

第18课

对象军团出发

引用 对象交互与 null

两张标签可能找到同一只狼

shadow

alias

宠物对象

name = 影子
health = 17

会把对象变量理解为引用；能解释两个变量指向同一对象的效果；能在使用对象前处理 null；能通过参数让玩家与宠物安全互动。

前置背包：会声明类、创建对象、访问字段并调用实例方法。今天要看清对象变量这张“坐标纸条”到底指向哪里。

阅读方式：先猜结果，再运行核对；每次只改一个条件。

本章使用 JDK 21，无需预览功能。故事与插画借用方块世界，练习运行在普通 Java 控制台。

开始前：把示例放对地方

本章使用 JDK 21，不需要预览功能。每个普通示例都给出完整程序，按标注的文件名保存。多个示例各放在自己的文件夹，避免辅助类同名。

终端命令：把中文占位符替换成实际名称

```
javac -encoding UTF-8 示例文件名.java  
java 示例类名
```

类名区分大小写。完整挑战代码若分成多段，请按顺序接在同一个文件中，不要为续段另建类。运行核对只列程序输出；需要输入的地方另给输入序列。

学习动作：先写下你的预测，运行后比较，再只改一个条件。标有“故意错误”的实验用于认识 Bug；它们与可直接运行的示例有清楚区分。

01 引用像对象坐标

对象变量不像 `int` 那样直接装着全部数据。它保存一个引用，让程序能找到对象并访问字段或方法。

对象变量保存一张引用

wolf

宠物对象

name = 影子
health = 20

L18Example01.java • 完整独立示例

```
class PetWolf {  
    String name;  
    int health = 20;  
    int trust;  
    void heal(int n) {  
        if (n > 0) health += Math.min(n, 20 - health);  
    }  
    void eat() { trust += Math.min(10, 100 - trust); }  
}  
public class L18Example01 {  
    public static void main(String[] args) {  
        PetWolf wolf = new PetWolf();  
        wolf.name = "影子";  
        System.out.println(wolf.name);  
    }  
}
```

运行核对

影子

阅读路线

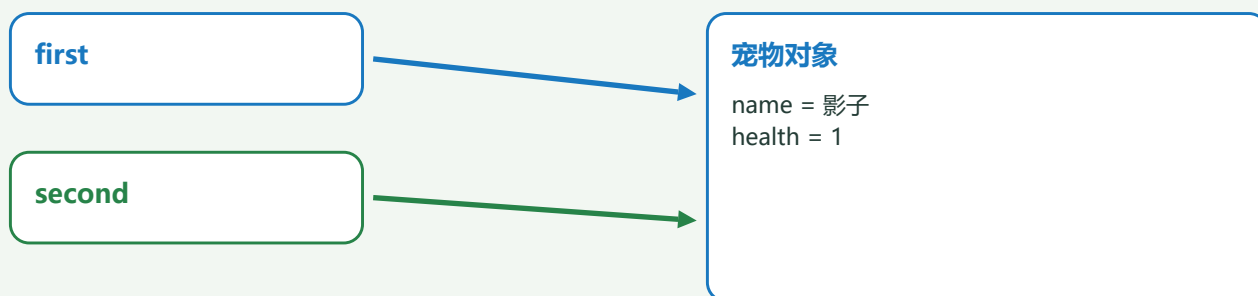
- `new PetWolf()` 创建对象。
- `wolf` 保存指向该对象的引用。
- `wolf.name` 顺着引用找到字段。
- `wolf.heal(3)` 顺着引用调用方法。

不是复制箱子：把 `wolf` 赋给另一个对象变量时，默认不会复制一只新狼，只会多一张指向原对象的纸条。

02 两个变量共享一个对象

引用赋值会复制引用。first 和 second 指向同一对象，因此通过 second 修改 health 后，first 读取到的也是新值。

两张引用可以指向同一只宠物



箭头表示引用关系；不会自动复制对象。

L18Example02.java • 完整独立示例

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
}

public class L18Example02 {
    public static void main(String[] args) {
        PetWolf first = new PetWolf();
        PetWolf second = first;
        second.health = 1;
        System.out.println(first.health);
        System.out.println(first == second);
    }
}
```

运行核对

