Java内部类的深入理解及各种内部类的访问

刘Java 2021-07-19 09:54 2327

详细介绍了Java中的内部类的概念，及各种内部类的访问案例演示。包括非静态内部类、静态内部类、局部内部类、匿名内部类。

1 内部类的定义

内部类是指在一个类的内部再定义一个类。内部类作为外部类的一个成员，并且依附于外部类而存在的。内部类可为静态，可用protected和private修饰（而外部类只能使用public和缺省的包访问权限）。

内部类主要有以下几类：**成员内部类、局部内部类、静态内部类、匿名内部类。**

2 内部类的特性

1. 内部类仍然是一个独立的类，在编译之后内部类会被编译成独立的.class文件，但是前面冠以外部类的类名和$符号 。
2. 内部类不能用普通的方式访问。
3. 内部类声明成静态的，就不能随便的访问外部类的成员变量了，此时内部类只能访问外部类的静态成员变量 。
4. 外部类不能直接访问内部类的的成员，但可以通过内部类对象来访问。

3 静态内部类与普通内部类的区别

1. 静态内部类不持有外部类的引用 在普通内部类中，我们可以直接访问外部类的属性、方法，即使是private类型也可以访问，这是因为内部类持有一个外部类的引用，可以自由访问。而静态内部类，则只可以访问外部类的静态方法和静态属性（即使是private权限也能访问，这是由其代码位置所决定的），其他则不能访问。
2. 静态内部类不依赖外部类 普通内部类与外部类之间是相互依赖的关系，内部类实例不能脱离外部类实例，也就是说它们会同生同死，一起声明，一起被垃圾回收器回收。而静态内部类是可以独立存在的，即使外部类消亡了，静态内部类还是可以存在的。

类，没有任何限制。

4 为什么普通内部类不能有静态变量

1. 成员内部类 之所以叫做成员 就是说他是类实例的一部分 而不是类的一部分。
2. 结构上来说 他和你声明的成员变量是一样的地位 一个特殊的成员变量 而静态的变量是类的一部分和实例无关。
3. 你若声明一个成员内部类 让他成为主类的实例一部分 然后又想在内部类声明和实例无关的静态的东西 你让JVM情何以堪啊。
4. 若想在内部类内声明静态字段 就必须将其内部类本身声明为静态。
5. 非静态内部类有一个很大的优点：可以自由使用外部类的所有变量和方法。

5 内部类的加载

1. 内部类是延时加载的，也就是说只会在第一次使用时加载。不使用就不加载，所以可以很好的实现单例模式。
2. 不论是静态内部类还是非静态内部类都是在第一次使用时才会被加载。
3. 对于非静态内部类是不能出现静态模块（包含静态块，静态属性，静态方法等）
4. 非静态类的使用需要依赖于外部类的对象。

简单来说，类的加载都是发生在类要被用到的时候。内部类也是一样：

1. 普通内部类在第一次用到时加载，并且每次实例化时都会执行内部成员变量的初始化，以及代码块和构造方法。
2. 静态内部类也是在第一次用到时被加载。但是当它加载完以后就会将静态成员变量初始化，运行静态代码块，并且只执行一次。当然，非静态成员和代码块每次实例化时也会执行。

总结一下Java类代码加载的顺序，万变不离其宗：

1. 规律一、初始化构造时，先父后子；只有在父类所有都构造完后子类才被初始化
2. 规律二、类加载先是静态、后非静态、最后是构造函数。静态构造块、静态类属性按出现在类定义里面的先后顺序初始化，同理非静态的也是一样的，只是静态的只在加载字节码时执行一次，不管你new多少次，非静态会在new多少次就执行多少次
3. 规律三、java中的类只有在被用到的时候才会被加载。
4. 规律四、java类只有在类字节码被加载后才可以被构造成对象实例。

6 内部类的访问

6.1 非静态内部类

6.1.1 通过外部类的普通方法访问

步骤：

1. 创建调用方法:
   i. 在外部类的普通方法中创建内部类对象。
   ii. 外部类的普通方法中通过内部类对象调用内部类方法。
2. 主方法中调用：
   i. 创建外部类对象。
   ii. 通过外部类对象调用该外部类普通方法。

6.1.2 通过外部类的静态方法访问

步骤：

1. 创建外部类对象。
2. 通过外部类引导 创建一个内部类对象。
3. 内部类对象调用内部类的方法。
4. 主方法中：直接调用该外部类静态方法。

也可以直接在main方法执行（1） （2） （3）

> 外部类名.内部类名 对象名 = 外部类对象.内部类对象; Outer.Inner oi = new Outer().new Inner();

