异常相关面试题

1. Java 中的 Exception 是什么?

这个问题经常在第一次问有关异常的时候或者是面试菜鸟的时候问。我从来没见过面高级或者资深工程师的时候有人问这玩意，但是对于菜鸟，是很愿意问这个的。简单来说，异常是 Java 传达给你的系统和程序错误的方式。在 java 中，异常功能是通过实现比如 Throwable，Exception，RuntimeException 之类的类，然后还有一些处理异常时候的关键字，比如 throw，throws，try，catch，finally 之类的。 所有的异常都是通过 Throwable 衍生出来的。Throwable 把错误进一步划分为 java.lang.Exception 和 java.lang.Error. java.lang.Error 用来处理系统错误，例如 java.lang.StackOverFlowError 或者 Java.lang.OutOfMemoryError 之类的。然后 Exception 用来处理程序错误，请求的资源不可用等等。

2. Java 中的检查型异常和非检查型异常有什么区别？

这又是一个非常流行的 Java 异常面试题，会出现在各种层次的 Java 面试中。检查型异常和非检查型异常的主要区别在于其处理方式。检查型异常需要使用 try, catch 和 finally 关键字在编译期进行处理，否则编译器会报错。对于非检查型异常则不需要这样做。Java 中所有继承自 java.lang.Exception 类的异常都是检查型异常，所有继承自 RuntimeException 的异常都被称为非检查型异常。你也可以查看下一篇文章来了解 更多关于检查型异常和非检查型异常之间的区别。

3. Java 中的 NullPointerException 和 ArrayIndexOutOfBoundException 之间有什么相同之处？

在 Java 异常面试中这并不是一个很流行的问题，但会出现在不同层次的初学者面试中，用来测试应聘者对检查型异常和非检查型异常的概念是否熟悉。顺便说一下，该题的答案是，这两个异常都是非检查型异常，都继承自 RuntimeException。该问题可能会引出另一个问题，即 Java 和 C 的数组有什么不同之处，因为 C 里面的数组是没有大小限制的，绝对不会抛出 ArrayIndexOutOfBoundException。

4. 在 Java 异常处理的过程中，你遵循的那些最好的实践是什么？

这个问题在面试技术经理是非常常见的一个问题。因为异常处理在项目设计中是非常关键的，所以精通异常处理是十分必要的。异常处理有很多最佳实践，下面列举集中，它们提高你代码的健壮性和灵活性：

1) 调用方法的时候返回布尔值来代替返回 null，这样可以 NullPointerException。由于空指针是 java 异常里最恶心的异常，你可以参考一下下面的技术文章 coding best practices to minimize NullPointerException。去看看里面具体的例子。

2) catch 块里别不写代码。空 catch 块是异常处理里的错误事件，因为它只是捕获了异常，却没有任何处理或者提示。通常你起码要打印出异常信息，当然你最好根据需求对异常信息进行处理。

3)能抛受控异常（checked Exception）就尽量不抛受非控异常(checked Exception)。通过去掉重复的异常处理代码，可以提高代码的可读性。

4) 绝对不要让你的数据库相关异常显示到客户端。由于绝大多数数据库和 SQLException 异常都是受控异常，在 Java 中，你应该在 DAO 层把异常信息处理，然后返回处理过的能让用户看懂并根据异常提示信息改正操作的异常信息。

5) 在 Java 中，一定要在数据库连接，数据库查询，流处理后，在 finally 块中调用 close()方法。我已经在我的文章 Top 10 Java exception handling best practices 中分享了关于这方面的很多知识，你们也可以看看这篇文章。

5. 既然我们可以用 RuntimeException 来处理错误，那么你认为为什么 Java 中还存在检查型异常?

这是一个有争议的问题，在回答该问题时你应当小心。虽然他们肯定愿意听到你的观点，但其实他们最感兴趣的还是有说服力的理由。我认为其中一个理由是，存在检查型异常是一个设计上的决定，受到了诸如 C++等比 Java 更早的编程语言设计经验的影响。绝大多数检查型异常位于 java.io 包内，这是合乎情理的，因为在你请求了不存在的系统资源的时候，一段强壮的程序必须能够优雅的处理这种情况。通过把 IOException 声明为检查型异常，Java 确保了你能够优雅的对异常进行处理。另一个可能的理由是，可以使用 catch 或 finally 来确保数量受限的系统资源（比如文件描述符）在你使用后尽早得到释放。 Joshua Bloch 编写的 Effective Java 一书 中多处涉及到了该话题，值得一读。

6. throw 和 throws 这两个关键字在 java 中有什么不同?

