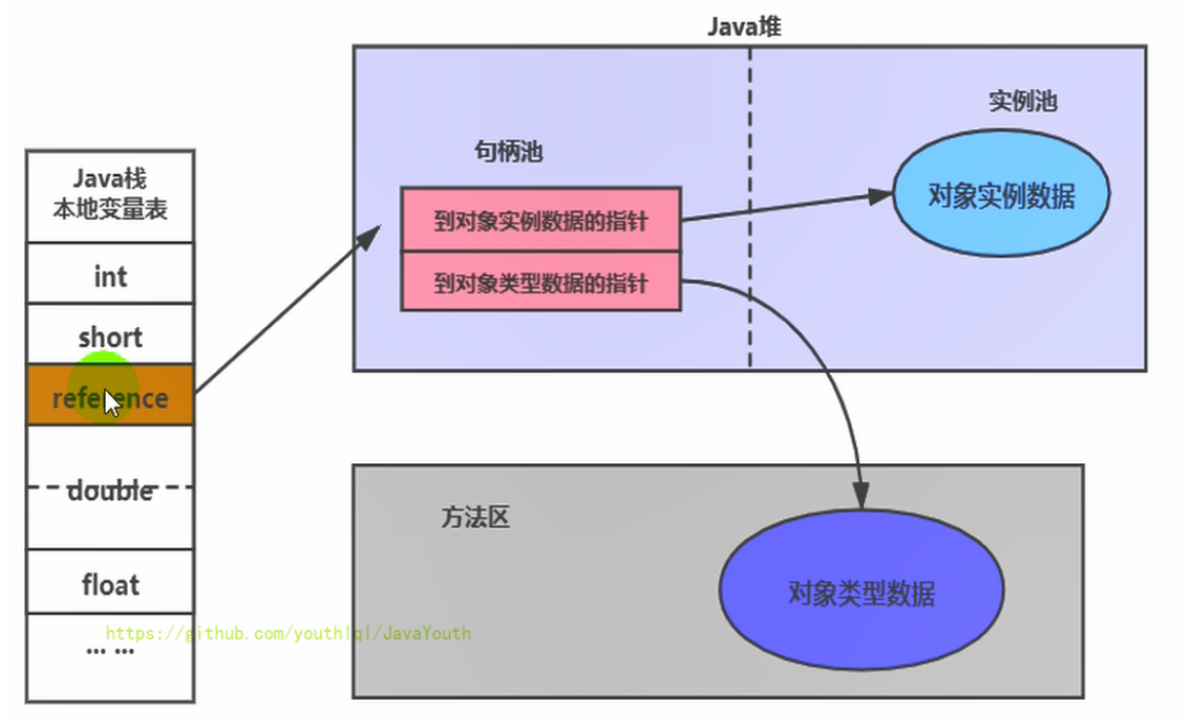


定位，通过栈上reference访问

对象的两种访问方式：句柄访问和直接指针

1、句柄访问

1. 缺点：在堆空间中开辟了一块空间作为句柄池，句柄池本身也会占用空间；通过两次指针访问才能访问到堆中的对象，效率低
2. 优点：reference中存储稳定句柄地址，对象被移动（垃圾收集时移动对象很普遍）时只会改变句柄中实例数据指针即可，reference本身不需要被修改



2、直接指针（HotSpot采用）

1. 优点：直接指针是局部变量表中的引用，直接指向堆中的实例，在对象实例中有类型指针，指向的是方法区中的对象类型数据
2. 缺点：对象被移动（垃圾收集时移动对象很普遍）时需要修改 reference 的值