注意：

1. 当内部类当中定义了和外部类同名的成员变量，采用就近原则的方式访问。
2. 内部类中要想访问外部类的非静态成员，需要使用外部类进行引导：外部内名.this.属性的名称;想要访问外部类静态成员时，可以省略this：外部类名.属性的名称。

6.1.3 案例

java 复制代码

```java
public class Outer1 {
    private static int outer1;
    private int outer2;

    /**
     * 非静态内部类
     */
```

```
10          private int inner2;
11
12          public Inner1() {
13              //非静态内部类访问外部类的静态成员   外部类名.属性的名称
14              this.inner1 = Outer1.outer1;
15              //非静态内部类访问外部类的非静态成员   外部内名.this.属性的名称
16              this.inner2 = Outer1.this.outer2;
17          }
18      }
19
20      /**
21       * 通过外部类的普通方法访问非静态内部类
22       *
23       * @return Inner1
24       */
25      public Inner1 getInner1() {
26          //直接new内部类对象返回
27          return new Inner1();
28          //或者
29          //return new Outer1.Inner1();
30      }
31
32      /**
33       * 通过外部类的普通方法访问非静态内部类
34       *
35       * @return Inner1
36       */
37      public static Inner1 getInner1Sta() {
38          //通过外部类引导 创建一个内部类对象,然后返回
39          return new Outer1().new Inner1();
40      }
41
42  }
```

6.2 静态内部类

6.2.1 通过外部类的静态方法或者是非静态方法访问

实现步骤：

1. 创建内部类对象。
2. 通过内部类对象直接调用。

> 外部类名.内部类名 对象名 = new 外部类名.内部类名(); Outer.Inner oi = new Outer.Inner();

内部类的静态成员直接通过：外部类名.内部类名.成员变量/方法 访问

当内部类当中定义了和外部类同名的成员变量，采用就近原则的方式访问。 要想访问外部类的成员，需要使用外部类进行引导： 外部类类名.属性的名称;并且该外部类成员是被static修饰的。

总结:

1. 静态内部类不能直接访问外部类的非静态成员。
2. 内部类当中定义了和外部类同名的成员变量时候，需要使用类名直接引导。
3. 定义的静态内部类，在类加载的时候，会自动提升为一个顶层的类。就是一个独立的类，相当于定义在类的外面，在创建对象的时候，不需要使用任何对象引导。

6.2.2 案例

在上面的外部内中在定义一个静态内部类

java 复制代码

```java
/**
 * 静态内部类
 */
public static class Inner2 {
    private int inner3;
    private int inner4;

    public Inner2() {
        //静态内部类访问外部类的静态成员  外部类名.属性的名称 或者直接 属性的名称(
        this.inner3 = Outer1.outer1;
        this.inner3 = outer1;
        //静态内部类访问外部类的非静态成员  无法直接访问
        //this.inner2 = Outer1.this.outer2;
    }

}

/**
 * 通过外部类的普通方法访问静态内部类
 *
 * @return Inner2
 */
public Inner2 getInner2() {
    //直接new内部类对象返回
    return new Inner2();
    //或者
    //return new Outer1.Inner2();
}

/**
```

```
     * @return Inner2
     */
    public static Inner2 getInner2Sta() {
        //直接new内部类对象返回
        return new Inner2();
        //或者
        //return new Outer1.Inner2();
    }
```

6.3 局部内部类

> 在方法的内部定义的内部类就是局部内部类。

要点:

1. 该内部类没有任何的访问控制权限。只能在方法的内部有效。
2. 外部类看不见方法中的局部内部类（无法直接访问，但是可以当作返回值返回），但是局部内部类可以访问外部类的任何成员以及本方法体中的常量，即用final修饰的成员。
3. 方法体中可以访问局部内部类，但是访问语句必须在定义局部内部类之后。

当内部类和外部类当中定义了同名的成员并且要访问时：

1. 外部类的成员：外部类名.this.成员变量的名称;
2. 方法中的成员：不能引导访问，只能设置不同名称
3. 内部类的成员：this.成员的名称; this 可以省略。

6.3.1 局部内部类访问局部变量的注意事项

必须被final修饰！为什么呢?

