static_final

@M了个J
李明杰

https://github.com/CoderMJLee
http://cnblogs.com/mjios

小码哥教育 SEEMYGO
实力IT教育 www.520it.com

码拉松

小码哥教育 SEEMYGO

static

- static 常用来修饰类的成员：成员变量、方法、嵌套类

- 成员变量
  - 被 static 修饰：类变量，静态变量，静态字段
    - ✓ 在程序运行过程中只占用一份固定的内存（存储在方法区）
    - ✓ 可以通过实例、类访问
  - 没有被 static 修饰：实例变量
    - ✓ 在每个实例内部都有一份内存
    - ✓ 只能通过实例访问，不可以通过类访问

- 方法
  - 被 static 修饰：类方法、静态方法
    - ✓ 可以通过实例、类访问
    - ✓ 内部不可以使用 this
    - ✓ 可以直接访问类变量、类方法
    - ✓ 不可以直接访问实例变量、实例方法
  - 没有被 static 修饰：实例方法
    - ✓ 只能通过实例访问，不可以通过类访问
    - ✓ 内部可以使用 this
    - ✓ 可以直接访问实例变量、实例方法
    - ✓ 可以直接访问类变量、类方法

- 不推荐使用实例访问类变量、类方法
- 在同一个类中
  - 不能有同名的实例变量和类变量，不能有相同签名的实例方法和类方法

小码哥教育 SEEMYGO

静态导入

- 使用了静态导入后，就可以省略类名来访问静态成员（成员变量、方法、嵌套类）

```
package com.mj.model;

public class Person {
    public static int age = 1;
    public static void show() {
        System.out.println("age is " + age);
    }
    public static class Foot {
        public void run() {
            System.out.println("run");
        }
    }
}
```

```
import static com.mj.model.Person.*;

System.out.println(age); // 1
show(); // age is 1
Foot foot = new Foot();
foot.run(); // run
```

小码哥教育 SEEMYGO

静态导入的经典使用场景

```
import static java.lang.Math.PI;

System.out.println(2 * PI * 10);
System.out.println(2 * PI * 20);
```

- 正确使用静态导入，可以消除一些重复的类名，提高代码可读性
- 过度使用静态导入，会让读者分不清静态成员是在哪个类中定义的
- 建议：谨慎使用静态导入

成员变量的初始化

- 编译器会自动为未初始化的成员变量设置初始值

- 如何手动给实例变量提供初始值?
  - 在声明中
  - 在构造方法中
  - 在初始化块中
    - ✓ 编译器会将初始化块复制到每个构造方法的头部（每创建一个实例对象，就会执行一次初始化块）

- 如何手动给类变量提供初始值?
  - 在声明中
  - 在静态初始化块中
    - ✓ 当一个类被初始化的时候执行静态初始化块
    - ✓ 当一个类第一次被主动使用时，JVM 会对类进行初始化

小码哥教育 SEEMYGO

初始化块、静态初始化块

```
public class Person {
    static { // 静态初始化块
        System.out.println("static block");
    }
    { // 初始化块
        System.out.println("block");
    }
    public Person() {}
    public Person(int age) {}
}

new Person();
// static block
// block
new Person(20);
// block
```

- 可以有多个（静态）初始化块，按照在源码中出现的顺序被执行

小码哥教育 SEEMYGO

初始化块、静态初始化块

```
public class Person {
    static {
        System.out.println("Person static block");
    }
    { System.out.println("Person block"); }
    public Person() {
        System.out.println("Person constructor");
    }
}
```

```
public class Student extends Person {
    static {
        System.out.println("Student static block");
    }
    { System.out.println("Student block"); }
    public Student() {
        System.out.println("Student constructor");
    }
}
```

```
new Student();
/*
Person static block
Student static block
Person block
Person constructor
Student block
Student constructor
 */
```

小码哥教育 SEEMYGO

单例模式（Singleton Pattern）

- 如果一个类设计成单例模式，那么在程序运行过程中，这个类只能创建一个实例

```
// 饿汉式
public class Rocket {
    private static Rocket instance = new Rocket();
    private Rocket() {}
    public static Rocket getInstance() {
        return instance;
    }
}
```

```
// 懒汉式（有线程安全问题）
public class Rocket {
    private static Rocket instance = null;
    private Rocket() {}
    public static Rocket getInstance() {
        if (instance == null) {
        instance = new Rocket();
        }
        return instance;
    }
}
```

小码哥教育 SEEMYGO

final

- 被 `final` 修饰的类：不能被子类化，不能被继承
- 被 `final` 修饰的方法：不能被重写
- 被 `final` 修饰的变量：只能进行1次赋值

常量（Constant）

- 常量的写法

```
public static final double PI = 3.14159265358979323846;
private static final int NOT_FOUND = -1;
```

- 如果将基本类型或字符串定义为常量，并且在编译时就能确定值
  - 编译器会使用常量值替代各处的常量名（类似于 C 语言的宏替换）
  - 称为编译时常量（ compile-time constant）