```
1
true
```

侦探问题：若你本来想复制宠物，引用赋值并没有做到。以后可用构造器或复制方法明确创建新对象。

03 null 表示没有对象

对象变量可以暂时不指向任何对象，此时值是 null。点号左边若为 null，程序无法找到字段或方法。

空引用没有坐标可走

null

当前没有对象

检查

先判断是否非null

存在

调用对象方法

缺失

提示或返回失败

L18Example03.java • 完整独立示例

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
}

public class L18Example03 {
    public static void main(String[] args) {
        PetWolf selected = null;
        if (selected != null) System.out.println(selected.name);
        else System.out.println("还没有选择宠物");
    }
}
```

运行核对

还没有选择宠物

空箱提示：null 不是空字符串，也不是生命为0的对象。它表示当前没有对象可用。

04 把对象传给方法

对象可以作为参数传入方法。传进去的仍是引用，所以方法修改对象状态，调用处能看见变化。

方法参数也能接收对象引用

实参

传入一个引用值

形参

接收该引用值的副本

改对象

调用者看到状态变化

改参数

不改变调用者的引用

Java 始终按值传递。对象参数得到引用值的副本：沿副本修改同一对象，调用者能看见；把参数改指向新对象，不会改变调用者的 wolf 变量。

参数检查：公共帮助方法接收对象时，先决定是否允许 null。允许就明确处理，不允许就尽早拒绝。

04 把对象传给方法（续）

L18Example04.java • 完整独立示例

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
}
public class L18Example04 {
    static void healPet(PetWolf pet, int n) {
        if (pet != null) pet.heal(n);
    }
    static void replacePet(PetWolf pet) { pet = new PetWolf(); }
    public static void main(String[] args) {
        PetWolf wolf = new PetWolf(); wolf.health = 12;
        healPet(wolf, 5);
        replacePet(wolf);
        healPet(null, 5);
        System.out.println(wolf.health);
    }
}
```

运行核对

17

05 对象通过方法互动

对象互动不是让一个类随便改另一个类的字段。更清楚的方式是调用对方的方法。

玩家发出动作 宠物维护自己的状态

Player

负责选择互动对象

feed

调用宠物eat

PetWolf

维护信任范围

缺失

明确处理null

L18Example05.java • 完整独立示例

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
}
class Player {
    void feed(PetWolf pet) { if (pet != null) pet.eat(); }
}
public class L18Example05 {
    public static void main(String[] args) {
        Player alex = new Player();
        PetWolf pet = new PetWolf();
        alex.feed(pet);
        alex.feed(null);
        System.out.println(pet.trust);
    }
}
```

运行核对

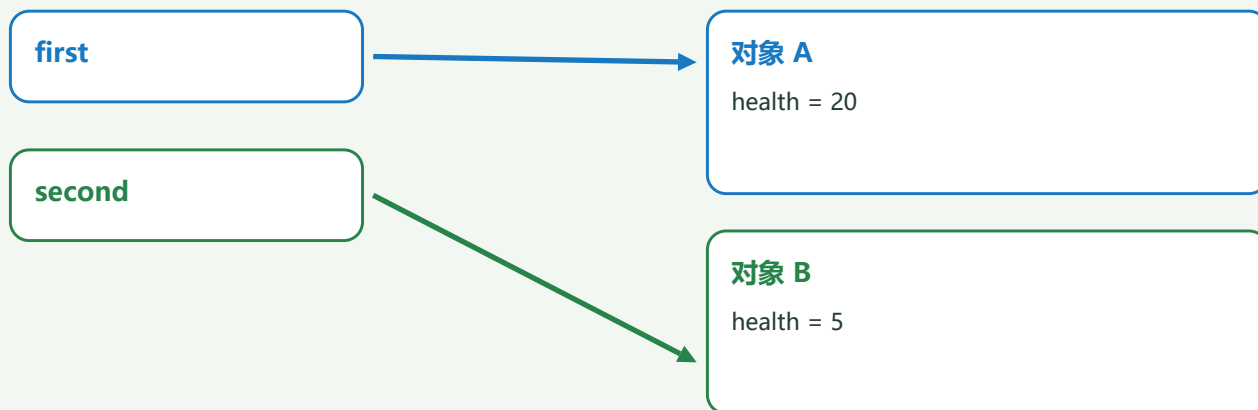
10

敲门访问：Player 不直接写 `pet.trust += 10`，而是调用 `pet.eat()`。宠物自己守住信任值上限。

06 改引用和改对象不同

`second.health = 5` 修改对象；`second = new PetWolf()` 改变 `second` 保存的引用。`first` 仍指向旧对象。

