final、finally、finalize的区别

版本号: 2018/8/28-1(16:57)

面试题

1、final、finally、finalize有什么不同?

> 1. final: 可以修饰class、方法、变量
>   1. final修饰的class无法被继承扩展
>   2. final修饰的方法，无法被重写(overide)
>   3. final修饰的变量，无法被更改
> 2. finally: 用于try-catch，保证发生异常时会进行清理工作，如释放资源、执行unlock等操作。
> 3. finalize： `java.lang.Object` 的方法，本意是在GC前做一些回收工作，但是会造成性能急剧下降。
>   1. JDK9中已经被废弃
>   2. 执行时间不确定
>   3. 严重影响GC性能(40~50倍)

final(7)

1、final的作用

> 1. 可以修饰class、方法、变量

> 2. final修饰的class无法被继承扩展
> 3. final修饰的方法，无法被重写(overide)
> 4. final修饰的变量，无法被更改

2、final的好处

> 1. 保护代码，避免以外的编程错误。
> 2. 防止别人去修改核心内容，导致功能不正确以及避免潜在的安全隐患

3、final能有助于JVM将方法内联，改善编译器进行条件编译的能力，从而提高性能?

> 错误！
>
> 1. 这种结论是基于 猜测和假设
> 2. 现代高性能的虚拟机，如 `HotSpot` 非常智能，并不是通过 `final` 来判断是否能内联。

4、final修饰的变量就是常量?

> 错误！
> 1-修饰的基本数据类型，一旦赋值就无法改变。该变量必须要先初始化。

```
final int a = 10;
```

> 2-修饰的引用类型，可以间接修改。包括List等。

```
final StringBuilder a = new StringBuilder("aaaa");

StringBuilder b = a;
b.append(" 间接修改了final的引用a所指向的内容");

System.out.println(a+""); // 结果为： aaaa 间接修改了final的引用所指向的内容
```

5、如何创建一个immutable(不可变)的List?

> final不具备immutable的特性：

```
final List<String> list = new ArrayList<>();
list.add("Hello");
list.add("final 阻止不了我");
```

> 利用Connections生成不可修改的集合：

```
List<String> immutable = Collections.unmodifiableList(list);
immutable.add("添加内容会报错");
```

实现immutable Class

6、如何实现immutable(不可改变)的类?

> 1. final修饰class：避免通过扩展来绕开限制。
> 2. private和final修饰所有成员变量，并且不要实现setter方法。
> 3. 使用 深拷贝 来构造对象：不采用直接复制，避免被间接修改内容。
> 4. getter等方法采用 `copy-on-write` 原则：将内容copy一份提供出去。

实际问题

7、匿名内部类中访问的局部变量为什么要用final?

> 1. Java内部类会去copy一份变量，而不是采用局部变量。
> 2. final能保证数据一致。

finally(7)

1、finally的作用

> 1. 用于try-catch，保证发生异常时会进行清理工作，如释放资源、执行unlock等操作。
> 2. 推荐使用Java 7的 `try-with-resources` 来替代 `finally`
> 3. 能有效避免finally丢失异常的情况。
> 4. try-with-resources，采用 语法糖 来实现，编码量少且规范。

2、不要在finally处理会返回的数值、对象。

3、在try中调用break、continue、return都会进入finally

4、不要在try中去return返回值，否则一定要保证finally不会修改返回值(数值、对象)。

5、try中有return，也一定会去执行finally。

exit

6、finally和return、System.exit(1)的关系

> 1. finally前执行return，还是会执行 `finally` 的内容。
> 2. finally前执行 `System.exit(1)` ，会立即退出，不会执行 `finally` ，然而这种场景并没有什么用。

7、finally在哪些情况中不会执行?

> 1. System.exit(-1)：异常退出
> 2. try-catch中无限循环
> 3. finally所处于的线程被杀死

finalize(6)

1、finalize的作用

> 1. 本意是在GC前进行资源回事。
> 2. 实际上 `Object.finalize()` 在Java 9中已经被废弃。
> 3. 缺点非常明显：
>   1. 无法保证finalize什么时候执行，是否满足预期。
>   2. 会影响性能，且容易导致死锁等各种问题。
> 4. 除了try-catch-finally和resources两种方法，还可以利用Java 9的cleaner机制。

致命缺点

2、finalize的性能极差

> 1. 在专门的 `benchmark` 上测试，finalize在GC时间上，大约有 `40~50倍` 的性能差距。极差！
> 2. 即使使用 `System.runFialization()` 也没有多大用途，finalize拖慢GC回收，甚至可能会导致 `OOM`

3、finalize会掩盖资源回收时的异常信息

> 1. 根据JDK的源码: java.lang.ref.Finalizer
> 2. 资源回收时出现的异常，会通过 `Throwable` 捕获，并且不作任何处理！

```
try{
    // 资源回收工作
}catch (Throwable x){
    // 居然吞掉了Throwable
    super.clear();
}
```

替代品：Cleaner机制

4、有什么机制可以替换掉finalize?

> 1. Java 平台正逐渐使用 `java.lang.ref.Cleaner` 来替换掉原有的 `finalize` 实现。
> 2. Cleaner的实现利用了 幻想引用(`PhantomReference`)
> 3. 采用一种常见的 `post-mortem` 清理机制
> 4. Cleaner利用幻想引用和引用队列，保证了对象销毁前的清理工作，比如关闭文件描述符等。
> 5. 比finalize更加轻量、更加可靠

5、Cleaner的优点

> 1. 每个Cleaner的操作都是独立的，有自己的运行线程，可以避免意外死锁等问题。

6、Cleaner依旧有缺陷

> 1. 如果由于一些原因导致 幻想引用 堆积，同样会出现问题。
> 2. 只适合作为最后的手段。

知识储备

1、Java 9 Cleaner机制

> 参考资料：https://docs.oracle.com/javase/9/docs/api/java/lang/ref/Cleaner.html