一个 java 初学者应该掌握的面试问题。 throw 和 throws 乍看起来是很相似的尤其是在你还是一个 java 初学者的时候。尽管他们看起来相似，都是在处理异常时候使用到的。但在代码里的使用方法和用到的地方是不同的。throws 总是出现在一个函数头中，用来标明该成员函数可能抛出的各种异常，你也可以申明未检查的异常，但这不是编译器强制的。如果方法抛出了异常那么调用这个方法的时候就需要将这个异常处理。另一个关键字 throw 是用来抛出任意异常的，按照语法你可以抛出任意 Throwable (i.e. Throwable 或任何 Throwable 的衍生类)，throw 可以中断程序运行，因此可以用来代替 return．最常见的例子是用 throw 在一个空方法中需要 return 的地方抛出 UnSupportedOperationException 代码如下：

```
1       private static voidshow()
{

2           throw new UnsupportedOperationException("Not
yet implemented");

3       }
```

可以看下这篇 文章查看这两个关键字在 java 中更多的差异 。

7. 什么是“异常链”？

“异常链”是 Java 中非常流行的异常处理概念，是指在进行一个异常处理时抛出了另外一个异常，由此产生了一个异常链条。该技术大多用于将“受检查异常”（checked exception）封装成为“非受检查异常”（unchecked exception)或者 RuntimeException。顺便说一下，如果因为因为异常你决定抛出一个新的异常，你一定要包含原有的异常，这样，处理程序才可以通过 getCause()和 initCause()方法来访问异常最终的根源。

8. 你曾经自定义实现过异常吗？怎么写的?

很显然，我们绝大多数都写过自定义或者业务异常，像 AccountNotFoundException。在面试过程中询问这个 Java 异常问题的主要原因是去发现你如何使用这个特性的。这可以更准确和精致的去处理异常，当然这也跟你选择 checked 还是 unchecked exception 息息相关。通过为每一个特定的情况创建一个特定的异常，你就为调用者更好的处理异常提供了更好的选择。相比通用异常（general exception)，我更倾向更为精确的异常。大量的创

建自定义异常会增加项目 class 的个数，因此，在自定义异常和通用异常之间维持一个平衡是成功的关键。

9. JDK7 中对异常处理做了什么改变？

这是最近新出的 Java 异常处理的面试题。JDK7 中对错误(Error)和异常(Exception)处理主要新增加了 2 个特性，一是在一个 catch 块中可以出来多个异常，就像原来用多个 catch 块一样。另一个是自动化资源管理(ARM), 也称为 try-with-resource 块。这 2 个特性都可以在处理异常时减少代码量，同时提高代码的可读性。对于这些特性了解，不仅帮助开发者写出更好的异常处理的代码，也让你在面试中显的更突出。我推荐大家读一下 Java 7 攻略，这样可以更深入的了解这 2 个非常有用的特性。

10. 你遇到过 OutOfMemoryError 错误嘛？你是怎么搞定的？

这个面试题会在面试高级程序员的时候用，面试官想知道你是怎么处理这个危险的 OutOfMemoryError 错误的。必须承认的是，不管你做什么项目，你都会碰到这个问题。所以你要是说没遇到过，面试官肯定不会买账。要是你对这个问题不熟悉，甚至就是没碰到过，而你又有 3、4 年的 Java 经验了，那么准备好处理这个问题吧。在回答这个问题的同时，你也可以借机向面试秀一下你处理内存泄露、调优和调试方面的牛逼技能。我发现掌握这些技术的人都能给面试官留下深刻的印象。你们也可以到 how to fix java.lang.OutOfMemoryError 去看看我写的另一篇关于这个问题更详细细节的文章

