第3讲 | 谈谈final、finally、 finalize有什么不同？

2018-05-10 杨晓峰

第3讲 | 谈谈final、finally、 finalize有什么不同？
杨晓峰
00:00 / 11:03

Java语言有很多看起来很相似，但是用途却完全不同的语言要素，这些内容往往容易成为面试官考察你知识掌握程度的切入点。

今天，我要问你的是一个经典的Java基础题目，谈谈final、finally、 finalize有什么不同？

典型回答

final可以用来修饰类、方法、变量，分别有不同的意义，final修饰的class代表不可以继承扩展，final的变量是不可以修改的，而final的方法也是不可以重写的（override）。

finally则是Java保证重点代码一定要被执行的一种机制。我们可以使用try-finally或者try-catch-finally来进行类似关闭JDBC连接、保证unlock锁等动作。

finalize是基础类java.lang.Object的一个方法，它的设计目的是保证对象在被垃圾收集前完成特定资源的回收。finalize机制现在已经不推荐使用，并且在JDK 9开始被标记为deprecated。

考点分析

这是一个非常经典的Java基础问题，我上面的回答主要是从语法和使用实践角度出发的，其实还有很多方面可以深入探讨，面试官还可以考察你对性能、并发、对象生命周期或垃圾收集基本过程等方面的理解。

推荐使用final关键字来明确表示我们代码的语义、逻辑意图，这已经被证明在很多场景下是非常好的实践，比如：

- 我们可以将方法或者类声明为final，这样就可以明确告知别人，这些行为是不许修改的。

如果你关注过Java核心类库的定义或源码， 有没有发现java.lang包下面的很多类，相当一部分都被声明成为final class？在第三方类库的一些基础类中同样如此，这可以有效避免API使用者更改基础功能，某种程度上，这是保证平台安全的必要手段。

- 使用final修饰参数或者变量，也可以清楚地避免意外赋值导致的编程错误，甚至，有人明确推荐将所有方法参数、本地变量、成员变量声明成final。
- final变量产生了某种程度的不可变（immutable）的效果，所以，可以用于保护只读数据，尤其是在并发编程中，因为明确地不能再赋值final变量，有利于减少额外的同步开销，也可以省去一些防御性拷贝的必要。

final也许会有性能的好处，很多文章或者书籍中都介绍了可在特定场景提高性能，比如，利用final可能有助于JVM将方法进行内联，可以改善编译器进行条件编译的能力等等。坦白说，很多类似的结论都是基于假设得出的，比如现代高性能JVM（如HotSpot）判断内联未必依赖final的提示，要相信JVM还是非常智能的。类似的，final字段对性能的影响，大部分情况下，并没有考虑的必要。

从开发实践的角度，我不想过度强调这一点，这是和JVM的实现很相关的，未经验证比较难以把握。我的建议是，在日常开发中，除非有特别考虑，不然最好不要指望这种小技巧带来的所谓性能好处，程序最好是体现它的语义目的。如果你确实对这方面有兴趣，可以查阅相关资料，我就不再赘述了，不过千万别忘了验证一下。

对于finally，明确知道怎么使用就足够了。需要关闭的连接等资源，更推荐使用Java 7中添加的try-with-resources语句，因为通常Java平台能够更好地处理异常情况，编码量也要少很多，何乐而不为呢。

另外，我注意到有一些常被考到的finally问题（也比较偏门），至少需要了解一下。比如，下面代码会输出什么？

```
try {
  // do something
  System.exit(1);
} finally{
  System.out.println("Print from finally");
}
```

上面finally里面的代码可不会被执行的哦，这是一个特例。

对于finalize，我们要明确它是不推荐使用的，业界实践一再证明它不是个好的办法，在Java 9中，甚至明确将Object.finalize()标记为deprecated！如果没有特别的原因，不要实现finalize方法，也不要指望利用它来进行资源回收。