重新赋值只改变那张引用



L18Example06.java • 完整独立示例

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
}

public class L18Example06 {
    public static void main(String[] args) {
        PetWolf first = new PetWolf();
        PetWolf second = first;
        second = new PetWolf();
        second.health = 5;
        System.out.println(first.health + "," + second.health);
        System.out.println(first == second);
    }
}
```

运行核对

20,5
false

画箭头：遇到引用题就画对象方块和变量箭头。箭头一画，末影人也很难把逻辑瞬移走。

07 引用相等和内容相等

对对象使用 `==`，比较的是两个引用是否指向同一对象。两只同名宠物仍可能是不同对象。

双等号比较是否指向同一对象

==

比较引用身份

sameName

本例比较名字内容

同名

不代表同一对象

equals

是否重写会影响语义

L18Example07.java • 完整独立示例

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
    boolean sameName(PetWolf other) {
        return other != null && java.util.Objects.equals(name, other.name);
    }
}

public class L18Example07 {
    public static void main(String[] args) {
        PetWolf a = new PetWolf(); a.name = "影子";
        PetWolf b = a;
        PetWolf c = new PetWolf(); c.name = "影子";
        System.out.println(a == b);
        System.out.println(a == c);
        System.out.println(a.sameName(c));
    }
}
```

运行核对

```
true
false
true
```

别混淆：String 的 equals 比较文字内容；普通自定义对象若没重写 equals，仍使用引用身份规则。

08 对象军团测试台

测试对象程序时，记录变量指向哪个对象，以及每个对象当前字段值。

测试引用图比只看输出更可靠

两次new

验证独立状态

共享赋值

验证修改可见

重新指向

验证旧引用未变化

null

验证安全拒绝

验收表

场景	操作	预期
独立	两次new	两个对象互不影响
共享	b = a	b修改后a也看到
改引用	b = new PetWolf()	a仍指向旧对象
空引用	b = null	检查后不调用方法

证据优先：不要只说“应该一样”。写出 a==b 的结果，并打印两个对象的生命值。

09 null 故障诊所

NullPointerException 通常给出具体行号。找到该行所有点号，检查每个对象变量的来源。

沿着异常栈找到第一个危险点号

看行号

定位自己的代码

找引用

检查点号左边

查来源

追踪赋值与返回

修复

为缺失对象安排分支

L18Example09.java • 完整独立示例

```
class PetWolf {
    String name;
    int health = 20;
    int trust;
    void heal(int n) {
        if (n > 0) health += Math.min(n, 20 - health);
    }
    void eat() { trust += Math.min(10, 100 - trust); }
}

public class L18Example09 {
    static PetWolf findPet(String name) { return null; }
    public static void main(String[] args) {
        PetWolf found = findPet("影子");
        if (found != null) found.heal(5);
        else System.out.println("没有这只宠物");
    }
}
```

运行核对

没有这只宠物

本例用“总是没找到”的最小查找函数测试缺失分支。真正的选择系统要在宠物列表中查找；无法找到时返回 null，调用处明确处理。

09 null 故障诊所（续）

急诊顺序

- 读异常类型和第一条属于自己代码的行。
- 圈出点号左边的对象变量。
- 追踪它从哪里赋值或返回。
- 选择检查 null、提供默认对象或拒绝输入。

别吞异常：不要用空 catch 假装修好。真正的修复是让对象缺失成为明确分支。

10 挑战 蓝图 玩家与宠物

挑战 程序叫 PlayerPetTeam。玩家持有当前宠物引用；选择成功后才能喂食或治疗。

三个对象方法完成喂食 治疗 and 选择

Player.selected
保存当前选择

PetWolf
维护生命与信任

互动
调用宠物方法

取消
null清除选择

验收条件

- 两只宠物是独立对象。
- 玩家选择第一只宠物后喂食，信任值增加。
- 把另一个变量指向同一宠物并治疗，原引用看到新生命。
- 取消选择设为null后，互动返回false而不是崩溃。

