# 1.对象的实例化

## 大厂面试题

美团：

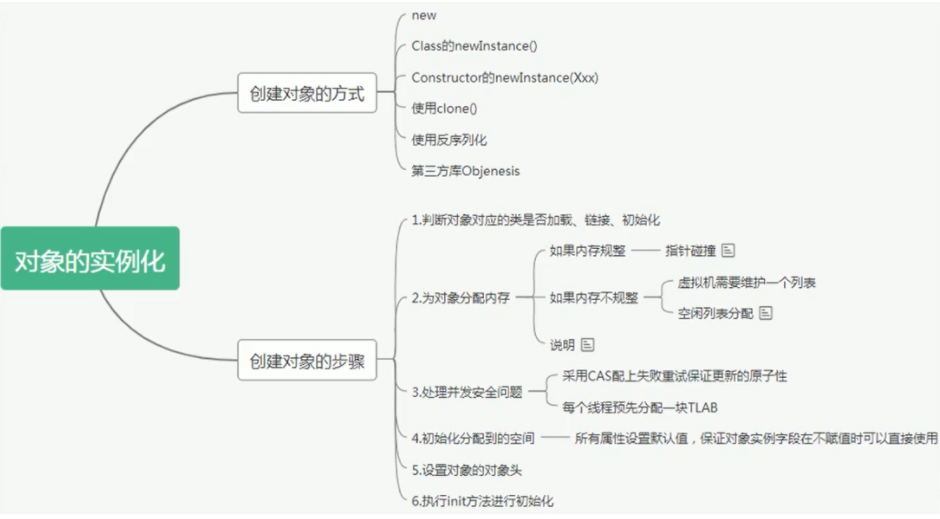
1. 对象在JVM中是怎么存储的？
2. 对象头信息里面有哪些东西？

蚂蚁金服：

二面：java对象头里有什么

****对象实例化****

## 学习大纲



## 创建对象的几种方式

1. new

应用场景：

普通的new;

单利模式中通过类的静态方法来获得实例;

工厂模式;

1. Class对象的newInstance()方法 反射

在JDK9里面被标记为过时的方法

此方式只能调用无参构造器，而且构造器的权限要求必须是public

1. Constructor的newInstance(XXX)方法 反射

此方式可以调用无参、含参构造器，且对构造器权限没有要求

1. 使用clone()

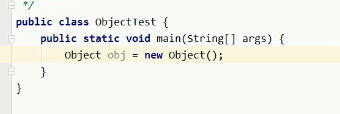
不用任何构造器，只需要类实现Cloneable接口，实现clone()

1. 反序列化

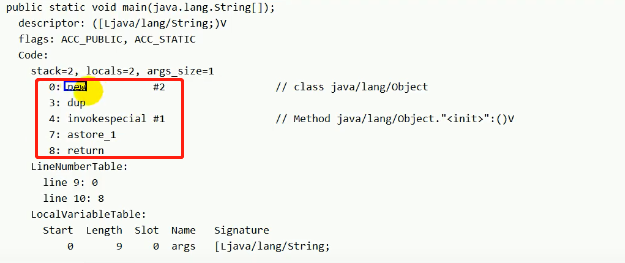
从文件中、网络中获取一个对象的二进制流

1. 第三方库Objenesis

## 创建对象的过程







### 指令解析：

1.new指令，#2 是运行时常量池中的字面量。

（1）当遇到new指令时，首先去检查这个指令的参数是否能够在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经过类加载 初始化过。如果没有就必须进行类加载。

（2）并且为对象分配内存空间、属性的默认初始化。int short byte boolean 引用类型 都占四个字节，所以对象所占堆空间能够确定下来，并且这些对象属性进行默认初始化

2.然后是dup指令

把操作数栈中的引用变量赋值一份，就有两个引用指向堆空间对象实体，栈底的用于赋值操作，上面的用于句柄 其他操作。

3.inovkespecial

调用构造器方法 对对象进行初始化

### 具体过程分析：

#### 判断对象是否进行过类加载

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）。

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。

#### 为对象分配内存空间

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。

(1)对象的内存空间大小计算

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

int short byte boolean 引用类型 都占四个字节，所以对象所占堆空间能够确定下来，并且这些对象属性进行默认初始化

1. 所需要的内存计算好了，开始进行分配

内存分配策略：

1. **指针碰撞**

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。

规整的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，为对象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。

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

1. **空闲列表：**

Java堆内存并不规整，空闲的和被使用的相互交错在一起，此时无法用指针碰撞，JVM需要维护一个空闲列表，记录可用内存块，在分配内存的时候，从空闲列表找一个足够大的空间划分给对象实例，并更新表记录。

**两种分配策略的选择**

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。

#### 处理并发安全问题

对象在创建过程中，是修改指针所指向的位置，并发情况下该指针不是线程安全的。

解决方案：

1. 对分配内存空间的动作进行同步处理

虚拟机采用CAS + 失败重试的方式保证更新操作的原子性。

1. 内存分配动作按照线程划分在不同的空间中进行，也就是每个线程在java堆中预先分配一小块线程私有内存，叫做本地线程分配缓冲区（TLAB）,对象在TLAB中分配，TLAB用完了，分配新的缓存区时才需要同步锁定。