为什么呢？简单说，你无法保证finalize什么时候执行，执行的是否符合预期。使用不当会影响性能，导致程序死锁、挂起等。

通常来说，利用上面的提到的try-with-resources或者try-finally机制，是非常好的回收资源的办法。如果确实需要额外处理，可以考虑Java提供的Cleaner机制或者其他替代方法。接下来，我来介绍更多设计考虑和实践细节。

知识扩展

1.注意，final不是immutable！

我在前面介绍了final在实践中的益处，需要注意的是，final并不等同于immutable，比如下面这段代码：

```
final List<String> strList = new ArrayList<>();
strList.add("Hello");
strList.add("world");
List<String> unmodifiableStrList = List.of("hello", "world");
unmodifiableStrList.add("again");
```

final只能约束strList这个引用不可以被赋值，但是strList对象行为不被final影响，添加元素等操作是完全正常的。如果我们真的希望对象本身是不可变的，那么需要相应的类支持不可变的行为。在上面这个例子中，List.of方法创建的本身就是不可变List，最后那句add是会在运行时抛出异常的。

Immutable在很多场景是非常棒的选择，某种意义上说，Java语言目前并没有原生的不可变支持，如果要实现immutable的类，我们需要做到：

- 将class自身声明为final，这样别人就不能扩展来绕过限制了。
- 将所有成员变量定义为private和final，并且不要实现setter方法。
- 通常构造对象时，成员变量使用深度拷贝来初始化，而不是直接赋值，这是一种防御措施，因为你无法确定输入对象不被其他人修改。
- 如果确实需要实现getter方法，或者其他可能会返回内部状态的方法，使用copy-on-write原则，创建私有的copy。

这些原则是不是在并发编程实践中经常被提到？的确如此。

关于setter/getter方法，很多人喜欢直接用IDE一次全部生成，建议最好是你确定有需要时再实现。

2.finalize真的那么不堪？

前面简单介绍了finalize是一种已经被业界证明了的非常不好的实践，那么为什么会导致那些问题呢？

finalize的执行是和垃圾收集关联在一起的，一旦实现了非空的finalize方法，就会导致相应对象回收呈现数量级上的变慢，有人专门做过benchmark，大概是40~50倍的下降。

因为，finalize被设计成在对象被垃圾收集前调用，这就意味着实现了finalize方法的对象是个"特殊公民"，JVM要对它进行额外处理。finalize本质上成为了快速回收的阻碍者，可能导致你的对象经过多个垃圾收集周期才能被回收。

有人也许会问，我用System.runFinalization()告诉JVM积极一点，是不是就可以了？也许有点用，但是问题在于，这还是不可预测、不能保证的，所以本质上还是不能指望。实践中，因为finalize拖慢垃圾收集，导致大量对象堆积，也是一种典型的导致OOM的原因。

从另一个角度，我们要确保回收资源就是因为资源都是有限的，垃圾收集时间的不可预测，可能会极大加剧资源占用。这意味着对于消耗非常高频的资源，千万不要指望finalize去承担资源释放的主要职责，最多让finalize作为最后的"守门员"，况且它已经暴露了如此多的问题。这也是为什么我推荐，资源用完即显式释放，或者利用资源池来尽量重用。

finalize还会掩盖资源回收时的出错信息，我们看下面一段JDK的源代码，截取自java.lang.ref.Finalizer

```
private void runFinalizer(JavaLangAccess jla) {
//  ... 省略部分代码
try {
   Object finalizee = this.get();
   if (finalizee != null && !(finalizee instanceof java.lang.Enum)) {
      jla.invokeFinalize(finalizee);
      // Clear stack slot containing this variable, to decrease
      // the chances of false retention with a conservative GC
      finalizee = null;
   }
 } catch (Throwable x) { }
   super.clear();
}
```

结合我上期专栏介绍的异常处理实践，你认为这段代码会导致什么问题？

是的，你没有看错，这里的Throwable是被生吞了的！也就意味着一旦出现异常或者出错，你得不到任何有效信息。况且，Java在finalize阶段也没有好的方式处理任何信息，不然更加不可预测。