内部类编译成功后，它会产生一个class文件，该class文件与外部类并不是同一class文件，仅仅只保留对外部类的引用。当外部类传入的参数需要被内部类调用时，从java程序的角度来看是直接被调用：

java 复制代码

```
public class OuterClass {
    public void display(final String name, String age) {
        class InnerClass {
            void display() {
                System.out.println(name);
            }
        }
    }
```

从上面代码中看好像name参数应该是被内部类直接调用？其实不然，在java编译之后实际的操作如下：

java 复制代码

```java
public class OuterClass {
    public OuterClass() {
    }
    public void display(String name, String age) {
        class InnerClass {
            InnerClass(String var2) {
                this.val$name = var2;
            }

            void display() {
                System.out.println(this.val$name);
            }
        }

    }
}
```

所以从上面代码来看，内部类并不是直接调用方法传递的参数，而是利用自身的构造器对传入的参数进行备份，自己内部方法调用的实际上时自己的属性而不是外部方法传递进来的参数。

在内部类中的属性和外部方法的参数两者从外表上看是同一个东西，但实际上却不是，所以他们两者是可以任意变化的，也就是说在内部类中我对属性的改变并不会影响到外部的形参，而然这从程序员的角度来看这是不可行的。

毕竟站在程序的角度来看这两个根本就是同一个，如果内部类改变了，而外部方法的形参却没有改变这是难以理解和不可接受的，所以为了保持参数的一致性，就规定使用final来避免形参的不改变。

简单理解就是，拷贝引用，为了避免引用值发生改变，例如被外部类的方法修改等，而导致内部类得到的值不一致，于是用final来让该引用不可改变。

故如果定义了一个匿名内部类，并且希望它使用一个其外部定义的参数，那么编译器会要求该参数引用是final的。

JDK1.8:

在JDk1.8中，默认为内部类要访问的成员加上final,因此可以不主动加final。以上，**匿名内部类**同理!

6.3.2 案例

在上面的类中，定义局部内部类

java 复制代码

```java
/**
 * 对于局部内部类,外部类无法直接访问局部内部类,但是可以返回,但是必须是objeet类型
 *
 * @return Inner1
 */
public static Object getInner3Sta() {
    //不能使用访问控制修饰符修饰 局部内部类,比如private、protected、public
    class Inner3 {
        private int inner3;

        private Inner3(int inner3) {
            this.inner3 = inner3;
        }

        /*@Override
        public String toString() {
            return "Inner3{" +
                    "inner3=" + inner3 +
                    '}';
        }*/
    }
    return new Inner3(3);
}

public static void main(String[] args) {
    Object inner3Sta = getInner3Sta();
    System.out.println(inner3Sta);
}
```

6.4 匿名内部类

定义:

> new抽象类或者是接口，能够获得一个抽象类或者是接口的一个实现类对象，这个实现类对象没有名字，所以称之为匿名内部类。本质是一个继承了类或者实现了接口的子类匿名对象。

语法:

> new 类名或者接口名() {重写方法;}

匿名内部类的使用:

1. 匿名内部类不能有构造方法。
2. 匿名内部类不能定义任何静态成员、方法和类。

5. 一个匿名内部类一定是在new的后面，用其隐含实现一个接口或实现一个类。
6. 因匿名内部类为局部内部类，所以局部内部类的所有限制都对其生效。
7. 一般情况，当这个接口的实现类或者是抽象类的实现类，只被用到一次，或者几次的时候，可以优先采用匿名内部类。同时抽象类或者接口中的方法一般不超过三个。

6.4.1 案例

```java
new Thread(new Runnable() {
    @Override
    public void run() {
        //只使用该一次线程时,可以这么定义线程和线程任务
    }
}).start();
```

7 使用内部类的好处

静态内部类的作用：

1. 只是为了降低包的深度，方便类的使用，静态内部类适用于包含类当中，但又不依赖与外在的类。
2. 由于Java规定静态内部类不能用使用外在类的非静态属性和方法，所以只是为了方便管理类结构而定义。于是我们在创建静态内部类的时候，不需要外部类对象的引用。

非静态内部类的作用：

1. 内部类继承自某个类或实现某个接口，内部类的代码操作创建其他外围类的对象。所以你可以认为内部类提供了某种进入其外围类的窗口。
2. 使用内部类最吸引人的原因是:每个内部类都能独立地继承自一个(接口的)实现，所以无论外围类是否已经继承了某个(接口的)实现，对于内部类都没有影响
3. 如果没有内部类提供的可以继承多个具体的或抽象的类的能力，一些设计与编程问题就很难解决。 从这个角度看，内部类使得多重继承的解决方案变得完整。接口解决了部分问题，而内部类有效地实现了"多重继承"。

标签： Java

文章被收录于专栏：