TLAB是否使用，通过-XX:+/-UseTLAB参数来决定。

#### 初始化分配到的空间

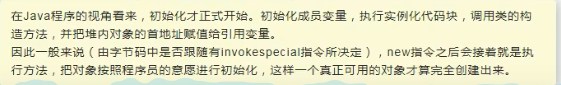
将对象的所有属性设置默认值(不涉及对象头)，保证对象实例字段在不赋值可以直接使用



#### 设置对象的对象头

对象头存储：对象所属的类（类的元数据）、对象的哈希值、对象的GC分代年龄信息、锁信息。

#### 执行init()方法，完成对象的初始化



init()方法包括：

对象属性的 1.显示赋值 2.代码块赋值（按代码顺序） 3.构造器赋值（最后）

#### 总结

前五步骤相当于给捏小人，最后一步相当于上色。

# 2.对象内存布局

## 2.1对象头

对象头包含两部分：运行时元数据（Mark Word）和类型指针

运行时元数据

类型指针

指向类元数据InstanceKlass，确定该对象所属的类型。指向的其实是方法区中存放的类元信息

说明：如果对象是数组，还需要记录数组的长度

### Mark Word

就是对象自身运行时所需要的元数据

* 哈希值（HashCode），可以看作是堆中对象的地址
* GC分代年龄（年龄计数器）
* 锁状态标志
* 线程持有的锁
* 偏向线程ID
* 偏向时间戳
* 如果对象是一个数组，那么对象头中还必须存储数组长度。

**Mark Word在32位和64位虚拟机中，长度分别为32bit和64bit**

### 类型指针

对象指向该对象的类型元数据(存放在方法区中)的指针，此指针用于确定该对象是哪个类的实例。

并不是说要获取对象的类型信息都要经过对象本身（对象头），这要看对象是如何访问定位的。

## 2.2 实例数据

对象真正存储的有效信息。包含在Java程序中定义的各种类型的字段(包括从父类继承下来的，以及自身定义的字段)

存储规则：

1. 相同宽度的字段被分配到一起存放 8字节放一起，2字节放一起....

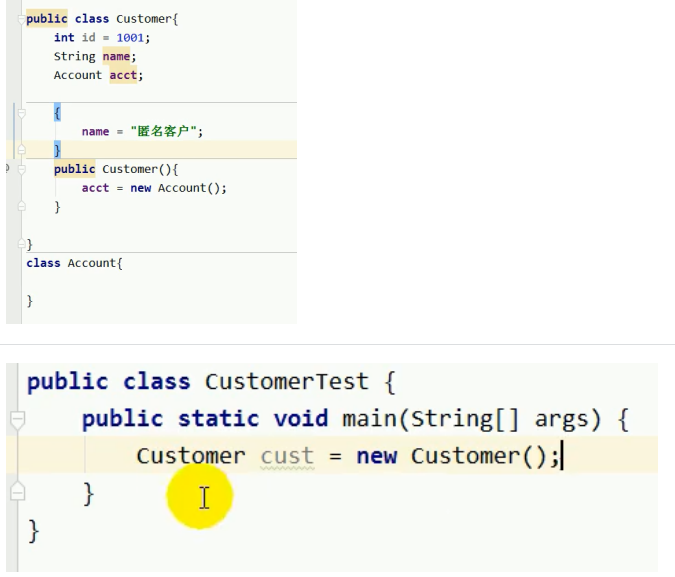
double/long->int->shorts/chars->bytes/booleans->oops

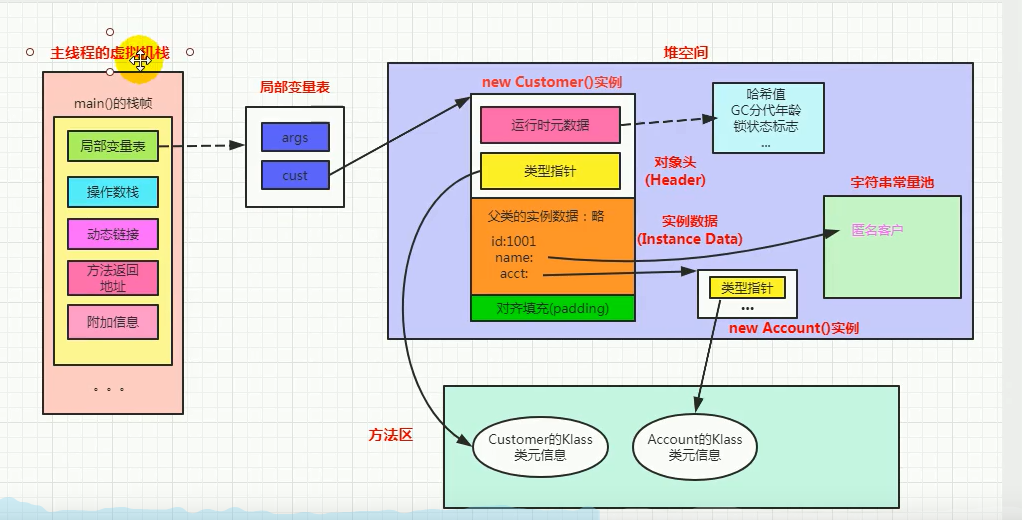
1. 在规则(1)的基础上，父类中定义的变量会出现在子类之前
2. 如果JVM设置 +XX:CompactFields 参数值为true,子类中较窄的变量会允许插入到父类变量的缝隙中，以节省一些空间。