3.有什么机制可以替换finalize吗？

Java平台目前在逐步使用java.lang.ref.Cleaner来替换掉原有的finalize实现。Cleaner的实现利用了幻象引用（PhantomReference），这是一种常见的所谓post-mortem清理机制。我会在后面的专栏系统介绍Java的各种引用，利用幻象引用和引用队列，我们可以保证对象被彻底销毁前做一些类似资源回收的工作，比如关闭文件描述符（操作系统有限的资源），它比finalize更加轻量、更加可靠。

吸取了finalize里的教训，每个Cleaner的操作都是独立的，它有自己的运行线程，所以可以避免意外死锁等问题。

实践中，我们可以为自己的模块构建一个Cleaner，然后实现相应的清理逻辑。下面是JDK自身提供的样例程序：

```
public class CleaningExample implements AutoCloseable {
        // A cleaner, preferably one shared within a library
        private static final Cleaner cleaner = <cleaner>;
        static class State implements Runnable {
            State(...) {
                // initialize State needed for cleaning action
            }
            public void run() {
                // cleanup action accessing State, executed at most once
            }
        }
        private final State;
        private final Cleaner.Cleanable cleanable
        public CleaningExample() {
            this.state = new State(...);
            this.cleanable = cleaner.register(this, state);
        }
        public void close() {
            cleanable.clean();
        }
    }
```

注意，从可预测性的角度来判断，Cleaner或者幻象引用改善的程度仍然是有限的，如果由于种种原因导致幻象引用堆积，同样会出现问题。所以，Cleaner适合作为一种最后的保证手段，而不是完全依赖Cleaner进行资源回收，不然我们就要再做一遍finalize的噩梦了。

我也注意到很多第三方库自己直接利用幻象引用定制资源收集，比如广泛使用的MySQL JDBC driver之一的mysql-connector-j，就利用了幻象引用机制。幻象引用也可以进行类似链条式依赖关系的动作，比如，进行总量控制的场景，保证只有连接被关闭，相应资源被回收，连接池才能创建新的连接。

另外，这种代码如果稍有不慎添加了对资源的强引用关系，就会导致循环引用关系，前面提到的MySQL JDBC就在特定模式下有这种问题，导致内存泄漏。上面的示例代码中，将State定义为static，就是为了避免普通的内部类隐含着对外部对象的强引用，因为那样会使外部对象无法进入幻象可达的状态。

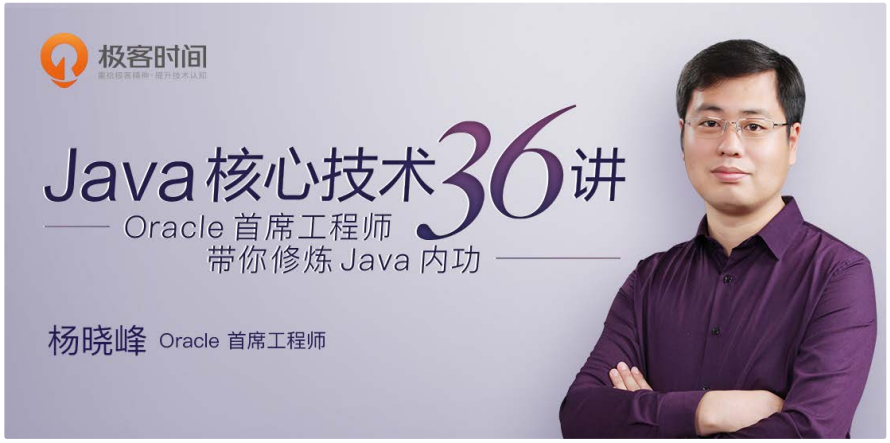
今天，我从语法角度分析了final、finally、finalize，并从安全、性能、垃圾收集等方面逐步深入，探讨了实践中的注意事项，希望对你有所帮助。