11. 如果执行 finally 代码块之前方法返回了结果，或者 JVM 退出了，finally 块中的代码还会执行吗？

这个问题也可以换个方式问： “如果在 try 或者 finally 的代码块中调用了 System.exit()，结果会是怎样” 。了解 finally 块是怎么执行的，即使是 try 里面已经使用了 return 返回结果的情况，对了解 Java 的异常处理都非常有价值。只有在 try 里面是有 System.exit(0)来退出 JVM 的情况下 finally 块中的代码才不会执行。

12. Java 中 final,finalize,finally 关键字的区别

这是一个经典的 Java 面试题了。我的一个朋友为 Morgan Stanley 招电信方面的核心 Java 开发人员的时候就问过这个问题。final 和 finally 是 Java 的关键字，而 finalize 则是方法。final 关键字在创建不可变的类的时候非常有用，只是声明这个类是 final 的。而 finalize()方法则是垃圾回收器在回收一个对象前调用，但也 Java 规范里面没有保证这个方

法一定会被调用。finally 关键字是唯一一个和这篇文章讨论到的异常处理相关的关键字。在你的产品代码中，在关闭连接和资源文件的是时候都必须要用到 finally 块。更多看 here

13. 下面的代码都有哪些错误：

```
01  public static void start() throws IOException,
    RuntimeException{
02
03     throw new RuntimeException("Not
    able to Start");
04  }
05
06  public static void main(String
    args[]) {
07     try {
08           start();
09     } catch (Exception
    ex) {
10              ex.printStackTrace();
11     } catch (RuntimeException
    re) {
12              re.printStackTrace();
13     }
14  }
```

这段代码会在捕捉异常代码块的 RuntimeException 类型变量“re”里抛出编译异常错误。因为 Exception 是 RuntimeException 的超类，在 start 方法中所有的 RuntimeException 会被第一个捕捉异常块捕捉，这样就无法到达第二个捕捉块，这就是抛出“exception java.lang.RuntimeException has already been caught”的编译错误原因。

14. 下面的 Java 代码都有哪些错误：

```
01  public classSuperClass
    {
02      public void start() throws IOException{
03          throw new IOException("Not
    able to open file");
04      }
05  }
06
07  public class SubClass extendsSuperClass{
08      public void start() throws Exception{
09          throw new Exception("Not
    able to start");
10      }
11  }
```

这段代码编译器将对子类覆盖 start 方法产生不满。因为每个 Java 中方法的覆盖是有规则的，一个覆盖的方法不能抛出的异常比原方法继承关系高。因为这里的 start 方法在超

类中抛出了 IOException，所有在子类中的 start 方法只能抛出要么是 IOExcepition 或是其子类，但不能是其超类，如 Exception。

15. 下面的 Java 异常代码有什么错误：

```
01  public static void start(){
02     System.out.println("Java
    Exception interivew
    question Answers for
    Programmers");
03  }
04
05  public static void main(String
    args[]) {
06     try{
07        start();
08     }catch(IOException
    ioe){
09        ioe.printStackTrace();
10     }
11  }
```

上面的 Java 异常例子代码中，编译器将在处理 IOException 时报错，因为 IOException 是受检查异常，而 start 方法并没有抛出 IOException，所以编译器将抛出“异常， java.io.IOException 不会在 try 语句体中抛出”，但是如果你将 IOException 改为 Exception，编译器报错将消失，因为 Exception 可以用来捕捉所有运行时异常，这样就不需要声明抛出语句。我喜欢这样带有迷惑性的 Java 异常面试题，因为它不会让人轻易的找出是 IOException 还是 Exception。你也可以在 Joshua Bloach 和 Neil Gafter 的 Java 谜题中找到一些有关 Java 错误和异常的具有迷惑性问题。

我也经常看到在一些新人和有经验的 Java 面试者遇到有关 Java 错误和异常的面试题。当然还有很多有关异常的问题我没有涉及到，如果你有一些好的问题，请告诉我，我将尽力在这个 java 异常问答系列中收录这些问题。还有最后一个我留给伙伴们的问题是“为什么 Java 异常被认为是比返回错误代码要好”，告诉我你对于这里的 Java 异常面试问答系列有什么想法。