## 2.3对齐填充

无特别含义，作用仅仅是占位符。HotSpot虚拟机自动内存管理系统要求对象起始地址必须是8字节的整数倍，对象的大小必须是8字节的整数倍，如果没对齐，就用这个填充对齐。

## 2.4 创建对象时的内存分布图解



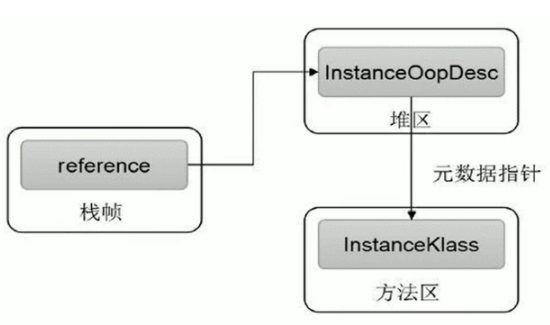


# 3.对象访问定位



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？

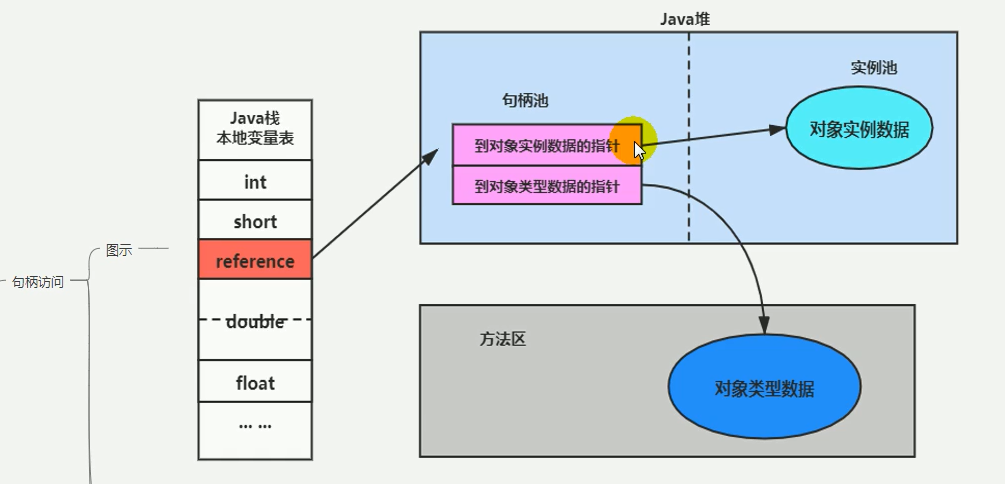
解决如何通过栈上的引用数据来操作堆上的具体对象。



## 对象的两种访问方式

### 1.句柄

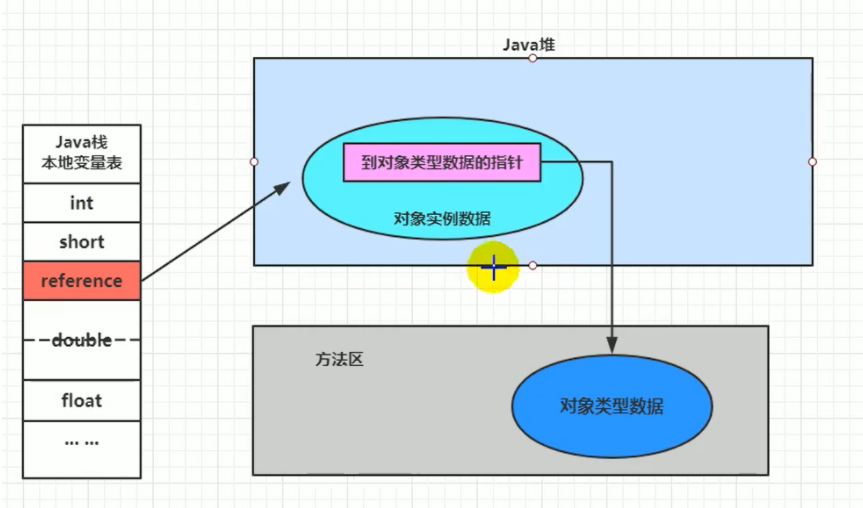
首先，java堆中划分出一块内存作为句柄池，栈上的引用存储的信息就是对象的句柄地址。一个对象对应的句柄中包含了 到对象实例数据的指针 和 到对象类型数据的指针，一个指向堆中的对象实例，一个指向方法区中对象的类型信息存储地址。



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

栈上的引用存储的是java堆中对象实例数据的地址，到对象的类型信息的指针 就存储在对象头中，这种对象访问定位方式 如果要获取对象数据类型数据就必须经过对象实例才可。



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