一课一练

关于今天我们讨论的题目你做到心中有数了吗？也许你已经注意到了，JDK自身使用的Cleaner机制仍然是有缺陷的，你有什么更好的建议吗？

请你在留言区写写你的建议，我会选出经过认真思考的留言，送给你一份学习鼓励金，欢迎你与我一起讨论。

你的朋友是不是也在准备面试呢？你可以“请朋友读”，把今天的题目分享给好友，或许你能帮到他。



zjh

2018-05-11

一直不懂为什么这三个经常拿来一起比较，本身就一点关系都没有啊，难道仅仅是长的像。我觉得final倒是可以和volatile一起比较下

石头狮子

2018-05-10

列几个 fianlly 不会被执行的情况:
1. try-cach 异常退出。
try{
system.exit(1)
}finally{
print(abc)
}

2. 无限循环
try{
while(ture){
print(abc)
}
}finally{
print(abc)

}

3. 线程被杀死
当执行 try，finally 的线程被杀死时。finally 也无法执行。

总结
1，不要在 finally 中使用 return 语句。
2，finally 总是执行，除非程序或者线程被中断。

WolvesLeader

2018-05-10

能不能帮我分析一哈，匿名内部累，访问局部变量时，局部变量为啥要用final来修饰吗？

作者回复

2018-05-10

这个因为Java inner class实际会copy一份，不是去直接使用局部变量，final可以防止出现数据一致性问题

★神峰★

2018-05-12

你们都看懂了吗？我怎么什么都不知道��

有渔@祭

2018-05-10

1.你说那异常被生吞，是指没写e.print...语句吧？另外我有个疑惑：super.clear()为什么写在exception里，理论上super方法写第一行，或finally里。2.在一个对象的生命周期里，其finalize方法应该只会被调用1次。3.强软弱虚引用大家都知道，这虚幻引用相比较有什么特别的吗？请再深入点。4.final是不是都在编译后确定位置？比如final List这样的，内存布局是怎样的？谢谢

皮卡皮卡丘

2018-05-10

"将 State 定义为 static，就是为了避免普通的内部类隐含着对外部对象的强引用，因为那样会使外部对象无法进入幻象可达的状态。"这个该怎么理解呢？

作者回复

2018-05-11

内部类如果不是static，它本身对外面那个类有引用关系，这一点其实从构造阶段就能看出来，你可以写段代码试试；有强引用就是strong reachable状态

小哥。

2018-05-10

copy-on-write 原则，学习了

---

sharp

2018-06-12

这三个就是卡巴斯基和巴基斯坦的关系，有个基巴关系。。。

Hesher

2018-05-10

见过一些写法是将对象手动赋值为null来让GC更快的回收，不过能起多少作用就不知道了。关于JVM中那几种引用了解不多，平时可以怎么使用呢？

---

ls

2018-05-13

Java中有说：finalize 有一种用途：在 Java 中调用非 Java 代码，在非 Java 代码中若调用了C的 malloc 来分配内存，如果不调用 C 的free 函数，会导致内存泄露。所以需要在 finalize 中调用它。

面试中会有问：为什么 String 会设计成不可变？想听听老师的解释

作者回复

2018-05-14

是的，很多资源都是需要使用本地方式获取和释放

云学

2018-06-12

请问这篇文章中涉及的知识点是java中最重要的吗？我感觉有点剑走偏锋，这种知识了解就好了，应该有很多知识比这更重要的吧，虽说面试中可能会问，但不能以面试为中心，而要把实际应用中最有用的真正核心的东西分享出来，把它讲透彻，不追求面面俱到，也不想成为语言专家，我期望通过这个专栏可以获得java中最核心最实用特性的本质认识，希望有一种醍醐灌顶的感觉，在阅读java开源框架代码时不再困惑。我有多年的c++开发背景，希望通过这个专栏对java也有提纲契领的本质认识。

refusecruder

2018-05-10

杨老师，关于final不能修改我想请教下，代码如下，class util {
public final Integer info = 123;
}

@Test
public void test() throws NoSuchFieldException, IllegalAccessException {
util util = new util();
Field field = util.getClass().getDeclaredField("info");
field.setAccessible(true);
field.set(util,789);
System.out.println(field.get(util));
System.out.println(util.info);
}
这里final修饰的被改了，如果不加accessible这句会报错，刚刚试了几个，似乎是基本数据类型改不了，封装类型都能改，请杨老师解答下我的疑惑，感谢。