验收表

方法	输入	输出或效果
select	PetWolf	保存当前引用
feedSelected	无	信任增加或false
healSelected	数量	生命增加或false
status	无	打印宠物状态

接口分工：Player 决定选择谁；PetWolf 决定吃东西和治疗后状态是否合法。

11 挑战 实现 玩家宠物队

新建 PlayerPetTeam.java。运行前预测每行输出，再画出 alex.selected、alias、shadow 与 snow 指向的对象。

引用关系在完整程序里变成可验证行为

选择影子

selected指向影子

喂食

信任增加10

alias治疗

原对象生命变17

取消

互动返回false

验收：解释 alias 为什么能改变 shadow；确认 snow 不受影响；取消选择后程序仍正常结束。

PlayerPetTeam.java • 第1/2段，顺序接在同一文件

```
public class PlayerPetTeam {
    static class PetWolf {
        String name;
        int health = 20;
        int trust = 0;
        void eat() { trust += Math.min(10, 100 - trust); }
        void heal(int n) {
            if (n > 0) health += Math.min(n, 20 - health);
        }
        void status() {
            System.out.println(name + " 生命=" + health + " 信任=" + trust);
        }
    }
    static class Player {
        String name;
        PetWolf selected;
        void select(PetWolf pet) { selected = pet; }
        boolean feedSelected() {
            if (selected == null) return false;
            selected.eat(); return true;
        }
        boolean healSelected(int n) {
            if (selected == null || n <= 0) return false;
            selected.heal(n); return true;
        }
    }
    public static void main(String[] args) {
        PetWolf shadow = new PetWolf(); shadow.name = "影子";
```

11 挑战 实现 玩家宠物队（续）

PlayerPetTeam.java • 第2/2段，顺序接在同一文件

```
PetWolf snow = new PetWolf(); snow.name = "雪球";
shadow.health = 12;
Player alex = new Player(); alex.name = "Alex";
alex.select(shadow);
System.out.println(alex.feedSelected());
PetWolf alias = shadow;
alias.heal(5);
shadow.status();
snow.status();
alex.select(null);
System.out.println(alex.healSelected(3));
    }
}
```

运行核对

```
true
影子 生命=17 信任=10
雪球 生命=20 信任=0
false
```

12 拓展任务与通关复盘

作业除代码外，再交一张引用图：变量画成标签，对象画成方块，用箭头连接。

三档任务训练对象箭头

木镐

证明共享修改可见

铁镐

证明参数按值传递

钻石镐

三只宠物选择系统

复盘

画变量与对象箭头

三档作业

- 木镐：创建一个 Mob，让两个变量共享它并证明修改可见。
- 铁镐：写方法 heal(Mob target, int amount)，安全处理 null，并证明给参数重新赋值不改变调用者的引用。
- 钻石镐：制作三只宠物的选择系统，支持选择、取消、喂食和状态查询。

验收表

自评	能	还要练
我能画引用箭头	☐	☐
我能解释共享对象	☐	☐
我会安全处理null	☐	☐

下一课：第19课研究对象的生成点。构造方法会在对象出生时完成初始化和第一轮数据体检。

自己的验收记录

运行之前写预期，运行之后记录实际结果。正常案例说明程序会做事；边界案例说明它知道何时停；错误案例说明它能拒绝不合理的请求。

把证据写进记录

类别	你选择的输入	预期与实际
正常	一个常用值或一次成功操作	是否逐行一致？
边界	最小值、最大值及相邻值	有没有少一格、多一步？
错误	非法数量、缺失对象或错误语法	是否明确拒绝？状态是否保持？

- 把一个 Bug 写成：触发条件、现象、原因、最小修复。
- 整理一个值得记住的规则，用自己的话解释，不能只复制定义。
- 改变一个示例，保留测试记录，让结果证明你的理解。

技术核对依据：Java SE 21 语言规范，作用域与访问控制（第6章）、类与构造器（第8章）、方法调用（第15章）。 <https://docs.oracle.com/javase/specs/jls/se21/html/index.html>