7.9 对象创建的流程分析

在面向对象编程中，**对象创建**是核心操作之一。理解对象从“代码定义”到“内存实例”的完整流程，不仅能帮我们掌握 Java 底层机制，也是面试高频考点。本节将通过案例拆解对象创建的每一步，清晰还原其背后的运行逻辑。

7.9.1 案例引入：从代码到对象的诞生

我们先看一段简单的代码，观察对象创建的完整过程：

```
class Person {
    // 显式初始化：给 age 赋值 90
    int age = 90;
    // 未显式赋值，默认值为 null
    String name;

    // 构造器：用于对象初始化
    Person(String n, int a) {
        // 构造器内赋值
        name = n;
        age = a;
    }
}

public class ObjectCreation {
    public static void main(String[] args) {
        // 创建对象：触发完整流程
        Person p = new Person("小倩", 20);
    }
}
```

接下来，我们将逐环节分析 `new Person("小倩", 20)` 执行时，Java 虚拟机（JVM）如何完成对象创建。

7.9.2 对象创建的核心流程

对象创建可拆解为 **4 个关键步骤**，每个步骤都有明确的职责，共同完成从“类定义”到“可用对象”的转换。

1. 加载类信息：类的“首次加载”机制

步骤描述：
当 JVM 执行 `new Person(...)` 时，会先检查内存中是否已加载 `Person.class` 的类信息。

- 若未加载：JVM 会通过“类加载器”（ClassLoader）从磁盘读取 `Person.class` 文件，解析成 **运行时类对象**（`Class` 对象），并存储到方法区（MetaSpace）。
- 若已加载：直接复用已存在的类信息（类加载是“懒加载 + 单例”模式，一个类仅加载一次）。

作用：
类信息是对象创建的“模板”，包含属性定义、方法字节码、构造器等内容，必须先加载到内存，才能基于它创建对象。

2. 分配堆内存：为对象 “预留空间”

步骤描述：
JVM 在 **堆内存** 中为新对象分配一块连续的内存空间，用于存储对象的属性值。

细节说明：

- 内存大小由类的属性决定（如 `Person` 的 `age` 占 4 字节，`name` 是引用类型占 8 字节（64 位 JVM），总大小会包含对象头、对齐填充等额外空间）。
- 分配后，内存中的属性会被赋予

  默认初始值

  （这是对象初始化的第一步）：
  - 基本类型（如 `int`）默认值为 `0`；
  - 引用类型（如 `String`）默认值为 `null`。

示例：
执行此步骤后，`Person` 对象在堆中的初始状态为：

```
age = 0（默认值）
name = null（默认值）
```

3. 完成对象初始化：3 层初始化逻辑

对象初始化是**多阶段叠加**的过程，依次执行 **默认初始化** → **显式初始化** → **构造器初始化**，最终确定属性的真实值。

（1）默认初始化

- **时机**：堆内存分配完成后立即执行。
- **操作**：JVM 为对象的所有属性赋予 “类型默认值”（如 `int` 为 `0`，`String` 为 `null`）。
- **作用**：保证属性有 “初始值”，避免未定义的错误。

（2）显式初始化

- **时机**：默认初始化之后执行。
- **操作**：执行类中属性的 “显式赋值代码”（如 `int age = 90;`）。
- 示例

  Person类中age = 90;

  会在此阶段执行，覆盖默认值0，此时对象状态变为：

  ```
  age = 90（显式赋值）
  name = null（未显式赋值，仍为默认值）
  ```

（3）构造器初始化

- **时机**：显式初始化之后执行。
- **操作**：调用构造器（如 `Person(String n, int a)`），执行构造器内的赋值逻辑。
- 示例：

  构造器中name = n;（n为"小倩"）和age = a;（a为20）会覆盖之前的值，最终对象状态为：

```
age = 20（构造器赋值）
name = "小倩"（构造器赋值）
```