作者回复

2018-05-10

setAccessible是"流氓"，不问题出在定义为基本数据类型，会被当作constant，可以反编译看看

��Children乏

2018-05-14

用final修饰的class，这可以有效避免 API 使用者更改基础功能，某种程度上，这是保证平台安全的必要手段。这个地方真的很需要个例子去帮助理解。比如大家都知道String类是被final修饰不可被继承，但假如没有被final修饰，很好奇会出现什么样不安全的后果。

作者回复

2018-05-14

谢谢反馈

公号-Java大后端

2018-05-10

1定义不可变对象类，当构造函数传入可变对象引用时、当getter函数返回可变对象引用时，容易掉坑。
2 在不可变对象类的构造函数中，如果传入值包括了可变对象，则clone先。
3 从不可变对象类的getter函数返回前，如果返回值为可变对象，则clone先。
4 Java默认的clone方法执行浅拷贝，对于数组、对象引用只是拷贝地址。浅拷贝在业务实现中可能是一个坑，需要多加注意。

5 如果步骤2、3中的浅拷贝无法满足不可变对象要求，请实现“深拷贝”。

echo _ 陈

2018-05-10

回答上面一个人的问题。
被final修饰的变量不可变。如果初始化不赋值，后续赋值，就是从null变成你的赋值，违反不可变

Do

2018-05-12

final修饰变量参数的时候，其实理解为内存地址的绑定，这样理解是不是更直观，基本类型指向栈中，引用类型指向堆中。老师后期文章能不能说下java堆栈的区别，还有变量局部变量的生命周期，最好能附上图，加深理解。

作者回复

2018-05-12

会有

小绵羊拉拉

2018-05-10

首先 这篇文章比上一讲明显感觉到由浅入深 很不错 有个问题请教一下finalize方法是用来回收对外内存是不是可以这么理解 类似于本地方法申请的能源 new出来的对象实现了这个方法当垃圾回收的时候会将对象放在fqueue等待被执行 不过是异步不知道啥时候被执行 可能被执行的时候对象已经置空导致不安全 可以这么理解吗 新的jdk引入的clear方法 能完全取代虚拟机中finalize方法吗

说重点丶

2018-05-10

常被问，string类为什么用final修饰

Colingo

2018-05-10

finally 里面的打印为什么不会被调用?

loveluckystar

2018-05-10

个人理解，finalize本身就是为了提供类似c或c++析构函数产生的，由于java中gc本身就是自动进行的，是不希望被干扰的，(就像System.gc()，并不一定起作用)所以与其费心研究如何使用这个，不如老老实实在finally中把该做的事情做了来的实惠。

作者回复

2018-05-10

对，有些特别情况需要额外处理，毕竟无法保证编程都按规范来

feifei

2018-06-30

JDK 自身使用的 Cleaner 机制仍然是有缺陷的，你有什么更好的建议吗?

1，临时对象，使用完毕后，赋值为null,可以加快对象的回收
2，公用资源对象，比如数据库连接，使用连接池
3，native调用资源的释放，比如一个进程初始化调用一次，退出调用一次，这类场景可以考虑使用cleaner
4，对尽量try-finally中完成资源的释放，即使用完毕就释放，最小化的使用，下次使用在申请。
5，可以使用钩子进行程序的正常退出清理操作。

此为我个人的一点小心得，欢迎老师指正，谢谢

jeff

2018-05-17

回答张勇的问题
注意你的final 值得作用域 生命周期

不吃老鼠的猫

2018-05-10

我有个这样一个场景，在service方法里，处理自己的逻辑，因为是调用外部接口，所以可能会报错，那么我用try catch补获异常，返回上层结果，但是问题来了，我有个log表，要保存每次请求，响应数据，我补获到外部异常后，处理了下异常，又抛出去，现在我的处理逻辑是在finally里，来保存所有请求和响应的数据，但觉得有点鸡肋，finally理论上不应该涉及到逻辑代码，求老师指正?

凡旅

2018-05-10

请问为什么被final修饰的变量需要显示赋值

梅丹隆

2018-07-11

形容词，副词和名词

团结屯儿王二狗他二大爺

2018-07-10

前半部分可以看懂，后面涉及幻象引用就不懂了。完全看懂感觉需要看一下那篇引用的文章。文章整体不错，从多个角度分析一个问题。我想这也是一个工程师应该具备的思维之一，希望自己能够不断提高，与君共勉。

待时而发

2018-06-19

有点懵

jacy

2018-05-28

加final的state为啥还能赋值呢?

大熊

2018-05-27

对我这种刚入门的来说 是有点高深了

徐金铎

2018-05-26

如果是考虑gc回收，推荐大家更关注weakreference, softreference等结合referencequeue来考虑。这样对gc更友好些。

徐金铎

2018-05-26

Final关键字，还有禁止代码前编译器重排序的作用，可以用做构造溢出的解决方案。

密码123456

2018-05-25

1。final修饰的类，不可被继承，修饰的方法不可被重写，修饰的变量不可多次赋值。通过final能够得到性能上的优化，但是不明显，如果大量使用可能会干扰代码，不能表达出本来具有的含义。故不使用。匿名内部类，访问局部变量要求传入的参数，必须是final是要保证数据一致性问题。
2。finally。代码中总是会执行的代码段。除了退出虚拟机外。
3。finalize。在虚拟机回收改对象前进行调用。此种方式不可取。因为java虚拟机不知道在什么时候才对对象进行回收。

曲高和寡

2018-05-17

关于匿名内部类访问局部变量必须final的原因，还有一个关键是匿名内部类的生命周期可能比外部类要长，而如果外部类已经被垃圾回收了，那内部类访问的就是一个空变量。final可以防止被回收，老师对么？

作者回复

2018-05-19

我理解不是

---

haoz

2018-05-16

List.of()方法我的jdk1.8中没有 网上也没有相关资料。是不是老师写错了呢?还望老师多多指教!

作者回复

2018-05-17

Java 9引入的，可能忘了介绍；
8已经4、5年了，马上9月份发布jdk 11，下一个长期支持版本，建议开始实验新的了

微笑的向日葵

2018-05-16

一般来说在框架中做资源创建和回收，都是通过aop

Z

2018-05-16

有List.of（）方法吗我怎么找不到

作者回复

2018-05-17

忘了说Java 9新增的，回头修改下

程序猿的小浣熊

2018-05-15

关于copy_on_write实现getter方法可以有例子吗？因为我理解的该原则是懒修改策略，但是不变类不应该不做任何修改么？希望可以解答一下。

作者回复

2018-05-15

用词也许有点歧义，就是只暴露copy

樱の空

2018-05-14

说到final的话，现在更多的会谈到static.许多代码都会使用到static final来同时修饰一个变量(这里称为常量更合适哈)。从而可以达到一个编译期常量的作用。这可以使得我们不需要初始化一个类就能够直接访问其成员，对节约资源效率上有不少的提升。
所以我觉得老师可以顺带提一下static,或者放些补充学习的资料哈。
而以上东西其实可以进一步深挖，这就会关系到老师在第一讲提到过的类加载，验证，链接，初始化。这个过程，介于篇幅原因未能进一步展开，有兴趣的同学可以翻看TIJ和深入理解JVM进行学习哈。
以上是个人理解，若有错误，还望指正。

作者回复

2018-05-14

个人认为static之类主要还是注重反应语义的需求，过早考虑节省资源太片面

张勇

2018-05-11

匿名内部类为什么访问外部类局部变量必须是final的？private Animator createAnimatorView(final View view, final int position) {□
MyAnimator animator = new MyAnimator();□
animator.addListener(new AnimatorListener() {□
@Override□
public void onAnimationEnd(Animator arg0) {□
Log.d(TAG, "position=" + position);
}□
});□
return animator;□
}