4. 返回对象引用：让对象 “被使用”

步骤描述：
堆内存中对象初始化完成后，JVM 会将 **对象在堆中的内存地址** 返回给栈中的引用变量（如 `main` 方法中的 `p` ）。

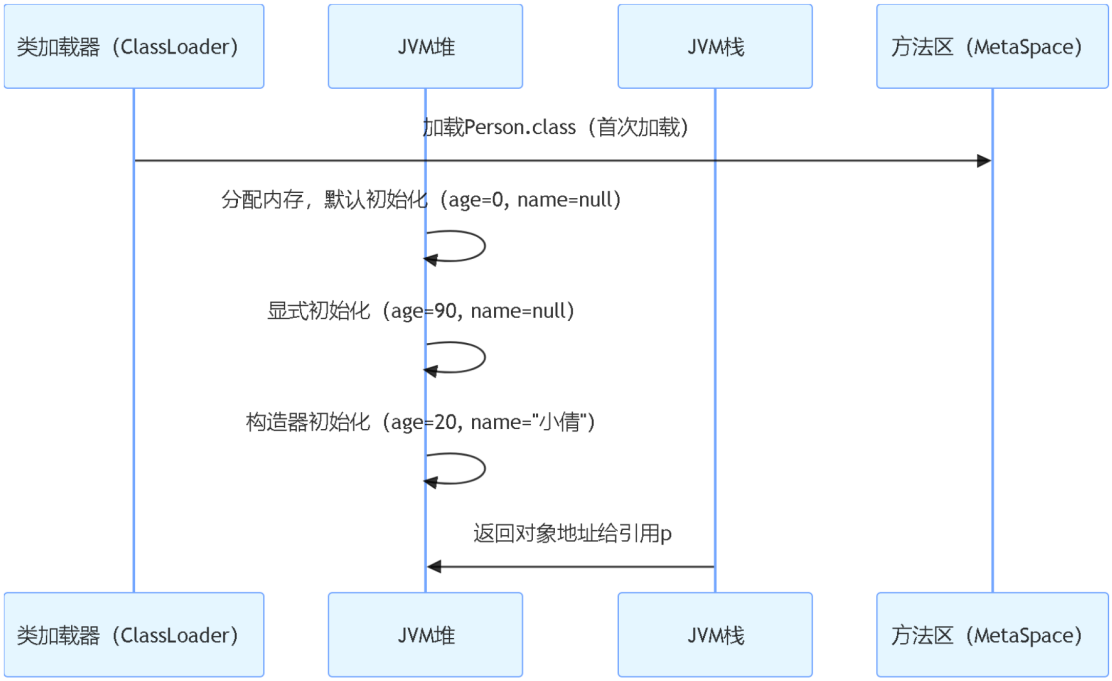
本质：

- `p` 是 “对象引用”，存储在栈中；
- 真正的对象数据存储在堆中，`p` 指向堆中对象的地址。

作用：
通过引用 `p`，我们可以操作堆中的对象（如 `p.name` 访问属性、`p.sayHello()` 调用方法）。

7.9.3 完整流程梳理（面试重点）

为了更清晰地理解，我们用时序图总结对象创建的完整流程：



7.9.4 关键知识点总结

1. **类加载的特点**：
   - 类加载是 “懒加载”：首次使用类时才加载（如执行 `new Person()` ）。
   - 类加载是 “单例”：一个类仅加载一次，后续创建对象直接复用类信息。
2. **初始化的三层逻辑**：
   默认初始化（JVM 赋予默认值）→ 显式初始化（代码中直接赋值）→ 构造器初始化（构造器内赋值），**后执行的步骤会覆盖之前的值。**
3. **堆与栈的分工**：
   - 堆：存储对象的真实数据（属性值）。
   - 栈：存储对象的引用（地址），方便快速访问堆中的对象。