作者回复

2018-05-11

文中介绍了，它其实实现是会copy一份，final可以避免一致性问题

李润林

2018-05-11

finalize应该是从c++的析构函数那里继承来的一个东西，随着垃圾收集算法的不断改进，变得完全不可控了。

作者回复

2018-05-11

本身使用场景和需求是存在的，只是存在一些弊端，实在不行也得用，毕竟准确性、可靠性是基础

张勇

2018-05-11

老师咨询下，我小面这段代码String类型的参数用final修饰为啥我每次可以穿不同的参数进去，并且也可以运行成功final不是不可变么，public class MainActivity extends
AppCompatActivity {□
private static final String TAG="MainActivity";□
@Override□
protected void onCreate(Bundle savedInstanceState) {□
super.onCreate(savedInstanceState);□
setContentView(R.layout.activity_main);□
test("AAAAAAA");□
test("BBBBBBB");□
test("CCCCCCC");□
test("DDDDDDD");□
}□

public void test(final String str){□
Log.d(TAG,str);□
}

}

作者回复

2018-05-11

看不出来super内部的逻辑，再说final这东西有太多可以绕过去的方式。比如setaccessible或者配合修改modifier

冯召坤

2018-05-11

分析的不错

老胡

2018-05-11

不可变。最基本是行为不可变，不提供可变的操作。变量私有化，没有增加，修改等方法，final只是保证指针不可变，无法保证内容不可变。

老胡

2018-05-11

Cleaner机制会对jvm回收造成负担，因为gc回收的时候需要检测这个对象十分是Cleaner，然后处理。如果处理过长，十分影响gc的效率。好点方案，容器管理对象，比如spring的sopce，或者对象单利等等，gc负担是一个致命问题，所以Cleaner谨慎使用，甚至应该禁止

作者回复

2018-05-11

未必有这么可怕，比finalize有数量级的提高，spring这个不错，但不能解决所有场景

FF

2018-05-11

IDEA 的版本是2018.1.2，throw 一个方法没有声明的受检查 ex 也是合规的写法吗？即使 try 里面没有显示 throw

作者回复

2018-05-11

这是rethrow，我理解不存在模棱两可，或者哪位高手有精力去翻spec，谢谢

独嘉记忆

2018-05-10

老师，能否之后多开几讲，两三千字的限制怕知识点后面不够或者没法深入。

作者回复

2018-05-11

我也在考虑补充一些，很多地方还没展开就3000多了.....

增煌

2018-05-10

final修饰变量表示不能被修改应该是不能被赋值更合适一点吧？对基本类型来说就是不能修改，对string和object只是引用不能修改，引用的对象还是可变的。

学无止境

2018-05-10

老师，您好，我想请教一下finalize或者cleaner使用场景是什么？谢谢。

life is cool

2018-05-10

老师，final对象变量Jvm是怎么被垃圾回收器回收的呢？

一笑奈何

2018-05-10

先说答案在分析挺好��

曹铮

2018-05-10

之前甚至不知道有cleaner这回事…希望杨老师后面有机会详细说说

作者回复

2018-05-10

好的，2、3千字篇幅限制大，后面补充吧，你可以看看jdk源码用的内部cleaner

FF

2018-05-10

请教杨老师一个异常处理的问题，我们使用 eclipse 的同学通常这样处理异常：
try{
。。。。
}catch(Exception ex){
throw ex; //这里抛出异常，
}

但方法并没有声明 throws Exception，而 eclipse 通常也能编译执行，这不是违反了基本语法了吗，为何 eclipse 里面没有任何问题的？

这是什么原因？但这样的代码在使用 IntelliJ IDEA 的同学那里是完全没法编译执行的，直接就提示语法错误了

很困惑，网上没找到相关答案，望解答，感谢！

作者回复

2018-05-10

IDE有自己的编译实现，如果try里没有显式throw，只是catch那么写，应该是合规的，猜测是idea的bug，你用什么版本？

yao

2018-05-10

@mongo 另外可以起到语义上的作用

mongo

2018-05-10

final修饰引用类型的话，引用值不能被修改。但是引用值指向的内容可以被修改，这样看来修饰引用类型并不是线程安全的。什么场景下会使用final修饰引用类型呢？

作者回复

2018-05-10

引用不希望被修改的时候，仍然有一定保护作用