4. **构造器的核心作用**：
   构造器是对象初始化的“最后一步”，用于**最终确定属性值**，保证对象创建后处于“可用状态”。

7.9.5 面试真题模拟

问题 1：对象创建时，属性的初始化顺序是怎样的？

回答：
依次执行 **默认初始化 → 显式初始化 → 构造器初始化。**后执行的步骤会覆盖之前的值，最终属性值由构造器初始化决定（若构造器中赋值）。

问题 2：类加载和对象创建的关系是什么？

回答：
类加载是对象创建的**前提**：JVM 必须先加载类的 `.class` 文件，生成运行时类对象，才能基于它在堆中创建实例。且一个类仅加载一次，可创建多个对象。

问题 3：构造器在对象创建流程中扮演什么角色？

回答：
构造器负责**最终的初始化逻辑**，在默认初始化、显式初始化之后执行，用于为属性赋予“业务所需的值”（如通过参数传入的 `name` 和 `age` ），确保对象创建后即可用。

通过本节学习，你不仅掌握了对象创建的“表面流程”，更理解了 JVM 底层的初始化逻辑。这对编写健壮代码（如避免属性未初始化的空指针问题）、应对面试都有重要意义。下一节我们将学习“`this` 关键字”，它能让构造器的使用更灵活。

课后练习

一、选择题（10 题）

1. 下列关于类加载的说法，正确的是（ ）
   A. 类加载在程序启动时一次性加载所有类
   B. 一个类可以被多次加载，每次加载生成新的 Class 对象
   C. 类加载是对象创建的前提，必须先加载类才能创建对象
   D. 类加载后，类信息存储在堆内存中
2. 对象创建时，堆内存中属性的默认初始化值由谁赋予（ ）
   A. 程序员手动赋值
   B. JVM 自动赋予
   C. 构造器初始化时赋予
   D. 显式初始化时赋予
3. 下列不属于对象初始化三层逻辑的是（ ）
   A. 默认初始化
   B. 显式初始化
   C. 静态初始化
   D. 构造器初始化
4. 对于 `Person p = new Person();`，下列说法正确的是（ ）
   A. `p` 存储在堆内存中
   B. `new Person()` 返回的是对象本身
   C. `p` 存储的是对象在堆中的地址

D. 若 `p = null`，堆中的对象会立即被删除

5. 下列代码中，`age`的最终值是（ ）

```
class User {
    int age = 30;
    User() {
        age = 20;
    }
}
User u = new User();
```

A. 0
B. 20
C. 30
D. 不确定

6. 类加载的 "懒加载" 特性是指（ ）
A. 类永远不会被加载
B. 仅在程序启动时加载必要的类
C. 首次使用类时才加载，不使用则不加载
D. 类加载后会延迟初始化属性

7. 下列关于堆和栈的分工，正确的是（ ）
A. 堆存储对象引用，栈存储对象数据
B. 堆和栈都可存储对象数据
C. 堆存储对象数据，栈存储对象引用
D. 堆和栈都可存储对象引用

8. 构造器在对象创建流程中的作用是（ ）
A. 负责类的加载
B. 分配堆内存
C. 最终确定属性的初始化值
D. 存储类的方法信息

9. 下列哪种情况会触发类加载（ ）
A. 声明类的引用变量（如`Person p;`）
B. 调用类的静态方法（如`Person.getName()`）
C. 仅定义类而不使用
D. 打印类名（如`System.out.println("Person")`）

10. 若一个类的属性未显式赋值且构造器中也未赋值，则该属性的值是（ ）
A. 编译错误
B. 随机值
C. 类型默认值
D. 由 JVM 随机决定

二、填空题（10 题）

1. 对象创建的 4 个核心步骤依次是：加载类信息、__、完成对象初始化、返回对象引用。
2. 类加载器读取磁盘上的__文件，解析成运行时类对象。
3. 对象的属性初始化顺序为：默认初始化 → __ → 构造器初始化。
4. 堆内存中存储对象的__，栈内存中存储对象的__。
5. 类加载的 "单例" 特性是指__。
6. 当执行`new Person("张三", 18)`时，构造器的作用是__。

7. 基本类型`boolean`的默认初始化值是__，引用类型的默认初始化值是__。
8. 方法区（MetaSpace）用于存储__。
9. 若一个类从未被使用，则它的类信息__（会 / 不会）被加载到内存。
10. 构造器初始化是对象初始化的__（第一步 / 最后一步），用于保证对象创建后处于可用状态。

三、判断题（10 题）

1. 类加载后，可基于该类创建多个对象。（ ）
2. 对象的属性初始化时，显式初始化一定会覆盖默认初始化的值。（ ）
3. `new Person()`执行时，若类未加载，则先加载类再分配堆内存。（ ）
4. 堆内存中分配的对象空间大小仅由类的属性数量决定。（ ）
5. 构造器可以独立于`new`关键字被调用。（ ）
6. 一个类的`.class`文件可能被多个类加载器加载，生成多个 Class 对象。（ ）
7. 若构造器中对属性赋值，则该值会覆盖显式初始化的值。（ ）
8. 声明引用变量（如`Person p;`）会触发对象创建。（ ）
9. 对象的引用变量存储在栈中，若引用变量被赋值为`null`，则堆中的对象会立即被回收。（ ）
10. 类加载是 "懒加载"，即只有当执行`new`关键字时才会加载类。（ ）

四、简答题（10 题）

1. 简述类加载与对象创建的关系。
2. 解释对象属性的三层初始化逻辑及其执行顺序。
3. 为什么说构造器是保证对象 "可用" 的关键？
4. 堆内存和栈内存在对象存储中各有什么作用？
5. 类加载的 "懒加载" 和 "单例" 特性分别是什么？
6. 执行`Person p = new Person();`时，若`Person`类未加载，JVM 会执行哪些操作？
7. 若一个属性在显式初始化时赋值为 5，构造器中赋值为 10，则最终属性值是多少？为什么？
8. 为什么对象的引用变量存储在栈中，而对象数据存储在堆中？
9. 类的静态成员（如静态属性、静态方法）的加载时机与实例成员有何不同？
10. 简述 "默认初始化" 的作用。

五、编程题（10 题）

1. 定义一个`Book`类，包含属性`name`（String）和`price`（double），显式初始化`price`为 50.0，定义构造器接收`name`和`price`参数并赋值。创建`Book`对象时，输出属性的默认初始化值、显式初始化值和构造器初始化值。
2. 编写代码验证：类加载仅执行一次（提示：可在静态代码块中打印信息，多次创建对象观察输出）。
3. 定义一个`Student`类，属性`score`未显式赋值，构造器中也不赋值，创建对象后打印`score`的值，解释结果。
4. 编写代码证明：引用变量存储在栈中，对象数据存储在堆中（提示：通过两个引用指向同一对象，修改其中一个引用的属性值，观察另一个引用的属性值变化）。
5. 定义`Teacher`类，包含属性`subject`（String），显式初始化`subject`为 "数学"，构造器中赋值为 "语文"，创建对象后打印`subject`的值，说明初始化顺序。
6. 编写代码触发类加载（至少两种方式），并观察类加载的时机。
7. 定义`Car`类，属性`color`（String）和`speed`（int），通过构造器初始化所有属性，创建 3 个不同的`Car`对象，打印每个对象的属性值。
8. 验证 "默认初始化值"：定义包含所有基本类型和引用类型属性的`Test`类，创建对象后打印所有属性值。
9. 编写代码演示：当引用变量为`null`时，调用对象方法会发生什么（提示：观察是否抛出异常）。

10. 定义`Person`类，包含属性`age`，构造器中判断`age`是否为负数，若为负数则赋值为 0，创建对象时传入负数，验证构造器对属性的初始化控制。